University of Bari “Aldo Moro”
Department of Computer Science

A thesis submitted in partial fulfillment of the requirements for the Degree of Bachelor of Science in

Computer Science and Technologies for Software Production

Insert Your Title Here

Filippo Chinni Carella

Supervisor:
Prof. Corrado Mencar

May 15, 2024

Acknowledgement

Do not forget to acknowledge the supervisor ☺

PLEASE, FOLLOW THE GUIDELINES GIVEN IN THIS TEMPLATE IN EACH CHAPTER, ESPECIALLY IN CHAPTER 2!

Abstract

An *abstract* is a summary of a research article, thesis, review, conference proceeding, or any in-depth analysis of a particular subject or discipline. It is often used to help the reader quickly ascertain the article's purpose. Therefore, an abstract should be self-contained and understandable on its own. When used, an abstract always appears at the beginning of a manuscript, acting as the entry point for any academic paper or patent application. Abstracting and indexing services for various academic disciplines are aimed at compiling a body of literature for that particular subject.

Contents

List of Figures

List of Tables

List of Abbreviations

AI	- Artificial Intelligence.
ASHA	- Asynchronous Successive Halving Algorithm.
BO	- Bayesian Optimization.
BOHB	- Bayesian Optimization and Hyperband.
CMA-ES	- Covariance Matrix Adaptation Evolution Strategy.
CPU	- Central Processing Unit.
CV	- Cross-Validation.
DL	- Deep Learning.
DNN	- Deep Neural Network.
GPU	- Graphics Processing Unit.
HP	- Hyperparameter.
HPO	- Hyperparameter Optimization.
MAB	- Multi-Armed Bandit.
ML	- Machine Learning.
NCV	- Nested Cross-Validation.
NN	- Neural Network.
NSGAII	- Non-dominated Sorting Genetic Algorithm II.
OO	- Object-Oriented.
PBT	- Population Based Training.

PSO	- Particle Swarm Optimization.
QMC	- Quasi-Monte Carlo.
SGD	- Stochastic Gradient Descent.
SHA	- Successive Halving Algorithm.
SVM	- Support Vector Machine.
TPE	- Tree-structured Parzen Estimator.

List of Symbols

X - Hyperparameter Configuration (or Candidate Solution, or Trial).

χ - Space of all possible Hyperparameters Configurations.

$\mathbf{X}$ - Search Space.

f - Objective Function.

Chapter 1

Introduction

The introduction should present the topic of the thesis to specify the purpose and importance of the work. It is common practice to have a summary at the beginning of each chapter, like this one.

1.1 Aims and Objectives

The introduction should familiarize the reader with the problem to be addressed and describes:

- The current level of knowledge (with references) as a basis for the work;
- The knowledge gap the study wants to fill;
- Outline of the objectives without anticipating results.

And possibly:

- The chosen approach;
- The structure of the thesis.

1.2 Overview of the Thesis

The rest of this thesis is structured as follows:

- Chapter 2 discusses background knowledge…
- …

Chapter 2

Background

In Machine Learning, Hyperparameter Optimization or "Tuning" is the process of choosing the optimal Hyperparameters for the learning algorithm [1]. Where an "Hyperparameter" is a value used to control the learning process, basically a parameter of the Learning Algorithm and not of the model itself [2].

Thus, Hyperparameter Optimization has the goal to determine the best tuple of Hyperparameters which can minimize the value of the Loss Function [3]. The process of selecting the right Hyperparameters configuration is called "Hyperparameter Optimization" or "Hyperparameter Tuning".

2.1 Introduction to Hyperparameter Optimization

Reason for Tuning Hyperparameters:

The Hyperparameters allow Machine Leaning models to be customized, or tuned, for specific tasks or dataset [2]. Basically, one important aspect of Hyperparameters is to allow a particular model to be reused in different situations and different datasets.

Generally, what each Hyperparameter does when considered alone is known; but it is hard to predict what the best combination of Hyperparameters is the best for a specific problem or specific dataset.

Necessity and Importance of Hyperparameter Optimization:

Hyperparameter Optimization is generated from the necessity to automate the boring and expensive process of manually setting via trial-and-error the Hyperparameters of Machine Learning models [2]. Choosing the optimal architecture for the model is not an easy task, and in many cases the possibilities are so many it is impossible for a human to select the best possible configuration. Thus, the ideal solution would be to make the machine to learn the optimal configuration by itself, exploring different model architectures to find the better performing one. Without automated Hyperparameter Optimization the programmer can only rely on trial-and-error and empirical experience.

Hyperparameter Optimization (HPO) algorithms have not only to automate the process of researching for the best Hyperparameters combination, but also to make this search computationally viable even for more complex models and problems. The main challenge of Hyperparameter Optimization is that is hard to predict how the Hyperparameters could influence each other's. This uncertainty makes the problem even more complex computationally, as pratically all possibilities must be explored.

2.1.1 Introduction

Machine Learning is about predicting and classifying data. ML Models do so using parameters, meaning that the Models are Parametrized so that their behaviour is tuned for any given problem. The large term "Parameter" in Machine Learning can divide in two categories: Parameter (or Model Parameter) and Hyperparameter.

2.1.1.1 Parameters

A model Parameter is a configuration variable internal to the model. The Parameters are part of the model, are learned by training from data, they define the skill of the model and are thus required to make predictions. The Parameters in Machine Learning models have the same function that variables have in traditional programming.

Examples of Parameters in various Machine Learning models: Weights (in Neural Networks); Support Vectors (in SVM); Coefficients (in Linear or Logistic Regression).

2.1.1.2 Hyperparameters

A model Hyperparameters is a configuration that is external to the model. The Hyperparameters cannot be learned from the data and are often set by the programmer. Their values are either determined by heuristics or by optimization techniques.

Basically, Hyperparameters are not Parameters of the Model itself, but are rather Parameters of the leaning process/algorithm used to train the model.

Examples of Hyperparameters in various Machine Learning models: Learning Rate and Epochs (in Neural Networks); C and δ (in SVM); k (in k-nearest neighbors).

2.1.2 Basic Concepts of Hyperparameter Optimization

Two fundamental concepts of Hyperparameter Optimization, indispensable in the understanding of HPO, are Objective Function and Search Space.

2.1.2.1 Objective Function

An Objective Function, in Optimization problems, is the opposite of a Loss Function. More precisely, is a “scoring function”, it serves as a metric to define the performance of the trained model [3] [2].

The performance of a learning algorithm can be seen a function: $f : X \to \mathbb{R}$, that maps the Hyperparameters space $X \in \mathbf{X}$ to the result of the Validation function. Basically $f(X)$ represents an evaluation of the model trained with the tuple X of Hyperparameters.

The goal of Hyperparameter Optimization is to Minimize or Maximize (depends on the Evaluation metric) the value of f, particularly, the tuple X of Hyperparameters that permits so.

2.1.2.2 Search Space

In order to search for the best combination of Hyperparameters, a Search Space has to be defined. The Search Space, or Configuration Space, denoted with $\mathbf{X}$, is the multi-dimensional table of all the possible values the Hyperparameters can be [2].

A Seach Space is a Multidimensional object, where each dimension represents a Hyperparameter, and each point represents a particular Hyperparameters configuration. More specifically, each point in the Search Space is vector, or tuple, containing the values for each Hyperparameter of that combination.

2.1.3 Fundamentals Components of a HPO Algorithm

In general, all complex HPO algorithms two principal components: Samplers and Pruners (also called Searching and Scheduling).

The Sampler, or Search Algorithm, decides which configuration to try. The Pruner, or Scheduler, decides how to allocate the computational budget and when to stop a trial.

On top of these two components, it is to be considered the Tuner, the class that executes the optimization process. Reaching the total three main components.

The concept of a class (the Tuner) which abstracts and executes the Optimization Process, and the concept of Searchers (Sampler) and Schedulers (Pruner) are widely applied in many Hyperparameter Optimization libraries.

2.1.3.1 Sampler

A Sampler, or Searcher, in HPO is the component of the HPO algorithm which selects (“samples”) the next candidate configurations to try and evaluate. Simple Samplers like Grid Sampler (for Grid Search) and Random Sampler (for Random Search) just

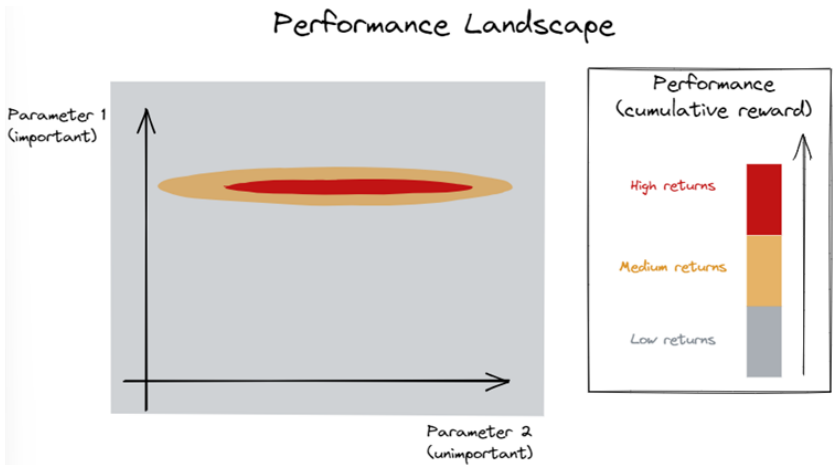


Figure 2.1.1: *A Graph showing how the importance of Hyperparameters can be spread in the Search Space. Source: [4]*

select candidates sequentially and randomly respectively; more complex Samplers, used for example in Bayesian Optimization, make the decision based on previous trials.

In the making of the choice of the Sampler, it is important to understand that Hyperparameters not always have the same importance in the results of the model. Also, some Samplers are more effective with certain models than others. (See Fig. 2.1.1)

2.1.3.2 Pruner

A Pruner, or Scheduler, in HPO is the component of the HPO algorithm which determines if and when a trial needs to be stopped in order to dedicate more resources to more promising candidates. The Pruner has the goal to identify and then discard the bad performing configurations. The final objective is to ensure more computational resources to the remaining configurations.

It is important to understand when to prune. If pruning too early, the selection would be too premature, and some potentially good configuration may be discarded. If pruning too late, resource would have been wasted on bad configurations.

The most basic Pruners, on which more complex Pruners are based, are Median Pruner and Successive Halving. The Median Pruner is the simplest Scheduler (Pruner). At each step, all the configuration with a worse performance than the median of the results in that step, gets discarded. (Fig. 2.1.2)

Successive Halving is a slightly more complex Pruning algorithm than Median Pruner. The idea is simple: it starts with some configurations, at some point they get pruned, and only the most promising ones remain. The complex part is that this whole process is determined by Hyperparameters. Which are: the minimum budget, the initial number of trials, and the percentage of configurations to discard at each step.

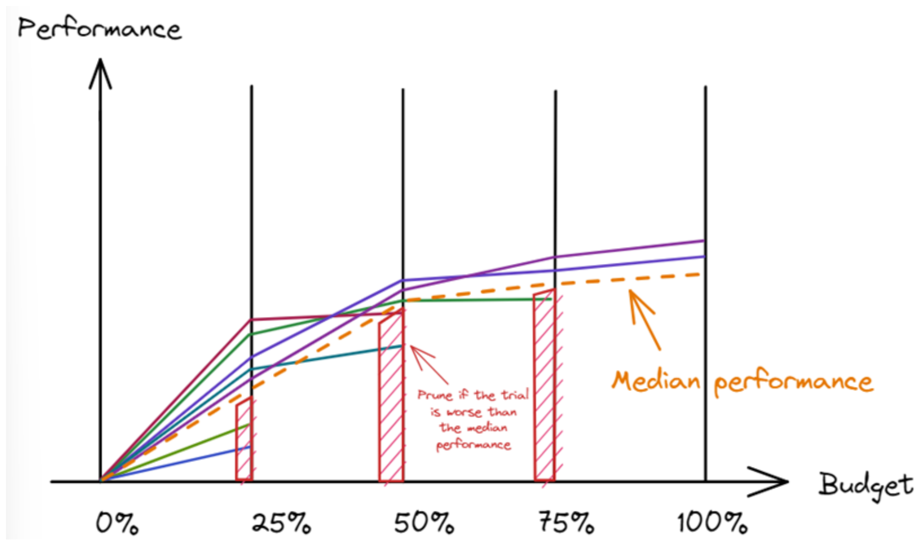


Figure 2.1.2: *A Graph showing the generic functioning of a Pruner Algorithm; in this case, a Median Pruner is showed. Source: [4]*

More of Successive Halving Algorithm and Pruning algorithms in general will be discussed in the following sections.

2.1.3.3 Tuner

The Tuner, in HPO is the function that wraps up the Optimization Process. The Tuner runs the Sampler and the Pruner, manages the optimization of the process (like parallelizing), records the results and the possible errors of each trial. The Tuner can keep records of various information of the Optimization Process, such as errors, runtime, index of the configuration, suggestions of the Samplers, pruning actions of the Pruner.

2.2 Hyperparameter Optimization Algorithms

The goal of a Hyperparameter Optimization Strategy is to find the configuration of Hyperparameters, which lead, after the training of the model, to the minimum error and the maximum performance. There are many Hyperparameter Optimization techniques, some are specialized for specific ML models, other are more generic.

The HPO algorithms can be divided into 2 main categories: Black-Box Optimization algorithms, and Multi-Fidelity HPO algorithms (Fig. 2.2.1). The first category is manly based to the use of Samplers, the second category uses Early Stopping and Pruners.

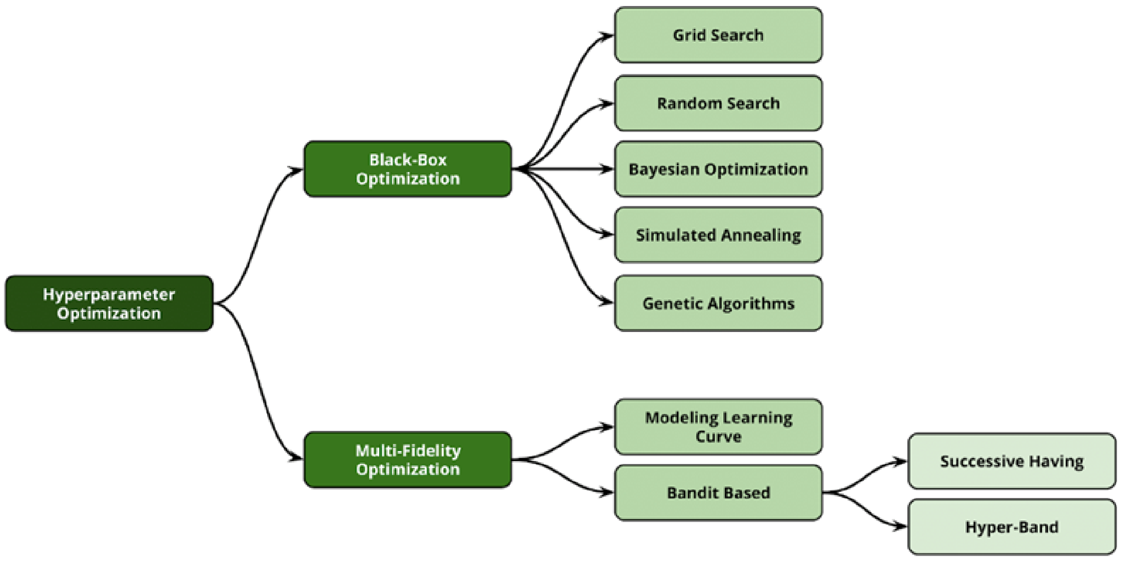


Figure 2.2.1: *Categories of Hyperparameter Optimization Algorithms. Source: [5]*

2.2.1 Grid Search

Grid Search is a traditional way to perform Hyperparameter Optimization. It simply consists of an "Exhaustive Search" of the Optimal Hyperparameters between a subset of specified Hyperparameters.

Grid Search is an approach that will methodically build and evaluate the model for each possible combination of Hyperparameters' values (Fig. 2.2.2).

The performance of a determined configuration of the Hyperparameters for the training of the model is evaluated using techniques such as Cross-Validation or Hold-One-Out. This evaluation is done for each tuple of Hyperparameters. At the end, the configuration with the best performance is the optimal set of Hyperparameters for that model.

Application of Grid Search:

For each Hyperparameter, the programmer defines the range of values that each Hyperparameter can be. Each of these lists of values can be seen as an Array of values, each referring to a Hyperparameter. The whole of these Arrays will then form a Grid. In the Grid Search technique, the Cartesian Product is applied to obtain all the possible combinations of values [6]. The outputs are the setting (the configuration of Hyperparameter) which achieved the highest score, the best performance.

Advantages and Disadvantages of Grid Search:

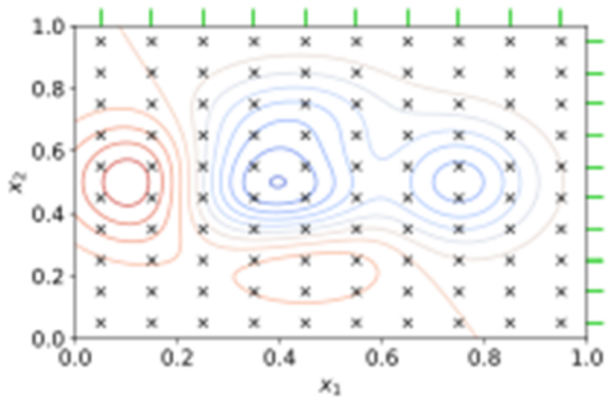


Figure 2.2.2: *Distribution of Candidate Configurations over the Search Space in Grid Search. Source: [3]*

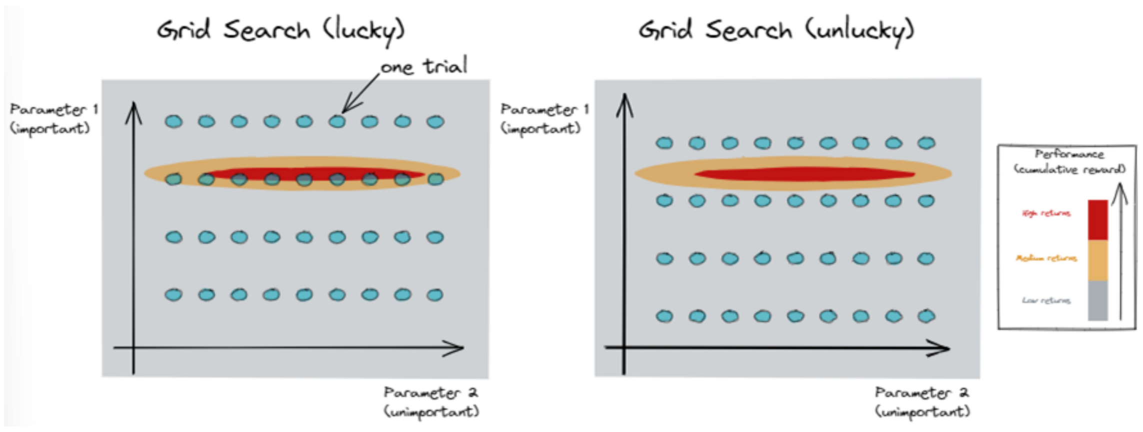


Figure 2.2.3: *Graph comparison showing the phenomenon of "Unlucky Seach" in Grid Search. Source: [4]*

The main Pro of Grid Search is that is an "Embarrassingly Parallel" problem, which means that it is very easy to parallelize the operations. The big Con, on the other hand, is that this technique suffers from the "Curse of Dimensionality", its complexity tends to escalate exponentially very easily, given that there is a Cartesian Product involved [1].

Also Grid Search can suffer from "Unlucky Search", where none of the possible combinations obtained from the Param Grid obtains high results. (Fig. 2.2.3)

2.2.2 Random Search

Random Search is an alternative Hyperparameter Optimization technique to Grid Search [7]. It replaces the "Exhaustive Enumeration" of all possible Hyperparameter combinations with a simple Random Selection of these combinations (Fig. 2.2.4).

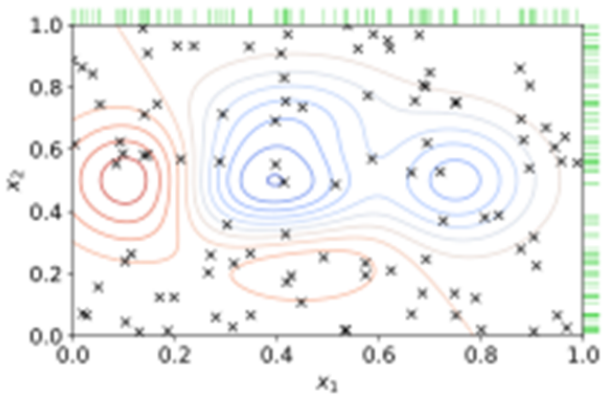


Figure 2.2.4: *Distribution of Candidate Configurations over the Search Space in Random Search. Source: [3]*

Random Search is an approach where rather than defining determined values for the Hyperparameters, statistical distributions are defined, which are used to randomly select values from them.

Differently from Grid Search, is able to explore much more values for both discrete and continuous Hyperparameters; it is an “Explorative” technique. It is considered an important baseline for comparing the performance of the newer Hyperparameters Optimization techniques.

Advantages of Random Search:

Random Search can outperform Grid Search, especially when the number of Hyperparameter which heavily effects the learning process is small [7]. As Grid Search, Random Search is also an Embarrassingly Parallel problem.

Despite these advantages, Random Search is still very basic, and it is not rigorous.

Comparing Grid Search and Random Search:

Random Search can resolve the “Unlucky Search” problem of Grid Search (Fig. 2.2.5). Random Search is also better performing to the “Curse of Dimensionality”, that is, when there are many dimensions involved.

The advantage of Random Search lies in the fact that Hyperparameters, most of the time, are not equally important, and so strategies that examine all the possibilities, are most of the times losing time on useless evaluations [6].

Choosing between Grid Search and Random Search:

Grid Search and Random Search are the simplest Hyperparameters Optimization techniques, so are utilized when the problem is fairly simple and does not involve extreme

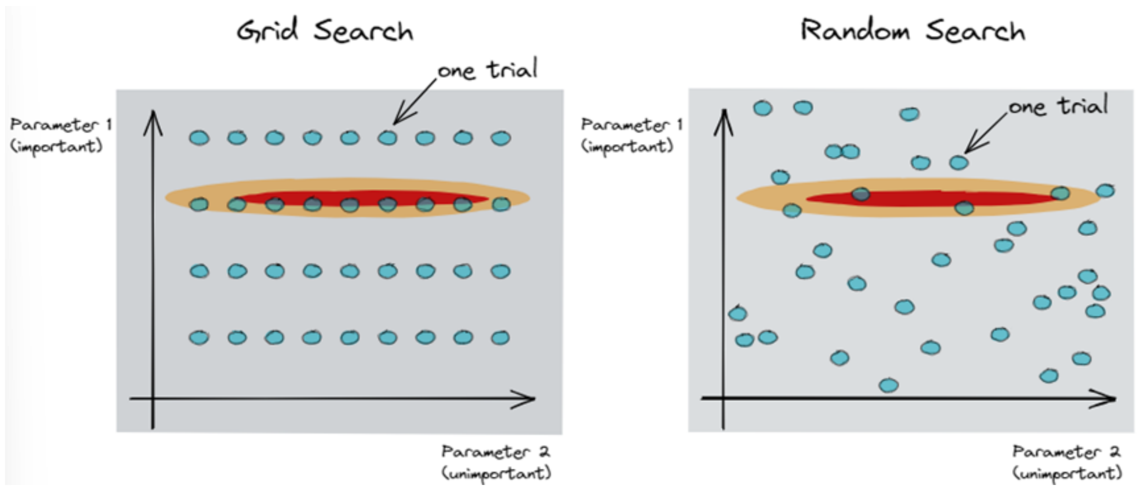


Figure 2.2.5: *Graph comparison between the distribution of candidate configurations in Grid Search and Random Search. Source: [4]*

complexity. Also, both Grid and Random Search tend to become very expensive when the complexity grows, Grid Search especially [8].

In general, Grid Search is more appropriate for problems where the most common best values for a Hyperparameter are already known, and so the search is about finding the best combination between those already good-performing values.

Random Search on the other hand, is more appropriate for discovering new Hyperparameters values or new unexpectedly good combinations.

2.2.3 Bayesian Optimization

Bayesian Optimization is a Hyperparameter Optimization technique which builds a probabilistic model of the target function mapping the Hyperparameters values to the performance obtained on the Validation Set [9].

Basically, this methos smartly explores the potential optimal configurations of the Hyperparameters' values, basing its decision on the performance of the previous configuration. (Fig. 2.2.6)

Exploitation vs Exploration:

In HPO, an important trade-off is the one between Exploitation vs Exploration. Exploitation is the concept of exploiting, using, those value which are already expected to be closed to the optimum. Exploration is the concept of trying new unexplored risky values which could improve the performance and add new knowledge about a particular Hyperparameter.

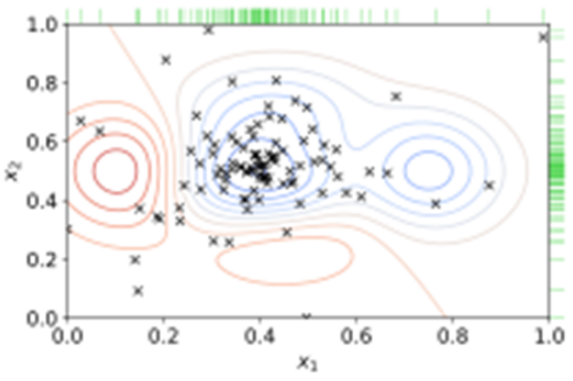


Figure 2.2.6: *Distribution of Candidate Configurations over the Search Space in Bayesian Optimization. Source: [3]*

Bayesian Optimization balances Exploitation and Exploration, considering both concept in order to get closer to the Optimum.

Application of Bayesian Optimization:

The main idea of Bayesian Optimization is to estimate the performance of a certain configuration before that configuration gets tried. The Bayesian Optimization starts randomly from one configuration, or from a configuration which is already known to be at least decent. Then a probabilistic model (Acquisition Function) of the Objective Function is built, the idea is to predict which configuration will maximize the value of the performance, the found configuration will be the next. At each iteration the Acquisition Function is updated, and thus, the prediction becomes more and more accurate. Bayesian Optimization can find the best Hyperparameter Configuration much faster than traditional search algorithms.

2.2.4 Gradient-based Optimization

Gradient-Based Optimization is a Hyperparameter Optimization technique used with specific Machine Learning model such as Neural Networks and SVMs [10] [3].

The Hyperparameters are optimized just like the model Parameters, with Gradient Descent. The more traditional approach of this optimization algorithm uses “Automatic Differentiation” to calculate the so called “Hypergradients”, to update the values of Hyperparameters. Basically, the Gradient of the model is also used during HPO.

While using this method would actually be more efficient, it requires to calculate second-order derivates, which are computationally expensive to approximate. These techniques are therefore very rarely used, because are difficult to apply.

More recent and complex techniques are now being developed for the Gradient-Based

Optimization, making it more efficient and able to work with Discrete Hyperparameters.

2.2.5 Evolutionary Optimization

Evolutionary Optimization is a Hyperparameter Optimization technique which follows a process inspired by biological concept of Evolution. Evolutionary Optimization is applied in almost all fields of Machine Learning, from Statistical ML to Neural Networks and especially in Deep Learning [3] [11] [12].

Steps of Evolutionary Optimization:

Evolutionary Optimization consists in a series of steps, each of which is inspired by a step of the Theory of Evolution.

1. Create an initial population of random solutions (random Hyperparameters values configurations)
2. Evaluate the Hyperparameters tuples (configurations) and obtain their "Fitness Function" (a particular type of Objective Function that summarizes the results of each score metric)
3. Rank the Hyperparameters tuples by their Fitness Function value.
4. Replace the worst tuples with new tuples generated through crossover and mutation (where Crossover and Mutation are the same concepts as they are in Biology)
5. Go back to step 2 until the performance goal of the algorithm is reached or until the performance is no longer improving.

2.2.6 Population-based Training

Population-Based Training (PBT) is a Hyperparameter Optimization algorithms family; it is a broader optimization technique, applied to learn both Hyperparameters values and network weights (or model's parameters in general).

Population-Based Training is very similar to Evolutionary Optimization, actually the latter is a particular case of Population-Based Training.

PBT is very flexible and can be applied to a wide range of different models or problem domains. The optimization process does not depend on aspects like Network Architecture, particular Loss Function, etc.

Main Difference with Evolutionary Optimization:

Like Evolutionary Optimization, Population-Based Training (PBT) involves iteratively replacing poorly performing configuration with better performing modified.

The first difference between PBT and Evolutionary Optimization is that in PBT are not necessarily used Evolutionary Algorithms, but rather a wider range of techniques are applied. Another main difference is the fact that configurations at each generation are not replaced, but rather updated.

In PBT, when possible, the concept of "Warm Starting" is often used, where models are initialized with hyperparameters and weights from already known better configurations to speed-up the training process [3].

2.2.7 Early Stopping-based Techniques

Early Stopping-Based Training is a Hyperparameter Optimization technique family, or rather, are techniques that are often used in combination with the other approaches in order to improve the efficiency of the optimization process.

These techniques are mainly applied for models that have a large number of Hyperparameters and a large number of possible values for those.

The main idea of Early Stopping-Based Training is to iterate between some configurations in searching of the most promising one, using statistical tests to discard bad configurations [4].

Early Stopping-Based Algorithms:

Successive Halving Algorithms (SHA) begins as a random search of the optimal configuration, and periodically halves the trials, removing the worst-performing ones.
Asynchronous Successive Halving Algorithms (ASHA) improves the performance of SHA, executing the process asynchronously.
Hyperband invokes SHA or ASHA multiple times, with different levels of "aggressiveness".
Bayesian Optimization and Hyperband (BOHB) is a combination of Bayesian Optimization and Hyperband.

These and other Early Stopping-Based Algorithms will be discussed in the following section regarding Multi-Fidelity Hyperparameter Optimization.

2.3 Multi-Fidelity Hyperparameter Optimization

Hyperparameter Optimization is one of the most computationally expensive processes in the Training of a Machine Learning model. At the start of Hyperparameter Optimization, a number n of configurations to try is chosen. The ideal solution of course

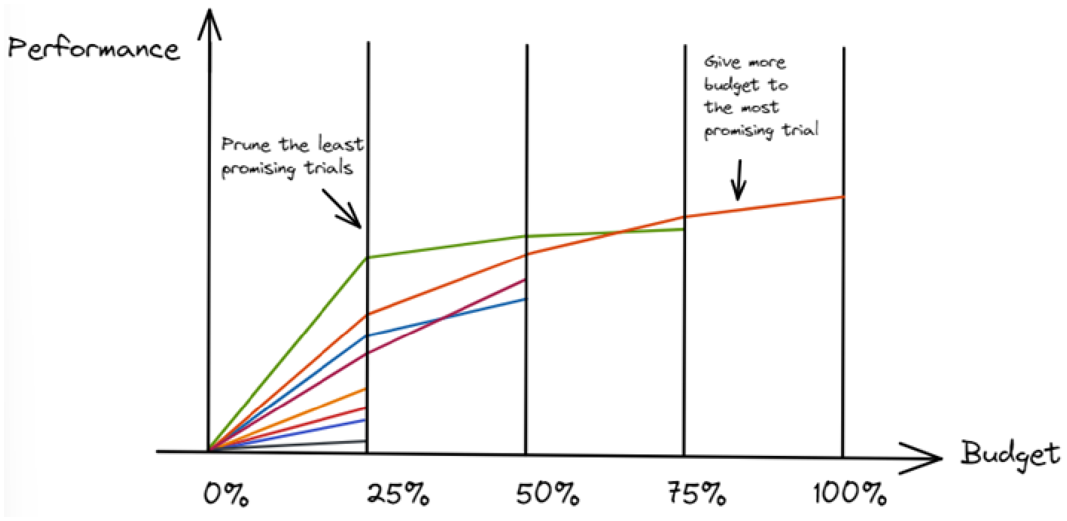


Figure 2.3.1: *Graph showing an example of an Multi-Fidelity HPO; this particular example could represent different types of Pruners as Median, SHA or Hyperband. Source: [4]*

is to try every single configuration, to be sure to find the best one. But when the model is to complex, the budget, intended as computational budget, is limited, and try every single configuration fully is not the most efficient strategy. The solution is to use Multi-Fidelity Hyperparameter Optimization.

Optimizing the Tuning Process:

The main idea is to early-stop a trial of a particular configuration if that trial is not promising. At the start of the search, each configuration is given the same (low) amount of computational budget. At one point, the least promising configurations up to that point are abandoned, and the remaining are given more budget.

This process gets iterated, until at the last iteration only a small set of configurations will have the whole budget to complete their evaluation. (Fig. 2.3.1)

2.3.1 Introduction to Multi-Fidelity HPO

When applying for Hyperparameter Optimization algorithms on Neural Networks, the workload is much bigger than any other Machine Learning model. Using parallelization techniques enables to reduce the time of the processes, but the workload itself still remains the same.

With Multi-Fidelity Hyperparameter Optimization, is possible to reduce the total computational workload the machine has to go through during the Optimization Process. Multi-Fidelity HPO uses Early Stopping and Pruners. Pruners determine if and when an unpromising trial needs to be stopped early, pruned.

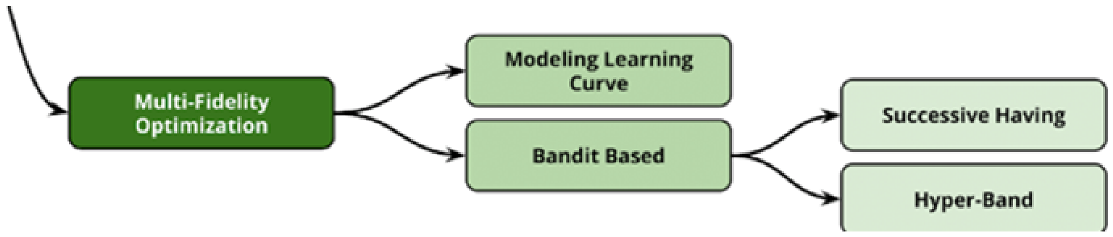


Figure 2.3.2: *Categories of Multi-Fidelity Hyperparameter Optimization Algorithms. Source: [5]*

Process of Multi-Fidelity HPO:

After a few epochs of the trial of one Hyperparameters Configuration, it is already visually evident if that trial is or not promising. The Pruner algorithm decides when and how to stop those unpromising trials, allowing only the promising trials to complete the number of epochs and thus fully complete the training.

During each "Rung" ("Step") of the process, each trial is partially evaluated using the Validation Error, the Validation Error (or any other metric, evaluated on the Validation Set) is used as an estimate of the final score of the configuration. Which precise value the trial should reach to proceed to the next Rung, depends on the Pruning algorithm.

2.3.2 Multi-Fidelity HPO Algorithms

Multi-Fidelity HPO algorithms divides in Multi-Armed Bandit approaches, and Modeling Learning Curve approaches. The most important ones are the Multi-Armed Bandit (MAB) approaches. (Fig. 2.3.2)

The main characteristics of MAB algorithms are the Dynamic allocation of resources, and the balance between Exploration and Exploitation.

The most important MAB algorithms are: Successive Halving (SHA), Hyperband, Iterated Racing, ASHA, BOHB. The most basic Multi-Fidelity HPO MAB algorithm is Successive Halving (SHA). All the basic aspects of SHA, like the trade-off between trials and budget, are valid for all MAB algorithms.

2.3.2.1 Successive Halving

Successive Halving can be seen as a tournament among the candidates (Hyperparameter Configurations) of the HPO process. The Successive Halving algorithm divides the "searching" process in iterations, that are often called "Rungs". At the end of each Rung level, only the best (most promising) candidates pass to the next level [13] [14]. (Fig. 2.3.3)

Number of Candidates and Amount of Resources:

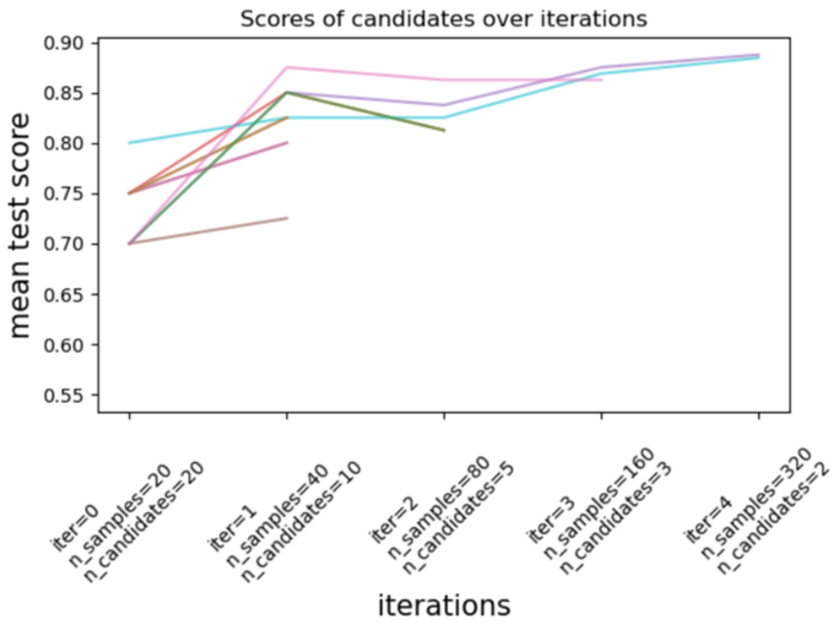


Figure 2.3.3: *Graph showing an example of an Optimization using Successive Halving Algorithm. Source: [13]*

The two most important aspects of Successive Halving are the Number of Candidates and the Amount of Resources dedicated at each iteration. Each Rung has a number of Candidates and an Amount of Resources; the rule which determines how and how much those quantities should vary at each Rung depends on the Pruning algorithm. In simpler Pruning algorithms like Successive Halving, the parameter which control how much the two quantities should vary is called Factor [14].

Factor:

In simpler Pruning algorithms like Successive Halving, the parameter which control how much the two quantities should vary is called Factor.

Factor controls the rate at which the resources grow and the number of candidates decreases. At each Rung, the number of resources (per candidate) is multiplied by Factor, and the number of candidates is divided by Factor [14].

Resource:

The resource parameter is the particular data or metadata to use as “budget”. The choice of this “Resource” depends on the Machine Learning model object of the HPO.

For a model like RandomForest, a good resource is “n_samples”, which is the fraction of the Dataset on which to train the model.

For Neural Networks the most used data a resource is the Hyperparameter “epochs”.

Basically, during successive halving, only the most promising configuration have the opportunity to complete progressively more epochs [14].

Min_Resources:

The parameter min_resources is the amount of resources allocated at the iteration (per candidate). This is a very important number for the success of the algorithm, as if it is too small, potentially good candidates may have a bad start and get pruned prematurely; if it is too big, the algorithm will be inefficient, as most candidates, even the worst ones, will consume a lot of resources at the first iterations [14].

2.3.2.2 Hyperband

Hyperband is an HPO algorithm, it is a form of Multi-Fidelity algorithm [15] [10]. The idea of Hyperband is to efficiently allocate resources to the Optimization Process, combining Random Searching, Successive Halving, and Early Stopping techniques.

Hyperband work is especially utilized in settings where the amount of available computational resources is limited.

Starting from the total set of Hyperparameters Configuration, Early Stopping and Successive Halving are applied during the training process in order to maintain only the best promising configurations. At each iteration the amount of resources assigned to each trial is enlarged, allocating thus more resources to best promising configurations.

Hyperband outperforms most traditional HPO algorithms, it is usable for many ML models, it is scalable and parallelizable; it is particularly used in Deep Learning. Most modern studies are trying to combine Hyperband with Bayesian Optimization, to obtain an even more powerful HPO algorithm.

2.3.2.3 Iterated Racing

Iterated Racing is a complex HPO algorithm based on efficient computational resources allocation to promising configurations [10].

The idea of Iterated Racing is to iteratively race configurations against each other, in pair, and then allocate more computational resources to the winning ones, discarding the losing. The configurations basically participate in a sort of tournament, which ends when a single configuration is remaining or other termination criteria are met.

2.3.2.4 ASHA and BOHB

Asynchronous Successive Halving Algorithm (ASHA) is the Asynchronous version of SHA [16]. Trials are executed asynchronously, gaining in efficiency, but increasing the risk of undeservedly promoting bad configurations to the next Rung.

Bayesian Optimization and Hyperband (BOHB) is an extremely complex HPO algorithm, which mixes ASHA, Hyperband and Bayesian Optimization [1] [17] [5]. Basically, BOHB is a combination of all the best performing HPO algorithms. It is probably, at least results-wise, the best HPO algorithm at the state-of-the-art.

2.4 Hyperparameter Optimization In-Depth

Hyperparameter Optimization is an expensive procedure, both computationally and effort-wise, so good approaches need to be used to make it more feasible.

Techniques like Parallelization makes the HPO algorithm more efficient. Overfitting mitigation techniques prevents the HPs to overfit the validation set. Good practices of HPO coming from Practical Aspects, allow the ML programmer to reduce the time of Trial-and-Error during the set of the algorithm.

2.4.1 Parallelization in HPO

Even when using complex HPO algorithms, the process can take hours or days to finish depending on how big the dataset is, how complex the model to train is, and how many Hyperparameters and Hyperparameter's values are there.

The waiting time could be massively reduced by distributing the trials across parallel resources. There can be distinguish three different types of scheduling: Sequential, Synchronous and Asynchronous [2].

Parallelization in Standard HPO:

In Sequential HPO there is no parallelization, each trial is executed only after the previous trial.

In Synchronous HPO, the new batch of configurations (set of trials) is sampled when the previous set finishes. This means that if a batch requires more time to finish, the next batch will have to wait for the resource to be released.

In Asynchronous HPO, each new Hyperparameters Configuration is evaluated immediately after the previous one is finished; so, each parallel worker is always busy [2]. (Fig. 2.4.1)

Parallelization in Multi-Fidelity HPO:

Pruning algorithms have to be parallelized just like Sampling algorithms.

In Synchronous Pruning algorithms, a worker has to go through idle time in wait that all other workers on complete their subset of trials for the current Rung level. (Fig. 2.4.2)

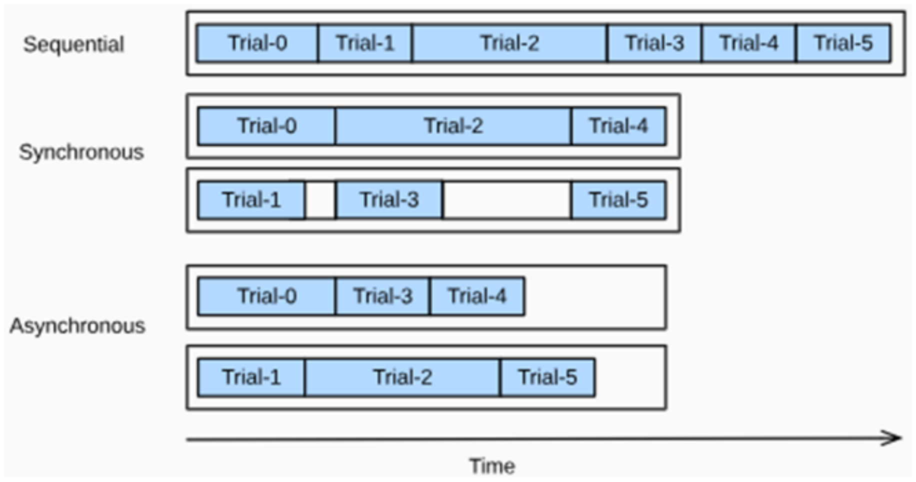


Figure 2.4.1: *Comparsion of Sequential, Synchronous and Asynchronous Optimizations over time. Source: [2]*

In Asynchronous Pruning algorithms, a subset of configurations is promoted as soon as a specified number of observations are collected at the current Rung level. This may indeed lead to reduce the accuracy of promotions, but the impact on the final result is modest [2]. (Fig. 2.4.3)

2.4.2 Hyperparameters Overfitting

When the Hyperparameter Optimization is completed, the chosen set of Hyperparameters are fitted on the training set and gets the best score. The risk is that the model is overfitting. The Hyperparameter Optimization may overfit the hyperparameters values to the validation set.

Solution to Possible Hyperparameters Overfitting:

The solution is to evaluate the generalization performance (score) of the final model on a set that is completely independent to the one used to optimize the Hyperparameters. This can be done with Nested Cross-Validation, a more complex Cross-Validation technique.

In Nested Cross-Validation each fold in further divided into folds. The inner folds are used to tune the Hyperparameters, while the outer folds are used to evaluate the model's performance as usual.

2.4.3 Practical Aspects of HPO

Some practical aspects of HPO.

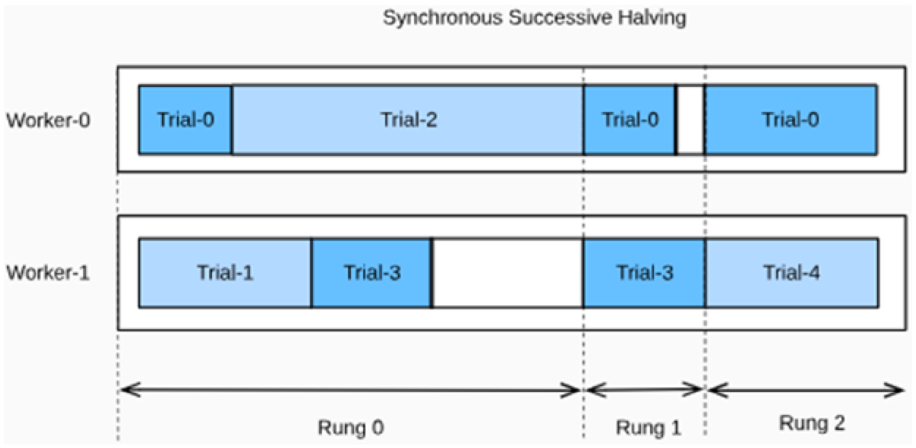


Figure 2.4.2: *Functioning of Synchronous Optimization in Multi-Fidelity HPO techniques. Source: [2]*

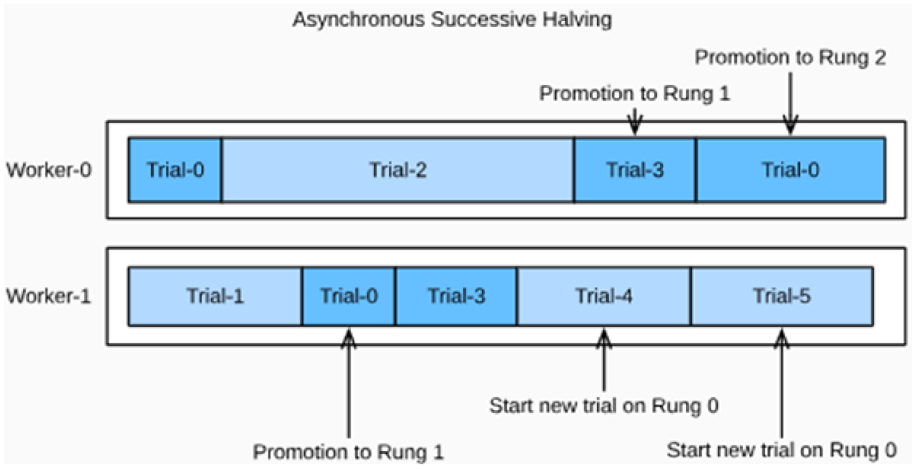


Figure 2.4.3: *Functioning of Asynchronous Optimization in Multi-Fidelity HPO techniques. Source: [2]*

Choosing Evaluation Method:

One often forgotten component of a HPO algorithm is the Evaluation method of the model or trial. An ideal evaluator should be accurate, fast and simple. However, a trade-off remains between accuracy and speed; the more accurate, the probably more expensive it would be.

The most used evaluation method is the traditional, that is, evaluation the current trial using the target Dataset, calculate its Accuracy (or other metrics). Although this is by far the most precise method, it can be slow and resource wasting. Especially in DNNs tuning, where the number of configurations is very high, it is too expensive to evaluate traditionally each trial.

Techniques like Early Stopping Strategies, which evaluate the model performance with an estimate function, rather than the actual evaluation function, allow to save time and resources during the evaluation process [10].

Choosing Performance Metrics:

The choice of the Performance Metric to use for an Optimization Process, should derive from the real world context the model will work on. In general, the most used metric is Accuracy, but in real world contexts it is a bad metric, and depending on the situation, Precision or Recall could be better; also F1-Score if both Precision and Recall are needed.

Nevertheless, it remains true that for the most applications, it is impossible to determine a single metric able to capture all aspects of a model quality in a balanced manner [10].

Choosing the Search Space:

Hyperparameters can be classified based on their domain of possible values. Speaking of numeric HPs, there are mainly two different situations: Bounded in a closed interval (example: $[a, b]$); Bounded from below (example: $[0, \infty]$).

HPs bounded from below can be tuned without modifications. HPs bounded in a closed interval need to be tuned on a logarithmic scale (example: $[\log a, \log b]$), in order to optimize performances.

The size of the Search Space is also important. If the Search Space is too small, it may not contain well-performing Hyperparameter Configurations. If there are too many possible values for each HP, the cost of the search becomes too big. If there are too many HPs to tune, the algorithm will suffer from the Curse of Dimensionality. Because of the problems coming from the size of the Search Space, it would be better to tune as few HPs as possible, each having as few possible values as possible [10].

Choosing the HPO Algorithm:

The principal factor which determines which HPO algorithm to choose is the number of Hyperparameters.

With a very small number of Hyperparameters, between 2 and 3, Grid Search could be the best option as it searches all the possible combinations; furthermore, is the easiest algorithm to interpretate and execute.

As the number of HPs raises, between 3 and 10, Bayesian Optimization is the best option.

If the number of HPs is very high, but the number of impactful HPs is low, then

Random Search and Hyperband are the best choices.

For Search Spaces which are both very large and complex, Evolutionary Algorithms and Iterative Racing are the best choices.

One other factor that influences the algorithm choice is expensiveness of performance evaluation. If the evaluation process is expensive, then Multi-Fidelity algorithms like Hyperband are the best choices [10].

Choosing when to Terminate the HPO:

There are different options for this problem.

Option 1: defining an amount of runtime after which to stop the HPO, solely based on empirical data and intuition.

Option 2: setting a lower bound regarding the generalization error. The problem with this solution is that the specified value could never be reached or could take the algorithm too much time to reach.

Option 3: if no significant progress is made in a specified amount of time, the process is stopped. This option risks to stop the process too early.

Option 4: (only for Bayesian Optimization and similar) when the acquisition function estimates that further progress is little or unlikely [10].

2.5 Hyperparameter Optimization Libraries

Hyperparameter Optimization, as explained before, is an extremely computationally expensive operation, therefore each HPO algorithm needs to be the more optimized as possible. Furthermore, HPO is a boring operation for the ML programmer, who seeks more automated approaches than Trial-and-Error and rerunning.

For these reasons, libraries specifically designed to implement the HPO process exist. Wrapping all HPO components and approaches in a single aimed library, allow to make the process the most optimized and organized as possible.

2.5.1 Scikit-learn

Scikit-Learn is a Python API for Machine Learning. It provides different algorithms for Hyperparameters Tuning [18]. Scikit-Learn HPO algorithms versions are highly optimized and complete. Including parallelization, built-in resampling and OO approach.

GridSearchCV and RandomSearchCV Classes:

The simplest algorithm are Grid Search and Random Search, Scikit-Learn offer the two respective classes GridSearchCV and RandomSearchCV. Both classes use Cross-Validation to evaluate the performance of the model, both classes require as input the model object of the tuning, and the Search Space, also called param_grid.

Other input parameters are: cv (number of folds for the Cross-Validation), scoring (the metric of evaluation used) and n_jobs (number of CPU cores on which to parallelize the search).

To start the Search for the best Hyperparameter combination, the fit() function must be called on the class object, passing as input X and y. The fit() method has as output the object result, which contains the tuple of the best Hyperparameters and the score that particular tuple achieved.

2.5.2 Optuna

Optuna is a Python API for Machine Learning. In particular, the main functions Optuna provides concern Hyperparameter Optimization [19].

2.5.2.1 Introduction to Optuna

Hyperparameter Optimization algorithms are often slow and poorly optimized. Moreover, these algorithms work better on some models and are bad on others. Optuna is specifically designed to work with any Machine Learning or Deep Learning Framework [20].

Adavantages of Optuna:

Optune combines all the good practices for the quality and the efficiency of Hyperparameter Optimization. The search for the optimal Hyperparameters, for the best Hyperparameters combination is automated. Searching and Pruning are automated and use the most efficient algorithms. This allows to obtain the best efficiency. The parallelization process is easy and inexpensive, the search is run on multiple threads and multiple CPUs.

2.5.2.2 HPO in Optuna

Differently from most traditional approaches to Hyperparameter Tuning, Optuna divides the process in rigorous and ordered procedures.

Workflow of Optuna:

The workflow for the use of Optuna, is divided into 3 phases:

1. Defining the Objective Function
2. Creating a Study Object

3. Running the Optimization Process

Optuna Phase 1 - Defining the Objective Function:

The first big difference compared to traditional Hyperparameter Optimization is that the Search Space is not created from examples but is based on Suggestions.

The Objective Function takes as an input a Trial object, on this object they can be called the suggest_categorical() and suggest_float() functions, respectively to generate suggestions (values) for Discrete and Continuous Hyperparameters. (There are also other suggestions like: suggest_int(), suggest_loguniform(), suggest_discrete_uniform(), suggest_uniform(). Each of which can take as input the low and high limits and the step).

Inside the Objective Function, there are not only defined the Hyperparameters, but there is also the code for the definition, training, and evaluation of the ML model object of the tuning.

Optuna Phase 2 - Creating a Study Object:

The study object is created via the create_study() function, which also defines the direction of the optimization (which depends on the evaluation criteria chosen).

For Classification's evaluation metrics, the direction has to be “maximize”, for Regression's evaluation metrics, the direction has to be “minimize”.

Optuna Phase 3 - Running the Optimization Process:

The Optimization Process runs start form the study object, using the method optimize(). The method takes in input the Objective Function and the number of Trials.

The method gives as output the number of the best trial, the accuracy of the best trial, and the tuple of Hyperparameters which represented the best trial.

2.5.2.3 Samplers in Optuna

In the field of study of Hyperparameter Optimization, Samplers define the way to sample the Hyperparameter values. Basically, the Samplers are the Search Algorithms. The create_study() function can take as input a Sampler, which will then be used as the search algorithm in the optimization process. Optuna offers different Samplers.

Tree-Structured Parzen Estimator (TPE) Sampler (TPESampler):

Tree-Structured Parzen Estimator (TPE) Sampler is the default Sampler that Optuna uses. It is a particular case of Bayesian Optimization technique [9].

Random Sampler (RandomSampler):

Random Sampler samples Hyperparameters values randomly. It is equivalent to Random Search [7].

Grid Sampler (GridSampler):

Grid Sampler search across all different Hyperparameters in the Search Space. It is equivalent to Grid Search.

CMA-ES Based Algorithm (CmaEsSampler):

CMA-ES Based Algorithm (Covariance Matrix Adaptation Evolution Strategy) is a powerful search algorithm for continuous Search Spaces [12].

Partial Fixed Sampler (PartialFixedSampler):

Partial Fixed Sampler fixes the values of some Hyperparameters while optimizing the others. It is useful to explore the effects of single Hyperparameters or single smaller groups of Hyperparameters.

Nondominated Sorting Genetic Algorithm II Sampler (NSGAIISampler):

Nondominated Sorting Genetic Algorithm II Sampler is used to optimize multiple objectives simultaneously [11].

Quasi Monte Carlo Sampling Algorithm (QMCSampler):

Quasi Monte Carlo Sampling Algorithm is a less randomic version of the Random Search. Provides better coverage of the Search Space compared to Random Sampling [7].

2.5.2.4 Pruners in Optuna

In the field of study of Hyperparameter Optimization, Pruners define the way to determine if and when to stop unpromising trials early. Pruners allow to reduce the amount of time and resources used for the optimization process. Are used especially in situations where the dataset is very large, or the model is very complex.

The create_study() function can take as input a Pruner, which will then be used to valuate when to stop early a trial.

Optuna offers different Pruners. (Although SuccessiveHalvingPruner and HyperbandPruner outperforms all the other, and are thus the only actually useful ones)

MedianPruner:

MedianPruner is based on the Median Pruning algorithm.

NopPruner:

NopPruner is based on the Non-Pruning algorithm.

PatientPruner:

PatientPruner is a Pruning algorithm with tolerance.

PercentilePruner:

PercentilePruner is a Pruning algorithm where the specified percentage of trials pass.

SuccessiveHalvingPruner:

SuccessiveHalvingPruner is based on the Asynchronous Successive Halving algorithm [14] [16].

HyperbandPruner:

HyperbandPruner is based on the Hyperband algorithm [15].

ThresholdPruner:

ThresholdPruner is based on the Threshold algorithm.

2.5.2.5 Other Aspects of Optuna

Optuna is a big and powerful library, therefore offers a lot of specific functionalities.

Callbacks in Optuna:

Optuna allows to use Callback functions during the Optimization Process. The defined Callback functions define a behaviour, an action, to execute at the end of each trial. One common example of a callback function in this situation is a print function that prints some information regarding the current trial.

Saving and Resuming Optimization Sessions in Optuna:

Optuna allows to save permanently the results of an optimization, to interrupt an optimization session to resume it later, and even to share an optimization session across different machines. All of this is done through the use of an SQLite Database, on which to store the data related to an optimization session.

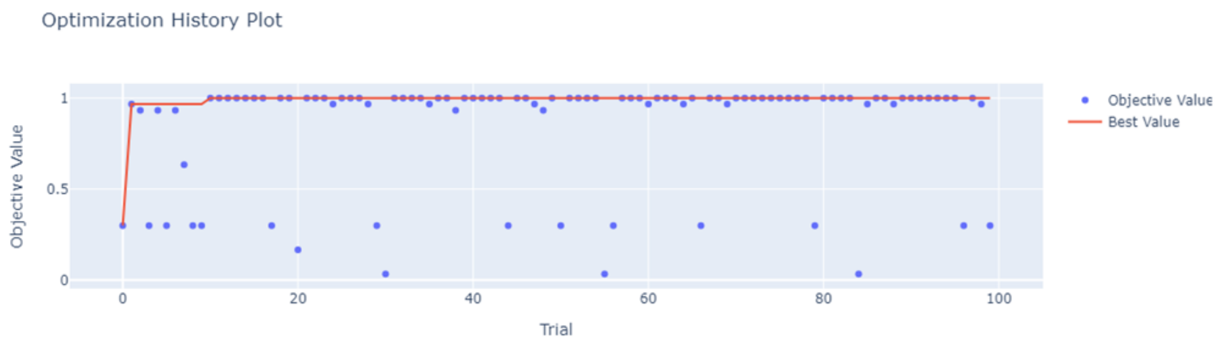


Figure 2.5.1: *Optuna Optimization History Plot*

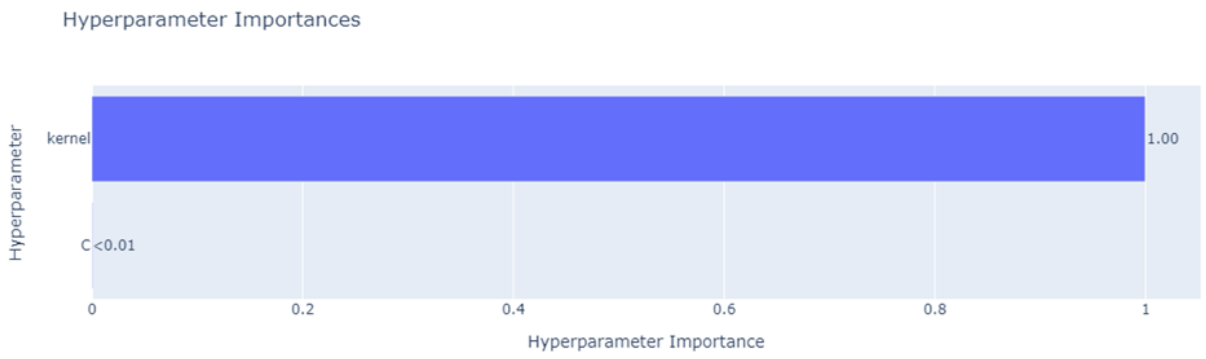


Figure 2.5.2: *Optuna Hyperparameters Importance Plot*

Feature Selection in Optuna:

Optuna can be used in combination with apposite functions of Machine Learning libraries such as Scikit-learn to perform Feature Selection on the model in question. Using the Optuna's suggestion functions, is possible to also tune the function for the Feature Selection.

Visualizing Optimization Results in Optuna:

Optuna preserves inside the study object some metadata that can be use to plot the history of the Optimization Process. Through the module Visualization, Optuna offers different functions to plot different aspects of the Optimization Process.
It can be plotted the History of the Optimization Process. (Fig. 2.5.1)
It can be plotted the importance of some Hyperparameters over others. (Fig. 2.5.2)
It can be plotted the relationship between Hyperparameters. (Fig. 2.5.3)
It can be plotted the distribution of the trial result for each Hyperparameter value. (Fig. 2.5.4)

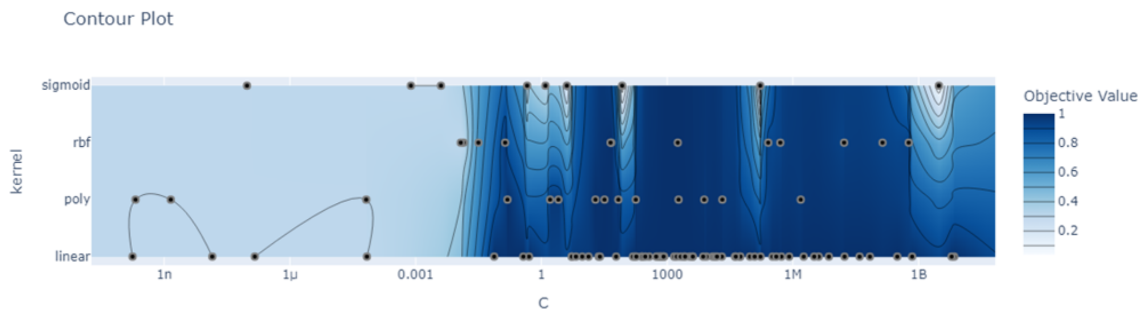


Figure 2.5.3: *Optuna Contour Plot*

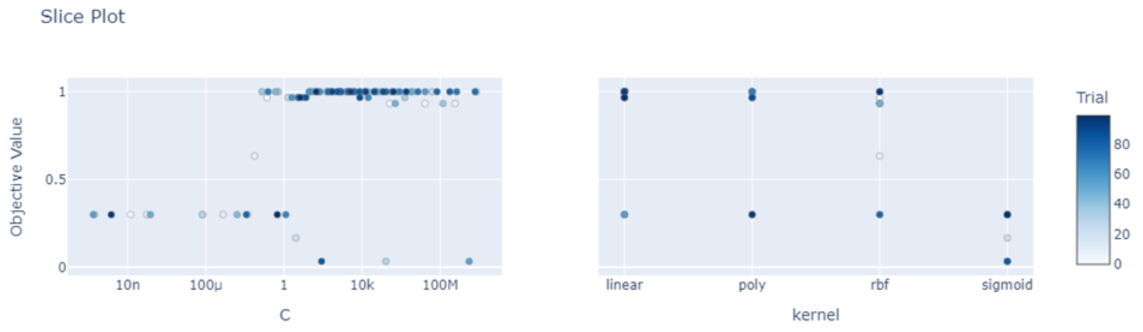


Figure 2.5.4: *Optuna Trial Distribution Plot*

2.6 Hyperparameter Optimization for Neural Networks

HPO is a very computationally expensive process, which cost progressively increments as the Search Space, and so the number of HPs considered enlarges [17]. Neural Networks, especially Deep Neural Networks, are the most complex type of ML model, and the number of HPs in a DNN model is very high.

2.6.1 Main Hyperparameters for Neural Networks

Therefore, an important aspect of HPO is understanding which of the Hyperparameters have the stronger effects on Parameters (in the case of NNs, Weights) during the training.

Hyperparameters, especially when talking about Neural Networks, can be divided into two categories: Hyperparameters for Training, and Hyperparameter for model design.

In Neural Networks, the most used Training related Hyperparameters are: Learning Rate, Batch-size, Activation Function, Optimizer (algorithm to update the weights, e.g. Gradient Descent, Adam). (Table 2.6.1)

In Neural Networks, the most used Model-Design related Hyperparameters are: Number of Hidden Layers, Width of Layers. (Table 2.6.1)

Table 2.6.1: *Table showing the most common Hyperparameters in Neural Networks and their suggested ranges of values.*

Name	Type	Hyperparameter Range	log-scale
learning rate	float	$[10^{-6}, 10^{-1}]$	yes
batch size	integer	$[8, 256]$	yes
momentum	float	$[0, 0.99]$	no
activation function	categorical	{relu, sigmoid, tanh}	-
number of units	integer	$[16, 1024]$	yes
number of layers	integer	$[1, 6]$	no

Learning Rate:

Learning Rate is an Hyperparameter which determines the length of a "Learning Step", it basically represents the velocity of learning. The optimal value for the Learning Rate is different for every kind of problem, so in general only a few possibilities, one for each order of magnitude, are considered.

More complex algorithms, use techniques such as Learning Rate Decay, where the value of Learning Rate decreases over time during the training, additional Hyperparameters are also introduced to tune this sub-algorithm [17].

Optimizer:

Optimizer is the Hyperparameter which represents the algorithm used to update the Weights of the Neural Network. More specifically is that algorithm which is used to minimize the Loss Function during the training, calculating the gradients of the loss function.

The traditional Optimizer is Stochastic Gradient Descent; but nowadays there multiple more complex and better performing Optimizers, such as: Adam, RMSprop, Adagrad, Adadelta. Most complex Optimizers have their own Hyperparameters, which optimal values choice depends on the type of the model and its other Hyperparameters.

The choice of the Optimizer principally depends on the other HPs values. In general, Adam is well performing in most cases, RMSprop is good for Deep Networks [17].

Model Design Related Hyperparameters:

The two most important HPs of this category for Neural Networks are Number of Hidden Layers, Width of Layers (or Number of Neurons in each Layer).

The Number of Hidden Layers, generally speaking, grants the best accuracy results when it is a big number. Of course, this is a trade-off because the more Hidden Layers there are, the less lightweight the model will become, and so the more expensive it will become to train and tune.

The Number of Neurons in each Layer, if it is too small will cause the model to underfit, if it is too big the model will overfit, and the cost of training would be too high. So, some suggestion can be followed: [17]

$$w_{input} < w < w_{output} \tag{2.6.1}$$

$$w = \frac{2}{3} w_{input} + w_{output} \tag{2.6.2}$$

$$w < 2w_{output} \tag{2.6.3}$$

Regularization:

Regularization is a Hyperparameters which represents the Regularization function, that is, a function used to mitigate the model's overfitting and reduce the Generalization error.

In general, the two most used Regularization techniques are L1 and L2, which both have their pros and cons, with L2 being the most used one. Although widely used for most ML models, for Neural Networks there are better alternatives.

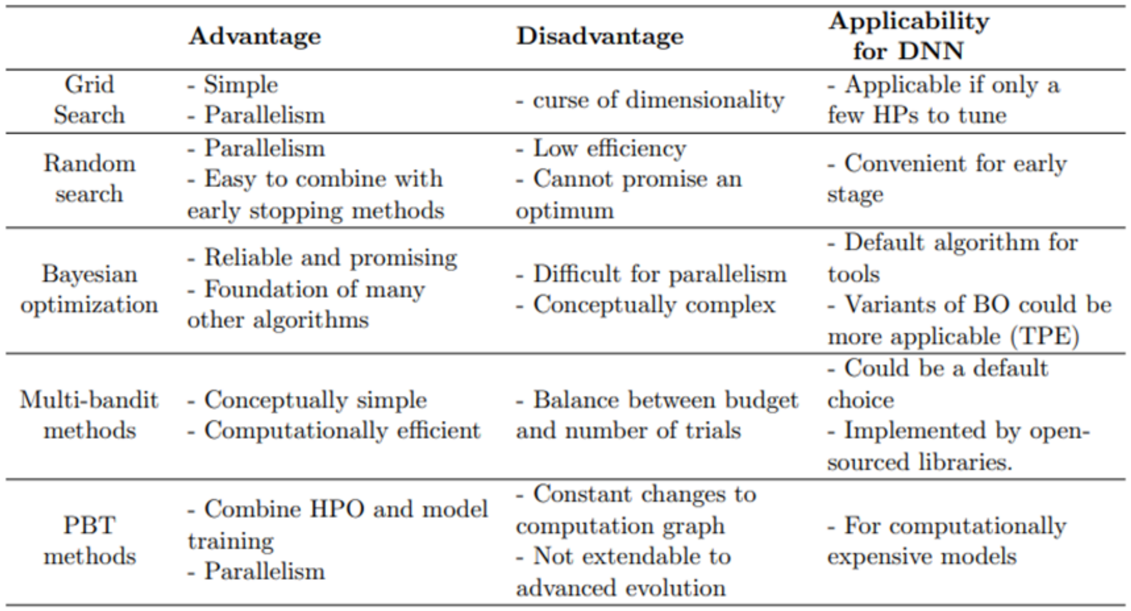
Data Augmentation add fake synthetic data to the training Dataset, in order to enhance the generalization power of the model.

Dropout is the most used Regularization technique for NNs; during the training some Weights are "nulled", simplifying the model and making it less likely to Overfit the training set, improving the generalization power [17].

Activation Function:

Activation Function is a Hyperparameter which represents the mathematical function applied to the output of each Neuron. The goal of Activation Function is to introduce non-linearity into the network, without which the model could not learn complex patterns.

The most famous Activation Functions are the following: Sigmoid, Hyperbolic Tangent (tanh), ReLU, Softmax. Sigmoid works well for simple Neural Networks, is less efficient for more complex models. Softmax is only used in the output layer of NNs for



	Advantage	Disadvantage	Applicability for DNN
Grid Search	- Simple - Parallelism	- curse of dimensionality	- Applicable if only a few HPs to tune
Random search	- Parallelism - Easy to combine with early stopping methods	- Low efficiency - Cannot promise an optimum	- Convenient for early stage
Bayesian optimization	- Reliable and promising - Foundation of many other algorithms	- Difficult for parallelism - Conceptually complex	- Default algorithm for tools - Variants of BO could be more applicable (TPE)
Multi-bandit methods	- Conceptually simple - Computationally efficient	- Balance between budget and number of trials	- Could be a default choice - Implemented by open-sourced libraries.
PBT methods	- Combine HPO and model training - Parallelism	- Constant changes to computation graph - Not extendable to advanced evolution	- For computationally expensive models

Figure 2.6.1: *Summary of main Hyperparameter Optimization Algorithms and their applicability to Neural Networks. Source: [17]*

Multiclass Classification.

The most used Activation Function nowadays is ReLU, convient especially for its simplicity. Many variants of ReLU have being developed in the recent years such as Leaky ReLU, PReLU, ElU, SeLU. Alternatives to ReLU are Maxout and Swish, which while they do resolve some problems of ReLU, are not that good for every situation [17].

2.6.2 HPO Algorithms applied to Neural Networks

An overview on potential of each Major HPO algorithms category and their applicability for DNNs.

A brief summary of the most important HPO algorithms, with advantages, disadvantages and applicability for Neural Networks, is showed in the following table. (Fig. 2.6.1)

Applicability to Deep Neural Networks:

Grid Search is applicable only when a just a small subset of the DNN's HPs is searched. Moreover, the user needs to already have knowledge about empirical good values of the selected HPs to further narrow the search.

Random Search can be used, only if combined with Early Stopping techniques, for the early stages of HPO.

Bayesian Optimization, at least in its original form, is not ideal for DNNs tuning. Variants like TPE (Tree-Structured Parzen Estimator), which introduce parallelization, are appliable to many DNNs models.

Multi-Armed Bandit algorithms (SHA, Hyperband, ASHA) are the default choice for DNNs' tuning. They outperform most algorithms in DNNs' models.

Population Based Techniques are a good choice for large models and large datasets [17].

Chapter 3

Methodology

It is worth noting that the structure of this chapter is highly general and can be modified to suit specific needs. For example, in a Computer Vision and Deep Learning thesis, you may want to describe the proposed method(s).

3.1 Requirements

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3.2 Particle Swarm Optimization

3.2.1 Introduction to Particle Swarm Optimization

Particle Swarm Optimization (PSO) is an Optimization approach originally proposed by Kennedy and Eberhart in 1995 [21]. The idea was to replicate the behaviour of certain animal species which moves in groups, like flocks of birds or shoal of fishes.

The reason is the belief that each singular individual in the group can profit form the experience of all other members; basically, each individual can benefit from what he learns from other individuals, and in the same way, it can share its discoveries to the other individuals [22]

3.2.1.1 Particle Swarm Optimization as an Optimization Problem

Translating this behaviour into specific terms, each individual can be interpreted as an agent which “flies” in the Search Space, where each physical position is a candidate solution in the n-dimensional Search Space, and the best solution found by the whole group is the best solution.

The best solution found by the whole group will probably not be the real global optimal solution but is a solution which is near the global optimal.

Formalization of Particle Swarm Optimization:

The goal of the Particle Swarm Optimization, as all Optimization problem, is to find value which maximizes (or minimizes) the value of an objective (or Fitness) function f.

Where f is defined on a Search Space $\mathbf{X}$, the multi-dimensional vector representing all the possible solutions in the defined limits. The PSO algorithm will return the vector X, which represent a single solution, which maximizes (or minimizes) the value of f.

General Procedure of PSO:

To find the maximum (or minimum) of the Fitness function f, of course the best solution would be to perform an exhaustive search on all the possible solutions. This approach is of course too computationally expensive, and basically unapplicable to higher-dimensional spaces [22].

Therefore, in PSO, the same way as a flock of birds searches for food, moving in the air, the algorithm starts with a number of random points in the search spaces, which are called Particles, and have them look for an optimal value by roaming in random directions.

At every atomic step, each Particle (individual) search for its Local Optimal value, basing its research on both the current Local Optimum, and the current Global Optimum of the whole Swarm. After a certain number of iterations, the maximum (or minimum) value found as Global Optimum is considered the optimal value for the function f.

3.2.1.2 Particle Swarm Optimization Algorithm

At the start of each iteration, each Particle has a position $x_i(t)$ and a velocity $v_i(t)$.

Update of Particles:

At the next iteration, the update function will update the position and velocity of a Particle applying the following rules:

$$v_i(t+1) = \alpha v_i(t) + \beta_1(x_i^{(local)}(t) - x_i(t)) + \beta_2(x^{(global)}(t) - x_i(t)) \tag{3.2.1}$$

$$x_i(t+1) = x_i(t) + v_i(t) \tag{3.2.2}$$

Parameter α represents an "inertia", it decreases over time, that is when t increases.

Parameters β_1 e β_2, are the weight assigned to the Local and Global "parts" respectively. They are usually chosen randomly at each step. They are called Cognitive Coefficient and Social Coefficient respectively.

Parameter $x_i^{(local)}$ is the Local Memory of an individual (Particle), represents the best coordinates in the Search Space visited by that individual. It is updated as follows:

$$x_i^{(local)} = x_i(\arg\max f(x_i(u))) \tag{3.2.3}$$

Parameter $x^{(global)}$ is the Global Memory of the Swarm, represents the best coordinates in the Search Space visited by an individual in the Swarm, basically the best solution so far. It is updated as follows:

$$x^{(global)}(t) = x_j^{(local)}(t). \tag{3.2.4}$$

Whenever Local Optimum (for each Particle) and Global Optimum (for the whole Swarm) is found, these values are updated.

Advantages of PSO Algorithm:

Differently from other more traditional Optimization algorithms, PSO does not depend on the gradient of the objective function. Basically, differently from Gradient Descent, the movement of a Particle does not depend on which direction is "uphill" or "downhill", because the Particle is guided by Local Optimum and Global Optimum only.

This advantage makes PSO Algorithms suitable for problems which objective functions are not differentiable. It is thus an algorithm appliable to a wider range of optimization problems. Another advantage is that PSO is an "Embarrassingly Parallel" problem; a type of problem very easy to parallelize, as each particle can be updated in parallel.

Visual Example of PSO:

In the figure (Fig. 3.2.1), it can be observed how the particles progressively converge to the Global Optimum, which is the best solution found by the whole Swarm.

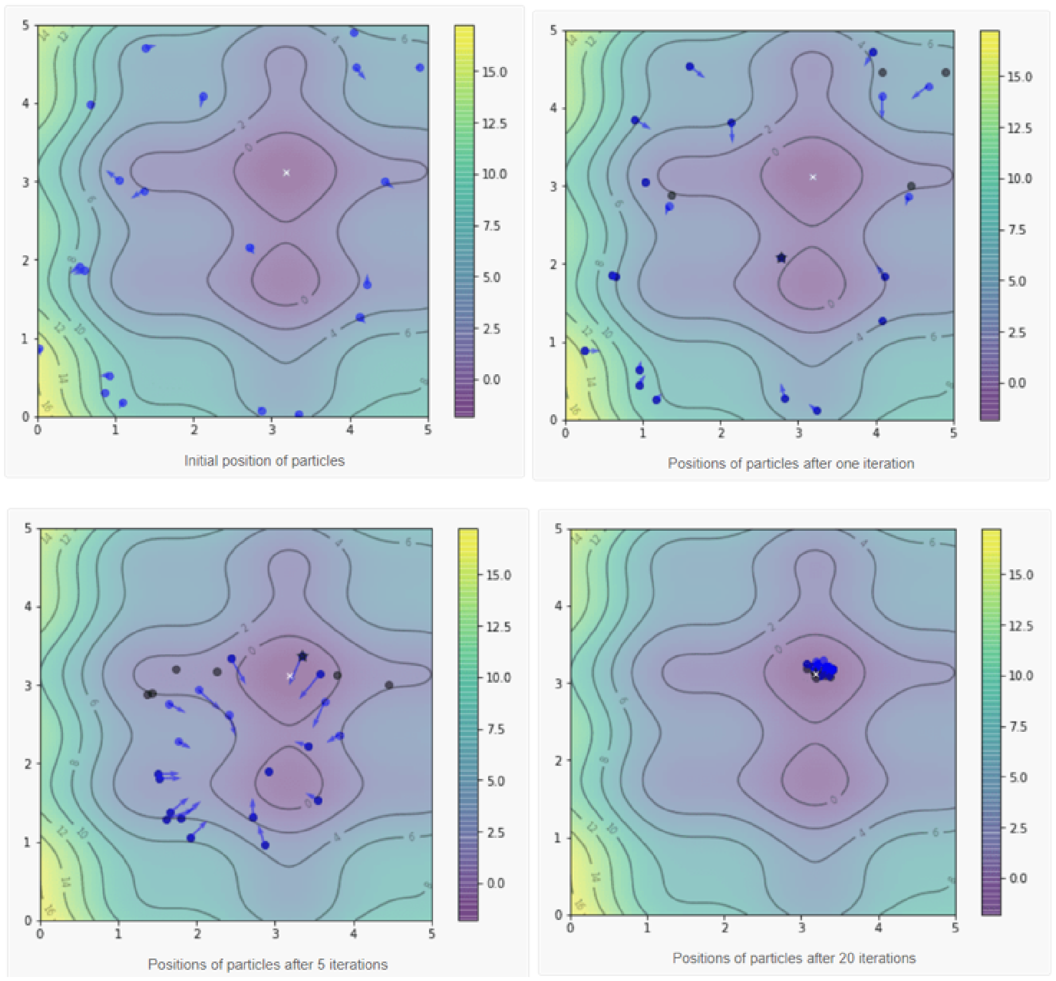


Figure 3.2.1: *Visual representation of the convergence of the Particles to the Global Optimum in the Search Space in a Particle Swarm Optimization Algorithm. Source: [22]*

3.2.2 Components of Particle Swarm Optimization

Particle Swarm Optimization can have better results, be faster and be cheaper compared to other methods. It is an easy problem to parallelize. Does not require the target function to be differentiable. It has few and not much complex hyperparameters.

In short, PSO is a has a large set of advantages, it is a modern solution to perform optimization tasks. The final objective remains the same as every optimization problem, minimize (or maximize) a given function.

The three components of PSO algorithm are: Particle, Swarm and Optimization.

3.2.2.1 Particle

The Particle Swarm Optimization is inspired by the behaviour of flocks of birds. Therefore, the term “Particle”, refers to a single individual in the Swarm.

Every Particle is defined by its Position and their Velocity in the Search Space. The Position in the Search Space allows for the evaluation of the values corresponding to that position, whereas the Velocity allows Particles to move stochastically in the Search Space, in seek of new better positions, and thus solutions.

Initialization of Particles:

At the start of the optimization process, the positions of the Particles are defined randomly: random values of the Search Space are assigned to the Particles. The Velocity and direction of the Particles is also randomly generated.

Evaluation of Particles:

The Particle in the PSO is thus an agent, the position this agent has in the Search Space is a potential solution, a candidate solution. Each Particle has Fitness values, which are evaluated by the Fitness Function, which is the function object of the optimization. The Fitness Function evaluates a Particle’s Positions, taking in input the values of the Search Space corresponding to that position.

3.2.2.2 Swarm

The Swarm is the Population of Particles of the optimization process, it represents the flock of birds.

PSO has similarities with Genetic Algorithms, but differently from them, there are no Evolution Operators to update the individual for the next generation. In PSO the next generation of individuals is an update of the former generation, which Position and Velocity are updated in order to improve the Fitness.

Inertia, Cognitive Intuition and Social Intuition:

After each iteration in the Search Space, the Velocity of each Particle is stochastically accelerated; consequently, also its Position will change.

The value the Velocity is going to be updated is influenced by three factors: Inertia, Cognitive Intuition and Social Intuition. (Fig. 3.2.2)

Inertia is the tendency to keep the Velocity from the previous iteration. Cognitive Intuition is what makes the Particle accelerate toward the previous best Local Position, which is the best Position (corresponding to best Fitness) which that Particle has achieved so far. Social Intuition is what makes the Particle accelerate toward the

$$P_i^{t+1} = P_i^t + V_i^{t+1}$$

$$V_i^{t+1} = \underbrace{wV_i^t}_{\text{Inertia}} + \underbrace{c_1 r_1 (P_{best(i)}^t - P_i^t)}_{\text{Cognitive (Personal)}} + \underbrace{c_2 r_2 (P_{bestglobal}^t - P_i^t)}_{\text{Social (Global)}}$$

Figure 3.2.2: *Alternative update functions for the Velocity of the Particles in a Particle Swarm Optimization Algorithm, where the three components of the update function are represented, and additional parameters, c_1 and c_2 are introduced. Source: [23]*

previous best Global Position, which is the best Position (corresponding to best Fitness) which the Particles in the Swarm have achieved so far.

3.2.2.3 Optimization

The Optimization component of PSO, consists in the actual process of update of the parameters, with the correlate setting of Hyperparameters. Inertia, Cognitive and Social coefficients have the function to control the levels of Exploitation and Exploration.

Exploitation means to use the good solutions found so far in order to search for even better solutions in that mathematical neighbourhood. Exploration means to explore distant sections of the Search Space, in seek of new information on the Space or potentially improving solutions.

Weighting of Acceleration:

At each iteration, both the Cognitive section and the Social section of the update formula are weighted by random terms [23] [24]. Basically, Cognitive acceleration and Social acceleration are stochastically adjusted by weights, to make the update process more random and less deterministic. The possible values that the coefficients can assume are in Fig. 3.2.3.

Value of Inertia:

The value of the Inertia coefficient defines the ability of the Swarm to change direction.

Lower values of Inertia coefficient lead to better convergence; so low Inertia increases the exploitation of the best solutions.

Higher values of Inertia coefficient increase the exploration around the best solutions. Values too much high for Inertia, values > 1, cause divergence of the Particles [23].

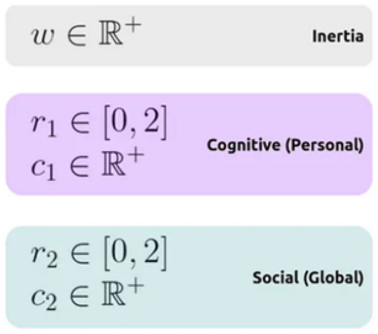


Figure 3.2.3: *Ranges of values for the Parameters of a Particle Swarm Optimization Algorithm. Source: [23]*

Value of Cognitive Coefficient:

The value of Cognitive coefficient defines the ability of the Swarm to be influenced by the personal solutions of each Particle.

If the Cognitive coefficient is too high, then there will be no convergence, as each individual would be too focused on its optimal solution [23].

Value of Social Coefficient:

The value of Social coefficient defines the ability of the Swarm to be influenced by the best global solutions of the whole Swarm.

Auto Hyperparameters:

Inertia, Cognitive and Social coefficients are all three Hyperparameters for the optimization process of PSO.

Searching for the optimal values of there coefficient is a complex and expensive task, as it would require another optimization process. Moreover, the optimal value of these parameter changes during the optimization process, as the iterations go on.

Therefore, a satisfactory solution is to update the coefficients over the iterations [23]. Good reccomended update formulas, coming from empirical studies, are the equations below.

The initial values should guarantee high exploration and more individuality, so high Inertia and Cognitive and low Social.

Toward the end of the optimization, values should guarantee high exploitation and convergence to local optimum, so low Inertia and Cognitive and high Social.

$$w = 0.4\frac{(t-N)}{N^2} + 0.4 \tag{3.2.5}$$

$$c_1 = -3\frac{t}{N} + 3.5 \tag{3.2.6}$$

$$c_2 = +3\frac{t}{N} + 0.5 \tag{3.2.7}$$

3.2.3 Principles, Applications and Variants of Particle Swarm Optimization

Principles of Swarm Intelligence, real-world applications of Particle Swarm Optimization, and PSO variants.

3.2.3.1 Principles of Particle Swarm Optimization

Particle Swarm Optimization has evolved a lot during its first experimental phase. While it was originally meant to simulate the behaviour of a Flock of birds, the final form of the algorithm resembles more a Swarm than a Flock. For this reason, it took the name of Swarm Optimization.

The first researcher who talked about Swarm Intelligence, Millonas [25], defined 5 principles of Swarm Intelligence. Particle Swarm Optimization adhere to all the 5 principles.

1 - Proximity Principle:

The population should be able to perform simple space and time computations.

2 - Quality Principle:

The population should be able to respond to quality factors in the environment.

The PSO adheres to this principle because the population, the Swarm, tend to follow the positions Local Optimum and Global Optimum, which are the environment's factors.

3 - Diverse Response Principle:

The population should ensure enough diversity in its responses.

The PSO adheres to this principle because the responses range from the Local Optimum of the Particle and Global Optimum of the Swarm, ensuring the diversity.

4 - Stability Principle:

The population should not change its behaviour at each environmental change.

The PSO adheres to this principle because the population only changes when the Global Optimum is updated and is therefore stable.

5 - Adaptability Principle:

The population should be able to change its behaviour when it Is worth the computational price.

The PSO adheres to this principle because the population does change its behaviour when the Global Optimum is updated.

3.2.3.2 Applications of Particle Swarm Optimization

One of the main reasons why Particle Swarm Optimization is widely used as optimization technique, is that it is well appliable to a wide range of problems [26]. The advantage of PSO is that it has a small number of Hyperparameters to set, this allow the algorithm to be easily appliable to specific applications [26] [22] [21] [24].

Evolution of Neural Networks: Particle Swarm Optimization can substitute traditional methods for the optimization of a Neural Network weights. PSO is able to reach or outperform traditional approaches like Backpropagation. PSO can work so well with Neural Networks, that not only can be used to optimize the networks' weights, but also their structure. PSO applies is effective for any network architecture [26].

Human Tremor Diagnosis: PSO has been used in combination with Neural Networks for the diagnosis of Humar Tremor conditions, for example Parkison's Disease [26].

End Milling Manufacturing: PSO has been used in combination with Neural Networks for End Milling manufacturing [26].

Voltage Stability: PSO has been used for a dynamic power and voltage control in a Japanese electric establishment [26].

Determination of Battery State: PSO has been used in combination with Neural Networks for estimating the state-of-charge of electrical or hybrid vehicles [26].

3.2.3.3 Variants of Particle Swarm Optimization

The Particle Swarm Optimization is a popular and effective optimization technique; therefore, many variants of the approach have been developed [23].

Variants primarily focalize of adding evolutionary capabilities to PSO or improving performance with Hyperparameters.

Hybrid of Genetic Algorithm and PSO (GA-PSO):

Hybrid of Genetic Algorithm and PSO (GA-PSO) implements the mainly aspects of GA approach as the capability of breeding and crossover.

Hybrid of Evolutionary Programming and PSO (EPSO):

Hybrid of Evolutionary Programming and PSO (EPSO) implements the tournament selection in PSO, where the losing Particles changes their position.

Adaptive PSO (APSO):

Adaptive PSO (APSO) applies Fuzzy Logic to the Inertia coefficient. In addition, uses another PSO to perform the Hyperparameter Optimization for the first PSO.

Multi Objective PSO (MOPSO):

Multi Objective PSO (MOPSO) implements the concept of Pareto Dominance to determine which Particle should set the Global Optimum [24].

Discrete PSO (DPSO):

Discrete PSO (DPSO) makes the performance of optimization better, for example using mixed search approach [24].

3.3 Analysis

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3.4 Design

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3.5 Implementation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Chapter 4

Results

Before discussing the results, you may want to describe the experimental setup: the dataset used, hardware platforms, hyperparameters, performance metrics, etc.

4.1 Experiment 1

4.1.1 Weed Map Dataset

4.1.1.1 Introduction to Weed Map Dataset

Weed Map Dataset is the dataset for the research "WeedMap: A large-scale semantic weed mapping framework using aerial multispectral imaging and deep neural network for precision farming" [27].

Each picture of the Dataset shows an image of a cultivated field, where there can been distinguished Dirt, Weeds and Crops.

The image below (Fig. 2.1.1) is an example on one of the images. It is first showed the entire Orthomosaic Map, and then different zoom levels. This puts in evidence the large scale of the image and the difficulty in distinguishing between Crops and Weeds.

Data Collection:

The data included in the dataset was collected from aerial images (with Drones) from two different Sugar Beet fields: Eschikon (Switzerland) and Rheinbach (Germany). (Fig. 4.1.2)

In particular, the utilized Drones are two commercial quadrotor UAV, equipped with multispectral cameras.

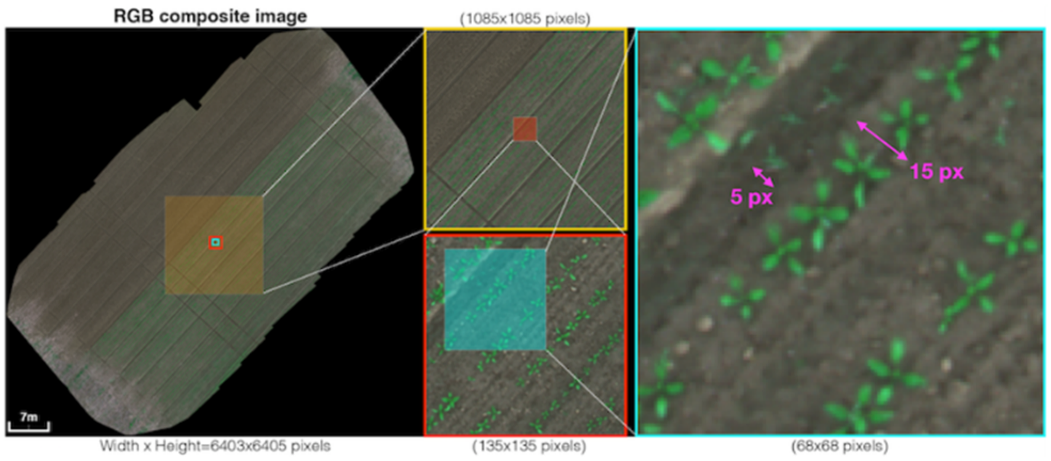


Figure 4.1.1: *Example of image in Weed Map Dataset; with different zoom levels. Source: [27]*

Description	1st campaign	2nd campaign
Location	Eschikon, Switzerland	Rheinbach, Germany
Date, Time	5-18/May/2017, around 12:00 PM	18/Sep/2017, 9:18-40 AM
Aerial platform	Mavic pro	Inspire 2
Sensor[a]	Sequoia	RedEdge
# Orthomosaic map	3	5
Training/Testing multispectral images[b]	227/210	403/94
Crop	Sugar beet	
Altitude	10 m	

Figure 4.1.2: *Information on the campaigns of data collection of Weed Map Dataset. Source: [27]*

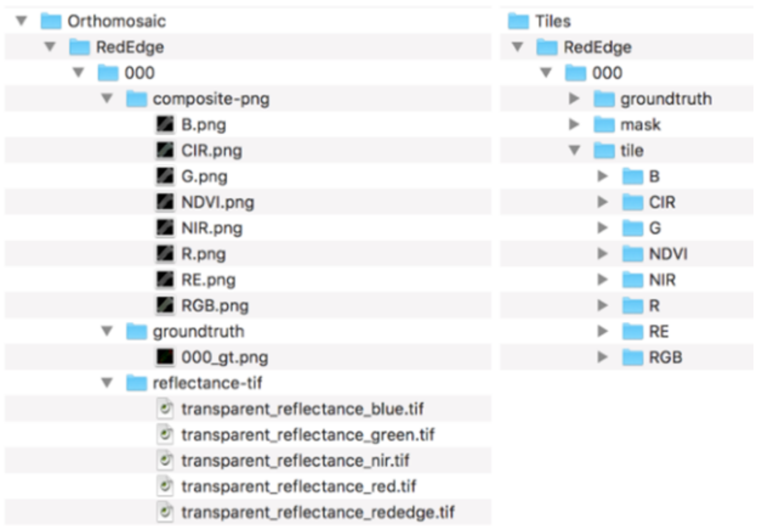
4.1.1.2 Structure of Weed Map Dataset

The full Dataset consist of 129 Directories for a total of 18746 images files. The total weight is 5.36GB.

Folder Structure:

There are two main folders, which are Orthomosaic and Tiles. Orthomosaic contains the full orthomosaic maps. Tiles contains, for each orthomosaic map, a folder containing images which represents cropped sections of the original orthomosaic map. All cropped sections together form the full map.

Both Orthomosaic and Tiles contain RedEdge and Sequoia subfolders, containing 8 Orthomosaic maps in total (5 RedEdge and 3 Sequoia). (Fig. 4.1.3)



```
▼ Orthomosaic                                    Tiles
  ▼ RedEdge                                      ▼ RedEdge
    ▼ 000                                          ▼ 000
      ▼ composite-png                                ▶ groundtruth
          B.png                                      ▶ mask
          CIR.png                                    ▼ tile
          G.png                                        ▶ B
          NDVI.png                                     ▶ CIR
          NIR.png                                      ▶ G
          R.png                                        ▶ NDVI
          RE.png                                       ▶ NIR
          RGB.png                                      ▶ R
      ▼ groundtruth                                    ▶ RE
          000_gt.png                                   ▶ RGB
      ▼ reflectance-tif
          transparent_reflectance_blue.tif
          transparent_reflectance_green.tif
          transparent_reflectance_nir.tif
          transparent_reflectance_red.tif
          transparent_reflectance_rededge.tif
```

Figure 4.1.3: *Table showing the folder structure of Weed Map Dataset. Source: [27]*

Each Orthomosaic map stand in a folder indexed 000 to 007.

Groundtruth:

The Groundtruth images are pictures of the fields where each pixel has been manually labelled. (Fig. 4.1.4) Each class is represented in a different colour:

- Background (Dirt and part and part of the image which is not the crop field): is Black (code 0)
- Crops: is Green (code 1)
- Weeds: is Red (code 2)

The Groundtruth images are utilized to evaluate the precision of each prediction.

4.2 Experiment 2

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis

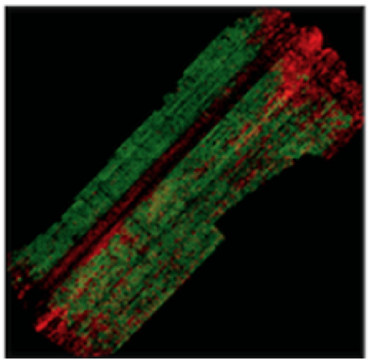


Figure 4.1.4: *Example of GroundTruth image in Weed Map Dataset. Source: [27]*

nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Chapter 5

Conclusions

The summary, as the last section of the text, includes clear and critical statements about:

- The results of the work and their importance;
- The limits of validity and the progress compared with the level of knowledge at the beginning of the work;
- The applicability of the results; and, possibly
- Recommendations for further tasks.

Bibliography

[1] Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2020.07.061. URL `https://www.sciencedirect.com/science/article/pii/S0925231220311693`.

[2] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. Cambridge University Press, 2023. URL `https://d2l.ai/chapter_hyperparameter-optimization/index.html`.

[3] Wikipedia contributors. Hyperparameter optimization — Wikipedia, the free encyclopedia, 2024. URL `https://en.wikipedia.org/w/index.php?title=Hyperparameter_optimization&oldid=1193594191`. [Online; accessed 14-May-2024].

[4] Antonin Raffin. Automatic Hyperparameter Tuning - A Visual Guide, 2023. URL `https://araffin.github.io/post/hyperparam-tuning/`.

[5] Radwa Elshawi, Mohamed Maher, and Sherif Sakr. Automated machine learning: State-of-the-art and open challenges. *arXiv preprint arXiv:1906.02287*, 2019.

[6] Sayak Paul. Hyperparameter Optimization in Machine Learning Models, 2021. URL `https://www.datacamp.com/tutorial/parameter-optimization-machine-learning-models`.

[7] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13(null):281–305, feb 2012. ISSN 1532-4435.

[8] Jason Brownlee. Hyperparameter Optimization With Random Search and Grid Search, 2020. URL `https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search/`.

[9] Shuhei Watanabe. Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance. *arXiv preprint arXiv:2304.11127*, 2023.

[10] Bernd Bischl, Martin Binder, Michel Lang, Tobias Pielok, Jakob Richter, Stefan Coors, Janek Thomas, Theresa Ullmann, Marc Becker, Anne-Laure Boulesteix, Difan Deng, and Marius Lindauer. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *WIREs Data Mining and Knowledge Discovery*, 13(2):e1484, 2023. doi: https://doi.org/10.1002/widm.1484. URL `https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.1484`.

[11] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6 (2):182–197, 2002. doi: 10.1109/4235.996017.

[12] N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. pages 312–317, 1996. doi: 10.1109/ICEC.1996.542381.

[13] scikit-learn developers. Tuning the hyper-parameters of an estimator, 2020. URL `https://scikit-learn.org/stable/modules/grid_search.html`.

[14] Zohar Karnin, Tomer Koren, and Oren Somekh. Almost optimal exploration in multi-armed bandits. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1238–1246, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL `https://proceedings.mlr.press/v28/karnin13.html`.

[15] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018. URL `http://jmlr.org/papers/v18/16-558.html`.

[16] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. In I. Dhillon, D. Papailiopoulos, and V. Sze, editors, *Proceedings of Machine Learning and Systems*, volume 2, pages 230–246, 2020. URL `https://proceedings.mlsys.org/paper_files/paper/2020/file/a06f20b349c6cf09a6b171c71b88bbfc-Paper.pdf`.

[17] Tong Yu and Hong Zhu. Hyper-parameter optimization: A review of algorithms and applications. *arXiv preprint arXiv:2003.05689*, 2020.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[19] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[20] datagy.io. Python Optuna: A Guide to Hyperparameter Optimization, 2023. URL `https://datagy.io/python-optuna/`.

[21] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995. doi: 10.1109/ICNN.1995.488968.

[22] Adrian Tam. A Gentle Introduction to Particle Swarm Optimization, 2021. URL `https://machinelearningmastery.com/a-gentle-introduction-to-particle-swarm-optimization/`.

[23] Axel Thevenot. Particle Swarm Optimization (PSO) Visually Explained, 2020. URL `https://towardsdatascience.com/particle-swarm-optimization-visually-explained-46289eeb2e14`.

[24] Dongshu Wang, Dapei Tan, and Lei Liu. Particle swarm optimization algorithm: an overview. *Soft Computing*, 22, 01 2018. doi: 10.1007/s00500-016-2474-6.

[25] Mark M Millonas. Swarms, phase transitions, and collective intelligence. *arXiv preprint adap-org/9306002*, 1993.

[26] Eberhart and Yuhui Shi. Particle swarm optimization: developments, applications and resources. In *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*, volume 1, pages 81–86 vol. 1, 2001. doi: 10.1109/CEC.2001.934374.

[27] I. Sa, M. Popovic, R. Khanna, Z. Chen, P. Lottes, F. Liebisch, J. Nieto, C. Stachniss, A. Walter, and R. Siegwart. Weedmap: A large-scale semantic weed mapping framework using aerial multispectral imaging and deep neural network for precision farming. *MDPI Remote Sensing*, 10(9), Aug 2018. doi: doi:10.3390/rs10091423.