AI Applications Lecture 15

Image Generation AI 5: Convolutional Neural Networks for Image Generation

SUZUKI, Atsushi
Jing WANG

Outline

Introduction

Roadmap Recap

We will review the content learned so far. Three lectures ago, we learned about the **Variational Autoencoder (VAE)** as a natural image decoder [3]. Two lectures ago, we learned about the **reverse diffusion process** that generates low-resolution latent images, namely the **denoising scheduler** [1]. In the previous lecture, we mathematically understood the sense in which the reverse diffusion process performs **distribution learning**, using the continuity equation, score, and KL divergence (see [6, 2]).

Roadmap Recap

We will review the content learned so far. Three lectures ago, we learned about the **Variational Autoencoder (VAE)** as a natural image decoder [3]. Two lectures ago, we learned about the **reverse diffusion process** that generates low-resolution latent images, namely the **denoising scheduler** [1]. In the previous lecture, we mathematically understood the sense in which the reverse diffusion process performs **distribution learning**, using the continuity equation, score, and KL divergence (see [6, 2]).

In this lecture, we will focus our discussion on **practical image generation AI** and look in detail at the **neural network architectures** used in denoising schedulers and VAEs. In particular, we will focus on the differences, as the **architecture in the original paper** and the **architecture used in actual implementations** often differ (e.g., Latent Diffusion/Stable Diffusion [4]).

Learning Outcomes

By the end of this lecture, students should be able to:

- **Mathematically describe** the **neural network architectures used in practical image generation AI**.

Learning Outcomes

By the end of this lecture, students should be able to:

- **Mathematically describe** the **neural network architectures used in practical image generation AI**.
- **Explain** how practical image generation AI achieves support for **variable input/output sizes** by using specific **layers.**

Learning Outcomes

By the end of this lecture, students should be able to:

- **Mathematically describe** the **neural network architectures used in practical image generation AI**.
- **Explain** how practical image generation AI achieves support for **variable input/output sizes** by using specific **layers.**
- **Explain** how the neural network architectures used in practical image generation AI have **changed from their original proposals**.

Preparation: Mathematical Notations

Notation: Definitions and Sets (1/2)

- **Definition:**
  - $(\mathrm{LHS}) := (\mathrm{RHS})$: Indicates that the left-hand side is defined by the right-hand side. For example, $a := b$ indicates that a is defined as b.
- **Set:**
  - Sets are often denoted by uppercase calligraphic letters. E.g., $\mathcal{A}$.
  - $x \in \mathcal{A}$: Indicates that element x belongs to set $\mathcal{A}$.
  - $\{\}$: The empty set.
  - $\{a, b, c\}$: The set consisting of elements a, b, c (set-builder notation by extension).
  - $\{x \in \mathcal{A} \mid P(x)\}$: The set of elements in set $\mathcal{A}$ for which the proposition $P(x)$ is true (set-builder notation by intension).
  - $|\mathcal{A}|$: The number of elements in set $\mathcal{A}$ (in this lecture, used only for finite sets).

Notation: Numbers and Ranges (2/2)

- $\mathbb{R}$: The set of all real numbers. Similarly for $\mathbb{R}_{>0}$, $\mathbb{R}_{\geq 0}$, etc.
- $\mathbb{Z}$: The set of all integers. Similarly for $\mathbb{Z}_{>0}$, $\mathbb{Z}_{\geq 0}$, etc.
- $[1, k]_{\mathbb{Z}}$: For $k \in \mathbb{Z}_{>0} \cup \{+\infty\}$, if $k < +\infty$, then $\{1, \ldots, k\}$; if $k = +\infty$, then $\mathbb{Z}_{>0}$.

Notation: Functions and Vectors

- **Function:**
  - $f : \mathcal{X} \to \mathcal{Y}$ denotes a mapping.
  - $y = f(x)$ denotes the output $y \in \mathcal{Y}$ for the input $x \in \mathcal{X}$.
- **Vector:**
  - Vectors are denoted by bold italic lowercase letters. E.g., $\boldsymbol{v}$. $\boldsymbol{v} \in \mathbb{R}^n$.
  - The i-th component is written as v_i:

$$\boldsymbol{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}.$$

  - Standard inner product:

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle := \sum_{i=1}^{d_{\text{emb}}} u_i v_i.$$

Notation: Sequences, Matrices, and Tensors i

- **Sequence:**
  - We call $\boldsymbol{a} : [1, n]_{\mathbb{Z}} \to \mathcal{A}$ a sequence of length n. If $n < +\infty$, $\boldsymbol{a} = (a_1, \ldots, a_n)$; if $n = +\infty$, $\boldsymbol{a} = (a_1, a_2, \ldots)$.
  - The length is written as $|\boldsymbol{a}|$.
- **Matrix:**
  - $\boldsymbol{A} \in \mathbb{R}^{m,n}$ with elements $a_{i,j}$,

$$\boldsymbol{A} = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{bmatrix}.$$

Notation: Sequences, Matrices, and Tensors ii

- Transpose $\boldsymbol{A}^\top \in \mathbb{R}^{n,m}$,

$$\boldsymbol{A}^\top = \begin{bmatrix} a_{1,1} & \cdots & a_{m,1} \\ \vdots & \ddots & \vdots \\ a_{1,n} & \cdots & a_{m,n} \end{bmatrix}.$$

- Vector row:

$$\boldsymbol{v}^\top = \begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix}.$$

- **Tensor:**
  - A tensor as a multi-dimensional array is denoted by an underlined bold italic uppercase letter $\underline{\boldsymbol{A}}$.
  - $\odot$: elementwise multiplication.

General Theory: Reconfirming Architectural Freedom

Noise Estimator: Architecture-Independent Training and Inference

The training of the **noise estimator** used in the denoising scheduler was given by the optimization problem of minimizing the squared error of the noise estimation. The objective function is identical to the previous lecture:

$$\min_{\boldsymbol{\theta}} \sum_{i=1}^{m} \left\| \boldsymbol{\epsilon}^{(i)} - \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}\left(\boldsymbol{\zeta}^{(i)}, \boldsymbol{c}^{(i)}, t^{(i)}\right) \right\|_2^2 .$$

Noise Estimator: Architecture-Independent Training and Inference

The training of the **noise estimator** used in the denoising scheduler was given by the optimization problem of minimizing the squared error of the noise estimation. The objective function is identical to the previous lecture:

$$\min_{\boldsymbol{\theta}} \sum_{i=1}^{m} \left\| \boldsymbol{\epsilon}^{(i)} - \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}\left(\boldsymbol{\zeta}^{(i)}, \boldsymbol{c}^{(i)}, t^{(i)}\right) \right\|_2^2 .$$

(**??**)

Noise Estimator: Architecture-Independent Training and Inference

The training of the **noise estimator** used in the denoising scheduler was given by the optimization problem of minimizing the squared error of the noise estimation. The objective function is identical to the previous lecture:

$$\min_{\boldsymbol{\theta}} \sum_{i=1}^{m} \left\| \boldsymbol{\epsilon}^{(i)} - \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}\left(\boldsymbol{\zeta}^{(i)}, \boldsymbol{c}^{(i)}, t^{(i)}\right) \right\|_2^2 .$$

(**??**)

(**??**) is defined **independently of the neural network's architecture.** Inference also just involves inserting the trained $\hat{\epsilon}_{\boldsymbol{\theta}}$ into a sequential algorithm, which is also **architecture-independent** (e.g., DDPM/DDIM steps [1]).

VAE Training and Inference are Likewise Architecture-Independent

VAE training is regularized reconstruction error minimization [3]. The training scheme is also **architecture-independent.** Once the decoder is trained, inference is **only the application of the decoder**.

VAE Training and Inference are Likewise Architecture-Independent

VAE training is regularized reconstruction error minimization [3]. The training scheme is also **architecture-independent.** Once the decoder is trained, inference is **only the application of the decoder**.

Recall: Encoders:

$$\boldsymbol{z}_{\boldsymbol{\epsilon}} := \text{MeanEnc}_{\boldsymbol{\eta}_{\text{mean}}}(\underline{\boldsymbol{X}}) + \text{SDEnc}_{\boldsymbol{\eta}_{\text{SD}}}(\underline{\boldsymbol{X}}) \odot \boldsymbol{\epsilon} \tag{1}$$

Decoder:

$$\hat{\boldsymbol{X}}_{\boldsymbol{\epsilon}} := \text{Dec}_{\gamma}(\boldsymbol{z}_{\boldsymbol{\epsilon}}). \tag{2}$$

Using a reconstruction loss function $\ell : \mathcal{I} \times \mathcal{I} \to \mathbb{R}_{\geq 0}$, the objective function is

$$\mathcal{L}(\boldsymbol{\eta}, \gamma) := \mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})}\Big[\underbrace{\ell(\underline{\boldsymbol{X}}, \underline{\hat{\boldsymbol{X}}_{\boldsymbol{\epsilon}}})}_{\text{Reconstruction term}} \Big] + \beta \underbrace{\sum_{i=1}^{d} \left(\mu_i(\underline{\boldsymbol{X}})^2 + \sigma_i(\underline{\boldsymbol{X}})^2 - \log \sigma_i(\underline{\boldsymbol{X}})^2 - 1\right)}_{\text{Regularization (concentration to origin)}}. \tag{3}$$

Consequence of the General Theory

From the above, it is clear that, outside the context of image generation, both the noise estimator and the VAE decoder can adopt **any architecture.**

Necessity of Variable I/O Sizes and Convolutional Layers

Necessity of Variable Input/Output Sizes

In practical image generation, the **output resolution (dimensions)** depends on user requirements and cannot be fixed at training time. Therefore, an architecture with **variable input/output sizes**, which allows selecting the output size by choosing the input size (latent resolution), is necessary.

Layer Achieving Variable I/O Sizes: Convolutional Layer

To construct variable input/output sizes, each layer only needs to be a **parametric function compatible with variable sizes**. A typical example is the **convolution layer (implemented as cross-correlation)**. The noise estimator in Stable Diffusion 1.5 is a **U-Net** [5] family (conditional, with attention), and the VAE is also constructed with convolutional systems [4].

Overview of Stable Diffusion 1.5
U-Net and VAE with Diagrams

Different Motivations Drive Architectural Differences

U-Net and VAE have very different expected functionalities.

- U-Net's purpose is to generate a **globally coherent** low-resolution latent image from noise (which has no information), using information from a text encoder.
- VAE's purpose is to convert a low-resolution latent image, which already has some global coherence, into a high-resolution natural image by defining the details.

This is evident even when observing the intermediate states of image generation.

Different Motivations Drive Architectural Differences

Corresponding to these differences in motivation, the architectures actually used for U-Net and VAE encoders also differ.

- U-Net, to achieve global coherence, uses downsampling, giving it a structure that efficiently allows pixels at one edge to influence pixels at the opposite edge with a relatively small number of layers.
- The VAE encoder is composed of pure convolutional layers and upsampling layers, adopting a structure that restricts the use of parameters to local value transformations.

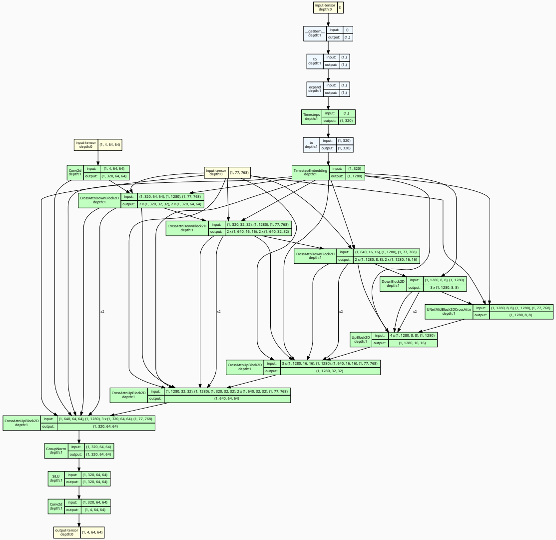
Let's confirm these differences by looking at the actual architectures.

Implementation Inspection via Computation Graphs

The actual architecture may differ in details from the paper's description. Using tools like **torchview**, one can visualize the **computation graph** from the implemented model, making it easier to grasp implementation differences[1].

[1] `torchview`: `https://github.com/mert-kurttutan/torchview`

Block Diagram of Conditional U-Net (Stable Diffusion 1.5)

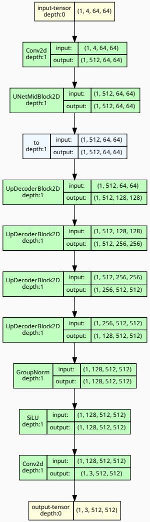


Remark: Difference from Original U-Net

Remark

Difference from the original U-Net: Ronneberger et al.'s U-Net [5] is an **image-to-image** mapping for **medical image segmentation**, where both input and output are images. In image generation AI, it needs to accept **time (noise level)** and **text conditions** as input, and perform **noise estimation (or v-prediction).** Therefore, the significant differences are the addition of a **time encoder** and **cross-attention layers (for text)** [4].

Block Diagram of VAE Decoder (Stable Diffusion 1.5)



Confirmation of Variable Input/Output Sizes

The fact that U-Net and VAE have variable input/output sizes follows from each component layer (convolution, normalization, attention, up/down-sample) being defined **convolutionally (translationally equivariant under isomorphism)** with respect to the **spatial resolution** $H \times W$.

Remark

The **variable input/output sizes** mentioned here refer to the **variability in the image width W and height H**; the **number of channels C is fixed** (although the channel width may change in steps inside the U-Net, the number of channels at the input/output interface is specified).

Formal Definitions of Layers

Cross-Correlation (2D "Convolution") — Definition

Definition (Conv2d (Convolution in practice is cross-correlation))

For input $\underline{\boldsymbol{X}} \in \mathbb{R}^{C_{\text{in}} \times H \times W}$ and output $\underline{\boldsymbol{Y}} \in \mathbb{R}^{C_{\text{out}} \times H' \times W'}$, we fix **hyperparameters** kernel size (k_h, k_w), stride (s_h, s_w), and padding (p_h, p_w). The set of **learnable parameters** (weights and biases) is

$$\Theta_{\text{Conv2d}} = \left\{ \boldsymbol{W}^{(o)} \in \mathbb{R}^{C_{\text{in}} \times k_h \times k_w},\ b_o \in \mathbb{R} \right\}_{o=1}^{C_{\text{out}}}$$

At this time,

$$\left(\text{Conv2d}_{\Theta_{\text{Conv2d}}}^{(k_h,k_w;\ s_h,s_w;\ p_h,p_w)}(\underline{\boldsymbol{X}})\right)_{o,i,j} = b_o + \sum_{c=1}^{C_{\text{in}}} \sum_{u=1}^{k_h} \sum_{v=1}^{k_w} W_{c,u,v}^{(o)}\, X_{c,\ i\cdot s_h + u - p_h,\ j\cdot s_w + v - p_w}.$$

The output spatial size is $H' = \left\lfloor \dfrac{H - k_h + 2p_h}{s_h} \right\rfloor + 1$, $W' = \left\lfloor \dfrac{W - k_w + 2p_w}{s_w} \right\rfloor + 1$.[2]

[2] torch.nn.Conv2d / torch.nn.functional.conv2d

Remark: Convolution vs Cross-Correlation

Remark

"Convolution" in implementations is cross-correlation (**does not flip the kernel**) and matches the displayed elementwise formula.

Linear Layer

Definition (Linear)

For input $\boldsymbol{x} \in \mathbb{R}^{d_{\text{in}}}$, output $\boldsymbol{y} \in \mathbb{R}^{d_{\text{out}}}$, and learnable parameters $\Theta_{\text{Linear}} = \{\boldsymbol{W} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}},\ \boldsymbol{b} \in \mathbb{R}^{d_{\text{out}}}\}$,

$$\text{Linear}_{\Theta_{\text{Linear}}}(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}.$$

Activation: SiLU (Swish) and Softmax

Definition (SiLU (Swish))

For a component u of an arbitrary-dimensional tensor,

$$\mathsf{SiLU}(u) = u\,\sigma(u), \quad \sigma(u) = \frac{1}{1+e^{-u}}.$$

Activation: SiLU (Swish) and Softmax

Definition (SiLU (Swish))

For a component u of an arbitrary-dimensional tensor,

$$\mathsf{SiLU}(u) = u\,\sigma(u), \quad \sigma(u) = \frac{1}{1+e^{-u}}.$$

Definition (Softmax)

For $\boldsymbol{x} \in \mathbb{R}^n$, fixing the temperature $\tau > 0$,

$$\left(\mathsf{Softmax}^{(\tau)}(\boldsymbol{x})\right)_i = \frac{\exp(x_i/\tau)}{\sum_{j=1}^{n} \exp(x_j/\tau)}.$$

Normalization: GroupNorm

Definition (GroupNorm**)**

For input $\underline{\boldsymbol{X}} \in \mathbb{R}^{C\times H\times W}$, **hyperparameter** number of groups $G \mid C$, and **learnable parameters** $\Theta_{\text{GroupNorm}} = \{\gamma \in \mathbb{R}^C,\ \beta \in \mathbb{R}^C\}$, the mean and variance for each group g are

$$\mu_g = \frac{1}{|S_g|} \sum_{(c,i,j)\in S_g} X_{c,i,j}, \qquad \sigma_g^2 = \frac{1}{|S_g|} \sum_{(c,i,j)\in S_g} (X_{c,i,j} - \mu_g)^2,$$

The output is

$$\left(\text{GroupNorm}^{(G)}_{\Theta_{\text{GroupNorm}}}(\underline{\boldsymbol{X}})\right)_{c,i,j} = \gamma_c \frac{X_{c,i,j} - \mu_{g(c)}}{\sqrt{\sigma^2_{g(c)} + \varepsilon}} + \beta_c.$$

Downsample and Upsample: Operators (1/2)

Downsample by Strided Cross-Correlation

$$\mathsf{Downsample2D}^{(2)}_{\Theta_{\mathrm{Down}}}(\underline{\boldsymbol{X}}) := \mathsf{Conv2d}^{(k_h,k_w;\ 2,2;\ p_h,p_w)}_{\Theta_{\mathrm{Down}}}(\underline{\boldsymbol{X}}).$$

Downsample and Upsample: Operators (1/2)

Downsample by Strided Cross-Correlation

$$\mathsf{Downsample2D}^{(2)}_{\Theta_{\text{Down}}}(\underline{\boldsymbol{X}}) := \mathsf{Conv2d}^{(k_h,k_w;\ 2,2;\ p_h,p_w)}_{\Theta_{\text{Down}}}(\underline{\boldsymbol{X}}).$$

Nearest and Average Pooling (Element-wise)

$$Y_{c,i,j} = X_{c,\,2i,\,2j} \quad ; \quad Y_{c,i,j} = \frac{1}{4}\sum_{u=0}^{1}\sum_{v=0}^{1} X_{c,\,2i+u,\,2j+v}.$$

Downsample and Upsample: Operators (2/2)

Nearest-Neighbor Interpolation

$$Z_{c,\,2i+u,\,2j+v} = X_{c,i,j} \quad (u, v \in \{0, 1\})$$

Downsample and Upsample: Operators (2/2)

Nearest-Neighbor Interpolation

$$Z_{c,\,2i+u,\,2j+v} = X_{c,i,j} \quad (u, v \in \{0, 1\})$$

Upsample via Interpolate + Conv

$$\mathsf{Upsample2D}^{(2)}_{\Theta_{\mathrm{Up}}}(\underline{\boldsymbol{X}}) := \mathsf{Conv2d}^{(k_h,k_w;\ 1,1;\ p_h,p_w)}_{\Theta_{\mathrm{Up}}}\big(\mathsf{Interpolate}^{(\times 2,\mathrm{nearest})}(\underline{\boldsymbol{X}})\big).$$

Illustration: 1D Downsample

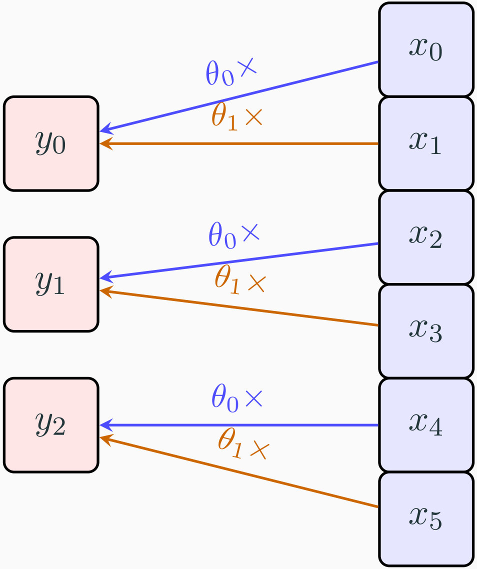
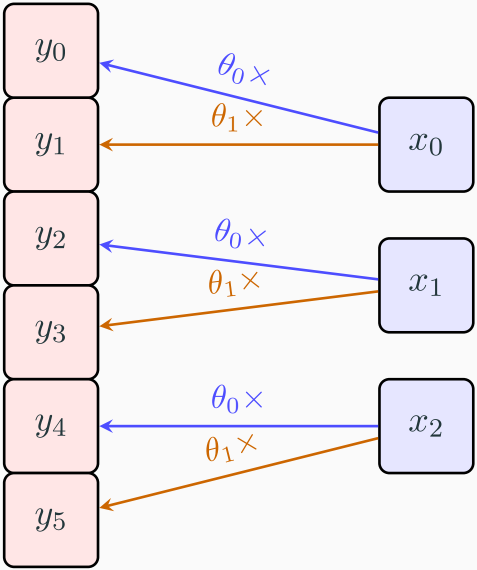


Illustration: 1D Upsample



Reshape and Concatenation

For $\underline{\boldsymbol{X}} \in \mathbb{R}^{C\times H\times W}$,

$$\mathsf{flatten}_{(H,W)}(\underline{\boldsymbol{X}}) \in \mathbb{R}^{(HW)\times C}, \quad \big(\mathsf{flatten}_{(H,W)}(\underline{\boldsymbol{X}})\big)_{(i-1)W+j,\ c} = X_{c,i,j},$$

$$\mathsf{unflatten}_{(H,W)}(\boldsymbol{Y}) \in \mathbb{R}^{C\times H\times W}, \quad \big(\mathsf{unflatten}_{(H,W)}(\boldsymbol{Y})\big)_{c,i,j} = Y_{(i-1)W+j,\ c},$$

$\mathsf{Concat}(\underline{\boldsymbol{A}}, \underline{\boldsymbol{B}}) = \underline{\boldsymbol{A}} \oplus \underline{\boldsymbol{B}}$ (concatenation along the channel dimension).

Timestep Embedding (Noise Level)

Definition ($\mathsf{TimestepEmbedding}$ **(Sinusoidal + MLP))**

For scalar $t \in \mathbb{R}$ and frequency sequence $\omega_r = \omega_0 \beta^{\,r-1}$ $(r = 1, \ldots, R)$,

$$\boldsymbol{e}(t) = \big[\cos(\omega_1 t), \sin(\omega_1 t), \ldots, \cos(\omega_R t), \sin(\omega_R t)\big]^\top \in \mathbb{R}^{2R}.$$

For learnable parameters
$\Theta_{\mathrm{TE}} = \{\boldsymbol{U}_1 \in \mathbb{R}^{d_h \times 2R},\ \boldsymbol{b}_1 \in \mathbb{R}^{d_h},\ \boldsymbol{U}_2 \in \mathbb{R}^{d_t \times d_h},\ \boldsymbol{b}_2 \in \mathbb{R}^{d_t}\}$,

$$\mathsf{TimestepEmbedding}_{\Theta_{\mathrm{TE}}}(t) = \boldsymbol{U}_2\, \mathsf{SiLU}(\boldsymbol{U}_1\, \boldsymbol{e}(t) + \boldsymbol{b}_1) + \boldsymbol{b}_2 \in \mathbb{R}^{d_t}.$$

Remark

Components with small ω represent the **coarse position (low frequency, long period)** of t, while components with large ω represent the **fine position (high frequency, short period).** This is analogous to the **positional numeral system**, where **upper digits** are useful for approximate estimation, and **lower digits**

Scaled Dot-Product Attention

Definition (ScaledDotProductAttention**)**

For query $\boldsymbol{Q} \in \mathbb{R}^{N \times d}$, key $\boldsymbol{K} \in \mathbb{R}^{M \times d}$, and value $\boldsymbol{V} \in \mathbb{R}^{M \times d_v}$,

$$\mathsf{ScaledDotProductAttention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \mathsf{Softmax}^{(\sqrt{d})^{-1}} \left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d}} \right) \boldsymbol{V}.$$

Multihead Attention (with Projections)

Definition (MultiheadAttention (SDPA with Projections))

For input sequence $\boldsymbol{X} \in \mathbb{R}^{N \times d_{\text{in}}}$ and context sequence $\boldsymbol{C} \in \mathbb{R}^{M \times d_{\text{ctx}}}$, using learnable parameters

$$\Theta_{\text{MHA}} = \{\boldsymbol{W}_Q \in \mathbb{R}^{d_{\text{in}} \times d},\ \boldsymbol{W}_K \in \mathbb{R}^{d_{\text{ctx}} \times d},\ \boldsymbol{W}_V \in \mathbb{R}^{d_{\text{ctx}} \times d_v},\ \boldsymbol{W}_O \in \mathbb{R}^{d_v \times d_{\text{out}}}\}$$

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q, \quad \boldsymbol{K} = \boldsymbol{C}\boldsymbol{W}_K, \quad \boldsymbol{V} = \boldsymbol{C}\boldsymbol{W}_V,$$

$$\mathsf{MultiheadAttention}_{\Theta_{\text{MHA}}}(\boldsymbol{X}, \boldsymbol{C}) = \mathsf{ScaledDotProductAttention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V})\,\boldsymbol{W}_O.$$

Programming Intuition: Attention as Soft Dictionary

A **dictionary (map)** in programming is a correspondence of $\{\text{key} : \text{value}\}$, a structure that retrieves the corresponding value when a key is given.

Exercise (Python Dictionary Analogy)

For $\texttt{D} = \{\texttt{"cat"} : 1,\ \texttt{"dog"} : 2\}$, $\texttt{D["dog"]} = 2$. The ScaledDotProductAttention in attention implements a **soft dictionary** that "retrieves a weighted sum of values closest to the key" in a continuous vector space.

Proposition: Attention as Soft Dictionary

Proposition (Attention as a Soft Dictionary)

Let $\boldsymbol{K} = [\boldsymbol{k}_1^\top; \dots; \boldsymbol{k}_M^\top] \in \mathbb{R}^{M \times d}$, $\boldsymbol{V} = [\boldsymbol{v}_1^\top; \dots; \boldsymbol{v}_M^\top] \in \mathbb{R}^{M \times d_v}$*, and consider a single query* $\boldsymbol{q} \in \mathbb{R}^d$ *with* $\boldsymbol{Q} = [\boldsymbol{q}^\top]$*. Then*

$$\mathsf{ScaledDotProductAttention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \Big[\sum_{m=1}^{M} \pi_m(\boldsymbol{q})\, \boldsymbol{v}_m\Big], \quad \pi_m(\boldsymbol{q}) = \frac{\exp\Big(\langle \boldsymbol{q}, \boldsymbol{k}_m \rangle / \sqrt{d}\Big)}{\sum_{j=1}^{M} \exp\Big(\langle \boldsymbol{q}, \boldsymbol{k}_j \rangle / \sqrt{}$$

Here, $\pi(\boldsymbol{q})$ *is the first row of* $\mathsf{Softmax}\big((\boldsymbol{Q}\boldsymbol{K}^\top)/\sqrt{d}\big)$ *and corresponds perfectly to the softmax in attention.*

Remark and Numerical Example of SDPA

Remark

If $\boldsymbol{\pi}(\boldsymbol{q})$ becomes a one-hot vector (1 for some $m^\star$, 0 otherwise), then $\sum_m \pi_m(\boldsymbol{q})\,\boldsymbol{v}_m = \boldsymbol{v}_{m^\star}$, which matches the exact retrieval from a dictionary.

Example (Numerical Calculation of ScaledDotProductAttention)

Let $d = 2$, $\boldsymbol{q} = (1,0)^\top$, $\boldsymbol{k}_1 = (1,0)^\top$, $\boldsymbol{k}_2 = (0,1)^\top$, $\boldsymbol{v}_1 = (2,0)^\top$, $\boldsymbol{v}_2 = (0,3)^\top$. At this time, the inner products are

$$\langle \boldsymbol{q}, \boldsymbol{k}_1 \rangle = 1, \quad \langle \boldsymbol{q}, \boldsymbol{k}_2 \rangle = 0$$

and the scaled exponentials and resulting weights and outputs follow numerically as detailed in the lecture note.

Exercise and Answer: SDPA (2D)

Exercise (Numerical Example with 2D Vectors)

Let $d = 2$, $\boldsymbol{q} = (2, 1)^\top$, $\boldsymbol{k}_1 = (1, 0)^\top$, $\boldsymbol{k}_2 = (0, 1)^\top$, $\boldsymbol{v}_1 = (1, 2)^\top$, $\boldsymbol{v}_2 = (4, -1)^\top$. Calculate the output vector $\boldsymbol{o}$ of scaled dot product attention both as an exact expression and numerically.

Exercise and Answer: SDPA (2D)

Exercise (Numerical Example with 2D Vectors)

Let $d = 2$, $\boldsymbol{q} = (2, 1)^\top$, $\boldsymbol{k}_1 = (1, 0)^\top$, $\boldsymbol{k}_2 = (0, 1)^\top$, $\boldsymbol{v}_1 = (1, 2)^\top$, $\boldsymbol{v}_2 = (4, -1)^\top$. Calculate the output vector $\boldsymbol{o}$ of scaled dot product attention both as an exact expression and numerically.

Answer

The inner products, exponentials, softmax weights, and final output follow exactly and numerically as shown in the lecture note’ s derivation.

Proposition: Hard Dictionary Limit

Proposition (Limit to a Hard Dictionary)

Suppose for some $m^\star$*,* $\boldsymbol{k}_{m^\star} \parallel \boldsymbol{q}$ *and* $\boldsymbol{k}_m \perp \boldsymbol{q}$ $(m \neq m^\star)$*. Then, for any* $\alpha > 0$*, let* $\boldsymbol{q}_\alpha = \alpha \boldsymbol{q}$*,*

$$\lim_{\alpha \to +\infty} \pi_m(\boldsymbol{q}_\alpha) = \begin{cases} 1, & m = m^\star, \\ 0, & m \neq m^\star. \end{cases}$$

Remark

$\boldsymbol{k}_{m^\star} \parallel \boldsymbol{q}$ means query and key share direction, so the corresponding value is retrieved, matching hard dictionary behavior.

Time-Conditioned Residual Block

Time-Conditioned Residual Block

Definition of ResnetBlock2D

Definition (ResnetBlock2D **(Affine modulation by time embedding; FiLM))**

For input $\underline{\boldsymbol{X}} \in \mathbb{R}^{C_{\text{in}} \times H \times W}$ and time embedding $\boldsymbol{h} \in \mathbb{R}^{d_t}$, using learnable parameters

$$\Theta_{\text{ResnetBlock2D}} = \left(\Theta^{(1)}_{\text{Conv2d}},\ \Theta^{(2)}_{\text{Conv2d}},\ \Theta^{(s)}_{\text{Conv2d}},\ \Theta^{(1)}_{\text{GN}},\ \Theta^{(2)}_{\text{GN}},\ \Theta^{(\gamma)}_{\text{Linear}},\ \Theta^{(\beta)}_{\text{Linear}}\right),$$

Definition

$$\underline{\boldsymbol{U}}_1 = \mathsf{GroupNorm}^{(G)}_{\Theta^{(1)}_{\mathrm{GN}}}(\underline{\boldsymbol{X}}), \quad \underline{\boldsymbol{V}}_1 = \mathsf{SiLU}(\underline{\boldsymbol{U}}_1), \quad \underline{\boldsymbol{W}}_1 = \mathsf{Conv2d}^{(k,k;\ 1,1;\ p,p)}_{\Theta^{(1)}_{\mathrm{Conv2d}}}(\underline{\boldsymbol{V}}_1), \tag{4}$$

$$\gamma(\boldsymbol{h}) = \mathsf{Linear}_{\Theta^{(\gamma)}_{\mathrm{Linear}}}(\boldsymbol{h}), \quad \boldsymbol{\beta}(\boldsymbol{h}) = \mathsf{Linear}_{\Theta^{(\beta)}_{\mathrm{Linear}}}(\boldsymbol{h}), \tag{5}$$

$$\underline{\boldsymbol{U}}_2 = \mathsf{GroupNorm}^{(G)}_{\Theta^{(2)}_{\mathrm{GN}}}(\underline{\boldsymbol{W}}_1), \quad \widehat{\underline{\boldsymbol{U}}}_2 = \gamma(\boldsymbol{h}) \odot \underline{\boldsymbol{U}}_2 + \boldsymbol{\beta}(\boldsymbol{h}), \tag{6}$$

$$\underline{\boldsymbol{V}}_2 = \mathsf{SiLU}(\widehat{\underline{\boldsymbol{U}}}_2), \quad \underline{\boldsymbol{W}}_2 = \mathsf{Conv2d}^{(k,k;\ 1,1;\ p,p)}_{\Theta^{(2)}_{\mathrm{Conv2d}}}(\underline{\boldsymbol{V}}_2), \tag{7}$$

$$\underline{\boldsymbol{S}} = \mathsf{Conv2d}^{(1,1;\ 1,1;\ 0,0)}_{\Theta^{(s)}_{\mathrm{Conv2d}}}(\underline{\boldsymbol{X}}) \quad \text{(channel matching)}, \tag{8}$$

$$\mathsf{ResnetBlock2D}_{\Theta_{\mathrm{ResnetBlock2D}}}(\underline{\boldsymbol{X}}, \boldsymbol{h}) = \underline{\boldsymbol{S}} + \underline{\boldsymbol{W}}_2. \tag{9}$$

[3]

[3] Diffusers ResnetBlock2D implementation:
https://github.com/huggingface/diffusers/blob/main/src/diffusers/models/resnet.py.

U-Net Construction Blocks (Down/Up/Mid)

DownBlock2D

Definition (DownBlock2D)

Taking the number of residual layers within the level $n \in \mathbb{Z}_{>0}$ as a hyperparameter, and for learnable parameters $\Theta_{\text{DownBlock2D}} = \left(\{\Theta_{\text{Res}}^{(r)}\}_{r=1}^{n},\ \Theta_{\text{Down}}\right)$,

$$\underline{\boldsymbol{H}}_0 = \underline{\boldsymbol{X}}, \quad \underline{\boldsymbol{H}}_r = \text{ResnetBlock2D}_{\Theta_{\text{Res}}^{(r)}}(\underline{\boldsymbol{H}}_{r-1}, \boldsymbol{h}) \ (r = 1, \dots, n) \tag{10}$$

$$\text{DownBlock2D}_{\Theta_{\text{DownBlock2D}}}(\underline{\boldsymbol{X}}, \boldsymbol{h}) = \text{Downsample2D}_{\Theta_{\text{Down}}}^{(2)}(\underline{\boldsymbol{H}}_n). \tag{11}$$

4

[4] Diffusers block implementation: `https://github.com/huggingface/diffusers/blob/main/src/diffusers/models/unet_2d_blocks.py`.

UpBlock2D

Definition (UpBlock2D)

Combining the skip connection $\underline{\boldsymbol{S}}$ and the input from the bottom $\underline{\boldsymbol{X}}$ with Concat, and for learnable parameters $\Theta_{\text{UpBlock2D}} = \left(\{\Theta_{\text{Res}}^{(r)}\}_{r=1}^{n},\ \Theta_{\text{Up}}\right)$,

$$\underline{\boldsymbol{Y}}_0 = \text{Concat}\left(\text{Upsample2D}_{\Theta_{\text{Up}}}^{(2)}(\underline{\boldsymbol{X}}),\ \underline{\boldsymbol{S}}\right), \tag{12}$$

$$\underline{\boldsymbol{Y}}_r = \text{ResnetBlock2D}_{\Theta_{\text{Res}}^{(r)}}(\underline{\boldsymbol{Y}}_{r-1}, \boldsymbol{h})\ (r = 1, \ldots, n), \tag{13}$$

$$\text{UpBlock2D}_{\Theta_{\text{UpBlock2D}}}(\underline{\boldsymbol{X}}, \underline{\boldsymbol{S}}, \boldsymbol{h}) = \underline{\boldsymbol{Y}}_n. \tag{14}$$

5

[5] Implementation reference: `https://github.com/huggingface/diffusers/blob/main/src/diffusers/models/unet_2d_blocks.py`.

Definition (MidBlock2D **(Using Self/Cross Attention))**

For learnable parameters $\Theta_{\mathrm{MidBlock2D}} = \left(\Theta^{(1)}_{\mathrm{Res}},\ \Theta^{\mathrm{self}}_{\mathrm{MHA}},\ \Theta^{\mathrm{cross}}_{\mathrm{MHA}},\ \Theta^{(2)}_{\mathrm{Res}}\right)$ and text context $\boldsymbol{C} \in \mathbb{R}^{M \times d_{\mathrm{ctx}}}$,

$$\underline{\boldsymbol{A}}_0 = \mathsf{ResnetBlock2D}_{\Theta^{(1)}_{\mathrm{Res}}}(\underline{\boldsymbol{X}}, \boldsymbol{h}), \tag{15}$$

$$\boldsymbol{X}_{\mathrm{flat}} = \mathsf{flatten}_{(H,W)}(\underline{\boldsymbol{A}}_0) \in \mathbb{R}^{(HW) \times d_{\mathrm{in}}}, \tag{16}$$

$$\boldsymbol{B}_1 = \mathsf{MultiheadAttention}_{\Theta^{\mathrm{self}}_{\mathrm{MHA}}}(\boldsymbol{X}_{\mathrm{flat}}, \boldsymbol{X}_{\mathrm{flat}}), \tag{17}$$

$$\boldsymbol{B}_2 = \mathsf{MultiheadAttention}_{\Theta^{\mathrm{cross}}_{\mathrm{MHA}}}(\boldsymbol{B}_1, \boldsymbol{C}), \tag{18}$$

$$\underline{\boldsymbol{A}}_1 = \mathsf{unflatten}_{(H,W)}(\boldsymbol{B}_2), \tag{19}$$

$$\mathsf{MidBlock2D}_{\Theta_{\mathrm{MidBlock2D}}}(\underline{\boldsymbol{X}}, \boldsymbol{h}, \boldsymbol{C}) = \mathsf{ResnetBlock2D}_{\Theta^{(2)}_{\mathrm{Res}}}(\underline{\boldsymbol{A}}_1, \boldsymbol{h}). \tag{20}$$

6

Final Projection: Conv1x1

Definition (Conv1x1 (Final Projection))

We define $\text{Conv1x1}_{\Theta_{\text{out}}} := \text{Conv2d}_{\Theta_{\text{out}}}^{(1,1;\ 1,1;\ 0,0)}$ and use it for the mapping to RGB output $\mathbb{R}^{3\times H\times W}$.

Why U-Net Reaches the Entire Area "Shallowly": Quantitative Comparison of Receptive Fields

Receptive Field of a Pure CNN (Conv2d only)

When L layers of Conv2d with kernel size $k = 3$, stride 1, and padding 1 are stacked, the **receptive field** in one dimension is

$$R_{\text{pure}}(L) = 1 + (k-1)L = 1 + 2L. \tag{21}$$

The condition to reach the entire width W is $R_{\text{pure}}(L) \geq W$, i.e.,

$$L \;\geq\; \frac{W-1}{2}. \tag{22}$$

Receptive Field of U-Net (with staged Downsample2D)

Performing Downsample2D$^{(2)}$ with stride 2 L times at each level, and performing n_ℓ 3×3 Conv2d (stride 1) at each resolution, one step at the final (coarsest) level corresponds to 2^L pixels in the original resolution. Therefore, the receptive field converted to the original resolution is

$$R_{\text{unet}} = \; 1 \; + \; \sum_{\ell=0}^{L} \left(2^{\ell}\right) \cdot (k-1)\, n_\ell \; = \; 1 \; + \; 2 \sum_{\ell=0}^{L} 2^{\ell} n_\ell. \tag{23}$$

If we uniformly set $n_\ell = n$,

$$R_{\text{unet}} = 1 + 2n\,(2^{L+1} - 1). \tag{24}$$

Theorem: $\mathcal{O}(\log W)$ Depth for U-Net

Theorem (U-Net reaches the entire area with $\mathcal{O}(\log W)$ depth)

Assuming $k = 3$ and $n \geq 1$ layers at each level, the sufficient condition $R_{\text{unet}} \geq W$ to reach the entire width W is

$$L \;\geq\; \log_2\!\left(\frac{W-1}{2n} + 1\right) - 1. \tag{25}$$

*Therefore, the required number of levels L is $\mathcal{O}(\log W)$, which is **significantly fewer layers** to express dependencies from end to end compared to the $\mathcal{O}(W)$ of a pure CNN (*`Conv2d`* only) in (22).*

Full Definition of U-Net and VAE Decoder "as Functions"

The U-Net (Conditional) Overall Function

Definition (Parametric Function of UNet2DConditionModel)

The inputs are latent $\underline{\boldsymbol{Z}} \in \mathbb{R}^{C \times H \times W}$, time $t \in \mathbb{R}$, and text embedding sequence $\boldsymbol{C} \in \mathbb{R}^{M \times d_{\text{ctx}}}$. The learnable parameter vector is

$$\boldsymbol{\Theta}_{\text{U}} = \left(\Theta_{\text{TE}},\ \{\Theta_\ell^{\downarrow}\}_{\ell=1}^{L},\ \Theta^{\text{mid}},\ \{\Theta_\ell^{\uparrow}\}_{\ell=1}^{L},\ \Theta^{\text{out}}\right) \tag{26}$$

Injecting the time embedding $\boldsymbol{h} = \mathsf{TimestepEmbedding}_{\Theta_{\text{TE}}}(t)$ into each residual block, (cont.)

The U-Net (Conditional) Overall Function (continued)

Definition (Parametric Function of UNet2DConditionModel)

$$\underline{\boldsymbol{D}}_0 = \underline{\boldsymbol{Z}}, \tag{27}$$

$$\underline{\boldsymbol{D}}_\ell = \text{DownBlock2D}_{\Theta_\ell^{\downarrow}}\big(\underline{\boldsymbol{D}}_{\ell-1},\ \boldsymbol{h}\big), \quad \ell = 1, \dots, L, \tag{28}$$

$$\underline{\boldsymbol{B}} = \text{MidBlock2D}_{\Theta^{\text{mid}}}\big(\underline{\boldsymbol{D}}_L,\ \boldsymbol{h},\ \boldsymbol{C}\big), \tag{29}$$

$$\underline{\boldsymbol{U}}_L = \text{UpBlock2D}_{\Theta_L^{\uparrow}}\big(\underline{\boldsymbol{B}},\ \underline{\boldsymbol{D}}_L,\ \boldsymbol{h}\big), \tag{30}$$

$$\underline{\boldsymbol{U}}_{\ell-1} = \text{UpBlock2D}_{\Theta_{\ell-1}^{\uparrow}}\big(\underline{\boldsymbol{U}}_\ell,\ \underline{\boldsymbol{D}}_{\ell-1},\ \boldsymbol{h}\big), \quad \ell = L, \dots, 1, \tag{31}$$

$$\underline{\hat{\boldsymbol{E}}} = \text{Conv1x1}_{\Theta^{\text{out}}}(\underline{\boldsymbol{U}}_0) \in \mathbb{R}^{C \times H \times W}, \tag{32}$$

$$\underline{\hat{\boldsymbol{E}}} = \text{UNet2DConditionModel}_{\boldsymbol{\Theta}_{\text{U}}}(\underline{\boldsymbol{Z}},\ t,\ \boldsymbol{C})\,. \tag{33}$$

7

[7] Diffusers UNet2DConditionModel (U-Net Conditional): https://huggingface.co/docs/diffusers/api/models/unet2d.

VAE Decoder Overall Function

Definition (Decoder **(VAE Decoder))**

For input latent $\underline{\boldsymbol{Z}} \in \mathbb{R}^{C_z \times H_z \times W_z}$, using learnable parameters

$$\boldsymbol{\Theta}_{\text{Dec}} = \left(\Theta^{\text{in}},\ \{\Theta_\ell^{\uparrow}\}_{\ell=1}^{L_d},\ \Theta^{\text{out}}\right) \tag{34}$$

$$\underline{\boldsymbol{H}}_0 = \text{ResnetBlock2D}_{\Theta^{\text{in}}}(\underline{\boldsymbol{Z}}, \mathbf{0}) \quad \text{(no time dependence, so } \boldsymbol{h} = \mathbf{0}), \tag{35}$$

$$\underline{\boldsymbol{H}}_\ell = \text{ResnetBlock2D}_{\Theta_\ell^{\uparrow}}\Big(\text{Upsample2D}^{(2)}_{\Theta_{\text{Up}}^{(\ell)}}(\underline{\boldsymbol{H}}_{\ell-1}),\ \mathbf{0}\Big), \quad \ell = 1, \ldots, L_d, \tag{36}$$

$$\hat{\underline{\boldsymbol{X}}} = \text{Conv1x1}_{\Theta^{\text{out}}}(\underline{\boldsymbol{H}}_{L_d}) \in \mathbb{R}^{3 \times H \times W}, \quad (H = 2^{L_d} H_z,\ W = 2^{L_d} W_z). \tag{37}$$

AutoencoderKL Decoder Mapping

Definition (AutoencoderKL (Decoder part))

We define the decoder mapping $\mathcal{D}$ of AutoencoderKL as

$$\mathcal{D}_{\mathbf{\Theta}_{\text{Dec}}} : \mathbb{R}^{C_z \times H_z \times W_z} \to \mathbb{R}^{3 \times (2^{L_d} H_z) \times (2^{L_d} W_z)}, \quad \mathcal{D}_{\mathbf{\Theta}_{\text{Dec}}}(\underline{\boldsymbol{Z}}) = \text{Decoder}_{\mathbf{\Theta}_{\text{Dec}}}(\underline{\boldsymbol{Z}}) \quad (38)$$

[8]

[8] Diffusers AutoencoderKL implementation (includes Decoder): `https://github.com/huggingface/diffusers/blob/main/src/diffusers/models/autoencoder_kl.py`, API: `https://huggingface.co/docs/diffusers/api/models/autoencoderkl`.

Summary

Summary

- **Mathematical description of architectures**: We **formally defined as functions** the U-Net and VAE decoder.

Summary

- **Mathematical description of architectures**: We **formally defined as functions** the U-Net and VAE decoder.
- **Explanation of variable I/O sizes**: We confirmed that U-Net/VAE satisfy variable input/output sizes because each layer is defined in a **form independent of spatial size**.

Summary

- **Mathematical description of architectures**: We **formally defined as functions** the U-Net and VAE decoder.
- **Explanation of variable I/O sizes**: We confirmed that U-Net/VAE satisfy variable input/output sizes because each layer is defined in a **form independent of spatial size**.
- **Explanation of differences from the proposal**: We clarified the configuration of the U-Net in image generation AIs is different from the originally proposed form.

Next Lecture Preview

Next Lecture Preview

Next time, we will explain the **Text Encoder**.

[1] Jonathan Ho, Ajay Jain, and Pieter Abbeel.
Denoising diffusion probabilistic models.
In Advances in Neural Information Processing Systems (NeurIPS), 2020.

[2] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine.
Elucidating the design space of diffusion-based generative models.
In Advances in Neural Information Processing Systems (NeurIPS), 2022.

[3] Diederik P. Kingma and Max Welling.
Auto-encoding variational bayes.
In International Conference on Learning Representations (ICLR), 2014.

[4] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer.
High-resolution image synthesis with latent diffusion models.
In IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2022.

[5] Olaf Ronneberger, Philipp Fischer, and Thomas Brox.
U-net: Convolutional networks for biomedical image segmentation.
In Medical Image Computing and Computer-Assisted Intervention (MICCAI), 2015.

[6] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole.
Score-based generative modeling through stochastic differential equations.
In International Conference on Learning Representations (ICLR), 2021.