Recherche d'architecture de réseaux de neurones pour la classification extrême et dans un contexte d'apprentissage partiellement étiqueté

Loïc Pauletto

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

École doctorale : MSTII - Mathématiques, Sciences et technologies de l'information, Informatique
Spécialité : Informatique
Unité de recherche : Laboratoire d'Informatique de Grenoble

Recherche d'architecture de réseaux de neurones pour la classification extrême et dans un contexte d'apprentissage partiellement étiqueté

Neural network architecture search for extreme classification and in a partially labeled learning context

Présentée par :

Loïc PAULETTO

Direction de thèse :

Massih-Reza AMINI — Directeur de thèse
Professeur, Université Grenoble Alpes

Rapporteurs :

HICHEM SAHBI
Chargé de recherche HDR, CNRS DELEGATION PARIS CENTRE
STEPHANE CANU
Professeur des Universités, INSA ROUEN

Thèse soutenue publiquement le **4 juillet 2022**, devant le jury composé de :

MASSIH-REZA AMINI — Directeur de thèse
Professeur des Universités, UNIVERSITE GRENOBLE ALPES
HICHEM SAHBI — Rapporteur
Chargé de recherche HDR, CNRS DELEGATION PARIS CENTRE
STEPHANE CANU — Rapporteur
Professeur des Universités, INSA ROUEN
GEORGES QUENOT — Examinateur
Directeur de recherche, CNRS DELEGATION ALPES
JAKOB VERBEEK — Examinateur
Ingénieur HDR, META PLATFORMS
PATRICK REIGNIER — Président
Professeur des Universités, UNIVERSITE GRENOBLE ALPES

Invités :

NICOLAS WINCKLER
Ingénieur, Atos SE, Échirolles

Abstract

Deep learning applications are rapidly expanding and show no signs of slowing down. Neural network topologies are becoming larger and more complex for challenging real-life problems. This increased complexity necessitates more time and expertise from professionals, as well as a significant financial investment for AI companies. Neural Architecture Search is a novel Machine Learning paradigm that seeks to determine the best NN architecture for a given problem. NAS techniques, on the other hand, have only been studied and developed in limited, well-defined Machine Learning problems, which are not representative of all existing ML scenarios. This thesis focuses on the research and development of the NAS approaches for new tasks, as well as a new learning framework that is more relevant to real-world applications. We suggested using a neuro-evolutionary NAS framework to solve the extreme multi-label classification challenge in particular. We combined convolution and recurrent networks to provide a more appropriate space search for this assignment. On several datasets, we evaluate the performance of the searched network. We also looked at the challenge of reconstructing an RSSI map, which is a more difficult process due to the lack of input data and the fact that it is only partially annotated. In this way, we provide a system for semantic segmentation task dynamic architecture search with a minimal number of annotated samples. We investigated multiple semi-supervised learning algorithms in this framework to see which one was the most successful at using unlabeled samples. We looked at a number of strategies, including "traditional" and "new" semi-supervision approaches, as well as self-supervision approaches.

Résumé

Les applications d'apprentissage profond se développent rapidement et ne montrent aucun signe de ralentissement. Les topologies des réseaux neuronaux deviennent de plus en plus grandes et complexes pour résoudre les problèmes de la vie réelle. Cette complexité accrue nécessite plus de temps et d'expertise de la part des professionnels, ainsi qu'un investissement financier important pour les entreprises d'IA. La recherche d'architecture neuronale (RAN) est un nouveau paradigme d'apprentissage automatique qui cherche à déterminer la meilleure architecture de réseau neuronal pour un problème donné. Les techniques de RNA, d'autre part, n'ont été étudiées et développées que dans des problèmes d'apprentissage automatique limités et bien définis, qui ne sont pas représentatifs de tous les scénarios d'apprentissage automatique existants. Cette thèse se concentre sur la recherche et le développement des approches RAN pour de nouvelles tâches ainsi que sur un nouveau cadre d'apprentissage qui est plus pertinent pour les applications du monde réel. Nous avons proposé d'utiliser un cadre RAN neuro-évolutif pour résoudre le défi extrême de la classification multi-label en particulier. Nous avons combiné des réseaux de convolution et récurrents pour fournir une recherche spatiale plus appropriée à cette tâche. Sur plusieurs jeux de données, nous évaluons la performance du réseau recherché. Nous avons également examiné le défi de la reconstruction d'une carte RSSI, qui est un processus plus difficile en raison du manque de données d'entrée (c'est-à-dire données partiellement annotée). De cette façon, nous fournissons un système de recherche d'architecture dynamique pour les tâches de segmentation sémantique avec un nombre minimal d'échantillons annotés. Nous avons étudié plusieurs algorithmes d'apprentissage semi-supervisé dans ce cadre afin de déterminer celui qui réussit le mieux à utiliser des échantillons non étiquetés. Nous avons examiné un certain nombre de stratégies, y compris des approches de semi-supervision "traditionnelles" et "nouvelles", ainsi que des approches d'auto-supervision.

Contents

1 Introduction

This first chapter will provide a general overview of deep learning, neural networks, and its potential applications and evolutions. The main objectives of this thesis will next be presented. Finally, the thesis outline will be presented, along with a brief summary of each chapter.

Contents

1.1 Overview

Deep learning has grown rapidly in recent years and has become very popular for achieving various tasks in numerous areas. Among these tasks we find game reasoning (Silver et al., 2016), language translation (Bahdanau et al., 2015), natural language processing (Devlin et al., 2019a), speech recognition (Hinton et al., 2012; Zhang et al., 2017) or image identification (Krizhevsky et al., 2012). This large field of application is mainly due to the automatization process of features engineering. Indeed, this process is no longer done by "hand", but by feature extractors that are driven in specific ways for each task. This democratization, as well as this automation, is partly due to the increase in the number of available architectures, all of which require architectures that are increasingly complex to design manually. This has led many researchers to conduct studies in this direction and thus to propose new architectures for a whole range of applications : (He et al., 2016; Chen et al., 2015; Chen et al., 2017; Szegedy et al., 2015; Krizhevsky et al., 2012; Howard et al., 2017; Zhang et al., 2018b; Ronneberger et al., 2015; Vaswani et al., 2017a; Hu et al., 2018a; Yu et al., 2021b; Dosovitskiy et al., 2021; Liu et al., 2022)

Among the large number of available tasks, one acts as a reference, this is the ImageNet image classification (Deng et al., 2009). This challenge consists of a large data set, of the same name, in which objects must be either classified, detected or located according to the chosen task. Being a major axis in the field of deep learning for computer vision, this challenge has received a lot of studies on the architecture in order to obtain each time a better performance. In the early days of modern deep learning research, the AlexNet (Krizhevsky et al., 2012) network established early benchmark results in 2012 by achieving a top 5 accuracy at 15.3%. The proposed architecture concept was then improved over time, in order to be ever more efficient. This has given in 2014 networks like VGG (Simonyan and Zisserman, 2015), Inception-V1 (Szegedy et al., 2015), the still widely used ResNet (He et al., 2016) or more recently SENets (Hu et al., 2018b), which almost halves the error rate compared to previous performance. Figure 1.1 illustrates the improvement over years. If we look at them from a high level, all these networks are similar, and have a similar construction which consists of a succession of convolution layers with different kernel sizes, pooling operations or fully connected. Figure 1.2 illustrates the structures of some of these networks. However, if we take a closer look, most recent structures uses much more complex blocks. Those blocks are like mini deep neural network with multiples operations, branches and data flow, with a higher complexity than simply stacking convolution operations.

However, this neural network architecture design process has some drawbacks. One of the most important is that the architecture design phase is a long process, filled with trial and error. It also requires certain human skills as a starting point, like prior knowledge on the domain, data and task. All this represents a blow, which can be a brake for some companies or for the application of neural networks to new domains or applications. The second major drawback is that the potential architectures proposed are limited to the imagination and background of each researcher. One way to solve these two major problems, but also others, would be to automate this design process, by learning an architecture directly from the available data. This part of automatic architecture discovery is a separate field of research, commonly called Neural Architecture Search (NAS).

To be more specific, the NAS is part of a much larger automation domain: AutoML (He et al., 2021). In this super-field, we generally group together all the research that is related to the automation of the machine learning pipeline using data. This pipeline could include the composition of the pre-processing or the choice of the most suitable training procedure. Two other topics of AutoML, which are greatly related to the topic of NAS, are hyperparameters optimization (Yu and Zhu, 2020) (HPO) and meta-learning (Thrun and Pratt, 2012; Hospedales et al., 2020). All these topics can be treated under the same prism of the two-level optimization. Indeed, the choice of hyperparameters, deep neural network structures or parameters of meta-learning can be seen as the

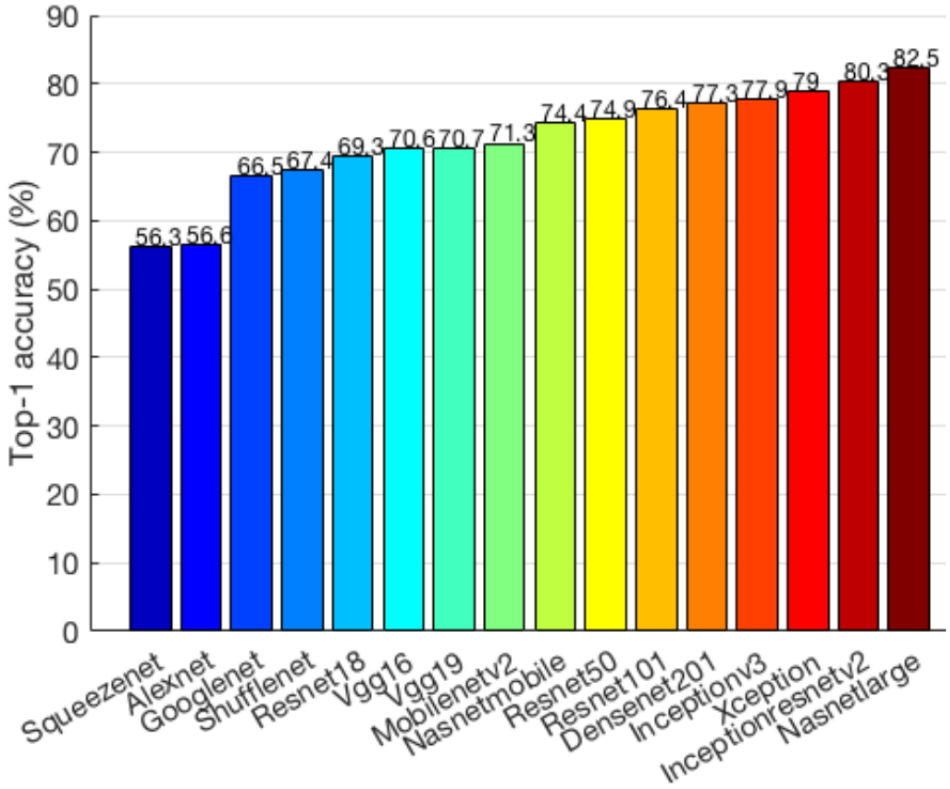


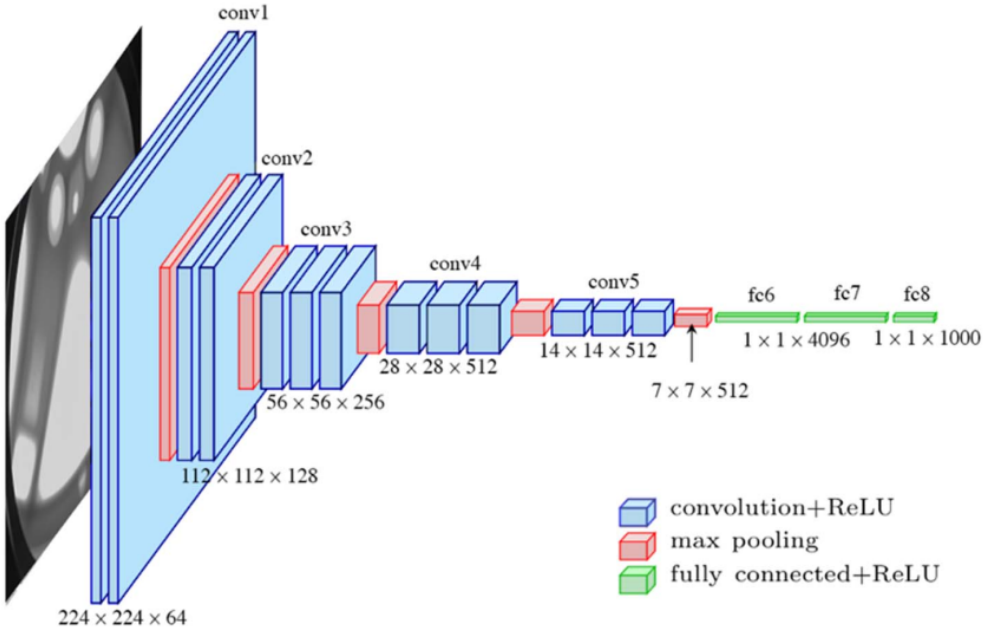
Figure 1.1: Improvement of Top-1 accuracy of various neural network, over years. Illustration from (Zhang and Davison, 2020)

first level of optimization. The second level would be the optimization of the parameters (weights) of the different operations constituting the neural network. Unlike hyperparameter optimization, which typically assumes that optimization is a black box, it is easily transposable to many machine or deep learning algorithms, the NAS methods often go further than black box by using prior knowledge of proposed neural architectures. This usage lead to reduction of computational costs and time compared to black box optimization. These savings are crucial for NAS methods, because evaluating a network requires training, which usually takes several hours or even days.

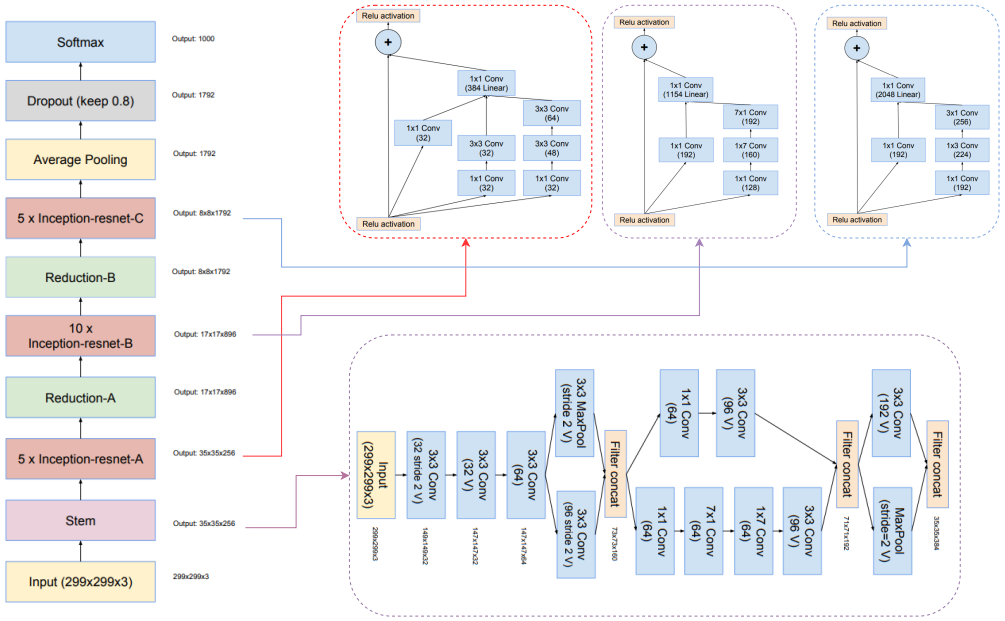
At present, the performance of automatically found architectures has already exceeded the performance of manually designed ones on various tasks. As pictured in Fig. 1.1, NasNet (Zoph et al., 2018) obtains performances exceeding all other architectures for image classification. This situation is similar for many other tasks such as semantic segmentation, object detection and natural language processing. Moreover, this automation will allow the growth of deep learning to new applications, which are impossible today because of budget, time constraints or lack of expert or knowledge in a particular field.

1.2 Thesis objectives

The growing enthusiasm around the NAS topic is accompanied by a number of challenges to be addressed. In this section, we will present and explain the objectives that will be addressed in this thesis. The main challenge we hoped to address with this thesis was the relatively restricted



(a) Illustration of network VGG16. Illustration from (Ferguson et al., 2017)



(b) Illustration of network Inception-Resnet-V2. Constructed from (Szegedy et al., 2017)

Figure 1.2: Illustration of two neural networks, showing the evolution of architectures over the years

application of NAS methods. We have decided to approach this broad spectrum challenge from three distinct perspectives:

1.2.1 Objective 1 : Explore the application to new tasks

The first perspective is the use of the NAS for new tasks. The NAS approaches are researched in a variety of disciplines, but only on a small number of tasks, which frequently return to image classification (Zoph et al., 2018; Pham et al., 2018; Liu et al., 2019b). However, this task is only a tiny part of the vast array of tasks (e.g. several thousands tasks listed on paperswithcode[1] website) that neural networks can be used. Indeed, the majority of research is concentrated on presenting new methods that are always more efficient, with little interest in applying these approaches to new, less researched areas. Our first goal was to examine how NAS approaches adapt and behave on new tasks, in order to determine if it was possible to find equal or even better performing designs on certain aspects than "hand-made" architecture on these new tasks.

1.2.2 Objective 2 : Learning from sparsely annotated data

The second perspective was to investigate NAS techniques in a scenario with a limited amount of labeled data. Indeed, because the annotation process may be time-consuming and tiresome, businesses may be hesitant to engage in it. As a result, there are many real-world applications where the amount of annotated data is limited. However, the majority of the approaches conducted architecture research in a completely supervised environment. Approaches have only recently focussed on architecture search in a context where there are no labels (Liu et al., 2020; Kaplan and Giryes, 2020) but the final architecture is trained in a supervised way.

As a result, we sought to investigate this direction, concentrating on the semi-supervised scenario. In this semi-supervised environment, we intend to investigate the joint search capabilities of the architecture and the parameters.

1.2.3 Objective 3 : Be as efficient as possible

A more transversal objective was to use the most efficient approach that meet our requirements, and produce a network as efficient as possible. Indeed, most of the time, resources are a limiting issue because they are costly to purchase or rent, thus it is critical to make the greatest use of these resources. Furthermore, as environmental challenges become more prevalent, it is critical to lessen the energy impact and, as a result, the environmental footprint that is associated with it.

1.3 Outline and contributions

Based on the questions proposed above, we will follow the following outline in order to introduce the concepts and present the contributions that have been made. In the course of this thesis, several domains were approached (e.g. text, images). An introduction to the different application domains will be given at the beginning of each chapter in order to introduce the concepts.

The first three chapters will be devoted to the introduction as well as the presentation of the related works, after which the various contributions made during this thesis will be outlined.

[1] https://paperswithcode.com/sota

1.3.1 Chapter 2: Neural Networks

To make this text as accessible as possible, this chapter will first reconstruct the history of the neural network, beginning with the initial efforts more than 70 years ago. Then, a brief overview of supervised learning and the gradient descent concept, which serves as the foundation for neural network optimization. Finally, a review of the many forms of neural networks found in the literature. Furthermore, we will delve into further depth on two networks that were "pioneers" and are employed by many ways in their work.

1.3.2 Chapter 3: Introduction to Neural Architecture Search

This chapter will introduce the field of NAS and this various concept, in order to give a complete overview of this constantly evolving field to the researcher. To complete this, we will categorize the body of work into three main parts (search space, search algorithm and evaluation of performance). The first axis is the search space, this space defines the way in which the researched architecture will be represented. Its definition varies according to several parameters, we generally find architectures in chain structure (classic feed-forward networks), or more recently cell-based networks.

Once the search space is defined, it is necessary to choose the way to search in this space, this is the role of the search algorithm or strategies. Among popular algorithms, some of the first research use black box based algorithms such as Bayesian optimization, Reinforcement learning or evolutionary algorithm. Recently, specific design approach have been studied by taking advantage of a search space continuous relaxation and using gradient descent optimization.

Lastly, the estimation of network performance, which is a crucial point in this research. Indeed, the evaluation of a network is a costly operation if it is done too often. To overcome this, some estimation methods have been proposed, among which we find low fidelity estimator or weight sharing.

1.3.3 Chapter 4: Learning from partially annotated data

Data collection and annotation has always been one of the obstacles to the expansion of deep learning. This collection of annotated data is extremely costly both in time and money, so that some methods have proposed to use data directly without annotations or using a small amount of annotated data. Among these methods, we can distinguish for example those that involve a single network or those that require a second network.

Using only one network, we can cite the self-learning technique, which has recently been improved by injecting noise during training phase. Differently, the methods involving two networks rely on the idea of consistency in the predictions and those despite the various perturbations that can be injected. These disturbances can be induced in different forms, for example by the disturbance of the inputs or by the architecture itself. In this chapter, we will provide an introduction to those various techniques that have been proposed.

1.3.4 Chapter 5: XMCNAS : NAS for extreme classification

The architecture search methods were essentially tested on the image domain, where they showed their efficiency and their performances. However, the image domain represents only a fraction of the domains where the use of deep learning would be interesting. Therefore, one of the challenges to address would be the adaptation and testing of certain methods to other domains or tasks. To get into the idea of democratization of NAS, in this chapter we will present the way we have adapted

NAS methods to the text and, more precisely, to the task of extreme classification (XMC). Moreover, we wanted to go further than the simple application of NAS methods on a new domain, we also wanted to understand how the architecture was impacted by each new change and also if all operations that compose it were used in the same way. To achieve this goal, we used and adapted an evolutionary algorithm, which allowed us to do this impact analysis at a fine level.

On various benchmark datasets, we also evaluate and compare the performance of the produced architecture to that of several specific human-designed architecture and methods.

1.3.5 Chapter 6: NAS for RSSI map reconstruction

In the extension of the previous work, we have explored another task, this time more complex. Indeed, because the available data is only partial, the task of RSSI map reconstruction is more difficult. As a consequence, we investigated several NAS methods in order to find the one that would allow us to have the most efficient architecture. We then used a self-learning technique on the unannotated data to optimize the parameters of the model.

The results of the various approaches are then compared using state-of-the-art non-learning interpolation techniques.

1.3.6 Chapter 7: NAS with Partially labeled data for Semantic Segmentation

To go deeper into the exploration of NAS in a regime with a limited amount of annotated data, we are interested in the semantic segmentation task, which is more widespread and has numerous application cases. We analyzed various methods, including traditional semi-supervised and self-supervised approaches, in order to determine what is the most efficient method for utilizing unannotated data.

Then, based on this method, we proposed a method for searching for efficient semantic segmentation architecture in a semi-supervised context.

Part I

Related works

2
Neural Networks

In this chapter, we will recall the fundamentals of neural networks. We will start with a historical review of artificial neurons, and we will describe he basics of supervised learning. Then, we will present the mathematical foundations of learning with the gradient descent algorithm. Finally, we will review neural networks and present some important networks in the community.

Contents

2.1 Introduction

The study of neural networks began with the pioneering work of neurologist Santiago Ramon y Cajal near the end of the nineteenth century. Ramon y Cajal studied a natural neuron and demonstrated how its dendrites convey information to other neurons connected to it. Years later, McCullogh (a neurologist) and Pitts (a logician) devised a formal neuron, which is a mathematical model of a neuron. A formal neuron receives a signal $\mathbf{x} = (x_1, \ldots, x_d)$ in the form of a set of real-valued characteristics and uses a vector of weights $\boldsymbol{w} = (w_1, \ldots, w_d)$ to estimate a weighted sum of these features. This model returns a binary result based on whether the weighted sum is greater than or less than a given threshold b. In their paper (Mcculloch and Pitts, 1943), they showed that this formal neuron is able to learn basic logic operators like the *logical AND* and the *logical OR*. Their study was in line with previous efforts at the time to create a machine with artificial intelligence, and the formal neuron became one of the principals models in this investigation. Several works then tackled the tedious question of how to determine the weights of the formal neuron, enabling one to resolve, basic logical operations. Different strategies have then been proposed for more than a decade to learn these weights, such as the Hebb rule which states that *neurons which wire together, fire together*; however, we must wait for the inspiring work of Frank Rosenblatt, a psychologist, who was the first to successfully use the analogical computer; ENIAC; to efficiently learn these weights.

Rosenblatt, 1958's *Perceptron* algorithm, named in reference to a human perception neuron, was created to detect a given form presented on a 2D board by turning on and off lamps with respect to the others. This was a ground-breaking end-to-end model that had two major innovations that were adopted by other learning models years later. The first point concerns the representation of forms that are not hand-crafted and are acquired from random placement of captures. The second is in relation to the stochastic gradient descent technique, which works by updating the weights sample by sample, by reducing the distance between a misclassified example and the current hyperplane discovered by the formal neuron. This model sparked interest and paved the way for the development of other similar techniques based on stochastic gradient descent to learn the weights of a formal neuron. Researchers from other fields were also interested in the learnability of these models, such as statistician (Novikoff, 1962), who proved the convergence of the perceptron algorithm in the case where there exist a hyperplane that separates the classes. (Minsky and Papert, 1969)'s paper showing that a formal neuron is not able to learn the exclusive OR (XOR), marked a stop to most of these works.

The resuscitation of Neural Networks had to wait until the mid-1980s, when it was discovered that by adding a layer between the input and output of a formal neuron, called the hidden layer, it was feasible to modify the representation space and solve the XOR problem in the new space associated with this new layer. This discovery paved the way for the development of complex models with several hidden layers and even loops between neurons, to solve advanced prediction problems. A Neural Network is defined then as an oriented graph with three sets of neurons; that are in the input, in the output and in the hidden layers. When the number of layers exceeds two, these models are sometimes referred to as *Deep Neural Networks*. The crucial question in this case is how to update the weights that connect the neurons of layers after the last layers of the NN? The answer to this question was given by Rumelhart et al., 1986 who proposed the backpropagation algorithm that is based on the gradient descent and the chain rule to recursively update all weights. This powerful method, which is still employed in the learning of NNs, along with the finding that a NN with a single layer may be a universal approximator, allowed the domain to evolve quickly. However, NN models with complex architectures and a large number of parameters, require a consequent training set to be learned. This feature proved to be a disadvantage for NNs who competed in the Caltech 101 competition[1] in 2004 (Fei-Fei et al., 2004). The goal of this

[1] http://www.vision.caltech.edu/Image_Datasets/Caltech101/

challenge was to build a model to categorize an image into 101 categories using a training set of 3000 images (with around 30 images per class). The winner of this competition was not a NN-based system, sounding the death knell for these models, which were entering their second winter.

The rebirth of Neural networks happened in 2012 with the ILSVRC[2] challenge using a subset of ImageNet[3] with about 1.2 million images categorized in 1000 classes (Russakovsky et al., 2015). The creation of this big collection was combined with the usage of Graphic Processing Units (GPUs), which significantly reduce processing time. The winner of this competition was AlexNet (Krizhevsky et al., 2012), a Neural Network with about 60 million parameters that was trained over six days using two NVIDIA GTX 580 3GB GPUs (vs. 42 days training using CPUs under use at that time). The availability of massive datasets, as well as ongoing technological advancements in the construction of increasingly powerful GPUs, have paved the road for widespread use of Neural Networks across a wide range of applications.

In this chapter, we will briefly describe Neural Networks by first providing a rapid presentation of Supervised Learning framework under which these models were first developed (Section 2.2). Then the gradient descent method that is generally used to train these models in Section 2.3. In Section 2.4, we will finish up with the backpropagation method and a description of the most common Neural-Networks.

2.2 A brief presentation of Supervised learning

A learning algorithm induces a prediction function from a set of *examples*. Each example is a pair *(observation, response)*; and the function returned by the algorithm must make it possible to predict the response associated with a given observation. More precisely, it must make it possible to predict the response associated with *new observations*. The underlying assumption is that the examples are, in one way or another, representative of the prediction problem on which the function will be used.

This is the supervised learning paradigm in which algorithms strive to find the mapping function between an input space $\mathcal{X}$, which is often $\mathcal{X} \subset \mathbb{R}^d$, and an output space $\mathcal{Y}$. In the case where $\mathcal{Y} = \mathbb{R}$ we deal with a regression problem and when $\mathcal{Y}$ is a discrete set, we have a classification problem.

The aim is to identify the mapping f from a set of hypothesis $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$ over a finite set of examples $S = (x_i, y_i)_{1 \leq i \leq m}$, called the training set, that has the lowest probability of error on new observations that do not belong to S.

2.2.1 Central hypothesis and the ERM principle

The fundamental assumption of statistical learning theory is that all examples are *independently generated* by a *fixed, but unknown, probability distribution*. It will be denoted $\mathcal{D}$. The immediate consequence is that for any set of examples S, *examples* (x_i, y_i) *are generated independently according to* $\mathcal{D}$. We then say that S is a sample of i.i.d. (independent and identically distributed) according to $\mathcal{D}$.

Informally, the *identically distributed* assumption defines the notion of the stationarity of the underlying phenomena; that is training examples, as well as future ones, come from a single

[2] https://image-net.org/challenges/LSVRC/2012/
[3] https://www.image-net.org/

source. The *independently distributed* assumption stipulates that each example brings maximum information for learning.

The second fundamental notion in learning is the notion of loss, also called *risk*. Given a prediction function f, the disagreement between the prediction $f(x)$ and the desired output y for a pair (x, y) is measured using a function called the *instantaneous loss* – a function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$. Intuitively, $\ell(f(x), y)$ measures the proximity between the predicted response and the actual one. It is therefore generally a distance on the set $\mathcal{Y}$. In classification, the instantaneous error generally considered is:

$$\ell_c(f(x), y) = \mathbb{1}_{f(x) \neq y},$$

where $\mathbb{1}_\pi$ equals 1 if the predicate π is true and 0 otherwise. In regression, the commonly used instantaneous loss function is the distance on $\mathbb{R}$ defined by the square of the absolute value:

$$\ell_r(f(x), y) = |f(x) - y|^2.$$

In general, other error functions can be used (for example in regression $\ell'_r(f(x), y) = |f(x) - y|$).

Let remind that the objective of a learning algorithm is to choose a function capable of performing good predictions on observations that are not part of the learning set. In other words, the prediction function must be able to *generalize*. As each pair (observation, desired output) is assumed to be generated by $\mathcal{D}$ independently of the other examples, the *generalization error* of a prediction function f is naturally defined by:

$$R(f) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[\ell(f(x), y)] = \int_{\mathcal{X}\times\mathcal{Y}} \ell(f(x), y) d\mathcal{D}(x, y), \tag{2.1}$$

where $\mathbb{E}_{(x,y)\sim\mathcal{D}}[X(x, y)]$ designates the expectation of the random variable X when the couple (x, y) follows the distribution $\mathcal{D}$.

Given a training set S, the *empirical error* defined below is used to measure the performance of a prediction function f on S:

$$R_m(f, S) = \frac{1}{m} \sum_{i=1}^{m} \ell(f(x_i), y_i). \tag{2.2}$$

The motivation for this error is that for a given function f, $R_m(f, S)$ is an *unbiased estimator* of the generalization error of f: indeed, since we assume that the examples (x_i, y_i) of S are independent, the random variables $\ell(f(x_i), y_i)$ can also be considered independent. We then have:

$$\mathbb{E}_{S\sim\mathcal{D}^m} R_m(f, S) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{E}_{S\sim\mathcal{D}^m} \ell(f(x_i), y_i) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{E}_{(x_i,y_i)\sim\mathcal{D}} \ell(f(x_i), y_i) = R(f).$$

From this observation, it is then natural, to determine a prediction function on a training set, to choose in $\mathcal{F}$ a prediction function f_S which minimizes the empirical error. This learning method is called the *Empirical Risk Minimization* (or ERM) principle. An important part of the theory of Vapnik, 1998 aims to study the generalization error of the function f_S according to its empirical error $R_m(f_S, S)$ and the class of functions $\mathcal{F}$ considered.

The central question in learning theory is to determine if the function $f_S \in \mathcal{F}$ found by minimizing the empirical error $R_m(f, S)$ on a training set S will have a good generalization error? In general, the answer to this question is negative.

Thus, there are two desirable characteristics for a learning algorithm: **(1)** the *algorithm must return a function capable of generalizing*, i.e. its learning error must somehow reflect his generalization error. Moreover, **(2)** the objective of an algorithm can be more ambitious: not only the chosen function must generalize, but in addition, if the number of training examples is large, the algorithm must find a function which minimizes the generalization error in the class of functions considered.

2.2.2 Consistency of ERM and the SRM principle

An algorithm with the two previous properties is called *consistent*. The question posed by Vapnik, 1998: considering a learning algorithm minimizing the empirical risk, under what conditions is it consistent? Note that the learning algorithm depends on two factors: the class of functions $\mathcal{F}$ considered in the minimization, and the function chosen when several functions achieve the minimum. We consider algorithms for which this choice is deterministic.

The formal definition of the consistency of the ERM principle, for a given learning problem, is as follows: the following two conditions must be verified by the function returned by the algorithm minimizing the empirical risk:

- $\forall \epsilon > 0; \lim_{m\to\infty} \mathbb{P}(|R(f_S) - \inf_{g\in\mathcal{F}} R(g)| > \epsilon) = 0,$
- $\forall \epsilon > 0; \lim_{m\to\infty} \mathbb{P}(|R_m(f_S, S) - \inf_{g\in\mathcal{F}} R(g)| > \epsilon) = 0,$

remembering that the limit is taken when m, the size of the training set S, tends to infinity.

The study of the consistency of the ERM principle led to the following fundamental result of statistical learning theory (Vapnik, 1998, theorem 2.1): *The ERM principle is consistent if and only if:*

$$\forall \epsilon > 0, \lim_{m\to\infty} \mathbb{P}\left(\sup_{f\in\mathcal{F}} [R(f) - R_m(f, S)] > \epsilon\right) = 0. \tag{2.3}$$

Note that this relation is less constraining than the two sufficient conditions presented above, as it is no longer the absolute value of $R(g) - R_m(g, S)$ which is considered. Classical derivations of (2.3) lead to generalization bounds that take the following form:

$$\forall S, \forall \delta \in]0, 1[; \mathbb{P}\left(R(f) \leq R_m(f, S) + \mathfrak{C}(\mathcal{F}, S) + \sqrt{\frac{\log \frac{1}{\delta}}{2m}}\right) \geq 1 - \delta,$$

where $\mathfrak{C}(\mathcal{F}, S)$ is a measure of the capacity of the class of functions $\mathcal{F}$ that can be estimated over the training set S. The greater the capacity, the more it is possible to obtain a low empirical risk, without guaranteeing a lower generalization error. The difficulty of learning is therefore to achieve a compromise between a low empirical error and a low capacity of the set of functions in order to minimize the generalization error. This compromise is called *Structural Risk Minimization* principle (Vapnik, 1998). This principle can be resumed as follows: considering several sets of functions $\mathcal{F}_1$, ..., $\mathcal{F}_N$, it minimizes the empirical risk in each of these sets separately, then select the prediction function that minimizes the bound on the generalization error.

In practice, the SRM principle is applied by regularizing the minimization of the empirical loss. That is to minimize the empirical error together with a term that penalizes the choice of complex functions. In general, this penalization term is a ℓ_2-norm of the parameters of the function. The minimization of the empirical error with the regularization term then resumes to choose the function $\hat{f} \in \mathcal{F}$ such that:

$$\hat{f} = \text{argmin}_{f\in\mathcal{F}} R_m(f, S) + \lambda \|f\|^2, \tag{2.4}$$

where λ is called the regularization term.

2.3 Gradient Descent Algorithm

Minimization problems related to the ERM or SRM principles are solved using optimization techniques, and their advancement is closely tied to that of the Machine Learning field. Among the various optimization approaches, the Gradient Descent (GD) algorithm is without a doubt the most commonly utilized in ML models. This algorithm finds a (local) minimum of a convex differentiable surrogate function of the (regularized) 0/1 loss by only relying on the information provided by the gradient of the loss function, and it is hence referred to as a first-order optimization process.

The approach is based on the observation that if the loss function, $\hat{\mathcal{L}}$, to be minimized is defined and differentiable in the vicinity of a weight vector, $\boldsymbol{w}^{(t)}$, then the loss reduces as one moves one step $\eta_t \in \mathbb{R}_+$, called the learning rate, away from the actual value of the loss, $\hat{\mathcal{L}}(\boldsymbol{w}^{(t)})$ in a descent direction $\mathbf{p}_t$ defined as $\mathbf{p}_t^\top \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)}) \leq 0$.

At each iteration, if we define the new weight vector $\boldsymbol{w}^{(t+1)}$ as

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta_t \mathbf{p}_t, \tag{2.5}$$

with a small learning rate η_t, we get

$$\hat{\mathcal{L}}(\boldsymbol{w}^{(t+1)}) \leq \hat{\mathcal{L}}(\boldsymbol{w}^{(t)}). \tag{2.6}$$

The decreasing condition above does not guarantee that the loss function will reach a local minimum, or the global minimum, of the loss function if η_t is too small. For each iteration of GD, the following sufficient conditions, known as Wolfe conditions, have been proposed in order to ensure the algorithm convergence. These conditions are depicted in Figure 2.1.

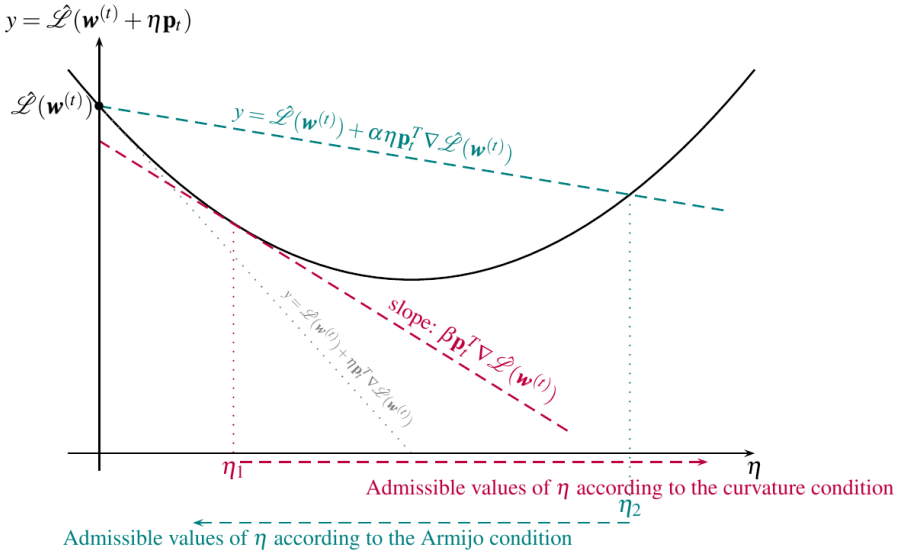


Figure 2.1: Illustration of the Wolfe conditions for a convex function $\hat{\mathcal{L}}(\boldsymbol{w}^{(t)} + \eta \mathbf{p}_t)$ with respect to the learning rate η. $\eta = 0$ corresponds to the actual value of the loss function, and the gray dashed line corresponds to the tangent of the loss at this value. For a given $1 > \alpha > 0$; the line in teal $y = \hat{\mathcal{L}}(\boldsymbol{w}^{(t)}) + \alpha\eta_t \mathbf{p}_t^\top \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)})$ delimits the admissible values for η with respect to the Armijo condition (2.7). The line in purple with the slope $\beta \mathbf{p}_t^\top \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)})$ shows the admissible values of η with respect to the curvature condition (2.8). Admissible values of the learning rate with respect to the Wolfe conditions are in between these two values.

Armijo condition. In relation to the length of the jumps, the decrease in $\hat{\mathcal{L}}$ should not be too small. As a result, for a given $1 > \alpha > 0$:

$$\forall t \in \mathbb{N}^*, \hat{\mathcal{L}}(\boldsymbol{w}^{(t)} + \eta_t \mathbf{p}_t) \leqslant \hat{\mathcal{L}}(\boldsymbol{w}^{(t)}) + \alpha \eta_t \mathbf{p}_t^\top \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)}). \tag{2.7}$$

Curvature condition. Each update should result in a change in the curvature of the loss function. Alternatively, the slope should drop sufficiently; i.e. $\exists \beta \in (\alpha, 1)$ such that

$$\forall t \in \mathbb{N}^*, \mathbf{p}_t^\top \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)} + \eta_t \mathbf{p}_t) \geq \beta \mathbf{p}_t^\top \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)}). \tag{2.8}$$

In practice, the learning rate is determined at each iteration using a line search algorithm. It includes beginning with a high value of the learning rate and repeatedly reducing it by multiplying the current value by a factor $1 > a > 0$ (i.e., *backtracking*) until the Armijo condition is satisfied.

When the loss function is convex and differentiable, and its gradient is Lipschitz continuous, with parameter $L > 0$ defined as

$$\forall \boldsymbol{w}, \boldsymbol{w}'; \|\nabla \hat{\mathcal{L}}(\boldsymbol{w}) - \nabla \hat{\mathcal{L}}(\boldsymbol{w}')\|_2 \leq L \|\boldsymbol{w} - \boldsymbol{w}'\|_2, \tag{2.9}$$

the GD algorithm is ensured to converge to a (local) minimum of the loss function according to the following result.

Theorem 1 (Zoutendijk, 1966)**.** *Let $\hat{\mathcal{L}}$ be a differentiable objective function with a Lipschtiz continuous gradient and lower bounded. Suppose that the GD algorithm generates $(\boldsymbol{w}^{(t)})_{t \in \mathbb{N}}$ defined by $\forall t \in \mathbb{N}, \boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \eta_t \mathbf{p}_t$; where $\mathbf{p}_t$ is a descent direction of $\hat{\mathcal{L}}$ and η_t a learning rate verifying both Wolfe conditions (2.7) and (2.8). By considering the angle θ_t between the descent direction $\mathbf{p}_t$ and the direction of the gradient $\cos(\theta_t) = \frac{\mathbf{p}_t^\top \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)})}{||\nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)})|| \times ||\mathbf{p}_t||}$; the following series is convergent*

$$\sum_t \cos^2(\theta_t) ||\nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)})||^2.$$

2.4 Neural Network : an overview

In this section, we will provide an overview of Neural Networks that we will consider throughout this thesis. As mentioned in the introduction, (Mcculloch and Pitts, 1943)' model perform a weighted sum of the inputs which is then composed with an activation function; H, also called transfer function.

$$f : x \mapsto H(\boldsymbol{w}^\top x + w_0)$$

This operation is illustrated in Figure 2.2. In the case of the Perceptron, the activation function is the indicator function of positive real numbers, also called the *Heaviside* function; which is equal to 1 if the weighted sum is positive and 0 otherwise.

2.4.1 Activation functions

As mentioned in the introduction, a Neural Network is a graph of interconnected nodes in which each node in the input corresponds to a feature of an example and is connected to neurons in the first hidden layer. A hidden layer neurons are linked to a group of neurons that might be from the same hidden layer or from another hidden layer. The last hidden layer neurons are linked to the output layer neurons.

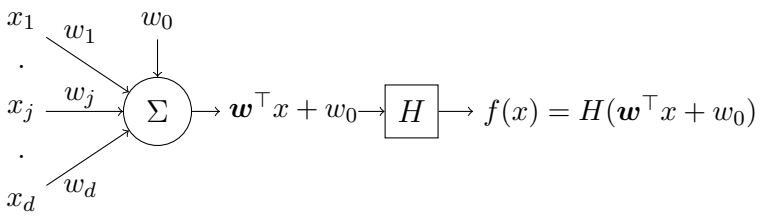


Figure 2.2: Illustration of the operation of a formal neuron whose output is calculated after a composition of an activation function H with a weighted sum of the characteristics with the model weights.

The values of each node in the hidden and output layers are estimated in the same way that they are in the Perceptron model: they are a composition of the values of previous nodes connected to that node by an activation function. The most popular activation functions are:

$$\textbf{ReLu}: \quad H(z) = \begin{cases} z, & \text{if } z \geq 0 \\ 0, & \text{otherwise.} \end{cases}$$

$$\textbf{Sigmoid}: \quad H(z) = \frac{1}{1+e^{-z}}$$

$$\textbf{Hyperbolic Tangent}: \quad H(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

These activation functions are illustrated in Figure 2.3.

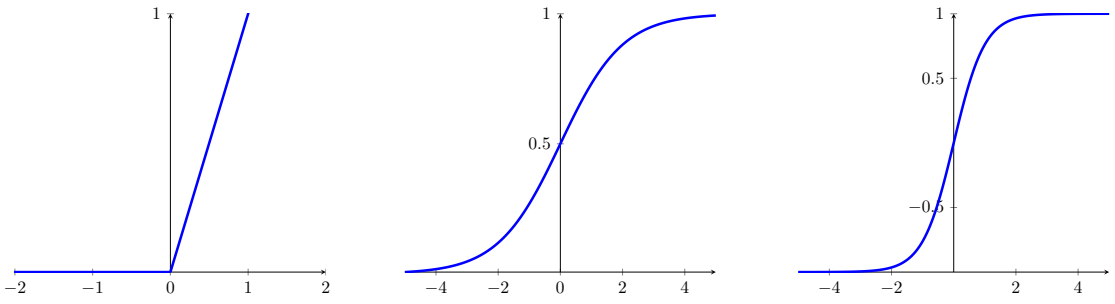


Figure 2.3: Illustration of some popular activation functions: (a) ReLu, (b) sigmoid and (c) hyperbolic tangent.

In the literature, other activation functions have been proposed recently ; such as Swish (Ramachandran et al., 2018) and GeLU (Hendrycks and Gimpel, 2016). According to this definition, a neural network output is a series of T no-linear compositions of any input x:

$$\forall x, f_{\mathbf{W}}(x) = f^T_{W^T} \circ \ldots \circ f^1_{W^1}(x),$$

where $\mathbf{W} = \{W^1, \ldots, W^T\}$ is the set of all model parameters; and each W^n; $1 \leqslant n \leqslant T$ is a matrix of weights. Suppose that, l_n is the number of neurons in layer n, and, $z^n = (z^n_1, \ldots, z^n_{l_n})^\top$ the vector of values of neurons in that layer, obtained recursively. Hence, we have

$$z^0 = x \tag{2.10}$$

$$z^n = f^n_{W^n}(z^{n-1}) = f^n_{W^n} \circ \ldots \circ f^1_{W^1}(x), \text{ and}$$

$$\forall j \in [l_n], z^n_j = H(a^n_j) \tag{2.11}$$

where $[l_n] = \{1, \ldots, l_n\}$, $a_j^n = \boldsymbol{w}_j^{n.\top} z^{n-1} + w_0^n$ with $\boldsymbol{w}_j^{n.}$ the vector column of matrix W^n that connects all neurons of layer $n-1$ with the neuron j from layer n. Each W^n can also encapsulate more complex structures, such as weight tensors, but for the sake of presentation, at the level considered here they will only appear as matrices, the structures they contain being always serialized. The activation function H is mostly the same all over the network; there are some studies that consider different activation functions depending on the layer, but in all case their activation functions are continuous and differentiable with respect to the data and the parameters.

In summary, as in the case of the perceptron, a layer of neurons is often implemented in two functions, corresponding to two different types of layers (or sub-layers): one called the linear layer, which corresponds strictly to the product of a matrix by a vector for the linear part, and the other called the point-to-point layer, which corresponds to the application of the activation function component by component for the non-linear part. Additional functions, such as normalization or regularization, are occasionally added.

2.4.2 Parameter learning with the backpropagation algorithm

To train the parameters of a neural network, most algorithms employ a method based on stochastic gradient descent using mini-batches.

Considering a training set $S = (x_i, y_i)_{1 \leq i \leq m}$; we seek to minimize a *regularized* objective function which is in line with the the SRM principle :

$$\hat{\mathcal{L}}_m(f_{\mathbf{W}}, S) = \frac{1}{m} \sum_{(x,y) \in S} \ell(f_{\mathbf{W}}(x), y) + \lambda \Omega(W) \tag{2.12}$$

where the loss function ℓ is assumed to be continuously differentiable, and $\Omega(W)$ is a regularization term whose purpose is to prevent overfitting. The function Ω is also assumed to be continuously differentiable, and the parameter λ is used to determine the relative *impact* of the regularization term. In the stochastic mini-batch variant of the gradient descent technique, weights $\mathbf{W}$ are updated based on a subset of the training set, $S^b \subseteq S$ termed mini-batch, as follows:

$$\mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} - \frac{\eta}{\mid S^b \mid} \sum_{(x,y) \in S^b} \nabla_{\mathbf{W}} \mathcal{L}_{(x,y)}(\mathbf{W}^{(t)}) \tag{2.13}$$

with :

$$\mathcal{L}_{(x,y)}(\mathbf{W}) = \ell(f_{\mathbf{W}}(x), y) + \lambda \Omega(\mathbf{W}) \tag{2.14}$$

For example, $\ell(.)$ could be the square loss and $\Omega(.)$ the squared norm of the weights :

$$\begin{aligned} \mathcal{L}_{(x,y)}(\mathbf{W}) &= \frac{1}{2}(f_{\mathbf{W}}(x) - y)^2 + \frac{\lambda}{2}\|\mathbf{W}\|^2 \\ &= \frac{1}{2}\sum_{j=1}^{l_T}(H(a_j^T) - z_j^T)^2 + \frac{\lambda}{2}\sum_{n=1}^{T}\sum_{i=1}^{l_{n-1}}\sum_{j=1}^{l_n}(w_{ji}^n)^2 \end{aligned} \tag{2.15}$$

To implement gradient descent as described in the equations (2.13) and (2.14), we need to be able to compute $\nabla_{\mathbf{W}} \ell(f_{\mathbf{W}}(x), y)$ for the example (x, y). This achieved recursively using the backpropagation algorithm (Rumelhart et al., 1986).

- The first part of this algorithm, called *forward pass*, corresponds to the spreading of the input layer per layer recursively as in (2.11).
- The second part is done in two steps;

– First *backpropagation according to data*, is done as part of a *back pass*. In this part of the backward pass, we calculate by recurrence for decreasing values of n starting from T, the gradients with respect to the data a^n from error ℓ by using the chain rule :

$$\forall j \in [l_T], \delta_j^T = \frac{\partial \ell(z^T, y)}{\partial a_j^T} \tag{2.16}$$

$$\begin{aligned} \forall n \in \{T-1, \dots, 1\}, \forall j \in [l_n]; \delta_j^n = \frac{\partial \ell(z^T, y)}{\partial a_j^n} &= \sum_{k=1}^{l_{n+1}} \frac{\partial \ell(z^T, y)}{\partial a_k^{n+1}} \frac{\partial a_k^{n+1}}{\partial a_j^n} \\ &= H'(a_j^{n+1}) \sum_{k=1}^{l_{n+1}} \delta_k^{n+1} w_{kj}^{n+1} \end{aligned} \tag{2.17}$$

– The second step, called *back-propagation according to parameters*, does not involve a recurrence relation. The gradients with respect to the parameters $W^n = [w_{ji}^n]$ of the error ℓ are calculated by using the chain rule once again :

$$\begin{aligned} \forall n \in \{T, \dots, 1\}, \forall i \in [l_{n-1}], \forall j \in [l_n]; \frac{\partial \ell(z^T, y)}{\partial w_{ji}^n} &= \frac{\partial \ell(z^T, y)}{\partial a_j^n} \frac{\partial a_j^n}{\partial w_{ji}^n} \\ &= \delta_j^n z_i^{n-1} \end{aligned} \tag{2.18}$$

After (2.18), parameters are updated using the GD rule (2.5). The backpropagation procedure is illustrated in figure 2.4.

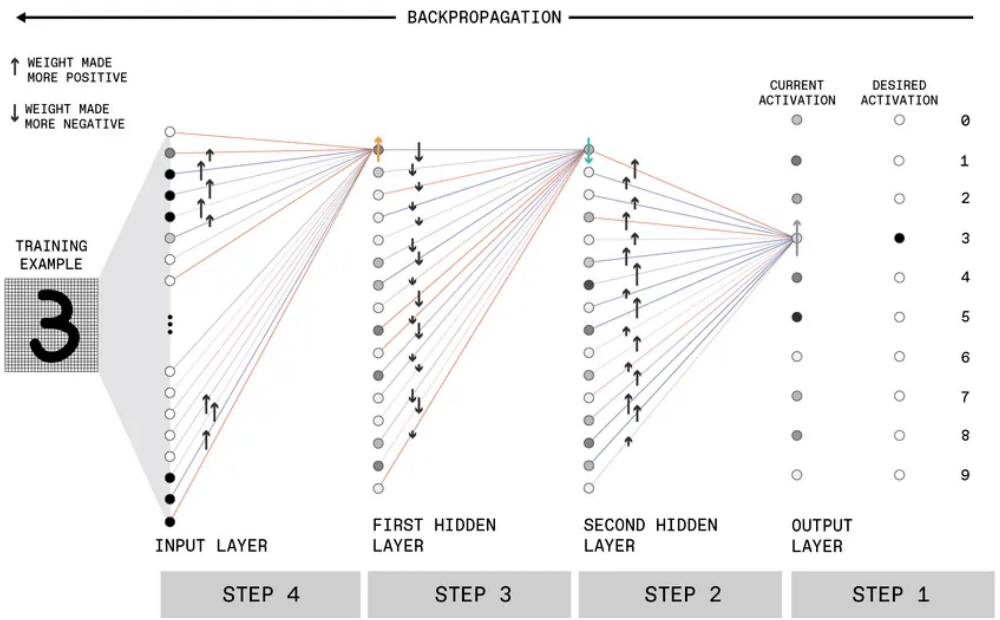


Figure 2.4: Illustration of a backward step. The weights between neurons are optimized, positively by increasing (up arrow) or negatively by decreasing (arrow down). The backward goes layer-by-layer starting with last layer to input layer, as indicated by steps. Illustration from (Schneider, 2021)

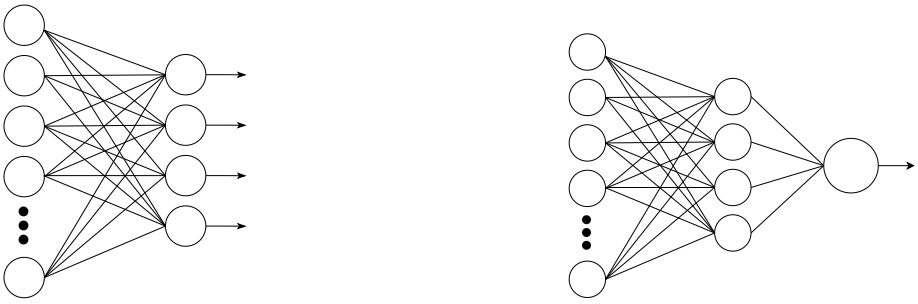
Now that we have covered the principles of deep learning, we will go through some of the most common types of neural network models.

2.4.3 Different types of networks

Deep learning is rapidly changing, with new networks and architectures appearing every few months. We will briefly present the most common types of networks in this section, which are feed-forward, recurrent and convolutional networks.

Feed Forward Network. The feed forward model is the most prevalent type of neural network, in which information is propagated from the input to the output in one way.

The single layer perceptron network is the simplest type of feed-forward network, consisting of a single layer of output nodes with the inputs fed directly to the outputs via a set of matrix weights (see fig. 2.5a). The multi-layer perceptron is a straightforward expansion of the single-layer perceptron (MLP). This network is made up of numerous layers of computing units coupled in a feed-forward method. Directed connections connect each neuron in one layer to the neurons in the next layer. (see fig. 2.5b). A large majority of the actual neural network are direct descendant of this type of network.



(a) *(b)*

Figure 2.5: The left-hand side is a single layer perceptron, on the right-hand side its direct extension, that is a multi-layer perceptron (MLP)

Recurrent Neural Network. RNN is a widely used and popular deep learning approach, especially in natural language processing and speech processing (Cho et al., 2014b). RNNs, unlike typical neural networks, deal with sequential data. In many applications, when the intrinsic structure in the data sequence transfers vital information, this quality is critical (e.g. to understand a word in a sentence the context is required). RNNs are similar to typical neural networks in that they consist of an input layer, numerous hidden layers (commonly referred to as states), and an output layer. A traditional RNN is depicted in the figure 2.6. "Input-to-Hidden," "Hidden-to-Output," and "Hidden-to-Hidden" are three deep RNN techniques proposed by (Pascanu et al., 2014). A deep RNN is built based on these three strategies that not only takes advantage of deeper

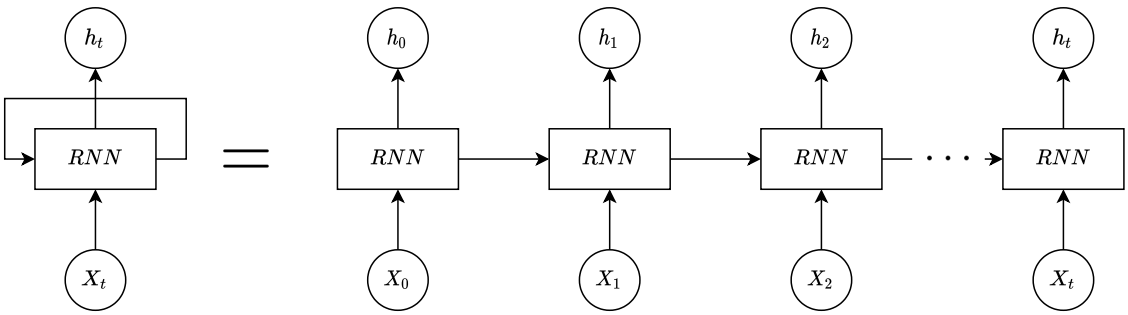


Figure 2.6: Illustration of an RNN and its unrolled version

RNNs but also reduces the complexity of deep network learning. Later, to replace recurrent cells, additional improvements such as Long Short-Term Memory (Hochreiter and Schmidhuber, 1997) or Gated Recurrent Unit (Cho et al., 2014a) were developed.

Convolutional Neural Network. The CNN could be the most popular type of network, as it is so widely used in deep learning (LeCun, Bengio, et al., 1995) community. This type of network has been employed in a variety of applications, including NLP (Jacovi et al., 2018) and Computer Vision (Khan et al., 2018). The application of computer vision will be the emphasis of this paragraph. Local connections and shared weights are employed in the network instead of fully linked networks to fully use the two-dimensional structure of incoming data (e.g., image). This method produces a network with considerably fewer parameters, making training faster and easier. Short segments of a scene are more receptive to these procedures than the complete scene. To put it another way, the cells operate as local filters on the input, extracting data with spatially constrained correlation. More formally, a CNN is made of several convolutional layers, where in each one several filters of size $\ell \times \ell \times k$ where ℓ is the size of the filter and k is the numbers of filters. The figure 2.7 illustrates a convolution with $\ell = 3$ and $k = 1$.

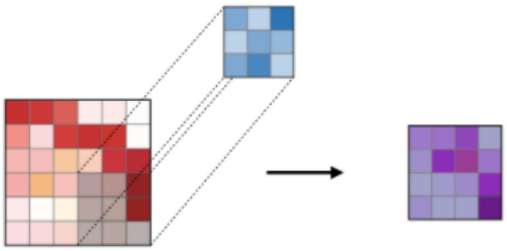


Figure 2.7: Illustration of a convolution, with a filter size of 3×3*. Illustration from (Amidi, 2018)*

2.4.4 ResNet and its variants

We will go through some of the most often utilized model in computer vision, which is ResNet. We will demonstrate one such modification of a reference architecture that was initially created for image classification, but has been extended to a variety of workloads. This section is not meant to be exhaustive; rather, it is meant to highlight ResNet that we will also consider in our contributions.

ResNet. Residual Network (ResNet) introduced by He et al., 2016 have been rapidly adopted by the community, to become one of the most used neural architecture in the deep learning community. Initially, this network has shown significant performance by winning the ILSVRC 2015 image classification challenge[4] with an error rate of 3.57% which is lower than the percentage of error that humans make on this problem (which is around 5%). To solve a complex problem, deep neural networks are usually stacked with additional layers, which improves accuracy and performance. The idea behind adding more layers is that these layers will gradually learn more complex features. In the case of image recognition, for example, the first layer can learn to detect edges, the second layer can learn to identify textures, and the third layer can learn to detect objects, and so on. However, it has been discovered that the traditional convolutional neural network model has a maximum depth threshold. As depicted in fig. 2.8 adding more layers on top of a

[4] https://image-net.org/challenges/LSVRC/2015/

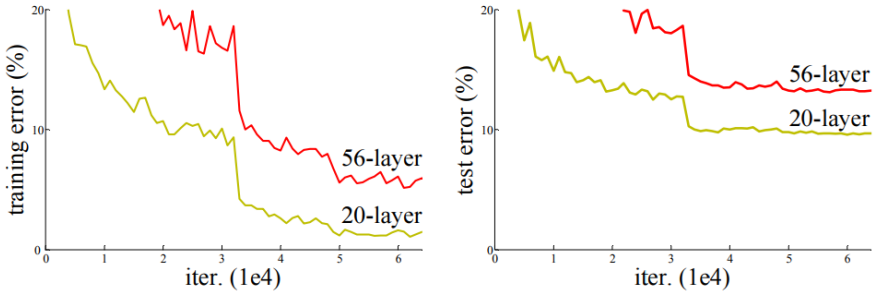


Figure 2.8: Plot of the training and test error obtained with two deep plain networks, with different number of layers. The network with a larger number of layers perform worse than its smaller counterpart. Illustration from (He et al., 2016)

network degrades its performance. This could be attributed to the optimization function, network initialization, and, most importantly, the vanishing gradient problem.

To overcome this problem of training a deep neural network, the authors propose to use "*Residual block*" (Figure 2.9). The first thing you notice is that there is a direct connection between the two that skips a few layers. This is known as a "skip connection," and it is at the heart of the residual blocks. Because of this skip link, the layer output is no longer the same. Without this jump connection, the input x is multiplied by the layer weights before being multiplied by a bias term.

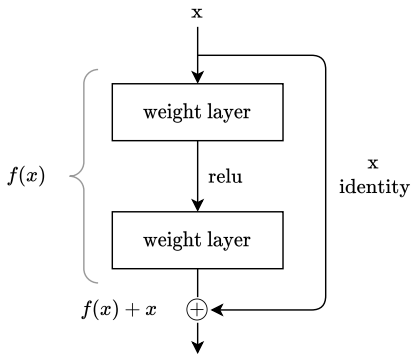


Figure 2.9: Illustration of a residual block.

This block can have some modification in the case where $f(x)$ and x do not have the same dimension. In this case, we usually use a projection layer (a convolution 1×1) to match the dimension, the final result is so $f(x) + \mathbf{W}x$. ResNet skip connections are extremely beneficial in that they

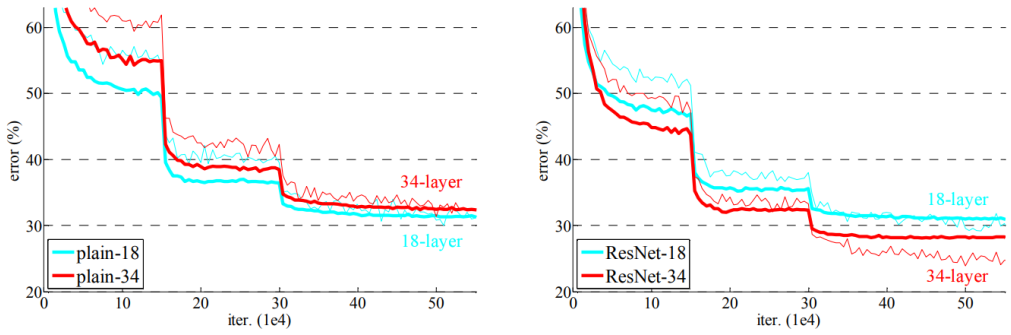


Figure 2.10: Plot of the error obtained by a plain network (on left) and by ResNet (on right) with the same number of layer. Illustration from (He et al., 2016)

solve the problem of gradient varnishing in deep neural networks by enabling the gradient to flow through this alternate shortcut. These connections are also beneficial in helping the model to learn identity functions, ensuring that the top layer performs at least as well as the bottom layer, if not better. By using residual blocks, the performance of a deeper network are increased and overpass the performance of "*shallow*" networks, as shown in fig. 2.10.

In the literature, two versions of ResNets are widely used ResNet-50 and ResNet-101, which have 50 and 101 layers, respectively.

Variant. Currently, many methods use a ResNet as a backbone (or feature extractor) and propose improvements around it depending on the task to be performed. Because there are almost as many neural networks as there are tasks, we will take the segmentation task as an example. This task consists in the prediction of a semantic mask from an image (e.g. figure 2.11). More details on this task will be given in a later chapter. This problem can be reduced to a pixel by pixel classification, which is harder to solve than a whole image classification, and so require a more complex neural network. Many works have been proposed to address this task, but we will focus on one of the most used DeepLabV1 (Chen et al., 2015) and those evolutions DeepLabV2 (Chen et al., 2018b), V3 (Chen et al., 2017) and V3+ (Chen et al., 2018c).

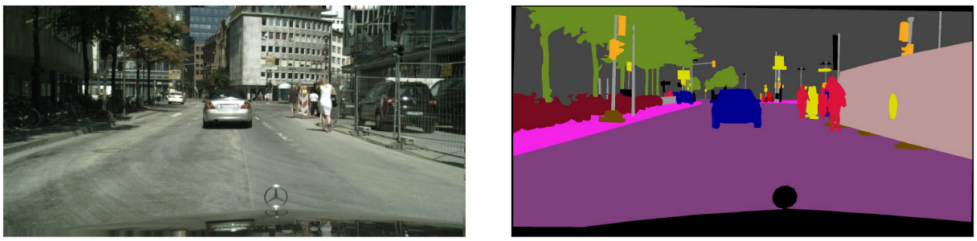
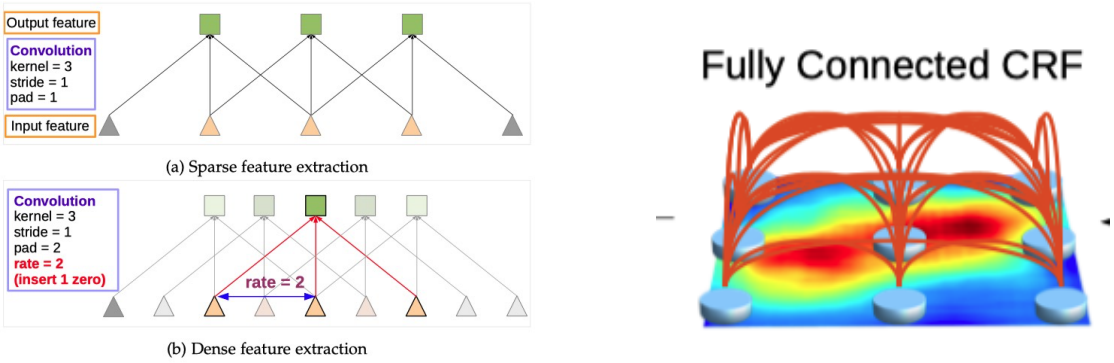


Figure 2.11: Example of the segmentation task, with the input image and the associated segmentation mask. Illustration from (Chen et al., 2018b)

The nature of the first changes is based on the following points: the use of new types of convolutions called *atrous* (e.g. figure 2.12a) and a new module called Fully-connected Conditional Random Field (CRF) (e.g. figure 2.12b). All these modifications helps to produce a quality segmentation mask.

The following versions will bring new improvements in order to have always better performances. Among the most important new features, we find (in historical order): the appearance



(a) *(b)*

Figure 2.12: Illustration of atrous convolution and fully connected CRF proposed in Deeplab. Illustration from (Chen et al., 2015)

of an Atrous Spatial Pyramid Pooling (ASPP) module (see fig. 2.13a), the use of a multiscale structure, the abandonment of the CRF module, and the implementation of an encoder-decoder architecture. The most recent architectures (DeepLabV3 and V3+) are shown in figure 2.13. In DeepLabV3 (Fig. 2.13a), the architecture is based on the ResNet backbone, with atrous convolutions and ASPP at the third ResNet block in order to predict meaningful masks. In DeepLabV3+, a ResNet network is also used as backbone, but they add a decoder part. In this case, features are extracted from low level stage and merged with deeper features thanks to the decoder.

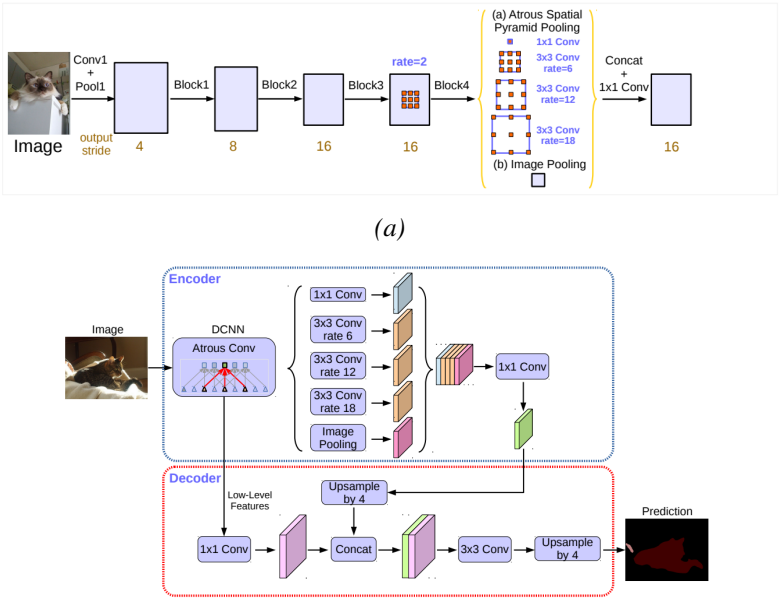


Figure 2.13: Illustration of the networks used un DeeplabV3 and V3+. Illustration from (Chen et al., 2017) and (Chen et al., 2018c)

2.5 Summary

Now that we've defined and explained what a neural network is, we can see that the majority of the networks that comprise the state of the art are hand-crafted. However, the design process is frequently time-consuming and costly. Indeed, designing a network necessitates specific skills and expertise of one or more disciplines. This is frequently a lengthy iterative process of trial and error. Furthermore, since hardware systems advance at a quick pace, it becomes more difficult to choose a network that will best fit the user's hardware. To solve this architecture design dilemma, a new branch of research has just evolved. This is the Neural Architecture Search, which allows users to find the architecture that is best suited to their task, data, and hardware automatically. These approaches will be discussed in more details in the next chapter.

3

Introduction to Neural Architecture Search

This chapter gives an overview of the area of Neural Architecture Search (NAS). More precisely, we will go through the many concepts that form the foundation of Neural Architecture Search. Then we will go through some of the many search algorithms that have been presented by the community in recent years. Finally, we will discuss several estimating strategies that have been utilized to minimize the amount of time necessary to discover an architecture. We will also discuss some potential research options for this new but rapidly expanding field of study.

Contents

3.1 Introduction

The automation of feature engineering has largely contributed to a wide use of deep learning in many applications, such as perception. In traditional machine learning algorithms, the definition or the selection of data features are done manually, which are long and costly in general. Deep learning overcomes this difficulty, thanks to feature extractors learned in an end-to-end way. However, these feature extractors rely on architectures that are still manually designed and with the rapid development of field the design of an adapted NN model becomes tedious in many situations.

A new field of research known as **N**eural **A**rchitecture **S**earch (NAS) has recently arisen to address this problem. In applications such as image segmentation and classification, Neural Networks with automatically obtained architectures have already proven their efficiency by surpassing traditional Neural Networks with hand-crafted structures. The NAS study is part of a much wider super-area in Machine Learning known as AutoML. This super-domain collects all the research on Deep Learning pipeline automation, including data preparation and selection, hyperparameter optimization, and even meta-learning. The three dimensions of the NAS pipeline are: search space, search strategy or architecture optimization, and performance evaluation strategy.

- ***Search Space :*** The search space specifies the design concepts for various neural architectures. This space is specific to each scenario, usually it incorporates the prior knowledge gained from human well-designed architectures. This knowledge is important as it helps to reduce this space and keep it simple, this also has some drawbacks, as it is at this stage that biases can be introduced and potentially force the research in the wrong direction.
- ***Search Strategy :*** This phase, also known as Architecture Optimization, outlines how to efficiently explore the search space in order to discover the best performing architecture. Because the aim is to uncover efficient structures rapidly, but sticking inside a restricted region might lead to poor findings, this is a real example of the trade-off between exploration and exploitation.
- ***Evaluation Method :*** After a model has been constructed, its performance must be assessed, frequently using previously unseen data. The easiest method is to train the model on training data until it converges, then assess it on a validation set. However, this technique necessitates a high degree of computing as well as sufficient time to sample and train a sufficient number of models, limiting the number of architectures that may be studied. Some new strategies promise to speed up this procedure at the expense of fidelity in the model evaluation. Some recent methods propose to accelerate this process at the cost of a loss of fidelity in the evaluation of the model.

Figure 3.1 illustrates these three components and how they interact. In the next section, we will present the various types of search space, then section 3.3 will cover the different strategies, finally the evaluation methods will be discussed in section 3.4.

3.2 Search Space

Generally, a Neural Architecture can be represented as a Direct Acyclic Graph (DAG) composed of N nodes, where each of the nodes represents a latent feature vector and each directed edge is an operation. The computational process at each node can be represented by the equation 3.1.

$$F_d = \sum_{i=1}^{N_d} o_i(I_i) : o \in \mathcal{O} \tag{3.1}$$

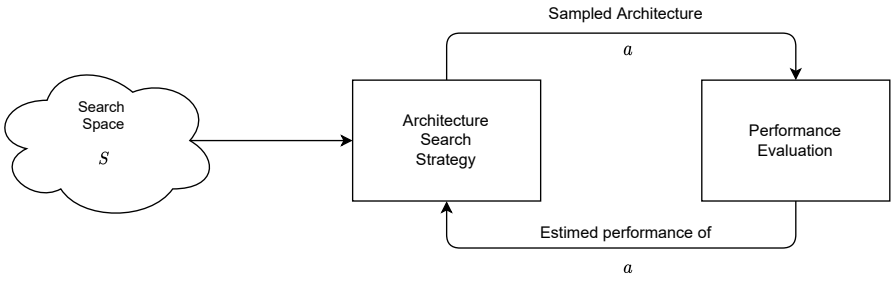


Figure 3.1: General neural architecture search pipeline. This pipeline is formed of three major components, namely Search Space, Search Strategy and Performance evaluation. To begin, one must first define the search space, which must then be combined with a search technique. This approach generates architecture a, which will be evaluated, and the strategy will then utilize the architecture of a to steer the search inside the search space.

where $F_d, d \in 1, ..., N$ is the latent feature vector associated with node d, N_d is the node index of F_d, I_i and o_i are the i-th input tensor and the associated operation, respectively. $\mathcal{O}$ is the set of candidates operations, which can contain basic operations like convolution, pooling, activation or concatenation. Recent research suggests using more complex operations to get better and better performances or more efficient networks; among these new operations are depth-wise separable convolution, squeeze-and-excitation, and others (Hu et al., 2018b). The search space specifies the structure that the Search Strategy will investigate; as a result, this stage is critical for the remainder of the process, but it is also complicated in that the operations' selection and order will vary based on the beginning space. In the following sections, we will present the two most commonly used search spaces. It is important to know that other techniques exist and that they will be presented if necessary in the corresponding chapters.

3.2.1 Macro Search Space

The macro (or entire-structured) search space, was one of the first to be proposed because of its simplicity. Figure 3.2 illustrates various types of macro search space, in this situation each node represents an operation layer. The left side of fig. 3.2, is the simplest, because it is built from a stack of operations of a predefined number of operations and linked in chain form, the middle representation is more complex as it allows the creation of skip-connections between nodes, these connections have shown their efficiency on human-designed models. These connections have also allowed the creation of more complex networks, called multi-branch, these networks can have parallel operations as illustrated on the right side of the figure 3.2. In this case, the input tensor I_i is described as a combination function, which can be a sum (He et al., 2016) or a concatenation (Huang et al., 2017). However, despite this simplicity, this search space structure has some disadvantages. The search for a very deep neural architecture is extremely computationally expensive, which limits the size of the architectures and thus the ability of the architectures to generalize. In addition, in some cases, the architecture found on a small dataset is not transferable or is not as efficient on larger datasets.

3.2.2 Micro Search Space

Inspired by high performing handcrafted architecture with repeated motifs such as in ResNet (He et al., 2016) or DenseNet (Huang et al., 2017), some researchers proposed to use a *cell-based* search space (i.e. also known as *micro* search space) (Zoph et al., 2018; Zhong et al., 2018). In this case the goal is to search those motifs, called *cells*, rather than the whole architecture. The final structure is composed of a fixed number of structures of these cells repeated. This search

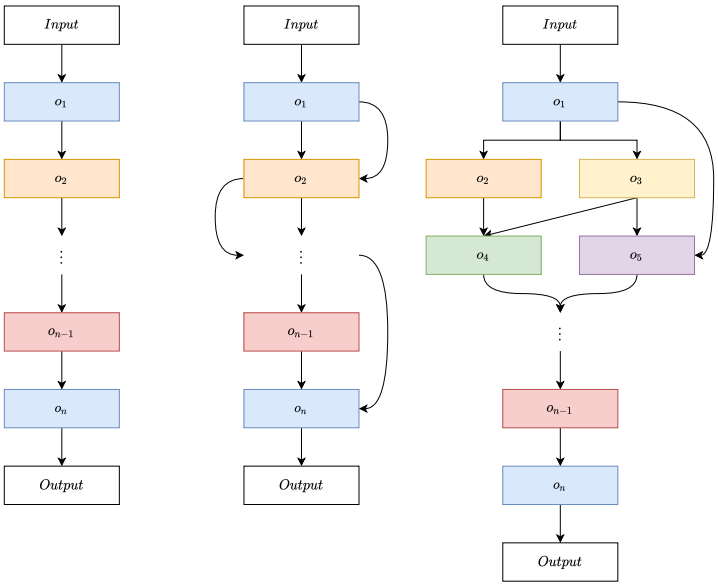


Figure 3.2: Illustration of a macro search space. The left side, is a chain-like structure, with a simple feed-forward paradigm. The center, is more advanced as it contains complex structure like skip-connection. Finally, the right side is even more complex, as it contains multi-path and skip-connection in the structure.

spaces as several advantages compared to the macro search space :

- Since cells often have fewer layers than entire architectures, the size of the search space is considerably reduced.
- This search space allows for more transferability and adaptability because it only requires stacking more cells or change some filter size to construct a larger network, with no re-searching step.

Figure 3.3 depicts a cell-based architecture, such as that presented by (Zoph et al., 2018) . It is made up of two types of cells: normal and reduction, which retain and reduce dimensionality, respectively, and which are similar to many commonly used structures.

The internal design of these cells refers in most of the NAS research to those proposed by (Zoph et al., 2018), being the first to have experimented with this type of search space.

All these advantages and performances have allowed the rapid adoption of this search space in the most recent papers. However, some drawbacks need to be mentioned. When adopting a cell-based search space, a new design choice arises: how to determine the macro-architecture: how many cells should be used, and how should they be connected to construct the real model? Zoph et al., 2018 construct a sequential model from cells in which each cell receives as input the outputs of the two preceding cells, whereas Cai et al., 2018 use the high-level structure of well-known manually designed architectures, such as DenseNet (Huang et al., 2017), and their cells within these models. In theory, cells can be arbitrarily combined, for example, within the previously described multi-branch space, by replacing layers with cells. Rather than optimizing the micro-architecture only, both the macro-architecture and the micro-architecture (i.e., the structure of the cells) should ideally be jointly optimized; otherwise, after finding a high-performing cell, one may have to manually build the macro-architecture. Research is beginning to address this topic, as with a hierarchical search space. Another point is that the NAS methods of cell-based search space are often divided into two phases: search and evaluation. The best-performing model is chosen first in the search phase, and then it is trained from scratch or fine-tuned in the assessment phase. However, there is a significant difference in model depth between the two phases. While

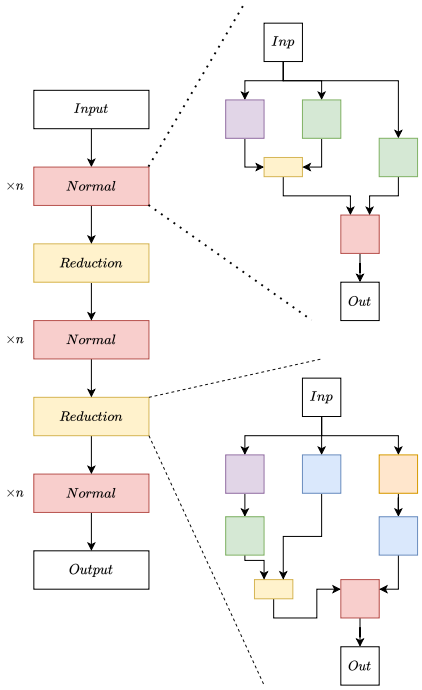


Figure 3.3: Illustration of a micro-search space. The right part represents the cells sought, here is the reduction and normal types. Once these cells are built, they are then stacked to form the structure on the left, which will represent the final structure. The "$\times n$" terms denotes the number of times this cell is repeated, "n" being a hyperparameter.

the search phase identifies the optimum cell structure for the shallow model, this does not imply that it is still appropriate for the deeper model in the evaluation phase. In other words, merely increasing the number of cells may degrade model performance. To close this gap, recent research such as (Chen et al., 2019) proposed progressive DARTS (P-DARTS), a strategy that separates the search period into many stages and gradually raises the depth of the searched networks at the conclusion of each step.

3.2.3 Hierarchical Search Space

A new search space called "hierarchic" has emerged, inspired by the two forms of search space shown above. In the cell-based search space (micro), most of the proposed method follow a bi-level hierarchy : the inner level is the cell level, which determines the operation and connection for

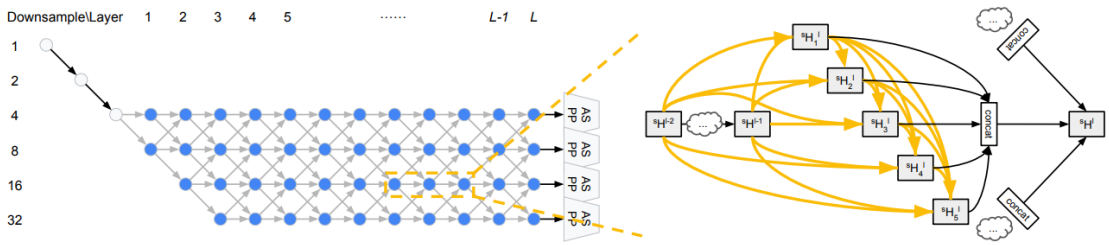


Figure 3.4: Illustration of the hierarchical search space proposed by AutoDeepLab. The left part represent the dense network among which the path will be searched, the x-axis represent the layer index, the y-axis represent the downsampling ratio compared to the input size. For each down sampling level, the number of features is multiplied by 2 compared to the previous level (e.g. 64,128,256,512). Illustration from (Liu et al., 2019a).

each node in the cell, and the outer level is the network level, which regulates spatial-resolution changes. However, these techniques are cell-centric and neglect the network level. The spatial dimension can only stay the same or be halved by adding a reduction cell. (Liu et al., 2019a) established a broad formulation for a network-level structure, represented in Figure 3.4, from which many current acceptable network designs can be duplicated in order to jointly learn an appropriate combination of repeatable cell and network structures. This allows us to fully examine the varying number of channels and feature map sizes for each layer. Transversal works propose to use a hierarchical search space, but in a more flexible manner via dynamic routing (Li et al., 2020); we will go over this in greater detail in Chapter 7. However, these works can be applied to NAS in the sense that we search for the best path for each entry using a cell mesh similar to Autodeeplab. Furthermore, a "common" path, can be extracted if necessary to keep only the necessary cells.

Liu et al., 2017a have used this notion of hierarchy to build progressively more complex cell. As example, starting from level one with basic operations such as convolutions with various kernel size (e.g. $3 \times 3; 5 \times 5$), pooling. The level two use, the level one operation to build basic cell, then the level three uses those basic cells to build more complex cells and so on. Finally, the highest level represent the full architecture. An illustration of HierNAS (Liu et al., 2017a) process is given in figure 3.5.

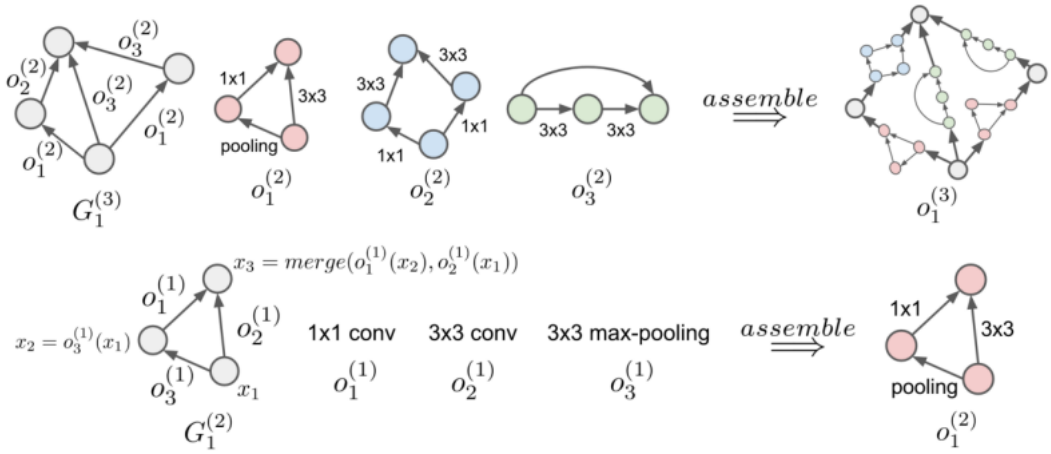


Figure 3.5: Illustration of the hierarchical search space used by HierNAS. In this figure, the graph $G_1^{(2)}$ is composed operation of level 1 (o_1^1, o_2^1,...) to form the operation of level 2 ($o_1^{(2)}$). Then the operation of level 2 ($o_1^{(2)}, o_2^{(2)}$,...) will be used in graph $G_1^{(3)}$ to build operation of level 3 and so on. Illustration from (Liu et al., 2017a)

The following section discusses search strategies that are well-suited for these types of search spaces.

3.3 Search Algorithm

Following the definition of the search space, we must search for the best-performing architecture, a process known as Search Algorithm (also known as Search Strategy or Architecture Optimization). Various search strategies, including random search, Bayesian optimization, evolutionary methods, reinforcement learning (RL), and gradient-based methods, can be employed to explore the space of neural networks. This process is heavily reliant on human specialists and ne-

cessitates a significant amount of time and resources for trial and error. We will present the most commonly used algorithms.

3.3.1 Evolutionary Algorithm

Chronologically speaking, evolutionary algorithms were the first to be used in NAS (Miller et al., 1989). The evolutionary algorithm (EA) (also called neuro-evolution) is a population-based metaheuristic optimization algorithm inspired by biological evolution. EA is a global optimization approach with excellent robustness and extensive applicability when compared to traditional optimization algorithms such as exhaustive methods. It can solve complex issues that standard optimization algorithms cannot solve without being bound by the nature of the problem. In general, the EA is divided into three stages:

- **Selection :** This stage entails selecting a subset of all created networks for mutation, with the goal of retaining well-performing neural architectures while discarding the unpromising ones. This selection can be done in different ways. Performance selection is a mechanism in which the probability of a network being chosen is proportional to its performance value. Another used method is tournament selection, in this case, k (tournament size) networks are randomly chosen from the population and ordered according to their performance in each iteration; then, the best network is chosen with a probability of p, the second-best network with a probability of $p \times (1 - p)$, and so on. Lastly, Elsken et al., 2019 propose using an inverse density to pick parents from a multi-objective Pareto front.

- **Mutation :** After the parent has been selected, a mutation process is applied, and it can take different forms. Xie and Yuille, 2017 use one of the most common processes, called point mutation, which involves randomly and independently flipping each bit in an encoded version of the architecture. In (Miikkulainen et al., 2019) authors propose two types of mutation, one turns on or off a connection between two levels, while the other adds or eliminates skip connections between two nodes or layers. (Real et al., 2017) build a collection of mutation operators, such as changing the learning rate and eliminating skip links between nodes.

- **Update :** The new networks formed by completing the preceding processes are numerous, and because of computing resource constraints, some of these must be deleted. Real et al., 2017 propose to simply remove the least performing network, in (Real et al., 2019), authors delete the oldest one. Some studies as in (Suganuma et al., 2017) also remove models but at regular interval, while in (Liu et al., 2017a), models are not removed at all.

Many methods for optimizing architecture and weights employ evolutionary algorithms similar to the one presented above, however, these approaches run into scale issues when millions of parameters need to be tuned. Recent works propose to combine gradient descent and evolutionary algorithm, to optimize the parameters and the architecture, respectively (Suganuma et al., 2017; Real et al., 2017). Most of the presented methods use random initialization of child networks to generate offspring, however Elsken et al., 2019 use Lamarckian inheritance, in which information (in the form of learned weights) is transmitted from a parent network to its offspring. In (Real et al., 2017), authors allow an offspring to inherit all of its parent parameters that are not involved by the selected mutation. Compared to random initialization, this type of inheritance, which is not exactly function preserving, may speed up learning.

3.3.2 Reinforcement Learning

The first attempts to apply the Reinforcement Learning (RL) concept to NAS stated that the creation of an architecture can be thought of as the action of the agent, with the action space being

similar to the search space. The figure 3.6 illustrates the process RL-based NAS. The controller, is usually a recurrent neural network (RNN) which sample an architecture A at each time step t from the predefined search space. This architecture A is then trained and evaluated on unseen data, the obtained performance is then used as reward r to update the controller sampling policy.

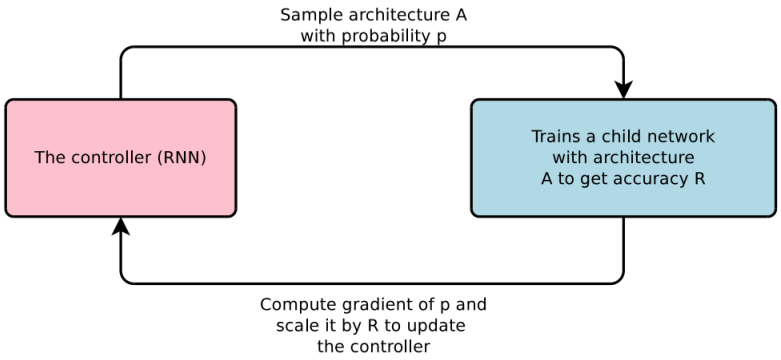


Figure 3.6: Operating scheme used in the RL-based architecture search. Illustration from (Zoph and Le, 2017).

Various RL techniques differ in how they describe and optimize the policy of the agent. Zoph and Le, 2017 use the REINFORCE policy gradient algorithm to optimize the controller policy. However, in (Real et al., 2017), authors use another optimization method called proximal policy optimization (PPO) (Schulman et al., 2017). Figure 3.7 illustrates the output of the RNN. Baker et al., 2017 propose MetaQNN a meta modeling method that use Q-learning to train a policy that searches for neural architecture sequentially.

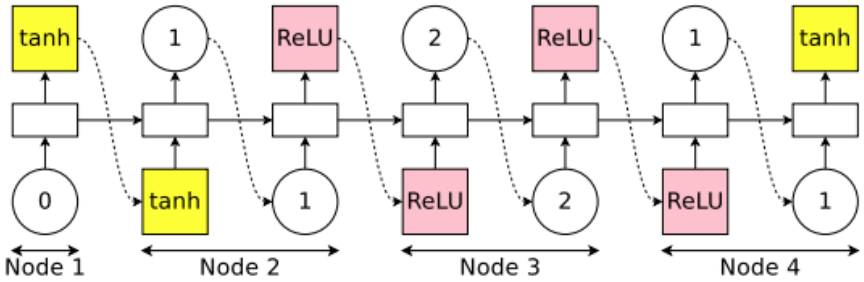


Figure 3.7: Illustration of the result generated by the controller, here, it is a case of micro search space with 4 nodes per cell. The controller takes as input either the id of the input node or the operation applied, and produces as output the operation applied or the id of the input node for the following node, respectively. Illustration from (Pham et al., 2018)

This strategy allowed to obtain good performances on the image classification but also on other domain such as language modeling with the usage of this technique on the Penn Treebank dataset (Marcus et al., 1994), nevertheless this method remains very greedy in computing resources. Indeed, in (Baker et al., 2017), authors used 800 GPU for more than 20 days to obtain best-performing architecture, while Zoph and Le, 2017, used 10 GPU for 10 days. Recent works, proposes to drastically reduce the amount of resources needed as in Efficient Neural Architecture Search (ENAS) (Pham et al., 2018), where the authors use weight sharing across all child architecture, which avoids the need to train from scratch each child. By this strategy, they reduce the required time by $\times 1000$, using a single GPU for 10 hours.

3.3.3 Gradient Descent based

Different from the previously presented search strategy which sample operation from a discrete search space, (Liu et al., 2019b) propose differentiable architecture search (DARTS) a continuous relaxation of the search space to make it differentiable. This relaxation enable the usage of the gradient, which has been successfully proposed by DARTS. The search space is then represented as in eq. (3.2).

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

where the operation mixing weights for a pair of nodes (i, j) are parameterized by a vector $\alpha^{(i,j)}$ of dimension $|\mathcal{O}|$. The architecture search problem is then reduced to learn a set of continuous variables $\alpha = \{\alpha^{(i,j)}\}$ (edges on figure 3.8). After the relaxation, the tasks of architecture search is then transformed to a joint optimization problem, between the neural architecture α and the associated weight θ. Thus, this is a bi-level optimization, where those parameters are alternatively optimized. More specifically, α and θ are not optimized with the same set, validation and training, respectively. The optimization process is then given by eq. (3.3).

$$\begin{aligned} \min_{\alpha} \quad & L_{val}(\theta^*, \alpha) \\ \text{s.t.} \quad & \theta^* = argmin_{\theta} L_{train}(\theta, \alpha) \end{aligned} \tag{3.3}$$

where L_{train} and L_{val} are the training and validation loss, respectively. The task of searching for architecture is then reduced to learning a set of continuous variables $\alpha = \{\alpha^{(i,j)}\}$. By replacing each mixed operation $\overline{o}^{(i,j)}$ with the most likely operation (i.e. $o^{(i,j)} = argmax_{i \in \mathcal{O}} \alpha_o^{(i,j)}$) at the end of the search, a discrete architecture can be obtained. The whole process is illustrated in fig. 3.8.

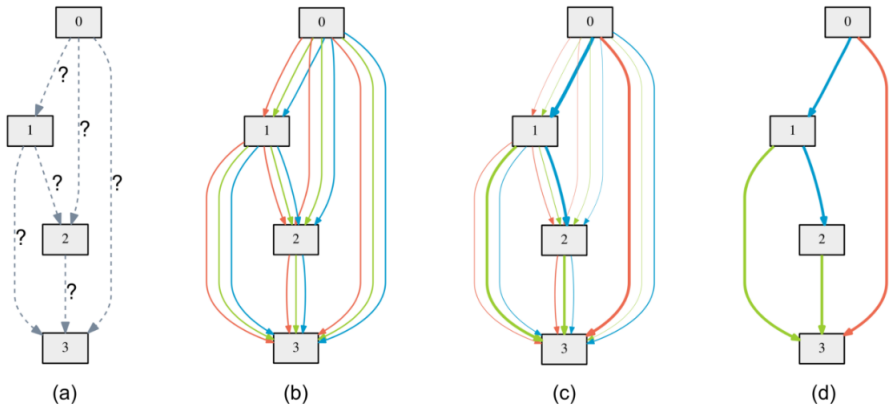


Figure 3.8: Illustration of differentiable architecture search, with $|\mathcal{O}| = 3$. In this overview, edges represent different operations, nodes represent the latent feature vector. The node 0 and 3 represent the input and output of a cell, respectively. (a) Operations on the edges are unknown at start. (b) represents the continuous relaxation, replacing each edges by operation mixing given by eq. (3.2), (c) is the joint optimization of mixing probabilities and the network weights, finally, (d) is the final architecture from the learned probabilities. Illustration from (Liu et al., 2019b)

.

Despite the promising advantages offered by gradient-based methods, they also suffer from some drawbacks. The first one is that the bi-level optimization is a complex problem to solve, because the two parameters α and θ are high dimensional parameters. A solution proposed by the authors is to approximate the gradient of $L_{val}(\theta^*, \alpha)$ by using $\nabla_{\alpha} L_{val}(w - \xi \nabla_w L_{train}(w, \alpha), \alpha)$,

where w are the current weights of the architecture, and ξ the learning rate. The key idea here is to approximate $w^*(\alpha)$ by adapting w using only a single training step, without solving the inner optimization (e.q. 3.3) completely by training until convergence. A second point is, in the proposed method, each edge is a weighted sum of candidates operations, which in the training stage, leads to a linear increase of the required amount of GPU memory with the number of candidates operations. To tackle this issue, for example Xie et al., 2019b propose to have a differentiable sampling, to sample a child architecture from the supernet by using the Gumbel Softmax trick (Jang et al., 2017). The architecture can be modeled by a concrete distribution (Maddison et al., 2017), this allows gradient back propagation and gives an efficient technique to sampling a child architecture. By using this trick, the equation 3.2 can be rewritten as :

$$\overline{o}_k^{(i,j)}(x) = \sum_{k=1}^{K} \frac{\exp((\log \alpha_k^{(i,j)} + \mathbf{G}_k^{(i,j)})/\lambda)}{\sum_{k=1}^{K} \exp((\log \alpha_k^{(i,j)} + \mathbf{G}_k^{(i,j)})/\lambda))} o^k(x) \tag{3.4}$$

In this equation, K is the number of operations, $\mathbf{G}_k^{(i,j)} = -\log(-\log(u_k^{i,j}))$ is the k-th Gumbel variable, $u_k^{i,j}$ is a uniform random variable and λ is the softmax temperature. When the temperature $\lambda \to \infty$, the possibility distribution of all operations between each node pair approximates to one-hot distribution. Going further, Cai et al., 2019 propose to use route binarization to reduces the massive resource usage. It converts the real-valued path weights to binary gates using BinaryConnect, which activates only one path of the mixed operations and therefore addresses the memory problem (Courbariaux et al., 2015).

3.3.4 Surrogate Model-based Optimization

Another type of search strategy, related to Bayesian optimization, is sequential model-based optimization (SMBO) algorithms (Kandasamy et al., 2018; Negrinho et al., 2019). The key notion of SMBO is that it iteratively keeps a record of prior assessment outcomes to develop a surrogate model of the objective function, and then utilizes the surrogate model to anticipate the most promising architecture. Using this prediction, these methods can significantly reduce search time and increase efficiency. SMBO algorithms use surrogate models, which can be broadly divided into *"classical"* Bayesian optimization (BO) methods and neural networks. The BO is widely used for hyperparameter optimization, but has been applied only rarely to the NAS research. (Kandasamy et al., 2018) use a derived version of kernel function for the search space in order to use a classical Gaussian Process(Rasmussen, 2003) to guide the search of architecture. Other techniques exist such as random forest (RF) (Hutter et al., 2011), tree-structured Parzen estimator (TPE) (Bergstra et al., 2011), such techniques have demonstrated high performance across a wide range of problems, by combining the architecture search and the associated hyperparameter (Bergstra et al., 2013). Other studies propose to substitute the "classical" BO methods by neural networks as surrogate model. (Liu et al., 2018) propose use a LSTM (Hochreiter and Schmidhuber, 1997) as surrogate model, in (Luo et al., 2018) the authors use a simple feed-forward network and achieves better results than (Liu et al., 2018).

3.4 Evaluation Estimation Strategy

After setting up a new neural network, its performance must be evaluated. A straightforward approach is to train the network to convergence and then evaluate its performance. However, this process necessitates a significant amount of time and computing power. As an example, AmoebaNet (Real et al., 2019) and NASNet (Zoph et al., 2018) required 500 P100 GPUs and 450 K40

GPUs, respectively, which is unaffordable in most cases. This massive amount of required computing resources leads to the development of strategies for improving the speed of performance estimation, which we shall now explore.

3.4.1 Low fidelity estimator

Lower fidelities of real performance after full training can be used to estimate performance (also denoted as proxy metrics). Among lower fidelities techniques we can find training on a subset (Klein et al., 2017), with a lower-resolutions (Chrabaszcz et al., 2017) or with a smaller model with fewer filters per layers or fewer cells (Real et al., 2019; Zoph et al., 2018). Recently, (Mellor et al., 2021) developed a new approach for estimating architecture performance without even needing to train it. The main idea behind their method is that there is a relationship between the network local linear maps (ReLU output) correlation and its overall performance. Such techniques help to drastically reduces the required amount of computational resources, they also inject bias into the estimation because performance is usually underestimated. This may not be a concern if the search technique is limited to rating distinct architectures and the relative ranking remains steady. Recent results (Zela et al., 2018), however, show that when the difference between the cheap approximations and the "full" evaluation is too large, this relative ranking might vary substantially.

3.4.2 Extrapolation

Some other method for evaluating an architecture performance is to use learning curve extrapolation and practice early stopping in the not promising case. (Domhan et al., 2015) presents a learning-curve model that is a weighted mixture of a collection of parametric curve models chosen from the literature, allowing the network performance to be predicted. In Progressive NAS (PNAS) (Liu et al., 2018), the authors propose to not use learning curve extrapolation, but instead support predicting performance based on architectural/cell features and extrapolating to architectures/cells larger than those seen during training. The fundamental issue for forecasting neural architecture performance is that, in order to accelerate the search process, make good predictions in a reasonably large search space must be produced based on relatively few evaluations.

3.4.3 One-Shot Architecture Search

In this kind of methods, all architectures are treated as separate subgraphs of a supergraph (the one-shot model), and weights are shared between architectures that share edges with this supergraph, we can also define that as weight sharing. By these techniques only the weights of a single one-shot model must be optimized, and architectures (which are simply subgraphs of the supergraph model) can then be evaluated without additional training by inheriting trained weights from the one-shot model. Unfortunately, the one-shot model has a bias in that it underestimates the real performance of the best architectures; yet, it permits ranking architectures, which would be sufficient if the predicted performance significantly correlates with the actual performance. However, it is currently unclear whether or not this is the case (Yu et al., 2019).

There are variants related to the one-shot method such as, the weights sharing technique used in (Pham et al., 2018) where parameters are shared among child architecture, network morphism where also the child network can inherit from previous network. In DARTS, the technique used can be related to the one-shot model, as it optimizes all weight with a continuous relaxation of the search space, different from that SNAS (Xie et al., 2019b) propose to optimize a distribution over those operations, using the concrete distribution (Maddison et al., 2017) and reparametrization trick (Kingma and Welling, 2014) to make the discrete distribution differentiable. (Bender et al., 2018) train a one-shot model only once, and demonstrate that simply deactivating sections of the model stochastically during training with route dropout is sufficient to achieve good results.

To achieve that, they use a fixed distribution, which might indicate that only a carefully selected distribution and weights sharing are required to have a good performing one-shot NAS model.

Despite the many advantages of one-shot NAS, there are a few drawbacks. The first, which is also related to the domain of NAS, is that the supergraph incorporates the human a prior, which can limit the possible representation of subgraphs. In addition, hardware limitations still exist, indeed if the entire supergraph must be loaded on GPU there will be a limitation in the size of the network, or it will require large resources. While weight-sharing methods have considerably decreased the processing resources necessary for NAS, it is still uncertain what biases they add into the search (Bender et al., 2018). For example, an initial bias toward investigating specific areas of the search space more than others may lead to the parameters of the model one-shot being more tailored to specific structures, thus reinforcing the search bias toward those portions of the search space.

3.5 Summary

In this chapter, we have covered Neural Architecture Search as well as the most common methodologies established in this field, mostly developed for image classification. On the one hand, many NN models have been developed for this task under the supervised learning framework, and they are not easily outperformed by NAS. As a result, we believe it is important to extend beyond image classification challenges by using NAS to less explored research fields or under new train settings (e.g. semi-supervised, self-supervised). Some works have started to explore new tasks such as semantic segmentation (Nekrasov et al., 2019a), transfer learning (Wong et al., 2018) or optimization of existing neural network (Rawal and Miikkulainen, 2018; Tan and Le, 2019). In addition, (Liu et al., 2020) have recently explored the potential of self-supervised architecture search (i.e., without using labels).

To round out the concept of new training settings, it is not uncommon in real-world applications to have two sets of data, one of which is not annotated because annotation is a time-consuming and possibly costly operation. Despite the fact that it is not annotated, it contains information that a neural network can use. As a result, there are numerous approaches of training a neural network with partially annotated data. This is what will be discussed in the following chapter, from two perspectives that employ unannotated data in distinct ways.

4

Learning from partially annotated data

Semi-supervised algorithms aim to learn prediction functions from a small set of labeled data and a large set of unlabeled data. Since this framework is applicable in a wide range of applications, it has piqued the interest of both academics and industry. Among the existing techniques, self-training methods have undoubtedly attracted greater attention in recent years. From another perspective, self-supervised methods also propose to leverage unlabeled data, by using dedicated transformation or as pretraining. In this chapter, we introduce self-training and self-supervised methods.
This chapter present works from the following contributions:

- Self-Training: A Survey [1]

Contents

4.1 Introduction

Semi-supervised learning boils down to the fundamental dilemma of how to infer from partially labeled data, and it has become a predominant technique in Machine Learning. Dealing with circumstances where there are a small number of labeled training points together with a large number of unlabeled examples is important to a range of applications where obtaining labeled data is costly, such as image classification and segmentation. Semi-supervised learning as a subject uses a variety of methods and shows, on a small scale, the intricate machinery found in numerous disciplines of machine learning, such as Neural Networks.

Similar to semi-supervised learning, new neural network-based methods focusing on the use of unannotated samples have emerged. This field, known as self-supervised learning, takes advantage of the vast volumes of unannotated data accessible to give efficient ways for label-free learning and ever-increasing performance. These techniques are employed as efficient label-free pretraining methods, which are then fine-tuned with a small number of annotated samples. Next, a traditional self-training method can be employed (Chen et al., 2020b).

The remainder of this chapter is organized as follows. In the next section, we introduce the semi-supervision paradigm. Next, in Section 4.3 we present the framework and notations used throughout the chapter. In Section 4.4, we go over the self-training method in detail, covering pseudo-labeling and its variants, as well as providing some insights into current theoretical studies. Other related approaches are described in Section 4.5. An introduction to self-supervised approaches is given in Section 4.6. Finally, views and future prospects are discussed in Section 4.7.

4.2 Central hypothesis and main approaches

4.2.1 Semi-supervised central hypothesis

Smoothness is a key assumption in semi-supervised learning, which stipulates that two instances in a high density region should have the same class labels (Chapelle et al., 2010; Amini and Usunier, 2015). This means that if two points belong to the same group or cluster, their class labels will almost always be the same. On the other hand, if they are separated by a low density zone, their desired labels should be different. Unlabeled training data may thus contribute in finding the partition border more effectively than labeled training examples if the instances of the same class form a partition.

4.2.2 Three main semi-supervised learning families

The smoothness hypothesis is adapted in three main families of semi-supervised learning techniques as a consequence of the assumption used in diverse contexts.

The working premise of generative techniques is data clustering, which use a mixture model to assign class labels to partitions based on the labeled data they contain (Nigam et al., 2006; Kingma et al., 2014). The cluster assumption, which underpins these approaches, asserts that if two examples are in the same group, they are likely to be in the same class. This hypothesis may be explained as follows: if a group is formed by a large number of instances, it is rare that they belong to different classes. This does not imply that a class is constituted by a single group of examples, but rather that two examples from distinct classes are unlikely to be found in the same cluster.

If we consider the partitions of instances to be high density areas, a form of the cluster assumption known as low density separation entails determining the decision boundary over low density regions, and it constitutes the basis of discriminant techniques. The main difference between generative and discriminant techniques is that discriminant approaches find directly the prediction function without making any assumption on how the data are generated (Amini and Gallinari, 2002; Oliver et al., 2018).

Density estimation is often based on a notion of distance, which for high dimensional spaces may also become meaningless. A third hypothesis, known as the manifold assumption, stipulates that in high-dimensional spaces, instances reside on low-dimensional topological spaces that are locally Euclidean, which is supported by a variety of semi-supervised models called graphical approaches (Belkin and Niyogi, 2004; Chong et al., 2020).

4.2.3 Compatibility

Although semi-supervised algorithms have been successfully applied in many situations, there have been cases where unlabeled data have been shown to have no effect on the performance of a learning task (Singh et al., 2009). Several attempts have been made to investigate the value of unlabeled data in the training process (Castelli and Cover, 1995; Li and Zhou, 2011), and the capacity of semi-supervised learning approaches to generalize (Rigollet, 2007; Maximov et al., 2018). The bulk of these studies are founded on the notion of *compatibility* defined in (Balcan and Blum, 2006), and they strive to exhibit the connection between the marginal data distribution and the target function to be learned. According to these findings, unlabeled data will only be beneficial for training if such a relationship exists.

In generative approaches, the marginal distribution is viewed as a mixture of class conditional distributions, and when compared to the supervised case, semi-supervised learning has been shown to achieve lower finite-sample error bounds in some general cases, or a faster rate of error convergence in others (Castelli and Cover, 1995; Rigollet, 2007; Maximov et al., 2018; Singh et al., 2009). In this line, (Ben-David et al., 2008) showed that accessing the marginal distribution on unlabeled training data would not provide sample size guarantees superior to those obtained by supervised learning unless very strong assumptions about conditional distribution on class labels are done.

For graph-based approaches, (Niyogi, 2013) provided a context in which such algorithms may be studied and perhaps justified; the key finding of the study is that unlabeled data can aid learning in some situations by defining explicitly the manifold.

Finally, discriminant approaches mostly embed a margin maximization method that searches the decision boundary in low-density regions by pushing it from the unlabeled data (Joachims, 1999). In this survey, we focus on self-training algorithms that follow this principle by assigning pseudo-labels to high-confidence unlabeled training examples and include these pseudo-labeled samples in the learning process.

4.3 Framework and notations

We consider classification problems where the input and the output spaces are respectively $\mathcal{X} \subseteq \mathbb{R}^d$ and $\mathcal{Y}$. We further assume available a set of labeled training examples $S = (\mathbf{x}_i, y_i)_{1 \leqslant i \leqslant m} \in (\mathcal{X} \times \mathcal{Y})^m$ generated from a joint probability distribution $\mathbb{P}(\mathbf{x}, y)$ (denoted as $\mathcal{D}$) and a set of un-

labeled training examples $X_{\mathcal{U}} = (\mathbf{x}_i)_{m+1 \leqslant i \leqslant m+u} \in \mathcal{X}^u$ supposed to be drawn from the marginal distribution $\mathbb{P}(\mathbf{x})$.

The classic case corresponds to $m \ll u$, and the issue is thrown into the unsupervised learning framework if S is empty. The opposite extreme scenario is when $X_{\mathcal{U}}$ is empty and the problem is reduced to supervised learning. Given a hypothesis set of functions $\mathcal{H}$ mapping $\mathcal{X}$ to $\mathcal{Y}$; the learner receives a labeled set S and an unlabeled set $X_{\mathcal{U}}$ and outputs a hypothesis $h \in \mathcal{H}$ which is assumed to have a generalization error $R(h) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[\mathbb{1}_{h(\mathbf{x})\neq y}]$ smaller than if just S was used to find the prediction function, whereby $\mathbb{1}_\pi$ denotes the indicator function equal to 1 if the predicate π is true and 0 otherwise.

In practice, classifiers are defined based on a scoring function f from a class of functions $\mathcal{F} = \{f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}\}$, and for an example $\mathbf{x}$ the corresponding classification function h outputs the class for which the score of f is the highest:

$$h(\mathbf{x}) = \text{argmax}_{y\in\mathcal{Y}} f(\mathbf{x}, y).$$

We define the margin $\rho_f(\mathbf{x}, y)$ of a function f for an example $\mathbf{x} \in \mathcal{X}$ and a class $y \in \mathcal{Y}$ as:

$$\rho_f(\mathbf{x}, y) = f(\mathbf{x}, y) - \max_{y'\neq y} f(\mathbf{x}, y').$$

In the binary case, $\mathcal{Y} = \{-1, +1\}$, we define the unsigned margin of a classification function $f \in \mathcal{F}$ over an example $\mathbf{x}$ (Buc et al., 2001; Amini et al., 2008) as:

$$m_f(\mathbf{x}) = |\rho_f(\mathbf{x}, +1)|.$$

In the multi-class classification case, $\mathcal{Y} = \{1, \ldots, K\}$, the unsigned margin (Buc et al., 2001; Feofanov et al., 2019) is defined as

$$m_f(\mathbf{x}) = \sum_{y\in\mathcal{Y}} f(\mathbf{x}, y)\rho_f(\mathbf{x}, y).$$

The maximization of the unsigned margin tends to find a decision boundary that passes through low density regions, and hence follows the low density separation assumption.

Based on this idea, self-training algorithms define a pseudo-labeling strategy for assigning pseudo-labels to the examples of $X_{\mathcal{U}}$. This strategy can be characterized by a function, called *pseudo-labeler*:

$$\Phi_\ell : \mathcal{X} \times \mathcal{F} \to \mathcal{X} \times \mathcal{Y}.$$

We denote $\tilde{y}$ the pseudo-label of an unlabeled $\mathbf{x} \in X_{\mathcal{U}}$ assigned by Φ_ℓ and $X_{\tilde{\mathcal{Y}}}$ the set of pseudo-labeled examples.

4.4 Self-Training approaches

Self-training, also known as decision-directed or self-taught learning machine, is one of the earliest approach in semi-supervised learning (Scudder, 1965; Fralick, 1967) that has risen in popularity in recent years.

This wrapper algorithm starts by learning a supervised classifier on the labeled training set S. Then, at each iteration, the current classifier selects a part of the unlabeled data, $X_{\tilde{\mathcal{Y}}}$, and assigns pseudo-labels to them using the classifier predictions. These pseudo-labeled unlabeled examples are removed from $X_{\mathcal{U}}$ and a new supervised classifier is trained over $S \cup X_{\tilde{\mathcal{Y}}}$, by considering these pseudo-labeled unlabeled data as additional labeled examples. To do so, the classifier $h \in \mathcal{H}$ that

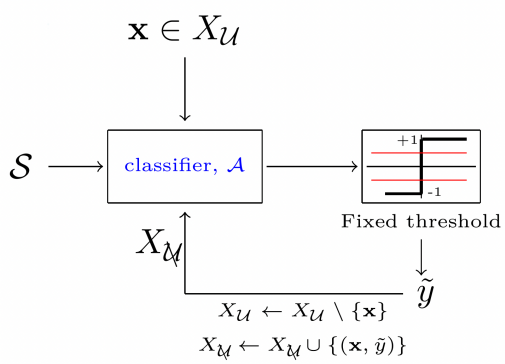


Figure 4.1: Illustration of the traditional self-learning approach.

is learned at the current iteration is the one that minimizes a regularized empirical loss over S and $X_{\mathcal{U}}$:

$$\frac{1}{m} \sum_{(\mathbf{x},y)\in S} \ell(h(\mathbf{x}), y) + \frac{\gamma}{|X_{\mathcal{U}}|} \sum_{(\mathbf{x},\tilde{y})\in X_{\mathcal{U}}} \ell(h(\mathbf{x}), \tilde{y}) + \lambda\|h\|^2,$$

where $\ell : \mathcal{Y} \times \mathcal{Y} \to [0, 1]$ is an instantaneous loss most often chosen to be the cross-entropy loss, γ is a hyperparameter for controlling the impact of pseudo-labeled data in learning, and λ is the regularization hyperparameter. This process of pseudo-labeling and learning a new classifier continues until the unlabeled set $X_{\mathcal{U}}$ is empty or there is no more unlabeled data to pseudo-label. The self-training algorithm process is illustrated in fig.4.1 and the pseudocode of is shown in Algorithm 1.

4.4.1 Pseudo-labeling strategies

At each iteration, the self-training selects just a portion of unlabeled data for pseudo-labeling, otherwise, all unlabeled examples would be pseudo-labeled after the first iteration, which would actually result in a classifier with performance identical to the initial classifier (Chapelle et al., 2010). Thus, the implementation of self-training arises the following question: how to determine the subset of examples to pseudo-label?

A classical assumption, that stems from the low density separation hypothesis, is to suppose that the classifier learned at each step makes the majority of its mistakes on observations close to the decision boundary. In the case of binary classification, preliminary research suggested assigning pseudo-labels only to unlabeled observations for which the current classifier is the most confident (Tür et al., 2005). Hence, considering thresholds θ^- and θ^+ defined for respectively the negative and the positive classes, the pseudo-labeler assigns a pseudo-label $\tilde{y}$ to an instance $\mathbf{x} \in X_{\mathcal{U}}$ such that:

$$\tilde{y} = \begin{cases} +1, & \text{if } f(\mathbf{x}, +1) \geqslant \theta^+, \\ -1, & \text{if } f(\mathbf{x}, -1) \leqslant \theta^-, \end{cases} \tag{4.1}$$

and $\Phi_\ell(f, \mathbf{x}) = (\mathbf{x}, \tilde{y})$. An unlabeled example $\mathbf{x}$ that does not satisfy the conditions (4.1) is not pseudo-labeled; i.e. $\Phi_\ell(f, \mathbf{x}) = \emptyset$.

Intuitively, thresholds should be set to high absolute values, as pseudo-labeling examples with low confidence would increase chances of assigning wrong labels. However, thresholds of very high value imply excessive trust in the confidence measure underlying the model, which, in reality, can be biased due to the small labeled sample size. Using several iterations makes also the situation more intricate, as at every iteration the optimal threshold might be different.

One way to select the thresholds is to set them equal to the average of respectively positive and negative predictions (Tür et al., 2005). In this line, and in the context of multi-class classification,

Alg. Self-Training

Input : $S = (\mathbf{x}_i, y_i)_{1 \leqslant i \leqslant m}$, $X_{\mathcal{U}} = (\mathbf{x}_i)_{m+1 \leqslant i \leqslant m+u}$.
$k \leftarrow 0$, $X_{\mathcal{Y}} \leftarrow \emptyset$.
repeat
  Train $h^{(k)}$ on $S \cup X_{\mathcal{Y}}$ ▷ Learning the classifier
  $\Pi \leftarrow \{\Phi_\ell(h^{(k)}, \mathbf{x}), \mathbf{x} \in X_{\mathcal{U}}\}$ ▷ Pseudo-labeling
  $X_{\mathcal{Y}} \leftarrow X_{\mathcal{Y}} \cup \Pi$
  $X_{\mathcal{U}} \leftarrow X_{\mathcal{U}} \setminus \{\mathbf{x} \mid (\mathbf{x}, \tilde{y}) \in \Pi\}$
  $k \leftarrow k + 1$
until $X_{\mathcal{U}} = \emptyset \vee \Pi = \emptyset$
Output : $h^{(k)}$

Lee, 2013 used Neural Networks as the supervised classifier and chose the most confident class to infer pseudo-labels for unlabeled data using the current model' outputs. The pseudo-labeled examples were then added to the labeled training set and treated similarly as labeled examples.

Zou et al., 2018 adapted the idea of Tür et al., 2005 for multi-class classification by not choosing thresholds but rather fixing a proportion p of the most confident unlabeled data to be pseudo-labeled and then increasing this proportion at each iteration of the algorithm until $p = 0.5$ was reached.

Following this idea, (Cascante-Bonilla et al., 2021; Zhang et al., 2021) proposed an adaptation of curriculum learning to pseudo-labeling, which entails in learning a model using easy-to-learn observations before moving on to more complex ones. The principle is that at the step k of the algorithm, the pseudo-labeler selects unlabeled examples having predictions that are in the $(1 - \alpha_k)^{th}$ percentile of the distribution of the maximum probability predictions assumed to follow a Pareto distribution, and where $\alpha_k \in [0, 1]$ is a hyperparameter that varies from 0 to 1 in increments of 0.2

Considering the distribution of predictions over unlabeled data, and the voted-majority classifiers, such as Random Forest or Adaboost (Schapire et al., 1997), it is possible to automatically choose a threshold for pseudo-labeling. Formally, the learning of a voted-majority classifier with partially labeled data can be defined as follows. After observing the training set $S \cup X_{\mathcal{Y}}$, the task of the learner is to choose a posterior distribution Q over a set of hypothesis $\mathcal{H}$ such that the Q-*weighted majority vote* classifier B_Q defined by:

$$\forall \mathbf{x} \in \mathcal{X}, B_Q(\mathbf{x}) = \text{argmax}_{y \in \mathcal{Y}} \mathbb{E}_{h \sim Q} \left[\mathbb{1}_{h(\mathbf{x})=y} \right],$$

will have the smallest possible risk on examples of $X_{\mathcal{U}}$. The associated *Gibbs* classifier, G_Q, is defined as the random choice of a classifier h according to the distribution Q, and its error over an unlabeled set $X_{\mathcal{U}}$, is given by:

$$\hat{R}_u(G_Q) = \frac{1}{u} \sum_{\mathbf{x}' \in X_{\mathcal{U}}} \mathbb{E}_{h \sim Q}[\mathbb{1}_{h(\mathbf{x}') \neq y'}],$$

where, for every unlabeled example $\mathbf{x}' \in X_{\mathcal{U}}$ we refer to y' as its true unknown class label. For binary and multi-class classification, (Amini et al., 2008; Feofanov et al., 2019) showed that a tight upper-bound on the Gibbs's classifier risk that holds with high probability over the random choice of S and $X_{\mathcal{U}}$, guarantees a tight bound on the error of the Bayes classifier over the unlabeled training set:

$$\hat{R}_u(B_Q) = \frac{1}{u} \sum_{\mathbf{x}' \in X_{\mathcal{U}}} \mathbb{1}_{B_Q(\mathbf{x}') \neq y'}$$

This bound is mainly based on the distribution of predictions over unlabeled data and the derivations can be extended to bound the risk of voted majority classifiers having margins greater than a threshold θ, $\hat{R}_{u\wedge\theta}(B_Q)$, defined as:

$$\hat{R}_{u\wedge\theta}(B_Q) = \frac{1}{u}\sum_{\mathbf{x}'\in X_{\mathcal{U}}} \mathbb{1}_{B_Q(\mathbf{x}')\neq y'\wedge m_{B_Q}(\mathbf{x}')>\theta}.$$

One of the practical aspects that arises from this result is the possibility to specify a threshold θ which minimizes an upper-bound of the conditional risk of a voted majority classifier B_Q over the unlabeled training set, $X_{\mathcal{U}}$, defined as:

$$\hat{R}_{u|\theta}(B_Q) = \frac{\hat{R}_{u\wedge\theta}(B_Q)}{\frac{1}{u}\sum_{\mathbf{x}\in X_{\mathcal{U}}} \mathbb{1}_{m_{B_Q}(\mathbf{x})\geqslant\theta}},$$

where the denominator is the proportion of the unlabeled examples with the confidence higher than the threshold θ, and the numerator is the joint Bayes risk on $X_{\mathcal{U}}$. Thus, the criterion can be interpreted as a trade-off between the number of examples going to be pseudo-labeled and the error they induce. Furthermore, these bounds are shown to be tight in the case where the majority vote classifier makes its error mostly on low margin regions.

Feofanov et al., 2019 demonstrated that this technique outperforms conventional fixed-threshold pseudo-labeling strategies on different multiclass classification problems.

4.4.2 Self-training with two classifiers

In the wake of works utilizing only a single classifier in self-training algorithms, new studies have been proposed with the use of two classifiers, where each model learns on the output of the other (Xie et al., 2020b; Chen et al., 2021; Karamanolakis et al., 2021). Most of these techniques are based on the idea of consensus in predictions between two classifiers and were inspired by the seminal work of (Blum and Mitchell, 1998) who proposed the co-training algorithm.

In co-training, examples are defined by two modalities that are comparable but not entirely correlated. Each view of an example is expected to contain complementary information about the data and if there are enough labeled training data, each of them is supposed to be sufficient for learning. The goal of learning is to train a classifier on each view by first initializing it with the labeled training set that is available. Then, one of the classifiers assigns pseudo-labels to unlabeled data, which the other one will use to learn. Following training, the classifiers switch roles, with the learned classifier assigning pseudo-labels to unlabeled examples, which will then be used to train the first classifier. As for self-training algorithms with a single classifier, this procedure continues until there are no more unlabeled instances to be pseudo-labeled. In practice, several studies artificially generated the two modalities for classification problems where examples are *mono-viewed* and described by a vector representation. These approaches create the two modalities out of one by selecting at random the set of features that should correspond to each modality from the initial set of features; and their efficiency was empirically proved on various applications.

Co-training can thus be thought of as a form of self-training; as rather than training a classifier on its own previous predictions; each classifier is trained on the predictions of another classifier that was trained on the predictions of the former. Without splitting the input feature set, (Chen et al., 2021) proposed Cross Pseudo Supervision for semantic segmentation in images, illustrated in fig. 4.2. This method employs two neural networks as supervised classifiers having the same images as inputs. Each neural-network is learned at every mini-batch by considering the pseudo-labels generated by the other network for unlabeled instances as ground-truths.

The learnability of co-training was studied under the PAC framework (Valiant, 1984), which also accounts for noise in the class labels of unlabeled examples caused by pseudo-labeling. The injection of noisy labels in the pseudo-labeling step is in fact inherent to any self-training algorithm. Taking into account noisy labels in training, a model was first considered in supervised

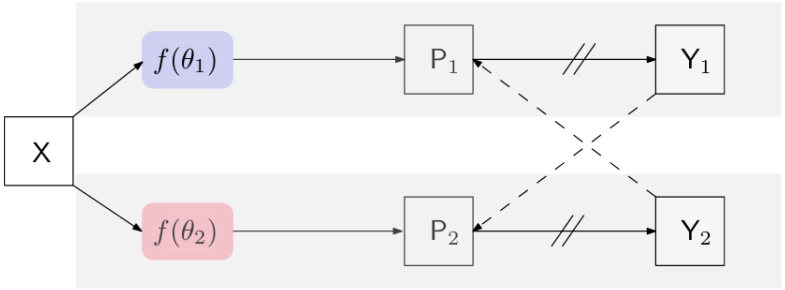


Figure 4.2: Illustration of the Cross Pseudo Supervision used in (Chen et al., 2021). $f(\theta_1)$ and $f(\theta_2)$ are two networks differently initialized, P_1 and P_2 are the two corresponding prediction. Finally, the two one-hot encoded prediction Y_1, Y_2 are swapped and used as ground truth by the other network. Illustration from (Chen et al., 2021)

learning, following the paradigm of learning with an imperfect supervisor in which training data contains an unknown portion of imperfect labels (Natarajan et al., 2013; Han et al., 2018). Most of these studies tackle this problem from an algorithmic point of view, employing regularization (Miyato et al., 2019) or estimating mislabeling errors by modeling the transition probability between noisy and true labels (Patrini et al., 2017). Other types of noise considerations for image (Xie et al., 2020b) and text classification (Karamanolakis et al., 2021) have recently been proposed, which involve mostly perturbing the input examples (data augmentation in images, for example) and constraining the two models involved in self-training to produce the same predictions on unlabeled examples. The figure 4.3 illustrate the noisy process used in (Xie et al., 2020b), a similar process can be transposed to the language processing.

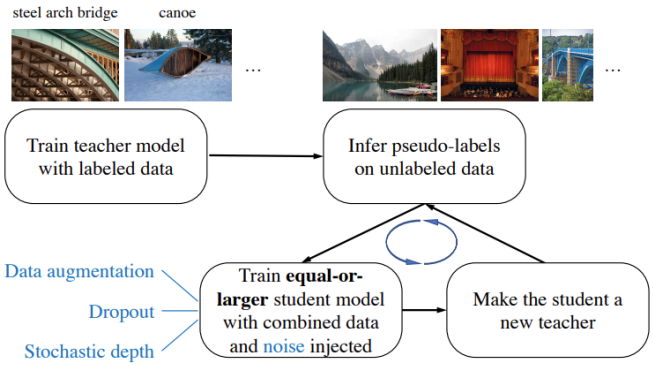


Figure 4.3: Representation of the noisy student pipeline. This pipeline differs from the original self-learning pipeline in the sense that here, during the second learning phase, noise is injected. This noise can be of different natures, such as strong data augmentations, drop out or stochastic depth. Illustration from (Xie et al., 2020b)

4.4.3 Theoretical studies

Several studies have recently looked into the theoretical properties of self-training algorithms. In this line, Wei et al., 2021 suggest a new concept of *expansion* defined as the quantity of data dispersion in an neighbor example, where the term *neighbor* refers to adding adversarial perturbations (Miyato et al., 2019) or augmentations (Sohn et al., 2020) to the example. The study establishes distributional guarantees of self-training when the label distribution meets such expansion properties and classes are suitably separated according to neighbors. The study generates finite sample

bounds for Deep Neural Networks (DNNs) by combining generalization bounds with DNN generalization bounds. Experiments with a Generative Adversarial Network (GAN) are also used to verify the expansion assumption.

Frei et al., 2022 examine a self-training algorithm with linear models for the binary classification using gradient-based optimization of the cross-entropy loss after supervised learning with a few samples. The classifier is a mixture model with concentration and anti-concentration properties. The authors show that utilizing $O(d/\epsilon^2)$ unlabeled observations in the self learning algorithm, with d the number of input variables, suffices to achieve the classification error of the Bayes-optimal classifier up to an ϵ error if the initial pseudo-labeling strategy has a classification error smaller than an absolute constant C_{err}. Furthermore, the authors demonstrate that a constant number of labeled examples is sufficient for optimal performance in a self-training algorithm by demonstrating that using only $O(d)$ labeled examples, the standard gradient descent algorithm can learn a pseudo-labeling strategy with a classification error no more than C_{err}. Zhang et al., 2022 study the generalization ability of self-training in the case where the base classifier is a two-layer neural network with the second layer weights all fixed to one, and assuming that the ground truth is realizable, the labels are observed without noise, and the labeled and unlabeled instances are drawn from two isotropic Gaussian distributions. The authors show that, given some plausible assumptions about the initial point and the amount of unlabeled training examples, the algorithm converges to the ground truth with fewer observations than needed when no unlabeled data is provided. The reader can refer to Zhong et al., 2017 for a broader context. Zhang et al., 2022 extend their main result to a more general setting, where it is shown that the model still converges towards a given convex combination of the ground truth and the initial point, and is guaranteed to outperform the initial supervised model, without fixing any requirement on the number of labeled training examples.

4.5 Related approaches

In semi-supervised learning, there are probably two main other areas of research that are related to self-training. The first, referred to as *consistency learning*, uses classifier predictions over unlabeled data as a confidence indicator and constrains model outputs to be comparable for similar unlabeled examples without assigning pseudo-labels. The second method, known as *transductive learning*, is based on the low density separation assumption and tends to give class labels for only the set of unlabeled training samples. This framework is driven by some applications for which it is sufficient to properly predict the outputs of instances of an unlabeled or test set rather than learning a general rule.

4.5.1 Consistency-based approaches

Early studies in this line, see for example (Zhu et al., 2003) for binary classification, were proposed to learn a single classifier defined from a scoring function $f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ penalized for quick changes in its predictions. The similarity matrix $\mathbf{W} = [W_{ij}]_{\substack{1 \leqslant i \leqslant u \\ 1 \leqslant j \leqslant u}}$, constructed over the unlabeled training data, is used to measure the similarity between instances. The penalization is mostly expressed as a regularization term in the learning objective. As an example, adapting the work of (Zhu et al., 2003) to multiclass classification, the penalization term can be written as:

$$\Omega_{\mathbf{W}}(X_{\mathcal{U}}) = \sum_{i,j=1}^{u} W_{ij} \| f(\mathbf{x}_{m+i}, .) - f(\mathbf{x}_{m+j}, .) \|^2$$

where for a given example $\mathbf{x}$, $f(\mathbf{x}, .) = (f(\mathbf{x}, k))_{k \in \mathcal{Y}}$ is the vector class predictions of f. In terms of learning, $\Omega_{\mathbf{W}}$ can be seen as a regularization term, constraining the model to have the same predictions on similar unlabeled instances.

Other types of penalization have been studied in the literature. Maximov et al., 2018 suggested an approach that partitions partially labeled data and then uses labeled training samples to identify dense clusters having predominant classes with a fraction of non-predominant classes below a given threshold. In this situation, the proposed penalization term measures the learner's inability to predict the predominant classes of the identified clusters, which in turn constrains the supervised classifier to be consistent with the structure of the dense clusters.

Later, without explicitly stating a penalization term, consistency learning was extended to cases with two classifiers. The Mean-Teacher approach (Tarvainen and Valpola, 2017) is perhaps one of the earliest popular techniques that have been proposed in this context. This method employs Neural Networks (NNs) as supervised classifiers, and it is based on the assumption that two close models with the same input should make the same prediction. One of the models is called the *teacher*, while the other is referred to as the *student*. These two NN models are structurally identical, and their weights are related in that the teacher's weights are an exponential moving average (Laine and Aila, 2017) of the student' weights. In this scenario, the student model is the only one that is trained over the labeled training set, and the consistency loss is computed between the teacher's probability distribution prediction and the student's one using the mean square error or the Kullback-Leibler divergence. An Illustration of the Mean-Teacher technique is given by figure 4.4.

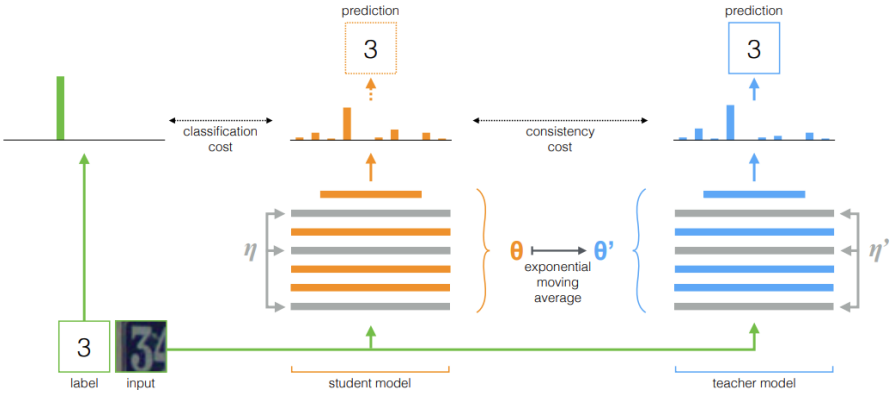


Figure 4.4: Illustration of the mean teacher operating scheme. The student network is the only networks which have access to the label. The input is given to the student and teacher, where the teacher has the same structure as the student, but have an exponential moving average of the student weights. Moreover, each network can add perturbations η, η' such as dropout. Illustration from (Tarvainen and Valpola, 2017)

Other studies refined the Mean-Teacher approach with a data-augmentation technique by combining two images with random patches to improve prediction consistency (French et al., 2020a; Xie et al., 2020a) The process used by (Xie et al., 2020a), is also a consistency-based approach, despite here they only use one network. More recently, (Olsson et al., 2021) advocated using network predictions as a guidance to retain object boundaries during the mixing phase of image segmentation. They also employed the *teacher* predictions as pseudo-labels for the student's learning, replacing the consistency regularization term with an empirical cross-entropy error estimated over the pseudo-labeled data.

4.5.2 Transductive learning

The goal of transductive learning, as previously stated, is to assign pseudolabels to samples from an unlabeled training set, $X_{\mathcal{U}}$. As this set is finite, the considered function class $\mathcal{F}$, for finding the transductive prediction function, is also finite. $\mathcal{F}$ can be defined using a nested structure according to the structural risk minimization principle, $\mathcal{F}_1 \subseteq \mathcal{F}_2 \subseteq \ldots \subseteq \mathcal{F}$ (Vapnik, 1998). Transductive techniques often employ the distribution of unsigned margins of unlabeled examples to guide the search for a prediction function, limiting it to following the low density separation assumption in order to find the best function class among the current ones.

Transductive approaches also assume that the function class structure should reflect prior knowledge of the learning problem at hand, and that it should be built in such a way that the correct prediction of class labels of labeled and unlabeled training examples is contained in a function class $\mathcal{F}_j$ of small size with a high probability. In particular, (Derbeko et al., 2004) show that with high probability the error on the unlabeled training set of a function from a class function $\mathcal{F}_j$ is bounded by its empirical loss on the labeled training set plus a complexity term that depends on the number of labeled examples m, the number of unlabeled examples u, and the size of the class function $\mathcal{F}_j$.

The Transductive Support Vector Machines (TSVM) (Joachims, 1999) developed for the binary case is based on this paradigm, and is looking for the optimal hyperplane in a feature space that separates the best labeled examples while avoiding high density areas. TSVM does this by building a structure on a function class $\mathcal{F}$ and sorting the outputs of unlabeled samples by their margins. The solutions to the associated optimization problem are the pseudo-labels of unlabeled examples for which the resulting hyperplane separates the examples of both labeled and unlabeled training sets with the largest margin. The main difference with self-training is that in TSVM, pseudo-labeled examples will not be used to train the classifier again.

(Shi et al., 2018) extended this idea to the multiclass classification case with Neural Networks. Similar to TSVM, class labels of unlabeled examples are treated as variables, and the algorithm tries to determine their optimal values, along with the optimal NNs parameter set get by minimizing a cross-entropy loss estimated over both labeled and unlabeled training sets through an iterative training process. The authors employ the MinMax Feature regularization to constrain the neural network to learn features of same-class images to be close, and features of different classes to be separated by a preset margin, in order to overcome incorrect label estimations on outliers and noisy samples.

4.6 Self-supervised approaches

When you give supervised learning a task and enough labels, it can solve it quite well. Good performance normally necessitates a large number of labels, but gathering manual labels is costly (e.g., ImageNet (Deng et al., 2009)) and difficult to scale up. Given that the amount of unlabeled data (e.g., free text, all photos on the Internet) is significantly greater than the restricted number of human-curated tagged datasets, it seems wasteful not to use them. Unsupervised learning, on the other hand, is difficult and typically performs significantly less efficient than supervised learning, for comparable sample size.

The main reason of the rapid use of self-supervised learning, is producing a dataset with clear labels is hard and expensive, but unlabeled data is constantly generated. A method to make use of this much bigger amount of unlabeled data is to appropriately design the learning objectives so that the data itself can supervise you. The self-supervised task, also known as the pretext task,

leads to the supervised loss function. However, we are frequently unconcerned about the outcome of this created task. We are more interested in the learned intermediate representation, with the assumption that it will have good semantic meanings and will be useful in a range of practical downstream tasks. Indeed, as previously said, self-supervision approaches are generally used to pre-train networks before focusing on a downstream task that the user is interested in. We expect that by applying these strategies, the model will be able to learn high-quality latent variables for real-world applications such as building an object classifier with very few labeled examples, like illustrated in fig.4.5.

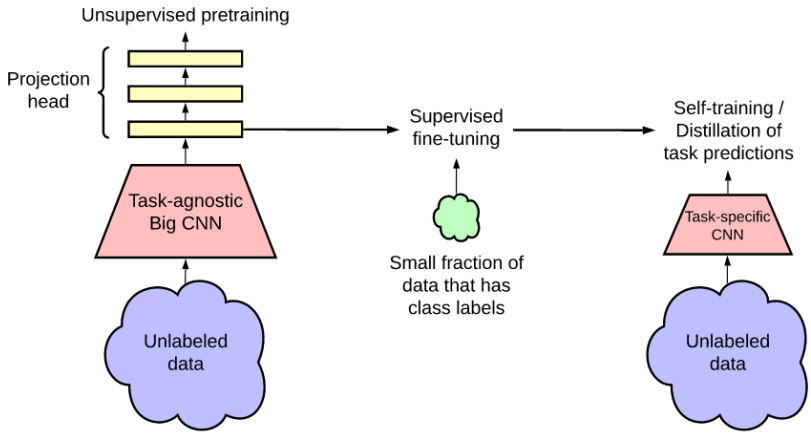


Figure 4.5: Illustration of the pipeline proposed by (Chen et al., 2020b), which involve self-supervised learning as pretraining and the small fraction of supervised data as fine-tuning. Illustration from (Chen et al., 2020b)

4.6.1 Image-based approaches

Many ideas for self-supervised image representation learning have been proposed. A standard image classification approach is to train a model on one or more pretext tasks with unlabeled photos, and then feed one intermediate feature layer of this model to a classifier.

The first proposed methods are based on image distortion, indeed slightly distorted images keep the same semantic meaning as the original image, so the learned representations should be invariant to distortions. Dosovitskiy et al., 2014 proposes to create a training dataset called "surrogate". This new dataset contains a transformed version of the original dataset, where random transformations have been applied. All these new images, created from a single image, belong to the same surrogate class. The goal of the method (i.e. the pretext task) is to discriminate between a set of surrogate classes. Some other task, such as rotation (Gidaris et al., 2018), were also used. Here, each image is randomly rotated by a multiple of $90°$ (i.e. $\{0°, 90°, 180°, 270°\}$). Then the model is trained to solve a 4-class classification problem. Using this pretext task, the model is force to learn semantic concepts, by recognizing high level object, such as heads or eyes and their positions. The figure 4.6 illustrates the rotation pretext task proposed by (Gidaris et al., 2018).

During the same years, other researches have followed another approach by relying instead on image patches. The pretext task proposed by (Doersch et al., 2015) is defined as guessing the relative position of two random patches from a single image. In order to determine the relative position of parts, a model must grasp their spatial context. To works, these methods apply a 3x3 grid to the images, but only use two patches from this grid. To take advantage of other patches, and make this task more difficult, Noroozi and Favaro, 2016 designed a jigsaw puzzle as pretext task, the model learn how to place the shuffled patches back to the original locations, the fig. 4.7 illustrate this process.

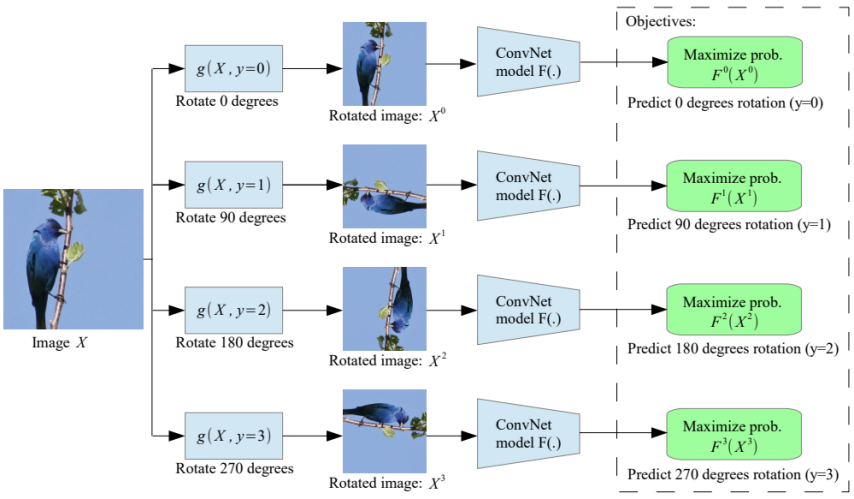


Figure 4.6: Illustration of the rotation pretext task proposed by (Gidaris et al., 2018). The unlabeled image X is randomly rotated among 4 predefined angles, which is associated as label for this image. The task is to predict the rotation angle. Illustration from (Gidaris et al., 2018)

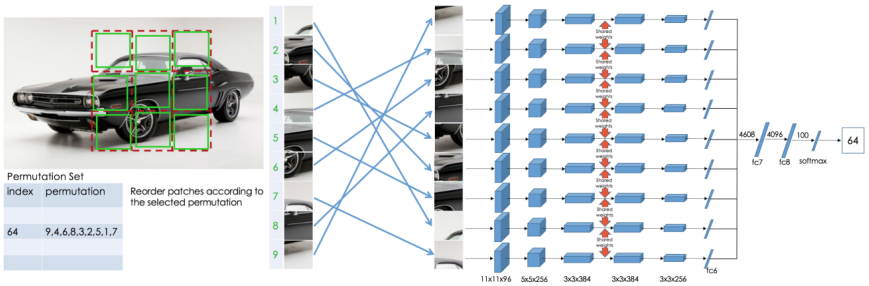


Figure 4.7: Illustration of the jigsaw solving pretext task. A random crop in the image is divided in 9 patches. Each patch are permuted according to a randomly sampled permutation order (i.e. id). Then, all the permuted patches given as input to the network, which need to identify the id of the selected permutation. Illustration from (Noroozi and Favaro, 2016)

Other techniques propose to use image reconstruction as a pretext task. (Zhang et al., 2016) use colorization techniques as task, the model is then trained to color a grayscale input image; more specifically, the aim is to transfer this image to a distribution of quantized color value outputs. Instead of using the standard RGB format, authors use CIE Lab* color space, which is designed to be closer to the human vision. The model is fed with the L channel and need to predict the ab channels. Because of the multimodal nature of the colorization problem, cross-entropy loss of predicted probability distribution across binned color values outperforms L2 loss of raw color values.

The approach used by (Pathak et al., 2016) is called *context encoder*, this technique is related to generative modeling (see fig. 4.8). The model is trained to fill up a missing picture element. The model is trained using both the reconstruction (L2).

During recent years, new methods based on the latent representation, those methods have made huge improvement in performance of the resulting model, specially image based methods.

4.6.2 Representation based approaches

Most of the research presented in this section is based on the principle of contrastive learning. The first works, laying the foundations of this type of learning, go back more than 15 years ago with

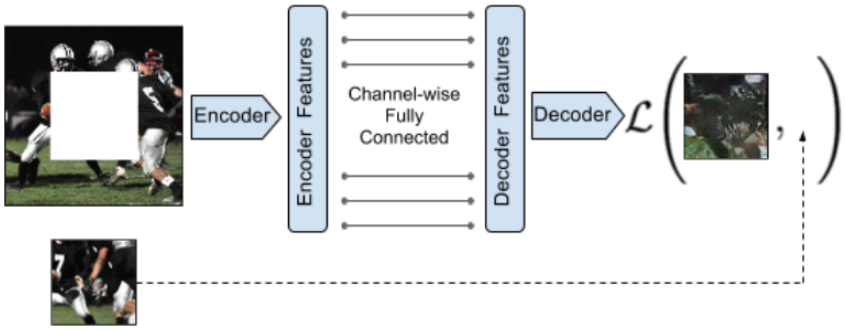


Figure 4.8: Pretext task proposed by (Pathak et al., 2016), here the pretext task is simple, a random part of the image is masked. The network goal is simply to recreate the masked portion as good it can. Illustration from (Pathak et al., 2016)

the works of (Chopra et al., 2005), which propose **contrastive loss** as one of the first deep metric learning training objectives. They define the contrastive loss as follows: Given a list of samples $\{x_i\}$, each with a corresponding label $y_i \in \{1, ..., L\}$ in L classes, the objective is to learn an embedding function $f_\theta(.) : \mathcal{X} \to \mathbb{R}^d$, that encode x_i to a vector in such a way that samples from the same class have similar embeddings while samples from different classes have significantly distinct embeddings. Thus, given a pair of inputs, (x_i, x_j), the contrastive loss reduces the embedding distance when they belong to the same class but maximizes the distance when they don't. The equation 4.2 give a formal version of this loss.

$$\begin{aligned}\mathcal{L}_{\text{contrast}}(\mathbf{x}_i, \mathbf{x}_j, \theta) =& \mathbb{1}[y_i = y_j] \|f_\theta(\mathbf{x}_i) - f_\theta(\mathbf{x}_j)\|_2^2 + \\ & \mathbb{1}[y_i \neq y_j] \max(0, \epsilon - \|f_\theta(\mathbf{x}_i) - f_\theta(\mathbf{x}_j)\|_2)^2\end{aligned} \tag{4.2}$$

Based on this concept, many methods have been proposed. In the following, we will focus on image-based technique. However, similar techniques for language processing have evolved, such as the work proposed by(Devlin et al., 2019b; Su et al., 2020; Shi et al., 2021). Firstly, we will give some general requirements. The majority of approaches for contrastive representation learning, in the computer vision domain, focus on generating a noisy version of a sample (i.e. x_i, x_j) through a series of data augmentation techniques. The augmentation should dramatically alter its aesthetic look while maintaining the semantic content.

There are numerous methods for modifying an image while keeping its semantic value. The simplest way is to apply or multiple basic augmentation such as random cropping, color jittering or Gaussian blur (cf. fig 4.9). Build upon that, many frameworks, such as AutoAugment(Cubuk et al., 2018) or RandAugment(Cubuk et al., 2020), propose to learn good data augmentation strategies. To achieve that, new augmentation methods have been designed to build new data point by mixing existing ones. Among those techniques, we can find MixUp(Zhang et al., 2018a) or MoCHi (Kalantidis et al., 2020).

The image-based contrastive approach can be divided into three major groups, and we will give a major method by group.

Simultaneous Augmentation

A framework for contrastive learning of visual representations was proposed by (Chen et al., 2020a) called SimCLR. It learns visual input representations by maximizing agreement across multiple augmented perspectives of the same sample using a contrastive loss in the latent space.

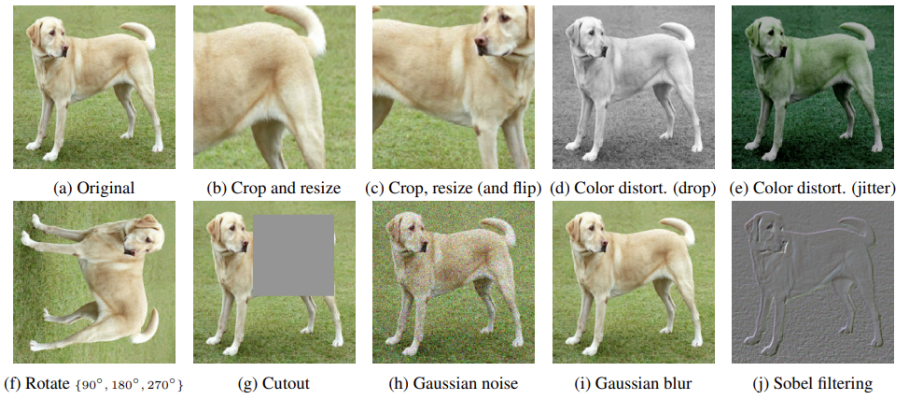


Figure 4.9: Example of the different types of augmentation used in (Chen et al., 2020a). Illustration from (Chen et al., 2020a)

This framework works as the following, and is illustrated in fig. 4.10 :

1. A batch of N samples is chosen at random, and each sample is treated to two different data augmentation algorithms, resulting in a total of $2N$ augmented samples.

$$x_i = a(x) \quad x_j = a'(x) \quad a, a' \sim \mathcal{A}s$$

   where a, a' are two separate data augmentation functions sampled from a set of augmentations $\mathcal{A}$ which include functions such as random crop, resize, blur, ...

2. Given a single positive pair, the remaining 2(N-1) data points are regarded as negative samples. The representation is created by an encoder $f(.)$

$$\mathbf{h}_i = f(x_i), \quad \mathbf{h}_j = f(x_j)$$

3. Here the contrastive loss is defined using the cosine similarity $sim(.,.)$. It should be noted, that the loss operates on an extra layer, called *projection layer*, of the representation $g(.)$ rather than on the representation space directly. This *projection layer* can take various form such as convolution or fully-connected layers. However, only the representation $\mathbf{h}$ is used for downstream operations.

$$\mathbf{z}_i = g(\mathbf{h}_i), \quad \mathbf{z}_j = g(\mathbf{h}_j)$$
$$\mathcal{L}_{\text{SimCLR}}^{(i,j)} = -\log \frac{\exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_k)/\tau)}$$

   $\mathbb{1}_{[k \neq i]}$ is an indicator function which is equal to 1 if $k \neq i$, 0 otherwise.

Memory Bank

He et al., 2020 created MoCo (Momentum Contrast), an unsupervised learning visual representation system, as a dynamic dictionary look-up. The dictionary is organized as a huge FIFO queue of encoded sample representations.

From a sample x_q, called *query sample*, we obtain a query representation through an encoder model $q = f_q(x_q)$. On the other side, we have a momentum encoder $k_i = f_k(x_i^k)$, which encodes a list of key representation noted $\{k_1, k_2, \dots\}$. Assume there is just one positive key $k+$ in the dictionary that matches the query q. $k+$ is build, in this method, by employing a noisy copy of

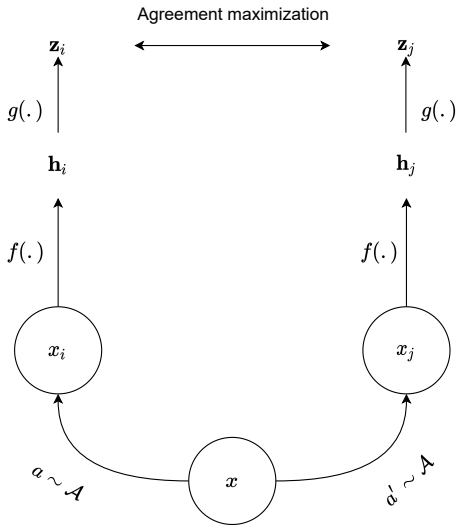


Figure 4.10: Schema of SimCLR workflow. Given an input x, two different augmentations a and a' are sampled from a set of augmentations operations $\mathcal{A}$ and applied on x to create x_i and x_j respectively. Those augmented samples are then passed through the encoder $f(.)$, to obtain the representations h_i and h_j. Then, the representations h_i and h_j are projected via the layer $g(.)$ into $\mathbf{z}_i, \mathbf{z}_j$, respectively. Finally, the agreement maximization loss is performed on the projected representation.

x_q with varied augmentation. Then the contrastive loss proposed by (Oord et al., 2018) is used on one positive sample and $N-1$ negative samples.

$$\mathcal{L}_{\text{MoCo}} = -\log \frac{\exp(q \cdot k^+/\tau)}{\sum_{i=1}^{N} \exp(q \cdot k_i/\tau)} \tag{4.3}$$

However, because the MoCo dictionary cannot be differentiated as a queue, we cannot rely on back-propagation to update the key encoder f_k. The straightforward approach would be to use the same encoder for both f_q and f_k. MoCo, on the other hand, recommended using a momentum-based update with a momentum coefficient $\alpha \in [0, 1)$. Assume that the parameters of f_q and f_k are designated θ_q and θ_k, respectively, the equation (4.4) give the momentum formula.

$$\theta_k \leftarrow \alpha\theta_k + (1-\alpha)\theta_q \tag{4.4}$$

This framework has several advantages over the previous one (SimCLR). The major one is that MoCo decouples batch size from the number of negatives samples, whereas SimCLR requires a high batch size to have enough negative samples and suffers performance decreases when the batch size is reduced.

An evolution called MoCoV2 has been proposed by (Chen et al., 2020c), this evolution take advantages of designs proposed in SimCLR. Precisely they incorporate the MLP projection head and a stronger data augmentation, this two novelty allow this new version to achieve better transfer performance with still no dependency on a very large batch size.

Clustering

(Caron et al., 2020) suggested an algorithm for online contrastive learning. It generates codes ($\mathbf{Q}$) from an augmented version of the image and attempts to estimate it using another augmented version of the same image. This algorithm is called Swapping Assignments between multiple Views (SwAV), and illustrated by fig. 4.12.

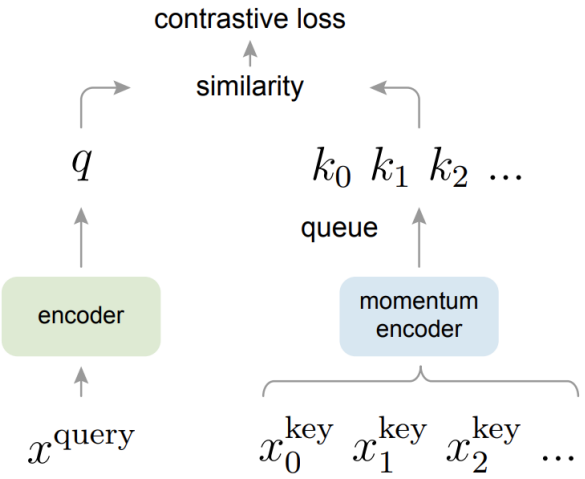


Figure 4.11: A diagram illustrating how Momentum Contrast (MoCo) learns visual representations. Illustration from (He et al., 2020)

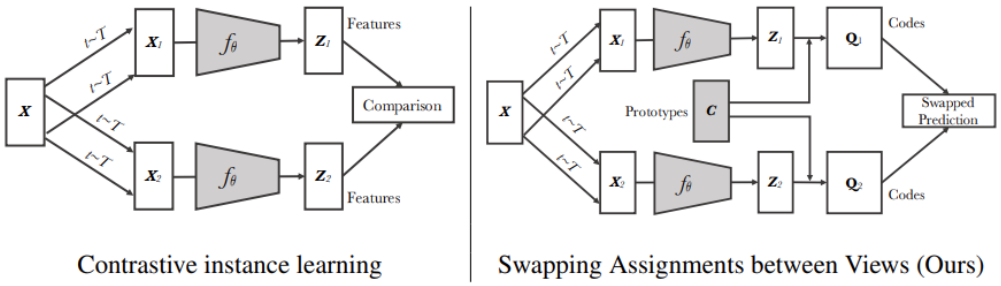


Figure 4.12: Illustration and comparison between the method proposed in SwAV and contrastive instance learning (Wu et al., 2018)(similar to MoCo). Illustration from (Caron et al., 2020)

Given an image with two different augmentation x_t, x_s, we obtain two set of features z_t, z_s by passing those input through an encoder f_θ. Then SwAV generates the corresponding codes $\mathbf{q}_t, \mathbf{q}_s$, and the final loss $\mathbf{L}_{\text{SwAV}}$ is computed by swapping two codes, using $\ell(.)$ function to measure the agreement between a feature and a code.

$$\mathcal{L}_{\text{SwAV}}(z_t, z_s) = \ell(z_t, \mathbf{q}_s) + \ell(z_s, \mathbf{q}_t) \tag{4.5}$$

The predicted swapped agreement is determined by the cross entropy between the generated code and a set of K trainable prototype vectors $C = c_1, ..., c_K$. The prototype vector matrix is shared across batches and represents the anchor clusters to which each instance should be grouped. Then the loss function $\ell(.)$ is defined as eq. (4.6).

$$\begin{aligned} \ell(z_t, \mathbf{q}_s) &= -\sum_k \mathbf{q}_s^{(k)} \log \mathbf{p}_t^{(k)} \\ \mathbf{p}_t^{(k)} &= \frac{\exp(z_t^\top c_k/\tau)}{\sum_{k'} \exp(z_t^\top c_{k'}/\tau)} \end{aligned} \tag{4.6}$$

The term τ is the temperature hyperparameter. Given a batch of B features $\mathbf{B} = [z_1, z_2, \ldots, z_B]$, and the set of prototype vectors C, the matrix mapping between them (code) is defined as $\mathbf{Q} = [q_1, q_2, \ldots, q_B]$. The objective is to maximize the similarity between the features and the proto-

types:

$$\max_{\mathbf{Q}\in\mathcal{Q}} \quad \text{Tr}(\mathbf{Q}^\top \mathbf{C}^\top \mathbf{Z}) + \varepsilon H(\mathbf{Q})$$
$$\text{where } \mathcal{Q} = \left\{\mathbf{Q} \in \mathbb{R}_+^{K\times B} \mid \mathbf{Q}\mathbf{1}_B = \frac{1}{K}\mathbf{1}_K, \mathbf{Q}^\top \mathbf{1}_K = \frac{1}{B}\mathbf{1}_B\right\}$$

where H is the entropy function $H(\mathbf{Q}) = -\sum_{ij} \mathbf{Q}_{ij} \log \mathbf{Q}_{ij}$ and ε is a parameter that controls the smoothness of the mapping. To find the optimal solution for $\mathbf{Q}$ SwAV use the Sinkhorn-Knopp algorithm (Cuturi, 2013).

4.7 Summary

In this chapter, we provided an overview of self-training approaches for semi-supervised learning and an introduction to some self-supervised techniques that have received increasing attention in recent years. First, we discussed the various strategies for selecting unlabeled samples for pseudo-labeling that have been proposed. In particular, we discussed the importance of the distribution of margins over unlabeled data as a key component in the development of these strategies. Second, we overviewed different variants of self-training that have been studied in the literature, as well as some related approaches. Next, we discussed recent theoretical advances achieved in this area of research. Finally, we provided an overview of recent techniques developed around the self-supervision.

Recently, the self-training is not only restricted to semi-supervised learning, but has also been extensively applied for unsupervised domain adaptation (Saito et al., 2017; Zou et al., 2018; Zou et al., 2019), where the goal is to transfer knowledge from the labeled source domain to the target unlabeled one. However, as there exists a distribution shift between the source and the target, the prediction confidence given by the training model may be highly biased towards the source, and thus may be not reliable for pseudo-labeling the target.

Recent works tend to close the gap between self-supervision and semi-supervision like int (Chen et al., 2020b), in which they start by pretraining a model in a self-supervised manner and then fine-tuning it to a downstream task with a limited number of annotated samples.
However, we can notice that all the presented methods are based on a fixed human architecture, thus it would be interesting to examine if this learning context with limited data can be transferred to a NAS application.
We have seen an overview of the concepts and related works essential to the main theme of this thesis through these chapters; The second part of the thesis describes the contributions, starting with the first objective of this thesis.

Part II

Contributions

5

NAS for extreme multi-label classification

Extreme classification and Neural Architecture Search (NAS) are research topics, which have gain a lot of interest recently. In this study, we extend the scope of NAS to the extreme multi-label classification (XMC) tasks. We propose a neuro-evolution approach, that has been found most suitable for a variety of tasks. Our NAS method automatically finds architectures that give competitive results to the state of the art with faster convergence. Furthermore, the weights of the architecture blocks have been analyzed to give insight on the importance of the various operations that have been selected by the method.
This chapter present works from the following contributions:

- Neural Architecture Search for Extreme Multi-label Text Classification [5; 6]

Contents

Supervised classification problems comprise different types. The most basic is undoubtedly supervised monolabel classification learning, sometimes known as binary learning. Its goal is to associate an object, represented by a vector of attribute variables, with a single class (target variable), referred to as label in this context. For example, one might want to create a learning system that can decide whether a stock should be bought or sold based on last tendency (here only two labels are possible "buy" or "sell"). The multiclass classification paradigm, which is currently the most extensively used, was created to cover a broader family of problems. It enables dealing with situations in which the output variable has more than two options. In this scenario, each object is assigned to a class from a list of more than two classes. For example, it enables determining the numbers written on images of handwritten text defined by pixels, and the output variable thus takes a single value among "1", "2",..., "10" (e.g. MNIST (LeCun et al., 2010)).

To solve this kind of tasks, Neural Networks (NNs) have demonstrated outstanding performance in a wide range of tasks. In the field of natural language processing (NLP), a significant challenge in using NNs for this purpose is designing an architecture capable of successfully capturing text semantics. Many methods such as convolutional neural networks (Zhang et al., 2015), recurrent neural networks (Liu et al., 2015) ,and a combination of both (Zhou et al., 2015) have been studied. However, this design process is complex and frequently necessitates the involvement of a human with extensive understanding of the topic and data.

NAS research has paved the way in recent years for the development of neural networks tailored to a specific task or data set to address this design phase. However, as previously stated, most NAS research has concentrated on a small number of tasks (e.g., image classification). Accordingly, we will address in this chapter the first objective of this thesis by focusing on extreme multi-label classification (XMC) task, which is of growing interest in the big data era and has not previously been explored using NAS techniques. In this work, we propose `XMC-NAS`, a NAS-based method for autonomously developing an architecture for the extreme multi-label text classification challenge with minimal prior knowledge. In addition, we establish a search space with natural language processing (NLP) specific operations (e.g., RNN, Convolution, etc.).

We used three large scale XMC datasets with an increasing number of labels to evaluate our method. During the search phase, we employ a proxy dataset to train and assess architectures, similar to common NAS approaches. The discovered architecture achieves competitive results on the proxy dataset in comparison to the state of the art while achieving faster convergence. The highest performing design is then transferred to different datasets and evaluated. In addition, this work includes a study on the significance of operation kinds and network depth in relation to the acquired results.

In the following section, we briefly present some background. In Section 5.2, we present our solution to extreme multi-label classification with neural architecture search. Experimental results are presented in Section 5.3 and the conclusion with an outcome of this work are presented in Section 5.4.

5.1 Background

In addition to the examples given above, for some applications, objects must be specified by many labels. For example, in text annotation, one would want to create a system that describes a document as hilarious, about sports, and in English all at once. In a music collection, one may also wish to annotate a piece of music as being both tragic and classical. This last scenario is known as multi-label classification, and it allows an object to be assigned to one or more labels from a list of predetermined options. However, since the first works, the size of the data has significantly increased due to the advent of big data and larger and larger platforms (e.g., Amazon,

Wikipedia), and most of the pioneer algorithms are no longer able to withstand these new scales. This progression resulted in the extreme multi-label classification problem (XMC). The analyzed data in this context are distinguished by a large number of objects, necessitating the ability to process a large number of attribute variables in input and label variables in output (of the order of 10^4 to 10^7) (Liu et al., 2017b; Nam et al., 2017). As example, the dataset WikiLSHTC (Partalas et al., 2015) include 2.4 millions article, where the goal is to categorize them among the 350 000 possibilities (e.g. fig. 5.1).

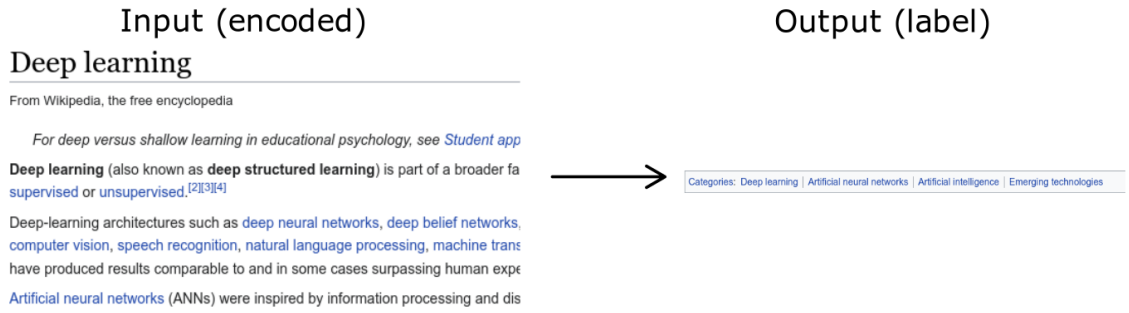


Figure 5.1: Example of an extreme multi-class multi-label classification case. Here, a Wikipedia article need to be classified into a small portion of relevant categories among a large number of possibilities.

For this task, the most common way to evaluate the performance of an approach is to use the two following metrics. The Precision at k denoted by $P@k$, and the normalized discounted cumulative gain at k denoted by $nDCG@k$ (Jain et al., 2016). Both metrics are standard and widely used in the state-of-the-art references.

$$P@k = \frac{1}{k} \sum_{l \in rank_k(\hat{\mathbf{y}})} \mathbf{y}_l \tag{5.1}$$

In the eq. (5.1), $\mathbf{y}$ denotes the binarized true label vector ($\mathbf{y} \in \{0,1\}^L$), $\hat{\mathbf{y}}$ represent the predicted labels vector and $rank_k(\hat{\mathbf{y}})$ returns the k largest indices of $\hat{\mathbf{y}}$ ranked in descending order l in the predicted result.

To address the stakes posed by the extreme multi-label text classification (Babbar and Schölkopf, 2017; Babbar and Schölkopf, 2019; Khandagale et al., 2020), different methods have been proposed that can be classified in the four following family.

The One-Vs-All (OVA) method, such as DiSMEC (Babbar and Schölkopf, 2017), is the most traditional and straightforward strategy for XMC. It simply treats each label independently and learns a classifier (e.g. SVM) for each label. OVA has demonstrated good accuracy, but the computation is too expensive for excessively large label sets. Over time, more complex approaches such as embedding and tree-based techniques have been developed.

The key idea behind embedding-based methods (e.g. AnnexML (Tagami, 2017)) is, since the label size is huge, to compress the labels and use the compressed labels for training, and finally, compressed labels are decompressed for prediction. More formally, given a training instance (x_i, y_i) where x_i is the features vector and y_i the one-hot encoded L-dimensional label vector. This family of methods will compress y_i into a lower L'-dimensional label embedding vector e_i using a compression function $e_i = f_C(y_i)$. Then the method train a regressor model f_R to predict e_i from x_i. Finally, a decompression model, f_D, is used to generate the prediction $\hat{y}$ from e_i.

Tree-based approaches (e.g. PfastreXML (Jain et al., 2016)) are based on the concept of a decision tree. They build a tree by recursively splitting provided instances by features at non-terminal nodes, yielding a basic classifier with only a few active labels at each leaf. Following

the random forest concept, most tree-based algorithms produce an ensemble of trees by randomly selecting (sampling) a feature subset at each node of the trees.

The most recent of those methods are deep learning based such as XML-CNN (Liu et al., 2017b), a structure of convolutional neural network (CNN) and pooling in order to get a precise text representation. However, it is hard for CNN to capture the most relevant part of a text and the long term dependency. Other methods, more similar, to Seq2seq methods have been applied as discussed in MLC2Seq (Nam et al., 2017), SGM (Yang et al., 2018) and AttentionXML (You et al., 2018). Those methods used recurrent neural network (RNN) to classify the text. Moreover, a significant interest has been given on attention mechanism in the last few years (Lin et al., 2017). Attention mechanism has demonstrated great performance in sequence modeling, in particular in NLP domain, and has therefore been also applied in the context of XMC (Yang et al., 2018; You et al., 2018).

5.2 `XMC-NAS`: NAS for XMC

In this work, we propose `XMC-NAS`, a tool to automatically design architecture for the extreme multi-label classification task. Our approach is based on three main components: i) the text embedding, ii) the search of the architecture, and iii) output classification. These three components form a pipeline in which components i) and iii) are fixed and excluded from the search task. Thus, the architecture search task is performed only on the component ii). The first component of our methods consists in transforming the text into word embedding, i.e. numerical vectors. This embedding step should allow the model to use these representations to produce a more accurate prediction. The second step is the search phase for the most performing architecture, using an *evolutionary algorithm* (c.f. Fig.5.2).

Finally, the last component classifies the output, in several categories. This last component is based on attention mechanism and fully connected layers.

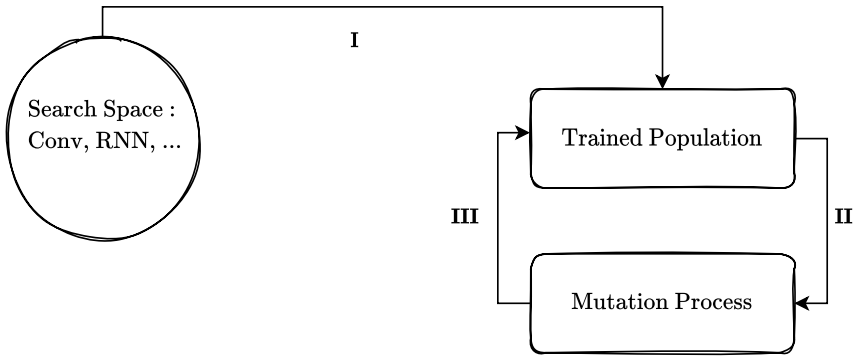


*Figure 5.2: **I**. Architectures are constructed from randomly sampled operations and then trained and evaluated, **II**. Randomly sample 10 architectures, and rank them by Precision@5 obtained on test set. The most performing one is selected for mutation, **III**. The newly mutated architecture, is trained and evaluated. Then placed in the trained population. The oldest architecture is removed from the population.*

In the following section, we describe our approach in detail. First, we present the search space, the search algorithm used and their specificities; and finally, we describe the different parts of the discovered network.

5.2.1 Architecture search phase

The search phase can be broken down as follows. The architecture is searched in a *search space* that defines the possible structure of the final architecture. In this search space, we have *candidate operations* that can be used in the architecture. Finally, to research the architecture, we use a *search algorithm* that searches for best architectures in the search space.

Search Space.

In this work, we used a macro search space. We chose this option, because it is generally this kind of architecture that is used in the natural language processing. Furthermore, this type of search space allowed us to dynamically add operations, either in multipath or in a chained manner. This flexibility allowed us to investigate the impact that the depth of the network could have on the final result. More precisely, we build our graph as follows: first, the nodes are sequentially sampled (i.e. an operation is selected) to create a graph of N nodes. Then the input of a node j is selected in the set of previous nodes (i.e. nodes from 1 to $j-1$). At start, these set is initialized with the embedding layer. Finally, when the node j has an operation and an input node, it is added to the set of the previous nodes.

Candidate Operations.

To build our set of *candidate operations*, we have selected the most common operations in NLP field, which consist of a mixture of convolutional, pooling and recurrent layers. We have defined four variants of 1D convolutional layers, with a kernel of different size: 1, 3, 5 and 7 respectively. All convolution layers use a stride of 1 and use padding if necessary to keep a consistent shape. We used the two types of pooling layers that calculate either the average or the maximum on the filter size, this size is set to 3 for both. Similarly to the convolution layers, the pooling layers use a stride of 1 and use padding if necessary. Finally, we used the two popular types of recurrent layers, namely the Gated Recurrent Unit (GRU) (Cho et al., 2014a) and the Long-Short Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997), which are able to capture long-term dependencies. Specifically, we use bidirectional LSTM and GRU.

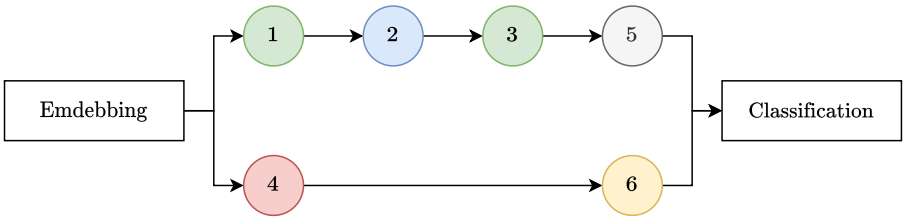


Figure 5.3: Illustration of a simple architecture, with 6 layers. The numbers represent the sampling order of the layers. The limit of maximum number of previous layers that can be used as input is set to 5 (e.g. the layer 6 could hence take as input only nodes from 1 to 5). Here, different operations are illustrated with different colors.

Search Algorithm.

As NAS search algorithm we use a neuro-evolution process (*regularized evolution*) as described in (Real et al., 2019). We chose this approach because it allows us to have a fine vision of the impact of each operation on the final result. Regarding the mutation aspect, we use the same configuration as described in (Real et al., 2019):

- Randomly select an input from a node on the network and modify it with a new input.

- Randomly select an operation from the network and change it with a new sample.

Figure 5.2 illustrates the search algorithm of the *regularized evolution*. In order to see the impact of the number of layers on the final results, a third mutation, corresponding to the addition of a new layer, has been introduced. The choice among these mutations is random. We also seek to better understand how operations perform together, i.e. to evaluate the importance of the various operations with respect to the final results. To do this, we use a linear combination of outputs from each layer, where weights are learned during the training process.

5.2.2 Components

Our network has certain parts fixed, namely the embedding, attention and classification modules. This section will introduce them. The full pipeline of `XMC-NAS` is illustrated in fig. 5.4.

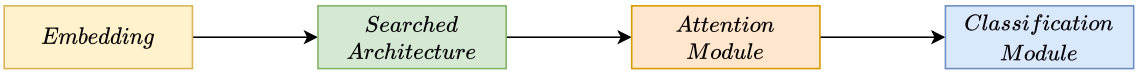


Figure 5.4: Illustration of a `XMC-NAS` pipeline.

Text Embedding.

The embedding layer produces a fixed length representation. This layer is an embedding map, which means that each word is mapped to a vector; these vectors are designed in such a way that words with similar meanings should have close representations, resulting in a cluster if we project this vector in 2D space (e.g. Figure 5.5).

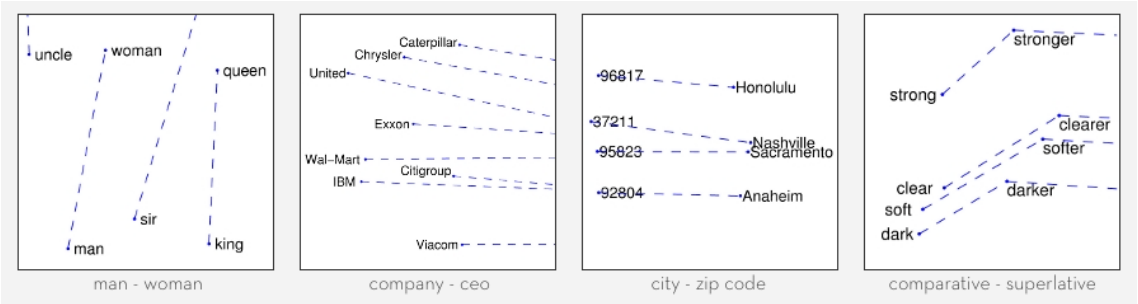


Figure 5.5: Illustration of the projected GloVe embedding. Words with close meaning should have close embedding.

To initialize this embedding map, we used the GloVe (Pennington et al., 2014) embedding 840B-300d[1] version, which allows us to skip the step of learning a new embedding from scratch.

Attention Module.

We use a self-attention mechanism based on the one demonstrated in (Lin et al., 2017), similarly to (You et al., 2018). The attention process helps to grasp the important parts of the text. This mechanism uses a vector $\mathbf{c_t}$ that represents the relevant *context* for the label t, where t is in $1, \ldots, T$. For an input sequence of size N, the context vector is given in equation 5.2.

[1] `http://nlp.stanford.edu/data/wordvecs/glove.840B.300d.zip`

$$\mathbf{c_t} = \sum_{i=1}^{N} \alpha_{t,i} \mathbf{h_i}, \tag{5.2}$$

$$\alpha_{t,i} = \frac{e^{\mathbf{W_t}^T \cdot \mathbf{h_i}}}{\sum_{k=1}^{N} e^{\mathbf{W_t}^T \cdot \mathbf{h_k}}} \tag{5.3}$$

Where, $\mathbf{h_i}$ denotes the hidden representation, i.e. the output of RNN encoder states or of the convolution. In the case of BiRNN, layer $\mathbf{h_i}$ is the concatenation of vectors from the forward $\overrightarrow{h_i}$ and backward $\overleftarrow{h_i}$ passes. The term $\alpha_{t,i}$ is called attention factor (Eq. (5.3)). The set of attention factors $\{\alpha_{t,i}\}$ represents how much of each input should be considered for each output. In Eq. (5.3), $\mathbf{W_t}$ is the attention weight (i.e. a learnable weight matrix) for the t-th label.

Classification Module.

The final part of the network is composed of 2 or 3 fully connected layers, which reduces the output of the attention module. The output is then fed into a classification layer that predict the labels associated to the inputs. This module is illustrated by figure 5.6.

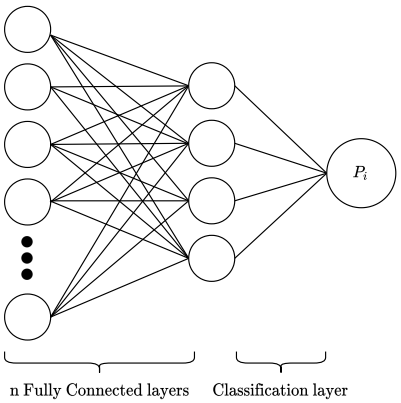


Figure 5.6: Illustration of the classification module, the n number of fully connected layer is specified in table 5.2. The P_i is the probability prediction for the i-th label.

5.2.3 Analysis of Operations Importance

This section presents an analysis of the weights of the linear combination, particularly the impact of each operation on the final results, and whether different operations combine efficiently. We address this analysis in two steps. In the first step, we focus on how operations combine with each other. In a second step, we analyze the results and the impact of operations as the networks deepen.

First Step: In this step, the base population is randomly initialized, meaning that the input and operation of each node is chosen at random. We try to determine which operation is the most important in the first layers by scaling their outputs with trainable weights of the linear combination. The Fig.5.7 shows three examples of the first layers for different combinations of operations, as well as the corresponding learned weights assigned to each operation. The block "Rest of the network" represents the attention and classification modules. The blocks in the hatched areas of the Fig.5.7 were not part of the mutation process and were fixed. For each architecture example, the displayed weights are the averages obtained over several runs of the NAS. The different gray scales indicate different experiments. We observe in Fig.5.7 that pairs of operations of the same type (i.e. BiLSTM), tend to have almost equal weights. However, some trends could be observed in the case of the combination of two convolutions; those with a larger kernel size have higher weights. This effect is particularly pronounced in the case of the kernel size of 1, reflecting the

need for sequence modeling blocks at this level. In the case of mixed operations, it turned out that BiLSTM operations systematically have higher weights. An example of a run with mixed operation is presented in the right-hand side of Fig.5.7c. More generally, our results show that architectures, which contain BiLSTM at the first layer, perform better. This first step shows that the result is based mainly on the long-term dependencies captured by BiLSTM rather than on the combinations of local features generated by the convolution.

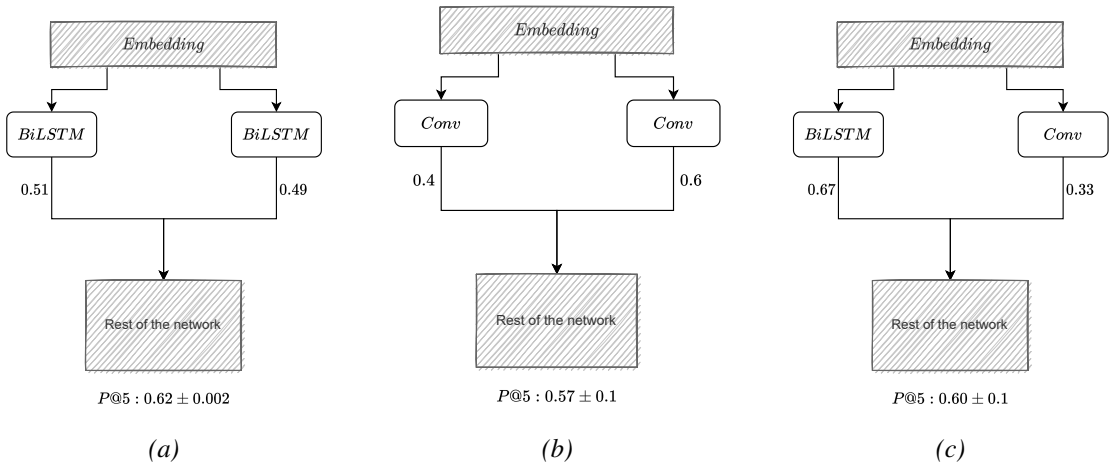


Figure 5.7: Visualization of the network architecture with the applied weight on each operation. The weights have been averaged over multiple runs, the range of $P@5$ are obtained on the proxy dataset. For the central case, we also averaged over the kernel size.

Second Step: This second stage of analysis aims to quantify the impact of the number of layers on the final results, as well as the weights assigned to each operation. According to the results obtained in the first step, which show that the network with BiLSTM layers works better, in this second step, a part of the population has the constraint to start with at least one BiLSTM layer, which takes as input the embedding. For the analysis of the impact of the number of operations, we calculated the average $P@5$ based on the number of operations. The number of operations ranged from 2 to 6. We observed that the average precision is almost constant, regardless of the number of operations in the network, with a range of results close to what we have previously obtained. This result is corroborated by the analysis of the combination of operation weights. The Fig.5.8 shows examples of architecture for different combinations of operations with associated weights, RoN stand for the rest of the network. This time the operations are assigned sequentially (i.e. one after the other) forming a deeper network. The blocks in hatched areas in Fig.5.8 are partially or totally part of a constraint, as in the previous subsection. Here, the blocks "Rest of the network" represent the following layers in the network, not shown for the sake of clarity, and still followed by the attention and classification modules. As previously, the weights displayed for each type of architecture are the averages obtained from multiple runs of the NAS. We note on the Fig.5.8 that the weights on additional layers are small compared to those that bypass it. This trend has been observed in all experiments and suggests that, given our operations pool, additional layers do not provide much more information. Thus, the information important for the result is extracted by layers that take the embedding as input.

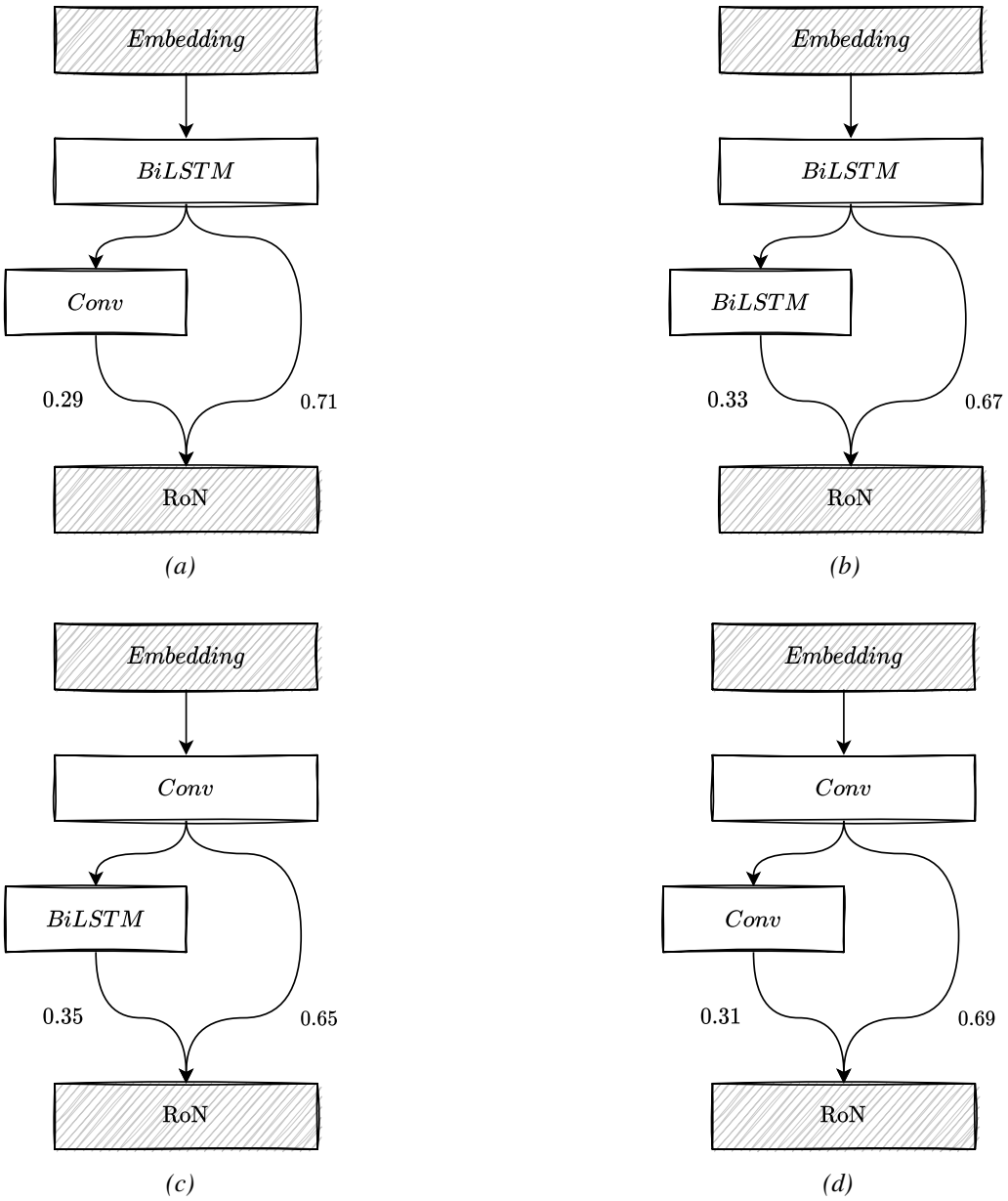


Figure 5.8: Visualization of the network architecture, when the network is deeper. The weighted line represents the linear combination of all the layers outputs, with the weight applied on each output. The weighted line is the linear combination of all layer outputs, with a weight applied to each output. The weights show that the output of the first layer is more relevant for the final result.

5.3 Experiments

We have conducted a number of experiments to evaluate how the proposed XMC−NAS method can help design an efficient neural network model for multi-label text classification.

5.3.1 Experimental Setup

Datasets and Evaluation Metrics

We conducted our study on three of the most popular XMC benchmark datasets downloaded from the XMC repository[2]. These datasets are considered large scale, with the number of class labels ranging from 4,000 to 30,000, which are listed from smallest to largest (in terms of number of labels) by EURLex-4K (Mencía and Fürnkranz, 2008), AmazonCat-13K (McAuley and Leskovec, 2013), and Wiki10-31K (Zubiaga, 2012) summarized in Table 5.1. We followed the same pre-processing pipeline as the one used in (You et al., 2018). To create the validation set, we perform a split with the same initialization seed for all experiments.

Table 5.1: Statistics of XMC datasets used in our experiments. L: # of classes.

Dataset	# of Training examples	# of Test examples	L	Avg. of class labels per example	Avg. size of classes
EURLex-4K	15,539	3,809	3,993	25.73	5.31
Wiki10-31k	14,146	6,616	30,938	8.52	18.64
AmazonCat-13K	1,186,239	306,782	13,330	448.57	5.04

Architecture Search Evaluation

This section presents the data and the hyperparameters that we have used during the search phase of our method. Finally, we present the most performing architecture that has been discovered on the proxy dataset.

Parameters and Data. We performed the search phase on the relatively small EURLex-4K dataset for scalability considerations, we call it the proxy dataset. In each experiment, we create a base population of 20 networks. We then apply 50 rounds of mutations. For our experiments we have used the same hyperparameters as in (You et al., 2018), for the training of each sampled network. Namely, the optimizer was Adam (Kingma and Ba, 2014) with a learning rate set to 0.001, and the maximum number of epochs were set to 30 epochs with early stopping. To be consistent with, (You et al., 2018) we have used the same number of hidden states for the LSTM, which are specified in Table 5.2. The training stops if the performance of the network does not increase during 50 consecutive steps. We have used the cross-entropy loss function as proposed in (Liu et al., 2017b) for training the models.

The Discovered Architecture. The architecture, found by XMC−NAS, consists of two BiLSTMs that take the same input and holds their own representation. The outputs of the two BiLSTMs is then concatenated along the hidden dimension, and given as the input of the self-attention block. Finally, we use a chain of fully connected networks to classify the sequence. For the training detail, we use the same as presented in the previous paragraph (see also Table 5.2).
The architecture of the network discovered by XMC−NAS is presented in Fig.5.9.

[2] http://manikvarma.org/downloads/XC/XMLRepository.html

Table 5.2: Hyperparameters used for the training of the discovered model.

Dataset	Valid size	BiLSTM Hidden size	Fully connected
EURLex-4K	200	256	[512,256]
Wiki10-31k	200	256	[512,256]
AmazonCat-13K	4000	512	[1024,512,256]

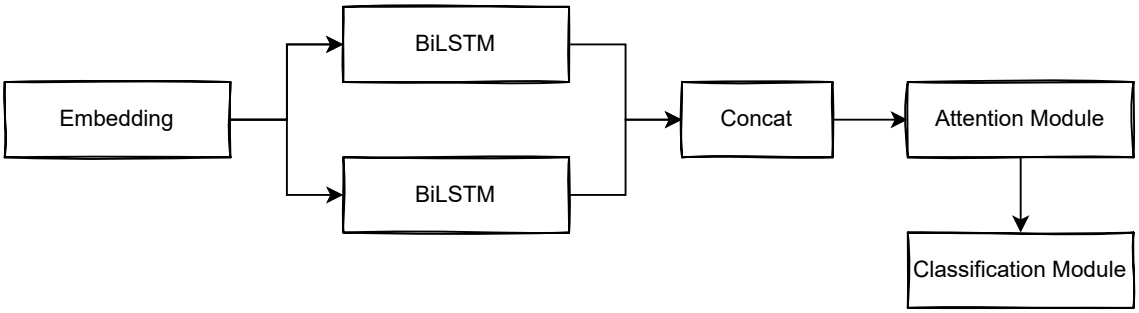


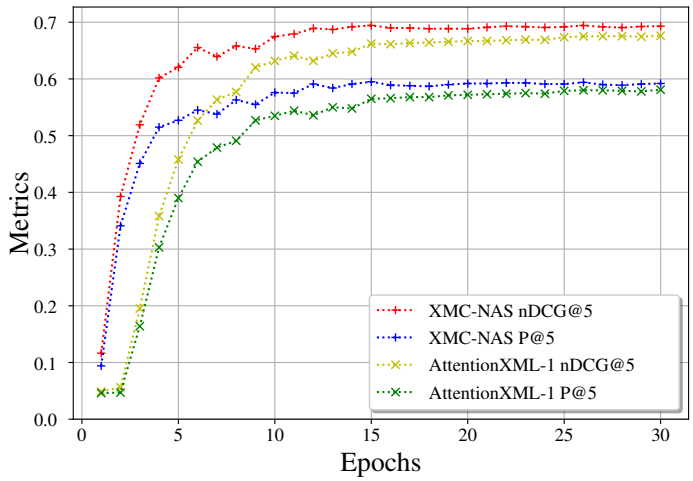
Figure 5.9: Illustration of a XMC-NAS pipeline.

5.3.2 Experimental Results

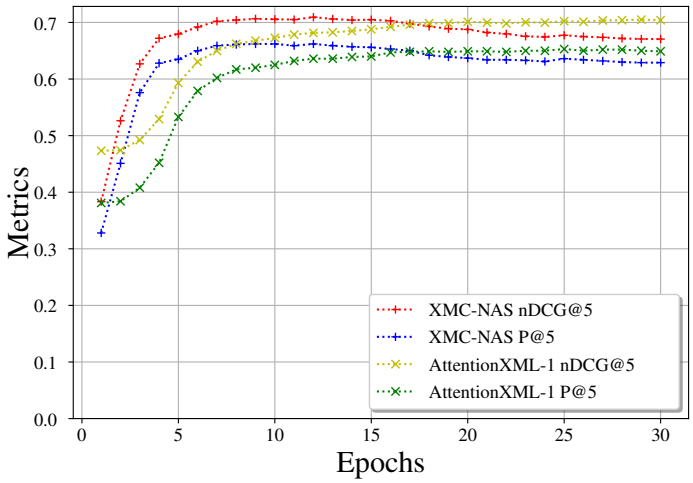
In this section, we will present the results obtained by the XMC-NAS discovered architecture (Figure 5.9) on various XMC datasets (Table 5.1). First, we present the results obtained on the proxy dataset (EURLex-4K) used for the search phase. Finally, we evaluate the performance of this discovered architecture, transferred to the other datasets. To train our network, we use 2 Nvidia GV100, with data parallelism training. The search time on the proxy dataset, depending on the configuration, ranges from 6 hours to 5 days. We compare the results of our method to the most representative methods on XMC (with the results provided by the authors in corresponding papers). Some of these techniques are deep learning based like MLC2Seq (Nam et al., 2017), XML-CNN (Liu et al., 2017b), Attention-XML (You et al., 2018). The others techniques are, AnnexML (Tagami, 2017), DiSMEC (Babbar and Schölkopf, 2017), PfastreXML (Jain et al., 2016) and Parabel (Prabhu et al., 2018).

Table 5.3: Comparison performance table on three datasets. Our methods surpass the state of the art in 4 cases and get competitive results that are really close to the state of the art otherwise.

	EURLex-4K			Wiki10-31K			AmazonCat-13K		
Methods	P@1	P@3	P@5	P@1	P@3	P@5	P@1	P@3	P@5
PfastreXML (2016)	0.731	0.601	0.505	0.835	0.686	0.591	0.917	0.779	0.636
AnnexML (2017)	0.796	0.649	0.535	0.864	0.742	0.642	0.935	0.783	0.633
DiSMEC (2017)	0.832	0.703	0.587	0.841	0.747	0.659	0.938	0.791	0.640
MLC2Seq (2017)	0.627	0.591	0.513	0.807	0.585	0.546	0.942	0.694	0.575
XML-CNN (2017)	0.753	0.601	0.492	0.814	0.662	0.561	0.932	0.770	0.614
Parabel (2018)	0.821	0.689	0.579	0.841	0.724	0.633	0.930	0.791	0.645
AttentionXML-1 (2018)	0.854	0.730	0.611	**0.870**	**0.777**	0.678	**0.956**	**0.819**	**0.669**
XMC-NAS	**0.858**	**0.738**	**0.620**	0.849	0.772	**0.681**	0.951	0.813	0.664



(a) EURLex-4K



(b) Wiki10-31K

Figure 5.10: Plot of the nDCG@5 and P@5 on the validation set, on two different datasets. We notice, the discovered architecture have a faster convergence compared to the current state of the art. In the 5.10a our method get better final results, in 5.10b our final results (around epoch 15) are close.

EURLex-4K Results.

As presented in the left-hand side of the Table 5.3 we have obtained an improvement on P@1, P@3 and P@5 with respect to the state of the art. The shown precision are averaged over 3 different initializations. A significant improvement is obtained on the precision at 3 and 5, where we obtain 0.738 and 0.620, respectively, compared to 0.730 and 0.611 before. The Fig.5.10a presents the evolution of P@5 and the nDCG@5 over the validation set with respect to the number of epochs. We can observe that our network has a faster convergence. The results are obtained around 15 epochs and after this point, the improvement is relatively small, which indicates that the network might overfit. It is not impossible that the improvement obtained is due to a larger network. However, we have systematically observed faster convergence in all the cases we have experienced. Furthermore, the contribution of the embedding or attention module on the results is

not yet clear, as we have not yet studied the impact of these modules.

Architecture Transfer Results.

We train and evaluate the discovered architecture following the same training procedure as defined in section 5.3.1 and using the hyperparameters presented in Table 5.2 on the two other datasets. The middle and right side of Table 5.3 show the comparison of the architecture discovered by `XMC-NAS` on EURLex-4K with others methods. We notice that the best discovered architecture transferred to larger datasets obtains results close to the current state of the art. In some cases we slightly exceed the results, as in the case of $P@5$ on the Wiki10-31K. Moreover, we notice in Fig.5.10b that our methods still have a faster convergence, the same trend as observed on proxy dataset (cf. Fig.5.10a). Moreover, our results on Wiki10-31K and AmazonCat-13K are obtained in half of the epochs required by AttentionXML-1.

5.4 Summary

We have presented in this work a way to extend the field where NAS have been by applied, by proposing an automated method to discover architecture for the specific task of extreme multi-label classification, based on the regularized evolution (Real et al., 2019) and with a domain oriented pool of operations. This method has found architectures that have provided competitive results with the existing state-of-the-art methods (You et al., 2018), and in some cases overpassed them. Moreover, our method showed faster convergence rates on all datasets, which are more likely due to a higher complexity of the model. In addition, trainable weights were introduced on each operation of the pool in order to provide more understanding on the impact of each architecture blocks.

Motivated by these initial encouraging results, we intended to replicate them on another task. The rebuilding of the RSSI map, on the other hand, is a more difficult operation. Indeed, in its optimal form, this endeavor necessitates the collection of a large number of values at various points on a map for reconstruction purposes. Unfortunately, in practice, collecting all of these locations is very expensive; thus, the goal is to recreate the map from a specific spot. In the following contribution, we will see how the NAS technique is being adapted for this new task in a situation with limited amount of annotated data.

6

NAS for RSSI map reconstruction

In this work, we present a Neural Network (NN) model based on Neural Architecture Search (NAS) and self-learning for received signal strength (RSS) map reconstruction out of sparse single-snapshot input measurements, in the case where data-augmentation by side deterministic simulations cannot be performed. The approach first finds an optimal NN architecture and simultaneously train the deduced model over some ground-truth measurements of a given (RSS) map. These ground-truth measurements along with the predictions of the model over a set of randomly chosen points are then used to train a second NN model having the same architecture. Experimental results show that signal predictions of this second model outperforms non-learning interpolation state-of-the-art techniques and NN models with no architecture search on five maps of RSS measurements. This chapter present works from the following contributions:

- Self-learning for Received Signal Strength Map Reconstruction with Neural Architecture Search [2; 3]

Contents

The integration of low-cost sensor and radio chips in a plurality of connected objects on the Internet of Things (IoT) has been contributing to the fast development of large-scale physical monitoring and crowd sensing systems in various kinds of smart environments (e.g., smart cities, smart homes, smart transportations, etc.). In this context, the ability to associate accurate location information with the sensor data collected in the field opens appealing perspectives in terms of both location-enabled applications and services (Khelifi et al., 2019).

Typical fingerprinting methods applied to wireless localization (Vo and De, 2016) ideally require the prior knowledge of a complete map of such radio metrics, covering the area of interest. However, in real life systems, it is impractical to collect measurements from every single location of the map and one must usually rely uniquely on sparse and non-uniform field data. Since only partial data is available, this approach falls within the category of semi-supervised learning.

To tackle this problem, classical map interpolation techniques, such as radial basis functions (RBF) or kriging (Choi et al., 2018), have been used in this context. These approaches are simple and fast, but they are quite weak in predicting the complex and heterogeneous spatial patterns usually observed in real life radio signals (e.g., sudden and/or highly localized transient variations in the received signal due to specific environmental effects). Data augmentation techniques have thus been proposed for artificially increasing the number of measurements in such radio map reconstruction problems.

However, the application of neural networks to this task is still limited, which motivates the study of NAS methods in this particular domain.

We consider Received Signal Strength Indicator (RSSI) map reconstruction in a constrained low-cost and low-complexity IoT context, where one can rely only on few ground-truth (i.e., GPS-tagged) single-snapshot field measurements and for which data-augmentation techniques based on side deterministic simulations cannot be applied, due to their prohibitive computational cost and/or to *a prior* unknown environment physical characteristics. This problem of map interpolation is similar to the task of image restoration, for which, NN based models with fixed architectures have been already proposed (Ulyanov et al., 2017). In the case where there are few observed pixels in an image, these approaches fail to capture its underlying structure that is often complex. In this context, we will therefore address the two main objectives of this thesis. We propose a first NN model find using Neural Architecture Search (NAS) for the design of the most appropriate model given a RSSI map with a small number of ground truth measurements. For this purpose, we develop two strategies based on genetic algorithms and dynamic routing for the search phase. We show that with the latter approach, it is possible to learn the model parameters while simultaneously searching the architecture. Ultimately, after selecting the best architecture and in order to enhance the model predictions, the proposed approach uses also some extra data of the map with the predictions of the optimized NN in non-visited positions together with the initial set of ground-truth measurements for learning a final model.

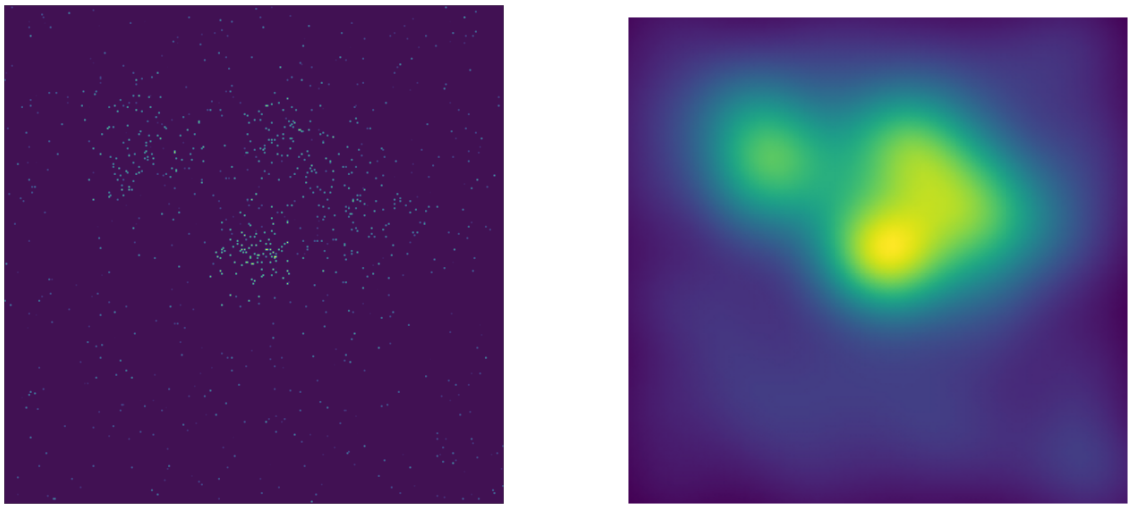
6.1 Background

As presented earlier, it is important for a whole range of applications to associate an accurate location associated with the sensor data. To acquire this localization information, it is possible to use technologies such as Global Positioning Systems (GPS) , which have been widely used in outdoor environments for the last past decades. However, these systems still suffer from high power consumption, which is hardly compliant with the targeted IoT applications.

In order to preserve both nodes with low complexity and fairly good localization performances, an alternative is to interpret radio measurements, such as the Received Signal Strength Indicator (RSSI) (i.e., received power of sensor data packets sent by IoT nodes and collected at their

serving base station(s)), as location-dependent fingerprints for indicating the positions of mobile devices (Burghal et al., 2020; Cheng et al., 2012; Dargie and Poellabauer, 2010; Tahat et al., 2016; Yu et al., 2009).

However, as previously noted, fingerprinting approaches ideally require prior knowledge of a comprehensive map of such measures, which is impossible to achieve in a real-world system. To circumvent the limitation of a partially available map, a more complete map is constructed from the available points (cf. Figure 6.1). Several approaches have been proposed for this, and we will provide an overview of them in the following section.



(a) *(b)*

Figure 6.1: Illustration of `RSSI` *map reconstruction, with a random map. The left side is the input, where the datapoint are collected. On the right side is the interpolated reconstructed map. The intensity of each point represents the received signal intensity in decibels (dB). Axis denote the x and y positions*

6.1.1 Interpolation techniques

Various spatial interpolation methods have been proposed for radio map reconstruction in the wireless context.

One first approach, known as kriging or Gaussian process regression (Li and D.Heap, 2011), exploits the distance information between measured points, while trying to capture their spatial dependencies. Another popular method is based on radial basis functions (`RBF`) (Choi et al., 2018; Enrico and Redondi, 2018; Redondi, 2018). This technique is somehow more flexible, makes fewer assumptions regarding the input data (i.e., considering only the dependency on the distance) and is shown to be more tolerant to some uncertainty (Rusu and Rusu, 2006). In (Choi et al., 2018) for instance, the authors have divided all the points of a database of outdoor `RSSI` measurements into training and testing subsets, and compared different kernel functions for the interpolation. The two methods above, which rely on underlying statistical properties of the input data (i.e., spatial correlations) and kernel techniques, require a significant amount of input data to provide accurate interpolation results. Accordingly, they are particularly sensitive to sparse initial datasets. They have thus been considered in combination with crowdsensing. In (Liao et al., 2019) for instance, so as to improve the performance of basic kriging, one calls for visiting new positions/cells where the interpolated value is still presumably inaccurate. In our case though, we can just rely on a `RSSI` map with few ground truth initial measurements.

Another approach considered in the context of indoor wireless localization relies on both col-

lected field data and an a priori path loss model that accounts for the effect of walls attenuation between the transmitter and the receiver (Kubota et al., 2013). In outdoor environments, local path loss models (and hence, particularized RSSI distributions) have been used to catch small-scale effects in clusters of measured neighboring points, instead of using raw RSSI data (Ning and al., 2016). However, those parametric path loss models are usually quite inaccurate and require impractical in-site (self-)calibration.

Data-augmentation approaches. One more way to build or complete radio databases relies on deterministic simulation means, such as Ray-Tracing tools (e.g., (Laaraiedh et al., 2012; Raspopoulos et al., 2012; Sorour et al., 2015)). The latter aim at predicting in-site radio propagation (i.e., simulating electromagnetic interactions of transmitted radio waves within an environment). Once calibrated with a few real field measurements, such simulation data can relax initial metrology and deployment efforts (i.e., the number of required field measurements). Nevertheless, these tools require a very detailed description of the physical environment (e.g., shape, constituting materials and dielectric properties of obstacles, walls...). Moreover, they usually require high computational complexity. Finally, simulations must be re-run again, likely from scratch, each time minor changes are introduced in the environment.

6.1.2 NN based models trained after data-augmentation

Machine and deep learning approaches have been recently applied for RSSI Map reconstruction. These methods have shown to be able to retrieve unseen spatial patterns with highly localized topological effects and hidden correlations. Until now, these methods have been trained over simulated datasets given by data-augmentation approaches.

In (Levie et al., 2020), given an urban environment, the authors introduce a deep neural network called RadioUNet, which outputs radio path loss estimates trained on a large set of generated using the Dominant Path Model data and UNet architecture (Ronneberger et al., 2015). In another contribution, the authors have shown that using the feedforward neural network for path loss modelling could improve the kriging performance (Sato et al., 2019), as conventional parametric path loss models admit a small number of parameters and do not necessarily account for shadowing besides average power attenuation.

Besides wireless applications, similar problems of map restoration also exist in other domains. In (Zhu et al., 2020) for instance, the authors try to build topographic maps of mountain areas out of sparse measurements of the altitudes. For this purpose, they use a Generative Adversarial Network (GAN) architecture, where in the discriminator they compare pairs of the input data and the so-called "received" map, either generated by the generator or based on the full true map. Another close problem making extensive use of neural networks is the image inpainting problem, where one needs to restore missing pixels in a single partial image. By analogy, this kind of framework could be applied in our context too, by considering the radio map as an image, where each pixel corresponds to the RSSI level for a given node location. Usually, such image inpainting problems can be solved by minimizing a loss between true and predicted pixels, where the former are artificially and uniformly removed from the initial image. This is however impossible in our case, as only a few ground-truth field measurements can be used.

In contrast to the previous approaches, we consider practical situations where data-augmentation techniques cannot be used mainly because of unknown environment characteristics and computational limitations, and, where there is only a small amount of ground-truth measurements. Our approach automatically searches an optimized Neural Network model for the RSSI map reconstruction in hand, and, it is based on self-training for learning an enhanced NN model with the initial ground-truth and pseudo-labeled measurements obtained from the predictions of the first NN model on a set of randomly chosen points in the map.

Alg. $\texttt{SL}_{\texttt{NAS}}$

Input : A training set: (X_ℓ, Y_ℓ); a validation set: (X_v, Y_v) and a set of $2D$ locations without measurements: X_u.
Init : Using $(X_\ell, Y_\ell) \cup (X_v, Y_v)$, find interpolated measurements $\tilde{Y}_u$ over X_u using the RBF interpolation method;

Step 1 : Search the optimal NN architecture using $(X_\ell, Y_\ell) \cup (X_u, \tilde{Y}_u)$;
Step 2 : Find the parameters $\theta_1^\star$ of the NN model f_θ :
$\theta_1^\star = \arg\min_\theta \mathcal{L}(X_\ell, Y_\ell, \theta)$ ▷ Eq. 6.1;
Step 3 : Choose $X_u^{(k)}$ randomly from X_u and find the new parameters $\theta_2^\star$ of the model f_θ :

$\theta_2^\star = \arg\min_\theta \mathcal{L}(X_\ell \cup X_u^{(k)}, Y_\ell \cup f_{\theta_1^\star}(X_u^{(k)}), \theta)$;

Output : $f_{\theta_2^\star}$,$\tilde{Y}_u$

Algorithm 1: Algorithm used to jointly search architecture of f and pseudo annotated data $\tilde{Y}_u$

6.2 Methods

In this section, we first introduce our notations and setting, and then present our main approach, denoted as $\texttt{SL}_{\texttt{NAS}}$ in the following.

6.2.1 Notations and Setting

For a given base station X, let $Y \in \mathbb{R}^{H \times W}$ be the whole matrix of ground-truth signal measurements , where $H \times W$ is the size of the (discretized) area of interest. We suppose to have access to only some ground truth measurements Y_m in Y, that is $Y_m = Y \odot M$, where $M \in \{0,1\}^{H \times W}$ is a binary mask indicating whether each pixel includes one available measurement or not, and $\odot$ is the Hadamard's product. Here, we suppose that the number of non-null elements in Y_m is much lower than $H \times W$. We further decompose the measurements set Y_m into three parts Y_ℓ (for *training*), Y_v (for *validation*) and Y_t (for *test*), such that $Y_\ell \oplus Y_v \oplus Y_t = Y_m$, where $\oplus$ is the matrix addition operation. Let X_ℓ, X_v, X_t, X_m be the associated 2D node locations (or equivalently, the cell/pixel coordinates) with respect to base station X and X_u be the set of 2D locations for which no measurements are available.

Our approach is based on three main phases i) *architecture search phase* - the search of an optimal architecture of a Neural Network model; ii) *data-augmentation phase* - the assignment of pseudo-labels to randomly chosen unlabeled data using the predictions of the found NN model trained over Y_ℓ; and iii) *self-learning phase* - the training of a second NN model with the same architecture over the set of initial ground truth measurements and the pseudo-labeled examples. In the next sections, we present these phases in more detail. These phases are resumed in Algorithm 1.

6.2.2 Architecture Search phase

Here, we consider a first reference RSSI map as an input image, where unknown measurements in X_u are obtained with a RBF using points in the train and validation sets; $(X_\ell, Y_\ell) \cup (X_v, Y_v)$. The latter was found the most effective among other state-of-the-art interpolation techniques (Choi et al., 2018). Let $\tilde{Y}_u$ be the set of interpolated measurements given by RBF over X_u. For the search phase of the NAS we have employed two strategies described below.

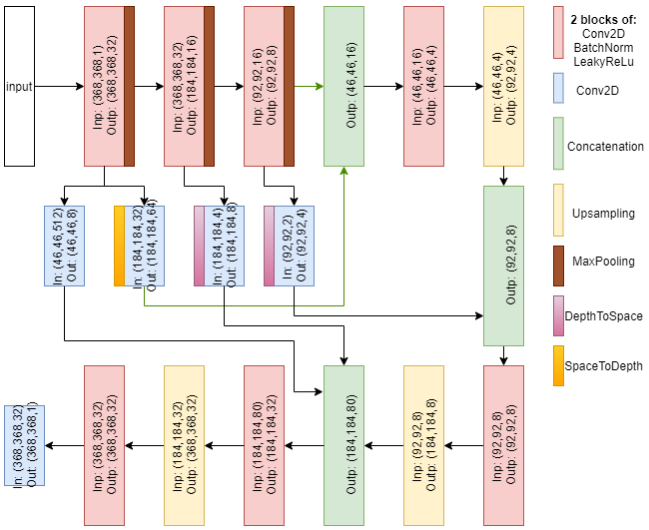


Figure 6.2: Example of the Neural network architecture found by the Architecture Search phase for the `RSSI` *Map of the city of Grenoble used in our experiments.*

Genetic Algorithm (GA)

From the set $(X_\ell, Y_\ell) \cup (X_u, \tilde{Y}_u)$, we use an evolutionary algorithm similar to (Real et al., 2019). Here, the validation set (X_v, Y_v) is put aside for hyperparameter tuning. Here, we also use a macro search space, where edges of the DAG represent data flow with only one input for each node, which is a single operation chosen among a set of candidate operations. We consider usual operations in the image processing field; that are a mixture of convolutional and pooling layers. We also consider three variants of 2D convolutional layers as in (Ulyanov et al., 2017) with kernels of size 3, 5 and 7; and two types of pooling layers that compute either the average or the maximum on the filter of size 4. Candidate architectures are then built from randomly sampled operations and the corresponding `NN` models are trained. The 30 resulting architectures are then ranked according to a pixel-wise Mean Absolute Error (`MAE`) criterion between the interpolated result of the network and the set of interpolated measurements given by `RBF` $\tilde{Y}_u$. The most performing one is finally selected for mutation and placed in the trained population. The oldest architecture is removed in order to keep the size of the population equal to 20 as in (Real et al., 2019). Figure 6.2 illustrates such an optimized architecture with 18 nodes, which was found for the `RSSI` Map of the city of Grenoble used in our experiments (Section 6.3.1).

Dynamic Routing (DR)

For the training phase, we employ the same structure and routing process as those proposed in (Li et al., 2020) (Figure 6.3). The structure is composed of 4 down-sampling level, where the size of the features map is divided by 2 at each level, but the depth of the latter is multiplied by 2 using a 1×1 2D-convolution. In ours experiments, we use a network of 9 layers, which correspond to 33 cells in total (in yellow on Fig.6.3). The structure also contains an *"upsampling aggregation"* module at the end (red part on Fig.6.3). The goal of this module is to combine the features maps from all levels and reconstruct a map of the size of the input. Different from (Li et al., 2020), here, each cell contains three *transforming* operations (i.e. 2D-convolution with a kernel size of 3, 5 or 7) to have a good point of comparison with the method described above. However, due to the structure of the network we decided not to use pooling operations, as this could have been potentially redundant. In addition, we have left the possibility of creating residual connections by adding operation identity in each cell. Moreover, we did not use the first two convolutions,

originally used to reduce the size of the input, in order to keep as much information as possible. Instead, we used a 1×1 2D-convolution (in purple on Fig.6.3).

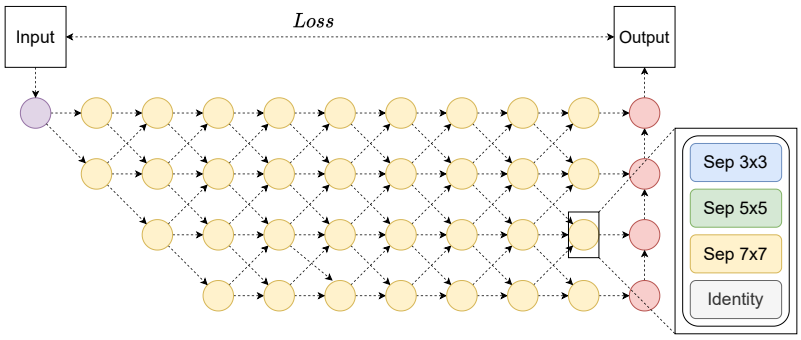


Figure 6.3: Diagram of the architecture used in our experiments. The purple, yellow and red dots represents respectively the "stem" convolution, the cells and the "upsampling aggregation" module. The arrows represent the data flow.

6.2.3 Data-augmentation and Self-Learning phases

After the search phase, the found NN model with parameters θ, f_θ is trained on (X_ℓ, Y_ℓ) by minimizing the following loss :

$$\mathcal{L}(X_\ell, Y_\ell, \theta) = \ell(f_\theta(X_\ell), Y_\ell) + \lambda\|\theta\|_2^2 + \mu\Omega(f_\theta(X_\ell)) \tag{6.1}$$

where $\ell(.)$ is the Mean Absolute Error (MAE), and $\Omega(f_\theta(X_\ell))$ is the total variation function defined as:

$$\Omega(Z) = \sum_{i,j} \mid z_{i+1,j} - z_{i,j} \mid + \mid z_{i,j+1} - z_{i,j} \mid, \tag{6.2}$$

with $z_{i,j}$ the measurement value of a point of coordinates i, j in some signal distribution map Z. This function estimates the local amplitude variations of points in Z that is minimized in order to ensure that neighbor points will have fairly close predicted measurements (i.e., preserving signal continuity/smoothness). Here, λ and μ are hyperparameters for respectively the regularization and the total variation terms, and they are found by cross-validation.

With Dynamic routing used in the search phase, we optimize the network structure and the learning of parameters minimizing (Eq. 6.1) at the same time. Referring to Algorithm 1, the step 1 and 2 are combined in this case.

Let $\theta_1^\star$ be the parameters of the optimized NN model found by minimizing the loss (6.1) on ground truth measurements (X_ℓ, Y_ℓ). This model is then applied to randomly chosen points, $X_u^{(k)}$, in X_u and pseudo-RSSI measurements $\tilde{Y}_u^{(k)}$ are obtained from the predictions of the optimized NN model $f_{\theta_1^\star}$: $\tilde{Y}_u^{(k)} = f_{\theta_1^\star}(X_u^{(k)})$.

With the same NN architecture, a second model $f_{\theta_2^\star}$ is obtained by minimizing the loss (6.1) over the augmented training set $(X_\ell, Y_\ell) \cup (X_u^{(k)}, \tilde{Y}_u^{(k)})$.

6.3 Experiments

In this section, we will first describe our experimental setup and then present our experimental results.

6.3.1 Experimental Setup.

In all experiments, we considered maps of size 368×368 cells and tested our algorithm on field data from two distinct urban environments, namely the cities of Grenoble (France) and Antwerp (The Netherlands). We aggregated and averaged the given measurements in cells/pixels of size 10 meters x 10 meters. The Antwerp dataset is described in detail in (Aernouts et al., 2018) on which we considered three base stations, BS'_1, BS'_2 and BS'_3, with respectively 5969, 6450 and 7118 ground-truth measurements. For the Grenoble dataset, we collected GPS-tagged LoRa RSSI measurements with respect to 2 base stations located in different sites BS_1 and BS_2 with respectively 16577 and 7078 ground truth measurements. To perform in-cell data aggregation, we measured the distances based on local East, North, Up (ENU) coordinates. Then in each cell, we also computed the mean received power over all in-cell measurements (once converted into RSSI values), before feeding our algorithm and the averaged RSSI values have been normalized between 0 and 1.

For each base station, 8% of the pixels with ground-truth measurements were chosen for training (X_ℓ, Y_ℓ), 2% for validation (X_v, Y_v) and the remaining 90% for testing (X_t, Y_t).The unlabeled data used in **Step 3** of Algorithm 1 were selected at random from the remaining 4% of each map cells with no ground truth measurements. Results are evaluated over the test set using the MAE, dB, estimated after re-scaling the normalized values to the natural received signal strength ones. The reported errors are averaged over 20 random sets (training/validation/test) of the initial ground-truth data, and unlabeled data were randomly chosen for each experiment.

We compare Total Variation (TV) in-painting (Eg. 6.2), Radial basis functions (RBF) (Bishop, 2006) with linear kernel that were found the most performant, kriging (KRIG) (Oliver and Webster, 1990), and Navier-Stocks (NS) (Bertalmio et al., 2001) state-of-the-art interpolation techniques with the proposed SL_{NAS} approach. For the latter, we employ both search phase methods based on Genetic Algorithm (GA) and Dynamic Routing (DR) and respectively referred to as SL_{NAS}-GA and SL_{NAS}-DR. For SL_{NAS}-GA we also evaluate the impact of the self-training step (**Step 3**) (called $\text{SL}_{\text{NAS}}\text{-GA}(f_{\theta_2^\star})$) by comparing it with the NN model found at **Step 1** (called $\text{SL}_{\text{NAS}}\text{-GA}(f_{\theta_1^\star})$). The evolutionary algorithm in the architecture search phase (Section 6.2.2) was implemented using the NAS-DIP (Ho et al., 2020) package[1]. The latter was developed over the Deep Image Prior (DIP) method (Ulyanov et al., 2017) which is a NN model proposed for image reconstruction. By considering RSSI maps as corrupted images with partially observed pixels (ground-truth measurements), we also compare with this technique by training a NN model having the same architecture as the one presented in (Ulyanov et al., 2017) and referred to as DIP in the following. All experiments were run on Tesla V100 GPU.

As an unsupervised method, DIP (Ulyanov et al., 2017) uses an exact architecture of generative neural network, as a prior to the image for each of the tasks (in the original paper, they do super resolution, inpainting, denoising and image reconstruction). The network does not need external dataset for training, only the structure of the generative network itself to complete corrupted image. For each exact image, the algorithm finds the best parameters of the network for observed values of the pixels

6.3.2 Experimental Results.

Table 7.1 summarizes results obtained on the five considered RSSI maps. We use boldface to indicate the lowest errors. The symbol $^\downarrow$ indicates that the error is significantly higher than the best result with respect to Wilcoxon rank sum test used at a p-value threshold of 0.01 (Wilcoxon, 1945). In all cases, SL_{NAS}-GA and SL_{NAS}-DR perform better than other state-of-the-art models even without the data-augmentation and self-training steps ($\text{SL}_{\text{NAS}}\text{-GA}(f_{\theta_1^\star})$). We notice that DIP

[1] https://github.com/Pol22/NAS_DIP

Table 6.1: Average values of the MAE, dB of different approaches on all base stations.

	Grenoble		Antwerp		
	BS_1	BS_2	BS'_1	BS'_2	BS'_3
KRIG (1990)	5.68$^{\downarrow}$	4.21$^{\downarrow}$	3.69$^{\downarrow}$	4.39$^{\downarrow}$	4.91$^{\downarrow}$
NS (2001)	5.11$^{\downarrow}$	3.14$^{\downarrow}$	4.28$^{\downarrow}$	3.45	3.87
RBF (2006)	5.03$^{\downarrow}$	3.16$^{\downarrow}$	3.58$^{\downarrow}$	3.35	3.90
DIP (2017)	5.14$^{\downarrow}$	3.22$^{\downarrow}$	3.53	3.41	3.92
TV	5.13$^{\downarrow}$	2.89	3.76	3.51	3.83
SL_{NAS}-DR	4.82	2.82	3.48	3.42	3.81
SL_{NAS}-GA($f_{\theta_1^\star}$)	4.79	2.81	3.39	**3.27**	3.75
SL_{NAS}-GA($f_{\theta_2^\star}$)	**4.76**	**2.79**	**3.33**	**3.27**	**3.74**

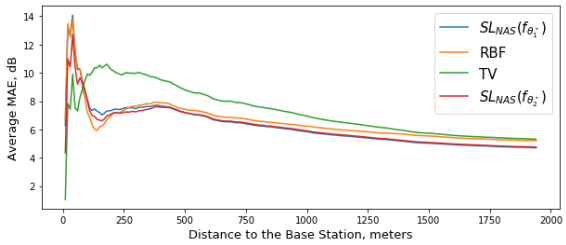


Figure 6.4: MAE, dB with respect to the distance to the base station, BS_1.

which is also a NN based model but with a fixed architecture has similar results than RBF. These results show that the search of an optimized NN model is effective for RSSI map reconstruction in a constrained low-cost and low-complexity IoT context.

Figure 6.4 depicts the average MAE in dB with respect to the distance to the Base Station BS_1 for the city of Grenoble. For a distance above 250 m, SL_{NAS}-GA($f_{\theta_2^\star}$) provides uniformly better predictions in terms of MAE. These findings point to future research into how the model predicts the signal dynamics in regions where the signal is more irregular and where the dynamics are strong (for example near the base stations), especially in the cases where extra contextual knowledge about the physical environment may be included into the learning process (e.g., typically as a side information channel or the city map).

Figure 6.5 displays the MAE, dB boxplots of DIP, RBF and SL_{NAS}-GA ($f_{\theta_2^\star}$) on BS_1 for different percentages of unlabeled data used in the self-learning phase (Section 6.2.3). We notice that by increasing the size of unlabeled examples, the variance of MAE for SL_{NAS}-GA($f_{\theta_2^\star}$) increases mostly due to the increase of noisy predicted signal values by $f_{\theta_1^\star}$. This is mostly related to learning with imperfect supervisor that has been studied in semi-supervised learning (Amini et al., 2009; Krithara et al., 2008). As future work, we plan to incorporate a probabilistic label-noise model in **step 3** of algorithm 1 and to learn simultaneously the parameters of the NN and the label-noise models.

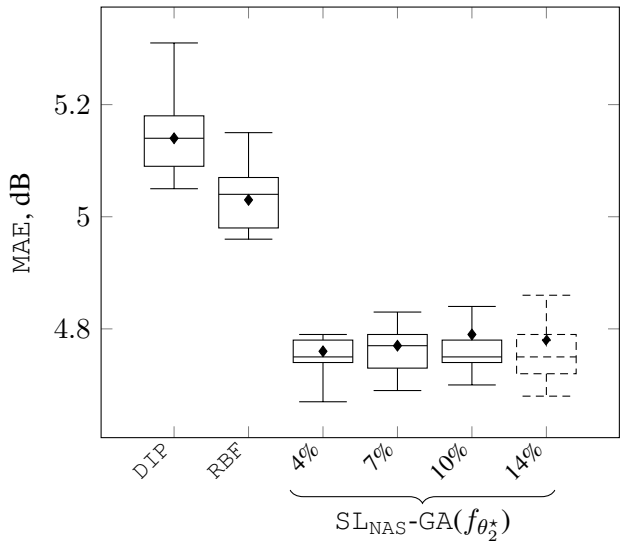


Figure 6.5: Boxplots showing the MAE, dB distributions of DIP, RBF and SL_{NAS}-GA $(f_{\theta_2^\star})$ on BS_1 for different percentage of unlabeled data $\{4, 7, 10, 14\}$ used in the self-learning phase.

6.4 Summary & Discussion

In this chapter, we presented a Neural Network model based on NAS and self-learning for RSS map reconstruction from sparse single-snapshot input measurements in the absence of data augmentation via side deterministic simulations. We presented two variants for the search phase of NAS based on Genetic algorithm and Dynamic routing. Experimental results on five large-scale maps of RSS measurements reveal that our approach outperforms non-learning based interpolation state-of-the-art techniques and NN with manually designed architecture.

This work showed that, despite a limited amount of annotated data, it was able to find a powerful architecture using adapted techniques. To further explore the NAS approach under a regime with minimal annotated data, we have investigated the task of semantic segmentation since 1) it has a particularly cumbersome annotation process, and hence labels are usually difficult to acquire, and 2) state-of-the-art models are usually complex and hand-crafted, and therefore well suited to study efficient NAS. This approach under a semi- and self-supervision regime will be discussed in the next chapter.

7

NAS with Partially Labeled Data for Semantic Segmentation

In this chapter, we propose an architecture search framework with the constraint of using only a small amount of annotated data. As a result, we investigated many approaches, more or less complex, to determine which was the most efficient and how certain parameters could affect performance. We tested these approaches on the two most popular datasets of the state-of-the-art. Furthermore, we offered a modification of this framework in order to broaden the search to a higher level; the results will be reviewed as well.
This chapter present works from the following contributions:

- Se^{2}NAS: Self-Semi-Supervised architecture optimization for Semantic Segmentation [7; 8; 4]

Contents

Semantic segmentation entails in assigning a specific class to each pixel in an image with the overall aim of discovering objects. It is a key task in the field of computer vision, and has a wide range of applications, including autonomous driving (Badrinarayanan et al., 2017), medical research (Ronneberger et al., 2015), facial recognition (Müller et al., 2021) or person reidentification (Wu et al., 2020). In comparison to other computer vision tasks, the equivalent of this pixel-level label is a difficult and time-consuming effort.

The key challenges here are taking into account the context of objects inside images (Mottaghi et al., 2014), and, learning with a small set of annotated data together with a large set of unlabeled data. To address these issues, different approaches have been proposed. For automatically taking into account the context, various approaches have been proposed under the self-supervised framework, which consists in exploiting the underlying data structure in order to gain supervision for an auxiliary task and learn a model by resolving both the auxiliary and the semantic segmentation problems simultaneously. For the problem of learning with partially labeled training data, i.e. in semi-supervised learning setting, existing approaches mostly assign pseudo-labels to unlabeled training data in order to augment the labeled training set using an auxiliary loss; under various perturbations such as images augmentations (French et al., 2020a), features (Ouali et al., 2020) or network (Ke et al., 2019) perturbations; for enforcing the consistency of predictions.

In this chapter, we intend to address the second objective of this thesis in further detail, as well as to optimize this work to be as efficient as possible. We propose an end-to-end method combining NAS and dynamic routing in a semi-supervised setting. Our approach is based on firstly searching cell operation and secondly searching for the most adapted path according to the input on the fly.

Moreover, we conduct the preliminary study with different settings where the model architecture is obtained by dynamic routing in order to identify the most efficient way to leverage unlabeled data. We, then, transpose the most efficient way to leverage unlabeled data to the proposed end-to-end methods. We compare the performance of the proposed methods on various partitions in end-to-end way to the previously established baselines.

The performance of the architecture search is shown on known semantic segmentation benchmarks under different partition set-ups and are further compared to a state-of-the-art hand-crafted architecture.

This chapter is organized as follows. Section 7.1 presents some background of semantic segmentation. Section 7.2 provides details about the techniques studied in our framework. Section 7.3, exhibits our approach, and finally, in Section 7.4 we present experimental results obtained during the study and by the proposed approach on Cityscapes and Pascal VOC 2012 benchmarks.

7.1 Background

Semantic segmentation consists in classifying each pixel of an image into a class, where each class represents an object or a portion of the image (Liu et al., 2019c) (e.g. fig. 7.1). This task is part of the scene understanding concept, which is much more complex than image classification, as it requires apprehending the whole context of an image. To comprehend a scene, each piece of visual data must be assigned to an entity while taking into account the spatial information.

To measure the model performance on this task, the Mean Intersection over Union (mIoU) is used. The IoU is first computed by class, following equation 7.1 (see also fig. 7.2).

$$IoU_c = \frac{TP}{TP + FP + FN} \tag{7.1}$$

Figure 7.1: Example of image and the associated segmentation mask. Illustration from (Chen et al., 2017)

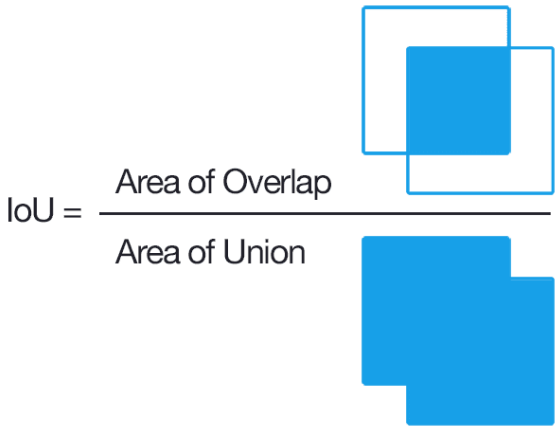


Figure 7.2: Visual example of IoU

Where for a given class c, TP and FP denote the number of true and false positives, respectively, and FN denotes the number of false negatives. The IoUs are then averaged to obtain mIoU, the more this metric is high, the better the network is in its predictions.

Recent research on this topic has largely relied on Neural-Networks, as these models have been shown to outperform other techniques in scene analysis (Long et al., 2015).

Hand-crafted architectures, designed by experts in the field, are the most popular way to create specific NN models for semantic segmentation. In this category, a wide range of architectures requiring high computational resources exist, including U-Net (Ronneberger et al., 2015), Conv-Deconv (Fourure et al., 2017) or FCN (Long et al., 2015). However, (Yu et al., 2018) have shown that, by dissociating context information from the spatial information, it is possible to achieve highly efficient models with lighter architectures.

Recently, (Chen et al., 2018a; Nekrasov et al., 2019b), have studied how Neural Architecture Search (NAS) can be applied to the decoder in order to improve the performance for semantic segmentation. However, different from the proposed study, these proposed NAS techniques rely on a fully supervised learning framework.

7.2 Baseline

In this section, we will present the baselines as well as some definitions of notations that will be used throughout this chapter.

7.2.1 Definitions

We assume that we have a labeled training set $\mathcal{D}_\ell = (x_l, y_l)_{1 \leqslant l \leqslant m}$ of size m, and a possibly much larger set of unlabeled training examples $\mathcal{D}_u = (x_u)_{m+1 \leqslant u \leqslant m+n}$ of size n. We further consider that θ represents the set of all network weights.

In our setting, we consider the jigsaw solving pretext task as the self-supervised method. The main motivation here is that the jigsaw task have shown good performance of discovering architecture on the dataset of interest (Liu et al., 2020). On the other hand, there is no apparent consensus in the literature on which semi-supervised approach is the most efficient for semantic segmentation. Here we considered the Mean-Teacher, the Co-Teaching and Augmentation based approaches, which have been increasingly popular in recent years. Our goal is to investigate their efficacy in the context of semantic segmentation using neural architecture search. Depending on the self-supervised and semi-supervised $method$, we define $\mathcal{L}_u^{method}$ as the unsupervised loss related to the considered approach (i.e. $method \in \{ssl, ssup\}$). In all scenarios, the supervised loss $\mathcal{L}_s$ is based on an individual loss ℓ_d defined as the cross-entropy loss.

7.2.2 Self-supervised regularization

For the self-supervised learning method, a geometric transformation is applied to the inputs for the pretext task, and a label is generated. For each labeled training example, $(x_l, y_l) \in \mathcal{D}_\ell$ this transformation on the input x_l acts as an augmentation and the same transformation is applied on y_l. For an unlabeled example $x_u \in \mathcal{D}_u$, the label produced by the transformation, y_u, is used as the ground truth with respect to the pretext task. For the jigsaw task, proposed in (Noroozi and Favaro, 2016), the key idea is to learn visual representations for puzzles solving. In practice, this task consists in cutting images in 9 patches from a grid of 3×3. The patches are then mixed using specified random permutations, and the network is trained to predict the permutation in question in order to solve the problem. Along with the supervised semantic segmentation problem, one or more distinct pretext tasks can be considered in this framework. In addition, unlike other state-of-the-art approaches, just one network is employed, and the perturbation is applied to the input via geometric transformation.

In our experiments, we followed a similar approach to (Zhai et al., 2019), by training the network in a multitask manner, where a supervised loss ($\mathcal{L}_s$) is minimized along a self-supervised loss ($\mathcal{L}_u^{ssl}$) that acts as a regularizer. In this case, for a given permutation $jig \in \{1, \ldots, k\}$ where k is the total number of considered permutations; the problem of jigsaw solving can be formulated as a classification task using the produced pretext labels for both labeled $(y_l^{jig})_{1 \leqslant l \leqslant m}$ and unlabeled training samples $(y_u^{jig})_{1 \leqslant u \leqslant n}$. The self-supervised loss function is, in this case, the average cross-entropy loss:

$$\mathcal{L}_u^{ssl} = \frac{1}{k} \sum_{jig=1}^{k} \left(\frac{1}{m} \sum_{\mathcal{D}_\ell} \ell_{ce}(p_l^{jig}, y_l^{jig}) + \frac{1}{n} \sum_{\mathcal{D}_u} \ell_{ce}(p_u^{jig}, y_u^{jig}) \right)$$

where $\ell_{ce}(.)$ is the individual cross-entropy loss function. In all of ours experiments, we used a fixed set of $k = 100$ permutations, as in (Noroozi and Favaro, 2016). As supervised loss, the averaged cross-entropy over the labeled training set is used:

$$\mathcal{L}_s = \frac{1}{m} \sum_{\mathcal{D}_l} \ell_d(p_l, y_l), \tag{7.2}$$

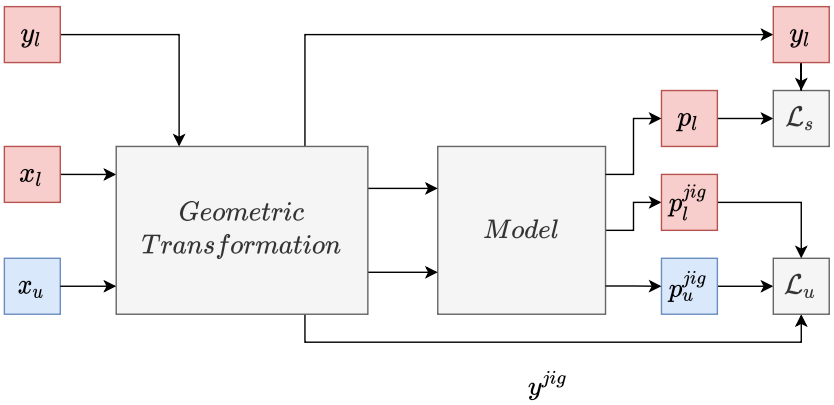


Figure 7.3: Illustration of the self-supervised learning strategy, x_u is an unlabeled sample and (x_l, y_l) is a labeled training example. The label y_l^{jig} and y_u^{jig} are the transformations associated to x_l and x_u. $\mathcal{L}_s$ and $\mathcal{L}_u$ are respectively the supervised and unsupervised loss functions. $(p_l)_{1\leqslant l\leqslant m}$ are predictions for the supervised semantic segmentation task, p_l^{jig} and p_u^{jig} are the prediction for the pretext task.

7.2.3 Semi-supervised Mean Teacher method

The mean-teacher method was developed for semi-supervised classification (Tarvainen and Valpola, 2017) and has lately been adapted to semi-supervised semantic segmentation (French et al., 2020a).

This approach is based on consistency regularization that constraints a model to have the same output for a given input. The underlying model is made of two neural networks of the same structure, one denoted as *student*, $f(\theta)$, and the other called *teacher*, $f(\overline{\theta})$, where $\overline{\theta}$ denote the exponential moving average (i.e. EMA) of the parameters of the student, and are iteratively computed as:

$$\overline{\theta}_t = \alpha\overline{\theta}_{t-1} + (1-\alpha)\theta_t, \tag{7.3}$$

with α a hyperparameter used to control the dependency between the two networks. In this method, only the *student* is trained using the labeled training set and the predictions of the teacher. Let $(p_l^s)_{1\leqslant l\leqslant m}$ and $(p_l^t)_{1\leqslant l\leqslant m}$ be the outputs predicted by the *student* and *teacher* networks on labeled examples; and, $(p_u^s)_{1\leqslant u\leqslant n}$ and $(p_u^t)_{1\leqslant u\leqslant n}$ predictions of both networks on unlabeled samples. The consistency regularization is achieved by restricting the distribution outputs of both networks to be as close to each other as possible on labeled and unlabeled samples provided as inputs to both models; and that by minimizing the following unlabeled loss by the student:

$$\mathcal{L}_u^{ssup-mt} = \frac{1}{m}\sum_{\mathcal{D}_\ell} \ell_{MSE}(p_l^t, p_l^s) + \frac{1}{n}\sum_{\mathcal{D}_u} \ell_{MSE}(p_u^t, p_u^s) \tag{7.4}$$

where ℓ_{CE} is the per-pixel cross-entropy summed up over all pixels and classes. The supervised loss of the student is based on the *teacher* outputs and the ground truth of the labeled training examples:

$$\mathcal{L}_s = \frac{1}{m}\sum_{\mathcal{D}_l} \ell_d(p_l^t, y_l), \tag{7.5}$$

Figure 7.4 illustrates this strategy. At the beginning, both networks have the same initial weights. At each iteration, the parameters of the student are first updated by minimizing both the supervised and unsupervised losses; then the parameters of the teacher are updated by EMA (c.f. eq. (7.3)). Following (French et al., 2020a), we fixed α to 0.99 in our experiments.

7.2.4 Semi-supervised co-teaching method

This method is well-known in the deep learning community, and it has been widely used in different problems, including semantic segmentation (Chen et al., 2021). Similarly to the mean teacher

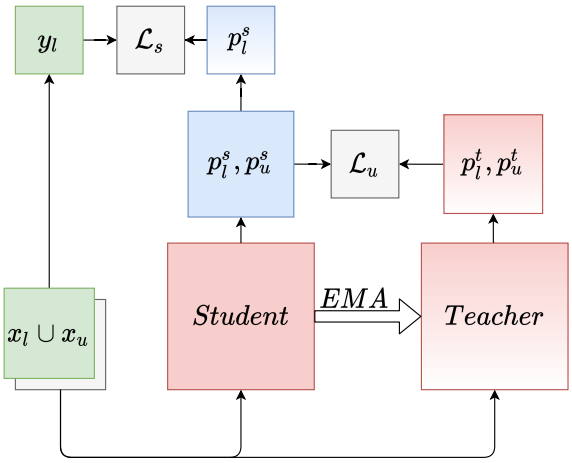


Figure 7.4: Illustration of mean teacher process. p_l^s, p_u^s and p_l^t, p_u^t are predictions for labeled and unlabeled samples, by the student and the teacher, respectively. The EMA arrow stand for exponential moving average of weights from the student to the teacher.

approach, co-teaching uses two networks of the same architecture (e.g. $f(\theta_a)$, $f(\theta_b)$). However, differently from the previous approach, here the two networks update their weights and are independently trained. The main idea is that each network can learn from the other one. Formally, given an unlabeled input x_u, each network, i.e. $f(\theta_a)$ and $f(\theta_b)$, predicts an output p_u^a and p_u^b. Then by one-hot encoding, these outputs are transformed to pseudo-labels; $\tilde{y}_u^a$ and $\tilde{y}_u^b$; which serve as ground truth for the other network, and the definition of a cross-pseudo supervision loss over the unlabeled data:

$$\mathcal{L}_u^{cps} = \frac{1}{n} \sum_{\mathcal{D}_u} (\ell_{ce}(p_u^a, \tilde{y}_u^b) + \ell_{ce}(p_u^b, \tilde{y}_u^a)), \tag{7.6}$$

Similarly to $\mathcal{L}_u^{cps}$, a cross-pseudo supervision loss $\mathcal{L}_l^{cps}$ can be defined on the examples in $\mathcal{D}_\ell$ and the unsupervised loss is defined as the sum of these two losses:

$$\mathcal{L}_u^{ssup-ct} = \mathcal{L}_u^{cps} + \mathcal{L}_l^{cps} \tag{7.7}$$

The supervised loss that is used for the training of both networks is defined as:

$$\mathcal{L}_s = \frac{1}{m} \sum_{\mathcal{D}_l} (\ell_d(p_l^a, y_l) + \ell_d(p_l^b, y_l)), \tag{7.8}$$

Figure 7.5 illustrates the whole process of co-teaching. In our setup both models, $f(\theta_a)$ and $f(\theta_b)$ are dynamic, and their weights are found independently one from the other.

7.2.5 Strong Data Augmentation

Data augmentation techniques are a common practice among deep learning community, and demonstrated as affective in a semi-supervised (Xie et al., 2020a) or self-supervised (He et al., 2020) learning context. Motivated by the simplicity and the performance of FixMatch (Sohn et al., 2020), we investigate the performance of the architecture optimization using a similar learning scheme. The main idea of this learning scheme is based on the consistency error minimization, which constraints the network prediction of a sample and its perturbed (augmented) version to be the same. To achieve this, two types of augmentation (weak and strong) are used on the unlabeled sample data. First, a weakly augmented unlabeled sample is run through a teacher to obtain a prediction that will serve as pseudo annotations. Weak augmentation typically includes random scaling, random cropping or random vertical/horizontal flipping. Then, this same weakly augmented image

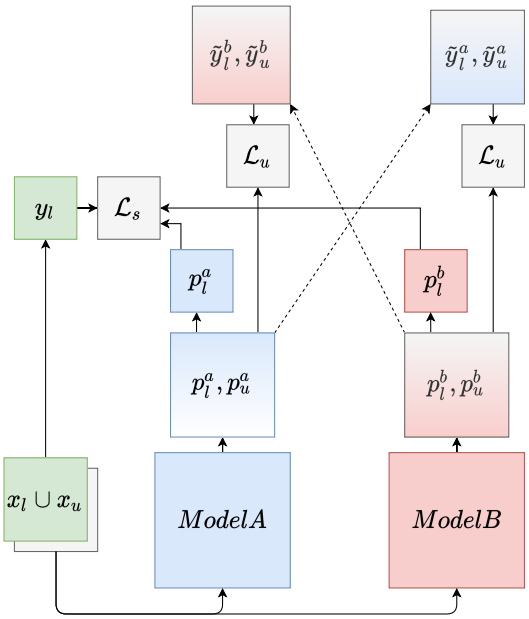


Figure 7.5: An illustration of the co-teaching workflow, where x_u is an unlabeled sample, (x_l, y_l) is a labeled training example. The terms p_l^a, p_u^a and p_l^b, p_u^b are predictions for labeled and unlabeled samples of the Model A and B respectively. $\tilde{y}_l^a, \tilde{y}_u^a, \tilde{y}_l^b, \tilde{y}_u^b$ are the associated pseudo-labels obtained with one-hot encoding.

is passed through other so-called strong augmentations, to obtain a strongly augmented image. Finally, this strongly augmented image will pass through the student and the prediction obtained will be compared with the pseudo-annotation obtained previously. This process is illustrated in Figure 7.6. More formally, we define a weak augmentation as $\mathcal{A}_w$ and strong augmentations as $\mathcal{A}_s$. As strong augmentations, the most commonly used set has been selected, which is formed of colorjitter, Gaussian blur and grayscale as in many works (Chen et al., 2020c; Chen et al., 2020a). To complete this set with different transformations, the impact of three image mixing methods, namely CowMix (French et al., 2020b), CutMix (French et al., 2020a) and ClassMix (Olsson et al., 2021), has been investigated. Those augmentations act like perturbation and are combined

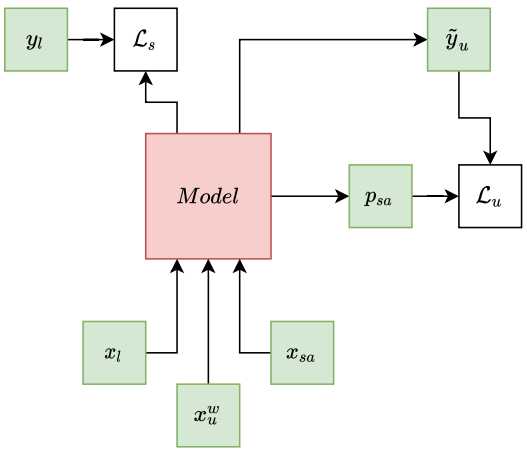


Figure 7.6: Illustration of Strong Data Augmentation process, as proposed in (Sohn et al., 2020). In this illustration, (x_l, y_l) is a pair of annotated samples, x_u^w and x_{sa} are the the weakly and strongly augmented version of the unlabeled sample x_u, respectively. The term p_{sa} is the model prediction for the input x_{sa} and $\tilde{y}_u$ is the per-pixel pseudo label generated by the model from x_u^w.

with a weak augmentation. More formally, a strongly augmented sample is given by :

$$x_{sa} = \mathcal{A}_s(\mathcal{A}_w(x_u)) \tag{7.9}$$

The unlabeled loss is given by the following equation :

$$\mathcal{L}_u^{ssup-sda} = \frac{1}{n} \sum_{\mathcal{D}_u} \ell_{CE}(p_{sa}, \tilde{y}_u) \tag{7.10}$$

where $p_{sa} = Model(x_{sa})$ is the prediction of the network on the strongly augmented unlabeled sample x_{sa} and $\tilde{y}_u$ is the per-pixel pseudo label generated by the network on the weakly augmented unlabeled sample x_u^w (i.e. $\tilde{y}_u = \max(Model(\mathcal{A}_w(x_u)))$. The function ℓ_{CE} is the standard cross-entropy loss function.

7.3 Se^{2}NAS: Semi-Supervised learning with Neural Architecture Search

This section presents our semi-supervised learning approach for semantic segmentation based on Neural Architecture Search, that we call Se^{2}NAS.

7.3.1 Se^{2}NAS learning scheme

This study take place in a context similar to that proposed by AutoDeepLab (Liu et al., 2019a). The end-to-end search, in this work, is defined as: searching for (1) the inner cell operation and (2) the optimal paths to use to navigate through those cells depending on the input. Here, the learning problem for semantic segmentation is to jointly find an optimized architecture of a neural network and its parameters θ, that minimize the weighted sum of a supervised loss, $\mathcal{L}_s$; and up to two unsupervised losses defined from self-supervision, $\mathcal{L}_u^{ssl}$, and semi-supervised learning $\mathcal{L}_u^{ssup}$:

$$\mathcal{L}_{total} = \lambda_0 \mathcal{L}_s + \lambda_1 \mathcal{L}_u^{ssl} + \lambda_2 \mathcal{L}_u^{ssup} \tag{7.11}$$

Different from (Liu et al., 2019a), which propose a two-stage search by firstly find the structure and secondly perform a retraining, we investigate the possibility of doing it all at once. Moreover, this process is performed in a semi-supervised manners. The methods that have been used will be described in the two following subsections. The process is illustrated in figure 7.7 and pseudocode is given by Algorithm 2.

Step 1: Cell search

Instead of searching for a complex cell with multiples operations, repeated through the network such as in (Liu et al., 2019a), we take another approach by seeing each cell independently. Nevertheless, instead of searching for multiples layers and connections per cells, we simplify the problem by searching among predefined types of cells. We based this part of the search on the work proposed by DSNAS (Hu et al., 2020). In this part, each cell is constructed by sampling a one-hot vector $\mathbf{Z}$ from a categorical architecture distribution $p_\beta(\mathbf{Z})$, where β is the parameters for architecture distribution. This parameter β will be optimized through time to select the most suited operation type for each cell. During this step, the searching of optimal path is disabled by setting all connections between cells to 1. Similar to (Hu et al., 2020), the list of predefined cells are inspired by ShuffleNet V2 (Ma et al., 2018), which seems to have good performance in segmentation (Yu et al., 2021a). We investigate 4 types of cells, namely, one with three different

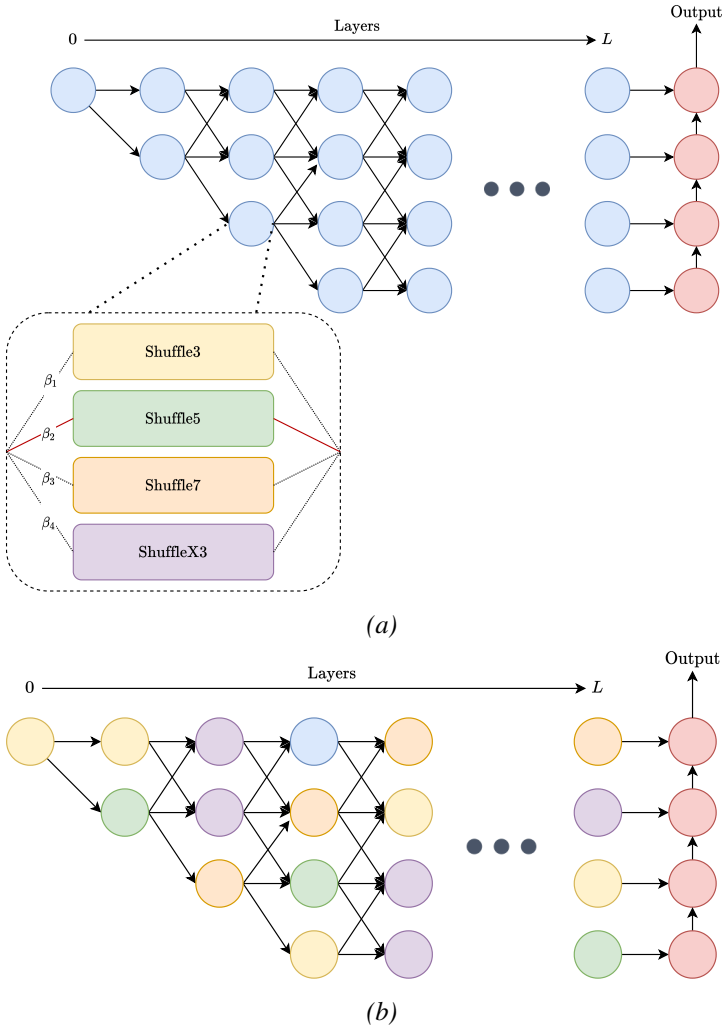


Figure 7.7: Illustration of the 2 steps of end-to-end search in `Se`^{2}`NAS`. *The top figure shows the step 1, where the goal is to search the most fitted operation in each cell, the connections between layers (arrow) are fixed to 1.0. The bottom figure shows the step 2: once the operation have been found and fixed (step 1), the connection between layers are searched (i.e. Routing process).*

kernel size ($K = 3, 5, 7$) and one deeper. Those cells are illustrated in figure 7.8a and 7.8b respectively. This process is illustrated in Figure 7.7a. In this figure, each cell have a list of associated β, among which at each iteration we sample one operation per cell (i.e. red path) to build our vector $\mathbf{Z}$. During the backward pass, this list of β is optimized with respect to the loss. In the proposed framework, this option can be enabled by setting `use_cell_search=True` in Alg. 2. This special case is called `Se`^{2}`NAS-b`.

Step 2: Routing process

Dynamic networks have exhibited superiority in network capacity and greater performance with budgeted resource use, by fitting the model architecture to the input data. Among different approaches, dynamic routing (Li et al., 2020), on which we base our routing algorithm, has the advantage of allowing the transfer of weights from a prior training, that has become more essential in terms of time savings.

In our approach, the routing space (or structure) noted f_θ, is defined as a 4-level network with L layers composed of cells (Figure 7.7b). Each level in this structure represents a stride rate, where the size of the input is successively reduced by descending in the network. The strides rates

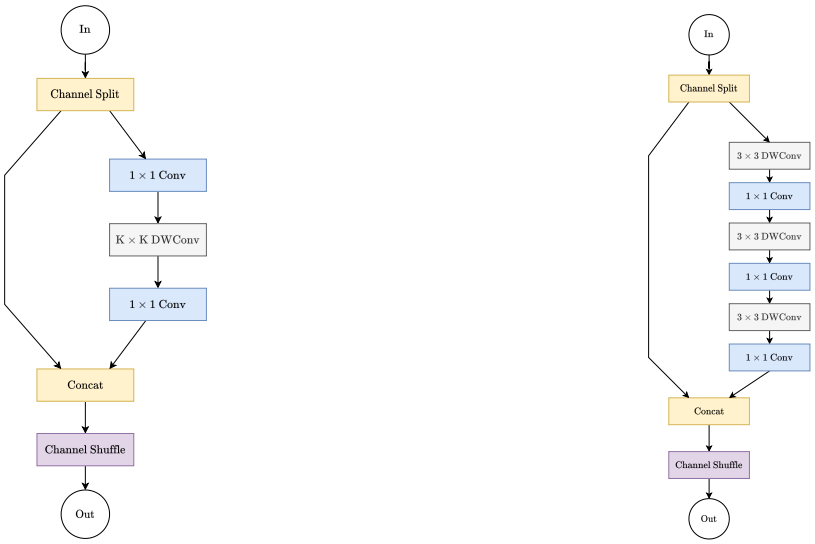


Figure 7.8: Illustration of the ShuffleNetV2 cells. We studied 4 types of cells distributed as follows, 3 of type (a) with different kernel size 3,5,7 and one of type (b). DWConv denote depth-wise convolution.

are thus $1/4$ for the highest level and $1/32$ for the deepest one. Depending on whether the level is greater or lower, the image ratio is then lowered or raised by 2. The path through the levels is performed by a convolution with a kernel size of 1, while the size is reduced, the convolution increases the number of feature map by 2.

For the initialization of the dynamic structure, we set a fixed 3-layer block *'STEM'*, used to reduce the resolution of the input to $1/4$. Note that in this block, separated convolution are used. At the end, we find a simple decoder (red cells in Figure 7.7b), which go from bottom to top of the levels. This decoder is just a composition of convolution and upsampling operations in order to add the features of each level. Once added, the features go through a last classification layer. Concerning the cell aspect (various color in 7.7b), each cell consists of an operation manually selected or an operation chosen by search (c.f. Step 1 of Sect. 7.3.1) according to the selected context. This structure also allows the multi-path routes and the creation of skip-connections. The choice of the paths in this structure and the parameters of each operation, are all optimized using the gradient descent algorithm.

7.4 Experiments

In this section, the used experimental setup and results will be presented.

7.4.1 Experimental setup

Datasets

In our experiments, we have considered two popular data collections for image segmentation, which are Pascal VOC 2012 and Cityscapes.

Alg. Se^{2}NAS

Input : $\mathcal{D}_u$, $\mathcal{D}_\ell$, E: *epochs*, M: *method*, $\lambda_0, \lambda_1, \lambda_2$: *losses weights*, $p_\beta(\mathbf{Z})$: *arch distribution*

$N \leftarrow \max(|\mathcal{D}_u|, |\mathcal{D}_\ell|)$;
$f_\theta \leftarrow$ `DR_struture()`;
$f_{\theta_0} \leftarrow$ `init()`;
for $e \in$ `range(0,E)` **do**
  for $i \in$ `range(0, N)` **do**
    $t \leftarrow i + e$;
    $\mathfrak{B}_\ell \leftarrow \mathcal{D}_\ell$;
    $\mathfrak{B}_u \leftarrow \mathcal{D}_u$;
    if `use_cell_search` **then**
      $\mathbf{Z} \leftarrow p_\beta(\mathbf{Z})$:
    else
      $\mathbf{Z} \leftarrow$ `fixed_operation()`;
    end if
    $\mathcal{L}_s, \mathcal{L}_u^{ssl}, \mathcal{L}_u^{ssup} \leftarrow M(\mathfrak{B}_l, \mathfrak{B}_u, f_{\theta_t}, \mathbf{Z})$;
    $\mathcal{L}_{total} \leftarrow \lambda_0 \mathcal{L}_s + \lambda_1 \mathcal{L}_u^{ssl} + \lambda_2 \mathcal{L}_u^{ssup}$;
    $f_{\theta_{t+1}}, \beta \leftarrow$`optimize`$(f_{\theta_t}, \mathbf{Z}, \mathcal{L}_{total})$;
  end for
end for
Output : $f_{\theta^\star}$: *Network trained after* E *epochs*

Algorithm 2: Pseudocode of Se^{2}NAS

Pascal VOC 2012 The Pascal VOC dataset (Everingham et al., 2010), which contains 20 object classes and one background category, is a widely used dataset in object semantic segmentation. The original dataset contains almost 13000 images, including 1464 images for training, 1449 for validation, and 1456 for testing as standard splits. We employ the augmented version provided by (Hariharan et al., 2011) as a standard base for our work, raising the total number of usable images for training to 10582.

Cityscapes The Cityscapes dataset (Cordts et al., 2016) is frequently utilized, mostly in the context of analyzing urban scenes. The collection contains 5000 finely annotated images, each with a per-pixel label from one of 19 semantic classes. There are 2975 images for training, 500 for validation, and 1525 for testing in the splits provided, with each image having a resolution of 1024x2048. Following other studies, we solely use the training and validation sets in our experiments.

Evaluation The results are reported on the full validation set for each dataset (500 for Cityscapes, 1449 for Pascal VOC) using the standard mean Intersection-over-Union (mIoU) metric. In all of our experiments, we use single scale testing with no augmentation, the size of the input in evaluation stage is 1024x2048 and 512x512 for Cityscapes and Pascal VOC, respectively.

Preliminary analysis

In this preliminary analysis, we first determine which of the baseline (i.e., which loss) is the most effective to solve the learning problem. For this study, has been kept network simple, therefore

the cell search part was disabled. The cells was defined as in (Li et al., 2020), namely two separable convolution 3×3. So only the routing process is active here (i.e. Step 2), by setting `use_cell_search` to `False` in the Algorithm 2.

For both datasets, similar weak data augmentation to (Chen et al., 2021; Li et al., 2020) have been used, including random scaling, followed by random horizontal flipping, and random square cropping. The scaling factor was taking values in $\{0.5, 0.75, 1, 1.25, 1.5, 2.0\}$. We set the hyper-parameter of the supervised loss in (7.11); $\lambda_0 = 1$ in all of our experiments.

To investigate the effects of semi-supervised and self-supervised settings on the learning of parameters, we respectively set the corresponding hyperparameters, λ_1 and λ_2, to 0 in eq. (7.11). These scenarios will be presented in the next section. We deploy an extra classifier for the pretext task in the self-supervised experiments. Accordingly, the dynamic routing output is taken in the other direction (from top to bottom in Figure 7.7). Then, before the fully connected layer, we apply global average pooling to the features.

For experiments using strong data augmentation, we set the intensity of colorjitter as in (Zou et al., 2021), for grayscale and blur same as (Chen et al., 2020c). The mixing method is applied with a probability of 0.75 to a batch of unlabeled samples. The encoder of the network is initialized by the ImageNet pre-trained weights, provided by (Li et al., 2020), while the others weights are initialized using Kaiming initialization (He et al., 2015). For parameter updates, we used a standard mini-batch SGD with momentum of 0.9, with an initial learning rate of 0.02. In addition, similar to other approaches (Chen et al., 2017; Li et al., 2020), we adopt a polynomial learning rate decay with a power of 0.9. The training batch size is 8 for Cityscapes and 16 for Pascal VOC. Concerning the dynamic routing structure, we follow (Li et al., 2020) and set $L = 16$ to be consistent with a ResNet-50 (He et al., 2016).

End-to-end search

We studied the performance of the proposed end-to-end research compared to the previously stated baseline. So, `use_cell_search` is set to `True` in the algorithm 2 (i.e. `Se`^{2}`NAS-b`). For these experiments, we follow a similar set-up as the previously identified most efficient approach, MOT_{SDA} (see Sect. 7.4.2), during the two phases. As mentioned in 7.3, we decompose the end-to-end research in to two phases. During the first phase, we search for the most suited operation in each cell, similar to (Hu et al., 2020), we apply an early-stopping threshold, in order to save time by not selecting poorly performing operation. When the majority of the cell operation satisfy this early-stopping, we move to the second phase of searching. In this second phase, we fix the cell to the most promising operation selected in phase one, we also transfer the associated parameters and discard all others parameters. We then continue the training process, with enabling the routing process. The β architecture parameters are updated with Adam optimizer (Kingma and Ba, 2014) with an initial learning rate of 0.001 as in (Hu et al., 2020).

7.4.2 Experimental results

For all experiments, the fractions of the labeled dataset are the same as those proposed in (Chen et al., 2021). More precisely, they are four fractions ($1/16$, $1/8$, $1/4$, $1/2$) which represent 186, 372, 744, 1488 and 662, 1323, 2646, 5291 images for Cityscapes and Pascal VOC, respectively. We used random crop of size 768 for Cityscapes and 512 for Pascal VOC, with padding, and an ignored value if necessary.

Preliminary analysis

In this section, we present the experimental results obtained for the preliminary study, to determine the most efficient baseline settings. In all the experiments, we define the "Baseline" as the supervised only version (i.e. $\lambda_1 = \lambda_2 = 0$).

Self-supervised regularization (SSR) : We begin by examining the obtained gain by performing the jigsaw pretext problem discovered by self-supervised learning (Section 7.2.2) simultaneously with the pixel classification task for semantic segmentation. In this scenario, the corresponding hyperparameter of self-supervised learning in eq. (7.11); λ_1 was set to 0.1; and; we disabled the effect of semi-supervised learning by setting the hyperparameter λ_2 to 0. The corresponding model is denoted by $\texttt{Se}^2\texttt{NAS}_{\lambda_2=0}$ and it is compared to the fully supervised setting, in which no self-supervised nor semi-supervised learning is utilized. In the following, $\texttt{Se}^2\texttt{NAS}_{\lambda_1=\lambda_2=0}$, stands for the fully supervised model. Tables 7.1a and 7.1b summarizes results obtained for different fraction of the labeled training set on Pascal VOC and Cityscapes, respectively. The highest performance rates are indicated in boldface. It turns out that the pretext task effectively adds information to semantic segmentation, although the benefits are limited. This could be due to the fact that images are cut using a 3×3 grid for puzzle solving, and the resolution of the jigsaw problem may introduce noise into the pixel classification, particularly where puzzle pieces are cut.

Table 7.1: mIoU of Baseline ($\texttt{Se}^2\texttt{NAS}_{\lambda_1=\lambda_2=0}$) and SSR ($\texttt{Se}^2\texttt{NAS}_{\lambda_2=0}$) obtained on val. set of Cityscapes/Pascal VOC for different fractions of the labeled training set.

(a) Cityscapes

	Fraction of the labeled training set			
	$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{2}$
Baseline	61.07	68.42	72.02	74.84
SSR	**61.41**	**68.87**	**72.18**	**75.11**

(b) Pascal VOC

	Fraction of the labeled training set			
	$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{2}$
Baseline	53.33	59.45	65.21	69.02
SSR	**53.80**	**60.33**	**65.25**	**69.27**

Semi-supervised learning (SSL) : We now investigate the effect of semi-supervised learning alone by setting λ_1 to 0. The resulting model is referred to as $\texttt{Se}^2\texttt{NAS}_{\lambda_1=0}$ in the following. By setting λ_1 to 0, we hence disable the associated geometric transformation.

We compared the effect of various semi-supervised techniques on the proposed framework $\texttt{Se}^2\texttt{NAS}$. In this section, we compared the performance obtained using the Mean Teacher and Co-teaching technique. The performance of each method, with 1/8 of the annotated data used in training, is presented in table 7.2. We can observe that in our context, the results are not consistent. We believe that the discrepancy in performance, with the Co-Teaching approach, is due to the noise introduced when each of the classifiers assigns pseudo-labels to unlabeled examples and results in a snowball effect, reinforcing the predictions of the models in these errors and leading to a performance degradation in the final model. On the other hand, Mean Teacher is based on the consistency regularization with the only constraint that the outputs of the student and the teacher networks should be as close as possible on the same unlabeled examples. There is no label-noise propagation in this situation, and $\texttt{Se}^2\texttt{NAS}$ is able to take advantage of the lack of label information by exploiting the structure of the data more efficiently using the unlabeled set.

Table 7.2: mIoU of $\texttt{Se}^2\texttt{NAS}$ using different techniques of semi-supervison under the 1/8 training settings.

	Cityscapes	Pascal VOC
Co-teaching	**73.10**	64.02
Mean Teacher	72.34	**68.64**

Due to this discrepancy, we employed Mean-Teacher (Section 7.2.3) technique, as semi-supervised techniques. For Mean-Teacher method, the hyperparameter λ_2 in eq. (7.11) was

empirically set to 100 as in (Chen et al., 2021). Tables 7.3a and 7.3b show the performance of $\texttt{Se}^2\texttt{NAS}_{\lambda_1=0}$ using the Mean-Teacher technique, for different fractions of the labeled training sets of Cityscapes and Pascal VOC datasets, respectively.

Table 7.3: mIoU of Baseline ($\texttt{Se}^2\texttt{NAS}_{\lambda_1=\lambda_2=0}$) and SSL ($\texttt{Se}^2\texttt{NAS}_{\lambda_1=0}$) with MT approach, obtained on val. set of Cityscapes/Pascal VOC for different fractions of the labeled training set.

(a) Cityscapes

	Fraction of the labeled training set			
	$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{2}$
Baseline	61.07	68.42	72.02	74.84
SSL	**67.54**	**72.34**	**75.28**	**77.74**

(b) Pascal VOC

	Fraction of the labeled training set			
	$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{2}$
Baseline	53.33	59.45	65.21	69.02
SSL	**65.05**	**68.64**	**72.19**	**73.95**

Mixing of techniques (MOT) : We compare the performance of $\texttt{Se}^2\texttt{NAS}$ under various mixed semi-supervised learning setting using the Mean-Teacher technique to its other versions discussed above on the Cityscapes and Pascal VOC datasets, respectively.

– SSL+SSR : We define MOT_{SSR} as a combination of mean-teacher technique with an added self-supervised regularization. We use the same value of λ as defined before (i.e. $\lambda_1 = 0.1$ and $\lambda_2 = 1$). Results are presented in Tables 7.5a and 7.5b. We can observe in MOT_{SSR} that the pretext task has also here a limited benefit on semi-supervised learning, comforting the idea that, while complementing, puzzle solving and pixel classification are not totally correlated.

– SSL+SDA : In order to improve stability in the predictions for the mixing method, as in (Olsson et al., 2021; French et al., 2020a) , we study the performance of the strong data augmentation (SDA) combined with Mean-Teacher (i.e. MOT_{SDA}). Here we set $\lambda_1 = 0$ and $\lambda_2 = 1$. The pseudo label of the weakly augmented unannotated image is generated by the teacher (i.e. $\tilde{y}_u = Teacher(x_w)$) and the student see the supervised and strongly augmented samples.

For this part, we firstly conduct an ablation study to investigate the impact of the various strong augmentation in the learning scheme. Especially, we have investigated the impact of a mixing method, colorjittering, Gaussian Blur and grayscale. All the results are presented in the smallest setting $(1/16)$ of Cityscapes.

Table 7.4: Comparison of $\texttt{Se}^2\texttt{NAS}$ performance under different augmentations on the 1/16 fraction of Cityscapes

(a) mIoU of $\texttt{Se}^2\texttt{NAS}$ under different mixing type

	mIoU
CowMix	70.80
CutMix	72.13
ClassMix	**72.22**

(b) mIoU of $\texttt{Se}^2\texttt{NAS}$ using "classic" augmentation

	mIoU
Gaussian Blur	67.26
Grayscale	68.06
ColorJitter	69.20
All	**71.57**

We can see in table 7.4a, that the ClassMix is the most effective mixing method. Moreover, we observe in table 7.4b the individual effects of the so-called "classic" augmentations. Finally, the figure 7.9 summarizes the performance of both augmentation

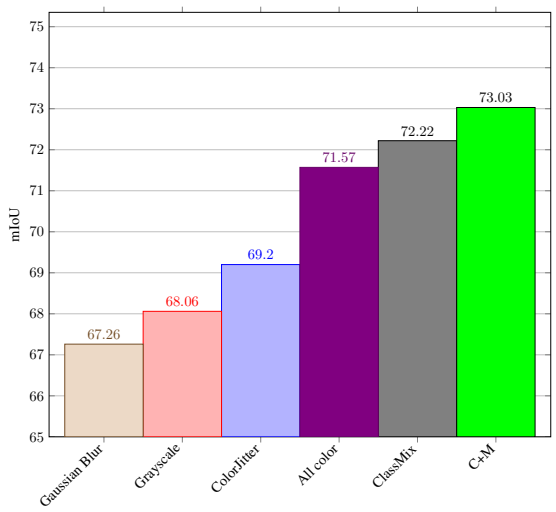


Figure 7.9: Comparison of obtained mIoU for each augmentation on the 1/16 fraction of Cityscapes. C+M stand for all colors augmentations plus ClassMix

types. Note that the combination of ClassMix and all "classic" augmentations gives the best performance.

This combination of augmentations is the one that will be used in our experiments. We can observe, that MOT_{SDA} improve the performance by a large step in all the experiments for the both datasets. This gain is even more noticeable on the smallest fractions on both datasets. Those results confirm that in NAS, as for traditional networks, strong perturbations are also important to achieve good results.

Table 7.5: mIoU of Se^{2}NAS *with different strategies obtained on the val. set of Cityscapes / Pascal VOC.*

(a) Cityscapes

	Fraction of the labeled training set			
	$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{2}$
Baseline	61.07	68.42	72.02	74.84
SSR	61.41	68.87	72.18	75.11
SSL	67.54	72.34	75.28	77.74
MOT_{SSR}	67.80	72.74	75.42	77.91
MOT_{SDA}	**73.03**	**75.33**	**76.25**	**78.02**

(b) Pascal VOC

	Fraction of the labeled training set			
	$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{2}$
Baseline	53.33	59.45	65.21	69.02
SSR	53.80	60.33	65.25	69.27
SSL	65.05	68.64	72.19	73.95
MOT_{SSR}	65.32	68.96	72.49	74.47
MOT_{SDA}	**70.82**	**71.97**	**73.40**	**75.29**

Outcome : It turns out that Se^{2}NAS, with fixed operation, is the most effective under SDA approach. Furthermore, Se^{2}NAS can obtain competitive results to more traditional and complex approaches in an automated manner, like presented in Table 7.6. Moreover, the proposed approach use up to 4 times less floating-point operations during inference stage with respect to this measure than the neural-network with the hand-crafted architecture, more precisely using 117 GFLOPs in the Cityscapes experiments with an input of size 1024x2048 and 15 GFLOPs in the Pascal VOC experiments with an input size of 512x512, compared to the 479 and 62 GFLOPs respectively used by DeepLabV3+ (estimated from corresponding setting).

Table 7.6: mIoU of Se^{2}NAS with MOT_{SDA} strategies compared to DeepLabV3+ based methods on the val. set of Cityscapes / Pascal VOC.

(a) Cityscapes

	Fraction of the labeled training set			
	$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{2}$
CPS (2021)	74.47	76.61	77.83	78.77
MOT_{SDA}	73.03	75.33	76.25	78.02

(b) Pascal VOC

	Fraction of the labeled training set			
	$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{2}$
CPS (2021)	71.98	73.67	74.90	76.15
MOT_{SDA}	70.82	71.97	73.40	75.29

Se^{2}NAS-b: Extension to full search

In the previous experiments, we have identified the most effective way to exploit the unlabeled data, which seems to be the MOT_{SDA} approach. We will now transpose this approach in an end to end manners.

In end-to-end settings, we follow the setup described in 7.4.1 for various split size. In the results presented in Table 7.7a, we can see that Se^{2}NAS-b, in an end-to-end context, can find an architecture that achieves reasonable performance compared to its fixed operations counterpart, knowing that this version was not pre-trained on ImageNet. In average, the drop of mIoU is only around 3.89 points. However, using the new cell type, the resulting network has a decrease of almost twice in terms of FLOPs compared to the original version, reducing it from 117 to an average of 69, despite the fact that some of these new blocks have larger kernel sizes. We also transpose this approaches to the Pascal VOC dataset. However, the results presented in table 7.7b are not as conclusive as those obtained on Cityscapes, and they are significantly lower than those obtained with the fixed counterpart. One possible explanation for the comparison is that the training hyperparameters were not fully discovered during the Pascal VOC training.

Table 7.7: Comparison between Se^{2}NAS & Se^{2}NAS-b using MOT_{SDA} on the val. set of Cityscapes and Pascal VOC.

(a) mIoU of Se^{2}NAS with different strategies obtained on the val. set of Cityscapes.

	Fraction of the labeled training set		FLOPs(G)
	$\frac{1}{8}$	$\frac{1}{4}$	
Se^{2}NAS	**75.33**	**76.25**	117
Se^{2}NAS-b	70.40	73.62	**69**

(b) mIoU of Se^{2}NAS with different strategies obtained on the val. set of Pascal VOC.

	Fraction of the labeled training set		FLOPs(G)
	$\frac{1}{8}$	$\frac{1}{4}$	
Se^{2}NAS	**71.97**	**73.40**	15
Se^{2}NAS-b	52.09	57.14	**9**

7.5 Summary

We suggested in this chapter to investigate a variety of techniques for using unlabeled data to perform Neural Architecture Search for semantic segmentation. On the Pascal VOC and Cityscapes datasets, we demonstrated that employing strong augmentation in conjunction with a teacher-student approach in this NAS setting may result in performance that is comparable to more sophisticated approaches based on hand-crafted networks while consuming fewer FLOPs. Based on our results, we sought to extend the dynamic routing architecture by searching for and adding new operations for each cell in an end-to-end manner. We have shown in tables 7.7 that in both cases a decrease of the number of FLOPs can be observed. This is promising, however, the mIoU performance is inferior to the fixed dynamic routing version, especially for the Pascal VOC experiments.

Conclusions and Outlook

Deep learning applications continue to expand and are not about to slow down. Neural network architectures are becoming larger and more complex in order to meet these demands. This increasing complexity necessitates an increasing amount of time and knowledge from humans, and possible cost for companies. As described in this thesis, the NAS domain would reduce this demand by adding automation. However, the NAS methods have only been studied and presented in small and well-defined domains, which are unfortunately not representative of the potential real-world applications.

In this thesis, we mainly studied the exploration and extension of the NAS approach for new tasks and a new learning context that was intended to be more representative of real world application cases.

Summary of contributions

In Chapter 5, we proposed applying a neuro-evolutionary NAS framework to the extreme multi-label classification task. We redefined a more appropriate space search for this task, combining convolution and recurrent network. We evaluate the performance of the searched network on various datasets. The resulting network worked well on the dataset that was searched, producing results that were comparable to or slightly better than the state of the art. The behavior was comparable when transferred to other datasets. In all scenarios studied, the automatically discovered network converged faster than the hand-crafted networks. In addition, we performed an impact analysis of the network operations to determine and comprehend the importance of each layer or operation in the final result.

Then, in Chapter 6, to continue on the previous line, we further investigated the NAS behavior on a new task, namely the reconstruction of RSSI map. This reconstruction task is particularly complex in real situation since it demands the measurement of the power at each position (x, y), which is in practice not feasible. Therefore, RSSI map reconstruction is usually performed with partially annotated data. To solve this task from the NAS perspective, different search algorithms to determine which one produced the best performing architecture, have been studied. Then, this architecture was trained in two steps according to a self-learning scheme. Then, the performance of the obtained architectures as well as the different training phases have been shown on several datasets and several fractions. The performances are then compared to other state-of-the-art methods, and the results obtained show well the performance of this approach.

Finally, in Chapter 7, a framework for dynamic architecture search with small amount of annotated samples for the semantic segmentation task, has been proposed. In this framework, different semi-supervised learning strategies to determine which was the most effective to leverage unlabeled samples, have been studied. We examined at a variety of methodologies, including "classic" and more current semi-supervision approaches, as well as self-supervision approaches. We also conducted a more in-depth effect analysis

of potential augmentations to determine the ideal combination. We tested these strategies on the most popular datasets in the state-of-the-art. The results obtained are competitive with those of more complex, state-of-the-art methods based on hand-made networks. Moreover, the search space of the framework allows the number of floating point operations to be reduced by 4 times. An extension of this framework is also proposed with an end-to-end search, including in addition the search of network operations. The results of this extension are encouraging, with a contained decrease in performance compared to the base version. However, more work remains to be done on this extension.

Now that we have summarized the contributions of this thesis, we will look at some avenues for future work.

Future directions

- **Introduce new operations :** Many new architectures with new blocks have emerged as a result of the research dynamics and since the completion of these works. Indeed, with the revolution brought by Transformer (Vaswani et al., 2017b) in the NLP field could be the starting point of a new search space. As an example, for the work in chapter 5, the search space could be extended with the attention module of the encoder. In (So et al., 2019) have applied evolutionary algorithm to the transformer structure and find some modification that improve the final performance. Moreover, in the last works (Chap. 7) we used variants of a block only, it would be interesting to extend this to more blocks, such as (Yu et al., 2021b), in order to obtain better performances or have a lighter network.

- **Design more flexible search spaces :** The search spaces of the current approaches are relatively restrictive and still require expert knowledge. While a search space that is carefully designed by experts clearly facilitates research, it also contradicts the idea of having a system that can be used by non-experts. This type of search space has also raised the question of whether NAS methods outperform random search. There is some evidence that the gap between random search and NAS widens when moving to less trivial spaces. (Xie et al., 2019a) show that architectures obtained by randomly generating graphs that are then mapped to a neural network architecture already give good performance. First solutions proposed with hierarchical search spaces are beginning to prove their worth, but can still be extended. For example, to be as generic as possible, the search space should consist of elementary operations and elementary functions, as studied in (Real et al., 2020). However, a larger and more diverse search space will naturally come with an increase in search costs.

- **Continue to explore new context :** As we have begun to do with these, we should continue to explore new tasks or contexts, and ideally as close as possible to real-world problems. Currently, NAS techniques are often run against a given set of hyperparameters and a specific learning pipeline, so the architecture is determined by the parameter utilized, even though alternative architectures are likely to require different hyperparameters. The selection of these hyperparameters is critical, because in some circumstances, such as the one discussed in Chapter 7, the architecture produced is only marginally efficient. Ideally, the architecture should be tuned in conjunction with all the other factors influencing the training outcome, such as hyperparameters and data augmentations. Moreover, in recent years, new research has boosted the performance of learning methods without annotated data, it could be interesting, as (Liu et al., 2020; Kaplan and Giryes, 2020) has started to do, to

apply such methods to architecture search. Another area to investigate is multitask learning. Indeed, in order to come closer to general AI, various challenges must be resolved, and to address this, a new form of architecture known as Pathways [1] has recently emerged. This type of architecture has already demonstrated its performance on numerous NLP tasks (Chowdhery et al., 2022) and might thus serve as the foundation for a new type of NAS. However, this kind of architecture requires a very large number of parameters (more than 500 billion) and therefore new adaptations in the NAS methods will surely be considered in order to scale up to that point. Finally, as previously said, there is a wide panel of prospective neural network applications and hence new sectors to be studied under the NAS aspect.

[1] https://blog.google/technology/ai/introducing-pathways-next-generation-ai-architecture/

Personnal Contributions

[1] Amini, Massih-Reza, Vasilii Feofanov, Loïc Pauletto, Emilie Devijver, and Yury Maximov (2022). "Self-Training: A Survey". In: *CoRR* abs/2202.12040. arXiv: 2202.12040 (cit. on p. 40).

[2] Malkova, Aleksandra, Loïc Pauletto, Christophe Villien, Benoît Denis, and Massih-Reza Amini (2021a). "Self-learning for Received Signal Strength Map Reconstruction with Neural Architecture Search". In: *International Conference on Artificial Neural Networks ICANN*, pp. 515–526 (cit. on p. 75).

[3] Malkova, Aleksandra, Loïc Pauletto, Christophe Villien, Benoît Denis, and Massih-Reza Amini (2021b). "Techniques d'Auto-Apprentissage et Recherche Automatique d'Architecture Neuronale pour la Reconstruction de Cartes de Puissance Radio". In: *Conférence sur l'apprentissage Automatique (CAp)* (cit. on p. 75).

[4] Pauletto, Loïc and Massih-Reza Amini (Feb. 10, 2022). "Procédé, programme d'ordinateur et dispositif d'entrainement d'un réseau neuronal convolutif à architecture dynamique pour la segmentation sémantique d'image". U.S. pat. 22305268.9 (submitted) (cit. on p. 86).

[5] Pauletto, Loïc, Massih-Reza Amini, Rohit Babbar, and Nicolas Winckler (2020a). "Neural Architecture Search for extreme multi-label classification: an evolutionary approach". In: *The Fourth International Workshop on Automation in Machine Learning (AutoML)* (cit. on p. 61).

[6] Pauletto, Loïc, Massih-Reza Amini, Rohit Babbar, and Nicolas Winckler (2020b). "Neural Architecture Search for Extreme Multi-label Text Classification". In: *27th International Conference on Neural Information Processing (ICONIP)*, pp. 282–293 (cit. on p. 61).

[7] Pauletto, Loïc, Massih-Reza Amini, and Nicolas Winckler (2022a). "Se^{2}NAS: Self-Semi-Supervised architecture optimization for Semantic Segmentation". In: *International Conference on Pattern Recognition,ICPR* (cit. on p. 86).

[8] Pauletto, Loïc, Massih-Reza Amini, and Nicolas Winckler (2022b). "The Use of Unlabeled Data for Neural Architecture Search". In: *In preparation for submission to IEEE Transactions on Neural Networks and Learning Systems* (cit. on p. 86).

Bibliography

Aernouts, M., R. Berkvens, K. Van Vlaenderen, and M. Weyn (2018). "Sigfox and LoRaWAN Datasets for Fingerprint Localization in Large Urban and Rural Areas". In: *Data* 3.2 (cit. on p. 82).

Amidi (2018). *Cheatsheet: Convolutional Neural Networks* (cit. on p. 22).

Amini, M.-R., F. Laviolette, and N. Usunier (2008). "A Transductive Bound for the Voted Classifier with an Application to Semi-supervised Learning". In: *Advances in Neural Information Processing Systems - NeurIPS*, pp. 65–72 (cit. on pp. 43, 45).

Amini, M.-R. and N. Usunier (2015). *Learning with Partially Labeled and Interdependent Data*. Springer (cit. on p. 41).

Amini, Massih R., Nicolas Usunier, and François Laviolette (2009). "A Transductive Bound for the Voted Classifier with an Application to Semi-supervised Learning". In: *Advances in Neural Information Processing Systems*, pp. 65–72 (cit. on p. 83).

Amini, Massih-Reza and Patrick Gallinari (2002). "Semi-supervised logistic regression". In: *European Conference in Artificial Intelligence - ECAI*, pp. 390–394 (cit. on p. 42).

Babbar, Rohit and Bernhard Schölkopf (2019). "Data scarcity, robustness and extreme multi-label classification". In: *Machine Learning* 108.8-9, pp. 1329–1351 (cit. on p. 63).

Babbar, Rohit and Bernhard Schölkopf (2017). "Dismec: Distributed sparse machines for extreme multi-label classification". In: *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pp. 721–729 (cit. on pp. 63, 71).

Badrinarayanan, Vijay, Alex Kendall, and Roberto Cipolla (2017). "Segnet: A deep convolutional encoder-decoder architecture for image segmentation". In: *IEEE transactions on pattern analysis and machine intelligence* 39.12, pp. 2481–2495 (cit. on p. 87).

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2015). "Neural Machine Translation by Jointly Learning to Align and Translate". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun (cit. on p. 2).

Baker, Bowen, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar (2017). "Designing Neural Network Architectures using Reinforcement Learning". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net (cit. on p. 34).

Balcan, M.-F. and A. Blum (2006). "An Augmented PAC Model for Semi-Supervised Learning". In: *Semi-Supervised Learning*, pp. 396–419 (cit. on p. 42).

Belkin, M. and P. Niyogi (2004). "Semi-supervised learning on Riemannian manifolds". In: *Machine Learning* 56.1-3, pp. 209–239 (cit. on p. 42).

Ben-David, S., T. Lu, and D. Pál (2008). "Does Unlabeled Data Provably Help? Worst-case Analysis of the Sample Complexity of Semi-Supervised Learning". In: *Conference on Learning Theory - COLT* (cit. on p. 42).

Bender, Gabriel, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc V. Le (2018). "Understanding and Simplifying One-Shot Architecture Search". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by Jennifer G. Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 549–558 (cit. on pp. 37, 38).

Bergstra, James, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl (2011). "Algorithms for Hyper-Parameter Optimization". In: *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*. Ed. by John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger, pp. 2546–2554 (cit. on p. 36).

Bergstra, James, Daniel Yamins, and David Cox (2013). "Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research. Atlanta, Georgia, USA: PMLR, pp. 115–123 (cit. on p. 36).

Bertalmio, M., A. L Bertozzi, and G. Sapiro (2001). "Navier-stokes, fluid dynamics, and image and video inpainting". In: *CVPR* (cit. on pp. 82, 83).

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag (cit. on pp. 82, 83).

Blum, A. and T. Mitchell (1998). "Combining Labeled and Unlabeled Data with Co-Training". In: *Conference on Learning Theory - COLT* (cit. on p. 46).

Buc, F. d'Alché, Y. Grandvalet, and C. Ambroise (2001). "Semi-supervised marginboost". In: *Advances in Neural Information Processing Systems - NeurIPS*, pp. 553–560 (cit. on p. 43).

Burghal, Daoud, Ashwin T. Ravi, Varun Rao, Abdullah A. Alghafis, and Andreas F. Molisch (2020). *A Comprehensive Survey of Machine Learning Based Localization with Wireless Signals*. arXiv: 2012.11171 [eess.SY] (cit. on p. 77).

Cai, Han, Jiacheng Yang, Weinan Zhang, Song Han, and Yong Yu (2018). "Path-Level Network Transformation for Efficient Architecture Search". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by Jennifer G. Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 677–686 (cit. on p. 30).

Cai, Han, Ligeng Zhu, and Song Han (2019). "ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net (cit. on p. 36).

Caron, Mathilde, Ishan Misra, Julien Mairal, et al. (2020). "Unsupervised Learning of Visual Features by Contrasting Cluster Assignments". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (cit. on pp. 55, 56).

Cascante-Bonilla, P., F. Tan, Y. Qi, and V. Ordonez (2021). "Curriculum labeling: Revisiting pseudo-labeling for semi-supervised learning". In: *AAAI Conference on Artificial Intelligence* (cit. on p. 45).

Castelli, V. and T.M. Cover (1995). "On the exponential value of labeled samples". In: *Pattern Recognit. Lett.* 16.1, pp. 105–111 (cit. on p. 42).

Chapelle, O., B. Schölkopf, and A. Zien (2010). *Semi-Supervised Learning*. 1st. The MIT Press (cit. on pp. 41, 44).

Chen, Liang-Chieh, Maxwell D. Collins, Yukun Zhu, et al. (2018a). "Searching for Efficient Multi-Scale Architectures for Dense Image Prediction". In: *NeurIPS*, pp. 8713–8724 (cit. on p. 88).

Chen, Liang-Chieh, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille (2018b). "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 40.4, pp. 834–848 (cit. on p. 24).

Chen, Liang-Chieh, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille (2015). "Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego,*

CA, USA, May 7-9, 2015, Conference Track Proceedings. Ed. by Yoshua Bengio and Yann LeCun (cit. on pp. 2, 24).

Chen, Liang-Chieh, George Papandreou, Florian Schroff, and Hartwig Adam (2017). "Rethinking Atrous Convolution for Semantic Image Segmentation". In: *CoRR* abs/1706.05587. arXiv: 1706.05587 (cit. on pp. 2, 24, 25, 88, 97).

Chen, Liang-Chieh, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam (2018c). "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation". In: *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VII*. Ed. by Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss. Vol. 11211. Lecture Notes in Computer Science. Springer, pp. 833–851 (cit. on pp. 24, 25).

Chen, Ting, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton (2020a). "A Simple Framework for Contrastive Learning of Visual Representations". In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Vol. 119. Proceedings of Machine Learning Research. PMLR, pp. 1597–1607 (cit. on pp. 53, 54, 92).

Chen, Ting, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey E. Hinton (2020b). "Big Self-Supervised Models are Strong Semi-Supervised Learners". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (cit. on pp. 41, 51, 57).

Chen, X., Y. Yuan, G. Zeng, and J. Wang (2021). "Semi-supervised semantic segmentation with cross pseudo supervision". In: *Conference on Computer Vision and Pattern Recognition - CVPR*, pp. 2613–2622 (cit. on pp. 46, 47, 90, 97, 99, 101).

Chen, Xin, Lingxi Xie, Jun Wu, and Qi Tian (2019). "Progressive DARTS: Bridging the Optimization Gap for NAS in the Wild". In: *CoRR* abs/1912.10952. arXiv: 1912.10952 (cit. on p. 31).

Chen, Xinlei, Haoqi Fan, Ross B. Girshick, and Kaiming He (2020c). "Improved Baselines with Momentum Contrastive Learning". In: *CoRR* abs/2003.04297. arXiv: 2003.04297 (cit. on pp. 55, 92, 97).

Cheng, Long, Chengdong Wu, Yunzhou Zhang, et al. (Dec. 2012). "A Survey of Localization in Wireless Sensor Network". In: *International Journal of Distributed Sensor Networks* 2012 (cit. on p. 77).

Cho, Kyunghyun, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio (2014a). "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches". In: *Proceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 25 October 2014*. Ed. by Dekai Wu, Marine Carpuat, Xavier Carreras, and Eva Maria Vecchi. Association for Computational Linguistics, pp. 103–111 (cit. on pp. 22, 65).

Cho, Kyunghyun, Bart van Merrienboer, Çaglar Gülçehre, et al. (2014b). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. Ed. by Alessandro Moschitti, Bo Pang, and Walter Daelemans. ACL, pp. 1724–1734 (cit. on p. 21).

Choi, Wongeun, Yoon-Seop Chang, Yeonuk Jung, and Junkeun Song (2018). "Low-Power LoRa Signal-Based Outdoor Positioning Using Fingerprint Algorithm". In: *ISPRS International Journal of Geo-Information* 7.11 (cit. on pp. 76, 77, 79).

Chong, Y., Y. Ding, Q. Yan, and S. Pan (2020). "Graph-based semi-supervised learning: A review". In: *Neurocomputing* 408, pp. 216–230 (cit. on p. 42).

Chopra, Sumit, Raia Hadsell, and Yann LeCun (2005). "Learning a Similarity Metric Discriminatively, with Application to Face Verification". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), 20-26 June 2005, San Diego, CA, USA*. IEEE Computer Society, pp. 539–546 (cit. on p. 53).

Chowdhery, Aakanksha, Sharan Narang, Jacob Devlin, et al. (2022). "PaLM: Scaling Language Modeling with Pathways". In: *CoRR* abs/2204.02311. arXiv: 2204.02311 (cit. on p. 106).

Chrabaszcz, Patryk, Ilya Loshchilov, and Frank Hutter (2017). "A downsampled variant of imagenet as an alternative to the cifar datasets". In: *arXiv preprint arXiv:1707.08819* (cit. on p. 37).

Cordts, Marius, Mohamed Omran, Sebastian Ramos, et al. (2016). "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: *CVPR*. IEEE Computer Society, pp. 3213–3223 (cit. on p. 96).

Courbariaux, Matthieu, Yoshua Bengio, and Jean-Pierre David (2015). "BinaryConnect: Training Deep Neural Networks with binary weights during propagations". In: *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*. Ed. by Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, pp. 3123–3131 (cit. on p. 36).

Cubuk, Ekin D., Barret Zoph, Jonathon Shlens, and Quoc V. Le (2020). "Randaugment: Practical automated data augmentation with a reduced search space". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2020, Seattle, WA, USA, June 14-19, 2020*. Computer Vision Foundation / IEEE, pp. 3008–3017 (cit. on p. 53).

Cubuk, Ekin Dogus, Barret Zoph, Dandelion Mané, Vijay Vasudevan, and Quoc V. Le (2018). "AutoAugment: Learning Augmentation Policies from Data". In: *CoRR* abs/1805.09501. arXiv: 1805.09501 (cit. on p. 53).

Cuturi, Marco (2013). "Sinkhorn distances: Lightspeed computation of optimal transport". In: *Advances in neural information processing systems* 26 (cit. on p. 57).

Dargie, W. and C. Poellabauer (2010). *Fundamentals of Wireless Sensor Networks: Theory and Practice*. John Wiley & Sons (cit. on p. 77).

Deng, Jia, Wei Dong, Richard Socher, et al. (2009). "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee, pp. 248–255 (cit. on pp. 2, 50).

Derbeko, P., R. El-Yaniv, and R. Meir (2004). "Explicit learning curves for transduction and application to clustering and compression algorithms". In: *Journal of Artificial Intelligence Research* 22, pp. 117–142 (cit. on p. 50).

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2019a). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. Association for Computational Linguistics, pp. 4171–4186 (cit. on p. 2).

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2019b). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. Association for Computational Linguistics, pp. 4171–4186 (cit. on p. 53).

Doersch, Carl, Abhinav Gupta, and Alexei A. Efros (2015). "Unsupervised Visual Representation Learning by Context Prediction". In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, pp. 1422–1430 (cit. on p. 51).

Domhan, Tobias, Jost Tobias Springenberg, and Frank Hutter (2015). "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves". In: *Twenty-fourth international joint conference on artificial intelligence* (cit. on p. 37).

Dosovitskiy, Alexey, Lucas Beyer, Alexander Kolesnikov, et al. (2021). "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net (cit. on p. 2).

Dosovitskiy, Alexey, Jost Tobias Springenberg, Martin A. Riedmiller, and Thomas Brox (2014). "Discriminative Unsupervised Feature Learning with Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. Ed. by Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, pp. 766–774 (cit. on p. 51).

Elsken, Thomas, Jan Hendrik Metzen, and Frank Hutter (2019). "Efficient Multi-Objective Neural Architecture Search via Lamarckian Evolution". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net (cit. on p. 33).

Enrico, A. and C. Redondi (2018). "Radio Map Interpolation Using Graph Signal Processing". In: *IEEE Communications Letters* 22.1, pp. 153–156 (cit. on p. 77).

Everingham, Mark, Luc Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman (2010). "The Pascal Visual Object Classes (VOC) Challenge". In: *Int. J. Comput. Vis.* 88.2, pp. 303–338 (cit. on p. 96).

Fan, Xiaochen, Xiangjian He, Chaocan Xiang, et al. (2018). "Towards System Implementation and Data Analysis for Crowdsensing Based Outdoor RSS Maps". In: *IEEE Access* 6, pp. 47535–47545.

Fei-Fei, L., R. Fergus, and Pietro Perona (2004). "Learning Generative Visual Models From Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories". In: (cit. on p. 12).

Feofanov, V., E. Devijver, and M.-R. Amini (2019). "Transductive Bounds for the Multi-Class Majority Vote Classifier". In: *AAAI Conference on Artificial Intelligence*, pp. 3566–3573 (cit. on pp. 43, 45, 46).

Ferguson, Max, Ronay ak, Yung-Tsun Lee, and Kincho Law (Dec. 2017). "Automatic localization of casting defects with convolutional neural networks". In: pp. 1726–1735 (cit. on p. 4).

Fourure, Damien, Rémi Emonet, Élisa Fromont, et al. (2017). "Residual Conv-Deconv Grid Network for Semantic Segmentation". In: *BMVC* (cit. on p. 88).

Fralick, S. C. (1967). "Learning to Recognize Patterns without a Teacher". In: *IEEE Transactions on Information Theory* 13.1, pp. 57–64 (cit. on p. 43).

Frei, S., D. Zou, Z. Chen, and Q. Gu (2022). "Self-training Converts Weak Learners to Strong Learners in Mixture Models". In: *International Conference on Artificial Intelligence and Statistics - AISTATS* (cit. on p. 48).

French, G., S. Laine, T. Aila, M. Mackiewicz, and G. D. Finlayson (2020a). "Semi-supervised semantic segmentation needs strong, varied perturbations". In: *British Machine Vision Conference - BMVC* (cit. on pp. 49, 87, 90, 92, 99).

French, Geoff, Avital Oliver, and Tim Salimans (2020b). "Milking CowMask for Semi-Supervised Image Classification". In: *CoRR* abs/2003.12022. arXiv: 2003.12022 (cit. on p. 92).

Gidaris, Spyros, Praveer Singh, and Nikos Komodakis (2018). "Unsupervised Representation Learning by Predicting Image Rotations". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net (cit. on pp. 51, 52).

Han, B., Q. Yao, X. Yu, et al. (2018). “Co-teaching: Robust training of deep neural networks with extremely noisy labels”. In: *Advances in Neural Information Processing Systems - NeurIPS* (cit. on p. 47).

Hariharan, Bharath, Pablo Arbelaez, Lubomir D. Bourdev, Subhransu Maji, and Jitendra Malik (2011). “Semantic contours from inverse detectors”. In: *ICCV*. IEEE Computer Society, pp. 991–998 (cit. on p. 96).

He, Kaiming, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross B. Girshick (2020). “Momentum Contrast for Unsupervised Visual Representation Learning”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. Computer Vision Foundation / IEEE, pp. 9726–9735 (cit. on pp. 54, 56, 91).

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, pp. 770–778 (cit. on pp. 2, 22, 23, 29, 97).

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2015). “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *ICCV*. IEEE Computer Society, pp. 1026–1034 (cit. on p. 97).

He, Xin, Kaiyong Zhao, and Xiaowen Chu (2021). “AutoML: A Survey of the State-of-the-Art”. In: *Knowledge-Based Systems* 212, p. 106622 (cit. on p. 2).

Hendrycks, Dan and Kevin Gimpel (2016). “Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units”. In: *CoRR* abs/1606.08415. arXiv: 1606.08415 (cit. on p. 18).

Hinton, Geoffrey, Li Deng, Dong Yu, et al. (2012). “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups”. In: *IEEE Signal processing magazine* 29.6, pp. 82–97 (cit. on p. 2).

Ho, Kary, Andrew Gilbert, Hailin Jin, and John Collomosse (2020). *Neural Architecture Search for Deep Image Prior*. arXiv: 2001.04776 [cs.CV] (cit. on p. 82).

Hochreiter, Sepp and Jürgen Schmidhuber (1997). “Long short-term memory”. In: *Neural computation* 9.8, pp. 1735–1780 (cit. on pp. 22, 36, 65).

Hospedales, Timothy, Antreas Antoniou, Paul Micaelli, and Amos Storkey (2020). “Meta-learning in neural networks: A survey”. In: *arXiv preprint arXiv:2004.05439* (cit. on p. 2).

Howard, Andrew G., Menglong Zhu, Bo Chen, et al. (2017). “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”. In: *CoRR* abs/1704.04861. arXiv: 1704.04861 (cit. on p. 2).

Hu, Jie, Li Shen, and Gang Sun (2018a). “Squeeze-and-excitation networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7132–7141 (cit. on p. 2).

Hu, Jie, Li Shen, and Gang Sun (2018b). “Squeeze-and-Excitation Networks”. In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. Computer Vision Foundation / IEEE Computer Society, pp. 7132–7141 (cit. on pp. 2, 29).

Hu, Shoukang, Sirui Xie, Hehui Zheng, et al. (2020). “Dsnas: Direct neural architecture search without parameter retraining”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12084–12092 (cit. on pp. 93, 97).

Huang, Gao, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger (2017). “Densely Connected Convolutional Networks”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, pp. 2261–2269 (cit. on pp. 29, 30).

Hutter, Frank, Holger H. Hoos, and Kevin Leyton-Brown (2011). “Sequential Model-Based Optimization for General Algorithm Configuration”. In: *Learning and Intelligent Optimization - 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers*.

Ed. by Carlos A. Coello Coello. Vol. 6683. Lecture Notes in Computer Science. Springer, pp. 507–523 (cit. on p. 36).

Jacovi, Alon, Oren Sar Shalom, and Yoav Goldberg (2018). "Understanding Convolutional Neural Networks for Text Classification". In: *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Association for Computational Linguistics, pp. 56–65 (cit. on p. 22).

Jain, Himanshu, Yashoteja Prabhu, and Manik Varma (2016). "Extreme multi-label loss functions for recommendation, tagging, ranking & other missing label applications". In: *Proceedings of the 22nd ACM SIGKDD ICKDD*, pp. 935–944 (cit. on pp. 63, 71).

Jang, Eric, Shixiang Gu, and Ben Poole (2017). "Categorical Reparameterization with Gumbel-Softmax". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net (cit. on p. 36).

Joachims, T. (1999). "Transductive Inference for Text Classification Using Support Vector Machines". In: *International Conference on Machine Learning - ICML*, pp. 200–209 (cit. on pp. 42, 50).

Kalantidis, Yannis, Mert Bülent Sariyildiz, Noé Pion, Philippe Weinzaepfel, and Diane Larlus (2020). "Hard Negative Mixing for Contrastive Learning". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (cit. on p. 53).

Kandasamy, Kirthevasan, Willie Neiswanger, Jeff Schneider, Barnabás Póczos, and Eric P. Xing (2018). "Neural Architecture Search with Bayesian Optimisation and Optimal Transport". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, et al., pp. 2020–2029 (cit. on p. 36).

Kaplan, Sapir and Raja Giryes (2020). "Self-supervised Neural Architecture Search". In: *CoRR* abs/2007.01500. arXiv: 2007.01500 (cit. on pp. 5, 105).

Karamanolakis, G., S. Mukherjee, G. Zheng, and A. Awadallah (2021). "Self-training with Weak Supervision". In: *North American Conference on Chinese Linguistics - NAACL* (cit. on pp. 46, 47).

Ke, Zhanghan, Daoye Wang, Qiong Yan, Jimmy S. J. Ren, and Rynson W. H. Lau (2019). "Dual Student: Breaking the Limits of the Teacher in Semi-Supervised Learning". In: *ICCV*. IEEE, pp. 6727–6735 (cit. on p. 87).

Khan, Salman, Hossein Rahmani, Syed Afaq Ali Shah, et al. (2018) (cit. on p. 22).

Khandagale, Sujay, Han Xiao, and Rohit Babbar (2020). "Bonsai: diverse and shallow trees for extreme multi-label classification". In: *Machine Learning*, pp. 1–21 (cit. on p. 63).

Khelifi, Fekher, Abbas Bradai, Abderrahim Benslimane, Priyanka Rawat, and Mohamed Atri (2019). "A survey of localization systems in internet of things". In: *Mobile Networks and Applications* 24.3, pp. 761–785 (cit. on p. 76).

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (cit. on pp. 70, 97).

Kingma, Diederik P. and Max Welling (2014). "Auto-Encoding Variational Bayes". In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun (cit. on p. 37).

Kingma, D.P., S. Mohamed, D. Jimenez, and M. Welling (2014). "Semi-supervised Learning with Deep Generative Models". In: *Advances in Neural Information Processing Systems - NeurIPS* (cit. on p. 41).

Klein, Aaron, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter (2017). "Fast bayesian optimization of machine learning hyperparameters on large datasets". In: *Artificial intelligence and statistics*. PMLR, pp. 528–536 (cit. on p. 37).

Krithara, Anastasia, Massih-Reza Amini, Jean-Michel Renders, and Cyril Goutte (2008). "Semi-Supervised Document Classification with a Mislabeling Error Model". In: *30th European Conference on Information Retrieval*. Glasgow, pp. 370–381 (cit. on p. 83).

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*, pp. 1106–1114 (cit. on pp. 2, 13).

Kubota, R., S. Tagashira, Y. Arakawa, T. Kitasuka, and A. Fukuda (2013). "Efficient Survey Database Construction Using Location Fingerprinting Interpolation". In: *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, pp. 469–476 (cit. on p. 78).

Laaraiedh, M., B. Uguen, J. Stephan, et al. (2012). "Ray tracing-based radio propagation modeling for indoor localization purposes". In: *2012 IEEE 17th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pp. 276–280 (cit. on p. 78).

Laine, S. and T. Aila (2017). "Temporal Ensembling for Semi-Supervised Learning". In: *International Conference on Learning Representations - ICLR* (cit. on p. 49).

LeCun, Yann, Yoshua Bengio, et al. (1995). "Convolutional networks for images, speech, and time series". In: *The handbook of brain theory and neural networks* 3361.10, p. 1995 (cit. on p. 22).

LeCun, Yann, Corinna Cortes, and CJ Burges (2010). "MNIST handwritten digit database". In: *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* 2 (cit. on p. 62).

Lee, D.-H. (2013). "Pseudo-Label : The Simple and Efficient Semi-Supervised Learning Method for Deep Neural Networks". In: *ICML 2013 Workshop on Challenges in Representation Learning* (cit. on p. 45).

Levie, R., Ç. Yapar, G. Kutyniok, and G. Caire (2020). "Pathloss Prediction using Deep Learning with Applications to Cellular Optimization and Efficient D2D Link Scheduling". In: *ICASSP*, pp. 8678–8682 (cit. on p. 78).

Li, Jin and Andrew D.Heap (2011). "A review of comparative studies of spatial interpolation methods in environmental sciences: Performance and impact factors". In: *Ecological Informatics* 6.3, pp. 228 –241 (cit. on p. 77).

Li, Y.-F. and Z.-H. Zhou (2011). "Towards Making Unlabeled Data Never Hurt". In: *Proceedings of the 28th International Conference on Machine Learning*, pp. 1081–1088 (cit. on p. 42).

Li, Yanwei, Lin Song, Yukang Chen, et al. (2020). "Learning Dynamic Routing for Semantic Segmentation". In: *CVPR*. Computer Vision Foundation / IEEE, pp. 8550–8559 (cit. on pp. 32, 80, 94, 97).

Liao, Jianxin, Qi Qi, Haifeng Sun, and Jingyu Wang (Feb. 2019). "Radio Environment Map Construction by Kriging Algorithm Based on Mobile Crowd Sensing". In: *Wireless Communications and Mobile Computing* 2019, pp. 1–12 (cit. on p. 77).

Lin, Zhouhan, Minwei Feng, Cicero Nogueira dos Santos, et al. (2017). "A structured self-attentive sentence embedding". In: *arXiv preprint arXiv:1703.03130* (cit. on pp. 64, 66).

Liu, Chenxi, Liang-Chieh Chen, Florian Schroff, et al. (2019a). "Auto-DeepLab: Hierarchical Neural Architecture Search for Semantic Image Segmentation". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, pp. 82–92 (cit. on pp. 31, 32, 93).

Liu, Chenxi, Piotr Dollár, Kaiming He, et al. (2020). "Are Labels Necessary for Neural Architecture Search?" In: *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part IV*. Ed. by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Vol. 12349. Lecture Notes in Computer Science. Springer, pp. 798–813 (cit. on pp. 5, 38, 89, 105).

Liu, Chenxi, Barret Zoph, Maxim Neumann, et al. (2018). "Progressive neural architecture search". In: *Proceedings of the European conference on computer vision (ECCV)*, pp. 19–34 (cit. on pp. 36, 37).

Liu, Hanxiao, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu (2017a). "Hierarchical Representations for Efficient Architecture Search". In: *CoRR* abs/1711.00436. arXiv: 1711.00436 (cit. on pp. 32, 33).

Liu, Hanxiao, Karen Simonyan, and Yiming Yang (2019b). "DARTS: Differentiable Architecture Search". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net (cit. on pp. 5, 35).

Liu, Jingzhou, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang (2017b). "Deep learning for extreme multi-label text classification". In: *Proceedings of the 40th International ACM SIGIR*, pp. 115–124 (cit. on pp. 63, 64, 70, 71).

Liu, Xiaodong, Jianfeng Gao, Xiaodong He, et al. (2015). "Representation learning using multi-task deep neural networks for semantic classification and information retrieval". In: (cit. on p. 62).

Liu, Xiaolong, Zhidong Deng, and Yuhan Yang (2019c). "Recent progress in semantic image segmentation". In: *Artif. Intell. Rev.* 52.2, pp. 1089–1106 (cit. on p. 87).

Liu, Zhuang, Hanzi Mao, Chao-Yuan Wu, et al. (2022). "A ConvNet for the 2020s". In: *arXiv preprint arXiv:2201.03545* (cit. on p. 2).

Long, Jonathan, Evan Shelhamer, and Trevor Darrell (2015). "Fully convolutional networks for semantic segmentation". In: *CVPR*. IEEE Computer Society, pp. 3431–3440 (cit. on p. 88).

Luo, Renqian, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu (2018). "Neural architecture optimization". In: *Advances in neural information processing systems* 31 (cit. on p. 36).

Ma, Ningning, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun (2018). "Shufflenet v2: Practical guidelines for efficient cnn architecture design". In: *Proceedings of the European conference on computer vision (ECCV)*, pp. 116–131 (cit. on p. 93).

Maddison, Chris J., Andriy Mnih, and Yee Whye Teh (2017). "The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net (cit. on pp. 36, 37).

Marcus, Mitchell P., Grace Kim, Mary Ann Marcinkiewicz, et al. (1994). "The Penn Treebank: Annotating Predicate Argument Structure". In: *Human Language Technology, Proceedings of a Workshop held at Plainsboro, New Jerey, USA, March 8-11, 1994*. Morgan Kaufmann (cit. on p. 34).

Maximov, Y., M.-R. Amini, and Z. Harchaoui (2018). "Rademacher Complexity Bounds for a Penalized Multi-class Semi-supervised Algorithm". In: *Journal of Artificial Intelligence Research* 61, pp. 761–786 (cit. on pp. 42, 49).

McAuley, Julian J. and Jure Leskovec (2013). "Hidden factors and hidden topics: understanding rating dimensions with review text". In: *Seventh ACM Conference on Recommender Systems, RecSys '13, Hong Kong, China, October 12-16, 2013*. Ed. by Qiang Yang, Irwin King, Qing Li, Pearl Pu, and George Karypis. ACM, pp. 165–172 (cit. on p. 70).

Mcculloch, Warren and Walter Pitts (1943). "A Logical Calculus of Ideas Immanent in Nervous Activity". In: *Bulletin of Mathematical Biophysics* 5, pp. 127–147 (cit. on pp. 12, 17).

Mellor, Joe, Jack Turner, Amos Storkey, and Elliot J Crowley (2021). "Neural architecture search without training". In: *International Conference on Machine Learning*. PMLR, pp. 7588–7598 (cit. on p. 37).

Mencía, Eneldo Loza and Johannes Fürnkranz (2008). "Efficient Pairwise Multilabel Classification for Large-Scale Problems in the Legal Domain". In: *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML/PKDD 2008, Antwerp, Belgium, September 15-19, 2008, Proceedings, Part II*. Ed. by Walter Daelemans, Bart Goethals, and Katharina Morik. Vol. 5212. Lecture Notes in Computer Science. Springer, pp. 50–65 (cit. on p. 70).

Miikkulainen, Risto, Jason Liang, Elliot Meyerson, et al. (2019). "Evolving deep neural networks". In: *Artificial intelligence in the age of neural networks and brain computing*. Elsevier, pp. 293–312 (cit. on p. 33).

Miller, Geoffrey F., Peter M. Todd, and Shailesh U. Hegde (1989). "Designing Neural Networks using Genetic Algorithms". In: *Proceedings of the 3rd International Conference on Genetic Algorithms, George Mason University, Fairfax, Virginia, USA, June 1989*. Ed. by J. David Schaffer. Morgan Kaufmann, pp. 379–384 (cit. on p. 33).

Minsky, Marvin and Seymour Papert (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press (cit. on p. 12).

Miyato, T., S. Maeda, M. Koyama, and S. Ishii (2019). "Virtual Adversarial Training: A Regularization Method for Supervised and Semi-Supervised Learning". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 41.8, pp. 1979–1993 (cit. on p. 47).

Mottaghi, Roozbeh, Xianjie Chen, Xiaobai Liu, et al. (2014). "The Role of Context for Object Detection and Semantic Segmentation in the Wild". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 891–898 (cit. on p. 87).

Müller, David, Andreas Ehlen, and Bernd Valeske (2021). "Convolutional neural networks for semantic segmentation as a tool for multiclass face analysis in thermal infrared". In: *Journal of nondestructive evaluation* 40.1, pp. 1–10 (cit. on p. 87).

Nam, Jinseok, Eneldo Loza Mencía, Hyunwoo J Kim, and Johannes Fürnkranz (2017). "Maximizing subset accuracy with recurrent neural networks in multi-label classification". In: *Advances in neural information processing systems*, pp. 5413–5423 (cit. on pp. 63, 64, 71).

Natarajan, N., I. S Dhillon, P. K. Ravikumar, and A. Tewari (2013). "Learning with noisy labels". In: *Advances in Neural Information Processing Systems*, pp. 1196–1204 (cit. on p. 47).

Negrinho, Renato, Matthew R. Gormley, Geoffrey J. Gordon, et al. (2019). "Towards modular and programmable architecture search". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, et al., pp. 13715–13725 (cit. on p. 36).

Nekrasov, Vladimir, Hao Chen, Chunhua Shen, and Ian D. Reid (2019a). "Fast Neural Architecture Search of Compact Semantic Segmentation Models via Auxiliary Cells". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, pp. 9126–9135 (cit. on p. 38).

Nekrasov, Vladimir, Hao Chen, Chunhua Shen, and Ian D. Reid (2019b). "Fast Neural Architecture Search of Compact Semantic Segmentation Models via Auxiliary Cells". In: *CVPR*. Computer Vision Foundation / IEEE, pp. 9126–9135 (cit. on p. 88).

Nigam, K., A. McCallum, and T. Mitchell (2006). "Semi-Supervised Text Classification Using EM." In: *Semi-Supervised Learning*. The MIT Press, pp. 32–55 (cit. on p. 41).

Ning, C. and et al. (2016). "Outdoor Location Estimation Using Received Signal Strength-Based Fingerprinting". In: *Wireless Pers Commun* 99, 365–384 (cit. on p. 78).

Niyogi, P. (2013). "Manifold Regularization and Semi-supervised Learning: Some Theoretical Analyses". In: *Journal of Machine Learning Research* (cit. on p. 42).

Noroozi, Mehdi and Paolo Favaro (2016). "Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles". In: *ECCV*. Vol. 9910. Lecture Notes in Computer Science. Springer, pp. 69–84 (cit. on pp. 51, 52, 89).

Novikoff, A. B. (1962). "On Convergence Proofs on Perceptrons". In: *Proceedings of the Symposium on the Mathematical Theory of Automata*, pp. 615–622 (cit. on p. 12).

Oliver, A., A. Odena, C. Raffel, E.-D. Cubuk, and I. Goodfellow (2018). "Realistic Evaluation of Deep Semi-Supervised Learning Algorithms". In: *Advances in Neural Information Processing Systems - NeurIPS* (cit. on p. 42).

Oliver, M.A. and R. Webster (1990). “Kriging: a method of interpolation for geographical information systems”. In: *International Journal of Geographical Information System* 4.3, pp. 313–332 (cit. on pp. 82, 83).

Olsson, Viktor, Wilhelm Tranheden, Juliano Pinto, and Lennart Svensson (2021). “ClassMix: Segmentation-Based Data Augmentation for Semi-Supervised Learning”. In: *IEEE Winter Conference on Applications of Computer Vision, WACV 2021, Waikoloa, HI, USA, January 3-8, 2021*. IEEE, pp. 1368–1377 (cit. on pp. 49, 92, 99).

Oord, Aaron van den, Yazhe Li, and Oriol Vinyals (2018). “Representation learning with contrastive predictive coding”. In: *arXiv preprint arXiv:1807.03748* (cit. on p. 55).

Ouali, Yassine, Céline Hudelot, and Myriam Tami (2020). “Semi-Supervised Semantic Segmentation With Cross-Consistency Training”. In: *CVPR*. Computer Vision Foundation / IEEE, pp. 12671–12681 (cit. on p. 87).

Partalas, Ioannis, Aris Kosmopoulos, Nicolas Baskiotis, et al. (2015). “LSHTC: A Benchmark for Large-Scale Text Classification”. In: *CoRR* abs/1503.08581. arXiv: 1503.08581 (cit. on p. 63).

Pascanu, Razvan, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio (2014). “How to Construct Deep Recurrent Neural Networks”. In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun (cit. on p. 21).

Pathak, Deepak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros (2016). “Context encoders: Feature learning by inpainting”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2536–2544 (cit. on pp. 52, 53).

Patrini, G., A. Rozza, A. K. Menon, R. Nock, and L. Qu (2017). “Making Deep Neural Networks Robust to Label Noise: A Loss Correction Approach”. In: *Conference on Computer Vision and Pattern Recognition - CVPR*, pp. 2233–2241 (cit. on p. 47).

Pennington, Jeffrey, Richard Socher, and Christopher D Manning (2014). “Glove: Global vectors for word representation”. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543 (cit. on p. 66).

Pham, Hieu, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean (2018). “Efficient Neural Architecture Search via Parameter Sharing”. In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by Jennifer G. Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 4092–4101 (cit. on pp. 5, 34, 37).

Prabhu, Yashoteja, Anil Kag, Shrutendra Harsola, Rahul Agrawal, and Manik Varma (2018). “Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising”. In: *Proceedings of the 2018 World Wide Web Conference*, pp. 993–1002 (cit. on p. 71).

Ramachandran, Prajit, Barret Zoph, and Quoc V. Le (2018). “Searching for Activation Functions”. In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net (cit. on p. 18).

Rasmussen, Carl Edward (2003). “Gaussian Processes in Machine Learning”. In: *Advanced Lectures on Machine Learning, ML Summer Schools 2003, Canberra, Australia, February 2-14, 2003, Tübingen, Germany, August 4-16, 2003, Revised Lectures*. Ed. by Olivier Bousquet, Ulrike von Luxburg, and Gunnar Rätsch. Vol. 3176. Lecture Notes in Computer Science. Springer, pp. 63–71 (cit. on p. 36).

Raspopoulos, M., C. Laoudias, L. Kanaris, et al. (2012). “3D Ray Tracing for device-independent fingerprint-based positioning in WLANs”. In: *2012 9th Workshop on Positioning, Navigation and Communication*, pp. 109–113 (cit. on p. 78).

Rawal, Aditya and Risto Miikkulainen (2018). “From Nodes to Networks: Evolving Recurrent Neural Networks”. In: *CoRR* abs/1803.04439. arXiv: 1803.04439 (cit. on p. 38).

Real, Esteban, Alok Aggarwal, Yanping Huang, and Quoc V Le (2019). "Regularized evolution for image classifier architecture search". In: *Proceedings of the aaai conference on artificial intelligence*. Vol. 33. 01, pp. 4780–4789 (cit. on pp. 33, 36, 37, 65, 73, 80).

Real, Esteban, Chen Liang, David R. So, and Quoc V. Le (2020). "AutoML-Zero: Evolving Machine Learning Algorithms From Scratch". In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Vol. 119. Proceedings of Machine Learning Research. PMLR, pp. 8007–8019 (cit. on p. 105).

Real, Esteban, Sherry Moore, Andrew Selle, et al. (2017). "Large-scale evolution of image classifiers". In: *International Conference on Machine Learning*. PMLR, pp. 2902–2911 (cit. on pp. 33, 34).

Redondi, A. E. C. (2018). "Radio Map Interpolation Using Graph Signal Processing". In: *IEEE Communications Letters* 22.1, pp. 153–156 (cit. on p. 77).

Rigollet, P. (2007). "Generalization Error Bounds in Semi-supervised Classification Under the Cluster Assumption". In: *Journal of Machine Learning Research* 8, pp. 1369–1392 (cit. on p. 42).

Ronneberger, O., P.Fischer, and T. Brox (2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Vol. 9351. LNCS. Springer, pp. 234–241 (cit. on pp. 2, 78, 87, 88).

Rosenblatt, Frank (1958). "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain". In: *Psychological Review* 65.6 (cit. on p. 12).

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1986). "Learning Representations by Back-propagating Errors". In: *Nature* 323.6088, pp. 533–536 (cit. on pp. 12, 19).

Russakovsky, Olga, Jia Deng, Hao Su, et al. (2015). "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3, pp. 211–252 (cit. on p. 13).

Rusu, V. and C. Rusu (Oct. 2006). "Radial Basis Functions Versus Geostatistics in Spatial Interpolations". In: vol. 217 (cit. on p. 77).

Saito, K., Y. Ushiku, and T. Harada (2017). "Asymmetric tri-training for unsupervised domain adaptation". In: *International Conference on Machine Learning*. PMLR, pp. 2988–2997 (cit. on p. 57).

Sato, K., K. Inage, and T. Fujii (2019). "On the Performance of Neural Network Residual Kriging in Radio Environment Mapping". In: *IEEE Access* 7, pp. 94557–94568 (cit. on p. 78).

Schapire, R.E., Y. Freund, P. Barlett, and W. S. Lee (1997). "Boosting the margin: A new explanation for the effectiveness of voting methods". In: *International Conference on Machine Learning - ICML*, pp. 322–330 (cit. on p. 45).

Schneider (2021). *How Deep Learning Works - Inside the neural networks that power today's AI* (cit. on p. 20).

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (2017). "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (cit. on p. 34).

Scudder, H. (1965). "Adaptive communication receivers". In: *IEEE Transactions on Information Theory* 11.2, pp. 167–174 (cit. on p. 43).

Shi, Tian, Liuqing Li, Ping Wang, and Chandan K. Reddy (2021). "A Simple and Effective Self-Supervised Contrastive Learning Framework for Aspect Detection". In: *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, pp. 13815–13824 (cit. on p. 53).

Shi, W., Y. Gong, C. Ding, et al. (2018). "Transductive Semi-Supervised Deep Learning using Min-Max Features". In: *European Conference on Computer Vision - ECCV* (cit. on p. 50).

Silver, David, Aja Huang, Chris J. Maddison, et al. (2016). "Mastering the game of Go with deep neural networks and tree search". In: *Nat.* 529.7587, pp. 484–489 (cit. on p. 2).

Simonyan, Karen and Andrew Zisserman (2015). "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun (cit. on p. 2).

Singh, A., R. Nowak, and J. Zhu (2009). "Unlabeled data: Now it helps, now it doesn't". In: *Advances in Neural Information Processing Systems - NeurIPS* (cit. on p. 42).

So, David R., Quoc V. Le, and Chen Liang (2019). "The Evolved Transformer". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 5877–5886 (cit. on p. 105).

Sohn, K., D. Berthelot, N. Carlini, et al. (2020). "Fixmatch: Simplifying semi-supervised learning with consistency and confidence". In: *Advances in Neural Information Processing Systems* 33, pp. 596–608 (cit. on pp. 47, 91, 92).

Sorour, S., Y. Lostanlen, S. Valaee, and K. Majeed (2015). "Joint Indoor Localization and Radio Map Construction with Limited Deployment Load". In: *IEEE Transactions on Mobile Computing* 14.5, pp. 1031–1043 (cit. on p. 78).

Su, Weijie, Xizhou Zhu, Yue Cao, et al. (2020). "VL-BERT: Pre-training of Generic Visual-Linguistic Representations". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net (cit. on p. 53).

Suganuma, Masanori, Shinichi Shirakawa, and Tomoharu Nagao (2017). "A genetic programming approach to designing convolutional neural network architectures". In: *Proceedings of the genetic and evolutionary computation conference*, pp. 497–504 (cit. on p. 33).

Szegedy, Christian, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi (2017). "Inception-v4, inception-resnet and the impact of residual connections on learning". In: *Thirty-first AAAI conference on artificial intelligence* (cit. on p. 4).

Szegedy, Christian, Wei Liu, Yangqing Jia, et al. (2015). "Going deeper with convolutions". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, pp. 1–9 (cit. on p. 2).

Tagami, Yukihiro (2017). "Annexml: Approximate nearest neighbor search for extreme multi-label classification". In: *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 455–464 (cit. on pp. 63, 71).

Tahat, A., G. Kaddoum, S. Yousefi, S. Valaee, and F. Gagnon (2016). "A Look at the Recent Wireless Positioning Techniques With a Focus on Algorithms for Moving Receivers". In: *IEEE Access* 4, pp. 6652–6680 (cit. on p. 77).

Tan, Mingxing and Quoc V. Le (2019). "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 6105–6114 (cit. on p. 38).

Tarvainen, A. and H. Valpola (2017). "Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results". In: *Advances in Neural Information Processing Systems - NeurIPS*, pp. 1195–1204 (cit. on pp. 49, 90).

Thrun, Sebastian and Lorien Pratt (2012). *Learning to learn*. Springer Science & Business Media (cit. on p. 2).

Tür, G., D. Tür, and R.-E. Schapire (2005). "Combining active and semi-supervised learning for spoken language understanding". In: *Speech Communication* 45, pp. 171–186 (cit. on pp. 44, 45).

Ulyanov, Dmitry, Andrea Vedaldi, and Victor Lempitsky (2017). "Deep Image Prior". In: *CoRR* abs/1711.10925. arXiv: 1711.10925 (cit. on pp. 76, 80, 82, 83).

Valiant, L.G. (1984). "A theory of the learnable". In: *Communications of the ACM* 27.11, pp. 1134–1142 (cit. on p. 46).

Vapnik, V. (1998). *Statistical Learning Theory*. Wiley-Interscience (cit. on pp. 14, 15, 50).

Vaswani, Ashish, Noam Shazeer, Niki Parmar, et al. (2017b). "Attention is All you Need". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, et al., pp. 5998–6008 (cit. on p. 105).

Vaswani, Ashish, Noam Shazeer, Niki Parmar, et al. (2017a). "Attention is all you need". In: *Advances in neural information processing systems* 30 (cit. on p. 2).

Vo, Q. D. and P. De (2016). "A Survey of Fingerprint-Based Outdoor Localization". In: *IEEE Communications Surveys & Tutorials* 18.1, pp. 491–506 (cit. on p. 76).

Wei, C., K. Shen, Y. Chen, and T. Ma (2021). "Theoretical Analysis of Self-Training with Deep Networks on Unlabeled Data". In: *International Conference on Learning Representations - ICLR* (cit. on p. 47).

Wilcoxon, F. (1945). "Individual Comparisons by Ranking Methods". In: *Biometrics* 1.6, pp. 80–83 (cit. on p. 82).

Wong, Catherine, Neil Houlsby, Yifeng Lu, and Andrea Gesmundo (2018). "Transfer Learning with Neural AutoML". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, et al., pp. 8366–8375 (cit. on p. 38).

Wu, Xifang, Songlin Sun, and Meixia Fu (2020). "Person Re-identification Based on Semantic Segmentation". In: *Signal and Information Processing, Networking and Computers*. Ed. by Yue Wang, Meixia Fu, Lexi Xu, and Jiaqi Zou (cit. on p. 87).

Wu, Zhirong, Yuanjun Xiong, Stella X. Yu, and Dahua Lin (2018). "Unsupervised Feature Learning via Non-Parametric Instance-level Discrimination". In: *CoRR* abs/1805.01978. arXiv: 1805.01978 (cit. on p. 56).

Xie, Lingxi and Alan L. Yuille (2017). "Genetic CNN". In: *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*. IEEE Computer Society, pp. 1388–1397 (cit. on p. 33).

Xie, Q., Z. Dai, E. H. Hovy, T. Luong, and Q. Le (2020a). "Unsupervised Data Augmentation for Consistency Training". In: *Advances in Neural Information Processing Systems - NeurIPS* (cit. on pp. 49, 91).

Xie, Q., M.-T. Luong, E. H. Hovy, and Q. V. Le (2020b). "Self-Training With Noisy Student Improves ImageNet Classification". In: *Conference on Computer Vision and Pattern Recognition - CVPR*, pp. 10684–10695 (cit. on pp. 46, 47).

Xie, Saining, Alexander Kirillov, Ross B. Girshick, and Kaiming He (2019a). "Exploring Randomly Wired Neural Networks for Image Recognition". In: *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*. IEEE, pp. 1284–1293 (cit. on p. 105).

Xie, Sirui, Hehui Zheng, Chunxiao Liu, and Liang Lin (2019b). "SNAS: stochastic neural architecture search". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net (cit. on pp. 36, 37).

Yang, Pengcheng, Xu Sun, Wei Li, et al. (2018). "SGM: sequence generation model for multi-label classification". In: *arXiv preprint arXiv:1806.04822* (cit. on p. 64).

You, Ronghui, Suyang Dai, Zihan Zhang, Hiroshi Mamitsuka, and Shanfeng Zhu (2018). "Attentionxml: Extreme multi-label text classification with multi-label attention based recurrent neural networks". In: *arXiv preprint arXiv:1811.01727* (cit. on pp. 64, 66, 70, 71, 73).

Yu, Changqian, Jingbo Wang, Chao Peng, et al. (2018). "BiSeNet: Bilateral Segmentation Network for Real-Time Semantic Segmentation". In: *ECCV*. Vol. 11217. Lecture Notes in Computer Science. Springer, pp. 334–349 (cit. on p. 88).

Yu, Changqian, Bin Xiao, Changxin Gao, et al. (2021a). "Lite-hrnet: A lightweight high-resolution network". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10440–10450 (cit. on p. 93).

Yu, Kaicheng, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann (2019). "Evaluating the search phase of neural architecture search". In: *arXiv preprint arXiv:1902.08142* (cit. on p. 37).

Yu, Kegen, Ian Sharp, and Y. Guo (June 2009). *Ground-Based Wireless Positioning*. John Wiley & Sons, Ltd (cit. on p. 77).

Yu, Tong and Hong Zhu (2020). "Hyper-parameter optimization: A review of algorithms and applications". In: *arXiv preprint arXiv:2003.05689* (cit. on p. 2).

Yu, Weihao, Mi Luo, Pan Zhou, et al. (2021b). "MetaFormer is Actually What You Need for Vision". In: *CoRR* abs/2111.11418. arXiv: 2111.11418 (cit. on pp. 2, 105).

Zela, Arber, Aaron Klein, Stefan Falkner, and Frank Hutter (2018). "Towards automated deep learning: Efficient joint neural architecture and hyperparameter search". In: *arXiv preprint arXiv:1807.06906* (cit. on p. 37).

Zhai, Xiaohua, Avital Oliver, Alexander Kolesnikov, and Lucas Beyer (2019). "S4l: Self-supervised semi-supervised learning". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1476–1485 (cit. on p. 89).

Zhang, B., Y. Wang, W. Hou, et al. (2021). "Flexmatch: Boosting semi-supervised learning with curriculum pseudo labeling". In: *Advances in Neural Information Processing Systems* 34 (cit. on p. 45).

Zhang, Hongyi, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz (2018a). "mixup: Beyond Empirical Risk Minimization". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net (cit. on p. 53).

Zhang, Richard, Phillip Isola, and Alexei A Efros (2016). "Colorful image colorization". In: *European conference on computer vision*. Springer, pp. 649–666 (cit. on p. 52).

Zhang, S., M. Wang, S. Liu, P.-Y. Chen, and J. Xiong (2022). "How unlabeled data improve generalization in self-training? A one-hidden-layer theoretical analysis". In: *International Conference on Learning Representations - ICLR* (cit. on p. 48).

Zhang, Xiang, Junbo Zhao, and Yann LeCun (2015). "Character-level convolutional networks for text classification". In: *Advances in neural information processing systems*, pp. 649–657 (cit. on p. 62).

Zhang, Xiangyu, Xinyu Zhou, Mengxiao Lin, and Jian Sun (2018b). "Shufflenet: An extremely efficient convolutional neural network for mobile devices". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6848–6856 (cit. on p. 2).

Zhang, Youshan and Brian D. Davison (2020). "Impact of ImageNet Model Selection on Domain Adaptation". In: *IEEE Winter Applications of Computer Vision Workshops, WACV Workshops 2020, Snowmass Village, CO, USA, March 1-5, 2020*. IEEE, pp. 173–182 (cit. on p. 3).

Zhang, Yu, William Chan, and Navdeep Jaitly (2017). "Very deep convolutional networks for end-to-end speech recognition". In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2017, New Orleans, LA, USA, March 5-9, 2017*. IEEE, pp. 4845–4849 (cit. on p. 2).

Zhong, K., Z. Song, P. Jain, P. L. Bartlett, and I. S. Dhillon (2017). "Recovery Guarantees for One-hidden-layer Neural Networks". In: *International Conference on Machine Learning - ICML*, pp. 4140–4149 (cit. on p. 48).

Zhong, Zhao, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu (2018). "Practical Block-Wise Neural Network Architecture Generation". In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. Computer Vision Foundation / IEEE Computer Society, pp. 2423–2432 (cit. on p. 29).

Zhou, Chunting, Chonglin Sun, Zhiyuan Liu, and Francis Lau (2015). "A C-LSTM neural network for text classification". In: *arXiv preprint arXiv:1511.08630* (cit. on p. 62).

Zhu, Di, Ximeng Cheng, Fan Zhang, et al. (2020). "Spatial interpolation using conditional generative adversarial neural networks". In: *International Journal of Geographical Information Science* 34.4, pp. 735–758 (cit. on p. 78).

Zhu, X., Z. Ghahramani, and J.D. Lafferty (2003). "Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions". In: *International Conference on Machine Learning - ICML*, pp. 912–919 (cit. on p. 48).

Zoph, Barret and Quoc V. Le (2017). "Neural Architecture Search with Reinforcement Learning". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net (cit. on p. 34).

Zoph, Barret, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le (2018). "Learning Transferable Architectures for Scalable Image Recognition". In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. Computer Vision Foundation / IEEE Computer Society, pp. 8697–8710 (cit. on pp. 3, 5, 29, 30, 36, 37).

Zou, Y., Z. Yu, B. Kumar, and J. Wang (2018). "Unsupervised domain adaptation for semantic segmentation via class-balanced self-training". In: *European conference on computer vision - ECCV*, pp. 289–305 (cit. on pp. 45, 57).

Zou, Y., Z. Yu, X. Liu, B. Kumar, and J. Wang (2019). "Confidence regularized self-training". In: *International Conference on Computer Vision - ICCV*, pp. 5982–5991 (cit. on p. 57).

Zou, Yuliang, Zizhao Zhang, Han Zhang, et al. (2021). "PseudoSeg: Designing Pseudo Labels for Semantic Segmentation". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net (cit. on p. 97).

Zoutendijk, G. (1966). "Nonlinear programming: a numerical survey". In: *SIAM Journal on Control* 4.1, pp. 194–210 (cit. on p. 17).

Zubiaga, Arkaitz (2012). "Enhancing Navigation on Wikipedia with Social Tags". In: *CoRR* abs/1202.5469. arXiv: 1202.5469 (cit. on p. 70).

List of Figures

List of Tables