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

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 [10], the theoretical foundations of deep learning appeared long before the invention of the 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"[3].

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	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

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.

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

The parameters θ are estimated from the training samples.

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 [8].

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.
- $W^{[l]} \in \mathbb{R}^{\text{\# of neurons in the next layer = j x \# of neurons in the previous layer = k}}$ 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 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"[9]. Today, they are called "artificial neurons" or simply "neurons".
A perceptron j is a function f of the 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:

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

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 where perceptrons are assembled together, the output of a perceptron is given as input to the others (see 3.2.4). Consequently, if only linear functions were used, linear outputs would be 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.

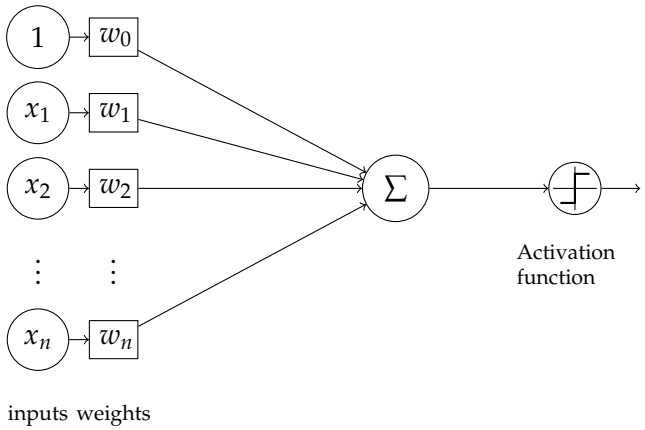

FIGURE 3.2: The perceptron model

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 be 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 [7] 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 [2] the most commonly used activation functions are:

Sigmoid function
The sigmoid function is defined as:

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

Its derivative is:

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

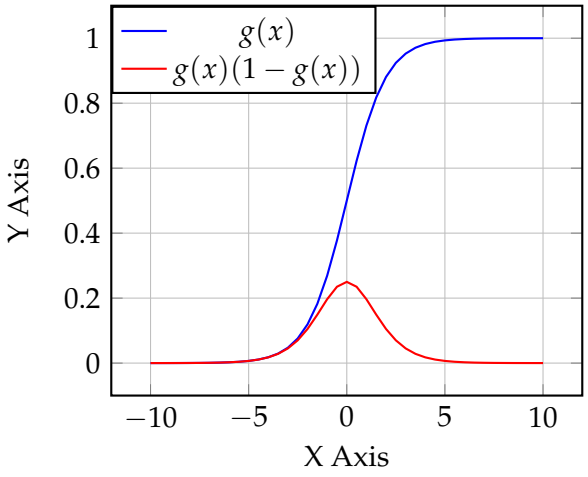

FIGURE 3.3: The sigmoid function and its derivative

Tanh function

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

Its derivative is:

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

ReLU function

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

ELU function

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

Its derivative is:

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

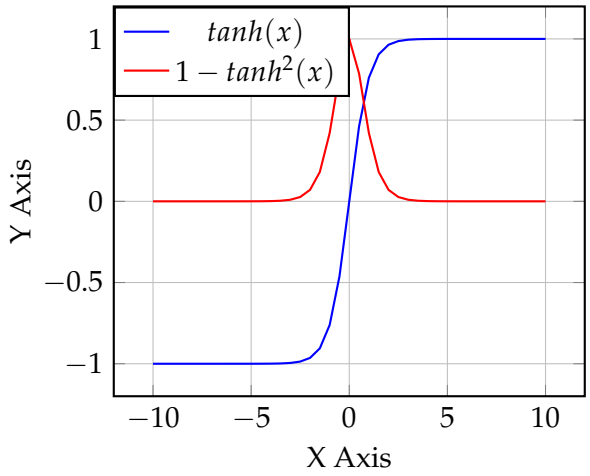

FIGURE 3.4: The tanh function and its derivative

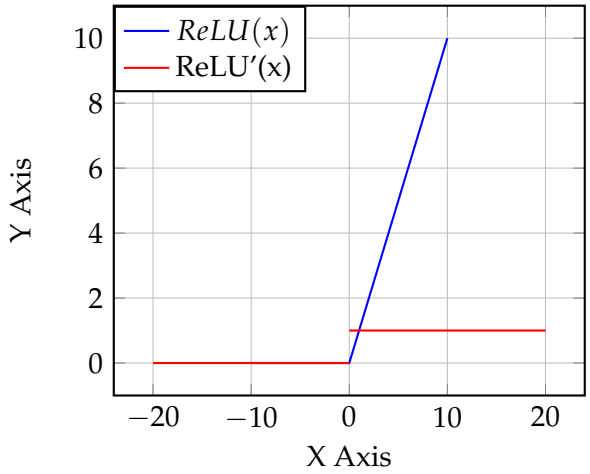

FIGURE 3.5: The ReLU function and its derivative(not defined when x=0)

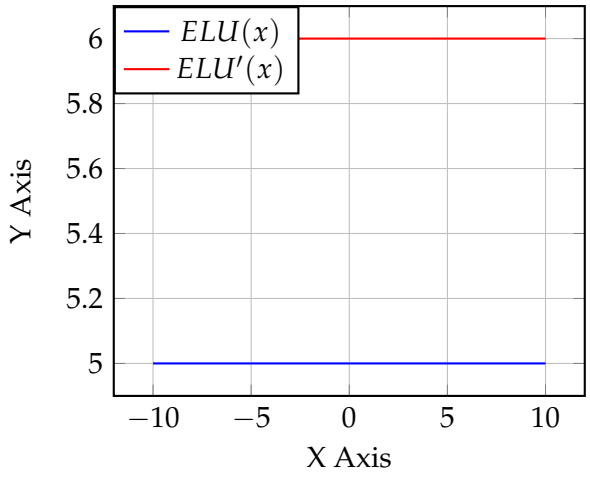

FIGURE 3.6: The ELU function and its derivative

3.2.4 Multilayer perceptrons

The figure 3.7 globally illustrates the concept of a multilayer perceptrons: all inputs are given to all neurons of the first layer. Each neuron sums the incoming inputs, passes the result through a non linear function and sends it to all neuron of the next layer along an edge. Each edge between two neurons (i.e perceptrons) has a certain weight (a numerical value) that weights the importance of the output of the previous neuron.

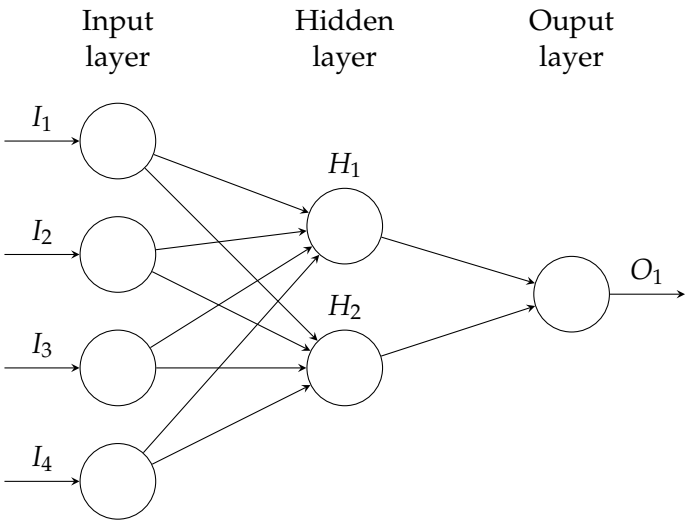

FIGURE 3.7: Neural Network example

A multilayer perceptron is an artifical neural network type. Article [**23**] defines the multilayer perceptrons in the following way: "Multilayer perceptron are feed-forward 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 perceptrons is composed of multiple perceptrons that are arranged into multiple layers. When the input layer receives the signal, transmits it to the hidden layer

3.2.5 Forward propagation

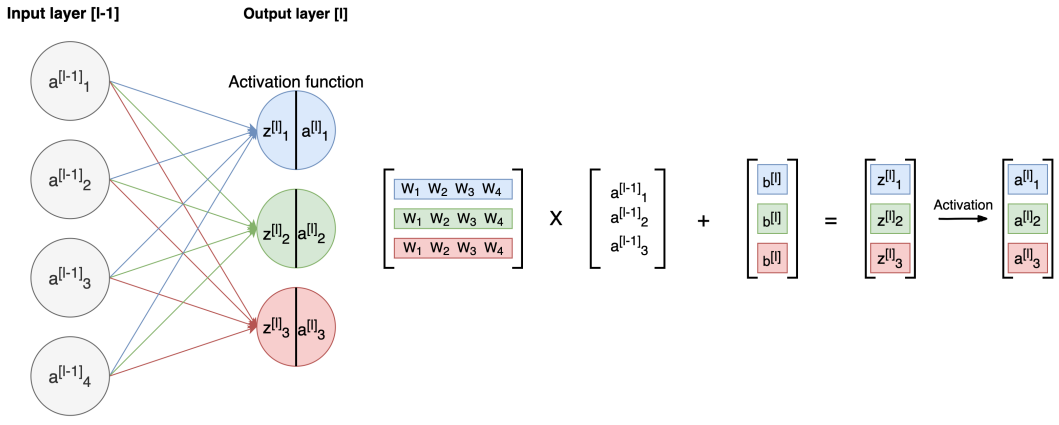

FIGURE 3.8: Forward propagation

3.2.6 Backpropagation

Loss function

Weight update

3.3 Training a neural network

3.3.1 Data

Training set, validation set, test set, supervised learning, unsupervised learning,

3.3.2 Weight initialization

3.3.3 Hyperparameters

lr, batch size, epochs, optimizer, learning rate decay, ...

3.4 Deep learning models

xxx

Convolutional Neural Networks

Recurrent Neural Networks

3.5 Transfer learning

3.6 Deep learning frameworks

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" [6] and "The Hallmarks of Cancer: The Next Generation" [5], 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 [1]. The earlier they are detected, the higher the chances of recovering are.

4.2 Medical imaging

4.2.1 Types

MRI

CT

Mammography

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:

$$HU = m * P + b \tag{6.1}$$

   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:

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

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

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

   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 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.

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

procedure 16_TO_8_BITS_CONVERSION($pixel\_array$)
    $Pixel_{min} \leftarrow$ minimal pixel value in $pixel\_array$
    $Pixel_{max} \leftarrow$ maximal pixel value in $pixel\_array$
    for $pixel\_value$ in $pixel\_array$ **do**
        $pixel\_value \leftarrow \frac{(pixel\_value - Pixel_{min}) * 255.0}{Pixel_{max} - Pixel_{min}}$
    Modify object type to 8 bits
    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), "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.

Algorithm 2 PROSTATEx preprocessing

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

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

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

        for $visit$ *in* $patient\_id$'s folder **do**
            for mri_type in $visit$ **do**
                if mri_type starts with "mri_type_number-" **then**
                    for $dicom\_file$ in mri_type **do**
                        if $slice\_number == dicom\_file.InstanceNumber$ **then**
                            $slice \leftarrow normalize\_dicom(dicom\_file)$ ▷ see section 6.1.3
                            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 ← 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 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 (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.

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 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 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 ← {"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
```

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 $temp\_image$

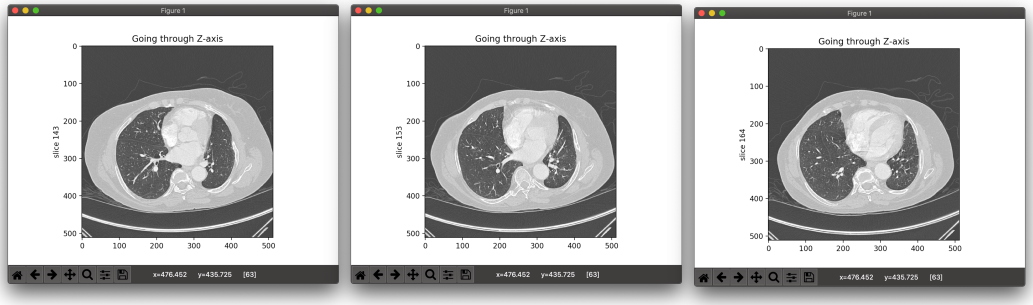

FIGURE 6.1: Visualization of a folder of DICOM files

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 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 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.

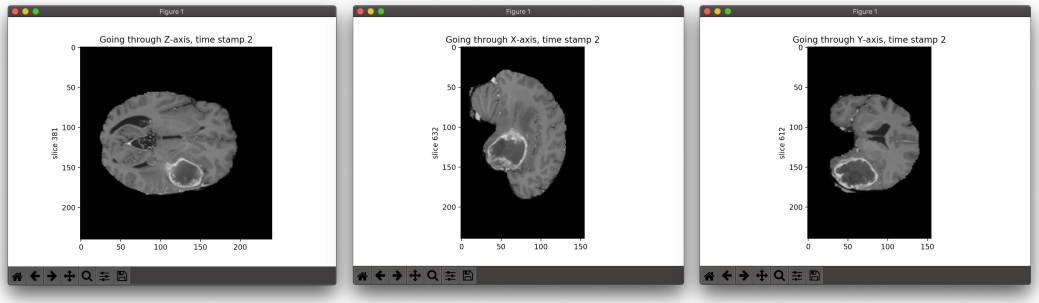

FIGURE 6.2: Four-dimensional NIfTI visualization

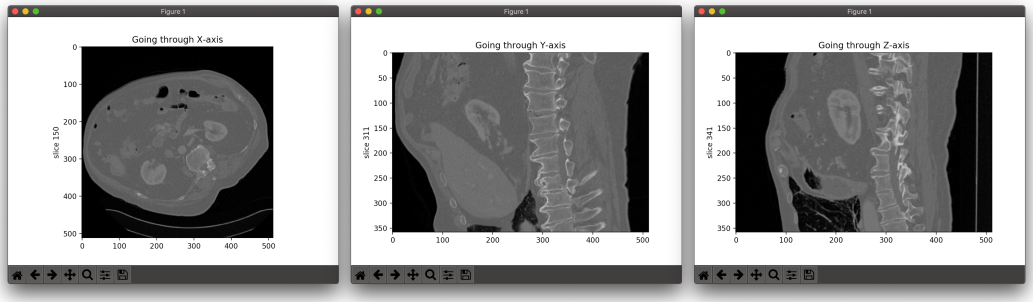

FIGURE 6.3: Three-dimensional RAW visualization

Bibliography

[1] American Society of Clinical Oncology (ASCO). *Stages of Cancer*. 2018. URL: https://www.cancer.net/navigating-cancer-care/diagnosing-cancer/stages-cancer (visited on 02/12/2019).

[2] Thomas Epelbaum. "Deep learning: Technical introduction". In: *arXiv e-prints*, arXiv:1709.01412 (2017), arXiv:1709.01412. arXiv: 1709.01412 [stat.ML].

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[4] 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.

[5] 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.

[6] 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. URL: http://www.sciencedirect.com/science/article/pii/S0092867400816839.

[7] 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.

[8] Andrew Y. Ng. *Standard notations for Deep Learning*. https://d3c33hcgiwev3.cloudfront.net/_106ac679d8102f2bee614cc67e9e5212_deep-learning-notation.pdf?Expires=1575417600&Signature=ihxy8W4DX8SGIralHZjhqsCxITe7bBrHRMghV73obps
QVKpmd5q1rPYPGcfoU1AzgMqkDZbao2IdnaS9Lp7hh-iYd1-bCPvFo6AjhaJGOnECq1SWayfeVZCLleB41zbA
&Key-Pair-Id=APKAJLTNE6QMUY6HBC5A. 2019.

[9] Michael A. Nielsen. *Neural Networks and Deep Learning*. misc. 2018. URL: http://neuralnetworksanddeeplearning.com/.

[10] Haohan Wang and Bhiksha Raj. "On the Origin of Deep Learning". In: (Feb. 2017).