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

Data processing

1.1 DICOM file format

1.1.1 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.

1.1.2 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.

1.1.3 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:

   $$HU = m * P + b \tag{1.1}$$

   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:

   $$\text{if } (P \leq c - 0.5 - \frac{w-1}{2}), \text{ then } y = y_{min} \tag{1.2}$$

   $$\text{else if } (P > c - 0.5 + \frac{w-1}{2}), \text{ then } y = y_{max} \tag{1.3}$$

   $$\text{else } y = (\frac{P - (c - 0.5)}{w-1}) + 0.5) * (y_{max} - y_{min}) + y_{min} \tag{1.4}$$

   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 1.2, 1.3 and 1.4 ensure that the pixel values are correctly distributed within the output range.

1.2 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/`

1.3 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.

1.4 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 1 was applied.

1.5 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"

Algorithm 1 16 to 8 bits conversion

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

(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

```
1: procedure MAIN($dataset\_folder, findings\_CSV, slices\_CSV, output\_folder$)
2:     Create output directories: $"output\_folder/True", "output\_folder/False"$
3:
4:     $findings \leftarrow read\_CSV(findings\_CSV)$          ▷ ProstateX-Findings-Train.csv
5:     $slices \leftarrow read\_CSV(slices\_CSV)$              ▷ ProstateX-Images-Train.csv
6:     $meta \leftarrow merge(findings, slices)$  ▷ 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:        $slice\_number \leftarrow img\_k + 1$       ▷ 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:                            $slice \leftarrow normalize\_dicom(dicom\_file)$  ▷ see section 1.1.3
22:                            Save $slice$ in $"output\_folder/clinical\_significance"$
```

1.6 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

```
procedure MAIN($dataset\_folder, train\_CSV, test\_CSV, output\_folder$)
    Create output directories: $"output\_folder/True", "output\_folder/False"$

    $csv\_training \leftarrow read\_CSV(train\_CSV)$                      ▷ CalibrationSet.csv
    $csv\_test \leftarrow read\_CSV(test\_CSV)$                          ▷ TestSet.csv
    $csv\_concatenated \leftarrow concat(csv\_training, csv\_test)$    ▷ Both CSV files contain similar information about different patients.

    for $row$ in $csv\_concatenated$ do
        $patient\_id \leftarrow row["ScanNumber"]$
        $slice\_number \leftarrow row["NoduleCenterImage"]$             ▷ Value in $[1, \infty]$
        $finding\_id \leftarrow row["NoduleNumber"]$
        $clinical\_significance \leftarrow row["Diagnosis"]$

        if $clinical\_significance == "malignant"$ or $"Primary\ lung\ cancer"$ then
            $clinical\_significance \leftarrow True$
        else if $clinical\_significance == "benign"$ or $"Benign\ nodule"$ then
            $clinical\_significance \leftarrow False$
        else if $"clinical\_significance == "Suspicious\ malignant\ nodule"$ then
            Continue

        for $visit\ in\ patient\_id$'s folder do
            for $mri\_type$ in $visit$ do
                for $dicom\_file$ in $mri\_type$ do
                    if $slice\_number == dicom\_file.InstanceNumber$ then
                        $slice \leftarrow normalize\_dicom(dicom\_file)$    ▷ see section 1.1.3
                        Save $slice$ in $"output\_folder/clinical\_significance"$
```

1.7 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.

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

```
procedure CREATE_PNGS($dataset\_folder, output\_folder, split$)
    Create output directories: $"output\_folder/[Train|Val]/[0|1]"$

    $true\_nparrays\_dict \leftarrow \{"patient\_id" : [file\_name\_1, ...]\}$
    $number\_of\_patients\_true \leftarrow len(true\_nparrays\_dict)$
    $number\_of\_training\_patients\_true \leftarrow \lfloor number\_of\_patients\_true * split \rfloor$
    $false\_nparrays\_dict \leftarrow \{"patient\_id" : [file\_name\_1, ...]\}$

    $index \leftarrow 0$
    for $patient\_id, file\_names$ in $true\_nparrays\_dict$ do
        if $index < number\_of\_training\_patients\_true$ then    ▷ Training set, True
            for $file\_name$ in $file\_names$ do
                $image\_array \leftarrow load(file\_name)$
                $image \leftarrow convertArrayToGrayscaleImage(image\_array)$
                Save image to $"output\_folder/Train/1"$
        else    ▷ Validation set, True
            for $file\_name$ in $file\_names$ do
                $image\_array \leftarrow load(file\_name)$
                $image \leftarrow convertArrayToGrayscaleImage(image\_array)$
                Save image to $"output\_folder/Val/1"$
        $index \leftarrow index + 1$

    $number\_of\_training\_patients\_false \leftarrow \lfloor number\_of\_patients\_false * split \rfloor$
    $false\_nparrays\_dict \leftarrow \{"patient\_id" : [file\_name\_1, ...]\}$
    $number\_of\_patients\_false \leftarrow len(false\_nparrays\_dict)$
    $false\_nparrays\_dict \leftarrow \{"patient\_id" : [file\_name\_1, ...]\}$

    $index \leftarrow 0$
    for $patient\_id, file\_names$ in $false\_nparrays\_dict$ do
        if $index < number\_of\_training\_patients\_false$ then    ▷ Training set, False
            for $file\_name$ in $file\_names$ do
                $image\_array \leftarrow load(file\_name)$
                $image \leftarrow convertArrayToGrayscaleImage(image\_array)$
                Save image to $"output\_folder/Train/0"$
        else    ▷ Validation set, False
            for $file\_name$ in $file\_names$ do
                $image\_array \leftarrow load(file\_name)$
                $image \leftarrow convertArrayToGrayscaleImage(image\_array)$
                Save image to $"output\_folder/Val/0"$
        $index \leftarrow index + 1$
```