APPENDIX

A. Small, adversarially-selected perturbations

Here we explain how we find adversarially-selected perturbations, and give additional details on the effect of those perturbations.

A.1. Finding adversarial perturbations through optimization

Adversarial perturbation are designed to be small in the measurement domain, but to induce significant reconstruction artifacts in the image recovered by the under-sampled version of the perturbed ground-truth data. An adversarial perturbation depends on the image, the forward model, and the reconstruction method.

Finding adversarial perturbations for trained neural networks: Let $\mathbf{A}$ be the linear forward model mapping an image $\mathbf{x} \in \mathbb{R}^N$ to a measurement $\mathbf{y} \in \mathbb{R}^M$. In our context of MRI, the forward model consists of taking the Fourier transform and applying a mask. In the multi-coil setup the forward model takes the measurements for all coils, and the measurement $\mathbf{y}$ contains the measurements of all coils.

Let $\Psi \colon \mathbb{R}^M \to \mathbb{R}^N$ be a trained neural network Ψ mapping a measurement to an image. In order to find an adversarial perturbation $\mathbf{z} \in \mathbb{R}^M$ that is additive in the measurement domain for the neural network Ψ and for a given image $\mathbf{x}^*$, we apply projected gradient descent to the following optimization problem:

$$\min_{\mathbf{z}\colon \|\mathbf{z}\|_2 \leq \epsilon \|\mathbf{A}\mathbf{x}^*\|_2} -\frac{1}{2}\|\Psi(\mathbf{A}\mathbf{x}^*) - \Psi(\mathbf{A}\mathbf{x}^* + \mathbf{z})\|_2^2, \tag{2}$$

In words, this optimization problem seeks a perturbation $\mathbf{z}$ which causes maximal discrepancy between the reconstruction from a clean measurment $\mathbf{A}\mathbf{x}^*$ and a perturbed measurement $\mathbf{A}\mathbf{x}^* + \mathbf{z}$.

In order to run projected gradient descent, we need to be able to compute gradients of the loss in equation (2) with respect to the perturbation $\mathbf{z}$, thus the reconstruction operator Ψ needs to be differentiable. This is the case for the trained neural network based reconstruction methods in this paper by using standard auto-differentiate packages like PyTorch. However, this method is not directly applicable for reconstruction with the ConvDecoder and ℓ_1-minimization, as discussed more below.

Also note that the optimization problem is non-convex, thus projected gradient descent might converge to a sub-optimal perturbation, i.e., a perturbation that does not induce the maximal reconstruction error. This problem is inherent to finding adversarial perturbations for most machine learning methods.

We finally note that in order to find an adversarial perturbation, Antun et al. (2020) proposed to minimize

$$\mathcal{L}(\mathbf{z}) = -\frac{1}{2}\|\Psi(\mathbf{A}\mathbf{x}^*) - \Psi(\mathbf{A}\mathbf{x}^* + \mathbf{M}\mathbf{z})\|_2^2 + \frac{\lambda}{2}\|\mathbf{z}\|_2^2.$$

Here, the parameter λ controls the magnitude of the perturbation. We found it difficult to tune this parameter for finding a perturbation with a given norm controlled by ϵ, and therefore work with projected gradient descent, which is guaranteed to give a perturbation of norm ϵ.

Finding adversarial perturbations for un-trained neural networks and ℓ_1-minimization: In principle, we could find adversarial perturbation for any reconstruction method Ψ by solving the optimization problem in equation (2). However, for un-trained methods and for ℓ_1-minimization it is not clear how to solve this optimization problem because gradients of Ψ cannot easily be computed, because the reconstruction map is an optimization problem itself.

There are at least two approaches to overcome the difficulty of computing gradients. The first is to numerically compute a gradient. For example, the numerical gradient of the first term in the loss function w.r.t $\mathbf{z}$, assuming $g(\mathbf{z}) = -\frac{1}{2}\|\Psi(\mathbf{A}\mathbf{x}^*) - \Psi(\mathbf{A}\mathbf{x}^* + \mathbf{z})\|_2^2$, can be computed as:

$$\left[\frac{\partial g(\mathbf{z})}{\partial \mathbf{z}}\right]_i \approx \frac{g(\mathbf{z} + \mathbf{e}_i h) - g(\mathbf{z})}{h},$$

where $\mathbf{e}_i$ is the i-th unit vector and h a very small constant. However, estimating the gradient like this is computationally extremely expensive as we need to evaluate the reconstruction method Ψ for each estimate of the gradient, and the perturbation $\mathbf{z}$ is of high dimension in practice, so we have to evaluate the reconstruction method many times.

A second method, pursued by Genzel et al. (2020), is to realize that many optimization methods including ℓ_1-minimization are solved numerically with an iterative algorithm, and that the corresponding iterations can be written as a neural network. For this un-rolled network we can compute gradients with standard software packages for automatic differentiation. This is in principle even possible for the un-trained neural networks that we consider.

As a perhaps simpler alternative, we propose a two-step procedure for constructing adversarial perturbations for un-trained methods through optimization. This approach is inspired by optimizing a loss function similar to equation (2).

Step 1. We illustrate our approach with ℓ_1-minimization. For ℓ_1-minimization (recall equation (1)), given a measurement $\mathbf{y}$, we minimize the loss function $\mathcal{L}_1(\mathbf{x},\mathbf{y}) = \frac{1}{2}\|\mathbf{A}\mathbf{x}-\mathbf{y}\|_2^2 + \lambda\|\mathbf{H}\mathbf{x}\|_1$ with respect to $\mathbf{x}$, where $\mathbf{A}$ is the forward model and $\mathbf{H}$ is the sparsifying basis. The idea is to combine the optimization problems of (i) reconstructing the image and (ii) finding a perturbation into one optimization problem.

In the first step of our method, we minimize the loss function:

$$\mathcal{L}(\mathbf{z},\mathbf{x}) = \mathcal{L}_1(\mathbf{x},\mathbf{A}\mathbf{x}^* + \mathbf{z}) \; - \beta\|\mathbf{x}-\mathbf{x}^*\|_2^2 \tag{3}$$

by performing gradient descent steps with respect to the reconstructed image $\mathbf{x}$ and projected gradient descent steps with respect to the perturbation $\mathbf{z}$, where, as before, we project onto the ℓ_2-ball $\{\mathbf{z}\colon \|\mathbf{z}\|_2 \leq \epsilon\|\mathbf{A}\mathbf{x}^*\|_2\}$. Note that the first term reconstructs an image consistent with the measurement, and the second term penalizes solutions that are close to the original solution $\mathbf{x}^*$. Larger values of the hyper-parameter β give larger perturbations $\mathbf{z}$. Thus, while ℓ_1-norm minimization is reconstructing an image, the adversarial perturbation $\mathbf{z}$ is reconstructed simultaneously. By optimizing the loss in equation (3) in a few gradient iterations of Projected Gradient Descent (PGD), we obtain a candidate perturbation $\hat{\mathbf{z}}$ with ℓ_2 norm equal to ϵ.

Step 2. As the second step of our method, we run ℓ_1-norm minimization on $\mathbf{y} = \mathbf{A}\mathbf{x}^* + \mathbf{z}$ to see what effect this adversarial perturbation causes to the reconstruction process. This step is necessary because the image obtained from the first step is irrelevant, in that the optimization problem solved in the first step is meant for obtaining the perturbation (and therefore the output image of the first step is just a noisy image).

We emphasize that this algorithm is applicable to both ℓ_1-minimization, the ConvDecoder, and other, similar, optimization-based un-trained methods. Specifically, for the ConvDecoder, we simply apply the method to the loss function of the ConvDecoder $\mathcal{L}(\mathbf{C},\mathbf{y})$ defined in equation (4), in order to find adversarial perturbations for ConvDecoder:

$$\mathcal{L}(\mathbf{C},\mathbf{y}) = \frac{1}{2}\sum_{i=1}^{n_c}\|\mathbf{y}_i - \mathbf{A}G_i(\mathbf{C})\|_2^2. \tag{4}$$

Here, $G_i(\mathbf{C})$ is the reconstructed image for coil i.

A.2. Reconstruction examples for adversarial perturbations

As discussed in Section 4, both trained and un-trained methods are vulnerable to small, adversarial perturbations. In Figure 9, we provide sample reconstruction for all of the considered methods (ℓ_1-norm minimization, ConvDecoder, U-net, and the end-to-end VarNet) for perturbations with $\epsilon = \frac{\|\text{perturbation}\|_2}{\|k\text{-space}\|_2} = 0.08$.

A.3. The choice of sparsifying basis is irrelevant for small, adversarially-selected perturbations

In Section 4, we demonstrated that both trained and un-trained methods suffer from small perturbations in the input. As a sparsifying basis for sparsity based reconstruction, we chose the Wavelet-basis as it is one of the most popular ones used in MRI. However, the particular choice of basis is not critical for our results, and our findings continue to hold for other popular sparsifying bases: Figure 10 shows sample attacks for Wavelet, Fourier, and Discrete Cosine Transform (DCT) sparsifying bases, and it can be seen that for all those choices, the perturbations have a similar effect.

B. Distribution shifts

In this section we give additional details on distribution shifts, in particular we provide details on the adversarially-filtered shifts.

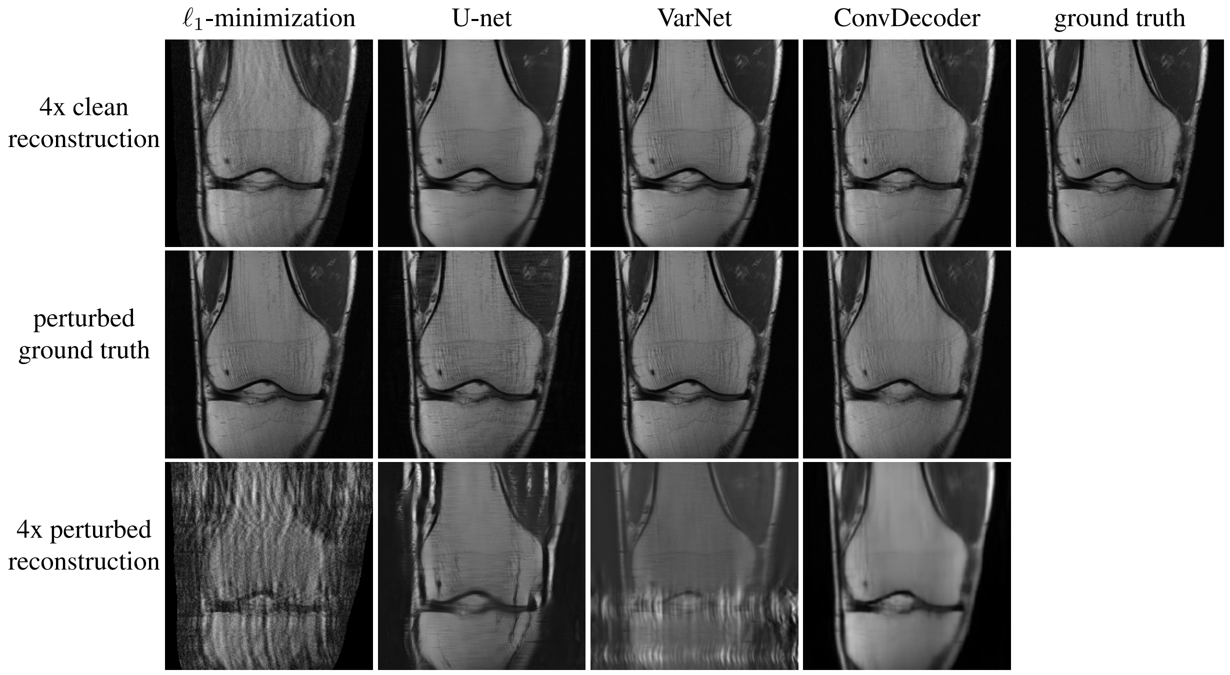

Figure 9: **End-to-end variational network (VarNet), U-net, ConvDecoder, and ℓ_1-norm minimization are all vulnerable to small adversarially-selected perturbations. First row:** The ground-truth image and reconstructions from the clean 4x under-sampled measurements for a validation image from the multi-coil fastMRI dataset. **Second row:** Perturbed ground-truth images after finding one perturbation specifically for each method ($\epsilon = 0.08$ which is the maximal perturbation considered in this study). **Third row:** Reconstructions from the perturbed 4x under-sampled measurements.

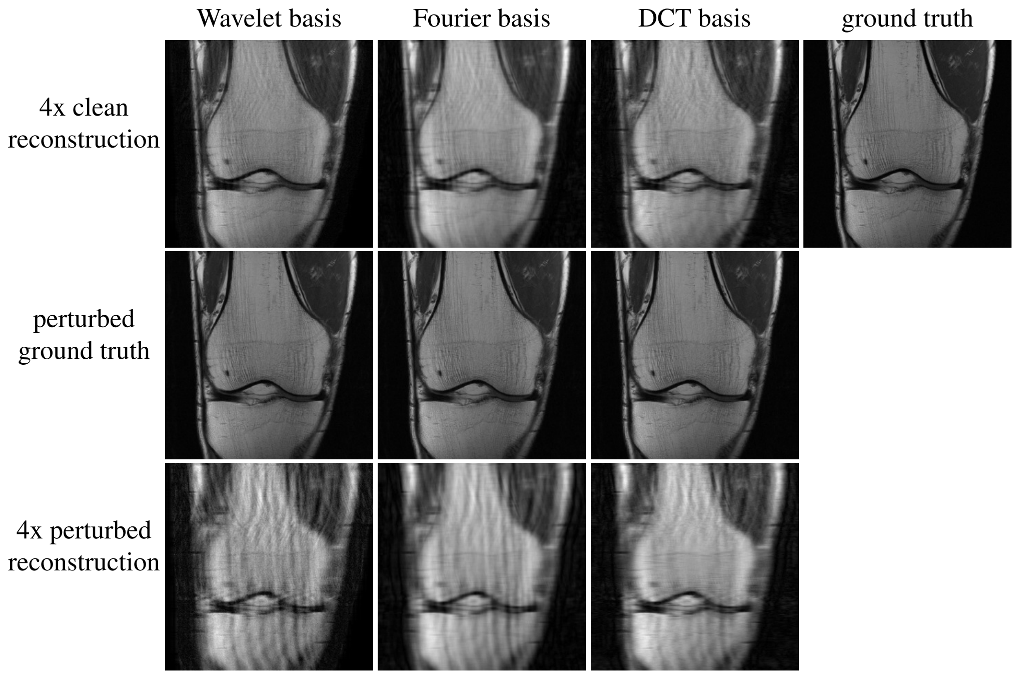

Figure 10: Perturbations have a similar effect for different common choices of sparsifying bases. The experiment shows that an adversarial perturbation with $\epsilon = 0.08$ specific for each basis can be found.

B.1. Reconstruction examples for the dataset shift

As shown in Section 5.1, all of the models that have been trained or tuned on the fastMRI domain perform worse when evaluated on the Stanford set. In Figure 11, we provide reconstruction examples for such a distribution shift, showing that a distribution shift induces visible performance degradation.

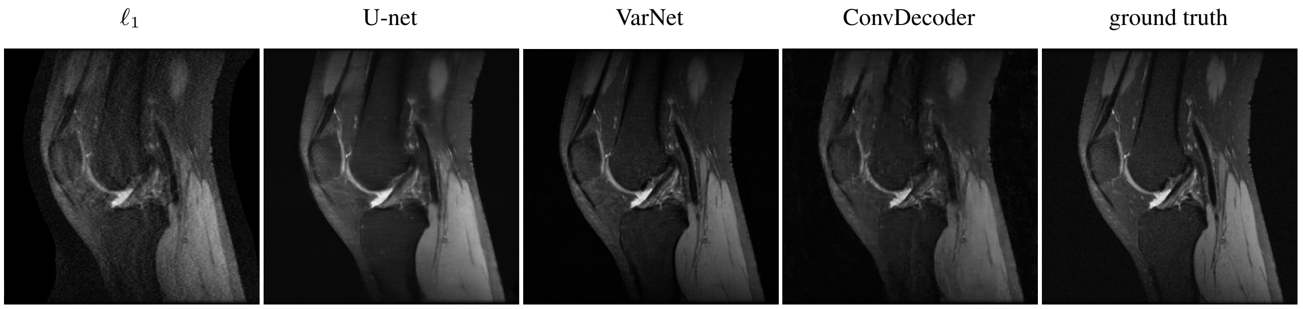

Figure 11: **fastMRI reconstruction methods do not generalize well to the Stanford set.** All of the considered reconstruction methods linearly lose performance when being evaluated on the Stanford set. Reconstructions (for 4x acceleration) show that each method either comes with reconstruction artifacts or fails to recover all the details.

B.2. Reconstruction examples for anatomy shift

We saw in Section 5.2 that a dataset shift (from brain to knee images and vice versa) causes a performance loss for all reconstruction methods. In Figure 5 in the main body, we provide sample reconstructions when shifting the domain from knee to brain images which show that trained neural networks tend to generated artifacts with such a shift. In Figure 12, we provide additional sample reconstruction when shifting the domain from brain to knee images. As shown, trained neural networks generate artifacts in this case as well.

Figure 4 in Section 5.2 shows that reconstruction quality degrades with an anatomy shift. To demonstrate this, we plot in panel 1 of Figure 4 the SSIM reconstruction scores of all methods when trained (or tuned) on knee images and tested on brain images, compared to the case that they are trained and tested on brain images.

Another way to visualize the exact same data is to plot the SSIM reconstruction scores of a model trained on knee images evaluated on brains over the reconstruction scores of the same model evaluated on knees. This alternative evaluation can be found in Figure 13. The Figure shows, in addition to our previous findings, that brain images are naturally easier to reconstruct than knee images as all models achieve higher scores for them.

B.3. Details on the adversarially-filtered shift

In Section 5.3, we showed how a group of images can be naturally difficult to reconstruct for all image reconstruction methods. We further hypothesized that these samples are challenging due to less concentration of energy in the center region of their k-space, which corresponds to low-frequency components of the image. In this section, we confirm this hypothesis

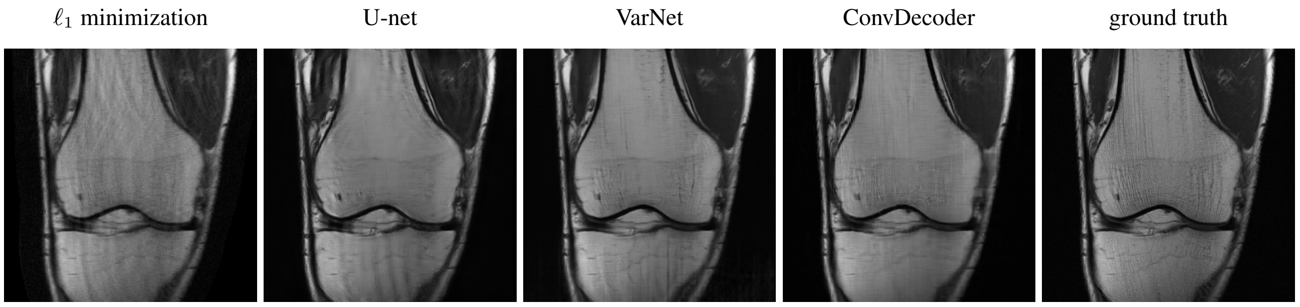

Figure 12: **Anatomy shift from brain to knee.** End-to-end variational network (VarNet) and U-net generate vertical reconstruction artifacts in the lower side of the image when being trained on brain and tested on knee. Reconstructions are acquired from 4x under-sampled measurements.

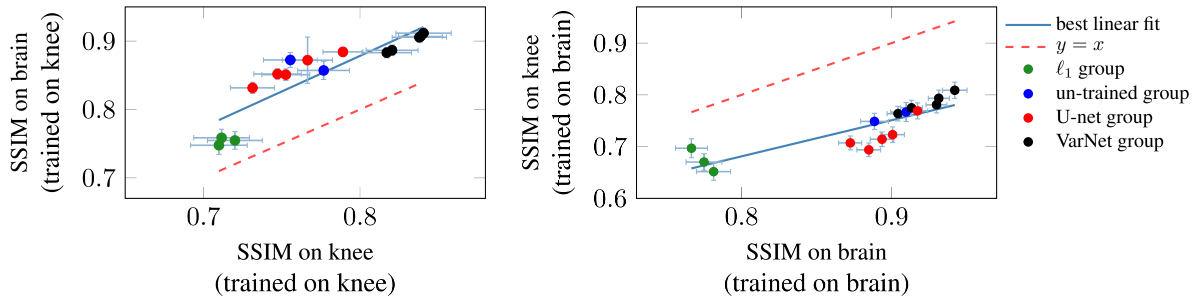

Figure 13: **Anatomy shift evaluation.** Another visualization of the results in Fig. 4: in each plot, we consider the same model trained on the same dataset, but evaluate on different test sets (knee and brain). Validation results for a group of 100 brain images and a group of 100 knee images from the fastMRI validation set show that trained an un-trained models achieve higher scores for brain images.

and then depict a number of samples from the fastMRI-A dataset.

In order to demonstrate that challenging samples are difficult to reconstruct because of the energy distribution in their frequency-domain representation, we performed the following experiment. We first define the low-frequency proportion of an image as

$$\text{low-frequency proportion} = \frac{\text{energy of the center frequencies}}{\text{energy of the whole } k\text{-space}}.$$

An image with a high low-frequency proportion has most energy in the low-frequency component and is therefore relatively smooth, has less detail, and is relatively easy to reconstruct. Figure 14, left panel, shows that the challenging images indeed have a low low-frequency proportion, i.e., most challenging images have much of their energy concentrated in high frequency components, which are sub-sampled.

Second, we calculate the energy in the frequency domain as a function of the frequency along a vertical line in the k-space. The figure shows again that challenging images have much of it's energy concentrated on high frequencies.

From this we can conclude that difficult-to-reconstruct samples are difficult to reconstruction because relative to other samples, (i) they contain less energy concentrated on their low (center) frequencies, and (ii) they contain more high-frequency information which increases the probability of not being recorded during the scanning process.

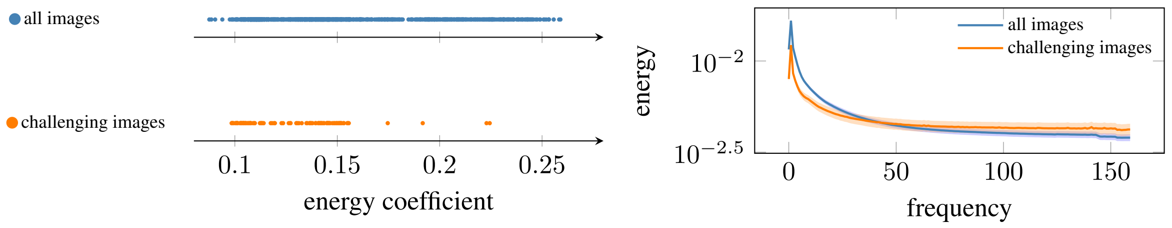

Figure 14: Difficult-to-reconstruct images contain more energy on the high frequencies as the average fastMRI image. **Left:** Scatter plot of the fraction of signal energy concentrated on the low, fully-sampled, center frequencies over the energy of the whole k-space. Difficult-to-reconstruct images have smaller such coefficient, indicating that more energy is on the high frequencies, which are only sub-sampled. **Right:** Amount of energy as a function of the frequency along a vertical line in the k-space (in order to be consistent with the sampling trajectory). The cross between the two lines at frequency = 45 shows the difficult-to-reconstruct images have more energy on the high frequencies, relative to the average fastMRI image.

Finally, to give a visual impression of the fastMRI-A dataset, we provide a number of samples along with their 4x-accelerated reconstructions by the considered methods (ℓ_1-norm minimization, ConvDecoder, U-net, and the end-to-end variational network (VarNet)), in Figure 15. The figure shows that fastMRI-A contains samples that are naturally difficult to reconstruct. However, there are a few samples whose ground-truth image is corrupted with measurement noise. Consequently, all reconstruction methods yield low scores for such samples.

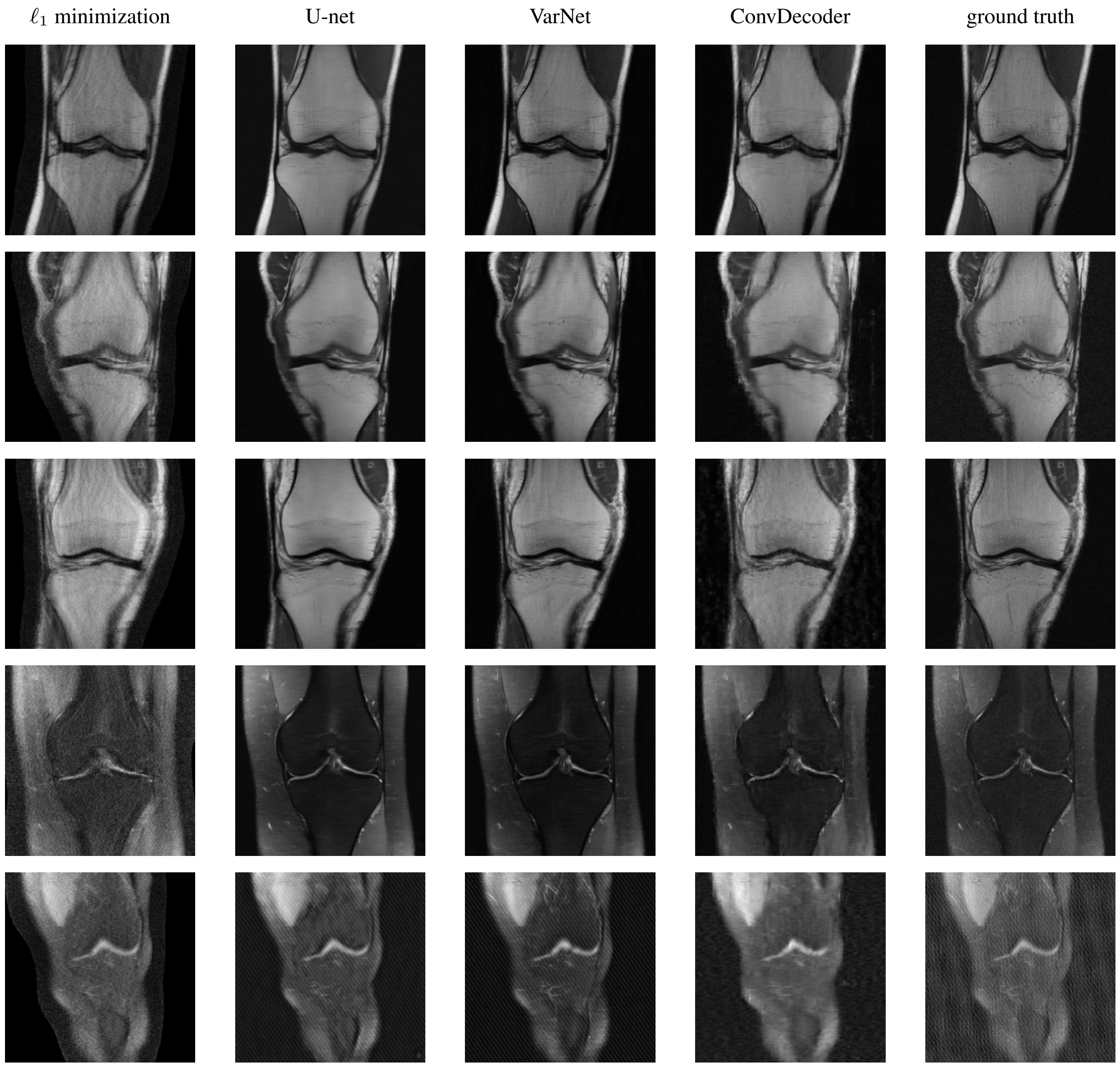

Figure 15: Samples from the fastMRI-A dataset and their corresponding reconstructions by ℓ_1-norm minimization, ConvDecoder, U-net, and the end-to-end variational network (VarNet).

C. Details on recovering small features in an image

In Section 6, we showed that (i) there is a location dependence associated with each reconstruction method when recovering small features, and (ii) reconstruction quality is correlated with small feature recovery error when evaluating different reconstruction methods on images which contain real-world features. In this section, we provide further details on recovering fine details based on the framework we proposed for determining the location dependence.

C.1. Recovering small features via an un-trained network

It was shown in Section 6 that different reconstruction methods are sensitive to different regions in an image when recovering tiny details. According to Figure 7, the end-to-end variational network, U-net, and ℓ_1-norm minimization all exhibit such location dependency. However, the un-trained network ConvDecoder shows a random dependence across the image when being optimized by the Adam optimizer (Kingma & Ba, 2015). In this section, we explain such behavior and provide empirical conditions under which ConvDecoder also shows dependency to different regions when recovering a small feature.

The choice of optimizer is critical for the performance of an un-trained neural network, as the interaction of optimizer with the network determines the prior (Heckel & Soltanolkotabi, 2020b;a). For our setup, the Adam optimizer yields a significantly better reconstruction compared to Gradient Descent (GD), because Adam chooses learning rates associated with each layer in a way that imposes an effective prior, see Zalbagi Darestani & Heckel (2020) for concrete experiments.

Furthermore, when optimizing the network parameters over n_{iter} number of iterations, we typically take the best-performing set of parameters. The best-performing set is chosen based on the loss function which measures the closeness of network output and the under-sampled data.

These components of the method result in a potential randomness in recovering small feature at different locations in an image. To elaborate, the best-performing network at iteration i—according to the quality of the whole reconstructed image—may not have yet reconstructed the small feature. This can be mitigated by controlling the convergence smoothness, as we discuss next.

There are numerous ways to achieve a smoother convergence for optimizing ConvDecoder. For example, one way is to choose a larger β_1 (we chose 0.98 instead of the default 0.9 value) for Adam. Alternatively, one can use GD instead of Adam for a smoother convergence. In this case, as shown in Figure 16, smoother convergence brings location dependency at the cost of losing small-feature-recovery performance (since the original Adam, on average performs significantly better than GD or Adam with larger β_1 for recovering small features).

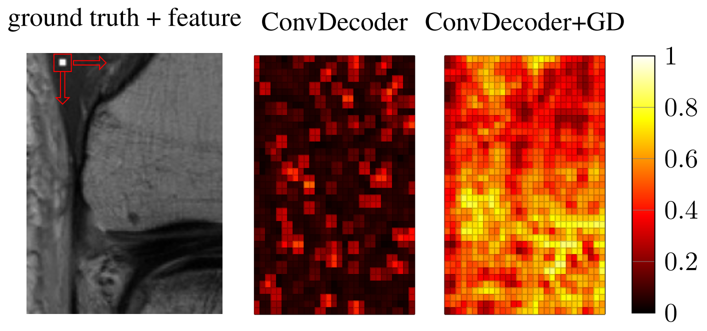

Figure 16: Smoother convergence brings location dependency at the cost losing average small-feature recovery performance. For Gradient Descent (GD), we use a linearly-increasing stepsize schedule.

C.2. Quantitative analysis based on artificial features

Based on our framework described in Section 6, we performed the following experiment to gain a quantitative understanding of artificial feature recovery, similar to the results in Figure 8 for real-world features.

For all considered reconstruction methods, we put the fixed window of size k on different locations and averaged the Mean-Squared Error (MSE) over 200 randomly-chosen images from the fastMRI knee validation set and also varied the window size in $\{2, 3, 4, 5\}$. Then, we randomly picked one of the images and put the fixed window on a random location and measured how much of it is lost during the reconstruction process. We performed this experiment 100 times for each reconstruction method to obtain an average information loss value as a metric for how reliable each method is in recovering fine details in an image.

Figure 17 shows the results for this experiment. By comparing this result to the one shown in Figure 8 for real-world features, we see somewhat contradicting results: For example, VarNet performs much worse than ConvDecoder in recovering such details. This could be due to one of the following: Either artificial features are ineffective for evaluating robustness w.r.t. recovering fine details and real-world features are essential, or a more crafted procedure to synthesize these features is needed to generate real-world-like features.

C.3. Reconstruction examples for real-world small feature recovery

As shown in Section 6, overall reconstruction quality correlates with accurate recovery of extremely small features in an image. In Figure 18, we provide reconstruction examples for three annotated images from the fastMRI validation set[1].

[1]Annotated slices are provided by https://github.com/fcaliva/fastMRI_BB_abnormalities_annotation

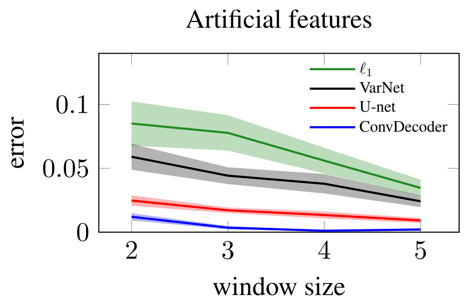

Figure 17: **Natural features are essential for evaluating robustness in recovering small features.** Normalized information recovery error based on feature size (note that window size = 2 represents a 2x2 feature). For all considered window sizes, the un-trained netowrk ConvDecoder performs best. The numbers are averaged over 100 runs on the 4x accelerated knee multi-coil measurements from the fastMRI dataset.

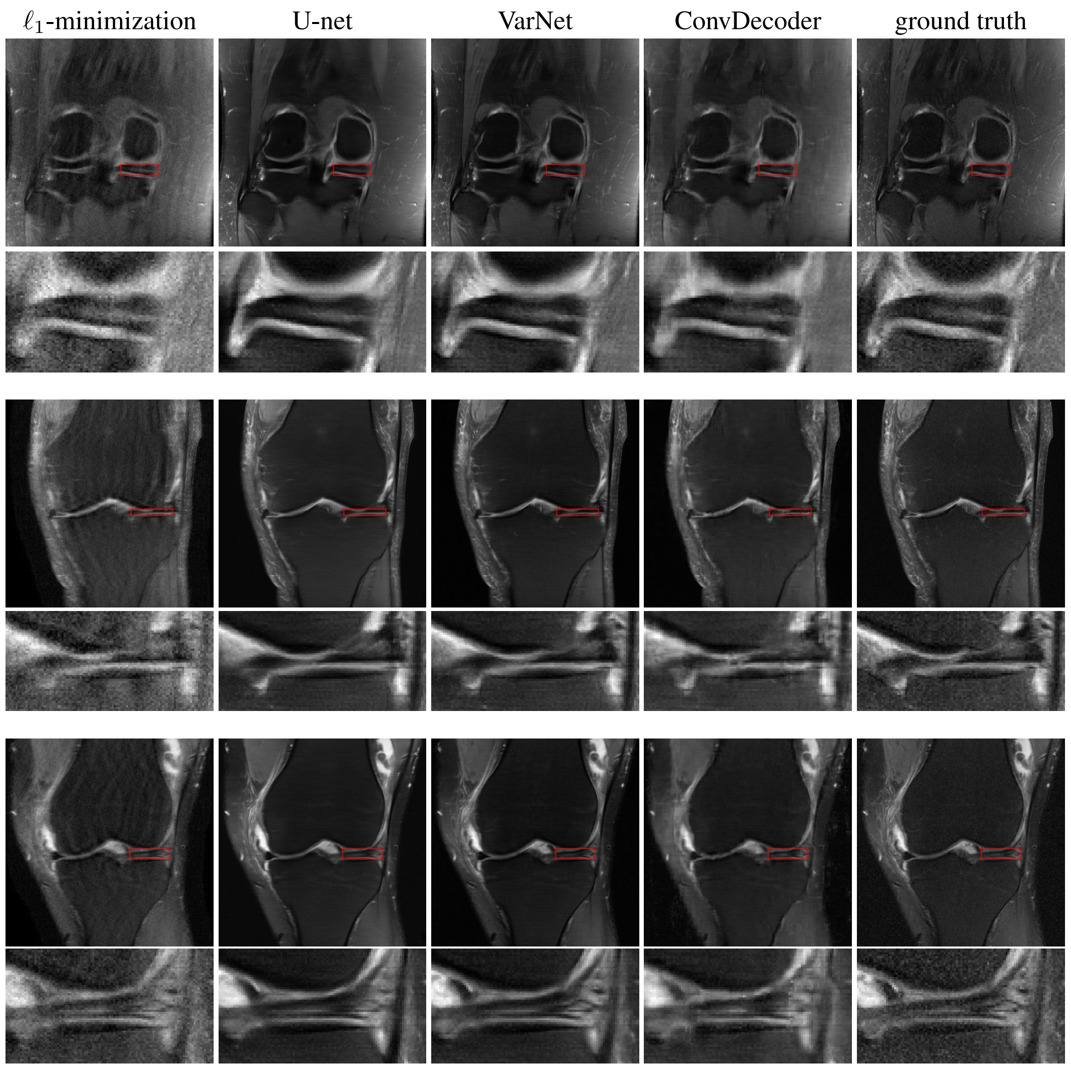

Figure 18: Reconstruction examples for real-world small feature recovery.

D. Parameter setup

Throughout the paper, we have used four reconstruction methods: (i) the end-to-end variational network (VarNet), (ii) the U-net, (iii) the ConvDecoder, and (iv) ℓ_1-norm minimization. The parameter setup we chose for VarNet and U-net are shown in Table 1 and Table 2, respectively, and are as selected by the organizers of the fastMRI challenge[2] (Zbontar et al., 2018). The parameters of ConvDecoder (8 layers and 256 channels) and Deep Decoder (10 layers and 368 channels) are chosen according to the results discussed by Zalbagi Darestani & Heckel (2020).

For distribution shifts (Section 5.2), in order to be able to study a larger variety of the methods, we studied differently sized networks. Specifically, we considered VarNet group (by altering the number of cascades and channels), U-net group (by changing the number of layers and channels), un-trained group (by considering Deep Decoder in addition to ConvDecoder), and ℓ_1 group (by considering Wavelet, Fourier, and DCT bases).

Method	#channels (or width)	convolutional kernel size	#pools	#sens-pools	#sens-channels	#cascades
VarNet-ref	18	3	4	4	8	12
VarNet2	9	3	4	4	4	12
VarNet3	9	3	4	4	4	6
VarNet4	4	3	4	4	2	12
VarNet5	4	3	4	4	2	6

Table 1: Model parameters for VarNet and its variants.

Method	#layers	#channels (or width)	convolutional kernel size	#pools
U-net-ref	8	32	3	4
U-net2	8	16	3	4
U-net3	4	16	3	4
U-net4	8	8	3	4
U-net5	4	8	3	4

Table 2: Model parameters for U-net and its variants. #channels denotes the width factor of each U-net (i.e., the number of channels for the first layer).

[2] https://github.com/facebookresearch/fastMRI/tree/master/models