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

Your introduction chapter here.

Chapter 2

Literature review

Song et al. [19] proposed a DCNN method to detect prostate cancer 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 which were split into a training, validation and test set. 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 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 these processing techniques were 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. 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 with the highest b-value only, 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.

Saifeng et al. [18] created another architecture called XMasNet which was tested on the actual PROSTATEx challenge, achvieving the second best performance with an AUC of 0.84. The AUC on the validation set reached 0.92. Their data processing approach stacked different combination of the available sequences as the three channels: 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 the reach the best performance possible.

Mehrtash et al. [13] used a different approach. First of all, the input was feeded 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 are merged into a common decision maker. Furthermore, 3-dimensional convolutions instead 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. Finally, this model achieved 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.

Chapter 3

Deep learning

This chapter provides the theoretical foundation in deep learning for the understanding of the rest of the work. It starts with the historical story of the deep learning before describing what is a neural network. 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 with another devoted to a special type of neural networks, the "convolutional neural networks". They are used in many computer vision problems due to their great performance for these tasks. Finally, the concept of "transfer learning" is discussed since 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, since 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 be 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. [7], 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 are 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: Milestones - Wang et al. [21]

3.1.2 What is a neural network?

From a descriptive point of view, neural networks can simply be seen as a 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 to an input x 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

In machine learning, two different kinds of tasks exist. The first one is "supervised learning". It includes learning algorithms whose training samples are associated to 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 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 $a^{[L]}$ 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" [17]. Today, they are called "artificial neurons" or simply "neurons".
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:

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

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 linear functions only 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 "multilayer feedforward networks are capable of approximating any measurable function to any desired degree of accuracy, in a very specific and satisfying sense" [10]. According to Thomas Epelbaum [5], the most commonly used activation functions are:

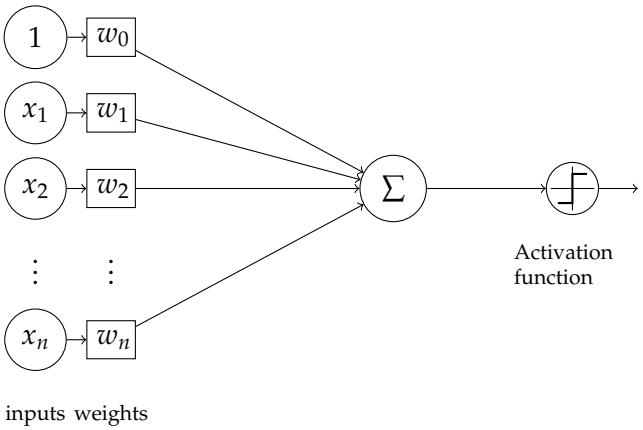

FIGURE 3.2: The perceptron model

Sigmoid function
The sigmoid function is defined as:

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

Its derivative is:

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

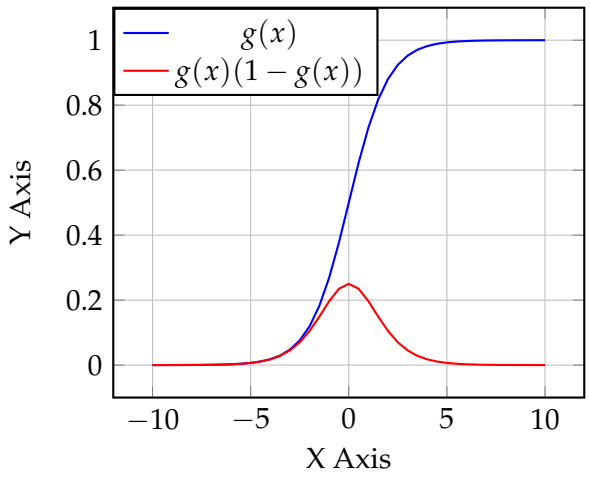

FIGURE 3.3: The sigmoid function and its derivative

Tanh function

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

Its derivative is:

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

ReLU function

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

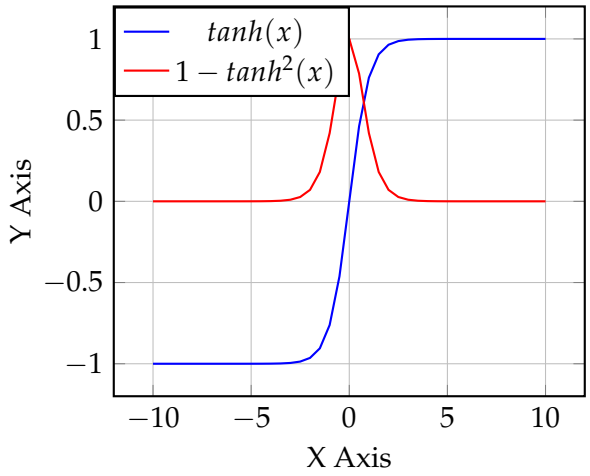

FIGURE 3.4: The tanh function and its derivative

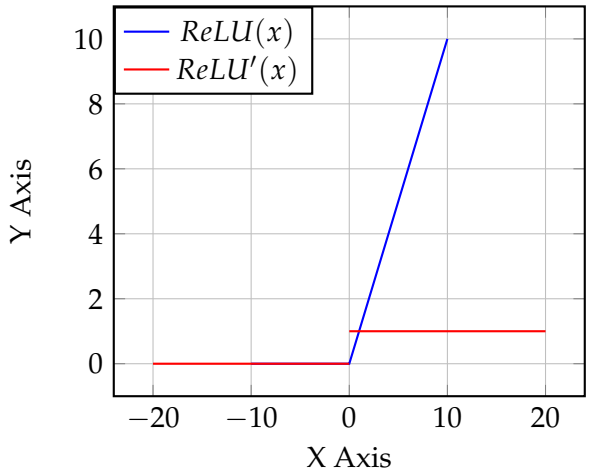

FIGURE 3.5: The ReLU function and its derivative (undefined when $x = 0$)

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:

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

3.2.4 Multilayer perceptrons

A multilayer perceptron is a type of artifical neural networks. Du et al. [3] 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:

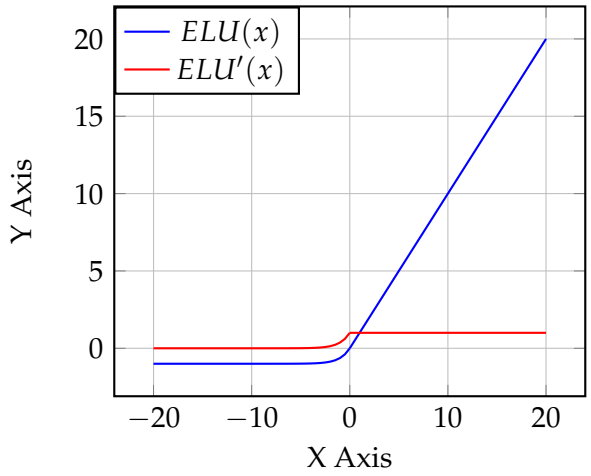

FIGURE 3.6: The ELU function and its derivative

- 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 feeded 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^l_{ij} corresponds to the weights associated with the connection between perceptron j in layer l and perceptron i in layer $l+1$. By analogy, $b^{[l]}_i$ 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 beeing 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 $b^{[1]} = 2$, $b^{[2]} = 1$.

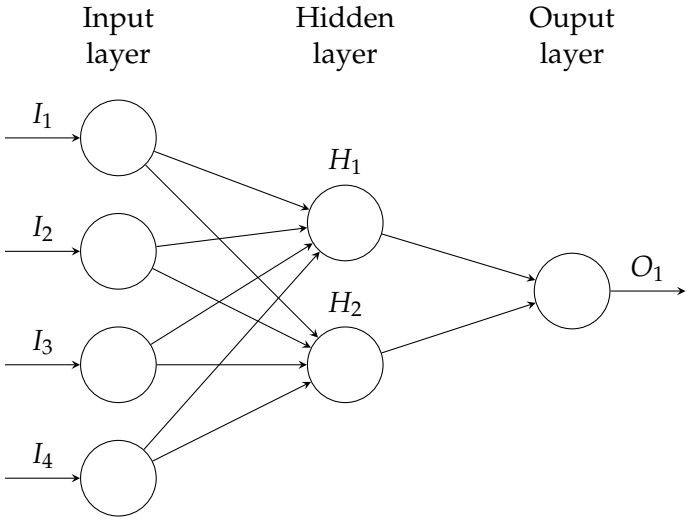

FIGURE 3.7: Multilayer perceptrons

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^l_{jk} 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 $a^{[l-1]}_j$ before adding the bias $b^{[l]}$. The result is given as input to the activation function.

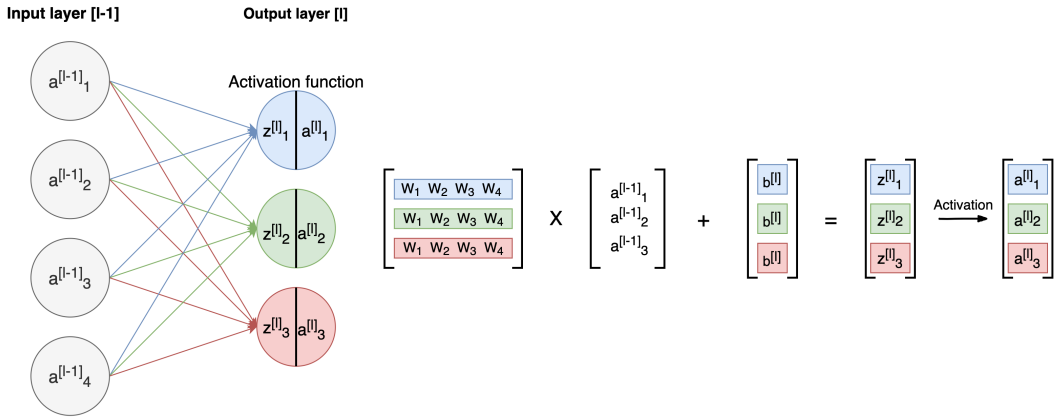

FIGURE 3.8: Forward propagation

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" [5]. 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" [16].

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". Inthe 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 [6]:

- 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" [14].

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 the dataset that 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.

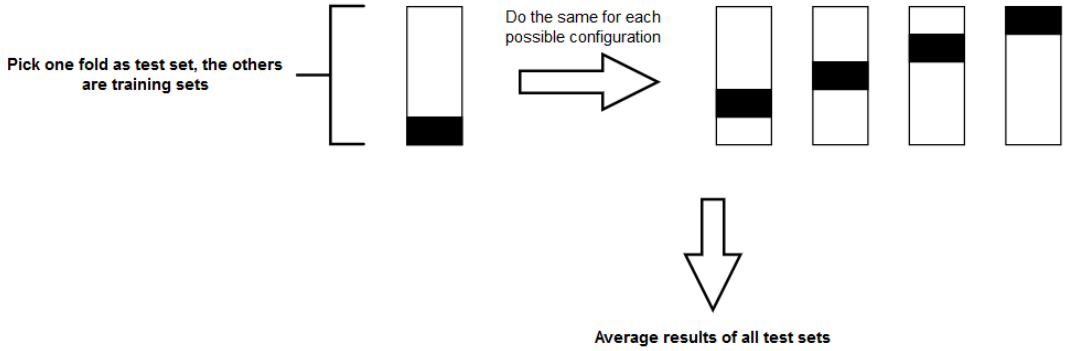

FIGURE 3.9: 5-folds cross-validation

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" [2].
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 a acceptable level of generalization).
Regarding the learning rate α, its value has to be neither too large or too small. A too large learning rate is recognizable by analyzing the training loss curve: if "the loss is exploding or fluctuating" or if "it has stopped improving and it is wandering around a suboptimal local optima" [12], the learning rate is too high and should be decreased. On the contrary, if "the learning is very slow and the loss is decreasing consistently" or if "the model is overfitting" [12], 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" [12]. 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" [11]. Furthermore, "networks train faster because the weights update is done after each propagation" [11]. Nevertheless, "the smaller the batch the less accurate the estimate of the gradient will be" [11]. 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 "early stopping", 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". A filter can be seen as a an 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.

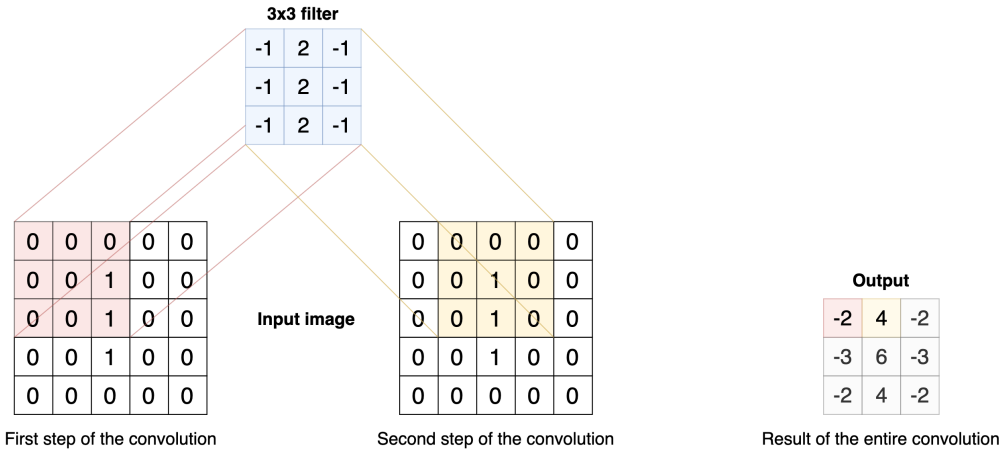

FIGURE 3.10: Convolutions - Basic convolution in a CNN

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 "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 $5x5$. 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.

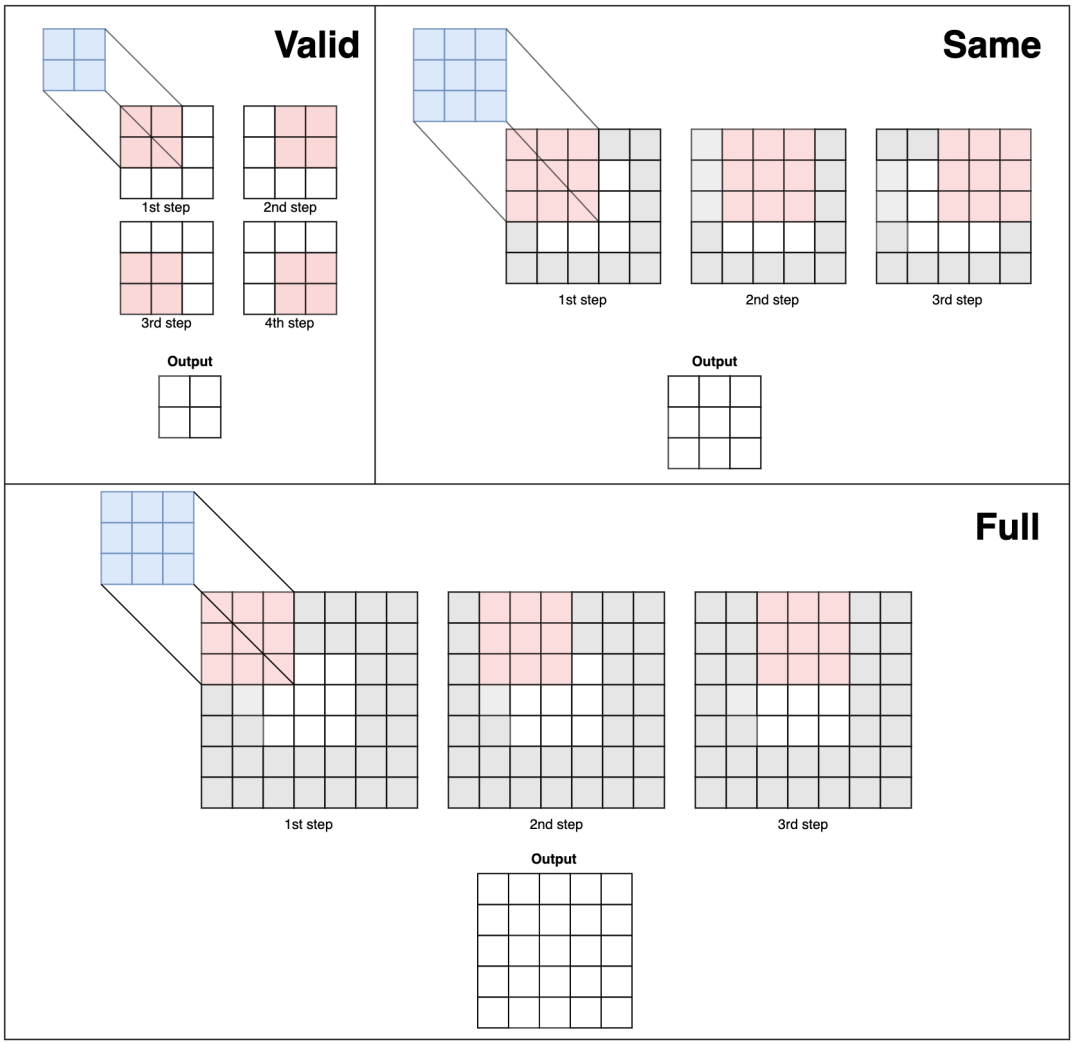

FIGURE 3.11: Convolutions - Different padding methods

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" [1]. 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 [1]:

- 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" [1].

Chapter 4

Medical information

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" [9] and "The Hallmarks of Cancer: The Next Generation" [8], 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 the only responsible for their 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 [4]. 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 [20]. The earlier they are detected, the higher the chances of recovering are.

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 outputed 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 7.1 shows a lung CT scan.

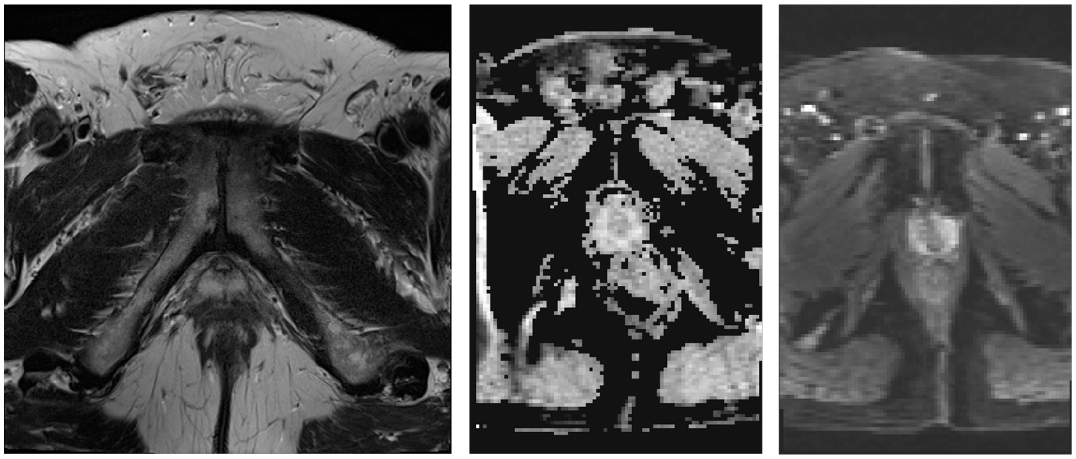

FIGURE 4.1: MRI - PROSTATEx - From left to right: T2-weighted, ADC and DWI

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 to use 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: https://brainder.org/2012/09/23/the-nifti-file-format/

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 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 relies on the article "Computer-Aided Diagnosis of Prostate Cancer Using a Deep Convolutional Neural Network from Multiparametric MRI" from Song et al.[19], 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, which overcome the lack of available data in the medical field (see chapter 6).
Song et al. [19] 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 available dataset for prostate cancer classification. The two different output classes of the latter are benign lesion (class 0) and malignant lesion (class 1). This paper explains all the steps to follow with the goal of building a computer-aided-diagnosis (CAD) system for prostate cancer detection. Moreover, it also provides results about the algorithm performance, which can be used as a benchmark to compare with the results of the reproduction of the experiment. However, since the article gives no information about the results their model got on the official PROSTATEx challenge test set, the reproduction of the experiment will fill that gap.
This chapter is built top-down: first a superficial overview of the entire process is established in order to understand the purpose of the experiment as a whole. Then, each part or theoretical notion of the experiment is deeply described. The structure of the dataset is detailed, all steps of its processing are explained and the techniques used as verification of the proper functioning of the algorithm are presented. After the processing of the data, the training phase is deepened and all hyperparameters, options and implementation choices are given to ensure the reproductibility of the experiment. This part is followed by the presentation of the raw results, which is itself followed by their analysis in the "discussion" section.

5.1 Process overview

Schematically, the entire experiment process can be represented as shown in figure 5.1. The PROSTATEx dataset is composed of samples whose clinical significance is provided (labeled data) and samples without this information (unlabeled data). The latter are used for the challenge, which consists in predicting the probability of patients lesions to be malignant (class 1) or benign (class 0). In both cases, the data is processed before been used (see 5.2 for details). After the processing, the labeled data are split into training, validation and test sets using respectively 80%, 10% and 10% of all available data for each set. The training set is then given in batches as input to the neural network, which update its weight consequently. At the end of

each epoch, the current model is tested on the validation set (that the model has never seen previously) and the metrics are plotted on Tensorboard (see 5.3.3). If the current model has a bigger AUC and a bigger accuracy than the previous best model, it becomes the best model and it is saved. When the training phase reaches the defined number of epochs, the model is then tested on the test set. This same model is also used to take part in the PROSTATEx challenge to output predictions into a .csv file for each patient and findings.

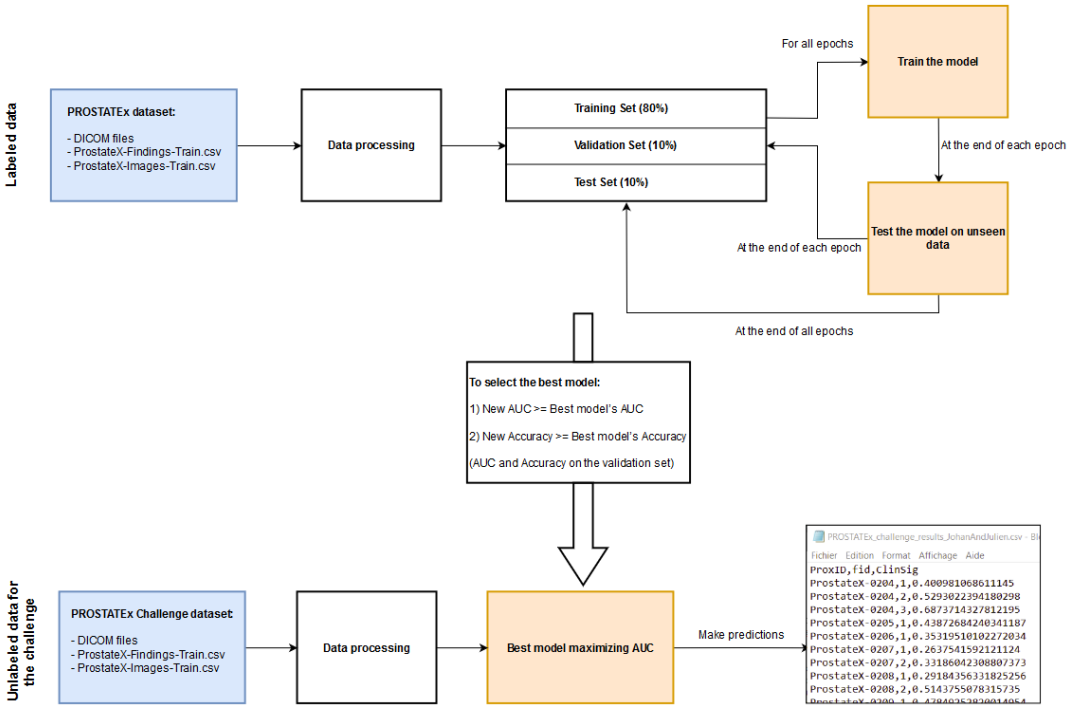

FIGURE 5.1: Paper reproduction experiment

5.2 PROSTATEx: Data processing

5.2.1 Dataset description

5.2.2 From DICOM to NumPy arrays

5.2.3 From NumPy arrays to augmented stacked images

5.2.4 From NumPy arrays to augmented non-stacked images

5.2.5 Data processing verification

Cropping verification using red dots

Alignment visualization

5.3 Training the neural network

5.3.1 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 of 14 millions images belonging to 1000 classes. The figure 5.2 illustrates its structure and its corresponding implementation in Python with the PyTorch Framework.
It is first composed by 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 activation function. The same principle is used for 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 or 1x1, 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 layer since the task is simpler than the original one, it uses exponential linear units instead of rectified linear units as activation functions and adds dropout and batch normalization layers.

5.3.2 Criterion to save the best model

5.3.3 Tensorboard

During the experiment the following metrics are registered: 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 across all epochs and plotted on the same figure thanks to Tensorboard at the end of the process. Subsequently, this is also the case for the metrics measured on the test set.
"Tensorboard provides the visualization and tooling needed for machine learning experimentation:

- Tracking and visualizing metrics.
- 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"[**39**].

In this experiment, Tensorboard is mainly used to plot the Matplotlib figures of the model performance. In addition to this, written reports regarding which model was the best and the results it reached are also added on the dashboard.

5.3.4 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. In order to achieve this goal, the Python module "argparse" is extremely useful. The latter automatically generates help/usage messages and displays errors when the argument given by the user are invalid. The table shows all options accepted by the script.

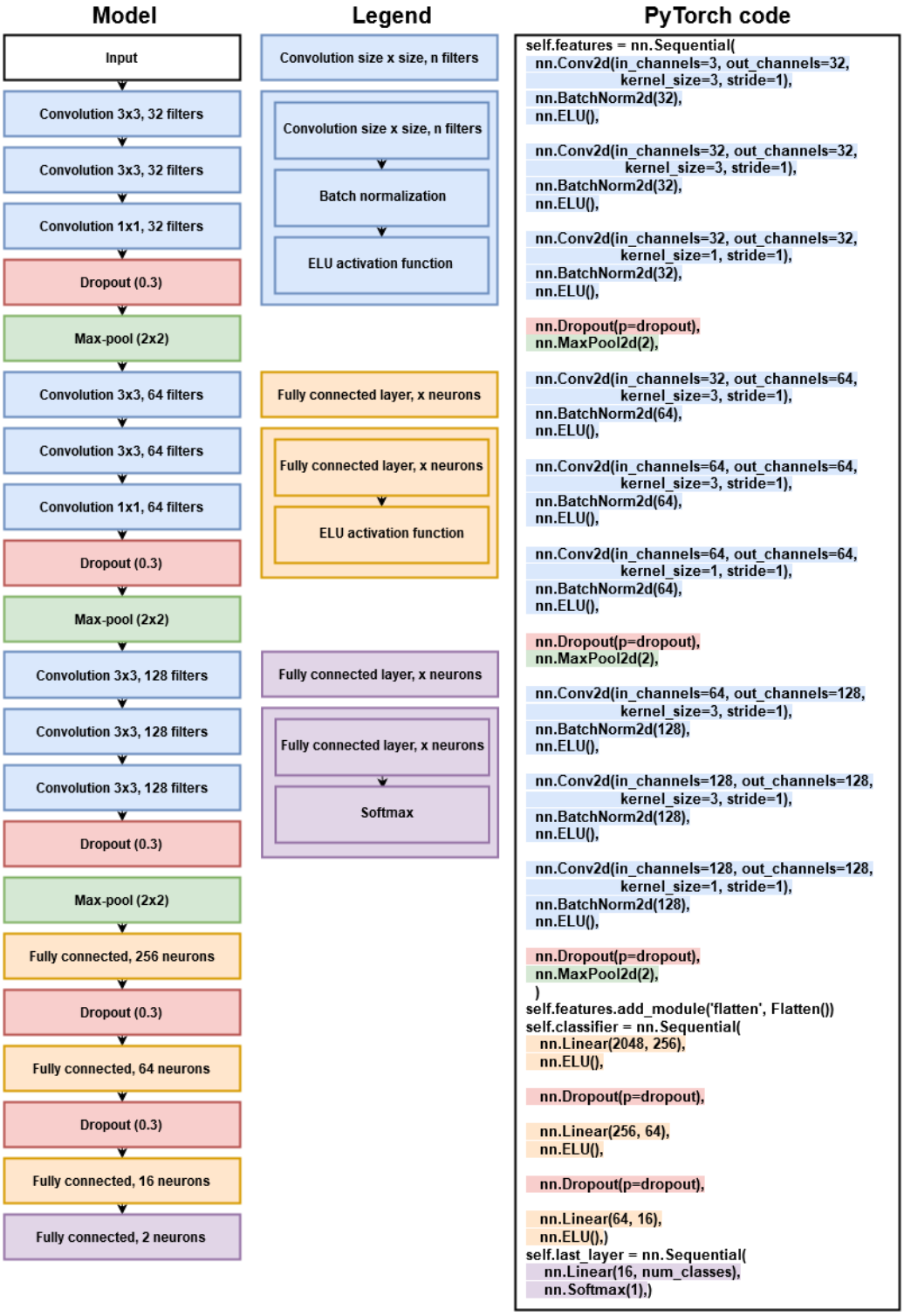

FIGURE 5.2: Model architecture with the corresponding PyTorch code

5.3.5 Experimental setup

The best performing model was implemented with PyTorch using the ADAM optimizer to update its weights, a learning rate of 1e-8 and the cross-entropy as loss function. The data was organized in batches of 128 samples and the experiment ran for 1300 epochs. The best model was selected with respect to the area under curve. Regarding the weights initialization, the best model from the roulette that

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

maximized the F1-score was chosen as initial model.
The model was trained on a Nvidia GeForce GTX Titan X graphic card during approximately 15 hours.

5.3.6 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. Concretely, in this case, large gradients are computed, which results in 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 not been face to such problems. To achieve this goal, a visualization of all gradients of the network is implemented. 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 5.3 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.

5.4 Results

5.5 Discussion

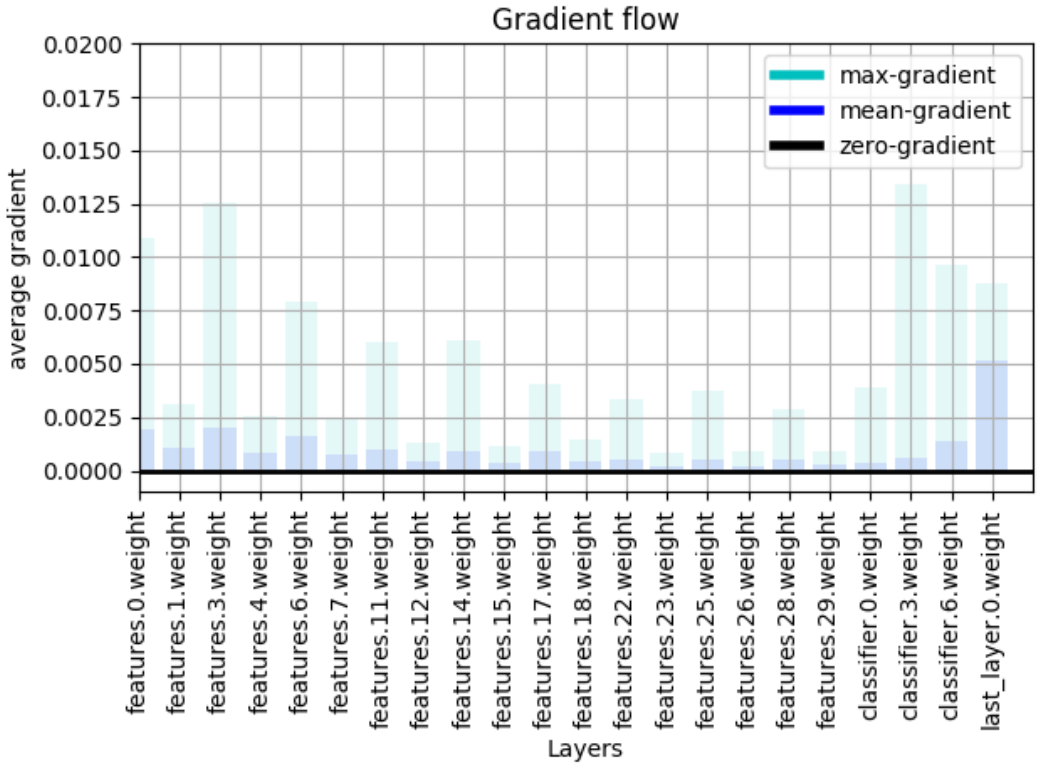

FIGURE 5.3: Gradient flow at epoch 0, batch 6 of the experiment

Chapter 6

Improving performance using transfer learning

What we do: improve cancer detection/classification on medical imaging How we do it: transfer learning

6.1 Goal

6.2 Process overview

6.3 Data processing

6.3.1 PROSTATEx

From DICOM to augmented DWI NumPy arrays

Keep only the DWI because white lesion, crop the lesion etc

6.3.2 LungCTChallenge

Dataset description

From DICOM to augmented NumPy arrays

6.3.3 Kaggle Brain

Dataset description

Ground truth creation

From PNG to NumPy arrays

Manually create the csv file from scratch, crop the lesion etc

6.3.4 Verification

Visual checking

6.4 Transfer learning implementation

6.4.1 Architecture generalities

6.4.2 Script options

6.4.3 Visualization of the usefulness of the features learned on each dataset

6.4.4 Experimental setup

6.5 Results

6.6 Discussion

Chapter 7

Data processing

7.1 Processing flow

Schema Datasets -> xxx_preprocessing.py -> create_dataset.py -> Train/Val

7.1.1 PROSTATEx: From DICOM to NumPy arrays

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, from 1 to ∞), "ClinSig" (clinical significance, TRUE or FALSE), "DCMSerNumber" (digit before the dash in the folder name containing DICOM files) and "ijk" (position of the lesion: slice number k at coordinates (i, j), $i, j, k \in [0, \infty]$). Both CSV files are complementary to each other. Algorithm 2 describes the steps involved in converting PROSTATEx's DICOM files to NumPy arrays.

7.1.2 Lung CT Challenge - From DICOM to NumPy arrays

Lung CT Challenge is composed of two different subdatasets: one is considered as a calibration set (10 patients) and the other as a test set (60 patients). Since labels were provided for both sets and the amount of data is fairly low, they were merged and used as a training set.

Regarding labelling, two Excel files, *TestSet_NoduleData_PublicRelease_wTruth* and *CalibrationSet_NoduleData*, contain labels for these images. In order to facilitate label managing, two CSV files were manually created: *TestSet.csv* and *CalibrationSet.csv*. Contrary to PROSTATEx, more than two labels were used for this dataset. Both "malignant" and "Primary lung cancer" were considered as positive, whereas "benign" and "Benign nodule" 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 treated and included in the training data in order to avoid any noise. Algorithm 3 shows the various processing steps.

7.1.3 NumPy arrays to PNG files

Exporting to NumPy arrays instead of image files directly has multiple advantages. First of all, converting medical files to NumPy arrays or PNG files takes more time than converting NumPy arrays to image files. Reason for that is that the former requires a lot of operation as well as pixel normalization, whereas the latter is a mere conversion of one format to the other. This makes data more reusable. Then,

Algorithm 2 PROSTATEx preprocessing

```
procedure MAIN(dataset_folder, findings_CSV, slices_CSV, output_folder)
    Create output directories: "output_folder/True", "output_folder/False"

    findings ← read_CSV(findings_CSV)          ▷ ProstateX-Findings-Train.csv
    slices ← read_CSV(slices_CSV)              ▷ ProstateX-Images-Train.csv
    meta ← merge(findings, slices)  ▷ Both CSV files are complementary to each other.

    for row in meta do
        patient_id ← row["ProxID"]
        finding_id ← row["fid"]
        mri_type_number ← row["DCMSerNumber"]
        clinical_significance ← row["ClinSig"]
        img_i, img_j, img_k ← row["ijk"]
        slice_number ← img_k + 1      ▷ CSV indexing in [0, ∞], DICOM in [1, ∞]

        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 ← normalize_dicom(dicom_file)  ▷ see section 4.2.2
                            Save slice in "output_folder/clinical_significance"
```

it facilitates the debugging process by separating the conversion from medical files to PNG files into two different steps. Finally, the same processing script can be used to convert NumPy arrays from any dataset to PNG images, which eases operations such as generating different training-validation splits, augmenting data, etc.

Algorithm 4 describes how data was split and organized in order for PyTorch to make use of it. To do so, they were split by class (True $\equiv 1$, False $\equiv 0$) and role (training or validation). Furthermore, data are split by patients and not by slices. In fact, multiple slices are usually assigned to each patient. Instead of considering each slice separately, which allows to divide a single patient's slices into the training and validation folders, they were treated as a whole. To sum up, an 80-20 split between the training and validation data (split argument set to 0.8) will use 80% of the patients as training data and 20% of the patients as validation data, regardless of the number of slices.

7.1.4 Data augmentation

The amount of publicly available data is relatively limited. However, deep learning models require a lot of training data to be able to learn and generalize well. Therefore, augmenting data allows to create more training examples from the ones available. To achieve this, multiple techniques such as rotation, flipping, cropping and shifting can be used. Algorithm 5 describes how data was augmented. Given an MR image, a large patch centered on the lesion was cropped first, which makes the rotation process easier. In fact, rotating the image without cropping it first also moves the region of interest. In this case, finding the exact coordinates becomes

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"
```

Algorithm 4 Create dataset - NumPy arrays to PNG conversion, organizing data into training and validation data

```
1:  procedure CREATE_PNGS(dataset_folder, output_folder, split)
2:      Create output directories: "output_folder/[Train|Val]/[0|1]"
3:
4:      true_nparrays_dict ← {"patient_id" : [file_name_1, ...]}
5:      number_of_patients_true ← len(true_nparrays_dict)
6:      number_of_training_patients_true ← ⌊number_of_patients_true * split⌋
7:      false_nparrays_dict ← {"patient_id" : [file_name_1, ...]}
8:
9:      index ← 0
10:     for patient_id, file_names in true_nparrays_dict do
11:         if index < number_of_training_patients_true then      ▷ Training set, True
12:             for file_name in file_names do
13:                 image_array ← load(file_name)
14:                 image ← convertArrayToGrayscaleImage(image_array)
15:                 Save image to "output_folder/Train/1"
16:         else                                                  ▷ Validation set, True
17:             for file_name in file_names do
18:                 image_array ← load(file_name)
19:                 image ← convertArrayToGrayscaleImage(image_array)
20:                 Save image to "output_folder/Val/1"
21:         index ← index + 1
22:
23:     number_of_training_patients_false ← ⌊number_of_patients_false * split⌋
24:     false_nparrays_dict ← {"patient_id" : [file_name_1, ...]}
25:     number_of_patients_false ← len(false_nparrays_dict)
26:     false_nparrays_dict ← {"patient_id" : [file_name_1, ...]}
27:
28:     index ← 0
29:     for patient_id, file_names in false_nparrays_dict do
30:         if index < number_of_training_patients_false then     ▷ Training set, False
31:             for file_name in file_names do
32:                 image_array ← load(file_name)
33:                 image ← convertArrayToGrayscaleImage(image_array)
34:                 Save image to "output_folder/Train/0"
35:         else                                                  ▷ Validation set, False
36:             for file_name in file_names do
37:                 image_array ← load(file_name)
38:                 image ← convertArrayToGrayscaleImage(image_array)
39:                 Save image to "output_folder/Val/0"
40:         index ← index + 1
```

more complicated. Therefore, a first large patch was created and then rotated. Afterwards, a smaller patch around the region of interest was extracted because of the potential padding induced by the rotation. This guarantees the addition of the minimal amount of padding pixels around the image to plug the holes, which decreases the probability of adding artificial information to the image.

Algorithm 5 Create augmented dataset - Augmentation method

procedure AUGMENT($image, img\_i, img\_j, available\_combinations\_list$)
    $index \leftarrow$ random index in [0,len(available_combinations_list)−1]
    $degree, prob\_flipping = available\_combinations\_list[index]$
    Delete element $available\_combinations\_list[index]$ ▷ Avoid duplication

    $temp\_image \leftarrow image$.crop(64x64, center=$(img\_i, img\_j)$) ▷ 1^{st} crop
    $temp\_image \leftarrow temp\_image$.rotate(degree) ▷ Rotate 1^{st} crop

    $width, height \leftarrow temp\_image$.size
    $x\_center \leftarrow \lfloor \frac{width}{2} \rfloor, y\_center \leftarrow \lfloor \frac{height}{2} \rfloor$ ▷ Find center pixel of 1^{st} crop
    $temp\_image \leftarrow image$.crop(32x32, center=$(x\_center, y\_center)$) ▷ 2^{nd} crop

    if $prob\_flipping > 0.5$ **then**
        $temp\_image \leftarrow temp\_image.horizontal\_flip()$
return $temp\_image$

7.1.5 Data visualization and verification

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 developped. Their goal was to compare our visual representation of an image to the one obtained in professional pieces of software. Also, PNG conversion was meticulously tested by comparing the pixel values from the original NumPy arrays with the PNG pixel values.

DICOM

Visualizing DICOM files is pretty straightforward since each file represent a single two-dimensional image. However, the way this standard works require a lot of pixel transformation and normalization to obtain the desired result (see section 4.2.2), which may cause errors. Our tools allows to display a single DICOM file as well as a sequence of files if the function is feeded with a directory. Users can then scroll through the Z-axis, displaying the next or previous slice.

NIfTI

As NIfTI files can be three or four-dimensional (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 allows to jump to the corresponding slice with respect to another timestamp. When reaching the end of a timestamp with the mouse, the first slice of the next timestamp is displayed. Furthermore, the

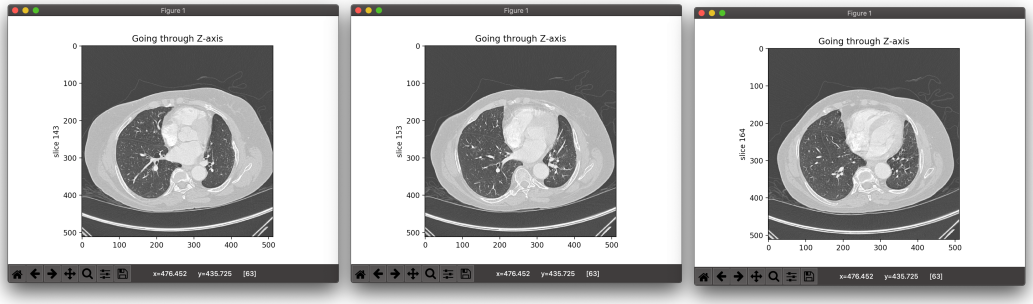

FIGURE 7.1: Visualization of a folder of DICOM files

three-dimensionality implies that the volume is viewable under 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.

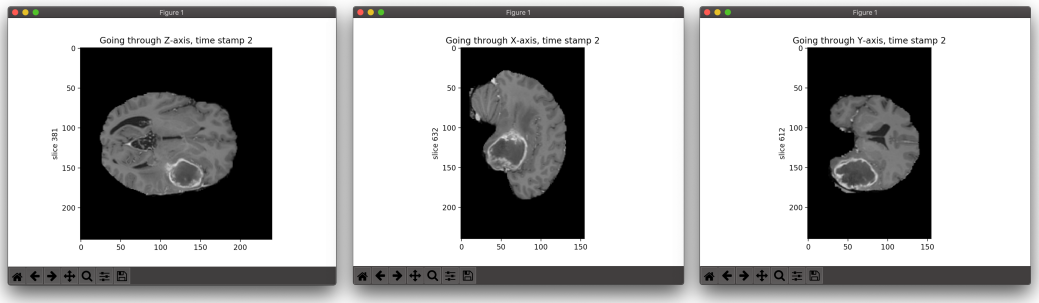

FIGURE 7.2: Four-dimensional NIfTI visualization

RAW

Medical RAW files are three-dimensional. Scrolling with the mouse goes through slices over a specific axis. Like NIfTI, the user can profit from the three-dimensionality by navigating through slices under three different perspectives.

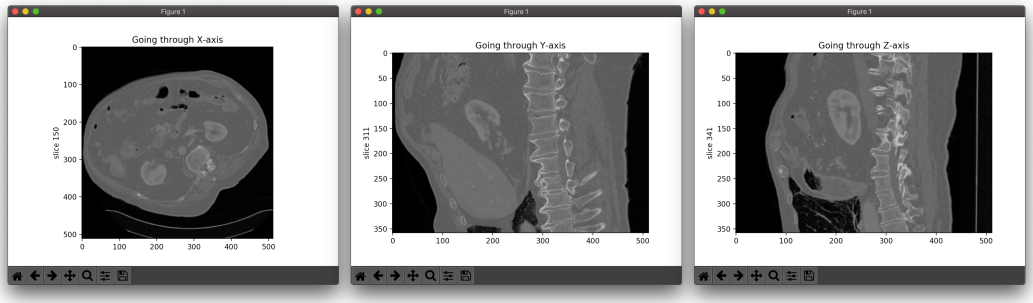

FIGURE 7.3: Three-dimensional RAW visualization

Bibliography

[1] 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).

[2] 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).

[3] 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).

[4] 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.

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

[6] 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).

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

[8] 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).

[9] 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).

[10] 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).

[11] 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).

[12] 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).

[13] 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).

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

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

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

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

[18] 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).

[19] 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).

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

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