Training Basic Diffusion Models and Exploring Advanced Techniques for Improvement and Analysis

Project - ECE 285

Xuanbin Peng
Electrical and Computer Engineering
U09858781

Yixin Zhang
Electrical and Computer Engineering
U09843034

June 12, 2024

Abstract

This project presents a comprehensive exploration of diffusion models, focusing primarily on the training of basic models and the subsequent application of advanced techniques for enhancement and analytical purposes. Initially, we delineate the methodology employed in training basic diffusion models, emphasizing the foundational algorithms and their implementation. Subsequently, we introduce an improved diffusion model strategy that incorporates Classifier Free Guidance and Exponential Moving Average techniques to refine the generative capabilities and stability of these models. Through a series of experiments, we analyze the performance improvements these techniques offer over standard diffusion models. Our results on *FashionMNIST* demonstrate advancements, providing a promising outlook for the application in other data environments.

1 Problem Definition

Diffusion models, inspired by non-equilibrium thermodynamics [5], have emerged as a formidable new category of deep generative models. These models have demonstrated record-breaking performance across various applications such as image synthesis, video generation, and molecule design, outperforming GANs in several benchmarks [1]. This project implements the Denoising Diffusion Probabilistic Models [3], which we refer to as "diffusion models" for brevity. A diffusion model is a parameterized Markov chain, trained via variational inference, to produce samples that match the target data distribution after a finite sequence of transitions. These transitions are designed to invert a diffusion process—a Markov chain that incrementally introduces noise into the data, effectively reversing the data's signal degradation over time. When this diffusion process involves minor increments of Gaussian noise, it permits the sampling chain's transitions to also be modeled as conditional Gaussians. This characteristic simplifies the neural network's parameterization, facilitating a more straightforward implementation.

A denoising diffusion model is a two-step process consisting of:

- **Forward diffusion process** — a Markov chain of diffusion steps that gradually and randomly adds noise to the original data.
- **Reverse diffusion process** — attempts to reverse the diffusion process to regenerate the original data from the noise.

In the forward diffusion process, Gaussian noise is slowly and incrementally added to the input image x_0 through a series of T steps. Initially, a data point x_0 is sampled from the real data distribution $q(x)$, represented as $x_0 \sim q(x)$. Subsequently, Gaussian noise with variance β_t is added to x_{t-1}, resulting in a new latent variable x_t with distribution $q(x_t|x_{t-1})$:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

The entire sequence from x_0 to x_T is described by:

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$$

The reverse diffusion process involves training a neural network to recover the original data by reversing the noise addition applied in the forward phase. Estimating $q(x_{t-1}|x_t)$ is challenging as it might require the entire dataset. A parameterized model p_θ (Neural Network) is therefore used to learn the parameters. For sufficiently small β_t, the model assumes a Gaussian distribution, which is parameterized solely by its mean and variance:

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

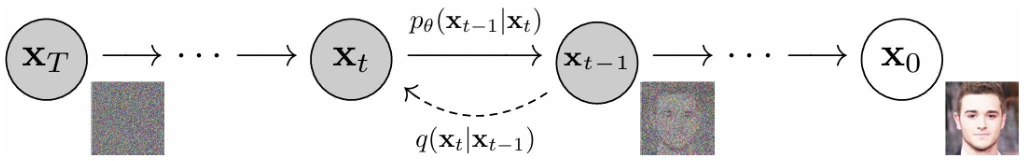

$\mathbf{x}_T$
$\mathbf{x}_t$
$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$
$\mathbf{x}_{t-1}$
$q(\mathbf{x}_t|\mathbf{x}_{t-1})$
$\mathbf{x}_0$

Figure 1: The directed graphical model considered in this proeject.

Apart from the basic model, we also aim to explore advanced techniques to improve the diffusion model: classifier-free guidance [2] and exponential moving average [4]. We conduct experiments on the FASHION-MINIST dataset to demonstrate both qualitative improvements from these techniques.

2 Method

2.1 Basic Diffusion Model Training

The training algorithm for a diffusion model iteratively optimizes the model parameters to better approximate the data distribution. At each iteration, an original sample x_0 is drawn from the data distribution $q(x_0)$. A timestep t is uniformly sampled, and noise ϵ is generated from a normal distribution. The model then takes a gradient descent step to minimize the error between the noise ϵ and the noise predicted by the model ϵ_θ. This predicted noise is computed using the noisy version of x_0, which is a combination of x_0 scaled by $\sqrt{\overline{\alpha_t}}$ and ϵ scaled by $\sqrt{1 - \overline{\alpha_t}}$, at timestep t. The training continues until convergence criteria are met.
The sampling algorithm is used to generate data from the trained diffusion model starting from noise. Initially, x_T is sampled from a normal distribution. The process then iteratively computes previous timesteps by denoising and adjusting x_t using the learned model ϵ_θ. If t is greater than 1, noise z is sampled; otherwise, z is set to zero. The computation for x_{t-1} involves correcting x_t using the model's noise prediction adjusted by α_t and $\overline{\alpha_t}$, and adding scaled noise $\sigma_t z$ to introduce variability. This backward process continues until x_0 is recovered, which is then returned as the generated sample.

Algorithm 1 Training

```
1: repeat
2:     $x_0 \sim q(x_0)$
3:     $t \sim \text{Uniform}\{1, \ldots, T\}$
4:     $\epsilon \sim \mathcal{N}(0, I)$
5:     Take gradient descent step on $\nabla_\theta \|\epsilon - \epsilon_\theta(\sqrt{\overline{\alpha}_t} x_0 + \sqrt{1 - \overline{\alpha}_t}\epsilon, t)\|^2$
6: until converged
```

Algorithm 2 Sampling

```
1: $x_T \sim \mathcal{N}(0, I)$
2: for $t = T, \dots, 1$ do
3:     $z \sim \mathcal{N}(0, I)$ if $t > 1$, else $z = 0$
4:     $x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right) + \sigma_t z$
5: end for
6: return $x_0$
```

2.2 Time Embedding

The introduction of time into a UNet architecture involves embedding time-based positional information, which can be critical in tasks where sequence or timing is important. We achieve this through a positional encoding mechanism that uses a combination of sine and cosine functions with exponentially decreasing frequencies. The inverse frequency calculation is described as follows:

Given a total number of channels denoted by channels, the inverse frequency inv_freq for each dimension i (where i ranges from 0 to $\frac{\text{channels}}{2} - 1$), is computed by:

$$\text{inv\_freq} = \frac{1}{10000^{\left(\frac{2i}{\text{channels}}\right)}}$$

This calculation produces a series of frequencies that decrease exponentially across the dimensions. For each time step or position t in the sequence, the positional encoding components are calculated using sine and cosine functions:

$$\text{pos\_enc\_a} = \sin(t \cdot \text{inv\_freq})$$

$$\text{pos\_enc\_b} = \cos(t \cdot \text{inv\_freq})$$

The final positional encoding for each time step t is then constructed by concatenating these sine and cosine components:

$$\text{pos\_enc} = [\text{pos\_enc\_a}; \text{pos\_enc\_b}]$$

This concatenation yields a positional encoding vector for each time step, alternating between sine and cosine values. Each frequency corresponds to a specific dimension within the embedding space. The dual use of sine and cosine ensures that the encoding captures both phase and amplitude information, which can enhance the model's ability to discern and interpret sequence order.

2.3 Noise Schedule

Given initial noise level β_{init} and final noise level β_{end}, the noise schedule is a linear interpolation over N steps from β_{init} to β_{end}. The noise level at step i can be expressed as:

$$\beta_i = \beta_{\text{init}} + \frac{i}{N-1}(\beta_{\text{end}} - \beta_{\text{init}})$$

where $i = 0, 1, 2, \dots, N-1$ and N is the total number of steps.

2.4 Network Architecture

We use **UNet** as our network backbone. **UNet**

- inc: ConvBlock
  - Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  - GroupNorm(1, 32, eps=1e-05, affine=True)
  - GELU(approximate='none')

  - Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  - GroupNorm(1, 32, eps=1e-05, affine=True)

- down1: DownSample
  - MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  - ConvBlock
    * Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    * GroupNorm(1, 32, eps=1e-05, affine=True)
    * GELU(approximate='none')
    * Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    * GroupNorm(1, 32, eps=1e-05, affine=True)
  - ConvBlock
    * Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    * GroupNorm(1, 64, eps=1e-05, affine=True)
    * GELU(approximate='none')
    * Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    * GroupNorm(1, 64, eps=1e-05, affine=True)
  - embedded_layer: Sequential
    * SiLU()
    * Linear(in_features=128, out_features=64, bias=True)

- sa1: SelfAttention
  - MultiheadAttention
    * NonDynamicallyQuantizableLinear(in_features=64, out_features=64, bias=True)
  - LayerNorm((64,), eps=1e-05, elementwise_affine=True)
  - Sequential
    * LayerNorm((64,), eps=1e-05, elementwise_affine=True)
    * Linear(in_features=64, out_features=64, bias=True)
    * GELU(approximate='none')
    * Linear(in_features=64, out_features=64, bias=True)

- down2: DownSample
  - MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  - ConvBlock
    * Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    * GroupNorm(1, 64, eps=1e-05, affine=True)
    * GELU(approximate='none')
    * Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    * GroupNorm(1, 64, eps=1e-05, affine=True)
  - ConvBlock
    * Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    * GroupNorm(1, 128, eps=1e-05, affine=True)
    * GELU(approximate='none')
    * Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

    - GroupNorm(1, 128, eps=1e-05, affine=True)
  - embedded_layer: Sequential
    - SiLU()
    - Linear(in_features=128, out_features=128, bias=True)

- sa2: SelfAttention
  - MultiheadAttention
    - NonDynamicallyQuantizableLinear(in_features=128, out_features=128, bias=True)
  - LayerNorm((128,), eps=1e-05, elementwise_affine=True)
  - Sequential
    - LayerNorm((128,), eps=1e-05, elementwise_affine=True)
    - Linear(in_features=128, out_features=128, bias=True)
    - GELU(approximate='none')
    - Linear(in_features=128, out_features=128, bias=True)

- bot1: ConvBlock
  - Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  - GroupNorm(1, 64, eps=1e-05, affine=True)
  - GELU(approximate='none')
  - Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  - GroupNorm(1, 64, eps=1e-05, affine=True)

- up1: UpSample
  - Upsample(scale_factor=2.0, mode='bilinear')
  - ConvBlock
    - Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    - GroupNorm(1, 128, eps=1e-05, affine=True)
    - GELU(approximate='none')
    - Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    - GroupNorm(1, 128, eps=1e-05, affine=True)
  - ConvBlock
    - Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    - GroupNorm(1, 64, eps=1e-05, affine=True)
    - GELU(approximate='none')
    - Conv2d(64, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    - GroupNorm(1, 32, eps=1e-05, affine=True)
  - embedded_layer: Sequential
    - SiLU()
    - Linear(in_features=128, out_features=32, bias=True)

- sa3: SelfAttention
  - MultiheadAttention
    - NonDynamicallyQuantizableLinear(in_features=32, out_features=32, bias=True)
  - LayerNorm((32,), eps=1e-05, elementwise_affine=True)

- Sequential
  - LayerNorm((32,), eps=1e-05, elementwise_affine=True)
  - Linear(in_features=32, out_features=32, bias=True)
  - GELU(approximate='none')
  - Linear(in_features=32, out_features=32, bias=True)

- up2: UpSample
  - Upsample(scale_factor=2.0, mode='bilinear')
  - ConvBlock
    - Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    - GroupNorm(1, 64, eps=1e-05, affine=True)
    - GELU(approximate='none')
    - Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    - GroupNorm(1, 64, eps=1e-05, affine=True)
  - ConvBlock
    - Conv2d(64, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    - GroupNorm(1, 32, eps=1e-05, affine=True)
    - GELU(approximate='none')
    - Conv2d(32, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    - GroupNorm(1, 16, eps=1e-05, affine=True)
  - embedded_layer: Sequential
    - SiLU()
    - Linear(in_features=128, out_features=16, bias=True)

- sa4: SelfAttention
  - MultiheadAttention
    - NonDynamicallyQuantizableLinear(in_features=16, out_features=16, bias=True)
  - LayerNorm((16,), eps=1e-05, elementwise_affine=True)
  - Sequential
    - LayerNorm((16,), eps=1e-05, elementwise_affine=True)
    - Linear(in_features=16, out_features=16, bias=True)
    - GELU(approximate='none')
    - Linear(in_features=16, out_features=16, bias=True)

- outc: Conv2d(16, 1, kernel_size=(1, 1), stride=(1, 1))

- label_embedding: Embedding(10, 128)

3 Techniques

3.1 Techniques 1: Classifier Free Guidance

During training, Classifier-Free Guidance requires training two models: an unconditional generation model and a conditional generation model. However, these two models can be represented by the same model architecture. During training, it is only necessary to nullify the condition with a certain probability.

At inference time, the final result can be obtained through linear extrapolation of both conditional and unconditional generation. The quality of generation can be adjusted by the guidance coefficient, balancing the realism and diversity of the generated samples.

The primary benefit of CFG is that it allows the model to generate high-fidelity samples without the need for a separate network or additional classifier to guide the generation process. This simplifies the architecture and reduces the computational burden associated with maintaining and training multiple networks. Furthermore, Classifier Free Guidance provides a way to adjust the fidelity and diversity of generated samples dynamically by altering a single guidance scale parameter during inference. This flexibility is particularly useful in applications where control over the sample characteristics is crucial, offering a balance between adherence to the training data and creative variations of the output.

Algorithm 3 Conditional sampling with classifier-free guidance

```
Require: $w$: guidance strength
Require: $c$: conditioning information for conditional sampling
Require: $\lambda_1, \ldots, \lambda_T$: increasing log SNR sequence with $\lambda_1 = \lambda_{\min}$, $\lambda_T = \lambda_{\max}$
 1: $z_1 \sim \mathcal{N}(0, I)$
 2: for $t = 1, \ldots, T$ do
 3:     Form the classifier-free guided score at log SNR $\lambda_t$
 4:     $\tilde{\epsilon}_t = (1 + w)\epsilon_\theta(z_t, c) - w\epsilon_\theta(z_t)$
 5:     Sampling step (could be replaced by another sampler, e.g., DDIM)
 6:     $\hat{x}_t = (z_t - \sigma_{\lambda_t}\tilde{\epsilon}_t)/\alpha_t$
 7:     if $t < T$ then
 8:         $z_{t+1} \sim \mathcal{N}(\hat{\mu}_{\lambda_{t+1}}(z_t, x_t), (\sigma^2_{\lambda_{t+1}}/\alpha_{\lambda_t})^{1-v}(\sigma^2_{\lambda_{t+1}}/\alpha_{\lambda_t})^v)$
 9:     else
10:         $z_{t+1} = \hat{x}_t$
11:     end if
12: end for
13: return $z_{T+1}$
```

3.2 Techniques 2: Exponential Moving Average

In the training of diffusion models, the Exponential Moving Average (EMA) is employed to smooth the updates of model parameters. This technique updates the weight w by blending the previous weights w_{old} with the newly obtained weights w_{new}, where β is a coefficient between 0 and 1 that represents the retention ratio of the old weights. EMA helps to stabilize the training process and enhances the generalization ability of the model.

The update formula is given by:

$$w = \beta \cdot w_{\text{old}} + (1 - \beta) \cdot w_{\text{new}}$$

4 Experiments

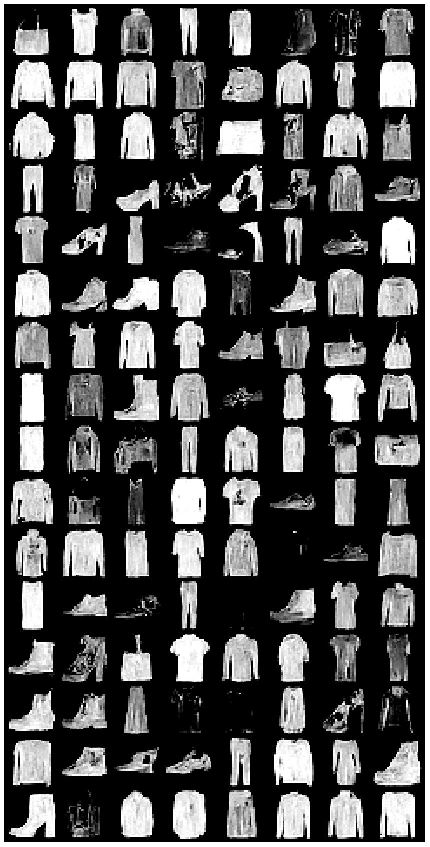
In this section, our objective is to evaluate and compare the performance of various enhancements made to the basic model, utilizing the FASHION-MNIST dataset. This dataset has been resized to 28x28 pixels and is categorized into 10 distinct classes. Our analysis will focus on three model variants:

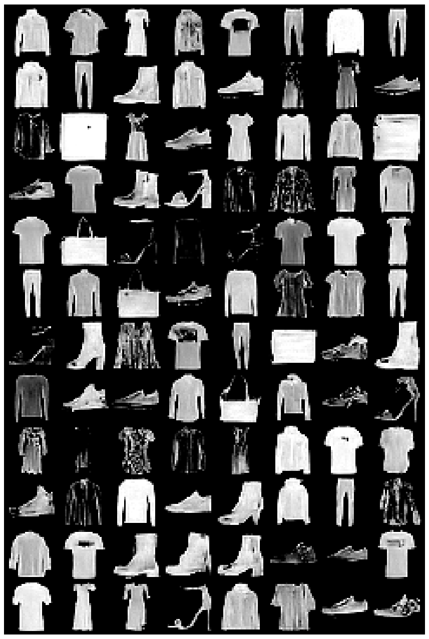
- Basic Model: This serves as our baseline for comparison.
- Basic Model with Classifier Free Guidance: This variant incorporates classifier-free guidance to potentially improve model fidelity by reducing reliance on labeled data during training.
- Basic Model with Classifier Free Guidance and Exponential Moving Average (EMA): This model further includes EMA, which smooths model parameters over time, aiming to stabilize training and enhance the quality of generated outputs.

4.1 Qualitative Results

4.1.1 Best State of Each Method

Figure 2 showcases the best generated images for each model variant at their optimal epoch. The models are evaluated based on the visual quality and fidelity of the generated images.


(a) Basic Model


(b) DDPM with Guidance

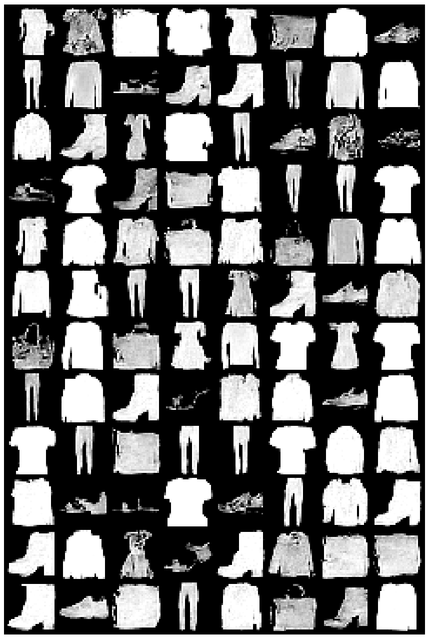
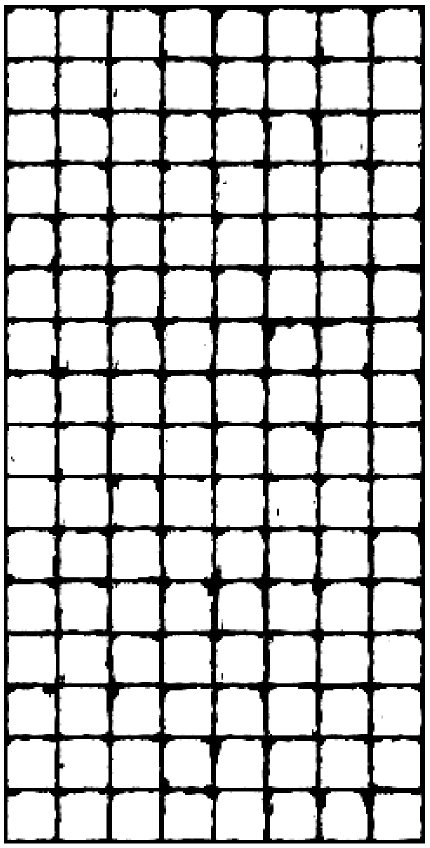

(c) DDPM with Guidance and EMA

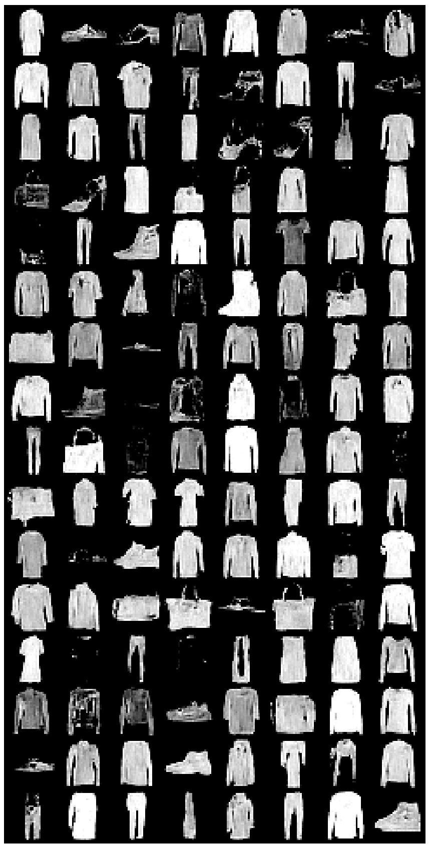
Figure 2: Best generated images for each model variant. From left to right: (a) Basic Model, (b) DDPM with Classifier-Free Guidance, and (c) DDPM with Classifier-Free Guidance and EMA.

4.1.2 Evolution of Images with Epochs

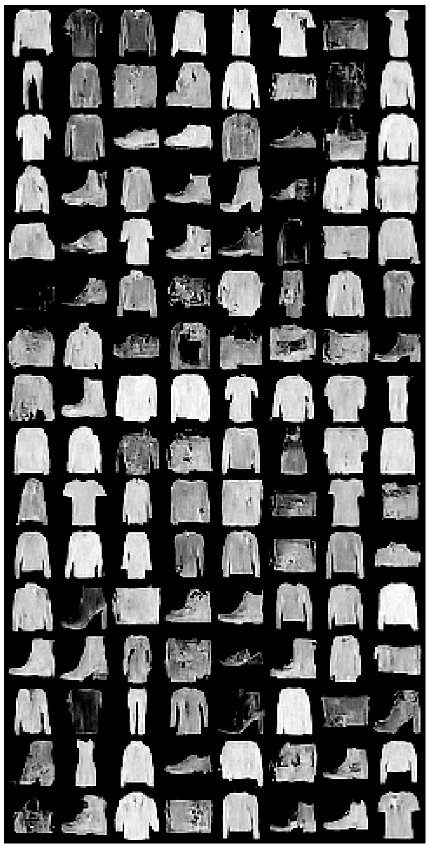
To visualize the training progression, Figure 5 illustrates the evolution of generated images across different epochs for each model variant. This provides insights into how quickly and effectively each model converges to produce high-quality images. (a) Basic Model Epoch 5 (b) Basic Model Epoch 10 (c) Basic Model Epoch 15



(a) Epoch 5

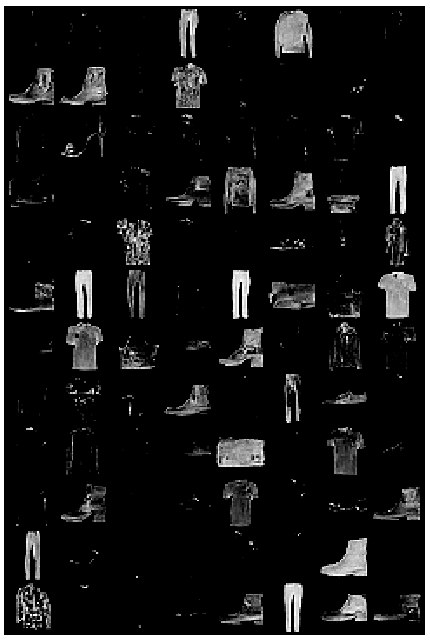


(b) Epoch 10

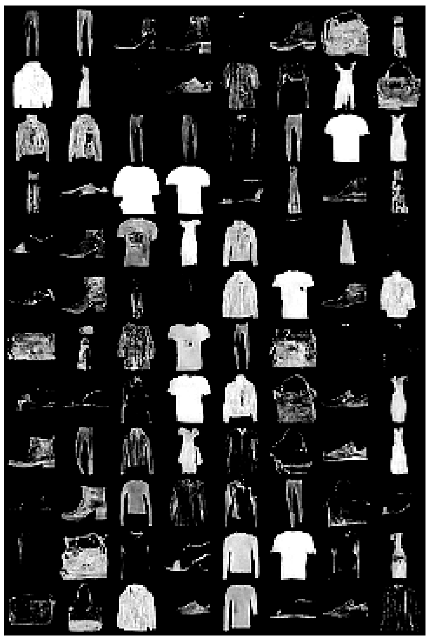


(c) Epoch 15

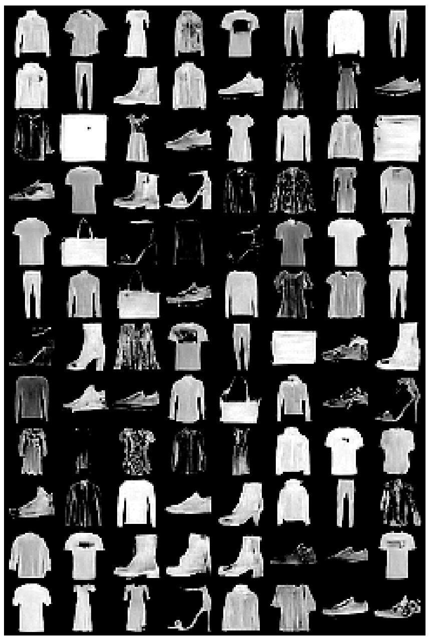
Figure 3: Evolution of generated images across epochs for Basic Model



(a) Epoch 5



(b) Epoch 10



(c) Epoch 15

Figure 4: Evolution of generated images across epochs for Basic Model DDPM with Classifier-Free Guidance

4.1.3 Epoch Comparison for Generating Perfect Images

Table 1 summarizes the number of epochs required by each model variant to generate high-quality images. The criteria for determining "perfect" images are based on visual assessment and the quality metrics used

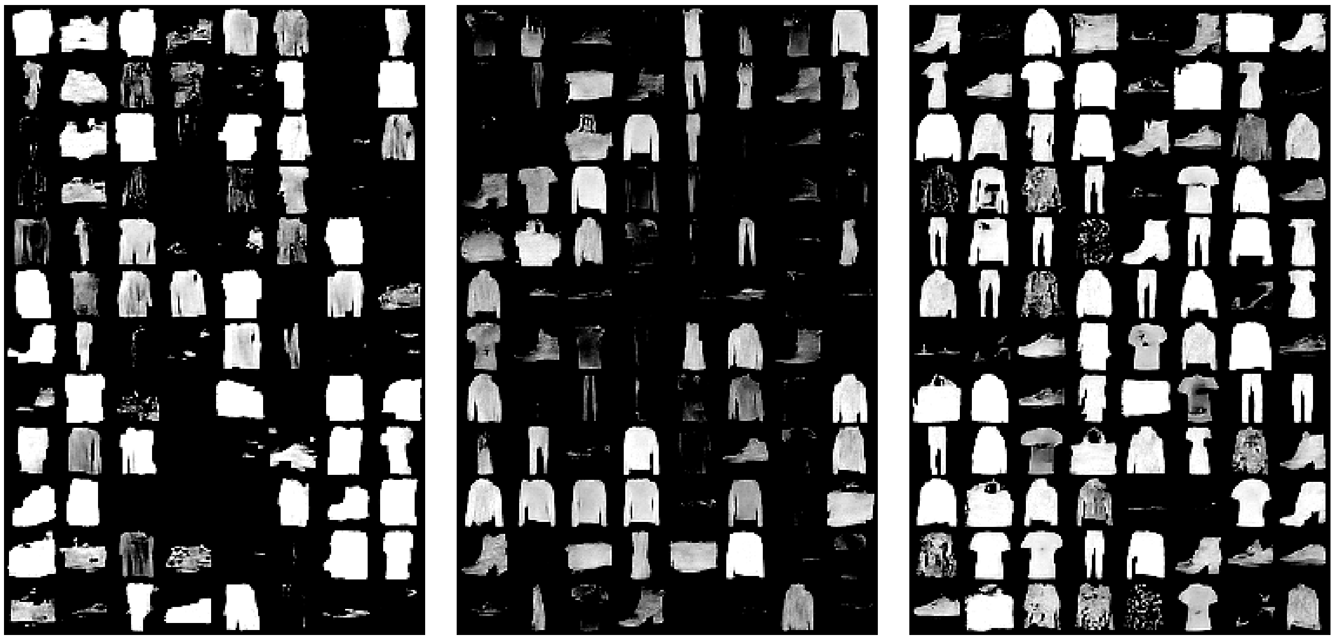

(a) Epoch 3 (b) Epoch 6 (c) Epoch 10

Figure 5: Evolution of generated images across epochs for DDPM with Classifier-Free Guidance and EMA.

during training.

Model Variant	Epochs to Perfect Image
Basic Model	17
DDPM with Classifier-Free Guidance	15
DDPM with Classifier-Free Guidance and EMA	11

Table 1: Number of epochs required to generate high-quality images for each model variant.

4.1.4 Discussion and Analysis

The results in Table 1 indicate a clear improvement in the efficiency of the model training process when advanced techniques are applied:

- **Basic Model**: The basic diffusion model required 17 epochs to generate high-quality images. This baseline performance illustrates the model's capability without additional enhancements.
- **DDPM with Classifier-Free Guidance**: Introducing classifier-free guidance reduced the number of epochs needed to 15. This improvement can be attributed to the enhanced guidance during the generation process, which helps the model converge faster.
- **DDPM with Classifier-Free Guidance and EMA**: Further incorporating the Exponential Moving Average (EMA) technique led to the model achieving high-quality images in just 11 epochs. The EMA helps stabilize the training process and improve the generalization of the model, contributing to faster convergence and better performance.

These findings highlight the efficacy of advanced techniques in enhancing the training efficiency and quality of diffusion models. The synergistic application of classifier-free guidance and Exponential Moving Average (EMA) markedly decreases the number of epochs needed to achieve high-quality image generation. Additionally, the use of EMA and classifier-free guidance contributes to more stable training throughout the process.

References

[1] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 8780–8794. Curran Associates, Inc., 2021. URL `https://proceedings.neurips.cc/paper`$_f iles/paper/2021/file/49ad23d1ec9fa4bd8d77d02681df5cfa - Paper.pdf$.

[2] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2021. URL `https://openreview.net/forum?id=qw8AKxfYbI`.

[3] 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. URL `https://proceedings.neurips.cc/paper`$_f iles/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b - Paper.pdf$.

[4] Tero Karras, Miika Aittala, Jaakko Lehtinen, Janne Hellsten, Timo Aila, and Samuli Laine. Analyzing and improving the training dynamics of diffusion models, 2024.

[5] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2256–2265, Lille, France, 07–09 Jul 2015. PMLR. URL `https://proceedings.mlr.press/v37/sohl-dickstein15.html`.