DDPM vs VAE On Image Synthesis

Maha Mapara
maha.mapara@tufts.edu

Tina Wu
yiyun.wu@tufts.edu

Abstract

Purpose: The goal of our project is to understand how Denoising Diffusion Probabilistic Models (DDPM) [Ho et al. (2020)] work and explore its advantages and limitations by comparing its performance against VAE's [Kingma and Welling (2013)] on image generation tasks across a variety of datasets. This is an exciting project, for the opportunity of researching one of the cutting-edge techniques in the field.
Hypothesis: Given the same training input, DDPM will slightly outperform Convolutional VAE on image synthesis, because DDPM retains higher dimensions and it is more flexible. Performance will be evaluated via sampled image qualities and how well sampled images resemble the input.
Evaluation plan: We will use MNIST [Lecun et al. (1998)] and CIFAR-10 [Krizhevsky (2009)] data sets: the first data set produces visually analyzable results, which are especially useful when implementing a newer algorithm, and the latter one assesses model's capability on natural images and the diversity of output samples. We will measure model loss using the variational bound and quantify image qualities with FID [Heusel et al. (2017)].
Conclusion: VAE with less complicated network architectures and noticeably shorter training time outperforms DDPM on MNIST data, while DDPM ouperforms VAE on CIFAR-10. Type of data and difference in architecture should also be factors in deciding which model to use for a task.

1 Introduction and Goals

The goal of this project was to learn a relatively new technique in the literature, DDPM [Ho et al. (2020)], and explore especially its advantages and weaknesses. To achieve this goal, we implemented a DDPM model and compared it against an existing generative model, VAE [Kingma and Welling (2013)], on image generation, a task that takes in an image dataset as input and produces sampled images as output. To bridge the gap between the architectures of the two models, we used a Convolutional VAE, i.e. VAE with convolutional and deconvolutional layers in the encoder and decoder, for a more controlled comparison. Performances of both models were evaluated via ELBO and quality of the sampled images, quantified by FID score [Heusel et al. (2017)].

We hypothesized that DDPM would outperform Convolutional VAE slightly on image synthesis (produce better quality images), because latent variables in DDPM retain the exact dimensionality as the input, while VAE is restricted by a bottleneck layer. In VAE, information loss is introduced through dimensionality reductions. Additionally, VAE is known for sometimes suffering from blurry samples, which are caused by the regularization of forcing approximated posterior onto a Gaussian prior distribution [Zhao et al. (2017)].

This is an interesting project because diffusion models are currently an exciting area of research in generative models, especially for image synthesis tasks; many popular architectures use diffusion models for this purpose, like DALLE-2 and Imagen. This was a good opportunity to gain in-depth exposure about the architecture and modeling approach used for diffusion models. In particular, we were inspired by the paper 'Diffusion Models Beat GAN on Image Synthesis' [Dhariwal and Nichol (2021)], where the authors compare DDPM performance with GANs on image synthesis using ImageNet and LSUN image data. Instead of comparing DDPM to GAN, we compared diffusion models with VAEs. This is because:

1. We have not seen a comparison between VAE and diffusion models in the literature on our choice of data sets.
2. The models have some similarities:
   - Forward process that turns data into latent representations
   - Reverse process that involves sampling some latent variable
   - Training objective which is a lower bound on data likelihood
3. It would be interesting to see how different the model performances are, and evaluate the expected underperformance of VAEs based on our understanding of the key differences between DDPMs and VAEs.

2 Methods

2.1 DDPM

The key probabilistic model we are focusing on is the DDPM. There are two main steps involved: the forward process, denoted with q distribution, and the reverse process, denoted as p_θ. We used the random variable, x_0, to represent the input training data. There are many variations to and evolution built on top of the original diffusion model [Sohl-Dickstein et al. (2015)], and we decided to implement mainly the algorithms described in [Ho et al. (2020)]. Diffusion models are composed of two Markov chains, namely forward training process and reverse sampling process, and we have described the details in the subsections below.

2.1.1 Forward Process

The forward process is a Markov chain that converts the input variable, x_0, to a standard Gaussian latent variable with the same size, x_T, in T steps. The transformation is achieved by adding Gaussian noise at each intermediate steps of the chain, $\{x_1, x_2, ..., x_t, ..., x_{T-1}\}$. Therefore, the estimated posterior can be defined as a joint distribution of all noised latent variables, $\{x_1, x_2, ..., x_T\}$, given x_0 [Sohl-Dickstein et al. (2015)]:

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1}). \tag{1}$$

Each intermediate distributions, corresponding to the RHS of Equation 1, is defined as a Gaussian distribution:

$$q(x_t|x_{t-1}) = N(\sqrt{1-\beta_t}x_{t-1}, \beta_t I), \tag{2}$$

where β_t is the variance schedule, which governs the noise added at each step. There are a number of choices for the noise schedule, $\beta_{1:T}$, such as linear, cosine, quadratic, constant, etc [Sohl-Dickstein et al. (2015)]. However, one pain with sampling x_t from Equation 2 is its inefficiency: every intermediate latent variable, x_t, is dependent on all previous variables, $\{x_1, x_2, ..., x_{t-1}\}$. To simplify it, instead of conditioning on x_{t-1} and calculate every single step in the chain, we will condition the probability distribution $q(x_t)$ on the input data, x_0, and derive $\overline{\alpha}_t$ from $\beta_{1:T}$ using the properties of marginals of Gaussians:

$$q(x_t|x_0) = N(\sqrt{\overline{\alpha}_t}x_0, (1-\overline{\alpha}_t)I), \tag{3}$$

with $\overline{\alpha}_t = \prod_{s=1}^{t} 1 - \beta_s$. Equation 3 allows us to sample x_t from $q(x_t|x_0)$ at any arbitrary timestep t with the following [Ho et al. (2020)]:

$$x_t = \sqrt{\overline{\alpha}_t}x_0 + \sqrt{(1-\overline{\alpha}_t)}\epsilon, \tag{4}$$

Table 1: Hyperparameters in DDPM

Hyperparameter	Justification
Timesteps=300	Fixed to a large value for simplicity
num_epochs=100	Fixed to a large value for simplicity
batch_size=128	Fixed for simplicity
variance schedule β	used linear variance, same as the paper
Learning rate	Fit by looping through a list of values linearly in $[0.0001, 0.001]$

where ϵ is a standard Gaussian. For this project's specific implementation, the goal of the model is learn and predict ϵ, the noise transition needed to sample x_t in Equation 4, such that it can be used in the sampling process to denoise each latent variable, $\{x_T, x_{T-1}, ..., x_1\}$. To achieve the goal, the algorithm in [Ho et al. (2020)] defines $\epsilon_\theta(x_t, t)$, a neural network that outputs predicted noise ϵ_θ, parameterized over x_t and t. Thus, [Ho et al. (2020)] also suggests that the training objective, or loss function, can be simplified from variational lower bound to MSE between true noise used in sampling x_t and predicted noise of the model:

$$||\epsilon - \epsilon_\theta(x_t, t)||^2, \tag{5}$$

where x_t is sampled using Equation 4. Based on Equation 4 and Equation 5, the purpose of t in the neural network is to determine the noise level, or how noised are batch x_t from input x_0.

2.1.2 Reverse Process

The sampling process, on the other hand, is a full Markov chain, starting with the prior being a standard Gaussian distribution, $x_T \sim N(0, I)$, and our goal is to sample x_0 at the end of the process through a gradual denoising process. Same as the forward process, the reverse process can be defined as a joint distribution of intermediate distributions in the Markov chain:

$$p(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p(x_{t-1}|x_t)$$

Every intermediate latent variables, $\{x_{T-1}, x_{T-2}, ..., x_0\}$, can be sampled from a Gaussian distribution with

$$\mu = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\overline{\alpha}_t}}\epsilon_\theta(x_t, t)), \qquad \sigma = \sqrt{\frac{1-\overline{\alpha}_{t-1}}{1-\overline{\alpha}_t}\beta_t}z,$$

where
ϵ_θ is the Gaussian transition learned by the model during training,
$\alpha_t = 1 - \beta_t$,
and z is sampled from a standard Gaussian distribution [Ho et al. (2020)].

2.1.3 DDPM Hyperparameters

See Table 1 for detailed hyperparameters and strategies.

2.2 VAE

Variational Autoencoders (VAEs) were the baseline model we used in our experiments. They provide a probabilistic manner for describing an observation in the latent space. [Kingma and Welling (2013)]

There are two main parts of a VAE: the encoding and decoding. If we liken VAE to DDPM, the encoding is similar to the DDPM's forward process and the decoding is similar to DDPM's decoding process. Similar to DDPM, encoding process is denoted with q_θ distribution, and the decoding process is denoted as p_θ. We will use the random variable, x, to represent the input training data.

2.2.1 Encoding

In the encoding process, the input, x, is compressed to a latent space, z. The encoder is a neural network denoted by $q_\phi(z|x)$, parameterized over $\phi = \{W, b\}$. The latent representation has lower

Table 2: Hyperparameters in VAE

Hyperparameter	Justification
hidden_layer_size=128	Tried [16, 32, 64, 128], large latent dimensions retains more information about the features
Variance=0.2 & 0.6	Looped over np.linspace(0.1, 1.0, 0,1), 0.6 gives lowest loss for CIFAR10 dataset
batch_size=128	Fixed for simplicity
num_epochs=150	Fixed to a large value for simplicity
Learning rate	Fit by looping through a list of values linearly in $[0.0001, 0.001]$

dimensions than x; usually with nonlinear dimensionality reduction. This is typically referred to as a 'bottleneck', because the encoder must learn an efficient compression of the data into this lower-dimensional space. Approximated posterior $q_\theta(z|x)$ should be as close to the prior $p(z)$, a Guassian distribution, as possible. Such conversion are trained with the KL divergence term in the loss function. We can sample mean and variance from reduced features and use reparameterization trick to form latent probability distribution z. [Altosaar (2021)]

2.2.2 Decoding

Decoder, denoted by $p_\theta(x|z)$, usually has a similar but reverse network structure as encoder. The goal of the decoding process is to reconstruct input images by mapping the latent representation z back to the original input dimensionality. In order to train the model for producing high quality images, VAE has a second punishing term in the loss function that calculates the reconstruction error between the input data and decoder output. Usually BCE is used, because probability distribution ranges [0,1].

Both encoder and decoder give rises to one form of the variational objective for VAE, which is the lower bound on marginal likelihood, $\sum logp_\theta(x)$ [Kingma and Welling (2013)]:

$$L = -D_{KL}(q_\phi(z|x)||p_\theta(z)) + \mathbb{E}[logp_\theta(x|z)]. \tag{6}$$

The KL divergence between the approximate posterior and the prior forces the first distribution into a Gaussian distribution. The second term in the objective measures the reconstruction error by sampling from the decoder.

2.2.3 VAE Hyperparameters

See Table 2 for detailed VAE hyperparameters and strategies.

3 Experiments

3.1 Data Description

Our two datasets are MNIST and CIFAR-10.

MNIST [Lecun et al. (1998)] is an image dataset, composed of handwritten digits of 0-9, with 60,000 examples in the training split. It is 28x28 pixels in grayscale, each pixel ranging 0 to 255, with digits centered in the image. We heavily tested MNIST for our implementation, because MNIST is clear and easy to interpret. Among all image datasets, MNIST is very simple (gray-scale) and takes a shorter time to train, which was appropriate for implementing the algorithm and working through coding issues. Also, MNIST can comprehensively be used to analyze image quality. Therefore, we compared the results of the diffusion model on MNIST against VAE on MNIST easily.

CIFAR10 [Krizhevsky (2009)] is composed of 10 different classes of natural images, including cats, dogs, cars, etc. It is more complicated than MNIST, considering that it has 3 channels instead of 1, and each image is 32x32 pixels. We relied more on CIFAR-10 to evaluate the true performance of each method, mainly because it resembles true real-world data more than MNIST.

Table 3: FID scores of sampled images

Dataset	DDPM	VAE
MNIST	93.74	87.27
CIFAR10	227.68	301.43

3.2 Experimental protocol

We used the shuffling 'train' split of the dataset to train the model, and validate model performance at the end of each epoch using the 'test' split. We used log likelihood to measure the accuracy of each model. This was also our loss function, which told us how good the model fits the dataset and whether there were signs of overfitting or underfitting. Loss was plotted for each model (figures 3,4,7,8).
To generate images for testing the hypothesis, we started the sampling processes for both models with standard Gaussian in their respectively sizes. FID scores Heusel et al. (2017) were used to measure how good the qualities of the generated samples are, by calculating the difference between feature vectors of real images and the feature vectors of generated images. The lower the FID score, the fewer distortions in the image and better the image quality. Therefore, the model that produced images with lower FID had a better performance.

If both variational bound and FID for DDPM were lower than for VAE, our hypothesis was confirmed, showing that DDPM outperformed VAE in both accuracy and quality of sampled images. If both metrics in DDPM were higher than those in VAE, our hypothesis was disproved; otherwise, it was a 'tie', or we would need to perform further experiments to evaluate the performance.

3.2.1 Model Architecture

DDPM The overall learning process of the model was built and trained with pytorch library and Adam optimizer [Paszke et al. (2019)]. Other libraries used were numpy [Harris et al. (2020)], math [Van Rossum (2020)], and matplotlib [Hunter (2007)]. As suggested in the DDPM paper [Ho et al. (2020)], we adopted U-net from Rogge and Rasul (2022) as the backbone architecture to our neural network. U-net is composed of a contracting and an expansive path. Similar to CNN, the input image is down-scaled in dimensions and up-scaled in channels during the contracting path, and the process is reversed in the expansive path to recover the original dimensionality for outputs. [Ronneberger et al. (2015)].

Convolutional VAE To bridge the gap between the architectural differences of the two models, we implemented convolutional VAE as the baseline control. We used four convolutional layers and four deconvolutional layers in the encoder and decoder to mimic the process of an U-net. Similarly, we used the pytorch neural network with Adam optimizer [Paszke et al. (2019)] and no regularization to train the VAE. Specifically, we used under-complete VAE, which maps the input, (28x28), to 128-dimension space.

3.3 Results

See Table 3 for respective FID scores calculated on 64 generated samples.

MNIST See Figure 1 & 2 for sampled MNIST images from both models. Figure 3 & 4 show their respective loss plots.

CIFAR10 See Figure 5 & 6 for CIFAR10 images sampled from both models. Figure 7 & 8 show their respective loss plots.

3.4 Interpretation

3.4.1 MNIST

Looking at the loss plots (Fig. 3 & 4), both models achieved variational lower bound below 0.2. With both train and validation loss being low and close together, there were no obvious signs of overfitting or underfitting for both models on the graph. In terms of the FID score (Table 3), our implementation

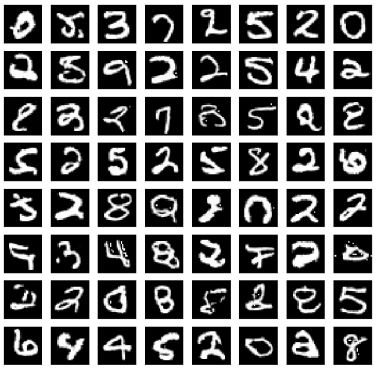

Figure 1: Sampled MNIST images from DDPM.

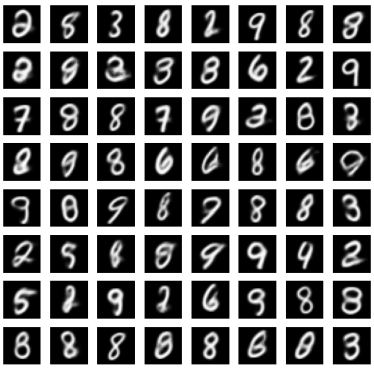

Figure 2: Sampled MNIST images from VAE.

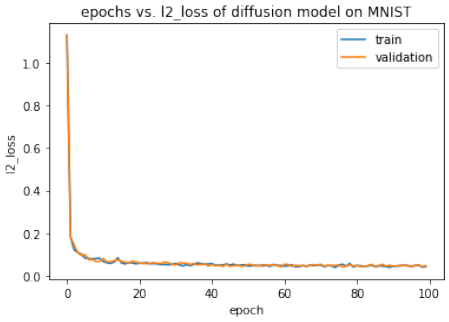

Figure 3: DDPM loss on MNIST

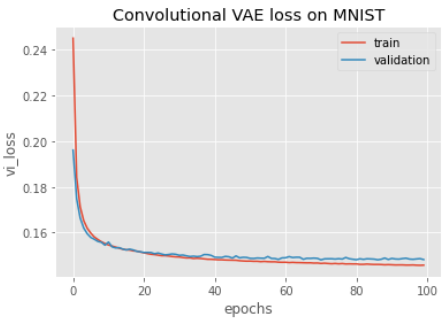

Figure 4: VAE loss on MNIST

Figure 5: Sampled CIFAR10 images from DDPM.

Figure 6: Sampled CIFAR10 images from VAE.

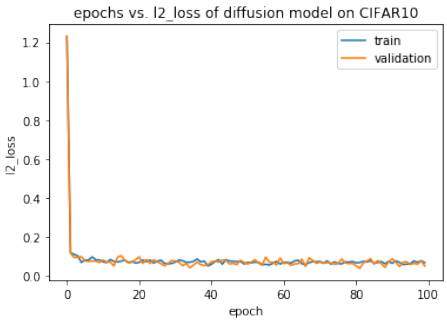

Figure 7: DDPM loss on CIFAR10

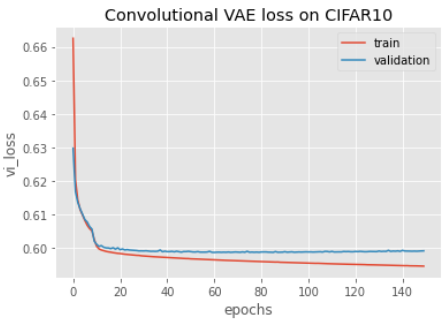

Figure 8: VAE loss on CIFAR10

of VAE achieved slightly higher score than DDPM on MNIST dataset. The better (lower) FID score for VAE was also reflected in the generated samples (Fig. 1 & 2). Digits generated by VAE appeared to be more consistent with respect to styles, sizes, etc; however, those generated by DDPM were less clear. For example, some digits look 'bolded' and distorted, and some did not resemble digits. Moreover, as described and shown in much of the literature, one flaw of VAE generated images is the blurriness [Zhao et al. (2017)]. Some blurriness is observed but digits were easily and clearly recognized. In summary, VAE performed better on image synthesis than DDPM on the MNIST data.

Generally, one of the reasons VAE does not perform well on image generation tasks is that one important hyperparameter in autoencoders is the bottleneck layer, or latent representation, of all under-complete AE. Sometimes pre-processing input with dimensionality reduction might help to recognize interesting underlying feature representations, but for datasets like MNIST, where each data example is relatively simple compared to others, there is inevitably information loss introduced to the original structure of the data, when input feature is mapped to lower-dimensional space. However, we mapped a 28×28 image to 128-dimensions, which is a fairly large space for MNIST. Additionally, we have convolutional layers that retain mutual information between the input and output. Based on the results, many latent structures are successfully captured by the model.

Additionally, one of the reasons for VAEs blurry images lies in the loss function: the KL divergence. There are two major terms in the ELBO for VAE, namely the reconstruction error and the KL divergence. The reconstruction error measures the difference between the input and output of decoder, and DDPM uses a similar concept of measuring the distance between the true and predicted output. However, instead of mapping input data to an encoding vector, VAE's encoding function maps input to a probability distribution and approximate posterior to a Gaussian distribution. It is very rare that natural images fall into a Gaussian distribution, thus the blurry characteristic of VAE is very persistent in many implementations.

Moreover, the higher FID score for DDPM (Table 3) could be seen in the lower sampled image quality in Fig. 1. While DDPM had a clear separation between the background black color and white color of the digits, we observed leftover noises in the images, while VAE did not appear to have this issue. This is because DDPM samples images by denoising images from a pure Gaussian noise, and it is reasonable to assume there were noises in the images that were not removed.

3.4.2 CIFAR-10

The results were reversed for the CIFAR-10 dataset. In terms of the loss plots (Fig. 7 & 8), DDPM was consistent with below 0.2 loss and no signs of over- and under-fitting. However, although the model had converged, the loss for VAE on CIFAR-10 was significantly higher, around 0.6, than those for DDPM. The FID score (Tab. 3) and the sampled images (Fig. 5 & 6) showed the same results. DDPM's FID score was much lower, and sampled images had identifiable classes, such as boats and birds. Specifically, there were many more details in DDPM sampled images, like the color transitions and defined object borders. In comparison, VAE lead to unrealistic, blurry images, where no animal or object could be identified.

We believed that the success of DDPM relied particularly on its ability to maintain input dimensions throughout the training process, especially pertaining to preserving and producing image details. With the same latent dimension (128), VAE was unable to compress all information in $32 \times 32 \times 3$ pixels onto a space that has a 96% reduction in dimensionality. Therefore, we could see some vague color patches in sampled images but none of the details. The same flaws discussed for MNIST also appeared for CIFAR-10. The assumption that images fall into a Gaussian prior distribution was even more unrealistic for CIFAR-10's natural images, which are more complicated than MNIST. Therefore, when dealing with data that has complex distributions, the sampling leads to blurry images.

Moreover, for DDPM, same as on the MNIST data, sampled images from CIFAR-10 suffer from unremoved noises. Although it was not be as obvious as in MNIST, there were 'neon color' noises in CIFAR-10 samples that stood out, because they completely did not integrate into the surroundings colors. We suspected that the leftovers noises in two datasets signalled an incompletion of the denoising process, and thus, a higher timestep T, should be tried in the future. Instead of 300 steps, it is justifiable to extend the Markov chains in both processes, to see if the denoising step can be 'completed' with no noises visible on the samples.

We also want to note that it is possible that the architectural differences between DDPM and VAE enhance differences in sampled image quality, in addition to the intrinsic model differences. Particularly, DDPM's U-net architecture reduced distortion by preserving the original structure of the image [Ronneberger et al. (2015)] and was much deeper than the implementation of Convolutional VAE in this project.

3.5 Conclusion

For CIFAR-10, DDPM outperformed VAE by a lower FID score, while for MNIST, VAE outperformed DDPM. If we were to draw an conclusion on whether the experimental results proved or disproved the proposed hypothesis, it is reasonable to say that the current DDPM implementation outperforms VAE on complex images but not on simpler gray-scale images.

4 Reflection and Outlook

Based on our results and as mentioned in the analysis, to improve DDPM, the simplest improvement is to try higher values for timestep T. T regulates the essential noising and denoising process in DDPM; increasing T will help prolong the sampling chains, and perhaps allow for more denoising. Another hyperparameter to vary is the variance schedule, where non-linear variances can be experimented with, such as cosine or quadratic. It is important that the variance schedule fits the specific data set that is being trained on.

Long term work centers around improving the model architecture for both DDPM and VAE, such as implementing a similar U-net architecture for VAE. In our view, making the architectural effects on model performance equal is more important than improving the architecture for one particular model.

One takeaway we have from our comparison of DDPM and VAE is that model choice for an image generation task should not default to the newer, 'fancier' model. In our experiments, VAE did better than DDPM on MNIST, which a simpler data set in comparison to CIFAR-10. Perhaps the type of image data to sample should be a factor in deciding which generative model should be used.

References

Altosaar, J. (2021). Tutorial - what is a variational autoencoder? *jaan.io*.

Dhariwal, P. and Nichol, A. (2021). Diffusion models beat gans on image synthesis. *CoRR*, abs/2105.05233.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2017). Gans trained by a two time-scale update rule converge to a local nash equilibrium.

Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models. *CoRR*, abs/2006.11239.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Technical report.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Rogge, N. and Rasul, K. (2022). The annotated diffusion model.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597.

Sohl-Dickstein, J., Weiss, E. A., Maheswaranathan, N., and Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. *CoRR*, abs/1503.03585.

Van Rossum, G. (2020). *The Python Library Reference, release 3.8.2*. Python Software Foundation.

Zhao, S., Song, J., and Ermon, S. (2017). Towards deeper understanding of variational autoencoding models. *CoRR*, abs/1702.08658.