PET Image Reconstruction Using Deep Learning

Ishaan Pathak

Michigan State University

April 2022

Abstract

Neural Networks are a novel technology that have a variety of applications in a lot of fields. They are a very powerful modeling tool, especially in cases where a perfect mathematical model is difficult to establish between the dependent and independent variables. Although traditional methods of image reconstruction work well in many situations, the time required to complete the reconstruction remains quite high. The aim is to determine whether neural networks can allow for better reconstruction times and examine whether we also observe a change in the accuracy of the final reconstruction.

Background and Motivation

Traditional Tomographic reconstruction techniques require converting multiple projections into images. There are multiple methods for tomographic reconstruction, but they suffer from either being too slow or introducing noise and artifacts in the reconstruction process. Given that medical imaging scans such as PET and CT are performed for diagnostic purposes, both the speed and accuracy of reconstruction are significant factors which may have a direct impact on patient outcome. It is also quite important to mention that Filtered Back Projection is among one of the faster methods of Image Reconstruction, however it tends to produce less accurate results compared to algorithms like Ordered Subset Expectation Maximization (OSEM). So, the key question that this project is looking to answer is: Can deep learning methods provide a faster and more accurate reconstruction approach compared to traditional methods like FBP?

Methodology

There were a variety of methods used throughout this project but among some of the most important ones are Filtered Back Projection (FBP), Convolutional Neural Networks (CNNs), and Hyperparameter Optimization of CNNs. I used the open-source dataset known as MedMNIST[1] (specifically the OrganAMNIST) which contains 58850 images of dimensions 28px by 28px and has the in-built feature to create train/test datasets. The only pre-processing step used was to apply histogram equalization which also normalizes the images.

Exploratory Data Analysis

Upon importing the images into python, one the first steps performed was to make the train/test split and preview some of the images to get an idea of what preprocessing steps may be necessary. Next, to get an idea of what the FBP reconstructions look like, the Radon Transform is applied to a smaller batch of the train set and consequently, the iRadon transform was applied to this new batch and the Root Mean Squared Error between the original image and the iRadon reconstruction was calculated for this small subset of images to get a baseline value for this error.

Filtered Back Projection (FBP)

This method can be subdivided into two parts which are: Back Projection and Filtration. FBP was one of the first methods developed to reconstruct anatomically accurate images from raw CT data. Raw CT data graphically looks like several blurred sine waves with different amplitudes and phases which is why it is often called a Sinogram. The mathematical operation at the heart of this algorithm is the Inverse Radon Transform which solves the back projection part of the problem. To complete the filtering part, a 'ramp filter' is applied to the inverse radon function which, in layman's terms, is written in a way that attenuates high frequency features from the reconstructed image. The FBP algorithm is implemented using the scikit-image library's built-in function known as iRadon. An example of the iRadon function in action is shown below:

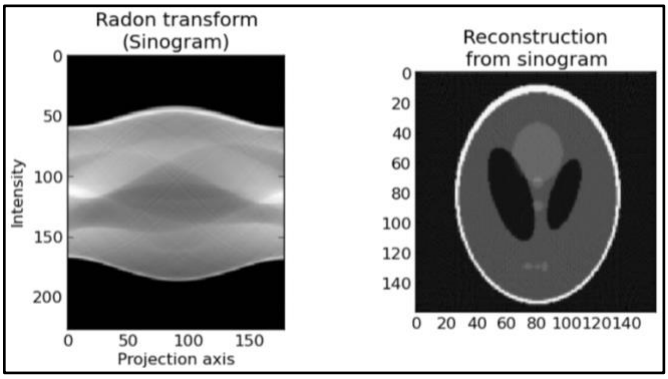

Figure 1[2]: This figure shows the Sinogram (like the raw data from a scanner) and the on the right is the reconstructed image produced using the iRadon function.

The same scikit-image class that contains the iRadon function also contains the Radon function which I used to 'simulate' the sinograms corresponding to each training and testing image. Although, each image in the original dataset has an appropriate label attached to it, to achieve the end goal of this project, they are of no use. The Radon function is used to generate

the Input Data for the CNN and the original images are used as ‘ground truth’ for training and evaluation.

Convolutional Neural Network (CNN)

As the name suggests, a convolutional neural network is a type of Neural Network which makes use of Convolutional Layers in its architecture. The package TensorFlow was used to develop and implement the Neural Network architecture in Python. A convolutional layer consists of a user-defined number of ‘kernels’ whose size is also determined by the user. Each kernel can be visualized as being a matrix and these matrices are iteratively multiplied with similarly sized sub-sections of the image one at a time and moved by an amount decided by the ‘stride’ factor defined by the user. Each filter is passed over the image to produce a new array/tensor and the size of this array/tensor depends on the arguments mentioned above.

In the model used for this project, the first layer of the first draft of the Neural Network consisted of a function call that specified a kernel size of 2x2 with a total of 54 kernels in this layer. This means that the output at this layer is 54 arrays of size 28px by 28px. The activation function used for this layer was ‘ReLU’ which stands for Rectified Linear Unit and the idea behind this activation function is that it leaves all the positive values unchanged, while mapping all the negative values to 0. The architecture of the model used in this project is as follows:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 54)        270

 conv2d_1 (Conv2D)           (None, 28, 28, 28)        6076

 max_pooling2d (MaxPooling2D  (None, 14, 14, 28)       0
 )

 flatten (Flatten)           (None, 5488)              0

 dense (Dense)               (None, 784)               4303376

 re_lu (ReLU)                (None, 784)               0

 reshape (Reshape)           (None, 28, 28)            0

=================================================================
Total params: 4,309,722
Trainable params: 4,309,722
Non-trainable params: 0
_________________________________________________________________
```

Hyperparameter Tuning

Once the initial model was designed and trained, the next logical step was to alter some hyperparameters in the model in hopes of potentially encountering a model with fewer parameters/nodes and comparable or better results. Even though it may seem like the model shown above is quite small, there are way too many hyperparameters to optimize in a time efficient manner. So, it was necessary to apply some sort of 'best guess estimate' to decide which specific hyperparameters were best to tune. So according to my intuition, one of the goals of this project is to see whether Neural Networks can perform image reconstruction faster than FBP.

I decided to focus on seeing if the number of kernels in the two convolutional layers could be reduced. The reason behind this choice was that the fewer the number of convolutional kernels, the fewer the mathematical operation this model would require to produce a reconstruction. Moreover, since the second convolutional layer's outputs are flattened, and connected to a Dense Layer of 724 nodes, reducing the number of kernels by 1 in this convolution step means a reduction of roughly 150,000 or so trainable parameters and the fewer the number of trainable parameters, the faster a model can be trained.

The tuning process was performed using the Bayesian Optimization function in the keras_tuner package. The search for optimal hyperparameters was done for over 20 trials and the objective was to keep the mean squared error of this model as low as possible while potentially removing redundant convolutional kernels. Upon receiving the optimal hyperparameters from this function, a new model was created with those values for the convolution kernels and the model was retrained on the same data for the same number of epochs as the original model so that the two could be directly compared with one another.

Results

Exploratory Data Analysis

One of the key findings when initially exploring the data was that none of the images were normalized. Moreover, a lot of the images had highly varying intensity values (meaning, some had bright spots with a maximum absolute intensity value close to 1 whereas, the same

value was a mere 0.6 in some other image. This would give the model another parameter that it'd have to predict correctly in every reconstruction which would not only increase the number of epochs needed to train the model to a sufficiently low reconstruction error, but it may also be require more convolutional layers to perfect. Fortunately, this problem was rather easily countered by implementing histogram equalization when generating the simulated sinograms as well as when applying circular masks to the original image.

Filtered Back Projection (FBP)

Circular Masks were also applied to the original image because the FBP algorithm can only reconstruct circular images because the method relies on back-projecting images from the polar coordinate domain. Next, the small sample of images from the training set mentioned in the Methodology section were used to get an idea of what the rms error of the FBP reconstructions would be. The results of this analysis are shown in the figure below.

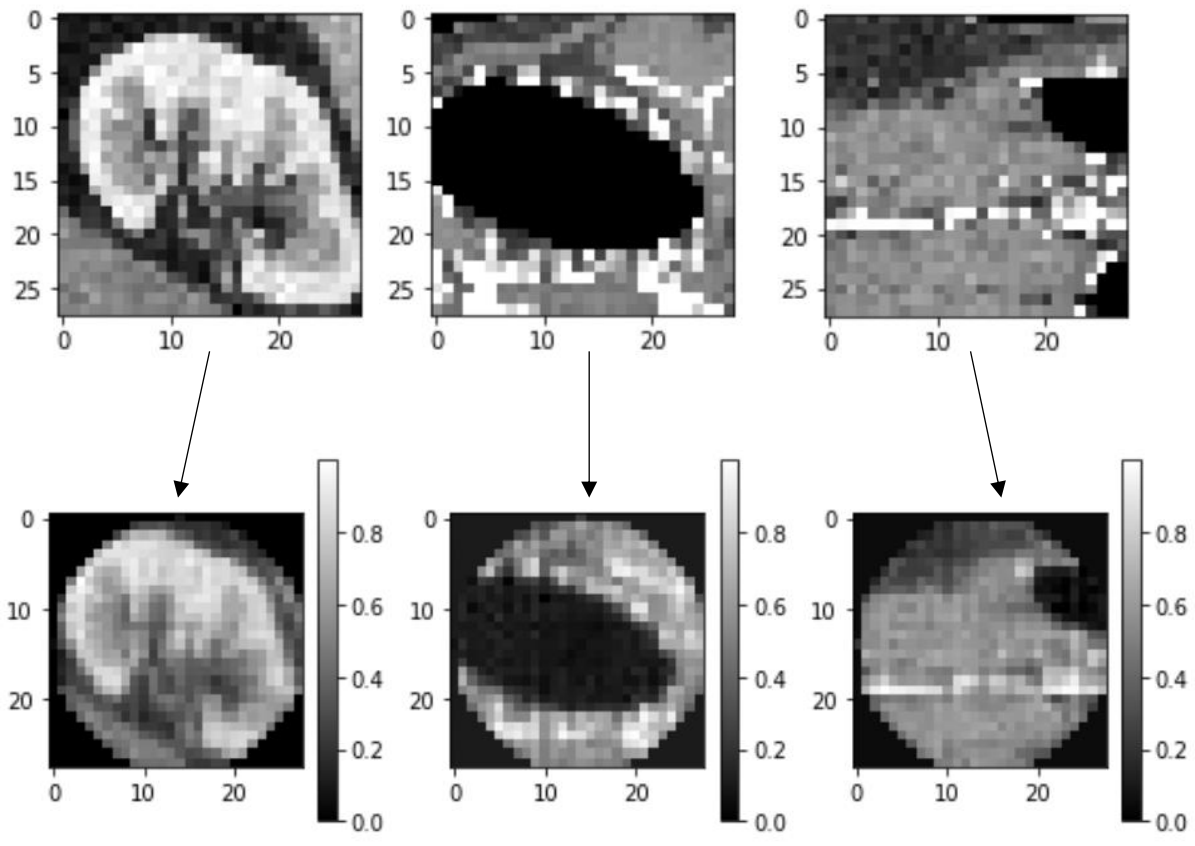

Figure 2: Three of the Original Images (w/o Preprocessing) and their FBP Reconstructions.

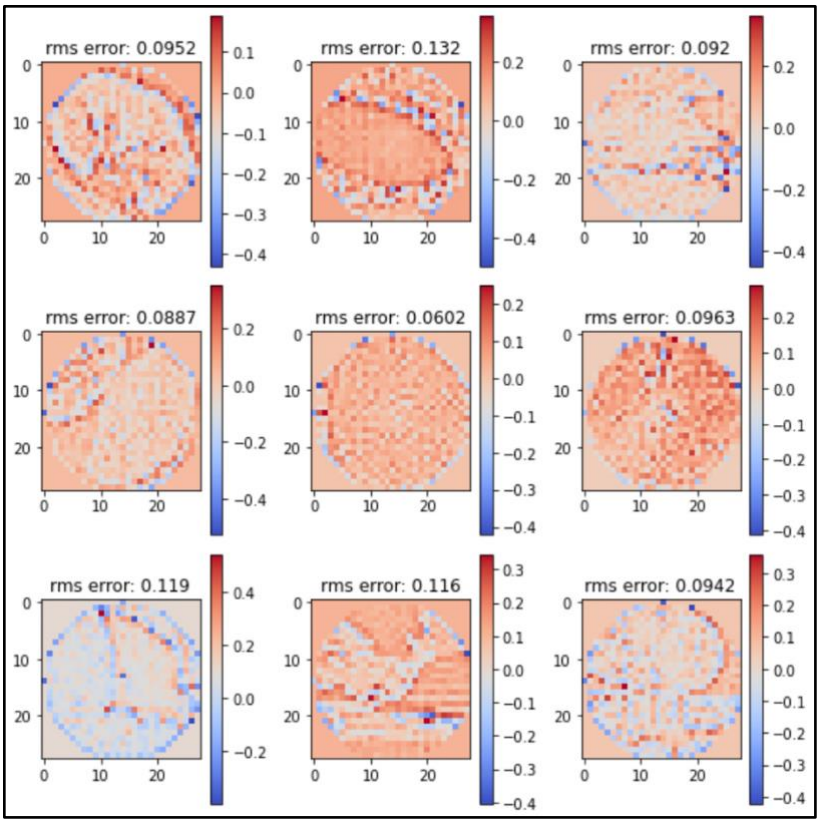

Figure 3: The RMSE measurements of some of the initial FBP reconstructions vs. the pre-processed version of the original. (Shown for proof of concept).

Convolutional Neural Network (CNN)

The Neural Network was trained for 10 epochs and the Mean Squared Error was used as the 'loss' measure during training and the standard 'adam' optimizer was used with the default learning rate of 0.001. The Validation Loss seemed to have relatively flattened out after 10 epochs which is where the model was stopped during training to avoid overfitting the model to the training set.

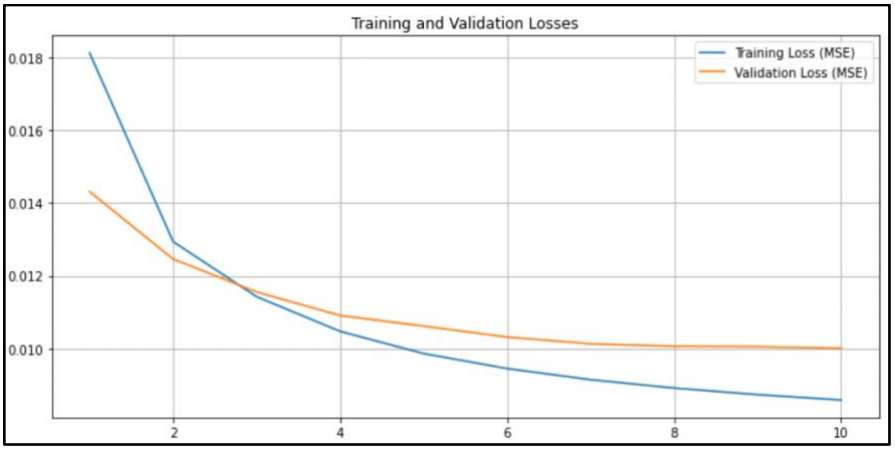

Figure 4: The MSE measurements of the Training and Validation sets at the end of each epoch.

Upon training the Neural Network, its performance was evaluated against the FBP algorithm to measure how the speed of reconstruction of the entire test set as well the average MSE for these reconstructions by these two algorithms stacked against one another. Although, it is unclear what the Amortized Time Complexities of either of these algorithms may be, if we proceed with the assumption that it takes the same amount of time to reconstruct a 28px-by-28px image, over the course of thousands of reconstructions, the curve would be perfectly linear.

It was found that the Neural Network completed the reconstruction in ~1.37 seconds whereas, FBP took ~3.89 seconds to finish the same task which is an improvement by a factor of ~2.83 times in favor of the Neural Network.

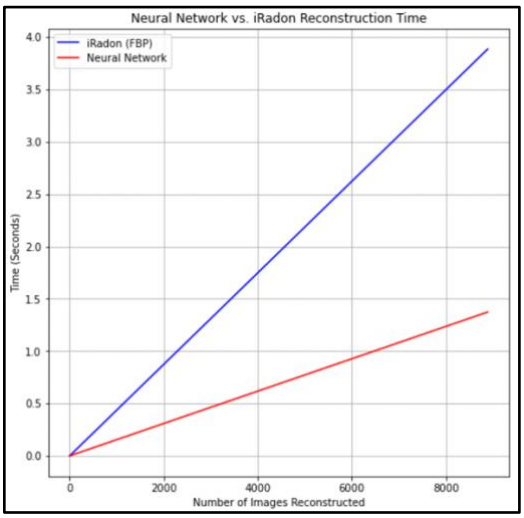

Figure 5: The reconstruction times for both the FBP method (blue) vs. the Neural Network (red).

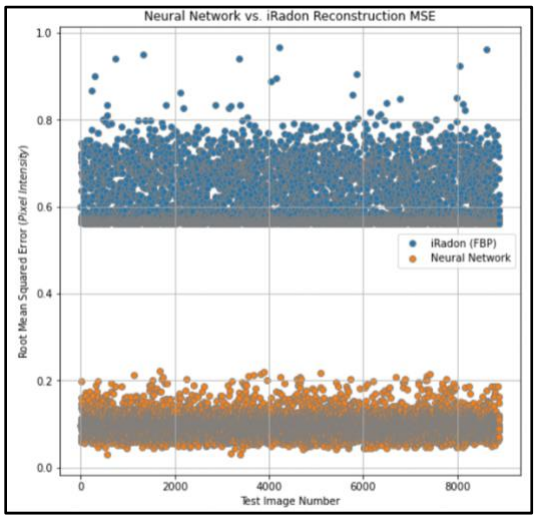

Figure 6: The MSE measurements of the FBP Reconstructions (blue), and Neural Network Reconstructions (orange).

Moreover, the Mean Squared Error in the reconstructions made by the Neural Network (average of ~0.097 for the whole set) were also drastically better than its FBP counterpart (average of ~0.600 for the whole set), marking an improvement by a factor of ~6.17 times in favor of the Neural Network. Not only this, looking at the reconstructions visually told much the same story as the average MSE values as it seemed that the Neural Network's reconstruction was able to recreate some of the slight nuances in the Original Images more accurately than the FBP algorithm could. A side-by-side comparison of some of these reconstructions is shown below although visual confirmation is a less rigorous method of validation, it is still worth looking at.

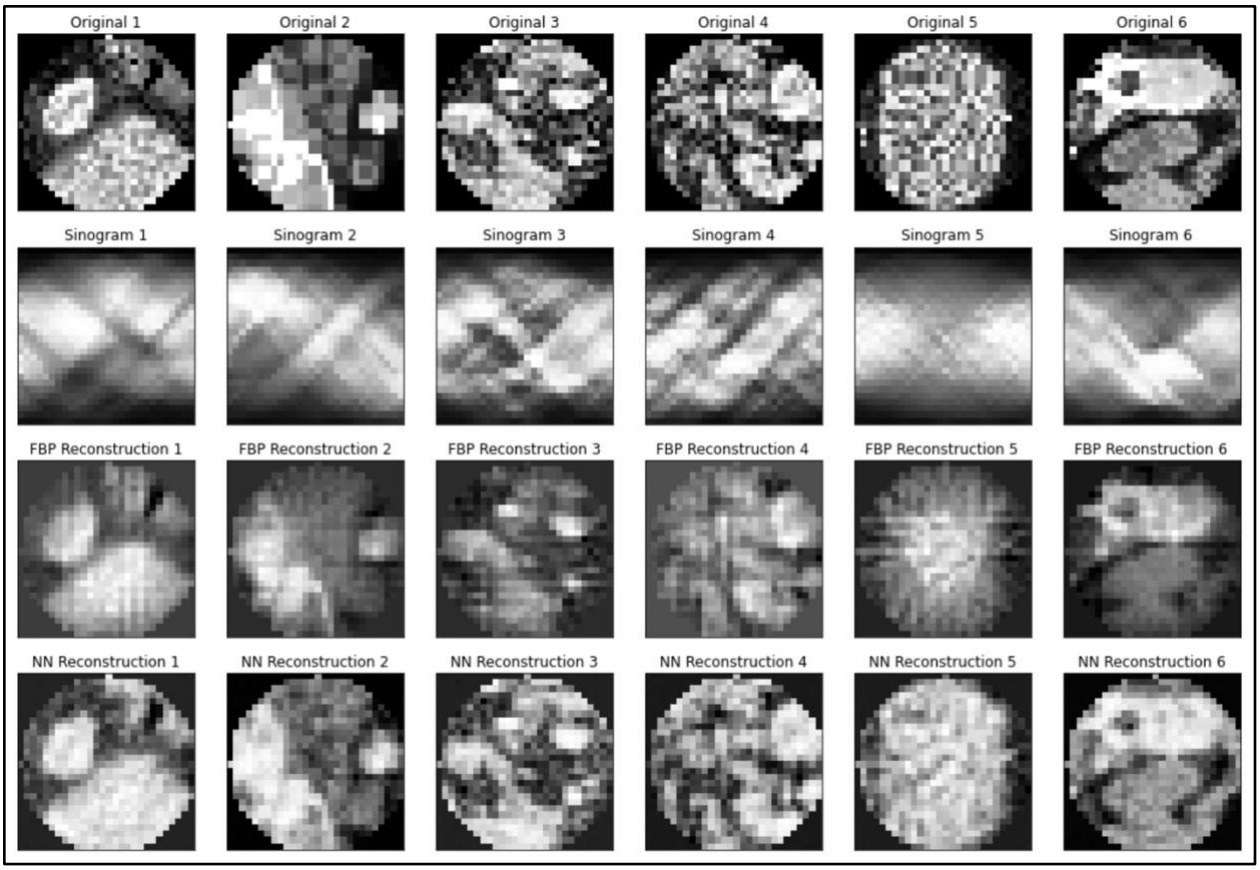

Figure 7: A direct visual comparison between the two methods of reconstruction (compared column-wise).

Based on the results obtained above, it seems as if the neural Network based approach at Image Reconstruction does quite well that its FBP counterpart in this setting. However, determining the generalizability of a model trained in this fashion remains to be seen.

Hyperparameter Tuning

Now that we had some good results with a model that performed quite well when compared to FBP, the next step was to perform some Hyperparameter tuning to see if any of the convolutional kernels in the two Convolution layers were redundant and could be dropped. As mentioned before, doing so (especially in the second convolutional layer) could theoretically lead to faster training as well as reconstruction times. So, a new model was set up in a way that the keras_tuner could choose a number between 27 and 54 for the number of kernels in the first convolutional layer, and similarly, a number between 10 and 27 for the second convolutional layer. It was allowed to change these numbers by a step size of 2 per iteration.

A new model was then created, trained, and tested with a variety of different initial conditions and the Bayesian Optimizer was used to converge towards the solution that produced the most accurate results. It took a while for the tuner to finish generating all the results but by the last iteration, it had settled on the values of 53 and 26 for the number of convolutional kernels which was extremely close to the original model which used 54 and 28 kernels for the first and second convolutional layers respectively. Although these new hyperparameters were still worth examining because the second layer had returned 2 fewer kernels which had reduced the number of trainable parameters in this new 'Optimal Model' to ~4 million which was lesser by nearly ~300,000. The 'Optimized' model architecture is shown below:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 53)        265

 conv2d_1 (Conv2D)           (None, 28, 28, 26)        5538

 max_pooling2d (MaxPooling2D  (None, 14, 14, 26)       0
 )

 flatten (Flatten)           (None, 5096)              0

 dense (Dense)               (None, 784)               3996048

 re_lu (ReLU)                (None, 784)               0

 reshape (Reshape)           (None, 28, 28)            0

=================================================================
Total params: 4,001,851
Trainable params: 4,001,851
Non-trainable params: 0
_________________________________________________________________
```

Upon training this new 'Optimal Model' for 10 epochs, a very similar curve for the Validation Loss by epoch was obtained (compared to the original) as shown below:

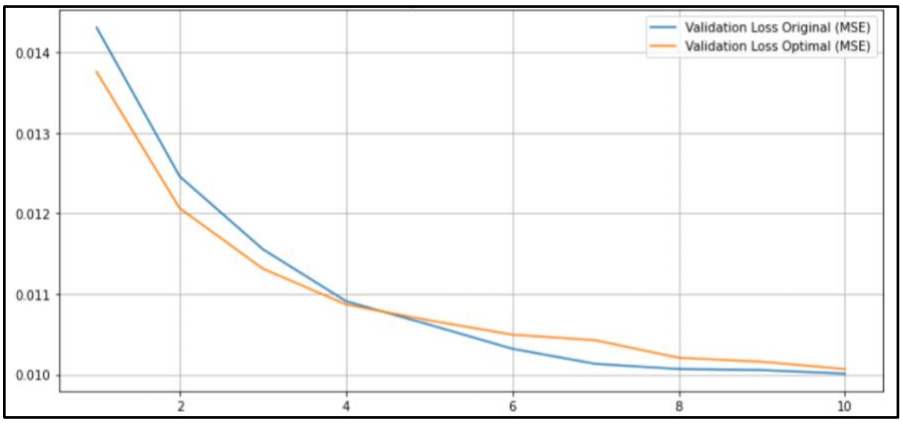

Figure 9: The MSE measurements of the Validation sets at the end of each epoch for the Original and 'Optimized' Models.

It was also observed that despite the obvious reduction in the total number of trainable parameters, the new 'Optimized' Model performed worse in terms of the time required to complete the reconstruction as well as the MSE of these reconstructions:

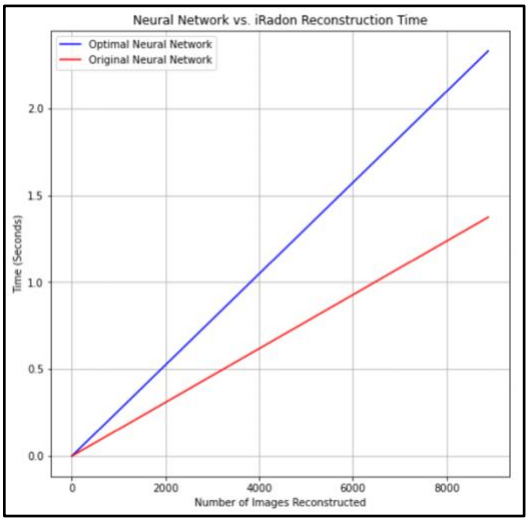

Figure 10: The reconstruction times for both the Optimized Model (blue) and the Original Model (red).

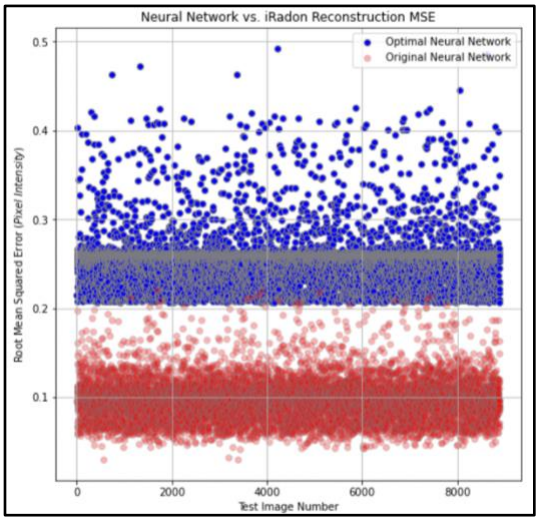

Figure 11: The MSE measurements of the Optimized Model (blue), and the Original Model (Red).

Although it was anticipated that the 'Optimized' Model would perform slightly better or roughly the same as the original model at the reconstruction task, what these results show us is the fact that even the slightest alterations to model parameters can lead to drastically different results. In this case, the Optimized Model was ~1.70 times slower and ~2.62 times less accurate than the original model. This also goes to show that the assumptions made about the relationship between the number of convolutional filters, the number of trainable parameters and the time taken to finish the reconstruction were most likely incorrect. Moreover, none of the convolutional filters in the original model were redundant and reducing them in either layer would lead to a worse accuracy score (since the Bayesian optimizer started from the lowest number of kernels available in the function and converged to a higher value towards the last few trials).

Conclusions and Future Works

Finally, to wrap things up, this project has been extremely helpful in allowing me to understand and implement complex topics related to Neural Network architecture design and optimization. The initial Neural Network outperformed FBP which neatly answered the initial question posed in this project, however, it remains to be seen whether this model would be universally applicable in a clinical setting or not. More testing in a relevant setting would need to be done to determine this universality. Even though the Optimization results were the opposite of what was to be expected, they still managed to show that some of the initial estimations of the complex relationships between hyperparameters and the time and accuracy of reconstruction may have been incorrect and needs a much closer evaluation. In the same vein, more tuning and optimization of some other hyperparameters may return better results.

References

[1] Xuanang Xu, Fugen Zhou, et al., "Efficient multiple organ localization in CT image using 3d region proposal network," IEEE Transactions on Medical Imaging, vol. 38, no. 8, pp. 1885–1898, 2019.

[2] Van der Walt, S., Schonberger, Johannes L, Nunez-Iglesias, J., Boulogne, Franccois, Warner, J. D., Yager, N., … Yu, T. (2014). scikit-image: image processing in Python. *PeerJ*, *2*, e453. [URL]