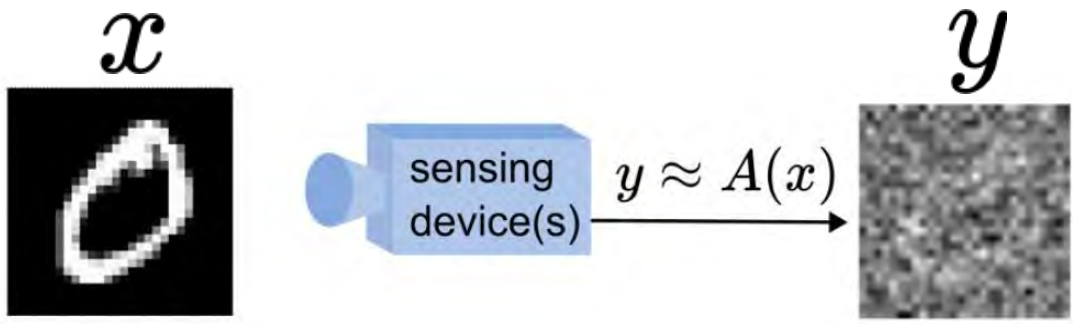
DeepInverse: a Pytorch library for imaging with deep learning

October 2024
Workshop Exa-DI "Artificial Intelligence for HPC@Exscale"

Julián Tachella
CNRS
Physics Laboratory
École Normale Supérieure de Lyon

Inverse Problems in a Nutshell



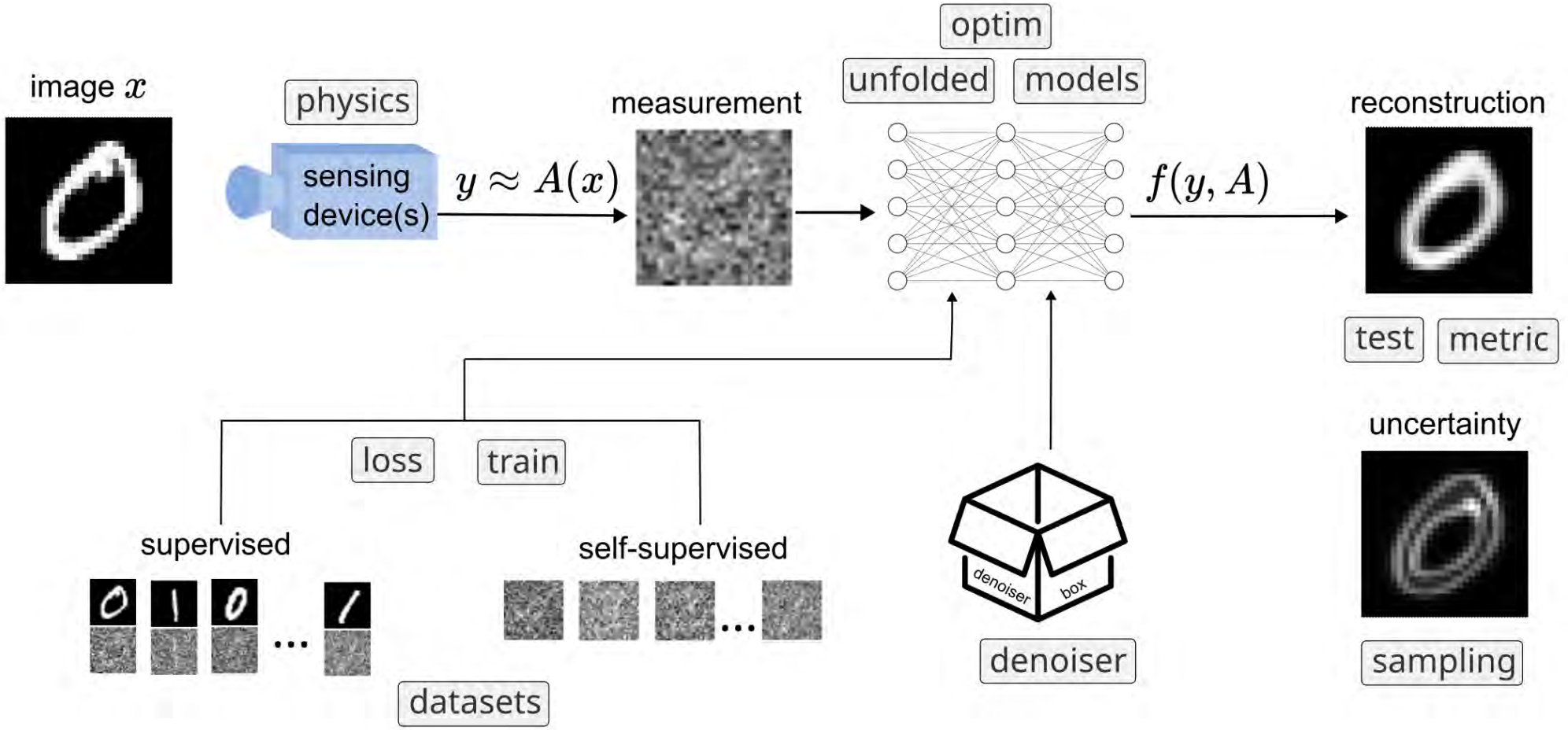
Forward problem

$$p(y|x) = \mathcal{N}(Ax, I\sigma^2)$$

- Incomplete & noisy measurements

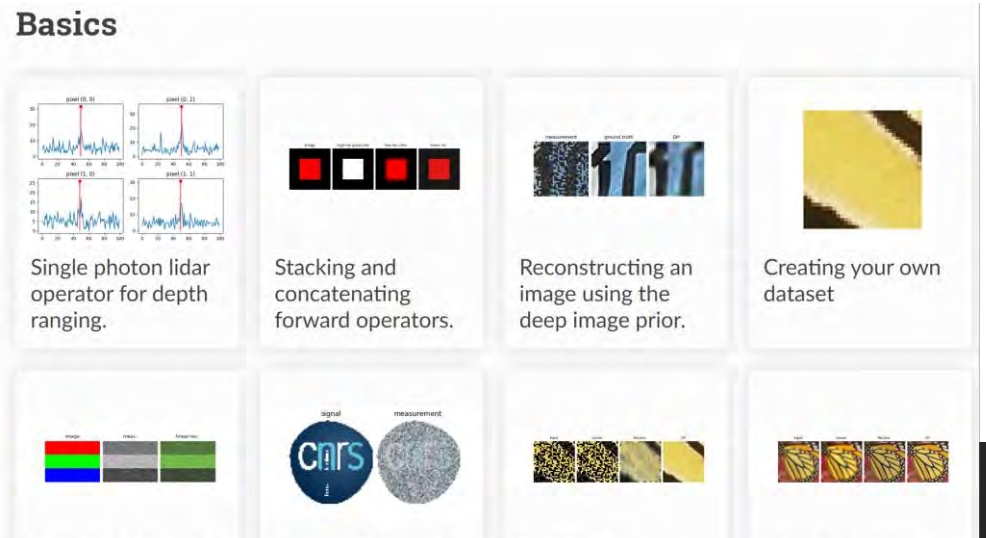
- Learn $p(x|y)$ or $p(x)$ from data!
- Maximize $p(x|y)$: PnP, unfolded
- Sample $p(x|y)$: Diffusion, MCMC

Deep Inverse



Overview

- Launched in July 2023
- Install the library `pip install deepinv` then imp `import deepinv as dinv`
- Everything is a PyTorch module, i.e., you can backprop through
- **Detailed** docs + lots of **jupyter notebook** examples!



When Should I Use It?

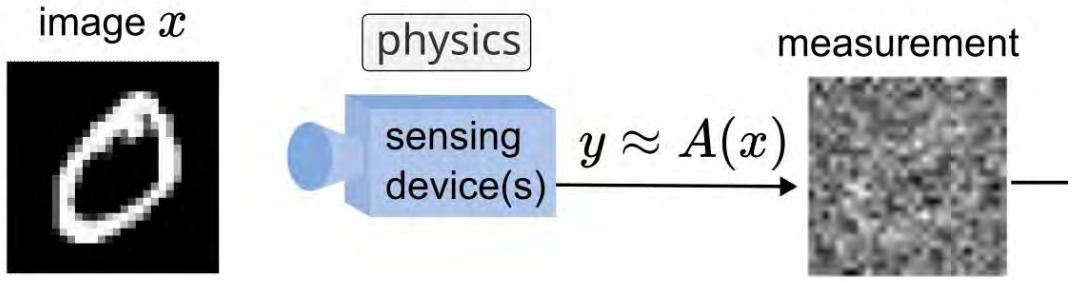
Developing **new reconstruction methods**

- Build your algo without reinventing the wheel
- Try out new method in multiple inverse problems
- Reproducibility

Solving your **specific inverse problem**

- Quickly try SOTA methods
- Create & share dataset
- Reproducibility

Forward Operators



- Access to useful properties

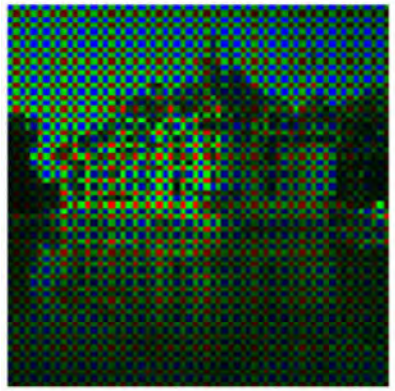
```
xhat = operator.A_dagger(y)
norm = operator.compute_norm(x)
```

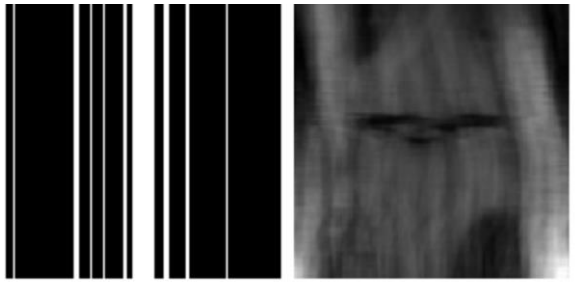
```
operator = dinv.physics.Blur(…)
```

```
y = operator(x)
```

Forward Operators

- **Forward operators**
  - Pixelwise operators
  - Blur & Super-Resolution
  - Magnetic Resonance Imaging
  - Tomography
  - Remote Sensing
  - Compressive operators
  - Radio interferometric imaging
  - Single-photon lidar
  - Dehazing
  - Phase retrieval





Forward Operators

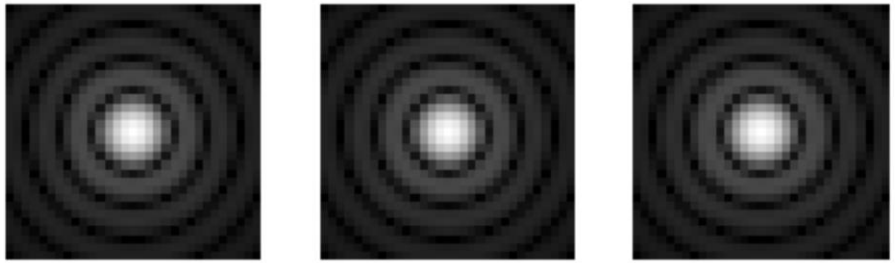
- Physics generators

```
kernels = kernel_generator.step()
y = physics(x, kernels)
```

Different length and regularity

Examples of randomly generated diffraction blurs

Airy pattern



Denoisers $p(x)$

- Classical

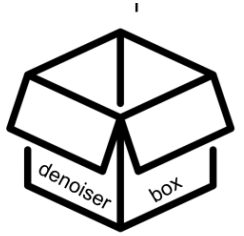
```
denoiser = dinv.models.BM3D(…)
denoiser = dinv.models.TGV(…)
```

- Pretrained

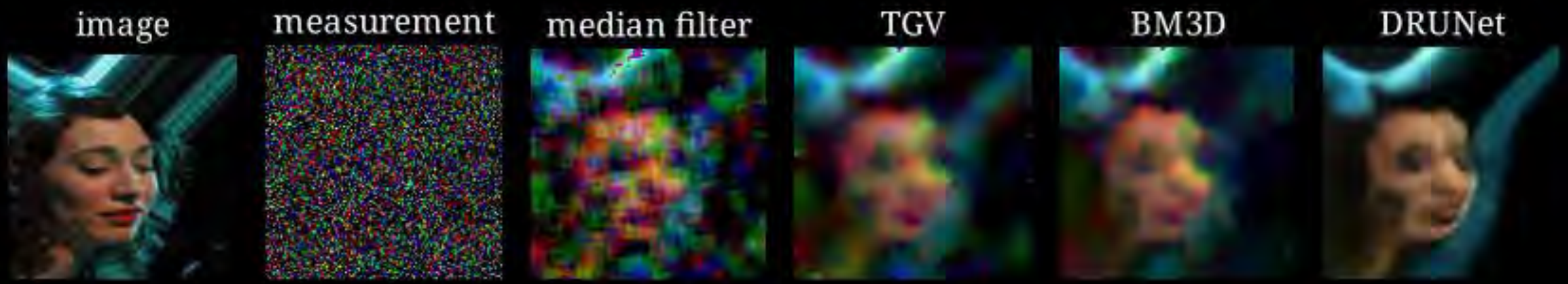
```
denoiser = dinv.models.DnCNN(…)
denoiser = dinv.models.DRUNet(…)
```

- Evaluate denoiser

```
xhat = denoiser(y, sigma)
```



denoiser



Gaussian denoising: $y \sim \mathcal{N}(x, I\sigma^2)$, $\sigma = 1.00$

```python
import deepinv as dinv

physics = dinv.physics.Denoising(dinv.physics.GaussianNoise(1.00))
y = physics(x)

denoiser1 = dinv.models.MedianFilter()
denoiser2 = dinv.models.TGV()
denoiser3 = dinv.models.BM3D()
denoiser4 = dinv.models.DRUNet(pretrained='download', ...)

xhat = denoiserx(y, sigma=1.00)
```

PnP, Diffusion & Unfolded

- Plug-and-Play

```
model = dinv.optim.optim_builder("PGD", denoiser, …)
```

- Diffusion

```
model = dinv.sampling.DiffDPIR(denoiser)
```

- Unfolded

```
model = dinv.optim.unfolded_builder("PGD", …)
```

- Reconstruct

```
xhat = model(y, operator)
```

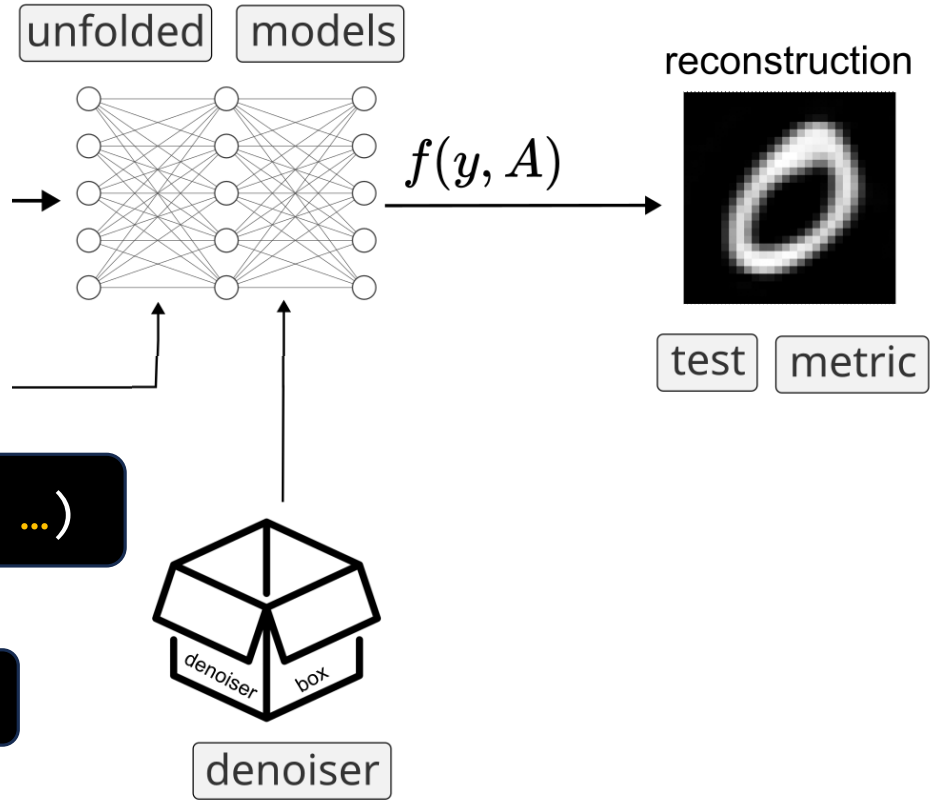


Image inpainting: $y = \text{diag}(m)x$, $m_i \sim \mathcal{B}e(p = 1.00)$

```python
import deepinv as dinv

physics = dinv.physics.Inpainting(mask=1.00, ...)
y = physics(x)

model = dinv.optim.optim_builder("PGD", max_iter=2e3, ...)
model = dinv.unfolded.unfolded_builder("PGD", max_iter=4, ...)
model = dinv.sampling.DDRM(denoiser=DRUNet(...), ...)
model(y, physics)
```

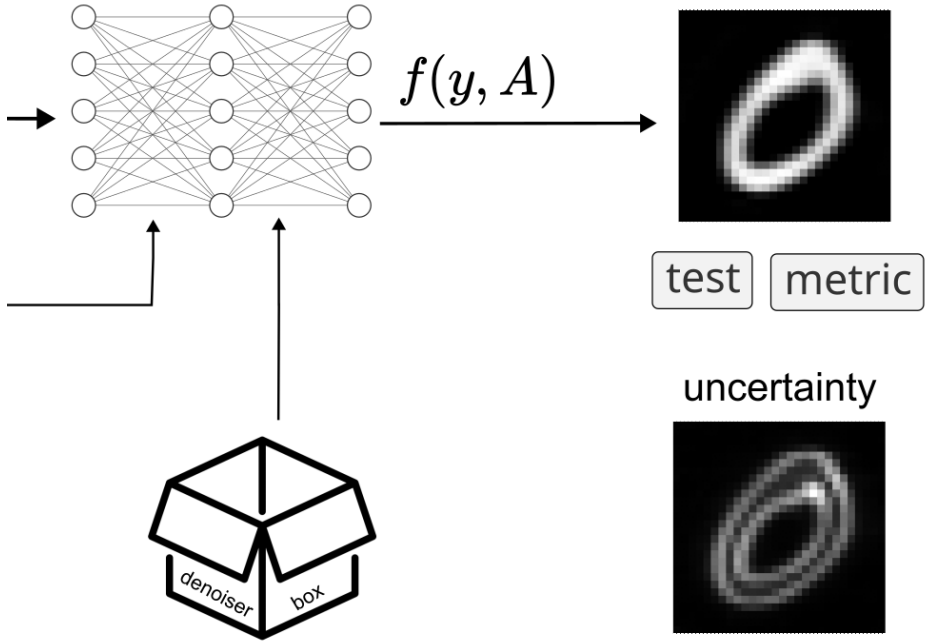
Uncertainty Quantification $p(x|y)$

- MCMC methods

```
prior = dinv.optim.ScoreDenoiser(denoiser)
likelihood = dinv.optim.L2(sigma)
model = dinv.sampling.ULA(prior, likelihood, …)
```

- Diffusion

```
model =
dinv.sampling.DiffusionSampler(diff)
```

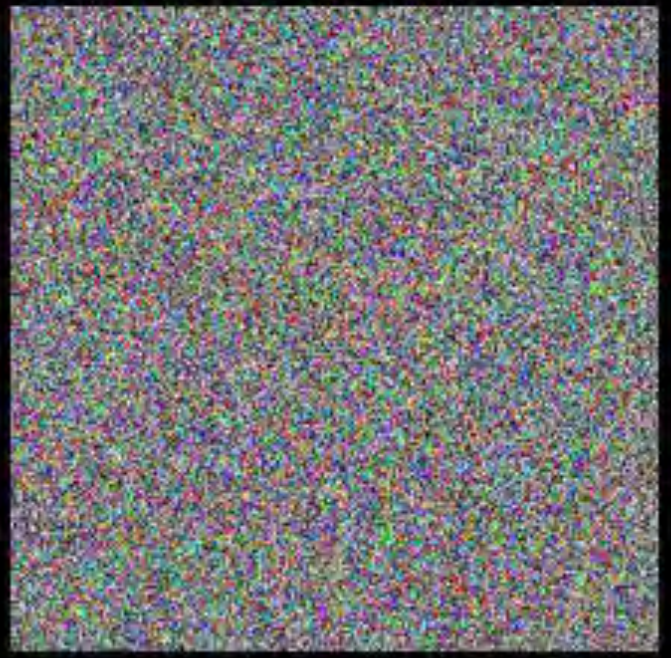


- Quantify:

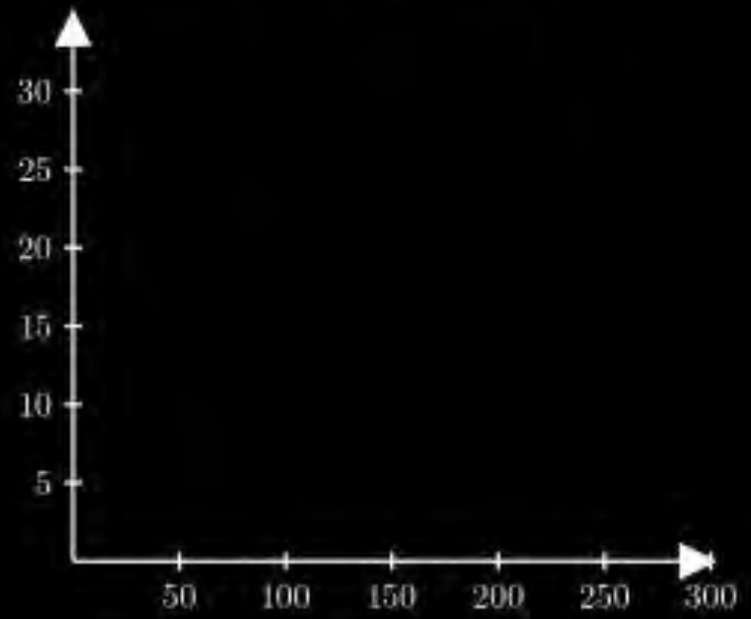
```
mean, variance = model(y, operator)
```

measurement

estimate



PSNR [dB]




```python
import deepinv as dinv
model = dinv.sampling.DPS(model=dinv.models.DiffUNet())
xhat = model(y, physics)
```

Datasets and training

Generate dataset

```
dinv.datasets.generate_dataset(MNIST, operator, …)
```

Train

Test

```
Trainer.test(…)
```

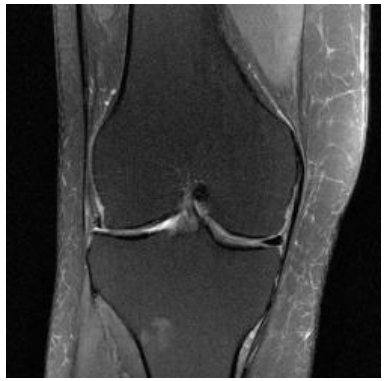
```
Trainer.train(…)
```

Datasets and training

Loaders for popular datasets

```
dinv.datasets.DIV2K(…)
```

```
dinv.datasets.FastMRI(…)
```



```
dinv.datasets.LidcIdri(…)
```

Training Losses

- Network regularization

```
dinv.loss.JacobianSpectralNorm(…)
```

Noisy data with known noise

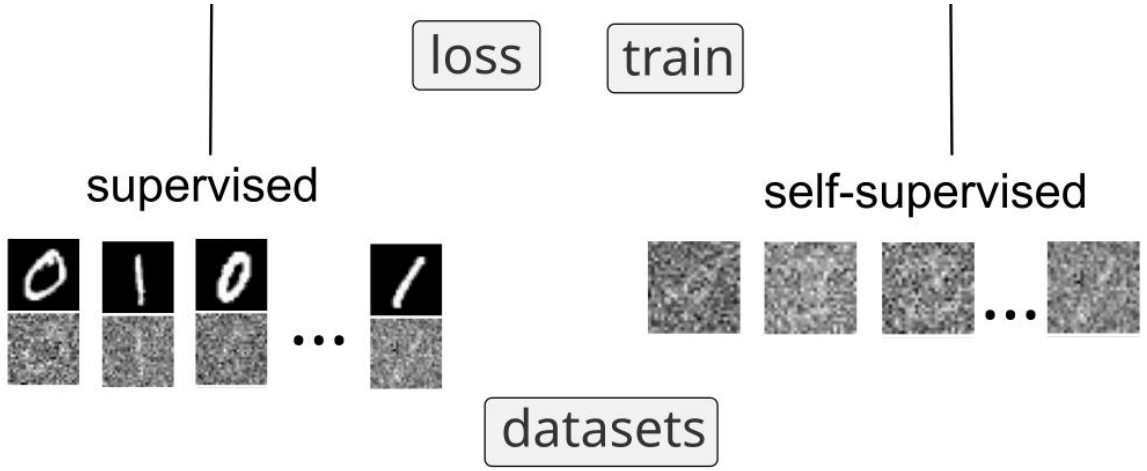
```
dinv.loss.SUREGaussianLoss(…)
```

Noisy data with uncorrelated noise

```
dinv.loss.Neighbor2Neigbor(…)
```

Incomplete data

```
dinv.loss.EILoss(…)
```



What about HPC?

CNRS PNRIA project: help from IDRIS engineers (Maxime Song, Antoine Regnier)

Deep Inverse Team

Matthieu Terris
(Inria Saclay)

Samuel Hurault
(IMBordeaux)

Dongdong Chen
(Heriot-Watt University)

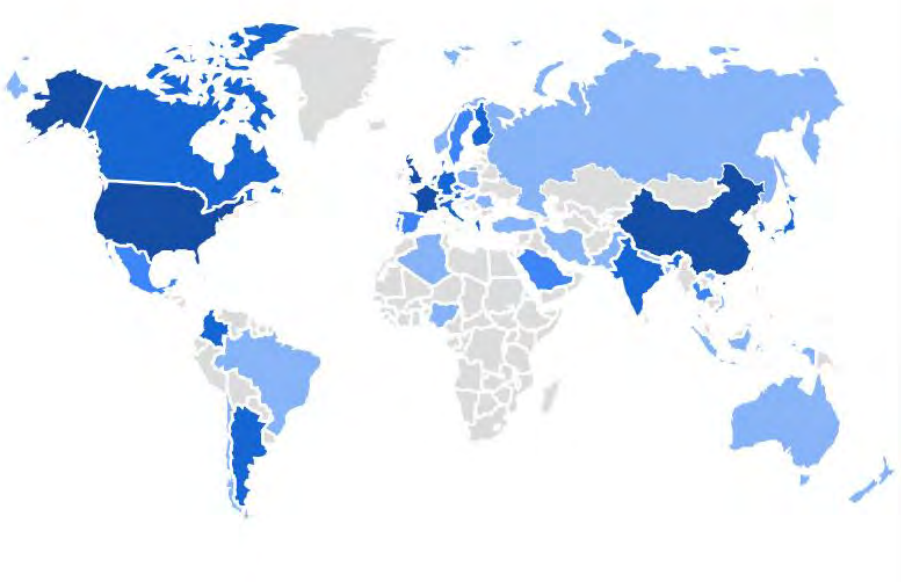
Andrew Wang
(University of Edinburgh)

And all the contributors of the library!

Contributing

All help is welcome!

- Library is continuously evolving
- Community-driven effort
- All contributions (small or big) will be appropriately acknowledged



How to start?

- Chat with us
- Raise an issue in the GitHub repo
- Join the conversation in our discord channel

Coming Soon

- More diffusion-based methods, learnable forward operators
- Integration with Benchopt for benchmarking inverse problems
- More forward operators (multicoil MRI, etc)
- Coding sprints (October’24 @ CIRM)!

Thanks for your attention!

Tachella.github.io

- ✓ Codes
- ✓ Presentations
- ✓ … and more

 @TachellaJulian