DST Reflection Data At Scale

Tom Blain

1 Introduction

For this project we selected a brain tumor MRI dataset. The task was "Data at scale" which naturally lends itself to image classification and a neural network pipeline which enables unrivalled parallelism and scalability options. CNNs are easily the most widely used class of machine learning algorithm for images, since they incorporate "convolutional layers" which is where a neuron recieves input from a restricted area of the previous layer, such as how we might look at an image and the local areas that make up an image rather than the whole thing at once.

My specific role in this project was to write a highly scalable solution.

2 Scalability

2.1 What does this mean?

Brain MRIs are standardised around the world. If we think about a real world model therefore, especially with the critical nature of the context, we would always want to select the highest performing and most well tested model. It is therefore important to make sure the model can handle a huge amount of data (for both training and rigorous testing), and this problem introduces lots of scalability issues such as how we can make our code effectively run on multiple GPUs, how we can allow the model to handle new data.

2.2 Scalable Data Processing

In this project, we had to create a fully new pipeline for the data handling to enable a new influx of data at any time, and an effective way to process a dataset thousands of times bigger in magnitude. A key idea is for the dataset to be processed in mini batches, as this would theoretically allow as many machines as we want to process these images in parallel. Full batch processing as we implemented in 03-preprocessing suffers from issues when you introduce new data, since it is not so obvious how to handle it. Mini batch processing here easily lets us concatenate the processed batches together rather than having to run the entire dataset through if we added more data.

An advanced idea that we did not look into in this section would be to process the data and run our training epochs in parallel. This could be achieved since we use batches in preprocessing and batches in training, so one machine could be processing the data while another can be using a GPU for training the algorithm on the mini batches that were just processed. This might be necessary if we were to recieve a constant stream of data, but I doubt this would be the case in our real world MRI scan application, where there is a large amount of data we could collect but it isnt going to be rapidly coming in 24/7.

2.3 Parallelism in Training

To train effectively if we scale this system up to a large amount of data we focused on implementing dataloaders which contain batches of data, allowing efficient memory use since we never run into memory issues. The batch size can be carefully selected depending on computational power available and quantity of data available, as I focused in researching in my Portfolio B. With this in place, we can then parallise the algorithm by running these batches on different GPUs and taking an appropriate gradient step and parameter update. How to process batches independently and take the optimum step is not something we looked into, and left this instead to inbuilt functionality in PyTorch to implement. PyTorch is often used for deep learning and hugely scalable algorithms, so felt satisfied that the in built handling of parallel optimisations would be more fine tuned and optimised, perhaps even on a low level, than anything manually coded in Python.

A further consideration was early stopping criteria. With training on huge sets of data it is very difficult to choose an appropriate number of epochs to balance overfitting and underfitting, so early stopping conditions monitor the validation loss and stop before we are overfitting our training loss with a loss in accuracy in our validation loss. This means, as long as the training is going well and the validation loss is similar to the training loss, we are likely to avoid this.

Finally, a huge advantage to using a neural network for our model is the ability for model checkpointing. We can save our learnt weights and biases at any point during training so we could come back to these later if we have more data. Relating the problem back to our context, we are likely to receive more labelled data in the future (as more brain tumors are confirmed and MRIs are taken), so we have the ability to reload these parameters and continue training the model.

With all of these considerations for scalable and parallism in place, we are confident this creates an efficient pipeline that would have the potential to handle any scalability problems we thought of in the context of medical imaging.

3 Reflection

This project was challenging and a great learning experience. I am glad it was an opportunity to apply a wide range of concepts we have learnt throughout the year, and see the improvements that I have personally made to my knowledge as a data scientist.

3.1 What went well

The resultant scalable model had impressive predictive power, very fast runtime, no memory issues and consistent performance. I could not see how to improve this further. I am glad to have used a large amount of PyTorch in my research project alongside personal projects that allowed me to understand how to implement a PyTorch solution that would run smoothly.

3.2 What we could have done better

I felt a disconnect to my group members throughout the project. We met up often and would have active discussions together to solve the problems we came across, but there was always serious problems with the replicability of the code in other sections limiting how much I could actually use the other sections. This was because of infrequent pushes to github and confusing file handling systems, with data stored elsewhere than the project folders. There was also a wide variety of python packages used throughout the project, which felt inconsistent.

Bluecrystal didn't work for us, but we fell into the trap of "my code is horribly inefficient, instead of making it efficient, I need to run it on a supercomputer". This ultimately was a learning experience and helped me to write my scalable section because I realised the importance of using the pytorch data structures and pytorch vision package where possible, since this is obviously the format a pytorch neural network prefers and is optimised for.

Our dataset was the main cause of issues throughout the project. It did not accurately represent a real world dataset, since all images were taken at different orientations and angles of the brain, which is unrealistic for the real world where you would have a few dozen images of the same individual which forms a 3d image of the brain. The neural network would then likely take in the full batch of the patients images for even better results, or make predictions on the whole set of patient images to then make a probabilistic statement available for the doctors on whether to investigate further. The inconsistency in angle of images was unresolved mostly, but it just wouldn't be a factor in a real world application.

3.3 Final Remarks

This was a difficult project we were likely unprepared for, but that's definitely how you improve your knowledge and skill. I am very glad to have gotten more experience now with PyTorch and using pretrained models. It also stresses the importance of carefully considering a dataset to use, and spending some more time here will likely save you a long time later!