Summer Student Project Outline

Objective:

The combination magnetic resonance (MR) scanner and linear accelerator allows for radiation therapy (RT) to be performed under real-time MR visualization, “MR-guided RT”. The benefit of MRgRT is the ability to visualise daily anatomical changes over the course of RT and then adapt the treatment plan during each treatment session. However, imaging artifacts which can arise from several sources, metal implants, patient motion, etc. may make acquired images unsuitable for plan adaptation. The clinical MRgRT workflow would benefit from a way to flag images contaminated with artifacts and identify in real-time when images need to be reacquired or a fall back to the original treatment plan is necessary.

Rough List of Steps for the first few weeks

1. Familiarize yourself with PyTorch. A free textbook is available on the PyTorch site, here :

<https://pytorch.org/assets/deep-learning/Deep-Learning-with-PyTorch.pdf>

This text is written very clearly and gives a high-level overview of the PyTorch library and the basics about how neural networks work. After reading the first half you should have no trouble implementing simple/standard deep learning models in PyTorch.

1. Look at the Gitlab repository I have prepared for you on Spinecho, which contains an image classifier and 2 datasets to try out. The first is simply photos of bees and ants, the second is images from the BRATS18 dataset (<https://www.med.upenn.edu/sbia/brats2018/data.html>) containing slices of brain MRIs sorted into containing tumor vs no tumor. I have tried to make most of the adjustable options very clear e.g model architecture, epoch number, batch size, etc. Try and run this code and use what you have learned from the text to understand what is going on. In most cases I have mentioned in the comments which section of the textbook is relevant.
2. Try to prepare some results. Explore what seems interesting to you, some example questions to answer to check your understanding are as follows:

Note: these are just examples, do whatever seems interesting to you!

* Describe the datasets we are using (there are technically three, ImageNet for pretraining, the Hymenoptera dataset and the BRATS18 dataset for fine-tuning and testing). How big are the datasets? What types of images are in them? Size of images? How are they being preprocessed? Are the classes balanced? Are Samples IID? etc.
* Play around with different options (hyperparameters) at the beginning of main.py e.g batch size, number of epochs, loss function, different models other than the vgg. Compare classification accuracy/final ROC plot.
* Are there other evaluation metrics you can try?
* Find your own dataset (on kaggle or just on the internet somewhere) and try it out. How does it compare to the previous datasets? Did hyperparameter choices that did better on the other datasets perform the best on your new dataset?
* I included the option to train from scratch vs pretrain on imagenet then “fine tune” the network with our additional data. Which is better for each dataset? Why do you think that is?
* In this case, I've used the pretrained network as a feature extractor, i.e the only weights that change are those in the final classification layer, the convolutional layers stay the same. Can you edit the optimizer so that more weights change? e.g update the full network, some number of middle layers, etc. What is the result on accuracy on the BRATS18 dataset?
* In the dataloader try different transforms. What is the effect of including or getting rid of different data augmentation techniques?
* Do you notice differences when using normal images and medical images? Can either be data preprocessing, or results. i.e is transfer learning on imagenet more or less effective on the brain images than bees vs ants?
* Currently the network is set to pretrain on imagenet, can you make it pretrain on the bee's dataset then apply to BRATS or another dataset? How does accuracy compare to pretraining on imagenet?
* What is the "saliency map" and what does it do?
* Usually when you build a network, you will have files called something like “train.py” and “test.py” which train/test. What I called “main.py” basically serves as “train.py”. It saves the training history and models weights, which need to be loaded to test. I also wrote the “check\_image” function to test a single image, but not the whole validation set. As a knowledge check, generalize the “check\_image” function to “test.py”. The steps are, make a new instance of the model, load in the previously trained weights, load the validation images using a dataloader, feed them into the model, save the output, and then generate ROC curves + whatever other metrics you want to use for evaluation.

1. Using data from Sunnybrook, update the network (or build your own from scratch, if you would prefer) to be able to classify if a given image has an artifact or not.
2. Explore the dataset. How and from where were the images acquired? Learn about the acquisition parameters. Are the prediction classes balanced? Do you foresee challenges unique to this dataset?
3. Prepare the dataset. Medical images have their own format, usually. nii or DICOM. You’ll need to build a dataloader, or preprocessing pipeline which can take these images and get them ready for input to the network. You will also need to determine some sort of data augmentation scheme.
4. Train the network using the dataset and perform some hyperparameter optimization to see if we can find optimal training conditions.
5. Add an additional dataset. Maybe a different type of artifact,e.g if you start on metal, maybe add motion artifacts. Or you can try a different anatomical site, brain vs spine for example (or even try to include non-medical images and do artifact vs no artifact vs bees). Is it better to keep classification binary, (artifact or no artifact) or specify the nature of the artifact (e.g clean image, metal, or motion)?

To get Pytorch running on Spinecho, first you’ll need to make a conda (or pip if you prefer) environment.

You can find more info on environments here : <https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html#activating-an-environment> but I’ll just list the steps

Start with the command:

Conda create -n “env\_name” python

Then to activate/deactivate the environment, use

Conda activate “env\_name” / conda deactivate

In your environment run

Conda install pytorch torchvision torchaudio cudatoolkit=11.1 -c pytorch -c conda-forge

Then you can check the pytorch version to make sure it installed correctly by going

**import** torch

print(torch.\_\_version\_\_)

If you use an IDE like pycharm or spyder you might need to make the default environment the one you just created instead of “base”.