Tiny Image Generation

Viraj Chhajed Raayan Dhar Johnathan Harding William Zhou

Abstract

This paper presents a study of small image size generative modeling using diffusion models and PixelCNN. Our experiments are focused on two 'tiny image' datasets, optimized for our computational constraints. By analyzing the interpolation in our diffusion models, investigating various beta and noise schedules, and evaluating the denoising capabilities at different sampling steps, we provide novel insights into the mechanisms and effectiveness of each model type.

We measure the quality of generated images using FID scores and validate model performance through qualitative and quantitative assessments. The outcomes of this study illuminate the strengths and limitations of each approach, with particular attention to the adaptation of diffusion models for small image sizes. The results have promising implications for the application of these models in domains where computational efficiency is paramount.

1. Introduction

Historically, the post-CNN image generation paradigm has developed into a branching field spanning many model types, including PixelCNNs, Generative Adversarial models, diffusion models, score-based models, and normalizing-flow based models. In this paper, we explore the performance of Pixel Convolutional Neural Networks (PixelCNNs) [23], Denoising Diffusion Probabilistic Models (DDPMs) [8], and Denoising Diffusion Implicit Models (DDIMs) [20] for image generation tasks.

PixelCNNs, as a form of autoregressive model, capitalize on the spatial hierarchical structure of images to predict pixel values sequentially. On the other hand, DDPMs and DDIMs, both rooted in the framework of diffusion models, represent a class of generative models that transform a distribution of random noise into a distribution of coherent images through a process of gradual denoising. DDPMs iteratively refine this process through a fixed number of steps, while DDIMs propose an accelerated approach, allowing for faster convergence with fewer steps without compromising the quality of the generated images. By comparing these models, we aim to shed light on their image-generation abilities by examining their output visual quality, speed, and flexibility. We quantitatively measure quality via the FID (Frechet Inception Distance) score metric [7].

2. Results

2.1. PixelCNN

Our PixelCNN results demonstrated the profound difficulty of image generation with smaller, older architectures. While we were unable to replicate the quality of our diffusion results with PixelCNN, our model achieved some representation of the spatial relationship between different colors and what appears to be edge detection/background vs. foreground, as shown in [6]. We achieved significantly reduced L_1 loss (from 58.4 to about 0.14) after a single epoch, and then had an difficult time reducing loss further. We were able to reach minimum L_1 losses of around 0.05 when we trained on a single image for 450000 iterations in an effort to assess the capacity of the PixelCNN to learn on an invariant input.

2.2. DDPM

The DDPM models performed qualitatively well, generating realistic images when trained on both the CIFAR-10 [1] and Anime-Faces [4a] datasets. On the CIFAR-10 dataset, the best DDPM model achieved an FID score of 123.53 (using the sigmoid scheduler) whereas the best DDPM model on the Anime-Faces dataset achieved an FID score of 92.93. While these are not state-of-the-art, our DDPM implementation significantly outperforms random noise (FID of ~ 550) and PixelCNN results and maintains high image quality on inspection.

2.2.1 Beta Schedules

We used cosine [1a], linear [1b], sigmoid [1c] and quadratic [1d] beta schedules. We found that by far, the cosine beta scheduled formed the worst. We can visually see this in [1a], as the generated images are of much worse quality, with very high contrast and saturation.

Furthermore, the FID score for the cosine beta schedule actually increased across sampling steps after the initial drops, which is unlike any of the other beta schedules, which all decreased across sampling steps. This suggests that the model is capable of doing the majority of its denoising work in a few steps, which lends credence to the idea that the cosine scheduler improves fast sampling performance, discussed further in [3.2.1]. The validation loss for the cosine beta schedule was also the highest at nearly all epochs [2b], but note that the sigmoid beta schedule hovered around the same values as well. The linear beta schedule had by far the lowest validation loss, being at least 0.01 lower on average.

It is important to note that the validation loss isn't perfectly correlated with accuracy - the linear scheduler, despite having the best validation loss, performs the second-worst in terms of FID score. This indicates that a more accurate per-step noise estimate doesn't necessarily yield a better final denoising result.

2.2.2 Sampling

The quality of image samples increases with the number of DDIM sampling steps taken. Note the difference between [4a] and [4c]; the 500-step process generates images with significantly higher quality reflections, structure, and high-resolution details. This result is mirrored quantitatively in the CIFAR-10 FID scores [2a], where FID distances decrease as sampling steps increase (for all schedulers other than cosine).

3. Discussion

3.1. PixelCNN

Image generation remains a non-trivial task to implement if one has access to limited computing power and model complexity. We realized the limitations of the PixelCNN to generate high-quality images when trained on the entirety of CIFAR-10, and therefore considered reducing the scope of the problem, first by training solely on images of a single class, and then by training on a single image alone. We include surprising output images [6] sampled after hundreds of thousands of iterations through a single image, but they are still unsatisfactory compared to what is known to be possible with similar architectures like the PixelRNN [22] or PixelCNN++ [18]. The prior includes more advanced recurrent stages, with LTSMs [9], and PixelCNN++ included more advances improving weight initialization, adding layer normalization [1], and a more advanced form of convolutional layers referred to as "gated" convolution. We hypothesize that in order to create a more powerful PixelCNN we would require significant architectural changes. The immediate change, unrelated to PRNN or PCNN++ is that that our base-line model is likely too small to operate at the level of our diffusion architectures, with less than 1% of the number of parameters of our DDPM implementation. Despite these results, we believe there was a tangible amount of learning, with a drastic reduction in L1 loss from the initialization of the model to after the first epoch (58.4 to 0.14). This further supports the difficulty of training image generation models. We believe that our PixelCNN learns features from a potential mean image in the training set, as well as spatial relationships between colors. The 'stripiness' in our images [6] may indicate some level of the representation of edges, and placement of green and pink in these images may indicate a level of foreground vs. background distinction.

3.2. DDPMs

DDPMs [8], work on the principle of iteratively removing Gaussian noise from a pure-noise seed image. The training process of a DDPM model consists of two stages:

1. The forward (unparameterized) noise addition process, which gradually adds Gaussian noise to an image until only noise remains. We can formulate it as:

$$x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon$$

Where x_0 is the initial image sampled from our data distribution, β_t is the **variance schedule** at time t, and ϵ is an I.I.D noise sampled per timestep from $\sim \mathcal{N}(0, I)$. Intuitively, the variance schedule can be understood as the scale of the noise added at each timestep. Typically, β_t increases with t, as the later stages in the forward diffusion process are intended to destroy most of the information in the original image.

Given that the summation of Gaussian random variables have additive variance, we can reformulate the above into this one-step equation:

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

Where $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=0}^{t} \alpha_t$. See [5] for an illustration of the forward process.

2. To train the model, we learn the noise added between x_0 and x_t. A typical training process goes as follows: a. Sample a t uniformly between 0 and T. T is the number of total forward (noise-adding) steps, which is a fixed value. b. Calculate x_t via the formulation above, and save ϵ. b. We train the model to predict $\epsilon_\theta(x_t, t)$. Note that the model's prediction is conditioned on t via a positional embedding of the timestep.

3.2.1 Variance Schedule Insight

It was shown in [14] that the cosine beta scheduler is designed to retain significantly higher amounts of original im-

age signal late in the noising process. This causes the task of extrapolating image features from noise to be significantly more difficult earlier in the process, which the authors discovered causes the model to perform better when denoising steps are skipped. However, this did not align with our results in [2.2.1]. In our case, it's likely our relatively small model was unable to learn the more challenging task that the cosine beta schedule posed. This phenomenon can be seen in the noise addition figures [5a], where the cosine noise scheduler has significantly less noise at the $t = 750$ step than all the other schedulers.

3.2.2 Loss Function Insight

Effectively, our network is a learned denoiser trying to predict the exact noise **added to the image**. Specifically, the noise ϵ before being scaled by β_t and added to $\sqrt{\bar{\alpha}_t}x_0$. We minimize the Huber loss [4] between ϵ_θ and ϵ:

$$\mathcal{L} = \begin{cases} \frac{1}{2}(\epsilon - \epsilon_\theta)^2 & \text{if } |(\epsilon - \epsilon_\theta)| < 1 \\ (\epsilon - \epsilon_\theta) - \frac{1}{2} & \text{otherwise} \end{cases}$$

In the early stages of training, DDPM models trained with Huber Loss [4] produced significantly clearer images than those trained with L_2 loss and more diverse images than those trained with L_1 loss. The Huber loss acts as L_2 loss for elements of ϵ_θ below 1, and acts identically to L_1 for elements above 1. We believe that the Huber loss aids stability by placing a cap on the gradient norm, while achieving diversity by providing a gentler gradient signal when the error is small.

3.2.3 Denoising Insight

In [14], it was shown that linear noise schedules work better for higher-resolution images, but for images of lower-resolution (i.e., 64×64 and 32×32) to be suboptimal. Namely, the authors claim that the linear noise schedule is too noisy at the end of the forward noise process, and thus using a noise schedule such as the cosine noise scheduling can be thought of as 'reducing overfitting.' This was visually supported and discussed in our results [2.2.2]. Further discussion of the sampling process can be found in [5.4].

3.3. DDIMs

DDIM models [20] are DDPM models with a reformulated sampling process. This property allows us to train a singular model using the training loop discussed in the DDPM formulation [8] and sample from it using the DDIM formulation. We find DDIM sampling has two distinct advantages over DDPM sampling:

1. DDIM sampling can be **deterministic**. The injected noise, z, in the DDPM formulation causes the same latent image x_T to yield different x_0s at the end of the sampling process between different sampling trials. On the other hand, DDIM processes have no stochasticity. This also implies a *consistency* property, where a smooth walk in latent space yields a smooth walk in image space. We discuss this further in [3.3.1].

2. DDIM sampling can take an **arbitrary number of steps**. A large speedup can be achieved by taking fewer denoising steps than the - typically quite large - number of noising steps. This speedup comes at a minor tradeoff with image quality, as shown in the FID vs Sampling Steps chart [2a].

3.3.1 Interpolation

As shown in [3], a smooth interpolation between two latent noise images yields gradually transitioning images when the interpolated latents are denoised to image space. Note that the formulation for DDIM sampling [5.4.2] has no stochastic element, which causes the denoising process to establish a mapping between latent noise and generated image. This process is not possible for DDPM models; passing the exact same noises used for DDIM interpolation into the DDPM sampling process yields a completely different set of generated images [3b].

3.4. Further Work & Limitations

3.4.1 Limitations

Due to computational constraints, we did not try larger image sizes (e.g., 256×256), but we expect generated images from these large image size datasets to have even better quality. Furthermore, the FID score resizes images to be 299×299 using bilinear interpolation, so images of larger size will have more accurate FID scores.

3.4.2 Further Work

To improve the quality of our generations, we were interested in investigating:

1. Latent Diffusion Models (LDMS) [16], where images are encoded and decoded via a VAE. Diffusion happens in latent space, making LDMs far more parameter and compute-efficient.

2. PixelRNNs [22] and PixelCNN++ [18] to gain more insight into our PixelCNN's performance.

3. Fourier positional embeddings, learned positional embeddings, and relative positional embeddings.

4. Class conditions via positional embeddings. We hypothesize that the DDIM denoising formulation will allow us to smoothly interpolate between class conditions, yielding an interesting semantic-space walk.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. 2

[2] S. Churchill and B. Chao. Anime faces dataset, 2020. 6

[3] Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html. 6

[4] Kaan Gokcesu and Hakan Gokcesu. Generalized huber loss for robust learning and its efficient minimization for a robust statistics, 2021. 3

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. 6

[6] Dan Hendrycks and Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415, 2016. 6

[7] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. 1

[8] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc., 2020. 1, 2, 3, 6

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. 2

[10] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. 6

[11] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020. 6

[12] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Large scale learning of general visual representations for transfer. *CoRR*, abs/1912.11370, 2019. 6

[13] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009. 6

[14] Alex Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. *CoRR*, abs/2102.09672, 2021. 2, 3

[15] Siyuan Qiao, Huiyu Wang, Chenxi Liu, Wei Shen, and Alan L. Yuille. Weight standardization. *CoRR*, abs/1903.10520, 2019. 6

[16] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. *CoRR*, abs/2112.10752, 2021. 3

[17] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. 6

[18] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P. Kingma. Pixelcnn++: A pixelcnn implementation with discretized logistic mixture likelihood and other modifications. In *ICLR*, 2017. 2, 3, 6

[19] Zhuoran Shen, Mingyuan Zhang, Haiyu Zhao, Shuai Yi, and Hongsheng Li. Efficient attention: Attention with linear complexities. *CoRR*, abs/1812.01243, 2018. 6

[20] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *CoRR*, abs/2010.02502, 2020. 1, 3, 6

[21] Serge Turukin. Pixelcnn, 2017. [Blog post]. 6

[22] Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *CoRR*, abs/1601.06759, 2016. 2, 3, 6

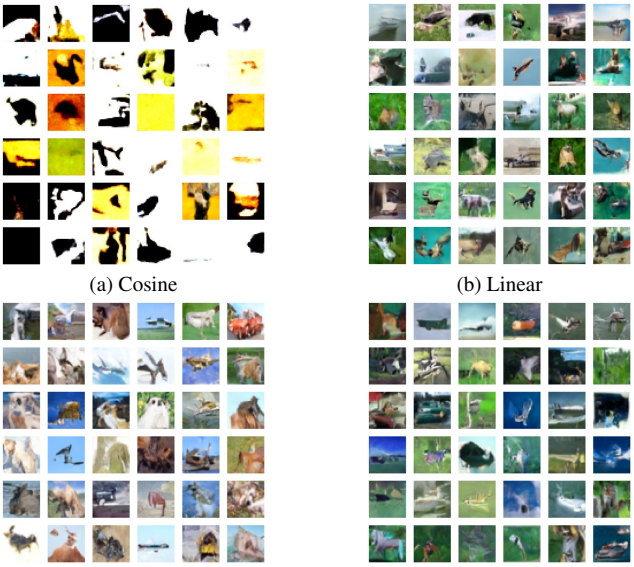
[23] Aäron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional image generation with pixelcnn decoders. *CoRR*, abs/1606.05328, 2016. 1, 6

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. 6

[25] Yuxin Wu and Kaiming He. Group normalization. *CoRR*, abs/1803.08494, 2018. 6

[26] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016. 6

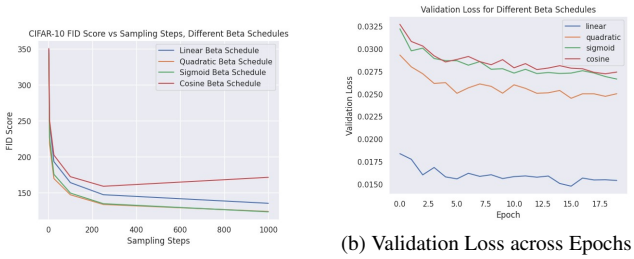
4. Figures



(a) Cosine (b) Linear

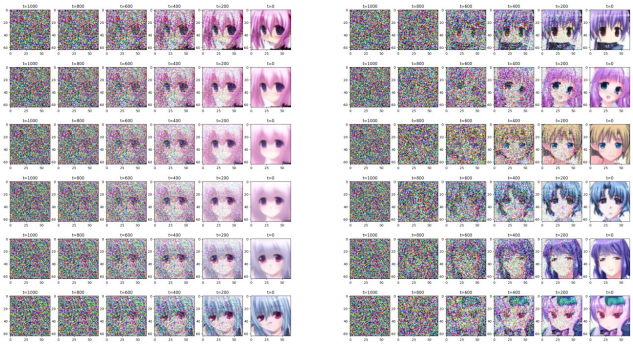
(c) Sigmoid (d) Quadratic

Figure 1. Beta schedules



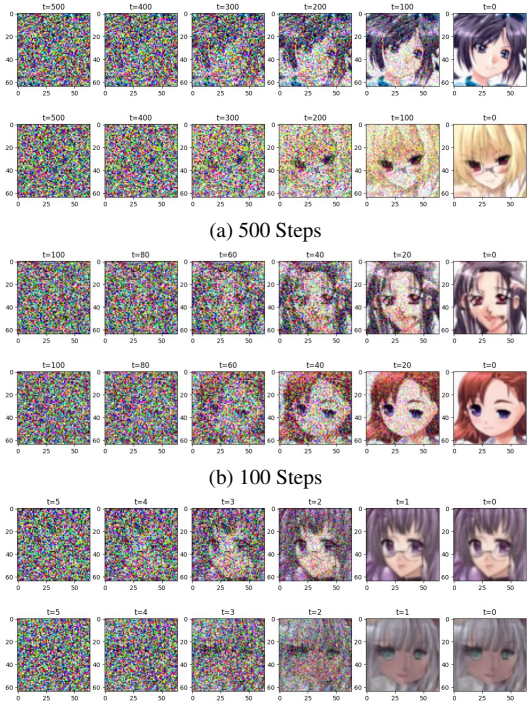
(a) CIFAR10 FID Scores (b) Validation Loss across Epochs among Different Beta Schedules

Figure 2. Graphs



(a) DDIM Interpolation (b) DDPM Interpolation

Figure 3. Interpolation

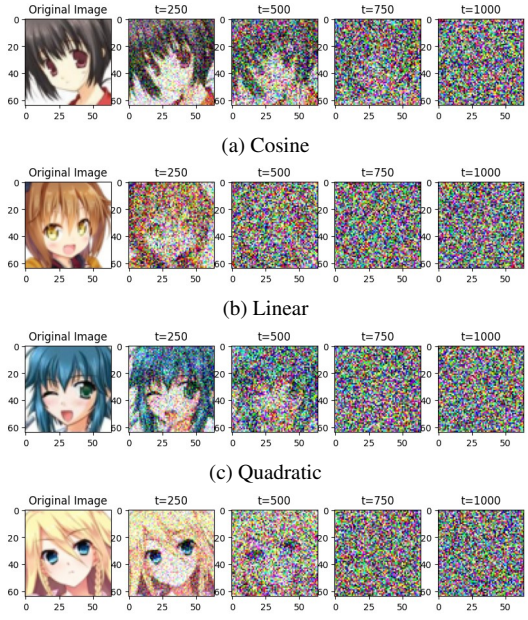


(a) 500 Steps

(b) 100 Steps

(c) 5 Steps

Figure 4. DDIM Denoising starting at different steps



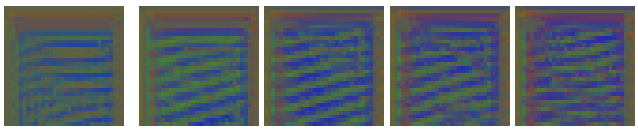
(a) Cosine

(b) Linear

(c) Quadratic

(d) Sigmoid

Figure 5. Forward Noise Scheduling



(a) 50k (b) 200k (c) 250k (d) 350k (e) 450k

Figure 6. PixelCNN Samples across Epochs Trained

5. Methods

5.1. Data

We primarily trained on and report only two datasets, CIFAR10 [13] and an Anime Faces dataset from Kaggle [2]. We chose these datasets since their image size was small and thus conducive to our limited computational resources. We downscaled the Anime Faces dataset to be 64×64.

5.2. U-Net

5.2.1 Training

Outside of model choices (beta schedules, noise schedules, etc), we trained all our U-Nets on the entirety of the train partitions of their datasets for 20 epochs with batch size of 32, learning rate of $1e{-}4$ and with 1000 forward steps. The total number of parameters was 171.39 million.

5.2.2 Architecture

Faithful to the DDPM paper [8], we use a backbone of PixelCNN++ [18], a U-Net [17] based on a Wide ResNet [26] with group normalization [25] instead of batch normalization. However, we use weight-standardized ResNets which were empirically seen to work better with group normalization [12, 15]. Specifically, our model architecture consists of a time embedding, a down-sampling path, mid-level (bottleneck) blocks, and up-sampling paths. For our time embedding, we use a sinusoidal positional embedding layer to encode time step information [6, 24]. This time embedding is passed through an MLP (linear $\rightarrow$ GeLU $\rightarrow$ linear) to obtain a time-dependent feature vector, which is then added to each residual block to enable the network to have knowledge of which particular time step it is operating. In our down-sampling path, we use two wide weighted-standardized ResNet blocks, followed by a linear attention [11, 19] layer which applies attention along the spatial dimensions and a convolution layer that reduces spatial dimensions by a factor of 2. After the down-sampling path, we use a wide weight-standardized ResNet block, followed by an attention block [24] which applies attention along both the spatial and channel dimensions, followed by another wide weight-standardized ResNet block. Then, our up-sampling path consists of two wide weight-standardized ResNet blocks, followed by a linear attention block along the spatial dimensions and an up-sampling layer, which increases the spatial dimensions by a factor of 2 using nearest neighbor sampling. Finally, our feature maps from the up-sampling path are concatenated with the initial feature map, and a final wide weight-standardized ResNet block is applied to the concatenated feature maps. Then, a final convolutional layer is used to produce our output tensor.

5.3. PixelCNN

5.3.1 Training

We trained our PixelCNN on the entire CIFAR-10 dataset for up to 200 epochs as well as a single image for up to 450,000 epochs both with batch size of 32. Optimization primarily consisted of augmenting the number of training epochs, as we use the learning rate ($1e{-}3$) employed by in the original PixelCNN paper [22, 23]. The total number of tunable parameters used was 1,307,651.

5.3.2 Architecture

Inspired by [22, 23], our PixelCNN is fundamentally based on a series of masked convolutions of an input image with size $3 \times 32 \times 32$. Our architecture included a total of four convolutional layers, two of which were masked to enforce an autoregressive structure. The output of each convolutional layer was followed by each of three stages, the feature maps produced by each convolutional batch-normalization layer [10] followed by the ReLU activation layer. Our architecture also employed a residual stream [3] (information travelling in the residual connections [5]) incorporated at the end of the second masking to prevent excessive information loss due to masking [21] though it is important to note that this architecture is not considered an RNN implementation.

5.4. Sampling

5.4.1 DDPM

We reproduce the sampling process in the DDPM paper published by *Ho et al* [8]. The reverse process, also known as the denoising process, begins with pure noise and iterates backward over T steps.

$$x_T \sim \mathcal{N}(0, I)$$

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right) + z \sqrt{\beta_t \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}}$$

Where $z \sim \mathcal{N}(0, I)$.

5.4.2 DDIM

Mathematically, the DDIM sampling process is formulated as [20]:

$$x_{\tau_{i-1}} = \sqrt{\frac{\bar{\alpha}_{\tau_{i-1}}}{\bar{\alpha}_{\tau_i}}} \left(x_{\tau_i} - \sqrt{1-\bar{\alpha}_\tau} \epsilon_\theta(x_{\tau_i}, \tau_i) \right)$$

$$+ \epsilon_\theta(x_{\tau_i}, \tau_i) \sqrt{1-\bar{\alpha}_{\tau_{i-1}}}$$

Where τ is a series of monotonically increasing values spanning 0 to T, inclusive. Note that τ does not have to correspond to t; $\tau = [0, 5, 10, ..., 1000]$ is also a valid sampling schedule.