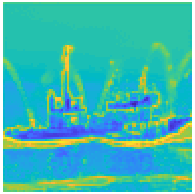
Networks for nonlinear diffusion problems

Andreas Hauptmann
(andreas.hauptmann@oulu.fi)
joint with Simon Arridge

University of Oulu
Research Unit of Mathematical Sciences
&
University College London
Centre for Medical Image Computing
Department of Computer Science



2nd IMA Conference On Inverse Problems
From Theory To Application 2019
University College London, 04 September 2019

Deep Learning in imaging

Find a non-linear mapping $A_\Theta : X \to Y$ parametrised by a finite set of parameters Θ, which need to be learned, such that

$$g = A_\Theta u$$

A few questions arise:

- how and to what extent can learned models replace physical models?
- what are appropriate architectures for the learned models, what is the size of the parameter set Θ that needs to be learned, and how can these be interpreted?

[Ye, Han, Cha; 2018], [Haber, Ruthotto; 2018], [Ruthotto,Haber; 2018]

Popular network architectures

The most popular network architectures are based on (discrete) convolutions.

We recall that the convolution of two functions $f, g \in L^1(\mathbb{R}^d)$ is defined by

$$(g * f)(x) = \int_{\mathbb{R}^d} g(x-y)f(y)\mathrm{d}y.$$

Two major question arise:

- Are convolutions suited to capture the physics for a large class of problems?
- Can we improve interpretability by using other underlying operations?

Considering general linear integral transforms

Consider mappings between images in dimension d, e.g. $X = Y = L^p(\Omega \subset \mathbb{R}^d)$.

For example a general linear integral transform such as

$$u^{\mathrm{obs}}(x) = \int_\Omega K(x, y) u^{\mathrm{true}}(y) \mathrm{d}y$$

includes stationary convolution as a special case if the kernel is translation invariant, i.e. $K(x, y) \equiv K(x - y)$.

Outline

Outline

General diffusion processes

We consider the general diffusion process in $\mathbb{R}^d$. On a fixed time interval with some diffusivity $\gamma(u)$, we have

$$\begin{cases} \partial_t u = \nabla \cdot (\gamma(u)\nabla u) & \text{in } \mathbb{R}^d \times (0, T] \\ u(x,0) = u_0(x) & \text{in } \mathbb{R}^d. \end{cases}$$

We denote the spatial derivative by

$$\left(\mathcal{L}(\gamma)u\right)(x,t) := \nabla \cdot (\gamma(u)\nabla u(x,t)).$$

Stationary diffusion as convolution

For *isotropic diffusion* the differential operator becomes the spatial Laplacian $\mathcal{L}(\gamma = 1) = \Delta$.

In this case, the solution u at time T is given by convolution with a *Green's function*

$$u_T(x) = G_{\sqrt{2T}}(x) * u_0(x)$$

where $G_{\sqrt{2T}} = \frac{1}{(4\pi T)^{d/2}} \exp\left[-\frac{x^2}{4T}\right]$ in dimension d.

Nonlinear diffusion

In the general case for an *anisotropic diffusion flow* (ADF) we are interested in a scalar *diffusivity* $\gamma \in [0,1]$ that depends on u itself, i.e.

$$\partial_t u = \nabla \cdot (\gamma(u) \nabla u) \,.$$

This is an example of a non-linear evolution

$$u_T(x) = \mathcal{K}_T u_0 = \int_0^T \int_{\mathbb{R}^d} K^{\mathrm{ADF}}(x, y, u(y,t)) u_0(y) \mathrm{d}y \mathrm{d}t$$

where $K^{\mathrm{ADF}}(x, y, u(x,t))$ is now a non-stationary, non-linear and time-dependent kernel.

In general there is no explicit expression for K^{ADF} and numerical methods are required for the solution.

[Weickert, 1998]

Time discretisation

To establish a process between two states of the function u, denoted at time t_n by $u^{(n)} = u(x, t_n)$, then with $t_{n+1} = t_n + \delta t$:

$$u^{(n+1)} = u^{(n)} + \int_{t_n}^{t_{n+1}} \mathcal{L}(\gamma) u(x, t) \, \mathrm{d}t.$$

Consider a fixed diffusivity at each time instance $\gamma^{(n)} = \gamma(u(x, t = t_n))$, then the integral is approximated by

$$\delta t \mathcal{L}(\gamma^{(n)}) u^{(n)} = \delta t (\nabla \cdot \gamma^{(n)} \nabla u^{(n)}).$$

We obtain an approximate solution by iterating for time steps δt using either an *explicit* scheme

$$\mathcal{D}_{\delta t}^{\mathrm{Expl}}(\gamma^{(n)}) u^{(n)} = \left(Id + \delta t \mathcal{L}(\gamma^{(n)}) \right) u^{(n)} \, ,$$

or an *implicit* scheme

$$\mathcal{D}_{\delta t}^{\mathrm{Impl}}(\gamma^{(n)}) u^{(n)} = \left(Id - \delta t \mathcal{L}(\gamma^{(n)}) \right)^{-1} u^{(n)} \, .$$

Outline

Motivation: From continuum to discrete

We formulate a continuum network, where each layer is interpreted as time instance:

$$F_k = u(x, t_k)$$

Layers are then connected by diffusion layer operator, with learnable parameter $\Theta = \{\gamma, \delta t\}$, and given by the time stepping scheme

$$\mathcal{D}_\Theta F_{k-1} = \mathcal{D}_{\delta t}^{\mathrm{Expl}}(\gamma^{(k-1)})u^{(k-1)} = (Id + \delta t \mathcal{L}(\gamma^{(k)}))u^{(k-1)} = F_k.$$

Discretisation: Diffusion layer

Consider two-dimensional $n \times n$ images and denote the discrete vectorised layers as $\mathbf{F}_k \in \mathbb{R}^{n^2}$.

We consider the explicit diffusion layer operator

$$\mathcal{D}_{\delta t}^{\mathrm{Expl}}(\gamma^{(k-1)})u^{(k-1)} = (Id + \delta t \mathcal{L}(\gamma^{(k)}))u^{(k-1)}$$

.

Then the differential operator $\mathcal{L}(\gamma^{(k)}) = \nabla \cdot \gamma^{(k)} \nabla$ can be approximated by the stencil

$$\mathsf{L}_{\delta t}(\gamma^{(k)}) = \delta t \begin{pmatrix} & \gamma_1^{(k)} & \\ \gamma_2^{(k)} & -\sum_i \gamma_i^{(k)} & \gamma_4^{(k)} \\ & \gamma_3^{(k)} & \end{pmatrix}.$$

Linear diffusion network

The discrete diffusion layer is then given by

$$\mathbf{F}_k = (\mathrm{Id} + \mathrm{L}_{\delta t}(\gamma^{(k)}))\mathbf{F}_{k-1}.$$

The basis of learning a linear diffusion network is now given as estimating the diagonals of $\mathrm{L}_{\delta t}(\gamma^{(k)})$ and the time-step δt:

$$\begin{tikzcd}
\boxed{\mathbf{F}_0} \arrow[r, "\mathrm{L}_{\delta t}(\gamma^{(1)})"] \arrow[r, bend left, "+\mathrm{Id}"] & \boxed{\mathbf{F}_1} \arrow[r, "\mathrm{L}_{\delta t}(\gamma^{(2)})"] \arrow[r, bend left, "+\mathrm{Id}"] & \boxed{\mathbf{F}_2} \arrow[r, "\mathrm{L}_{\delta t}(\gamma^{(3)})"] \arrow[r, bend left, "+\mathrm{Id}"] & \boxed{\mathbf{F}_3}
\end{tikzcd}$$

Discretisation: Inverse filtering

For the inversion task we need to include regularisation.

Especially in the presence of noise, the inverse operator $\mathcal{E}_{\delta t}(\zeta)$ has to be regularised by addition of a smoothing operation $\mathcal{S}$:

$$\mathcal{E}_{\delta t}(\zeta) + \alpha\mathcal{S} \to \mathsf{Id} - \mathsf{L}_{\delta t}(\zeta) + \alpha\mathsf{S} =: \mathsf{Id} - \mathsf{W}_{\delta t}(\zeta)$$

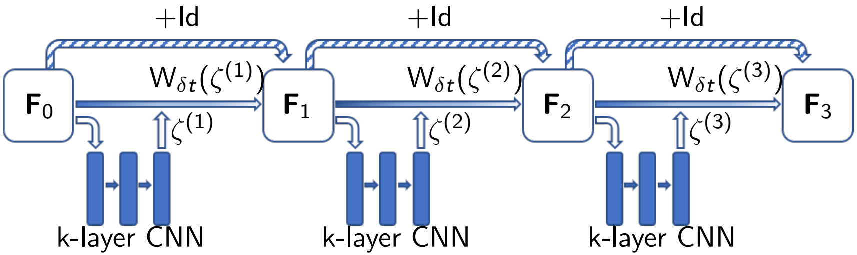
With the general filter matrix

$$\mathsf{W}_{\delta t}(\zeta) = \delta t \begin{pmatrix} & \zeta_1 & \\ \zeta_2 & -\zeta_5 & \zeta_4 \\ & \zeta_3 & \end{pmatrix}.$$

A nonlinear diffusion network: DiffNet

For nonlinear diffusion/filtering, the filters depend on the solution u
$\Rightarrow$ we can not learn the filters explicitly.

We rather estimate the filters ζ implicitly by a small k-layer convolutional neural network (CNN), which are then applied to the image in the filtering layer:



Outline

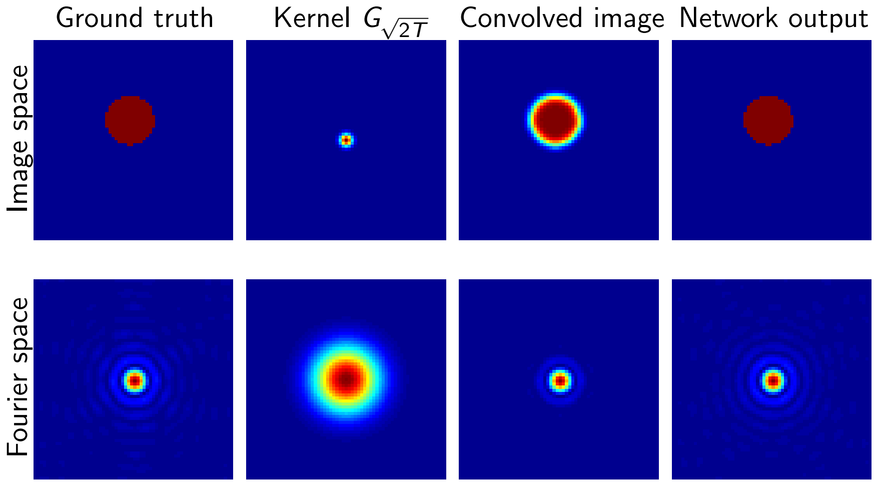
First experiment: Simple deconvolution

We first examine a simple deconvolution experiment to determine what features DiffNet learns in an inverse problem.
We consider:

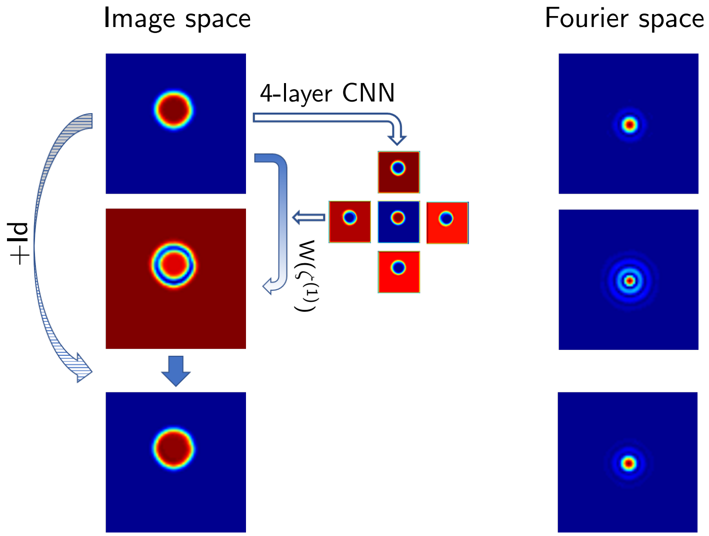
$$\begin{cases} \partial_t u = \nabla \cdot (\gamma \nabla u) & \text{in } \Omega^d \times (0, T] \\ \partial_\nu u = 0 & \text{on } \partial\Omega \times (0, T] \\ u(x,0) = u_0(x) & \text{in } \Omega^d. \end{cases}$$

With constant diffusivity $\gamma \equiv 1$ and $T = 1$.

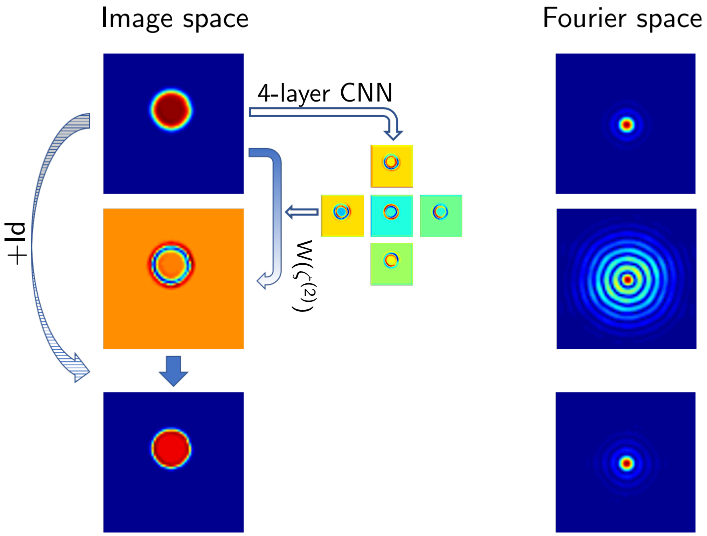
Illustration



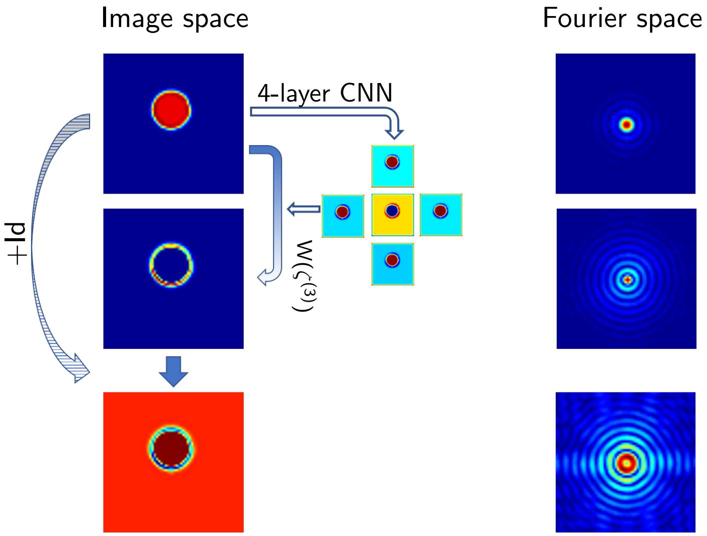
Break down of DiffNet: First layer



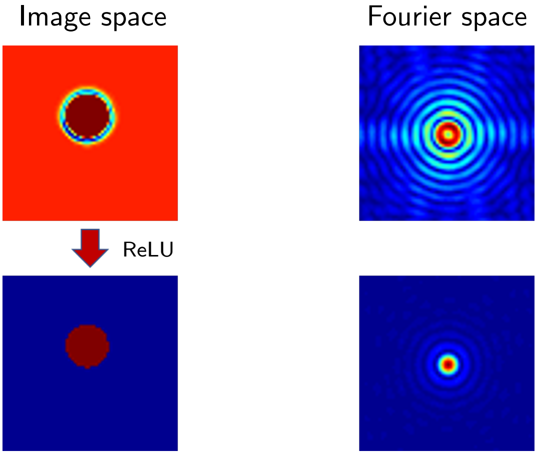
Break down of DiffNet: Second layer



Break down of DiffNet: Third layer



Break down of DiffNet: Output



Nonlinear diffusion

Let us now consider the nonlinear diffusion process with the Perona-Malik filter function [Perona & Malik, 1990], that is the diffusivity is given as a function of the gradient

$$\gamma(|\nabla u|^2) = \frac{1}{1 + |\nabla u|^2/\lambda^2}$$

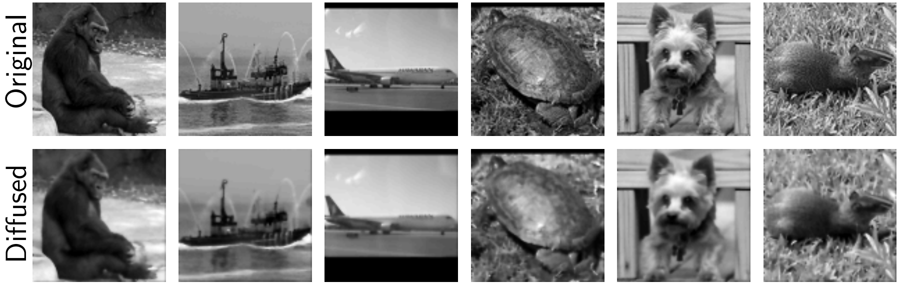
with contrast parameter $\lambda > 0$.

We concentrate on the inverse problem of restoring an image that has been diffused and contaminated by noise.

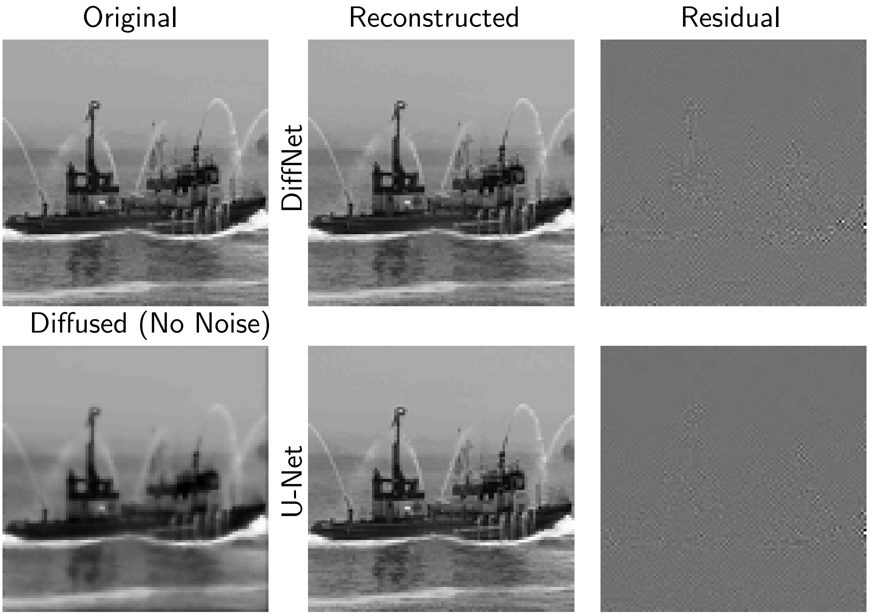
Training data

Data from the STL-10 database [Coates & Ng, 2011]: 100,000 images, resolution 96×96. (90,000 training, 10,000 test).

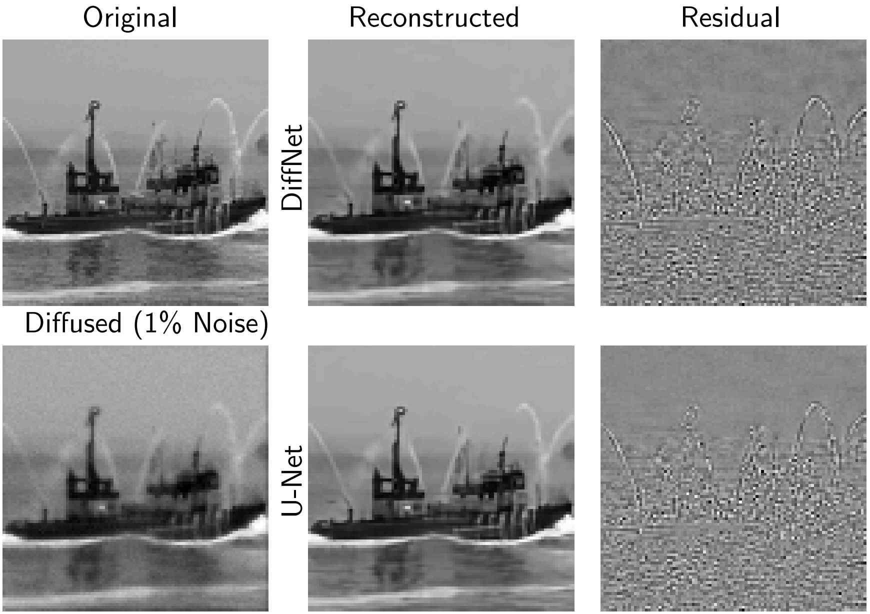
Diffused for 4 time steps with $\delta t = 0.1$ and $\lambda = 0.2$:



Comparison: Inversion (no noise)



Comparison: Inversion (1% noise)

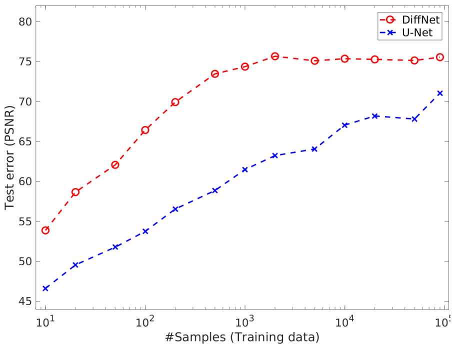


Some remarks

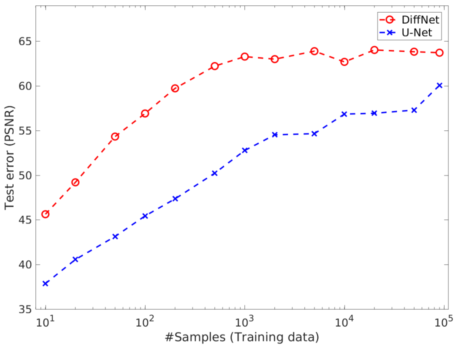
- Updates in DiffNet are performed explicitly and the CNN in the architecture is only used to produce the filters ζ.
- Hence DiffNet needs to learn a problem specific processing, in contrast to a purely data driven processing in a CNN.
- Amount of necessary learnable parameters is much lower:
  5-layer DiffNet: 101,310 parameters;
  U-Net with filter size 3×3: 34,512,705
- DiffNet uses only $\sim 0.3\%$ of parameters compared to U-Net.

Test error vs. training size

Forward Problem

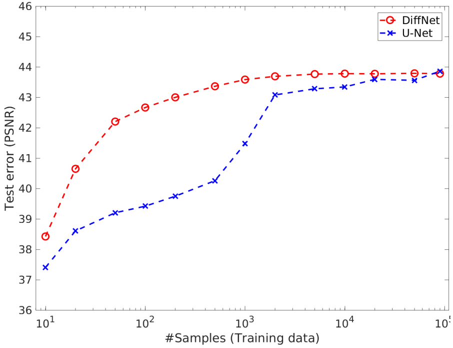


Inverse Problem (no Noise)

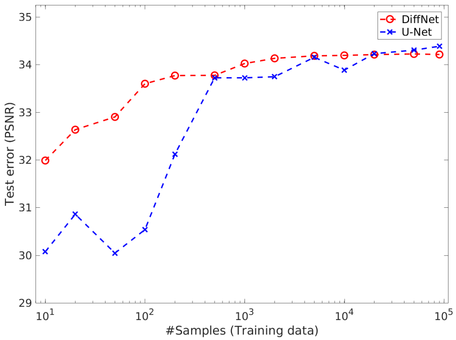


Test error vs. training size

Inverse Problem (0.1% noise)



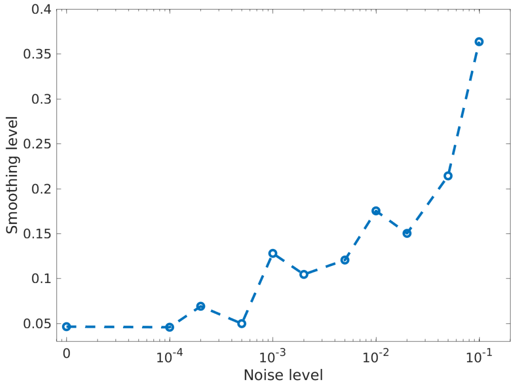
Inverse Problem (1% noise)



Increasing smoothing with noise level

We conjectured, that the learned update filters can be decomposed $W(\zeta) = L(\zeta) + \alpha S(\zeta)$. Thus, the magnitude α has to increase with higher noise.

We computed an estimate of S as $\sum_{i=1}^{4} \zeta_i - \zeta_5$ and interpret the smoothing level α as the mean of the absolute value of S.



Take home message

- By taking physics into account, we can obtain more expressive network architectures
- Reduction in necessary parameters and some interpretability

Reference: [Arridge, H, Journal of Mathematical Imaging and Vision (accepted)]

Acknowledgements

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

EPSRC
Engineering and Physical Sciences
Research Council

Finnish Centre of Excellence in
Inverse Modelling and Imaging
2018-2025

NVIDIA.

Thank you for your attention