Image Generation

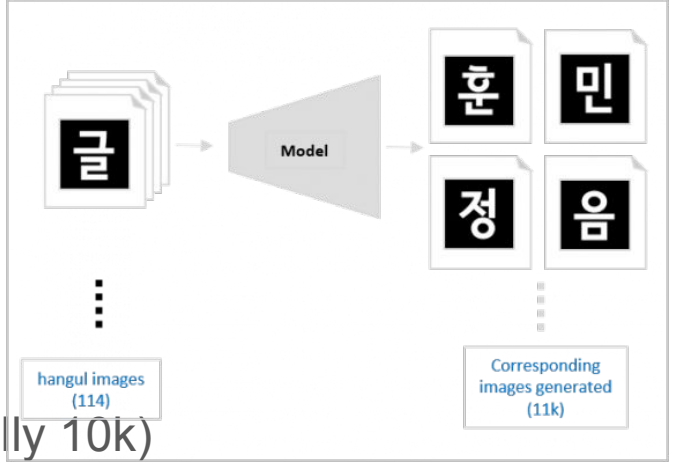
Data Science Club
Generative Models Workshop

What is a Image Generation model?

A model that inputs any type of data and returns image data.

- random image
- text to image
- image to image

My work



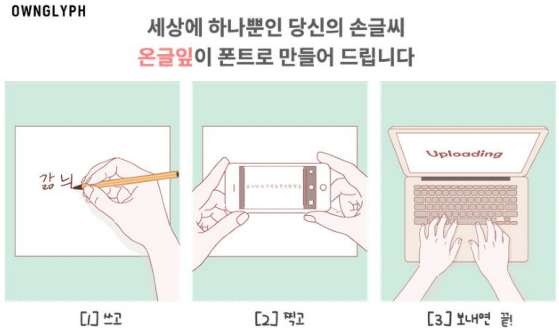
https://www.ownglyph.com/trial/397a1b8a-85a6-41ea-8dea-4352a6a70370/

https://www.instagram.com/p/CZR5vM-P4iAX61CyA_rGTHhyOJRBr0VvAqtUmw0/

Font generation is expensive in Asian countries (typically 10k)

We generated fonts with like 30 sample characters as input -> cost 30 bucks

How would you do this?



Why image generation is trickier than text (in my humble opinion)

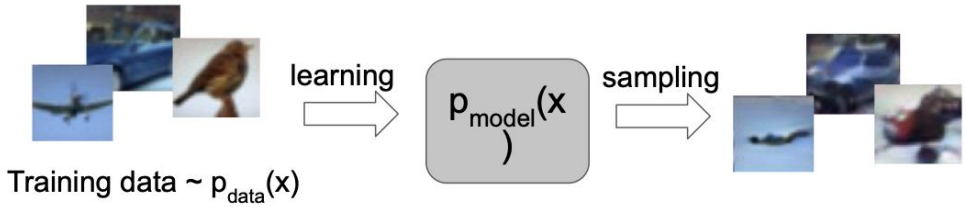
Text is discrete -> text generation is basically classification (classifying the class of the next (sub)word in the sentence with like 10,000 classes)

Image is continuous -> our task is not classifying the next word but generating a 256 ^ (h x w) vector. Stuff like pixel cnn does not work as well as llms

Random image models

Given training data, generate new samples from same distribution

Usually condition on randomly noise sampled from gaussian



Objectives:

1. Learn $p_{model}(x)$ that approximates $p_{data}(x)$
2. **Sampling new x from $p_{model}(x)$**

Taxonomy of Generative Models

Today: discuss 3 most popular types of generative models today

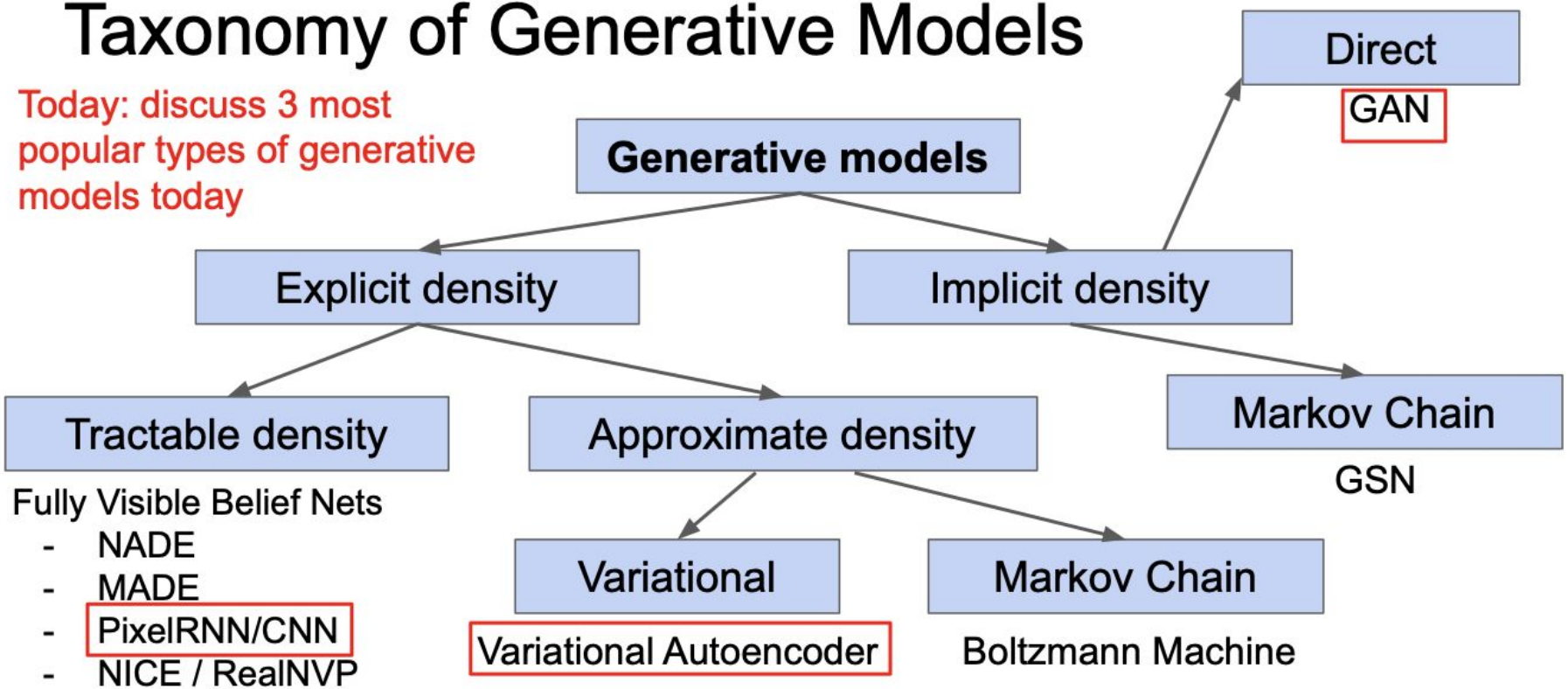
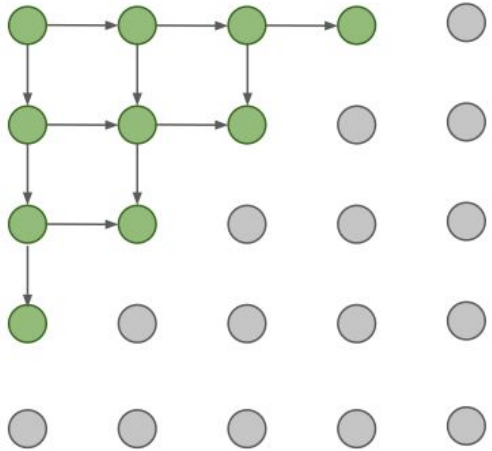


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

PixelRNN *[van der Oord et al. 2016]*

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)



Drawback: sequential generation is slow in both training and inference!

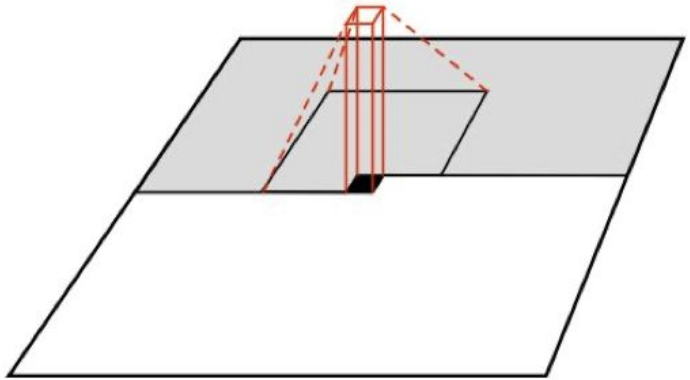
PixelCNN *[van der Oord et al. 2016]*

Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region (masked convolution)

Training is faster than PixelRNN
(can parallelize convolutions since context region values known from training images)

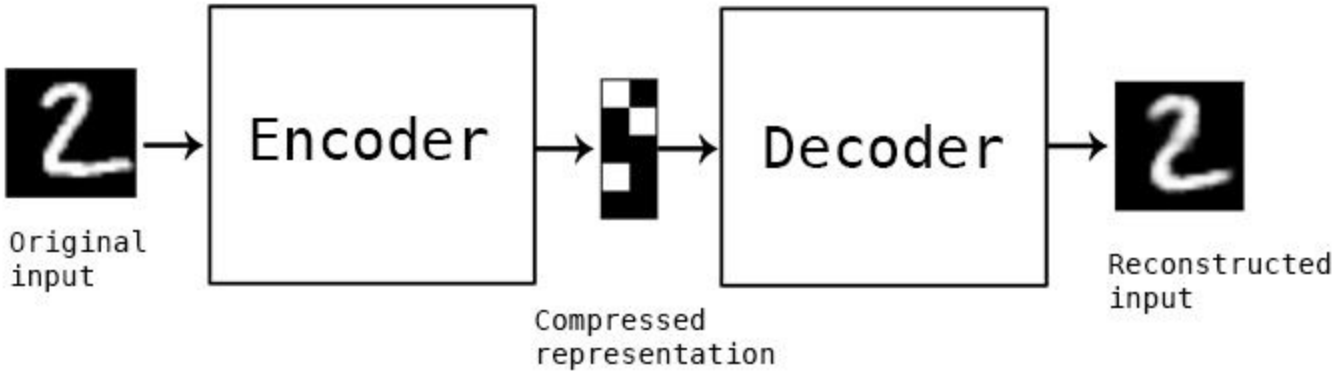
Generation is still slow:
For a 32x32 image, we need to do forward passes of the network 1024 times for a single image



VAE

Variational Auto Encoders

Autoencoders?



Autoencoders are cool, can we use it to generate?

Can we use autoencoders to generate images?

Just sample random latent vector and pass it to decoder?

Unfortunately, the degree of freedom of the viable latent vectors are too high
-> random sampling a vector will result in a weird image

VAE comes to the rescue!

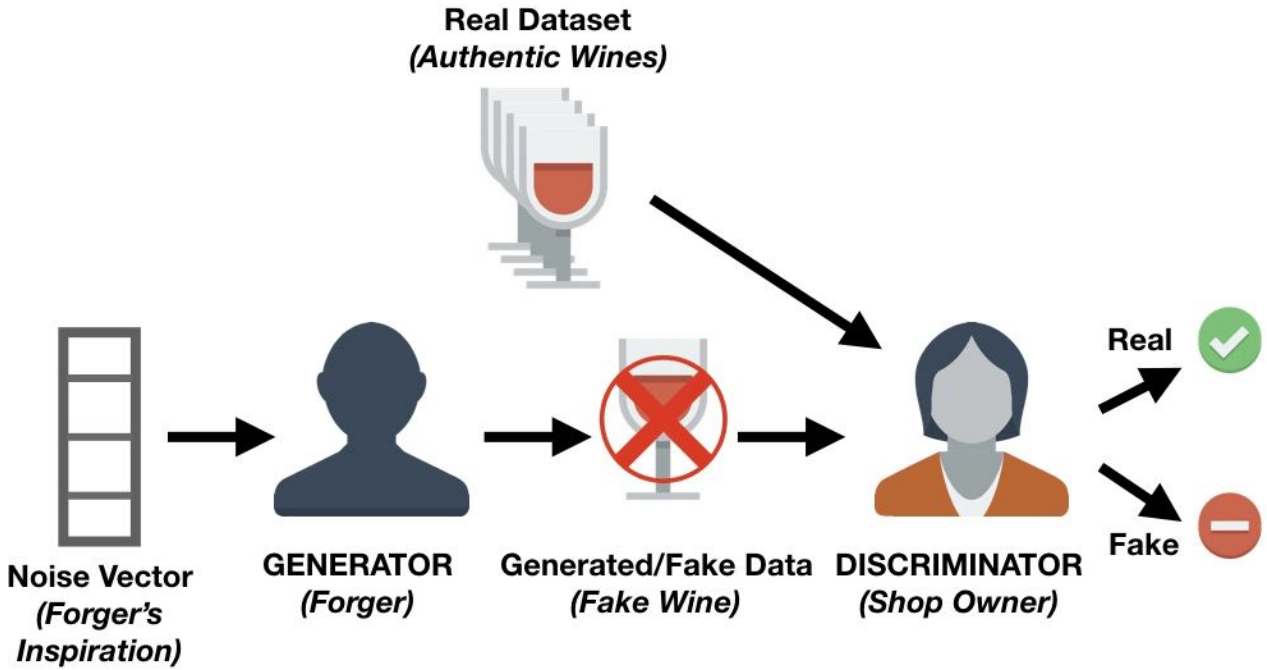
In a nutshell, a VAE is an autoencoder whose encodings distribution is regularised during the training in order to ensure that its latent space has good properties allowing us to generate some new data.

```
generation_loss = mean(square(generated_image - real_image))
latent_loss = KL-Divergence(latent_variable, unit_gaussian)
loss = generation_loss + latent_loss
```

If you want to delve in the math : https://www.cs.cmu.edu/~bhiksha/courses/deeplearning/Spring.2017/slides/lec12.vae.pdf

GAN

Generative Adversarial Network



GANs are hard to train

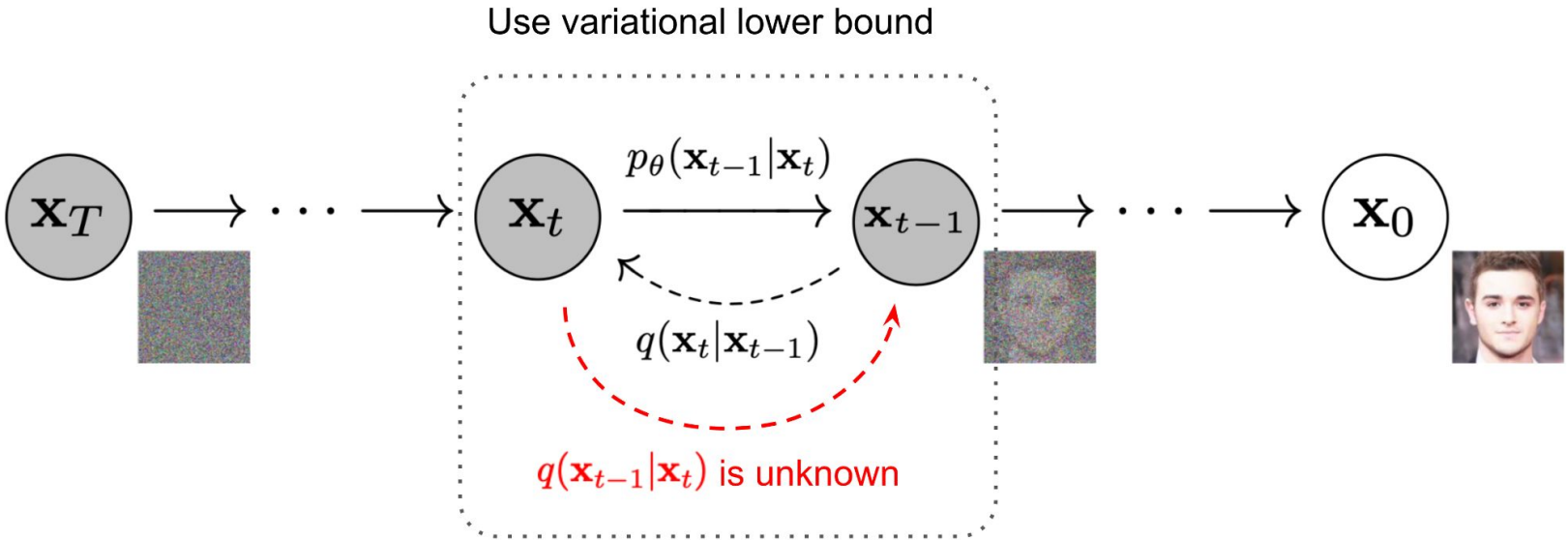
- If the discriminator behaves badly, the generator does not have accurate feedback and the loss function cannot represent the reality.
- If the discriminator does a great job, the gradient of the loss function drops down to close to zero and the learning becomes super slow or even jammed.

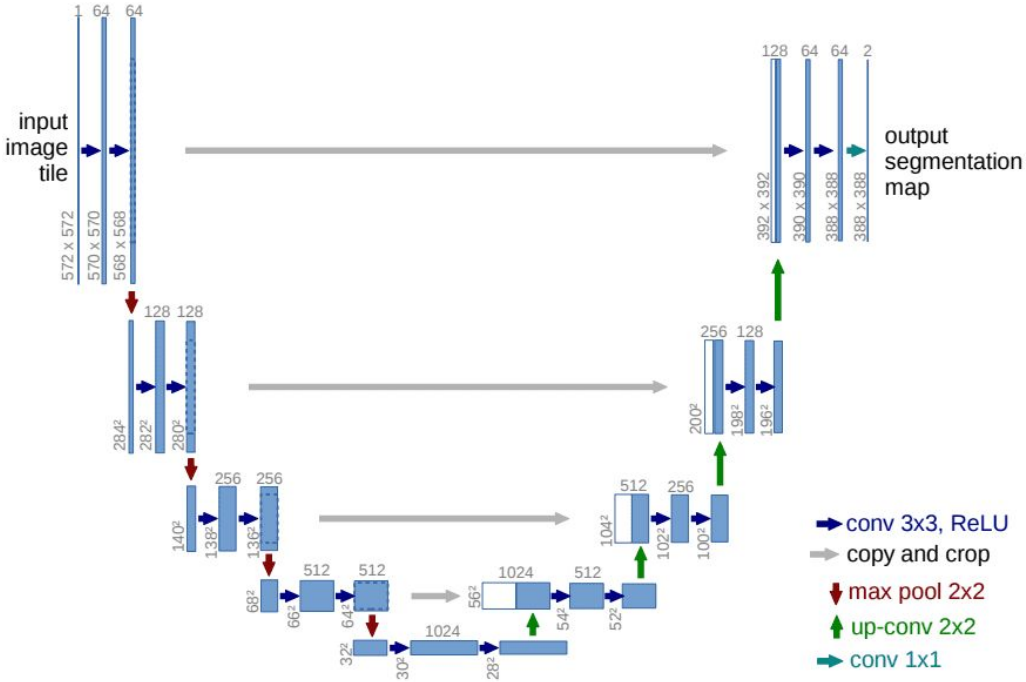
There are techniques to overcome this

-> Wassertein/Hinge loss, EMA (exponential moving average loss)

-> All just trying to make the job a bit easier for generators

Diffusion Models





Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
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

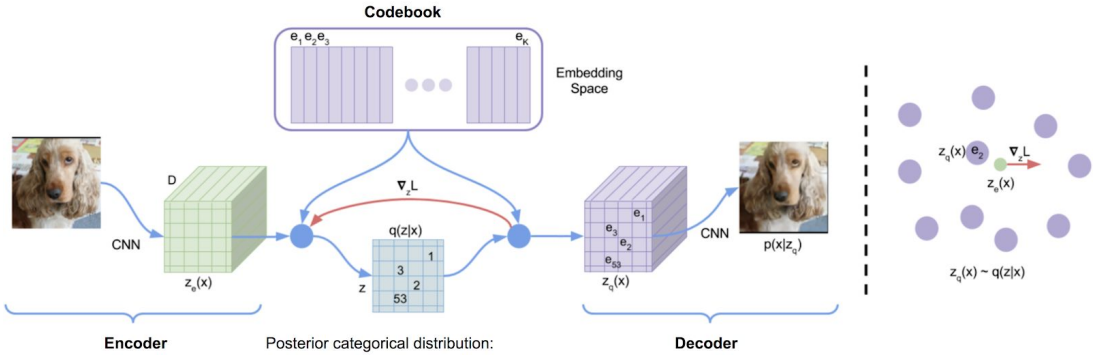
1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}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

Method	Train Speed	Sample Speed	Num. Params.	Resolution Scaling	Free-form Jacobian	Exact Density	FID	NLL (in BPD)
Generative Adversarial Networks								
DCGAN [182]	★★★★★	★★★★★	★★★★★	★★★☆☆	✓	✗	37.11	-
ProGAN [114]	★★★★☆	★★★★★	★★★★☆	★★★★★	✓	✗	15.52	-
BigGAN [19]	★★★★☆	★★★★★	★★★★★	★★★★☆	✓	✗	14.73	-
StyleGAN2 + ADA [115]	★☆☆☆☆	★★★★★	★★★☆☆	★★★★★	✓	✗	2.42	-
Energy Based Models								
IGEBM [46]	★★★☆☆	★★☆☆☆	★★★★★	★★★☆☆	✓	✗	37.9	-
Denoising Diffusion [87]	★★★★★	★★☆☆☆	★★★☆☆	★★★★☆	✓	(✓)	3.17	≤ 3.75
DDPM++ Continuous [206]	★★★★★	★★☆☆☆	★★★☆☆	★★★★☆	✓	(✓)	2.20	-
Flow Contrastive (EBM) [55]	★☆☆☆☆	★★☆☆☆	★★☆☆☆	★★☆☆☆	✓	✗	37.30	≈ 3.27
VAEBM [247]	★★☆☆☆	★★★★☆	★☆☆☆☆	★★★★☆	✓	✗	12.19	-
Variational Autoencoders								
Convolutional VAE [123]	★★★★★	★★★★★	★★★★★	★★☆☆☆	✓	(✓)	106.37	≤ 4.54
Variational Lossy AE [29]	★★★☆☆	★☆☆☆☆	★☆☆☆☆	★★☆☆☆	✗	(✓)	-	≤ 2.95
VQ-VAE [184], [235]	★★☆☆☆	★★★☆☆	★☆☆☆☆	★★★★★	✗	(✓)	-	≤ 4.67
VD-VAE [31]	★☆☆☆☆	★★★★★	★★★☆☆	★★★★★	✓	(✓)	-	≤ 2.87
Autoregressive Models								
PixelRNN [234]	★★☆☆☆	★☆☆☆☆	★☆☆☆☆	★★★☆☆	✗	✓	-	3.00
Gated PixelCNN [233]	★★★☆☆	★☆☆☆☆	★☆☆☆☆	★★★☆☆	✗	✓	65.93	3.03
PixelIQN [173]	★★★☆☆	★☆☆☆☆	★☆☆☆☆	★★★☆☆	✗	✓	49.46	-
Sparse Trans. + DistAug [32], [110]	★★★★☆	★☆☆☆☆	★★★☆☆	★★★☆☆	✗	✓	14.74	2.66
Normalizing Flows								
RealNVP [43]	★★☆☆☆	★★★★★	★★★★★	★★★☆☆	✗	✓	-	3.49
GLOW [124]	★☆☆☆☆	★★★★★	★★☆☆☆	★★★★☆	✗	✓	45.99	3.35
FFJORD [62]	★☆☆☆☆	★★★☆☆	★★★★★	★★☆☆☆	✓	(✓)	-	3.40
Residual Flow [26]	★★☆☆☆	★★★★☆	★★★★☆	★★★★☆	✓	(✓)	46.37	3.28

From “Deep Generative Modelling: A Comparative Review of VAEs, GANs, Normalizing Flows, Energy-Based and Autoregressive Models”

DALL-E 1

VQVAE



$$q(\mathbf{z} = \mathbf{e}_k|\mathbf{x}) = \begin{cases} 1 & \text{if } k = \arg\min_i \|\mathbf{z}_e(\mathbf{x}) - \mathbf{e}_i\|_2 \\ 0 & \text{otherwise.} \end{cases}$$

Transformers predict the codebook based on text

DALL-E 2

- *Autoregressive (AR)* prior: the CLIP image embedding z_i is converted into a sequence of discrete codes and predicted autoregressively conditioned on the caption y.
- *Diffusion* prior: The continuous vector z_i is directly modelled using a Gaussian diffusion model conditioned on the caption y.

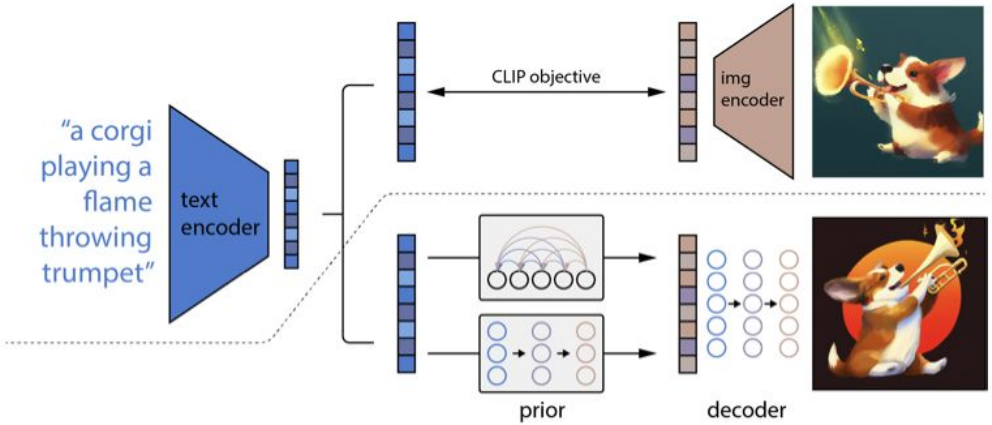
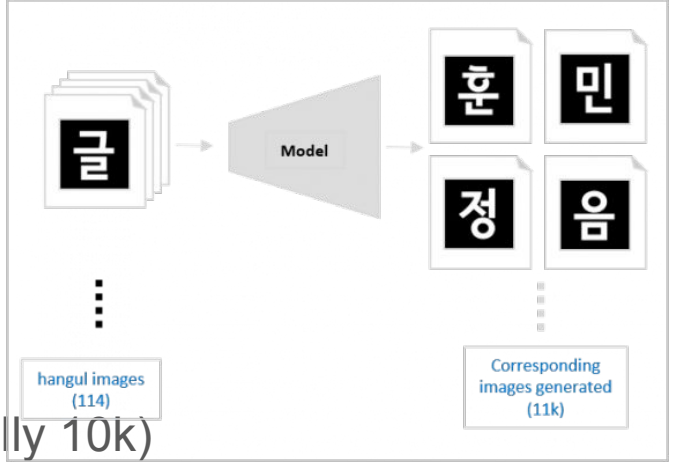


Figure 2: A high-level overview of unCLIP. Above the dotted line, we depict the CLIP training process, through which we learn a joint representation space for text and images. Below the dotted line, we depict our text-to-image generation process: a CLIP text embedding is first fed to an autoregressive or diffusion prior to produce an image embedding, and then this embedding is used to condition a diffusion decoder which produces a final image. Note that the CLIP model is frozen during training of the prior and decoder.

My work



https://www.ownglyph.com/trial/397a1b8a-85a6-41ea-8dea-4352a6a70370/

https://www.instagram.com/p/CZR5vM-P4iAX61CyA_rGTHhyOJRBr0VvAqtUmw0/

Font generation is expensive in Asian countries (typically 10k)

We generated fonts with like 30 sample characters as input -> cost 30 bucks

How would you do this?