

AI-based denoising of scientific images

(Image to Image translation, e.g., inpaint, denoise, segmentation etc.)

Zhengchun Liu

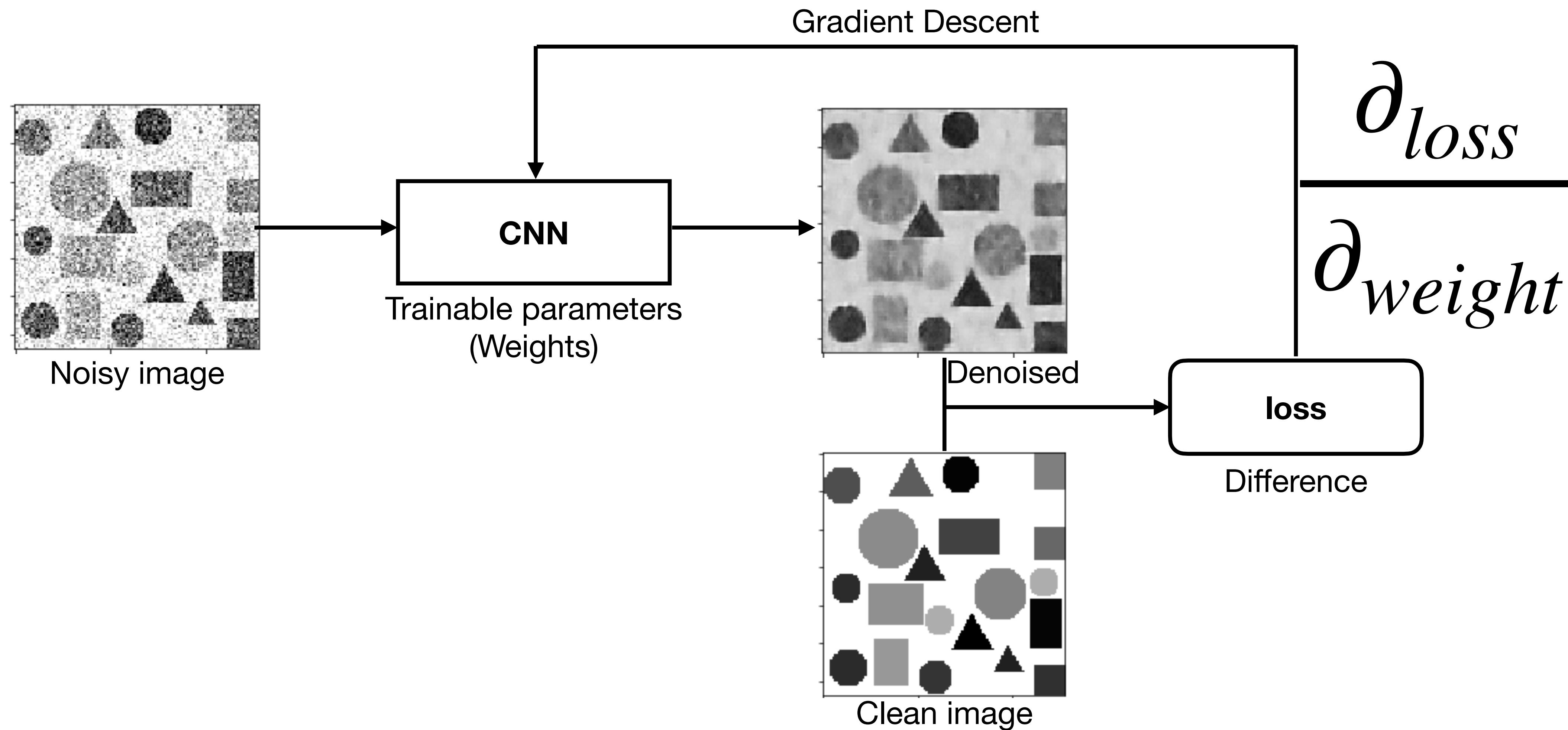
Assistant Computer Scientist at Data Science and Learning Division

Agenda

- ❑ Using CNN for denoising
- ❑ TomoGAN for denoising (e.g., artifact removal)
- ❑ [Advanced] Self-supervised learning to train denoise models without clean images
- ❑ [Advanced] Noise2Noise: Learning Image Restoration without Clean Data
- ❑ [Advanced] Segmentation

Fully Convolution Neural Network for denoising

Supervised learning - train a CNN using clean (noise-free images) as ground truth

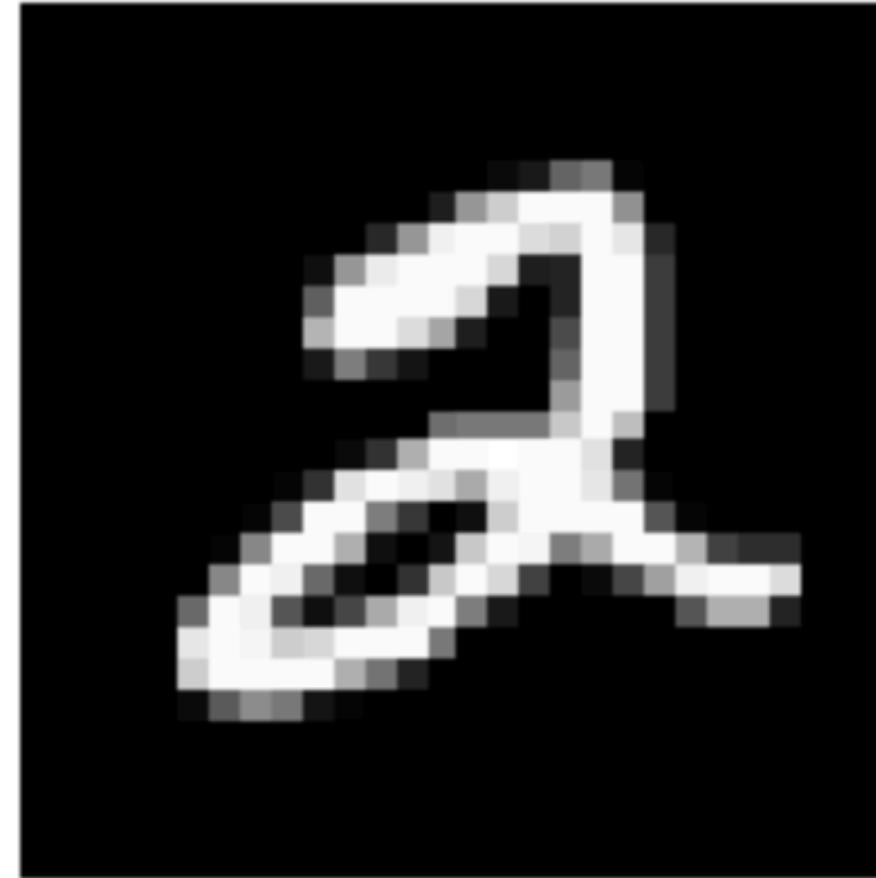


Switch to hands-on, open Jupiter notebook cnn4denoise

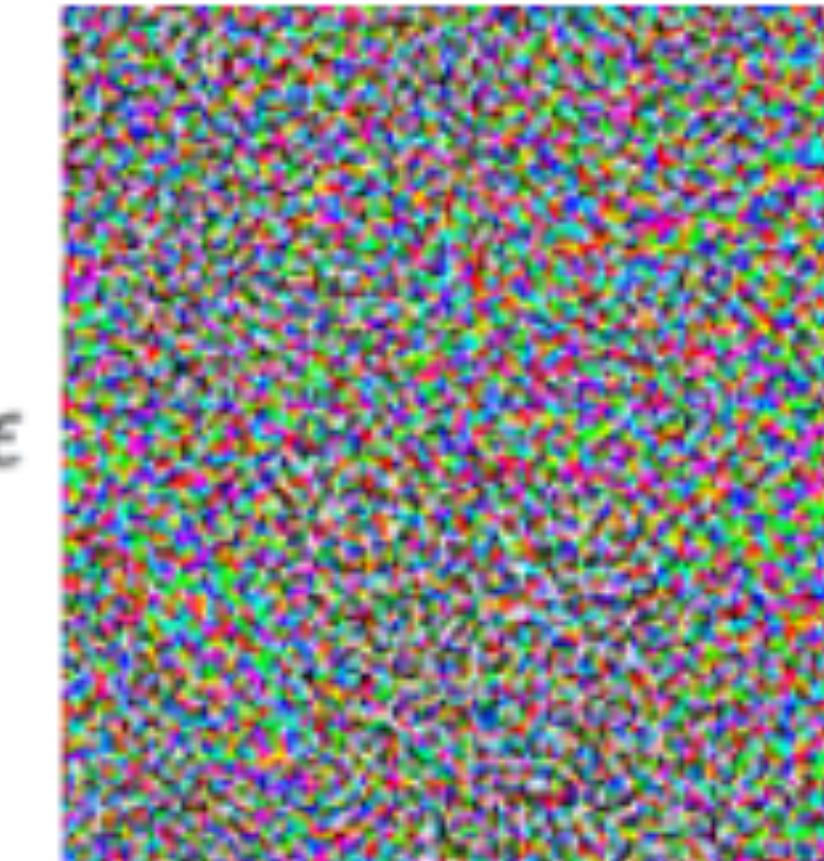
What is a Generative Adversarial Network (GAN)?

Intuition behind GAN

Original goal of GAN: Take as input training samples from some distribution and learn a model that represents that distribution.



$+ \epsilon$



=



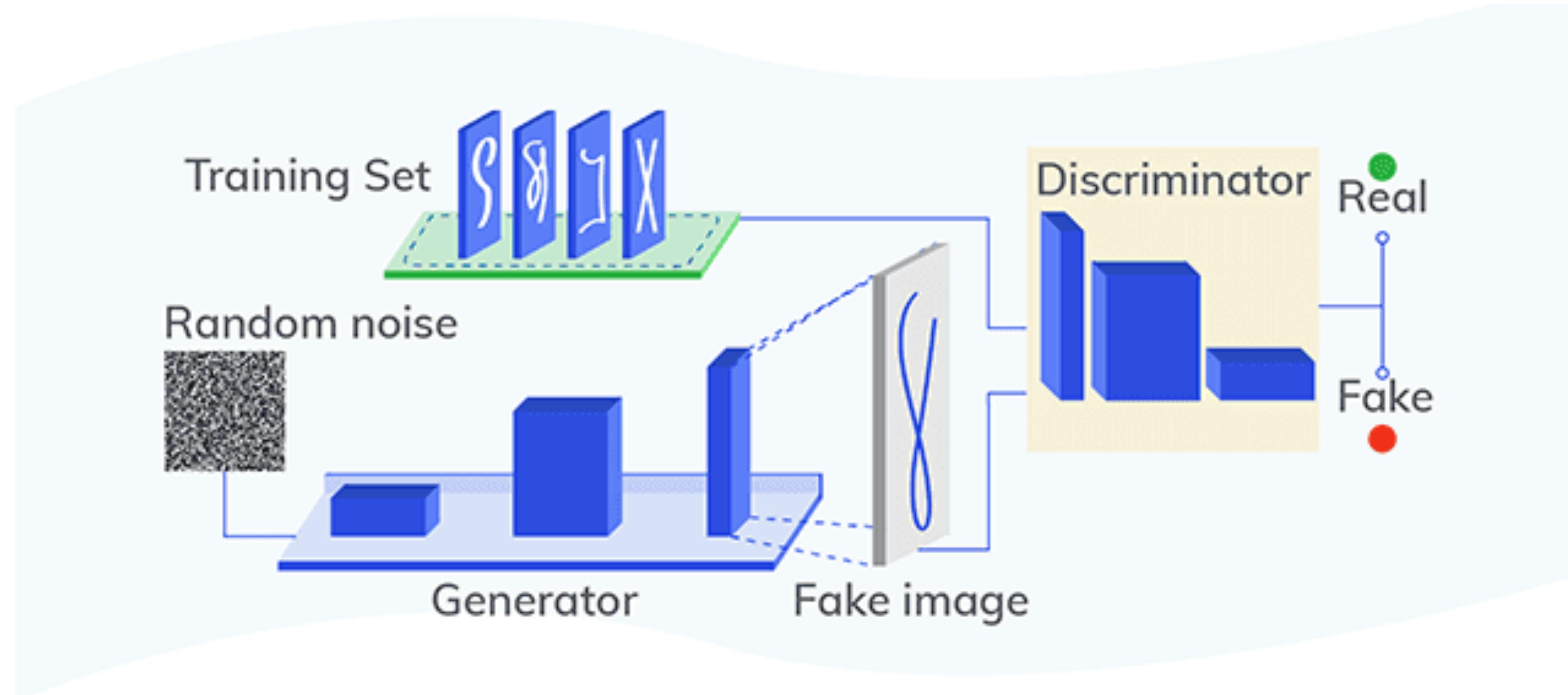
“panda”

57.7% confidence

“gibbon”

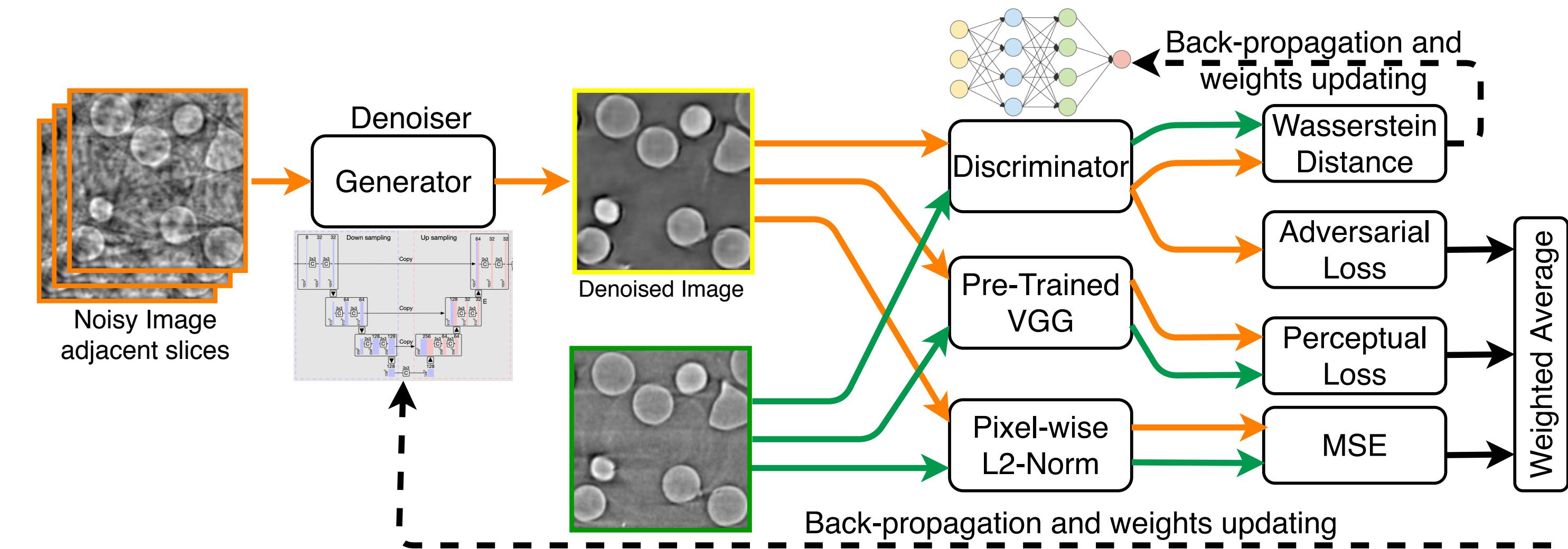
99.3% confidence

In general, a GAN consists of two CNNs, one is a Discriminator (D) and the other is a Generator (G). The Generator trains to generate images. The Discriminator tries to tell if the image is “real” or a fake.



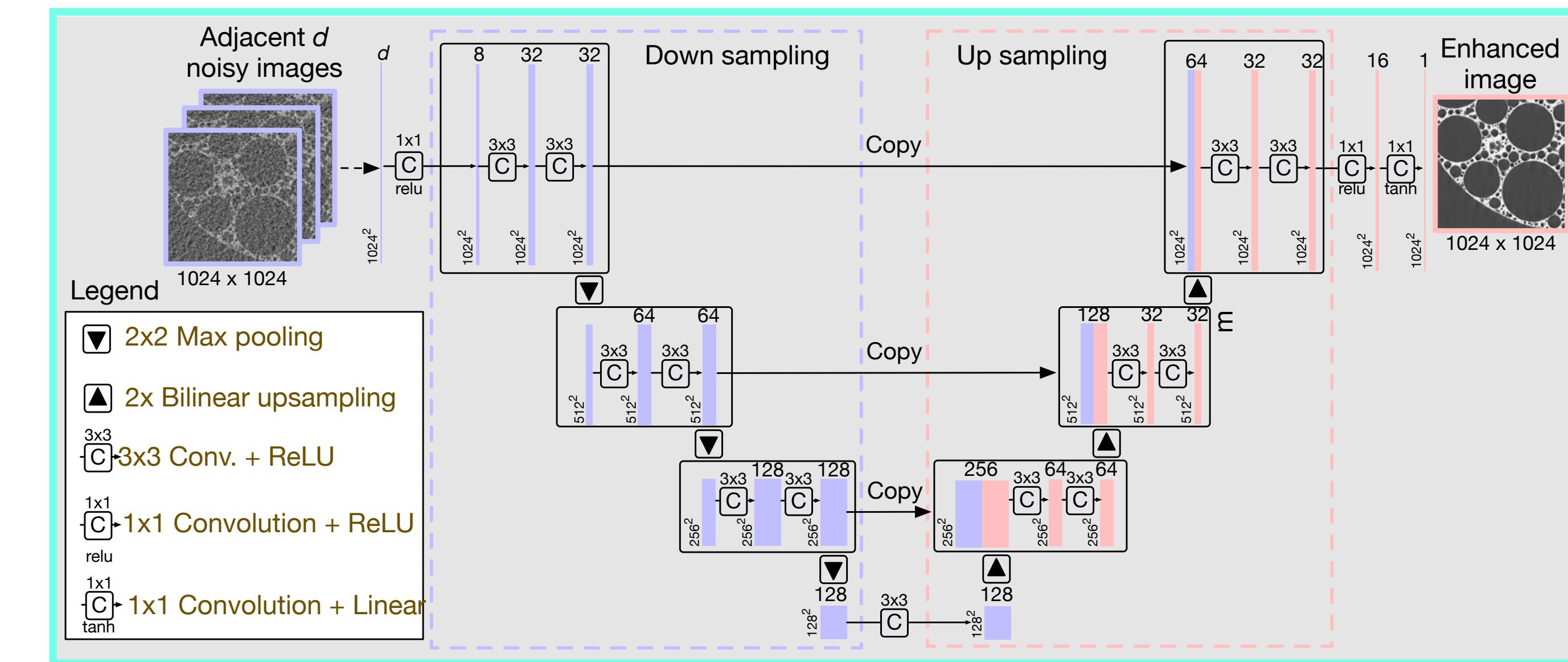
Method

We will use a GAN to denoise realistic images. The loss output is a little bit more complicated than in the previous slide as we compute several different kinds of error and back-propagate information to update weights.



In our model, the discriminator's job remains unchanged, but the generator is tasked not only with fooling (indistinguishable) the discriminator but also with being near the ground truth output in an L2 sense.

The discriminator works as a helper to train the generator that we need to denoise images.



Our Generator Architecture

Training uses several different loss functions

Discriminator Wasserstein GAN

$$L(\theta_D) = \frac{1}{m} \sum_{i=1}^m \left[D\left(G(I_{LD}^i)\right) - D\left(I_{ND}^i\right) \right]$$

Generator Weighted average of Adversarial loss, Perceptual loss, and Pixel-wise MSE

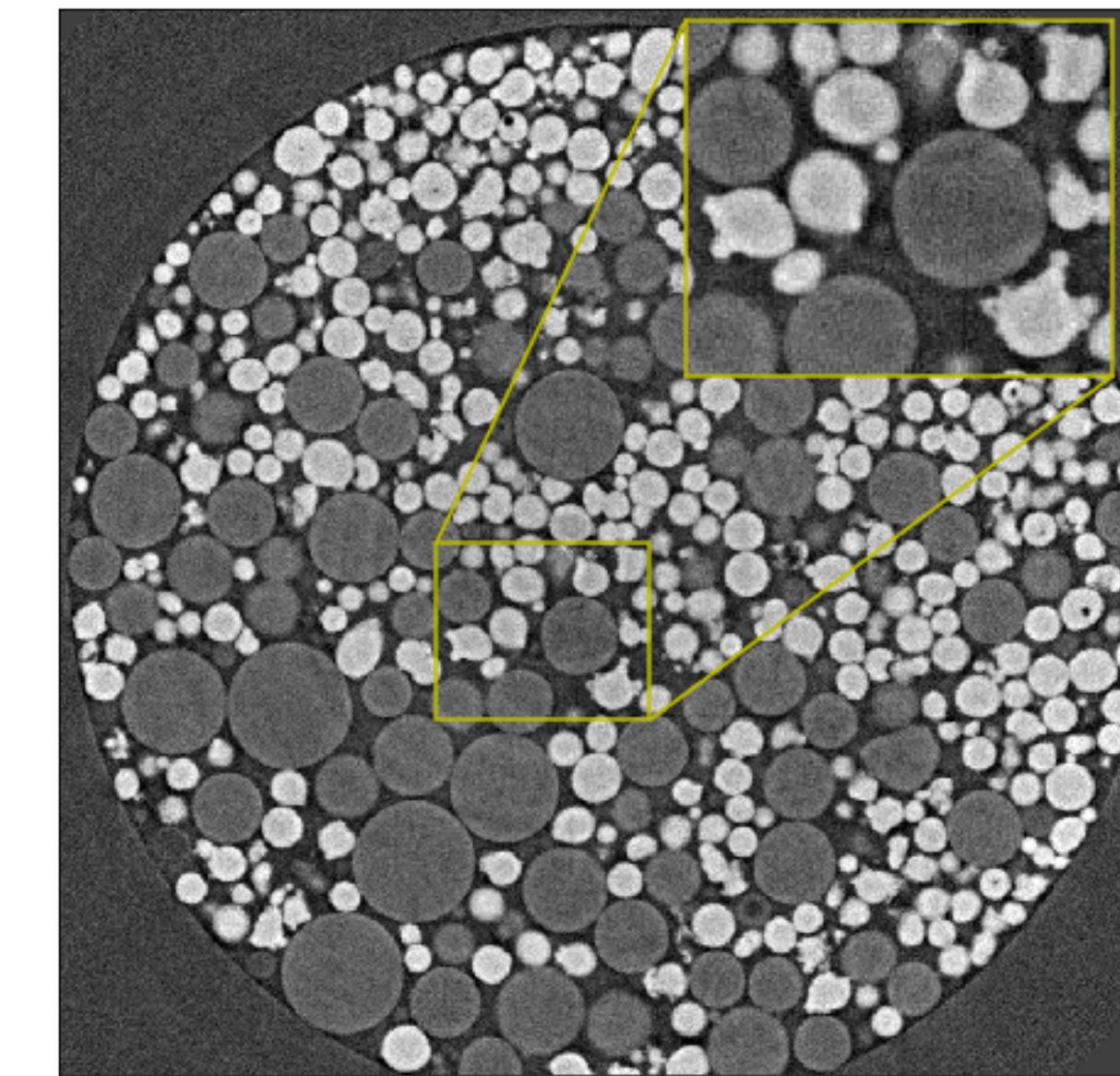
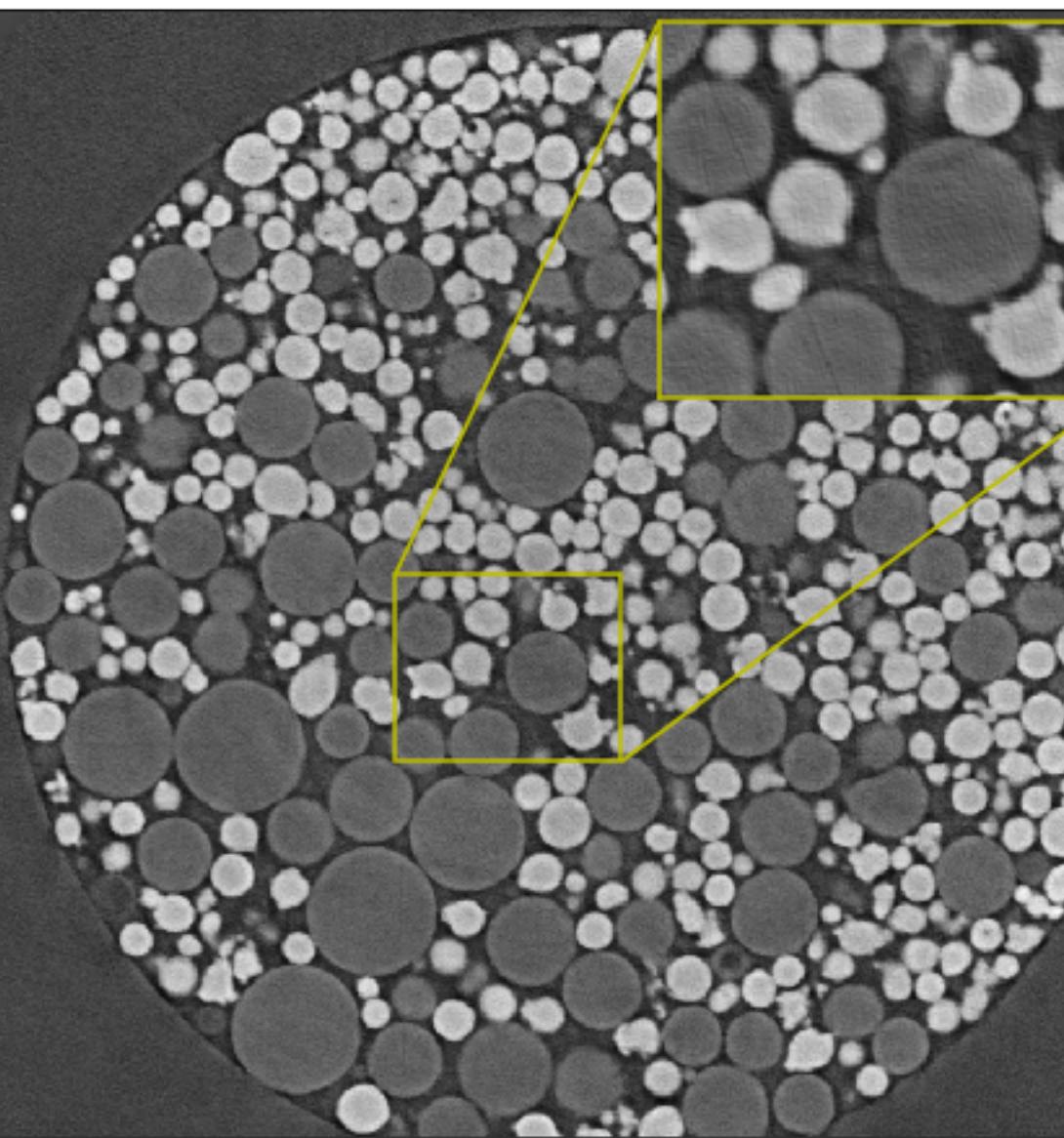
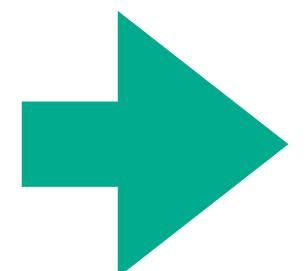
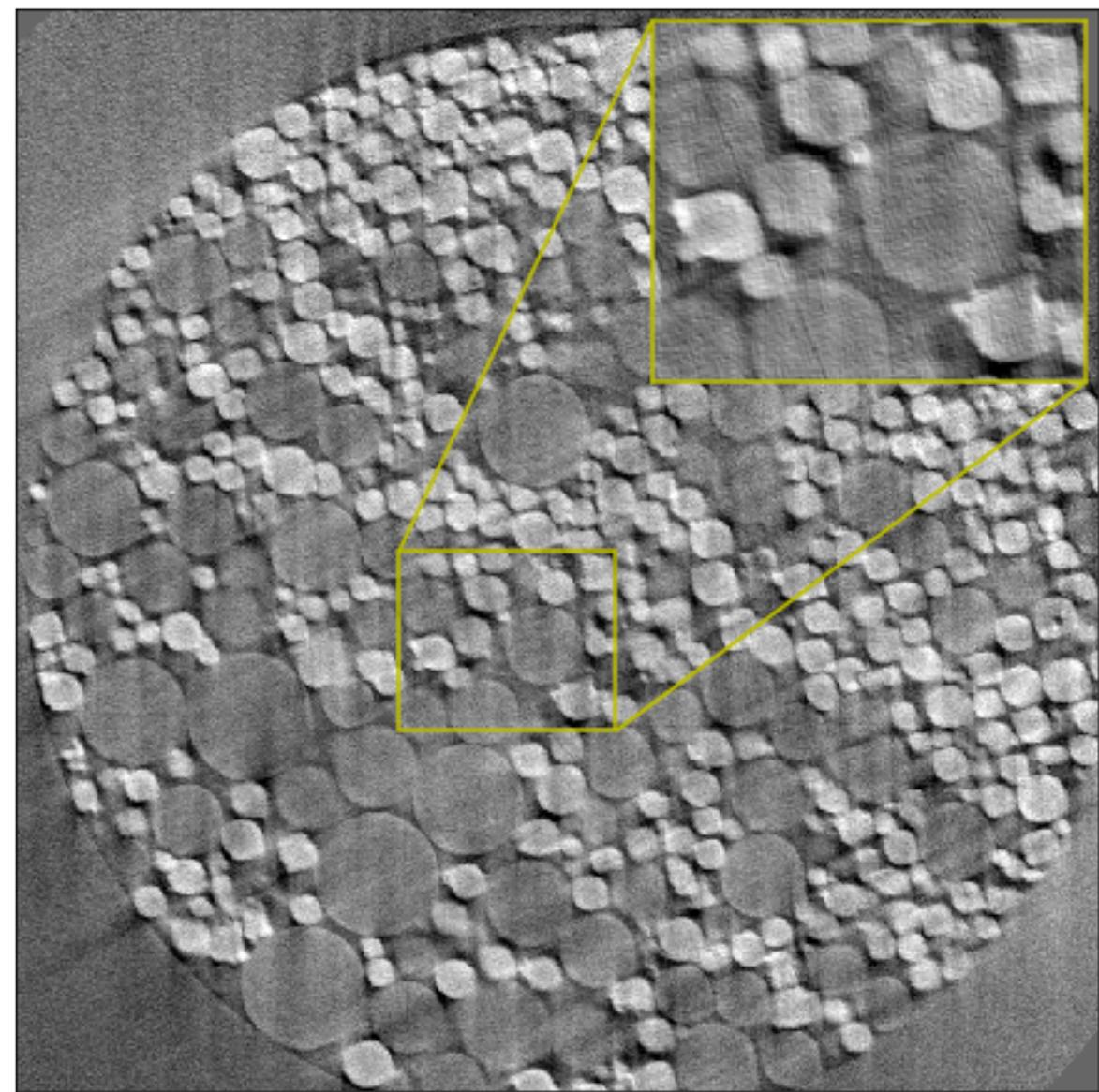
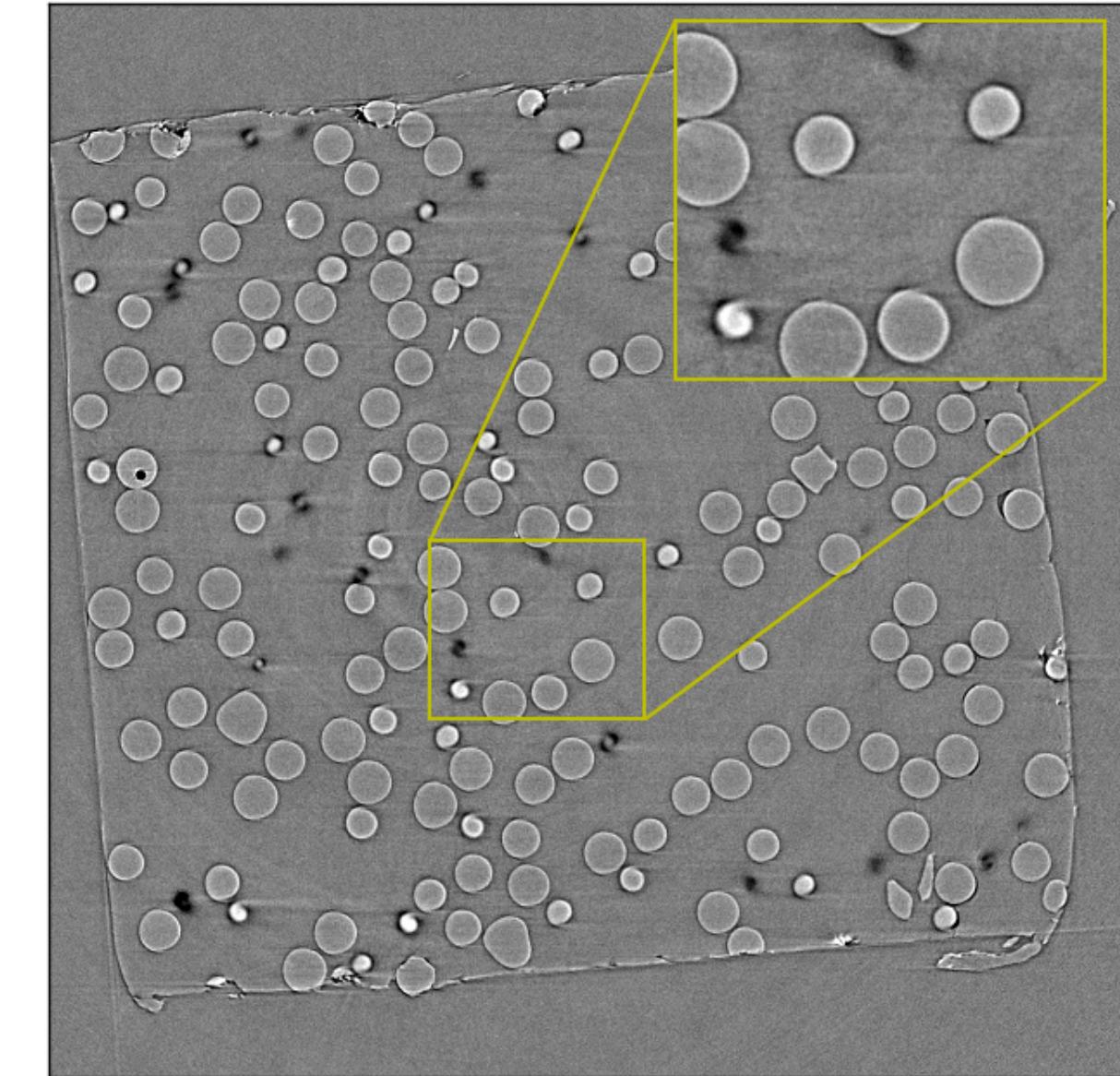
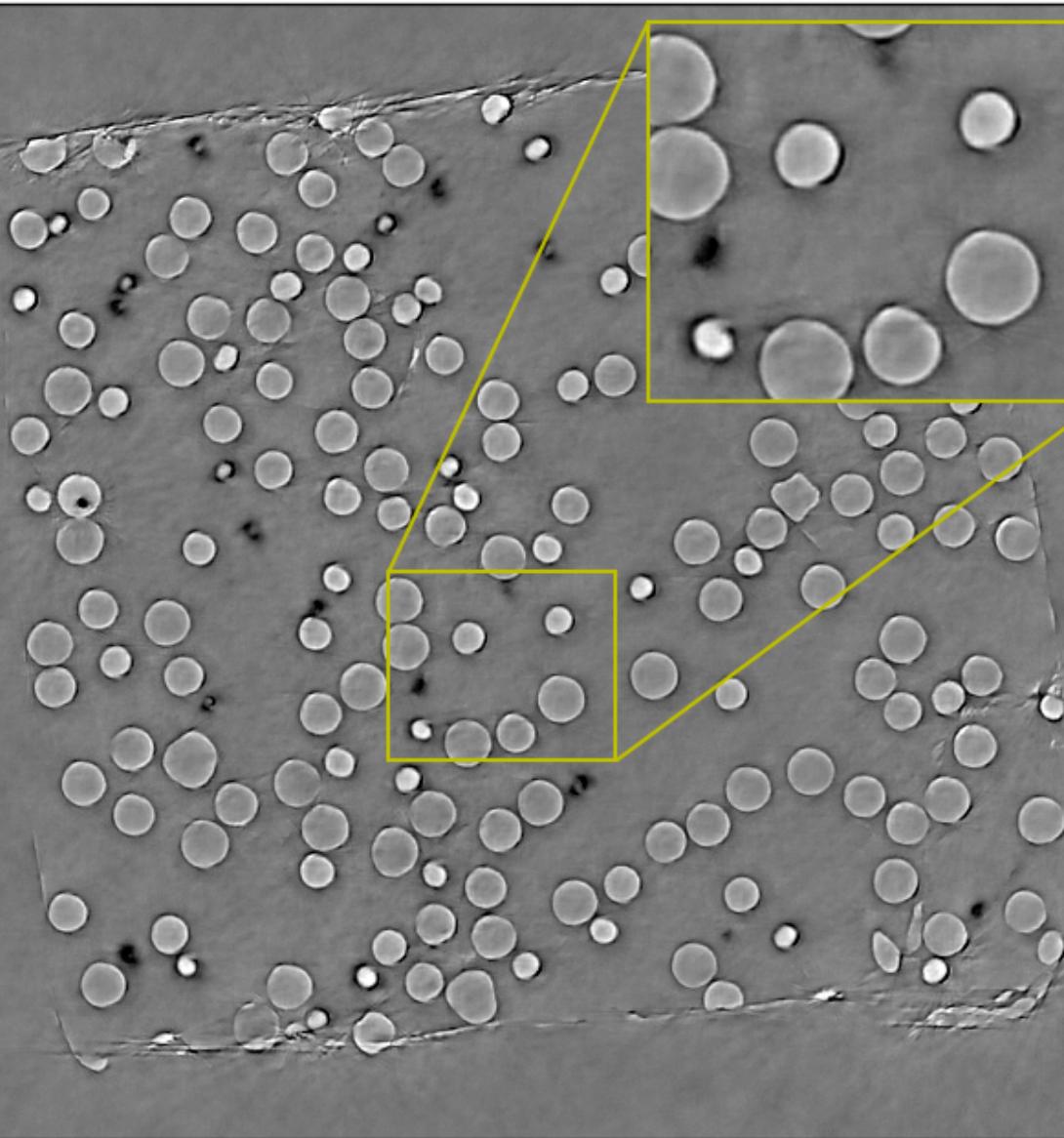
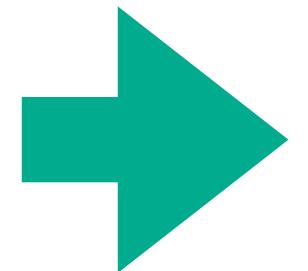
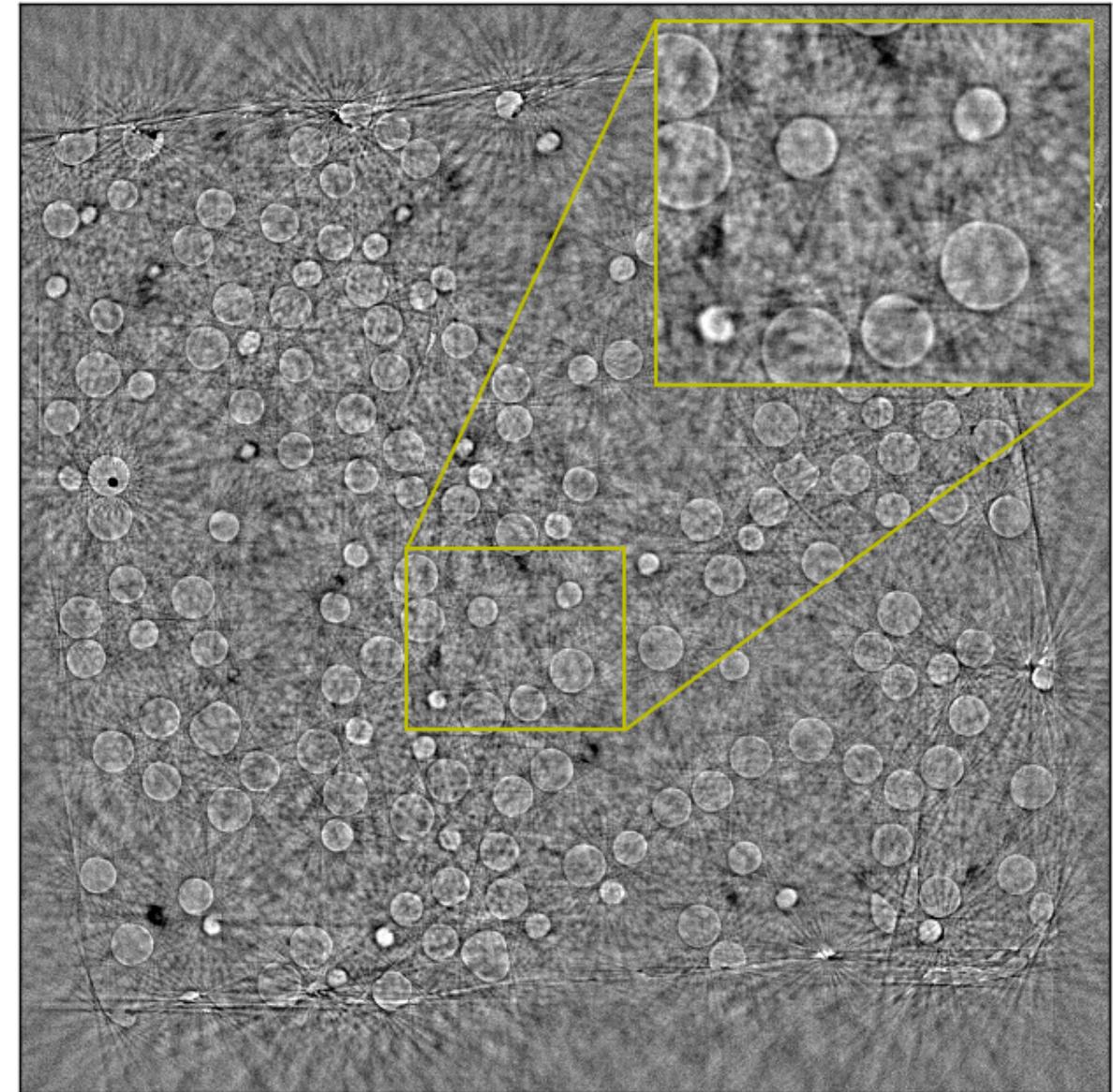
$$\ell^G = \lambda_g \ell_{adv} + \lambda_p \ell_{mse} + \lambda_v \ell_{vgg}$$

$$\ell_{adv}(\theta_G) = -\frac{1}{m} \sum_{i=1}^m D\left(G(I_{LD}^i)\right)$$

$$\ell_{vgg} = \sum_{i=1}^{W_f} \sum_{j=1}^{H_f} \left(V_{\theta_{vgg}}(I^{ND})_{i,j} - V_{\theta_{vgg}}\left(G_{\theta_G}(I^{LD})\right)_{i,j} \right)^2$$

$$\ell_{mse} = \sum_{c=1}^W \sum_{r=1}^H \left(I_{c,r}^{ND} - G_{\theta_G}(I^{LD})_{c,r} \right)^2$$

TomoGAN: noise and artifacts removal



Best practice (unnecessary in most cases)

- ❑ Unlike RGB camera images that only holds 0-255, experimental data may have outliers. Remove outliers (e.g., bad pixel) from your data as they may skew the MSE.
- ❑ Hyper-parameters, weights of losses: -ladv, -lmse, -lperc ideally all have similar scale after weighting
- ❑ Input patch size, 256 - 512

Switch to demo on training with a given dataset - TomoGAN-prod jupyter notebook

AI-based denoise of scientific images

**What if there is no clear image (ground truth) to train the model?
This is often the case in scientific applications.**

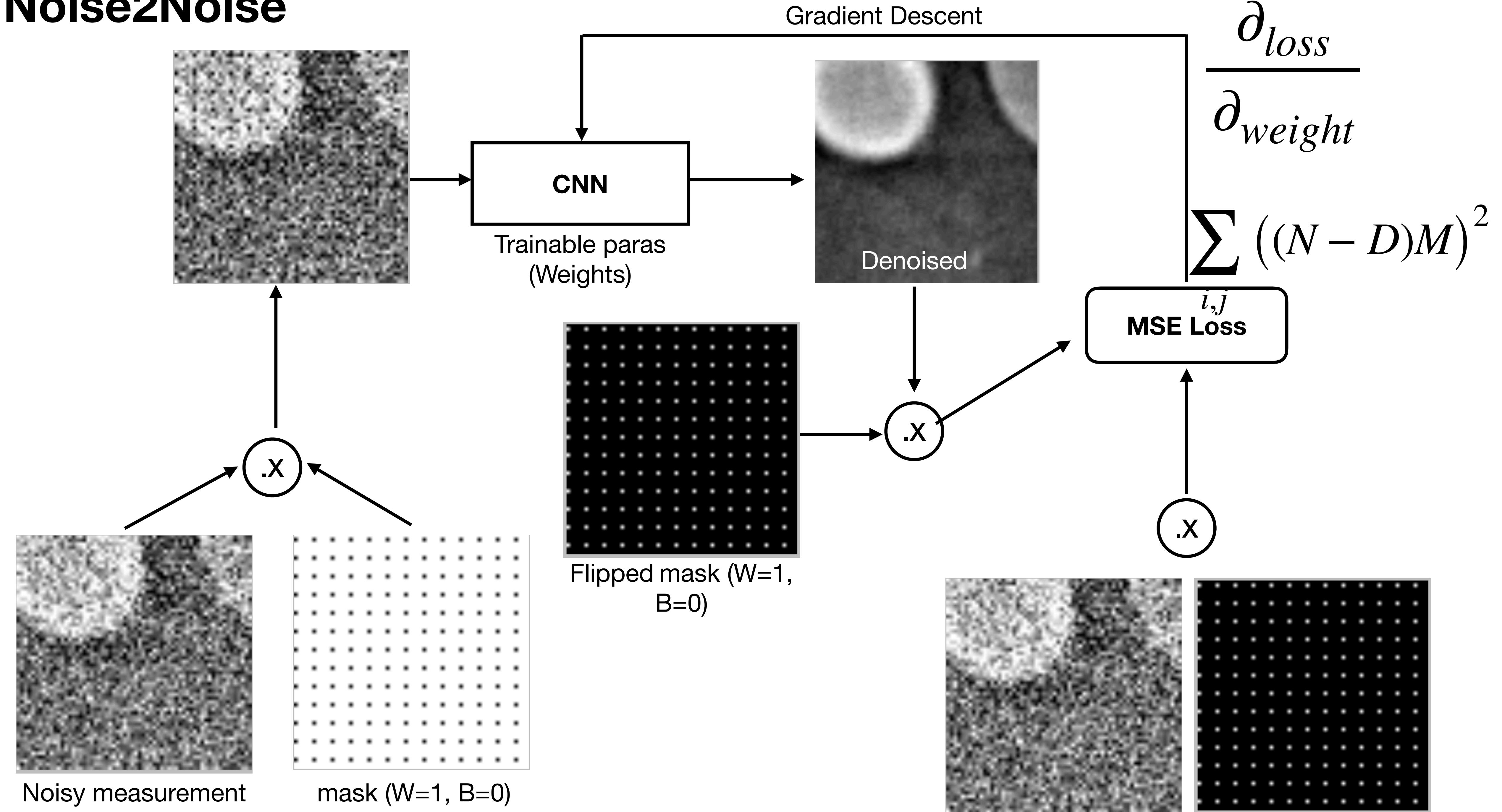
In this case, one can either (i) use supervised training on some other data set and apply the trained model to the “real” data (and example of so-called transfer learning), or (ii) use unsupervised learning in which the model has to teach itself.

We will explore the latter here.

We have to make some fundamental assumptions about the noise. First, we assume

- (i) the noise is element-wise statistically independent and has zero mean (not artifact), and
- (ii) the images exhibit some spatial correlation.

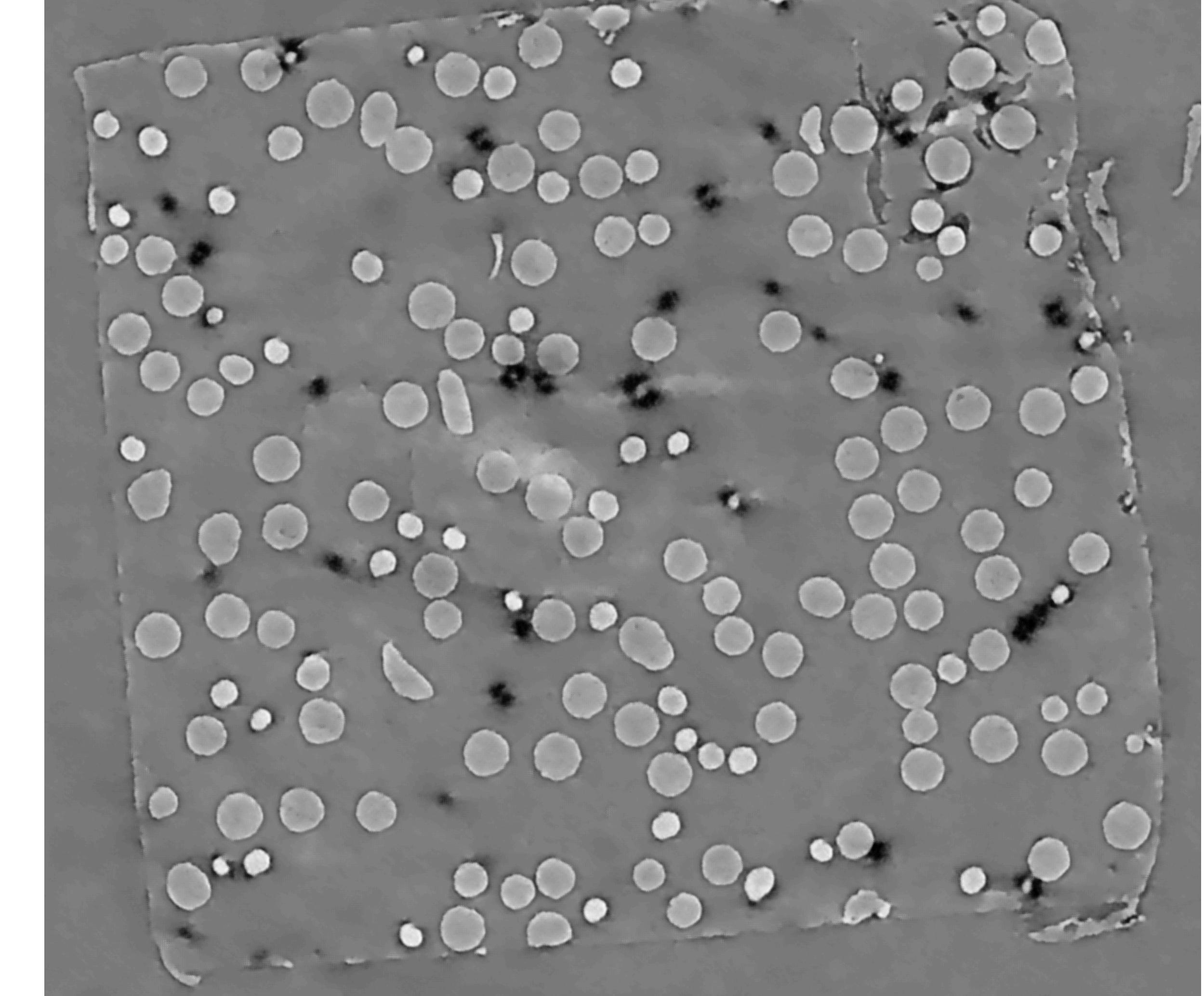
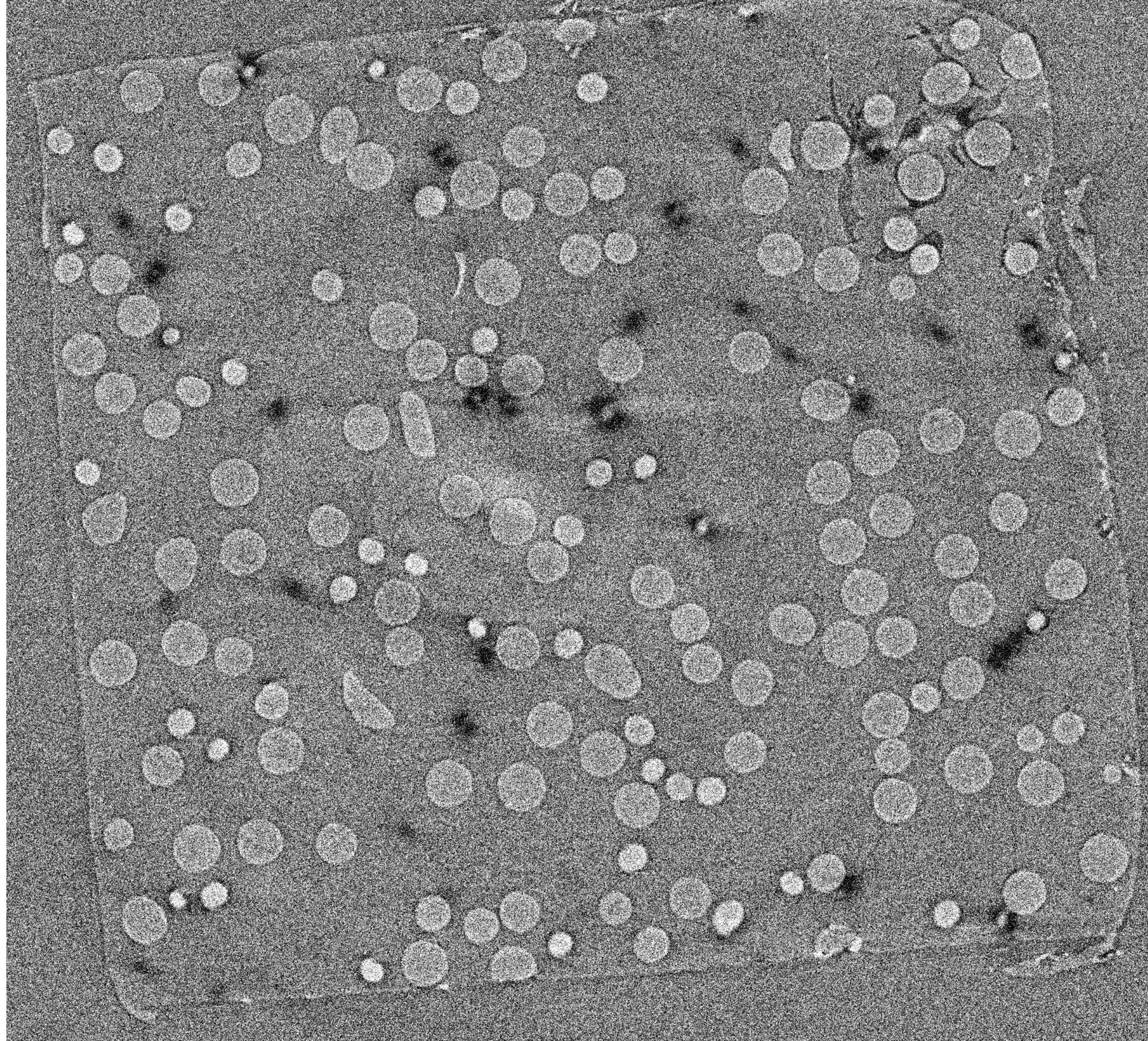
Noise2Noise



Noise2Self

Self-supervised learning.

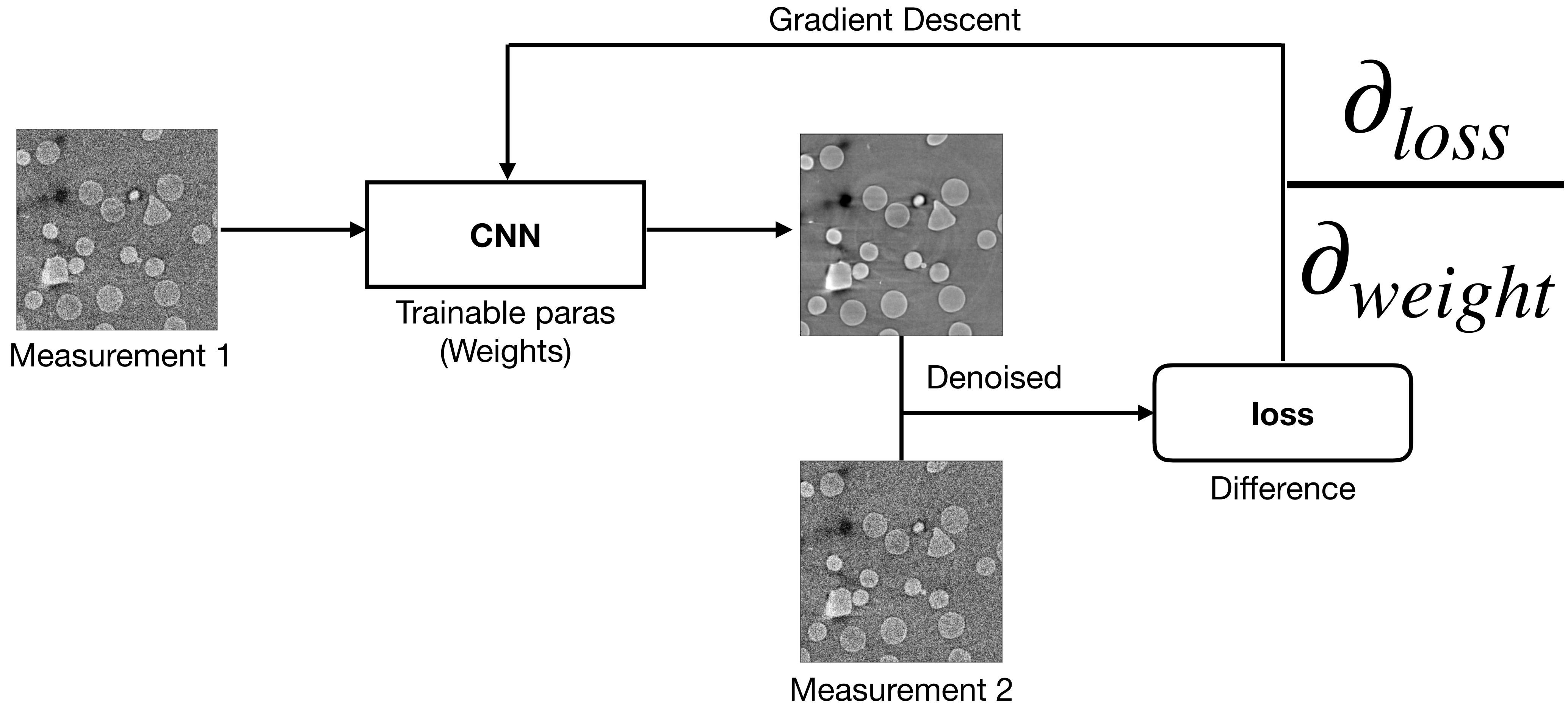
noise is assumed element-wise statistically independent and mean-zero, and that the clean images exhibit some spatial correlation.



In another approach with no clean images,
one can measure **independent** instances of the noise several times to gather
information about the noise.

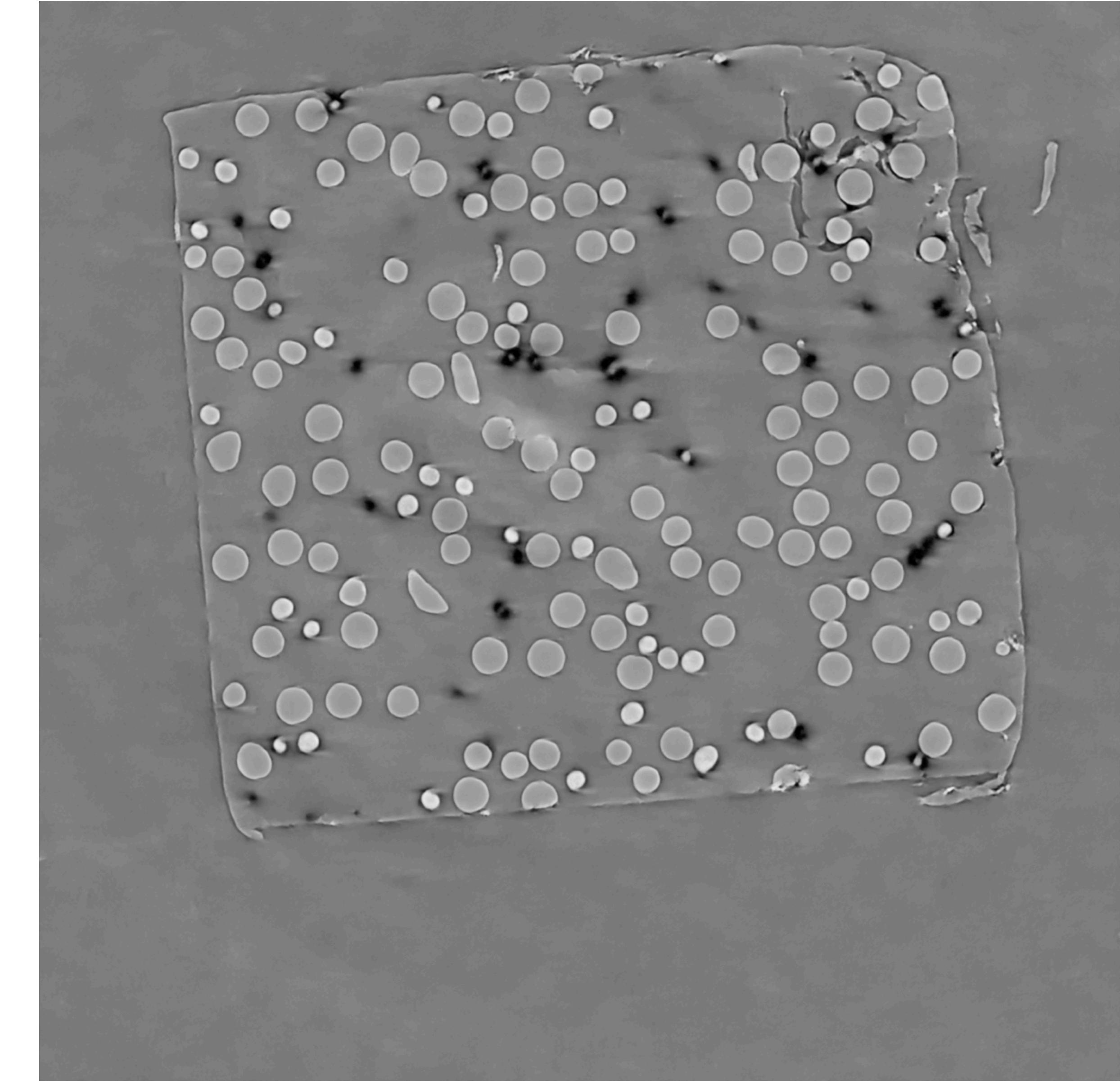
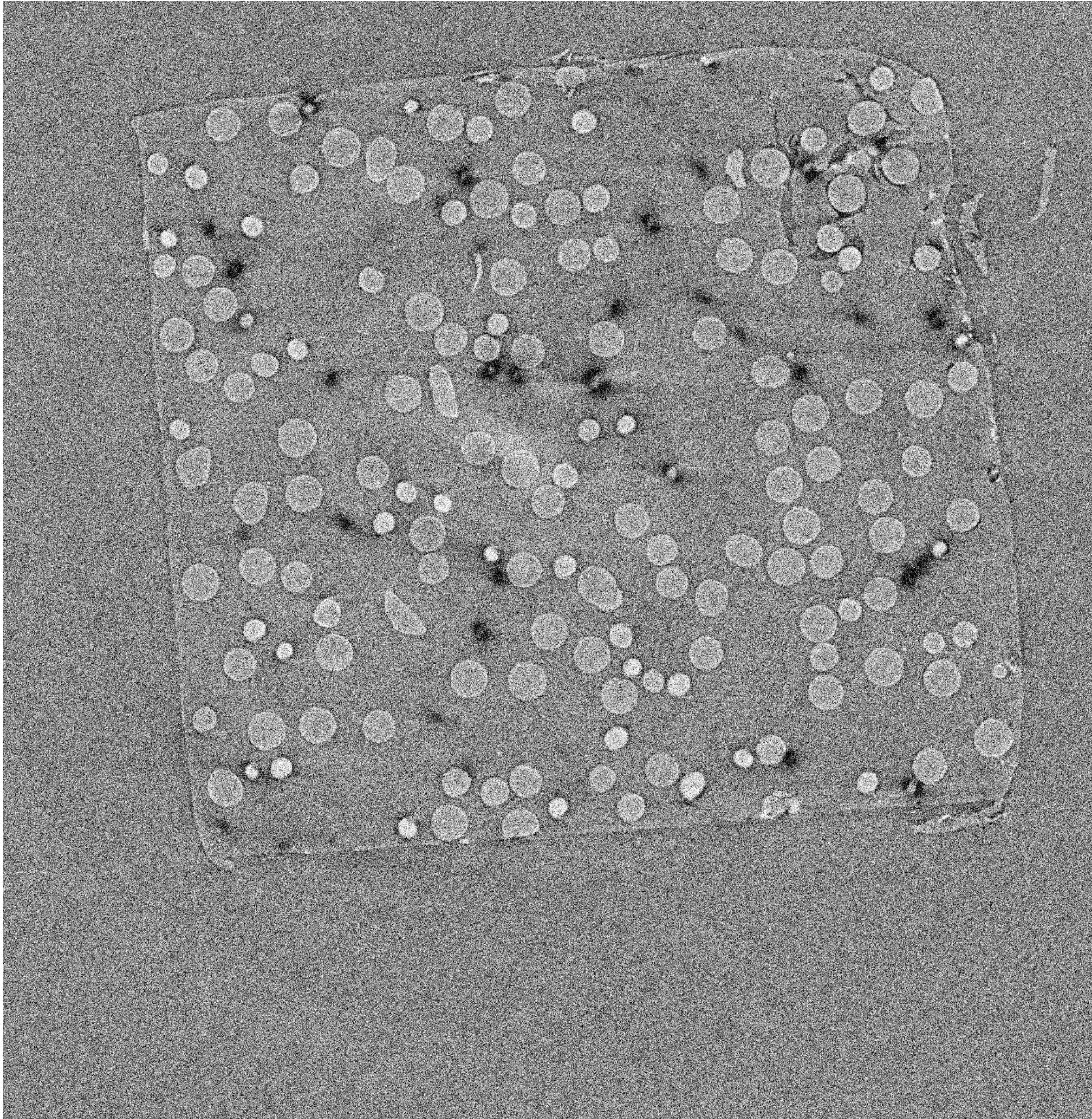
Noise2Noise

no clean images are needed, but one can measure **independent** instances of the noise for each image.

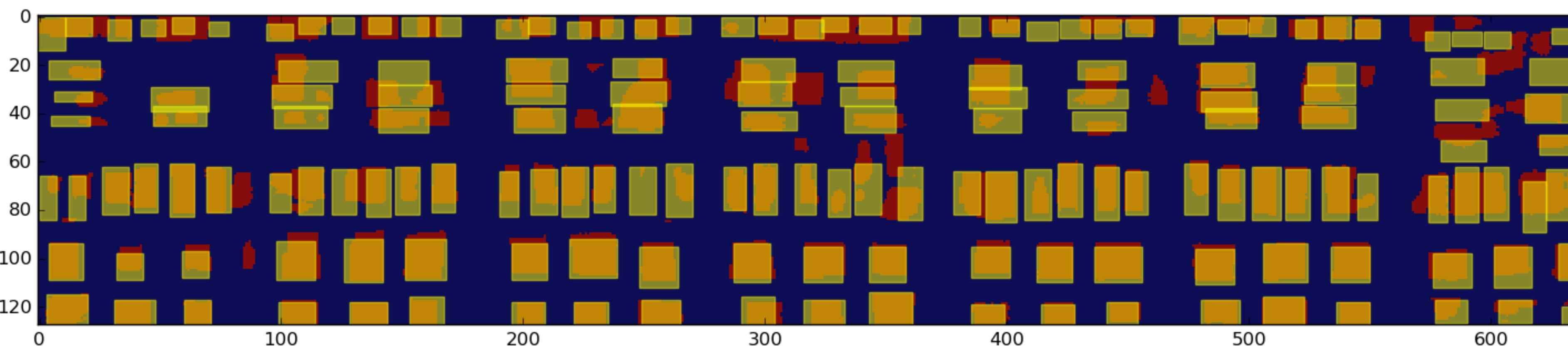
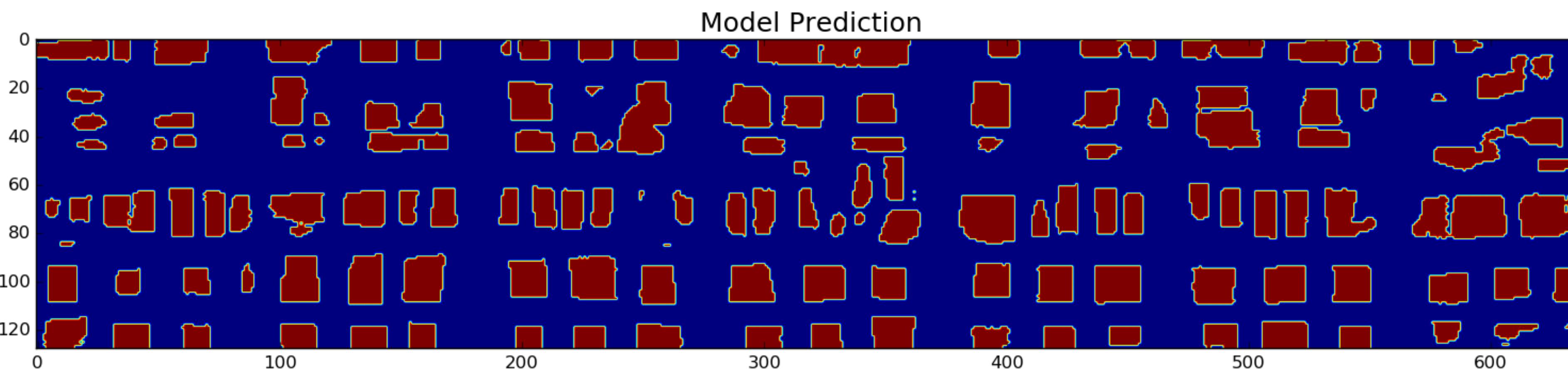
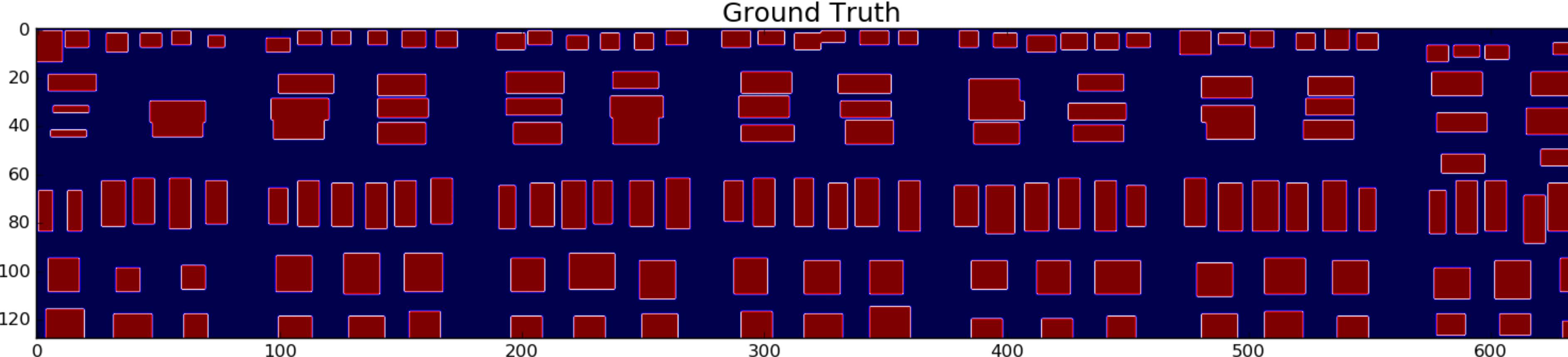


Noise2Noise

no clean images are needed, but one can measure independent instances of the noise for each image.



Also works for Segmentation, only need to change loss function



Thanks!

Want to try?

Open source at: <https://github.com/ramsesproject/TomoGAN>

python: Tensorflow and Keras based;

C++ : DNNL(MKL-DNN) based, good for CPU based e.g., KNL;

C++, CUDA: cuDNN and cuda based, good for NVIDIA GPU;

Pytorch: upon request