Vision Specificities of Diffusion Modeling

Prof. Vincent Sitzmann

High-Level Recap: Diffusion & Flow Matching

Empirical Data Distribution

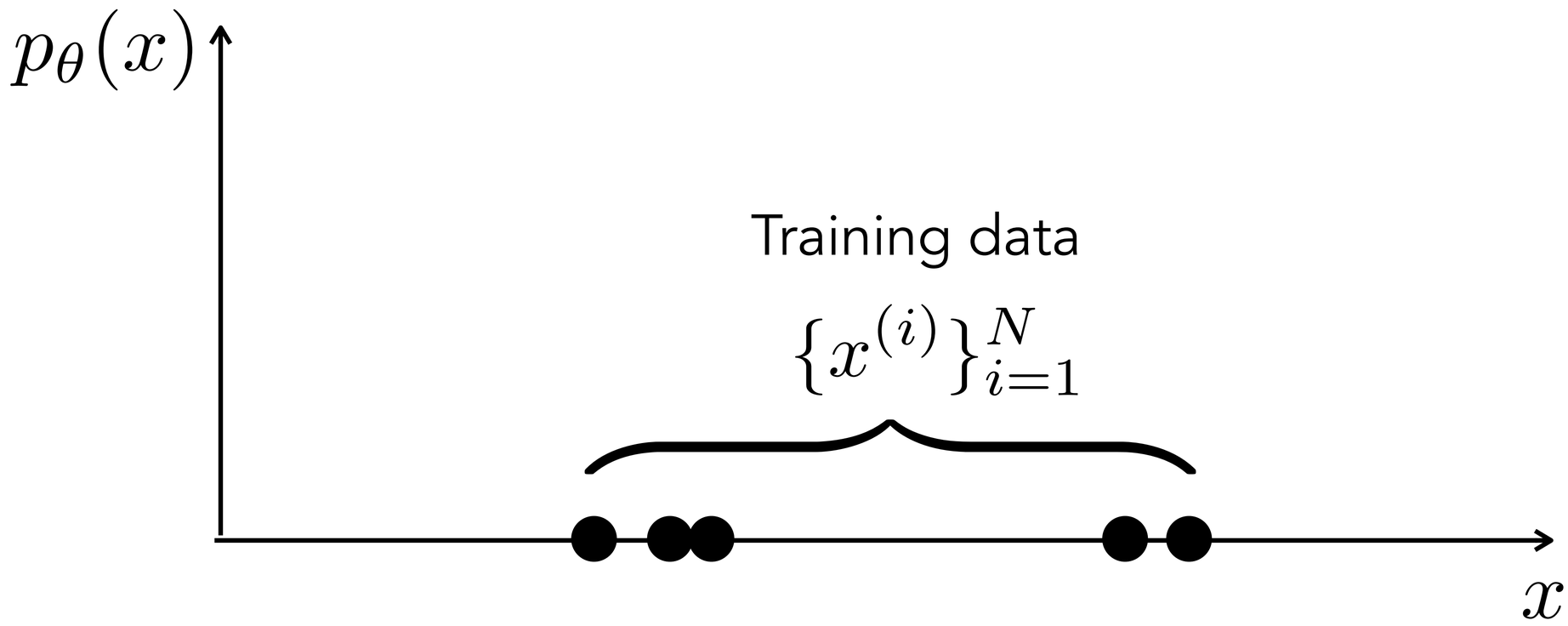
$$p_\theta : \mathcal{X} \rightarrow [0, \infty)$$

Empirical Data Distribution

$$p_\theta : \mathcal{X} \rightarrow [0, \infty)$$

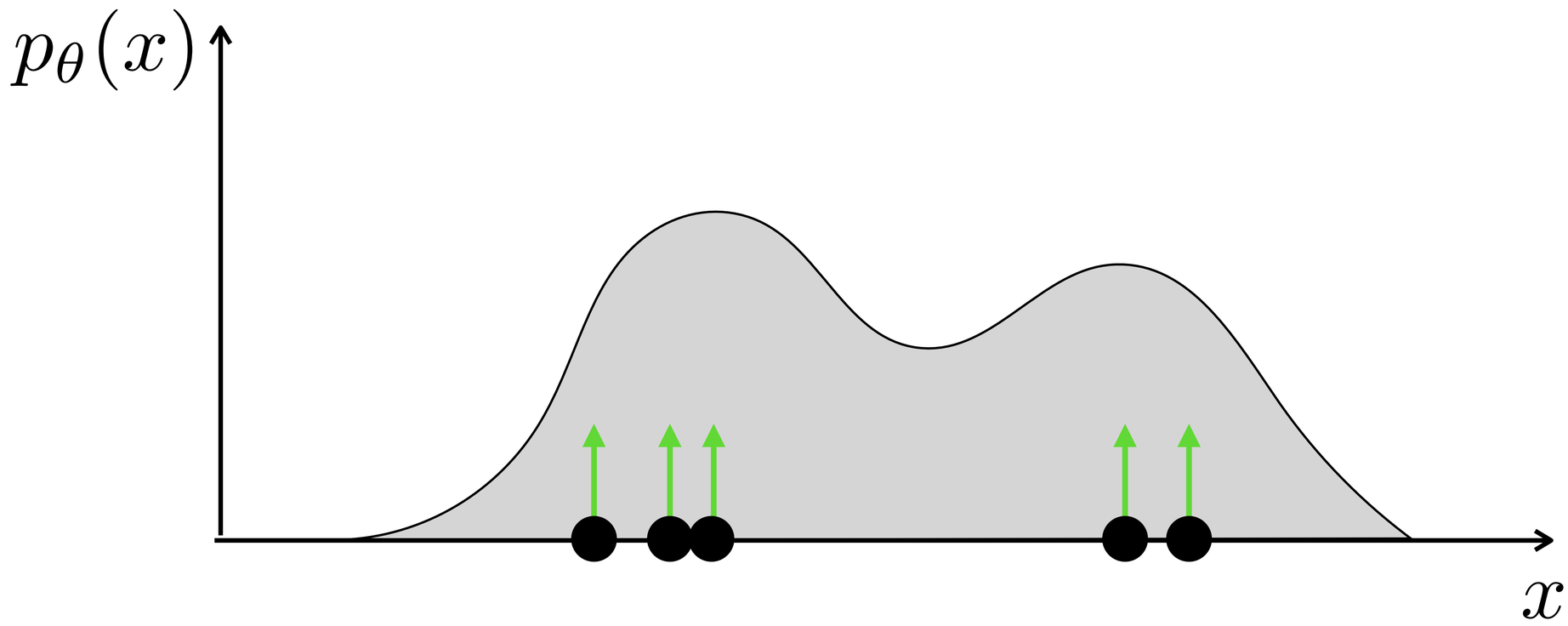
Empirical Data Distribution

$$p_\theta : \mathcal{X} \rightarrow [0, \infty)$$



Density Models

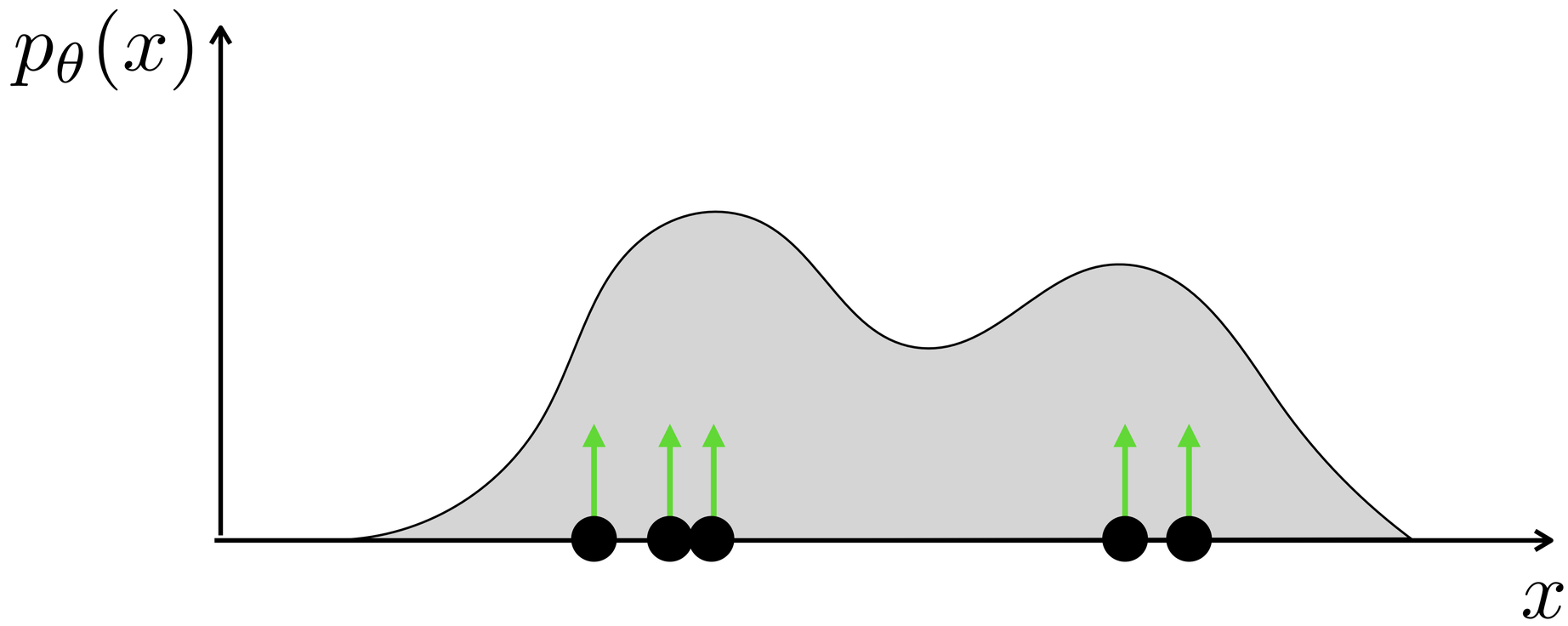
$$p(\mathbf{x}) = \frac{e^{-f_\theta(\mathbf{x})}}{Z_\theta}$$



Density Models

$$p(\mathbf{x}) = \frac{e^{-f_\theta(\mathbf{x})}}{Z_\theta}$$

Can use to define a continuous data distribution by defining $f_\theta(\mathbf{x})$, a real-valued function.

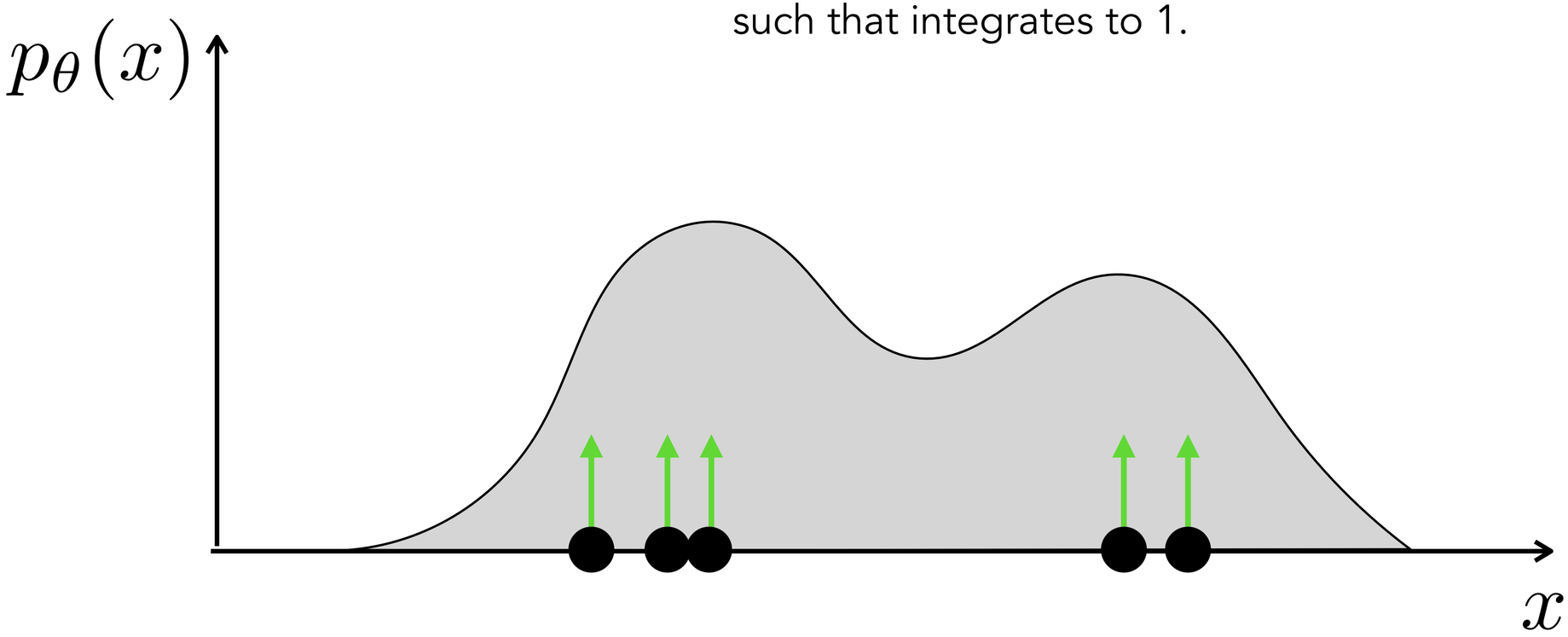


Density Models

$$p(\mathbf{x}) = \frac{e^{-f_\theta(\mathbf{x})}}{Z_\theta}$$

Can use to define a continuous data distribution by defining $f_\theta(\mathbf{x})$, a real-valued function.

Z_θ is a normalizing constant such that integrates to 1.



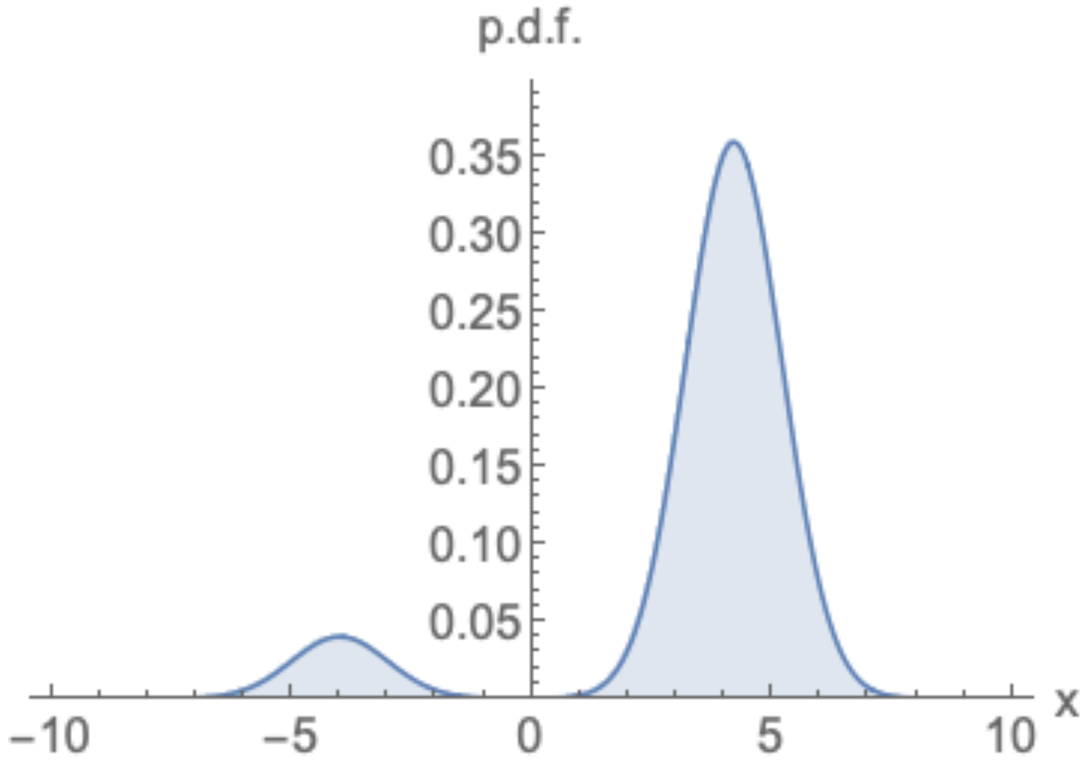
Score-Based Generative Modeling

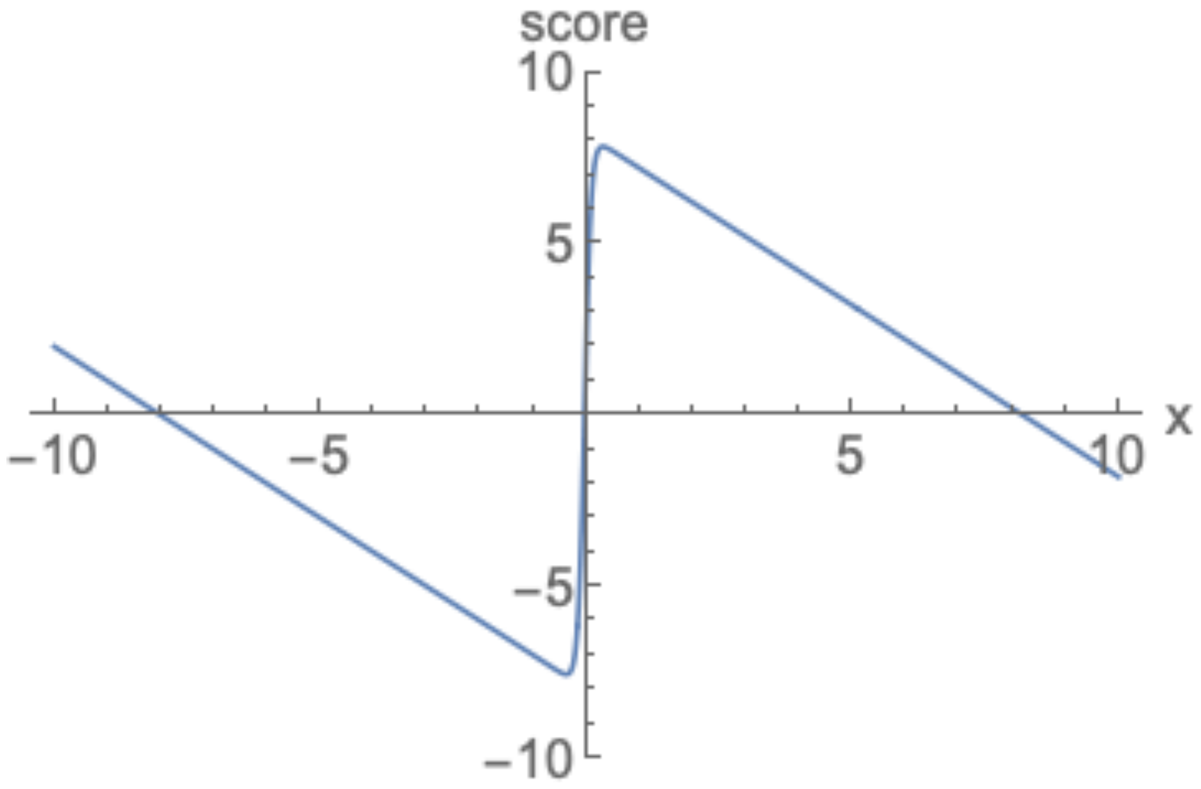
$$p(\mathbf{x}) = \frac{e^{-f(\mathbf{x})}}{Z_\theta}$$

An important tool in generative modeling is the *score* of $p(\mathbf{x})$. For a particular datapoint $\mathbf{x}$, the score tells you the direction towards higher-probability regions:

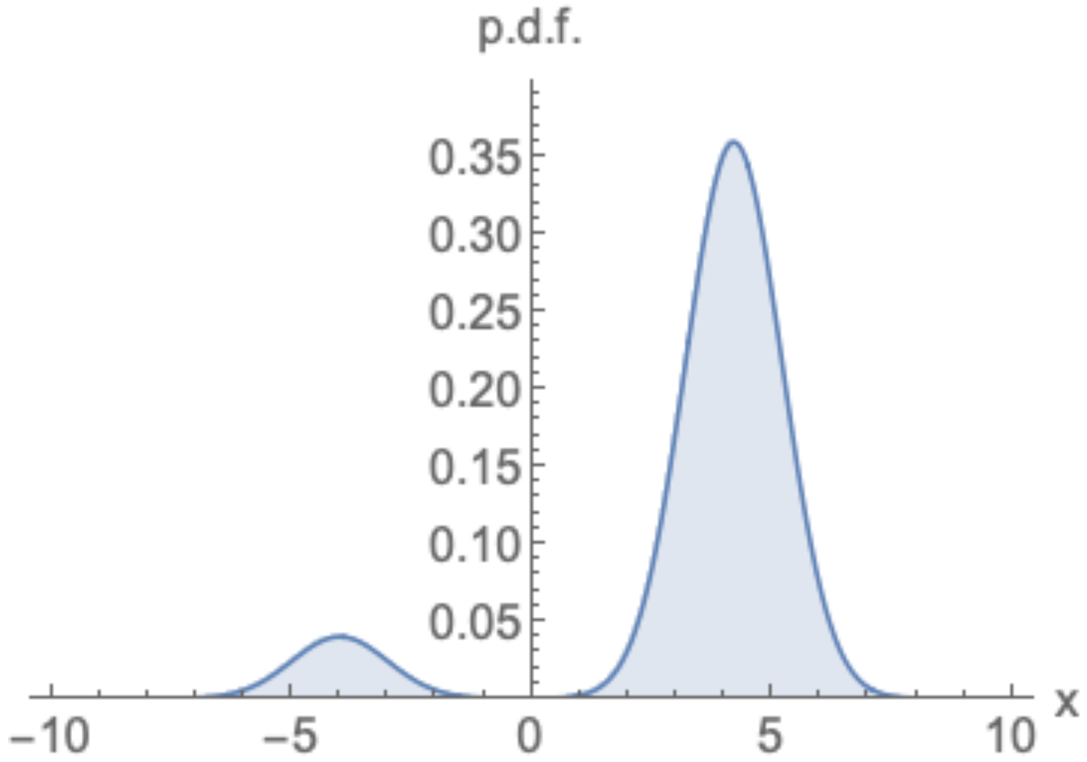
$$\nabla \log p_\theta(\mathbf{x}) = -\nabla_{\mathbf{x}} f_\theta(\mathbf{x}) - \underbrace{\nabla_{\mathbf{x}} \log Z_\theta}_{=0} = -\nabla_{\mathbf{x}} f_\theta(\mathbf{x})$$

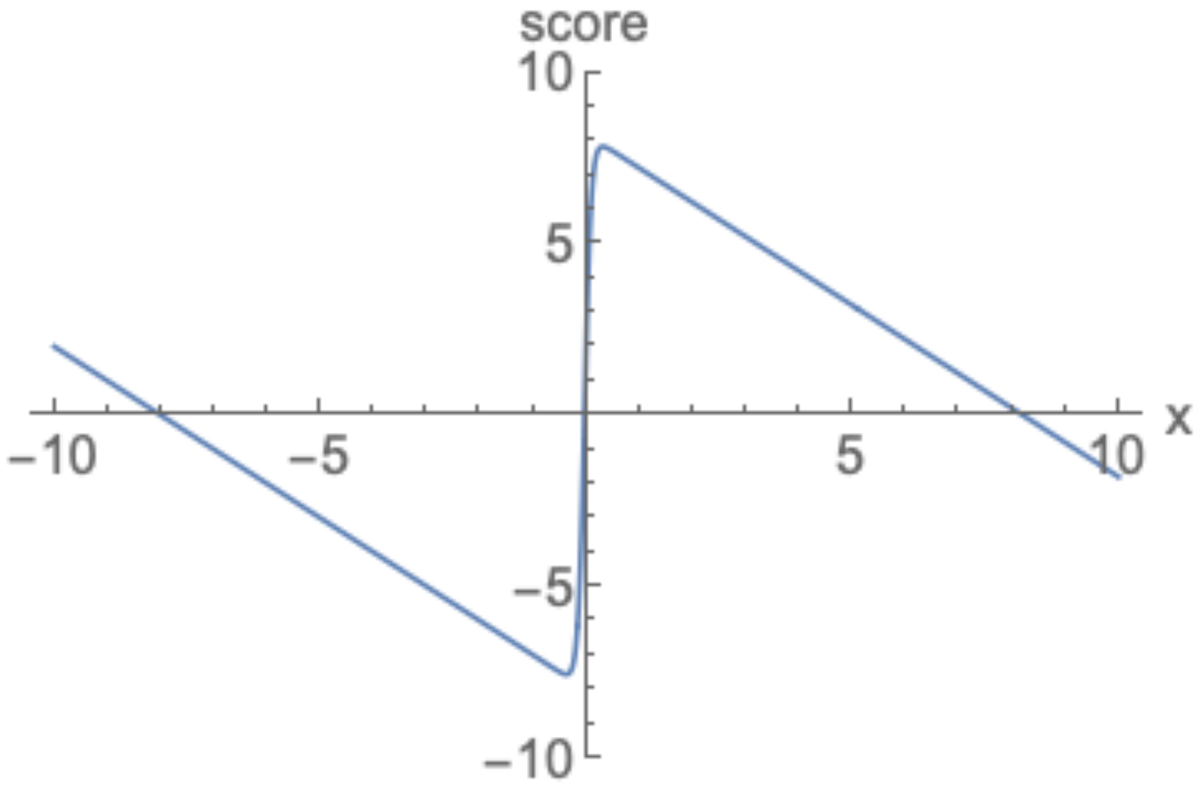
Score Functions = Gradients of the log-likelihood





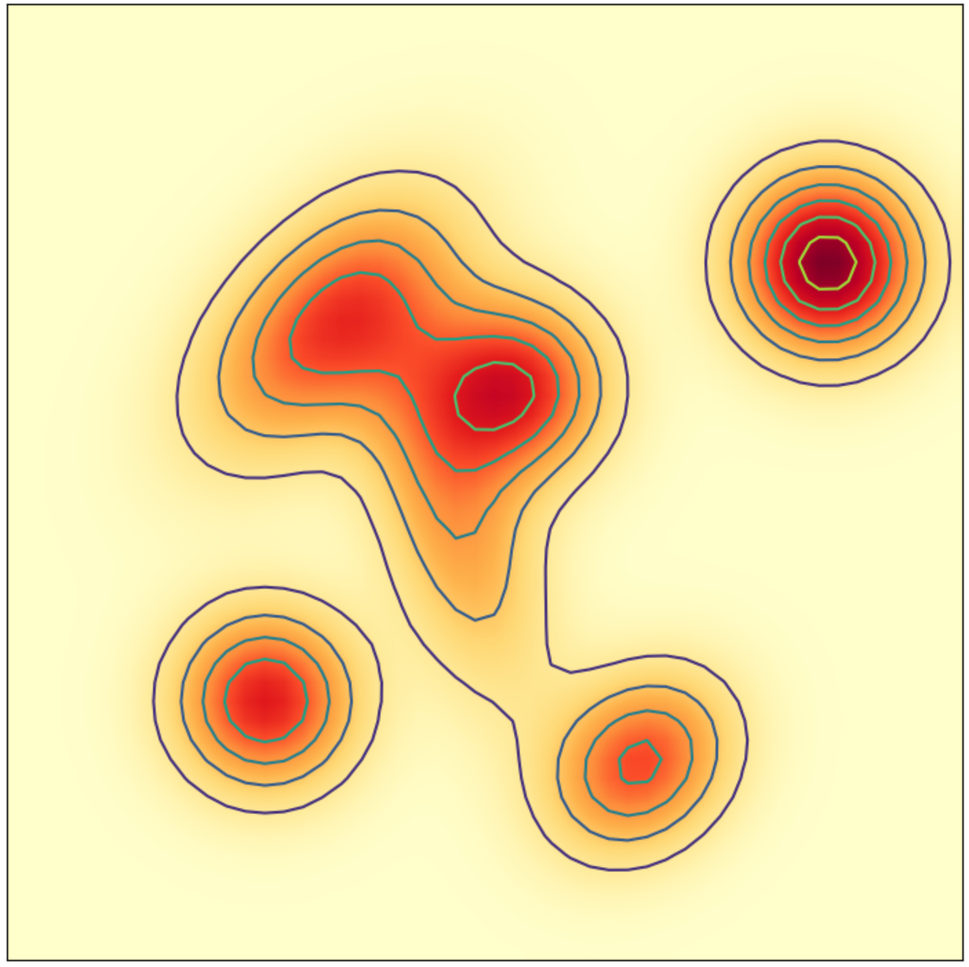
Score Functions = Gradients of the log-likelihood



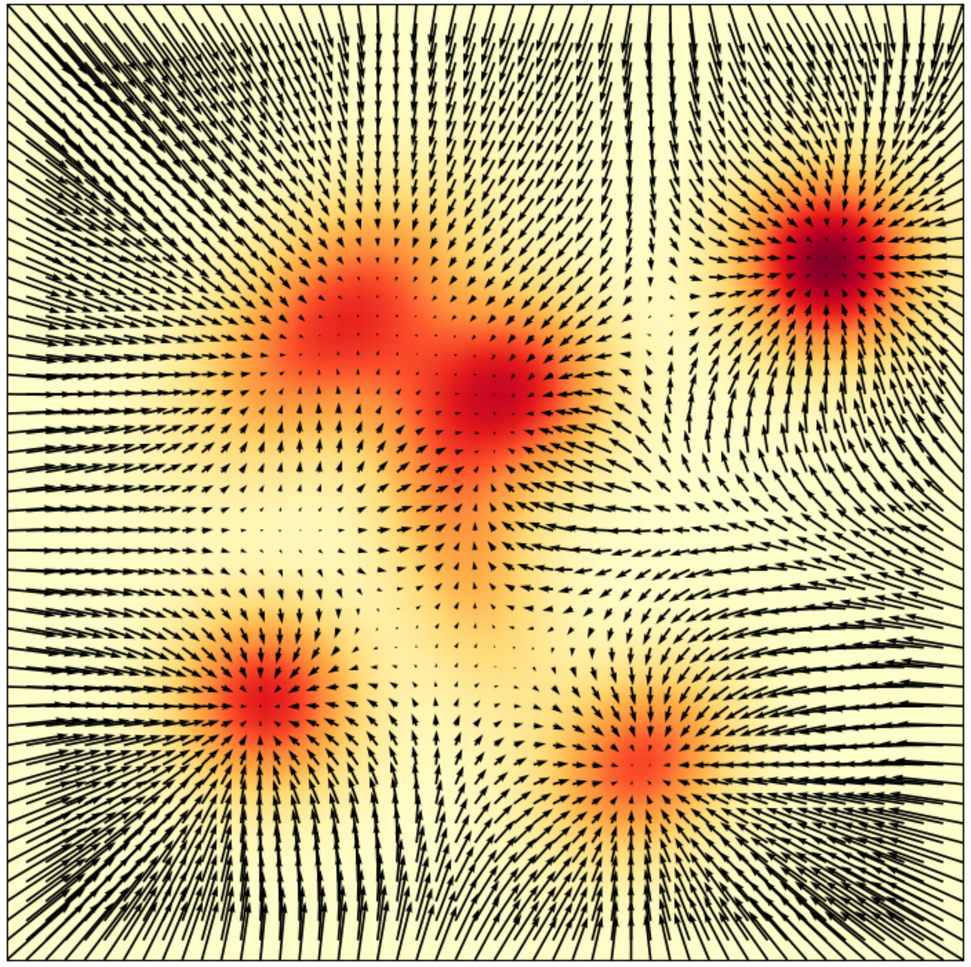


Score Functions = Gradients of the log-likelihood

Log-likelihood: $\log q(x)$



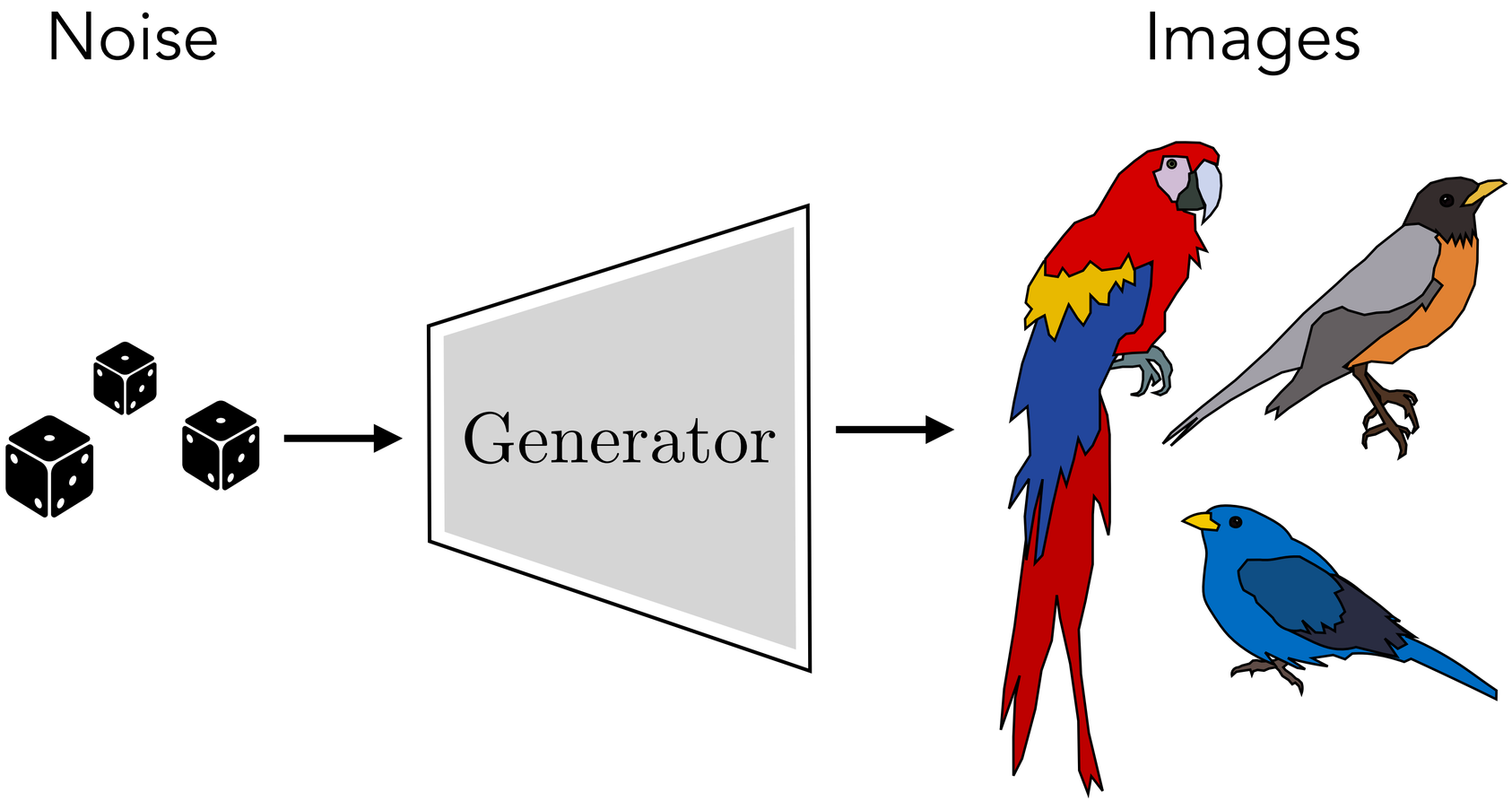
Score function: $\nabla \log q(x)$



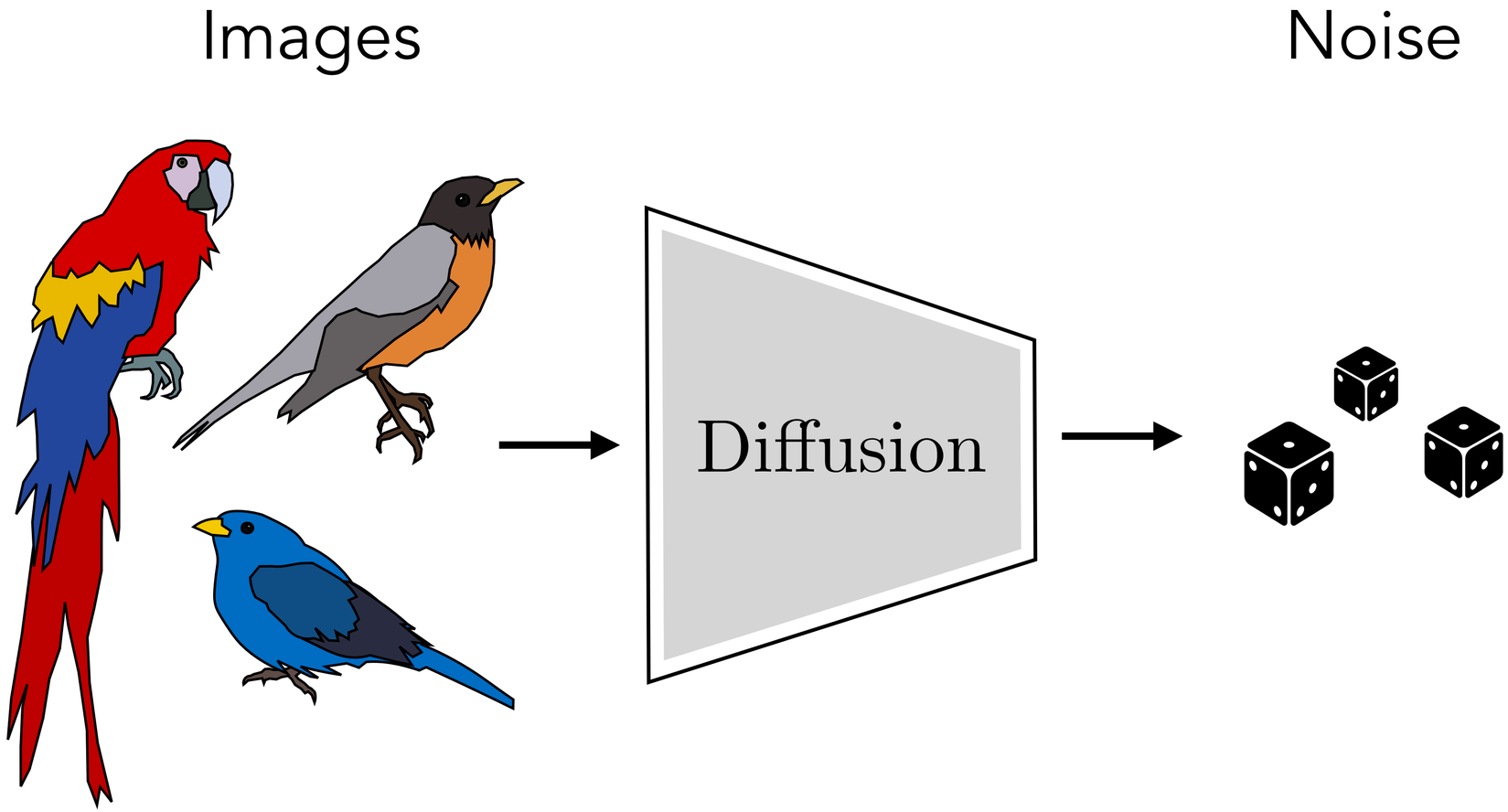
Diffusion Models as Score-Based Generative Modelers

One of many useful perspectives on diffusion models.

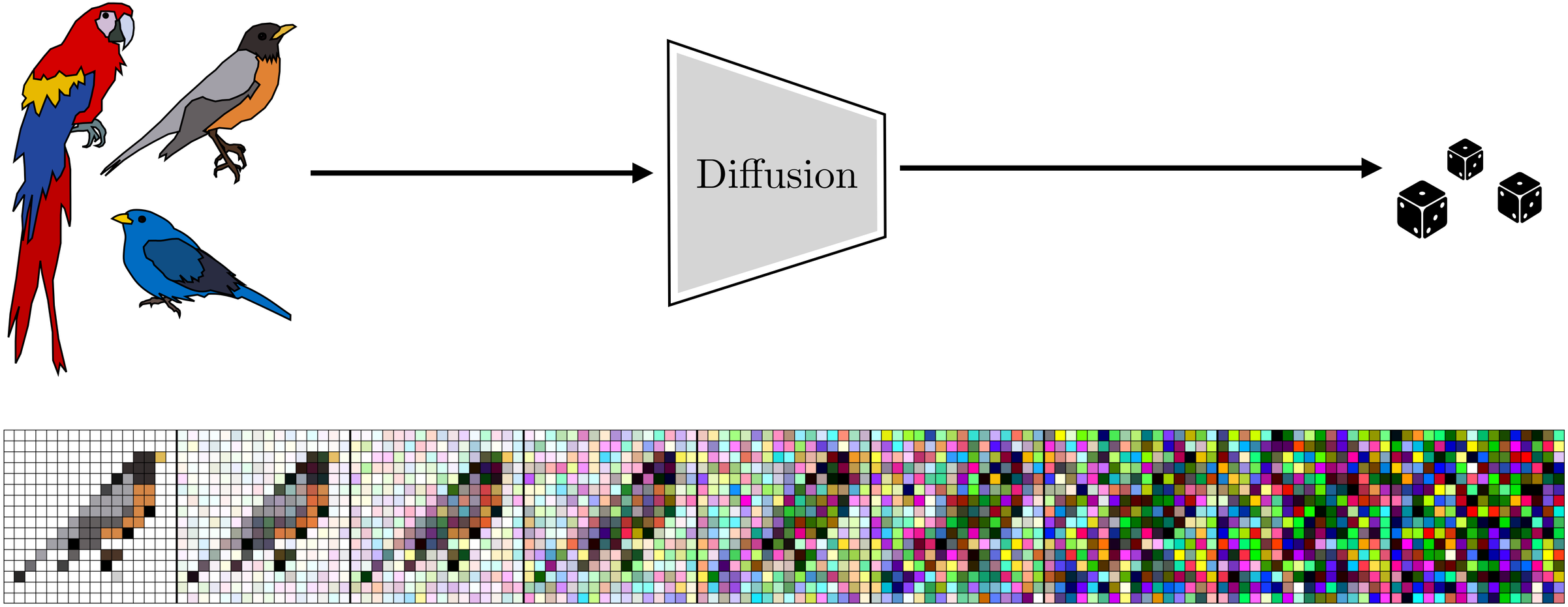
Diffusion Models



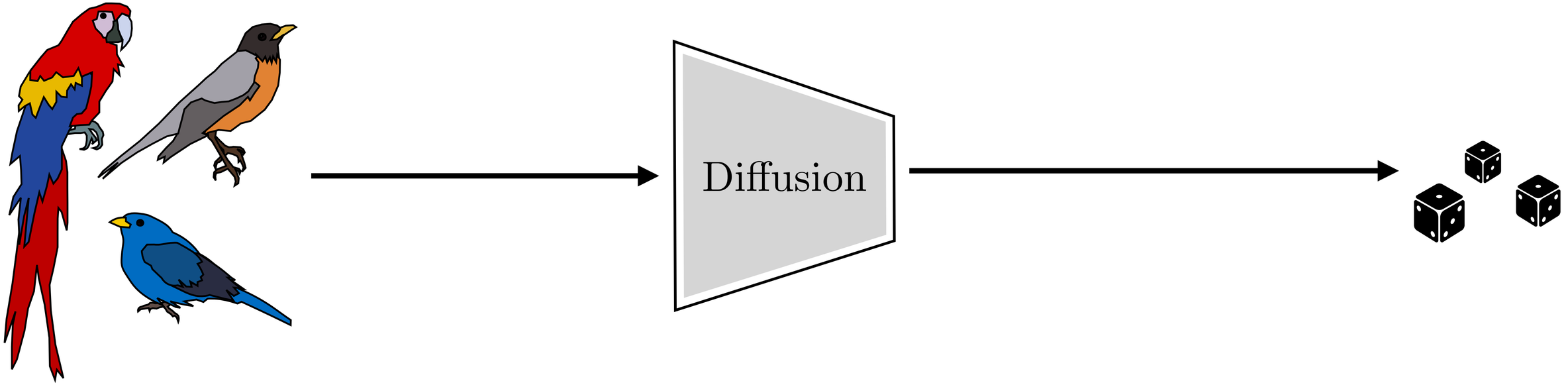
Diffusion Models

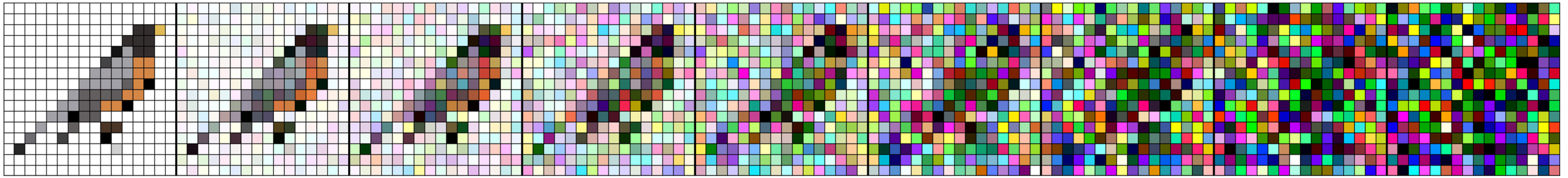


Diffusion Models



Diffusion Models

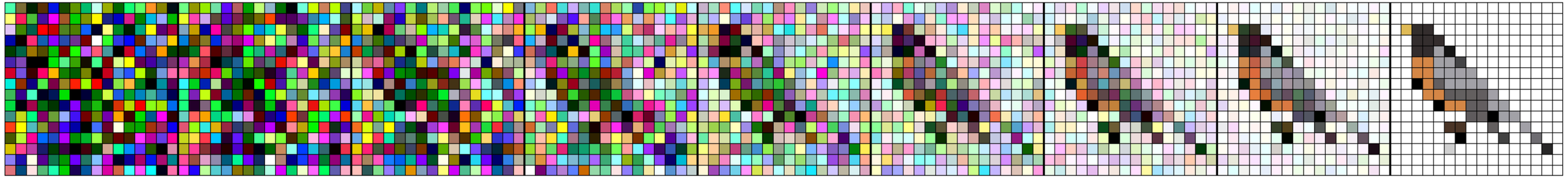




Diffusion: Just add noise

Diffusion Models

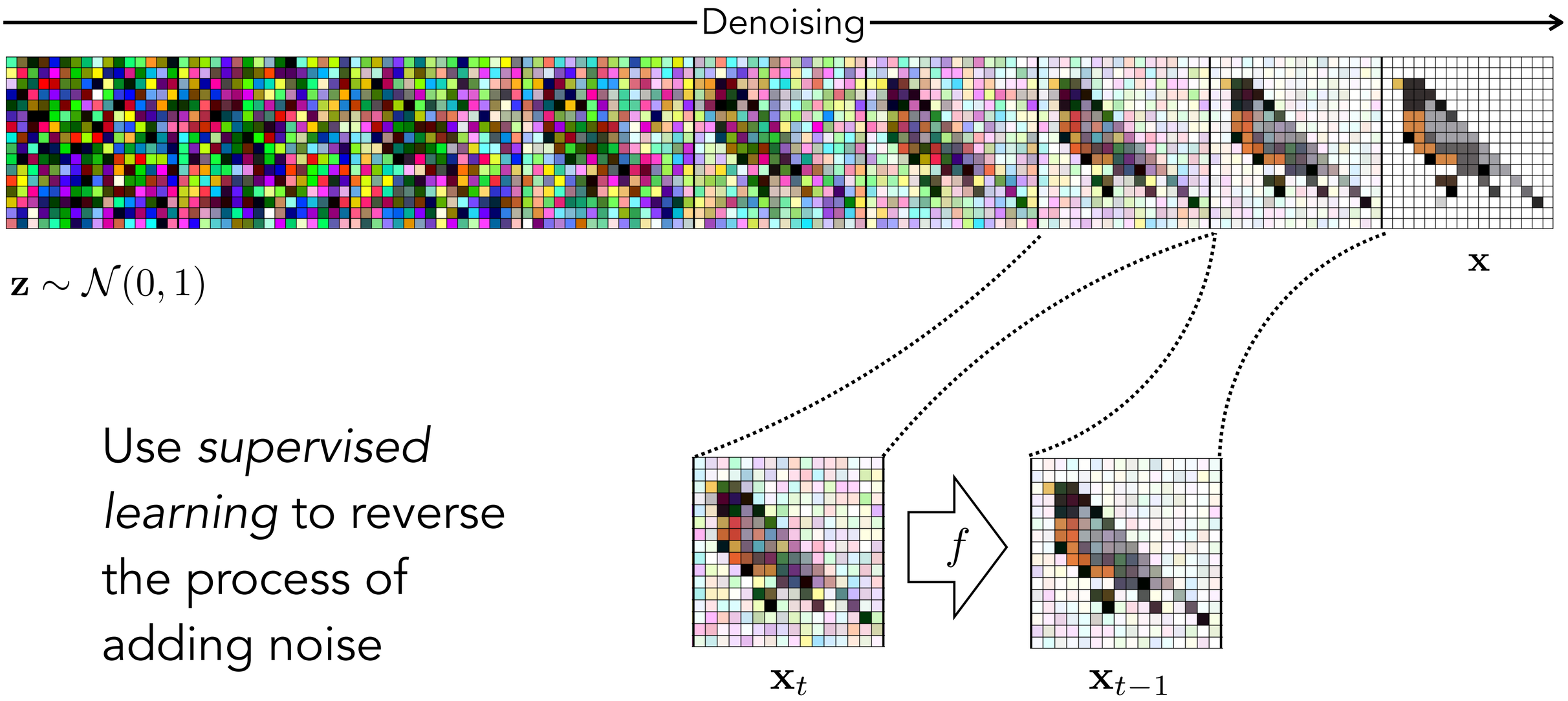
Denoising



$\mathbf{x}$

$\mathbf{z} \sim \mathcal{N}(0, 1)$

Diffusion Models

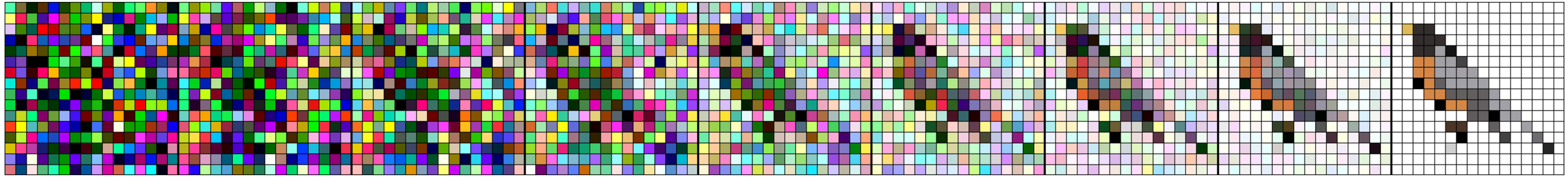


Use *supervised learning* to reverse the process of adding noise

Diffusion Models

$\mathbf{z} \sim \mathcal{N}(0, 1)$

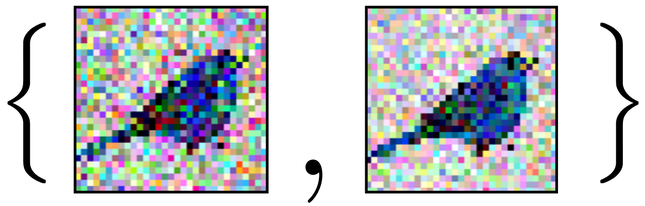
$\mathbf{x}$



Denoising

Training data

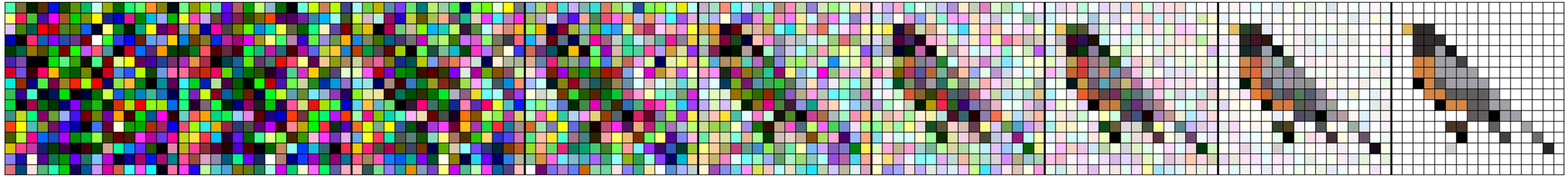
$\mathbf{x}_t$ $\mathbf{x}_{t-1}$



Diffusion Models

$\mathbf{z} \sim \mathcal{N}(0,1)$

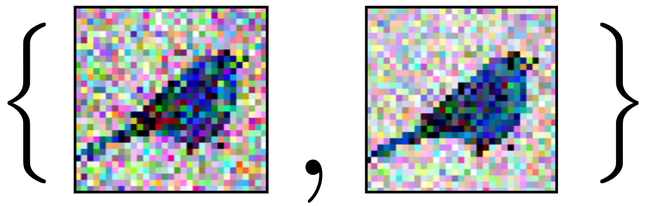
$\mathbf{x}$



Denoising

Training data

$\mathbf{x}_t$ $\mathbf{x}_{t-1}$

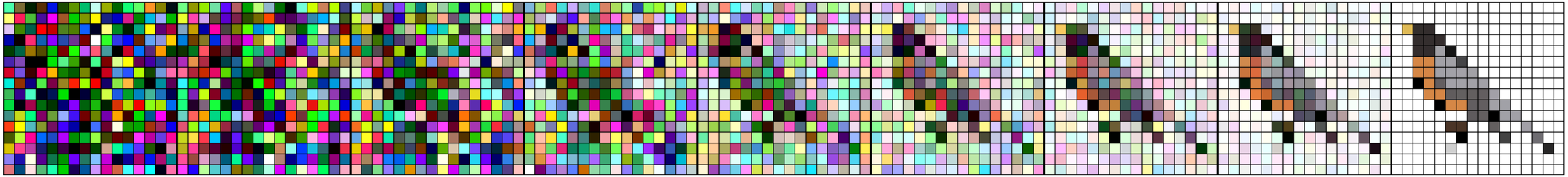


$$\arg\min_{f \in \mathcal{F}} \sum_{i=1}^{N} \mathcal{L}(f(\mathbf{x}_t), \mathbf{x}_{t-1})$$

Diffusion Models

$\mathbf{z} \sim \mathcal{N}(0, 1)$

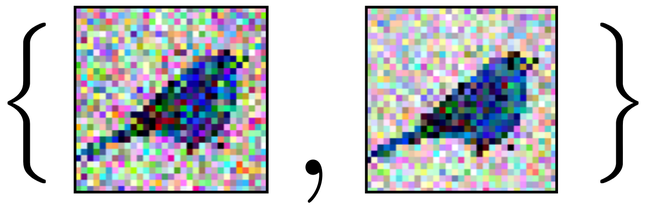
$\mathbf{x}$



Denoising

Training data

$\mathbf{x}_t$ $\mathbf{x}_{t-1}$



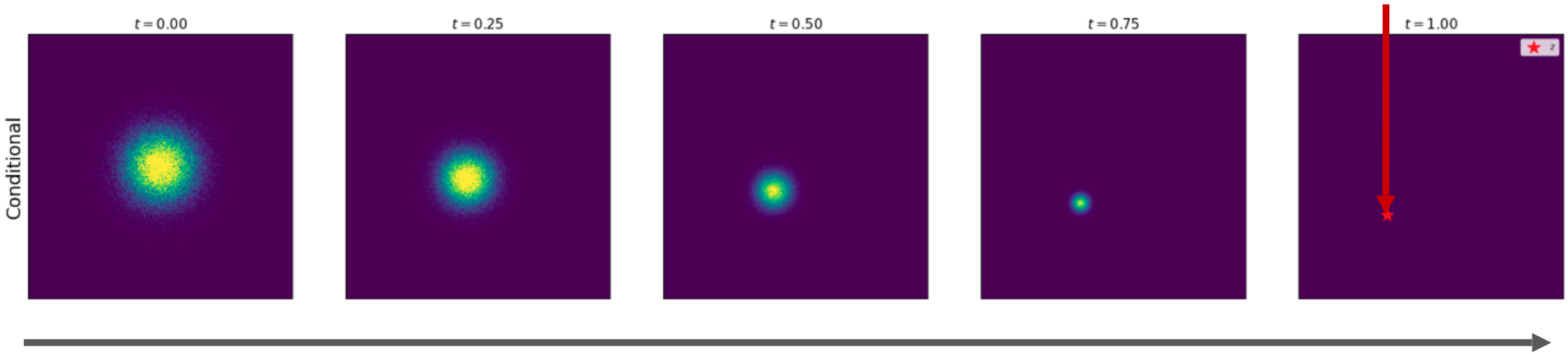
$$\arg\min_{f \in \mathcal{F}} \sum_{i=1}^{N} \mathcal{L}(f(\mathbf{x}_t), \mathbf{x}_{t-1})$$

Converts generative modeling into a series of supervised prediction problems

Conditional Probability Path $p_t(\cdot|z)$

p_{init}

z



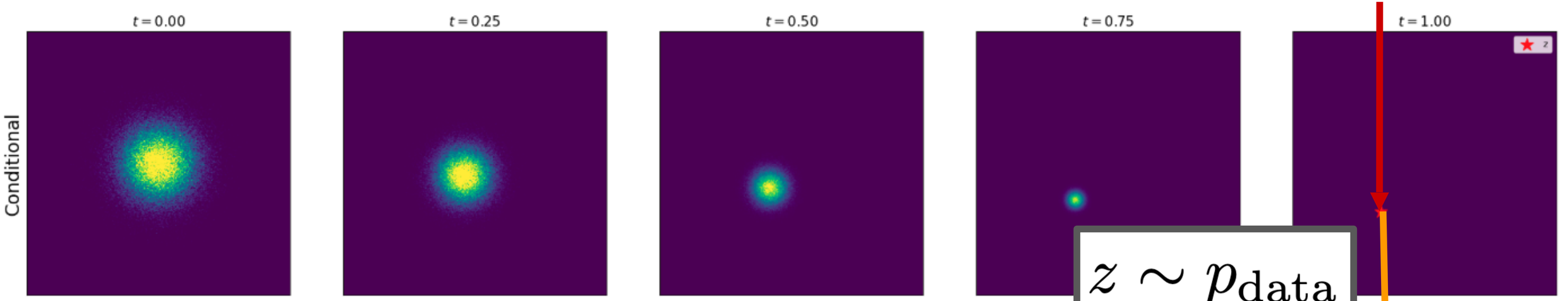
t=0

t=1

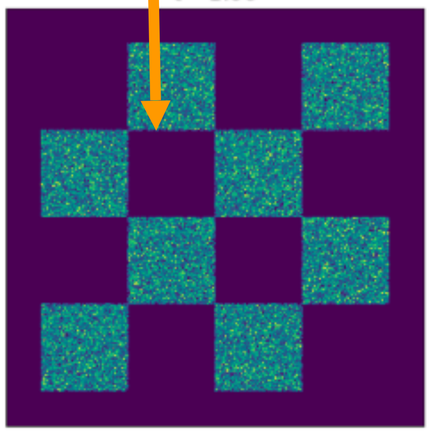
p_{init}

Conditional Probability Path $p_t(\cdot|z)$

z



$z \sim p_{\text{data}}$

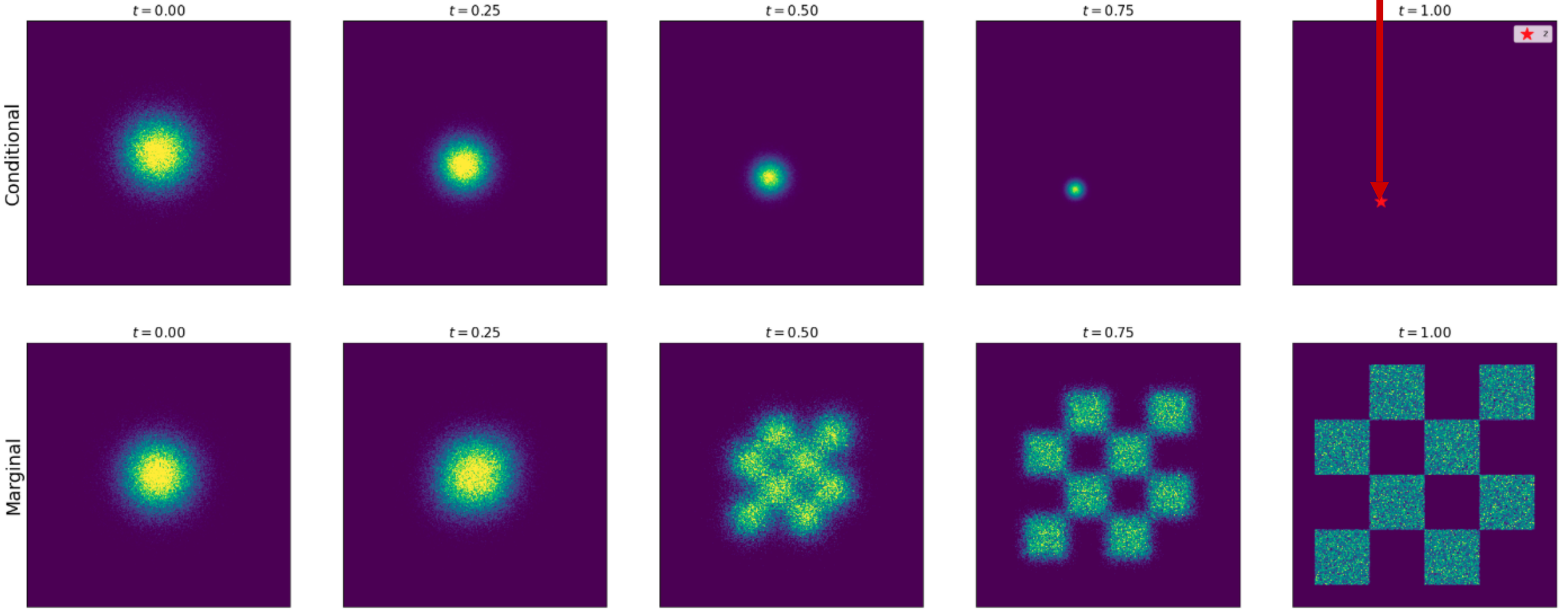


p_{data}

p_{init}

Conditional Probability Path $p_t(\cdot|z)$

z

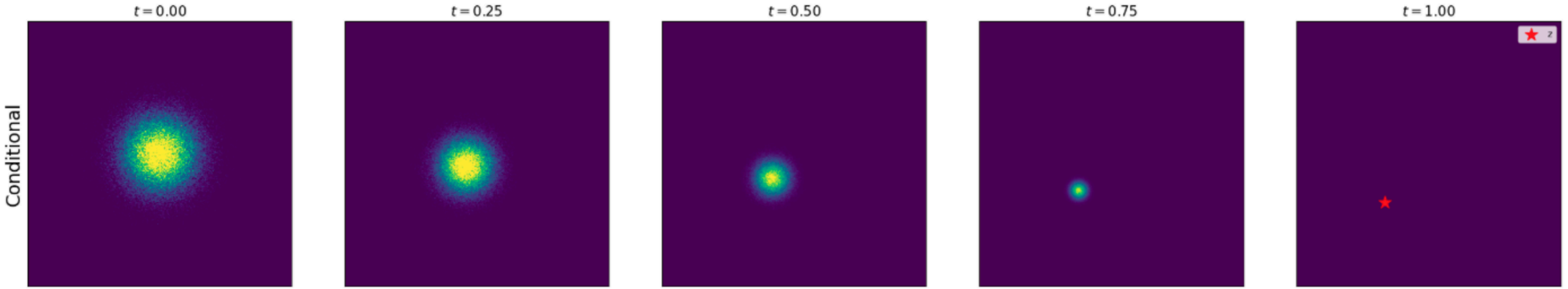


p_{init}

Marginal Probability Path p_t

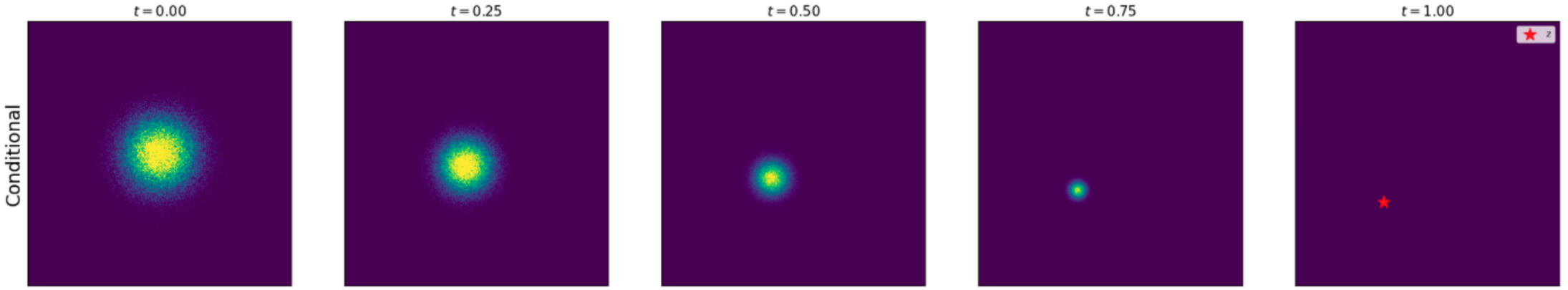
p_{data}

What is different about diffusion of, say, 2D points and images?

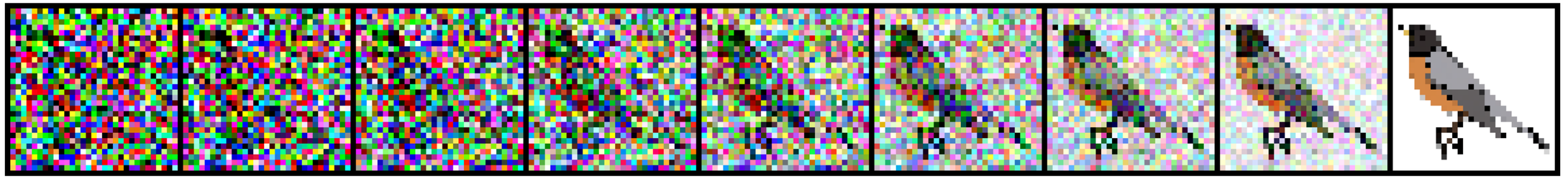


Different geometry!

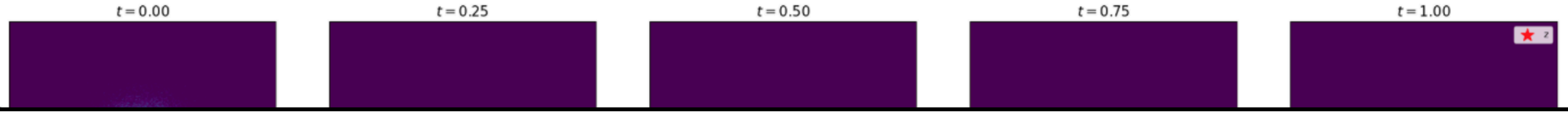
What is different about diffusion of, say, 2D points and images?



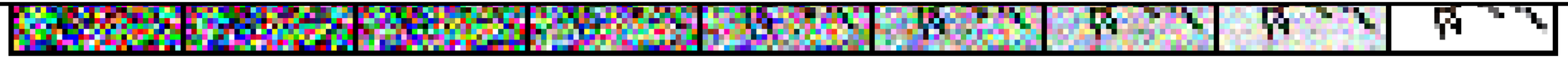
Different geometry!



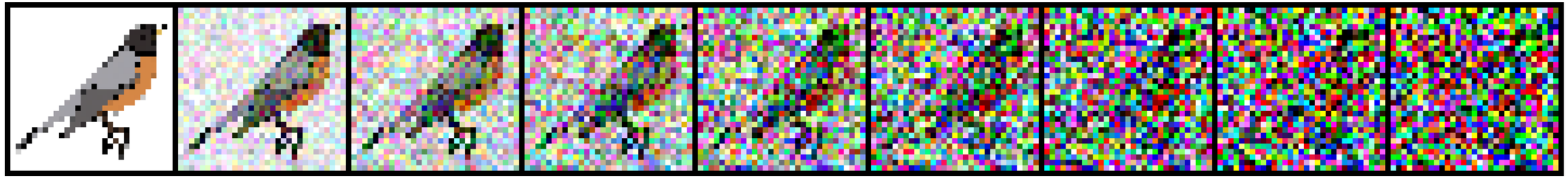
What is different about diffusion of, say, 2D points and images?



Let’s bring some of the tools for image processing to bear that we have learned about so far!

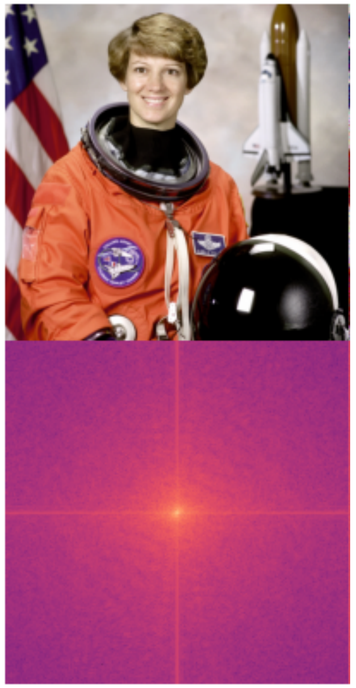


What is different about diffusion of, say,
2D points and images?



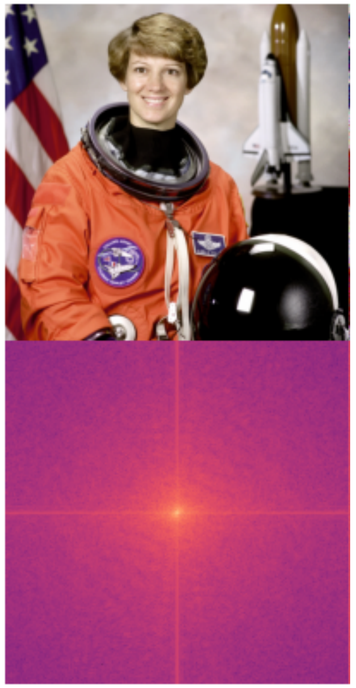
Let’s look at what happens to these images in frequency space!

Forward Diffusion in Frequency Space

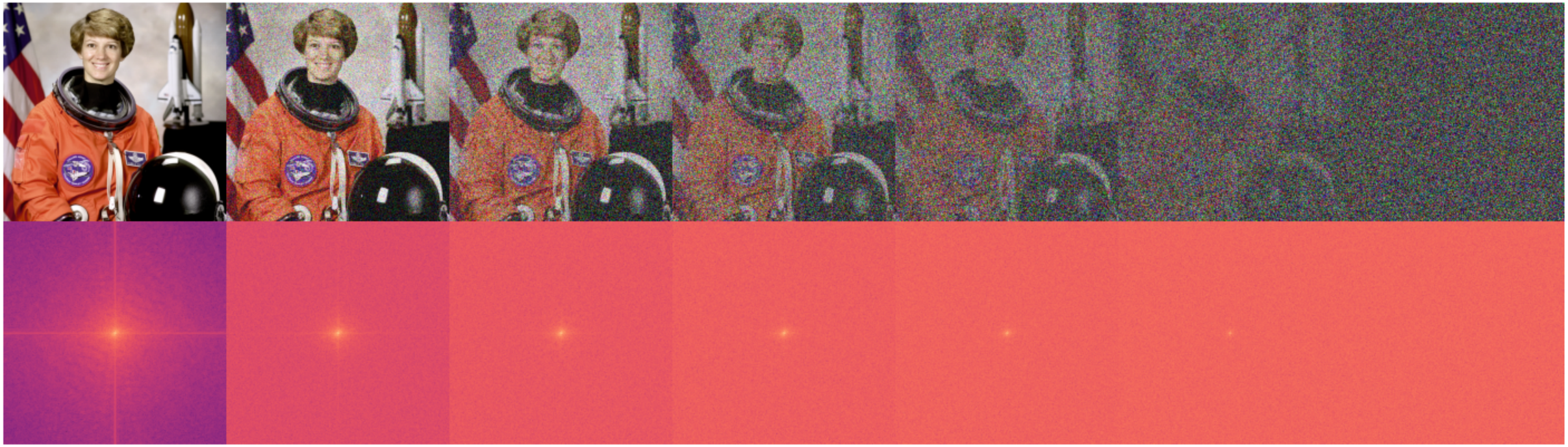


Forward Diffusion in Frequency Space

Diffusion

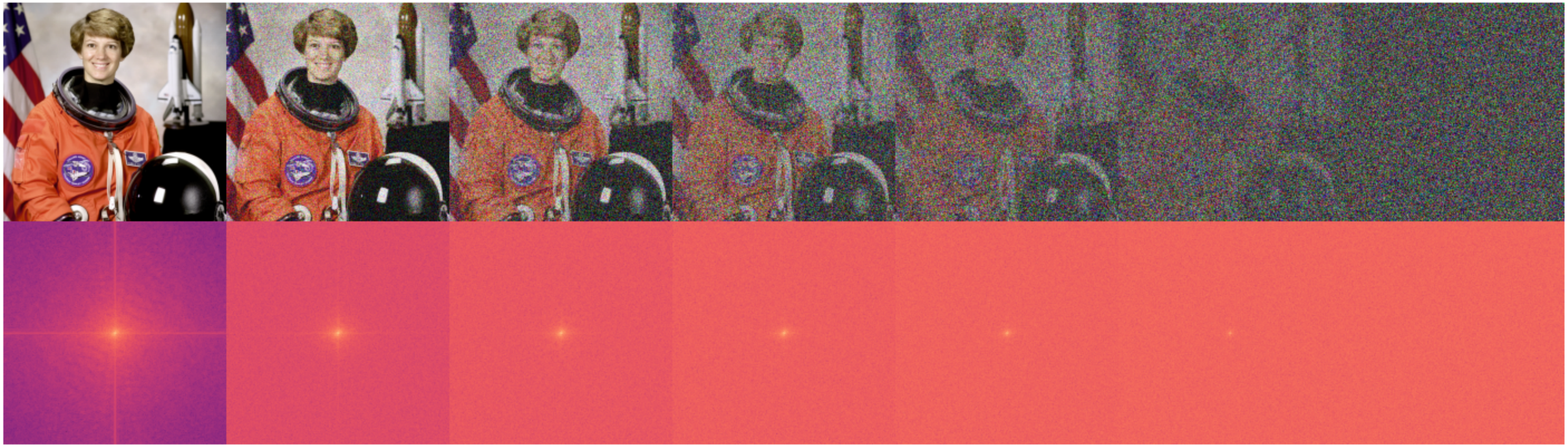


Forward Diffusion in Frequency Space

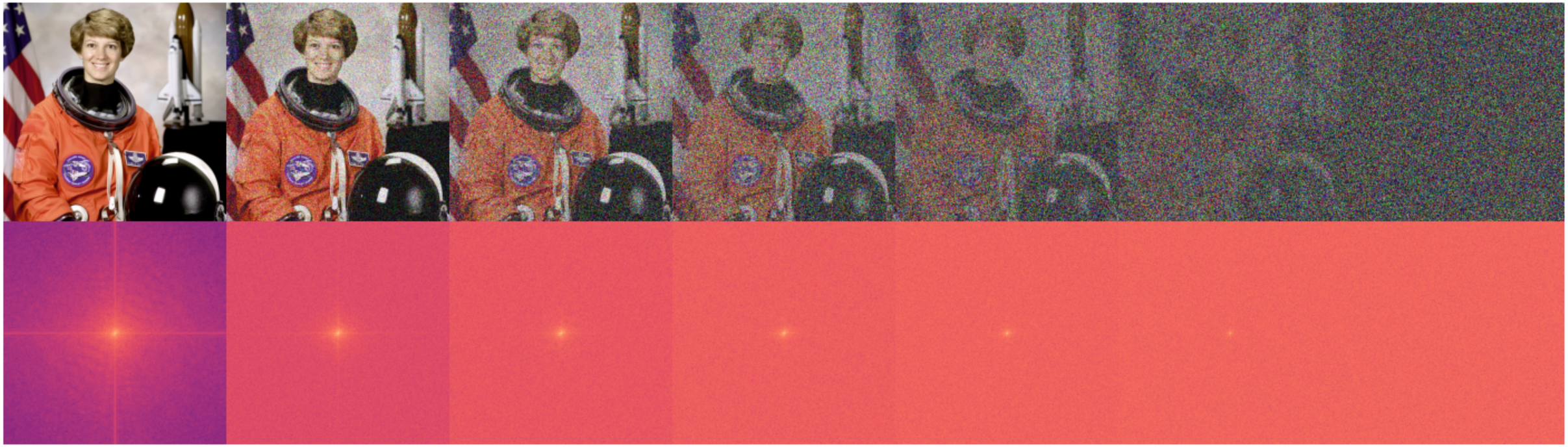


Forward Diffusion in Frequency Space

Diffusion



Forward Diffusion in Frequency Space

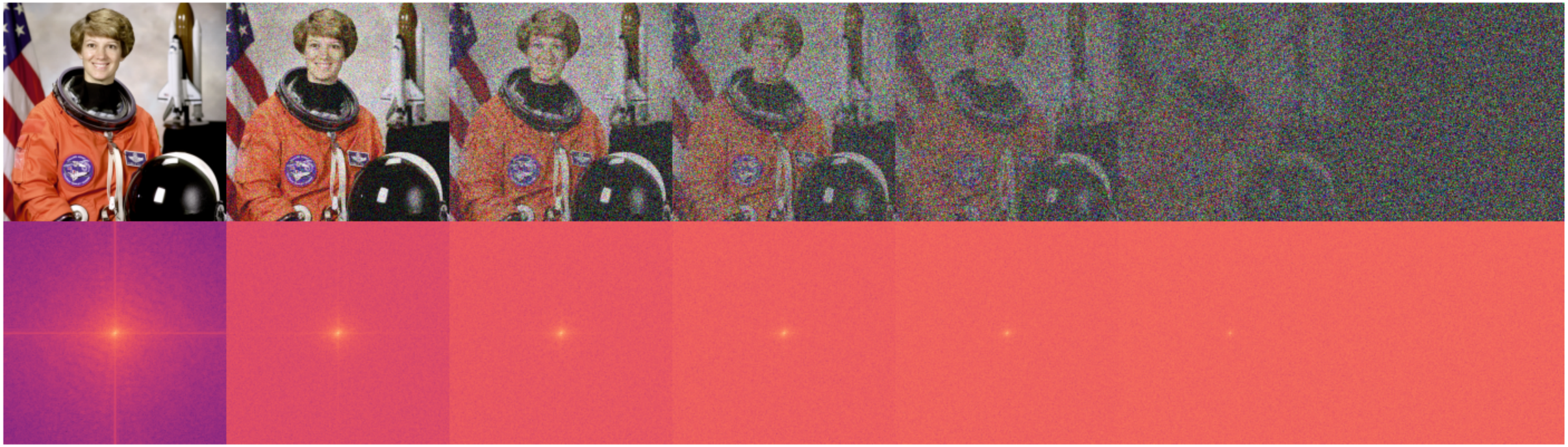


What do we see?

Low frequencies get “drowned out” last!

Forward Diffusion in Frequency Space

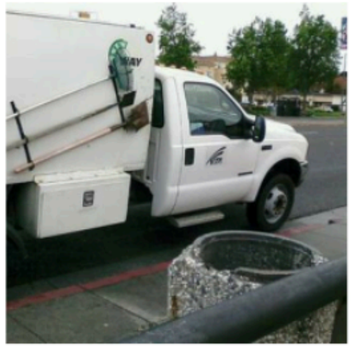
Diffusion →



What do we see?

Low frequencies get “drowned out” last!

The spectral density of natural images

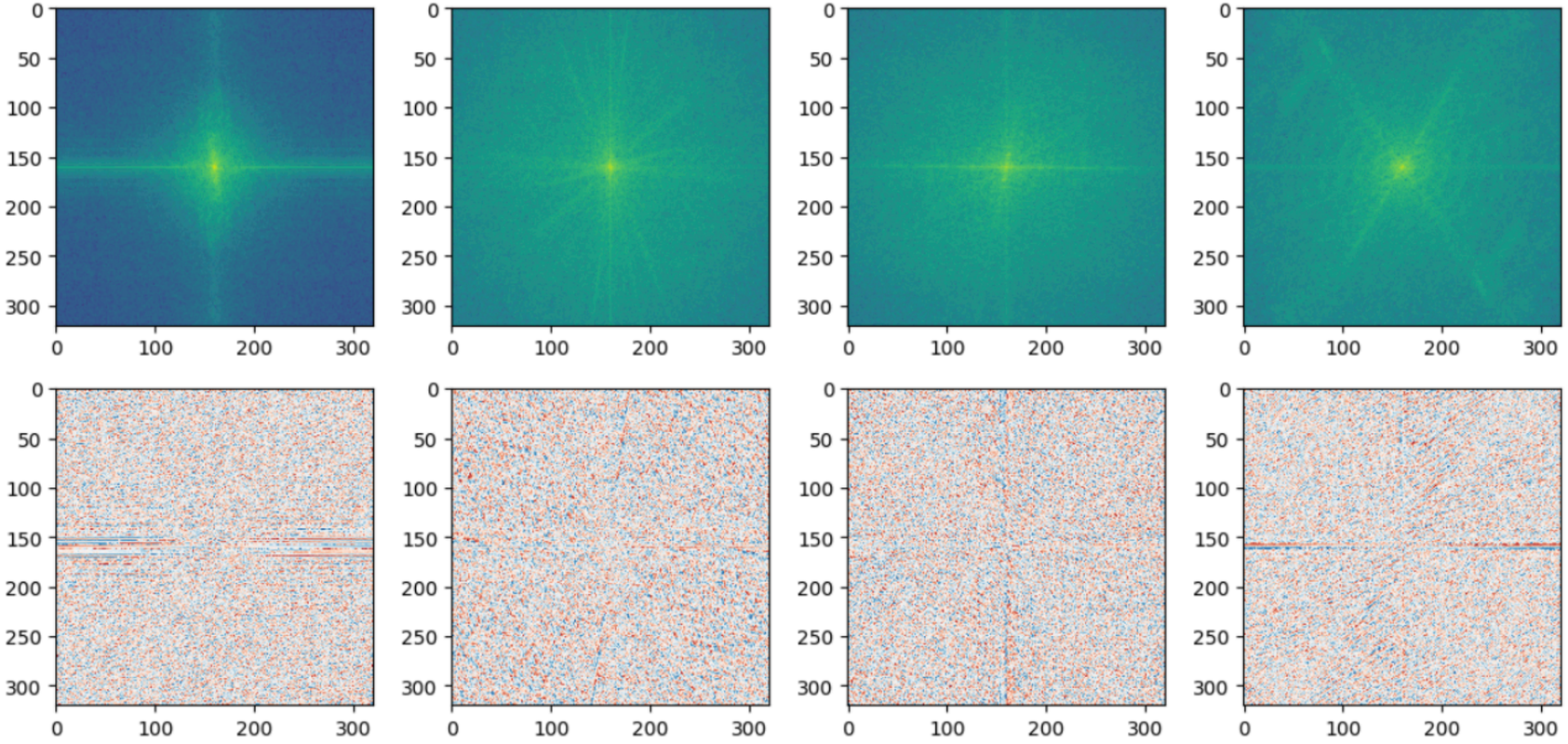


Illustration from Sander Dieleman, “Diffusion is spectral autoregression”

```
spec = np.fft.fft2(image)
spec = np.fft.fftshift(spec)
spdj = np.log(np.abs(spec))
```

The spectral density of natural images

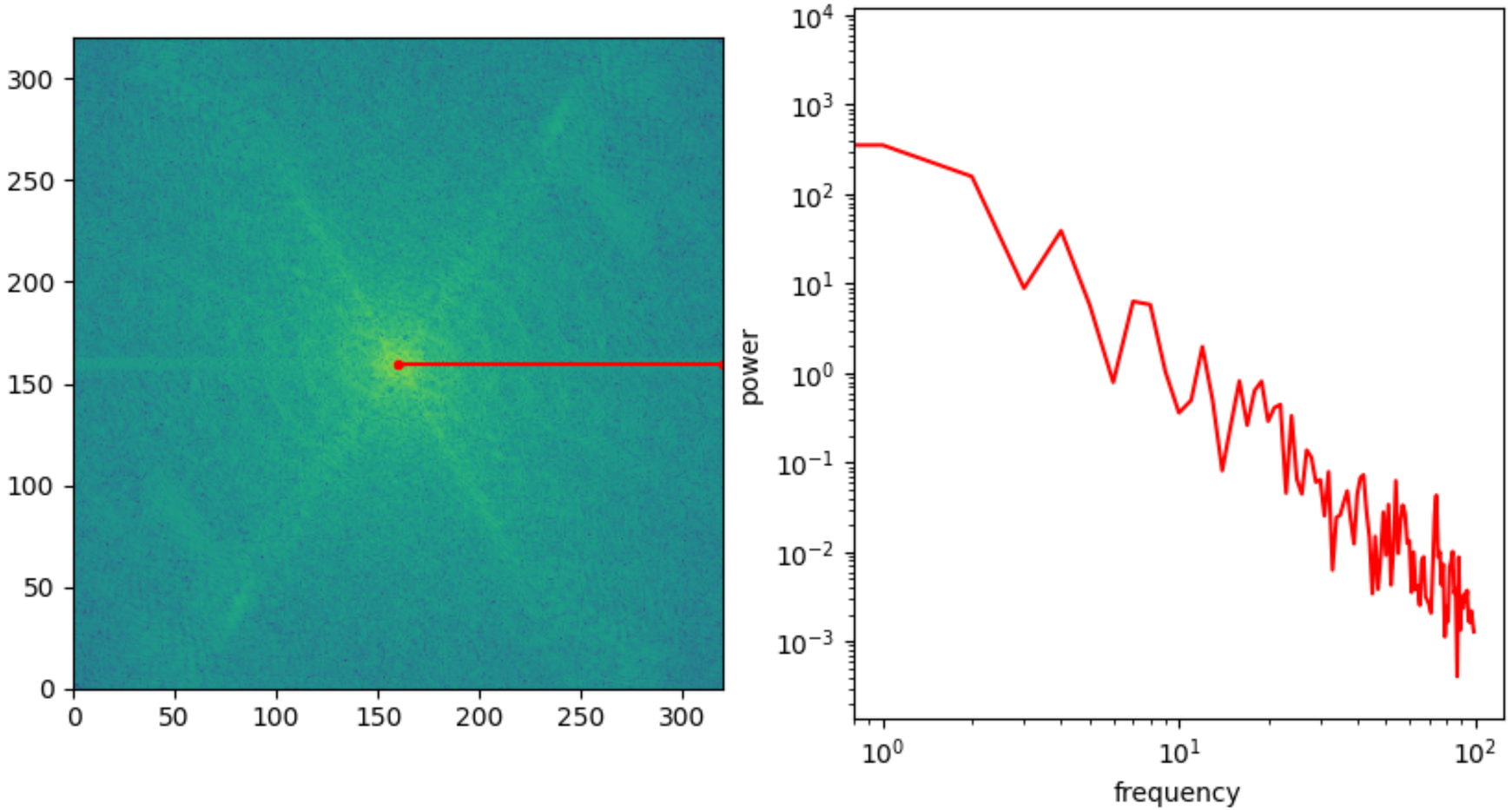


Illustration from Sander Dieleman, “Diffusion is spectral autoregression”

The spectral density of natural images

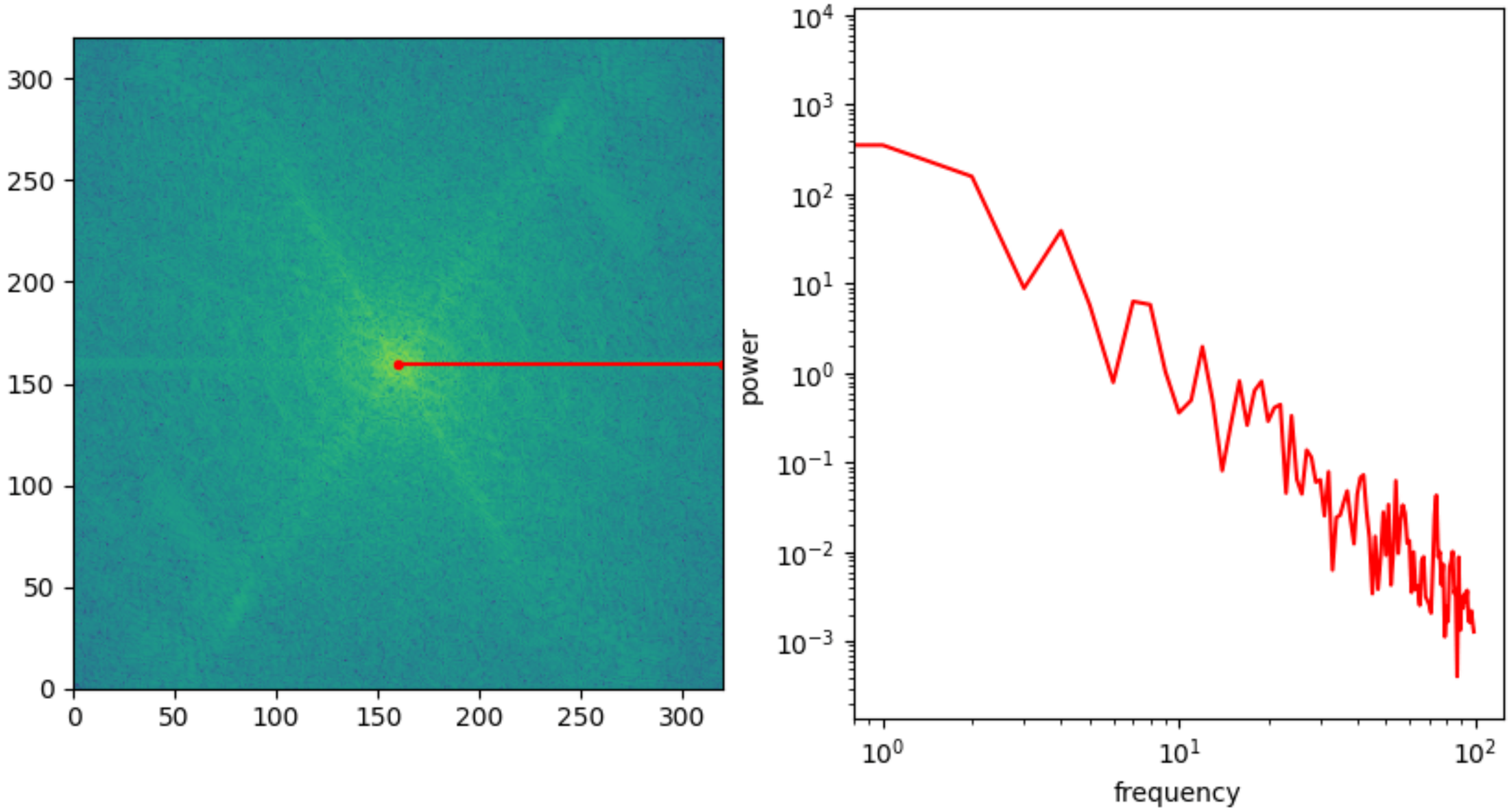


Illustration from Sander Dieleman, “Diffusion is spectral autoregression”

The spectral density of natural images

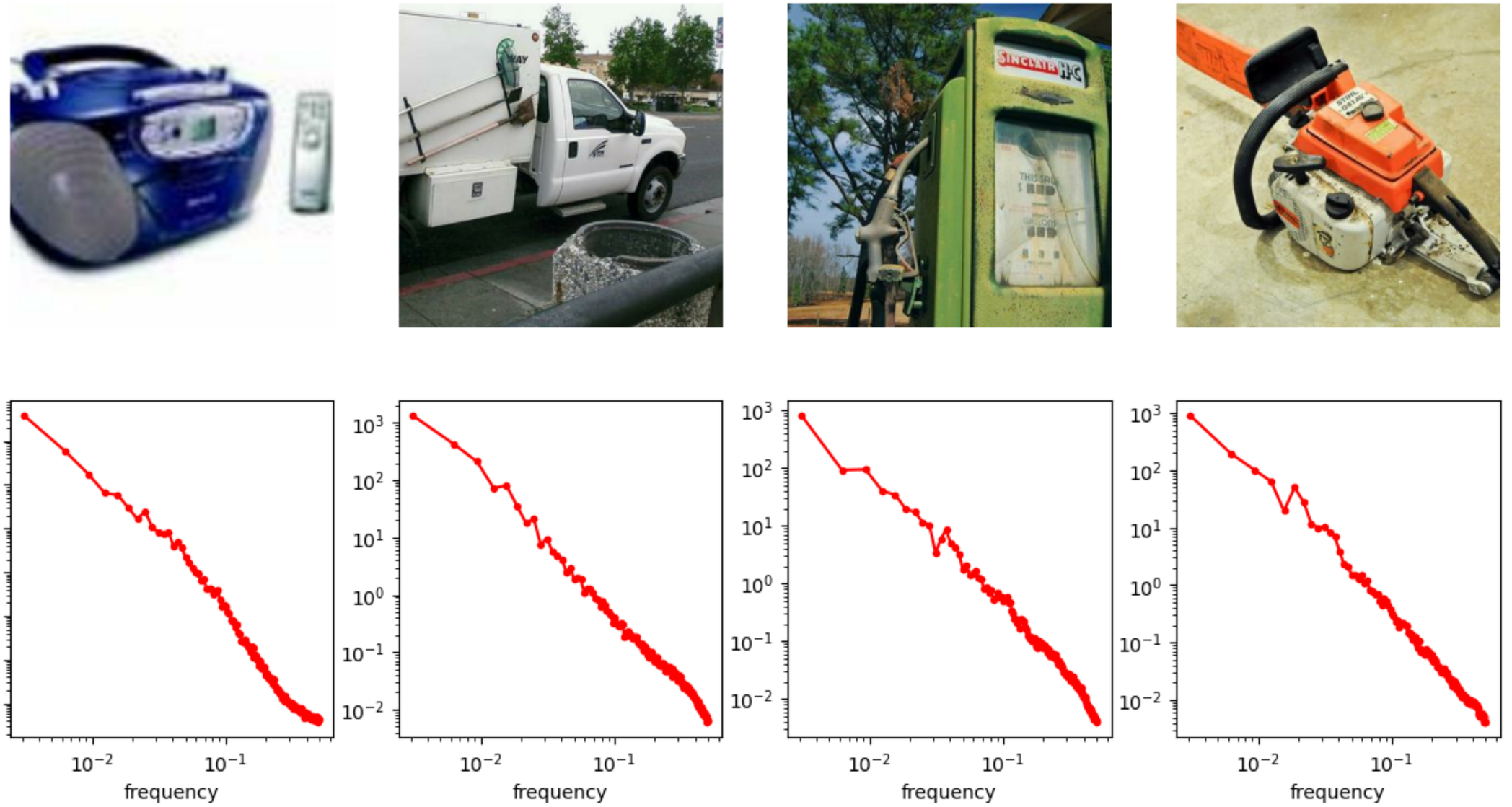


Illustration from Sander Dieleman, “Diffusion is spectral autoregression”

The spectral density of natural images

Insight: Spectral Power of images follows a “power law”, i.e., it’s linear in the log domain.

Note: this is true *on aggregate*, it is easy to find images that behave differently.

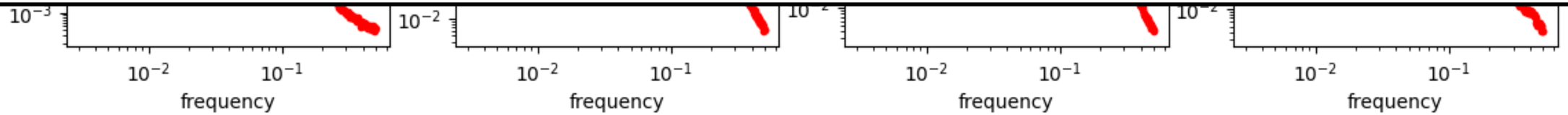


Illustration from Sander Dieleman, “Diffusion is spectral autoregression”

What happens when we add noise?

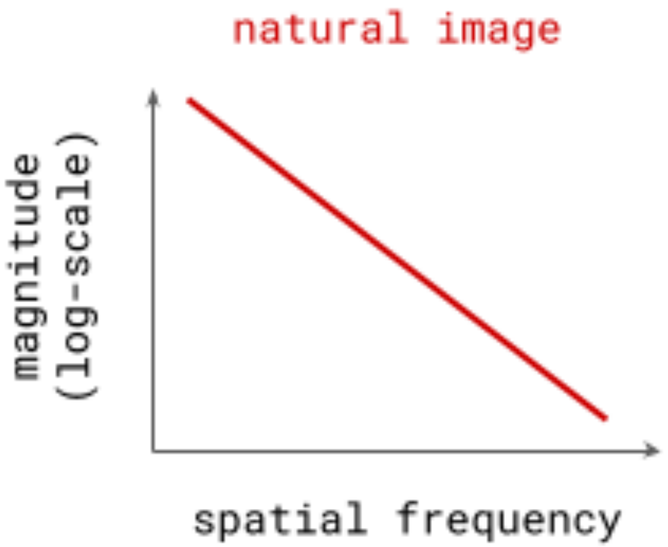


Illustration from Sander Dieleman, “Diffusion is spectral autoregression”

What happens when we add noise?

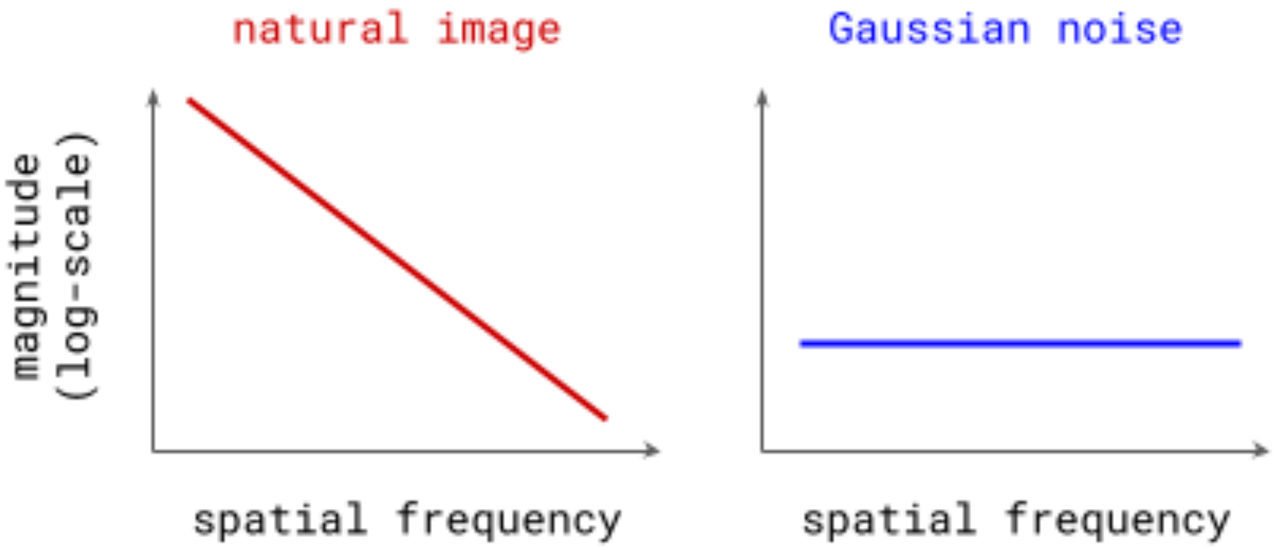


Illustration from Sander Dieleman, “Diffusion is spectral autoregression”

What happens when we add noise?

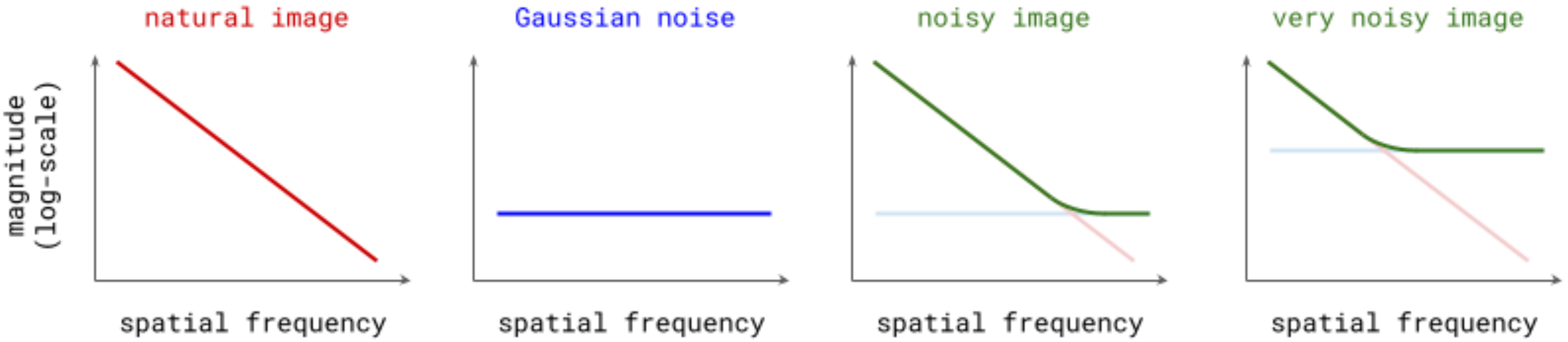
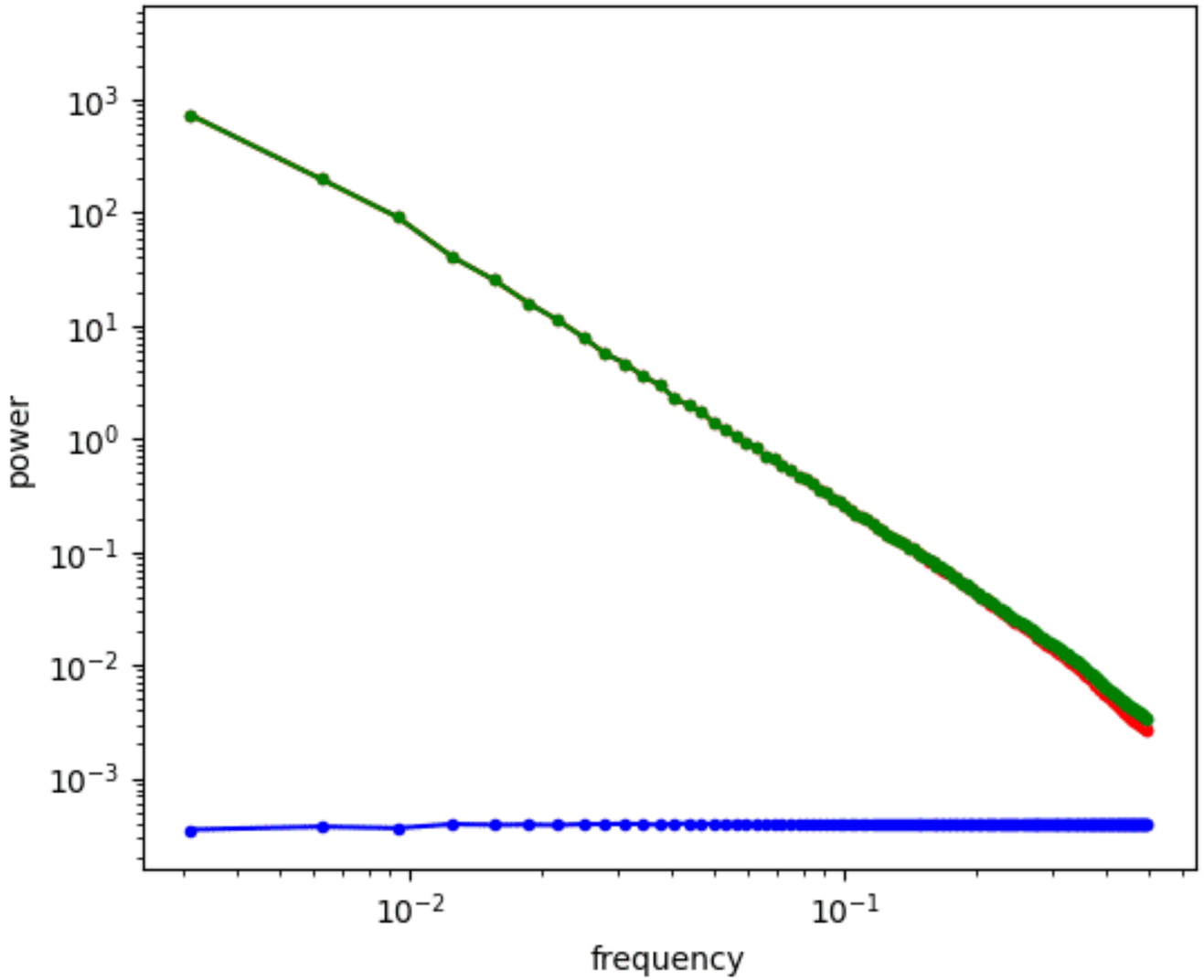
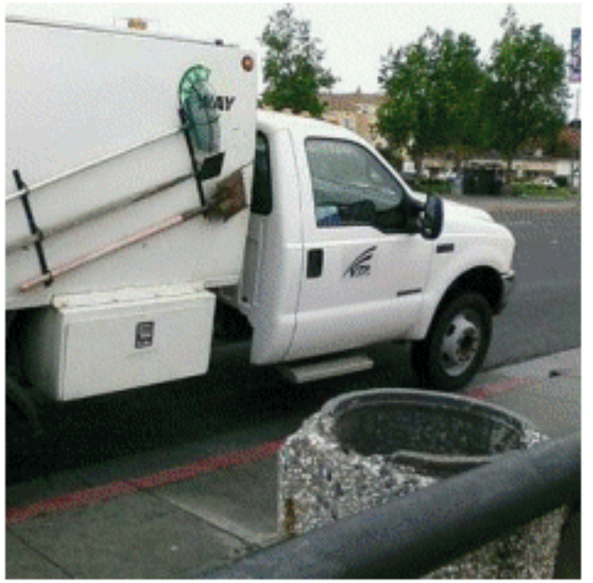


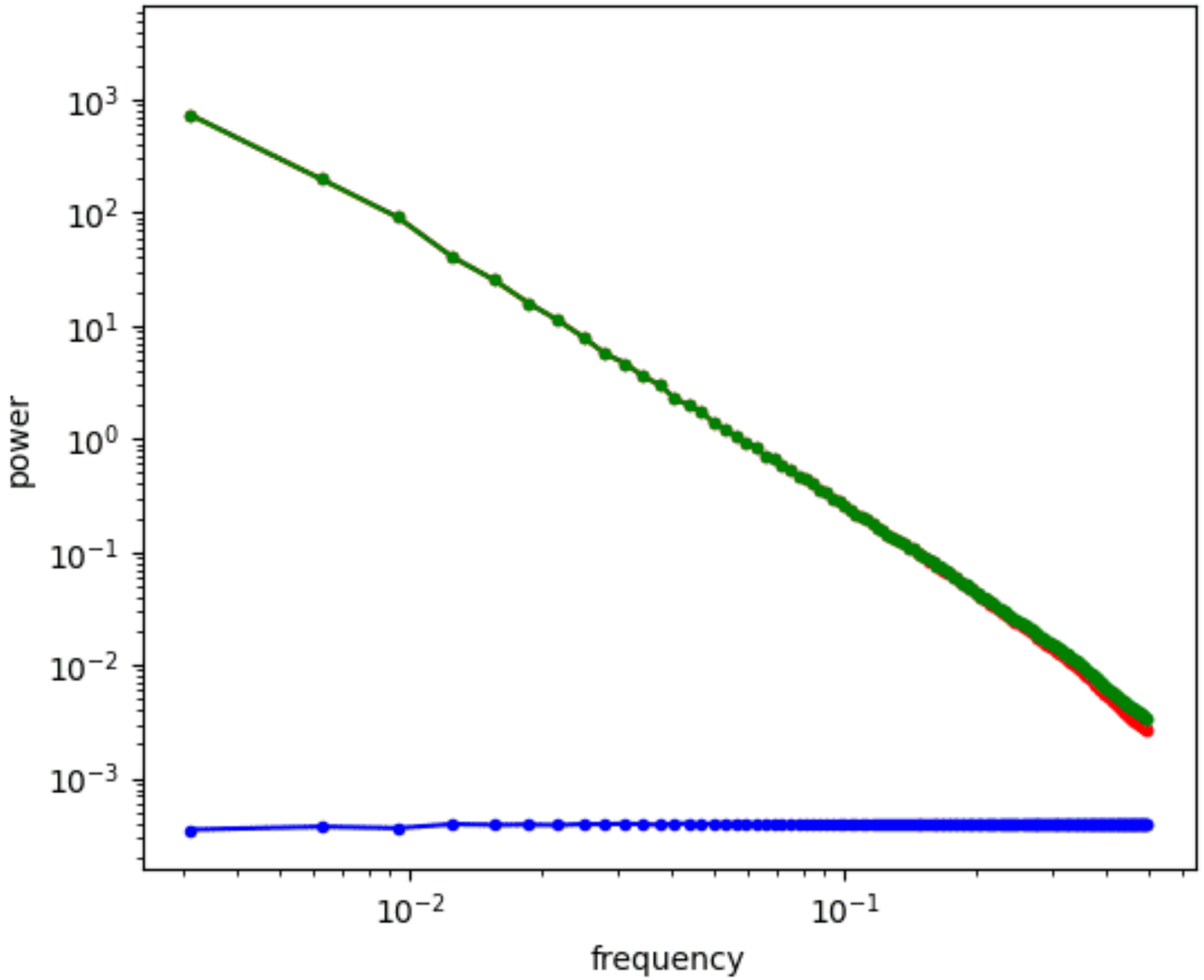
Illustration from Sander Dieleman, “Diffusion is spectral autoregression”

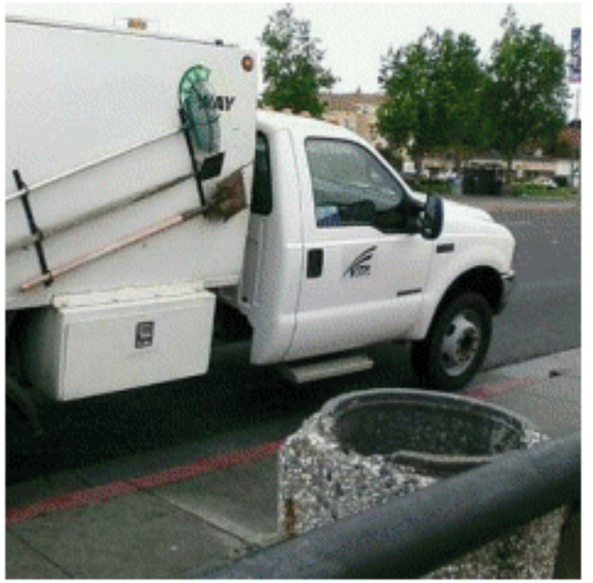
What happens when we add noise?



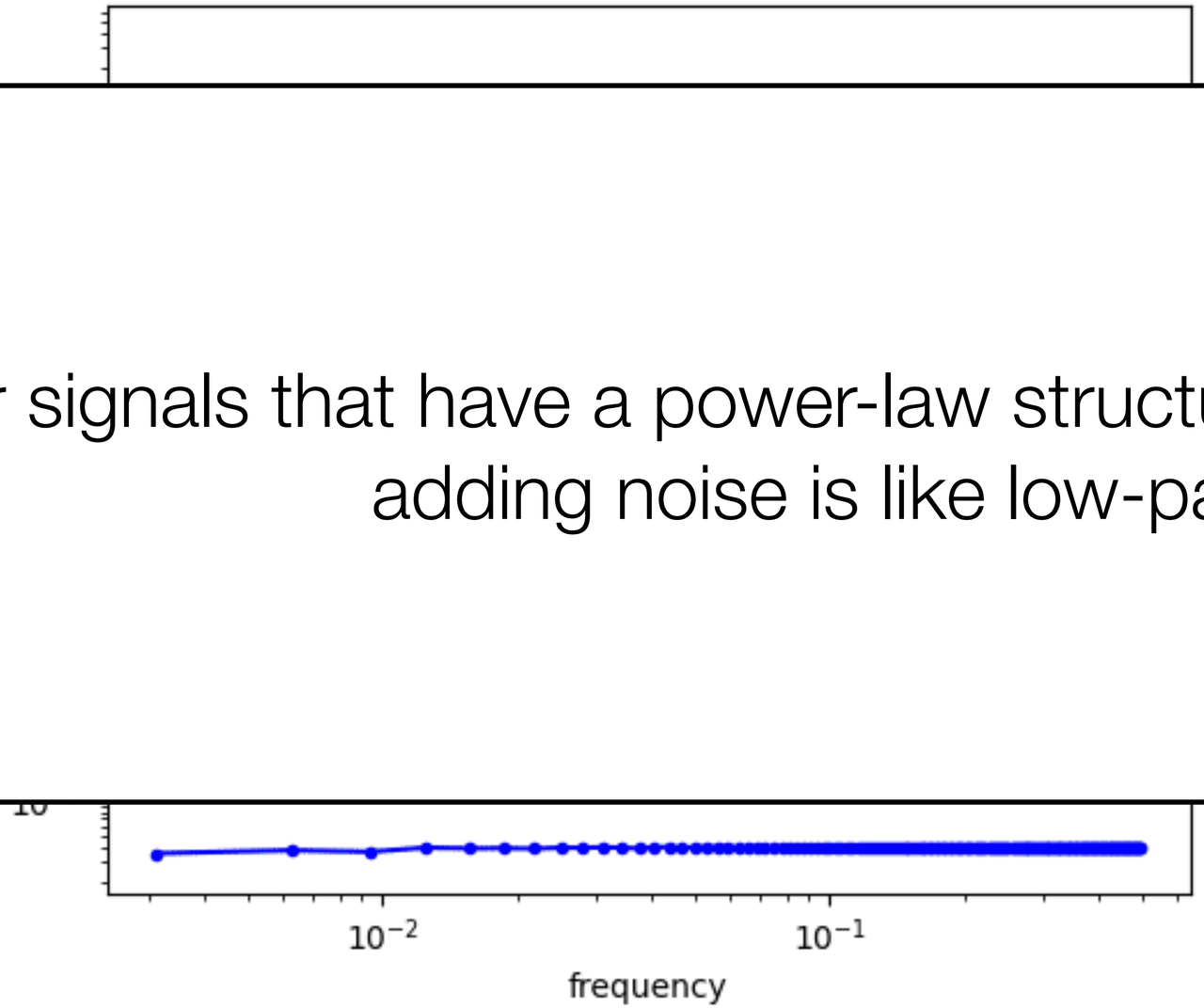


What happens when we add noise?



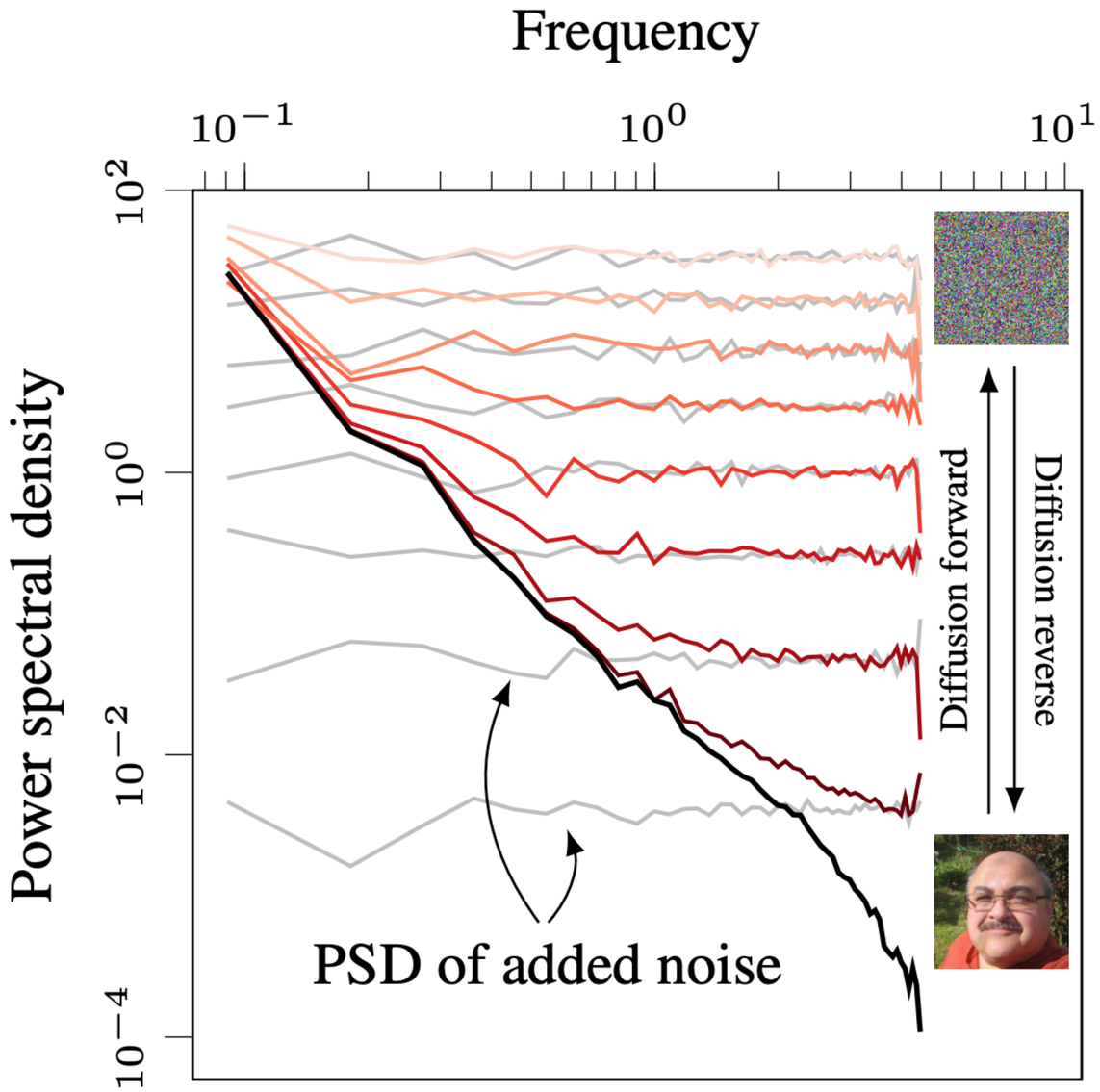


What happens when we add noise?



For signals that have a power-law structure in the spectral density, adding noise is like low-pass filtering!

What is Diffusion Doing?



- Diffusion models can be seen as doing “spectral auto-regression”
- Given lower frequency of image, incrementally predict higher frequencies!

Generative Modeling with Inverse Heat Dissipation, Rissanen et al. 2020

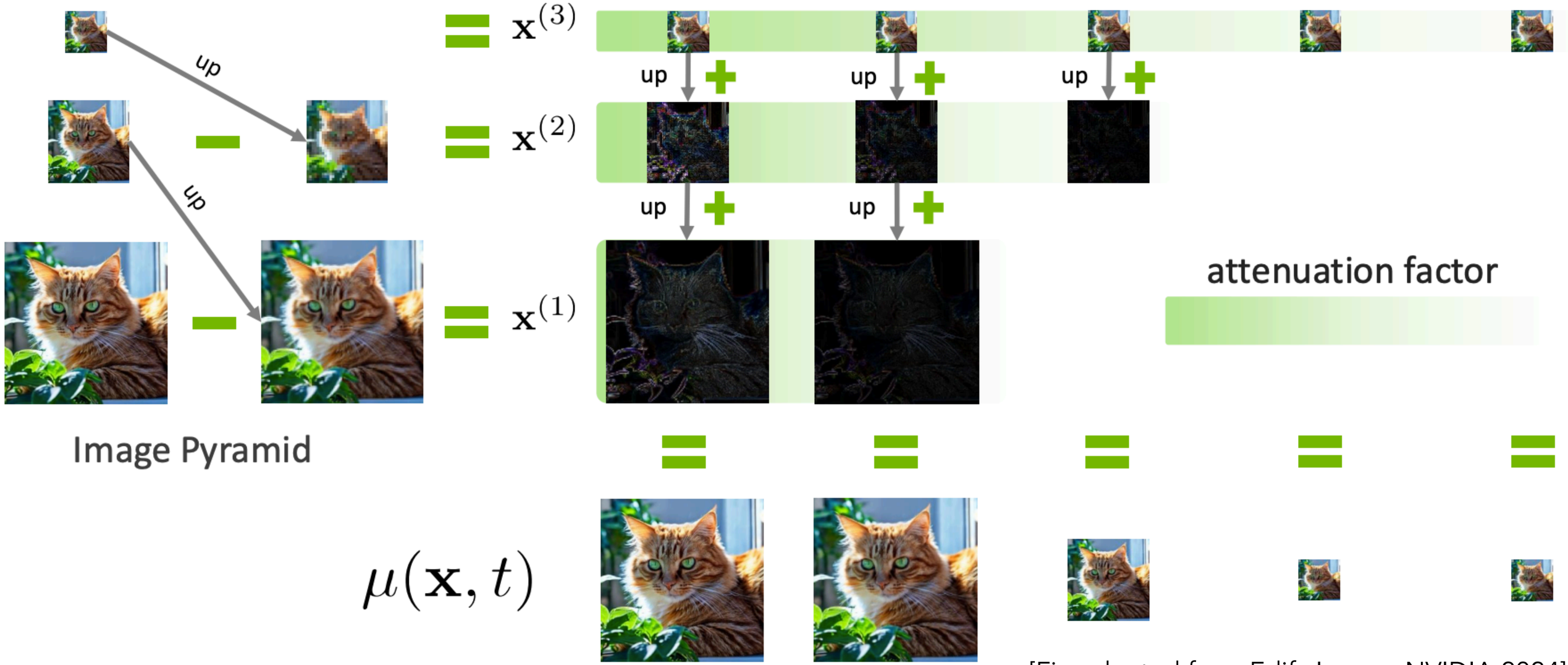
Edify Image: High-Quality Image Generation with Pixel Space Laplacian Diffusion Models

NVIDIA[1]

A photo of a couple doing pottery together in a well-lit room

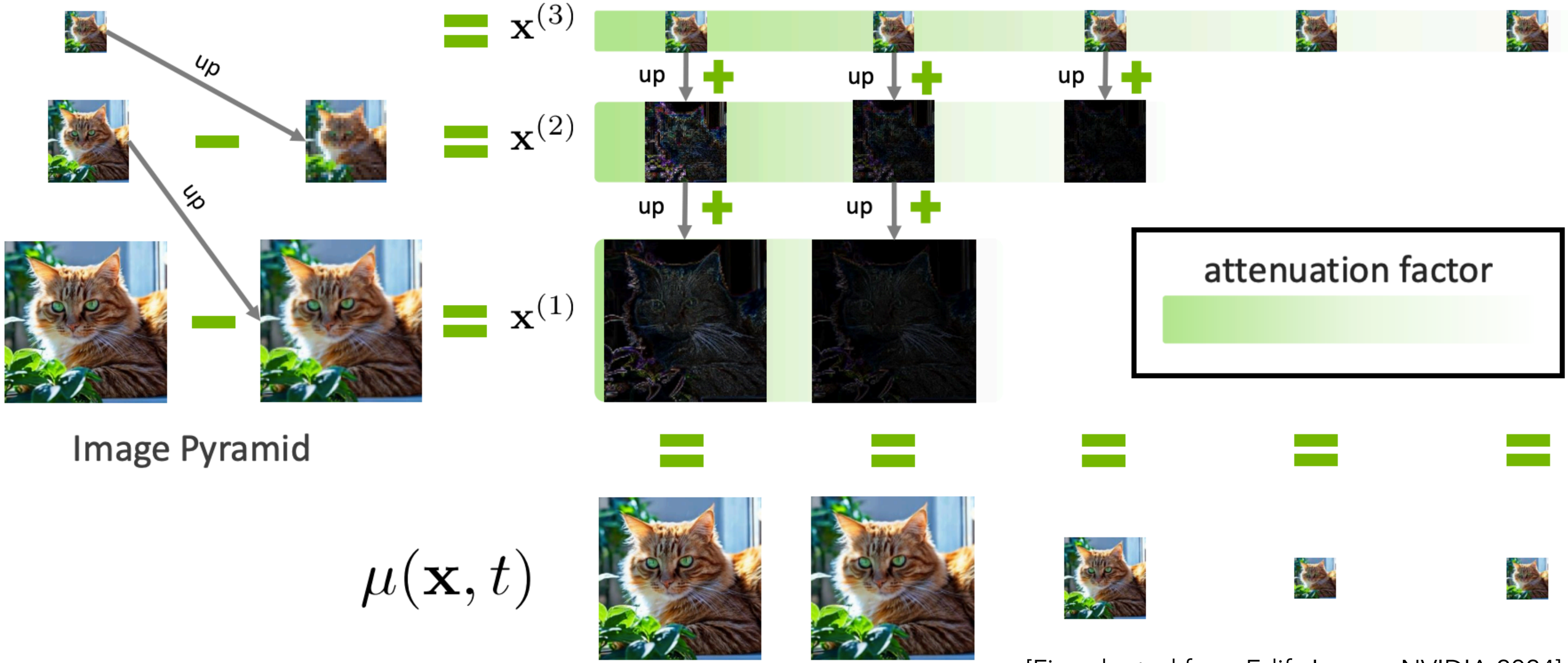
A chameleon showing colorful scales

Foundation: Laplacian Pyramid



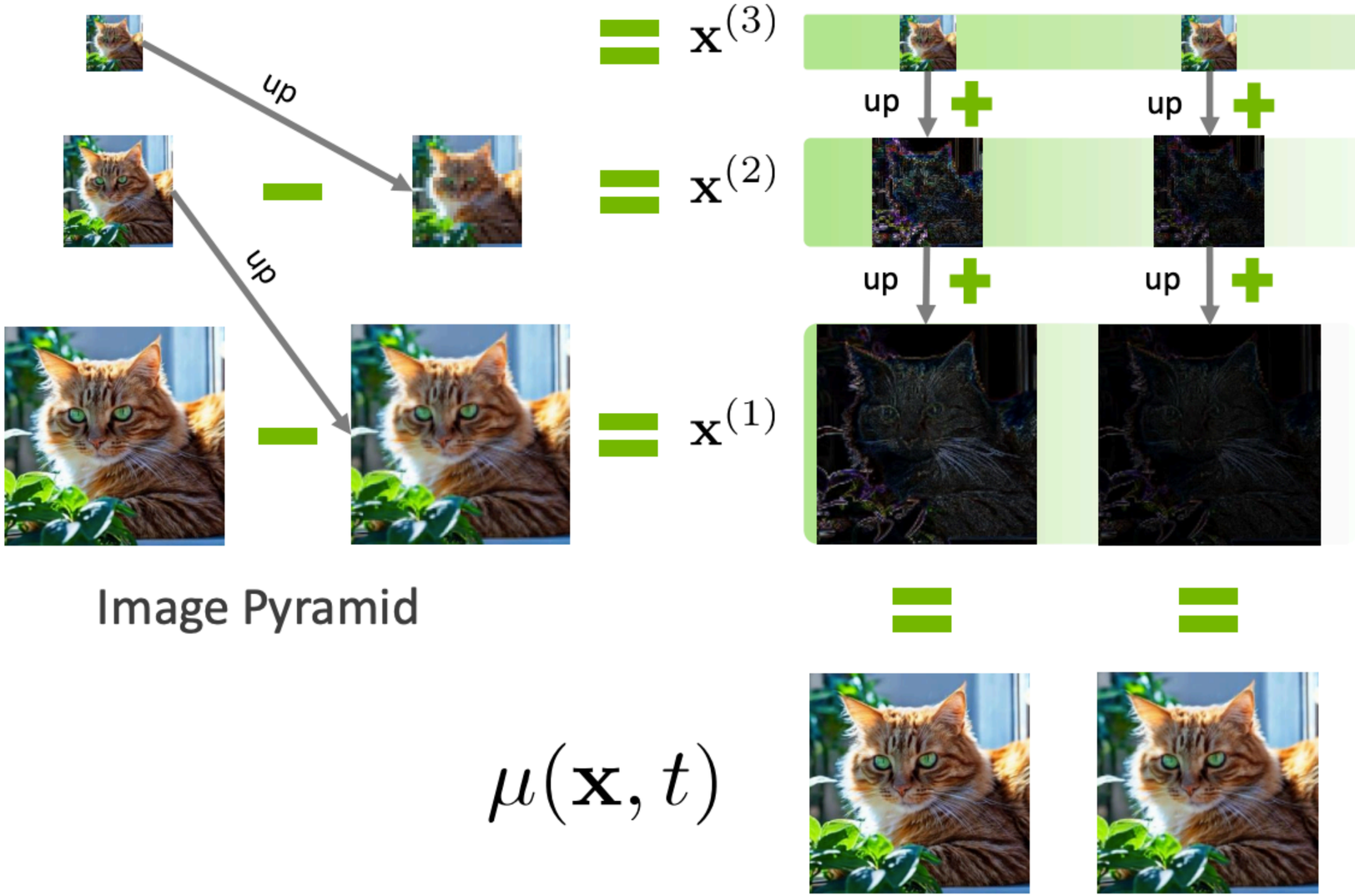
[Fig adapted from Edify Image, NVIDIA 2024]

Foundation: Laplacian Pyramid



[Fig adapted from Edify Image, NVIDIA 2024]

Foundation: Laplacian Pyramid



Core idea: Noise higher frequencies faster

[Fig adapted from Edify Image, NVIDIA 2024]

[Fig adapted from Edify Image, NVIDIA 2024]

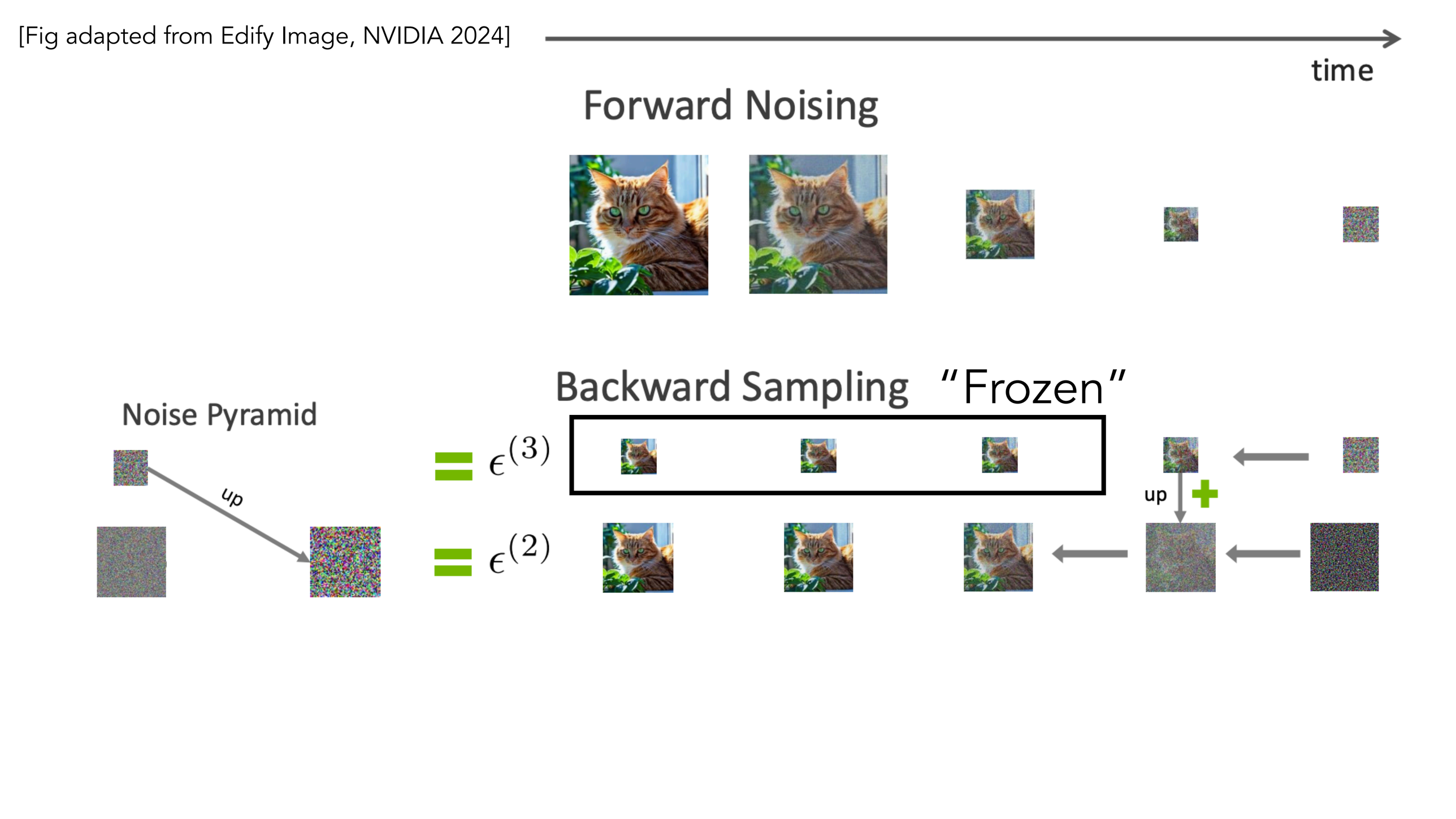
time

Forward Noising

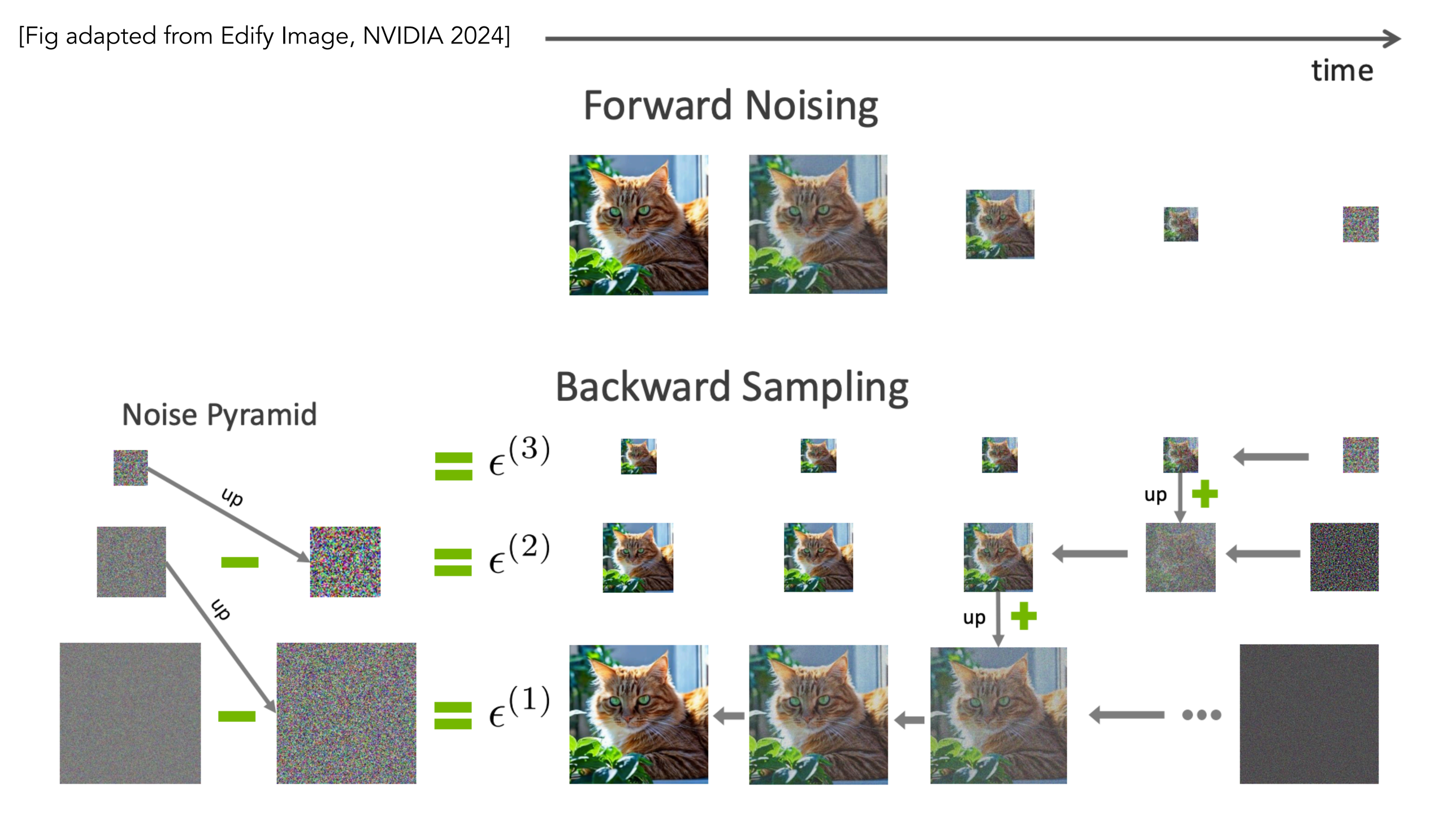
Backward Sampling

Noise Pyramid

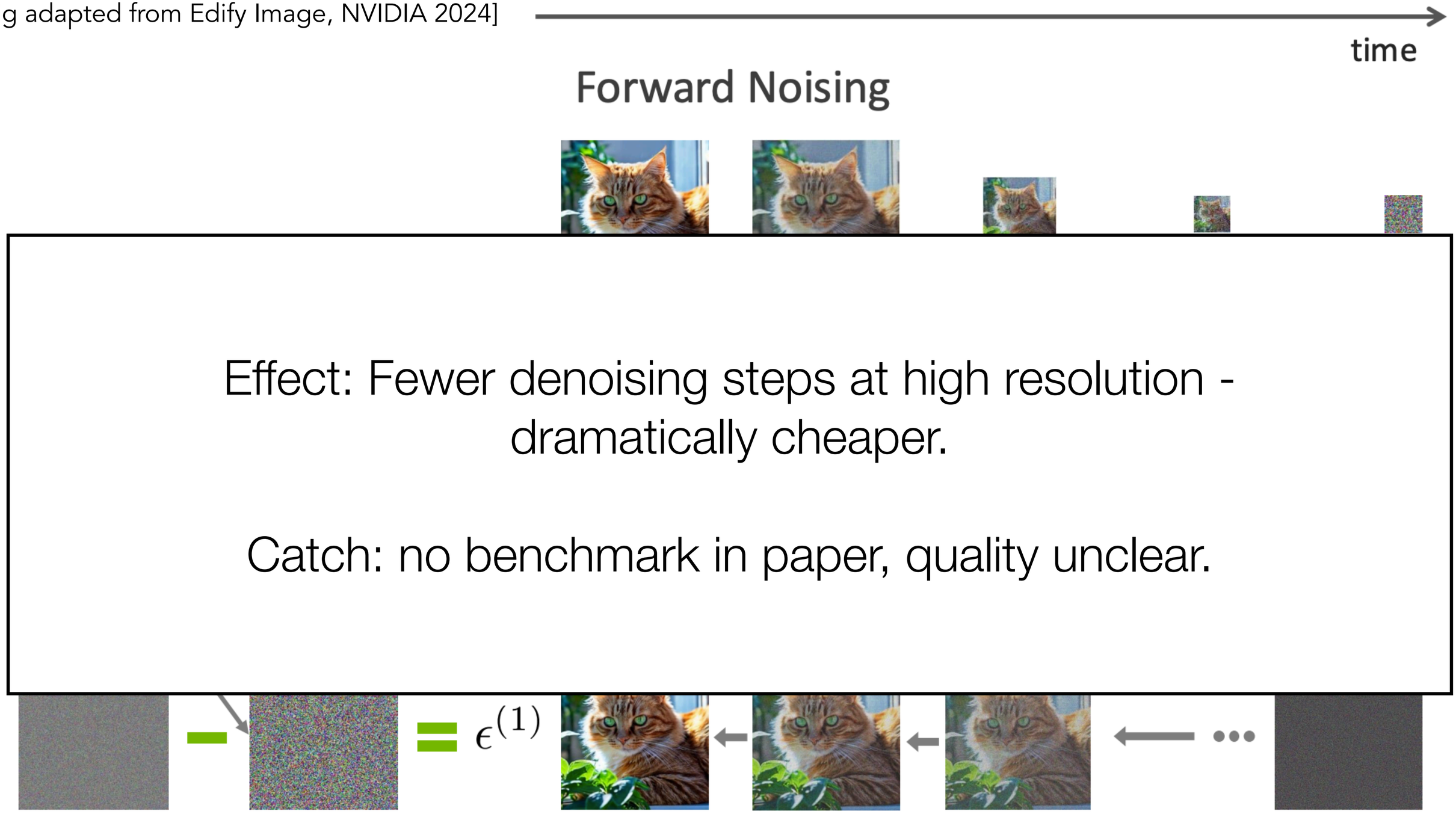
[Fig adapted from Edify Image, NVIDIA 2024]



[Fig adapted from Edify Image, NVIDIA 2024]



[Fig adapted from Edify Image, NVIDIA 2024]



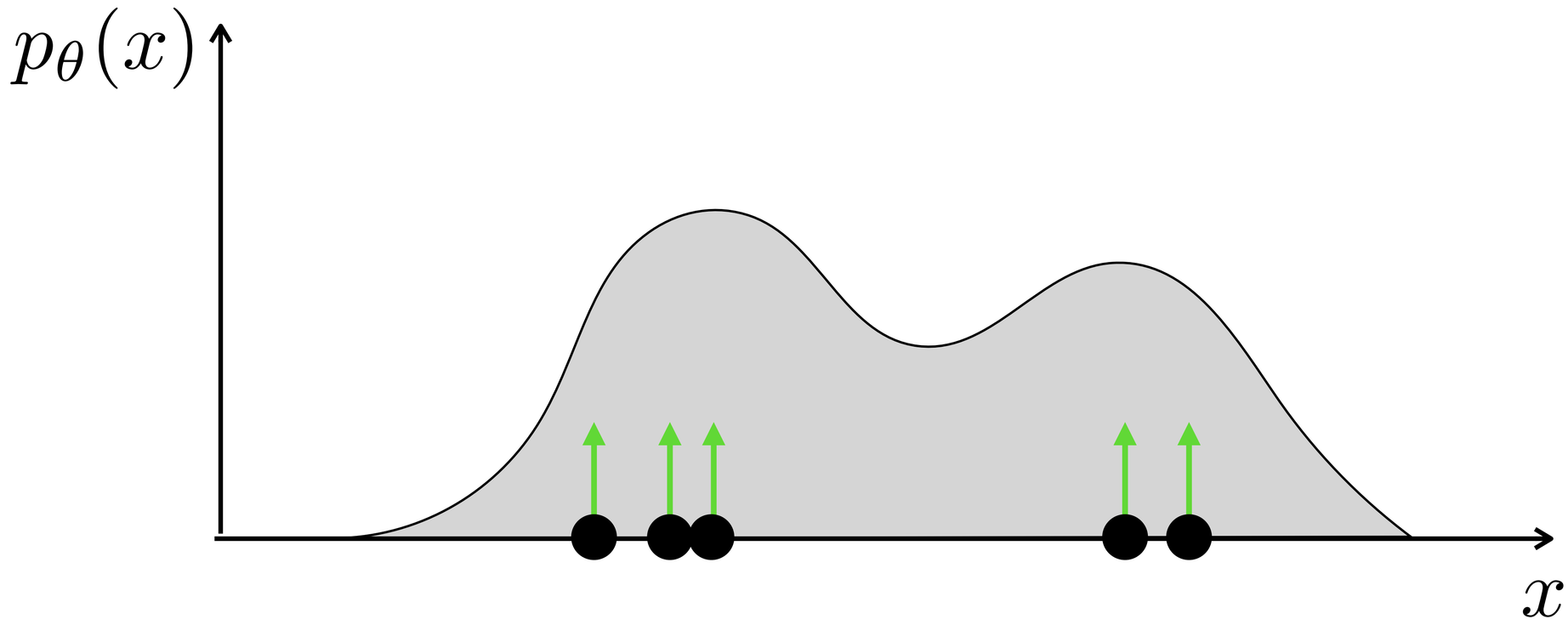
Effect: Fewer denoising steps at high resolution - dramatically cheaper.

Catch: no benchmark in paper, quality unclear.

What can we say about the images generated by a diffusion model?

Diffusion's Analytical Probability Density

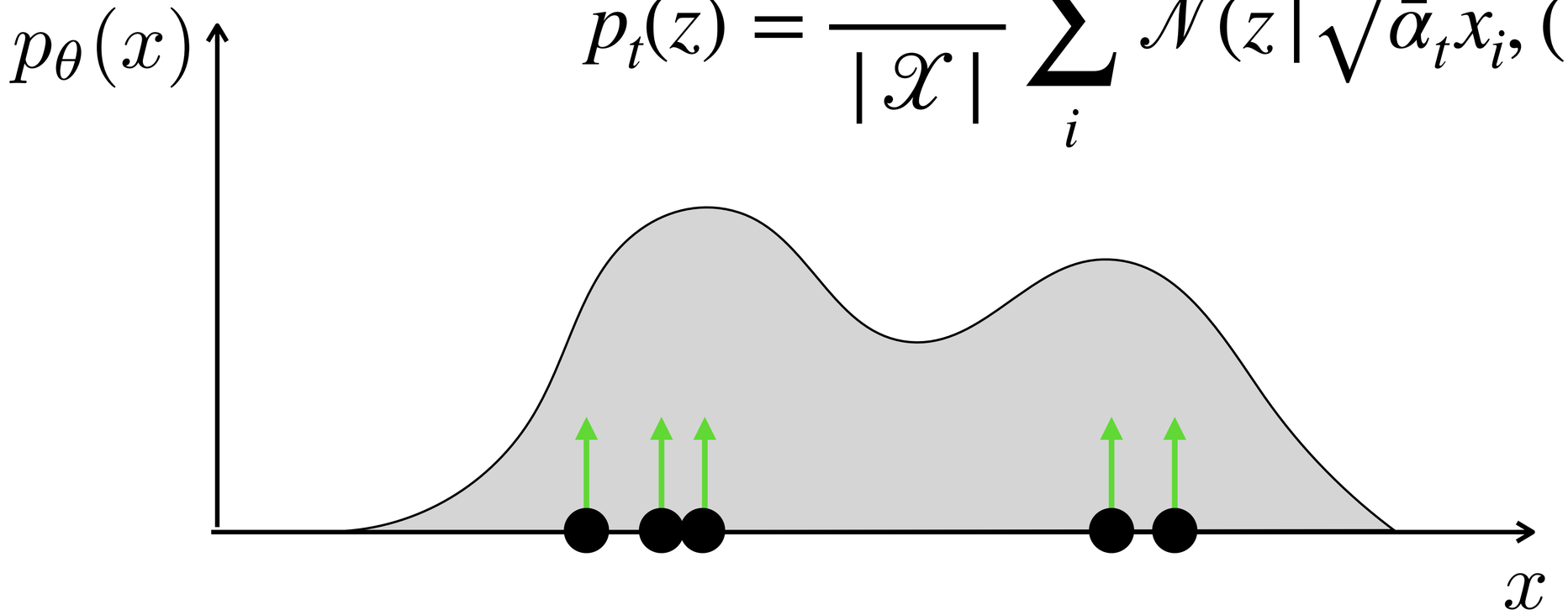
For diffusion models, $p_t(z)$ is a mixture of Gaussians centered at each data point $x_i \in \mathcal{X}$, where the variance depends on the timestep!



Diffusion's Analytical Probability Density

For diffusion models, $p_t(z)$ is a mixture of Gaussians centered at each data point $x_i \in \mathcal{X}$, where the variance depends on the timestep!

$$p_t(z) = \frac{1}{|\mathcal{X}|} \sum_i \mathcal{N}(z \,|\, \sqrt{\bar{\alpha}_t} x_i, (1 - \bar{\alpha}_t) I)$$

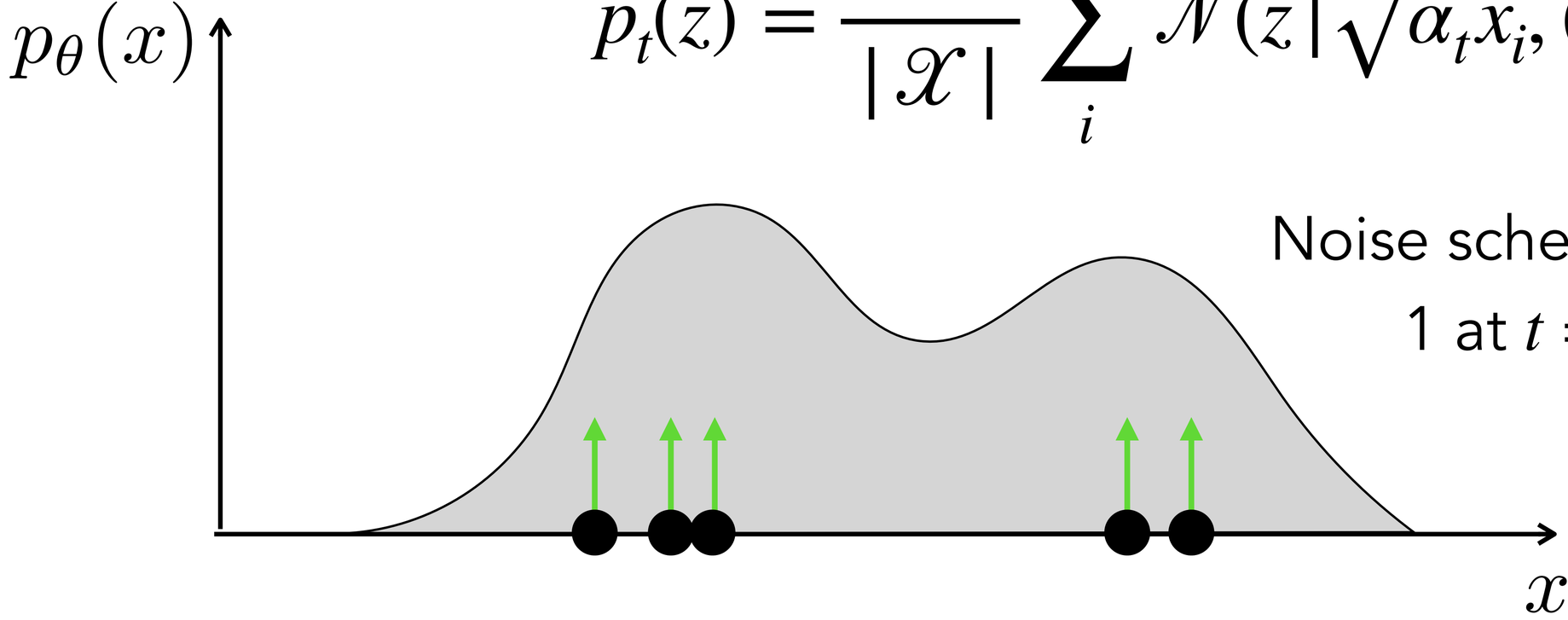


Diffusion's Analytical Probability Density

For diffusion models, $p_t(z)$ is a mixture of Gaussians centered at each data point $x_i \in \mathcal{X}$, where the variance depends on the timestep!

$$p_t(z) = \frac{1}{|\mathcal{X}|} \sum_i \mathcal{N}(z \,|\, \sqrt{\bar{\alpha}_t} x_i, (1 - \bar{\alpha}_t) I)$$

Noise schedule, decreases from 1 at $t = 0$ to 0 at $t = T$



Diffusion's Analytical Probability **Score**

$$p_t(z) = \frac{1}{|\mathcal{X}|} \sum_i \mathcal{N}(z | \sqrt{\bar{\alpha}_t} x_i, (1 - \bar{\alpha}_t) I)$$

Diffusion's Analytical Probability **Score**

$$p_t(z) = \frac{1}{|\mathcal{X}|} \sum_i \mathcal{N}(z | \sqrt{\bar{\alpha}_t} x_i, (1 - \bar{\alpha}_t) I)$$

This has an analytical score !

Diffusion's Analytical Probability **Score**

$$p_t(z) = \frac{1}{|\mathcal{X}|} \sum_i \mathcal{N}(z | \sqrt{\bar{\alpha}_t} x_i, (1 - \bar{\alpha}_t) I)$$

This has an analytical score !

$$\nabla \log p_t^*(z) = \frac{1}{(1-t)^2} \left(k_t(z) - z \right)$$

$$k_t(z) = \sum_{i=1}^{N} softmax \left(-\frac{\|z - tX\|^2}{2(1-t)^2} \right)_i tx_i$$

Diffusion's Analytical Probability **Score**

$$p(z) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{N}(z \mid \sqrt{\bar{\alpha}}\, x_i, (1-\bar{\alpha})I)$$

What happens if we generate samples using this score?

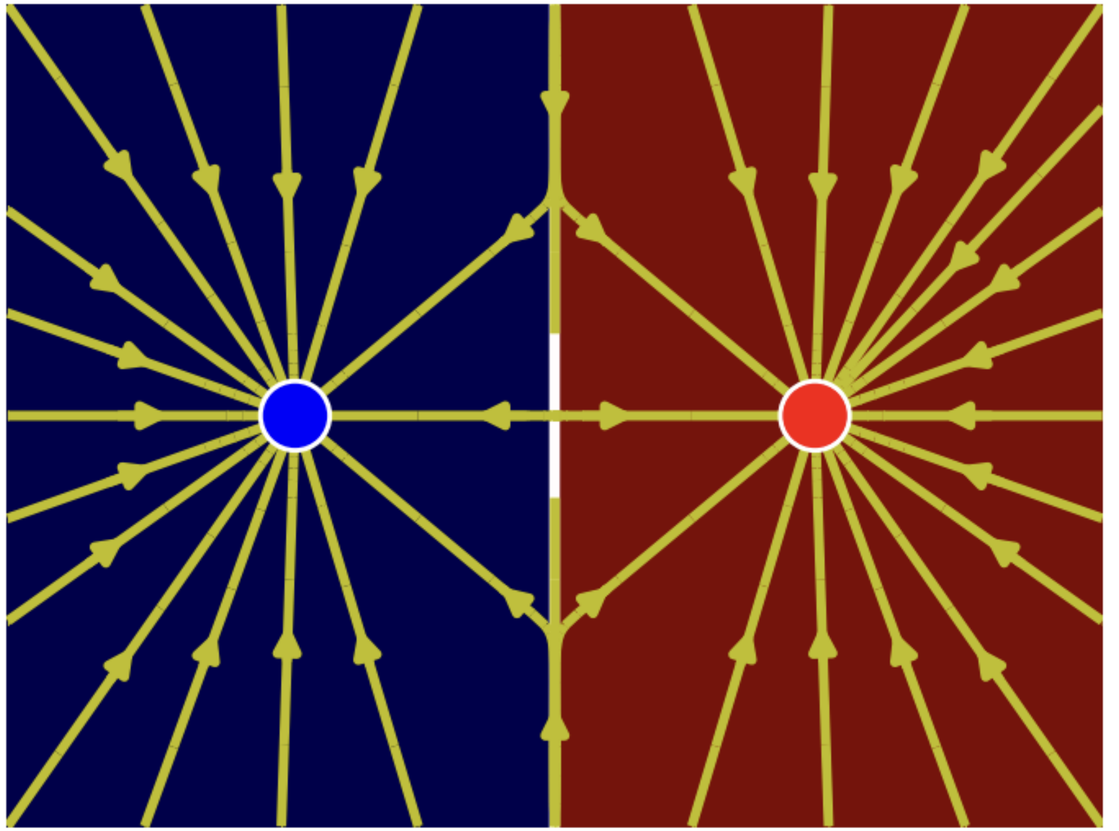
$$k_t(z) = \sum_{i=1}^{N} softmax\left(-\frac{\|z - tX\|^2}{2(1-t)^2} \right)_i tx_i$$

“Optimal Denoisers” or closed-from diffusion

$$\nabla \log p_t^*(z) = \frac{1}{(1-t)^2}\left(k_t(z) - z\right)$$

$$k_t(z) = \sum_{i=1}^{N} softmax\left(-\frac{\|z - tX\|^2}{2(1-t)^2}\right)_i tx_i$$

Consider two “training images”
(red and blue)



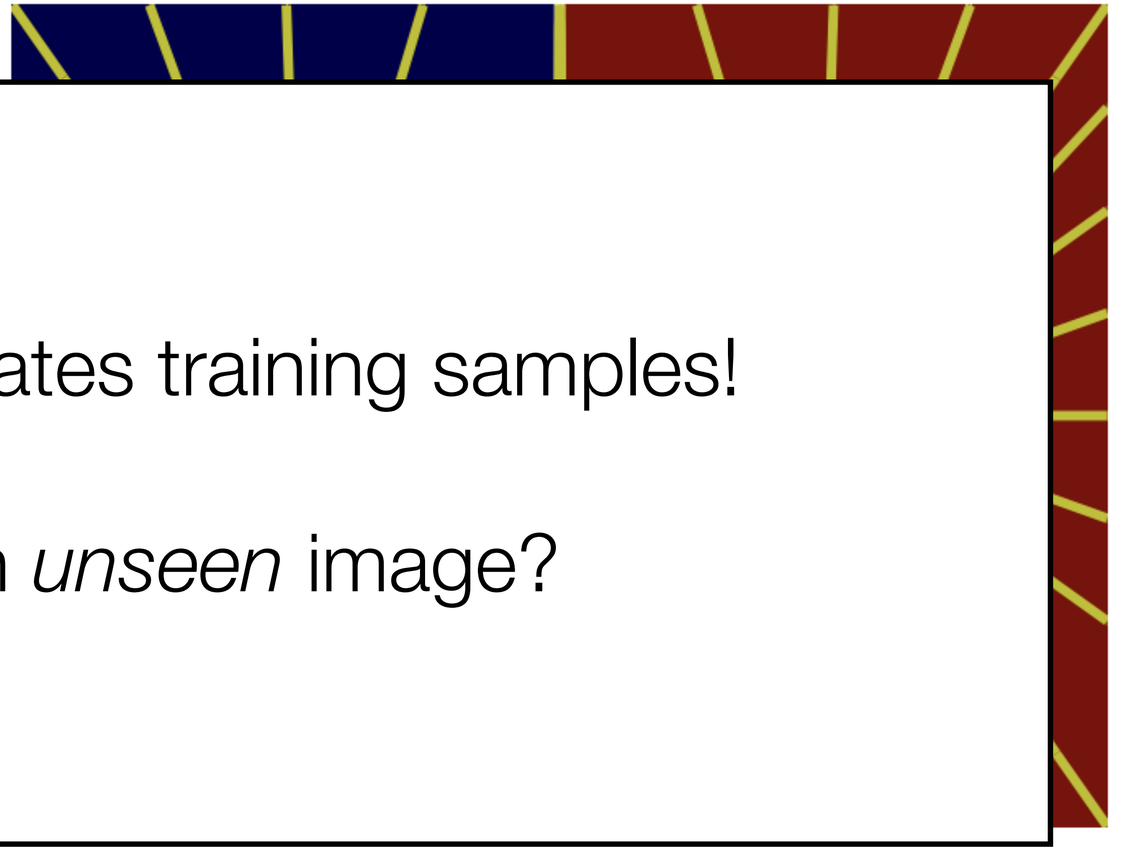
(a) Closed-form score

Diffusion will just retrieve one of them…

[Closed-form Diffusion Models, Scarvelis et al 2023]

“Optimal Denoisers” or closed-from diffusion

$$\nabla \log p_t^*(z) = \frac{1}{(1-t)^2}\left(k_t(z) - z\right)$$



The *optimal* denser just generates training samples!

How can we generate an *unseen* image?

Diffusion will just retrieve one of them…

(a) Closed-form score

[Closed-form Diffusion Models, Scarvelis et al 2023]

Where does generalization come from…?

Closed-Form Diffusion Models

Christopher Scarvelis *scarv@mit.edu*
MIT CSAIL

Haitz Sáez de Ocáriz Borde *chri6704@ox.ac.uk*
University of Oxford

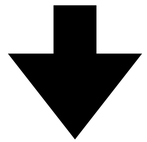
Justin Solomon *jsolomon@mit.edu*
MIT CSAIL

Abstract

Score-based generative models (SGMs) sample from a target distribution by iteratively transforming noise using the score function of the perturbed target. For any finite training set, this score function can be evaluated in closed form, but the resulting SGM memorizes its training data and does not generate novel samples. In practice, one approximates the score by training a neural network via score-matching. The error in this approximation

Idea: Introduce Error Into Score via Smoothing

$$\nabla \log p_t^*(z) = \frac{1}{(1-t)^2}\left(k_t(z) - z\right)$$



$$s_{\sigma,t}(z) = \frac{1}{(1-t)^2}\left(\frac{1}{M}\sum_{m=1}^{M} k_t(z + \sigma\epsilon_m) - z\right)$$

Idea: Introduce Error Into Score via Smoothing

$$\nabla \log p_t^*(z) = \frac{1}{(1-t)^2}\left(k_t(z) - z\right)$$

$$s_{\sigma,t}(z) = \frac{1}{(1-t)^2}\left(\frac{1}{M}\sum_{m=1}^{M} k_t(z + \sigma\epsilon_m) - z\right)$$

We are *averaging* the weights k_t over a few different noise perturbations of the current noisy sample z.

Idea: Introduce Error Into Score via Smoothing

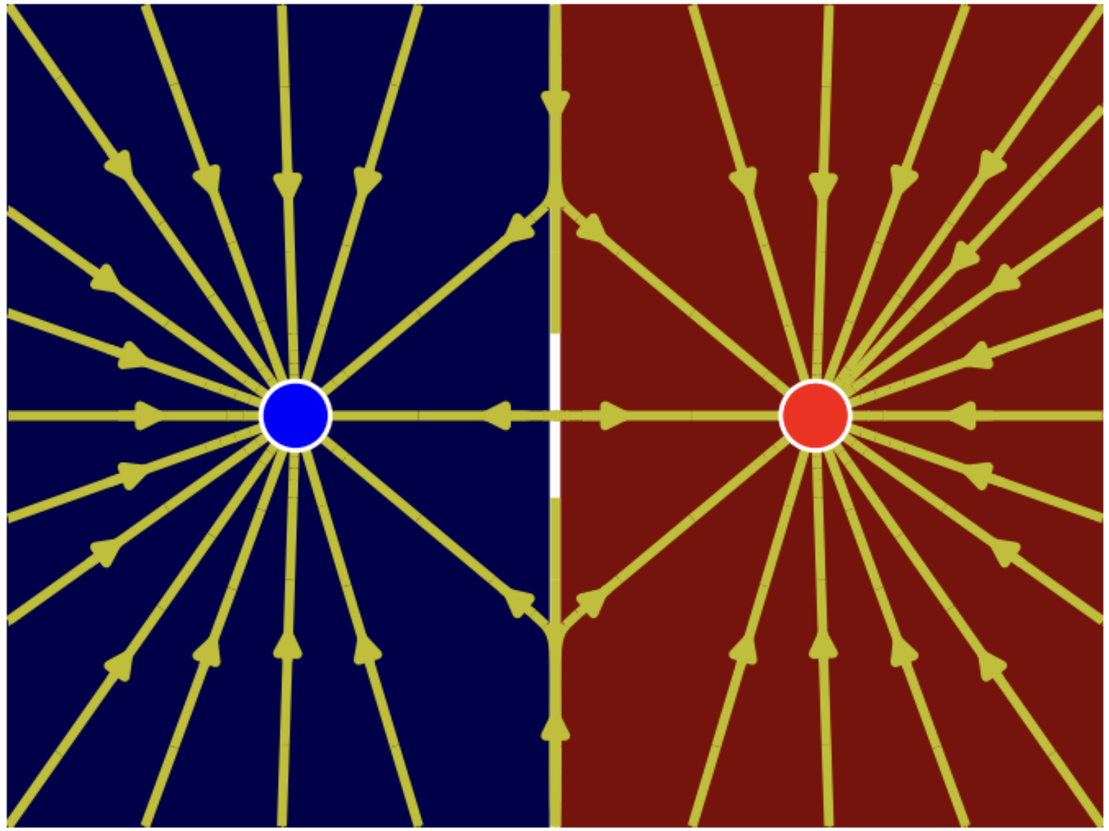
$$s_{\sigma,t}(z) = \frac{1}{(1-t)^2}\left(\frac{1}{M}\sum_{m=1}^{M} k_t(z+\sigma\epsilon_m) - z\right)$$

Idea: Introduce Error Into Score via Smoothing

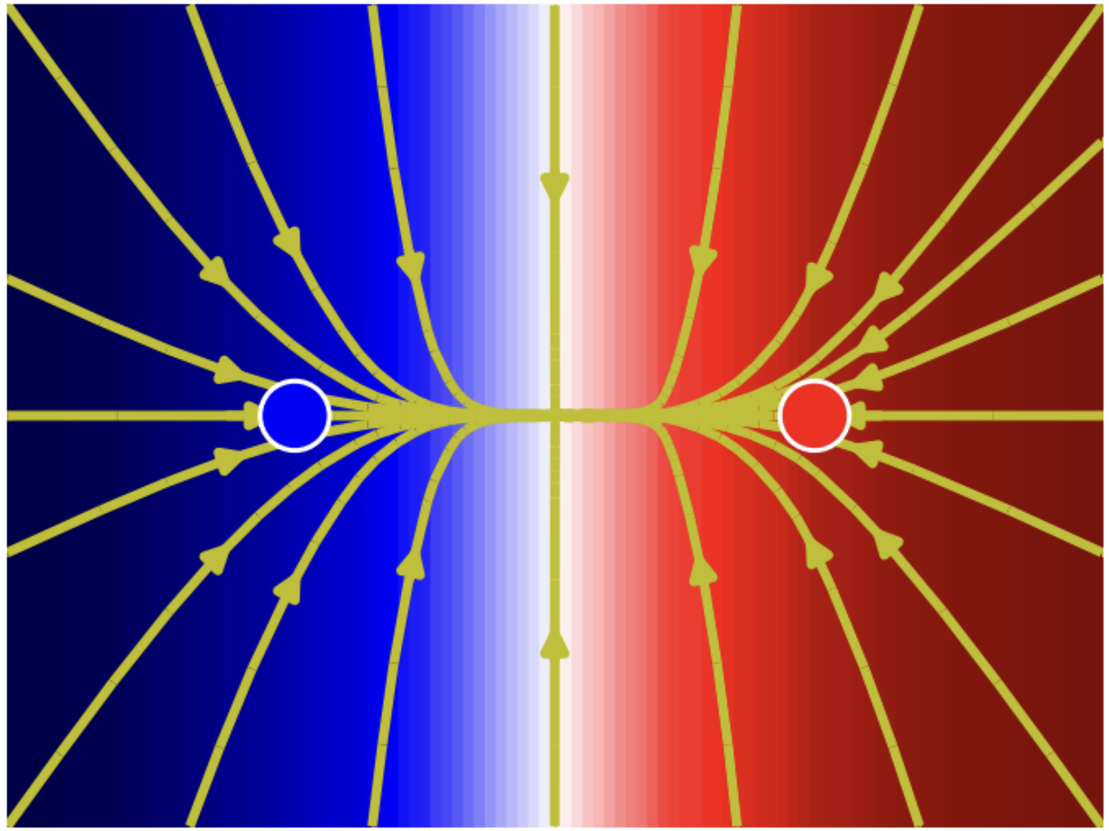
$$s_{\sigma,t}(z) = \frac{1}{(1-t)^2}\left(\frac{1}{M}\sum_{m=1}^{M} k_t(z+\sigma\epsilon_m) - z\right)$$

In [Closed-form Diffusion Models, Scarvelis et al 2023], the authors show that this score points towards *barycenters* of images in the training set.

Idea: Introduce Error Into Score via Smoothing



(a) Closed-form score



(b) Smoothed score

Samples from CelebA with Smoothed Diffusion

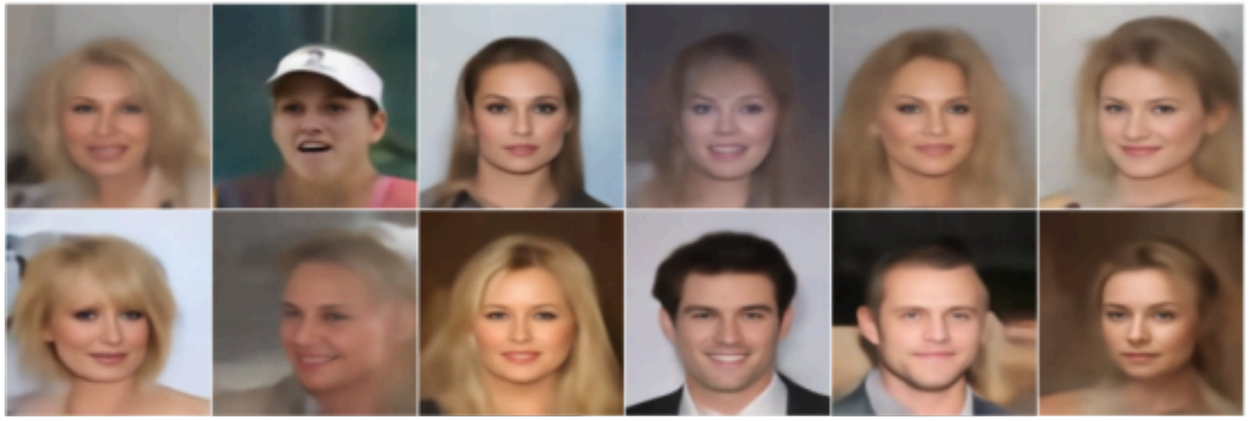
(a) Ground truth samples

(b) VAE samples

(c) σ-CFDM samples

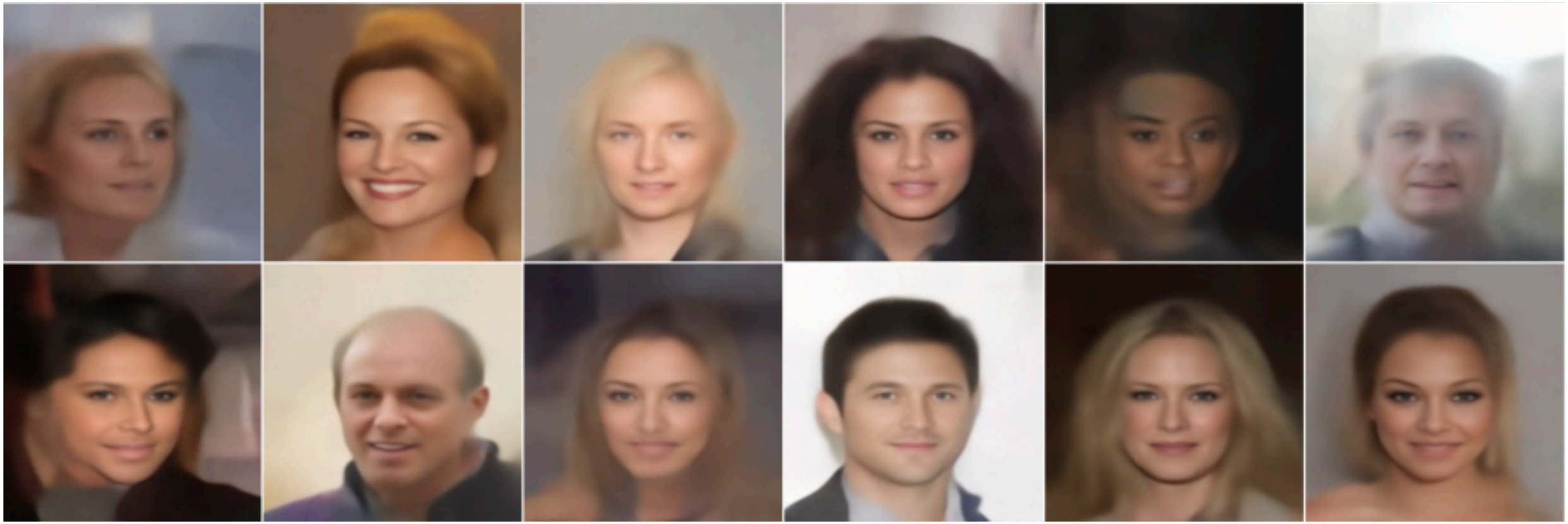
Samples from CelebA with Smoothed Diffusion

(b) VAE samples



(c) σ-CFDM samples

Samples from CelebA with Smoothed Diffusion



Not amazing quality, but *better* than a VAE, new faces, and *without* a NN altogether!

Flux.1-dev 12B

“a beaver eating a hamburger on a skateboard”

Flux.1-dev 12B

“a beaver eating a hamburger on a skateboard”

Flux.1-dev 12B

“a beaver eating a hamburger on a skateboard”

“**a photo of a** beaver eating a hamburger on a skateboard”

Flux.1-dev 12B

“a beaver eating a hamburger on a skateboard”

“**a photo of a** beaver eating a hamburger on a skateboard”

Flux.1-dev 12B

“a beaver eating a hamburger on a skateboard”

“**a photo of a** beaver eating a hamburger on a skateboard”

“a photo of a beaver eating a hamburger on a skateboard **in front of Toyota 4Runner**”

Flux.1-dev 12B

“a beaver eating a hamburger on a skateboard”

“**a photo of a** beaver eating a hamburger on a skateboard”

“a photo of a beaver eating a hamburger on a skateboard **in front of Toyota 4Runner**”

Flux.1-dev 12B

Clearly, these images are *not* just averaging of training images…

Flux.1-dev 12B

Clearly, these images are *not* just averaging of training images…

“a beaver eating a hamburger on a skateboard”

Flux.1-dev 12B

Clearly, these images are *not* just averaging of training images…

“a beaver eating a hamburger on a skateboard”

Flux.1-dev 12B

Clearly, these images are *not* just averaging of training images…

“a beaver eating a hamburger on a skateboard”

“**a photo of a** beaver eating a hamburger on a skateboard”

Flux.1-dev 12B

Clearly, these images are *not* just averaging of training images…

“a beaver eating a hamburger on a skateboard”

“**a photo of a** beaver eating a hamburger on a skateboard”

Flux.1-dev 12B

Clearly, these images are *not* just averaging of training images…

“a beaver eating a hamburger on a skateboard”

“**a photo of a** beaver eating a hamburger on a skateboard”

“a photo of a beaver eating a hamburger on a skateboard **in front of Toyota 4Runner**”

Flux.1-dev 12B

Clearly, these images are *not* just averaging of training images…

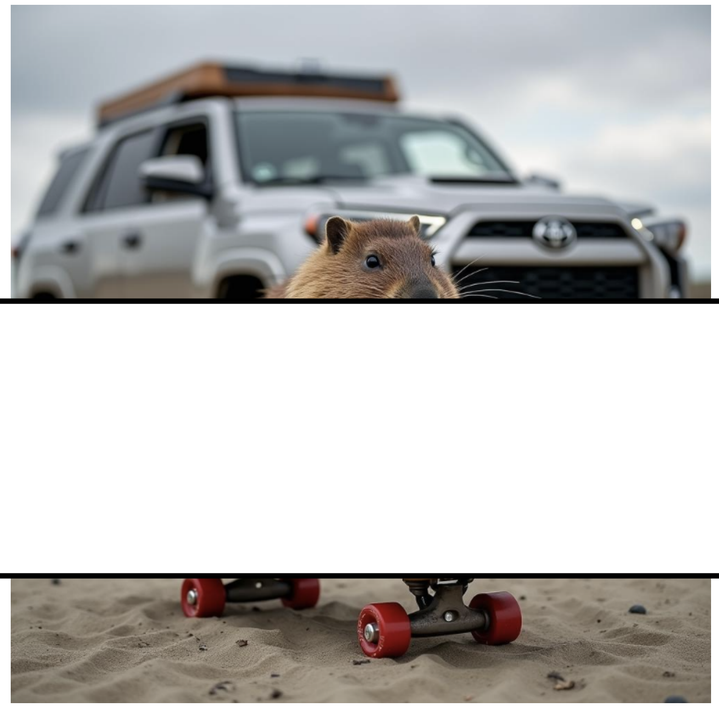
"a beaver eating a hamburger on a skateboard"

"**a photo of a** beaver eating a hamburger on a skateboard"

"a photo of a beaver eating a hamburger on a skateboard **in front of Toyota 4Runner**"

Flux.1-dev 12B

Clearly, these images are *not* just averaging of training images…



What might be missing?

“a beaver eating a hamburger on a skateboard”

“**a photo of a** beaver eating a hamburger on a skateboard”

“a photo of a beaver eating a hamburger on a skateboard **in front of Toyota 4Runner**”

Flux.1-dev 12B

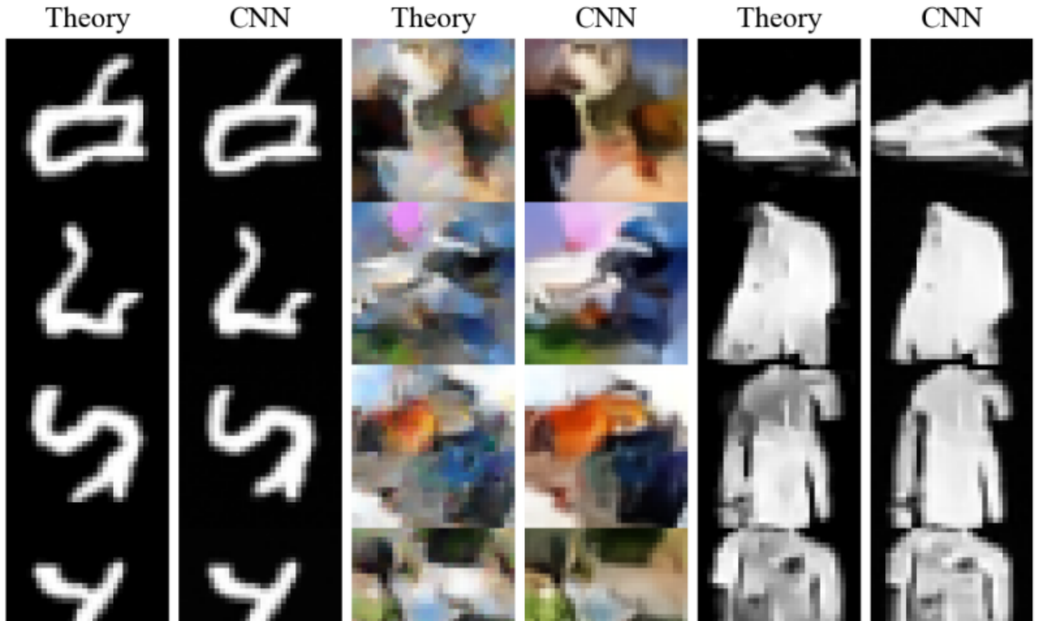
Where does generalization come from…?

An analytic theory of creativity in convolutional diffusion models

Mason Kamb [1] **Surya Ganguli** [1]

Abstract

We obtain the first analytic, interpretable and predictive theory of creativity in convolutional diffusion models. Indeed, score-based diffusion models can generate highly creative images that lie far from their training data. But optimal score-matching theory suggests that these models should only be able to produce memorized training examples. To reconcile this theory-experiment gap, we identify two simple inductive biases, locality and equivariance, that: (1) induce a form



Diffusion's Analytical Probability **Score**

$$p_t(z) = \frac{1}{|\mathcal{X}|} \sum_i \mathcal{N}(z | \sqrt{\bar{\alpha}_t} x_i, (1 - \bar{\alpha}_t) I)$$

This has an analytical score !

$$\nabla \log p_t^*(z) = \frac{1}{(1-t)^2} \left(k_t(z) - z \right)$$

$$k_t(z) = \sum_{i=1}^{N} softmax \left(-\frac{\|z - tX\|^2}{2(1-t)^2} \right)_i tx_i$$

Diffusion's Analytical Probability **Score**

$$p_t(z) = \frac{1}{|\mathcal{X}|} \sum_i \mathcal{N}(z \,|\, \sqrt{\bar{\alpha}_t} x_i, (1 - \bar{\alpha}_t) I)$$

This has an analytical score !

$$\nabla \log p_t^*(z) = \frac{1}{(1-t)^2} \left(k_t(z) - z \right)$$

$$k_t(z) = \sum_{i=1}^{N} softmax \left(-\frac{\|z - tX\|^2}{2(1-t)^2} \right)_i tx_i$$

This kernel is computed over the *whole* image.

Transition to different notation…

$$\nabla \log p_t^*(z) = \frac{1}{(1-t)^2}\left(k_t(z) - z\right)$$

$$k_t(z) = \sum_{i=1}^{N} softmax\left(-\frac{\|z - tX\|^2}{2(1-t)^2}\right)_i tx_i$$

$$s_t(\phi) = \frac{1}{1-\bar{\alpha}_t}\sum_{\varphi \in \mathcal{D}} (\sqrt{\bar{\alpha}_t}\varphi - \phi) W_t(\varphi|\phi)$$

$$W_t(\varphi|\phi) = \frac{\mathcal{N}(\phi|\sqrt{\bar{\alpha}_t}\varphi, (1-\bar{\alpha}_t)I)}{\sum_{\varphi' \in \mathcal{D}} \mathcal{N}(\phi|\sqrt{\bar{\alpha}_t}\varphi', (1-\bar{\alpha}_t)I)}$$

This is to be consistent with the notation in the paper.

Transition to different notation…

$$\nabla \log p_t^*(z) = \frac{1}{(1-t)^2}\left(k_t(z) - z\right)$$

$$k_t(z) = \sum_{i=1}^{N} softmax\left(-\frac{\|z - tX\|^2}{2(1-t)^2}\right)_i tx_i$$

$$s_t(\phi) = \frac{1}{1-\bar{\alpha}_t}\sum_{\varphi \in \mathcal{D}} (\sqrt{\bar{\alpha}_t}\varphi - \phi) W_t(\varphi|\phi)$$

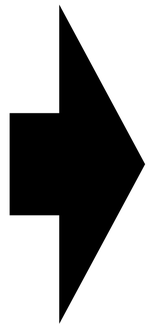
$$W_t(\varphi|\phi) = \frac{\mathcal{N}(\phi|\sqrt{\bar{\alpha}_t}\varphi, (1-\bar{\alpha}_t)I)}{\sum_{\varphi' \in \mathcal{D}} \mathcal{N}(\phi|\sqrt{\bar{\alpha}_t}\varphi', (1-\bar{\alpha}_t)I)}$$

This is to be consistent with the notation in the paper.

Transition to different notation…

$$\nabla \log p_t^*(z) = \frac{1}{(1-t)^2}\left(k_t(z) - z\right)$$

$$k_t(z) = \sum_{i=1}^{N} softmax\left(-\frac{\|z - tX\|^2}{2(1-t)^2}\right)_i tx_i$$



$$s_t(\phi) = \frac{1}{1-\bar{\alpha}_t}\sum_{\varphi \in \mathcal{D}} (\sqrt{\bar{\alpha}_t}\varphi - \phi) W_t(\varphi|\phi)$$

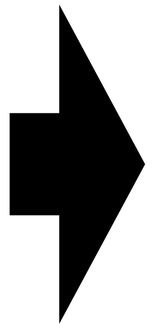
$$W_t(\varphi|\phi) = \frac{\mathcal{N}(\phi|\sqrt{\bar{\alpha}_t}\varphi, (1-\bar{\alpha}_t)I)}{\sum_{\varphi' \in \mathcal{D}} \mathcal{N}(\phi|\sqrt{\bar{\alpha}_t}\varphi', (1-\bar{\alpha}_t)I)}$$

This is to be consistent with the notation in the paper.

Transition to different notation…

$$\nabla \log p_t^*(z) = \frac{1}{(1-t)^2}\left(k_t(z) - z\right)$$

$$k_t(z) = \sum_{i=1}^{N} softmax\left(-\frac{\|z - tX\|^2}{2(1-t)^2}\right)_i tx_i$$



$$s_t(\phi) = \frac{1}{1-\bar{\alpha}_t}\sum_{\varphi \in \mathcal{D}} (\sqrt{\bar{\alpha}_t}\varphi - \phi) W_t(\varphi|\phi)$$

$$W_t(\varphi|\phi) = \frac{\mathcal{N}(\phi|\sqrt{\bar{\alpha}_t}\varphi, (1-\bar{\alpha}_t)I)}{\sum_{\varphi' \in \mathcal{D}} \mathcal{N}(\phi|\sqrt{\bar{\alpha}_t}\varphi', (1-\bar{\alpha}_t)I)}$$

This is to be consistent with the notation in the paper.

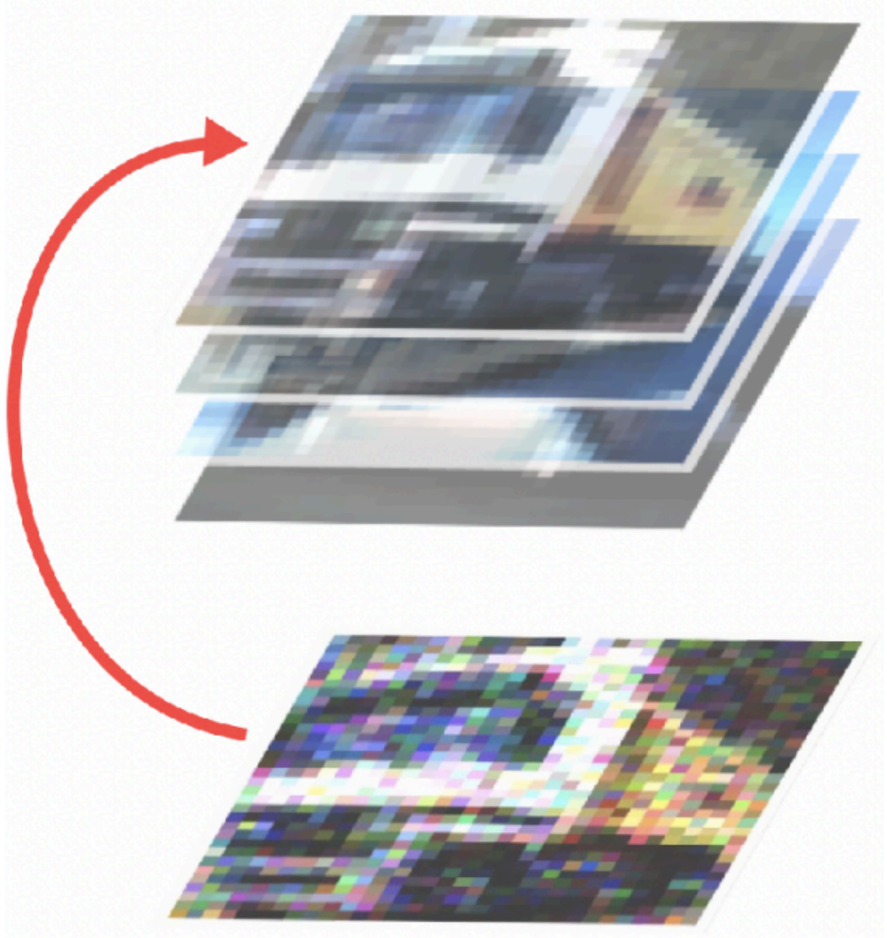
Transition to different notation…

$$s_t(\phi) = \frac{1}{1-\bar{\alpha}_t} \sum_{\varphi \in \mathcal{D}} (\sqrt{\bar{\alpha}_t}\varphi - \phi) W_t(\varphi|\phi)$$

$$W_t(\varphi|\phi) = \frac{\mathcal{N}(\phi|\sqrt{\bar{\alpha}_t}\varphi, (1-\bar{\alpha}_t)I)}{\sum_{\varphi' \in \mathcal{D}} \mathcal{N}(\phi|\sqrt{\bar{\alpha}_t}\varphi', (1-\bar{\alpha}_t)I)}$$

Time-dependent score function $s_t(\phi)$, current latent ϕ, images in training set φ, noise schedule $\bar{\alpha}_t$, softmax weights W_t

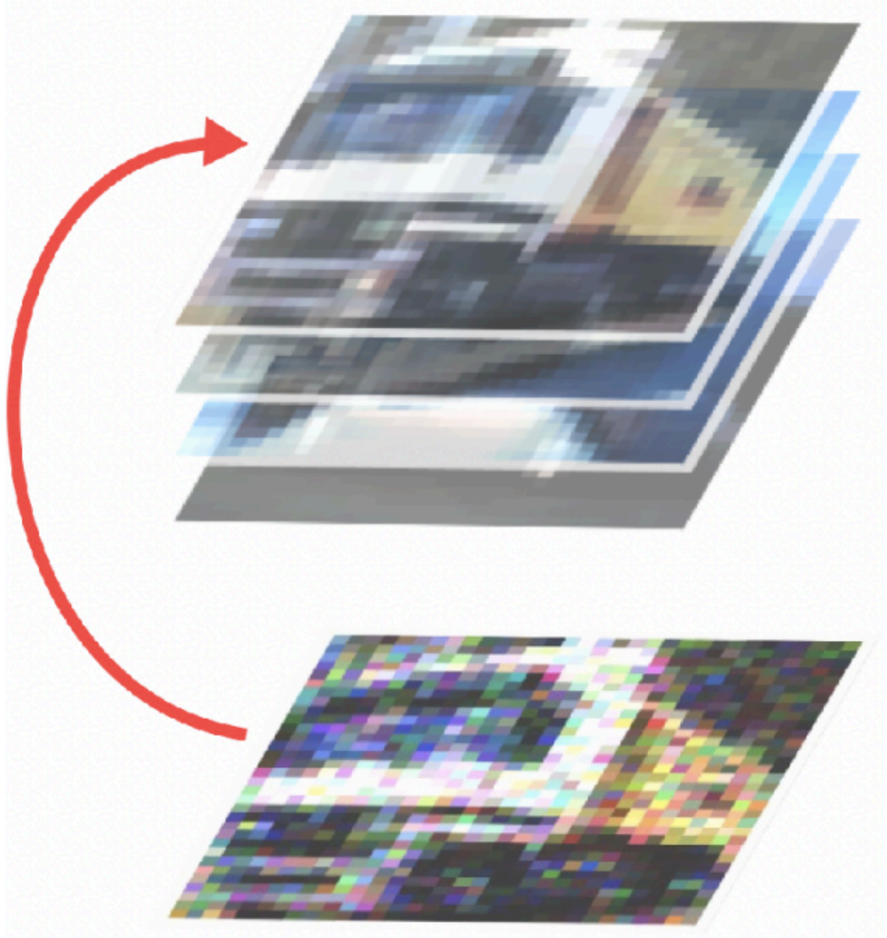
So far: Analytical Score Machine = Weighted Sum of Full Training Images



Weighted sum over *full-image* scores…

$$s_t(\phi) = \frac{1}{1-\bar{\alpha}_t} \sum_{\varphi \in \mathcal{D}} (\sqrt{\bar{\alpha}_t}\varphi - \phi) W_t(\varphi|\phi)$$

So far: Analytical Score Machine = Weighted Sum of Full Training Images



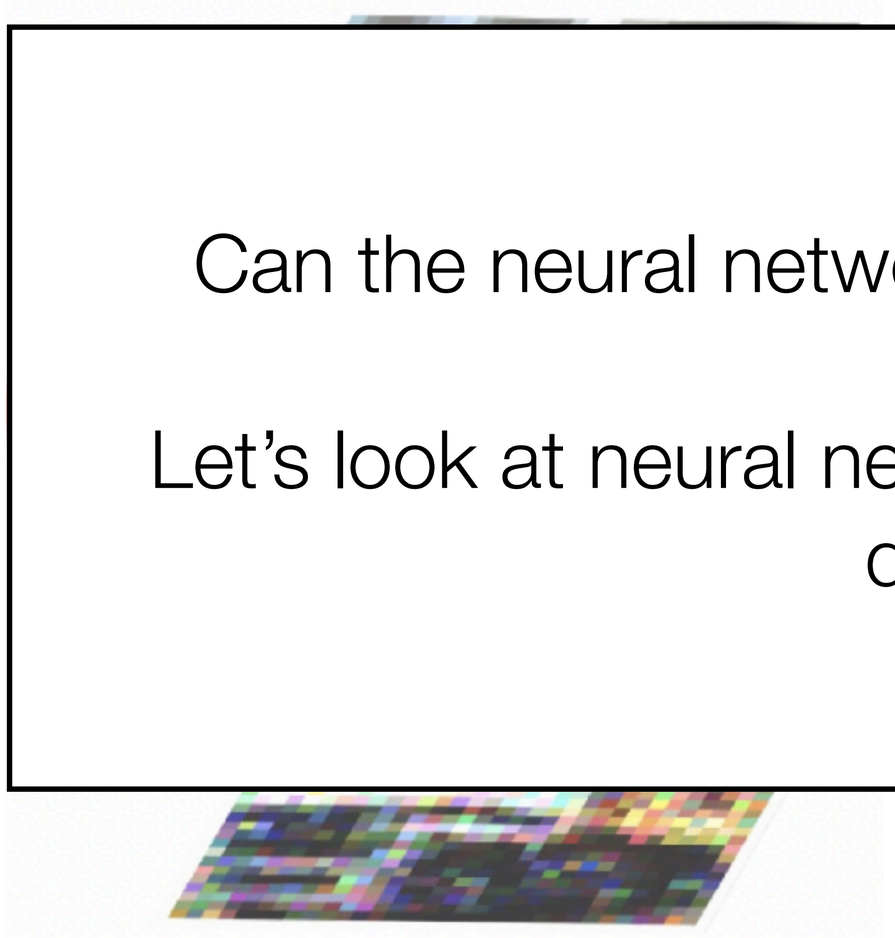
Weighted sum over *full-image* scores…

$$s_t(\phi) = \frac{1}{1-\bar{\alpha}_t} \sum_{\varphi \in \mathcal{D}} (\sqrt{\bar{\alpha}_t}\varphi - \phi) W_t(\varphi|\phi)$$

$$W_t(\varphi|\phi) = \frac{\mathcal{N}(\phi|\sqrt{\bar{\alpha}_t}\varphi, (1-\bar{\alpha}_t)I)}{\sum_{\varphi' \in \mathcal{D}} \mathcal{N}(\phi|\sqrt{\bar{\alpha}_t}\varphi', (1-\bar{\alpha}_t)I)}$$

…where weights are computed by comparing *full images.*

So far: Analytical Score Machine = Weighted Sum of Full Training Images

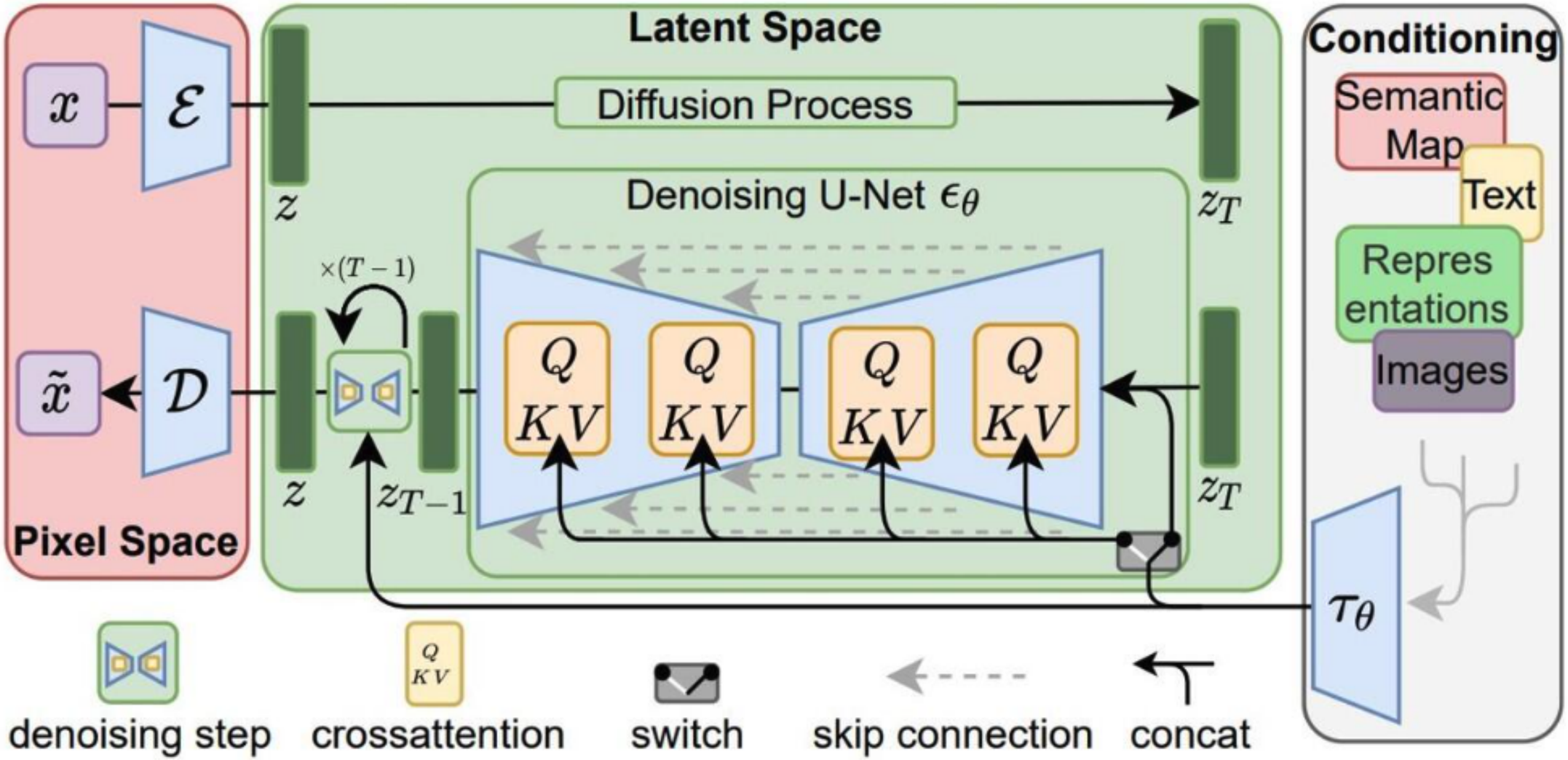


Can the neural network actually *learn* to predict this score?

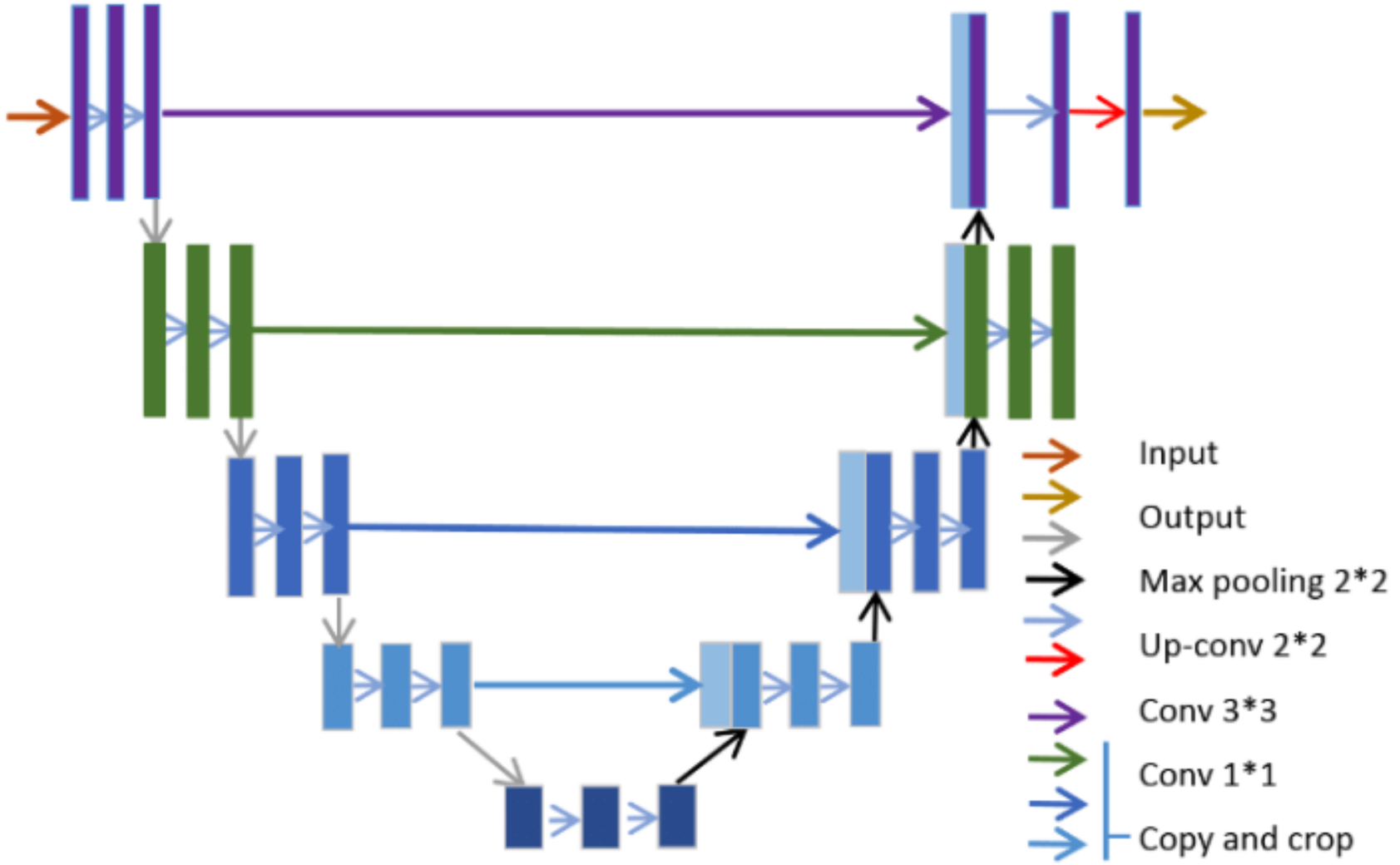
Let's look at neural net architectures that are actually used for diffusion in practice.

…where weights are computed by comparing *full images*.

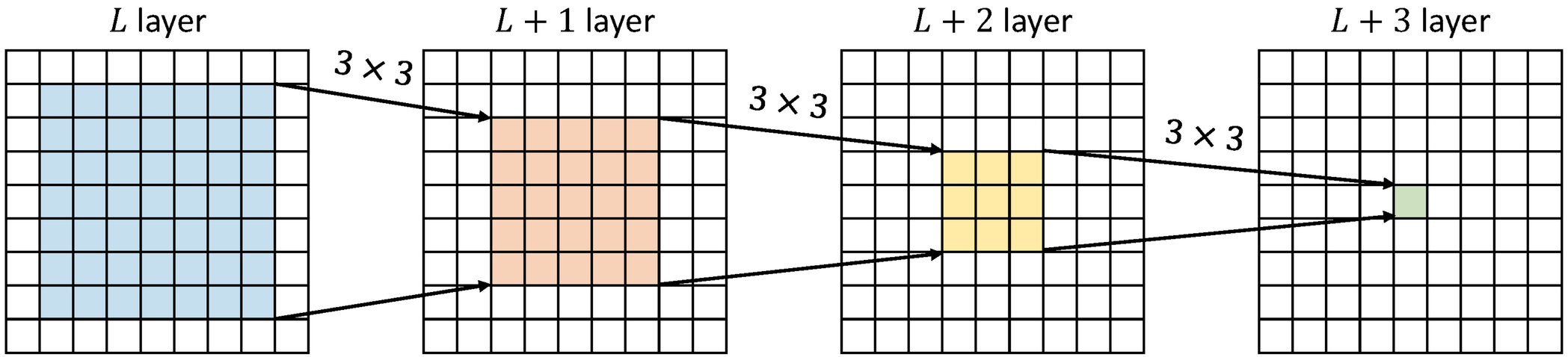
Stable Diffusion: Diffusion U-Nets



U-Nets: A Particular Kind of *Fully Convolutional* Neural Networks

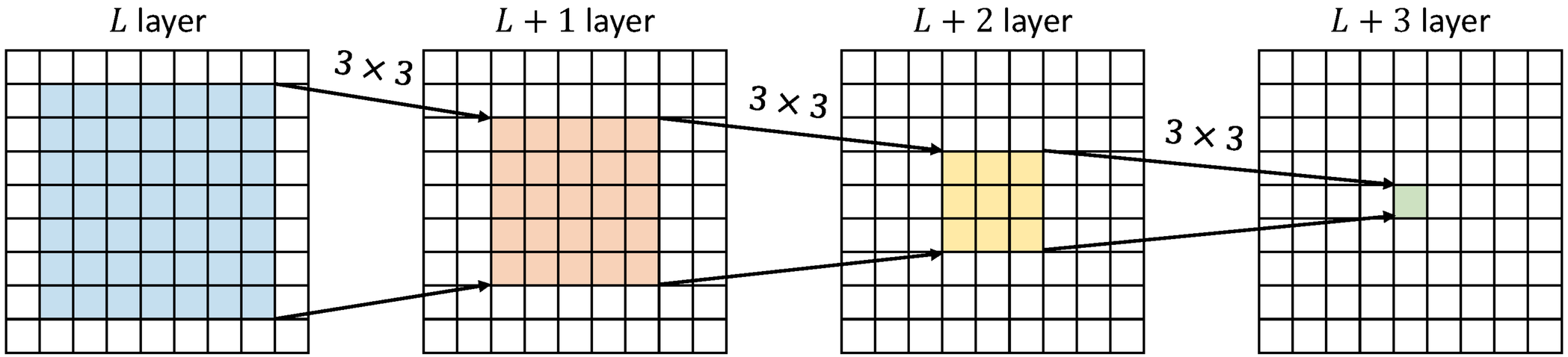


Fully Convolutional Neural Networks have a *Locality Bias.*



[Figure from Multi-Scale Receptive Fields Convolutional Network for Action Recognition, Dong et al. 203]

Fully Convolutional Neural Networks have a *Locality Bias.*



A given pixel in the output is a function of only the input pixels in its *receptive field.*

[Figure from Multi-Scale Receptive Fields Convolutional Network for Action Recognition, Dong et al. 203]

In practice, *Effective Receptive Fields* are smaller

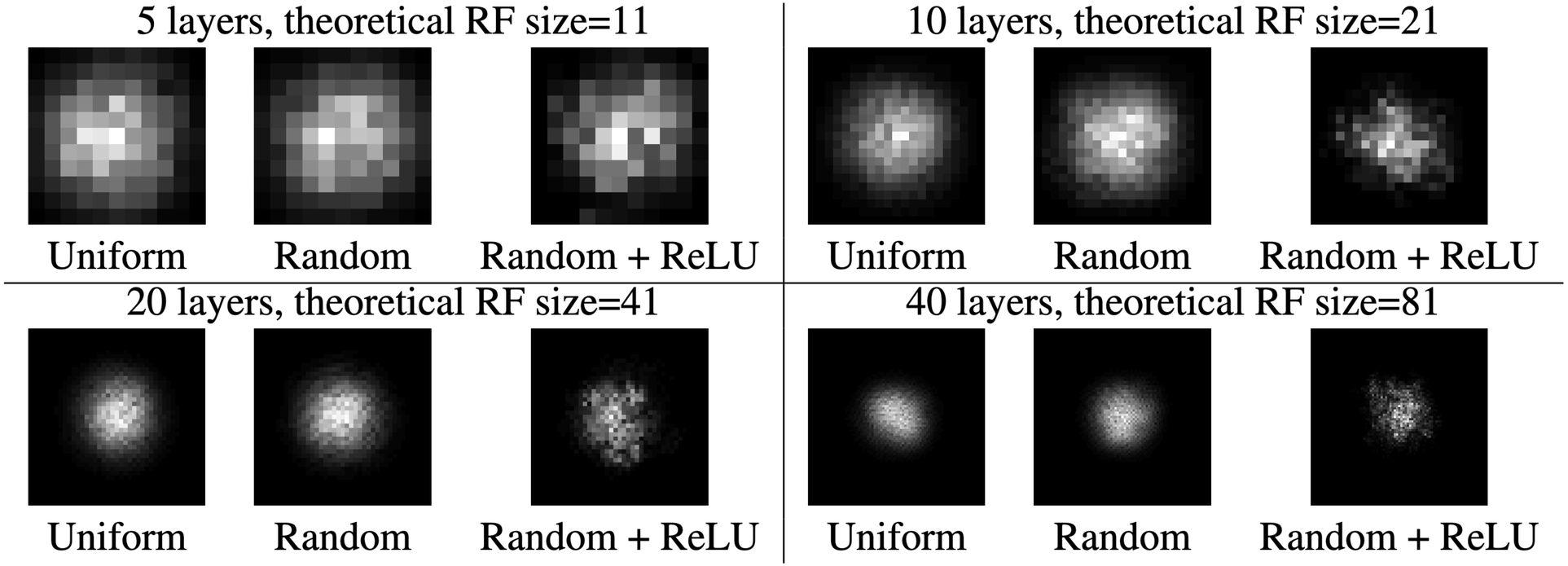
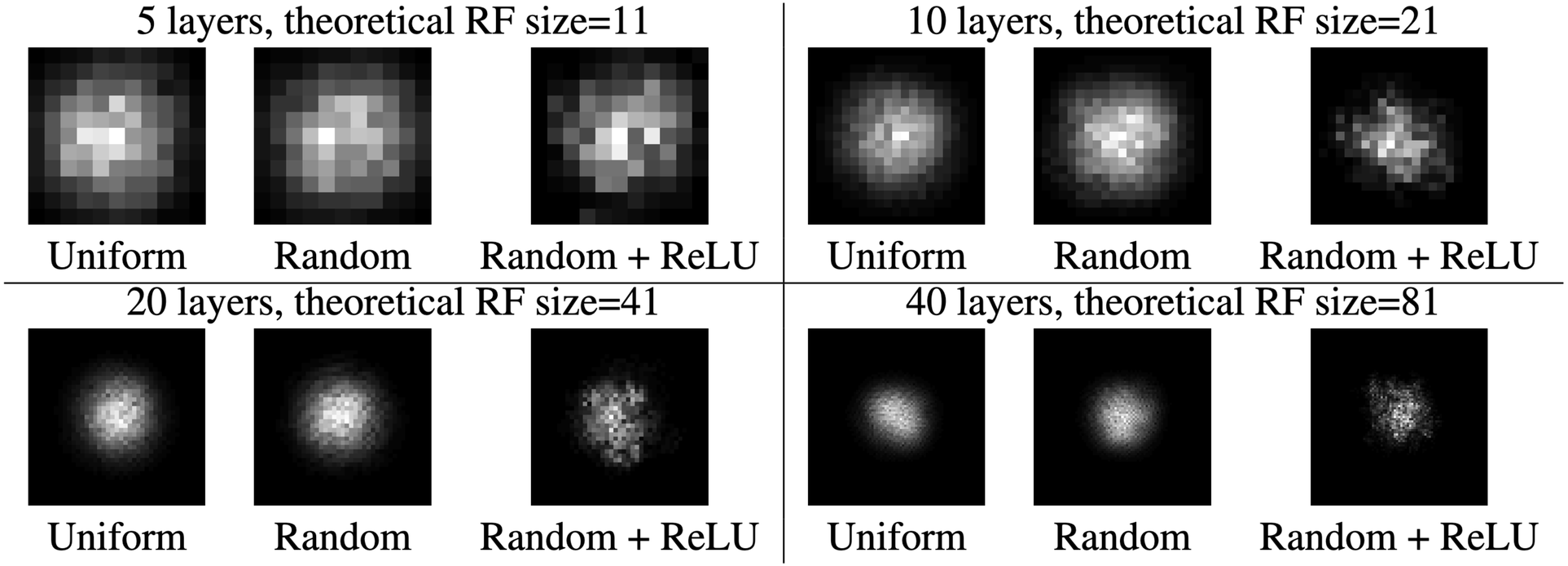


Figure from [Understanding the Effective Receptive Field in Deep Convolutional Neural Networks, Luo, Li et al. 2016]

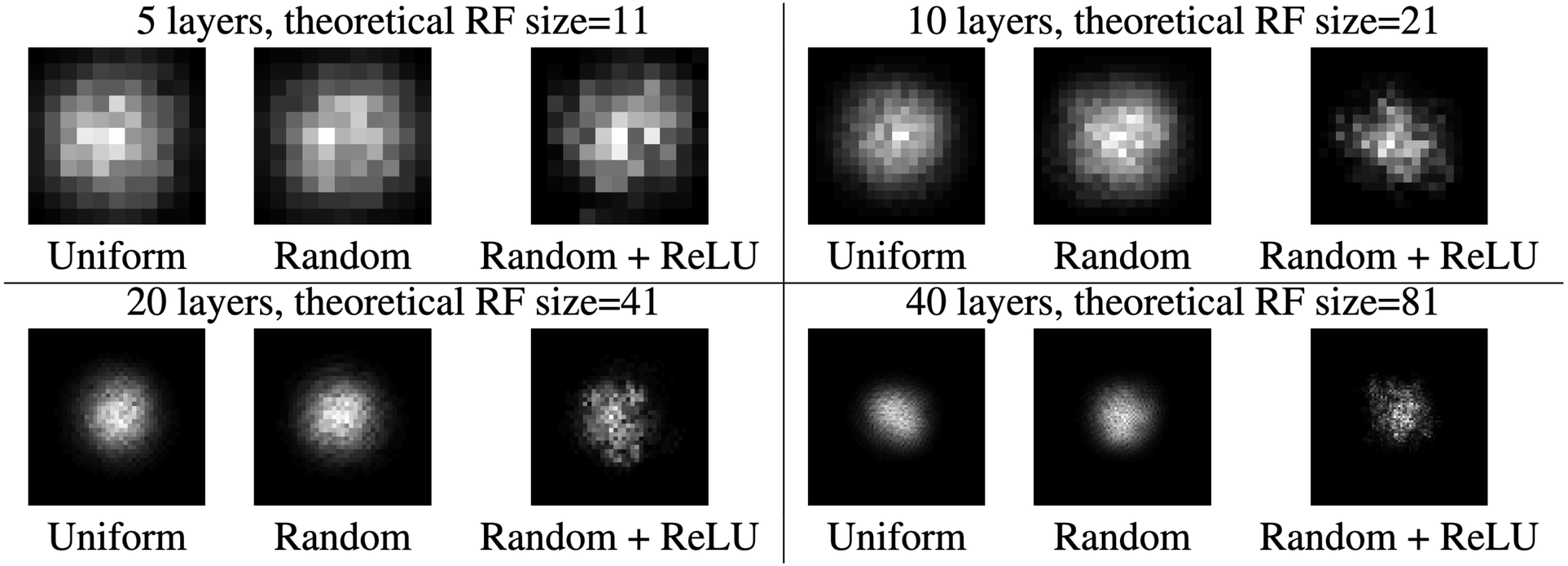
In practice, *Effective Receptive Fields* are smaller



Sensitivies (=magnitudes of gradients) of output pixel wrt input pixels.

Figure from [Understanding the Effective Receptive Field in Deep Convolutional Neural Networks, Luo, Li et al. 2016]

In practice, *Effective Receptive Fields* are smaller

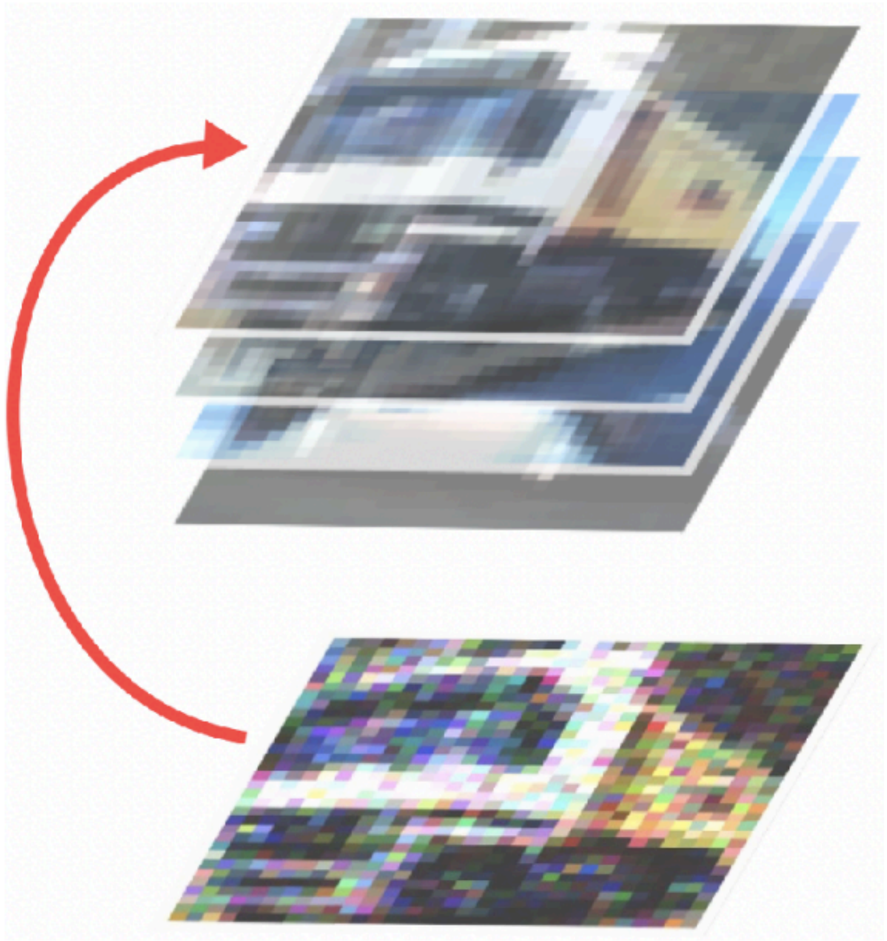


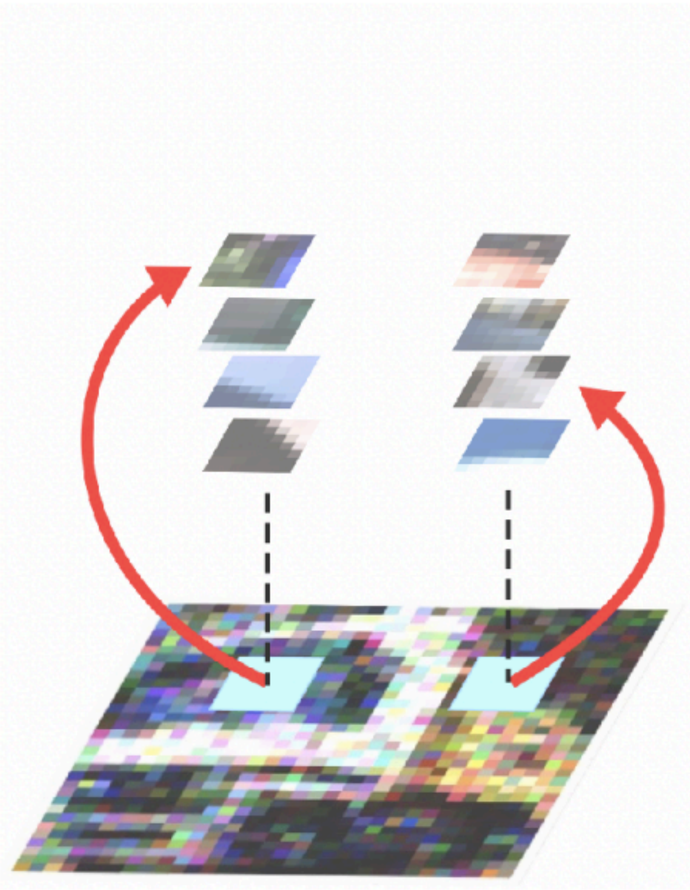
Sensitivies (=magnitudes of gradients) of output pixel wrt input pixels.

Denote effective receptive field as Ω_x.

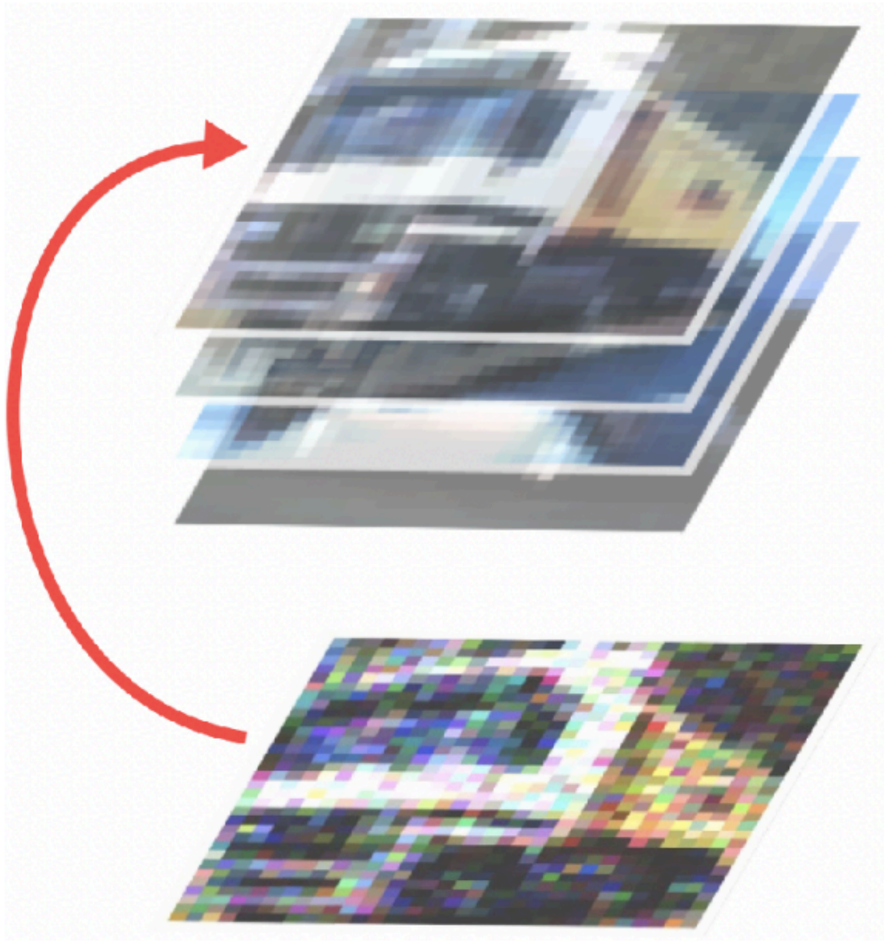
Figure from [Understanding the Effective Receptive Field in Deep Convolutional Neural Networks, Luo, Li et al. 2016]

Idea: Make Score Machine *Local*

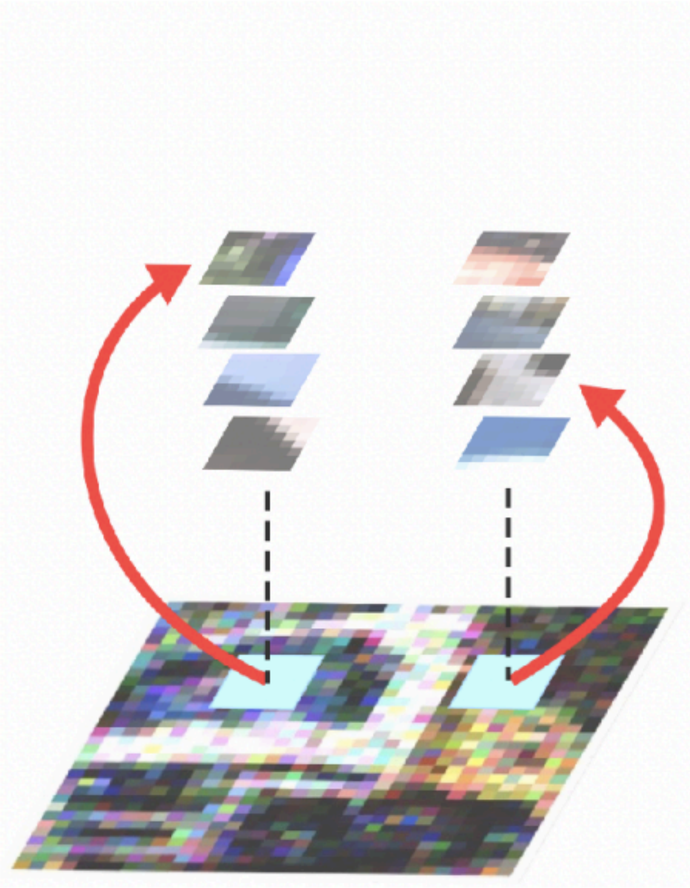




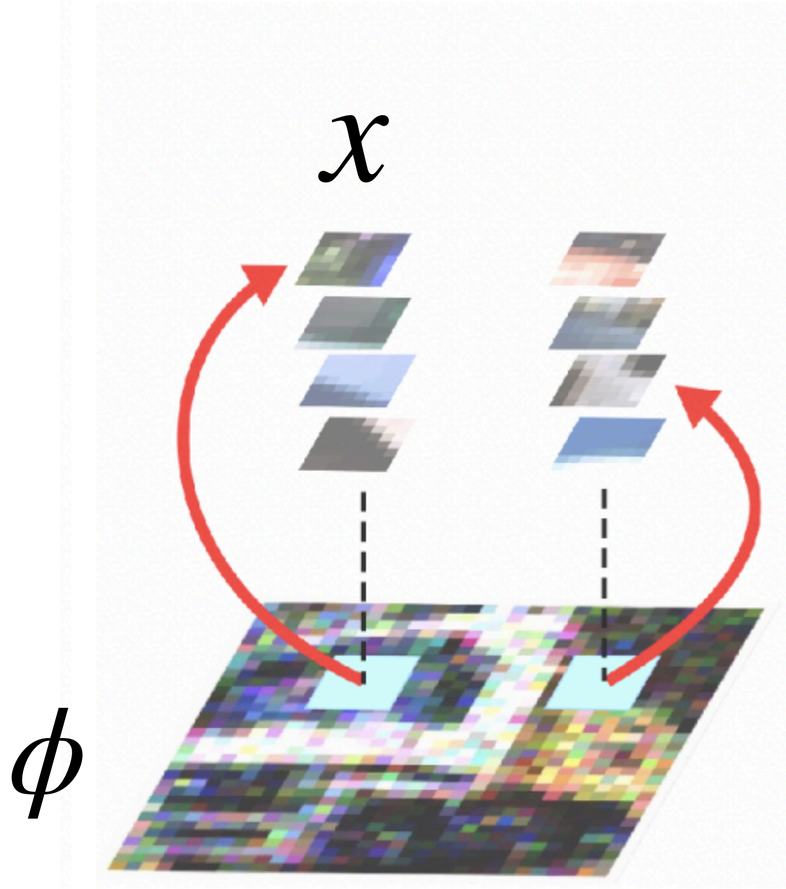
Idea: Make Score Machine *Local*



Each *patch* in the diffused image is a weighted sum of *patches* in training set!



Idea: Make Score Machine *Local*



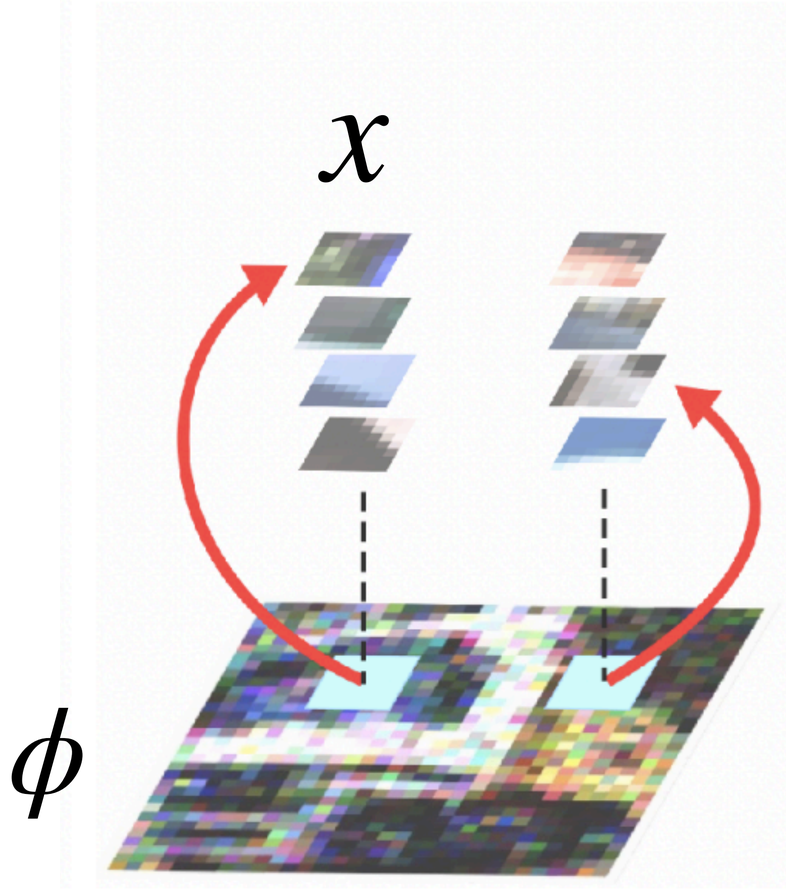
Idea: Make Score Machine *Local*

Denote:

Image as ϕ

Pixel coordinate as x

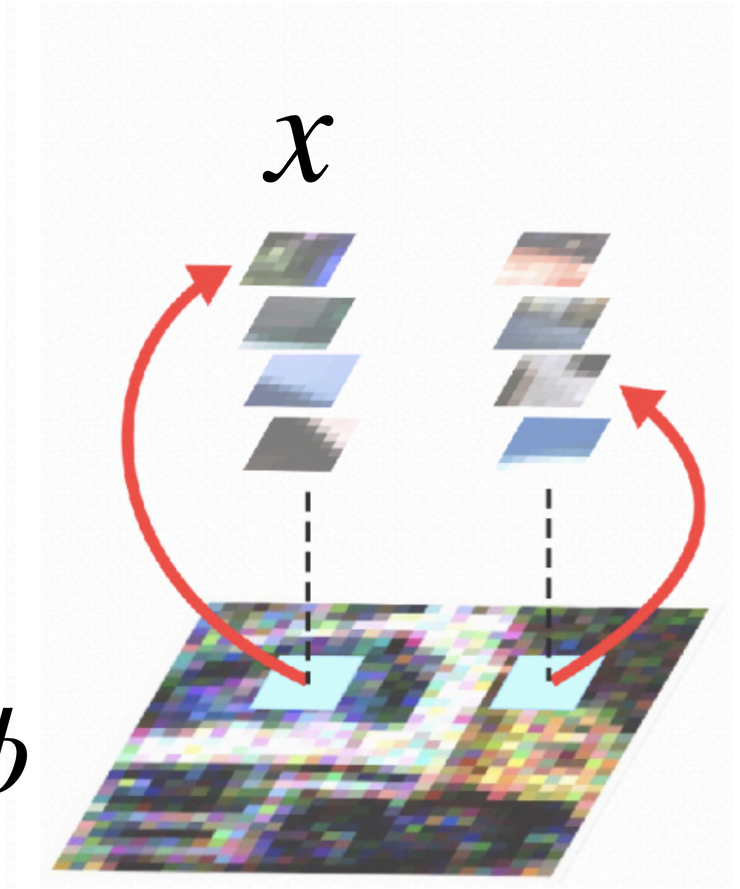
And score for pixel at x
in image ϕ as $M_t[\phi](x)$



Idea: Make Score Machine *Local*

$$M_t[\phi](x) = \sum_{\varphi \in \mathcal{D}} \frac{(\sqrt{\bar{\alpha}_t}\varphi(x) - \phi(x))}{1 - \bar{\alpha}_t} W_t(\varphi_{\Omega_x} | \phi_{\Omega_x})$$

$$W_t(\varphi_{\Omega_x} | \phi_{\Omega_x}) = \frac{\mathcal{N}(\phi_{\Omega_x} | \sqrt{\bar{\alpha}_t}\varphi_{\Omega_x}, (1 - \bar{\alpha}_t)I)}{\sum_{\varphi' \in \mathcal{D}} \mathcal{N}(\phi_{\Omega_x} | \sqrt{\alpha}_t \varphi'_{\Omega_x}, (1 - \bar{\alpha}_t)I)}$$

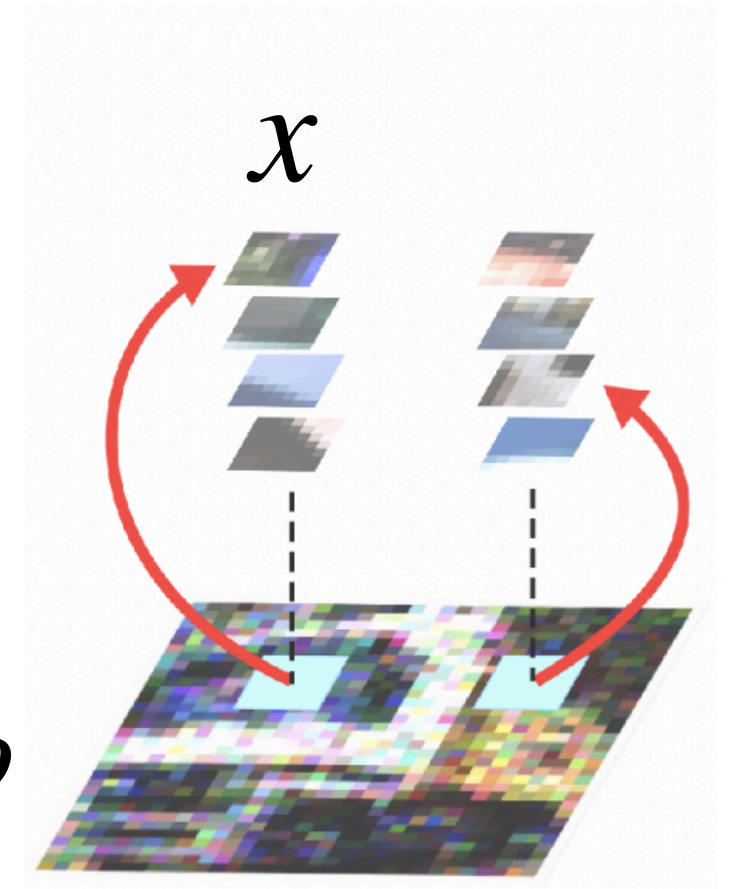


Idea: Make Score Machine *Local*

$$M_t[\phi](x) = \sum_{\varphi \in \mathcal{D}} \frac{(\sqrt{\bar{\alpha}_t}\varphi(x) - \phi(x))}{1 - \bar{\alpha}_t} W_t(\varphi_{\Omega_x} | \phi_{\Omega_x})$$

$$W_t(\varphi_{\Omega_x} | \phi_{\Omega_x}) = \frac{\mathcal{N}(\phi_{\Omega_x} | \sqrt{\bar{\alpha}_t}\varphi_{\Omega_x}, (1 - \bar{\alpha}_t)I)}{\sum_{\varphi' \in \mathcal{D}} \mathcal{N}(\phi_{\Omega_x} | \sqrt{\alpha}_t \varphi'_{\Omega_x}, (1 - \bar{\alpha}_t)I)}$$

Pixel at x in image ϕ is pulled towards training set pixel values $\varphi(x)$ according to similarities of effective receptive fields Ω_x.

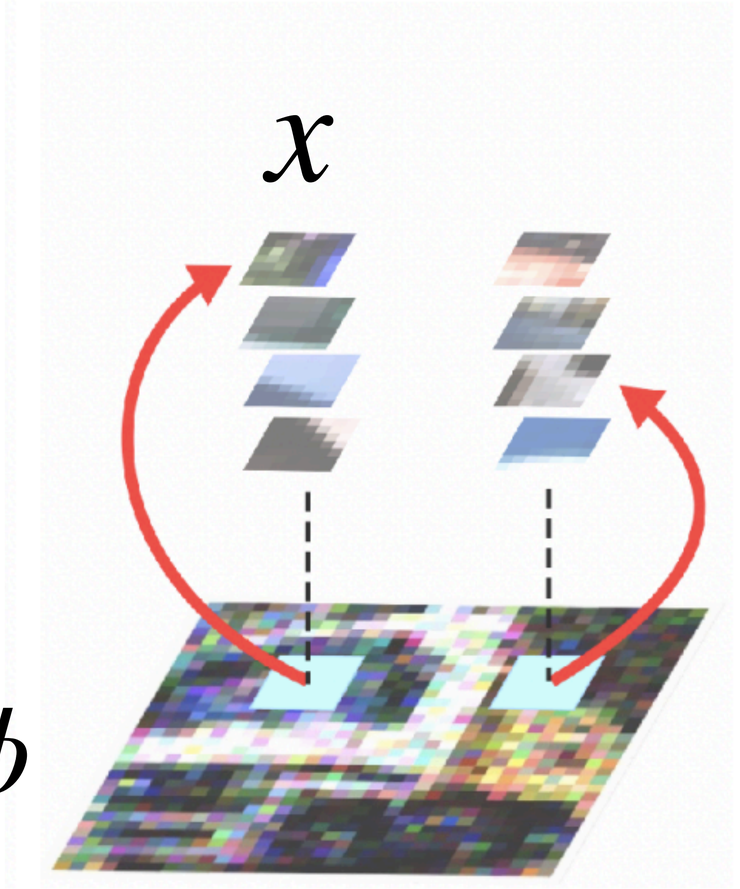


Idea: Make Score Machine *Local*

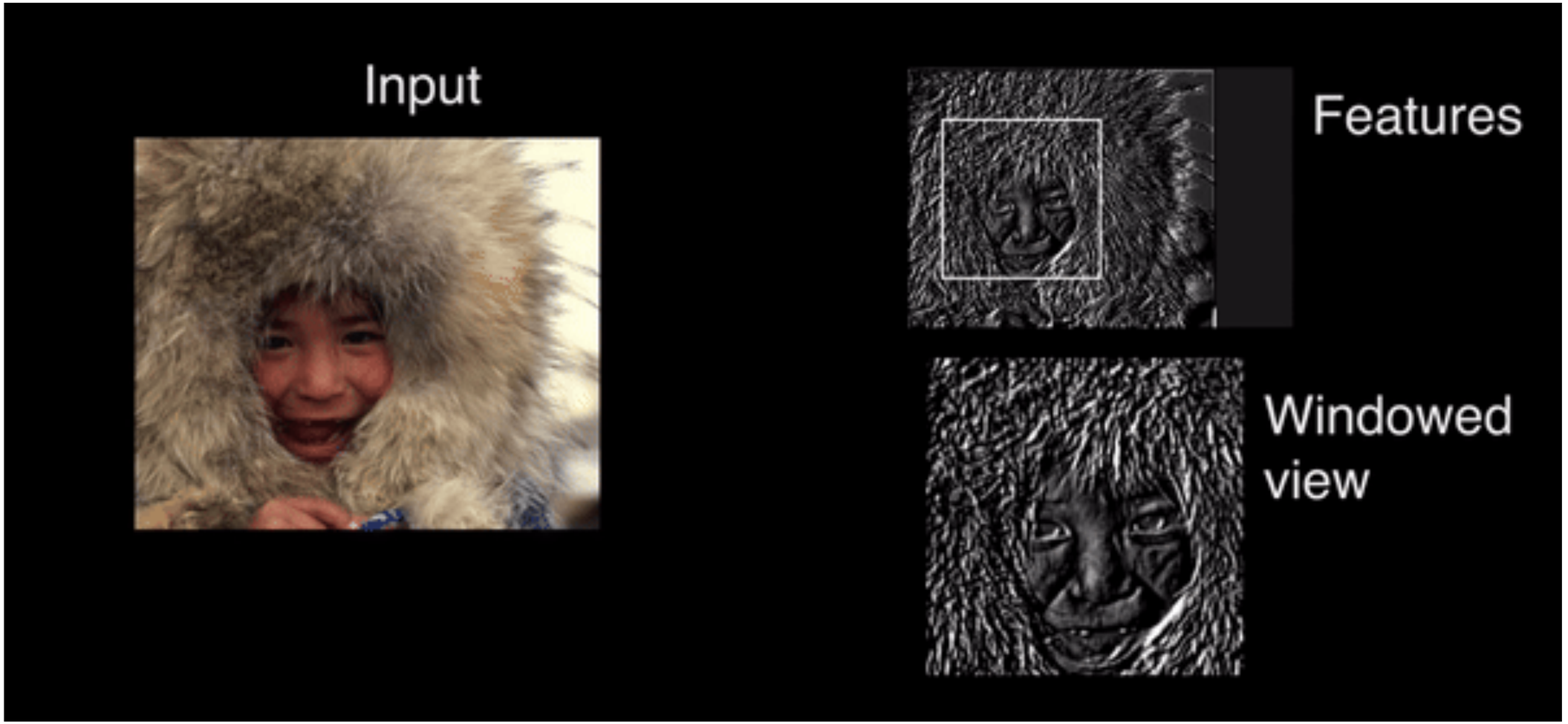
$$M_t[\phi](x) = \sum_{\varphi \in \mathcal{D}} \frac{(\sqrt{\bar{\alpha}_t}\varphi(x) - \phi(x))}{1 - \bar{\alpha}_t} W_t(\varphi_{\Omega_x} | \phi_{\Omega_x})$$

What has changed?

The softmax weights can now be *different* for each pixel in the image!

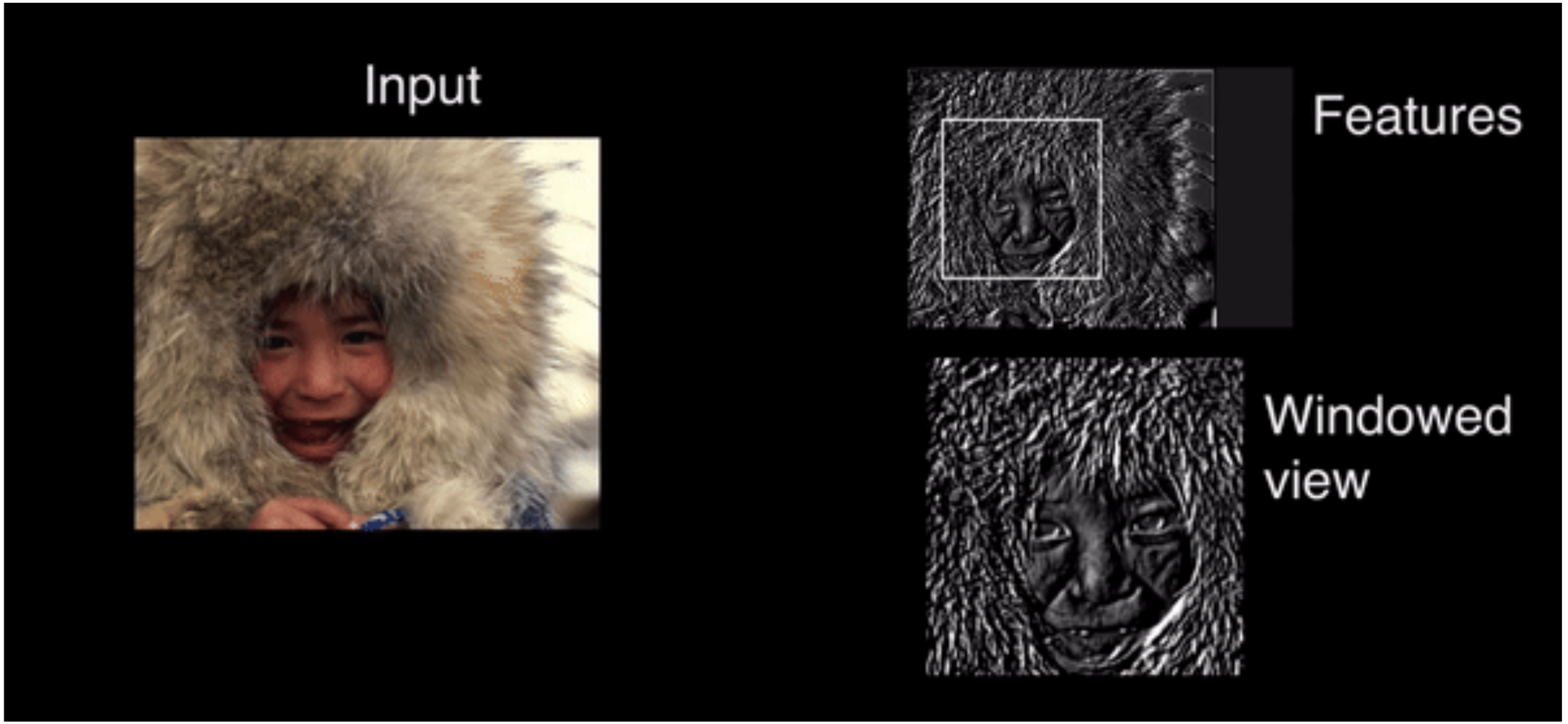


CNNs are *shift-equivariant!*



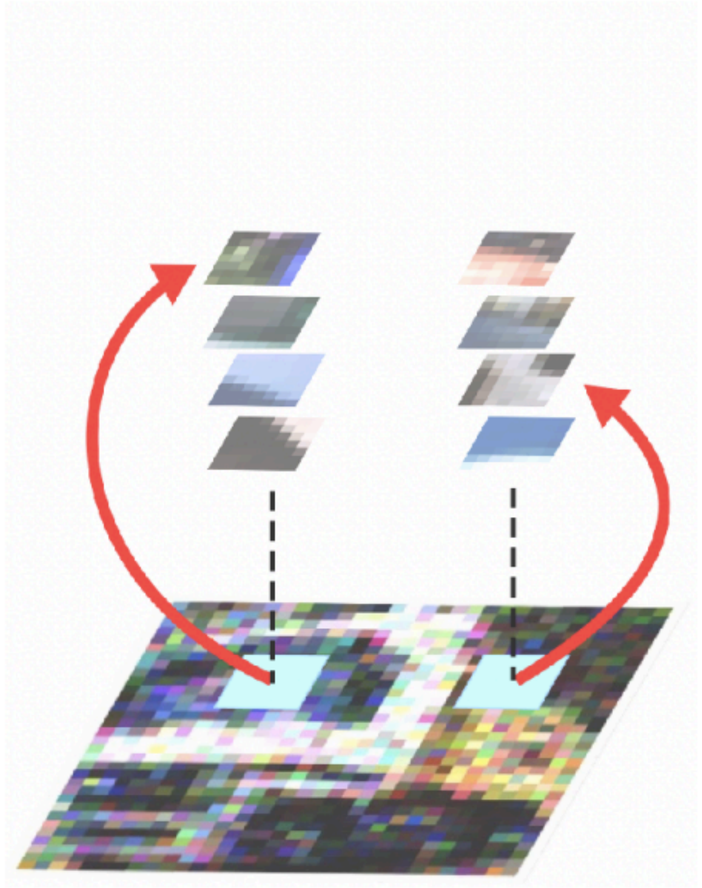
*** CNNs and Equivariance, Fabian Fuchs**

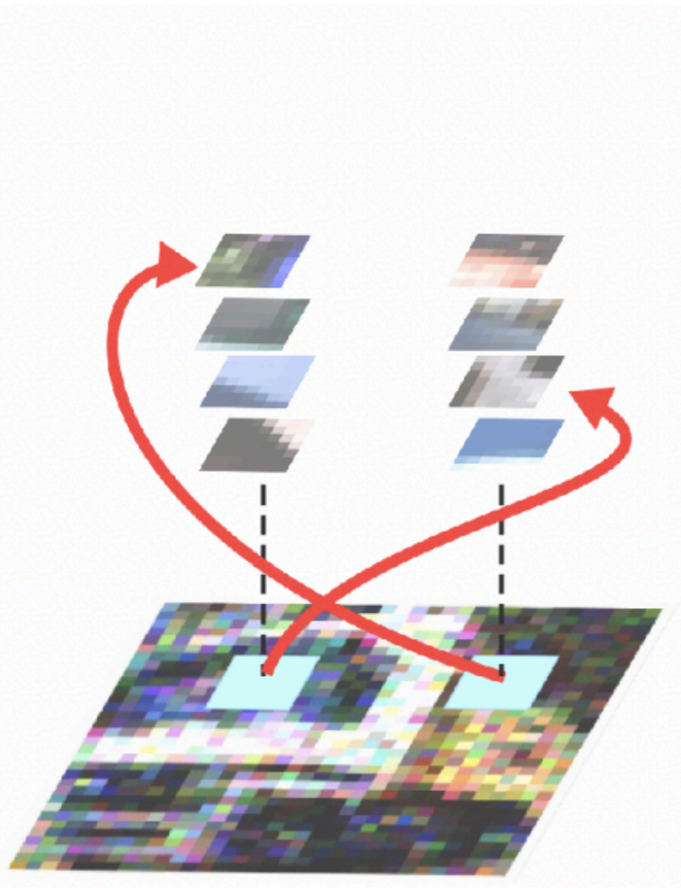
CNNs are *shift-equivariant!*



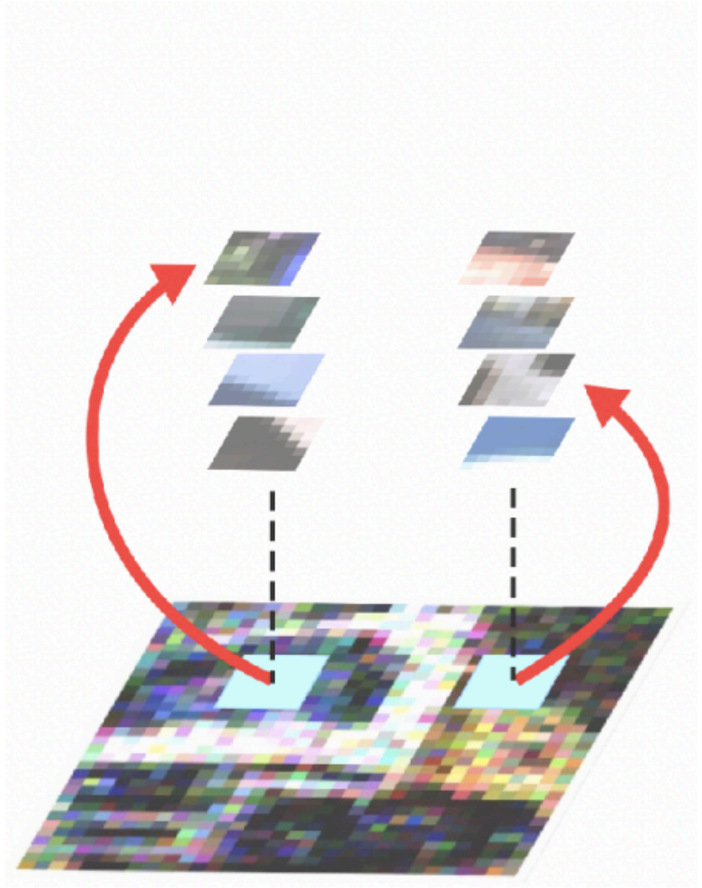
*** CNNs and Equivariance, Fabian Fuchs**

Idea: Make Local Score Machine
Shift-Equivariant

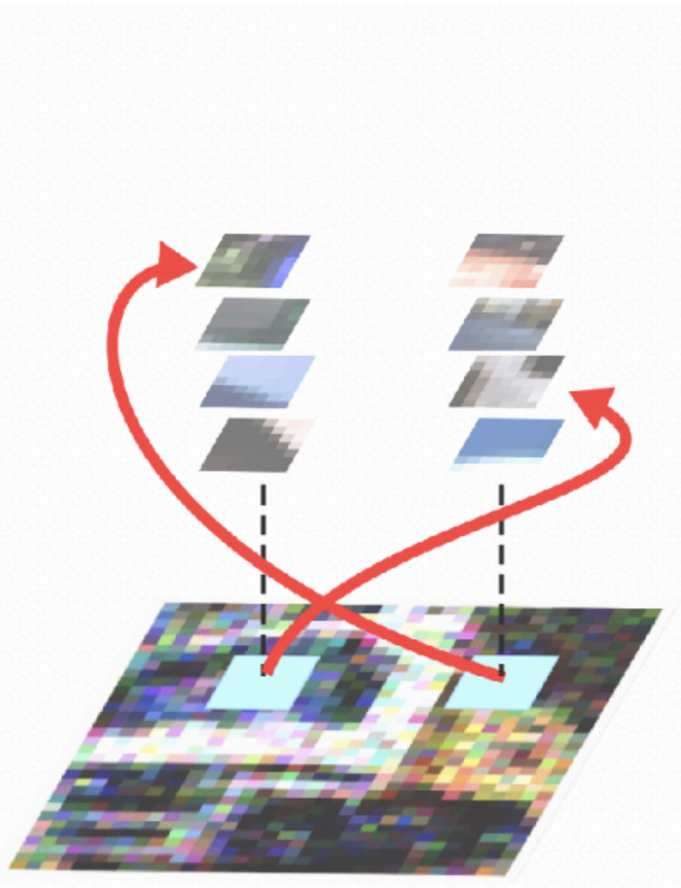


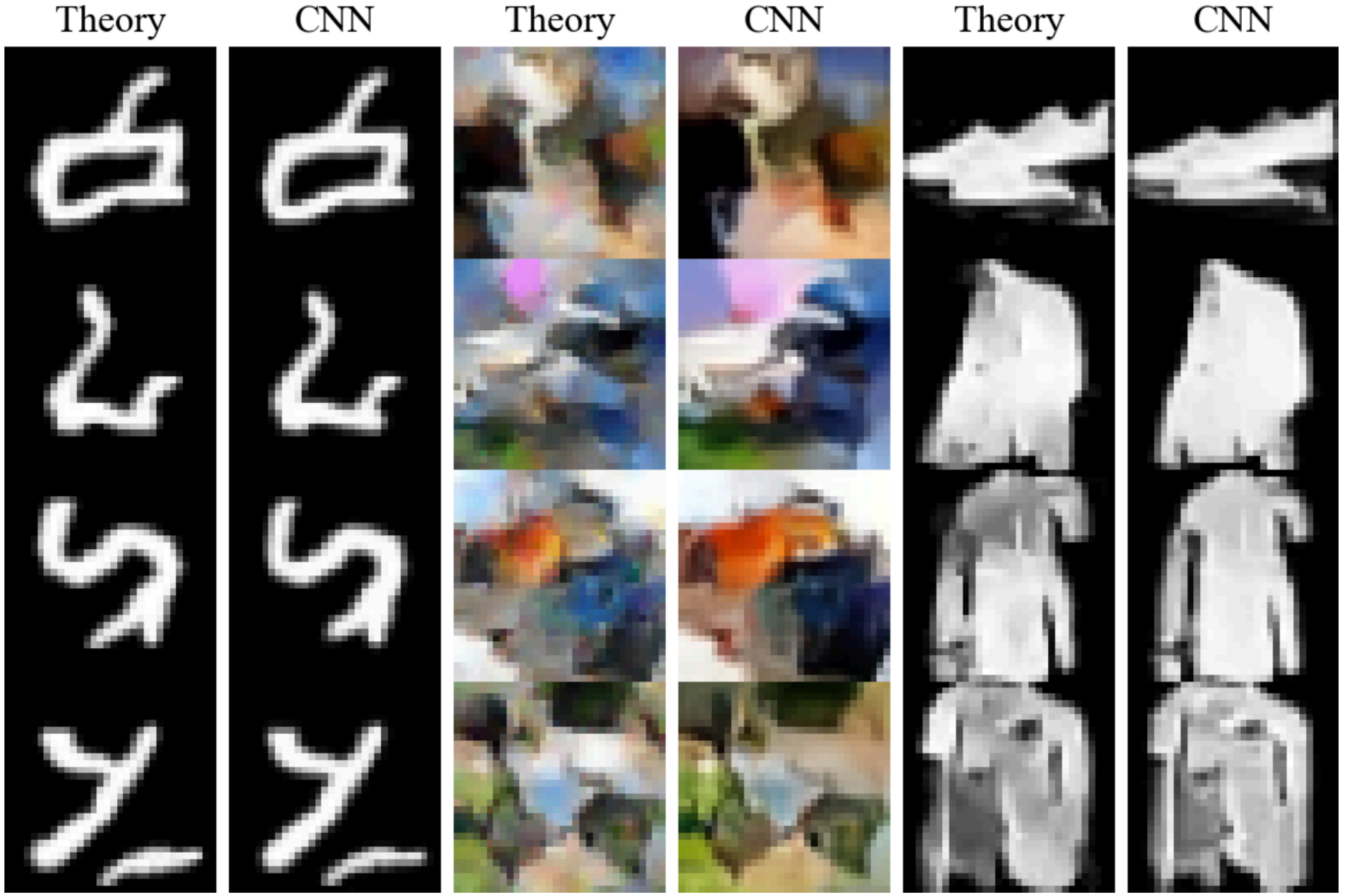


Idea: Make Local Score Machine *Shift-Equivariant*



The diffusion model doesn't know where in the image the patches came from!





In many ways images are *worse* than the prior ones on CelebA, but impressively, *they predict the behavior of a trained U-Net Diffusion Model.*

Caveats - this only explains “hobbled” UNets

The trained U-Net is a very *hobbled* U-Net. Adding a few standard components (layer norm, more layers, etc) dramatically improves performance in ways not explained by the closed-form model.

The model does not account for any kind of compression at all; but clearly, the UNet *cannot* store all the training images for large datasets.

Caveats - this only explains “hobbled” UNets

But, core insight: A theory of “creativity” in these models likely exists, it is *not* magic, and I’d be shocked if in 5 years from now, someone will not learn about it in this very class ;)

components (layer norm, more layers, etc) dramatically improves performance in ways not explained by the closed-form model.

The model does not account for any kind of compression at all; but clearly, the UNet *cannot* store all the training images for large datasets.