Stable Diffusion

Parth Pujari

Contents

1 Introduction

Stable Diffusion is a deep learning, text-to-image model released in 2022. It is primarily used to generate detailed images conditioned on text descriptions, though it can also be applied to other tasks such as inpainting, outpainting, and generating image-to-image translations guided by a text prompt.

Stable Diffusion uses a kind of diffusion model (DM), called a latent diffusion model. Diffusion models are trained with the objective of removing successive applications of Gaussian noise on training images, which can be thought of as a sequence of denoising autoencoders.

Stable Diffusion consists of 3 parts: the variational autoencoder (VAE), U-Net, and an optional text encoder. The VAE encoder compresses the image from pixel space to a smaller dimensional latent space, capturing a more fundamental semantic meaning of the image. Gaussian noise is iteratively applied to the compressed latent representation during forward diffusion. The U-Net block, composed of a ResNet backbone, denoises the output from forward diffusion backwards to obtain a latent representation. Finally, the VAE decoder generates the final image by converting the representation back into pixel space.

Figure 1: a photograph of an astronaut riding a horse

2 The Autoencoder

2.1 The Variational Autoencoder

A variational autoencoder (VAE), is an artificial neural network architecture belonging to the families of probabilistic graphical models and variational Bayesian methods.These architectures require neural networks as only a part of their overall structure. The neural network components are typically referred to as the encoder and decoder for the first and second component respectively. The first neural network maps the input variable to a latent space that corresponds to the parameters of a variational distribution. In this way, the encoder can produce multiple different samples that all come from the same distribution. The decoder has the opposite function, which is to map from the latent space to the input space, in order to produce or generate data points. Both networks are typically trained together with the usage of the reparameterization trick, although the variance of the noise model can be learned separately.

2.2 Variational Bayes

Bayes Theorem

$$P(Z|X) = \frac{P(X|Z)P(Z)}{P(X)} = \frac{P(X|Z)P(Z)}{\int_Z P(X|Z')\,dZ'} \tag{1}$$

The marginalization over $\mathbf{Z}$ to calculate $\mathbf{P(X)}$ is usually intractable. Therefore we seek an approximation $\mathbf{Q}(\mathbf{Z}) \approx \mathbf{P}(\mathbf{Z}|\mathbf{X})$

The distribution Q(Z) is restricted to belong to a family of distributions of simpler form than P(Z|X) (e.g. a family of Gaussian distributions), selected with the intention of making Q(Z) similar to the true posterior.

KL Divergence

The most common type of variational Bayes uses the Kullback–Leibler divergence (KL-divergence) of Q from P as the choice of dissimilarity function. This choice makes this minimization tractable. The KL-divergence is defined as

$$D_{KL}(Q||P) \triangleq \sum_Z Q(Z) \log \frac{Q(Z)}{P(Z|X)} \tag{2}$$

Note that Q and P are reversed from what one might expect. This use of reversed KL-divergence is conceptually similar to the expectation-maximization algorithm.

Evidence Lower Bound
Given that,

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

We can rewrite the KL Divergence as,

$$D_{KL}(Q||P) \triangleq \sum_Z Q(Z) \log \frac{Q(Z)}{P(Z|X)} = \sum_Z Q(Z) \left[\log \frac{Q(Z)}{P(Z,X)} + \log P(X) \right]$$

$$= \sum_Z Q(Z)[\log Q(Z) - \log P(Z,X)] + \sum_Z \log P(X)$$

Because $\mathbf{P(X)}$ is constant with respect to $\mathbf{Q(Z)}$, $\sum_Z Q(Z) = 1$,we have,

$$D_{KL}(Q||P) = \sum_Z Q(Z)[\log Q(Z) - \log P(Z,X)] + \log P(X)$$

Which on rearranging gives,

$$\log P(X) = D_{KL}(Q||P) - \sum_Z Q(Z)[\log Q(Z) - \log P(Z,X)] = D_{KL}(Q||P) + \mathcal{L}(Q) \tag{3}$$

As the evidence P(X) is fixed with respect to Z, maximizing the the final term $\mathcal{L}(Q)$ minimizes the KL divergence of Q with respect to P. By appropriate choice of Q, this term becomes tractable and we have an analytical approximation for the posterior $P(Z|X)$ and a lower bound for the log evidence since the KL divergence is non negative.

Mean field Approximation
The variational distribution (Q(Z)) is typically assumed to factorize over some partition of the latent variables $\mathbf{Z}$ into $\mathbf{Z_1}, \mathbf{Z_2}....\mathbf{Z_M}$

$$Q(Z) = \prod_{i=1}^{M} q_i(Z_i|X) \tag{4}$$

It can be shown using the **calculus of variations** (hence the name "variational Bayes") that the "best" distribution q_j^* for each of the factors q_j (in terms of the distribution minimizing the KL divergence, as described above) satisfies:

$$q_j^*(Z_j|X) = \frac{e^{E_{q_j^*}[ln(p(Z,X))]}}{\int e^{E_{q_j^*}[ln(p(Z,X))]} dZ_j} \tag{5}$$

Where $E_{q_j^*}[ln(p(Z,X))]$ is the expectation of the joint probability of the data and the latent variables taken with respect to q^* over all variables not in the partition.

2.3 Probabilistic Modelling of the Autoencoder

The idea of the variational autoencoder is to maximize the likelihood of the data x by the chosen parameterization $p_\theta(x)$ (typically chosen to be a gaussian). The key point to note is that vanilla autoencoders encode data into a latent space that is generally not regular. It is somewhat overfitted to match the data and this creates a problem while generating new data. Come to think of it, this lack of structure among the encoded data into the latent space is pretty normal. Nothing in the task the autoencoder is trained for enforces it to get such organisation: the autoencoder is solely trained to encode and decode with as few loss as possible, no matter how the latent space is organised.

This is the reason why we encode our data into a variational distribution instead of single data points in a latent space. Analogous to the variational bayes formulation, we have:

$$\begin{gathered} Prior : p_\theta(z) \\ Likelihood : p_\theta(x|z) \\ Posterior : p_\theta(z|x) \end{gathered}$$

As seen in the variational Bayes formulation, $p_\theta(x)$ is difficult to calculate and typically intractable, so $p_\theta(z|x)$ is approximated to some $q_\phi(z|x)$

We parameterize the encoder as $\mathbf{E}_\phi$ and the decoder as $\mathbf{D}_\theta$.

The problem is to now find a good probabilistic autoencoder, where the conditional likelihood $p_\theta(x|z)$ is computed by the decoder and the posterior $q_\phi(z|x)$ is computed by the encoder.

As calculated earlier, the **Evidence Lower Bound** is as follows:

$$\mathcal{L}_{\phi\,\theta}(x) = \mathbb{E}_{z\sim q_\phi(.|x)}\left[ln\frac{p_\phi(x,z)}{q_\theta(z|x)}\right] = ln(p_\theta(x)) - D_{KL}(q_\phi(.|x)||p_\theta(.|x)) \tag{6}$$

and the task at hand is to maximize the lower bound (note that here I've written it as a function of x, however the task is to find the appropriate ϕ and θ to maximize $\mathcal{L}$ for every x.[1]

$$\phi^*, \theta^* = \arg\max_{\phi\,\theta} \mathcal{L}_{\phi\,\theta}(x) \tag{7}$$

The following equivalent form of the loss is easier to optimize:

$$\mathcal{L}_{\phi\,\theta}(x) = \mathbb{E}_{z\sim q_\theta(.|x)}[ln(p_\theta(x|z))] - D_{KL}(q_\phi(.|x)||p_\theta(.)) \tag{8}$$

The distributions of $p_\theta(z)$ and $q_\phi(z|x)$ are modelled as Gaussians as $z \sim \mathcal{N}(0, I)$ and $z|x \sim \mathcal{N}(E_\phi(x), \sigma_\phi(x)^2 I)$. $ln(p_\theta(x|z))$ is implemented as $-\frac{\mathbf{1}}{\mathbf{2}}||\mathbf{x} - \mathbf{D}_\theta(\mathbf{z})||_{\mathbf{2}}^{\mathbf{2}}$, since that is up to an additive constant what $x \sim \mathcal{N}(D_\theta(z), I)$ yields. We now use the KL Divergence for Gaussians:

$$\mathcal{L}_{\phi\,\theta}(x) = -\frac{1}{2}\mathbb{E}_{z\sim q_\theta(.|x)}[||x - D_\theta(z)||_2^2] - \frac{1}{2}(N\sigma_\phi(x)^2 + ||\mathbb{E}_\phi(x)||_2^2 - 2Nln(\sigma_\phi(x)) + Const \tag{9}$$

Here N is the dimension of z

[1] We simultaneously maximize the log likelihood and minimize the KL divergence by maximizing the evidence loss

2.4 Reparameterization

$$\phi^*, \theta^* = \arg\max_{\phi\,\theta} \mathcal{L}_{\phi\,\theta}(x)$$

The usual method to find this is gradient descent.
The computation of the following is straightforward,

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

However the following is not as we cannot put the ∇_ϕ inside the expectation.

,

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

The **reparameterization trick** (also known as stochastic backpropagation) bypasses this.

When $z \sim q_\phi(.|x)$ is a Gaussian, $\mathcal{N}(\mu_\phi(x), \Sigma_\phi(x))$, it can be parameterized using $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ as a "standard random number generator" and construct $z = \mu_\phi(x) + L_\phi(x)\epsilon$. Here $L_\phi(x)$ is found using the Cholesky Decomposition. The Cholesky decomposition is the decomposition of a Hermitian, positive-definite matrix into the product of a lower triangular matrix and its conjugate transpose.

$$\Sigma_\phi(x) = L_\phi(x) L_\phi(x)^T \tag{10}$$

Then we get,

$$\nabla_\phi \, \mathbb{E}_{z\sim q_\phi(.|x)} \left[ln \frac{p_\phi(x,z)}{q_\theta(z|x)} \right] = \nabla_\phi \, \mathbb{E}_\epsilon \left[ln \frac{p_\phi(x, \mu_\phi(x) + L_\phi(x)\epsilon)}{q_\theta(\mu_\phi(x) + L_\phi(x)\epsilon|x)} \right] \tag{11}$$

and so we obtained an unbiased estimator of the gradient, allowing **stochastic gradient descent**.

Since we parameterized z, we want to find $q_\phi(z|x)$. Let q_0 be the probability density func. for ϵ,

$$ln \, q_\phi(z|x) = ln \, q_0(\epsilon) - ln|det(\partial_\epsilon(z))|$$

Where $\partial_\epsilon(z)$ is the **Jacobian matrix** of ϵ with respect to z. Since $z = \mu_\phi(x) + L_\phi(x)\epsilon$, this becomes,

$$ln \, q_\phi(z|x) = -\frac{1}{2}||\epsilon||^2 - ln|det(L_\phi(x))| - \frac{n}{2} ln(2\pi)$$

2.5 Neural Network model

The idea is that now our approximation to the posterior $q_\phi(z|x)$ is going to be a neural network and the parameters θ and ϕ are jointly optimized using stochastic gradient descent. I modelled the distribution (of which the parameters in the latent space are to be found) as a Gaussian. The network outputs the mean and log-variance parameters of a factorized Gaussian. I output log-variance instead of the variance directly for numerical stability. The network layers are fully connected layers (or convolutions and maxpools) for the encoder and fully connected (or convolutions and upsampling) for the decoder. The reparameterization is implemented as mentioned in the section above. I use KL divergence as well as binary cross entropy loss. The results as well as the code are on my github.

3 The Diffusion Model

This is based on the paper **Denoising Diffusion Probabilistic Models**. The authors define diffusion probabilistic models as parameterized Markov chains trained using variational inference to produce samples matching the data after finite time. Transitions of this chain are learned to reverse a diffusion process, which is a Markov chain that gradually adds noise to the data in the opposite direction of sampling until signal is destroyed. When the diffusion consists of small amounts of Gaussian noise, it is sufficient to set the sampling chain transitions to conditional Gaussians too, allowing for a particularly simple neural network parameterization.

Some notation is as follows;
x_t is a latent variable (similar to that in variational inference), where x_0 is the image in the dataset while x_T is complete noise.

3.1 Background

We define the Diffusion model as a *latent variable* model of the form

$$p_\theta(x_0) = \int p_\theta(x_{0:T})\, dx_{1:T}$$

where $x_1 ... x_T$ are the latents as defined earlier. Note that they have the same dimensionality as the data x_0. The joint distribution $p_\theta(x_0)$ is called the *reverse process* and is defined as a Markov chain with learned Gaussian transitions starting from $p(x_T) = \mathcal{N}(x_T; \mathbf{0}, \mathbf{I})$ and

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) \qquad p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \boldsymbol{\mu}_\theta(x_t, t), \boldsymbol{\Sigma}_\theta(x_t, t)) \tag{12}$$

The *forward process* is fixed to a Markov chain that gradually adds Gaussian noise to data as per a variance schedule $\beta_1 .... \beta_T$:

$$q(x_{1:T}) = \prod_{t=1}^{T} q(x_t|x_{t-1}) \qquad q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}\, x_{t-1},\, \beta_t \mathbf{I}) \tag{13}$$

Training is performed (similar to variational Bayes) by minimizing the negative log likelihood;

$$\mathbb{E}[-\log p_\theta(x_0)] \leq \mathbb{E}_q \left[-\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)} \right] = \mathbb{E}_q \left[-\log p(x_T) - \sum_{t \geq 1} \log \frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})} \right] =: L \tag{14}$$

The forward process parameters are held constant (as hyperparameters). The forward process admits sampling x_t at an arbitrary time step in closed form, the notation used is : $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$. Then,

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t} x_0,\, (1 - \bar{\alpha}_t)\mathbf{I}) \tag{15}$$

Owing to this closed form, we can efficiently train models by optimizing random terms of L with stochastic gradient descent. L can be variance reduced as:

$$\mathbb{E}_q \left[\underbrace{D_{KL}(q(x_T|x_0) \,||\, p(x_T))}_{L_T} + \sum_{t>1} \underbrace{D_{KL}(q(x_{t-1}|x_t, x_0) \,||\, p_\theta(x_t|x_{t-1}))}_{L_{t-1}} - \underbrace{\log p_\theta(x_0|x_1)}_{L_0} \right] \tag{16}$$

Equation (16) uses KL divergence to directly compare $p_\theta(x_{t-1}|x_t)$ against forward process posteriors, which happen to be tractable when conditioned on x_0,

$$q(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1}; \tilde{\boldsymbol{\mu}}_t(x_t, x_0), \tilde{\beta}_t \mathbf{I}), \text{where} \tag{17}$$

$$\tilde{\boldsymbol{\mu}}_t(x_t, x_0) := \frac{\sqrt{\bar{\alpha}_t - 1}\beta_t}{1 - \bar{\alpha}_t} x_0 + \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} x_t \quad \text{and} \quad \tilde{\beta}_t := \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t}\beta_t \tag{18}$$

Consequently, all KL divergences in equation (16) are comparisons between Gaussians, so they can be calculated in a **Rao-Blackwellized** fashion with closed form expressions instead of high variance Monte Carlo estimates.

3.2 Denoising Autoencoders

3.2.1 Forward Process and L_T

We ignore the fact that the forward process variances β_t are learnable by reparameterization and instead fix them to constants. Thus, in our implementation, the approximate posterior q has no learnable parameters, so L_T is a constant during training and can be ignored.

3.2.2 Reverse Process and $L_{1:T-1}$

Note that we want to optimize random terms of L. Let's discuss our choices in $p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \boldsymbol{\mu}_\theta(x_t, t), \boldsymbol{\Sigma}_\theta(x_t, t))$

First we set $\boldsymbol{\Sigma}_\theta(x_t, t)$ to $\sigma_t^2 \mathbf{I}$ and set σ_t^2 to β_t. That is, we won't be predicting any variances. Second, to represent $\boldsymbol{\mu}_\theta(x_t, t)$ we use a parameterization:

$$L_{t-1} = \mathbb{E}_q \left[\frac{1}{2\sigma_t^2} ||\tilde{\boldsymbol{\mu}}_t(x_t, x_0) - \boldsymbol{\mu}_\theta(x_t, t)||^2 \right] + C \tag{19}$$

Hence, the most straightforward parameterization of $\boldsymbol{\mu}_\theta$ is a model that predicts $\tilde{\boldsymbol{\mu}}_t$, the forward process posterior mean. By reparameterizing equation (15) as $x_t(x_0, \epsilon) = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon$ for $\epsilon \sim \mathcal{N}(0, \mathbf{I})$, we can expand (19) as :

$$L_{t-1} - C = \mathbb{E}_{x_0, \epsilon} \left[\frac{1}{2\sigma_t^2} \left\| \frac{1}{\sqrt{\alpha_t}} \left(x_t(x_0, \epsilon) - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon \right) - \boldsymbol{\mu}_\theta(x_t(x_0, \epsilon), t) \right\|^2 \right] \tag{20}$$

Which reveals that $\boldsymbol{\mu}_\theta$ must predict $\frac{1}{\sqrt{\alpha_t}} \left(x_t(x_0, \epsilon) - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon \right)$ given x_t. Since x_t is available as input to the model, the following parameterization is chosen:

$$\boldsymbol{\mu}_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right) \tag{21}$$

where ϵ_θ predicts ϵ from x_t

Finally we can write $p_\theta(x_{t-1}|xt)$ in its parameterized form and sample x_{t-1} as:

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right) + \sigma_t \mathbf{z} \tag{22}$$

where $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
The parameterization in equation (20) simplifies to:

$$L_{t-1} - C = \mathbb{E}_{x_0,\epsilon}\left[\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}\left\|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)\right\|^2\right] \tag{23}$$

We're left with the mean squared error loss between the actual noise added and the noise predicted! To summarize, we can train the reverse process mean function approximator $\boldsymbol{\mu}_\theta$ to predict $\tilde{\mu}_t$, or by modifying its parameterization, we can train it to predict ϵ.

Algorithm 1: Training

while *not converged* **do**
- $x_0 \sim q(x_0)$
- $t \sim$ Uniform($\{1, ...T\}$)
- $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
- Gradient descent step on $\nabla_\theta \left\|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)\right\|^2$

end

Algorithm 2: Sampling

$x_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T....1$ **do**
- $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = 0$
- $x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right) + \sigma_t \mathbf{z}$

end
return x_0

That sums up the diffusion probabilistic model! Let's move on to implementation.

3.3 Experiments

The paper sets T = 1000 for all its experiments. The forward process constants are set to be constants, linearly increasing from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$

To represent the reverse process I use a U-net framework based on ResNet. The following section describes a U-Net framework for Image segmentation that I developed based on the paper; U-Net: Convolutional Networks for Biomedical Image Segmentation.

4 U-Net

The point of the U-net is to use data augmentation for efficient training. As the name suggests, the U-net is a U shaped network, wherein the dimension in each layer first reduces and then increases. To quote the authors - *" The architecture consists of a contracting path to capture context and a symmetric expanding path that enables precise localization"*.

I would like to note that I will be training a U-net in the context of image segmentation. The typical use of convolutional networks is on classification tasks, where the output to an image is a single class label. However, in many visual tasks, especially in biomedical image processing, the desired output should include localization, i.e., a class label is supposed to be assigned to each pixel.

The paper introduced a structure (for reasons explained in the paper) called the fully convolutional network. The main idea is to supplement a usual contracting network by successive layers, where pooling operators are replaced by upsampling operators. Hence, these layers increase the resolution of the output. In order to localize, high resolution features from the contracting path are combined with the upsampled output. A successive convolution layer can then learn to assemble a more precise output based on this information. One important modification in their architecture is that the upsampling part has a large number of feature channels, which allow the network to propagate context information to higher resolution layers. As a consequence, the expansive path is more or less symmetric to the contracting path, and yields a u-shaped architecture. The network does not have any fully connected layers and only uses the valid part of each convolution, i.e., the segmentation map only contains the pixels, for which the full context is available in the input image. This strategy allows the seamless segmentation of arbitrarily large images by an overlap-tile strategy

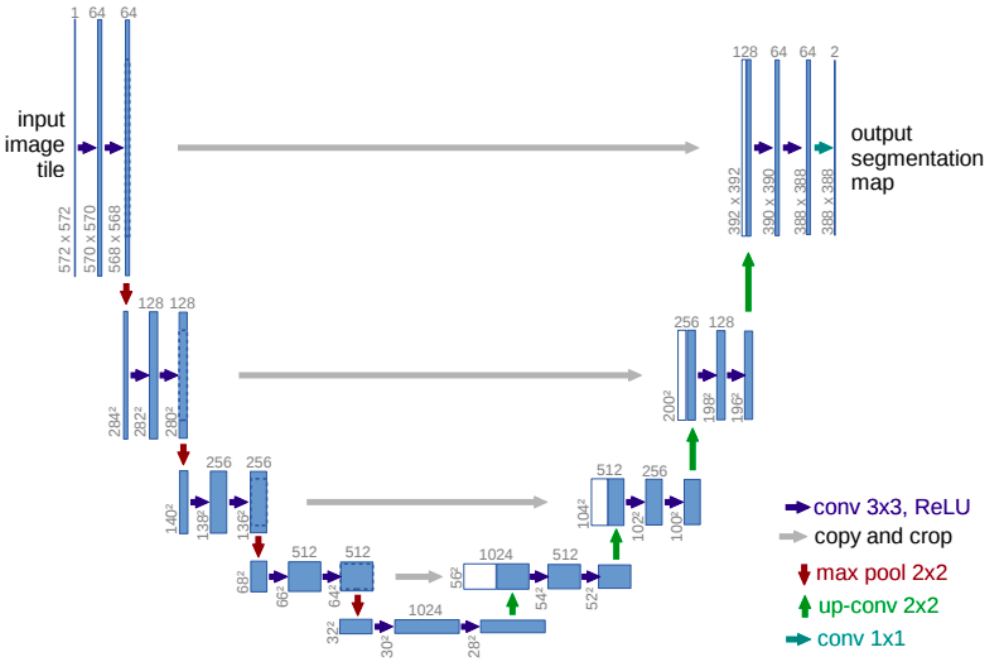

Figure 2: The U-net Structure

5 U-Net for DDPM

The DDPM U-Net must perform the *reverse process*. It takes as input t and $\sqrt{\bar{\alpha}}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$ for an image x_0 and outputs an image that corresponds to ϵ.

The U-Net contains various blocks, (including residuals, attentions) and incorporates time-step embeddings. The implementation details are not specified here.

6 Latent Diffusion

So far, the diffusion U-Net has taken noisy image as an input. We saw that variational encoders can encode images into a latent space and can derive the image back even after the application of noise to the latent space. We utilise these encoders to train faster diffusion U-Nets by performing diffusion in the latent space, i.e.,

7 Tackling a trilemma

We plan to reproduce the paper on "TACKLING THE GENERATIVE LEARNING TRILEMMA WITH DENOISING DIFFUSION GANs" by Zhisheng Xiao, Karsten Kreis and Arash Vahdat. The gist of the paper is as follows.

A wide variety of deep generative models have been developed in the past decade. Yet, these models often struggle with simultaneously addressing three key requirements including: high sample quality, mode coverage, and fast sampling. We call the challenge imposed by these requirements the generative learning trilemma, as the existing models often trade some of them for others. Particularly, denoising diffusion models have shown impressive sample quality and diversity, but their expensive sampling does not yet allow them to be applied in many real-world applications. In this paper, the authors argue that slow sampling in these models is fundamentally attributed to the Gaussian assumption in the denoising step which is justified only for small step sizes. To enable denoising with large steps, and hence, to reduce the total number of denoising steps, they propose to model the denoising distribution using a complex multimodal distribution. They introduce denoising diffusion generative adversarial networks (denoising diffusion GANs) that model each denoising step using a multimodal conditional GAN. Through extensive evaluations, we aim to show that denoising diffusion GANs obtain sample quality and diversity competitive with original diffusion models while being 2000× faster on the CIFAR-10 dataset. Compared to traditional GANs, this model exhibits better mode coverage and sample diversity. To the best of our knowledge, denoising diffusion GAN is the first model that reduces sampling cost in diffusion models to an extent that allows them to be applied to real-world applications inexpensively.

The datasets used in the paper are the CIFAR-10, the stacked MNIST and/or CelebA-HQ all of which are found on Kaggle.