A scalable and component-based deep learning parallelism platform : an application to convolutional neural networks for medical imaging segmentation.

Soulaimane Guedria

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

Spécialité : Informatique

Arrêté ministériel : 25 mai 2016

Présentée par

Soulaimane GUEDRIA

Thèse dirigée par **Noël DE PALMA**, Université Grenoble Alpes
et codirigée par **Nicolas VUILLERME**, Université Grenoble Alpes

préparée au sein du **Laboratoire d'Informatique de Grenoble et AGEIS**
dans **l'École Doctorale Mathématiques, Sciences et technologies de l'information, Informatique**

Une plateforme d'apprentissage profond à base de composants qui passe à l'échelle: une application aux réseaux de neurones convolutionnels pour la segmentation en imagerie médicale.

A scalable and component-based deep learning parallelism platform: an application to convolutional neural networks for medical imaging segmentation.

Thèse soutenue publiquement le **8 Juillet 2020,**
devant le jury composé de :

Monsieur Lionel SEINTURIER
Professeur, Université de Lille, Rapporteur

Monsieur Alain TCHANA
Professeur, Université de Lyon, Rapporteur

Monsieur François ESTÈVE
PU-PH, Université Grenoble Alpes et CHU Grenoble Alpes, Examinateur, Président du Jury

Monsieur Philippe SABATIER
Professeur, Université de Lyon, Examinateur

Monsieur Nicolas VUILLERME
Maître de conférences, HDR, Université Grenoble Alpes et Institut Universitaire de France, Co-Directeur de thèse

Monsieur Noël DE PALMA
Professeur, Université Grenoble Alpes, Directeur de thèse

Remerciements

Tout d'abord je tiens à remercier vivement et témoigner toute ma reconnaissance à mes directeurs de thèse Pr. Noël DE PALMA et Dr. Nicolas VUILLERME, HDR pour leur encadrement, la confiance qu'ils m'ont accordée, leur disponibilité ainsi que leur soutien tout au long de ces trois années.

Je voudrais également remercier vivement Pr. Lionel SEINTURIER, Pr. Alain TCHANA, Pr. François ESTÈVE et Pr. Philippe SABATIER d'avoir accepté de participer au jury de thèse et de m'avoir fait l'honneur de bien vouloir arbitrer ma soutenance avec rigueur et bienveillance.

Je souhaite également témoigner ma reconnaissance et toute ma gratitude pour toute ma famille pour leur soutien constant et leurs encouragements. Je suis redevable à mes parents, Hassine et Hamide, ma soeur Khouloud et mon frère Khalil qui m'ont offert leur soutien indéfectible et leur entière confiance. Enfin, j'ai une pensée toute particulière à celles et ceux qui nous sont chers et qui nous ont quittés avant cet aboutissement. En particulier, mes grands-parents, Khamis et Feyza GUEDRIA et Khalifa et Om Saad WANNES ainsi que si Larbi HADJ SAID. Que leurs âmes reposent en paix!

Je souhaite témoigner ma reconnaissance également à tous les membres du LIG et particulièrement l'équipe ERODS pour la bonne ambiance de travail mais également pour les nombreux bons moments passés ensemble et toutes nos discussions autour d'un café.

Je voudrais exprimer ma profonde gratitude et mes sincères remerciements au Dr. Edouard SORIANO pour son humanisme, sa compétence hors normes et sa capacité d'écoute exceptionnelle. Il a su m'apporter le soulagement tant attendu, et renforcer ma passion pour la recherche appliquée au domaine médical.

Ce travail n'aurait pu être mené à bien sans l'aide de différents financeurs, en outre, l'agence nationale de la recherche (ANR) à travers le programme "Investissements d'avenir" (ANR-10-AIRT-05), les projets Virtual Studio et Smart Support Center (SSC) FEFER EU, le projet Hydda ainsi que l'organisation EIT Health et l'Institut Carnot LSI.

Merci infinement à tous!

Grenoble, 28 September 2020 S.G.

Preface

This thesis is the result of the research conducted to pursue a Ph.D. in Computer Science from the University of Grenoble Alpes. It took place mainly in the ERODS team (Efficient and RObust Distributed Systems) at Laboratoire d'Informatique de Grenoble (LIG) alongside in AGEIS team at Grenoble Faculty of Medicine. The Ph.D research activities have been co-supervised by Prof. Noël De Palma (ERODS) and Nicolas Vuillerme, Ph.D., HDR (AGEIS).

This thesis introduces a novel scalable and component-based deep learning parallelism platform with a particular application on convolutional neural networks for medical imaging segmentation. This work led to the following publications:

- Automating CNN Parallelism with Components [67], *(2019 IEEE International Conference on Computational Science and Computational Intelligence (CSCI-ISHI) at: Las Vegas, Nevada, USA)(DOI: 10.1109/CSCI49370.2019.00179).*

- Auto-CNNp: a component-based framework for automating CNN parallelism [66], *(IEEE BigData 2019 PEASH at Los Angeles, CA, USA)(DOI: 10.1109/BigData47090.2019.9006175).*

- R2D2: A scalable deep learning toolkit for medical imaging segmentation [65], *Software: Practice and Experience* (DOI:10.1002/spe.2878).

- Variability and reproducibility in neural network for medical imaging segmentation [160], *scientific report* (DOI: 10.1038/s41598-020-69920-0).

The source code of the software solutions developed throughout this thesis were protected via a couple of APP (Agency for the Protection of Programs) copyright deposits with the Université Grenoble Alpes (UGA) as a depositor :

- R2D2 (Rapid & Robust Digital Diagnostic) APP deposit

- Auto-CNNp APP deposit

This thesis has also a potential economic-spin off as it was selected as a laureate of the *Out of Labs* researchers competition category of the *Linksium* incubator.
Grenoble, September 2020 S.G.

Abstract

Deep neural networks (DNNs) and particularly convolutional neural networks (CNNs) trained on large datasets are getting great success across a plethora of paramount applications. It has been providing powerful solutions and revolutionizing medicine, particularly, in the medical image analysis field. However, deep learning field comes up with multiple challenges: (1) training Convolutional Neural Networks (CNNs) is a computationally intensive and time-consuming task (2) introducing parallelism to CNNs in practice is a tedious, repetitive and error-prone process and (3) there is currently no broad study of the generalizability and the reproducibility of the CNN parallelism techniques on concrete medical imaging segmentation applications.

Within this context, the present PhD thesis aims to tackle the aforementioned challenges. To achieve this goal, we conceived, implemented and validated an all-in-one scalable and component-based deep learning parallelism platform for medical imaging segmentation. First, we introduce R2D2, an end-to-end scalable deep learning toolkit for medical imaging segmentation. R2D2 proposes a set of new distributed versions of widely-used deep learning architectures (FCN and U-Net) in order to speed up building new distributive deep learning models and reduce the gap between researchers and talent-intensive deep learning. Next, this thesis also introduces Auto-CNNp, a component-based software framework to automate CNN parallelism throughout encapsulating and hiding typical CNNs parallelization routine tasks within a backbone structure while being extensible for user-specific customization. The evaluation results of our proposed automated component-based approach are promising. It shows that a significant speedup in the CNN parallelization task has been achieved to the detriment of a negligible framework execution time, compared to the manual parallelization strategy.

The previously introduced couple of software solutions (R2D2 and Auto-CNNp) at our disposal led us to conduct a thorough and practical analysis of the generalizability of the CNN parallelism techniques to the imaging segmentation applications. Concurrently, we perform an in-depth literature review aiming to identify the sources of variability and study reproducibility issues of deep learning training process for particular CNNs training configurations applied for medical imaging segmentation. We also draw a set of good practices recommendations aiming to alleviate the aforementioned reproducibility issues for medical imaging segmentation DNNs training process. Finally, we make a number of observations based on a broad analysis of the results of the already conducted CNN parallelism experimental study which led us to propose a guideline and recommendations for scaling up CNNs for segmentation applications. We succeeded to eliminate the accuracy loss with scale for the U-Net CNN architecture and alleviate the accuracy degradation for the FCN CNN architecture.

Key words: Deep learning, software engineering, distributed optimization, distributed systems, high performance computing, medical imaging, semantic segmentation,

Résumé

Les réseaux neuronaux profonds (DNNs), et plus particulièrement les réseaux neuronaux convolutifs (CNN) entraînés sur des grandes quantités de données, rencontrent un vif succès dans une multitude d'applications capitales, et particulièrement en imageries médicales. Cependant, l'entraînement de réseaux de neurones convolutifs (CNN) (1) est une tâche chronophage. De plus, (2) distribuer l'entraînement des CNNs est un défi ardu en pratique car il s'agit d'un processus fastidieux, répétitif et sujet aux erreurs. En outre, (3) il n'y a actuellement aucune étude approfondie sur la généralisation et la reproductibilité des techniques de parallélisation des CNNs particulièrement sur des applications concrètes de segmentation en imagerie médicale.

Dans ce contexte, cette thèse vise à relever les défis susmentionnés. Pour cela, nous avons conçu, implémenté et validé une plateforme d'apprentissage profond à base de composants qui passe à l'échelle pour la segmentation en imagerie médicale. Au début, on introduit R2D2, une boîte à outils d'apprentissage profond de bout en bout qui passe à l'échelle. En effet, R2D2 introduit également un ensemble de nouvelles versions distribuées d'architectures d'apprentissage profond populaires afin d'accélérer l'entraînement effectif des modèles CNNs innovants dans des délais raisonnables et réduire l'écart entre les chercheurs et l'apprentissage en profondeur exigeant des compétences accrues. En outre, cette thèse introduit également Auto-CNNp, un nouveau framework basé sur les composants logiciels pour automatiser la parallélisation des CNNs en encapsulant et en cachant les tâches de routine de parallélisation au sein d'une structure de base tout en gardant la solution logicielle suffisamment flexible et extensible pour une personnalisation spécifique à l'utilisateur. Les résultats de l'évaluation de notre approche automatisée basée sur les composants sont prometteurs. Ils montrent qu'une accélération significative de la tâche de parallélisation CNN a été réalisée au détriment d'un temps d'exécution du framework négligeable, par rapport au temps nécessaire à la stratégie de parallélisation manuelle.

Le couple de solutions logicielles précédemment introduites (R2D2 et Auto-CNNp) nous ont donné les outils appropriés pour effectuer une analyse expérimentale approfondie afin d'étudier la généralisation des techniques de parallélisation des CNNs vers la tâche de segmentation. Simultanément, nous avons mené une revue de littérature visant à étudier les sources de la reproductibilité dans l'entraînement des modèles d'apprentissage profond pour une configuration d'entraînement particulière de segmentation en imagerie médicale. Nous proposons également quelques recommandations de bonnes pratiques afin d'atténuer ces problèmes précités de reproductibilité d'entraînement des DNNs pour la segmentation en imagerie médicale. Enfin, nous faisons un certain nombre d'observations en nous basant sur une analyse approfondies des résultats de l'étude expérimentale déjà menée sur le parallélisation des CNNs, qui nous ont permis de proposer des directives et des recommandations pour distribuer l'entraînement des CNNs pour une segmentation sans perte de précision.

Mots clefs : Apprentissage Profond, Génie Logiciel, Optimisation Distribuée, Systèmes Distribués, Calcul haute performance ,Imagerie Médicale, Segmentation Sémantique

Contents

List of Figures

List of Tables

1 Introduction

1.1 Context

Deep learning is a subfield of machine learning where machines are able to learn without being explicitly programmed. Deep learning field relies on a network of artificial neurons inspired by the human brain. Indeed, it has gained a significant and unprecedented popularity in recent years. As can be seen in Figure 1.2, the number of citations related to deep learning topic which have been identified by the Web of Science research tool [44] during the last decade are increasing at an exponential rate. These recent deep learning breakthroughs have been achieved thanks to the development of computing power that is available nowadays and the increased availability of big data. Indeed, deep learning has been outperforming classical machine learning techniques across a wide range of relevant fields of applications such as speech recognition [62], video processing [41] and many other domains including medical image analysis domain [64] and specifically the medical imagining semantic segmentation applications.

1.1.1 Semantic segmentation in medical imaging

Semantic segmentation of medical imaging involves detecting and contouring boundaries of regions of interest in medical images like lesions, anatomical structures, or any other meaningful morphological structures [175]. It consists in a pixel level classification of images, (i.e. by assigning a label to every pixel in every image, we can split input images into semantically meaningful regions [175]). Semantic segmentation plays a fundamental role in in computer aided diagnosis [176, 163], clinical studies and medical treatment planning [173]. However, manual medical image segmentation is not only a tedious, extensive and time consuming task, but also it has to be performed by medical experts. Recent advances in deep neural networks [114] (DNNs) and particularly convolutional neural networks [114] (CNNs) come to address this issue. In fact, CNN-based applications are revolutionizing medicine. They have been shown to be powerful tools to successfully tackle most common medical images challenges [173, 64] and in particular medical semantic segmentation tasks [138, 174].

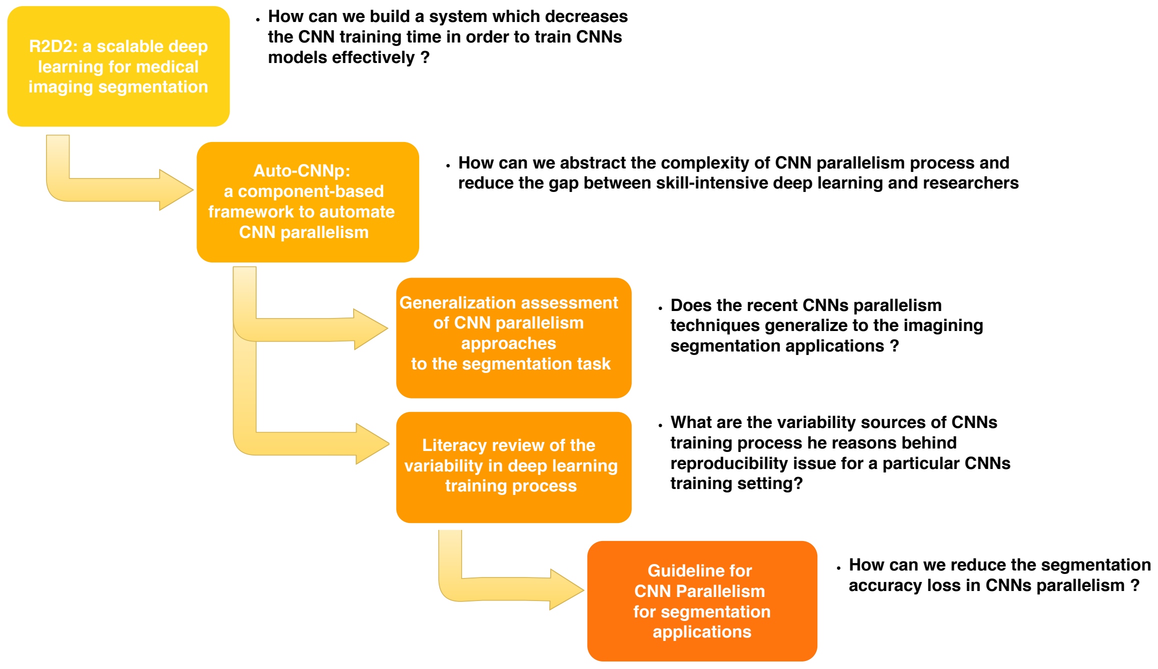

R2D2: a scalable deep learning for medical imaging segmentation
• How can we build a system which decreases the CNN training time in order to train CNNs models effectively ?
Auto-CNNp: a component-based framework to automate CNN parallelism
• How can we abstract the complexity of CNN parallelism process and reduce the gap between skill-intensive deep learning and researchers
Generalization assessment of CNN parallelism approaches to the segmentation task
• Does the recent CNNs parallelism techniques generalize to the imaginig segmentation applications ?
Literacy review of the variability in deep learning training process
• What are the variability sources of CNNs training process he reasons behind reproducibility issue for a particular CNNs training setting?
Guideline for CNN Parallelism for segmentation applications
• How can we reduce the segmentation accuracy loss in CNNs parallelism ?

Figure 1.1 – An overview of the main objectives of the PhD thesis alongside with research questions addressed accordingly in each study.

1.1.2 Problems description & Thesis Goals

Figure 1.1 provides an overview of the main goals of this thesis and outlines the corresponding research questions tackled in each study with respect to their related chronological order.

1.1.2.1 "How can we build a system which decreases the CNN training time in order to train CNNs models effectively ?"

It is the first research question we aim to address in this thesis. Indeed, although convolutional neural networks (CNNs) based medical applications have been providing powerful solutions, efficiently training of CNNs models is a tedious and challenging task. It is a computationally intensive process taking long time which represents a significant hindrance to scientific research progress. In classic CNNs, the model is defined by a huge number of parameters (i.e., typically in range of millions [107]) and requires a considerable volume of data and long time (e.g., approximately 21 day to train *GoogleNet* [86] on ImageNet [1] dataset using a single GPU), in order to adequately tune these parameters and train the CNN model effectively. Hence, decreasing the training duration of CNNs throughout scaling up the training process has become one of the most active areas of research making deep learning converge to high performance computing (HPC) problems. These observations inspired us to try to address the aforementioned issue by introducing R2D2, a novel scalable deep learning toolkit dedicated

[1]http://www.image-net.org/

for medical imaging segmentation aiming to decrease the training duration and effectively train CNN models.

1.1.2.2 How can we abstract the complexity of CNN parallelism process and reduce the gap between skill-intensive deep learning and researchers ?

It is the second research question we intend to study in this thesis. This research question stems and arises directly as a result of the previous research question (subsubsection 1.1.2.1). Indeed, introducing parallelism to CNNs is a challenging task in practice. It is a tedious, repetitive and error-prone process. Moreover, scaling up CNNs training process in practice requires a considerable practical mastery of both deep learning and distributed optimization techniques. The aforementioned problems which we stated while building R2D2 led us to realize the importance and the need for not only automating routine tasks to avoid duplication of effort while scaling up CNNs training, but also to adopt a component reuse approach while considering the software extensibility principle. Hence, we intent to introduce also in this thesis, Auto-CNNp, a component-based framework to automate CNN parallelism.

1.1.2.3 Does the recent CNNs parallelism techniques generalize to the imagining segmentation applications ? What are the sources of variability of CNNs training process and the reasons behind reproducibility issue for a particular CNNs training setting ?

Unfortunately there is currently (1) no broad study of the generalizability of the last published works [61, 183, 121, 208, 82] to the CNN parallelism for the segmentation tasks neither (2) a study of the variability in the deep learning training task. The previously introduced couple of research questions (subsubsection 1.1.2.1, subsubsection 1.1.2.2) led us to build two software solutions (R2D2 and Auto-CNNp). The latters led us to go one step further by conducting a thorough and practical analysis of the generalizability of the CNN parallelism techniques to the imaging segmentation applications. Concurrently, this thesis aims to analyse and identify the sources of variability in deep learning training process throughout an in-depth literature review in order to better understand the challenges and the issues of the reproducibility for a particular CNNs training configuration applied for medical image segmentation before proposing a set of good practices recommendations aiming to alleviate the identified reproducibility issues for medical imagining segmentation DNNs training process.

1.1.2.4 How can we reduce the segmentation accuracy loss in CNNs parallelism ?

Finally, based on a thorough analysis of the results of all the already conducted CNN parallelism experimental study we make a number of findings which led us to propose a guideline and recommendations for scaling up CNNs for segmentation applications without accuracy loss.

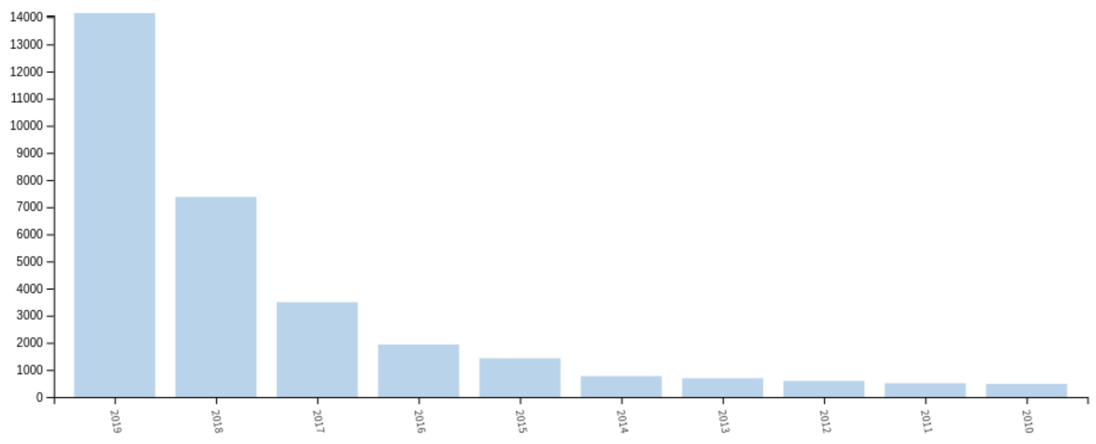


Figure 1.2 – Deep learning topic citations number evolution per year during the last decade

In summary, in this thesis we propose an ultimate, all-in-one, integrated scalable and component-based parallelism platform with a particular focus on medical imagining segmentation applications. We believe that gathering R2D2, Auto-CNNp and our introduced parallelism guideline can offer an interesting technological platform to address the aforementioned challenges and reduce the gap between skill-intensive deep learning and researchers for a better understanding of the emerging deep learning field and paradigms.

1.2 Thesis Contributions

The significant contributions of this thesis are fourfold. They can be summarized as follows:

1. Firstly, we introduce R2D2, a scalable deep learning toolkit with the goal to assist health professionals in the medical imaging analysis field to automatically identify pathologies with more precision using deep-learning-based approaches throughout an intuitive end-to-end medical imaging processing pipeline. R2D2 introduces as well a set of a novel distributed versions of widely-used deep learning architectures in order to speed up building new cutting-edge deep learning models. Furthermore, R2D2 empowered us with the suitable tool to perform an thorough and practical experimental analysis to study the generalization of the CNN parallelism techniques to the segmentation task.

2. Secondly, we introduce Auto-CNNp, a novel component-based software framework to automate CNN parallelism. Auto-CNNp aims to streamline routine tasks throughout (1) capturing cumbersome CNNs parallelization tasks within a backbone structure while (2) keeping the framework flexible enough and extensible for user-specific personalization.

3. We perform an in-depth literature review aiming to identify and study the sources of

variability in deep learning training process with the goal to better understand the challenges and the issues of the reproducibility of neural networks training task for medical image segmentation. We also propose some recommendations in order to alleviate these issues.

4. Finally, based on a broad analysis of the already conducted CNN parallelism experimental studies results, we make a number of observations which enabled us to propose a guideline with the goal of providing researches with enough information and recommendations for scaling up CNNs for segmentation applications with a minimum possible accuracy loss.

1.3 Thesis Organisation

The rest of this thesis is organised as follows:

- **Chapter 2** provides state of the art and background information on artificial neural networks and demystifies distributed deep learning training strategies. Also, this chapter presents our evaluation environment and case studies. The presented concepts and approaches are used throughout this manuscript. Our contributions are compared with their related work in their own chapters.
- **Chapter 3** presents the first building block of our proposed platform which we denominate R2D2. The latter is a scalable deep learning toolkit with a particular focus on medical imaging segmentation. We evaluated R2D2 on a couple of the introduced medical imagining use cases. Concurrently, in this chapter we conducted an assessment of the generalization of recent CNN parallelism approaches to the segmentation task. This chapter looks alike a revised version of an article introducing R2D2 toolkit which is under review for publication in the Software: Practice and Experience (SPE) journal.
- **Chapter 4** introduces Auto-CNNp, our proposed component-based framework for automating CNN parallelism. We conduct a comprehensive assessment of our proposal on a couple of medical imaging segmentation case studies. This chapter is very similar to our accepted for publication articles in the IEEE BigData PEASH'19 workshop and the IEEE CSCI'2019 international conference.
- **Chapter 5.2** studies the variability and reproducibility of CNN parallelism for medical imagining segmentation applications. A revised version of an article is under review for publication in an international peer-reviewed journal (Scientific Reports).
- **Chapter 6** presents our proposed guideline aiming to help researchers towards bringing CNN parallelism into practice without accuracy loss for imagining segmentation applications.
- **Chapter 7** concludes this thesis and presents some future research directions that we believe are worth investigating.

2 State of the Art & Background

This chapter overviews the state of the art and provides the necessary background related to the next chapters of this thesis. In section 2.1, we introduce deep neural networks key abstractions with a particular focus on convolutional neural networks (CNNs). We then review the main approaches for distributed training of deep neural networks in section 2.2. Finally, we present the evaluation use cases alongside the assessment environment for the different buildings blocks of our proposed scalable and component-based deep learning parallelism platform.

2.1 Deep Neural Networks

In this section, we outline the basic concepts in deep artificial neural networks and we explain the way that these networks operate, before diving in particularly widely-used convolutional neural networks architectures.

2.1.1 Artificial Neural Networks

As the name suggests, artificial neural networks are inspired by the brain operating mechanism. Its computation system is analogous to biological networks of neurons. Actually, human brain is made up of millions of interconnected neurons that exchange electrical impulses through synapses which enables us to make sense and learn new concepts [109]. How these artificial neural networks work will be explained in the following.

2.1.1.1 Artificial Neural Networks principles

The core building blocks of artificial neural networks are neurons. Perceptron [162] is one of the first and simplest artificial neural networks which was proposed by Frank Rosenblatt in 1957. It is a single layer binary classifier with good performances in classifying linearly separable problems. Giving that linear classifiers are limited in their complexity to represent

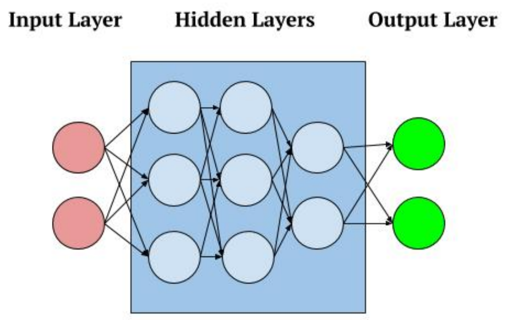


Figure 2.1 – Neural network architecture

non trivial problems [139], artificial neural networks with layered architecture were introduced [83] in 1965 [88, 170]. As shown in Figure 2.1, we can recognize three types of layers: the *input layer*, the *hidden layer*, and the *output layer*. The input layer takes the raw input data. There are one or more optional hidden layers which are all the layers between the input layer and the output layer. The neurons in the hidden layers (represented by circles in Figure 2.1) compute a weighted sums of all of their inputs, add a constant, known as the bias, and feed these sums through a non linear activation function to the next layer (either a hidden layer or the output layer if it is the last one).

This forward movement of calculations is done starting from the input layer to the output layer which is a representation of the output passing by hidden layers. These calculations are known as *forward propagating*. Once this forward movement of calculations is finished, the output is compared with the expected value and we calculate the gradient of the loss function. Next, the *backpropagation optimization* method is performed using gradient descent or any other minimization strategies [116]. The gradient descent is an iterative minimization approach that uses search directions to find a local minimum of a function (i.e. the loss function during the backward propagation of errors). The search starts at an arbitrary starting point and then takes a series of steps towards the minimal downhill, i.e. the direction opposite to the gradient, and ideally do not stuck in saddle points [115]. The gradient descent optimization strategy has multiple varieties such as stochastic gradient descent (SGD) [114], AdaGrad [43], RMSProp [195], AdaDelta [213], and Adam [103].

Several iterations of this process, i.e the forward propagation and the backpropagation, is repeated on the entire training data, in order to minimize the loss function and adjust DNN parameters. The final valid settings of parameters is used later for predictions. The multiple processing layers in the artificial neural networks differ according to the hierarchical representation levels of the features learned by the neurons at the same layer [214]. In the image classification task for example, initial hidden layers which are close to the input layer, recognize low abstraction level features (e.g. edges, motifs and colors). Once we navigate to deeper layers, high level features like familiar objects (e.g. doors, windows) might be detected [169, 114, 116]. The neural network architecture introduced in Figure 2.1 is called a fully-connected neural network because each neuron is connected to every neurons of the previous layer [113]. Moreover, deeper networks with additional hidden networks can efficiently learn and capture more complexes patterns. Machine learning methods that learn hierarchical representations in data using deep artificial neural networks are known as deep learning [11, 116, 56].

2.1.1.2 Activation Function

Activation function is a crucial feature during the learning phase of artificial neural networks in order to solve complex nonlinear problems. It's a scalar-to-scalar function that converts the inputs of a neuron into an output signal called «unit's activation level». The latter will decide if a neuron should be activated or not depending on the relevance of its inputs for the learning process. Several activation functions exist (e.g. Sigmoid, Tanh [100], ReLu [143]) and their two main common properties are nonlinearity and differentiability. In fact, giving the limitations of linear functions, more robust functions are needed to modelize more complex issues. Hence, non-linear activation functions are used in order to introduce nonlinearity to models. Moreover, considering that need to calculate the gradient of the loss function during the backpropagation phase [89], the activation function should be differentiable to make backpropagation possible. Finally, since during the calculation of weighted sums, we may come out with numbers in any range, another main reason behind the use of activation functions is their ability to restrain the amplitude of the outputs by squashing them in a certain range [100].

2.1.1.3 Regularization

A core issue in machine learning is when a model performs well on training data but it has poor performances when it faces new inputs (i.e. during test phase for instance). This problem is known as *overfitting* [6, 56, 168] and it occurs when the model is too complex and has too many degrees of freedom that he will be unable to learn by making generalizations of new concepts. Fortunately, a significant amount of research has been done to come up with strategies to solve this issue. These strategies are called *regularizations* techniques [56].

The most widely-used regularization techniques are the L1 and L2 methods and data augmentation and dropout techniques [56]. The L1 and L2 methods [145, 55] penalize high value network parameters by adding a *regularization term* in order to simplify the model and make it less sensitive to overfitting. More details on dropout and data augmentation regularization approaches will be given below.

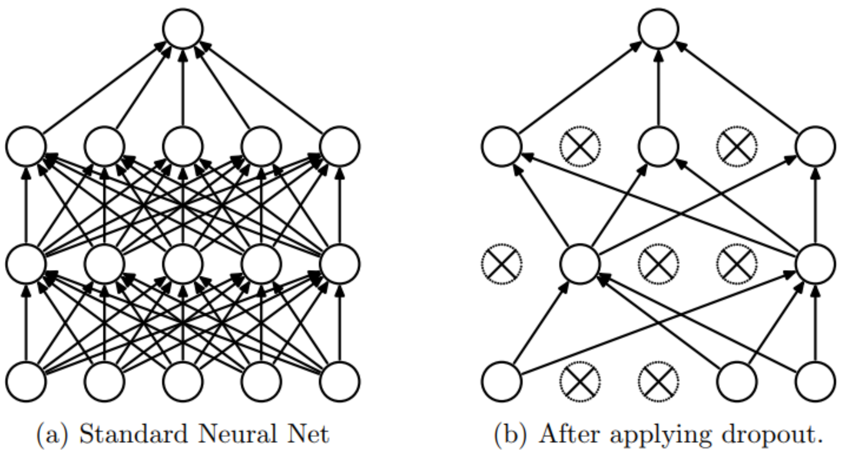


Figure 2.2 – Neural network before (a) and after (b) applying dropout regularization technique [185].

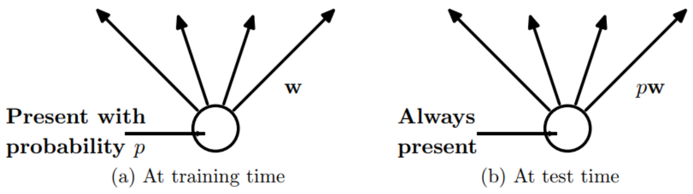


Figure 2.3 – **Left:** A neuron with a probability p of presence at training time. **Right:** A neuron always present during test phase [185].

Dropout

Dropout is a regularization method for neural networks introduced by Srivastava, et al. [185] in 2014. It is an efficient technique to prevent complex co-adaptations of neurons on training data and to reduce the complexity of the network. This leads to a better generalization of the model and hence prevent overfitting [56]. As shown in Figure 2.2, the dropout regularization technique randomly omits a set of neurons along with their corresponding activations with a certain probability (p). A dropout probability of 50% for the hidden layers while keeping all the input neurons has been proved to perform well on a wide range of tasks [185]. The dropout method is applied only during the training phase to force our model to learn the same patterns with different configurations of neurons. Therefore, the model will be less prone to overfit the training data. It is not applied during test time (i.e. we do not ignore any neurons). Otherwise, as can be seen in Figure 2.3, the output of each neuron is scaled by the dropout probability (p) [185].

Data augmentation

One technique to bypass overfitting consists in perform the training of the model on bigger training datasets in order to forbid the model from memorization the networks parameters rather than learning new concepts by generalization. This method is known as the data augmentation technique [107, 131]. When it is performed on images for CNNs training it consists in increasing the training dataset by adding new images derived from the original dataset (i.e. generally by introducing rotations, cropping and distortions to the initial dataset).

Weight decay approach

The weight decay [56, 108] is a widely-used regularisation technique adopted not only to improve the model generalization and hence prevent it from overfitting, but also to achieve a faster convergence of the model training process and a better overall performance. Indeed, the weight decay strategy which is also know as weight regularization technique is an effective alternative especially in the context where large training datasets are rare and hard to acquire such as deep-learning-based medical applications [56]. The weight decay strategy aims to control the growth of the DNNs weights by also adding a regularization term to the loss function.

2.1.1.4 Terminology

For the reminder of this manuscript, the term 'CNN architecture' refers to the global structure of the neural network (i.e., the number, order, size, etc., of each network's layer). The term 'hyperparameter' refers to a variable which is required to be defined before the CNN training task begins. Also, the term 'CNN model' denotes the output of the training process of a specific CNN architecture on a particular training dataset and hyperparameters. The word 'epoch' denotes a single cycle through the full training dataset.

2.1.1.5 Deep Learning architectures

The main deep learning architectures are, restricted Boltzmann machines (RBM) [79], deep Boltzmann machines (DBMs) [166], deep belief networks (DBNs) [80], autoencoders (AEs) [200, 201], recurrent neural networks (RNNs) [63], and convolutional neural networks (CNNs) [107, 116, 102]. Since convolutional neural networks (CNNs) are the most widely-used and successful architectures in computer vision tasks [179, 191] especially in medical imaging field [161, 176, 99, 72], the remaining of this thesis, focuses only on deep convolutional neural networks methods.

2.1.2 Convolutional Neural Networks

As with artificial neural networks, convolutional neural networks (CNNs) are inspired by nature, in particular, visual cortex structure [84, 36]. They are a powerful multilayer neural networks designed to deal with images. Actually, if we use the fully-connected layers network architecture, described in section 2.1.1.1, to work with images, we will end up with millions of parameters to tune in order to train our model, to potentially caption patterns in images [113]. A gray scale image for instance, is a 2D matrix. Feeding it as an input to a fully-connected network will lead to an exploding number of network connections and weights [33, 116]. Hence, training these fully-connected networks models would be impractical even using the highest performing hardware like GPUs [148, 167]. CNNs address this computational problem using some pretty basic ideas applied in a clever way. Actually, unlike fully-connected networks where every neuron is connected to all its predecessors, each neuron in convolutional neural networks is connected to a local subdivision of the neurons in the underlying layer. This size of the region formed by this subset of neurons is a CNN hyperparameter [184, 158] known as the *Receptive Field* [32]. The limited range of the latter makes CNNs easier to train because they have much less parameters to tweak than fully-connected networks [107]. Another advantage of CNNs is their ability to introduce some degree of shift, scale and distortion invariance [116, 113] to the learning process. In fact, for image classification task for instance, and since images are a 2D structures, the position of the classified objects in the image can vary a lot. Training a typical fully-connected network to recognize the spatial configurations of objects will lead to an increasing number of parameters and training instance. Due to the replication of parameters and since we need to cover all the combinations and possible variations of objects positions during the training process [116, 113]. CNNs address this issue by forcing the extracting of local features by limiting the receptive field of hidden unit to be local [116, 105] and by sharing the same parameters across all neurons in order to recognize the same pattern. This concept is known as *Parameter Sharing* [114].

2.1.2.1 Architecture Overview

The architecture of convolutional neural network can differ according to the types and the numbers of layers included. In a classic CNN architecture, multiple layers are stacked for a specific number of times. How CNNs operate and what are their building components will be further investigated in the following.

Inputs

In CNNs, the initial input layer has the size of the raw input images and holds their pixel values [107, 111]. It has the following dimensions: width (w), height (h) and depth (d). The depth of an image is the number of its color channels. A grayscale image, for instance, has 1 color channel and an RGB image has 3 color channels.

Convolutional Layer

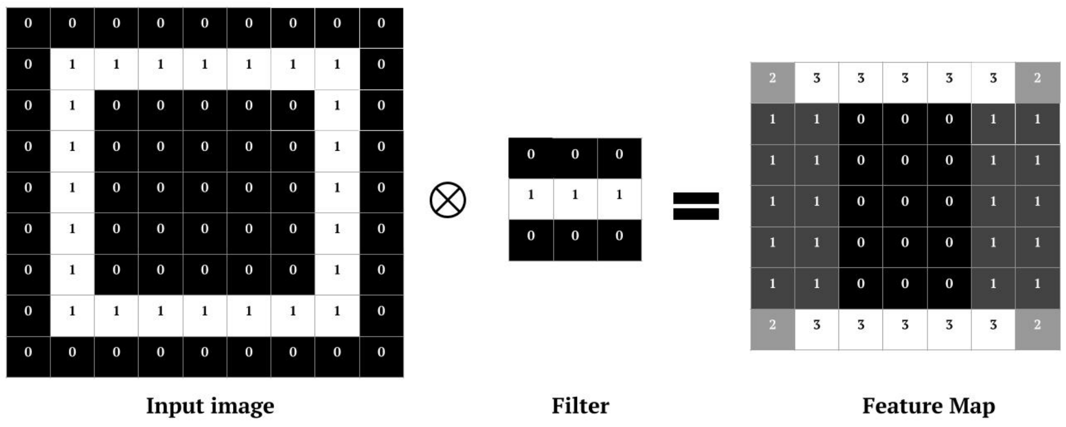


Figure 2.4 – Convolution operation

The convolutional layer is the main building component of a CNN that does the heaviest computational workload [2]. Actually, it relies on the mathematical convolution operation which is a computationally intensive procedure which takes two input functions, f() and g() for instance, and returns a third one (e.g. h). The convolution operation expresses the extent of the overlap between the first function (f) as it is slided over the second one (g) [202].

A convolutional layer has several filters (or kernels) of equal size. Filters have smaller dimensions than the input images. As showing in Figure 2.4 filters are small matrices of weights which are applied to local regions in the input images. The convolution operation in CNNs consists in lining up these filters and the corresponding input images patches (or receptive fields). The next step is to multiply each pixel in the receptive field by the corresponding filter pixel. After that, the filter slides (or convolves) along the input image directions (vertically and horizontally) computing the dot product. The step size with which it advances is a hyperparameter known as the *stride*. This process is repeated with all the filters for all the input images, and the output volumes produced by the convolution operations are known as the feature maps. The convolution operation core objective in CNNs is to extract visual features of the input volume. In fact, convolution aims to search for every possible match between filters and receptive fields. This process will make the CNN learn to recognize some local patterns in input images when the filters get activated every time they identify some type of visual feature like oriented edges. This explains also the reason behind using multiple filter. Actually, by different choices of kernels, different visual features can be identified. Finally, the feature maps are stack along the depth dimension to generate the output volume which constitutes the input of the following layer in CNN architecture [116, 113, 11, 114, 111].

Nonlinear Layer

In the same perspective as for the role of activation functions in neural networks, its common to apply nonlinear layer just after the convolutional layer in CNNs, so as to introduce the powerful modelisation properties of nonlinear functions to our CNN. Previously, some non-linear functions like sigmoid [196] and tanh [17] were adopted before Vinod Nair et al. [143] proposed the ReLU nonlinear activation function. Nowadays, it's the mainly used method in the nonlinear layer in CNNs for two main reasons [56]. First, it decreases the training time while keeping the same model accuracy because it's computational efficient. Second, it permits to avoid the gradient vanishing problem during training time. In CNNs, the ReLU layer changes all the negative values of the input volume to zero. It's formally described by the following formula.

$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$$

Pooling Layer

After introducing non-linearity to the CNN, the next layer is the pooling (or downsampling) layer. Its main role is to progressively reduce the spatial dimensions of the input feature maps. Therefore, this permits to decrease the number of the network parameters, which both reduces the training time and controls overfitting. Downsampling operation is performed using pooling layer on each feature map separately. It reduces their height and width without affecting the depth. Different pooling functions might be used in pooling layer such as max pooling and average pooling. However, the most popular one is the max pooling operation. As can be seen in Figure 2.5, downsampling is performed with a 2x2 size filter applied with a stride of 2. With max pooling filter, the pooling layer keeps the pixels with the maximum values in every input volumes. Whereas, with average pooling filter, it computes the average of every input volumes [107, 116, 113].

Fully Connected Layer

Similar to regular artificial neural networks already seen in subsubsection 2.1.1.1, neurons in fully connected layer are connected to all neurons in the previous layer. Fully connected layer is generally the final layer used to perform classification tasks with CNN. Actually, the value of each neuron in fully connected layer indicates a classification score for a specific class [107, 116].

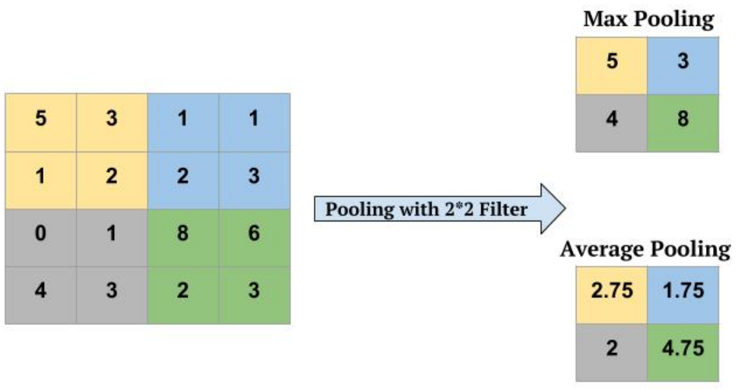


Figure 2.5 – Pooling operation.

2.1.2.2 Classical CNN architectures:

Throughout this section, we will briefly introduce the commonly used CNNs architectures that significantly influenced the current state-of-the-art in the field. As we will notice, most of these architectures share the same general design guidelines fundamentals of stacking repeatedly almost the same building components (layers). Starting with applying convolutional layer to the raw input to extract features followed by nonlinear layer to introducing nonlinearity to the CNN immediately after. The next applied layer is a pooling layer which aims to gradually reduce the dimensional extents of the input feature maps. The main influential architectures became notorious by winner the *ImageNet Large Scale Visual Recognition Competition (ILSVRC)* [164].

LeNet-5

LeNet-5 [116] is the pioneer convolutional neural network architecture which was introduced by Yann Lecun in 1998. This successful first CNN was used to recognize and classify digits in the MNIST database [117, 115]. It was designed specifically for postal services to automatically recognize handwritten digits zip codes.

As can be seen in Figure 2.6, LeNet-5 takes a single channel input image of size 23 x 23, performs convolution with six 5 x 5 kernels with a stride of one, then applies a 2 x 2 max pooling subsampling layer. This convolution-subsampling layers sequence is repeated another time before concluding with two fully connected layers and a final fully connected softmax classification layer of size 10 to output the result. LeNet-5 has a total of 60 850 parameters [116].

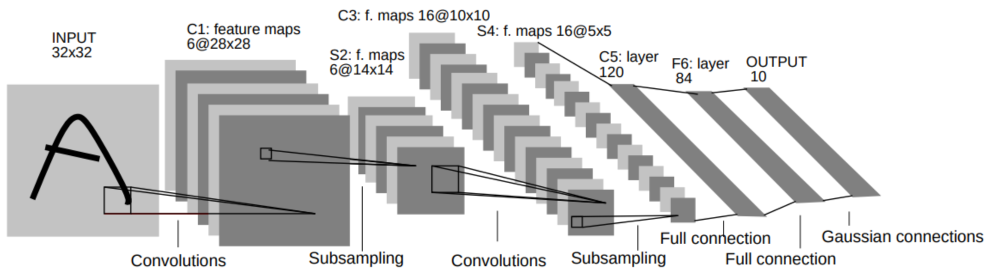


Figure 2.6 – LeNet-5 Architecture [116].

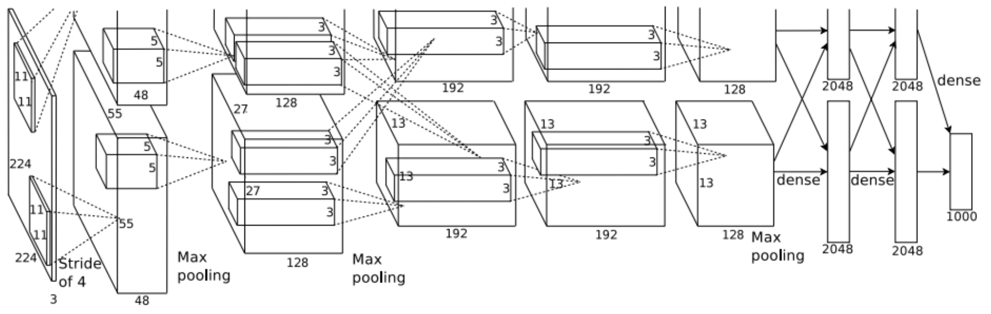


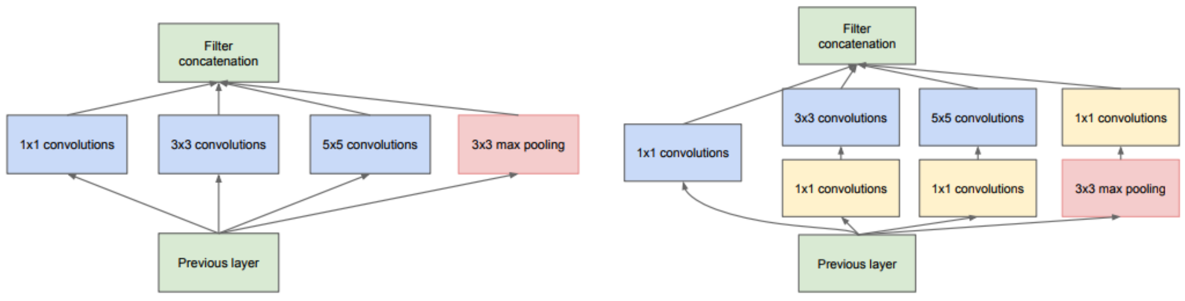
Figure 2.7 – AlexNet Architecture [107].

AlexNet

AlexNet [107] was introduced by Alex Krizhevsky et al. in 2012 and represents one of the major contributing architectures to the recent development of CNNs. This architecture won the 2012 ImageNet [164] competition by largely dominating the other traditional computer vision approaches (a top-5 error of 15.3% with 10.8 % points ahead of the runner up). This major breakthrough draw the attention of the community to the powerful aspect of CNNs, and hence contributed to the development of better CNNs architecture in the future. Actually, as illustrated in Figure 2.7, AlexNet architecture is pretty similar to LeNet-5 design. Although, it is a deeper network with 5 convolutional layers, downsampling layers, dropout layers, and 3 fully connected layers. Moreover, the ReLU [143] activation function was applied to introduce nonlinearity and the dropout [185, 107] regularization technique was performed to avoid overfitting. The dataset size was increased through data augmentation technique [153]. Since AlexNet is a deeper CNN with 60 million parameter, the training has been divided into two streams and was performed using two GTX 580 GPUs in order to decrease the execution time of the computationally expensive training process.

GoogLeNet

GoogLeNet [191] was developed by Szegedy et al. in 2014. This CNN introduced by researchers



(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 2.8 – Inception module [191].

at google was basically inspired from AlexNet and won the 2014 ILSVRC classification and detection challenges with a top 5 error rate of 6.7%. GoogLeNet has 22 layers and introduced a novel concept known as *the inception module* which has considerably decreased the parameters number in the network (4 million parameters which are 12 times less number of parameters than AlexNet).

The *inception modules* are inspired by the Network-in-Network architecture [125] and are stacked multiple times on top of each other throughout the network. As can bee seen in Figure 2.8, the *inception module* consists to apply a multi-scale parallel sequence of convolutions before aggregating the results at the end. Actually, applying convolutions at different scales makes GoogLeNet able to extract multi-scale features as well. The inception module contains also a 1 x 1 convolutions which are performed in order to reduce the inception module channel depth and hence save computations. There are multiple follow-up versions of the inception module [190, 192].

ResNet

The Deep Residual Network (ResNet) is a CNN which was introduced by Kaiming He et al. [75] from Microsoft Research in 2015. ResNet is an ultra deep CNN (152 layers) which won the ILSVRC 2015 challenge [164] with a top-5 error rate of 3.57%. Actually, increase the depth of the network by adding naively more layer will lead to worse results. This phenomena is known as degradation problem. ResNet architecture moved the depth limits of previously introduced CNNs even further by introducing *residual block* which solves the degradation problem [75].

The idea behind a *residual block* is illustrated in Figure 2.9. In fact, in addition to the classic calculations path followed by the input of a certain layer, the residual block offers *skip connection* which permits to bypass the classic path and pass the information directly to the output

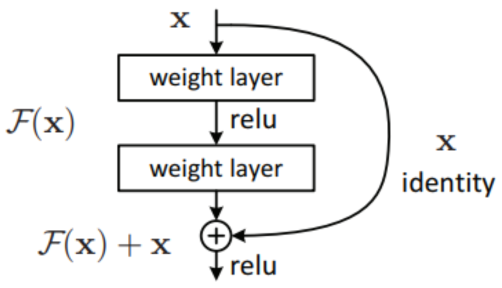

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2.9 – Residual Block [75].

of the next layer. This process reduces the information loss during the gradient computations and hence solves the vanishing gradient problem. Moreover, since the information is kept throughout the calculations, residual blocks offer a resilience to layer deletion.

2.2 Distributed training of deep neural networks

After presenting in section 2.1 an overview of the state-of-the-art in deep neural networks with a focus on convolutional neural networks, we briefly introduce throughout this section, some background information on distributed training approaches of deep neural networks. These methods are mainly divided into three different categories: data, model and hybrid parallelism techniques.

2.2.1 CNN parallelism approaches

Distributed training approaches of DNNs are mainly divided into three different categories: model, data and hybrid parallelism techniques.

2.2.1.1 Model Parallelism

Some DNNs models have a considerable size, and hence, they are not adapted to the memory size of an individual training device (one GPU for instance) [206]. These models require to be partitioned across all the nodes in the distributed system and every node trains a different part of the model on the whole training dataset. In Figure 2.10, the blue rectangle stands for a training node, the light blue circle represents a subset of a DNN model (e.g., a DNN layer) and the arrow represents a connection between two successive subsets of the same model. As can be seen, every node performs the training of only a specific subset of the model. This parallelization schema is known as *model parallelism* technique [112, 39, 10].

Since the DNN model needs to be split across several nodes, the synchronization of computa-

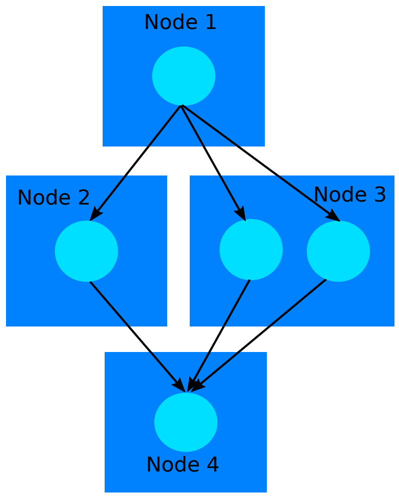


Figure 2.10 – Model parallelism strategy.

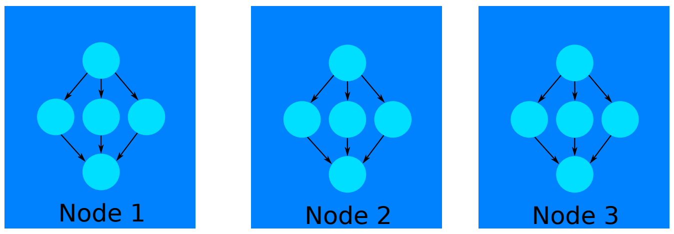


Figure 2.11 – Data parallelism strategy.

tions during the training phase creates a communication overhead which in turn increases the training runtime. Moreover, ensuring fault tolerance for the distributed system and the implementation of model parallelism are a challenging tasks in model parallelism approach [10, 39]. That is why model parallelism is rarely used in practice and will not investigated further in this document.

2.2.1.2 Data Parallelism

The second distributed training approach of deep neural networks is called *Data parallelism* approach. As illustrated in Figure 2.11, all nodes in the distributed system have the same complete copy of the model. However, the training is done independently on each node using a different subset of the whole training dataset, at the end of every training iteration, the results of computations from all the nodes are combined using different synchronization approaches [112, 39, 10]. These approaches and the differences between them will be discussed in subsection 2.2.2.

2.2.1.3 Hybrid Parallelism

It is possible to combine both previously mentioned distributed training approaches (i.e. model parallelism for every node and data parallelism across nodes [10]). However, data parallelism has become the most popular [92, 61, 107, 183] distributed training approach for the following reasons :

- The practical simplicity to introduce data parallelism [10] compared to the model parallelism strategy when the model size fits in the training device's memory.
- The recent success achieved by the data parallelism method [61, 183] to considerably increase the minibatch size without significant segmentation accuracy loss.

Therefore, the rest of this background section focuses only on data parallelism approach.

2.2.2 Synchronous vs asynchronous approaches

In recent years, the largest amount of research done to scale up the training of DNNs evolves around methods that aim to parallelize the computation of the gradient during the gradient descent optimization method. Researchers has been mainly focused on two approaches that will be introduced in the following.

2.2.2.1 Synchronous parameters averaging

The synchronous parameters averaging method is the most straightforward and the easiest approach for data-parallelism. It is an iterative method that consists in the following steps: First, we initialize the network parameters (i.e. weights and biases) and we propagate the current version of the parameters to every worker. Next, each worker performs the training of its local model using a different subset from the current mini-batch of training dataset. Once the training is completed, every worker shares its locally computed gradients with other workers directly or indirectly through a central parameter server [122]. Only when all workers finish their calculations and propagate their gradients successfully, we update the model with the average of all the parameters received from each worker. After that, the last updated version of the model is sent to every worker along with their corresponding subset of the next mini-batch. These steps are repeated while there is still more data to process for each mini-batch in every iteration of each epoch [42, 10, 215].

Even if scaling up the training of DNNs can significantly reduce the training runtime, sometimes some factors can introduce an overhead. In fact, a slow network will create a bottleneck and slows the training process [42, 180]. Moreover, a slow device as well will increase the training runtime, due to the synchronous nature of this approach which forces all workers to wait for updates from other workers in order to move to next iteration. Furthermore, this leads to one crucial question. How often should we average the parameters ? If we average the network's parameters after every iteration, the communication overhead will be high and will submerge the extra time gain we earned from scaling up the training. And if we average the network's parameters rarely, the local parameters in every worker can diverge a lot. Some preface research has been done to find a trade-off between averaging frequency and the model accuracy . For instance, Hang Su et al. [189] propose an averaging period of 10 to 20 mini-batchs to accelerate the training runtime while keeping a reasonably good results [42, 10]. Another approach to decrease the communication overhead suggests to the mini-batch size as much as possible to decrease the number of iterations during the training without affecting the accuracy of the model. Facebook researchers suggest a linear scaling rule for the learning rate [61] to solve this issue. This rule consists in multiplying the learning rate by the same factor as the mini-batch size when the latter increases, in order to avoid the accuracy loss during distributing training of DNNs.

2.2.2.2 Asynchronous Stochastic Gradient Descent

There are two major differences between the synchronous and the asynchronous stochastic gradient descent (SGD) approaches. Firstly, in contrary to the synchronous method where workers share the whole parameters with each others after the computation of the gradient, in asynchronous approach only the updates of the training parameters are shared between workers [215, 180]. Secondly, in asynchronous stochastic gradient descent the synchronous update condition is relaxed. In fact, when a worker finishes its computation, it shares its local parameters updates instantaneously with other workers [61, 42, 10] without being forced to wait for them to complete their local computations. The straightforward advantage for this approach is that it increases the throughput of the distributed system [180]. However, it introduces a new additional problem know as *the stale gradient issue [1].* In fact, by the time a worker completes its local computations of gradient, the global parameters might be updated many times by other workers who were faster than that worker [1]. An elementary application of asynchronous SGD leads to a large amount of gradient staleness. For instance, Gupta et al. [70] demonstrate that the average gradient staleness corresponds in general to the number of workers which slows considerably the convergence of the DNN [39]. An amount of research has been done to deal with the gradient staleness issue. Such as, the soft synchronisation technique proposed by Zhang et al. [216] in 2016, or by limiting the staleness effect by postponing faster workers. The latter technique was introduced by Ho et al. [81] in 2013. One last difference between the synchronous and the asynchronous SGD approaches is that the latter is more resilient to machines failure than the synchronous update-based method [39].

2.2.3 Centralized vs Decentralized stochastic gradient descent

As mentioned before in subsubsection 2.2.2.1, when workers share directly their training parameters. These latter are aggregated through a central server known as a *parameter server*

(PS) [122] and illustrated in Figure 2.12. The parameter server used in this centralized approach is not required to be a single physical server in practice. In fact, a unique PS architecture creates a communication congestion [10]. Other alternatives architectures to the unique PS approach have been proposed, such as the hierarchical parameter servers methods introduced by Gupta et al. [70] and Yu el al. [212] in 2016. Another alternatives approaches are used to scale up the training of DNNs consists to get rid of the central parameter servers. For instance, N Iandola et al. [187] proposed in 2015 a decentralized SGD where a peer to peer communication is used to share parameter updates between all the nodes. Moreover, this approach performs multiple compression techniques to the parameter updates vectors (e.g quantization compression applied to a sparse update vectors) in order to decrease the size of network communications. However, these compression techniques create an additional computation overhead on worker nodes. Another decentralized approach known as *the Ring Allreduce* architecture [150, 171, 42]. As illustrated in Figure 2.13, it consists in the following steps: First, each worker node reads its own subset of the current mini-batch. After that, it computes its gradients, and communicates it to its nearby successor on the ring and get in turn the calculated gradients from its predecessor neighbor. In a ring that counts N workers, it takes N-1 communications of gradients between workers, so that every worker receives the required gradients values to compute the updated model. Finally, The Ring Allreduce architecture is bandwidth optimal approach compared to the parameter server architecture as it drastically reduces the communication overhead [53, 171, 42, 10].

As we have seen above, diverse approaches may be adopted in order to parallelize the training process of DNNs. However, distributed training is not without a cost (e.g. synchronization and network communications overhead, distributed training infrastructure setup time, DNN model accuracy loss). In fact, synchronous parameters averaging method achieves better results regarding the accuracy of the DNN models compared to the asynchronous approach, particularly, with a short synchronisation period [216] alongside with a fast network interconnects infrastructure (InfiniBand[1] for instance). On the other hand, asynchronous method further decreases the distributed training runtime but at the expense of DNN model accuracy. Yet, good results may be achieved in practice if gradient staleness issue is properly managed.

2.3 Medical imaging evaluation case studies

2.3.1 CNN architectures for semantic segmentations

2.3.1.1 Fully Convolutional Network CNN architecture

The previously introduced CNN architectures in subsubsection 2.1.2.2 are used for image classification tasks. However, CNNs are having a great success in multiple other use cases including the images semantic segmentation task. The fully convolutional network (FCN) [127]

[1]More informations on the InfiniBand network interconnect can be found at the following link: https://en.wikipedia.org/wiki/InfiniBand

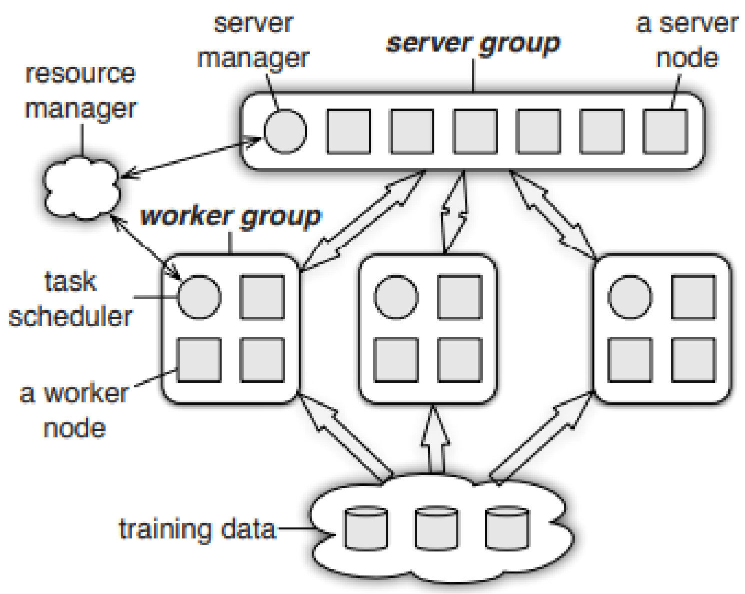


Figure 2.12 – Parameter server architecture [122]

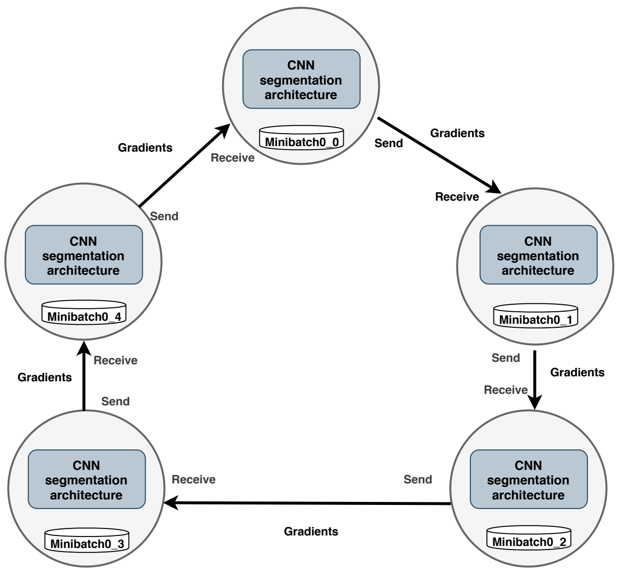


Figure 2.13 – Ring-Allreduce Algorithm [42, 53].

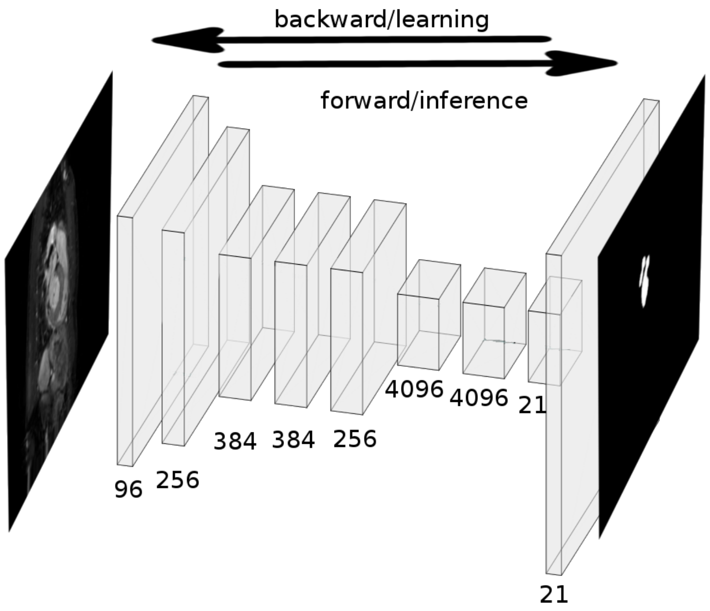


Figure 2.14 – Fully Convolutional Network for semantic segmentation architecture [175]

is one of the most popular CNN architecture for semantic segmentation [2]. It was introduced by Jonathan Long et al. for image semantic segmentation task. As can be seen in Figure 2.14, FCN is a convolutional neural network where the last typical fully connected layer is replaced by an additional convolutional layer which makes the network able to deal with arbitrary-sized input images [175]. Given that the input image dimensions gets smaller when we get deeper in the network due to convolution operations. The FCN uses the transposed convolution [50] technique during the upsampling step so that the output dimensions match the original input image dimensions. However, this technique causes a loss of spatial information. That is why the FCN uses skip connections to reduce the information loss during convolution operations [175, 147].

2.3.1.2 U-Net CNN architecture

Another widely-used[3] CNN architecture for semantic segmentation tasks is the U-Net architecture [161]. It is the second CNN architecture we introduce its distributed version in our proposed solution. In figure Figure 2.15, each blue box denotes a multi-channel feature map. The number of channels is represented on top of the box. White boxes denote copied feature map and the arrows illustrate the diverse operations [161]. U-Net is an encode-decoder style architecture. The encoder consists of a sequence of convolution, max pooling and ReLU activation layers which reduce the spatial dimensions of the input volume. On the other hand, the decoder gradually restores the initial input spatial dimensions through transposed

[2]13927 citations for FCN by the end of January 2020
[3]11291 citations for U-Net by the end of January 2020

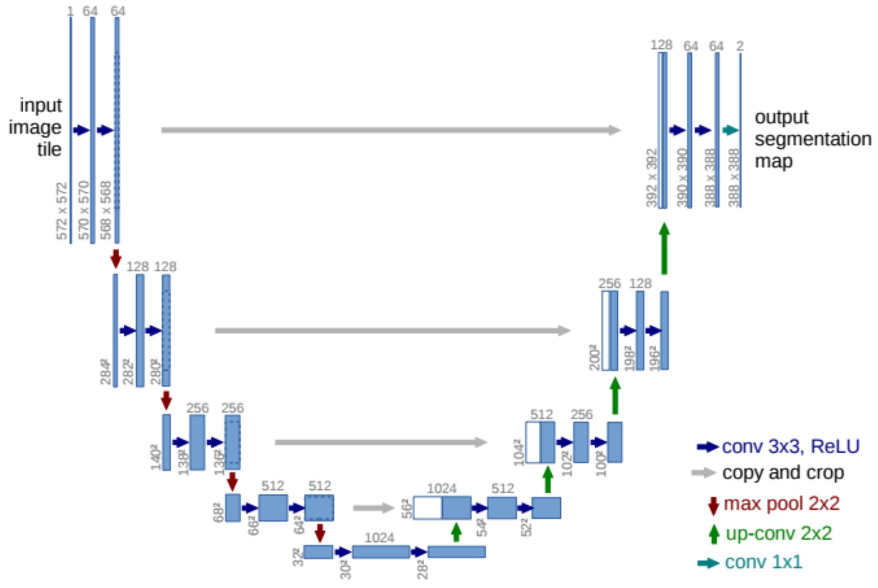


Figure 2.15 – U-Net network architecture [161]

convolution operation [50]. The U-Net architecture might be used in various tasks but it is initially designed and mainly used for biomedical image segmentation [161].

2.3.2 Brain tumor segmentation use case

The first use case we have chosen in order to evaluate our proposed platform is a brain tumor segmentation task; It was proposed during the decathlon medical segmentation challenge [4]. Actually, the brain tumor segmentation involves isolating the different tumor tissues in the brain from healthy ones [87, 219, 60]. It is a crucial and challenging task in medical image analysis because it plays an influential role in early diagnosis of brain tumors which in turn enhance treatment planning and raise the survival rate of the patients [87, 60]. Yet, it is a tedious and time consuming task because it might take hours even if it is manually performed by expert radiologists [87].

2.3.2.1 Dataset:

The dataset has been provided during decathlon medical segmentation challenge for the brain tumors segmentation task. It is a mix of two other datasets that have been initially made publicly available during the *«Multimodal Brain Tumor Segmentation Challenge: MICCAI BRATS [135] 2016 and 2017»*. It contains multimodal MRI scans (i.e., 4D MRI scans [87]) of complex and heterogeneously-located brain tumors that were captured using multiple distinct MRI acquisition protocol [87] from 19 different institutional data contributors [135]. The

[4]More informations on the decathlon segmentation challenge can be found at the following links: http://medicaldecathlon.com/ and https://decathlon.grand-challenge.org/

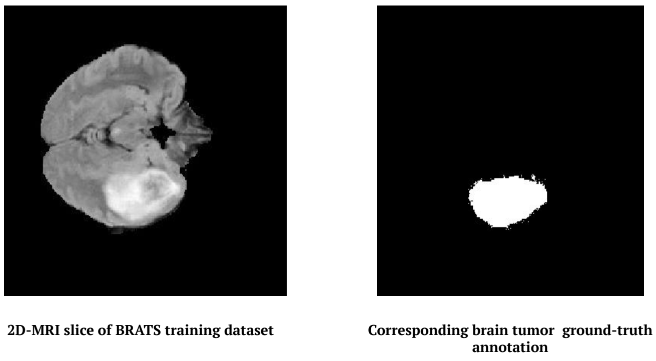

2D-MRI slice of BRATS training dataset
Corresponding brain tumor ground-truth annotation

Figure 2.16 – BRATS MRI scan frame with its corresponding ground truth annotation

BRATS datasets have been initially manually segmented by one to four raters, using the same annotation protocol. After that, the multimodal brain tumor MRI scans along with all their corresponding ground truth labels were manually-reexamined and approved by experienced neurologists [135]. Figure 2.16 shows an example of a brain MRI scan slice containing a tumor and its related annotated MRI scan slice. The final dataset provided by decathlon and used to build our model contains in total 750 annotated MRI scans. It was split into two data subsets. The first partition is a training and validation dataset with 484 annotated MRI scans. The second subset contains 266 annotated MRI scans dedicated to the testing phase.

2.3.2.2 Pre-processing pipeline:

Since decathlon original dataset involves multimodal MRI scans (4D), it was pre-processed in order to extract the corresponding 2D images alongside with their annotations for every MRI scan in the provided dataset. In order to do so, the initial dataset was reduced to T1-weighted MRI scans (3D) [96]. After that, we extracted 70 2D-MRI slices per MRI scan. Therefore, at the end of the pre-processing pipeline, the final training and validation dataset counts in total 33 880 2D-MRI images alongside with their related annotations. This contributes to avoid overfitting without performing data augmentation regularization technique on the training dataset. Also, the same pre-processing pipeline was applied to the testing dataset which counts at the end 18 620 annotated 2D-MRI images.

2.3.3 Left atrium segmentation use case

The left atrial segmentation task was provided by the king's college london university during the left atrial segmentation challenge (LASC) [197]. As illustrated in Figure 2.17, the left atrium segmentation consists in isolating the left atrium body from its surrounding organs structures [197]. It plays a key role during the treatment protocol of patients with atrial fibrillation disease [197] which is the most frequent cardiac electrical disorder provoked by abnormal electrical discharges in the left atrium [24]. Besides that, the latter left atrium segmentation task is also

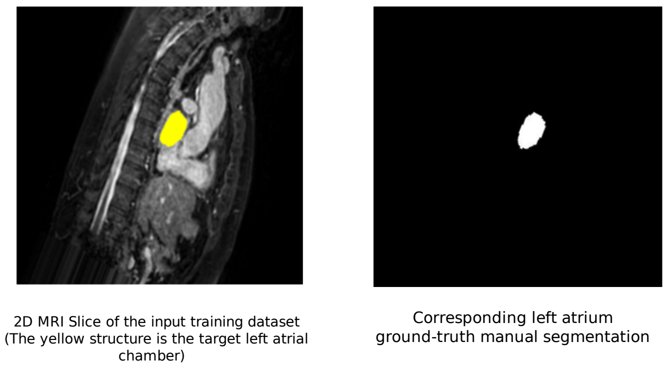

2D MRI Slice of the input training dataset
(The yellow structure is the target left atrial chamber)
Corresponding left atrium ground-truth manual segmentation

Figure 2.17 – Left atrium segmentation training dataset

essential for cardiac biophysical modelling procedure. Nonetheless, the thinness of the atrial wall tissue makes the left atrium segmentation process a challenging task [197].

2.3.3.1 Dataset:

The dataset has been made publicly available by Philips Technologie GmbH, Hamburg, DE, and King's College London during the 2013 LASC challenge [5]. Unlike the BRATS datasets, the left atrium segmentation dataset is a small one with wide quality levels variability as it only includes 30 mono-modal 3D cardiac MRI scans. This will allow us to further assess the transferability of the `Dist-Training` module. The dataset was split such that 20 MRI scans were provided with their corresponding ground truth annotations for the training and the validation steps. The remaining 10 MRI scans were supplied as a test set. The ground-truth masks were initially annotated using automatic model based segmentation. Afterwards, a manual corrections were performed by human experts [197].

2.3.3.2 Pre-processing pipeline:

The pre-processing workflow of the provided datasets involves the following steps: (1) 70 2D-MRI slices were initially extracted from each 3D cardiac mri scan through the `ImgMed`-implemented reshaping operation; (2) A downsampling operation to a size of 512x512 pixels has been carried on each image in order to fit the memory constraint of the GPU (NVIDIA GeForce GTX 1080 Ti); (3) In order to break the curse of small datasets and avoid overfitting, data augmentation technique has been performed on the LASC dataset with rotation, zooming and translation operations as detailed in Table 2.1. Hence, at the end of the pre-processing pipeline, the final training and validation dataset counts in total 35 000 2D-MRI images

[5]The left atrium segementation dataset is available at the following link https://www.cardiacatlas.org/challenges/left-atrium-segmentation-challenge/

Table 2.1 – Data augmentation operations parameters of the left atrium dataset

Operations	Parameters
Rotation	$rotation\_range \, \varepsilon \, [-45, 45]$
Zoom	$zoom\_range \, \varepsilon \, [0.8, 1.2]$
	$(x + shift\_range, y = 0), \; shift\_range \, \varepsilon \, [0, 25.6]$
Translation	$(x = 0, y + shift\_range), \; shift\_range \, \varepsilon \, [0, 25.6]$

alongside with their related annotations.

2.4 Conclusion

In this chapter, we have provided an overview of the required background in order to fit our contributions within their related context. We have also detailed our medical imaging segmentation case studies and the corresponding CNN architectures we have adopted for an in-depth assessment of the main building blocks of our proposed scalable and component-based CNN parallelism platform. Even though a considerable progress has been made recently, the deep learning era remains at its beginning with multiple research questions which have not yet been answered. In the following chapters, we start introducing and detailing separately our four distinct main contributions.

3 R2D2: a scalable deep learning toolkit for medical imaging segmentation

Our first contribution consists in introducing R2D2 [65], a scalable deep learning toolkit for medical imaging segmentation. R2D2 represents the first main building block of our proposed platform. To the best of our knowledge, R2D2 is the first work that aims to tackle the challenge of decreasing the training for CNNs in medial imaging by offering a novel distributed versions of two well-known and widely used CNN segmentation architectures (i.e., FCN and U-Net). We introduce the design and the core building blocks of R2D2. We further present and analyze its experimental evaluation results on two different concrete medical imaging segmentation use cases. R2D2 achieves up to 17.5x and 10.4x speedup than single-node based training of U-Net and FCN respectively with a negligible, though still unexpected segmentation accuracy loss. R2D2 offers not only an empirical evidence and further investigates the latest published works but also it facilitates and significantly reduces the effort required by researchers to quickly prototype and easily discover cutting-edge CNN configurations and architectures.

3.1 Introduction

As outlined in the Introduction Chapter, building efficient CNN models requires an effective and tedious training process. Indeed, multiple CNN architectures have to be investigated. Concurrently, an hyperparameters optimization process [128] has to be performed for every CNN candidate architecture. The hyperparameters optimization task aims to select the optimal set of hyperparameters in order to optimize the CNN performance. It involves performing various hyperparameters optimization strategies [13] which generally require executing multiple training runs. Hence, training CNN models is computationally intensive and time consuming process. For instance training DeepMedic [97] brain tumor segmentation CNN architecture with a particular set of hyperparameters requires approximately a day using a single NVIDIA GTX Titan X GPU. Therefore, decreasing the training duration of DNNs is crucial to accelerate hyper-parameters optimization process. Moreover, it enables researchers to not only build effective CNNs, but also prototype and explore not yet investigated CNN configurations and architectures through an iterative and adaptive experimentation approach.

In order to address this challenge, we propose and evaluate R2D2 (Rapid & Robust Digital Diagnostic) a research-dedicated scalable deep learning toolkit for medical imaging segmentation. Our proposed toolkit introduces (1) a couple of an innovative ready-to-use distributed versions of two popular CNN segmentation architectures (FCN [127] and U-Net[161]) alongside with (2) a high-level end-to-end deep learning medical imaging processing pipeline. The latter aims to reduce the learning curve and overcome talent-intensive deep learning technology adoption barriers for non-specialists. Furthermore, R2D2 integrates a couple of real-time visualization components in order to track both (1) system resources and (2) CNN training metrics evolution during the distributed training process. They offer an extensive overview for a better understanding and easier debugging of the CNN distributed training task progress. We achieve up to 97% and 58% scaling efficiency for U-Net and FCN CNNs respectively when moving from 1 to 18 Nvidia GTX 1080 Ti GPUs without significant, yet still mysterious segmentation accuracy degradation. Indeed, our work constitutes a deep empirical investigation for the latest published papers [61, 183] and confirms state of the art results of related works [121, 208, 82]. Furthermore, in order to prove that R2D2 generalizes for a wider variety of data-sets and tasks, we assess our proposed scalable CNN architectures on two practical medical imaging segmentation use cases. The first one is a brain tumor segmentation challenge, and the second use case is a cardiac left atrium segmentation task. This extensive evaluation provides an in-depth assessments and comparison of the performances of two popular CNN architectures on a couple of challenging case studies.

The remainder of the Chapter is structured as follows: In Section 3.2, we explore some related work. In Section 4.2, we present R2D2 and its design, building blocks and architecture. In Section 3.4, we evaluate our proposed solution, based on two different concrete medical imaging segmentation case studies. We finally conclude in Section 4.5.

3.2 Related Work

A special effort has been made since a long time to develop medical imaging processing tools [119]. They can be classified according to the extent, scope and nature of their application areas. Some generic medical imaging solutions have been around for a while (e.g. (MITK [205], VTK and ITK [156]). They propose a comprehensive set of common medical imaging tasks (e.g., registration [129], segmentation, visualization, and reconstruction [16]). Other generic medical imaging tools yet pathology specialized solutions have been introduced. For instance, FSL (FMRIB Software Library) [90] and Freesurfer software suite [90] are two popular medical image analysis tools specialized in neuroimaging. Finally, a suite of task specific solutions have been proposed (e.g., NiftySeg [25] for segmentation, NiftySim [94] for simulation and Camino [35] for Diffusion).

The previously mentioned medical imaging tools are neither DNN-based solutions, nor distributed applications. However, the recent deep learning breakthroughs led to the emergence of a new set of DNN-based medical image analysis tools. For instance, NiflyNet [54] is an open source deep-learning-based platform of medical imaging which is built on top of *TensorFlow* library. It provides a modular medical imaging processing pipeline alongside with a set of established pre-trained domain specific models. The deep learning toolkit (DLTK) [151] is another open source *TensorFlow*-based medical imaging toolkit implementing baseline versions classic network architectures. DeepInfer [133] is an additional deep learning toolkit for image-guided therapy with a focus on the deployment and the reuse of pre-trained models.

Other related medical image analysis tools exist [157, 110, 5]. Furthermore, although medical imaging DNN-based solutions built on top of *Tensorflow* (e.g., NiftyNet and DLTK) natively support the standard built-in *Tensorflow* parallelization approach, they don't come up with an all set, ready-to-use distributed versions of CNN architectures (i.e., they require a large amount of talent-extensive code modification [171]). Moreover, even if some of them present some similarities with R2D2 (i.e., NiftyNet deep learning medical imaging pipeline, DeepInfer and DLTK proposed pre-trained models), to the best of our knowledge, no existing medical imaging solution offers all the features of R2D2, in particular :

1. The novel and ready-to-use `Dist-FCN` and `Dist-U-Net` distributed versions of the immensely popular FCN and U-Net CNN segmentation architectures respectively.
2. The integrated monitoring platform for system resources supervision and visualization which offers a deeper insights on, not yet investigated, system resources evolution patterns during the distributed training of CNNs. The real time monitoring platform also allows the user to early stop the CNNs training in the case of the divergence of the latter's training process. Thus, the early stopping in such situations will avoid the waste of resources and energy.

The above-mentioned novel features integrated in our proposed toolkit are the main contributions of our work. That is why, they make R2D2 stand out from the rest of existing solutions in medical imaging deep-learning-based solutions.

3.3 R2D2 System Description

This section introduces R2D2 our proposed scalable toolkit. First, we provide a global overview on R2D2 main scope, features, design and system architecture, before diving into the details of its building blocks and core components.

3.3.1 Scope & Architecture

R2D2 toolkit brings the power of distributed systems to use in medical imaging deep-learning-based applications, while considering software engineering best practices. Figure 3.1 depicts the overall scope into which R2D2 operates.

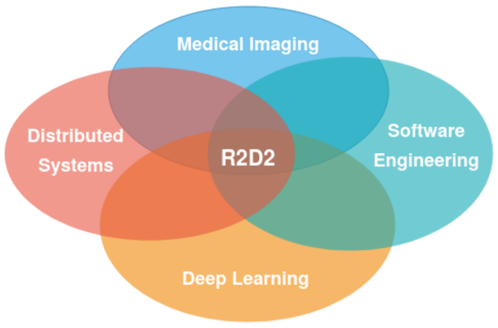


Figure 3.1 – R2D2 scope

Our proposed toolkit follows an extensible and modular design. As illustrated in Figure 3.2 which provides a high-level overview on our distributed toolkit architecture, R2D2 offers an end-to-end support for a typical deep learning workflow in medical imaging by introducing a high-level of abstraction for common components in a regular medical CNNs processing pipeline.

The end-user might interact with the toolkit through different front-ends. He can either use a web-based graphical user interface (GUI) or a command line interface (CLI). The toolkit user has a set of tools at his disposal which are as follows.

- The **R2D2 Engine** is the entry point for the R2D2 toolkit. It is the main system controller operating as a key interface between the toolkit user and the available modules.
- The **ImgMed Library** is a high-level medical imaging pre-processing library which offers a typical medical imaging pre-processing operations.
- The **Dist-Training Module** is the core component of R2D2. It contains `Dist-FCN` and `Dist-U-Net`, a novel distributed versions of widely-adopted FCN and U-Net CNN segmentation architectures respectively.
- The **SegEval Library** is an evaluation library which proposes implementations for a collection of common empirical evaluation methods [218] for semantic segmentation.

R2D2 includes also a set of pre-trained, pathology specific CNN models for renal cortex, liver and brain lesion segmentations. These pre-trained models constitute a model zoo and they might be used to leverage transfer learning approach while building new CNN models. The transfer learning strategy [149] consists in reusing an already pre-trained models as a starting point for the training process of new CNN models in order to, potentially, accelerate model training while improving its performance. Furthermore, it is also possible for the R2D2 user to publish his newly trained models and enrich the pre-trained models collection. Finally, the user can use a web-based graphical interface for a real-time monitoring of the system resources during the distributed training phase. Concurrently, the toolkit user can also visualize the CNN training metrics evolution.

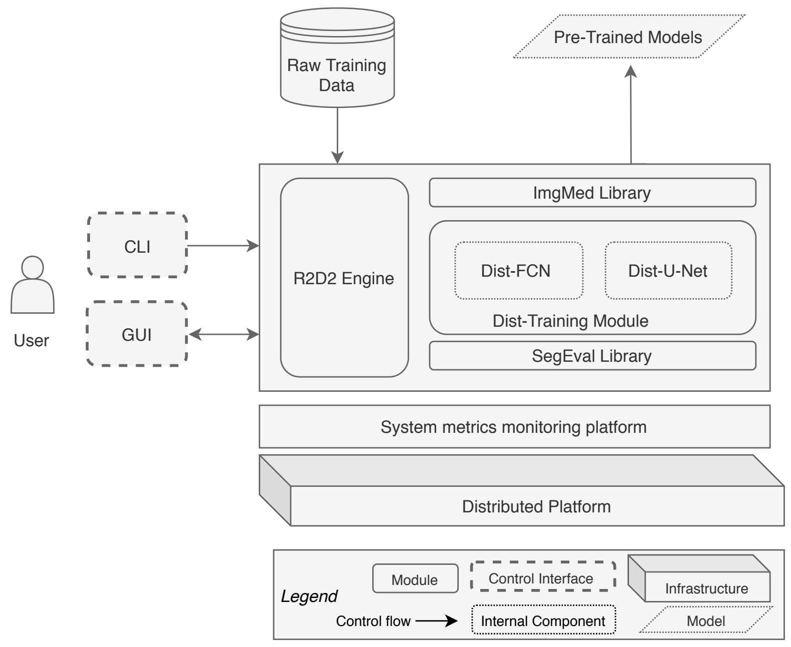


Figure 3.2 – System architecture

3.3.2 Core building blocks

3.3.2.1 ImgMed: Medical imaging pre-processing library

We built `ImgMed` which is a new library dedicated to medical imaging pre-processing workflow. In fact, the data pre-processing phase is an important and key step in machine learning development pipeline. It is a challenging task because it not only conditions the effectiveness of the developed model, but data pre-processing is also a time consuming task as it represents up to 70% of the whole project time [211]. The `ImgMed` library intent to tackle this challenge by proposing an all set high-level medical imaging pre-processing solution.

`ImgMed` includes, but is not limited to, the following typical medical imaging pre-processing operations:

- Image format conversion (e.g., from JPG to PNG)
- Image reshaping (e.g., from 3D to 2D)
- Image resizing
- Data augmentation

We have chosen *Python* as a programming language for `ImgMed` reference implementation for its simplicity. The `ImgMed` library is built upon matplotlib [85], *OpenCV* [20] and *SciPy* [95] libraries.

An example of an end-to-end medical imaging pre-processing workflow implemented using `ImgMed` high-level library is shown in Listing 3.1. We consider a set of N Niftti files as a raw input dataset. The first step is to reshape the 4D input files to an RJB JPG images. Next, another image reshaping procedure is performed, followed by a format conversion operation from JPG to PNG. The final step of the pre-processing workflow example involves resizing the PNG images to 350x350 pixels (note that the sources and destinations file paths can be tuned according to the user needs).

```
from R2D2.img import ImgMed

def preprocessing_workflow(src_0, dist_f):
    """
    Input: N 4D NIfTII files in src_0 path
    Output:N 2D PNG 350*350 images in dest_f path
    """
    ImgMed.convert_4DNI_to_RGBJPG(src_0, dest_0,dim)
    ImgMed.reshape_JPG_to2D(dest_0, dest_1)
    ImgMed.convert_JPG_to_PNG(dest_1, dest_2):
    ImgMed.resize_img(dest_2,dest_f,'png',350,350)
```

Listing 3.1 – Typical medical imaging pre-processing workflow

As it can be noticed, the example in Listing 3.1 not only shows the easiness with which it is possible to implement a complete and classic pre-processing pipeline with only a few lines of code, but it also highlights the considerable impact of `ImgMed` in reducing duplication of effort during the pre-processing step of medical imaging pipeline.

3.3.2.2 Dist-Training Module

The `Dist-Training` module is the core component of R2D2 toolkit. It provides a set of scalable CNN segmentation architectures (`Dist-FCN` and `Dist-U-Net`). Yet, above all, in order to introduce parallelism to FCN and U-Net CNN architectures, a number of choices have to be made among the various distributed training strategies already introduced in section 4.2. Our selection criteria of the considered parallelism method are threefold: (1) The distributed method ***model accuracy preservation***, (2) while taking into account its ***network bandwidth optimality*** (3) and without forgetting to consider the ***burden of its practical implementation***.

In the fist place, we decided to adopt the data parallelism approach for the following reasons :

- In model parallelism, the workload of the partitioning task of a model across multiple nodes is left to the programmer [120, 71, 10], that makes the effective implementation of model parallelism method a challenging task unlike the data parallelism one. For this reason, the model parallelism schema is mainly considered as a final alternative approach when a model size does not fit in a single node's memory [71, 39].
- Since the model parallelism approach involves partitioning the model across several

training agents, and given that the DNN architectures naturally generate a layer interdependencies [10], the synchronization of computations during the training phase in model parallelism strategy creates a communication overhead which increases the training runtime [10, 39].

- Considering that the level of scalability of the data parallelism method is naturally determined by the minibatch hyperparameter size [10], and since recent published works [61, 3] have succeeded to considerably increase the minibatch size without significant segmentation accuracy loss, data parallelism has become the most preferred distributed training approach.

Then, we chose to scale up the training of FCN and U-Net architectures using a synchronous parallelism approach. Our selection criterion for the latter chosen strategy was the trade-off between the CNN model accuracy and the training speedup. In fact, synchronous methods achieve better results regarding the accuracy of the CNN models compared to the asynchronous approaches [10, 71], particularly, with a short synchronization period [216].

The main steps in the selected synchronous distributed data parallelism strategy are as follows [171]: (1) compute the model updates (gradients of the loss function) using a minibatch on each training agent (2) compute the average of gradients of all training agents (3) update the model. Hence, we have to select the parameters updates communication and propagation schema. Even if both centralized and decentralized parameters updates communication approaches have advantages and drawbacks, we decided to go for a decentralized Ring-Allreduce [171, 53] algorithm for the following reasons.

- Since the network bandwidth is classified among the rarest resources in datacenters [123], and even if the centralized parameter server is one of the popular approaches in distributed machine learning with better fault tolerance, it suffers from a bandwidth bottleneck especially with large scale systems [10, 123].

- Although the parameter server congestion issue might be alleviated through some alternative parameter server infrastructures (e.g., shared parameter server), selecting the appropriate ratio of parameter servers in these alternative configuration is still a challenging task [171].

- The Ring-Allreduce algorithm is built on a HPC approach proposed in 2009 by Patarasuk and Yuan [150]. It is a highly scalable and bandwidth optimal approach as it remarkably reduces the network communications overhead [171], which perfectly corresponds to our aforementioned parallelism schema selection criterion.

In summary, we decided to adopt a ***decentralized synchronous Ring-Allreduce data parallelism strategy*** in order to bring `Dist-FCN` and `Dist-U-Net` into practice.

We distributed the training of FCN and U-Net CNNs as follows: First and foremost, we introduced data parallelism by deploying the same CNN segmentation architecture on each training node (i.e, either FCN or U-Net). After that, we sat up the Ring All-reduce algorithm. The latter steps are as follows: Initially, each worker node reads its own subset of the current mini-batch. After that, it computes its gradients, and communicates it to its nearby successor on the ring and get in turn the calculated gradients from its predecessor neighbor. In a ring that counts N workers, it takes N-1 communications of gradients between workers, so that every worker receives the required gradients values to compute the updated model. Also, we ensured the system fault tolerance through a checkpoint/restart schema. Last but not least, considering that we scaled up the training of FCN and U-Nets CNNs using a data parallelism schema, we applied the *learning rate linear scaling rule* which consists in adjusting the learning rate as a function of the minibatch size [61] in order to distribute the training of our CNNs without considerable segmentation accuracy loss.

3.3.2.3 SegEval: Segmentation Evaluation Library

Once we have finished the distributed training of our CNN architecture, the next step in a typical processing pipeline is to evaluate the trained model. To this end, R2D2 provides an evaluation library which implements a set of common evaluation metrics for both binary and multi-label semantic segmentation tasks.

We denote $n_{cl} \varepsilon \mathbb{N}$ the number of classes. Also, we denote n_{ij} the number of pixels of class i predicted to belong to class j and $t_i = \sum_j n_{ij}$ the total number of pixels of class i. The `SegEval` library offers the following widely-adopted evaluation metrics [127].

- The **Dice** score reports the percentage of overlap between the predicted segmentations and the ground truth masks: $Dice = \frac{1}{n_{cl}} \sum_i \frac{2n_{ii}}{2n_{ii}+n_{ij}+n_{ji}}$
- The **Pixel Accuracy** (PA) is a measure of the percentage of the image pixels that were correctly classified : PA = $\frac{\sum_i n_{ii}}{\sum_i t_i}$
- The **Mean Accuracy** (MA) is the mean of the Pixel Accuracy across the n_{cl} classes: MA = $\left(\frac{1}{n_{cl}}\right) \sum_i \frac{n_{ii}}{t_i}$
- The **Mean Intersection Over Union** (mean.IoU) is a measure of the area of overlap divided by the area of union between both predicted and groundtruth images : (mean.IoU) = $\left(\frac{1}{n_{cl}}\right) \sum_i \frac{n_{ii}}{t_i+\sum_j n_{ji}-n_{ii}}$ and the **frequency weighted IoU** (f.w.IoU): $\left(\sum_k t_k\right)^{-1} \sum_i \frac{t_i n_{ii}}{t_i+\sum_j n_{ji}-n_{ii}}$

3.4 Evaluation

In this section, we conduct a comprehensive evaluation of the `Dist-Training` module which is the core building block of our proposed R2D2 toolkit. We first introduce our experimental environments. Afterwards, we present the experimental evaluation results of the distributed CNN architectures (`Dist-FCN` and `Dist-U-Net`) on a brain tumor segmentation use case. Finally, in order to validate the `Dist-Training` module transferability to other segmentation use cases, we assess the module on a second medical imaging segmentation task, which involves locating the heart's left atrium structure.

3.4.1 Experimental Environments

3.4.1.1 Hardware

We accomplished the distributed training experiments on the Nancy Grid'5000 [8] testbed site. The experiments were conducted on Grele GPU cluster which contains Dell PowerEdge R730 physical machines where each node is equipped with 2 Nvidia GeForce GTX 1080 Ti GPUs. We use the Grid'5000 Network File System (NFS) to share the training dataset between all training agents. The nodes are interconnected using *InfiniBand* [177] high-speed interconnect.

3.4.1.2 Software

The FCN and U-Net architectures were mutually built on top of google's *Tensor-Flow* library. Furthermore, the U-Net CNN was also implemented using the high-level *keras* [30] API to ensure an easier architecture prototyping task. After that, in order to practically implement the `Dist-FCN` and `Dist-U-Net` by introducing the considered synchronous Ring-Allreduce data parallelism schema, we take advantage of the *Horovod* [171] based implementation of the Ring-Allreduce algorithm. The latter is built concurrently on both, *Open MPI* [49] and *NCCL 2.0* [1] communication libraries. Moreover, during the experimental evaluation, we simultaneously make use of the proposed R2D2 module incorporating *TICK Stack* monitoring platform [2] in order to collect system metrics data during the distributed training. The collected datasets are stored in the time series database*InfluxDB*. Also, since the `Dist-Training` module is partially built using the *TensorFlow* library, we leverage the natively integrated *TensorBoard* visualization component, in order to enable R2D2 users to have an extensive overview on the training progress and hence facilitating the debugging of the CNNs training step. Finally, to consider software reusability, and ensure research reproducibility, the `Dist-Training` module and its runtime components were containerized into a debian 9 stretch-based docker image without network isolation (i.e., by directly using the host networking driver for an optimal network performance [45]). Figure 3.3 details the physical architecture of our experimental environment.

[1] https://developer.nvidia.com/nccl

[2] More informations on TICK Stack platform can be found at the following link https://www.influxdata.com/time-series-platform/

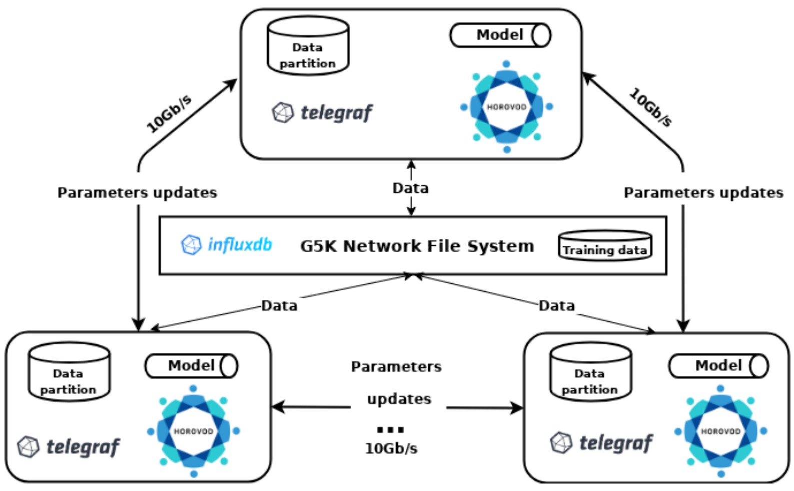


Figure 3.3 – Distributed training experimental environment architecture

3.4.2 Dist-Training module training time evolution with scalability

In this subsection, we evaluation the `Dist-Training` module training time evolution while increasing the number of GPUs. Figure 3.4 and Figure 3.5 illustrate the decrease in the training time while scaling up the training of `Dist-U-Net` and `Dist-FCN` respectively. Multiple runs have been conducted for each configuration to assess `Dist-U-Net` and `Dist-FCN` evaluation results variability. we run each experimental training configuration between 3 and 5 times and we consider the average of the measured execution duration as our reference inference results. The `Dist-U-Net` reaches **17.5x** speedup and **97%** scaling efficiency going from 21 hours and 40 minutes for single-GPU based U-Net, to 1 hour and 14 minutes for `Dist-U-Net` trained on 18 Nvidia GTX 1080 Ti GPUs. In the other side, the `Dist-FCN` achieves **10.4x** speedup and **58%** scaling efficiency reducing the training time from 35 hours and 40 minutes for a single-GPU based `Dist-FCN` to 3 hours and 25 minutes for `Dist-FCN` trained on 18 Nvidia GTX 1080 Ti GPUs.

The `Dist-U-Net` and `Dist-FCN` have been evaluated concurrently on the two previously introduced rain tumor and left atrium segmentation use cases. Figure 3.4 and Figure 3.5 both show that the brain tumor segmentation case study training time evolution curve closely match the left atrium one which establishes the transferability of our proposal. Also, we notice that the baseline U-Net CNN converges faster than the FCN one in both segmentation case studies in only 21 hours and 40 minutes. This observation matches and confirms the findings of Li et al. [121] which highlights that residual connections (similar to the ones that exist in U-Net architecture) produce a smoother loss function which leads to an easier and faster convergence.

The disparity of the scaling efficiency and speedup of `Dist-FCN` and `Dist-U-Net` is mainly due to the nature of the experimental setup and the difference in their corresponding implementation strategies. In particular, during the training process of `Dist-U-Net`, the entire

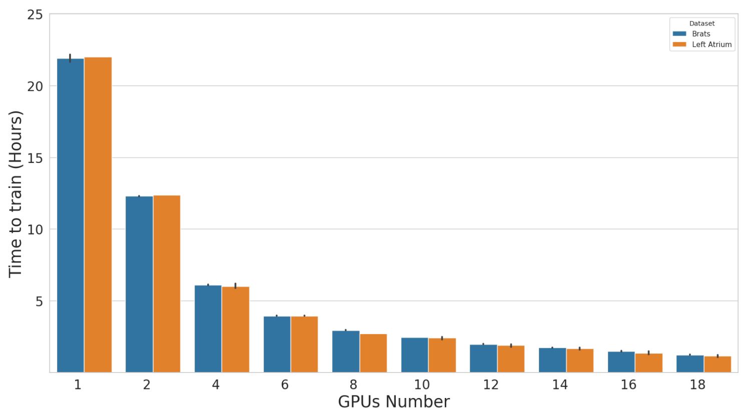


Figure 3.4 – Training time evolution with scale for Dist-U-Net

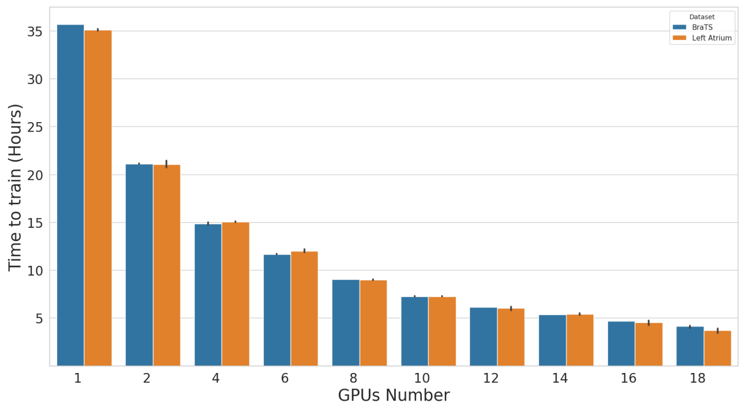


Figure 3.5 – Training time evolution with scale for Dist-FCN

training and validation sets are loaded into the random access memory (RAM) of each training agent. On the other hand, the `Dist-FCN` implementation takes advantage of the GPU's dedicated memory to iterate through the training and validations datasets in order to load each minibatch through a Network File System (NFS) which represents a communication overhead. Furthermore, the `Dist-U-Net` implementation contains highly optimized operations to accelerate the distributed validation step.

We assess the testing time of `Dist-FCN` and `Dist-U-Net`. Indeed, in order to eliminate the system routine operations influence in time measures, we run each experimental setup 10 times and we consider the average of the measured execution duration as our reference inference results. The testing workstation is equipped with an NVIDIA GTX 1080 TI GPU. The obtained testing times per image are 153 ms and 182 ms for `Dist-U-Net` and `Dist-FCN` respectively.

3.4.3 Dist-Training segmentation accuracy evolution with scalability

Throughout this subsection, we evaluate the impact of increasing the GPUs number on the segmentation accuracy. Even though it is common to use a unique segmentation evaluation metric, we consider the entire evaluation metrics provided by the previsouly-introduced `SegEval` module in order to comprehensively assess our proposal. Similarly to aforementioned in subsection 3.4.2, we run each experimental training configuration between 3 and 5 times and we consider the average of the measured metric as our reference inference results.

3.4.3.1 Dist-FCN for brain tumor segmentation

The adopted `Dist-FCN` architecture for the brain tumor segmentation consists of a total of 16 fully convolutional network layers. It takes the input volume throughout a sequence of an increasing number (i.e., n=2 for the first two blocks and n=4 for the rest of blocks) of convolutional layers which are immediately followed by ReLU activation function. At the end, a max-pooling layer is applied. The sequence n x (convolution + ReLU) is repeated again 4 times before performing *upsampling* through a transposed convolution upsampling layer [147] and applying a softmax layer [127, 107] for the pixel-wise classification task. Moreover, dropout regularization technique [127] was applied during the training phase to avoid overfitting. Finally, the network parameters were initialized through *Transfer Learning* using a pre-trained VGG-16 model on the ImageNet dataset [3] .

Training Settings:

The training was performed using the mini-batch stochastic gradient descent (SGD) optimization algorithm (see subsubsection 2.1.1.1 of chapter 2) with a mini-batch size of 10 (to fit the GPU memory limit). The training process was done for a total of 120 epochs and a learning rate of $1e-5$ (see subsubsection 2.1.1.4 of chapter 2). All training hyperparameters were kept unchanged except of the learning rate which was adjusted according to *the learning rate linear scaling rule*. In the first place, no learning rate warmup strategy was applied, before performing a gradual warmup schema afterwards. The gradual warmup schema consists in applying progressively a low learning rate for the first few epochs in order to overcome convergence issues at the start of the training, i.e., the DNN loss function starts to diverge after starting to have a converging trend before [61].

Evaluation:

We considered `SegEval` integrated evaluation library of R2D2 in order to assess our conducted

[3] More informations and the download link for the pre-trained VGG-16 model on the ImageNet dataset can be found at the following link: http://www.vlfeat.org/matconvnet/pretrained/

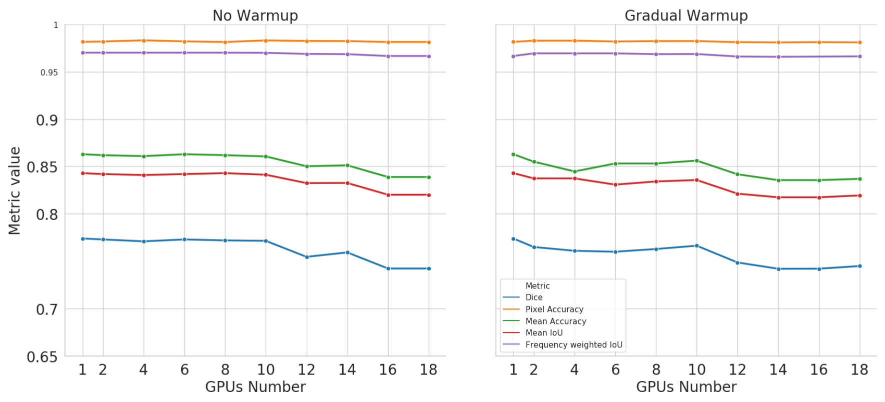


Figure 3.6 – Segmentation metrics evolution with scale for Dist-FCN for brain tumor segmentation

experiments. We found that the brain tumor segmentation trained on 18 GPUs without learning rate warmup strategy reaches 74.22% *dice score* accuracy, 84.29% *Mean IoU* and 86.28% *mean accuracy* which are 4.08% , 2.7% and 3.35% respectively lower than the single GPU based model. At the same time, the gradual warmup strategy enhance the segmentation accuracy loss by 3.76%, 2.8% and 3.01% for the *dice score*, *Mean IoU* and the *mean accuracy* metrics correspondingly. Our practical experiments results show an interesting unexpected segmentation accuracy loss increasing with the parallelism degree.

3.4.3.2 Dist-U-Net for brain tumor segmentation

Training Settings:

For training, we use the mini-batch SGD optimization algorithm. The training phase was done with a mini-batch size of 7, during 100 epochs and while using an initial base learning rate of $1e-5$.

Evaluation:

Figure 3.7 introduces the brain tumor segmentation accuracy evolution when scaling up the `Dist-U-Net` (1) with no warmup phase and (2) while performing a gradual warmup for the first 5 epochs. As can be seen, the *dice score* decreased by 0.44% going from 0.890 in 1-GPU implementation to 0.886 in 18 GPUs in the case of no warmup phase. Similarly, the *mean.IoU* and *mean accuracy* metrics drop with 0.55% and 0.61% respectively. On the other hand, the

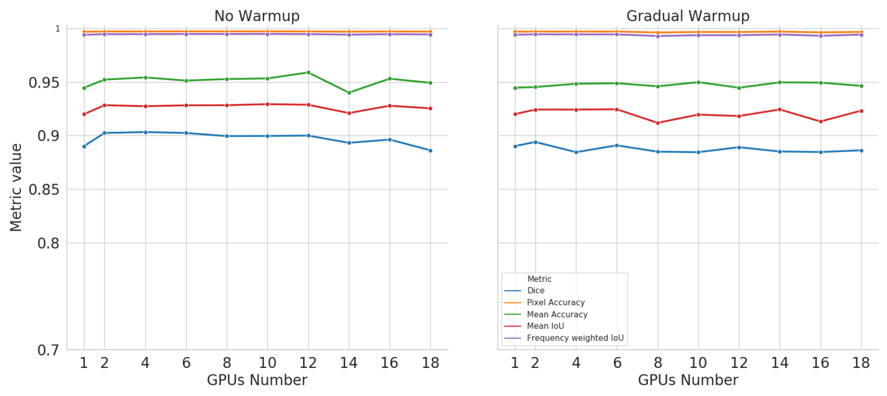


Figure 3.7 – Segmentation metrics evolution with scale for Dist-U-Net for brain tumor segmentation

gradual warmup strategy achieves the same *dice score* accuracy loss as the no warmup strategy. Nonetheless, the gradual warmup strategy seems not to be effective at low scalability level as is does not help the network to converge faster. This statement gives us a hint that a headline application of the linear scaling rule with gradual warmup is naturally biased since it alternates the hyperparameters tuning process. Finally, no accuracy degradation is reported in the pixel accuracy and *f.w.IoU* metrics regardless of the adopted warmup schema. To sum up, our experiments highlights an unexpected segmentation accuracy degradation with scale, nevertheless its small value.

3.4.3.3 Dist-FCN for left atrium segmentation

We adopted a 16 layers `Dist-FCN` CNN architecture similar to aforementioned one in subsubsection 3.4.3.1. Similarly, we also leveraged transfer learning approach using pre-trained VGG-16 model on the ImageNet dataset.

Training Settings:

The training was performed using the mini SGD optimization approach, a mini-batch size of 10, for a total of 120 epochs. We also applied the *learning rate linear scaling rule* starting with an initial learning rate of $3e-5$.

Evaluation:
Figure 3.8 illustrates the segmentation accuracy metrics evolution when scaling up the `Dist-FCN` for left atrium segmentation before and after performing Gradual warmup strategy.

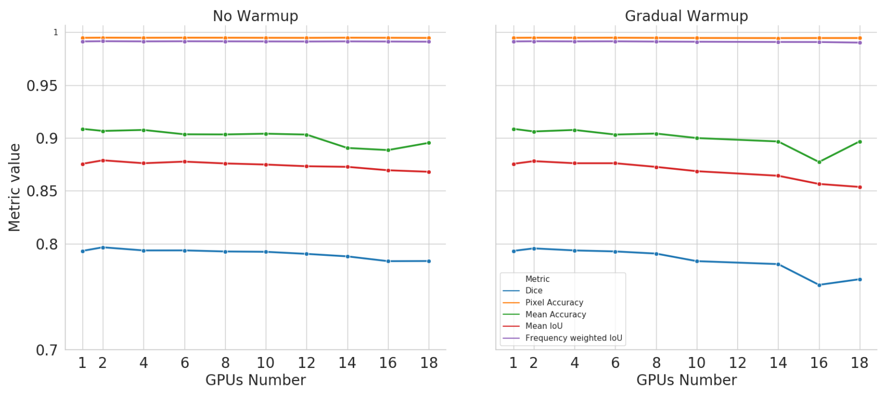


Figure 3.8 – Segmentation metrics evolution with scale for Dist-FCN for left atrium segmentation

It illustrates a *dice score* accuracy and *mean accuracy* fall of 3.38% and 1.3% accordingly for a gradual warmup initialization learning rate approach. Yet, with no warmup strategy, the `Dist-FCN` acheives better results with 1.21%, 0.86% and 1.46% segmentation accuracy decrease for the *dice score*, *mean accuracy* and *mean.IoU* respectively. However, no accuracy loss is reported for the *f.w.IoU* and the *pixel accuracy* metrics. Finally, once again, even if the linear scaling rule is supposed to eliminate the accuracy loss, our experiments show a quite surprising accuracy degradation when scaling up the considered GPUs number.

3.4.3.4 Dist-U-Net for left atrium segmentation

Training Settings:

For training, we use the mini-batch SGD optimization algorithm. The training phase was done with a mini-batch size of 7, during 100 epochs and while using a learning rate of $2e-5$.

Evaluation:

As can be seen in Figure 3.9, scaling up the training of `Dist-U-Net` for left atrium segmentation to 18 GPUs without gradual learning rate warmup strategy achieves 79.48% *dice score* accuracy and 96.41% *mean accuracy* which are 2.26% and 1.53% respectively lower than the single GPU `Dist-U-Net` baseline trained model. However, the gradual warmup approach improves the accuracy degradation to only 1.34% and 0.31% for the dice score and *mean accuracy* metric correspondingly. Yet again, our experimental results reveal a quite unexpected segmentation accuracy loss when scaling up the CNNs training process. Also, these results show that the *PA*

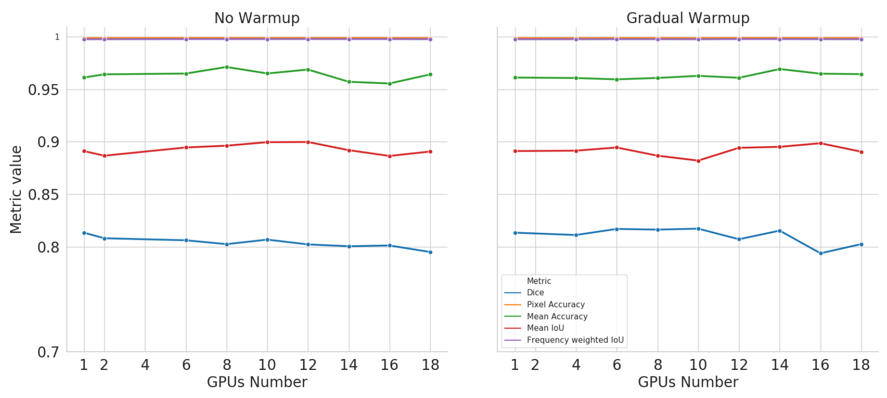


Figure 3.9 – Segmentation metrics evolution with scale for Dist-U-Net for left atrium segmentation

and *f.w.IoU* metrics are not very relevant for our experiments assessment process. Indeed, they suffer from a unbalanced variability range due to the disproportional size of every class in our segmentation case studies (e.g., the disproportional size between the small left atrium body and large background class size)

3.4.4 Discussion

We evaluated our proposed `Dist-FCN` and `Dist-U-Net` training time and segmentation accuracy metrics evolution with scale on a couple of challenging medical imaging segmentation case studies (1) BRATS: a dataset with small targets (tumors) in large MRI images (2) Left Atrium: a small training set with large variability. The case studies evaluation results led us to not only assess the segmentation accuracy evolution when scaling up the `Dist-FCN` and `Dist-U-Net` architectures, but also to compare FCN and U-Net performances in a couple of different segmentation tasks. Actually, the evaluation results showed that the U-Net CNN architecture achieves a far better performances than the FCN one in the brain tumor segmentation task with 90.23% dice score. Also, the U-Net and FCN CNNs produce a close results in term of performances for the left atrium segmentation with an advantage of 1.8% in the dice score for the U-Net architecture. These findings confirm the need to perform multiple CNNs training runs in order to investigate the best suited CNN architecture for a particular task alongside with its corresponding optimal hyperparameters set. Hence, the interest of R2D2 in accelerating the prototyping and the development of cutting-edge CNNs which in turn is a capital software engineering principal.

The aforementioned empirical evaluation results led us also to perform a deeper experimental analysis of the generalization of last published works [61, 183, 121, 208, 82] to the segmentation task. Actually, the segmentation task is a more complex task compared to the classification task which was mainly used in the state of the art works to assess the linear scaling rule and its corresponding warmup schemas [61]. The experimental results showed that there was no segmentation accuracy loss until 12 GPUs. Starting from that scalability level, the learning rate scaling rule breaks down. Indeed, even though the observed accuracy loss is insignificant compared to the remarkable scaling efficiency, it is yet still curiously unexpected for the investigated segmentation task in particular. On the other hand, these results are in line with the 1% increase of error rate reported by Krizhevsky [106] when increasing the minibatch size from 128 to 1024 for classification task. Also, You et al. [208] outline also an accuracy deterioration of 5.7% by using the linear scaling rule and the warmup schema for CNNs applied for classification task. Furthermore, Hoffer et al. [82] show that there is still an accuracy degradation for CIFAR10 [4] classification task even while using the linear scaling rule. Hence, our experiments confirm the results of these works [106, 208, 82] and call into question the extent of the linear scaling rule to the segmentation task.

3.5 Conclusion

In this Chapter, we proposed and evaluated a scalable deep learning toolkit for medical imaging segmentation named R2D2. The main goal of R2D2 is to speed-up the research in the deep-leaning-based medical imaging applications with a particular focus on semantic segmentation task. We exposed R2D2 concepts and design and detailed its inner buildings components, while justifying our design and implementation choices. We then evaluated our scalable toolkit on two distinct concrete medical imaging segmentation case studies to show the effectiveness of our proposal. The conducted experimental study offers an empirical evidence and further investigates the latest published works. Indeed, R2D2 achieves up to 17.5x and 10.4x speedup than single-node based training of U-Net and FCN respectively with a slight, yet nonetheless an unforeseen segmentation accuracy degradation with scale. This contribution has been published in Software: Practice and Experience journal [65]. The R2D2 source code was the subject an APP deposit.

[4]https://www.cs.toronto.edu/~kriz/cifar.html

4 Auto-CNNp: a component-based framework for automating CNN parallelism

After presenting R2D2, we present in this chapter Auto-CNNp [66, 67], the second building block of our introduced platform. Auto-CNNp [67] is a novel framework to automate CNNs training parallelization task. To achieve this goal, Auto-CNNp introduces a key component which is called *CNN-Parallelism-Generator*. The latter component encapsulates and hides typical CNNs parallelization routine tasks while being extensible for user-specific customization. Our proposed reference implementation provides a high level of abstraction over MPI-based CNNs parallelization process, despite the CNN-based imaging task and its related architecture and training dataset. The quantitative and qualitative assessment of our proposal on two case studies show its (1) effectiveness in accelerating the process of scaling up CNNs training and (2) its generalization for a wider variety of use cases.

4.1 Introduction

Setting up distributed training of CNNs in practice is a laborious task entailing a significant degree of experience and expertise in both (1) deep learning and (2) distributed optimization approaches. Moreover, introducing parallelism to CNNs training is a manual, redundant, time-consuming and error-prone process. For instance, even though *Tensorflow* natively includes a standard built-in parallelization approach[1], going distributed using it is a laborious and challenging task [171]: It requires a large amount of knowledge from the user of a considerable low level abstractions of *Tensorflow* and a lot of manual code modifications. The aforementioned problems which we particularly encountered while conducting the practical experimental study introduced in the previous chapter, led us to realize the importance and the need for not only **automating** routine tasks to avoid duplication of effort while scaling up CNNs training, but also to adopt a **component reuse** approach while considering the software **extensibility** principle.

We leverage this opportunity by introducing Auto-CNNp (Automatic CNN parallelization) [66],

[1]More information on distributed *Tensorflow*: https://www.tensorflow.org/guide/distribute_strategy

a component-based framework that fully automates scaling up MPI-based CNNs training task in order to bring talent-intensive distributed deep learning to non-experts users. To the best of our knowledge, the present work is the first that aims to tackle this issue by introducing a novel component-based approach. We present the design and the core building blocks of our proposed framework. The *CNN-Parallelism-Generator* component aims to streamline routine tasks throughout (1) capturing cumbersome CNNs parallelization tasks within a backbone structure while (2) keeping the framework flexible enough and extensible for user-specific personalization. The user defines the specific framework behavior through an easy-to-understand configuration file.

Our contribution lies within the proposal of a standard component-based approach to parallelize CNNs training regardless of the (1) CNN-based image processing task, (2) its corresponding CNN architecture and (3) training dataset. Furthermore, although our proposed Auto-CNNp Proof-of-Concept (POC) reference implementation is based on both (1) *Ring-Allreduce* parallelism approach and (2) *MPI* communication protocol, it is indeed possible to port the framework to additional CNN parallelism and communication approaches as the framework's fundamentals remain valid. The evaluation result of our proposed automated component-based approach are promising. It shows that a significant speedup in the CNN parallilization task has been achieved to the detriment of a negligible framework execution time, compared to the manual parallelization strategy.

This chapter is organized as follows: Section 4.2 provides background information on component-based software engineering and reviews some related work. Section 4.3 describes our approach to automatize the parallelization of our POC MPI-based CNNs training. We present the evaluation of our proposal in Section 4.4, and conclude in Section 4.5.

4.2 Component-based software engineering background

In this section, we overview some background on *Component-based Software engineering (CBSE)* [78]. CBSE is far from being a recent research area. Indeed, it aims to build software systems by composition of software components building blocks [154]. It has become a paramount approach to accelerate the development, deployment, management of large and complex software systems. The *component-based* approaches have been adopted in a wide range of relevant fields of applications, such as e-commerce [47], robotics software [141] and web applications development [23].

Component-based parallel systems development is a not a novel concept neither. Bramley et al. [21] introduced a component-based approach to build scientific and engendering applications. Also, *COMDES-II* [101] is a framework to develop parallel real-time control applications.

Other parallel systems implementations tools exist. For instance, *JaSkel* [46] is a Java framework for parallel and grid applications implementation. It shares some common concepts with Auto-CNNp. Particularly, encapsulating recurring parallelism routines and hiding low-level implementation details. Yet, JaSkel is not a component-based system.

The previously cited systems are not DNN-based solutions. However, With the recent growing interest to deep learning, a lot of distributed deep learning frameworks have emerged (e.g., TensorFLow, Horovod[2], DL4J[3], BigDL[4]). Nevertherless, to the best of our knowledge, no existing solution offers all the features of Auto-CNNp. In particular:

- Auto-CNNp adds an additional high level of abstraction over MPI-based CNNs parallelism techniques by fully automating the scaling up process for various CNN-based image processing task, regardless of its corresponding CNN architecture and training dataset.
- Our proposal is the first easily extensible component-based deep learning parallelism framework.

Hence, our proposed framework accelerates the research in the CNN-based field by prototyping and exploring cutting-edge and not yet investigated CNN configurations and architectures through an iterative and adaptive experimentation approach.

4.3 System description

This section describes Auto-CNNp our proposed framework. First, we provide a global overview on Auto-CNNp main scope, design and system architecture, before diving into the details of its building blocks and core components.

4.3.1 Framework Scope

Figure 4.1 pinpoints the overall scope into which Auto-CNNp operates. Indeed, the operating-system-level environment deployment on the training nodes is currently out of scope of the Auto-CNNp framework. We suppose that the training is performed on an all-set, already deployed distributed system environment (i.e., in the context where the operating system was already sat up on beforehand using tools like *SaltStack* [5] or *Puppet* [6]).

Also, we take advantage of a containerization technique to package the distributed deep learning application with its related default runtime environment (i.e., libraries, binaries and dependencies). It is indeed **within** this specific range of execution context where our proposed framework operates. Particularly, Auto-CNNp provides a backbone for a new way to ***automatically configure and customize*** the specific libraries of a distributed CNN-based

[2]https://eng.uber.com/horovod/
[3]https://deeplearning4j.org/
[4]https://bigdl-project.github.io
[5]More informations can be found at https://www.saltstack.com/
[6]More informations can be found at https://puppet.com/

application runtime environment (i.e., mainly by (1) setting up the configuration of communication libraries and (2) establishing the related deep learning user-specific execution schema).

Regarding the distributed deep learning application level, and as stated previously in section 4.2, multiple CNN parallelism approaches exist. We decided to adopt a ***decentralized synchronous Ring-Allreduce data parallelism strategy*** for the Auto-CNNp reference implementation for the following reasons:

- Considering that the level of scalability of the data parallelism method is naturally determined by the minibatch hyperparameter size [10], and since recent published works [61, 183] have succeeded to considerably increase the minibatch size without significant segmentation accuracy loss, data parallelism has become the most preferred distributed training approach.

- We chose a synchronous parallelism approach. Our selection criterion for the latter chosen strategy is the trade-off between the CNN model accuracy and the training speedup. In fact, synchronous methods achieve better results regarding the accuracy of the CNN models compared to the asynchronous approaches [10, 71], particularly, with a short synchronization period [216].

- The *Ring-Allreduce* algorithm is built on a HPC approach proposed in 2009 by Patarasuk and Yuan [150]. It is a highly scalable and bandwidth optimal approach as it remarkably reduces the network communications overhead [171]. Moreover, Since the network bandwidth is classified among the rarest resources in datacenters [123], and even if the centralized parameter server is one of the popular approaches in distributed machine learning with better fault tolerance, it suffers from a bandwidth bottleneck especially with large scale systems [10, 123].

Also, we adopted MPI as a communication protocol for the framework reference implementation. Indeed, MPI communication libraries have achieved remarkable performances in distributed deep learning applications due to the similar characteristics between distributed deep learning and HPC applications [10].

Nevertheless, as stated previously, it is possible to **port and extend** the framework implementation to support other parallelism approach and communication mechanisms as the framework's core principals remain well-founded. However, further modifications should be applied due to the eventual dependencies between the CNN training parallelism methods and the adopted communication protocols. These dependencies will be further discussed in subsection 4.3.5.

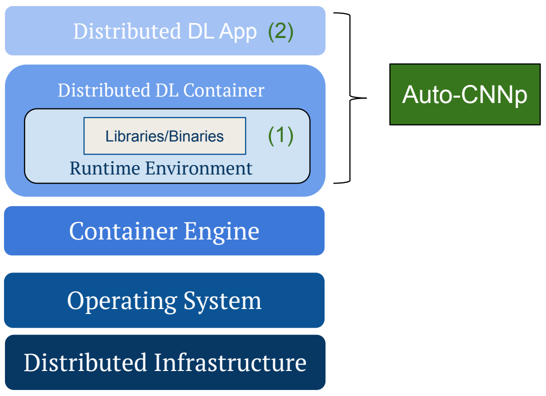


Figure 4.1 – Auto-CNNp operating scope: (1) runtime environment configuration and (2) user-specific deep learning execution schema definition.

4.3.2 Framework Architecture

As illustrated in Figure 4.2 which shows an overview of the architecture of our proposed system, Auto-CNNp framework follows a modular design. Its different building blocks are as follows:

- The **Engine** is the Auto-CNNp controller (i.e., it manages the framework's control flow). It is the central access point operating as an orchestrator of the framework's components interactions.
- The **CNN-Parallelism-Generator** is the core component of Auto-CNNp framework. It aims to simplify the task of scaling up CNNs training by separating typical parallelization strategies patterns from task-specific CNN applications. To achieve this goal, the *CNN-Parallelism-Generator* component captures common routine tasks (i.e., which are shared by all MPI-based deep learning distributed training approaches) and enables users to customize the remaining applications-specific parts.
- The **Run & Manage** component applies the final execution schema of the framework once all the training agents are ready for the distributed training. Indeed, the *Run & Manage* component is activated by the *engine* in order to initiate and launch the distributed training process.
- The **Training Config File** contains a set of an rules used by the *engine* to govern the execution mechanism of the Auto-CNNp framework.
- As its name suggests, the **distributed training infrastructure** is the execution infrastructure for the distributed training of CNNs.

Further details regarding the aforementioned Auto-CNNp core components are given later in this section.

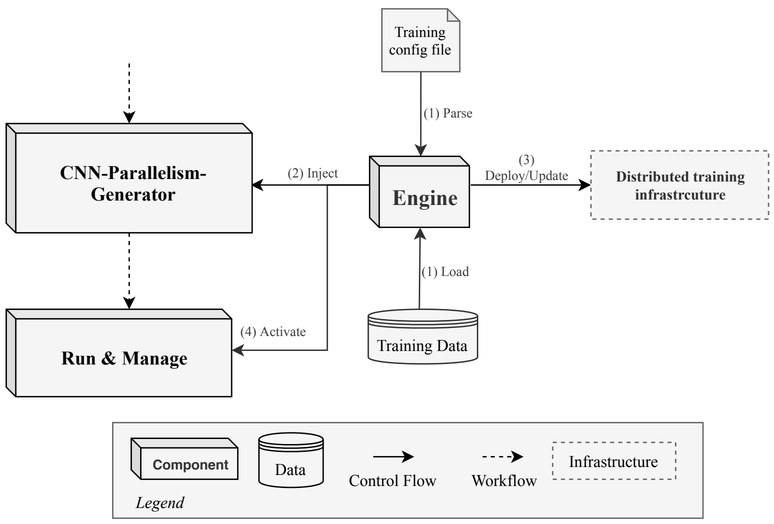


Figure 4.2 – Auto-CNNp System Architecture Overview.

4.3.3 Framework Execution Flow

The Auto-CNNp framework is a configuration-driven framework. The framework's execution flow steps are the followings.

1. First and foremost, the user provides an XML-based *training configuration file*
2. The framework's *engine* parses the *configuration file* and extracts the user-specific application behavior.
3. The *CNN-Parallelism-Generator* is deployed/updated on all the training nodes. Concurrently, the *run & manage* component is only deployed on the training node which initiates the training.
4. Lastly, when all the training agents are ready, the end-user activates the *run & manage* component through the *engine* in order to start the distributed training.

4.3.4 Component Detail: The Engine

As illustrated in Figure 4.2, the architecture of Auto-CNNp is based around the `engine`. The latter implementation has to be fast, to decrease the overhead of the framework to the utmost possible degree. Its functionalities are fourfold. In particular (1) parsing the *configuration file*, (2) based on that, the *engine* establishes the *CNN-Parallelism-Generator* final shape (i.e., its final comprising sub-components and modules). In order to do so, the *engine* parameterizes, customizes and loads the *CNN-Parallelism-Generator* building blocks. Next, the *engine* deploys/updates the *cnn-parallelism-generator* on the training nodes. Lasty, it activates the *run & manage* component in order to start the distributed training task.

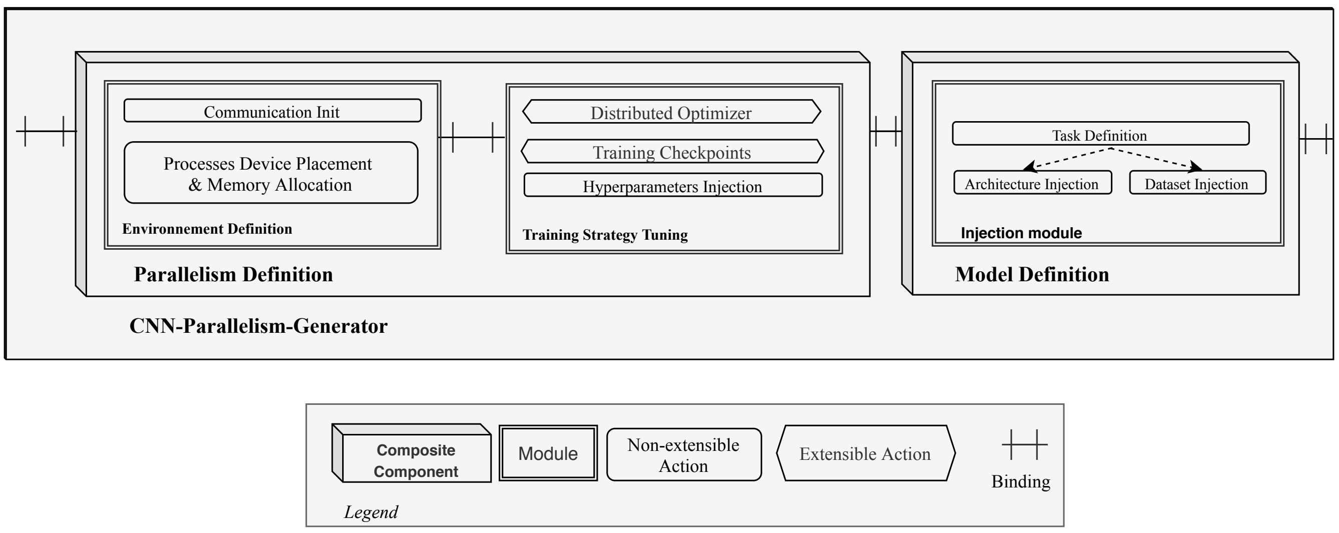


Figure 4.3 – CNN-Parallelism-Generator Component-Based Architecture.

4.3.5 Component Detail: The CNN-Parallelism-Generator

The *CNN-Parallelism-Generator* is the paramount component of the Auto-CNNp framework. It encapsulates and hides reusable CNNs training parallelization patterns to the framework end users in order to provide a higher level of abstraction. As shown in Figure 4.3, the *CNN-Parallelism-Generator* has a linear design following the typical workflow of steps to parallelize our MPI-based CNNs (presented in greater detail later). It is composed of a tree of hierarchically organized building blocks. The different abstractions used in the CNN-Parallelism-Generator are:

- `Components` are the building blocks of the *CNN-Parallelism-Generator*. They are classified into two categories : either (1) **modules** or (2) **composites** components. The **modules** do not contain other components while composites components might be composed of one or several composites components and/or modules Also, components are connected by so-called binding connectors. Composite components are a standalone components which can be reused and replaced without affecting the framework's fundamentals.

- `Modules` are a primitives components. They contains a set of task-related `actions` cooperating towards a particular CNNs MPI-based parallelization milestone (e.g., the parallelism and the model definition). `Modules` present an intra non-functional dependencies within each others In other words, overwriting `modules` requires the user to change/adjust the corresponding related `modules` within the same `component`.

- `Actions` are a standard collective parallelization steps. They may be classified according to their expandability property into (1) `non-extensible actions` and (2) `extensible actions`. The `non-extensible actions` constitutes a set of generic functionalities which have a unique and static implementation. They are independent from the CNNs parallelization schema, can be parametrized but cannot be extensible by the user. On the other hand, the `extensible actions` can further support extensibility by the framework's end user in order to expand or override the framework's supported functionalities.

- `Bindings` aims to connect `components` with each other. These connectors regulate interactions between components by mainly ensuring the transfer and control of data between them.

As illustrated in Figure 4.3, the *CNN-Parallelism-Generator* is a composite of two components (*Parallelism Definition and Model definition*). The latter are in turn are a composite of a couple and a single modules respectively.

4.3.5.1 Module Details: Training Environment Definition

As its name suggests, the *training environment definition* is a primitive module aiming to establish the global CNNs distributed training ecosystem. In particular, it contains the followings actions:

- *Communication Init* is a `non-extensible action` that initializes the adopted communication approach. In our POC implementation, it initializes the MPI default supported protocol.
- *Processes Device Placement & Memory Allocation* is a `non-extensible action` which establishes the custom *TensorFlow-based* processes device placement strategy on the training agents alongside with he adopted memory allocation strategy [7].

4.3.5.2 Module Details: Training Strategy Tuning

The *training strategy tuning* module defines the broad lines of the adopted CNNs training approach alongside its corresponding hyperparameters and particular customized training checkpoints. In more details, it contains :

- *Distributed Optimizer* is an `extensible action` which establishes the adopted CNN parallelism strategy. The default supported approach is the *Ring-Allreduce* algorithm. However, it is possible to adopt another approach (e.g, *Parameter Server*) strategy, etc.). An example of the required modifications to change the parallelism strategy is detailed in the next section.
- *Training Checkpoints* is an `extensible action`. It enables the Framework's user to set up the custom *TensorFlow-based* training checkpoints.

[7] more informations at https://www.tensorflow.org/guide/using_gpu

- *Hyperparameters Injection* is a `non-extensible action` training hyperparameters. It specifies the user-specific training hyperparameters.

4.3.5.3 Module Details: Injection Module

The *injection module* consists of the following three of `non-extensible actions` :

- The *Task Definition* is a `non-extensible action` which determines the CNN-based image processing task (e.g., segmentation or classification).
- The *Architecture and Dataset Injection* are `non-extensible actions`. They enable an easy loading of the CNN architecture from its corresponding config file alongside with the training dataset path.

4.3.6 Component Detail: The Training Config File

As stated previously, the *training config file* defines the control flow of the system. In particular, it contains:

- The *CNN-Parallelism-Generator* structure definition and the interaction policy of its inner modules.
- The CNN description.
- The CNN training hyperparameters [128].
- The training dataset metadata (e.g., the training data file system location path, the format)

Listing 4.1 shows an example of a *training config file* of Auto-CNNp for an image segmentation use case. The *config file* defines the final shape of the *CNN-Parallelism-Generator*: (1) The training runtime environment is customized (e.g., we consider local rank strategy for the device placement and soft placement as memory allocation approach) (2) We adopt the default supplied *Ring-Allreduce* CNN parallelism approach and extend the training checkpoints with a specific plugin. (3) We define the training, validation and test datasets alongside with the CNN architecture (through python *keras-based* CNN description) and the training hyperparameters that will be loaded/injected into their adequate location in the *CNN-Parallelism-Generator* structure.

```
<system_config>
  <task name="imaging_segmentation">
    <parallelism-degree>3</parallelism-degree>
    <CNN_archi><path>/archi/U-Net.py</path></CNN_archi>
    <CNN-Parallelism-Generator>
      <module name="train_env_def">
        <action class="non_extensible" type="device_placement">
          <value>local_rank</value>
        </action>
        <action class="non_extensible" type="memory_allocation">
          <value>soft_placement</value>
        </action>
      </module>
      <module name="train_srategy_def">
          <action class="extensible" type="dist_strategy">
            <value>ring_allreduce</value>
          </action>
            <action class="extensible" type="Tr_Checkpoint">
            <value name="LRSchedule">
              <path>/data/checkpoint/ckpt1.py</path>
            </value>
          </action>
      </module>
    </CNN-Parallelism-Generator>
    <data>
       <train><path>/data/brain-train</path></train>
       <valid><path>/data/brain-validation</path></valid>
       <test><path>/data/brain-test</path></test>
    </data>
    <hyperparameters>
        <property name="lr">1e-5</property>
        <property name="optimiser">SGD</property>
        <property name="loss">dice</property>
        <property name="minibarch_size">10</property>
        <property name="start_epoch">0</property>
        <property name="end_epoch">120</property>
    </hyperparameters>
  </task>
</system_config>
```

Listing 4.1 – Training config file example

4.4 Evaluation

In this section, We first introduce our experimental environments. Afterwards, we conduct a (1) quantitative and (2) qualitative evaluation of our proposal.

4.4.1 Experimental Environments

4.4.1.1 Hardware

We accomplished the distributed training experiments on the Nancy Grid'5000 [8] testbed site. The experiments were conducted on Grele GPU cluster which contains Dell PowerEdge R730 physical machines where each node is equipped with 2 Nvidia GeForce GTX 1080 Ti GPUs. We use the Grid'5000 Network File System (NFS) to share the training dataset and the *CNN-Parallelism-Generator* component between all training agents. The nodes are interconnected using *InfiniBand* [177] high-speed interconnect.

4.4.1.2 Software

We have chosen *Python* as a programming language for Auto-CNNp reference implementation. Indeed, Auto-CNNp prototype is concurrently built on top of *Tensor-Flow* and *Keras* deep learning libraries. We take advantage also of the *Horovod* [171] implementation of the *Ring-Allreduce* algorithm in order to introduce the latter adopted synchronous data parallelism approach. In addition, we consider *Open MPI* [49] implementation of the MPI standard as a communication library. Also, we use *Beautiful Soup* python library for the xml *config file* parsing. Furthermore, to ensure research reproducibility, the *CNN-Parallelism-Generator* component alongside with its runtime environment are containerized into a debian 9 stretch-based *docker*[8] image. Lastly, we use *docker swarm* for the container orchestration task.

4.4.1.3 Evaluation case studies

To assess our component-based automatic training parallelism approach, we consider U-Net [161] and FCN [127] as a baseline CNN architectures applied to tackle the two different medical imaging use cases the brain tumor segmentation [175] task and the left atrial segmentation task already presented in section 2.3 of Chapter 2.

4.4.1.4 Quantitative evaluation

We assess the cost benefit trade-off of automating CNNs parallelization task through a quantitative assessment approach. In order to do so, we measure the execution time of the Auto-CNNp *engine* for the previsouly mentioned two evaluation case studies tackled by a couple of widely used CNN architectures (U-Net and FCN). For reliability reasons, we run each experimental setup 100 times and we consider the average of the measured execution duration as our reference results. The execution times were measured using the linux `/usr/bin/time`. The outputs of the latter command are threefold: (1) *real* metric stands for the overall execution time from start to finish of the call, (2) *user* metric denotes the amount of CPU time spent in user-mode and (3) *sys* metric is the CPU time spent in kernel mode by the program.

[8]More information can be found at https://www.docker.com/

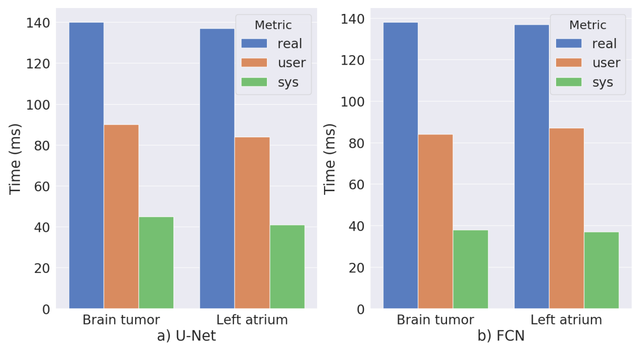


Figure 4.4 – : Auto-CNNp engine execution time evaluation (a) U-Net CNN architecture and (b) FCN CNN architecture for brain tumor and cardiac left atrium case studies segmentation tasks

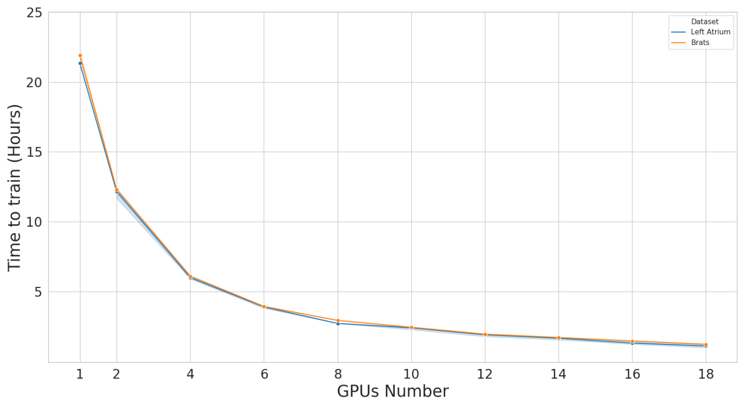


Figure 4.5 – Training time evolution with scale for U-Net CNN architecture for brain tumor segmentation task

Figure 4.4 depicts the evaluation results for the framework's *engine* execution time. It shows that the execution times for the four setups are approximately similar which confirms the generalizability of our proposal. The *engine's real* execution time is about 139 ms which is a negligible time compared to the typical time-consuming CNN training task duration (21 hours and 40 minutes for U-Net and 35 hours and 40 minutes for FCN for a single *Nvidia GTX 1080* GPU based training). The difference between the *real* execution time and the sum of both of *user* and *sys* times is almost 18 ms. It is due to the fact that the engine is blocked on disk I/O during the deployment or update step of the *CNN-Parallelism-Generator* component on all training agents through the NFS server.

Furthermore, in order to evaluate the framework's impact on the CNN-based task performances, we compare the obtained segmentation accuracy when scaling up CNNs training manually with the segmentation accuracy we get after using Auto-CNNp. In order to do so, we perform the distributed training of U-Net CNN architecture for brain tumor and left atrium segmentation on 18 GPUs. As shown in Figure 4.5, we achieved 17.5x speed-up than single-node based training for both segmentation tasks with a segmentation *dice score* of 0.886 and 0.794 for brain tumor and left atrium segmentation respectively. We achieved exactly the same results after performing the U-Net CNN parallelization using Auto-CNNp for both evaluation case studies. Hence, using Auto-CNNp does not impact the performances of the CNN parallelization process compared to the manual approach.

4.4.1.5 Qualitative evaluation

Auto-CNNp intents to offer a high level of abstraction over MPI-based CNN parallelism by instrumenting common routines. In order to do so, the framework is driven throughout a high level *training config file*. To qualitatively evaluate Auto-CNNp reaches, we investigate the impact of the framework in reducing the burden of practically scaling up CNNs training.

We consider the Listing 4.1 as a starting *training config file*. We adopt U-Net CNN architecture and *Ring-Allreduce* parallelism strategy to tackle our first evaluation use case which is the brain tumor segmentation task. After that, we aim to test a different CNN architecture (FCN) to deal with the same evaluation use case. In order to do so, we only need to change the `<CNN_archi>` tag in the *config file* and its related CNN architecture file. Also, if the framework's end user wants to tackle a different segmentation use case using the same initial CNN architecture, he exclusively needs to change the `<data>` tag siblings in the *config file*. All of this shows the easiness with which the framework's user can switch from one training dataset use case to another and/or to test different CNN architectures by minimal code changes.

As mentioned earlier, the adopted *Ring-Allreduce* algorithm constitutes a POC example for our proposal implementation. For instance, it is possible to adopt another CNN parallelism approach. In order to do so, the *Distributed Optimizer* `extensible action` needs to be overwritten alongside with its corresponding *Training Strategy tuning* module. Also, the *Environment Definition* module might require to be replaced since it shares the same *CNN-Parallelism-Generator* component as the *Training Strategy tuning* module. Yet, the *Model Definition* component can be reused to generate the new component-based *CNN-Parallelism-Generator*. Lastly, the operating-system-level environment might need to be adapted but as discussed earlier in subsection 4.3.1, it is out of scope of our proposed framework.

4.5 Conclusion

We introduced and evaluated Auto-CNNp, a framework which permits to automate CNNs distributed training task. Our proposed system offers a high level of abstraction over skill-intensive distributed deep learning by introducing a component-based approach. The latter provides a generic tool that encapsulate many common CNNs parallelism patterns while being flexible sufficiently to be extensible for user-specific customization. The evaluation results of Auto-CNNp on a couple of case studies confirm its validity and transferability to other use cases. Indeed, the quantitative assessment of Auto-CNNp showed an execution overhead of 139 ms which is insignificant compared to the long CNNs training process duration. Also, the qualitative evaluation of Auto-CNNp highlighted its impact in reducing the burden of practically scaling up CNNs training while not affecting the CNN parallelization process compared to the manual approach. This contribution has been published in the IEEE International Conference on Computational Science and Computational Intelligence (DOI: 10.1109/CSCI49370.2019.00179) and IEEE BigData 2019 PEASH (DOI: 10.1109/BigData47090.2019.9006175). The Auto-CNNp source code was the subject an APP deposit.

5 Variability and reproducibility in deep learning for medical imaging segmentation

In the previous chapter, we introduced Auto-CNNp, our component-based framework aiming to abstract the complexity of CNN parallelism process. However, in chapter 3, after presenting R2D2 the first building block of our introduced integrated scalable and component-based deep learning parallelism platform, the practical experimental study we have conducted to investigate the generalization of the linear scaling rule to the imaging segmentation applications have drawn our attention to a number of issues regarding the reproducibility of CNNs training process for medical image segmentation. Indeed, even though deep learning approach outperforms classical machine learning methods for medical imaging segmentation, it is yet still a complex approach and subject to an inherent high range of variability putting into question the reproducibility of the CNNs process training results. In this chapter, we first enumerate and study the sources of variability in deep learning training ecosystem before identifying the main causing issues of CNNs training reproducibility for a particular CNNs training setting. After that, we perform a literature review which intends to further increase our understanding of the main challenges and issues of reproducibility, before drawing some good practices recommendations aiming to alleviate the aforementioned reproducibility issues for medical imaging segmentation DNNs training process [160].

5.1 Introduction

There are manifold sources of variability of the results of deep learning training process. The most influential origins of DNNs training process variability are the followings: The intrinsic variability of the dataset, the stochastic-based optimization process, the different hyperparameters tuning and regularization processes, the training infrastructure (i.e., type training node and the training approach) and the CNNs architecture nature itself. Indeed, as highlighted by the Joelle Pineau's reproducibility checklist[93] provided during the NeurIPS 2019, a clear description of a machine learning model plays a crucial role for reproducibility process. And last but not least, the evaluation strategy of the CNNs segmentation results is another additional variability origin for the disparity of CNNs training process results. Actually, as aforementioned in chapter 3, there are a plethora of segmentation evaluation metrics [193] leading to a numerous possible approaches to compare methods performances.

Hence, a set of research questions related to the variability and reproducibility of DNNs training process arise: (1) Is there enough information in the literature of medical imaging segmentation with DNNs enabling us to correctly reproduce the results? (2) In case enough information has been provided, has the variability aspect of DNNs been considered? (3) Does the evaluation approach adopted to assess the segmentation performances correctly reflects variability?

The remainder of the Chapter is structured as follows: In Section 5.2, we deal with the notion of reproducibility in medical image segmentation. In Section 5.3, we identify and study the sources of variability in deep learning training process before pinpointing the major causing issues of DNNs training reproducibility for particular training setting. In Section 5.4, we conduct a literature review aiming to to better understand and have an overview on the reproducibility practices and issues in neural networks field for medical image segmentation applications. We finally conclude in Section 5.6.

5.2 Reproducibility and evaluation of segmentation in medical imaging

Reproducibility is a hot topic which has always aroused a lot of interest in science [144]. Indeed, multiple articles [7, 188] highlight a potential *crisis of reproducibility* in science's different fields. Also, many scientists have been experiencing failure to reproduce research results [7], i.e., more than 50% for their own research works in medicine physics and engineering fields and above than 75% for other researchers works sharing with them the same research areas.

For the reminder of this manuscript, we consider the following definition of reproducibility introduced in the report of the National Academies of Sciences, Engineering, and Medicine[144]: "*reproducibility means obtaining consistent results using the same input data, computational steps, methods, and conditions of analysis; it is synonymous with computational reproducibility.*" Moreover, the latter report contains also the following recommendation (recommendation 5-1, page 7 of [144]): "*researchers should provide an accurate and appropriate characterization of relevant uncertainties when they report or publish their research.*" We should note here that the aforementioned sources of uncertainties include the stochastic ones among all others.

The reproducibility aspect of a research work might be assessed using different policies. One classic mathematical score to analyse works's reproducibility is the Intra Class Correlation (ICC) proposed by Shrout and Fleiss [178] which compares ntra-individual and inter-individual variabilities degrees. The ICC score value is within an interval of 0 and 1 ,i.e., 0 for poor and 1 for perfect reproducibility respectively. Moreover, the Analysis of the Variance[48] (ANOVA) approach is another statistical tool generalizing the ICC score which quantifies the interaction between repeatability and reproducibility. It provides a collection of tools focusing on the

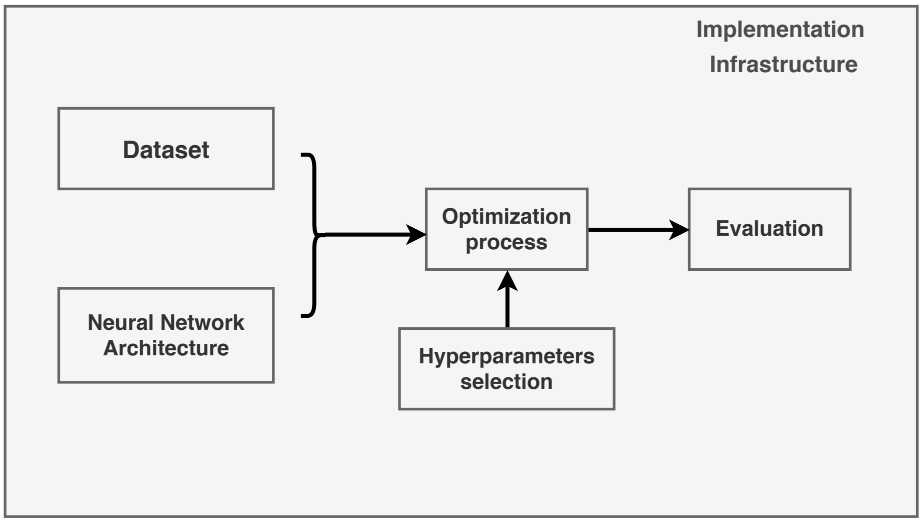

Implementation Infrastructure
Dataset
Neural Network Architecture
Optimization process
Evaluation
Hyperparameters selection

Figure 5.1 – CNNs training ecosystem origins of variability

variability of the means among groups.

5.3 Variability in the deep learning training process ecosystem

In order to identify the the main causing issues of CNNs training reproducibility crises. We first broadly examine in subsection 5.3.1 the main general origins of variability in deep learning training ecosystem before identifying the main reasons behind CNNs reproducibility issues for a particular CNNs training configuration. Figure 5.1 shows the core inherent sources of variability in CNNs training process ecosystem. The different steps of a CNNs training process have been displayed in solid line boxes. They are examined in details in the followings.

5.3.1 Sources of variability in the deep learning training process ecosystem

The deep learning training process ecosystem includes, but not limited to, the following origins of variability :

5.3.1.1 The dataset as a source of variability

Training a CNNs model for segmentation in particular or any machine learning model generally requires splitting the global dataset into three parts. The first one consists in a training dataset used to estimate and fit the model parameters during the training task. It contains both a raw input dataset and its corresponding labels. The second dataset is a validation dataset which is used to estimate the model performances during the training phase in order to tune the training hyperparameters. Furthermore, the validation dataset might be used also as an indicator for an early-stopping procedure during the training stage in order to avoid over-fitting. The last dataset is called a testing dataset. Its main role involves providing an unbiased evaluation of the final CNNs model performances.

The ratio of every dataset part among the global dataset size depends on the dataset size and can deeply impact the degree of expected generalization. For instance, if we consider a basic example where a testing dataset contains one and only one sample, the CNNs performance evaluation is highly dependant of the selected sample. In the same fashion, selecting a few samples from the training dataset leads the model to perfectly fit the training dataset which results to a generalization issue. For this reason, in order to avoid bias in the data selection procedure, many strategies exist (see subsubsection 2.1.1.3 of chapter 2). Among them, the cross-validation approach which consists in dividing the dataset in several folds, which will be assigned to training, validation and testing datasets. Even though the cross validation approach enables taking the dataset variability into consideration during the training phase, it is paradoxical also an additional source of variability itself as there are multiple strategies to put it into practice, e.g., leave-one-out or k-fold. Also, data augmentation is one more regularisation technique in order to avoid overfitting (chapter 2 subsubsection 2.1.1.3) which is also used to alleviate the issue of limited datasets size specially in medical imaging field. However, the data augmentation is also one more source of variability in CNNs training process since there is no consensus on which transformation to perform and the parameters of the transformation are generally randomly chosen.

Also, one of the main source of variability in machine learning is originated from the difference between the observed samples of the dataset and the real distribution of the dataset. Actually, the fact that each learning iteration of the algorithm is made only on a different part of the dataset distribution (which is often randomly shuffled and chosen) can affect the reproducibility and particularly the replication of the results.

Last but no last, the deep-learning-based segmentation applications, in particular, are more complex to reproduce as it have additional sources of variability like : (1) the medical imaging modality (MRI, scanner, echography, ...) and (2) the studied pathology [198]. Furthermore the masks of the segmentation in the raw dataset are usually generated manually which results into some intra- and inter- raters variability.

5.3.1.2 The CNN architecture as a source of variability

As earlier reported in subsection 2.1.2 of Chapter 2, multiple properties have to be taken into account when it comes to conceive a CNN architecture like, including but not limited to, the number, the order, the size and the type of layers (e.g., convolutional, pooling, dense, ...) which makes the number of potential combination of possible architectures infinite.

In practice, the main strategies adopted for CNNs architectures conception are threefold :

1. The first one consists in considering a widely-used CNN architecture which has prac-

tically proved good performance in the literature [126],e.g., U-Net architecture. This method is most frequently adopted for clinical application field. However, even though it does not guarantee having the best possible candidate architecture for a specific segmentation application, it has the advantage of being cost free for the CNN architecture design and selection process.

2. Another approach consists in designing the CNN architecture from scratch by manually handcrafting the future candidate architecture without any without any formal restrictions. It leads to a plethora of possible architectures [126]. However, it is generally not considered in practice as it does not warranty the best architecture and it remains limited and mainly used in the research field.

3. The third strategy is called Network Architecture Search (NAS), is involves automatically creating and designing a CNN architecture [77]. The NAS architecture engineering policy aims to find and select the design of a machine learning model which performs best among all other potential candidates architectures for a specific task trained on a particular dataset. Yet, the NAS is a quite challenging approach because the best potential candidate architecture search process has the drawback of being time and resources intensive. For instance, the NAS policy proposed in [221] has tested 20,000 candidate architectures during 4 days using 500 GPUs.

5.3.1.3 The hyperparameters optimization as a source of variability

As previously mentioned, several hyperparameters have to be tuned throughout the hyperparameters optimization process in order to effectively train a CNN architecture. However, the hyperparameters optimization procedure is an additional source of variability for the CNNs training process as there are multiple different strategies to tune these hyperparameters. The main policies are the followings:

- The manual search approach is the first one. The choices of the hyperparameters is based on the personal experience and judgement of the developers. The training, evaluation stages are repeated in loop until a satisfactory accuracy is reached. Even though this hyperparameters tuning policy limits the space exploration size, it does not guarantee an optimal results since it only a rough approximation of the best hyperparameters set is expected which is highly dependent on the developer's experience.
- Another strategy is the *Grid Search* one where every possible hyperparameters configuration is tested. It has the advantage of optimization the chances to find the optimal set of hyperparameters but it suffers from a high computation cost correlated with the hyperparameters number.
- The *Random Search* is another automatic hyperparameters optimization technique. It involves selecting the potential candidate hyperparameters randomly from the configuration space. James Bergstra and Yoshua Bengio [14] published a paper in 2012 reports

that the random search technique is more effective than the manual and the grid search policies since it provides higher accuracy with less training cycles, for problems with high dimensionality. However, it suffers from unintuitive results as it is difficult to justify the choice of hyperparameters.

- The *Bayesian optimization* [15] is an automated hyperparameters optimization technique based on an automatic space exploration for the optimal set of hyperparameters technique which automatically infers a new combination of hyper parameters based on its previous evaluations. All of this reduces the size of the explored space since it is driven by previous experiences. The cost of space exploration in Bayesian optimization is lower thab Grid and Random search approaches. We should note that there are additional hyperparameters tuning strategies like the automated genetic algorithm [210].

5.3.1.4 The optimization process as a source of variability

DNNs often have millions of parameters, making the optimization process a challenging task due to extremely high-dimensional search space, compared with classic machine learning approaches. The problem of a high-dimensional search space is that adding a unique new dimension dramatically increases the distance between points in this space. which drastically increases the search space [57]. Moreover, the cost function is generally non convex [57] which leads to multiple issues:

1. The presence of local minimums and flat regions with the constrain of the high-dimensionality of the search space.
2. Even though the Stochastic Gradient Descent (SGD) is widely adopted in the DNNs optimization process, there is no guarantee that the it will converge to the best potential solution (even a good local optimum) [118]. Nevertheless, recent works may suggest that perhaps local minimums and flat regions may be less of a challenge than it was previously believed [31, 37, 58]. Indeed, Choromanska et al. [31], show that almost all local minimums have very similar function value to the global optimum, and hence finding a local minimum is good enough. It is important to mention that the latter results were obtained on classification tasks. However, the crucial convolutional step of the segmentation is not considered neither in Choromanska et al. [31] nor in Dauphin et al. [37].
3. The widely-adopted minibatch stochastic gradient descent alongside its varieties are eager to naturally promote the non-determinism aspect of the DNNs training procedure due to their intrinsic stochastic nature [118].

As far as we know, a single conference paper [155] addresses the issue of the stochastic optimization non-determinism aspect in the context of CNNs applied for medical imaging segmentation. Indeed, the authors show differences in the obtained results while training a CNN model multiple times using the same dataset, even though the outcome assessment metrics are not statistically distinct.

5.3.1.5 The evaluation process as a source of variability

Right after the training process comes the evaluation phase. It is a crucial process to assess the segmentation quality of the trained model, yet it is a challenging task. In fact, there is no consensus on the best set of approaches to consider in order to extensively assess the trained CNNs model for segmentation applications in particular. There are various metrics for segmentation quality assessment (e.g., the *dice score*, *Pixel Accuracy*, *Mean Accuracy* etc,.) previously introduced in subsubsection 3.3.2.3 of Chapter 2. Table 5.1[193]. Table 5.1 shows additional evaluation metrics like the the true positive rate (TPR) a.k.a. the *Sensitivity (Sens.)*, the true negative rate (TNR) a.k.a. *Specificity (Spef.)*, and the Average Volume Distance (AVD) (linked with the Hausdorff distance). Every metric deals with a specific aspect of the segmentation [193]. For example, one metric can correctly reflect the good overlapping between a mask of segmentation and a gold standard, but not the contours smoothness. Hence, an adequate segmentation policy which considers the CNNs variability aspect should include several metrics [198, 193].

5.3.1.6 The training infrastructure as a source of variability

The adopted CNNs training approach and its corresponding training infrastructure are a major source of CNNs training variability. In fact, there are multiple approaches in order to bring a CNNs training policy intro practice for both single-based and distributed CNNs training process. For example, a review of the different deep learning implementations characteristics in term of the supported platform and DNNs training strategies, programming language, etc., can be found in [204]. However, as far as we know, there is no prior work which deals with CNNs training reproducibility issues regarding the training infrastructure in particular.

Several options exist in term of training nodes infrastructure types [203], (e.g., CPUs, GPUs, Tensor Processor Unit (TPU)) which is an additional source of CNNs training variability. Indeed, some technical aspects like the memory precision with different size can affect the accuracy of the results [69]. Also, some non-deterministic GPU-native operations can lead to large differences in performance between training runs and thus, non reproducibility of the outcome results [142].

As previously stated, the DNNs training might be a time consuming procedure. Various CNN parallelism techniques exist (see chapter 2 section 2.2) aiming to alleviate this issue where every strategy has its own advantages and drawbacks (see chapter 3 subsubsection 3.3.2.2). Adopting CNNs parallelism techniques might further increase the variability and the non-determinism of the training process which impacts the reproducibility of the outcomes. For instance, during the optimizing phase of the cost function using either a parameters server or

Metric	Equation	Range	Signification
True Positive Rate (TPR)	$\frac{TP}{TP+FN}$	0-1	Sensitivity
True Negative Rate (TNR)	$\frac{TN}{TN+FP}$	0-1	Specificity
Average Volume Distance (AVD)	$max\Big(d_H(Mask, Ground\ Truth), d_H(Ground\ Truth, Mask)\Big)$	≥ 0	Precision

Table 5.1 – Segmentation Metrics. Mask = segmentation mask; Ground Truth = ground truth mask; TP = true positive, voxels correctly segmented as region of interest; TN = true negative, voxels correctly segmented as background; FP = false positive, voxels incorrectly segmented as region of interest; FN = false negative, voxels incorrectly segmented as background. d_H corresponds to the directed Average Hausdorff metric define as $d_H(A,B) = \frac{1}{N}\sum_{a\in A}\min_{b\in B}||a-b||$, where N is the number of pixels/voxels considered.

Ring-Allreduce CNN parallelism techniques, each single work's partial result will aggregated with the remaining training nodes own partial results. Merging these computations values might be done by computing the mean value. Hence, considering a mean value of an already stochastic process (i.e., supposing that we adopt SGD) further increases the non-determinism aspect of the CNNs training procedure.

5.3.2 Main causes of deep learning training reproducibility issues for a particular DNNs training configuration

The previously introduced study which intends to identify and investigate the main origins of DNNs training process variability constitutes a first mandatory step towards having an overview of the general variability sources of deep learning training ecosystem. Hence, in order to have deeper insights about the core causes leading to reproducibility issue for a **particular** DNNs training setting, we start by studying which among these aforementioned variability origins may be fixed during a specific training setting. It is indeed possible to freeze the considered (1) neural network architecture, (2) selected hyperparameters, and (3) the output evaluation procedure throughout multiple runs of a particular integral DNNs training process. Actually, at the end of an hyperparameters optimization procedure, when the final set of training hyperparameters is selected, it is feasible to freeze to adopted DNNs architecture (i.e., since it is related to the selected optimal hyperparameters set which conventionally performed best among all others). It is also possible to consider a unique evaluation procedure for all training execution iterations (e.g., which should include several assessment metrics as recommended in subsubsection 5.3.1.5)

Regarding the optimization process, if we are hoping to reduce the degree of the non-determinism process, the only option we have is to consider a classical gradient descent approach. Yet, it is never used in practice because it is a cumbersome process requiring to take into account all the samples in the training dataset before performing a single update of the model's parameters. That is why the minibatch stochastic gradient descent and its related varieties are commonly adopted in practice (see Figure 5.3) since they only consider a minibatch of several samples to update the model's weights and achieve a considerably close accuracy as the classic gradient decent approach in much less time frame [118]. However, these popular stochastic optimization approaches are eager to naturally promote the non-determinism aspect of the DNNs training procedure due to their intrinsic stochastic nature. Furthermore, even though, it is completely possible to fix the training/validation/test datasets ratios, it is not recommended practically to omit shuffling the training/validation datasets between successive learning iterations because it will reduce the datasets variance which leads to issues of generalization for the trained model which will consequently have more chances to overfit [134].

Regarding the DNNs strategy infrastructure implementation, it is only conceivable to freeze the software part (e.g., used implementation framework and libraries) alongside the training agents hardware properties aspect (e.g., the considered GPU reference). However, it is not the case when a distributed training strategy is adopted due to its intrinsic ability to increase the non-determinism aspect of DNNs training procedure (see subsubsection 5.3.1.6).

Therefore, if is possible in practice to fix the neural network architecture, selected hyper-parameters, and the output evaluation procedure throughout multiple runs of a particular integral DNNs training process without a considerable constraints. Also, to a minimum extent, it is also feasible to alleviate the non-deterministic property of the DNNs training task by adopting the same single hardware training agent and the same datasets ratios. Nevertherless, the stochastic-based optimization approaches with their related training dataset are among the core causes of reproducibility issues for a particular DNNs training setting due to their inherent stochastic properties.

5.4 Literature review

In this section, we will first introduce how the literature review was conducted before broadly introducing its main results.

5.4.1 Methods

As far as we know, there is currently no recognized consensus standard for DNNs reproducibility and evaluation for medical imaging segmentation applications. Through this literature review study we aim to investigate commonly-adopted practices for DNNs applied to medical imaging segmentation field. In order to do so, our introduced literature review is broadly investigating the followings questions:

1. Has the DNN training process been properly described so that the work can be easily and correctly reproduced ?
2. Have all DNNs training process variability sources been taken into consideration ?

3. How the assessment procedure of the outcome models been carried out and the outcome results been reported ?

Our literature review includes a total of 23 papers presented in the survey article [126]. More specifically, in the "Tissue/anatomy/lesion/tumor segmentation" survey section. We have selected the latter survey paper since it is among the most relevant papers that appears in Google Scholar search results (i.e., using keywords 'medical image segmentation neural network' and 'review survey'). [1]. The majority of reviewed articles are recent. The oldest one was published 2014 [68] while the average publication year is 2016. [2]

We investigate the potential variability phenomenons introduced by the considered dataset itself, the optimization strategies and its associated hyperparameters selection selection and tuning process, by the DNNs training implementation infrastructure, and last but nit least, the assessment policies. Throughout our proposed review, we look over for the existing of each inspected studied property, and if so, its corresponding potential value(s) are reported. Concurrently, considering the dataset variability phenomenon, we explore multiple criteria including : (1) whether the DNNs have been tested on several datasets or not, (2) public or private dataset, (3) the number of available data samples, (4) whether the data augmentation regularization policy has been carried out or no, (5) the ratio of training/validation/testing datasets and the potential adoption of a cross validation method. As for the optimization process part, we inspect (1) if the different hyperparameters have been properly reported and (2) the adopted hyperparameters optimization policy (e.g., manual, random search, etc,) in case of available information. We also review the DNNs strategy infrastructure implementation details which a particular attention to the type of used training nodes infrastructures. Concurrently, we explore whether the DNNs training was conducted in a single-node-based or a distributed fashion. Finally, for the assessment process, we study the number, type of the metric and if the variability aspect was taken into consideration.

The assessment of different DNNs output models is performed using the dice score, the true positive rate (TPR), the true negative rate (TNR) and the Average Volume Distance (AVD). We consider various metrics since, as previously stated, each metric has its own drawbacks and advantages and deals with a particular segmentation evaluation aspect [198, 193].

5.4.2 Synthesis of literature review

Our introduced literature review main results are highlighted in Table 5.2, Table 5.3, Table 5.4 and Table 5.5. Indeed, Table 5.2 is mainly dedicated to study the data variability aspect, Table 5.3 is basically studying the variability due to the evaluation procedure, Table 5.4 investigates DNNs training variability sources from the optimization process angle, and Table 5.5 particularly focuses on multiple DNNs training approach infrastructure implementations.

[1]3107 citations for the paper *'A survey on deep learning in medical image analysis'* by the beginning of May 2020

[2]Citations statistics of all reviewed articles by December 2019 are (median = 97, min = 20 , max = 1074)

5.4.2.1 Description of the DNNs training process

In this section, we not only investigate whether or not some methods have been considered, but rather if they have been properly and clearly described.

We found that that only a couple papers [98, 152] (i.e., 9% of the papers) sufficiently describe the DNNs training hyperparameters and the considered dataset with the goal to reproduce their works. On the other hand, a unique paper [172] is missing just one hyperparameter (i.e., the minibatch size), but the code source is publicly available and well documented. Figure 5.2 shows the main statistics regarding both the dataset and the optimization process. Also, Table 5.2 analyses the considered datasets in all reviewed papers. Indeed, all papers properly introduced datasets and their corresponding sizes while 17% of papers do not mention the training dataset ratio. Also, only 57% of reviewed papers clearly report whether or not a validation dataset was adopted, and 35% if a data augmentation regularization technique was performed.

Table 5.4 is principally dedicated to study the selected hyperparameters for the optimization process. We found that 17% completely ignore to describe the optimization process. Also, a unique reviewed paper [68] cites a generic optimization approach name (GDM for Gradient Based Method) without any additional explanations. Yet, the learning rate hyperparameters was generally reported with its related initial values (or range of values) expect of four papers.

Three reviewed papers [4, 18, 22] consider AdaDelta as an optimization strategy, yet, none of them details the learning rate hyperparameters. Also, only a unique paper among them reports the sensitivity ratio for the DNNs approach assessment. Furthermore, more than half of reviewed papers (52% of the papers) do not mention the minibatch size, and just 35% among them clearly precise its value. Furthermore, the dropout method, which is more dedicated for regularization purpose, is present in 61% of the papers (only 43% precises the dropout ratio) and 43 % of papers report considering the Stochastic Gradient Descent (SGD) optimization approach.

Table 5.5, illustrates that 35% of investigated articles do not report at all the DNNs infrastructure implementation. Moreover, 26% of the articles do not provide a clear description such as the kind of considered infrastructure. Furthermore, in case we suppose that a correct GPU's description should include at least (1) the name of the constructor, (2) the class and the memory size, only 30% clearly mention these details. It can also been observed in Table 5.5 that there is no consensus when it comes to report the DNNs training infrastructure.

Even though publicly sharing the source code might be the best way to facilitate the DNNs training procedure, only 17% of reviewed papers have their source code publicly accessible.

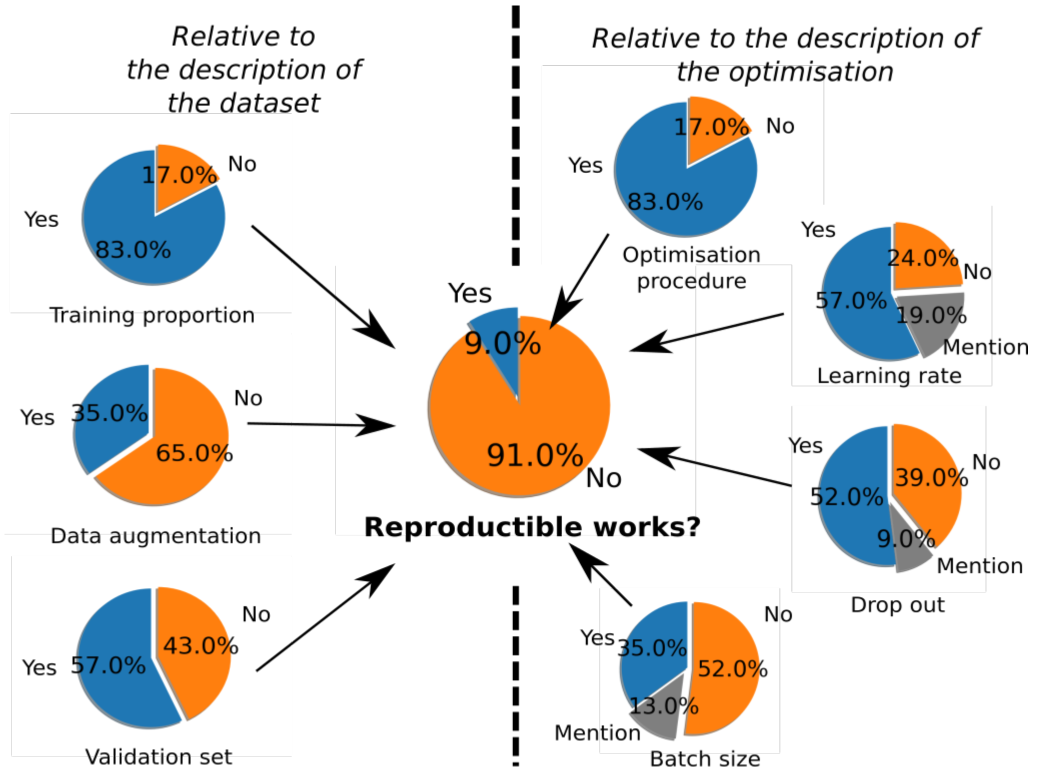


Figure 5.2 – The left side, resp. the right, of the figure is relative to the description of the dataset, resp. the optimisation. The description of the training proportion is present in 83% of the articles. The terms of data augmentation, resp. the validation set, are described in 35%, resp. 57% in the papers. For the optimisation procedure, the name of the optimisation algorithm are missing in 17.4% of the papers. For the hyper parameters learning rate, drop out and batch size, their values are available only in 55%, 52.2% and 34.8% respectively. These coefficients are mentioned in the text without any values in 20%, 8.7% and 13% respectively. Finally, only 9% of the evaluated papers has enough information to be reproducible.

5.4.2.2 Variability in the deep learning training process ecosystem

Throughout a set of selected reviewed articles, we will broadly study the variability properties at the dataset, the optimization, the hyperparameters, the DNNs architectures, the implementations and the infrastructures levels. Some of the mains results of our proposed literature review are detailed in Figure 5.3.

Dataset Variability

Table 5.3 highlights data variability aspect of the DNNs training process. It shows that beyond half of reviewed DNNs techniques are evaluated on more than a single dataset using available public dataset (generally provided during data challenges like BRATS [136]. On the other hand, 30% of the studied papers test their algorithms using only a private dataset.

Furthermore, a limited number of 6 datasets include more than 100 samples, and 4 among them are coming from the same public one which is BRATS. All of this highlights and confirms the difficulty to acquire rare large datasets. For this reason, like previously noted, the data augmentation is a paramount approach for deep-larning-based medical imaging segmentation. However, 13 % of reviewed articles do not clearly detail if either a data augmentation technique or a patches strategy are considered, and if so, and how many patches are selected.

Furthermore, even though cross validation strategies permit to alleviate DNNs training variability originated from the chosen dataset, 52 % of the articles do not perform any cross validation strategies.

Variability in the optimization

One article [98] presents an original strategy to manage the intrinsic variability of the DNNs optimization process: It involves merging the results of three CNNs models which leads to better performances compared to a single model. No other reviewed papers adopt a similar approach.

Hyperparameters variability

As can be seen in Table 5.4 a unique article [73] clearly explains all the tuning procedure steps of its hyperparameters using Grid Search strategy. However, another article [140] claimed to automatically tune its hyperparameters without any clear explanations. In all reviewed articles in Table 5.4, the main considered strategies are threshold: (1) the SGD with Momentum, the RMS-prop and the AdaDelta techniques.

The learning rate is one crucial hyperparameter which varies in a significant way from 10^{-2} to 10^{-4}. For example, a couple of articles [137, 152] conducted an hyperparameters tuning process while varying the learning rate within a range of values. We can notice that the training dataset ratio in Table 5.2 which is another training hyperparameter has a wide range of variability (from 20 % to 95 % of the dataset total size). Indeed, the training dataset ratio is highly dependent on the total raw dataset size which pinpoints the high degree of variability of the training hyperparameters.

Variability in DNNs the architecture

Table 5.3, highlights that the CNNs architectures are undoubtedly the main used type of architectures of DNNs for segmentation applications. The RNNs are adopted in a couple of reviewed papers. In total, both CNNs and RNNs architectures represent 91% of all reviewed DNNs architecture types. We should note that two articles [18, 130] have tested several different DNNs architectures in their framework, i.e., 5 for [18] and 2 for [130]. Meanwhile, only a single article [73] followed a grid-search-based strategy for the architecture-related hyperparameters conception (e.g., kernel, max pooling size for each layer and the number of layers) in order to determine the optimal final CNNs architecture.

Variability in the infrastructure

Table 5.5, shows that several deep learning implementations have been considered. Particularly, widely-used set of deep learning frameworks include Theano [194], Mat-ConvNet [199], Caffe [91] and Pylearn2 [59] alongside a single one in-house adopted framework implementation [22]. 13% of all reviewed articles make use of an additional high-level API (e.g., Keras [30] or Lasagne [40]) in addition to the aforementioned deep learning framework. GPUs are by far the widely-adopted training infrastructure as it was considered in all studied papers. Nevertherless, no article pinpoints adopting a particular DNNs distributed training strategies.

5.4.3 Evaluation of the variability

Almost half reviewed papers consider less than 3 metrics which is in line with the recommended number in [198] (see Table 5.3 and Figure 5.4). Also, only one quarter of papers assess the metrics variability aspect, e.g., using Standard Deviation for instance. In some cases, this can be explained by a particular data challenge evaluation platform context. Meanwhile, the variability is reported using boxplots graphics in the the majority of articles. However, only a couples of articles report the whole evaluation results for every data challenge participant.

Regarding the adopted segmentation evaluation metrics, the *dice score* has been considered in all articles. Yet, there is a large number of other metrics which has been adopted to assess segmentation in the reviewed works. We have enumerated 22 different names of metric throughout our literature review. Yet, some of them are synonyms for the same metric, e.g., True Positive Rate, Recall and Sensitivity.

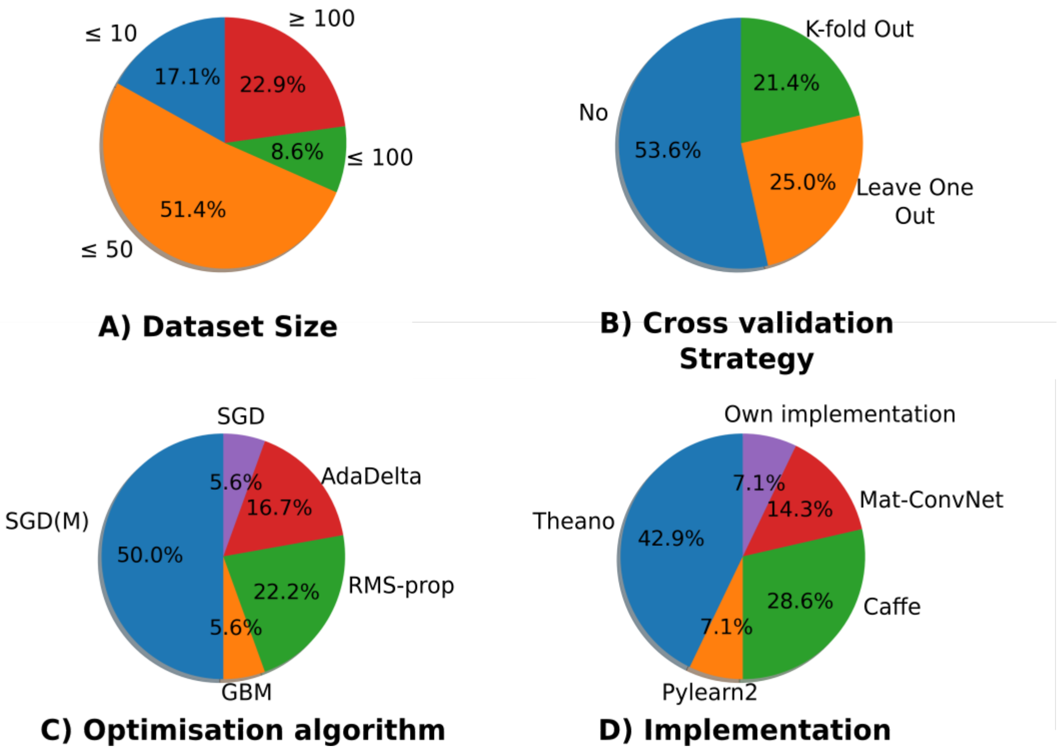


Figure 5.3 – The figure displays 4 different sources of variability. A) there a large variability in the dataset size. 68.5% of the number of samples of the dataset are less or equal than 50. B) In general, no cross-validation strategy is considered (more than 50%). C) There are 5 different optimisation algorithms introduced in the different papers. The main approach is the SGD based on Momentum (SGM(M)). D) 5 different DNNs implementations. The Theano implementation are used in 42.9% of the considered papers, there is no consensus in the implementations.

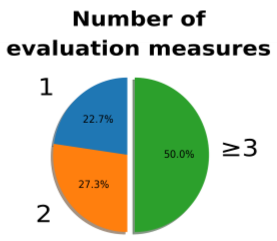


Figure 5.4 – Number of evaluation measures used in each article.

Paper	Training size	DA	DA term	VS term	Training dataset ratio	CV Strategy
Guo (2014) [68]	≤ 50	Patchs	No	No	Not clearly detailed	LOO
de Brebisson (2015) [38]	≤ 50	Patchs	No	Yes	43%	No
Choi (2016) [29]	≤ 50	Patchs	No	Yes	75%	No
Stollenga (2015) [186]	≤ 50	Patchs	Yes	No	50%, 25%	No
Zhang (2015) [217]	≤ 10	Patchs	No	Yes	87.50%	LOO
Andermatt (2016) [4]	≤ 10	Yes	Yes	No	25%	No
Bao (2016) [9]	≤ 10, ≤ 50	Patchs	No	No	50%, 50%	No
Birenbaum (2016) [18]	≤ 10	Patchs	Yes	Yes	80%	LOO
Brosch (2016) [22]	≤ 50, ≤ 50, ≥ 100	Not described	No	Yes	46%, 95%, 80%	No / LOO / No
Chen (2016a) [27]	≤ 10	Not described	No	No	25%	LOO
Ghafoorian (2016b)	≥ 100	Patchs	No	Yes	90%	No
Ghafoorian (2016a)	≥ 100	Patchs	No	Yes	89%	No
Havaei (2016b) [74]	≤ 50, ≥ 100, ≥ 100	Not described	No	Yes	70%	No
Havaei (2016a) [73]	≤ 50, ≥ 100	Patchs	Yes	Yes	46%, 84%	No / 7 FO
Kamnitsas (2017) [98]	≤ 100, ≥ 100, ≤ 50	Patchs	Yes	Yes	80%, 72%, 44%	5 FO
Kleesiek (2016) [104]	≥ 100, ≤ 100	Patchs	Yes	No	50%, 50%	2 FO / 3 FO
Mansoor (2016) [130]	≥ 100	Patchs	No	No	Not clearly detailed	Not clearly detaile
Milletari (2016a) [137]	≤ 100, ≤ 50	Patchs	No	Yes	82%, 33%	No
Moeskops (2016a) [140]	≤ 50, ≤ 50, ≤ 50	Patchs	No	No	20%; 25% ; 33%	LOO / No / No
Nie (2016b) [146]	≤ 10	Patchs	No	No	Not clearly detailed	LOO
Pereira (2016) [152]	≤ 50	Patchs	Yes	Yes	46%, 84%	No
Shakeri (2016) [172]	≤ 50, ≤ 50	Patchs	Yes	Yes	66% , 50%	3 FO / 2 FO
Zhao (2016) [220]	≤ 50	Patchs	No	No	Not clearly detailed	Not clearly detaile

Table 5.2 – The table displays for each article the number of training size, the kind of data augmentation (DA), the presence of the DA term and the validation set (VS) term, the training size ratio and the cross validation (CV) strategy. In the CV can be Leave One Out (LOO) or k Fold Out (k FO).
For example, Kamnitsas et al. [98] presents 3 datasets, with training size ≤ 100, ≥ 100 and ≤ 50. The data augmentation is based on a patch strategy. The training dataset ratio of the 3 datasets are 80%, 72% and 44%. Finally the authors used 5 Fold Out for the CV strategy.

Paper	NN	Nb DS	Dataset type	Type of Meas.	Nb of Meas.	Var. of Meas.
Guo (2014) [68]	SAE	1	Private	DC	1	Values
de Brebisson (2015) [38]	CNN	1	Public	DC	1	No
Choi (2016) [29]	CNN	2	Public	DC, P, R	3	Values, Graph
Stollenga (2015) [186]	RNN	2	Public	DC, MHD, AVD	3	No
Zhang (2015) [217]	CNN	1	Private	DC, MHD	2	Values, Graph *
Andermatt (2016) [4]	RNN	1	Public	DC, MHD, AVD	3	No
Bao (2016) [9]	CNN	2	Public	DC, VD, SD, TPR, FPR	1	No
Birenbaum (2016) [18]	CNN	1	Public	DC,Score	2	No
Brosch (2016) [22]	CNN	3	2 Public & Private	DC, AVD, LTPR, LFPR	4	Graph
Chen (2016a) [27]	CNN	1	Public	DC, MHD, AVD	3	No
Ghafoorian (2016b)	CNN	1	Private	DC, AUC	2	Graph
Ghafoorian (2016a)	CNN	1	Private	DC, AUC	2	Graph
Havaei (2016b) [74]	CNN	3	Public	DC,VD,SD,TPR,FPR	5	No
Havaei (2016a) [73]	CNN	2	Public	DC,Sens.,Spe	3	Graph
Kamnitsas (2017) [98]	CNN	3	Private &2 Public	DC, P, Sens, ASSD, HD	5	Values, Graph
Kleesiek (2016) [104]	CNN	4	3 Public & 1 Private	DC,Sens.,Spe	3	Values, Graph
Mansoor (2016) [130]	SAE	1	Private	DC, ALSD	2	Values, Graph
Milletari (2016a) [137]	CNN	2	Private	DC, MDEC, FR	3	Graph
Moeskops (2016a) [140]	CNN	3	Public	DC, MSD	2	Values, Graph
Nie (2016b) [146]	CNN	1	Private	DC	1	Values *
Pereira (2016) [152]	CNN	2	Public	DC, PPV, Sens	3	Graph
Shakeri (2016) [172]	CNN	2	Public & Private	DC, HD, CMD	3	Graph
Zhao (2016) [220]	CNN	1	Public	DC	1	Graph

Table 5.3 – The table displays the different NN models, kind of datasets (number of datasets, noted Nb DS, and the kind of dataset (public or private) and the kind of evaluation (type, number and variability of measures). For the type of measures, DC = Dice Coefficient, P = Prediction, R = Recall, MHD = Modified Hausdorff Distance, AVD = Average Volume Distance, TPR = True Positive Rate, FPR = False Positive Rate, AUC = Area Under the Curve, Sens. = Sensitivity, Spe. = Specificity. The variability of the measures corresponds to the presence of the standard deviations values or displays in a graph. The (*) means that the values for all subjects are reported.
For example, the article written by Kamnitsas et al. [98] is based on CNN. Their models are evaluated on 3 datasets where one are private and two public. To evaluate their segmentations, they used the DC, P., Sens., ASSD and HD metrics (5 different metrics). The variability of the measures are displayed on a graph and corresponding values are reported in the text.

Paper	Optimization	HP Handcrafted	Learning rate (V./P.)	Batch size (V./P.)	Drop out (V./P.)
Guo (2014) [68]	GBM	Yes	No/No	No	No
de Brebisson (2015) [38]	SGD (M)	Yes	Yes (0.05) /yes	Yes / yes	No
Choi (2016) [29]	SGD (M)	Yes	Yes (0.001)/yes	No /yes	Yes/yes
Stollenga (2015) [186]	RMS-prop	Yes	Yes (0.01)/yes	No	Yes/yes
Zhang (2015) [217]	SGD (M)	Yes	Yes (0.0001)/yes	No	Yes/yes
Andermatt (2016) [4]	AdaDelta	Yes	omit	No	Yes/yes
Bao (2016) [9]	Not described	Yes	No	No	No
Birenbaum (2016) [18]	AdaDelta	Yes **	omit	No	Yes/yes
Brosch (2016) [22]	AdaDelta	Yes	Sensitivity ratio Yes/yes	No	No
Chen (2016a) [27]	Not described	Yes	No	No	No
Ghafoorian (2016b)	RMS-prop	Yes	No/Yes	Yes/yes	Yes/yes
Ghafoorian (2016a)	RMS-prop	Yes	No/Yes	Yes/yes	Yes/yes
Havaei (2016b) [74]	SGD (M)	Yes	Yes (0.001)/yes	No	No/Yes
Havaei (2016a) [73]	SGD (M)	No (Grid Search)	Yes(0.005)/Yes	No/Yes	Yes/yes
Kamnitsas (2017) [98]	RMS-prop	Yes	Yes(0.0001)/Yes	Yes/Yes	Yes/yes
Kleesiek (2016) [104]	SGD	Yes	Yes(0.00001)/Yes	Yes/Yes	No
Mansoor (2016) [130]	SGD (M)	Yes **	No	Yes/Yes	No
Milletari (2016a) [137]	SGD (M)	Yes	Yes (Range Values)/Yes	Yes/Yes	Yes/yes
Moeskops (2016a) [140]	RMS-prop	No (not explained)	No/Yes	No/Yes	No/Yes
Nie (2016b) [146]	Not described	Yes	No/Yes	No	No
Pereira (2016) [152]	SGD (M)	Yes	Yes (Range Values)/yes	Yes/yes	Yes/yes
Shakeri (2016) [172]	SGD (M)	Yes	Yes(0.01)/yes	No	Yes/yes
Zhao (2016) [220]	Not described	Yes	No	No	No

Table 5.4 – The Table displays the kind of optimization, if the hyper parameters (HP) are handcrafted, the learning rate (the Value (V.) and the presence (P.) of the term), the batch size (the Value (V.) and the presence (P.) of the term), the drop out regularization (the Value (V.) and the presence (P.) of the term) and if the code is open source. The (M) in the optimization column signify that the Momentum algorithm is performed. The ** in the HP Handcrafted means that several architectures of NNs have been tested.
For example, the article written by Kamnitsas et al. [98] used a RMS-prop strategy for optimisation. The different hyper parameters are handcrafted. The learning rate, the batch size and the drop out are mentioned in the text, and their corresponding values are given.

Papers	**Implementation**	**Infrastructure**	**Open Source**
Guo (2014) [68]	Not described	Not described	No
de Brebisson (2015) [38]	Theano	NVIDIA Tesla K40 GPU-12GB	No
Choi (2016) [29]	Mat-ConvNet	GPU (GTX TITAN)	No
Stollenga (2015) [186]	Not described	NVIDIA GTX TITAN X GPU-12GB	No
Zhang (2015) [217]	Not described	Tesla K20c GPU	No
Andermatt (2016) [4]	Caffe	NVIDIA GTX Titan X GPU-12GB	No
Bao (2016) [9]	Not described	Not described	No
Birenbaum (2016) [18]	Keras + Theano	NVIDIA GeForce GTX 980 Ti GPU	No
Brosch (2016) [22]	own implementation	GeForce GTX 780	No
Chen (2016a) [27]	Caffe	NVIDIA TITAN X GPU	Yes (*)
Ghafoorian (2016b) [52]	Theano	Not described	No
Ghafoorian (2016a) [52]	Not described	Titan X card	No
Havaei (2016b) [74]	Keras	Nvidia TitanX GPU	No
Havaei (2016a) [73]	Pylearn2	NVIDIA Titan black card.	No
Kamnitsas (2017) [98]	Theano	NVIDIA GTX Titan X GPU-12GB	Yes
Kleesiek (2016) [104]	Theano	NVIDIA Titan-3GB	No
Mansoor (2016) [130]	Not described	Not described	No
Milletari (2016a) [137]	Caffe	NVIDIA ”Tesla k40” or ”Titan X”-12GB	No
		Test on Nvidia GTX 980-4GB	No
Moeskops (2016a) [140]	Not described	NVIDIA Tesla K40 GPU (**)	No
Nie (2016b) [146]	Caffe	Not described	No
Pereira (2016) [152]	Theano + Lasagne	GPU NVIDIA GeForce GTX 980	Yes
Shakeri (2016) [172]	Mat-ConvNet	Described in github	Yes
Zhao (2016) [220]	Not described	Not described	No

Table 5.5 – In the second column, the different implementations are described (Theano [3], Mat-ConvNet [4], Caffe [5], Keras [6], Pylearn2 [7] and Lasagne [8]). For the infrastructure details, the materials are described as they are referred in the papers. If the global memory is reported in the paper, it will be noted. The last column 'Open Source' shows if the code source is available. The (*) is the code source is not available but a detailed prototype of the algorithm is provided. The (**) corresponds when the infrastructure is detailed in the Acknowledgement section. For example, the article written by Kamnitsas et al. [98] used the Theano implementation on an infrastructure based on a NVIDIA GTX Titan X GPU-12GB. Their code is released ad Open source.

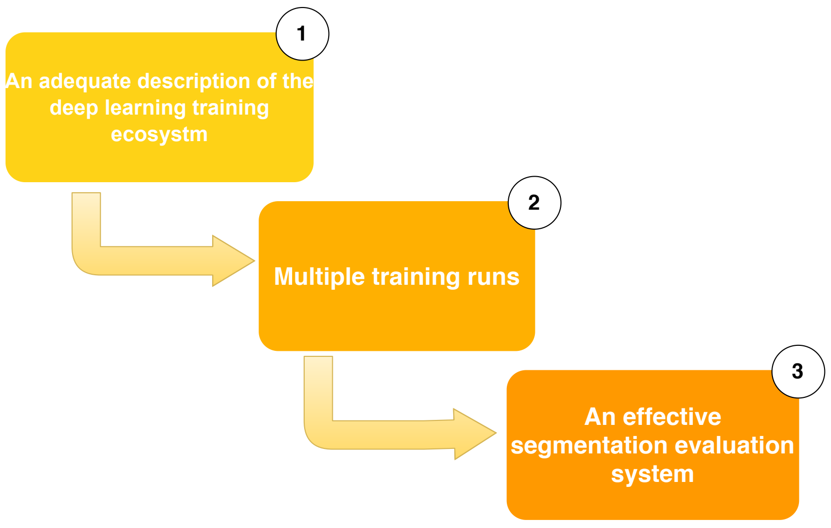

1
An adequate description of the deep learning training ecosystm
2
Multiple training runs
3
An effective segmentation evaluation system

Figure 5.5 – Good practices recommendations for reproducibility for DNNs training process for medical imaging segmentation

5.5 Good practices recommendations for reproducibility for DNNs training process for medical imaging segmentation

Driven by the introduced literature review, as can be seen in Figure 5.5, our recommended good practices focus on three main aspects: (1) an adequate description of the deep learning training ecosystem, (2) reiterating the training process multiple times(4) and an effective evaluation system of the segmentation outputs performances. Indeed, it is crucial to correctly describe all aspects of the the deep learning training ecosystem, going from the DNNs model and its corresponding hyperparameters to the evaluation system, etc. Researcher should clearly describe the DNNs architecture by including, for instance, a schema giving an overview on the introduced architecture specially when it is a complex one.

Regarding the dataset aspect, our recommendations are threefold:

- A complete description of the dataset is required, including the type of acquisitions method (i.e. MRI, scanner, . . .), the images dimensions and the total sample size. If the dataset is publicly available, a downloadable link should be provided.
- Concerning the data preprocessing phase, if the case of data augmentation is considered , the different kinds of transformation must be described and the final number of samples should be included.
- The ratio of the training / validation / testing datasets should be clearly reported. In

case no validation set is considered, this choice should be mentioned and well-argued.

For the optimization approach, the chosen algorithm should be clearly referenced alongside the different adopted hyperparameters values, like the learning rate or the minibatch size, should be reported. In order to alleviate DNNs variability issues We caused by optimization process variability [31, 37, 58], we recommend to perform multiple runs for each training setting while freezing whenever possible variability sources (e.g., fixing the training hyperparameters, the CNN architecture, the evaluation procedure). Indeed, CNN parallelism policies alongside recent powerful training infrastructures (e.g. GPUs) have facilitated reiterating DNNs training process in much shorter time frames. Yet, if several evaluations have been performed, the number of runs should also be provided. On the other hand, regarding the hyperparameters tuning process, the adopted selection should be reported, e.g., manual, grid search. Moreover, the adopted CNNs training approach and its corresponding training infrastructure should be detailed including technical hardware specifications (e.g., considered GPU reference, memory size). Last but not least, the containerization of the experimental environment alongside its specific module and its dependencies and runtime components is a key good practice helping towards easing the DNNs training reproducibility.

Last but not least, since there is no recognized consensus standard for DNNs evaluation for medical imaging segmentation applications, three assessment metrics should be considered at least in order to evaluate the segmentation output performances. Considering that multiples metrics are correlated [193], the segmentation assessment metrics should be chosen wisely [198]. These good practices recommendations are in line with [193].

5.6 Conclusion

This chapter aims to emphasize the complexity and the high degree of variability in the deep learning training ecosystem. In the era of reproducibility crisis [7, 188], in order to be able to pinpoint the main causing issues of DNNs training reproducibility for a specific training setting, we broadly investigate the principal sources of variability in the in deep learning training ecosystem. After that, we conduct a literature review aiming to give us deeper insights about main reproducibility issues of CNNs for medical imaging segmentation.

An important first step would be to extensively describe the entire deep learning ecosystem with enough appropriate information to be easily reproduced. Moreover, it is very important for the researchers to be aware of the variability aspects while building new DNNs models and whenever possible assess them. Also, particular applications such as segmentation should be considered with their complete specificities.

This variability might be also seen sometimes as a blessing. For instance, merging the results of multiples CNNs models might improve the segmentation accuracy bu hiding and alleviating some abnormalities which leads to more robust solutions in some cases [98].

Finally, since there is no recognized consensus standard for DNNs reproducibility and evaluation for medical imaging segmentation applications, an interesting future work would be introducing an in-depth unified methodology including good practices to be followed by researchers aiming to alleviate reproducibility issues for the segmentation use cases.

This contribution has been published in Scientific Reports peer-reviewed journal [160].

6 Towards fast and accurate large mini-batch CNN parallelism for imaging segmentation applications

We have previously introduced R2D2 and Auto-CNNp, our first contributions and building blocks of our proposed integrated platform in chapter 3 and chapter 4 respectively. Next, based on a set of observations we have made while building the aforementioned solutions, two research questions have arisen : (1) "Does the recent CNNs parallelism techniques generalize to the imaging segmentation applications ?" and (2) "What are the variability sources of the CNNs training process and the reasons behind reproducibility issues for the same particular training setup of a CNNs training". We have investigated the generalizability of recent CNN parallelism techniques to the imaging segmentation applications concurrently in section 3.4 of chapter 3. Afterward, we conducted, in chapter 5, a literacy review to identify the sources of variability in the deep learning training process and pinpoint the reasons behind reproducibility issues.

Our observations confirmed also that there is also some way to go before achieving an effective CNNs training in a distributed fashion with no accuracy loss. Hence, another related research question has emerged : "How can we reduce the segmentation accuracy loss in the CNNs parallelism task?". In the current new chapter, we aim to deal with the degradation of the segmentation accuracy when distributing the CNN training process. In order to do so, we present our fourth and final contribution which consists of introducing a guideline including a set of recommendations and directives helping researchers during the decision-making process of the training phase of CNNs with the goal to achieve CNN parallelism without segmentation accuracy loss.

6.1 Introduction

It is generally acknowledged that the human decision-making process is a complex procedure which is subject to several flaws. In fact, it is a naturally limited, faulty, and biased process [132]. DNNs training task is no exception as it requires a human-based decision-making process in order to effectively train models either on a single or specially when the training is performed in a distributed approach. Indeed, a number of decisions and choices had to be made and several questions commonly arise during a typical DNNs training process. For instance, which training hyperparameter has to be selected first and tuned during the DNNs hyperparameters optimization process? Is there an order for the hyperparameters tuning in DNNs? Based on a current hyperparameters optimization run results, should we increase or decrease the value of the selected hyperparameter for the next run? Actually, selecting the adequate hyperparameters in deep learning is a cumbersome and skill-intensive task which requires solid understanding of deep learning optimization fundamentals [56] and years of experience and expertise to acquire. Moreover, additional factors come into play especially when we adopt a distributed training approach, among them the parallelism degree and the linear scaling rule coefficients for example.

The study of the results of the practical experiments which have been conducted throughout this thesis and particularly in chapter 3, subsection 3.4.3 led us to draw several findings, some of which have not been previously discovered. In particular, the segmentation accuracy loss that we stated when scaling up the CNNs training starting from 12 GPUs. Accordingly, this observation led us also to call into question the extent of the learning rate linear scaling rule to the segmentation task for larger minibatch size/parallelism degree.

Unfortunately, to the best of our knowledge, there is currently no clear methodology aiming to deal with the segmentation accuracy degradation when the CNNs training is performed in a distributed fashion. In this chapter, we intend to tackle this challenge by introducing a novel guideline aiming to alleviate the accuracy loss cost of CNNs parallelism in the context of segmentation applications. Our proposal aims to increase the chances for the researchers to select the most adequate choice within the possible alternatives throughout the CNNs distributed training process. Indeed, our introduced guideline-based approach is mainly built on GreScale, a novel learning rate hyperparameter scaling rule we introduce. The latter is a variety of the classic Facebook's gradual warmup linear scaling approach where we take advantage of the learning rate decay technique in an innovative way. The scenario-based assessment results of our proposed guideline on a couple of medical imaging segmentation case studies are promising. Indeed, following our guideline recommendations, we succeeded to empirically prove the effectiveness of our proposal by completely eliminating the accuracy loss for the U-Net CNN architecture on both case studies. On the other hand, even though our proposed guideline is slightly less effective for the FCN CNN architecture, we succeeded to alleviate the accuracy loss compared to the Lr linear scaling rule with gradual warmup strategy.

The remainder of the chapter is structured as follows: In Section 6.2, we explore some background and related work. In Section 6.3, we present and discuss our proposed guideline while justifying our recommendations. In Section 6.4, we conduct a scenario-based case studies assessment of our proposed guideline in order to evaluate its effectiveness. We finally conclude in Section 6.5.

6.2 Background & Related work

6.2.1 Learning Rate hyperparameter

The Learning Rate (Lr) is a crucial hyperparameter commonly considered as the most important hyperparameter to tune for an effective DNNs models training [56, 12]. Like aforementioned in subsubsection 2.1.1.1 of chapter 2, training a DNN is an iterative global optimization problem where an optimization approach is adopted in order to minimize of loss function with to goal to adjust and find an optimal DNNs parameters configuration. Several optimization approaches may be adopted. The Stochastic Gradient Descent (GD) is a widely-adopted optimization algorithm [19]. For instance, we consider SGD optimization strategy. We denote L_0 the loss function; ∇ the gradient of the loss function; η the learning rate and i the current iteration. We update the parameters x of a DNN by the following Formula [213, 56]:

$$x_t = x_{t-1} - \eta \nabla L_x \tag{6.1}$$

As illustrated in Equation 6.1, the learning rate η controls the extent and speed at which the model learns and trains. The Lr is a positive scalar establishing the step size with which the weights are updated during the training process [56]. In practice, tuning the Lr is a challenging task. As can be seen in Equation 6.1, a too high Lr allows the model to train and learn faster at the expense of achieving a non-optimal final weights and may even diverge [12]. On the other hand, a too small Lr can lead to a slower learning because the model would need much more updates before achieving convergence, but it can permit the model to reach more optimal weights configuration.

Several Lr tuning policies exist for an efficient DNN models training procedure. The constant Lr strategy is the most straightforward one which is generally considered as the default baseline approach. It consists in keeping the Lr hyperparameter value unchanged throughout the whole training process. However, a considerable effort has been made in order to improve the constant Lr policy and push the limits further by proposing a set of alternative strategies. The latters are generally incorporating dynamic Lr methods, which aim to adjust and adapt the Lr rate during the DNNs learning procedure. These various Lr strategies will be investigated in the following.

6.2.2 Dynamic learning rate policies

There is abundant literature about dynamic Lr strategies. They are generally schedule-based and/or adaptive Lr policies. The schedule-based approaches adjust the Lr value following specific schedules during different stages of the training process. For instance, as stated previously, when the SGD optimization strategy approaches a minimum in the loss function with inadequate Lr value, the model may start to suffer from instabilities. The learning rate decay strategy which is also known as learning rate annealing is one approach which intends to tackle this problem by decreasing the Lr which, in its turn, slows down the learning process and hence promotes DNNs training convergence. The Lr decay may be introduced manually during the learning process or throughout an automatically scheduled fashion. There are other Lr decay schedule-based policies, such as, e.g., exponential [51, 124], polynomial [34], staircase [183]. Moreover, the Cyclic Learning Rates (CLRs) approaches [181] are an additional family of schedule-based Lr adjustment strategies involving changing the Lr value within a pre-fixed value interval cyclically throughout every predefined LR stage. On the other hand, adaptive Lr policies supervise and evaluate the performance of the learning strategy and adapt the Lr value according to it. The most popular ones are AdaGrad [43], RMSProp [195], AdaDelta [213], and Adam [103]. The main drawback of Lr-based regularization techniques comes from the correlation between the step-size and the noise during the learning process.

6.2.3 Learning rate linear scaling rule

Even though the previously-mentioned Lr policies are general approaches as they may be adopted to improve the convergence speed of DNNs learning either for single or distributed training, there is another set of approaches dealing also with the Lr value but they are rather specific for CNNs distributed training. However, these approaches bring in another CNNs training hyperparameter into play which is the minibatch size of the CNNs distributed training.

The linear scaling rule is a straightforward Lr schedule technique which scales the learning rate with the batch size linearly. It was proposed by Goayl et al. [61] from Facebook. The main intuition behind the linear scaling policy consists in accelerating the learning process by performing larger steps thanks to a higher Lr value. The LR linear scaling rule consists in the following rule:

"*Multiply the learning rate by k when the mini-batch size is multiplied by k [61].* "

Concurrently, Goyal et al. also introduced a couple of warmup schemas that come alongside with the latter Lr linear scaling rule . The constant warmup schema involves using a low constant Lr only for the first few epochs of training, whereas the gradual warmup policy consists in starting the training with a low initial Lr before progressively increasing it to the target Lr (η = k * η, k: number of training nodes) throughout the first 5 epochs. After the warmup stage, training is continued with the classic Lr policy. The warmup phase is important to alleviate the issue of diverging gradients at the beginning of training due the high Lr value and unstable DNNs during the first stages of training [61]. Goyal et al. successfully managed to train ResNet-50 (see chapter 2 subsubsection 2.1.2.2) using ImageNet [165] dataset with a minibatch size of 8192 on 256 Tesla P100 GPUs in one hour. They used the linear scaling rule alongside a warmup policy and achieved 90% scaling efficiency with no accuracy degradation making their approach considered as the "state-of-the art" recipe for CNNs large batch training. Cho et al. [28] from IBM almost reproduced Facebook's work but using a different communication algorithm. They succeeded to train ResNet-50 in 50 minutes

(1 minutes less than Goyal et al.). However, their approach suffers from 1.3 % accuracy degradation compared to Facebook's work.

6.2.4 Layer-wise Adaptive Rate Scaling (LARS)

It has been observed that the Facebook's Lr linear scaling rule is not effective for mini batches larger than 8,192 as it breaks down [26, 61] if the previously mentioned minibatch size threshold is exceeded for classification tasks. Indeed, in practice, the DNNs training process leans to diverge for higher Lr values correlated with a larger minibatch sizes according to the linear scaling rule which results in a degradation in the validation accuracy [26]. For instance, increasing further the minibatch size of AlexNet to 4K decreases the accuracy to 53.1% from a baseline of (B=256) of 57.6% [207].

You et al. introduced the Layer Adaptive Learning Rates (LARS) approach [207] which aims to address the aforementioned issue. It consists in using a different local adaptive Lr value for each different layer of the DNN based on a trust metric. The latter involves the ratio between the norm of the layer weights and norm of gradients update. Using LARS technique, You et al. scaled up the training of ResNet-50 from a minibatch size of8K to 32K in 20 minutes using 2048 KNL. They also succeeded to train AlexNet with a minibatch size of 32k on ImageNet dataset in 11 minutes using 1024 Skylake CPUs chips [209].

6.2.5 Lr-linear-scaling-based versus adaptive-Lr-based CNN parallelism strategies

It is commonly acknowledged that there is no unique and universal approach that works best on all contexts and cases. Indeed, a number of works in the literature adopt a mixed-solution combining and taking advantage of both (1) Lr-linear-scaling- and (2) adaptive-Lr-based strategies and particularly the Lr decay policy to improve DNNs learning convergence particularity when the training is performed in a distributed fashion. For instance, Goyal et al. were inspired by [76] and adopted a fixed schedule Lr annealing approach by decreasing the Lr by 1/10 at the 30-th, 60-th, and 80-th epochs simultaneously alongside with their introduced Lr linear scaling rule. Furthermore, Smith et al. [183] tried to investigate the relations between The Lr and the batch size hyperparameters during DNNs distributed training. They empirically proved that increasing the batch size has almost the same effect as decaying the learning rate on test accuracy after similar number of training epochs using (SGD) and a number of its varieties. Also, increasing the Lr and scaling the batch size accordingly would decrease the number of parameter updates during the learning process which will result in a better convergence and shorter DNNs training time.

6.3 Towards CNN parallelism without segmentation accuracy loss

The previously introduced building blocks of our integrated CNN parallelism platform empowered us with the suitable tools to conduct a thorough empirical study of CNN parallelism approaches with a particular focus on medical imaging segmentation applications. Firstly, R2D2 toolkit (see chapter 3) and particularly the introduced distributed versions of U-Net and FCN enabled us to reduce the DNNs research cycle duration by training CNNs in less training time, which facilitated performing our empirical study by accelerating testing and exploring novel CNNs parallelism techniques. Furthermore, the real time monitoring platform integrated in R2D2 led us also to supervise the training process, which made it easier for us to investigate the sources of the accuracy degradation when the CNNs training is performed in a distributed fashion. Secondly, the Auto-CNNp component-based framework (see chapter 4) led us to further accelerate setting up a distributed CNNs training process throughout streamlining CNNs parallelism routine tasks in an easy to use and high-level fashion. All of this, enabled us first (1) to call into question the extent of the linear scaling rule to the segmentation task particularly for a high scalability level, before (2) investigating and debugging the sources of the segmentation accuracy loss, with the goal to propose an alternative approach alleviating the aforementioned issue. This section is organised as follows: We first enumerate the sources of accuracy degradation in linear-scaling-based Lr policy. Afterwards, we introduce our novel GreScale Lr scaling rule before detailing our proposed GreScale-based CNN parallelism guideline.

6.3.1 Sources of segmentation accuracy degradation in linear-scaling-based Lr CNN parallelism

As previously detailed in section 3.4 of chapter, we investigated the generalization of the Lr linear scaling rule with a gradual warmup schema to the segmentation task throughout our in-depth experimental study. We believe that the causes behind the observed segmentation accuracy degradation with scale are threefold, in particular :

1. It is commonly stated that the image semantic segmentation task is more complex than the classification one [61]. Indeed, while image classification task involves assigning classes to the entire set of pixels of images, the segmentation task consists in a pixel-level classification of images by assigning a label to every pixel in every image.

2. The segmentation accuracy degradation when the CNNs training is done in distributed fashion might be due to the high Lr value related to linear scaling rule which leads to instabilities in the trained DNN models. Indeed, while monitoring the CNNs training process evolution using the R2D2 integrated supervision platform, we observe a correlation between the raise of the accuracy loss and the corresponding increase of the Lr value during the CNNs training warmup stage. As can be seen in , the CNNs training loss increased suddenly after the 3rd epoch, even though it was starting to converge

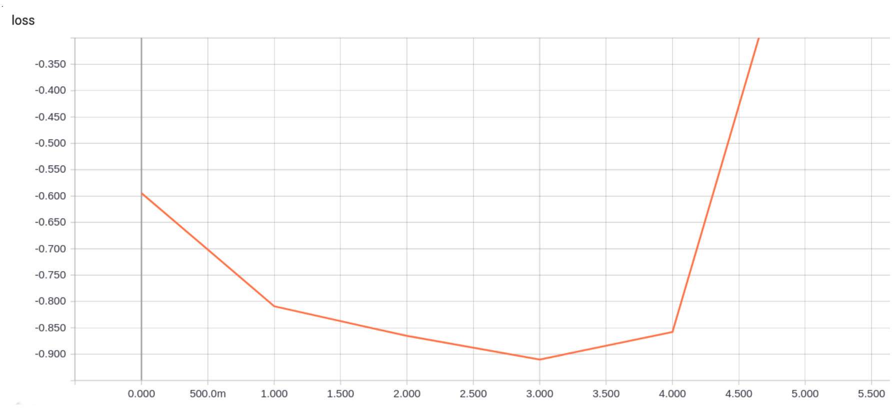


Figure 6.1 – DNNs training diverging before having started to converge due the increase in the Lr value during the training gradual warmup phase.

for the first 3 epochs. We believe that our observations provide additional support to [207, 26, 10, 182] which further substantiate our claim.

3. Even if Goyal et al. argue that the introduced linear scaling policy generalizes well to the segmentation GPUs, and despite they reported that their approach show good generalization behavior to the segmentation transfer task they only assess the generality of the linear scaling rule using Mask R-CNN trained using a maximum of 8 GPUs. We push the boundaries further by investigating the efficiency and consistency of both (1) the Lr linear scaling rule alongside (2) the warmup strategy for higher scalability levels (i.e., as we have reached a total of 18 Nvidia GeForce GTX 1080 Ti GPUs scalability level).

We aim to tackle the previously mentioned challenge by introducing a novel CNNs parallelism guideline specific for segmentation applications and based around a new Lr scaling policy which we denominate GreScale. Our proposed guideline alongside GreScale Lr scaling approach will be introduced in the couple following subsections.

6.3.2 GreScale learning rate scaling rule

Since a high Lr value is most likely the core reason causing the the networks to diverge, and considering that the high Lr value is related to the linear scaling rule and the warmup schema CNN parallelization recipe, one brute-force and straightforward technique to overcome this challenge consists in proposing an alternative approach which adjusts and deals directly with both (1) the Lr high value and (2) the warmup stage duration. Indeed, our GreScale Lr scaling rule is a novel hybrid approach combining both (1) an adjusted version of the Lr linear scaling rule with gradual warmup and (2) an adaptive feed-back-adjusted Lr decay strategy. Figure 6.2 outlines our introduced GreScale learning rate scaling rule. It involves three main phases, in particular :

1. The warmup stage is the first phase of GreScale Lr scaling policy. It directly deals with

the high Lr value by first decreasing and then tuning the initial learning rate as a first step. Moreover, it was reported that the DNNs training is unstable during the early stages of training due to the fact that the network is changing rapidly [61, 207]. Hence, to stabilize the initial training phase, GreScale replaces the classic **static** 5-epochs-based warmup strategy proposed by Goyal et al. [61] by an **adaptive** variety of warmap phase. It consists in gradually increasing the warmup duration starting from 5 epochs while keeping adjusting the warmup period in each training iteration based on the feedback-based guideline which we be presented in the next section.

2. The stabilization stage comes just in the aftermath of the warmup phase. It is a variety of the approach adopted by Goyal et al. and Krizhevsky [106] which starts right after reaching the target Lr of (η = k * η, k: number of training nodes). Our introduced modified stabilization stage has some similarities with Goyal et al. and Krizhevsky Lr scaling policies as it shares with them the same main common objective. The latter consists in accelerating the learning process by performing larger steps thanks to the high Lr value (see subsection 6.2.3). However, the main differences between our proposed version of stabilization stage and the aforementioned Lr scaling policies are twofold : (1) It does not have a fixed period (i.e,. training epochs number) like Goyal et al. technique particularly. It rather has a **dynamic** mutual interdependence duration changing depending on the previous warmup stage duration (e.g., the stabilization duration will decrease admitting that the warmup phase duration increases according to the guideline recommendation for instance). (2) The stabilization stage of our proposal adopts also a smaller and feed-back-adjusted Lr value throughout the stabilization phase in order to come over the issue of DNNs training instabilities.

3. The Learning rate decay stage is the third and final phase of our proposed GreScale Lr scaling policy. Our proposed decay stage has a similar key goal as Goyal et al. **fixed** three-staged Lr decay linear schedule (i.e., 30-th, 60-th, and 80-th epochs) which involves accelerating the learning convergence especially at the final stage of training thanks to the smaller Lr values. However, The GreScale Lr decay phase is instead a **variable adaptive** guideline-based policy. Indeed, the Lr decay coefficient is continuously readjusted each training run depending on the performances of the trained model. Nevertheless, a couple of questions arise: (1) which Lr decay slope should we adopt in order to efficiently accelerate the CNNs learning process ? (2) Should we increase or decrease the Lr after each training run ? The subsequently detailed guideline aims to give answers to these questions.

The default initial durations for each stage are 5 epochs, 4/7 and 3/7 of the total remaining training duration for the warmup, stabilization and Lr decay stages respectively. As previously stated, the warmup and stabilization phases have variable interchangeable durations whereas the Lr decay phase has a fixed one.

Furthermore, even though our introduced GreScale Lr scaling rule seems to have some addi-

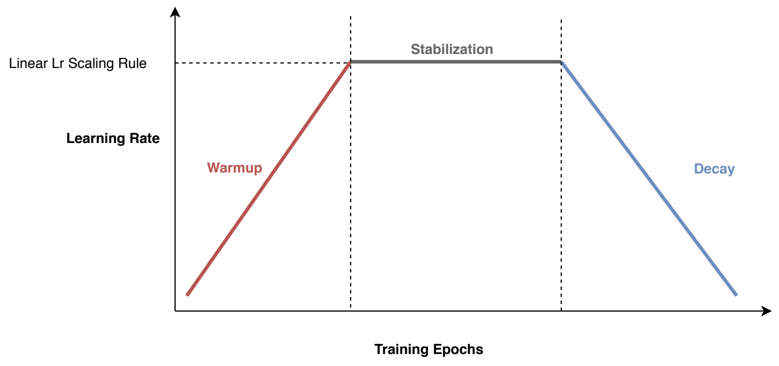


Figure 6.2 – GreScale learning rate scaling rule

tional similarities with some existing approaches, e.g., the hybrid Lr linear scaling annealing policy introduced by Goyal et al. [61], or the LARS approach .Our proposal stands out from the state-of-art approaches because:

1. Our proposal is rather **specific** to CNNs parallelism for **segmentation** applications which is, like previously mentioned (1) a more complex task than the classification one and (2) as far as we know, the CNN parallelism for segmentation has not been yet exhaustively explored and investigated in the literature. Indeed, like previously outlined in subsection 6.3.1, we go beyond the 8 GPUs scalability level threshold reported by [61].

2. We adopt a **dynamic, adaptive, feed-back adjusted and guideline-based** Lr scaling approach different from the classic Lr linear scaling approach based on both **static** (1) warmup period and (2) constant three-staged Lr decay schedule. Indeed, taking into account the observed inner sources of variability of a CNNs training task (see chapter 4), we aim to widen the scope of the supported CNNs segmentation architecture throughout adjusting and fine-tuning the main identified interfering CNNs parallelism hyperparameters (e.g., the initial Lr, the Lr decay coefficient and the warmup period particularly) based on the results and feedback of each CNNs training run and following our guideline recommendations which will be detailed in the followings.

3. Unlike LARS technique where You et al. who assign a different **local** adaptive Lr value for each different layer of the CNN depending on a trust ratio (see subsection 6.2.4), we rather adopt a **global**, guideline-based and adaptive Lr policy. Indeed, our GreScale technique assigns instead a dynamic but global and common Lr for all CNNs layers throughout the learning process.

4. In Adam adaptive Lr policy [103], each parameter in the DNN has an associated specific adaptive Lr value varying from zero (i.e.no update) to a *lambda* maximum value representing an upper limit (i.e., the initial learning rate). Even though the Lr values in Adam are adapted throughout the training process using exponential annealing for instance, it is not possible to go beyond the *lambda* threshold value specially during the last

training epochs which would require setting a very small Lr in order to avoid the DNN model from diverging. In contrast to our proposal where it is possible to adapt the Lr decay coefficient during the final phase of our introduced training protocol according to the specificities and the results of the training case study and **without any restrictions** on the decay coefficient. Moreover, unlike our proposed approach, common adaptive Lr approaches and Adam Lr policy in particular, are not specific for CNNs parallelism techniques. Indeed, our proposed approach is rather a multi-stage feed-back adjusted CNN parallelism **specific** approach.

6.3.3 Guideline for CNN parallelism for segmentation applications

Figure 6.3 illustrates our proposed CNN parallelism guideline. Diamonds in the figure represent important decision points in the process. They show a set of key questions needing answers and helping in the decision-making process in order to achieve a fast and accurate CNN parallelism for semantic segmentation applications. The grey ovals represents the recommended workflow actions/steps to follow to the researchers. Also, the grey rectangles illustrate global stages in the guideline. The latters include a set of task-related steps cooperating towards a particular milestone of the CNNs parallelization guideline.

In addition to the earlier-introduced standard durations for GreScale different stages (see subsection 6.3.2), our proposed Lr scaling policy comes also with a set of default parameters setting which represents the recommended initial training configuration set, before starting to readjust them following up the suggested feed-back-based guideline directives. In particular, based on our observations during our empirical study, and for an optimal and effective use of the GreScale Lr scaling policy, we recommend researchers and developers to adopt :

- SGD as a standard optimization algorithm as it is the most common adopted approach.
- A default warmup phase period of 5 epochs (similar to Goyal and al. technique).
- An initial Lr of $1e-5$. Indeed, following the recommendations of [159] to adopt a Lr value less than 1.0 and greater than $1e-6$, we decided to select an initial Lr close to the minimum recommended value in order to alleviate the high Lr value issue during the distributed CNNs learning process.

Following our introduced methodological flowchart in Figure 6.3, we should adopt in the first place, the aforementioned default parameters settings of the introduced GreScale method. Afterwards, if the parallelism degree is less than 12 training nodes (e.g., GPUs), we directly proceed to the training stage. Otherwise, a GreScale method parameters and training hyper-parameters tuning phase is required. The latter stage includes five ordered workflow actions (e.g., decrease the initial learning rate, increase the learning rate decay coefficient, increase the warmup period, increase the decay phase period and change the optimizer). We start by applying the first workflow action (i.e., decrease the initial learning rate). After that, we

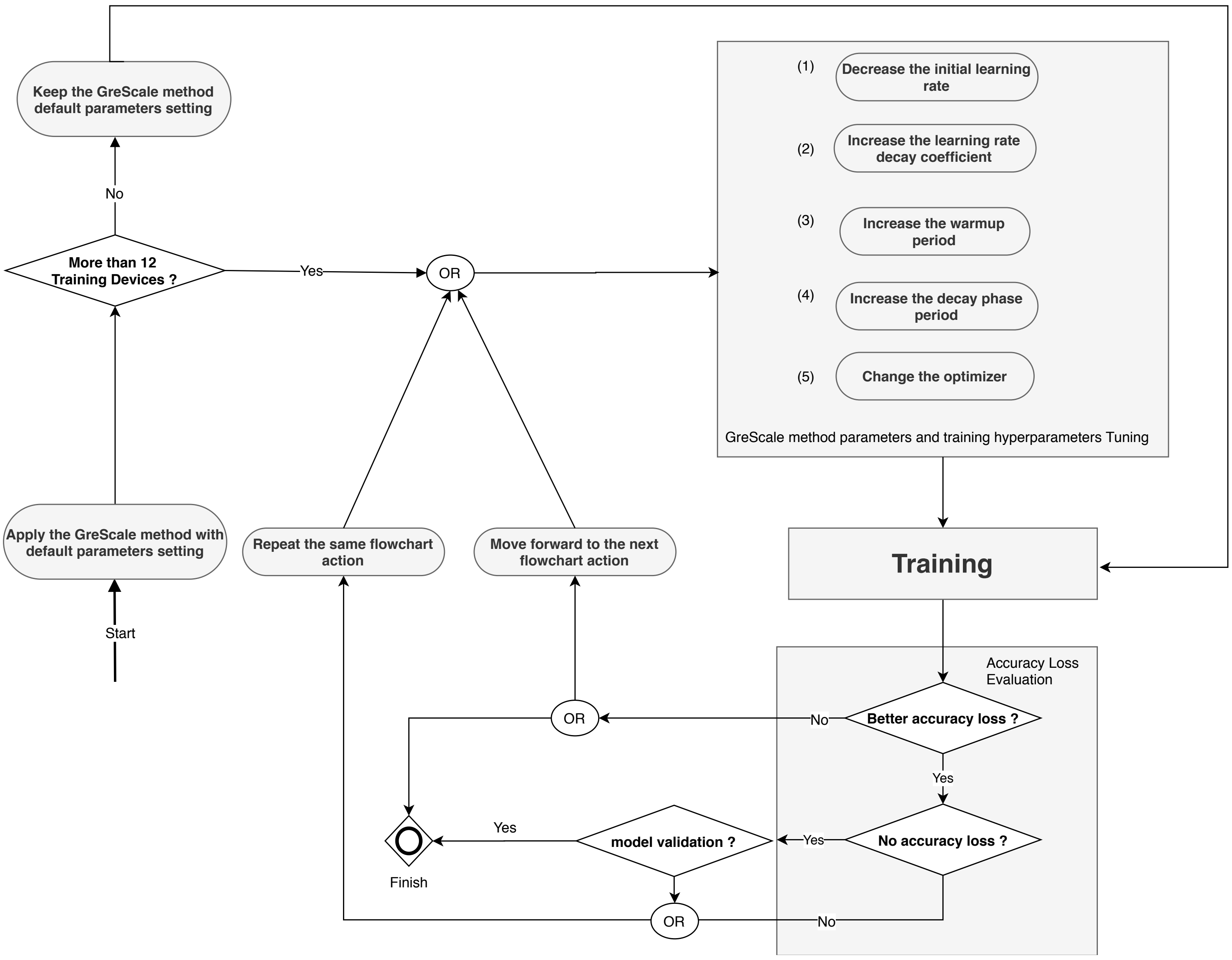

Keep the GreScale method default parameters setting
No
More than 12 Training Devices ?
Yes
OR
(1) Decrease the initial learning rate
(2) Increase the learning rate decay coefficient
(3) Increase the warmup period
(4) Increase the decay phase period
(5) Change the optimizer
GreScale method parameters and training hyperparameters Tuning
Apply the GreScale method with default parameters setting
Start
Repeat the same flowchart action
Move forward to the next flowchart action
Training
Accuracy Loss Evaluation
Better accuracy loss ?
No
OR
Yes
No accuracy loss ?
Yes
model validation ?
Yes
Finish
OR
No

Figure 6.3 – Guideline flowchart representing the steps to follow for a fast and accurate CNN parallelism for semantic segmentation

move forward to the training phase before moving to the accuracy loss evaluation phase of the training model. Afterwards, depending on the trained model accuracy performances, we should either repeat the same parameters tuning flowchart action in case of an improvement in the accuracy degradation or either move forward to the next action when otherwise. We highly recommend to gradually change the GreScale parameters setting with a fixed small steps change during each corresponding workflow action training iteration.

6.4 GreScale & Guideline assessment

We assess our proposed guideline throughout a set of distributed training scenarios of the previously introduced couple of segmentation case studies in chapter 2 (the brain tumor segmentation and left atrium segmentation case studies). We executed a battery of tests for each (1) CNN architecture applied for a particular (2) segmentation case study for (3) multiple parallelism degree following our introduced guideline. Table 6.1, Table 6.2, Table 6.3 and Table 6.4 illustrate U-Net parallelization scenario workflow steps on 18, 16, 14 and 12 GPUs respectively for left atrium segmentation use case. In the same vein, Table 6.5, Table 6.6, Table 6.7 and Table 6.8 also deeply detail every workflow parallelization step for multiple parallelism degree going from 12 to 18 GPUs for brain tumor segmentation case study. On the other hand, Table 6.9 and Table 6.10 present the parallelization scenario steps for FCN architecture for left atrium segmentation on 18 and 16 GPUs based on our introduced CNN parallelism guideline.

For instance, considering the parallelization scenario assessment for U-Net CNN architecture applied to the left atrium segmentation case study on 14 GPUs using our POC parallelism policy, the researchers starts by applying GreScale method with default parameters setting before asking if the parallelism degree is superior to 12 GPUs. The first action in the parameters and training hyperparameters tuning stage involves decreasing the initial Lr value. After going throughout the training and validation phases and since we did not achieve the baseline model performances (i.e., model trained on a single GPU), we repeat the same flowchart action for the next iteration which involves decreasing again the initial Lr value. Once again, since we did not reach the baseline performances and considering that is an improvement in the model segmentation performances compared to the previous iteration we repeat once again the same guideline action for the next iteration which consists in decreasing the initial Lr value. However, we observe a degradation in the model accuracy levels which leads us to move to the next flowchart action for the next iteration which consists in increasing the Lr decay coefficient at this time. We are at the iteration ranked four within our guideline-based parallelization workflow. After going through the training and validation phases once again, we observe that we finally reached a model with segmentation accuracy better than our baseline model segmentation performances. The next training iteration includes the reiteration the training with the same hyperparameters in order to alleviate the issue of CNN training variability and confirm the obtained model performances.

It is important to note that the difference between a couple of successive training phases performances might be evaluated according to several assessment policies depending on researchers needs and priorities. It might be for instance an average of a set metrics or a weighted sum of various segmentation metrics. In our case, since the *PA* and *f.w.IoU* metrics suffer from a unbalanced variability range due to the disproportional size of every class in our segmentation case studies (e.g., the disproportional size between the small left atrium body and large background class size), we decided to consider the three remaining segmentation evaluation metrics ,i.e., *dice*, *MA* and *mean.IoU*, in descending order of priority in order to assess the segmentation performance evolution between two successive training iterations.

As can be seen in Figure 6.5 and Figure 6.4, our introduced guideline-based GreScale Lr scaling policy has been **always** outperforming the baseline model segmentation accuracy performances in all training scenarios with *U-Net* CNN architecture applied for both brain tumor and left atrium segmentation tasks for all assessed parallelism degrees. Indeed, guideline-based GreScale Lr scaling strategy outperforms the segmentation performances of the baseline model by (0.11%,1.60%), (0.37%,1.15%) and (0.15%,0.49%) for the *dice*, *MA* and *mean.IoU* respectively for the left atrium case study and by (0.04%,0.40%), (0.78%,2.69%) and (0.39%,1.89%) for the *dice*, *MA* and *mean.IoU* respectively for the brain tumor segmentation use case.

However, even if Lr linear scaling rule with no warmup strategy surpasses sometimes our GreScale guideline based policy model segmentation accuracy performances (e.g., dice score for 12, 14 and 16 GPUs for left atrium segmentation), it is important to note that our main goal and priority is to eliminate the segmentation accuracy loss **regardless** of the CNN parallelism scenario, which is not the case with other volatile policies depending on an initial fixed hyperparameters.

Moreover, the battery of distributed training scenarios experiments showed us and confirmed that it is possible to surpass the baseline performances in most cases when we go distributed using our introduced guideline or sometimes using other policies (e.g., U-Net for left atrium segmentation of 12, 14 and 16 GPUs) because a constant Lr value throughout the whole training process even if it was very well tuned is rarely the best option to consider.

All these observations further confirm that applying Lr linear scaling rule even though with its corresponding warmup schemas, as it is, suffers from several flaws. Indeed, like aforementioned, beside the fact that adopting the Lr linear scaling rule might lead the model to diverge specially for higher parallelism degree [12, 207], adopting a classic fixed Lr rule is not always the best option to consider since it ignores tuning all remaining different hyperparameters which come also into play throughout CNN parallelism process. Hence, these findings provide additional support and insights for the need to rather adopt an adaptive guideline based Lr scaling policy to not only alleviate the issue of high Lr value with scale but, to confirm also the fact that CNN parallelization task should not, be confined solely to only tuning Lr value but it should additionally consider meticulously turning all corresponding CNN parallelism training hyperparameters throughout a feed-back adjusted guideline allowing us to obtain

Table 6.1 – Parallelization workflow steps on *18 GPUs* of *U-Net* CNN architecture for *left atrium* segmentation

Iteration	Action	Evaluation metrics				
		Dice	PA	MA	mean.IoU	f.w.IoU
0	Apply default parameters					
1	Decrease the initial Lr value	76.02%	99.84%	96.11%	88.93%	99.72%
2	Decrease the initial Lr value	76.09%	99.82%	93.88%	86.62%	99.69%
3	Decrease the initial Lr value	83.74%	99.86%	95.96%	87.96%	99.76%
4	Model validation	82.64%	99.87%	95.37%	89.34%	99.76%
Baseline model performances		81.32%	99.86%	96.11%	89.10%	99.76%

and adaptive parallelism approach that are truly robust comparing to a static Lr linear scaling rule.

Finally, as can be seen in Figure 6.6, even though our introduced GreScale guideline based strategy has not succeeded to achieve the baseline model performances with *FCN* CNN architecture for left atrium segmentation (Table 6.9 and Table 6.10), it has always enabled us to overstep the Lr linear scaling rule with gradual warmup even though the no warmup strategy have achieved in some scenarios better performances which confirms that even if our introduced guideline is an important step towards fast and accurate CNN parallelism there is still some way to go before achieving a universal CNN prallelism approach.

A word of caution: despite the fact that our proposed GreScale rule and its corresponding guideline show very promising results, theses guidelines are cursory, because (1) the deep learning era remains at its beginning and even if a considerable progress has been made recently, it still has a black box aspect where multiple research questions have still not yet been answered, (2) meticulously training a CNN in a distributed fashion is significantly subordinate to several interdependent intrinsic and extrinsic factors involved in the CNNs learning process, e.g., the case studies datasets, the hyperparameters, the adopted parallelism approach, etc. As already indicated, our introduced guideline rather gives suggestions and indications to developers and researchers in order to increase the odds to roughly select the most adequate possible alternative, for an optimal and effective use of our novel adaptive GreScale Lr scaling policy.

Table 6.2 – Parallelization workflow steps on *16 GPUs* of *U-Net* CNN architecture for *left atrium* segmentation

Iteration	Action	Evaluation metrics				
		Dice	PA	MA	mean.IoU	f.w.IoU
0	Apply default parameters					
1	Decrease the initial Lr value	77.89%	99.84%	95.29%	87.37%	99.73%
2	Decrease the initial Lr value	68.44%	99.77%	84.16%	76.88%	99.60%
3	Increase the Lr decay coefficient	81.19%	99.87%	96.43%	89.94%	99.77%
4	Model validation	81.41%	99.87%	96.47%	89.45%	99.69%
Baseline model performances		81.32%	99.86%	96.11%	89.10%	99.76%

Table 6.3 – Parallelization workflow steps on *14 GPUs* of *U-Net* CNN architecture for *left atrium* segmentation

Iteration	Action	Evaluation metrics				
		Dice	PA	MA	mean.IoU	f.w.IoU
0	Apply default parameters					
1	Decrease the initial Lr value	80.75%	99.87%	95.24%	88.58v	99.77%
2	Decrease the initial Lr value	80.93%	99.87%	95.74%	89.41%	99.76%
3	Decrease the initial Lr value	77.14%	99.83%	93.57%	85.85%	99.71%
4	Increase the Lr decay coefficient	81.41%	99.87%	96.47%	89.45%	99.77%
5	Model validation	81.55%	99.87%	97.22%	89.54%	99.77%
Baseline model performances		81.32%	99.86%	96.11%	89.10%	99.76%

Table 6.4 – Parallelization workflow steps on *12 GPUs* of *U-Net* CNN architecture for *left atrium* segmentation

Iteration	Action	Evaluation metrics				
		Dice	PA	MA	mean.IoU	f.w.IoU
0	Apply default parameters	80.02%	99.86%	96.20%	89.09%	99.77%
1	Decrease the initial Lr value	79.88%	99.87%	96.59%	89.22%	99.77%
2	Increase the Lr decay coefficient	82.41%	99.87%	95.62%	89.25%	99.77%
3	Model validation	81.89%	99.87%	95.65%	89.41%	99.77%
Baseline model performances		81.32%	99.86%	96.11%	89.10%	99.76%

Table 6.5 – Parallelization workflow steps on *18 GPUs* of *U-Net* CNN architecture for *brain tumor* segmentation

Iteration	Action	Evaluation metrics				
		Dice	PA	MA	mean.IoU	f.w.IoU
0	Apply default parameters					
1	Decrease the initial Lr value	87.63%	99.66%	93.66%	90.93%	99.36%
2	Decrease the initial Lr value	80.85%	99.36%	92.97%	88.03%	
3	Increase the Lr decay coefficient	88.18%	99.67%	94.58%	91.58%	99.37%
4	Increase the Lr decay coefficient	89.01%	99.71%	95.01%	92.49%	99.45%
5	Model validation	89.08%	99.70%	95.21%	92.39%	99.43%
Baseline model performances		89.01%	99.70%	94.47%	92%	99.42%

Table 6.6 – Parallelization workflow steps on *16 GPUs* of *U-Net* CNN architecture for *brain tumor* segmentation

Iteration	Action	Evaluation metrics				
		Dice	PA	MA	mean.IoU	f.w.IoU
0	Apply default parameters					
1	Decrease the initial Lr value	87.14%	99.68%	94.88%	91.91%	99.38%
2	Decrease the initial Lr value	69.33%	98.93%	85.79%	81.45%	98.02%
3	Increase the Lr decay coefficient	86.25%	99.61%	94.52%	91.37%	99.27%
4	Increase the Lr decay coefficient	69.56%	98.89%	88.05%	80.14%	98.08%
5	Increase the warmup period	88.43%	99.68%	96.89%	93.27%	99.41%
6	Increase the warmup period	88.41%	99.68%	93.83%	91.66%	99.39%
7	Increase the decay phase period	89.43%	99.71%	95.13%	92.52%	99.45%
8	Model validation	89.21%	99.69%	96.31%	93.73%	99.42%
Baseline model performances		89.01%	99.70%	94.47%	92%	99.42%

Table 6.7 – Parallelization workflow steps on *14 GPUs* of *U-Net* CNN architecture for *brain tumor* segmentation

Iteration	Action	Evaluation metrics				
		Dice	PA	MA	mean.IoU	f.w.IoU
0	Apply default parameters					
1	Decrease the initial Lr value	88.61%	99.68%	96.28%	93.13%	99.40%
2	Decrease the initial Lr value	88.24%	99.71%	95.14%	92.49%	99.45 %
3	Increase the Lr decay coefficient	87.29%	99.69%	94.43%	91.93%	99.42%
4	Increase the warmup period	89.52%	99.70%	96.70%	94%	99.44%
5	Model validation	89.05%	99.69%	97.08%	93.74%	99.43%
Baseline model performances		89.01%	99.70%	94.47%	92%	99.42%

Table 6.8 – Parallelization workflow steps on *12 GPUs* of *U-Net* CNN architecture for *brain tumor* segmentation

Iteration	Action	Evaluation metrics				
		Dice	PA	MA	mean.IoU	f.w.IoU
0	Apply default parameters	88.24%	99.69%	95.34%	92.82%	99.42%
1	Decrease the initial Lr value	86.25%	99.61%	94.52%	91.37%	99.27%
2	Increase the Lr decay coefficient	88.97%	99.70%	93.99%	91.99%	99.41%
3	Model validation	89.37%	99.70%	95.28%	92.36%	99.43%
Baseline model performances		89.01%	99.70%	94.47%	92%	99.42%

Table 6.9 – Parallelization workflow steps on *18 GPUs* of *FCN* CNN architecture for *left atrium* segmentation

Iteration	Action	Evaluation metrics				
		Dice	PA	MA	mean.IoU	f.w.IoU
0	Apply default parameters					
1	Decrease the initial Lr value	77.10%	99.45%	88.90%	85.56%	99.07%
2	Decrease the initial Lr value	76.56%	99.45%	88.76%	85.24%	99.06%
3	Increase the Lr decay coefficient	77.02%	99.44%	88.95%	85.26%	99.05%
4	Increase the Lr decay coefficient	77.47%	99.46%	89.08%	85.77%	99.09%
Baseline model performances		79.32%	99.47%	90.86%	87.55%	99.12%

Table 6.10 – Parallelization workflow steps on *16 GPUs* of *FCN* CNN architecture for *left atrium* segmentation

Iteration	Action	Evaluation metrics				
		Dice	PA	MA	mean.IoU	f.w.IoU
0	Apply default parameters					
1	Decrease the initial Lr value	77.28%	99.44%	89.43%	85.46%	99.06%
2	Decrease the initial Lr value	78.01%	99.46%	89.85%	86.28%	99.09%
3	Decrease the initial Lr value	78.34%	99.47%	89.62%	86.31%	99.11%
Baseline model performances		79.32%	99.47%	90.86%	87.55%	99.12%

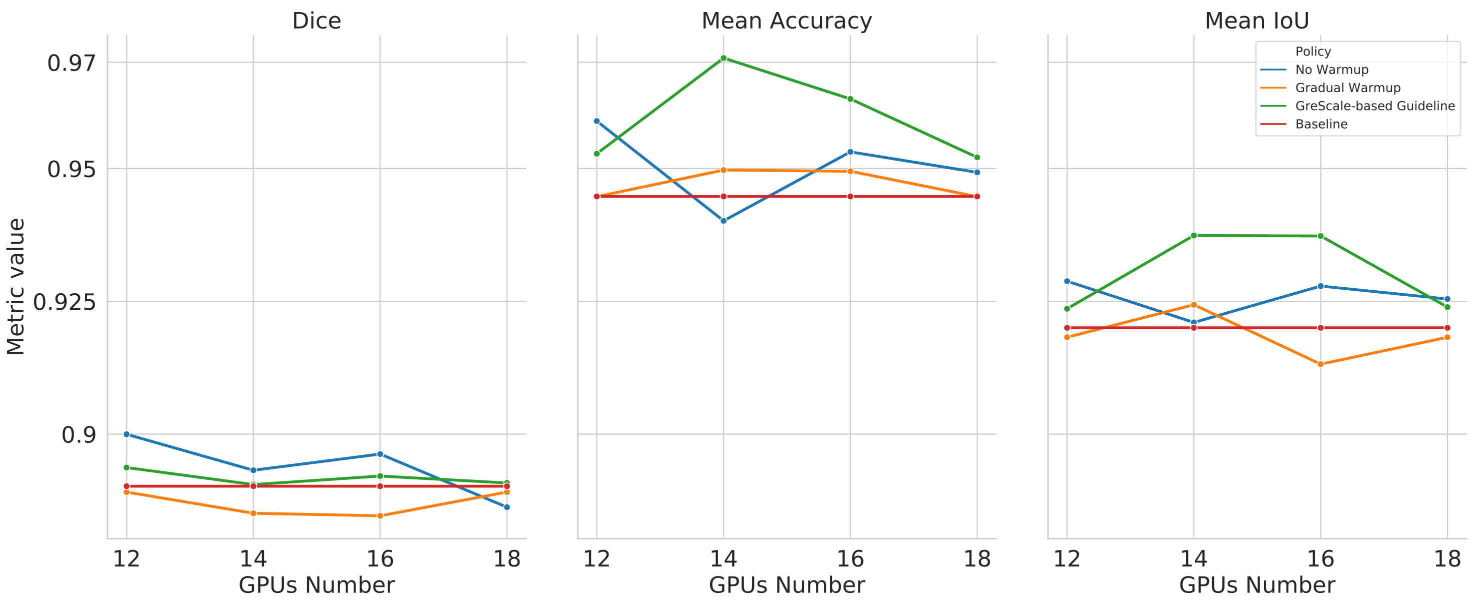


Figure 6.4 – Our introduced Guideline based GreScale Lr rule performances compared to classic Lr scaling polices for *U-Net* CNN architecture applied to brain tumor segmentation task

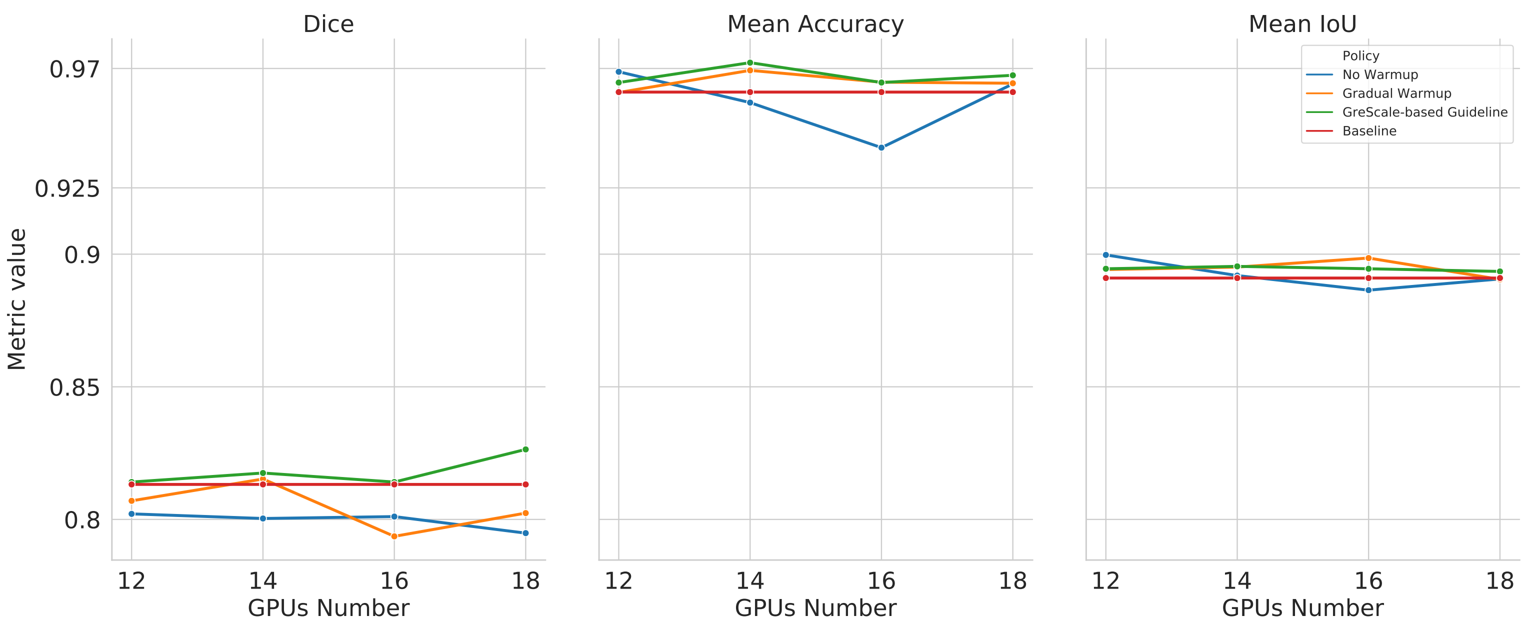


Figure 6.5 – Our introduced Guideline based GreScale Lr rule performances compared to classic Lr scaling polices for *U-Net* CNN architecture applied to left atrium segmentation task

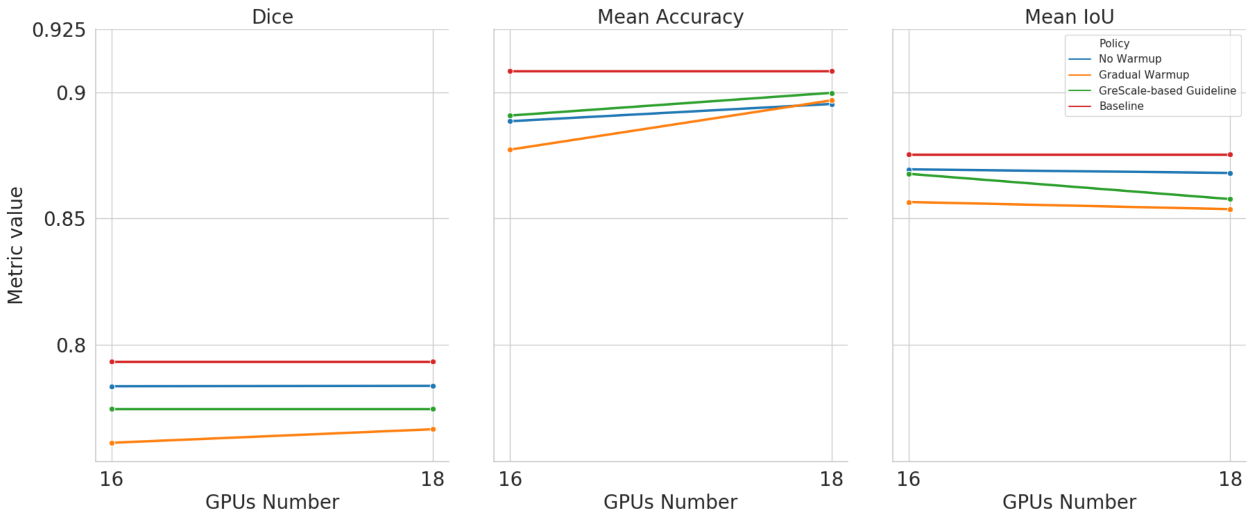


Figure 6.6 – Our introduced Guideline based GreScale Lr rule performances compared to classic Lr scaling polices for *FCN* CNN architecture applied to left atrium segmentation task

6.5 Conclusion

We presented a guideline for the CNNs parallelism process in the context of medical imaging segmentation applications. Concurrently, we explained our guideline main recommended steps motivations. We then evaluated our proposal on a couple of scenario-based segmentation cases studies and provided an empirical evidence of the effectiveness of our proposal. We believe that our work is an important first step which provides the backbone for a promising new way towards a fast and accurate CNN parallelism with no accuracy degradation loss. Our guideline-based Lr scaling strategy has been always outperforming the baseline model segmentation accuracy performances in all training scenarios with *U-Net* CNN architecture tasks for all assessed parallelism degrees. Indeed, it surpasses the segmentation performances of the baseline model by (0.11%,1.60%), (0.37%,1.15%) and (0.15%,0.49%) for the *dice*, *MA* and *mean.IoU* respectively for the left atrium case study and by (0.04%,0.40%), (0.78%,2.69%) and (0.39%,1.89%) for the *dice*, *MA* and *mean.IoU* respectively for the brain tumor segmentation use case. However, nonetheless our proposed guideline is slightly less effective for the FCN CNN architecture, we succeeded to alleviate the accuracy loss compared to the Lr linear scaling rule with gradual warmup strategy.

7 Conclusion

The deep learning era remains at its beginning and even if a considerable progress has been made recently, it still has a black box aspect where multiple research questions have still not yet been answered. In this thesis, we have presented an all-in-one integrated scalable and component-based CNN parallelism solution with a particular focus on medical imagining segmentation. The main building blocks of our introduced platform involves the results built while we were trying to tacked the causally and chronologically related research questions which have arisen during this Ph.D. thesis project timeline.

Our first study introduces R2D2, a scalable deep learning toolkit for medical imaging segmentation which aims to address the following question: How can we build a system which decreases the CNN training time in order to train CNNs models effectively ?
We leveraged distributed optimization approaches in order to build R2D2 toolkit and offer researchers a couple of ready-to-use widely adopted CNNs segmentation architectures. R2D2 also involves an end-to-end processing pipeline gathering the classic deep learning medical imaging main processing milestones in a higher-level fashion in order to offer researchers foundations to quickly prototype and easily discover cutting-edge CNN configurations and architectures. We brought to light R2D2 main concepts and design and detailed its inner buildings components, while justifying our design and implementation choices. We then assessed our scalable toolkit on two distinct concrete medical imaging segmentation case studies to show its effectiveness. The conducted experimental study offers an empirical evidence and further investigates the latest published works. Indeed, R2D2 achieves up to 17.5x and 10.4x speedup than single-node based training of U-Net and FCN respectively with a negligible, yet nonetheless an unforeseen segmentation accuracy degradation with scale.

Secondly, since we stated that putting distributed deep learning into practice is hard and still in its inception, we introduced Auto-CNNp, a component-based framework to automate CNN parallelism which intents to tackle the following research question: How can we abstract the complexity of CNN parallelism process and reduce the gap between skill-intensive deep learning and researchers ?
Our proposed Auto-CNNp system goes one step further than classic manual CNN parallelism

strategy by offering a high level of abstraction over skill-intensive distributed deep learning throughout introducing a novel component-based approach. The latter provides a generic tool that encapsulate many common CNNs parallelism patterns while being flexible sufficiently to be extensible for user-specific customization. Indeed, the component-based strategy brings researchers the main common CNNs parallelism building blocks they need, leaving them the only responsibility of personalizing and configuring them according to their needs. The assessment results of Auto-CNNp on a couple of use cases confirm its validity and transferability to other use cases. Indeed, the quantitative assessment of Auto-CNNp showed an execution overhead of 139 ms which is insignificant compared to the time-consuming CNNs training process. Moreover, the qualitative evaluation of Auto-CNNp showed its impact in reducing and easing the heavy workload of practically distributing CNNs training task while not affecting the CNN parallelization process compared to the manual approach.

Furthermore, answering the couple of previous research questions empowered us with the appropriate tools to put also into question the generalizability of recent CNN parallelism techniques to the imagining segmentation applications. Hence, we conducted thorough practical analysis of the generalizability of the CNN parallelism techniques to the imaging segmentation applications. Concurrently, we conducted in-depth literature review aiming to study the sources of variability in deep learning training process aiming to have deeper insights about the challenges and the issues of the reproducibility. We also propose a set of good practices recommendations aiming to alleviate the identified reproducibility issues for medical imagining segmentation DNNs training process.

Finally, notwithstanding the fact that distributed deep learning strategies are belong the driving forces behind decreasing the CNNs training process time, our observations confirmed that there is still some way to go before training CNNs effectively in a distributed fashion without an accuracy loss cost. However, based on a broad analysis of the results of the already conducted CNN parallelism experimental studies, we introduce a guideline including a set of recommendations and directives helping researchers during the decision-making process of the training phase of CNNs with the goal to achieve CNN parallelism without segmentation accuracy loss. Our introduced guideline-based Lr scaling policy has been always outperforming the baseline model segmentation accuracy performances in all training scenarios with *U-Net* CNN architecture tasks. Indeed, it outperforms the segmentation accuracy of the baseline model by (0.11%,1.60%), (0.37%,1.15%) and (0.15%,0.49%) for the *dice*, *MA* and *mean.IoU* respectively for the left atrium case study and by (0.04%,0.40%), (0.78%,2.69%) and (0.39%,1.89%) for the *dice*, *MA* and *mean.IoU* respectively for the brain tumor segmentation use case. However, nonetheless our proposed guideline is lightly less effective for the FCN CNN architecture, we succeeded to attenuate the accuracy loss comparing to the classic Lr linear scaling rule with gradual warmup strategy.

Even though we believe that our work is an important milestone contributing to the democratization and a better understanding of the rising deep learning field, our proposed parallelism platform is one among the first steps in this area with exciting prospects for future work that follow naturally from this thesis. In the light of the results presented in this thesis, needless to say, many challenges are still not solved yet. Hence, we describe additional research directions that would be interesting to follow, in particular:

- We aim to broaden the spectrum of supported CNNs in R2D2 by not only implementing other scalable CNN-based segmentation architectures, but also through supporting a wider range of medical imaging tasks (e.g., registration, classification). Another promising area of research is to analyze the collected data during the distributed training, with the purpose of getting valuable insights and revealing correlations and hidden patterns within collected datasets. We plan also to shift our distributed training platform from Grid'5000 testbed towards private cloud solutions in order to further evaluate our proposed solution scalability on a production ready environment.
- Regarding Auto-CNNp building block, we plan to widen the number of supported CNN-based tasks by introducing the automated distributed training of other CNN-based applications (e.g., CNN-based text classification task). Also, we are in the process of porting Auto-CNNp in order to support additional platforms and libraries (e.g., *PyTorch* [1]). Lastly, we intent to integrate some infrastructure configuration management tools (e.g., *SaltStack* or *Puppet*) to the Auto-CNNp ecosystem.
- Concerning our introduced guideline-base GreScale Lr scaling rule, many challenges are still not solved yet such as investigating the generalizability and transferability of our proposal to (1) additional CNN parallelism setups and approaches, and (2) other CNN-based applications such as classification suffering also from accuracy degradation with scale. Other interesting futures research direction would be also to study the effectiveness our our proposed guideline on other segmentation applications different from the medical ones we deal with in this thesis. Also, we plan to evaluate and statistically study the percentage of adoption of each step of our methodology compared to all other steps overall. Furthermore, another promising research direction involves introducing automation to our guideline-based context-aware segmentation CNN-parallelism decision-making process. This will lead to take-over and improve the error-prone human decision-making process by an automated, independent and self-supported CNN distributed training approach for image segmentation applications.
- Despite the fact that our introduced parallelism platform building blocks are practically independent, the platform end-user might concurrently take advantage of all of them as they can work seamlessly together for the same purpose. Nevertherless, we believe that going one step further by adding an extra level of synergy between them is a promising and interesting research direction. Indeed, one likely optimistic approach consists in building and including an automatic and self-adaptive decision-making approach to Auto-CNNp framework based on our introduced CNNs parallelism guideline. For instance, the end-user will not have to manually fill the training hyperparameters in

[1] More informations can be found at https://pytorch.org/

Auto-CNNp training config file. Instead, a higher software layer would take advantage of the CNNs parallelism guideline in order automate the process of tuning the adapted training hyperparameters rather than a manual user-centric hyperparameter tuning approach. Moreover, it would be interesting to homogenize more our proposal by fusing both Auto-CNNp and R2D2 engines into a unique solution controller.

- Also, as future work, it would be interesting to widen the spectrum of our POC reference implementation by building and including the main additional CNN parallelism approaches building blocks to our proposal.
- Furthermore, we believe that further research is needed to design a software solution which would either (1) give the main steps and recommendations or (2) automate the change/adjustment of the `module CNN-Parallelism-Generator` which is Auto-CNNp key component. For instance, if an end-user aims to change the Auto-CNNp POC adopted parallelism approach, he actually needs to manually overwrite the *Distributed Optimizer* `extensible action` alongside with its corresponding *Training Strategy tuning* module and also the *Environment Definition* module. Instead, adding a software solution to the stack to automate the aforementioned steps would bring strong added value to our proposed parallelism platform. Furthermore, pushing the boundaries further by studying also the required operating-system-level environment adjustment while switching from one parallelism approach to another would also be a major contribution to the field.

Finally, we should point out that throughout this thesis, we have dealt with a wide variety of heterogeneous fields such as component-based software engineering, distributed systems, mathematical optimization approaches, medical imaging and particularly an emerging black box deep learning domain. Hence, it is also appropriate to emphasize that in a context of a multi disciplinary project like ours, several other factors come into play in term of scientific competitiveness aspect such as the disproportionate physical and technical available resources (e.g., number of GPUs, datasets, research funding) and the scientific notoriety comparing with GAFA *Big Tech* companies which have been having increasing interest towards these emerging fields for the last few years.

Bibliography

[1] Hogwild: A lock-free approach to parallelizing stochastic gradient desc.

[2] R. Adolf, S. Rama, B. Reagen, G. Wei, and D. M. Brooks. Fathom: Reference workloads for modern deep learning methods. *CoRR*, abs/1608.06581, 2016.

[3] T. Akiba, S. Suzuki, and K. Fukuda. Extremely large minibatch sgd: Training resnet-50 on imagenet in 15 minutes. *arXiv preprint arXiv:1711.04325*, 2017.

[4] S. Andermatt, S. Pezold, and P. Cattin. Multi-dimensional gated recurrent units for the segmentation of biomedical 3d-data. In *Deep Learning and Data Labeling for Medical Applications*, pages 142–151. Springer, 2016.

[5] B. B. Avants, N. Tustison, and G. Song. Advanced normalization tools (ants). *Insight j*, 2:1–35, 2009.

[6] M. A. Babyak. What you see may not be what you get: a brief, nontechnical introduction to overfitting in regression-type models. *Psychosom. Med*, pages 411–421, 2004.

[7] M. Baker. 1,500 scientists lift the lid on reproducibility. *Nature News*, 533(7604):452, 2016.

[8] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, and L. Sarzyniec. Adding virtualization capabilities to the Grid'5000 testbed. In I. I. Ivanov, M. van Sinderen, F. Leymann, and T. Shan, editors, *Cloud Computing and Services Science*, volume 367 of *Communications in Computer and Information Science*, pages 3–20. Springer International Publishing, 2013.

[9] S. Bao and A. C. Chung. Multi-scale structured cnn with label consistency for brain mr image segmentation. *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, 6(1):113–117, 2018.

[10] T. Ben-Nun and T. Hoefler. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *CoRR*, abs/1802.09941, 2018.

[11] Y. Bengio. Learning deep architectures for ai. *Found. Trends Mach. Learn.*, 2(1):1–127, Jan. 2009.

[12] Y. Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.

[13] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, Feb. 2012.

[14] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(Feb):281–305, 2012.

[15] J. Bergstra, D. Yamins, and D. D. Cox. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in science conference*, pages 13–20. Citeseer, 2013.

[16] J. Beutel, H. L. Kundel, and R. L. Van Metter. *Handbook of medical imaging*, volume 1. Spie Press, 2000.

[17] H. K. D. H. Bhadeshia. Neural networks in materials science. *ISIJ International*, 39(10):966–979, 1999.

[18] A. Birenbaum and H. Greenspan. Longitudinal multiple sclerosis lesion segmentation using multi-view convolutional neural networks. In *Deep Learning and Data Labeling for Medical Applications*, pages 58–67. Springer, 2016.

[19] L. Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012.

[20] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[21] R. Bramley, K. Chiu, S. Diwan, D. Gannon, M. Govindaraju, N. Mukhi, B. Temko, and M. Yechuri. A component based services architecture for building distributed applications. In *Proceedings the Ninth International Symposium on High-Performance Distributed Computing*, pages 51–59, Aug 2000.

[22] T. Brosch, L. Y. Tang, Y. Yoo, D. K. Li, A. Traboulsee, and R. Tam. Deep 3d convolutional encoder networks with shortcuts for multiscale feature integration applied to multiple sclerosis lesion segmentation. *IEEE transactions on medical imaging*, 35(5):1229–1239, 2016.

[23] A. Brown, S. Johnston, and K. Kelly. Using service-oriented architecture and component-based development to build web service applications. *Rational Software Corporation*, 6:1–16, 2002.

[24] H. Calkins, K. H. Kuck, R. Cappato, J. Brugada, A. J. Camm, S.-A. Chen, H. J. Crijns, R. J. Damiano Jr, D. W. Davies, J. DiMarco, et al. 2012 hrs/ehra/ecas expert consensus statement on catheter and surgical ablation of atrial fibrillation. *Europace*, 14(4):528–606, 2012.

[25] M. Cardoso, M. Clarkson, M. Modat, and S. Ourselin. Niftyseg: open-source software for medical image segmentation, label fusion and cortical thickness estimation. In *IEEE International Symposium on Biomedical Imaging, Barcelona, Spain*, 2012.

[26] K. S. Chahal, M. S. Grover, K. Dey, and R. R. Shah. A hitchhiker's guide on distributed training of deep neural networks. *Journal of Parallel and Distributed Computing*, 137:65–76, 2020.

[27] X. Chen, L. Xu, Y. Yang, and J. Egger. A semi-automatic computer-aided method for surgical template design. *Scientific reports*, 6:20280, 2016.

[28] M. Cho, U. Finkler, S. Kumar, D. S. Kung, V. Saxena, and D. Sreedhar. Powerai DDL. *CoRR*, abs/1708.02188, 2017.

[29] H. Choi and K. H. Jin. Fast and robust segmentation of the striatum using deep convolutional neural networks. *Journal of neuroscience methods*, 274:146–153, 2016.

[30] F. Chollet et al. Keras. https://keras.io, 2015.

[31] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun. The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics*, pages 192–204, 2015.

[32] L. O. Chua and T. Roska. The cnn paradigm. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 40(3):147–156, Mar 1993.

[33] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, pages 1237–1242. AAAI Press, 2011.

[34] V. Codreanu, D. Podareanu, and V. Saletore. Scale out for large minibatch sgd: Residual network training on imagenet-1k with improved accuracy and reduced time to train. *arXiv preprint arXiv:1711.04291*, 2017.

[35] P. Cook, Y. Bai, S. Nedjati-Gilani, K. Seunarine, M. Hall, G. Parker, and D. C. Alexander. Camino: open-source diffusion-mri reconstruction and processing. In *14th scientific meeting of the international society for magnetic resonance in medicine*, volume 2759, page 2759. Seattle WA, USA, 2006.

[36] D. Cox and T. Dean. Neural networks and neuroscience-inspired computer vision. *Current Biology*, 24:921–929, 2014. Computational Neuroscience.

[37] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941, 2014.

[38] A. de Brebisson and G. Montana. Deep neural networks for anatomical brain segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 20–28, 2015.

[39] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng. Large scale distributed deep networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1223–1231, USA, 2012. Curran Associates Inc.

[40] S. Dieleman, J. Schluter, C. Raffel, E. Olson, S. K. Sønderby, D. Nouri, et al. Lasagne: First release., Aug. 2015.

[41] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. *CoRR*, abs/1411.4389, 2014.

[42] J. Dowling. Distributed tensorflow. https://www.oreilly.com/ideas/distributedtensorflow, 2017.

[43] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul):2121–2159, 2011.

[44] M. E. Falagas, E. I. Pitsouni, G. A. Malietzis, and G. Pappas. Comparison of pubmed, scopus, web of science, and google scholar: strengths and weaknesses. *The FASEB journal*, 22(2):338–342, 2008.

[45] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. An updated performance comparison of virtual machines and linux containers. In *2015 IEEE international symposium on performance analysis of systems and software (ISPASS)*, pages 171–172. IEEE, 2015.

[46] J. F. Ferreira, J. L. Sobral, and A. J. Proença. Jaskel: A java skeleton-based framework for structured cluster and grid computing. In *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, volume 1, pages 4–pp. IEEE, 2006.

[47] P. Fingar. Component-based frameworks for e-commerce. *Communications of the ACM*, 43(10):61–61, 2000.

[48] R. A. Fisher. *Statistical methods for research workers.* Genesis Publishing Pvt Ltd, 2006.

[49] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, et al. Open mpi: Goals, concept, and design of a next generation mpi implementation. In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*, pages 97–104. Springer, 2004.

[50] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, and J. G. Rodríguez. A review on deep learning techniques applied to semantic segmentation. *CoRR*, abs/1704.06857, 2017.

[51] A. P. George and W. B. Powell. Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming. *Machine learning*, 65(1):167–198, 2006.

[52] M. Ghafoorian, N. Karssemeijer, T. Heskes, I. W. van Uden, C. I. Sanchez, G. Litjens, F. E. de Leeuw, B. van Ginneken, E. Marchiori, and B. Platel. Location sensitive deep convolutional neural networks for segmentation of white matter hyperintensities. *Scientific Reports*, 7(1):5110, 2017.

[53] A. Gibiansky. Bringing hpc techniques to deep learning. Technical report, Baidu Research, Tech. Rep., 2017, http://andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/. com/bringing-hpc-techniques-deep-learning/. Bingjing Zhang TESTS & CERTIFICATIONS IBM Certified Database Associate-DB2 Universal Database, 2017.

[54] E. Gibson, W. Li, C. Sudre, L. Fidon, D. I. Shakir, G. Wang, Z. Eaton-Rosen, R. Gray, T. Doel, Y. Hu, et al. Niftynet: a deep-learning platform for medical imaging. *Computer methods and programs in biomedicine*, 158:113–122, 2018.

[55] T. Goldstein and S. Osher. The split bregman method for l1-regularized problems. *SIAM Journal on Imaging Sciences*, 2(2):323–343, 2009.

[56] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[57] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[58] I. J. Goodfellow, O. Vinyals, and A. M. Saxe. Qualitatively characterizing neural network optimization problems. *arXiv preprint arXiv:1412.6544*, 2014.

[59] I. J. Goodfellow, D. Warde-Farley, P. Lamblin, V. Dumoulin, M. Mirza, R. Pascanu, J. Bergstra, F. Bastien, and Y. Bengio. Pylearn2: a machine learning research library. *arXiv preprint arXiv:1308.4214*, 2013.

[60] N. Gordillo, E. Montseny, and P. Sobrevilla. State of the art survey on mri brain tumor segmentation. *Magnetic resonance imaging*, 31 8:1426–38, 2013.

[61] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017.

[62] A. Graves, A. Mohamed, and G. E. Hinton. Speech recognition with deep recurrent neural networks. *CoRR*, abs/1303.5778, 2013.

[63] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649. IEEE, May 2013.

[64] H. Greenspan, B. van Ginneken, and R. M. Summers. Guest editorial deep learning in medical imaging: Overview and future promise of an exciting new technique. *IEEE Transactions on Medical Imaging*, 35(5):1153–1159, May 2016.

[65] S. Guedria, N. De Palma, F. Renard, and N. Vuillerme. R2d2: A scalable deep learning toolkit for medical imaging segmentation. *Software: Practice and Experience*.

[66] S. Guedria, N. De Palma, F. Renard, and N. Vuillerme. Auto-cnnp: a component-based framework for automating cnn parallelism. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 3330–3339. IEEE, 2019.

[67] S. Guedria, N. De Palma, F. Renard, and N. Vuillerme. Automating cnn parallelism with components. In *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 943–948. IEEE, 2019.

[68] Y. Guo, G. Wu, L. A. Commander, S. Szary, V. Jewells, W. Lin, and D. Shen. Segmenting hippocampus from infant brains by sparse patch matching with deep-learned features. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 308–315. Springer, 2014.

[69] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. In *International Conference on Machine Learning*, pages 1737–1746, 2015.

[70] S. Gupta, W. Zhang, and F. Wang. Model Accuracy and Runtime Tradeoff in Distributed Deep Learning:A Systematic Study. *ArXiv e-prints*, Sept. 2015.

[71] A. Harlap, D. Narayanan, A. Phanishayee, V. Seshadri, N. Devanur, G. Ganger, and P. Gibbons. Pipedream: Fast and efficient pipeline parallel dnn training. *arXiv preprint arXiv:1806.03377*, 2018.

[72] M. Havaei, A. Davy, D. Warde-Farley, A. Biard, A. Courville, Y. Bengio, C. Pal, P.-M. Jodoin, and H. Larochelle. Brain tumor segmentation with deep neural networks. *Medical Image Analysis*, 35:18 – 31, 2017.

[73] M. Havaei, A. Davy, D. Warde-Farley, A. Biard, A. Courville, Y. Bengio, C. Pal, P.-M. Jodoin, and H. Larochelle. Brain tumor segmentation with deep neural networks. *Medical image analysis*, 35:18–31, 2017.

[74] M. Havaei, N. Guizard, N. Chapados, and Y. Bengio. Hemis: Hetero-modal image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 469–477. Springer, 2016.

[75] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

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

[77] X. He, K. Zhao, and X. Chu. Automl: A survey of the state-of-the-art, 2019.

[78] G. T. Heineman and W. T. Councill. Component-based software engineering. *Putting the pieces together, addison-westley*, page 5, 2001.

[79] G. E. Hinton. A practical guide to training restricted boltzmann machines. In G. Montavon, G. B. Orr, and K.-R. Müller, editors, *Neural Networks: Tricks of the Trade (2nd ed.)*, volume 7700 of *Lecture Notes in Computer Science*, pages 599–619. Springer, 2012.

[80] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006.

[81] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing. More effective distributed ml via a stale synchronous parallel parameter server. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 1223–1231. Curran Associates, Inc., 2013.

[82] E. Hoffer, I. Hubara, and D. Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, pages 1731–1741, 2017.

[83] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[84] D. H. Hubel and T. N. Wiesel. Receptive fields of single neurons in the cat's striate cortex. *Journal of Physiology*, 148:574–591, 1959.

[85] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90, 2007.

[86] F. N. Iandola, M. W. Moskewicz, K. Ashraf, and K. Keutzer. Firecaffe: near-linear acceleration of deep neural network training on compute clusters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2592–2600, 2016.

[87] A. Isin, C. Direkoglu, and M. Sah. Review of mri-based brain tumor image segmentation using deep learning methods. *Procedia Comput. Sci.*, 102(C):317–324, Dec. 2016.

[88] A. Ivakhnenko. Cybernetic predicting devices. Technical report.

[89] K. Janocha and W. M. Czarnecki. On loss functions for deep neural networks in classification. *CoRR*, abs/1702.05659, 2017.

[90] M. Jenkinson, C. F. Beckmann, T. E. Behrens, M. W. Woolrich, and S. M. Smith. Fsl. *Neuroimage*, 62(2):782–790, 2012.

[91] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[92] J. Jiang, B. Cui, C. Zhang, and L. Yu. Heterogeneity-aware distributed parameter servers. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 463–478, New York, NY, USA, 2017. ACM.

[93] P. Joelle. The machine learning reproducibility checklist. https://www.cs.mcgill.ca/~jpineau/ReproducibilityChecklist.pdf.

[94] S. F. Johnsen, Z. A. Taylor, M. J. Clarkson, J. Hipwell, M. Modat, B. Eiben, L. Han, Y. Hu, T. Mertzanidou, D. J. Hawkes, et al. Niftysim: A gpu-based nonlinear finite element package for simulation of soft tissue biomechanics. *International journal of computer assisted radiology and surgery*, 10(7):1077–1095, 2015.

[95] E. Jones, T. Oliphant, and P. Peterson. {SciPy}: Open source scientific tools for {Python}. 2014.

[96] P. Julkunen, R. K. Korhonen, M. J. Nissi, and J. S. Jurvelin. Mechanical characterization of articular cartilage by combining magnetic resonance imaging and finite-element analysis—a potential functional imaging technique. *Physics in Medicine and Biology*, 53(9), 2008.

[97] K. Kamnitsas, E. Ferrante, S. Parisot, C. Ledig, A. V. Nori, A. Criminisi, D. Rueckert, and B. Glocker. Deepmedic for brain tumor segmentation. In *International workshop on Brainlesion: Glioma, multiple sclerosis, stroke and traumatic brain injuries*, pages 138–149. Springer, 2016.

[98] K. Kamnitsas, C. Ledig, V. F. Newcombe, J. P. Simpson, A. D. Kane, D. K. Menon, D. Rueckert, and B. Glocker. Efficient multi-scale 3d cnn with fully connected crf for accurate brain lesion segmentation. *Medical image analysis*, 36:61–78, 2017.

[99] K. Kamnitsas, C. Ledig, V. F. J. Newcombe, J. P. Simpson, A. D. Kane, D. K. Menon, D. Rueckert, and B. Glocker. Efficient multi-scale 3d CNN with fully connected CRF for accurate brain lesion segmentation. *CoRR*, abs/1603.05959, 2016.

[100] B. Karlik and V. Olgac. Performance analysis of various activation functions in generalized mlp architectures of neural networks. *IJAE*, 2011.

[101] X. Ke, K. Sierszecki, and C. Angelov. Comdes-ii: A component-based framework for generative development of distributed real-time control systems. In *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007)*, pages 199–208. IEEE, 2007.

[102] Y. Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014.

[103] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[104] J. Kleesiek, G. Urban, A. Hubert, D. Schwarz, K. Maier-Hein, M. Bendszus, and A. Biller. Deep MRI brain extraction: a 3d convolutional neural network for skull stripping. *NeuroImage*, 129:460–469, 2016.

[105] D. J. Kriegman, P. N. Belhumeur, and A. S. Georghiades. Representations for recognition under variable illumination. In *Shape, Contour and Grouping in Computer Vision*, pages 95–131, 1999.

[106] A. Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

[107] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[108] A. Krogh and J. A. Hertz. A simple weight decay can improve generalization. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 950–957. Morgan-Kaufmann, 1992.

[109] R. Kurzweil. *How to Create a Mind: The Secret of Human Thought Revealed*. Penguin Books, New York, NY, USA, 2013.

[110] I. Larrabide, P. Omedas, Y. Martelli, X. Planes, M. Nieber, J. A. Moya, C. Butakoff, R. Sebastián, O. Camara, M. De Craene, et al. Gimias: an open source framework for efficient development of research tools and clinical prototypes. In *International Conference on Functional Imaging and Modeling of the Heart*, pages 417–426. Springer, 2009.

[111] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back. Face recognition: a convolutional neural-network approach. *IEEE Transactions on Neural Networks*, 8(1):98–113, Jan 1997.

[112] Q. V. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. S. Corrado, J. Dean, and A. Y. Ng. Building high-level features using large scale unsupervised learning. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, pages 507–514, USA, 2012. Omnipress.

[113] Y. LeCun and Y. Bengio. The handbook of brain theory and neural networks. chapter Convolutional Networks for Images, Speech, and Time Series, pages 255–258. MIT Press, Cambridge, MA, USA, 1998.

[114] Y. LeCun, Y. Bengio, and G. E. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[115] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989.

[116] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Intelligent Signal Processing*, pages 306–351. IEEE Press, 2001.

[117] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010.

[118] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[119] L. Lee and S.-C. Liew. A survey of medical image processing tools. 08 2015.

[120] S. Lee, J. K. Kim, X. Zheng, Q. Ho, G. A. Gibson, and E. P. Xing. On model parallelization and scheduling strategies for distributed machine learning. In *Advances in neural information processing systems*, pages 2834–2842, 2014.

[121] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*, pages 6389–6399, 2018.

[122] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su. Scaling distributed machine learning with the parameter server. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 583–598, Berkeley, CA, USA, 2014. USENIX Association.

[123] M. Li, D. G. Andersen, A. J. Smola, and K. Yu. Communication efficient distributed machine learning with the parameter server. In *Advances in Neural Information Processing Systems*, pages 19–27, 2014.

[124] Z. Li and S. Arora. An exponential learning rate schedule for deep learning. *arXiv preprint arXiv:1910.07454*, 2019.

[125] M. Lin, Q. Chen, and S. Yan. Network in network. *CoRR*, abs/1312.4400, 2013.

[126] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. Van Der Laak, B. Van Ginneken, and C. I. Sánchez. A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88, 2017.

[127] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[128] I. Loshchilov and F. Hutter. Cma-es for hyperparameter optimization of deep neural networks. *arXiv preprint arXiv:1604.07269*, 2016.

[129] J. A. Maintz and M. A. Viergever. A survey of medical image registration. *Medical image analysis*, 2(1):1–36, 1998.

[130] A. Mansoor, J. J. Cerrolaza, R. Idrees, E. Biggs, M. A. Alsharid, R. A. Avery, and M. G. Linguraru. Deep learning guided partitioned shape model for anterior visual pathway segmentation. *IEEE transactions on medical imaging*, 35(8):1856–1865, 2016.

[131] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928, Sept 2015.

[132] D. McCaughey and N. S. Bruning. Rationality versus reality: the challenges of evidence-based decision making for health policy makers. *Implementation science : IS*, 5:39, 2010.

[133] A. Mehrtash, M. Pesteie, J. Hetherington, P. A. Behringer, T. Kapur, W. M. Wells III, R. Rohling, A. Fedorov, and P. Abolmaesumi. Deepinfer: Open-source deep learning deployment toolkit for image-guided therapy. In *Proceedings of SPIE–the International Society for Optical Engineering*, volume 10135. NIH Public Access, 2017.

[134] Q. Meng, W. Chen, Y. Wang, Z.-M. Ma, and T.-Y. Liu. Convergence analysis of distributed stochastic gradient descent with shuffling. *Neurocomputing*, 337:46–57, 2019.

[135] B. Menze, A. Jakab, S. Bauer, J. Kalpathy-Cramer, K. Farahani, J. Kirby, Y. Burren, N. Porz, J. Slotboom, R. Wiest, L. Lanczi, E. Gerstner, M.-A. Weber, T. Arbel, B. Avants, N. Ayache, P. Buendia, L. Collins, N. Cordier, J. Corso, A. Criminisi, T. Das, H. Delingette, C. Demiralp, C. Durst, M. Dojat, S. Doyle, J. Festa, F. Forbes, E. Geremia, B. Glocker, P. Golland, X. Guo, A. Hamamci, K. Iftekharuddin, R. Jena, N. John, E. Konukoglu, D. Lashkari, J. Antonio Mariz, R. Meier, S. Pereira, D. Precup, S. J. Price, T. Riklin-Raviv, S. Reza, M. Ryan, L. Schwartz, H.-C. Shin, J. Shotton, C. Silva, N. Sousa, N. Subbanna, G. Szekely, T. Taylor, O. Thomas, N. Tustison, G. Unal, F. Vasseur, M. Wintermark, D. Hye Ye, L. Zhao, B. Zhao, D. Zikic, M. Prastawa, M. Reyes, and K. Van Leemput. The Multimodal Brain Tumor Image Segmentation Benchmark (BRATS). *IEEE Transactions on Medical Imaging*, 34(10):1993–2024, Oct. 2014.

[136] B. H. Menze, A. Jakab, S. Bauer, J. Kalpathy-Cramer, K. Farahani, J. Kirby, Y. Burren, N. Porz, J. Slotboom, R. Wiest, et al. The multimodal brain tumor image segmentation benchmark (brats). *IEEE transactions on medical imaging*, 34(10):1993–2024, 2014.

[137] F. Milletari, S.-A. Ahmadi, C. Kroll, A. Plate, V. Rozanski, J. Maiostre, J. Levin, O. Dietrich, B. Ertl-Wagner, K. Bötzel, et al. Hough-cnn: deep learning for segmentation of deep brain regions in MRI and ultrasound. *Computer Vision and Image Understanding*, 164:92–102, 2017.

[138] F. Milletari, N. Navab, and S. Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *2016 Fourth International Conference on 3D Vision (3DV)*, pages 565–571, Oct 2016.

[139] M. Minsky and S. A. Papert. *Perceptrons: An introduction to computational geometry.* MIT press, 2017.

[140] P. Moeskops, M. A. Viergever, A. M. Mendrik, L. S. de Vries, M. J. Benders, and I. Išgum. Automatic segmentation of mr brain images with a convolutional neural network. *IEEE transactions on medical imaging*, 35(5):1252–1261, 2016.

[141] C. More, L. Colaco, and R. Sardinha. Application of component-based software engineering in building a surveillance robot. In *Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) 2014*, pages 651–658. Springer, 2015.

[142] P. Nagarajan, G. Warnell, and P. Stone. The impact of nondeterminism on reproducibility in deep reinforcement learning. 2018.

[143] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress.

[144] E. National Academies of Sciences and Medicine. *Reproducibility and Replicability in Science.* The National Academies Press, Washington, DC, 2019.

[145] A. Y. Ng. Feature selection, l1 vs. l2 regularization, and rotational invariance. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 78–, New York, NY, USA, 2004. ACM.

[146] D. Nie, L. Wang, Y. Gao, and D. Shen. Fully convolutional networks for multi-modality isointense infant brain image segmentation. In *2016 IEEE 13Th international symposium on biomedical imaging (ISBI)*, pages 1342–1345. IEEE, 2016.

[147] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. *CoRR*, abs/1505.04366, 2015.

[148] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips. Gpu computing. *Proceedings of the IEEE*, 96(5):879–899, 2008.

[149] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, Oct 2010.

[150] P. Patarasuk and X. Yuan. Bandwidth optimal all-reduce algorithms for clusters of workstations. *Journal of Parallel and Distributed Computing*, 69(2):117–124, 2009.

[151] N. Pawlowski, S. I. Ktena, M. C. H. Lee, B. Kainz, D. Rueckert, B. Glocker, and M. Rajchl. DLTK: state of the art reference implementations for deep learning on medical images. *CoRR*, abs/1711.06853, 2017.

[152] S. Pereira, A. Pinto, V. Alves, and C. A. Silva. Brain tumor segmentation using convolutional neural networks in MRI images. *IEEE transactions on medical imaging*, 35(5):1240–1251, 2016.

[153] L. Perez and J. Wang. The effectiveness of data augmentation in image classification using deep learning. *CoRR*, abs/1712.04621, 2017.

[154] N. Pessemier, L. Seinturier, L. Duchien, and T. Coupaye. A component-based and aspect-oriented model for software evolution. 2008.

[155] G. Piantadosi, S. Marrone, and C. Sansone. On reproducibility of deep convolutional neural networks approaches. In *International Workshop on Reproducible Research in Pattern Recognition*, pages 104–109. Springer, 2018.

[156] S. Pieper, B. Lorensen, W. Schroeder, and R. Kikinis. The na-mic kit: Itk, vtk, pipelines, grids and 3d slicer as an open platform for the medical image computing community. In *3rd IEEE International Symposium on Biomedical Imaging: Nano to Macro, 2006.*, pages 698–701, April 2006.

[157] S. Pieper, B. Lorensen, W. Schroeder, and R. Kikinis. The na-mic kit: Itk, vtk, pipelines, grids and 3d slicer as an open platform for the medical image computing community. In *3rd IEEE International Symposium on Biomedical Imaging: Nano to Macro, 2006.*, pages 698–701. IEEE, 2006.

[158] C. E. Rasmussen. *Gaussian Processes in Machine Learning*.

[159] R. Reed and R. J. MarksII. *Neural smithing: supervised learning in feedforward artificial neural networks*. Mit Press, 1999.

[160] F. Renard, S. Guedria, N. De Palma, and N. Vuillerme. Variability and reproducibility in deep learning for medical image segmentation. *Scientific Reports*, 10(1):1–16, 2020.

[161] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[162] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[163] H. R. Roth, L. Lu, J. Liu, J. Yao, A. Seff, K. M. Cherry, L. Kim, and R. M. Summers. Improving computer-aided detection using convolutional neural networks and random view aggregation. *CoRR*, abs/1505.03046, 2015.

[164] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vision*, 115(3):211–252, Dec. 2015.

[165] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[166] R. Salakhutdinov and G. Hinton. An efficient learning procedure for deep boltzmann machines. *Neural Comput.*, 24(8):1967–2006, Aug. 2012.

[167] J. Sanders and E. Kandrot. *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

[168] W. S. Sarle. Stopped training and other remedies for overfitting. In *Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics*, pages 352–360, 1995.

[169] J. Schmidhuber. Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828, 2014.

[170] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[171] A. Sergeev and M. D. Balso. Horovod: fast and easy distributed deep learning in tensorflow. *CoRR*, abs/1802.05799, 2018.

[172] M. Shakeri, S. Tsogkas, E. Ferrante, S. Lippe, S. Kadoury, N. Paragios, and I. Kokkinos. Sub-cortical brain structure segmentation using f-cnn's. In *2016 IEEE 13th International Symposium on Biomedical Imaging (ISBI)*, pages 269–272. IEEE, 2016.

[173] N. Sharma and L. Aggarwal. Automated medical image segmentation techniques. *Journal of Medical Physics*, 35(1):3–14, 2010.

[174] D. Shen, G. Wu, and H.-I. Suk. Deep learning in medical image analysis. *Annual review of biomedical engineering*, 19:221–248, 2017.

[175] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905, Aug. 2000.

[176] H. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. J. Mollura, and R. M. Summers. Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning. *CoRR*, abs/1602.03409, 2016.

[177] G. M. Shipman, T. S. Woodall, R. L. Graham, A. B. Maccabe, and P. G. Bridges. Infiniband scalability in open mpi. In *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*, pages 10–pp. IEEE, 2006.

[178] P. E. Shrout and J. L. Fleiss. Intraclass correlations: uses in assessing rater reliability. *Psychological bulletin*, 86(2):420, 1979.

[179] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[180] Skymind. Distributed deep learning, part1 : An introduction to distributed training of neural networks. https://blog.skymind.ai/distributed-deep-learning-part-1-an-introduction-to-distributed-training-of-neural-networks/.

[181] L. N. Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472. IEEE, 2017.

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

[183] S. L. Smith, P. Kindermans, and Q. V. Le. Don't decay the learning rate, increase the batch size. *CoRR*, abs/1711.00489, 2017.

[184] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2951–2959. Curran Associates, Inc., 2012.

[185] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, Jan. 2014.

[186] M. F. Stollenga, W. Byeon, M. Liwicki, and J. Schmidhuber. Parallel multi-dimensional lstm, with application to fast biomedical volumetric image segmentation. In *Advances in neural information processing systems*, pages 2998–3006, 2015.

[187] N. Strom. Scalable distributed dnn training using commodity gpu cloud computing. In *INTERSPEECH*, 2015.

[188] A. Stupple, D. Singerman, and L. A. Celi. The reproducibility crisis in the age of digital medicine. *NPJ digital medicine*, 2(1):2, 2019.

[189] H. Su and H. Chen. Experiments on parallel training of deep neural network using model averaging. *CoRR*, abs/1507.01239, 2015.

[190] C. Szegedy, S. Ioffe, and V. Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016.

[191] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[192] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[193] A. A. Taha and A. Hanbury. Metrics for evaluating 3d medical image segmentation: analysis, selection, and tool. *BMC medical imaging*, 15(1):29, 2015.

[194] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[195] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[196] H. tien Lin and C.-J. Lin. A study on sigmoid kernels for svm and the training of non-psd kernels by smo-type methods. Technical report, 2003.

[197] C. Tobon-Gomez, A. J. Geers, J. Peters, J. Weese, K. Pinto, R. Karim, M. Ammar, A. Daoudi, J. Margeta, Z. Sandoval, et al. Benchmark for algorithms segmenting the left atrium from 3d ct and mri datasets. *IEEE transactions on medical imaging*, 34(7):1460–1473, 2015.

[198] J. K. Udupa, V. R. Leblanc, Y. Zhuge, C. Imielinska, H. Schmidt, L. M. Currie, B. E. Hirsch, and J. Woodburn. A framework for evaluating image segmentation algorithms. *Computerized medical imaging and graphics*, 30(2):75–87, 2006.

[199] A. Vedaldi and K. Lenc. Matconvnet – convolutional neural networks for matlab. In *Proceeding of the ACM Int. Conf. on Multimedia*, 2015.

[200] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1096–1103, New York, NY, USA, 2008. ACM.

[201] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, Dec. 2010.

[202] H. Wang, B. Raj, and E. P. Xing. On the origin of deep learning. *CoRR*, abs/1702.07800, 2017.

[203] Y. Wang, G. Wei, and D. Brooks. Benchmarking tpu, gpu, and CPU platforms for deep learning. *CoRR*, abs/1907.10701, 2019.

[204] Wikipedia. Comparison of deep-learning software. https://en.wikipedia.org/wiki/Comparison_of_deep-learning_software.

[205] I. Wolf, M. Vetter, I. Wegner, T. Böttger, M. Nolden, M. Schöbinger, M. Hastenteufel, T. Kunert, and H.-P. Meinzer. The medical imaging interaction toolkit. *Medical image analysis*, 9(6):594–604, 2005.

[206] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

[207] Y. You, I. Gitman, and B. Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017.

[208] Y. You, I. Gitman, and B. Ginsburg. Scaling sgd batch size to 32k for imagenet training. 2017.

[209] Y. You, Z. Zhang, C. Hsieh, and J. Demmel. 100-epoch imagenet training with alexnet in 24 minutes. *CoRR*, abs/1709.05011, 2017.

[210] S. R. Young, D. C. Rose, T. P. Karnowski, S.-H. Lim, and R. M. Patton. Optimizing deep learning hyper-parameters through an evolutionary algorithm. In *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, pages 1–5, 2015.

[211] L. Yu, S. Wang, and K. K. Lai. An integrated data preparation scheme for neural network data analysis. *IEEE Transactions on Knowledge and Data Engineering*, 18(2):217–230, 2006.

[212] Y. Yu, J. Jiang, and X. Chi. Using supercomputer to speed up neural network training. In *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*, pages 942–947, Dec 2016.

[213] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[214] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

[215] S. Zhang, A. E. Choromanska, and Y. LeCun. Deep learning with elastic averaging sgd. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and p. . C. u. . h. R. Garn pages = 685–693, year = 2015, editors, *Advances in Neural Information Processing Systems 28*.

[216] W. Zhang, S. Gupta, X. Lian, and J. Liu. Staleness-aware async-sgd for distributed deep learning. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 2350–2356. AAAI Press, 2016.

[217] W. Zhang, R. Li, H. Deng, L. Wang, W. Lin, S. Ji, and D. Shen. Deep convolutional neural networks for multi-modality isointense infant brain image segmentation. *NeuroImage*, 108:214–224, 2015.

[218] Y. J. Zhang. A survey on evaluation methods for image segmentation. *Pattern recognition*, 29(8):1335–1346, 1996.

[219] L. Zhao and K. Jia. Multiscale cnns for brain tumor segmentation and diagnosis. *Comp. Math. Methods in Medicine*, 2016:8356294:1–8356294:7, 2016.

[220] L. Zhao and K. Jia. Multiscale cnns for brain tumor segmentation and diagnosis. *Computational and mathematical methods in medicine*, 2016, 2016.

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