Denoising Diffusion Probabilistic Models: Implementing diffusion models for image generation

Abstract

Diffusion probabilistic models are a part of a family of deep generative models that are used to synthesise high quality images. This paper details an attempt to implement this type of generative model to synthesise high quality images. This class of generative models were inspired by considerations of non equilibrium thermodynamics. The best results were obtained by ..... On the CIFAR10 data set I obtain scores of ..... The implementation has been submitted on learn ultra and will be available on GitHub soon.

1 Introduction

There are a number of different models that can be used for artificial image generation. Each model has different strength and weaknesses. Most of these models are able to yield high quality samples in a number of data modalities. Generative Adversarial Networks (GANs), Auto-regressive Models, Flows and Variational Auto-encoders (VAEs) have all demonstrated the ability to synthesise incredibly high quality image samples. In this paper I present an implementation of a Diffusion Probabilistic Model (also known as Diffusion models) based on the paper written by Jonathan Ho and colleagues in 2020 titled, "Denoising Diffusion Probabilistic Models". The aim of this implementation was to synthesise images of similar quality of the ones presented by Jonathan Ho but at lower resolutions of 128x128 or 94x94 on the STL10 Data set. A diffusion probabilistic model (diffusion model) is a parameterized Markov chain trained using variational inference to produce samples matching the data after finite time (Ho et al., 2020). By using this chain the aim is to teach a neural network to reverse the forward diffusion process that is iteratively applied to the images until they reach a perfect Gaussian distribution. This can be seen below in figure 1.

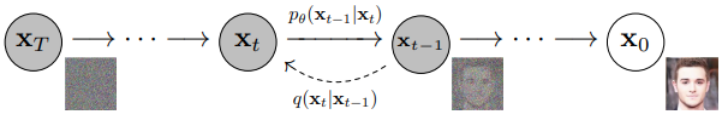

Figure 1: Figure taken from: Rogge and Rasul, 2022

The details about how this works will be discussed in the following section

2 Methodology

2.1 Theoretical Background

A diffusion model essentially is a machine learning model that has been taught to generate images by reversing the forward diffusion process q from an image that is pure Gaussian noise into a realistic image that is lies within a data distribution of realistic images of a certain type. q represents the forward diffusion process.

Let $q(x_0)$ be a data distribution of real images. By sampling images from this data distribution, we now have an image with which we can begin the forward diffusion process. We

have the approximated posterior $q(x_{1:t}|x_0)$ which we call the forward process. Defining the diffusion process $q(x_t|x_{t-1})$ where we obtain image x_t by adding noise to the previous image image x_{t-1} we get:

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

where a Gaussian distribution is essentially defined by two parameters, the mean and the variance μ and σ^2 respectively, where $\sigma^2 \geq 0$. The above states that each new slightly noiser image at some arbitrary time step t is drawn from a conditional Gaussian distribution with $\mu_t = \sqrt{1-\beta_t}x_{t-1}$ and ${\sigma_t}^2 = \beta_t$. This can be done by drawing from the normal distribution $\mathcal{N}(0, I)$ and then setting $x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon$. Now it's important to remember β_t aren't constant at each time step and are in fact defined by a variance schedule (linear or otherwise) and so after a finite number of time steps from 0 to T for images x_0 to x_T we get image x_T which is pure noise. Performing the reverse process requires us to teach the machine to approximate the conditional data distribution as it is intractable. This is because it is not feasible to know the distribution of the set of all possible images in out data distribution via calculations. However, even though it is intractable, we can still leverage a neural network in order to learn or rather estimate this conditional distribution. We therefore define this neural network and its parameters $p_\theta(x_{t-1}|x_t)$ where θ are the network parameters. So by using the neural network we aim to model the reverse process and therefore it must be able to represent a conditional probability distribution of the reverese process. Assuming the reverese process is Gaussian as well (since the forward process was Gaussian) then we know it is defined by the its mean and variance which are parameterized by μ_θ and Σ_θ respectively. therefore the process can be parameterized as follows:

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu(x_t, t), \Sigma(x_t, t)) \tag{2}$$

where the variance and mean are also conditioned on the noise at time step t. Similar to the paper written by Ho et al., 2020 I keep the variance fixed and only aim to let the neural network to learn to represent the mean μ_θ of the probability distribution. It is important to note we can allow the neural network to learn the variance however we fix it the implementation discussed in this paper. Due to the nature of the variational lower bound, which the authors of DDPM note that q and p_θ can be seen as VAEs, it is found that by construction of the forward and backward processes, each term of the loss is the KL divergence between 2 Gaussian distributions which can be written as an L2-loss with respect to the means (Ronneberger et al., 2015). Also since the sum of Gaussians is also Gaussian we can sample x_t at any time step. Which means we do not need to repeatedly apply the diffusion process for t times steps to obtain x_t. This is done below.

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

Where each a_t is just $1-\beta_t$ and so $\overline{a}_t$ is just the cumulative product of alphas a_0 to a_t better written as $\overline{a}_t := \prod_{s=1}^{t} a_s$. This allows us to sample the appropriate amount of Gaussian noise with respect to the time step and add it to the image without having to iteratively calculate (or refer to a look up table) the amount every time we change the number timesteps. As should be obvious, since we $\overline{a}_t$ is calculated from the variance schedule β_t it can be precomputed and stored. Furthermore, due to the nature of Gaussian noise and the reliance of the neural network on it for the forward and backward processes, we are able to perform what is known as the "reparamtization trick". This "trick" involves reparametrizing the mean in order to allow the neural network to lean and predict the added noise for each noise level. Therefore, our neural network essentially becomes a noise prediction neural network instead of a mean predictor. However we can still calculate the mean using the following method.

$$\mu_\theta(x_t, t) = 1/\sqrt{a_t}(x_t - \beta_t/\sqrt{1-\overline{a}_t}\epsilon_\theta(x_t, t)) \tag{4}$$

where ϵ_θ is just our neural network predicting the noise level at timestep t given image x_t. Therefore, our objective function for some arbitrary timestep t given $\epsilon \quad \mathcal{N}(0, I)$ looks like this:

$$||\epsilon - \epsilon_\theta(x_t, t)||^2 = ||\epsilon - \epsilon_\theta(\sqrt{\overline{a}_t}x_0 + \sqrt{(1-\overline{a}_t)}\epsilon, t)||^2 \tag{5}$$

Where x_0 is our initial image. ϵ is the pure noise sampled at each time step t which is given by the already defined forward process and $\epsilon_\theta(x_t, t)$ is our neural network which is trying to predict the noise level at time step t given image x_t. The neural network is optimised using a mean squared error between the true and the predicted Gaussian noise.

2.2 Training

The training loop runs as follows: 1. Sample x_0 from the data distribution $q(x_0)$ 2. Sample noise at level t uniformly between 1 and T 3. Sample some Gaussian distribution and corrupt the input by this noise at level t 4. Train the Neural network to predict this noise based on the corrupted image x_t

2.3 Architecture

Similar to the authors of DDPM I implement a U-Net architecture which was introduced by (Ronneberger et al., 2015) . This network learns only the most important information due to the bottleneck layer present in the middle. I also add residual connections to allow gradients to flow through the network and alleviate the vanishing gradient problem. In order to tell the network the time step (noise level) the current image is received at I employ Sinusoidal positions embeddings which was obtained from the (Rogge and Rasul, 2022) implementation and inspired by Vaswani et al., 2017. I also employ attention blocks to try get the model to pay attention to the most important information. and ignore things like tensor padding that makes sure images are of the same shape. The rest of the code used in the implementation is based on the theory already discussed therefore it will not be reiterated.

3 Results

After 2 different training sessions, 1 on the ffhq data set and the other on the STL10 data set the results were less than optimal which will be discussed in the Limitations section. The model was trained on 128 x 128 pixel images for the FFHQ data set and 96x96 for the STL10 data set

3.1 FFHQ

The FFHQ data set is a data set of 70000 realistic images of faces. An example of what the learned reverse process of the model produced.

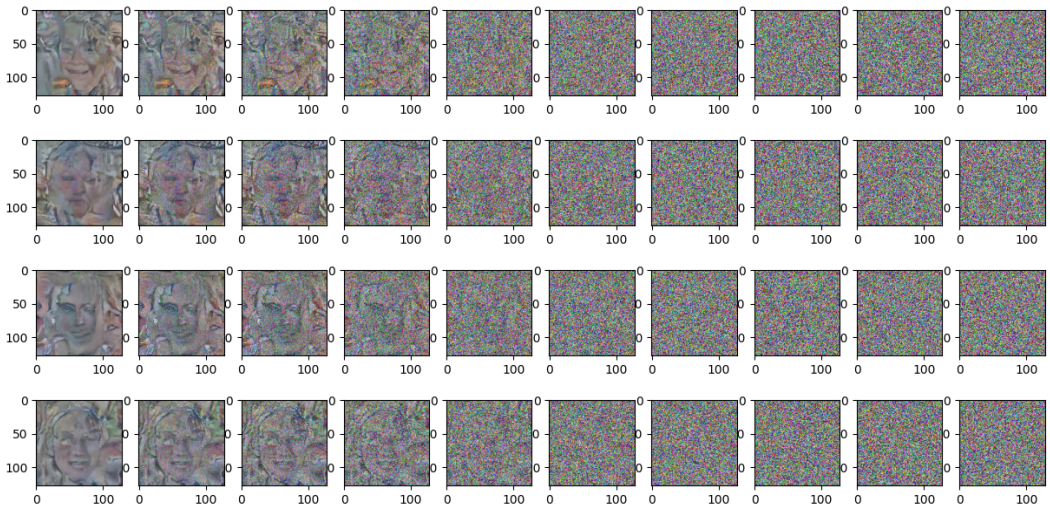

Figure 2: Examples of Images Generated during reverse diffusion process for the FFHQ data set

As we can see the images are very unclear with some facial features barely being present in the output

3.2 STL10

Unlike the previously shown results the STL10 data set seemed to learn more reasonable color mapping but failed to learn the appropriate shapes as shown by the example output below.

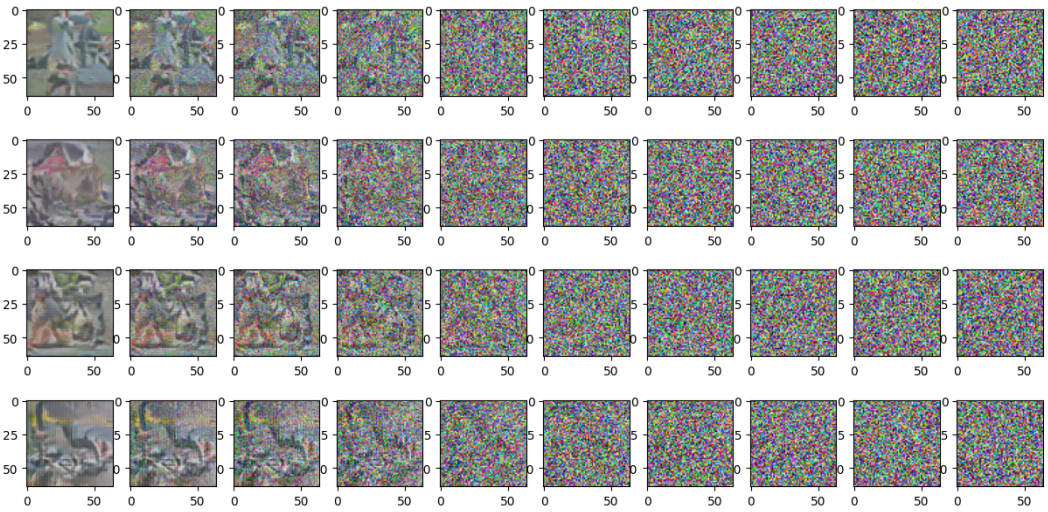

Figure 3: Examples of Images Generated during reverse diffusion process for the STL10 data set

4 Limitations

The results do not resemble realistic images. For the STL10 data set it is possible there was a problem with how i handled telling the neural network it was observing. For the FFHQ data set I did not download the full data set which may have meant that there were not enough data samples for the network to observe. Considering it was able to produce the core features of what resembles a face there is a high likelihood that slight changes to some of the architecture and an increased training time with a larger data set size would yield better results. I used a linear beta scheduler which might have destroyed image signals too quickly which might have contributed to the issues faced in the FFHQ results. Lastly the model might have been too simplistic to capture the complex details of the data distribution. There much room for improvement and further investigation into what went wrong in the training process.

Bonuses

This submission has a total bonus of +8 as the model was trained on the 128 x 128 FFHQ data set (of course it was also trained on the STL10 94x94 in order to check where the weird color artefacts on the generated face were coming from) .

References

Ho, J., Jain, A., & Abbeel, P. (2020). Denoising diffusion probabilistic models. *CoRR*, *abs/2006.11239*. https://arxiv.org/abs/2006.11239

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

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

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. *CoRR*, *abs/1706.03762*. http://arxiv.org/abs/1706.03762