Mini Project

Callum Lau, 19102521

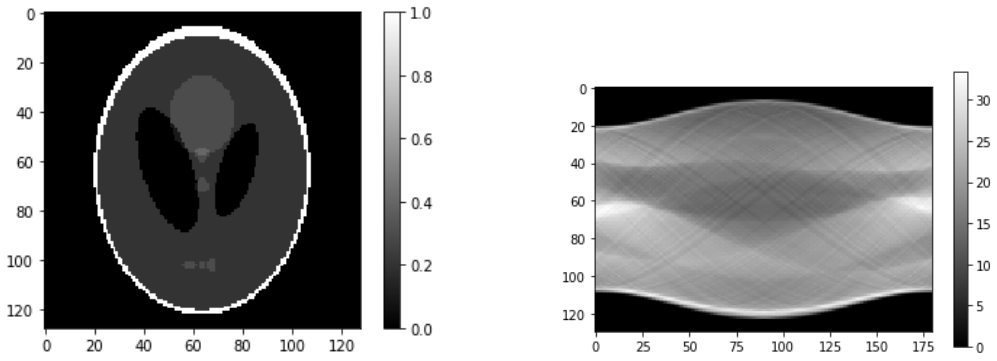
Inverse Problems

April 22, 2020

1 Calculate the Radon transform of an image and test the back-projection method.

1.1 The Shepp-Logan phantom and Radon transform

The Shepp-Logan phantom of size 128×128 is displayed in Figure 1.1a. For the rest of the report this will be refered to as $\boldsymbol{f}_{\text{true}}$. In order to generate the Radon transform $\boldsymbol{g} = \mathcal{R}\boldsymbol{f}_{\text{true}}$ of the phantom, the python package `astra` was used. The key construction parameters for the projection geometry (sinogram space) are the number of detector pixels, the angle range, and the number of samples taken in that range. The size of the sinogram is derived from these parameters, namely size($\boldsymbol{g}$) = detector pixels $\times$ number of samples . The size of the sinogram generated from 1-degree intervals over the angle range 0-179 with 130 detector pixels is therefore of size 130×180, and the resulting image is shown in Figure 1b.



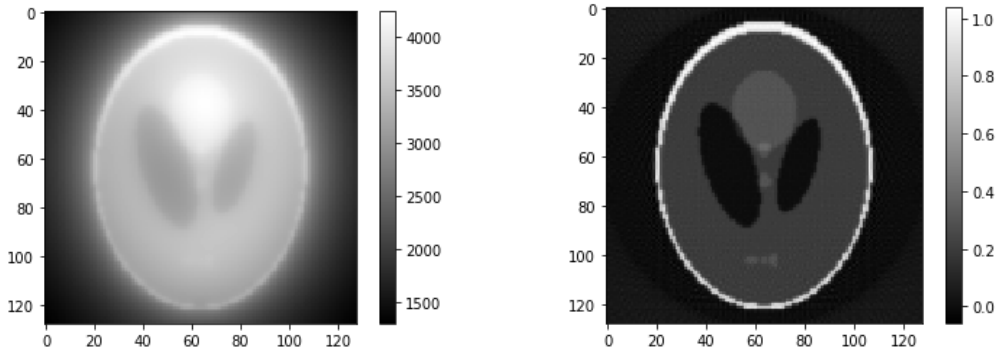
(a) $\boldsymbol{f}_{\text{true}}$. (b) $\boldsymbol{g} = \mathcal{R}\boldsymbol{f}_{\text{true}}$.

Figure 1.1: Shepp-Logan phantom and resulting sinogram

1.2 The Unfiltered and Filtered Backprojection

The *unfiltered* backprojection is found by simply taking the adjoint Radon transform of the sinogram $\mathcal{R}^*\boldsymbol{g}$. The resulting (unsatisfactory) reconstruction of the data-generating image $\boldsymbol{f}_{\text{recon}}$ is shown in Figure 1.2a. The adjoint operator must project the data image to the same space as the data-generating image, and therefore the reconstruction will have dimesion 128×128.

Conversely, the *filtered* backprojection provides an exact reconstruction formula and hence an exact reconstruction as shown in Figure 1.2b. We can verify that the filtered backprojection gives a good estimate of the inverse Radon transform by computing the *structural similarity index measure* (SSIM) between $\boldsymbol{f}_{\text{recon}}$ and $\boldsymbol{f}_{\text{true}}$. The `skimage.metrics` package provides such a function and gave a value of $\approx$`0.94`, indicating high similarity between the two images.



(a) $\boldsymbol{f}_{\text{recon}} = \mathcal{R}^*\boldsymbol{g}$.

(b) $\boldsymbol{f}_{\text{recon}} = \mathcal{R}^*\boldsymbol{g}$.

Figure 1.2: Unfiltered and filtered backprojection

1.3 Analysing reconstruction error

In order to assess how the reconstruction error grows with noise level σ, Gaussian noise with mean 0 and variance σ^2 was added to the data $\boldsymbol{g}$ for varying levels of σ. The reconstruction error was quantified by mean-square error. The resulting error versus noise curve is plotted in Figure 1.3. Interestingly, we observe a roughly *linear* relationship between the MSE and noise level. Furthermore, examples of reconstructions for increasing noise levels are plotted in Figure 1.4.

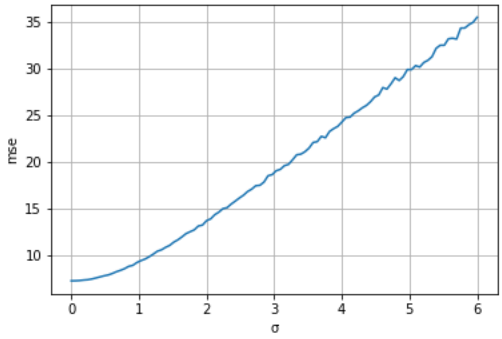


Figure 1.3: Noise level vs reconstruction error.

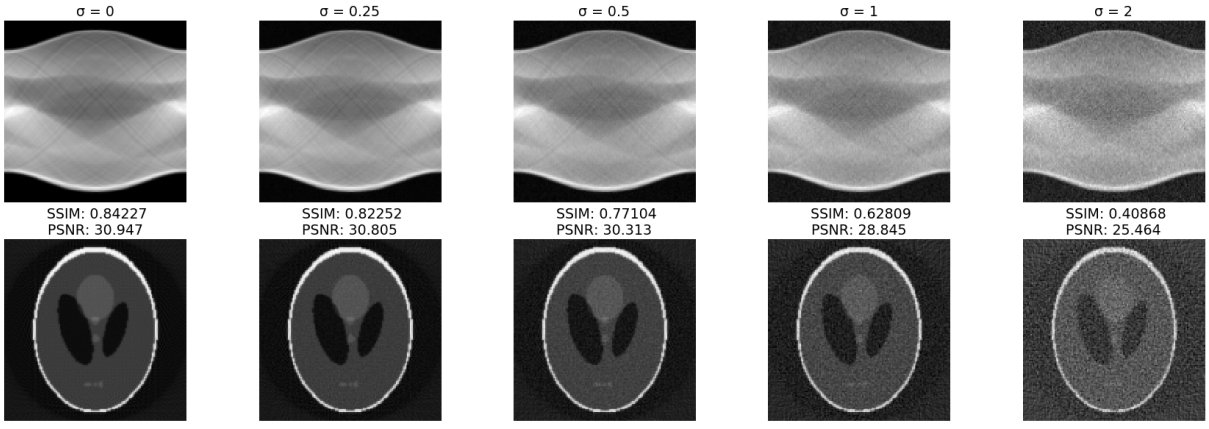
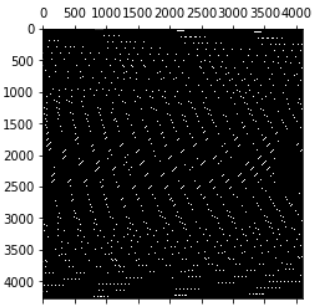


Figure 1.4: Example reconstructions for increasing noise level.

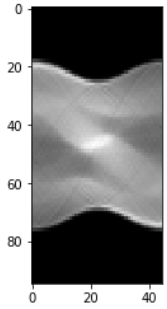
2 Calculate an explicit matrix form of the Radon transform and investigate its SVD.

2.1 Constructing the matrix

The explicit matrix form A of the Radon transform was calculated for a 64×64 resizing of the Shepp-Logan phantom using the method as described in the question. For illustrative purposes we show the matrix A for 45 angles and 95 projection samples and sinogram $\mathsf{A}\boldsymbol{f}_{\text{true}}$ in Figure 2.1. Note this matrix has 45×95 rows and 64×64 columns and is therefore of size 4275×4096.



(a) Radon matrix A.

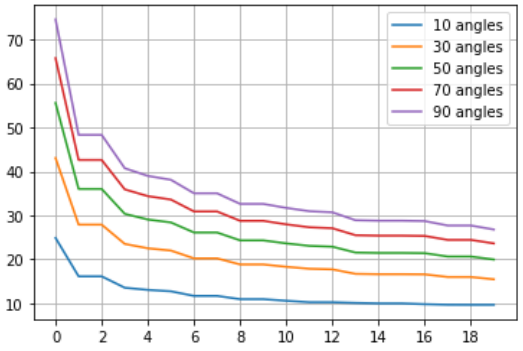


(b) $\boldsymbol{g} = \mathsf{A}\boldsymbol{f}_{\text{true}}$.

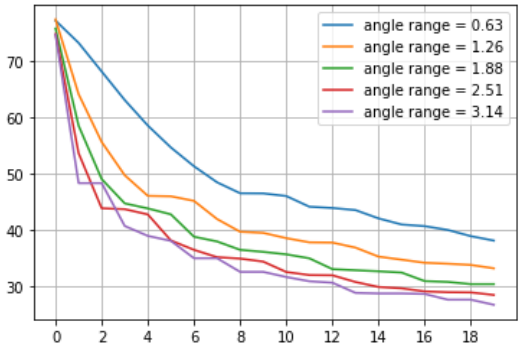
Figure 2.1: Radon matrix and resulting sinogram

2.2 Evaluating the SVD

The SVD of the matrix A was taken after constructing A over a varying number of angles and ranges. Plots of the resulting singular value spectrum are displayed in Figure 2.2a (fixed range of π, varying number of angles) and Figure 2.2b (fixed number of 90 angles, varying range). Only the first 20 singular values are given.



(a) SVD for varying number of angles.



(b) SVD for varying angle range.

Figure 2.2: SVD spectrum

In Figure 2.2a we observe that increasing the number of angles seems to lead to strictly greater singular values σ_i at each index i. In Figure 2.2b we observe that increasing angle range generally leads to smaller singular values and potentially a faster decaying spectrum. In combination then a possible conclusion is that increasing both the number of angles and the angle range will lead to greater ill-conditioning of the inverse problem, since with an increasing but faster decaying spectrum of singular values the condition number of the matrix A increases.

3 Implement a matrix-free regularised least-squares solver for the Radon Transform.

3.1 Method

The regularised solution to the inverse Radon transform can be found by solving

$$(\mathsf{A}^\top \mathsf{A} + \alpha \mathsf{L})\boldsymbol{f}_* = \mathsf{A}^\top \boldsymbol{g} \tag{1}$$

however for large problems direct inversion is infeasible. A matrix-free iterative Krylov solver can sidestep this issue, and one such solver is the LSQR method, which we employ here. Equation 1 can be reformulated as the iterative problem

$$\boldsymbol{f}_{\alpha,L} = \underset{\boldsymbol{f}}{\operatorname{argmin}} \left\| \begin{pmatrix} \mathsf{A} \\ \sqrt{\alpha}\mathsf{L} \end{pmatrix} \boldsymbol{f} - \begin{pmatrix} \boldsymbol{g} \\ \boldsymbol{0} \end{pmatrix} \right\|^2 \doteq \underset{\boldsymbol{f}}{\operatorname{argmin}} ||\mathsf{M}\boldsymbol{f} = \boldsymbol{b}||^2 \tag{2}$$

This can be solved via the LSQR method given augmented (zero-padded) data $\boldsymbol{b}$ and a general linear operator M implementing the (possibly matrix-free) operations $\mathsf{M}\boldsymbol{f}$ and $\mathsf{M}^\top \boldsymbol{b}$. For Zero-Order Tikhonov (ZOT) L is the identity operator I and for First-Order Tikhonov (FOT) the gradient operator $\begin{pmatrix} \nabla_x & \nabla_y \end{pmatrix}^\top$.

There exists the additional problem of determining the magnitude of the regularisation parameter α. If there is an accurate estimate of the noise level σ within the data $\boldsymbol{g}$ then the *Discrepancy Principle* is a suitable method. The method finds an α by solving the non-linear optimisation problem:

$$\mathbf{DP}(\alpha) \doteq \frac{1}{n}\|\boldsymbol{r}_\alpha\|^2 - \sigma^2 = 0$$

where n is the size of $\boldsymbol{g}$ and

$$\boldsymbol{r}_\alpha = \boldsymbol{g} - \mathsf{A}\boldsymbol{f}_{\alpha,L}$$

3.2 Experiments

3.2.1 Varying number of angles

For the first experiment, the Radon transform operator A was created with a detector count (number of projection samples) of 130 over the full *range* of angle (π radians) but a limited *number* of angles - 30, 45 and 60. Hence sinograms $\mathsf{A}\boldsymbol{f}_{\text{true}} = \boldsymbol{g}$ of size 130×30, 130×45 and 130×60 respectively were produced, and Gaussian noise with standard deviation $\sigma = 0.5$ added. The goal is to recover $\boldsymbol{f}_{\text{true}}$. The LSQR method using both ZOT and FOT was used as well as FBP. For the ZOT and FOT the regularisation parameter α was chosen via the Discrepancy Principle *for each* recovery problem (i.e. a different value was found depending on the number of angles used in the forward operator). A comparison of the reconstructions - both in terms of the recovered image and the SSIM/PSNR values is provided in Figure 3.1.

Inspecting the SSIM values for the reconstruction over the varying number of angles, it appears that ZOT and FOT regularisation is superior in performance to FBP. For 30 and 60 angles over the full range, the ZOT reconstructions have the highest SSIM values of 0.64912 and 0.78532. For 45 angles FOT has the highest value at 0.72973. However, the results between ZOT and FOT methods are very similar upon examining both SSIM values and visual quality. The only clear conclusion is that FBP is significantly worse as it contains significantly more 'streaky' artificats over all number of angles and additionally struggles to eliminate noise. This is to be expected since ZOT and FOT can be intrepretated as enforcing regularisation via a smoothness prior, whereas FBP is simply an adjustment to regular backprojection accounting for the 'density' of sampling lines. It is also important to note that the ZOT and FOT methods took significantly more time to converge, so there is a clear trade-off between reconstruction quality and runtime in the case of limited number of angles.

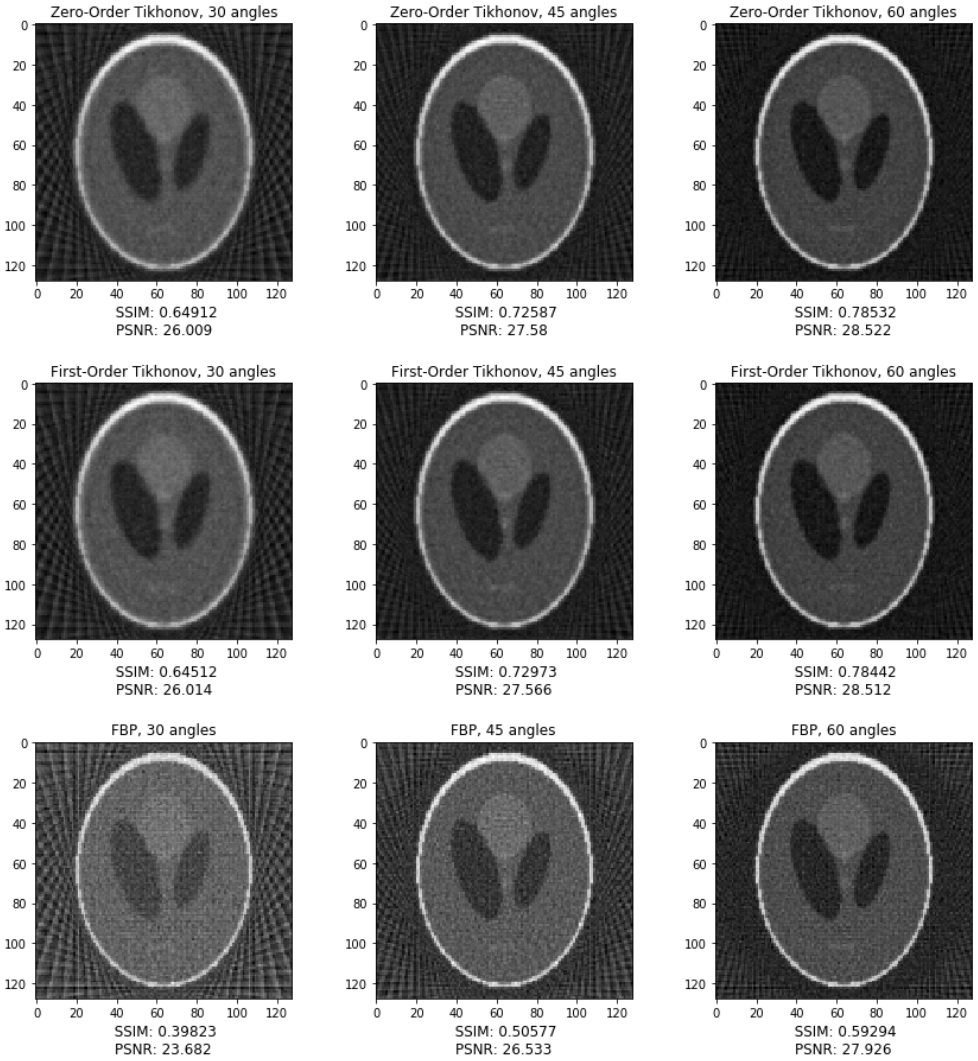


Figure 3.1: Comparison of regularisation methods for varying number of angles.

3.2.2 Varying range of angles

For the second experiment, the detector count remained the same, but the number of angles was held fixed at 180 whilst the range took the values $\frac{\pi}{2}$, $\frac{3\pi}{4}$ and π (full range). The data noise level and methods remained the same as in the first experiment. The results are provided in Figure 3.2.

Again, no clear conclusions on the superiority of a single method can be made, but the analysis remains larger the same. The ZOT and FOT have higher SSIM scores over all ranges (with the reconstructions of both methods being very similar). For the angle range $\pi/2$ FOT produces the highest SSIM score at 0.55012, with ZOT second highest at 0.4529. For the angle range $3\pi/4$ ZOT produces the highest SSIM of 0.59900 with FOT second at 0.51529. For the full angle range ZOT and FOT produce very similar results at 0.83814 and 0.82929 respectively. We can make the tentative conclusion that FOT is superior for angle range $\pi/2$, whereas ZOT is superior for $3\pi/4$ and both methods are roughly equal over the full range. A final important note to take is that the iterative algorithm employing ZOT and FOT regularisation took significantly more compute time than FBP. The time spent to find the optimal α using the discrepancy principle and the iterative procedure of the LSQR algorithm was significantly greater than the FBP method.

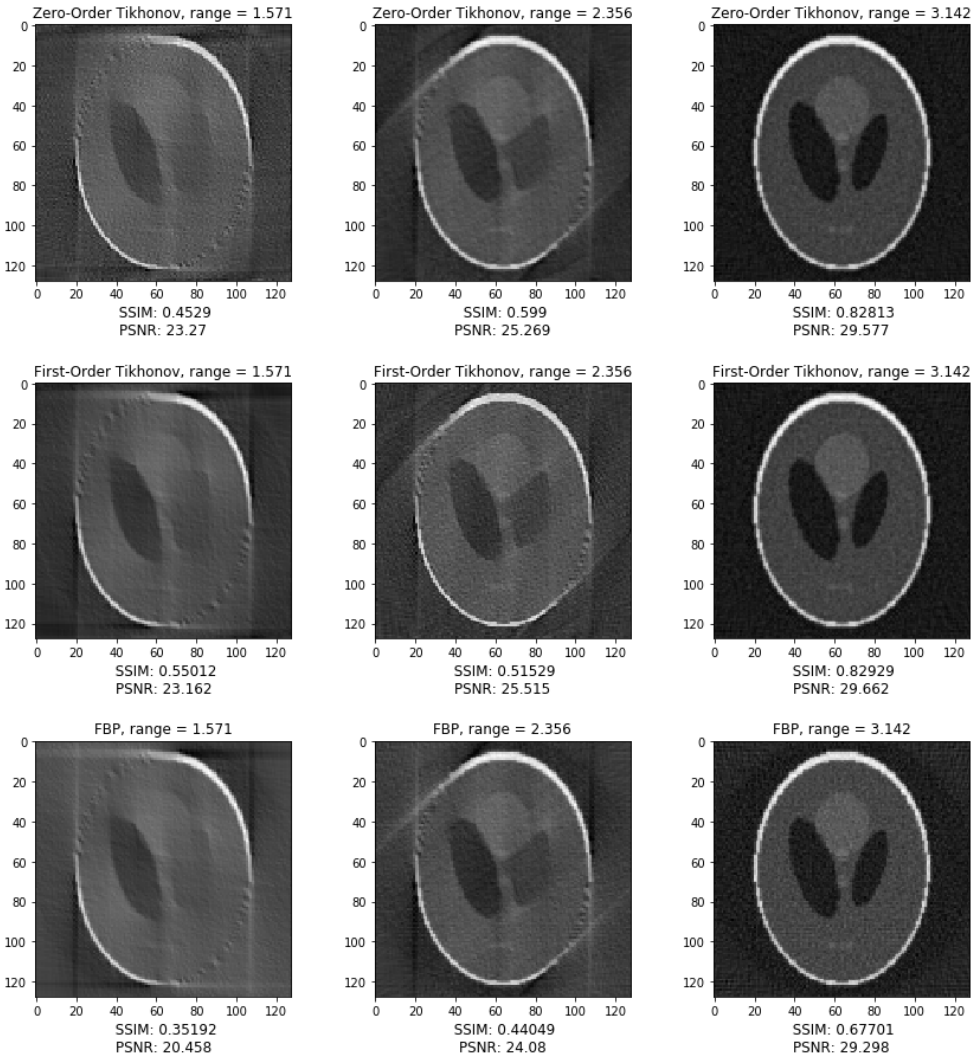


Figure 3.2: Comparison of regularisation methods for limited range.

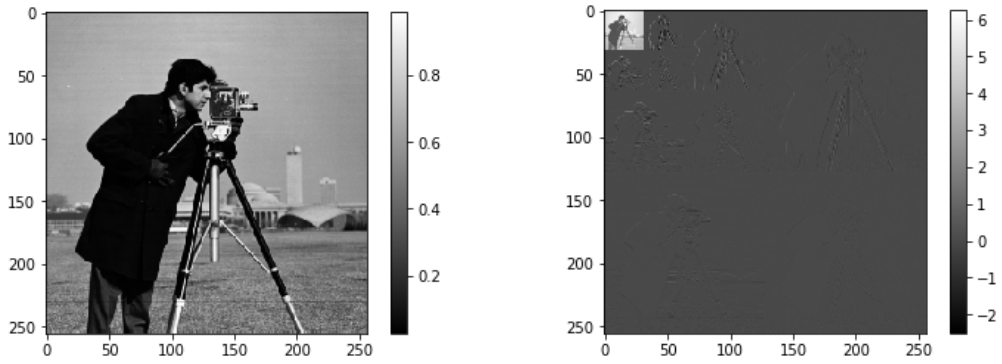
4 Write a Haar wavelet denoiser.

4.1 Examining coefficients

For this section we use the Cameraman image of size 256×256 as $\boldsymbol{f}_{\text{true}}$ shown in Figure 4.1a. In this image there are 2^8 pixels and therefore the Haar wavelet transform can be performed over 8 levels. For illustrative purposes the Haar wavelet coefficients are plotted in Figure 4.1b over only the first 3 levels. In a wavelet transform the original image $\boldsymbol{f}_{\text{true}}$ is decomposed by succesively applying either a high-pass filter (H) or a low-pass filter (L) along any combination of row and column. For instance, we can denote the application of a high-pass filter along the rows and a low-pass filter along the columns as HL. The purpose of the low-pass filter is to compress the 'key' structure in the image - the *low* frequency space, whereas the high-pass filter is used to capture finer details, namely *edges* which occupy *high*-frequency space. In combination then the effect of a HL filter is to isolate the vertical detail, whereas a LH filter isolates the horizontal detail and HH the diagonal.

In the top left of Figure 4.1b we see the effect of applying the LL filter three times on the image, producing the 'approximation' coefficients. Note at each stage of filtering the resolution is reduced by a half and hence the upper left approximation on level 3 is of size 32×32. Conversely in the bottom right a HH filter has been applied just once and hence we see a 128×128 image of coefficients on level 1. The other off-diagonal

coefficients show the application of LH or HL filters. The structure of the entire decomposition for the 3 level case is illustrated in Figure 4.2 (which has been lifted from[1]).



(a) $\boldsymbol{f}_{\text{true}}$ (b) Haar wavelet coefficients

Figure 4.1: Cameraman image and wavelet coefficients.

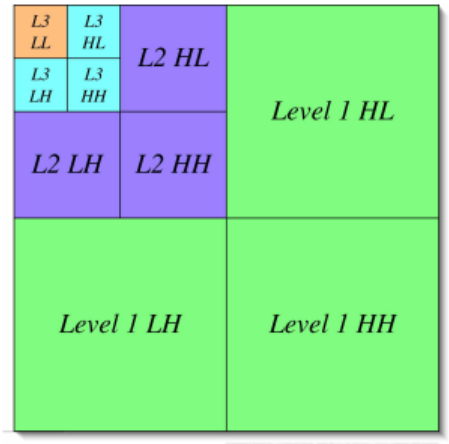


Figure 4.2: Structure of wavelet coefficients

The python package `pywt` was used for this task. The following code calculates the Haar wavelet transform and displays the coefficients -

```
# load in cameraman image
f_true = mpimg.imread('Cameraman256.png')

# calculate maximum level
maxlevel = int(math.floor(np.log2(f_true.shape[0])))

# compute wavelet transform
coeffs = pywt.wavedec2(f_true, wavelet = 'haar', level = maxlevel)

# display coefficients
arr, coeff_slices = pywt.coeffs_to_array(coeffs)
plt.imshow(arr, cmap='gray')
```

4.2 Reconstructing via inverse wavelet transform

To reconstruct the image from the coefficients `coeffs` to find $\boldsymbol{f}_{\text{recon}}$ simply call

[1] https://mil.ufl.edu/nechyba/www/eel6562/course_materials/t5.wavelets/intro_dwt.pdf

```
f_recon = pywt.waverec2(coeffs, wavelet = 'haar')
```

By calculating the normed difference $\|\boldsymbol{f}_{\text{true}} - \boldsymbol{f}_{\text{recon}}\|_2$ we can verify that the reconstruction coincides with the original. This gave a value of $\approx 5.66\text{e}{-5}$ indicating the reconstruction is almost exact.

4.3 Implementing thresholding and denoising

If the observed image $\boldsymbol{g}$ is noisy, the discrete wavelet transform can be used for denoising. The algorithmic process is as follows: first, take the discrete wavelet transform to compute the wavelet coefficients, second, apply thresholding to the coefficients (attempting to eliminate the high-frequency coefficients that correspond to noise), and thirdly, apply the inverse discrete wavelet transform on the thresholded coefficients to recover a less noisy image.

In order to estimate a suitable thresholding level, we sort all the coefficients across scales by absolute value and take the value corresponding to having $x\%$ of coefficients are smaller. The function `est_thresh` was used to accomplish this -

```
def est_thresh(f, x, level):
    """
    Function to estimate the threshold value for a given percentile cut off.
    """
    tc = pywt.wavedec2(f,'haar', level)
    arr, coeff_slices = pywt.coeffs_to_array(tc)
    arr = np.sort(abs(arr.flatten()))
    tVal =  arr[int(arr.size*x)]
    return tVal
```

For a given threshold value, thresholding can be applied to the coefficients for a specific range of levels or scales. A multitiude of thresholding functions such as 'hard' or 'garrote' can be applied on the coefficients, however we chose to use the standard soft thresholding. This was accomplised using the `oywt` library function `pywt.threshold` as well as the code below -

```
def thresholdFunction(coeffs,tRange,tVal):
    coeffsT = []
    coeffsT.append(coeffs[0])
    # loop over all coefficients
    for i in range(1, len(coeffs)):
        # threshold coefficients in the selected levels
        if i in tRange:
            lcoeffs = [pywt.threshold(key, value=tVal, mode='soft') for key in coeffs[i]]
            coeffsT.append(lcoeffs)
        # append all coefficients otherwise
        else:
            lcoeffs = [key for key in coeffs[i]]
            coeffsT.append(lcoeffs)
    return coeffsT
```

Finally, the denoising process as summarised above is given by the function `denoise` specified here -

```
def denoise(data, tRange, tVal, level):
    """
    Denoise an image by taking the Haar wavelet transform, thresholding
    the coefficients then applying the inverse transform.
    Input:
        - data: input image
        - tRange: scales to threshold over
        - tVal: threshold value
        - level: max scale/level of transform.
    """
    # Apply Haar wavelet transform
    coeffs = pywt.wavedec2(data,'haar',level = level)
    # Threshold coefficients
    coeffsT = thresholdFunction(coeffs, tRange, tVal)
    # Apply inverse transform on thresholded coefficients
    denoised = pywt.waverec2(coeffsT,'haar')
```

```
    return denoised
```

4.4 Experiments

Experiments to determine the effectiveness of this procedure were then carried out. Gaussian noise with standard deviation $\sigma = 0.1$ was added to the image $\boldsymbol{f}_{\text{true}}$.

4.4.1 Varying range

The first experiment was to examine how varying the range over which to threshold the coefficients (the different scales of the wavelet coefficients) affected the quality of the denoised image. Three such ranges were tested - full range (thresholding levels 1-8), leaving out the lowest scale (thresholding levels 1-7), and leaving out the highest 4 scales (thresholding levels 5-8). The threshold parameter was estimated using the `est_thresh` function with a cut-off percentile of 90% giving a value of ≈ 0.19, and kept the same for this section. The results are displayed in Figure 4.3.

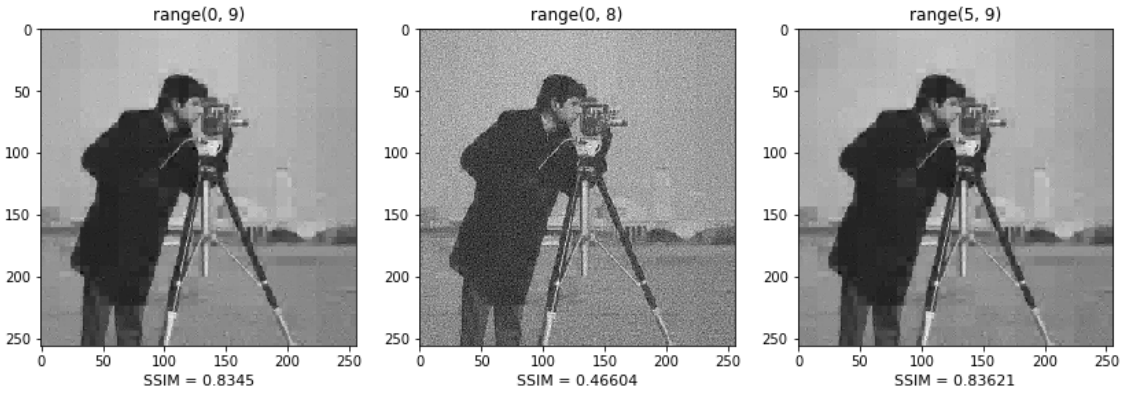


Figure 4.3: Denoising by thresholding over limited ranges of coefficient levels.

The key detail is that if thresholding is not performed on the highest level, the quality of the denoising is extremely poor. This middle denoised image has an SSIM of 0.46604 as well as noticable noisy artifacts and is much worse compared to leftmost image where thresholding was carried out over all levels and with and SSIM of 0.83450. This is to be expected since the noise components are certain to occupy the lowest scale coefficients, so not thresholding over the lowest level is counterproductive. Conversely, if we do not threshold over the highest level (lowest scale) coefficients, as is shown in the rightmost image the denoising is still just as effective. The image has an SSIM of 0.83621 as well as being clearer much smoother in quality.

4.4.2 Varying threshold

The second experiment was to investigate how the threshold parameter affected the performance of the denoising. For this experiment, the full range was used on all tests and three threshold value were obtained from taking the 80th, 90th and 95th percentiles in the `est_thresh` function applied to the image. The results are displayed in Figure 4.4. We note that thresholding at too low a value (80th percentile) or indeed too high a value (95th percentile) can lead to unfiltered noise or excessive 'blocky' smoothing respectively. The threshold value obtained from the 90th percentile seemed to provide a reasonable compromise between the two, leading to the highest SSIM score of ≈ 0.83.

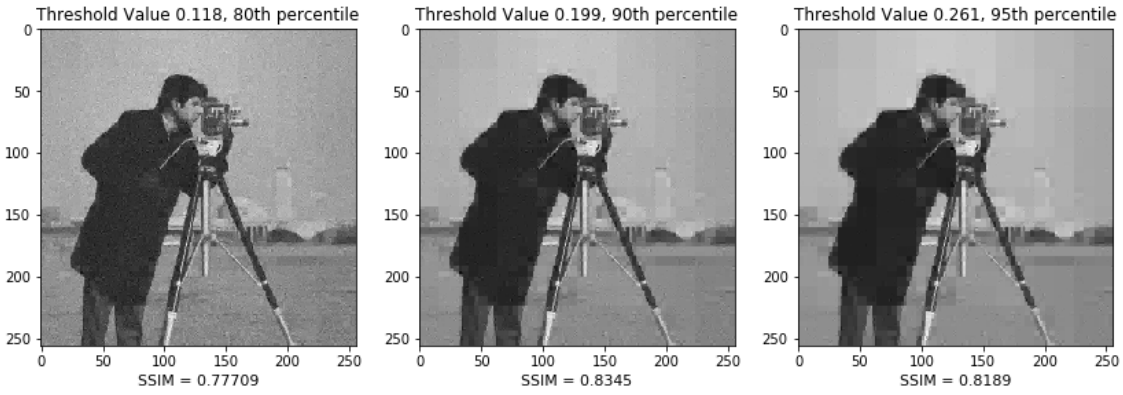


Figure 4.4: Denoising with varying threshold values.

5 Iterative soft-thresholding for X-ray tomography.

5.1 Algorithm

In implementing the ISTA algorithm, the following design choices where made:

- The initial iterate $\boldsymbol{f}_0$ was chosen as the filtered back-projection of the data $\boldsymbol{g}$. This gave the algorithm a sufficient 'warm-start' and is the most sensible choice.
- Two stopping criterion were used. The first was to ensure that the residual of the data fit was always increasing. This is enforced by calculating the residual $\text{res}^{(k)} = \|A\boldsymbol{f}_{\text{recon}}^{(k)} - \boldsymbol{g}\|$ and checking that $\text{res}^{(k)} > \text{res}^{(k-1)}$. The stopping condition safeguards against the case the when the step size parameter λ causes instability. Under a stable scheme, the difference in the reconstructions tend to zero and hence the main stopping condition was to terminate if the norm between the reconstructions $\|\boldsymbol{f}_{\text{recon}}^{(k)} - \boldsymbol{f}_{\text{recon}}^{(k-1)}\|$ fell below some small tolerance level ϵ.
- A non-negativity prior can be introduced a non-negativity constraint and hence an additional regularisation term. The objective function then becomes

$$\frac{1}{2}\|\mathsf{A}\boldsymbol{f} - \boldsymbol{g}\|_2^2 + \alpha\|\mathsf{W}\boldsymbol{f}\|_1 + \mathbb{I}_{[0,\infty)}(\boldsymbol{f})$$

  where $\mathbb{I}_{[0,\infty)}$ is an indicator function defined as

$$\mathbb{I}_{[0,\infty)}(\boldsymbol{f}) = \begin{cases} 0, & \boldsymbol{f} \succeq 0 \\ +\infty, & \text{otherwise} \end{cases}$$

  The proximal operator of this term is simply the projection onto the nonnegative orthant, i.e taking $\max(f_i, 0)$. This proximal operators from each regularisation term can be taken in any order during the thresholding step[2].

5.2 Experiments

For the following experiments we investigated the effect of changing the noise level, number of angles, and angle range on the quality of the reconstructions. Throughout the experiments the detector count (130) was kept the same, and the SSIM used as a tool to quantitively assess the reconstructions.

[2]https://papers.nips.cc/paper/4863-on-decomposing-the-proximal-map.pdf

5.2.1 Varying noise

For the first experiment, sinograms were generated using the full angle range π radians, and 90 angles. Subsequently Gaussian noise of standard deviation $\sigma = 0.0, 0.20, 0.5, 1$ and 2 was added to the sinograms to test how the ISTA reconstructions varied with noise. The results are displayed in Figure 5.1. Despite taking half as many angles as in section 1 - 90 compared to 180 - judged on SSIM scores, in all cases the method improves upon FBP (see Figure 1.4). However, it is important to note that the non-negativity imposed does contribute significantly to this score, and the visual improvement is less significant.

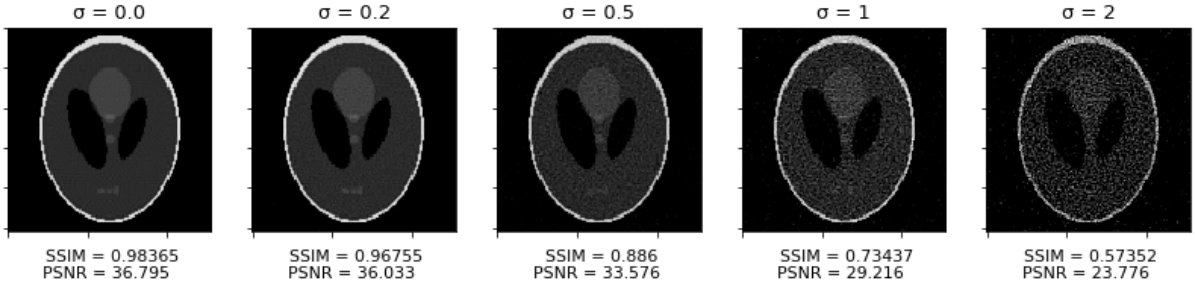


Figure 5.1: ISTA with varying noise

5.2.2 Varying number of angles

For this experiment, the noise level $\sigma = 0.5$ and angle range π remained fixed, and the number of angles took the values 30, 45, and 60. The results are displayed in Figure 5.2. The equivalent experimental results for the Tikhonov and FBP methods are displayed in Figure 3.1.

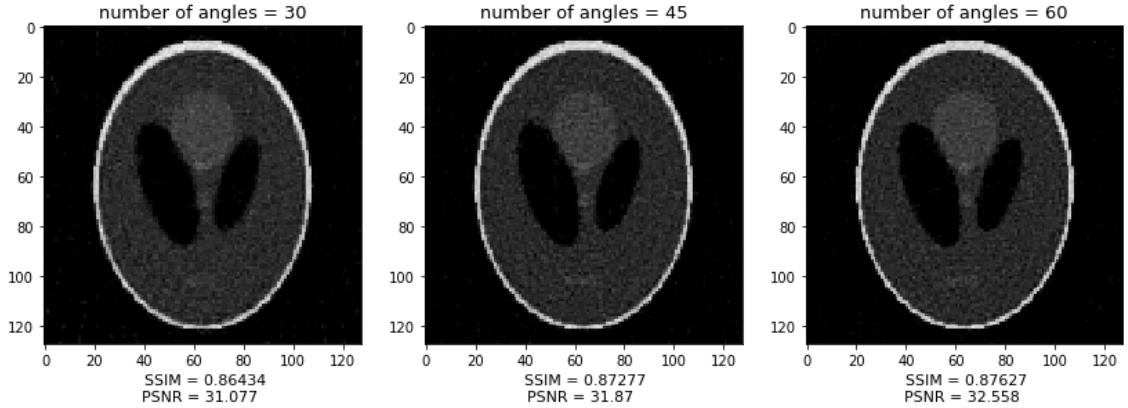


Figure 5.2: ISTA with varying number of angles.

5.2.3 Varying angle range

For the final experiment, the noise level was set $\sigma = 0.5$, the number of angles remained fixed at 90, and the range of angle varied over $\frac{\pi}{2}, \frac{3*\pi}{4}$ and π radians. The results are displayed in Figure 5.3 and the equivalent Tikhonov and FBP results in Figure 3.2. The ISTA algorithm again has higher SSIM score for all ranges. In this experiment it is particularly clear that ISTA is superior at eliminating artifacts as compared to the previous methods. In particular note how the majority of the 'streaky' lines are absent as compared to the Tikhonov and Backprojection methods.

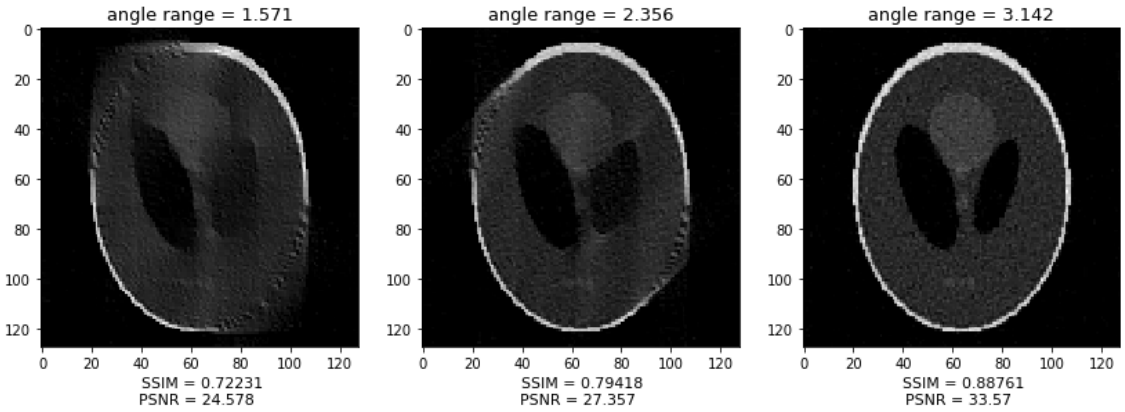


Figure 5.3: ISTA with varying angle range.

6 Advanced Topic: Learning Based Reconstruction

In this section we explore the Deep Learning approach to reconstruction. There exists two frameworks for this approach: initial filtered backprojection and learned post-processing, and learned model-based iterative reconstruction. The PyTorch framework was used for implementation. Both networks are trained in a supervised manner and will therefore need a training dataset.

6.1 Producing a training set

As suggested we write a function `generate_phantom` to generate images of between 5 and 20 ellipses of random location, size and intensity. The function is given below:

```
from skimage.draw import ellipse

  def generate_phantom(dim):
        def generate_ellipse(dim):
          # Generate a single random ellipse
          temp_img = np.zeros((dim, dim), dtype=np.double)
          r, c = np.random.randint(low=dim//8, high=dim - dim//8, size=2)
          r_radius, c_radius = np.random.randint(low=4, high=dim//4, size=2)
          rr, cc = ellipse(r, c, r_radius, c_radius, (dim, dim))
          temp_img[rr, cc] = np.random.uniform(0.3,1)
          return temp_img

    img = np.zeros((dim, dim), dtype=np.double)
    # Superimpose random ellipses
    for _ in range(np.random.randint(5, 21)):
        img += generate_ellipse(dim)
    return (img/np.max(img))
```

We chose to generate 500 such phantoms of size 128×128 which serve as ground-truth target images `y_train`. In this section we will explore the effectiveness of the post-processing and model-based networks in relation to the *limited number of angles* problem. Hence, for the experiments sinograms were generated over the full range of π radians, but only 30 angles and a resolution (detector count) of 130 was used. The sinograms were therefore of size 30×130. Gaussian noise of standard deviation $\sigma = 0.5$ was subsequently added. This setup follows the experimental procedure as in Part A. Examples are given in Figure 6.1

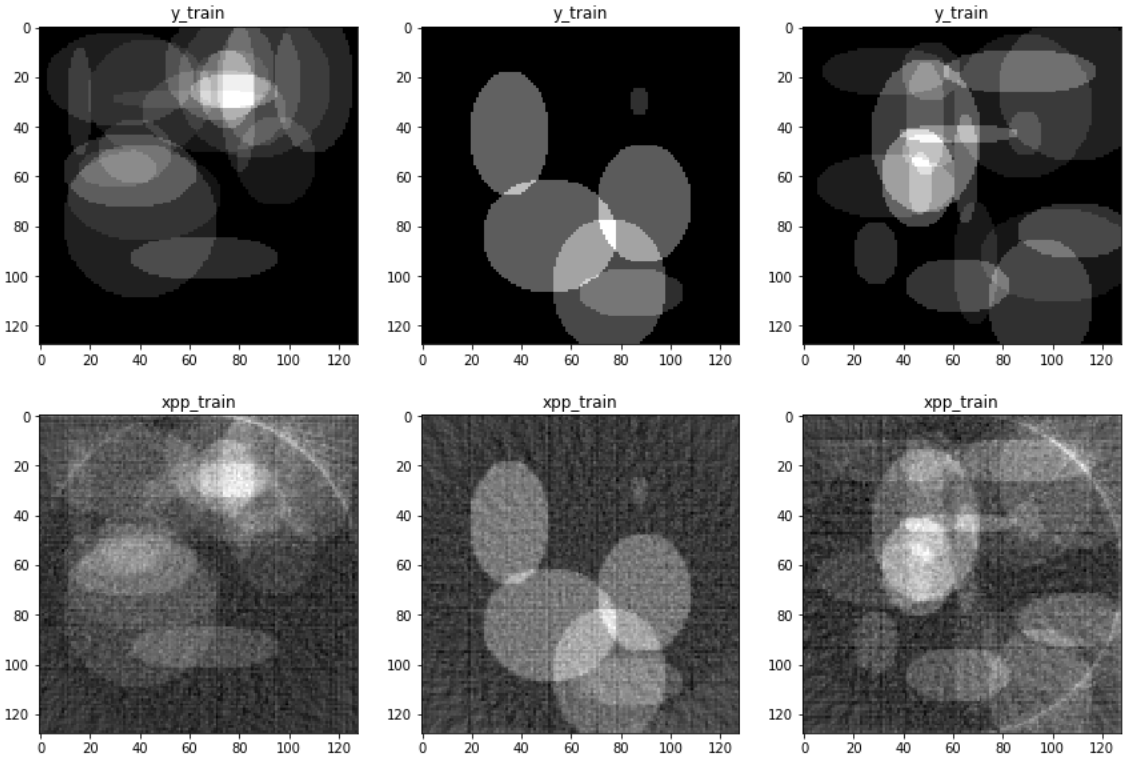


Figure 6.1: Random phantoms `y_train` and filtered back-projections `xpp_train`

6.2 Designing a Post-Processing Network

The task is to design a convolutional neural network G_θ that corrects the initial reconstruction by filtered backprojection. This network will aim to remove the artifacts and noise in the reconstructions as displayed in Figure 6.1. Training is performed to find the optimal set of parametersθ such that the loss function is minimised:

$$\text{loss}(\theta; \boldsymbol{g}) = \|G_\theta(\boldsymbol{f}_0) - \boldsymbol{f}_{\text{true}}\|_p$$

The choice of $p = 2$ was chosen as default. This choice is sensible since squared loss corresponds to maximising the log-likelihood of the data fit assuming Gaussian noise (which is what we have here). However, note that this is not necessarily the best metric for human observers - the MSE for a given image could be lower but the visual inspection may see no improvement. A more sophisticated error functional could be used to reflect this.

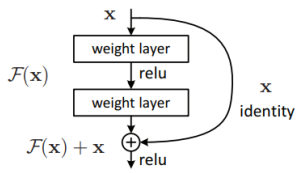


Figure 6.2: Residual block structure

As suggested a simple ResNet-type architecture was implemented. This is described as a "repeated residual block that consists of 2 convolutional layers and an additive residual connection". This block corresponds to 1 iteration and so repeating the block over n times corresponds to n iterations. One requirement therefore is that the size of the input image is the same as the size of the output image. The diagramatic specification of a single block is given in[3] (see Figure 6.2). In particular we let the weights be represented by 3×3 convolutions and choose 32 filters per layer. A standard ReLU activation function is applied between each layer. Finally a residual identity skip connection is applied element-wise at the end of each block. The code to create a single block is given below. For our use case the network receives and outputs a single channel greyscale image and therefore `in_channels = 1`. The Adam optimiser was used with learning rate $\alpha = 0.001$.

[3]Deep Residual Learning for Image Recognition, Kaiming He et al., 2015

```
class Block(nn.Module):
    def __init__(self, out_channels):
        super(Block, self).__init__()
        self.conv1 = nn.Conv2d(1, out_channels, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(out_channels, 1, kernel_size=3, padding=1)
        self.relu = nn.ReLU(inplace=True)

    def forward(self, x):
        x = x.view(-1, 1, dim, dim)
        identity = x
        out = self.conv1(x)
        out = self.relu(out)
        out = self.conv2(out)
        out = out + identity
        out = self.relu(out)
        return out
```

The ResNet can therefore be designed as a sequential stacking of these blocks. The code to accomplish this is as follows:

```
class ResNet(nn.Module):

    def __init__(self, block, blocks, out_channels):

        super(ResNet, self).__init__()
        self._blocks = blocks
        self._layers = nn.ModuleList([block(out_channels) for _ in range(blocks)])

    def forward(self, x, y=None):

        xtemp = x.view(-1, 1, dim, dim)
        for layer in self._layers:
            if y is None:
                x = layer(x)
            else:
                x = layer(x, y)
        return x

model = ResNet(Block, blocks = 8, out_channels = 32)
```

6.3 Designing a Model-Based Network

As suggested in[4], we can also design a partially learned gradient descent algorithm which incorporates the gradient of the data fit $\nabla d(y, Ax_k) = A^*(Ax^k - y)$. The update to x^k can now be written:

$$x^{k+1} \leftarrow G_{\theta_k}(\nabla d(y, Ax_k), x^k)$$

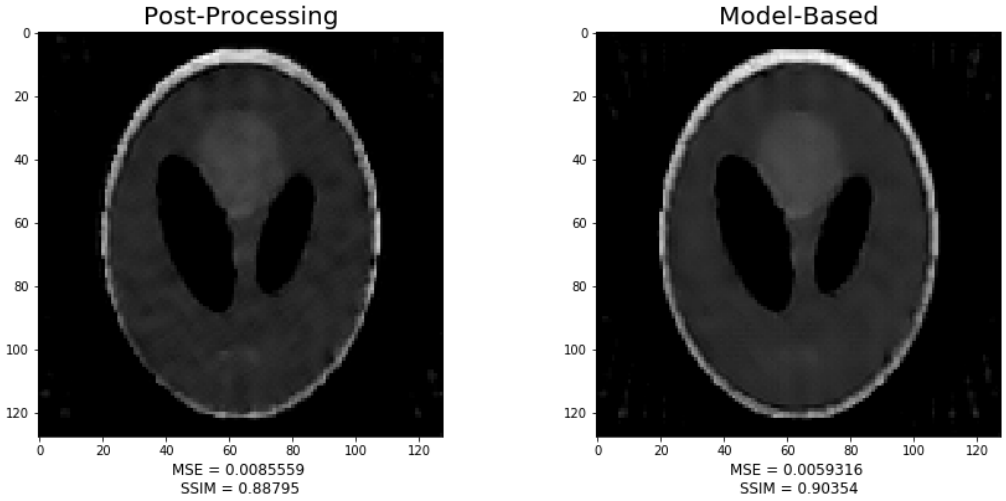
In particular we let the first convolutional layer now receive a two-channel input of the iterate x_k and the gradient term $\nabla d(y, Ax_k)$. The convolutional layers retained 3×3 filters with 32 channels, but now with the first layer receiving two input channels. The second layer outputs another two-channel image and so we slice the first channel to recover a one-channel image. We learn a residual function δx^k and after applying an elementwise residual connection as before the output is the reconstruction $x^{k+1} \leftarrow x^k + \delta x^k$. 8 blocks were stacked to form the network. These choices of hyperparameters were not chosen during learning but were simply selected since they seemed to exhibit good convergence properties and optimal SSIM values. Finally, the `astra` toolkit does not support PyTorch accelerated tensor computations and hence the Radon transform operator A was instantiated as a discrete matrix tensor as in Section 2.1. We can use the ResNet type structure defined above by simply redefining the Blocks to compute and concatenate the gradient of the data fit in the forward pass. The code given below accomplishes this task.

[4]Solving ill-posed inverse problems using iterative deep neural networks, Jonas Adler et al., 2017

```
class ModelBlock(nn.Module):

    def __init__(self, out_channels):
        super(ModelBlock, self).__init__()
        self.conv1 = nn.Conv2d(2, out_channels, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(out_channels, 2, kernel_size=3, padding=1)
        self.relu = nn.ReLU(inplace=True)

    def forward(self, x, y):
        xt = x.view(x.size(0), -1)
        # Data gradient term
        t1 = torch.transpose(A, 1, 0)
        t2 = torch.transpose(xt, 1, 0)
        t3 = torch.mm(A, t2) - torch.transpose(y, 1, 0)
        t4 = torch.mm(t1, t3)
        t4 = t4/t4.max(t4)
        gradx = t4.view(-1,1,dim,dim)

        # Concatenate identity and gradient of data fit
        x = x.view(-1, 1, dim, dim)
        identity = x
        update = torch.cat((x, gradx), 1)

        # Apply Convolutional layers
        update = self.relu(self.conv1(update))
        update = self.conv2(update)
        x = self.relu(identity + update[:,0:1,:,:])

        return x
```

6.4 Results & Comparison

The reconstructions of the Shepp-Logan phantom by the post-processing and model based networks are displayed in Figure 6.3, 6.4 and 6.5. In Figure 6.3 the final iterates (reconstructions) of both networks are displayed along with their SSIM and MSE scores. Furthermore we also visualise the outputs of the networks at the 'iterations' 2, 4, 6 and 8 (i.e. the iterates of the respective layers in the neural net) in Figures 6.4 and 6.5. This shows how the reconstructions gradually increase in quality deeper into the networks.



(a) Post-Processing (b) Model-Based

Figure 6.3: Final Iterates

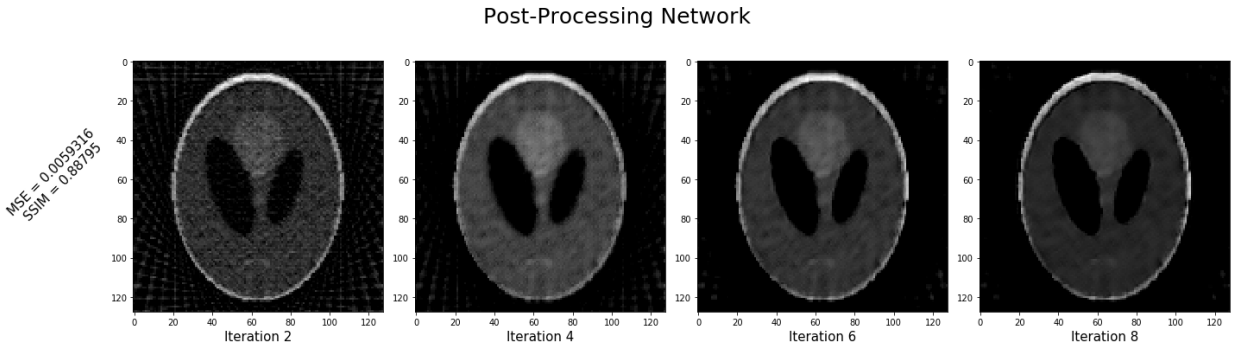


Figure 6.4: Post-Processing Reconstructions over Iterates 2, 4, 6 and 8

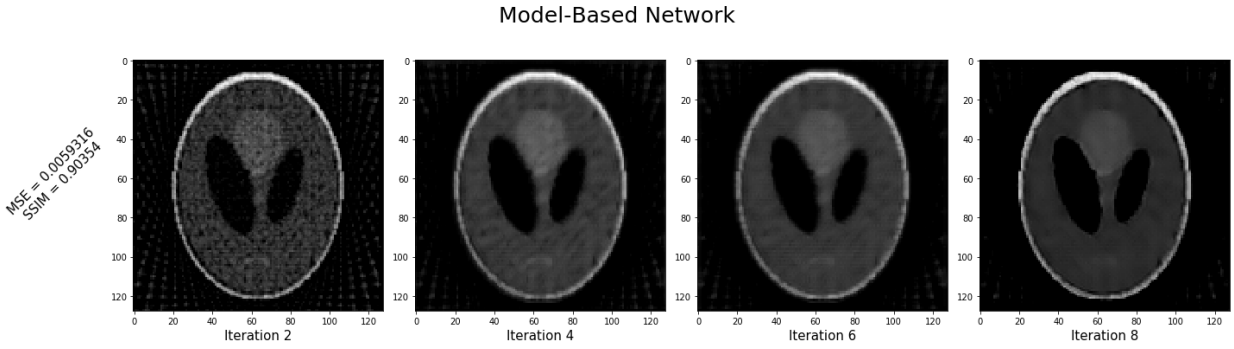


Figure 6.5: Model-Based Reconstructions over Iterates 2, 4, 6 and 8

On both metrics the Model-based network outperforms the Post-Processing one. This is to be expected since it is taking account the physics of the forward operator A. The SSIM of the final reconstruction (Iterate 8) for the Post-Proccessing model was 0.88795 whilst the SSIM of the Model-Based network was 0.90354. Additionally the MSE for the Post-Processing network was 0.0085559, whilst the MSE of the Model-Based was lower at 0.0059316. On visual inspection we also see that the result is slighty better.

Finally we rank the quality of the reconstructions for the limited number of angles (30) problem of all the methods included in this report: Filtered Back-Projection, Zero-Order Tikhonov, First-Order Tikhonov, Iterative Soft-Thresholding, Post-Processing and Model-Based Learned Networks. Again, we use the SSIM as the metric. The relevant reconstructions have been displayed throughout this report.

SSIM reconstruction scores	
FBP	0.39823
FOT	0.64512
ZOT	0.64912
ISTA	0.86434
Post-Processing	0.88795
Model-Based	**0.90354**