POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

A Journey to Improve Neural Architecture Search: Advancements in Neural Architecture Transfer and Once-For-All

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Simone Sarti**

Student ID: 968786
Advisor: Prof. Matteo Matteucci
Co-advisors: Eugenio Lomurno
Academic Year: 2021-2022

Abstract

Today, artificial neural networks are the state-of-the-art machine learning models for solving a large variety of complex tasks, especially image classification. Research into Neural Architecture Search (NAS) techniques, which automate the design of deep neural networks, has also grown rapidly in recent years. Among the latest NAS techniques are Once-For-All (OFA) and Neural Architecture Transfer (NAT). Although they take different approaches, their main goal is the same: to avoid having to perform a complete and expensive NAS process every time it becomes necessary to find a new specialised network. With the aim of improving NAT, which is itself partially based on OFA, this thesis is developed along three works.

Firstly, a new technique for training and optimising early-exit neural networks is proposed. Called *Anticipate, Ensemble and Prune* (AEP), it works by creating a weighted ensemble of the network's multiple exits. Extensive testing has shown that the AEP technique can provide accuracy improvements of up to 15%, in terms of average percentage change over to results obtained by the corresponding single-exit networks trained under the same initial conditions, while reducing the number of parameters by up to 41%, the number of operations by up to 18%, and latency by up to 16.3%.

Secondly, OFAv2 is presented, an extension of OFA that aims to improve performance while maintaining its computationally advantageous approach to NAS. The OFAv2 networks and algorithms support the presence of new architectural designs and components such as early exits, parallel blocks and dense skip connections. To properly train the modified networks, the OFA training algorithm has been extended with two new steps called Elastic Level and Elastic Exit. Furthermore, a new AEP-based ensemble knowledge distillation technique is presented, and an improved teacher network selection strategy is proposed. With these modifications, the accuracy of OFAv2 networks on the Tiny ImageNet dataset increased by up to 12% compared to the original version of OFA.

Finally, NATv2 extends NAT to allow it to use of any of the OFAv2 supernets as starting points. A revised methodology for sampling subnets and managing the archive of best architectures has been implemented, complemented by a new pre-processing step. Two

alternative post-processing steps have also been introduced, one of which is based on the AEP technique. Compared to the best subnets found by NAT using the OFA supernet, those resulting from the application of NATv2 to the OFAv2 supernets achieve in far superior performance. In addition to a significant reduction in the number of parameters, operations and latency, accuracy increased by 2.3% on CIFAR-10, by 5% on CIFAR-100 and by 12% on Tiny ImageNet, among the other datasets.

Keywords: Neural Architecture Transfer, Once-For-All, AEP, OFAv2, NATv2, Image Classification, CNN

Abstract in lingua italiana

Al giorno d'oggi le reti neurali artificiali sono, tra i modelli di machine learning, lo stato dell'arte per la risoluzione di una grande varietà di problemi complessi, tra cui quello della classificazione di immagini. La ricerca nell'ambito delle tecniche di Neural Architecture Search (NAS), che automatizzano la progettazione di reti neurali, è cresciuta rapidamente negli ultimi anni, e tra le più recenti tecniche di NAS vi sono Once-For-All (OFA) e Neural Architecture Transfer (NAT). Sebbene adottino approcci diversi, entrambe mirano a raggiungere un obiettivo: evitare di dover eseguire un completo, e costoso processo di NAS, ogni qual volta si renda necessario trovare una nuova rete specializzata. Questa tesi che ha l'obiettivo di migliorare NAT, che è parzialmente basato su OFA, si sviluppa su tre lavori.

Per prima cosa viene proposta una nuova tecnica per l'addestramento e l'ottimizzazione di reti neurali con uscite anticipate. Questa tecnica, denominata *Anticipate, Ensemble and Prune* (AEP), funziona creando un ensemble pesato delle uscite presenti nella rete. Una lunga serie di test ha dimostrato che utilizzando AEP si possono ottenere miglioramenti in accuratezza fino al 15%, in termini di variazione percentuale media rispetto ai risultati ottenuti dalle corrispondenti reti a uscita singola addestrate a partire dalle stesse condizioni iniziali, riducendo al contempo il numero di parametri fino al 41%, di operazioni fino al 18% e di latenza fino al 16,3%.

Successivamente viene presentato OFAv2, un'estensione di OFA che mira a migliorarne le prestazioni mantenendone l'approccio computazionalmente vantaggioso all'esecuzione di NAS. Le reti e gli algoritmi di OFAv2 supportano la presenza di nuovi design e componenti architetturali, tra cui le uscite anticipate, blocchi paralleli e connessioni residuali dense. Per addestrare correttamente le superreti modificate, l'algoritmo di OFA è stato arricchito con due nuovi step chiamati Elastic Level e Elastic Exit. Vengono inoltre presentate ed utilizzate sia una nuova tecnica di knowledge distillaton basata su AEP, che una migliore strategia per migliorare la selezione della rete insegnante. Con queste modifiche, l'accuratezza delle reti di OFAv2 sul dataset Tiny ImageNet è aumentata fino al 12% rispetto a quella raggiunta utilizzando la versione originale di OFA.

Infine viene proposto NATv2, un'estensione di NAT atta a permettere l'utilizzo di una qualsiasi delle nuove superreti di OFAv2 come punto di partenza per la ricerca. In NATv2 è stata rivista sia la metodologia utilizzata per il campionamento delle sottoreti, complementata da un nuovo step di pre-elaborazione, che la modalità utilizzata per la gestione dell'archivio contenente le migliori architetture. Sono state anche introdotte due nuovi step alternativi di post-elaborazione, uno dei quali basato sulla tecnica AEP. Rispetto alle migliori sottoreti trovate utilizzando NAT sulla superrete standard di OFA, quelle risultanti dall'applicazione di NATv2 alle superreti di OFAv2 possono raggiungere prestazioni nettamente superiori. Nello specifio, oltre ad una significativa riduzione del numero di parametri, di operazioni e di latenza, l'accuratezza è aumentata fino al 2.3% su CIFAR-10, al 5% su CIFAR-100 e al 12% su Tiny ImageNet.

Parole chiave: Neural Architecture Transfer, Once-For-All, AEP, OFAv2, NATv2, Classificazione di Immagini, CNN

Contents

1 | Introduction

Over the past few years, deep learning has emerged as one of the most interesting and prolific fields of machine learning and, more generally, of computer science. Deep learning models have the outstanding ability to learn autonomously to identify significant features directly from raw input, eliminating the necessity for human feature engineering. This property has made them highly suitable for solving tasks where humans struggle to describe relevant features mathematically, such as Computer Vision (CV) and Natural Language Processing (NLP). Despite the significant amounts of data and computational resources required to train them, the remarkable performance of deep learning models has led to their widespread adoption, fundamentally transforming the way many tasks are approached and performed in today's world.

Convolutional Neural Networks (CNNs) have become the most widely used deep learning models for solving computer vision tasks, especially image classification. CNNs differ from standard neural networks in that they use convolutional layers as their primary building blocks. These layers extract the relevant features from input images by applying filters through convolution operations. The set of intermediate representations produced by CNNs are known as feature maps, and they encapsulate the most informative features extracted from the input images. CNNs include not only convolutional layers, but also pooling layers, which are used to downsample feature maps; dropout layers, which play an important role in counteracting overfitting; and batch normalization layers, the presence of which improves training stability. Activation functions are also present in CNNs, the introduction of non-linearities helps the network to learn more sophisticated representations of the input data.

The public release of benchmark datasets, such as ImageNet [1], has enabled researchers to train deep neural networks on millions of labelled images. This in turn played a key role in the development of AlexNet [2], the first high-performance CNN for image classification. Since then, a series of increasingly rapid discoveries has led to the production of models that set ever higher standards of accuracy, efficiency and scalability. The major milestone along this path have been VGG [3], InceptionNet[4], ResNet [5], DenseNet [6],

MobileNet [7], EfficientNet [8], and most recently, ConvNeXt [9]. These architectures are characterised by a wide variety of designs and components, most notably the use of skip connections, dense blocks and early exits.

Despite the excellent results achieved by hand-crafted CNN models, human assumptions about the design of neural architectures can become a limiting factor to innovation, preventing the discovery of new and more efficient patterns. Furthermore, manually tuning the structure of networks to maximise their performance for a specific application is extremely time-consuming. For these reasons, a family of techniques known as Neural Architecture Search (NAS) has been developed. NAS algorithms are capable of automatically designing and/or finding neural networks that optimally solve a given task while satisfying a set of constraints, minimising or eliminating the need for human intervention. Although capable of producing extremely powerful networks, traditional NAS algorithms suffer from significant drawbacks: they require very large amounts of computational resources and time to run, and often produce models that are so task-specific that they are difficult to reuse, thus requiring the algorithm to be run many times to find an optimal network for each condition.

Two NAS techniques have recently been proposed to overcome these limitations. They are called Neural Architecture Transfer (NAT) [10] and Once-For-All (OFA) [11], and will serve as the basis for this thesis.

OFA aims to design high-performance neural networks that can be easily adapted to different hardware configurations, while minimising power consumption and eliminating the need for retraining. This is achieved by using a single large model with a dynamic architecture called a *supernet*. Such a model is trained using a special algorithm called Progressive Shrinking (PS), which allows the fine-tuning of progressively smaller *subnets* within the supernet. Once the supernet has been trained, search steps can be performed to directly find subnets that best suit specific platforms and resource constraints.

NAT aims to produce neural architectures that perform well under a variety of objectives by transferring and adapting knowledge from pretrained supernet models. It alternates transfer learning steps with many-objective evolutionary search steps, adapting only those parts of the supernet that correspond to subnets found on the trade-off front by the search algorithm. At the end of the process, both the best subnets and the adapted supernet are returned.

Using NAT as a starting point, the original goal of this thesis was to extend its supernet with new architectural elements such as early exits, parallel blocks and skip connections, and to analyse the impact of these modifications. Such changes to the supernet design

would allow for more diverse set of sub-architectures to be explored and evaluated after an appropriate redesign of the original NAT algorithm, potentially leading to better performing final subnets. However, as the NAT supernet and its adaptation algorithm are derived from those of OFA, a full understanding of OFA and a revision of its networks and algorithms was required before any changes could be made to NAT. The most significant architectural improvement planned was the integration of early exits into the networks, as this would provide much greater flexibility in the selection of subnets.

Thus, a study of the behaviour of CNNs enriched with early exits is the first of the topics addressed in this thesis. In particular, a new technique is proposed for training and optimising early exit networks via a weighted ensemble of their exits; it is called Anticipate, Ensemble and Prune (AEP). This technique has been rigorously tested on many of the best-known neural architectures and datasets for image classification to understand which factors contribute to the greatest improvements.

OFAv2 is then presented. This is the extended version of OFA that is capable of training supernets supporting any combination of the three architectural modifications mentioned above. To make this possible, the Extended Progressive Shrinking (EPS) algorithm was created by introducing two new elastic steps in PS, so that progressively smaller subnets can be sampled and fine-tuned even when early exits or parallel blocks are present within the supernet. Furthermore, a new form of knowledge distillation for early exit networks is proposed, and an improved method for obtaining the teacher network is adopted. The algorithms used in OFAv2 to train and manage early exit networks are based on the AEP technique. All OFAv2 supernets were evaluated and compared on the Tiny ImageNet dataset [12].

Finally, by building upon OFAv2, the NATv2 algorithm was developed. It extends NAT by allowing its many-objective search and adaptation algorithms to run on any of the enriched dynamic supernets generated by OFAv2 and trained with the Extended Progressive Shrinking algorithm. In addition, both a pre-processing step and post-processing step have been introduced into the algorithm pipeline. A revised methodology for sampling subnets and managing the archive of best architectures was also adopted, along with a number of time-saving measures. NATv2 networks were tested on a variety of datasets, and the impact of different architectures on the performance of the optimal subnets found was evaluated and compared. The combination of the extended supernets and an updated algorithm meant that NATv2 was ultimately able to produce optimal subnets that significantly outperformed those found by NAT within the baseline supernet, both in terms of accuracy and additional objectives.

This thesis is divided into a further six chapters: Chapter 2 introduces all the fundamental elements and concepts needed to understand the techniques presented in the following chapters. Chapter 3 focuses on the analysis of state-of-the-art work and is divided into three sections: first, Section 3.1 examines the main works in the literature on the introduction and exploitation of early exits in neural networks, then Section 3.2 presents the Once-For-All technique, and finally Section 3.3 describes the Neural Architecture Transfer algorithm. The newly developed techniques are then presented, thoroughly studied and evaluated. In particular, Chapter 4 presents AEP, Chapter 5 presents OFAv2, and Chapter 6 presents NATv2. Finally, Chapter 7 concludes the thesis by summarising the main contributions and suggesting possible future research directions.

The research conducted for this thesis led to the publication of two papers, namely *"Anticipate, Ensemble and Prune: Improving Convolutional Neural Networks via Aggregated Early Exits"* [13] for AEP, and *"Enhancing Once-For-All: A Study on Parallel Blocks, Skip Connections and Early Exits"* [14] for OFAv2.

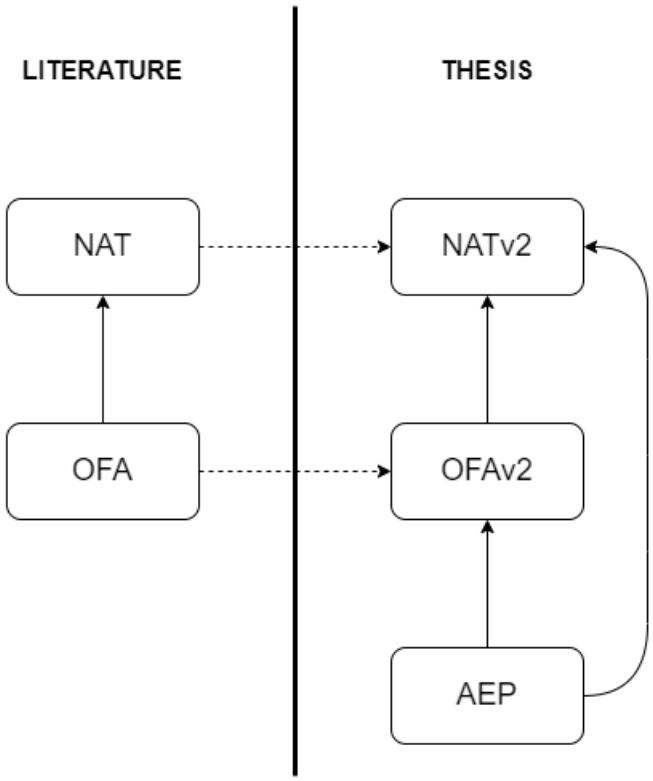


Figure 1.1: Thesis development outline

2 | Background

This chapter serves as an introduction to a number of important machine learning techniques and to the neural architectures and components most relevant to this thesis, knowledge of which is a prerequisite for understanding the concepts and techniques presented in the following chapters. These concepts may be related to the literature on which the thesis is based, or they may be starting points for the new techniques developed in this thesis. Each section focuses on the description of a single concept or technique. The order of presentation is such that if concept B is based, even partially, on concept A, then concept A is presented first. If two concepts are logically independent, the order in which they are presented does not reflect any order of importance.

2.1. Genetic Algorithms

Genetic algorithms (GA) are optimisation algorithms used to solve complex constrained and unconstrained problems by emulating the process of natural selection. They work by progressively altering, through recombination operations, a population of potential solutions (or "chromosomes") represented by arrays of variables (or "genes") of fixed length. The value associated with a gene is called an "allele". At each iteration (or "generation"), the candidate solutions are evaluated using a "fitness function", which numerically assesses the goodness of each solution. These fitness values are then used by a selection process to extract a number of solutions from the population, which are made to reproduce by applying a crossover operation. A child solution is created by inheriting the genes of its two parents: the child inherits the genes of the first parent until a crossover point is found, after which it inherits the genes of the second parent, and so on for each crossover point. After the child solution has been constructed, there is a small chance that one of its genes will undergo a "mutation" resulting in a slightly different value; this improves the diversity of solutions and allows better exploration of the search space. The process continues until a satisfactory solution is found or the algorithm runs out of time/iterations.

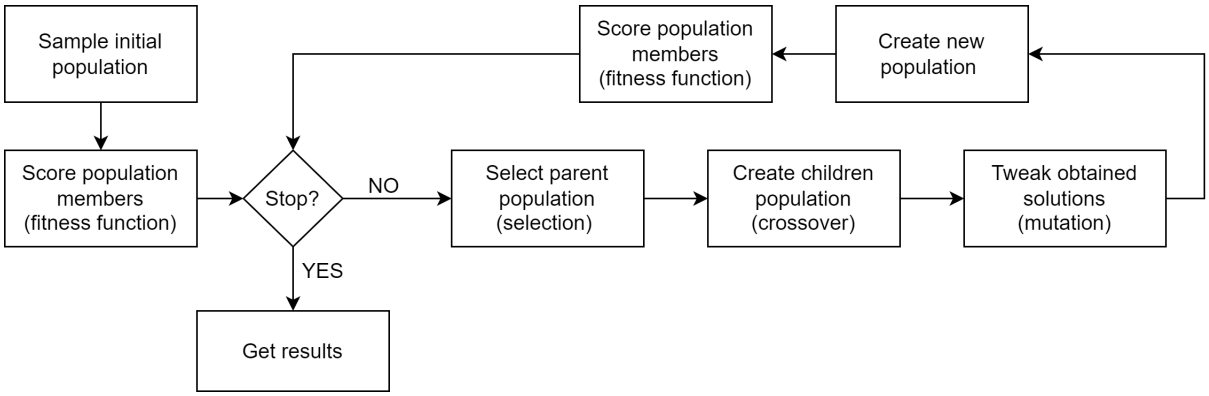


Figure 2.1: Outline of a genetic algorithm

2.1.1. NSGA-II

NSGA-II (Non-dominated Sorting Genetic Algorithm v2) [15] is a Multi-Objective Evolutionary Algorithm (MOEA). Before explaining the procedure used by NSGA-II, it is important to clarify the concept of "domination": in the context of multi-objective optimisation problems, solution S_1 is said to dominate solution S_2 if, for all objectives, S_1 is at least as good as S_2, and there exists at least one objective for which S_1 is strictly better than S_2. This means that solutions can be partitioned into domination ranks, and that solutions assigned to a given rank dominate all solutions assigned to higher ranks, while not being dominated by the others assigned to the same rank.

NSGA-II works as follows: for the first iteration, the members of the parent population are randomly selected, then the *Fast-non-dominated-sort* algorithm (Algorithm 2.1) is applied to sort the members of the population into non-domination fronts. According to the division into fronts, a fitness value is assigned to each solution. A lower rank corresponding to a lower fitness value, which must be minimised. The operations of selection, recombination and mutation are then applied to produce N offspring from N selected parents, and the two populations are merged, ensuring elitism. From this point on, at each iteration, the Fast-non-dominated-sort algorithm is applied to assign each solution to a front. The fronts are then taken in order of increasing rank until the total number of solutions is $\geq N$. If the value of N is exceeded when the last front considered is taken, only some of the solutions from that front are selected, in particular those that maximise diversity. The value of the diversity metric is based on the calculation of the *Crowding Distance* (Algorithm 2.2), which is calculated as the sum for all objectives of the normalised distance between two neighbouring points found around the point being evaluated. The solutions with the highest crowding distance (the most isolated ones) are selected.

Algorithm 2.1 Fast-non-dominated-sort

Input: population P
for all $p \in P$ **do**
  $S_p = \emptyset$
  $n_p = 0$
  for all $q \in P$ **do**
    if $(p \prec q)$ {if p dominates q} **then**
      $S_p = S_p \cup \{q\}$ {add q to the set of solutions dominated by p}
    else
      if $(q \prec p)$ **then**
        $n_p = n_p + 1$ {increment the domination counter of p}
      end if
    end if
  end for
  if $n_p = 0$ {p belongs to the first front} **then**
    $p_{rank} = 1$
    $F_1 = F_1 \cup \{p\}$
  end if
end for
$i = 1$ {initialize the front counter}
while $F_i \neq \emptyset$ **do**
  $Q = \emptyset$ {used to store members of the next front}
  for all $p \in F_i$ **do**
    for all $q \in S_p$ **do**
      $n_q = n_q - 1$
      if $n_q = 0$ {q belongs to the next front} **then**
        $q_{rank} = i + 1$
        $Q = Q \cup \{q\}$
      end if
    end for
  end for
  $i = i + 1$
  $F_i = Q$
end while

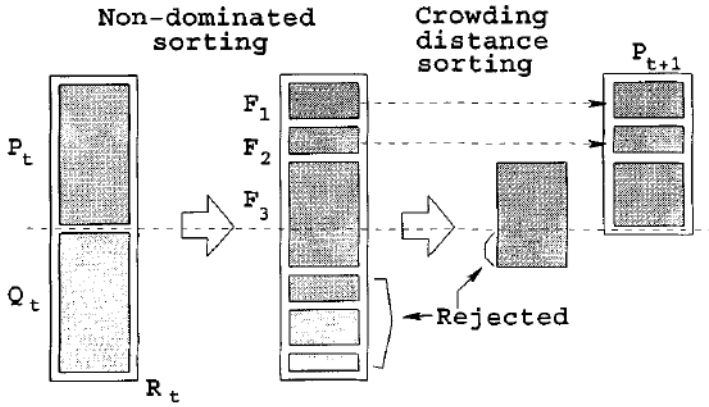


Figure 2.2: The NSGA-II procedure [16].

Algorithm 2.2 Crowding Distance

Input: non-dominated set I
$l = |I|$ {number of solutions I}
for all i **do**
    set $I[i]_{distance} = 0$ {initialize distance}
end for
for all objective m **do**
    $I =$sort(I, m) {sort using each objective value so that boundary points are always selected}
    $I[1]_{distance} = I[l]_{distance} = \infty$
    for $i = 2$ to $(l - 1)$ {for all other points} **do**
        $I[i]_{distance} = I[i]_{distance} + (I[i + 1].m - I[i - 1].m)/(f_m^{max} - f_m^{min})$
    end for
end for

2.1.2. NSGA-III

NSGA-III [17] is designed to handle both multi-(≤ 3) and many-(> 3) objective problems. The latter in particular are problematic for typical MOEAs because as the number of objectives increases, more solutions become non-dominated, genetic recombination becomes less effective, and performance metrics become expensive to compute.

NSGA-III retains NSGA-II non-dominated-sorting approach to candidate selection, but replaces the crowding distance operator for selecting solutions to be preserved from the last front. Given a set of reference points, they are mapped onto a normalised hyperplane that is equally inclined to all objectives. If the reference points are computed using

the Das-and-Dennis method [18] they already lie on the hyperplane and no mapping is required. Reference directions are generated by connecting each reference point to the origin with lines. Each solution is associated with the reference point whose reference line is closest in terms of orthogonal distance. Figure 2.3 shows how this association is performed. The number of solutions associated with each reference point and not in the last front is called the "niche count". One by one, the reference points with the lowest niche count are considered, and if a solution from the last front is associated with it, that solution is selected to be part of the new population and the niche count of that reference point is incremented (the solution with the smallest distance is selected if more than one is associated with that reference point, the reference point is ignored if it has no associated solutions). Since the reference points are well distributed on the hyperplane, their associated solutions are likely to be well distributed on the Pareto front.

Algorithm 2.3 Generation t of NSGA-III procedure

1: **Input**: H structured reference points Z^s or supplied aspiration points Z^a, parent population P_t
2: **Output**: P_{t+1}
3: $S_t = \emptyset, i = 1$
4: Q_t = Recombination+Mutation(P_t)
5: $R_t = P_t \cup Q_t$
6: $(F_1, F_2, ...)$ = Non-dominated-sort(R_t)
7: **repeat**
8: $S_t = S_t \cup F_i$ and $i = i + 1$
9: **until** $|S_t| \geq N$
10: Last front to be included: $F_l = F_i$
11: **if** $|S_t| = N$ **then**
12: $P_{t+1} = S_t$, break
13: **else**
14: $P_{t+1} = \cup_{j=1}^{l-1} F_j$
15: Points to be chosen from $F_l : K = N - |P_{t+1}|$
16: Normalize objectives and create reference set Z^r : Normalize(f^n, S_t, Z^r, Z^s, Z^a)
17: Associate each member s of S_t with a reference point: $[\pi(s), d(s)]$=Associate(S_t, Z^r)
% $\pi(s)$: closest reference point, d: distance between s and $\pi(s)$
18: Compute niche count of reference point $j \in Z^r : \rho_j = \sum_{s \in S_t/F_l}((\pi(s) = j)?1 : 0)$
19: Choose K members one at a time from F_l to construct P_{t+1}: Niching($K, \rho_j, \pi, d, Z^r, F_l, P_{t+1}$)
20: **end if**

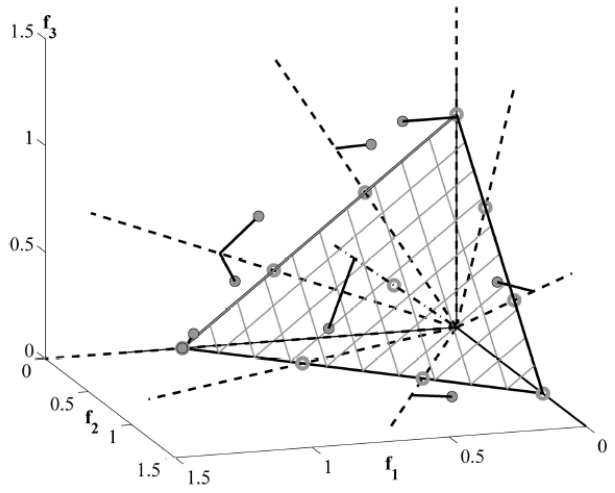


Figure 2.3: Assignment of population members to reference points in NSGA-III [17].

Algorithm 2.4 NSGA-III Niching

Input: $K, \rho_j, \pi(s \in S_t), d(s \in S_t), Z^r, F_l$
Output: P_{t+1}
$k = 1$
while $k \leq K$ **do**
  $J_{min} = \{j : \text{argmin}_{j \in Z^r} \rho_j\}$
  $\bar{j} = \text{random}(J_{min})$
  $I_{\bar{j}} = \{s : \pi(s) = \bar{j}, s \in F_l\}$
  if $I_{\bar{j}} \neq \emptyset$ **then**
    if $\rho_{\bar{j}} = 0$ **then**
      $P_{t+1} = P_{t+1} \cup (s : \text{argmin}_{s \in I_{\bar{j}}} d(s))$
    else
      $P_{t+1} = P_{t+1} \cup \text{random}(I_{\bar{j}})$
    end if
    $\rho_{\bar{j}} = \rho_{\bar{j}} + 1$, $F_l = F_l \setminus s$
    $k = k + 1$
  else
    $Z^r = Z^r / \{\bar{j}\}$
  end if
end while

2.2. Regression

Regression is a supervised machine learning technique that aims to model the relationship between one or more independent input variables and a target variable, when the target variable can take continuous values. As a supervised learning technique, labelled data is required to train a regressor. Regression models are most commonly used as predictive models; once the function that best maps the input variables to the output variable has been estimated from the available data, it can be used to make predictions about the values of the dependent variable given previously unseen input data. Regression is also very useful for understanding the underlying relationship between variables. In simple linear regression, the relationship between the single independent variable x and the dependent variable y is modelled as a straight line, represented by the equation

$$y = \beta_0 + \beta_1 x + \epsilon \tag{2.1}$$

where β_0 is the intercept, β_1 is the slope, and ϵ is the error term that captures the unexplained variation in the data. When the relationship between two or more more independent variables $x_1, ...x_n$ and the dependent variable y needs to be modelled, multiple linear regression is used as a linear combination of the input variables. The formula for multiple linear regression is

$$y = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n + \epsilon \tag{2.2}$$

where β_0 is the intercept, $\beta_1, \beta_2, ..., \beta_n$ are the coefficients associated with the input variables, and ϵ is the error term. Non-linear regression models are used when complex mappings need to be captured, in which case non-linear functions such as polynomials, exponentials, or sigmoids are used to model the relationship.

A variety of methods can be used to estimate the coefficients in the regression equations. The ordinary least squares method achieves the objective by minimising the sum of the squared differences between the observed values of the target variable and the predicted values from the regression equation. When large amounts of data are available, gradient optimisation techniques such as the stochastic gradient descent algorithm are preferred. In this case, an iterative process progressively refines the regression coefficients until a minimum in a loss function is reached. Regularisation techniques, such as Lasso and Ridge regression, can be used to prevent overfitting in regression models. These methods add a penalty term to the loss function, which shrinks the coefficients towards zero and helps to reduce the impact of irrelevant or noisy features in the data.

2.3. Ensemble Learning

Ensemble learning is a machine learning technique that consists of training several different models to solve the same task and then, at inference time, using the knowledge from all of them to make a better final prediction than any of the individual models could make on their own.

There are two main types of ensemble learning, called *bagging* and *boosting*. Bagging involves training multiple models independently on different subsets of the data. These subsets are typically generated by performing random sampling with replacement on the original dataset. The predictions from each model are then combined using an averaging or voting strategy to produce the final prediction. Averaging is used in regression tasks, while voting is used in classification tasks. The use of bagging can significantly improve the performance of unstable learners, i.e. models whose results vary significantly with small changes in the dataset. Boosting, on the other hand, involves the sequential training of several weak models, with each successive model attempting to correct the errors of the previous model. Each time, the training set is re-weighted so that samples where the previous model made an error are given greater weight. The predictions from each model are then weighted and combined to produce the final prediction.

2.4. Convolutions

Convolutional layers are the main building blocks of convolutional neural networks. These layers perform an operation called *convolution* to extract relevant features from the input data. In this context, a convolution is defined as a linear operation consisting of a dot product (i.e. an element-wise multiplication) between a set of weights, organised in structures called *filters* or *kernels*, and a patch of input data of the same size as the filter. The dot product returns a scalar value. By progressively shifting the filter over the input data and repeatedly performing the convolution operation with the current patch, a new set of data derived from the input is obtained, and it is called a *feature map*.

If the input consists of several data channels, as is the case with RGB images, the filter used must have the same number of channels. In this case too, the element-wise multiplication results in a single scalar value. Thus, convolving a filter with the input data always results in a single feature map being produced, regardless of the number of channels. To obtain multiple feature maps, multiple filters must be convolved with the same input data.

The depthwise convolution is a special form of convolution in which each channel of the image is separately convolved with a single-channel filter to produce a corresponding

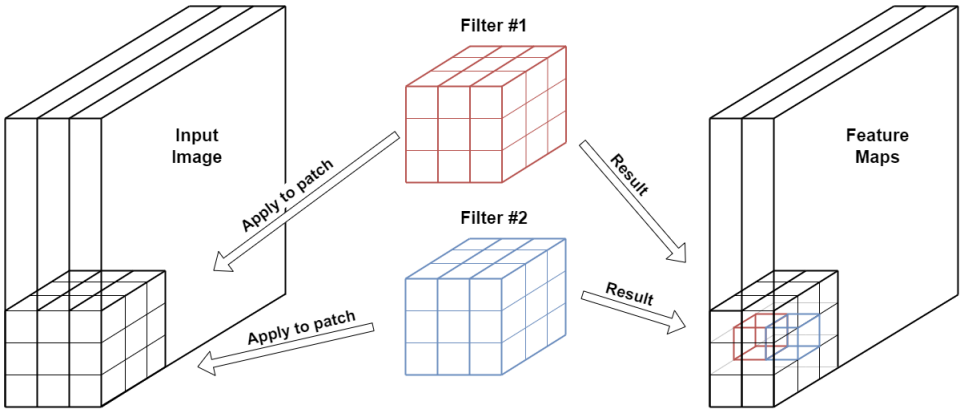


Figure 2.4: Representation of a standard convolution applied to a three-channel input image using two filters.

output feature map. It follows that the number of filters must be equal to the number of channels in the input image, so depthwise convolutions cannot change the number of feature maps.

Classical convolutions are described by the formula:

$$G_{k,l,n} = \sum_{i,j,m} K_{i,j,m,n} \cdot F_{k+i-1,l+j-1,m} \tag{2.3}$$

while depthwise convolutions are described by:

$$\hat{G}_{k,l,m} = \sum_{i,j} \hat{K}_{i,j,m} \cdot F_{k+i-1,l+j-1,m} \tag{2.4}$$

Here G, F, K represent the output feature maps, the input feature maps and the convolution kernel respectively. These formulas assume invariant squared feature maps.

Classical convolutions are more computationally expensive than depthwise convolutions. Being D_k the kernel size, M the number of input channels, N the number of output channels and D_f their dimension, a classical convolution costs:

$$D_k \cdot D_k \cdot M \cdot N \cdot D_f \cdot D_f \tag{2.5}$$

while the computational cost of a depthwise convolutions is:

$$D_k \cdot D_k \cdot M \cdot D_f \cdot D_f \tag{2.6}$$

Depthwise separable convolutions corresponds to the sequential application of a depthwise convolution and a pointwise convolution (a classical convolution with a kernel of size 1x1). The depthwise separable convolution can be considered as the factorisation of a classical convolution, splitting the filtering operation and the recombination of values (performed by the depthwise convolution and the pointwise convolution, respectively). This separation of roles reduces the number of operations required, which in this case is equal to:

$$D_k \cdot D_k \cdot M \cdot D_f \cdot D_f + M \cdot N \cdot D_f \cdot D_f \tag{2.7}$$

for a cost reduction factor with respect to classical convolutions of:

$$\frac{1}{N} + \frac{1}{D_k^2} \tag{2.8}$$

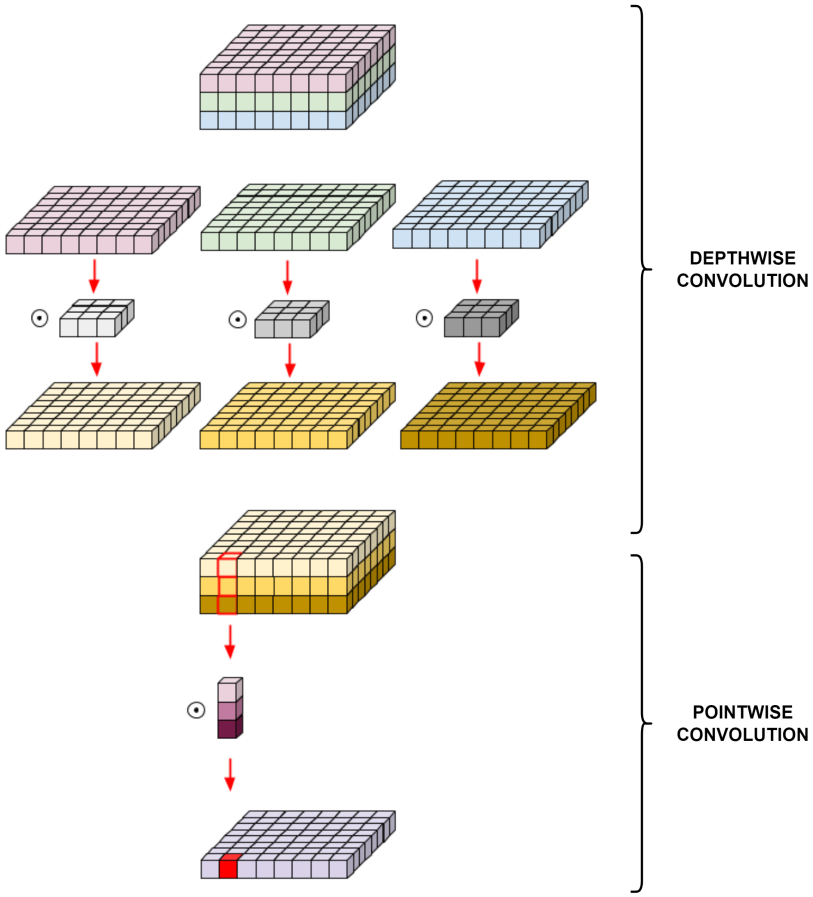


Figure 2.5: Schema of a depthwise separable convolution [19].

2.5. Transfer Learning

Transfer Learning (TL) [20] is the practice of exploiting the knowledge acquired by a well-generalising model trained on a large dataset and reusing it as a starting point for a new learner acting on a different but similar task. Transferring knowledge across domains should help the new model improve its ability to learn and generalise, and it is particularly useful when there isn't much data available from which to learn the new task. When the transferred weights are not kept frozen but instead are refined and adapted to the new data, we talk about Fine-Tuning (FT).

2.6. Knowledge Distillation

In machine learning, the term Knowledge Distillation (KD) [21] refers to a family of compression and acceleration techniques used to distill the knowledge from a large pretrained model (known as the teacher) into a simpler and smaller model (known as the student); in practice, the small model is trained to behave and predict similarly to the teacher model, so that the high accuracy and generalisation capabilities of the teacher are inherited by the student.

The knowledge transfer process typically involves training the larger model on a given dataset and then using it to generate soft targets (i.e. probabilities of the classes rather than one-hot vectors) for the same dataset. These soft targets are then used to train the smaller model, along with the original hard targets (i.e. one-hot vectors). The soft targets provide additional information to the smaller model, helping it to learn from the more complex model.

Knowledge distillation is particularly useful when there is a need to deploy a model, but constraints on computing resources or memory size make it impossible to deploy the large model directly. In this case, the student model can be deployed instead.

2.7. Neural Architecture Search

The term Neural Architecture Search (NAS) [22] refers to a family of techniques and algorithms capable of automatically designing or finding the optimal neural architecture to solve a given task, with minimal or no human intervention in the process. The goal is to achieve better performance than architectures designed by human experts and to improve the efficiency of the design process.

Early NAS techniques were based on Reinforcement Learning (RL) [23, 24], in this case a

controller (often a Recursive Neural Network - RNN) is used to select networks from the search space, these network are then evaluated and a Reinforcement Learning technique such as Q-Learning is used to optimise the controller strategy. Later, NAS techniques based on evolutionary algorithms such as [25, 26] were developed; these searches progress by iteratively modifying a population of architectures via selection, reproduction and mutation operations. A comprehensive summary of the evolutionary construction of neural networks can be found in [27].

One of the main drawbacks of classical NAS techniques is that they require massive amounts of computing resources, therefore many strategies have been developed in recent years to make the use of such algorithms more widespread:

- Defining the NAS search space as the set of feasible neural architecture solutions using a cell-based search space [28–31] rather than a global search space drastically reduces the complexity of the process while having the advantage that the cells are built from known well-performing architectural patterns.
- Transforming the search space into a continuous form so that gradient optimisation techniques can be used, an example of which is [31], which uses a softmax to select the most likely operation among those possible within a cell.
- Taking advantage of the best network already obtained and modifying them [32, 33] rather than picking new ones from the search space.
- Incomplete training, which can be put into practice in two main ways: by avoiding training from scratch or via early-termination. The most common way to avoid training from scratch is the application of weight sharing techniques [11, 33–35]; here subnetworks inherit their weights from a larger network called supernet, so that a full training phase for each of them is not necessary. The main problem that arises when parameter sharing is used is the so-called the “multi-model forgetting” problem [36]; in fact, when weight sharing is used to sequentially train a series of neural architectures, the performance of the previous neural architecture tends to be compromised due to the shared weights being altered. To mitigate this problem, techniques such as a diversity-maximisation loss in [37] and the Progressive Shrinking algorithm in [11] have been proposed. For what concerns early-termination, it consists in interrupting the training of candidate architectures identified as “bad”. This can be done by applying techniques such as the one proposed in [38], which uses a probability method to simulate the learning curve of an architecture from its early learning iterations.

- Utilizing surrogate performance predictors [10, 39–41] instead of fully validating each candidate architecture.
- Decoupling the training and search steps [11, 35, 42] so that training only needs to be performed once.

Currently, NAS is one of the main topics of study in the field of deep learning, and many of today's best networks are the result of its application.

2.8. Squeeze and Excitation Modules

Squeeze and Excitation (SaE) modules have been introduced in SENet [43]. The goal of these modules is to perform feature recalibration on a channel-by-channel basis to emphasise important features while suppressing less informative features.

As the name suggests, the SaE block performs two operations:

1. **Squeeze**: a Global Average Pooling (GAP [44]) operation is applied to the feature maps to produce a "channel descriptor", a tensor of N_c values, where N_c is the number of original feature maps. The use of GAP (or similar) is important to capture global spatial information.
2. **Excitation**: Using a Feed-Forward Neural Network (FFNN) consisting of two fully connected layers performing a reduction-expansion by a factor of r, weights for the N_c channels are derived by applying a sigmoid function to the outputs. The two fully connected layers are interleaved by a Rectified Linear Unit (ReLU) [45] activation function. The original feature maps are then multiplied by the excitation weights just obtained on a channel-by-channel basis, resulting in a set of recalibrated feature maps.

SaE modules are computationally inexpensive, both in terms of additional parameters and latency, and can be easily integrated into existing architectures.

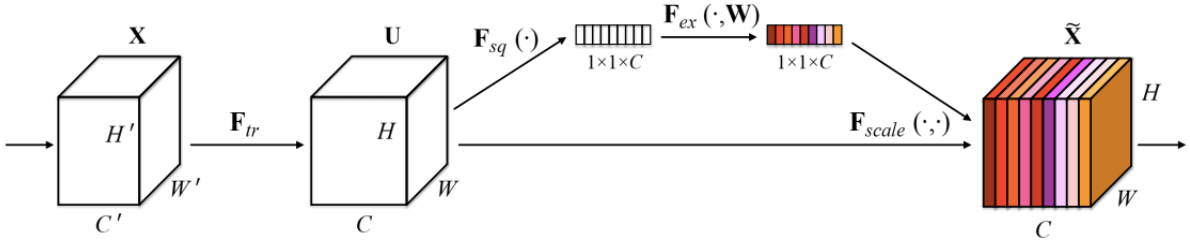


Figure 2.6: Inner workings of a squeeze and excitation block [43].

2.9. MobileNets

The MobileNet family is a group of convolutional neural networks designed to work in constrained environments, therefore these networks are smaller and have lower latency than typical neural networks. The network used as a starting point for this thesis is based on MobileNetV3, so the structure of this network and the relevant improvements brought by its predecessors will now be explained.

2.9.1. MobileNet

MobileNet [46] is a network based on depthwise separable convolutions, using a Batch Normalization layer (BN) [47] and ReLU activation function after each depthwise and pointwise convolution. The spatial reduction of the feature maps is achieved by using stride 2 in some of the depthwise convolutions.

A hyperparameter called Width Multiplier (WM) is introduced in this paper, it is a multiplicative factor that linearly scales the number of filters used by each block of the network.

2.9.2. MobileNetV2

MobileNetV2 [48] improves on its predecessor by introducing the Inverted Residual Bottleneck (IRB) block, the structure of which is shown in Figure 2.7. In each block, BN layers are present after each convolution, and are followed by ReLU6 activation functions [49] with the exception of the last convolution. The MobileNetV2 paper empirically demonstrates the importance of removing nonlinearities after the last convolution of the IRB in order not to lose representational power.

The main advantage of using IRB blocks is that they are very memory efficient. The residual connection is omitted when the size or number of feature maps varies between input and output; henceforth, these blocks will simply be referred to as Inverted Bottleneck (IB) blocks. IBs are usually the first block in a series of IRBs that all produce the same number of output feature maps. They are responsible for changing the number of feature maps via the 1x1 convolution, and for reducing the spatial dimension of those same feature maps via the depthwise convolution if the stride value is greater than 1.

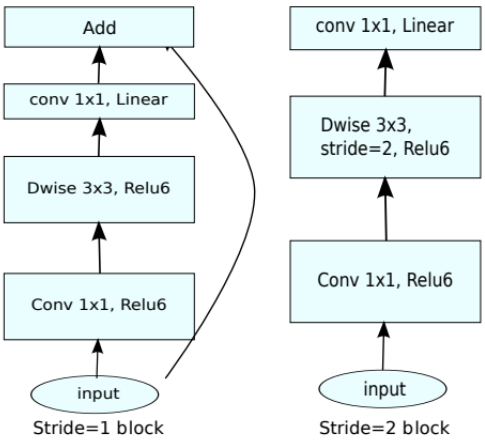


Figure 2.7: MobileNetV2 inverted residual bottleneck block [48].

2.9.3. MobileNetV3

There are two MobileNetV3 [7] models: Large and Small; both have been used in this thesis, but the main work has been done on MobileNetV3Large. MobileNetV3 further refines the architecture of MobileNetV2 by introducing in some of the IRB/IB blocks (in positions corresponding to those in MnasNet [29]) a SaE module after the depthwise convolution, so that the SaE block works on the expanded representation. For reference, see Figure 2.8. The size of the SaE is fixed to 1/4 of the number of channels in the expansion layer.

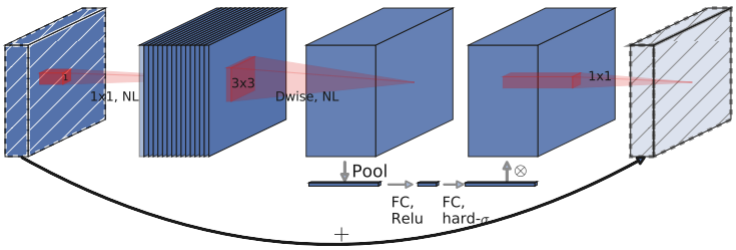


Figure 2.8: MobileNetV3 inverted residual bottleneck block [7].

Another alteration is the replacement of the ReLU activation function with the Hard Swish activation function in the first layer and in the second part of the network. The Hard Swish activation function is defined by the following formula:

$$\text{H-swish}(x) = x\frac{\text{ReLU6}(x+3)}{6} \tag{2.9}$$

Finally, the last 1x1 convolution was moved after a GAP layer to reduce latency.

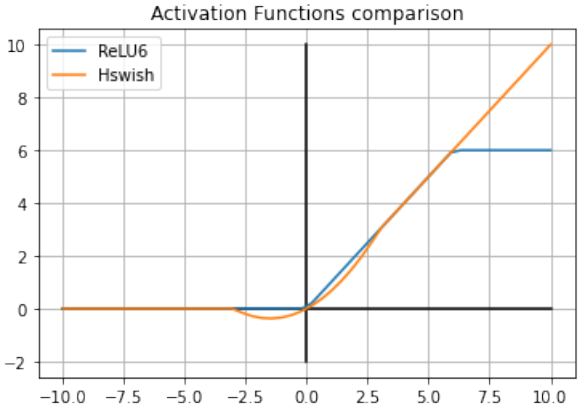


Figure 2.9: Hswish compared to ReLU6

NAS techniques were used at both block level and layer level (NetAdapt algorithm [50]) to optimise the structure of MobileNetV3. Table 2.1 shows the architecture of MobileNetV3Large.

Table 2.1: Structure of MobileNetV3Large [7].

Input	Operator	Expand to	Out	SaE	Activation	Stride
224^2 x 3	conv2d	-	16	-	Hswish	2
112^2 x 16	bneck, 3x3	16	16	-	ReLU	1
112^2 x 16	bneck, 3x3	64	24	-	ReLU	2
56^2 x 24	bneck, 3x3	72	24	-	ReLU	1
56^2 x 24	bneck, 5x5	72	40	✓	ReLU	2
28^2 x 40	bneck, 5x5	120	40	✓	ReLU	1
28^2 x 40	bneck, 5x5	120	40	✓	ReLU	1
28^2 x 40	bneck, 3x3	240	80	-	Hswish	2
14^2 x 80	bneck, 3x3	200	80	-	Hswish	1
14^2 x 80	bneck, 3x3	184	80	-	Hswish	1
14^2 x 80	bneck, 3x3	184	80	-	Hswish	1
14^2 x 80	bneck, 3x3	480	112	✓	Hswish	1
14^2 x 112	bneck, 3x3	672	112	✓	Hswish	1
14^2 x 112	bneck, 5x5	672	160	✓	Hswish	2
7^2 x 160	bneck, 5x5	960	160	✓	Hswish	1
7^2 x 160	bneck, 5x5	960	160	✓	Hswish	1
7^2 x 160	conv2d, 1x1	-	960	✓	Hswish	1
7^2 x 960	pool, 7x7	-	-	-	-	1
1^2 x 960	conv2d, 1x1, No BN	-	1280	-	Hswish	1
1^2 x 1280	conv2d, 1x1, No BN	-	k	-	-	1

3 | State Of The Art

In this chapter, which is divided into three sections, an overview of the main works in the literature on which this thesis is based is given. The first section focuses on research related to the introduction and use of early exits in neural networks, mainly but not exclusively in convolutional neural networks, and the different techniques for constructing an ensemble with these new exits. In the second section, Once-For-All is explained, focusing in particular on how its Progressive Shrinking training algorithm works. Finally, the last section describes the iterative Neural Architecture Transfer algorithm and each of its steps, as well as the techniques used to make the whole process efficient.

3.1. Early Exit Neural Networks

In recent times, there has been growing interest in the study of neural networks endowed with early exits. Such networks consist of a backbone structure to which branches containing predictor layers are attached at intermediate levels between the input layer and the traditional single output layer. Determining the optimal number of branches and their placement is a crucial aspect in this context, however, since the number of possible placements increases exponentially with the size of the backbone, this is still an active area of research. Branches could also have their own branches, but we won't consider this case.

Incorporating early exits into neural network models can significantly improve their efficiency, enabling them to make predictions before the input data has been processed by all layers, thereby saving time and computational resources. This technique is particularly effective in image classification tasks, where most samples are not too complex and do not require the full power of the neural network to be correctly classified.

The work of Panda *et al.* was among the first to apply the early exit technique to convolutional neural networks [51]. Their goal was to develop an algorithm that could determine the optimal depth of the classification network under study, allowing for dynamic adjustments in computational effort without sacrificing accuracy. In a parallel

effort, BranchyNet [52] investigated early exits in CNNs, with a focus on demonstrating the predictive abilities of classifiers placed at intermediate points in the network, so that the easiest samples could be processed with fewer hidden layers, and more challenging ones with the entire architecture.

A more recent approach to reducing the energy cost and complexity of single-output convolutional networks has been proposed by Wang *et al.*, achieving an extremely favourable trade-off between accuracy and floating-point operations [53]. Their technique uses early exits and weighted loss functions on architectures with skip connections. Additionally, Pacheco *et al.* highlighted the significant advantages of using early exit architectures in the field of edge computing [54]. In particular, they demonstrated how the ability to classify non-anomalous samples at the shallow levels of a CNN allows for no loss in performance compared to single-exit classification.

The most common way to take advantage of early exit networks has often been to have their exits sequentially produce predictions until from one of them a confidence value above/below a threshold is derived, there the computation is halted and the sample classified. The confidence value can be derived from a variety of metrics, such as the predicted classes probabilities [51, 55, 56](↑), the measure of entropy [52](↓), the loss [53](↓) or the number of votes [57](↑). In [58] the exit selection logic is instead trained together with the network.

However, the exit-by-exit decision on whether to stop the computation is not the best strategy to employ, because even if the early classifiers are not performant or confident enough, the information they produce can still be valuable. A better approach would be to devise an aggregation strategy to exploit their joint potential. Newer techniques don't let the previous outputs information (for which computation has already been spent) go to waste, instead they take advantage of it, combining results in an ensemble-like manner to better decide whether to stop the computation or not. Examples of this are the works by Wołczyk *et al.* [56] and Sun *et al.* [57].

Early exits ensembling is an interesting approach because the models that make up the ensemble share some of their structure and parameters, but are able to work on different features given the same input data. An advantage of using this ensembling method is that only one model needs to be trained and loaded, rather than several different ones. This could be particularly useful where there are constraints on training time or available memory.

For what concerns the training of early exit networks, joint training is the most common approach and consists in formulating a single optimisation problem whose loss depends on

all branches, in particular the network loss is often calculated as the weighted sum of the branches' losses [52, 55, 59]. Strategies have also been devised to dynamically adjust the exits losses weights during training [53] and to improve the ensemble by combining the prediction loss with a a diversity loss [57, 60]. When early exits are trained jointly with the backbone network, they favor the learning of more discriminative features at each layer and lead to faster convergence, while also acting as regularization (stronger if larger weights are given to earlier exits), reducing overfitting and mitigating the vanishing/exploding gradient problem [52, 61]. An overview on the use of early exits can be found in [59].

The output of an early exits ensemble network can also be computed in a variety of ways, for example as the arithmetic mean of the outputs [62], as the arithmetic mean of the predictions [55, 60], via geometric ensemble [56], or via voting strategies [57].

The effectiveness of early exits ensembles is not limited to image classification, in fact they've also recently been used for image captioning [63], natural language processing [57], for uncertainty quantification and biosignal classification [60, 64], and to improve robustness against adversarial attacks [55]. Moreover, early exits ensembles were employed to produce a teacher-free knowledge distillation technique by treating the ensemble prediction as the teacher prediction [65]. In [62], a technique for fine-tuning early exit networks by only adapting a subset of classifiers using differently pre-processed data has been proposed.

3.2. Once-For-All

The study and application of NAS techniques has revolutionised the field of deep learning in recent years, resulting in extremely powerful networks being obtained. However, most NAS techniques have some important drawbacks: the process is very slow, they require massive amounts of computational resources, and the models produced are often so constrained to the specific task or hardware that it is almost impossible to reuse these models in a different scenario. When this need arises, the only solution may be to start the whole process from scratch. This is extremely costly in terms of time, computational resources required and energy consumed.

In this context, the NAS technique called Once-For-All (OFA) [11] has been proposed as a solution for designing high-performance neural networks that are easily adaptable to different hardware configurations while minimising power consumption. OFA achieves this goal by training a single large model with a dynamic architecture, called *supernet*, through the smaller architectures it contains, called *subnets*. This is done using a special training algorithm called Progressive Shrinking (PS), which consists of many phases to progressively fine-tune smaller and smaller networks from within the supernet.

Once the supernet has been successfully trained, a search step is responsible for finding and extracting the subnets that best adapt to some specific platform and resource constraints. As the training and search steps are decoupled, the search process can be run as many times as required without the need to repeat the training phase. This means that the OFA training step, although computationally expensive, is a one-time cost that is amortised as the number of searches increases.

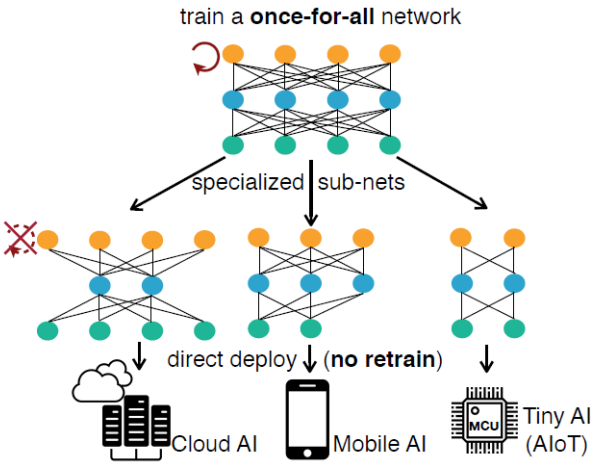


Figure 3.1: Once-For-All: Train a single network, then select the best subnetwork within it for any given deployment scenario without the need for retraining [11].

3.2.1. The OFAMobileNetV3 Network

Among the major macro-architectures presented by the authors, the primary one, referred to as OFAMobileNetV3 and presented in Table 3.1, will serve as the basis for most of this thesis work. The OFAMobileNetV3 network is based on MobileNetV3Large (Table 2.1), but is larger because it will be up to the combination of OFA's train and search steps to find the best subnet. By default, the network consists of 5 stages, each consisting of 4 blocks. Its structure is also based on Inverted Residual Bottleneck (IRB) blocks, which are replaced by Inverted Bottleneck (IB) blocks when residual connections cannot be used.

Table 3.1: Structure of the OFAMobileNetV3 network, for width multiplier 1.0

Stage	Layer	Out Channels	Stride	Activation	BN	SaE
Head	conv3x3	16	2	Hswish	✓	✗
	IRB	16	1	ReLU	✓	✗
Stage 1	IB	24	2	ReLU	✓	✗
	IRB	24	1	ReLU	✓	✗
	IRB	24	1	ReLU	✓	✗
	IRB	24	1	ReLU	✓	✗
Stage 2	IB	40	2	ReLU	✓	✓
	IRB	40	1	ReLU	✓	✓
	IRB	40	1	ReLU	✓	✓
	IRB	40	1	ReLU	✓	✓
Stage 3	IB	80	2	Hswish	✓	✗
	IRB	80	1	Hswish	✓	✗
	IRB	80	1	Hswish	✓	✗
	IRB	80	1	Hswish	✓	✗
Stage 4	IB	112	1	Hswish	✓	✓
	IRB	112	1	Hswish	✓	✓
	IRB	112	1	Hswish	✓	✓
	IRB	112	1	Hswish	✓	✓
Stage 5	IB	160	2	Hswish	✓	✓
	IRB	160	1	Hswish	✓	✓
	IRB	160	1	Hswish	✓	✓
	IRB	160	1	Hswish	✓	✓
Tail	conv1x1	960	1	Hswish	✓	✗
	GAP	960				
	conv1x1	1280	1	Hswish	✗	✗
	Linear	n classes				

3.2.2. Training via Progressive Shrinking

The core algorithm of OFA, which allows the creation of optimal subsets within the supernet, is called Progressive Shrinking (PS). As the name suggests, the idea is to start training from a large network and gradually increase the number of subnets that can be activated and trained in successive phases by adding subnets of progressively reduced complexity to the sampling space (the set of the possible activatable subnets up to that particular PS phase). Smaller subnets are obtained by reducing the number and sizes of convolutional filters.

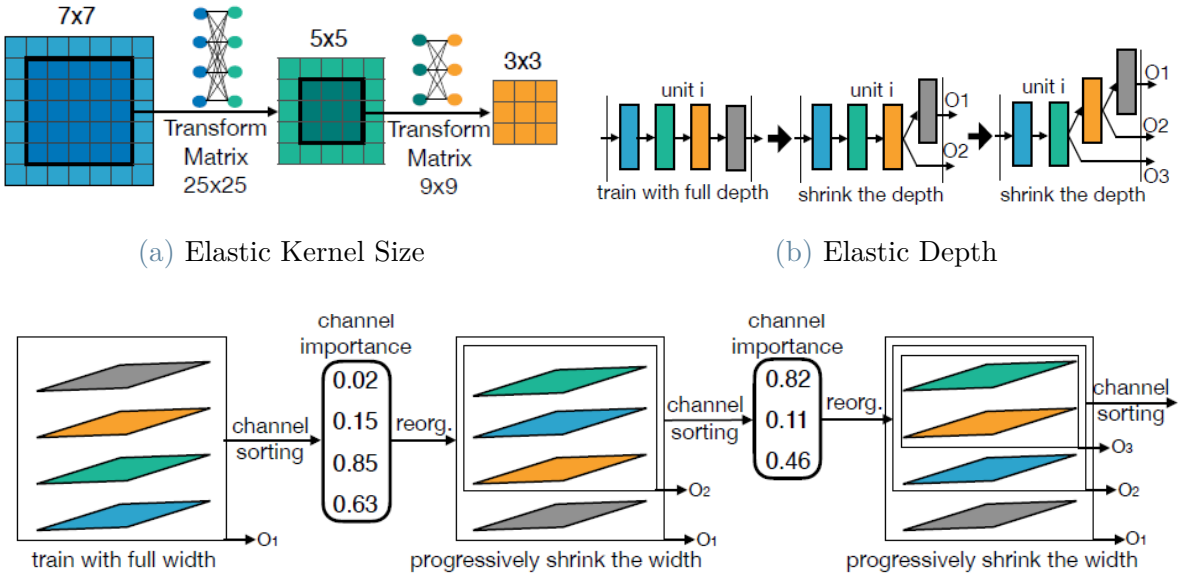
The results proposed by the authors show that, thanks to the use of the Progressive Shrinking algorithm, OFA models are able to overcome the "multi-model forgetting" problem [36], i.e. the problem that arises when weight sharing is used to sequentially train a number of neural architectures within the supernet, which generally leads to severe performance degradation due to subnetworks interfering with each other. Furthermore, by using PS there is no need to enumerate and process all the subnets, which in the case of OFA would be $\approx 10^{19}$, a clearly infeasible task.

The PS algorithm is divided into four "elastic steps", some of which consist of several phases. Considering that each phase corresponds to a complete training run of the supernet, the elastic steps proposed by the authors are the following:

- **Elastic Resolution (ER)**: the size of the input images entering the network varies randomly for each batch with respect to the set of default values [128, 160, 192, 224]. The set of possible image sizes can be extended to include all sizes in steps of 4 between the minimum and maximum values in the list above. This step involves a single phase, therefore all image sizes can be sampled from the start.
- **Elastic Kernel Size (EKS)**: subnets can be extracted from the supernet so that the kernel size of each IRB/IB block can be selected from the set of values [3,5,7]. The weights associated with these convolution operations are computed throughout the training of the supernet. Starting from the kernel of size 7, the equivalent versions of the kernels of size 5 and 3 are computed using two small neural networks, with the aim of learning the matrix transformations that convert the kernels from size 7 to 5 and then from 5 to 3. This step also involves a single phase.
- **Elastic Depth (ED)**: subnets can be extracted from the supernet such that, for each stage, only the first d of $k = 4$ blocks are active. Before this elastic training step is activated, all phases use a of $d = 4$. Once activated, Elastic Depth consists of two phases: in phase 1, the d value can be selected in [3,4], while in phase 2 the

d value can be selected in [2,3,4].

- **Elastic Width (EW)**: subnets can be extracted from the supernet such that the expansion ratio of each IRB/IB block can be selected from a list of allowed values. Before this elastic training step is activated, the width, i.e. the number of channels within the intermediate convolutions of the IRB/IB blocks, is set to the maximum value chosen by the authors, which corresponds to those in Table 3.1 multiplied by a factor $e = 6$. Once activated, this elastic step consists of two phases: in phase 1, the e value can be chosen in [4,6], while in phase 2, the e value can be chosen in [3,4,6]. Whenever values of e less than 6 are selected for a block, the algorithm orders the channel weights of the intermediate layers hierarchically according to their L1 norm. It then calculates the number of channels to be used in the current phase and fills them with the best weights taken from the ranking.



(a) Elastic Kernel Size

(b) Elastic Depth

(c) Elastic Width

Figure 3.2: Elastic steps acting on the dynamic OFA supernet [11].

At the beginning of the algorithm, the maximal network is defined, and it corresponds to the network architecture with all PS parameters set to their maximum value. The training steps and phases of PS are then executed in the order described above. All values unlocked by a given phase for a given elastic step remain available for selection by all subsequent training phases, allowing smaller and smaller networks to be added to the sampling space. The training of the maximal network corresponds to the first step of the process, and the resulting network is reused in all subsequent phases of PS as the teacher network to perform knowledge distillation [66]. The OFA algorithm uses a form

of KD where the loss of the student network is computed as the sum of its cross-entropy loss and the loss computed on the soft labels of the teacher network, where this second term is weighted by a KD ratio factor. From the moment Elastic Kernel is activated, for each batch of images, a certain number of subnets are sampled from the current sampling space, their gradients are cumulated, and then a single weight update step is performed. The progressive addition of smaller subnets to the sampling space, whose weights are also derived from the tuning of progressively smaller networks, minimises the chance of interference between subnets, thus limiting the impact of the multi-model forgetting problem.

Algorithm 3.1 Progressive Shrinking, Train One Epoch

```
for batch = 0 to size(DataLoader) − 1 do
    batch_images, batch_labels = DataLoader(batch)
    ElasticResizing.update(batch)
    if kd_ratio > 0 then
        teach_sl = TeacherNet.getSoftLabels()
    end if
    subents_losses = [ ]
    for j = 1 to Nsubnets do
        subnet_config = Network.sampleSubnetConfig()
        Network.activateSubnet(subnet_config)
        output = Network(batch_images)
        if kd_ratio = 0 then
            loss = LossCriterion(output, batch_labels)
        else
            kd_loss = KDLossCriterion(output, teach_sl)
            sub_loss = LossCriterion(output, batch_labels)
            loss = sub_loss + kd_loss · kd_ratio
        end if
        subents_losses.append(loss)
        Network.updateAccuracy(output, batch_labels)
        loss.backpropagate()
    end for
    Network.optmizierStep()
end for
return subents_losses.avg(), Network.getAccuracy()
```

3.2.3. Search Step

Once the supernet training process is complete, the search process can begin. The OFA search process starts by randomly sampling a set of 16K subnets from the trained supernet, along with their input image sizes. All these subnets are evaluated in terms of accuracy on 10K validation images, and in terms of latency on a number of hardware platforms. The data obtained is used to train an accuracy predictor (3 layers, 400 units per layer FFNN) and a latency lookup table. Given some constraints to be met by the desired subnet, the predictor guided search uses an evolutionary algorithm to find the best solution.

3.3. Neural Architecture Transfer

Neural Architecture Transfer (NAT) [10] is a very recent NAS technique, and like OFA it proposes a new way to get around the problem of having to run NAS algorithms multiple times to find networks that are suitable for different constraints or hardware platforms. The goal of NAT is to progressively transform a pretrained generic supernet into a task-specific supernet, from which subnets that achieve the best possible tradeoff between a wide range of objectives can be directly searched and extracted, without the need for re-training. This is achieved by repeatedly executing, in an alternating manner, a phase of transfer learning to optimise the supernet, and a many-objective evolutionary search to find subnets that are currently on the objectives tradeoff front.

To make the supernet adaptation process as efficient as possible, only those parts of the supernet corresponding to subnets whose structure can be directly sampled from the distribution of those lying on the tradeoff front are fine-tuned, rather than all possible subnets. In this way, computation can be saved by adapting only those parts of the supernet that effectively contribute to improvements in the current task.

As far as the search process is concerned, the many-objective evolutionary search is accelerated and guided by a performance predictor model that is fitted online using only the best subnets and their scores. This practice allows NAT to save a lot of time that would be otherwise be spent evaluating the each member of the intermediate populations produced by the evolutionary search. It also helps to keep the performance of the predictor high despite using a rather small number of subnets as its training samples.

Step by step, the best architectures found are progressively added to an archive. At the end of the process, NAT returns the set of non-dominated subnets from the archive, the set of high-tradeoff subnets (i.e. those for which choosing any of the neighbour solutions would result in the largest average loss in some objectives with respect to a unitary gain in others), and finally the obtained task-specific supernet, which could be further reused as a starting point for a new NAT process in the context of a different deployment scenario. Figure 3.3 shows an overview of the NAT process.

Although the NAT algorithm was theoretically designed to work for a generic supernet and a correspondingly designed search space, the main study in the NAT paper was carried out using as initial supernet, or more precisely supernets, the PS-trained OFAMobileNetV3 architectures for width multipliers 1.0 and 1.2. This is the reason why it was necessary to study the methods that were presented and used in Once-For-All.

Starting from the supernets trained on ImageNet, 11 datasets spanning a large variety of

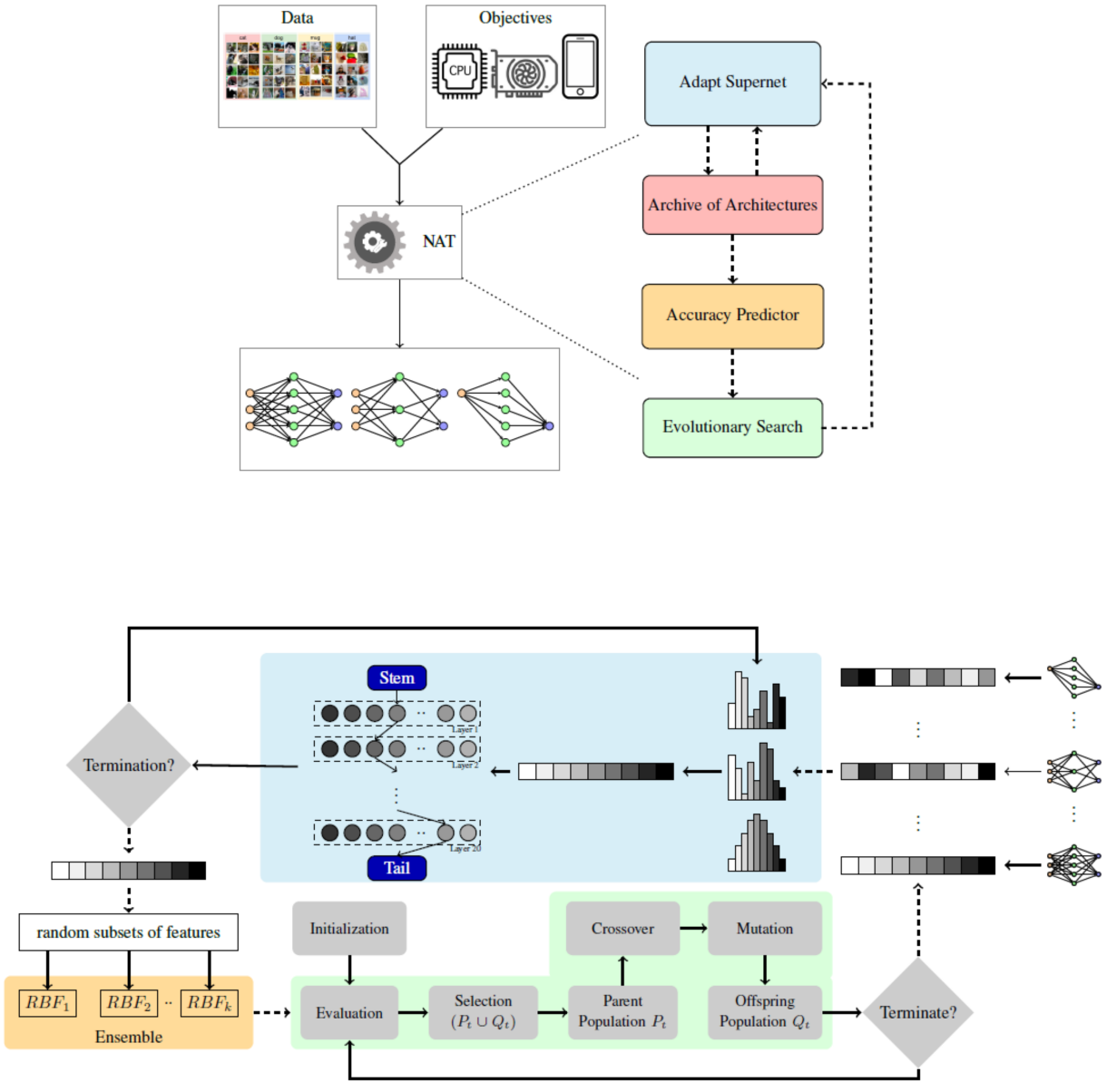


Figure 3.3: NAT overview [10]

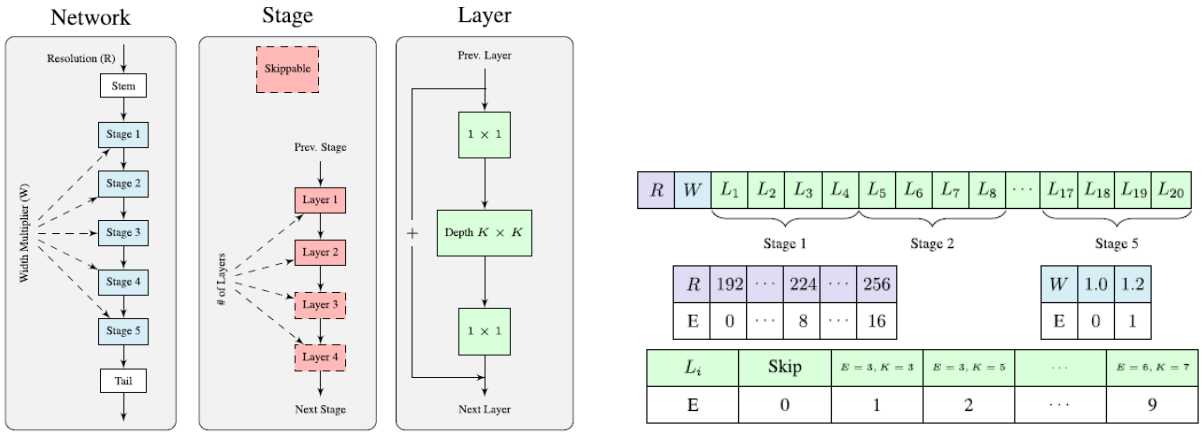
sizes and number of classes were used as targets for the NAT technique. Their properties are shown in Table 3.2.

Table 3.2: NAT benchmark datasets.

Dataset	Train Size	Test Size	Classes
ImageNet [1]	1 281 167	50 000	1 000
CINIC-10 [67]	180 000	9 000	10
CIFAR-10 [68]	50 000	10 000	10
CIFAR-100 [68]	50 000	10 000	10
STL-10 [69]	5 000	8 000	10
Food-101 [70]	75 750	25 250	101
Stanford Cars [71]	8 144	8 041	196
FGVC Aircraft [72]	6 667	3 333	100
DTD [73]	3 760	1 880	47
Oxford-IIIT Pets [74]	3 680	3 369	37
Oxford Flowers102 [75]	2 040	6 149	102

3.3.1. Search Space and Encodings

The many-objective evolutionary search of NAT requires the solutions (i.e. the subnets) to be represented in an encoded form. In particular, for the OFAMobileNetV3 architecture, NAT proposes to represent their configurations using an integer encoding consisting of strings of 22 values. In each compressed representation, the first value corresponds to the encoding of the resolution R of the image to be fed to the network. Different image sizes can be used because the concept of Elastic Resolution has been carried over to NAT. The second value represents the encoding of the active width multiplier W. The following 20 encoded values represent, for each of the 20 internal IRB blocks, the pairs of values formed by the active kernel size and expansion ratio (K,E). Recall that the backbone of the OFAMobileNetV3, already presented in Section 3.2.1, can be conceptually divided into 5 stages composed of 4 IRB/IB blocks each, for a total of 20 blocks. The set of initial layers (or "stem") and the set of final layers (or "tail") are common to all subnets, therefore they are and not subject to the NAT search and therefore are not represented in the encoded strings. The values assigned to create the integer encoding start from zero and increase sequentially. If the encoded value for an IRB block is zero, it means that the block is being skipped due to Elastic Depth. In accordance with the PS algorithm used to train the supernets, only the 3rd and 4th IRB blocks within each stage can be skipped.



(a) Network search space

(b) Search space encoding

Figure 3.4: NAT search space [11].

3.3.2. Error Predictor

Due to the large number of subnets generated during the execution of the evolutionary search process, the mere need to evaluate each of their performances would make the algorithm computationally impractical, regardless of the use of weight sharing. For this reason, instead of performing inference on each of the subnets, NAT relies on the use of a performance predictor model, the adoption of which reduces the time required to evaluate candidate solutions from minutes or hours to seconds.

The predictor performs a regression task and is trained online. At the start of each NAT iteration, the predictor model is fitted by being given as input the set of encodings corresponding to the subnets currently stored in the archive, and their top1 error as targets. Based on this data, the predictor model must learn to predict the error value for previously unseen architectural encodings.

The use of architectures that are close to, or directly on, the current objectives tradeoff front to train the error predictor model leads to more reliable (i.e. with high correlation to actual performance) and consistent (i.e. with consistent quality across datasets) predictions, as well as minimising the number of architectures needed as a training samples to obtain an accurate model.

There are two main differences between what is described in the paper and what appears in the code regarding the predictor:

- The features extracted from encoding of the architectures and fed to the predictor are in the form of a one-hot encoding rather than an integer encoding.

- The performance predictor, previously structured as an ensemble of Radial Basis Function (RBF) models [76], has been replaced by an ensemble of Light Gradient Boosting Machine (LGBM) models [77].

3.3.3. Warmup Phases

Before starting the execution of the NAT algorithm, a series of three warmup phases are performed, consisting of fine-tuning the two initial pretrained supernets on the NAT target dataset. The first two warmup phases can be performed in parallel as they consist of extracting the maximal networks from each of the two supernets independently, and adapting them to the new dataset after replacing the classifier layer. In the third phase, the weights obtained from the previous two phases are loaded into two new dynamic OFAMobileNetV3 supernets, managed by a structure capable of switching between them by selecting the active width multiplier. Then the fine-tuning starts, for each batch a number of subnets within the supernets are activated and along these structures the supernets are further tuned on the new dataset. The methodology for activating and training subnets within the supernet is the same as in Once-For-All. In this third phase, the network obtained in phase 2, i.e. the maximal network for width multiplier 1.2, is used as a teacher network to perform knowledge distillation on the active subnets.

3.3.4. The NAT Algorithm

Algorithm 3.2 Neural Architecture Transfer

Input: Training data D_{tr}, Validation data D_{val}, Objectives $objs$, Supernet S, number of iterations N_{iter}, Initial archive size A_s, finetuning epochs e_{ft}, number of architectures to search K, number of top architectures $topN$, number of subents to return N_{ret}
$archive$ = initialize_archive(A_s, S, D_{val}, $objs$)
for $iter = 1$ to N_{iter} **do**
  $predictor$ = fit_predictor($archive$)
  $candidates$ = evolutionary_search(S, $archive$, $predictor$, $objs$, K)
  $candidates\_metrics$ = candidates_evaluation($candidates$, $objs$, S, D_{val})
  $correlation$ = predictor_evaluation($predictor$, $candidates$, $candidates\_metrics$)
  archive_growth($archive$, $candidates$)
  $distributions$ = model_distributions($archive$, $topN$)
  adapt_supernet(S, D_{tr}, D_{val}, e_{ft}, $distributions$)
  $archive$ = update_archive($archive$, $objs$, S, D_{val})
  save_iteration(S, $archive$, $correlation$)
end for
Return: S, non_dominated($archive$, $objs$), high_tradeoff($archive$, $objs$, N_{ret})

The following list describes each of the NAT steps shown in Algorithm 3.2.

1. **Archive Initialisation**: First, a number $A_s - 2$ of architecture encodings are uniformly sampled from the search space, i.e. for each of the 22 variables that make up the encoding, a value is chosen via uniform sampling among the possible encodings for that variable. In addition, the two encodings corresponding to the maximal and minimal networks are also considered. The presence of these two extreme encodings aids exploration during the search process. The encodings are then converted into valid network configurations. The corresponding subnets are then extracted from the supernet and evaluated in terms of validation accuracy and any additional objective specified, which may be the number of parameters, the number of MACs or the latency. Once all the architectures have been evaluated, the archive is constructed as a list of their configurations with the corresponding values for each objective.

2. **Predictor fitting**: For each architecture in the archive, its top1 error is calculated from the recorded accuracy. Then, the configurations of the architectures are encoded, and from each encoding a corresponding set of features is extracted. Finally, the predictor model is fitted using the features as independent variables and the errors as target variables. The ensemble is constructed from the the different models obtained after performing a 10-fold cross-validation.

3. **Evolutionary search**: The search process consists of two parts: a *candidate finding problem* and a *subset selection problem*. The goal of the first problem is to find a set of new candidate architectures, from which the second problem selects a subset to be evaluated with high fidelity and added to the archive. The technique used to solve the candidate finding problem depends on the number of objectives. If the only objective is accuracy, a genetic algorithm is used. For a number of objectives equal to two, NSGA-II is used. If instead the number of objectives is three or more, NSGA-III is used. For all these optimisation techniques, a uniform crossover operation with probability 0.9 is used in conjunction with polynomial mutation. The architectures belonging to the populations generated during the running of the problem are evaluated accurately in terms of all additional objectives, instead their classification performance is estimated using the error predictor. The set of solutions obtained is then compared with the architectures already in the archive to avoid duplication; if matches are found, such candidate solutions are discarded. The subset selection problem is solved using a genetic algorithm that produces as solutions binary strings representing which K candidate architectures should be kept and added to the archive, this selection is based on the scores for the additional

objectives. If the final solution points to less than K architectures, the candidate architectures with the best predicted accuracy are added to the set of solutions so that the desired number K is reached.

4. **Candidates evaluation**: The architectures obtained as a result of the search step are evaluated in terms of all the objectives. Unlike the previous step, where the predictor is used to estimate the performance of the networks, here their accuracy is precisely evaluated on validation images.

5. **Predictor evaluation**: Given the real error values for the survived architectures, as well as their predicted errors, the performance of the error predictor model is evaluated by calculating the correlation between the two sets of measures. The measured correlation coefficients are Pearson's (r,[78]), Spearman's (ρ,[79]) and Kendall's (τ,[80]). The Root Mean Square Error (RMSE) is also computed.

6. **Archive growth**: The newly found architectures are added to the archive.

7. **Model Distributions**: The best *topN* architectures in the archive are selected and encoded. Then, a set of arrays equal in size to the number of variables in the encodings is created. The M^{th} array will contain all the values of the M^{th} positions for the encodings of the architectures. Finally, a set of probability distributions are fitted to the data, so that the most representative distribution is selected for each position. The list of distributions is returned.

8. **Supernet adaptation**: The supernet is fine-tuned for a number e_{ft} of epochs. In particular, for each batch of images, a number of architectural encodings are obtained by sampling values for the variables from the set of distributions obtained in the previous step. These encodings are then converted into subnet configurations that are sequentially activated within the supernet. The gradients computed after passing the images in the batch through the activated subsections of the supernet are accumulated, and then a single optimisation step is performed.

9. **Archive update**: Because of the fact that the weights of the supernet have been updated, and that subnets inherit their weights from the supernet due to weight sharing, the previously computed accuracies of the subnets in the archive are no longer correct. In this step, for each subnet in the archive, the new weights are inherited from the supernet and the classification performance is accurately recalculated. The new values are then substituted to the old ones.

10. **Save iteration**: The supernet checkpoint, the configurations of the architectures in the archive and their scores, the correlation metrics calculated for the error predictor

and the hypervolume metric are all saved.

11. **Return**: The NAT algorithm returns not only the updated supernet, but also the set of non-dominated architectures contained in the archive, as well as a number N_{ret} of "high tradeoff" architectures. Such architectures are defined as those with the highest ratio between the average objective loss and average objective gain among pairs of solutions in the set of m nearest neighbours $B(i)$, as described by Equation 3.3 A solution with the highest tradeoff value indicates that it causes the largest average loss in some objectives to make a unit average gain in other objectives to choose any of its neighbors. The formulas describing average loss and average gain are reported in Equation 3.1 and Equation 3.2 respectively.

$$Avg\ Loss(i,j) = \frac{\sum_{k=1}^{m} \max(0, f_k(j) - f_k(i))}{\sum_{k=1}^{m} \{1 | f_k(j) > f_k(i)\}} \tag{3.1}$$

$$Avg\ Gain(i,j) = \frac{\sum_{k=1}^{m} \max(0, f_k(i) - f_k(j))}{\sum_{k=1}^{m} \{1 | f_k(i) > f_k(j)\}} \tag{3.2}$$

$$Tradeoff(i) = \max_{j=1,\ j \in |B(i)|} \frac{Avg\ Loss(i,j)}{Avg\ Gain(i,j)} \tag{3.3}$$

4 | Anticipate, Ensemble and Prune

Motivated by the intention to later add early exits to OFA and NAT networks, this chapter presents a large-scale study analysing the behaviour of many of the main convolutional neural networks in the literature when early exits are added to their structure and used to create an ensemble. To do this, a new early exits weighted ensemble technique called *Anticipate, Ensemble and Prune* (AEP) is proposed and thoroughly evaluated.

The aim of this study was to understand whether the early-exit ensemble approach is a valid one and, if so, to discover which learning conditions could favour improvements or lead to a degradation in image classification performance compared to that of the baseline single-exit version of the same networks. These studied conditions include aggregation strategies for both the loss function and inference, the effect of working with smaller or larger images, the influence of network and dataset structures, and how training the networks from scratch compares to fine-tuning them.

Compared to most of the work in the literature, such as the articles discussed in Section 3.1, the aim of AEP is not to reduce latency as much as possible without sacrificing accuracy, but rather to maximise the accuracy performance of the given model. Finally, by employing a pruning step, AEP is able to reduce the number of parameters, operations and network latency, while further increasing accuracy by extracting an optimal sub-ensemble network.

This study was submitted for publication and it was accepted. The associated paper is titled *"Anticipate, Ensemble and Prune: Improving Convolutional Neural Networks via Aggregated Early Exits"* [13]. The AEP technique has been extensively tested in computer vision to solve image classification tasks, but can easily be extended to other types of data and purposes. The code for this project is available on GitHub.

4.1. Proposed Method

The training of an AEP model proceeds as it follows: for each batch of B images, given C target classes, the algorithm lets the images flow through the network such that every classifier outputs a tensor of B elements, each representing one image and containing C values, one for each class c. In this setting, the network loss is computed as the weighted sum of the *Categorical Cross-Entropy Loss* L_i (Equation 4.1) of each exit $i \in [1, N]$, meaning a joint training.

$$L_i = L_{cce,i} = -\frac{1}{B}\sum_{b=1}^{B}\sum_{c=1}^{C}(p_{b,c}\log(y_{b,c})) \tag{4.1}$$

Unlike other approaches to early exits, in AEP the last exit is not treated differently from the intermediate ones, since its contribution to the final loss is weighted following the same weight assignment strategy as the others. Equation 4.2 represents the overall network loss, where α_i is the weight assigned to the loss associated with exit i.

$$L = \sum_{i=1}^{N}\alpha_i \cdot L_i \tag{4.2}$$

The prediction obtained after the ensemble, from now on referred to as $\hat{y}$, is computed as a weighted sum of the outputs of each i-th output, namely O_i. The calculation of each classification metric is performed after applying the *Argmax* operator to the vector obtained through the ensemble. Equation 4.3 thus represents the prediction step, where β_i is the weight assigned to the output associated with exit i. Since the outputs produced by the exits are not consumed in the process, their individual accuracies can be tracked throughout the training process.

$$\hat{y} = \text{argmax}(\sum_{i=1}^{N}\beta_i \cdot O_i) \tag{4.3}$$

The strategy for selecting weights $[\alpha_1, ..., \alpha_N]$ and $[\beta_1, ..., \beta_N]$, which are in principle different, is a crucial step in AEP and is discussed in subsection 4.2.3.

After completing the multi-output network training, a pruning step is applied as a post-processing technique to optimise its performance. In order to apply the proposed pruning operation, the resulting network is loaded and validated in all possible combinations of exit activation states, using the same exits weights used during the training phase. Then,

the best sub-network is extracted and evaluated on the test set. This selection process can aid in further improving accuracy in comparison to both the full ensemble and the single-exit network, while also reducing the number of parameters, operations, and latency as entire sections of the original network can be removed.

The proposed methodology is schematized in Figure 4.1 and the steps to perform training and evaluation of the early-exit networks are described in Algorithm 4.1.

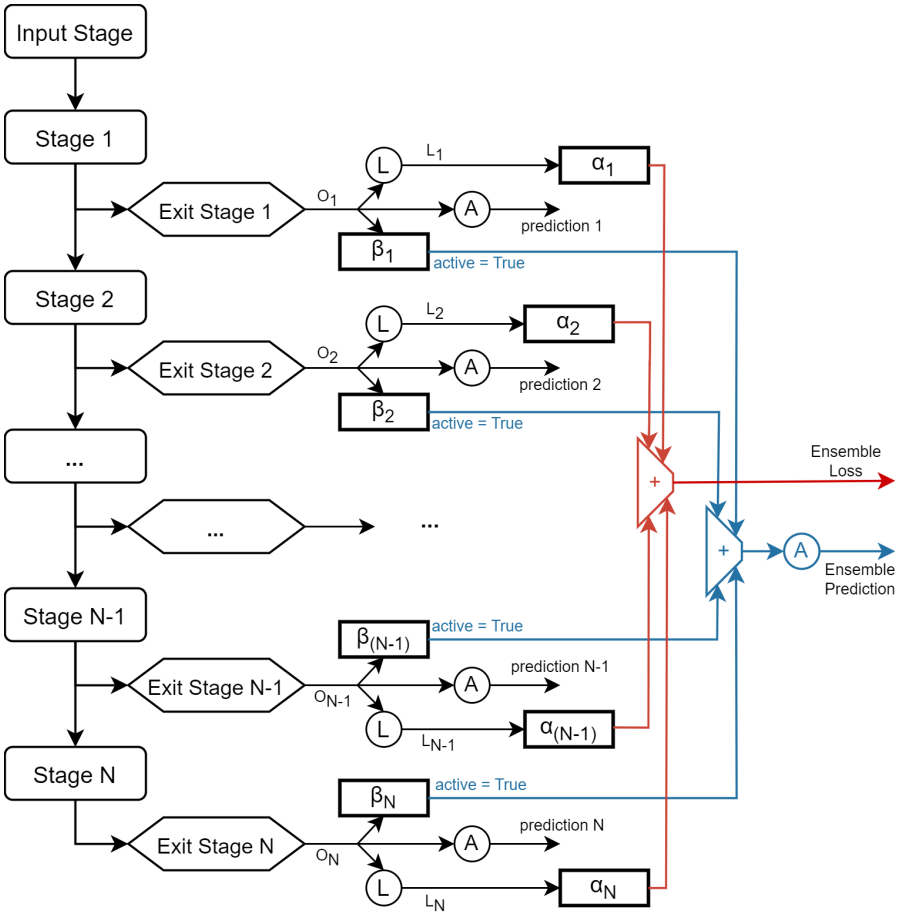


Figure 4.1: Outline of the AEP technique. O_i represents the output of the i-th exit stage with linear activation. Ⓛ represents the loss function, i.e., the Categorical Cross-Entropy loss, while Ⓐ represents the Argmax function used to obtain the class prediction. α_i and β_i represent the weights assigned to the loss and the output of exit stage i, respectively. The activation of the outputs is relative to the pruning step with which the proposed method terminates.

Algorithm 4.1 AEP, Train One Epoch

```
for batch, (images, labels) in train_loader do
    branches_outputs = Network(images)
    w_branches_losses = [ ]
    net_loss = 0
    for i = 0 to n_branches − 1 do
        branch_loss = LossCriterion(branches_outputs[i], labels)
        w_branches_losses[i] = branch_loss · losses_weights[i]
        net_loss = net_loss + w_branches_losses[i]
    end for
    w_branches_outputs = [ ]
    for i = 0 to n_branches − 1 do
        w_branches_outputs[i] = branches_outputs[i] · outputs_weights[i]
    end for
    net_output = stack(sum(w_branches_outputs))
    net_loss.backpropagate()
    Network.optimizerStep()
    net_loss_meter.update(net_loss)
    net_acc_meter.update(accuracy(net_output, labels))
    branches_losses_meter.update(w_branches_losses)
    branches_acc_meter.update(accuracies(branches_outputs, labels))
end for
Return(
net_loss_meter.getAvg(),
branches_losses_meter.getAvgs(),
net_acc_meter.getAcc(1,5),
branches_acc_meter.getAccs(1,5))
```

4.2. Experiments

To evaluate the effectiveness of the AEP technique, experiments were performed on several well-known CNNs, using many of the major image classification benchmarks. In addition, all experiments were performed with two different input image sizes and, for early-exit networks, with four different exits weighting schemes. Finally, each network was both trained from scratch and fine-tuned using the weights learned on ImageNet.

For each experiment, a set of performance metrics was collected, including Top1 and Top5

accuracies, number of parameters, number of MACs (Multiply-ACcumulate operations), latency, and training time. Additionally, for multi-output networks, the Top1 and Top5 accuracies of each exit were also recorded.

4.2.1. Networks

The AEP technique was tested on 5 well-known network architectures, thus encompassing many of the most relevant architectural trends and patterns of convolutional neural networks (CNNs). These 5 networks were chosen with the goal of spanning different network sizes, design patterns and internal components. Below the list of the networks used in the experiments:

- **ResNet50** [5]: In terms of size, it is considered an average size ResNet architecture. ResNets are characterized by the use of skip/residual connections between layers, which should help in improving gradient flow through the network.
- **VGG16** [3]: It is a purely sequential CNN, and while smaller than other VGG models, it is still by far the most demanding in terms of operations required.
- **DenseNet169** [6]: It is considered as an average size DenseNet architecture and is characterised by a dense blocks structure, where the features obtained after one layer are not only passed to the following ones but also concatenated to their outputs, so that the information in a block is better preserved and exploited.
- **MobileNetV3Small** [7]: Designed to work under mobile settings, it was a good candidate to see what would happen with tiny architectures. This network is characterised by the use of a highly memory efficient block called the Residual Inverted Bottleneck.
- **EfficientNetB5** [8]: Part of the EfficientNet family, currently one of the best performing architecture families, this model was chosen because with more than 500 layers it allows very deep architectures to be studied.

For each network, the series of repeating blocks were identified, extracted and transformed into the stages for the backbone of a new network. Given the list of backbone stages, it was then possible to determine how many exit stages were needed and where to attach them. To create the exit stages, the final sections of the original networks were replicated and the number of features proportionally rescaled based on where the newly created exit stage would be attached to the new network. The only exit stage that was modified from its original structure was that of the VGG16 model, which is extremely large and over-parameterised by default. In this case, a simpler Global Average Pooling layer was used,

followed by a dense layer with a number of units equal to the number of possible classes. Each network was also enriched with a function to extract a subnetwork by specifying which exits to keep, which is functionally necessary to perform the pruning step.

The experiments conducted considered each network both with a number of exits equal to 4, for comparison purposes, and with a number of exits equal to the number of stages in the original model. In particular, while the experiments on ResNet50 and DenseNet169 did not need to be repeated, since 4 was already the correct number of exits to match the number of stages, VGG16 and MobileNetV3Small also required an additional set of experiments with 5 exits, while EfficientNetB5 required tests with 7 exits. We refer to these 3 network variants by adding the keyword "full" to their names: **VGG16full**, **MobileNetv3Smallfull** and **EfficientNetB5full**, bringing the total number of networks tested to 8.

4.2.2. Datasets

The experimental evaluation of the AEP algorithm was carried out using six different image classification datasets. These datasets were chosen to cover a wide range of characteristics in order to assess the generalisability and robustness of the algorithm. They span a range of class sizes from tens to hundreds, can be composed of either monochrome or RGB images, some of them show class imbalances and, finally, the amount of available training data varies significantly between them. The properties of these datasets are summarised in Table 4.1.

Table 4.1: Datasets used to conduct the experiments and their characteristics.

Dataset	Classes	Balanced	Channels	Train	Validation	Test
CIFAR-10 [68]	10	yes	RGB	45 000	5 000	10 000
CIFAR-100 [68]	100	yes	RGB	45 000	5 000	10 000
EuroSAT [81]	10	no	RGB	17 500	4 000	5 500
FashionMNIST [82]	10	yes	Grayscale	51 000	9 000	10 000
GTSRB [83]	43	no	RGB	33 209	6 000	12 630
Tiny ImageNet [12]	200	yes	RGB	90 000	10 000	10 000

4.2.3. Exits Weights

In this AEP implementation, the weights associated with the losses and outputs of the exits were chosen to be linearly increasing/decreasing with respect to the order of the exits. Furthermore, the values of these weights have been chosen to be strictly positive,

with a sum equal to one. In this way, it is possible to maintain the desired properties by having adaptive scales of weights regardless of the number of exits within the network.

In the initial set of experiments, there was no difference between the weight assigned to an exit to compute its loss and that used to compute its output. Specifically, the weights were assigned according to an ever-increasing or ever-decreasing pattern along the structure of the network. To differentiate between these two cases, the experiments in which the weights were assigned in a descending order for both the losses and the outputs are referred to as "EEdesc", while those in which the weights were assigned in an ascending order for both the losses and the outputs are referred to as "EEasc".

Ablation studies demonstrated that, while the use of decreasing weights generally leads to better exits when considered individually, particularly for earlier exits, networks with ascending weights perform better in terms of ensemble prediction. To exploit this, the two sets of weights have been decoupled, and "EEmix" networks are proposed, in which the exits losses weights are assigned in descending order while the outputs weights are assigned in ascending order.

In addition, a uniform weighting mode was tested, in which all exits are given the same weight. These experiments are referred to as "EEunif" in the experimental section. A summary of the different weight modes can be found in Table 4.2.

Table 4.2: Exits weights assignment strategies

Modes Names	Losses Weights	Outputs Weights
DESC	decreasing	decreasing
ASC	increasing	increasing
MIX	decreasing	increasing
UNIF	uniform	uniform

4.2.4. Learning Scenarios

To gain a broader understanding of the behaviour of the early-exit ensembles compared to their single-exit counterparts in as wide a range of situations as possible, each experiment was run under 4 different learning scenarios. The first pair of settings compared the baselines and the ensemble networks when both were trained from scratch versus when they were being fine-tuned from standard (ImageNet) weights. The second pair of settings compared them in scenarios where more or less information was present in the images, for this reason experiments were run with both the 224x224 and the less informative 64x64 input image sizes for all datasets.

4.2.5. Hyperparameters Settings

The goal of this research was to evaluate the performance of single-exit networks compared to early-exit ensembles rather than to match or beat state of the art results. To achieve this, a simple and standardised training approach was employed: 100 training epochs, a batch size of 64 and a learning rate of 10^{-4} with the Adam optimizer [84]. An early stopping technique was used, the patience value was set to 12 epochs with respect to the validation loss. No data augmentation or learning rate schedules were utilized. The only pre-processing applied to the images was resizing to 224 or 64 via bicubic interpolation and normalization. A small learning rate was chosen to ensure that the same hyperparameters could be used for both training from scratch and in fine-tuning, thus ensuring comparable results. The models were implemented using PyTorch 1.12.1 and ran on an NVIDIA Quadro RTX 6000 GPU.

4.3. Results and Discussion

The results of experiments on the AEP technique are reported in this section. Results are presented in terms of averaged percentage changes in order to show increases and decreases in performance relative to the the original single-exit version of the networks. For reference, the percentage change (increase or decrease) of the value X_2 with respect to the value X_1 is calculated as:

$$\frac{X_2 - X_1}{|X_1|} \cdot 100 \tag{4.4}$$

The symbol '*' will indicate experiments whose early exit networks were pruned, in accordance with the complete AEP procedure. In contrast, the absence of the symbol '*' represents full-ensemble networks. The abbreviations "SE" and "EE" stand for "Single-Exit" and "Early-Exit" networks respectively.

4.3.1. Classification Accuracy

Table 4.3 and Table 4.4 show, for each learning scenario, the results of the experiments in terms of the percentage change in Top1 accuracy compared to the performances of single-exit networks. Two tables are shown for each learning scenario, the first grouping the results by neural network, the second by dataset. Table 4.5 provides a more condensed view of the results, showing only the average performance gains and losses for each scenario.

Table 4.3: PART 1: Training.Early-Exit vs Single-Exit Top1 accuracy comparisons as average percentage change. Results are first averaged across architectures and then across datasets for each scenario. The best results are highlighted in bold.

TRAIN-224

Network	EEdesc	EEasc	EEmix	EEunif	EEdesc*	EEasc*	EEmix*	EEunif*
ResNet50	7,962	7,948	9,901	8,510	8,617	7,997	**10,088**	8,831
VGG16	0,849	**4,653**	3,670	4,360	2,596	4,651	3,688	4,407
VGG16full	0,922	4,063	3,517	4,035	3,012	4,217	3,524	**4,859**
DenseNet169	-1,335	0,804	-0,066	0,395	-0,410	0,803	0,046	**1,046**
MobileNetV3Small	12,046	9,672	11,644	9,882	**12,289**	9,588	11,679	10,055
MobileNetV3Smallfull	11,361	9,803	12,727	10,249	**13,661**	9,817	12,650	10,591
EfficientNetB5	1,630	2,100	2,108	**2,362**	1,885	2,113	2,239	2,086
EfficientNetB5full	3,710	3,046	4,443	4,320	4,065	2,958	**4,480**	4,359
Average	4,643	5,261	5,993	5,514	5,714	5,268	**6,049**	5,779

Dataset	EEdesc	EEasc	EEmix	EEunif	EEdesc*	EEasc*	EEmix*	EEunif*
CIFAR-10	3,795	3,758	5,232	4,182	4,261	3,822	**5,299**	4,365
CIFAR-100	13,972	11,019	16,117	12,339	15,888	11,020	**16,229**	12,680
EuroSAT	3,027	2,586	2,996	2,956	3,046	2,622	**3,103**	2,993
FashionMNIST	0,628	0,912	0,942	1,122	0,995	0,926	0,946	**1,208**
GTSRB	-5,024	1,719	-2,914	-0,746	-3,846	**1,873**	-3,149	-0,013
Tiny ImageNet	11,462	11,573	13,586	13,232	**13,942**	11,346	13,867	13,442
Average	4,643	5,261	5,993	5,514	5,714	5,268	**6,049**	5,779

TRAIN-64

Network	EEdesc	EEasc	EEmix	EEunif	EEdesc*	EEasc*	EEmix*	EEunif*
ResNet50	20,039	10,444	16,952	14,368	**20,642**	10,655	17,790	14,402
VGG16	6,395	4,273	5,078	5,704	**6,942**	4,252	5,376	5,607
VGG16full	5,288	4,359	5,025	5,743	**6,277**	4,452	5,174	5,909
DenseNet169	9,852	6,805	8,222	9,117	**10,268**	6,806	8,751	9,221
MobileNetV3Small	26,040	16,323	23,086	20,008	**26,129**	16,377	23,848	19,956
MobileNetV3Smallfull	26,940	17,934	24,223	21,844	**28,230**	17,750	25,559	21,572
EfficientNetB5	7,367	2,250	6,616	4,983	7,477	2,378	**7,482**	5,861
EfficientNetB5full	13,607	4,323	11,213	9,007	**14,026**	5,142	12,500	9,879
Average	14,441	8,339	12,552	11,347	**14,999**	8,477	13,310	11,551

Dataset	EEdesc	EEasc	EEmix	EEunif	EEdesc*	EEasc*	EEmix*	EEunif*
CIFAR-10	10,510	5,643	9,763	8,233	**10,595**	5,811	10,449	8,333
Cifat100	24,093	12,637	21,284	18,241	**25,590**	12,778	22,409	18,772
EuroSAT	9,491	4,462	8,317	7,316	**9,774**	5,051	9,189	7,718
FashionMNIST	1,793	1,071	1,727	1,772	**1,804**	1,062	1,736	1,734
GTSRB	5,309	7,208	5,881	6,853	5,196	**7,236**	5,961	6,860
Tiny ImageNet	35,450	19,012	28,340	25,665	**37,035**	18,921	30,115	25,888
Average	14,441	8,339	12,552	11,347	**14,999**	8,477	13,310	11,551

Table 4.4: PART 2: Fine-Tuning. Early-Exit vs Single-Exit Top1 accuracy comparisons as average percentage change. Results are first averaged across architectures and then across datasets for each scenario. The best results are highlighted in bold.

FINETUNE-224

Network	EEdesc	EEasc	EEmix	EEunif	EEdesc*	EEasc*	EEmix*	EEunif*
ResNet50	-4,199	0,288	-2,126	-0,889	-3,150	**0,352**	-2,218	-0,490
VGG16	-0,417	1,756	0,945	1,845	0,669	1,709	0,943	**1,886**
VGG16full	-1,635	1,895	0,987	1,550	0,255	1,977	0,932	**2,114**
DenseNet169	-6,263	-1,196	-4,153	-2,520	-4,870	**-1,004**	-3,961	-2,018
MobileNetV3Small	-3,813	0,810	-1,274	0,119	-1,691	**0,833**	-1,255	0,593
MobileNetV3Smallfull	-4,619	0,536	-1,274	-0,087	-0,993	**0,784**	-1,164	0,542
EfficientNetB5	-4,883	-0,343	-1,431	-1,186	-1,704	**-0,228**	-1,107	-0,283
EfficientNetB5full	-4,920	-0,604	-1,956	-1,342	-2,535	**-0,332**	-1,809	-0,649
Average	-3,844	0,393	-1,285	-0,314	-1,752	**0,511**	-1,205	0,212

Dataset	EEdesc	EEasc	EEmix	EEunif	EEdesc*	EEasc*	EEmix*	EEunif*
CIFAR-10	-3,247	0,070	-1,000	-0,690	-1,393	**0,260**	-0,892	-0,229
CIFAR-100	-6,428	0,973	-1,718	0,268	-2,540	**1,015**	-1,593	0,867
EuroSAT	0,492	**0,812**	0,556	0,698	0,627	0,805	0,556	0,701
FashionMNIST	-0,485	**0,040**	-0,065	-0,030	-0,096	0,039	-0,090	0,030
GTSRB	-2,653	-0,207	-0,940	-0,787	-1,529	**-0,201**	-0,919	-0,354
Tiny ImageNet	-10,742	0,669	-4,546	-1,340	-5,583	**1,150**	-4,292	0,257
Average	-3,844	0,393	-1,285	-0,314	-1,752	**0,511**	-1,205	0,212

FINETUNE-64

Network	EEdesc	EEasc	EEmix	EEunif	EEdesc*	EEasc*	EEmix*	EEunif*
ResNet50	1,048	**4,252**	0,861	3,687	1,233	3,895	0,697	3,660
VGG16	2,516	**3,500**	2,148	2,555	2,811	3,403	2,151	2,584
VGG16full	1,805	3,714	2,108	3,454	2,738	**3,762**	2,110	3,508
DenseNet169	-0,399	2,790	0,113	1,985	-0,156	**2,979**	0,190	2,211
MobileNetV3Small	6,060	6,084	5,949	**6,646**	6,417	5,850	6,009	6,585
MobileNetV3Smallfull	5,097	6,119	5,773	6,233	6,418	6,102	5,719	**6,500**
EfficientNetB5	5,181	5,768	5,137	6,040	5,243	5,771	5,126	**6,045**
EfficientNetB5full	6,189	6,075	6,120	6,771	6,291	6,123	6,397	**6,792**
Average	3,437	**4,788**	3,526	4,671	3,874	4,736	3,550	4,736

Dataset	EEdesc	EEasc	EEmix	EEunif	EEdesc*	EEasc*	EEmix*	EEunif*
CIFAR-10	2,722	3,401	2,983	3,737	2,966	3,415	3,063	**3,768**
CIFAR-100	5,402	7,594	5,835	7,876	6,123	7,563	5,891	**7,954**
EuroSAT	3,480	3,186	3,298	3,395	**3,512**	3,230	3,289	3,494
FashionMNIST	0,835	0,916	1,043	1,093	0,926	0,907	1,093	**1,138**
GTSRB	4,433	**5,640**	4,571	5,136	4,712	5,355	4,417	5,070
Tiny ImageNet	3,750	**7,989**	3,426	6,791	5,007	7,943	3,547	6,990
Average	3,437	**4,788**	3,526	4,671	3,874	4,736	3,550	4,736

Table 4.5: Early-Exit vs Single-Exit Top1 accuracy comparisons as average percentage change. Values are averaged over architectures and datasets. The best results are highlighted in bold.

	EEdesc	**EEasc**	**EEmix**	**EEunif**	**EEdesc***	**EEasc***	**EEmix***	**EEunif***
TR-224	4,643	5,261	5,993	5,514	5,714	5,268	**6,049**	5,779
TR-64	14,441	8,339	12,552	11,347	**14,999**	8,477	13,310	11,551
FT-224	-3,844	0,393	-1,285	-0,314	-1,752	**0,511**	-1,205	0,212
FT-64	3,437	**4,788**	3,526	4,671	3,874	4,736	3,550	4,736

The experiments where the networks were trained from scratch with 224x224 images (TRAIN-224) show impressive average improvements regardless of the weighting strategy used, in fact a configuration that improves accuracy can be identified for each neural network and dataset. In this scenario, AEP provides the most significant benefits when the datasets used have a large number of classes, as is the case with CIFAR-100 and Tiny ImageNet. On these two datasets, the use of AEP leads to average accuracy improvements of 16.23% and 13.94% respectively. In the first case the best configuration is EEmix*, in the second case EEdesc*. As far as networks are concerned, ResNet50 and the two versions of MobileNetV3Small show the most significant improvements over the baseline.

The results of the experiments where the networks were trained from scratch with smaller 64x64 images (TRAIN-64) confirm and emphasise the observations made for the TRAIN-224 scenario. As a result of the reduced input information, the classification task faced by the networks is more complex. This is the context in which AEP is most beneficial, improving model quality regardless of architecture, dataset and weight configuration, sometimes making the difference between a poorly performing model and a winning one. In this scenario, the EEdesc* technique appears to be the clear favourite, improving accuracy by an average of 15%.

Moving on to the fine-tuning scenario, and in particular when 224x224 images (FINETUNE-224) are used, a different behaviour can be observed. In this case, the addition of early exits has, on average, a detrimental effect. This is reasonable in view of the fact that the initial models, pretrained on ImageNet images of similar resolution, have already learned a set of weights that favour the production of features that allow accurate predictions from the single output at the bottom of the network. The use of ascending weights for the losses and in the inference phase is the optimal choice in this scenario, as confirmed by the results obtained in the EEasc and EEasc* experiments and shown in Table 4.4. The ascending weight assignment works because it exploits the behaviour of the original single exit model rather than overriding it.

Finally, a different behaviour can be observed when looking at experiments where the networks were fine-tuned using 64x64 images (FINETUNE-64). In this case, the change in spatial domain led to an overall improvement in the accuracy of the AEP-based models compared to their single-exit counterparts. This is most likely due to the higher complexity of the problem, which is inversely proportional to the size of the input. This allows better exploitation of classifiers trained on lower-level and thus less elaborate features, as it is not the case with large images requiring more complex transformations. Nevertheless, using ascending weights proves to be the winning move here as well, able to improve the accuracy of the models by an average of 4.79%.

In general, early exit ensemble networks trained with the AEP method improve much more over the baseline when trained from scratch than when fine-tuned. This is because training from scratch via joint optimisation allows the network to improve the features of each layer towards a good classification result, producing better features in the layers closer to the network input. The presence of good classification features in early layers also facilitates the learning of better classification features by later layers. On the other hand, fine-tuning starts from the weights obtained after the training of corresponding single-exit networks, meaning that the features of the early layers are not good classification features in themselves, but they contribute to obtaining good classification features in the final layers. In this case, the introduction of early exits into the networks may lead to contradictory behaviour, as the optimisation tries to override the initial weights to solve a task which is already optimally solved at a later stage. By assigning higher weights to exits closer to the input, we end up assigning high classification importance to layers that should not be able to do so, which can lead to worse performance than the baseline.

For each learning scenario, the application of the AEP pruning step results in networks that exhibit the same behaviour as their full-ensemble counterparts, but with slightly better accuracy on average. The average percentage changes in top1 accuracy of the pruned networks with respect to the baseline networks are collected in the "*" columns of Table 4.3 and Table 4.4.

Table 4.6 instead compares the accuracy performance of the pruned versions of the early exits ensemble networks with their full-ensemble counterparts. Among the different types of networks, EEdesc networks are the ones that benefit the most from the pruning step, especially in fine-tuning, which also happens to be the learning context in which they perform the worst.

Table 4.6: Pruned Early-Exit networks (*) vs full-ensemble Early-Exit networks Top1 accuracy comparisons as average percentage change. Values are averaged over architectures and datasets. The best results are highlighted in bold.

	EEdesc*	EEasc*	EEmix*	EEunif*
TR-224	**1,006**	0,009	0,050	0,260
TR-64	0,435	0,142	**0,625**	0,193
FT-224	**2,280**	0,121	0,087	0,539
FT-64	**0,421**	-0,048	0,021	0,063

4.3.2. Exits Analysis

Another very interesting aspect to study is the quality of the classifiers obtained from the joint training. For each of the EEmodes (DESC, ASC, MIX, UNIF), for networks with exactly 4 exits, Table 4.7 shows the average percentage change in accuracy of each exit compared to the average performance achieved by single-exit networks.

Table 4.7: Early-Exit branches vs Single-Exit output Top1 accuracy comparisons as average percentage change (for 4-exits networks). Values are averaged over architectures and datasets. The best results are highlighted in bold.

TR-224	exit1	exit2	exit3	exit4
EEdesc	-30,155	-4,193	2,508	**2,793**
EEasc	-42,465	-12,538	1,171	**3,041**
EEmix	-31,434	-5,444	2,884	**3,115**
EEunif	-35,224	-8,517	1,942	**2,970**

TR-64	exit1	exit2	exit3	exit4
EEdesc	-10,432	**9,515**	9,442	7,739
EEasc	-25,613	-1,243	**4,999**	4,373
EEmix	-11,434	8,350	**9,957**	8,060
EEunif	-17,433	4,527	**7,027**	5,692

FT-224	exit1	exit2	exit3	exit4
EEdesc	-37,867	-13,969	-5,713	**-3,027**
EEasc	-48,667	-19,457	-5,634	**-0,528**
EEmix	-39,664	-16,120	-5,777	**-2,574**
EEunif	-41,946	-16,633	-5,089	**-1,066**

FT-64	exit1	exit2	exit3	exit4
EEdesc	-20,932	-2,276	**1,181**	0,649
EEasc	-32,504	-7,237	1,902	**2,773**
EEmix	-22,601	-3,186	**1,173**	0,874
EEunif	-25,106	-3,932	1,891	**1,915**

Firstly, it should be noted that networks trained with decreasing weights for losses (EEdesc and EEmix) tend to produce better performances for exits closer to input than those obtained with other weighting strategies, even in a fine-tuning scenario, although this strategy has previously been shown not to be beneficial in terms of overall ensemble performance. Instead, when training from scratch, the decreasing weights strategy results in better performance for all exits.

The second notable result is that the AEP joint training, regardless of the EEmode, can produce early exits with an average performance superior to that of the single-exit network. This is true for all learning scenarios except FINETUNE-224. Looking at the

TRAIN-224 and FINETUNE-64 scenarios, it can be seen that both the third and fourth exit are advantageous alternatives to the traditional single-exit network, regardless of the AEP configuration. For the TRAIN-64 scenario, the results are even more promising, with all weighting strategies except ASC resulting in better average accuracy than single-exit networks from the second exit onwards.

These results clearly show that sometimes human intuition, such as increasing the complexity of a model, does not necessarily lead to better performance. On the contrary, better results can be achieved by optimising the learning process and by taking advantage of different levels of abstraction of the input data, as is done by AEP.

4.3.3. Parameters, Operations, Latency and Training Time

Adding branches to a network inevitably increases the number of parameters, operations required and latency. The magnitude of the increase is directly related to the complexity of such branches. The average percentage changes in number of parameters, operations and latency between the AEP early-exit ensemble networks and the baseline single-exit networks are shown in Table 4.8, Table 4.9 and Table 4.10 respectively.

Full-ensemble networks metrics results are grouped together because changing the way the weights are assigned to the exits doesn't affect the structure of the networks. These structural equivalences do not exist for pruned networks, so the results were only averaged by the exits weight assignment strategy.

The addition of early exits has a minimal impact on the increase in the number of parameters, averaging only a 2.88% increase over single exit architectures, as can be seen from the first column of Table 4.8. Key here is the impact of the pruning step, which provides significant benefits almost across the board. Particularly relevant is the case of EEmix* under the TRAIN-64 scenario, where AEP achieves an average parameter reduction of 41.02% and simultaneously an average accuracy gain of 13.31%, as described in Table 4.5.

For what concerns the number of MACS, the presence of all branches leads to an average increase of about 5%, as shown in Table 4.9. Again, the pruning step has a beneficial effect, producing networks that in most cases have fewer MACs on average than their single-exit counterparts. The reduction in the number of MACs is as high as 17.95% in the EEmix* case under the TRAIN-64 scenario.

In terms of latency, the average increase in time due to the presence of additional branches is quite small, between 6% and 7%, as shown in Table 4.10. For this metric, the benefits of pruning are still there, but not as pronounced as in the previous two cases. Once

again, the best scenario is that of EEmix* in the TRAIN-64 scenario, where the average time saving is 16.325%. As for training time per epoch, the observed difference between early-exit and single-exit networks is approximately a 5% increase.

Table 4.8: Early-Exit vs Single-Exit Parameters comparisons as average percentage change. Values are averaged over architectures and datasets. The best results are highlighted in bold.

	EE	**EEdesc***	**EEasc***	**EEmix***	**EEunif***
TR-224	2,876	-0,566	-10,422	**-16,508**	-1,773
TR-64	2,876	-12,731	-22,549	**-41,018**	-25,300
FT-224	2,876	**-5,791**	1,773	-1,902	-0,720
FT-64	2,876	-4,260	-14,305	**-19,351**	-11,292

Table 4.9: Early-Exit vs Single-Exit MACs comparisons as average percentage change. Values are averaged over architectures and datasets. The best results are highlighted in bold.

	EE	**EEdesc***	**EEasc***	**EEmix***	**EEunif***
TR-224	5,444	1,816	-1,908	**-5,438**	2,118
TR-64	5,260	-3,302	-8,549	**-17,952**	-9,078
FT-224	5,444	**-1,475**	2,830	2,514	1,115
FT-64	5,260	0,059	-1,954	**-4,085**	-1,050

Table 4.10: Early-Exit vs Single-Exit Latency comparison as average percentage change. Values are averaged over architectures and datasets. The best results are highlighted in bold.

	EE	**EEdesc***	**EEasc***	**EEmix***	**EEunif***
TR-224	5,795	3,782	0,387	**-2,729**	3,220
TR-64	7,771	-0,776	-6,061	**-16,325**	-7,482
FT-224	6,645	**0,407**	4,959	4,177	3,143
FT-64	7,877	2,264	-1,511	**-3,111**	0,122

In summary, these results show that the AEP pruning step not only helps to increase accuracy, but also to reduce the values of other important metrics, as shown in Figure 4.2 Among the configurations, the MIX weight assignment strategy seems to be able to produce networks that can be pruned in a more aggressively than the others without compromising accuracy. Fine-tuning scenarios tend to benefit less from the pruning step; the addition of early exits is only partially beneficial in these cases, as opposed to scenarios characterised by training from scratch.

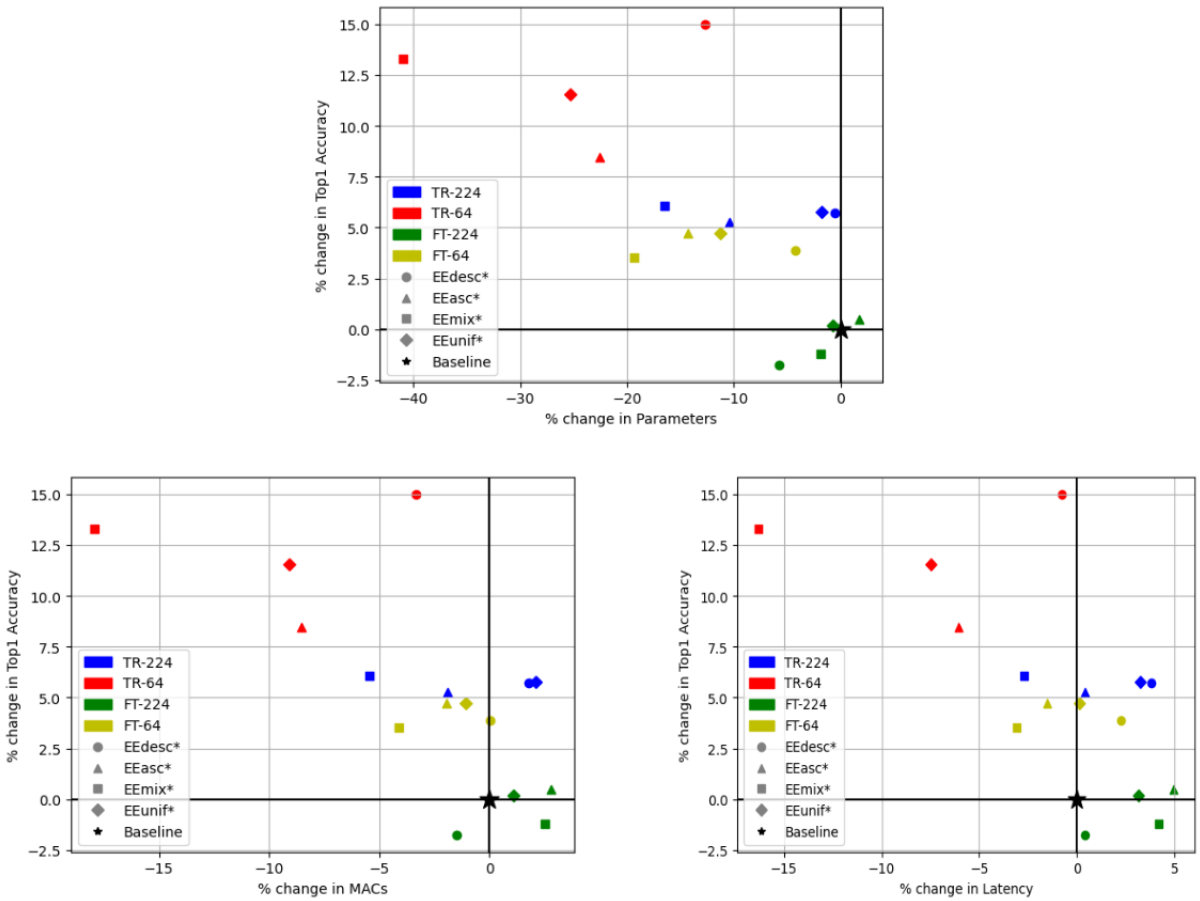


Figure 4.2: The plots show the results obtained using the complete AEP technique compared to those obtained using single-exit networks (baseline). The average variation in accuracy, on the y-axis, is represented as a function of the average variation in another network metric, shown on the x-axis. Each data point identifies a learning scenario/weighting strategy pair, the former represented by the colour of the data point, the latter by its shape. The closer a data point is to the top left-hand corner of the plot, the better the trade-off it represents.

4.3.4. Inter-Learning-Scenario Comparisons

The performance variation between pairs of learning scenarios are summarised in Table 4.11. The first pair of rows in the table compares fine-tuning and training from scratch given input images of the same size. The second pair of rows instead compares performances when the it is the size of the images to change.

While it's no surprise that neural networks perform better when fine-tuned than when trained from scratch, the results also show that the size of the images has a visible effect on performance for both learning modalities, despite the fact that the additional information

present in larger images was artificially created by interpolation. In all cases, single-exit networks show the largest inter-scenario improvements, followed in order by EEasc, EEunif, EEmix and finally EEdesc trained networks.

Table 4.11: Inter-learning-scenarios Top1 accuracy comparisons as average percentage change. Values are averaged over architectures and datasets. The best results are highlighted in bold.

	SE	**EEdesc**	**EEasc**	**EEmix**	**EEunif**	**EEdesc***	**EEasc***	**EEmix***	**EEunif***
FT-224 vs **TR-224**	**24,877**	12,847	18,129	14,841	16,728	14,522	18,317	14,891	17,162
FT-64 vs **TR-64**	**32,126**	18,241	27,183	20,344	23,217	18,216	26,937	19,551	23,011
TR-224 vs **TR-64**	**23,101**	12,334	19,534	15,688	16,425	12,938	19,352	14,960	16,443
FT-224 vs **FT-64**	**16,618**	7,471	11,340	10,632	10,566	9,647	11,556	10,717	11,156

4.4. Conclusion

The results presented in this chapter clearly show how the introduction of early exits into the structure of convolutional neural networks, and their use as an ensemble according to the AEP technique, can lead to significant improvements in image classification performance compared to the corresponding single-exit versions of the networks. Given the positive results obtained, the introduction and use of early exits in OFA and NAT networks will be investigated in the following chapters. The hope is that the presence of early exits will help these NAS techniques to find better performing networks with lower parametric complexity, number of operations and inference time.

Although the results obtained in this chapter are remarkable, the AEP technique could still be improved. In a follow-up study, AEP will incorporate an automatic optimisation process to find the best loss weights, as well as a search process to find the best output weights. In addition, the pruning step could be extended to perform multi-objective selection. These modifications are likely to lead to further improvements and possibly reveal undiscovered design patterns.

5 | Enhancing Once-For-All

This chapter presents an extension of the work proposed in Once-For-All (previously discussed in Section 3.2), investigating the effects of enriching the OFAMobileNetV3 supernet with new architectural components and designs such as early exits, skip connections and parallel blocks. These changes increase the number and possible types of subnets that can be expressed within the supernet. This extended version of OFA will be called OFAv2, and its goal is to allow the obtainment of better-performing supernets while preserving the energy and climate sustainability properties of the original algorithm.

The new architectural patterns have been applied and evaluated both singularly and in combination, to better understand how their presence and interactions influence the learning behaviour of the supernet.

In order to support the training of new subnets that can now be derived due to the presence of early exits and parallel blocks, the Progressive Shrinking algorithm has been extended with two new elastic steps, namely Elastic Exit and Elastic Level. Furthermore, new algorithms have been developed for the training and using of Knowledge Distillation in the presence of early exits, their inner workings are based on those used in AEP and discussed in Chapter 4. Finally, an improved way of obtaining the architecture to be used as teacher network throughout the training process is proposed and evaluated.

The work done on OFAv2 only concerns the training part of OFA, as for the search step, it will be replaced by an extended version of NAT the search process, which will be discussed in the next chapter.

This study was submitted for publication under the title *"Enhancing Once-For-All: A Study on Parallel Blocks, Skip Connections and Early Exits"* [14] and it was accepted. The code for these two projects will be made available on GitHub.

5.1. Method

This section describes in detail how OFAv2 has extended the OFAMobileNetV3 architecture by adding dense skip connections, early exits and parallel blocks. The training

technique called Extended Progressive Shrinking (EPS) is also introduced and its two new steps are described. Finally, the algorithms used to manage the training and knowledge distillation of early exit networks are examined in detail.

5.1.1. Architectural Modifications

The work focused on 3 main architectural changes with respect to OFAMobileNetV3, for a total of 7 new supernets extending the structure of the original one. In detail, the modifications introduced in this work are:

- **Dense Skip Connections (D)**: The supernet is extended with a dense scheme of skip connections between the blocks belonging to the same stage. Compared to the original OFAMobileNetV3 structure, where the output of each block is a residual obtained through the summation with the input, in this case the output of the previous blocks in the stage is also added. The additional connections perform two-block skips so as not to duplicate the skip connection already present in the IRB blocks. When blocks in the second part of a stage are deactivated due to Elastic Depth, all skip connections going beyond the cut point are automatically deactivated as well.
- **Early Exits (EE)**: The supernet is characterised by the presence of multiple intermediate outputs, also known as early exits. In particular, one exit is placed after each of the five internal stages, for a total of five exits. In contrast, the original network has only a single exit, placed after the last stage. The idea behind the use of early exits is that intermediate outputs could have comparable or even better performance than the final one, making it possible to consider the use of computationally lighter models [13]. The ability to exit the network early could also lead to significant savings in terms of parameters, flops and latency, as well as facilitating the search for subnets to deploy on very resource-constrained devices.
- **Parallel Blocks (P)**: The supernet is extended with two new blocks that can be executed in parallel with the IRBs/IBs. We refer to as *level* the set of blocks executed in parallel at a given depth of a stage. This modification increases the variety of architectures that are searchable by the algorithm, since subpaths consisting of one or more blocks can be selected at each level. The first additional block consists of a sequence composed of one pointwise convolutional layer, a batch normalisation layer, and an activation function corresponding to the one used in the IRB/IB at the same level. The goal of this block is to mix the features in the channel dimension. The second block, on the other hand, focuses on fast transformations

and is a much lighter alternative to the IRB block. This block consists only of a batch normalization layer, followed by an activation function corresponding to the one used in the IRB/IB, preceded by additional operators in case the output tensor needs to be reshaped to match those of the other blocks in the same level. Specifically, a max-pooling operation can be applied to reduce the spatial dimension of the feature maps, and, similarly, a 1x1 convolution can be used to increase the number of feature maps.

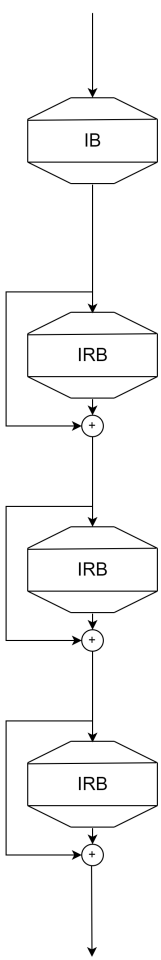
The original idea was for the second block to be a self-attention module, specifically the one described in [85]. The presence of a self-attention block would have resulted in a potentially more interesting structure, but initial experiments showed that its presence made the networks unable to learn under the OFA settings. This also resulted in an unacceptable increase in training time and a significant increase in the number of additional parameters and operations. For this reason, the final decision was made to use simple additional blocks, dictated by the desire to limit the inevitable increase in training time, number of parameters and number of MACs as much as possible.

At the implementation level, these new networks are built on a true staged structure, as opposed to the original one, which tracked the indexes of the layers to determine where each stage started and ended. This change made the networks much more flexible and manageable, allowing for reuse of code and for better abstraction in the algorithms to handle both the supernet and its subnets. The network management algorithms now work at two levels: at the network level, independent of the type of stage, and at the stage level, where the peculiarities of each modified stage are properly managed. For maximum flexibility, stages containing parallel blocks have been implemented so that individual blocks can be easily replaced with different ones.

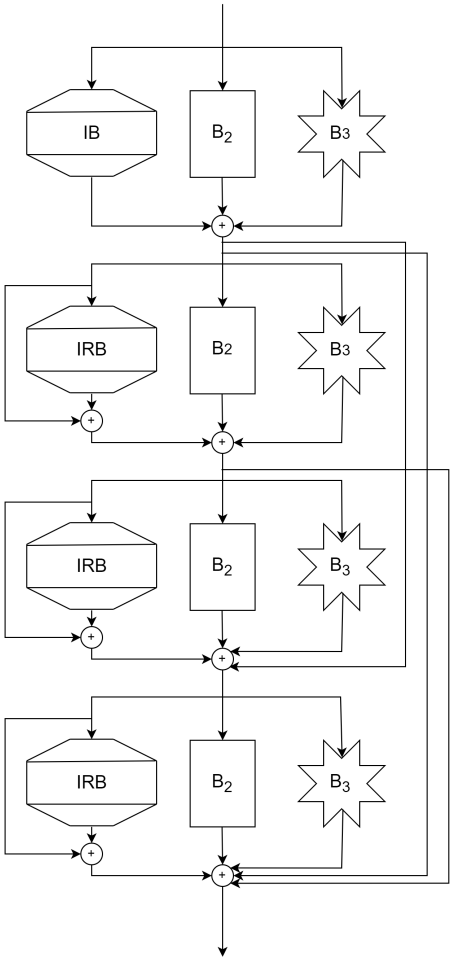
The names and characteristics of the networks tested are summarised in Table 5.1. We adopt the following naming convention for new configurations, consisting of two prefixes. The first prefix can be either SE or EE, representing single-exit and early-exit networks respectively. The second prefix refers to inter-stage modifications: D indicates the presence of dense skip connections, P that of parallel blocks, and B (base) the absence of these last two modifications. Figure 5.1 and Figure 5.2 graphically represent the architectural modifications made at the stage and network levels respectively.

Table 5.1: Network names and their corresponding structural changes

Network Name	Modular Stages	Early Exits	Dense Residuals	Parallel Blocks
OFAMobileNetV3	✗	✗	✗	✗
SE_B_OFAMobileNetV3	✓	✗	✗	✗
SE_D_OFAMobileNetV3	✓	✗	✓	✗
SE_P_OFAMobileNetV3	✓	✗	✗	✓
SE_DP_OFAMobileNetV3	✓	✗	✓	✓
EE_B_OFAMobileNetV3	✓	✓	✗	✗
EE_D_OFAMobileNetV3	✓	✓	✓	✗
EE_P_OFAMobileNetV3	✓	✓	✗	✓
EE_DP_OFAMobileNetV3	✓	✓	✓	✓

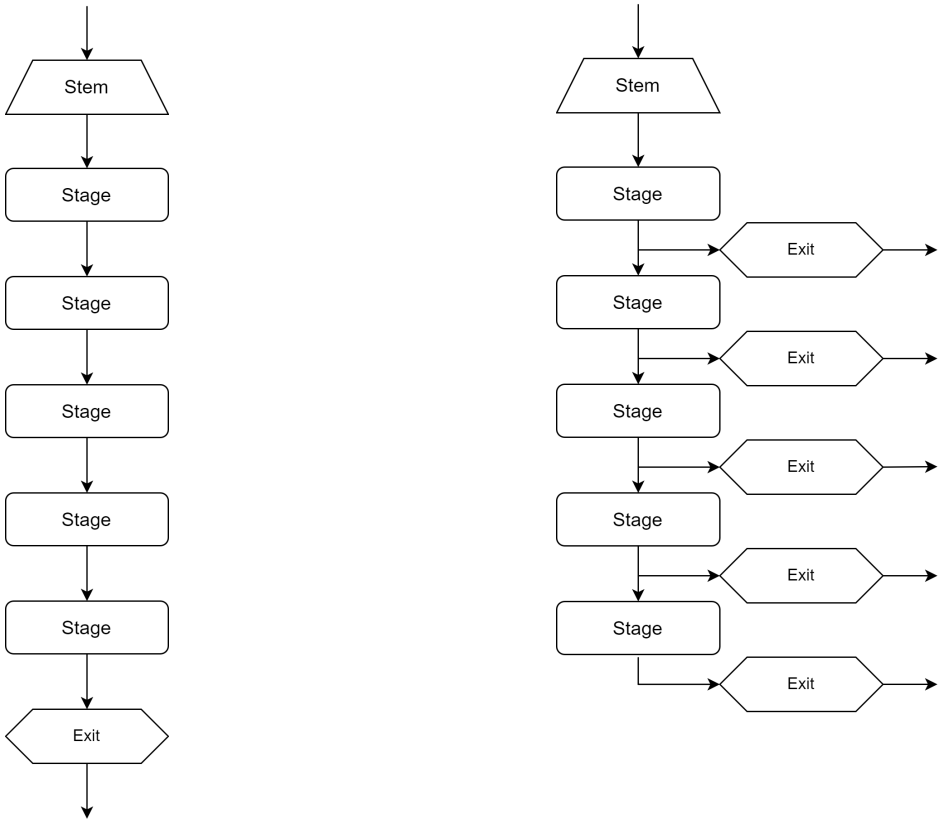


(a) Structure of a stage in the baseline network, without any additional architectural alteration



(b) Structure of a stage in a network enriched with both parallel blocks and dense skip connections

Figure 5.1: OFAv2 networks: stage-level alterations



(a) Staged structure of the baseline network

(b) Staged structure of a network enriched with early exits

Figure 5.2: OFAv2 networks: network-level alterations

5.1.2. Extended Progressive Shrinking

The Extended Progressive Shrinking (EPS) algorithm is a modified version of the original PS algorithm, and has been designed to allow the training of the newly introduced, architecturally extended supernets, while still following the same philosophy described in the PS, i.e. training the largest architectures first and progressively sampling smaller subnets. To achieve this goal, two new steps have been added, namely:

- **Elastic Level (EL)**: this elastic step is only performed if the supernet being trained contains parallel blocks. When active, subnets can be extracted from the supernet so that, at each level, one or more blocks can be selected to be part of said subnets. An integer value is assigned to each level, the corresponding 3-bit one-hot coding representing the activation state of the 3 blocks that make up the level. Before this elastic training step is activated, all previous training phases use a value fixed to 7 (or 111), meaning that all 3 blocks in each level are trained together. Once

activated, Elastic Level consists of two phases: phase 1 allows EL to train levels by also selecting any pair of blocks, phase 2 further extends the choices to allow training of just one of the three.

- **Elastic Exit (EX)**: this elastic step is only performed if the supernet being trained is enriched with early exits. When active, subnets can be extracted from the supernet so that they are are single-exit networks terminating at the N^{th} exit. Before this elastic training step is activated, all previous training phases use a value of N=5, meaning that the extracted subnets always select the last exit. Once activated, Elastic Exit consists of four phases: phase 1 allows N to be chosen in [4,5], phase 2 extends the choice to [3,4,5], phase 3 to [2,3,4,5], and finally phase 4 to [1,2,3,4,5].

When both these steps are required, the Elastic Level step is performed before the Elastic Exit step and after the Elastic Kernel Size step. In this same case, the maximal network is defined as the one with all values for each elastic step fixed to its maximum, and all 5 exits active. The full EPS pipeline is shown in Figure 5.3.

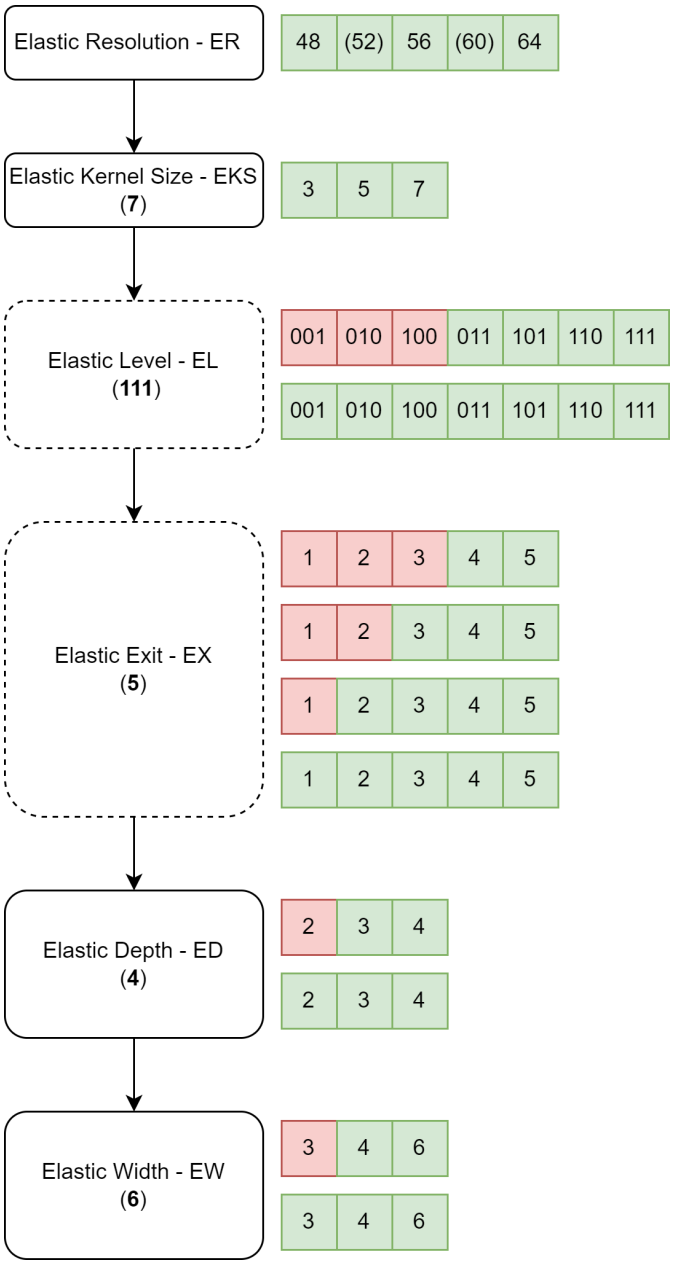


Figure 5.3: Extended Progressive Shrinking: each step can be composed of multiple phases, allowing the training algorithm to progressively activate smaller sub-networks. The figure illustrates the sequential steps performed by the training procedure, accompanied on the right by the allowed set of values in our experiments for each phase of that step. The available options for a given phase are shown in green, while the unavailable ones are reported in red. When a step has not been reached yet, the value for the corresponding modifier is set to the maximum one, reported under the name of the step. Dashed steps are performed only on supernets supporting the architectural modifications touched by the step, namely parallel blocks for the Elastic Level, and early exits for the Elastic Exit.

5.1.3. Progressively Extracting the Teacher Network

Throughout the training process, the original PS implementation uses the version of the maximal network obtained after the Elastic Resolution step as the teacher network to perform Knowledge Distillation. However, this is arguably not the optimal way to proceed. Since the maximal network is just another subnet of the dynamic supernet, a version of it could theoretically be extracted from the supernet at any time. Since during each training phase the weights of the supernet shared with what would be the maximal network are also updated, using as maximal network the subnet extracted from the results of the previous training phase rather than the initial one allows PS (and EPS) to work with a progressively better teacher network from which to apply Knowledge Distillation, resulting in improved performances.

5.1.4. Training Early-Exit Networks

During the first training step of supernets with early exits, the main goal is to maximise the performance of such networks with all exits active. In EPS, the training and evaluation of these networks follows the the early exits ensemble approach proposed in AEP, following the DESC weights assignment strategy. Given that we are in a scenario that requires training from scratch and the use of small images (for reasons explained in 5.2.1), the results presented in 4.3.1 show that DESC is the optimal strategy to adopt in this case. The post-processing pruning step is not applied because the whole network is needed as a teacher by the following EPS training steps.

The presence of early exits also required the implementation of a new form of Knowledge Distillation. Two new approaches to KD were tested, the first being an exit-to-exit KD technique (E2E-KD), and the second being an ensemble KD technique (ENS-KD). In the former, Knowledge Distillation for a subnet exiting after a certain stage is performed with respect to the corresponding exit of the maximal network. In the latter, Knowledge Distillation for a subnet exiting after a certain stage is performed with respect to the ensemble of all the exits of the maximal network. Initial tests showed ENS-KD to be the superior method, so it was the one chosen to carry out the experiments. Once again, the ensemble is generated following the technique described in AEP and using the DESC weights assignment strategy. Algorithm 5.1 describes the EPS algorithm with ENS-KD.

Algorithm 5.1 Early-Exit Progressive Shrinking with ENS-KD, Train One Epoch

```
for batch = 0 to size(DataLoader) − 1 do
    batch_images, batch_labels = DataLoader(batch)
    ElasticResizing.update(batch)
    if kd_ratio > 0 then
        teach_soft_logits = TeacherNet(batch_images)
        w_teach_slog = [ ]
        for i = 0 to len(ens_weights) do
            w_teach_soft_logits[i]= ens_weight[i] · teach_soft_logits[i]
        end for
        teach_soft_labels = softmax(sum(stack(w_teach_soft_logits)))
    end if
    subents_losses = [ ]
    for j = 1 to Nsubnets do
        subnet_config = Network.sampleSubnetConfig()
        Network.activateSubnet(subnet_config)
        output = Network(batch_images)
        if kd_ratio = 0 then
            loss = LossCriterion(output, bath_labels)
        else
            kd_loss = KDLossCriterion(output, teach_soft_labels)
            sub_loss = LossCriterion(output, labels)
            loss = sub_loss + kd_loss · kd_ratio
        end if
        subents_losses.append(loss)
        Network.updateAccuracy(output, batch_labels)
        loss.backpropagate()
    end for
    Network.optmizierStep()
end for
return subents_losses.avg(), Network.getAccuracy()
```

5.2. Experiments

Experiments were run for all the eight supernet configurations previously shown in Table 5.1, ignoring SE_B_OFAMobileNetV3 which is just a re-implementation of the OFAMo-

bileNetV3 network using modular stages. In this study, the original OFAMobileNetV3 is the baseline network to study the impact of each architectural change on the final performance. Each of the supernet configurations underwent the training process four times, as they were trained for each combination of width multiplier (1.0 and 1.2) and teacher network strategy (fixed to the one obtained after the first step or being progressively extracted phase by phase).

The same experimental setup was used for each supernet to ensure a fair comparison of the results. The dataset used, as well as the set of hyperparameters and the setup of the EPS procedure, are described in this section.

5.2.1. Dataset

Given the increased number of supernets to be tested, as well as the necessity to run each of them twice (for width multipliers WM 1.0 and 1.2), training time was a very important factor to consider. For this reason, while the original OFA architecture was trained on ImageNet [86], the OFAv2 algorithm was tested on the Tiny ImageNet dataset [12]. Tiny ImageNet is a subset of ImageNet consisting of 100 000 training images (15% of which were used for validation) and 10 000 test images, downsampled to a 64x64 resolution. Due to the reduced number of images and their smaller size, the experiments could be completed in a reasonable amount of time, taking an average of one and a half days when run on a single NVIDIA Quadro RTX 6000 GPU. In contrast, the original OFA training on ImageNet was reported to take a day and a half on a cluster of 32 NVIDIA V100 GPUs, making it prohibitively expensive to reproduce without a significant hardware investment.

5.2.2. Hyperparameters

For the training procedure, the SGD optimizer was adopted with momentum equal to 0.9 and a weight decay parameter set to $3 \cdot 10^{-5}$. The learning rate was adjusted using a Cosine Annealing scheduler [87]. The batch size was set to 200. Batch Normalization layers were set with momentum equal to 0.1 and an epsilon value of 10^{-5}. Elastic Resolution was run with image sizes within the range [48, 56, 64], using the original image size as the upper limit, as done by OFA for ImageNet. The specific hyperparameters used in each EPS training phase are reported in Table 5.2. The models were implemented using PyTorch 1.12.1 and ran on an NVIDIA Quadro RTX 6000 GPU. The OFAv2 code extends that of OFA by supporting the use of any input dataset, not just ImageNet, and the new Extended Progressive Shrinking pipeline is supported for both distributed and non-distributed training.

Table 5.2: Hyperparameters for the Extended Progressive Shrinking phases

Step	Learning Rate	Epochs	Warmup Ep.	#Subnets
Full	$1.0 \cdot 10^{-3}$	180	0	0
KS	$3.0 \cdot 10^{-2}$	120	5	1
NW1	$2.5 \cdot 10^{-3}$	25	0	2
NW2	$7.5 \cdot 10^{-3}$	120	5	2
ND1	$2.5 \cdot 10^{-3}$	25	0	2
ND2	$7.5 \cdot 10^{-3}$	60	5	2
ND3	$1.0 \cdot 10^{-2}$	90	5	2
ND4	$3.0 \cdot 10^{-2}$	120	5	2
D1	$2.5 \cdot 10^{-3}$	25	0	2
D2	$7.5 \cdot 10^{-3}$	120	5	2
E1	$2.5 \cdot 10^{-3}$	25	0	4
E2	$7.5 \cdot 10^{-3}$	120	5	4

5.3. Results and Discussion

The experiments were performed for both 1.0 and 1.2 width multipliers, as mentioned in Section 5.2. After each phase of the proposed EPS algorithm, the accuracy scores over the test set are calculated. For each of the training phases, all possible combinations of selectable elastic parameters are sampled, taking into account also those unlocked in previous phases and steps. The subnets that are tested are those that can be extracted from the dynamic supernet by activating those combinations of parameters, while keeping their value constant throughout the network. For example, testing EE_B_OFAMobileNetV3 in phase EX2 (second phase of Elastic Exit) would activate all subnets obtained by selecting resolutions in [48,56,64], kernel size in [3,5,7], network exit in [3,4,5], stage depth in [4] and expansion ratio in [6], for a total of 27 subnets. The results for the two width multipliers are summarised in Table 5.3 and Table 5.4 respectively. Only the results obtained after each EPS step are shown. For each of the steps, the average accuracy values obtained by the sampled subnets (avg) and the performance of the best subnet (best) are shown. Finally, for each configuration, the two approaches to teacher network selection are compared: the results obtained using the standard technique presented in OFA are in the top row, while those obtained using the progressive extraction of the teacher network are in the bottom row.

Looking at the results of the first step, where only Elastic Resolution is applied, the best configuration is, both as an average and as the best subnet, the one with a single exit, dense skip connections and parallel blocks. This result prove how it is possible to better generalise the information contained in images of different sizes by having a more varied

set of blocks and a greater number of connections. These results are confirmed regardless of the width multiplier considered.

The Elastic Kernel Size step represents an overall improvement for each configuration, especially for the models enriched with early exits, which gain approximately 20% accuracy both on average and as a better subnet performance regardless of the width multiplier. The best average configuration is still the one with a single exit, dense skip connections, parallel blocks, when exploiting the progressively updated teacher network. The best subnet is the one with a single exit and dense skip connections, again using the updated teacher network.

The Elastic Level step introduced in EPS is applied to all configurations that support parallel blocks. In this case the configuration with the best accuracy is EE_DP_OFAMobileNetV3. Already in this EPS step, the best subnet found is better than the overall best performing model obtained through OFA algorithm with fixed teacher network, regardless of the width multiplier considered. As far as the average performance is concerned, it can be seen that the choice of very light blocks in parallel networks leads to subnets that are very heterogeneous among themselves and with a poor performing majority.

The Elastic Exit step introduced in EPS is applied to all configurations with early exits. From this step onwards, for both width multiplier configurations, it can be observed that single-block architectures benefit more from EPS than those with parallel blocks. The use of the updated teacher network also proves to be particularly beneficial in terms of average and best accuracy. As for the Elastic Depth and Elastic Width steps, the same considerations made regarding the Elastic Exit step continue to hold true.

Thus, looking at the final result of the OFAv2 algorithm, it can be said that this represents a substantial improvement over the baseline represented by OFA. In particular, by looking at Table 5.3, we can seen that the EE_D_OFAMobileNetV3 configuration gains 12.07% in accuracy compared to the reference OFA algorithm. Looking at Table 5.4, we see the same behaviour, with a difference of 11.53%. It is also interesting to note that the progressive teacher updating technique alone results in an improvement in the model of more than 3% accuracy, regardless of the width multiplier. Finally, the results show that the average performance of OFAv2 is better than that of the best subnet obtained by OFA.

Regarding the width multipliers, the results obtained confirm that OFAv2 scales well with the increase in the number of filters used, gaining almost 2% accuracy with this simple parameter change. From an overall perspective, the results show that the use of the defined parallel blocks does not have a beneficial effect on the final performance

obtained after the whole the training procedure. This may be due to the nature of the data set rather than the nature of the task, and could therefore have the opposite effect in other configurations. On the contrary, the addition of early exits produces noticeable improvements over the baseline.

Interestingly, networks with early exits perform worse than their single-exit counterparts in the first training step, a rather strange behaviour compared to that shown in AEP [13], where early-exit networks clearly outperform single-exit networks. This behaviour is most likely to be attributed to the use of Elastic Resolution. Single-exit networks are more suited for steadily optimizing the weights for various image sizes, while updating multiple exits to support all image sizes appears to be a difficult goal to achieve. In any case, from the Elastic Kernel Size step onwards, networks with early exits close the gap and overcome the performance of single-exit networks, providing a significant accuracy improvement at the end of the training. The addition of dense skip connections provides a marginal improvement on the final architectures. Although the performance gain is minimal compared to that provided by the early exits, their improvements stack up, making it possible to combine these two techniques to maximise subnets performance.

Finally, the use of progressively extracted teacher networks, revised with the best configuration of weights after each phase of the algorithm, induces a significant, consistent improvement in most of the EPS steps and network configurations. In this way, the student networks are able to learn better from the more accurate teacher network, but they also improve the weights shared with the teacher, creating a mutual learning benefit.

These results indicate that the applied architectural changes, as well as the new algorithms implemented to manage the early-exit network and to take advantage of a better teacher network, are effective in improving the performance of all the subnets contained in the supernet, in some cases achieving, on average, a better accuracy than the best subnet from in the baseline network.

Table 5.3: Results obtained after every Extended Progressive Shrinking step for WM=1.0, in terms of accuracy of the best tested subnet ("best") and of their average accuracy ("avg"). For each network, the first row represents the results obtained when the teacher network is fixed to the one obtained at the first step of the process, while the second row shows the results obtained when the teacher network is extracted from the supernet obtained as a result of the previous phase of the algorithm. The symbol "*" represents a new EPS steps, not present in OFA. The value "X" indicates that for a specific network, that EPS step is not necessary. The best results for each step are highlighted in bold. The best overall results are highlighted with underlining.

NETWORK (WM = 1.0)	RESOLUTION		KERNEL SIZE		LEVEL *		EXIT *		DEPTH		WIDTH	
	avg	best-sub	avg	best-sub	avg	best-sub	avg	best-sub	avg	best-sub	avg	best-sub
OFAMobileNetV3 (Baseline)	27,28	28,14	37,46	38,97	X	X	X	X	37,67	39,35	38,11	39,83
	27,03	28,02	37,90	39,38	X	X	X	X	39,45	41,46	40,03	42,23
SE_D_OFAMobileNetV3	28,49	30,03	38,90	40,49	X	X	X	X	38,85	40,91	38,77	40,93
	29,50	30,56	39,73	**41,60**	X	X	X	X	40,37	42,34	40,83	43,07
EE_B_OFAMobileNetV3	18,08	19,01	37,08	38,51	X	X	41,49	47,17	40,55	48,24	40,68	48,80
	17,81	18,76	36,94	38,51	X	X	**43,93**	48,76	**43,47**	50,00	44,18	51,64
EE_D_OFAMobileNetV3	17,68	18,36	38,35	40,00	X	X	41,04	46,37	40,11	47,63	40,30	48,53
	17,96	18,59	38,11	39,76	X	X	43,48	**48,82**	43,31	**50,62**	**44,22**	**51,90**
SE_P_OFAMobileNetV3	28,21	29,41	38,68	40,15	23,80	39,89	X	X	24,73	38,89	24,51	37,93
	27,98	28,95	37,99	39,36	23,65	39,35	X	X	24,03	37,92	23,42	36,17
SE_DP_OFAMobileNetV3	30,32	31,40	39,71	41,10	25,47	41,26	X	X	26,28	40,39	26,09	39,13
	30,39	**31,98**	**39,75**	41,29	24,84	41,00	X	X	24,46	38,78	23,27	36,55
EE_P_OFAMobileNetV3	16,06	16,74	37,19	38,73	23,59	39,36	26,97	44,33	27,33	43,85	27,96	44,12
	16,49	17,47	37,02	38,53	24,17	40,79	27,49	45,73	28,12	45,23	29,18	46,06
EE_DP_OFAMobileNetV3	15,96	16,96	37,89	39,36	25,10	40,84	27,58	45,73	27,72	44,89	28,10	44,67
	16,19	17,07	38,03	39,63	**25,38**	**42,42**	28,37	46,74	28,95	45,89	30,20	46,24

Table 5.4: Results obtained after every Extended Progressive Shrinking step for WM=1.2, in terms of accuracy of the best tested subnet ("best") and of their average accuracy ("avg"). For each network, the first row represents the results obtained when the teacher network is fixed to the one obtained at the first step of the process, while the second row shows the results obtained when the teacher network is extracted from the supernet obtained as a result of the previous phase of the algorithm. The symbol "*" represents a new EPS steps, not present in OFA. The value "X" indicates that for a specific network, that EPS step is not necessary. The best results for each step are highlighted in bold. The best overall results are highlighted with underlining.

NETWORK (WM = 1.2)	RESOLUTION		KERNEL SIZE		LEVEL *		EXIT *		DEPTH		WIDTH	
	avg	best-sub	avg	best-sub	avg	best-sub	avg	best-sub	avg	best-sub	avg	best-sub
OFAMobileNetV3 (Baseline)	27,48	28,37	38,38	39,75	X	X	X	X	39,45	41,46	40,03	42,23
	27,74	28,66	39,54	40,62	X	X	X	X	40,54	41,82	41,15	42,71
SE_D_OFAMobileNetV3	30,05	31,38	40,48	41,98	X	X	X	X	40,23	42,17	40,32	42,44
	29,90	30,82	41,55	**43,11**	X	X	X	X	42,45	44,20	43,20	45,44
EE_B_OFAMobileNetV3	21,32	22,24	39,13	40,45	X	X	43,42	47,79	42,38	48,40	42,64	49,49
	22,07	23,38	39,51	40,97	X	X	46,26	50,52	46,09	51,83	47,13	53,37
EE_D_OFAMobileNetV3	20,86	21,98	39,77	41,46	X	X	43,24	47,47	42,56	48,47	42,79	49,93
	21,06	21,87	40,12	41,91	X	X	**46,70**	**51,07**	**46,62**	**52,47**	**47,46**	**53,76**
SE_P_OFAMobileNetV3	29,19	30,54	39,75	40,95	24,75	40,98	X	X	26,11	40,66	26,38	39,87
	28,37	29,65	39,01	40,61	24,57	40,23	X	X	25,35	39,01	24,87	37,35
SE_DP_OFAMobileNetV3	31,12	32,38	40,46	42,11	**26,38**	41,77	X	X	27,28	41,16	27,57	40,14
	31,31	**32,97**	**40,92**	42,53	26,19	41,93	X	X	26,45	39,73	25,97	38,02
EE_P_OFAMobileNetV3	19,39	20,28	38,48	40,08	24,52	40,87	28,90	46,18	29,19	45,65	29,89	46,01
	19,47	20,59	38,79	40,36	24,89	41,55	29,73	48,12	30,71	47,84	32,26	48,44
EE_DP_OFAMobileNetV3	18,71	18,99	38,38	40,24	25,89	42,03	29,51	46,88	30,03	46,60	30,28	46,74
	18,68	19,26	38,33	39,89	26,23	**42,52**	30,27	48,43	31,83	48,93	32,76	48,62

5.4. Conclusion

In this chapter we have shown how, by enriching the OFAMobileNetV3 network with new design patterns and architectural components such as dense skip connections and early exits, it is possible to obtain new supernets that significantly improve performance on the Tiny ImageNet dataset.

To enable the training of these new networks, the Extended Progressive Shrinking algorithm was presented, with two new steps added to the PS pipeline to handle parallel networks and early exit networks. These steps are called Elastic Level and Elastic Exit respectively. The main contribution to these performance gains is largely due to the presence of the early exits. The structure of early exit networks and their new training algorithms follow the previously proposed AEP methodology, allowing for better training of the intermediate layers. Furthermore, the results showed that the progressive extraction of the teacher network used to perform Knowledge Distillation offers advantages over the use of the maximal network trained in the first step, providing an additional performance boost.

Given these improvements, it is reasonable to expect that the subnets obtained from the new supernets, after the application of a search step, may vastly outperform those obtained from the standard version of OFA, both in terms of accuracy and network simplicity.

6 | Enhancing Neural Architecture Transfer

As explained in Section 3.2, NAT is a powerful NAS algorithm which, given a pretrained supernet as a starting point, is able to find a set of optimal subnets satisfying a set of objectives for a wide variety of tasks. This is achieved by combining online transfer learning with a many-objective predictor-guided evolutionary search.

Unlike OFA, whose search for the best subnet is restricted to the dataset on which the supernet was trained, NAT allows the search for the best subnet to be performed on different datasets. For this reason, it was decided not to adapt the search part of OFA in OFAv2, instead, an extension of the NAT algorithm built on top of OFAv2 and called NATv2 is presented in this chapter. NATv2 makes it possible to perform search and find the best subnet starting from any of the EPS-trained supernets introduced in OFAv2.

Given the wider range of possible architectural configurations searchable by the algorithm due to the use of the new supernets, it would be reasonable to assume that NATv2 would return better architectures than those obtained by starting from the PS-trained OFAMobileNetV3 network. To test this assumption, this chapter conducts an extensive study to see how the new architectural designs affect the structure and performance of the architectures returned. In addition, the impact of various algorithmic improvements and the presence of additional execution steps over the original NAT algorithm are evaluated, and their benefits assessed.

6.1. Method

This section describes how NATv2 extends the original NAT technique to work with any of the new supernet architectures introduced in OFAv2 as a starting point. Changes to the original algorithm that are not related to the specific supernet used are also discussed, including a new methodology for sampling subnets and a revised way of managing the archive of architectures. Finally, two new execution steps introduced in NATv2 are ex-

plained. These are a pre-processing step to complement the new archive initialisation method, and two alternative post-processing methods to further increase the accuracy of the subnets returned at the end of the algorithm.

6.1.1. Extended Search Space

Due to the existence of new supernet architectures, it becomes necessary to define an encoding methodology for them as well, in order to be able to perform the evolutionary search. In particular, such encodings shall include information representing which exit is selected when early exits are present, and shall uniquely identify the activation state of blocks within the same level when the network supports parallel blocks. As for the activation state of dense skip connections, it can be inferred directly from the depth of the stage, so there is no need to represent it explicitly in the encoding.

The encodings for the new supernets, shown in Figure 6.1, are created by extending the integer encoding proposed by NAT for the OFAMobileNetV3 network. Again, the encoded strings are of fixed length. To represent which exit is active in early exit networks, a new variable is added to the third position of the encoded strings; its value is directly related to the selected exit. To represent the activation state of blocks in parallel networks, the set of possible encodings for the 20 IRB blocks has been extended by transforming the kernel size and expansion ratio pairs into triples, where the integer value corresponding to the activation state of each level is also present. The same rule as in OFAv2 is used to encode the activation state of the level.

6.1.2. Training Early-Exit Networks

As in OFAv2, also in NATv2 it becomes necessary to develop algorithms to properly train the newly introduced early exit supernets. The structure of such algorithms is based on the AEP and EPS algorithms presented earlier. For the warmup phases, in which the maximal networks are trained with all their exits active, the AEP algorithm is used following the DESC weighting strategy. For the adaptation phase, in which the supernet is fine-tuned by sequentially activating subnets within it, a training method corresponding to the last phase of the EPS algorithm is used, since all values of the elastic parameters are selectable from the start. NAT uses Knowledge Distillation to improve subnets during the supernet adaptation phase; the ENS-KD technique proposed in OFAv2 is used in NATv2 to adapt early-exit supernets. To be consistent with the EPS training that the supernets underwent in OFAv2, the subnets activated during the supernet adaptation step are always single-exit networks.

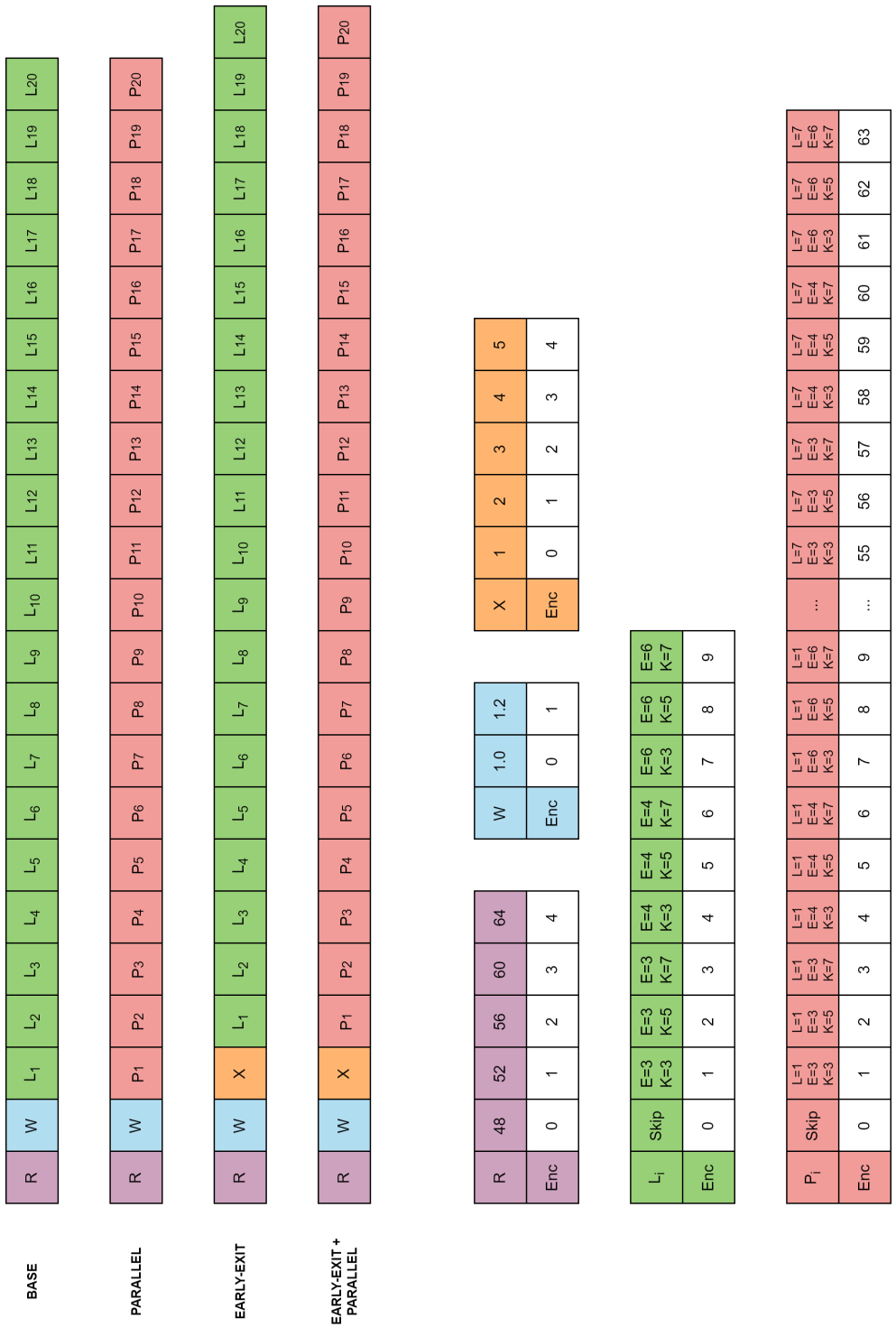


Figure 6.1: Search space encodings for the different types of supernet architecture. R is the encoded value of the size of the images entering the network. W is the encoded value width multiplier of the of the network. X is the encoded value of selected exit, for supernets that support early exits. For non-parallel networks, L_i is the encoding of the $i-th$ IRB/IB block, which depends on the active kernel size and expansion ratio pair . Similarly, for parallel networks, P_i is the encoding of the $i-th$ level, which depends on the triplet of active kernel size, expansion ratio and activation states of each block in that level.

6.1.3. Updated Archive Management

One of the major changes introduced by NATv2 over its original counterpart is the way in which the architectures archive is managed throughout the execution of the algorithm. These changes mainly focus on two steps of the NAT algorithm: the archive initialisation step and the archive growth step.

Focusing on the archive initialisation step, NAT performs it by uniformly sampling architectures from the search space. To generate the encodings associated with the set of initial architectures, for each variable in the search space, one value is uniformly sampled from the possible values that can occur at that position. The resulting strings are then decoded to obtain a valid subnet configurations. The main problem with this approach is that it results in the selection of networks that are strongly biased towards those with maximum stage depths. In fact, the possible encodings for a skippable IRB block (i.e. the 3^{rd} or 4^{th} in a stage) are the values from 0 to 9, but only sampling the value 0 results in skipping that block. Because of the uniform sampling, all values have the same probability of being selected, so stages with depth 3 are not very common, and stages with depth 2 are quite rare. The problem is exacerbated when parallel blocks are supported, in which case the encodings go from 0 to 63, but again only a value of 0 means that the block must be skipped. To mitigate this, NATv2 performs the archive initialisation step by sampling subnets from the architecture space rather than the search space. This ensures that the depths of the stages in the sampled networks are evenly distributed, while maintaining a uniform distribution in the structure of the retained IRB blocks.

As for the archive growth step, NATv2 replaces it with a subnets substitution step. Instead of selecting a limited number of architectures to initially fill the archive and then letting it grow iteration by iteration, NATv2 directly populates the archive with a larger number of architectures. At each iteration, the worst architectures from the archive are replaced by those found by the evolutionary search, so that the size of the archive remains constant. In this way, the average quality of the architectures in the archive should improve, and so should the quality of the error predictor, whose model is learned from those same architectures. This strategy also gives the predictor model access to a larger pool of input data from which to learn in the initial iterations.

6.1.4. Pre-Processing

In addition to sampling from the architecture space, NATv2 incorporates a pre-processing step into the archive initialisation step. This pre-processing involves sampling and evaluating a number of architectures proportionally larger than the desired archive size A_s.

Only the top $A_s - 2$ network, plus the maximal and minimal networks, are selected to form the initial archive. By starting with a set of high quality architectures, it may be possible to end up with better performing subnets at the end of the algorithm. The use of the pre-processing step comes at the cost of increased execution time, as a large number of subnets must be evaluated initially.

6.1.5. Post-Processing

Due to the many-objective optimisation nature of the search process, it is possible that the subnets returned upon completion of the algorithm may excel in some of the additional objectives, but have not yet reached their full classification potential. In order to maximise the accuracy of these networks, NATv2 introduces two different post-processing methods. While the first method is applicable to subnets derived from any supernet, the second is reserved for those extracted from supernets with early exits. Both methods involve two fine-tuning phases, which are performed sequentially.

In the first phase, the network is fine-tuned and its performance is continuously evaluated on the validation set. This first phase is used to find the optimal number of epochs for which to fine-tune the network. In the second phase, the network is fine-tuned using both the training set and the validation set as training data, for a number of epochs equal to that found in the first phase. Finally, the classification performance of the fine-tuned network is evaluated on the test set.

The difference between the two post-processing methods lies in the algorithm used to perform the fine-tuning. While the first method directly fine-tunes the single-exit networks returned by NATv2 as they are, the second method in based on AEP. In particular, all the exits above the one selected by NATv2 for the subnet are extracted from the supernet and reattached to the subnet, then a joint fine-tuning process is performed. The advantage of this second method is that it allows greater accuracy gains than the traditional fine-tuning method, but at the cost of increasing the score for other objectives relative to the single-exit network found by NATv2. For this reason, its use is only suggested on subnets whose scores for additional objectives are low enough that even a moderate increase would still satisfy the constraints.

6.2. Experiments

NATv2 experiments were performed on the eight OFAv2 supernets trained with the Extended Progressive Shrinking algorithm. To perform NAT and NATv2 experiments, two

versions of the same pretrained supernet (for width multipliers of 1.0 and 1.2) are required. The same hyperparameters were used for all experiments to ensure comparable results. As in OFAv2, the OFAMobileNetV3 network is used in NATv2 as the baseline to study the effects of architectural changes.

The experiments on were carried out in four phases: In the first, using the new search space encodings, NATv2 was tested on the OFAv2 supernets for a wide range of datasets, and the LGBM [77] error predictor was used. This phase represents the baseline for studying the effects of algorithmic changes. From the second phase, the input features fed to the error predictor were created using an integer encoding, the new archive management strategy was adopted and the time-saving measures began to be applied. In the third phase, the pre-processing step was introduced. In the fourth and final phase, the post-processing steps were added, and OFAv2 supernets trained with the progressively extracted teacher network (which was a later addition) were used as a starting point.

Experiments were also conducted to evaluate and compare the performance of the error predictor models under different conditions, such as using different amounts of training data and different feature encodings. Finally, a study was carried out to find the best optimisation strategy to be used in the post-processing step.

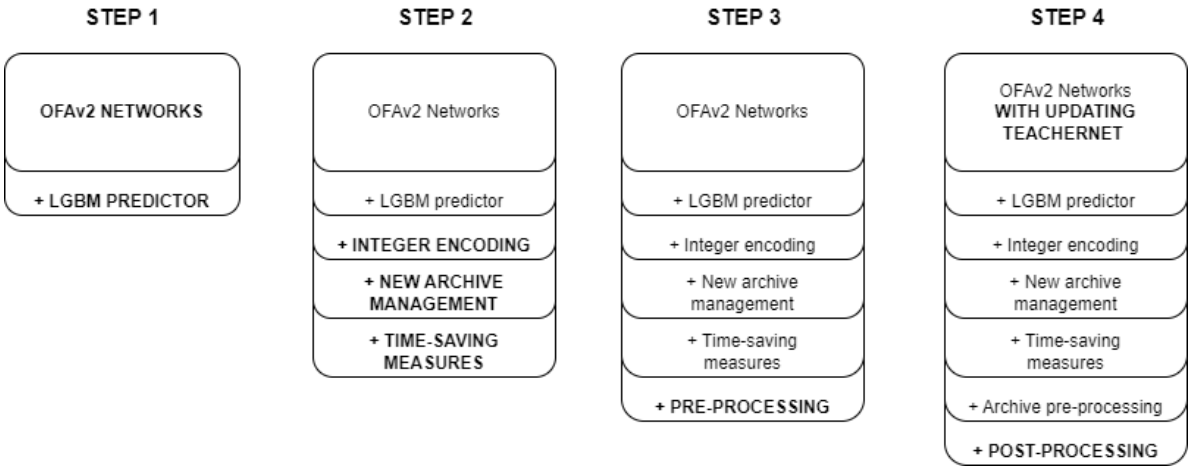


Figure 6.2: Evolution of NATv2

6.2.1. Datasets

The experiments were performed on the same benchmark image classification datasets used in NAT, with the exception of ImageNet. Unlike OFA, the OFAv2 networks were trained on Tiny ImageNet instead of ImageNet. For consistency and time reasons, the Tiny ImageNet dataset was also used in NATv2 instead of ImageNet. In contrast to

NAT, for datasets where a validation set already existed, this was used. Although 11 datasets were available, due to the extremely large number of experiments to be run and the considerable time required to complete each one, most of the experiments were run on CIFAR-10, CIFAR-100 and Tiny ImageNet. Table 3.2 shows the properties of the datasets used in the NATv2 experiments.

Table 6.1: NATv2 benchmark datasets

Dataset	Classes	Train size	Validation size	Test size
Tiny ImageNet [12]	200	85 000	15 000	10 000
CINIC-10[67]	10	90 000	90 000	90 000
CIFAR-10 [68]	10	45 000	5 000	10 000
CIFAR-100 [68]	100	45 000	5 000	10 000
STL-10 [69]	10	4 250	750	8 000
Food-101 [70]	101	64 288	11 362	25 250
Stanford Cars [71]	196	6 923	1 221	8 041
FGVC Aircraft [72]	100	3 334	3 333	3 333
DTD [73]	47	1 880	1 880	1 880
Oxford-IIIT Pets [74]	37	3 128	552	3 669
Oxford Flowers102 [75]	102	1 040	1 000	6 149

6.2.2. Performance Predictors

The performance of all the predictor models implemented in NAT, with the addition of the CatBoost model, was thoroughly tested and compared under a multitude of aspects. The set of predictor models evaluated consisted of Radial Basis Function (RBF) [76], Radial Basis Function ensemble (RBFs), Multi-Layer Perceptron (MLP) [40], End-to-End Random Forest-based Performance Predictor (E2EPP) [88], Classification and Regression Tree (CARTS) [89], Gaussian Process (GP) [90], Support Vector Regressor (SVR) [91], Ridge Regressor [92], K-Nearest Neighbours Regressor (KNN) [93], Bayesian Ridge Regressor [94], Light Gradient Boosting Machine (LGBM) [77] and CatBoost [95].

To test them, the predictors were fitted using as input 100, 150, 200, 250 and 300 randomly extracted sub-architectures from the OFAMobileNetV3, EE_B_OFAMobileNetV3 and SE_P_OFAMobileNetV3 supernets, and their exact error as the target. The subnets were extracted from NATv2-trained supernets on the CIFAR-10 dataset. These experiments were also performed by feeding the predictor models with both one-hot and integer encoded features extracted from the compressed representations of the architectures. For

each experiment, the test set consisted of 25 architectures, again randomly extracted from the supernets. The results were recorded in terms of RMSE, time taken to fit the model and the three correlation metrics of Pearson, Spearman and Kendall. To ensure meaningful results, each experiment was run 10 separate times using different architectures as training data, but the same test set.

From the results obtained, it is possible to understand how each model behaves with different amounts of training data and the effects of using different feature encodings, but also to study how different models compare under the same initial conditions.

6.2.3. Time-Saving Optimisations

In order to reduce the long execution time of the NAT algorithm, two optimisations have been introduced in NATv2 that guarantee significant time savings. Firstly, by re-implementing the originally sequential distribution estimation process as a multi-processing algorithm, the distributions for different variables in the encoding can now be fitted in parallel, taking advantage of the multiple cores available on the machine, saving a few minutes of computation per iteration.

Secondly, and by far the most beneficial improvement, instead of validating the performance of the supernet after each fine-tuning epoch of the adaptation step, validation is only performed after the final fine-tuning epoch of each adaptation step. The supernet is validated by means of its subnets, in particular, the tested subnets are those whose structure can be activated by setting and fixing the combinations of elastic parameters of the dynamic supernet (kernel size, expansion ratio, ...) to the minimum or maximum, similarly to the scheme presented in 5.3 for testing OFAv2 subnets but without the intermediate values. In this case, all possible values for the elastic parameters are unlocked from the start.

This reduced-validation approach in no way compromises the effectiveness of NATv2, since the only validation steps whose results are of any use are those performed after the completion of the last fine-tuning epoch of each adaptation step, i.e. the validations that are still being performed. Performing inference on large numbers of subnets is a very time-consuming process, and the very reason why the performance predictor was introduced into evolutionary search in the first place. By avoiding unnecessary intermediate validation steps, the execution time of NATv2 can be almost halved.

6.2.4. Post-Processing

Many combinations of optimizers and learning rates were tested to maximise the effectiveness of the post-processing step. These experiments were run on subnets obtained from the first set of NATv2 experiments on the CIFAR-100 dataset, in particular on the subnets obtained from the OFAMobileNetV3, SE_D_OFAMobileNetV3, EE_B_OFAMobileNetV3 and EE_D_OFAMobileNetV3 supernets when NATv2 was run with the objectives "accuracy & parameters" and "accuracy & MACs". The optimal combinations found were later used when post-processing was added to the NATv2 pipeline in the fourth experimental phase. The optimizers tested were SGD, AdamW [96] and Ranger (a combination of LookAhead and RAdam) [97, 98], and the initial learning rates were set to either 10^{-4} or 10^{-5}. In the case of the AEP-based post-processing, experiments were performed for all four exits weighting strategies.

6.2.5. Hyperparameters

To train the NATv2 models, the SGD optimizer was used with a momentum of 0.9 and a weight decay parameter set to $3 \cdot 10^{-4}$. The learning rate, initially set to $2.5 \cdot 10^{-3}$, was adjusted using a cosine annealing scheduler. The batch size was set to 256. During each supernet adaptation epoch, 4 subnets per batch were sampled. The same hyperparameters were used for the warmup phases, with the exception of the initial learning rate which was set to $7.5 \cdot 10^{-3}$.

For the post-processing, the previously reported values for momentum and weight decay were used for SGD, while no additional hyperparameters were specified for the other two optimizers. The batch size was set to 64. The networks were fine-tuned for a maximum of 150 epochs, using a cosine annealing learning rate scheduler. An early stopping technique was used, the patience value was set to 30 epochs with respect to the validation loss.

All models were implemented using PyTorch 1.12.1 and experiments were run on an NVIDIA Quadro RTX 6000 GPU. The evolutionary algorithms used to perform the NATv2 search step were taken from the pymoo library [99].

6.3. Results and Discussion

This section presents the results obtained by NATv2, and assesses the benefits of using different supernet architectures from the baseline OFAMobileNetV3. It also reports on the impact of modifications to the algorithm. In addition, the results of two studies, one aimed at identifying the best performance predictor to be used during search and

the other aimed at identifying the best optimisation strategies to be used during post-processing, are presented and discussed. For the sake of simplicity, we will use notations such as “EE_D subnets” and “baseline subnets” instead of “subnets derived from the EE_D_OFAMobileNetV3 supernet” and “subnets derived from the OFAMobileNetV3 supernet”. The same applies to the other supernets.

6.3.1. Performance Predictors Analysis

First, the predictors were compared by fixing the size of the training set and the type of encoding used to generate the input features. The performance of the surrogate models was directly compared in terms of the correlation obtained for subnets extracted from the OFAMobileNetV3, EE_B_OFAMobileNetV3 and SE_P_OFAMobileNetV3 supernets, in order to investigate the effect of the main architectural changes and of the new compressed representations on the predictors.

The models that achieve the highest correlations are CatBoost and LGBM, closely followed by CARTS and E2EPP. CatBoost and LGBM also consistently have the smallest confidence intervals. The results show that, in most cases, the highest correlations are obtained for EE_B subnets, and more rarely for baseline subnets. Correlation values for parallel subnets are almost always lowest. In terms of RMSE, the lowest results are obtained for baseline subnets, although correlation is a more important metric for our purposes. In terms of the time taken to fit the models, given the relatively small size of the training set, they all could be fitted fairly quickly, although of these, E2EPP and RBFs are the slowest, followed by CatBoost. It is also apparent that some of the models, such as MLP and KNN, struggle when the integer encoding is used to generate the input features, while they perform decently when the one-hot encoding is used. Figure 6.3 shows, as an example, the rho correlations obtained by each predictor on each type of supernet, in the case where the number of training samples is set to 300 and the integer feature encoding is used.

Complementarily, the predictors can be compared for different sizes of the training set when the supernet and feature encoding are fixed. Regardless of the encoding used, the correlation values tend to increase with the number of architectures in the training set. This trend is particularly noticeable for subnets represented by larger compressed representations, such as those extracted from early exit or parallel supernets. Figure 6.4 shows, as an example, the rho correlations obtained by each predictor in the case where the initial supernet is EE_B_OFAMobileNetV3 and integer feature encoding is used.

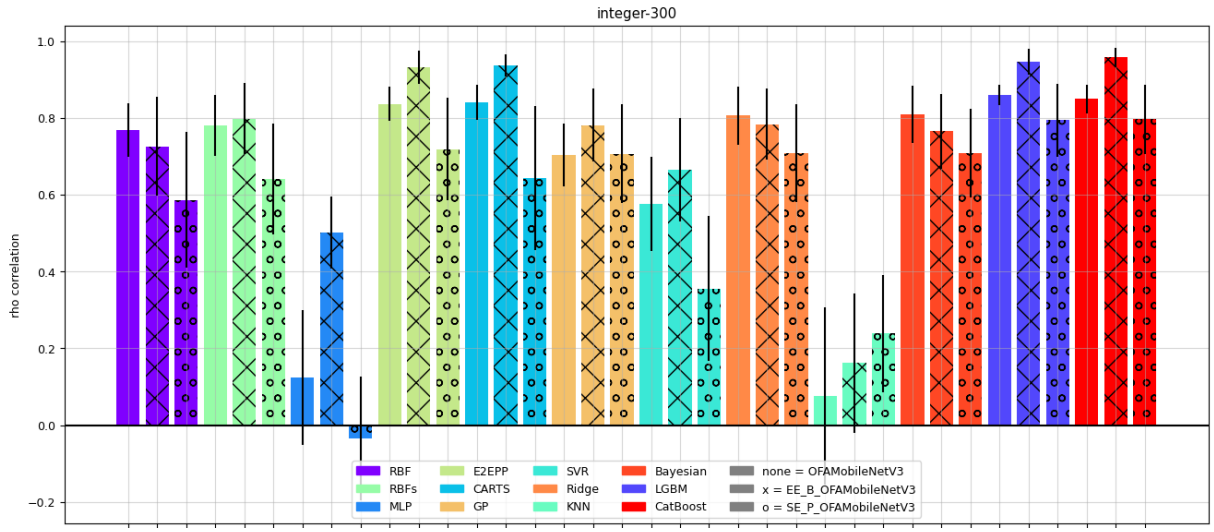


Figure 6.3: Rho correlation values for different error predictor models and network types. Predictors were trained on 300 samples using integer-encoded input features. Each set of three bars represents the correlation scores achieved by the same predictor on different network types, represented by the different patterns. Different predictors are represented by different colours, and the order of presentation follows that of the columns in the legend. See subsection 6.2.2 for the list of predictors and their acronyms.

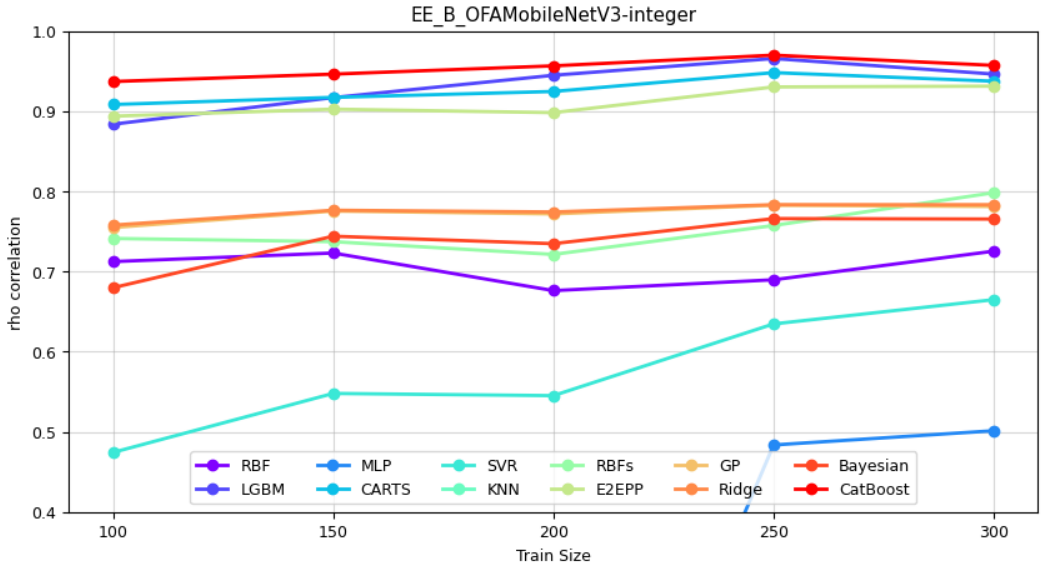


Figure 6.4: Rho correlation values for different error predictor models and training set sizes. Predictors were trained on subnets extracted from EE_B_OFAMobileNetV3 using integer-encoded input features. Different predictors are represented by different colours. See subsection 6.2.2 for the list of predictors and their acronyms.

Finally, when the predictor is fixed, the correlations are evaluated for different training set sizes when the input features were created using integer encoding and one-hot encoding. The results show that there is no clearly superior encoding. Rather, the optimal one depends on the predictor model chosen. For example, for the CatBoost model, integer encoding always outperforms one-hot encoding, and LGBM behaves very similarly. On the other hand, regardless of the supernet or training set size, the MLP and KNN models perform poorly when the integer encoding is used. In the case of the E2EPP and CARTS models, the best encoding depends on the type of subnet; in particular, for EE_B subnets, the one-hot encoding works better, while for baseline and EE_P subnets the integer encoding is the best solution. Figure 6.5 shows, as an example, the rho correlations achieved by the LGBM predictor for different input feature encodings as a function of the number of training samples.

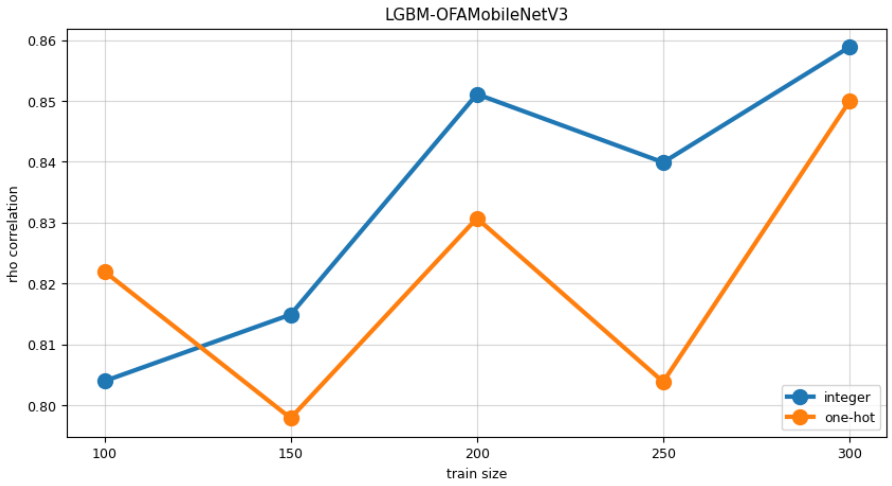


Figure 6.5: Rho correlation values achieved by the LGBM predictor for different training set sizes and input feature encodings. The model was trained on subnets extracted from OFAMobileNetV3.

In general, the choice of error predictor should be based on maximising of the correlation between the subnets true errors and the errors predicted by the model. However, this is not the only factor to consider, as the time required to fit each model must also be taken into account. As mentioned above, of the predictor models tested, CatBoost and LGBM are the two that achieve the highest correlations. CatBoost is slightly better than LGBM, but it is also much slower, which is why LGBM was ultimately chosen as the predictor model for the NATv2 experiments. The fact that the correlation increases with the size of the training set also justifies the decision to replace the small but growing archive of NAT with a larger fixed-size archive in NATv2, thus improving the performance of the predictor model, especially in the early iterations.

6.3.2. Post-Processing Optimisation

As mentioned in 6.2.4, a study was conducted to determine the best combinations of optimizers and learning rates to use for the NATv2 post-processing step.

The first notable result of this study is that, regardless of the type of post-processing, for each optimizer the average performance obtained by setting the initial learning rate to 10^{-4} is better than that obtained by setting it to 10^{-5}.

Secondly, the results show that when post-processing is performed directly on the single-exit networks returned by NATv2, the best optimisation strategy to use depends on the architecture of the supernet from which the subnets were derived, in particular whether it lacked or contained early exits. In the former case, using the SGD optimizer with the initial learning rate set to 10^{-4} proved to be the best optimisation strategy. This increased the average accuracy of the tested subnets from 64.52% to 65.90%. In the latter case, however, SGD didn't perform as well as AdamW and Ranger. The results obtained with these two optimizers were similar on average, but AdamW's results were less prone to oscillations. For this reason, in the context of performing the standard post-processing on subnets extracted from early-exit supernets, the best strategy turned out to be to use the AdamW optimizer with an initial learning rate set to 10^{-4}. In this case, the average accuracy of the subnets increases from the 65.89% to 67.62%.

Regarding the AEP-based post-processing method, SGD performed poorly regardless of the learning rate used, whereas both AdamW and Ranger performed well, especially when the ASC or UNIF weighting strategies were used. Using the AdamW optimizer with the initial learning rate set to 10^{-4} and the UNIF weighting strategy resulted in the best performance, with the average subnet accuracy increasing from 65.89% to 68.55%.

When comparing the two post-processing methods, the AEP-based method proved superior in terms of raw performance improvement, producing the most accurate subnet in 15 out of 16 cases. However, it should be noted that this ensemble method also leads to an increase in the values of other metrics due to the reintroduction of branches into the subnets.

Figure 6.6 shows, for each set of subnets obtained by running NATv2 on a given supernet for a set of objectives, the accuracy scores originally obtained by those subnets, together with the scores obtained with the two best optimisation strategies for both post-processing methods, where applicable. In this case, the optimisation strategies shown are the best on average for the specific supernet/objectives pair, so they don't necessarily correspond to the best overall strategies discussed above.

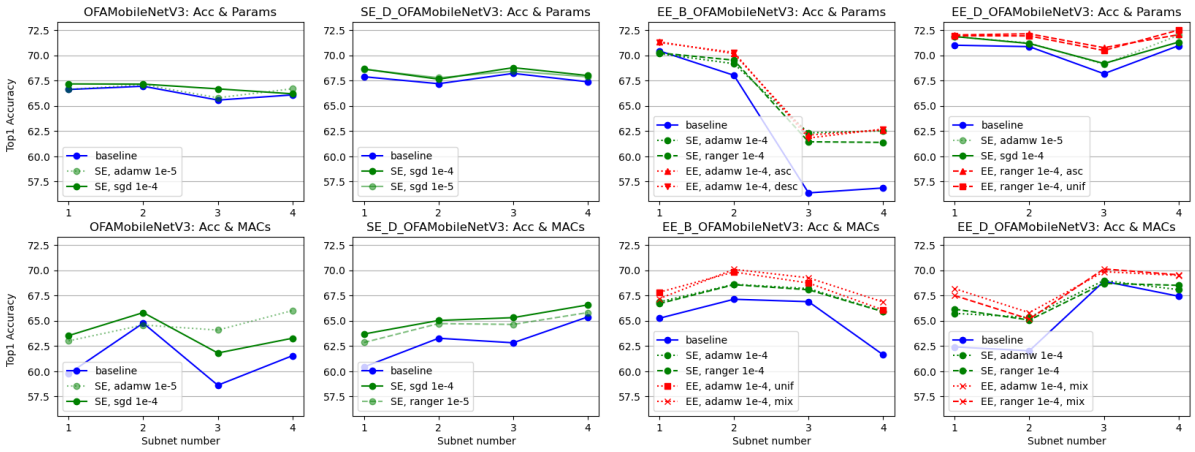


Figure 6.6: The best post-processing optimisation strategies for each tested supernet/objectives pair on CIFAR-100. Plots in the same column show results for the same supernets while varying the set of NATv2 objectives, whereas plots in the same row show results for different supernets optimised for a fixed set of objectives. Results in green represent the use of the standard post-processing method. Results in red represent the used of the AEP-based post-processing method. Different line styles represent different optimizers (solid=SGD, dotted=AdamW, dashed=Ranger). Different transparencies represent different initial learning rates (opaque=10^{-4}, transparent=10^{-5}). Different markers represent different exits weighting strategies (upward triangle=ASC, downward triangle=DESC, square=UNIF, cross=MIX).

6.3.3. New Supernets and Algorithm

This analysis begins by comparing the results obtained by NATv2 in its first incarnation (i.e. the version of the algorithm that uses the LGBM predictor and that supports the OFAv2 networks and their search spaces, but not the other modifications discussed in 6.1), on all eight OFAv2 pretrained supernets. With the exception of the different predictor model, this version of NATv2 matches NAT when the supernet being experimented on is OFAMobileNetV3, so this experimental setting is our baseline. This first set of experiments was run with CIFAR-10, CIFAR-100 and Tiny ImageNet as target datasets, and "accuracy", "accuracy & parameters", "accuracy & MACs" and "accuracy & latency" as target objectives. The results are summarised in the plots in Figure 6.7. The focus here is to understand whether architectural changes alone can help to achieve subnets that are better than those found using the OFAMobileNetV3 supernet as a starting point. It is clear from the figure that, regardless of the target dataset and set of objectives, there are subnets derived from the new supernets that largely outperform those derived from the base supernet.

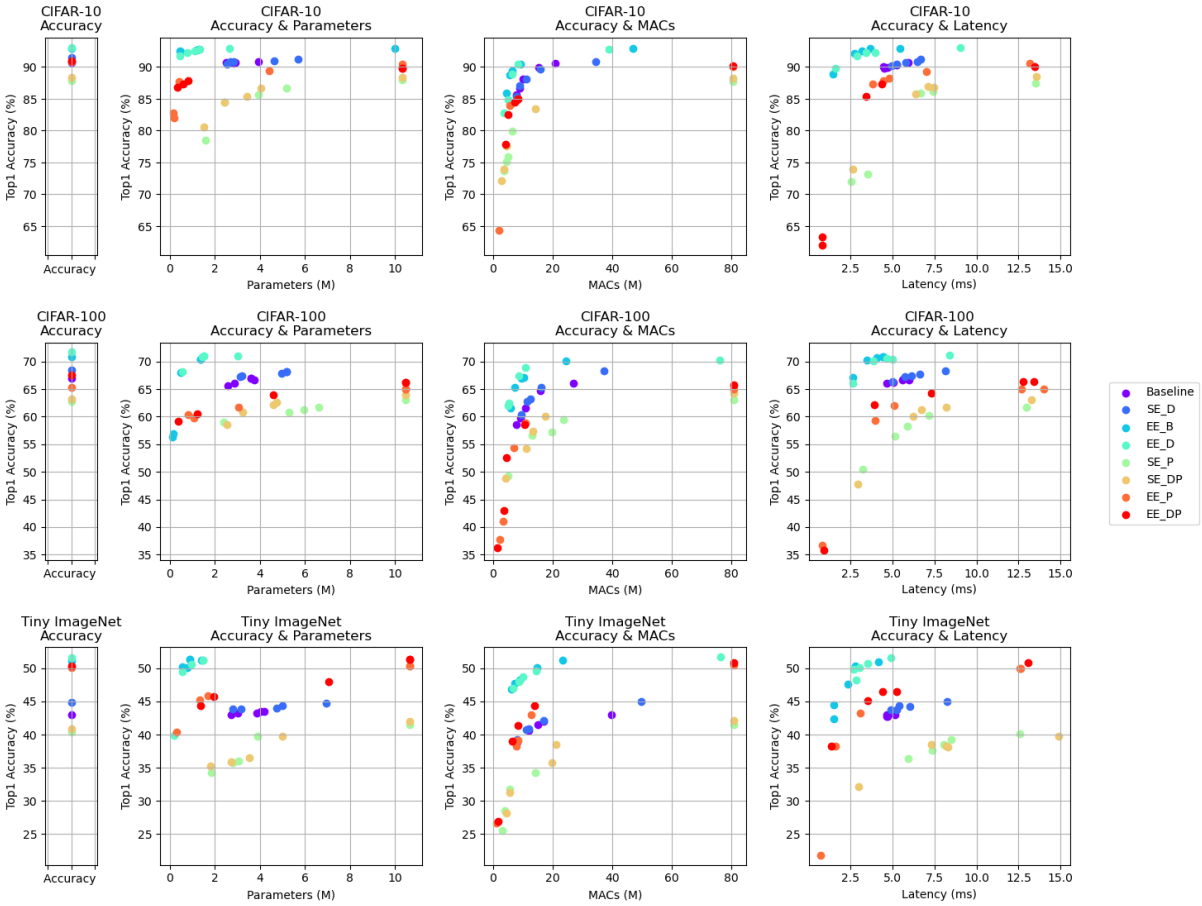


Figure 6.7: Results of the first NATv2 experimental phase, using all eight OFAv2 supernets as starting points. Rows show the results obtained on different datasets. Columns show the results obtained for different target objectives. For bi-objective optimisations, the more a result is in the top left-hand corner of the plot, the better it is.

Similar to what was shown in OFAv2, EE_D subnets are the most powerful, closely followed by EE_B subnets. Not only do the EE_D and EE_B subnets achieve far greater accuracy than the baseline subnets on all three datasets, they do so while significantly reducing the number of parameters, MACs or network latency (depending on the additional objective). For example, on Tiny ImageNet, while the most accurate high-tradeoff baseline subnets achieve 43,45% and 4,045 M parameters, 42,00% and 17,006 M MACs, and 43,01% and 5,160 ms latency for the three optimisations, the most accurate high-tradeoff EE_B subnets achieve 51,30% and 0.893 M parameters, 50,09% and 14,871 M MACs, and 50,40% and 2,825 ms latency respectively. If instead a similar accuracy to the best baseline subnet is used as a reference, some EE_D or EE_B subnets manage to reduce the number of parameters by up to 10 times, the number of MACs by up to 5 times and the latency by more than half. The difference in pure accuracy is also significant. In

addition, SE_D subnets consistently outperformed baseline subnets in terms of accuracy for a similar number of parameters, MACs or network latency.

Parallel subnets, however, show a different behaviour. SE_P and SE_DP subnets are mostly worse than the baseline subnets. The addition of early exits to their supernets leads to some decidedly more accurate EE_P and EE_DP subnets, which on Tiny ImageNet can beat the baseline subnets in both accuracy and objective scores, but the same isn't true on the CIFAR datasets. In general, the parallel subnets returned by NATv2 seem to be at the extremes, either being quite accurate but with very high objective scores, or with very low objective scores but also poor accuracy.

The introduction of the new archive management systems in NATv2, in the second experimental phase, mitigated some of these problems with parallel networks. Although their final accuracy wasn't very affected by this change, objective scores were generally lower. Nevertheless, parallel subnets were no match for EE_B and EE_D subnets. Given the limited computing resources available and the long time required to run each experiment, parallel networks were mostly abandoned at this point as they didn't meet expectations.

Moving on, in this first phase of NATv2 testing, experiments were also carried out on datasets other than the three mentioned above, and the results obtained are shown in the plots in Figure 6.8. The trend highlighted above continues to hold true regardless of the target dataset and objectives, with the EE_D and EE_B subnets dominating the baseline subnets, and with SE_D subnets also being more accurate. It can be seen that the greater the number of dataset classes, the greater the differences in accuracy between the baseline and new subnets. The consequences of using as a starting point for NATv2 supernets trained on the small Tiny ImageNet images rather than on ImageNet are particularly evident here. In fact, for datasets characterised by very small training sets, many classes and large images, such as FGVC-Aircraft and Stanford Cars, the accuracy is low regardless of the initial supernet. Even in these cases, the baseline subnets remain the worst of the bunch.

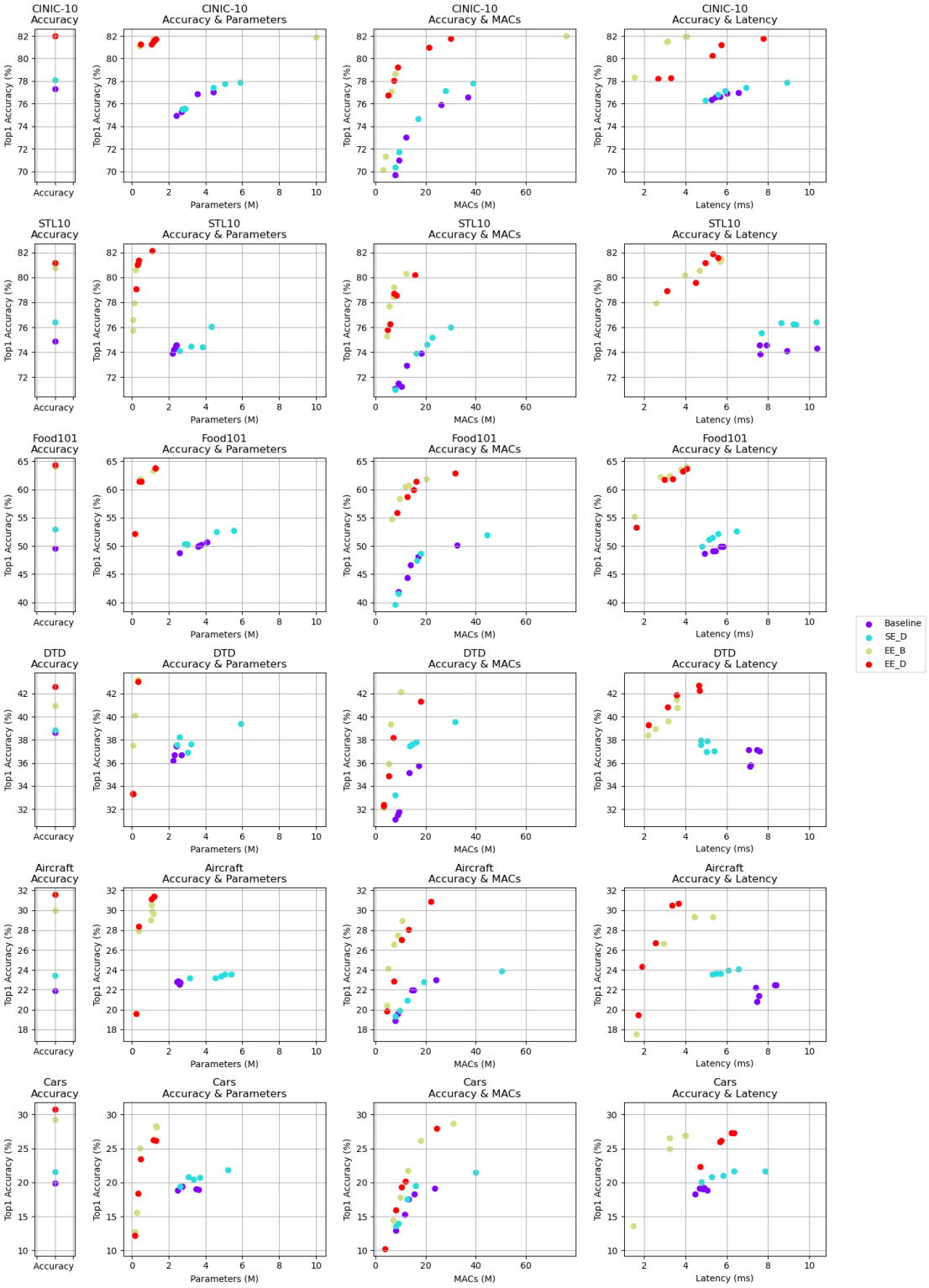


Figure 6.8: Results of the first NATv2 experimental phase on the other distastes. Parallel supernets were discarded. Rows show the results obtained on different datasets. Columns show the results obtained for different target objectives. For bi-objective optimisations, the closer a result is to the top left-hand corner of the plot, the better it is.

We now proceed to compare and discuss the differences in the results obtained throughout the various NATv2 experimental phases. The results were collected for the CIFAR-10, CIFAR-100 and Tiny ImageNet datasets, and for the OFAMobileNetV3, SE_D_OFAMobileNetV3, EE_B_OFAMobileNetV3 and EE_D_OFAMobileNetV3 supernets. Those related to the CIFAR-100 dataset are shown in Figure 6.9. The results shown in the plots correspond to the scores achieved by the subnets as found by NATv2, meaning that the subnets shown for phase four have not yet undergone the post-processing step. The effect of applying post-processing to that set of subnets is discussed below.

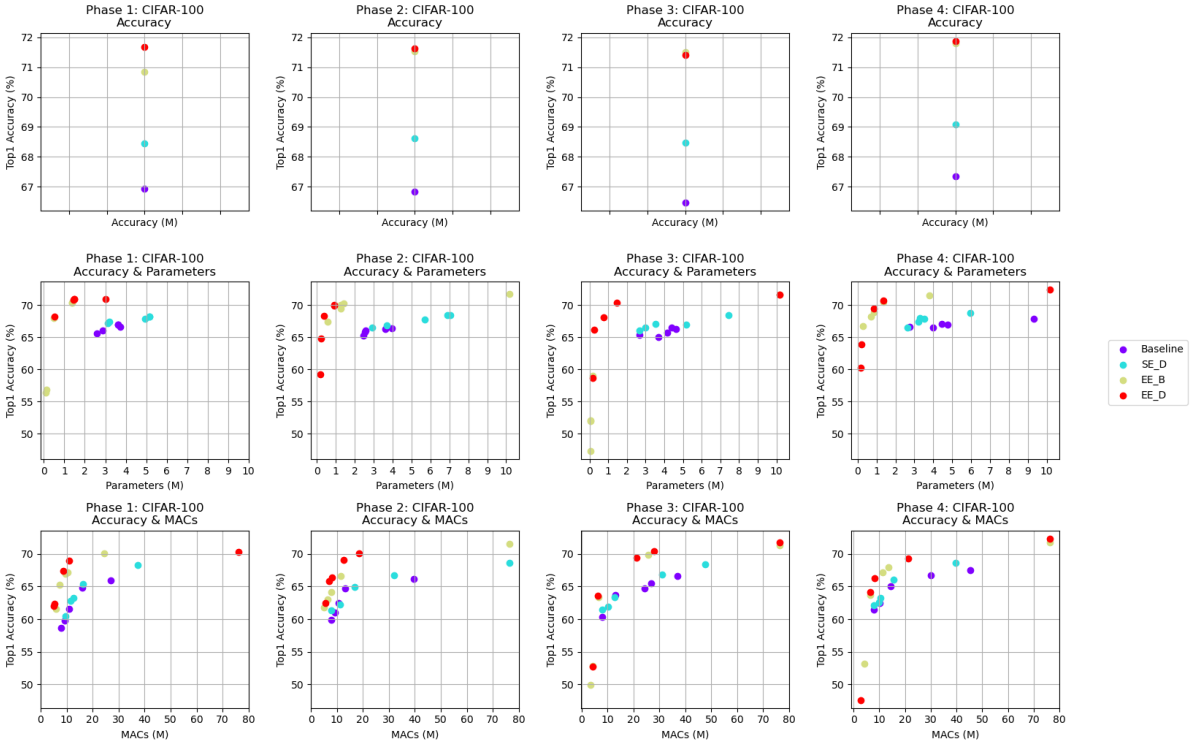
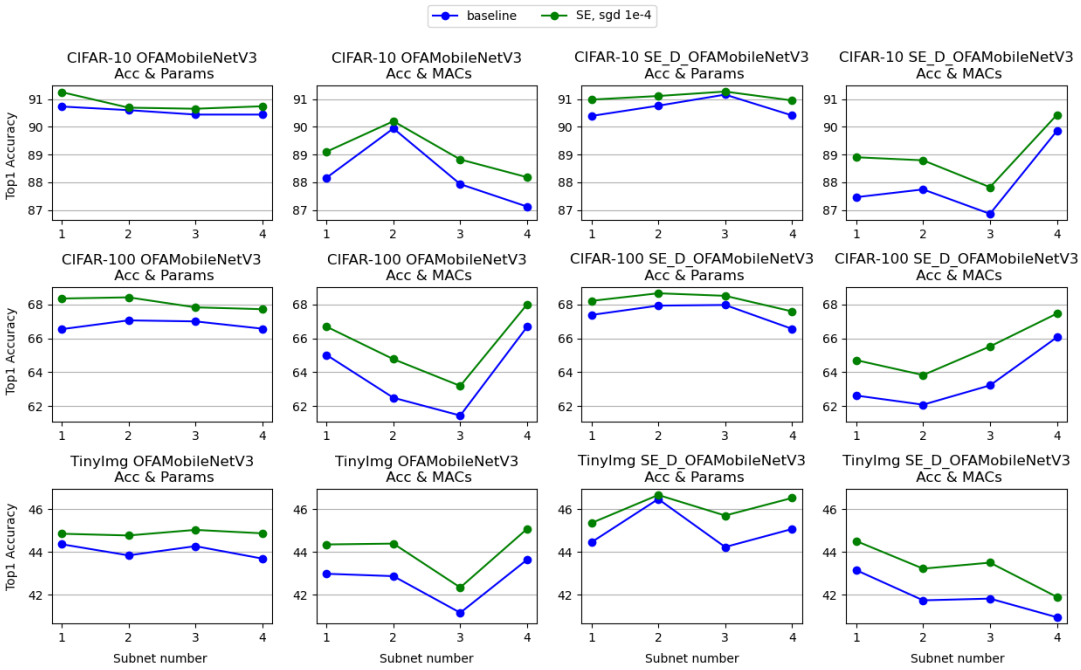


Figure 6.9: Results on CIFAR-100 across the NATv2 experimental phases. Rows show the results obtained for different target objectives. Columns show the results obtained in the different phases. For bi-objective optimisations, the closer a result is to the top left-hand corner of the plot, the better it is.

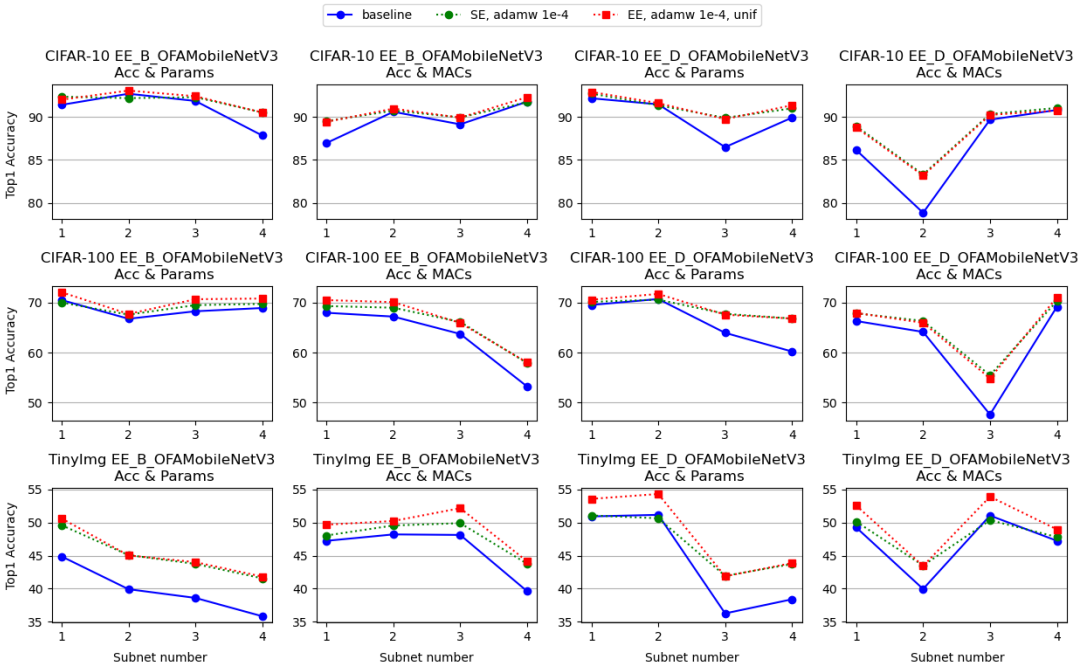
Firstly, it can be seen that regardless of the experimental phase, the baseline subnets are still the worst performing, so algorithmic changes do not affect the already established rankings. Compared to the first phase, the modifications introduced in the second phase, in the case where accuracy is the only objective to be optimised, seems to lead to better subnets especially when the number classes is larger, as in Tiny ImageNet. On the contrary, in the case of bi-objective optimisations, the greatest improvements are obtained on the CIFAR datasets. The introduction of the pre-processing step in the third phase leads to find more accurate subnets than those found in the second phase for bi-objective

optimisations, but these subnets also tend to be more spread out in terms of the secondary objective, so this is a tradeoff. The subnets obtained in phase four are mostly an improvement over those obtained in phase three, thanks to the use of better supernets as starting points. These same subnets were then post-processed using the optimisation strategies identified in 6.3.2. As in that study, the application of the post-processing step, in both its forms, led to improvements in accuracy across the board. For the EE_D and EE_B subnets, it is interesting to see that the more classes the dataset to which the supernet was adapted has, the more frequently the ensemble post-processing method beats its standard counterpart. Figure 6.10a and Figure 6.10b show, for each set of subnets obtained by running the final set of NATv2 experiments on a given supernet for a set of objectives, the accuracy scores originally obtained by the subnets, together with the accuracy obtained after applying both post-processing methods (where applicable).

Finally, the set of subnets found within the baseline OFAMobileNetV3 supernet after the first set of NATv2 experiments, i.e. the one most similar to NAT, and the subnets found by NATv2 during the final experimental phase (this time directly including the post-processing step) can be compared. The results for bi-objective optimisations are shown in Figure 6.11. The EE_B and EE_D subnets are the best ones, so their results are shown alongside those of the baseline subnets. For a fairer comparison, these final EE_B and EE_D subnets underwent the standard post-processing, so their number of parameters and MACs wasn't altered by any branch re-introduction operation. It is clear from the figure that the subnets that can be found using the full NATv2 procedure are significantly superior to the baseline subnets. If, on the other hand, we are willing to compromise on the second objective, then the use of AEP-based post-processing also becomes an option. In this case, where accuracy is the main concern, comparing only the most accurate networks from the two groups, i.e. the phase one baseline subnets group and the full NATv2 subnets group, the full version of NATv2 leads to jumps in accuracy from 90.73% to 93.06% on CIFAR-10, from 66.93% to 72.03% on CIFAR-100 and from 43.45% to 54.31% on Tiny ImageNet, for the same bi-objective optimisation as above.



(a) Post-processing on subnets derived from single-exit supernets



(b) Post-processing on subnets derived from early-exit supernets

Figure 6.10: The results of the application of the post-processing step to the subnets found by the final set of NATv2 experiments. Results in green represent the use of the standard post-processing method. Results in red represent the used of the ensemble post-processing method. The optimisation strategies adopted are shown in the legends.

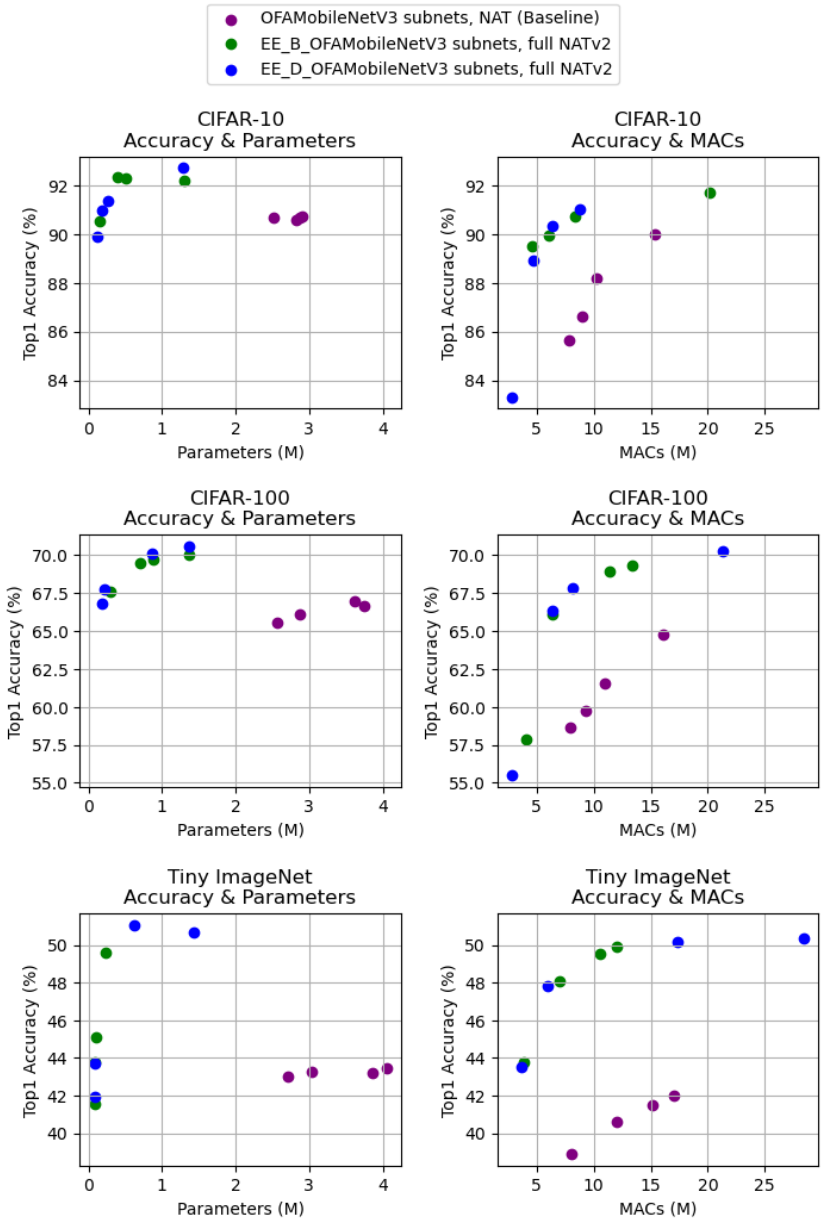


Figure 6.11: Results of the baseline subnets obtained after running NATv2 in the first experimental phase, the closest to the original NAT, compared to the results of the best sets of subnets (the EE_B and EE_D subnets) obtained using the complete version of NATv2, that of the fourth experimental phase. Different rows show the results obtained on different datasets. Different columns show the results obtained for different target objectives. Results are shown for the bi-objective optimisations "accuracy & parameters" and "accuracy & MACs". The closer a result is to the top left-hand corner of the plot, the better it is.

6.4. Conclusion

In this chapter we have examined NATv2, the extension of NAT based on OFAv2 and that takes advantage of AEP, and ultimately demonstrated its ability to outperform the original version of NAT. In particular, it has been shown that changes to the architectural design of the supernet, as well as changes at the algorithmic level, can improve performance. Between the two, the changes related to the use of new supernet architectures, in particular the introduction of early exits, and their associated search spaces, helped to achieve by far the largest improvements. Of the major algorithmic changes, the introduction of the post-processing step, which allows further refinement of the returned subnets, proved to be the best addition.

Although NATv2 was designed to find the best tradeoff subnets with a single exit to allow fairer performance comparisons with the original NAT, in a future study NATv2 could be extended to allow searching for subnets with multiple exits, a possibility that was only briefly explored in this work in the context of the AEP-based post-processing.

7 | Conclusions and Future Developments

The journey recounted in this thesis, which began with AEP and continued with OFAv2, has now come to an end with NATv2. In it, we demonstrated how the introduction of early exits into the structure of convolutional neural networks, and the use of the information they provide through the weighted ensemble and pruning phases of the AEP technique, can result in networks that are both smaller and more accurate at classifying images than the basic single-exit models could be when trained under the same initial conditions. The fact that AEP doesn't rely on any special training tricks or significantly increase training time could make it an important step towards optimising the efficiency of neural architectures in general. Moving on to OFAv2, we proved how architectural changes to the OFA supernet, in particular the introduction of early exits and to a lesser extent that of skip connections, as well as a number of algorithmic improvements, most notably Extended Progressive Shrinking, but also ensemble-KD and the progressive extraction of the teacher network, can contribute to the creation of supernets from which much more accurate subnets can be extracted. Finally, NATv2 was the culmination of this thesis, combining the results and techniques from the other two projects with a series of modifications and additions to the original NAT algorithm. The revised algorithm has been shown to be capable of finding subnets, regardless of the target dataset, that are significantly better than those obtained by combining the basic supernet with the original algorithm.

Despite the positive and overall encouraging results obtained in each of the three works presented, much could still be done to further improve them. In particular, modifications already planned for the second version of AEP include a revised optimisation method to jointly improve network weights and losses weights, the use of the diversity loss, the introduction of a search process to optimise output weights and, finally, support for multi-objective evaluation in the pruning phase. For OFAv2, a study analysing the performance variations associated with introducing a wider variety of potentially more complex blocks into parallel networks may reveal combinations that produce significantly more accurate

networks than those obtained in the thesis. The need to use very simple blocks due to time and computational constraints has probably hurt performance more than it has helped. However, this does not mean that parallel networks do not work in general. With respect to NATv2, the most interesting extension to its current capabilities might be to allow multi-exit subnet searches, a possibility that has only been briefly and indirectly explored in the context of the AEP-based post-processing. It would also be very interesting to run OFAv2 and NATv2 on ImageNet, with the explicit intention of beating the state of the art results.

Bibliography

[1] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

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

[4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

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

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

[7] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1314–1324.

[8] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

[9] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A convnet for the 2020s," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 11 976–11 986.

[10] Z. Lu, G. Sreekumar, E. Goodman, W. Banzhaf, K. Deb, and V. N. Boddeti, "Neural architecture transfer," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 9, pp. 2971–2989, 2021.

[11] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," *arXiv preprint arXiv:1908.09791*, 2019.

[12] Y. Le and X. Yang, "Tiny imagenet visual recognition challenge," *CS 231N*, vol. 7, no. 7, p. 3, 2015.

[13] S. Sarti, E. Lomurno, and M. Matteucci, "Anticipate, ensemble and prune: Improving convolutional neural networks via aggregated early exits," *arXiv preprint arXiv:2301.12168*, 2023.

[14] S. Sarti, E. Lomurno, A. Falanti, and M. Matteucci, "Enhancing once-for-all: A study on parallel blocks, skip connections and early exits," *arXiv preprint arXiv:2302.01888*, 2023.

[15] J. Fang, Y. Sun, K. Peng, Q. Zhang, Y. Li, W. Liu, and X. Wang, "Fast neural network adaptation via parameter remapping and architecture search," *arXiv preprint arXiv:2001.02525*, 2020.

[16] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[17] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: solving problems with box constraints," *IEEE transactions on evolutionary computation*, vol. 18, no. 4, pp. 577–601, 2013.

[18] I. Das and J. E. Dennis, "Normal-boundary intersection: A new method for generating the pareto surface in nonlinear multicriteria optimization problems," *SIAM journal on optimization*, vol. 8, no. 3, pp. 631–657, 1998.

[19] "Depthwise separable convolution," https://eli.thegreenplace.net/2018/depthwise-separable-convolutions-for-machine-learning.

[20] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2020.

[21] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, no. 6, pp. 1789–1819, 2021.

[22] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang, "A comprehensive survey of neural architecture search: Challenges and solutions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 4, pp. 1–34, 2021.

[23] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[24] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," *arXiv preprint arXiv:1611.02167*, 2016.

[25] L. Xie and A. Yuille, "Genetic cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 1379–1388.

[26] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the aaai conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 4780–4789.

[27] X. Zhou, A. K. Qin, M. Gong, and K. C. Tan, "A survey on evolutionary construction of deep neural networks," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 5, pp. 894–912, 2021.

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

[29] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2820–2828.

[30] Y. Shu, W. Wang, and S. Cai, "Understanding architectures learnt by cell-based neural architecture search," *arXiv preprint arXiv:1909.09569*, 2019.

[31] R. Shin, C. Packer, and D. Song, "Differentiable neural network architecture search," 2018.

[32] T. Chen, I. Goodfellow, and J. Shlens, "Net2net: Accelerating learning via knowledge transfer," *arXiv preprint arXiv:1511.05641*, 2015.

[33] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search

by network transformation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[34] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *International conference on machine learning*. PMLR, 2018, pp. 4095–4104.

[35] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *International conference on machine learning*. PMLR, 2018, pp. 550–559.

[36] Y. Benyahia, K. Yu, K. B. Smires, M. Jaggi, A. C. Davison, M. Salzmann, and C. Musat, "Overcoming multi-model forgetting," in *International Conference on Machine Learning*. PMLR, 2019, pp. 594–603.

[37] M. Zhang, H. Li, S. Pan, X. Chang, and S. Su, "Overcoming multi-model forgetting in one-shot nas with diversity maximization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 7809–7818.

[38] T. Domhan, J. T. Springenberg, and F. Hutter, "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves," in *Twenty-fourth international joint conference on artificial intelligence*, 2015.

[39] B. Baker, O. Gupta, R. Raskar, and N. Naik, "Accelerating neural architecture search using performance prediction," *arXiv preprint arXiv:1705.10823*, 2017.

[40] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 19–34.

[41] Z. Lu, K. Deb, E. Goodman, W. Banzhaf, and V. N. Boddeti, "Nsganetv2: Evolutionary multi-objective surrogate-assisted neural architecture search," in *European Conference on Computer Vision*. Springer, 2020, pp. 35–51.

[42] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Smash: one-shot model architecture search through hypernetworks," *arXiv preprint arXiv:1708.05344*, 2017.

[43] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[44] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[45] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[46] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[47] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

[48] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[49] A. Krizhevsky and G. Hinton, "Convolutional deep belief networks on cifar-10," *Unpublished manuscript*, vol. 40, no. 7, pp. 1–9, 2010.

[50] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam, "Netadapt: Platform-aware neural network adaptation for mobile applications," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 285–300.

[51] P. Panda, A. Sengupta, and K. Roy, "Conditional deep learning for energy-efficient and enhanced pattern recognition," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 475–480.

[52] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2016, pp. 2464–2469.

[53] M. Wang, J. Mo, J. Lin, Z. Wang, and L. Du, "Dynexit: A dynamic early-exit strategy for deep residual networks," in *2019 IEEE International Workshop on Signal Processing Systems (SiPS)*. IEEE, 2019, pp. 178–183.

[54] R. G. Pacheco, K. Bochie, M. S. Gilbert, R. S. Couto, and M. E. M. Campista, "Towards edge computing using early-exit convolutional neural networks," *Information*, vol. 12, no. 10, p. 431, 2021.

[55] L. Qendro and C. Mascolo, "Towards adversarial robustness with early exit ensembles," in *2022 44th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*. IEEE, 2022, pp. 313–316.

[56] M. Wołczyk, B. Wójcik, K. Bałazy, I. T. Podolak, J. Tabor, M. Śmieja, and T. Trzcinski, "Zero time waste: Recycling predictions in early exit neural networks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 2516–2528, 2021.

[57] T. Sun, Y. Zhou, X. Liu, X. Zhang, H. Jiang, Z. Cao, X. Huang, and X. Qiu, "Early exiting with ensemble internal classifiers," *arXiv preprint arXiv:2105.13792*, 2021.

[58] S. Scardapane, D. Comminiello, M. Scarpiniti, E. Baccarelli, and A. Uncini, "Differentiable branching in deep networks for fast inference," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 4167–4171.

[59] S. Scardapane, M. Scarpiniti, E. Baccarelli, and A. Uncini, "Why should we add early exits to neural networks?" *Cognitive Computation*, vol. 12, no. 5, pp. 954–966, 2020.

[60] L. Qendro, A. Campbell, P. Lio, and C. Mascolo, "Early exit ensembles for uncertainty quantification," in *Machine Learning for Health*. PMLR, 2021, pp. 181–195.

[61] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu, "Deeply-supervised nets," in *Artificial intelligence and statistics*. PMLR, 2015, pp. 562–570.

[62] K. Hirose, S. Takamaeda-Yamazaki, J. Yu, and M. Motomura, "Selective fine-tuning on a classifier ensemble: Realizing adaptive neural networks with a diversified multi-exit architecture," *IEEE Access*, vol. 9, pp. 6179–6187, 2020.

[63] Z. Fei, X. Yan, S. Wang, and Q. Tian, "Deecap: Dynamic early exiting for efficient image captioning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 12 216–12 226.

[64] A. Campbell, L. Qendro, P. Liò, and C. Mascolo, "Robust and efficient uncertainty aware biosignal classification via early exit ensembles," in *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2022, pp. 3998–4002.

[65] H. Lee and J.-S. Lee, "Students are the best teacher: Exit-ensemble distillation with multi-exits," *arXiv preprint arXiv:2104.00299*, 2021.

[66] G. Hinton, O. Vinyals, J. Dean *et al.*, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, vol. 2, no. 7, 2015.

[67] L. N. Darlow, E. J. Crowley, A. Antoniou, and A. J. Storkey, "Cinic-10 is not imagenet or cifar-10," *arXiv preprint arXiv:1810.03505*, 2018.

[68] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[69] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 215–223.

[70] L. Bossard, M. Guillaumin, and L. Van Gool, "Food-101–mining discriminative components with random forests," in *European conference on computer vision*. Springer, 2014, pp. 446–461.

[71] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3d object representations for fine-grained categorization," in *Proceedings of the IEEE international conference on computer vision workshops*, 2013, pp. 554–561.

[72] S. Maji, E. Rahtu, J. Kannala, M. Blaschko, and A. Vedaldi, "Fine-grained visual classification of aircraft," *arXiv preprint arXiv:1306.5151*, 2013.

[73] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi, "Describing textures in the wild," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 3606–3613.

[74] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. Jawahar, "Cats and dogs," in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 3498–3505.

[75] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*. IEEE, 2008, pp. 722–729.

[76] A. G. Bors, "Introduction of the radial basis function (rbf) networks," in *Online symposium for electronics engineers*, 2001.

[77] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, 2017.

[78] K. Pearson, "Note on Regression and Inheritance in the Case of Two Parents," *Proceedings of the Royal Society of London Series I*, vol. 58, pp. 240–242, Jan. 1895.

[79] C. Spearman, "The proof and measurement of association between two things." 1961.

[80] M. G. Kendall, "A new measure of rank correlation," *Biometrika*, vol. 30, no. 1/2, pp. 81–93, 1938.

[81] P. Helber, B. Bischke, A. Dengel, and D. Borth, "Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 12, no. 7, pp. 2217–2226, 2019.

[82] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[83] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural networks*, vol. 32, pp. 323–332, 2012.

[84] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[85] P. Ramachandran, N. Parmar, A. Vaswani, I. Bello, A. Levskaya, and J. Shlens, "Stand-alone self-attention in vision models," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[86] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[87] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.

[88] Y. Sun, H. Wang, B. Xue, Y. Jin, G. G. Yen, and M. Zhang, "Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 2, pp. 350–364, 2019.

[89] W.-Y. Loh, "Classification and regression trees," *Wiley interdisciplinary reviews: data mining and knowledge discovery*, vol. 1, no. 1, pp. 14–23, 2011.

[90] C. Williams and C. Rasmussen, "Gaussian processes for regression," *Advances in neural information processing systems*, vol. 8, 1995.

[91] H. Drucker, C. J. Burges, L. Kaufman, A. Smola, and V. Vapnik, "Support vector regression machines," *Advances in neural information processing systems*, vol. 9, 1996.

[92] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.

[93] E. Fix and J. L. Hodges, "Discriminatory analysis. nonparametric discrimination: Consistency properties," *International Statistical Review/Revue Internationale de Statistique*, vol. 57, no. 3, pp. 238–247, 1989.

[94] M. E. Tipping, "Sparse bayesian learning and the relevance vector machine," *Journal of machine learning research*, vol. 1, no. Jun, pp. 211–244, 2001.

[95] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: unbiased boosting with categorical features," *Advances in neural information processing systems*, vol. 31, 2018.

[96] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[97] M. Zhang, J. Lucas, J. Ba, and G. E. Hinton, "Lookahead optimizer: k steps forward, 1 step back," *Advances in neural information processing systems*, vol. 32, 2019.

[98] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, "On the variance of the adaptive learning rate and beyond," *arXiv preprint arXiv:1908.03265*, 2019.

[99] J. Blank and K. Deb, "Pymoo: Multi-objective optimization in python," *IEEE Access*, vol. 8, pp. 89 497–89 509, 2020.

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

Abbreviation	Meaning
AEP	Anticipate, Ensemble and Prune
BN	Batch Normalization
CART	Classification And Regression Tree
CNN	Convolutional Neural Network
CV	Computer Vision
DNN	Deep Neural Network
DT	Decision Tree
EA	Evolutionary Algorithm
ED	Elastic Depth
EE	Early-Exit
EH	Elastic Height
EKS	Elastic Kernel Size
EL	Elastic Level
ER	Elastic Resolution
EW	Elastic Width
E2EPP	End-to-End random forest-based Performance Predictor
FFNN	Feed-Forward Neural Network
FLOPS	Floating Point Operations
FT	Fine-Tuning
GA	Genetic Algorithm
GAP	Global Average Pooling
GP	Gaussian Process
Hswish	Hard Swish
KD	Knowledge Distillation

Abbreviation	Meaning
KNN	K-Nearest Neighbors
LR	Learning Rate
LGBM	Light Gradient Boosting Machine
MACs	Multiply-Accumulate operations
MLP	Multi-Layer Perceptron
MOEA	Multi-Objective Evolutionary Algorithm
NAS	Neural Architecture Search
NAT	Neural Architecture Transfer
NLP	Natural Language Processing
NN	Neural Network
NSGA	Non-dominated Sorting Genetic Algorithm
OFA	Once-For-All
PS	Progressive Shrinking
RBF	Radial Basis Function
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RMSE	Root Mean Square Error
RNN	Recursive Neural Network
SaE	Squeeze and Excitation
SE	Single-Exit
SOTA	State Of The Art
SVR	Support Vector Regression
TL	Transfer Learning
WM	Width Multiplier

Acknowledgements

First of all, I would like to express my sincere appreciation to my supervisor, Eugenio, for his constant support over the past year. Your invaluable feedback and ideas have been instrumental in improving the quality of my work, and your guidance has helped me to develop my research skills and make meaningful contributions to the scientific community.

I would also like to thank Professor Matteucci for inspiring me to pursue this path and for giving me the opportunity to work on this project under his expert guidance.

Finally, I would like to express my deepest gratitude to my family and closest friends for the unwavering love they have shown me throughout my academic journey. It was your encouragement and emotional support that kept me going through all the low points, when my faith in myself was completely shattered; I could not have done it without you.

This is just the beginning, I will make you all proud of me, that's a promise.