Project P4 : Dreaming Data

Deepak Singh
Department of Robotics Engineering
Worcester Polytechnic Institute
Email: dsingh1@wpi.edu

Sai Hitesh Viswasam
Department of Robotics Engineering
Worcester Polytechnic Institute
Email: sviswasam@wpi.edu

Samuel Markwick
Department of Robotics Engineering
Worcester Polytechnic Institute
Email: spmarkwick@wpi.edu

***Abstract*—This report presents our project on implementing the key concepts of diffusion models for the purpose of generating new training data for an image classifier network. Initially we trained a diffusion model implementing the forward and the reverse processes to denoise a noisy input into each of the classes from the CiFAR-10 dataset so that additional synthetic data is generated. Next, this synthetic data is augmented with the existing dataset and a classification neural network is trained using this data. We present a comparative study of the accuracy of the network with and without the augmentation to show the improved robustness due to the augmentation. Finally, we also performed Parameter ablation studies on the diffusion model to experiment with the effects of different parameter combinations and the effects of tuning the parameters is described.**

I. Introduction

The rapid advancements in deep learning have led to remarkable breakthroughs in tasks such as image generation, classification, and segmentation. However, the increasing complexity of neural networks often requires vast amounts of training data to achieve high accuracy and generalization. Traditional generative models like Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs), while effective, still face challenges in learning robust data distributions, especially when data is scarce or imbalanced. These models often suffer from issues such as mode collapse (in GANs) or blurry reconstructions (in VAEs), limiting their capacity to capture the full complexity of high-dimensional data distributions. **Diffusion models** [1], in contrast, offer a more robust approach to this challenge by leveraging a principled process of iterative denoising, which allows them to generate high-quality samples with fewer data requirements and a more stable training process.

Diffusion models generate new data by learning how to reverse a gradual noise-adding process, leading to highly realistic outputs. Recent works such as Denoising Diffusion Probabilistic Models (DDPMs) [2], has demonstrated that diffusion-based approaches can achieve results on par with or surpassing GANs in terms of image quality and diversity. Additionally, diffusion models exhibit advantages such as improved likelihood estimation, better mode coverage, and increased interpretability due to their close connection with probabilistic modeling.

In this work, we have implemented the Denoising Diffusion Probabilistic Model as described in the paper [2] to generate additional dataset for an image classifier network. The training of a DDPM network involves a forward process and a backward process which are described as below:

1) *Forward Process*: The forward process gradually corrupts the data by adding Gaussian noise at each time step. This process is formulated as a Markov chain, where at each step, the data becomes progressively noisier. Over a sequence of steps, the data eventually becomes indistinguishable from random noise. This noising process is described by a variance schedule that determines how much noise is added at each step, allowing the model to control the rate of degradation. Mathematically, it can be represented as:
   Given an input image x_0, the forward diffusion process creates noisy versions of the image at time step t as:

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

   where β_t controls the variance (noise level) at time step t. Here β_t is the noise scheduler term that generally follows a linear or a cosine distribution, with β_t gradually increasing over time.

2) *Backward Process*: The backward process aims to reconstruct the original data from pure noise by gradually removing the noise added in the forward process. The model learns to predict the noise at each step and then removes it to reverse the diffusion process step by step. This reverse process is parameterized by a neural network that estimates the noise at each time step. Mathematically, the backward process aims to generate samples x_0 by denoising the noisy sample x_T from the forward process. The model learns a conditional distribution:

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

   where μ_θ and Σ_θ are learned functions predicting the mean and variance of the image at each step t.

One key feature of DDPM is that instead of directly predicting the denoised data itself at each time step, the model is trained to predict the **noise** added at each step. The training objective minimizes the mean squared error (MSE) between the predicted noise and the actual noise added during the

forward process. The model learns to gradually reverse the corruption by predicting the noise that needs to be removed at each step. Therefore the objective function that the model tries to minimise is given by

$$L(\theta) = E_{x_0,\epsilon,t}\left[\|\epsilon - \epsilon_\theta(x_t, t)\|^2\right]$$

where $\epsilon_\theta(x_t, t)$ is the predicted noise by the model, and ϵ is the true noise added during the forward process.

Further, using the reparametrization trick on the normal distribution, the above equations for the noise addition and noise sampling can be expressed in terms of the input data x_0 itself instead of the data at intermediate time steps by simply introducing another variable α as:

$$x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1 - \bar{\alpha}_t}\, \epsilon$$

for the forward process and

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right) + \sigma_t z$$

for the backward process where

$$\alpha_t = 1 - \beta_t$$

$$\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$$

$\sigma_t^2 = \beta_t$, and $z \sim \mathcal{N}(0, I)$.

Using the above expressions, we can now implement and train a DDPM network to generate images for each subclass within the CiFAR-10 dataset. The further report is divided as follows. Section II describes the training and implementation of the Diffusion model, Section III describes the comparitive study of the performance of the classifier network with and without the data augmentation. Section IV describes the parameter ablation studies performed on the diffusion model.

II. DDPM Implementation and Training

The implementation and training of the DDPM network can be performed by following the algorithms describing the training and the noise sampling processes respectively, as presented in [2].

Checkpoint Linear Schedule: Linear schedule.

Checkpoint Cosine Schedule: Cosine schedule.

A. Hyper-parameters and Training Details

Table[I] shows the hyper-parameters we used for training the diffusion model with linear schedule.

The model was trained on CIFAR10 dataset. The dataset classes are as shown in the Table[II]

The outputs for each class from this training of diffusion model with linear scheduling are as shown in Fig[1]. The sampling steps used for this generation is same as shown in Table[I]

The input noise was generated using Deepak's date of birth, which is 12 Feb, 1997. So we set random seed as 12. The noise generated is as shown in Fig[2]

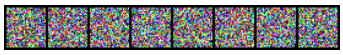

Fig. 2. Noise for inference

1) Loss Function: We used Mean Squared Error (MSE) loss to compute the difference between the added noise and the predicted noise in our diffusion model. In this setup, Gaussian noise is added to the data at each timestep, and the model is trained to predict this noise. The MSE loss minimizes the squared difference between the actual noise and the predicted noise, guiding the model to learn the reverse diffusion process more accurately.

The MSE loss is defined as shown in Equation 1:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{\epsilon}_\theta(x_t, t) - \epsilon\right)^2 \tag{1}$$

where $\hat{\epsilon}_\theta(x_t, t)$ is the predicted noise by the model at timestep t for the input x_t, and ϵ is the actual Gaussian noise added to the data. The objective is to minimize this loss to ensure the model correctly predicts the noise added during the forward diffusion process.

III. Effect of Augmentation on Classification

After training the model with linear scheduling, we then trained a CNN classifier using categorical cross entropy loss.

Algorithm 1 Training

repeat
  Sample $x_0 \sim q(x_0)$
  Sample $t \sim \text{Uniform}(\{1, \ldots, T\})$
  Sample noise $\epsilon \sim \mathcal{N}(0, I)$
  Take gradient descent step on:

$$\nabla_\theta \left\|\epsilon - \epsilon_\theta\left(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon, t\right)\right\|_2^2$$

until converged

Algorithm 2 Sampling

Sample $x_T \sim \mathcal{N}(0, I)$
for $t = T$ to 1 **do**
  if $t > 1$ **then**
    Sample $z \sim \mathcal{N}(0, I)$
  else
    Set $z = 0$
  end if
  Compute $x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\sqrt{1-\alpha_t}}{1-\bar{\alpha}_t}\epsilon_\theta(x_t, t)\right) + \sigma_t z$
end for
return x_0

Hyperparameter	Value
Sampling steps (T)	500
Img_Size	32
Epochs	100
Batch Size	64
Optimizer	ADAM
Learning Rate	1e-4
Guidance (w)	1.8
Grad Clip	1.0
Beta1	1e-4
BetaT	0.028
Dropout	0.15

TABLE I
DIFFUSION MODEL TRAINING HYPERPARAMETERS

Class Label	Class Name
0	Airplane
1	Automobile
2	Bird
3	Cat
4	Deer
5	Dog
6	Frog
7	Horse
8	Ship
9	Truck

TABLE II
CIFAR-10 CLASS LABELS AND NAMES

We used the same model that we created in Project-P1 and used the Custom CNN implementation. For each class, we generated 5016 images using the above trained diffusion model and then we trained the Custom CNN model on the combined CIFAR10 and augmented dataset. The accuracy of the model increased by around 3%. A comparison of validation accuracy for 40 epochs is shown below (with and without augmentation) in Fig[3]

The validation accuracy we achieved after augmentation was **57.12%** as compared to 54% without augmentation.

Now, the confusion matrix for the case of no augmentation is shown below

	plane	car	bird	cat	deer	dog	frog	horse	ship	truck
plane	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
car	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
bird	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000
cat	0.333	0.000	0.000	0.000	0.333	0.000	0.333	0.000	0.000	0.000
deer	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
dog	0.000	0.000	0.000	0.333	0.000	0.667	0.000	0.000	0.000	0.000
frog	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
horse	0.000	0.000	0.000	0.000	0.000	0.333	0.000	0.667	0.000	0.000
ship	0.000	0.000	0.000	0.333	0.000	0.000	0.000	0.000	0.667	0.000
truck	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

TABLE III
CONFUSION MATRIX WITHOUT AUGMENTATION

IV. PARAMETER ABLATION STUDIES

The confusion matrix for the case of augmented images is shown in Table[IV]

The updated confusion matrix indicates improved performance for certain classes, such as *ship*, *truck*, *plane*, and *car*, compared to the original model. However, to evaluate the model's performance more thoroughly, we need to compute relevant metrics, including accuracy, precision, recall, and the F1 score.

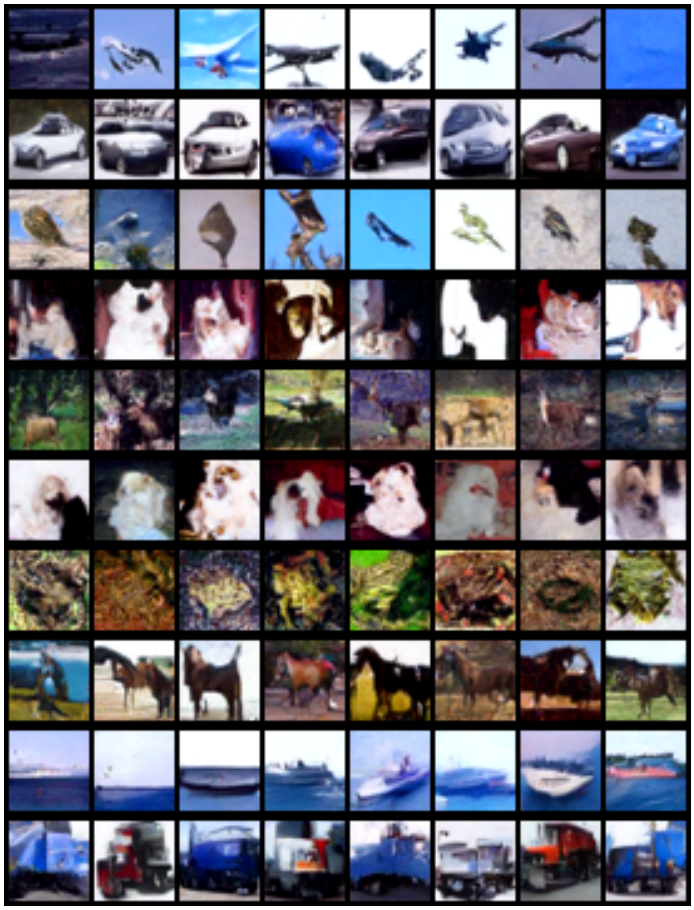

Fig. 1. Diffusion model training output with linear scheduling. The classes are shown in sequence as shown in Table[II. From top to down, the classes are Airplane, Automobile, Bird, Car, Deer, Dog, Frog, Horse, Ship and Truck

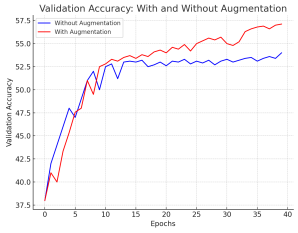

Fig. 3. Validation Accuracy Comparison

- **Accuracy**: The overall accuracy measures the percentage of correctly classified instances. It can be calculated as:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

- **Precision**: Precision measures the percentage of positive predictions that were correct for each class.

	plane	car	bird	cat	deer	dog	frog	horse	ship	truck
plane	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
car	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
bird	0.000	0.000	0.100	0.100	0.700	0.100	0.000	0.000	0.000	0.000
cat	0.300	0.100	0.000	0.000	0.300	0.000	0.300	0.000	0.000	0.000
deer	0.000	0.100	0.100	0.100	0.200	0.100	0.200	0.100	0.100	0.000
dog	0.000	0.100	0.000	0.300	0.000	0.600	0.000	0.000	0.000	0.000
frog	0.000	0.000	0.000	0.000	0.000	0.000	0.900	0.100	0.000	0.000
horse	0.000	0.000	0.000	0.100	0.000	0.200	0.000	0.700	0.000	0.000
ship	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.900	0.100
truck	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.100	0.900

TABLE IV
CONFUSION MATRIX FOR TRAINING WITH AUGMENTATION

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

- **Recall**: Recall measures how many of the actual positive instances were correctly classified for each class.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

- **F1 Score**: The F1 score is the harmonic mean of precision and recall, which balances the two metrics. It is given by:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Changes in Model Performance:

- **Improved Classes (Ship, Truck, Plane, Car)**: The increased accuracy for these classes, up to 0.900 for *ship* and *truck*, suggests that the model is better at distinguishing these categories. The diffusion-based augmentation has likely helped the model capture more distinct features, improving both precision and recall.
- **Slight Performance Decrease (Bird, Cat, Deer, Dog)**: A minor decrease in performance for these classes may be due to the synthetic images introducing some noise or making it harder for the model to generalize well. This could result in a slightly lower precision or recall for these specific classes.

Overall, the introduction of diffusion-generated images has generally improved model performance, particularly for vehicle-related classes, where the augmented data provided clearer distinctions between categories. The slight decrease in other classes may reflect the increased complexity of the synthetic data.

V. ABLATION STUDY AND COSINE SCHEDULE TRAINING

We trained another version of our diffusion model, but this time with cosine schedule. The hyperparameters for training the model remain same as shown in Table[I]
In a linear schedule, the noise level varies from a value β_{min} to β_{max} linearly with respect to a time step $t \in 1, 2, ..., T$ as:

$$\beta_t = \beta_{min} + \frac{t}{T}(\beta_{max} - \beta_{min})$$

For a cosine schedule, the noise level α_t is defined as a function of time step t using a cosine function. The variance

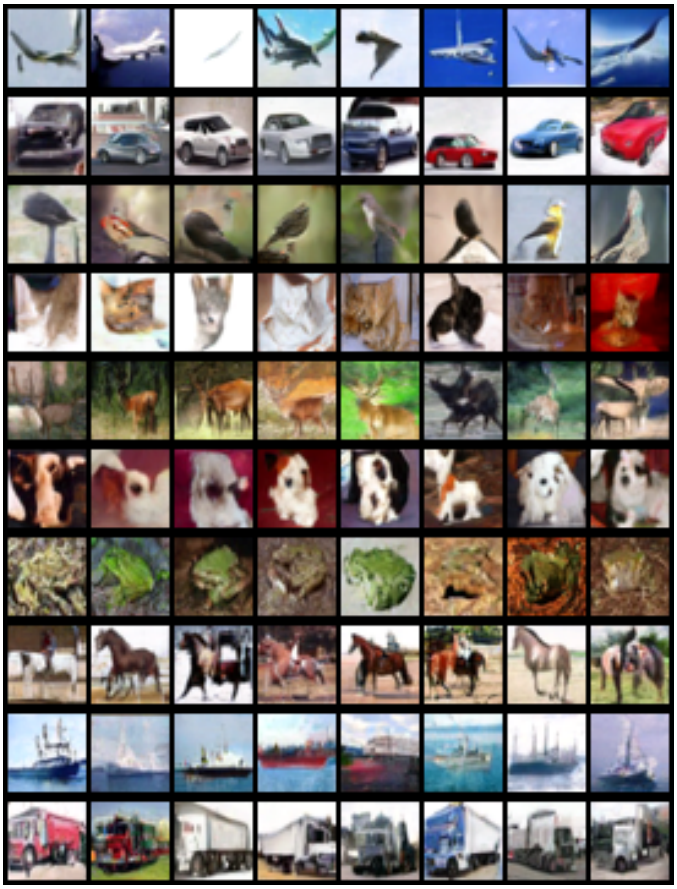

Fig. 4. Output from Diffusion Model for each class, using cosine scheduler

β_t is derived from this α_t. The equations for the scheduler are derived from [3].

$$\alpha_t = \frac{\cos^2\left(\frac{\pi}{2} \cdot \frac{1+s \cdot t/T}{1+s}\right)}{\cos^2\left(\frac{\pi}{2} \cdot \frac{1+s}{s}\right)}$$

where s is a small constant (often set to 0.008) which is included in the equations for numerical reasons so that it ensures the cosine schedule smoothly starts from a non-zero value. Here T is the total number of diffusion steps and α_t is the cumulative product of $1-\beta_t$ and hence β_t can be recovered as:

$$\beta_t = 1 - \frac{\alpha_t}{\alpha_{t-1}}$$

The cosine variance schedule introduces a smoother, non-linear increase in the noise level, inspired by the cosine function. It can be more effective than the linear schedule as it increases the noise more slowly at the beginning and more sharply near the end, aligning better with the data's structure.

Fig[4] shows the output after training the diffusion model with cosine scheduler. We used the same noise input for inference, as we did with the linear schedule.
We also did ablation studies for diffusion models with linear and cosine schedules and varied the number of timesteps

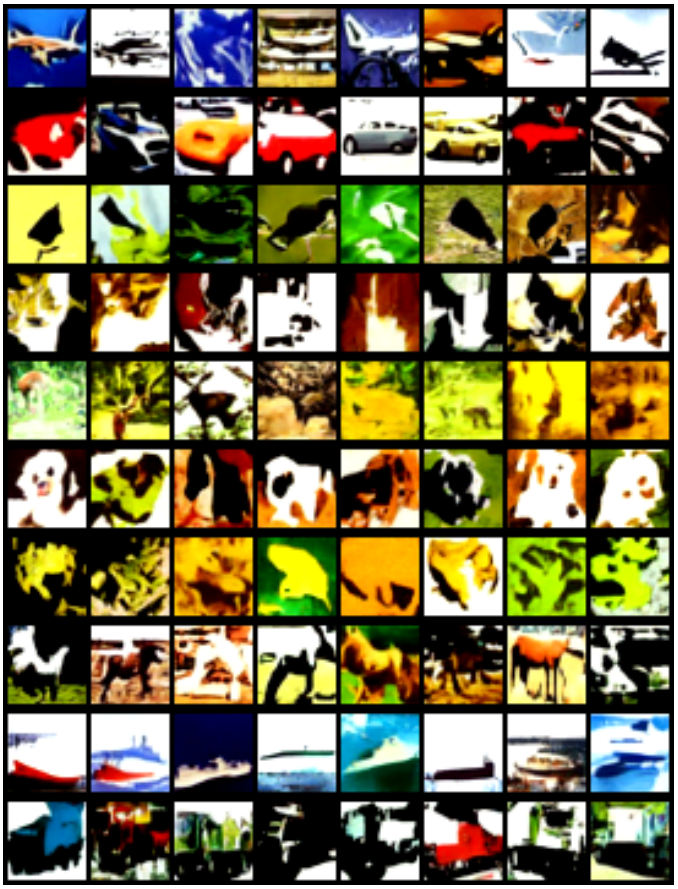

Fig. 5. Outputs generated by diffusion model trained with linear scheduling, on 100 sampling steps. Image ordering remain same like previous images

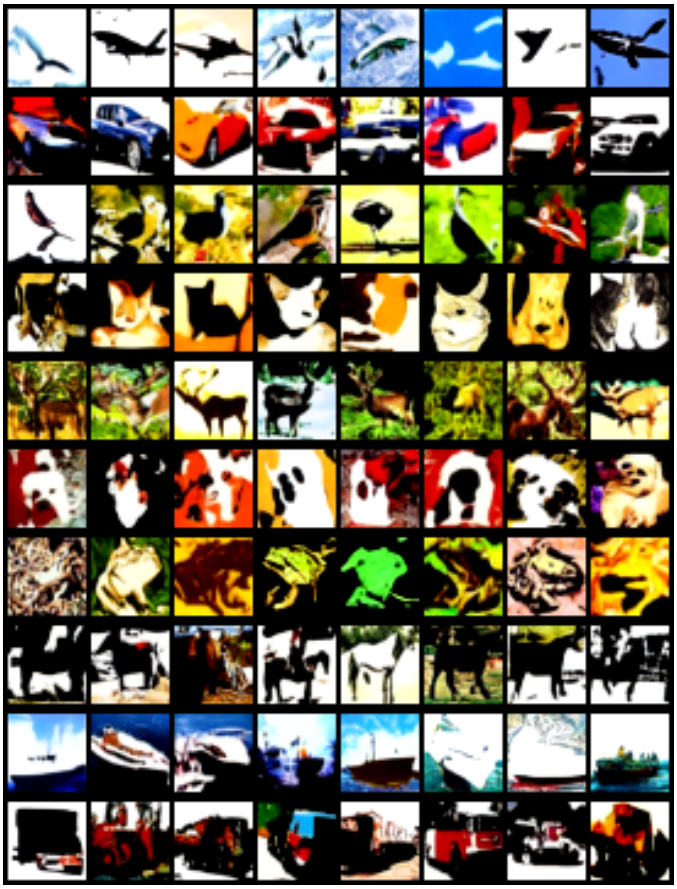

Fig. 6. Outputs generated by diffusion model trained with cosine scheduling, on 100 sampling steps. Image ordering remain same like previous images

during inference. The generations shown in Fig[1] and Fig[4] use 500 sampling steps. We now show what happens when we reduce sampling steps.

A. *Outputs with 100 sampling steps*

Fig[5] shows the output of diffusion model with linear scheduling, when we use 100 sampling steps in inference.

Now, for the diffusion model trained with cosine scheduling, output on 100 sampling steps is shown in Fig[6]

VI. Conclusion

In this project, we implemented diffusion model. We trained them and performed inference for generating data. We observed that model with cosine scheduler gave better outputs than the model with linear scheduling, and this is expected given how quickly image deteriorates in linear scheduling. We also used the images generated by this approach for a CNN based multi-class classifier, and achieved a 3% higher classification accuracy, by augmenting training data with diffusion generated images. After this, we experiment with the sampling steps and observe that if we reduce sampling steps, the output quality decreases, for linear as well as cosine scheduling.

References

[1] Sohl-Dickstein, Jascha, Weiss, Eric A., Maheswaranathan, Niru, and Ganguli, Surya. "Deep Unsupervised Learning using Nonequilibrium Thermodynamics." *arXiv preprint arXiv:1503.03585*, 2015. Available at: https://arxiv.org/abs/1503.03585.

[2] Ho, Jonathan, Jain, Ajay, and Abbeel, Pieter. "Denoising Diffusion Probabilistic Models." *arXiv preprint arXiv:2006.11239*, 2020. Available at: https://arxiv.org/abs/2006.11239.

[3] Nichol, Alex and Dhariwal, Prafulla. "Improved Denoising Diffusion Probabilistic Models." *arXiv preprint arXiv:2102.09672*, 2021. Available at: https://arxiv.org/abs/2102.09672.