Accelerating Diffusion Models using Adaptive Neural Network Architecture

Tomer Giladi, Shelly Meir, Bahjat Kawar, Michael Elad
GIP Lab, Computer Science Department, Technion, Haifa, Israel
{tomergiladi@cs, shellymeir@campus, bahjat.kawar@cs, elad@cs}.technion.ac.il

1 Introduction

Denoising diffusion probabilistic models [1] use Langevin dynamics to generate random images from a learned distribution [4, 5]. Existing diffusion models reach good results, but the generation of an image takes a long time to compute. Our goal is to shorten the generation time of an image. We reduce the computation time by using smaller neural networks when generating highly noisy images, at the beginning of the diffusion process. As the process progresses and the images get less noisy, we add more layers to the neural network in order to get better denoising diffusion results.

2 Background

2.1 Score-Based Generative Modeling

Using the score function $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ of a probability density $p(\mathbf{x})$, one can use Langevin dynamics to generate a sample from the distribution $P(\mathbf{x})$. However, as was shown in [4], the estimated score function is inaccurate, and possibly undefined, in regions with low probability, which means that the generation process will be inaccurate. Hence, this paper and its successors [5, 1] define noisy versions $\{\mathbf{x}_t\}_{t=1}^{T}$ of real images $\mathbf{x} \sim p(\mathbf{x})$ and use them in the context of *annealed* Langevin dynamics, where Langevin dynamics is applied over gradually less noisy distributions $p_t(\mathbf{x}_t)$. For example, the authors of [4] define $\mathbf{x}_t = \mathbf{x} + \mathbf{z}_t$ where $\mathbf{z}_t \sim \mathcal{N}(0, \sigma_t^2 \mathbf{I})$. In the paper [4], it was shown that it is possible to train a neural network $s_\theta(\mathbf{x}_t, t)$ to approximate the score function, $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$, by using samples from $p_t(\mathbf{x})$. Thus, for a small final noise σ_T, we get approximate samples from $p(x)$.

2.2 U-Net

U-Net [2] is a convolutional neural network that is based on the fully convolutional network with modifications. The network consists of an initial contracting path, followed by an expansive path. In the contracting path, the number of feature channels is doubled at each downsampling step and the image size is halved. The expansive path does the opposite - at each upsampling step the image size is doubled and the number of feature channels is halved. Every upsampling step consists of a concatenation with the correspondingly cropped feature map from the contracting path. Diffusion models such as [4, 1] usually rely on a U-Net-like neural network architecture for their score estimation networks, which are essentially trained for denoising.

3 Our Work

In order to improve the existing architecture's inference runtime performance, we have decided to utilize the fact that the output of each iteration of the annealed Langevin dynamics process is only used to initialize the next iteration, and therefore the output of the earlier iterations which produce highly noised images needs not to be as accurate as the latter ones.

Preprint. Under review.

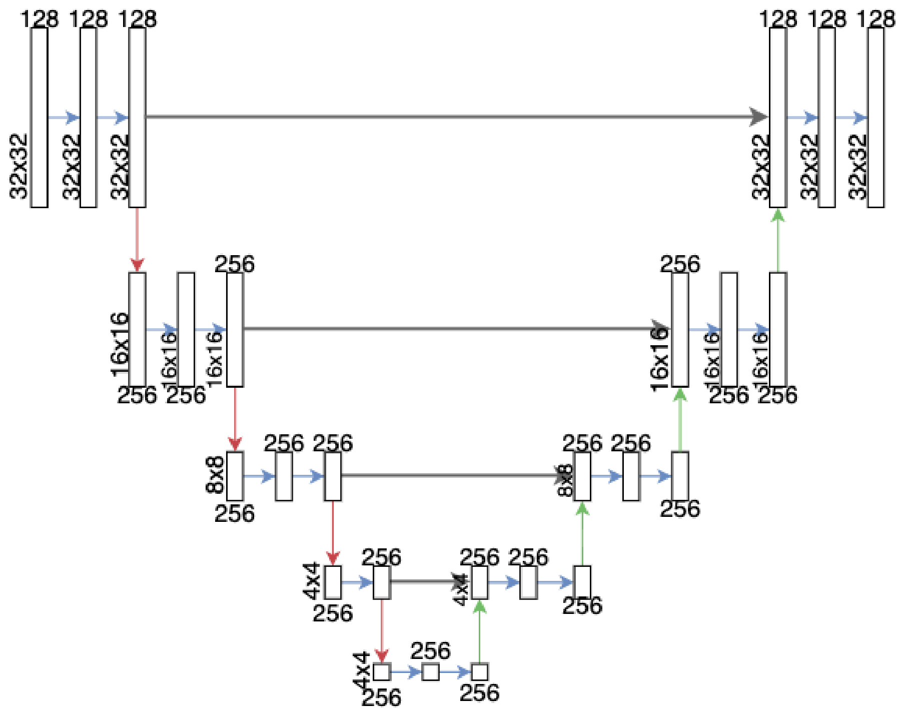

Figure 1: The U-Net architecture used in the paper [3], which we in turn attempt to accelerate.

Our main idea is to utilize the existing skip connections in the network and use them to shortcut over the additional layers when generating the highly noisy images, while using the entire network when generating the lesser noised images. This way, we avoid the computation of these layers in the early stages of the generation process, which in turn accelerates it.

We faced two major challenges when approaching this task. The first one was to choose a training scheme that would allow our network to learn how to generate images when the first part of the generation process uses less of the network and therefore less accurate. The second challenge was to choose how to implement the skip connections during the image generation stages.

In order to implement the skip connections we made use of the U-Net layer design and made a shortcut from each layer in the contracting path to its corresponding layer in the expansive path, avoiding the computation of the lower-scale layers. However, this results in a missing input for the first active layer in the expansive path. In order to account for that, we propose three different methods. The first method is to simply use a $\mathbf{0}$-vector as the input. The second method is to duplicate the vector from the contracting path into both inputs of the expansive layer. Finally, the third method is to add a simple path from each layer in the contracting path to the middle block of the net and then add a corresponding simple path to the expansive layer, where each path uses a bicubic interpolation in order to match the dimensions of the middle block.

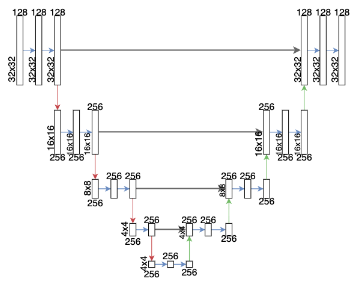

Figure 2: U-Net architecture with 0 shortcuts

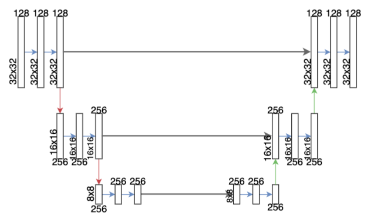

Figure 3: U-Net architecture with 1 shortcut

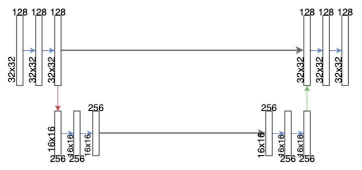

Figure 4: U-Net architecture with 2 shortcuts

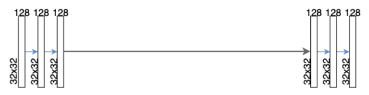

Figure 5: U-Net architecture with 3 shortcuts

In order to train our network we have looked at two training schemes. The first training scheme was to choose k, the number of spatial layer levels to shortcut over, to be randomly sampled from a uniform distribution between 0 and $N-1$ at each batch of the training phase, where N is the total number of levels in the architecture. While the noise levels of the samples of the batch are uniformly sampled as in the original architecture [4] Our motivation behind this training scheme is that each part of the network should learn to denoise an image with any noise level independently. This will give a good approximation of the denoised image at all steps of the generation process which will result in a good image generation scheme.

In the inference process, we want to use a smaller network for the highly noised images and to add more layers to the net as the image gets less noisy. Since there's a trade-off between the amount of time needed to produce a sample and the sample's quality, we considered using the following family of parameterized functions

$$k(t) = \min\left(N-1,\ \lfloor(N - N\cdot(1-t)^c)\rfloor\right) \tag{1}$$

using $c \in \{1,2,3,5\}$, where N is the number of spatial layer levels in the network. Setting $c = 0$ is equivalent to running the regular non-accelerated network. As we can see in Figures 6 through 9 this gives a family of increasing step functions such that higher values of t which are used in the generation of the highly noised images gets an higher k and lower values of t which are used in the generation less noised images we get lower k. This means that during the generation of highly noised images we use only a small part of the network, while generating the less noised images we use the entire network. As we can see in the figures higher values of c will cause k increase k which means will use less of the network and gain higher speed ups, but lose on the accuracy of the network.

As we can see from the figures our family of functions for k are always a non-strictly monotonically increasing step functions which led us to construct an improve training scheme where for each batch the training data that is passed to the network $\{\tilde{\mathbf{x}}_\mathbf{i}, t_i\}_{i=1}^{B}$ is chosen such that $k(t_i)$ is constant for every sample in the batch and then at each batch we choose k to be that constant. This means that the deeper layers of the network will only train on denoising low noised images while the shallow layers would train on fully denoising highly noised images. The inference process will remain the same.

Table 1: FID scores for different skip types, under different timestep schedule partitions. We show scores for the pre-trained DDPM model (with zero skipping in inference time) in the first row as a baseline.

Skip type	$c = 0$	$c = 1$	$c = 2$	$c = 3$	$c = 5$
DDPM	**3.17**	16.38	73.15	115.09	143.34
Zero	4.58	**4.46**	5.59	9.40	16.19
Duplicate	4.91	4.55	7.62	14.69	31.79
Interpolate	4.96	4.71	**5.03**	**7.02**	**11.47**

Table 2: Speed-up for different skip types, under different timestep schedule partitions.

Skip type	$c = 0$	$c = 1$	$c = 2$	$c = 3$	$c = 5$
Zero	0%	30.59%	**47.52%**	51.44%	57.46%
Duplicate	0%	**32.15%**	45.63%	**51.99%**	**58.38%**
Interpolate	0%	20.86%	29.34%	32.68%	36.19%

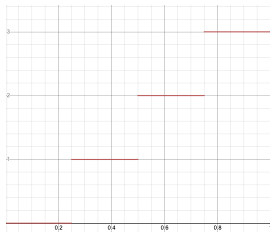

Figure 6: c = 1

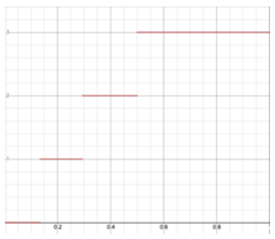

Figure 7: c = 2

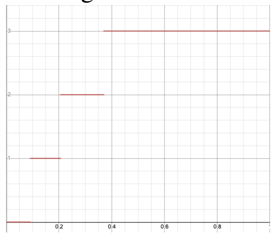

Figure 8: c = 3

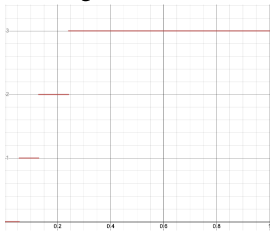

Figure 9: c = 5

4 Results

4.1 Speed-up

In Table 2, we report the time speed-up percentage of all skip types, under different timestep schedule partitions as shown in figures 6, 7, 8, and 9. As expected, the speed-up percentage becomes higher as

Table 3: FID scores for our model trained on zero skipping for $c = 2$, evaluated using DDIM schedule [3]. We also show original DDIM scores as baseline. Two timesteps of our model are roughly equal in time to one timestep in the pre-trained DDPM model.

	Our Model					DDPM / DDIM				
	20	40	100	200	1000	10	20	50	100	1000
$\eta = 0$	**16.80**	**12.48**	9.74	8.75	7.79	**13.46**	**6.84**	**4.67**	**4.16**	4.04
$\eta = 0.5$	17.96	12.754	**9.43**	**8.23**	6.90	16.66	8.35	5.25	4.46	4.29
$\eta = 1$	29.00	18.71	11.90	8.92	6.00	41.07	18.36	8.01	5.78	4.73
DDPM	209.72	124.69	56.15	28.01	**4.57**	367.43	133.37	32.72	9.99	**3.17**

we increase c, meaning we skip more layers of the neural network for longer period of time in the inference process. For $c = 2$ we get a speed-up of 47.52%, so for this c our model is roughly twice as fast as the DDPM model, on average.

4.2 Sample Quality

In Table 1 we show FID-scores of the pre-trained DDPM model and of our model with all three methods for skip connections. We observe that for $c = 0$ the DDPM model gets better results than our model, but as we increase c its quality is severely reduced. For our models we see that the quality deteriorates at a lower rate as we increase c so our models are capable of producing good enough noised images with partial networks.

It is clear that there is a tradeoff between the speedup and the samples' quality. Since our goal is to shorten the generation time of an image, we want to choose c that is large enough. On the other hand, we don't want to produce low-quality images. The sample quality seems to decrease most drastically for $c \geq 3$. Therefore, we choose $c = 2$. For this c we get an FID score of 5.03 with Interpolate skip-type, and 5.59 with Zero skip-type. However, in Table 2 we see that for $c = 2$ the Zero skip-type gets a speedup of 47.52% where the Interpolate skip-type only gets a speedup of 29.34%. Hence, we choose to train our net with the Zero method for the skip connections.

4.3 Comparing to Other Methods

In Table 3 we show the FID scores of our model, trained with zero-skipping and $c = 2$, compared to the FID scores of DDPM model. Our model is approximately twice as fast as the DDPM model, and it has achieved the FID score of 4.57, compared to 3.17 which is the FID score of DDPM model.

Moreover, we show the scores of our model evaluated using different Langevin update rule schedules suggested by DDIM [3], compared to the original scores reported in DDIM using a DDPM [1] model. One timestep of the DDPM model takes roughly twice as much time as our model on average. Thus, in order to compare between the two methods we should compare the DDIM FID scores to our corresponding FID scores with twice the timesteps. Yet we observe that the scores of the DDIM model are better than ours.

5 Conclusion

Diffusion models are good at image generation, but take long time to generate an image. We propose an acceleration scheme using shortcuts in the U-Net architecture, exploring several methods for achieving that, such as different skip types and different timestep schedule partitions. We get an FID score of 4.57 at a 47% speedup. We achieved our goal and managed to accelerate the generation process, but the tradeoff between the sample quality and generation runtime that we achieved is overshadowed by the one achieved in DDIM [3], where they got better FID scores at better speedup rates. We also tried combining the two methods yet the DDIM was still preferable.

References

[1] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[2] 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.

[3] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[4] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in Neural Information Processing Systems*, 32, 2019.

[5] Yang Song and Stefano Ermon. Improved techniques for training score-based generative models. *Advances in neural information processing systems*, 33:12438–12448, 2020.