Algerian Democratic and Popular Republic
Ministry of Higher Education and Scientific Research

KASDI MERBAH UNIVERSITY OUARGLA
Faculty of New Information and Communication Technologies
Department of Computer Science and Information Technology

Thesis Submitted in Candidacy for a Master Degree in computer science, option

Artificial Intelligence and Data Science

by Basma Dokkar & Bouthayna Meddour

Theme

First order Optimization Methods for Deep Learning.

Evaluation date: 18/06/2023
Jury Members:

Dr.	Oussama AIADI	Jury Chair	UKM Ouargla
Dr.	Khadra BOUANANE	Supervisor	UKM Ouargla
Prof.	Mohamed Lamine KHERFI	Co-supervisor	ENSIA Algiers
Dr.	Abdelhakim CHERIET	Examiner	UKM Ouargla

Academic year: 2022/2023

Acknowledgments

First of all, all praise and thanks to Allah Almighty who helped us and gave us the patience and courage to achieve this work.
We would like to acknowledge and give our warmest thanks to our supervisor "Dr. Khadra BOUANANE", who made this work possible. Her guidance, advice, and support carried us through all the stages to accomplish our thesis.
Our thanks to "Prof. KhERFI Mohammed Lamine" for his supervision and for trusting us.
Our thanks to "Dr. Oussama AIADI" and "Dr. Abdelhakim CHERIET" for accepting to review and evaluate this work.
We thank our parents and siblings who have supported us during this academic journey and our entire lives.

Dedication

Words can never express my deep love and gratitude to the people who have supported me throughout my life.
I dedicate this work to my dear mother Fatima and my dear father Salah.
To my dear siblings Lamine with his wife Aicha and son Ghaith, Nasma, Maroua, Rahim, Nasim, and Rahaf.
To my best friend Maria who never let me down.
To my teacher Darifa you'll always be remembered.
To my dear supervisor DR. Bouanane and my dear partner Basma.
Thanks for making me see this adventure through to the end.

Bouthayna Meddour

I dedicate this work to the pure soul of my grandmother Fatma.
To the most supportive family who I can never thank, my parents Nabila and Mohammed, my siblings Issam, Soumeya, Younes and Soror and baby Clarence.
To the real people who stood by my side, Firdaous, Souheil, Hadjer and Atidel.
Last but not least, my teachers, my supervisor DR. Bouanane and my partner Bouthayna.
Thank you all for making the effort I put into this work worth it.

Basma Dokkar

Abstract

Deep learning has emerged as a transformative technology in various domains, ranging from computer vision to natural language processing. The success of deep learning models heavily relies on effective optimization algorithms. In this thesis, two main contributions are presented. In Contribution 1, which is a two-fold comparative study, we first explore the impact of various first-order optimization techniques on the learning process of U-Net for the task of Change Detection. Namely, Gradient descent with Momentum (Momentum GD), Nesterov Accelerated Gradient (NAG), Adaptive Gradient (AdaGrad), Root Mean Square Propagation optimizer (RMSProp), and the adaptive moment estimation optimizer (Adam). The results show that RMSProp, NAG, and AdaGrad reached the highest validation accuracies: 0.976, 0.978, and 0.979 with 10^{-2}, 10^{-3}, and 10^{-4} respectively, while Adam was the fastest to converge and scored the lowest validation loss. Moreover, Adam scored the highest precision and F1 score across all learning rate values with 0.491 and 0.376 respectively. Nevertheless, we noticed that Adam's performance could be significantly influenced by the data sparsity. In light of this hypothesis, the second part of Contribution 1 investigates the impact of sparsity on the performance of Adam optimizer. We compare different sparsity-level models, U-Net, DenseU-Net, and DenseNet using Adam optimizer for BCE and focal Tversky losses, on dense and sparse datasets for three ML tasks: Change detection, image segmentation, and object recognition. According to the obtained results, the Adam optimizer seems to be more sensitive to the model than the data sparsity. In Contribution 2, we propose a new method that aims to improve Adam's performance. In this approach, we combine a simulated annealing strategy with a dynamic learning rate to overcome the generalization gap which characterizes adaptive methods. We assess the several variants of the proposed approach compared to Adam, stochastic Gradient Descent, and Adabound. For this purpose, a simple 3-layer CNN is trained on two datasets MNIST and CIFAR-10.

Key words: Deep learning, optimization, first order optimization, Adam, CNN, U-Net.

ملخص

ظهرت التعلم العميق كتكنولوجيا محوّلة في مجالات مختلفة، بدءًا من رؤية الحاسوب وحتى معالجة اللغة الطبيعية. يعتمد نجاح نماذج التعلم العميق بشكل كبير على خوارزميات التحسين الفعّالة. في هذه الرسالة، يتم تقديم مساهمتين رئيسيتين. في المساهمة الأولى، وهي دراسة مقارنة مزدوجة، نستكشف أثر تقنيات تحسين الدرجة الأولى المختلفة على عملية التعلم لنموذج U-Net في مهمة اكتشاف التغييرات. على وجه الخصوص، نستخدم الانحدار التدرجي مع الزخم (Momentum GD) ، التسارع النستيروفي للانحدار التدرجي (NAG) ، التحسين التدرجي المتكيف (AdaGrad) ، محسن انتشار جذر المتوسط المربع (RMSProp) ، ومحسن تقدير الزخم المتكيف (Adam) . تظهر النتائج أن RMSProp و NAG و AdaGrad قد حققت أعلى دقة التحقق: 0.976 و 0.978 و 0.979 على التوالي مع 10^{-2} و 10^{-3} و 10^{-4} على التوالي، في حين كان ادم أسرع في التوصل إلى التقارب وحصل على أدنى فقدان التحقق. وعلاوة على ذلك، حقق Adam أعلى دقة ومقياس F1 عبر جميع قيم معدل التعلم بقي 0.491 و 0.376 على التوالي. ومع ذلك، لاحظنا أن أداء ادم قد يتأثر بشكل كبير بندرة البيانات. وفي ضوء هذا الافتراض، تستكشف الجزء الثاني من المساهمة الأولى أثر الندرة على أداء

محسن ادم. نقارن بين نماذج ذات مستويات ندرة مختلفة، او ت و ضنسـ ت و ضنس ت باستخدام محسّن Adam لخسائر BCE و focal Tversky ، على مجموعات.

الكلمات المفتاحية: التعلم العميق ، التحسين من الدرجة الاولى، Adam, CNN, U-Net.

Résumé

L'apprentissage profond a émergé en tant que technologie transformative dans divers domaines, allant de la vision par ordinateur au traitement du langage naturel. Le succès des modèles d'apprentissage profond repose largement sur des algorithmes d'optimisation efficaces. Dans cette thèse, deux contributions principales sont présentées. Dans la Contribution 1, qui est une étude comparative à double volet, nous explorons d'abord l'impact de différentes techniques d'optimisation du premier ordre sur le processus d'apprentissage de l'U-Net pour la tâche de détection de changements. Plus précisément, la descente de gradient avec moment (Momentum GD), le gradient accéléré de Nesterov (NAG), le gradient adaptatif (AdaGrad), l'optimiseur de propagation de la racine carrée de la moyenne quadratique (RMSProp) et l'optimiseur d'estimation du moment adaptatif (Adam) ont été étudiés. Les résultats montrent que RMSProp, NAG et AdaGrad ont atteint les précisions de validation les plus élevées : respectivement 0,976, 0,978 et 0,979 avec des taux d'apprentissage de 10^{-2}, 10^{-3} et 10^{-4}, tandis qu'Adam a été le plus rapide à converger et a obtenu la perte de validation la plus faible. De plus, Adam a obtenu la précision et le score F1 les plus élevés pour toutes les valeurs du taux d'apprentissage, avec respectivement 0,491 et 0,376.

Cependant, nous avons remarqué que les performances d'Adam peuvent être significativement influencées par la raréfaction des données. En tenant compte de cette hypothèse, la deuxième partie de la Contribution 1 examine l'impact de la raréfaction sur les performances de l'optimiseur Adam. Nous comparons différents modèles de niveau de raréfaction, U-Net, DenseU-Net et DenseNet, en utilisant l'optimiseur Adam pour les pertes BCE et focal Tversky, sur des ensembles de données denses et rares pour trois tâches d'apprentissage automatique : la détection de changements, la segmentation d'images et la reconnaissance d'objets. Selon les résultats obtenus, l'optimiseur Adam semble être plus sensible au modèle qu'à la raréfaction des données.

Dans la Contribution 2, nous proposons une nouvelle méthode visant à améliorer les performances d'Adam. Dans cette approche, nous combinons une stratégie de recuit simulé avec un taux d'apprentissage dynamique pour surmonter l'écart de généralisation qui caractérise les méthodes adaptatives. Nous évaluons plusieurs variantes de l'approche proposée par rapport à Adam, à la descente de gradient stochastique et à Adabound. Dans cette optique, un simple réseau de neurones à trois couches est entraîné sur deux ensembles de données, MNIST et CIFAR-10.

Mots-clés: l'apprentissage en profondeur , optimisation, Optimisation du premier ordre, Adam, CNN, U-Net.

Contents

List of Figures

General Introduction

1 Introduction

Deep Learning(DL) is a powerful tool of Machine Learning(ML) that particularly reached unmatched performance for various tasks. However, there is a high cost associated with obtaining such a performance. A current challenge for all DL practitioners is developing efficient models that aim to reach an optimal trade-off between the performance of the model and its cost [41].

A DL model consists of three major components: The data set, the model's architecture, and the training process. The latter is the dynamic component through which the final model learns from the data set to achieve a predefined task.

Optimization is an essential phase in the training process. It consists in minimizing the cost function in order to determine the optimal set of parameters for the corresponding DL model. However, optimization in turn is very time and material-consuming.

Different optimizers were established to address the issues that arise during training due to the characteristics of data and/or models. These methods are typically categorized into adaptive and non-adaptive.

Starting with Stochastic Gradient Descent(SGD) [54] and Momentum SGD [49], non-adaptive optimizers update all the parameters of the model with equal step sizes. Conjugate gradient [59] and Nesterov Accelerated gradient(NAG) [45] also fall under this category. Nevertheless, the aforementioned algorithms show a remarkable delay throughout the training.

To accelerate the process, adaptive methods as the name suggests, refer to the methods where the learning rate is adapted to each parameter. Adaptive Moment Estimation(ADAM) [30], Adaptive Gradient(AdaGrad) [17], and Root Mean Squared Propagation(RMSProp) [62] among various other methods and variants which employed adaptivity.
Although the second family of optimizers succeeded to gain in speed, there is an impactful generalization gap between these and their non-adaptive counterparts [67]. Several works proposed empirical and theoretical analysis to explore the phenomenon. Zhang et al. [69] linked this issue to direction missing which occurs due to detaching individual weights from the network. Zhou et al. [70] confirmed that ADAM-like optimizers suffer from heavy-tailed gradient noise. Conversely, Gupta et al. [25] observed that for classification problem type, networks tend to large values when trained with adaptive methods causing loss flattening.
An ongoing research trend to enhance the performance of these fast yet overfitted methods resulted in a number of new methods. AMSGrad [50] and AdaBound [39] addressed the problem from a heavy-tailed noise perspective, Logit Attenuating Weight Normalization(LAWN) [25] proposed a fix for loss flattening and regarding the direction missing, Normalized Direction Preserving ADAM(ND-ADAM) [69] was developed to resolve the direction missing by normalizing hidden layers' weights.
With that being said, as far as we know, no adaptive methods considered exploring solutions and preventing extreme learning rate values (heavy-tailed noise) within a single framework. In this thesis, we first conduct a two-fold comparative study investigating the performance of common adaptive methods and then present a set of optimizers. Our contributions can be summarized as follows:

- The first comparative study comprises five major adaptive and nonadaptive optimization techniques on the specific case of U-Net [55] for Change Detection(CD) task.
- The second, consists of a deeper investigation of the behavior of ADAM optimizer on different models and architectures with an eye toward the effect of sparsity.
- Finally, we develop a set of optimization methods based on ADAM and Simulated Annealing(SA) strategy [31].

2 THESIS STRUCTURE

This thesis is organized as follows:

In Chapter 1, we start by introducing machine learning and its main paradigms: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning with a focus on some tasks that we used in our work: change detection, image segmentation, and image recognition. After that, we present the three different components of the ML framework: data set, model, and learning process, presenting the required concepts needed for this work.

In Chapter 2, we introduce optimization in ML and DL, highlighting the crucial role it plays in the enhancement of ML and DL models. In this context, we focus on first-order optimization and the most common gradient-based optimization methods.

Chapter 3 is dedicated to our first contribution when we first conduct a comparative study of five common gradient-based optimization techniques to highlight the effect of the optimizer on the performance and effectiveness of a deep learning method. Then, we investigate through another comparative analysis the effect of the sparsity on the perfomance of ADAM optimizer considering different ML tasks.

Our second contribution is presented in Chapter 4. In this contribution, we propose a set of optimizers that make use of the simulated annealing strategy. The main purpose we aim to address is the generalization issue that characterizes adaptive methods. Experiments are conducted using two of the popular ML datasets: MNIST and CIFAR-10.

CHAPTER 1

Introduction to Machine Learning and Deep Learning

1 Introduction

Machine learning and deep learning have made significant strides in recent years, revolutionizing a variety of fields and industries. The training and performance enhancement of deep learning and machine learning models both heavily rely on optimization. In this chapter, We will introduce the field of ML and its paradigms with a focus on some tasks of ML and then we present ml framework components: data, model, and process learning specifying some models that we used in our work. Finally, we will address the importance of optimization in the learning process.

2 Machine learning

2.1 Definition

In 1959, Arthur Samuel defined machine learning as " *a field of study that gives computers the ability to learn without being explicitly programmed*" [58], a subset of artificial intelligence (AI) that focuses on developing algorithms and models that can analyze and interpret data, identify patterns, and make predictions or decisions based on that analysis. With machine learning, computers can learn from past experiences and improve their performance over time, making them valuable tools for many applications.

2.2 Machine Learning Paradigms

ML can be classified into four major types: Supervised Learning (SL), Unsupervised Learning (UL), Semi-supervised Learning, and Reinforcement Learning (RL).

2.2.1 Supervised Learning

Supervised Learning (SL) is one of the most popular and effective paradigms in machine learning. This setting entails training a model using labeled data, enabling it to acquire knowledge from this information and then, utilize this knowledge to make accurate predictions for new, unseen data.

Mathematically, we have a training dataset represented as a set of pairs (x, y), where $x \in \mathbb{R}^d$ is the input or features vector, and y is the output. The goal is to find the best mapping function f such that $y = f(x)$ that maps the inputs to the correct outputs using machine learning algorithms [38].

Depending on the output type, two main predictive tasks can be identified: classification and regression. In classification, the algorithm predicts a categorical label for new data based on the labeled data it was trained on, whereas in regression, the algorithm predicts a real value for new data based on patterns learned from labeled data.

2.2.2 Unsupervised Learning

Unsupervised Learning (UL) is another type of machine learning where, unlike supervised learning, the algorithm learns from unlabeled data, UL algorithms are used to identify patterns and relationships in data without prior knowledge of the outcome, which means the data does not come with a specified target but only as an input of features $x \in \mathbb{R}^d$. The goal is to learn a function f describing the unknown process $P(x)$ from which the examples were derived in some way [38]. This makes UL useful for tasks such as clustering, Visualization, and dimensionality reduction.

2.2.3 SEMI-SUPERVISED LEARNING(SSL)

Semi-supervised learning is a branch of machine learning that involves using both labeled and unlabeled data to accomplish learning tasks. It sits between supervised and unsupervised learning methods, allowing the utilization of large amounts of unlabeled data alongside smaller sets of labeled data, which is often the case in real-world scenarios. The model can thus learn more about the underlying structure of the data and potentially improve its generalization ability by using unlabeled data.

Formally and in addition to the labeled dataset $D_L = \{((x_i, y_i)), i = 1, \ldots, l\}$, we make use of unlabeled dataset $D_U = \{x_i, i = l + 1, \ldots, n\}$ for the learning task [64].

2.2.4 REINFORCEMENT LEARNING(RL)

Unlike supervised learning, RL does not rely on labeled examples but instead learns through trial-and-error interactions with the environment. This type of learning has three primary components: the agent, the environment, and the actions. where an agent learns to make decisions by exploring and interacting with an environment to maximize cumulative rewards. The primary objective is for the agent to select actions that yield the highest expected rewards within a specified time frame. RL has been successfully applied in various fields, including robotics, gaming, and autonomous vehicles.

Figure 1.1 provides an illustration of the four paradigms aforementioned.

2.3 MACHINE LEARNING TASKS

ML is applied to a wide range of tasks in various domains like Healthcare[66], Recommender Systems[48], Financial Analysis[13], Natural Language Processing (NLP)[44], etc. In our work, we focus on three tasks and will address each of them: change detection, image segmentation, and object recognition.

2.3.1 CHANGE DETECTION TASK

Change detection is a task in which machine learning algorithms are employed to identify and analyze alterations or differences in data over time. In other words, change detection is

[1] https://www.enjoyalgorithms.com/blogs/supervised-unsupervised-and-semisupervised-learning

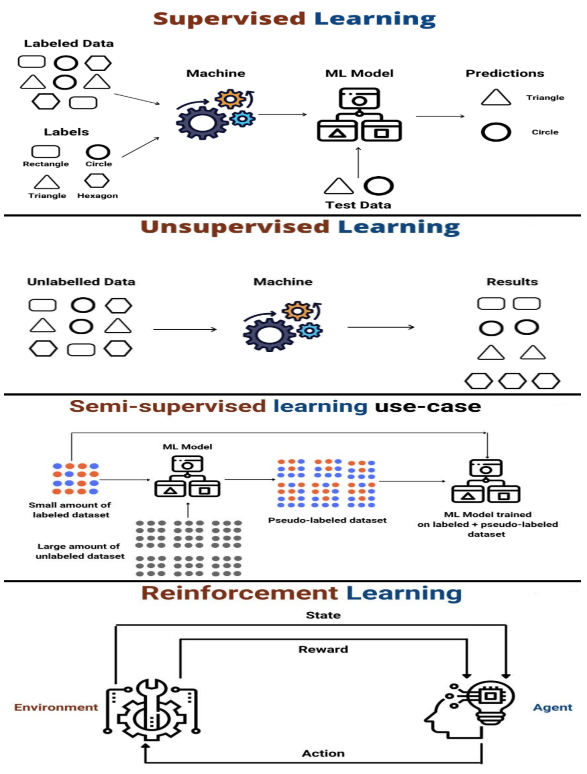


Figure 1.1: Main Machine learning paradigms.[1]

the process of identifying differences in the state of an object or phenomenon by observing it at different times[60]. The process requires multi-temporal images and certain criteria specified by the mask. A mask is a binary image where black regions correspond to areas with negligible change while white regions represent significant changes[53]. It is a crucial step for analyzing temporal Earth observation sequences in order to build evolution maps of land cover, urban expansion, deforestation, etc. Figure 1.2 is a sample from the Onera Satellite Change Detection dataset where the change mask corresponds to the bi-temporal images. Therefore, Change Detection is reduced to binary image segmentation.

Difference image techniques are suitable for medium-resolution images like those of the dataset used in our experiments, based on the observation that the smaller the resolution, the less spatial-contextual information affects the resulting CD map[46] since fewer pixels represent an object. DL-based approaches are summarized under three categories: feature-based, patch-based, and image-based[46]. The method we use falls under patch-based DL methods, where different images are calculated and then divided into smaller patches of

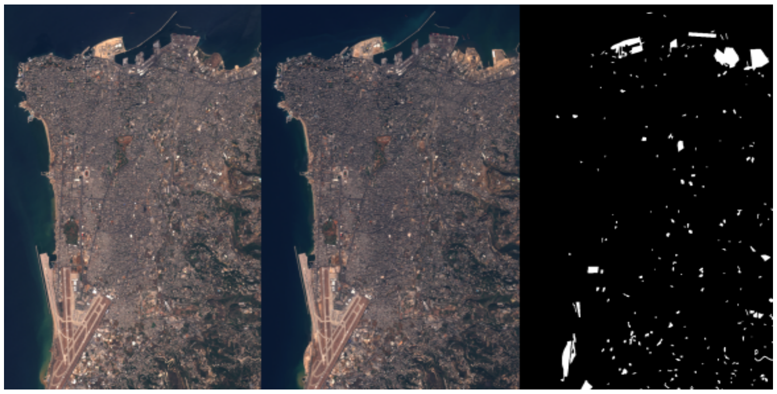


Figure 1.2: Beirut city. Sample from Onera Satellite Change Detection dataset created by Daudt et al[12].

256×256 due to the large size of the images.

2.3.2 IMAGE SEGMENTATION TASK

Image segmentation is the task of finding groups of pixels that “go together”. In statistics, this problem is known as *cluster analysis* and is a widely studied area with hundreds of different algorithms[28]. Image segmentation is a computer vision task that involves dividing an image into multiple regions or segments to identify and extract meaningful objects or areas. The goal is to partition the image into different homogeneous regions based on properties such as color, texture, or intensity. Segmentation plays a central role in a broad range of applications including medical image analysis (e.g., tumor boundary extraction and measurement of tissue volumes), autonomous vehicles (e.g., navigable surface and pedestrian detection), video surveillance, and augmented reality[21]. It enables the extraction of meaningful information from images, allowing for further analysis, interpretation, and decision-making. Numerous image segmentation algorithms have been developed[42], such as thresholding, edge-based methods, region-based approaches, and deep learning-based models, and are employed to achieve accurate and efficient image segmentation results, depending on the specific requirements and characteristics of the task at hand. Figure 1.3 show an example of segmentation results.

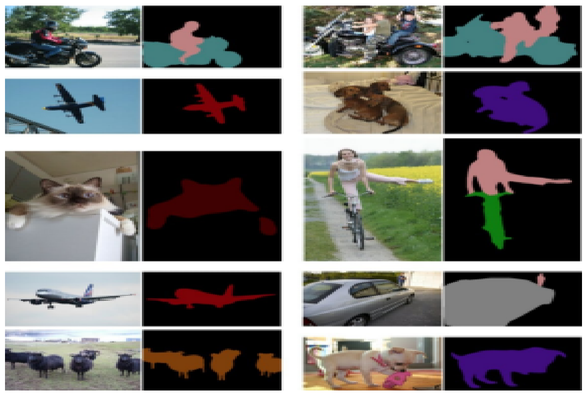


Figure 1.3: example of segmentation results[9]

2.3.3 OBJECT RECOGNITION TASK

Object recognition is a core task of computer vision, which helps detect and analyze images to enable the automation of a specific task. It is a technology that is capable of identifying places, people, objects, and many other types of elements within an image or video frame and drawing conclusions from them by analyzing them. This involves two main steps: object detection, where potential object regions are identified using techniques like convolutional neural networks (CNNs)[34] or region proposal methods[22], and object classification, where labels or categories are assigned to the detected objects using machine learning algorithms. Deep learning models like AlexNet, VGGNet, GoogLeNet, ResNet, MobileNet, YOLO[51], and Faster R-CNN[52] have significantly advanced object recognition performance, with applications in areas such as autonomous vehicles, surveillance, and augmented reality. These models have been developed and refined through research efforts in computer vision, leveraging large-scale annotated datasets like ImageNet for training and evaluation. igure 1.4 represents objects classified through image recognition.

3 MACHINE LEARNING FRAMEWORK'S COMPONENTS

A machine learning (ML) framework typically involves several components that aim to facilitate the development, training, evaluation, and deployment of machine learning mod-

[2]https://www.techtarget.com/searchenterpriseai/definition/image-recognition

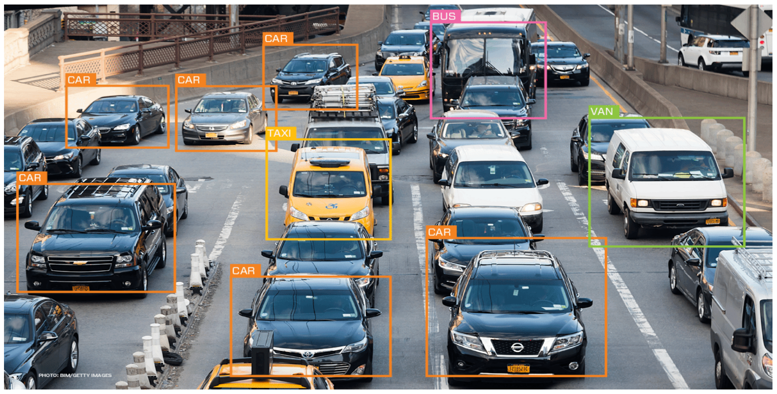


Figure 1.4: Objects are classified through image recognition.[2]

els [23]. We will specify three components very important: Dataset, Model, and Learning process.

3.1 DATASET

The dataset is a fundamental component of any machine learning framework. It consists of a collection of labeled or unlabeled examples that are used to train, validate, and test machine learning models. The dataset serves as the input for the learning process and contains features and corresponding labels for each example. The dataset can come in two forms: sparse or dense depending on the specific problem, data collection methods, and the characteristics of the dataset itself.

Sparse dataset: Sparse data refers to a dataset or data representation where a large portion of the entries or features is missing or contain zero values, resulting in a low density of available data and a high degree of sparsity [24].

Dense datasets: Dense data refers to a dataset or data representation where a significant majority of the entries or features contain non-zero values, resulting in a high density of available data and a low degree of sparsity [4]

The choice of dataset depends on the specific problem being tackled as well as the available processing and storage resources where the quality, size, and diversity of the dataset greatly influence the performance and generalization ability of the trained models.

3.2 Model

The model is a central component of the machine-learning framework and refers to a mathematical representation or algorithm that learns patterns and relationships and makes predictions or decisions based on input data. The choice of the best function or model depends on several factors, including the nature and characteristics of the data and the type of problem that you want to solve. Some common machine-learning models are Support Vector Machines (SVM), K-Nearest Neighbor (KNN), K-means, Principal Component Analysis (PCA), Artificial Neural Networks (ANN), etc. A model can be trained with respect to a selected paradigm to perform a specific task.
For the purpose of this work, we make focus on ANN that are explicitly detailed in the following subsection.

3.2.1 Artificial Neural Network (ANN)

An artificial neural network (ANN) is a computational model inspired by the structure and function of biological neural networks, such as those in the human brain. ANNs consist of interconnected artificial neurons organized into layers. There are three primary types of layers in an ANN: the input layer, the hidden layer(s), and the output layer. The input layer receives the external data and passes it on to the hidden layers, which are the intermediate layers that separate the other layers and perform the computations using the feedforward algorithm. The output layer produces the final result or prediction based on the processed data, and then the ANN uses the backpropagation algorithm to adjust the weights of the connections between neurons depending on the error rate between the target and the actual output. Figure 1.5 show the general structure of ANN.

3.2.2 Deep Neural Networks (DNNs)

Deep neural networks (DNNs) are artificial neural networks (ANNs) with multiple hidden layers between the input and output layers. It leverages the power of depth to learn hierarchical representations of data, enabling it to capture complex patterns and relationships [23]. Each layer in a DNN consists of multiple artificial neurons that compute weighted sums of inputs and apply activation functions to produce outputs. This process

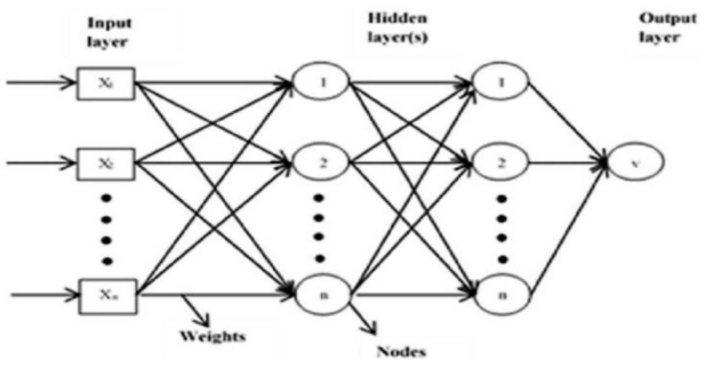


Figure 1.5: The general structure of ANN [57]

is known as feedforward. The DNN's training process involves iteratively adjusting the weights and biases to minimize the difference between the predicted outputs and the actual outputs using techniques like backpropagation This allows the DNN to learn complex representations and make accurate predictions or classifications. Figure 1.6 represent an example of the structure of DNN with three hidden layers.

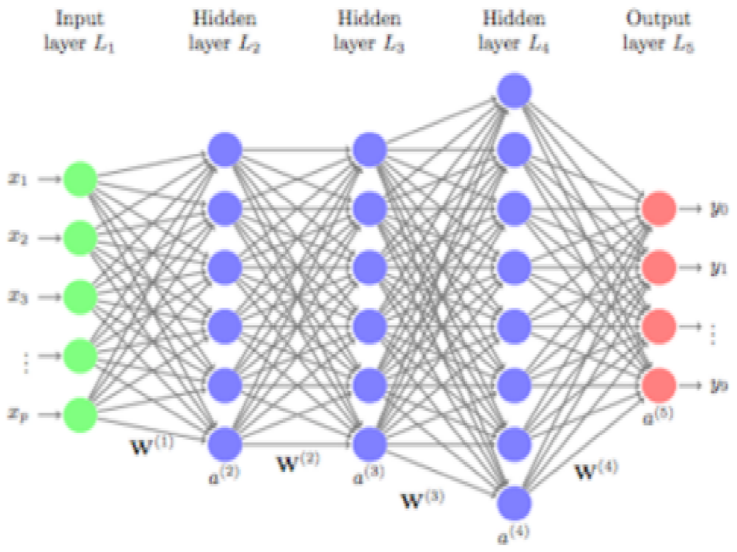


Figure 1.6: Structural representation of a Deep Neural Network with three hidden Layers [63]

3.2.3 CONVOLUTIONAL NEURAL NETWORKS (CNN)

CNN [34]is a type of deep artificial neural network commonly used in image recognition and processing. It is designed to automatically and efficiently learn spatial hierarchies of features from input images. CNNs are composed of multiple layers, including convolutional, pooling, and fully connected layers, which work together to extract and classify visual features in an image.

Convolutional layer: These layers apply a set of learnable filters (also known as kernels) to the input image. Each filter convolves across the input image, computing dot products between the filter weights and the pixel values in the receptive field. The output of each filter is a feature map that highlights the presence of specific visual patterns or features.

pooling layer: The pooling layer, also known as downsampling, is typically inserted after one or more convolutional layers. Its main purpose is to reduce the spatial dimensions (width and height) of the input feature maps while retaining the most important information. The most commonly used pooling method is max pooling, where the filter moves (often 2x2 or 3x3 in size) over the input feature map and selects the pixel with the maximum value. This selected value is used to form the output feature map.

Fully connected layer: These layers are typically located towards the end of the network. The main purpose of fully connected layers is to learn complex relationships and make predictions based on the features extracted by earlier layers in the network. The output of the previous layers is flattened into a one-dimensional vector and connected to a set of neurons that produce the final prediction. Figure 1.7 represent the CNN architecture.

3.2.4 U-NET

U-Net[55] is an encoder-decoder architecture. The encoder part consists of several blocks. Each block contains two successive convolutions of 3x3, doubling the number of channels in the feature map, a ReLU unit, and max pooling. In the decoder part, each block consists of an upsampling, a 2x2 up-convolution that halves the number of feature channels. Concatenation with the correspondingly cropped feature map from the contracting path and

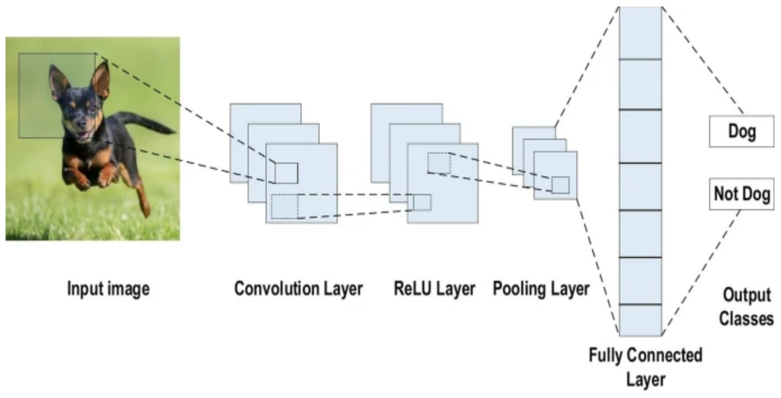


Figure 1.7: The CNN architecture [2]

two 3x3 convolutions, each followed by a 3x3 convolution. Figure 1.8 illustrates a UNet architecture.

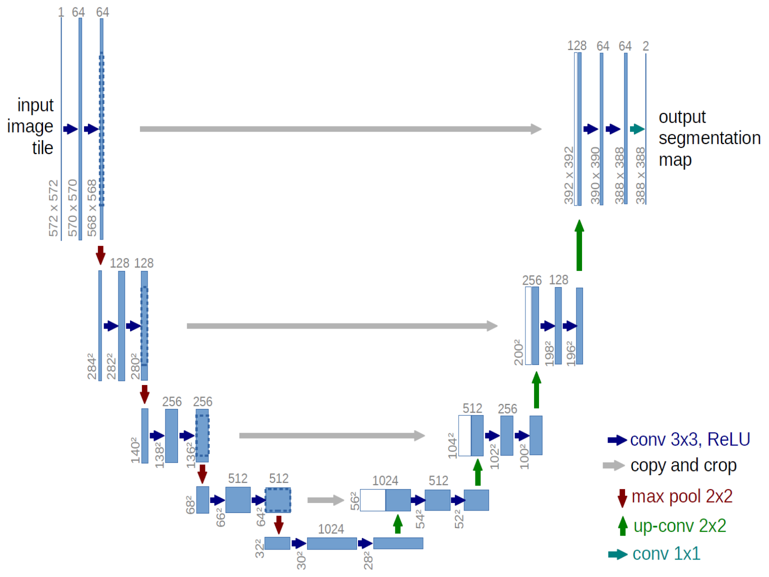


Figure 1.8: UNet original architecture where blue boxes represent feature maps, white boxes copied feature maps with numbers of channels on top. Different operations are denoted with arrows of different colors [55]

3.2.5 DENSEUNET

DenseUNet[8], also known as Dense U-Net, is a deep learning architecture that combines the concepts of3.2.4U-Net and DenseNet[27], two well-known neural network architectures. It was specifically designed to address the challenges of semantic segmentation, which involves pixel-level classification and labeling of images. DenseUNet retains the encoder-decoder structure of U-Net while incorporating dense connections from DenseNet. These dense connections enable efficient information flow and enhanced feature reuse throughout the network. By leveraging multi-scale features and incorporating skip connections, DenseUNet captures both local and global contextual information, making it effective for accurate and robust semantic segmentation. DenseUNet has shown promising results in various medical image segmentation tasks and continues to be an active area of research.

3.2.6 DenseNet

The Dense Convolutional Network (DenseNet) [27] is a deep-learning architecture for computer vision tasks. It is known for its dense connectivity pattern, where each layer receives direct inputs from all preceding layers, facilitating information flow and promoting feature reuse. This connectivity pattern addresses the vanishing gradient problem and enhances model expressiveness.
DenseNet consists of dense blocks which are the building blocks of the network, and transition layers, which control the spatial dimensions and the number of feature maps. Within a dense block, each layer is connected to all previous layers, while transition layers are responsible for downsampling the spatial dimensions and reducing the number of feature maps before passing them to the next dense block, and its effectiveness has been demonstrated through state-of-the-art performance on various computer vision tasks. Figure 1.9 illustrates DenseNet architecture.

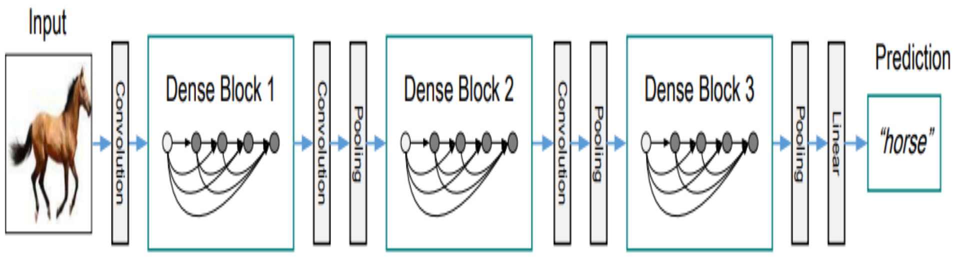


Figure 1.9: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.[27]

3.3 LEARNING PROCESS

The learning process is a crucial component of machine learning frameworks. It refers to the phase where a model is trained on a dataset to learn patterns, relationships, and rules that can be used for making predictions or decisions on new, unseen data after the selection of appropriate algorithms, the preparation and cleaning of data, the tuning of hyperparameters, and the evaluation of model performance. The learning process can be performed according to supervised, unsupervised, semi-supervised, or any other learning paradigm, depending on the type of data availability and the desired outcome.

In the supervised setting, and during the learning process, the loss function plays a crucial role by quantifying the model's error or mismatch between the predicted output and the true target values. The goal is to minimize this error by iteratively adjusting the model's parameters through an optimization algorithm. By minimizing the loss function, the model aims to improve its performance and ability to make accurate predictions. There are several functions that are used to measure the error loss for different types of machine learning problems. We can mention

Root Mean Squared Error (RMSE): Which calculates the square root of the average of squared differences between the predicted and true values. Its formula is given by

$$RMSE(h) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(h(x_i) - y_i)^2} \tag{1.1}$$

Where h represents a hypothesis or prediction function, $x_i, y_i,$ for $i = 1, \dots, N$ represent the input and the corresponding true or target value for the ith data point, respectively.

Binary Cross-Entropy Loss: Which is used when the target variable has two classes. It measures the dissimilarity between the predicted probabilities and the true binary labels. Its formula is given as follows

$$\text{Binary Cross-Entropy Loss } = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)] \tag{1.2}$$

Where y_i is the true label (either 0 or 1) for the ith sample, and p_i represents the predicted probability for the ith sample to belong to Class 1.

Categorical Cross-Entropy Loss: Suitable for multiclass classification problems. It measures the dissimilarity between the predicted class probabilities and the true class labels.

$$\text{Categorical Cross-Entropy Loss } = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(p_{ij}) \tag{1.3}$$

Where C represents the total number of classes, y_{ij} represents the true label (0 or 1) for the i sample belonging to class j, and p_{ij} represents the predicted probability for the i sample belonging to class j.

Focal Tversky Loss: Focal Tversky Loss (FTL)[1] is used in image segmentation tasks where ground truth annotations are available for training. It is used as the objective function during the training process to guiding the model to produce accurate segmentation. The FTL combines elements from the Tversky index and the focal loss defined as:

$$FocalTverskyloss = -\log(T)\cdot(1-T)^{\gamma} \tag{1.4}$$

Where γ is a hyperparameter that controls the degree of focusing, and T is The Tversky index defined as:

$$T = \frac{TP}{TP + \alpha\cdot FP + \beta\cdot FN} \tag{1.5}$$

Where TP represents the number of true positive pixels, FP represents the number of false positive pixels, and FN represents the number of false negative pixels. The parameters α and β control the relative weighting of false positives and false negatives, respectively.

3.3.1 THE LEARNING PROCESS OF DEEP NEURAL NETWORKS

The learning process in a deep neural network is a crucial step too. The latter uses two key steps for training the model, namely, feedforward and backpropagation.

FeedForward In the feedforward, the input data is fed into the neural network, and the network computes the output predictions where each input is multiplied by a random weight. Then the result which is the product of each input and its weight is added to a value called bias. Finally, an activation function is applied to the result Then, the output of each neuron becomes the input of the neurons into the next layer, and the same process is repeated until the final layer [23].

Backpropagation After the feedforward comes the step of backpropagation. The goal is to minimize the error or loss of the neural network by adjusting its weights. The algorithm works by propagating the error from the output layer back to the input layer, hence the name "backpropagation." It utilizes the chain rule of calculus to compute the partial derivatives of the loss function with respect to the weights of the network [23].

3.3.2 THE IMPORTANCE OF OPTIMIZATION IN THE LEARNING PROCESS

Optimization plays a critical role in the learning process of deep learning, specifically in training deep neural networks. Deep neural networks often have a large number of parameters, making the optimization task complex. Efficient optimization algorithms are crucial for minimizing the loss function and updating the network's parameters iteratively. They help in navigating the high-dimensional parameter space, avoiding local minima, and finding optimal solutions. Optimization techniques also contribute to regularization, generalization, scalability, and computational efficiency [23].

4 CONCLUSION

In this chapter, we have presented some concepts from machine learning and Deep learning that are mainly related to our work. We started by introducing the field of ML and its paradigms with a focus on some tasks that we used in our work, then we got onto the ML framework components: dataset, model, and the learning process with specifying some models. Finally, we focused on the importance of optimization in the learning process. In the upcoming chapter, we will explore the concept of optimization in the context of machine learning (ML) and deep learning (DL) and will present various optimization techniques that are employed in this field.

CHAPTER 2

Optimization methods

1 Introduction

Optimization is a field that encompasses a collection of mathematical principles and methods used to solve quantitative problems across various disciplines. It provides a framework for tackling physics, biology, engineering, economics, and business problems. In ML and DL, optimization is crucial, whereby utilizing optimization techniques, ML and DL models can achieve, through an optimal learning process, higher accuracy, faster convergence, and better scalability. Leading the machine learning model to greater effectiveness. In this chapter, we discuss optimization along with its specific type, and finally, we will address some optimization methods.

2 Optimization

Optimization is the process of finding the best possible solution to a problem or achieving the best possible outcome with respect to some criteria[33].
Formally, an optimization problem aims to find a vector of variables $x \in \mathbf{E}$, called also unknowns or parameters, that minimizes (or maximizes) a function $f(x)$ over the set $\mathbf{E}$, defined from the constraints on its variables [68].

Optimization plays a crucial role in machine learning (ML) and deep learning (DL) by enabling models to learn and improve their performance. In this field, optimization refers to the process of finding the optimal set of parameters or weights for a model that minimizes a specified loss or cost function. The objective is to iteratively adjust the model's parameters based on training data to optimize its performance and enhance its ability to make accurate predictions [23].

The usual and most common mathematical setting that is used in ML and DL is unconstrained optimization. In this setting the set $\mathbf{E} = \mathbb{R}^n$, where n denotes the number of parameters.

One way to solve such problems is to look closer at *stationary* or *critical* points. These are points when the gradient of the function f vanishes.

Various algorithms are employed to solve unconstrained optimization problems, including gradient-based methods like gradient descent and its variants. These algorithms calculate the gradients of the objective function with respect to the model's parameters and update the parameters iteratively in a direction that decreases the value of the objective function. By repeating this process over multiple iterations, the model gradually converges toward the optimal set of parameters that minimizes the objective function [5].

2.1 First order optimization

First-order optimization is based on the observation that the cost function $C(w_1, .., w_n)$ is differentiable and a minimum exists. In DL models specifically, the parameter space is very high dimensional, and the layered nature of DNNs results in highly complicated cost functions [23].
That is to say, a closed form solution of $\nabla C(w_1, .., w_n) = 0$ does not exist or otherwise, it is memory-consuming to compute. The iterative process of finding a local minimum namely gradient descent [56] is the adopted solution where we start from an initial solution and move in the parameter space using the negative of the gradient at the current position (which points at a local minimum) scaled by a learning rate α. The updating rule of GD is given by the following equation:

$$w_t = w_{t-1} - \alpha \frac{\partial C}{\partial w_t} \tag{2.1}$$

However, it is to mention that performance of GD, i.e. the ability to successfully minimize the cost, is subject to many factors; The training data properties, the DNN's architecture 3.2.2 as well as the choice of the hyper-parameters could potentially affect the convergence time and the overall training process. Several variants of GD emerged to address this problem [61].

3 Gradient-Based Optimization methods

3.1 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD) [54] is in fact the standard algorithm for training Deep Neural Networks (DNNs). It is a popular gradient descent optimization algorithm variant that updates model parameters based on the gradient of the objective function with respect to a small subset of training examples, known as a mini-batch. The SGD update rule is as follows:

$$w_{t+1} = w_t - \alpha \nabla_{w_t} J(w_t, \mathbf{x}i:t, \mathbf{y}i:t), \tag{2.2}$$

where w_t is the parameter vector at time step t
$J(w_t, \mathbf{x}i:t, \mathbf{y}i:t)$ is the loss function evaluated on the mini-batch of training examples $(\mathbf{x}i:t, \mathbf{y}i:t)$, and $\nabla_{w_t} J(w_t, \mathbf{x}i:t, \mathbf{y}i:t)$ is the gradient of the loss function with respect to the parameters at time step t.
SGD is a simple and efficient algorithm that can work well for a wide range of machine-learning tasks. However, it has some limitations such as being sensitive to the choice of learning rate and mini-batch size, and it can get stuck in local minima [56]. To address these limitations, various extensions and modifications of SGD have been proposed, such as momentum [49], and adaptive learning rate methods.

3.2 Momentum

In order to accelerate SGD, momentum SGD [49] keeps track of the history of gradients and applies exponential decay to place more emphasis on the newer direction. The below equation (2.3) is how the moving average of gradients is calculated:

$$v_t = \beta v_{t-1} + (1-\beta)\frac{\partial C}{\partial w_t} \tag{2.3}$$

Such that:
β: the momentum hyperparameter which controls the amount of history to include.
v_{t-1}: the moving average at time t-1.
$\frac{\partial C}{\partial w_t}$: the partial derivative of the cost function w.r.t the weight parameter vector w_t. Hence, the update rule:

$$w_{t+1} = w_t - \alpha v_t \tag{2.4}$$

3.3 NESTEROV ACCELERATED GRADIENT(NAG)

The main enhancement NAG [45] suggests in relation to momentum, is to determine directions based on history, and not just updates. Therefore, the moving average becomes as follows:

$$v_t = \beta v_t + \alpha \frac{\partial C}{\partial (w_t - \beta v_t)} \tag{2.5}$$

The derivative of the cost is computed w.r.t $(w_t - \beta v_t)$ which acts as an estimation of the new position of w allowing more careful steps which help avoid overshooting.

3.4 ADAPTIVE GRADIENT (ADAGRAD)

Adagrad [17] works by adapting the learning rate of each parameter based on the historical gradient information of that parameter. This adaptive learning rate helps Adagrad converge faster on parameter values that are important for the optimization problem. Equation (2.6) is the sum of squared gradients, (2.7) is the adaptive learning rate, and (2.8)is the update rule.

$$v_t = \sum_{i=0}^{t} (\frac{\partial C}{\partial w_i})^2 \tag{2.6}$$

$$\alpha_t = \alpha_{t-1} / (\sqrt{v}_t + \epsilon) \tag{2.7}$$

$$w_t = w_{t-1} - \alpha_t \frac{\partial C}{\partial w_t} \tag{2.8}$$

However, Adagrad may suffer from a diminishing learning rate problem as the historical gradient information accumulates over time, which can slow down convergence and prevent further learning. To address this issue, newer optimization algorithms such as Adam and RMSprop [62] have been developed.

3.5 ROOT MEAN SQUARE PROPAGATION (RMDPROP)

The RMSProp optimizer[62] is similar to the Adagrad optimizer, but it seeks to address one of its limitations, which is that the learning rate can become too small over time due to the accumulation of squared gradients. RMSProp solves this issue by introducing a decay factor that controls the rate at which the past gradients influence the learning rate. This decay factor allows the algorithm to adapt to changing gradients and adjust the learning rate accordingly. Equation (2.9) is the moving average of squared gradients and equation (2.10) is the update rule.

$$v_t = \beta v_{t-1} + (1-\beta)(\frac{\partial C}{\partial w_t})^2 \tag{2.9}$$

$$w_t = w_{t-1} - \frac{\alpha}{\sqrt{v_t} + \epsilon}\frac{\partial C}{\partial w_t} \tag{2.10}$$

3.6 ADAPTIVE MOMENT ESTIMATION (ADAM)

Adam[30]algorithms combine the heuristics of both Momentum and RMSProp.Hence, two moving averages are calculated (Equations (2.11), (2.12)). A bias correction is then applied on the calculated moments (Equations (2.13), (2.14)). Equation (2.15) is the update rule.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{2.11}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{2.12}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{2.13}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{2.14}$$

$$w_t = w_{t-1} - \alpha\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{2.15}$$

where m_t is the moving average of gradients, and v_t is the moving average of squared gradients. β_1, $\beta_2 \in]0,1]$ control the m_t and v_t respectively. $\hat{m}_t$ and $\hat{v}_t$ are the bias corrected m_t and v_t respectively.

3.7 ADABOUND

ADAPTIVE MOMENT ESTIMATION WITH DYNAMIC BOUND (ADABOUND)[39] addresses the problem of extreme step size caused by the unbounded learning rates of adaptive methods i.e these latter can explode or vanish uncontrollably due to excessively large or small gradients. Bounding the learning rates combines the stability of SGD (non-adaptive gradient-based methods) hence fair final generalization, and the fast convergence of ADAM (adaptive methods) [39]. by introducing a dynamic range that converges towards constant final values to prevent extremely large or small step sizes. Below is the formalization of the ADABOUND optimizer:

```
Input: $x_1 \in F$, initial step size $\alpha$, $\{\beta_{1t}\}_{t=1}^{T}$, $\beta_2$, lower bound function $\eta_l$, upper
       bound function $\eta_u$
Set $m_0 = 0$, $v_0 = 0$;
for $t = 1$ to $T$ do
    $g_t = \nabla f_t(x_t)$;
    $m_t = \beta_{1t} m_{t-1} + (1 - \beta_{1t}) g_t$;
    $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ and $V_t = \text{diag}(v_t)$;
    $\hat{\eta}_t = \text{Clip}\left(\frac{\alpha}{\sqrt{V_t}}, \eta_l(t), \eta_u(t)\right)$ and $\eta_t = \frac{\hat{\eta}_t}{\sqrt{t}}$;
    $x_{t+1} = \Pi_{F, \text{diag}(\eta_t^{-1})}(x_t - \eta_t \odot m_t)$;
end
```

Algorithm 1: AdaBound Algorithm

4 CONCLUSION

Optimization is very important for ML and DL because it helps improve the accuracy and efficiency of the models. By optimizing the algorithms and parameters, we can reduce the computational resources required for training and inference, which can save time and money. Additionally, optimization can also help prevent overfitting and improve generalization performance, making the models more robust and reliable. In this chapter, we talked about optimization in ML and DL, and we particularly discussed first-order optimization by highlighting six first-order optimizers.

CHAPTER 3

Contribution 1: Comparative Study

1 Introduction

In this Chapter, we will present a comparative study of five common gradient-based optimization techniques to highlight the effect of the optimizer on the performance and effectiveness of deep learning. Next, we will conduct another comparative analysis for the purpose of studying the effect of the sparsity of the model and that of the dataset considering different ML tasks.

Part I: A Comparative Study of the Impact of Different First-Order Optimizers on the Learning Process of U-Net for Change Detection Task.

Our first comparative study [15] is a more in-depth analysis than the one initiated in [6].

This study is conducted to determine the best-performing optimizer among five: Gradient descent with Momentum (Momentum GD) (3.2), Nesterov Accelerated Gradient (NAG) (3.3), Adaptive Gradient (AdaGrad) (3.4), Root Mean Square Propagation optimizer (RMSProp) (3.5), and the adaptive moment estimation optimizer (Adam) (3.6) where we train U-Net[55] architecture on CD dataset ONERA.

UNet has demonstrated its effectiveness in achieving robust learning even with a limited

number of data images, which makes it a suitable choice for change detection (CD) tasks using multi-spectral datasets like ONERA, where the available image count is relatively small [11, 35, 65].
Recent research has investigated the use and performance of UNet architecture in detecting changes in remote sensing images [7, 10, 36, 43]. In [7], UNet and UNet++ were trained and evaluated using high-resolution satellite images. The authors analyzed the impact of different loss functions, data augmentation, and deep supervision techniques on the models' performance.
In [43], the generality and performance of UNet and its variants were assessed on datasets with class imbalance and small region of interest size. They examined the suitability of these models for such challenging scenarios. Lv et al. [40] ccompared an existing deep learning-based change detection approach with a UNet-based approach proposed in their work. In [37], Li et al. evaluated a residual UNet model using Sentinel-1 SAR change detection dataset for urbanization.

However, it is worth noting that no specific study has been conducted to investigate the impact of different optimization methods on the performance of UNet architecture for change detection tasks. We believe that such a study would provide valuable insights into designing more efficient and high-performing UNet models for change detection, particularly when resources are limited.

2 METHODOLOGY

The below subsections detail the experimental setup requirements of the comparative study. Namely, the data and the preprocessing performed on it, the training setting, and finally the evaluation metrics.

2.1 DATASET

In this work we use the data set ONERA for the experiments, The CD data set contains 24 pairs of multi-spectral images captured by Sentinel-2 satellites between 2015 and 2018, depicting various cities worldwide, with spatial resolution range between 10m, 20m, and 60m. The data set includes the corresponding masks, where the white segments denote changes and the black regions denote areas that haven't changed or haven't undergone irrelevant modification [12]. It is important to note that the size of this data set is regarded as inadequate for training a typical deep neural network. However, the authors [55] claim that UNet can be effectively trained with just a few images.

2.2 PREPROCESSING

To preprocess the dataset, the original images were split into 777 patches of 256x256 images. The root mean squared error (3.3RMSE) between the difference image and the ground truth was used to calculate the image difference between the pairs. As a result, we decided to combine average intensity (AI) and absolute distance (AD).

2.3 TRAINING

The related experiments were performed using Google Colab with the following listed hardware specifications:
GPU: NVIDIA T4 16GB, 2560 CUDA cores, 585 MHz.
RAM: 12GB, 3200 MHz memory clock.
The model was trained by each of the five optimizers with three different learning rates: $10^-2, 10^-3$, and 10^-4

2.4 EVALUATION METRICS

We used Three metrics to evaluate the model's performance: Accuracy (acc) for an overall evaluation, Precision (prec) measure the relevant patterns that are correctly predicted from the total predicted patterns within the class, and F1-score (F1) which is the harmonic mean of the precision and the recall (another measure of the fraction of correctly classified relevant patterns)[26]. These latter are defined by the below equations :

$$acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

$$prec = \frac{TP}{TP + FP} \tag{3.2}$$

$$F1 = \frac{2TP}{2TP + FP + FN} \tag{3.3}$$

Where TP and TN, the True Positive and The True Negative, refer to the correctly classified patterns inside the class and outside of it respectively. Whereas FP and FN represent the misclassifications.

3 RESULTS

This experiment aims to understand the behavior of the UNet model when trained for the optimization of the binary cross entropy using the five optimization techniques with different learning rates. Tab. 3.1 provides us with the accuracy, precision, and F1-score of the model when trained over 200 epochs while Tab.3.2,3.3,3.4,3.5 and 3.6 illustrates the plots of the training and validation loss over epochs for the three learning rates values.

Table 3.1: The results of training UNet using the five optimization methods for 200 epochs.

learning rate	10^{-2}			10^{-3}			10^{-4}		
	Acc	prec	F1	Acc	prec	F1	Acc	precision	F1
Momentum	0.975	**0.382**	0.219	0.956	0.243	0.209	0.977	0.027	0.011
NAG	0.960	0.0002	0.0004	**0.978**	0.250	0.208	0.977	0.0006	0.0004
AdaGrad	0.971	0.290	0.200	0.972	0.201	0.198	**0.979**	0.002	0.004
RMSProp	**0.976**	0.274	0.249	0.975	0.435	**0.373**	0.973	0.350	**0.330**
Adam	0.971	0.330	**0.310**	0.977	**0.491**	**0.376**	0.972	**0.388**	**0.331**

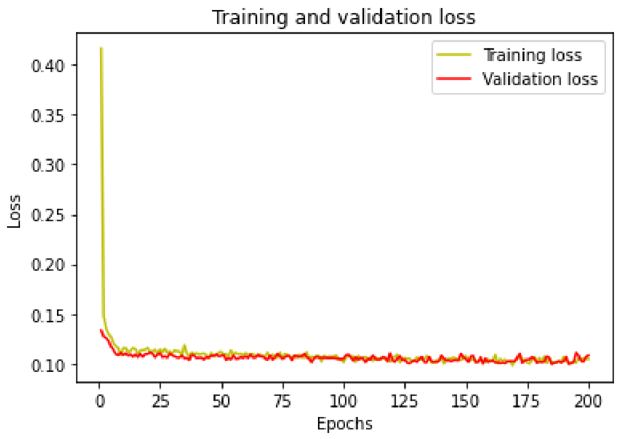
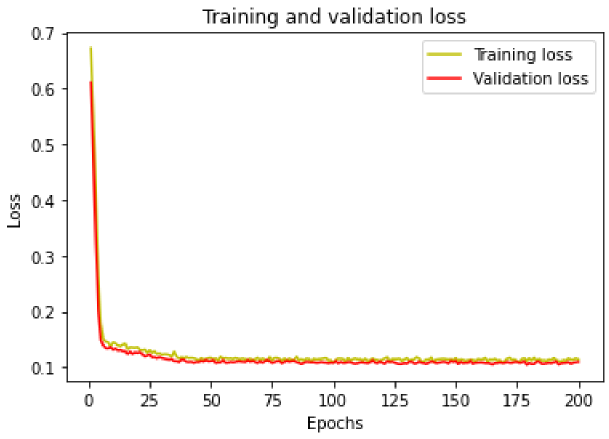
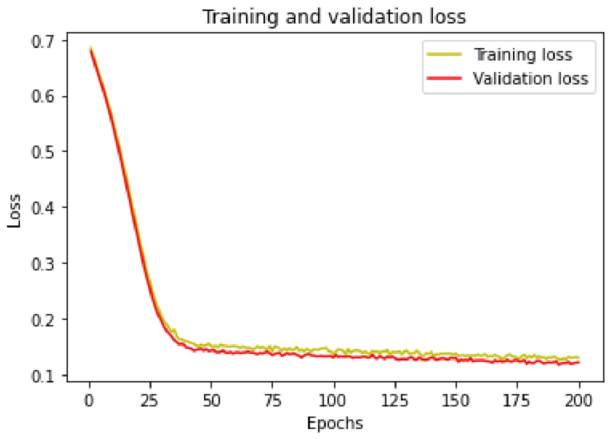
Learning rate LR=10^{-2}	
Learning rate LR=10^{-3}	
Learning rate LR=10^{-4}	

Table 3.2: SGD with Momentum: Training and Validation Loss.

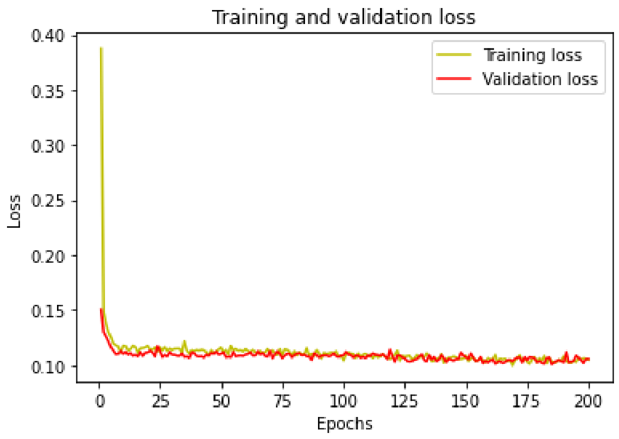
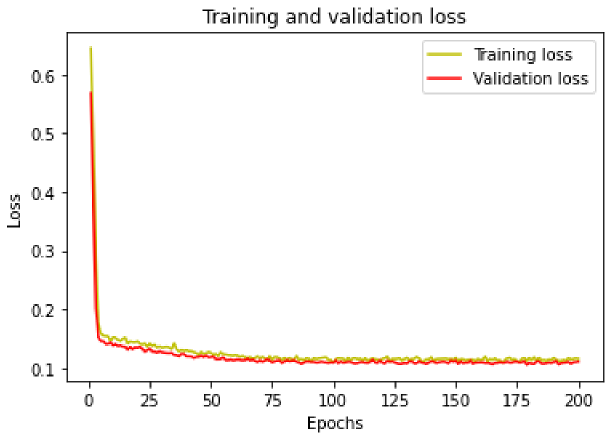
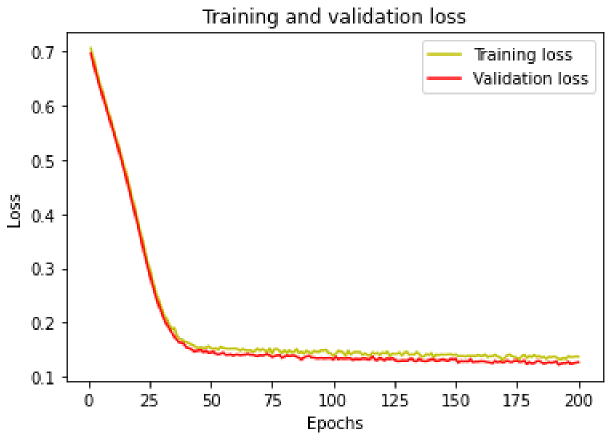
Learning rate LR=10^{-2}	
Learning rate LR=10^{-3}	
Learning rate LR=10^{-4}	

Table 3.3: NAG: Training and Validation Loss.

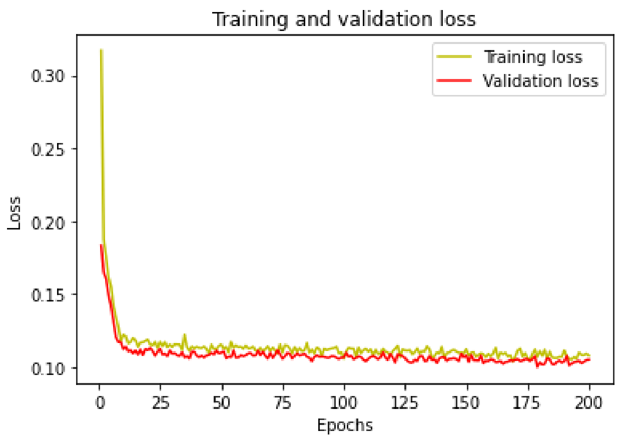
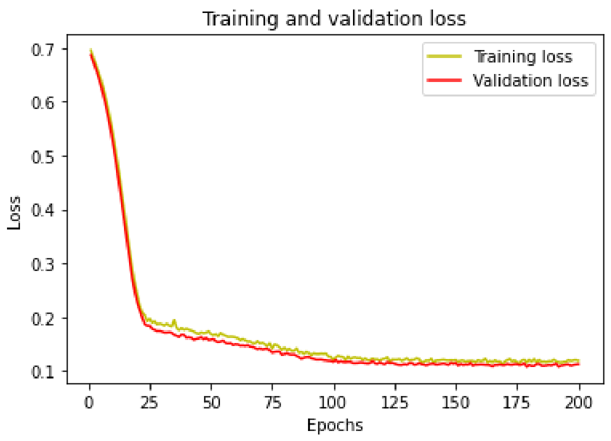
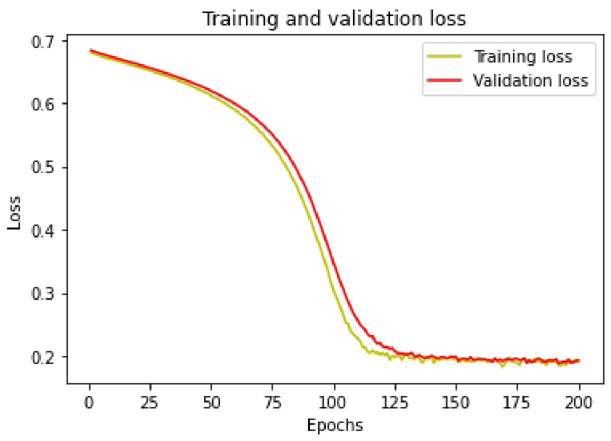
Learning rate LR=10^{-2}	
Learning rate LR=10^{-3}	
Learning rate LR=10^{-4}	

Table 3.4: AdaGrad: Training and Validation Loss.

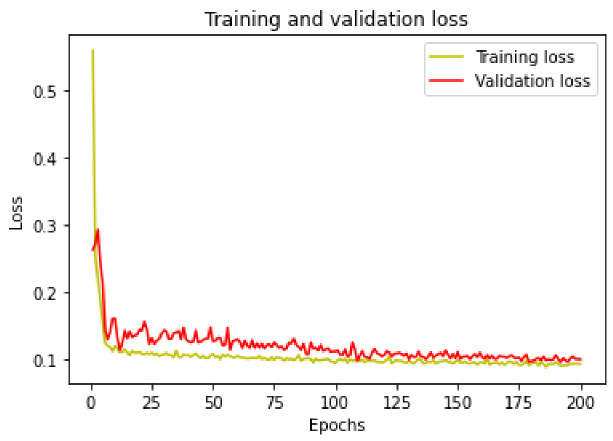
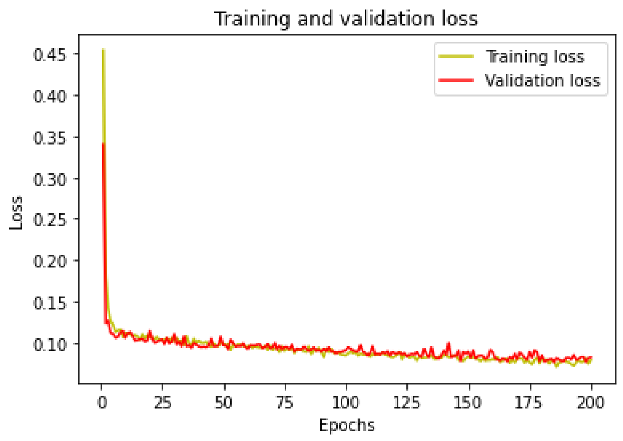
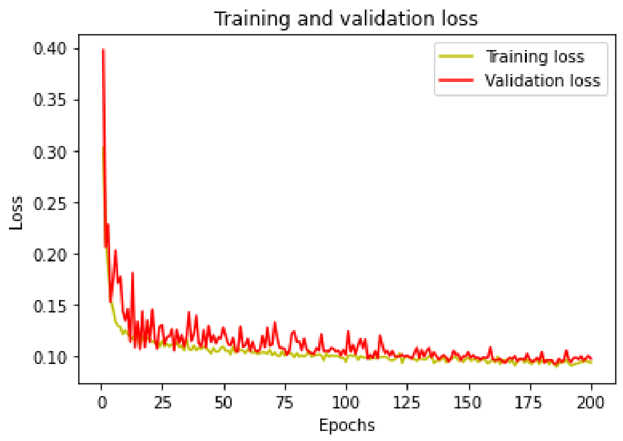
Learning rate LR=10^{-2}	
Learning rate LR=10^{-3}	
Learning rate LR=10^{-4}	

Table 3.5: RMSProp: Training and Validation Loss.

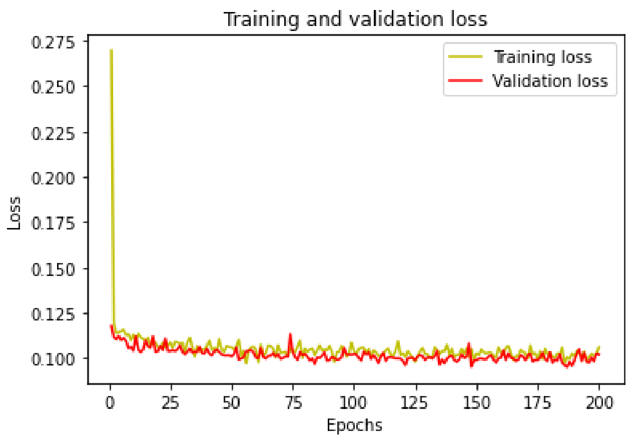
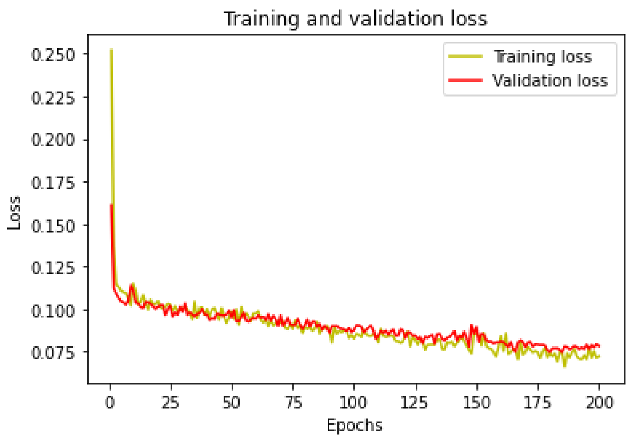
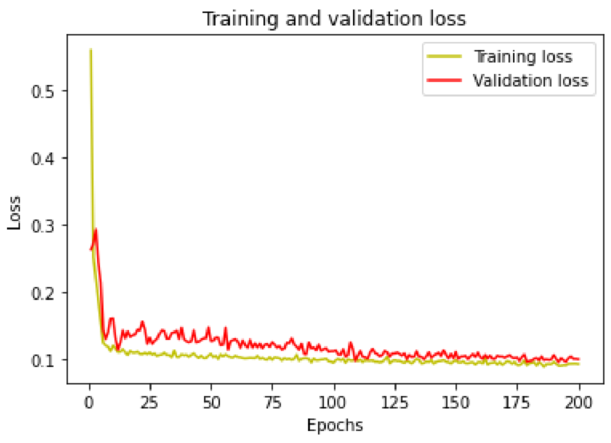
Learning rate LR=10^{-2}	
Learning rate LR=10^{-3}	
Learning rate LR=10^{-4}	

Table 3.6: Adam: Training and Validation Loss.

3.1 DISCUSSION

From Tables. (3.2), (3.3), (3.4), (3.5) and (3.6), we can notice that Momentum-based methods, i.e., Momentum and NAG, have more stable convergence w.r.t the learning rate than the rest of the methods. However, all of the optimizers stop significantly converging through the early epochs. On the other hand, at 10^{-3}, Adam, AdaGrad, and RMSProp converge even further. These three optimizers eliminate the effect of different scales among the model's parameters. Therefore, they converge with more accurate steps than Momentum and NAG and are less prone to falling into local minima (empirically shown by Kingma and Lei Ba [30] that Adam outperforms other methods in cases of non-convexity).

Moreover, regardless of the learning rate value, AdaGrad, Momentum, and NAG have large loss values compared to Adam and RMSProp. We observe that Momentum and NAG have similar behavior while the rest of the methods behave in another way similarly. We can interpret these two different ways as follow:

- Momentum and NAG Arrive at a local minimum due to large steps biased toward large-scale parameters.
- AdaGrad did not converge because of vanishing gradients.

From Tab. 3.5, despite RMSProp scoring decent quantitative results, it is the most sensitive to changing the learning rate value. In contrast, Momentum and NAG are the least sensitive. These latter demonstrated an interesting behavior of similarity in the way that they converge, while they have different update mechanisms (3.2, 3.3). During optimization, Momentum and NAG start with large steps in the first epochs and then slow down, given that in the early iteration, these two algorithms have equivalent update rules (moving average equals 0). Accelerating learning seems to have no significant effect on NAG since the convergence has not improved, reinforcing the hypothesis that Momentum and NAG arrive at local minima. At 10^{-4}, all methods' convergence slows remarkably.

It should be mentioned that better qualitative and quantitative results can be obtained with more training. Although the authors of [55] stated that UNet can be trained effectively with small data sets, this does not seem to be valid for all tasks and data patterns. The idea is that with more exposure to the training data, the model can learn the patterns and relationships in the data and thus make better predictions. However, over-fitting occurs when a model becomes too complex and starts to memorize the training data instead of learning the underlying patterns, resulting in poor performance on unseen data. Since ONERA is a small dataset, this is exactly what happens when training the UNET model on this dataset using different optimization techniques. Deep learning models typically require a large amount of data to learn complex patterns and generalize to new data. Even after we increased the dataset size using data augmentation techniques, it was not enough to train the model well. The medium-resolution dataset (ONERA) has irregular and non-continuous change patterns which make it harder for the model to generalize. Hence, the obtained quantitative results. This issue arises in similar tasks like crack detection [47] where the authors also attributed the difficulty of the task to the challenging patterns of cracks.

Part II: The Impact of Different Sparsity Levels of Models and Datasets on ADAM Optimizer

This second stage of experiments is motivated by the results of Part I, where we found that ADAM is the best-performing optimizer and a learning rate of 10^{-3} is the most suitable. Nevertheless, we noticed that ADAM's performance could be significantly influenced by the data sparsity.

To investigate this observation, we compare different sparsity-level models, U-Net, DenseU-Net, and DenseNet. Speaking of model sparsity, here we refer to the number of connections w.r.t all possible connections in the network architecture. The density/sparsity of the network can be calculated by referring to graph connectivity as follows:

$$\frac{\text{the number of connections}}{S} \tag{3.4}$$

Where S is the maximum possible number of connections in the same network:

$$S = \frac{1}{2}\sum_{j=0}^{N}\sum_{i=0}^{N} |L_i|.|L_j| \tag{3.5}$$

Such that L_i and L_j are layer number i and layer number j respectively and $|.|$ is the cardinality of a given layer (the number of neurons (vertices)).

We train using ADAM optimizer for BCE and focal Tversky losses, on dense and sparse datasets for three ML tasks, i.e. CD, image segmentation and object recognition. DenseU-Net and DenseNet-121 are used in the following experiments for segmentation and recognition, respectively, as dense architectures. In contrast, U-Net and its encoder are considered as sparser similarly. Moreover, in contrast to the first stage and based on the observation that ADAM will converge even further after the specified number of epochs, we use early stopping in a way that the training stops when the loss is no longer significantly changing.

4 METHODOLOGY

This section presents the details of the experimental setup of this study where we used the same training environment as in Part 1. Additional datasets and evaluation metrics are detailed hereafter.

4.1 DATASETS

- **ONERA** is the CD dataset which we use in the first comparative study.
- **Annotated Web Ears (AWE)** [19], AWE is a dataset of ear images that were collected from the web where each straining has 10 images with a total number of 100 subjects.
- **Sparse Annotated Web Ears** is the sparse version of AWE which we create by adding black a background to ear images with the ear forming only a tiny region of the training image.
- **Electron microscopy (EM)** [18] represents a 5x5x5μm section taken from the CA1 hippocampus region of the brain. It comprises two 1065x2048 images and their masks where the white segments correspond to mitochondria, and the black regions are irrelevant backgrounds.
- **Stanford Background (SB)** The data set contains 715 images chosen from public data sets: LabelMe, MSRC, PASCAL VOC, and Geometric Context. The selection criteria for the images were of outdoor scenes, having approximately 320-by-240 pixels, with at least one foreground object, having the horizon position within the

image (it need not be visible). Semantic and geometric labels were obtained using Amazon's Mechanical Turk (AMT).[3]

4.2 EVALUATION METRICS

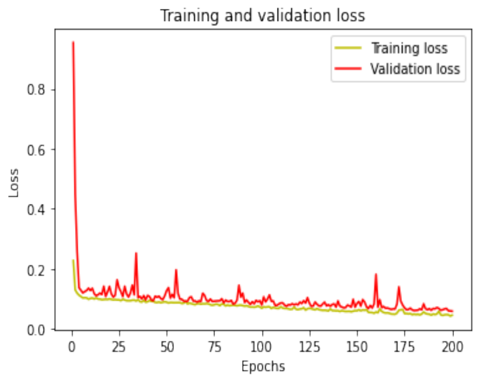
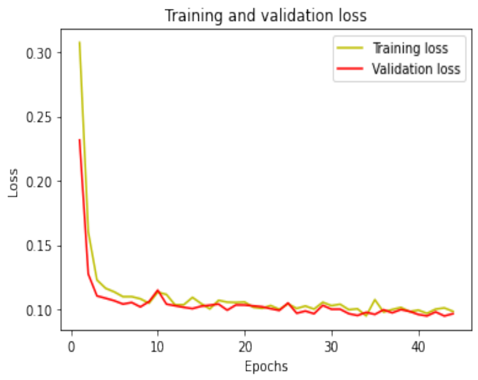
In addition to the three metrics given in Subsection (2.4), we use an estimated training speed where the convergence speed is measured according to the following equation:

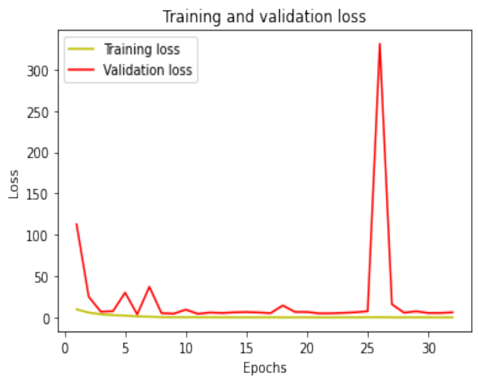
$$s = W/t \tag{3.6}$$

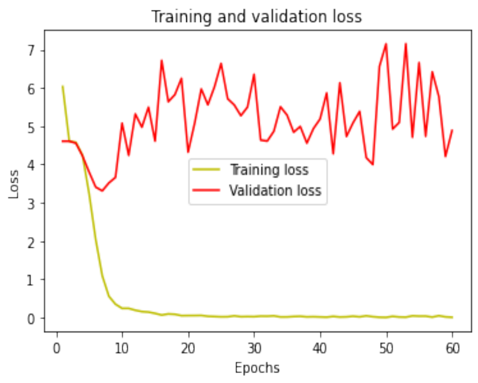
Where W is the number of trainable parameters, and t is the time step at which the training stopped.

4.3 RESULTS

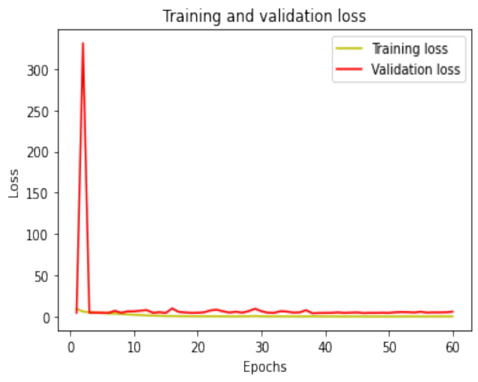
The following tables illustrate the experimental results of Part II(3.1).
The first two tables (3.7,3.8) show the cost plots for BCE and FTL cost functions respectively. The last two tables (3.9,3.10) show quantitative results of the aforementioned costs.

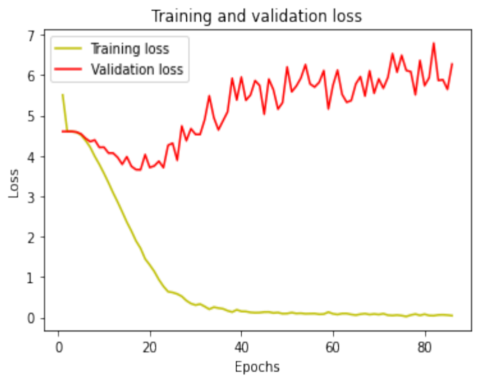
Dataset	Dense architecture	U-Net
ONERA		



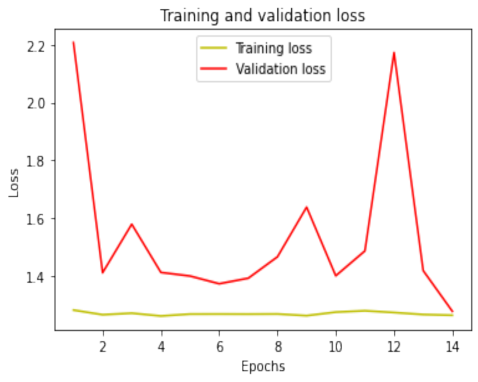


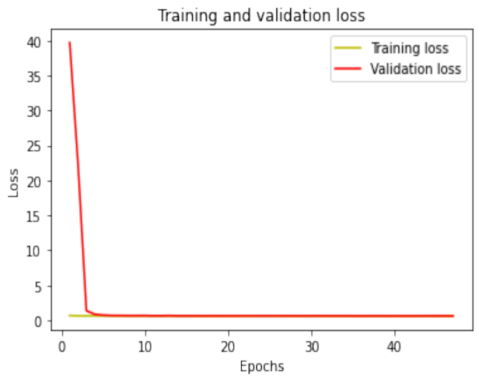
AWE





Sparse AWE





SB

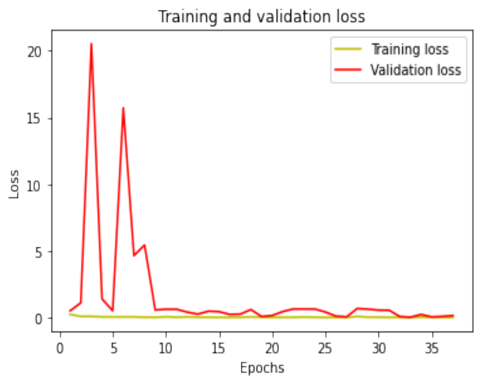
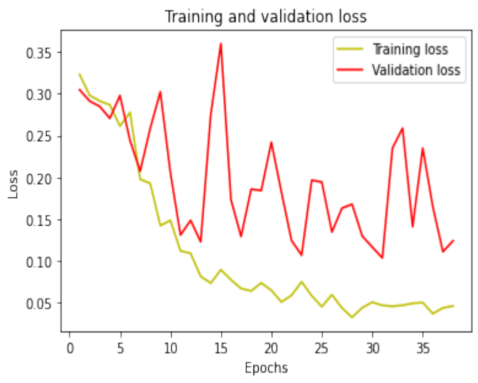
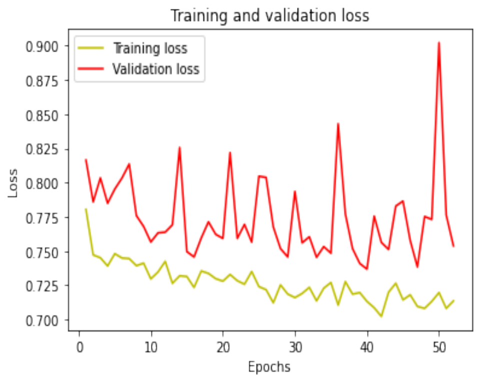
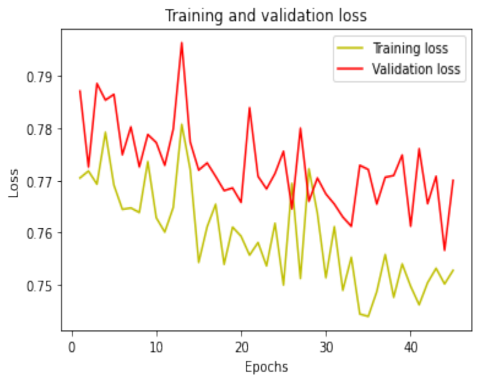
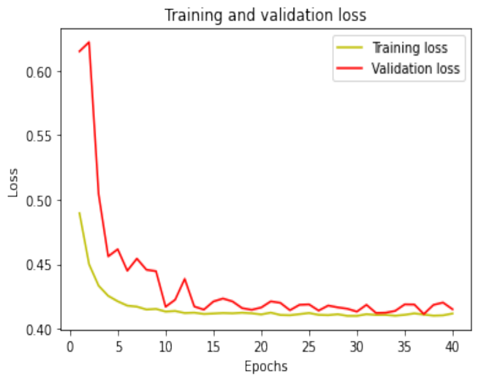
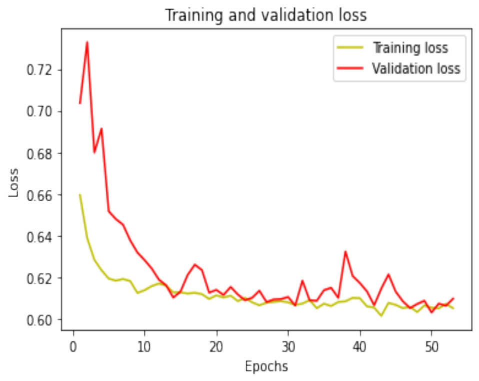
EM		

Table 3.7: The training and validation BCE cost on five different datasets with Adam optimizer and a learning rate of 10^{-3}. The number of epochs here depends on early stopping with a patience of 10 epochs.

Dataset	Dense architecture	U-Net
ONERA CD		
SB		

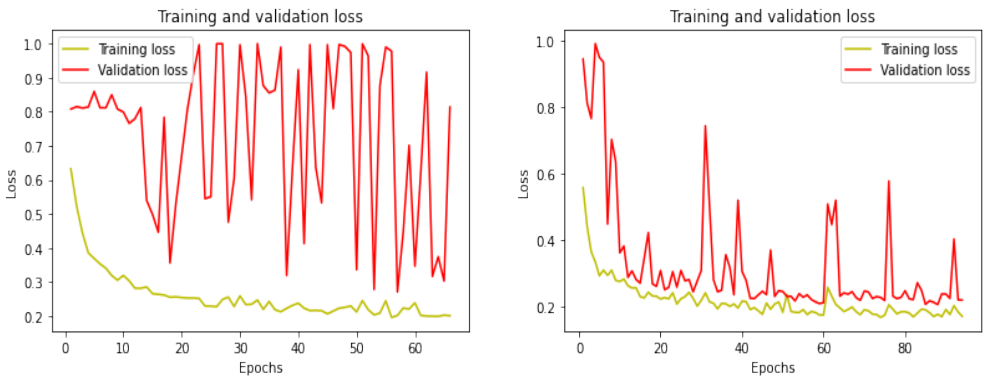
EM 

Table 3.8: The training and validation focal tversky cost of three different data sets with Adam optimizer and a learning rate of 10^{-3}. The number of epochs here depends on early stopping with a patience of 10 epochs.

Table 3.9 presents quantitative results of the Dense/sparse comparison in terms of the convergence speed.

Architecture	Dense				Sparse			
	speed	Acc	precision	F1	speed	Acc	precision	F1
ONERA	28,759.58	**0.982**	**0.43**	**0.33**	**44,116.66**	0.973	0.34	0.31
AWE	**319,711.12**	0.451	**0.52**	0.46	46,949.66	**0.501**	0.51	**0.48**
sparse AWE	**170,512.6**	0.397	0.41	0.31	33,140.94	**0.528**	**0.51**	**0.46**
SB	**616,276.78**	0.744	0.44	0.5	59,935.74	**0.829**	**0.66**	**0.67**
EM	**233,185.81**	0.941	0.76	0.52	74,131.05	**0.963**	**0.84**	**0.86**

Table 3.9: The results of training dense and sparse architectures (based on Densenet and UNet, respectively) with dense and sparse data sets. **loss = binary cross-entropy**

Architecture	Dense				Sparse			
	speed	Acc	precision	F1	speed	Acc	precision	F1
ONERA	**165,920.67**	**0.982**	**0.23**	**0.26**	43,135.66	0.912	0.19	0.26
SB	**215,696.87**	0.475	0.42	0.51	45,050.01	**0.800**	**0.69**	**0.65**
EM	**132,736.53**	0.888	0.30	0.16	25,673.66	**0.991**	**0.92**	**0.95**

Table 3.10: The results of training dense and sparse architectures (based on DenseNet and UNet, respectively) with dense and sparse data sets. **loss = focal Tversky** [1]

5 Discussion

As a first observation from the model perspective, from table 3.7, the dense model has remarkably more stable training convergence than the sparser one. In terms of training speed, the dense model outperformed the sparse model with an average of **235,689.17** parameter per epoch compared to **46,516.67** scored by the sparse model. This can be explained by the dense connectivity preventing the gradients from vanishing through the layers, which ensures updating all the weights, even those of earlier layers. The model is a major determining factor for object recognition because the training seems somewhat invariant to the data (sparse or dense) but varies remarkably with respect to the model.

From a dataset perspective, the most ideal case in all configurations is the EM data set which, despite having similarities with ONERA (sparse images which are hard to interpret by the model's tables (3.7) and (3.8) show that EM and ONERA have the most oscillating losses), the subject to the segment has fewer variations of shape (all mitochondria look very similar) whereas this is not the case of change which can have any shape and usually is a small region of the image. This interpretation is reinforced by the superior convergence speed of the EM dataset compared to ONERA. The sparsity of the EM data set can be viewed as a regularising factor since, for both losses, EM scored the highest metrics mostly. Also, the sparse model was less over-fitting (less validation loss oscillations).

Additionally, from a task perspective, the sparse model is more prone to over-fitting when the task is specifically object recognition. In fact, this can be interpreted by the classification/ranking loss flattening investigated by [25](we can see that the loss value equals 0 but the validation loss is drastically high and fluctuating). Next, from a loss perspective, for image segmentation, the region-based loss FTL seems to be superior to the distribution-based loss BCE with denser models but takes a lot longer to converge.

In conclusion, the ADAM optimizer seems to be more sensitive to the model than data sparsity since from the figures of tables (3.7) and (3.8) U-Net is more overfitted than the denser architectures. Furthermore, as for the first loss BCE, ADAM either gets trapped into local minima, in the cases where both the model and the data are sparser (U-Net with

ONERA and U-Net with EM), and in the case of denser architecture and denser data (SB with DenseU-Net). A possible interpretation could be that ADAM is highly prone to missing the global minimum in the cases of more severe sparsity or density although the optimizer does not necessarily overfit.
A major conclusion that we can draw from the experiment, is that ADAM suffers from overfitting due to falling into local minima, especially in sparser settings.

6 CONCLUSION

Throughout this chapter, we performed a two-fold comparative study whose first level examines the effect of the chosen optimizer on the U-Net deep model's effectiveness and performance. The second level was a more in-depth comparison where we focused on the best-performing optimizer: ADAM where different aspects such as the loss function, the training data, and the architecture were taken into consideration. The obtained results provided deeper insights for potential improvements. The upcoming chapter will detail experiments and the results as long as the description and the algorithms for our proposed optimizers.

CHAPTER 4

CONTRIBUTION 2: SIMULATED ANNEALING WITH DYNAMIC LEARNING RATE FOR ADAM OPTIMIZER

1 INTRODUCTION

To overcome the generalization issue which characterizes adaptive methods along the lines of ADAM, we attempted to develop an optimization method through a series of experiments using two of the popular ML datasets MNIST and CIFAR-10.
In this chapter, we will present in detail the formalization of the proposed methods, the experimental setup, the obtained results, and the discussion.

2 SIMULATED ANNEALING (SA)

Simulated annealing is a probabilistic algorithm proposed by [31] which simulates the cooling of materials in a heat bath, i.e., annealing. This latter can be used for both continuous and discrete problems [16]. The intuition behind this algorithm is to begin with random moves (ways of changing the system's configuration, steps in our case) and gradually start to make more careful ones until the optimal configuration is attained. Within an annealing system, we aim to find the best atoms positions value which maximizes the energy of the material, analogous to finding the best configuration that minimizes the cost of the model. The algorithm goes as follows:

```
$sc \leftarrow s0$;
$T \leftarrow$ system temperature;
**while** $T > T_{min}$ **do**
    Select $s \in E(sc)$;
    $\delta \leftarrow f(s) - f(sc)$;
    **if** $\delta < 0$ **then**
        $sc \leftarrow s$;
        **else**
            $r \leftarrow$ rand$[0, 1]$;
            **if** $e^{-\delta/T} > r$ **then**
                $sc \leftarrow s$
            **end**
        **end**
    **end**
    $T \leftarrow \alpha(T)$
**end**
```

Algorithm 2: Simulated Annealing

Where sc is the initial configuration, E is the energy function, δ is the difference in energy between the previous and the current configurations, $e^{-\delta/T}$ is the probability of an increase in energy magnitude which is compared with a random probability, and $\alpha(T)$ is the cooling function.

SA has been adopted into first-order optimization problems like SGD [20] where the authors proposed a metaheuristic to determine the best learning rate value from a discrete set of recommended values aiming to enhance the method's solution quality. AdaBound [39] uses the concept of 'annealing' yet without referring to the solution's quality.

3 ADAM WITH SA

Based on ADAM optimizer and SA, we attempted to develop an algorithm with an exploratory behavior in order to improve ADAM's capacity to find the global solution.

The below algorithm is a general scheme of our simulated annealing-based algorithms. The methods can be viewed as dual-phase optimizers which have an exploration and an optimization phase. The randomness of the learning rate α allows risky moves which in turn allow escaping the local minima, especially at the beginning of the training.

The value of the probability $\exp -\frac{\delta}{T}$ decreases as the value of T increases such that less moves (updates) are accepted.

VARIANT 1: ADAM WITH SA-BASED LEARNING RATE SELECTION (ADAM SA)

For the first variant, $[a, b]$ corresponds to $]0, 1]$ where α is sampled according to uniform distribution ($\alpha \leftarrow rand_U]0, 1]$). A new learning rate α is explored in each iteration meaning $nb\_epoch = 1$.

VARIANT 2: ADAM WITH SA-BASED LEARNING RATE DECAY (ADAM SA-DECAY)

Similarly to variant 1, $\alpha \sim U]0, 1]$ and $nb\_epoch = 1$, however cooling is applied on the learning rate itself that is $\alpha = T$. The direct effect of temperature and learning rate being the same value is that both the acceptance probability and the learning rate α decay.

VARIANT 3: ADAM WITH SA-BASED DYNAMIC UPPER BOUND (ADAM SA-DUB):

In order to prevent large learning rates, we introduced a dynamic upper bound which decreases over iterations while maintaining the same configuration in terms of α's distribution and nb_epoch. In this case, α and T are independent yet b decreases proportionally to T. In other words, no decaying is performed on α value.

$$b = 1 - \frac{1}{T} \tag{4.1}$$

First, we upper-bounded the learning rates and randomly sampled values within $[0, b)$ according to equation (4.1) from the uniform distribution. The bound failed to generalize and had a remarkably slow convergence.
Second, we changed the bounding method that is given in equation (**??**) and the initial range, based on the observation that large values prevent ADAM from converging or lead to unstable convergence. Besides, instead of updating the range for each iteration, it is given more iterations to try different solutions from the same range.

VARIANT 4: ADAM WITH SA-BASED DYNAMIC RANGE (ADAM SA-DR)

Likewise, to prevent any extreme learning rates from dominating the training, both upper and lower bounds are made dynamic, new a and b are computed and updated according to equations (4.2) and (**??**).

$$a = \frac{1}{T^p} \tag{4.2}$$

Our method is very similar to AdaBound as they both dynamically bound the learning rate with a gradually decreasing range. However, the main difference between AdaBound and ours lies in the learning rate selection. ADAM SA-DR selects the learning rate iteratively from a decreasing range whereas AdaBound selects it randomly at the first iteration than bounds it iteratively towards a smaller range. This makes AdaBound initialize exactly like any adaptive method.

We found experimentally that sampling α from uniform distribution is more consistent than normal distribution in terms of performance. A recommended configuration for ADAM SA-DR to reach the best performance: $(T = 10, p = 1, k = 10, nb\_epochs = 10$

ADAM WITH NORMALIZED LEARNING RATE (ADANORM):

This method is a variant of ADAM where we employed the l2 norm of the learning rate as a step size for each nb_epochs. Normalization was introduced into ADAM in the form of several methods. Normalized direction preserving ADAM [69] applies normalization on the weight vectors of hidden layers in order to preserve direction w.r.t input vector. The authors suggested that the generalization issue arises because adaptation leads ADAM to lose the correct direction of gradients. Keskar and Socher [29] proposed a strategy to improve generalization by switching from ADAM to SGD when an appropriate condition is satisfied. Furthermore, Logit attenuating weight normalization (LAWN) [25] addressed the problem from a loss perspective and linked the inferior generalization of ADAM to *the loss of weight adaptivity*. This latter proposed LAWN where weights are constrained after a number of epochs. The difference between our method and the existing methods consists in the following:

- The normalization is applied each nb_epochs instead of switching completely (correcting the direction gradually preventing the loss of adaptivity)

- The learning rate for the non-adaptive step is the norm of the adaptive learning rate (which is a vector in this case).

4 EXPERIMENTS, RESULTS, AND DISCUSSION

The following experiments are a series of comparisons of the proposed methods with existing adaptive and non-adaptive ones to validate our methods. For this purpose, we trained a simple 3 layer CNN, on two datasets namely MNIST [14] and CIFAR-10 [32].

- In the first experiment, we compared the two variants of SA ADAM namely ADAM with SA and ADAM SA-Decay to ADAM, SGD, SA SGD as shown in figures(4.1)(4.2)(4.3)(4.4).

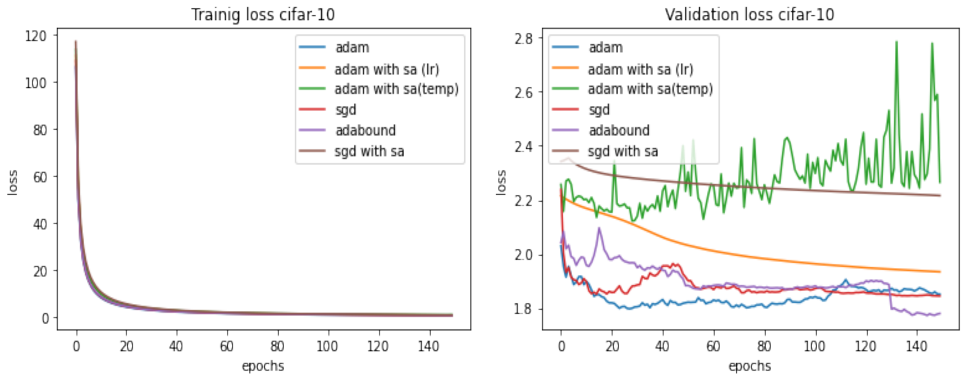


Figure 4.1: Training and validation loss of ADAM, ADAM with SA, ADAM with SA-Decay, SGD, AdaBound and SGD with SA on CIFAR-10

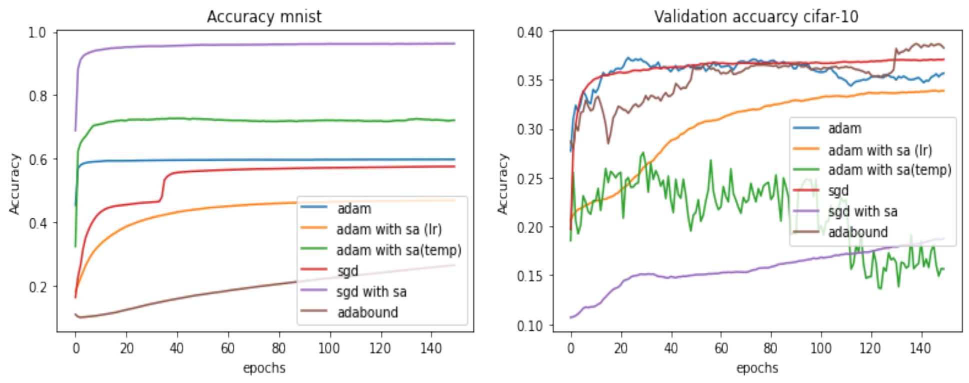


Figure 4.2: Training and validation accuracy of ADAM, ADAM with SA, ADAM with SA-Decay, SGD, AdaBound and SGD with SA on CIFAR-10

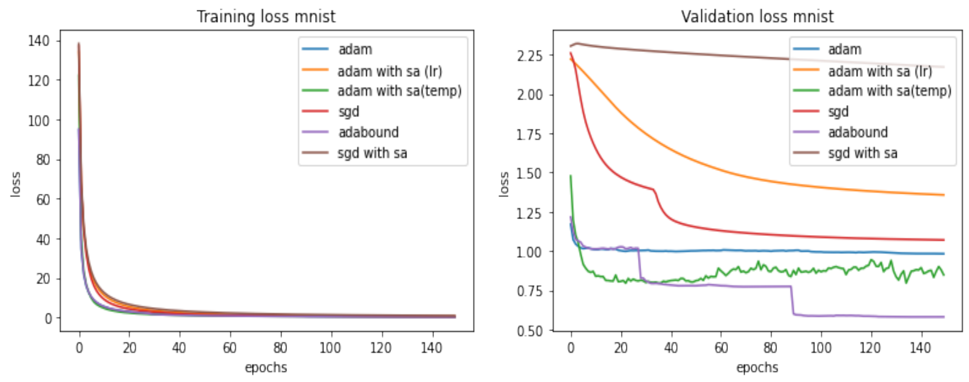


Figure 4.3: Training and validation loss of ADAM, ADAM with SA, ADAM with SA-Decay, SGD, AdaBound and SGD with SA on MNIST

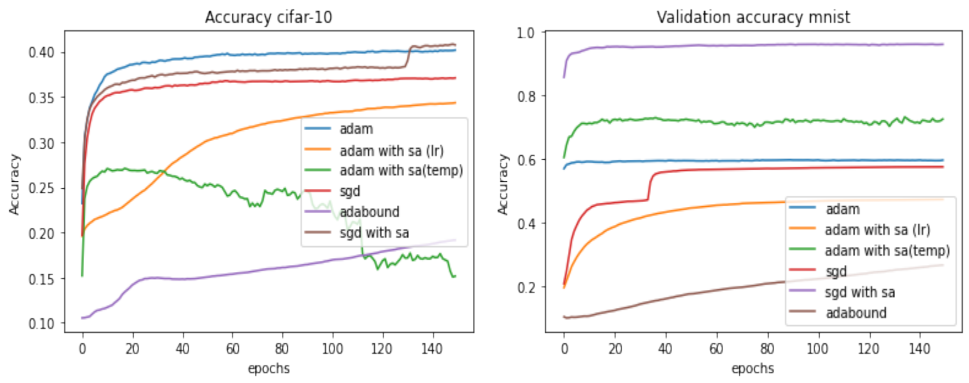


Figure 4.4: Training and validation accuracy of ADAM, ADAM with SA, ADAM with SA-Decay, SGD, AdaBound and SGD with SA on MNIST

- Second, we introduced other variants, ADAM SA-DUB, ADAM SA-DR, and ADANORM, and compared them with all previous methods plus AdaBound. Results in figures(4.5)(4.6)(4.7)(4.8)

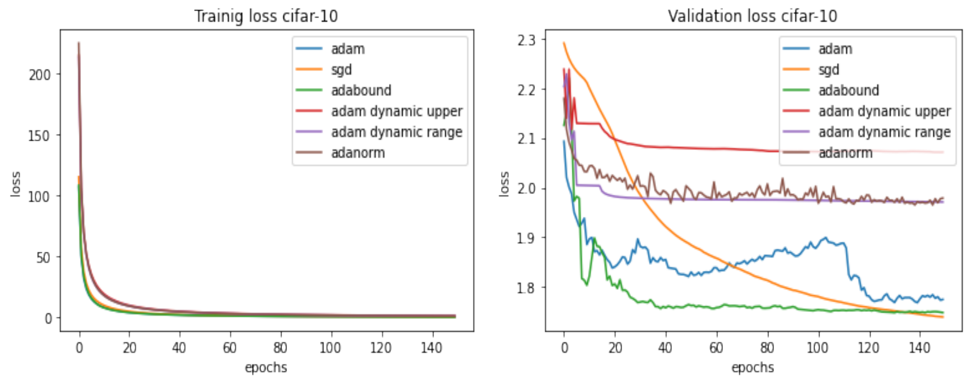


Figure 4.5: Training and validation loss of ADAM, SGD, AdaBound, ADAM with Dynamic Range, and Normalised LR on CIFAR-10

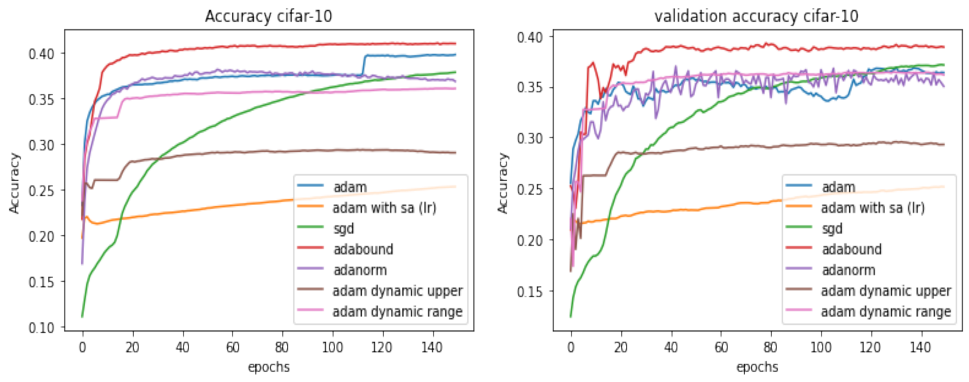


Figure 4.6: Training and validation accuracy of ADAM, SGD, AdaBound, ADAM with Dynamic Range, and Normalised LR on CIFAR-10

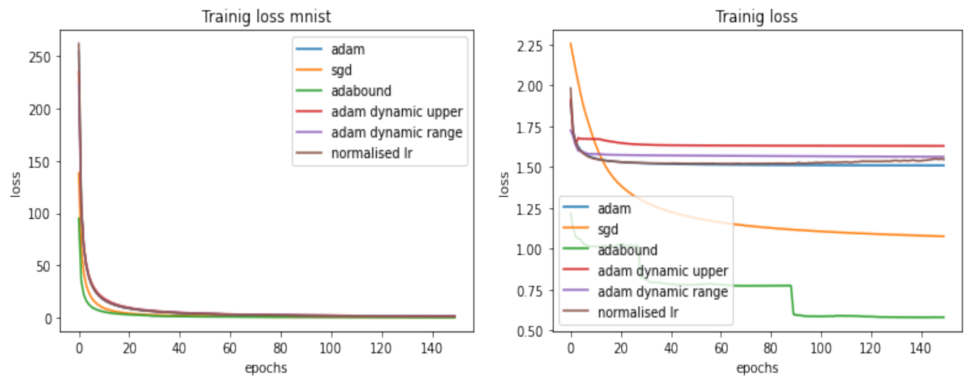


Figure 4.7: Training and validation loss of ADAM, SGD, AdaBound, ADAM with Dynamic Range, and Normalised LR on MNIST

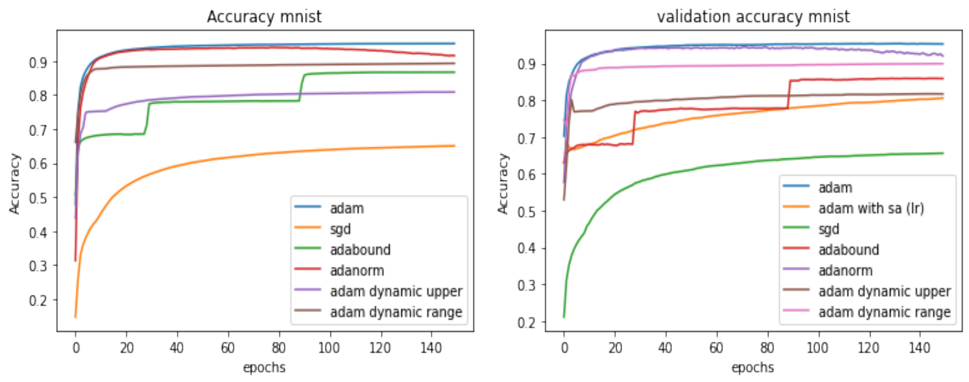


Figure 4.8: Training and validation accuracy of ADAM, SGD, AdaBound, ADAM with Dynamic Range, and Normalised LR on MNIST

- the final experiments was about determining the best configuration for our proposed method (4.9)(4.10)(4.11)(4.12)

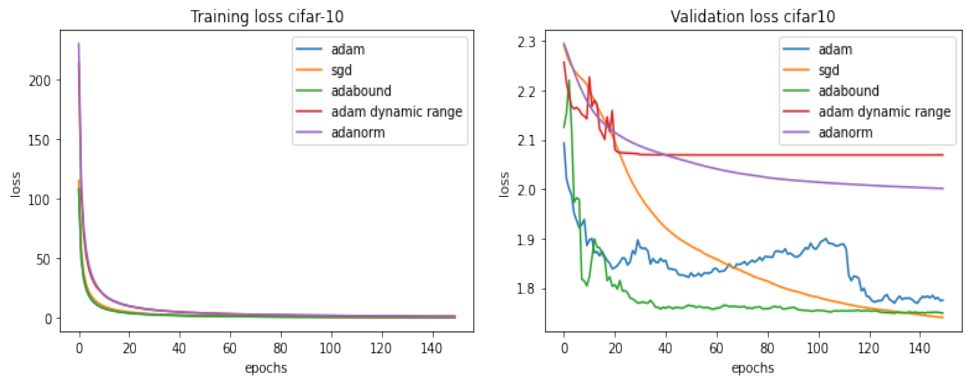


Figure 4.9: Training and validation loss of ADAM, SGD, AdaBound, ADAM with Dynamic Range, and Normalised LR on CIFAR-10

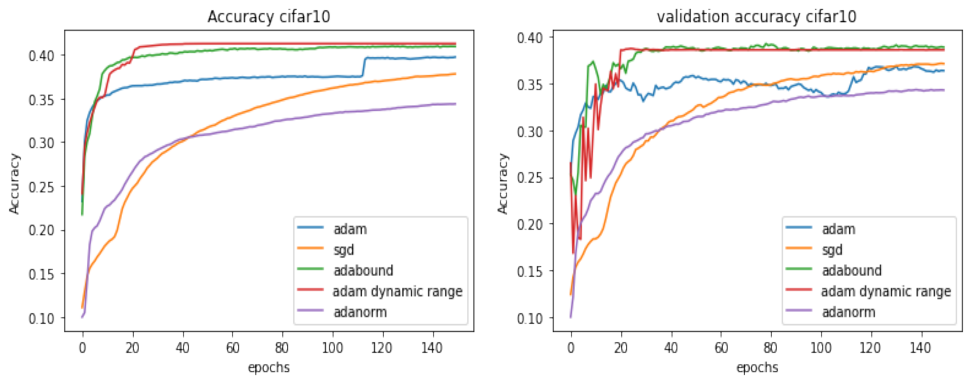


Figure 4.10: Training and validation accuracy of ADAM, SGD, AdaBound, ADAM with Dynamic Range, and Normalised LR on CIFAR-10

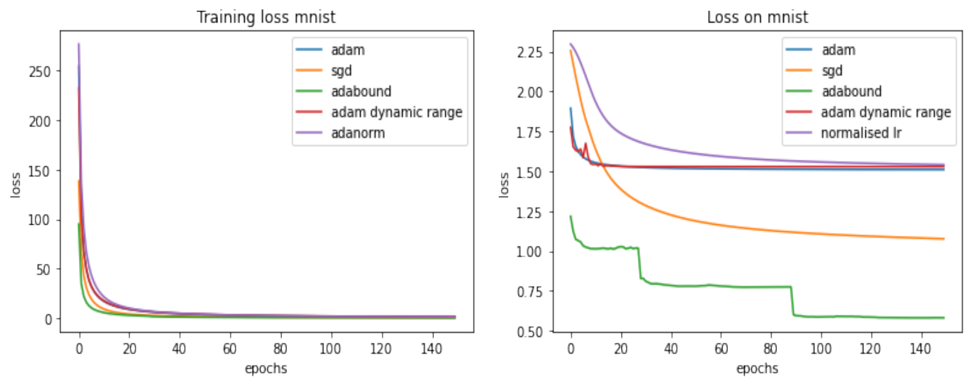


Figure 4.11: Training and validation loss of ADAM, SGD, AdaBound, ADAM with Dynamic Range, and Normalised LR on MNIST

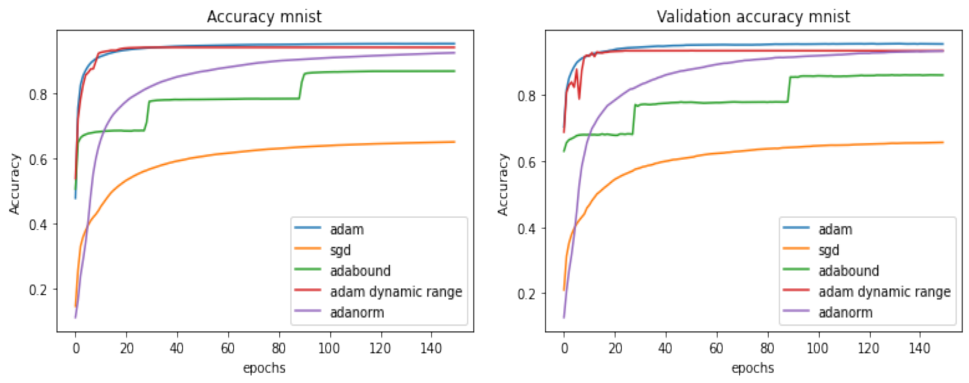


Figure 4.12: Training and validation accuracy of ADAM, SGD, AdaBound, ADAM with Dynamic Range, and Normalised LR on MNIST

-Compared to ADAM, ADAM SA-Decay heavily relies on the initial guess making it very sensitive to the initialization i.e. there is a much more remarkable difference among initial as well as final loss and accuracy values.
-Even though simulated annealing inspired first-order optimization methods are essentially based on the ’risky choices’ as it allows the algorithm to ’search’ for the global minimum, starting with a large step should be very careful as it potentially leads to not converging and ending up with very poor accuracy and performance.
From the first experiment, ADAM with SA-Decay has more stable convergence, w.r.t to training epochs and dataset than ADAM with SA. As a result, it is less overfitted. This can be interpreted by the following points:

- The randomness of the learning rate in ADAM with SA causes an instability through epochs.
- MNIST has easier patterns with well defined shapes and edges, which means ADAM with SA is potentially sensitive to noise in input data due to its randomness.

In the second experiment, we can see from the results that even with random selection, bounding the learning rate has a stabilizing effect. Also, the noisier the data (even though CIFAR-10 is numerically denser than MNIST which might be thought to prevent values near zero), the more extreme values of the learning rate has a greater effect degrading the performance. In other words, despite only considering extremely large values in ADAM SA-DUB, the difference in terms of performance was less significant on MNIST than CIFAR-10.

From the experimental results, we can see that even when our method ADAM SA-DR validation accuracy oscillates through the early epochs, it successfully stabilizes unlike ADAM. The last experiment shows that our method ADAM SA-DR generalizes better or the same as ADAM and performs superior or similar to AdaBound when tuned properly. Furthermore our method ADANORM despite being slower than ADAM SA-DR outperformed AdaBound on MNIST and as well generalized better than ADAM (has more stable validation accuracy).

5 Conclusion

Within this last chapter, we built a set of first-order optimizers upon a series of experiments where we adopted the SA algorithm to enhance the generalization of adaptive optimizers. Throughout the experiments, we compared our methods to existing adaptive and non-adaptive methods and obtained promising results.

General Conclusion

In conclusion, this thesis has empirically examined the impact of optimizer on DL, the properties of ADAM-like optimizers under different training conditions and finally proposed a set of adaptive optimizers that aim to address the generalization gap issue.

The first chapter was an overview about ML and DL where we introduced the field and explained related concepts. We put more emphasis on DL along with its main components and accentuated the importance of the learning process in DL.

The second chapter expanded the major learning process elements on which we focus through this work, namely optimization with the relevant methods and basic mathematical background about first-order optimizers.

In the third chapter, we conducted a first comparative study to investigate more about the impact of the optimizer on the training process and the performance of U-Net for CD. In light of the obtained results, further investigation of the performance of ADAM was elaborated. Its main objective is to analyze the effect of different kinds of sparsity on ADAM's performance through an extensive set of experiments.

In the last chapter, we proposed a set of ADAM-like optimizers for the purpose of addressing the generalization gap issue. To do so, we make use of the simulated annealing strategy that allows us to explore the space search and prevents the early convergence to local minima.

According to experiments, some of the variants we proposed showed promising results as they outperformed baseline algorithms.
Future work will consider further analysis and enhancements of the proposed optimizers.

Bibliography

1. Abraham, N. & Khan, N. M. *A novel focal tversky loss function with improved attention u-net for lesion segmentation* in *2019 IEEE 16th international symposium on biomedical imaging (ISBI 2019)* (2019), 683–687.
2. Alzubaidi, L., Zhang, J., Humaidi, A. J. & et al. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data* **8,** 53. https://doi.org/10.1186/s40537-021-00444-8 (2021).
3. Ashwath, B. *Stanford Background Dataset* Sept. 2020. https://www.kaggle.com/datasets/balraj98/stanford-background-dataset?resource=download.
4. Bishop, C. M. & Nasrabadi, N. M. *Pattern recognition and machine learning* **4** (Springer, 2006).
5. Bottou, L., Curtis, F. E. & Nocedal, J. *Optimization Methods for Large-Scale Machine Learning* (SIAM, 2018).
6. Boukraa, I., Dokkar, B., Meddour, B. & Kherfi, M. *Optimization Methods in Machine Learning and Deep Learning* (2021).
7. Bousias Alexakis, E. & Armenakis, C. Evaluation of UNet and UNet++ architectures in high resolution image change detection applications. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* **43,** 1507–1514 (2020).
8. Cao, Y., Liu, S., Peng, Y. & Li, J. DenseUNet: densely connected UNet for electron microscopy image segmentation. *IET Image Processing* **14,** 2682–2689 (2020).

9. Chen, L.-C., Papandreou, G., Schroff, F. & Adam, H. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587* (2017).
10. Daudt, R. C., Le Saux, B. & Boulch, A. *Fully convolutional siamese networks for change detection* in *2018 25th IEEE International Conference on Image Processing (ICIP)* (2018), 4063–4067.
11. Daudt, R. C., Le Saux, B., Boulch, A. & Gousseau, Y. Multitask learning for large-scale semantic change detection. *Computer Vision and Image Understanding* **187,** 102783 (2019).
12. Daudt, R. C., Le Saux, B., Boulch, A. & Gousseau, Y. *Urban change detection for multispectral earth observation using convolutional neural networks* in *IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium* (2018), 2115–2118.
13. De Prado, M. L. *Advances in financial machine learning* (John Wiley & Sons, 2018).
14. Deng, L. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* **29,** 141–142 (2012).
15. Dokkar, B., Meddour, B., Bouanane, K., Allaoui, M. & Kherfi, M. L. *A Comparative Study of the Impact of Different First Order Optimizers on the Learning Process of UNet for Change Detection Task.* in *ISPR, The international conference on Intelligent Systems and Pattern Recognition* (Springer, May 2023).
16. Dowsland, K. A. & Thompson, J. Simulated annealing. *Handbook of natural computing,* 1623–1655 (2012).
17. Duchi, J., Hazan, E. & Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research* **12** (2011).
18. *Electron microscopy dataset* https://www.epfl.ch/labs/cvlab/data/data-em/.
19. Emeršič, Ž., Štruc, V. & Peer, P. Ear recognition: More than a survey. *Neurocomputing* **255,** 26–39 (2017).
20. Fischetti, M. & Stringher, M. Embedded hyper-parameter tuning by simulated annealing. *arXiv preprint arXiv:1906.01504* (2019).
21. Forsyth, D. & Ponce, J. *Computer Vision: A Modern Approach* (Prentice Hall Professional Technical Reference, 2002).

22. Fujiyoshi, H., Hirakawa, T. & Yamashita, T. Deep learning-based image recognition for autonomous driving. *IATSS Research* **43,** 244–252 (2019).

23. Goodfellow, I., Bengio, Y. & Courville, A. *Deep Learning* (MIT Press, 2016).

24. Görner, M., Gillard, R. & Lakshmanan, V. *Practical Machine Learning for Computer Vision* (O'Reilly Media, 2017).

25. Gupta, A. *et al.* Logit attenuating weight normalization. *arXiv preprint arXiv:2108.05839* (2021).

26. Hossin, M. & Sulaiman, M. N. A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process* **5,** 1 (2015).

27. Huang, G., Liu, Z., Van Der Maaten, L. & Weinberger, K. Q. *Densely connected convolutional networks* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017), 4700–4708.

28. Jain, A. K., Topchy, A., Law, M. H. C. & Buhmann, J. M. *Landscape of clustering algorithms* in *International Conference on Pattern Recognition (ICPR 2004)* (2004), 260–263.

29. Keskar, N. S. & Socher, R. Improving generalization performance by switching from adam to sgd. *arXiv preprint arXiv:1712.07628* (2017).

30. Kingma, D. P. & Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

31. Kirkpatrick, S., Gelatt, C. D. & Vecchi, M. P. Optimization by Simulated Annealing. *Science* **220,** 671–680 (1983).

32. Krizhevsky, A., Nair, V. & Hinton, G. CIFAR-10 (Canadian Institute for Advanced Research). http://www.cs.toronto.edu/~kriz/cifar.html.

33. Lange, K. *Optimization* (Springer Science & Business Media, 2013).

34. LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* **86,** 2278–2324 (1998).

35. Leenstra, M., Marcos, D., Bovolo, F. & Tuia, D. *Self-supervised pre-training enhances change detection in Sentinel-2 imagery* in *Pattern Recognition. ICPR International Workshops and Challenges: Virtual Event, January 10-15, 2021, Proceedings, Part VII* (2021), 578–590.

36. Lei, T. *et al. End-to-end change detection using a symmetric fully convolutional network for landslide mapping* in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2019), 3027–3031.

37. Li, L., Wang, C., Zhang, H. & Zhang, B. *Residual UNet for urban building change detection with Sentinel-1 SAR data* in *IGARSS 2019-2019 IEEE International Geoscience and Remote Sensing Symposium* (2019), 1498–1501.

38. Lodi, A. & Zarpellon, G. On learning and branching: a survey. *Top* **25,** 207–236 (2017).

39. Luo, L., Xiong, Y., Liu, Y. & Sun, X. Adaptive gradient methods with dynamic bound of learning rate. *arXiv preprint arXiv:1902.09843* (2019).

40. Lv, Z. *et al.* Simple multiscale unet for change detection with heterogeneous remote sensing images. *IEEE Geoscience and Remote Sensing Letters* **19,** 1–5 (2022).

41. Menghani, G. Efficient deep learning: A survey on making deep learning models smaller, faster, and better. *ACM Computing Surveys* **55,** 1–37 (2023).

42. Minaee, S. *et al.* Image segmentation using deep learning: A survey. *IEEE transactions on pattern analysis and machine intelligence* **44,** 3523–3542 (2021).

43. Moustafa, M. S., Mohamed, S. A., Ahmed, S. & Nasr, A. H. Hyperspectral change detection based on modification of UNet neural networks. *Journal of Applied Remote Sensing* **15,** 028505–028505 (2021).

44. Nadkarni, P. M., Ohno-Machado, L. & Chapman, W. W. Natural language processing: an introduction. *Journal of the American Medical Informatics Association* **18,** 544–551 (2011).

45. Nesterov, Y. *A method for unconstrained convex minimization problem with the rate of convergence O (1/k^ 2)* in *Doklady an ussr* **269** (1983), 543–547.

46. Peng, D., Zhang, Y. & Guan, H. End-to-end change detection for high resolution satellite images using improved unet++. *Remote Sensing* **11,** 1382 (2019).

47. Polovnikov, V., Alekseev, D., Vinogradov, I. & Lashkia, G. V. DAUNet: Deep augmented neural network for pavement crack segmentation. *IEEE Access* **9,** 125714–125723 (2021).

48. Portugal, I., Alencar, P. & Cowan, D. The use of machine learning algorithms in recommender systems: A systematic review. *Expert Systems with Applications* **97,** 205–227 (2018).

49. Qian, N. On the momentum term in gradient descent learning algorithms. *Neural networks* **12,** 145–151 (1999).

50. Reddi, S. J., Kale, S. & Kumar, S. *On the Convergence of Adam and Beyond* 2019. arXiv: 1904.09237 [cs.LG].

51. Redmon, J., Divvala, S., Girshick, R. & Farhadi, A. *You only look once: Unified, real-time object detection* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), 779–788.

52. Ren, S., He, K., Girshick, R. & Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems* **28** (2015).

53. Renza, D., Martinez, E. & Arquero, A. A new approach to change detection in multispectral images by means of ergas index. *IEEE Geoscience and Remote Sensing Letters* **10,** 76–80 (2012).

54. Robbins, H. & Monro, S. A stochastic approximation method. *The annals of mathematical statistics,* 400–407 (1951).

55. Ronneberger, O., Fischer, P. & Brox, T. *U-net: Convolutional networks for biomedical image segmentation* in *International Conference on Medical image computing and computer-assisted intervention* (2015), 234–241.

56. Ruder, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).

57. Sairamya, N., Susmitha, L., George, S. T. & Subathra, M. in *Intelligent Data Analysis for Biomedical Applications* 253–273 (Elsevier, 2019).

58. Samuel, A. L. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development* **3,** 210–229 (1959).

59. Shewchuk, J. R. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain* Technical Report (Carnegie Mellon University, 1994).

60. Singh, A. Review article digital change detection techniques using remotely-sensed data. *International Journal of Remote Sensing* **10,** 989–1003 (1989).

61. Smith, L. N. *A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay* 2018. arXiv: 1803.09820 [cs.LG].

62. Tieleman, T. & Hinton, G. Rmsprop: Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning. *COURSERA Neural Networks Mach. Learn* **17** (2012).

63. Ukaoha, K. & Igodan, E. Architecture Optimization Model for the Deep Neural Network. *International Journal of Intelligent Computing and Information Sciences* **19,** 1–16 (Oct. 2019).

64. Van Engelen, J. E. & Hoos, H. H. A survey on semi-supervised learning. *Machine learning* **109,** 373–440 (2020).

65. Wang, Q., Yuan, Z., Du, Q. & Li, X. GETNET: A general end-to-end 2-D CNN framework for hyperspectral image change detection. *IEEE Transactions on Geoscience and Remote Sensing* **57,** 3–13 (2018).

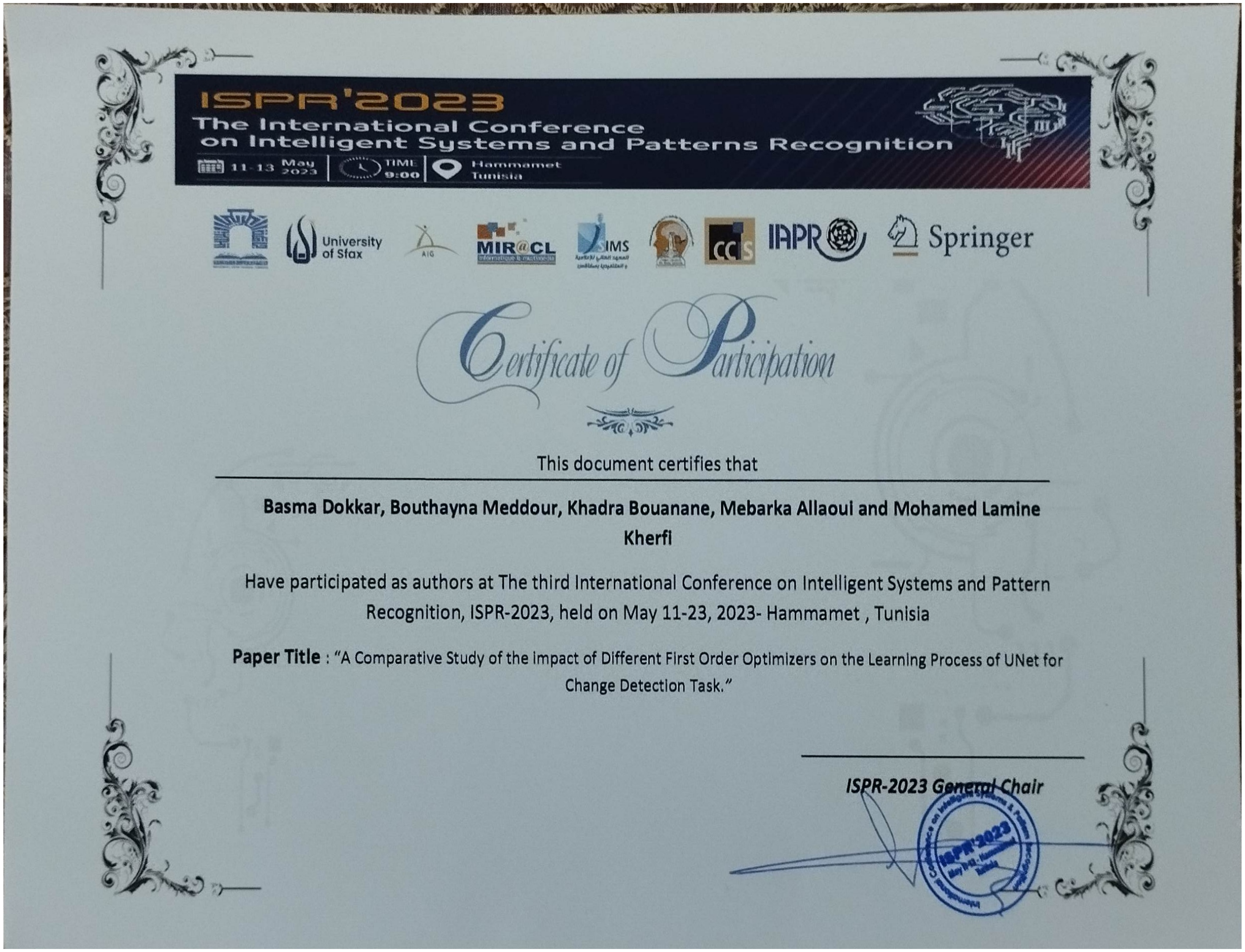
66. Wiens, J. & Shenoy, E. S. Machine learning for healthcare: on the verge of a major shift in healthcare epidemiology. *Clinical Infectious Diseases* **66,** 149–153 (2018).

67. Wilson, A. C., Roelofs, R., Stern, M., Srebro, N. & Recht, B. The marginal value of adaptive gradient methods in machine learning. *Advances in neural information processing systems* **30** (2017).

68. Wright, S. J. *Numerical optimization* 2006.

69. Zhang, Z., Ma, L., Li, Z. & Wu, C. Normalized direction-preserving adam. *arXiv preprint arXiv:1709.04546* (2017).

70. Zhou, P., Feng, J., Ma, C., Xiong, C., Hoi, S. C. H., *et al.* Towards theoretically understanding why sgd generalizes better than adam in deep learning. *Advances in Neural Information Processing Systems* **33,** 21285–21296 (2020).



ISPR'2023
The International Conference
on Intelligent Systems and Patterns Recognition
11-13 May 2023 | TIME 9:00 | Hammamet Tunisia

University of Sfax
AIG
MIR@CL
IMS
IAPR
Springer

Certificate of Participation

This document certifies that

Basma Dokkar, Bouthayna Meddour, Khadra Bouanane, Mebarka Allaoui and Mohamed Lamine Kherfi

Have participated as authors at The third International Conference on Intelligent Systems and Pattern Recognition, ISPR-2023, held on May 11-23, 2023- Hammamet , Tunisia

Paper Title : "A Comparative Study of the Impact of Different First Order Optimizers on the Learning Process of UNet for Change Detection Task."

ISPR-2023 General Chair

[illegible]

International Conference on Intelligent Systems & Pattern Recognition
ISPR'2023

République Algérienne Démocratique Et Populaire

Université Kasdi Merbah- Ouargla
Faculté des Nouvelles Technologies
de l'Information et de la Communication
Département d' Informatique et technologie de l'information

AUTORISATION DE SOUTENANCE Master II
Année universitaire : 2022/2023

Encadreur :
Nom : Bouanane
Prénom : Khadra

Candidats :
Nom /Prénom : Dokkar Basma
Nom / Prénom : Meddour Bouthayna

Spécialité : Intelligence Artificielle et Sciences de données.

Titre du mémoire :
First Order Optimization Methods for Deep Learning.

Ouargla le : 13/06/2023
Signature
[illegible]