

Image Registration

Author: Anton Zhitomirsky

Contents

1 Inverse Problems in Imaging	3
1.1 Classical Approach	3
1.1.1 Problem	3
1.1.2 Solution	3
1.2 General Approach	4
1.2.1 Why normalize gradients instead of image intensities?	4
1.3 Regularisation in inverse problems	5
1.4 Forward model/inverse model	5
2 Solving Inverse Problems with Deep Learning	5
2.1 Model Agnostic Approach	5
2.1.1 Solving the inverse problems	5
2.2 Decoupled Approach	6
2.2.1 Deep Proximal Gradient	6
2.2.2 General Adversarial Network	6
2.3 Unrolled optimisation	7
2.3.1 Gradient descent networks	7
3 Deep Learning for Image Super-Resolution	7
3.1 Problem formulation	7
3.2 Example Frameworks	8
3.3 How to implement upsampling?	8
3.4 Loss functions for super-resolution	9
3.5 GANs for Loss	9
3.6 Deep Image Prior — Parameterizing images via generative networks	10
3.6.1 Application to inpainting	10
4 Deep Learning for Image Reconstruction	11
4.1 Going from Sinogram to CT image	11

4.2 Going from K-space to MR image	12
5 AI-enabled Super resolution Images	14

1 Inverse Problems in Imaging

Inverse problems in imaging

- Observe:
$$y = Ax + n$$

- Goal:

Recover x from y

<ul style="list-style-type: none"> • Inpainting • Deblurring • Denoising • Super-resolution • Image Reconstruction (Medical imaging)

y







x







Inpainting
Deblurring
Super-resolution

The matrix A operates on some input x which generates an output with noise n . We want to try to reconstruct the original input out of the output.

For example, **Inpainting**, matrix A would act as a masking operator.

1.1 Classical Approach

1.1.1 Problem

Classical Approach

- Example: deblurring
- Least squares problem:
$$\arg \min_x \mathcal{D}(Ax, y)$$

$$\mathcal{D}(Ax, y) = \frac{1}{2} \|Ax - y\|_2^2$$

- Least squares solution:
$$\hat{x} = (A^T A)^{-1} A^T y$$









Wildly different solutions

- In de-blurring, classically we can formulate this as a least squares problem, equation on the left. A describes the forward map (here a linear operation). In other words, after applying a blurring operation to x we get y .
- The solution to this problem includes matrix manipulation.
- You can solve the problem without knowing what A is, but here we do.

The problem here, is that if the data is slightly different, then you apply the same de-blurring approach, by solving the least squares solution at the bottom, you can potentially get wildly different solutions with a slightly different initial condition. This is called a **ill-posed problem** (where a slight perturbation in the data can skew results greatly).

1.1.2 Solution

Classical Approach

- Example: deblurring
- Least squares problem:
$$\arg \min_x \mathcal{D}(Ax, y) + \lambda \mathcal{R}(x)$$

$$\mathcal{D}(Ax, y) = \frac{1}{2} \|Ax - y\|_2^2 \quad \text{and} \quad \mathcal{R}(x) = \frac{1}{2} \|x\|_2^2$$

- Least squares solution with Tikhonov regularisation:
$$\hat{x} = (A^T A + \lambda I)^{-1} A^T y$$

Better conditioned (and noise suppression)

Even with classical methods you can solve this by adding a regularisation term.

- In addition to minimising the distance between the observed data and the forward operation on the original data, you add a second term
- The second term in this example is a regularisation term, which is added to the solution, which modifies the behaviour of the optimisation problem.

- This gives a different solution (we effectively add an identity matrix scaled by lambda before we invert $A^T A$). The λ defines the strength of the regularisation; if you increase λ you get smoother

solutions because you pay less and less attention to the actual distance measure which you have in the first term.

- This inversion means that $A^\top A$ becomes better conditioned, which means it's numerically more stable and you suppress noise.

1.2 General Approach

General approach for images

$$\arg \min_x \mathcal{D}(Ax, y) + \lambda \mathcal{R}(x)$$

$\mathcal{D}(Ax, y)$ Data consistency term
 $\mathcal{R}(x)$ Regularisation term (encoding prior knowledge on x)
 λ Regularisation parameter

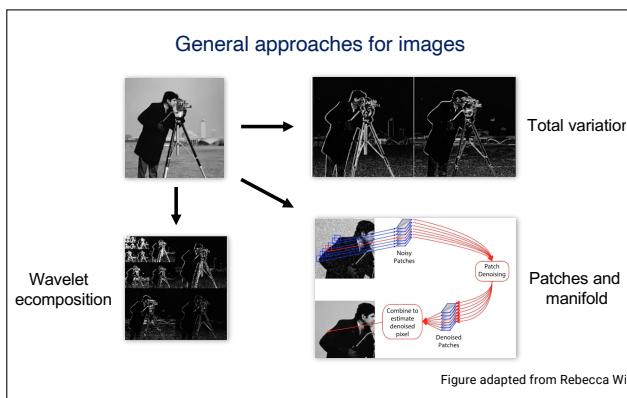
Common regularisers:

$\mathcal{R}(x) = \ \nabla x\ _2^2$	Tikhonov regularisation
$\mathcal{R}(x) = \ \nabla x\ _{2,1} = \sum_{i=0}^n \sqrt{\sum_d (\nabla x)_i^{(d)}^2}$	Total variation regularisation
$\mathcal{R}(x) = \ \Psi x\ _1$	Wavelet regularisation

- This regularisation approach encodes the prior knowledge we have on our desired solution x .
- If we can encode this prior knowledge then we have a good regularisation strategy.
- The common regularisers shown to the side don't operate on image intensities, but rather on the image gradient.
- For example, the Tikhonov regularisation uses L-2 norm to regularise the gradients of the image intensities

- The second term, minimises the L-1 norm of the gradient of image intensities.
- Another common solution is to not regularise the image intensities of the gradient of image gradients, but first apply a sparsifying transformation, e.g. wavelet and then minimise that.

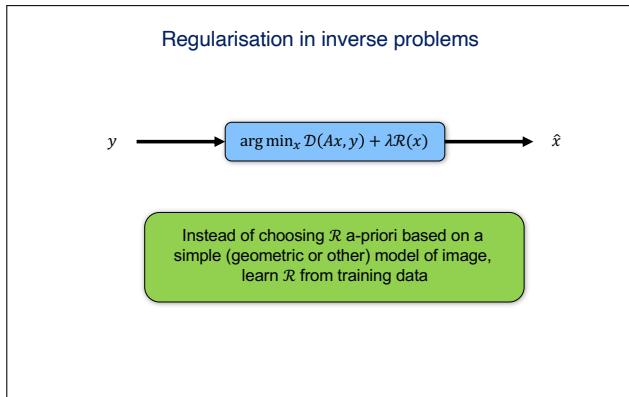
1.2.1 Why normalize gradients instead of image intensities?



- The problem with looking only at the gradients, is that the results are very sparse, which is a common thing in every-day images.
- If you apply L-1 norm on the wavelength decomposition, then you're effectively minimising the number of non-zero wavelength coefficients. This technique uses this wave decomposition in order to maximise sparsity or minimise the L-1.

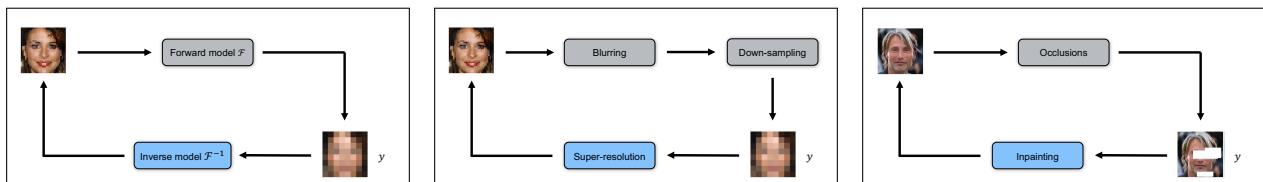
- The third approach is **Dictionary Learning** where you extract small patches from the image and build up a patch library. Then you encode each patch by a sparse combination of (e.g.) wavelet coefficients, denoising and combine these to reconstruct the original image.

1.3 Regularisation in inverse problems



- we have an optimisation function made of two terms, the data consistency term and the regularisation.
- Can we learn a regulariser from our data rather than using these very ad hoc measures.

1.4 Forward model/inverse model



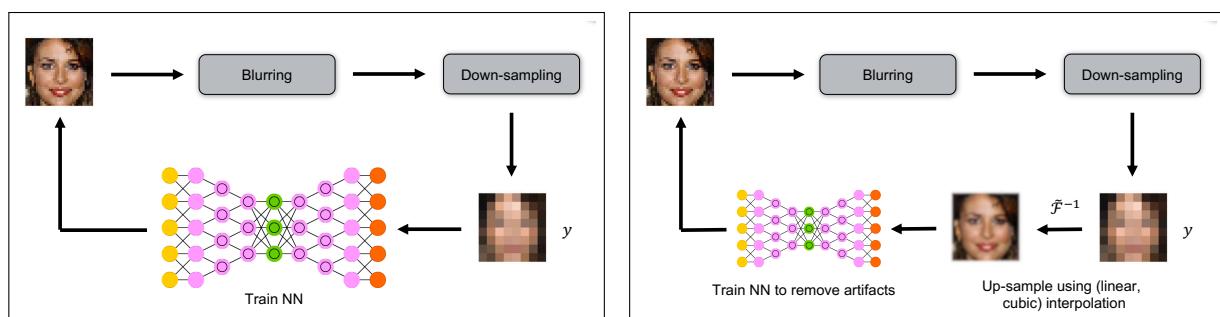
2 Solving Inverse Problems with Deep Learning

There are three classes of methods

1. Model agnostic (ignores forward model)
2. Decoupled (First learn, then reconstruct)
3. Unrolled optimisation

2.1 Model Agnostic Approach

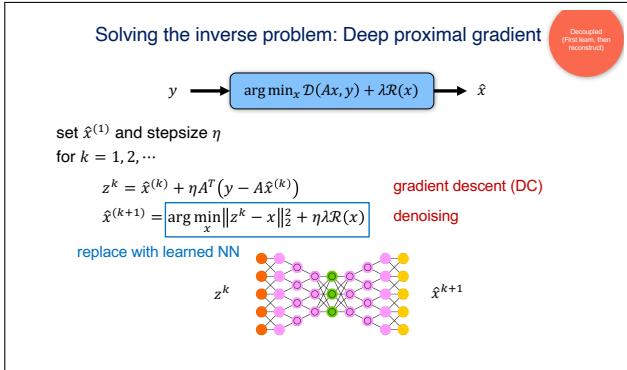
2.1.1 Solving the inverse problems



Assume that the paired data (x, y) which are easy to generate. Then take a neural network to solve the inverse problem. The issue here, is that you don't use any information about what happened in the forward operation. The model can be made more intelligent by being partly model agnostic; we can upsample using a classical algorithm, like linear interpolation, which aren't perfect but produce something useful. Then, after this approximation, you can train a network to remove the difference between the upsampled version (which may have artifacts) and the original.

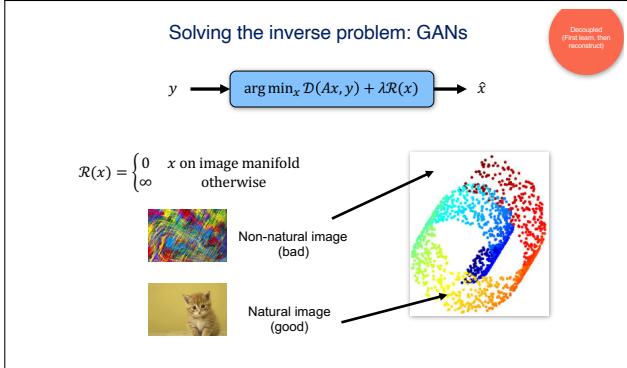
2.2 Decoupled Approach

2.2.1 Deep Proximal Gradient

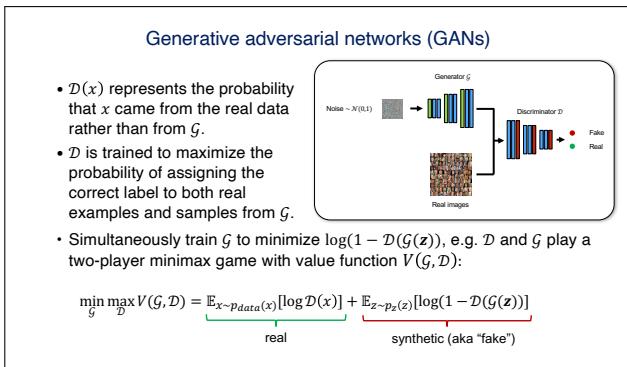


- We're still trying to solve an optimization problem, this time with an iterative optimization.
- Compute one step that minimises D and one step that minimises the image regularisation term. We estimate using gradient descent.
- The second equation is replaced with a neural network, which takes the estimate from the gradient descent step and produces a new estimate for the data.

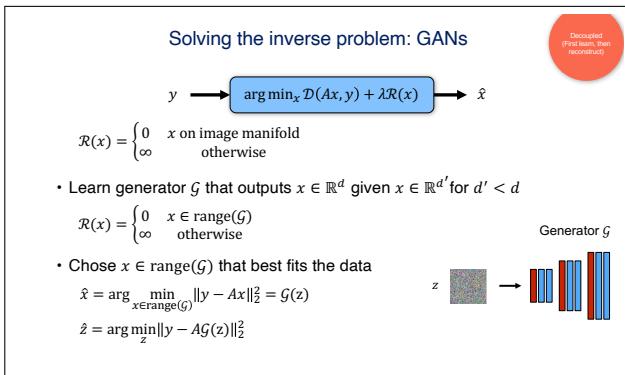
2.2.2 General Adversarial Network



- We want to learn a regularisation function which gives a value of 0 if the image looks realistic and otherwise an infinitely large value.
- If you're on the image manifold then you generate an image which looks real.



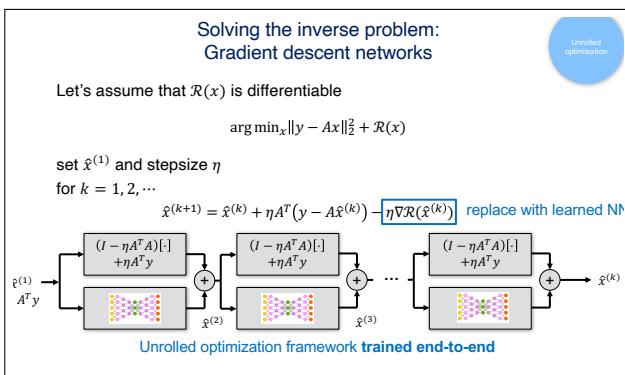
- A generator takes a vector of noise and generates images from this noise.
- The output is fed into a discriminator along with real images which tries to understand if an image is real or fake.
- Both are trained simultaneously



Here, we assume that we know A .

2.3 Unrolled optimisation

2.3.1 Gradient descent networks

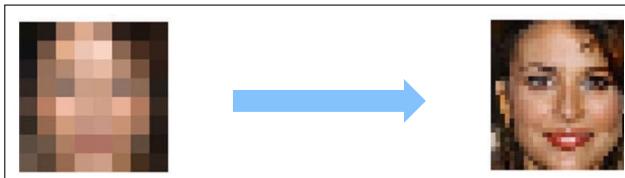


- We can now start plugging in the discriminator or generator into the inverse problem.
- We can choose only solutions of x that can be generated from the generator because the generator can only generate realistic looking images. In the optimisation, we can use the generator to generate solutions that look real.
- Note here, we have restricted the space to the image manifold shown above using the generator.

- We can replace the second step with a neural network.
- The diagram: shows an iterative optimisation step, where you start with an initial estimate, you add the gradient with respect to data consistency and the gradient with respect to regularisation and make a step in the direction of the combined gradient.
- the bottom term was replaced with a nerual network. Originally, it was the $\eta \nabla \mathcal{R}[\cdot]$
- Since we have pairs of x, y we can simulate the transformation and train the network with the transformed output.

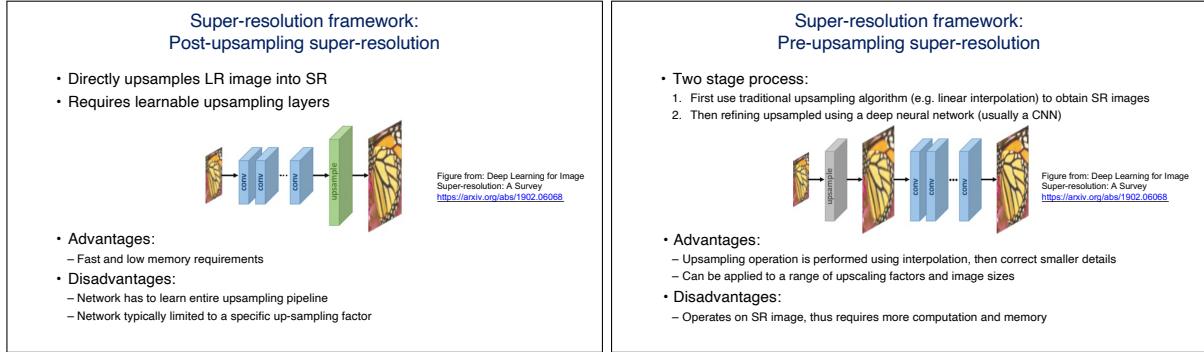
3 Deep Learning for Image Super-Resolution

3.1 Problem formulation

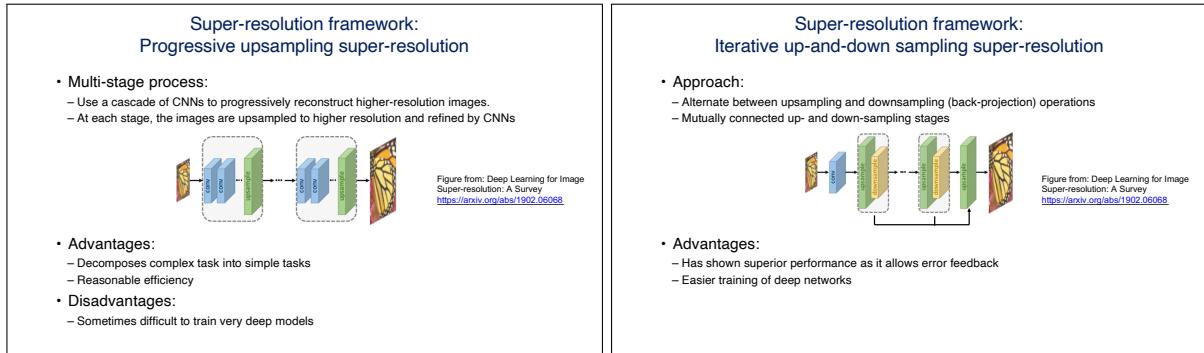


Upsample low-resolution (LR) images to high-resolution (HR or SR) image. The forward model (going from high-to-low-resolution) is straightforward and involves some image degradation followed by downsampling. The Inverse model is for example an interpolation-based model.

3.2 Example Frameworks

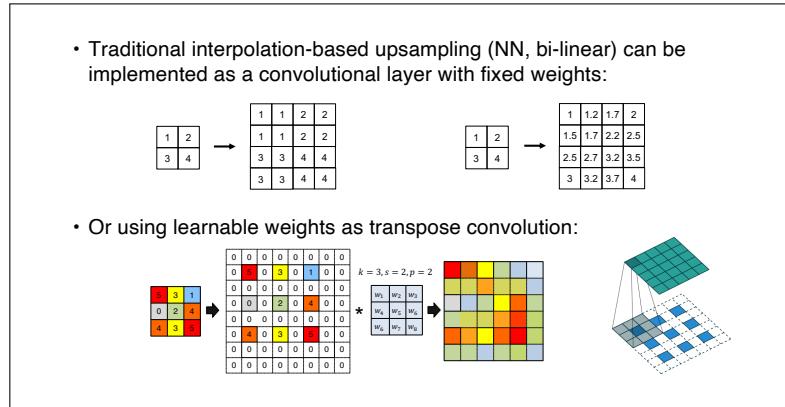


(f) We can have a network with a couple of convolutions and some upsampling (g) Can upsample data to the desired resolution, then have layers to remove artifacts and add details.



(h) A cascade by upsample by increasing factors (by a bit) (i) upsample then downsample. Compute the error between the two, and use it in future iterations to improve performance.

3.3 How to implement upsampling?



3.4 Loss functions for super-resolution

Loss functions for super-resolution

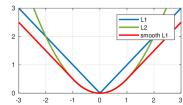
- Pixel-wise loss function (either L1 or L2)

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N |x_i - \hat{x}_i|^p$$

- Alternative: Huber loss function

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N \delta(x_i - \hat{x}_i)$$

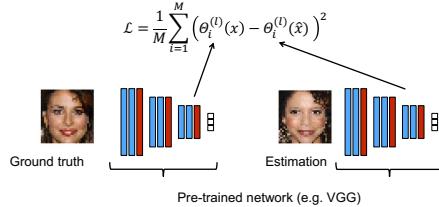
$$\delta(a) = \begin{cases} 0.5a^2 & \text{for } |a| \leq 1 \\ |a| - 0.5 & \text{otherwise} \end{cases}$$



- either take the square or absolute difference between intensities depending on the norm chosen.
- Or a huber loss function (a mix between L1 and L2)
- L1 is not differentiable which causes numerical problems, so people use something that behaves like L1 but is easier to work with. This is the Huber Loss function.

Loss functions for super-resolution

- Perceptual loss: Computes loss on the output θ of an intermediate layer l of a pre-trained network:



- Don't compare L1 and L2 losses between pixel values. Instead, compute activations in a pre-trained network, like a VGG or ResNet, then compute it for the original data and for the upsampled data.
- Then compute the loss not in terms of pixel intensities but in terms of activations in the network.

- The network has been pre-trained on a task to classify objects for instance, then the activations describe the image in more perceptual terms and that might be a better way of computing values.

Loss functions for super-resolution

- Total variation

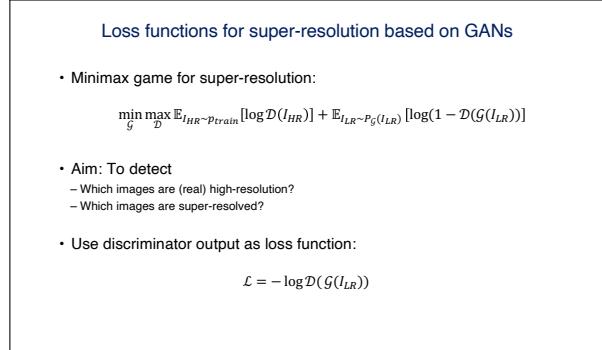
$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N \sqrt{\sum_d (\nabla x_i^{(d)})^2}$$

- Assumption: Absolute value of gradient of the image is low, e.g. image is piecewise constant
- Implementation for 2D image (using forward differences to approximate the gradient):

$$\mathcal{L} = \frac{1}{n_1 n_2} \sum_{i=1}^{n_1-1} \sum_{j=1}^{n_2-1} \sqrt{|x_{i-1,j} - x_{i,j}|^2 + |x_{i,j-1} - x_{i,j}|^2}$$

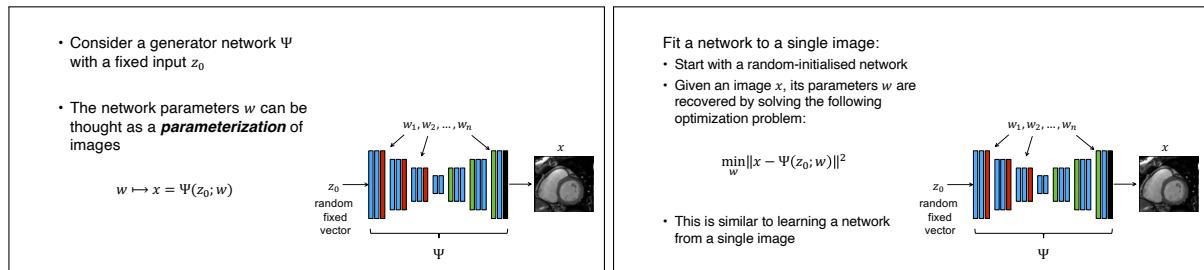
3.5 GANs for Loss

Recall, that the generator tries to generate realistic looking images, and the discriminator tries to identify which images are real and which ones are fake. You can use generative adversarial networks as a loss function.

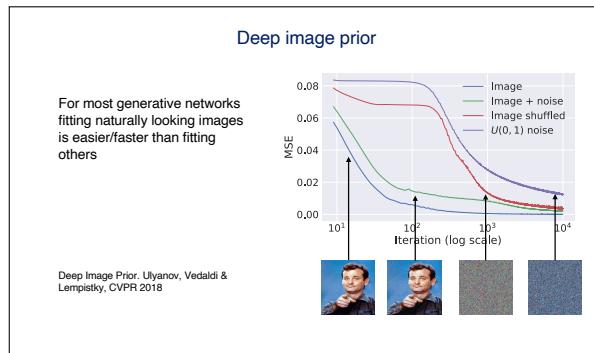


- Generate with a neural network a realistic super-resolved Image
- Use the discriminator to identify how realistic the image is looking.
- If the discriminator tells me the image is realistic, we have a low cost function, otherwise, we get a high value.

3.6 Deep Image Prior — Parameterizing images via generative networks

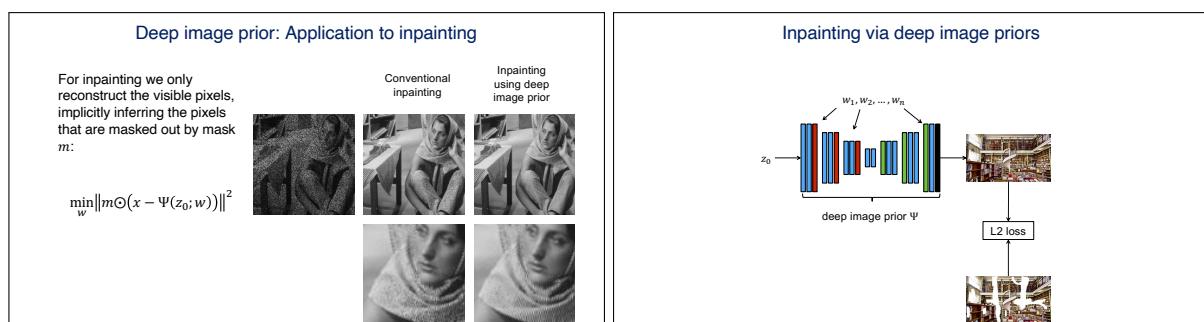


- (j) Deep image Prior: introduce the idea to represent an image not by its pixel values, but by the weights of a vector. We take a random fixed vector z_0 as input, and optimise the parameters to obtain an image x as output.
- (k) If you learn the weights in this manner, this works well for natural images.



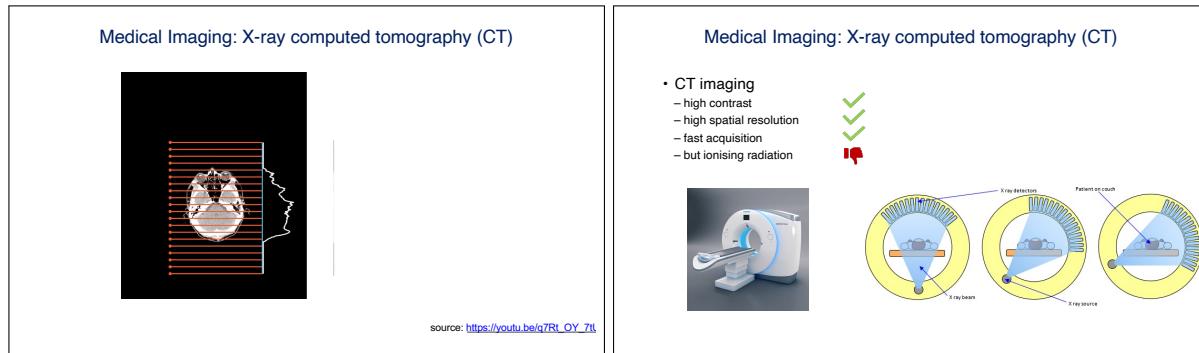
- (l) Here we see that generating noise is harder, but real images converge quicker. We can use this to solve image-in-painting. We compare the observed image with the parameterisation through the neural network.

3.6.1 Application to inpainting



Then, the pixels which are masked out by m and optimise the function over all network weights and learn how to do in-painting. We can then reconstruct the iamge by fitting a noise vector into the image, and reconstruct the image. We achive the following by optimising weights and minimising L2 loss between image generated and the observed image minus the occlusions.

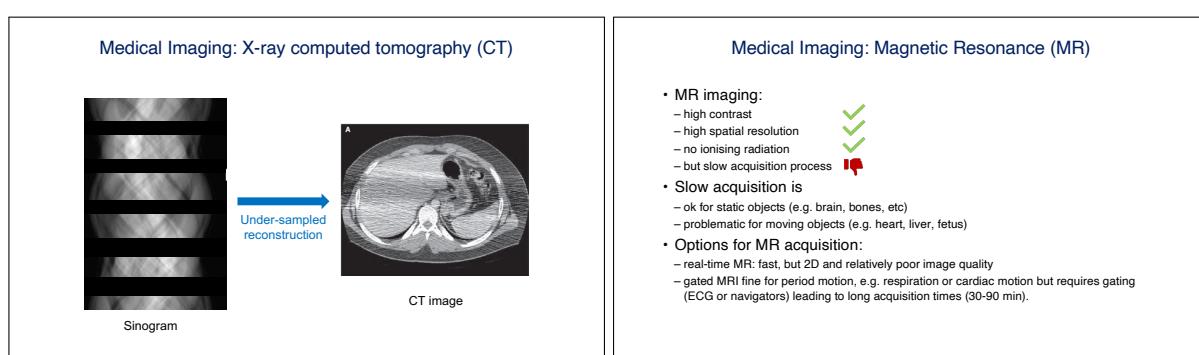
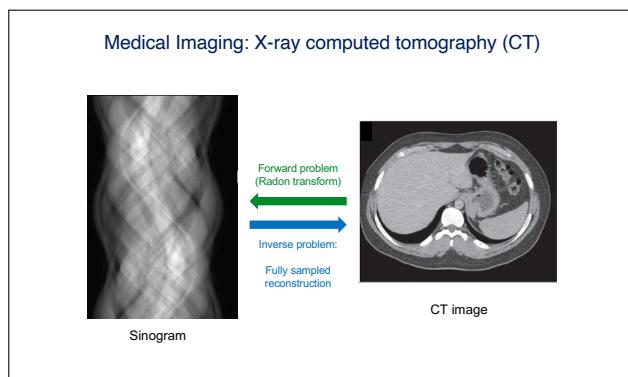
4 Deep Learning for Image Reconstruction



An x-ray scan captures information by measuring the x-ray energy on the opposite side of the emitter as it passes through soft-tissue. This is rotated around the patient

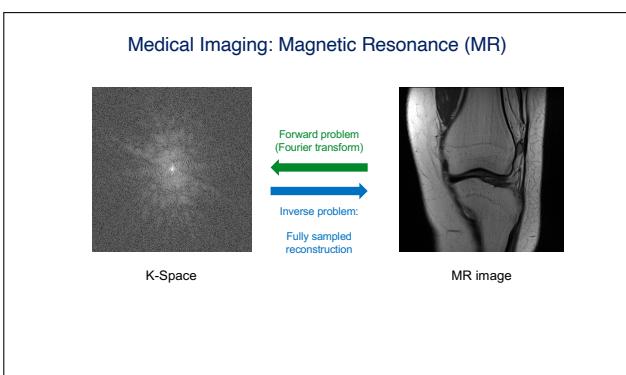
4.1 Going from Sinogram to CT image

- We know how to solve the forward transform.
- However, how do we solve the inverse problem?
- Typically if we have measured a lot of data then you can just compute the inverse radon trasnform but typically you can't do that because you dont have enough data. This may be that for example, a patient may have to hold their breath when giving an MR heart scan, so you don't want them in there for long.



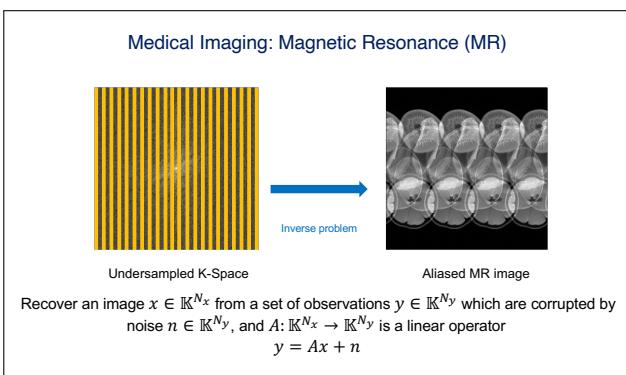
(q) Can also contain artifacts

4.2 Going from K-space to MR image

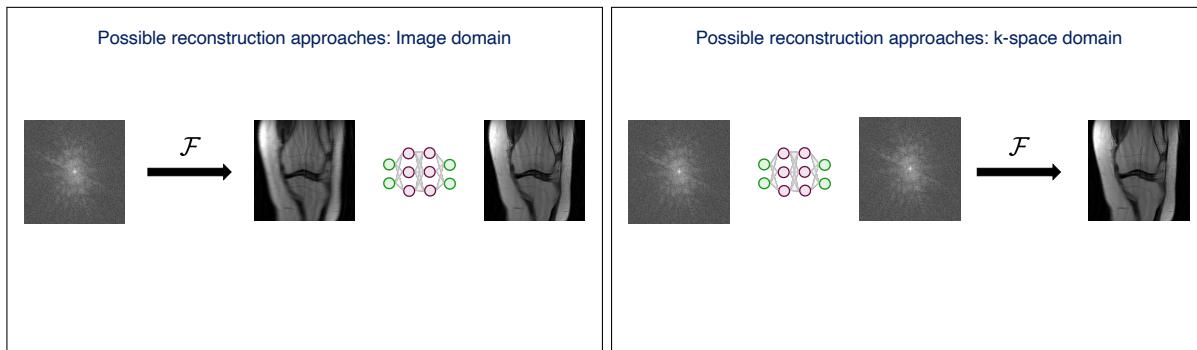


- Similarly, we know the forward problem, but the inverse is harder.
 - you can't really tell what the original problem was, you can scan something random and the output looks

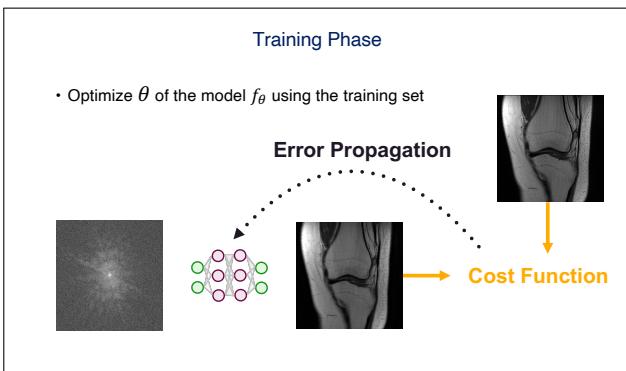
The task becomes “Recover an image $x \in \mathbb{K}^{N_x}$ from a set of observations $y \in \mathbb{K}^{N_y}$ which are corrupted by a noise $n \in \mathbb{K}^{N_y}$, and $A : \mathbb{K}^{N_x} \rightarrow \mathbb{K}^{N_y}$ is a linear operator $y = Ax + n$ ”



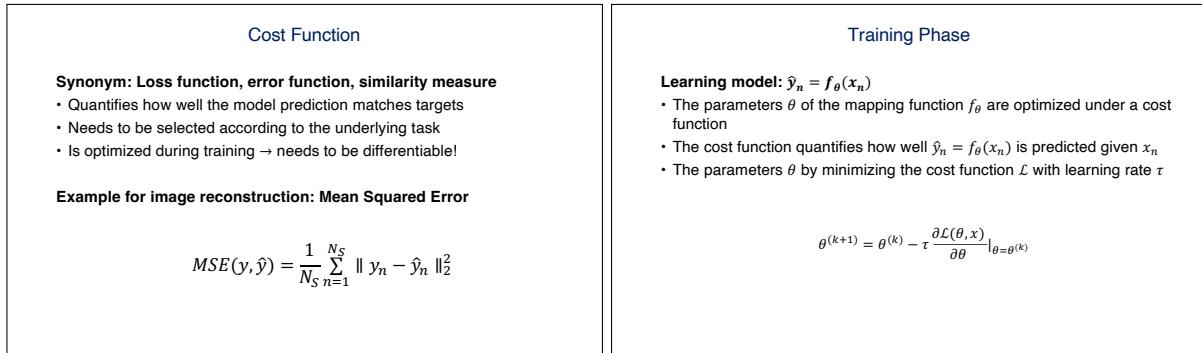
- We can similarly have an issue with under-sampled data.



- (s) How can we formulate the problem? Use as much physics knowledge as possible (we know there is a Fourier frequency space and then upsample with inverse of Fourier transformation), but this gives a noisy intermediate representation. Therefore, use an nn to remove artifacts.

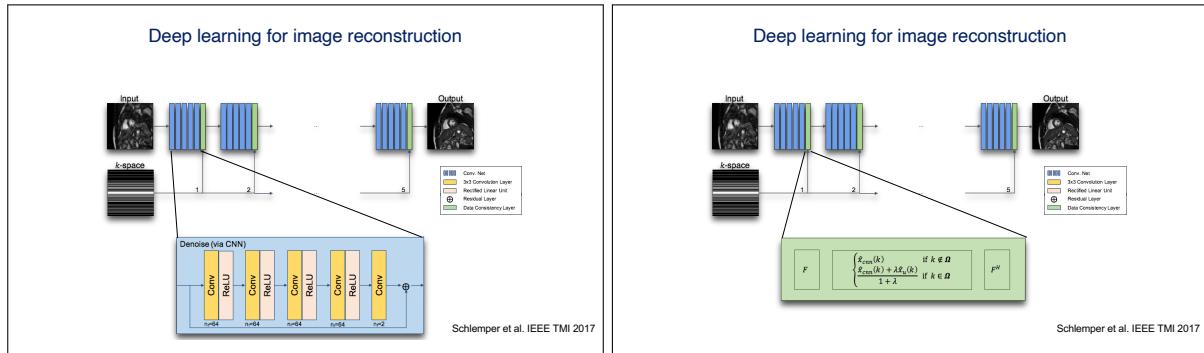
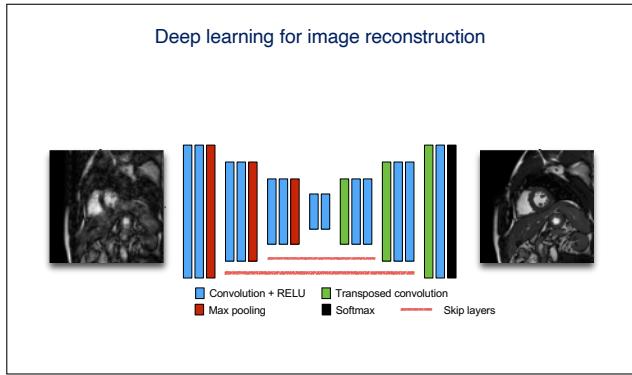
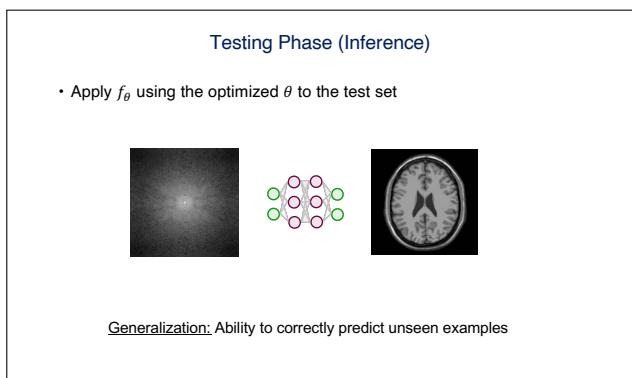


- generate the training data using the forward model
 - compare the reconstructed image to the ground truth image
 - minimise the cost function, the loss function between these reconstructed image and ground truth images
 - back propagation through the neural network in the standard way (e.g. with back-propagation)



(u) If using the framework above, you need a loss function which is differentiable; so L2 loss between the ground truth image and reconstructed image (MSE) or you can also use the GAN loss function

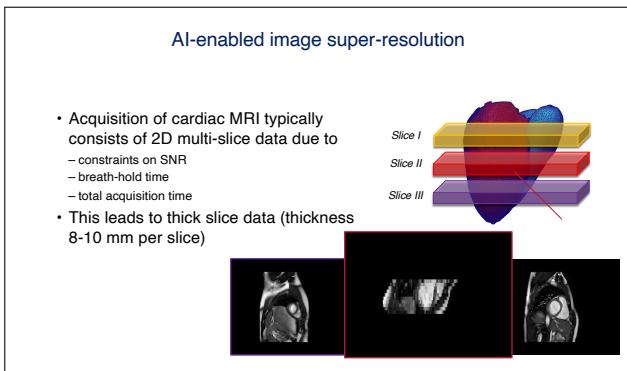
(v) Backpropagation



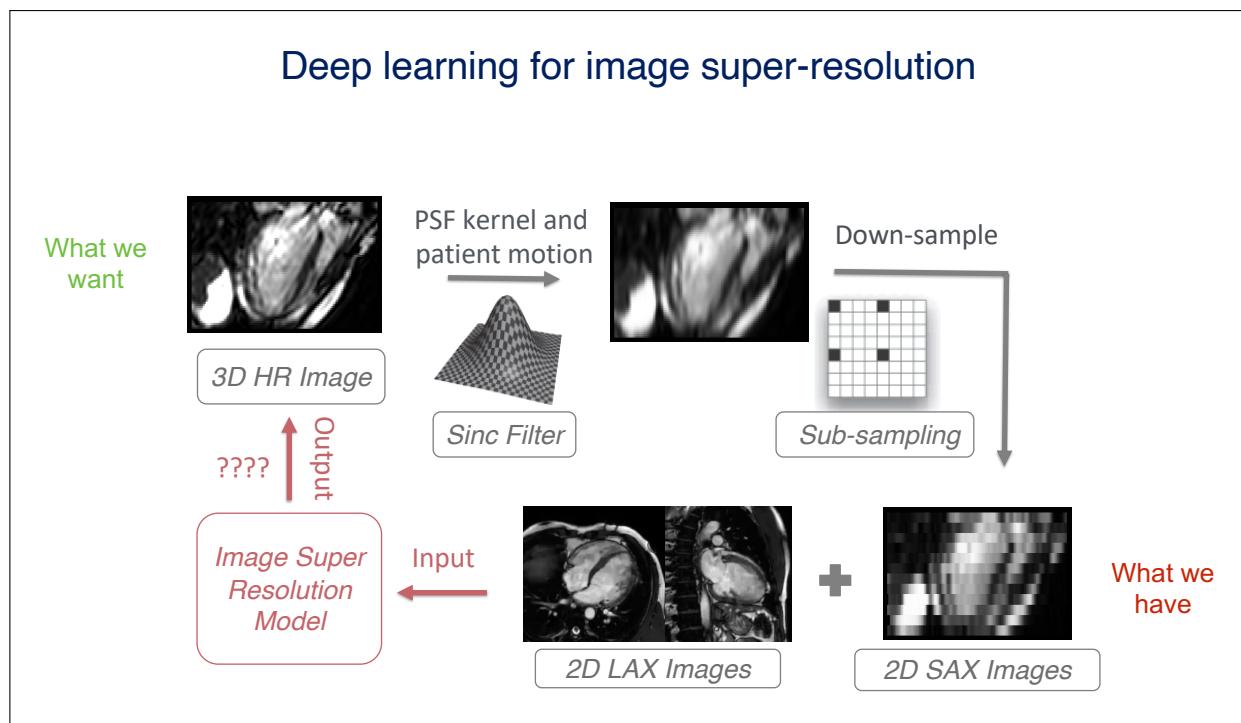
(w) Have an nn which in the image space does de-noising operations

(x) Then also goes with a Fourier transform into the signal domain and makes sure that the data you have reconstructed is consistent with the data you have acquired.

5 AI-enabled Super resolution Images

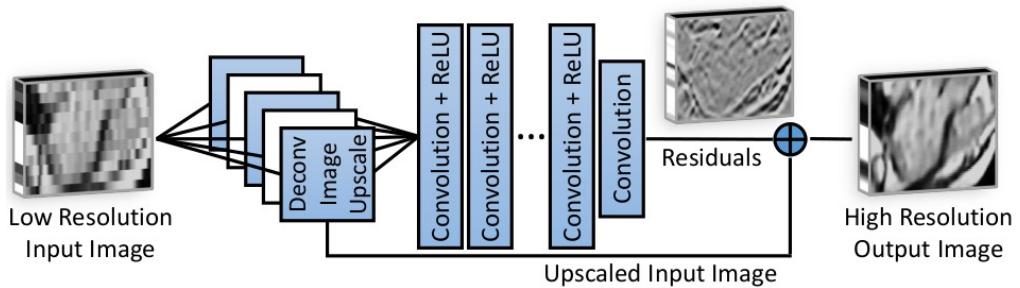


- acquire thick slices through the heart. The data doesn't look very high resolution



Formula a deep learning pipeline, and take the 3D image and 2d images from which we want to recover our 3D data and develop a simulation for the forward process and generate pairs of high and low resolution data and train the nn to absorb all the images again to a high resolution.

Deep learning for image super-resolution



O. Oktay et al. IEEE TMI 2017

This is something you can use a residual network for