UNIVERSITY OF CAMBRIDGE

Data Science in Medical Imaging

MPhil in Data Intensive Science

Thursday 15th February 2024 (3:00 pm to 4:00 pm)

Minor 2: Data Science Applications to Medical Imaging
- Problem Sheet 2: Medical Imaging Data Formats
Dr L. Escudero Sánchez

In this practical session you will gain experience in understanding and using DICOM, the standard data format conventionally used for medical imaging, and how to convert it to a numpy array.

*Attempt **all** questions.*

We will work through the questions during the practical session on the date indicated above.

Solutions to these questions will be available on Moodle after such practical session.

[Prep work before the session]

- Make a local copy of the dataset shared in the following GitHub repository: https://github.com/loressa/DataScience_MPhill_practicals.git. This dataset corresponds to publicly available images from the Lung CT Segmentation Challenge 2017 (LCTSC) as detailed in the LICENSE.txt. By accessing the data you agree to the policies described in the LICENSE.txt, as discussed in the lecture. You will need a local copy as you will be modifying the files.
- In your python environment, install the package `pydicom`. [*Hint: You can use Google Colab and use pip to install* `pip install pydicom`*. See the official pydicom documentation (*`https://pydicom.github.io/pydicom/stable/tutorials/installation.html`*) for installation options.*]
- Make sure you also have installed `matplotlib` or any other library for plotting that you prefer, as well as `skimage`.

```
    git clone https://github.com/loressa/DataScience_MPhill_practicals.git
    pip install pydicom
```

---

1 - We will start having a look at the dataset. For each patient (case) in the folder Images, open one DICOM file using the `pydicom` library and print out its metadata. [*Hint: look at the* `pydicom` *official documentation:* `https://pydicom.github.io/pydicom/stable/index.html`]

```python
import pydicom
from pydicom import dcmread
import os
from os import listdir
from os.path import isfile, join

# Input path and files
dataset_path = 'Dataset/Images'

# QUESTION 1
# Use for example os.walk to get all cases and files:
for dirpath, subdirs, filenames in os.walk(dataset_path):
    for subdir in subdirs:
        case_id = subdir.split('_')[-1]
        print('***** Case ID = %s *****' %case_id)
        subdir_path = join(dataset_path, subdir)
        list_files = [file_name for file_name in listdir(subdir_path)]
        dicom_file = join(subdir_path,list_files[0]) #choose any file
        print(dicom_file)
        metadata = dcmread(dicom_file)
        print(metadata)
```

2 - Three of the available cases have not been completely de-identified (Note that the patient information is not real, can you guess which fictional characters they are?). Find out which cases need further de-identification. Which DICOM tags need to still be de-identified?

Cases: 003 (PatientName, PatientID and PatientsBirthDate), 006 (PatientsBirthDate and PatientsBirthTime) and 007 (PatientName and PatientID)

3 - De-identify the cases discovered in question 2. You will need to do something different depending on the tag to be de-identified, why? Don't forget to save the new files after manipulating them. Remember you will have to do it for all the files (slices) of the scans, so write your code to do this in batch. Once you have completed this step, check that your files are correctly de-identified (you can use what you did for question 1)

PatientID, PatientName and PatientsBirthDate are type 2 tags, so they are required. Instead, PatientsBirthTime is a type 3 tag, meaning it's optional and can be removed entirely. Generally, tags of type 2 need to remain but they can be empty; this is reasonable for PatientsBirthDate but normally a de-identification (such as a trial ID) is used for PatientID and/or PatientName, to ease the handling of multiple patients.
It is a good practice when manipulating metadata tags to check the DICOM standard. In particular, have a look at the Confidentiality Profiles in Table E.1-1 in PS3.15 that we saw during lecture 6
`https://dicom.nema.org/medical/dicom/current/output/html/part15.html`. In the table you can see that the change expected for PatientsBirthTime is X (remove) while it is Z (empty or zero-length value) for PatientsBirthDate.

```
import pydicom
from pydicom import dcmread
import os
from os import listdir
from os.path import isfile, join

# Input path and files
dataset_path = 'Dataset/Images'
case_ids = ['Case_003', 'Case_006', 'Case_007']
```

(TURN OVER

```python
for case_id in case_ids:
    subdir_path = join(dataset_path, case_id)
    for file_name in listdir(subdir_path):
        dicom_file = join(subdir_path,file_name)
        metadata = dcmread(dicom_file)

        # Modify the tags that contain patient information
        metadata['PatientID'].value = case_id
        metadata['PatientName'].value = case_id
        metadata['PatientBirthDate'].value = ''
        # PatientBirthTime is optional, it might not be present
        try:
            del metadata['PatientBirthTime']
        except:
            pass
        # Don't forget to save the changes
        metadata.save_as(dicom_file)
```

---

4 - Find some tags that might be important in a data analysis, and compare them for multiple patients. This can be patient-based (e.g. gender or PatientSex) or acquisition-based (e.g. scanner manufacturer or model).

Some examples:

- Patient's gender/sex: there is a good balance between Male (6 cases) and Female (6 cases).
- Acquisition: all cases are obtained with scanners by SIEMENS manufacturer, all but one with the same model (Sensation Open).
- Acquisition: a constant voltage of 120 KVP was used for all scans.

---

5 - Choose one DICOM file (one 2D slice) and access the pixel values stored in the DICOM tag PixelData. What would you be getting if you use the information stored in PixelData directly? [*Hint: use the* `pixel_array` *handling method of* `pydicom` *that converts the PixelData to a* `numpy array`*.*] Visualise the image using `matplotlib`. What are the dimensions of your image?

By default the pixel data is usually read as raw bytes, which can be encoded with different data types (e.g. int, unsigned int, float) and/or compression. Therefore, PixelData is not directly useful, it needs to be interpreted. That's what the method `pixel_array` does for us.

```
import matplotlib.pyplot as plt
import pydicom
from pydicom import dcmread
file_name = 'Dataset/Images/Case_XXX/YYY.dcm'
metadata = dcmread(file_name)
image_array = metadata.pixel_array
print(image_array.shape)
imgplot = plt.imshow(image_array, cmap='gray')
plt.show()
```

The image size should be 512x512.

---

6 - Visualise different 2D slices. Try to identify different structures (e.g. bones vs organs vs air). What are the typical pixel values of those structures and why?

This is CT imaging, so the pixel values represent radiodensity and we expect bones to have large values, air to have small values, and other tissue (such as organs etc) to have values in between the two. In terms of Hounsfield Units (H.U.), air is expected to have large negative values (-1000) and bone to have large positive values (up to 1000). However, in these images, air appears to have a value of 24. If we look in more detail through the metadata, we find two tags at the end called RescaleIntercept and RescaleSlope. They define a linear transformation that allows us to convert the pixel values to HU (by default, or other units as specified). In our case, the slope for the rescaling is 1 and the intercept is -1024, describing a change:

$$v_{HU} = v * slope + intercept$$

that will bring the air values to -1000 as expected.

---

7 - Flatten (convert to 1D) the numpy array with the pixel values that you obtained in question 5 and build a histogram with the pixel values. Rotate the image 45 degrees and build the histogram again. After rotating, window the image (for example, set any pixels with value <100 to 100 and any pixels with value >1000 to 1000) and repeat the histogram. Do you see any differences in the histograms?

Example code for this question. First part, flatten and building a histogram of the original image:

```
import pydicom
from pydicom import dcmread
import numpy as np
import matplotlib.pyplot as plt
```

(TURN OVER

```python
file_name = 'Dataset/Images/Case_000/1-001.dcm' # Choose any file
metadata = dcmread(file_name)
image_array = metadata.pixel_array

flat_array = image_array.flatten()

nbins = 100
hist, edges = np.histogram(flat_array, nbins)
plt.hist(flat_array, edges)
plt.yscale('log') #Optional but improves visualisation
plt.show()
```

The rotation is a global transformation of the image. The value of the new rotated pixels will have to be interpolated, and when empty spaces appear due to the rotation, those pixels will be typically filled with 0 (black regions). The interpolation in `skimage` will create float values. Since our original image is type *uint16*, we can first convert it to float to avoid the normalisation that it will otherwise be done to convert to float during the rotation (dividing by 2^{16}, total number of grey levels for *uint16*).

```python
import pydicom
from pydicom import dcmread
import numpy as np
import matplotlib.pyplot as plt
from skimage.transform import rotate

file_name = 'Dataset/Images/Case_000/1-001.dcm' # Choose any file
metadata = dcmread(file_name)
image_array = metadata.pixel_array

image_float = image_array.astype(np.float64)
rotated_float = rotate(image_float, angle=45)
flat_rotated_float = rotated_float.flatten()

nbins = 100
hist, edges = np.histogram(flat_rotated_float, nbins)
plt.hist(flat_rotated_float, edges)
plt.yscale('log') #Optional but improves visualisation
plt.show()
```

With this example 2D slice, the histograms look like in Fig 1, original on the left and rotated on the right. We can see that the distribution is very similar, but values appear at 0 due to the additional black spaces appearing after the rotation. You can also check the min and max values of both arrays.

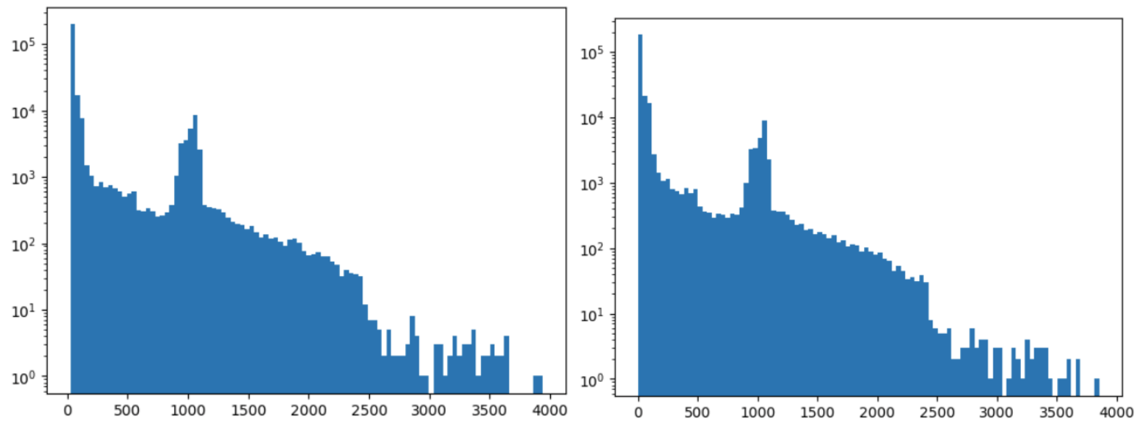

Figure 1: Histogram for the original (left) and rotated (right) images

An alternative method we can use is with OpenCV, which is a very popular and useful computer vision package for python. The official documentation can be found in `https://opencv.org`.An example doing the same rotation with OpenCV would be as follows:

```python
import pydicom
from pydicom import dcmread
import numpy as np
import matplotlib.pyplot as plt
import cv2

file_name = 'Dataset/Images/Case_000/1-001.dcm' # Choose any file
metadata = dcmread(file_name)
image_array = metadata.pixel_array

# For OpenCV we need to define the centre for the rotation
centre = (255,255)
R = cv2.getRotationMatrix2D(centre,45,1)
image_rotated = cv2.warpAffine(image_array,R,(512,512))
flat_rotated = image_rotated.flatten()

nbins = 100
hist, edges = np.histogram(flat_rotated_float, nbins)
plt.hist(flat_rotated_float, edges)
plt.yscale('log') #Optional but improves visualisation
plt.show()
```

In either case, the rotated image should look like Fig. 2 (to visualise, plot the 2D arrays).

(TURN OVER

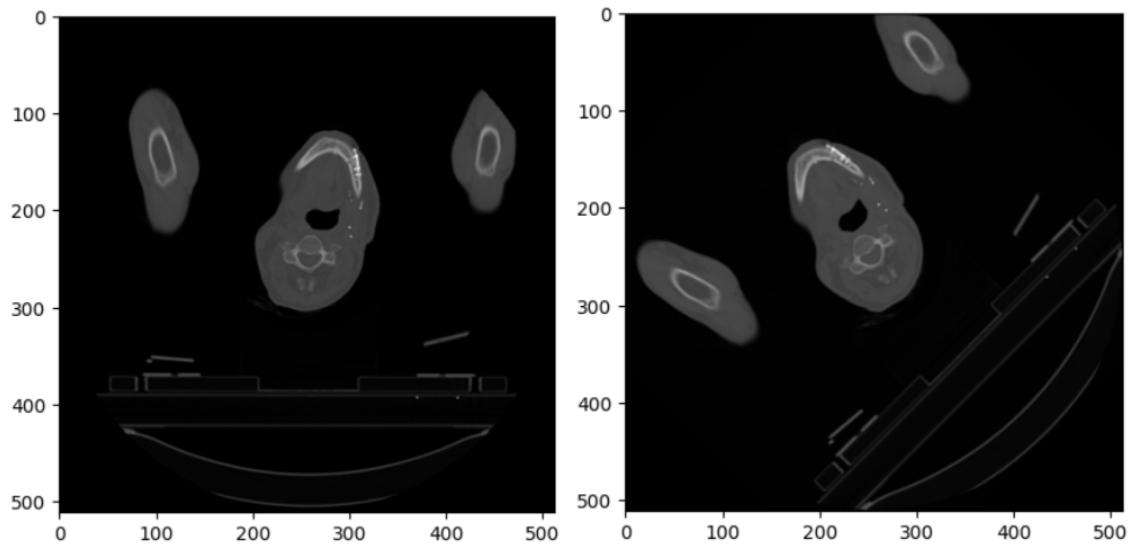

Figure 2: Example 2D slice before (left) and after (right) rotating by 45 degrees.

When windowing, we restrict the range of pixel values and artificially move the peak on the left to 100 and create another peak on the right side at 1000, as shown in Fig 3.

```
import pydicom
from pydicom import dcmread
import numpy as np
import matplotlib.pyplot as plt
from skimage.transform import rotate

file_name = 'Dataset/Images/Case_000/1-001.dcm' # Choose any file
metadata = dcmread(file_name)
image_array = metadata.pixel_array

image_float = image_array.astype(np.float64)
rotated_float = rotate(image_float, angle=45)
flat_rotated_float = rotated_float.flatten()

##### Windowing #####
flat_rotated_float[flat_rotated_float > 1000] = 1000
flat_rotated_float[flat_rotated_float < 100] = 100

nbins = 100
hist, edges = np.histogram(flat_rotated_float, nbins)
plt.hist(flat_rotated_float, edges)
plt.yscale('log') #Optional but improves visualisation
plt.show()
```

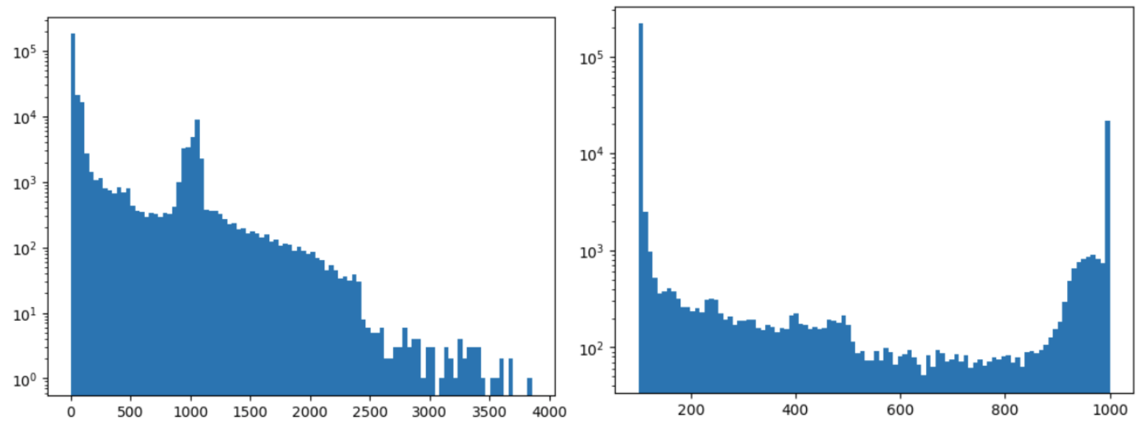

Figure 3: Histogram for the rotated image before (left) and after (right) windowing.

[BONUS] - What does the DICOM tag "SOP Class UID" mean? Which type of medical imaging corresponds to the following SOP Class UIDs?

1. 1.2.840.10008.5.1.4.1.1.2
2. 1.2.840.10008.5.1.4.1.1.4
3. 1.2.840.10008.5.1.4.1.1.6.1

Service-Object Pair or SOP defines the attributes of the Information Object Definition or IOD together with a DICOM Service Elements or DIMSE. It means it contains the rules that define the imaging object in a specific service. For example, Service Elements might be Store, Get, Move, etc. And IODs can be CT images, MR images, but also raw formats, wavelenghts and even other sort of files, such as reports etc. SOP Class UID is the map of unique identifiers for each SOP Class, and it is a mandatory tag in the metadata of every DICOM file.
From `https://dicom.nema.org/dicom/2013/output/chtml/part04/sect_B.5.html`

(a) CT Image Storage

(b) MRI Image Storage

(c) Ultrasound Image Storage

(TURN OVER