UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

eXascale Infolab

UNIVERSITY OF FRIBOURG

BACHELOR THESIS

Thesis Title

Author:
Author Name

Supervisor:
Prof. Dr. Philippe Cudré-Mauroux

Co-Supervisor:
Co-supervisor Name

January 01, 1970

eXascale Infolab
Department of Informatics

Boulevard de Pérolles 90 • 1700 Fribourg • Switzerland
phone +41 (26) 300 84 65 • diuf-secr@unifr.ch • www3.unifr.ch/inf

Abstract

Author Name

Thesis Title

Write the thesis abstract here. Should be between half-a-page and one page of text, no newlines.

Keywords: keywords, list, here

Contents

List of Figures

Chapter 1

Introduction

According to the World Health Organization, cancer caused 9.6 million deaths in 2018, making it the second leading cause of death [1]. An early detection and treatment increases the chances of recovery. In this context, Computer-Aided Diagnosis (CADx) systems can play a massive role by preventing health professionals from missing positive diagnoses. Thanks to the attention that deep learning has gotten over the last decade, newer and better diagnosis systems have transfered the advantages of deep learning to cancer detection, diagnosis and localization tasks.

Deep learning applications require a lot of data to perform well. While certain fields profit from massive amounts of publicly available data, the medical field is quite the opposite. First of all, medical information is protected by the doctor-patient confidentiality and cannot be shared freely. As a consequence, data first has to be collected, organized and anonymized. Then, additional information regarding the clinical significance of the samples, the location of the lesions, etc. must be provided so that deep learning models can make use of the data. This whole process is time consuming and medical institutions don't always see the benefits which could ensue from publishing datasets of good quality. As a consequence, the lack of data is one of the toughest challenge related to this field. In order to overcome it, techniques such as transfer learning exist. The latter aims at using independent but similar datasets in order to increase the performance of a model on a target dataset. In other words, models are trained on the former in order to learn relevant features which improve the results on the latter. In the case of cancer detection and classification, only few datasets are available for each body part. Therefore, the idea behind this work is to make use of datasets of different body parts to classify one specific type of cancer.

This work is divided into various parts. First, the related literature was reviewed in order to gather techniques and architectures which gave decent results.

Second, deep learning is presented from the ground up under the historical and technical points of view. This section demystifies the topic by making an overview of the mathematical concepts and definitions related to it, which makes the understanding of the technical part possible without any background in the field.

The next chapter deals with medical knowledge related to cancer. Characteristics of the disease are presented, before focusing on the various file formats which medical imaging data are exported into. The last part is critical since a lot of data processing was performed in order to be able to make use of these files.

Then, the architecture presented in a prostate cancer classification paper was reproduced. This chapter allows to set a baseline proving that the methods and implementations work, from the image preprocessing to the neural network.

Finally, the last part is dedicated to transfer learning and its application to cancer classification.

1.1 Motivation

xx

1.2 Contributions

This work proposes various methods to detect cancer on MRIs, with a focus on prostate cancer. Concretely, the major contributions of this work are:

- Processing scripts for the PROSTATEx dataset, the Lung CT Challenge dataset and the Kaggle Brain dataset. This includes the conversion of DICOM and PNG files to NumPy arrays, their registration (alignment and resizing to the same resolution for stacking purpose, ensuring the same amount of tissue on each image), their augmentation and the organization of the resulting images multiple subsets (training, validation and test) that can be used as input to machine learning algorithms.
- Visualization scripts for DICOM, NIfTI and RAW medical file formats. A system to navigate through 4D data (width, height, depth and time) using the directional arrows is implemented: left/right arrows to vary the time axis and up/down to navigate through the difference slices (i.e the depth) of all images of a patient. 3D data is also supported with the same functionalities (apart from navigating through the fourth dimension (time)).
- A state of the art deep learning model for prostate cancer detection. All steps are clearly described, the corresponding scripts are available on GitHub and the raw results are presented. This makes the experiment completely reproductible.
- Processing scripts for the PROSTATEx challenge. This goes from the preprocessing of the challenge test images to the generation of the CSV file containing the predictions of a given model, which are probabilities $[0, 1]$ of the lesion being malignant ("benign lesion" if < 0.5, "malignant tumor" otherwise).
- A transfer learning pipeline that makes use of different body parts to increase the classification performance on one chosen body part. This technique is a concrete solution to overcome the lack of available data for each body part. Furthermore, it increases the generalization ability of the neural network.
- Verification scripts to check the gradient flow of a neural network, the cropping of images, the presence of NaN value in images, etc. All these scripts can easily be imported in other projects.

Chapter 2

Literature review

Traditionally, building a computer-aided cancer diagnosis or detection (CAD) system consisted of two different parts: "Lesion detection and false-positive reduction. Lesion detection was primarily based on algorithms specific to the detection task, resulting in many candidate lesions. False-positive reduction was commonly based on traditional machine learning methods to reduce the false positive rate. However, even with these complicated and sophisticated programs, the general performance of conventional CAD systems was not good, thus hampering their widespread usage in clinical practice" [2].
Nowadays, these two steps tend to disappear thanks to deep learning algorithms, in particular the ones based on convolutional neural networks. In fact, the latter can be considered as single-step CADs. For this reason, CAD systems are currently a trending topic in deep learning.
There mainly exists two types of CAD systems based on convolutional neural networks. The first one is called CADe for "computer-aided detection". Its primary goal is "to increase the detection rate of diseases while reducing the false negative rate possibly due to the observers' mistakes or fatigue" [2]. This group includes medical image analysis tasks such as segmentation, identification, localization and detection. The other type is CADx for "computer-aided diagnosis" which aims at extracting the characteristics lesions.
This chapter presents the main research works about CADe systems based on convolutional neural networks for each body part used in sections 5 and 6, i.e. prostate, lung and brain. Furthermore, it exclusively focuses on studies that used the same datasets as the ones used in this work.

2.1 Prostate/PROSTATEx

As stated by Gao et al., "Prostate cancer is the most common malignancies among men and remains a second leading cause to deaths in men globally. It was predicted that there would be 1.7 million new cancer cases by 2030. The early detection and diagnosis of prostate cancer can help to survive nine out of 10 men for the last five years"[2]. Therefore, researchers proposed many different models to achieve good performance in prostate cancer detection. All the research works mentioned below are based on the SPIE-AAPM-NCI PROSTATEx Challenge dataset. This dataset is composed of multiparametric MRIs (T2W, DWI, ADC, DCE, PD, Ktrans) for a total of 204 training patients and 208 challenge patients (see Section 5.2.1).

Song et al. [3] presented a DCNN method to detect prostate cancer on multiparametric MRIs. Their data processing approach kept T2W, DWI and ADC grayscale images only. After resampling each image to the same resolution, T2W, DWI and ADC images were first cropped (65x65px patch) with the lesion in the center and stacked per patient, resulting in images containing three grayscale channels. Thanks

to this method, the same lesion is visible in the same area over the three channels. This increases the probability of detecting a cancer by ensuring a good visibilty for each lesion, since the latter is not necesarily as visible with each parameter. Images were then normalized based on the Z-score per patient and per sequence (T2W, DWI, ADC), i.e. by subtracting the mean before dividing by the standard deviation. The data was split into a training (80%), validation (10%) and test set (10%). The training (undefined number of times), validation (undefined number of times) and test images (11x) were augmented using -20° to 20° rotations, horizontal flipping, vertical sliding of less than 2 pixels and stretching by a factor between 0.9 and 1.1. Most of the processing techniques are reproductible, apart from the manual lesion contouring and labelling performed by a radiologist. Their model is a modified version of the well-known VGG16 model, including the addition of 1x1 convolutions and dropout layers after each max pooling layer, and the use of the ELU activation function (see figure 5.4). The evaluation method for each patient and finding made an average of the 11 predictions resulting from the 11-time augmentation of the test set. The best results were obtained by using DWI images only with the highest b-value, reaching an AUC of 0.944 with a 95% confidence interval (0.876-0.994). However, this model was not tested on the official PROSTATEx challenge images, which is an interesting benchmark to evaluate how well a model generalizes.

Liu et al. [4] created another architecture called XMasNet which was tested on the actual PROSTATEx challenge, achieving the second best performance at the time with an AUC of 0.84. The AUC on their validation set reached 0.92. Regarding data processing, their approach stacked different combinations of the available sequences as the three channels instead of defining a single combination like Song et al.: DWI-ADC-Ktrans, DWI-ADC-T2W, ADC-Ktrans-T2W and DWI-Ktrans-T2W. The data augmentation process differs in that the images are rotated in 3D, each lesion being sliced at 7 different orientations. These 2-dimensional slices were then augmented using rotation, shearing and translation of 1px, resulting in 207144 training samples. Both validation and testing test were also augmented in the same manner. This whole process allows to include 3-dimensional information in 2-dimensional images. The method used ensemble learning which combined different models to reach the best performance possible.

Mehrtash et al. [5] used a different approach. First of all, the input was fed to three separated parts of the model, each one responsible for a specific sequence among ADC, maximum b-value DWI and Ktrans. Then, each of these feature extractors' outputs were merged into a common decision maker. Furthermore, 3-dimensional convolutions instead of 2-dimensional ones were performed. In fact, 3-dimensional patches centered on the lesion were cropped. Augmentation including translation and flipping was used in order to balance the dataset. Apart from these differences, other minor differences such as normalizing the images within the range $[0, 1]$ exist compared to the previous papers. These tricks allowed their model to achieve an AUC of 0.80 on the PROSTATEx challenge. To make predictions, five different models were used, averaging the predictions of the four best models.

Armato et al [6] summarized the results obtained by all teams that took part in the PROSTATEx challenge in 2017. This challenge was split into two separate tasks. The first one was devoted to the diagnosis of prostate lesions (classification) whereas the other was about the segmentation and the determination of Gleason Grade Group. Thirty-two groups submitted results to the first challenge out of a total of 71 different methods (each group was allowed to submit up to three methods for evaluation). The article indicates that "most, but not all, methods outperformed random guessing (AUC=0.5)" [2]. The best performing method obtained an AUC

value of 0.87 (standard error 0.027) and the next three methods all achieved AUC values of 0.84 (with standard errors of 0.036, 0.032, and 0.032). The complete results are illustrated on figure 2.1. The median AUC on the challenge is approximately 0.68.

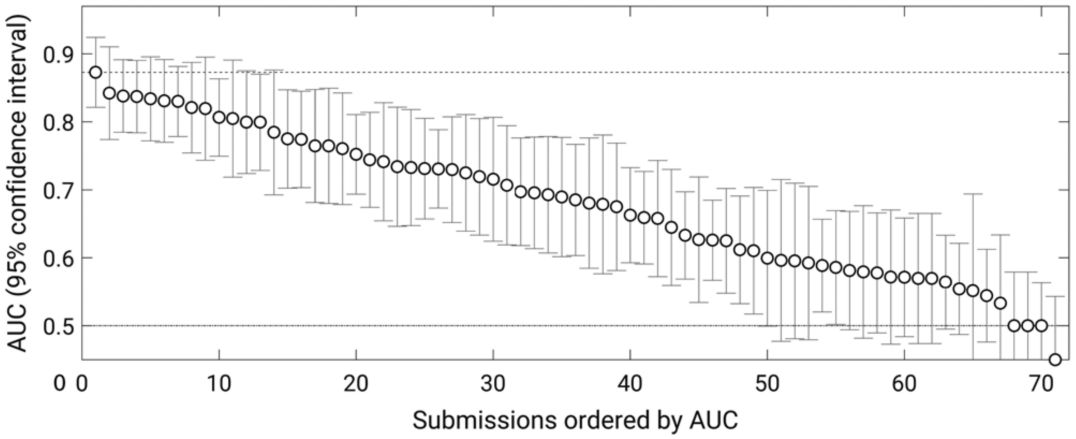

FIGURE 2.1: AUC values achieved by the 71 methods that participated in the PROSTATEx Challenge [6]

2.2 Lung/Lung CT Challenge

Gao et al. claimed that "lung cancer is one of the most frequent and leading causes of death all over the world. It was reported that there were approximately $1.8x10^6$ new cases of lung cancer globally in 2012. Early detection of lung cancer, which is typically viewed in the form of lung nodules, is an efficient way to improve the survival rate" [2]. The articles cited below make use of the SPIE-AAPM Lung CT Challenge dataset, which is composed of CT scans devoted to the classification of lung nodules (see 6.3.2 for further details).

Cengil et al. [7] built a fairly simple convolutional neural network to classify the images of the Lung CT Challenge dataset. The model takes 4-dimensional data as input (depth, height, width and channels) and performs 3D convolutions on it. The model consists of an input layer, five layers of 3D convolutions (the first is associated with a RELU activation function and pooling, the last with nothing, and the others with pooling) and a fully connected layer at the end. Regarding the model evaluation, authors announce an accuracy of 0.7 on their test set which contains 30 findings.

Armato et al. [8] described the LUNGx Challenge and the overall results. This challenge consisted in classifying the lung nodules as benign or malignant among a training set of 10 scans and a test set of 60 scans. Since the training set was extremely small, training the model on other datasets was allowed. The article describes the results of the proposed methods and compares them with the performance of six qualified radiologists on the same task. The article reports that "ten groups applied their own methods to 73 lung nodules (37 benign and 36 malignant) that were selected to achieve approximate size matching between the two cohorts. Area under the receiver operating characteristic curve (AUC) values for these methods ranged from 0.50 to 0.68. Only three methods performed statistically better than random

guessing. The radiologists' AUC values ranged from 0.70 to 0.85. Three radiologists performed statistically better than the best-performing computer method." [8]. Figure 2.2 gives an overview of the results.

Method	AUC value	SE	Nodule segmentation	Classifier	Cases to train
1	0.50	0.068	Voxel-intensity-based segmentation	SVM	LUNGx calibration
2	0.50	0.056	Region growing	WEKA	NLST
3	0.54	0.067	None required	Rules based on histogram-equalized pixel frequencies	LUNGx calibration
4	0.54	0.066	Bidirectional region growing	Uses tumor perfusion surrogate	LUNGx calibration
5	0.55	0.067	Region growing	WEKA	NLST
6	0.56	0.054	Graph-cut-based surface detection	Random forest	LIDC
7	0.59	0.066	Manual initialization, gray-level thresholding, morphological operations	SVM	LUNGx calibration
8	0.59	0.053	None required	Convolutional neural network	LIDC
9	0.61	0.054	GrowCut region growing with automated initial label points	SVM	NLST
10	0.66	0.063	Radiologist-provided nodule semantic ratings	Discriminant function	LUNGx calibration
11	0.68	0.062	Semiautomated thresholding	Support vector regressor	In-house dataset
Observer					
1	0.70	0.060			
2	0.75	0.057			
3	0.78	0.046			
4	0.82	0.049			
5	0.83	0.047			
6	0.85	0.044			

FIGURE 2.2: AUC values for the 11 computerized methods and six radiologists in the task of classifying malignant and benign nodules [8]

2.3 Brain/Kaggle Brain

Brain cancer is another major type of cancer. According to American Society of Clinical Oncology, "brain and other nervous system cancer is the 10th leading cause of death for men and women. It is estimated that 17760 adults (9910 men and 7850 women) will die from primary cancerous brain and central nervous system tumors this year" [9]. The following research papers are based on the Kaggle "Brain MRI Images for Brain Tumor Detection" dataset. This dataset, unlike the ones for the other body parts, does not come from a certified medical authority but from the Kaggle website [10]. However, since it is the only dataset available for brain tumor classification, some publications used it. Further details about this dataset can be found in section 6.3.3.

Saxena et al. [11] implemented three convolution neural networks to classify the brain tumors coming from the Kaggle "Brain MRI Images for Brain Tumor Detection" dataset. Their processing method used a cropping technique which removed extra black margin around the skull. Each border of the image merged with a part of the skull. Since the images come from different sources, their resolution vary quite

a lot. Therefore, the authors resized them to 224x224x3. Moreover, data was augmented by rotation, vertical shifting and horizontal shifting. As the cropping was performed before augmenting the images, small parts of the brain were outside of the augmented images due to rotation and shifting. The data was split into a training set, a validation set and a test set. Regarding the models, authors implemented three of them (a Resnet-50, a VGG-16 and an Inception-V3) in order to compare their performance. The best results on the test set were achieved by the Resnet-50 (AUC of 0.95 and accuracy of 0.95). The VGG-16 was close (AUC of 0.90 and accuracy of 0.90), whereas the Inception-V3 didn't perform well (AUC of 0.55 and accuracy of 0.55).

Habib Mohamed Ali [12] proposed his own convolutional neural network to classify the images of the Kaggle Brain dataset. First, the data was augmented. Then, it was cropped thanks to OpenCV so that the resulting images only contained the brain itself. Afterwards, they were resized and normalized in order to scale pixel values to the range $[0, 1]$. Once the processing part was over, the data was split into a training set (70%), a validation set (15%) and a test set (15%). Regarding the neural network structure, the model is simple. It is composed of only one convolutional layer with a batch normalization layer and ReLU activation function, followed by two max-pooling layers and a dense layer. This model achieved an accuracy of 89% on the test set.

Chapter 3

Deep learning

This chapter provides the theoretical foundation in deep learning which is required to understand the rest of the work. It starts with a historical timeline of deep learning before describing what a neural network is. Then, the notion of training a neural network, with all that is involved such as forward propagation, backpropagation, hyperparameters, data splitting or performance evaluation is explained. This part is followed by another section devoted to a special type of neural networks, the "convolutional neural networks". They are used in many computer vision applications due to their great performance on these tasks. Finally, the concept of "transfer learning" is discussed as the entire chapter 6 relies on it.

3.1 Introduction to deep learning

Deep learning is currently one of the trendiest topics in machine learning, a subset of artificial intelligence. Machine learning refers to statistical models that allow computers to perform specific tasks without having been explicitly programmed to solve them. In fact, these models try to find structural patterns within data in order to understand new incoming situations and react in the best possible way. There exist various techniques in machine learning such as k-NN, SVM, k-means, decision trees, association rules, etc. What mainly differentiates deep learning from these algorithms is the concept of neural networks (see Section 3.1.2) that are combined to form deep neural networks.
Neural networks are inspired from the biological neural networks of the brain. These systems try to learn how to solve a problem based on the data they receive as input. Many concrete applications make use of neural networks: autonomous vehicles, smarter translators, computer-aided diagnoses, personal assistants, art creation, robotics etc. The presence of deep learning techniques in these use cases clearly testifies to the enthusiasm of many areas for this technology. Furthermore, as this field has recently gained interest (see Section 3.1.1), a lot of research is still ongoing, which suggests that many exciting new applications will certainly be discovered in the near future.

3.1.1 Historical background

As described on figure 3.1, the theoretical foundations of deep learning appeared long before the invention of computers. From the first attempts to understand the human brain until today, huge progress was made to establish the basic components of modern neural networks. One could ask why deep learning took off recently if the theory was around for a long time.
As stated by Goodfellow et al. [13], the first part of the answer is computing power.

In fact, deep learning algorithms need a lot of data to work properly, which requires powerful CPUs/GPUs that either didn't exist or were only within few people's reach. One other main reason concerns the lack of data. Since deep learning algorithms "learn" from data, learning is impossible if good-quality data is not available. The era of Big Data enhanced deep learning possibilities. Finally, before the year 2012, the abilities of neural networks were still to be proven. This changed with the ImageNet Large Scale Visual Recognition Challenge (a competition where researchers evaluate their algorithms on several visual recognition tasks). In fact, the deep convolutional neural network called "AlexNet" achieved 16% of classification error rate, whereas the previous best scores were around 25%. This victory marked the beginning of a new craze for these types of algorithms.

Year	Contributer	Contribution
300 BC	Aristotle	introduced Associationism, started the history of human's attempt to understand brain.
1873	Alexander Bain	introduced Neural Groupings as the earliest models of neural network, inspired Hebbian Learning Rule.
1943	McCulloch & Pitts	introduced MCP Model, which is considered as the ancestor of Artificial Neural Model.
1949	Donald Hebb	considered as the father of neural networks, introduced Hebbian Learning Rule, which lays the foundation of modern neural network.
1958	Frank Rosenblatt	introduced the first perceptron, which highly resembles modern perceptron.
1974	Paul Werbos	introduced Backpropagation
1980	Teuvo Kohonen	introduced Self Organizing Map
	Kunihiko Fukushima	introduced Neocogitron, which inspired Convolutional Neural Network
1982	John Hopfield	introduced Hopfield Network
1985	Hilton & Sejnowski	introduced Boltzmann Machine
1986	Paul Smolensky	introduced Harmonium, which is later known as Restricted Boltzmann Machine
	Michael I. Jordan	defined and introduced Recurrent Neural Network
1990	Yann LeCun	introduced LeNet, showed the possibility of deep neural networks in practice
1997	Schuster & Paliwal	introduced Bidirectional Recurrent Neural Network
	Hochreiter & Schmidhuber	introduced LSTM, solved the problem of vanishing gradient in recurrent neural networks
2006	Geoffrey Hinton	introduced Deep Belief Networks, also introduced layer-wise pretraining technique, opened current deep learning era.
2009	Salakhutdinov & Hinton	introduced Deep Boltzmann Machines
2012	Geoffrey Hinton	introduced Dropout, an efficient way of training neural networks

FIGURE 3.1: Deep learning milestones - Wang et al. [14]

3.1.2 What is a neural network?

From a descriptive point of view, neural networks can simply be seen as non-linear applications that associate an input to an output with respect to certain parameters. The input can be an image, a sound or any input that can be converted into numerical features. The output of a neural network depends on the problem it tries to solve. In computer vision, the most common types of outputs are classes (for classification

problems) and pixel coordinates (for segmentation problems).
From a mathematical standpoint, a neural network can be defined as a non-linear function f that associates an input x to an output y with respect to parameters θ.

$$y = f(x, \theta) \tag{3.1}$$

The parameters θ are estimated from the training samples.

3.1.3 Supervised and unsupervised learning

Machine learning algorithms belongs can be classified in two classes. The first one is "supervised learning". It includes learning algorithms whose training samples are associated with their labels in order to find the optimal mapping between the input and the output. The second one is "unsupervised learning". In contrast to supervised algorithms, the latter rely on unlabeled data. Its main goal is to infer the natural structure present in the data. Since the models presented in this work belong to the "supervised learning" category, notions explained below refer to this kind of algorithms.

3.2 Neural networks basics

3.2.1 Notation

In order to keep the mathematical description of neural network consistent, this work will use Andrew Y. Ng's notation [15], who was a pioneer in deep learning.

General comment

- Superscript (i) denotes the i^{th} training example.
- Superscript $[l]$ denotes the l^{th} layer of the neural network.

Sizes

- m: number of examples in the dataset
- n_x: input size
- n_y: output size (or number of classes)
- $n_h^{[l]}$: number of hidden units (i.e neurons) of the l^{th} layer
- L: number of layers in the network

Neural networks components

- $X \in \mathbb{R}$ is the input matrix of a neural network.
- $x^{(i)} \in \mathbb{R}^{n_x}$ is the i^{th} example (sample) represented as a column vector.
- $Y \in \mathbb{R}^{n_y \times m}$ is the label matrix.
- $y^{(i) \in \mathbb{R}^{n_y}}$ is the output label for the i^{th} example.
- $W^{[l]} \in \mathbb{R}^{\text{\# of neurons in the next layer = j x \# of neurons in the previous layer = k}}$ is the weight matrix at layer $[l]$.

- $b^{[l]} \in R^{\#\text{ of units in next layer}}$ is the bias vector in the l^{th} layer.
- $\hat{y} \in R^{n_y}$ is the predicted output vector. It can also be denoted as $a^{[L]}$ where L is the number of layers in the whole network.
- $g^{[l]}(x)$ is the l^{th} activation function.
- $z^{[l]} = W_x x^{(i)} + b^{[l]}$ denotes the weighted sum of the input given to the l^{th} layer before passing through the activation function.

Forward propagation equations

- $a = g^{[l]}(W_x x^{(i)} + b^{[l]}) = g^{[l]}(z^{[l]})$ where $g^{[l]}$ denotes the l^{th} layer activation function.
- $a_j^{[l]} = g^{[l]}(\sum_k w_{jk}^{[l]} a_k^{[l-1]} + b_j^{[l]}) = g^{[l]}(z_j^{[l]})$ is the general activation formula at l^{th} layer.
- $J(x, W, b, y)$ and $J(\hat{y}, y)$ denote the cost function.

3.2.2 Perceptrons

Perceptrons are the main components of neural networks. They were "developed in the 1950s and 1960s by the scientist Frank Rosenblatt, inspired by earlier work by Warren McCulloch and Walter Pitts" [16]. Today, they are called "artificial neurons" or simply "neurons".
The output a of a perceptron j is a function f of input $x = (x_1, ..., x_n)$ weighted by a vector of weights $w = (w_1, ..., w_n)$, completed by a bias b_j and associated to a non-linear activation function g:

$$a_j = f_j(x) = g((\sum_{k=1}^{n} x_k * w_k) + b_j) \tag{3.2}$$

Schematically speaking, a perceptron can be represented as on figure 3.2. Each input is multiplied with its corresponding weight. The sum of this result then goes through a non-linear function, called "activation function". This activation function acts like a threshold that determines the proportion of the result that goes further in the network. There exist multiple activation functions (see 3.2.3). It is extremely important to use non-linear functions instead of a linear functions. In fact, the output of a perceptron is given as input to the others (see 3.2.4). Consequently, if only linear functions are used throughout the network, linear outputs are given as inputs to other linear functions. Since the composition of two linear functions is itself a linear function, assembling perceptrons to create neural networks of multiple layers would not make sense anymore.

3.2.3 Activation functions

Once the computation of the weighted sum of all inputs for a specific neuron is done, the latter has to pass the sum through an activation function that will decide the proportion of the result that will be sent to the next layer. Activation functions must be non-linear in order to approximate extremely complex functions. In fact, neural networks are considered as universal approximators. Hornik et al. claim that "multi-layer feedforward networks are capable of approximating any measurable function

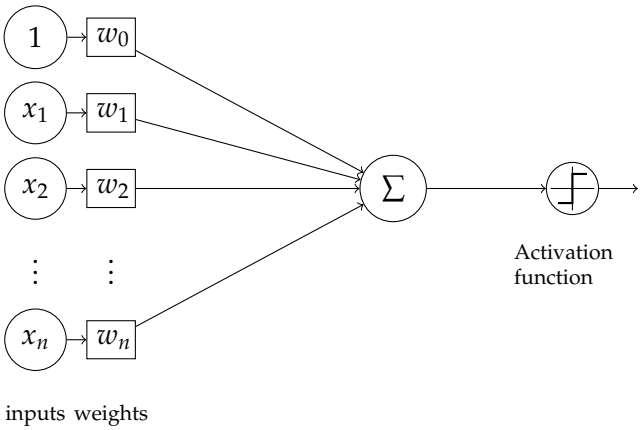

FIGURE 3.2: The perceptron model

to any desired degree of accuracy, in a very specific and satisfying sense" [17]. According to Thomas Epelbaum [18], the most commonly used activation functions are:

Sigmoid function
The sigmoid function is defined as:

$$g(x) = \frac{1}{1 + e^{-x}} \tag{3.3}$$

Its derivative is:

$$g'(x) = g(x)(1 - g(x)) \tag{3.4}$$

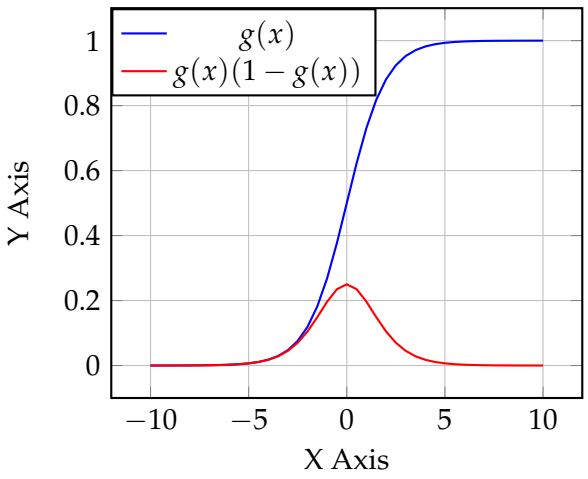

FIGURE 3.3: The sigmoid function and its derivative

Tanh function

$$g(x) = tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{3.5}$$

Its derivative is:

$$g'(x) = tanh'(x) = 1 - tanh^2(x) \tag{3.6}$$

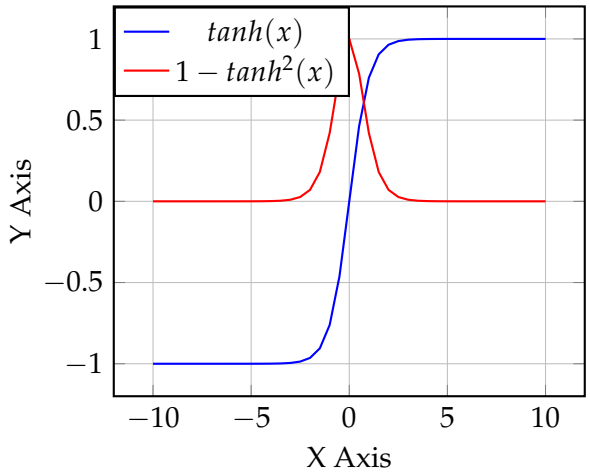

FIGURE 3.4: The tanh function and its derivative

ReLU function

$$g(x) = ReLU(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

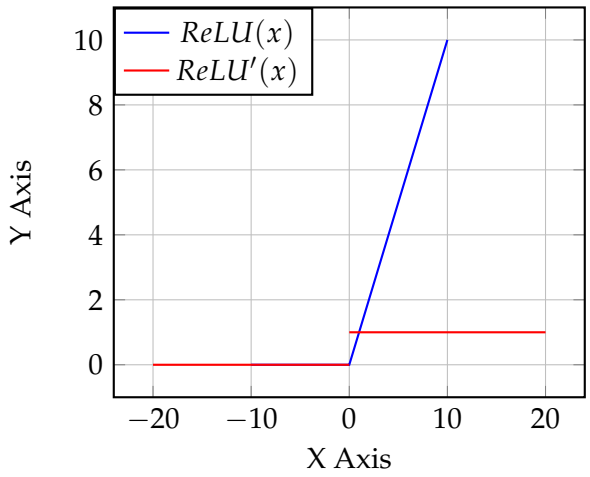

FIGURE 3.5: The ReLU function and its derivative (undefined when $x = 0$)

ELU function

$$g(x) = \begin{cases} x & \text{if } x \geq 0 \\ e^x - 1 & \text{otherwise} \end{cases} \tag{3.8}$$

Its derivative is:

$$g'(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ e^x & \text{otherwise} \end{cases} \tag{3.9}$$

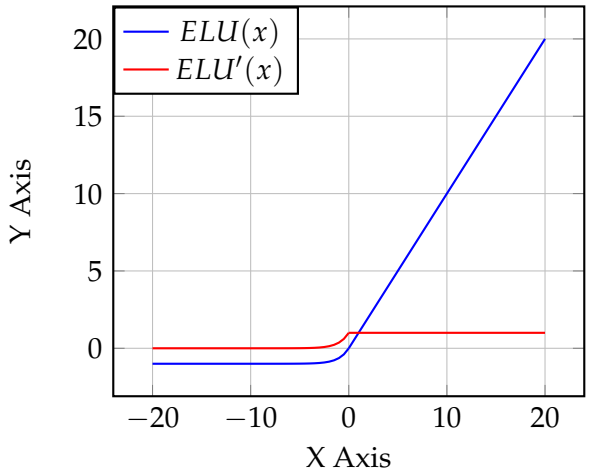

FIGURE 3.6: The ELU function and its derivative

3.2.4 Multilayer perceptrons

A multilayer perceptron is a type of artifical neural networks. Du et al. [19] define multilayer perceptrons as "feedforward networks with one or more layers of units between the input and output layers" where "the output units represent a hyperplane in the space of the input patterns."
A multilayer perceptron is composed of L layers, each of them composed of n_h^l perceptrons. The layers are organized in the following way:

- The input layer: It is the neural network entry point for the data. Generally speaking, the data is provided in the form of a matrix $X \in \mathbb{R}$ of size $(n_x \times batch\_size)$ with their corresponding labels $Y \in \mathbb{R}$ of size $(n_y \times batch\_size)$. The batch size defines the number of samples that will be given at the same time to the network and n_x is the dimension of each sample. Moreover, $x^{(i)}$ is the i^{th} sample represented as a column vector. The total number of training samples is given by m. Finally, $y^{(i)}$ is the output label for the i^{th} example. For instance, suppose the number of samples is 100 and the batch size 32. In this situation, the network will be fed with 4 batches of sizes [32, 32, 32, 4] respectively.

- The hidden layer(s): Hidden layers stand for all layers that are between the input layer and the output layer. Each of them has its own weights and biases (W, b), denoted by $W^{[l]} \in \mathbb{R}$ and $b^{[l]}$ respectively, where W_{ij}^l corresponds to the weights associated with the connection between perceptron j in layer l and perceptron i in layer $l+1$. By analogy, $b_i^{[l]}$ is the bias associated with perceptron i in layer l. Weights and biases are the parameters to optimize in order to obtain the best mapping between the input and the output of the network (see Section 3.3). Before training the neural network, the weights can be randomly initialized or initialized with more sophisticated methods such as "Xavier initialization" or "Kaiming initialization"(see Section 3.3.6).

- The output layer: It is the last layer of the neural network. Its role is essential since it produces the prediction of the network for a given input. The prediction of a neural network is given by $\hat{y} \in R^{n_y}$ with n_y representing the number of different labels. In a classification task, whose goal is to assign to each input

a specific class, the $\hat{y}$ is usually the probability that the input belongs to each class. In that case, the *softmax* activation function would be used at the end.

The advantage of organizing the weights, biases and inputs in matrices is due to the ability of modern CPUs/GPUs to quickly perform linear algebra computations. This way of structuring the network component is called "vectorization" which avoids using loops in the code, which would considerably slow down the computations. Figure 3.7 illustrates the concept of multilayer perceptrons. In this example, the total number of layers L is equal to 3, the input size n_x is equal to 4 and the number of units in in each layer is $n_h^1 = 4$, $n_h^2 = 2$, $n_h^3 = 1$. The network contains the weights $W^{(1)} \in \mathbb{R}^{2x3}$, $W^{(2)} \in \mathbb{R}^{1x2}$ and the biases $b^{[1]} = 2$, $b^{[2]} = 1$.

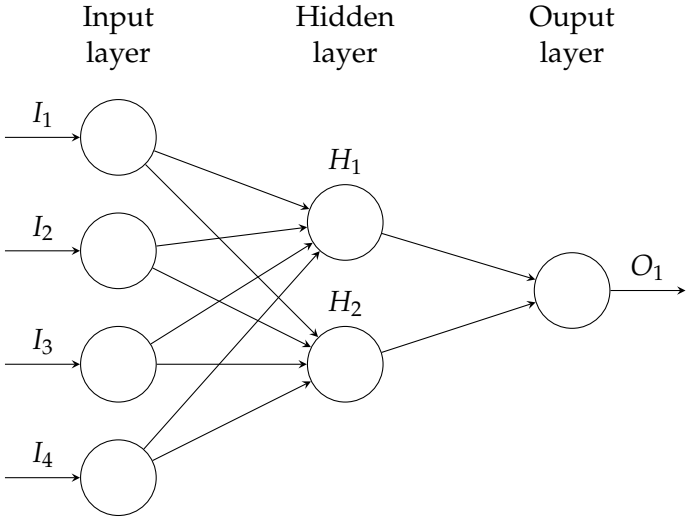

FIGURE 3.7: Multilayer perceptrons

3.3 Training a neural network

Training a neural network can be broken down into multiple steps. The first one is the "forward propagation" step. It consists in giving examples that need to be classified (or segmented, depending on the task) to the untrained neural network and to spread intermediate results through all layers of the network. After seeing every single batch, the loss is computed using a "loss function". The latter is used to evaluate the predictions of the neural network in comparison to their ground-truth. Then, the weights and biases of the networks are updated during a process called "backpropagation" in order find the global minimum of the loss function. As soon as the neural network has seen every single batch, the end of an "epoch" is reached. This process is repeated for a defined number of epochs.

3.3.1 Forward propagation

The forward propagation is used to transmit the input through the entire neural network. Mathematically speaking, the forward propagation step for a specific layer l is represented by two equations. The first equation denotes the weighted sum of the input given to the l^{th} layer before passing through the activation function g:

$$z^{[l]} = W_x^{[l]} x^{(i)} + b^{[l]} \tag{3.10}$$

The second equation describes the effect of the activation function:

$$a^{[l]} = g^{[l]}(z^{[l]}) \tag{3.11}$$

Since the output of the activation function is then given as input to all the neurons of the next layer, the whole forward propagation step can be defined as:

$$a_j^{[l]} = g^{[l]}(\sum_k w_{jk}^{[l]} a_k^{[l-1]} + b_j^{[l]}) = g^{[l]}(z_j^{[l]}) \tag{3.12}$$

Figure 3.8 illustrates the computation of the forward propagation pass for the l^{th} layer. The weight natrix W_{jk}^{l} represents the weights associated with the connection between perceptron k in layer l and perceptron j in layer $l+1$. This matrix is multiplied by the output of the previous layer $a_j^{[l-1]}$ before adding the bias $b^{[l]}$. The result is given as input to the activation function.

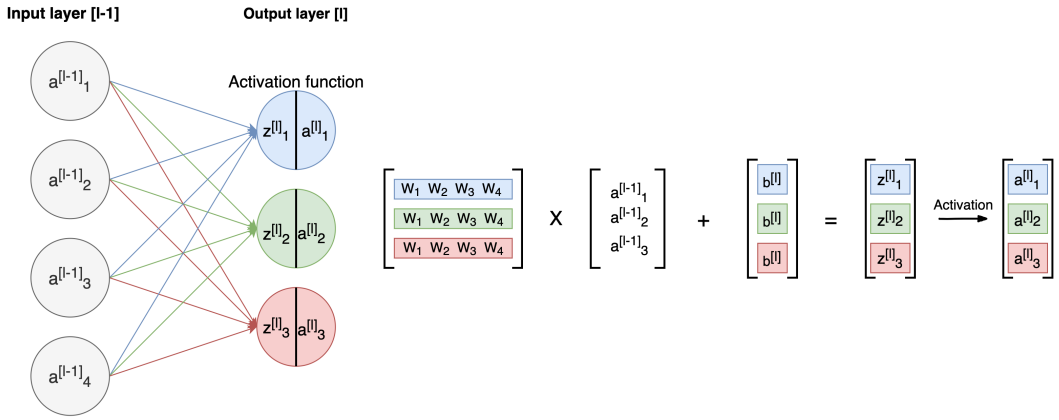

FIGURE 3.8: Forward propagation

3.3.2 Loss computation

As stated by Thomas Epelbaum, "the loss function evaluates the error performed by the neural network when it tries to estimate the data to be predicted" [18]. It is therefore useful to measure the penalty for a single input. On the contrary, when the goal is to get a more general overview of the error on the entire batch or on the entire dataset, the cost function J is used. The latter is represented by $J(\hat{y}, y)$ where $\hat{y}$ is the prediction of the neural network and y the real label.

There exist multiple cost functions. For a regression problem, a commonly used loss function is the mean square error:

$$J(\hat{y}, y) = \frac{1}{m}[\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2] \tag{3.13}$$

For classification problems, the cross entropy function is regularly used. We distinguish the binary classification where the number of classes $n_y = 2$ from the multiclass classification where $n_y > 2$. In the case of binary classification, the cross entropy is:

$$J(\hat{y}, y) = -\frac{1}{m}\sum_{i=1}^{m}[y_i * log(\hat{y}_i) + (1 - y_i) * log(1 - \hat{y}_i)] \tag{3.14}$$

In the case of multiclass classification, the categorical crossentropy is:

$$J(\hat{y}, y) = -\sum_{i=1}^{n_y}\sum_{j=1}^{m}(y_{ij} * log(\hat{y}_{ij})) \tag{3.15}$$

Since the cost function gives an estimation of the overall error of the network, the main objective of training a neural network is to update its weights in order to approach the minimum of the function. Therefore, deep learning problems can be considered as optimization problems. Solutions to these problems can be found using the gradient descent algorithm during backpropagation.

3.3.3 Backpropagation

Backpropagation relies on a technique called "gradient descent" to minimize the cost function $J(W, b)$. Generally speaking, "the intuition behind the backpropagation algorithm is as follows. Given a training example $(x^{(i)}, y^{(i)})$, we will first run a forward pass to compute all the activations throughout the network, including the output value of the network. Then, for each node i in layer l, we would like to compute an "error term" $\partial_i^{(l)}$ that measures how much that node was "responsible" for any errors in our output. For an output node, we can directly measure the difference between the network's activation and the true target value, and use that to define $\partial_i^{(n_l)}$(where layer n_l is the output layer). How about hidden units? For those, we will compute $\delta_i^{(l)}$ based on a weighted average of the error terms of the nodes that uses $a_i^{(l)}$ as an input" [20].

In other words, after each forward pass through the entire network, backpropagation performs a backward pass which aims at minimizing the cost function by adjusting parameters of the model. The way parameters are updated is defined by the gradients of the cost function with respect to each parameter of the network. The gradient of the cost function $J(x_1, x_2, ..., x_m)$ at point x is given by:

$$\frac{\partial J}{\partial x} = [\frac{\partial J}{\partial x_1}, \frac{\partial J}{\partial x_2}, ..., \frac{\partial J}{\partial x_1}] \tag{3.16}$$

The gradient shows how much all parameters that constitute x need to change to minimize the function. In neural networks, the parameters of the cost function are all weight matrices $W^{[l]}$ and biases $b^{[l]}$. The computation of all these gradients relies on the "chain rule". In the case of weights, the chain rule is:

$$\frac{\partial J}{\partial w_{jk}^l} = \frac{\partial J}{\partial z_j^l} * \frac{\partial z_j^l}{\partial w_{jk}^l} \tag{3.17}$$

Similarly, the chain rules has to be applied to the biases:

$$\frac{\partial J}{\partial b_j^l} = \frac{\partial J}{\partial z_j^l} * \frac{\partial z_j^l}{\partial b_j^l} \tag{3.18}$$

Once the gradients of each parameter are computed, these parameters are updated. The weights update is described by the following equation:

$$W^{[l]} = W^{[l]} - \alpha * \frac{\partial J}{\partial W^{[l]}} \tag{3.19}$$

The biases update corresponds to:

$$b^{[l]} = b^{[l]} - \alpha * \frac{\partial J}{\partial b^{[l]}} \tag{3.20}$$

The "learning rate" α determines the influence that the gradient has at each epoch. It is an hyperparameter and has to be tuned manually.

3.3.4 Metrics

In classification tasks, four separate output labels can occur:

- True Positive (TP): an output belongs to this class if the prediction that the latter **contains** a certain feature is **correct**.
- True Negative (TN): an output belongs to this class if the prediction that the latter does **not contain** a certain feature is **correct**.
- False Positive (FP): an output belongs to this class if the prediction that the latter **contains** a certain feature is **incorrect**.
- False Negative (FN): an output belongs to this class if the prediction that the latter does **not contain** a certain feature is **incorrect**.

From these four categories, multiple metrics with their own specificities can be computed [21]:

- Accuracy: ratio of the correctly labeled subjects to the whole pool of subjects.

  $$Accuracy = \frac{(TP + TN)}{TP + FP + FN + TN} \tag{3.21}$$

  Accuracy is a great measure in the case of symmetric data (i.e the number of FN $\approx$ FP and their cost is similar). When this condition is not fulfilled, accuracy can lead to bad models. For instance, let's define a binary classification model that always outputs class 0. If the data is composed of 99 samples from class 0 and 1 sample from class 1, the accuracy is equal to 99% but the model is not smart. Consequently, this metric has to be used in addition to other metrics.

- Precision: ratio of the correctly positive labeled subjects by the model to all positive labeled subjects.

  $$Precision = \frac{TP}{(TP + FP)} \tag{3.22}$$

  This metric is recommended when the confidence of the true positives predicted by the model is important. For instance, this happens in spam blockers where it is preferable to have a spam in mailbox rather than a regular mail in the spam box.

- Recall (sensitivity):

  $$Recall = \frac{TP}{(TP + FN)} \tag{3.23}$$

  This metric is recommended when the occurence of false negatives is intolerable and false positives are preferred. This makes perfect sense for disease detection models: labeling an healthy person as unhealthy is better than labeling an unhealthy person as healthy.

- F1-score:
$$F1 - score = \frac{2 * recall * precision}{(recall + precision)} \tag{3.24}$$
F1-score considers both precision and recall and is the highest if these two metrics are balanced. This metric is perfectly suitable when the cost of false positives and false negatives is not the same.

- Specificity:
$$Specificity = \frac{TN}{(TN + FP)} \tag{3.25}$$
This metric is recommended when the occurence of false positives is intolerable whereas true negatives are desired. For instance drug tests can not indicate false positives but they have to cover all true negatives.

- ROC curve and AUC: As explained by Sarang Narkhede, "The ROC curve is plotted with recall against the false positive rate (1-specificity) where recall is on y-axis and the false positive rate is on the x-axis. AUC - ROC curve is a performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability. It tells how much model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s" [22].

3.3.5 Data

In deep learning, data is essential. As seen previously, neural networks learn features from it. Therefore, data has to be handled carefully and in the right way. Usually, it is split into three different sets:

- The training set: This is the part of the dataset that is used to train the neural network (the weights and biases).

- The validation set: This dataset is used to evaluate a trained model. Usually, the evaluation on the validation set is performed every N epochs, where N is a fixed number. The validation set needs to come from the same distribution as the training set but should contain exclusively unseen data. This last point is crucial since the validation set shows how well the neural network generalizes on unknown data. The validation set can also be used as indicator to decide when the training should be stopped in order to prevent "overfitting" which is the behaviour of a model that fits to the training set too closely and don't generalize well. In fact, if the validation loss continuously increases for a certain number of epochs, going on with the training will increase the overfit. This fact of interrupting the training earlier is called "early stopping".

- The test set: This last part of the dataset is used to establish the final evaluation of the model. It also contains unseen data exclusively.

Regarding the way these three sets are split, it mostly depends on the number of samples available. If the latter is big, the data is split into training and testing using the ratio 80/20. Then the remaining training samples are also split into training and validation using the ratio 80/20. On the contrary, if there are few data available, k-fold cross-validation is a good practice. This technique consists in splitting the entire dataset into k folds. One fold is picked as test set and the others are considered as

training sets. The model is trained on training folds and tested on the test set. Then, another test set is picked and the same process is repeated until all possible tests set are picked. At the end of the process, the results of all test sets are averaged, which provides a good estimation of the model's performance. This technique is summarized on figure 3.9.

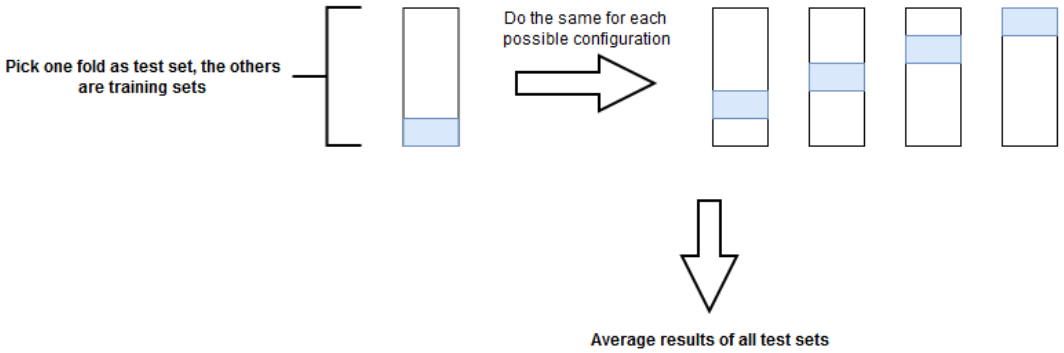

FIGURE 3.9: 5-folds cross-validation

3.3.6 Weight initialization

Before training a neural network, the weights have to be initialized in order to proceed to the first forward propagation. The initialization of the neural network weights is crucial since it will determine how quickly the network converges to an optimum. The idea behind weight initialization is to generate an initialization that "prevents layer activation outputs from exploding or vanishing during the course of a forward pass through a deep neural network. If either occurs, loss gradients will either be too large or too small to flow backwards beneficially, and the network will take longer to converge, if it is even able to do so at all" [23].
The simplest and least efficient technique to initialize neural network weights is to randomly generate them. The major problem of this technique comes from the fact that some initializations can lead to extremely small or big values, which lead to value near 0 or 1 for most activation functions. Consequently, the slope of the gradient changes slowly and the learning takes a lot of time.
To prevent this effect when the tanh activation function is used, "Xavier initialization" multiplies the random initialization by the fraction:

$$\frac{\sqrt{6}}{\sqrt{n_h^{[l]} + n_h^{[l+1]}}} \tag{3.26}$$

where $n_h^{[l]}$ is the number of incoming network connections to the layer and $n_h^{[l+1]}$ is the number of outgoing network connections from that layer.
For activation functions that are not symmetric around zero and don't have outputs inside [-1,1] such as ReLU or ELU, Kaiming initialization is an alternative. It consists in multiplying the randomly initialized weight matrix by:

$$\frac{\sqrt{2}}{\sqrt{n_h^{[l]}}} \tag{3.27}$$

where $n_h^{[l]}$ is the number of incoming connections coming into a given layer from the previous layer's output.

3.3.7 Hyperparameters tuning

Hyperparameters denote parameters that cannot be directly learned from the data. That is the case for the learning rate, the batch size and the number of epochs that were described in previous sections. So, these parameters have to be manually tuned in order to find the best configuration (i.e the one that minimizes the cost function and that keeps an acceptable level of generalization).
Regarding the learning rate α, its value has to be neither too large nor too small. A too large learning rate is recognizable by analyzing the training loss curve: if the loss is exploding or oscillating or if there is no improvement and it is stuck around a sub-optimal local optima, the learning rate is too high and should be decreased. On the contrary, if the learning is very slow and the loss does not decrease or if the model is overfitting, it is a clear sign that it should be increased. The learning rate can take a wide range of values. Consequently, the most used technique to find the optimal learning rate is simply the "trial and error", which consists in "trying widely different learning rates to determine the range of learning rates that need to be explored" [24]. There also exists methods that, instead of keeping a fixed learning rate for the entire training, reduce it after each epoch (learning rate decay) or each time the loss on the validation set does not decrease (learning rate scheduling).
Batch size is another important hyperparameter to tune. Training a network with a small batch size "requires less memory, since the latter is trained using fewer samples" [25]. Furthermore, "networks train faster because the weights update is done after each propagation" [25]. Nevertheless, "the smaller the batch the less accurate the estimate of the gradient will be" [25]. Indeed, due to the high weights update frequency, the gradient fluctuates much more than if was computed after a bigger number of samples.
Finally, the number of epochs during which the network is trained has to be carefully chosen. In fact, from a certain point in the training, neural networks don't learn anymore useful features in the data and start overfitting. This point corresponds to the moment where, the validation loss does not decrease anymore and starts to increase continuously. It is exactly at this moment that the training should stop. To achieve this goal, it is possible either to directly choose the right number of epochs or to use the so-called "early stopping" method, which stops the training as soon as the validation loss does not decrease for N epochs.

3.4 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a specific type of deep neural networks. They are particular in that they contain layers which perform a mathematical operation named convolution on the input data. CNNs are mostly used in image and video analysis.

To perform a convolution, numerical input data and a "filter" are required. A filter can be seen as a fxf numerical patch that moves across the entire input. It first moves horizontally until reaching the right-most border of the image. Then, it goes down a cell and starts from the border on the left-hand side. This process is repeated until the filter reaches the bottom-right corner of the image, which marks the end of the convolution. At each step, the dot product between the filter and the part of the

input covered by the filter is computed. Figure 3.10 showcases a simple convolution which aims at finding vertical lines in a black and white image. For example, the output of the first step of the convolution (in red) is computed by evaluating the dot product between the blue filter and the red part of the input image:

$$\begin{gathered}(-1)*0+2*0+(-1)*0\\+\\(-1)*0+2*0+(-1)*1\\+\\(-1)*0+2*0+(-1)*1\\=-2\end{gathered} \tag{3.28}$$

The output of the entire convolution shows negative values in the outside parts and large positive values in the center. This means that the 3x3 filter detected a vertical line in the center of the input image.

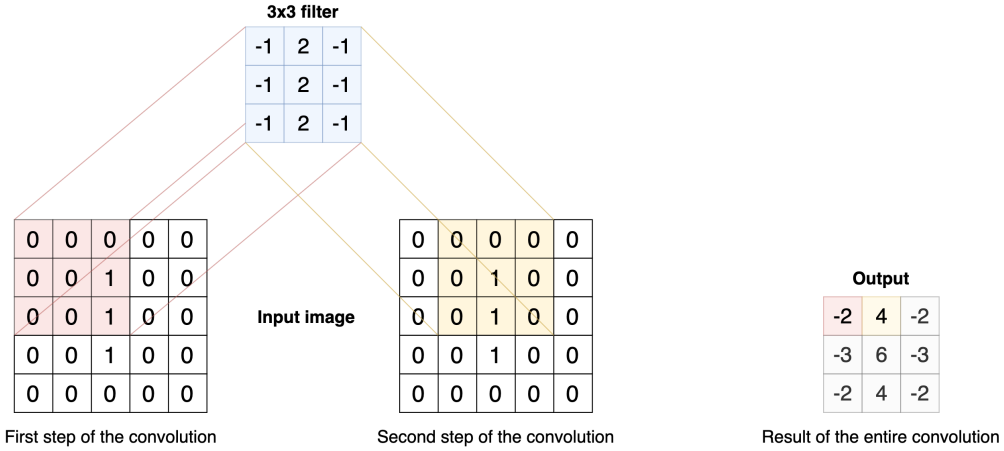

FIGURE 3.10: Convolutions - Basic convolution in a CNN

Different parameters can change the way a convolution behaves. First of all, the previous example relied on a filter moving by respectively one cell to the right and to the bottom. In this case, the so-called "stride" is equal to 1. Other applications could rely on a bigger stride. Furthermore, the previous example reduced the output size of 3x3 in proportion to the initial input size of 5x5. To influence the output size, a padding can be added to the outside of the input image, usually filled with 0s. Three ways of padding images are commonly used as shown on figure 3.11:

- **Valid:** The input image is not padded. This means that the filter only goes through existing pixel values, which makes the output size smaller than the input size.
- **Same:** The input image is padded in a way that makes the output size the same as the input size.
- **Full:** The input image is padded so that, in the first step of the convolution, only the bottom-right cell of the filter overlays the pixel value of the image, the rest overlaying padding cells. This makes the output size larger than the input size.

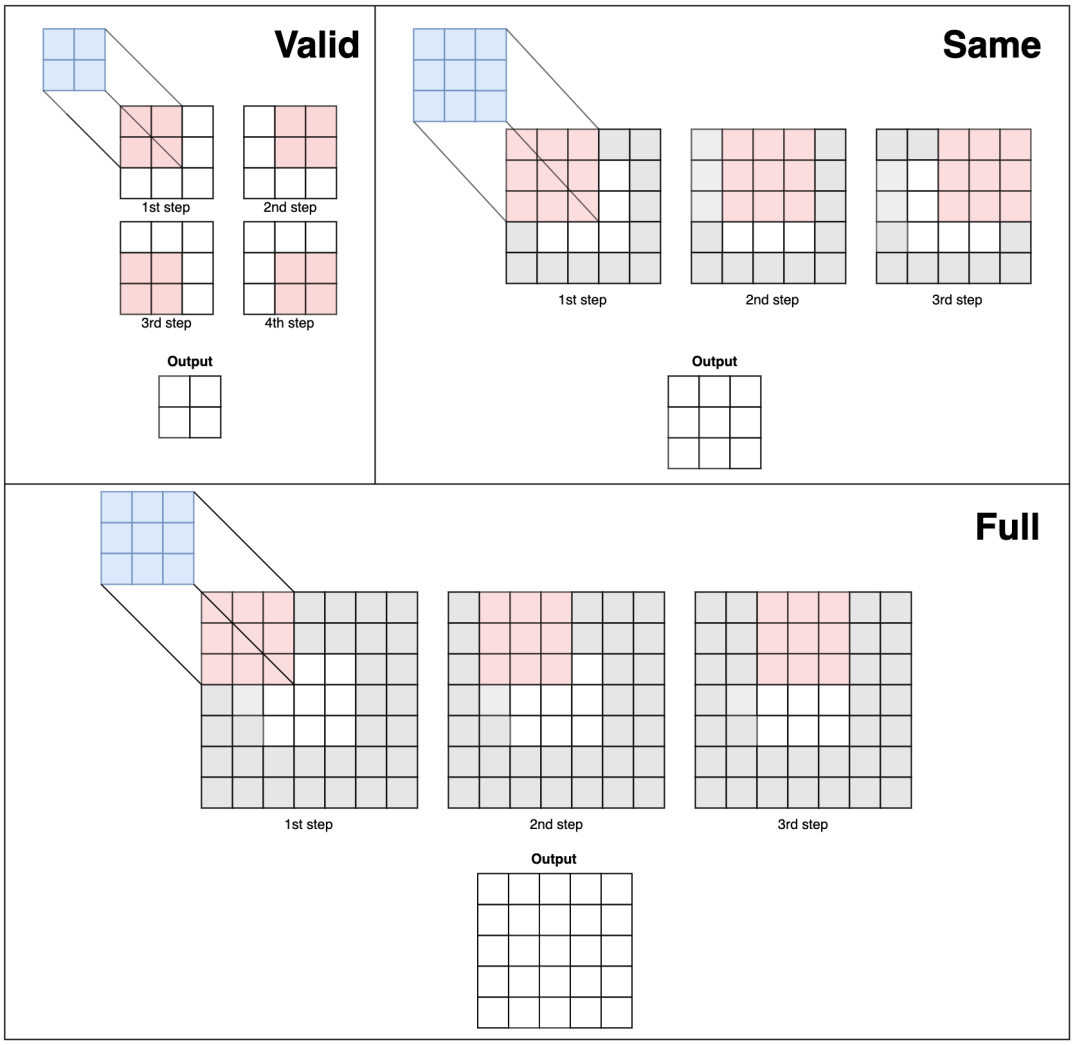

FIGURE 3.11: Convolutions - Different padding methods

3.5 Transfer learning

According to Jason Brownlee, "transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task" [26]. The model dedicated to the second task uses some or all parts of the first model (i.e. keeps the same weights and architecture or a part of them) and is then retrained on data available for this task. The first model can either be implemented from scratch if enough data is available or it can simply be downloaded from institutions that release large pretrained models for similar tasks.
There are three major benefits [26]:

- Higher start: the initial skill (before refining the model) on the source model is higher than it otherwise would be.
- Higher slope: the rate of skill improvement during training of the source model is steeper than it otherwise would be.
- Higher asymptote: the converged skill of the trained model is better than it otherwise would be.

Nevertheless, "in general, it is not obvious that there will be a benefit to using transfer learning in the domain until after the model has been developed and evaluated" [26].

Chapter 4

Medical information

This chapter gives an overview of the basic medical knowledge that is required to apprehend the following chapters smoothly. First of all, some notions about cancer are described in order to understand what it is and which effects it has on the human body. Second, medical data has its own file formats. To make use of them in a deep learning project, medical files must be processed in a certain way, depending on each format. In fact, formats represent 2D, 3D or even 4D data. Some of them require specific normalization in order to get the right rendering. Finally, some visualization tools were developed to display raw medical files easily.

4.1 Cancer

4.1.1 Basics

An accumulation of cells forming a mass is called a tumor. These tumors are detectable thanks to medical imaging (see section 4.2.1) and other symptoms. However, not every tumor is as dangerous as the other, as it can be benign (does not contain cancerous cells) or malignant (contains cancerous cells).

The term cancer refers to different phenomena which involve mutation, abnormal multiplication and spreading of cells. As stated by Hanahan et al. in "The Hallmarks of Cancer" [27] and "The Hallmarks of Cancer: The Next Generation" [28], every malignant tumor acquires six different capabilities during its evolution:

- **"Sustaining proliferative signaling"**
  Cancerous cells don't wait for the body's approval to grow and proliferate, contrary to normal cells. They become responsible for their own multiplication.
- **"Evading growth suppressors"**
  The body sends signals to contain cell growth within a tissue. Cancerous cells are insensitive to these.
- **"Activating invasion and metastasis"**
  Metastases are cells whose role is to propagate to other parts of the body in order to colonize and create new tumors.
- **"Enabling replicative immortality"**
  Healthy cells replication is limited to a certain amount, which is not the case for cancerous cells.
- **"Inducing angiogenesis"**
  Angiogenesis is the process of creating new blood vessels. Tumors have an influence on angiogenesis around them, since they need vascularization to continue growing.

- **"Resisting cell death"**
  Apoptosis is the programmed death of cells, which is part of the continuous regeneration of every cell within a body. Cancerous cells survive this programmed death.

4.1.2 Seriousness

Most cancers can be staged thanks to the TNM system. The T corresponds to the tumor size and its location; the N corresponds to whether or not the tumor has spread to draining lymph nodes; the M corresponds to the presence or absence of metastases in other parts of the body [29]. These pieces of information are used to classify cancer between four (I to IV) or sometimes five (0 to IV) different stages, reflecting the progression and the seriousness of the illness [30]. The earlier they are detected, the higher the chances of recovery are. This aspect makes cancer detection critical since every misjudgment can threaten someone's life.

4.2 Medical imaging file formats

4.2.1 Types of medical imaging

Multiple types of medical imaging exist. The most commonly used to detect cancer are Magnetic Resonance Imaging (MRI), CT (Computed Tomography) scans and mammographies.

MRI relies on magnetic fields to provide a three-dimensional view of body parts, which allows to see the generated images as a volume. Different settings, usually called sequences, make the look of the output vary, as shown on figure 4.1. Unlike MRI, CT is based on X-rays instead of magnetic fields, but still provides a three-dimensional representation of a body part. Figure 4.2 shows a lung CT scan.

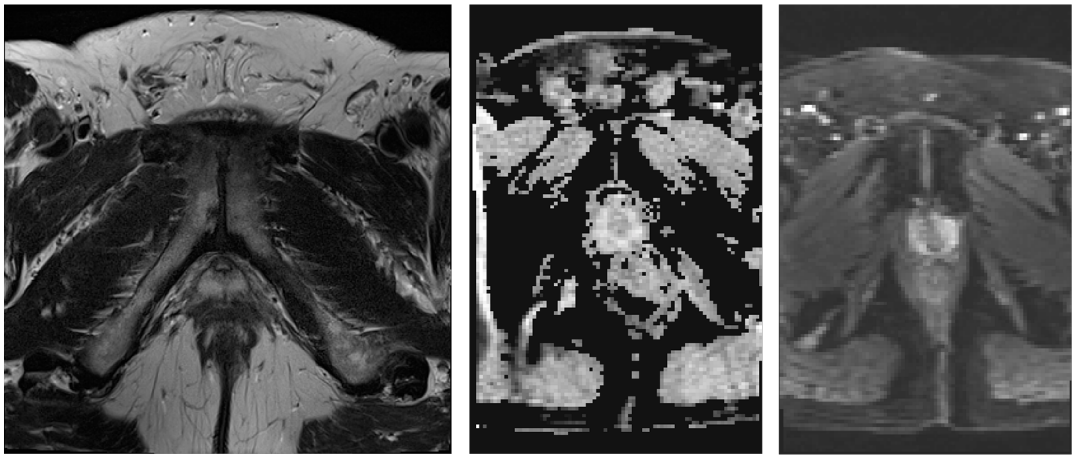

FIGURE 4.1: MRI - PROSTATEx - From left to right: T2-weighted, ADC and DWI

4.2.2 DICOM file format

Origin

The acronym DICOM stands for Digital Imaging and Communications in Medicine. Before the 1980's, images resulting from CT scans and MRIs were only decodable by machine manufacturers, while the medical community needed to export and share them for other tasks. For that reason, the ACR (American College of Radiology) and the NEMA (National Electrical Manufacturers Association) created a committee to build a standard. After two iterations with other names, DICOM was created in 1993. It standardized the representation of medical images and their transmission since it provided a network protocol built on top of TCP/IP.

Data format

DICOM files can be viewed as containers of attributes, also called tags. The values of the pixels themselves are stored under the "Pixel Data" tag. Every single DICOM file usually represents a 2-dimensional image, which will form a 3-dimensional volume when put all together.

Other useful information such as the patient name and ID is directly stored within the DICOM files. This approach aims at linking each image to a specific person and event in order not to mix them up. Each DICOM file can be seen as part of a bigger dataset.

Processing images

When manipulating DICOM files, multiple details must be taken into account.

Order

First of all, the name of the files within datasets is a 6-digit number, from 000000 to the number of images minus one. However, this order doesn't match the real order of the images. In fact, the correct order is given by the "Instance Number" tags contained in the various files. Therefore, converted images must be sorted by instance number.

Data manipulation

CT and MRI machines, as well as monitors, differ from one manufacturer to the other and even from one model to the other. DICOM takes this problematic into account by providing specific tags that allow to display the exact same representation of the data, no matter the hardware used. Otherwise, physicians may struggle to detect anomalies because of color and exposition-related variations. Therefore, before displaying or converting an image to any format (such as png), pixel data must be normalized.

The procedure depends on the tags "Window Width" and "Window Center" (one always come with the other). These are used to represent a range of values corresponding to the pixel values in the data. For instance, a window center of 0 and a window width of 200 imply pixel values between -100 and 100.

If they are missing, a simple conversion is sufficient. The parameters used to convert the data are given by two tags:

- Bits allocated: the number of bits used to represent a single pixel (value: 1 or a multiple of 8)
- Samples per pixel: the number of channels for each pixel

Examples:

- 1 bit, 1 sample: black and white
- 8 bits, 1 sample: grayscale
- 8 bits, 3 samples: RGB
- 16 bits, 1 sample: grayscale

If they are included in the DICOM header, a linear transformation must be done to convert the stored representation of the pixels to the correct visualizable one. To do this, two steps are required:

1. **Apply the Hounsfield correction**
   Hounsfield Units (HU) are used in CT images it is a measure of radio-density, calibrated to distilled water and free air. Provided that the rescale slope and the rescale intercept are included in the DICOM header, the correction is applied thanks to the following formula:

$$HU = m * P + b \tag{4.1}$$

   where m is the rescale slope, P the pixel value, b the rescale intercept.

2. **Apply a linear transformation**
   The result of the first operation then goes through a linear transformation based on the following conditions:

$$\text{if } (P \leq c - 0.5 - \frac{w-1}{2}), \text{ then } y = y_{min} \tag{4.2}$$

$$\text{else if } (P > c - 0.5 + \frac{w-1}{2}), \text{ then } y = y_{max} \tag{4.3}$$

$$\text{else } y = (\frac{P - (c - 0.5)}{w - 1}) + 0.5) * (y_{max} - y_{min}) + y_{min} \tag{4.4}$$

   where c is the window center, w window width, P the pixel input value, y the pixel output value, y_{min} the minimal value of the output range (usually 0), y_{max} the maximal value of the output range (usually 255). Equations 4.2, 4.3 and 4.4 ensure that the pixel values are correctly distributed within the output range.

4.2.3 NIfTI file format

Origin

The Neuroimaging Informatics Technology Initiative (NIfTI) file format is the successor of the ANALYZE file format. The main problem of the latter was that it was lacking information about orientation in space. Therefore, the interpretation of stored data could be problematic and inconsistent. For instance, there was a real confusion to determine the left and right sides of brain images. Hence, the NIfTI file format was defined to overcome this major issue.

Data format

Unlike the ANALYZE format that used two files to store the metadata and the actual data, the NIfTI file format stores them in one single file ".nii" but keeps this split between the real data and the header for compatibility. This has the advantage to facilitate the use of the data and avoid storing the data without the meta-data. The NIfTI format can also be compressed/decompressed on-the-fly using the "deflate" algorithm.

Overview of the header structure

With the goal of preserving the compatibility between the ANALYZE and the NIfTI formats, both headers have the same size of 348 bytes. "Some fields were reused, some were preserved, but ignored, and some were entirely overwritten". Details about the different fields contained in the header can be found here[1].

4.2.4 RAW and MHD file formats

Some datasets use a combination of RAW and MHD files. The latter contain metainformation about their corresponding RAW file(s) which contain the data. In most cases, each MHD file points to a unique RAW file whose name is the same as the MHD file name. A single RAW file can be used to represent three-dimensional data, i.e. the combination of multiple two-dimensional images. Libraries such as SimpleITK in Python allow to manipulate RAW images in an easy way.

4.3 Visualization tools

Processing data manually increases the probability of making mistakes. For that matter, visualization tools relying on the same processing code as the ones used to generate training images were developed. Their primary goal is to compare our visual representation of an image to the one obtained in professional pieces of software. These tools are convenient to visualize a dataset easily. In some cases (especially RAW/MHD images), no free software capable of reading the files was available, which made the corresponding tool useful.

4.3.1 DICOM

Visualizing DICOM files is pretty straightforward since each file represents a single two-dimensional image. However, a lot of pixel transformations and normalizations have to be applied to obtain the desired result (see section 4.2.2), which may be a source of errors. Our tools allows to display a single DICOM file as well as a sequence of files if the function is fed with a directory. Users can then scroll through the Z-axis, displaying the next or previous slice.

4.3.2 NIfTI

As a single NIfTI file can contain three- or four-dimensional data (the fourth dimension being time), our visualization tool takes this aspect into consideration by allowing to scroll through them: scrolling with the mouse goes through slices belonging to a specific timestamp over a specific axis, while the left and right arrows

[1] https://brainder.org/2012/09/23/the-nifti-file-format/

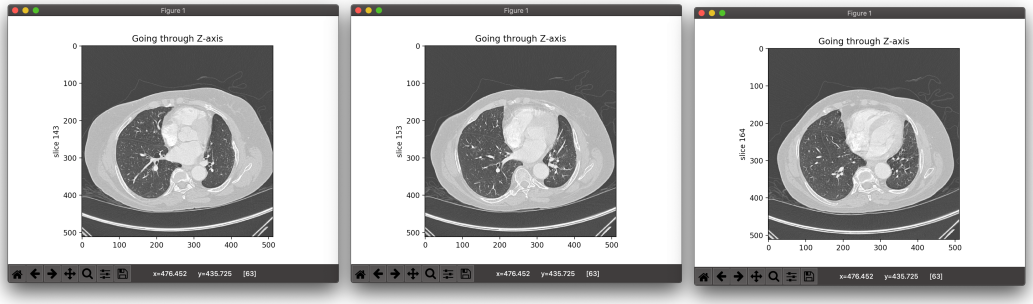

FIGURE 4.2: Visualization of a folder of DICOM files

allows to jump to the same slice at another timestamp. When reaching the end of a timestamp with the mouse, the first slice of the next timestamp is displayed. Furthermore, the three-dimensionality implies that the volume is viewable from three different perspectives. For example, a three-dimensional brain volume can display it from the top of the head to the bottom, from one ear to the other and from the back of the head to the person's face. Therefore, the user can choose a specific axis to navigate through. If no axis is chosen, all three perspectives are shown one after the other.

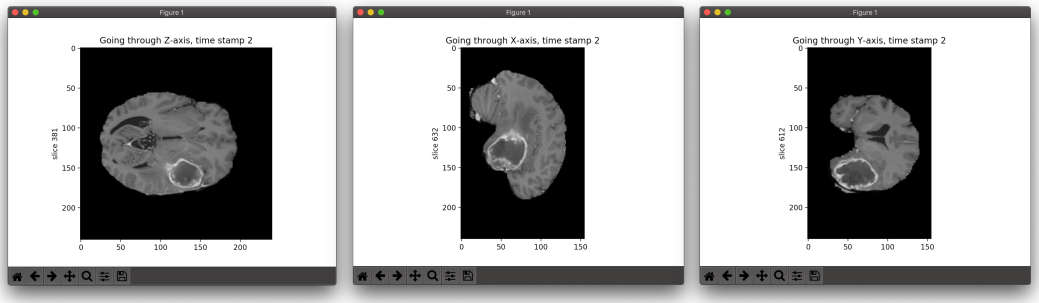

FIGURE 4.3: Visualization of a four-dimensional NIfTI file

4.3.3 RAW

Medical RAW files are three-dimensional, which means that a single file contains a volume, i.e. a succession of 2D slices. Scrolling with the mouse goes through slices over a specific axis. Like the NIfTI file format, the user can profit from the three-dimensionality by displaying the body part from three different perspectives. This tool was particularly useful since no free software capable of handling these files was available. Photography-related programs such as Photoshop can open RAW pictures but not these three-dimensional ones.

4.4 Conversion to PNG

In addition to the visualization tools, libraries allowing to convert medical files into PNG files were developped. The main one is called "anydatasettopng" and is the one to use to convert any file format to PNG since it makes use of the others. Given a root directory, it scans the latter and its subdirectories to find all DICOM, NIfTI and RAW/MHD files. Then, each file is converted to PNG and saved into the same

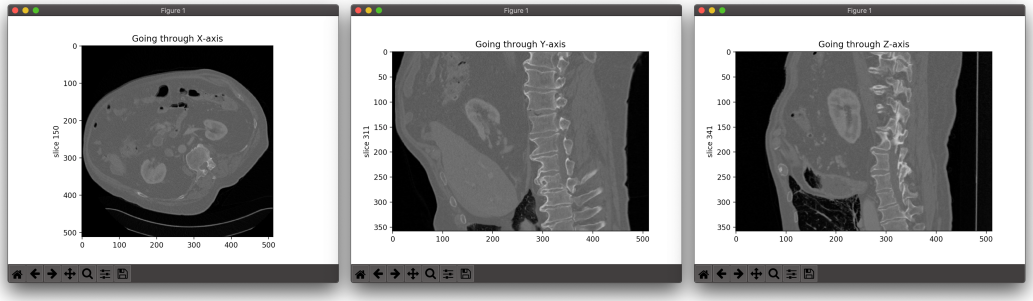

FIGURE 4.4: Visualization of a three-dimensional RAW file

directory as the one it came from. This library allows to keep an easily accessible visual representation of medical files without having to rely on a specific piece of software to display them. Finally, it also played the role of debugging tool since the conversion techniques are the same as the ones used further in this work.

4.4.1 8-bit conversion

Medical data is often represented over 16 bits. However, exporting images to PNG require 8-bit images. To transpose a 16-bit image to an 8-bit one, the procedure described by algorithm 1 was applied.

Algorithm 1 16 to 8 bits conversion

1: **procedure** 16_TO_8_BITS_CONVERSION($pixel\_array$)
2: $Pixel_{min} \leftarrow$ minimal pixel value in $pixel\_array$
3: $Pixel_{max} \leftarrow$ maximal pixel value in $pixel\_array$
4: **for** $pixel\_value$ in $pixel\_array$ **do**
5: $pixel\_value \leftarrow \frac{(pixel\_value - Pixel_{min}) * 255.0}{Pixel_{max} - Pixel_{min}}$
6: Modify object type to 8 bits
7: Export $pixel\_array$ to PNG

Chapter 5

Paper reproduction

This chapter is based on the article "Computer-Aided Diagnosis of Prostate Cancer Using a Deep Convolutional Neural Network from Multiparametric MRI" from Song et al. [3], shortly presented in chapter 2. It aims at reproducing the experiment of the paper in order to acquire the medical, theoretical and technical background before proposing a transfer learning method as a way to improve the classification using other body parts (see Chapter 6).
Song et al. [3] proposed a deep convolutional neural network (DCNN) method to detect prostate cancer based on the SPIE-AAPM-NCI PROSTATEx Challenge dataset. This dataset is one of the biggest dataset available for prostate cancer classification. The two different output classes of the latter are benign lesion (class 0) and malignant lesion (class 1). The dataset contains a total of 204 patients for the training (labels are known) and 208 patients for the challenge (labels are unknown). This paper explains all the steps to follow with the goal of building a computer-aided diagnosis system for prostate cancer detection. Moreover, it also provides results (AUC) about the algorithm performance on a test set they built themselves (which is not the same as the challenge test set). In fact, they split the PROSTATEx training set into a training set, a validation set and the mentioned test set. These results can be used as a benchmark to compare with the results of the reproduction of the experiment. However, the article gives no information about the results their model achieved on the official PROSTATEx challenge test set (208 patients without labels). Therefore, the reproduction of the experiment will fill that gap by taking part in actual challenge.

This chapter is built in a top-down fashion. First, an overview of the entire process is established in order to understand the purpose of the experiment as a whole. Then, each theoretical notion is described in detail. This includes the structure of the dataset, the steps involved in its processing, and the techniques used as a verification of the proper functioning of the algorithm.Then, the training phase is dealt with in depth and all hyperparameters, options and implementation choices are described to ensure the reproductibility of the experiment. This part is followed by the presentation of the raw results the model achieved on the test set (the one built for the experiment), which is itself followed by their analysis in the "Discussion" section. Finally, the last section is devoted to the so-called SPIE-AAPM-NCI Prostate MR Classification Challenge. It presents the results that the model achieved on the challenge test set using the whole training set for the training. The latter section is not part of the original paper.

5.1 Process overview

Schematically speaking, the entire experiment process can be represented as shown on Figure 5.1. The first part of the experiment is about the reproduction of the paper experiment, whereas the second one is devoted to the participation in the SPIE-AAPM-NCI PRO STATEx challenge since the authors of the article did not take part in it.
The former makes use of the PROSTATEx training set. This dataset is only composed of samples whose clinical significance is provided (labeled data). The first step of the reproduction is the data processing. In fact, many processing steps are involved to transform the original DICOM files into NumPy arrays which can be fed to the neural network. Data processing includes lesion cropping, normalization, MRI images stacking, data augmentation, etc. (see Section 5.2.1 for further details).
Once the processing part is completed, the labeled data are split into a training set, a validation set and a test set using respectively 80%, 10% and 10% of the available data for each subset. The training set samples are then fed to the neural network in batches. At the end of each epoch, the current model is tested on the validation set and the resulting metrics are plotted on Tensorboard (see 5.2.3). The validation set is independent of the training set and has never been seen by the model, which gives an indication of how well the latter generalizes. Then, if the current model reached a higher AUC and a higher accuracy than the previous best model, it becomes the new best model and is saved. Furthermore, if the validation AUC is decreasing for a defined number of epochs, the learning rate is reduced to prevent overfitting and avoid a progressive decrease in validation performance. This technique contributes to maintaining a good generalization level and optimizes the training with respect to the AUC. As soon as the training phase reaches the defined number of epochs, the model is tested on the test set with the authors' method called "enhanced prediction" to compute the AUC. The latter method consists in predicting an output for each augmented image before averaging the predictions for each lesion of each patient.

The second part of the experiment relies on the results of the first one. Indeed, the same data processing pipeline is used and the hyperparameters used to train the neural network are the same. The difference lies in the fact that the entire dataset (i.e. the entire PROSTATEx training set) is used as training set instead of splitting the data into a training/validation/test set. In this way, the model sees more training examples, which maximizes the performance of the model on the challenge test set by increasing its generalization ability. During the training phase, the model is saved at the end of each epoch. Once the training is over, multiple models are chosen around the epoch that produced the best result on the validation set during the first part of the experiment. These models are then evaluated on the challenge test set by submitting their predictions for each lesion to the Grand Challenge organization. The model with the best result on the challenge is considered as the most efficient one.

5.2 Reproducing the paper experiment

5.2.1 PROSTATEx: Data processing

This section describes all processing operations performed on the raw data until it passes through the neural network. All steps mentioned below are also valid for the second part of the experiment about the challenge (Section 5.3).

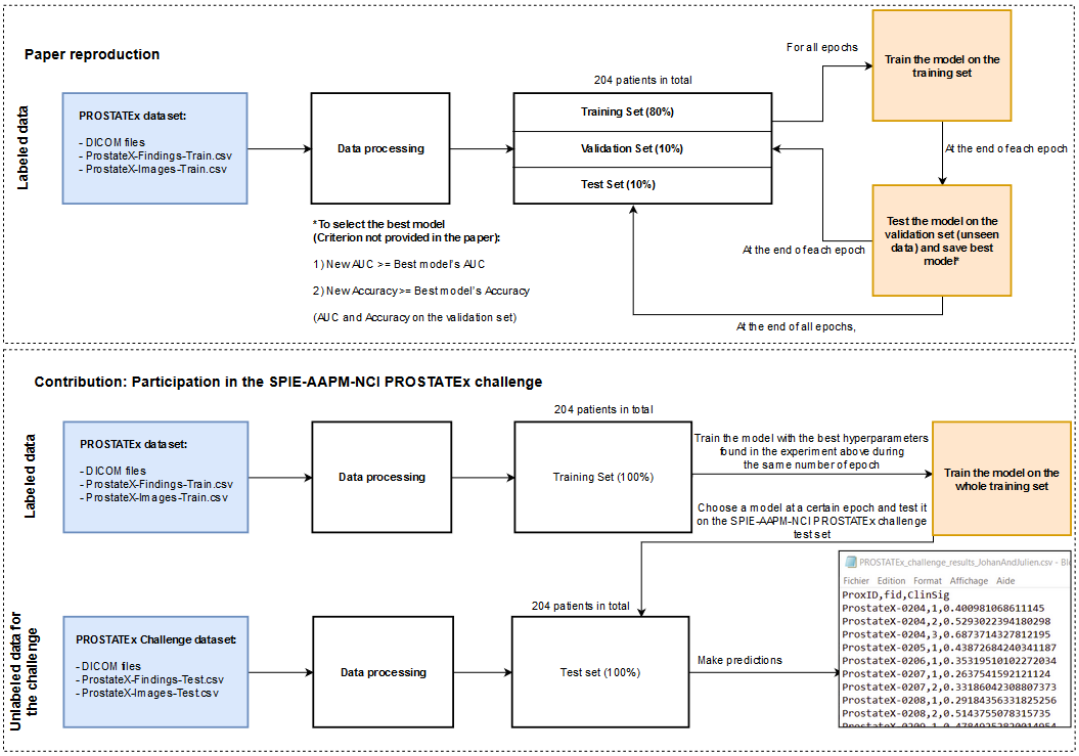

FIGURE 5.1: Overview of the whole experiment process

Dataset description

The SPIE-AAPM-NCI PROSTATEx Challenge dataset is publicly available on the Cancer Imaging Archive. This dataset is composed of multiparametric MRIs (T2W, DWI, ADC, DCE, PD, Ktrans) for a total of 204 training patients and 208 challenge patients. Images were taken under the sagittal, transverse and coronal planes.

The PROSTATEx dataset comes with two CSV files for the training set. The first one, *ProstateX-Findings-Train.csv*, lists all findings with their clinical significance. Multiple findings can be associated with the same patient (ProxID). The second one, *ProstateX-Images-Train.csv*, gives information about where to find the right DICOM file for each patient and each finding. Important labels are "ProxID" (patient ID), "fid" (finding ID $\in [1, \infty]$), "ClinSig" (clinical significance, TRUE or FALSE), "DCMSerNumber" (digit before the dash in the folder name containing DICOM files), "ijk" (position of the lesion: slice number k at coordinates (i, j), $i, j, k \in [0, \infty]$) and "VoxelSpacing" (3-dimensional vector representing the correspondance between a pixel and the space it occupies in the real world). Both CSV files are complementary to each other. Regarding the challenge patients, two analog CSV files are provided: *ProstateX-Findings-Test.csv* and *ProstateX-Images-Train.csv*. The only difference is the absence of clinical significance. Finally, images are in the DICOM file format (see Section 4.2.2).

Methodology

The methodology described in this section is the closest replication possible of the authors' data processing methodology. Most of it were reproductible in a similar way, apart from the manual lesion contouring which was performed by a qualified radiologist in the authors' case. Furthermore, the authors didn't mention any participation in the official PROSTATEx challenge. In order to get an unbiased evaluation

of the performance of our model, these images were also processed as explained below in order to take part in the challenge, but were not augmented. Nevertheless, the authors created their own test set by splitting the PROSTATEx training images into a training set (80%), a validation set (10%) and a test set (10%). They mentioned that the test set was augmented 11 times but gave no information about the training and validation sets. Since the training set is imbalanced (3 false for 1 true), the true class was augmented more times than the false class (60x) in order to create balanced training and validation sets.

From DICOM to NumPy arrays

Before anything else, T2W, DWI and ADC grayscale images were used, which means that DCE, PD and Ktrans images were left aside. Moreover, only images showing the prostate under the transverse plane were used. The reason for this is that the tumors are much more visible under this perspective. The first step consisted in converting DICOM files. Algorithm 2 describes the steps involved in converting PROSTATEx's DICOM files to NumPy arrays. The right slices were found thanks to the two CSV files. Important information such as the patient ID, the sequence, the lesion location and the voxel spacing were used as file output names, which allows to use these files independently for the next steps (i.e. without relying on the CSV files).

Algorithm 2 PROSTATEx preprocessing

procedure MAIN($dataset\_folder, findings\_CSV, slices\_CSV, output\_folder$)
    Create output directories: "$output\_folder/True$", "$output\_folder/False$"

    $findings \leftarrow read\_CSV(findings\_CSV)$ ▷ ProstateX-Findings-Train.csv
    $slices \leftarrow read\_CSV(slices\_CSV)$ ▷ ProstateX-Images-Train.csv
    $meta \leftarrow merge(findings, slices)$ ▷ Both CSV files are complementary to each other.

    for *row* in *meta* **do**
        $patient\_id \leftarrow row["ProxID"]$
        $finding\_id \leftarrow row["fid"]$
        $mri\_type\_number \leftarrow row["DCMSerNumber"]$
        $clinical\_significance \leftarrow row["ClinSig"]$
        $img\_i, img\_j, img\_k \leftarrow row["ijk"]$
        $slice\_number \leftarrow img\_k + 1$ ▷ CSV indexing in $[0, \infty]$, DICOM in $[1, \infty]$

        for *visit in patient_id*'s folder **do**
            for *mri_type* in *visit* **do**
                if *mri_type* starts with "*mri_type_number*-" **then**
                    for *dicom_file* in *mri_type* **do**
                        if $slice\_number == dicom\_file.InstanceNumber$ **then**
                            $slice \leftarrow normalize\_dicom(dicom\_file)$ ▷ see Section 4.2.2
                            Save *slice* in "$output\_folder/clinical\_significance$"

From NumPy arrays to augmented stacked images

This step can be split into two substeps: aligning and stacking images before augmenting them. Once full images were converted to NumPy arrays, the images related to a specific lesion of a specific patient needed to be aligned and resampled to the same resolution. In fact, T2-weighted images have a much higher resolution than the DWI and ADC images (in this dataset at least). Without this operation, a single pixel on a DWI image would have covered a lot more tissue than a pixel of the corresponding T2-image. To perform the alignment, the lesion was localized on the three sequences than to the CSV file. Then, the goal was to crop a large patch which contained the same amount of tissue on the three images. This patch had to be centered on the lesion to ease the augmentation process. So, a fixed patch size was defined for the T2-image since its resolution was the highest. Then, using the voxel spacing information, the patch size required to cover the same amount of tissue was computed for the two other sequences (DWI, ADC). Finally, the three images were cropped, resized to the same resolution and stacked. Stacking images consists in putting each single image (grayscale) into a single array, as three different channels. At this point, the pixel i, j of the three channels represents the exact same tissue, which means that the lesion is at the same position and is covered by the same number of pixels on each channel. This approach increases the probability of detecting a cancer by ensuring a good visibilty for each lesion, since the latter is not necessarily as visible on the three images.

An important part of this process is the normalization. In fact, images were normalized before beeing stacked, based on a Z-score, i.e.

$$Pixel_{i,j_{normalized}} = \frac{Pixel_{i,j} - \mu}{\sigma} \tag{5.1}$$

where $Pixel_{i,j} \in [0, 255]$ is a grayscale pixel value, μ is the mean value of all images of the corresponding sequence for this patient and σ the standard deviation of all images of the corresponding sequence for this patient. According to the authors, normalizing each sequence of each patient separately allows to keep slight contrast nuances which ultimately helps the final diagnosis. In other words, all T2-weighted images belonging to a specific patient were concatenated into a single NumPy array. Then, the mean and standard devation of the array were computed. These values served as μ and σ to normalize the T2-images of this specific patient. Same for the patient's DWI and ADC images. The same process was applied to all patients separately.

Finally, the dataset was augmented in the same way as the authors. Concretely, images were rotated (-20 to $20°$), flipped horizontally (probability of 0.5) and slid horizontally (value $\in [-1, 0, 1]$ pixel). These techniques alone allowed to create a sufficient amount of data. Therefore, two other augmentation methods used by the authors (horizontal stretching by a factor $\in [0.9, 1.1]$ and vertical sliding) were not used. Images were exported as NumPy arrays.

From NumPy arrays to augmented non-stacked images

Another way of processing and using the data was experimented. Instead of stacking the images, each image was cropped and augmented separately. At the end, each image was given as separate input to the neural network.

5.2.2 Data processing verification

Cropping verification using red dots

In order to check the correctness of the augmentation process, images with a red dot at the lesion coordinates were generated. First of all, a red dot was placed on the full image thanks to the "i,j,k" attribute in the CSV file. This full image was exported. Then, images were augmented in the exact same manner as described in section 5.2.1. The expected result satisfies the following properties:

- The red dot is localized in the center of the cropped and augmented image.
- The three cropped and augmented images within a stacked image contain the exact same tissue.
- The three cropped and augmented images within a stacked image are not rotated/shifted/flipped differently.
- A lesion is visible to the naked eye.

Figure 5.2 shows an example generated during this test. While beeing exactly the same for the 3 images, the red dot looks bigger on the images whose resolution was smaller since it was drawn before resampling the images. These images come from the same exam performed on the same patient on the same day.

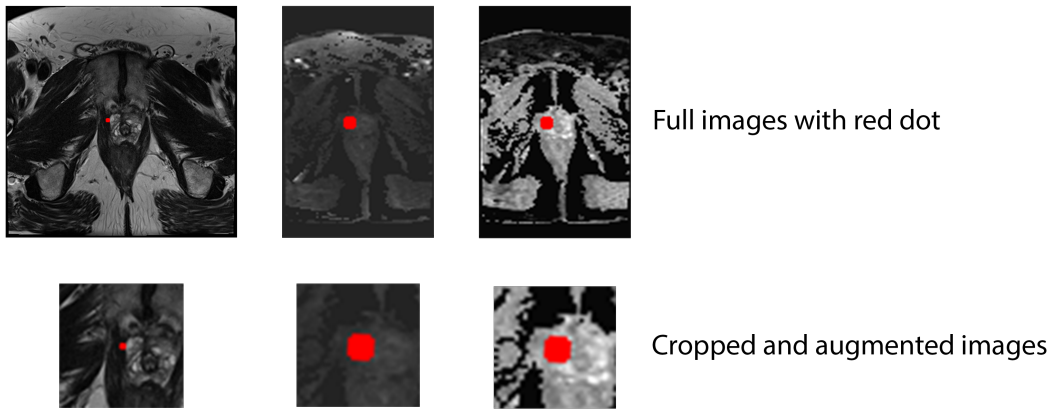

FIGURE 5.2: Red dot test - Patient 0082, Finding ID 1
From left to right: T2, DWI, ADC.

Alignment

In addition to the previous test, an additional one aiming at checking the alignment of the images was created. This test ensures that the three images show the exact same content at the same spot over the three channels.

5.2.3 Training the neural network

Training the neural network is the most important step of the experiment after the data processing. This section first describes the architecture of the model, then presents the script used for the training and finally provides details about the experimental setup such as the training procedure, the hyperparameters of the model and the configuration of the machine on which it was trained.

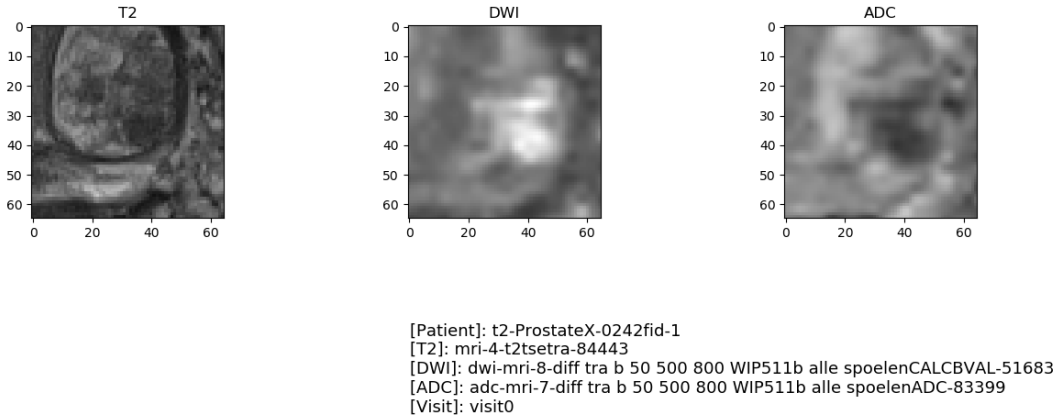

FIGURE 5.3: Alignment - Patient 0242 - Finding ID 1

Architecture

The model architecture used in the paper is a modified version of the VGG network from the Oxford's Visual Geometry Group (VGG). This model was initially designed as part of the Large Scale Visual Recognition Challenge 2014 that used the ImageNet dataset composed of 14 millions images belonging to 1000 classes. Figure 5.4 illustrates its structure and its corresponding implementation in Python with the PyTorch framework.
It is first composed of three convolution-dropout-max-pooling blocks followed by three fully-connected-dropout blocks. Each convolutional box (in blue) in the figure represents in reality three layers: the convolutional layer, the batch normalization layer and the exponential linear unit (ELU) activation function. The same principle applies to the fully connected layer box (in orange) that is divided into a fully connected layer followed by the exponential linear unit. The last fully connected box (in purple) has the same structure except that the exponential linear unit is replaced by a softmax function for classification.
In comparison to the original VGG, this model keeps the small filter size of 3x3, also doubles the number of filters after each convolution-dropout-max-pooling block and has a stride of 1 for all convolutions. It differs from the traditional VGG in that it makes use of a smaller number of layers since the task is simpler than the original one. Moreover, it uses exponential linear units instead of rectified linear units as activation functions and adds dropout and batch normalization layers. Also, it uses 1x1 convolutions.

Script options

Since training a neural network requires datasets, hyperparameters and many more configuration choices, the creation of a generic script that accepts multiple options is necessary. This requirement was solved thanks to the Python module called "argparse". The latter automatically generates help/usage messages and displays errors when the arguments typed in by the user are invalid. Table 5.1 shows all options accepted by the script, provides information about what they concretely mean and indicates their respective types.

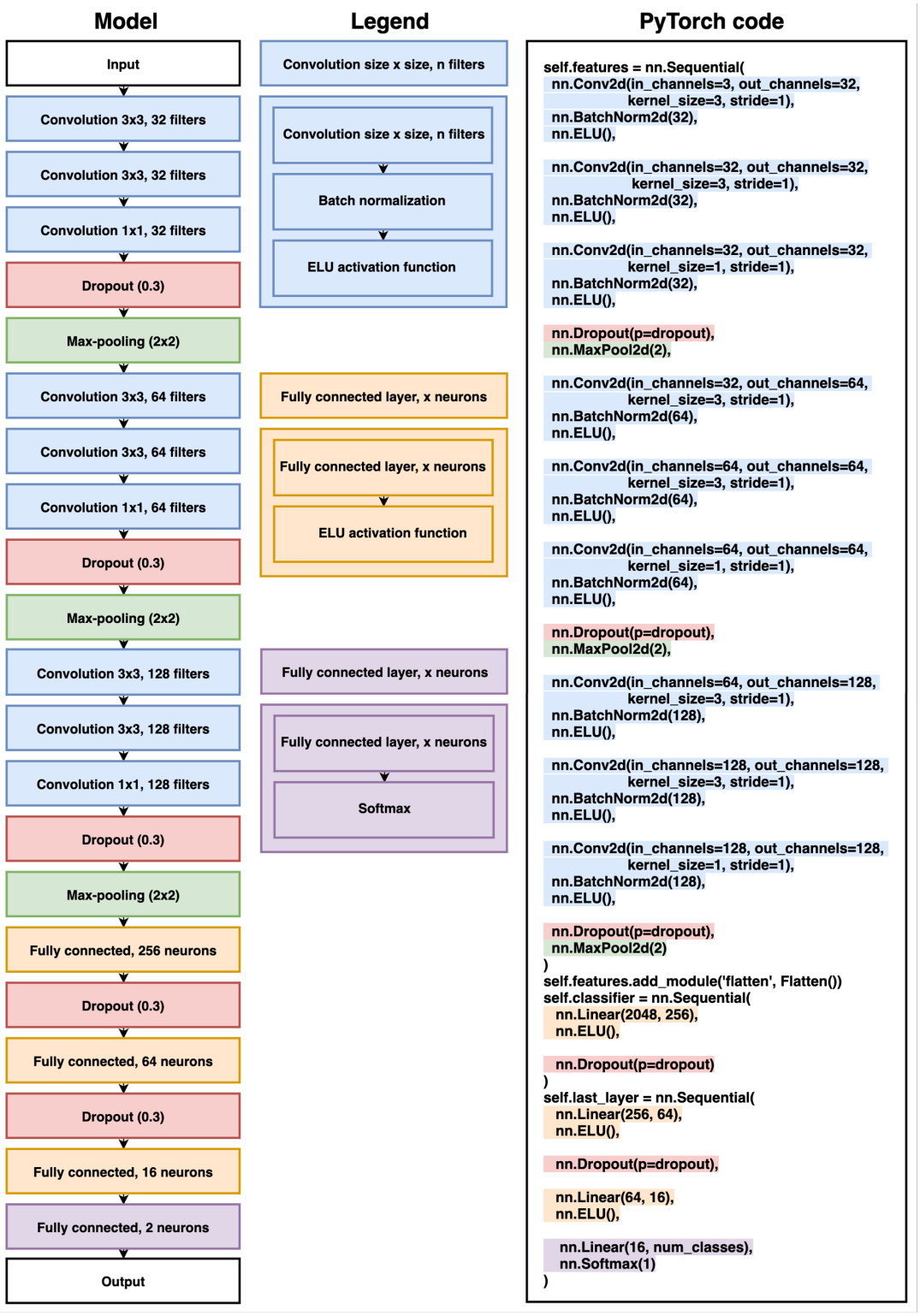

FIGURE 5.4: Model architecture with the corresponding PyTorch code

Tensorboard

Tensorboard's official website [31] claims that "Tensorboard provides the visualization and tooling needed for machine learning experimentation:

- Tracking and visualizing metrics.

Command	Description	Required	Type
–trainingSet1	Training set path	True	String
–validationSet1	Validation set path	True	String
–batchsize	Number of samples per batch	True	Int
–nbepochs	Number of epochs the training phase has to last	True	Int
–lr	Learning rate used in the optimizer	False Default: 1e-3	Float
–lossfunction	Loss function name [CrossEntropyLoss, L1Loss, MSELoss]	False Default: 'CrossEntropyLoss'	String
–cuda-device	GPU name to run the experiment	False Default: 'cuda'	String
–modeltoload	Pretrained model name If given, load it, otherwise randomly initialize it	False Default: ""	String
–dropout	Dropout probability	False Default: 0.3	Float
–optimized-metric	Metric to optimize The best model during the training will be saved according to it ['auc', 'accuracy', 'precision', 'recall', 'f1score', 'specificity']	False Default: 'auc'	String
–outputdirectory	Root of the output directory used by Tensorboard to save the models	True	String

TABLE 5.1: Complete list of script options

- Visualizing the model graph (ops and layers).
- Viewing histograms of weights, biases, or other tensors as they change over time.
- Projecting embeddings to a lower dimensional space.
- Displaying images, text, and audio data"[31].

In this experiment, Tensorboard is mainly used to plot Matplotlib figures of the model performance. In fact, during the training the following metrics are computed: loss, accuracy, precision, recall, F1-score, specificity and AUC. These metrics are computed separately on the training and on the validation sets at the end of each epoch. They are then stored and plotted on the same figure at the end of the process thanks to Tensorboard. In addition to this, written reports regarding which model was the best and the results it achieved are also added to the dashboard.

Model roulette

As stated by the authors, "the training process is sensitive to the parameter initialization" [3]. Experience showed us that the exact same hyperparameters with two different initializations could give opposite results. Therefore, in order not to train the model vainly, a script which generates a given number of initializations was created. Each untrained model was then tested on the validation set since the best model during training is saved according to its performance on the validation set. The model whose score was the highest on a given metric was saved and used as base model for the experiments. It can then be used as initialization model using the script option *–modeltoload <path>*.

Experimental setup

Keeping the exact same hyperparameters as the original paper lead the experiment to disappointing results. Consequently, new hyperparameters had to be found. A grid search approach driven by the AUC obtained on the test set (the one built for the experiment, not the one of the challenge) was chosen. In practical terms, this consisted in trying multiple combinations of hyperparameters and keeping the one that produced the best AUC on this test set. At the end of this process, it appeared

that the best model was obtained when the data was organized in batches of 32 samples, with an initial learning rate of $1e{-}^{7}$, a learning rate decay on plateau using a factor of 100 with a patience of 2 (the learning rate is divided by 100 when the validation AUC decreases for two consecutive epochs) and a dropout rate of 0.2. Note that the model was initialized with the best one among 200 models coming from a roulette maximizing the f1-score. This model was implemented with PyTorch using the Adam optimizer to update its weights and was trained on an Nvidia GeForce GTX Titan X graphics card during approximately 3 hours.

5.2.4 Training verification

In deep learning, two main problems can occur during training. The first one is called the "vanishing gradient" problem and refers to the fact that it is possible for the loss function to compute extremely small gradients, near zero. Consequently, the weight update is also extremely small, which makes the neural network hard or impossible to train. The second problem, the "exploding gradient" is the opposite. In this case, large gradients are computed, which leads to huge weight updates that can even reach "NaN" values. This makes the model unstable and unable to learn from training data.
As a result, it is important to analyse the gradient propagation across the network to ensure that it is not facing such problems. To achieve this goal, a visualization tool to display the gradients of the entire network was implemented. This implementation is based on the code of RoshanRane [32]. This allows to visually notice the gradient flow at a glance and to see its evolution during all epochs (one visualization per batch is generated).

Gradient flow visualization

The visualization displays the gradient flow through the entire neural network. It shows the name of each layer on the x-axis and its average gradient value on the y-axis (recall: for each layer the gradient is a matrix). Furthermore, it also gives information about the max value of the gradient and indicates in black if one of them is equal to zero.
The figure 5.5 shows the gradient flow at epoch 0, batch 6. From the output layer (called "last_layer0.weight" on the graph) to the first layer, the gradient is well propagated: no huge average gradients are registered, neither extremely small ones. The training process goes on correctly.

5.2.5 Results

Figure 5.6 shows the different metrics registered on the training and validation sets during the training phase. The best model was chosen at epoch 21 with an accuracy of 0.7525 and an AUC of 0.765 on the validation set (recall: the condition for a model to be chosen as best is that the current AUC and accuracy should be higher than the AUC and accuracy of the previous best model).
This best model was evaluated on the test set built for the experiment and achieved an AUC of 0.75 using their "enhanced prediction" technique (see figure 5.7).

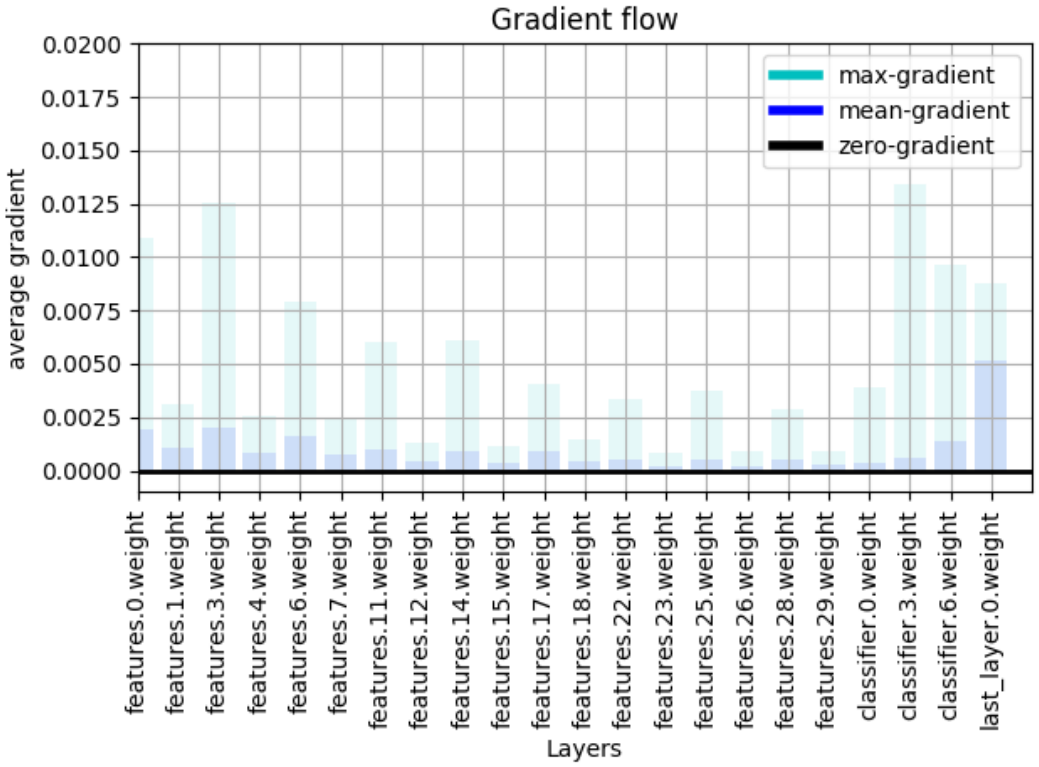

FIGURE 5.5: Gradient flow at epoch 0 of the experiment, batch 6

Finally, the model was also tested on the official SPIE-AAPM-NCI Prostate MR Classification Challenge test set and achieved an AUC of 0.71 (see figure 5.8). This result will be used as benchmark in comparison with the model trained on the whole dataset in the second part of the experiment.

5.2.6 Discussion

Thanks to the curves of Section 5.2.5, a lot of information regarding the model performance can be inferred. In fact, "reviewing learning curves of models during training can be used to diagnose problems with learning, such as an underfit or overfit model, as well as whether the training and validation datasets are suitably representative"[33].
First of all, when observing the loss plot, what is noticeable is that the training and validation losses have the same behavior during the 50 epochs. This is a clear sign of good fit of the model on the data. All the features learned by the model on the training data are also applicable to the validation set. This situation is the optimal one. Regarding the slope of the curve, a big change occurs at epoch 17. This moment corresponds to the one where the learning rate is reduced by a factor of 100 thanks to the *reducelronplateau* function. In fact, at this moment, the validation AUC (the training is optimized according to this metric) was at its optimum with this learning rate. If this technique had not been used, the training loss would have continued to go down whereas the validation loss would have gone up immediately, which would have lead to overfitting. Here, thanks to the learning rate decay, this scenario is avoided and the validation and training losses stay stable from this point onward. The same behaviour is clearly noticeable for all other metrics.
Regarding the overall performance of the best model chosen at epoch 21, the latter

generalizes well. This is particularly visible on the accuracy and AUC plots where the validation metrics are higher or equal to the training ones. The validation accuracy indicates that the model made the right class decision in approximately 75% of the time. Furthermore, since the model achieved a validation AUC of 0.76, it shows that it had a good distinction capacity. In fact, the highest the AUC is, the highest is the ability of the model to predict benign lesions as benign and malignant lesions as malignant. Samuel G. et al. [6] claim that a less-experienced radiologist can reach an AUC of 0.81 and an expert an AUC of 0.91. Consequently, this model is almost at the level of a radiologist, which is very promising. It is also important to keep in mind that the model was trained using only 164 patients (80% of the 204 training patients), which suggests that even better results can be reached with more data.
The precision plot, which is used to measure the proportion of malignant lesions correctly classified among all lesions classified as malignant lesions, confirms the hypothesis that the model has a good distinction capacity with a score of approximately 0.76 on the validation set.
Another extremely important aspect of a computer-aider diagnosis system is to avoid the number of false negatives as much as possible. In fact, the consequences of classifying a malignant lesion as benign are far worse than the opposite. This concept is measured by the recall, which quantifies the proportion of malignant lesions correctly classified among all malignant lesion. The model has a recall of around 0.74 on the validation set.
The two last metrics (precision and recall) are both considered in the f1-score that is equal to 0.76.
Finally, the specificity indicates the proportion of correctly classified benign lesions among all benign lesions and is equal to approximately 0.76.
Globally, this model is balanced as no metric is exploding at the expense of another. This stability constitutes a serious strength regarding its generalization ability.

As the goal of the experiment was to reproduce the experiment of the paper, a comparison between the two models is necessary. The reproducted model achieved an AUC of 0.75 on our test set (see Figure 5.7) using the authors' "enhanced prediction" technique, whereas the authors achieved an AUC $\in [0.876 - 0.994]$ with 95% of confidence on their test set. Consequently, the model presented in this work has a slightly smaller AUC of $[0.126 - 0.244]$ than the one achieved in the paper. Multiple factors can explain this difference. First, in their work, the cropping of lesions was performed by a qualified radiologist, whereas, in our case, it was automatically done using the information in the CSV files. Then, the way the test set is built can make the results vary a lot. In fact, the authors didn't provide any information about how they built their test set. It is therefore possible that the test set was chosen to maximize the performance of the model. In the current work, the test set is built randomly. Consequently, the comparison of the performance is difficult since the test set is not the same in the two works. The only way to seriously compare the two models would be to compare their performance on the official SPIE-AAPM-NCI Prostate MR classification challenge test set. The model of this section (trained with 80% of all the data available) reached an AUC of 0.71. Unfortunately, no information about the performance of the model of the paper is given. Knowing that the best result at the time of the challenge was an AUC of 0.87 on the challenge test set, it is highly surprising not to take part in it when claiming an AUC in the interval $[0.876 - 0.994]$ with 95% of confidence... Finally, another source of difference is the number of participants in the experiment and the level of experience they have. Indeed, seven authors contributed to the paper, including one bachelors' student and six PhD with one of them working for Siemens Heathcare and one for hImagingTek Ltd. These

Epoch	15	17	19	20	21	22	23	30
AUC	0.75	0.76	0.76	0.76	0.76	0.76	0.75	0.75

TABLE 5.2: AUC of the model saved at different epochs on the official SPIE-AAPM-NCI Prostate MR classification test set

big companies are specialized in the medical imaging and have departments dedicated to machine learning. On the opposite, this work was produced independently by two MSc students without previous experience in the medical imaging field.

5.3 SPIE-AAPM-NCI Prostate MR Classification Challenge

This section aims at maximizing the result on the SPIE-AAPM-NCI Prostate MR classification Challenge test set by using the entire dataset as training set, instead of using 80% of it, as it was done in the previous section. First, the changes needed to reach this new goal are detailed, then the raw results are presented and finally the latter are interpreted.

5.3.1 Training the neural network with the whole dataset

The first part of the experiment, in which the whole dataset was split into a training (80%), a validation (10%) and a test set (10%), was extremely useful to find the right hyperparameters. It also gave the opportunity to check that the model and the processing of the data worked as expected with the goal of creating a state of the art deep learning model for prostate lesions classification.
As a complement, this part of the experiment focuses on the challenge results in order to improve the 0.71 in AUC achieved by the model trained with 80% of all available data. In a general way, the more data are available the better will be the model. Consequently, the idea behind this section is to use the entire dataset as training set, discarding the validation and test sets from previous section. Apart from this new split, the data processing and the hyperparameters keep the same. The only difference in the training is about the learning rate decay. Indeed, since the validation set does not exist anymore, there is no possibility to use the *reducelronplateau* function to prevent overfitting. However, this does not cause any problem because, thanks to the results of the previous section, it is clearly visible that the model would have started to overfit at epoch 17 if no learning rate correction had been applied. Similarly, it is possible to conclude that in this case the best model should also be the one around epoch 17. In order to ensure this, the models at epoch 15, 17, 19, 20, 21, 22, 23 and 30 were evaluated on the official challenge test set. For each of these models, a csv file containing the predictions for each lesion of each patient was automatically generated and sent to the challenge.

5.3.2 Results on the challenge test set

Figure 5.9 presents the metrics achieved by the model trained with all available data as training set at each epoch.
Figure 5.2 shows the performance of the model saved at different epoch on the official challenge test set. The best AUC achieved on the challenge is 0.76. Figure 5.10 illustrates the participation to

5.3.3 Discussion

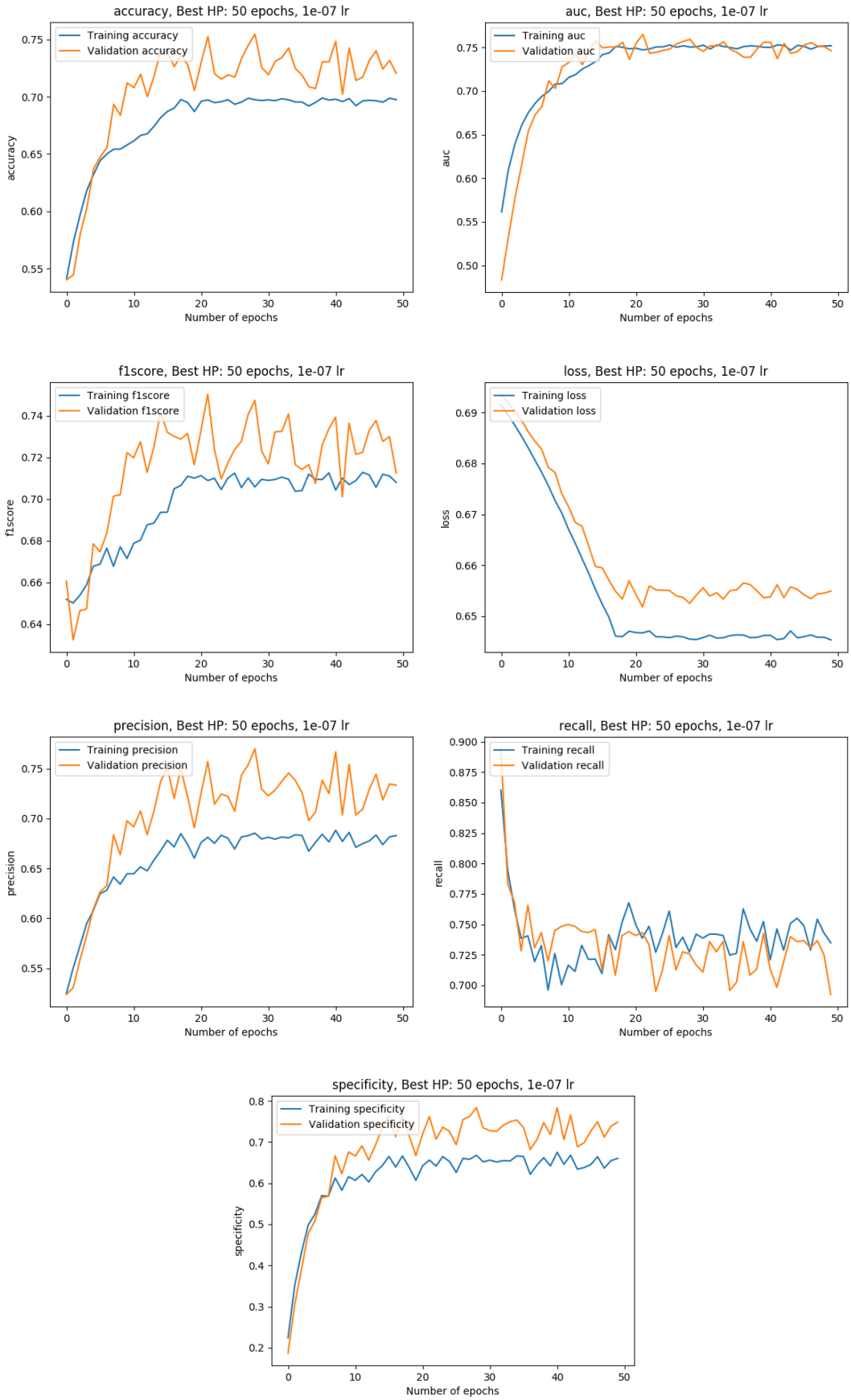

FIGURE 5.6: Metrics on the training and validation sets

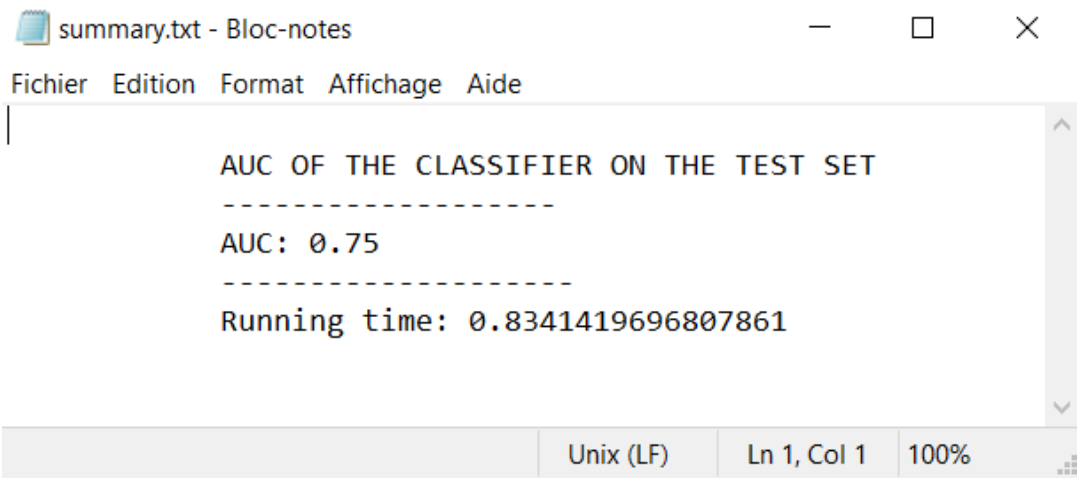

FIGURE 5.7: AUC of the best model on our test set

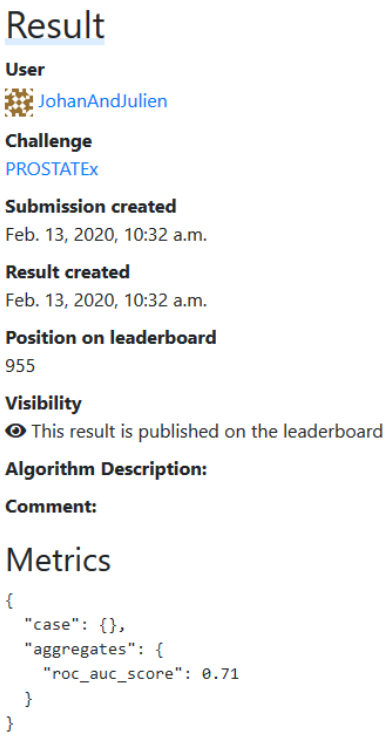

FIGURE 5.8: Score achieved in the PROSTATEx challenge using only our custom training set (80% of all available data) for the training

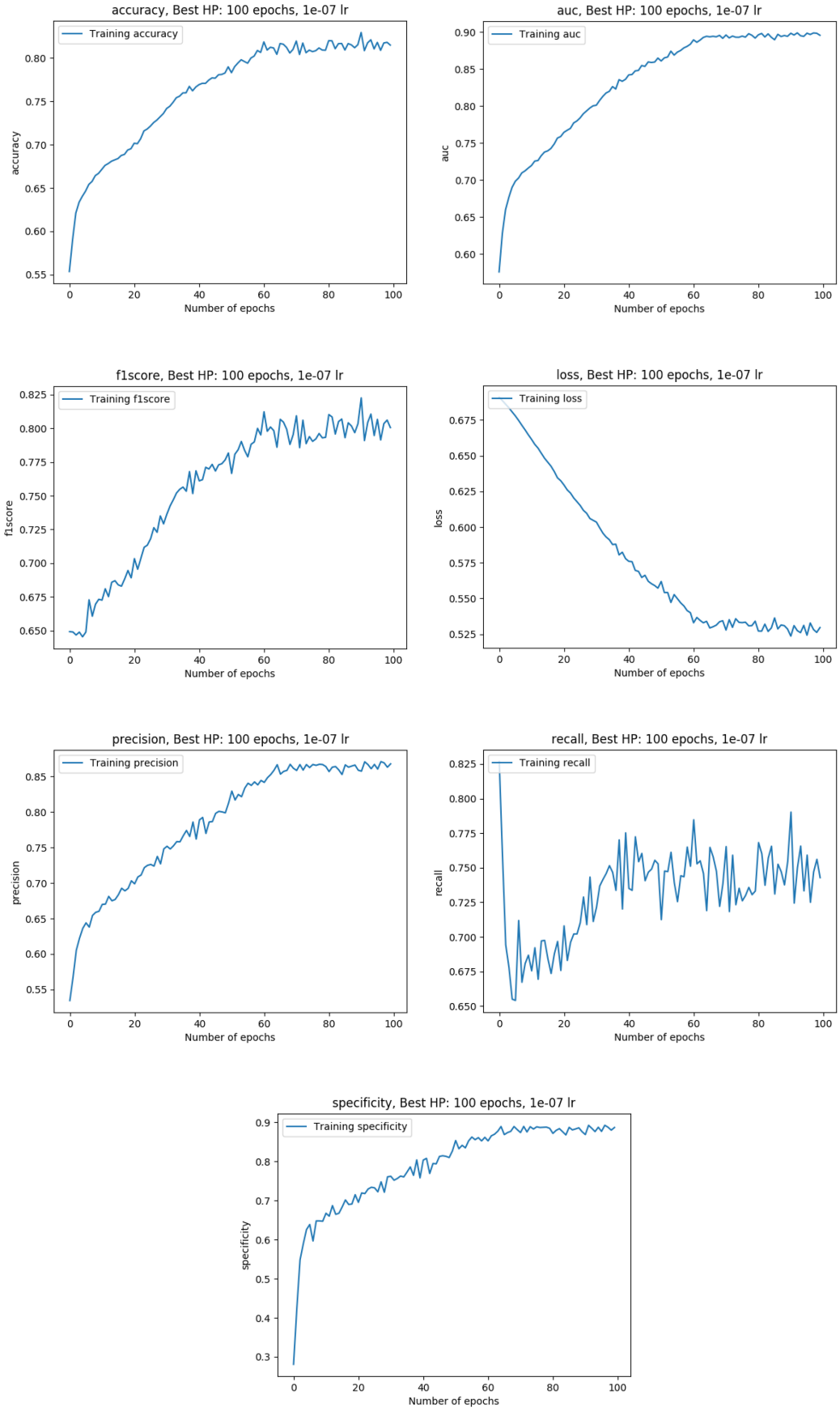

FIGURE 5.9: Metrics on the training using all data available as training set

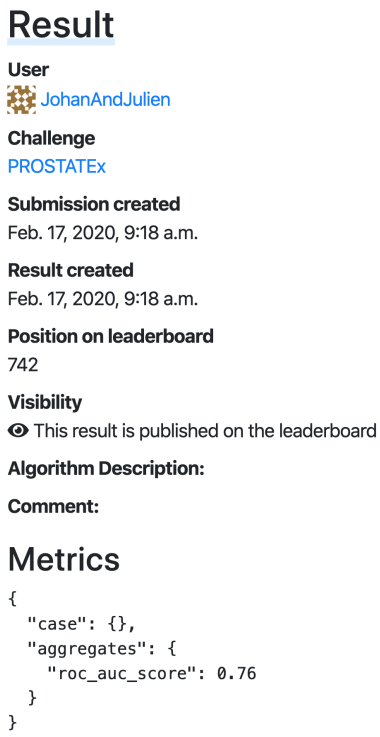

FIGURE 5.10: Score achieved in the PROSTATEx challenge using all available data as training set

Chapter 6

Improving performance using transfer learning

This chapter proposes a technique to improve cancer detection and classification despite the lack of publicly available medical data. To achieve this, transfer learning is used.
First, a general definition of transfer learning is given. Its utility in this specific application is discussed as it is useful but not in any case. Second, the different steps involved in the process are described in order to get an overview of the whole transfer learning pipeline that was applied. Then, a section focuses on the processing of new datasets that are used in the final experiment. Finally, the latter and its results are discussed.

6.1 Goal

Transfer learning consists in using a network which was pretrained on a dataset A to improve the performance of the task on a dataset B. This technique makes sense when the quantity of data available is not sufficient and when the data of both datasets share common features. Both conditions are fulfilled in the case of cancer detection. In fact, medical data are highly confidential and cannot be shared easily. Provided that medical institutions agree to share it, it has to undergo an anonymization process which follows very strict rules. Then, to be usable as part of a deep learning classification task, data must be organized and described precisely. In other words, additional information such as the clinical significance, the position of the lesion, etc. must be collected and put together. This whole process requires considerable effort, which is one of the reasons why medical data is hard to find. In this case, transfer learning aims at using various small datasets in order to achieve the same or better performance than with a single huge dataset.

6.2 Process overview

The approach consists in five different steps. Figure 6.1 gives a visual representation of the following explanations.
First of all, similar datasets are collected and processed in the same way. These datasets need to share common features. For example, DWI images (MRI sequence) and CT scans share the same color scheme and look alike (the same kind of tissue is visible, etc.).

Second, the convolutional neural network is built and split into two parts. It is still a single network but the first layers are treated as the feature extractor. As the name suggests, the role of these layers is to extract visual features characterizing

what a cancerous tumor looks like. On the other hand, the last layers are considered as decision maker. They will gather the features collected by the feature extractor and process them to make a decision on the class of the input sample.

Third, the model is trained on the target dataset and the model whose performance is the best is saved. The target dataset is the dataset which one wants to improve the performance of the model on. This first training establishes a baseline of performance and of features to learn further on thanks to the other datasets.

Fourth, the core of transfer learning process starts. For each non-target dataset, two different trainings take place:

1. The model saved at the end of the previous training is loaded. Its last layers (the decision maker) are completely reset, i.e. the weights are reset to another initialization, as if no training occured for these layers. Afterwards, the first layers (the decision maker) are frozen and the model is trained. Since the first layers have been frozen, their weights will remain the same. Only the weights of the last layers will change. This part of the transfer learning allows the model to make use of the previously learned features while adapting to the new body part by training the decision maker.

2. Once the training with frozen layers is over, the best model is loaded, the frozen layers are unfrozen, and a normal training is performed on the same dataset. Thanks to this, the model adapts the previously learned features and tries to learn new features which could be useful for the classification of the target dataset.

Fifth, the previous step is also applied to the target dataset. These final two trainings aim at keeping the interesting features from the other datasets while learning specializing the model in the target dataset again. Hopefully, the new weight initialization resulting from the transfer learning will allow to reach better metric values than before the transfer learning.

6.3 Data processing

6.3.1 PROSTATEx

As the other datasets used in this work don't contain multi-sequence exams (single-sequence MRIs and CT scans), it was required to extract single channel images from the PROSTATEx dataset. DWI images were selected because lesions are white on this sequence, as well as on CT scans. Hence, choosing DWI images ensured a consistent representation of tumors from one dataset to the other, which helps the cancer detection.

The overall process is the same as the one described in section 5.2.1, apart from the alignment and the stacking. A T2-weighted image was still used to align, crop and resize the corresponding DWI image but was not exported at the end, resulting in a single channel image (NumPy array).

Ideally, transfer learning should rely on multi-sequence MRIs only. In fact, the extra information provided by different sequences increases the performance a lot. However, publicly available datasets either don't contain the clinical significance of the samples or were purely made for segmentation purposes (where labels are also images).

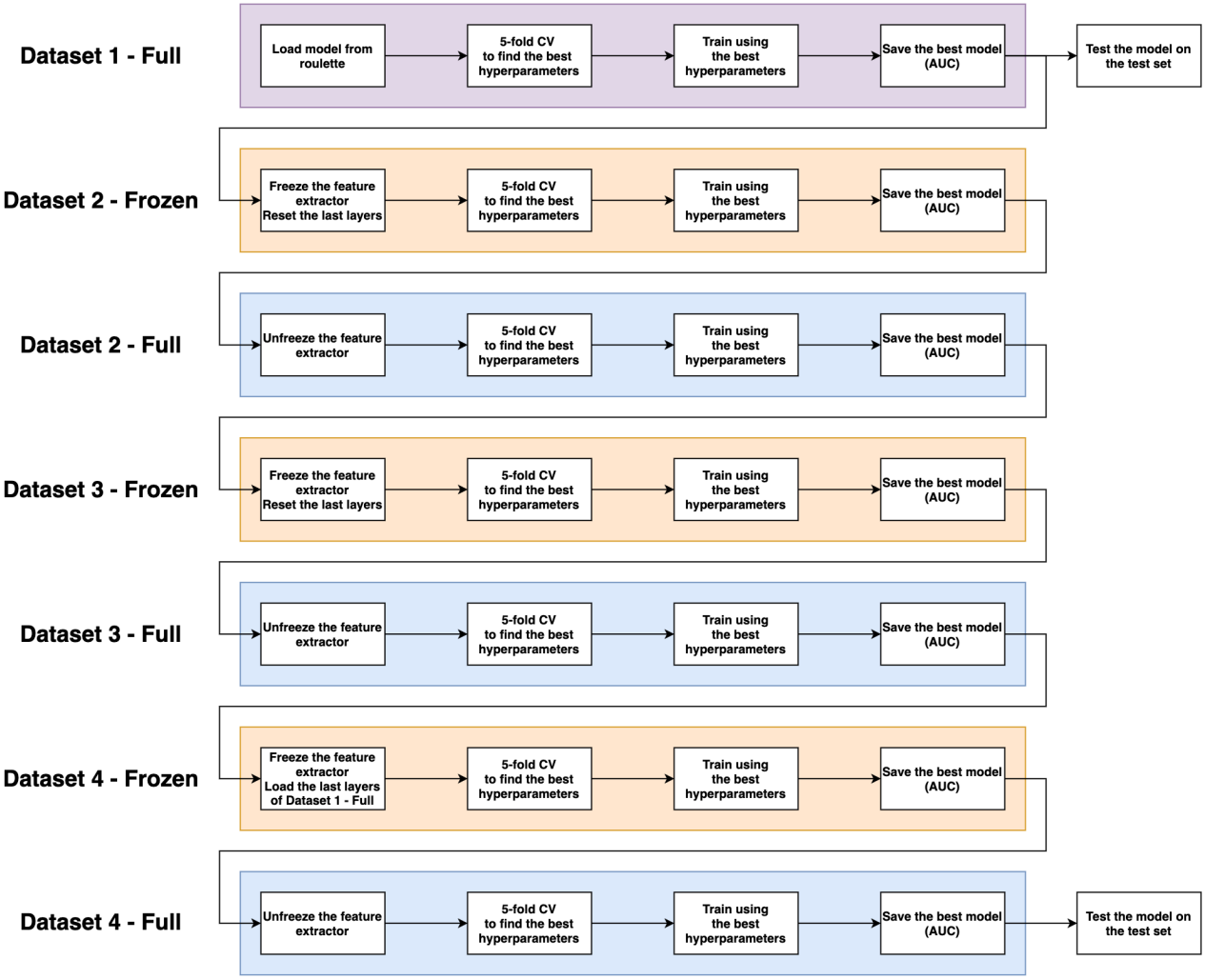

FIGURE 6.1: Transfer learning - Overview

6.3.2 LungCTChallenge

Dataset description

Lung CT Challenge is composed of two different subdatasets of CT scans: one is called "calibration set" (10 patients) and the other "test set" (60 patients). Each patient can have 1 or multiple findings. Since labels were provided for both sets and the amount of data is fairly low, they were merged and used as a global training set. For each finding, an abnormal mass is visible in the lungs, which means that the classification is going to differentiate malignant from benign tumors.

Regarding labelling, two Excel files, *TestSet_NoduleData_PublicRelease_wTruth* and *CalibrationSet_NoduleData*, contain labels for these images. In order to facilitate the handling of information in the code, two CSV files were manually created: *TestSet.csv* and *CalibrationSet.csv*. Contrary to PROSTATEx, more than two labels were used in this dataset. Both "malignant" and "Primary lung cancer" were considered as positive, whereas "benign" and "Benign nodule" were treated as negative. A third label called "Suspicious malignant nodule" appeared two times. Since the diagnosis was not clearly defined for those images, they were not included in the training data to avoid any noise.

From DICOM to augmented NumPy arrays

As PROSTATEx, the processing was split into two parts: preprocessing (DICOM to NumPy arrays) and augmentation (NumPy arrays to cropped augmented NumPy arrays.) Algorithm 3 shows the various preprocessing steps.

Algorithm 3 Lung CT Challenge preprocessing

```
1:  procedure MAIN(dataset_folder, train_CSV, test_CSV, output_folder)
2:      Create output directories: "output_folder/True", "output_folder/False"
3:
4:      csv_training ← read_CSV(train_CSV)                    ▷ CalibrationSet.csv
5:      csv_test ← read_CSV(test_CSV)                         ▷ TestSet.csv
6:      csv_concatenated ← concat(csv_training, csv_test)     ▷ Both CSV files contain
    similar information about different patients.
7:
8:      for row in csv_concatenated do
9:          patient_id ← row["Scan Number"]
10:         slice_number ← row["Nodule Center Image"]          ▷ Value in [1, ∞]
11:         finding_id ← row["Nodule Number"]
12:         clinical_significance ← row["Diagnosis"]
13:
14:         if clinical_significance == "malignant" or "Primary lung cancer" then
15:             clinical_significance ← True
16:         else if clinical_significance == "benign" or "Benign nodule" then
17:             clinical_significance ← False
18:         else if clinical_significance == "Suspicious malignant nodule" then
19:             Continue
20:
21:         for visit in patient_id's folder do
22:             for mri_type in visit do
23:                 for dicom_file in mri_type do
24:                     if slice_number == dicom_file.InstanceNumber then
25:                         slice ← normalize_dicom(dicom_file)    ▷ see section 4.2.2
26:                         Save slice in "output_folder/clinical_significance"
```

The augmentation process is close to the one applied to PROSTATEx. An advantage Lung CT Challenge has it that all images have the same resolution (512x512 pixels), making the alignment process used for PROSTATEx useless. Nevertheless, lung tumors are particular. What differentiates them from the others is their look. In fact, lung tumors usually look like spider webs with a central nodule and branches coming out of it, whereas brain and prostate tumors tend to have well-defined boundaries. Therefore, a larger patch size was chosen in order to keep the extra information provided by the branches in the peripheral tissue. Furthermore, a perfect cropping method would only crop the interesting part of the tissue and a margin around it. However, tumor size varies a lot. It can either be extremely small (a dot) or large (massive nodule with long branches). Consequently, the selected patch size makes the largest tumors fit perfectly in the cropped area, inducing more tissue for smaller tumors. Naturally, there were few outliers whose size was a lot larger than the chosen patch size. If the patch size was set accordingly, entire lungs were sometimes visible in other patients' case. For this reason, these large tumors were treated as outliers. Lastly, and contrary to PROSTATEx, the dataset was more or less balanced (39

positives and 42 negatives). Therefore, each class was augmented the same amount of times (240x).

Images were normalized with respect to equation 5.1. The mean and standard deviation were computed over the whole dataset.

6.3.3 Kaggle Brain

Dataset description

This dataset is publicly available on Kaggle. All images come from single-sequence MRIs. It is composed of two folders: one containing images with cancerous tumors and another one containing images either without tumor or with a non-cancerous tumor. Images whose quality was extremely low (resolution, picture of a screen, artefacts, etc.) were manually removed. In other cases, the image quality was good but disturbing objects such as left/right markers, arrows pointing the tumor, etc. were visible. These objects were manually removed using the object removal tool in Photoshop. File names and file formats were heterogeneous.

Ground truth creation

The particularity of this dataset is that no information apart from the clinical significance is provided. Therefore, it was necessary to create a ground truth CSV file from scratch so that the same processing methods as the other datasets could be applied on this one.

The first step was to standardize the file names and formats. The chosen convention is "$[yes|no]index.jpg$". The vast majority of the files were in the JPG file format, so the few PNG images were converted. Then, a CSV file containing the following columns was created: "PatientID", "Nodule Center Position", "fid", "Diagnosis". The "PatientID" is the filename without extension that was set during the first step (for example "*yes*99" or "*no*3"). The "Nodule Center Position" is the pixel coordinate of the lesion. Fortunately, brain tumors are clearly visible on brain MRIs, which eased the process. To find the coordinates, every single image was opened at its original size in an image visualization software. A rectangle selection tool was used to draw a rectangle from the top left corner of the image (because the top left pixel corresponds to the pixel at the coordinates $(0, 0)$ in NumPy arrays) to the center of the lesion. The pixel width and height of this rectangle gave the "i,j" coordinates of the lesion. These coordinates were reported manually. Then, the "fid" corresponds to the Finding ID. This parameter would be more pertinent if the same "PatientID" could have multiple findings, i.e. if the same patient could have multiple tumors. Since this information is made use of during the image processing of the other datasets, it was also added here. In this particular case, the "fid" was always set to "1". Finally, the "Diagnosis" was defined as "False" or "True" according to the folder which the image was in.

From JPG to NumPy arrays

Images had to be converted to single-channel NumPy arrays (grayscale) so that the values could be normalized further on. First, a mere conversion involving no content modification apart from going from three to one channel was performed. Second, images were normalized according to equation 5.1. The mean and standard deviation were computed over the whole dataset. Both classes were augmented by a common factor (200x) since both class contain a relatively close number

number of instances (133 positives and 96 negatives). Similarly to Lung CT Challenge, tumor sizes vary substantially. The same methodology was applied regarding the patch size, i.e. choosing a size which fits the majority of large tumors perfectly. Even though the images all show the head under the same perspective (same height/width ratio, visible skull, black margin around the skull), images were probably collected from different sources, which explains the heterogeneity of resolutions. As a consequence, the patch size could not be a fixed number of pixels for all images because the cropped area would be the entire head on one image and a small portion of the tumor on another. The trick consisted in finding the horizontal and vertical portions of the image occupied by the tumor. In fact, cropping an area centered on the lesion whose height was $\frac{2}{7}$ of the image and whose width was $\frac{1}{4}$ gave good results.

6.3.4 Image cropping

Multiple ways of cropping images were tested out. The differences lay in how much tissue was kept apart from the lesion itself. In some cases, more context around the lesion allows to capture details about the connections between the lesion and blood vessels for example. Moreover, the abrupt borders inducing a clear change of colors can help detect tumors. Experienced showed that more context helps when the images are single-sequence. On the other hand, cropping right around the lesion or even in the lesion gave better results for multi-sequence data, i.e. when T2-weighted, DWI and ADC images (MRIs) were simultaneously used to perform the tumor detection and classification. In the latter case, the lesion usually appears distinctively on the three sequences, and this impacts the decision a lot. On single-sequence images, there is a clear lack of information which makes it difficult to differentiate a tumor from another tissue without much context.

6.3.5 Verification

Visual checking

Regarding the Kaggle brain dataset, the data generated was paid close attention to since the ground truth CSV file was created by hand (see section 6.3.3). In fact, each generated image was taken a close look at, which allowed to spot 3 mistakes over the whole dataset. They concerned typos in the "Nodule Center Position", which created pitch black images since the cropped area was outside. Like PROSTATEx (see section 5.2.2), control images with red dots were also generated for the other datasets.

6.4 Transfer learning implementation

6.4.1 Architecture generalities

Layer freezing

After each dataset switch, the model is first trained with freezed layers. The freezed layers are those considered as feature extractor, i.e. the first layers of the model. The exact point where the model is split is arbitrary, but is usually located after the convolutions. To perform this freezing, multiple steps are required:

1. Reset the last layers. To do so, an instance of the model is created by loading the state dictionary of the previous one, before replacing the module containing the last layers with a new instance.

2. Iterate over the batch normalization layers and freeze them. Step 3 is not sufficient to freeze batch normalization layers. This can be achieved by calling module.eval() on each module containing such layers (after the model.train() call).

3. Only pass the parameters of these last layers to the optimizer.

Parallelization

Each experiment before transfer learning was performed on a single GPU. While being a lot faster than training on a CPU, the computation time was still relatively high. In order to get results from the transfer learning in a reasonable amount of time, parallelization was required. In fact, the longest transfer learning experiment consisted of $(5 \text{ folds} * 4 \text{ learning rates} * 1200 \text{ epochs} + 1200 \text{ epochs}) * 7 \text{ parts} = 176400$ epochs. On a single GPU, this experiment would have taken around two weeks to complete.

Thanks to parallelization, the cross-validation combinations could be run simultaneously on different GPUs (5 of them since 5-fold cross-validation was used). To achieve this while using Python and GPUs, we relied on a library called Ray. The function responsible for training the model needs to be defined as a Ray remote function thanks to a decorator (*@ray.remote* above the function declaration). This new function does not return the actual return values, but object IDs pointing to placeholders for the future results, which allows to continue the execution and run other parallel functions. Outside of the cross-validation loop, a call to $ray.get()$ on the object IDs allows to wait for the simultaneous executions to complete and to collect the different results.

6.4.2 Script options

A single Python script allows to run the entire transfer learning pipeline. Multiple command line arguments are required in order to load the right datasets and the right model, to define the hyperparameters, etc. Table 6.1 gives a exhaustive list of the script options. The particularity lies in the learning rate, number of epochs, batch size and dropout options. In fact, the pipeline consists in seven different phases of training: dataset 1 (full model), dataset 2 (frozen model), dataset 2 (full), dataset 3 (frozen), dataset 3 (full), dataset 4 (frozen) and dataset 4 (full). In such a case, each phase will probably use different hyperparameters. Therefore, these command line options need to specify seven different parameters. For example, a valid value for the "–lr" option could be "1e-8,1e-5,1e-6,1e-6,1e-5,1e-7,1e-8".

6.4.3 Visualization of the impact of the various datasets on the target task

On paper, each step of the transfer learning is supposed to make the model learn new features that are useful for the first dataset (the target dataset). As described in section 6.2, the model can be seen as a feature extractor and a decision maker. Hence, to see how much each step was actually helping the process, the decision maker resulting from the first step of the transfer learning, i.e. the training on the first/target dataset, was saved. As this decision maker is saved after the training on the target dataset, it is specialized in the classification of the latter. In other words, if the target

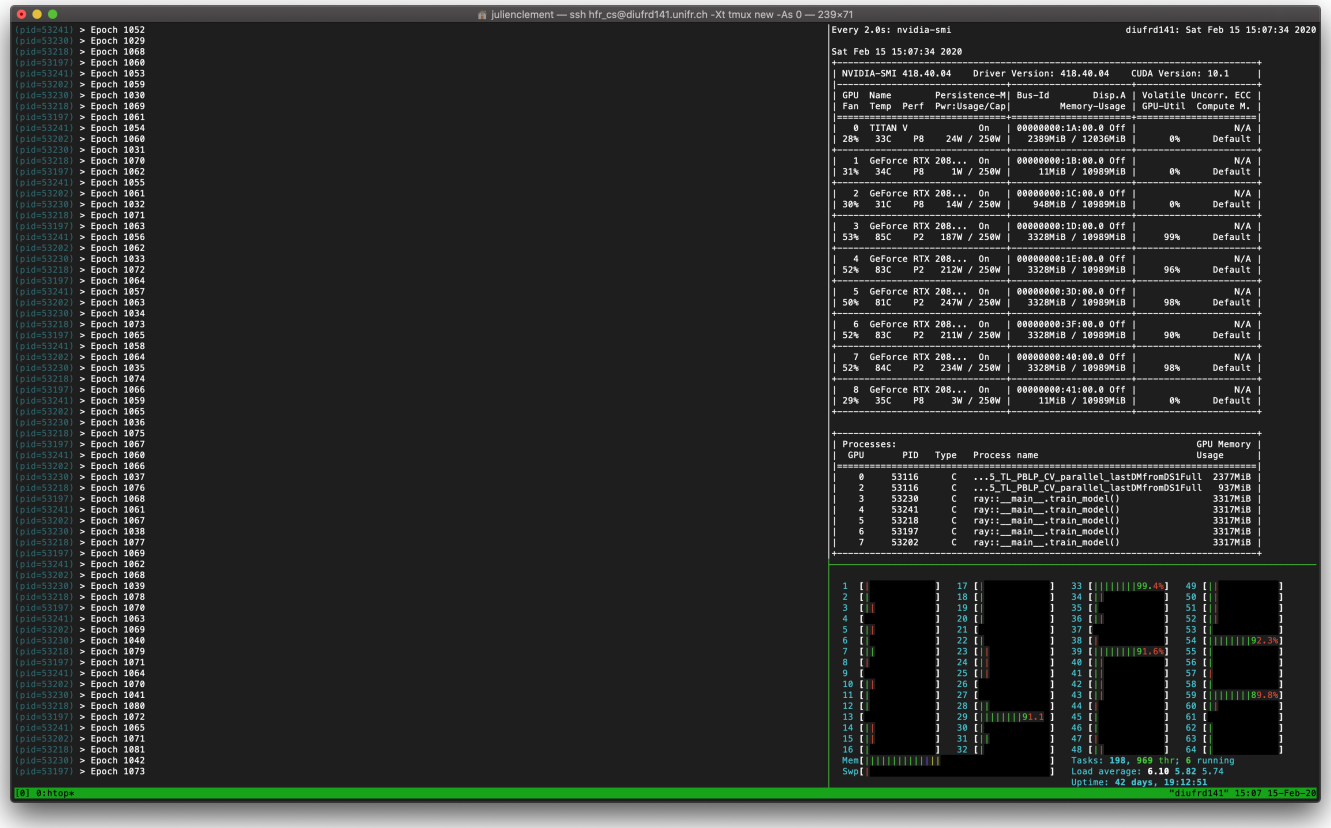

FIGURE 6.2: Transfer learning - Parallelization

dataset is a prostate dataset, it will classify prostate tumors with a relatively high precision.

Then, after each epoch of every other transfer learning step, the saved decision maker was attached to the current feature extractor. This temporary model was then tested on the target training and validation sets, which resulted in different training and validation metrics.

At the end of the transfer learning pipeline, a graph was generated for each metric. These curves show the evolution of the latter across the entire process, which allows to determine the impact of each non-target dataset on the target task. If a metric is increasing, the features learned thanks to the corresponding dataset are useful for the target one. Figure 6.3 shows an example of this graph for the AUC.
As expected, the curve is flat during the frozen parts. Since the same decision maker is attached to frozen layers (which remain the same by definition), the values of the metrics don't change at all.

6.4.4 Experimental setup

6.5 Results

6.5.1 Independent training

To get reference values and evaluate the impact of transfer learning on the model performance, a set of trainings on each dataset was performed independently. These

Command	Description	Required	Type
–training-set1	Path to training set 1	True	String
–validation-set1	Path to validation set 1	True	String
–test-set1	Path to test set 1	True	String
–training-set2	Path to training set 2	True	String
–validation-set2	Path to validation set 2	True	String
–training-set3	Path to training set 3	True	String
–validation-set3	Path to validation set 3	True	String
–batchsize	Number of samples per batch	True	Comma-separated ints
–nbepochs	Number of epochs the training phase lasts	True	Comma-separated ints
–lr	Learning rate used in the optimizer	True	Comma-separated floats
–lossfunction	Loss function name [CrossEntropyLoss, L1Loss, MSELoss]	False Default: 'CrossEntropyLoss'	String
–cuda-device	GPU name to run the experiment	False Default: 'cuda'	String
–modeltoload	Pretrained model name If given, load it, otherwise randomly initialize it	False Default: ""	String
–dropout	Dropout probability	True	Comma-separated floats
–optimized-metric	Metric to optimize The best model during the training will be saved according to it ['auc', 'accuracy', 'precision', 'recall', 'f1score', 'specificity']	False Default: 'auc'	String
–outputdirectory	Root of the output directory used by Tensorboard to save the models	True	String
–cv	Whether to use cross-validation or not ['True', 'False']	True	String

TABLE 6.1: Complete list of script options

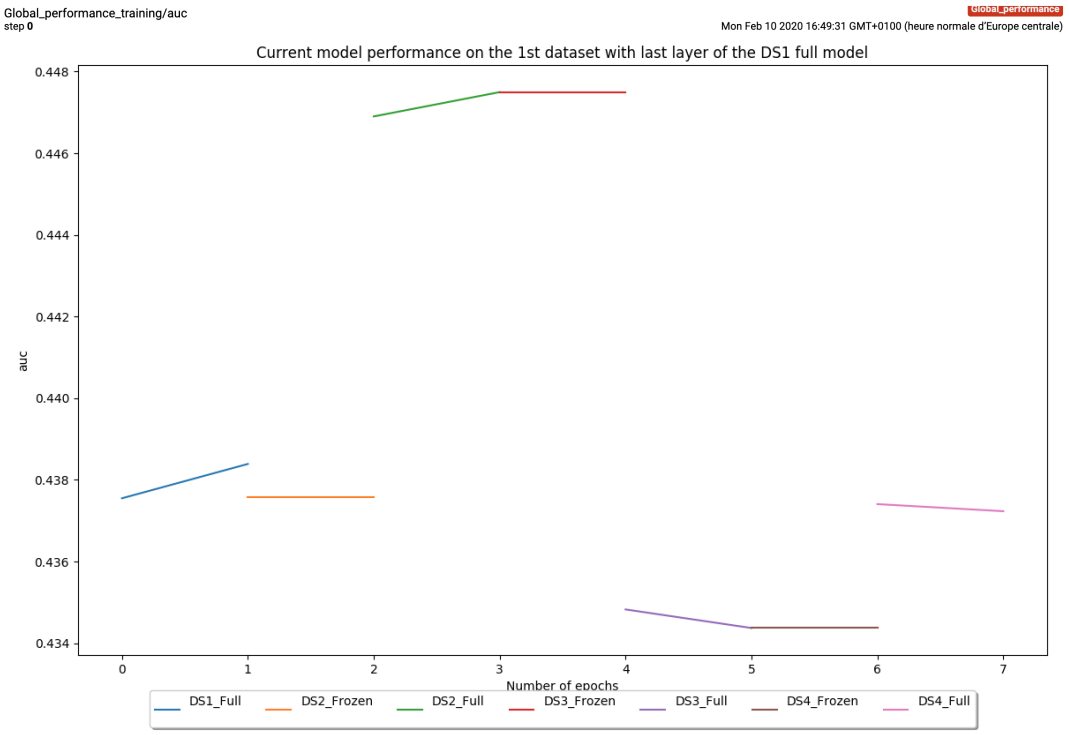

FIGURE 6.3: Please replace this figure with a real example as soon as the results are available!

experiments involved multiple hyperparameter combinations. The best results were used as benchmark values.

PROSTATEx

Kaggle brain

Lung CT Challenge

6.5.2 Transfer learning

6.6 Discussion

Chapter 7

Conclusion

7.1 Conclusion

In this work we...

7.2 Future Work

The continuation of this work includes...

Bibliography

[1] World Health Organization. *Fact Sheet about Cancer*. en. URL: https://www.who.int/news-room/fact-sheets/detail/cancer (visited on 02/06/2020).

[2] Jun Gao et al. "Convolutional neural networks for computer-aided detection or diagnosis in medical image analysis: An overview". en. In: *Mathematical Biosciences and Engineering* 16.6 (2019), pp. 6536–6561. ISSN: 1551-0018. DOI: 10.3934/mbe.2019326. URL: http://www.aimspress.com/article/10.3934/mbe.2019326 (visited on 02/04/2020).

[3] Yang Song et al. "Computer-aided diagnosis of prostate cancer using a deep convolutional neural network from multiparametric MRI: PCa Classification Using CNN From mp-MRI". en. In: *Journal of Magnetic Resonance Imaging* 48.6 (Dec. 2018), pp. 1570–1577. ISSN: 10531807. DOI: 10.1002/jmri.26047. URL: http://doi.wiley.com/10.1002/jmri.26047 (visited on 01/29/2020).

[4] Saifeng Liu et al. "Prostate cancer diagnosis using deep learning with 3D multiparametric MRI". en. In: ed. by Samuel G. Armato and Nicholas A. Petrick. Orlando, Florida, United States, Mar. 2017, p. 1013428. DOI: 10.1117/12.2277121. URL: http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.2277121 (visited on 12/19/2019).

[5] Nathan Lay et al. "A Decomposable Model for the Detection of Prostate Cancer in Multi-parametric MRI". en. In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2018*. Ed. by Alejandro F. Frangi et al. Vol. 11071. Cham: Springer International Publishing, 2018, pp. 930–939. ISBN: 978-3-030-00933-5 978-3-030-00934-2. DOI: 10.1007/978-3-030-00934-2_103. URL: http://link.springer.com/10.1007/978-3-030-00934-2_103 (visited on 01/29/2020).

[6] Samuel G. Armato et al. "PROSTATEx Challenges for computerized classification of prostate lesions from multiparametric magnetic resonance images". In: *Journal of Medical Imaging* 5.04 (Nov. 2018), p. 1. ISSN: 2329-4302. DOI: 10.1117/1.JMI.5.4.044501. URL: https://www.spiedigitallibrary.org/journals/journal-of-medical-imaging/volume-5/issue-04/044501/PROSTATEx-Challenges-for-computerized-classification-of-prostate-lesions-from-multiparametric/10.1117/1.JMI.5.4.044501.full (visited on 02/05/2020).

[7] Emine Cengil and Ahmet Cinar. "A Deep Learning Based Approach to Lung Cancer Identification". en. In: *2018 International Conference on Artificial Intelligence and Data Processing (IDAP)*. Malatya, Turkey: IEEE, Sept. 2018, pp. 1–5. ISBN: 978-1-5386-6878-8. DOI: 10.1109/IDAP.2018.8620723. URL: https://ieeexplore.ieee.org/document/8620723/ (visited on 01/29/2020).

[8] Samuel G. Armato et al. "LUNGx Challenge for computerized lung nodule classification". en. In: *Journal of Medical Imaging* 3.4 (Dec. 2016). LungCTChallenge dataset, p. 044506. ISSN: 2329-4302. DOI: 10.1117/1.JMI.3.4.044506.

URL: http://medicalimaging.spiedigitallibrary.org/article.aspx?doi=10.1117/1.JMI.3.4.044506 (visited on 01/29/2020).

[9] *Brain Tumor - Statistics*. en. June 2012. URL: https://www.cancer.net/cancer-types/brain-tumor/statistics (visited on 02/05/2020).

[10] *Brain MRI Images for Brain Tumor Detection*. en. URL: https://kaggle.com/navoneel/brain-mri-images-for-brain-tumor-detection (visited on 02/19/2020).

[11] Priyansh Saxena et al. "Predictive modeling of brain tumor: A Deep learning approach". en. In: *arXiv:1911.02265 [cs, eess]* (Dec. 2019). arXiv: 1911.02265. URL: http://arxiv.org/abs/1911.02265 (visited on 01/29/2020).

[12] Mohamed Ali Habib. *Brain Tumor Detection Using Convolutional Neural Networks*. en. URL: https://medium.com/@mohamedalihabib7/brain-tumor-detection-using-convolutional-neural-networks-30ccef6612b0 (visited on 02/06/2020).

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *15_DeepLearningBook.pdf*. MIT Press, 2016. URL: http://www.deeplearningbook.org.

[14] Haohan Wang and Bhiksha Raj. "On the Origin of Deep Learning". en. In: (), p. 71.

[15] Andrew Ng. *Deep learning notation*. URL: https://drive.google.com/open?id=1jPHsGU4G4GMnpHwJckSRDWud3z7osW4B.

[16] Michael Nielsen. "Neural Networks and Deep Learning". en. In: (), p. 224.

[17] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". en. In: *Neural Networks* 2.5 (Jan. 1989), pp. 359–366. ISSN: 08936080. DOI: 10.1016/0893-6080(89)90020-8. URL: https://linkinghub.elsevier.com/retrieve/pii/0893608089900208 (visited on 01/29/2020).

[18] Thomas Epelbaum. "Deep learning: Technical introduction". PhD thesis. Sept. 2017.

[19] Ke-Lin Du and M. N. S. Swamy. *Multilayer Perceptrons: Architecture and Error Backpropagation*. en. London: Springer London, 2014. ISBN: 978-1-4471-5570-6 978-1-4471-5571-3. DOI: 10.1007/978-1-4471-5571-3_4. URL: http://link.springer.com/10.1007/978-1-4471-5571-3_4 (visited on 01/29/2020).

[20] Andrew Ng. "Sparse autoencode". In: *Stanford University*. CS294A Lecture notes 72 (2011), pp. 1–19.

[21] Salma Ghoneim. *Accuracy, Recall, Precision, F-Score & Specificity, which to optimize on?* en. Apr. 2019. URL: https://towardsdatascience.com/accuracy-recall-precision-f-score-specificity-which-to-optimize-on-867d3f11124 (visited on 01/29/2020).

[22] Sarang Narkhede. *Understanding AUC - ROC Curve*. en. June 2018. URL: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5 (visited on 01/29/2020).

[23] James Dellinger. *Weight Initialization in Neural Networks: A Journey From the Basics to Kaiming*. en. Apr. 2019. URL: https://towardsdatascience.com/weight-initialization-in-neural-networks-a-journey-from-the-basics-to-kaiming-954fb9b47c79 (visited on 01/29/2020).

[24] Author keitakurita. *Learning Rate Tuning in Deep Learning: A Practical Guide*. en-US. June 2018. URL: https://mlexplained.com/2018/01/29/learning-rate-tuning-in-deep-learning-a-practical-guide/ (visited on 01/29/2020).

[25] itdxer. *What is batch size in neural network?* URL: https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network (visited on 01/29/2020).

[26] Jason Brownlee. *A Gentle Introduction to Transfer Learning for Deep Learning*. en-US. Dec. 2017. URL: https://machinelearningmastery.com/transfer-learning-for-deep-learning/ (visited on 01/29/2020).

[27] Douglas Hanahan and Robert A Weinberg. "The Hallmarks of Cancer". en. In: *Cell* 100.1 (Jan. 2000), pp. 57–70. ISSN: 00928674. DOI: 10.1016/S0092-8674(00)81683-9. URL: https://linkinghub.elsevier.com/retrieve/pii/S0092867400816839 (visited on 01/29/2020).

[28] Douglas Hanahan and Robert A. Weinberg. "Hallmarks of Cancer: The Next Generation". en. In: *Cell* 144.5 (Mar. 2011), pp. 646–674. ISSN: 00928674. DOI: 10.1016/j.cell.2011.02.013. URL: https://linkinghub.elsevier.com/retrieve/pii/S0092867411001279 (visited on 01/29/2020).

[29] Stephen B. Edge and American Joint Committee on Cancer, eds. *AJCC cancer staging manual*. en. 7th ed. OCLC: ocn316431417. New York: Springer, 2010. ISBN: 978-0-387-88440-0 978-0-387-88442-4.

[30] *Stages of Cancer*. en. May 2010. URL: https://www.cancer.net/navigating-cancer-care/diagnosing-cancer/stages-cancer (visited on 01/29/2020).

[31] *TensorBoard*. en. URL: https://www.tensorflow.org/tensorboard?hl=fr (visited on 01/31/2020).

[32] *Check gradient flow in network - PyTorch Forums*. URL: https://discuss.pytorch.org/t/check-gradient-flow-in-network/15063/10 (visited on 02/20/2020).

[33] Jason Brownlee. *How to use Learning Curves to Diagnose Machine Learning Model Performance*. en-US. Feb. 2019. URL: https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/ (visited on 02/02/2020).