扩散模型的前身——VAE的数学原理和模型

可能大多数人学习扩散模型都是从 2020年的 DDPM 开始的，或者是更早的2015年论文——《Diffusion Probabilistic Models》。但我个人选择从 VAE 入手。之所以这么选，是因为我发现 DDPM 的许多核心数学推导，其实都可以直接从 VAE 的框架稍作修改即可得到，理解了 VAE 之后再看 DDPM 会轻松很多。

而且，如果你已经对扩散模型有一定了解，那你一定知道：在 Latent Diffusion Models（LDM）中，VAE 作为“编码器”把高维的像素空间压缩到低维的隐空间，这一操作极大地降低了后续扩散过程的计算成本，也带来了显著的性能提升。更重要的是，VAE 的原始论文获得了 NeurIPS 的 Test of Time Award，是生成模型领域真正经得起时间考验的经典之作，值得每一位想深入研究生成模型的人认真学习。

下面我将系统地介绍 VAE 的数学原理，并穿插一些直观的理解和思考。

VAE来自论文《Auto-Encoding Variational Bayes》（https://arxiv.org/abs/1312.6114）。中文翻译过来叫做“变分自编码器”。

在了解VAE之前，我们简单理解一下AE（Autoencoder，自编码器）。

1. Autoencoder

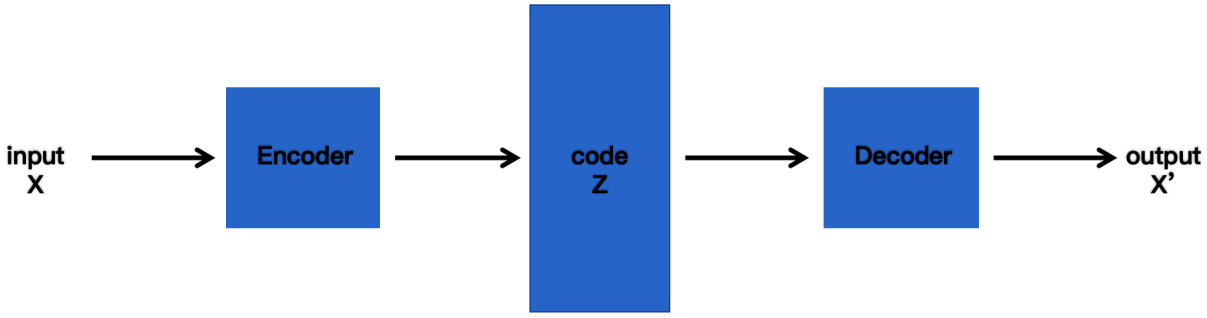
Autoencoder 是一种特殊的 **神经网络结构**，常用于 **无监督学习** （虽然比无监督学习还要早，一般可以追溯到1986年的《Learning representations by back-propagating errors》，https://www.nature.com/articles/323533a0）中去学习数据的有效表示（即编码），或者说提取最核心的特征。

它有两个组件，编码器和解码器。编码器负责将数据压缩为中间向量（连续的），解码器负责将从压缩的中间向量复原回来。

它的目标是把输入数据压缩成一个较低维度的表现（latent code），再从这个表示重建原始输入。
训练时网络被训练成输入 ≈ 输出，因此名字叫 **auto-encoder（自编码器）**。

简单来说：

- 它 **学习压缩数据**, 找出核心特征；
- 再从压缩结果重建原数据；
- 在这个过程中学到的数据编码就是我们关注的“潜在表示”



数学公式很简单：

$$h = f(x), x' = g(h),$$

f是编码器函数，g是解码器函数，我们的损失函数就是x, x'的均方误差或者其他的函数。code是向量，是连续的。

AE学会了编码、解码之后，这个模型有什么用?
经过训练的 Autoencoder 不只是一个压缩器，它学到的潜在表示 h 本质上是一种更“紧凑、更有信息量”的特征，这些特征可以被用到分类、聚类、去噪、异常检测等各种任务中。换句话说，当我们把 h 作为特征向量输入给其他模型时，往往比直接用原始高维数据能得到更好的结果。

或者说，我个人理解是：**深度学习在很多任务中的核心都是对现实数据进行有效的处理和理解，然后用这种“更简单、更表达性强”的表示来完成后续任务。**
就像卷积操作、注意力机制一样，本质上都是在学习一种更合适的输入表示，让后续的模型层更容易获得准确的输出；Autoencoder 中的潜在表示 h 也起到了类似的作用——它本质上是对原始数据的一种学习到的低维理解，这种理解能帮助后续模型更好地完成分类、聚类、生成等任务。

有意思的是，最近也有看到论文，说压缩就是去噪的说法，仅用压缩来完成去噪工作——《The Performance of Compression-Based Denoisers》。

拉回来，在此之后，学术界在各种具体领域，基于自编码器的思想，衍生了非常多的自编码器变体，比如去噪自编码（DAE）、Sparse Autoencoder、VQVAE（这个比较重要，可以看作是发展到stable diffusion的一条路径）、GAE等等。但那些都不是我们本文的重点，感兴趣的读者可以了解此后编码器的发展历程。

你可能会想：“AE 不就是做数据压缩和降维吗？那我用 PCA 等传统数学方法不就行了？”

这个问题看起来简单，但核心在于数据的**非线性结构**。传统方法像 PCA 只能做线性降维，它通过寻找数据方差最大的线性投影来压缩信息，这在很多情况下效率很高且易于解释，但它本质上只能处理线性关系。

线性关系和非线性，这是一个非常重要的区别！深度学习的世界中，大多数是非线性的。

而**自编码器（Autoencoder）是基于神经网络的模型，通过多层带非线性激活函数的网络来学习数据的隐式表示，这使得它能够捕捉远比线性方法更加复杂的非线性结构和特征关系。换句话说，AE 并不是简单的压缩，它是一种学习输入数据内在模式和分布的非线性映射**。在许多真实世界的数据（如图像）中，这些非线性结构极为丰富，线性方法往往无法有效表达，而 AE 可以更好地逼近数据的真实分布。

因此，简单地说：**使用非线性神经网络而不是传统线性方法的动机，是为了能够捕捉和建模更复杂的非线性关系，而这些关系在现实世界的数据中普遍存在。**这也正是深度学习相较于传统数学方法的一大优势。

2. Auto-Encoding Variational Bayes

既然AE已经能学习隐藏特征了，为什么还要有VAE?

AE有一个问题是过拟合，也就是“死记硬背”，而VAE记住的是数据分布。

这是因为现实世界图像的分布极其复杂，不仅包含像素级信息，还包含深层语义结构。在生成模型中，用 VAE 来学习潜在概率分布比简单降维（如 PCA 或 AE）要更适合生成任务，因为它不仅压缩数据，还学习如何从随机潜在变量采样来生成真实数据。Stable Diffusion 使用 VAE 作为 latent 空间的构造器，这个 latent 空间的质量直接影响扩散去噪过程的效果，而社区和研究界确实对改进 VAE 的编码质量表达了关注。

同时，像《Golden Noise for Diffusion Models: A Learning Framework》这样的论文提出了**从噪声分布视角改进扩散过程**的方向，强调了“不同噪声及其结构对生成质量的影响”这一更深层的问题。

总之，隐空间究竟什么样是最好的，最适合模型学习生成图像、处理图像，目前没有定论。

数据分布是一个比较抽象的概念，我再详细说明一下，按照我个人的理解：
扩散模型学到的是一个分布，在这个分布中，同类的图像会聚集在一起，比如我从高斯分布取一个点，这个点去噪出来是只猫，那么这个点的附近去噪出来的图像，是类似的图像，可能是别的动物什么的。这样扩散模型就不单单是死记硬背，一一对应，而是真正的能产生没有见过的图像了。

2.1 VAE的数学推导

在深度学习的生成模型中，我们将编码器的输出记为隐变量 Z，输入数据记为 X。

- **编码过程（推断）**：由 X 预测 Z，对应条件概率 $P(Z|X)$。
- **解码过程（生成）**：由 Z 生成 X，对应条件概率 $P(X|Z)$。

模型的完整概率表征为联合分布 $P(X, Z; \theta)$，其中 θ 是模型参数。X 是可观测的图像数据（Observed variable），而 Z 是不可观测的隐变量（Latent variable）。

根据贝叶斯公式，我们有：

$$P(Z|X) = \frac{P(Z)P(X|Z)}{P(X)}$$

- **后验概率** $P(Z|X)$：在观测到数据 X 后，隐变量 Z 的概率分布。
- **先验概率** $P(Z)$：对隐变量的初始假设（通常假设为标准正态分布 $\mathcal{N}(0, I)$）。
- **似然** $P(X|Z)$：在给定隐变量 Z 的情况下，观测到数据 X 的可能性。
- 证据 $P(X)$：观测数据出现的总概率，也称为边缘似然（Marginal Likelihood）。通过对隐变量 Z 进行 **边际化（Marginalization）** 得到：

$$p(x) = \int_z p(x, z) dz = \int_z p(z) p(x|z) dz$$

为了求解模型$P(X, Z; \theta)$的参数，我们需要使用极大似然估计的方法。
核心思想就是**选择那些让实际观察到的数据出现概率最大的参数值作为估计值**。
在数学上可以表示为：

$$\hat{\theta} = \arg\max_{\theta} L(\theta; X)$$

其中：

- X 是你观察到的数据
- θ 是你要估计的参数
- $L(\theta; X)$ 是似然函数，表示在参数为 θ 时，数据 X 出现的概率或概率密度。
  在实际中，我们一般会将似染函数取对数ln，然后求似然函数的极大化来求解参数。

因此，我们的目标是找到参数 θ 极大化对数似然：

$$\ell(\theta; X) = \ln p_\theta(x) = \ln \int_z p_\theta(x, z) dz$$

问题所在： 由于积分算子在对数函数 $\ln$ 内部，对于复杂的生成模型（如神经网络），这个积分是无法求闭式解的（Intractable），导致我们无法直接通过梯度下降来优化 θ。

这里就要开始使用另外一个数学工具了，也就是大名鼎鼎的ELBO（Evidence Lower Bound），证据下界。这个方法在DDPM也同样使用到了。

ELBO的核心思想是，原来的函数很难求解是吧，那我找到一个更简单的替代函数，这个替代函数的结果与极大化原来函数的结果是等价的。

这个函数是什么呢？是原来函数的下界函数，也就是替代函数 <= 原来函数，因为原来函数中的$p(x)$是证据，所以称之为**证据下界函数**，也就是ELBO。通过极大化替代函数，我们也能同步极大化原来函数，因此能求解了。

下面开始推导：
为了解决上述积分难求的问题，我们引入一个已知的简单分布 $q_\phi(z|x)$（由编码器参数 ϕ 决定）来逼近真实的后验分布 $p(z|x)$。

第一步：引入变分分布 $q_\phi(z|x)$
我们对对数似然进行变形，在积分中同时乘上并除以 $q_\phi(z|x)$：

$$\begin{aligned}\ln p_\theta(x) &= \ln \int_z p_\theta(x,z) dz \\ &= \ln \int_z q_\phi(z|x) \frac{p_\theta(x,z)}{q_\phi(z|x)} dz\end{aligned}$$

第二步：利用期望形式改写
将积分形式改写为关于分布 $q_\phi(z|x)$ 的期望：

$$\ln p_\theta(x) = \ln \mathbb{E}_{q_\phi(z|x)} \left[\frac{p_\theta(x,z)}{q_\phi(z|x)} \right]$$

第三步：利用 Jensen 不等式
由于 $\ln$ 是凹函数，根据 Jensen 不等式（即 $\ln \mathbb{E}[X] \geq \mathbb{E}[\ln X]$），我们可以将对数符号移入期望内部：

$$\ln p_\theta(x) \geq \mathbb{E}_{q_\phi(z|x)} \left[\ln \frac{p_\theta(x,z)}{q_\phi(z|x)} \right]$$

不等式右边的这项，即为证据下界 $\mathcal{L}(q,\theta)$，简称 ELBO。

第四步：分解 ELBO
我们从 Jensen 不等式得到的下界出发：

$$\text{ELBO} = \mathbb{E}_{q_\phi(z|x)} \left[\ln \frac{p_\theta(x,z)}{q_\phi(z|x)} \right]$$

1. 利用联合概率公式拆解
   根据概率论乘法法则，联合概率可以写成：$p_\theta(x,z) = p_\theta(x|z)p_\theta(z)$。将其代入上式：

$$\text{ELBO} = \mathbb{E}_{q_\phi(z|x)} \left[\ln \frac{p_\theta(x|z)p_\theta(z)}{q_\phi(z|x)} \right]$$

2. 利用对数性质拆分项
   根据 $\ln(\frac{a \cdot b}{c}) = \ln a + \ln(\frac{b}{c})$，我们可以将分式拆开：

$$\text{ELBO} = \mathbb{E}_{q_\phi(z|x)} \left[\ln p_\theta(x|z) + \ln \frac{p_\theta(z)}{q_\phi(z|x)} \right]$$

由于期望具有线性性质（$\mathbb{E}[A+B] = \mathbb{E}[A] + \mathbb{E}[B]$），进一步拆分为两项：

$$\text{ELBO} = \underbrace{\mathbb{E}_{q_\phi(z|x)}[\ln p_\theta(x|z)]}_{\text{第一项}} + \underbrace{\mathbb{E}_{q_\phi(z|x)} \left[\ln \frac{p_\theta(z)}{q_\phi(z|x)} \right]}_{\text{第二项}}$$

3. 引入 KL 散度
   这里我们需要用到 KL 散度 的定义。对于两个分布 $q(z)$ 和 $p(z)$，它们之间的 KL 散度（也就是两个分布有多像）定义为：

$$D_{KL}(q||p) = \mathbb{E}_{q(z)} \left[\ln \frac{q(z)}{p(z)} \right]$$

观察上面的第二项，它是 $\ln \frac{p}{q}$，正好是 KL 散度定义的倒数（多了个负号）：

$$\mathbb{E}_{q_\phi(z|x)}\left[\ln \frac{p_\theta(z)}{q_\phi(z|x)}\right] = -\mathbb{E}_{q_\phi(z|x)}\left[\ln \frac{q_\phi(z|x)}{p_\theta(z)}\right] = -D_{KL}(q_\phi(z|x)||p_\theta(z))$$

经过上述推导，ELBO 被改写成了神经网络最喜欢的样子：

$$\text{ELBO} = \underbrace{\mathbb{E}_{q_\phi(z|x)}[\ln p_\theta(x|z)]}_{\text{重构项 (Reconstruction)}} - \underbrace{D_{KL}(q_\phi(z|x)||p_\theta(z))}_{\text{正则化项 (Regularization)}}$$

怎么理解这两项?

- **第一项（重构项）**：
  - **含义**：在给定隐变量 z 的情况下，重建原始输入 x 的对数似然。
  - **深度学习实现**：这其实就是 **Decoder（解码器）** 的输出与原始输入之间的差异（比如 MSE 均方误差或交叉熵）。我们希望这一项越大越好（即重构得越准越好）。
- **第二项（KL 散度项）**：
  - **含义**：衡量编码器输出的分布 $q_\phi(z|x)$ 与我们预设的先验分布 $p_\theta(z)$（通常是标准高斯分布 $\mathcal{N}(0, I)$）有多像。
  - **深度学习实现**：它像一个**惩罚项**。如果没有这一项，模型为了重构精确，会给每个 x 分配一个极小的 z 区域，导致隐空间支离破碎。有了这一项，隐空间就会被约束成一个连续、紧凑的高斯分布。

总结：
原本我们要极大化难以计算的 $\ln p(x)$，现在我们转而极大化 ELBO。在实际写代码训练时，我们通常会将 ELBO 取负号，转化为极小化 Loss：

$$\text{Loss} = -\text{ELBO} = \text{重构损失} + \text{KL 散度损失}$$

有人可能会问：为什么极大化 ELBO 有效?
通过极大化这个下界，我们实际上在同时做两件事：

1. 尽可能提高观测数据 X 出现的概率（即训练生成器）。
2. 让我们的近似后验分布 $q_\phi(z|x)$ 尽可能接近真实的后验分布。

到这里，我们有了数学上的loss了，但是还没完，我们要把这个数学公式放到计算机上，还需要考虑更多的东西——如何在神经网络中实现?

在实际训练中，我们会遇到一个极其关键的工程问题——**随机性带来的梯度断裂**。

遇到的新坑：反向传播不通了!
观察 ELBO 的第一项（重构项）：$\mathbb{E}_{q_\phi(z|x)}[\ln p_\theta(x|z)]$。

在训练 VAE 时，流程如下：

1. **编码器**输出一个分布的参数（通常是均值 μ 和方差 σ^2）。
2. 从这个分布 $q_\phi(z|x) = \mathcal{N}(z; \mu, \sigma^2)$ 中**随机采样**出一个 z。
3. **解码器**根据这个 z 重构出 x。

问题在于“随机采样”：
采样（Sampling）是一个随机过程，它不是一个可微的算子。就像你在程序里调用了一个 random() 函数，梯度传到这里就“断”了，你没法通过反向传播来更新编码器（Encoder）的参数 ϕ。

这里就需要有个妙招——重参数化技巧 (Reparameterization Trick)。
为了解决这个问题，研究者们提出了**重参数化**。核心思想是：将随机性从需要优化的参数路径中“剥离”出来。

我们不再直接从 $\mathcal{N}(\mu, \sigma^2)$ 采样，而是：

1. 先从一个标准正态分布中采样一个无参数的随机噪声 $\epsilon \sim \mathcal{N}(0, 1)$。
2. 通过一个确定的变换公式得到 z：
3. $$z = \mu + \sigma \odot \epsilon$$

   （其中 $\odot$ 表示逐元素相乘）

为什么这招管用?

- **对于 z 来说**：它依然是一个服从 $\mathcal{N}(\mu, \sigma^2)$ 的随机变量，数学本质没变。
- **对于梯度来说**：μ 和 σ 现在是公式中的常数项/变量项，而随机性全在 ϵ 身上。梯度可以顺着 $z \to \mu$ 和 $z \to \sigma$ 的路径顺利回传给编码器。

下面对损失函数进行最终的处理：
在训练中，我们通过极小化负的证据下界（$-\text{ELBO}$）来求解参数。总损失函数定义为：

$$\text{Loss} = \underbrace{-\mathbb{E}_{q_\phi(z|x)}[\ln p_\theta(x|z)]}_{\text{Reconstruction Loss (重构损失)}} + \underbrace{D_{KL}(q_\phi(z|x)||p_\theta(z))}_{\text{KL Loss (KL散度损失)}}$$

1. 重构损失 (Reconstruction Loss)
   这一项决定了模型“画得像不像”。

- **数学背景：** 期望项 $\mathbb{E}_{q_\phi(z|x)}[\ln p_\theta(x|z)]$ 表示在给定隐变量 z 时，生成真实数据 x 的平均对数似然。
- 分布假设： 我们假设解码器的输出服从高斯分布，其均值由解码器网络生成，方差为常数 σ^2：

$$p_\theta(x|z) = \mathcal{N}(x; \text{decoder}(z), \sigma^2 I)$$

- 推导过程： 代入高斯分布的概率密度函数$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$并取对数：

$$\ln p_\theta(x|z) = \underbrace{\ln\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)}_{\text{常数}} - \frac{\|x - \text{decoder}(z)\|^2}{2\sigma^2}$$

- 工程实现： 在优化过程中，常数项不影响梯度。因此，极大化对数似然就等价于极小化均方误差（MSE）：

$$\text{Rec\_Loss} = \|x - \text{decoder}(z)\|^2$$

理想情况：我们要把隐空间 z 中所有可能的点都带入解码器跑一遍，然后根据它们的概率加权平均。但这需要做无穷次的积分，计算机算不出来。

现实方案（蒙特卡洛采样）：既然算不出精确值，我们就从分布 $q(z|x)$ 中随机抽几个点（样本），算出这些点的重构误差，用它们的**平均值**来代表整体。
理论上，采的点越多（L 越大），估计就越准。但在 VAE 训练中，每个 Batch 我们只对每个样本采 **一个** z，原因有二：

- **效率优先**：采 $L = 100$ 个点意味着解码器要运行 100 次，计算量暴增 100 倍，训练太慢。
- **Batch 的弥补作用**：虽然对**单张图片**只采了一个点，但一个 Batch 里有几百张图片。在多次迭代（Iteration）的过程中，模型实际上见过了一个分布里各种各样的 z。这种“积少成多”的效果足以让梯度下降找到正确的方向。

2. KL 散度损失 (KL Divergence Loss)
   如果说重构损失是为了“画得像”，那么 KL 损失就是为了“分得匀”。它强制编码器生成的隐空间分布向标准正态分布靠拢，避免模型退化为普通的自动编码器（AE）。

A. 核心目标与分布假设
我们希望通过最小化 KL 散度，约束编码器输出的分布 $q_\phi(z|x)$ 不要离预设的“模版”太远：

- **变分分布（预测值）：** 编码器针对每个输入 x，预测出一组均值 μ 和方差 σ^2。假设其服从高斯分布：$q_\phi(z|x) = \mathcal{N}(\mu, \sigma^2)$。
- **先验分布（目标值）：** 我们希望隐空间整体满足标准正态分布，即：$p_\theta(z) = \mathcal{N}(0, 1)$。

B. 解析解的数学推导
由于 q 和 p 都是高斯分布，它们的 KL 散度不需要像重构项那样进行随机采样，而是可以直接通过微积分算出**闭式解（Closed-form Solution）**。

其一维形式的推导逻辑如下：

1. 定义展开：
$$D_{KL}(q||p) = \int q(z) \ln \frac{q(z)}{p(z)} dz = \int q(z) \left(\ln q(z) - \ln p(z) \right) dz$$
2. 代入 PDF（概率密度函数）：
   将高斯分布的 $\ln f(z)$ 公式代入，$\ln q(z)$ 会产生 $-\frac{1}{2}\ln(2\pi\sigma^2) - \frac{(z-\mu)^2}{2\sigma^2}$，而 $\ln p(z)$ 会产生 $-\frac{1}{2}\ln(2\pi) - \frac{z^2}{2}$。
3. 利用二阶矩性质：
   积分过程中利用 $\int q(z)(z-\mu)^2 dz = \sigma^2$（方差定义）和 $\int q(z) z^2 dz = \mu^2 + \sigma^2$（二阶矩定义），化简后得到：

$$D_{KL} = \frac{1}{2}\left(\underbrace{\mu^2 + \sigma^2}_{\text{二阶矩}} - \underbrace{\ln(\sigma^2)}_{\text{熵相关}} - \underbrace{1}_{\text{归一化}} \right)$$

C. 工程实现：多维求和

在实际的神经网络中，隐变量 z 通常是 J 维向量。由于我们假设各维度相互独立，总的 KL 损失就是各维度损失的累加：

$$\text{KL\_Loss} = \frac{1}{2} \sum_{j=1}^{J} \left(\mu_j^2 + \sigma_j^2 - \ln(\sigma_j^2) - 1 \right)$$

> **代码小细节：** 在实际编写 PyTorch 或 TensorFlow 代码时，编码器通常输出的是 $\log(\sigma^2)$ 而不是 σ^2。这是因为 σ^2 必须为正，而神经网络输出层的数值范围是 $(-\infty, +\infty)$，取对数后可以更方便网络收敛且保证方差永远为正。

D. 直观理解：它在做什么？

观察公式中的各项：

- $\mu^2 \to 0$：希望分布的中心都在原点附近，防止不同类别的编码结果飘得太远。
- $\sigma^2 - \ln(\sigma^2) \to 1$：这是一个以 $\sigma^2 = 1$ 为最小值的函数，它防止方差塌缩为 0（变回 AE），也防止方差无限大。

一句话总结： KL 损失像一只无形的手，把编码器试图乱丢的样本点重新揉成一团规整的高斯云，从而保证了隐空间的**连续性**（相邻点解码出相似图像）和**完备性**（随机采样也能解码出合理图像）。

总结一下VAE的全貌：
我们从极大化对数似然出发：

1. 发现直接计算 $P(X)$ 太难 $\to$ **引入隐变量** Z。
2. 发现积分难解 $\to$ **引入变分分布** q **并利用 Jensen 不等式得到 ELBO**。
3. 将 ELBO 拆解为 **重构误差 + KL 散度**。
4. 为了让模型可训练 $\to$ **使用重参数化技巧**。

这套逻辑不仅在 VAE 中至关重要，也是后来 **扩散模型 (Diffusion Models)** 推导的基础。在扩散模型中，我们依然是在处理一个由多个隐变量（不同时刻的噪声图）组成的 ELBO。

3. 扩展

在最开始的Stable Diffusion中，VAE是8倍下采样，通道数为4，也就是一张[3, H, W]的图像变成[4, H/8, W/8]。但是我们知道VAE都是有损的，这种设置容易在一些微小的地方产生问题，比如文字之类的。

后续SD改成了8通道，SD3直接弄成了16通道，因为他们发现通道数越多，重建效果越好。

通道数与重建效果、生成效果具有一定的关系，可以看CVPR2025的VAVAE。

此外，关于什么样的隐空间分布适合图像生成，也产生了很多论文，是一个研究方向。

当然也有返璞归真的研究，只在像素空间进行处理，但尚未形成主流。

PS：
可能有错误，欢迎指正！