Deep learning

7.3. Denoising autoencoders

François Fleuret

https://fleuret.org/dlc/

UNIVERSITÉ
DE GENÈVE

Beside dimension reduction, autoencoders can capture dependencies between signal components to restore a degraded input.

In that case, we can ignore the encoder/decoder structure, and such a model

$$\phi : \mathcal{X} \to \mathcal{X}.$$

is referred to as a **denoising** autoencoder.

Beside dimension reduction, autoencoders can capture dependencies between signal components to restore a degraded input.

In that case, we can ignore the encoder/decoder structure, and such a model

$$\phi : \mathcal{X} \to \mathcal{X}.$$

is referred to as a **denoising** autoencoder.

The goal is not anymore to optimize ϕ so that

$$\phi(X) \simeq X$$

but, given a perturbation $\tilde{X}$ of the signal X, to restore the signal, hence

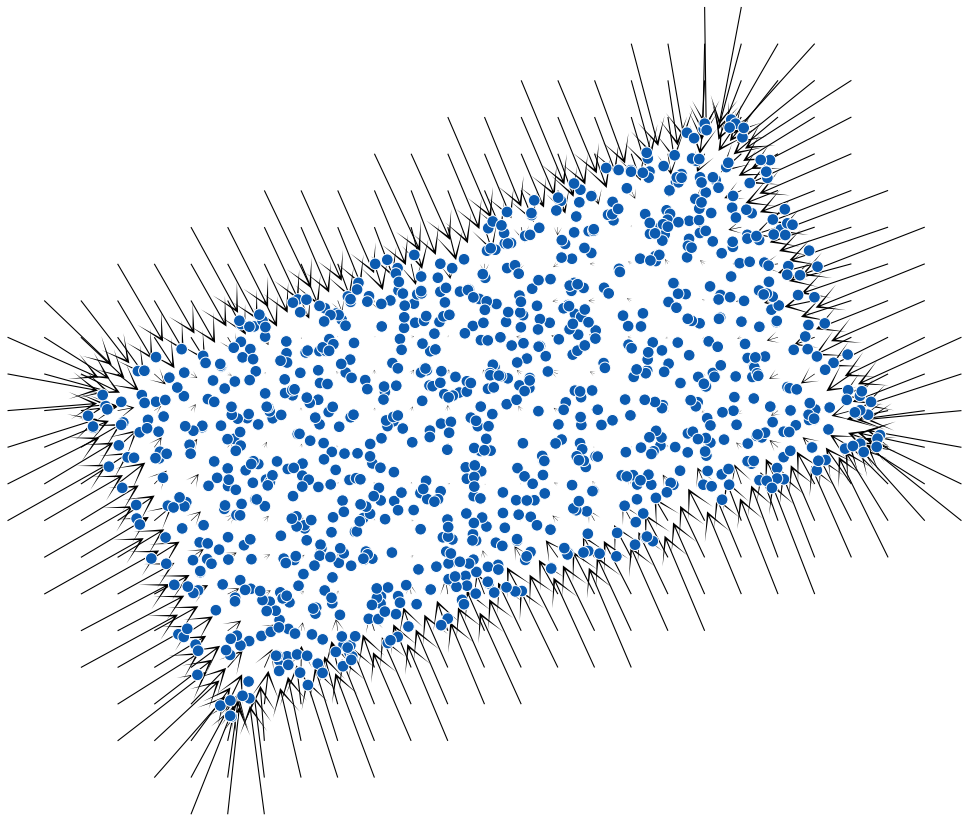
$$\phi(\tilde{X}) \simeq X.$$

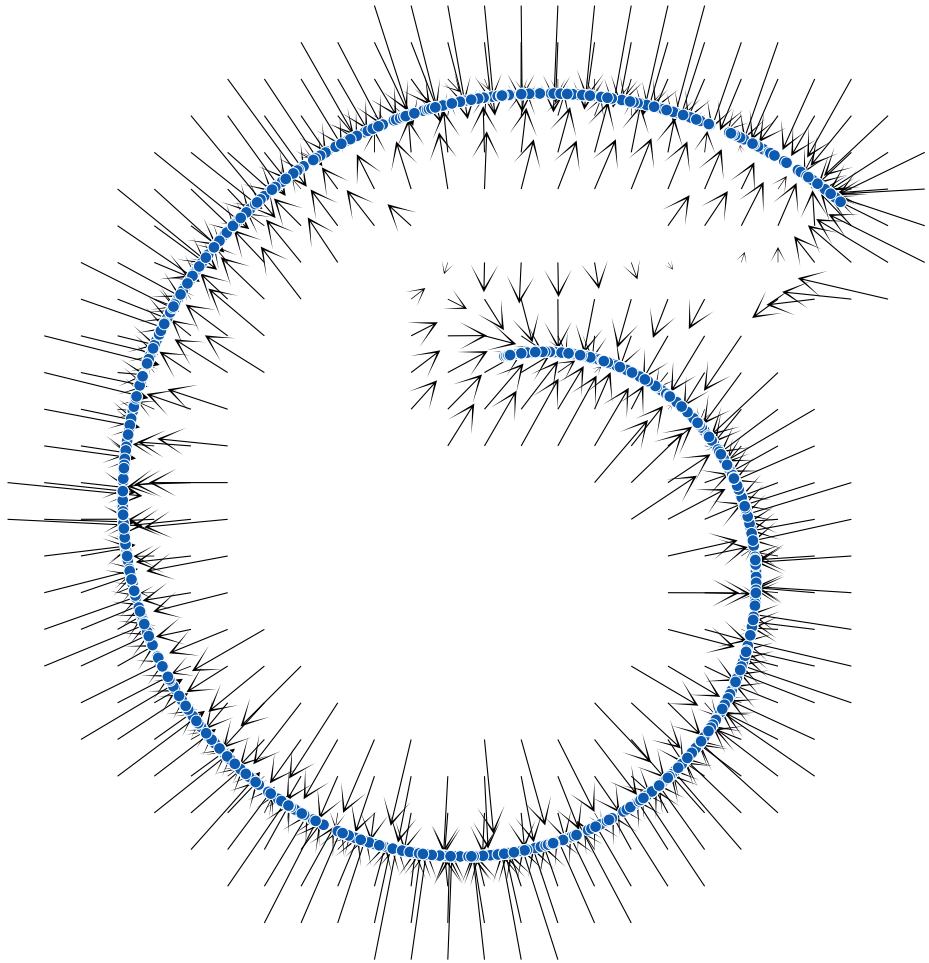
We can illustrate this notion in $2d$ with an additive Gaussian noise, and the quadratic loss, hence

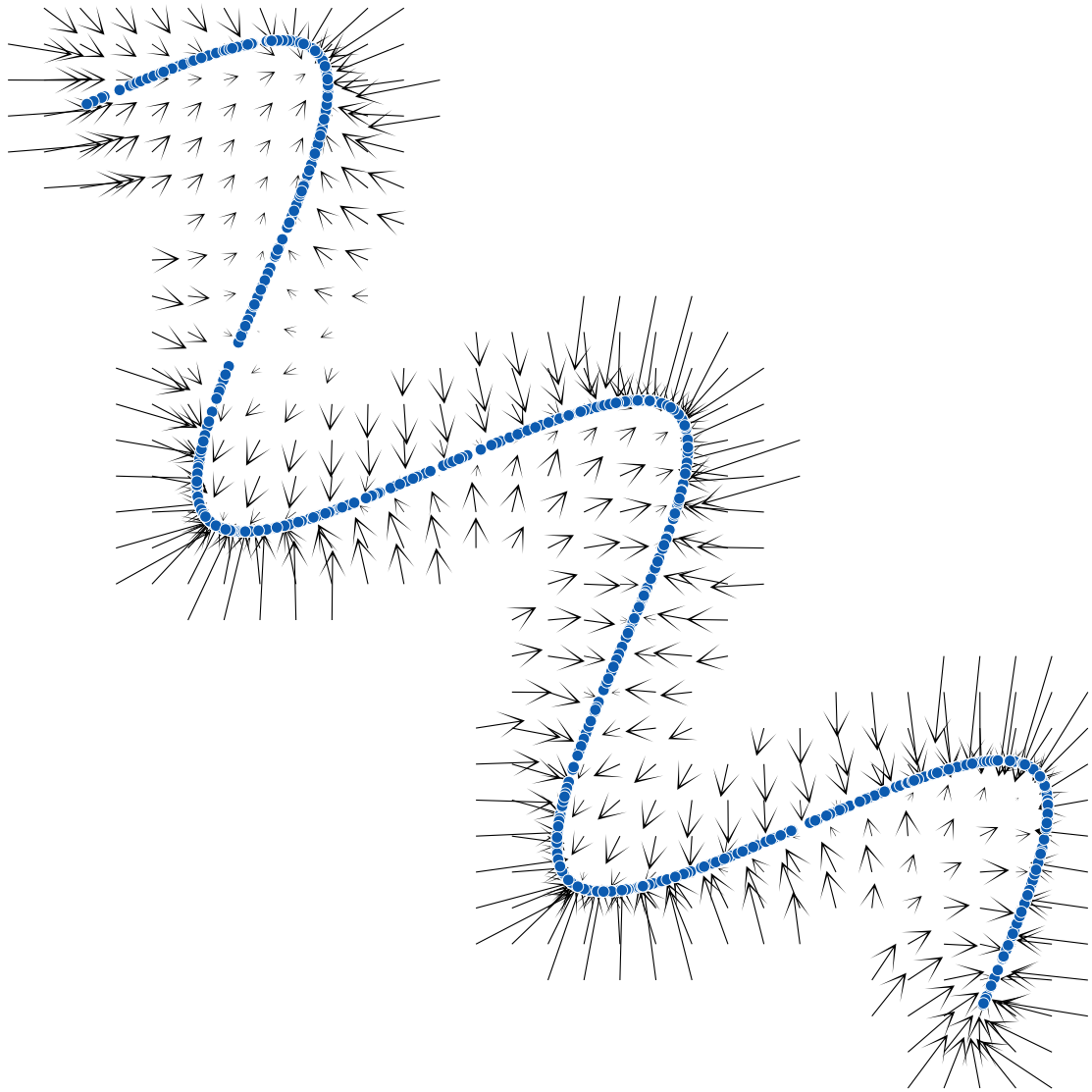
$$\hat{w} = \operatorname*{argmin}_{w} \frac{1}{N} \sum_{n=1}^{N} \|x_n - \phi(x_n + \epsilon_n; w)\|^2 ,$$

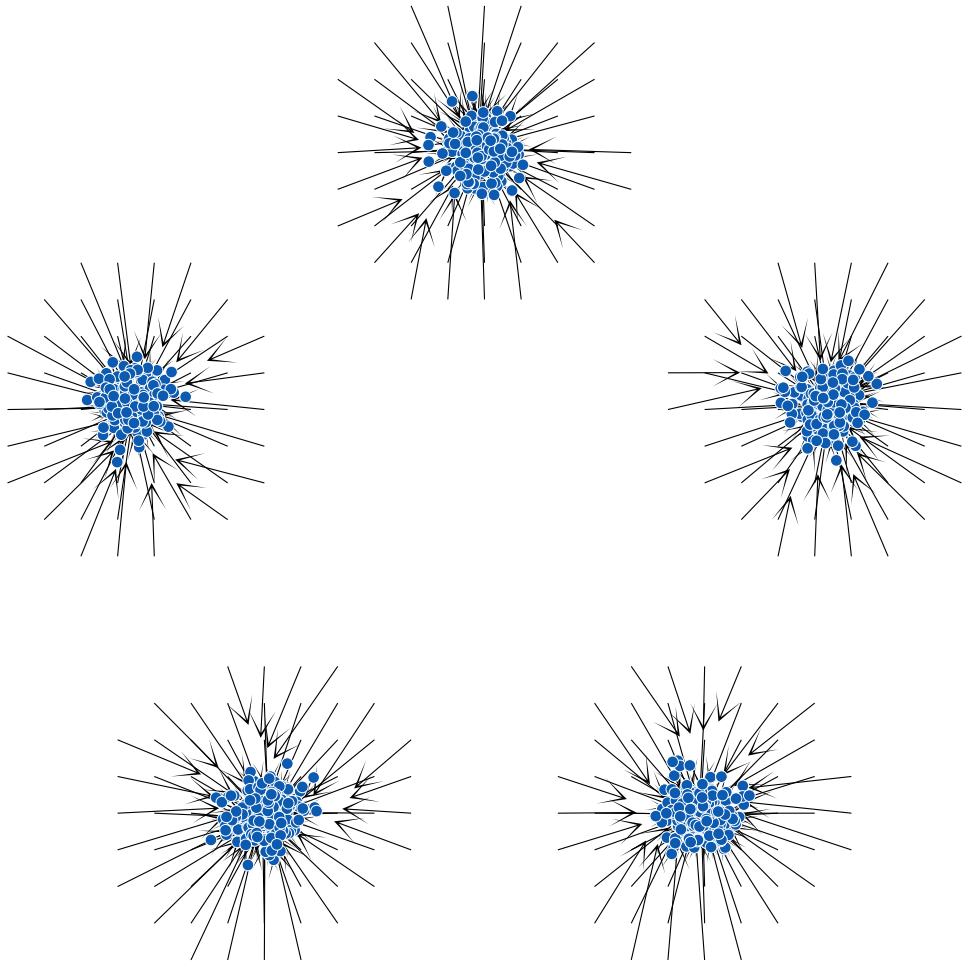
where x_n are the data samples, and ϵ_n are Gaussian random noise vectors.

```
model = nn.Sequential(
    nn.Linear(2, 100),
    nn.ReLU(),
    nn.Linear(100, 2)
)

batch_size, nb_epochs = 100, 1000
optimizer = torch.optim.Adam(model.parameters(), lr = 1e-3)
mse = nn.MSELoss()

for e in range(nb_epochs):
    for input in data.split(batch_size):
        noise = input.new(input.size()).normal_(0, 0.1)
        output = model(input + noise)
        loss = mse(output, input)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```









We can do the same on MNIST, for which we keep our deep autoencoder, and ignore its encoder/decoder structure.

```
corrupted_input = corruptor.corrupt(input)

output = model(corrupted_input)
loss = 0.5 * (output - input).pow(2).sum() / input.size(0)

optimizer.zero_grad()
loss.backward()
optimizer.step()
```

We can do the same on MNIST, for which we keep our deep autoencoder, and ignore its encoder/decoder structure.

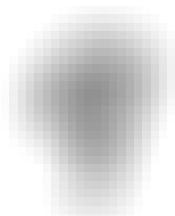
```
corrupted_input = corruptor.corrupt(input)

output = model(corrupted_input)
loss = 0.5 * (output - input).pow(2).sum() / input.size(0)

optimizer.zero_grad()
loss.backward()
optimizer.step()
```

We consider three types of corruptions, that go beyond additive noise:

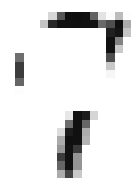
Original

Pixel erasure

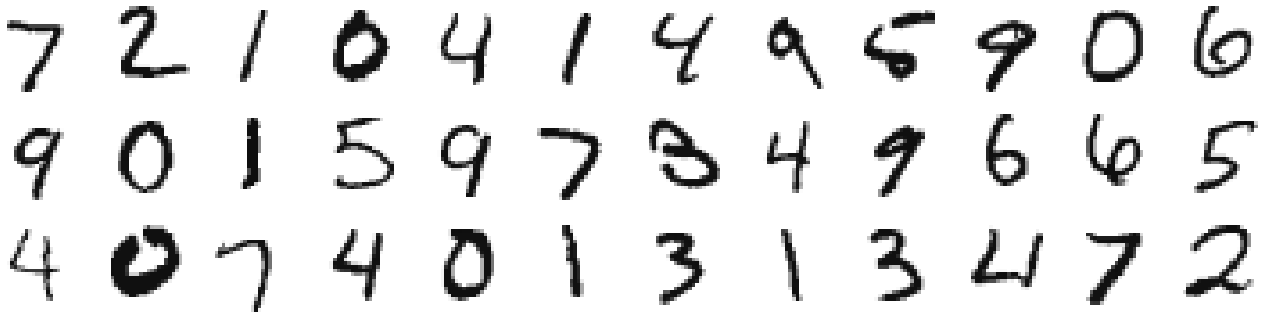
Blurring



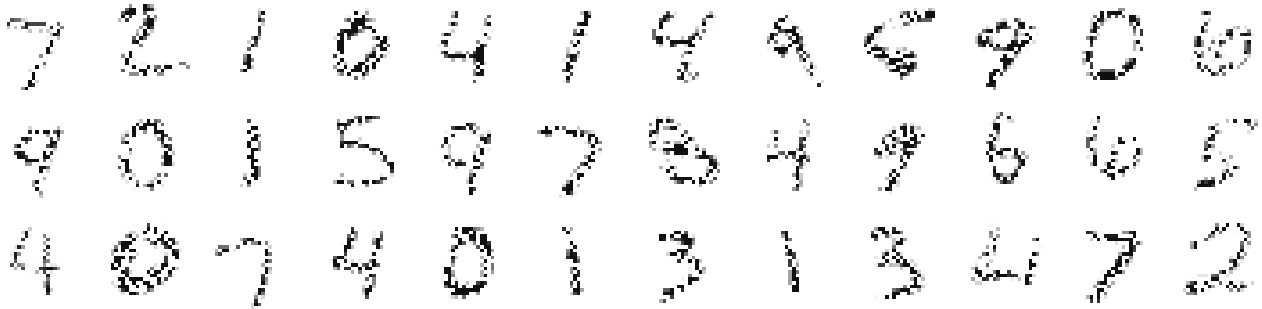
Block masking



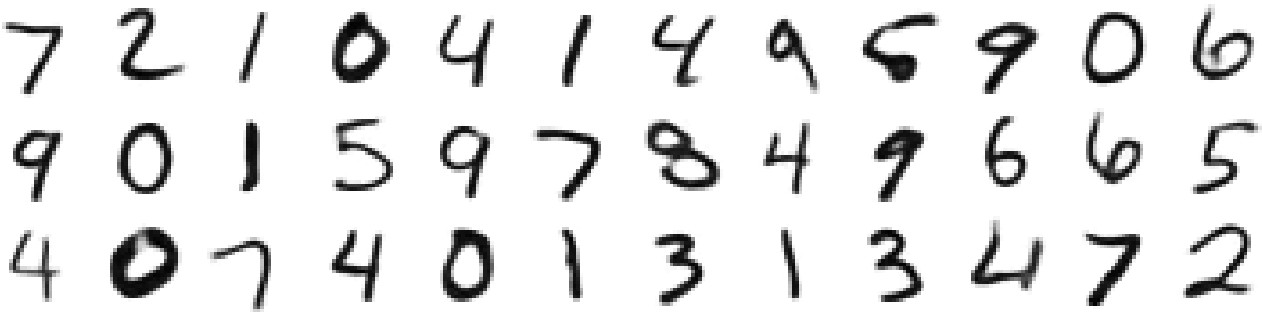
Original



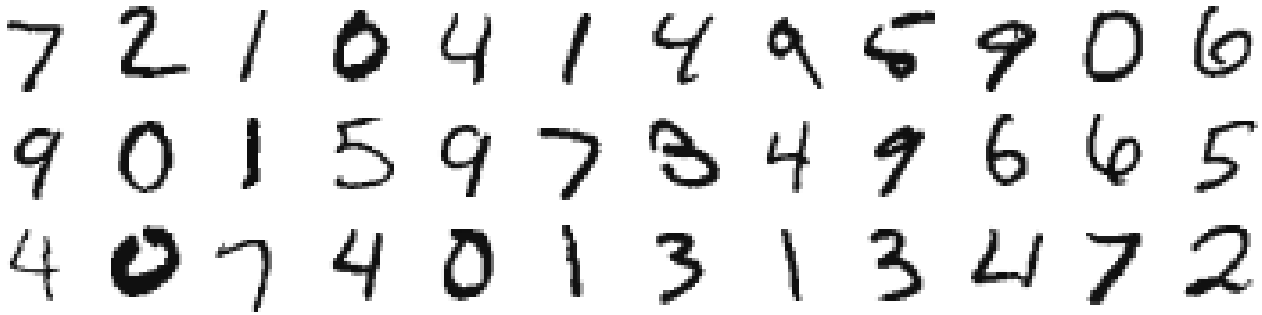
Corrupted ($p = 0.5$)



Reconstructed

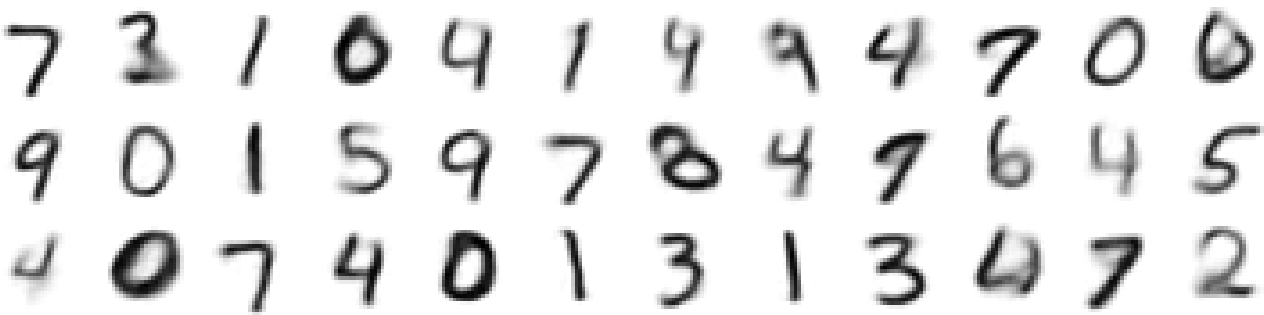


Original

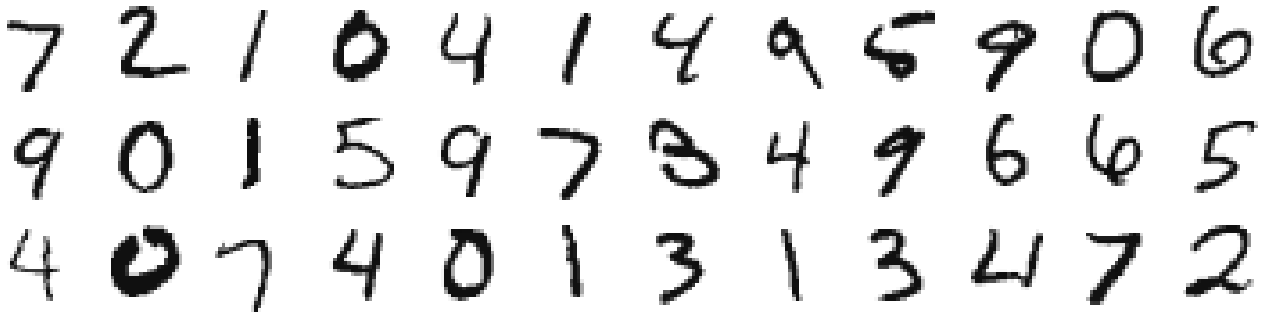


Corrupted ($p = 0.9$)

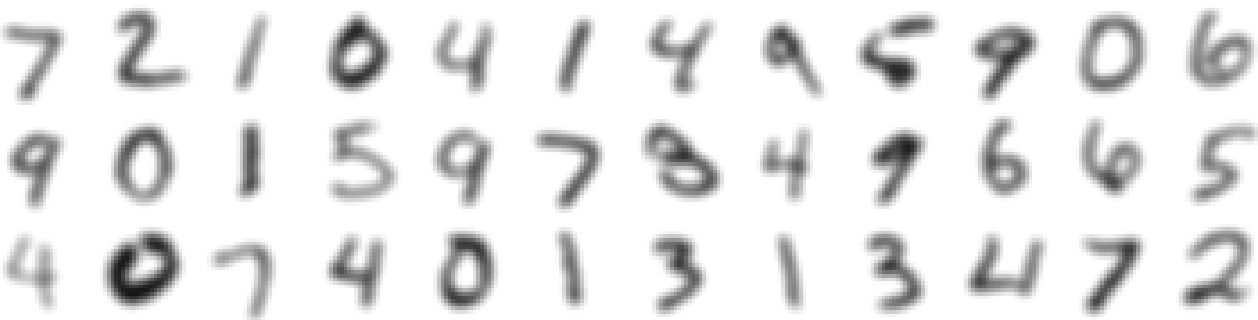
Reconstructed



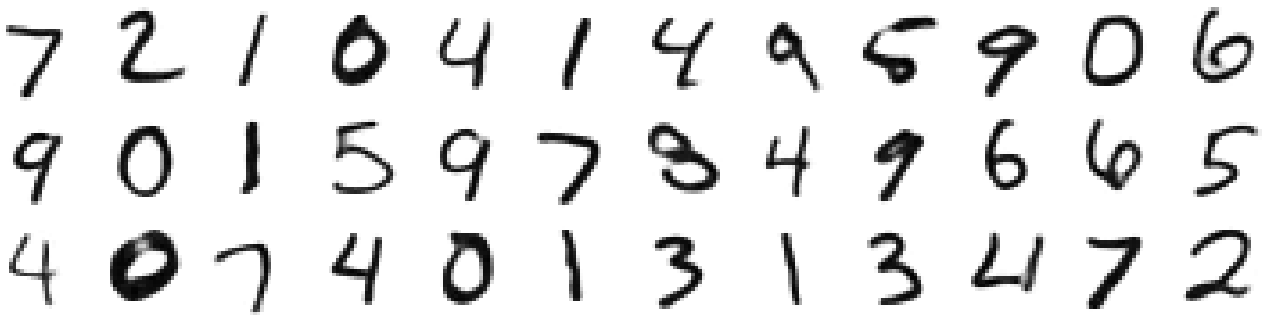
Original



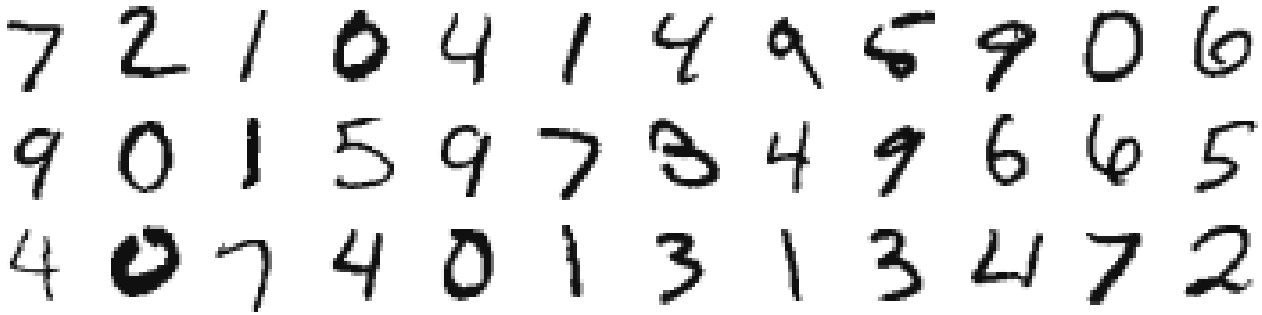
Corrupted ($\sigma = 2$)



Reconstructed

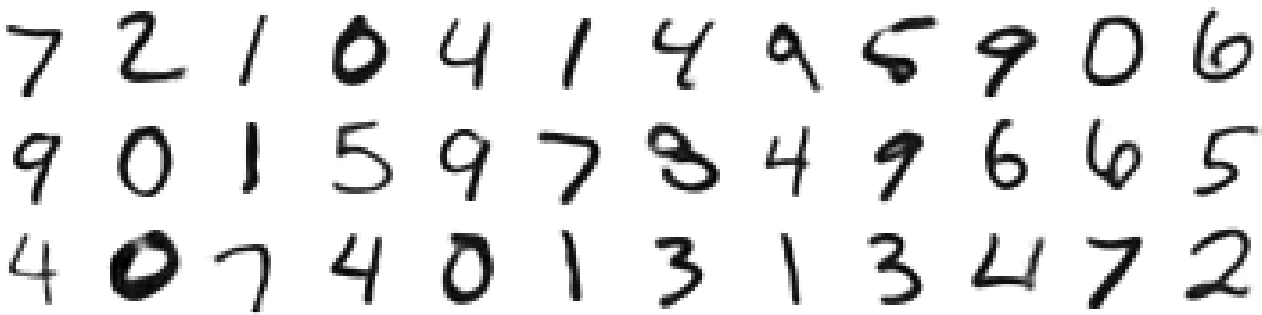


Original

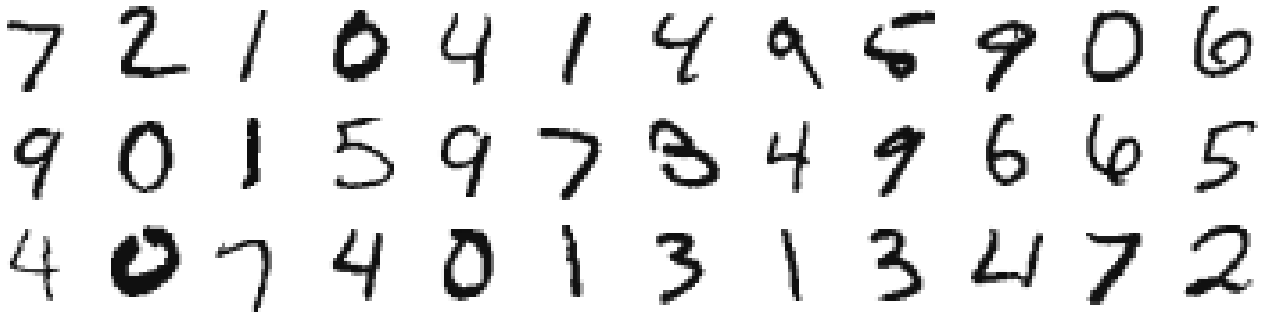


Corrupted ($\sigma = 4$)

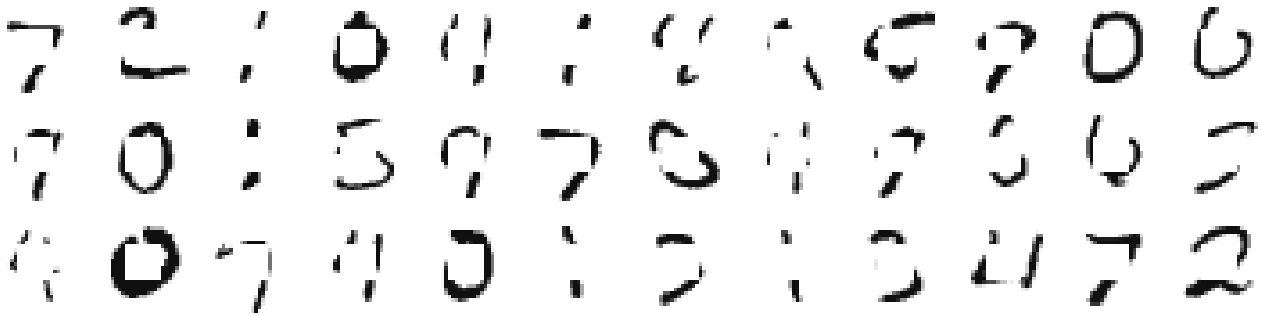
Reconstructed



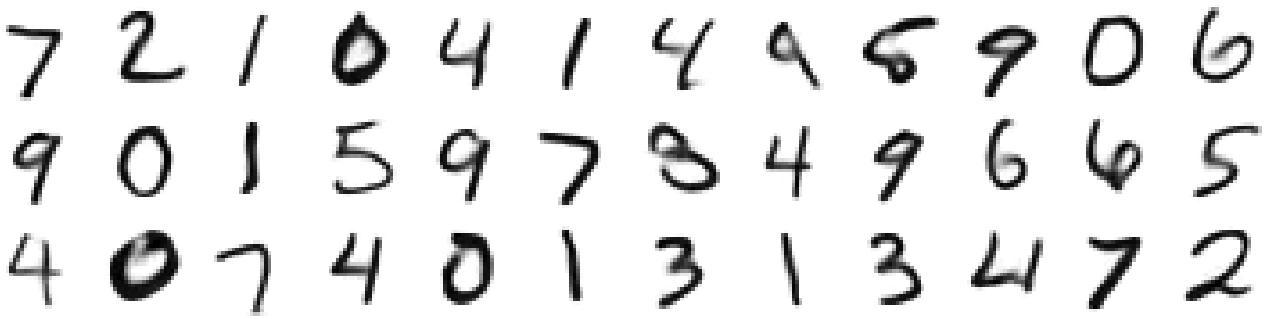
Original



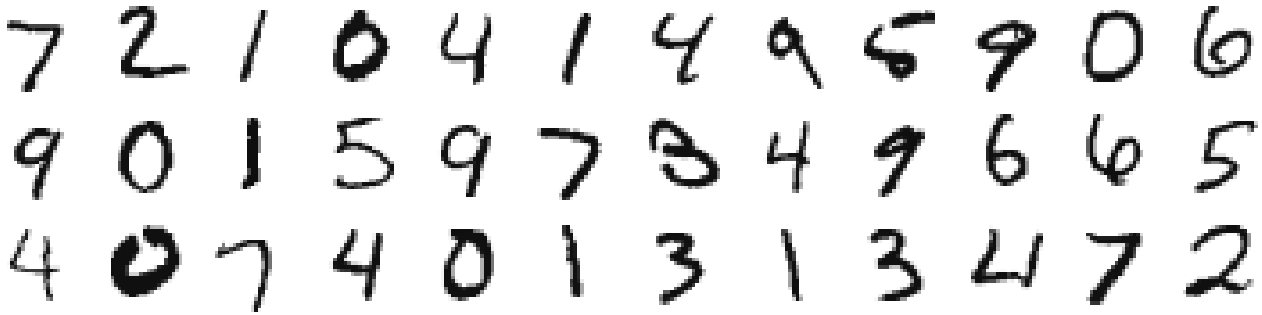
Corrupted (10×10)



Reconstructed

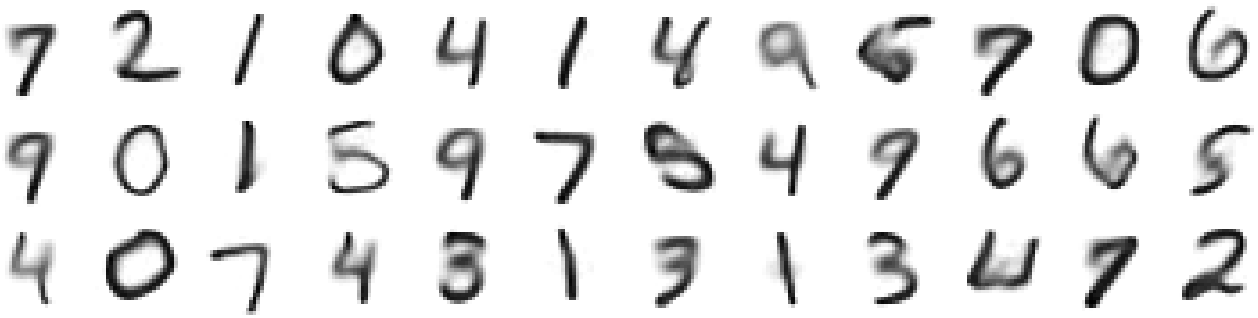


Original



Corrupted (16×16)

Reconstructed



A key weakness of this type of denoising is that the posterior

$$\mu_{X|\tilde{x}}$$

may be non-deterministic, possibly multi-modal.

A key weakness of this type of denoising is that the posterior

$$\mu_{X|\tilde{x}}$$

may be non-deterministic, possibly multi-modal.

If we train an autoencoder with the quadratic loss, the best reconstruction is

$$\phi(\tilde{X}) = \mathbb{E}\left[X \,\middle|\, \tilde{X}\right],$$

which may be very unlikely under $\mu_{X|\tilde{x}}$.

A key weakness of this type of denoising is that the posterior

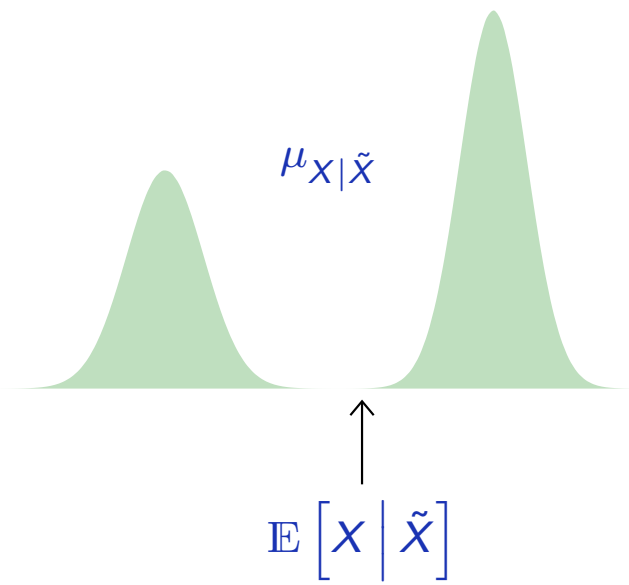
$$\mu_{X|\tilde{x}}$$

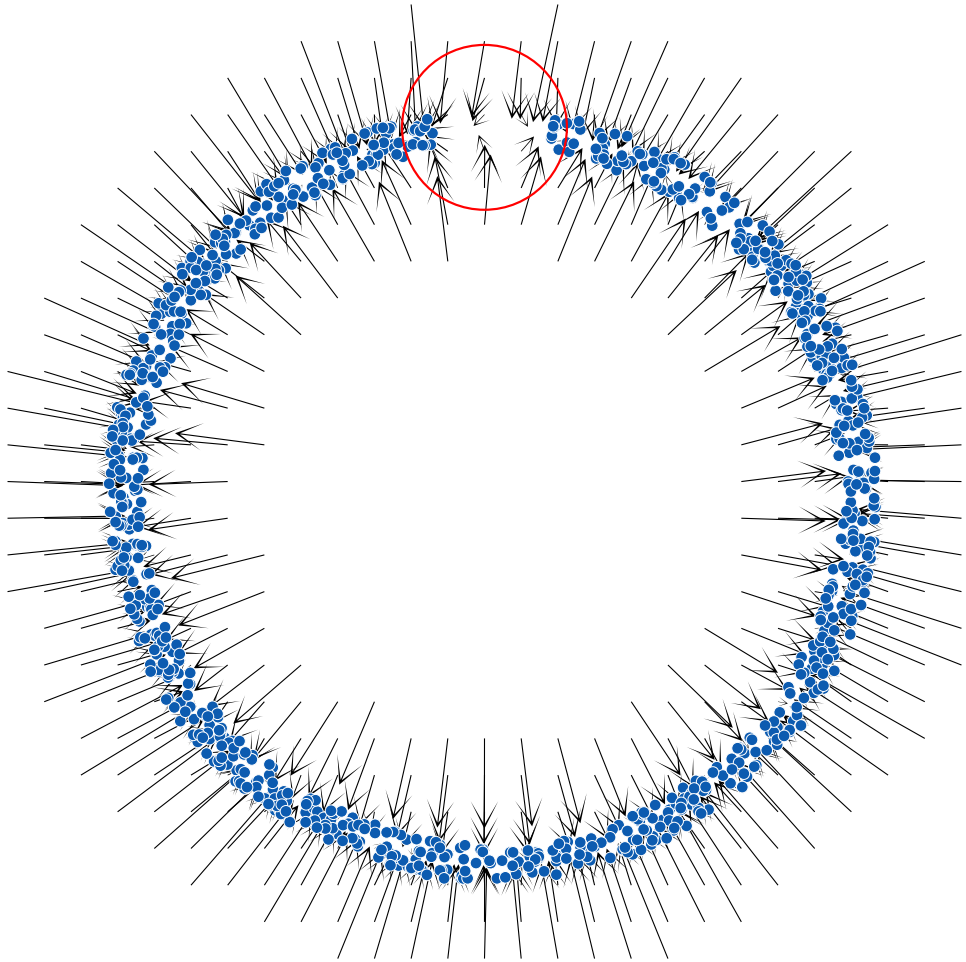
may be non-deterministic, possibly multi-modal.

If we train an autoencoder with the quadratic loss, the best reconstruction is

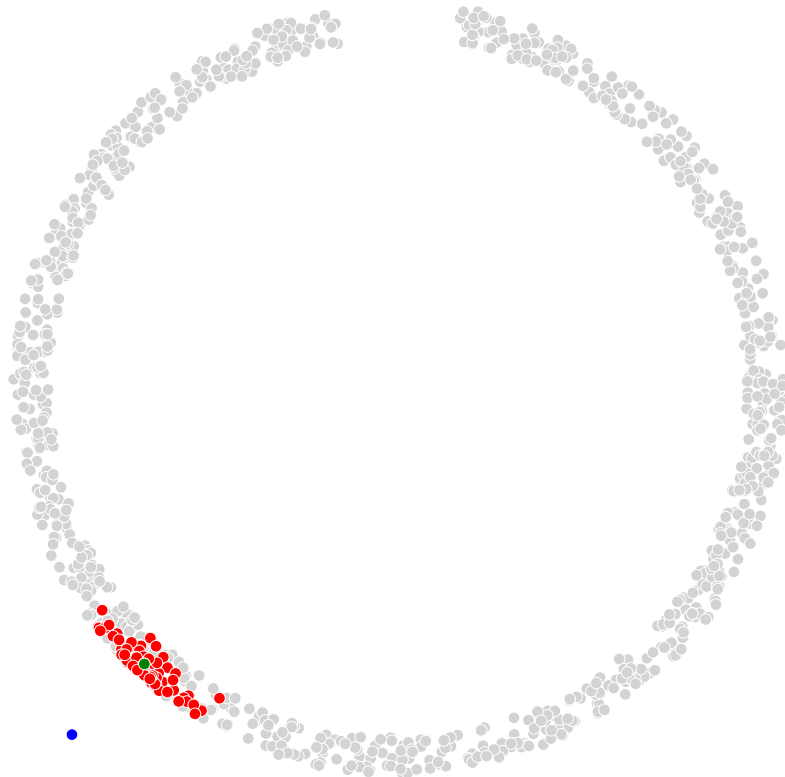
$$\phi(\tilde{X}) = \mathbb{E}\left[X \,\middle|\, \tilde{X}\right],$$

which may be very unlikely under $\mu_{X|\tilde{x}}$.

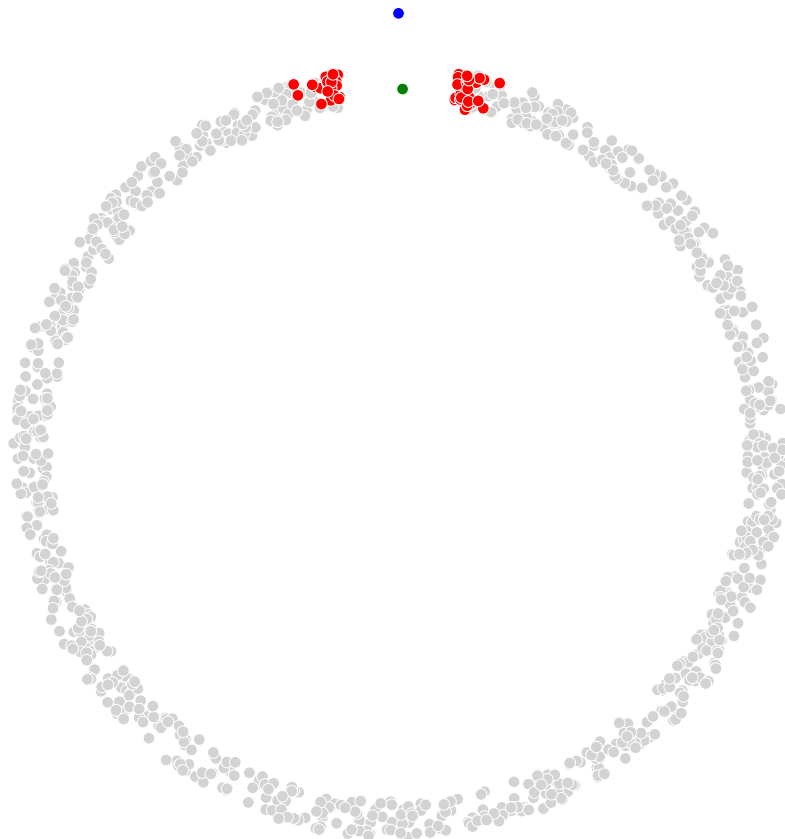




This phenomenon happens here in the area marked with the red circle. Points there can be noisy versions of points originally in either of the two extremities of the open circle, hence minimizing the MSE puts the denoised result in the middle of the opening, even though the density has no mass there.



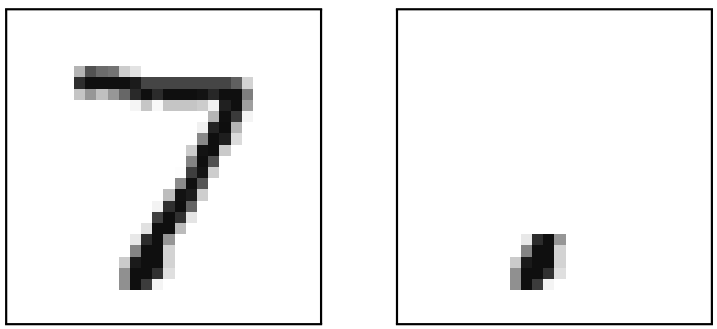
We can clarify this phenomenon given an $\tilde{x}$ (in blue) by sampling a large number of pairs $(X, \tilde{X})$, keeping only the Xs whose $\tilde{X}$ is very close to $\tilde{x}$, resulting in a sampling of $X|\tilde{X} = \tilde{x}$ (in red), whose mean $\mathbb{E}[X|\tilde{X} = \tilde{x}]$ (in green) minimizes the MSE.



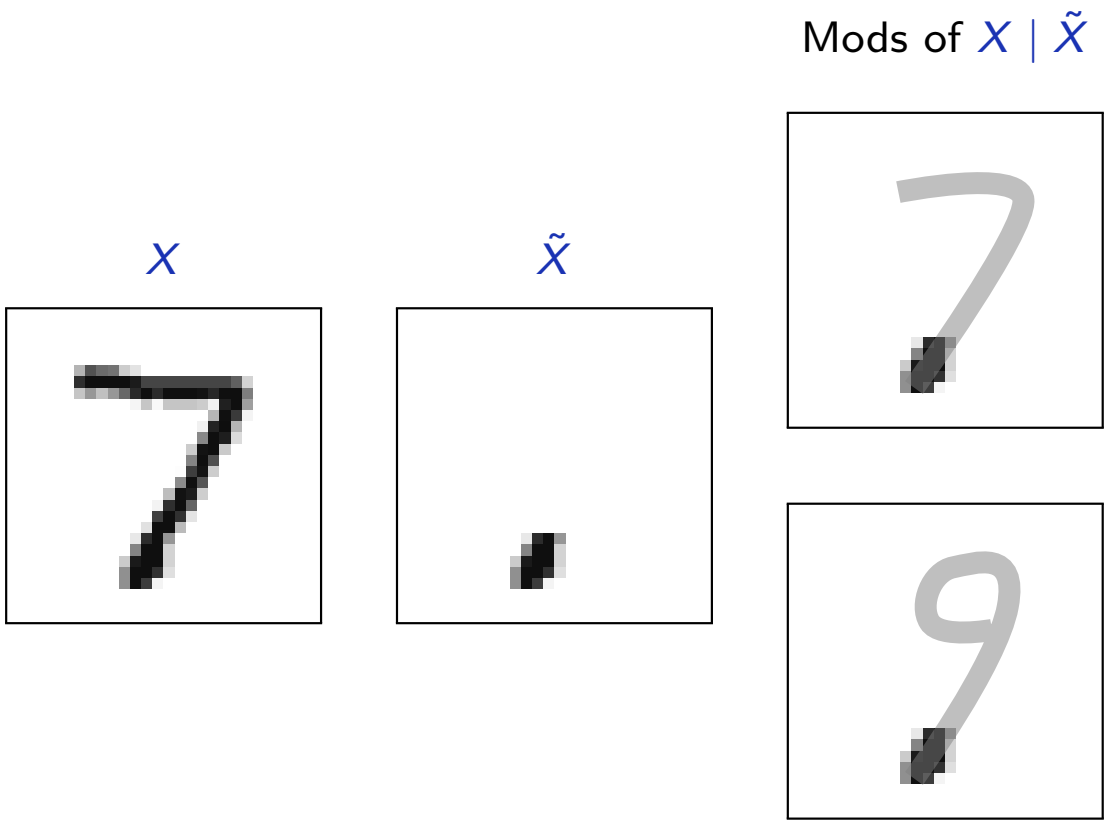
We can clarify this phenomenon given an $\tilde{x}$ (in blue) by sampling a large number of pairs $(X, \tilde{X})$, keeping only the Xs whose $\tilde{X}$ is very close to $\tilde{x}$, resulting in a sampling of $X|\tilde{X} = \tilde{x}$ (in red), whose mean $\mathbb{E}[X|\tilde{X} = \tilde{x}]$ (in green) minimizes the MSE.

We observe the same phenomenon with very corrupted MNIST digits.

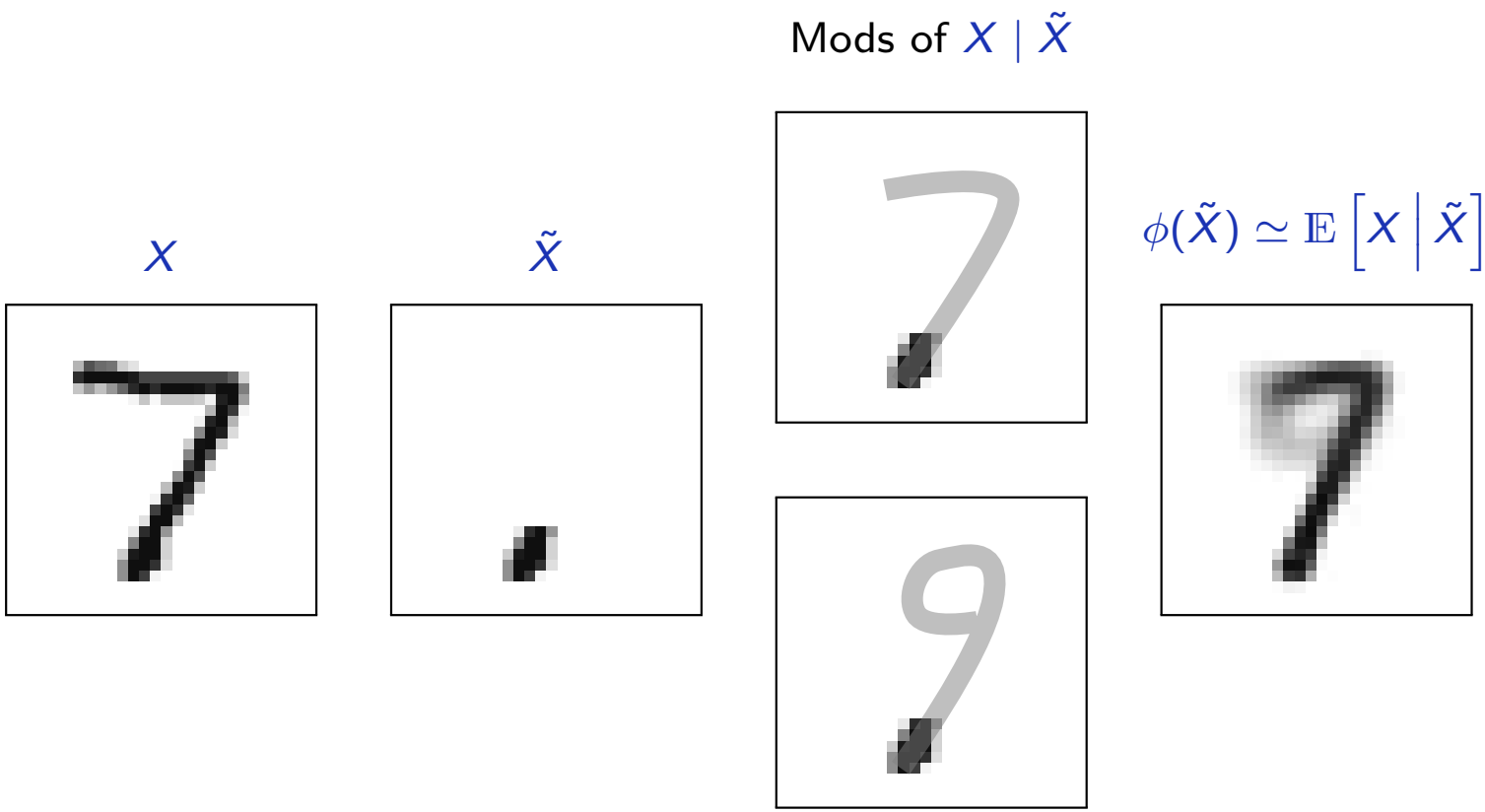
x $\tilde{x}$



We observe the same phenomenon with very corrupted MNIST digits.



We observe the same phenomenon with very corrupted MNIST digits.



This can be mitigated by using in place of loss a second network that assesses if the output is realistic.

Such methods are called **adversarial** since the second network aims at spotting the mistakes of the first, and the first aims at fooling the second.

It can be combined with a stochastic denoiser that samples an X according to $X \mid \tilde{X}$ instead of computing a deterministic reconstruction.

We will come back to that in lecture 11.1. "Generative Adversarial Networks".

Noise2Noise

Denoising can be achieved without clean samples, if the noise is additive and unbiased. Consider ϵ and δ two unbiased and independent noises. We have

$$\begin{aligned}
&\mathbb{E}\Big[\,\|\phi(X+\epsilon;\theta)-(X+\delta)\|^2\,\Big] \\
&\quad=\mathbb{E}\Big[\,\|(\phi(X+\epsilon;\theta)-X)-\delta\|^2\,\Big] \\
&\quad=\mathbb{E}\Big[\,\|\phi(X+\epsilon;\theta)-X\|^2\,\Big]-2\mathbb{E}\Big[\delta^\top(\phi(X+\epsilon;\theta)-X)\Big]+\mathbb{E}\Big[\|\delta\|^2\Big] \\
&\quad=\mathbb{E}\Big[\,\|\phi(X+\epsilon;\theta)-X\|^2\,\Big]-2\underbrace{\mathbb{E}\big[\delta\big]^\top}_{=0}\mathbb{E}\big[\phi(X+\epsilon;\theta)-X\big]+\mathbb{E}\Big[\|\delta\|^2\Big] \\
&\quad=\mathbb{E}\Big[\,\|\phi(X+\epsilon;\theta)-X\|^2\,\Big]+\mathbb{E}\Big[\|\delta\|^2\Big].
\end{aligned}$$

Hence

$$\operatorname*{argmin}_\theta \mathbb{E}\Big[\,\|\phi(X+\epsilon;\theta)-(X+\delta)\|^2\,\Big]=\operatorname*{argmin}_\theta \mathbb{E}\Big[\,\|\phi(X+\epsilon;\theta)-X\|^2\,\Big].$$

Using L_1 instead of L_2 estimates the median instead of the mean, and similarly is stable to noise that keeps the median unchanged.

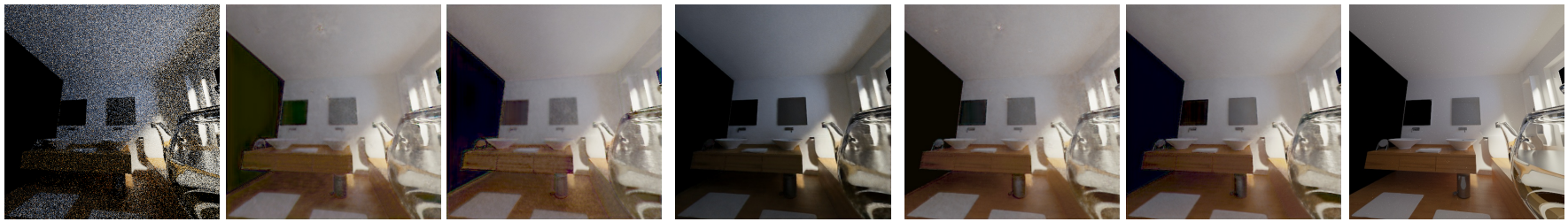
Lehtinen et al. (2018)'s Noise2Noise approach uses this for image restoration, as many existing image generative processes induce an unbiased noise.

> *In many image restoration tasks, the expectation of the corrupted input data is the clean target that we seek to restore. Low-light photography is an example: a long, noise-free exposure is the average of short, independent, noisy exposures.*
>
> *Physically accurate renderings of virtual environments are most often generated through a process known as Monte Carlo path tracing. /.../ The Monte Carlo integrator is constructed such that the intensity of each pixel is the expectation of the random path sampling process, i.e., the sampling noise is zero-mean.*

(Lehtinen et al., 2018)

NAME	N_{out}	FUNCTION
INPUT	n	
ENC_CONV0	48	Convolution 3×3
ENC_CONV1	48	Convolution 3×3
POOL1	48	Maxpool 2×2
ENC_CONV2	48	Convolution 3×3
POOL2	48	Maxpool 2×2
ENC_CONV3	48	Convolution 3×3
POOL3	48	Maxpool 2×2
ENC_CONV4	48	Convolution 3×3
POOL4	48	Maxpool 2×2
ENC_CONV5	48	Convolution 3×3
POOL5	48	Maxpool 2×2
ENC_CONV6	48	Convolution 3×3
UPSAMPLE5	48	Upsample 2×2
CONCAT5	96	Concatenate output of POOL4
DEC_CONV5A	96	Convolution 3×3
DEC_CONV5B	96	Convolution 3×3
UPSAMPLE4	96	Upsample 2×2
CONCAT4	144	Concatenate output of POOL3
DEC_CONV4A	96	Convolution 3×3
DEC_CONV4B	96	Convolution 3×3
UPSAMPLE3	96	Upsample 2×2
CONCAT3	144	Concatenate output of POOL2
DEC_CONV3A	96	Convolution 3×3
DEC_CONV3B	96	Convolution 3×3
UPSAMPLE2	96	Upsample 2×2
CONCAT2	144	Concatenate output of POOL1
DEC_CONV2A	96	Convolution 3×3
DEC_CONV2B	96	Convolution 3×3
UPSAMPLE1	96	Upsample 2×2
CONCAT1	$96+n$	Concatenate INPUT
DEC_CONV1A	64	Convolution 3×3
DEC_CONV1B	32	Convolution 3×3
DEV_CONV1C	m	Convolution 3×3, linear act.

(Lehtinen et al., 2018)



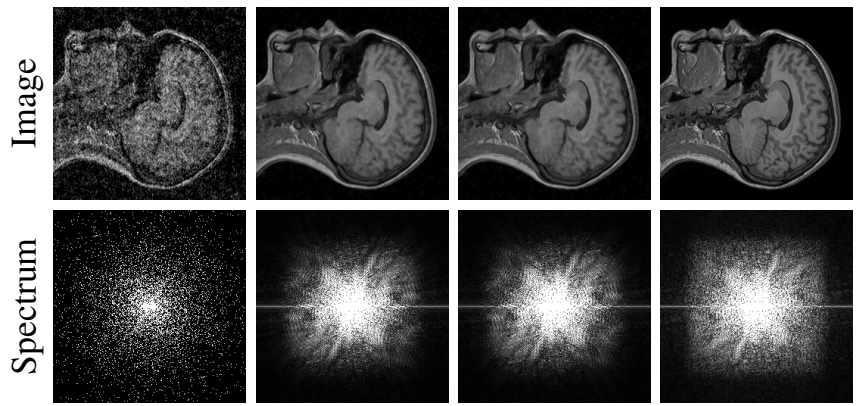
Input, 8 spp	L_2 with $\hat{x}, \hat{y}$	L_2 with $T(\hat{x}), \hat{y}$	L_2 with $T(\hat{x}), T(\hat{y})$	L_{HDR} with $\hat{x}, \hat{y}$	L_{HDR} with $T(\hat{x}), \hat{y}$	Reference, 32k spp
11.32 dB	25.46 dB	25.39 dB	15.50 dB	29.05 dB	30.09 dB	PSNR

Figure 6. Comparison of various loss functions for training a Monte Carlo denoiser with noisy target images rendered at 8 samples per pixel (spp). In this high-dynamic range setting, our custom relative loss L_{HDR} is clearly superior to L_2. Applying a non-linear tone map to the inputs is beneficial, while applying it to the target images skews the distribution of noise and leads to wrong, visibly too dark results.

(a) Input (64 spp), 23.93 dB (b) Noisy targets, 32.42 dB (c) Clean targets, 32.95 dB (d) Reference (131k spp)

Figure 7. Denoising a Monte Carlo rendered image. (a) Image rendered with 64 samples per pixel. (b) Denoised 64 spp input, trained using 64 spp targets. (c) Same as previous, but trained on clean targets. (d) Reference image rendered with 131 072 samples per pixel. PSNR values refer to the images shown here, see text for averages over the entire validation set.

(Lehtinen et al., 2018)



(a) Input 18.93 dB (b) Noisy trg. 29.77 dB (c) Clean trg. 29.81 dB (d) Reference

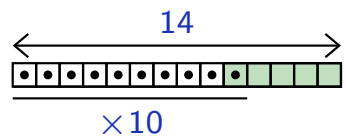
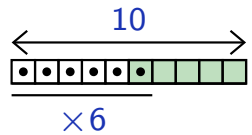
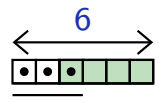
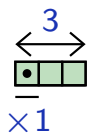
Figure 9. MRI reconstruction example. (a) Input image with only 10% of spectrum samples retained and scaled by $1/p$. (b) Reconstruction by a network trained with noisy target images similar to the input image. (c) Same as previous, but training done with clean target images similar to the reference image. (d) Original, uncorrupted image. PSNR values refer to the images shown here, see text for averages over the entire validation set.

(Lehtinen et al., 2018)

Super-resolution

A special case of denoising is to increase an image resolution. We use an encoder/decoder whose encoder's input is smaller than the decoder's output.

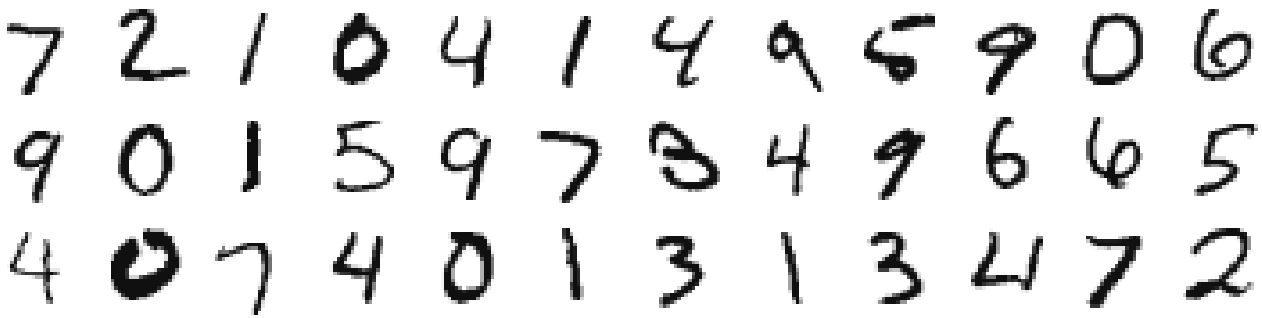
Encoder

Tensor sizes / operations	
$1 \times 14 \times 14$	
`nn.Conv2d(1, 32, kernel_size=5, stride=1)`	
$32 \times 10 \times 10$	
`nn.Conv2d(32, 32, kernel_size=5, stride=1)`	
$32 \times 6 \times 6$	
`nn.Conv2d(32, 32, kernel_size=4, stride=1)`	
$32 \times 3 \times 3$	
`nn.Conv2d(32, 32, kernel_size=3, stride=1)`	
$32 \times 1 \times 1$	

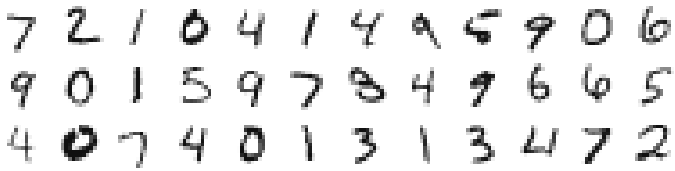
```
MNISTUpscaler(
  (encoder): Sequential(
    (0): Conv2d(1, 32, kernel_size=(5, 5), stride=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(32, 32, kernel_size=(5, 5), stride=(1, 1))
    (3): ReLU(inplace=True)
    (4): Conv2d(32, 32, kernel_size=(4, 4), stride=(1, 1))
    (5): ReLU(inplace=True)
    (6): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
  )
  (decoder): Sequential(
    (0): ConvTranspose2d(32, 32, kernel_size=(4, 4), stride=(1, 1))
    (1): ReLU(inplace=True)
    (2): ConvTranspose2d(32, 32, kernel_size=(3, 3), stride=(2, 2))
    (3): ReLU(inplace=True)
    (4): ConvTranspose2d(32, 32, kernel_size=(4, 4), stride=(2, 2))
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(32, 32, kernel_size=(5, 5), stride=(1, 1))
    (7): ReLU(inplace=True)
    (8): ConvTranspose2d(32, 1, kernel_size=(5, 5), stride=(1, 1))
  )
)
```

```
for original in train_input.split(batch_size):
    input = F.avg_pool2d(original, kernel_size = 2)
    output = model(input)
    loss = (output - original).pow(2).sum() / output.size(0)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

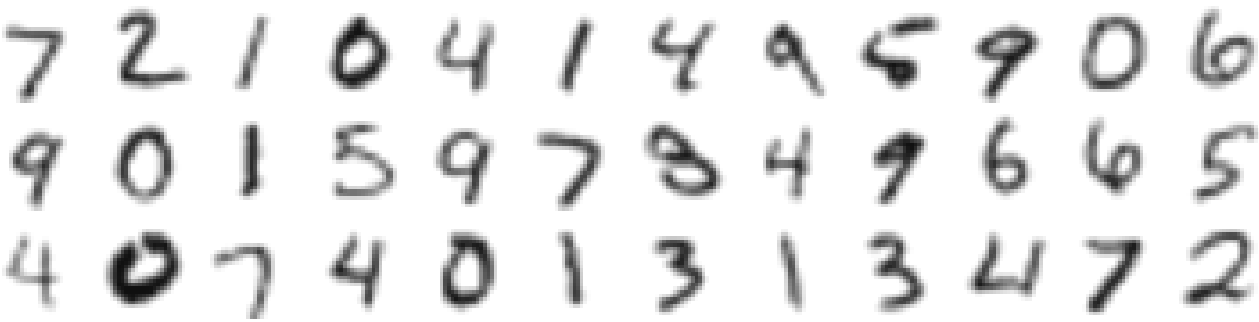
Original



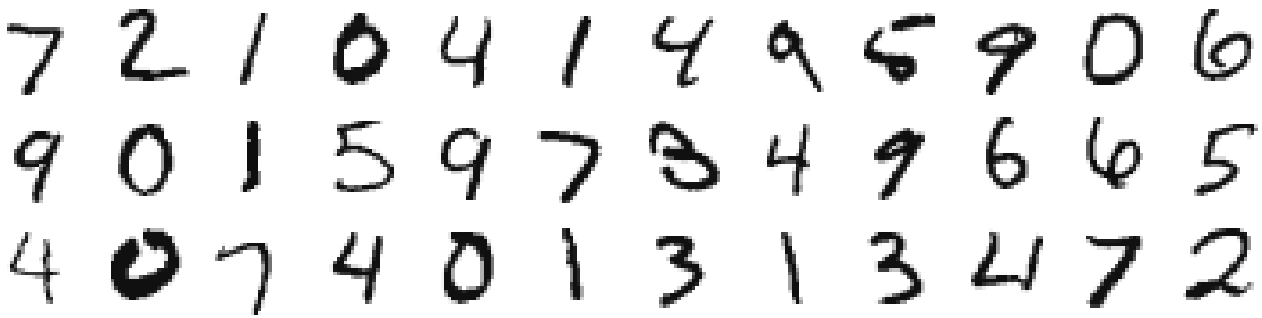
Input



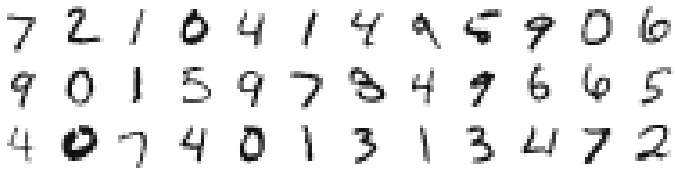
Bilinear interpolation



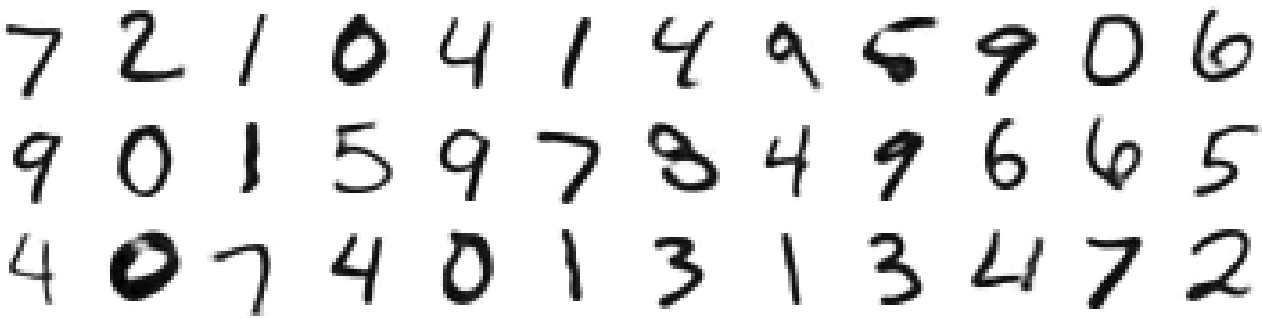
Original



Input

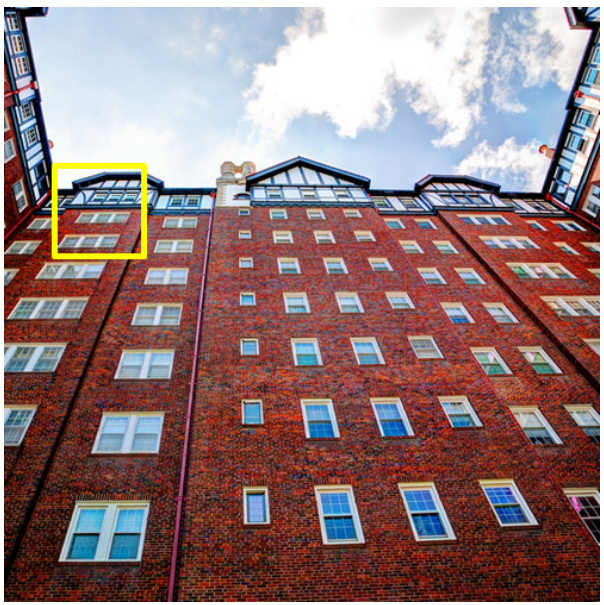


Autoencoder output

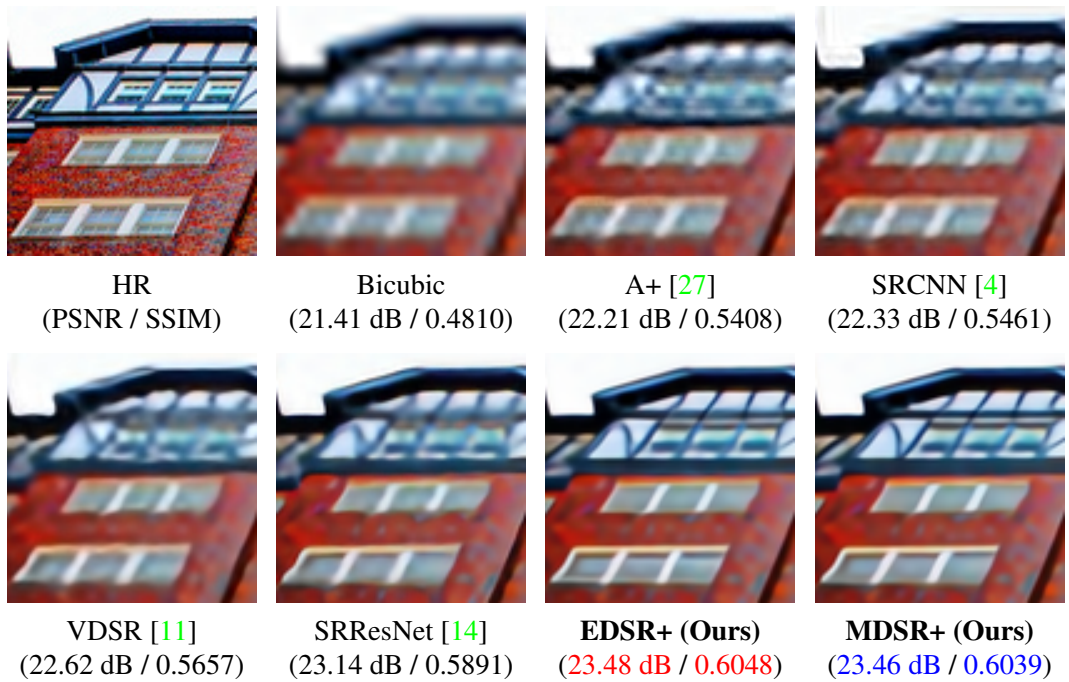


Lim et al. (2017) use two different resnets.

	EDSR	MDSR
Nb blocks	32	80
Channels	256	64
Nb parameters	43M	8M



img034 from Urban100 [10]



HR
(PSNR / SSIM)

Bicubic
(21.41 dB / 0.4810)

A+ [27]
(22.21 dB / 0.5408)

SRCNN [4]
(22.33 dB / 0.5461)

VDSR [11]
(22.62 dB / 0.5657)

SRResNet [14]
(23.14 dB / 0.5891)

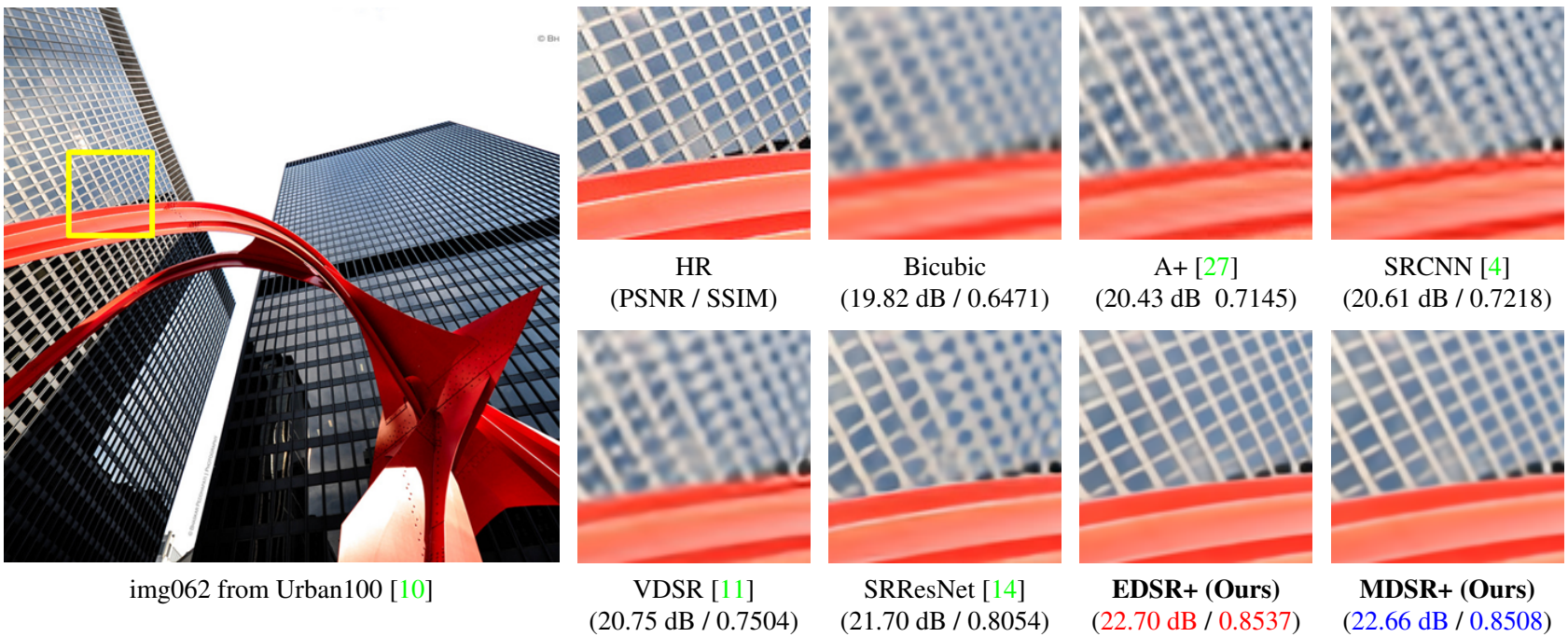
EDSR+ (Ours)
(23.48 dB / 0.6048)

MDSR+ (Ours)
(23.46 dB / 0.6039)

(Lim et al., 2017)

Lim et al. (2017) use two different resnets.

	EDSR	MDSR
Nb blocks	32	80
Channels	256	64
Nb parameters	43M	8M



(Lim et al., 2017)

Lim et al. (2017) use two different resnets.

	EDSR	MDSR
Nb blocks	32	80
Channels	256	64
Nb parameters	43M	8M

0869 from DIV2K [26]

HR
(PSNR / SSIM)

Bicubic
(22.66 dB / 0.8025)

A+ [27]
(23.10 dB / 0.8251)

SRCNN [4]
(23.14 dB / 0.8280)

VDSR [11]
(23.36 dB / 0.8365)

SRResNet [14]
(23.71 dB / 0.8485)

EDSR+ (Ours)
(23.89 dB / 0.8563)

MDSR+ (Ours)
(23.90 dB / 0.8558)

(Lim et al., 2017)

Autoencoders as self-training

Vincent et al. (2010) interpret training the autoencoder as maximizing the mutual information between the input and the latent states.

Vincent et al. (2010) interpret training the autoencoder as maximizing the mutual information between the input and the latent states.

Let X be a sample, $Z = f(X; \theta)$ its latent representation, and $q(x, z)$ the distribution of (X, Z).

Vincent et al. (2010) interpret training the autoencoder as maximizing the mutual information between the input and the latent states.

Let X be a sample, $Z = f(X; \theta)$ its latent representation, and $q(x, z)$ the distribution of (X, Z).

We have

$$\operatorname*{argmax}_{\theta} \mathbb{I}(X; Z) = \operatorname*{argmax}_{\theta} \mathbb{E}_{q(X,Z)} \Big[\log q(X \mid Z) \Big].$$

However, there is no expression of $q(X \mid Z)$ in any reasonable setup.

However, for any distribution p we have

$$\mathbb{E}_{q(X,Z)}\Big[\log q(X \mid Z)\Big] \geq \mathbb{E}_{q(X,Z)}\Big[\log p(X \mid Z)\Big].$$

However, for any distribution p we have

$$\mathbb{E}_{q(X,Z)}\Big[\log q(X \mid Z)\Big] \geq \mathbb{E}_{q(X,Z)}\Big[\log p(X \mid Z)\Big].$$

So we can in particular try to find a "good p", so that the right term is a good approximation of the left one.

If we consider the following model for p

$$p(\cdot \mid Z = z) = \mathcal{N}(g(z; \eta), \sigma)$$

where g is deterministic and σ fixed

If we consider the following model for p

$$p(\cdot \mid Z = z) = \mathcal{N}(g(z;\eta), \sigma)$$

where g is deterministic and σ fixed, we get

$$\mathbb{E}_{q(X,Z)}\Big[\log p(X \mid Z)\Big] = -\frac{1}{2\sigma^2}\mathbb{E}_{q(X,Z)}\left[\|X - g(f(X;\theta);\eta)\|^2\right] + k.$$

If we consider the following model for p

$$p(\cdot \mid Z = z) = \mathcal{N}(g(z;\eta), \sigma)$$

where g is deterministic and σ fixed, we get

$$\mathbb{E}_{q(X,Z)}\Big[\log p(X \mid Z)\Big] = -\frac{1}{2\sigma^2}\mathbb{E}_{q(X,Z)}\left[\|X - g(f(X;\theta);\eta)\|^2\right] + k.$$

If optimizing η makes the bound tight, the final loss is the reconstruction error

$$\operatorname*{argmax}_{\theta} \mathbb{I}(X;Z) \simeq \operatorname*{argmin}_{\theta}\left(\min_{\eta} \frac{1}{N}\sum_{n=1}^{N}\|x_n - g(f(x_n;\theta);\eta)\|^2\right).$$

If we consider the following model for p

$$p(\cdot \mid Z = z) = \mathcal{N}(g(z;\eta), \sigma)$$

where g is deterministic and σ fixed, we get

$$\mathbb{E}_{q(X,Z)}\Big[\log p(X \mid Z)\Big] = -\frac{1}{2\sigma^2}\mathbb{E}_{q(X,Z)}\left[\|X - g(f(X;\theta);\eta)\|^2\right] + k.$$

If optimizing η makes the bound tight, the final loss is the reconstruction error

$$\operatorname*{argmax}_{\theta} \mathbb{I}(X;Z) \simeq \operatorname*{argmin}_{\theta}\left(\min_{\eta} \frac{1}{N}\sum_{n=1}^{N}\|x_n - g(f(x_n;\theta);\eta)\|^2\right).$$

This abstract view of the encoder as "maximizing information" justifies its use to build generic encoding layers.

In the perspective of building a good feature representation, just retaining information is not enough, otherwise the identity would be a good choice.

In the perspective of building a good feature representation, just retaining information is not enough, otherwise the identity would be a good choice.

In their work, Vincent et al. consider a denoising auto-encoder, which makes the model retain information about structures beyond local noise.

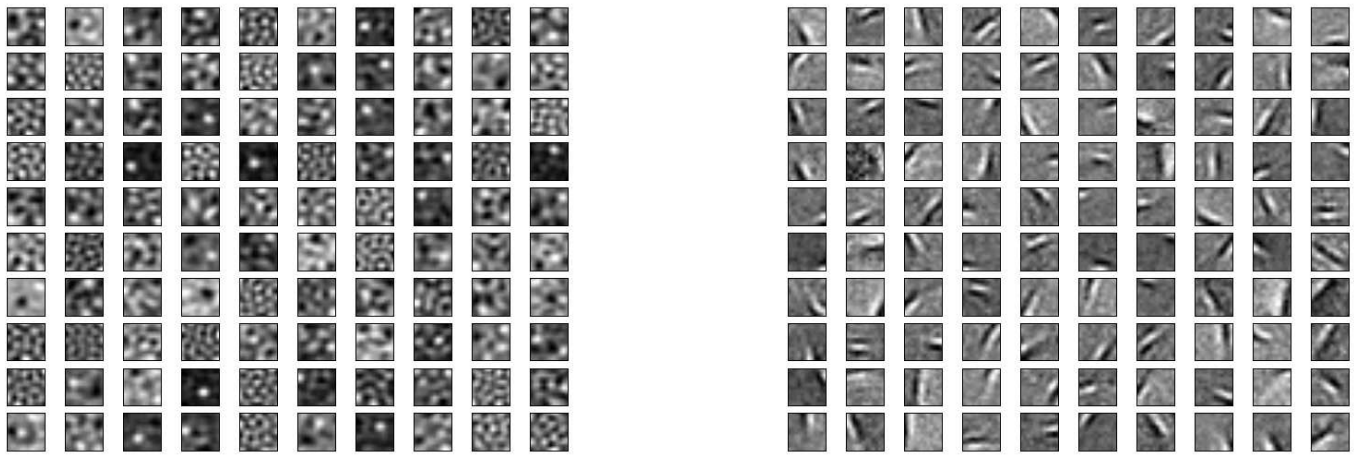


Figure 6: Weight decay vs. Gaussian noise. We show typical filters learnt from natural image patches in the over-complete case (200 hidden units). *Left:* regular autoencoder with weight decay. We tried a wide range of weight-decay values and learning rates: filters never appeared to capture a more interesting structure than what is shown here. Note that some local blob detectors are recovered compared to using no weight decay at all (Figure 5 right). *Right:* a denoising autoencoder with additive Gaussian noise ($\sigma = 0.5$) learns Gabor-like local oriented edge detectors. Clearly the filters learnt are qualitatively very different in the two cases.

(Vincent et al., 2010)

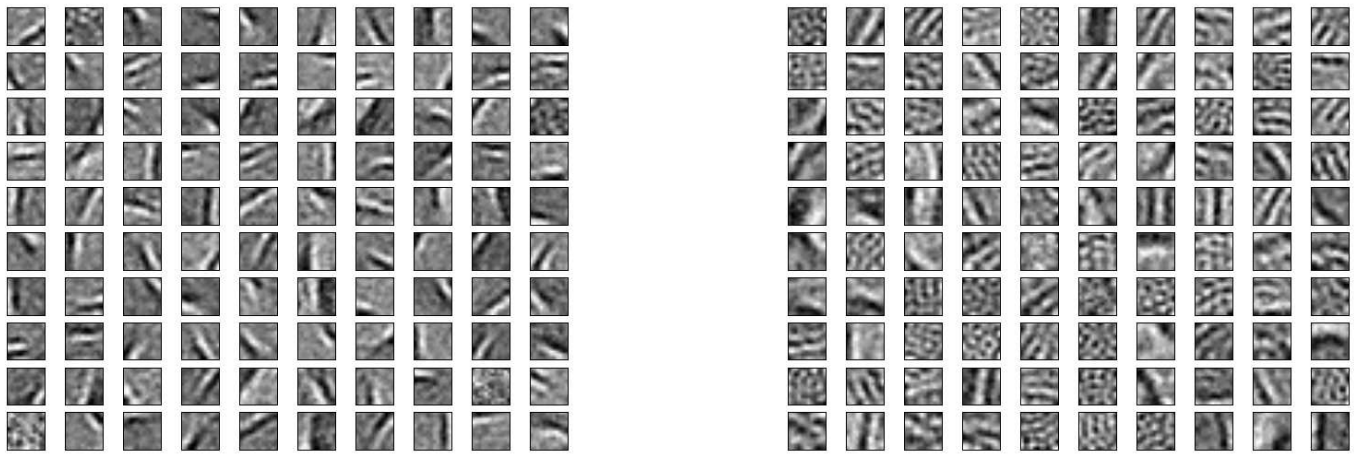
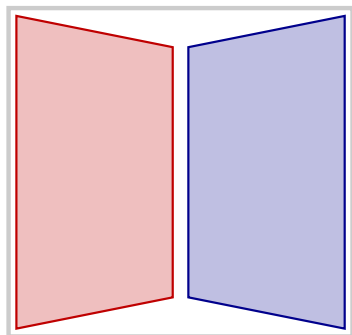


Figure 7: Filters obtained on natural image patches by denoising autoencoders using other noise types. *Left:* with 10% salt-and-pepper noise, we obtain oriented Gabor-like filters. They appear slightly less localized than when using Gaussian noise (contrast with Figure 6 right). *Right:* with 55% zero-masking noise we obtain filters that look like oriented gratings. For the three considered noise types, denoising training appears to learn filters that capture meaningful natural image statistics structure.

(Vincent et al., 2010)

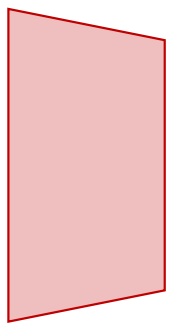
Vincent et al. build deep MLPs whose layers are initialized successively as encoders trained within a noisy autoencoder.

Vincent et al. build deep MLPs whose layers are initialized successively as encoders trained within a noisy autoencoder.

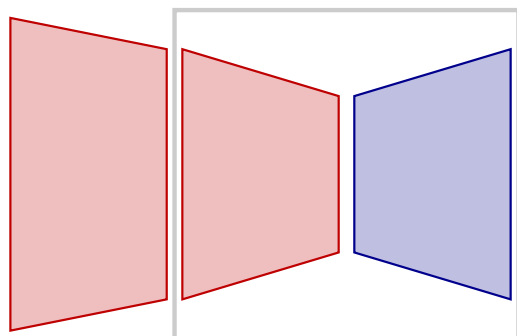


Autoencoder

Vincent et al. build deep MLPs whose layers are initialized successively as encoders trained within a noisy autoencoder.

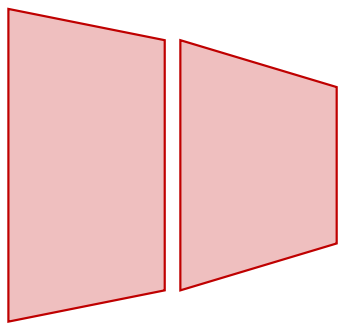


Vincent et al. build deep MLPs whose layers are initialized successively as encoders trained within a noisy autoencoder.

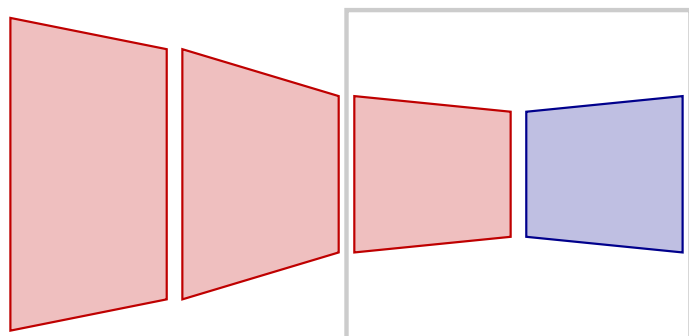


Autoencoder

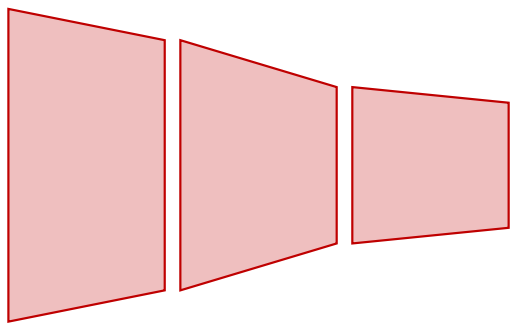
Vincent et al. build deep MLPs whose layers are initialized successively as encoders trained within a noisy autoencoder.



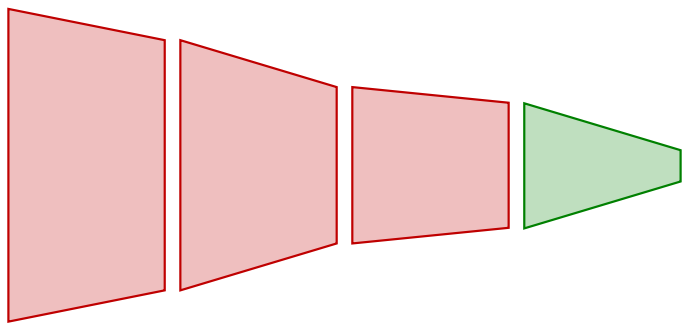
Vincent et al. build deep MLPs whose layers are initialized successively as encoders trained within a noisy autoencoder.



Vincent et al. build deep MLPs whose layers are initialized successively as encoders trained within a noisy autoencoder.



Vincent et al. build deep MLPs whose layers are initialized successively as encoders trained within a noisy autoencoder.



A final classifying layer is added and the full structure can be fine-tuned.

This approach, and others in the same spirit (Hinton et al., 2006), were seen as strategies to complement gradient-descent for building deep nets.

Data Set	SVM_{rbf}	DBN-1	SAE-3	DBN-3	SDAE-3 (ν)
MNIST	**1.40**±0.23	**1.21**±0.21	**1.40**±0.23	**1.24**±0.22	**1.28**±0.22 (25%)
basic	**3.03**±0.15	3.94±0.17	3.46±0.16	3.11±0.15	**2.84**±0.15 (10%)
rot	11.11±0.28	14.69±0.31	10.30±0.27	10.30±0.27	**9.53**±0.26 (25%)
bg-rand	14.58±0.31	9.80±0.26	11.28±0.28	**6.73**±0.22	10.30±0.27 (40%)
bg-img	22.61±0.37	**16.15**±0.32	23.00±0.37	**16.31**±0.32	**16.68**±0.33 (25%)
bg-img-rot	55.18±0.44	52.21±0.44	51.93±0.44	47.39±0.44	**43.76**±0.43 (25%)
rect	**2.15**±0.13	4.71±0.19	2.41±0.13	2.60±0.14	**1.99**±0.12 (10%)
rect-img	24.04±0.37	23.69±0.37	24.05±0.37	22.50±0.37	**21.59**±0.36 (25%)
convex	19.13±0.34	19.92±0.35	**18.41**±0.34	**18.63**±0.34	**19.06**±0.34 (10%)
tzanetakis	**14.41**±2.18	18.07±1.31	**16.15**±1.95	18.38±1.64	**16.02**±1.04(0.05)

(Vincent et al., 2010)

The end

References

G. E. Hinton, S. Osindero, and Y.-W. Teh. **A fast learning algorithm for deep belief nets**. Neural Computation, 18(7):1527–1554, July 2006.

J. Lehtinen, J. Munkberg, J. Hasselgren, S. Laine, T. Karras, M. Aittala, and T. Aila. **Noise2noise: Learning image restoration without clean data**. CoRR, abs/1803.04189, 2018.

B. Lim, S. Son, H. Kim, S. Nah, and K. Lee. **Enhanced deep residual networks for single image super-resolution**. CoRR, abs/1707.02921v1, 2017.

P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. **Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion**. Journal of Machine Learning Research (JMLR), 11:3371–3408, 2010.