REDUCTION OF NOISE ON CT RECONSTRUCTION

USING LEARN AND SART

Written by Pedro Goncalves Mokarzel

Based on the work done by Noah Paul McMichael, Pedro Goncalves Mokarzel, and Dr. Thomas Humphrey. Oriented by Dr. Thomas Humphrey and Dr. Dong Si.

Written 08/26/2019.

BACKGROUND

CT Reconstruction is the process in which a CT Image can be formed by an x-ray. Imperfections, called noise, may be created because of the nature of the projection of x-rays. Noise can obscure important details from an image. CT Reconstruction start with a projection step, which can be seen in figure 1.0. The Sinogram is created from one dimensional array ordered from each point the sensor receives radiation.

Projection

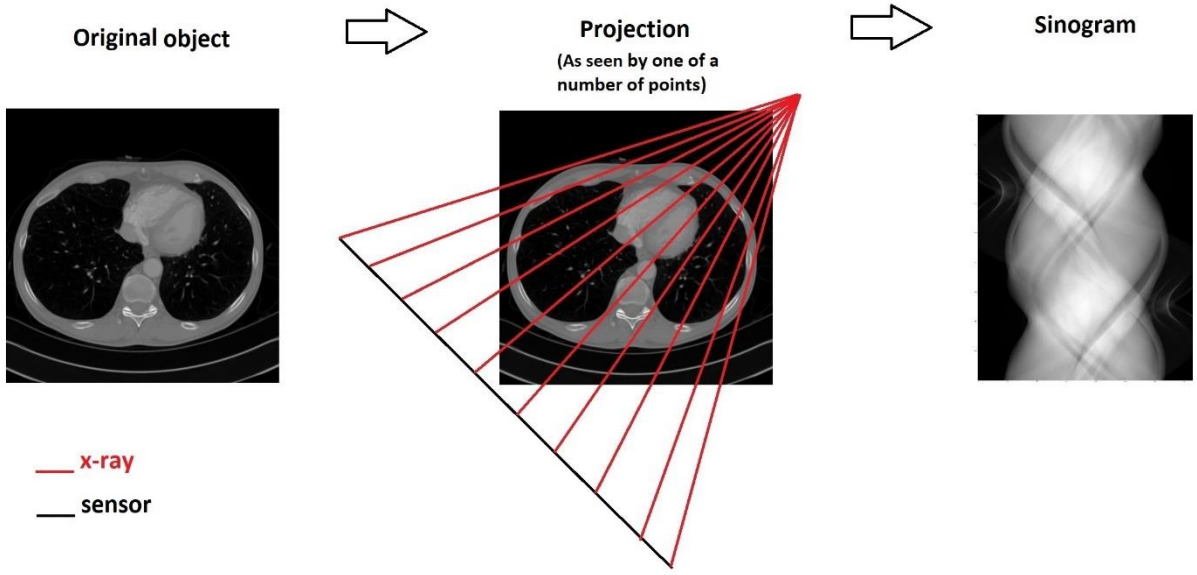

Figure 1.0: shows process of creation of sinogram through projection

The sinogram can be transformed into an image through the process of backprojection. Every point from the sinogram feeds that array point through the whole image. It does for all arrays at their corresponding angles while smoothing out their contributions such that overlapping points are highlighted, slowly creating the image.

ABSTRACT

One of the goas of CT Scanning is to increase the amount of radiation it uses in patients by either taking less pictures or decreasing the amount of radiation used in the pictures. However, both methods create noise in the image. Therefore, processes that eliminate noise are advantageous to allow a smaller amount of radiation to be used to gain images. Noise can be limited by the implementation of an estimated filter that is added to the image. To solve the problem, a potential solution is the inclusion of a convolutional neural network that creates a model for the filter based on the CT images. This paper will apply the SART and LEARN methods for CT reconstruction to see which algorithm is more efficient, and

effective in the creation of the model. From those results, the model will be compared against traditional estimating methods.

SART and LEARN

SART and LEARN context

The following equations describe the transformation executed by the neural networks. *A* and A^t are the transformations of projection and backprojection. Both transformations are performed by methods provided by the PYRO-NN framework. Both neural networks have a step which uses a convolutional neural network for training (*conv*). This network is structured equally between the two method, so the results will only reflect the structure of the training algorithm. The convolutional layers were made of a convolutional 2d layer with activation of tf.nn.relu, a normal convolutional layer, a batch normalization layer, and an output layer. In the equations described below, all mathematical transformations are executed with TensorFlow so the process becomes part of the layer it is under.

SART

SART is used to describe neural network where every set of layers is described as an ART layer, and a set of convolutional layers. The ART algorithm can be described as:

$$x^{tmp} = x^k - DA^tM(Ax^k - b)$$

In the equation, *A* is the process of projection; *b* is the sinogram input created from a previous created sinogram; *M* is created from the projection of a tensor of float32 of ones with the same shape as the input sinogram; A^t is the process of backprojection; *D* is created from the projection of a tensor of float32 of ones with the same as the original projected image; x^k is defined by the sequence of transformations applied by the equation where. In the first iteration, x^k consists of an array of zeroes.

For the training of the convolutional layers, x^{tmp} is used such that the final transformation can be described then as:

$$x^{k+1} = x^{tmp} - conv(x^{tmp})$$

For the software, the layers are created based on the number of iterations given by the user, such that each iteration creates one of the CNN_layer, and one of the ART_layer, as seen in figure 2.0. The general network created by these layers can be seen in 2.1.

One Layer of SART

General Network of SART

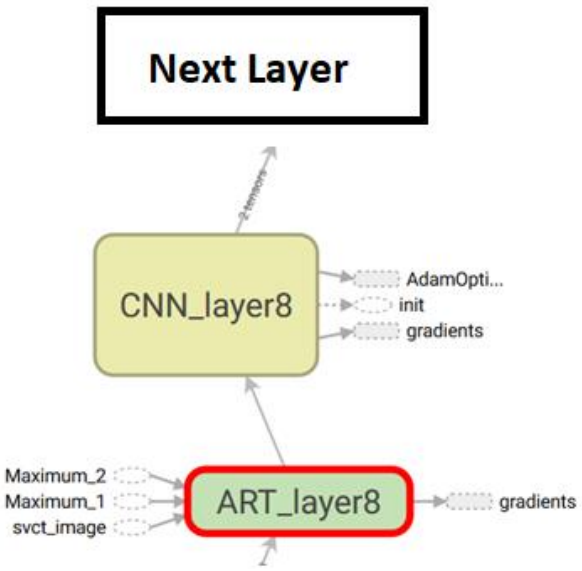

Figure 2.0: shows the set of layers created from one iteration of the algorithm.

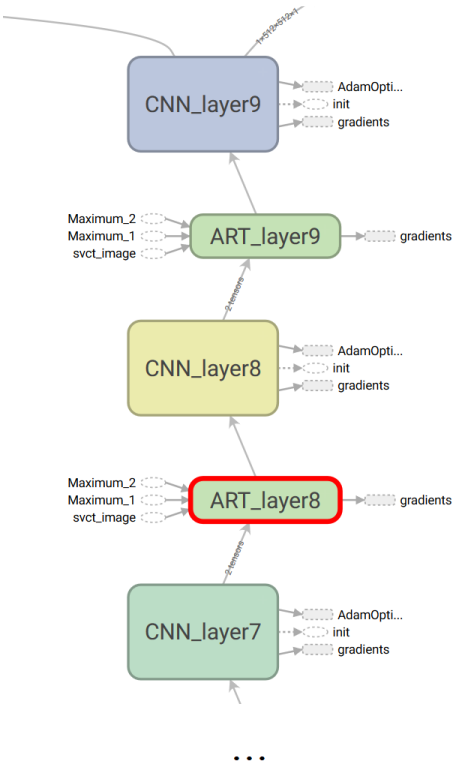

Figure 2.1: shows a slice of the overall neural network.

In the network, the Art layer is expanded in figure 2.2. This is the first layer created by the iteration, and it provides the input to the CNN_layer (seen in figure 2.3). The CNN layer trains based on the output of the ART layer, and then outputs to the next layer. The Last layer shapes the image for the rest of code.

Art Layer

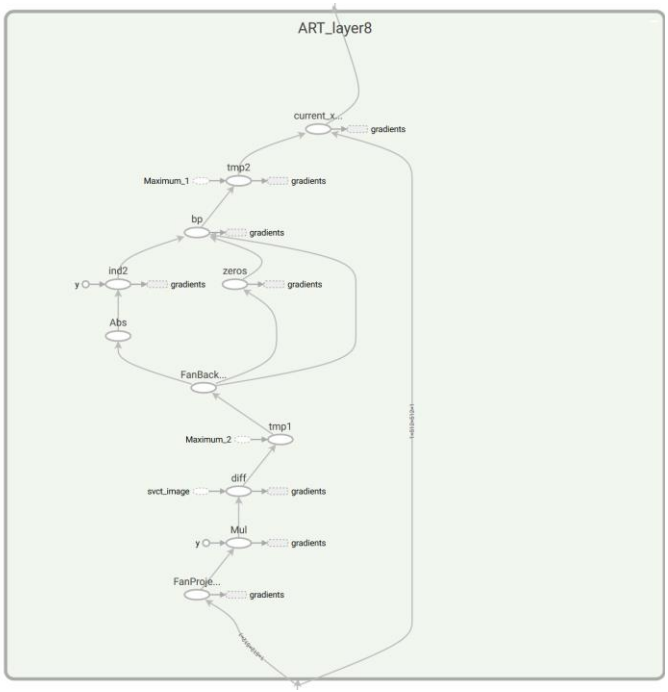

Figure 2.2: shows the nodes inside of the art layer

CNN Layer

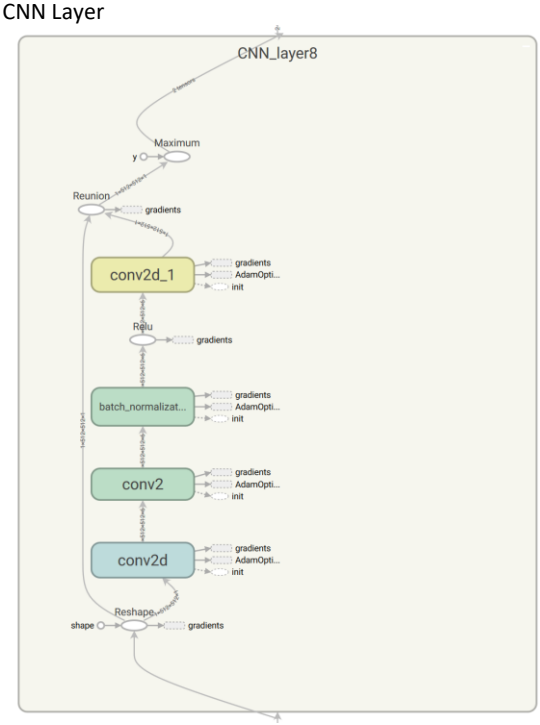

Figure 2.3: shows the layers and nodes from inside of CNN layer

LEARN

LEARN is used to describe neural network where every set of layers is described as a LEARN layer, and a set of convolutional layers. The difference between the LEARN layer and the SART layer is that, x^k is fed to the convolutional network before the application of the of the rest of the construction algorithm. The equation can describe then as:

$$x^{k+1} = x^k - A^t\left(Ax^k - b\right) - conv(x^k)$$

From the equation, *A* is the process of projection, performed by a PYRO-NN method; *b* is the sinogram input created from a previous created sinogram; x^k is defined by the sequence of transformations applied by the equation (in the first iteration, it consists of an array of zeroes); A^t is the process of backprojection. Through that equation, an iteration creates layers in figure 3.0 which come together in figure 3.1.

One layer of LEARN

General Network of LEARN

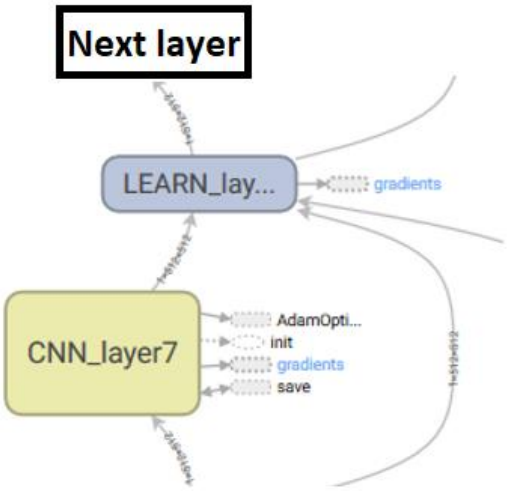

Figure 3.0: shows the layer created by 1 iteration of the LEARN algorithms

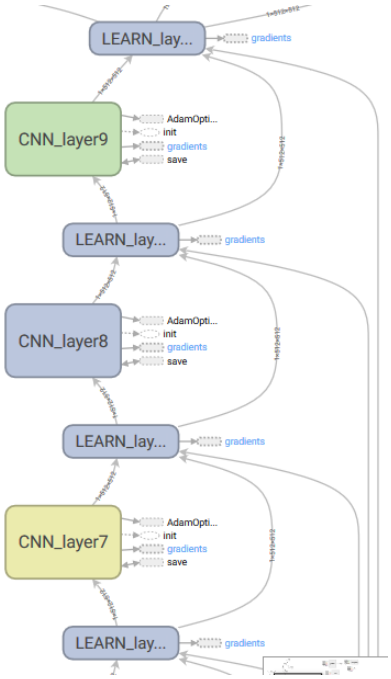

Figure 3.1: shows a slice of the overall neural network.

The *conv* step describes the same convolutional network as described in SART, and figure 2.2. The LEARN_layer consists of the reconstruction process. The layer can be described in figure 3.2. In the LEARN network, both layers can execute parallelly. For this network, the CNN layer is passed information first, then the LEARN layer executes the rest of the algorithm with both information from the CNN layer, and the previous LEARN layer output. The Last layer will properly shape the image for the rest of code.

Inside of LEARN layer

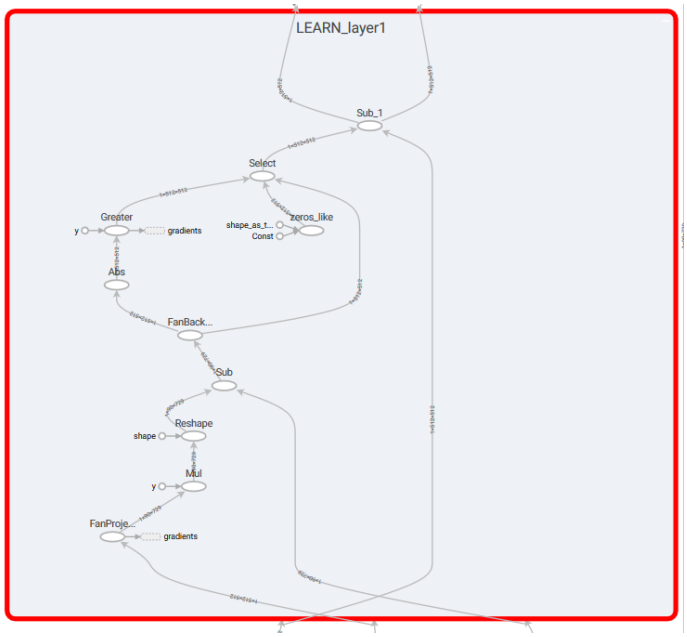

Figure 3.2: Shows the inside of LEARN

SOFTWARE

1. PYRO-NN

The research is implemented using the PYRO-NN framework. This is a framework created for CT reconstruction. It allows for the easy implementation of neural networks through TensorFlow. More information can be found in the original PYRO-NN paper (Syben, C., Michen, M., Stimpel, B., Seitz, S., Ploner, S. Maier, A.K, 2019).

PYRO-NN offers an easy way to specify the form of the scan. The shapes used for the implementation of this paper are fan-flat 2D, and parallel 2D. A larger focus was given to the training of fan-flat images as they better simulate the work usually done in the field. The framework also offer access to projection and backprojection processes.

During the construction of our code, version 0.04 was released, which works in allowing the use of batches inside of the PYRO-NN layers. The code was released based on the bug report that can be found at github.com/csyben/PYRO-NN/issues/4. Its first version only allows a batch of 1, but it made it made it so the layers, and the convolutional networks could work together.

2. Data generation

A class called SynoGerator.py was created in order to easily produce data to be trained. This class takes in arguments from the user and applies the projection process from a file or a given directory and directs the results to another given directory. The specifications are used to create the geometry object used by PYRO-NN to process the projection. With SynoGenerator, the user can process large numbers of images throw the projection phase to set it up for proper training. This class was inspired by “make-

sino.py" from previously created ASTRA code, provided at github.com/DrDongSi/CT_Image_Reconstruction/tree/master/ASTRA_recon_code. The code uses a straightforward subroutine architecture, using methods to facilitate the use of multiple shapes. User can specify a shape to base sinograms of, and file input (as either a specific file or a directory) and output (as a directory).

3. The Architecture of the Training Model

The general architecture chosen for this paper prioritized the comparison of the SART and LEARN algorithms. The general architecture is based on the work done in github.com/DrDongSi/CT_Image_Reconstruction/tree/master/DnCnnCode, which implemented code based on " Superiorization: An optimization heuristic for medical physics". A Data Flow Diagram for the current network can be seen in figure 4.0. It is believed that this should create classes with high usability.

Data Flow Diagram on network used for SART and LEARN

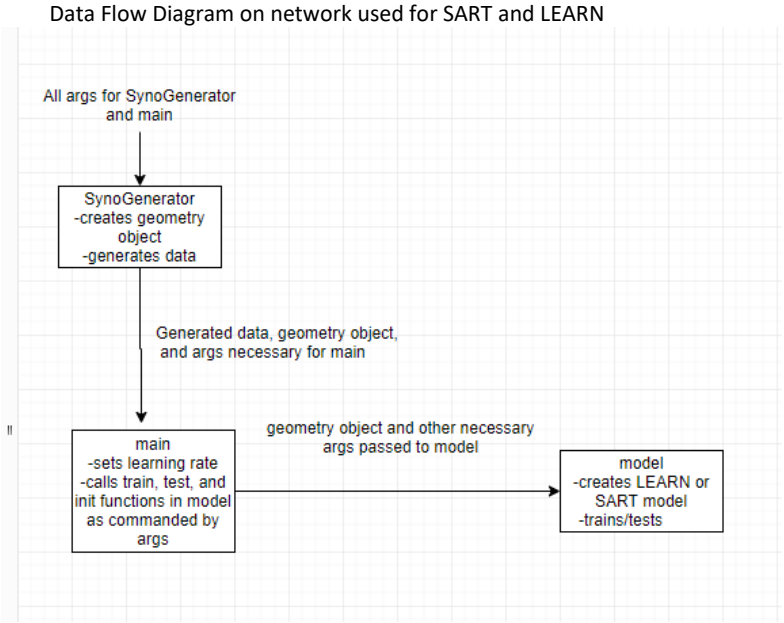

Figure 4.0: Shows how images are fed to SynoGenerator, and then projected into sinogram. The sinogram is saved, and then taken by main which quarries the information to model which creates the network. Model also trains and tests the network. DFD created by Noah Paul McMichael

4. Development process

There were 2 phases divided for the project, corresponding to the two directories that can be found at github.com/DrDongSi/CT_Image_Reconstruction under ./SART_LEARN/phase_1, and ./SART_LEARN/phase_2. Phase 1 consisted of implementing the SART algorithm with PYRO-NN with the ASTRA architecture without a convolutional neural network. Phase 2 consisted of implementing the convolutional layers with both SART and LEARN algorithms. During phase 2, version 0.04 was released, and therefore was adapted into the code.

Debugging code can be problematic due to the nature of Tensorflow, and CUDA. Most conventional debuggers (like pdb) do not work, so print statements were used to check the inner state of variables. A debugger that works is Tensorboard which wraps Tensorflow. In the software, the Tensorflow sess can have its gpu_options wrapped with:

```
gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=0.95)
```

5. Version specifications and conditions

PYRO-NN requires a specific version of tensorflow, and python. The python version must be 3.6. The tensorflow must be built up to version 1.9 with the PYRO-NN(found at github.com/csyben/PYRO-NN) inside of the tensor code (found at github.com/tensorflow/tensorflow). It is important that other software used can work with these versions.

The current used version of PYRO-NN (0.04). This version's PYRO-NN inputs are a tensor of shape [*x, y, z*] where *x* is the number of batches being trained, however, in this version *x* must equal 1. In the future, this feature is set to expand.

6. Issues

Data generation software seems to slow down as it executes. This is probably caused by memory leak inside of the loop that goes over the projection of artefacts. It might be advantageous in the future debug this leak to create data more efficiently.

Currently, when trying to train a network in a machine with other processes happening, even when specifying a GPU not used, a CUDA error is thrown due to a lack of use of memory. It might be an advantage to look at why this GPU issue is happening for more parallel work to happen in a machine at the same time.

The LEARN algorithm seems to work as intended, creating the model, and evaluating as expected, however the SART algorithm has a problem in it. The following bug is appearing when it is executed:

```
Traceback (most recent call last):
  File "main.py", line 244, in <module>
    tf.app.run()
  File "/home/NETID/pgmoka/anaconda3/envs/updated_pyro_0.04/lib/python3.6/site-
packages/tensorflow/python/platform/app.py", line 125, in run
    _sys.exit(main(argv))
  File "main.py", line 221, in main
    batch_size, ckpt_dir, num_epochs, lr, sample_dir)
  File "/data/pgmoka/CT_Image_Reconstruction/SART_and_LEARN/phase_2/model.py", line
398, in train
    summary_writer=writer, summ_img=img)  # eval_data value range is 0-255
  File "/data/pgmoka/CT_Image_Reconstruction/SART_and_LEARN/phase_2/model.py", line
362, in evaluate
    clean_image, noisy_image, output_clean_image)
```

```
   File "/data/pgmoka/CT_Image_Reconstruction/SART_and_LEARN/phase_2/utils.py", line
112, in save_images
     cat_image = np.concatenate([ground_truth, noisy_image, clean_image], axis=1)
   File "<__array_function__ internals>", line 6, in concatenate
    ValueError: all the input array dimensions for the concatenation axis must match
exactly, but along dimension 0, the array at index 0 has size 512 and the array at index
1 has size 90
```

A speculation is that this is caused by either the creation or processing of M, D, or x_initial when given to the SART execution algorithm. Further inquiry is needed in this area for the neural network to be created.

7. Future Steps

There is currently github branch for 'pyro-nn-layers', found at github.com/csyben/PYRO-NN/pull/3, that aims to bring PYRO-NN implementation to keras and version 2.0 of tensorflow. It could be advantageous in the future to use this due to the usability of keras.

For the evaluation of the image, the ssim process might generate more accurate results for CT images. This process could be advantageous in comparing the two algorithms as it can better analyze the details eliminated by the trained filters

In the current version, ranges have been adjusted in model.py and utils.py so that images are more visible from 0-255 to 0-2000. While this has proven to yield good results in the current version, in the future development it is important to keep in mind the possible ramifications of this change.

PYRO-NN 0.04 only implements the use of batch size 1. In future versions this is set to change to allow for larger batch sizes. When this happens, our code should change to properly implement PYRO-NN.

BIBLIOGRAPHY

Chen, et al. (2017). LEARN: Learned Experts' Assessment-based Reconstruction Network for Sparse-data CT. *IEEE Transactions on Medical Imaging*. National Natural Science Foundation of China. DOI 10.1109.

Herman, G. T. (2012). Superiorization: An optimization heuristic for medical physics. *Medical Physics: The International Journal of Medical Physics Research and Practice*. (39, 5532). DOI 10.1118

Syben, C., Michen, M., Stimpel, B., Seitz, S., Ploner, S. Maier, A.K.(2019). PYRO-NN: Python Reconstruction Operators in Neural Networks. 91058 Erlangen, Germany. University of Erlangen-Nurnberg