Classifier Guidance Image Generation on CIFAR-10 with DDPM

Chijie An, Zhihan Qin, Steven Zhang

Abstract

The current surge in generative models has significantly impacted various industries. Among these models, diffusion models have emerged as particularly effective. The generative models are typically capable of random sampling and generating random outputs, yet we seek to explore the feasibility of conditional control within the generative model framework. This endeavor is pivotal for steering the outputs of generative models, departing from purely random generation.

Our study leverages the DDPM (Diffusion Probabilistic Models). We meticulously replicated and employed classifier guidance within the existing model framework. Our primary endeavor lies in unraveling the impact of fixed classifiers on the guidance and optimization of image generation processes. We aim to unravel the nuanced interplay between classifier guidance and the enhancement of generative capabilities within DDPM. Code is available at https://github.com/ChijieAn/CV-final-project `https://github.com/ChijieAn/CV-final-project`.

1 Introduction

Denoising diffusion-based generative models represents a pioneering approach in the realm of generative modeling. At their core, these models leverage the principle of denoising to learn the underlying data distribution. The fundamental premise involves a step-by-step process of adding controlled noise to an input image, transforming it into a random noisy version. Subsequently, the model learns to iteratively denoise this corrupted version, aiming to recover the original image.

The novelty lies in this denoising process, which serves as the crux of learning to generate data. By effective denoising the perturbed images, the model gradually learns the complex patterns and structures inherent in the dataset. This approach aligns with the broader objective of generative modeling – to capture and synthesize high-dimensional data distributions faithfully.

Amidst various techniques within denoising diffusion models, the Diffusion Probabilistic Models (DDPM [1]) have emerged as a compelling methodology for

their robustness in handling noise and capturing intricate data features. DDPM introduces a diffusion process that progressively transforms an image from a random noisy state to its intended representation. This gradual refinement mechanism through diffusion aligns well with the denoising principles, enabling DDPM to excel in generating high-fidelity images.

DDPM stands out not only for its ability to progressively sample images but also for its underlying architecture, often featuring a UNet-like structure. This architecture enables the model to learn rich hierarchical representations, allowing for intricate and accurate reconstructions during the diffusion process.

An innovative addition to DDPM is the incorporation of Classifier Guidance, a technique that revolutionizes the control and conditioning aspects of generative models [2]. This augmentation facilitates conditional image generation without necessitating an entire model retraining. The essence lies in leveraging an additional, separately trained classifier to dictate the generated classes or conditions, offering more flexible and fine-grained control over the generation process.

The distinct advantage of Classifier Guidance over classifier-free guidance methodologies is evident in its efficiency. Unlike the latter, which demands the complete retraining of the diffusion model, Classifier Guidance solely requires the training of an external classifier. This approach significantly saves time and computational resources, making it a more practical and scalable solution for controlling and conditioning the generated outputs.

The subsequent methodology section will delve into the specifics of the trained datasets and the intricate design of our proposed model, illustrating its implementation and showcasing its capabilities in detail.

2 Methodology

2.1 Data

We harnessed the CIFAR-10 dataset, a widely acknowledged benchmark in computer vision, comprising 60,000 32x32 color images across ten classes. This dataset served a dual purpose in our study:

Classifier Training: We utilized CIFAR-10 to train distinct classifiers for various classes. Each classifier was trained to discern and classify images belonging to a specific category, enabling us to implement Classifier Guidance within our DDPM framework effectively.

DDPM Training: Additionally, the CIFAR-10 dataset played a pivotal role in training our Denoising Diffusion Probabilistic Model (DDPM). This involved leveraging the rich diversity and complexity of CIFAR-10's images to facilitate the learning process of DDPM's generative capabilities.

Using CIFAR-10 for both classifier and DDPM model training not only ensured a consistent dataset but also allowed for comprehensive testing and evaluation across various experimental setups.

2.2 DDPM and Classifier Guidance

The DDPM (Denoising Diffusion Probabilistic Model) can be mainly broken down into two processes: the forward process and the backward sampling process. The forward process is also called the diffusion process, which adds noise to the image, Both the forward and backward processes are parametrized Markov chains, with Markov properties. We denote the forward transition probability as $q(x_{t+1}|x_t)$. This transition probability is determined by some pre-set parameters β:

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

Here β is the variance for each sampling step and it's in the interval $(0, 1)$. And we often call β as the variance scheduler for the DDPM model, and it's pre-setted before the training of the model and it's fixed during the training process.

Then, by implementing a reparametrization trick and setting $\alpha_t = 1 - \beta_t$, we can get the following representation:

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I)$$

The backward transition probability we use to denoise and generate images can be represented as $p(x_t|x_{t+1})$. And this backward transition probability is what we want to get, so we use a neural network to approximate it, denoted as $p_\theta(x_t|x_{t+1})$. Our goal of training this neural network, which is U-net for the DDPM, is to let $p_\theta(x_t|x_{t+1})$ approximate $q(x_t|x_{t+1})$. However, it is hard to calculate $q(x_t|x_{t+1})$ directly, so we use calculate $q(x_t|x_{t+1}, x_0)$ instead, and we can calculate it using Bayers rule.

$$q(x_{t-1}|x_t, x_0) = \frac{q(x_t|x_{t-1}, x_0)}{q(x_t|x_0)}$$

So we can define loss function as:

$$\begin{aligned} D_{\mathrm{KL}}(q(x_T|x_0) \parallel p_\theta(x_T)) + \sum_{t=2}^{T} E_{q_\theta(x_{t-1}|x_0)} \left[D_{\mathrm{KL}} \left(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1}|x_t) \right) \right] \\ - E_{q_\theta(x_1|x_0)} \log p_\theta(x_0|x_1) \end{aligned}$$

This loss function is mainly composed of the KL divergence between distributions of $q(x_{t-1}|x_t, x_0)$ and $p_\theta(x_{t-1}|x_t)$ at each time step. Minimizing the KL divergence is equivalent to minimizing the difference between these two distributions for each time step.

Then, we can again apply simplification and reparametrization and further simplify the expression for the loss function. We can finally get the following expression:

$$L_{t-1}^{simple} = E_{x_0, \varepsilon \sim \mathcal{N}(0,1)} \left[\left\| \varepsilon - \varepsilon_\theta \left(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\varepsilon, t \right) \right\|^2 \right]$$

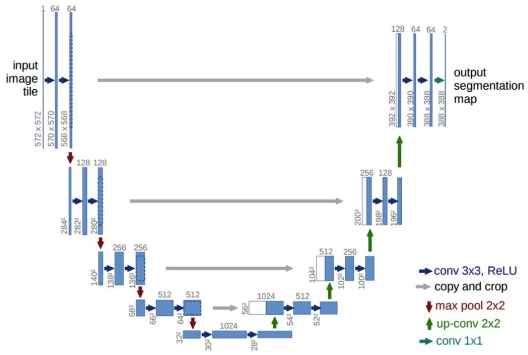

Figure 1:
The Unet Model Framework

And here, instead of predicting the mean value μ_θ using the U net model, we only predict ε_θ, the noise we add to the previous image in each time step. And when it comes to coding, this ε_θ is exactly the output of our Unet model.

We can view the backward process as a Markov chain composed of a series of Gaussian distributions parametrized by neural networks:

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

Then, we observe that the images we generated are from different classes. So how can we control the generation process? We tried to implement a method called classifier guidance generation. We further train a classifier $logp_\phi(y|x_t)$, which can predict which class of image will we get after finishing the generation process. We change the expression of the backward transition by adding a gradient guidance term of the $\nabla_{x_t} logp_\phi(y|x_t)$.

$$p_{\theta,\phi}(x_{t-1}|x_t, y) = \mathcal{N}(\mu_\phi(x_t) + \gamma \Sigma_\theta \nabla_{x_t} \log p_\theta(y|x_t), \Sigma_\theta)$$

By doing this, at each step of the backward sampling process, we are guiding the model in the direction of generating an image that maximizes the likelihood of being classified as the class we set. Here the hyperparameter γ, which usually takes value in the interval $(1, 100)$, is called the classifier scale, which represents classifier guidance strength in the sampling process. In this project, we launched experiments on trying different γ and analyzed how the hyperparameter influenced the results.

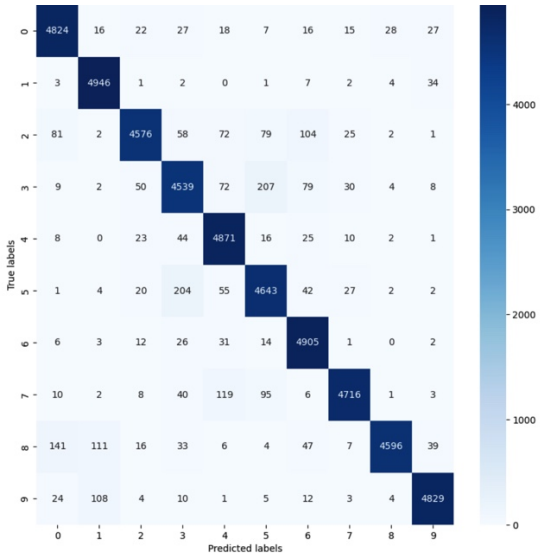

Figure 2:
Confusion matrix of the classifier used in the guiding process

3 Training of the Classifier

3.1 Training Methods

We attempted two different ways of classifier guidance. For the first approach, we trained a classifier on the original cifar-10 dataset. This classifier reached an accuracy of 94.89 on the cifar-10 dataset. Then we used this classifier to guide the generation process.

However, during the backward sampling process, the image is always under noise, so we think it's more accurate to train the classification model on the noisy image. Also, as we set the total number of timesteps in the Gaussian Diffusion process as 1000, it's the noise features will be very different from the images that added around 200 timesteps of noise and those which are added 800 timesteps of noise. So, we train a set of classifiers using the following strategy: We split the total 1000 timesteps into 10 different intervals, with the interval length of 100 each. Then, we create 10 different datasets, each consisting of images in the cifar-10 dataset which are added a random number of timesteps of noise in the corresponding interval. Each dataset consists of 5000 images which is the same size as the whole cifar-10 dataset. For instance, to create the first dataset, we traverse the cigar-10 dataset, for each image, we create a random number $\alpha \in (1, 100)$, and add α timesteps of noise to that image.

Then, we train 10 different classifers based on these 10 different datasets. During the denoising process when we reach a certain timestep, we use the classifier that corresponds to that step to guide the generation of the image.

3.2 Training Results

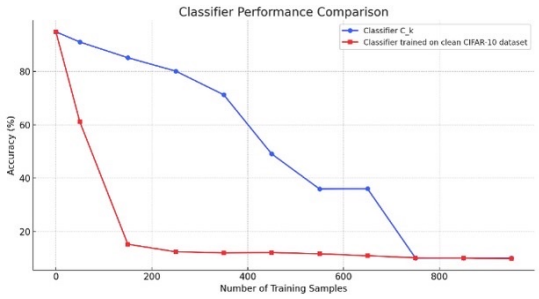

Figure 3:
The Accuracy of classifier C_t on dataset D_t

As we can see in the Figure, as the steps of adding noises, the accuracy of our classifier model trained on the corresponding noisy dataset decreases. For the convenience of notation, we denote the dataset with the noise between the range of $100t$ timesteps and $100(t+1)$ timesteps as D_t, and we denote the classifier trained on the corresponding dataset as C_t. We observe that the accuracy of C_1 on D_1 is over 80 percent, however, the accuracy of $C_{t\geq 7}$ on $D_{t\geq 7}$ is only around 10 percent, which has no big difference compared to random guess. This phenomenon can reflect the fact that the more noise we add to the image, the harder it is to preserve the original class features of the image.

And when we compare the accuracy of the classifier C_t on dataset D_t and the accuracy of the classifier trained on the clean cifar-10 dataset, which we will denote as C_{clean} for convenience, on the dataset D_t, we observed a huge difference. Just after adding around 150 timesteps of noise, the accuracy of the classifier C_{clean} on dataset D_t is already reduced to around 10 percent, which is almost a random guess. Meanwhile, the accuracy of the classifier C_t on dataset D_t is still above 80 percent. This is easy to explain, for the classifier C_t are particularly trained on the datasets D_t, they are capable of capturing the features on dataset D_t better than the classifier C_{clean} trained on the clean dataset.

And after 700 timesteps, the accuracy of classifiers C_t on datasets D_t also reduces to around 10 percent, which is similar to the accuracy of classifier C_{clean} on datasets D_t. This is because as we add more and more timesteps of noise, the noise covers the original features of each class in images in cifar-10 dataset, so the classifier can actually learn very little and its accuracy is almost the same as a random guess, which is supposed to be around 10 percents.

4 Experiments of Changing the Classifier Scale

We changed the classifier scale γ in the following sequence of numbers: $1, 25, 50, 75, 100$. For $\gamma = 1$, it represents there's no classifier guidance, as is seen

in Figure. We can observe that the image generated without the classifier and the model generated images without preferring one specific class. However, after viewing the result, we can see that the result is more inclined to classes $0, 1, 3, 9$, which indicates that even without the guidance of the classifier, the model still inclines to some of the classes.

For $\gamma = 25, 50, 75$, we can see that as γ increases, the sooner images that are generated by the model converge to the label we set. When $\gamma \geq 25$, the images generated after 1000 denoising steps are likely to be 100 percent belong to the class we set. However, after doing some human observations, we still find some mistakes, like an image in the result is classified as the class "planes" by the classifier, is actually a 'vehicle'. This might imply that there are some inaccuracies in the classifier. Also, there are still a lot of generated images that can not be recognized by human observation but are classified to the intended class. This may be due to the generated image indeed containing some of the features of the desired class, but due to the fact that we only trained the Unet model on cifar-10 for 200 epochs, the training dataset and the number of training epochs are limited. The model is still not good enough But generally speaking, the image generated by these values of γ are all of a decent quality.

Then, for the images generated by setting $\gamma = 100$, we observe that they are not as clear as the images generated by setting γ to a smaller number. This may be due to the classifier having too strong an influence on the generation process of the images, which disturbs the learned features of the original model.

All the generated images are in Appendix of this report.

5 Experiments of Using Different Guidance Strategies

In our project, we implemented two classifier guidance strategies:

The first strategy: use the classifier C_{clean} to guide the entire generation process.

The second strategy: use the classifier c_t to guide the corresponding timestep's generation process.

Our conclusion is, theoretically speaking, because C_t performs much better than C_{clean} on datasets D_t, it is expected that the second strategy will outperform the first strategy. However, after we ran the experiments, we found out that when fixing other hyperparameters γ and the label we want to generate, the images generated by the second strategy are always more vague and hard to recognize by the human eye. Also, when fixing $\gamma \geq 50$, the image generated by the first strategy can be 100 percent classified to the class we desire using C_{clean}. However, when $\gamma = 50$, the generation accuracy of the second strategy is only around 60 percent, and when we set $\gamma = 75$, the generation accuracy is also only around 70 percent, which is much lower compared to the first strategy. So far, we deduce that this phenomenon may be due to the training errors when we train the classifier C_t on a very noisy dataset, and C_t fails to capture the features

on the noisy dataset, and thus isn't able to provide the right guidance in the generation process. However, some deeper reasons still need further exploration.

6 Conclusion and Further Direction

In conclusion, this project explores generating images under the guidance of a classifier. However, there are still many more interesting directions, such as how to generate images using sentences and how to get more precise control during the image generation process. Also, we observe that the difference between the classifiers we trained plays a huge part in the generation of the images, and building a relationship of how the difference in the classifiers influence the image generation process may be an interesting topic.

7 References

References

[1] Ho, Jonathan; Jain, Ajay; Abbeel, Pieter. *Denoising Diffusion Probabilistic Models*. June 2020. `https://arxiv.org/abs/2006.11239`. eprint arXiv:2006.11239. *Keywords: Computer Science - Machine Learning; Statistics - Machine Learning.*

[2] Dhariwal, Prafulla; Nichol, Alex. *Diffusion Models Beat GANs on Image Synthesis*. May 2021. `https://arxiv.org/abs/2105.05233`. eprint arXiv:2105.05233. *Keywords: Computer Science - Machine Learning; Computer Science - Artificial Intelligence; Computer Science - Computer Vision and Pattern Recognition; Statistics - Machine Learning.*

[3] Liu, Xihui; Park, Dong Huk; Azadi, Samaneh; Zhang, Gong; Chopikyan, Arman; Hu, Yuxiao; Shi, Humphrey; Rohrbach, Anna; Darrell, Trevor. *More Control for Free! Image Synthesis with Semantic Diffusion Guidance*. December 2021. `https://arxiv.org/abs/2112.05744`. eprint arXiv:2112.05744 [cs.CV]. *Subjects: Computer Vision and Pattern Recognition (cs.CV); Graphics (cs.GR).* DOI: 10.48550/arXiv.2112.05744.

[4] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, Ben Poole. *Score-Based Generative Modeling through Stochastic Differential Equations*. arXiv preprint arXiv:2011.13456, 2020. `https://arxiv.org/abs/2011.13456`. *Subjects: Machine Learning (cs.LG); Machine Learning (stat.ML).*

8 Appendix

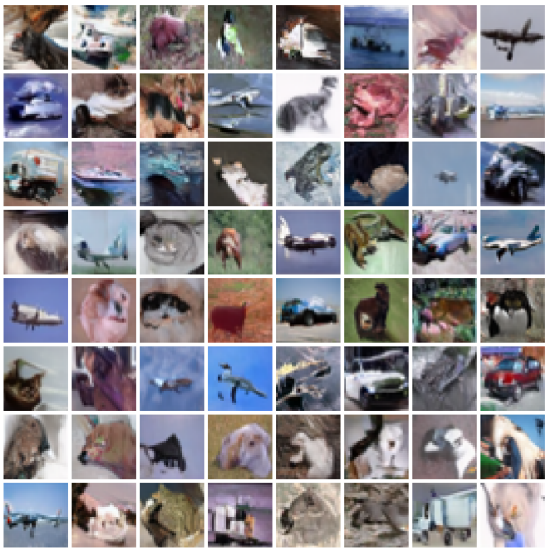

Figure 4: Without Classifier

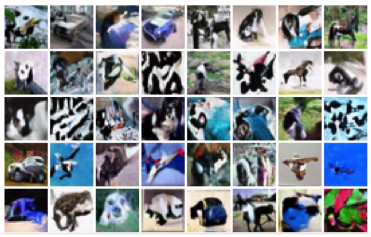

Figure 5: Classifier Scale100
Air Plane (all images generated as the correct label)

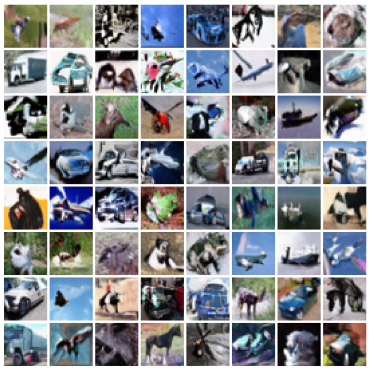

Figure 6: Classifier Scale75
Air Plane (all images generated as the correct label)

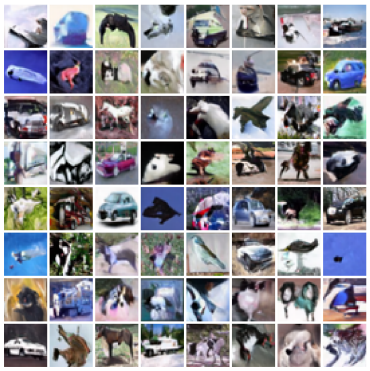

Figure 7: Classifier Scale50
Air Plane (all images generated as the correct label)

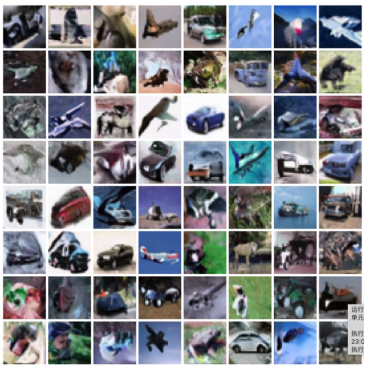

Figure 8: Classifier Scale 25
Air Plane (all images generated as the correct label)

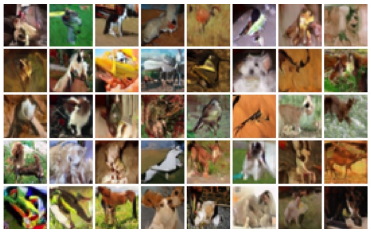

Figure 9: Classifier Scale100
Deer (all images generated as the correct label)

Figure 10: Classifier Scale75
Deer (all images generated as the correct label)

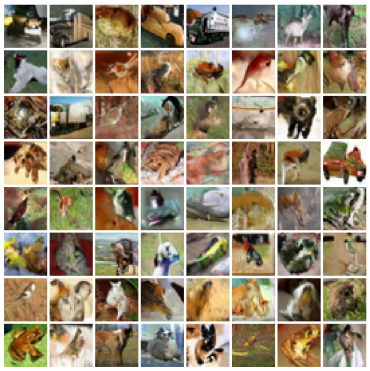

Figure 11: Classifier Scale50
Deer (all images generated as the correct label)

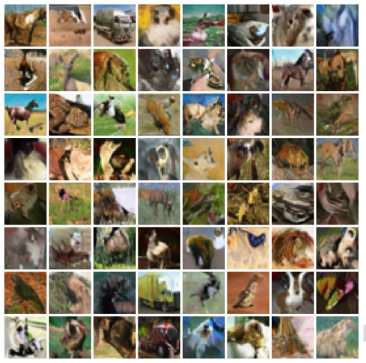

Figure 12: Classifier Scale25
Deer (all images generated as the correct label)

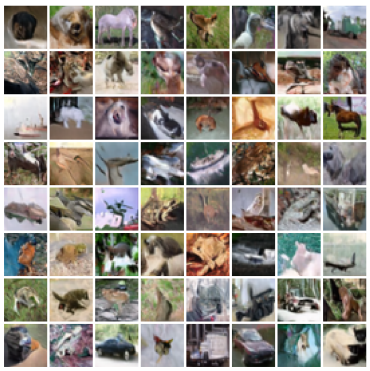

Figure 13: Classifier Scale75
Bird (all images generated as the correct label)

Figure 14: Classifier Scale50
Bird (all images generated as the correct label)

Figure 15: Classifier Scale50
Cat (all images generated as the correct label)

Figure 16: Classifier Scale75
Cat (all images generated as the correct label)

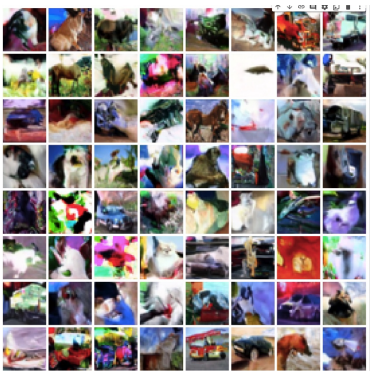

Figure 17: Guidance under classifier C_t Classifier Scale75
Air Plane (Around 70 percents of images generated as the correct label)

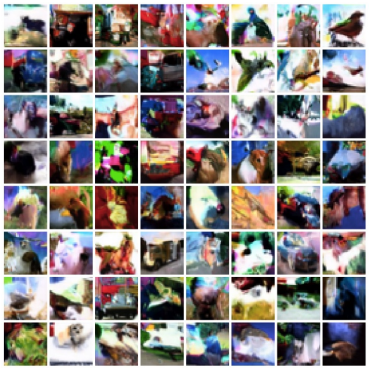

Figure 18: Guidance under classifier C_t Classifier Scale50
Air Plane (Around 60 percents of images generated as the correct label)