Generating Diverse High-Fidelity Images with VQ-VAE-2

Ben Zhang

ML Reading Group

Introduction

Generative models have been making noise in the news.

- GPT2
- DeepFakes
- Facial Image Synthesis

Papers

Neural Discrete Representation Learning,

Razavi et. Al (NIPS 2018)

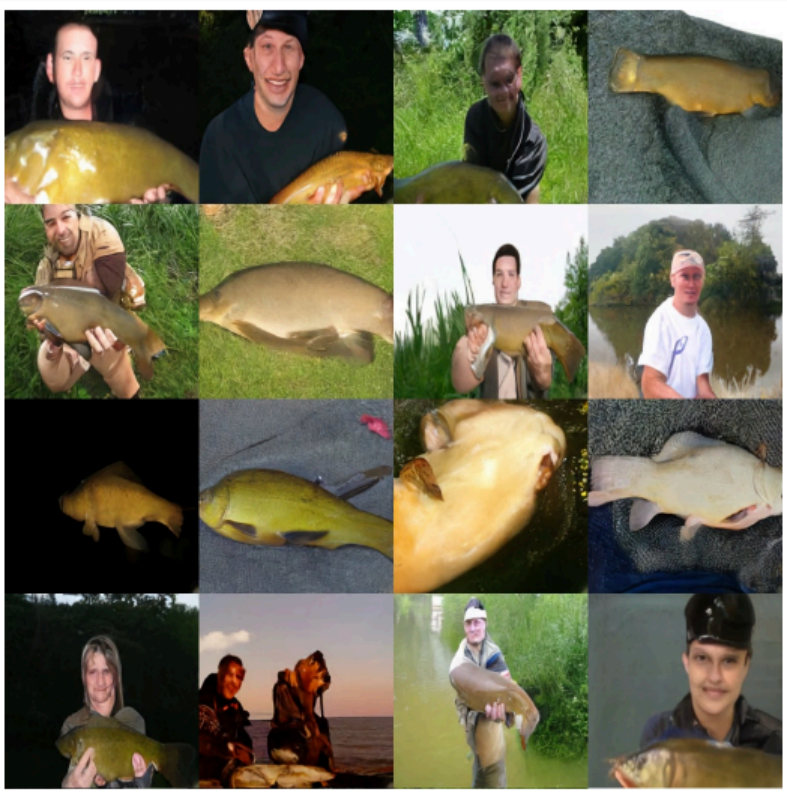
Generating Diverse High-Fidelity Images with VQ-VAE-2,

Razavi et. Al (Arxiv Preprint)

tl;dr

VQ-VAE-2 is a image synthesis model based on Variational Autoencoders.

It produces images that are high quality, comparable (FID/Inception) or perhaps better (CAS) than the state of the art model: BigGAN (ICLR 2019), which is a Google project that scaled up existing GAN architecture with TPUs and tons of compute.



VQ-VAE-2 faces

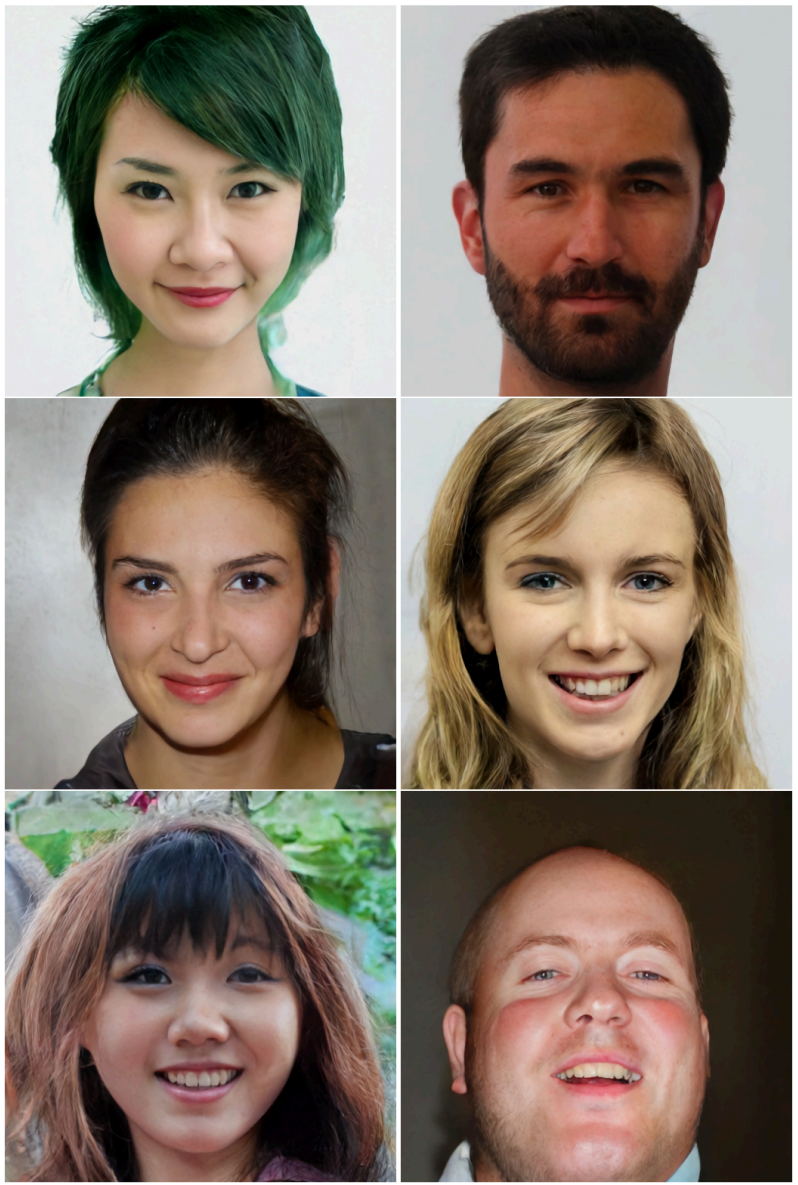


Table of contents

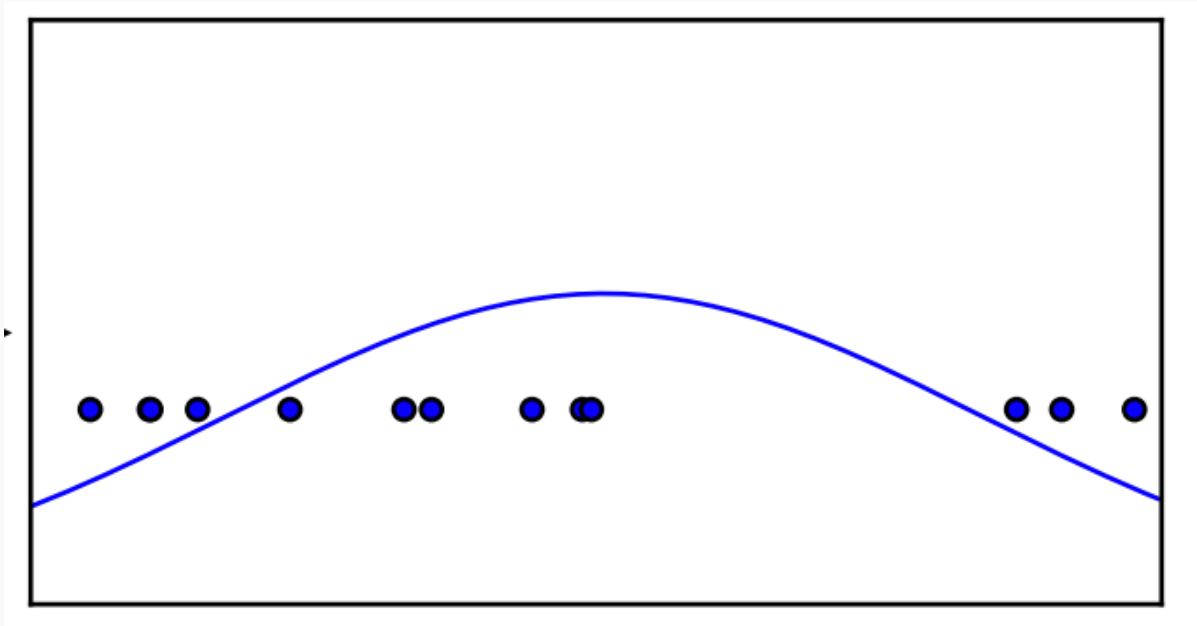
Generative Models

Generative Models

The Problem: Instead of performing classification or regression, we want to **generate** data from a distribution.

- Density Estimation: Learn the underlying density function of the sample data.
- Sample Generation: Sample points from the underlying density function.

Generative Models



Implicit vs Explicit distributions Models

Implicit Generative Model

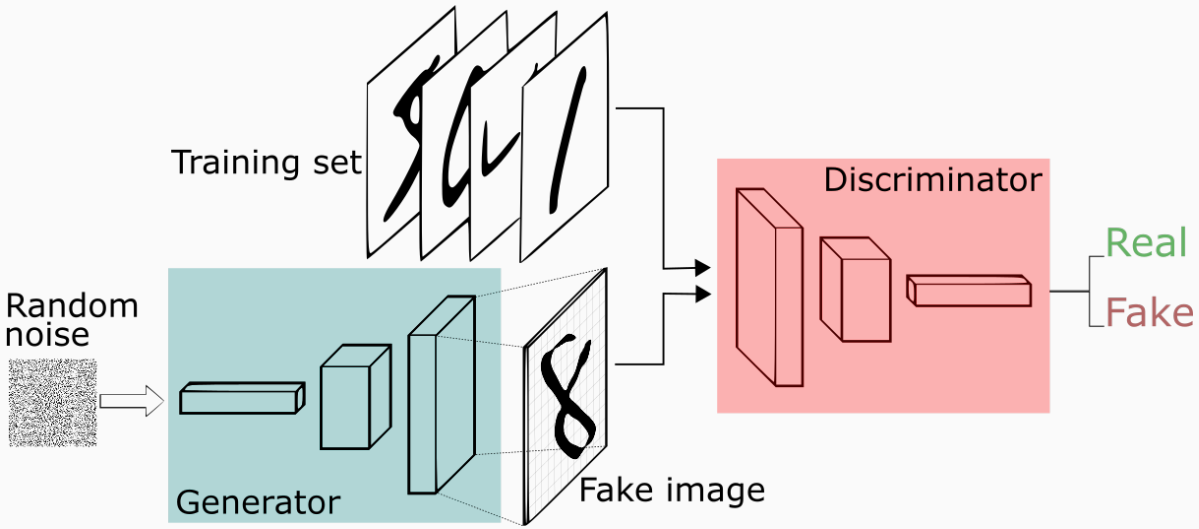
- No explicit distribution is learned.
- Eg. GANs
- (+): Really *really* good at producing realistic samples.
- (-): Difficult to converge, due to lack of metric. Optimizes for a minimax objective.
- (-): Low sample diversity due to mode collapse.

Explicit Generative model

- Explicit posterior distribution is learned.
- Eg. VAEs
- (+): Explicit metric for performance (log-likelihood) – therefore easier to train.
- (-): In the case of image synthesis, outputs are often blurry.

GANs

Overview


Training set
Discriminator
Real
Fake
Random
noise
Generator
Fake image

Generative Adversarial Networks

Two players (Neural Networks) that are adversarial to each other.

- The Discriminator, *D*: A classifier which takes a data point, and determines whether it is generated by *G*, or drawn from the true distribution.
- The Generator, *G*: generates adversarial examples from the sample distribution against the discriminator.

Example: Counterfeit Money vs Police

Consider this analogy in the real world.

- Counterfeiters create fake coins that try to look as real as possible.
- Police try to determine whether coins are real or fake.
- The two entities learn from each other, producing better coins, and better detectors.

What does this model converge to?

Generator & Discriminator

$G : Z \to X$, where Z is the latent domain, for example $Z := [0,1]^d$.

G is the generator network, which takes some random noise $z \in Z$ and produced $G(z) \in X$, a "synthetic data point". If X is the domain of pictures then $G(z)$ would be a generated picture.

$D : X \to \{0,1\}$, where is a neural network which takes in a data point from the domain X and determines whether it is real or not.

D is trained such that it maximizes the probability of labelling the inputs correctly. Its objective is to maximize:

$$J^{(D)} = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

G has the exact opposite objective function, though many variants exist.

Minimax

Then, training the two models together, we have the following minimax value function:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \quad (1)$$

Note that the order matters!

$$\min_G \max_D V(D, G) \neq \max_D \min_G V(D, G)$$

In the paper that first introduced GANs, it was shown that this optimization objective is equivalent to minimizing the Jenson-Shannon divergence of the generator's distribution, and the true distribution.

A Diagram

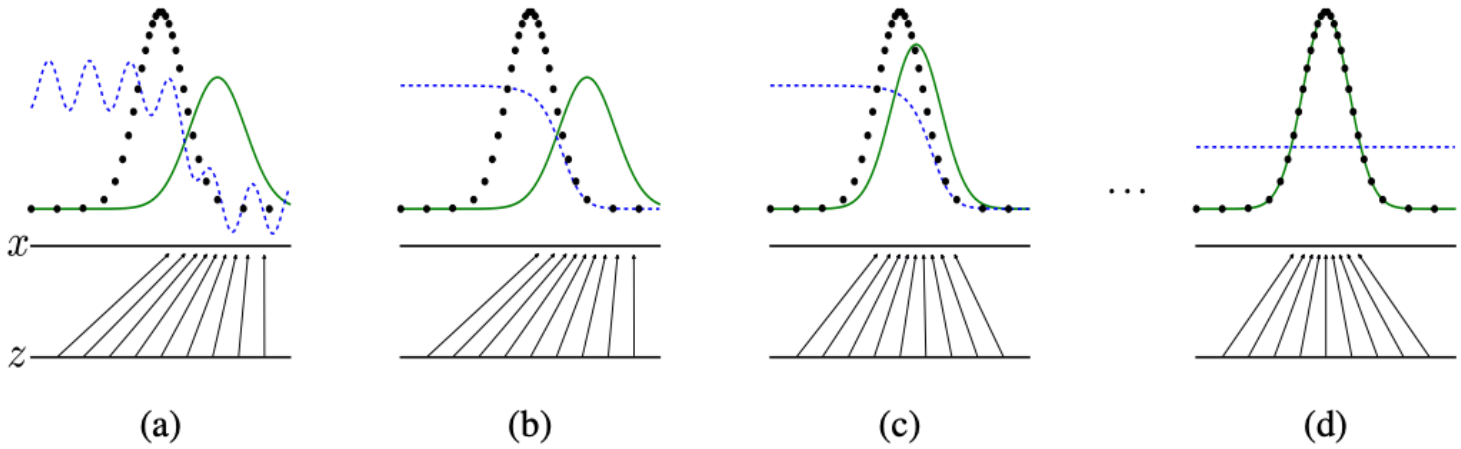


Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which $\boldsymbol{z}$ is sampled, in this case uniformly. The horizontal line above is part of the domain of $\boldsymbol{x}$. The upward arrows show how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$. (c) After an update to G, gradient of D has guided $G(\boldsymbol{z})$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(\boldsymbol{x}) = \frac{1}{2}$.

The Original Algorithm

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Issue: Mode Collapse, Low sample diversity.

Consider a bi-modal distribution: images that are mostly red or images that are mostly blue.

At any one point in the training, the Generator may simply attempt to generate samples that are either blue, or red – but never both. This is because the Generator is not incentivized to output diverse samples, only to fool the discriminator.

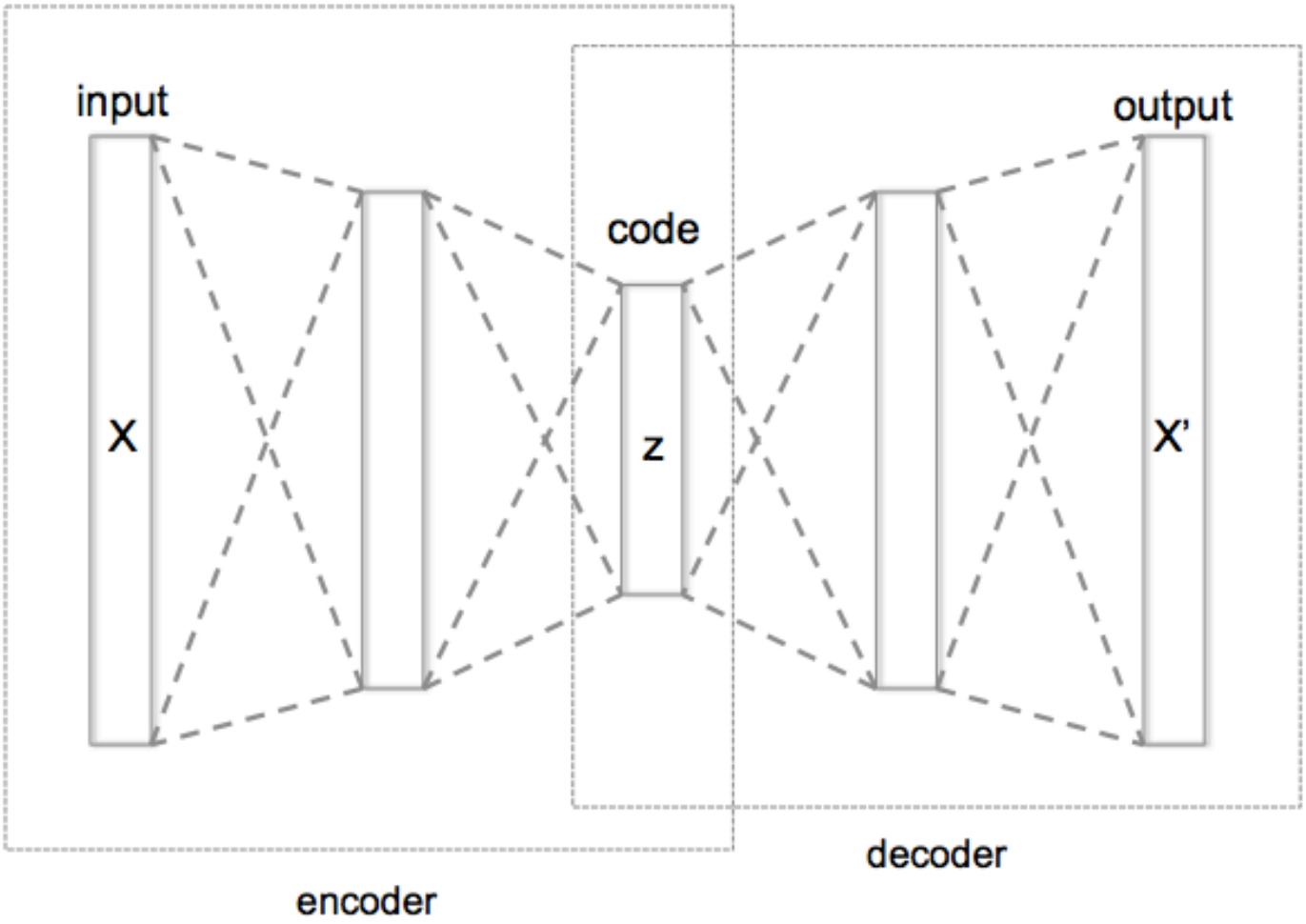
As a result, our generator will only output blue images or red images, leading to bad sample diversity.

(a) (b)

Figure 2: (a) The effects of increasing truncation. From left to right, the threshold is set to 2, 1, 0.5, 0.04. (b) Saturation artifacts from applying truncation to a poorly conditioned model.

VAEs

Autoencoders



Autoencoders

- Learns an low dimensional *latent embedding* of high dimensional data by attempting to reconstruct the high dimensional data after compressing it.
- Comparable to other dimensionality reduction technique, such as PCA. However, encoder and decoder can both be deep neural networks (non-linear).
- Contains 2 components: encoder and decoder.

Variational Autoencoders (2013)

- Instead of learning a latent space, we instead learn a latent distribution - from which we can sample latent vectors.
- Encoder no longer outputs a single latent vector z, but rather a distribution from which many z's can be sampled. For example, if we assume Gaussian latent distribution, the encoder takes x and outputs μ_z and σ_z. We can denote the encoder as the distribution $q_\theta(z|x)$.
- Decoder will take a sampled $z = \mu_z + \varepsilon * \sigma_z$ where $\varepsilon \sim N(0, 1)$ and outputs a distribution for x. We can denote the decoder as the distribution $p_\phi(x|z)$.

VAE loss function

- Has an explicit loss function: negative ELBO (Evidence Lower Bound). Evidence == Marginal likelihood.
- For datapoint x_i, loss is:

$$l_i(\theta, \phi) = -\mathbb{E}_{z \sim q_\theta(z|x_i)}[\log p_\phi(x_i|z)] + KL(q_\theta(z|x_i)||p(z)) \geq -\log p_\theta(x_i)$$

- The first part is known as the reconstruction loss. It maximizes the log likelihood of the posterior distribution.
- The second part is the regularization term. It minimizes the encoder distribution and the prior latent distribution, such that the encoder will learn to converge to the prior.
- This value is a bound on the evidence, $\log p(x)$ – proven with Jensen's Inequality.

Generating synthetic data with VAEs

You probably know that

$$p(x, z) = p(z) * p(x|z)$$

. We have the decoder to give us $p(x|z)$, so we can generate x given z.

But what is our choice of prior, $p(z)$?

In many cases, it is chosen to be a spherical Gaussian distribution. It is up to the encoder to recognize this as the true latent prior, and thus map x accordingly into the latent space.

Therefore, we can easily generate synthetic data with VAEs.

In conclusion, VAEs have 3 parts: Encoder, Decoder, Prior.

Issue: VAEs Generate Blurry Images

- Since the prior latent distribution is normal, there is a lot of noise when images are generated.
- This noise introduces blurriness when VAEs are tasked with image synthesis.

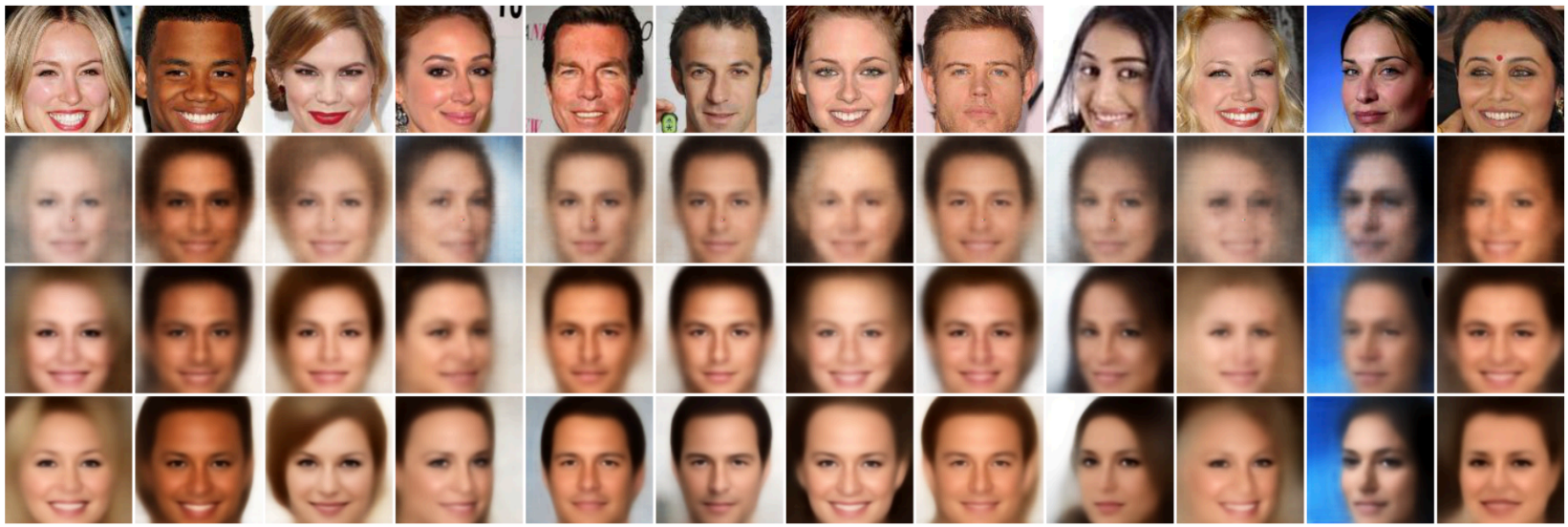


Figure 1: Comparison of reconstructed images from the CelebA dataset. The first row is the input images in the CelebA training set. The second row is the reconstructed images generated by the original VAE. The third and fourth rows are the results of deep residual VAE and multi-stage VAE, respectively.

Vector Quantized VAEs

PixelCNN

- Autoregressive model, that takes in a latent vector (or any conditional vector) as input, and produces a distribution over images. Used as the decoder for VQ-VAE.
- PixelCNN can perform *conditioned* image generation based on the latent vector it receives as input.
- In VQ-VAE, they assume the prior distribution $p(z)$ is simply a uniform categorical distribution - all codewords are equally likely. To condition the generated output, they encode the ”query”, and feed the output to the PixelCNN decoder.

Issue: Posterior Collapse

- PixelCNN is a powerful model that can capture a lot of complexity in its distribution. While this is great for generating high quality samples, a problem arises when using it for a VAE decoder: Posterior Collapse.
- If the decoder is too powerful, it may just ignore the latent variables. This happens when the decoder can take the reconstruction loss to be incredibly small, such that the regularization term dominates the loss function. In such a case, the encoder will learn to reduce the regularization term, and produce meaningless latents to match $p(z) = N(0, 1)$.
- Vector Quantization is meant to address this problem.

Vector Quantization

Instead of a continuous latent space, we restrict the set of all latent vectors to be in a discrete set of ”codewords” (think Coding Theory).

When considering a latent vector not in the codeword set, it is ”discretized” by converting it to its nearest neighbour in the codeword set.

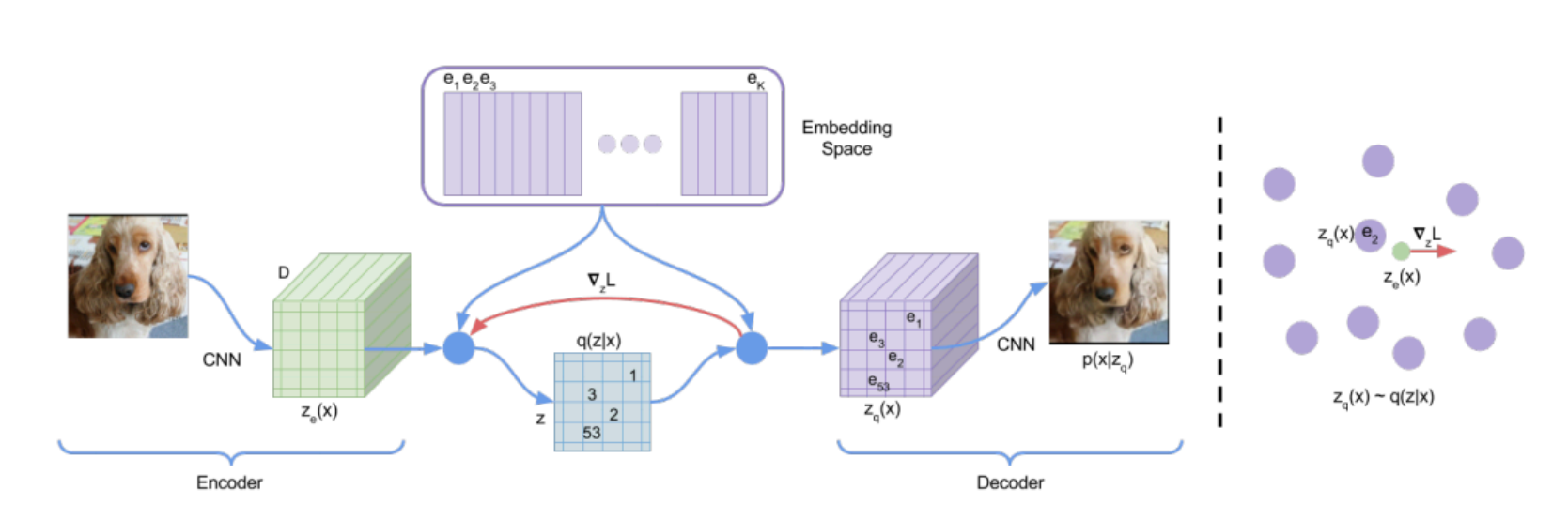
The set of codewords is also learned, and are included in the loss function:

$$L = \log p(x|z_q(x)) + ||sg[z_e(x)] - e||_2^2 + \beta||z_e(x) - sg[e]||_2^2$$

Here, $z_e(x)$ is the encoder output. Codewords are incentivized to move closer to the encoded latent vector.

By fixing the latent space to be discrete, the Encoder can no longer choose to encode 0 information in order to minimize the regularization term.

VQ-VAE


e_1 e_2 e_3
e_K
Embedding Space
D
CNN
$z_e(x)$
$\nabla_z L$
q(z|x)
z
1
3
2
53
e_1
e_3
e_2
e_{53}
$z_q(x)$
CNN
$p(x|z_q)$
Encoder
Decoder
$z_q(x)$ e_2
$\nabla_z L$
$z_e(x)$
$z_q(x) \sim q(z|x)$

VQ-VAE Results

Figure 2: Left: ImageNet 128x128x3 images, right: reconstructions from a VQ-VAE with a 32x32x1 latent space, with K=512.

- Does not suffer from posterior collapse when using PixelCNN as decoders in the way that vanilla VAEs do, since the prior distribution is a uniform sample over a discrete space.

VQ-VAE speech results

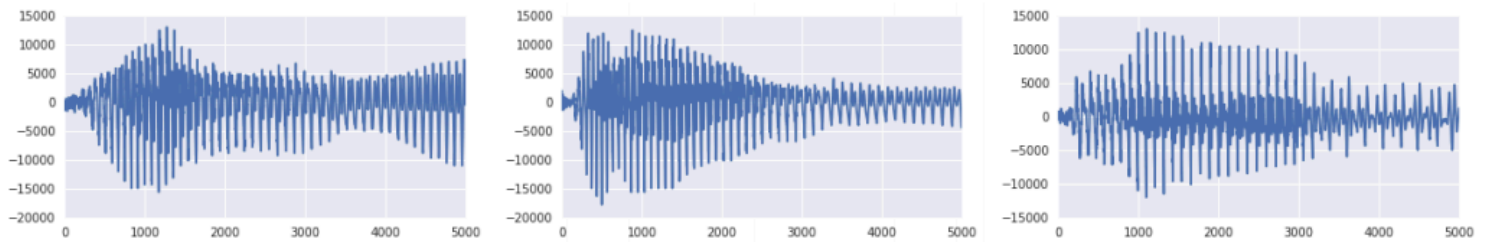


Figure 6: Left: original waveform, middle: reconstructed with same speaker-id, right: reconstructed with different speaker-id. The contents of the three waveforms are the same.

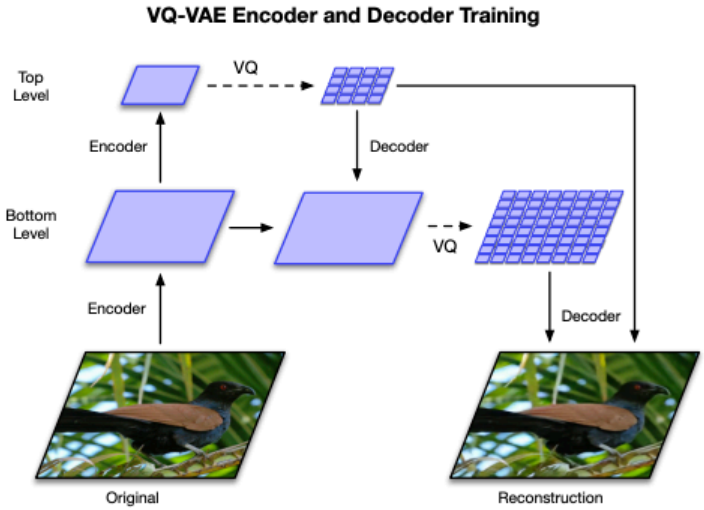
- When trained on raw speech data using WaveNet as a decoder, the encoder still provided a representative latent space, from which VQ-VAE was able to learn phonemes from speech alone in its latent space.
- By attempting to lower the dimensionality of the latent space, various speech textures disappear, but the essence of the text remains until the latent space is 128x lower in dimensionality.
- It does this unsupervised on a downstream task.

VQ-VAE-2

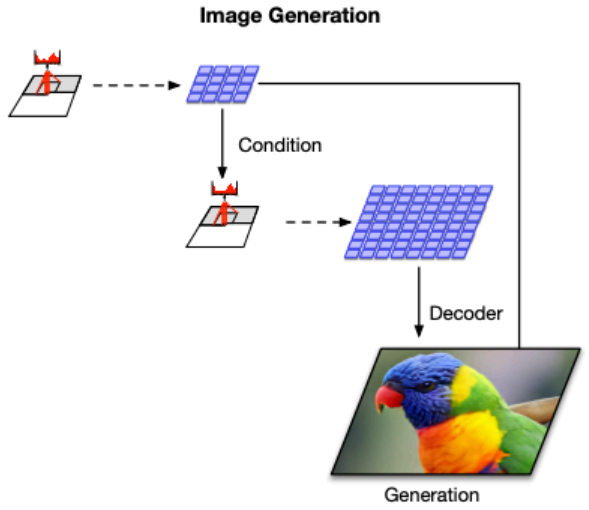
Changes from original VQ-VAE

- Hierarchical latent spaces
- Learned priors, instead of uniform distribution.

VQ-VAE-2



(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).

Figure 2: VQ-VAE architecture.

VQ-VAE-2: Hierarchical Latents

Figure 3: Reconstructions from a hierarchical VQ-VAE with three latent maps (top, middle, bottom). The rightmost image is the original. Each latent map adds extra detail to the reconstruction. These latent maps are approximately 3072x, 768x, 192x times smaller than the original image (respectively).

VQ-VAE-2: Learning Priors

Algorithm 1 VQ-VAE training (stage 1)

Require: Functions E_{top} , E_{bottom}, D, $\mathbf{x}$ (batch of training images)

$\mathbf{h}_{top} \leftarrow E_{top}(\mathbf{x})$

▷ quantize with top codebook eq 1

$\mathbf{e}_{top} \leftarrow Quantize(\mathbf{h}_{top})$

$\mathbf{h}_{bottom} \leftarrow E_{bottom}(\mathbf{x}, \mathbf{e}_{top})$

▷ quantize with bottom codebook eq 1

$\mathbf{e}_{bottom} \leftarrow Quantize(\mathbf{h}_{bottom})$

$\hat{\mathbf{x}} \leftarrow D(\mathbf{e}_{top}, \mathbf{e}_{bottom})$

▷ Loss according to eq 2

$\theta \leftarrow Update(\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}}))$

Algorithm 2 Prior training (stage 2)

$\mathbf{T}_{top}, \mathbf{T}_{bottom} \leftarrow \emptyset$ ▷ training set

for $\mathbf{x} \in$ training set **do**

  $\mathbf{e}_{top} \leftarrow Quantize(E_{top}(\mathbf{x}))$

  $\mathbf{e}_{bottom} \leftarrow Quantize(E_{bottom}(\mathbf{x}, \mathbf{e}_{top}))$

  $\mathbf{T}_{top} \leftarrow \mathbf{T}_{top} \cup \mathbf{e}_{top}$

  $\mathbf{T}_{bottom} \leftarrow \mathbf{T}_{bottom} \cup \mathbf{e}_{bottom}$

end for

$p_{top} = \texttt{TrainPixelCNN}(\mathbf{T}_{top})$

$p_{bottom} = \texttt{TrainCondPixelCNN}(\mathbf{T}_{bottom}, \mathbf{T}_{top})$

▷ Sampling procedure

while true **do**

  $\mathbf{e}_{top} \sim p_{top}$

  $\mathbf{e}_{bottom} \sim p_{bottom}(\mathbf{e}_{top})$

  $\mathbf{x} \leftarrow D(\mathbf{e}_{top}, \mathbf{e}_{bottom})$

end while

Results

Great Sample Diversity (compared to BigGAN)

VQ-VAE (Proposed) **BigGAN deep**

Figure 5: Sample diversity comparison for the proposed method and BigGan Deep for Tinca-Tinca (1st ImageNet class) and Ostrich (10th ImageNet class). BigGAN samples were taken with the truncation level 1.0, to yield its maximum diversity. There are several kinds of samples such as top view of the fish or different kinds of poses such as a close up ostrich absent from BigGAN's samples. Please zoom into the pdf version for more details and refer to the Supplementary material for diversity comparison on more classes.

8