BOSTON UNIVERSITY

Deep Learning for Data Science DS 542

Lecture 23
Diffusion Models

Slides originally by Thomas Gardos.
Images from [Understanding Deep Learning]() unless otherwise cited.

Final Projects

- Mail me this week about how your project is doing.
  - What are you currently working on?
  - What is working?
  - What needs help?

- Grading
  - Did you build / finetune an appropriate model for your problem?
  - What does your evaluation say you got out of the training process?
  - How does your model advance the real goals of your project?

Evaluating Your Models

- Basic training/validation statistics (loss, accuracy, etc)
- Final statistics
- Comparison to benchmarks
  - Original work if reproducing a paper.
  - Current state of the art work on the same problem.
- To the extent possible,
  - What modeling choices helped or hindered your model performance?
  - Quantitative comparison before speculation on why.
  - If reproducing a paper, you can vary parameter choices and assess the impact.
    - Could you find better settings than the original authors?
    - Repeat with different seeds for more confidence.

Generative Models

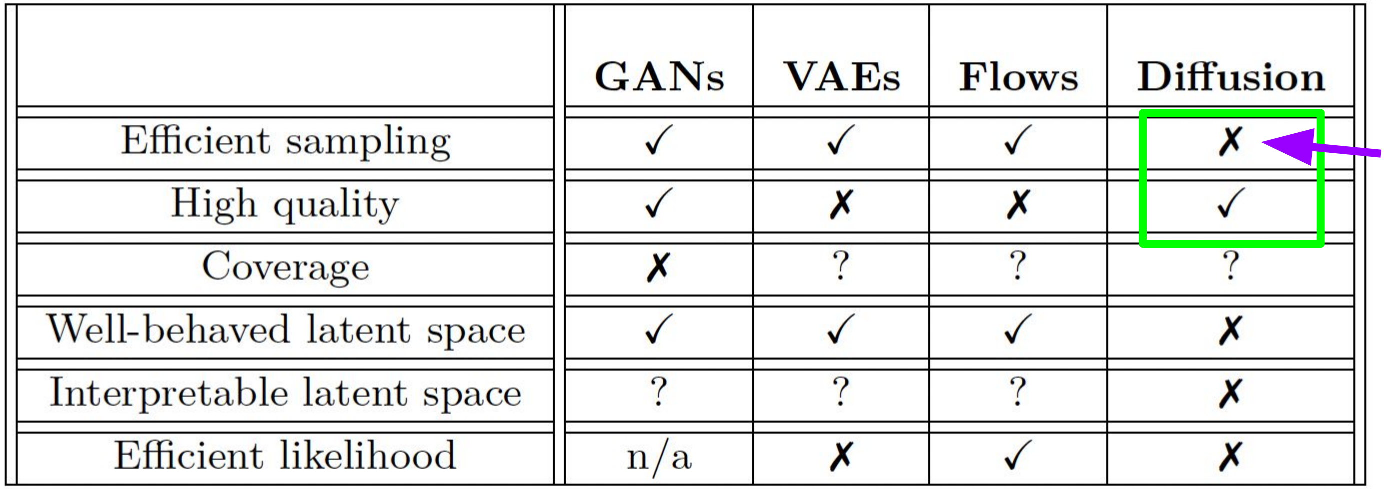
Last Time

- Normalizing Flows
  - Key design change is invertibility of each layer
  - Enables efficient probability computations

This Time

- Diffusion Models
  - Very high quality (after a few false starts)
  - Very fast (after moving to latent space)

Do we have good models?



	GANs	VAEs	Flows	Diffusion
Efficient sampling	✓	✓	✓	✗
High quality	✓	✗	✗	✓
Coverage	✗	?	?	?
Well-behaved latent space	✓	✓	✓	✗
Interpretable latent space	?	?	?	✗
Efficient likelihood	n/a	✗	✓	✗

Gets fast after switch to latent space.

Diffusion Models

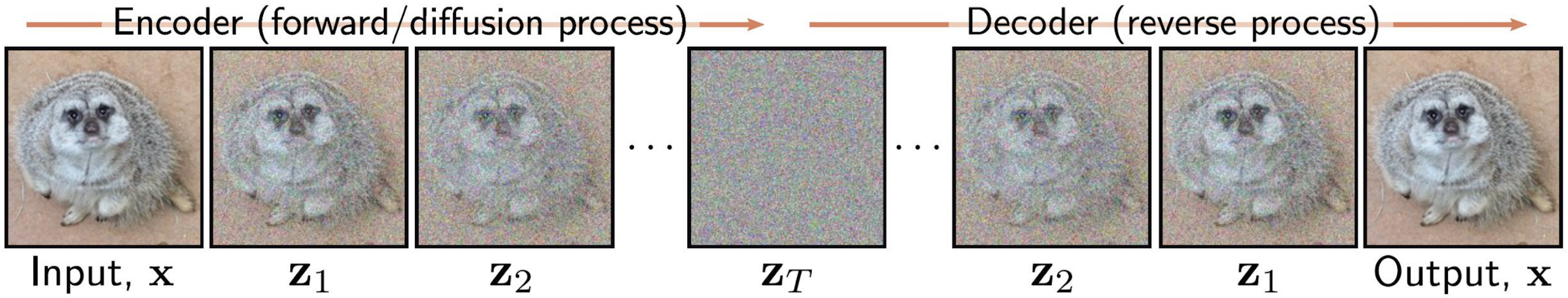


Figure 18.1 Diffusion models. The encoder (forward, or diffusion process) maps the input $\mathbf{x}$ through a series of latent variables $\mathbf{z}_1 \ldots \mathbf{z}_T$. This process is prespecified and gradually mixes the data with noise until only noise remains. The decoder (reverse process) is learned and passes the data back through the latent variables, removing noise at each stage. After training, new examples are generated by sampling noise vectors $\mathbf{z}_T$ and passing them through the decoder.

Diffusion Models - TLDR

TLDR

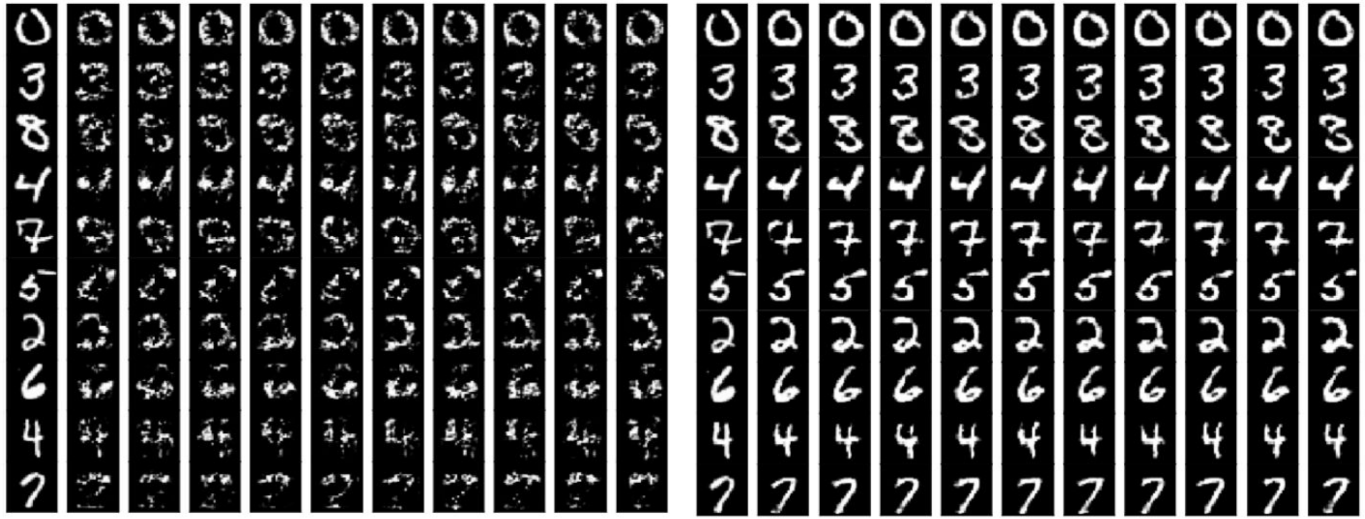
- Error diffusion adds noise everywhere over several steps.
- Reverse process removes it.
- Reverse process is a generative model.

Slightly more detail:

- Forward process repeatedly adds a little bit of Gaussian noise to every pixel.
  - This is easy.
- Reverse process repeatedly removes that noise.
  - This is hard.

Denoising Autoencoders

Early predecessors, but could not handle nearly as much noise.



(a) SAE (b) SDAE

“Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion” by Vincent et al (2010)

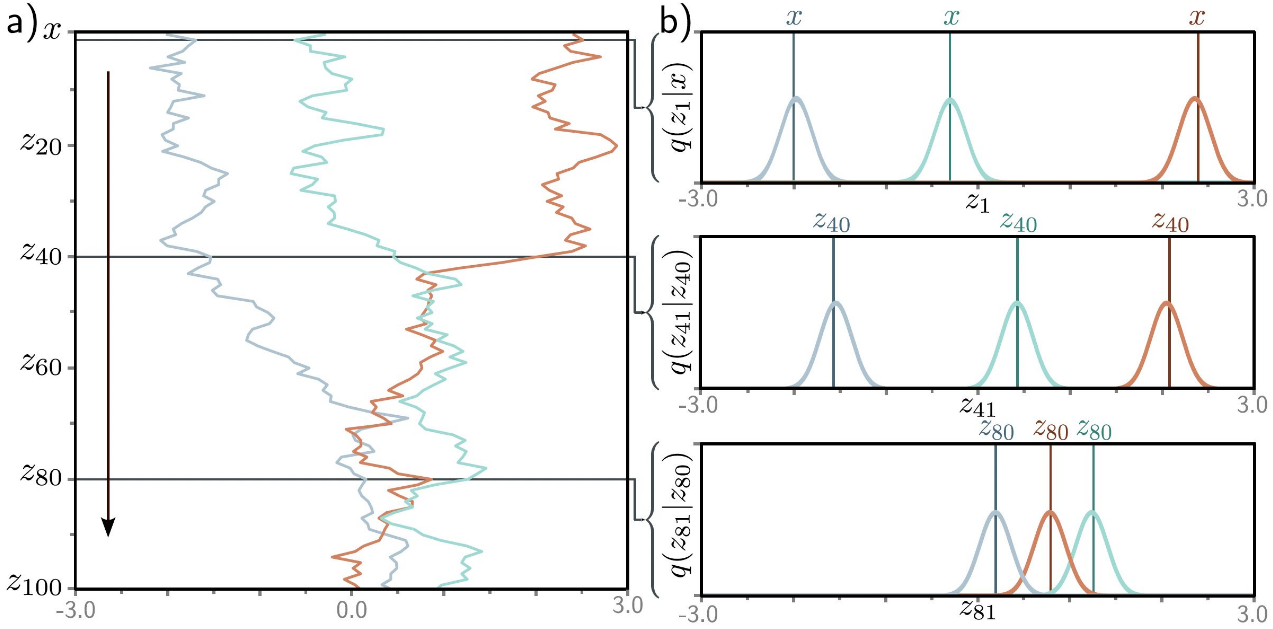


Figure 18.2 Forward process. a) We consider one-dimensional data x with $T = 100$ latent variables $z_1, \ldots, z_{100}$ and $\beta = 0.03$ at all steps. Three values of x (orange, green, and gray) are initialized (top row). These are propagated through $z_1, \ldots, z_{100}$. At each step, the variable is updated by attenuating its value by $\sqrt{1-\beta}$ and adding noise with mean zero and variance β (equation 18.1). Accordingly, the three examples noisily propagate through the variables with a tendency to move toward zero. b) The conditional probabilities $Pr(z_1|x)$ and $Pr(z_t|z_{t-1})$ are normal distributions with a mean that is slightly closer to zero than the current point and a fixed variance β_t (equation 18.2).

Diffusion Models

After many rounds of the forward process,

- All images have become pure noise?
  - close
- All images look like they are drawn from the same noise distribution
  - Almost
- All images drawn from similar noise distribution?
  - Yes
  - Original image survives with a low weight.
  - Noise has higher magnitude than original image.

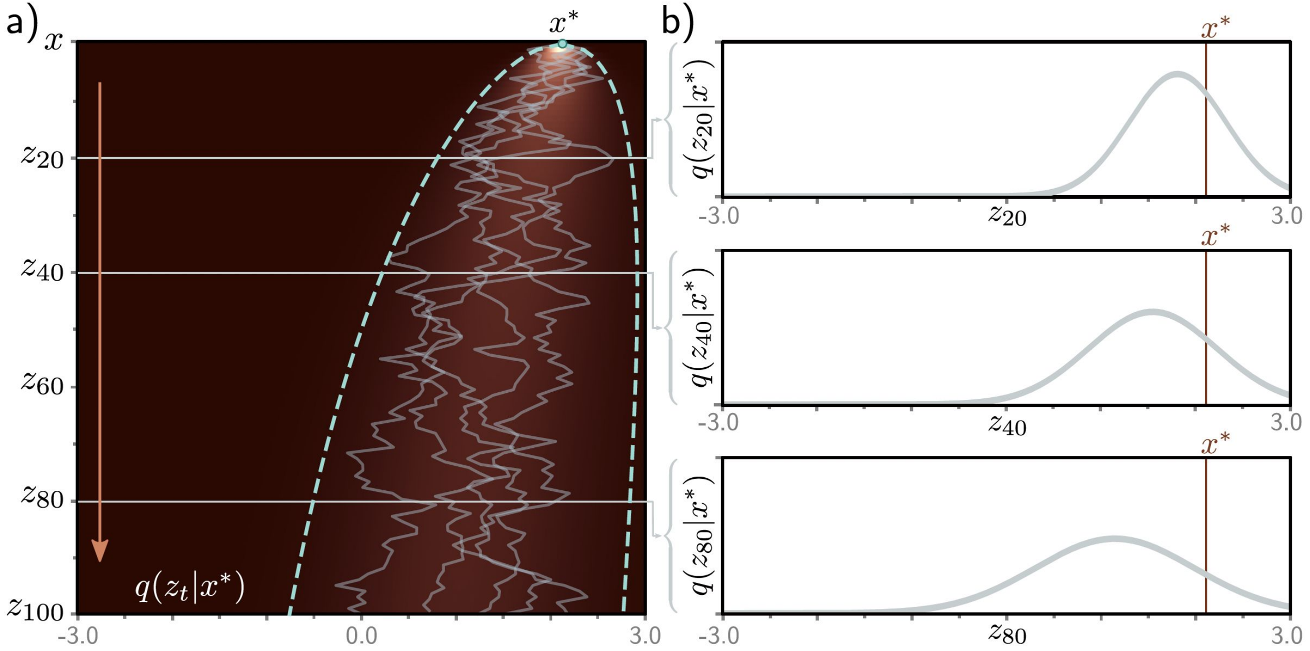


Figure 18.3 Diffusion kernel. a) The point $x^* = 2.0$ is propagated through the latent variables using equation 18.1 (five paths shown in gray). The diffusion kernel $q(z_t|x^*)$ is the probability distribution over variable z_t given that we started from x^*. It can be computed in closed-form and is a normal distribution whose mean moves toward zero and whose variance increases as t increases. Heatmap shows $q(z_t|x^*)$ for each variable. Cyan lines show ± 2 standard deviations from the mean. b) The diffusion kernel $q(z_t|x^*)$ is shown explicitly for $t = 20, 40, 80$. In practice, the diffusion kernel allows us to sample a latent variable z_t corresponding to a given x^* without computing the intermediate variables $z_1, \ldots, z_{t-1}$. When t becomes very large, the diffusion kernel becomes a standard normal.

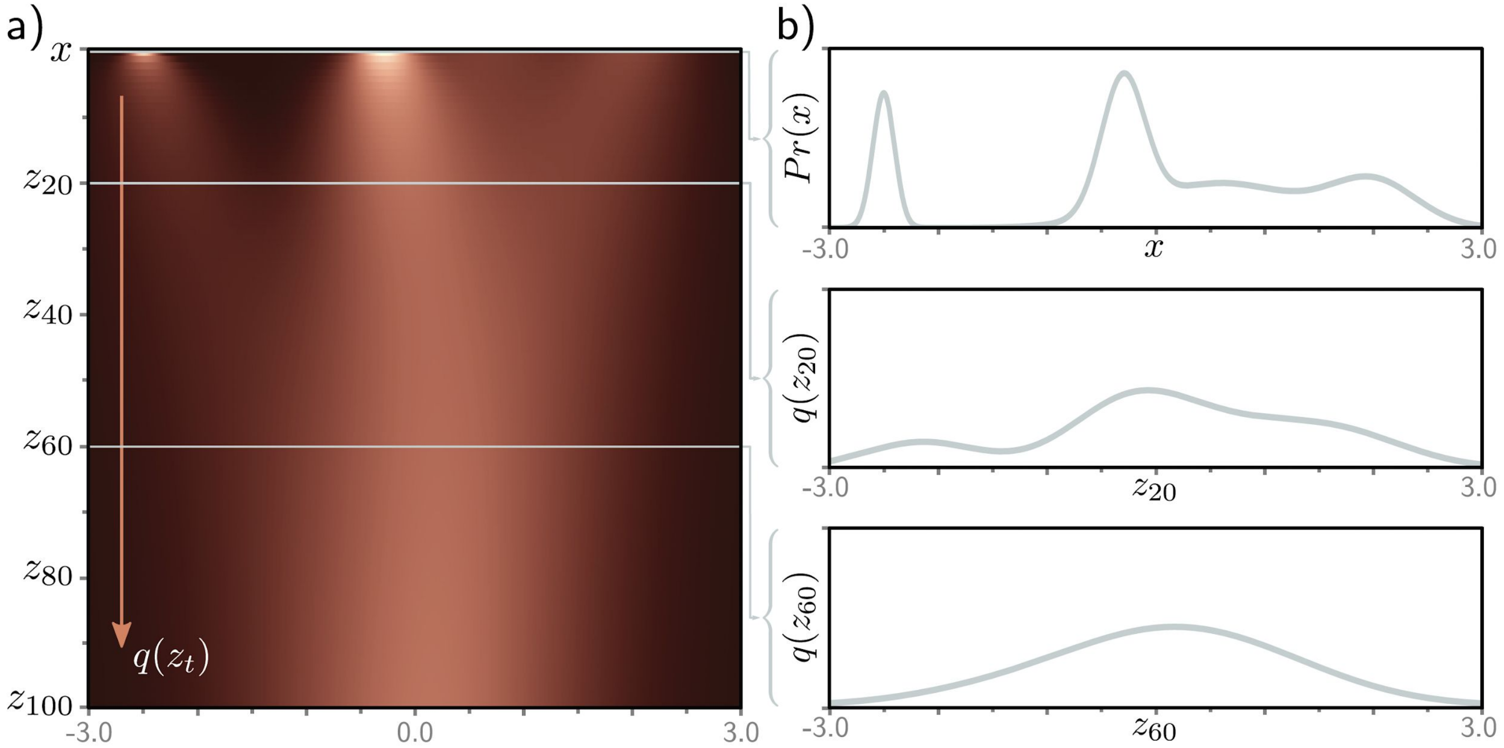
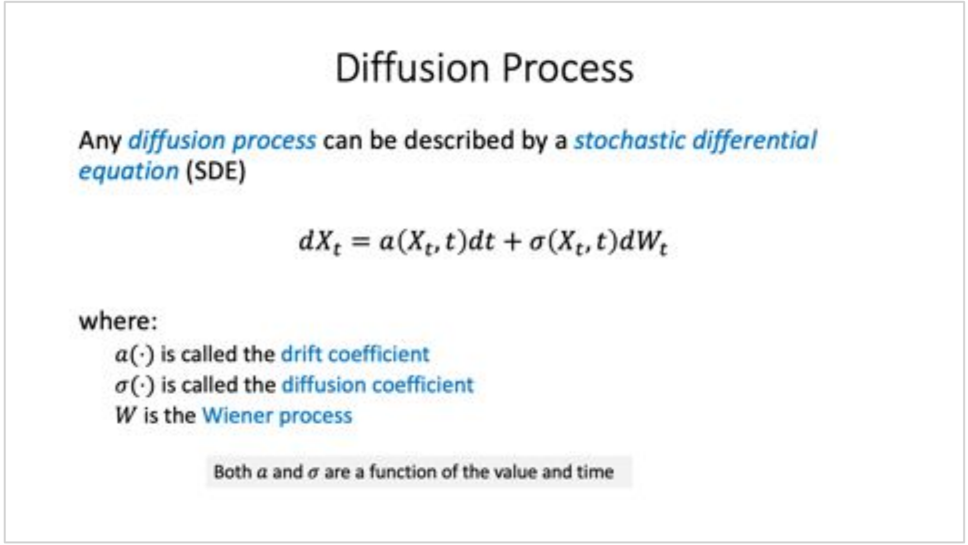


Figure 18.4 Marginal distributions. a) Given an initial density $Pr(x)$ (top row), the diffusion process gradually blurs the distribution as it passes through the latent variables z_t and moves it toward a standard normal distribution. Each subsequent horizontal line of heatmap represents a marginal distribution $q(z_t)$. b) The top graph shows the initial distribution $Pr(x)$. The other two graphs show the marginal distributions $q(z_{20})$ and $q(z_{60})$, respectively.

Mathematics of Diffusion Models

Assume the forward and reverse process operate in T steps.

Both forward and reverse process are discrete so becomes a *Markov chain* with *gaussian transition probability*.



Diffusion Process

Any *diffusion process* can be described by a *stochastic differential equation* (SDE)

$$dX_t = a(X_t, t)dt + \sigma(X_t, t)dW_t$$

where:

$a(\cdot)$ is called the drift coefficient
$\sigma(\cdot)$ is called the diffusion coefficient
W is the Wiener process

Both a and σ are a function of the value and time

$\boxed{\mathcal{N}(\mu, \sigma^2)}$

Mathematics of Diffusion Models

Denote x_0 as a sample from a distribution $q(x_0)$.

Forward process: gaussian transition probability

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{(1-\beta_t)}\, x_{t-1}, \beta_t I) \quad \text{where} \quad t \in \mathbb{N}$$

and where β_t indicates trade-off between info to be kept from previous step and new noise added.

Rocca, 2022

$\boxed{\mathcal{N}(\mu, \sigma^2)}$

Mathematics of Diffusion Models

Denote x_0 as a sample from a distribution $q(x_0)$.

Forward process: gaussian transition probability

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{(1-\beta_t)}\, x_{t-1}, \beta_t I) \quad \text{where} \quad t \in \mathbb{N}$$

and where β_t indicates trade-off between info to be kept from previous step and new noise added.

We can equivalently write

$$x_t = \sqrt{(1-\beta_t)}\, x_{t-1} + \sqrt{\beta_t}\, \epsilon_t \qquad \epsilon_t \sim \mathcal{N}(0, I)$$

Discretized diffusion process

Rocca, 2022

$\boxed{\mathcal{N}(\mu, \sigma^2)}$

Mathematics of Diffusion Models

Through recurrence, we can represent any step in the chain as directly represented from x_0:

$$q(x_t|x_0) = \mathcal{N}(x_t;\ \sqrt{\overline{\alpha_t}}\, x_0,\ (1 - \bar{\alpha}_t)I)$$

So we can run many forward steps at once.

where

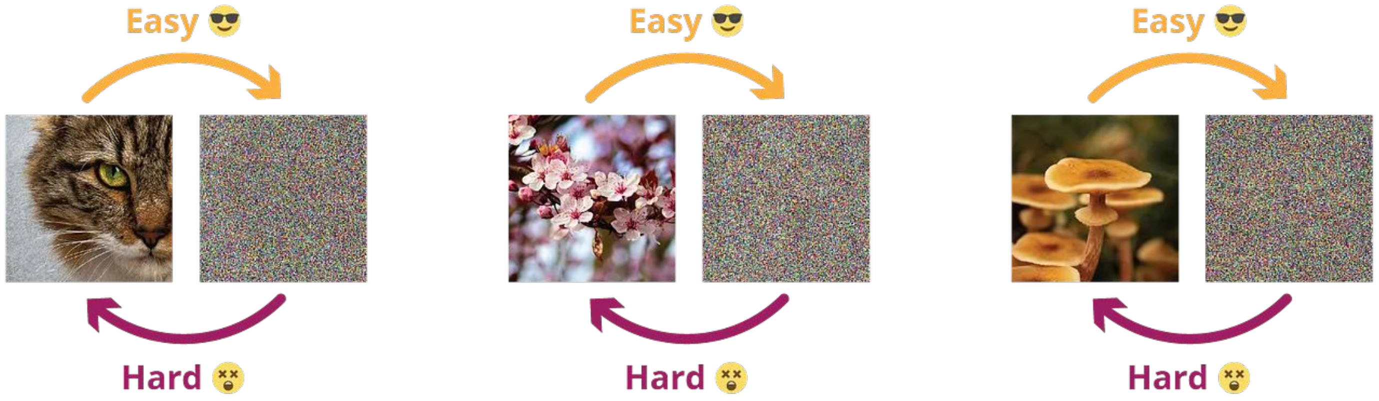
$$\alpha_t = (1 - \beta_t) \quad \text{and} \quad \bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i = \prod_{i=1}^{t} (1 - \beta_i)$$

and from the Markov property, the entire forward trajectory is

$$q(x_{0:T}) = q(x_0) \prod_{t=1}^{T} q(x_t|x_{t-1})$$

Rocca, 2022

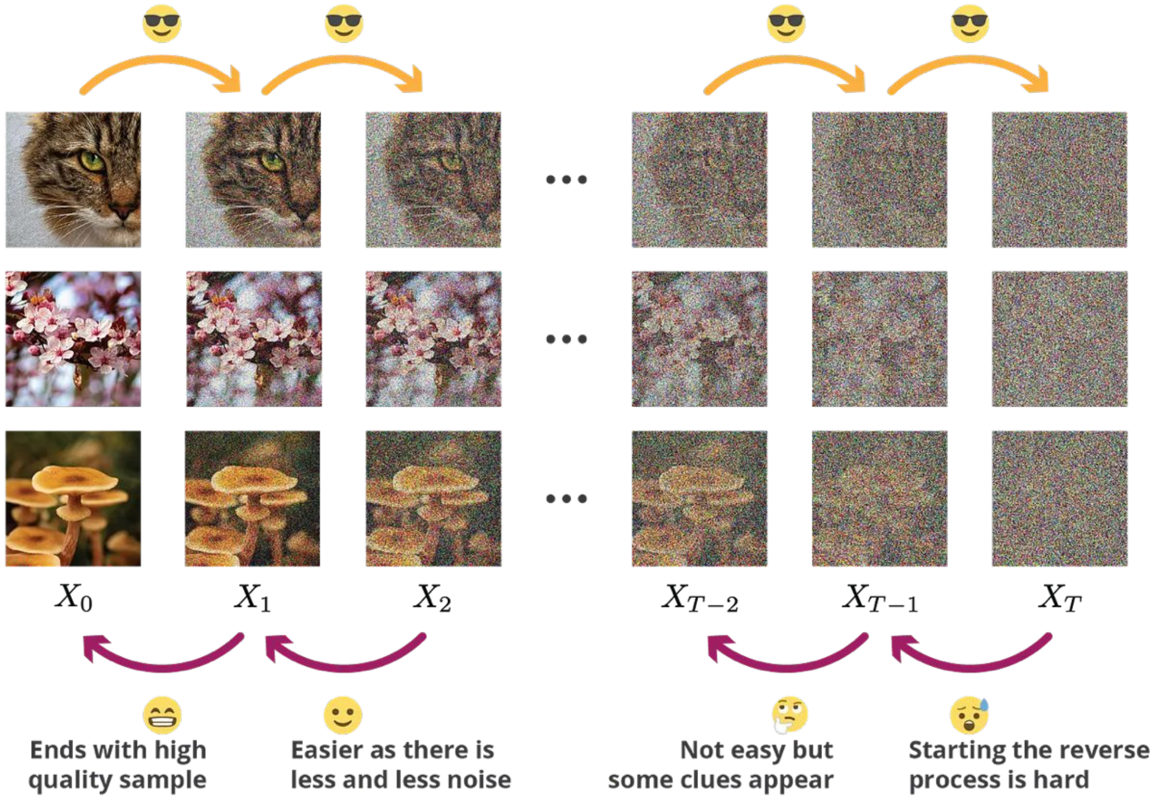
Intuition behind learning the reverse process



Reversing process in one step is extremely difficult

Rocca, 2022

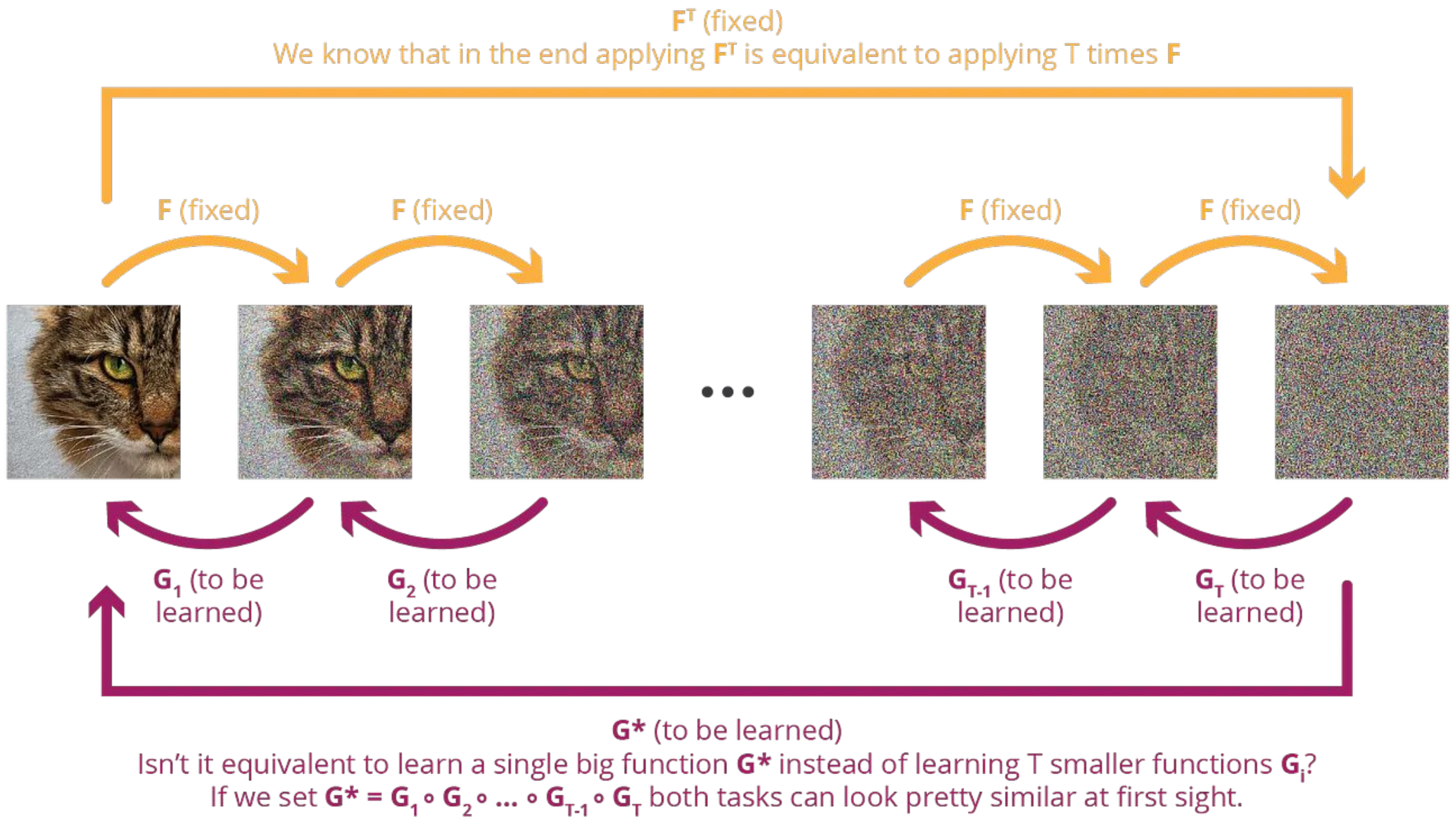
Doing it in steps gives us some clues



X_0
X_1
X_2
X_{T-2}
X_{T-1}
X_T
Ends with high quality sample
Easier as there is less and less noise
Not easy but some clues appear
Starting the reverse process is hard

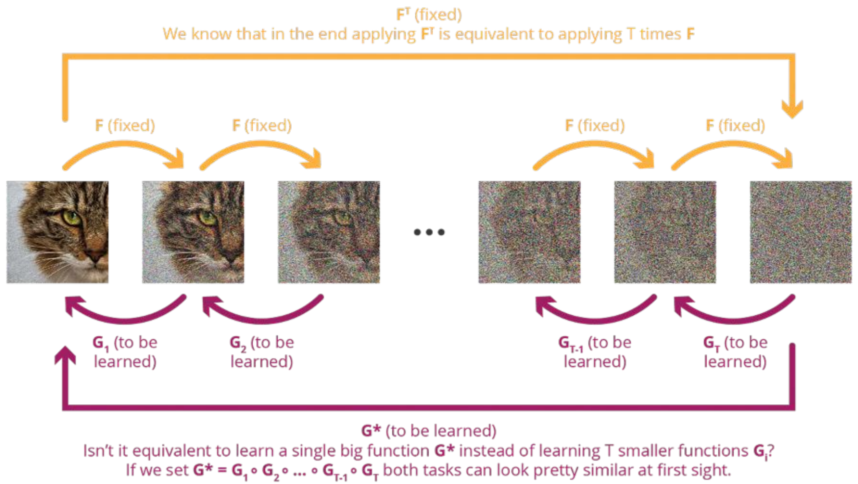
Rocca, 2022

One step versus multi-step


F^T (fixed)
We know that in the end applying F^T is equivalent to applying T times **F**
F (fixed)
F (fixed)
F (fixed)
F (fixed)
...
G_1 (to be learned)
G_2 (to be learned)
G_{T-1} (to be learned)
G_T (to be learned)
G* (to be learned)
Isn't it equivalent to learn a single big function **G*** instead of learning T smaller functions G_i?
If we set $G^* = G_1 \circ G_2 \circ \ldots \circ G_{T-1} \circ G_T$ both tasks can look pretty similar at first sight.

Rocca, 2022

Advantage of multi-step reverse process



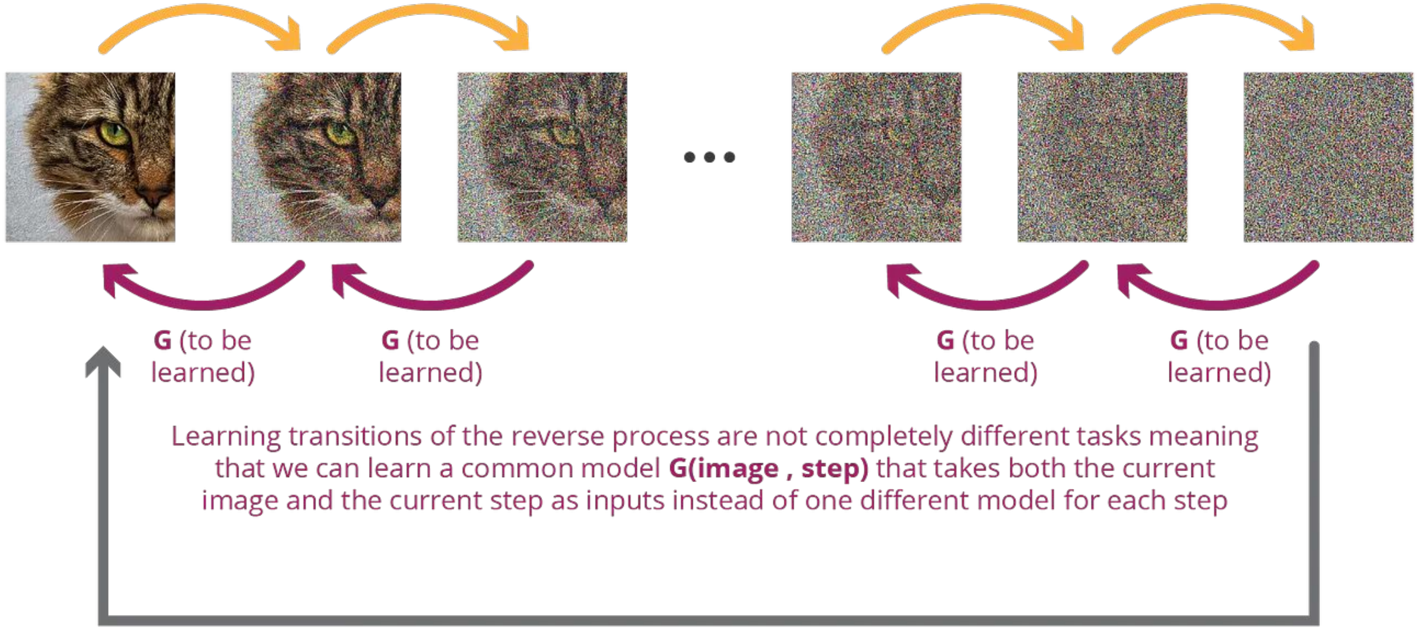
F^T (fixed)
We know that in the end applying F^T is equivalent to applying T times F
F (fixed)
F (fixed)
F (fixed)
F (fixed)
...
G_1 (to be learned)
G_2 (to be learned)
G_{T-1} (to be learned)
G_T (to be learned)
G^* (to be learned)
Isn't it equivalent to learn a single big function G^* instead of learning T smaller functions G_i?
If we set $G^* = G_1 \circ G_2 \circ \ldots \circ G_{T-1} \circ G_T$ both tasks can look pretty similar at first sight.

1. Don't have to learn a unique transform G_i for each step, but rather a single transform that is a function of the index step. Drastically reduces size of the model.

2. Gradient descent is much more difficult in one step and can exploit coarse to fine adjustments in multiple steps.

Rocca, 2022

Iterative versus one step



Learning transitions of the reverse process are not completely different tasks meaning that we can learn a common model **G(image , step)** that takes both the current image and the current step as inputs instead of one different model for each step

G* (to be learned)

G* can't rely on the same nice iterative structure than G, meaning that this unrolled version supposed to be equivalent to $G_1 \circ G_2 \circ \ldots \circ G_{T-1} \circ G_T$ will have more parameters and will be harder to train

Rocca, 2022

The reverse process

With the assumption on the drift and diffusion coefficients, the reverse of the diffusion process takes the same form.

Reverse gaussian transition probability

$$q(x_{t-1}|x_t)$$

can then be approximated by

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

where μ_θ and Σ_θ are two functions parameterized by θ and learned.

Rocca, 2022

The reverse process

Using the Markov property, the probability of a given backward trajectory can be approximated by

$$\mathrm{p}_\theta(x_{0:T}) = p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$$

where $p(x_T)$ is an isotropic gaussian distribution that does not depend on θ

$$p(x_T) = \mathcal{N}(x_T; 0, I)$$

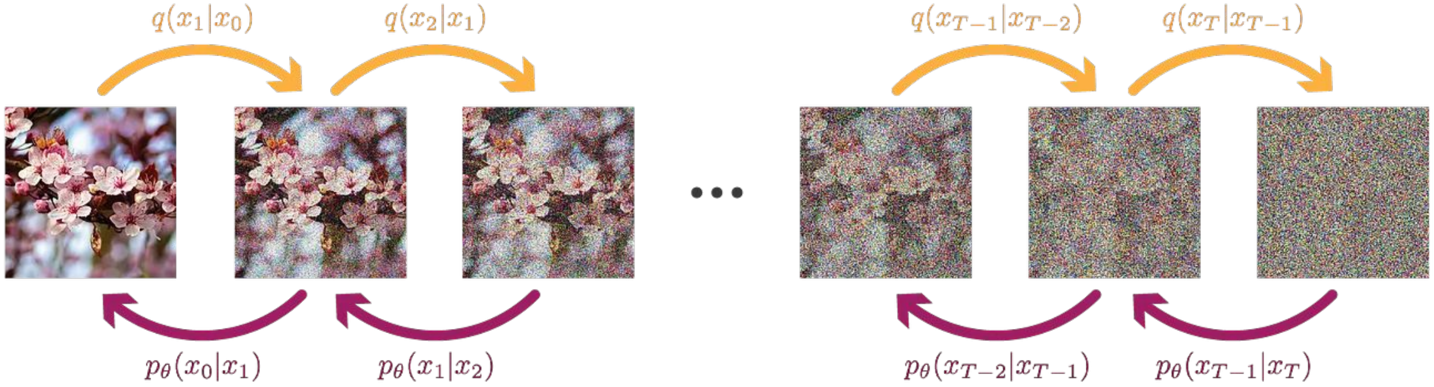
FIXED FORWARD PROCESS

Initial distribution

$q(x_0)$

Gaussian transition kernel

$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t} x_{t-1}, \beta_t I)$



Approximation of

$q(x_{t-1}|x_t)$

Gaussian transition kernel with parameters to be learned

$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$

Initial distribution

$p(x_T) = \mathcal{N}(x_t; 0, I)$

LEARNED BACKWARD PROCESS

Rocca, 2022

Questions

How do we learn the parameters θ for μ_θ and Σ_θ?

What is the loss to be optimized?

- We hope that $p_\theta(x_0)$, the distribution of the last step of the reverse process, will be close to $q(x_0)$

Optimization Objective

$$
\begin{aligned}
\mu_\theta^*, \Sigma_\theta^* &= \arg\min_{\mu_\theta, \Sigma_\theta} \left(D_{KL}(q(x_0) || p_\theta(x_0)) \right) \\
&= \arg\min_{\mu_\theta, \Sigma_\theta} \left(- \int q(x_0) \log \left(\frac{p_\theta(x_0)}{q(x_0)} \right) dx_0 \right) \\
&= \arg\min_{\mu_\theta, \Sigma_\theta} \left(- \int q(x_0) \log(p_\theta(x_0)) dx_0 \right)
\end{aligned}
$$

Skipping a lot more math

- Expand p-theta as marginalization integral
- Use Jensen’s inequality to define a slightly simpler upper bound to the loss
- Some manipulations with Bayes’ Theorem
- Properties of KL divergence of two gaussian distributions
- An additional simplification suggested by [Ho et al 2020]

Diffusion models in practice

We have the forward process

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I) \qquad q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I)$$

and our reverse process

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

and we want to train to minimize this simplified upper bound

$$\mathbb{E}_{x_0,t,\epsilon}\left(||\epsilon - \epsilon_\theta(x_t, t)||^2\right) = \mathbb{E}_{x_0,t,\epsilon}\left(||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)||^2\right)$$

Do We Really Need This Complicated Process?

Not a rhetorical question…

Why Can’t We Just Fit Function Directly?

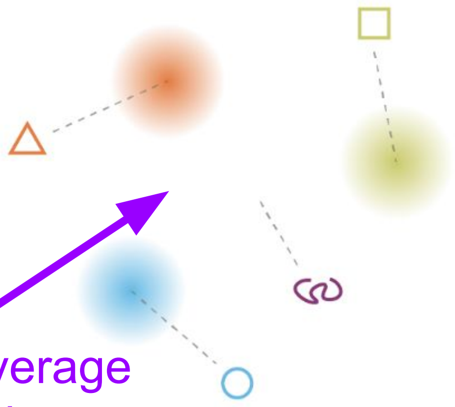
- The forward process can generate pairs of clean sample + noisy sample.
- So can we just train the reverse function by reversing inputs and outputs?
  - Train function: noisy sample → clean sample

Why Can’t We Just Fit Function Directly?

- The forward process can generate pairs of clean sample + noisy sample.
- So can we just train the reverse function by reversing inputs and outputs?
  - Train function: noisy sample → clean sample

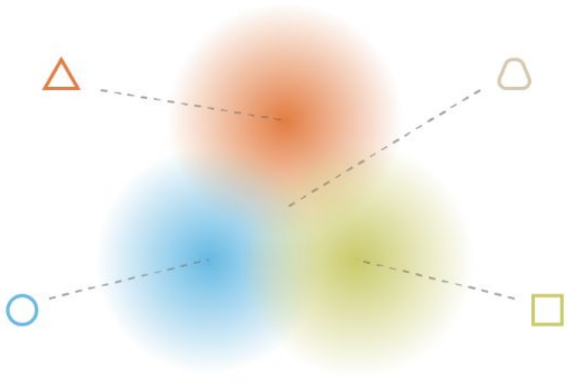
- Prone to overfitting instead of generalizing
  - Even if we generate many noisy samples per clean samples
  - Actually, probably will memorize those clean samples

Remember Variational Autoencoders?



Avoid coverage gaps like here.

without regularization ✗



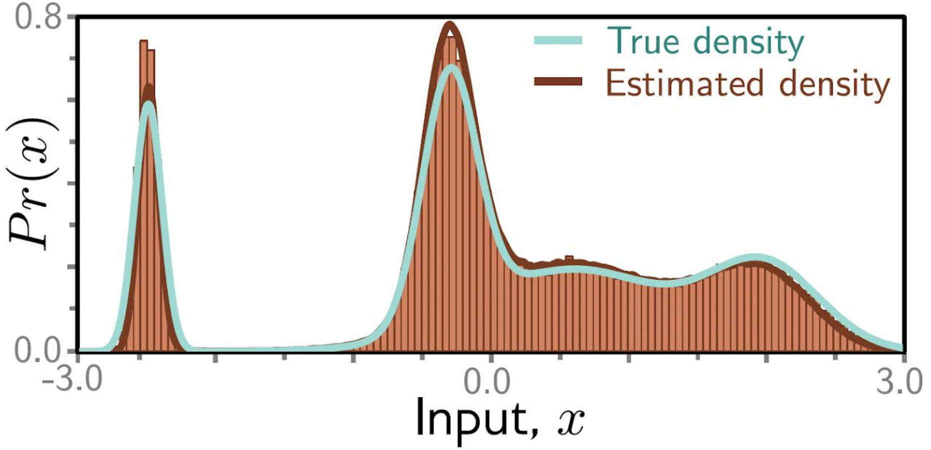
✓ with regularization

We have to regularize the means and the covariances too!
Regularize to a standard normal.

$$\text{loss} = \|x - \hat{x}\|^2 + \text{KL}[\,N(\mu_x, \sigma_x), N(0, I)\,]$$

Rocca, "Understanding Variational Autoencoders (VAEs)", 2019

Figure 18.8 Fitted model results. Cyan and brown curves are original and estimated densities and correspond to the top rows of figures 18.4 and 18.7, respectively. Vertical bars are binned samples from the model, generated by sampling from $Pr(\mathbf{z}_T)$ and propagating back through the variables $\mathbf{z}_{T-1}, \mathbf{z}_{T-2}, \ldots$ as shown for the five paths in figure 18.7.



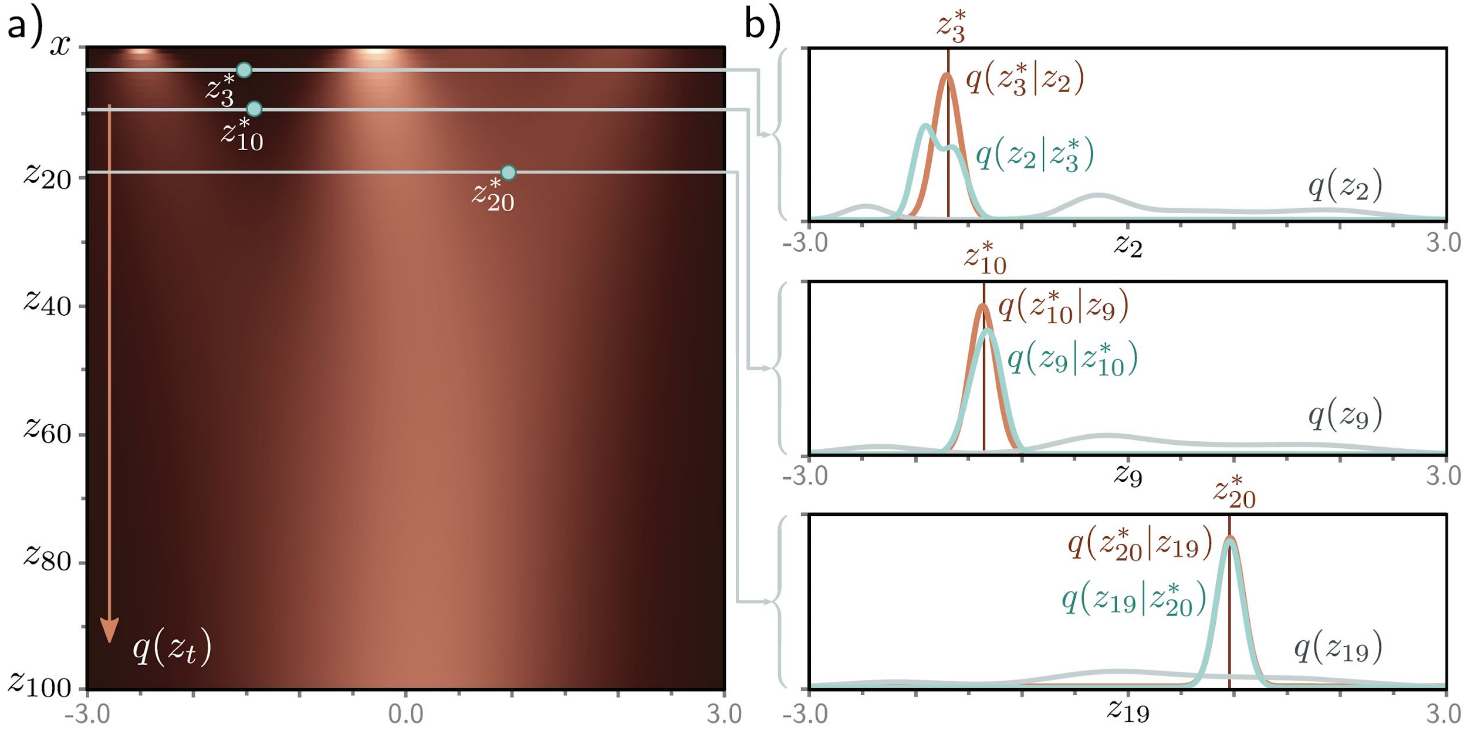


Figure 18.5 Conditional distribution $q(z_{t-1}|z_t)$. a) The marginal densities $q(z_t)$ with three points z_t^* highlighted. b) The probability $q(z_{t-1}|z_t^*)$ (cyan curves) is computed via Bayes' rule and is proportional to $q(z_t^*|z_{t-1})q(z_{t-1})$. In general, it is not normally distributed (top graph), although often the normal is a good approximation (bottom two graphs). The first likelihood term $q(z_t^*|z_{t-1})$ is normal in z_{t-1} (equation 18.2) with a mean that is slightly further from zero than z_t^* (brown curves). The second term is the marginal density $q(z_{t-1})$ (gray curves).

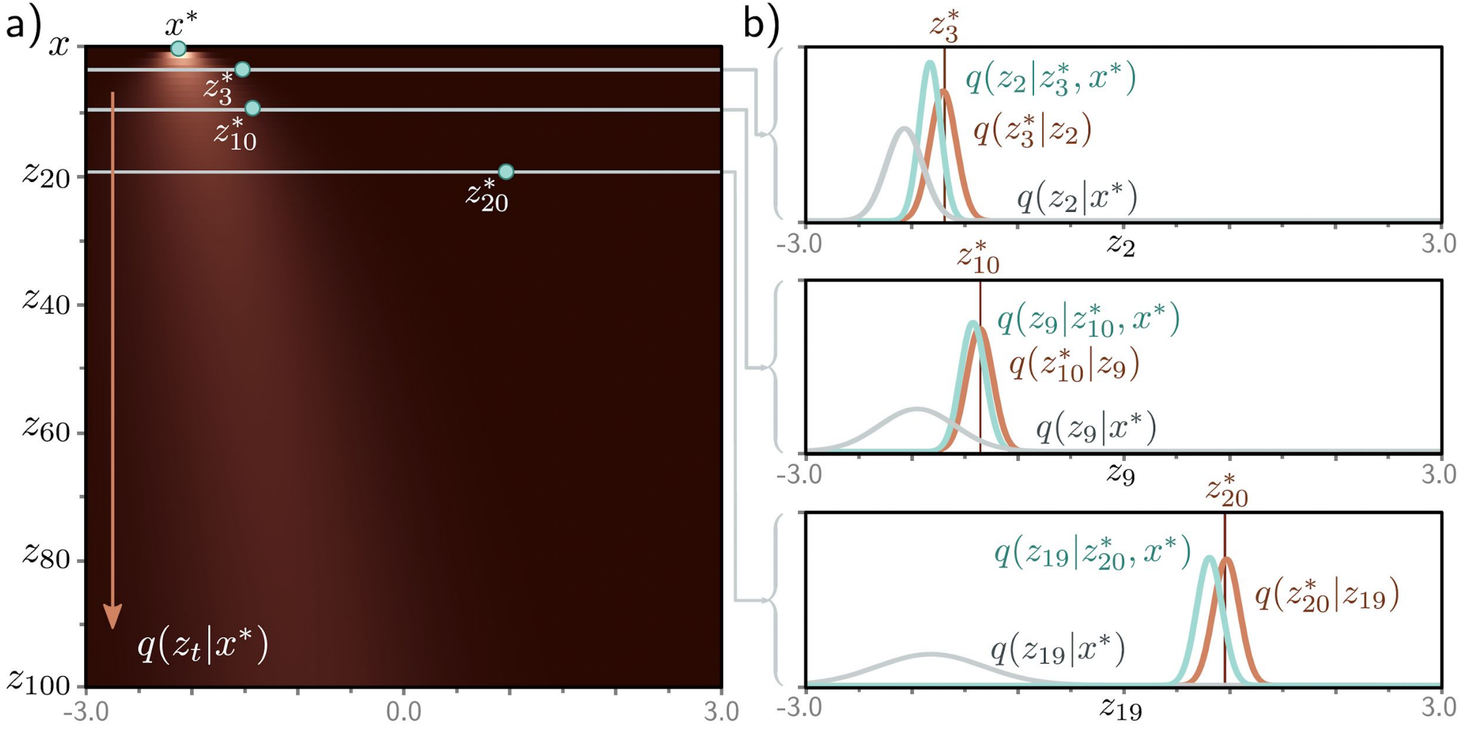


Figure 18.6 Conditional distribution $q(z_{t-1}|z_t, x)$. a) Diffusion kernel for $x^* = -2.1$ with three points z_t^* highlighted. b) The probability $q(z_{t-1}|z_t^*, x^*)$ is computed via Bayes' rule and is proportional to $q(z_t^*|z_{t-1})q(z_{t-1}|x^*)$. This *is* normally distributed and can be computed in closed form. The first likelihood term $q(z_t^*|z_{t-1})$ is normal in z_{t-1} (equation 18.2) with a mean that is slightly further from zero than z_t^* (brown curves). The second term is the diffusion kernel $q(z_{t-1}|x^*)$ (gray curves).

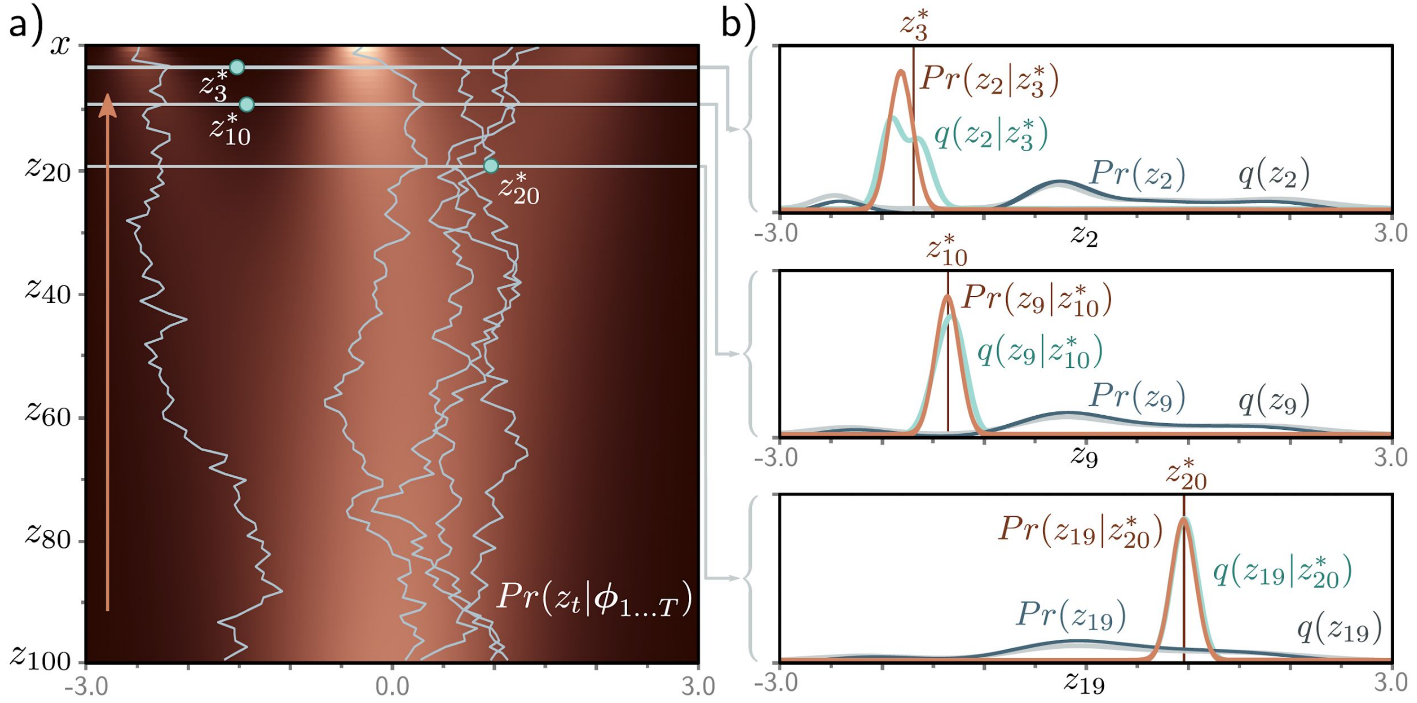


Figure 18.7 Fitted Model. a) Individual samples can be generated by sampling from the standard normal distribution $Pr(z_T)$ (bottom row)and then sampling z_{T-1} from $Pr(z_{T-1}|z_T) = \text{Norm}_{\mathbf{z}_{T-1}}[\text{f}_T[z_T, \boldsymbol{\phi}_T], \sigma_T^2\mathbf{I}]$ and so on until we reach x (five paths shown). The estimated marginal densities (heatmap) are the aggregation of these samples and are similar to the true marginal densities (figure 18.4). b) The estimated distribution $Pr(z_{t-1}|z_t)$ (brown curve) is a reasonable approximation to the true posterior of the diffusion model $q(z_{t-1}|z_t)$ (cyan curve) from figure 18.5. The marginal distributions $Pr(z_t)$ and $q(z_t)$ of the estimated and true models (dark blue and gray curves, respectively) are also similar.

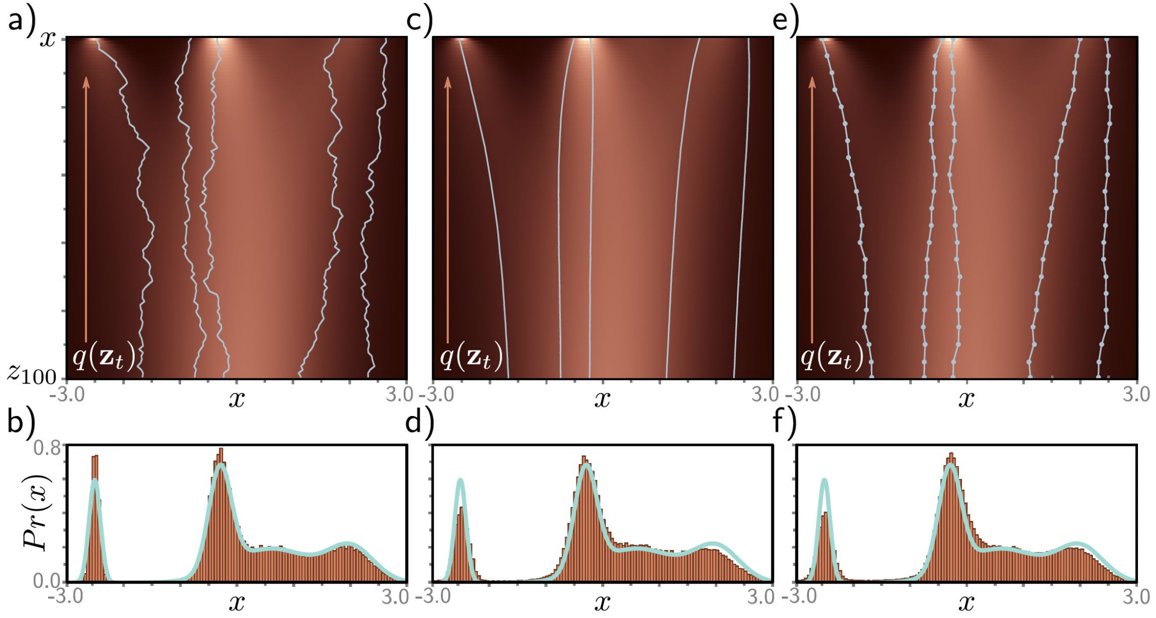


Figure 18.10 Different diffusion processes that are compatible with the same model. a) Five sampled trajectories or reparameterized model superimposed on ground truth marginal distributions. Top row represents $Pr(\mathbf{x})$ and subsequent rows represent $q(\mathbf{x}_t)$. b) Histogram of samples generated from reparameterized model plotted alongside ground truth density curve $Pr(\mathbf{x})$. The same trained model is compatible with a family of diffusion models (and corresponding updates in the opposite direction), including the denoising diffusion implicit (DDIM) model, which is deterministic and does not add noise at each step. c) Five trajectories from DDIM model. d) Histogram of samples from DDIM model. The same model is also compatible with accelerated diffusion models that skip inference steps for increased sampling speed. e) Five trajectories from accelerated model. f) Histogram of samples from accelerated model.

Conditional Generation for Diffusion Models

Can apply various conditioning throughout the diffusion process.

- Many ways to do this.
- Easy one:
  - concat conditioning vector as extra inputs to reverse steps.
- Will show some examples now, go into architecture later.

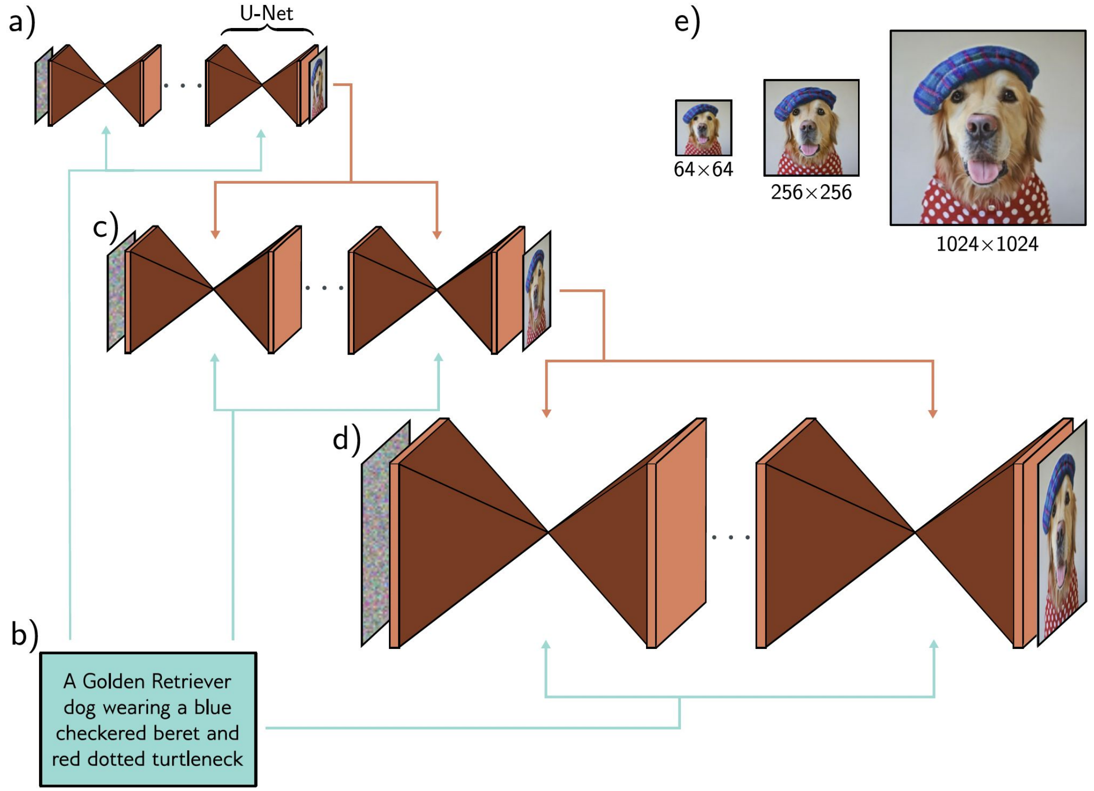

a)
U-Net
e)
64×64
256×256
1024×1024
c)
d)
b)
A Golden Retriever dog wearing a blue checkered beret and red dotted turtleneck

Figure 18.11 Cascaded conditional generation based on a text prompt. a) A diffusion model consisting of a series of U-Nets is used to generate a 64×64 image. b) This generation is conditioned on a sentence embedding computed by a language model. c) A higher resolution 256×256 image is generated and conditioned on the smaller image *and* the text encoding. d) This is repeated to create a 1024×1024 image. e) Final image sequence. Adapted from Saharia et al. (2022b).

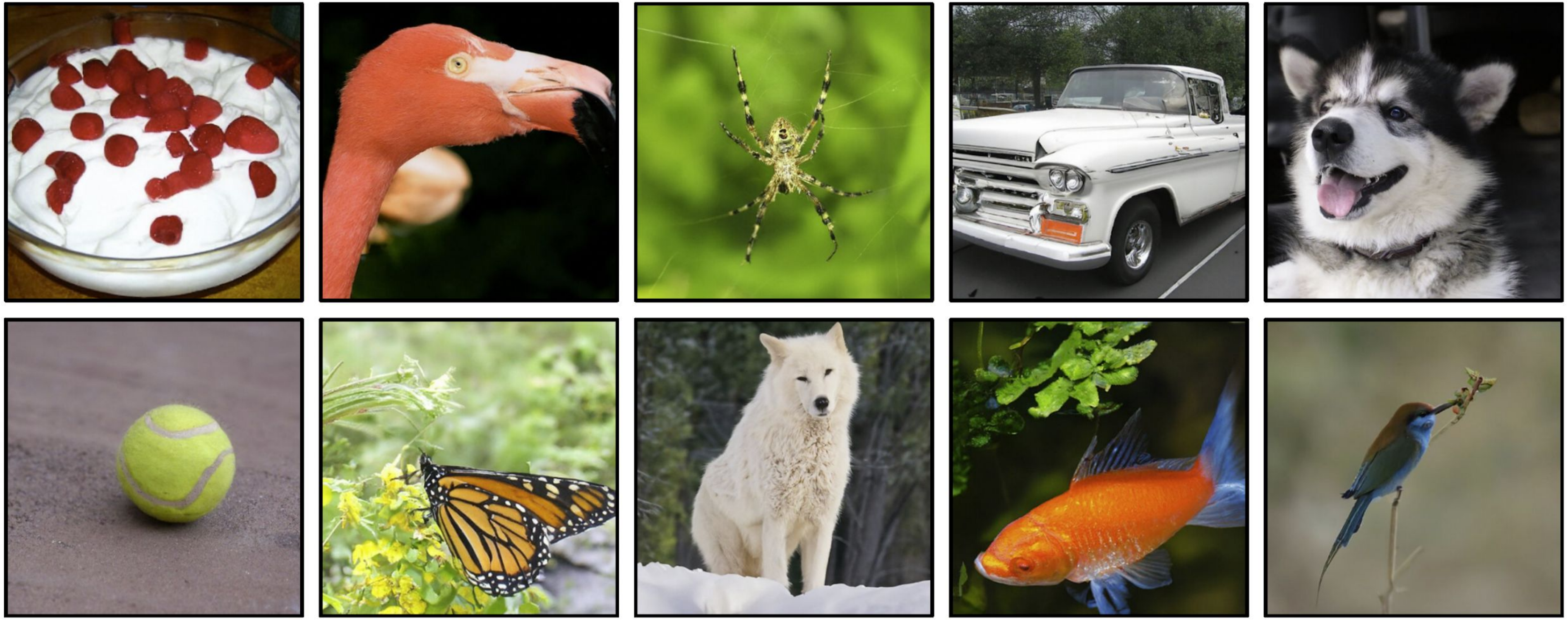


Figure 18.12 Conditional generation using classifier guidance. Image samples conditioned on different ImageNet classes. The same model produces high quality samples of highly varied image classes. Adapted from Dhariwal & Nichol (2021).

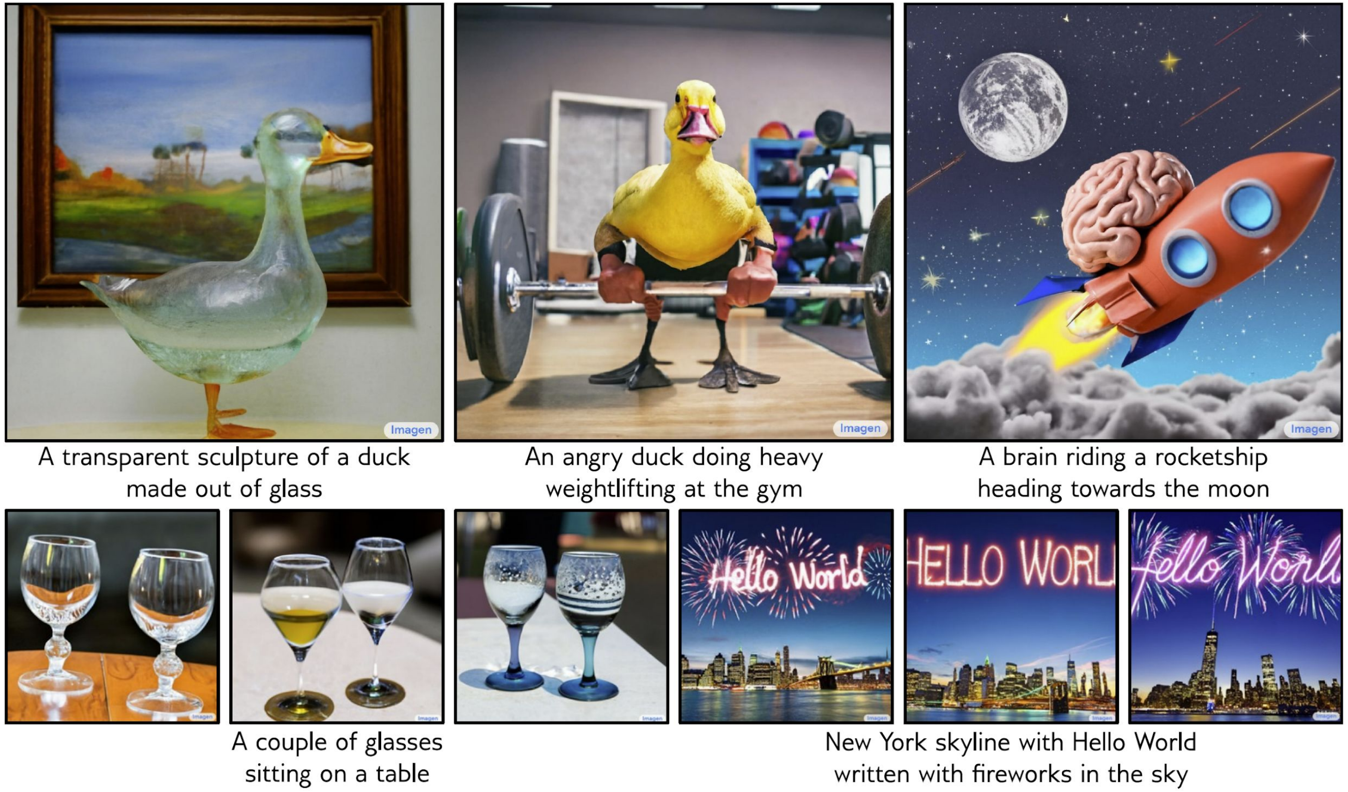

A transparent sculpture of a duck made out of glass
An angry duck doing heavy weightlifting at the gym
A brain riding a rocketship heading towards the moon
A couple of glasses sitting on a table
New York skyline with Hello World written with fireworks in the sky
Imagen

Figure 18.13 Conditional generation using text prompts. Synthesized images from a cascaded generation framework, conditioned on a text prompt encoded by a large language model. The stochastic model can produce many different images compatible with the prompt. The model can count objects and incorporate text into images. Adapted from Saharia et al. (2022b).

Diffusion Models

Eventually got quality comparable to GANs…

- Better than VAEs and normalizing flows
- But very very slow

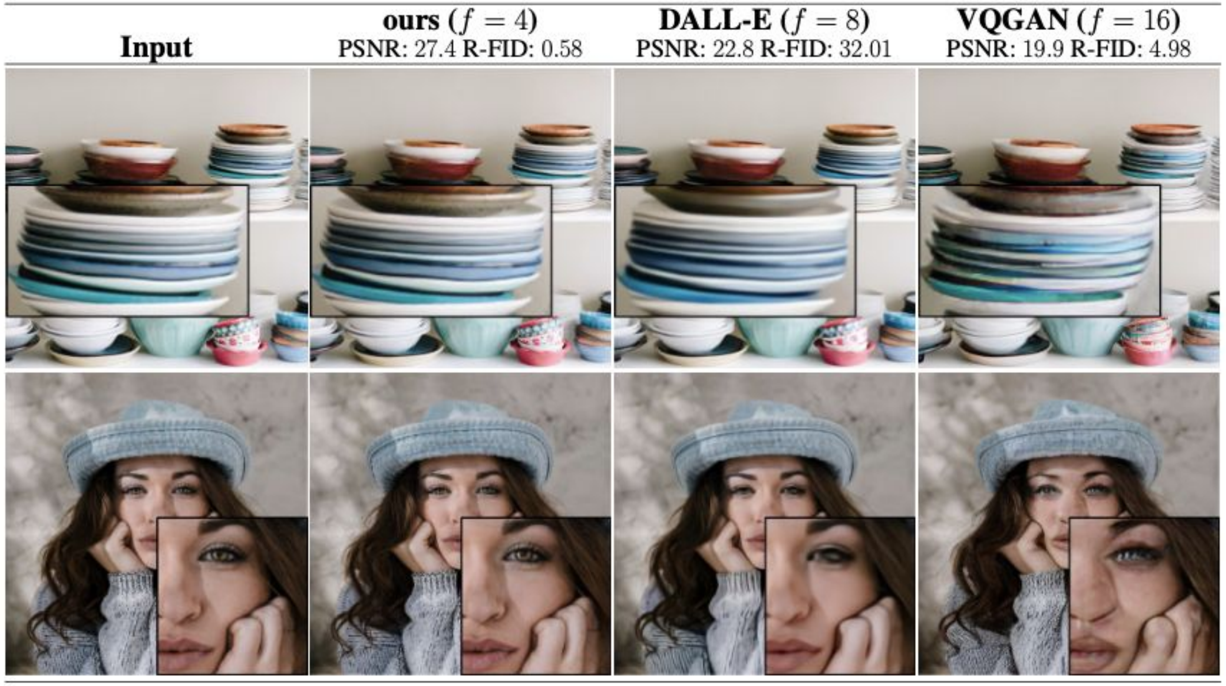
- Every step of the reverse process is working with full size images.
    - Full size input
    - Full size output
- Multi-resolution architectures are often a sign cost is an issue.
    - But also useful for global consistency, so do not disregard.

Latent Diffusion

Key idea:

- Pixel space is big.
- Run diffusion process in smaller latent space.

“High-Resolution Image Synthesis with Latent Diffusion Models”
By Rombach, Blattman, Lorenz, Esser and Ommer (2021)



Latent Diffusion Components

Latent diffusion models have two main components.

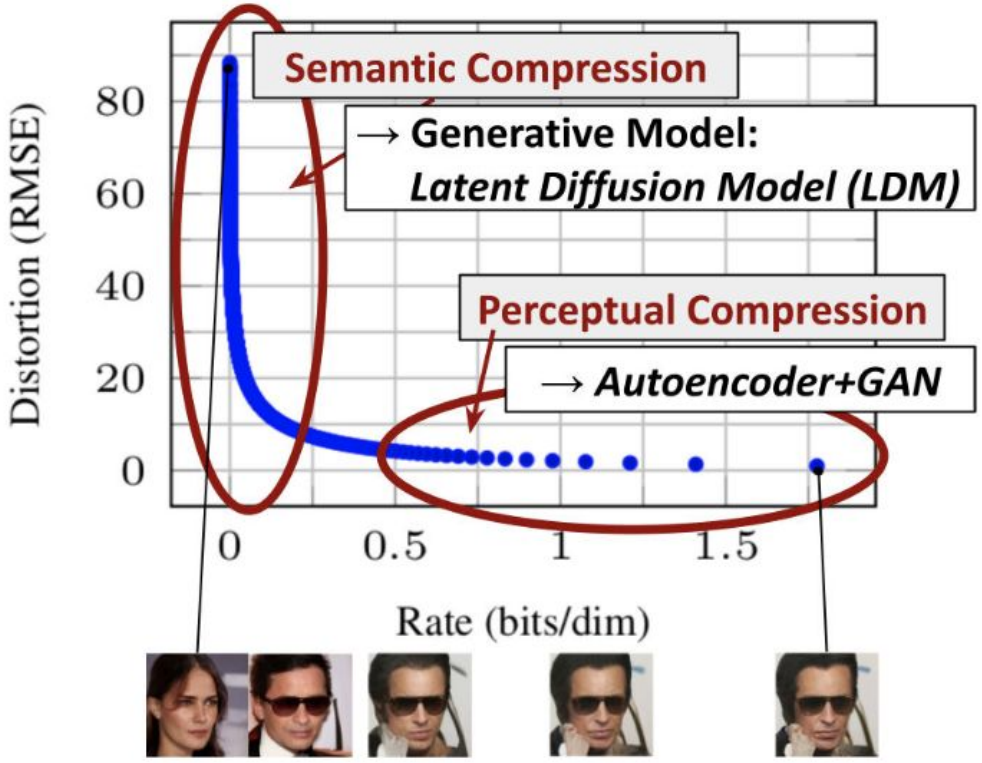
- Function mapping latent codes to high quality images.
  - This function does not need to have all the qualities we want from generative models.
  - In particular, much of the latent space may not make sense.
  - This can be off the shelf.
- Function mapping noise to latent codes of high quality images.
  - This is the latent diffusion part.
  - Same reverse process as before, but working in latent space.
  - This is smaller and easier to customize to different applications.

Different Models for Different Tradeoffs

Focus on lower dimension latent model that gets the semantic details right…

- Diffusion in latent space
- Use another high quality model for image generation.

“High-Resolution Image Synthesis with Latent Diffusion Models”
By Rombach, Blattman, Lorenz, Esser and Ommer (2021)

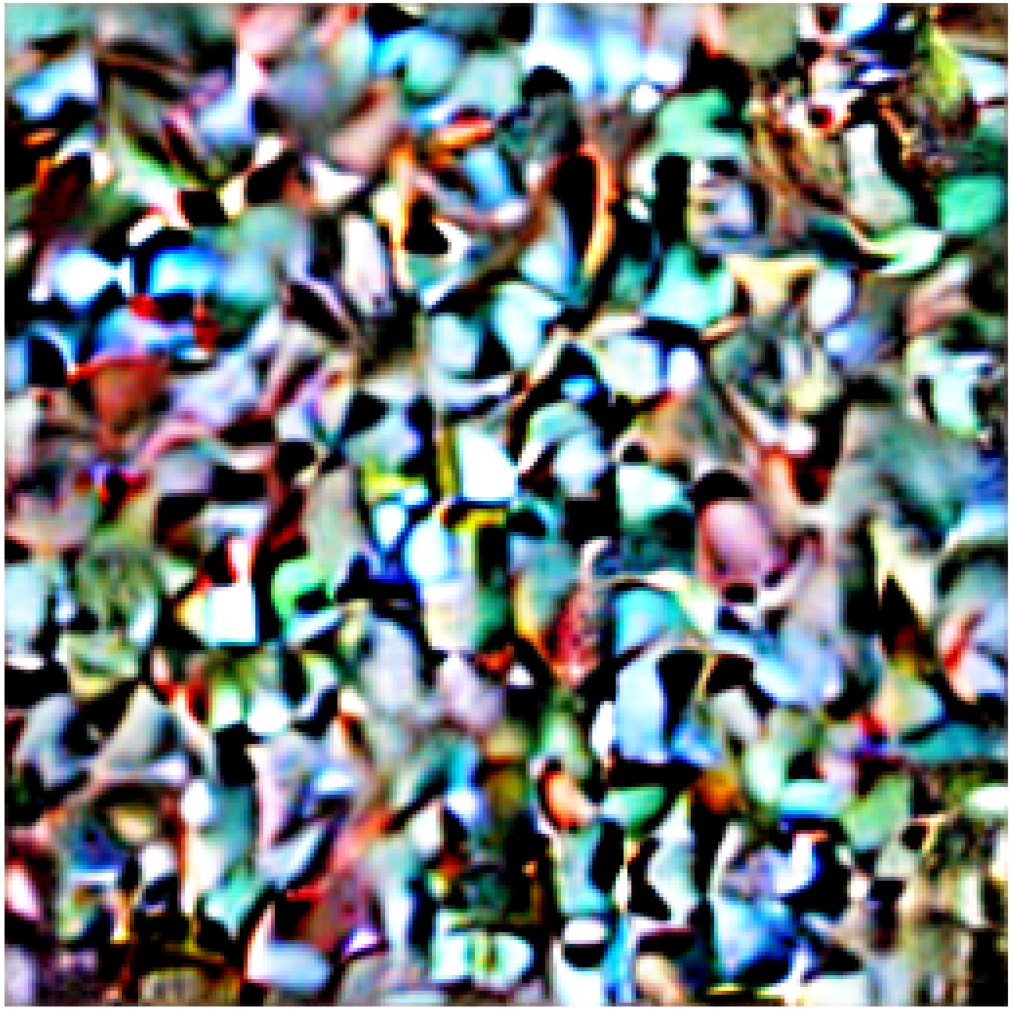


A Random Latent from Stable Diffusion

```
latent = torch.randn(1, 32, 32, 4)
latent_image = decode_latent(latent)
```

Image shape is 1x256x256x3.

This is that high quality image model?



https://stability.ai/

Picking a good latent is still a hard problem.

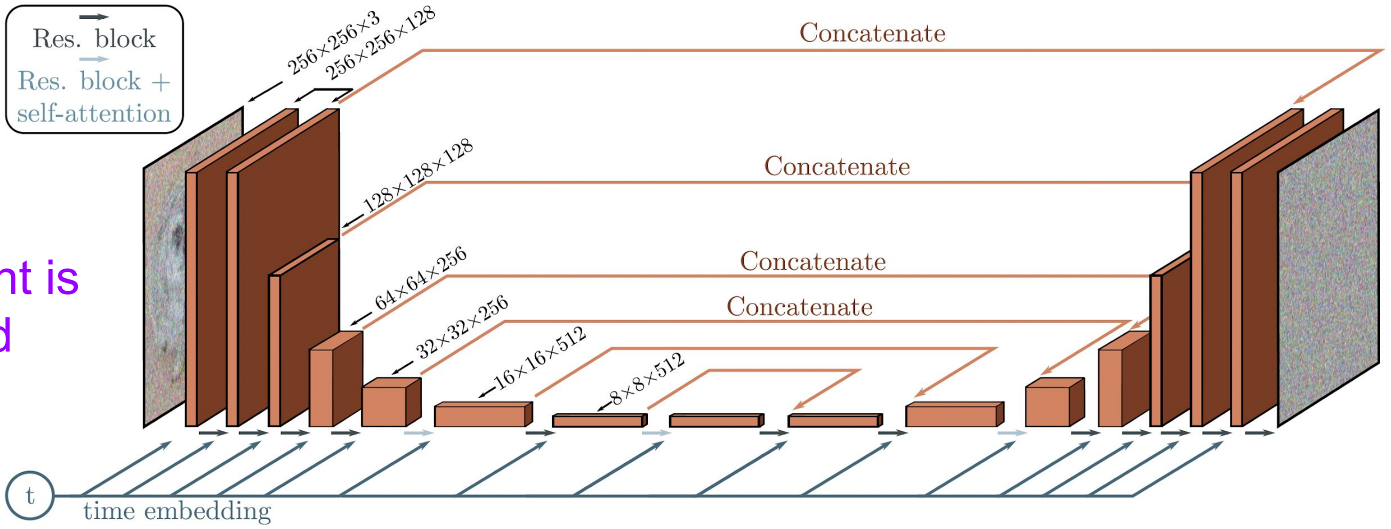
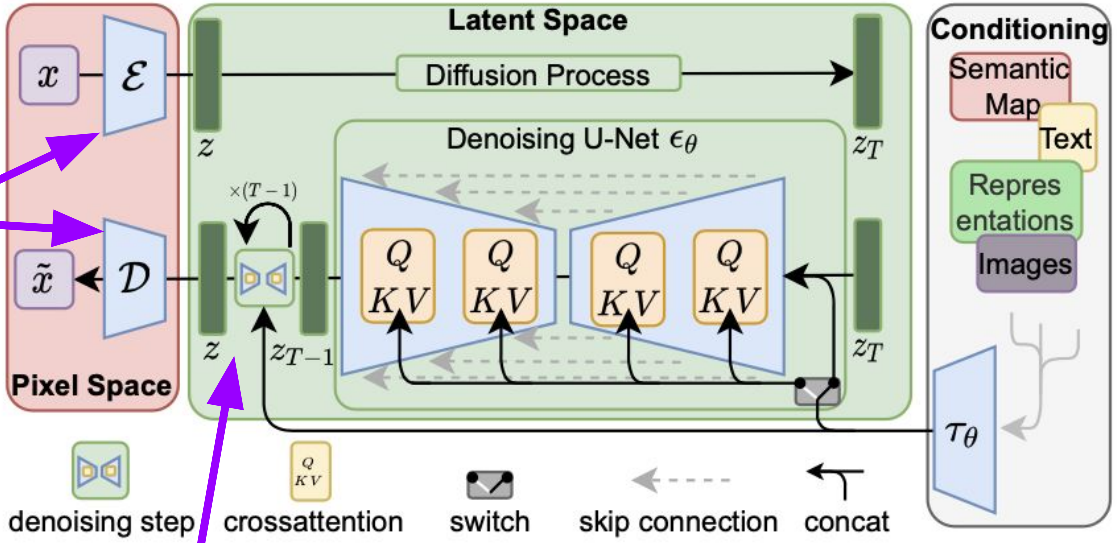


Figure 18.9 U-Net as used in diffusion models for images. The network aims to predict the noise that was added to the image. It consists of an encoder which reduces the scale and increases the number of channels and a decoder which increases the scale and reduces the number of channels. The encoder representations are concatenated to their partner in the decoder. Connections between adjacent representations consist of residual blocks, and periodic global self-attention in which every spatial position interacts with every other spatial position. A single network is used for all time steps, by passing a sinusoidal time embedding (figure 12.5) through a shallow neural network and adding the result to the channels at every spatial position at every stage of the U-Net.

Conditioning in Latent Space

Conditioning takes over all of latent sampling.

Pick your favorite autoencoder with a small latent space.



T total repetitions of denoising with conditioning.

Conditioning affects each denoising step.

Applications of Latent Diffusion Models

Latent diffusion models have two main components.

- Function mapping latent codes to high quality images.
  - This function does not need to have all the qualities we want from generative models.
  - In particular, much of the latent space may not make sense.
- Function mapping noise to latent codes of high quality images.
  - This is the latent diffusion part.
  - Just this part gets retrained for different applications.

Super Resolution

- Downsample training images 4x
- Upsample with bicubic interpolation.
- Train latent diffusion model to recover the finer-grained details.

“High-Resolution Image Synthesis with Latent Diffusion Models”
By Rombach, Blattman, Lorenz, Esser and Ommer (2021)

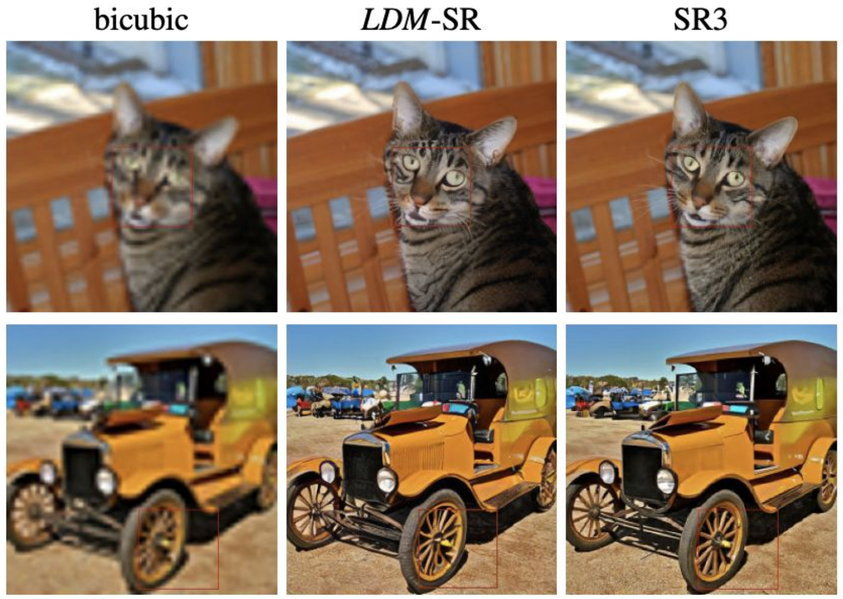


Figure 9. ImageNet 64→256 super-resolution on ImageNet-Val. *LDM-SR* has advantages at rendering realistic textures but SR3 can synthesize more coherent fine structures. See appendix for additional samples and cropouts. SR3 results from [70].

In Painting

Mask some of the image and reconstruct rest of the image to be consistent.

- Don't change the unmasked part!
- Keeping unmasked part mostly easy because latent code has spatial structure.

"High-Resolution Image Synthesis with Latent Diffusion Models"
By Rombach, Blattman, Lorenz, Esser and Ommer (2021)

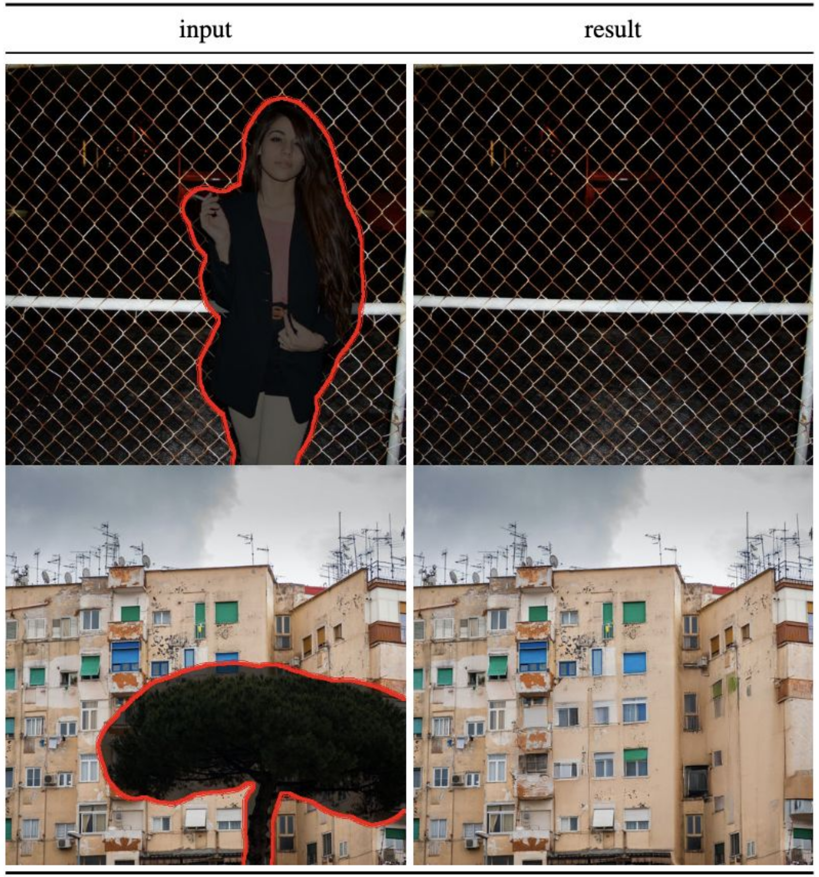
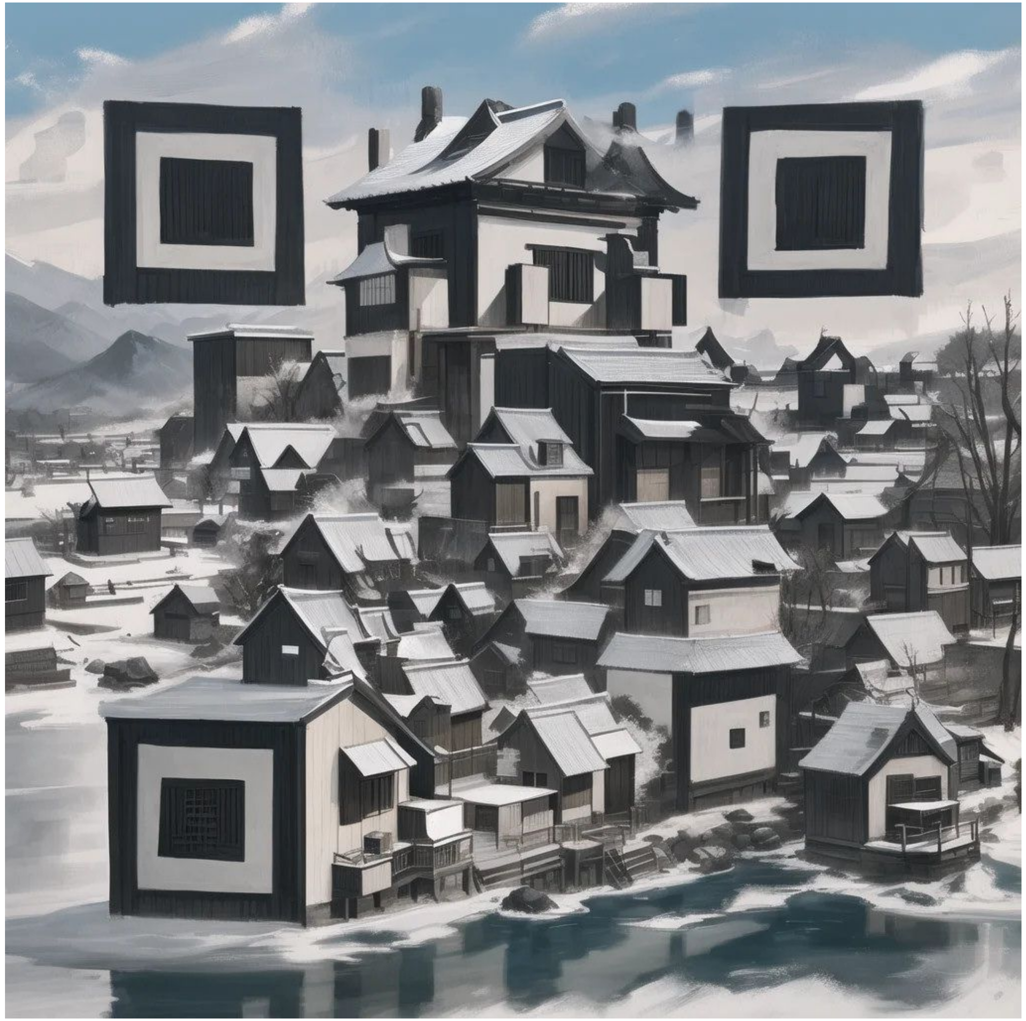


Figure 10. Qualitative results on object removal with our *big, w/ ft* inpainting model. For more results, see Fig. 21.

Custom QR Codes

https://mp.weixin.qq.com/s/i4WR5ULH1ZZYI8Watf3EPw



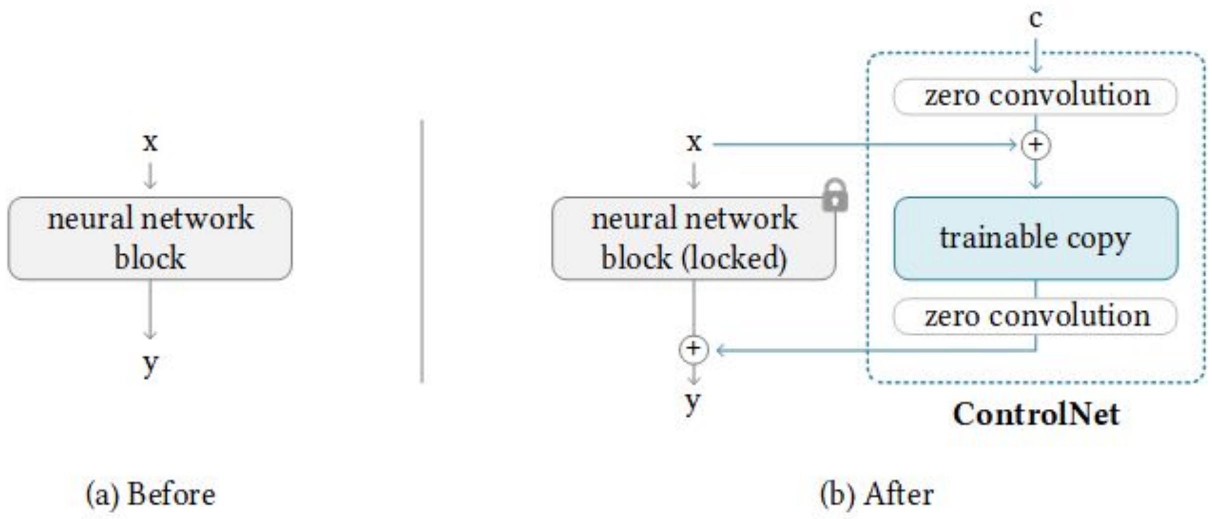
ControlNet

Previous QR Code example based on ControlNet.

- Generic technique to modify diffusion process to shape resulting output.
- Constant c initialized to zero to block action until helpful.

“Adding Conditional Control to Text-to-Image Diffusion Models” by Zhang, Rao and Agrawala (2023)

https://github.com/lllyasviel/ControlNet



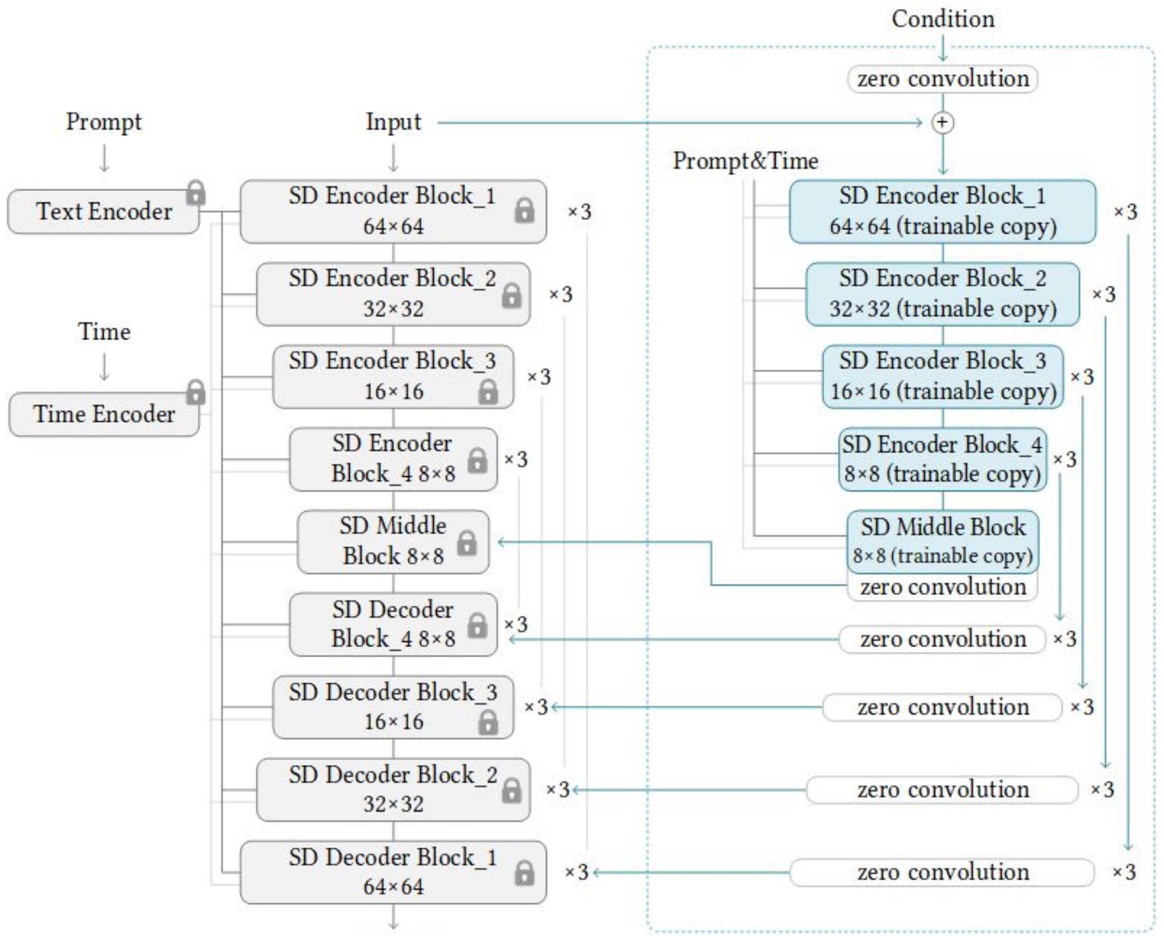
ControlNet

Previous QR Code example based on ControlNet.

- Generic technique to modify diffusion process to shape resulting output.
- Constant c initialized to zero to block action until helpful.

"Adding Conditional Control to Text-to-Image Diffusion Models" by Zhang, Rao and Agrawala (2023)

https://github.com/lllyasviel/ControlNet



(a) Stable Diffusion (b) ControlNet

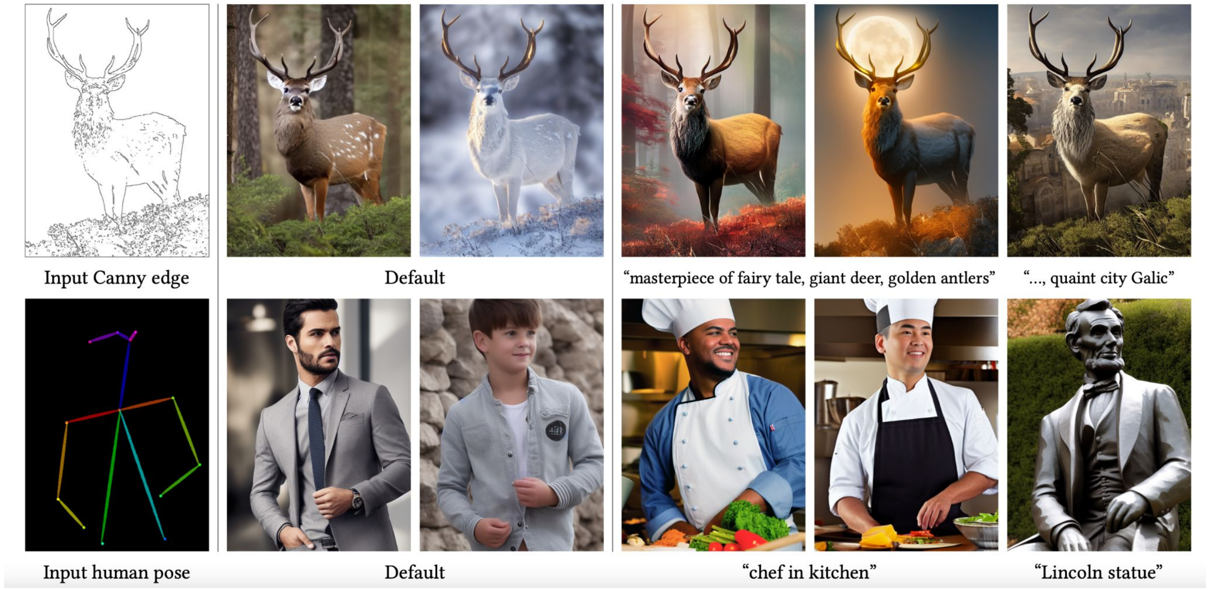
ControlNet

What are those control blocks for?

- Drive output image to have a particular property.
- Usually specified as feature extractor module and target output.

“Adding Conditional Control to Text-to-Image Diffusion Models” by Zhang, Rao and Agrawala (2023)

https://github.com/lllyasviel/ControlNet



Rest of the Semester

- Neural Fields (11/25)
- Thanksgiving break (11/27)
- Reinforcement learning (12/2)
- Project presentations (12/4)
- Project presentations (12/9)

Feedback?