Subtle Medical Coding Challenge

Subtle Medical, Inc.

September, 2018

Introduction

Congrats on making to this step of the interview process. This coding challenge is to get a sense of your technical abilities, including how you think and react to problems and if you can write quality code in a setting that is close to Subtle's. Please complete as much as of the coding challenge as you can, but it is better to answer part of it well than all of it poorly. Should you have any questions, please feel free to reach out.

This challenge consists of three tasks. You will start with writing command line tools to deal with DICOM I/O. Then, you will write a small mock algorithm module that simulates fast medical imaging acquisitions. Finally, you will build, train and test a deep learning model to perform image enhancement on the simulated data.

Please note that the last task will require the use of GPU. If you do not have adequate personal computing resource, you are welcome to use our AWS instance (`p2.xlarge`, IP address: 34.210.181.179) to complete this task. If you have not used AWS before, you can refer to `http://cs231n.github.io/aws-tutorial/` or other tutorials available online. If you choose to use our AWS instance, please use the PEM file (provided in the coding challenge email) to connect (`>> ssh -i SubtleCodingChallengeDLS.pem ubuntu@34.210.181.179`).

General Requirements

- The whole challenge should be completed in Python 3.5+.
- Questions, answers and images should be summarized in a write-up.
- Zip both the write-up and all your code into one package and send it to us via email. Please do not include the downloaded datasets in the package.
- Code should be written in a modular way so that unit testing and refactoring are relatively easy to implement.
- Code should be sufficiently commented.

Task I: DICOM I/O

DICOM, or Digital Imaging and Communication in Medicine, is the standard of medical image storage and communication. Most medical imaging data we deal with at Subtle are stored in the DICOM format. You can use the `pydicom` library for DICOM input and output. If you are not familiar with DICOM, documentation and examples about DICOM libraries can be easily found online, e.g., `https://pydicom.github.io/pydicom/stable/getting_started.html`.

Let us begin by writing a script that converts DICOM files to hdf5 files, which are easier to handle during model training. The goal is to develop a command line interface that supports the following arguments:

- `--input-dicom, --i  path to input DICOM directory`
- `--output-hdf5, --h  path to output hdf5 file`

Your first script should do the following things:

1. Read all the DICOM files in a directory
2. Build a 3D volume (3D `numpy` array) by sorting against each file's Slice Location DICOM tag in ascending order
3. Converting the data from the input data type to 32-bit float data type and normalize the 3D volume to the range between 0.0 and 1.0
4. Export the normalized 3D volume (3D `numpy` array) into a hdf5 file

The sample data can be found at `http://old.mridata.org/fullysampled/knees`. We will use the .zip file from the first case (the "Dicoms" column, 67 MB per case) for this part of the coding challenge.

After the above script is done, let us move on to the the second script that reverses the process (more or less). Your second script will take in the following arguments:

- `--input-dicom, --i` `path to input DICOM directory`
- `--input-hdf5, --h` `path to input hdf5 file`
- `--output-dicom, --o` `path to output DICOM directory`

Your second script should do the following things:

1. Read the DICOM files in the directory as a template
2. Build a 3D volume (3D `numpy` array) by sorting against each file's Slice Location DICOM tag in ascending order
3. Read the 3D volume stored in the hdf5 file and rescale this 3D volume to match the dynamic range and data type (check `pydicom.dataset.pixel_array.dtype`) of the 3D volume read in Step 2
4. Replace the 3D volume from Step 2 with the new one from Step 3 and save the results into new DICOM files into the output DICOM directory

In this task, the new DICOM files (after running your first and second script) and the original ones should be identical.

Task II: Simulating Fast Acquisitions

Ideally, clinical medical images should have good image quality, including sufficient signal to noise ratio (SNR), structural delineation, etc. However, many clinical applications require long data acquisition times (up to minutes or longer). Shortening the scan time can make the acquisition less sensitive to imaging artifacts, but with the cost of reduced SNR or image resolution. In this task, you are asked to write a simple function to simulate fast data acquisitions with reduced image resolution. Your function takes the 3D volume from Task I and generates a new 3D volume with the same matrix size but reduced image resolution using a Gaussian blurring filter (refer to `scipy.ndimage.gaussian_filter`). For simplicity, please apply the same 2D blurring filter (with a specified `sigma`) to the 2D images at each slice.

```
import numpy as np
def blurring3d(input3d: np.ndarray, sigma: float) -> np.ndarray:
```

Call this function to generate a blurry 3D volume with `sigma` = 5. Use the second script you just wrote in Task I to save blurry images into DICOM images. Now you have a pair of datasets with good image quality (original images) and poor image quality (blurry images).

Choose the central slice of the 3D volume and show the resulted image side by side along with the original image (without image blurring) in your write-up. Display your images in grayscale with proper window width and level so that the underlying anatomy is not too bright or too dark.

Task III: Super Resolution Using Deep Learning

Now let us implement a deep learning model to perform image enhancement. This task is open-ended. Feel free to choose your favorite model and implement it in TensorFlow, Keras, Caffe, or PyTorch. Your model basically takes a blurry 3D volume from Task II as the input and predicts an outcome that is as close as to the original 3D volume. GAN is recommended for this task.

Data

We will continue to use the datasets on `http://old.mridata.org/fullysampled/knees` for this task. Download all 20 knee cases from this website. For each case, generate a 3D volume (Task I) and a corresponding blurry volume with `sigma` = 5 (Task II). The first 16 cases will be used for training and cross-validation. The last 4 cases should ONLY be used for testing. (Update: the link to case 12 is broken, so you have 15 cases for training and cross-validation.)

Considerations

There are several considerations we recommend you to make. Please include them in the write-up.

- **Choice of model** You should choose your model based on the available data and computation time. Deep 3D models may not be good ideas if training data are limited. If necessary, you should consider data augmentation approaches to increase the training sets and improve the robustness of your model.
- **Loss functions** Justify your choice of loss functions for the given task and possible improvements. Include in the write-up a plot of your objective function vs. number of epochs. What are the stopping criteria?
- **Hyperparameters** Summarize the hyperparameters (regularization, learning rate, etc) of your model and how to choose them.

Once you have finished training your model on the first 16 cases, run inference tests on the remaining 4 cases. Again, choose the central slice of the 3D volume from Case 17 and show side by side the blurry image, your DL result and the ground-truth (original image) in your write-up. Define some numerical metrics to evaluate quantitatively how close is your predicted outcome compared to the ground truth.