Understanding and Optimizing Diffusion Models with Prompt Based Image Generation

Vinita Takawale
Columbia University
vt2365@columbia.edu

Alisha Varma
Columbia University
av3120@columbia.edu

Abstract

1. Introduction

Deep neural networks were been proposed over half a century ago. And while there has been a world of research and improvements since then there is still much more room to grow. Diffusion models are type of deep neural network that was only recently proposed in the last decade. Though there has been some key advancements in this model type it is still fairly new and room for optimization and use case growth.

Prompt generation content has become an up and coming topic with the creation of tools like ChatGPT and DALLE. Therefore we decided to explore what would happen if we were to combine the use of prompt generation and optimizations to diffusion models as a pipeline. In order to do so, we needed to pull from outside libraries and research backed computer vision techniques to test if this pipeline generation is possible.

In this paper, we propose a denoising diffusion probabilistic model that contains a combination of optimization techniques to generate flower images based on user input prompts. We aimed to improve the current proposed diffusion model process and combine proposed techniques with the help of libraries PyTorch and CLIP.

2. Related Work

Diffusion models were first introduced in 2015 and are loosely based on non-equilibrium statistical physics [8]. Sohel et. al proposes iteratively destroying the data distribution through forward diffusion and restoring the data through a reverse diffusion process in order to have a generative model of the data [8]. Diffusion Models use an iterative approach to both forward and reverse diffusion process, as it provides a more accurate and traceable generative model [8]. Forward diffusion involves the application of noise to an image, in this case, till it reaches the point of pure noise. In which the noise added in each time step is regulated through the scaling of the mean and variance. Then from that pure noise image, the reverse diffusion process the neural network model learn to iteratively remove the noise from said image resulting in an distinct image.

Building upon the diffusion models initial proposal, synthesis of high quality image in connection with diffusion models and a handful of improvements related to the denonisation of score matching and the use of Langevin dynamics [4]. Ho et. al used a linear scheduling of the forward diffusion process but provided valuable contributions in other areas. In addition, they used a U-net architecture that uses residual network and downsampling blocks in the forward diffusion process to a minute resolution of an image and then gradually rebuilds the image through the upsampling blocks [4]. There is the utilization of attention blocks at different resolutions layers and the use of Transfer sinusoidal position embedding [4, 11]. In our model, we chose to utilize the U-net architecture as a building block considering its improvements.

The aspect of cosine scheduling was later proposed to improve the forward diffusion process by slowing down the data destruction by noise and allowing for more informative data [1]. In our exploration, we choose to use cosine scheduling due to its improvements over linear scheduling. This paper also laid out that diffusion models are comparable, if not better, results than the GAN models [1].

One of the latest improvements of diffusion model architecture was Dhariwal and Nichol [5]. In this architecture, the discovered addition of more attention layers and heads, decreasing of width and increasing of depth, and the use of BigGAN-deep for upsampling and downsampling improved results greatly [5]. In addition to improvements on diffusion models, the discovery of Vision Transformer (ViT) has allowed for better computer vision application results than convolutional neural networks [2]. As well as utilizing less resources overall [2].

Though diffusion models are fairly new, the current range of work provides a solid basis for us to base our exploration off of. In the following we lay out our exploration of

diffusion models, in particular a denoising diffusion probalistic model for prompt based flower image generation.

3. Methodology

Our goal was to train a denoising diffusion probabilistic model in order to generate flower images based on given prompts. In order to train our diffusion model, we used the TF-FLowers dataset from the consolidated tensor flow datasets openly provided [10]. The dataset contains labels images of a flowers, like daisys, roses, tulips and sunflowers.

Diffusion models define a Markov chain of diffusion steps to slowly add random noise to data and then learn to reverse the diffusion process to construct desired data samples from the noise. Diffusion models are learned with a fixed procedure and the latent variable has high dimensionality (same as the original data). To build ours we did the following:

3.1. Forward Diffusion

Forward diffusion process comprises of adding small amount of gaussian noise to the sample in T steps, gradually producing a sequences of noisy samples x1, ….., xT . The timestamps are usually controlled by using a variance schedule. Letting T be the number of times we will add noise to an image. We can use t to keep track of the current timestep. We used a variance schedule, represented as β_t, or B in code. That describes how much noise will be added to our image at each timestep t.

In Section 4 of the paper Denoising Diffusion Probabilistic Models, the authors discuss the art of defining a good schedule. It should be large enough for the model to recognize noise was added (especially since the image may already be noisy), but still as small as possible [4]. A Normal Distribution has the following signature:

$$\mathcal{N}\left(x; u, \sigma^2\right) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Which reads as, "the normal distribution of x with parameters u (the mean) and σ^2 (the variance). When μ is 0 amd σ is 1 , we have a standard normal distribution $\mathcal{N}(x; 0, 1)$.

When we are altering our image with noise multiple times across many timesteps, we describe $\mathbf{x}_t$ as our image at timestep t. Then, $\mathbf{x}_{t-1}$ would be the image at the previous timestep and x_0 would be the original image. We will alter the image with following equation:

$$q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right) = \mathcal{N}\left(\mathbf{x}_t; \sqrt{1-\beta_t} \cdot \mathbf{x}_{t-1}, \beta_t \cdot \mathbf{I}\right)$$

Where q represents a probability distribution for the forward diffusion process and $q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right)$ describes the probability distribution for a new, noiser image $\mathbf{x}_t$ based on $\mathbf{x}_{t-1}$. We can sample from this probability distribution by first sampling from a standard normal distribution $\mathcal{N}(x; 0, 1)$ using torch.randn_like: noise = torch.randn_like (x_t)

We can then multiply and add to the noise to sample from q : x_t = torch.sqrt $(1 - \mathrm{B[t]}) * \mathrm{x\_t}$+ torch.sqrt $(\mathrm{B[t]})*$ noise

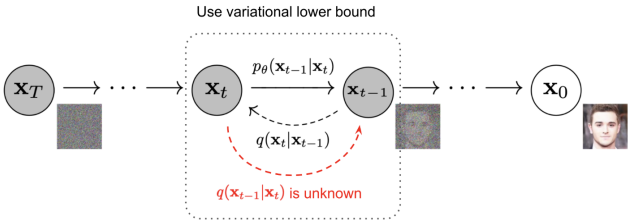

Figure 1. Markov chain of forward (reverse) diffusion process of generating a sample by slowly adding (removing) noise. Image is from Ho et. al and another [4, 12]

```
def get_loss(model, x_0, t):
    x_noisy, noise = q(x_0, t)
    noise_pred = model(x_noisy, t)
    return F.mse_loss(noise, noise_pred)
```

Figure 2. The Mean squared loss function that was utilized. As discussed in Section 3.2.

3.2. Loss Function

We are comparing the real noise that was added to the image and the predicted noise. Originally, the loss function was based on the Evidence Lower Bound (ELBO). The Mean Squared Error between the predicted noise and true noise was better in practice [4]. Therefore we chose to use the Mean Squared Error method.

3.3. Reverse Diffusion

From the above, we have a model that predicts the noise added to an image at timestep t, but generating images is not as easy as repeatedly subtracting and adding noise. The q function can be reversed such that we generate $\mathbf{x}_{t-1}$ from $\mathbf{x}_t$.

$$q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) = \mathcal{N}\left(\mathbf{x}_{t-1}; \tilde{\mu}\left(\mathbf{x_t}, \mathbf{x_0}\right), \tilde{\beta}_t \cdot \mathbf{I}\right)$$

Note: $\tilde{\beta}_t$ was originally calculated to be $\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$, but in practice, using only β_t is more effective.

Using Bayes' Theorem, we can derive an equation for the model_mean u_t at timestep t.

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

The image $\mathbf{x}_{t-1}$ can be estimated by $\tilde{\mu}_t + \tilde{\beta}_t \cdot \mathbf{I}$, so we used this equation to generate sample images recursively until we reached $t == 0$.

3.4. Optimization Techniques

We have applied following optimization techniques for performance metrics: (1) Group Normalization, (2) GELU, (3) Rearrange Pooling, (4) Time Embeddings, and (5) Residual Connections.

3.4.1 Group Normalization

Group Normalization (GN) divides the channels into groups and computes within each group the mean and variance for normalization. GN's computation is independent of batch sizes, and its accuracy is stable in a wide range of batch sizes. GN normalizes the output of a group of kernels for each sample image, effectively "grouping" a set of features. Considering color images have multiple color channels, this can have an interesting impact on the output colors of generated images.

We have applied Group Normalization technique over Batch Normalization as Group Normalization technique converts the output of each kernel channel to a z-score. It does this by calculating the mean and standard deviation across a batch of inputs. This is ineffective if the batch size is small.

3.4.2 GELU

Gaussian Error Linear Unit (GELU) is an activation function motivated by combining properties from dropout, zoneout, and ReLUs. GELUs relate to stochastic regularizers as there is an expectation of a modification for Adaptive Dropout suggesting probabilistic views of neuron output [3]. It has been shown that the novel nonlinearity matches and can exceeds models with ELUs or ReLUs across a variety of tasks in fields like natural language processing, computer vision, and automatic speech recognition [3].

ReLU is a popular choice for an activation function because it is computationally quick and easy to calculate the gradient for, but it isn't perfect. When the bias term becomes largely negative, a ReLU neuron "dies" because both its output and gradient are zero. At a slight cost in computational power, GELU seeks to rectify the rectified linear unit by mimicking the shape of the ReLU function while avoiding a zero gradient, hence we use GELU.

3.4.3 Rearrange Pooling

There are many types of pooling layers including Min Pooling and Average Pooling. In our model, we let the neural network decide what is important. We are using the einops library and the Rearrange layer [7]. We assign each layer a variable and use that to rearrange our values. Additionally, we can use parentheses () to identify a set of variables that are multiplied together. For example, in the code block below, we have:

Rearrange("b c (h p1) (w p2) $\rightarrow$ $b(cp1p2)hw$", $p1 = 2, p2 = 2$)

Where b is our batch dimension, c is our channel dimension, h is our height dimension, and w is our width dimension. We also have a p1 and p2 value that are both equal to 2. The left portion of the equation before the arrow is saying "split the height and width dimensions in half. The right portion of the equation after the arrow is saying "stack the split dimensions along the channel dimension".

3.4.4 Time Embedding

The better the model understands the timestep it is in for the reverse diffusion process, the better it will be able to correctly identify the added noise. We help our model better interpret this through time embedding. Positional encoding is a means of translating the location of objects in a sequence into information that a our model can understand and use. Therefore by feeding the output of the SinusoidalPositionEmbedding into our EmbedBlock we help our model inetrpret timesteps better.

3.4.5 Residual Connections

Lastly, to eliminate the checkerboard problem we are using the trick of adding more residual or skip connections.

3.4.6 CLIP

In addition to the optimization techniques, we use Contrastive Language-Image Pre-Training (CLIP) which is a text and image encoding tool used with many popular Generative AI models such as DALL-E and Stable Diffusion [6]. CLIP in itself is not a Generative AI model, but is instead used to align text encodings with image encodings. The goal of CLIP is to create the same vector embedding for both the image and the text description. Therefore, instead of using text description to create a text-image-pipeline we are able to generate images from prompts as we utilize image CLIP encodings to preprocess the text input.

4. Experimental Results & Discussion

Through this model creation, we needed to load image encodings and generate text encodings. To generate these text encodings, we did so through the use of CLIP described above in Section 3.4.6. During this process we first needed to calculate the cosine similarity between the text encodings and the image encodings to see which one of our text prompts, in our experiment, best describes the daisy, sunflower and rose. When the cosine similarity is 1, it's a perfect match. When the cosine similarity is -1, the two encodings are opposites. In Figure 3, we showcase a visual of

Figure 3. Comparison of the cosine similarity between the CLIP encoding results and images from the given prompts. The yellower the square the higher the similarity, the bluer the square the lower the similarity.

the cosine similarity between three different image and text encodings. This shows that the similarity is highest for the left diagonal with the most similarity between image and text for the daisy and rose.

From the encodings, we were able to train our model and generate flower images based on prompts that can be seen in Figure 4. This image showcases a collection of images from the given prompts. Which were: "A daisy", "A pink rose", "A yellow sunflower", and "A purple rose." The bottom row of images are those with the highest degree of similarity and what we consider a final result.

4.1. Limitations

This model fails to achieve high quality images without compromising on fast sample generation. Mainly, Denoising Probablistic Models require simulating a Markov chain for many steps in order to produce a sample and this inevitably impacts sample generation speed. An alternate approach could be to incorporate denoising diffusion implicit models as presented by Song et al., where they proposes implicit models aiming to generalize DDPMs using non-Markovian diffusion processes which can substantially improve sample generation speed [9]. Implicit Models aspire to achieve the same objective as denoising diffusion probablistic models but with generative processes that are deterministic leading to accelerted high quality samples.

4.2. Future Work

The scope of this experiment is decently small. We are only using TensorFlow's flowers, but it would be ideal to expand the scope of the data slowly as we have already optimized our pipeline to handle larger loads of data with techniques like GELUs over ReLUs. In addition, we would ideally like to add more optimization techniques to improve our models current loss as it could be much closer to zero.

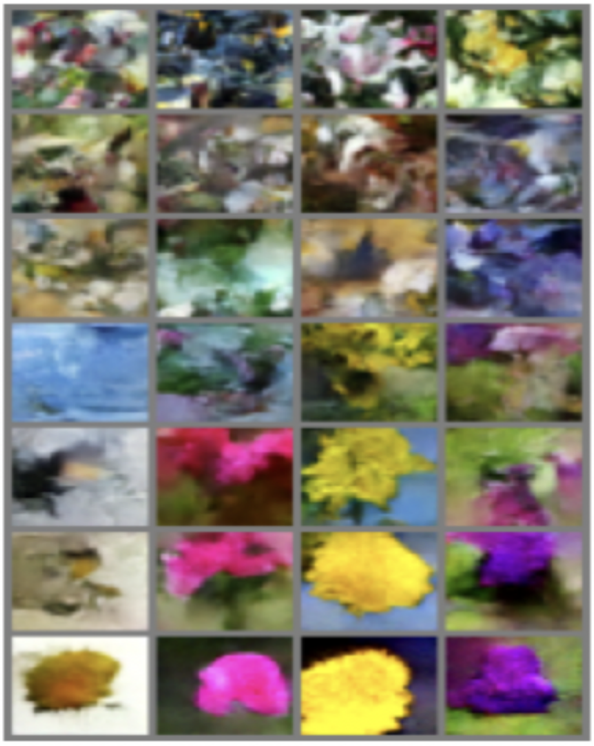

Figure 4. Image results of from the given prompts: "A daisy", "A pink rose", "A yellow sunflower", and "A purple rose".

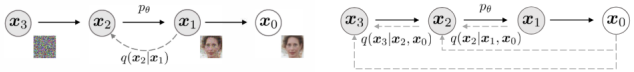

Figure 5. Shows inference models; diffusion on the left and non-Markovian on the right. Image is from Song et. al [9]. Discussed in Figure 4.1.

5. Conclusion

We have presented prompt generated flower images pipeline by training our own diffusion model, by combining a handful of optimization techniques and the utilization of prompt generation. We have found that a combination of optimization techniques improves the quality of the prompt generated flower images. While our model is not perfect, we look forward to investigating more optimization techniques to improve our model's performance and expand the scope of our current prompt generation.

6. Individual Contributions

Both team members contributed to this project. Vinita focused a bit more on the code and Alisha focused a bit more on the background research and paper writing.

7. Links

Code: https://github.com/vinitakawale/Diffusion_Model
Dataset: https://www.tensorflow.org/datasets/catalog/tf_flowers

References

[1] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021. 1

[2] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 1

[3] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016. 3

[4] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. 1, 2

[5] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International conference on machine learning*, pages 8162–8171. PMLR, 2021. 1

[6] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021. 3

[7] Alex Rogozhnikov. Einops: Clear and reliable tensor manipulations with einstein-like notation. In *International Conference on Learning Representations*, 2022. 3

[8] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR, 2015. 1

[9] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020. 4

[10] The TensorFlow Team. Flowers, jan 2019. 2

[11] 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. 1

[12] Lilian Weng. What are diffusion models? *lilianweng.github.io*, Jul 2021. 2