Stable Diffusion Model

Jianglong Yu

Master of Computer Science, Texas A&M University, College Station, TX

May 26, 2024

References

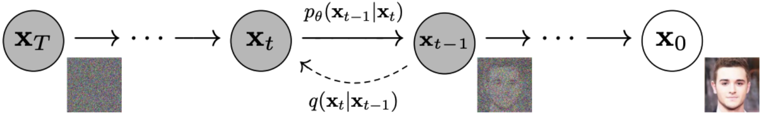
[1] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-Resolution Image Synthesis with Latent Diffusion Models. *CoRR*, abs/2112.10752, 2021. URL `https://arxiv.org/abs/2112.10752`.

[2] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. *CoRR*, abs/2006.11239, 2020. URL `https://arxiv.org/abs/2006.11239`.

[3] Hung-Yi Lee. Machine Learning 2023 Spring Course Slides. National Taiwan University. URL `https://speech.ee.ntu.edu.tw/~hylee/ml/2023-spring.php`.

[4] Lighting AI. Stable Diffusion Explained URL `https://www.youtube.com/watch?v=AQrMWH8aC0Q`.

Diffusion Model



Input: Random Noise (of the size of the image); Output: A clean Image.

Diffusion Model

Diffusion Process

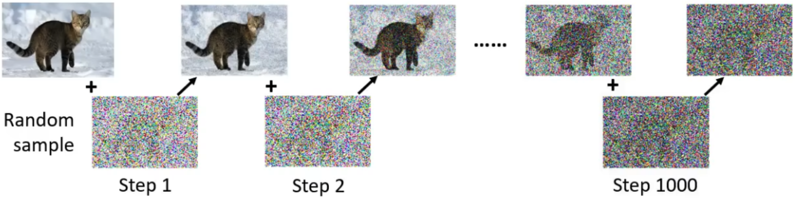


Figure: Diffusion Process. (3)

Noise sampled from Gaussian distribution $\epsilon \sim (0, I)$
Timestep $T \sim Uniform(1, ..., T)$. Make the model learn how to work with different levels of noise throughout the training process

Diffusion Model

Diffusion Process

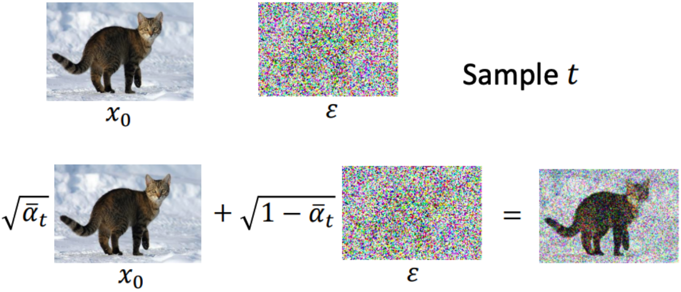


Figure: Diffusion Process. (3)

- Probability theory and the Maklov chain
- $\bar{\alpha}_t = \prod_t^T \alpha_t$, which is cumulative product

Diffusion Model

Denoise Process

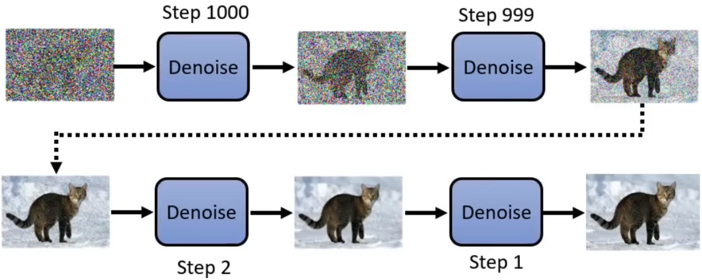


Figure: Diffusion Process. (3)

Only remove the noise step by step

Diffusion Model

Denoise Block

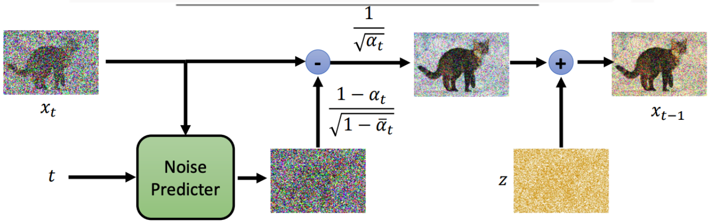


Figure: Diffusion Process. (3)

- Adding a new noise z can make the model more robust
- Noise z is sampled from Gaussian distribution $z \sim (0, I)$
- The noise predictor usually use U-Net

Diffusion Model

Training Model

Algorithm 1 Training
1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \mathrm{Uniform}(\{1, \ldots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on $\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$
6: **until** converged

Algorithm 2 Sampling
1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Figure: Algorithm (2)

The noise we added in as the ground truth for model training

Stable Diffusion Model: Motivation

$$\mathbb{E}_{x,\epsilon\sim N(0,1),t}\left[\|\epsilon-\epsilon_\theta(x_t,t)\|_2^2\right]$$

The most powerful DMs are often computationally demanding.

- Costly Training: UNet has typically $\approx$ 800M parameters; the model takes hundreds of GPU days to train, prone to spend excessive amounts of capacity on modeling imperceptible details
- Costly Evaluation: cost a lot of time and memory, must run the same architecture sequentially for many of steps.

Stable Diffusion Model: Novelty

Highlighted Novelty: Do Diffusion on **Latent Space**, and accept more general types of conditions.

- Operating on **latent space** of powerful pre-trained auto-encoders (1).
- **Less Costly:** Fast sampling, efficient training, one-step decoding to image space.
- **More Flexibility:** More general conditions.

Stable Diffusion Model: Components

Three Major Components:

- **Variational Autoencoder:** Handling perceptual image compression.
  1. Encoder $\mathcal{E}$, Decoder $\mathcal{D}$
  2. $z = \mathcal{E}(x)$ where the RGB image $x \in \mathbb{R}^{H*W*3}$ turns into latent representation $z \in \mathbb{R}^{h*w*c}$, while $\mathcal{D}(z)$ tries to reconstruct x
- **Denoiser:** Latent Diffusion Models(Unet)

$$L_{DM} = \mathbb{E}_{\S,\epsilon\sim N(0,1),t}\left[\|\epsilon - \epsilon_\theta(z_t, t)\|_2^2\right]$$

- **Conditioning Encoder:** can be arbitrary encoder that produces a sequence of tokens.

$$L_{DM} = \mathbb{E}_{\S,\epsilon\sim N(0,1),t}\left[\|\epsilon - \epsilon_\theta(z_t, t, \tau_\theta(y))\|_2^2\right]$$

τ_θ is domain specific encoder used to project y, e.g. τ_θ can be transformers(CLIP) when y are text prompts.

Stable Diffusion Model: Architecture

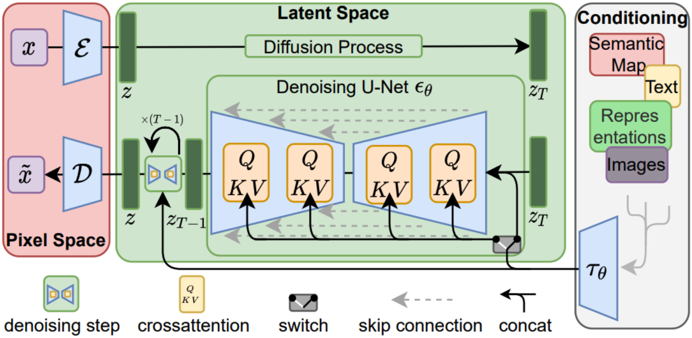


Figure: Architecture of stable diffusion (1)

Stable Diffusion Model: Architecture

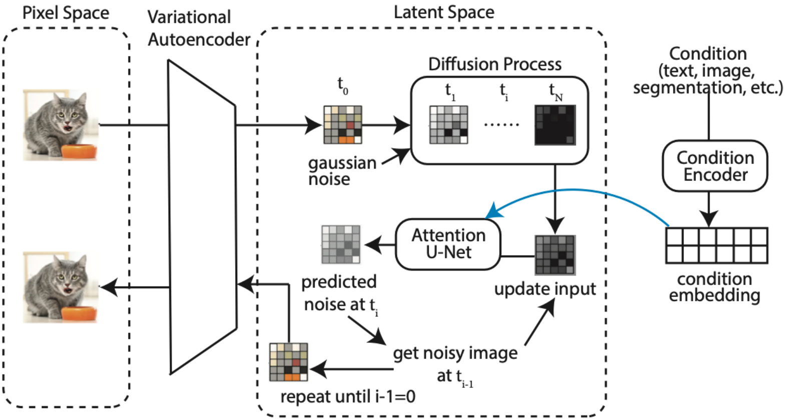


Figure: Architecture of stable diffusion (4)

VAE Architecture

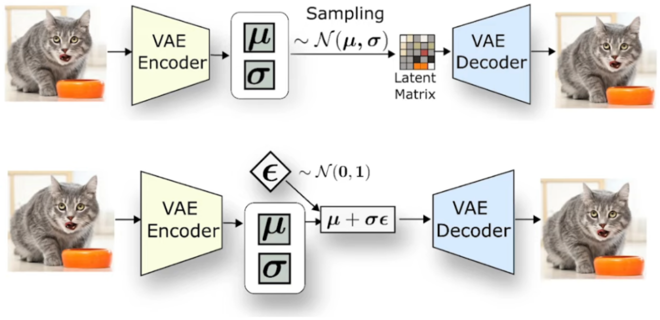


Figure: Architecture of VAE (4)

$$z = \mu + \sigma\epsilon$$

Using the Reparameterization trick.

VAE Train

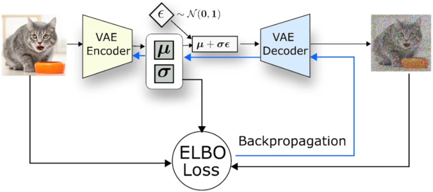


Figure: Architecture of VAE (4)

- **Reconstruction Loss:** Measuring the difference between the original input data and the data reconstructed through the VAE decoder

$$L_{recon} = \sum_{i=1}^{N} \|x_i - \hat{x}_i\|^2$$

- **KL Divergence:** Measurement of the difference between the latent distribution of the encoder output and the a priori latent distribution (usually assumed to be the standard normal distribution)

$$Loss = L_{recon} + \beta \cdot D_{KL}$$

Unet Architecture

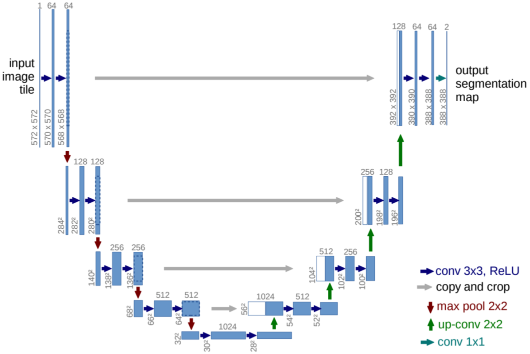


Figure: Architecture of Unet

- **Down sampling Block:** Decreasing size, increasing feature
- **Bottleneck:** Keep the same size, increasing feature
- **Up sampling Block:** increasing size, decreasing feature.

Attention Unet

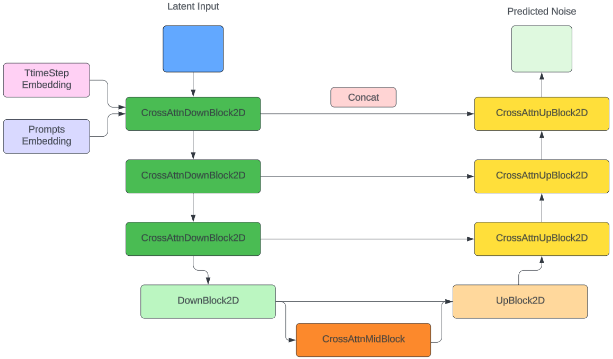


Figure: Architecture of Attention Unet

An attention mechanism is added to the unet, as well as embedding the input prompts.

Component: CrossAttnDownBlock2D

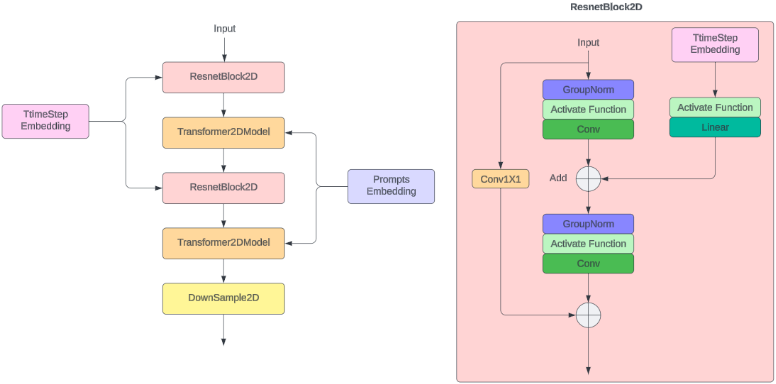


Figure: CrossAttnDownBlock2D

In ResnetBlock, if the input size is different from the final output size, it needs to convert the sizes first before to add.

Component: CrossAttnDownBlock2D

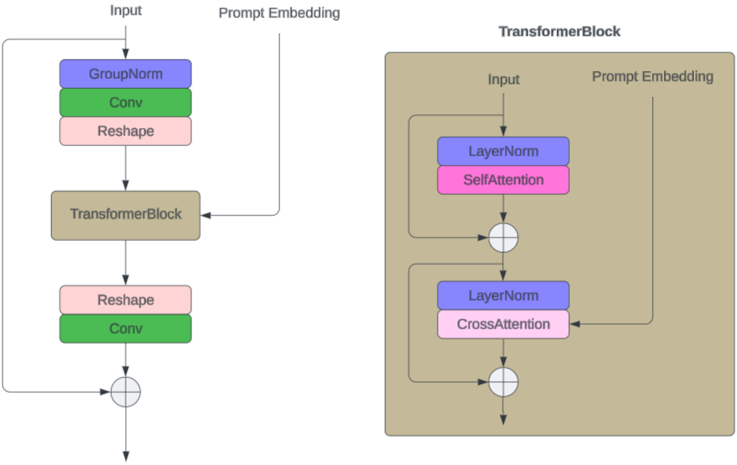


Figure: Transformer2DModel

Component: SelfAttention

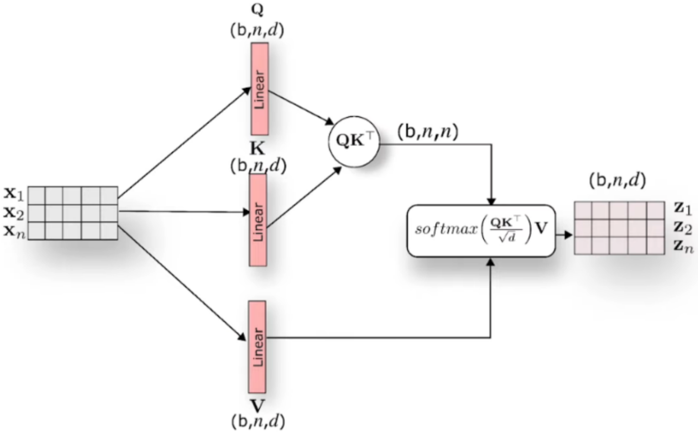


Figure: SelfAttention (4)

Component: Cross Attention

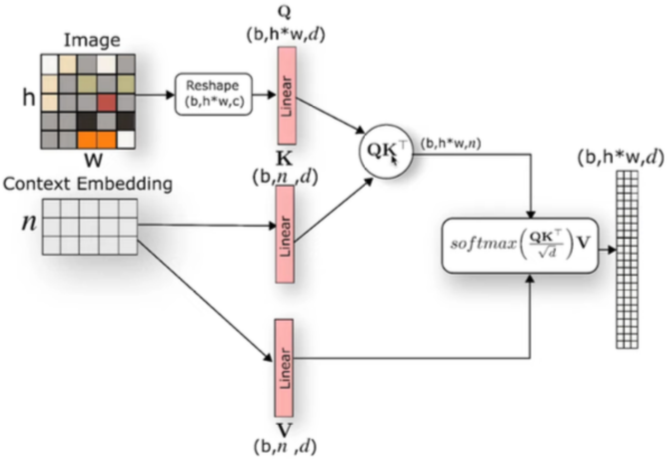
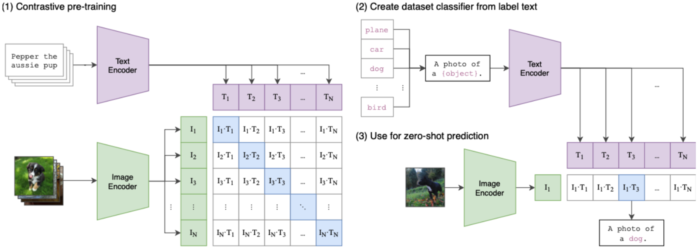


Figure: CrossAttention (4)

Contrastive Language-Image Pretraining(CLIP)



- Standard image models jointly train an image encoder and a linear classifier, whereas CLIP jointly trains an image encoder and a text encoder, to predict the correct pairings of (image, text). Enables zero-shot prediction at inference stage.
- Training: Contrastive loss. Minimize the distance between matched image and text pairs while maximizing the distance between mismatched pairs

Conclusion

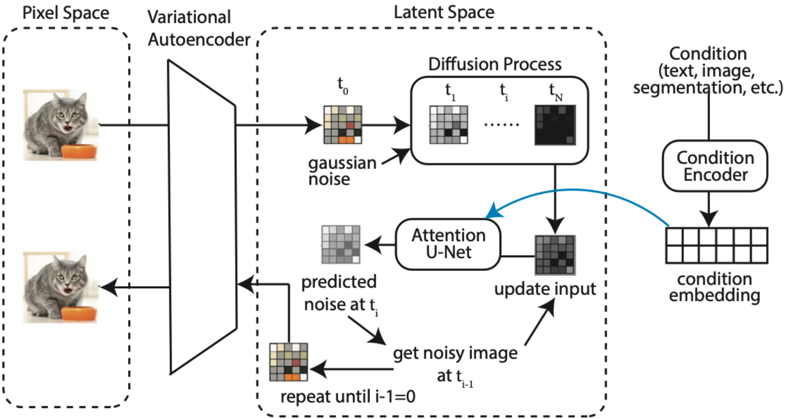


Figure: Architecture of stable diffusion (4)

Thank You!