Manga character generation with Denoising Diffusion Probabilistic Models (DDPM)

Chibuzor John Amadi[502623] & Kristian Perriu[505571]

University of Pavia, Pavia

University of Milano-Biccoca, Milan

University of Milano-Statale, Milan

1. Abstract

Denoising Diffusion Probabilistic Models (DDPM) [1], are a class of generative models used to produce high-quality data such as images. Such models are used in state-of-the-art models like DALL-E 2 [2], SORA [3], Imagen [4] and Stable Diffusion [5]. They work by reversing a diffusion process. Such process introduces noise in an incremental fashion to the data until the said data becomes pure noise. The model learns to reverse such process, thus gradually denoising the data and reconstructing the original given distribution.

These models have shows impressive performance in the task of generating high quality images and diverse data, thus making them a powerful tool in the field of image generation. Such models, due to their great performance, come with the cost of requiring multiple steps through a deep neural network and the generative process which are very expensive.

2. Introduction

2.1 Dataset

The dataset consists of 12000 images each scaled down to 64x64 pixels and it contains diverse pictures of anime/manga characters. Since the dataset contains only one folder with all the images, the class ImageFolder [6] is sufficient for such task. Please find more information about the dataset here.

2.2 Preprocessing

The preprocessing for such task was kept simple. At first the images were scaled down to 32x32 pixels to reduce the computational cost. Images were then subject to a random horizontal flip to help the model generalize better and then they were normalized. Images then were turned to tensors.

3. Methodology

The model used for such task is U-NET [7], a predefined model. This model was used in combination with the training and sampling algorithms to produce the images for such task. The only things subject to change will be the said algorithms.

3.1 U-NET

U-Net architecture known for its encoder-decoder structure, which includes skip connections between corresponding layers in the encoder and decoder paths. This design helps in capturing both high-level and detailed features.

The input noisy image is put through a series of convolutional layers, progressively downsampling it to capture more abstract features. At its deepest part which is the bottleneck, the most abstract features are captured and then they are passed through the decoding part. The decoder on the other hand, unsamples the noisy pictures and reconstructs the noisy image. This part of the architecture also has skip connections which make it possible for the model to preserve details and to retain important information that could be lost.

For this task, at each step of the diffusion process the model learns to predict the noise component and this predicted noise is removed from the image. Its skip connections contribute for such thing as they capture both global and local context in the pictures, thus improving the quality of the denoised images.

3.2 Training algorithm

The training algorithm for this task is fairly simple. A batch of training samples is taken and random time indices are taken. With these, noisy samples are generated which are then taken by the network to estimate the noise and compare the estimated noise it generated to the actual noise. The derivation of the loss is done by the formula below:

$$||\epsilon - \epsilon_\theta(x_t, t)||^2$$

The training algorithm itself works by sampling data from the true data distribution and then it also uniformly samples a timestep from the total number of timesteps T. Noise is then sampled from a standard normal distribution and then a gradient descent step is performed according to the formula:

$$\nabla_\theta ||\epsilon - \epsilon_\theta\left(\sqrt{\overline{\alpha}_t}x_0 + \sqrt{1-\overline{\alpha}_t}\epsilon, t\right)||^2$$

And such thing is repeated until the model converges.

For such task, unlike the l2 loss [8] used in the paper, l1 loss [9] was used as it was proved to perform better.

3.3 Sampling algorithm

In order to generate the new images, the sampling algorithm has to be applied. We now start from x_T sampled from the Gaussian distribution [10] above and sample back to x_0.

The sampling algorithm works by first starting with the sample of pure noise and it then iterates through the timesteps. It now updates the image, or more formally to calculate x_{T-1}, in order to remove the noise step by step by using the formula:

$$x_{T-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\overline{\alpha}_t}}\boldsymbol{z_\theta}(x_t, t)\right) + \sigma_t \boldsymbol{z}$$

Where z is the estimated noise for the full process, σ_t is used to scale it and x_t is the image at timestep t. this backwards process also requires a separate variance schedule σ_t^2 and for such thing we will be using:

$$\sigma_t^2 = \frac{1-\overline{\alpha}_{t-1}}{1-\overline{\alpha}_t}\beta_t$$

Such process, like the training algorithm will continue until we reach to x_0 and the final image will have no noise.

4. Evaluation

4.1 Experimental details

For such experiment, modifications were done. At first, the time steps were increased and decreased. The time steps that we experimented with were 100, 300 and 500. For each of these timesteps further experiments were carried in the sampling stage to see how the generated images would change. For the mentioned timestamps, the sampling algorithm was modified in such a way that at first x_t was deactivated and then the noise was deactivated to see how the output would be. All models were trained for 20 epochs.

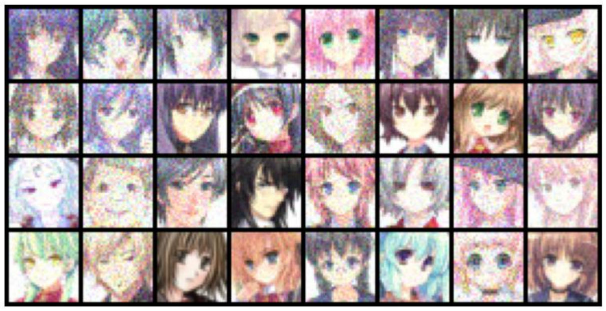
Figure 1, Model result with 100 time steps.

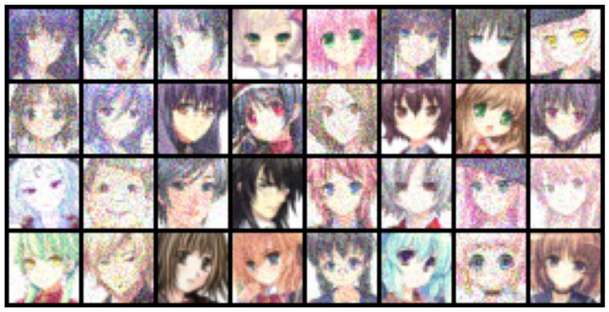
Figure 2, Model results with 100 time steps, no noise

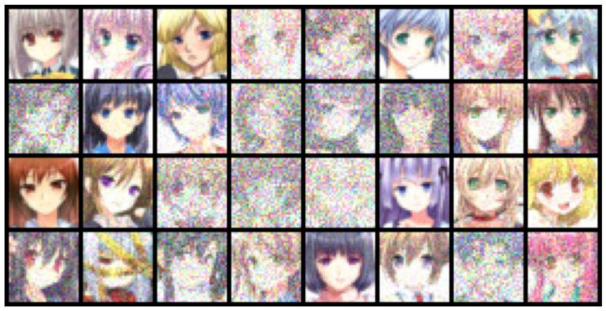
Figure 3, Model results with 200 time steps

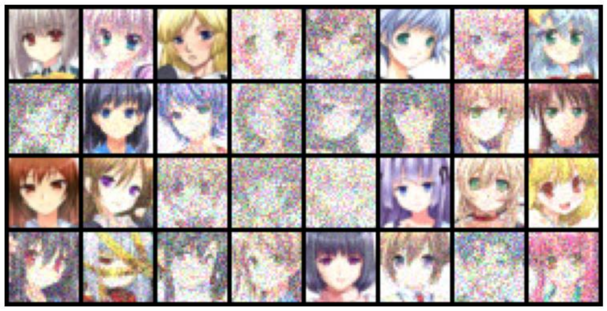
Figure 4, Model results with 200 time steps, no noise

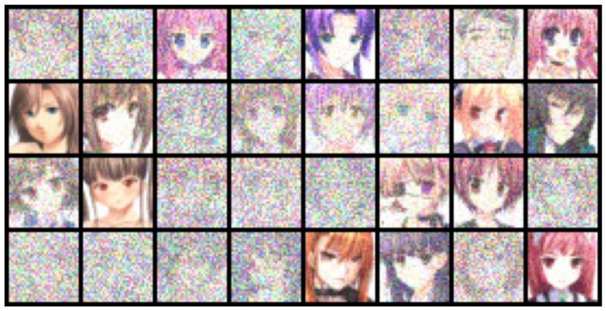
Figure 5, Model results with 500 time steps

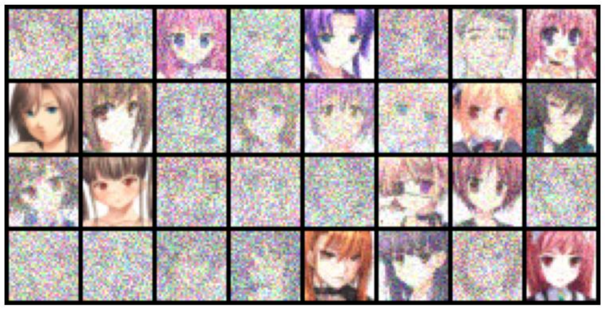
Figure 6, Model results with 600 time steps, no noise

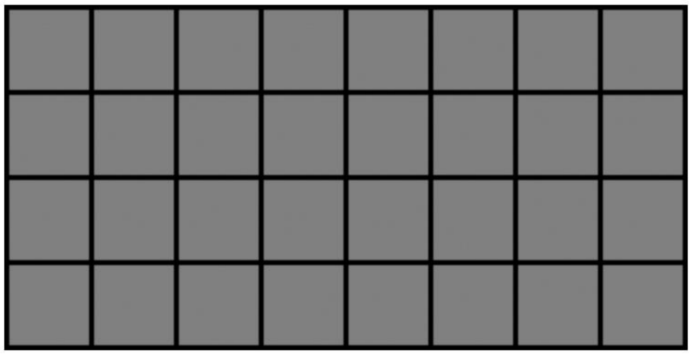

Figure 1, Model results without x_t

4.2 Result analysis

While studying the training proceedure, it was noticed that the more the time steps increased the faster the loss reduced. Not only that but the loss also started at a smaller value.

In figures 1 and 2 we have the results of the model with 100 time steps and with and without the noise. The pictures are fairly clear and they have no big difference apart from the fact that the pictures when noise was present are slightly birghter than the ones without the noise. The pictures also have slightly less noise when the noise is removed due to the formula now returning only the updated pictures.

The same phenomena described above can also be seen even when the time steps increased, in pictures 3, 4, 5 and 6 respectively. The only difference we can see from one of the time steps to the other is that the more the time steps increase the more noise is added to the pictures. When reaching 500 times steps such thing very visible and some pictures have so much noise that they are impossible to distinguish.

In figure 7, the output of the models when x_t was absent was the same. This is because the formula had no picture to update and thus it produced images with nothing on them.

5. Concluision

As proved by the pictures, the best results were geenrated from the model with 100 timesteps. Such thing can be contributed to the parameters which can be assumed to be better optimized for small time steps and also it could be that the initial conditions for the said model were better, meaning that the model started with less noise than the other models.

The very visible noise when the time steps were increased was a result of the model pronouncing a lot of noise due to the high number of steps but also failing to reduce such noise as the denoising process wasn't sufficiently robbust at each step. Such thing can also be due to the parameters mentioned above and also to the fact that the model itself was not complex enough to handle a high number of time steps.

Another reason for such performance on high time steps can also contributed to the errors in denoising predictions accumulating over a high number of steps and thus resolting in a very noisy output.

However, taking into account the simple preprocessing done and the small number of epochs the models performed fairly well. Another contributor to the poor performance was also the simplicity of the model as models made to handle such tasks have far more parameters than the one used in this report which has 2.4 million parameters, who proved to not be enough.

A possible approach to resolve the issue of a lot of noise when the time steps increase is to also optimize the variables for the sampling algorithm so that they can properly handle high time step numbers. For its simplicity, this model can be considered sufficient for the task at hand.

6. References

[1]	DDPM	https://arxiv.org/pdf/2006.11239
[2]	DALL-E 2	https://openai.com/index/dall-e-2/
[3]	SORA	https://openai.com/index/sora/
[4]	Imagen	https://imagen-ai.com/
[5]	Stable Diffusion	https://stability.ai/
[6]	ImageFolder	https://pytorch.org/vision/main/generated/torchvision.datasets.ImageFolder.html
[7]	U-Net	https://arxiv.org/abs/1505.04597
[8, 9]	L2 loss, L1 loss	https://amitshekhar.me/blog/l1-and-l2-loss-functions
[10]	Gaussian distribution	https://en.wikipedia.org/wiki/Normal_distribution