UNLEASHING THE POWER OF HYPERPARAMETER OPTIMIZATION FOR MACHINE AND DEEP LEARNING MODELS

by © Mohamed Moustafa

A Thesis submitted to the School of Graduate Studies in partial fulfillment of the requirements of the degree of

Master of Electrical Engineering/Memorial University/Faculty of Engineering and Applied Science

Memorial University of Newfoundland

August 2023

St. John’s, Newfoundland and Labrador

Abstract

Hyperparameter optimization is a crucial aspect for improving the performance of learning models through tuning their hyperparameters. This thesis introduces three effective techniques for tuning hyperparameters in various models: PointNet CNN, regularized and non-regularized standard feedforward neural networks, SVM, and PCA. The techniques employed are random search-based tuning, hybrid random and grid search-based tuning, and hybrid random and manual search-based tuning. The random search is performed through coarse and fine-tuning stages, while hybrid approaches combined the random search's benefits with grid or manual search. These techniques achieve competitive results while optimizing computation costs regarding time and storage usage, making them valuable for state-of-the-art work. Random search and hybrid random and grid search enhance PointNet to achieve high classification accuracy that surpasses related research with an average F1-score of 93.6%, while the hybrid random and manual search reduces computational time but results in a lower accuracy of 90.97%. The random search-based tuning and hybrid random and grid search achieve an accuracy of 96.67% for the make_moons dataset. Similarly, the hyperparameter tuning techniques lead to a binary classification accuracy of 97% for the SVM model, and they identify the minimum number of PCA components and retain 99.0042% of the original dataset's variance for a set of human face images.

Keywords: deep learning; random search; hybrid random and grid search; hybrid random and manual search

Acknowledgments

I would like to express my deepest gratitude to my program's supervisors at Memorial University: Dr. Ramachandran Venkatesan and Dr. Faisal Khan, for their academic and financial support. They gave me valuable guidance and advice during my journey to acquire my degree since I started taking courses and reading and replicating state-of-the-art literature relevant to the thesis topic until conducting the proposed work in this thesis.

I would like to extend my sincere thanks to all instructors who taught me the courses I finished in this program at Memorial University: Dr. Mohamed Taleb-Berrouane, Dr. Oscar De Silva, Dr. Jonathan Anderson, and Dr. Reza Shahidi. Moreover, my warm thanks to the mentors who had guided me to practical software tools I utilized to carry out this contribution I am presenting to the research community in this document: Dr. Augustine Uhunoma and Dr. Oscar De Silva.

Eventually, I would like to acknowledge the School of Graduate Studies (SGS)'s favor of funding me to enhance my academic education and boost my career in the future. Furthermore, they provided me with the most recent articles published by prominent journals, technical writing guides, and student-assistant software programs with paid licenses.

Table of Contents

List of Tables

List of Figures

List of Abbreviations and Symbols

Adam	Adaptive Moment Estimation
AI	Artificial Intelligence
API	Application Programming Interface
ASHA	Async. Successive Halving Algorithm
BO	Bayesian Hyperparameter Optimization
CIFAR	Canadian Institute for Advanced Research
CNN	Convolutional Neural Network
CV	Computer Vision
DNN	Deep Neural Network
EPBT	Evolutionary Population-Based Training
FIRE PBT	Fast Improvement PBT
FNN	Feedforward Neural Network
GAN	Generative Adversarial Network
GP	Gaussian Process
GS	Grid Search
MLP	Multi-Layer Perceptron
NLP	Natural Language Processing
NN	Neural Network

OVA	One Versus All
OVO	One Versus One
PBT	Population-Based Training
PCA	Principal Component Analysis
RBF	Radial Basis Function
ReLU	Rectified Linear Unit
RF	Random Forest
RNN	Recurrent Neural Network
ROS	Robot Operating System
RS	Random Search
SGD	Stochastic Gradient Descent
SMAC	Sequential Model Algorithm Configuration
SMBO	Sequential Model-Based Optimization
SMO	Sequential Minimal Optimization
SVD	Singular Value Decomposition
SVM	Support Vector Machine
Tanh	Hyperbolic Tangent Activation
TPE	Tree Parzen Estimator

VGG	Visual Geometry Group
α	Learning Rate
β	Momentum Coefficient
β_1	Momentum Coefficient for Adam
β_2	RMSProb Coefficient for Adam
λ	Regularization Parameter
C	Regularization Coefficient for SVM
σ	RBF kernel's Decay Coefficient

Chapter 1

Introduction

1.1. General Background

Machine learning [1, 2, 3] is a branch of artificial intelligence (AI) that focuses on creating algorithms and models capable of learning from data and making predictions or decisions without explicit programming. It involves applying statistical and mathematical techniques to enable machines to enhance their performance on specific tasks through experience. The fundamental idea behind machine learning is to develop models that can autonomously learn patterns and correlations from vast amounts of data. These models are trained using labeled datasets, where input data is associated with desired output values. By analyzing and processing this data, machine learning algorithms can identify patterns, extract relevant features, and make accurate predictions or classifications on new, unseen data. Machine learning encompasses various algorithm types, including supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. Supervised learning [4] involves training models using labeled data to predict or classify future instances. Unsupervised learning [5] focuses on discovering hidden patterns or structures within unlabeled data. Semi-supervised learning [6] combines labeled and unlabeled data for training, while reinforcement learning [7] involves learning through interactions with an environment and receiving feedback in the form of rewards or penalties. Machine learning has diverse applications across fields, such as image and speech recognition, natural language processing, recommendation systems, fraud detection, autonomous vehicles, and medical diagnostics, among others. It has revolutionized industries

and facilitated advancements in predictive analytics, data mining, and pattern recognition. Building machine learning models entails employing techniques and algorithms such as linear regression, decision trees, support vector machines [8], neural networks, and deep learning. These models require preprocessing of data, feature engineering, and evaluation methods to ensure reliable and precise outcomes. With the continuous expansion of data availability and computational resources, machine learning is rapidly evolving and becoming a vital tool for addressing complex problems and enabling data-driven decision-making across various domains.

Deep learning [9, 10] is a branch of machine learning that focuses on training artificial neural networks to learn and make intelligent decisions, drawing inspiration from the structure and function of the human brain. The distinguishing feature of deep learning is the use of deep neural networks, which consist of multiple interconnected layers of nodes or neurons. These networks excel at learning hierarchical representations of data, enabling them to capture intricate patterns and relationships. Deep learning has achieved remarkable success in various domains, including computer vision [11], natural language processing, and speech recognition. It has revolutionized tasks like image classification, object detection, and image generation, surpassing human-level performance. Training deep learning models involves feeding them large labeled datasets and iteratively adjusting internal parameters through backpropagation. This fine-tuning process minimizes the discrepancy between predicted outputs and true labels. Deep learning models typically require significant computational resources and extensive datasets. However, advances in hardware and the availability of vast amounts of data have propelled the progress of deep learning algorithms. Convolutional neural networks (CNNs) [12, 13] are popular for image analysis, recurrent neural networks (RNNs) for sequential data processing, and generative

adversarial networks (GANs) for generating new data samples. Deep learning has transformed various industries, including healthcare, finance, autonomous vehicles, and natural language understanding. Its ability to automatically learn complex patterns from raw data has led to substantial advancements in comprehension, analysis, and generation of intricate information. Ongoing research in deep learning aims to address its limitations, such as the need for extensive labeled data and the interpretability of complex models. Overall, deep learning represents a powerful solution for solving intricate problems and has the potential to drive transformative progress across diverse domains.

Hyperparameter optimization [14, 15, 16] is a fundamental aspect of the machine learning and deep learning workflow based on tuning the learning model's hyperparameters to optimize the cross-validation cost function or accuracy. Hyperparameters, such as learning rate, regularization parameters, batch size, and network architecture, shape the behavior and performance of models. The objective of hyperparameter optimization is to discover the most effective combination of hyperparameter values that maximizes the model's performance for a given task. This process involves exploring a potentially vast and complex space of hyperparameters to identify the optimal configuration. Various optimization techniques [17] are employed for hyperparameter tuning. Grid search [18] systematically evaluates hyperparameter values across a predefined grid. The hyperparameter values in random search [19], on the other hand, are randomly sampled from the hyperparameter space, offering a more efficient alternative to grid search. Advanced techniques like Bayesian optimization [20] leverage probabilistic models to guide the search process based on previous evaluations. Genetic algorithms [21], inspired by evolutionary principles, evolve hyperparameter configurations over multiple generations. The choice of optimization technique depends on factors such as problem

characteristics, available computational resources, and the balance between exploration and exploitation of the hyperparameter space. Hyperparameter optimization significantly influences model performance, as it determines the ideal hyperparameter values that can enhance accuracy, expedite convergence, and improve generalization. It is a critical step in developing robust and high-performing machine learning and deep learning models.

1.2. Motivation

The significant impact of deep-learning and machine-learning models' tuning hyperparameters [15, 22] on their accuracy has motivated researchers and developers to develop competent hyperparameter optimization techniques. Therefore, many competitive hyperparameter optimizers are available in the state-of-the-art literature in deep learning and machine learning. Hyperparameter tuning is not required only at the development phase but at application time as well whenever data is changed, as in the case of market clustering and website recommendations. Hyperparameters are manually selected configuration settings that significantly influence a model's accuracy, efficiency, and ability to generalize to new data. The purpose of hyperparameter tuning is to find the best combination of these settings that maximizes the model's performance on a given task. This involves exploring different values or ranges of hyperparameters and evaluating their impact on the model's performance metrics. The ultimate goal is to enhance accuracy, mitigate overfitting, expedite convergence, and improve the model's ability to generalize well to unseen data. Hyperparameter tuning enables the model to adapt its behavior and tailor it to the specific characteristics of the dataset and the task at hand. Hyperparameter tuning is vital because different datasets, problem domains, and models require specific configurations for optimal performance. It enables fine-tuning of the model's behavior, resulting in improved results and more reliable predictions. In essence, hyperparameter tuning

strikes a balance between exploration (trying out different settings) and exploitation (leveraging promising settings) of the hyperparameter space, leading to models that achieve optimal performance and robust generalization.

1.3. Objective

This thesis implements three effective techniques for tuning hyperparameters in five diverse machine learning and deep learning models. The three techniques used are:

- Random search-based tuning.
- Hybrid random and grid search-based tuning.
- Hybrid random and manual search-based tuning.

These techniques are applied to models such as PointNet CNN, regularized and non-regularized standard feedforward neural networks (FNNs), support vector machine (SVM), and principal component analysis (PCA). The random search-based tuning involves performing coarse and fine-tuning stages to optimize the hyperparameters. The hybrid random and grid search combines the benefits of random search and grid search to find optimal hyperparameter values efficiently. The hybrid random and manual search utilizes the insights gained from random search as prior knowledge to guide the manual search process.

These techniques aim to enhance the visual classification accuracy of convolutional deep neural networks, specifically PointNet. The hyperparameters targeted for tuning are the mini-batch size of stochastic gradient descent (SGD), momentum, and learning rate. The objective is to prevent overfitting, improve generalization, and achieve low variance in the model's performance. The evaluation of the tuned models was conducted using pool and test datasets, ensuring the models' ability to accurately classify unseen data from different distributions.

Overall, these hyperparameter tuning techniques aimed to optimize the performance and generalization capabilities of the machine learning and deep learning models, specifically in the context of visual classification tasks. Furthermore, the three tuning techniques successfully identified a model for a standard feedforward neural network (FNN) that achieved a high binary classification accuracy by tuning the regularization parameter (λ), learning rate (α), and momentum (β). Additionally, the three hyperparameter tuning techniques were effective in enabling the SVM model to achieve a high binary classification by the regularization parameter (C) and Gaussian radial basis function (RBF) kernel's decay coefficient (σ). Moreover, these techniques successfully determined the minimum number of PCA components that reduced the dimensionality of the human faces dataset in a computer vision application, keeping possession of most of its original variance before reduction. Overall, the hyperparameter tuning techniques showcased their ability to enhance the performance and accuracy of various models across different datasets and tasks.

1.4. Contribution

The work described in this thesis strives to improve the learning model's ability to generalize to an unseen dataset, either the cross-validation or test dataset, besides the training dataset. Random search, hybrid random and grid search, and hybrid random and manual search tuning techniques were utilized to optimize the cost function and generalization of some machine and deep learning models such as PointNet CNN, regularized and non-regularized standard feedforward neural networks (FNNs), support vector machine (SVM), and principal component analysis (PCA).

To the best knowledge of the thesis author that merging random search with grid search or with manual search has not been reported for tuning the hyperparameters of PointNet CNN, FNN, SVM, and PCA in the literature of machine and deep learning.

The main contributions of the thesis can be summarized in the following points:

- Using the hybrid random and grid search to optimize the grid search by taking advantage of random search to determine the efficient hyperparameter space. This optimized grid search avoids wasted time in tuning the hyperparameters with inefficient values that would never optimize the learning model's accuracy.
- Using the hybrid random and manual search to optimize the manual search by taking advantage of random search at having prior knowledge about the productive hyperparameter space. This optimized manual search avoids wasted time at tuning the hyperparameters with inefficient values that would never optimize the learning model's accuracy.
- Tuning the hyperparameters of PointNet CNN through three stages is unprecedented. Particularly, coarse and fine random search or hybrid random and grid search, or hybrid random and manual search are conducted using a cross-validation dataset. In addition, a test dataset-based evaluation was performed to ensure PointNet's capability to generalize to cross-validation and testing datasets. Accordingly, the learning model has a high potential to generalize to other unseen datasets, such as the application dataset.
- Proposing model-independent tuning techniques that can be applied to all deep CNN networks and machine learning models.

- Obtaining high classification accuracy by applying the three tuning techniques on standard FNN and SVM, outperforming the state-of-the-art work that uses the same dataset (make_moons of Sklearn library).
- Applying the three tuning techniques on a supervised learning model (PCA) for the sake of dimensionality reduction of image datasets.
- Obtaining high classification accuracy without the high memory usage and long computation time required by the other iterative hyperparameter tuning techniques like Bayesian, gradient descent, and population-based training techniques. The iterative method is effective when the CNN model has a high-dimensional hyperparameter space. This thesis work tunes only at most three hyperparameters.

The rest of the thesis is structured as follows:

Chapter 2 provides an overview of the most common hyperparameter tuning techniques and some machine and deep learning models. The methodology for implementing the proposed hyperparameter tuning techniques is described in detail in Chapter 3. Chapter 4 shows and discusses the results in detail. Eventually, Chapter 5 demonstrates this thesis' main conclusions and suggests recommendations that should be followed in the future to present a more efficient contribution.

Chapter 2

Literature Search

This chapter shows various hyperparameter tuning techniques, including used and unused techniques in this thesis, that can be applied to different machine and deep learning models. Furthermore, it presents a collection of the most common learning models used in this industry of deep and machine learning development; some are machine learning models, and others are deep learning ones. The thesis applies hyperparameter tuning techniques on five various learning models out of all learning models shown in this chapter. Moreover, it demonstrates the exploited datasets and their organization.

2.1. Hyperparameter Tuning Techniques

All iterative hyperparameter tuning techniques [23, 24], such as gradient-based, Bayesian, and population-based training (PBT), require a considerable memory size and high computation cost. They are worthwhile and superior over other tuning techniques only when the learning model has a high-dimensional hyperparameter space. While at most three hyperparameters are tuned in the different experiments of this thesis, iterative tuning techniques are not a good choice. Furthermore, the pure manual and grid search for three hyperparameters needs much computation time. Therefore, the random search-based coarse-to-fine, hybrid random and grid search, and hybrid random and manual search are the most suited hyperparameter tuning techniques for this thesis' work as they can give comparable or even better results within much less computation time and memory usage as discussed in Chapter 3 in detail. Here are the most common hyperparameter tuning techniques:

2.1.1. Manual Search (Babysitting One Model or Pandas Raising Approach)

Manual search [15], sometimes called Babysitting one model, needs deep experience and solid prior knowledge about the learning model to estimate the optimum hyperparameters' values that would maximize the model's accuracy in a limited time. Usually, this approach [14, 25] is employed when there is a scarcity of computing resources, such as limited CPUs and GPUs, despite having a large dataset. As a result, only one model or a small number of models can be trained at a time. In this scenario, the model needs to be closely monitored and guided throughout the training process. Initially, the model's parameters are set randomly on the first day, and the training begins. The progress is observed gradually, perhaps by tracing the learning curve, the cost function J, the cross-validation error on the dataset, or any other relevant metric, as shown in Figure 1. By the end of the first day, if it appears that the model is learning well, one might decide to increase the learning rate slightly and assess its performance. If the model improves, the progress continues on the second day. This iterative process of monitoring and adjusting continues, with regular check-ins on subsequent days. Sometimes, it may be necessary to decrease the learning rate if it was set too high the previous day. This pattern repeats day after day as the model is trained over a span of multiple days or weeks. This strategy involves closely attending to one model, observing its progress, and making subtle adjustments to the learning rate. It is a common practice when there is insufficient computational capacity to train multiple models simultaneously. In other words, the concept of the "Raising Pandas" approach is analogous to adult pandas taking care of and nurturing a single baby panda at a time. Similarly, in this context, it refers to focusing on training one model at a time. For instance, if one belongs to a smaller company or is at the initial stages of their AI journey, they might have limited infrastructure and computational resources to train, evaluate, and fine-tune only one model at a

time. This approach is particularly beneficial when a software company has a single data scientist, when there are few use cases for deep learning within the software organization, or when the model can be adequately trained with a moderate amount of data (less than 100 TB).

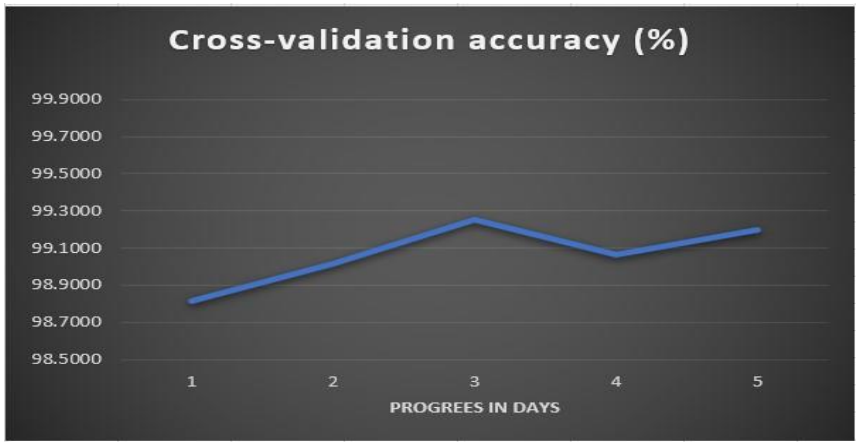

Figure 1. Manual Search (Babysitting One Model)

It is impractical to use this approach when the model has many hyperparameters or in the case of lacking the prior knowledge about the learning model. Thus, a novel technique, offered in Chapter 3, merges a random search with a manual search.

2.1.2. Grid Search (GS)

Grid search [26] is a brute-force or exhaustive search that covers all the possible combinations through predefined sets of hyperparameters' values, as shown in Figure 2. Each hyperparameter has a manual discretized set formed from its continuous range. To guide the grid search, a performance metric, usually determined through cross-validation on the training set or evaluation on a separate validation set, is employed. The grid search procedures can be summarized in three points as follows:

a) Initially, start with a large search space and step size.

b) Based on the previous results of successful hyperparameter configurations, narrow down the search space and step size.

c) Repeat step 2 multiple times until an optimum configuration is reached.

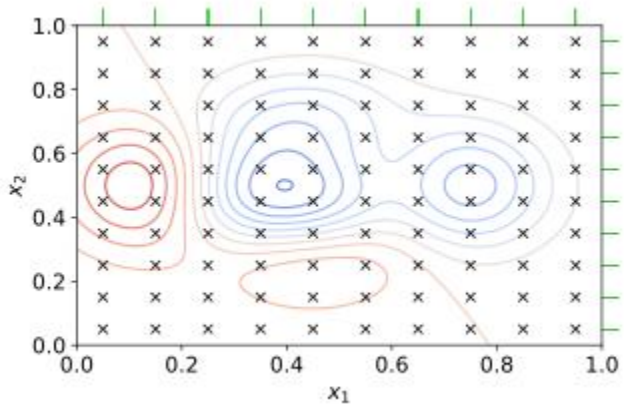

Figure 2. Grid Search. Reproduced From [27]

Hsu et al. in [28] use a grid search-based coarse-to-fine tuning technique to tune two hyperparameters in an SVM classifier and cross-validation accuracy as an evaluation metric. It enhances the model's classification accuracy but requires much computation time. A random search-based coarse-to-fine-tuning technique can obtain the same accuracy or even better. Grid search is a suitable choice when the model has low-dimensional hyperparameter space: at most, three hyperparameters. Chicco gives ten tips for machine learning in biomedical applications. The 6th tip shows the impact of tuning the number of clusters (k) as the single hyperparameter on the classification accuracy of an unsupervised learning clustering model. It is demonstrated that grid search is the best tuning technique for that application. In [21], Liashchynskyi et al. utilize grid search to tune two hyperparameters in their designed CNN image classifier fed by CIFAR-10 [29] dataset split as follows: 50,000 images for training and 10,000 for testing. They obtained a classification accuracy of around 83% using a grid search and 85.8% using a random

search within a shorter time. In [18], Belete et al. exploit the grid search to tune three hyperparameters in an SVM classifier fed by a health statistical dataset, achieving a classification accuracy of 87.4% in a very long time due to the relatively high dimensionality of hyperparameters space (> 3 hyperparameters). Using the grid search technique, Nugraha et al. in [30] tune the hyperparameters of seven different machine-learning classification algorithms to select the best-fit model based on cross-validation classification accuracy.

Grid is relatively straightforward to implement and can be parallelized. Nevertheless, its main drawback lies in its inefficiency when dealing with high-dimensional hyperparameter configuration spaces. As the number of hyperparameters increases, the number of evaluations grows exponentially, leading to what is known as the "curse of dimensionality." For GS, if there are k parameters, each with n distinct values, the computational complexity increases exponentially at a rate of O(n^k). Therefore, GS is only effective as a hyperparameter optimization method when the configuration space is small. Thus, a novel technique, offered in Chapter 3, merges random search with grid search to speed up the tuning and improve its efficacy.

2.1.3. Random Search (RS)

Random search hyperparameter tuning [31] is similar to grid search, but it randomly selects some combinations through the predefined continuous scales of hyperparameters without discretizing them. Random Search is a technique that replaces the exhaustive and systematic enumeration of all possible combinations with a random selection process, as shown in Figure 3. This approach [15] can be easily applied to discrete scenarios mentioned earlier, but it also extends to continuous and mixed spaces. It can be carried out in one stage or two stages; a two-stage random search is called random search-based coarse-to-fine hyperparameter tuning to

focus on the hyperparameters' values that enhance the accuracy of learning algorithms and discard others. Two-stage random search outperforms grid search with much less computation time when the learning model has more than two hyperparameters, so it is used in the work proposed in this thesis that tunes at most three hyperparameters in all experiments. It is superior over grid search due to the fact that only some hyperparameters have a significant impact on the learning model accuracy. This situation indicates that the optimization problem has a low intrinsic dimensionality. Furthermore, it allows incorporating prior knowledge by specifying the distribution from which to sample the hyperparameters. Despite its simplicity, Random Search remains an important benchmark for evaluating the performance of new hyperparameter optimization methods. Furthermore, the computational complexity remains the same with increasing the number of tuned hyperparameters at a rate of O(n). RS still involves a significant number of unnecessary evaluations as it does not leverage information from previously well-performing areas. Therefore, iterative tuning techniques, discussed in the following subsections, are preferred over random search to make full use of the previously efficient regions.

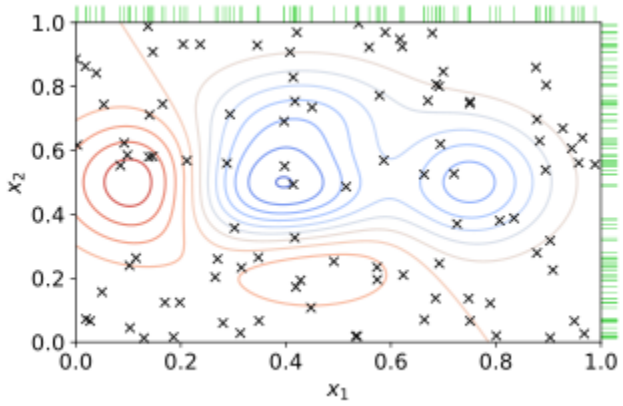

Figure 3. Random Search. Reproduced from [27]

Rojas et al. in [32] show that the one-stage random search gives results better than the grid search in tuning the number of hidden layers, the number of neurons per layer, and the learning rate of a feed-forward neural network fed by online real-time electricity energy readings to forecast the power consumption. In [33], Andonie et al. tune the number of convolution layers, the number of fully-connected layers, the number of output filters for each convolution layer, and the number of neurons for each fully connected layer of a CNN classifier fed by a CIFAR-10 dataset using a random search for 300 combinations of hyperparameters' values. Navon et al. in [19] present an analytical optimization in random search technique by putting a lower bound to the error of CNN's classification; once this error threshold is met, tuning should be stopped early as there is no possible further improvement in CNN's accuracy. It is demonstrated empirically and theoretically in [34] that random hyperparameters search outperforms grid search and manual search within a shorter computation time. As a matter of fact, grid search picks out a tremendous number of values representing all possible combinations between the values of manually predefined discretized scales for tuned hyperparameters. In contrast, random search selects random values within a manually predetermined continuous scale for each hyperparameter. Hence, the superiority of random search over grid search is more evident when the model uses more than two hyperparameters. This superiority of random search over grid search refers to the fact that hyperparameters do not have the same importance in optimizing the performance of deep NN, but only a few hyperparameters dominate. The difference between the two techniques is visualized in Figure 4. Mantovani et al. in [35] compare random search with genetic algorithm (GA) [36] and grid search at tuning the hyperparameters of an SVM classifier. The results clearly show the superiority of random search with a lower computation cost.

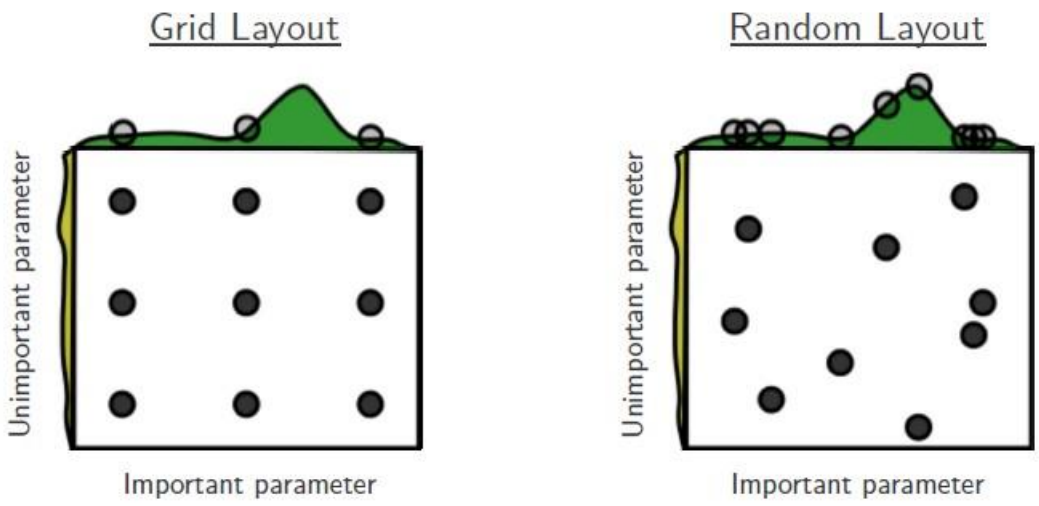

Figure 4. Grid Search vs. Random Search. Reproduced from [36]

This technique is utilized in all the experiments of the thesis due to simplicity and efficiency with low computation cost and in a limited time, as shown in Chapter 3.

2.1.4. Population-Based Training (PBT)

Population-based training (PBT) [37, 38] hyperparameter tuning is an iterative tuning technique and a prominent category of metaheuristic algorithms, inspired by genetic algorithms (GA), as each agent in the population can exploit the information of the rest of the agents. Each agent is a model with unique values of weights and hyperparameters; therefore, multiple models are trained asynchronously in parallel. Thus, both the learning model's parameters and hyperparameters are iteratively optimized jointly. The hyperparameters of each agent are randomly initialized from the predefined distributions of hyperparameters, similar to the random search method mentioned above. During training iterations, optimization of the learning model's weights and hyperparameters is carried out through exploitation and exploration processes. Therefore, PBT and its variations are considered adaptive methods since they dynamically update hyperparameters during model training. In contrast, non-adaptive methods employ a

suboptimal strategy of using fixed hyperparameter settings throughout the entire training process. The exploitation process is performed at every single training iteration by the truncation selection method. In the truncation selection method, each agent is ranked based on performance while iterating on the population's agents. If the agent of the current exploitation iteration is among the bottom 20% in ranking, it should then take a copy of the weights and hyperparameters of a randomly selected agent from the top 20%. Moreover, the exploration process is conducted periodically after a fixed number of training epochs. Perturbing and resampling are two separate ways to achieve the exploration process. For the perturbing way, each hyperparameter's value is randomly altered by one of two factors; one factor is greater than one, and the other is less than one. On the other hand, the resampling way is based on changing hyperparameters' values by randomly selecting numbers from the predefined distributions of hyperparameters. Figure 5 shows the mechanism of exploitation and exploration processes in PBT.

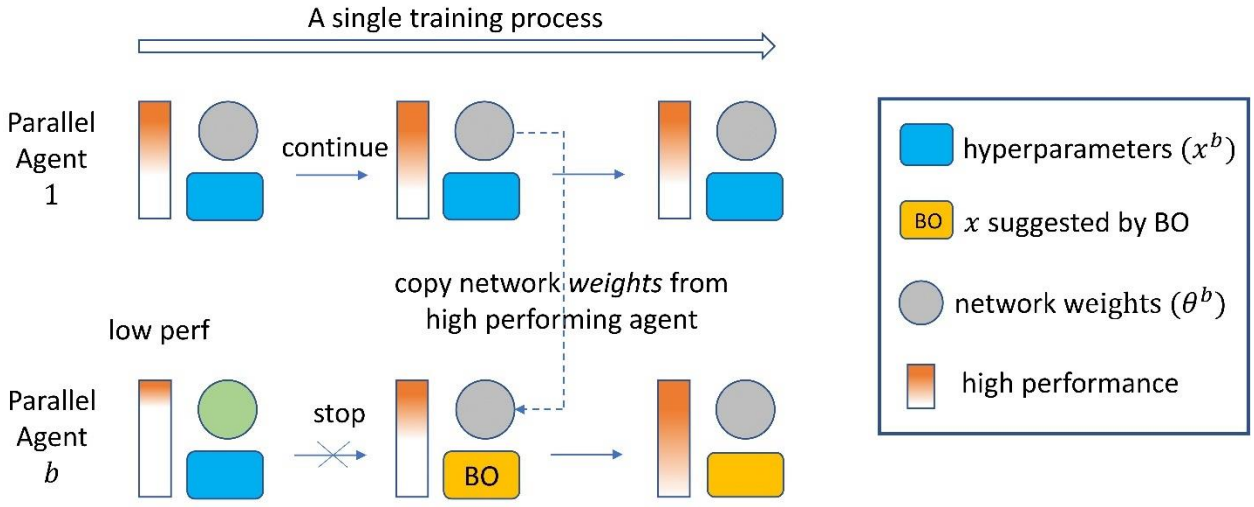

Figure 5. Population-Based Training (PBT). Reproduced From [39]

Jaderberg et al. in [38] apply PBT to a reinforcement learning algorithm and two supervised deep learning networks to show its effectiveness in tuning the hyperparameters of each model.

Notably, PBT requires a high computation cost because the deep supervised networks in that article were trained in 0.5 to 1 million iterations. In [40], Liang et al. propose an evolutionary population-based training (EPBT) method to tune the learning rate and momentum of a deep neural network (DNN) fed once by the CIFAR-10 dataset to obtain better results than the regular PBT. Dalibard et al. in [41] present an optimization for PBT by developing fast improvement PBT (FIRE PBT) that resolves the bias of PBT towards the population's models that have a short-term improvement in performance, which may lead to poor long-term accuracy. In [42], Li et al. apply a network's architecture-independent PBT on a state-of-the-art WaveNet generative model for human voice synthesis. In [43], Hassan et al. use a genetic algorithm (GA) to tune the hyperparameters, including the kernel size, of a CNN fed by chest X-ray images for detecting Covid-19 infection, achieving a classification accuracy of 98.48%.

2.1.5. Gradient-Based Hyperparameter Tuning Technique

Basically, the principle of the gradient-based hyperparameter tuning technique [44] is computing the partial derivatives of the objective optimization function, like cross-validation accuracy, with respect to each hyperparameter in the learning models. After that, the hyperparameters' values are iteratively optimized using gradient descent, as shown in Figure 6. Despite their faster convergence speed compared to other methods, gradient-based algorithms have limitations. According to [15], they can only optimize continuous hyperparameters since other types, such as categorical hyperparameters, lack gradient directions. Additionally, these algorithms are efficient primarily for convex functions, as non-convex functions may lead to reaching local optima instead of the global optimum. The time complexity of gradient-based algorithms for optimizing k hyperparameters, each has n distinct values, is O (n^k).

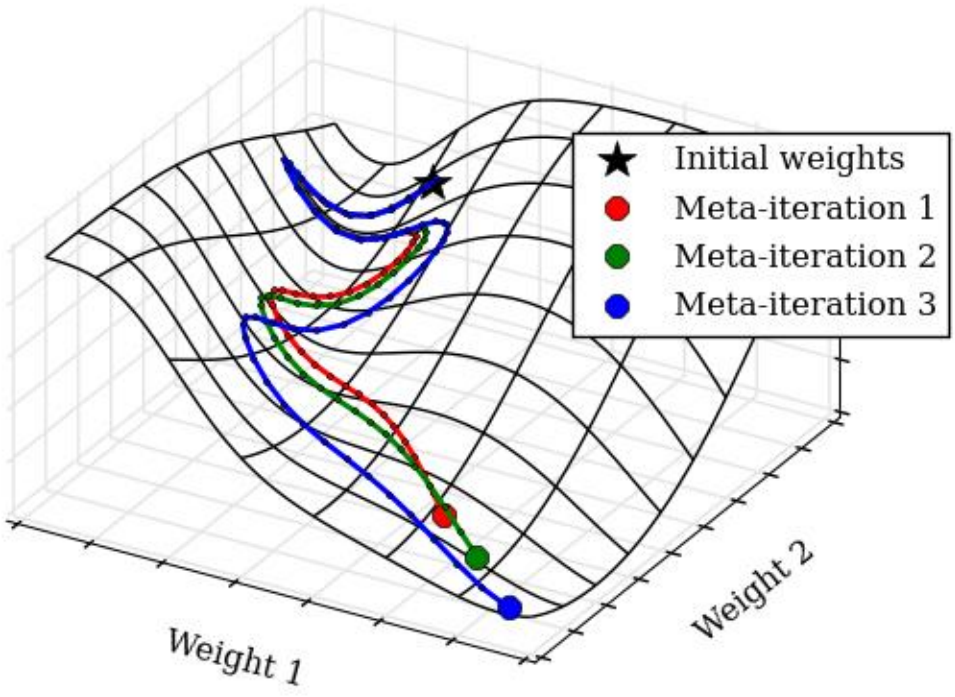

Figure 6. Gradient-Based Hyperparameter Optimization. Reproduced From [45]

Maclaurin et al. in [45] use the backpropagation concept to calculate the gradients of cross-validation accuracy with respect to each hyperparameter in the deep neural network. Bakhteev et al. in [46] compare gradient-based hyperparameter tuning with the random research technique concluding that it outperforms the random search only when the learning model has a high-dimensional hyperparameter space. In [47], Franceschi et al. tune the hyperparameters of a recurrent neural network (RNN) fed by a CIFAR-10 dataset using the forward and reverse modes of the gradient-based hyperparameter tuning. In [48], Micaelli et al. offer a novel algorithm to overcome memory scaling caused by the large-horizon (many gradient steps) gradient descent while searching the hyperparameters of a deep neural network fed by a CIFAR-10 dataset.

2.1.6. Bayesian Hyperparameter Optimization (BO)

Bayesian hyperparameter tuning [49] is an iterative technique that constructs a probabilistic hypothesis that maps the hyperparameters' values to the learning model's cross-validation accuracy. The hyperparameters with a high probability of obtaining optimum performance are exploited during training. Furthermore, the hyperparameters with an uncertain likelihood of getting optimum performance are explored periodically after every specific number of training epochs. Unlike grid search (GS) and random search (RS), BO utilizes past results to determine future evaluation points. It consists of two key components: a surrogate model and an acquisition function. The surrogate model aims to fit all the observed points to the objective function. Using the predictive distribution of this probabilistic surrogate model, the acquisition function balances exploration and exploitation. Common surrogate models used in BO include the Gaussian process (GP) [50], random forest (RF) [51], and the tree Parzen estimator (TPE) [52]. Exploration involves sampling instances from unexplored regions, while exploitation focuses on sampling from currently promising regions likely to contain the global optimum based on the posterior distribution. BO algorithms compromise between exploration and exploitation to identify the most likely optimal regions while avoiding neglecting better configurations in unexplored areas, as shown in Figure 7. However, BO belongs to sequential methods that are challenging to parallelize since they rely on previous evaluations.

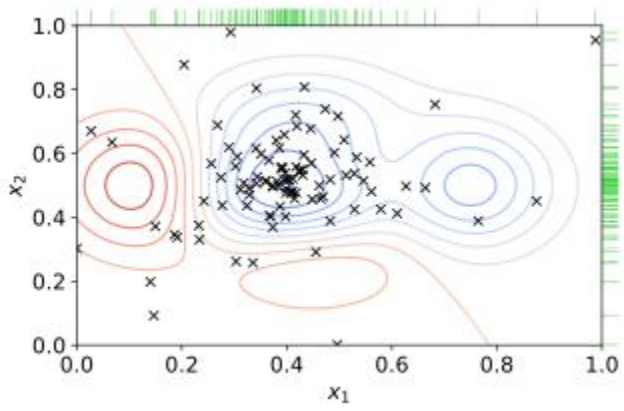

Figure 7. Bayesian-Based Hyperparameter Optimization. Reproduced From [27]

Snoek et al. in [53] tune nine hyperparameters of a CNN fed by a CIFAR dataset using Bayesian hyperparameter tuning. In [20], Masum et al. utilize Bayesian tuning to tune the hyperparameters of a deep neural network fed by an NSL-KDD dataset [54] for an intrusion detection application.

2.2. Hybrid Hyperparameter Tuning Techniques

Here are some hybrid hyperparameter tuning approaches:

1. Grid Search with Random Search:
    - Combine a coarse grid search with random search for a more efficient and effective hyperparameter search. Start with a broad grid search to identify promising regions, then apply random search within those regions.
2. Bayesian Optimization with Grid Search:
    - Use Bayesian optimization to explore the search space efficiently while incorporating domain knowledge. Initially, perform a grid search to identify a

rough region of interest and then apply Bayesian optimization for fine-tuning within that region.

3. Genetic Algorithms with Random Search:

    - Combine genetic algorithms with random search to evolve hyperparameter configurations. Genetic algorithms can create and mutate configurations, while random search can add diversity and exploration.

4. AutoML and Manual Tuning:

    - Employ automated machine learning (AutoML) tools to quickly generate and evaluate hyperparameter configurations. Then, fine-tune the best-performing configurations manually based on domain knowledge.

5. Hyperband with Bayesian Optimization:

    - Use the Hyperband algorithm to efficiently allocate resources among different configurations and combine it with Bayesian optimization for exploration within each allocation.

6. Successive Halving with TPE:

    - Apply Successive Halving to progressively allocate resources to a subset of configurations, and use Tree-structured Parzen Estimators (TPE) to select promising configurations within each round.

7. Ensembling Multiple Optimizers:

- Combine multiple hyperparameter optimization algorithms, such as grid search, random search, Bayesian optimization, and genetic algorithms, and use an ensemble to choose the final hyperparameter configuration.

8. Adaptive Strategies:

   - Develop hybrid strategies that adapt the optimization approach based on the performance of the model during training. For example, start with grid search, and if no improvement is observed, switch to Bayesian optimization.

9. Reinforcement Learning:

   - Utilize reinforcement learning techniques to optimize hyperparameters. Reinforcement learning agents can learn from past experiments to guide the search process efficiently.

2.3. Getting the Most out of the Parallel Programming for Hyperparameter Optimization

Ng presents in [25, 55] the Caviar approach as a hyperparameter tuning approach. The principle of this approach is training many models in parallel and selecting the best-fit one that has the best results based on the selected evaluation metric, either a cross-validation dataset or a test dataset. Hence, it is named after Caviar, which lays on so many eggs. This approach is applied to all hyperparameter tuning techniques except for manual hyperparameter search, called the "Babysitting One Model" or "Pandas Raising" approach. ASHA (Asynchronous Successive Halving Algorithm) [56] is utilized to apply the Caviar approach efficiently, getting the most out of the computation resources based on parallel programming. It is a widely used hyperparameter tuning algorithm that combines the principles of successive halving and asynchronous execution.

It is specifically designed to optimize resource allocation in distributed environments, where multiple trials can be conducted simultaneously.

The ASHA algorithm can be summarized as follows:

- Initial Configurations: ASHA begins by defining an initial set of hyperparameter configurations, also known as "arms." Each configuration is evaluated using a small fraction of the available resources, such as training epochs, to obtain an initial estimate of its performance.
- Successive Halving: In the successive halving phase, the algorithm iteratively divides the available resources among the surviving configurations. Only the top-performing configurations are retained at each stage, while the rest are discarded. The resource allocation is increased for the retained configurations, allowing them to receive more resources for further evaluation.
- Asynchronous Execution: ASHA introduces the concept of asynchronous execution, enabling concurrent evaluation of multiple configurations. This allows for parallel assessment of different configurations, significantly accelerating the optimization process.
- Early Stopping: ASHA incorporates early stopping criteria to terminate poorly performing configurations early on. If a configuration's performance lags significantly behind the best-performing configuration at the same resource allocation, it is terminated, freeing up resources for more promising alternatives.
- Iterative Process: ASHA continues the successive halving and asynchronous execution steps until only one configuration remains. The final configuration is considered the best solution found by the algorithm. ASHA excels in situations where training a machine

learning model with various hyperparameter configurations demands substantial time or computational resources. By dynamically allocating resources to the most promising configurations and leveraging parallel execution, ASHA efficiently explores the hyperparameter space to identify high-performing configurations.

Libraries and frameworks like Ray Tune and Optuna provide implementations of ASHA for hyperparameter tuning. These integrations seamlessly incorporate ASHA into their optimization frameworks, allowing users to harness its benefits without the need to implement the algorithm from scratch.

2.4. The State-of-the-Art Hyperparameter Optimization Frameworks

This section presents the most common hyperparameter optimization frameworks [15, 57] used in the industry, which were developed particularly to optimize the performance of iterative hyperparameter optimization techniques such as the Bayesian technique and population-based training (PBT). Therefore, this thesis does not employ these frameworks because all experiments deal with at most three hyperparameters, and there is no need for exploiting iterative tuning techniques and their associated frameworks. As a matter of fact, when the learning model has vast hyperparameter space, the iterative techniques are efficient, and their heavy computation load is worthwhile. Here are the most popular hyperparameter optimization frameworks utilized in the industry are:

- Optuna.
- Hyperopt.
- SMAC (Sequential Model-based Algorithm Configuration).
- Ax.

- Nevergrad.

Optuna [58] is a well-regarded framework for hyperparameter optimization developed by Preferred Networks, Inc. It offers a versatile and efficient solution for searching for the best hyperparameters for machine learning models. By employing sequential model-based optimization (SMBO), Optuna effectively explores the hyperparameter space to identify the optimal set of values. Here's a concise explanation of how Optuna operates for hyperparameter optimization [59]:

- Define the Objective Function: Begin by defining the objective function, which represents the function to be optimized. This function takes hyperparameters as input and produces a score or loss value that you aim to minimize or maximize. The objective function can be any Python function that assesses the model's performance using a specific set of hyperparameters.

- Define the Search Space: Specify the search space for the hyperparameters you wish to optimize. Optuna supports various types of hyperparameters, including continuous, discrete, and categorical variables. You can define the range of values that each hyperparameter can take within the search space.

- Create an Optuna Study: A study in Optuna represents a single optimization run. It keeps track of the hyperparameter configurations and their respective scores throughout the optimization process. You can create a study object using optuna.create_study() and specify the optimization direction (minimize or maximize) based on your objective function.

- Implement the Objective Function: Within the objective function, sample the hyperparameters using Optuna's suggested APIs. Optuna provides different methods to sample hyperparameters, such as suggest_uniform() for continuous variables, suggest_categorical() for categorical variables, and suggest_int() for integer variables. These sampled hyperparameters can be accessed within the objective function to train and evaluate the machine-learning model.

- Optimize the Objective Function: Initiate the optimization process by calling the study.optimize() method. Optuna will iteratively explore the hyperparameter space, suggesting new configurations based on past results and their scores. Advanced algorithms like Tree-structured Parzen Estimator (TPE) or Gaussian Process model the relationship between hyperparameters and scores.

- Access the Best Hyperparameters: Once the optimization process concludes, you can access the best set of hyperparameters and their corresponding score using the study.best_params and study.best_value attributes, respectively. These represent the optimal hyperparameters discovered during the optimization process.

Optuna also offers additional features like early stopping based on pruning, parallel execution of trials, and integration with popular machine learning frameworks like TensorFlow, PyTorch, and scikit-learn. Indeed, it is a highly effective framework for hyperparameter optimization, significantly enhancing the efficiency and efficacy of finding the optimal hyperparameters for machine-learning models.

Hyperopt [60] is a Python library developed by James Bergstra and collaborators. It utilizes the Tree-structured Parzen Estimator (TPE) algorithm to optimize hyperparameters through

Bayesian optimization. Hyperopt is flexible and allows the use of custom search spaces and optimization algorithms.

SMAC [61] is an optimization framework that incorporates surrogate models and evolutionary algorithms. It was developed by researchers at the University of Freiburg and is known for its efficient performance on complex and expensive-to-evaluate objective functions.

Ax [62] is a hyperparameter optimization library developed by Facebook AI, it uses Bayesian optimization to efficiently search for optimal hyperparameters. It is designed to be easy to use and provides built-in support for parallel evaluations.

Nevergrad [63] is also developed by Facebook AI; it is an optimization platform that includes a wide range of optimization algorithms, including Bayesian optimization and genetic algorithms. It is designed to be modular and versatile, making it suitable for a variety of optimization problems, including hyperparameter tuning.

2.5. Learning Models

In this chapter, a set of hyperparameter tuning techniques is presented, encompassing both utilized and unused methods in this thesis. These techniques are applicable to diverse machine and deep learning models. Additionally, the chapter showcases a comprehensive selection of commonly used models in the deep and machine learning field. Some of these models belong to traditional machine learning, while others are part of the realm of deep learning. Within the thesis, hyperparameter tuning techniques are specifically applied to five distinct learning models covering almost the most common types of learning models to perform 3D image semantic segmentation, binary classification, and dataset visualization. They are chosen from the broader

set of models introduced in this chapter, including supervised and unsupervised machine learning and deep convolutional and feedforward neural network.

2.5.1. PointNet Convolutional Neural Network (Deep Learning Model).

This subsection presents an overview of PointNet architecture and its hyperparameters and regularization approaches.

2.5.1.1. PointNet Structure

A PointNet deep neural network in [64] is selected among the five learning models tuned in the thesis. It performs 3D semantic segmentation and classification for pipes and valves introduced to CNN as a point cloud dataset. The source code of PointNet is available to the research community as an open source [65]. This CNN is a modified version of the original PointNet-based visual classifier in [66], whose architecture is shown in Figure 8. Based on this diagram, the PointNet CNN performs two main tasks: multi-class classification and 3D semantic segmentation. Notably, CNN classifies the objects in the input point clouds and uses their global features to perform pixel-wise classification, commonly called semantic segmentation. This CNN is mainly based upon convolutional deep neural networks [10] consisting of convolutional and maximum pooling layers. Convolutional layers are the basis of multi-layer perceptron (mlp) shown in Figure 8. CNNs act as feature extractors like VGG, ResNet, PointNet, and Inception. These extractors in VGGNet [13] produce features with a much smaller width and height than the input image based on padding and stride sizes but have a much bigger depth based upon the number of convolutional filters in the single convolutional layers. Nevertheless, PointNet CNN deals with input point clouds, so the convolutional layers (mlps) only increase the depth of points data representation. The maximum pooling layers generate global features invariant with small

translations in pixels of input points. Thus, this property of max pooling filters makes PointNet robust at object recognition and classification.

A Softmax layer is used as an output layer to apply cross-entropy loss function during training and cross-validation for multi-class classification purposes. This classifier has three classes: pipes, valves, and background. The Softmax layer has a set of activation units as many as the number of classified classes. These activation units give scores to the classes; the class with the highest score takes the value One, and the rest take zeros. These scores are updated during training until the cross-entropy loss function converges to a minimal value. [67] exhibits detailed information about the analytical analysis and applications of the cross-entropy loss function.

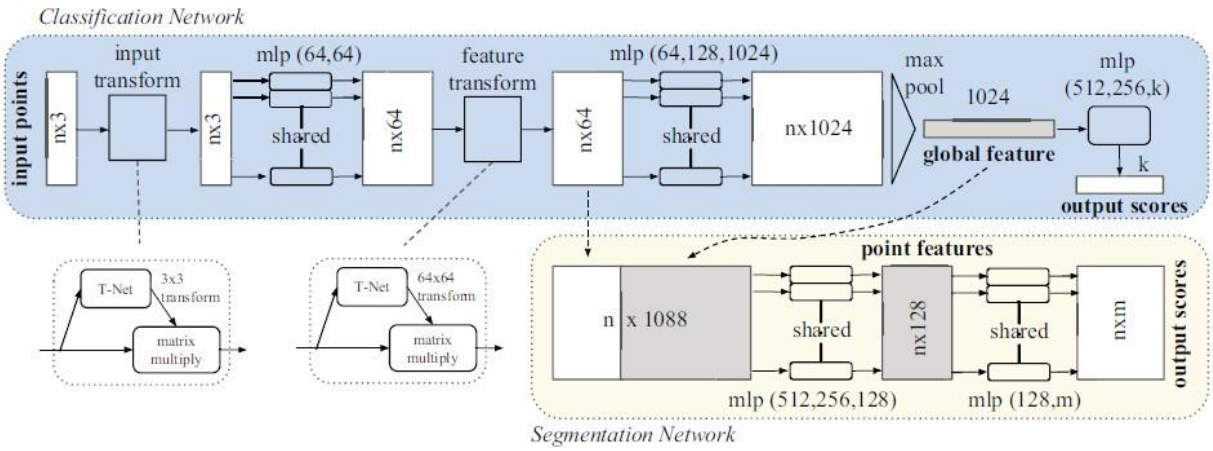

Figure 8. PointNet Architecture. Reproduced from [66]

2.5.1.2. Hyperparameters and Regularization Techniques of PointNet

The PointNet model has the following hyperparameters: number of convolutional and max pooling layers, block-to-stride ratio, number of points for each training example, mini-batch size, momentum (β_1), RMSProb (β_2), learning rate decay and learning rate (α).

The learning model is trained using stochastic gradient descent (SGD) [68], so the dataset samples are divided into several mini-batches depending on the mini-batch size selected by the developer. SGD is faster than the regular gradient descent, in which the whole dataset is trained in one big batch; nonetheless, SGD makes the model's weights converge to less optimal minima. Consequently, mini-batch size is a critical hyperparameter that significantly affects the deviation of classification accuracy between training and validation; in other words, it has a remarkable impact on variance. The smaller the mini-batch size is, the less variance and generalization gap the model has. Hence, [69] gives a practical recommendation to researchers and developers to select a mini-batch with a size of 16 or 32 dataset samples. This size would ensure that model can generalize to diverse cross-validation and test datasets and not overfit the training dataset.

Compared to batch gradient descent, SGD slows down convergence to optimal minima because the value of cross-entropy loss oscillates upwards and downwards, as shown in Figure 9. Accordingly, the hyperparameter, momentum (β_1), is exploited to damp this oscillation of SGD's cross-entropy loss to accelerate its convergence, as shown in Figure 9. Besides, it is proven empirically in [70] that the momentum supports the model's generalization, which is significant in getting the remarkable results obtained in this thesis work. The momentum's recommended value by Ng in [14] is from 0.9 to 0.999 for various deep neural networks.

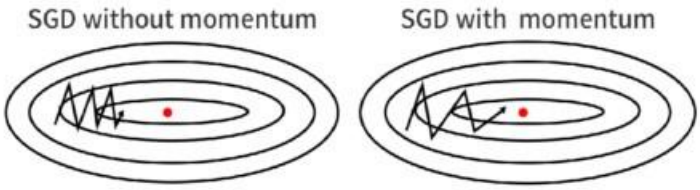

Figure 9. SGD vs. SGD +Momentum. Reproduced from [69]

Indeed, PointNet does not use a stand-alone momentum to optimize the SGD, but an Adaptive Moment Estimation (Adam) optimizer [71] is utilized to speed up learning by damping the fluctuation in the values of the cross-entropy loss function during training due to the noisy behavior of mini-batch gradient descent in this classifier. In this way, the loss function can converge to a minimal value in a much shorter time. It [72] is a dynamic optimization algorithm that incorporates both momentum and scaling. It combines the advantages of RMSProp [73] and SGD with Momentum [74]. This optimizer is specifically designed to be effective for non-stationary objectives and challenges involving gradients that are extremely noisy or sparse. The mathematical formula for updating the learning model's weights based on the Adam optimizer is as follows:

$$w_t = w_{t-1} - \alpha \frac{\hat{m}_t}{\epsilon + \sqrt{\hat{v}_t}},$$

with:

$$\hat{m}_t = \frac{m_t}{1-\beta_1},$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2},$$

$$m_t = \beta_1 \, m_{t-1} + (1 - \beta_1) \, dw_t,$$

$$v_t = \beta_2 \, v_{t-1} + (1 - \beta_2) \, dw_t^2,$$

where: α is the learning rate, β_1 is the momentum parameter, β_2 is the RMSProb parameter, and ϵ is a small number around 10^{-8} to avoid dividing by zero that would cause a fatal error during

training. Figure 10 shows the impact of the Adam optimizer to speed up the convergence of SGD of PointNet CNN.

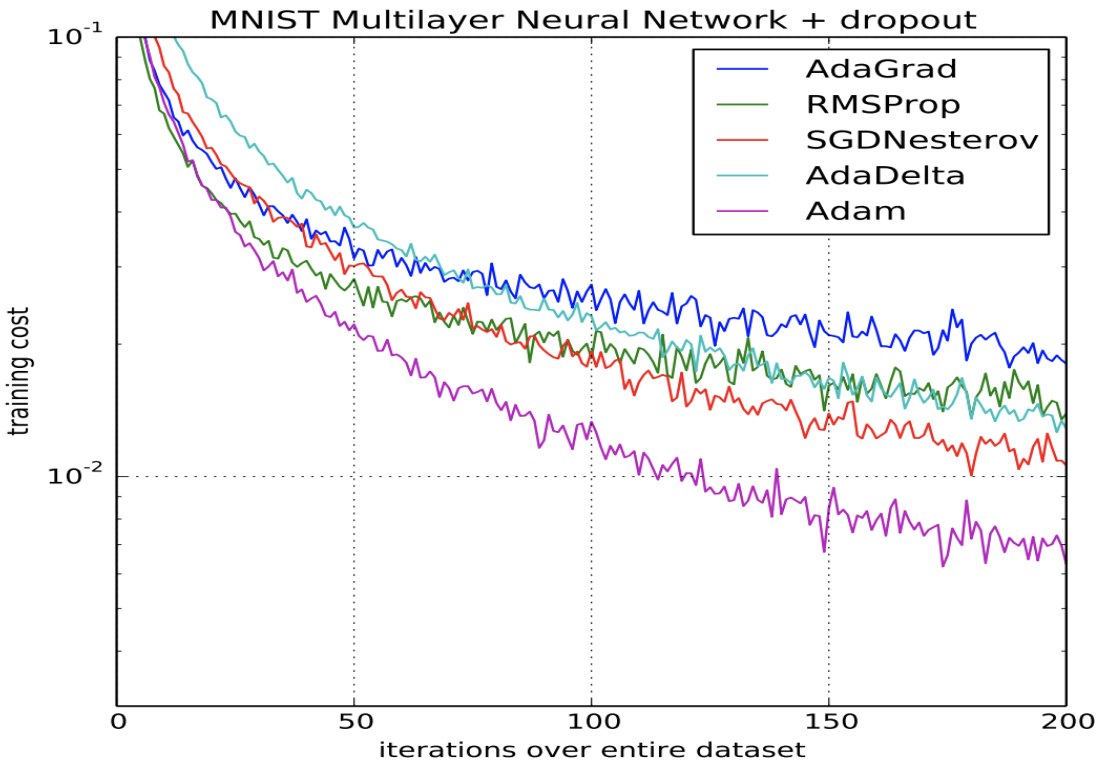

Figure 10. Adam Optimizer Impact on Convergence Speed of Gradient Descent. Reproduced From [72]

The learning rate is a crucial hyperparameter determining SGD's convergence accuracy and speed. For instance, if the learning rate is very high, SGD overshoots the optimal minima and converges to an inaccurate value. On the contrary, if the learning rate is very low, SGD converges so slowly. Then, the learning rate should be tuned carefully to avoid overshooting the optimum model's weights and the slow speed of the model learning. Ng advises researchers in [14] to tune the learning rate within the scale from 0.0001 to 1 for all types of deep neural networks.

Furthermore, A learning rate decay [75] is used as an optimizer to decrease the learning rate gradually from 0.5 to 0.99 over the training epochs to ensure the convergence of the loss function to local optimal minima. Thus, the mini-batch gradient descent slows down in the last epochs to precisely pick up the optimal minima. The learning rate decay has two techniques: fixed interval scheduling and continuous decay. Fixed interval scheduling has predefined discrete steps; the mini-batch gradient descent uses these steps consecutively. Each step is used as a learning rate decay during a fixed interval of training epochs, as shown in Figure 11. On the other hand, the continuous decay technique is based on a continuous mathematical function like exponential decay to make the learning rate decay over training epochs continuously.

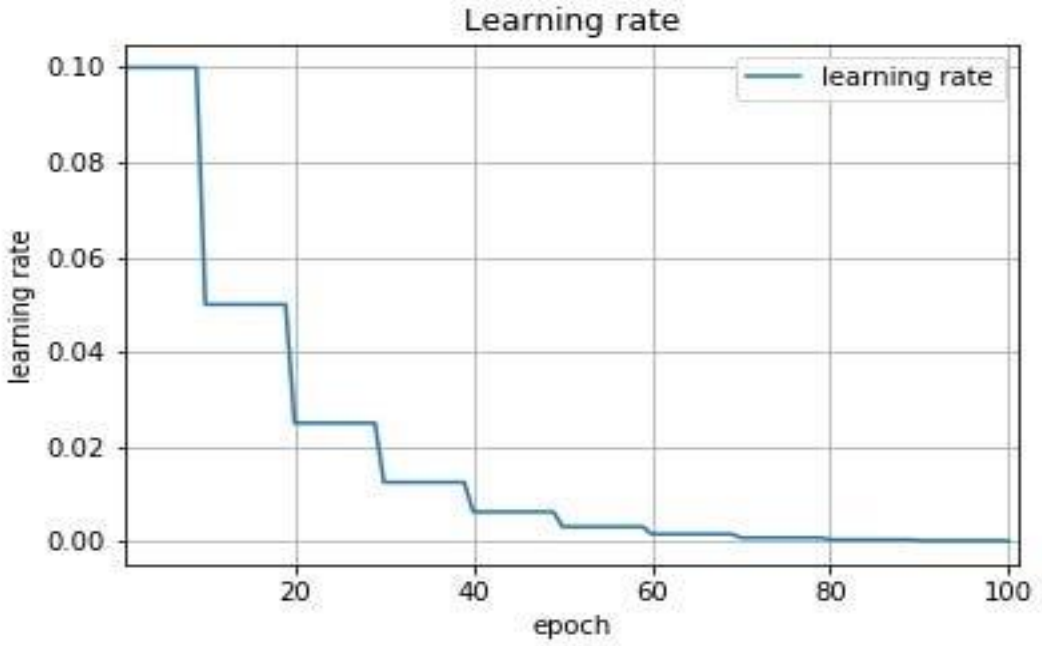

Figure 11. Fixed Interval Scheduling-Based Learning Rate Decay. Reproduced From [76]

A couple of regularization techniques, Dropout [77] and Early Stopping were exploited to prevent the model from overfitting the training dataset and make it generalize to the unseen dataset in the case of cross-validation and test datasets. Dropout was applied on the last fully-

connected layer direct before the Softmax output layer with a keep-prob ratio of 0.7, which means that seventy percent of the units in this layer are most likely to be dropped out during training to simplify NN architecture, thereby decreasing the generalization gap between training and validation. In other words, all incoming and outgoing links connected to the dropped units are removed. Dropout was turned off during testing to remove any source of randomness in the results due to the stochastic behavior of the dropout technique at removing the activation units in the CNN. Figure 12 shows applying the dropout technique on two consecutive hidden layers with a keep-prob ratio of 0.4.

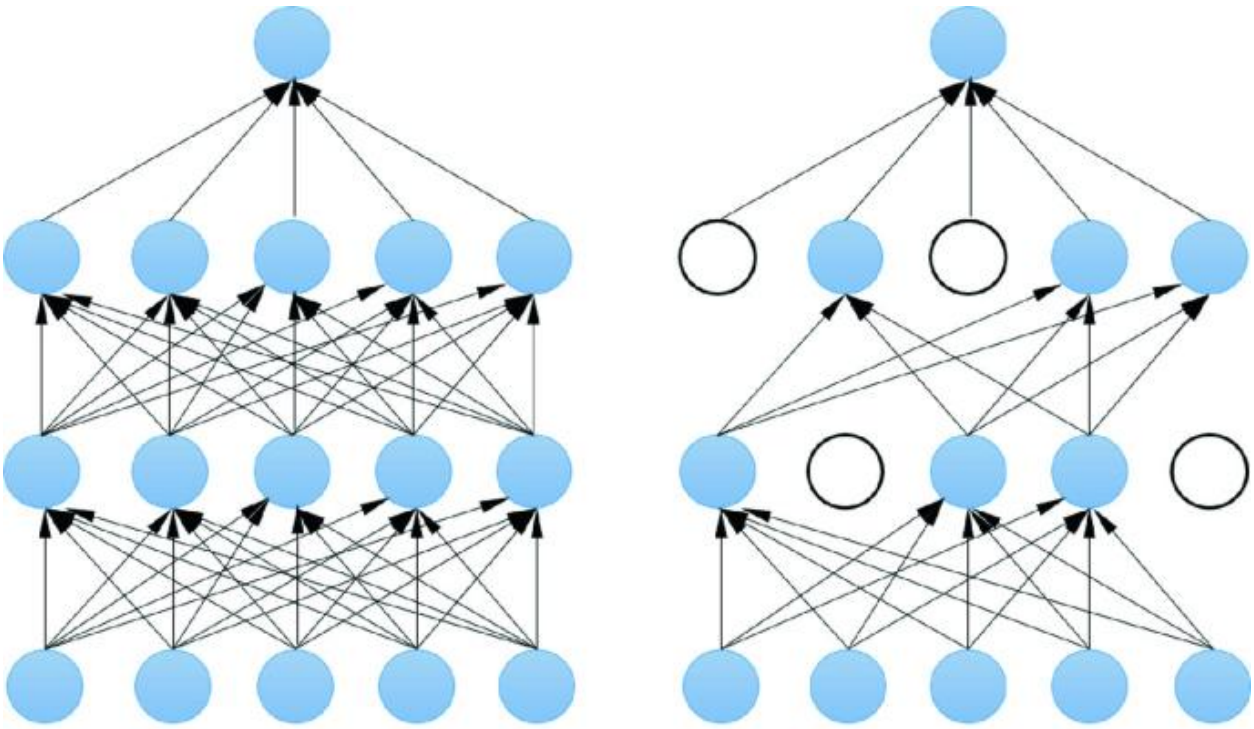

Figure 12. Dropout Regularization in Neural Networks. Reproduced From [77]

An early stopping strategy was operated to optimize cross-entropy loss and overcome overfitting simultaneously in a non-orthogonal manner. proposes a mathematical formula to guarantee selecting the minimum cross-validation loss while achieving convergence in training

cross-entropy loss function, as shown in Figure 13; otherwise, early stopping is canceled, and training continues till the last epoch, the 100th epoch in this model. Other hyperparameters like learning rate, momentum, and mini-batch size selection will be discussed in detail later in the following subsections.

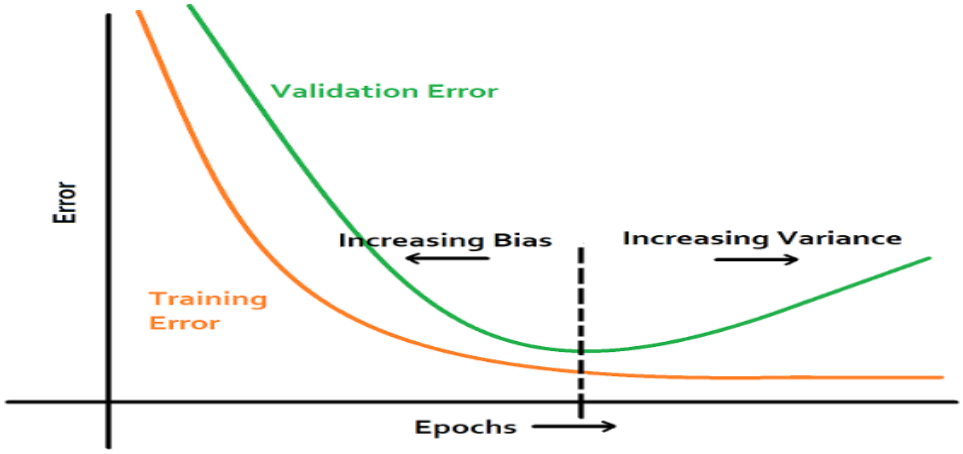

Figure 13. Early Stopping Strategy: Regularization Technique. Reproduced From [78]

TensorFlow framework [79] presents open-source APIs to ease the development of mini-batch gradient descent along with involving the associated optimizers and hyperparameters aforementioned.

2.5.2. Non-Regularized Standard Feedforward Neural Network (Deep Learning Model)

2.5.2.1. Feedforward Neural Network Structure

Deep feedforward networks [10], also known as feedforward neural networks (FNN) or multilayer perceptrons (MLPs), are fundamental models in the field of deep learning. Their purpose is to approximate a given function f*(x). For instance, in classification tasks, y = f*(x) maps an input x to a category y. A feedforward network defines a mapping y = f (x; θ) and learns the optimal parameter values θ that yield the best function approximation. These models are

referred to as "feedforward" because the information flows through the function being evaluated from the input x, through intermediate computations that define f, and finally to the output y. Unlike recurrent neural networks [80], feedforward networks lack feedback connections where the model's outputs are fed back into itself. Machine learning developers heavily rely on feedforward networks, as they serve as the foundation for various important applications. For example, convolutional networks used in image object recognition are a specialized type of feedforward network. Feedforward networks are essential in understanding and developing recurrent networks, which are widely used in natural language applications. The term "networks" in feedforward neural networks stems from the fact that they are typically constructed by combining multiple functions. The model is represented by a directed acyclic graph that illustrates how these functions are composed together. A common structure in neural networks is a chain-like composition of functions, where three functions $f^{(1)}$, $f^{(2)}$, and $f^{(3)}$ are connected in a sequence to form f(x) = $f^{(3)}(f^{(2)}(f^{(1)}(x)))$. Each function in this chain is referred to as a layer of the network, with $f^{(1)}$ being the first layer, $f^{(2)}$ being the second layer, and so on, as shown in Figure 14.

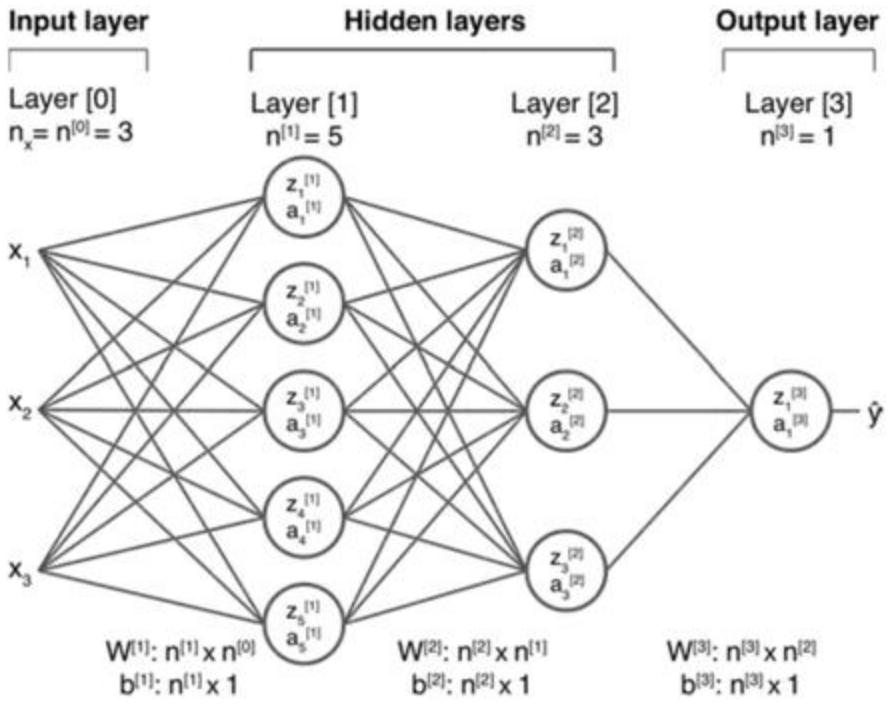

Figure 14. Three-Layer Feedforward Neural Network. Reproduced From [81]

The terminology "deep learning" comes from the fact that the depth of a model is determined by the overall length of the chain in a feedforward network. The last layer of the network is referred to as the output layer. During the training of a neural network, the objective is to make f(x) closely match f*(x). The training data provides noisy and approximate examples of f*(x) evaluated at different training points. Each example x is associated with a label y ≈ f*(x). The training examples explicitly specify the desired behavior of the output layer at each point x, aiming to produce a value that closely aligns with y. However, the behavior of the other layers is not directly specified by the training data. The learning algorithm must determine how to utilize these layers to achieve the desired output, but the training data does not indicate the specific tasks each individual layer should perform. These unspecified layers are referred to as hidden layers. The networks are called "neural" as they draw loose inspiration from neuroscience. Each hidden layer in the network typically consists of vector-valued elements, where the

dimensionality of these hidden layers determines the model's width. Each element in the vector can be seen as analogous to a neuron, with the layer representing a collection of units that function in parallel. Each unit, similar to a neuron, receives input from multiple other units and computes its activation value. The concept of using multiple layers with vector-valued representations is influenced by neuroscience. The selection of functions $f^{(i)}(x)$ used to compute these representations is loosely guided by observations of the functions computed by biological neurons. However, modern neural network research encompasses various mathematical and engineering disciplines, and the primary goal is not to precisely mimic brain functioning. It is more accurate to view feedforward networks as machines designed for function approximation that prioritize statistical generalization, occasionally drawing insights from our understanding of the brain, rather than serving as brain models themselves.

Standard feedforward neural networks are the basis of convolutional neural networks (CNNs) used for computer vision (CV) applications and recurrent neural networks (RNNs) utilized for natural language processing (NLP) applications.

2.5.2.2. Hyperparameters

The hyperparameters of non-regularized standard feedforward neural network are:

- Learning rate and its decay rate.
- Number of layers of the network.
- Number of nodes in each layer.
- Mini-batch size.
- Momentum (β_1).
- Type of activation functions through all layers.

Activation functions [14, 82] play a crucial role in the computations of feedforward neural networks by introducing non-linearity. They are applied to the outputs of individual neurons, determining their activation levels, which are then passed as inputs to the next layer. By enabling neural networks to learn and represent complex relationships between inputs and outputs, activation functions facilitate the solution of intricate problems. There exist several widely used activation functions in feedforward neural networks. A feedforward usually does not use all types of activation functions, but one or two of them are carefully selected for hidden and output layers of FNN depending upon the application, either logistic regression, multi-class classification, or linear regression. Let's explore some of them:

a) Sigmoid Activation [83]:

The sigmoid function also referred to as the logistic function, possesses an S-shaped curve, as shown in Figure 15 , mapping inputs to values ranging from 0 to 1. Its formula is: f(x) = 1 / (1 + e^(-x)). The sigmoid activation function proves valuable in scenarios where outputs need to represent probabilities, such as in binary classification problems.

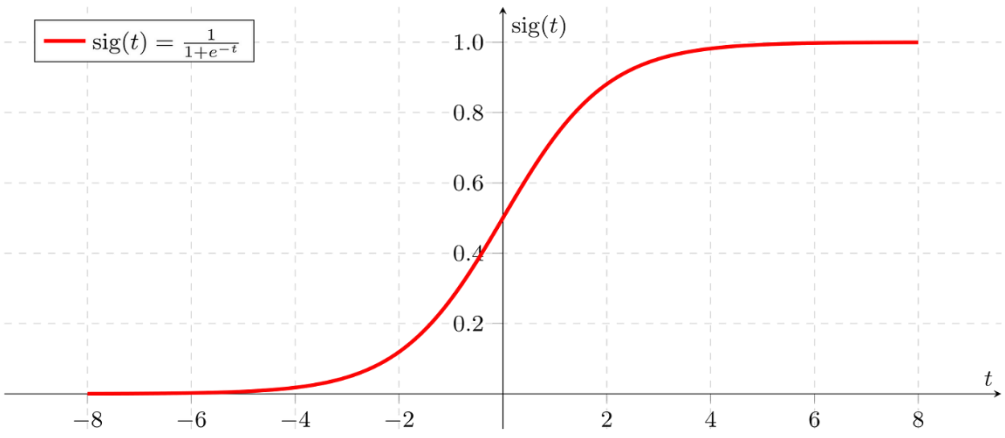

Figure 15. Sigmoid Function

b) Hyperbolic Tangent Activation (Tanh):

Similar to the sigmoid function, the hyperbolic tangent function maps inputs to values between -1 and 1, as shown in Figure 16. It is defined as: f(x) = (e^(x) - e^(-x)) / (e^(x) + e^(-x)). Tanh is commonly utilized in hidden layers of neural networks as it generates outputs centered around zero, aiding in training convergence.

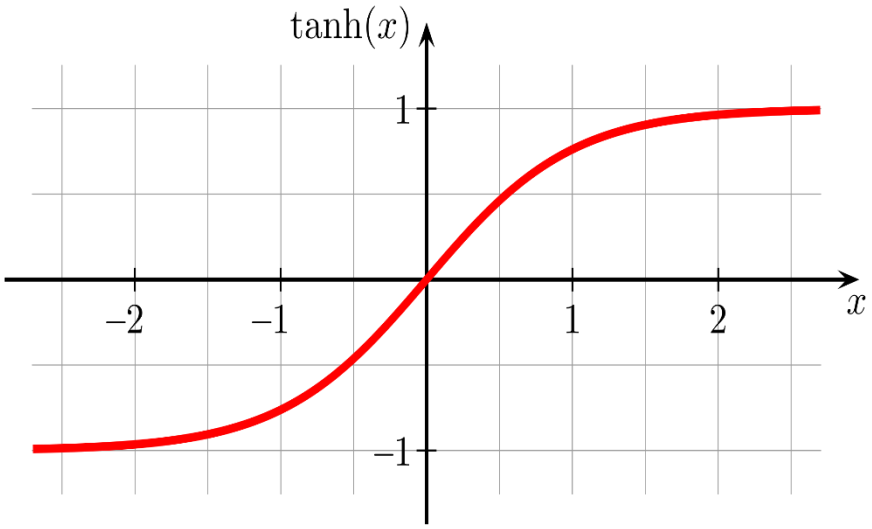

Figure 16. Hyperbolic Tangent Activation Function

c) Rectified Linear Unit (ReLU):

The ReLU function is a piecewise linear function that outputs the input if it is positive; otherwise, it outputs zero, as shown in Figure 17. Its definition is: f(x) = max (0, x). ReLU has gained popularity due to its simplicity and its ability to address the vanishing gradient problem, which can arise in deep neural networks. It is usually used for both hidden and output layers in non-negative linear regression applications.

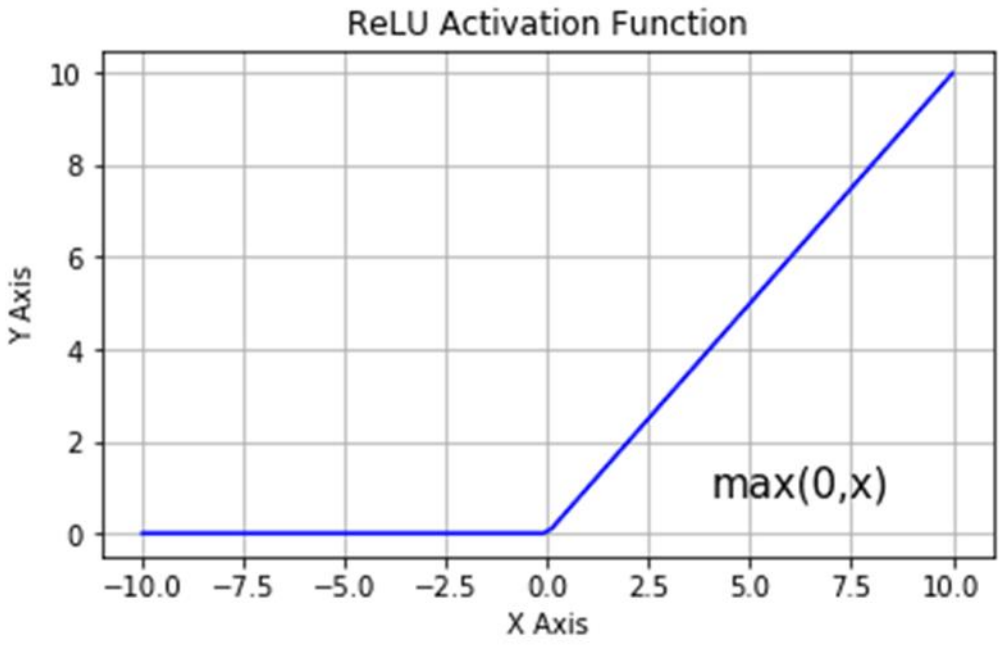

Figure 17. RELU Activation Function

d) Leaky ReLU: Leaky ReLU is an extension of the ReLU function that introduces a small negative slope for negative inputs, preventing the occurrence of dormant neurons, as shown in Figure 18. It is defined as f(x) = max (0.01x, x). The small slope ensures that neurons are not completely deactivated for negative inputs.

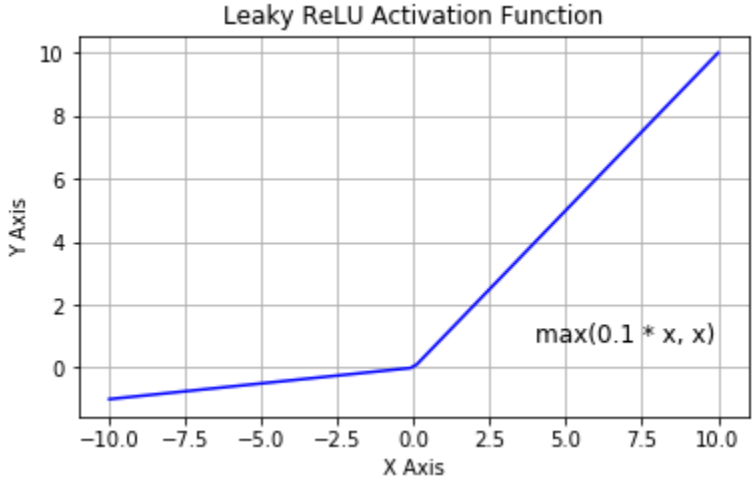

Figure 18. Leaky RELU Activation Function

e) Softmax Activation: The Softmax function is employed in the output layer of neural networks when dealing with multi-class classification tasks. Taking a vector of real numbers as input, it normalizes them into a probability distribution over the classes, ensuring that the sum of probabilities equals 1. It is defined as: $\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_i}}$ for i = 1, …, K, z = (z_1, z_2, …, z_K), and $\sum_1^K \sigma(z)_i = 1$.

2.5.3. Regularized Standard Feedforward Neural Network (Deep Learning Model).

The hyperparameters of non-regularized standard feedforward neural network are:

a) Learning rate and its decay rate.
b) Number of layers of the network.
c) Number of nodes in each layer.
d) Mini-batch size of SGD.
e) Momentum (β_1).
f) Type of activation functions through all layers.
g) Regularization Parameter (λ).

Regularization techniques are exploited to avoid the overfitting of feedforward neural network. When learning model overfits the training dataset, they cannot generalize well to cross-validation and test dataset. Therefore, regularization minimizes the generalization gap between cross-validation/testing and training datasets, as shown in Figure 19. The most common regularization techniques are [84]:

a) LP-Norm (Weight Decay) Regularization.
b) Dropout Regularization.

c) Early Stopping Strategy.

Dropout and early stopping techniques were discussed in detail in Section 2.4.1.2 above. Therefore, the scope of this subsection is explaining the principle of LP-norm regularization technique.

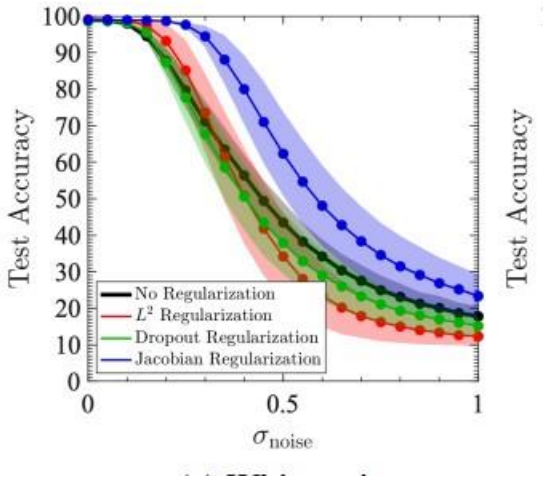

Figure 19. Impact of Regularization on Learning Model's Test Accuracy. Reproduced From [85]

2.5.3.1. LP-Norm (Weight Decay) Regularization

Regularization [10, 14] has been a common practice in machine learning for many years, even predating the emergence of deep learning. Linear models like linear regression and logistic regression have straightforward and effective regularization strategies. Various regularization approaches aim to limit the capacity of models, such as neural networks, linear regression, or logistic regression, by adding a parameter norm penalty $\Omega(\theta)$ to the objective function J. The regularized objective function, denoted as J, is defined as $J(\theta; X, y) = J(\theta; X, y) + \alpha \Omega(\theta)$, where $\alpha \in [0, \infty)$ is a hyperparameter that controls the contribution of the norm penalty term, Ω, relative to the standard objective function J. Setting α to 0 eliminates regularization, while larger values of α lead to stronger regularization. When our training algorithm minimizes the regularized

objective function J, it simultaneously reduces the original objective J on the training data and some measure of the parameter size, θ (or a subset of the parameters. Different choices for the parameter norm Ω can result in different preferred solutions.

In the case of neural networks, we typically choose a parameter norm penalty Ω that only penalizes the weights of the affine transformation at each layer, leaving the biases unregularized. Biases typically require less data than weights to accurately estimate their values. Each weight captures the interaction between two variables, thus requiring observations in various conditions to fit well. On the other hand, each bias controls a single variable, meaning that leaving them unregularized doesn't introduce excessive variance. Additionally, regularizing the bias parameters can lead to significant underfitting. Therefore, the vector w represents all the weights affected by the norm penalty, while the vector θ includes both w and the unregularized parameters.

In the context of neural networks, it may be desirable to use separate penalties with different α coefficients for each layer of the network. However, since searching for optimal values of multiple hyperparameters can be computationally expensive, it is reasonable to apply the same weight decay across all layers to reduce the search space.

There are three common forms of LP-norm regularization in the field of machine learning:

a) L1-Norm Regularization [86]:

It is used for logistic regression, including feedforward networks. The optimization function using this regularization is:

$$J(W; b) = \frac{1}{m}\sum_{i=1}^{m} J^{(i)}\left(W^{(i)};\ b^{(i)}\right) + \frac{\lambda}{2m}\ \|W\|_1,$$

$$\|W\|_1 = \sum_{j=1}^{n_x} |W_j|,$$

where: J (W; b) is the optimization cost function, W: logistic regression model's parameters/weights, $\|W^{[l]}\|_1$ is the value of L1-norm of the weights, b is the bias, λ is the regularization parameter, and m is the number of training examples.

b) L2-Norm Regularization [87]:

It is used for logistic regression, including feedforward networks. The optimization function using this regularization is:

$$J(W; b) = \frac{1}{m}\sum_{i=1}^{m} J^{(i)}(W^{(i)}; b^{(i)}) + \frac{\lambda}{2m}\|W\|_2^2,$$

$$\|W\|_2^2 = \sum_{j=1}^{n_x} W_j^2,$$

where: J (W; b) is the optimization cost function, W: logistic regression model's parameters/weights, b is the bias, $\|W\|_2^2$ is the squared value of L2-norm of the weights, λ is the regularization parameter, and m is the number of training examples.

c) Frobenius-Norm Regularization [88]:

It is used for neural networks, including feedforward networks. The optimization function using this regularization is:

$$J(W; b) = \frac{1}{m}\sum_{i=1}^{m} J^{(i)}(W^{(i)}; b^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L}\|W^{[l]}\|_F^2,$$

$$\|W^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l]}}\sum_{j=1}^{n^{[l-1]}}(W_{ij}^{[l]})^2,$$

where: J (W; b) is the optimization cost function, $W^{[l]}$: Feedforward neural network model's parameters/weights at layer number "l", b is the bias, $\|W^{[l]}\|_F^2$ is the squared value of Frobenius-norm of the weights, $n^{[l]}$ is the number of nodes in the layer number "l", λ is the regularization parameter, and m is the number of training examples.

2.5.4. Support Vector Machine (Supervised Machine Learning Model).

This subsection presents an overview of the main concepts and hyperparameters of the Support Vector Machine.

2.5.4.1. SVM Concepts

Support Vector Machine (SVM) [14] is a widely utilized and potent supervised machine learning technique suitable for both classification and regression tasks, particularly effective in handling intricate, high-dimensional data. Initially introduced in the 1990s by Vladimir Vapnik and colleagues, SVMs have gained substantial popularity in the machine learning domain. The primary objective of SVM is to locate an optimal hyperplane that can effectively separate data points belonging to different classes while maximizing the margin. This hyperplane acts as a decision boundary, enabling SVM to classify new data based on their features. The margin represents the distance between the decision boundary and the nearest data points from each class, and the SVM algorithm strives to maximize this margin. What sets SVM apart from other algorithms is its ability to handle both linearly separable and non-linearly separable data. When the data is linearly separable, SVM aims to find a hyperplane that distinctly separates the classes with a margin. However, in the case of non-linearly separable data, SVM employs a technique known as the kernel trick. This technique maps the original input space into a higher-dimensional feature space where the data becomes linearly separable. Consequently, SVM can effectively classify complex, non-linear patterns. SVMs are widely regarded as robust and versatile machine learning algorithms, offering several key advantages. They possess strong generalization capabilities, allowing them to perform well on unseen data. SVMs are also less prone to overfitting compared to some other algorithms. Furthermore, the concept of margin maximization employed by SVMs makes them less sensitive to outliers.

Nonetheless, SVMs do have certain limitations [3]. They can be computationally demanding, particularly when working with large datasets. Selecting an appropriate kernel function and tuning the hyperparameters of an SVM can be challenging and require expertise. Despite these limitations, SVMs continue to be popular in various fields, including image recognition, text classification, bioinformatics, and finance, owing to their effectiveness and versatility in handling complex data patterns.

Support Vector Machines (SVMs) are advanced supervised machine learning models utilized for classification and regression tasks. They excel particularly in solving binary classification problems, though they can also be extended to handle multi-class classification challenges. SVMs operate on the fundamental principle of identifying an optimal hyperplane that maximizes the margin between data points of different classes. This hyperplane serves as the decision boundary, maximizing the separation between the closest data points of distinct classes, known as support vectors. By maximizing the margin, SVMs aim to enhance generalization and robustness when dealing with new, unseen data.

Outlined below is a concise overview of the key elements and concepts in SVMs [8]:

a) Linear Separability:

   SVMs operate on the assumption that input data can be separated into distinct classes through a linear boundary. When the data is linearly separable, SVMs determine the hyperplane that maximizes the margin.

b) Kernel Trick:

To handle non-linearly separable data, SVMs employ the kernel trick. The kernel function (similarity function) implicitly maps input data to a higher-dimensional feature space, enabling linear separation. Common kernel functions include:

- Linear Kernel.

  It is a special case of the polynomial kernel and is restricted by non-negative numbers. Its kernel is defined as follows:

$$f_j^{(i)} = \text{kernel}\ (x^{(i)}, l^{(j)}) = (x^{(i)})^T\ l^{(j)},$$

  where f_j is the kernel function, x_j is the feature number "j", and l_j is the landmark number "j", $where$ i = 0, 1, 2, …, m and j = 0, 1, 2, …, n. "m" is the number of training examples and "n" is the number of features.

- Polynomial Kernel.

  It has a lower performance than Gaussian radial basis function (RBF) kernel. In addition, it is limited to non-negative

$$f_j^{(i)} = \text{kernel}\ (x^{(i)}, l^{(j)}) = ((x^{(i)})^T\ l^{(j)})^d$$

  $Where$ f_j is the kernel function, x_j is the feature number "j", l_j is the landmark number "j", and d is the polynomial degree, where i = 0, 1, 2, …, m and j = 0, 1, 2, …, n. "m" is the number of training examples and "n" is the number of features.

- Gaussian Radial Basis Function (RBF).

$$f_j^{(i)} = \text{kernel}\ (x^{(i)}, l^{(j)})\ = \exp\left(-\frac{\|x^{(i)} - l^{(j)}\|^2}{2\,\sigma^2}\right),$$

where f_j is the kernel function, x_j is the feature number "j", l_j is the landmark number "j", and σ is the decay coefficient of the kernel, where i = 0, 1, 2, …, m and j = 0, 1, 2, …, n. "m" is the number of training examples and "n" is the number of features.

- There are other kernels like sigmoid kernel, string kernel, chi-square kernel, histogram, and intersection kernel.

c) Margin:

The margin in SVMs represents the region between the decision boundary and the support vectors, as shown in Figure 20. SVMs strive to maximize this margin, as it corresponds to greater class separation and improves the model's ability to generalize to unseen data.

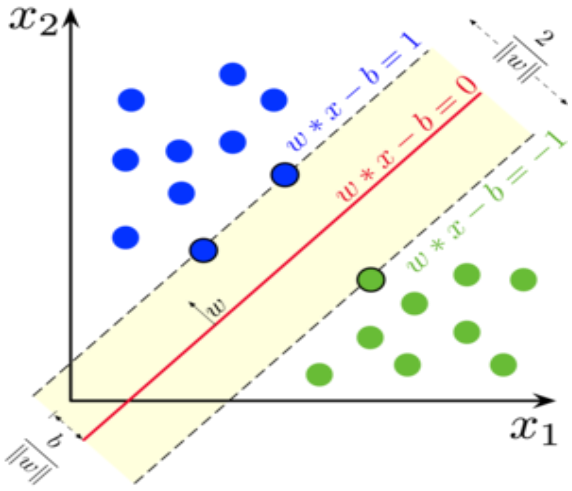

Figure 20. Support Vector Machine (SVM): Large Margin Classification. Reproduced From [89]

d) Soft Margin Classification:

In scenarios where perfect separability is not achievable, SVMs introduce the concept of soft margin classification. This involves allowing some misclassifications to strike a balance between maximizing the margin and minimizing classification errors. The regularization parameter "C" controls the trade-off between margin maximization and error minimization.

e) Support Vectors:

Support vectors are data points lying on the margin or misclassified. They play a pivotal role in defining the decision boundary and making predictions. SVMs rely exclusively on support vectors, making them memory-efficient compared to utilizing the entire dataset.

f) SVM Training:

Training SVMs involves solving a quadratic optimization problem to determine the optimal hyperplane. Various optimization algorithms, such as Sequential Minimal Optimization (SMO) and gradient descent, are employed to efficiently solve the optimization problem.

g) SVM Extensions:

SVMs have been extended to handle multi-class classification using techniques like One-vs-One (OvO) and One-vs-All (OvA). Additionally, SVMs can be applied to regression tasks by formulating the problem as an optimization task to find a suitable hyperplane within a specified margin.

Support Vector Machines have gained substantial popularity due to their strong theoretical foundations, versatility, and capability to handle complex datasets. Their robustness,

effectiveness in high-dimensional spaces, and generalization capabilities make them valuable tools in machine learning and pattern recognition applications.

2.5.4.2. SVM Hyperparameters

Hyperparameters can be extracted from the optimization function of SVM [90]. The cost function of SVM is as follows:

$$\text{Cost}(\theta) = C \sum_{i=1}^{m} \left[y^{(i)} \, cost_1\left(\theta^T f^{(i)}\right) + \left(1 - y^{(i)}\right) cost_0(\theta^T f^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^{n} {\theta_j}^2 (1),$$

where C ($\frac{1}{\lambda}$) is a regularization parameter to ensure trade-off between minimizing the cost function and maintaining large classification margin. It is the reciprocal of the regularization parameter used in linear regression, logistic regression, and deep neural networks. The kernel function, $f^{(i)} = \begin{bmatrix} f_0^{(i)} \\ \vdots \\ f_n^{(i)} \end{bmatrix}$, where n is the number of features.

The Cost Function is used to train the SVM. By minimizing the value of J (θ), it is ensured that the SVM is as accurate as possible. In the equation, the functions cost1 and cost0 refer to the cost for an example where y=1 and the cost for an example where y=0, as shown in Figure 21. For SVMs, the cost (θ) is determined by kernel (similarity) functions. The most efficient kernel is the Gaussian Radial Basis Function (RBF) kernel that satisfies the following relation:

$$f_j^{(i)} = \text{kernel}\,(x^{(i)}, l^{(j)}) = \exp\left(-\frac{\left\|x^{(i)} - l^{(j)}\right\|^2}{2\,\sigma^2}\right) (2),$$

Where f_j is the kernel function, x_j is the feature number "j", l_j is the landmark number "j", and σ is the decay coefficient of the kernel, where i = 0, 1, 2, …, m and j = 0, 1, 2, …, n. "m" is the number of training examples and "n" is the number of features.

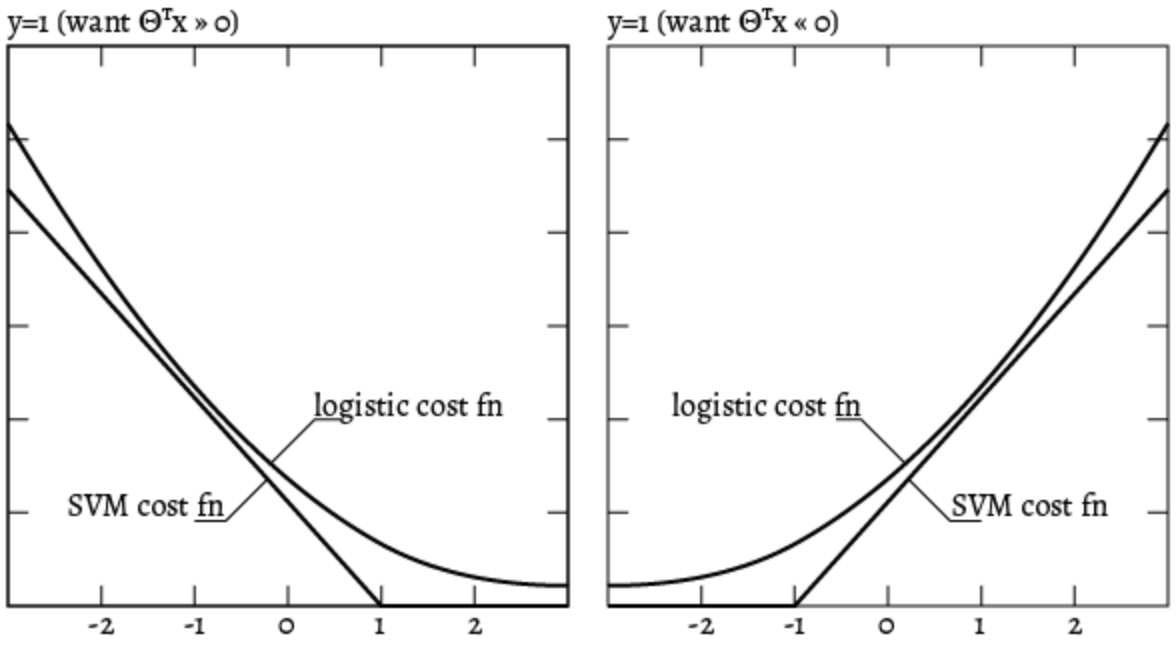

Figure 21. Cost Function: SVM VS Logistic Regression. Reproduced From [91]

According to the equations (1) and (2), the main hyperparameters of SVM are C-hyperparameter for regularization and sigma (σ) that controls the decay rate of the RBF kernel.

2.5.5. Principal Component Analysis (Unsupervised Machine Learning Model).

Principal Component Analysis (PCA) [3] is a widely utilized unsupervised method for reducing the dimensionality of data in the fields of speeding up machine learning model's training and compressing data analysis. It enables the transformation of a high-dimensional dataset into a lower-dimensional space while retaining the essential information. It was introduced by Karl Pearson in 1901. Moreover, PCA has become an indispensable tool for exploring and visualizing data, as well as extracting meaningful features. Indeed, PCA is inevitable to use when the learning model is trained very slowly or it consumes huge memory size. The primary goal of PCA is to identify a new set of uncorrelated variables, called principal components, that capture the maximum variance in the original dataset. Each principal

component is a linear combination of the original variables, arranged in descending order of variance explained. By selecting a subset of these components, the data's dimensionality can be effectively reduced while preserving as much pertinent information as possible.

2.5.5.1. PCA Procedures

Executing PCA involves several steps [92] as visualized in Figure 22:

a) Firstly, the data is standardized to have zero mean and unit variance, ensuring equal contributions from all variables by feature scaling and mean normalization as follows:

- Calculate the mean of each feature in the learning mode:

$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)},$$

where μ_j is the mean of feature number "j", m is the number of training examples, and $x_j^{(i)}$ is the feature's value at the training example number "i".

- Replace each $x_j^{(i)}$ with $x_j^{(i)}$ - μ_j to obtain zero mean.
- Normalize the mean:

$$x_j^{(i)} = \frac{x_j^{(i)}}{\sqrt{\frac{1}{m} \sum_{i=1}^{m} (x_j^{(i)})^2}}$$

b) Then, the covariance or correlation matrix of the standardized data is computed as follows:

$$\text{Sigma} = \text{Covariance matrix} = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)}) \ (x^{(i)})^T,$$

where $(x^{(i)})^T$ is the transpose of the features' matrix with a dimension of n x 1 for the training example number "i", where n is the number of learning model's features.

c) Subsequently, the eigen vectors and eigen values of this matrix are determined using the singular value decomposition (SVD) method [93], with the eigenvectors representing the principal component directions on which the original dataset's features are projected and the eigen values indicating the variance explained by each component.

$$[U\ S\ V] = \text{svd (Sigma)},$$

where U is the matrix of unit eigen vectors that represent the directions of all PCA's components before reduction, in the case that number of PCA's components equal the number of learning model's features, with a dimension of m x n, and S is the eigen-values matrix whose diagonal contains the variance of each PCA's component.

d) Finally, the original data is projected onto the eigen vectors to obtain the principal components as follows:

- Extract unit eigen vectors as many as the PCA's components whose number is K as follows:

$$U_{reduced} = U\ [1: END, 1: K],$$

where $U_{reduced}$ is the matrix of unit eigenvectors that represent the directions of all PCA's components after reduction, in the case that number of PCA's components (K) less than the number of learning model's features (n), with a dimension of m x K, m is the number of training examples, and K is the number of PCA's components after dimensionality reduction.

- Project original dataset X onto the eigenvectors in $U_{reduced}$ to obtain the PCA's components as follows:

$$Z = (U_{reduced})^T\ X,$$

where $U_{reduced}$ is the matrix of unit eigen vectors that represent the directions of all PCA's components after reduction, in the case that number of PCA's components (K) less than the number of learning model's features (n), with a dimension of m x K, m is the number of training examples, and K is the number of PCA's components after dimensionality reduction, and X is the features' matrix with a dimension of n x m for the training example number "*i*", where n is the number of learning model's features.

PCA offers several advantages in data analysis. It simplifies complex datasets by reducing the number of variables, facilitating easier data visualization and interpretation. Additionally, it aids in identifying important patterns, relationships, and trends within the data. Moreover, PCA can be employed for feature extraction, wherein the most informative principal components are selected to train machine learning models, resulting in reduced computational complexity and mitigated risks of overfitting.

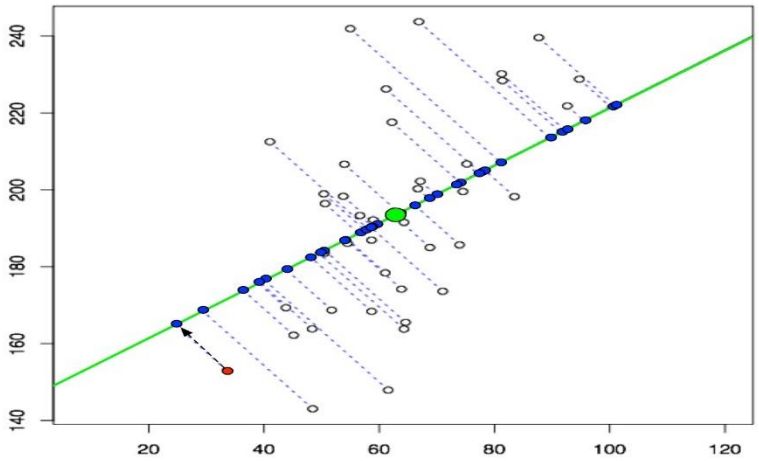

Figure 22. Principal Component Analysis (PCA): Dimensionality Reduction and Projection. Reproduced From [94]

However, it's important to consider the limitations of PCA. It assumes linearity in the data and may not effectively capture non-linear relationships. Interpreting the principal components can be challenging as they are combinations of the original variables. Furthermore, PCA relies on the assumption of variable normality, and outliers can significantly impact the results. Despite these limitations, PCA remains a valuable technique for exploratory data analysis, dimensionality reduction, and feature extraction tasks. Its applications span diverse domains such as image and signal processing, genetics, finance, and social sciences, where it provides insights and simplifies the analysis of complex datasets.

2.5.5.2. PCA Hyperparameters

The main hyperparameter at PCA learning models is the number of PCA components (K). This hyperparameter should be tuned to obtain the minimum value for K that would satisfy the following criterion [3, 14]:

$$\frac{\sum_{i=1}^{K} s_{ii}}{\sum_{i=1}^{n} s_{ii}} \geq 0.9,$$

where s_{ii} is an element of the diagonal of the "U" matrix calculated by the singular value decomposition of the covariance matrix as follows:

$$\text{Sigma} = \text{Covariance Matrix} = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)})\ (x^{(i)})^T$$

$$[\text{U S V}] = \text{svd (Sigma)}$$

$\frac{\sum_{i=1}^{K} s_{ii}}{\sum_{i=1}^{n} s_{ii}}$ is called the retained variance. It should be at least 0.9 to maintain the high accuracy of the learning model after the dimensionality reduction.

2.5.6. K-means Clustering (Unsupervised Machine Learning Model)

K-means clustering [95, 96] is a well-known unsupervised machine learning technique utilized to partition a given dataset into K clusters based on their similarities. The primary objective of the algorithm is to group similar data points together by associating them with the nearest cluster centroid. This process aims to minimize the overall intra-cluster variance or the sum of squared distances between data points and their respective centroids. The K-means algorithm operates through the following steps:

- Initialization: Initially, K data points are randomly chosen to serve as the initial cluster centroids.
- Assignment Step: Each data point is then assigned to the closest cluster centroid. This is accomplished by computing the distance, often using the Euclidean distance measure, between each data point and all centroids. The data point is allocated to the cluster whose centroid is closest to it.
- Update Step: After assignment, the centroids of each cluster are recalculated as the mean of all the data points assigned to that cluster. This step relocates the centroids to the centers of their respective clusters.
- Repeat Assignment and Update: Steps 2 and 3 are repeated iteratively until the algorithm converges. Convergence happens when data points are consistently assigned to the same clusters, or a predefined maximum number of iterations is reached.
- Finally, the algorithm terminates, and the data points are grouped into K clusters, each represented by its centroid.

Choosing an appropriate value of K is crucial in the K-means algorithm. Various methods, such as the elbow method, silhouette score, or domain knowledge-based selection, are employed to

determine the most suitable K. Despite some limitations, such as sensitivity to initial centroids and difficulties in handling non-spherical clusters or clusters of varying sizes, K-means remains widely used in fields like image segmentation, customer segmentation, anomaly detection, and others, due to its simplicity and efficiency. It serves as a foundational algorithm in clustering and serves as a stepping stone to understanding more intricate clustering algorithms.

K-means clustering is a relatively straightforward algorithm with fewer hyperparameters to adjust compared to more complex machine learning methods. Nevertheless, there are several crucial hyperparameters that can influence the performance and behavior of the K-means algorithm:

- Number of Clusters (K): The most vital hyperparameter in K-means is the number of clusters (K) to create. The choice of K significantly impacts the clustering result, determining how finely the data will be grouped. Selecting the right K value is crucial since too few clusters might oversimplify the data, while too many can lead to overfitting or capture noise.
- Initialization Method: To initiate the clustering process, K-means requires an initial set of K centroids. The algorithm's sensitivity to the initial centroids is a known issue, as different initializations can lead to varying clustering outcomes. Several initialization methods exist, such as random initialization, K-means++, and K-medoids (PAM). K-means++ is a popular choice as it attempts to place the initial centroids far apart, leading to more stable results.
- Maximum Number of Iterations: K-means is an iterative algorithm, updating cluster assignments and centroids in each iteration until convergence. You can set a hyperparameter for the maximum number of iterations to control how many iterations the

algorithm should perform before stopping. Typically, convergence is reached well before reaching the maximum iteration limit.

- Convergence Tolerance: To determine when the algorithm has converged, you can set a convergence tolerance or threshold value. The algorithm stops iterating when the change in cluster assignments or centroids becomes smaller than the specified tolerance.
- Distance Metric: By default, K-means employs the Euclidean distance metric to calculate the distance between data points and centroids. However, you can choose other distance metrics depending on your data's nature.

In practice, the most common hyperparameters to tune in K-means are the number of clusters (K) and the initialization method. The ideal K value often relies on the specific problem and domain knowledge. Techniques like the elbow method or silhouette score can aid in determining an appropriate K value, but it's essential to consider that K-means is sensitive to noisy data and outliers. Preprocessing the data and handling outliers before applying K-means clustering is a recommended practice.

2.5.7. Decision Trees (Supervised Machine Learning Model)

Decision Trees [97, 98] are a well-known supervised machine learning technique used for both classification and regression tasks. They are valued for their simplicity, interpretability, and effectiveness in making decisions based on input features. The core concept of Decision Trees involves recursively dividing the data into subsets based on different feature values, resulting in a tree-like structure where internal nodes represent decisions based on features, and leaf nodes correspond to predicted outputs. Here's a breakdown of how Decision Trees function:

- Tree Construction: The algorithm begins with the entire dataset at the root node of the tree. It selects the most informative feature to split the data based on certain criteria like

Gini impurity, entropy, or mean squared error, which assess the homogeneity of the created subsets. The data is then divided into branches, with each branch representing a unique value of the chosen feature.

- Recursive Splitting: The process of feature selection and branching continues for each subset (child node) until a stopping condition is met. Stopping criteria may involve reaching a maximum tree depth, having a minimum number of samples in a node, or achieving data homogeneity.
- Tree Pruning (Optional): After constructing the full tree, pruning techniques may be applied to prevent overfitting. Pruning involves removing branches that contribute little to improving the tree's predictive performance.
- Prediction: To predict outcomes for new data points, they traverse the tree from the root node to a leaf node, following the decisions made at each internal node based on input features. The prediction at a leaf node corresponds to the majority class (in classification) or the mean value (in regression) of the samples in that leaf.

Here, the advantages and disadvantages of the algorithm can be summarized as follows:

- Advantages of Decision Trees:
  - ✓ Easy to comprehend and interpret, making them suitable for visualization and explanation.
  - ✓ Can handle both categorical and numerical data without extensive data preprocessing.
  - ✓ Able to capture non-linear relationships between features and the target variable.
  - ✓ Implicitly perform feature selection by selecting important features near the top of the tree.
- Disadvantages of Decision Trees:

- ✓ Prone to overfitting, particularly when the tree becomes deep and complex.
- ✓ Sensitive to slight changes in the data, potentially resulting in different tree structures.
- ✓ May not be ideal for tasks with highly complex relationships between features and the target variable.

Ensemble methods like Random Forest and Gradient Boosting can mitigate some of the limitations of individual Decision Trees by combining multiple trees to enhance predictive performance and generalization. Despite their drawbacks, Decision Trees remain widely used due to their simplicity and interpretability, especially when dealing with smaller datasets and situations where model interpretability is crucial.

The Decision Tree algorithm has several hyperparameters that can be tuned to optimize its performance and prevent overfitting. Here are the main hyperparameters of the Decision Tree algorithm:

- Criterion: The criterion is the function used to measure the quality of a split. Two common criteria are:
- "Gini impurity": Used for classification tasks, it measures the degree of impurity or disorder of a node. A lower Gini impurity indicates more homogeneous classes.
- "Entropy": Also used for classification tasks, it measures the level of information or uncertainty in a node. A lower entropy implies more pure classes.
- Splitter: The splitter determines the strategy used to choose the best feature to split the data at each node. The two options are:
- "Best": The best feature is selected based on the criterion to optimize the split.
- "Random": A random subset of features is considered for each split.

- Max Depth: The maximum depth of the decision tree. Setting a maximum depth limits the number of levels in the tree and helps prevent overfitting. If not specified, the tree will expand until all leaves are pure or contain a minimum number of samples.
- Min Samples Split: The minimum number of samples required to perform a split at a node. If the number of samples at a node is less than this value, the node becomes a leaf node.
- Min Samples Leaf: The minimum number of samples required to be at a leaf node. If the number of samples at a leaf node is less than this value, the tree might prune it, or the node may be removed.
- Max Features: The maximum number of features considered when looking for the best split. Setting this parameter can help control overfitting and improve the model's generalization.
- Class Weights: For imbalanced datasets, this parameter allows assigning different weights to classes to balance the impact of minority classes during the training process.
- Presort: Whether to presort the data to speed up the fitting process for small datasets. For larger datasets, setting it to "True" might lead to longer training times.

It's essential to choose appropriate hyperparameter values through hyperparameter tuning techniques like grid search or random search to achieve the best model performance for a specific problem. The optimal hyperparameter values can vary depending on the dataset and the complexity of the problem being addressed.

2.6. Datasets and their organization

This thesis uses three different kinds of datasets that feed the various learning models utilized in this thesis to conduct 3D image semantic segmentation, binary classification, and dataset

visualization. These datasets are sufficient to feed all diverse deep convolutional and feedforward neural networks and supervised and unsupervised machine learning models. The three datasets are as follows:

2.6.1. Point Cloud Images of Submarine Pipelines

The Dataset used is obtainable from [64, 99] as an open source for researchers. It was acquired by a 3D RGB two-camera stereo rig mounted on an autonomous vehicle to provide depth and color information which are the backbone of the classification process. Calibration was performed using a chess board to obtain rectification, extrinsic matrices, and distortion coefficients for both cameras. Depth was calculated online through the disparity between pairs of images by applying epipolar matching techniques using a ROS package [100] that gives deep insight into stereo calibration and epipolar matching. The dataset was labeled, as shown in Figure 23, with three colors: pipe (0,255,0), valve (0,0,255), and background (0,0,0), Green, Blue, and Black, respectively. Therefore, classification accuracy was calculated by measuring the matching percentage between labeled ground-truth pixels and classified pixels.

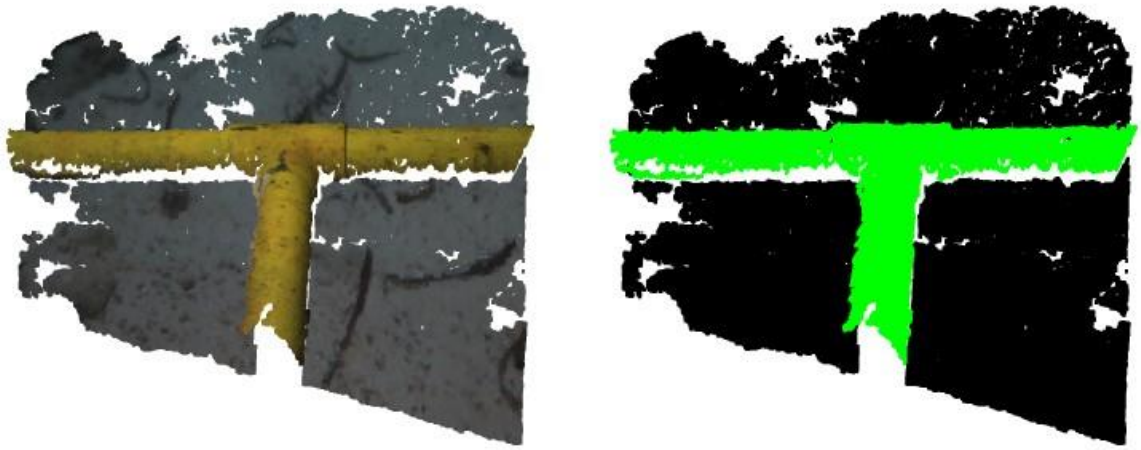

Figure 23. Original Dataset vs. Labeled Dataset. Reproduced from [65]

The dataset was partitioned into two subsets: 262 cloud points from an artificial pool and 22 points from a sea, as shown in Figure 24. The pool subset was used for training, validation, and base testing; it was split as follows: 80% of the pool subset is for training, 10% for validation, and 10% for base testing. Otherwise, the whole sea subset was used only for secondary testing. Indeed, Ng [55] recommends this data-splitting ratio when the dataset has a range of hundreds of examples.

Figure 24. Dataset Splitting. Reproduced from [64]

2.6.2. Sklearn's Dataset: sklearn.datasets.make_moons

The sklearn.datasets.make_moons function is a convenient tool provided by the scikit-learn library (sklearn) [101] for generating a synthetic dataset that consists of two interconnected half-moon shapes, as shown in Figure 25. Its primary use is in binary classification tasks and serves as a valuable resource for assessing algorithms capable of handling non-linearly separable data. By utilizing the make_moons function, you can adjust various parameters to tailor the characteristics of the generated dataset. These parameters include the number of samples, noise level, and random state. Here is an overview of these parameters:

- n_samples:

  This parameter allows you to specify the total number of data points to be created. The resulting dataset will contain an equal number of samples from each class.

- noise:

  The noise parameter determines the standard deviation of the Gaussian noise that is added to the data. Increasing the noise value introduces greater randomness, thereby making the dataset more challenging to classify.

- random_state:

  By setting the random_state parameter, you can control the seed for the random number generator. This ensures that the same dataset can be reproduced if the same random_state value is used.

The function returns a tuple comprising two arrays: the input features (X) and the corresponding class labels (y). The input features are represented as two-dimensional arrays, with each row denoting a data point and each column representing a feature. The class labels are stored as one-dimensional arrays, indicating the class membership for each data point.

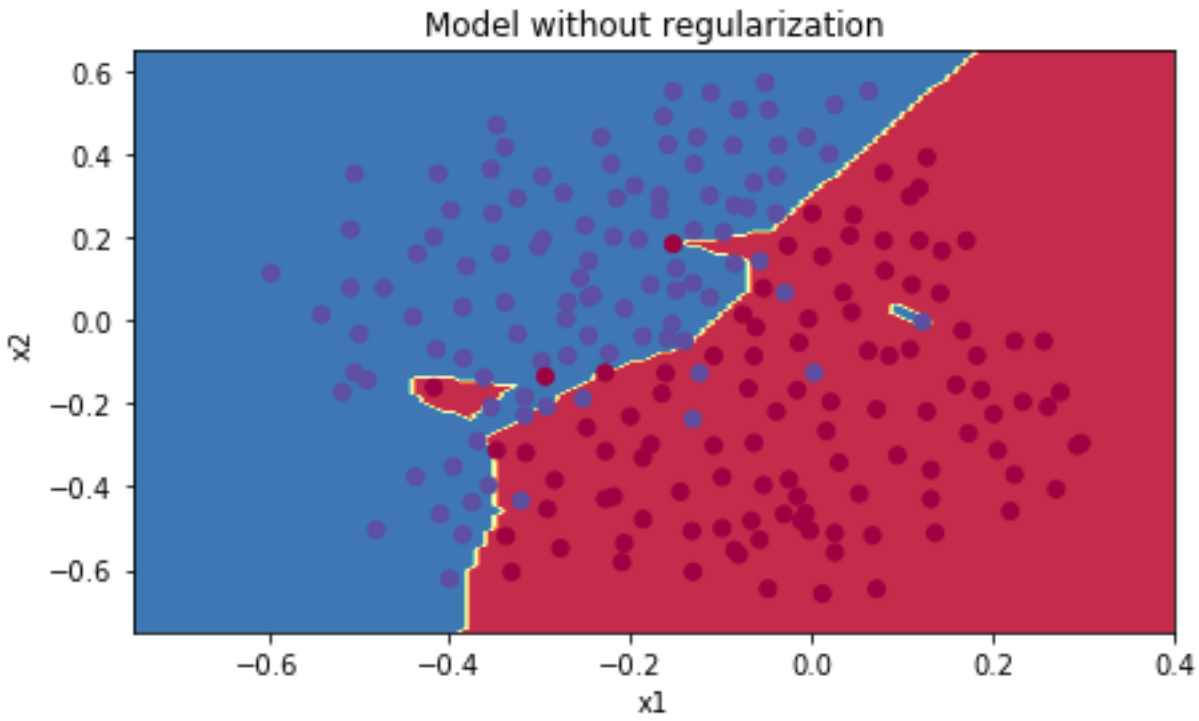

Figure 25. Make_Moons Dataset for Binary Classification. Reproduced From [101]

Here's an example of how to utilize the make_moons function in Figure 26:

```
from sklearn.datasets import make_moons

# Generate a moons dataset with 100 samples and a noise level of 0.1
X, y = make_moons(n_samples=100, noise=0.1, random_state=42)

# Display the shape of the generated dataset
print("Shape of X:", X.shape)
print("Shape of y:", y.shape)
```

Figure 26. How to use the sklearn library for generating the make_moons dataset

In this code snippet, a moons dataset is generated with 100 samples, a noise level of 0.1, and a specific random state of 42. It then proceeds to print the shape of the input features (X) and the class labels (y).

The make_moons function is highly valuable for producing synthetic datasets that exhibit non-linear patterns, enabling machine learning developers to evaluate and test machine learning algorithms on such data.

2.6.3. The Dataset of Human Faces

The dataset of human faces used in Ng's PCA course [102] is a curated collection of images showcasing various human faces. Its primary purpose is to illustrate the concepts and methodologies of PCA within the realm of face recognition and image analysis. This dataset consists of grayscale images, each representing a distinct individual's face. Each image is represented by an unrolled pixel vector with a size of 1024. To ensure consistency in the dataset, the images are typically aligned and standardized, guaranteeing uniform positioning and facial feature sizes across all samples. The faces in the dataset encompass a range of variations, including diverse poses, expressions, lighting conditions, and other factors, enabling a comprehensive analysis, as shown in Figure 27.

Figure 27. The Dataset of Human Faces. Reproduced From [102]

The human faces dataset is a valuable tool for comprehending the practical application of PCA in face recognition endeavors. Through the application of PCA on this dataset, one can observe how this technique proficiently reduces the dimensionality of facial images while retaining the most crucial features. This reduction in dimensionality facilitates efficient representation and examination of facial data, contributing to tasks such as face identification, expression analysis, and facial feature extraction.

Chapter 3
Methodology

This chapter presents three hyperparameter tuning techniques applied to various five machine learning and deep learning models, achieving superior classification accuracy while minimizing memory usage and computation time, unlike other iterative hyperparameter tuning methods such as Bayesian optimization, gradient descent, and population-based training techniques. The choice of a hyperparameter tuning approach is influenced by various factors, including the number of hyperparameters to be optimized. The selection of the most suitable method is not solely based on the count of hyperparameters, but it does play a role in determining the most effective approach [103]. Here's a general guide on how the number of hyperparameters might influence the choice of tuning method [104]:

1. **Grid Search**:
    - **Advantage**: Straightforward and intuitive. Suitable when dealing with a small number of hyperparameters.
    - **Limitation**: Becomes computationally expensive and impractical as the number of hyperparameters or their possible values increases, resulting in an exhaustive search.
2. **Random Search**:
    - **Advantage**: More efficient than grid search when dealing with a large number of hyperparameters.

- **Limitation**: Might not guarantee the optimal solution due to its random nature.

3. **Bayesian Optimization**:

    - **Advantage**: Efficient for optimizing a moderate number of hyperparameters and their respective ranges. It models the underlying function and narrows down the search space.

    - **Limitation**: Computationally intensive and may not scale well with an extremely high number of hyperparameters.

4. **Evolutionary Algorithms** (e.g., Genetic Algorithms):

    - **Advantage**: Can handle a large number of hyperparameters. It uses a population-based approach that can efficiently search the hyperparameter space.

    - **Limitation**: The computational cost might be high, and they might not converge quickly for complex problems.

5. **Gradient-Based Optimization**:

    - **Advantage**: Suitable for a large number of hyperparameters when combined with techniques like automatic differentiation.

    - **Limitation**: Depends on the differentiability of the objective function and might not be applicable in all scenarios.

6. **Hyperopt and other specialized libraries**:

- These libraries offer a variety of algorithms (TPE, Annealing, etc.) that can be efficient for different numbers of hyperparameters. They are versatile and might be suitable for varying scenarios.

Accordingly, when the learning model has a few hyperparameters, the search-based hyperparameter tuning methods are more efficient than the iterative tuning approaches. In the thesis, at most, three hyperparameters are needed to be tuned after determining the most influential hyperparameters and discarding others for each learning model based on the hyperparameter screening process. The thesis aims to show how efficient hyperparameter optimization is at increasing the efficiency of the most common and influential machine learning and deep learning models using even just a combination of conventional search-based hyperparameter tuning approaches. Therefore, the selected tuning learning models cover supervised and unsupervised machine learning models and feedforward and convolutional neural networks, proving the generality and versatility of these tuning approaches in the field of data science and artificial intelligence development. Accordingly, SVM was selected as an example of supervised machine learning models, PCA was selected as an example of unsupervised machine learning models, and FNN and PointNet CNN were selected as examples of deep learning-based neural networks. The iterative approach proves its effectiveness, particularly for CNN models with complex and high-dimensional hyperparameter space based on the criteria of hyperparameter optimization algorithms' selection in [16]. In this thesis, the focus is on tuning a maximum of three hyperparameters, so the following non-iterative tuning techniques are exploited, as mentioned in Chapter 2:

- Random search-based hyperparameter tuning.
- Hybrid random and grid search-based hyperparameter tuning.

- Hybrid random and manual search-based hyperparameter tuning.

These techniques are applied to five learning models:

- PointNet convolutional neural network.
- Standard feedforward neural network without regularization
- Standard feedforward neural network with regularization,
- Support vector machine (SVM)
- Principal component analysis (PCA).

Therefore, this chapter is partitioned into five sections to discuss the methodology and dataset utilized for each model.

The F1-score is used in this thesis as a metric in measuring the accuracy of learning models, because it provides a balance between precision and recall, which are two important aspects of model performance. It is particularly valuable when dealing with imbalanced datasets where one class significantly outnumbers the other.

Here's why the F1-score is a useful metric:

1. Balance between Precision and Recall: The F1-score is the harmonic mean of precision and recall. Precision measures the ability of the model to correctly identify positive instances, while recall measures the model's ability to find all the positive instances. By using the harmonic mean, the F1-score gives equal weight to precision and recall, ensuring that a model doesn't focus too much on one at the expense of the other.
2. Sensitivity to Imbalanced Data: In cases where one class is much smaller than the other, a high accuracy rate can be misleading if the model just predicts the majority class most of

the time. The F1-score takes into account false positives and false negatives, which are particularly important in such imbalanced scenarios.

3. Trade-off Awareness: By using the F1-score, a model's performance can be evaluated while considering the trade-off between precision and recall. Depending on the problem and its consequences, an F1-score threshold can be chosen based on specific needs. For example, in a medical diagnosis task, recall might be prioritized to ensure as many true positives as possible, even if it means accepting some false positives.

4. Robustness: The F1-score provides a single, easy-to-understand metric that combines multiple aspects of model performance. This makes it convenient for comparing different models or tuning hyperparameters.

First of all, let's discuss the main steps of each hyperparameter tuning technique mentioned above regardless of the specific settings of the five learning models and their datasets, which will be discussed later in this chapter.

3.1. The Procedures of Hyperparameter Tuning Techniques

3.1.1. Random Search-Based Hyperparameter Tuning

This proposed hyperparameter tuning technique is the random search-based coarse-to-fine hyperparameter tuning [105], an optimized version of the regular random hyperparameter search discussed above. Specifically, it is a random hyperparameter search performed through two or three stages instead of one step in the usual random search to ensure the model's generalization to the training, cross-validation, and test datasets. Two stages (coarse and fine-tuning) were conducted using the cross-validation dataset and the third stage was conducted using the test dataset for PointNet CNN only as follows:

a) Coarse Random Search-Based Hyperparameter Tuning (Cross-Validation Dataset-Based Evaluation):

- Setting hyperparameters scales:

  Initial scales were set for the tuned hyperparameters using the recommended values by experts for each learning model's type. These scales can be continuous or discrete.

- Hyperparameter search:

  The coarse random search-based tuning stage is performed by generating combinations of the corresponding randomly generated values within the manually predefined scales above. These random values satisfy the uniform distribution using the Python numpy library, as shown in Figure 28.

- Performing evaluation:

  Learning model's accuracy versus cross-validation dataset was set as an evaluation metric to narrow these scales to save time wasted at tuning the model's hyperparameters with values that can never optimize the accuracy of the learning model and then transit to the second stage (fine-tuning).

```
import random
import numpy as np
np.random.seed(3)
learning_rate = np.power(10, -4 * np.random.rand(30))
momentum = np.subtract(1,np.power(10, np.random.uniform(-3,-1,30)))
mini_batch_size = mini_batch_size = np.power(2, np.random.randint(4,6,30))
```

Figure 28. Code Snippet of Coarse Hyperparameter Tuning (Random Search)

b) Fine Random Search-Based Hyperparameter Tuning (Cross-Validation Dataset-Based Evaluation):

- Setting hyperparameters scales:

  Narrower scales were defined depending on evaluating hyperparameters' combinations in the coarse random search stage. The new scales contain the values that gave high accuracy for cross-validation in the coarse random search stage. These scales can be continuous or discrete.

- Hyperparameter search:

  The fine random search-based tuning stage is conducted by generating combinations of the corresponding randomly generated values within the new hyperparameters' scales. These random values satisfy the uniform distribution using the Python numpy library, as shown in the code snippet in Figure 29.

- Performing evaluation:

  Similar to the coarse random search-based tuning stage, the learning model's accuracy for cross-validation was used to evaluate the combinations for tuned hyperparameters.

```
import random
import numpy as np
np.random.seed(259)
learning_rate = np.power(10, np.random.uniform(-2,-4,250))
momentum = np.subtract(1,np.power(10, np.random.uniform(-1,-3,250)))
mini_batch_size = np.power(2, np.random.randint(4,6,250))
```

Figure 29. Code Snippet of Fine Hyperparameter Tuning (Random Search)

c) Test Dataset-Based Evaluation (Exclusive for PointNet Experiment):

This thesis employs the Caviar approach, as discussed in Section 2.2, for hyperparameters' values that made the model perform best according to the evaluation of the fine-tuning stage of the random search-based coarse-to-fine hyperparameter tuning technique. This work uses the overall average F1-score [106] of the mean F1-scores of the model for both pool and sea testing datasets mentioned in Section 2.5.1 as an evaluation metric to select the optimum model among the best models. F1-score should be calculated for each class (pipe, valve, or background) using the following mathematical equations:

$$\text{Precision} = \frac{TP}{TP+FP},$$

$$\text{Recall} = \frac{TP}{TP+FN},$$

$$\text{F1-score} = 2 \text{ x } \frac{Precision\ x\ Recall}{Precision+Recall},$$

where:

For each class (pipe, valve, or background):

- TP (True Positives): number of pixels correctly classified that they belong to the class.
- FP (False Positives): number of pixels incorrectly classified that they belong to the class.
- FN (False Negatives): number of pixels incorrectly classified that they do not belong to the class.

Lastly, applying the Caviar approach in the proposed work can be summarized as it is an approach that uses three measures as follows:

a. Mean F1-score of the model for pool testing dataset ([F1-score|pipe+ F1-score|valve+ F1-score|background] / 3).
b. Mean F1-score of the model for sea testing dataset ([F1-score|pipe+ F1-score|valve+ F1-score|background] / 3).
c. Overall average of mean F1-score|pool and mean F1-score|sea (test dataset-based evaluation metric).

3.1.2. Hybrid Random and Grid Search-Based Hyperparameter Tuning

Due to the effectiveness of random search at determining the best-performing hyperparameters' scales, this proposed technique merges random search with grid search. Accordingly, a random search is used as a guide for the grid search to help it focus on the well-performing regions in the hyperparameter space. Therefore, it avoids wasting computation time and storage resources at tuning the learning model's hyperparameters by ineffective values that would not make the learning model perform with high accuracy. As discussed in Section 2.1.2, grid search is very accurate at tuning small number of hyperparameters up to three hyperparameters.

As shown above in Figure 30, the hyperparameters of PointNet CNN were tuned through three stages:

a) Random Search-Based Hyperparameter Tuning (Cross-Validation Dataset-Based Evaluation):

It has the same procedures as the first stage of random search-based hyperparameter tuning in part (a) in Section 3.1.1, as follows:

- Setting hyperparameters scales:

  Initial scales were set for the tuned hyperparameters using the recommended values by experts for each learning model's type. These scales can be continuous or discrete.

- Hyperparameter search:

  Narrower scales were defined depending on evaluating hyperparameters' combinations in the random search stage. However, the mini-batch size remained unchanged in this stage. The new scales contain the values that gave high accuracy for cross-validation in the random search stage.

- Performing evaluation:

  The learning model's accuracy versus cross-validation dataset was set as an evaluation metric to narrow these scales to save time wasted at tuning the model's hyperparameters with values that can never optimize the accuracy of the learning model and then transit to the second stage (grid search-based tuning).

b) Grid Search-Based Hyperparameter Tuning (Cross-Validation Dataset-Based Evaluation):

- Setting hyperparameters scales:

  Narrower scales were defined depending on evaluating hyperparameters' combinations in the random search stage. The new scales contain the values that gave high accuracy for cross-validation in the random search stage. These scales must be discrete, so the scales resulting from the random search-based tuning must be discretized.

- Hyperparameter search:

  All possible combinations of the values within the new discretized scales of tuned hyperparameters were considered. For instance, if hyperparam1 ∈ {x1, x2} and hyperparam2 ∈ {y1, y2, y3}, then six combinations should be taken into consideration.

- Performing evaluation:

  Similar to the random search-based tuning stage, learning model's accuracy for cross-validation was used to evaluate all combinations coming from the grid search.

c) Test Dataset-Based Evaluation (Exclusive for PointNet Experiment):

This thesis employs the Caviar approach, as discussed in Section 2.2, for hyperparameters' values that made the model perform best according to the evaluation of the random search-based tuning stage. This work uses the overall average F1-score [106] of the mean F1-scores of the model for both pool and sea testing datasets mentioned in Section 2.5.1, as an evaluation metric to select the optimum model among the best models. F1-score should be calculated for each class (pipe, valve, or background) using the following mathematical equations:

- F1-score should be calculated for each class (pipe, valve, or background) using the following mathematical equations:

$$\text{Precision} = \frac{TP}{TP+FP},$$

$$\text{Recall} = \frac{TP}{TP+FN},$$

$$\text{F1-score} = 2 \text{ x } \frac{Precision\ x\ Recall}{Precision+Recall},$$

where:

For each class (pipe, valve, or background):

- ✓ TP (True Positives): number of pixels correctly classified that they belong to the class.
- ✓ FP (False Positives): number of pixels incorrectly classified that they belong to the class.
- ✓ FN (False Negatives): number of pixels incorrectly classified that they do not belong to the class.

- Lastly, applying the Caviar approach in this thesis can be summarized as it is an approach that uses three measures as follows:
  - ✓ Mean F1-score of the model for pool testing dataset ([F1-score|pipe+ F1-score|valve+ F1-score|background] / 3).
  - ✓ Mean F1-score of the model for sea testing dataset ([F1-score|pipe+ F1-score|valve+ F1-score|background] / 3).
  - ✓ Overall average of mean F1-score|pool and mean F1-score|sea (test dataset-based evaluation metric).

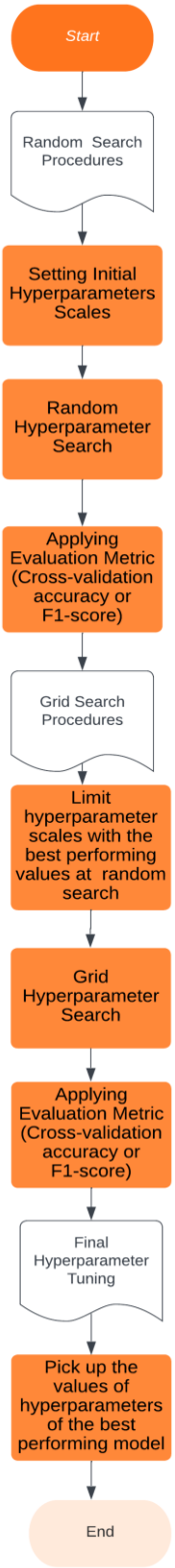

Figure 30. Hybrid Random and Grid Search-Based Hyperparameter Tuning Procedures

3.1.3. Hybrid Random and Manual Search-Based Hyperparameter Tuning

Due to the effectiveness of random search at determining the best effective areas in the hyperparameter space, this proposed technique merges random search with manual search. Accordingly, a random search is used as prior knowledge for the manual search to facilitate it by searching through narrow and effective hyperparameters' scales. Therefore, computation time and storage resources would be spent in a more effective manner than the regular manual search without any prior knowledge. Without prior knowledge about the recommended hyperparameters' scales, the manual search would be a tedious task, as discussed in Section 2.1.1. The manual search is an evitable option when human or computation resources are scarce. In this technique as shown in Figure 31, the hyperparameters of PointNet CNN are tuned through three stages:

a) Random Search-Based Hyperparameter Tuning (Cross-Validation Dataset-Based Evaluation):

It has the same procedures as the first stage of random search-based hyperparameter tuning in part (a) in Section 3.1.1 and the part (a) in Section 3.1.2, as follows:

- Setting hyperparameters scales:

  Initial scales were set for the tuned hyperparameters using the recommended values by experts for each learning model's type. These scales can be continuous or discrete.

- Hyperparameter search:

  The random search-based tuning stage is performed by generating combinations of the corresponding randomly generated values within the manually predefined

scales above. These random values satisfy the uniform distribution using the Python numpy library.

- Performing evaluation:

  The learning model's accuracy versus cross-validation dataset was set as an evaluation metric to narrow these scales to save time wasted at tuning the model's hyperparameters with values that can never optimize the accuracy of the learning model and then transit to the second stage (manual search-based tuning).

b) Manual Search-Based Hyperparameter Tuning (Cross-Validation Dataset-Based Evaluation):

- Setting hyperparameters scales:

  Narrower scales were defined depending on evaluating hyperparameters' combinations in the random search stage. The new scales contain the values that gave high accuracy for cross-validation in the random search stage. These scales must be discrete, so the scales resulted from the random search-based tuning must be discretized.

- Hyperparameter search:

  The hyperparameters' scales obtained by random search-based tuning were discretized. The hyperparameters were tuned by walking through the discretized scales one at a time. While a hyperparameter is being tuned by increasing its value within its discrete scale, the other hyperparameters were kept with constant values. The tuning process for each hyperparameter should stop when the learning model starts to give a lower cross-validation accuracy than before.

c) Test Dataset-Based Evaluation (Exclusive for PointNet Experiment):

This thesis uses the Caviar approach, as discussed in Section 2.2, for hyperparameters' values that made the model perform best according to the evaluation of the random search-based tuning stage. This work uses the overall average F1-score [106] of the mean F1-scores of the model for both pool and sea testing datasets mentioned in Section 2.5.1. as an evaluation metric to select the optimum model among the best models. F1-score should be calculated for each class (pipe, valve, or background) using the following mathematical equations:

$$\text{Precision} = \frac{TP}{TP+FP},$$

$$\text{Recall} = \frac{TP}{TP+FN},$$

$$\text{F1-score} = 2 \text{ x } \frac{Precision\ x\ Recall}{Precision+Recall},$$

where:

For each class (pipe, valve, or background):

- ✓ TP (True Positives): number of pixels correctly classified that they belong to the class.
- ✓ FP (False Positives): number of pixels incorrectly classified that they belong to the class.
- ✓ FN (False Negatives): number of pixels incorrectly classified that they do not belong to the class.

- Lastly, applying the Caviar approach in the proposed work can be summarized as it is an approach that uses three measures as follows:

- ✓ Mean F1-score of the model for pool testing dataset ([F1-score|pipe+ F1-score|valve+ F1-score|background] / 3).
- ✓ Mean F1-score of the model for sea testing dataset ([F1-score|pipe+ F1-score|valve+ F1-score|background] / 3).
- ✓ Overall average of mean F1-score|pool and mean F1-score|sea (test dataset-based evaluation metric).

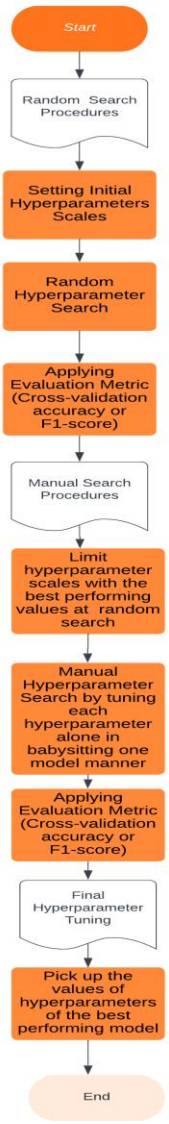

Figure 31. Hybrid Random and Manual Search-Based Hyperparameter Tuning Procedures

3.2. Hyperparameter Screening Procedures

In the field of machine learning and hyperparameter optimization, screening refers to the procedure of identifying and eliminating hyperparameters that do not notably influence the model's performance, as assessed by metrics such as the F1 score or cross-validation score. This process is commonly carried out to streamline the model, lower the computational expenses associated with hyperparameter optimization, and enhance the overall efficiency of the machine learning workflow.

Here is an outline of how the screening process was conducted in the thesis:

1. **Initial Hyperparameter Selection**: Initially, you compile a set of hyperparameters that you believe could be relevant to your model. This initial set can be extensive, especially when dealing with intricate models.

2. **Hyperparameter Optimization**: You employ methods like grid search, random search, or Bayesian optimization to explore various hyperparameter combinations and assess the model's performance using a chosen evaluation metric, such as the F1 score or cross-validation score.

3. **Performance Assessment**: Following the training and evaluation of the model for each set of hyperparameters, you amass data on how each set of hyperparameters has performed. This data enables you to compare their impact on the model's performance.

4. **Hyperparameter Screening**: During the screening phase, you pinpoint hyperparameters that exert minimal or insignificant influence on the chosen performance metric. These hyperparameters can either be excluded from consideration or set to default values,

simplifying the model and narrowing down the search space for hyperparameter optimization.

5. **Refinement**: Your attention can then be concentrated on the remaining hyperparameters, which have a more pronounced impact on the model's performance. You can allocate additional optimization resources to fine-tune these hyperparameters.

By sieving out non-significant hyperparameters, you streamline the hyperparameter tuning process, enabling you to allocate more resources to the optimization of crucial hyperparameters. This approach strikes a balance between identifying an effective model and conserving computational resources. It's important to exercise caution during screening, as seemingly unimportant hyperparameters may, at times, have non-linear or interactive effects with other hyperparameters, and their impact on the model's performance may not be immediately evident.

3.3. Experiments

3.3.1. PointNet Convolutional Neural Network

The learning rate, momentum, and mini-batch size were the tuned hyperparameters of the PointNet CNN which was fed by the point-cloud dataset mentioned in Section 2.5.1. This dataset contains images of pipes, valves, and backgrounds. The PointNet architecture takes point clouds as input and assigns a class label to each point. During training, the network is fed with ground truth point clouds where each point is labeled with its corresponding class. The PointNet employs a softmax cross-entropy loss along with an Adam optimizer. The number of cloud points used for each training example is 4096 points. The decay rate for batch normalization starts at 0.5 and gradually increases to 0.99. Additionally, a dropout with a keep-prob ratio of 0.7 is applied on the last fully connected layer before predicting class scores. By employing the early stopping technique described in detail in Section 2.4.1, the training process becomes more

generalized and avoids overfitting. This experiment aims to improve PointNet's accuracy at 3D semantic segmentation (pixel-wise classification) through the following procedures of hyperparameter tuning in Table 1.

Table 1. Tuning The Hyperparameters of PointNet CNN

Hyperparameter Tuning Technique			PointNet CNN
Random Search-Based Tuning	Coarse Random Search	Setting hyperparameters' scales	Based on the recommendations of Ng in [55] and Goodfellow et al. in [10] for tuning deep learning models, initial scales were set as follows: • Learning rate: [0.0001,1] • Momentum: [0.9,0.999] • Mini-batch size: {16,32}
		Hyperparameter search	Thirty ternary combinations were generated randomly based on the three manually predefined scales above.
		Evaluation	Based on Cross-validation (10% of the pool dataset) accuracy is the criterion of evaluation.
	Fine Random Search	Setting hyperparameters' scales	New and narrower three scales were deduced by the evaluation of coarse random search.
		Hyperparameter Search	250 ternary combinations were generated randomly based on the new scales.

		Evaluation	Based on Cross-validation (10 % of the pool dataset) accuracy, the 250 ternary combinations were evaluated in parallel according to the Caviar approach.
	Test Dataset-Based Evaluation	Final Evaluation	Based on the test dataset (10 % of pool dataset + sea dataset), the best 25 performing models in fine random search stage were evaluated based on F1-score: a. Mean F1-score of the model for pool testing dataset ([F1-score|pipe+ F1-score|valve+ F1-score|background] / 3). b. Mean F1-score of the model for sea testing dataset ([F1-score|pipe+ F1-score|valve+ F1-score|background] / 3). c. Overall average of mean F1-score|pool and mean F1-score|sea (Final evaluation metric).
Hybrid Random and Grid Search-	Random Search Tuning	Setting Hyperparameters' Scales	Based on the recommendations of Ng in [54] and Goodfellow et al. in [10] for tuning deep learning models, initial scales were set as follows:

			• Learning rate: [0.0001,1] • Momentum: [0.9,0.999] • Mini-batch size: {16,32}
		Hyperparameter Search	Thirty ternary combinations were generated randomly based on the three manually predefined scales above.
		Evaluation	Based on Cross-validation (10% of pool dataset) accuracy is the criterion of evaluation.
	Grid Search Tuning	Setting Hyperparameters' Scales	Narrower three scales were obtained by random search. These scales were discretized before applying grid search.
		Hyperparameter Search	All possible combinations of the values within the new discretized scales of learning rate and momentum. The mini-batch size was kept with the same scale. Random values as many as these combinations were generated for mini-batch size.
		Evaluation	Cross-validation (10 % of pool dataset) accuracy is the criterion of evaluation.
	Test Dataset Evaluation		Based on test dataset (10 % of pool dataset + sea dataset), the best performing models in random search stage were evaluated based on

		Final Evaluation	F1-score: a. Mean F1-score of the model for pool testing dataset ([F1-score|pipe+ F1-score|valve+ F1-score|background] / 3). b. Mean F1-score of the model for sea testing dataset ([F1-score|pipe+ F1-score|valve+ F1-score|background] / 3). c. Overall average of mean F1-score|pool and mean F1-score|sea (Final evaluation metric).
Hybrid Random and Manual Search Tuning	Random Search	Setting Hyperparameters' Scales	Based on the recommendations of Ng in [55] and Goodfellow et al. in [10] for tuning deep learning models, initial scales were set as follows: • Learning rate: [0.0001,1] • Momentum: [0.9,0.999] • Mini-batch size: {16,32}
		Hyperparameter Search	Thirty ternary combinations were generated randomly based on the three manually predefined scales above.

		Evaluation	Based on Cross-validation (10% of pool dataset) accuracy is the criterion of evaluation.
	Manual Search	Setting Hyperparameters' Scales	Narrower three scales were obtained by random search. These scales were discretized before applying manual search.
		Hyperparameter Search	The three hyperparameters were tuned by walking through the discretized scales one at a time. While a hyperparameter was being tuned by increasing its value within its discrete scale, the other hyperparameters were kept with constant values. The tuning process for each hyperparameter should stop when the learning model starts to give a lower cross-validation accuracy than before.
	Test Dataset-Evaluation	Final Evaluation	Based on test dataset (10 % of pool dataset + sea dataset), the best performing model in random search stage is evaluated based on F1-score: a) Mean F1-score of the model for pool testing dataset ([F1-score|pipe+ F1-score|valve+ F1-score|background] / 3).

			b) Mean F1-score of the model for sea testing dataset ([F1-score|pipe+ F1-score|valve+ F1-score|background] / 3). c) Overall average of mean F1-score|pool and mean F1-score|sea (Final evaluation metric).

3.3.2. Standard Feedforward Neural Network Without Regularization

A three-layer standard feedforward neural network was exploited to apply the three proposed hyperparameter tuning techniques on the following network's hyperparameters:

- Learning rate.
- Momentum ($\beta 1$).
- Mini-batch size.

The network consists of an input layer, two hidden layers, and an output layer, as shown in Figure 32. The input layer has two features, the first hidden layer has five activation units, the second hidden layer has two activation units, and the output layer has one activation unit because the network is used for the binary classification. ReLU activation function is used in the whole network except for the output layer that has Sigmoid activation function. The network was fed by make_moons dataset provided by scikit-learn library (sklearn), as mentioned in Section 2.5.2. The distribution of the features' values versus the output classification resembles two half-moons or crescents in this dataset.

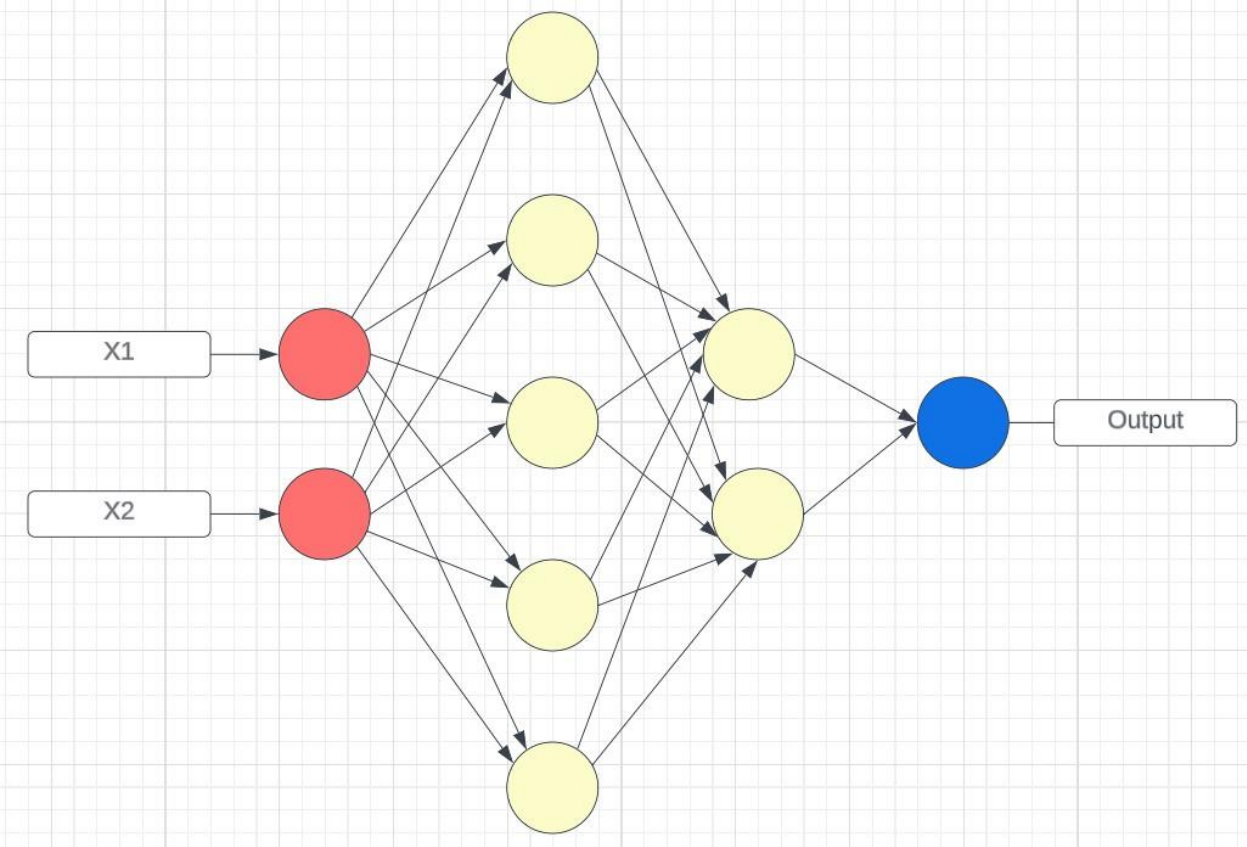

Figure 32. Three-layer Feedforward Neural Network

This experiment aims to improve the network's accuracy at the binary classification through the following procedures of hyperparameter tuning in Table 2.

Table 2. Tuning the hyperparameters of Standard Feedforward Neural Network.

Hyperparameter Tuning Technique			PointNet CNN
		Setting hyperparameters' scales	Based on the recommendations of Ng in [55] and Goodfellow et al. in [10] for tuning deep learning models, initial scales were set as follows: • Learning rate: [0.0001,1] • Momentum: [0.9,0.999]

Random Search-Based Tuning	Coarse Random Search		• Mini-batch size: {16,32}
		Hyperparameter search	50 ternary combinations were generated randomly based on the three manually predefined scales above.
		Evaluation	Based on Cross-validation (10% of pool dataset) accuracy is the criterion of evaluation.
	Fine Random Search	Setting hyperparameters' scales	New and narrower three scales were deduced by the evaluation of coarse random search.
		Hyperparameter Search	500 ternary combinations were generated randomly based on the new scales.
		Evaluation	Based on Cross-validation (10 % of the pool dataset) accuracy, the 500 ternary combinations were evaluated in parallel according to the Caviar approach.
	Random Search Tuning	Setting Hyperparameters' Scales	Based on the recommendations of Ng in [55] and Goodfellow et al. in [10] for tuning deep learning models, initial scales were set as follows: • Learning rate: [0.0001,1] • Momentum: [0.9,0.999] • Mini-batch size: {16,32}
		Hyperparameter Search	Fifty ternary combinations were generated randomly based on the three manually predefined

Hybrid Random and Grid Search-Based Tuning			scales above.
		Evaluation	Based on Cross-validation accuracy is the criterion of evaluation.
	Grid Search Tuning	Setting Hyperparameters' Scales	Narrower three scales were obtained by random search. These scales were discretized before applying grid search.
		Hyperparameter Search	All possible combinations of the values within the new discretized scales of learning rate and momentum. The mini-batch size was kept with the same scale. Random values as many as these combinations were generated for mini-batch size.
		Evaluation	Cross-validation accuracy is the criterion of evaluation.
	Random Search	Setting Scales	Based on the recommendations of Ng in [55] and Goodfellow et al. in [10] for tuning deep learning models, initial scales were set as follows: • Learning rate: [0.0001,1] • Momentum: [0.9,0.999] • Mini-batch size: {16,32}
		Hyperparameter Search	Fifty ternary combinations were generated randomly based on the three manually predefined scales above.

Hybrid Random and Manual Search-Based Tuning		Evaluation	Based on Cross-validation accuracy is the criterion of evaluation.
	Manual Search	Setting Hyperparameters' Scales	Narrower three scales were obtained by random search. These scales were discretized before applying manual search.
		Hyperparameter Search	The three hyperparameters were tuned by walking through the discretized scales one at a time. While a hyperparameter was being tuned by increasing its value within its discrete scale, the other hyperparameters were kept with constant values. The tuning process for each hyperparameter should stop when the learning model starts to give a lower cross-validation accuracy than before.

3.3.3. Standard Feedforward Neural Network With Regularization

A three-layer standard feedforward neural network was exploited to apply the three proposed hyperparameter tuning techniques on the following network's hyperparameters:

- Learning rate.
- Regularization parameter (λ).

The network consists of an input layer, two hidden layers, and an output layer, as shown in Figure 33. The input layer has two features, the first hidden layer has 20 activation units, the second hidden layer has three activation units, and the output layer has one activation unit because the network is used for the binary classification. ReLU activation function is used in the

whole network except for the output layer that has Sigmoid activation function. The network was fed by make_moons dataset provided by scikit-learn library (sklearn), as mentioned in Section 2.5.2. The distribution of the features' values versus the output classification resembles two half-moons or crescents in this dataset.

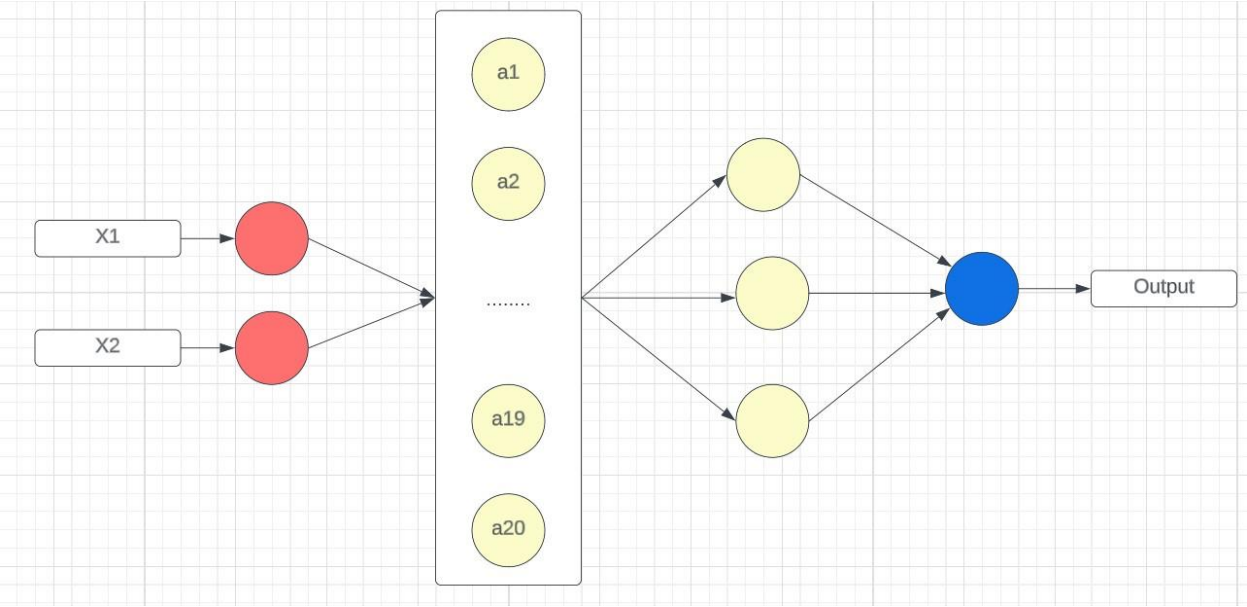

Figure 33. Three-layer Feedforward Neural Network With Regularization

This experiment aims to improve the network's accuracy at the binary classification through the following procedures of hyperparameter tuning in Table 3.

Table 3. Tuning the hyperparameters of a standard feedforward neural network with regularization

Hyperparameter Tuning Technique			PointNet CNN
Random Search Tuning	Coarse Random Search	Setting hyperparameters' scales	Based on the recommendations of Ng in [55] and Goodfellow et al. in [10] for tuning deep learning models, initial scales were set as follows:

			• Learning rate: [0.0001,1] • Regularization parameter (λ): [0.1,10]
		Hyperparameter search	50 ternary combinations were generated randomly based on the two manually predefined scales above.
		Evaluation	Based on Cross-validation accuracy is the criterion of evaluation.
	Fine Random Search	Setting hyperparameters' scales	New and narrower two scales were deduced by the evaluation of coarse random search.
		Hyperparameter Search	500 ternary combinations were generated randomly based on the new scales.
		Evaluation	Based on Cross-validation accuracy is the criterion of evaluation.
Hybrid Random and Grid Search	Random Search	Setting Scales	Based on the recommendations of Ng in [54] and Goodfellow et al. in [10] for tuning deep learning models, initial scales were set as follows: • Learning rate: [0.0001,1] • Regularization parameter (λ): [0.1,10]
		Hyperparameter Search	50 ternary combinations were generated randomly based on the two manually predefined scales above.
		Evaluation	Based on Cross-validation accuracy is the criterion of evaluation.

	Grid Search Tuning	Setting Hyperparameters' Scales	Narrower two scales were obtained by random search. These scales were discretized before applying grid search.
		Hyperparameter Search	All possible combinations of the values within the new discretized scales of learning rate and regularization parameter (λ). The mini-batch size was kept with the same scale. Random values as many as these combinations were generated for mini-batch size.
		Evaluation	Cross-validation accuracy is the criterion of evaluation.
Hybrid Random and Manual Search-Based Tuning	Random Search	Setting Scales	Based on the recommendations of Ng in [55] and Goodfellow et al. in [10] for tuning deep learning models, initial scales were set as follows: • Learning rate: [0.0001,1] • Regularization parameter (λ): [0.1,10]
		Hyperparameter Search	50 binary combinations were generated randomly based on the two manually predefined scales above.
		Evaluation	Based on Cross-validation accuracy is the criterion of evaluation.
		Setting Scales	Narrower two scales were obtained by random search. These scales were discretized before

			applying manual search to be ready for the manual search in the following step.
	Manual Search	Hyperparameter Search	The two hyperparameters were tuned by walking through the discretized scales one at a time. While a hyperparameter was being tuned by increasing its value within its discrete scale, the other hyperparameters were kept with constant values. The tuning process for each hyperparameter should stop when the learning model starts to give a lower cross-validation accuracy than before.

3.3.4. Support Vector Machine (SVM)

A support vector machine (SVM) model was exploited to apply the three proposed hyperparameter tuning techniques on the following model's hyperparameters:

- Regularization parameter (C=1/λ).
- sigma (σ).

The SVM model aims to optimize the binary classification of the make_moons dataset provided by scikit-learn library (sklearn), as mentioned in Section 2.5.2. The distribution of the features' values versus the output classification resembles two half-moons or crescents in this dataset. The optimization hypothesis, as mentioned in Section 2.4.4, is as follows:

$$\text{Cost}(\theta) = C \sum_{i=1}^{m} \left[y^{(i)}\, cost_1\left(\theta^T f^{(i)}\right) + \left(1 - y^{(i)}\right) cost_0\left(\theta^T f^{(i)}\right) \right] + \frac{1}{2} \sum_{j=1}^{n} {\theta_j}^2$$

$$f_j^{(i)} = \text{kernel}\left(x^{(i)}, l^{(j)}\right) = \exp\left(-\frac{\left\| x^{(i)} - l^{(j)} \right\|^2}{2\,\sigma^2}\right)$$

This experiment aims to improve the SVM model's accuracy at the binary classification through the following procedures of hyperparameter tuning in Table 4.

Table 4. Tuning the hyperparameters of SVM model

Hyperparameter Tuning Technique			PointNet CNN
Random Search-Based Tuning	Coarse Random Search	Setting hyperparameters' scales	Based on the recommendations of Ng in [55] and Goodfellow et al. in [10] for tuning deep learning models, initial scales were set as follows: • Regularization parameter (C=1/λ): [0.01,30] • sigma (σ): [0.01,30]
		Hyperparameter search	50 binary combinations were generated randomly based on the two manually predefined scales above for Regularization parameter (C) and sigma (σ).
		Evaluation	Based on Cross-validation accuracy is the criterion of evaluation.
	Fine Random Search	Setting hyperparameters' scales	New and narrower two scales were deduced by the evaluation of coarse random search.
		Hyperparameter Search	500 binary combinations were generated randomly based on the new scales.
		Evaluation	Based on Cross-validation accuracy is the criterion of evaluation.

Hybrid Random and Grid Search-Based Tuning	Random Search Tuning	Setting Hyperparameter Scales	Based on the recommendations of Ng in [55] and Goodfellow et al. in [10] for tuning deep learning models, initial scales were set as follows: • Regularization parameter (C=1/λ): [0.01,30] • sigma (σ): [0.01,30]
		Hyperparameter Search	50 binary combinations were generated randomly based on the two manually predefined scales above.
		Evaluation	Based on Cross-validation accuracy is the criterion of evaluation.
	Grid Search Tuning	Setting Hyperparameters' Scales	Narrower two scales were obtained by random search. These scales were discretized before applying grid search.
		Hyperparameter Search	All possible combinations of the values within the new discretized scales of C-hyperparameter and sigma.
		Evaluation	Cross-validation accuracy is the criterion of evaluation.
Hybrid Random and Manual		Setting Hyperparameters' Scales	Based on the recommendations of Ng in [55] and Goodfellow et al. in [10] for tuning deep learning models, initial scales were set as follows: • Regularization parameter (C=1/λ): [0.01,30] • sigma (σ): [0.01,30]

	Random Search	Hyperparameter Search	50 binary combinations were generated randomly based on the two manually predefined scales above.
		Evaluation	Based on Cross-validation accuracy is the criterion of evaluation.
	Manual Search	Setting Hyperparameters' Scales	Narrower two scales were obtained by random search to optimize the performance of the manual search. These scales were discretized before applying manual search.
		Hyperparameter Search	The two hyperparameters were tuned by walking through the discretized scales one at a time. While a hyperparameter was being tuned by increasing its value within its discrete scale, the other hyperparameters were kept with constant values. The tuning process for each hyperparameter should stop when the learning model starts to give a lower cross-validation accuracy than before.

3.3.5. Principal Component Analysis (PCA)

A principal component analysis (PCA) model was exploited to apply the three proposed hyperparameter tuning techniques on the following model's hyperparameter:

- Number of PCA components (K).

The PCA model aims to reduce the dimensionality of the dataset mentioned Section 2.5.3 for data compression. Data compression should be conducted while retaining at least 90 % of the variance of the original dataset before the dimensionality reduction. The following equations are the basis of PCA, as discussed in Section 2.4.5 for data compression

$$\text{Sigma} = \text{Covariance Matrix} = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)})\ (x^{(i)})^{T}$$

$$[\text{U S V}] = \text{svd (Sigma)}$$

$$\text{U}_{reduced} = \text{U [1: END, 1: K]}$$

This experiment aims to select the minimum number of PCA components that can keep the possession of at least 90 % of the variance of the original dataset before the dimensionality reduction through the following procedures of hyperparameter tuning in Table 5

Table 5. Tuning the hyperparameters of PCA model

Hyperparameter Tuning Technique			PointNet CNN
Random Search-Based Tuning	Coarse Random Search	Setting hyperparameters' scales	Initial scale was set as follows: • Number of PCA components:{1,2, …,1024}
		Hyperparameter search	50 unary combinations were generated randomly based on the manually predefined scale above to select the best-performing ones among them
		Evaluation	Selecting the minimum number of the PCA components (K) that retain at least 90 % of the variance of the original dataset.

	Fine Random Search	Setting hyperparameters' scales	A new and narrower scale was deduced by the evaluation of coarse random search.
		Hyperparameter Search	50 unary combinations were generated randomly based on the new scale.
		Evaluation	Selecting the minimum number of the PCA components (K) that retain at least 90 % of the variance of the original dataset.
Hybrid Random and Grid Search	Random Search Tuning	Setting Hyperparameters' Scales	Initial scale was set as follows: • Number of PCA components:{1,2, …,1024}
		Hyperparameter Search	50 unary combinations were generated randomly based on the manually predefined scale above.
		Evaluation	Selecting the minimum number of the PCA components (K) that retained at least 90 % of the variance of the original dataset. This ensures maintaining the visibility of human faces.
	Grid Search Tuning	Setting Hyperparameters' Scales	A new and narrower scale was deduced by the evaluation of coarse random search.

		Hyperparameter Search	50 unary combinations through the new discretized scale of the number of PCA components (K).
		Evaluation	Selecting the minimum number of the PCA components that retain at least 90 % of the variance of the original dataset.
Hybrid Random and Manual Search	Random Search	Setting Hyperparameters' Scales	Initial scale was set as follows: • Number of PCA components: {1,2, …,1024}
		Hyperparameter Search	50 unary combinations were generated randomly based on the two manually predefined scales above.
		Evaluation	Selecting the minimum number of the PCA components that retain at least 90 % of the variance of the original dataset.
	Manual Search	Setting Hyperparameters' Scales	A narrower scale was obtained by random search. This scale was discretized before applying manual search.
		Hyperparameter Search	The hyperparameter was tuned by walking through the discretized scale. The tuning process of the hyperparameter should stop when at least 90 % of the variance of the original dataset was retained

Chapter 4

Results and Discussion

4.1. PointNet Convolutional Neural Network

Based upon Section 3.2.1, the work was carried using three distinct hyperparameter tuning techniques: the random search-based tuning, the hybrid random and grid search-based tuning stage, and the hybrid random and manual search-based tuning.

4.1.1. Random Search-Based Hyperparameter Tuning Technique

4.1.1.1 Random Search-Based Coarse Tuning

As discussed in Section 3.2.1, the recommended scales by experts for mini-batch size, momentum, and learning rate are {16,32}, [0.9,0.999], and [0.0001,1], respectively. Therefore, 30 models, with distinct and random values for each hyperparameter, were composed by producing 30 ternary combinations from the three scales, as visualized in Figure 34.

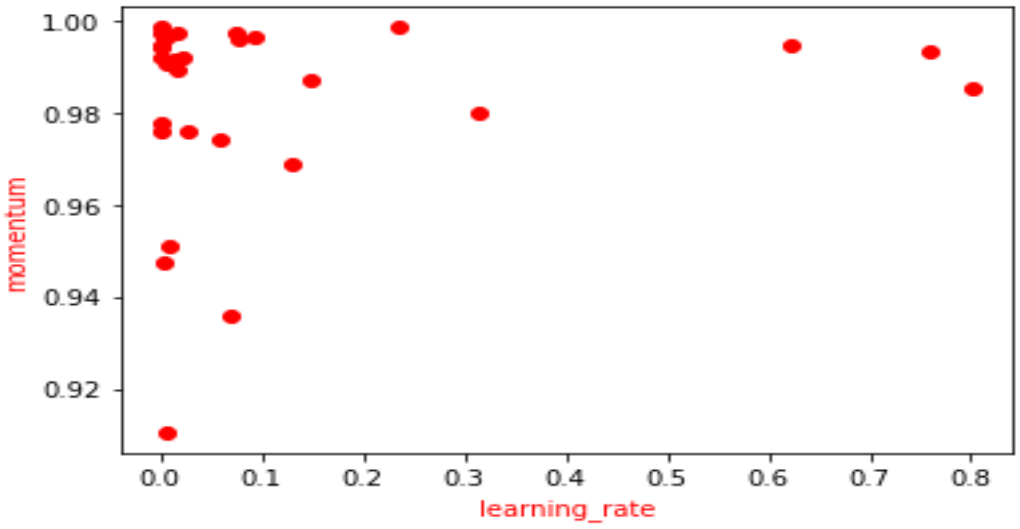

Figure 34. Random Search-Based Coarse Hyperparameter Tuning Stage

Subsequently, the model's classification accuracy for the cross-validation dataset was used to evaluate the 30 models, as shown in Figure 35 and Figure 36. The hyperparameters' values, that made the PointNet model achieve an accuracy of at least 99%, were exploited to shrink the aforementioned initial scales and set narrower ones that can optimize the learning model.

Table 6. Corse-to-Fine Tuning: Coarse-Tuning Evaluation

Batch_size	Learning_rate	Momentum	cross_validation accuracy (%)
16	0.060	0.999	96.000
32	0.031	0.977	95.667
16	0.428	0.988	95.667
32	0.129	0.997	95.667
16	0.015	0.978	95.667
16	0.092	0.912	95.667
32	0.073	0.976	95.667
32	0.068	0.969	95.667
16	0.073	0.995	95.333
32	0.024	0.972	95.333
32	0.036	0.929	95.333
16	0.013	0.991	95.333
16	0.315	0.974	95.000
16	0.017	0.972	95.000
32	0.148	0.969	95.000
16	0.017	0.976	95.000
16	0.802	0.999	95.000
16	0.126	0.952	94.667
32	0.759	0.994	94.667
32	0.077	0.987	94.667
16	0.006	0.997	94.333
32	0.236	0.991	94.333
32	0.084	0.952	94.333
32	0.069	0.989	94.333
16	0.028	0.998	94.333
16	0.077	0.986	94.333
16	0.006	0.997	94.333
16	0.006	0.997	94.333
16	0.004	0.992	94.000
16	0.622	0.976	94.000

16	0.007	0.984	94.000
32	0.015	0.979	94.000
16	0.022	0.997	93.667
16	0.003	0.995	93.667
32	0.005	0.981	93.667
16	0.009	0.999	93.667
16	0.015	0.998	93.667
16	0.002	0.996	93.333
16	0.002	0.980	92.333
32	0.002	0.974	90.000
32	0.001	0.997	87.667
32	0.002	0.987	87.333
32	0.001	0.999	79.667
16	0.000	0.995	75.000
16	0.000	0.998	74.333
32	0.000	0.920	73.333
32	0.000	0.933	71.667
32	0.000	0.991	69.667
16	0.000	0.998	69.000
32	0.000	0.998	67.333

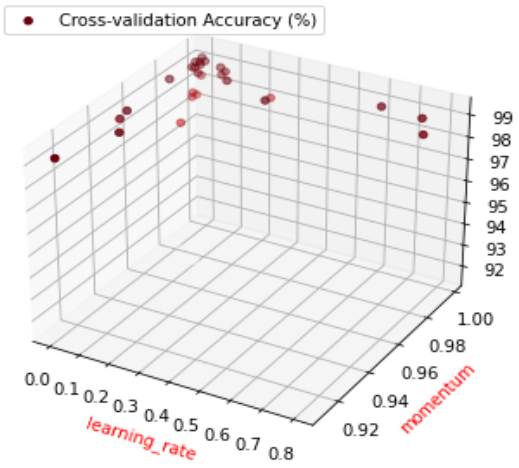

Figure 35. Random Search-Based Coarse Tuning Evaluation

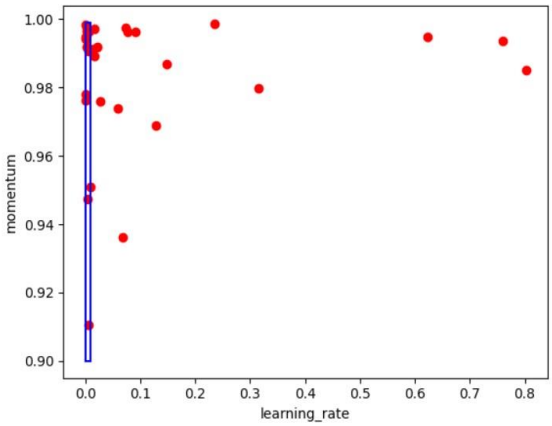

Figure 36. Random Search-Based Coarse Tuning Evaluation: Shrunk scales represented by a blue rectangle. The mini-batch size remained unchanged {16,32}

4.1.1.2 Random Search-Based Fine Tuning

According to the results in Section 4.1.1.1, the updated shrunk scales for mini-batch size, momentum, and learning rate were {16,32}, [0.9,0.999], and [0.0001,0.01], respectively. Thus, 250 models, with distinct values for each hyperparameter, were formed by generating 250 ternary combinations from the three updated scales, as shown in Figure 37.

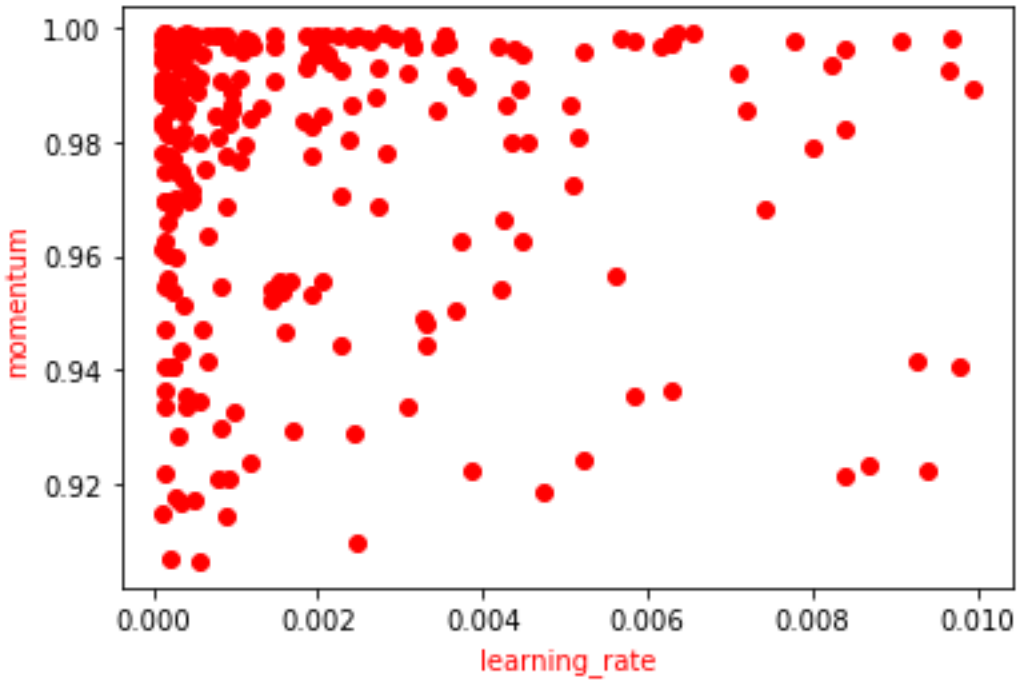

Figure 37. Random Search-Based Fine Hyperparameter Tuning

After that, the model's classification accuracy for the cross-validation dataset was used to evaluate the 250 models. The 25 (10% of 250 models) best-fit models shown in Table 7 and Figure 38 that achieved the highest classification accuracy for the cross-validation dataset were passed to the following phase (test dataset-based evaluation) to conduct the final evaluation discussed in Section 4.1.3.

Table 7. Random Search-Based Fine-Tuning Evaluation

Batch_size	Learning_rate	Momentum	Cross-Validation accuracy (%)
32	**0.002**	**0.985**	**99.341**
32	0.002	0.970	99.303
16	0.001	0.985	99.274
32	0.004	0.963	99.261
32	0.004	0.951	99.256
16	0.002	0.996	99.254
32	0.001	0.983	99.252
16	0.000	0.944	99.251
16	0.004	0.980	99.248
16	0.001	0.999	99.247
32	0.002	0.998	99.244
32	0.002	0.930	99.238
16	0.001	0.999	99.234
32	0.001	0.930	99.233
16	0.001	0.975	99.232
16	0.004	0.995	99.224
16	0.001	0.991	99.222
32	0.002	0.999	99.221
16	0.002	0.999	99.217
16	0.005	0.980	99.217
32	0.004	0.997	99.217
32	0.002	0.993	99.216
32	0.002	0.944	99.216
16	0.002	0.987	99.213
16	0.0003	0.960	99.209

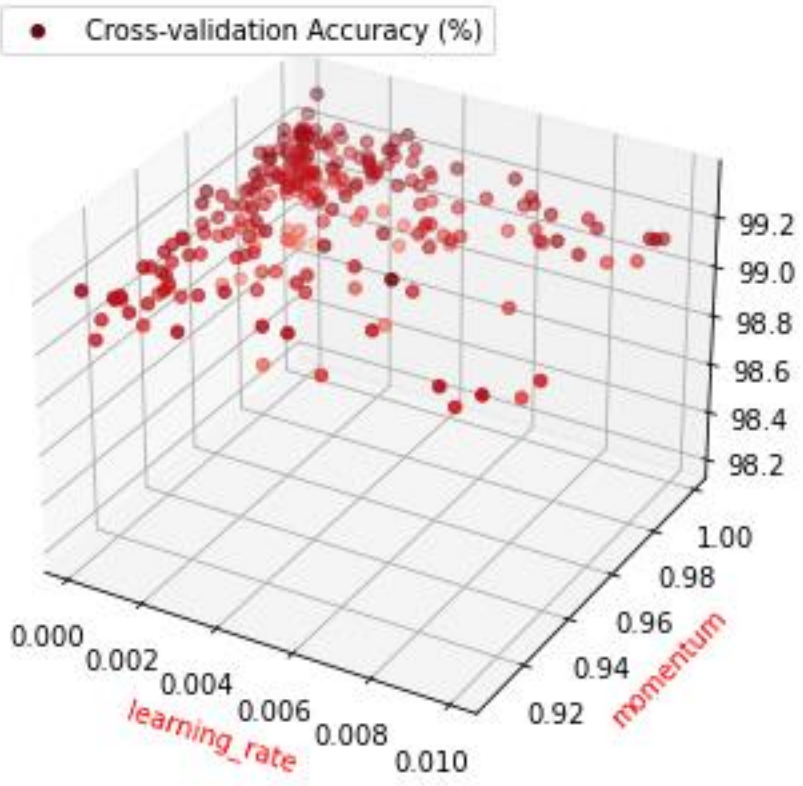

Figure 38. Random Search-Based Fine-Tuning Evaluation

4.1.1.3 Test Dataset-Based Evaluation

As mentioned in Section 3.1.1, the Caviar approach evaluated the 25 best-fit models in fine-tuning based upon the overall average F1-score of the model's mean F1-scores for both the pool and sea testing datasets described in Section 2.5.1. The mean F1 score for each test is the mean of the model's F1 scores at classifying pipes, valves, and backgrounds, as shown in Table 8 and Table 9. Table 10 exhibits all 25 models organized in descending order concerning the overall average F1 score.

Table 8. Model's F1-Score for SEA Test

			F1-score SEA (%)			
Batch_size	Learning_rate	Momentum	Pipe	Valve	Background	Mean F1-score SEA (%)
32	0.001	0.983	91.18379	87.20054	99.29950	92.561
16	0.004	0.980	91.18379	87.20054	99.29950	92.561
16	0.001	0.999	89.77786	83.43736	99.19445	90.803
16	0.001	0.985	91.64296	80.07675	99.31364	90.344
16	0.002	0.996	90.47661	81.90911	99.25996	90.549
16	0.001	0.991	89.31994	80.71519	99.11436	89.716
32	0.002	0.998	88.21873	79.40421	98.93944	88.854
16	0.004	0.995	90.14265	78.27166	99.33323	89.249
32	0.002	0.970	86.28244	82.00425	98.75982	89.016
16	0.005	0.980	84.83878	83.39958	98.91522	89.051
32	0.002	0.985	84.76974	81.87706	98.64838	88.432
32	0.004	0.951	87.23657	80.73881	98.87393	88.950
16	0.002	0.999	86.85116	80.4352	99.03736	88.775
16	0.002	0.987	85.39890	83.5376	98.74217	89.226
32	0.002	0.999	86.09997	78.00805	98.85162	87.653
32	0.002	0.930	86.75265	77.07091	98.77484	87.533
32	0.004	0.997	86.21083	77.62683	98.83800	87.559
16	0.001	0.975	85.49581	76.08517	98.82482	86.802
32	0.002	0.944	85.94014	75.34251	98.72862	86.670
32	0.002	0.993	85.03387	78.59784	98.65931	87.430
16	0.000	0.944	84.78496	75.02595	98.54909	86.120
16	0.001	0.999	83.03012	74.60763	98.97020	85.536
32	0.001	0.930	79.81216	74.22816	98.54594	84.195
32	0.004	0.963	82.97906	72.46437	98.69687	84.713
16	0.0003	0.960	77.30915	67.52302	98.30673	81.046

Table 9. Model's F1-Score for Pool Test

			F1-score POOL (%)			
Batch_size	Learning_rate	Momentum	Pipe	Valve	Background	F1-mean POOL (%)
32	0.001	0.983	95.92166	89.86706	99.73055	95.17309
16	0.004	0.980	95.92166	89.86706	99.73055	95.17309
16	0.001	0.999	96.69780	92.71112	99.75707	96.38866
16	0.001	0.985	96.37904	92.53906	99.73820	96.21877
16	0.002	0.996	96.30988	90.66188	99.74553	95.57243
16	0.001	0.991	96.31416	91.49583	99.74079	95.85026
32	0.002	0.998	96.60445	92.81558	99.74619	96.38874
16	0.004	0.995	96.21727	91.13641	99.73489	95.69619
32	0.002	0.970	96.34981	91.10408	99.75286	95.73558

16	0.005	0.980	96.26732	90.32199	99.74608	95.44513
32	0.002	0.985	96.47412	91.91982	99.75121	96.04839
32	0.004	0.951	96.22139	90.31585	99.73078	95.42267
16	0.002	0.999	96.2391	90.70055	99.74521	95.56162
16	0.002	0.987	96.17375	88.98667	99.72725	94.96256
32	0.002	0.999	96.72249	92.87800	99.76147	96.45399
32	0.002	0.930	96.44832	91.40134	99.76129	95.87032
32	0.004	0.997	96.08609	91.04505	99.73797	95.62304
16	0.001	0.975	96.47802	92.90641	99.74771	96.37738
32	0.002	0.944	96.52124	92.11995	99.73240	96.12453
32	0.002	0.993	95.92086	88.90118	99.74460	94.85555
16	0.000	0.944	96.53996	92.08869	99.74750	96.12538
16	0.001	0.999	95.75398	88.08890	99.75692	94.53327
32	0.001	0.930	96.11274	91.28621	99.73313	95.71070
32	0.004	0.963	95.41904	87.67930	99.66652	94.25495
16	0.000	0.960	95.58806	91.41910	99.68947	95.56554

Table 10. Overall Average PointNet's F1-Score (Sea and Pool Tests)

Batch_size	Learning_rate	Momentum	Mean F1-score SEA (%)	Mean F1-score POOL (%)	Overall Average F1-score (%)
32	0.001	0.983	92.561	95.17309	93.87
16	0.004	0.980	92.561	95.17309	93.87
16	**0.001**	**0.999**	**90.803**	**96.38866**	**93.60**
16	0.001	0.985	90.344	96.21877	93.28
16	0.002	0.996	90.549	95.57243	93.06
16	0.001	0.991	89.716	95.85026	92.78
32	0.002	0.998	88.854	96.38874	92.62
16	0.004	0.995	89.249	95.69619	92.47
32	0.002	0.970	89.016	95.73558	92.38
16	0.005	0.980	89.051	95.44513	92.25
32	0.002	0.985	88.432	96.04839	92.24
32	0.004	0.951	88.950	95.42267	92.19
16	0.002	0.999	88.775	95.56162	92.17
16	0.002	0.987	89.226	94.96256	92.09
32	0.002	0.999	87.653	96.45399	92.05
32	0.002	0.930	87.533	95.87031	91.70
32	0.004	0.997	87.559	95.62304	91.59
16	0.001	0.975	86.802	96.37738	91.59
32	0.002	0.944	86.670	96.12453	91.40

32	0.002	0.993	87.430	94.85555	91.14
16	0.000	0.944	86.120	96.12538	91.12
16	0.001	0.999	85.536	94.53327	90.03
32	0.001	0.930	84.195	95.71070	89.95
32	0.004	0.963	84.713	94.25495	89.48
16	0.000	0.960	81.046	95.56554	88.31

The results show that the model overfits neither the training dataset nor the cross-validation dataset (10 % of pool dataset) and generalizes to the sea and pool datasets even though the sea dataset is challenging and unseen data that comes from a different distribution from that distribution of the training and cross-validation datasets. The random search-based coarse-to-fine tuning and the test dataset-based evaluation managed to find a model that made PointNet NN classify pipes, valves, and backgrounds with high accuracy at pool and test datasets that have various distributions. Indeed, the proposed work succeeded in tuning hyperparameters to obtain, among the best-fit models shown in Table 10, a model that gave an overall average F1 score of 93.6% (F1-score|pool = 96.4%, F1-score|sea = 90.8%) using the following hyperparameter's values: mini-batch size = 16, momentum = 0.9985268, and learning rate = 0.001466154. By this score, the proposed technique outperforms and generalizes better to the sea dataset than the state-of-the-art work [64] which has a maximum overall average F1 score of 92.75% (F1-score|pool = 96.2%, F1-score|sea = 89.3%) with the same data and model but with different hyperparameters' values: mini-batch size = 16, momentum = 0.9, and learning rate = 0.001. This score indicates the low variance and small generalization gap that the model has, thanks to selecting a small mini-batch size and a significant momentum coefficient of more than 0.9 because both of them improve the model's generalization, as discussed in section 2.4.1.2.

In addition, these results surpass the pixel-wise accuracy of the literature [107], which achieved an accuracy of 73.4% using AlexNet NN [108] for classifying underwater pipes, and it did not pay much attention to tuning hyperparameters.

4.1.2. Hybrid Random and Grid-Search Based Hyperparameter Tuning Technique

4.1.2.1 Random Search-Based Tuning

As discussed in Section 3.2.1, the recommended scales by experts for mini-batch size, momentum, and learning rate are {16,32}, [0.9,0.999], and [0.0001,1], respectively. The same procedures and results exactly as Section 4.1.1.1 were conducted.

4.1.2.2 Grid Search-Based Tuning

According to the results in Section 4.1.2.1, the updated shrunk scales for mini-batch size, momentum, and learning rate were discretized as follows:

- Mini-batch size: {16,32}
- Momentum: {0.9,0.92475,0.9495,0.97425,0.999}
- Learning rate: {0.0001,0.00109,0.00208,0.00307,0.00406,0.00505, …,0.01}

Fifty five combinations of the values within the new discretized scales of learning rate and momentum. The mini-batch size was kept with the same scale. Fifty five random values as many as these combinations were generated for mini-batch size within {16,32}, as shown in Figure 39. Therefore, they were evaluated based on the PointNet CNN's cross-validation dataset-based visual classification accuracy, as shown in Figure 40.

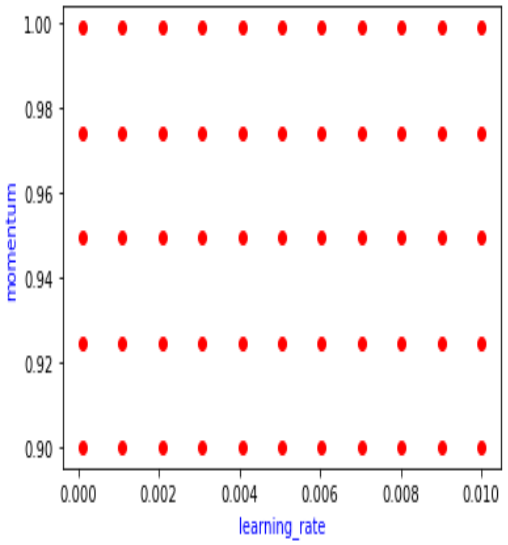

Figure 39. Grid search based on cross-validation accuracy

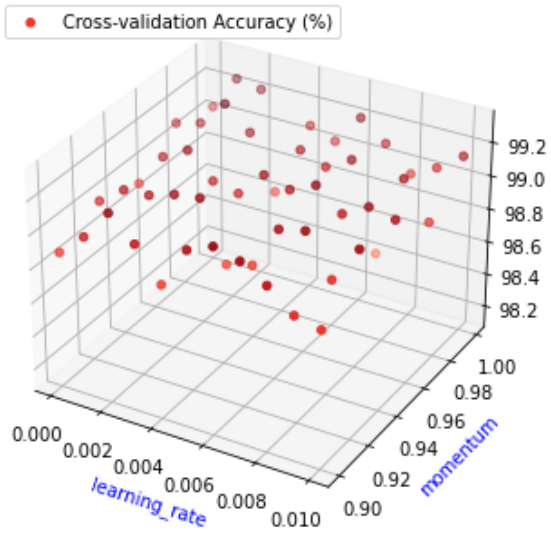

Figure 40. Cross-validation accuracy-based evaluation

Table 11 shows the best performing models ordered in descending order based on the PointNet's cross-validation accuracy.

Table 11. Cross-validation accuracy of grid search points

Batch-size	Learning-rate	Momentum	Cross-validation accuracy (%)
32	**0.00208**	**0.97425**	**99.3030**
16	0.00109	0.99900	99.2470
32	0.00208	0.99900	99.2210
32	0.00604	0.99900	99.2070
16	0.00703	0.94950	99.2041
32	0.00208	0.94950	99.2020
16	0.00208	0.90000	99.1983
32	0.00406	0.92475	99.1860
16	0.00604	0.90000	99.1820
16	0.00505	0.94950	99.1802
32	0.00703	0.97425	99.1730
32	0.00307	0.97425	99.1720
32	0.00802	0.92475	99.1690
16	0.00307	0.92475	99.1642
32	0.00901	0.94950	99.1630
32	0.00307	0.99900	99.1530
16	0.01000	0.92475	99.1517

16	0.00505	0.97425	99.1490
16	0.00109	0.97425	99.1490
16	0.00901	0.97425	99.1435
16	0.00703	0.90000	99.1426
16	0.01000	0.99900	99.1396
32	0.00604	0.94950	99.1360
16	0.00703	0.92475	99.1344
16	0.01000	0.94950	99.1316
16	0.00109	0.94950	99.1220
16	0.00505	0.90000	99.1174
16	0.00208	0.92475	99.1170
16	0.00010	0.97425	99.1090
16	0.00109	0.92475	99.1060
16	0.00703	0.99900	99.0931
16	0.00604	0.97425	99.0897
32	0.00406	0.99900	99.0810
16	0.00802	0.94950	99.0793
16	0.00307	0.90000	99.0630
32	0.00307	0.94950	99.0610
16	0.00802	0.90000	99.0497
16	0.00010	0.99900	99.0380
16	0.00901	0.99900	99.0292
16	0.00406	0.94950	99.0290
16	0.00505	0.99900	99.0264
16	0.00109	0.90000	99.0171
32	0.00010	0.92475	98.9980
16	0.00802	0.99900	98.9512
32	0.00901	0.92475	98.9320
16	0.01000	0.97425	98.9297
16	0.00901	0.90000	98.9247
32	0.00010	0.94950	98.9210
16	0.01000	0.90000	98.8897
32	0.00010	0.90000	98.8830
16	0.00604	0.92475	98.8812
16	0.00406	0.90000	98.8728
16	0.00406	0.97425	98.8565
16	0.00505	0.92475	98.8409
16	0.00802	0.97425	98.6542

4.1.2.3 Test Dataset-Based Tuning

As mentioned in Section 3.1.2, the Caviar approach evaluated the best-fit models in grid search-based tuning based upon the overall average F1-score of the model's mean F1-scores for both the pool and sea testing datasets described in Section 2.5.1. The mean F1 score for each test is the mean of the model's F1 scores at classifying pipes, valves, and backgrounds, as shown in Table 12 and Table 13. Table 14 exhibits all models organized in descending order concerning the overall average F1 score.

Table 12. F1-score evaluation based on SEA dataset

			F1-score SEA (%)		
Batch_size	Learning_rate	Momentum	Pipe	Valve	Background
16	0.00109	0.99900	89.7779	83.4374	99.1944
32	0.00208	0.97425	86.2824	82.0043	98.7598
32	0.00208	0.99900	86.1000	78.0081	98.8516
16	0.00505	0.9495	86.8926	78.5681	99.0078

Table 13. F1-score evaluation based on POOL dataset

			F1-score POOL (%)		
Batch_size	Learning_rate	Momentum	Pipe	Valve	Background
16	0.00109	0.99900	96.6978	92.7111	99.7571
32	0.00208	0.97425	96.3498	91.1041	99.7529
32	0.00208	0.99900	96.7225	92.8780	99.7615
16	0.00505	0.9495	96.4684	91.1416	99.7502

Table 14. Overall average of mean F1-scores on sea and pool datasets

Batch size	Learning rate	Momentum	F1-mean SEA (%)	F1-mean POOL (%)	Overall Mean (%)
16	**0.001**	**0.999**	**90.803**	**96.389**	**93.596**
32	0.002	0.974	89.016	95.736	92.376
32	0.002	0.999	87.653	96.454	92.054
16	0.005	0.950	88.156	95.787	91.971

The results prove the impact of random search on obtaining the productive hyperparameter space that would make the PointNet CNN achieve high visual classification accuracy in shorter time and with less computation units. For example, without using the random search, the grid search would not have been conducted using only 55 ternary combinations of hyperparameters' values. Otherwise, it would have been conducted using 100 or 200 or 300 ternary combinations or even more.

The results show that the efficiency of grid search at tuning the learning rate and momentum because model overfits neither the training dataset nor the cross-validation dataset (10 % of pool dataset) and generalizes to the sea and pool datasets even though the sea dataset is challenging and unseen data that comes from a different distribution from that distribution of the training and cross-validation datasets. The random search-based tuning and the grid search-based tuning evaluation managed to find a model that made PointNet NN classify pipes, valves, and backgrounds with high accuracy at pool and test datasets that have various distributions. Indeed, the proposed work succeeded in tuning hyperparameters to obtain, among the best-fit models shown in Table 10, a model that gave an overall average F1 score of 93.6% (F1-score|pool = 96.4%, F1-score|sea = 90.8%) using the following hyperparameter's values: mini-batch size = 16, momentum = 0.9985268, and learning rate = 0.001466154. By this score, the proposed technique outperforms and generalizes better to the sea dataset than the state-of-the-art work [64] which has a maximum overall average F1 score of 92.75% (F1-score|pool = 96.2%, F1-score|sea = 89.3%) with the same data and model but with different hyperparameters' values: mini-batch size = 16, momentum = 0.9, and learning rate = 0.001. This score indicates the low variance and small generalization gap that the model has, thanks to selecting a small mini-batch size and a

significant momentum coefficient of more than 0.9 because both of them improve the model's generalization, as discussed in section 2.4.1.2.

In addition, these results surpass the pixel-wise accuracy of the literature [107], which achieved an accuracy of 73.4% using AlexNet NN [108] for classifying underwater pipes, and it did not pay much attention to tuning hyperparameters.

4.1.3. Hybrid Random and Manual Search-Based Hyperparameter Tuning Technique

4.1.3.1 Random Search-Based Tuning

As discussed in Section 3.2.1, the recommended scales by experts for mini-batch size, momentum, and learning rate are {16,32}, [0.9,0.999], and [0.0001,1], respectively. The same procedures and results exactly as Section 4.1.1.1 were conducted.

4.1.3.2 Manual Search-Based Tuning

According to the results in Section 4.1.3.1, the updated shrunk scales for mini-batch size, momentum, and learning rate were discretized as follows:

- Mini-batch size: {16,32}
- Momentum: {0.9,0.92475,0.9495,0.97425,0.999}
- Learning rate: {0.0001,0.00109,0.00208,0.00307,0.00406,0.00505, …,0.01}

The three hyperparameters were tuned by walking through the discretized scales one at a time. While a hyperparameter was being tuned by increasing its value within its discrete scale, the other hyperparameters were kept with constant values. The tuning process for each hyperparameter should stop when the learning model starts to give a lower cross-validation accuracy than before, as shown in Figure 41 and Table 15.

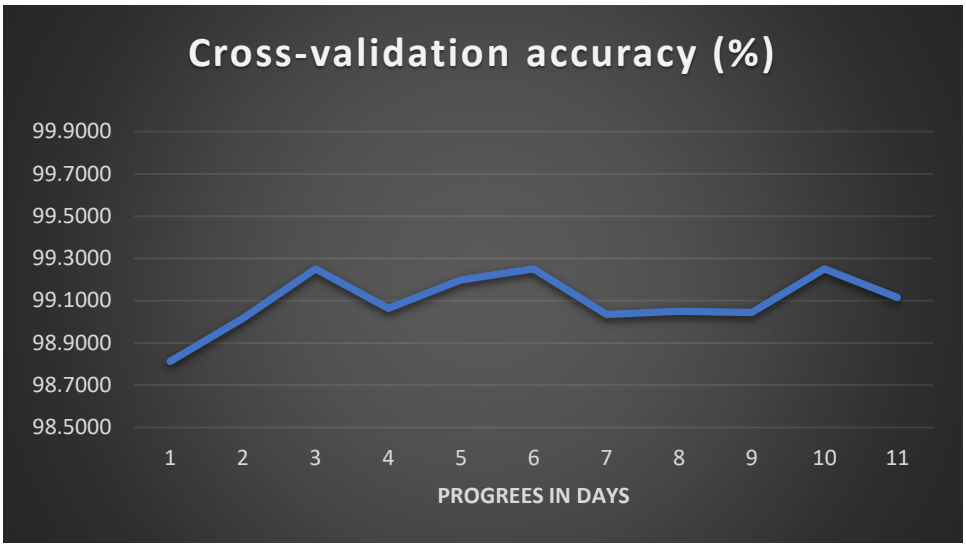

Figure 41. Manual Search based on Cross-Validation accuracy

Table 15. Manual Search based on Cross-Validation accuracy

Batch_size	Learning_rate	Momentum	Cross_validation accuracy (%)
Learning_rate Tuning			
16	0.00010	0.90000	98.8123
16	0.00109	0.90000	99.0171
16	**0.00159**	**0.90000**	**99.2504**
16	0.00307	0.90000	99.0630
16	0.00208	0.90000	99.1983
Momentum Tuning			
16	**0.00159**	**0.90000**	**99.2504**
16	0.00159	0.91238	99.0352
16	0.00159	0.92475	99.0494
16	0.00159	0.97425	0.990436
Batch Size Tuning			
16	**0.00159**	**0.90000**	**99.2504**
32	0.00159	0.90000	99.1148

4.1.3.3 Test Dataset-Based Tuning

As mentioned in Section 3.1.2, the Caviar approach evaluated the best-fit models in manual search-based tuning based upon the overall average F1-score of the model's mean F1-scores for

both the pool and sea testing datasets described in Section 2.5.1. The mean F1 score for each test is the mean of the model's F1 scores at classifying pipes, valves, and backgrounds, as shown in Table 16, Table 17, and Table 18. exhibits all models organized in descending order concerning the overall average F1 score.

Table 16. F1-score evaluation based on SEA dataset

			F1-score SEA (%)		
Batch_size	**Learning_rate**	**Momentum**	Pipe	Valve	Background
16	0.00159	0.90000	82.8764	76.5802	98.5995

Table 17. F1-score evaluation based on POOL dataset

			F1-score POOL (%)		
Batch_size	Learning_rate	Momentum	Pipe	Valve	Background
16	0.00159	0.90000	96.3089	91.7342	99.7330

Table 18. Overall average of mean F1-scores evaluation based on SEA and POOL datasets

Batch_size	**Learning_rate**	**Momentum**	**F1-mean SEA (%)**	**F1-mean POOL (%)**	**Overall Mean (%)**
16	**0.00159**	**0.90000**	**86.0186867**	**95.92534765**	**90.97201718**

The results prove the impact of random search on obtaining the productive hyperparameter space that would make the PointNet CNN achieve high visual classification accuracy in shorter time and with less computation units. This productive hyperparameter space guides the manual search with prior-knowledge that would accelerate the manual search-based hyperparameter tuning.

Furthermore, the results show that manual search-based hyperparameter tuning is less efficient than both random and grid search-based tuning. The tuned hyperparameters made the model achieve an overall average F1 score of 90.97% (F1-score|pool = 95.93%, F1-score|sea = 86.02%) using the following hyperparameter's values: mini-batch size = 16, momentum = 0.9, and

learning rate = 0.00159. These results include lower classification accuracy than the results of both random and grid search-based tuning mentioned in Section 4.1.3.1 and Section 4.1.3.2. However, it is a shortcut tuning technique that saves computation time and resources when prior-knowledge is available.

4.2. Standard Feedforward Neural Network Without Regularization

Based upon Section 3.2.2, the work was carried using three distinct hyperparameter tuning techniques: the random search-based tuning, the hybrid random and grid search-based tuning stage, and the hybrid random and manual search-based tuning. The standard feedforward NN was fed by the dataset, make_moons, as mentioned in Section 2.5.2.

4.2.1. Random Search-Based Hyperparameter Tuning Technique

4.2.1.1 Random Search-Based Coarse Tuning

As discussed in Section 3.2.2, the recommended scales by experts for mini-batch size, momentum, and learning rate are {16,32}, [0.9,0.999], and [0.0001,1], respectively. Therefore, 50 models, with distinct and random values for each hyperparameter, were composed by producing 50 ternary combinations from the three scales, as visualized in Figure 42.

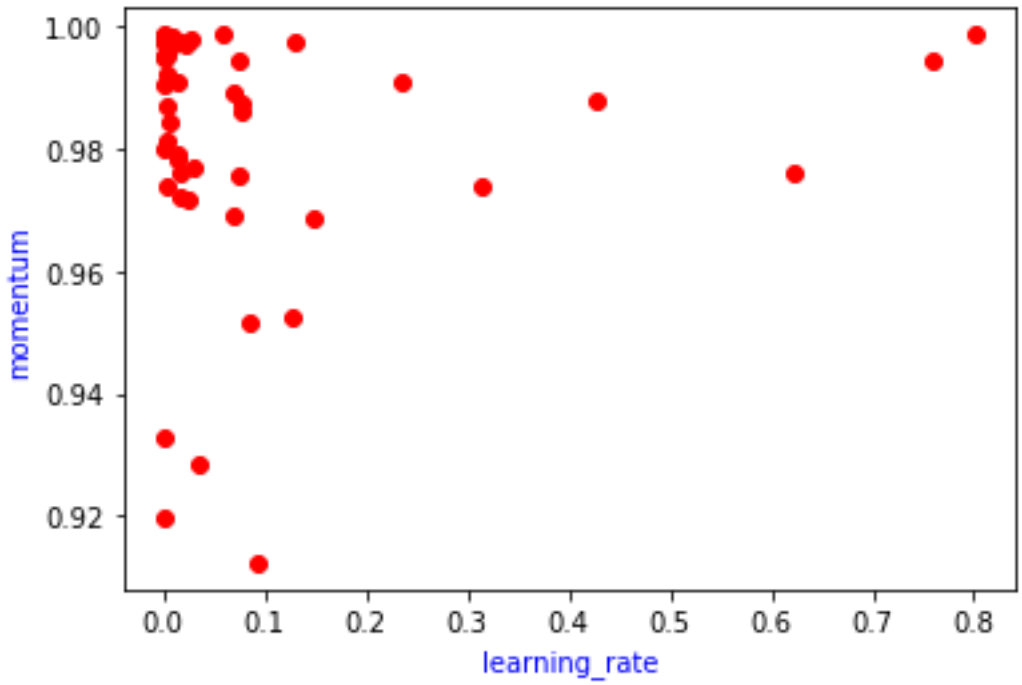

Figure 42. Coarse Random Search-Based Hyperparameter Tuning

Subsequently, the model's binary classification accuracy for the cross-validation dataset was used to evaluate the 50 models. The hyperparameters' values, that made the deep feedforward NN model achieve an accuracy of at least 95%, were exploited to shrink the aforementioned initial scales and set smaller ones, as shown in Table 19, Figure 43, and Figure 44.

Table 19. Cross-validation dataset-based evaluation of coarse random search

Batch_size	Learning_rate	Momentum	Cross-validation accuracy (%)
16	**0.05950**	**0.99860**	**96.0000**
32	0.03076	0.97692	95.6667
16	0.42769	0.98792	95.6667
32	0.12947	0.99750	95.6667
16	0.01488	0.97831	95.6667
16	0.09183	0.91226	95.6667
32	0.07343	0.97585	95.6667
32	0.06805	0.96913	95.6667
16	0.07346	0.99461	95.3333
32	0.02365	0.97164	95.3333
32	0.03579	0.92864	95.3333
16	0.01331	0.99087	95.3333
16	0.31453	0.97398	95.0000

16	0.01731	0.97195	95.0000
32	0.14826	0.96885	95.0000
16	0.01725	0.97610	95.0000
16	0.80181	0.99867	95.0000
16	0.12583	0.95231	94.6667
32	0.75944	0.99441	94.6667
32	0.07728	0.98724	94.6667
16	0.00582	0.99694	94.3333
32	0.23579	0.99108	94.3333
32	0.08376	0.95167	94.3333
32	0.06861	0.98933	94.3333
16	0.02806	0.99799	94.3333
16	0.07692	0.98602	94.3333
16	0.00587	0.99663	94.3333
16	0.00626	0.99742	94.3333
16	0.00433	0.99223	94.0000
16	0.62249	0.97589	94.0000
16	0.00663	0.98439	94.0000
32	0.01477	0.97908	94.0000
16	0.02186	0.99711	93.6667
16	0.00253	0.99522	93.6667
32	0.00452	0.98149	93.6667
16	0.00905	0.99851	93.6667
16	0.01519	0.99753	93.6667
16	0.00197	0.99552	93.3333
16	0.00169	0.98000	92.3333
32	0.00243	0.97404	90.0000
32	0.00147	0.99747	87.6667
32	0.00204	0.98722	87.3333
32	0.00076	0.99864	79.6667
16	0.00026	0.99470	75.0000
16	0.00024	0.99763	74.3333
32	0.00041	0.91979	73.3333
32	0.00036	0.93271	71.6667
32	0.00027	0.99073	69.6667
16	0.00012	0.99819	69.0000
32	0.00018	0.99841	67.3333

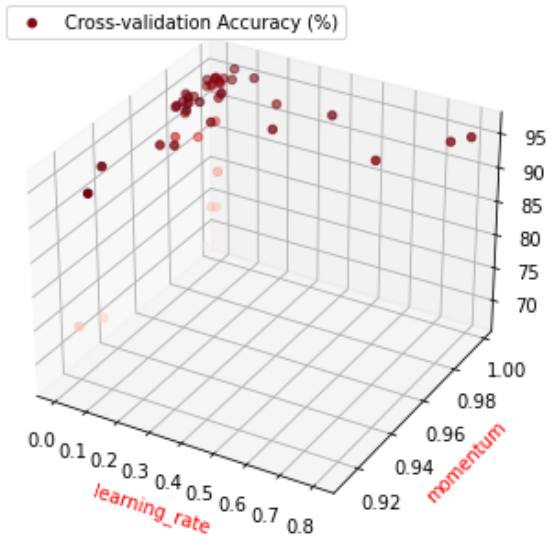

Figure 43. Random Search-Based Coarse Tuning Evaluation

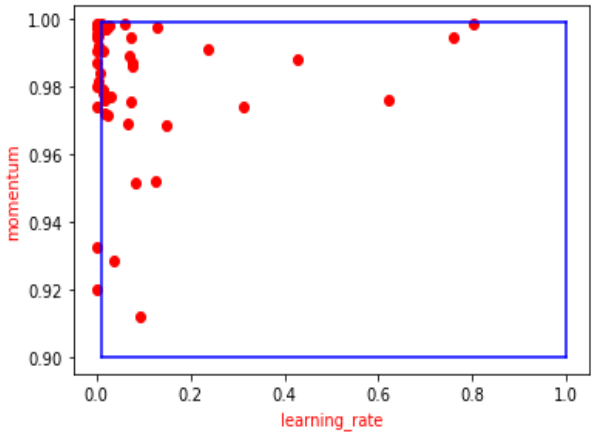

Figure 44. Random Search-Based Coarse Tuning Evaluation: Shrunk scales represented by a blue rectangle.

4.2.1.2 Random Search-Based Fine Tuning

According to the results in Section 4.2.1.1, the updated shrunk scales for mini-batch size, momentum, and learning rate were {16,32}, [0.9,0.999], and [0.01,1], respectively. Thus, 500 models, with distinct values for each hyperparameter, were formed by generating 500 ternary combinations from the three updated scales, as shown in Figure 45.

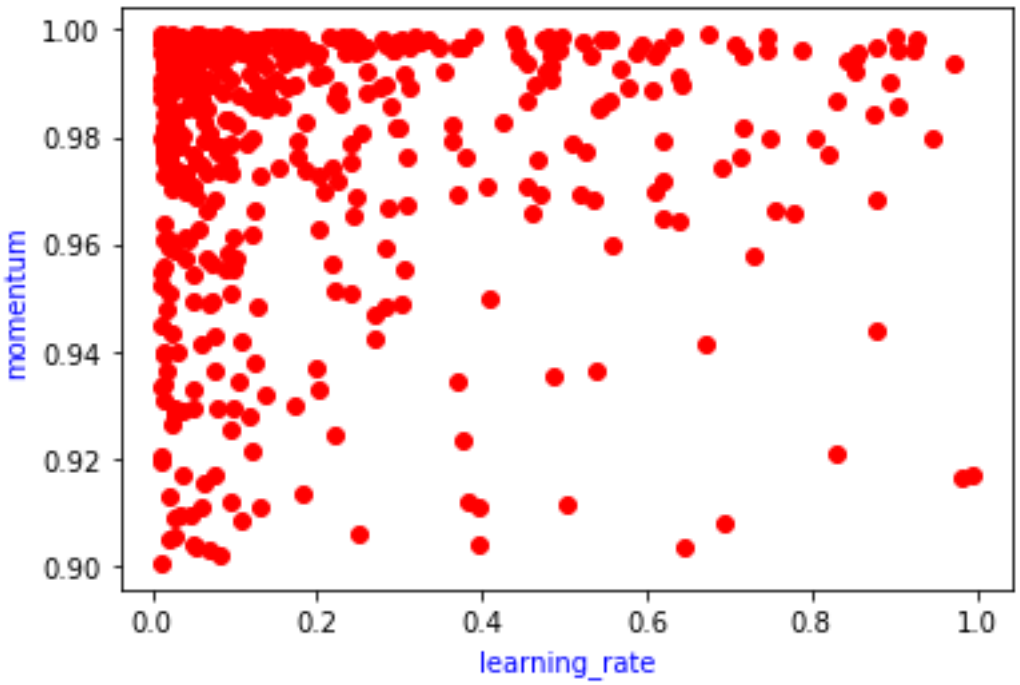

Figure 45. Random Search-Based Fine Hyperparameter Tuning

After that, the model's binary classification accuracy for the cross-validation dataset was used to evaluate the 500 models, as shown in Table 20 and Figure 46.

Table 20. Random Search-Based Fine-Tuning Evaluation

Batch-size	Learning-rate	Momentum	Cross-validation accuracy (%)
16	**0.35476**	**0.99200**	**96.6667**
32	0.80205	0.97973	96.3333
32	0.13542	0.93202	96.3333
16	0.47685	0.99226	96.3333
16	0.21756	0.95625	96.3333
32	0.05301	0.99897	96.3333
16	0.03833	0.97206	96.3333
16	0.25178	0.99638	96.3333
32	0.17398	0.99730	96.3333
32	0.22119	0.95138	96.3333
32	0.06899	0.99846	96.3333
16	0.20056	0.99555	96.3333
16	0.13113	0.99691	96.3333
16	0.10383	0.99835	96.0000

32	0.07460	0.99891	96.0000
32	0.05567	0.99893	96.0000
32	0.46492	0.98959	96.0000
32	0.45261	0.99345	96.0000
16	0.29367	0.99639	96.0000

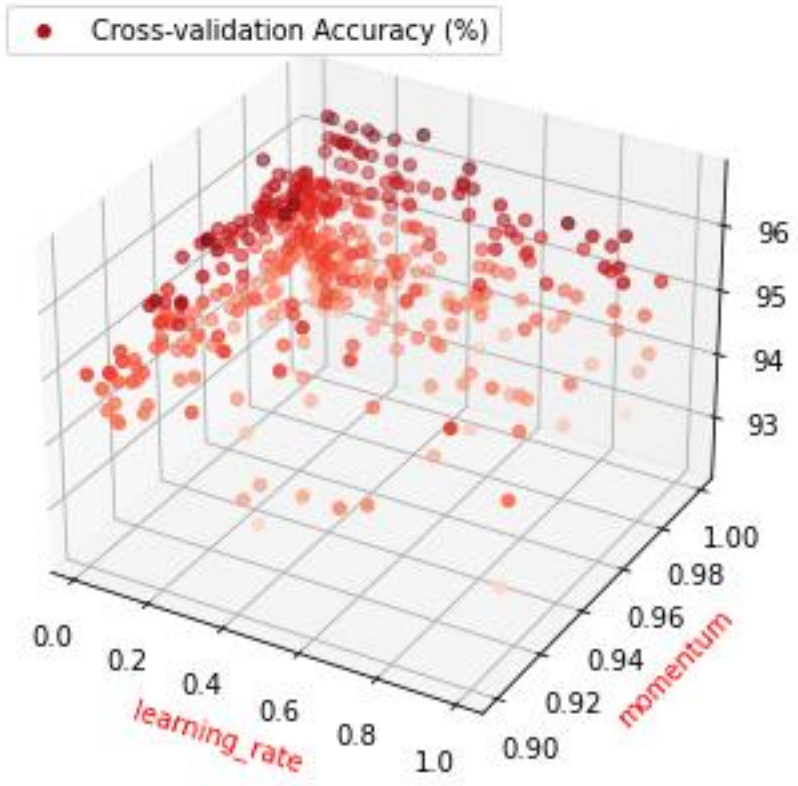

Figure 46. Random Search-Based Fine-Tuning Evaluation

The results show that the random search-based coarse-to-fine tuning managed to find a model that made standard feedforward NN conduct the binary classification with high accuracy of 96.67 % using the hyperparameters' values: mini-batch size = 16, learning rate = 0.35476, momentum = 0.99200, as shown in Table 20. By this binary classification accuracy shown in Figure 47, the deep feedforward neural network (FNN) outperforms the state-of-the-art work in [109] that achieved using a RBF kernel-based SVM a binary classification accuracy of 96 % for

the same dataset, make_moons provided by scikit-learn library (sklearn), as mentioned in Section 2.5.2. Therefore, this indicates the superiority of deep neural networks over traditional machine learning algorithm such as SVM.

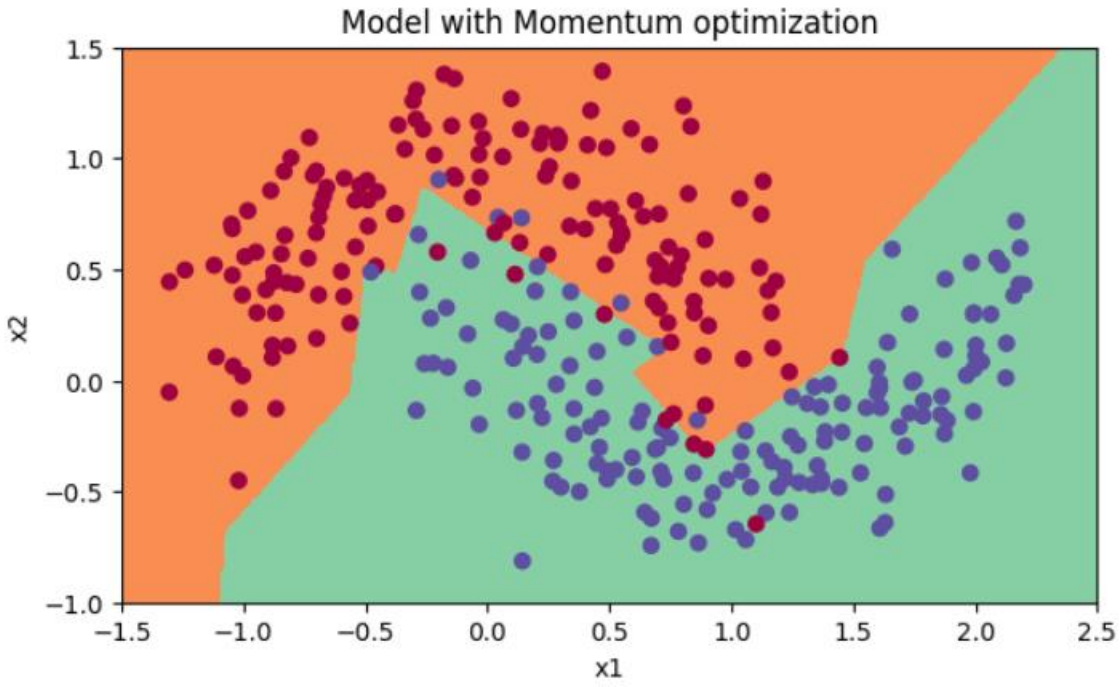

Figure 47. Binary Classification of make_moons with an accuracy of 96.67 %

4.2.2. Hybrid Random and Grid-Search Based Hyperparameter Tuning Technique

4.2.2.1 Random Search-Based Tuning

As discussed in Section 3.2.2, the recommended scales by experts for mini-batch size, momentum, and learning rate are {16,32}, [0.9,0.999], and [0.0001,1], respectively. The same procedures and results exactly as Section 4.2.1.1 were obtained.

4.2.2.2 Grid Search-Based Tuning

According to the results in Section 4.2.2.1, the updated shrunk scales for mini-batch size, momentum, and learning rate were discretized as follows:

- Mini-batch size: {16,32}
- Momentum: {0.9,0.9045,0.909,0.9135, …,0.999} with step size = 0.0045
- Learning rate: {0.01,0.0145,0.019, …,1} with step size = 0.0045

529 combinations of the values within the new discretized scales of learning rate and momentum. The mini-batch size was kept with the same scale. 529 random values as many as these combinations were generated for mini-batch size within {16,32}, as shown in Figure 48. Therefore, they were evaluated based on the learning model's cross-validation-based binary classification accuracy, as shown in Figure 49.

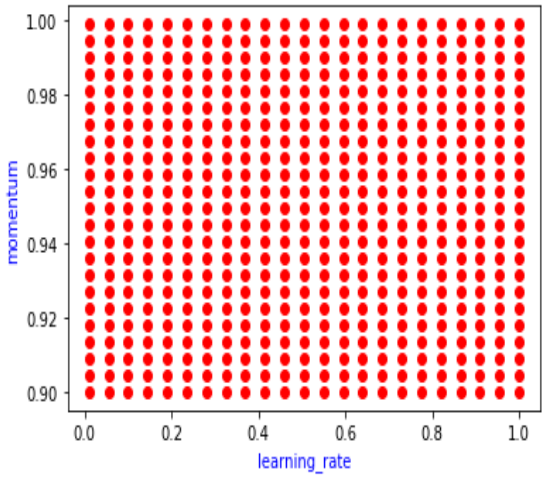

Figure 48. Grid search-based on cross-validation accuracy

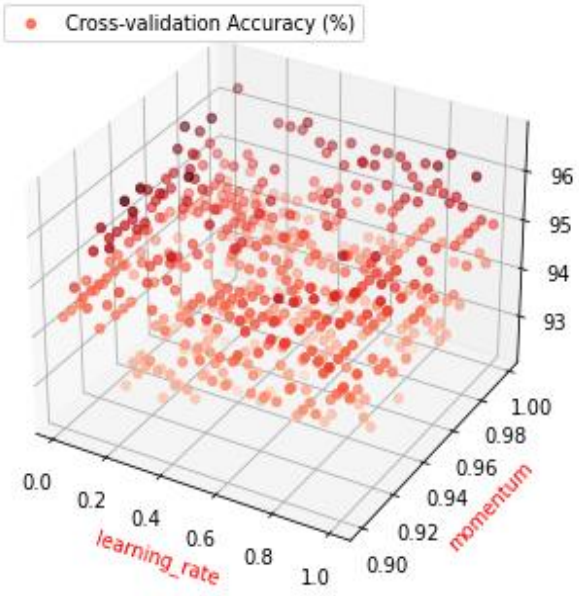

Figure 49. Cross-validation accuracy-based evaluation

Table 21 shows the best performing models ordered in descending order based on the learning model's cross-validation accuracy.

Table 21. Cross-validation accuracy of grid search points

Batch_size	Learning_rate	Momentum	Cross-validation accuracy (%)
16	**0.1450**	**0.9675**	**96.6667**
16	0.1000	0.9585	96.6667
32	0.1900	0.9090	96.6667
32	0.1900	0.9180	96.6667
32	0.2800	0.9315	96.6667
16	0.1000	0.9540	96.3333
16	0.1000	0.9675	96.3333
16	0.8650	0.9855	96.3333
16	0.0550	0.9990	96.3333
32	0.3700	0.9900	96.3333
32	0.1900	0.9135	96.3333
16	0.8650	0.9945	96.3333
32	1.0000	0.9855	96.3333
16	0.7300	0.9900	96.0000
16	0.1000	0.9360	96.0000

32	0.2350	0.9945	96.0000
16	0.5950	0.9945	96.0000
16	0.6400	0.9945	96.0000
16	0.6850	0.9945	96.0000

The results show that the hybrid random and grid search-based tuning managed to find a model that made standard feedforward NN conduct the binary classification with high accuracy of 96.67 % with hyperparameters' values: mini-batch size = 16, learning rate = 0.1450, momentum = 0.9675, as shown in Table 21. The results prove the impact of random search on obtaining the productive hyperparameter space that would make the feedforward NN achieve high binary classification accuracy in shorter time and with less computation units. For example, without using the random search, the grid search would not have been conducted using only 529 ternary combinations of hyperparameters' values. Otherwise, it would have been conducted using 1000 or 2000 ternary combinations or even more.

By this binary classification accuracy shown in Figure 50, the deep feedforward NN outperforms the state-of-the-art work in [109] that achieved using a RBF kernel-based SVM a binary classification accuracy of 96 % for the same dataset, make_moons provided by scikit-learn library (sklearn), as mentioned in Section 2.5.2. Therefore, this indicates the superiority of deep neural networks over traditional machine learning algorithm such as SVM.

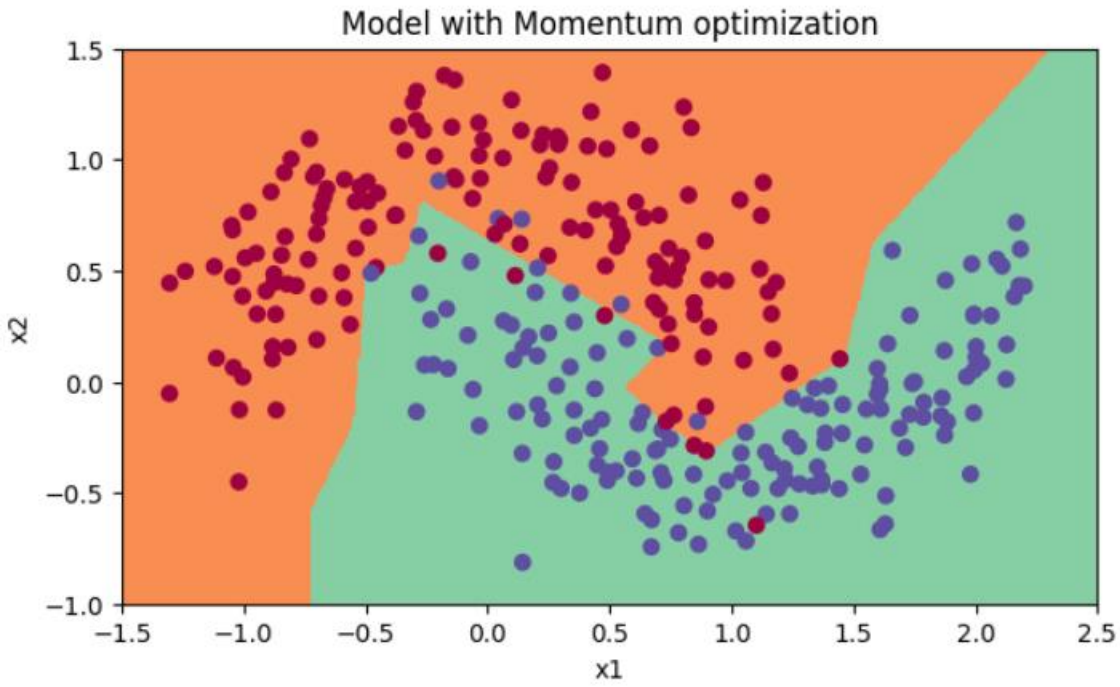

Figure 50. Binary Classification accuracy of 96.67: make_moons dataset

4.2.3. Hybrid Random and Manual Search-Based Hyperparameter Tuning Technique

4.2.3.1 Random Search-Based Tuning

As discussed in Section 3.2.2, the recommended scales by experts for mini-batch size, momentum, and learning rate are {16,32}, [0.9,0.999], and [0.0001,1], respectively. The same procedures and results exactly as Section 4.1.1.1 were conducted.

4.2.3.2 Manual Search-Based Tuning

According to the results in Section 4.2.2.1, the updated shrunk scales for mini-batch size, momentum, and learning rate were discretized as follows:

- Mini-batch size: {16,32}
- Momentum: {0.9,0.9045,0.909,0.9135, …,0.999} with step size = 0.0045
- Learning rate: {0.01,0.0145,0.019, …,1} with step size = 0.0045

The three hyperparameters were tuned by walking through the discretized scales one at a time. While a hyperparameter was being tuned by increasing its value within its discrete scale, the other hyperparameters were kept with constant values. The tuning process for each hyperparameter should stop when the learning model starts to give a lower cross-validation accuracy than before, as shown in Figure 51 and Table 22.

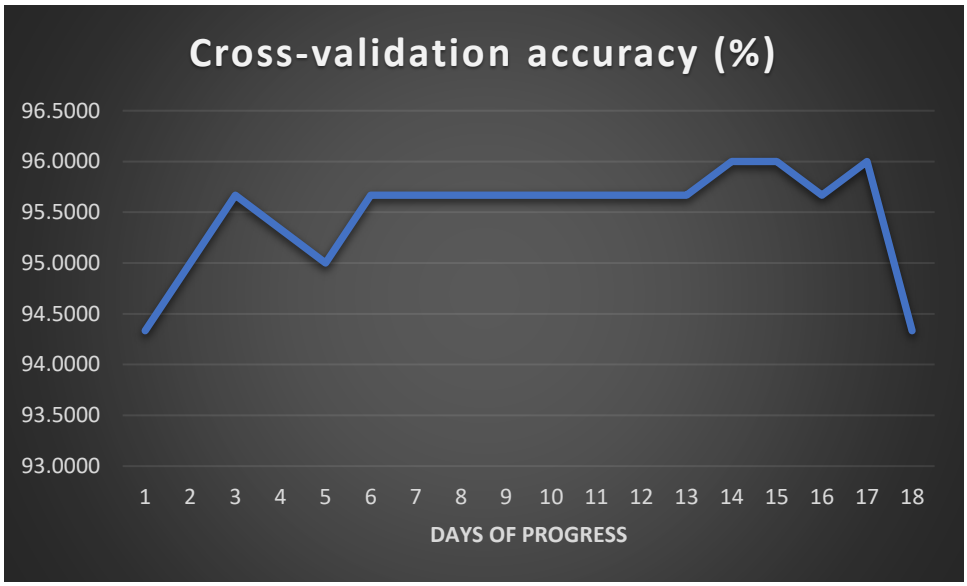

Figure 51. Manual Search based on Cross-Validation accuracy

Table 22. Manual Search based on Cross-Validation accuracy

Batch_size	Learning_rate	Momentum	Cross_validation accuracy (%)
Learning_rate Tuning			
16	0.0100	0.9000	94.3333
16	0.0550	0.9000	95.0000
16	**0.1000**	**0.9000**	**95.6667**
16	0.1225	0.9000	95.3333
16	0.1450	0.9000	95.0000
Momentum Tuning			
16	0.1000	0.9000	95.6667
16	0.1000	0.9045	95.6667
16	0.1000	0.9090	95.6667
16	0.1000	0.9135	95.6667
16	0.1000	0.9180	95.6667

16	0.1000	0.9225	95.6667
16	0.1000	0.9270	95.6667
16	0.1000	0.9315	95.6667
16	**0.1000**	**0.9360**	**96.0000**
16	0.1000	0.9383	96.0000
16	0.1000	0.9405	95.6667
Batch Size Tuning			
16	**0.1000**	**0.9360**	**96.0000**
32	0.1000	0.9360	94.3333

The results prove the impact of random search on obtaining the productive hyperparameter space that would make the standard feedforward NN achieve high binary classification accuracy of 96% with hyperparameters' values: mini-batch size = 16, learning rate =0.1000, momentum = 0.9360 in a shorter time and with less computation units. This productive hyperparameter space guides the manual search with prior-knowledge that would accelerate the manual search-based hyperparameter tuning. It is obvious when we compare these results with those in Section 4.2.1 and Section 4.2.2 that the hybrid random and manual search-based tuning is less efficient than both random search-based tuning and the hybrid random and grid search-based tuning.

By this binary classification accuracy shown in Figure 52, the deep feedforward NN have a comparable binary classification accuracy to the state-of-the-art work in [109] that achieved using a RBF kernel-based SVM a binary classification accuracy of 96 % for the same dataset, make_moons provided by scikit-learn library (sklearn), as mentioned in Section 2.5.2.

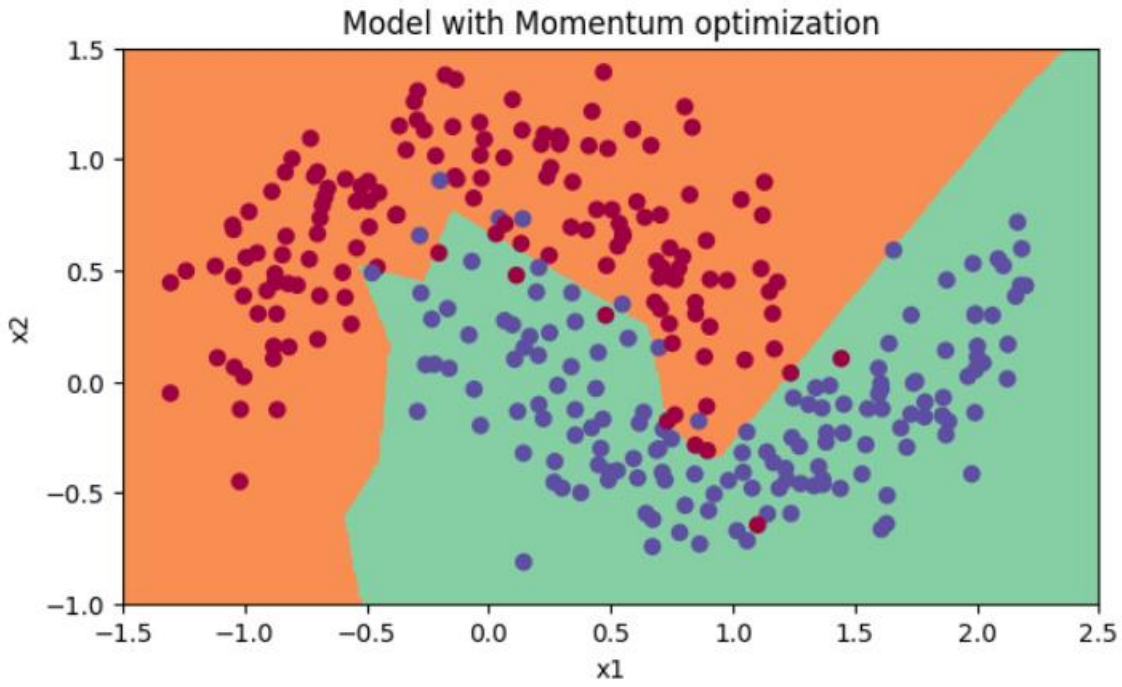

Figure 52. Binary classification accuracy of 96 %: make_moons dataset

4.3. Standard Feedforward Neural Network With Regularization

Based upon Section 3.2.3, the work was carried using three distinct hyperparameter tuning techniques: the random search-based tuning, the hybrid random and grid search-based tuning stage, and the hybrid random and manual search-based tuning. The standard feedforward NN was fed by the dataset, make_moons, as mentioned in Section 2.5.2

4.3.1. Random Search-Based Hyperparameter Tuning Technique

4.3.1.1 Random Search-Based Coarse Tuning

As discussed in Section 3.2.3, the recommended scales by experts for learning rate and regularization parameter (λ) are [0.0001,1] and [0.1,10], respectively. Therefore, 50 models, with distinct and random values for each hyperparameter, were composed by producing 50 binary combinations from the two scales, as visualized in Figure 53.

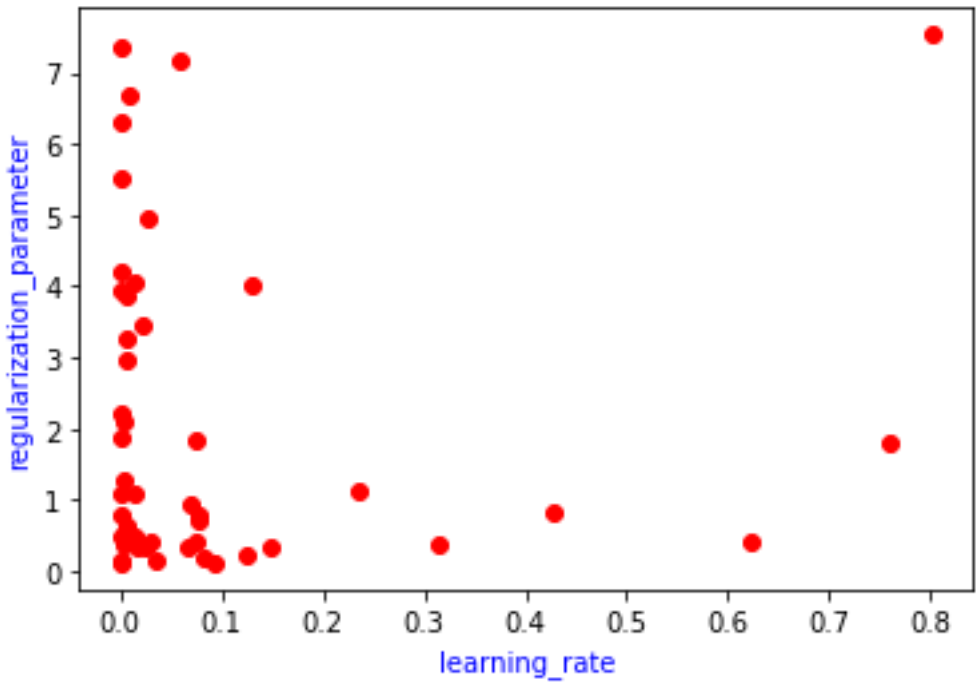

Figure 53. Coarse Random Search-Based Hyperparameter Tuning

Subsequently, the model's binary classification accuracy for the cross-validation dataset was used to evaluate the 50 models. The hyperparameters' values, that made the deep feedforward NN model achieve an accuracy of at least 93.5%, were exploited to shrink the aforementioned initial scales and set smaller ones, as shown in Table 23, Figure 54, and Figure 55.

Table 23. cross-validation accuracy evaluation of coarse random search-based tuning

Learning-rate	Regularization-parameter	Train-accuracy (%)	Cross-validation accuracy (%)
0.07343	**0.41404**	**93.83886**	**95.5**
0.14826	0.32100	93.83886	95.0
0.31453	0.38426	94.31280	95.0
0.06805	0.32396	93.83886	95.0
0.03076	0.43329	93.36493	95.0
0.02365	0.35266	93.36493	95.0
0.12583	0.20971	93.83886	94.5
0.03579	0.14014	93.36493	94.5
0.08376	0.20692	93.36493	94.5
0.01488	0.46103	92.89100	94.0
0.09183	0.11397	93.83886	94.0

0.01725	0.41845	92.89100	94.0
0.01477	0.47791	92.89100	94.0
0.01731	0.35655	92.89100	94.0
0.06861	0.93741	92.41706	93.5
0.07728	0.78399	92.41706	93.5
0.01331	1.09527	92.41706	93.5
0.07692	0.71517	92.89100	93.5
0.62249	0.41478	93.83886	93.5
0.23579	1.12156	92.41706	93.0
0.00663	0.64073	91.94313	93.0
0.00433	1.28739	92.41706	93.0
0.07346	1.85404	92.41706	93.0
0.00253	2.09075	91.94313	92.5
0.05950	7.15939	90.52133	92.5
0.00076	7.34617	88.62559	92.5
0.42769	0.82787	92.41706	92.5
0.00587	2.96638	91.46919	92.5
0.00147	3.95588	91.94313	92.5
0.00452	0.54034	91.94313	92.5
0.00905	6.69177	90.99526	92.5
0.00197	2.23042	91.94313	92.0
0.01519	4.04798	91.94313	92.0
0.00626	3.88311	91.94313	92.0
0.02186	3.45805	91.46919	92.0
0.00169	0.50009	91.94313	92.0
0.12947	3.99989	91.94313	92.0
0.02806	4.96910	91.94313	92.0
0.00582	3.27076	91.46919	92.0
0.00204	0.78243	91.94313	92.0
0.00243	0.38516	91.94313	92.0
0.00036	0.14861	90.04739	90.5
0.00041	0.12468	90.04739	90.5
0.75944	1.78978	88.15166	89.5
0.00026	1.88646	88.62559	89.0
0.00027	1.07852	89.09953	89.0
0.00024	4.21086	86.72986	88.0
0.00018	6.30654	86.72986	87.5
0.00012	5.51331	84.36019	84.5
0.80181	7.53255	78.67299	78.0

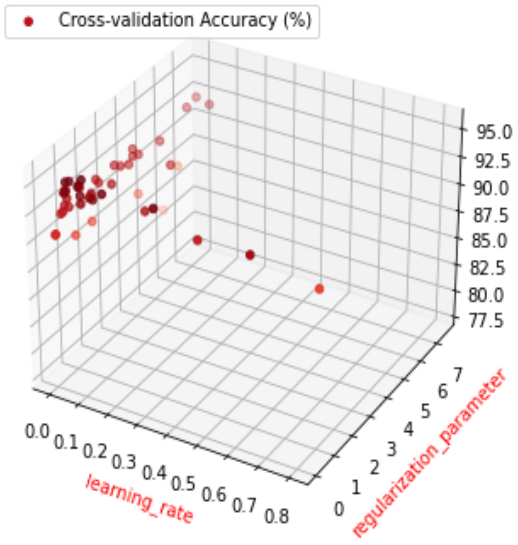

Figure 54. Random Search-Based Coarse Tuning Evaluation

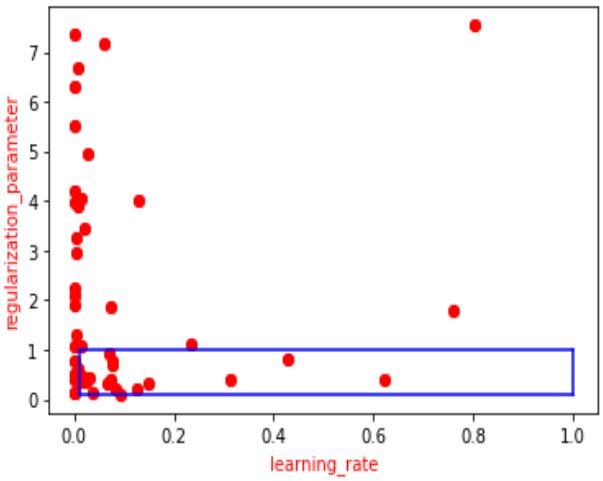

Figure 55. Random Search-Based Coarse Tuning Evaluation: Shrunk scales represented by a blue rectangle

4.3.1.2 Random Search-Based Fine Tuning

According to the results in Section 4.3.1.1, the updated shrunk scales for learning rate and regularization parameter (λ) were [0.01,1] and [0.1,1], respectively. Thus, 500 models, with distinct values for each hyperparameter, were formed by generating 500 binary combinations from the two updated scales, as shown in Figure 56.

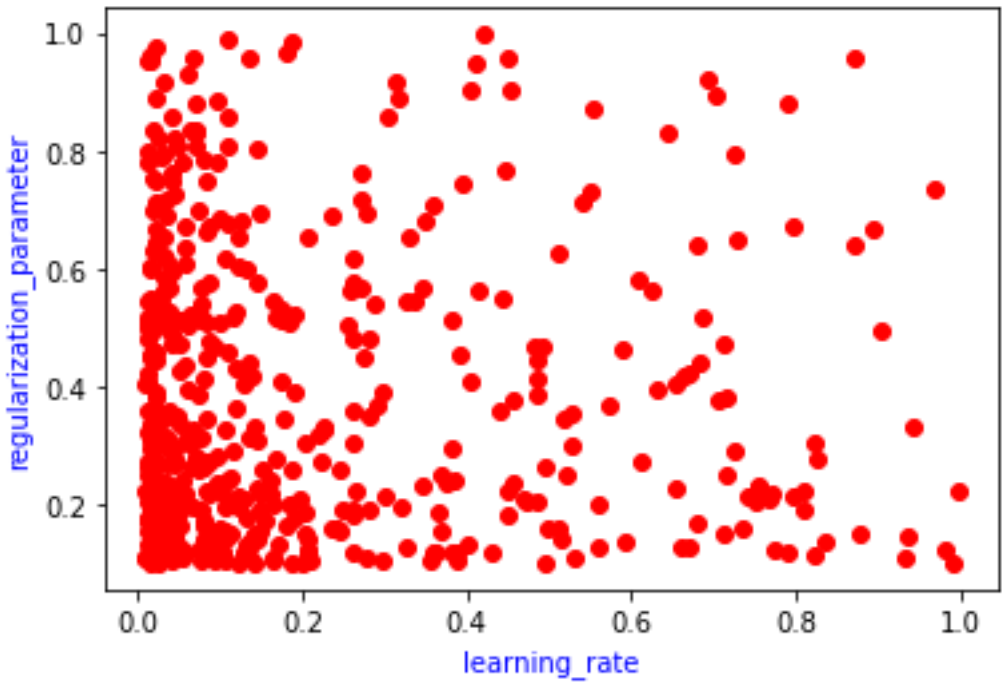

Figure 56. Random Search-Based Fine-Tuning

After that, the model's binary classification accuracy for the cross-validation dataset was used to evaluate the 500 models, as shown in Table 24 and Figure 57.

Table 24. cross-validation accuracy evaluation of fine random search-based tuning

Learning-rate	Regularization-parameter	Train-accuracy (%)	Cross-validation accuracy (%)
0.19088	0.39008	93.36493	96.0
0.47311	0.20664	94.31280	96.0
0.49691	**0.15732**	**94.31280**	**96.0**
0.48558	0.20245	93.83886	96.0
0.17311	0.51256	93.83886	96.0
0.17539	0.53131	94.78673	95.5
0.05001	0.33828	93.83886	95.5
0.25740	0.56234	92.41706	95.5
0.07656	0.51628	94.78673	95.5
0.08167	0.41539	93.83886	95.5
0.03405	0.52650	93.36493	95.5
0.04779	0.34942	93.83886	95.5
0.13871	0.41895	93.83886	95.5
0.11538	0.51971	94.78673	95.5

0.03527	0.31349	93.83886	95.5
0.12002	0.52794	94.78673	95.5
0.18391	0.50868	94.78673	95.5
0.16466	0.54635	94.78673	95.5
0.22304	0.27373	91.94313	95.5

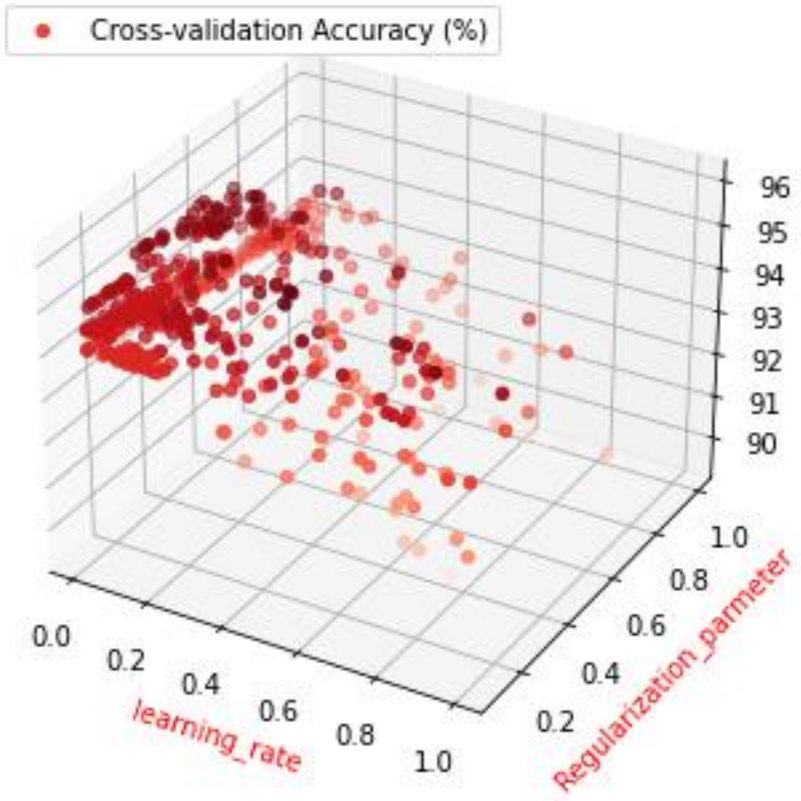

Figure 57. Random Search-Based Fine-Tuning Evaluation

The results show that the random search-based coarse-to-fine tuning managed to find a model that made standard feedforward NN conduct the binary classification with high accuracy of 96% with hyperparameters' values: learning rate = 0.49691 and momentum = 0.15732, as shown in Table 24. By this binary classification accuracy shown in Figure 58, the deep feedforward NN has comparable performance to the state-of-the-art work in [109] that achieved using a RBF kernel-based SVM a binary classification accuracy of 96% for the same dataset, make_moons

provided by scikit-learn library (sklearn), as mentioned in Section 2.5.2. Moreover, the tuning technique managed to decrease the generalization gap between the training accuracy (94.31280 %) and cross-validation accuracy (96 %), as visualized in Figure 58.

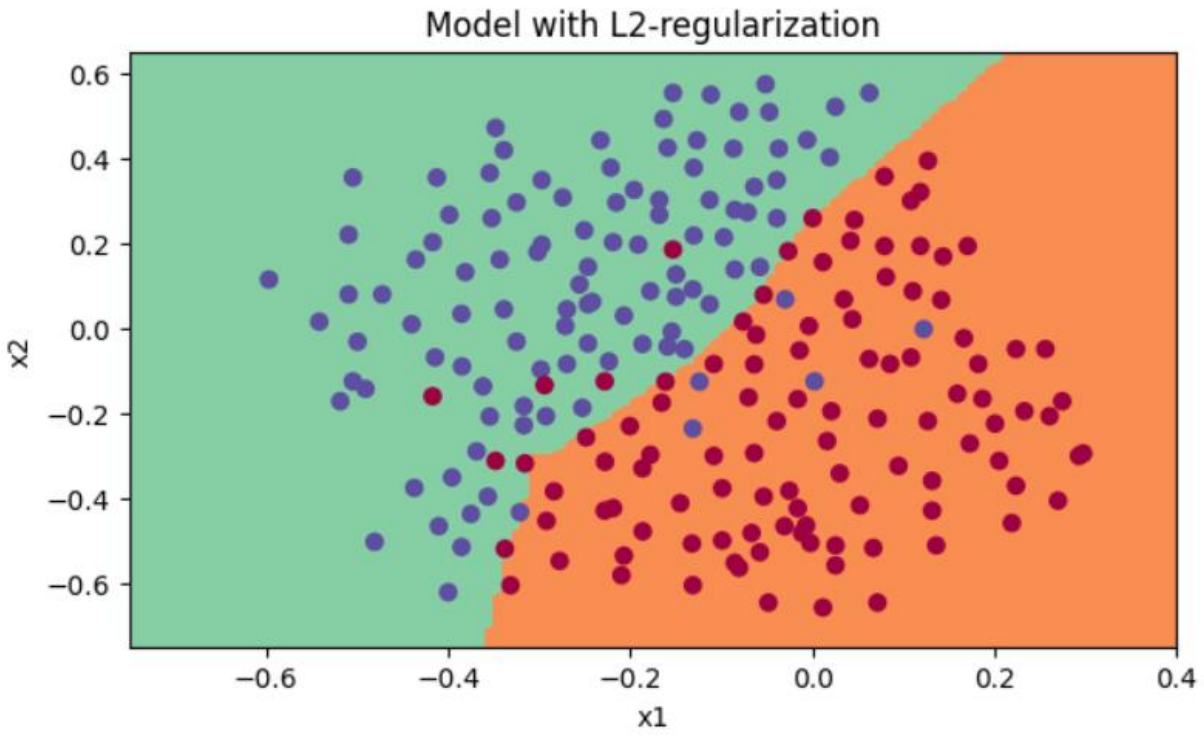

Figure 58. Binary Classification of make_moons with an accuracy of 96%

4.3.2. Hybrid Random and Grid-Search Based Hyperparameter Tuning Technique

4.3.2.1 Random Search-Based Tuning

As discussed in Section 3.2.3, the recommended scales by experts for learning rate and regularization parameter (λ) are [0.0001,1], and [0.1,10], respectively. The same procedures and results exactly as Section 4.3.1.1 were obtained.

4.3.2.2 Grid Search-Based Tuning

According to the results in Section 4.3.2.1, the updated shrunk scales for the learning rate and the regularization parameter (λ) were discretized as follows:

- Learning rate: {0.01,0.055,0.1,0.145, …,1} with step size = 0.045
- Regularization parameter (λ): {0.1,0.14091,0.18182, …,1} with step size = 0.0409

529 combinations of the values within the new discretized scales of the learning rate and regularization parameter (λ). Therefore, they were evaluated based on the learning model's cross-validation-based visualization accuracy, as shown in Figure 59.

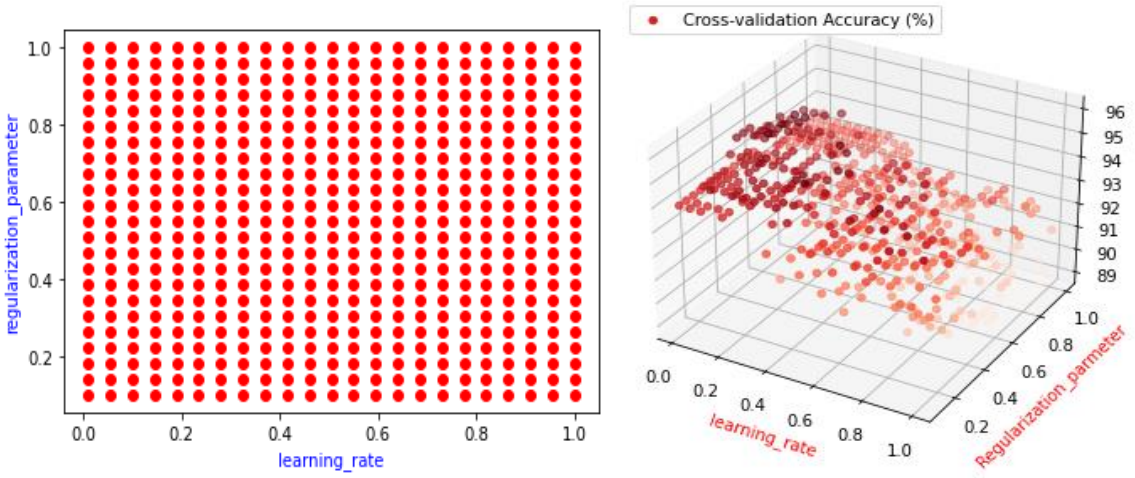

Figure 59. Grid Search-Based Hyperparameter Tuning Evaluation

Table 25 shows the best performing models ordered in descending order based on the learning model's cross-validation accuracy.

Table 25. Cross-validation accuracy of grid search points

Learning_rate	Regularization_parameter	train_accuracy (%)	cross_validation accuracy (%)
0.190	0.38636	93.3649	96.0
0.505	**0.14091**	**94.7867**	**96.0**
0.235	0.50909	93.8389	96.0
0.190	0.55000	94.3128	96.0
0.235	0.55000	92.8910	96.0
0.190	0.46818	93.8389	96.0
0.460	0.42727	93.3649	96.0
0.055	0.42727	93.8389	95.5
0.055	0.38636	93.8389	95.5
0.550	0.34545	93.3649	95.5
0.100	0.46818	93.8389	95.5
0.145	0.59091	93.8389	95.5
0.100	0.59091	93.8389	95.5
0.145	0.46818	93.8389	95.5
0.190	0.67273	93.3649	95.5
0.775	0.18182	94.7867	95.5
0.415	0.26364	92.8910	95.5
0.235	0.67273	93.3649	95.5
0.145	0.55000	94.7867	95.5

The results show that the hybrid random and grid search-based tuning managed to find a model that made standard feedforward NN conduct the binary classification with high accuracy of 96% with hyperparameters' values: learning rate = 0.505 and momentum = 0.14091, as shown in Table 25. By this binary classification accuracy shown in Figure 60, the deep feedforward NN has comparable performance to the state-of-the-art work in [109] that achieved using a RBF kernel-based SVM a binary classification accuracy of 96% for the same dataset, make_moons provided by scikit-learn library (sklearn), as mentioned in Section 2.5.2. The random search is used to narrow the hyperparameter space through which the grid search tunes the hyperparameters. Thus, the process of grid search was speeded up with less computation units. Moreover, the tuning technique managed to decrease the generalization gap between the training

accuracy (94.7867%) and cross-validation accuracy (96%) through selecting a suitable value for the regularization parameter (λ). This generalization gap is smaller than the gap obtained using the random search-based tuning technique.

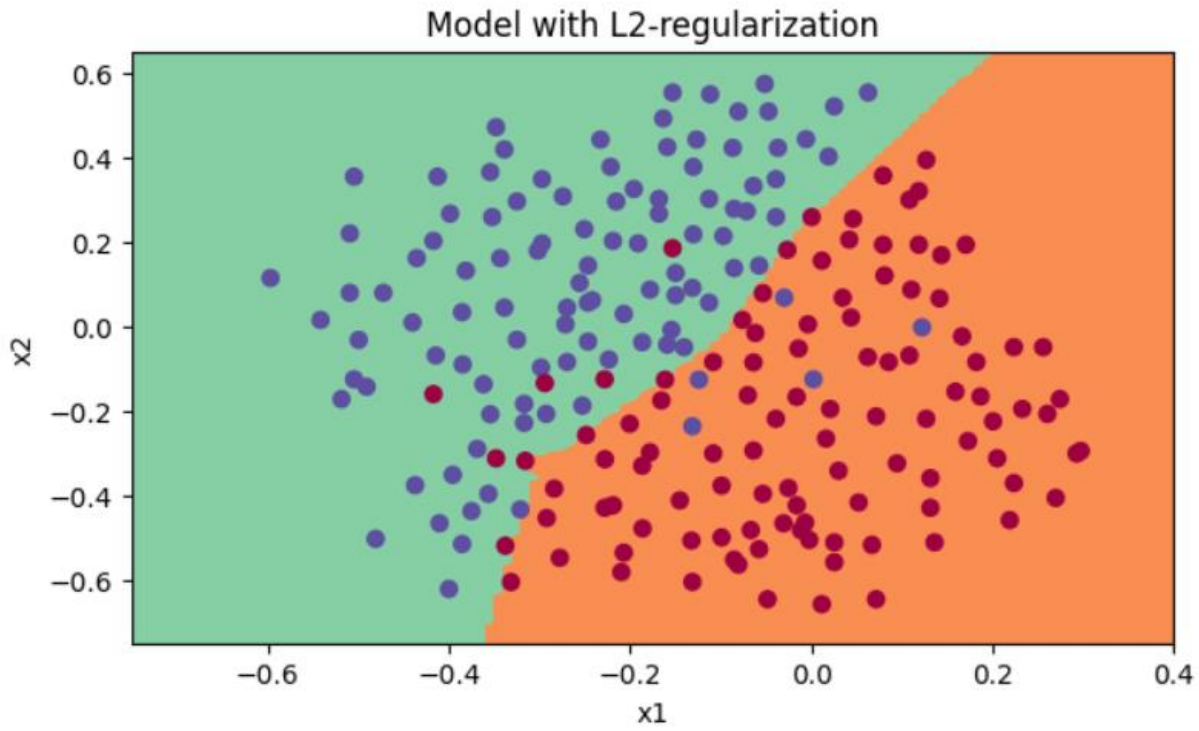

Figure 60. Binary Classification accuracy of 96%: make_moons dataset

4.3.3. Hybrid Random and Manual Search-Based Hyperparameter Tuning Technique

4.3.3.1 Random Search-Based Tuning

As discussed in Section 3.2.3, the recommended scales by experts for learning rate and regularization parameter (λ) are [0.0001,1], and [0.1,10], respectively. The same procedures and results exactly as Section 4.3.1.1 were obtained.

4.3.3.2 Manual Search-Based Tuning

According to the results in Section 4.3.3.1, the updated shrunk scales for the learning rate and the regularization parameter (λ) were discretized as follows:

- Learning rate: {0.01,0.055,0.1,0.145, …,1} with step size = 0.045
- Regularization parameter (λ): {0.1,0.14091,0.18182, …,1} with step size = 0.0409

The two hyperparameters were tuned by walking through the discretized scales one at a time. While a hyperparameter was being tuned by increasing its value within its discrete scale, the other hyperparameters were kept with constant values. The tuning process for each hyperparameter should stop when the learning model starts to give a lower cross-validation accuracy than before, as shown in Figure 61 and Table 26.

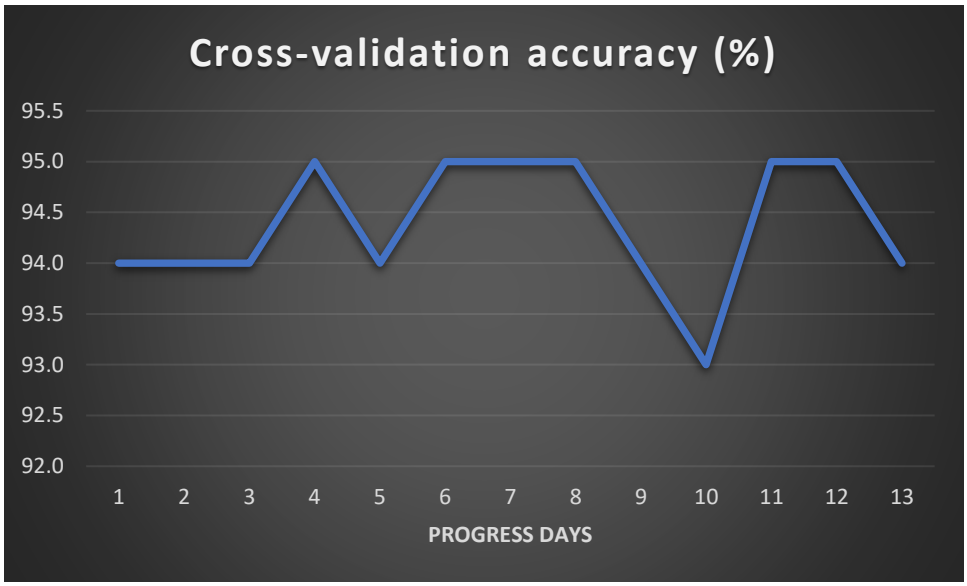

Figure 61. Manual Search based on Cross-Validation accuracy

Table 26. Manual Search based on Cross-Validation accuracy

Learning_rate	Regularization_parameter	Training accuracy (%)	Cross_validation accuracy (%)
Learning_rate			
0.0100	0.1000	92.8910	94.0
0.0550	0.1000	93.8389	94.0
0.1000	0.1000	92.8910	94.0
0.1450	0.1000	92.4171	95.0

0.1900	0.1000	93.3649	94.0
0.2350	0.1000	92.8910	95.0
0.2800	0.1000	92.4171	95.0
0.3250	**0.1000**	**93.8389**	**95.0**
0.3475	0.1000	94.3128	94.0
0.3700	0.1000	91.4692	93.0
Regularization_parameter			
0.3250	**0.1000**	**93.83886**	**95.0**
0.3250	0.1205	0.92417	95.0
0.3250	0.1409	92.89100	94.0

The results prove the impact of random search on obtaining the productive hyperparameter space that would make the standard feedforward NN achieve high binary classification accuracy of 95%, with hyperparameters' values: learning rate = 0.3250 and momentum = 0.1000, in shorter time and with less computation units. This productive hyperparameter space guides the manual search with prior-knowledge that would accelerate the manual search-based hyperparameter tuning. It is obvious when we compare these results with those in Section 4.3.1 and Section 4.3.2 that the hybrid random and manual search-based tuning is less efficient than both random search-based tuning and the hybrid random and grid search-based tuning. Moreover, the tuning technique managed to decrease the generalization gap between the training accuracy (93.83886%) and cross-validation accuracy (95%) through selecting a suitable value for the regularization parameter (λ), as visualized in Figure 62.

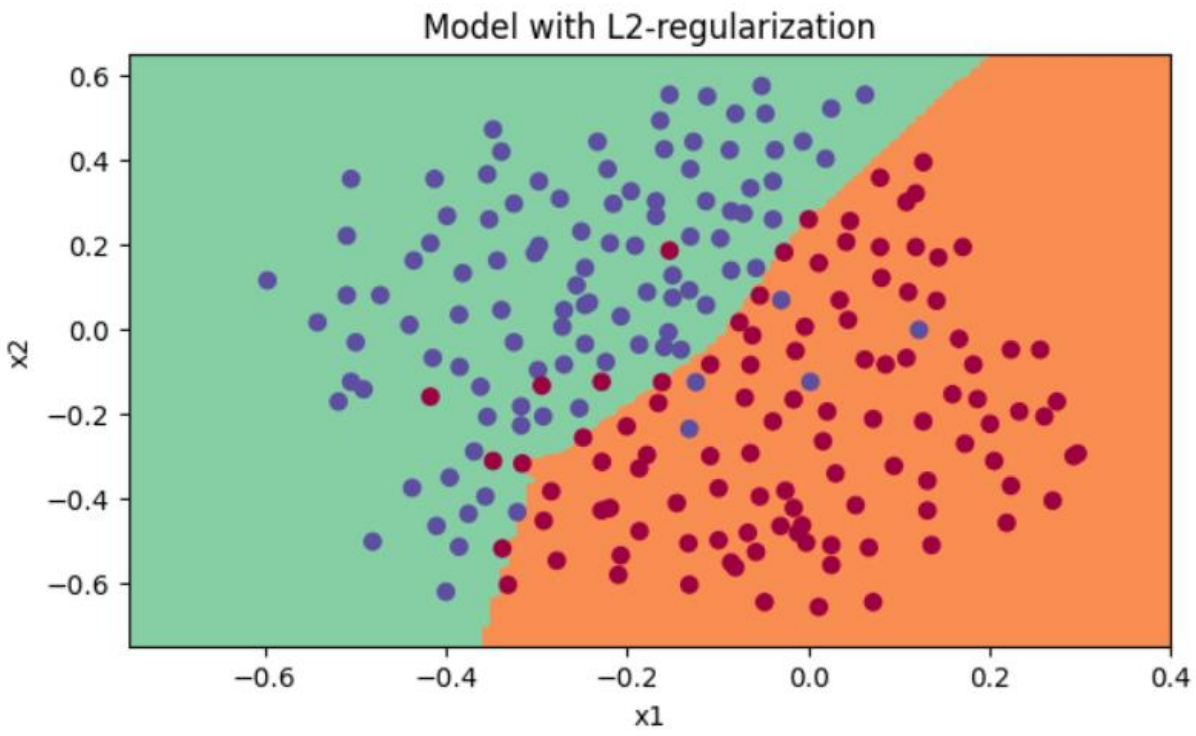

Figure 62. Binary classification accuracy of 95%: make_moons dataset

4.4. Support Vector Machine (SVM)

Based upon Section 3.2.3, the work was carried using three distinct hyperparameter tuning techniques: the random search-based tuning, the hybrid random and grid search-based tuning stage, and the hybrid random and manual search-based tuning. The standard feedforward NN was fed by the dataset, make_moons, as mentioned in Section 2.5.2.

4.4.1. Random Search-Based Hyperparameter Tuning Technique

4.4.1.1 Random Search-Based Coarse Tuning

As discussed in Section 3.2.3, the recommended scales by experts for the regularization parameter (C) and RBF kernel's decay coefficient (σ) are [0.01,30] and [0.01,30], respectively. Therefore, 50 models, with distinct and random values for each hyperparameter, were composed by producing 50 binary combinations from the two scales, as visualized in Figure 63.

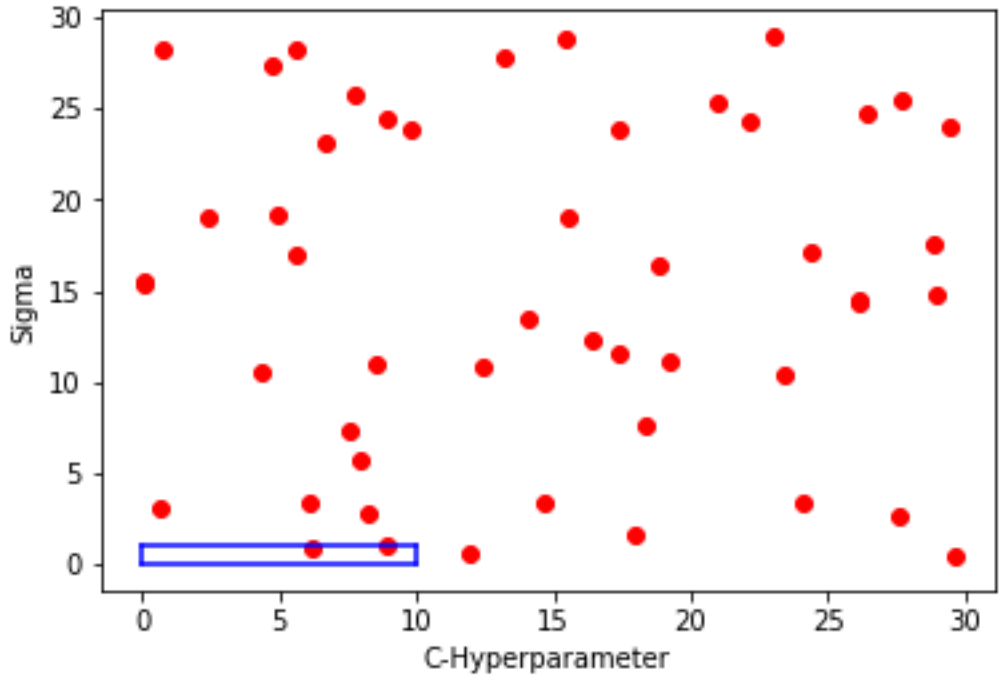

Figure 63. Evaluation of Coarse Random Search-Based Hyperparameter Tuning: Shrunk scales represented by a blue rectangle.

Subsequently, the model's binary classification accuracy for the cross-validation dataset was used to evaluate the 50 models. The hyperparameters' values, that made the SVM model achieve an accuracy of at least 93%, were exploited to shrink the aforementioned initial scales and set smaller ones, as shown in Table 27, Figure 63, and Figure 64.

Table 27. cross-validation accuracy evaluation of coarse random search-based tuning

C-Hyperparameter	Sigma	Cross-Validation accuracy (%)
07.08171	**0.074785**	**94.50**
08.37748	1.557405	93.00
06.13192	2.720513	92.50
04.41453	1.507646	92.50
15.42726	4.581572	92.50
09.23100	3.292534	92.50
24.84318	7.949606	91.00
07.96736	4.890161	90.50
21.40088	9.126181	90.00
15.78761	0.819916	90.00

11.18065	6.038645	90.00
07.67080	0.725054	90.00
20.76059	7.109374	90.00
18.49177	7.727771	90.00
24.49241	9.115573	90.00
21.57886	3.093985	89.50
28.06753	10.56337	89.50
13.30005	8.188076	89.50
11.23669	7.148095	88.50
16.91226	9.181184	88.50
22.62561	11.41278	87.00
12.33111	8.913598	86.50
23.03391	12.67465	85.50
17.08182	11.62793	85.50
26.50687	15.07422	85.00
18.55037	13.80177	83.50
09.65563	27.30578	82.00
08.09098	28.36566	82.00
01.79265	9.966723	82.00
03.87833	28.07651	82.00
25.02118	25.53341	82.00
06.52950	9.896007	82.00
04.43403	12.84301	82.00
13.11491	15.27467	82.00
28.65501	20.92113	82.00
25.62024	22.44929	82.00
15.47592	16.39478	82.00
06.56612	13.17037	82.00
00.06338	3.188395	82.00
23.69041	21.54339	82.00
10.60420	20.90456	82.00
16.67345	18.12112	82.00
14.31609	19.40928	81.50
09.39150	28.95847	81.50
10.18593	21.62891	81.50
01.13613	27.87041	81.50
12.33297	17.59264	81.50
22.88406	29.52208	81.50
05.80351	24.72276	81.50

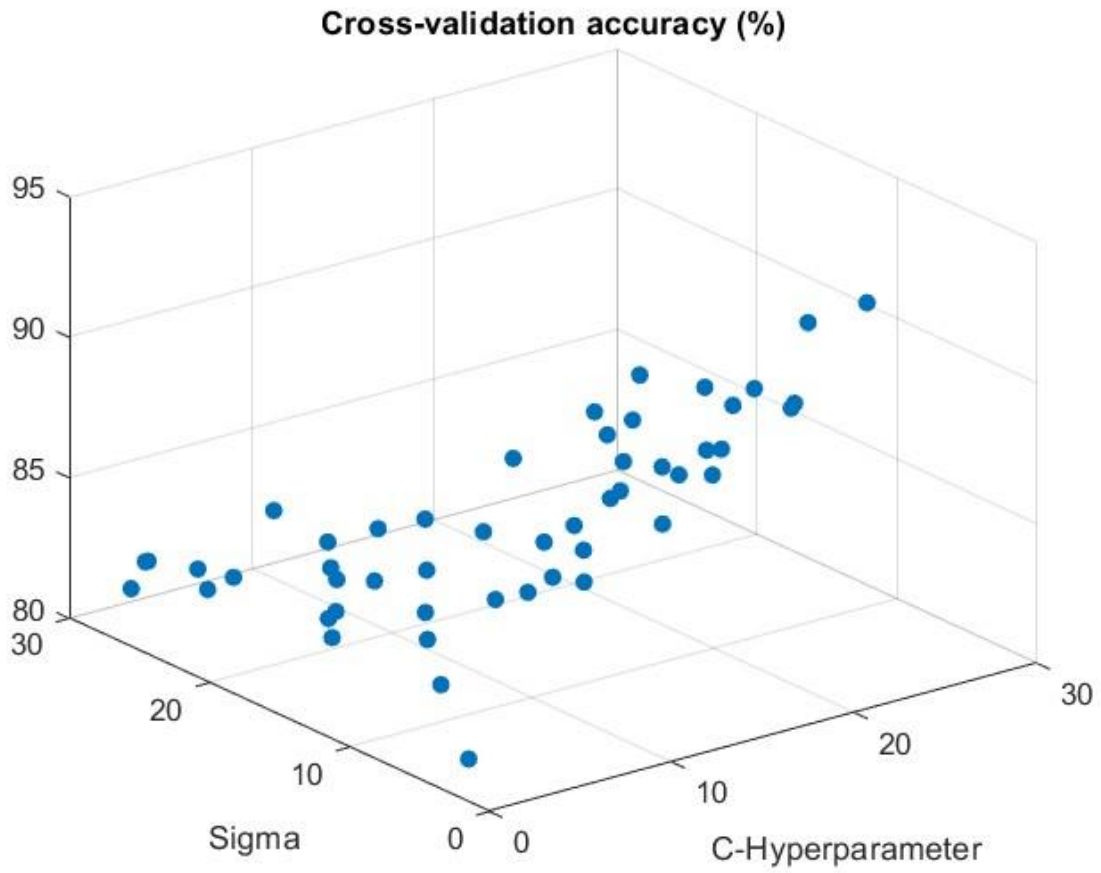

Figure 64. Random Search-Based Coarse Tuning Evaluation

4.3.1.2 Random Search-Based Fine Tuning

According to the results in Section 4.3.1.1, the updated shrunk scales for regularization parameter (C) and RBF kernel's decay coefficient (σ) were [0.01,10] and [0.01,1], respectively. Thus, 500 models, with distinct values for each hyperparameter, were formed by generating 500 binary combinations from the two updated scales, as shown in Figure 65.

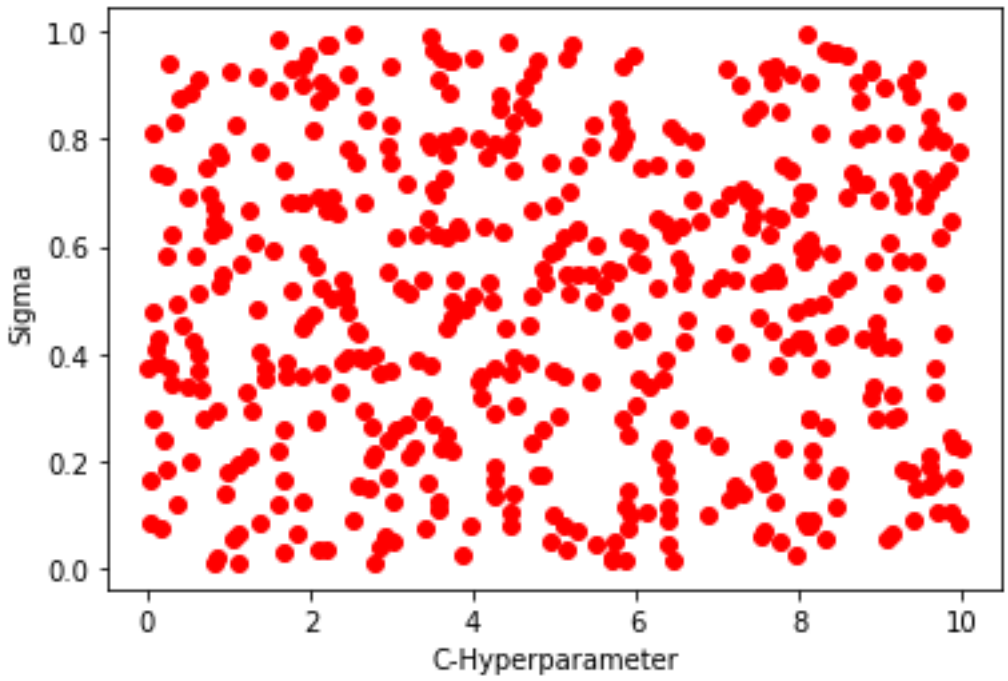

Figure 65. Random Search-Based Fine-Tuning

After that, the model's binary classification accuracy for the cross-validation dataset was used to evaluate the 500 models, as shown in Table 28 and Figure 66.

Table 28. cross-validation accuracy evaluation of fine random search-based tuning

C-Hyperparameter	Sigma	Cross-Validation accuracy (%)
0.63199	**0.09677**	**97.00**
1.03507	0.13515	97.00
1.89893	0.09040	96.50
0.04449	0.08337	96.50
0.87363	0.15585	96.50
1.55704	0.11433	96.50
2.29715	0.09496	96.50
1.23845	0.21886	96.00
0.27034	0.09654	96.00
0.67486	0.14998	96.00
5.30430	0.18807	96.00
2.73004	0.12081	96.00
7.97183	0.21742	96.00
0.58625	0.07106	96.00

5.46759	0.18783	96.00
1.79429	0.10790	96.00
2.13395	0.09709	96.00
0.87316	0.07537	96.00
2.86136	0.08906	96.00

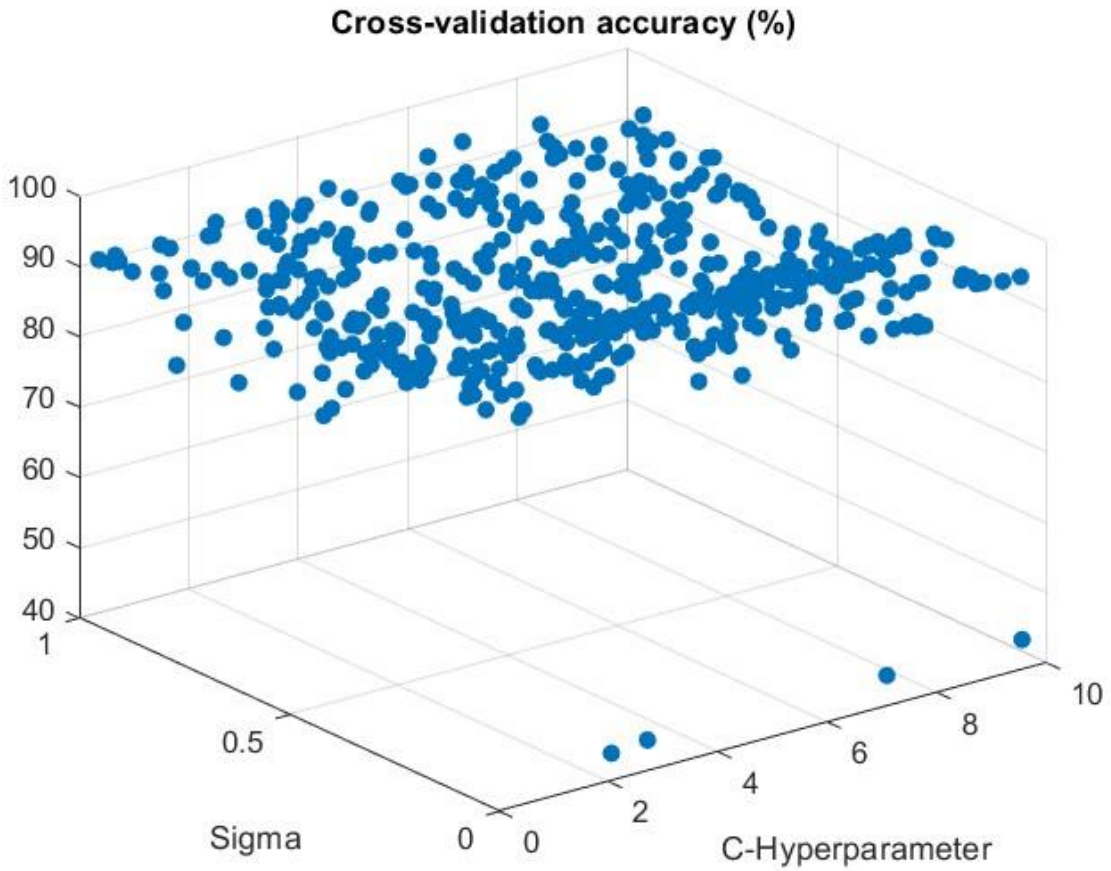

Figure 66. Random Search-Based Fine-Tuning Evaluation

The results show that the random search-based coarse-to-fine tuning managed to find a model that made SVM model conduct the binary classification with high accuracy of 97% with hyperparameters' values: C-hyperparameter = 0.63199 and Sigma = 0.09677, as shown in Table 28. By this binary classification accuracy shown in Figure 67, the deep feedforward NN has comparable performance to the state-of-the-art work in [109] that achieved using a RBF kernel-

based SVM a binary classification accuracy of 96% for the same dataset, make_moons provided by scikit-learn library (sklearn), as mentioned in Section 2.5.2.

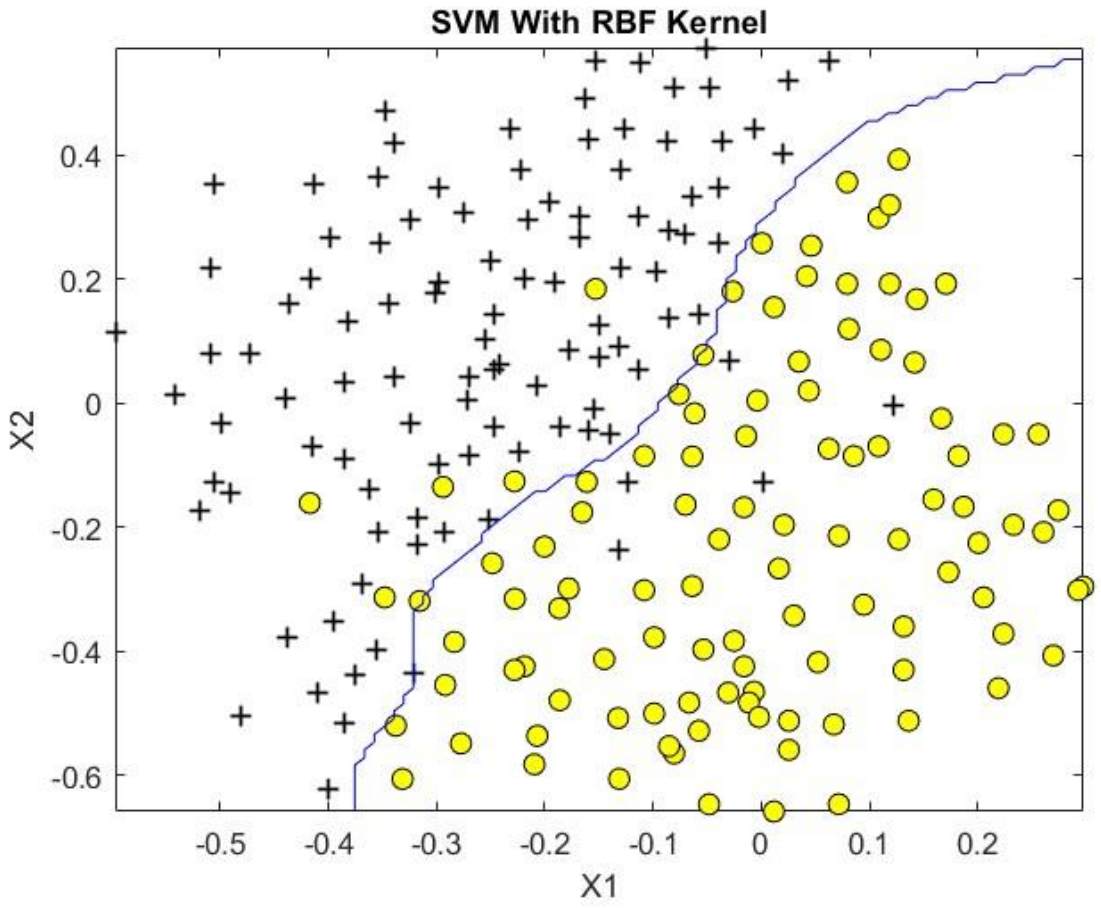

Figure 67. Binary Classification of make_moons dataset with an accuracy of 97%

4.4.2. Hybrid Random and Grid-Search Based Hyperparameter Tuning Technique

4.4.2.1 Random Search-Based Tuning

As discussed in Section 3.2.3, the recommended scales by experts for C-hyperparameter and RBF kernel's decay coefficient (σ) are [0.01,30], and [0.01,30], respectively. The same procedures and results exactly as Section 4.4.1.1 were obtained.

4.4.2.2 Grid Search-Based Tuning

According to the results in Section 4.4.2.1, the updated shrunk scales for the C-hyperparameter and RBF kernel's decay coefficient (σ) were discretized as follows:

- C-hyperparameter: {0.01,0.46409,0.91818, …,10} with step size = 0.45409
- RBF kernel's decay coefficient (σ): {0.01,0.055,0.1, …,1} with step size = 0.045

529 combinations of the values within the new discretized scales of the C-hyperparameter and RBF kernel's decay coefficient (σ). Therefore, they were evaluated based on the learning model's cross-validation binary classification accuracy, as shown in Figure 68.

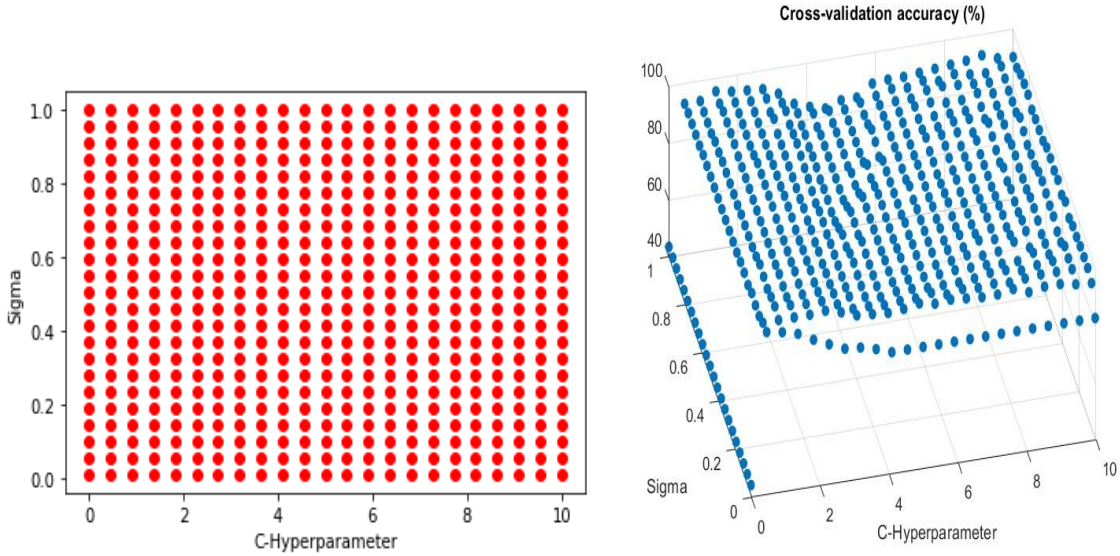

Figure 68. Grid Search-Based Hyperparameter Tuning Evaluation

Table 29 shows the best performing models ordered in descending order based on the learning model's cross-validation accuracy.

Table 29. Cross-validation accuracy of grid search points

C-Hyperparameter	Sigma	Cross_Validation accuracy (%)
0.91818	**0.100**	**97.00**
0.46409	0.010	96.50
0.91818	0.145	96.50
0.91818	0.280	96.50
1.37227	0.100	96.50
2.28045	0.055	96.50
0.46409	0.055	96.00
0.91818	0.055	96.00
0.91818	0.235	96.00
1.37227	0.055	96.00
1.37227	0.235	96.00
1.37227	0.775	96.00
1.37227	0.820	96.00
1.37227	0.865	96.00
1.37227	0.910	96.00
1.82636	0.055	96.00
1.82636	0.100	96.00
1.82636	0.235	96.00
1.82636	0.505	96.00

The results show that the hybrid random and grid search-based tuning managed to find a model that made SVM model conduct the binary classification with high accuracy of 97% with hyperparameters' values: C-hyperparameter = 0.91818 and sigma = 0.1, as shown in Table 29. By this binary classification accuracy shown in Figure 69, the deep feedforward NN has comparable performance to the state-of-the-art work in [109] that achieved using a RBF kernel-based SVM a binary classification accuracy of 96% for the same dataset, make_moons provided by scikit-learn library (sklearn), as mentioned in Section 2.5.2. The random search is used to narrow the hyperparameter space through which the grid search tunes the hyperparameters. Thus, the process of grid search was speeded up with less computation units.

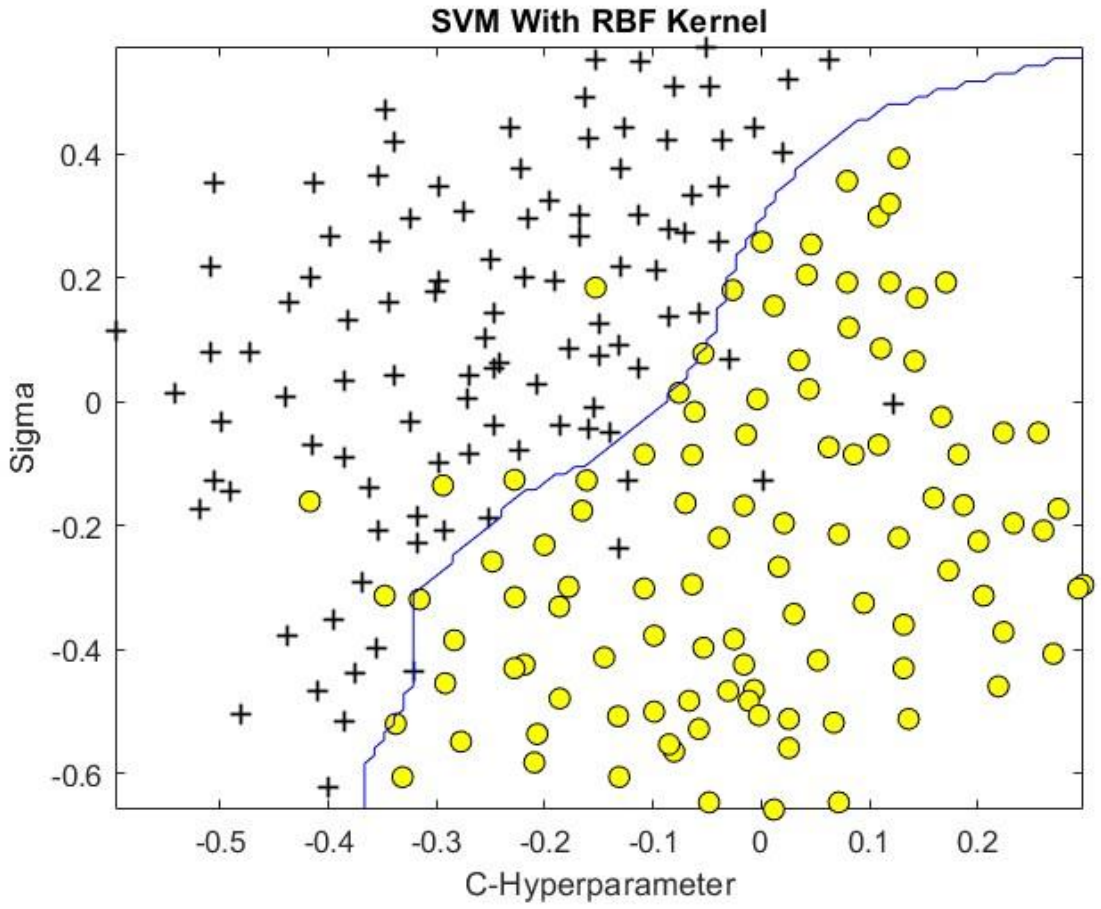

Figure 69. Binary Classification accuracy of 97%: make_moons dataset

4.4.3. Hybrid Random and Manual Search-Based Hyperparameter Tuning Technique

4.4.3.1 Random Search-Based Tuning

As discussed in Section 3.2.3, the recommended scales by experts for C-hyperparameter and RBF kernel's decay coefficient (σ) are [0.01,30], and [0.01,30], respectively. The same procedures and results exactly as Section 4.4.1.1 were obtained.

4.4.3.2 Manual Search-Based Tuning

According to the results in Section 4.4.3.1, the updated shrunk scales for the C-hyperparameter and RBF kernel's decay coefficient (σ) were discretized as follows:

- C-hyperparameter: {0.01,0.46409,0.91818, …,10} with step size = 0.45409
- RBF kernel's decay coefficient (σ): {0.01,0.055,0.1, …,1} with step size = 0.045

The two hyperparameters were tuned by walking through the discretized scales one at a time. While a hyperparameter was being tuned by increasing its value within its discrete scale, the other hyperparameters were kept with constant values. The tuning process for each hyperparameter should stop when the learning model starts to give a lower cross-validation accuracy than before, as shown in Figure 70 and Table 30.

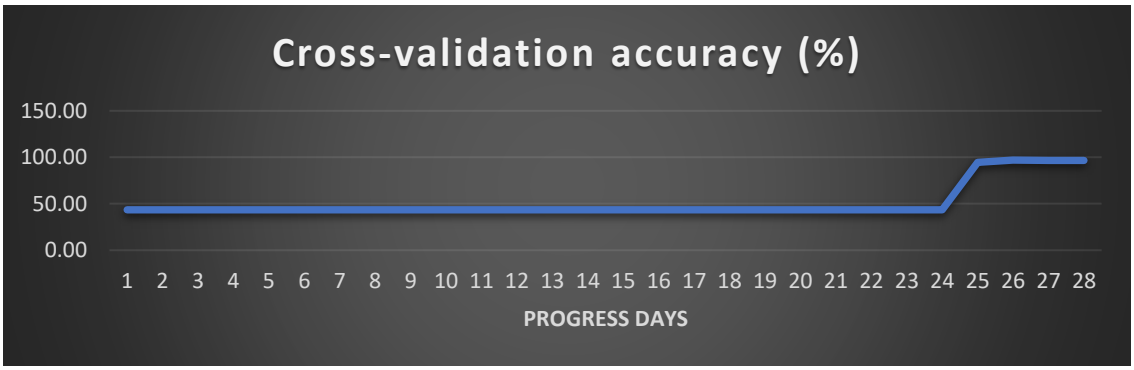

Figure 70. Manual Search based on Cross-Validation accuracy

Table 30. Manual Search based on Cross-Validation accuracy

C-Hyperparameter	Sigma	Cross-validation accuracy (%)
Sigma Tuning		
0.01000	0.010	43.50
0.01000	0.055	43.50
0.01000	**0.100**	**43.50**
0.01000	0.145	43.50
0.01000	0.190	43.50
0.01000	0.235	43.50
0.01000	0.280	43.50
0.01000	0.325	43.50
0.01000	0.370	43.50
0.01000	0.415	43.50
0.01000	0.460	43.50
0.01000	0.505	43.50

0.01000	0.550	43.50
0.01000	0.595	43.50
0.01000	0.640	43.50
0.01000	0.685	43.50
0.01000	0.730	43.50
0.01000	0.775	43.50
0.01000	0.820	43.50
0.01000	0.865	43.50
0.01000	0.910	43.50
0.01000	0.955	43.50
0.01000	1.000	43.50
C-Hyperparameter Tuning		
0.01000	0.100	43.50
0.46409	0.100	94.50
0.91818	**0.100**	**97.00**
1.14523	0.100	96.50
1.37227	0.100	96.5

The results prove the impact of random search on obtaining the productive hyperparameter space that would make the SVM model achieve high binary classification accuracy of 97%, with hyperparameters' values: C-hyperparameter = 0.91818 and momentum = 0.1000, in shorter time and with less computation units. This productive hyperparameter space guides the manual search with prior-knowledge that would accelerate the manual search-based hyperparameter tuning. It is obvious when we compare these results with those in Section 4.4.1 and Section 4.4.2 that the hybrid random and manual search-based tuning is as efficient as both random search-based tuning and the hybrid random and grid search-based tuning, as visualized in Figure 71.

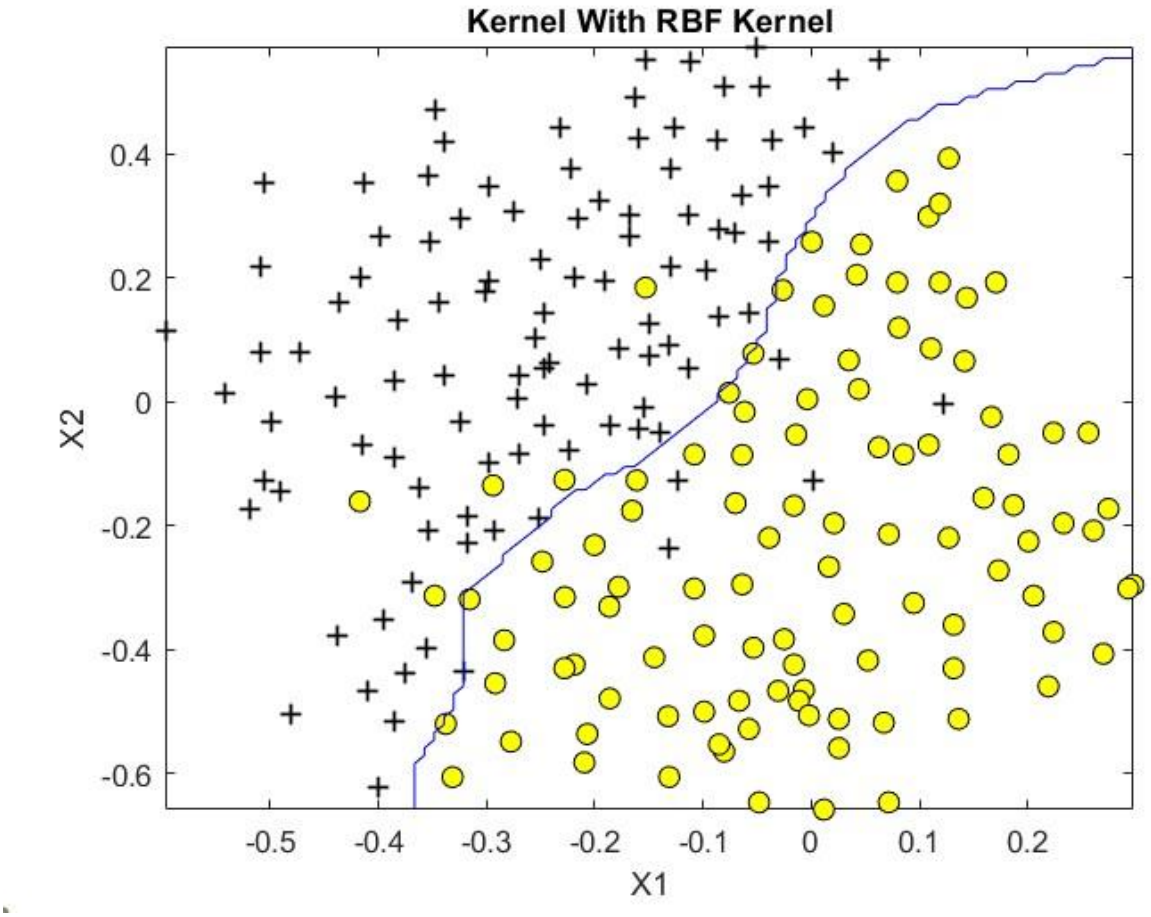

Figure 71. Binary classification accuracy of 97%: make_moons dataset

4.5. Principal Component Analysis

Based upon Section 3.2.3, the work was carried using three distinct hyperparameter tuning techniques: the random search-based tuning, the hybrid random and grid search-based tuning stage, and the hybrid random and manual search-based tuning. The PCA model was fed by the dataset, the human faces, as mentioned in Section 2.5.3.

4.5.1. Random Search-Based Hyperparameter Tuning Technique

4.5.1.1 Random Search-Based Coarse Tuning

As discussed in Section 3.2.3, the length of the unrolled vector of the images in the dataset is 1024. Therefore, the initial scale of the number of PCA components (K) was set as follows: $K \in \{1,2,3, \ldots,1024\}$ with a step size equal to 1. Therefore, 50 models, with distinct and random values for the hyperparameter, were composed by producing 50 unary combinations from the initial scale, as visualized in Figure 72.

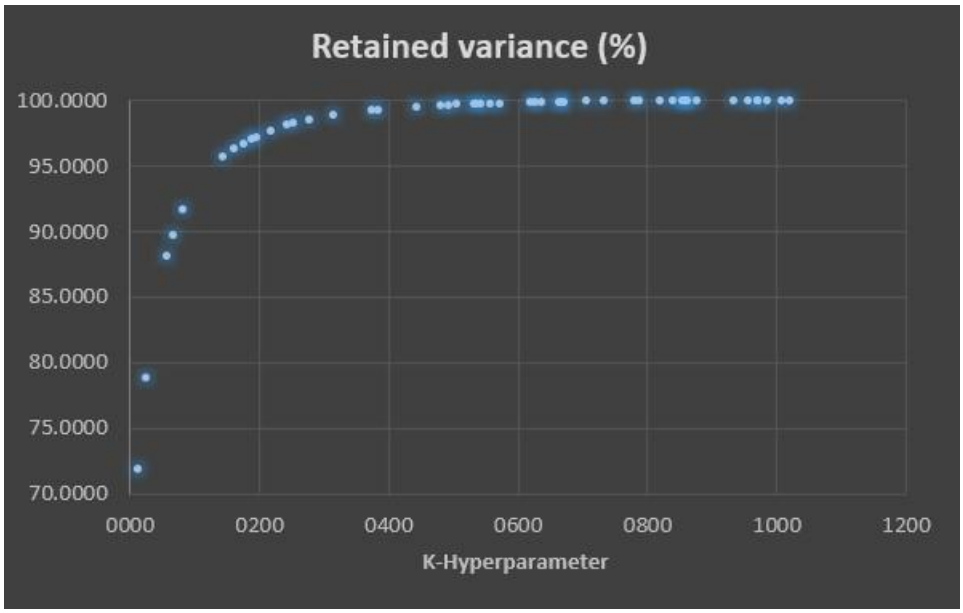

Figure 72. Evaluation of Coarse Random Search-Based Hyperparameter Tuning

Subsequently, the dataset's retained variance was used to evaluate the 50 models. The criterion of evaluation is to get the smallest K that produces dimensionally-reduced dataset with retained variance of at least 90% of the original dataset's variance, as described in Section 3.2.5. This evaluation was exploited to shrink the aforementioned initial scale and set a smaller one, as shown in Table 31.

Table 31. retained variance evaluation of coarse random search-based tuning

K-Hyperparameter	Retained variance (%)
1020	99.9999
1009	99.9995
0986	99.9984
0972	99.9974
0971	99.9973
0956	99.9960
0935	99.9938
0878	99.9847
0862	99.9813
0859	99.9807
0857	99.9802
0855	99.9797
0839	99.9757
0819	99.9699
0789	99.9598
0781	99.9567
0734	99.9350
0706	99.9187
0672	99.8949
0667	99.8909
0664	99.8885
0637	99.8647
0628	99.8559
0620	99.8477
0572	99.7894
0558	99.7689
0542	99.7433
0536	99.7331
0532	99.7260
0505	99.6736
0493	99.6474
0481	99.6190
0443	99.5126
0385	99.2881
0374	**99.2344**
0316	98.8644
0277	98.5033
0253	98.2149
0244	98.0907
0219	97.6864
0196	97.2181

0190	97.0774
0176	96.7129
0162	96.2890
0145	95.6718
0083	91.6812
0067	89.7531
0057	88.1543
0025	78.7695
0014	71.8058

4.5.1.2 Random Search-Based Fine Tuning

According to the result in Section 4.5.1.1, the updated shrunk scale for the number of PCA components (K) was {317,318,319, …,374} with a step size of 1. Thus, 50 models, with distinct values for the hyperparameter, were formed by generating 50 unary combinations from the updated scale, as shown in Figure 73.

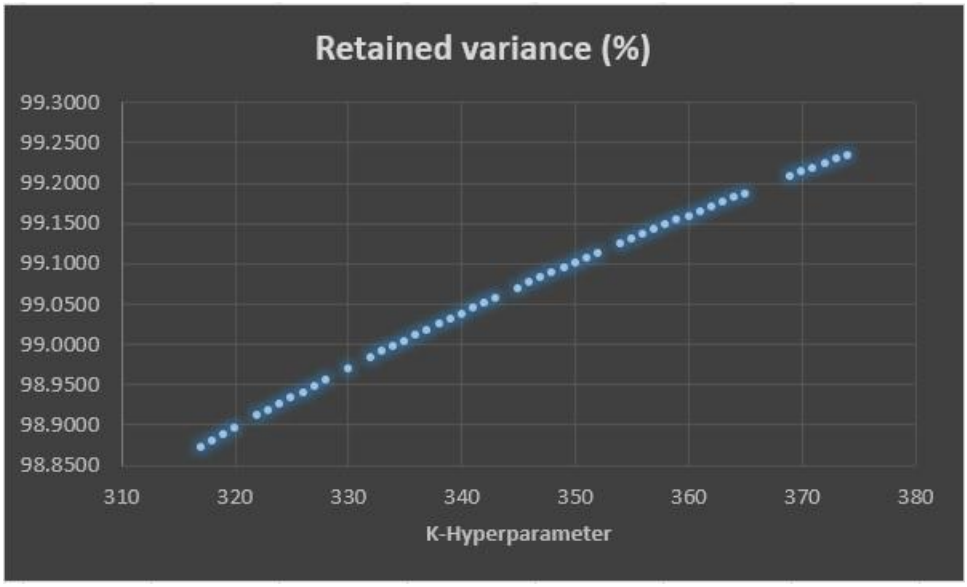

Figure 73. Random Search-Based Fine-Tuning

After that, the dimensionally-reduced dataset's retained variance was used to evaluate the 50 unary combinations of the K-hyperparameter, as shown in Table 32.

Table 32. retained variance evaluation of fine random search-based tuning

K-Hyperparameter	Retained variance (%)
374	99.2344
373	99.2293
372	99.2241
371	99.2189
370	99.2137
369	99.2084
365	99.1869
364	99.1814
363	99.1759
362	99.1704
361	99.1648
360	99.1591
359	99.1535
358	99.1478
357	99.1420
356	99.1362
355	99.1304
354	99.1246
352	99.1127
351	99.1067
350	99.1006
349	99.0945
348	99.0884
347	99.0822
346	99.0760
345	99.0697
343	99.0571
342	99.0507
341	99.0442
340	99.0378
339	99.0312
338	99.0245
337	99.0178
336	99.0111
335	**99.0042**
334	98.9973

333	98.9904
332	98.9835
330	98.9694
328	98.9550
327	98.9478
326	98.9405
325	98.9332
324	98.9258
323	98.9183
322	98.9108
320	98.8955
319	98.8878
318	98.8800
317	98.8722

The results show that the random search-based coarse-to-fine tuning managed to find a model that made PCA model conduct the dimensionality reduction with K = 335 retaining 99.0042% of the original dataset's variance, as shown in Table 32 and Figure 74.

Figure 74. Dimensionally-reduced human faces at K = 335

4.5.2. Hybrid Random and Grid-Search Based Hyperparameter Tuning Technique

4.5.2.1 Random Search-Based Tuning

As discussed in Section 3.2.3, the length of the unrolled vector of the images in the dataset is 1024. Therefore, the initial scale of the number of PCA components (K) was set as follows: $K \in \{1,2,3, \ldots,1024\}$ with a step size equal to 1. The same procedures and results exactly as Section 4.5.1.1 were obtained.

4.5.2.2 Grid Search-Based Tuning

According to the results in Section 4.5.2.1, the updated shrunk scale for the number of PCA components (K) was discretized as follows:

- K-hyperparameter: {317,318,319, …,374} with a step size of 1.

Therefore, these K values were evaluated based on the dimensionally-reduced dataset's retained variance of the original dataset's variance, as visualized in Figure 75.

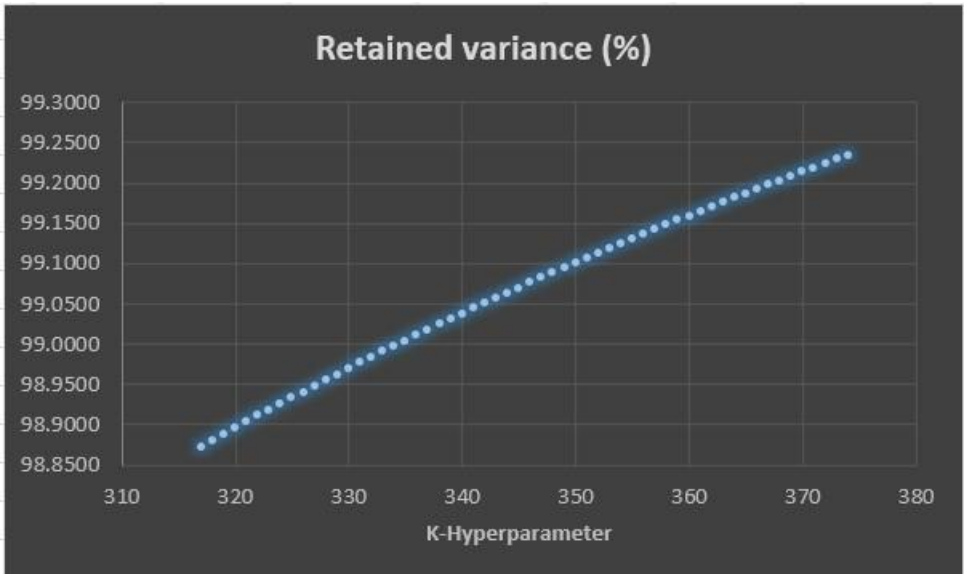

Figure 75. Grid tuning evaluation based on retained variance

Table 33 shows the best performing models ordered in descending order based on the learning model's cross-validation accuracy.

Table 33. retained variance evaluation of grid search

K-Hyperparameter	Retained variance (%)
374	99.2344
373	99.2293
372	99.2241
371	99.2189
370	99.2137
369	99.2084
368	99.2031
367	99.1977
366	99.1923
365	99.1869
364	99.1814
363	99.1759
362	99.1704
361	99.1648
360	99.1591
359	99.1535
358	99.1478
357	99.1420
356	99.1362
355	99.1304
354	99.1246
353	99.1187
352	99.1127
351	99.1067
350	99.1006
349	99.0945
348	99.0884
347	99.0822
346	99.0760
345	99.0697
344	99.0634
343	99.0571
342	99.0507
341	99.0442
340	99.0378
339	99.0312

338	99.0245
337	99.0178
336	99.0111
335	**99.0042**
334	98.9973
333	98.9904
332	98.9835
331	98.9764
330	98.9694
329	98.9622
328	98.9550
327	98.9478
326	98.9405
325	98.9332
324	98.9258
323	98.9183
322	98.9108
321	98.9032
320	98.8955
319	98.8878
318	98.8800
317	98.8722

The results show that the hybrid random and grid search-based tuning managed to find a model that made PCA model conduct the dimensionality reduction with K = 335 retaining 99.0042% of the original dataset's variance, as shown in Table 33 and Figure 76.

Figure 76. Dimensionally-reduced Images with K = 335

4.5.3. Hybrid Random and Manual Search-Based Hyperparameter Tuning Technique

4.5.3.1 Random Search-Based Tuning

As discussed in Section 3.2.3, the length of the unrolled vector of the images in the dataset is 1024. Therefore, the initial scale of the number of PCA components (K) was set as follows: K ∈ {1,2,3, …,1024} with a step size equal to 1. The same procedures and results exactly as Section 4.5.1.1 were obtained.

4.5.3.2 Manual Search-Based Tuning

According to the results in Section 4.5.3.1, the updated shrunk scale for the number of PCA components (K) was discretized as follows:

- K-hyperparameter: {317,318,319, …,374} with a step size of 1.

Therefore, these K values were evaluated based on the dimensionally-reduced dataset's retained variance, as shown in Figure 77 and Table 34. The hyperparameters were tuned by

walking through the discretized scale in order until getting retained variance of at least 90% of the original dataset's variance.

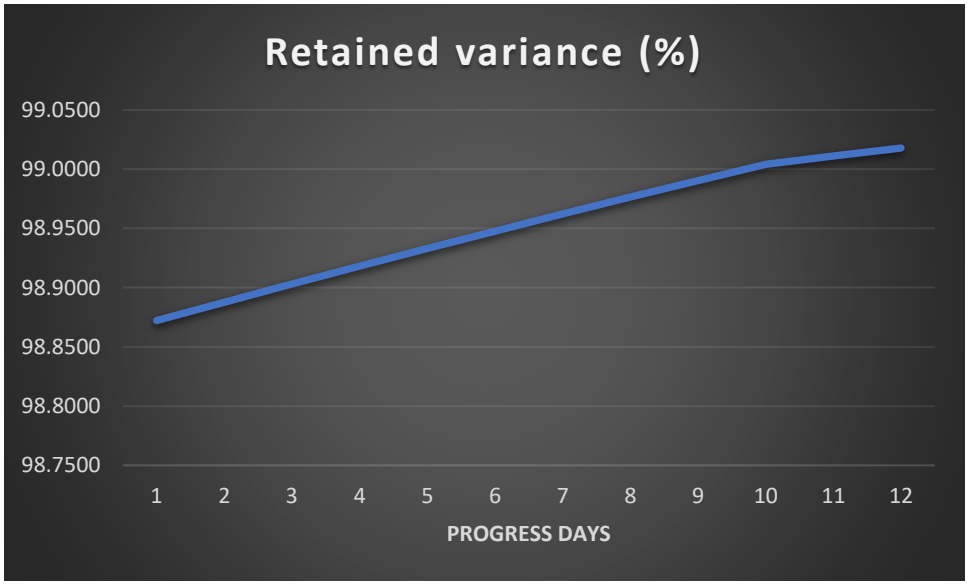

Figure 77. Retained variance evaluation of manual search

Table 34. Manual Search based on retained variance of the dimensionally-reduced dataset

K-Hyperparameter	Retained variance (%)
317	98.8722
319	98.8878
321	98.9032
323	98.9183
325	98.9332
327	98.9478
329	98.9622
331	98.9764
333	98.9904
335	**99.0042**
336	99.0111
337	99.0178

The results prove the impact of random search on obtaining the productive hyperparameter space that would make the PCA model achieve the dimensionality reduction keeping 99.0042% of the original dataset's variance using 335 PCA components in a shorter time and with less computation units, as shown in Figure 78. This productive hyperparameter space guides the manual search with prior-knowledge that would accelerate the manual search-based hyperparameter tuning. It is obvious when we compare these results with those in Section 4.5.1 and Section 4.5.2 that the hybrid random and manual search-based tuning is as efficient as both random search-based tuning and the hybrid random and grid search-based tuning.

Figure 78. Dimensionally-reduced Images with K = 335

4.6. Comparison Between the Hyperparameter Tuning Techniques for Each Learning Model Based on the Performance and Computation Time

It is clear, based on Table 35, that merging random search with grid search optimizes the performance of the grid search. Particularly, the hybrid random and grid search can improve

the learning model's accuracy as efficiently as the random search within the same computation time approximately while the hyperparameter space has low dimensionality up to three hyperparameters. Therefore, mixing random search with grid search mitigates the exhaustive computation effort needed by the pure grid search. Moreover, the hybrid random and manual search takes advantage of random search as prior knowledge to manually tune the hyperparameters within a narrow scale and, thereby, less computation time. It gives the learning model's accuracy less than both random search and the hybrid random and grid but in a shorter time. Hence, manual search or babysitting one model method is a shortcut hyperparameter tuning technique. The computation time in Table 35 is the total number of training epochs divided by 100. Notably, the model needs at most 100 training epochs to be trained, so the computation time in Table 35 is the percentage portion of the maximum training time corresponding to 100 training iterations.

When the learning model has low dimensional hyperparameter space, non-iterative hyperparameter optimization techniques such as random and grid search are more efficient in terms of computation time and storage resources consumption than iterative hyperparameter optimization techniques such as gradient-based, population-based training (PBT), and Bayesian techniques. Therefore, it is a wise decision to select these three hyperparameter optimization techniques and prefer them over the iterative hyperparameter optimization techniques in this thesis that uses at most three hyperparameters in all experiments discussed in Section 3.2. Yang et al proved empirically in [15] that grid and random search outperforms the iterative Bayesian-based hyperparameter optimization technique in terms of classification accuracy and computation time for a SVM model that has only two hyperparameters. Accordingly, if an iterative technique was applied in this

thesis, it would make the learning models obtain less efficient performance and consume more computation time and storage units for replacement and exploitation processes explained in Section 2.1.

Table 35. Comparison of the evaluation of the three hyperparameter tuning techniques for each learning model

Dataset	Learning Model		Random Search	Hybrid Random and Grid Search	Hybrid Random and Manual Search
Pointcloud-based images for submarine pipelines	PointNet CNN	F1-score of pixel-wise classification	overall average F1 score of 93.6% F1-score|pool = 96.4%, F1-score|sea = 90.8%	overall average F1 score of 93.6% F1-score|pool = 96.4%, F1-score|sea = 90.8%	overall average F1 score of 90.97% F1-score|pool = 95.93%, F1-score|sea = 86.02%
		Computation Time	280 time-unit	85 time-unit	41 time-unit
		Hyperparameter Values	Mini-batch size = 16 Learning rate = 0.001 Momentum = 0.999	Mini-batch size = 16 Learning rate = 0.001 Momentum = 0.999	Mini-batch size = 16 Learning rate = 0.00159 Momentum = 0.9
make_moons, provided by scikit-learn library (sklearn)	FNN Without Regularization	Cross-validation binary classification accuracy	96.67 %	96.67 %	96%
		Computation Time	550 time-unit	579 time-unit	68 time-unit
		Hyperparameter Values	Mini-batch size = 16 Learning rate = 0.35476 Momentum = 0.992	Mini-batch size = 16 Learning rate = 0.145 Momentum = 0.9675	Mini-batch size = 16 Learning rate = 0.1 Momentum = 0.936

make_moons, provided by scikit-learn library (sklearn)	FNN With Regularization	Cross-validation binary classification accuracy	Training: 94.31280 % cross-validation: 96 %	Training: 94.7867 % cross-validation: 96 %	Training: 93.83886% cross-validation: 95%
		Computation Time	550 time-unit	579 time-unit	63 time-unit
		Hyperparameter Values	Learning rate = 0.49691 Regularization Parameter = 0.15732	Learning rate = 0.505 Regularization Parameter = 0.14091	Learning rate = 0.325 Regularization Parameter = 0.1
make_moons, provided by scikit-learn library (sklearn)	RBF kernel-Based SVM	Cross-validation binary classification accuracy	97%	97%	97%
		Computation Time	550 time-unit	579 time-unit	78 time-unit
		Hyperparameter Values	C-hyperparameter = 0.63199 Sigma = 0.09677	C-hyperparameter = 0.91818 Sigma = 0.1	C-hyperparameter = 0.91818 Sigma = 0.1
Images of human faces	PCA	Number of PCA components (K)	335	335	335
		Retained Variance	99.0042%	99.0042%	99.0042%
		Computation Time	100 time-unit	108 time-unit	62 time-unit
		Hyperparameter Values	K = 335	K = 335	K = 335

4.7. Comparison Between Thesis Results and the Results of the State-of-the-art Publications

As shown in Table 36, the proposed work succeeded in tuning hyperparameters to obtain, among the best-fit models shown in Table 10, a model that gave an overall average F1 score of 93.6% (F1-score|pool = 96.4%, F1-score|sea = 90.8%) using the following hyperparameter's values: mini-batch size = 16, momentum = 0.9985268, and learning rate = 0.001466154. By this score, the proposed technique outperforms and generalizes better to the sea dataset than the state-of-the-art work [64] which has a maximum overall average F1 score of 92.75% (F1-score|pool = 96.2%, F1-score|sea = 89.3%) with the same data and model but with different hyperparameters' values: mini-batch size = 16, momentum = 0.9, and learning rate = 0.001. This score indicates the low variance and small generalization gap that the model has, thanks to selecting a small mini-batch size and a significant momentum coefficient of more than 0.9 because both of them improve the model's generalization, as discussed in section 2.4.1.2.

Moreover, the results show that the random search-based coarse-to-fine tuning managed to find a model that made standard feedforward NN either with or without regularization conduct the binary classification with high accuracy of 96.67 % and 96 %, respectively, using the hyperparameters' values: mini-batch size = 16, learning rate = 0.35476, momentum = 0.99200, as shown in Table 20. By this binary classification accuracy, the deep feedforward neural network (FNN) outperforms the state-of-the-art work in [109] that achieved using a RBF kernel-based SVM a binary classification accuracy of 96 % for the same dataset, make_moons provided by scikit-learn library (sklearn), as mentioned in

Section 2.5.2. Therefore, this indicates the superiority of deep neural networks over traditional machine learning algorithm such as SVM.

For the SVM model, the results show that the random search-based coarse-to-fine tuning managed to find a model that made the SVM model conduct the binary classification with a high accuracy of 97% with hyperparameters' values: C-hyperparameter = 0.63199 and Sigma = 0.09677, as shown in Table 28 and Table 36. By this binary classification accuracy, the model surpasses, in terms of classification accuracy, the state-of-the-art work in [109] that achieved using an RBF kernel-based SVM a binary classification accuracy of 96% for the same dataset, make_moons provided by scikit-learn library (sklearn), as mentioned in Section 2.5.2.

For PCA, the results show that the used tuning techniques managed to find a model that made PCA model conduct the dimensionality reduction with K = 335 retaining 99.0042% of the original dataset's variance, achieving comparable results with the state-of-the-art article [110].

Table 36. Comparing Obtained Results With the Recent Articles' Results

Dataset	Learning Model		Results	
	Thesis	The state-of-the-art Publications	Thesis	The state-of-the-art Publications
Pointcloud-based images for submarine pipelines	PointNet CNN	PointNet CNN	overall average F1 score of 93.6% F1-score|pool = 96.4%, F1-score|sea = 90.8%	overall average F1 score of 92.75% F1-score|pool = 96.2%, F1-score|sea = 89.3%) [64]

make_moons, provided by scikit-learn library (sklearn)	FNN Without Regularization	RBF kernel-Based SVM	Cross-validation binary classification accuracy: 96.67 %	Cross-validation binary classification accuracy: 96.67 % [109]
make_moons, provided by scikit-learn library (sklearn)	FNN With Regularization	RBF kernel-Based SVM	Training: 94.7867 % Cross-validation binary classification accuracy: 96 %	Cross-validation binary classification accuracy: 96.67 % [109]
make_moons, provided by scikit-learn library (sklearn)	RBF kernel-Based SVM	RBF kernel-Based SVM	Cross-validation binary classification accuracy: 97%	Cross-validation binary classification accuracy: 96.67 % [109]
Images of human faces	PCA	PCA	Number of PCA components (K) = 335 Retained Variance = 99.0042%	Number of PCA components (K) = 400 Retained Variance = 99% [110]

4.8. Limitations

To summarize the work in the thesis professionally, the three hyperparameter tuning methods presented in this thesis have demonstrated their efficacy in optimizing learning

model’s hyperparameters while significantly reducing computational overhead compared to iterative methods like gradient-based or Bayesian-based approaches. These discussed techniques, being non-iterative, are characterized by their efficiency, requiring minimal memory and shorter execution times.

Furthermore, these methods highlight the advantages of combining traditional tuning techniques to achieve more efficient hyperparameter optimization across various machine learning and deep learning applications. However, it's important to note that the validity of this conclusion is bounded by the number of hyperparameters associated with the learning model, which should not exceed three due to scalability considerations. Specifically, the proposed tuning approaches do not leverage distributed computing and parallelization during execution, so computation time dramatically increases when the number of tuned hyperparameters grows. Additionally, it's worth mentioning that grid search exhibits exponential time complexity as the number of hyperparameters increases. To put it in mathematical terms, if there are 'k' hyperparameters, each with 'n' distinct values, the computational complexity grows exponentially at a rate of O(n^k).

Chapter 5
Conclusions and Recommendations

This thesis applied three competent hyperparameter tuning techniques to tune the hyperparameters of five different machine and deep learning models. In this chapter, we explore three methods for tuning hyperparameters in machine learning and deep learning models. The following hyperparameter tuning techniques were employed: random search-based hyperparameter tuning, hybrid approach combining random and grid search-based hyperparameter tuning, and hybrid approach combining random and manual search-based hyperparameter tuning. These tuning techniques were implemented on five different models: PointNet convolutional neural network, regularized standard feedforward neural network, non-regularized standard, feedforward neural network, support vector machine (SVM), and principal component analysis (PCA).

5.1. Conclusions

The random search-based coarse-to-fine hyperparameter tuning is an enhanced version of the standard random hyperparameter search method. In this approach, the random hyperparameter search is performed in two or three stages instead of a single step, ensuring that the model's performance generalizes well across training, cross-validation, and test datasets. The first two stages, referred to as coarse and fine tuning, were conducted using the cross-validation dataset. The third stage, however, was specifically applied to the PointNet CNN model utilizing the test dataset for further tuning. The hybrid random and grid search combines the effectiveness of random search in determining optimal hyperparameter scales

with the benefits of grid search. By utilizing random search as a guiding mechanism, the grid search was directed towards the regions in the hyperparameter space that have shown promising performance. This approach avoided unnecessary computational time and storage resources spent on tuning hyperparameters with values that are unlikely to significantly improve the accuracy of the learning model. The grid search is particularly accurate in tuning a small number of hyperparameters, typically up to three. In order to leverage the effectiveness of random search in identifying optimal areas within the hyperparameter space, the hybrid random and manual search combines random search with manual search. By using random search as a preliminary step, it provided prior knowledge that helped guide the manual search towards narrower and more effective ranges for the hyperparameters. As a result, computational time and storage resources were utilized more efficiently compared to a standard manual search without any prior knowledge. Without this prior knowledge about recommended hyperparameter ranges, manual search could have been a time-consuming and challenging process. Manual search becomes a necessary option when there is a scarcity of human or computational resources. In this technique, the hyperparameters of the PointNet CNN model were tuned through three stages.

The three tuning techniques were used to improve the image classification accuracy of convolutional deep neural networks such as PointNet. The three hyperparameter tuning techniques tuned three hyperparameters: the mini-batch size of SGD, momentum, and learning rate. Moreover, tuning aimed to prevent the deep-learning model from overfitting the training or validation datasets and improved its generalization to distinct datasets. Therefore, two diverse datasets, pool and test datasets, were used for two independent tests to guarantee that the model has a low variance and can accurately classify the unseen data

model will confront at application time. The sea dataset comes from a different distribution from training and validation datasets.

The results of pool and sea tests demonstrate clear evidence of the distinct generalization and low variance that the model as the tuned model generalized to unseen datasets with high classification accuracy, surpassing the results of related recent research documents. Notably, the model gave an overall average F1 score of 93.6% (F1-score|pool = 96.4%, F1-score|sea = 90.8%) using both the random search and hybrid random and grid search. On the other hand, the hybrid random and manual search approach leveraged the insights gained from random search as prior knowledge to manually tune the hyperparameters within a narrow range. As a result, it reduced the computational time required for hyperparameter tuning. However, this approach yielded a lower classification accuracy compared to both random search and the hybrid random and grid search methods. The model gave an overall average F1-score of 90.97% (F1-score|sea = (F1-score|pool = 95.93%, F1-score|sea = 86.02%).

The random search-based coarse-to-fine tuning and the hybrid random and grid search managed to find a model that made standard feedforward NN conduct the binary classification with high accuracy of 96.67% for the dataset, make_moons, provided by scikit-learn library (sklearn). On the contrary, the hybrid random and manual search approach leveraged the insights gained from random search as prior knowledge to manually tune the hyperparameters within smaller ranges. As a result, it reduced the computational time required for hyperparameter tuning. However, this approach yielded a lower classification accuracy compared to both random search and the hybrid random and grid search methods and achieved a high binary classification accuracy of 96%.

The three hyperparameter tuning techniques successfully identified a model that enabled the SVM model to achieve a high binary classification accuracy of 97% for the dataset, make_moons, provided by scikit-learn library (sklearn). Moreover, these techniques had success at obtaining the minimum number of PCA components (K = 335) by which the dimensionality of the dataset (human faces) was reduced and kept the possession of 99.0042% of the original dataset's variance.

In conclusion, the three hyperparameter tuning techniques discussed in this thesis have succeeded in optimizing the learning model's hyperparameters with less computation cost than iterative approaches (gradient-based, Bayesian-based, etc.). The discussed techniques are non-iterative methods that do not require much memory or long execution time. Moreover, these techniques underscore the effectiveness of combining standard tuning techniques to obtain more efficient hyperparameter optimization for various machine and deep learning applications. The validity of this conclusion is limited by the number of learning model's hyperparameters, which should not exceed three hyperparameters due to scalability concerns. In particular, the proposed tuning methods do not exploit the parallelization feature at execution. In addition, grid search has a time complexity increasing exponentially with the number of hyperparameters, as follows: if there are k hyperparameters, each with n distinct values, the computational complexity increases exponentially at a rate of O(n^k).

5.2. Recommendations and Future Work

It is recommended in future work to replace the early stopping strategy in PointNet used for regularization with a better technique, such as LP-norm regularization. The LP-norm regularization is superior because it follows the orthogonalization concept in deep learning

as it deals with regularization and training as separate tasks; however, it needs the effort to tune the regularization factor as an additional hyperparameter in the model. On the other hand, the early stopping strategy performs regularization and training synchronously but with less classification accuracy. In addition, it is preferred to use an iterative hyperparameter tuning technique like the Bayesian approach, Gradient-based approach, or Population-based training (PBT) to iteratively optimize a significant number of hyperparameters (not only three) until obtaining the desired cross-validation classification accuracy. The state-of-the-art hyperparameter optimization frameworks such as Optuna should be exploited to implement the iterative hyperparameter tuning techniques. Moreover, ASHA algorithm should be utilized to make the full use of parallel programming available at modern multi-core supercomputers to reduce the computation time of the recommended iterative hyperparameter tuning techniques. Besides, a combination of random search or grid search and one of iterative tuning techniques can be implemented to obtain further hyperparameter optimization.

Furthermore, it is worth mentioning that addressing scalability concerns beyond hyperparameter optimization requires a bunch of tools, libraries, or techniques to be considered:

1. **Distributed Computing**: To handle scalability concerns related to training models on large datasets or performing complex data processing tasks, distributed omputing frameworks like Apache Spark or Dask may be needed to use.

2. **Data Preprocessing**: Large datasets often require efficient data preprocessing pipelines. Libraries like Apache Beam or Dask can help in parallelizing and scaling data preprocessing tasks.

3. **GPU/TPU Utilization**: When dealing with deep learning models, using GPUs or TPUs can significantly improve scalability. Libraries like TensorFlow and PyTorch have built-in support for GPU/TPU acceleration.

4. **Feature Engineering**: Scalable feature engineering can be achieved using tools like feature stores or distributed feature engineering libraries.

5. **Model Parallelism and Distributed Training**: For very large models, model parallelism and distributed training techniques can be employed to train models across multiple devices or machines. This can be achieved using frameworks like Horovod or Ray.

References

[1] E. Alpaydin, "Introduction to Machine Learning," United Kingdom, MIT Press, 2014.

[2] C. Sammut and G. I. Webb, "Encyclopedia of Machine Learning," Germany, Springer, 2011.

[3] A. Ng and T. Ma, CS229 Lecture Notes, California: Stanford University, 2012.

[4] A. Singh, N. Thakur and A. Sharma, "A review of supervised machine learning algorithms," in *3rd International Conference on Computing for Sustainable Global Development*, New Delhi, India, 2016.

[5] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu and A. Y. Ng, "Reading Digits in Natural Images with Unsupervised Feature Learning," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[6] N. N. Pise and P. Kulkarni, "A Survey of Semi-Supervised Learning Methods," in *2008 International Conference on Computational Intelligence and Security*, Suzhou, China, 2008.

[7] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction," in IEEE Transactions on Neural Networks," *IEEE,* vol. 9, no. 5, pp. 1054-1054, 1998.

[8] L. Wang, Support Vector Machines: Theory and Applications, Singapore: Springer, 2005.

[9] L. Alzubaidi, J. Zhang, A. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. Fadhel, M. Al-Amidie and L. Farhan, " Review of deep learning: concepts, CNN architectures, challenges, applications, future directions.," *Springer Nature: J Big Data,* vol. 8, 2021.

[10] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press, 2016.

[11] R. Szeliski, Computer vision: algorithms and applications, Springer Science & Business Media, 2010.

[12] K. Asifullah, S. Anabia, Z. Umme and A. S. Qureshi, "A Survey of the Recent Architectures of Deep Convolutional Neural Networks," *Artificial Intelligence Review,* 2020.

[13] K. Simonyan and A. Zisserman, "VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION," 10 April 2015. [Online]. Available: https://arxiv.org/pdf/1409.1556.pdf.

[14] A. Ng, "DLS Course 2: Lecture Notes," DeepLearning.AI, 7 June 2021. [Online]. Available: https://community.deeplearning.ai/uploads/short-url/bvDEh8h6bowXxp1zR52Lq9Yixkb.pdf.

[15] L. Yang and A. Shami, "On Hyperparameter Optimization of Machine Learning: Theory and Practice," *Elsevier,* 2022.

[16] B. Bischl, M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A.-L. Boulesteix, D. Deng and M. Lindauer, "Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. WIREs Data Mining and Knowledge Discovery," Springer, 2023

[17] T. Yu and H. Zhu, "Hyper-Parameter Optimization: A Review of Algorithms and Applications," in arXiv e-prints, 2020.

[18] D. M. Belete and M. D. Huchaiah, "Grid search in hyperparameter optimization of machine learning models for prediction of HIV/AIDS test results," *International Journal of Computers and Applications,* 2021.

[19] D. Navon and A. M. Bronstein, "Random Search Hyper-Parameter Tuning: Expected Improvement Estimation," 17 August 2022. [Online]. Available: https://arxiv.org/pdf/2208.08170v1.pdf.

[20] M. Masum, H. Shahriar, H. Haddad, M. J. H. Faruk, M. Valero, M. A. Khan, M. A. Rahman, M. I. Adnan and A. Cuzzocrea, "Bayesian Hyperparameter Optimization for Deep Neural Network-Based Network Intrusion Detection," in *IEEE International Conference on Big Data* , 2021.

[21] P. Liashchynskyi and P. Liashchynskyi, "Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS," *arXiv preprint arXiv:1912.06059,* 2019.

[22] J. Wong, T. Manderson, M. Abrahamowicz, D. Buckeridge and R. Tamblyn, "Can Hyperparameter Tuning Improve the Performance of a Super Learner?: A Case Study," *Epidemiology,* vol. 4, no. 30, pp. 521-531, 2019.

[23] M. Feurer and F. Hutter, "Hyperparameter optimization. Automated machine learning," Springer, pp. 3-33, 2019.

[24] P. P. Ippolito, "Hyperparameter Tuning In: Egger, R. (eds) Applied Data Science in Tourism. Tourism on the Verge," Springer, 2022

[25] K. Manthey, "Pandas, Caviar, and Deep Learning," 7 8 2018. [Online]. Available: https://www.dell.com/en-us/blog/ai-deep-learning-unstructured-data-isilon/#:~:text=%E2%80%9CCaviar%20refers%20to%20laying%20thousands,autonomous%20driving%20and%20fraud%20detection..

[26] N. Halder, "Harnessing the power of Grid Search for optimized machine learning models," 19 May 2023. [Online]. Available: https://medium.com/analysts-corner/harnessing-the-power-of-grid-search-for-optimized-machine-learning-models-5878e3abf2d6.

[27] "Hyperparameter optimization," 20 May 2023. [Online]. Available: https://en.wikipedia.org/wiki/Hyperparameter_optimization.

[28] C. W. Hsu, C. C. Chang and C. J. Lin, "A practical guide to support vector classification," *Technical Report, National Taiwan University,* 2010.

[29] "CIFAR-10," 2 June 2023. [Online]. Available: https://en.wikipedia.org/wiki/CIFAR-10.

[30] W. Nugraha and A. Sasongko, "Hyperparameter Tuning on Classification Algorithm with Grid Search," *SISTEMASI,* vol. 2, no. 11, pp. 391-401, 2022.

[31] A. Nair, "Improve Your Hyperparameter Tuning Experience With The Random Search," 25 October 2021. [Online]. Available: https://towardsdatascience.com/improve-your-hyperparameter-tuning-experience-with-the-random-search-2c05d789175f.

[32] I. Rojas, G. Joya and A. Catala, "Random Hyper-parameter Search-Based Deep Neural Network for Power Consumption Forecasting. In Advances in Computational Intelligence," *Springer International Publishing AG,* vol. 11506, p. 259–269, 2019.

[33] R. Andonie and A.-C. Florea, "Weighted Random Search for CNN Hyperparameter Optimization," *International Journal of Computers Communications & Control,* vol. 2, no. 15, 2020.

[34] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization," *Journal of Machine Learning Research,* 2012.

[35] R. G. Mantovani, A. L. D. Rossi, J. Vanschoren, B. Bischl and A. C. P. L. F. de Carvalho, "Effectiveness of Random Search in SVM hyper-parameter tuning," in *International Joint Conference on Neural Networks (IJCNN)*, Killarney, Ireland, 2015.

[36] D. Goldberg, "Genetic Algorithms in Search, Optimization and Machinen Learning," *Addison Wesley,* 1989.

[37] The Ray Team, "A Guide to Population Based Training with Tune," 2023. [Online]. Available: https://docs.ray.io/en/latest/tune/examples/pbt_guide.html.

[38] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando and K. Kavukcuoglu, "Population Based Training of Neural Networks," 27 November 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1711.09846.

[39] J. Parker-Holder and A. Kamsetty, "Population Based Bandits: Provably Efficient Online Hyperparameter Optimization," 16 November 2020. [Online]. Available: https://www.anyscale.com/blog/population-based-bandits.

[40] J. Liang, S. Gonzalez, H. Shahrzad and R. Miikkulainen, "Regularized Evolutionary Population-Based Training," 21 July 2021. [Online]. Available: https://doi.org/10.48550/arXiv.2002.04225.

[41] V. Dalibard and M. Jaderberg, "Faster Improvement Rate Population Based Training," 28 September 2021. [Online]. Available: https://doi.org/10.48550/arXiv.2109.13800.

[42] A. Li, A. Spyra, S. Perel, V. Dalibard, M. Jaderberg, C. Gu, D. Budden, T. Harley and P. Gupta, "A Generalized Framework for Population Based Training," 5 February 2019. [Online]. Available: https://arxiv.org/abs/1902.01894v1.

[43] M. R. Hassan, W. N. Ismail, A. Chowdhury, S. Hossain, S. Huda and M. M. Hassan, "A framework of genetic algorithm-based CNN on multi-access edge computing for automated detection of Covid-19," *The Journal of Supercomputing,* 2021.

[44] Y. Bengio, "Gradient-Based Optimization of Hyperparameters," *Neural computation,* vol. 12, pp. 1889-900, 2000.

[45] D. Maclaurin, D. Duvenaud and R. P. Adams, "Gradient-based Hyperparameter Optimization through Reversible Learning," 2 April 2017. [Online]. Available: https://arxiv.org/abs/1502.03492.

[46] O. Y. Bakhteev and V. V. Strijov, "Comprehensive analysis of gradient-based hyperparameter," *Neuro Science ,* 2019.

[47] L. Franceschi, M. Donini, P. Frasconi and M. Pontil, "Forward and Reverse Gradient-Based Hyperparameter Optimization," 12 December 2017. [Online]. Available: https://arxiv.org/pdf/1703.01785v3.pdf.

[48] P. Micaelli and A. Storkey, "Gradient-based Hyperparameter Optimization Over Long Horizons," 2021. [Online]. Available: https://openreview.net/forum?id=6x8tcREIL2W.

[49] W. Koehrsen, "A Conceptual Explanation of Bayesian Hyperparameter Optimization for Machine Learning," *Towards Data Science,* 24 June 2018.

[50] M. Seeger, " Gaussian processes for machine learning," *International Journal for Neural Systems,* pp. 69-106, 2004.

[51] F. Hutter, H. H. Hoos and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," *Proc. LION 5,* pp. 507-523, 2011.

[52] J. Bergstra, R. Bardenet, Y. Bengio and B. K´egl, "Algorithms for hyperparameter optimization," *Proc. Adv. Neural Inf. Process. Syst.,* p. 2546–2554, 2011.

[53] J. Snoek, H. Larochelle and R. P. Adams, "PRACTICAL BAYESIAN OPTIMIZATION OF MACHINE LEARNING Algorithms," 29 August 2012. [Online]. Available: https://arxiv.org/pdf/1206.2944.pdf.

[54] M. Tavallaee, E. Bagheri, W. Lu and A. Ghorbani, "A Detailed Analysis of the KDD CUP 99 Data Set," *IEEE Symposium on Computational Intelligence for Security and Defense Applications,* 2009.

[55] A. NJ, " DLS Course 3: Lecture Notes," DeepLearning.AI, [Online]. Available: https://community.deeplearning.ai/uploads/short-url/k1mj2jravHKqq3avIsMCEbl9iKj.pdf.

[56] L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, M. Hardt, B. Recht and A. Talwalkar. "A System for Massively Parallel Hyperparameter Tuning," in arXiv, 2018.

[57] S. Shekhar et al. "A Comparative study of Hyper-Parameter Optimization Tools," 2021 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE), 2021.

[58] Towards Data Science, "State-of-the-Art Machine Learning Hyperparameter Optimization with Optuna," 31 March 2022. [online]. Available: https://towardsdatascience.com/state-of-the-art-machine-learning-hyperparameter-optimization-with-optuna-a315d8564de1.

[59] T. Akiba, S. Sano, T. Yanase, T. Ohta and M. Koyama, " Optuna: A Next-generation Hyperparameter Optimization Framework," in arXiv, 2019.

[60] A. Singh, N. Thakur and A. Sharma, "Hyperopt: A Python Library for Optimizing the Hyperparameters of Machine Learning Algorithms," in *12th PYTHON IN SCIENCE Conference*, 2013.

[61] A. Singh, N. Thakur and A. Sharma, "Hyperopt: A Python Library for Optimizing the Hyperparameters of Machine Learning Algorithms," in *proceedings of Genetic and Evolutionary Computation Conference*, 2020.

[62] S. Baird, M. Liu and T. Sparks, " High-dimensional Bayesian optimization of 23 hyperparameters over 100 iterations for an attention-based network to predict materials property: A case study on CrabNet using Ax platform and SAASBO," Computational Materials Science, 2022.

[63] P. Bennet, C. Doerr, A. Moreau, J. Rapin, F. Teytaud and O. Teytaud, " Nevergrad: black-box optimization platform," ACM SIGEVOlution, 2021.

[64] M. Martin-Abadal, M. Piñar-Molina, A. Martorell-Torres, G. Oliver-Codina and Y. Gonzalez-Cid, "Underwater Pipe and Valve 3D Recognition Using Deep Learning Segmentation," *Marine Science,* 2021.

[65] R. Charles, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," GitHub, 24 February 2021. [Online]. Available: https://github.com/mmabadal/pointnet.

[66] R. Charles, H. Su, M. Kaichun and L. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation.," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, USA, 2017.

[67] A. Mao, M. Mohri and Y. Zhong, "Cross-Entropy Loss Functions:," 14 April 2023. [Online]. Available: https://arxiv.org/pdf/2304.07288.pdf.

[68] S. Ruder, "An overview of gradient descent optimization," 15 June 2017. [Online]. Available: https://arxiv.org/pdf/1609.04747.pdf.

[69] J. Brownlee, "Stochastic Gradient Descent," in *Better Deep Learning: Train Faster, Reduce Overfitting, and Make Better Performance*, Machine Learning Mastery, 2018, p. 31.

[70] S. Jelassi and Y. Li, "Towards understanding how momentum improves generalization in deep learning," 13 July 2022. [Online]. Available: https://arxiv.org/pdf/2207.05931.pdf.

[71] D. P. Kingma and J. L. Ba, "ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION," 30 January 2017. [Online]. Available: https://arxiv.org/pdf/1412.6980.pdf.

[72] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *The 3rd International Conference for Learning Representations*, San Diego, 2015.

[73] T. Tieleman and G. Hinton, "Lecture 6.5 - RMSProp," COURSERA: Neural Networks for Machine Learning Technical Report, 2012.

[74] I. Sutskever, J. Martens, G. Dahl and G. Hinton, "On the importance of initialization and momentum in deep learning," in *The 30th International Conference on Machine Learning (ICML-13)*, 2013.

[75] K. You, J. Wang, M. Long and M. I. Jordan, "HOW DOES LEARNING RATE DECAY HELP MODERN NEURAL NETWORKS?," 26 September 2019. [Online]. Available: https://arxiv.org/pdf/1908.01878.pdf.

[76] S. Lau, "Learning Rate Schedules and Adaptive Learning Rate Methods for Deep Learning," 29 July 2017. [Online]. Available: https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1.

[77] J. Park, V. Loia, Y. Pan, Y. Sung and H. Lim, "A Study on Dropout Techniques to Reduce Overfitting in Deep Neural Networks.," *Springer,* 2021.

[78] R. JAIN, "Why “early-stopping” works as Regularization?," 8 February 2020. [Online]. Available: https://medium.com/@rahuljain13101999/why-early-stopping-works-as-regularization-b9f0a6c2772.

[79] "Gradient Tape," TensorFlow, [Online]. Available: https://www.tensorflow.org/api_docs.

[80] A. Sherstinsky, "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network," *Elsevier "Physica D: Nonlinear Phenomena" journal,* vol. 404, 2020.

[81] M. S. Rahman, "Computations, optimization and tuning of deep feedforward neural networks," *Department of Neuroscience, Baylor College of Medicine, Houston, TX 77054,* 2019.

[82] A. Apicella, F. Donnarumma, F. Isgrò and R. Prevete, "A survey on modern trainable activation functions," 25 February 2021. [Online]. Available: https://arxiv.org/pdf/2005.00817.pdf.

[83] Towards Data Science, "Derivative of the Sigmoid function," 7 July 2018. [Online]. Available: https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e.

[84] C. F. G. d. Santos and J. P. Papa, "Avoiding Overfitting: A Survey on Regularization Methods for Convolutional Neural Networks," 10 January 2022. [Online]. Available: https://arxiv.org/pdf/2201.03299.pdf.

[85] J. Hoffman, D. A. Roberts and S. Yaida, "Robust Learning with Jacobian Regularization," 2019. [Online]. Available: https://arxiv.org/pdf/1908.02729.pdf.

[86] Y. Gen Li and J. D. Gu, "The Efficacy of L1 Regularization in Two-Layer Neural Networks," 2 October 2020. [Online]. Available: https://arxiv.org/pdf/2010.01048.pdf.

[87] C. Cortes, M. Mohri and A. Rostamizadeh, "L2 Regularization for Learning Kernels," in *The Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI2009)*, 2009.

[88] P.-C. Guo, "A Frobenius norm regularization method for convolutional kernels to avoid unstable gradient problem," 25 July 2019. [Online]. Available: https://arxiv.org/pdf/1907.11235.pdf.

[89] Wikipedia, the free encyclopedia, "Support vector machine," 19 June 2023. [Online]. Available: https://en.wikipedia.org/wiki/Support_vector_machine.

[90] R. G. Mantovani, A. L. D. Rossi, E. Alcobaça, J. Vanschoren and A. C. P. d. L. F. d. Carvalho, "A meta-learning recommender system for hyperparameter tuning: predicting when tuning improves SVM classifiers," 11 June 2019. [Online]. Available: https://arxiv.org/pdf/1906.01684.pdf.

[91] freeCodeCamp, "SVM Machine Learning Algorithm Explained," 24 January 2020. [Online]. Available: https://www.freecodecamp.org/news/support-vector-machines/.

[92] I. T. Jolliffe, Principal Component Analysis, Canterbury, England: Springer, 1986.

[93] M. Wall, A. Rechtsteiner and L. Rocha, "Singular Value Decomposition and Principal Component Analysis," *Springer, Boston, MA. ,* no. https://doi.org/10.1007/0-306-47815-3_5, 2003.

[94] Bits of DNA: Reviews and commentary on computational biology by Lior Pachter, "What is principal component analysis?," 26 May 2014. [Online]. Available: https://liorpachter.wordpress.com/2014/05/26/what-is-principal-component-analysis.

[95] S. Na, L. Xumin and G. Yong, "Research on k-means Clustering Algorithm: An Improved k-means Clustering Algorithm," 2010 Third International Symposium on Intelligent Information Technology and Security Informatics, Jian, China, 2010, pp. 63-67, doi: 10.1109/IITSI.2010.74.

[96] N. Omar, A. Al-zebari and A. Sengur, "Improving the Clustering Performance of the K-Means Algorithm for Non-linear Clusters," 2022 4th International Conference on Advanced Science and Engineering (ICOASE), Zakho, Iraq, 2022, pp. 184-187, doi: 10.1109/ICOASE56293.2022.10075614.

[97] A. Abdelhalim and I. Traore, "A New Method for Learning Decision Trees from Rules," 2009 International Conference on Machine Learning and Applications, Miami, FL, USA, 2009, pp. 693-698, doi: 10.1109/ICMLA.2009.25.

[98] W. Yanxia, "Student Information Management Decision System Based on Decision Tree Classification Algorithm," 2022 IEEE 5th International Conference on Information Systems and Computer Aided Education (ICISCAE), Dalian, China, 2022, pp. 827-831, doi: 10.1109/ICISCAE55891.2022.9927597.

[99] M. MARTIN-ABADAL, "3D RGB pointclouds of underwater pipes and valves," Kaggle, 2021. [Online]. Available: https://www.kaggle.com/dsv/2759939.

[100] R. Hartley and A. Zisserman, Multiple view geometry in computer vision, Cambridge university press, 2003.

[101] 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 {P}ython," *Journal of Machine Learning Research,* vol. 12, pp. 2825--2830, 2011.

[102] A. Ng, "Principal Component Analysis," 25 February 2018. [Online]. Available: https://www.coursera.org/learn/machine-learning-course/lecture/GBFTt/principal-component-analysis-problem-formulation.

[103] J. Wu, X.-Y. Chen, H. Zhang, L.-D. Xiong, H. Lei and S.-H. Deng, "Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimization," *Journal of Electronic Science and Technology,* pp. 26-40, 2019.

[104] B. Kainz, J. Passerat-Palmbach and C. Matache, "Efficient Design of Machine Learning," *Imperial College London,* 2019.

[105] S. Payrosangari, A. Sadeghi, D. Graux and J. Lehmann, "Meta-Hyperband: Hyperparameter Optimization with Meta-Learning and Coarse-to-Fine," *Springer-Verlag,* 2020.

[106] F. A. Khan, A. A. Ibrahim, M. S. Rais, P. Rajpoot, A. Khan and M. N. Akhtar, "Performance Analysis of Supervised Learning Algorithms based on Classification Approach," *IEEE 6th International Conference on Engineering Technologies and Applied Sciences,* 2019 .

[107] E. Guerra, J. Palacin, Z. Wang and A. Grau, "Deep Learning-Based Detection of Pipes in Industrial Environments," *Industrial Robotics - New Paradigms. IntechOpen,* 2020.

[108] A. Krizhevsky, I. Sutskever and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Association for Computing Machinery,* vol. 60, p. 84–90, 2017.

[109] P. Tiwari, S. Dehdashti, A. K. Obeid, M. Melucci and P. Bruza, "Kernel Method based on Non-Linear Coherent State," 15 July 2020. [Online]. Available: https://arxiv.org/pdf/2007.07887.pdf.

[110] L. H. Tran, T. Tran and M. An, “Improved Sparse PCA Method for Face and Image Recognition,” arXiv.org, 2021. https://doi.org/10.48550/arxiv.2112.00207.