TRAINING DIFFUSION MODELS

Deep learning - fall semester

Authors
Tamásy András - EA824Y
Takáts Bálint - PUWI1T
Hujbert Patrik - D83AE5

Contents

1 Introduction

In this project, we delved into the practical application and training of Denoising Diffusion Probabilistic Models (DDPMs), guided by the pioneering studies of Jaiming et al. [1] and Ho et al. [2]. Our investigative framework was anchored on the Caltech Birds dataset [3], a comprehensive collection featuring high-resolution images of common bird species found across the United States. This dataset, encompassing over ten thousand images spanning two hundred species, is enriched with detailed annotations and bounding boxes, offering a robust basis for our analysis.

Our approach involved a thorough experimental data analysis paired with strategic data augmentation to enrich the dataset further. Central to our methodology was implementing the renowned U-net architecture [4], enhanced with self-attention mechanisms. This allowed us to leverage the intricate patterns within the dataset effectively.

Throughout our model training process, we utilized Weights and Biases for meticulous experiment tracking. This not only provided us with real-time insights into the model's performance but also allowed for a streamlined and efficient iterative improvement process. Our work aims to contribute to the growing body of knowledge in machine learning, specifically in the application of DDPMs to complex image datasets.

The inspiration and the backbone of our code were built on the Keras documentation [5], which we rewrote in torch. We also built uppon the work of András Béres on generating flowers [6].

In the development of our project, we utilized advanced tools such as ChatGPT and Grammarly, which played a pivotal role in various aspects including code generation, visualization, grammar enhancement, effective phrasing, text production, and LaTeX formatting. These tools significantly enhanced the quality of our work, proving to be an indispensable component of the project.

2 Exploration

Our dataset, illustrated in Figure 1, exemplifies the diversity and richness of the Caltech Birds dataset. As shown in the figure, the dataset provides high-quality, detailed images of various bird species for our analysis.

To maintain consistency and focus on the primary subjects of our study, we preprocess the images from the Caltech Birds dataset. This preprocessing involves cropping the images using the provided bounding boxes to primarily focus on the birds, effectively minimizing background distractions. Subsequently, we resize the images to a uniform dimension of $64 \times 64 \times 3$. This standardization is crucial for our analysis, as it ensures that all images are of comparable scale and resolution, thereby maintaining the high quality of the dataset. Figure 2 shows a subsample of the processed images.

The preprocessing stage concludes with the standardization of the images. This critical step involves adjusting the pixel values of each image to have a mean of 0 and a standard deviation of 1. After preprocessing we split the dataset into train and test sets. The train set contains 9430 samples, while the test set contains 2350 images.

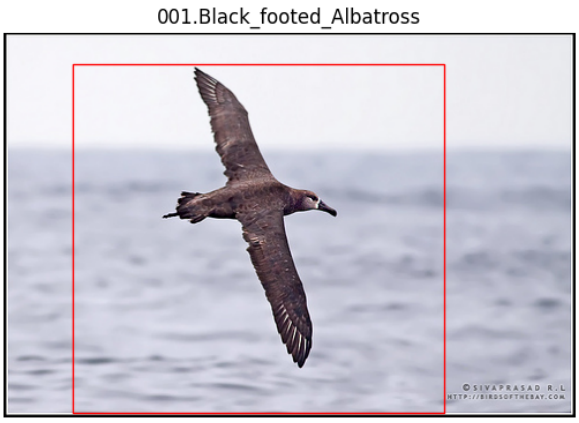

Figure 1: A sample from the Caltech dataset

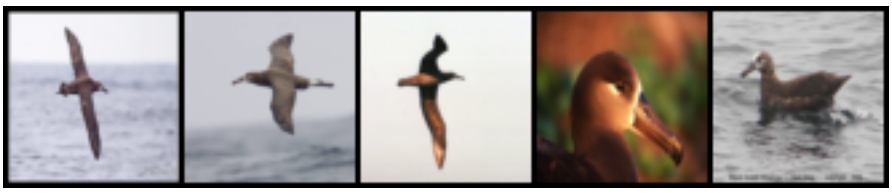

Figure 2: Four samples from the dataset after preprocessing.

3 Implementation

Firstly we implemented a relatively big U-Net model containing self attention layers, skip connections between the down and up blocks of the model. As input the model takes the noise images and the timestep of the noise schedule. We pass the timestamp indicating the level of noise through multiple blocks of the model, requiring fewer skip connections between individual blocks. Due to the size of the model, the training process is relatively slow. This model can be considered a baseline in terms of finding an architecture suitable for our limited resources.

Therefore, as the next step, we implemented a much smaller U-Net architecture based on the article by Beres Andras [6]. The goal was to find an optimal model with significantly fewer trainable parameters than the baseline model, making it faster to train, while not substantially compromising the quality of the generated images. The new model was implemented based on the experiences described in the article. Key differences include:

- We did not use self-attention layers.
- Layers have significantly fewer channels.
- Only the first layer receives the embedded timestamp as input, which is concatenated with the input noisy images.
- We added more skip connections to facilitate information flow in the network, eliminating the need for multiple injections of the timestamp.

This way, we managed to reduce the size of the model by orders of magnitude, speeding up the training process. We then further improved the model by adding attention layers.

During the implementation of diffusion steps, we primarily followed the approach outlined in the work of Ho et al. [2], incorporating both training and sampling algorithms from their paper. In subsequent improvements, we implemented the computationally less demanding implicit sampling based on the Keras documentation [5]. Another enhancement was replacing the linear scheduler with a cosine-based scheduler.

4 Training

We think that the most important parts of our codebase are the network definition, the metrics and the sampling process. For sampling, we use an algorithm based on Ho et al.'s work [2].

We train the network with the hyper-parameters described in Table 1:

Hyperparameter	**Value**
batch_size	32
epochs	200
lr	0.0003
beta_start	0.0001
beta_end	0.02
noise_steps	1000
s_parameter	0.008
ddim_sample_step	5

Table 1: Key Hyperparameters for the Model

Our training loss curves show a steady convergence shown in Figure 7 in the appendix. We also observe high GPU utilization throughout the training.

Completing our training on the Google Cloud Platform (GCP) was a tough achievement, especially considering the prevalent GPU shortage that posed significant challenges. Fortunately, we were able to obtain access to a P100 GPU, which we utilized to the fullest extent possible under the circumstances.

We logged the whole training duration to Wandb and we also created periodic samples from the network. Figure 3 illustrates the output after 1000 steps.

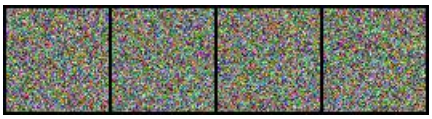

Figure 3: Random noise at the beginning of training.

TIt took an extensive amount of time to train the baseline model, and even after 100,000 steps, the generated images did not meet expectations. This is illustrated in Figure 4. These samples are from our initial training run for Milestone 2. The training of our next simplified model was also very slow in the early stages. The reason training took so long is that there was a bug in our code. After fixing it training became much faster and our results were way better too.

The Backward Denoising Process works by removing noise from the original image over 1000 steps. However, the denoised image is only saved at every 100th step in Figure 5.

Figure 4: Samples after 100.000 steps.

This results in 10 snapshots of the noise reduction process. Therefore, we observe the noise removal over these 10 steps.

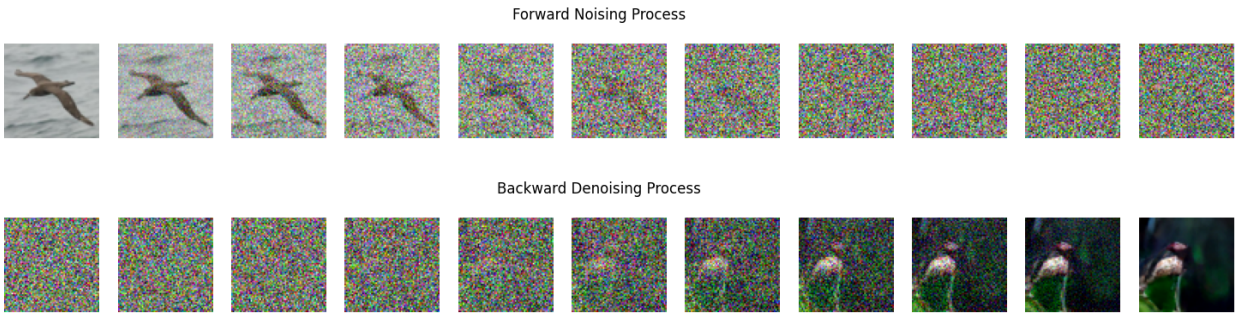

Figure 5: Forward and backward noising process

5 Evaluation

The Kernel Inception Distance (KID) and the Fréchet Inception Distance (FID) are two significant metrics used for evaluating the performance of generative models, particularly in the field of image generation. The KID metric measures the similarity between the distribution of generated images and the distribution of real images using a kernel-based technique. The FID metric, on the other hand, assesses the difference in statistical distribution between generated images and real images using the Inception model's feature space.

Metric	**Value**
KID	0.2089
FID	92.47

Table 2: KID and FID Evaluation Metrics

As indicated in Table 2, the KID value of 0.2089 and the FID value of 92.47 provide insights into the quality of the generated images. The KID value, being relatively low, suggests a good similarity between the distributions of generated and real images. However, the high FID value indicates a notable difference between the statistical properties of generated images compared to real images, implying potential areas for improvement in the model's performance.

In addition to these quantitative assessments, we also extensively relied on visual inspection to evaluate the generated images. One key observation we made was that the use of Attention mechanisms in our model led to a slight blurring effect in the background, while enhancing the details and focus on the birds themselves.

6 Results

For our longest train, we let the network learn for over 28.000 steps. Figure 6 presents our results. To reach this level of accuracy we had to train our network for over 8 hours on a P100 GPU on the GCP.

Figure 6: Best training run after 28.000 steps.

7 Summary

This semester, we successfully implemented and trained Denoising Diffusion Probabilistic Models (DDPMs) to generate realistic bird images, utilizing the Caltech Birds dataset. Our approach involved developing and optimizing a U-Net model with self-attention mechanisms, overcoming challenges in computational efficiency and model training. The models were evaluated using KID and FID metrics alongside visual inspections, leading to the generation of high-quality images. This project not only enhanced our understanding of DDPMs in image synthesis but also demonstrated the practical application of deep learning in generating detailed and realistic images.

References

[1] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," 2022.

[2] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," 2020.

[3] C. W. at al., "Caltech birds," Tech. Rep., 2011.

[4] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015.

[5] Keras, "Denoising diffusion implicit models." [Online]. Available: https://keras.io/examples/generative/ddim/

[6] B. András, "Clear diffusion in keras." [Online]. Available: https://github.com/beresandras/clear-diffusion-keras

8 Appendix

Training logs.

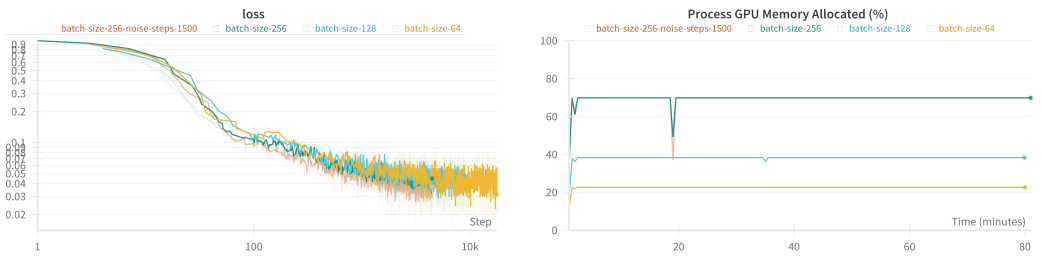

Figure 7: Loss with respect to different Hyperparameters and GPU memory utilization with different batchsizes.

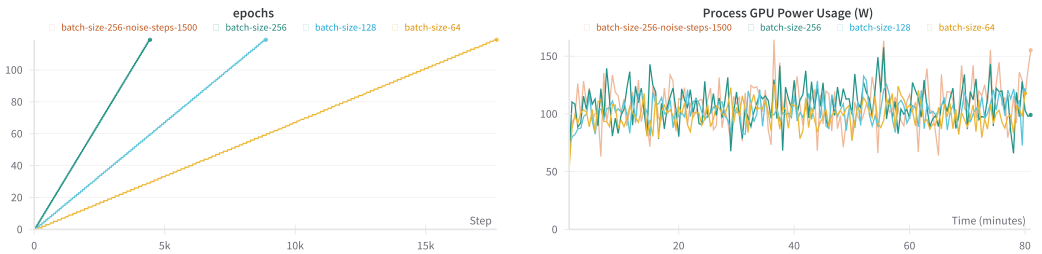

Figure 8: Steps with different batchsizes and power usage.