

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

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

Abstract	iii
1 Introduction	1
2 Literature review	3
3 Deep learning	5
3.1 Introduction to deep learning	5
3.1.1 Historical background	5
3.1.2 What is a neural network?	6
3.1.3 Supervised learning and Unsupervised learning	7
3.2 Neural networks basics	7
3.2.1 Notation	7
3.2.2 Perceptrons	8
3.2.3 Activation functions	8
3.2.4 Multilayer perceptrons	11
3.3 Training a neural network	12
3.3.1 Forward propagation	12
Loss computation	13
3.3.2 Backpropagation	14
3.3.3 Metrics	15
3.3.4 Data	16
3.3.5 Weight initialization	17
3.3.6 Hyperparameters tuning	18
3.4 Convolutional Neural Networks	18
3.5 Transfer learning	18
4 Medical information	21
4.1 Cancer	21
4.1.1 Basics	21
4.1.2 Seriousness	21
4.2 Medical imaging	22
5 Approach	23
5.1 Improving cancer classification/detection	23
5.2 Process overview	23
6 Data processing	25
6.1 DICOM file format	25
6.1.1 Origin	25
6.1.2 Data format	25
6.1.3 Processing images	25
Order	25

	Data manipulation	25
6.2	NIfTI file format	27
	Origin	27
	Data format	27
	Overview of the header structure	27
6.3	RAW and MHD file formats	27
6.4	Conversion	27
6.5	Processing flow	27
6.5.1	PROSTATEx: From DICOM to NumPy arrays	28
6.5.2	Lung CT Challenge - From DICOM to NumPy arrays	29
6.5.3	NumPy arrays to PNG files	29
6.5.4	Data augmentation	30
6.5.5	Data visualization and verification	30
	DICOM	30
	NIfTI	32
	RAW	32
	Bibliography	35

List of Figures

3.1	Milestones	6
3.2	The perceptron model	9
3.3	The sigmoid function and its derivative	9
3.4	The tanh function and its derivative	10
3.5	The ReLU function and its derivative(not defined when $x=0$)	10
3.6	The ELU function and its derivative	11
3.7	Multilayer perceptrons	12
3.8	Forward propagation	13
3.9	5-folds cross-validation	17
4.1	MRI - PROSTATEx - From left to right: T2-weighted, ADC and DWI .	22
6.1	Visualization of a folder of DICOM files	32
6.2	Four-dimensional NIfTI visualization	33
6.3	Three-dimensional RAW visualization	33

Chapter 1

Introduction

Your introduction chapter here.

Chapter 2

Literature review

Literature review.

Chapter 3

Deep learning

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 accordingly. 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 all mentioned use cases clearly attests that this technology is a real breakthrough. Furthermore, since this field has recently gained interest (see section 3.1.1), multiple researches are still ongoing, which suggests that many exciting new applications will certainly be discovered.

3.1.1 Historical background

As described in figure 3.1 from article [18], the theoretical foundations of deep learning appeared long before the invention of computers. From the first attempts to understand the human brain until today, a long way has been made to establish the basic components of modern neural networks. One could ask: why did deep learning take off recently if the theory has been around for a long time?

First part of the answer is about the computational power. Indeed, deep learning algorithms need a lot of data to work properly and this requires powerful CPU/GPU that either didn't exist or were only available to a minority of people. One other main reason concerns the lack of data. Since deep learning algorithms "learn" from data, if not data is available, no learning is possible. Consequently, the era of Big Data has enhanced the deep learning possibilities. These two points are summarised: "The increase in model size overtime, due to the availability of faster CPUs, the advent of general purpose GPUs, fasternetwork, connectivity and better software infrastructure for distributed computing, is one of the most important trends in the history of deep learning. This trend is generally expected to continue well into the future"[7].

Finally, before the year 2012, the abilities of neural networks had not been publicly 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 Kunihiko Fukushima	introduced Self Organizing Map introduced Neocogitron, which inspired Convolutional Neural Network
1982	John Hopfield	introduced Hopfield Network
1985	Hilton & Sejnowski	introduced Boltzmann Machine
1986	Paul Smolensky Michael I. Jordan	introduced Harmonium, which is later known as Restricted Boltzmann Machine defined and introduced Recurrent Neural Network
1990	Yann LeCun	introduced LeNet, showed the possibility of deep neural networks in practice
1997	Schuster & Paliwal Hochreiter & Schmidhuber	introduced Bidirectional Recurrent Neural Network 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

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 images, sounds or whatever features that is numerical. 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 point of view, 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 .

$$y = f(x, \theta) \quad (3.1)$$

The parameters θ are estimated from the training samples.

3.1.3 Supervised learning and Unsupervised learning

In machine learning, there mainly exists two different kinds of task. The first one is "supervised learning". It includes all learning algorithms, whose training samples contains 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 use non labeled data and its main goal is to infer the natural structure present in the data. Since the model presented in this work belongs to the "supervised learning" category, all notions explained below refer to this kind of algorithms.

3.2 Neural networks basics

3.2.1 Notation

In order to respect a consistent mathematical notation to describe neural networks, this work will use the notation of Andrew Y. Ng, one of the pioneers of deep learning, that can be found in [16].

General comment

- Superscript (i) will denote the i^{th} training example.
- Superscript [l] will denote 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.
- $W^{[l]} \in \mathbb{R}^{\# \text{ of neurons in the next layer} \times \# \text{ of neurons in the previous layer}} = j \times k$ is the weight matrix at layer $[l]$.
- $b^{[l]} \in \mathbb{R}^{\# \text{ of units in next layer}}$ 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 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.
- $J(x, W, b, y)$ or $J(\hat{y}, y)$ 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 the input $x = (x_1, \dots, x_n)$ weighted by a vector of weights $w = (w_1, \dots, w_n)$, completed by a bias b_j and associated to a non linear activation function g :

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

Schematically, a perceptron can be represented as shown in figure 3.2. Each input is multiplied with its corresponding weight and the sum of this result is then passed through a non linear function, called an "activation function". This activation function acts like a threshold that decides the proportion of the result that has to be given further. There exist multiple activation functions (see 3.2.3). It is extremely important to use a non linear functions instead of a linear function. Indeed, in a neural network the output of a perceptron is given as input to the others (see 3.2.4). Consequently, if only linear functions are used, linear outputs are given as inputs to other linear functions. Since the composition of two linear functions is itself a linear function, there would be no sense to assemble perceptrons to create neural networks of multiple layers.

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 to send further. Activation functions have to be non-linear in order to approximate extremely complex functions. In fact, neural networks are considered as universal approximators. Article [\[11\]](#) claims that "multilayer feedforward networks are capable of approximating any measurable function to any desired degree of accuracy, in a very specific and satisfying sense." According to [\[5\]](#) the most commonly used activation functions are:

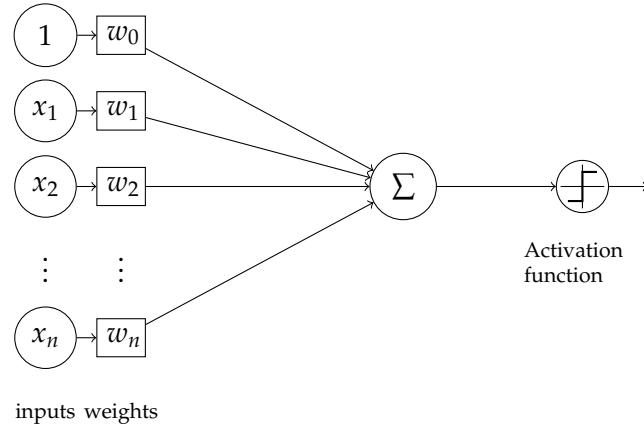


FIGURE 3.2: The perceptron model

Sigmoid function

The sigmoid function is defined as:

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

Its derivative is:

$$g'(x) = g(x)(1 - g(x)) \quad (3.4)$$

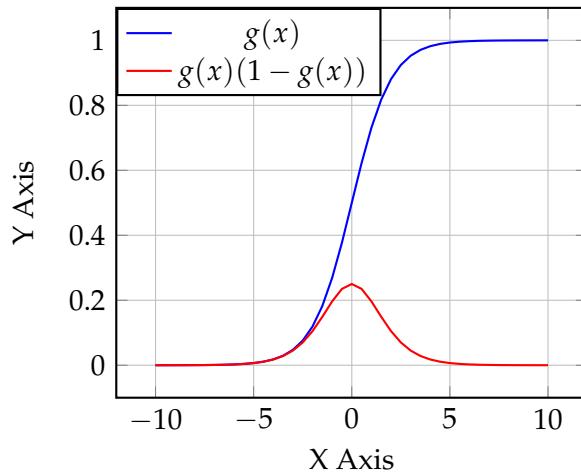


FIGURE 3.3: The sigmoid function and its derivative

Tanh function

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

Its derivative is:

$$g'(x) = \tanh'(x) = 1 - \tanh^2(x) \quad (3.6)$$

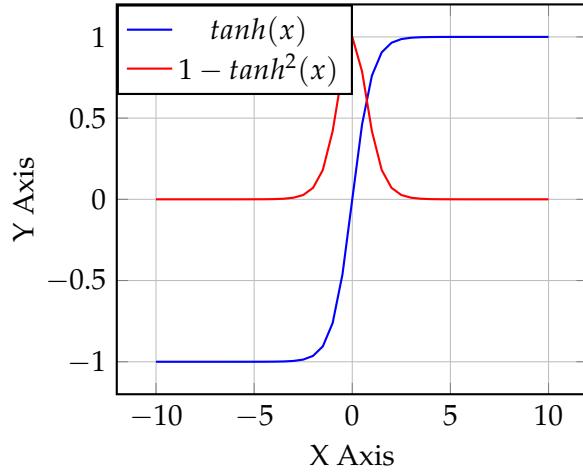


FIGURE 3.4: The tanh function and its derivative

ReLU function

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

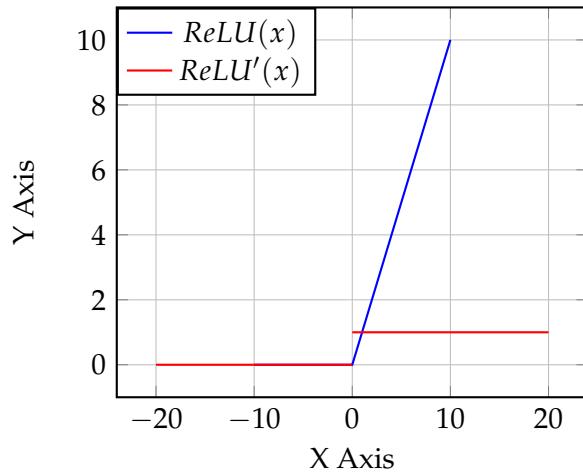


FIGURE 3.5: The ReLU function and its derivative(not defined when x=0)

ELU function

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

Its derivative is:

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

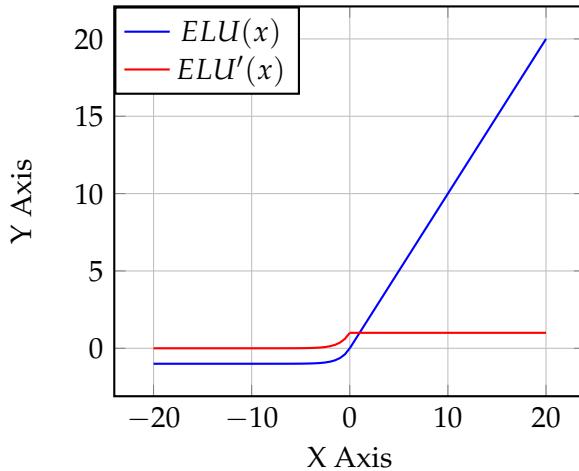


FIGURE 3.6: The ELU function and its derivative

3.2.4 Multilayer perceptrons

A multilayer perceptron is an artificial neural network type. Article [4] defines the multilayer perceptrons as "feedforward networks with one or more layers of units between the input and output layers. The output units represent a hyperplane in the space of the input patterns."

A multilayer perceptron is composed of L layers each of them having 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, the data is provided to the network 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 their dimension. 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, under the assumption that the number of samples is 100 and that the batch size is 32, 4 batches in total will be given as input to the network.
- The hidden layer(s): hidden layers stand for all layers that are between the input layer and the output layer. For each of them are associated weights and biases (W, b), respectively denoted by $W^{[l]} \in \mathbb{R}$ and $b^{[l]}$, where $W_{ij}^{[l]}$ correspond 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 $l + 1$. The weights and the biases of the network are the parameters that have to be optimized in order to have the best mapping between the inputs 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 3.3.5).
- 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 being the number of different labels. In a classification task, whose goal is to assign to each input a

specific class, the \hat{y} could be the probability of belonging to each class. In that case, the *softmax* activation function would be used.

The advantage of organizing the weights, biases and inputs in matrices is due to the ability of modern CPU/GPU to quickly perform linear algebra computations. This way of structuring the network component is called "vectorization" and avoids at most using loops in the code, which could considerably slow down the computations. The figure 3.7 globally illustrates the concept of a 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 each layer is $n_h^1 = 4$, $n_h^2 = 2$, $n_h^3 = 1$. The network has the weights $W^{(1)} \in \mathbb{R}^{2x3}$, $W^{(2)} \in \mathbb{R}^{1x2}$ and the biases $b^{[1]} = 2$, $b^{[2]} = 1$.

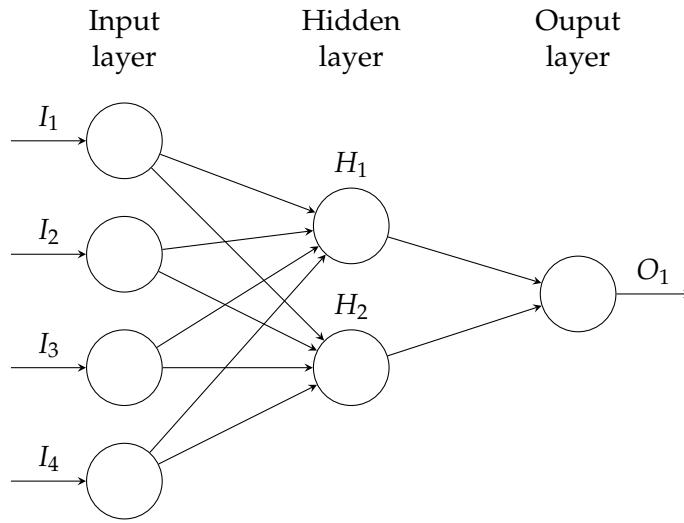


FIGURE 3.7: Multilayer perceptrons

3.3 Training a neural network

Training a neural network is composed of multiple steps. The first one is the "forward propagation step". It consists in giving to the untrained neural network examples that need to be classified (or segmented, depending on the task) and to transmit them through all layers of the network. Passing all batches as input to a network is called an "epoch". At the end of each 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 minima of the loss function. 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, the forward propagation for a given layer l is given by two equations:

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

This first equation denotes the weighted sum of the input given to the l^{th} layer before passing through the activation function g . The second equation describes the effect

of the activation function:

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

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

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

The figure 3.8 illustrates the computation of the forward propagation for the l^{th} layer. The matrix of weight $W_{jk}^{[l]}$ correspond to the weights associated with the connection between perceptron k in layer l and perceptron j in layer $l + 1$. The latter is multiplied by the output of previous layer $a_j^{[l-1]}$, the biased $b_j^{[l]}$ is added and the entire result is given as input to the activation function.

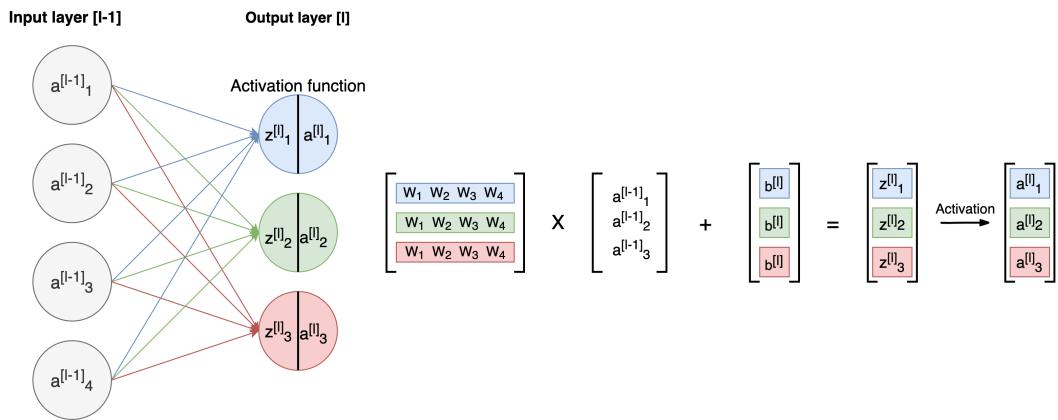


FIGURE 3.8: Forward propagation

Loss computation

"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 have 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 given 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:

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

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

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

In multiclass classification, the categorical crossentropy is:

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

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 weight 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.

3.3.2 Backpropagation

Backpropagation relies on a technique called "gradient descent" to minimize the cost function $J(W, b)$. Globally, "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" $\delta_i^{(l)}$ 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 $\delta_i^{(n_l)}$ (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"[15].

In other words, after each forward pass through the entire network, backpropagation performs a backward pass with the goal of minimizing the cost function by adjusting model's parameters. 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 $J(x_1, x_2, \dots, x_m)$ in point x is given by:

$$\frac{\partial J}{\partial x} = \left[\frac{\partial J}{\partial x_1}, \frac{\partial J}{\partial x_2}, \dots, \frac{\partial J}{\partial x_n} \right] \quad (3.16)$$

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". For weights parameters, the chain rule is:

$$\frac{\partial J}{\partial w_{jk}^l} = \frac{\partial J}{\partial z_j^l} * \frac{\partial z_j^l}{\partial w_{jk}^l} \quad (3.17)$$

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

$$\frac{\partial J}{\partial b_j^l} = \frac{\partial J}{\partial z_j^l} * \frac{\partial z_j^l}{\partial b_j^l} \quad (3.18)$$

Once the gradients of each parameters of the neural network are computed, they have to be updated. The weights update is described by the following equation:

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

The biases update corresponds to:

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

The hyperparameter α of these two equations is named "learning rate". It determines the gradient's influence at each epoch and has to be manually tuned.

3.3.3 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 established [6]:

- Accuracy: it is defined as the ratio of the correctly labeled subjects to the whole pool of subjects.

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

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: it is the ratio of the correctly positive labeled subjects by the model to all positive labeled subjects.

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

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):

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

This metric is recommended when the occurrence 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:

$$F1-score = \frac{2 * recall * precision}{(recall + precision)} \quad (3.24)$$

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:

$$Specificity = \frac{TN}{(TN + FP)} \quad (3.25)$$

This metric is recommended when the occurrence 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.

- AUC-ROC: "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]."

3.3.4 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 dataset that is used to train the neural network (the weights and biases).
- The validation set: this is the dataset that is used to evaluate a trained model. Usually, the evaluation on the validation set is made every x (a fixed number) epochs. 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 can generalize 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" (overfitting describes the behaviour of a model that too closely fits to the training set and don't generalize well). Indeed, if the validation loss continually increases for a certain number of epochs, going on with the training will increase the overfitting. This technique is called "early stopping".
- The test set: this last dataset is used to establish the final evaluation of the model and also contains unseen data.

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-folds 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, all results of all test sets are averaged, which provides a good estimation of the model performances. This technique is summarized on figure 3.9.

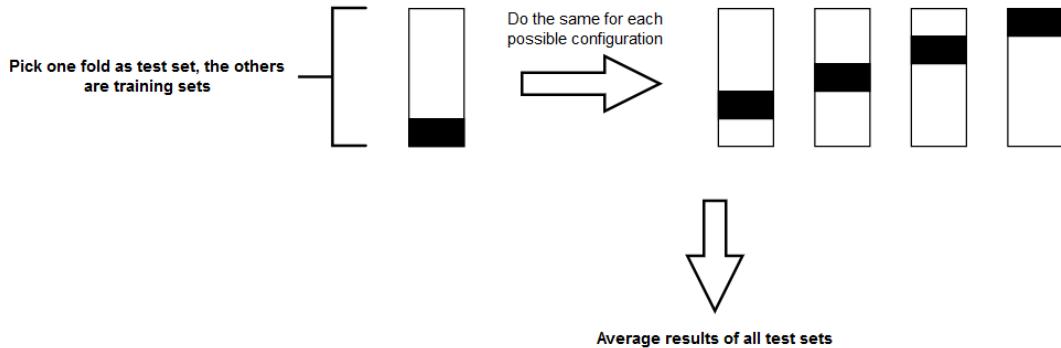


FIGURE 3.9: 5-folds cross-validation

3.3.5 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. 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"[3].

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:

$$\frac{\sqrt{6}}{\sqrt{n_h^{[l]} + n_h^{[l+1]}}} \quad (3.26)$$

where $n_h^{[l]}$ is the number of incoming network connection 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 about zero and don't have outputs inside [-1,1] such as ReLU or ELU, Kaiming initialization is an alternative. It consists in multiplying the random initialized matrix weight by:

$$\frac{\sqrt{2}}{\sqrt{n_h^{[l]}}} \quad (3.27)$$

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

3.3.6 Hyperparameters tuning

Hyperparameters denotes 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 at most the cost function keeping a good 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 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"^[13] 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"^[13], 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"^[13]. There also exists methods that, instead of keeping a fixed learning rate for the entire training, reduce it at each epoch (learning rate decay) or each time the loss on the validation set does not decrease (learning rate schedulers). 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"^[12]. Furthermore, "networks train faster because the weights update is done after each propagation"^[12]. Nevertheless, "the smaller the batch the less accurate the estimate of the gradient will be"^[12]. 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 network 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 continually increase. It is exactly at this moment that the training should be stopped. 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 x epochs.

3.4 Convolutional Neural Networks

3.5 Transfer learning

"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 all or 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 can simply be downloaded from institutions that release large and trained models for similar tasks.

It mainly provides three 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 improvement of skill 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) 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" [10] and "The Hallmarks of Cancer: The Next Generation" [9], 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 [8]. 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 [2]. The earlier they are detected, the higher the chances of recovering are.

4.2 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 outputted 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 6.1 shows a lung CT scan.

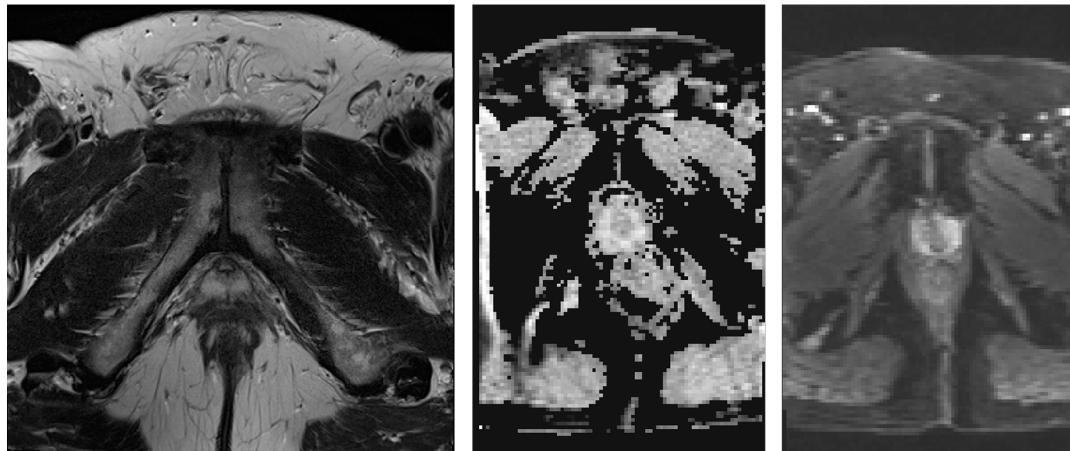


FIGURE 4.1: MRI - PROSTATEx - From left to right: T2-weighted, ADC and DWI

Chapter 5

Approach

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

5.1 Improving cancer classification/detection

5.2 Process overview

Find data -> Process data -> Verify data processing (visualization, comparing NumPy arrays and PNGs, etc.) -> Train

Chapter 6

Data processing

6.1 DICOM file format

6.1.1 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.

6.1.2 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.

6.1.3 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:

$$HU = m * P + b \quad (6.1)$$

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:

$$\text{if } (P \leq c - 0.5 - \frac{w - 1}{2}), \text{ then } y = y_{min} \quad (6.2)$$

$$\text{else if } (P > c - 0.5 + \frac{w - 1}{2}), \text{ then } y = y_{max} \quad (6.3)$$

$$\text{else } y = (\frac{P - (c - 0.5)}{w - 1}) + 0.5) * (y_{max} - y_{min}) + y_{min} \quad (6.4)$$

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 6.2, 6.3 and 6.4 ensure that the pixel values are correctly distributed within the output range.

6.2 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/>

6.3 RAW and MHD file formats

Some datasets use a combination of RAW and MHD files. The latter contain meta-information 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.

6.4 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.

6.5 Processing flow

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

Algorithm 1 16 to 8 bits conversion

```

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

```

6.5.1 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), "DCM-SerNumber" (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.

Algorithm 2 PROSTATEx preprocessing

```

1: procedure MAIN(dataset_folder, findings_CSV, slices_CSV, output_folder)
2:   Create output directories: "output_folder/True", "output_folder/False"
3:
4:   findings  $\leftarrow$  read_CSV(findings_CSV)            $\triangleright$  ProstateX-Findings-Train.csv
5:   slices  $\leftarrow$  read_CSV(slices_CSV)            $\triangleright$  ProstateX-Images-Train.csv
6:   meta  $\leftarrow$  merge(findings, slices)     $\triangleright$  Both CSV files are complementary to each
other.
7:
8:   for row in meta do
9:     patient_id  $\leftarrow$  row["ProxID"]
10:    finding_id  $\leftarrow$  row["fid"]
11:    mri_type_number  $\leftarrow$  row["DCMSerNumber"]
12:    clinical_significance  $\leftarrow$  row["ClinSig"]
13:    img_i, img_j, img_k  $\leftarrow$  row["ijk"]
14:    slice_number  $\leftarrow$  img_k + 1       $\triangleright$  CSV indexing in  $[0, \infty]$ , DICOM in  $[1, \infty]$ 
15:
16:   for visit in patient_id's folder do
17:     for mri_type in visit do
18:       if mri_type starts with "mri_type_number-" then
19:         for dicom_file in mri_type do
20:           if slice_number == dicom_file.InstanceNumber then
21:             slice  $\leftarrow$  normalize_dicom(dicom_file)  $\triangleright$  see section 6.1.3
22:             Save slice in "output_folder/clinical_significance"
```

6.5.2 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.

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  $\leftarrow$  read_CSV(train_CSV)                                 $\triangleright$  CalibrationSet.csv
5:   csv_test  $\leftarrow$  read_CSV(test_CSV)                                $\triangleright$  TestSet.csv
6:   csv_concatenated  $\leftarrow$  concat(csv_training, csv_test)     $\triangleright$  Both CSV files contain
      similar information about different patients.
7:
8:   for row in csv_concatenated do
9:     patient_id  $\leftarrow$  row["Scan Number"]
10:    slice_number  $\leftarrow$  row["Nodule Center Image"]            $\triangleright$  Value in  $[1, \infty]$ 
11:    finding_id  $\leftarrow$  row["Nodule Number"]
12:    clinical_significance  $\leftarrow$  row["Diagnosis"]
13:
14:    if clinical_significance == "malignant" or "Primary lung cancer" then
15:      clinical_significance  $\leftarrow$  True
16:    else if clinical_significance == "benign" or "Benign nodule" then
17:      clinical_significance  $\leftarrow$  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  $\leftarrow$  normalize_dicom(dicom_file)       $\triangleright$  see section 6.1.3
26:            Save slice in "output_folder/clinical_significance"
```

6.5.3 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, 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 ($\text{True} \equiv 1$, $\text{False} \equiv 0$) 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.

6.5.4 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 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.

6.5.5 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 developed. 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 6.1.3), 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.

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  $\leftarrow \{ "patient\_id" : [file\_name\_1, \dots] \}$ 
5:   number_of_patients_true  $\leftarrow \text{len}(\text{true\_nparrays\_dict})$ 
6:   number_of_training_patients_true  $\leftarrow \lfloor \text{number\_of\_patients\_true} * \text{split} \rfloor$ 
7:   false_nparrays_dict  $\leftarrow \{ "patient\_id" : [file\_name\_1, \dots] \}$ 
8:
9:   index  $\leftarrow 0$ 
10:  for patient_id, file_names in true_nparrays_dict do
11:    if index  $<$  number_of_training_patients_true then  $\triangleright$  Training set, True
12:      for file_name in file_names do
13:        image_array  $\leftarrow \text{load}(\text{file\_name})$ 
14:        image  $\leftarrow \text{convertArrayToGrayscaleImage}(\text{image\_array})$ 
15:        Save image to "output_folder/Train/1"
16:    else  $\triangleright$  Validation set, True
17:      for file_name in file_names do
18:        image_array  $\leftarrow \text{load}(\text{file\_name})$ 
19:        image  $\leftarrow \text{convertArrayToGrayscaleImage}(\text{image\_array})$ 
20:        Save image to "output_folder/Val/1"
21:    index  $\leftarrow \text{index} + 1$ 
22:
23:    number_of_training_patients_false  $\leftarrow \lfloor \text{number\_of\_patients\_false} * \text{split} \rfloor$ 
24:    false_nparrays_dict  $\leftarrow \{ "patient\_id" : [file\_name\_1, \dots] \}$ 
25:    number_of_patients_false  $\leftarrow \text{len}(\text{false\_nparrays\_dict})$ 
26:    false_nparrays_dict  $\leftarrow \{ "patient\_id" : [file\_name\_1, \dots] \}$ 
27:
28:    index  $\leftarrow 0$ 
29:    for patient_id, file_names in false_nparrays_dict do
30:      if index  $<$  number_of_training_patients_false then  $\triangleright$  Training set, False
31:        for file_name in file_names do
32:          image_array  $\leftarrow \text{load}(\text{file\_name})$ 
33:          image  $\leftarrow \text{convertArrayToGrayscaleImage}(\text{image\_array})$ 
34:          Save image to "output_folder/Train/0"
35:      else  $\triangleright$  Validation set, False
36:        for file_name in file_names do
37:          image_array  $\leftarrow \text{load}(\text{file\_name})$ 
38:          image  $\leftarrow \text{convertArrayToGrayscaleImage}(\text{image\_array})$ 
39:          Save image to "output_folder/Val/0"
40:    index  $\leftarrow \text{index} + 1$ 

```

Algorithm 5 Create augmented dataset - Augmentation method

```

1: procedure AUGMENT(image, img_i, img_j, available_combinations_list)
2:   index  $\leftarrow$  random index in  $[0, \text{len}(\text{available\_combinations\_list}) - 1]$ 
3:   degree, prob_flipping = available_combinations_list[index]
4:   Delete element available_combinations_list[index]            $\triangleright$  Avoid duplication
5:
6:   temp_image  $\leftarrow$  image.crop(64x64, center=(img_i, img_j))            $\triangleright$  1st crop
7:   temp_image  $\leftarrow$  temp_image.rotate(degree)                          $\triangleright$  Rotate 1st crop
8:
9:   width, height  $\leftarrow$  temp_image.size
10:  x_center  $\leftarrow$   $\lfloor \frac{\text{width}}{2} \rfloor$ , y_center  $\leftarrow$   $\lfloor \frac{\text{height}}{2} \rfloor$            $\triangleright$  Find center pixel of 1st crop
11:  temp_image  $\leftarrow$  image.crop(32x32, center=(x_center, y_center))       $\triangleright$  2nd crop
12:
13:  if prob_flipping  $> 0.5$  then
14:    temp_image  $\leftarrow$  temp_image.horizontal_flip()
15: return temp_image

```

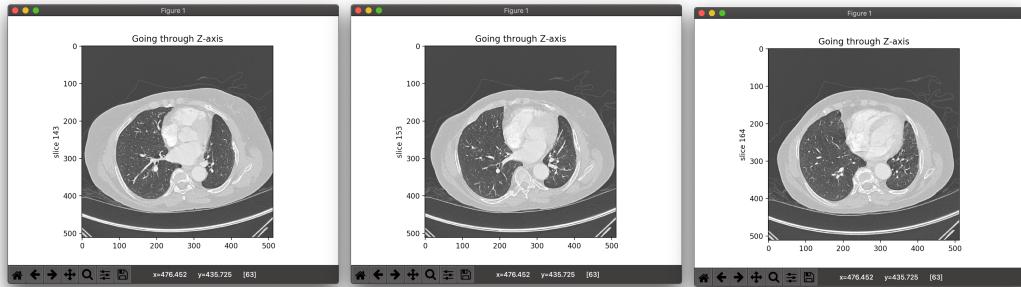


FIGURE 6.1: Visualization of a folder of DICOM files

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 allow 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 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.

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.

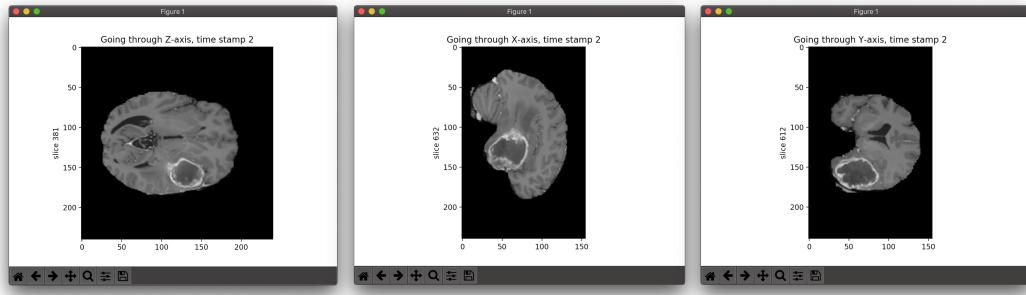


FIGURE 6.2: Four-dimensional NIfTI visualization

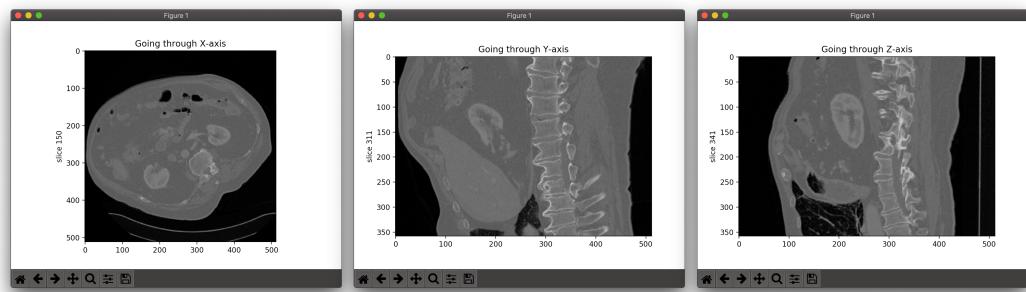


FIGURE 6.3: Three-dimensional RAW visualization

Bibliography

- [1] Jason Brwonlee. *A Gentle Introduction to Transfer Learning for Deep Learning*. 2017. URL: <https://machinelearningmastery.com/transfer-learning-for-deep-learning/>.
- [2] American Society of Clinical Oncology (ASCO). *Stages of Cancer*. 2018. URL: <https://www.cancer.net/navigating-cancer-care/diagnosing-cancer/stages-cancer> (visited on 08/12/2019).
- [3] James Dellinger. *Weight Initialization in Neural Networks: A Journey From the Basics to Kaiming*. 2019. URL: <https://towardsdatascience.com/weight-initialization-in-neural-networks-a-journey-from-the-basics-to-kaiming-954fb9b47c79> (visited on 12/11/2019).
- [4] K.-L Du and M.N.s Swamy. "Multilayer Perceptrons: Architecture and Error Backpropagation". In: Dec. 2014, pp. 83–126. ISBN: 978-1-4471-5570-6. DOI: 10.1007/978-1-4471-5571-3_4.
- [5] Thomas Epelbaum. "Deep learning: Technical introduction". In: *arXiv e-prints*, arXiv:1709.01412 (Sept. 2017), arXiv:1709.01412. arXiv: 1709.01412 [stat.ML].
- [6] Salma Ghoneim. *Accuracy, Recall, Precision, F-Score & Specificity, which to optimize on?* 2019. URL: <https://towardsdatascience.com/accuracy-recall-precision-f-score-specificity-which-to-optimize-on-867d3f11124> (visited on 12/11/2019).
- [7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [8] F.L. Greene et al. *AJCC Cancer Staging Handbook: TNM Classification of Malignant Tumors*. STAT!Ref electronic medical library. Springer New York, 2002. ISBN: 9780387952703. URL: <https://books.google.ch/books?id=0SP-gyH0-bMC>.
- [9] Douglas Hanahan and Robert A. Weinberg. "Hallmarks of Cancer: The Next Generation". In: *Cell* 144.5 (2011), pp. 646 –674. ISSN: 0092-8674. DOI: <https://doi.org/10.1016/j.cell.2011.02.013>. URL: <http://www.sciencedirect.com/science/article/pii/S0092867411001279>.
- [10] Douglas Hanahan and Robert A. Weinberg. "The Hallmarks of Cancer". In: *Cell* 100.1 (2000), pp. 57 –70. ISSN: 0092-8674. DOI: [https://doi.org/10.1016/S0092-8674\(00\)81683-9](https://doi.org/10.1016/S0092-8674(00)81683-9). URL: <http://www.sciencedirect.com/science/article/pii/S0092867400816839>.
- [11] K. Hornik, M. Stinchcombe, and H. White. "Multilayer Feedforward Networks Are Universal Approximators". In: *Neural Netw.* 2.5 (July 1989), pp. 359–366. ISSN: 0893-6080. DOI: 10.1016/0893-6080(89)90020-8. URL: [http://dx.doi.org/10.1016/0893-6080\(89\)90020-8](http://dx.doi.org/10.1016/0893-6080(89)90020-8).
- [12] itdixer. *What is batch size in neural network?* 2019. URL: <https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network> (visited on 12/17/2019).

- [13] Keitakurita. *Learning Rate Tuning in Deep Learning: A Practical Guide*. 2018. URL: <https://mlexplained.com/2018/01/29/learning-rate-tuning-in-deep-learning-a-practical-guide/>.
- [14] Sarang Narkhede. *Understanding AUC - ROC Curve*. 2018. URL: <https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5> (visited on 12/11/2019).
- [15] Andrew Ng et al. “Sparse autoencoder”. In: *CS294A Lecture notes 72.2011* (2011), pp. 1–19.
- [16] Andrew Y. Ng and Kian Katanforoosh. *CS229 Lecture Notes: Deep Learning*. http://cs229.stanford.edu/notes/cs229-notes-deep_learning.pdf. 2019.
- [17] Michael A. Nielsen. *Neural Networks and Deep Learning*. misc. 2018. URL: <http://neuralnetworksanddeeplearning.com/>.
- [18] Haohan Wang and Bhiksha Raj. “On the Origin of Deep Learning”. In: (Feb. 2017).