

Stable Diffusion on OASIS Dataset

The readme file should contain a title, a description of the algorithm and the problem that it solves (approximately a paragraph), how it works in a paragraph and a figure/visualisation.

Requirements (This should be removed when submitted)

1. The readme file should contain a title, a description of the algorithm and the problem that it solves (approximately a paragraph), how it works in a paragraph and a figure/visualisation.
2. It should also list any dependencies required, including versions and address reproducibility of results, if applicable.
3. provide example inputs, outputs and plots of your algorithm
4. The read me file should be properly formatted using GitHub markdown
5. Describe any specific pre-processing you have used with references if any. Justify your training, validation and testing splits of the data.

Project Overview

Run

Files

Non-Script

- `model_ckpt*`: models' checkpoints.
- `log*`: training log. (Can use `logger.py` to visualize it.)
- `*_vis*`: visualization of images when training.
- `playground`: we write some code to figure out some algorithms.
- `README.md` / `report.pdf`: The file you current see.

Common

- `util.py`: useful function, such as ssim, positional encoding.
- `module.py`: useful network module, such as resblock.
- `dataset.py`: get dataset from given folder.
- `logger.py`: script that can check the training log.

First Stage

- `model_VAE.py`: VAE and VQVAE model for first stage.
- `model_discriminator.py`: The GAN part when training VAE or VQVAE.
- `prestage_train.py`: Training Script for first stage.

Second Stage

- `pixelCNN.py`: Do pixelCNN when random generation from VQVAE model.
- `model_diffusion.py`: The model of stable diffusion. It's UNet.
- `stable_diffusion.py`: Do stable diffusion when random generation from VQVAE or VAE model.

Results

Diffusion Process GIF

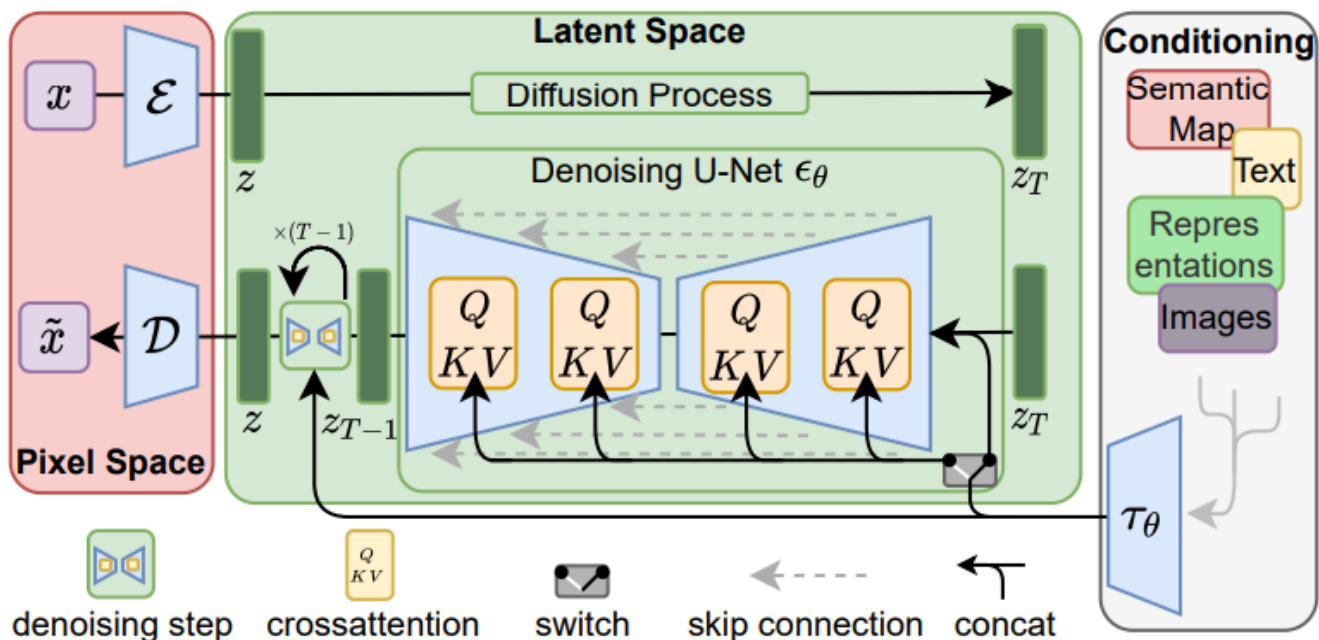
About Stable Diffusion

What is Diffusion Model?

[DDPM \(Denoising Diffusion Probabilistic Models\)](#)

What is Stable Diffusion?

[High-Resolution Image Synthesis with Latent Diffusion Models \(Stable Diffusion\)](#)



Detail in Stable Diffusion

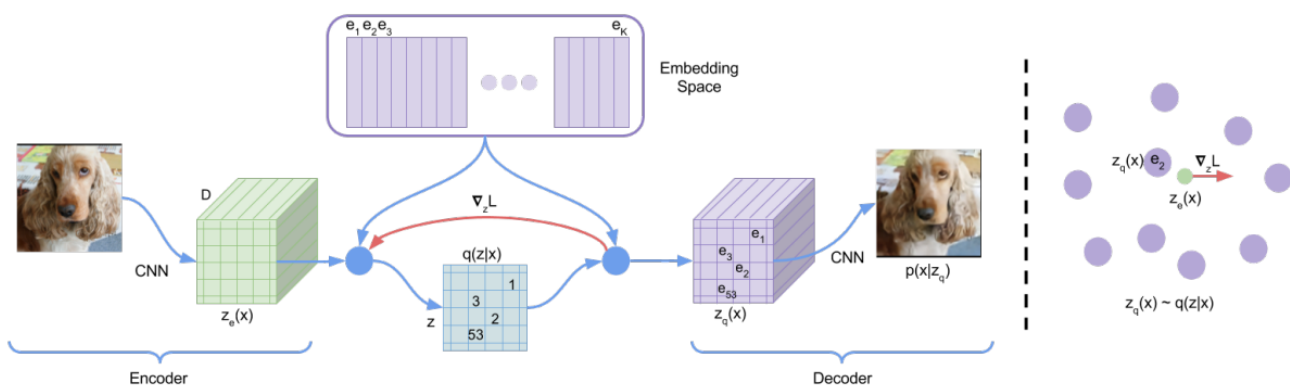


Figure 1: Left: A figure describing the VQ-VAE. Right: Visualisation of the embedding space. The output of the encoder $z(x)$ is mapped to the nearest point e_2 . The gradient $\nabla_z L$ (in red) will push the encoder to change its output, which could alter the configuration in the next forward pass.

Perceptual Image Compression

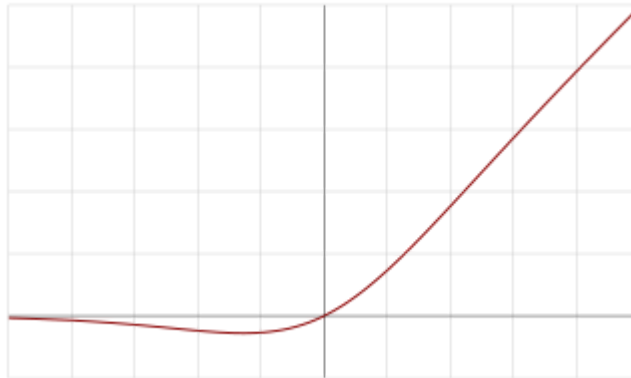
Swish

[SEARCHING FOR ACTIVATION FUNCTIONS, ICLR 2018 workshop](#)

Swish is an activation, which is defined as $f(x) = x \cdot \sigma(\beta x)$.

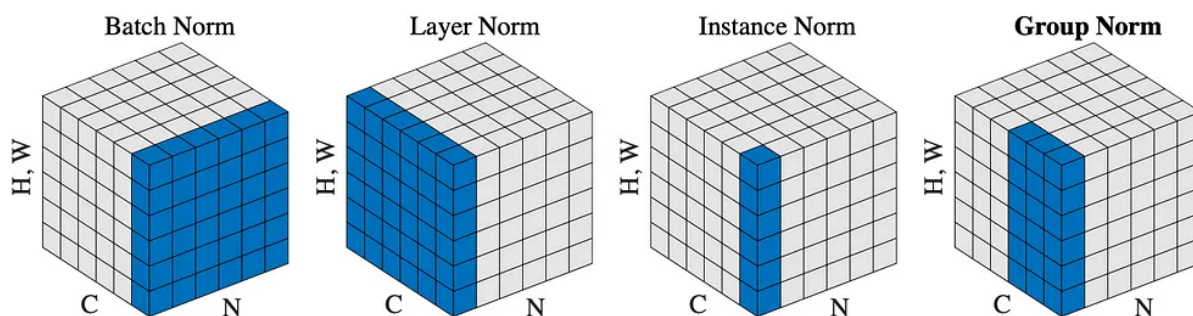
In our code, we set $\beta = 1$ in all the Swish activation, and which is also known as "Sigmoid Linear Unit (SiLU)".

- Update: Pytorch has implementation of [SiLU](#)



Group Normalization

[Group Normalization, ECCV 2018](#)



Normalization: A method to train model faster and more stable through normalization of tinputs by re-centering and re-scaling.

- Batch normalization: Normalization for each channel.
- Layer normalization: Normalization for each sample.
- Instance normalization: Normalization for each sample and each channel.
- Group normalization: Normalization for each sample group.

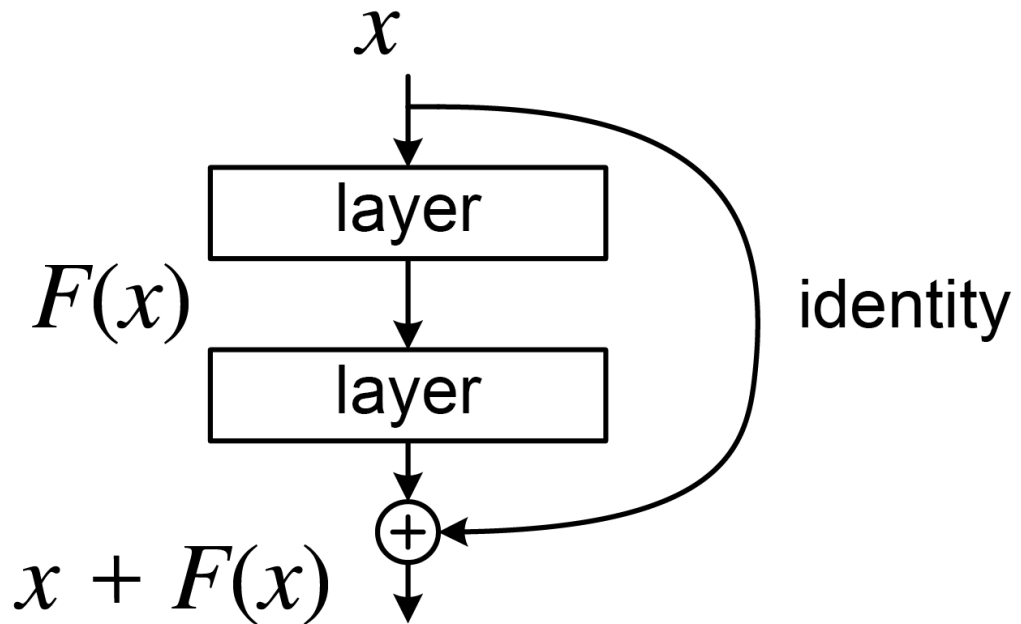
Note: If batch size is large enough, the performance: BN > GN > LN > IN

However, BN has GPU memory issue and cannot set large batch size sometimes.

Thus, we do GN in this task.

ResNet

[Deep Residual Learning for Image Recognition, CVPR 2016](#)



A popular module to very learn deep model by residual learning.

Self-Attention

[Attention Is All You Need, NIPS 2017](#)

Self-attention, also known as scaled dot-product attention, is a crucial concept in the field of natural language processing (NLP) and deep learning, particularly within the context of transformer-based models. In stable diffusion, authors employ self-attention in each feature map. Next, we will introduce what self-attention is.

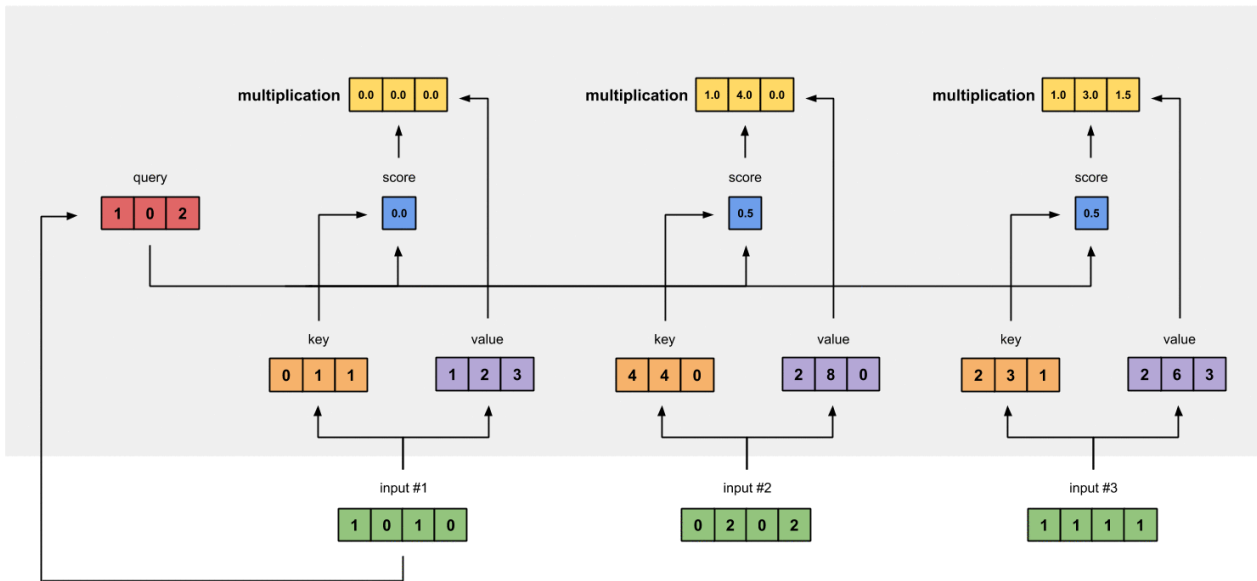
Self-attention involves three key vectors for executing "attention to itself":

1. Query vectors: These query other elements in the sequence.
2. Key vectors: These define the importance relative to the current element.
3. Value vectors: These generate the output vectors.

After generating these three vectors, we follow these steps for each element:

1. Calculate the attention score, which is $W_{current,i} = K_{current} \cdot Q_i$, where $i \in [1, C]$, and C is the input size.
2. Scale and softmax the score, which means $W_{current} = \text{softmax} \left(\frac{W_{current}}{\sqrt{C}} \right)$.
3. Generate the output, $H_{current} = W_{current} * V$.

Finally, to apply this method to convolutions, we can simply replace all linear layers with one-by-one convolutional layers, and everything will be fine.

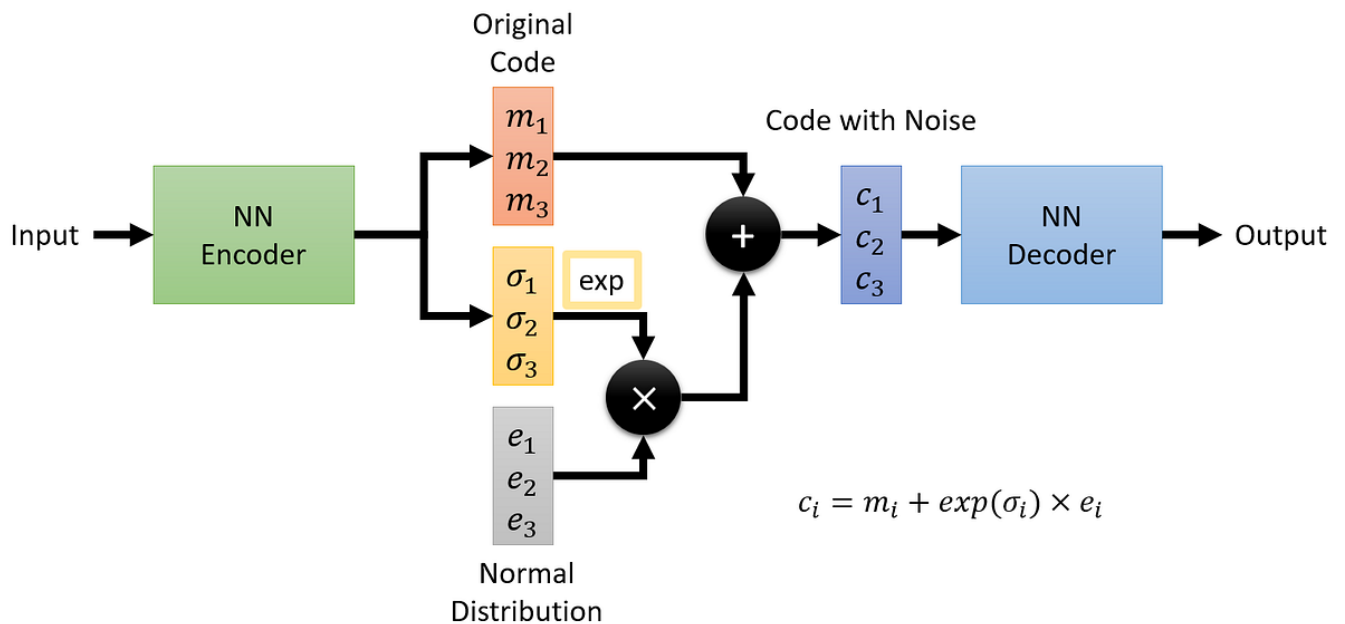


Flow chart of Self-Attention

Variational Autoencoder (VAE)

[Auto-Encoding Variational Bayes, ICLR 2014](#)

This paper introduces what VAE is.



Flow chart of VAE.

The most important thing that VAE aims to solve is to make it easier for us to generate random images from the latent space. We need to know what distribution the latent space, encoded by the encoder, follows, but this problem is quite challenging. Therefore, we can approach it from a different perspective: constraining the latent code generated by the encoder to be similar to a well-known distribution, typically a Gaussian distribution.

But how do we calculate the regularization term? Next, we will explore what the regularization term should be if we use the KL divergence to measure the distance between the latent space and a Gaussian distribution.

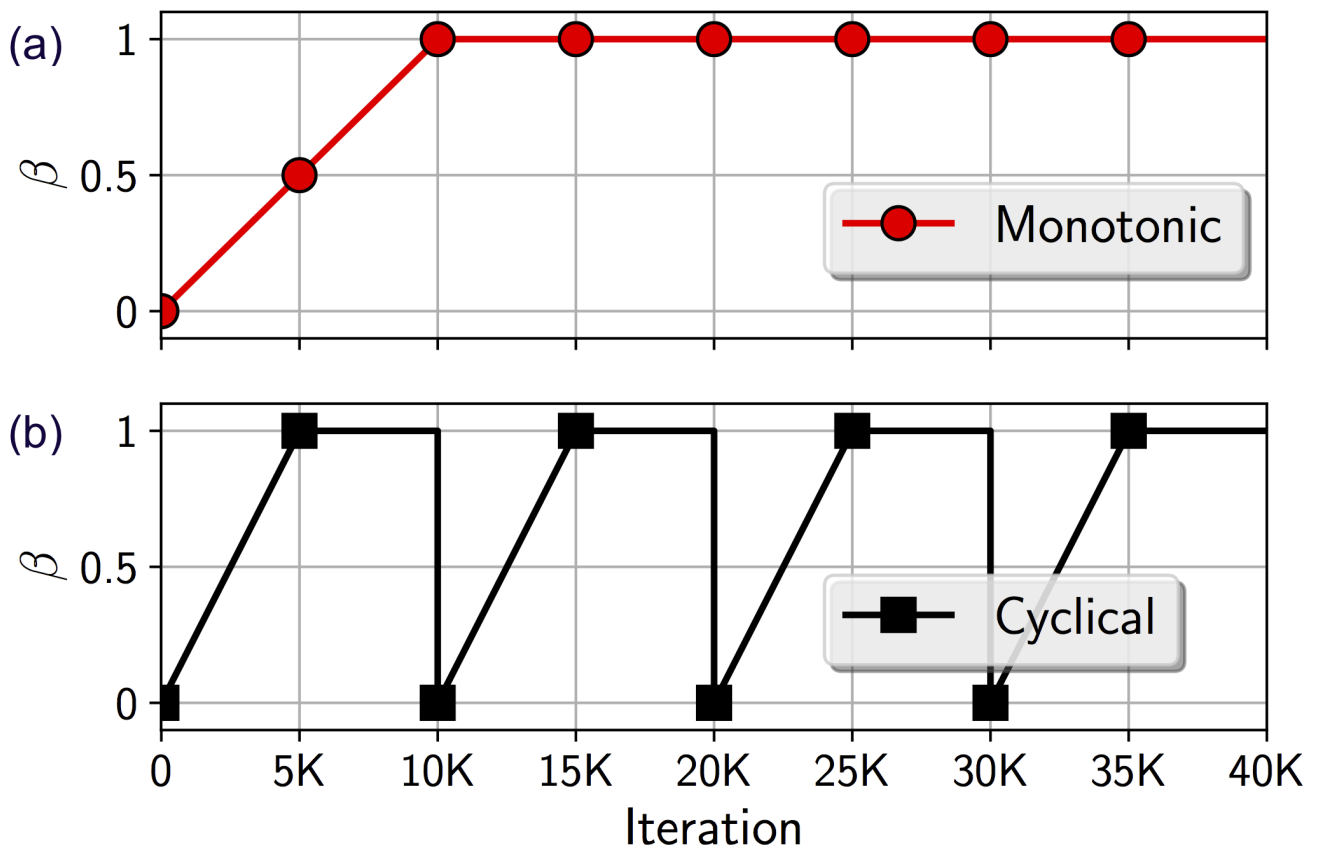
Notation: q is encoder, z is latent.

- For an showed image x , we need to maximize the log probability (because of max likelihood) x generated by AE, $p(x)$.
 - $\log p(x) = \log \int_z p(x, z) dz = \log \int_z p(x, z) \frac{q(z|x)}{q(z|x)} dz$
 - By Jensen's inequality, $\log p(x) \geq \mathbb{E}_{q(z|x)} \log \frac{p(x, z)}{q(z|x)} dz$
 - By Bayes' theorem, $\log p(x) \geq \mathbb{E}_{q(z|x)} \log \frac{p(x|z)p(z)}{q(z|x)}$
 - $= \mathbb{E}_{q(z|x)} \log p(x|z) + \mathbb{E}_{q(z|x)} \log \frac{p(z)}{q(z|x)}$
 - $= \mathbb{E}_{q(z|x)} \log p(x|z) + \int_z q(z|x) \log \frac{p(z)}{q(z|x)}$
 - $= \mathbb{E}_{q(z|x)} \log p(x|z) - D_{KL}[q(z|x)||p(z)]$
 - $= \text{Maxlikelihood} - D_{KL}[q(z|x)||p(z)]$
- Where we assume $p(z)$ is gaussian distribution $N(0, 1)$, and $q(z|x)$ is $N(\mu, \sigma)$
 - Recall that $N(\mu, \sigma) = \frac{1}{\sigma \sqrt{2\pi}} \exp(-\frac{1}{2}(\frac{x-\mu}{\sigma})^2)$
 - $\log N(\mu, \sigma) = \log \frac{1}{\sqrt{2\pi}} - \log \sigma - \frac{1}{2}(\frac{x-\mu}{\sigma})^2$
 - $D_{KL}[N(\mu, \sigma)||N(0, 1)] = \int N(\mu, \sigma) [(\log \frac{1}{\sqrt{2\pi}} - \log \frac{1}{\sqrt{2\pi}}) - (\log \sigma - \log 1) - \frac{1}{2}((\frac{x-\mu}{\sigma})^2 - (\frac{x-0}{1})^2)] dx$
 - $= \mathbb{E}_{N(\mu, \sigma)} [-\log \sigma - \frac{1}{2}((\frac{x-\mu}{\sigma})^2 - x^2)]$
 - $= -\log \sigma - \frac{1}{2\sigma^2} \mathbb{E}_{N(\mu, \sigma)} [(x - \mu)^2] + \mathbb{E}_{N(\mu, \sigma)} [x^2]$
 - $= -\log \sigma - \frac{1}{2} + \mathbb{E}_{N(\mu, \sigma)} [x^2]$
 - $\sigma^2 = \mathbb{E}[x^2] - \mathbb{E}[x]^2 \rightarrow \mathbb{E}[x^2] = \mu^2 + \sigma^2$
 - $= -\log \sigma - \frac{1}{2} - \frac{1}{2}(\sigma^2 + \mu^2)$
 - $= -\frac{1}{2}(1 + \log \sigma^2 - \sigma^2 - \mu^2)$
 - Thus, the regularization term for KL divergence is $= -\frac{1}{2}(1 + \log \sigma^2 - \sigma^2 - \mu^2)$. If we modify the encoder from an image to some μ and σ , and sample the latent variable from $N(\mu, \sigma)$, then we can apply this term.

KLD Loss Scheduler

[Cyclical Annealing Schedule: A Simple Approach to Mitigating KL Vanishing \(NAACL 2019\)](#)

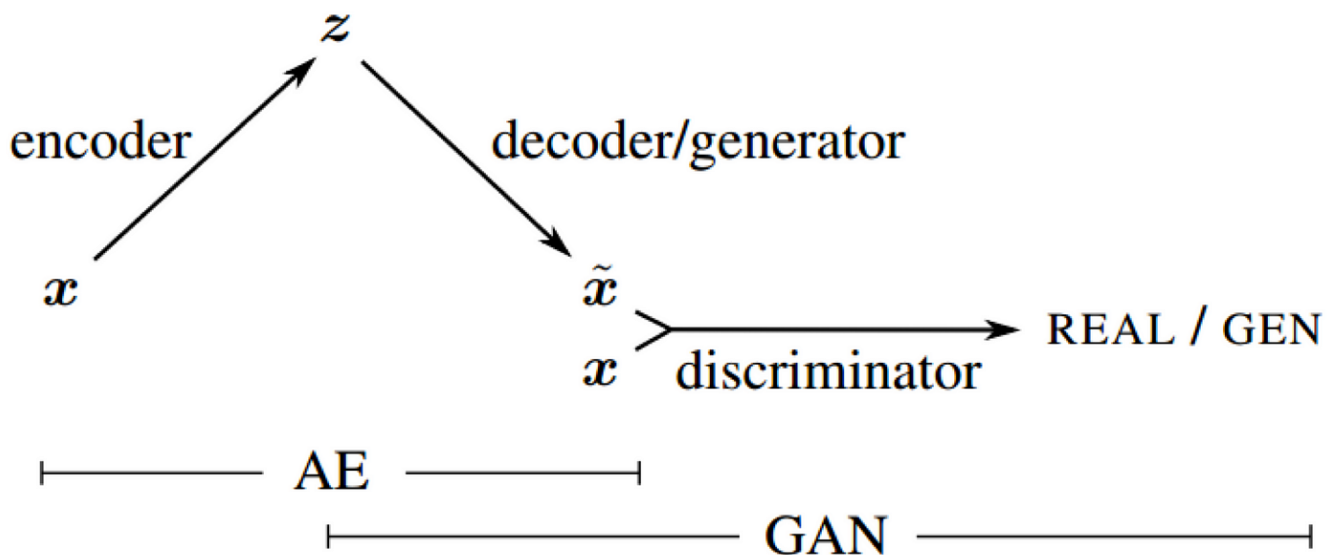
Balancing the weights of VAE_loss and KLD_loss is a challenging task. If we set VAE_loss too high, clear images won't be generated from sampling the latent space because it won't match the decoder's distribution. Conversely, if we set KLD_loss too high, the autoencoder may tend to experience mode collapse and ignore the reconstruction loss term. To mitigate this issue, we employ a technique known as the cyclical annealing scheduler, which periodically adjusts the weight of KLD_loss.



VAE-GAN

[autoencoding beyond pixels using a learned similarity metric \(ICML 2016\)](#)

Because the image generated from the vanilla VAE is too blurry, we can use GAN to mitigate this phenomenon. To prevent mode collapse from occurring too early, we only train the GAN part from a specific iteration.



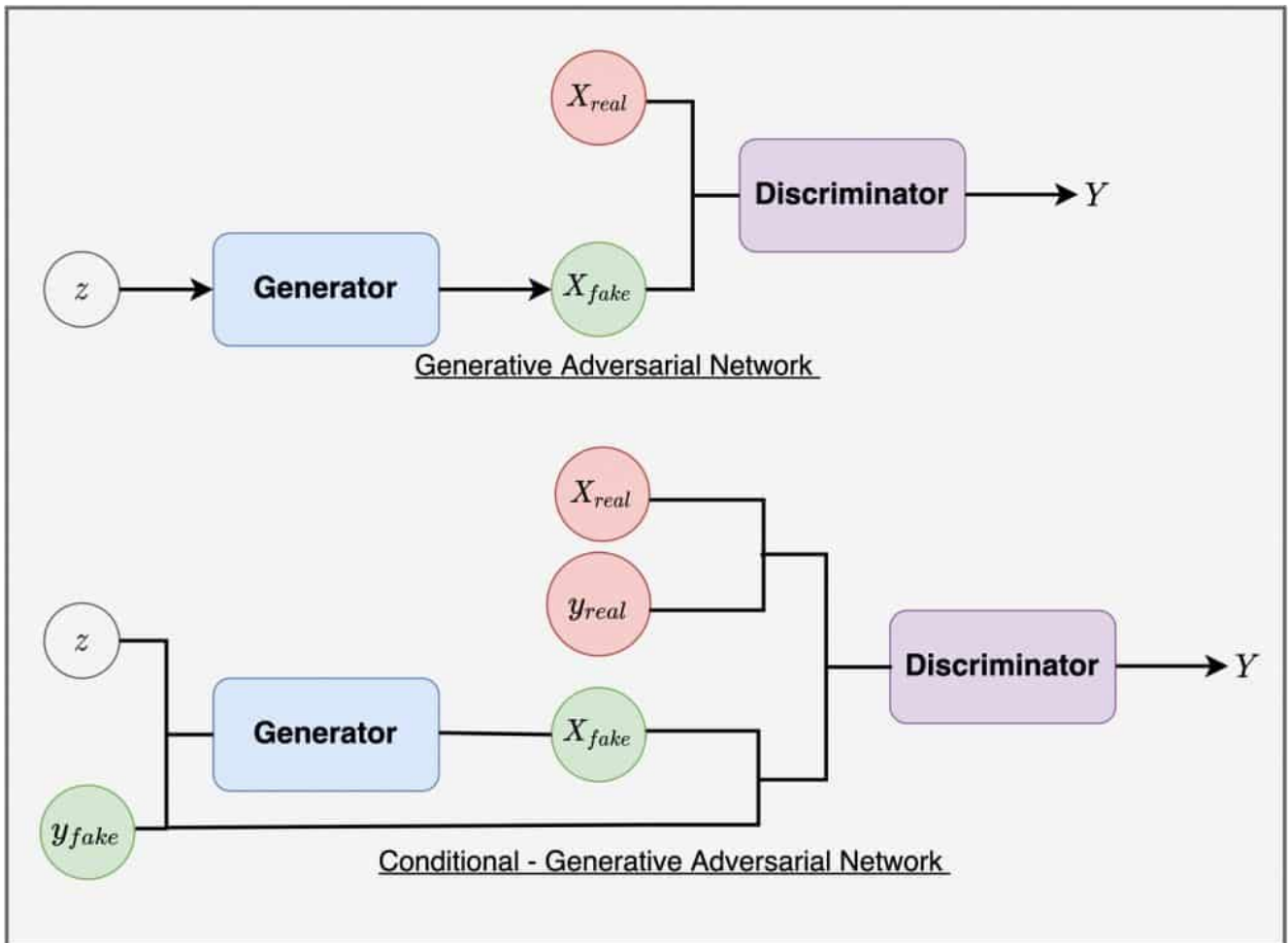
Conditional GAN & PatchGAN

[Image-to-Image Translation with Conditional Adversarial Networks \(CVPR 2017\)](#)

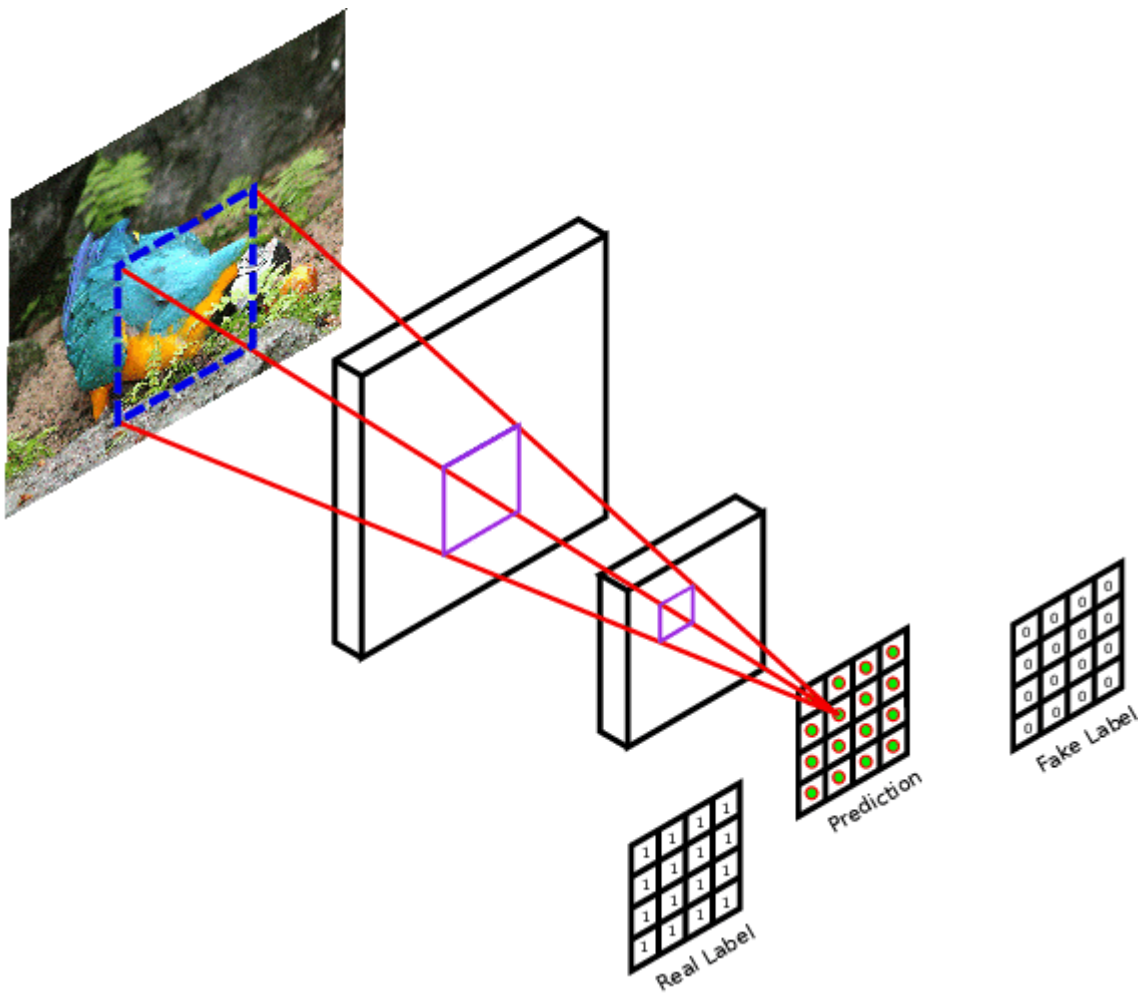
We have incorporated two concepts from this paper: conditional GAN and Patch GAN.

Our task is to reconstruct brains, and we have observed that each brain generates 32 images with different z-index values. This additional information enables us to implement conditional GAN, leveraging this extra condition to enhance the quality of randomly generated images.

Instead of using regular embedding, we have opted for sinusoidal embedding in the z -index and have added these positional encodings after the feature map passes through the convolutional layers in both the autoencoder and discriminator.



The original discriminator in GANs aims to predict whether the entire image is real or fake, whereas PatchGAN is designed to predict the reality of individual patches or windows within the image. I believe that this approach provides better guidance to the generator for capturing and generating detailed features in the image.



VQVAE

[Neural Discrete Representation Learning \(NIPS 2017\)](#)

Because the latent space in VAE is continuous, it is challenging to sample from it, even with the addition of a KLD regularization term during VAE training to constrain the latent space to be similar to the Normal Distribution $N(0, 1)$. This paper introduces a method to discretize the latent space, making it possible to sample from a discrete space. Furthermore, since the latent space is discrete and reduced in size, we can employ an powerful auto-regressive model to generate the latent values. For instance, the author of this paper uses PixelCNN as the decoder.

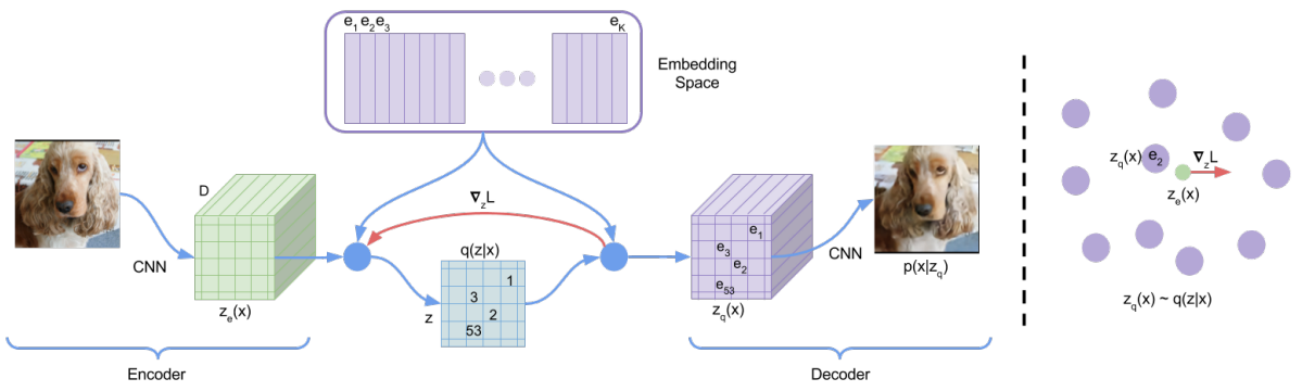


Figure 1: Left: A figure describing the VQ-VAE. Right: Visualisation of the embedding space. The output of the encoder $z(x)$ is mapped to the nearest point e_2 . The gradient $\nabla_z L$ (in red) will push the encoder to change its output, which could alter the configuration in the next forward pass.

The next problem is how to discretize the latent space while still allowing for network updates through gradients. The discretization layer is not differentiable. To address this challenge, VQVAE employs three losses:

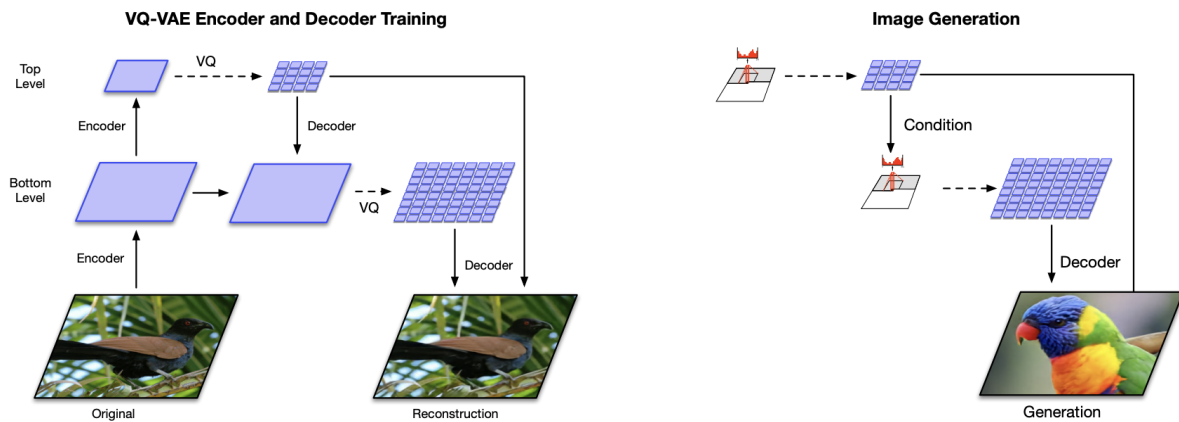
1. Reconstruction Loss: This is equivalent to VAE.
2. Loss to make the embedding layer (or discrete latent) similar to the latent vector generated by the encoder.

3. The same as 2, but in reversed.

- Note that we don't want the encoder to generate the discrete latent directly, so this term is multiplied by β , with the β value in the paper set to 0.25.

After applying these three losses, your VQVAE will be successfully trained.

Furthermore, there's VQVAE2, which discretizes the latent space into two layers. However, due to time constraints, I don't have the opportunity to explore this approach.



(a) Overview of the architecture of our hierarchical VQ-VAE. The encoders and decoders consist of deep neural networks. The input to the model is a 256×256 image that is compressed to quantized latent maps of size 64×64 and 32×32 for the *bottom* and *top* levels, respectively. The decoder reconstructs the image from the two latent maps.

(b) Multi-stage image generation. The top-level PixelCNN prior is conditioned on the class label, the bottom level PixelCNN is conditioned on the class label as well as the first level code. Thanks to the feed-forward decoder, the mapping between latents to pixels is fast. (The example image with a parrot is generated with this model).

VQGAN

Overview

- Construct an Unet-like autoencoder, and it used some module like Swish, Self-attention and resnetblock
- Use reparameterization in latent space to calculate KL divergence.
- Use GAN (or discriminator) with LPIPS score to optimize decoder.
 - LPIPS score is adopted in stable diffusion. But we have no pre-trained model to use.

Latent Diffusion Models

Conditioning Mechanisms

Time Embedding