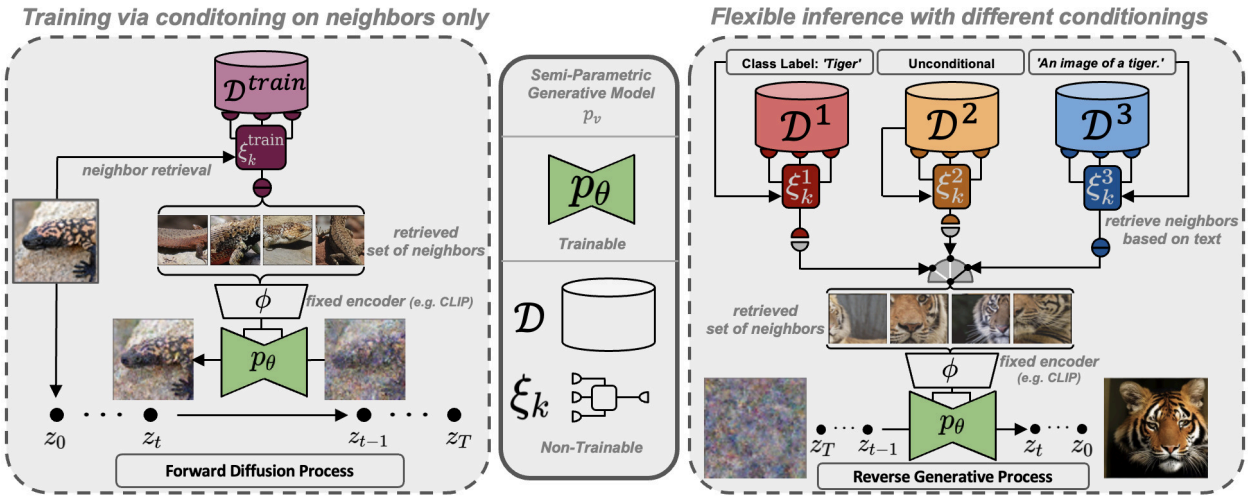
Paper

- https://arxiv.org/pdf/2112.10752.pdf

Code

- https://github.com/CompVis/latent-diffusion



Introduction to problem

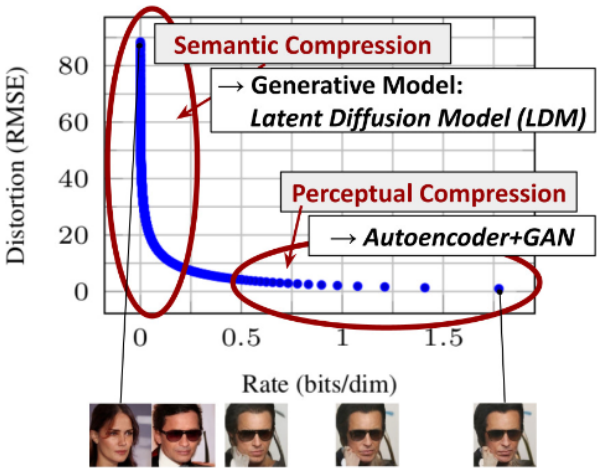
- Autoregressive transformers
- GANs are data confined and have limited variability
    - Does not scale well
- Here comes diffusion models
    - Built from denoising autoencoders
    -

Diffusion model downsides

- Needs to increase accessibility and reduce resource consumption

Deep dive into model

- The paper dives analyzes the latent space of DM to find improvements
- **Perceptual compression stage**
- Semantic compression



Contributions

- Paper claims to scale better with higher dimensional data
- Claims to achieve better performance on multiple tasks (unconditional image synthesis, inpainting, stochastic super-resolution) while significantly lowering computational costs and decreasing inference costs.
    - Unconditional image synthesis - task of generating *images* with no condition in any context (like a prompt text or another *image*).
    - Inpainting - **conservation process where damaged, deteriorated, or missing parts of an artwork are filled in to present a complete image.**
    - Stochastic super resolution - for increasing resolution of an image
- Their approach does not require a delicate weighting of reconstruction and generative abilities. This ensures extremely faithful reconstructions and requires very little regularization of the latent space.
- We find that for densely conditioned tasks such as super-resolution, inpainting and semantic synthesis, our model can be applied in a convolutional fashion and render large, consistent images of ~ 10242 px.
- They designed a general-purpose conditioning mechanism based on cross-attention, enabling multi-modal training
    - use it to train class-conditional, text-to-image and layout-to-image models.

Related Work

- Generative models for image synthesis
    - Difficult to optimize
    - Likelihood based methods are better
- Diffusion probabilistic models (DM)
- Two stage image synthesis

Method

- Introduces separation of compressive form from generative learning phase

Perceptual Image Compression

- Based on previous work
- Autoencoder trained with perceptual loss and patch-based adversarial objective
- Has KL-reg
- Done during training
- First, we train
- an autoencoder which provides a lower-dimensional (and
- thereby efficient) representational space which is perceptu-
- ally equivalent to the data space.

Semantic Image Compresision

- In latent space

Latent Diffusion Models

-

Conditioning Mechanisms

- augmenting their underlying UNet backbone with the cross-attention mechanis

Experiments

-

Discussions

- Is Unet pretrained?
  - Unsure yet
- Which part refers to the autoencoder
  - Its the initial pixel space
- Whats their loss for actual training
  - its only using 1 model

$$L_{DM} = \mathbb{E}_{x,\epsilon\sim\mathcal{N}(0,1),t}\left[\|\epsilon - \epsilon_\theta(x_t, t)\|_2^2\right], \qquad (1)$$

  -
  - They only train the denoiser
  - Squared L2 norm
  -
  -
  -
- What is T_theta
  - Apparently it is the output of the conditioning mechanisms
  - It is where you jointly train T_theta
  - unmasked transformer which processes a tokenized version of the input y
  - to infer a latent code which is mapped into the UNet via (multi-head) cross-attention (Sec. 3.3).
- What is time conditional unit ?
  - Seems to be a positional encoding

G. Details on Autoencoder Models

We train all our autoencoder models in an adversarial manner following [23], such that a patch-based discriminator D_ψ is optimized to differentiate original images from reconstructions $\mathcal{D}(\mathcal{E}(x))$. To avoid arbitrarily scaled latent spaces, we regularize the latent z to be zero centered and obtain small variance by introducing an regularizing loss term L_{reg}.
We investigate two different regularization methods: (i) a low-weighted Kullback-Leibler-term between $q_\mathcal{E}(z|x) = \mathcal{N}(z; \mathcal{E}_\mu, \mathcal{E}_{\sigma^2})$ and a standard normal distribution $\mathcal{N}(z; 0, 1)$ as in a standard variational autoencoder [46, 69], and, (ii) regularizing the latent space with a vector quantization layer by learning a codebook of $|\mathcal{Z}|$ different exemplars [96].
To obtain high-fidelity reconstructions we only use a very small regularization for both scenarios, *i.e.* we either weight the $\mathbb{KL}$ term by a factor $\sim 10^{-6}$ or choose a high codebook dimensionality $|\mathcal{Z}|$.

The full objective to train the autoencoding model $(\mathcal{E}, \mathcal{D})$ reads:

$$L_{\text{Autoencoder}} = \min_{\mathcal{E},\mathcal{D}} \max_{\psi} \Big(L_{rec}(x, \mathcal{D}(\mathcal{E}(x))) - L_{adv}(\mathcal{D}(\mathcal{E}(x))) + \log D_\psi(x) + L_{reg}(x; \mathcal{E}, \mathcal{D}) \Big) \tag{25}$$

DM Training in Latent Space Note that for training diffusion models on the learned latent space, we again distinguish two cases when learning $p(z)$ or $p(z|y)$ (Sec. 4.3): (i) For a KL-regularized latent space, we sample $z = \mathcal{E}_\mu(x) + \mathcal{E}_\sigma(x) \cdot \varepsilon =: \mathcal{E}(x)$, where $\varepsilon \sim \mathcal{N}(0, 1)$. When rescaling the latent, we estimate the component-wise variance

$$\hat{\sigma}^2 = \frac{1}{bchw} \sum_{b,c,h,w} (z^{b,c,h,w} - \hat{\mu})^2$$

from the first batch in the data, where $\hat{\mu} = \frac{1}{bchw} \sum_{b,c,h,w} z^{b,c,h,w}$. The output of $\mathcal{E}$ is scaled such that the rescaled latent has unit standard deviation, *i.e.* $z \leftarrow \frac{z}{\hat{\sigma}} = \frac{\mathcal{E}(x)}{\hat{\sigma}}$. (ii) For a VQ-regularized latent space, we extract z *before* the quantization layer and absorb the quantization operation into the decoder, *i.e.* it can be interpreted as the first layer of $\mathcal{D}$.

Code

- https://github.com/CompVis/latent-diffusion/tree/main/ldm/models
- https://github.com/CompVis/latent-diffusion/tree/main/ldm/models

Ddim and ddpm

- https://github.com/CompVis/latent-diffusion/blob/main/ldm/models/diffusion/ddim.py]

They use this mainly

- https://github.com/CompVis/latent-diffusion/blob/main/ldm/models/diffusion/ddim.py

Why add positional encodings vs concat

- It seems concat makes more sense because you dont change the weights ? intuitively it makes sense
- It seems add is more computationally efficient

- https://www.reddit.com/r/MachineLearning/comments/cttefo/d_positional_encoding_in_transformer/exs7d08/?context=3