Coding Challenge
MRI + Deep Learning

Dave Van Veen

February 27, 2019

1 Overview

This report summarizes my work for a 72-hour coding challenge of improving MRI reconstruction with deep learning techniques. The challenge consisted of three parts. First I started writing command line tools to handle various I/O tasks with DICOM, or Digital Imaging and Communication in Medicine, which is the standard format for medical imaging. Second I wrote a small mock algorithm module to simulate fast medical imaging acquisitions, i.e. blurring the original MRI scans. Lastly I built, trained, and tested a deep learning model to perform super resolution on the simulated data.

Table 1 provides an overview of my repository so that the reader can understand and implement my code. Note that data is not included in this submission, thus the user must acquire his or her own data from `http://old.mridata.org/fullysampled/knees`. Section 2 includes a discussion of various design decisions with regards to the model and optimization procedure. All code and results were written within the 72-hour time constraint; I suspect drastically improved performance could be attained by tuning hyperparameters, allowing the model to fully train, and leveraging three-dimensional information of neighboring MRI slices. This and other methods for improvement are discussed below.

2 Methods and Discussion - Deep Learning

This section contains a very brief overview of my methods for completing the third part of this challenge: building, training, and testing a deep learning model to enhance image quality via super resolution. I also display results, discuss various design trade-offs, and provide suggestions for improvement.

2.1 Model Choice

For the task of image super resolution, I chose to base my model upon the efficient subpixel convolutional neural network, or ESPCN [4]. This network builds off SRCNN [1], a

File Name	Task	Description
requirements.txt	N/A	System packages; run pip install -r requirements.txt
dcm_to_h5.py	I	Conversion of dicom files from directory to a single hdf5 file
h5_to_dcm.py	I	Conversion of single hdf5 file to individual dicom files
utils_io.py	I, II	Functions for performing data I/O
parser_io.py	I	Parser for command line input of data I/O tasks
configs_io.json	I	Default configurations for parser_io
blur.py	II	Gaussian blurring filter to each slice of 3D scan
model.py	III	Definition of neural network using PyTorch [3]
main.py	III	Main script for training and testing model
utils_model.py	III	Functions and class definitions for model
parser_model.py	III	Parser for command line input of model training parameters
configs_model.json	III	Default configurations for parser_model
prototype.ipynb	II, III	Code for plotting figure output

Table 1: Description of each file in the repository. Note the user must acquire his or her own data in order to run this code.

Figure 1: Task II Central slice of 3D volume (Case 1) for the original image (left) and the blurred image (right).

seminal algorithm for CNN super resolutions; however, ESPCN is significantly faster with comparable performance to SRCNN. Given the project's time constraint, computational cost was a driving factor for design decisions.

Other methods to reduce runtime include downsampling the network input images from (512,512) to (256,256), and then upsampling at the final layer. These input images are two-dimensional slices of the original three-dimensional MRI scans. Undoubtedly there is valuable information to be gained from neighboring slices; thus it would be ideal to use a

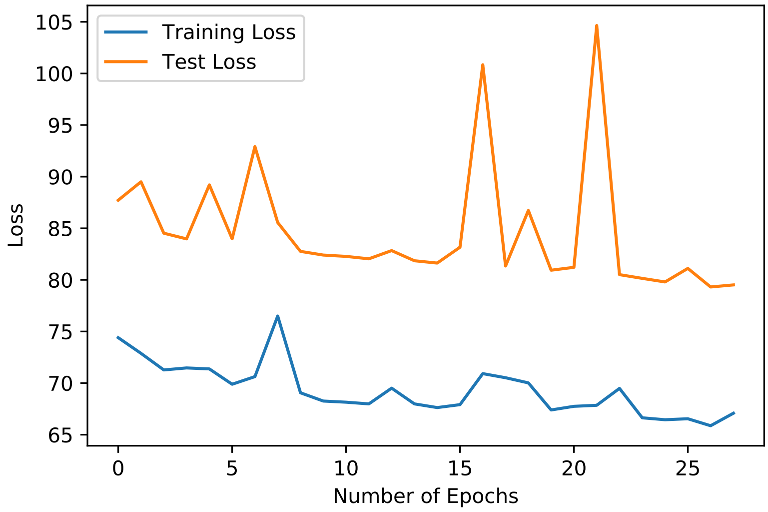

Figure 2: Task III Loss vs. Number of Epochs. Please see discussion in Section 2.2

3D CNN for this task. However, a three-dimensional model would have more parameters, which can introduce some additional problems.

The first problem is that the training process would require more time. With more parameters we want to use more training data, but our given dataset is fairly small. This could be addressed by performing data augmentation to increase the size of our training set. In general, common data augmentation techniques include flipping, rotating, or translating the original images. In this case, however, all the images are fairly homogeneous, i.e. the position and orientation of the knee within the frame is similar between MRI scans. Thus the first augmentation technique I would try is adding either Gaussian or salt-and-pepper noise.

Another issue with using a 3D CNN, or in general with using more parameters, is that the memory required could potentially increase beyond the capacity of a given system. One way to address this from the perspective of the model would be to implement some sort of patch-wise procedure, e.g. [2].

2.2 Loss Functions

In this model I default to using the standard ℓ_2 loss, or mean squared error between the original image and the predicted image. Often, but not always, this can be improved by using weight decay. Standard ℓ_2 is arguably the most common loss for image reconstruction, although it does not result in high image quality as perceived by humans [7]. Other metrics such as the structural similarity index SSIM [6] or its derivative MS-SSIM [5] have delivered state-of-the-art results with regards to perceptual quality. These are differentiable, which is a requirement for network backpropagation. Combining MS-SSIM with ℓ_1 loss has recently been shown to work exceptionally well results for super-resolution [8]. Clearly there are many possibilities for improvement with the loss function in this model, but I did not explore them.

Figure 2 contains both training and testing loss vs. number of epochs. Note that because the loss value after the first epoch was very large, it has hence been omitted from the graph to maintain a decipherable scale. Clearly there is more work to be done to determine the stopping criterion. On a high level, I would stop training at the point where test loss began to increase while training loss decreased, which would indicate overfitting.

2.3 Hyperparameters

While I did not perform an exhaustive hyperparameter search, there are many model adjustments that could potentially lead to improved results. One parameter I did search for was the learning rate, by analyzing the curve of Loss vs. Number of Epochs for a grid search over [0.1, 0.01, 0.001]. I found that a learning rate of 0.01 performed the best and kept that fixed moving forward.

Another hyperparameter that could be improved with regards to optimization include batch size. In addition it would be interesting to evaluate SGD with momentum and other optimizers; I chose Adam as it has commonly demonstrated strong performance. I also did not put much effort into adjusting the architecture of my network, which may also improve performance. I chose a larger receptive field for the first layer but afterward reduced kernel_size to 3 and padding to 1, which is common in CNNs when maintaining a constant spatial dimension between layers (assuming a stride of 1). I also could have tried different activation functions but chose ReLU as it is preferred in many recent applications.

Figure 3: Task III Central slice of 3D volume (Case 17) for the original image (left), the blurred image (middle), and the model result (right).

3 Conclusion and Future Work

In this report I summarize and discuss my results for the MRI + Deep Learning Coding Challenge in addition to providing a summary of each file in the repository. While the repository is fully functional, some of the code within it could be written more efficiently. As discussed in Section 2, I did not experiment very much with different model architectures, loss functions, or hyperparameters. I'm confident the model results could be significantly improved given time beyond the 72-hour limit.

References

[1] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):295–307, 2016.

[2] Chieh-Chi Kao, Yuxiang Wang, Jonathan Waltman, and Pradeep Sen. Patch-based image hallucination for super resolution with detail reconstruction from similar sample images. *arXiv preprint arXiv:1806.00874*, 2018.

[3] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[4] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1874–1883, 2016.

[5] Zhou Wang, Alan C Bovik, Hamid R Sheikh, Eero P Simoncelli, et al. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[6] Zhou Wang, Eero P Simoncelli, and Alan C Bovik. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, volume 2, pages 1398–1402. Ieee, 2003.

[7] Lin Zhang, Lei Zhang, Xuanqin Mou, and David Zhang. A comprehensive evaluation of full reference image quality assessment algorithms. In *2012 19th IEEE International Conference on Image Processing*, pages 1477–1480. IEEE, 2012.

[8] Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. Loss functions for image restoration with neural networks. *IEEE Transactions on Computational Imaging*, 3(1):47–57, 2017.