Assignment 2 Report CAP 5516

Michael Fielder
University of Central Florida
College of Engineering and Computer Science
Department of Computer Science
MichaelFielder@knights.ucf.edu

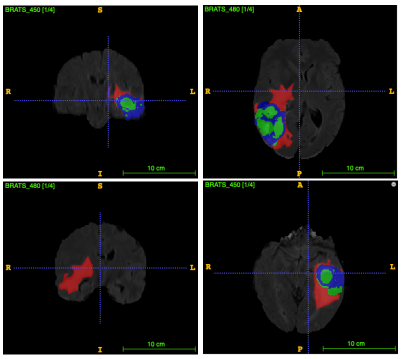

Figure 1. Examples of ITK-SNAP Brain MRI Segmentation

1. Introduction

This assignment involved training a U-Net model to segment areas of a brain MRI that contain signs of a tumor. Below is the link to access my Google Colab code.

Google Colab Link: https://bit.ly/3vl2mye

2. ITK-SNAP Examples

ITK-SNAP is a software used for segmenting and viewing existing segmentation of various medical imaging types. For this assignment, we are working with brain segmentation MRI's in the `.nii` filetype. **Figure 1** shows fours examples of brain MRI's with their accompanying brain tumor segmentation.

3. Code Details

In this section, I explain the details of my training. I tried three different U-Net models but was not able to achieve any results with either. Nevertheless, the details of my experimentation are shown below.

3.1. Nii File Loading

The files for this assignment are stored in the NIfTI format which need to be read differently than a simple image file format like a `.jpeg`. Python provides a library for handling this file type called NiBabel. The way it works is that it is loaded as a NifTil image which is handled can then be viewed or interacted with. This type cannot be handled as input to model, so it needs to be converted into a NumPy array. The problem I encountered was that the provided `get_fdata()` function stores the result in the cache which eventually crashes the program due to it running out of memory. The solution is to directly store the MRI scan directly as a NumPy array which avoids it getting stored in the cache.

3.2. Custom Dataset Creation

In order to store the data correctly, a custom dataset needed to be created in order to store an access the MRI slices. For this assignment, I created a custom dataset using the Dataset class in PyTorch. The images and labels provided are stored as NifTil wrappers. When a slice needs to be accessed, it calls the `__getitem__()` function which takes the full 3D scan, converts it into a NumPy array, and only returns the slice that was requested. Another thing to note is that the image output is normalized to be between 0 and 1 since there is such a wide range of values. There are 75020 deconstructed individual 2D slices in the dataset.

3.3. DataLoader Creation

Since there are over 70k slices in the dataset, I opted to use a subsample of 1000 slices to use for the training set. In order to do this, uniformally sampled without replacement 1000 different samples which is passed into the testloader. This made it possible to perform the training in any reasonable amount of time.

3.4. UNet Model Details

I experimented with three different UNet architectures. The first one I used was "U-NET FOR BRAIN MRI" by mateuszbuda found on PyTorch's website [1]. This is a U-Net model specifically designed to be used for brain segmentation, but I don't think it was set up for this specific

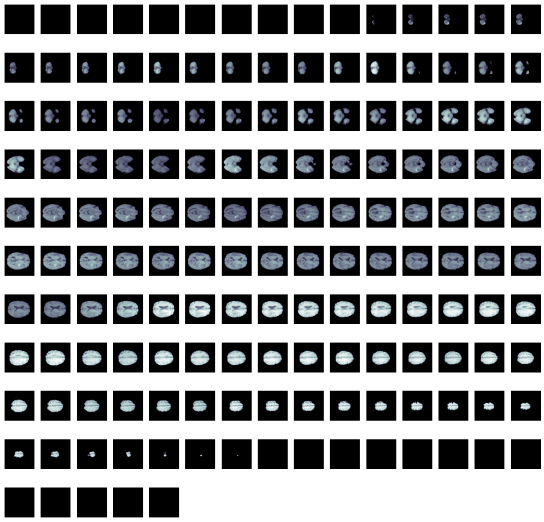

Figure 2. An example slices in the dataset. These are all the slices for one MRI sample

task. The other U-Net architecture I used was the one found in the Segmentation Models for Pytorch Library [2]. This model is pre-trained on ImageNet and uses ResNet for the encoder. The final model I trained with which had only a modicum of success was [3]. Something to also note is that all of these models have 4 `in_channels` and 4 `out_channels` which corresponds to the four classed of segmentation.

3.5. Training

For each of the two models above I used Cross-Entropy for the loss function and the Adam Optimizer. For Adam, I tried using a learning rate of 1e-3 to 1e-6 to little success.

4. Issues with Code

For some reason, I was not able to successfully train either of the first two models above. I tested the model [2] before any training and it returns the logits correctly. I then use the argmax function to find out which class has the highest probability per pixel. This means that I am correctly formatting the output.

The biggest problem that arises during training the second model is that the loss hovers around the same value of 0.7568 without changing. I believe that this is indicative of overfitting but was not able to achieve different results no matter what I tried.

The final model had similar problems with loss, but I was able to get some semblance of a segmentation from it. If I had more time I might have been able to solve the issue, but that isn't guaranteed.

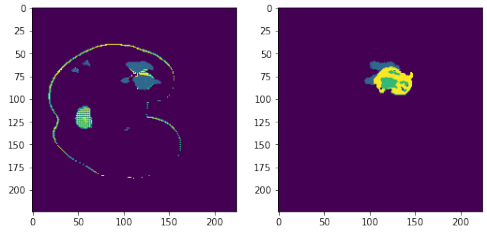

Figure 3. An Example of the output produced by [3]. As you can see, it somewhat detects the tumor, but not very well.

5. Conclusion

Overall, I was able to setup the dataset and thought I trained the model correctly. Unfortunately, I wasn't able to fix what I believe was a loss issue. Hopefully, I can present a better final project.

6. Sources

1. https://bit.ly/37oaqGr
2. https://bit.ly/3rtWnWS
3. https://bit.ly/3JZXhRm