B31XO Group Project Report

Group 9:
Yining Ding, Thomas Hodgson, Matthew Holden, Johanna Järvsoo,
Rene Lohmann, Jonathan Spence, Michael Tang

December 2020

1 Introduction

Computational imaging is a branch of applied mathematics which focuses on the development of novel techniques to reconstruct image data from physical measurements. Prominent examples can be found in various fields such as medicine or astronomy: in Magnetic Resonance Imaging (MRI), instead of using an optical measuring device, the physical measuring process generates a set of Fourier coefficients of the probed area such as a patient's body part. Methods from computational imaging can then be used to generate an image from these Fourier samples. Similarly, images can be constructed from the electromagnetic measurements taken by radio astronomy observatories that probe the sky.

Although the Nyquist sampling theorem can guarantee the perfect reconstruction of a continuous signal (such as an image) from discrete measurements provided that the sampling rate is high enough, the time, energy and memory demands associated with satisfying this sampling rate requirement can often be prohibitively expensive. During a typical MRI scan, a patient is required to lie still within a narrow tube for a period between 20 and 60 minutes [6]. Reducing the duration of the scan would not only improve the patient experience, but could also enable MRI scans of many more patients, including those whose condition or age prevent them from lying still for long periods without sedation.

Another famous example of sub-Nyquist sampling was in the celebrated image of the black hole M87* produced by the Event Horizon Telescope collaboration [1], which was constructed from data observed by a network of radio telescopes located across the globe. In this case the sampling procedure was restricted by the fixed physical locations of the small number of telescopes in the network, and the expense of building telescopes at new locations. As a final example, consider a space probe that takes images of other planets. Taking enough samples to satisfy the Nyquist sampling theorem in order to reconstruct a high resolution image could be taxing on the power supply of the probe, and even if it wasn't: sending great amounts of data back to Earth is bound to take a lot of time.

Current research in computational imaging thus deals with the question of whether images can be reconstructed even in cases where there are not enough measurements to satisfy the sampling theorem. Utilizing theory and methods from the fields of signal processing, convex optimization, and machine learning, the aim is to develop new reconstruction techniques so that high-quality images can be reconstructed from signals sampled at a sub-Nyquist rate.

The report is structured as follows: in Section 2 the mathematical framework is outlined. Section 3 gives descriptions and algorithms for a four different denoising techniques. In Section 4 the techniques are applied to a real dataset and finally Section 5 compares them.

2 Objectives

The aim of this project is to compare various methods for the recovery of signals from sub-Nyquist sampled and noisy measurements, where the focus lies on the reconstruction quality as well as the computational efficiency. The signals we consider are MR images of size 320×320. They are taken from the fastMRI dataset [18], provided to us as a training set of 7000 samples and a test set of 20 samples.

A measured signal $\boldsymbol{y}$ is then obtained from an image through the forward model

$$\boldsymbol{y} = \boldsymbol{\Phi}^\dagger \boldsymbol{x}^0 + \boldsymbol{n}, \tag{1}$$

with $\boldsymbol{x}^0 \in \mathbb{R}^N$ the flattened image (i.e. $N = 320^2$), $\boldsymbol{\Phi}^\dagger \in \mathbb{C}^{M \times N}$ the measurement operator, and $\boldsymbol{n} \in \mathbb{C}^M$ the additive noise. The measurement operator is given by $\boldsymbol{\Phi}^\dagger = \boldsymbol{M}\boldsymbol{F}$, where $\boldsymbol{F} \in \mathbb{C}^{N \times N}$ denotes the discrete Fourier transform and $\boldsymbol{M} \in \{0,1\}^{M \times N}$ denotes a selection matrix. This mirrors the sampling of an MRI scanner in that all low frequencies below a threshold are captured, beyond which they are sampled uniformly at random. This is represented by a central band of non-zero unit elements and then uniformly random columns of non-zero elements elsewhere, to select $M < N$ frequency components. Both real and imaginary part of $\boldsymbol{n}$ are sampled from a normal distribution with zero mean and a standard deviation set to a level based on the norm of the true image $\boldsymbol{x}^0$ so that the observed signal to noise ratio was similar for each image, $\sigma = \left\|\boldsymbol{x}^0\right\|_2 10^{-3/2}$, to emulate the noise seen in real-world MRI data.

3 Methodology and Implementation

This section describes 4 different methods to reconstruct an image from sub-Nyquist sampled and noisy measurements of its Fourier coefficients. The first, **M1**, is a pure optimisation method based on the ADMM algorithm. **M2** is a pure deep learning method exploiting the U-net architecture of a CNN. **M3**, and **M4** aim for a hybrid approach, known as a "Plug-and-Play" method, which aim to utilise the power of state of the art denoising engines within optimisation algorithms. The reconstruction results as well as a comparison of the performance of the different methods follow in Section 4 and Section 5.

3.1 M1-ADMM

The first method consists of a convex optimization scheme. Due to the sub-Nyquist measurement, the inverse problem to obtain the original image $\boldsymbol{x}^0$ from measurement $\boldsymbol{y}$ given by (1) is ill-posed: Even without additive noise, there are infinitely many solutions. In order to obtain a unique solution $\boldsymbol{x}^0$, one needs to regularize the problem.

A typical approach is to impose a sparsity condition on the signal, i.e. the assumption that the signal $\boldsymbol{x}^0$ is sparse in some sparsity basis $\boldsymbol{\Psi}$. This means that the representation $\boldsymbol{\alpha} = \boldsymbol{\Psi}^\dagger \boldsymbol{x}^0$ of the signal in the basis given by the columns of $\boldsymbol{\Psi}$ only has few nonzero entries. It can be shown (cf. [12], [4]) that the solution of the so-called basis-pursuit problem (BP)

$$\boldsymbol{x}^* = \mathrm{argmin}_{\boldsymbol{x}} \left\|\boldsymbol{\Psi}^\dagger \boldsymbol{x}^0\right\|_1 \quad \text{subject to } \left\|\boldsymbol{y} - \boldsymbol{\Phi}^\dagger \boldsymbol{x}^0\right\|_2 \leq \epsilon \tag{2}$$

leads to the recovery of $\boldsymbol{x}^0$ with overwhelming probability – up to an error that is governed by the noise level and (possible) non-sparsity of the signal for a sufficient number of measurements M. The smallest admissible M for this problem is determined by the coherence between $\boldsymbol{\Phi}$ and $\boldsymbol{\Psi}$, and by the signal size N.

An iterative method to solve problem (2) is a relaxed version of the Alternating Direction Method of Multipliers (ADMM, originally derived in its general form in [2]), given by the steps

$$\begin{aligned}
\boldsymbol{x}_{t+1} &= \boldsymbol{\Psi}\mathrm{shrink}\left(\boldsymbol{\Psi}^\dagger\left(\boldsymbol{x}_t - \delta \mathrm{Re}\left\{\boldsymbol{\Phi}\left(\boldsymbol{s}_t + \boldsymbol{n}_t - \overline{\boldsymbol{\nu}}_t\right)\right\}\right), \delta\rho^{-1}\right) && (3)\\
\boldsymbol{s}_{t+1} &= \boldsymbol{\Phi}^\dagger \boldsymbol{x}_{t+1} - \boldsymbol{y} && (4)\\
\boldsymbol{n}_{t+1} &= \mathcal{P}_{B(0,\epsilon)}\left(\overline{\boldsymbol{\nu}}_t - \boldsymbol{s}_{t+1}\right) && (5)\\
\overline{\boldsymbol{\nu}}_{t+1} &= \overline{\boldsymbol{\nu}}_t - \boldsymbol{s}_{t+1} + \boldsymbol{n}_{t+1}. && (6)
\end{aligned}$$

The shrinkage operator in (3) is defined as

$$\operatorname{shrink}(x,\lambda) \coloneqq \max(0,|x|-\lambda)\operatorname{sgn}(x) \text{ for } x \in \mathbb{R}, \lambda > 0,$$

and is applied componentwise. $\boldsymbol{s}_t$ and $\boldsymbol{n}_t$ are help variables, $\overline{\boldsymbol{\nu}}_t$ is a Lagrange multiplier introduced in the derivation of the algorithm, $\delta > 0$ a constant that constrols the stepsize[1], and ρ a hyperparameter that can be tuned to improve the quality of the results.

The operator $\mathcal{P}_{B(0,\epsilon)}$ in (5) is a projection onto the hyberball $B(0,\epsilon)$, given by

$$\mathcal{P}_{B(0,\epsilon)}(\boldsymbol{z}) = \min(\epsilon, \|\boldsymbol{z}\|_2)\frac{\boldsymbol{z}}{\|\boldsymbol{z}\|_2} \text{ for } \boldsymbol{z} \in \mathbb{R}^N, \epsilon > 0.$$

The initial values $\boldsymbol{x}_0$ and $\overline{\boldsymbol{\nu}}_0$ are set to zero vectors. The algorithm stops if either one of two convergence criteria is met:

$$\frac{\|\boldsymbol{x}_{t+1}-\boldsymbol{x}_t\|_2}{\|\boldsymbol{x}_{t+1}\|_2} \leq \alpha_1 \quad \wedge \quad \left\|\boldsymbol{y}-\boldsymbol{\Phi}^\dagger\boldsymbol{x}\right\|_2 \leq \epsilon \tag{7}$$

$$\left|\frac{\left\|\boldsymbol{\Psi}^\dagger\boldsymbol{x}_{t+1}\right\|_1 - \left\|\boldsymbol{\Psi}^\dagger\boldsymbol{x}_t\right\|_1}{\|\boldsymbol{\Psi}^\dagger\boldsymbol{x}_{t+1}\|_1}\right| \leq \alpha_2 \quad \wedge \quad \left\|\boldsymbol{y}-\boldsymbol{\Phi}^\dagger\boldsymbol{x}\right\|_2 \leq \epsilon \tag{8}$$

Here $a \wedge b = \min(a,b)$. If none of the criteria were met the algorithm was stopped after a maximum number of iterations, typically after a few thousand. For all created measurement images the noise level ϵ was chosen to be $\epsilon = \sigma\sqrt{M+2\sqrt{M}}$. The sparsity basis Ψ was given as the orthonormal Db4 wavelet basis, where the best results were obtained by setting the decomposition level to 8.

3.2 M2-Modified U-Net

The second method adopts emerging deep learning techniques to recover MR images from their sub-Nyquist noisy measurements. The principle behind supervised deep learning is to use a deep neural network to learn a highly non-linear mapping between input and output. The word 'deep' specifically means that the network has more than one hidden layer. The learning process is performed by tuning network parameters to minimise a loss function on a set of data (i.e. training data) in an 'automatic' manner called backpropagation. To achieve learning and evaluation, the available set of data is usually divided into three subsets, namely a training set, a validation set and a test set. The training set is used to learn the network parameters, the validation set is used to choose hyperparameters and to prevent overfitting, and the test set is used to evaluate the network's performance when responding to unseen data.

For this particular problem, we adapt an existing network structure from FBPConvNet [9], which originates from U-Net [13]. The network structure we use is depicted in Figure 1. This network is a convolutional neural network (CNN), which has been proved to be a very powerful tool in image processing, thanks to its structure akin to biological processes in a human vision system.

Unlike a CNN used for an object detection task whose output consists of class label probabilities together with bounding box locations, our network deals with an image-to-image regression problem. That is, it takes a noisy backprojected image as input and outputs an image of the same dimension attempting to remove the effect of noise and to reconstruct the original fully sampled image. The loss function to minimise is the mean square error over the data between the reconstructed and the ground truth images. As can be seen in Figure 1, the network contains multiple hidden layers performing the following operations: convolution (Conv), batch normalisation (BN), rectified linear unit (ReLU), max pooling, transposed convolution (Transposed Conv), concatenation (Concat) and skip connection (addition). Next, we will give a brief introduction of each of these operations.

Convolution - The convolutional layer essentially makes a CNN differ from a plain neural network (i.e. a multilayer perceptron), in which activations in each hidden layer have connections to all activations in the previous layer and in

[1]For technical reasons, we have $\delta \leq \sigma_{\max}^{-1}\left(\operatorname{Re}\left(\boldsymbol{\Phi}\boldsymbol{\Phi}^\dagger\right)\right)$ where $\sigma_{\max}(\boldsymbol{A})$ denotes the largest eigenvalue of $\boldsymbol{A}$. δ is set to 1 in all experiments.

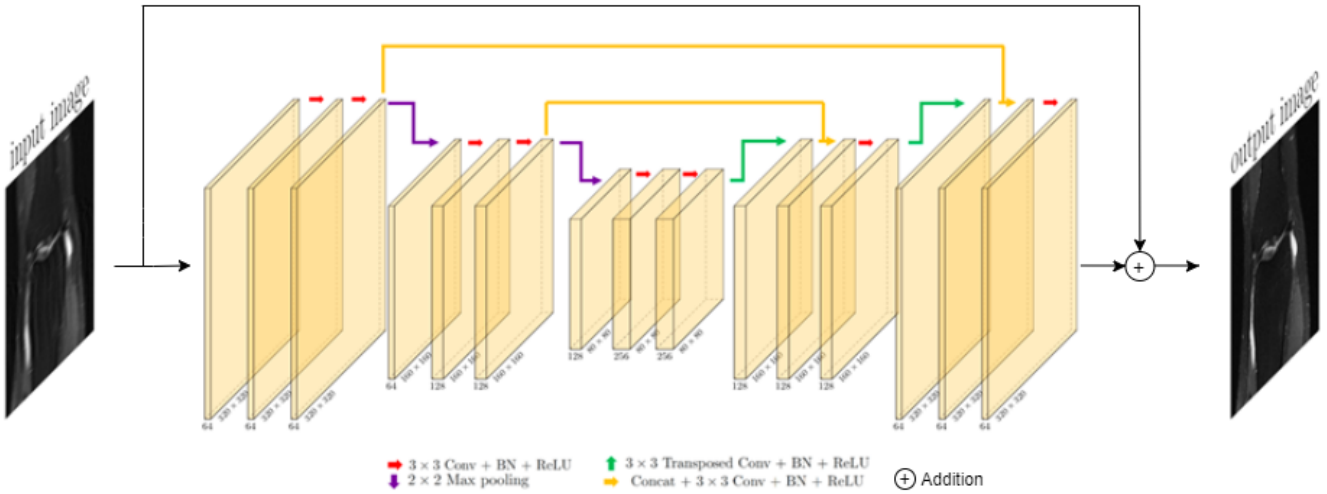

Figure 1: M2: Network structure

the following layer. In contrast, a convolutional layer comprises parameters which form a number of spatial filters. Each filter spatially convolves with the input and generate a 2D matrix. These 2D matrices are then concatenated in the third dimension to form channels as the input to the next layer. To prevent the 2D matrices from shrinking in size after the spatial convolution, zero-padding is usually performed beforehand. Assuming the image in each input channel is square and has size i by i, the square filter has size f by f, p zeros are padded on each side and the stride number is s, then the output will be of size $\lfloor (i + 2p - f)/s \rfloor + 1$ by $\lfloor (i + 2p - f)/s \rfloor + 1$. In our network, $p = 1$, $f = 3$ and $s = 1$ so the output size is the same as the input size. After training, the filters will be learnt to activate in response to certain patterns (e.g. edges, corners) in its input. This convolution architecture has two major differences from the fully connected architecture: 1) shared parameters - a feature detector which is useful in one part of the image tends to be useful in another part of the image; 2) sparse connections - each output value in each layer only depends on a small number of input values. These two attributes together make a CNN suitable in dealing with image and vision tasks, whose input dimension is usually very high and prone to overfitting. Reducing network parameters can help alleviate the issue.

Batch normalisation - Batch normalisation in a CNN normalises each activation map across all samples in a mini-batch by subtracting the mean then dividing by the standard deviation. More specifically, the normalisation is done per activation map (i.e. channel), that is, the mean and standard deviation is calculated among all activations in a mini-batch, over all locations. For instance, if the feature map dimension is i by i and there are m samples in the mini-batch, then the effective mini-batch size will be $m \times i^2$. After that, the normalised values are scaled then offset by a pair of parameters, which are learnt during the training process [8]. Therefore, the number of parameters to learn in a batch normalisation layer will be twice the number of channels in that layer. A batch normalisation layer usually sits between a convolutional layer and a non-linear activation layer (e.g. ReLU). It helps improve the training speed, reduce sensitivity to the network initialisation and regularise network parameters.

ReLU - ReLU is a type of non-linear activation functions which is widely used in deep neural networks. The non-linearity is crucial in making a deep neural network learn complex functions, without which the network would only learn linear mapping regardless of the number of hidden layers. ReLU has the following mathematical representation $f(x) = \max(x, 0)$ and this operation is performed element-wise. There exists other non-linear activation functions such as the sigmoid and the hyperbolic tangent. An advantage ReLU has over the others is that its gradient does not always plateau to zero when deviating from the origin, which help with gradient descent. The gradient of a ReLU activation layer can also be calculated more efficiently than other more complex activation functions, improving the speed of training.

Max pooling - The pooling layer is used to spatially decrease the dimension of intermediate images (hence the number of parameters) to reduce the computational time and make detected features more robust. This is done by extracting a single value from a patch in an image then moving to a next patch and repeating the process. Max pooling simply extracts the maximum value in a patch. There are other types of pooling such as average pooling.

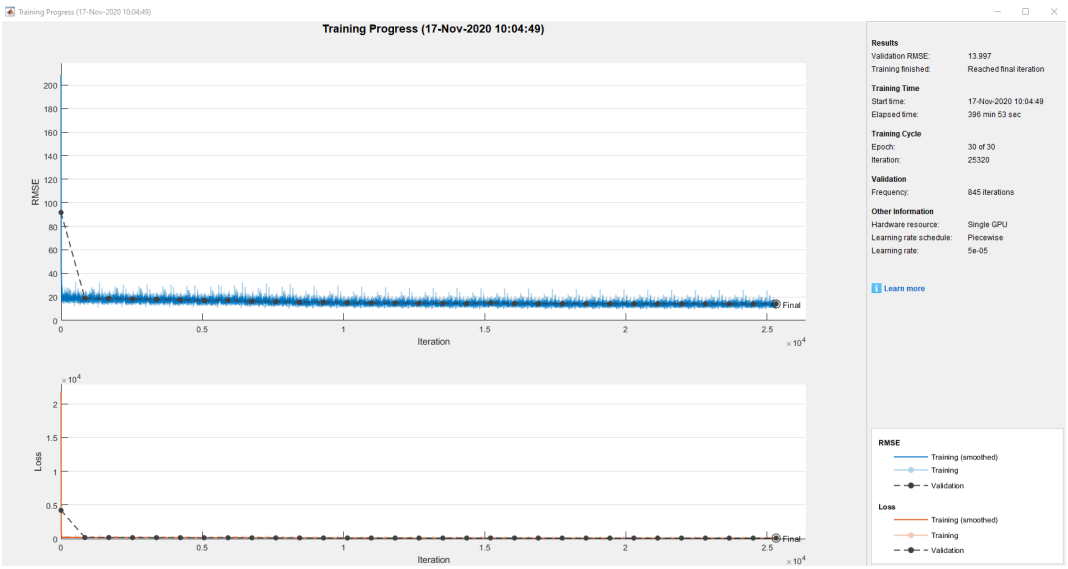

Figure 2: M2: Training plot

Transposed convolution - A transposed convolution layer is used to upsample the activation maps, that is, to spatially increase the size. In our network, all transposed convolutional layers are used to double the size of the input so that to allow concatenation of channels.

Concatenation - The concatenation operation simply concatenates activation maps to increase the number of channels. Note that the activation maps to concatenate have to be of the same size.

Skip connection - There is a skip connection (i.e. addition) directly between the input and the output in our network. This explicitly makes the network learn the difference between the input and the output. Skip connection refers to a connection between a layer and another layer deeper in the network. This technique has been shown to be able to ease optimisation and improve the network's performance, especially when the network is very deep [7].

To train our network, we first randomly shuffle the images in the training set. Then we divide them into two subsets: the first 6,759 images (i.e. 95%) are used for training purpose, and the last 356 images (i.e. 5%) are used for validation purpose (n.b. our validation set is only used to monitor the training process to prevent overfitting). To improve the training speed, we normalise every image such that its pixel values range between 0 and 1 before the training. The training options are shown as following:

- 'adam'
- 'MaxEpochs' - 30
- 'MiniBatchSize' - 8
- 'InitialLearnRate' - 5e-3
- 'LearnRateSchedule' - 'piecewise'
- 'LearnRateDropPeriod' - 10
- 'LearnRateDropFactor' - 0.1
- 'Shuffle' - 'never'

In addition, the validation frequency is set such that the validation is performed per epoch.

The training plot is shown in Figure 2.

3.3 M3-PnP ADMM

The third method used was a Plug-and-Play ADMM scheme [16]. This method is very similar to the ADMM algorithm of **M1**. The key insight in the development of the Plug-and-Play method is to notice that the proximal step (3) in the ADMM algorithm can be interpreted as applying a denoising operator, which takes as input a noisy image, and returns a less noisy image. Plug-and-Play methods replace the proximal step, which minimises an optimisation problem with an explicit regularisation term, with a state-of-the-art image denoiser which does not correspond to an optimisation problem. Examples of such denoisers include non-local means [3], Block-matching and 3D filtering (BM3D) [5] and, more recently, convolutional neural networks [19].

Denoting the denoising operator by $\mathcal{G}$, the Plug-and-Play ADMM algorithm is given by the steps

$$\begin{aligned} \boldsymbol{x}_{t+1} &= \mathcal{G}\left(\boldsymbol{\Psi}^{\dagger}\left(\boldsymbol{x}_t - \delta \mathrm{Re}\left\{\boldsymbol{\Phi}\left(\boldsymbol{s}_t + \boldsymbol{n}_t - \overline{\boldsymbol{\nu}}_t\right)\right\}\right)\right) && (9)\\ \boldsymbol{s}_{t+1} &= \boldsymbol{\Phi}^{\dagger}\boldsymbol{x}_{t+1} - \boldsymbol{y} && (10)\\ \boldsymbol{n}_{t+1} &= \mathcal{P}_{B(0,\epsilon)}\left(\overline{\boldsymbol{\nu}}_t - \boldsymbol{s}_{t+1}\right) && (11)\\ \overline{\boldsymbol{\nu}}_{t+1} &= \overline{\boldsymbol{\nu}}_t - \boldsymbol{s}_{t+1} + \boldsymbol{n}_{t+1}. && (12) \end{aligned}$$

where δ and ϵ are subject to the same conditions as in Section 3.1: $\delta \leq \sigma_{\max}^{-1}(\mathrm{Re}(\Phi\Phi^{\dagger}))$ and $\epsilon \leq \sigma\sqrt{M + 2\sqrt{M}}$.

Since Plug-and-Play schemes replace the proximal step with a denoiser, they no longer correspond to an explicit optimisation problem. As a result, one is not able to evaluate an objective function to test convergence. Although a restrictive class of denoising algorithms [14] have been proven to converge, in general there is no guarantee of convergence, and the algorithm is instead terminated after a fixed number of iterations.

By applying a denoising operator within an optimisation scheme, Plug-and-Play methods avoid computationally intensive requirement to train a new neural network for different forward operators and noise levels. Instead, the same denoiser can be applied for any measurement operator Φ, which is incorporated into the iterative scheme in steps (9) and (10). This allows Plug-and-Play methods to retain the flexibility of convex optimisation schemes such as ADMM while also taking advantage of the empirical success of deep learning methods.

For the method which we denote by **M3**, a convolutional neural network known as DnCNN [20] was used as the denoising operator. The structure of the network is depicted in Figure 3, and consists of 17 layers, where each layer combines a 2D convolution, batch normalisation and ReLU layer activation function. The architecture was modified slightly from the depicted image: the input to the network is a noisy image, and the output of the 17th layer is an estimate of the noise contained in the input image. Using a skip connection from the input layer, the noise estimate can then be removed from the noisy input image to return a denoised estimate of the clean image.

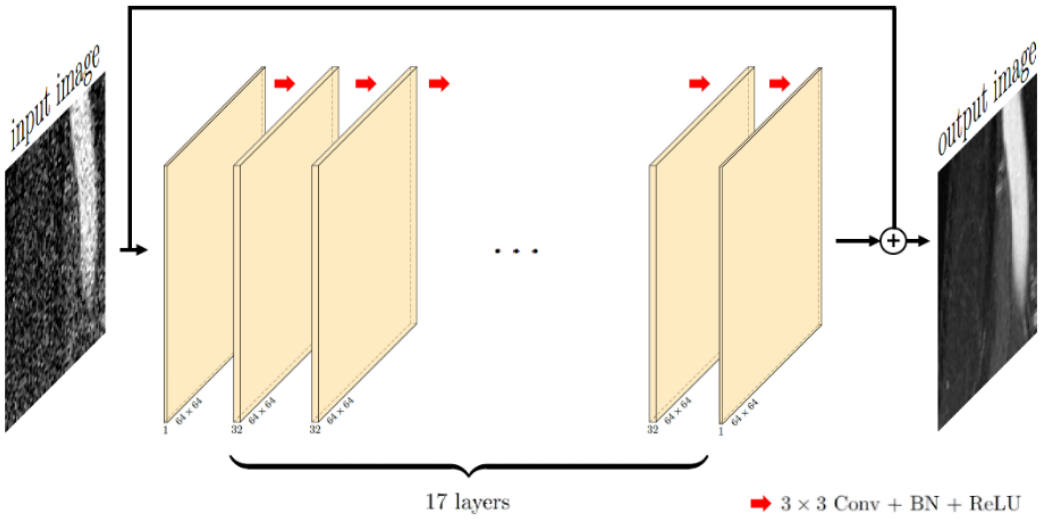

Figure 3: M3: Network structure

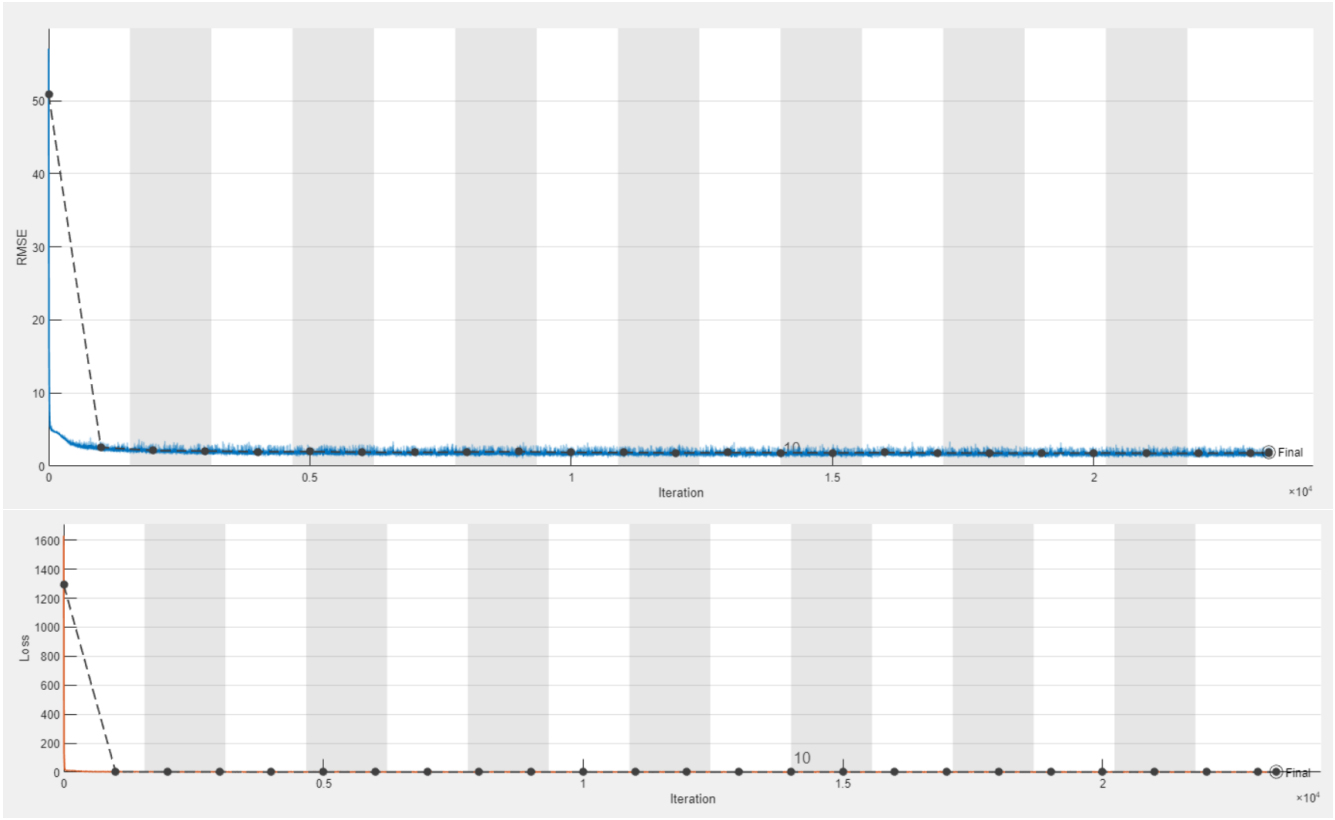

Figure 4: M3: Training of DnCNN algorithm over 15 epochs

For computational efficiency, the network was trained on image patches of size 64 by 64. This network can then be applied to the full-sized images, since the convolutions defined by the weights learned in training can be applied to images of any size. So long as the noise in the training data is i.i.d., the performance of a network trained on image patches should not suffer when applied to the full-sized images [20].

The training data was partitioned into training and validation sets as in 3.2. Training was then carried out using 30 patches per image, where the noise added to each image had standard deviation of 0.07, with the following options:

- 'adam'
- 'MaxEpochs' - 15
- 'MiniBatchSize' - 64
- 'InitialLearnRate' - 3e-4
- 'LearnRateSchedule' - 'piecewise'
- 'LearnRateDropPeriod' - 3
- 'LearnRateDropFactor' - 0.5
- 'SquaredGradientDecayFactor',0.99
- 'Shuffle' - 'every-epoch'

The training results are displayed in Figure 4

3.4 M4-Beyond M1, M2 and M3

The PnP-ADMM algorithm in Section 3.3 relies on the neural network $\mathcal{G}$ to perform the denoising operation of the shrinkage operator in (3). An advantage of using the neural network is that we are no longer required to enforce sparsity in a transform domain represented by Ψ in (3). However, successful neural networks require huge amounts of training time. Therefore, we also consider a version of PnP-ADMM using firstly, publicly available pretrained networks, and secondly, a state of the art denoising algorithm in place of $\mathcal{G}$ in (9)-(12).

3.4.1 PnP-ADMM with Pretrained Networks

With limited computing resources available, we adapt the PnP-ADMM methods described in Section 3.3 by using publicly available pretrained denoising CNNs. The implementation of these networks requires the use of MatConvNet in MATLAB [15].

Further to this a mixed method was employed, whereby images were recovered by PnP-ADMM using the IRCNN denoising network [20] and then final artefacts were removed by a final denoising using DnCNN3, [19]. The final validation RMSE was found to be 1.6572.

3.4.2 ADMM with Effective Denoiser

We consider the Block-Matching 3D (BM3D) filtering algorithm, described in [10] [11][2]. BM3D denoises images by splitting similar patches of the image into 'blocks' of 3D data, and filtering these blocks to enforce sparsity in a transform domain. Each pixel may occur in several different blocks so the results must be averaged over all occurrences. Results using BM3D as a denoiser in ADMM are presented in Section 4.4.1.

4 Validation and Results

In this section we apply the methods described above to a noisy subset of the fastMRI dataset [18]. The performance of each method is evaluated by three metrics, the signal-to-noise ratio (SNR), structural similarity index measure (SSIM),and the computation time. If $\boldsymbol{x}^*$ is a reconstruction of the true image $\boldsymbol{x}^0$, the reconstructed signal-to-noise ratio (SNR)is defined as

$$SNR_{\boldsymbol{x}^0}(\boldsymbol{y}) \coloneqq 20 \log_{10} \left(\frac{\left\| \boldsymbol{x}^0 \right\|_2}{\left\| \boldsymbol{x}^0 - \boldsymbol{y} \right\|_2} \right).$$

The SSIM [17] measures the similarity between two images, and its more complex definition is omitted here. The computational time clearly depends on the hardware used and valid comparisons can only be made by referring to the same machine. Throughout, neural networks were trained using a Nvidia GTX 1070 with 8GB GDDR5. Most other computations were performed on an Intel Core i5-8350U CPU – where this is not the case, it has been noted in the text.

4.1 M1

We employ the ADMM routine from Section 3.1 to measurements of the 20 images in the test data set. In order to obtain the best possible reconstruction quality we measure the mean of SNR and SSIM of the recovered images for various values of ρ, see Figure 5.

As can be seen, there is no obvious global maximum for the SNR or SSIM in the examined regime for ρ. The strong oscillations are an artifact of the noise in the measurement process in combination with the small sample size (20

[2] An implementation of the BM3D algorithm can be found at `http://www.cs.tut.fi/~foi/GCF-BM3D/index.html#ref_problems`

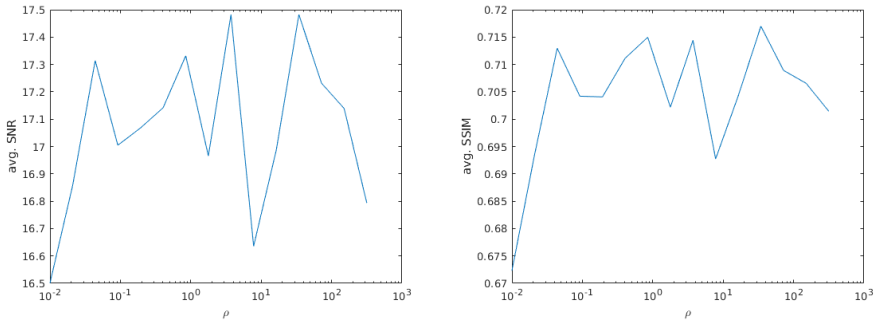

Figure 5: M1: Average SNR (left) and SSIM (right) measured over the test data set for various values of ADMM hyperparameter ρ.

images): Using different sequences of random numbers to create the measurements leads to significant differences in the obtained curves SNR(ρ) and SSIM(ρ).

Although we repeated the search on a finer ρ grid, the best possible ρ we found belongs to one of the two largest peaks in the average SNR plot of Figure 5, and we chose the one that leads to smaller SNR variance: $\rho = 3.7276$. This ρ value leads to mean SNR 17.482 dB, standard deviation SNR 0.9394 dB, mean SSIM 0.71441, standard deviation SSIM 0.046806, and a mean computation time for a single image of 32.014 s with standard deviation 5 s.

Three examples of recovery results are given in Figure 6. For each sample, the original image, the noisy backprojected image (i.e. the real part of $\boldsymbol{\Phi y}$), and the recovered image are shown.

4.2 M2

The validation and test results of **M2** are presented in Sections 4.2.1 and 4.2.2 respectively.

4.2.1 M2 Validation

As can be seen from Figure 2, the validation root-mean-square error (RMSE) gradually decreases and there is no obvious sign of overfitting. The final validation RMSE is 14.114. The SNR and SSIM results of the backprojected and reconstructed images in the validation set are shown in Table 1 and Table 2, respectively. As can be seen, both the mean SNR and the mean SSIM significantly increase, while their standard deviations decrease.

	SNR (dB)	**SSIM**
mean	7.996	0.633
std	4.343	0.078

Table 1: M2: Validation results of backprojected images

	SNR (dB)	**SSIM**
mean	15.041	0.782
std	3.051	0.068

Table 2: M2: Validation results of reconstructed images

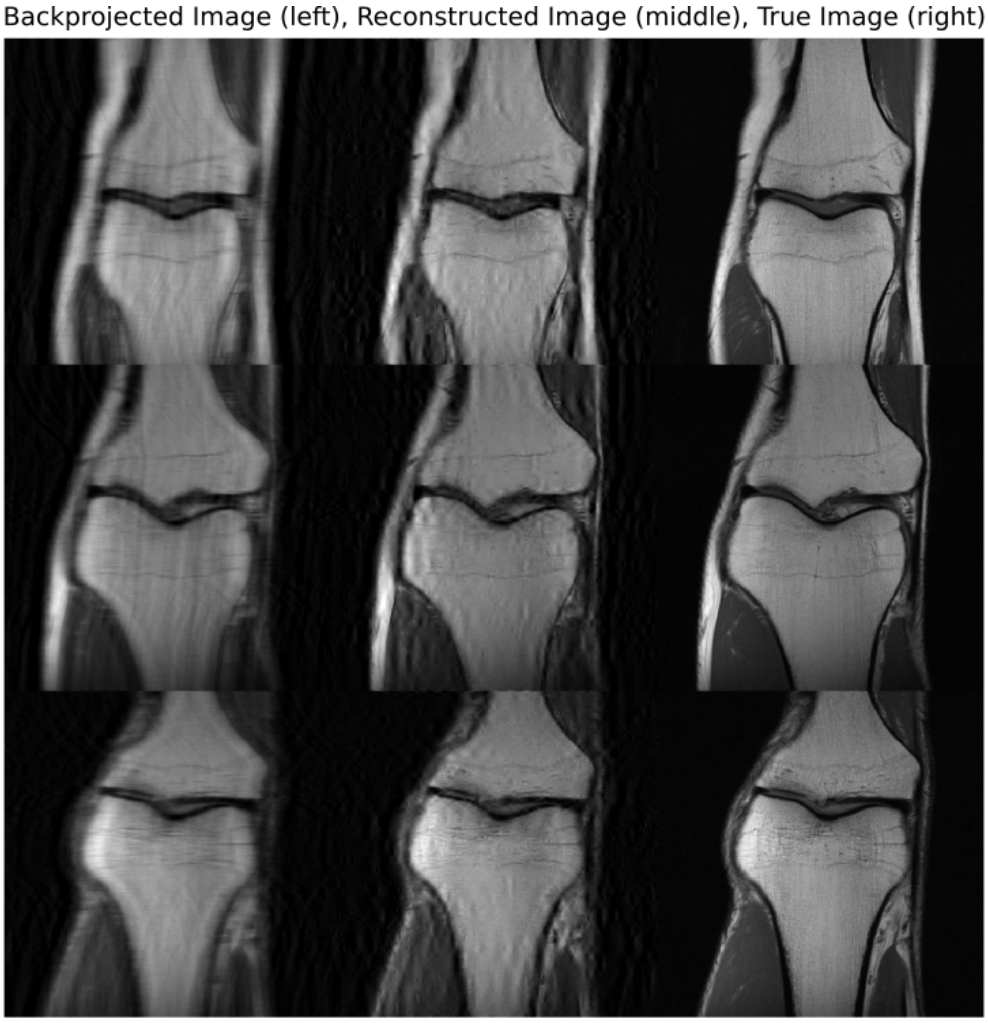

Figure 6: M1: Recovery results. 3 sample images of the test data set to ADMM hyperparameter $\rho = 3.7276$.

4.2.2 M2 Test

The learnt network is then supplied with noisy backprojections of the twenty images in the test set. The SNR and SSIM results of the backprojected and reconstructed images are shown in Table 3 and Table 4, respectively. Three sample results are shown in Figure 7. As can be seen from Table 3 and Table 4, both the mean SNR and the mean SSIM significantly increase, while their standard deviations decrease. Observations from Figure 7 are in line with the results in the tables.

	SNR (dB)	**SSIM**
mean	7.956	0.590
std	4.243	0.084

Table 3: M2: Test results of backprojected images

	SNR (dB)	**SSIM**
mean	14.917	0.774
std	3.270	0.050

Table 4: M2: Test results of reconstructed images

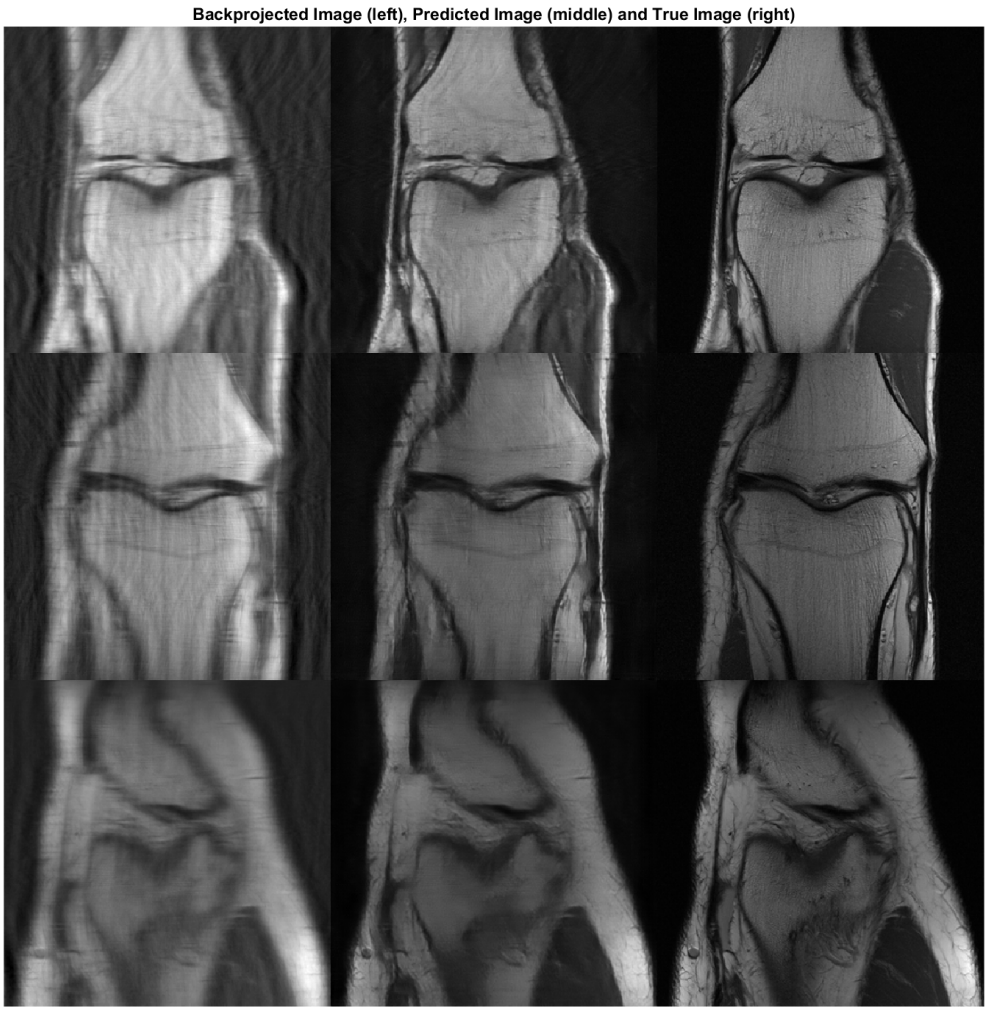

Figure 7: M2: Recovery results. 3 sample images of the test data

4.3 M3

The results obtained by Plug-and-Play ADMM using the trained DnCNN network improve on the results of both **M1** and **M2**. It was observed that results were stable and did not improve after 100 iterations. The results are displayed in Table 8, and illustrative examples are displaying in Figure 9. Visually, the reconstructions look similar to the true images, although textures and other details on fine scales are not always preserved.

	Run Time (s)	std	SNR	std	SSIM	std
PnP-ADMM	72.283	1.835	22.408	1.730	0.864	0.030

Figure 8: Reconstruction quality 100 iterations using trained DnCNN with $\sigma = 0.07$

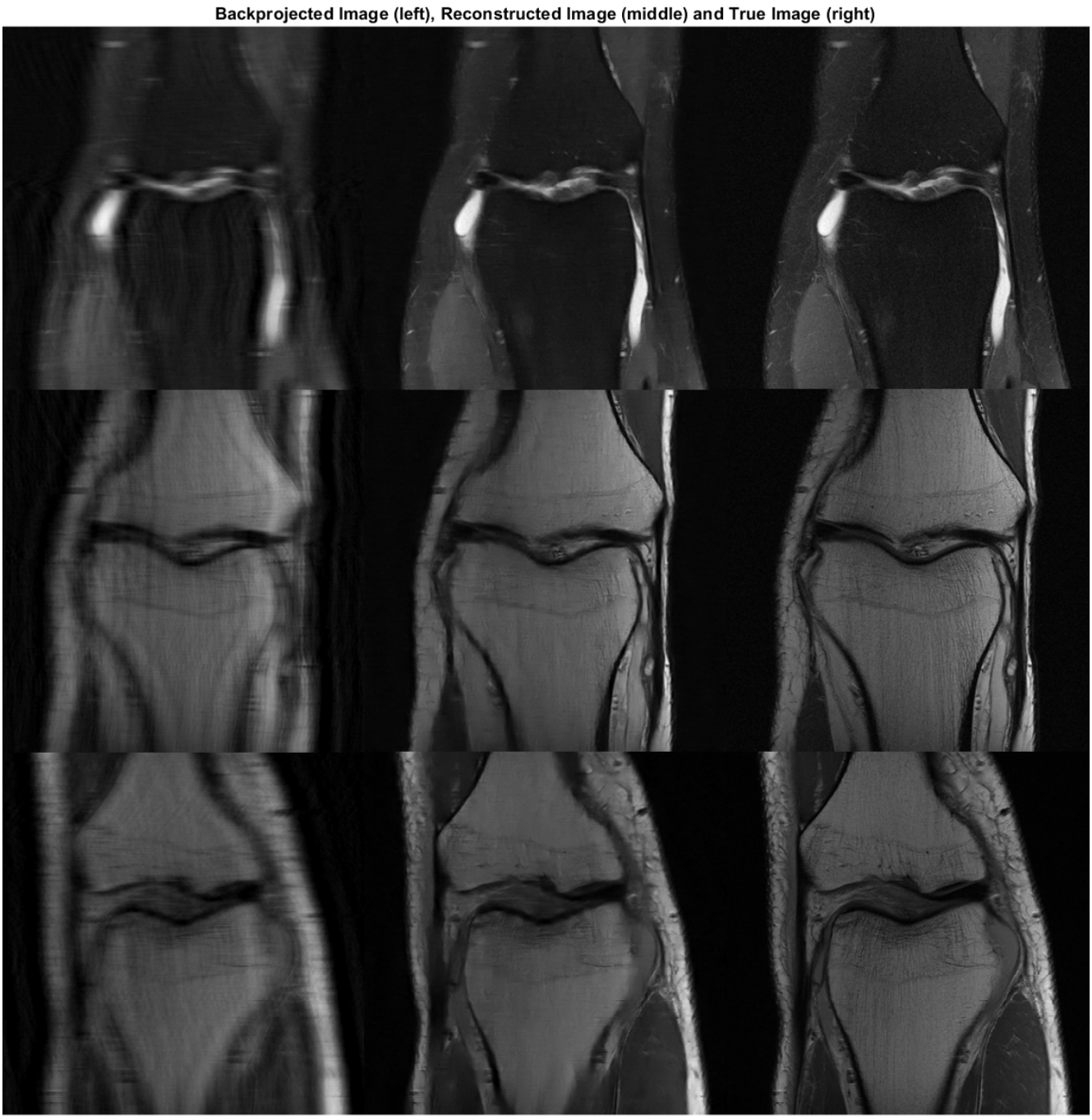

Figure 9: M3: Recovery results. 3 sample images of the test data

4.4 M4

By using a pretrained network available online in the PnP-ADMM algorithm, we were able to improve our results. The network is described in [20], and the same architecture trained with different noise levels is available. We use the network trained with noise level $\sigma = 8$, and refer to it as IRCNN. Recovery results are described in Table 11 and Figure 10. The large run time is mainly due to implementation via MatConvNet.

Some rough artefacts can be seen in the recovered image in Figure 10. These can be removed by an additional application of another (different) publicly available denoising network, DnCNN3 [19]. The results are listed as "PnP-ADMM + Denoiser" in Table 11. This marks a marginal improvement - much less than the standard deviations.

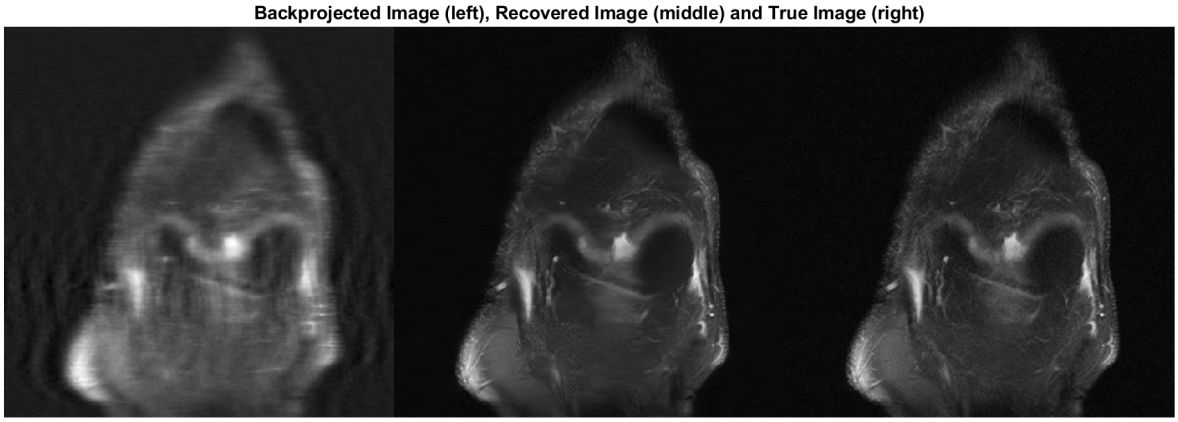

Figure 10: **M4** - PnP-ADMM without additional denoising step.

However the images appear better to the naked eye.

	Run time(s)	std	SNR	std	SSIM	std
PnP-ADMM	174.90	9.40	23.5073	1.8766	0.87779	0.035478
PnP-ADMM + Denoiser	183.1204	10.25	23.5112	1.8672	0.87791	0.035334

Figure 11: Reconstruction quality after 250 iterations using IRCNN with $\sigma = 8$, [20] and additional denoiser DNCNN3, [19]. Performed using an i7 processor.

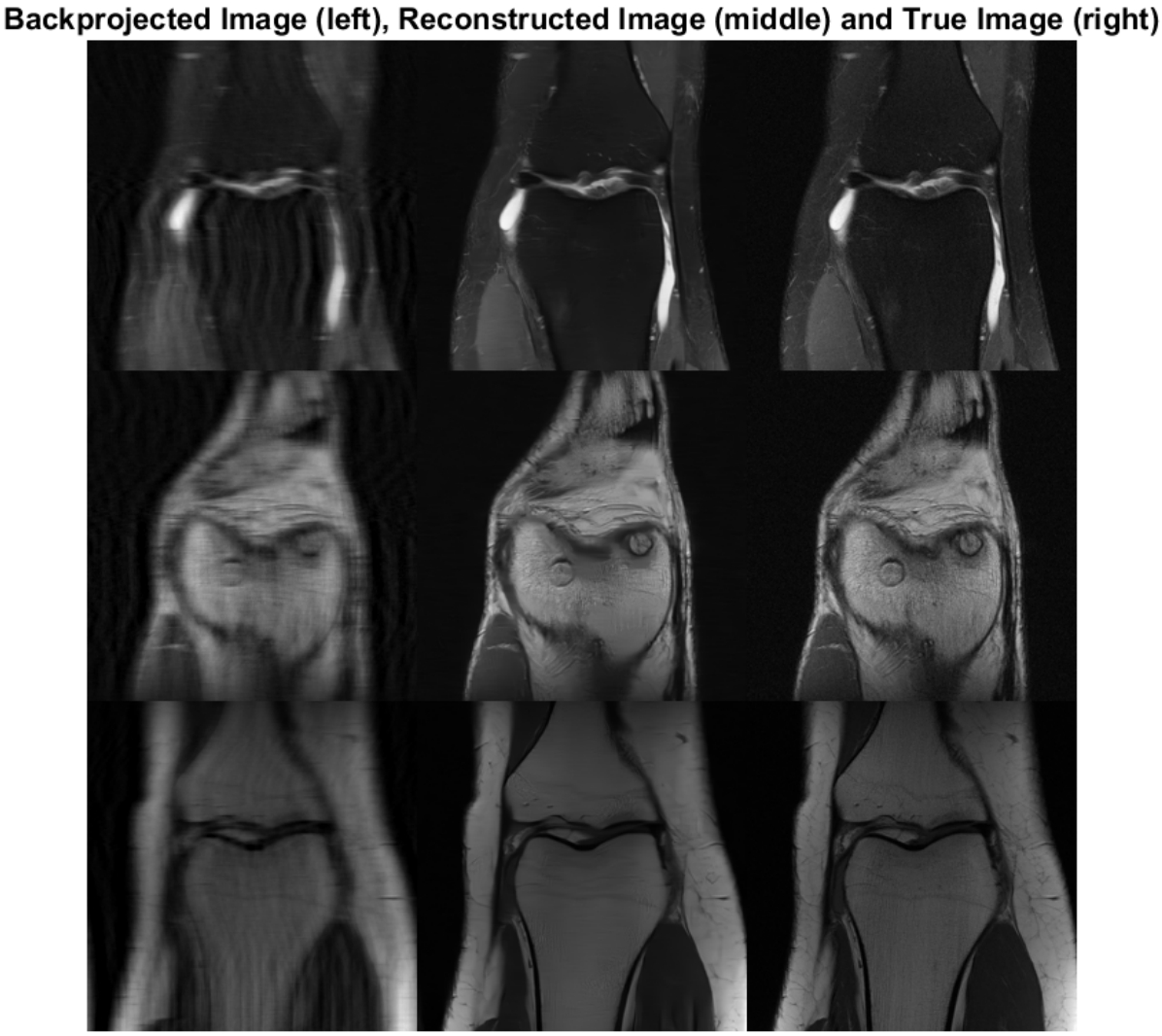

Figure 12: **M4** - Three example recovery images using PnP-ADMM with an additional denoiser. Backprojected Image (left), Recovered Image (middle) and True Image (right).

4.4.1 PnP-ADMM using BM3D

Using the BM3D algorithm in place of the neural network for the PnP-ADMM algorithm provides comparable results to the PNP-ADMM using the pretrained network. Results are shown after 50 iterations of ADMM. Typical image reconstructions are shown in Figure 13 and statistics over the test set of 20 images are shown in Table 5. The recovered images are almost of the same quality as the PnP algorithm using IRCNN. Note that there is a larger runtime when using BM3D than IRCNN, despite using one-fifth of the number of iterations. The BM3D algorithm is much more costly to run each iteration than a neural network. However, BM3D does have the advantage of not requiring an offline training stage.

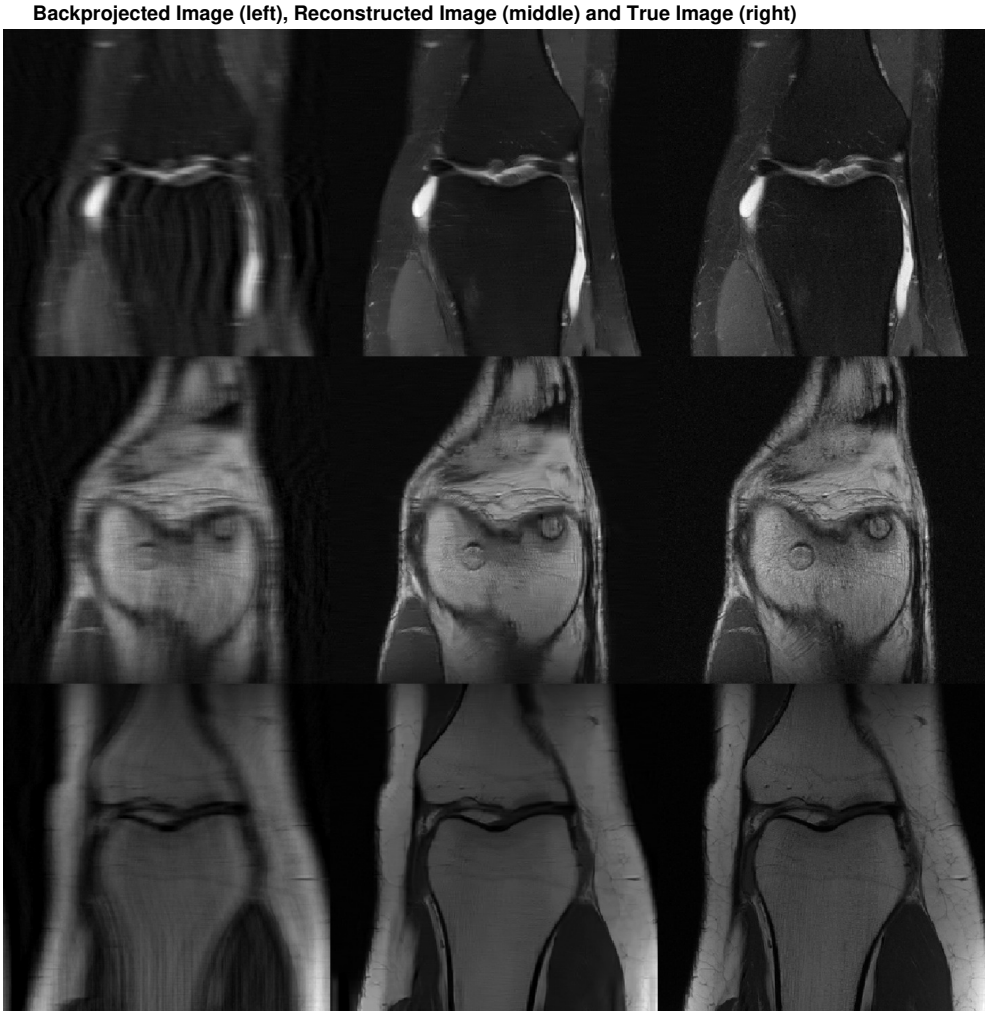

Figure 13: Typical reconstructions for PnP-ADMM using BM3D.

	Run time(s)	SNR	SSIM
Mean	257.4468	23.1389	0.8748
Standard Deviation	5.8005	1.6815	0.0304

Table 5: Statistics for PnP-ADMM using BM3D

4.5 Results Summary

Table 6 summarises results of all methods and Figure 14 illustrates a ground truth image with reconstructed images using the five methods proposed.

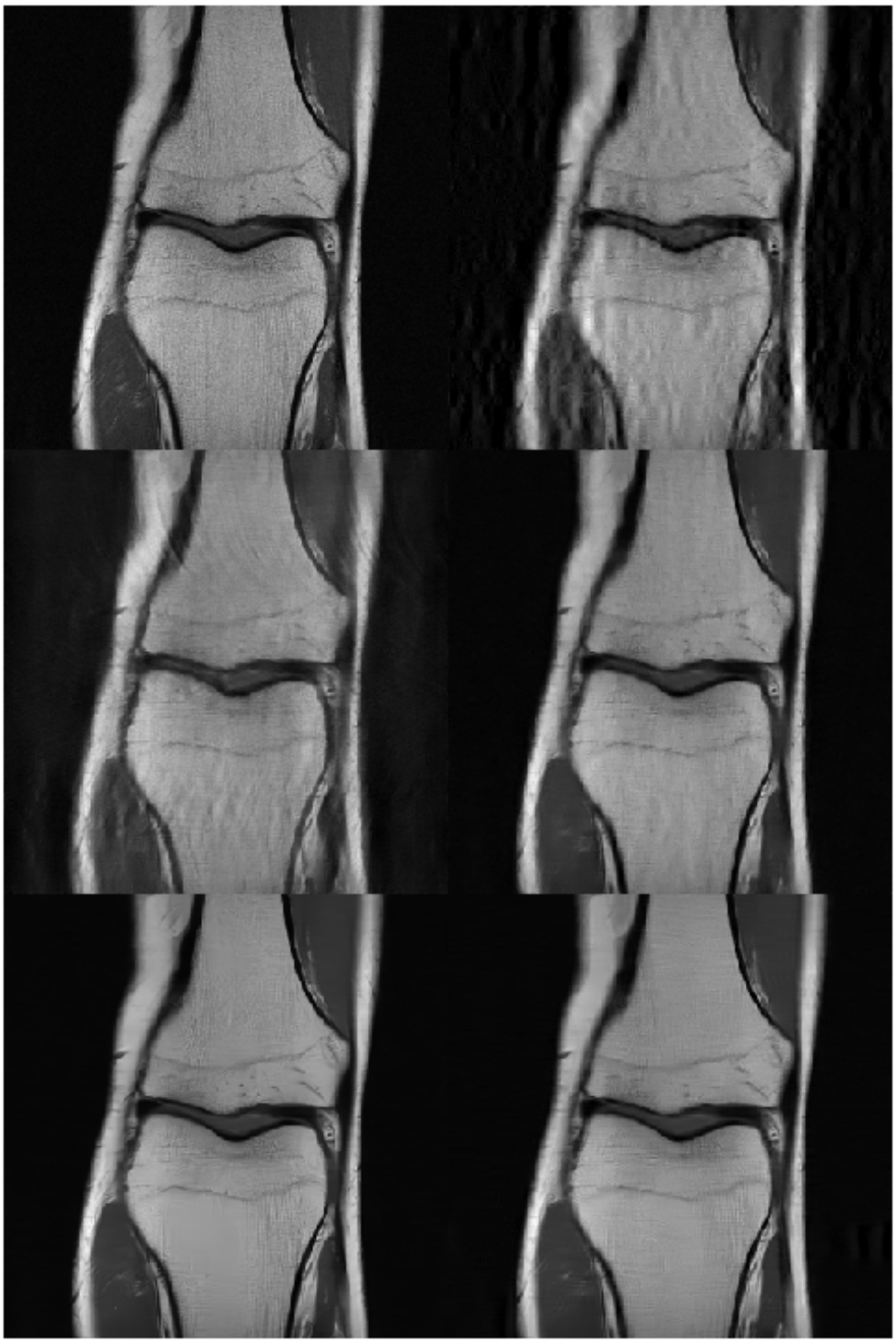

Figure 14: Ground truth (top left), M1 (top right), M2 (middle left), M3 (middle right), M4 PnP-ADMM + Denoiser (bottom left) and M4 BM3D (bottom right)

Method	SNR (dB)		SSIM		Reconstruction time (s)	
	Average	Standard dev.	Average	Standard dev.	Average	Standard dev.
M1 (ADMM)	17.482	0.939	0.714	0.046	29.387	3.478
M2 (Modified U-Net)	14.917	3.270	0.774	0.050	1.265	0.275
M3 (PnP DnCNN)	22.408	1.730	0.864	0.030	72.283	1.836
M4 (PnP + Denoiser)	23.511	1.867	0.878	0.035	183.120	10.250
M4 (BM3D)	23.139	1.682	0.875	0.030	257.447	5.801

Table 6: SNR, SSIM, and computation time results

5 Discussion and Conclusion

M1 and **M2** are considered as baselines. This is because both methods are in a single paradigm: M1 is purely based on convex optimisation whereas M2 is purely based on deep learning. Comparing to M1, M2 gives a lower average SNR with a higher SNR standard deviation. M2's average SSIM, however, is marginally higher than M1 and their SSIM standard deviation is very similar. Comparing Figure 6 with Figure 7, one can see that M2 is able to remove some line artefacts which are visible using M1. This could be caused by that SSIM considers image degradation as perceived change in structural information and human visual perception is highly adapted for extracting structural information [17]. Nevertheless, in M2 results there are still some visible artefacts left. Admittedly, M2 network's training process does take quite a long time (around 7 hours in our case, subject to hardware resources). However, once the network is trained, M2 has the shortest reconstruction time compared to all other methods. This is due to the fact that the the network is a simple feedforward network in which the reconstructed image is generated by passing the input through the network in one go. This process is non-iterative and in fact highly parallelisable hence can make use of one or multiple graphics processing units (GPUs) for further acceleration. Potential improvements of M1 include using the TV (total variation) norm instead of the L2 norm. Potential improvements of M2 include using a deeper network (i.e. more {Conv, BN, ReLU, Max pooling} layers), data augmentation (e.g. mirroring each image in the training set) and ensembling (i.e. training a number of networks independently and averaging their outputs at a test time).

M3 returned improved results over **M1** and **M2**, as measured by both SNR and SSIM. However, the performance is poorer than both alternative Plug-and-Play methods used in M4, which is likely to indicate an issue in the training process. It would be expected that a network trained on the specific dataset would be able to outperform a generic denoiser, but this is not observed here. With more training time, and altered parameters, it would be hoped that this could be improved. The reconstructed images are visually similar to the true images and do not obviously exhibit large-scale artefacts, but finer scale details such as textures are not always preserved. While, after training, **M3** is between 2 and 3 times as computationally expensive as **M1**, the results are a significant improvement.

The benefit of using networks that have been well trained for **M4** is seen in improved SSIM, SNR and lower standard deviations in Table 6. However, one can see in Figure 12 that there there are patches that are rough or fuzzy. The way that this was addressed was to use an additional denoising step with a different denoising neural network. Indeed, we can see there are no longer any such artefacts in Figure 10. There is only a small amount of fine detail that this method could not recover, and this occurred in areas where the image gradient was small. This may be possible to overcome by additional transfer learning of the general denoising network to this particular problem using parameters described in [20], using IRCNN as the initial training state.

Unfortunately, **M4** had to be run using MatConvNet. Although the networks used were a similar size to the one used in **M3**, this led to increased recovery times. By importing the weights to MATLAB and performing transfer learning, the network could then be used without MatConvNet which should reduce reconstruction time so that it becomes similar to that of **M3**.

It is not clear whether the marginal improvement using IRCNN rather than the proposed network in **M3** in PnP-ADMM can be attributed to architecture or training. Given that the two architectures are very similar, we suspect that the differences in results are due to differences in training. Indeed, the network for **M3** is much deeper than IRCNN, with 17 layers rather than 7. Therefore, we expect that with more extensive training, the network in **M3** could out-perform IRCNN.

To conclude, in this project we implement five different methods to achieve reconstruction of sub-Nyquist sampled and noisy measurements of MR images. M1 is purely based on compressed sensing and convex optimisation; M2 is purely based on deep learning; all **M3** and **M4** methods substitute the ℓ^1 proximal operator for a denoising operator in the ADMM algorithm, but lack rigorous theory to guarantee convergence. Using the BM3D algorithm as the denoising engine in the PnP-ADMM illustrates that high-quality results can be achieved using Plug-and-Play algorithms which do not rely on a deep-learning based approach. In particular, a similar image quality was achieved using BM3D and IRCNN, although with a higher on-line reconstruction time for BM3D. The observation that high-quality results can be achieved from generic denoisers illustrates the strength of the Plug-and-Play approach, as the computationally intensive training of networks can be avoided while still retaining excellent performance. While for many tasks it may well be possible to obtain improved results using end-to-end networks or by applying data-specific denoising operators to the Plug-and-Play algorithm, pretrained denoisers appear to set a high-quality baseline for any new method to overcome.

References

[1] Kazunori Akiyama, Antxon Alberdi, Walter Alef, Keiichi Asada, Rebecca Azulay, Anne-Kathrin Baczko, David Ball, Mislav Baloković, John Barrett, Dan Bintley, et al. First m87 event horizon telescope results. iv. imaging the central supermassive black hole. *The Astrophysical Journal Letters*, 875(1):L4, 2019.

[2] Stephen Boyd, Neal Parikh, and Eric Chu. *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

[3] Antoni Buades, Bartomeu Coll, and J-M Morel. A non-local algorithm for image denoising. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 60–65. IEEE, 2005.

[4] Emmanuel J Candes et al. The restricted isometry property and its implications for compressed sensing. *Comptes rendus mathematique*, 346(9-10):589–592, 2008.

[5] Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian. Image denoising by sparse 3-d transform-domain collaborative filtering. *IEEE Transactions on image processing*, 16(8):2080–2095, 2007.

[6] William A Edelstein, Mahadevappa Mahesh, and John A Carrino. MRI: time is dose—and money and versatility. *Journal of the American College of Radiology: JACR*, 7(8):650, 2010.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[8] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[9] Kyong Hwan Jin, Michael T McCann, Emmanuel Froustey, and Michael Unser. Deep convolutional neural network for inverse problems in imaging. *IEEE Transactions on Image Processing*, 26(9):4509–4522, 2017.

[10] Ymir Makinen, Lucio Azzari, and Alessandro Foi. Exact transform-domain noise variance for collaborative filtering of stationary correlated noise. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 185–189. IEEE, 2019.

[11] Ymir Makinen, Lucio Azzari, and Alessandro Foi. Collaborative filtering of correlated noise: Exact transform-domain variance for improved shrinkage and patch matching. *IEEE transactions on image processing*, 29:8339–8354, 2020.

[12] Holger Rauhut. Compressive sensing and structured random matrices. *Theoretical foundations and numerical methods for sparse recovery*, 9:1–92, 2010.

[13] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[14] Ernest K Ryu, Jialin Liu, Sicheng Wang, Xiaohan Chen, Zhangyang Wang, and Wotao Yin. Plug-and-play methods provably converge with properly trained denoisers. *arXiv preprint arXiv:1905.05406*, 2019.

[15] Andrea Vedaldi and Karel Lenc. Matconvnet: Convolutional neural networks for matlab. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 689–692, 2015.

[16] Singanallur V Venkatakrishnan, Charles A Bouman, and Brendt Wohlberg. Plug-and-play priors for model based reconstruction. In *2013 IEEE Global Conference on Signal and Information Processing*, pages 945–948. IEEE, 2013.

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

[18] Jure Zbontar, Florian Knoll, Anuroop Sriram, Tullie Murrell, Zhengnan Huang, Matthew J Muckley, Aaron Defazio, Ruben Stern, Patricia Johnson, Mary Bruno, et al. fastmri: An open dataset and benchmarks for accelerated mri. *arXiv preprint arXiv:1811.08839*, 2018.

[19] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *IEEE Transactions on Image Processing*, 26(7):3142–3155, 2017.

[20] Kai Zhang, Wangmeng Zuo, Shuhang Gu, and Lei Zhang. Learning deep cnn denoiser prior for image restoration. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3929–3938, 2017.