An Implementation of Denoising Diffusion Probabilistic Models

Yug Ajmera
University of Pennsylvania
yajmera@seas.upenn.edu

Athrva Pandhare
University of Pennsylvania
athrva@seas.upenn.edu

Abstract

In this project, we attempt to implement a Denoising Diffusion Probabilistic Model (DPPM) from scratch. We first develop the analysis required for the implementation of the DDPM. Subsequently, we show the results of the developed DDPM model on the FashionMNIST dataset. Finally, we show results on a more complicated dataset (Stanford Cars Dataset [1]). The purpose of this report is to build the theory of DDPMs and motivate their importance in generative tasks from the ground up and show results obtained from the implementation of the DDPMs. In addition to the implementation of the DDPM models, we also evaluate the model performance with different β schedules and different loss formulations for computing the error between the predicted noise and the actual noise at each time step.

1. Approach

Diffusion models are a new class of state-of-the-art generative models that generate diverse high-resolution images. There are already a bunch of different diffusion models that includes Open AI's DALL-E 2 and GLIDE, Google's Imagen, and Stability AI's Stable Diffusion. In this project, we will focus on the most prominent one, which is the Denoising Diffusion Probabilistic Models (DDPM) as initialized by Sohl-Dickstein et al [2] in 2015 and then improved by Ho. et al [3] in 2020.

The basic idea behind diffusion models is rather simple. It takes an input image $\mathbf{x}_0$ and gradually adds Gaussian noise to it through a series of T time steps. We will call this the forward process. A network is then trained to recover the original image by reversing the noising process. By being able to model the reverse process, we can start from random noise and denoise it step-by-step to generate new data.

1.1. Forward Diffusion Process

Consider an image $\mathbf{x}_0$ sampled from the real data distribution (or the training set). The subscript denotes the number of time step. The forward process denoted by q is modeled as a Markov chain, where the distribution at a particular time step depends only on the sample from the previous step. The distribution of corrupted samples can be written as,

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \tag{1}$$

At each step of the Markov chain we add Gaussian noise to $\mathbf{x}_{t-1}$ producing a new latent variable $\mathbf{x}_t$. The transition distribution forms a unimodal diagonal Gaussian as,

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t;\ \mu_t = \sqrt{1-\beta_t}\,\mathbf{x}_{t-1},\ \Sigma_t = \beta_t\mathbf{I}) \tag{2}$$

where β_t is the variance of Gaussian at a time step t. It is hyperparameter that follows a fixed schedule such that it increases with time and belongs in range $[0, 1]$. Ho et al. sets a linear schedule for the noise starting from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$, and $T = 1000$.

A latent variable $\mathbf{x}_t$ can be sampled from the distribution $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ by using the reparameterization trick is as,

$$\mathbf{x}_t = \sqrt{1-\beta_t}\,\mathbf{x}_{t-1} + \sqrt{\beta_t}\,\epsilon_t \tag{3}$$

where $\epsilon_t \sim \mathcal{N}(0, 1)$.

Equation 3 shows that we need to compute all the previous samples $\mathbf{x}_{t-1}, ..., \mathbf{x}_0$ in order to obtain $\mathbf{x}_t$, making it expensive. To solve this problem, we define,

$$\alpha_t = (1-\beta_t), \qquad \bar{\alpha_t} = \prod_{s=0}^{T} \alpha_s$$

and rewrite equation 3 in a recursive manner,

$$
\begin{aligned}
\mathbf{x}_t &= \sqrt{\alpha_t}\,\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\,\epsilon_t \\
&= \sqrt{\alpha_t}\,\left[\sqrt{\alpha_{t-1}}\,\mathbf{x}_{t-2} + \sqrt{1-\alpha_{t-1}}\,\epsilon_t\right] + \sqrt{1-\alpha_t}\,\epsilon_t \\
&= \sqrt{\alpha_t\alpha_{t-1}}\,\mathbf{x}_{t-2} + \sqrt{(\alpha_t)(1-\alpha_{t-1}) + (1-\alpha_t)}\,\epsilon_t \\
&= \sqrt{\alpha_t\alpha_{t-1}}\,\mathbf{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\,\epsilon_t \\
&\dots \\
&= \sqrt{\alpha_t\alpha_{t-1}\,\dots\,\alpha_0}\,\mathbf{x}_0 + \sqrt{1-\alpha_t\alpha_{t-1}\,\dots\,\alpha_0}\,\epsilon_t \\
&= \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\epsilon_t
\end{aligned} \tag{4}
$$

The close-form sampling at any arbitrary timestep can be carried out using the following distribution,

$$\mathbf{x}_t \sim q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t;\ \mu_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0,\ \Sigma_t = (1-\bar{\alpha}_t)\mathbf{I}) \tag{5}$$

Since β_t is a hyperparameter that is fixed beforehand, we can precompute α_t and $\bar{\alpha}_t$ for all timesteps and use Equation 4 to sample the latent variable $\mathbf{x}_t$ in one go.

1.2. Reverse Diffusion Process

As $T \rightarrow \infty$, $\bar{\alpha}_t \rightarrow 0$, the distribution $q(\mathbf{x}_T|\mathbf{x}_0) \approx \mathcal{N}(0,\mathbf{I})$ (also called isotropic Gaussian distribution), losing all information about the original sample. Therefore if we manage to learn the reverse distribution, we can sample $\mathbf{x}_T \sim \mathcal{N}(0,\mathbf{I})$, and run the denoising process step-wise to generate a new sample.

With a small enough step size ($\beta_t \ll 1$), the reverse process has the same functional form as the forward process. Therefore, the reverse distribution can also be modeled as a unimodal diagonal Gaussian. Unfortunately, it is not straightforward to estimate $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$, as it needs to use the entire dataset (It's intractable since it requires knowing the distribution of all possible images in order to calculate this conditional probability).

Hence, we use a network to learn this Gaussian by parameterizing the mean and variance,

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1};\ \mu_\theta(\mathbf{x}_t,t),\ \Sigma_\theta(\mathbf{x}_t,t)) \tag{6}$$

Apart from the latent sample $\mathbf{x}_t$, the model also takes time step t as input. Different time steps are associated with different noise levels, and the model learns to undo these individually.

Like the forward process, the reverse process can also set up as a Markov chain. We can write the joint probability of the sequence of samples as,

$$p_\theta(x_{0:T}) = p(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \tag{7}$$

Here, $p(\mathbf{x}_T) = \mathcal{N}(0,\mathbf{I})$ as we start training with a sample from pure noise distribution.

1.3. Loss Function

The forward process is fixed and its the reverse process that we soley focus on learning. Diffusion models can be seen as latent variable models, and are similar to variational autoencoders (VAEs), where $\mathbf{x}_0$ is an observed variable and $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T$ are latent variables.

Maximizing the variational lower bound (or also called evidence lower bound ELBO) on the marginal log-likelihood forms the objective in VAEs. For an observed variable x and latent variable z, this lower bound can be written as,

$$\log p_\theta(x) \geq \mathbf{E}_q[\log p_\theta(x|z)] - \mathbf{D}_{KL}\left(q(z|x)\ ||\ p_\theta(z)\right)$$

Rewriting it in the diffusion model framework and simplifying we get,

$$
\begin{aligned}
\text{ELBO} = \mathbf{E}_q\left[\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)\right] - \mathbf{D}_{KL}(q(\mathbf{x}_T|\mathbf{x}_0)\ ||\ p(\mathbf{x}_T)) \\
- \sum_{t>1}\mathbf{D}_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)\ ||\ p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))
\end{aligned}
$$

The objective of maximizing this lower bound is equivalent to minimizing a loss function that is its negation,

$$
\begin{aligned}
L = &\underbrace{\mathbf{D}_{KL}(q(\mathbf{x}_T|\mathbf{x}_0)\ ||\ p(\mathbf{x}_T))}_{L_T} + \\
&\sum_{t>1}\underbrace{\mathbf{D}_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)\ ||\ p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}}\underbrace{-\mathbf{E}_q\left[\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)\right]}_{L_0}
\end{aligned} \tag{8}
$$

The term L_T has no trainable parameters so it's ignored during training, furthermore, as we have assumed a large enough T such that the final distribution is Gaussian, this term effectively becomes zero. L_0 can be interpreted as a reconstruction term (similar to VAE).

The term L_{t-1} formulates the difference between the predicted denoising steps $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ and the approximated steps $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ (which is given as a target to the model). It is explicitly conditioned on the original sample $\mathbf{x}_0$ in the loss function so that the distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)$ takes the form of Gaussian.

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1};\ \tilde{\mu}_t(\mathbf{x}_t,\mathbf{x}_0)\ \tilde{\beta}_t)$$

Such that,

$$
\begin{aligned}
\tilde{\beta}_t &= \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\ \beta_t \\
\tilde{\mu}_t(\mathbf{x}_t,\mathbf{x}_0) &= \frac{\sqrt{\alpha_t}\ (1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\beta_t\ \sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_t}\ \mathbf{x}_0
\end{aligned} \tag{9}
$$

Equation 4 $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\ \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\ \epsilon_t$, we can be rewrite it as, $\mathbf{x}_0 = \frac{\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\ \epsilon_t}{\sqrt{\bar{\alpha}_t}}$. Substituting it in the above mean value we would obtain,

$$\tilde{\mu}_t = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\ \epsilon_t \right) \tag{10}$$

Recall that we learn a neural network that predicts the mean and diagonal variance of the Gaussian distribution of the reverse process. Ho et al. decided to keep the predicted variances fixed to time-dependent constants because they found that learning them leads to unstable training and poorer sample quality. They set $\Sigma_\theta(\mathbf{x}_t, t) = \sigma_t^2\ \mathbf{I}$, where $\sigma_t^2 = \beta_t$ or $\tilde{\beta}_t$ (both gave same results).

Because $\mathbf{x}_t$ is available as input at training time, instead of predicting the mean (equation 10), we make it predict the noise term ϵ_t using ϵ_θ. We can then write the predicted mean as,

$$\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\ \epsilon_\theta(\mathbf{x}_t, t) \right) \tag{11}$$

and the predicted de-noised sample can be written using the reparameterization trick as,

$$\begin{aligned} \mathbf{x}_{t-1} &= \mu_\theta(\mathbf{x}_t, t) + \sqrt{\Sigma_\theta(\mathbf{x}_t, t)}\ z_t \\ &= \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\ \epsilon_\theta(\mathbf{x}_t, t) \right) + \sigma_t z_t \end{aligned} \tag{12}$$

where $z \sim \mathcal{N}(0, 1)$ at each time step.

Simplifying the loss function with the above considerations,

$$\begin{aligned} L_{t-1} &= \mathbf{D}_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \ ||\ p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) \\ &= \frac{1}{2\sigma_t^2} ||\tilde{\mu}_t - \mu_\theta(\mathbf{x}_t, t)||^2 \\ &= \frac{\beta_t^2}{2\sigma_t^2\ \alpha_t\ (1-\bar{\alpha}_t)} ||\epsilon_t - \epsilon_\theta(\mathbf{x}_t, t)||^2 \end{aligned} \tag{13}$$

The objective reduces to a weighted L2-loss between the noises and second term of the loss function becomes, $\mathbf{E}_q[L_{t-1}] = \mathbf{E}_{\mathbf{x}_t, \epsilon_t}[w_t\ ||\epsilon_t - \epsilon_\theta(\mathbf{x}_t, t)||^2]$

Empirically, Ho et al. found that training the diffusion model works better with a simplified objective that ignores the weighting term in L_{t-1}. They also got rid of the term L_0 by altering the sampling method, such that at the end of sampling (t = 1), we obtain $\mathbf{x}_0 = \mu_\theta(\mathbf{x}_0, t = 1)$

The loss function for DDPM is given as,

$$\begin{aligned} L_{simple} &= \mathbf{E}_{\mathbf{x}_t, \epsilon_t}\left[||\epsilon_t - \epsilon_\theta(\mathbf{x}_t, t)||^2\right] && (14.1) \\ &= \mathbf{E}_{\mathbf{x}_0, \epsilon_t}\left[||\epsilon_t - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\ \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\ \epsilon_t, t)||^2\right] && (14.2) \end{aligned}$$

For full derivation please use this link: Full-Derivation.

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \mathrm{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$
6: **until** converged

Figure 1. The training algorithm of a standard DDPM as mentioned in [3]

Once the DDPM model is trained, the sampling algorithm is quite straightforward.

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Figure 2. Sampling from the latent distribution of a DDPM as mentioned in [3]

2. Novelty

In addition to implementing a vanilla DDPPM, we also tried out the following different configurations to achieve better results.

- **Varying the β schedules :**
We implemented the linear, cosine, quadratic and sigmoid schedules for varying the β schedule over the time steps. The linear schedule varies the variance linearly starting from $10e-4$ to 0.02, and this is the schedule used in the original DDPM paper. The same range is used for the quadratic and sigmoid functions. The cosine schedule introduced in [4] varies the variance with a cosine function.

- **Varying the measures for the loss :**
The final loss function obtained from Eq. (14.2) is the L2 loss between the predicted noise and the actual

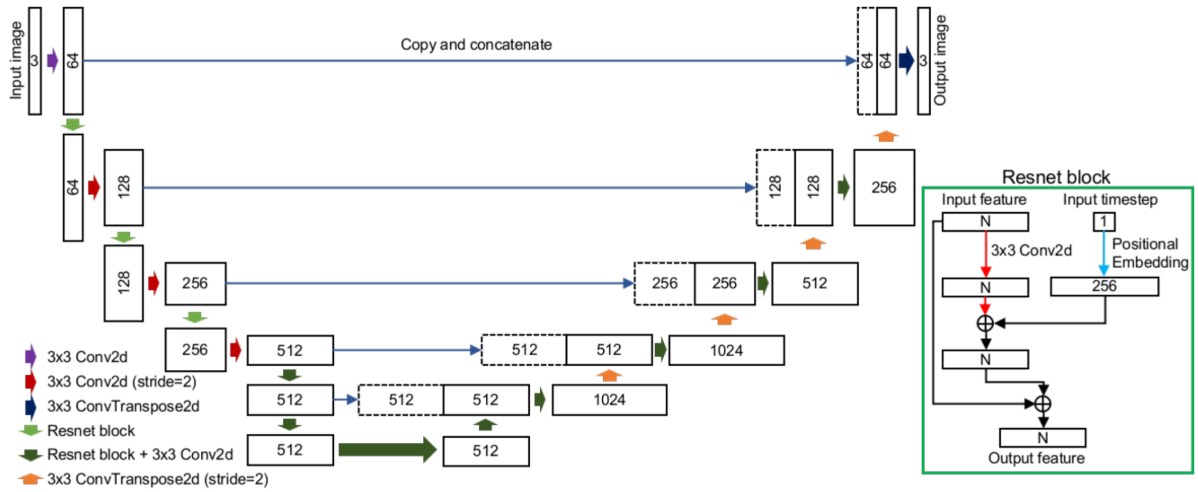

Figure 3. Full Conditional Unet Architecture

noise. With the L2 loss metric, we also evaluate L1 loss and Smooth L1 between these quantities.

$$L_{1;smooth} = \begin{cases} |x| & \text{if } |x| > \alpha; \\ \frac{1}{|\alpha|}x^2 & \text{if } |x| \leq \alpha \end{cases}$$

3. Architecture Details

To represent the reverse process, the authors used a U-Net backbone [5] similar to an unmasked PixelCNN++ [6] with group normalization throughout. Parameters are shared across time, which is specified to the network using the Transformer sinusoidal position embedding [7]. They also used self-attention at the 16×16 feature map resolution. The overall architecture is called the conditional UNet backbone and is shown in Figure 3.

The purpose of the network is to take in a batch of noisy images and their corresponding noise levels and predict the noise added to the input at that time step. More specifically, the network accepts a batch of noisy images and a batch of noise levels as input and produces a denoised representation of the input as it's output. Essentially, the network is constructed in the following manner: a convolutional layer is applied to the batch of noisy images and position embeddings are computed for the noise levels, followed by a series of downsampling stages that include ResNet blocks, group normalization, attention, a residual connection, with a downsampling operation. In the network bottleneck, additional ResNet blocks are applied and are combined with attention. Then, a series of upsampling stages are applied, each consisting of 2 ResNet blocks, group normalization, attention, a residual connection, and an upsample operation. Finally, a ResNet block and a convolutional layer are applied.

3.1. Position Embedding

The authors use sinusoidal position embeddings to encode t the time (or noise level) for each image in a batch in the neural network inspired by [7], which has shared parameters across time. This allows the network to "know" at which specific time step (noise level) it is operating for each image in a batch.

The Sinusoidal Positional Embedding module takes a tensor of noise levels for a batch of noisy images as input and outputs a tensor of position embeddings with a specified dimensionality. These position embeddings are then added to each block in the network. The embeddings are calculated in a heuristic manner as given in [7].

3.2. ResNet/ConvNeXT block

This is the core building block of the U-Net model. The DDPM authors employed a Wide ResNet block and we have also used the same one.

3.3. Attention Module

Attention is the building block of the famous Transformer architecture [7], which has shown great success in various domains of AI, NLP, and vision. We have used the linear attention variant whose time and memory requirements scale linearly in the sequence length, as opposed to quadratic for regular attention.

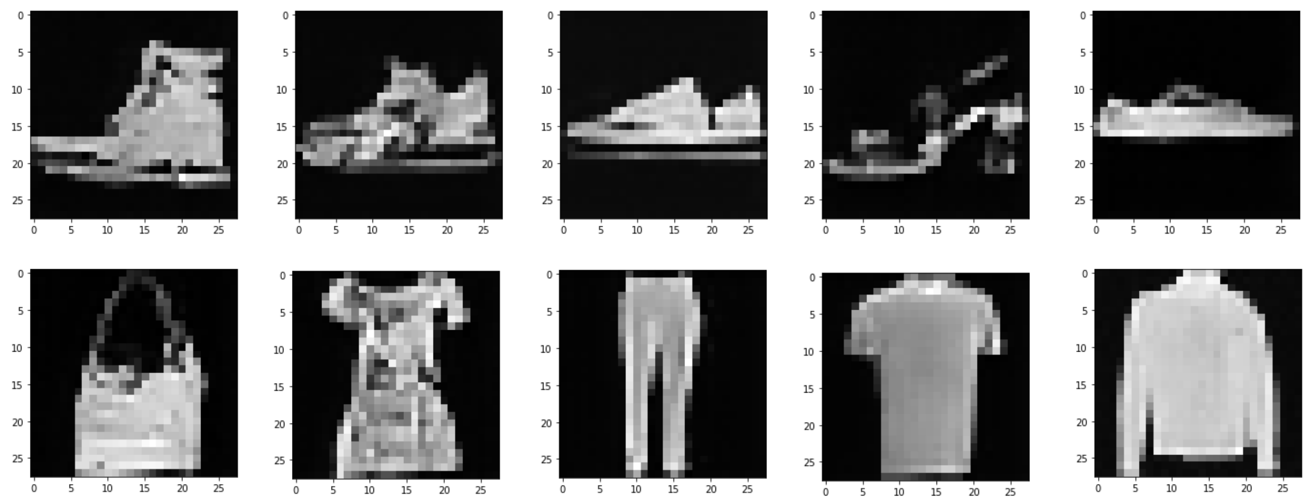

Figure 4. Generated Images using a smaller model on the FashionMNIST Dataset. It is clear that the produced images have sufficient visual fidelity.

3.4. Group Normalization

Another special operation that the authors of [3] use is Group Normalization. They incorporate group normalization (as in [8]) into the U-Net by inserting it between the convolutional and attention layers. They use Pre-Group Normalization to apply group normalization before the attention layer.

4. Results

4.1. Experiments

We trained our DDPM model on the MNIST Fashion Dataset using various schedules and losses. Adam optimizer was used with a learning rate of 0.001. The learning converged after 15 epochs and we used the Inception score (IS) for the evaluation of each of the models.

IS score measures the quality and diversity of the input image set. It is one of the most widely adopted scores for generative model evaluation [9]. It uses Inception network V3 pre-trained on ImageNet to predict the class labels represented as a likelihood $P(y|x_i)$ for generated image x_i. IS measures the average KL divergence between the conditional label distribution $p(y|x)$ of images (expected to have low entropy for easily classifiable images since their class is highly predictable. This means good image quality) and the marginal distribution $p(y)$ obtained from all the samples (expected to have high entropy if all classes are equally represented in the set of images that means high diversity). In general, a high IS score indicates good performance in photo-realistic and diversity.

Table 1 shows the results of our experiment. The trends show that the Quadratic schedule works best among all the schedules, given any loss function. Furthermore, smooth L1 works best among the loss functions we employed. This can be reasoned intuitively as combines the advantages of L1-loss (steady gradients for large values of loss) and L2-loss (less oscillations during updates when that loss is small).

Schedule	**Loss**	**Inception Score (IS)**
Linear	L1	3.3707
Cosine	L1	1.614
Quadratic	L1	3.6521
Sigmoid	L1	3.304
Linear	L2	3.3808
Cosine	L2	2.6078
Quadratic	L2	3.4592
Sigmoid	L2	3.3833
Linear	Smooth-L1	3.4503
Cosine	Smooth-L1	2.1523
Quadratic	Smooth-L1	3.878
Sigmoid	Smooth-L1	3.6121

Table 1. Results of the Ablation Study

It was also noted from experiments that by training even further (for 20 epochs), our models can achieve even higher IS scores, as high as the real test set. This is our best model and the results are shown in the table below 2.

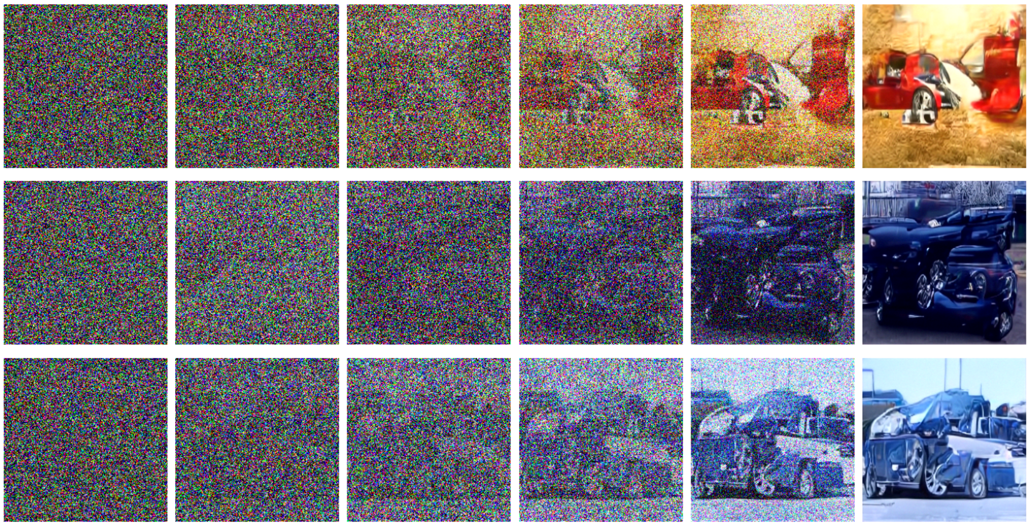

Figure 5. Results of training the DDPM model on the Stanford Cars dataset [1].

Model	Inception Score (IS)
Real Test Set	4.0885
Huber + Quadratic (20 epochs)	4.1842

Table 2. Results with test set

Figure 4 shows a qualitative sample of the produced images. Note that these images were generated at the end of the denoising schedule. For conciseness, we only include the final images.

5. Extending the Diffusion Model to a Complex Dataset

We extended the developed DDPM to the Stanford Cars dataset [1]. We used the same UNet architecture that was used for the FashionMNIST dataset. We increased the number of parameters by increasing the number of channels per convolutional layer. The images in the Stanford Cars dataset are large and had to be downsampled to $(224, 224, 3)$ for training. Furthermore, we used a linear schedule for β and the L_2 loss between the predicted noise and the true noise just as in the original DDPM paper. The results obtained in this experiment are shown in Fig. 5.

6. Conclusions and Future Work

In conclusion, we implemented a DDPM model for image generation from scratch. Our implementation was heavily based on the original papers, but we did a thorough analysis of the effects of changing the variance schedules and the metric for calculation of the distance between the predicted noise and the true noise. From our analysis, we saw that the smooth-L_1 loss coupled along with a quadratic schedule for β produced the best results. Finally, we extended the DDPM to a larger and more complicated dataset (viz. Stanford cars dataset) and qualitatively evaluated the results. It was seen that the produced images were of good quality (devoid of speckled/ white noise), but the model did not capture the underlying distribution very well. We suspect that this can be improved by either training a larger model, or implementing a Latent diffusion model.

References

[1] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013. 1, 6

[2] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *CoRR*, abs/1503.03585, 2015. 1

[3] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *arXiv preprint arxiv:2006.11239*, 2020. 1, 3, 5

[4] Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis. *CoRR*, abs/2105.05233, 2021. 3

[5] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 4

[6] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017. 4

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 4

[8] Yuxin Wu and Kaiming He. Group normalization. *CoRR*, abs/1803.08494, 2018. 5

[9] Ali Borji. Pros and cons of gan evaluation measures. *Computer Vision and Image Understanding*, 179:41–65, 2019. 5