Goal

- Goes through basic concepts in Computer Vision
  - Convolution
  - Pooling
  - Upsampling
  - Interpolation
  - Transposed Convolution

- Goes through details of VAE
  - Architecture
  - Math
  - Training & Sampling
  - More

Note: Diffusion is similar with VAE, so consider this as prequel of Diffusion!

Autoencoder Recap

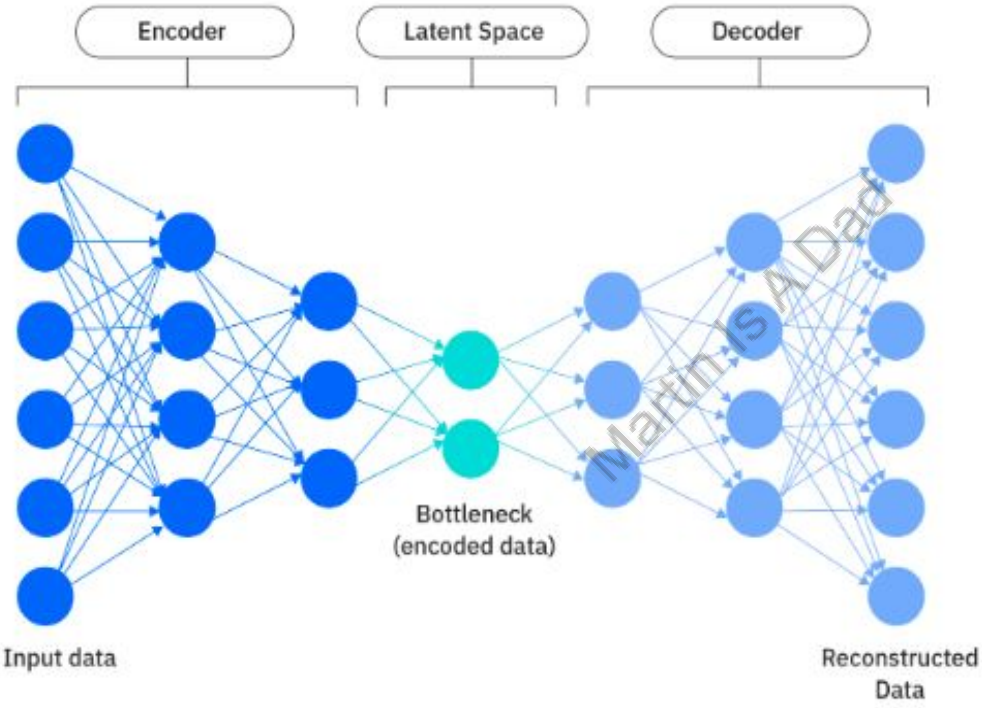
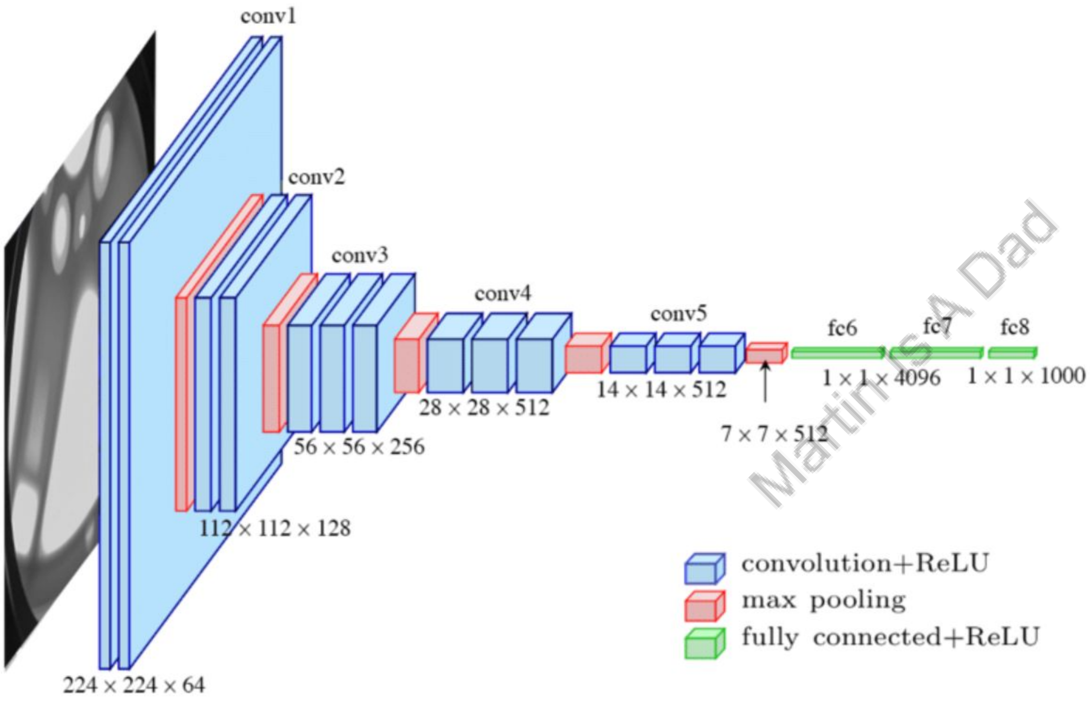


Image from IBM VAE blog

- Encoder will compress the original image to a lower dimension latent space. The latent dimension is supposed to capture the essential features of original image.
- Decoder will reconstruct the original image from the lower dimension latent space.

How Is It Done Exactly?

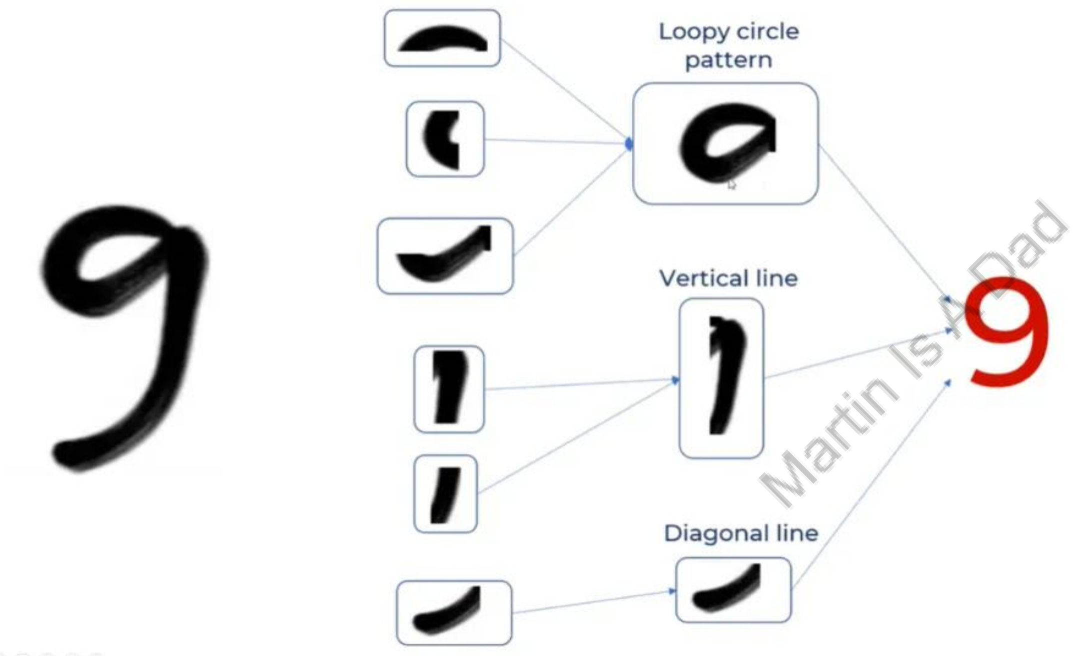
Encoder Architecture Details



Typical CNN example from vitalflux.com

- Convolution Layer: Extract features
- ReLU:
  - Allow model to learn non-linear complex relationship
  - Help vanishing gradients
  - Efficient computation
- Pooling:
  - Downsample feature maps
  - Reduce spatial dimensions and computational complexity while retaining important information,
  - Make the model more robust to variations in feature position
- Fully connected: Flatten input and map it to probability distribution we need

Convolution Layer Intuition



- For single image, we first identify simple micro patterns like straight lines, small curves
- Then based on these patterns' relationship with each other, identify shapes like circle, oval, squares etc
- Then based on these shapes' relationship with each other, identify more complex shapes and patterns, like legs, wheels etc
- The shape identifiers should ideally be reusable

Image from [Medium (sharathmanjunath)]

Convolution Layer Intuition

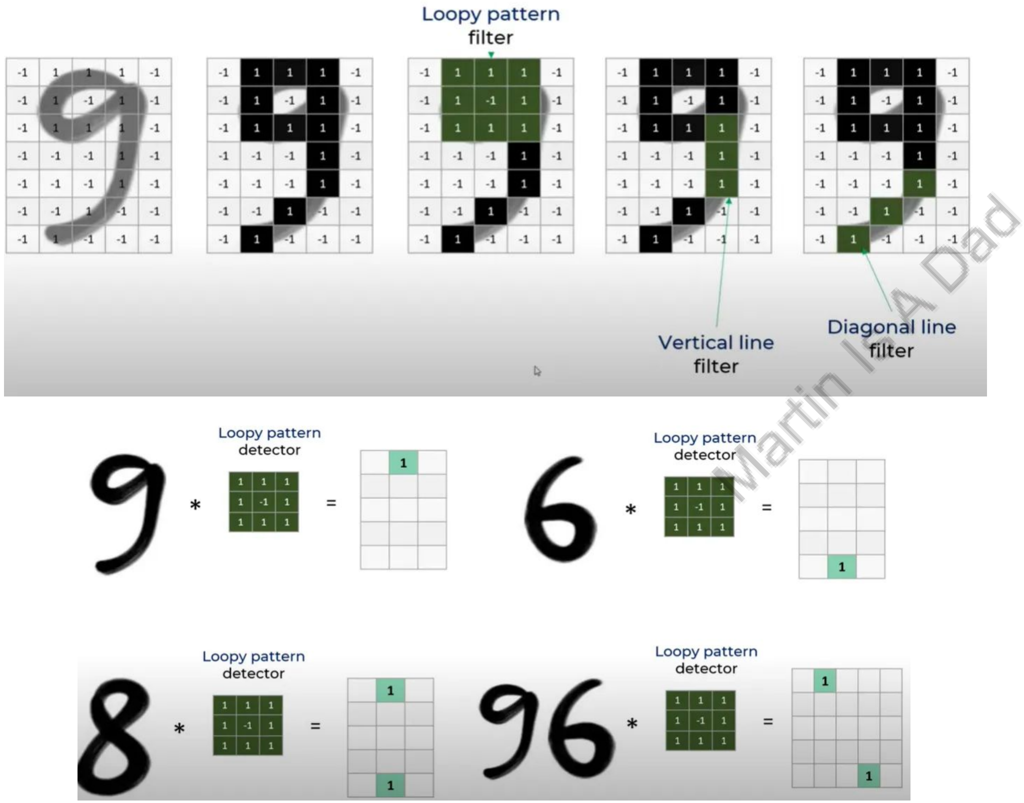


Image from [Medium (sharathmanjunath)]

- Use shape identifiers to identify patterns
- The shape identifiers are called filters or kernels

Convolution Operation with Kernel

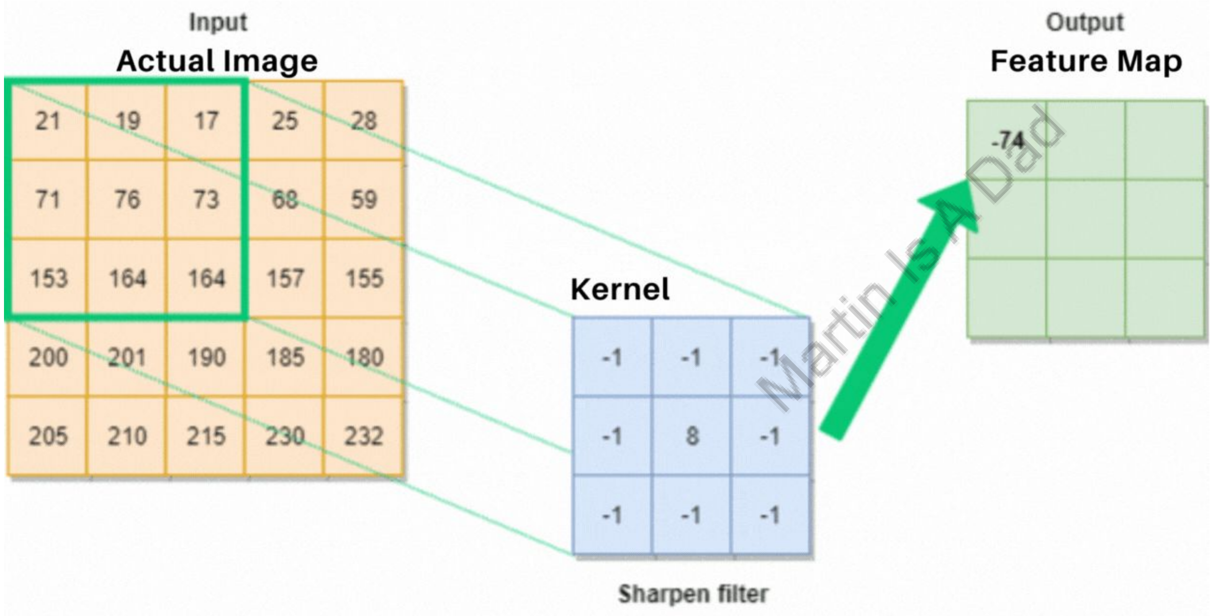


Image from [Medium (patale_akhil)]

- Values in the Kernel are all trainable parameters
- Convolution is to use the Kernel to scan through the image and compute matrix sum product
- Kernel values (learnable parameters of CNN) stay unchanged throughout the image, so the number of model's parameter does not depend on image size

Neural Network Activation Function

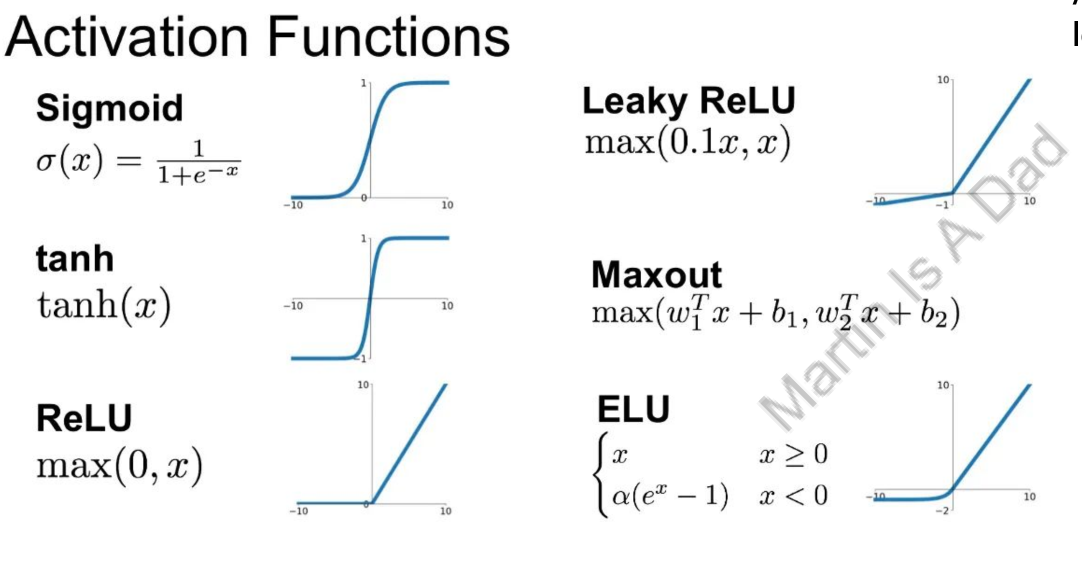
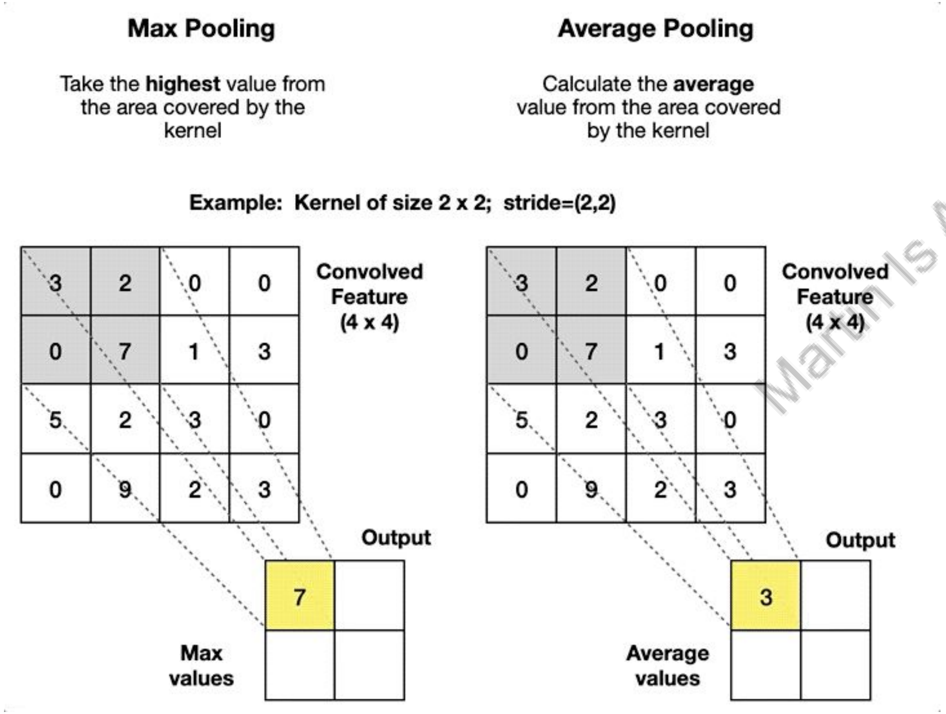


Image from [Medium (shrutijadon)]

Activation functions allow neural networks to learn nonlinear relationships:

- Sigmoid and tanh suffers from vanishing gradient. When value is too big or too small, the gradient will become 0.
- ReLU improves vanishing gradient since the gradient is 1 for all positive values. The computation is also more efficient since gradient constant (1 for positive and 0 for negative). ReLU can suffer if value is constantly negative causing 0 gradient, model will stop learning.
- Leaky ReLU improve on top of ReLU, where negative values still have non-zero positive gradient.

Pooling Layer



Pooling layer's purpose:

- Downsample feature maps
- Reduce spatial dimensions and computational complexity while retaining important information
- Make the model more robust to variations in feature position

Type of Pooling:

- Max Pooling (keep most apparent feature)
- Average Pooling (get average in a region)

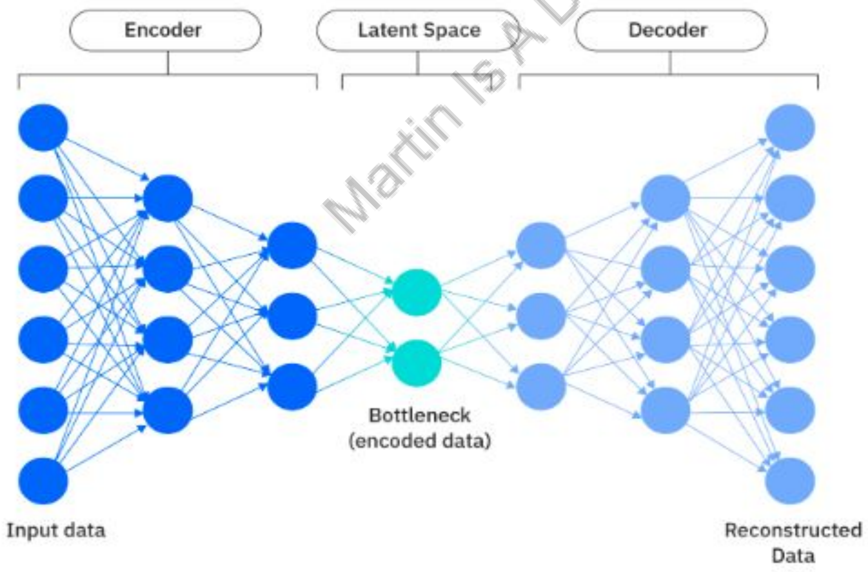
How it works:

- 2x2 pooling, with stride of 2, reduces both length and width of feature-map by 50%.
- No new parameters needed, just get the maximum or average
- Often added after convolution layer

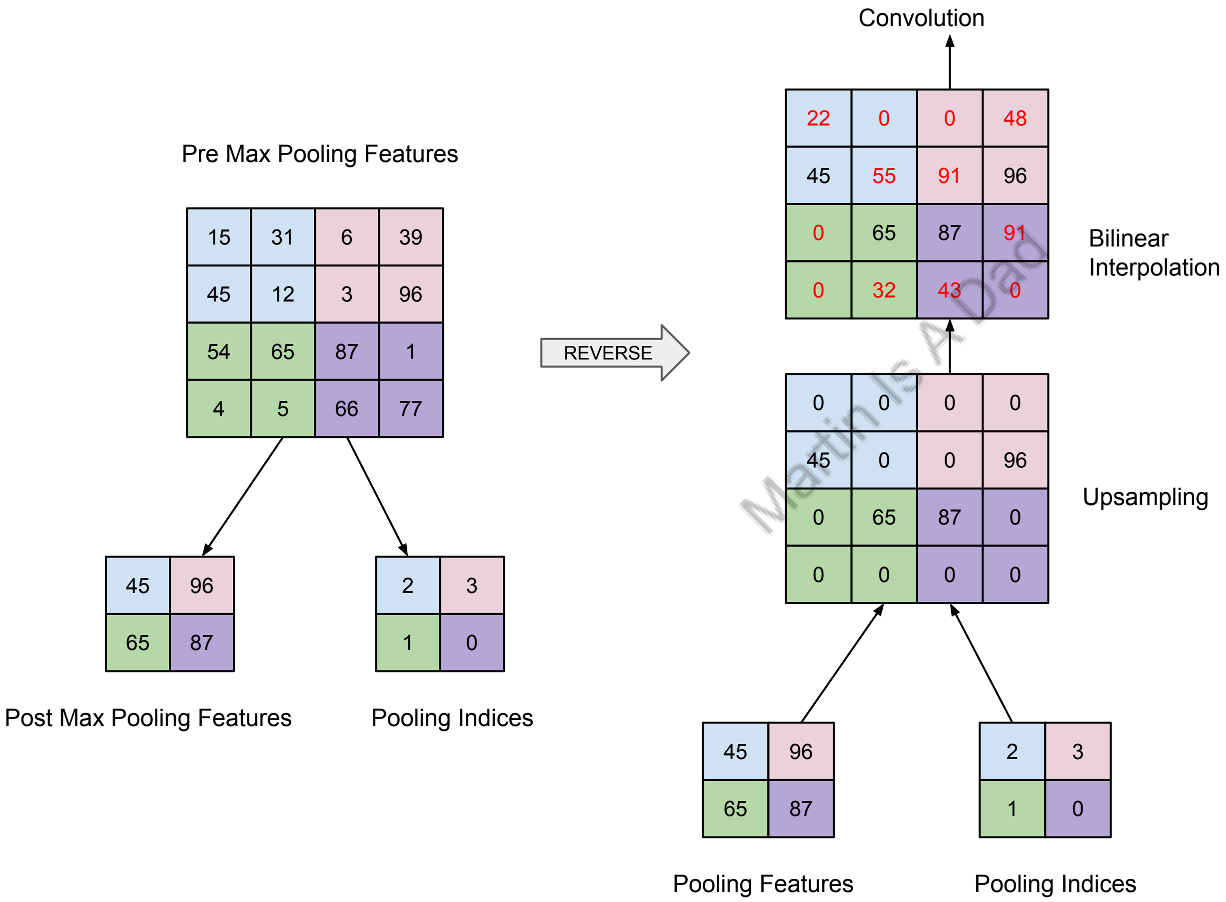
How Does Decoder Work?

- In Autoencoder, decoder reconstruct the original image from the lower dimension latent space.

- Basically reverse the process of what Encoder does.
    - Encoder: Convolution + Downsample
    - Decoder: Convolution + Upsample



Decoder Increase Image Size



- We can remember the pooling indices and use it to do upsampling
- After upsampling and interpolation, usually followed by convolutions to learn how to rebuild simple and complex patterns with latent space features
- Interpolation is used to estimate or predict values between known data points. Common ways include bilinear (closest 4 neighbors average), polynomial etc

Decoder Increase Image Size

1	3
2	4

Pooling Features

1	1	1
1	-1	1
1	1	1

Kernel (with learnable parameters)

1	1	1	
1	-1	1	
1	1	1	

Step 1

1	1+3	1+3	3
1	-1+3	1-3	3
1	1+3	1+3	3

Step 2

1	1+3	1+3	3
1+2	-1+3+2	1-3+2	3
1+2	1+3-2	1+3+2	3
2	2	2	

Step 3

1	1+3	1+3	3
1+2	-1+3+2+4	1-3+2+4	3+4
1+2	1+3-2+4	1+3+2-4	3+4
2	2+4	2+4	4

Step 4

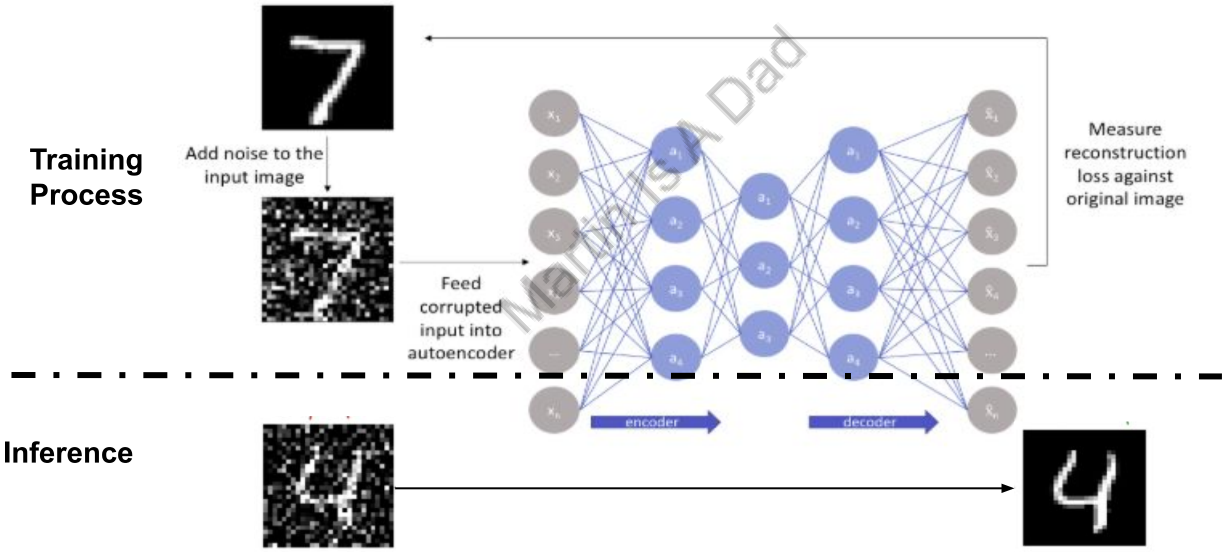
1	4	4	3
3	8	4	7
3	6	2	7
2	6	6	4

Output

- We can also do transposed convolution
- With the features map after encoder's pooling layer, and a Kernel (with learnable parameters) aiming to reconstruct image with latent space features, we can bring back the image to its original dimension

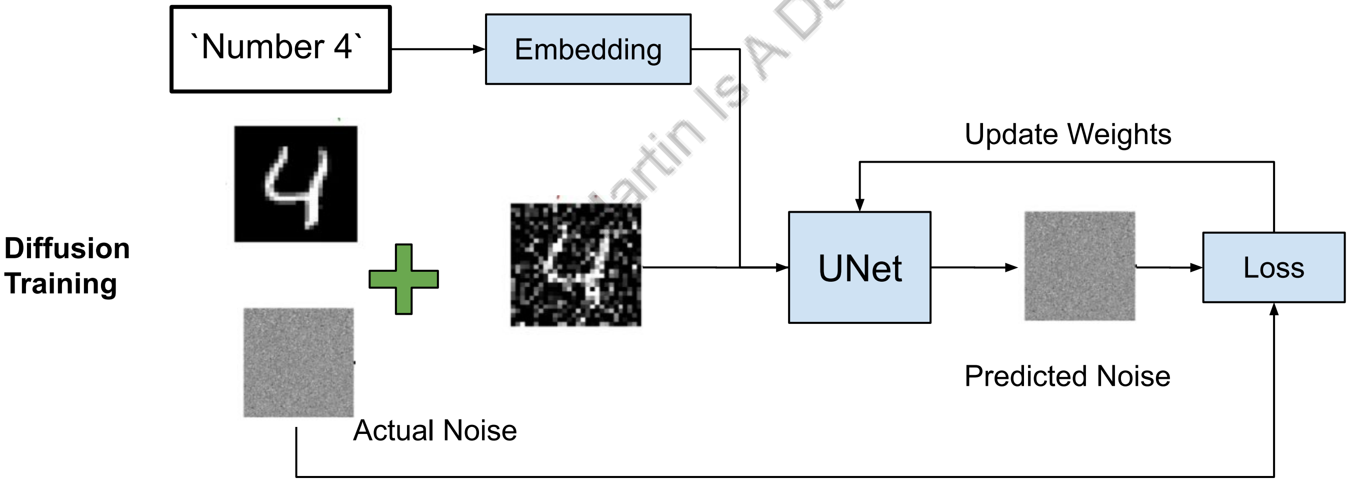
Autoencoder Recap

- During training, autoencoder's loss function is comparing pixel wise difference between output image and original image. That's also where the `auto` in the name comes from.
- Autoencoder is great in **denoising** image



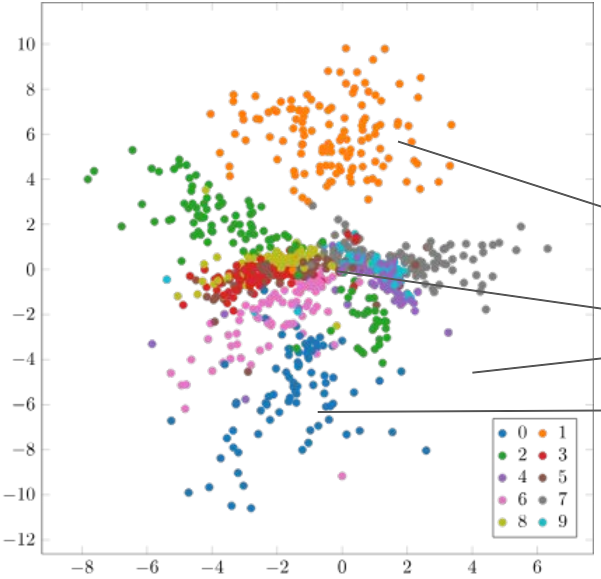
Diffusion Recap

- Diffusion's mechanism is also based on using model to predict noise for noisy image, so we have the capability to iteratively remove noises and generate good images.
- Many overlap in architecture, underlying math, training & sampling etc. Stay tuned!



From Autoencoders to Variational Autoencoders (VAE)

- Goal of autoencoder is to reconstruct the original image. However, for a generative model, what we want is **`Given observed samples *x* from a distribution of interest, learn to model the true data distribution *p(x)*`**
- In plain words, we want to generate *new* samples that *resemble* the original input. That's why AE is not good for Generative task. Other reasons are:
    - Autoencoder are fixed, deterministic mapping and discrete on the latent space



Latent space of MNIST data (number classification)

If sample a random point from this 2-D space and feed it to the decoder, will it generate a new image that is "similar" to one of the 0-9 digit images in the training set?

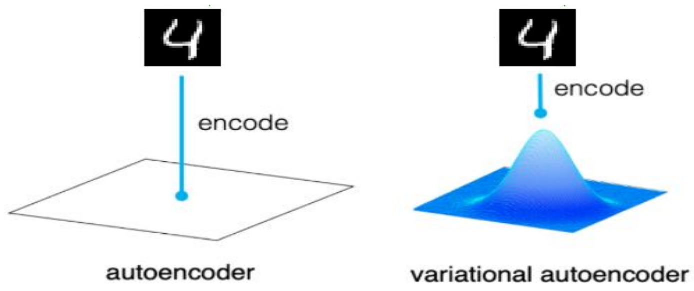
Probably will generate a good `1`

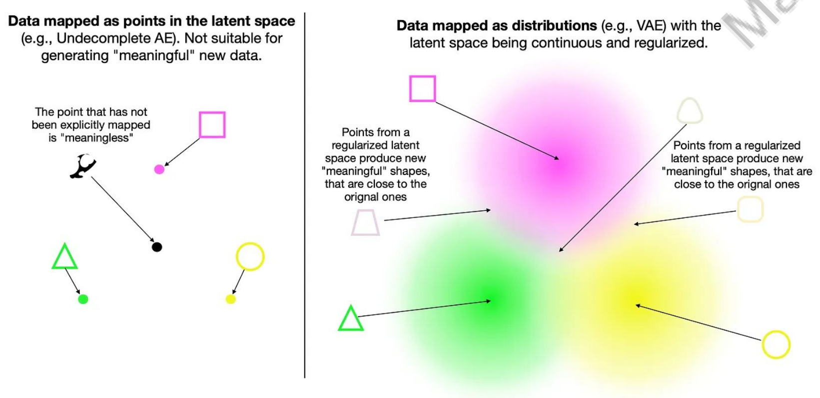
???

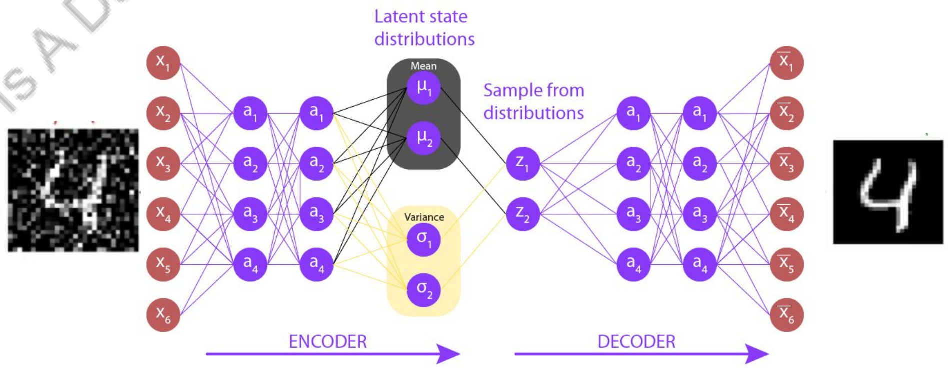
Probably will generate a good `0`

Variational Autoencoders (VAE)

- VAES are **probabilistic** models. VAEs encode latent variables of training data not as a fixed discrete value z, but as a continuous range of possibilities expressed as a probability distribution p(z), represented with mean vector (μ) and a std-deviation / variance vector (σ).

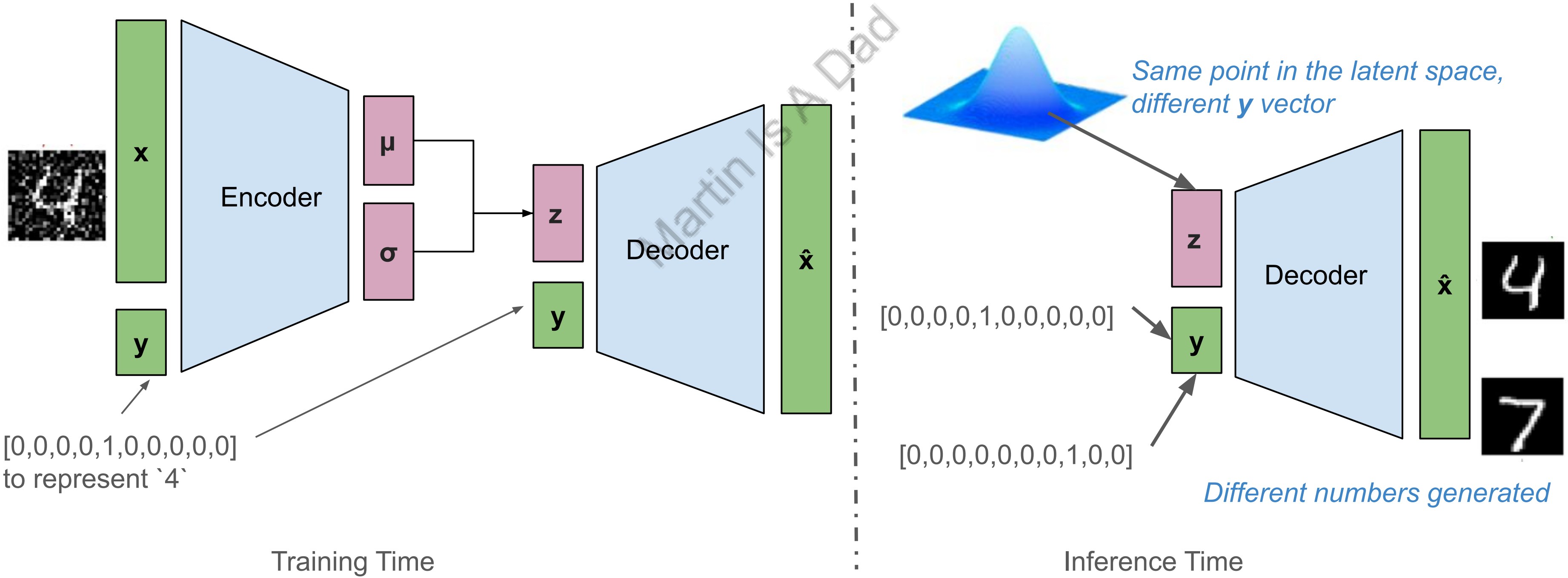






Conditional VAE

- With VAE, we can sample a random point in the latent space and generate new images similar to training set. However, we can't generate an new on demand (say generate a specific number with the MNIST example)
- With Conditional VAE, we will be able to achieve this
    - CVAE has an extra input to both the encoder and the decoder, usually an one-hot vector.



Underlying Math

- Assumptions
  - We have observed data $X = \{x_i\}_{i=1}^N$
  - We assume the data is generated from a latent variable $Z = \{z_i\}_{i=1}^N$
  - θ represents parameters of the model
- Goal
  - Maximize the log likelihood of observed data $\log p_\theta(X)$
    - Same as maximize likelihood directly (log function is monotonic), log is used for convenience to make math easier
- Given $p_\theta(X) = \int p_\theta(X|Z)p(Z)dZ$, can we calculate this directly?
  - This is computationally intractable
  - The prior distribution $p(Z)$ is often chosen to be a simple distribution, such as a standard Gaussian
  - However, $p_\theta(X|Z)$ is typically a complex, non-linear function defined modeled by neural network with parameters θ.
- So what to do now? Use ELBO (Evidence Lower Bound)

Underlying Math

- We introduce the approximate posterior $q_\phi(Z|X)$, then use **Jensen's inequality** to derive a lower bound.
  - $\boldsymbol{\phi}$ represents the approximation function's parameters

$$\log p_\theta(X) = \log \int p_\theta(X, Z) dZ = \log \int \frac{p_\theta(X, Z)}{q_\phi(Z|X)} q_\phi(Z|X) dZ$$

$$\log \int \frac{p_\theta(X, Z)}{q_\phi(Z|X)} q_\phi(Z|X) dZ \geq \int q_\phi(Z|X) \log \frac{p_\theta(X, Z)}{q_\phi(Z|X)} dZ$$

Lower bound of original log likelihood

- Expand the right hand side of equation

$$\int q_\phi(Z|X) \log \frac{p_\theta(X, Z)}{q_\phi(Z|X)} dZ = \int q_\phi(Z|X) \log p_\theta(X, Z) dZ - \int q_\phi(Z|X) \log q_\phi(Z|X) dZ$$

- Rewriting integrals in terms of expectations

$$Eq\phi(Z|X)[\log p_\theta(X, Z)] - Eq\phi(Z|X)[\log q_\phi(Z|X)]$$

$$Eq\phi(Z|X)[\log p_\theta(X|Z) p(Z)] - Eq\phi(Z|X)[\log q_\phi(Z|X)]$$

$$Eq\phi(Z|X)[\log p_\theta(X|Z) + \log p(Z)] - Eq\phi(Z|X)[\log q_\phi(Z|X)]$$

Underlying Math

- Rearrange orders

$$Eq\phi(Z|X)[\log p_\theta(X|Z)] + Eq\phi(Z|X)[\log p(Z)] - Eq\phi(Z|X)[\log q_\phi(Z|X)]$$

- Recall **Kullback-Leibler Divergence** (KL Divergence) between p(x) and q(x) is defined as:

$$D_{KL}(q(x)||p(x)) = \int q(x) \log \frac{q(x)}{p(x)} dx = Eq(x)[\log q(x)] - Eq(x)[\log p(x)]$$

- Rewrite lower bound expression

$$ELBO = Eq\phi(Z|X)[\log p_\theta(X|Z)] - D_{KL}(q_\phi(Z|X)||p(Z))$$

- Intuition of ELBO
  - **Reconstruction Loss**: $Eq\phi(Z|X)[\log p_\theta(X|Z)]$ encourages the decoder $p_\theta(X|Z)$ to reconstruct the input X well from the latent representation Z.
  - **KL Divergence Loss**: $D_{KL}(q_\phi(Z|X)||p(Z))$ encourages the approximate posterior $q_\phi(Z|X)$ to be close to the prior *p(Z)*, typically a standard Gaussian distribution.

- Note that we only care about decoder parameters of Θ, and encoder parameters of $\boldsymbol{\phi}$

Underlying Math

- **Reconstruction Loss** is typically computed by Mean Squared Error (MSE) between the original and reconstructed images:
    - f_θ is the decoder function
    - g_ϕ is the encoder function

$$L_{\mathrm{MSE}}(\theta, \phi) = \frac{1}{N}\sum_{i=1}^{N}(x_i - f_\theta(g_\phi(x_i)))^2$$

- Practically, we usually assume the prior distribution *p(Z)* is standard normal distribution. Thus the **KL Divergence Loss** is:
    - $G(Z_\mu, Z_\sigma)$ is the Gaussian distribution of latent space defined by the mean and standard deviation.

$$L_{\mathrm{KL}}[G(Z_\mu, Z_\sigma)|\mathcal{N}(0,1)] = -0.5 * \sum_{i=1}^{N} 1 + \log(Z_{\sigma_i^2}) - Z_{\mu_i}^2 - Z_{\sigma_i}^2$$

- Then we have the final loss function formula:

$$\mathcal{L}_{\mathrm{VAE}} = \mathcal{L}_{\mathrm{recon}} + \mathcal{L}_{\mathrm{KL}}$$

- We can now use gradient descent to maximize ELBO, with one last obstacle :<

Underlying Math

- Recap
  - We plan to use gradient descent to maximize ELBO: $\mathcal{L}_{\text{VAE}} = \mathcal{L}_{\text{recon}} + \mathcal{L}_{\text{KL}}$
  - The parameters we are updating is decoder parameters of Θ, and encoder parameters of $\boldsymbol{\phi}$

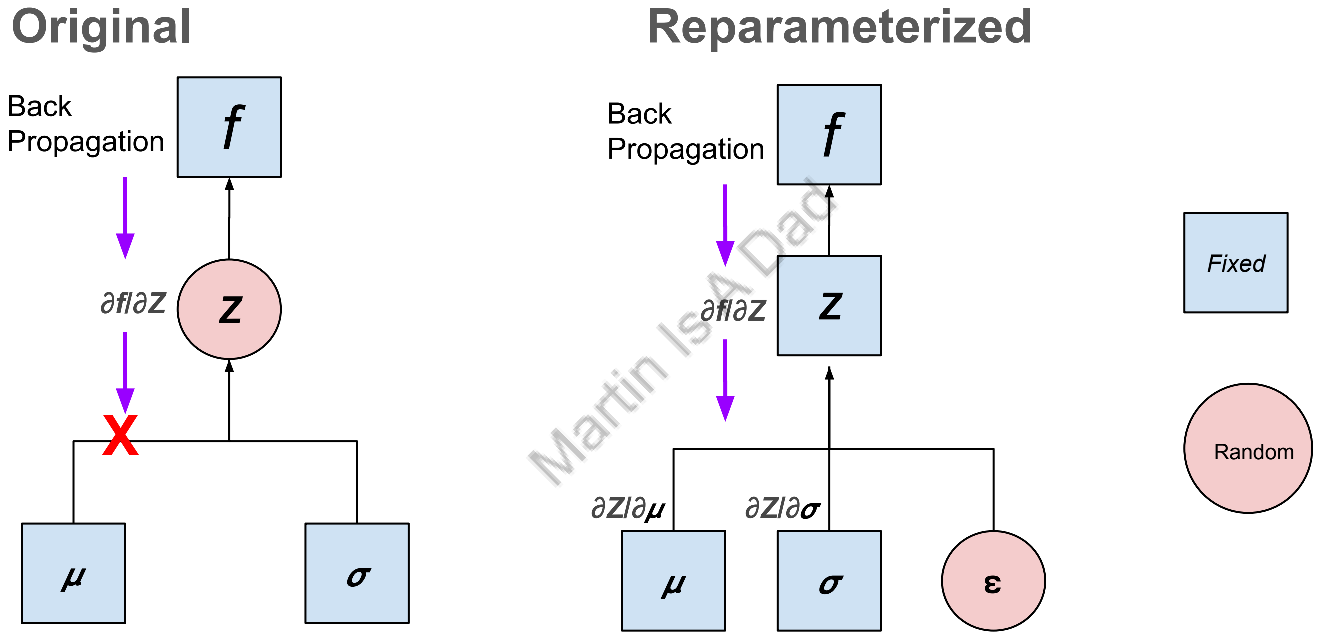
- **Problem:**
  - Although **ELBO** is a function of encoder parameter $\boldsymbol{\phi}$, it's not differentiable of $\boldsymbol{\phi}$ since Z_i is randomly picked from the latent space represented by $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$

- **Reparameterization Trick** to the rescue
  - The essence is change how sampling is executed. Instead of random sampling directly from $q_\phi(Z|X)$,introduce a random auxiliary variable **ε** from a distribution that is not parameterized by $\boldsymbol{\phi}$, usually from standard normal distribution, then pass it through $g_{\boldsymbol{\phi}}(X, \boldsymbol{\varepsilon})$ to sample $q_\phi(Z|X)$
    - Since **ε** is fixed and not dependent on $\boldsymbol{\phi}$, now **ELBO** differentiable against $\boldsymbol{\phi}$
  - For example $g_{\boldsymbol{\phi}}(X, \boldsymbol{\varepsilon}) = \boldsymbol{\mu} + \boldsymbol{\varepsilon}\,\boldsymbol{\sigma}$

```
def reparameterize(mu, logvar):
  std = torch.exp(0.5 * logvar)
  eps = torch.randn_like(std)
  return mu + eps * std

# Encoder network outputs mu and logvar
mu = encoder(input)[0]
logvar = encoder(input)[1]

# Reparameterization trick
z = reparameterize(mu, logvar)

# Decoder network takes z as input
reconstructed_input = decoder(z)

# Loss calculation and backpropagation
```

Reparameterization Pseudocode

Underlying Math



Now we are able to do gradient descent to find parameters that can maximize ELBO :D

Mission Accomplished!