Exploring Generative Modeling with Diffusion Models

António Figueiras
99402, MECD
Instituto Superior Técnico
Lisboa, Portugal

Tomás Juhos
99446, MMAC
Instituto Superior Técnico
Lisboa, Portugal

***Abstract*—**
This article investigates Denoising Diffusion Probabilistic Models (DDPMs) for image generation, offering a detailed examination of their theoretical foundation and practical implementation. This discussion covers the underlying mechanisms of DDPMs, which involve iteratively denoising a Gaussian noise distribution to produce complex images. The analysis is conducted using the CIFAR-10 dataset, focusing on metrics that encapsulate image quality and diversity. While DDPMs demonstrate potential, particularly in generating high fidelity images, the study highlights inherent challenges in model training and efficiency, suggesting areas for further research and development.

***Index Terms*—Denoising Diffusion Probabilistic Models (DDPMs), Image generation, Image quality metrics, Noise scheduling**

I. Introduction

In recent years, generative models have become a cornerstone of advancements in machine learning, driving significant progress in many fields, namely image synthesis. The concept of reversing a diffusion process was first introduced by Sohl-Dickstein et al. in 2015 [1], using techniques from non-equilibrium thermodynamics. Among the various generative approaches, Denoising Diffusion Probabilistic Models (DDPMs), introduced by Ho et al. in 2020 [2], have emerged as a highly promising image generation framework.
Central to DDPMs is the concept of iterative denoising, where a completely noisy data sample is progressively refined through multiple steps, each removing a small amount of noise to bring the sample closer to the final output. This process is akin to reversing a diffusion process, where noise is gradually added to images. These are the two main components of DDPMs, adding and removing noise from images, and will be explained in greater detail further ahead.
Throughout the years several improvements have been researched, from improvements to sampling speed [3], [4], to improvements in the generation quality and training effectiveness [5]. This report explores the theoretical foundations and implementation of two DDPMs (simple vs. improved), offering insights into some of their strengths and limitations. For our improved model we will implement Classifier-Free diffusion guidance, based on a later J. Ho and T. Salimans work [6], which allows the network to better distinguish images from different classes and offering an adjustable trade-off between quality and variety.

II. Theoretical Background

Denoising diffusion probabilistic models consist of two processes, the forward process and the reverse process, that given an image x_t will output a new one. The forward process consists of a fixed non-parametric Markov chain, that is defined by a fixed and predetermined noise schedule, and can be denoted as $q(x_t|x_{t-1})$. The reverse process consists of a parametrized Markov chain, where the transitions will be modeled by a neural network, and can be denoted as $p_\theta(x_{t-1}|x_t)$.

A. Forward Process

The objective of the forward process is to, given an image x_0, iteratively add Gaussian noise to it, to obtain a new image x_T, with the objective of transforming the original data distribution into a standard Gaussian distribution, over a chosen quantity of time steps T. We can mathematically define the forward diffusion process as:

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

where $\sqrt{1-\beta_t}x_{t-1}$ and $\beta_t\mathbf{I}$ are respectively, the mean, and the variance of the normal distribution, and x_t is the output. It's important to notice that we do not add the same amount of noise for each step, $\beta_1, ..., \beta_T$ is defined as the variance schedule and controls how much Gaussian noise is added. It is important that β_t consists of values between 0 and 1, so that the pixel intensities of x_t do not explode. In the original paper from 2020 [2], a linear variance schedule is used, with values ranging between 0.0001 and 0.02. To obtain the final x_T from x_0, we could iteratively apply the noise following the formula, however, for large T, this can be very computational expensive, mainly for training the backward process, and therefore it is important to notice that the forward process has a notable property that allows sampling x_t at

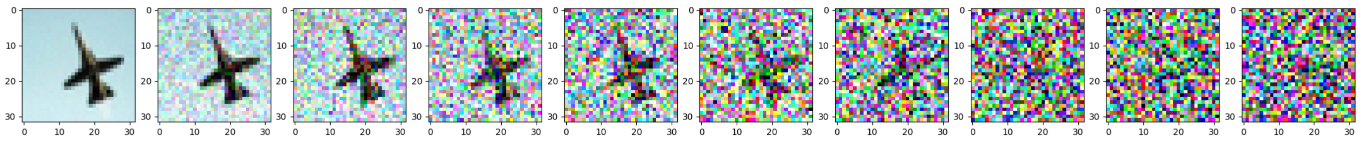

Fig. 1: Forward Diffusion process using a linear β schedule

an arbitrary t in closed form. Defining $\alpha_t = 1 - \beta_t$ and $\overline{\alpha}_t = \prod_{s=1}^{t} \alpha_s$, we can say that:

$$\begin{aligned} q(x_t|x_{t-1}) &= \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t\mathbf{I}) \\ &= \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon \\ &= \sqrt{\alpha_t}x_{t-1} + \sqrt{1-\alpha_t}\epsilon \\ &= \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\epsilon \\ &= \cdots \\ &= \sqrt{\overline{\alpha}_t}x_0 + \sqrt{1-\overline{\alpha}_t}\epsilon \end{aligned} \tag{2}$$

and therefore we can finally define:

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\overline{\alpha}_t}x_0, (1-\overline{\alpha}_t)\mathbf{I}) \tag{3}$$

The above expression will later be useful for training, since we will want to sample a random t and predict the added noise for that timestep t, without the need of calculating all the previous x_i, where $i < t$.
To conclude, in Figure 1 we can observe multiple iterations of an image throughout the forward process, iteratively adding noise to the pixel intensities.

B. Reverse Process

The objective of the reverse diffusion process is to reverse the forward diffusion process, denoising the given noisy images back to the original ones. We can mathematically define the reverse diffusion process as:

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t,t), \Sigma_\theta(x_t,t)) \tag{4}$$

We could then have two neural networks that would parameterize the mean and the variance of the normal distribution. However, as we said in the forward process, the authors of the paper from 2020 [2] chose to fix the variance to a defined schedule, so there is no need to train a second neural network, and therefore we consider $\Sigma_\theta(x_t,t) = \beta_t\mathbf{I}$. Nonetheless, it is worth noticing that later works also built models that could learn the variance of the reverse process, which allowed sampling with reduced computational cost and negligible differences in the quality of the generated images [5]. The training of the network is done by optimising the variational lower bound on the negative log likelihood:

$$\mathbb{E}[-\log(p_\theta)] \leq \mathbb{E}_q\left[-\log\frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}\right] =: L \tag{5}$$

After some clever formulations of the equation [5], shown in the paper [2], the authors arrive at the conclusion that the neural network must optimise the following formula,

$$\begin{aligned} \mathbb{E}_{x_0,\boldsymbol{\epsilon}}\Bigg[\frac{1}{2\sigma_t^2}\Bigg\|\frac{1}{\sqrt{\alpha_t}}\left(x_t(x_0,\boldsymbol{\epsilon}) - \frac{\beta_t}{\sqrt{1-\overline{\alpha}_t}}\right)\boldsymbol{\epsilon} - \\ -\boldsymbol{\mu}_\theta(x_t(x_0,\boldsymbol{\epsilon}),t)\Bigg\|^2\Bigg] \end{aligned} \tag{6}$$

Equation [6] reveals that our neural network must predict $\frac{1}{\sqrt{\alpha_t}}(x_t - \frac{\beta_t}{\sqrt{1-\overline{\alpha}_t}}\boldsymbol{\epsilon})$, given x_t. Since x_t is available as input to the model, we can reparametrize the above expression and obtain:

$$\boldsymbol{\mu}_\theta(x_t,t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\overline{\alpha}_t}}\boldsymbol{\epsilon}_\theta(x_t,t)\right) \tag{7}$$

This shows that for each iteration the neural network just needs to learn the noise added in that iteration by the forward diffusion process, since all of the other variables are already previously known. Therefore after some algebra we have that the function that the neural network needs to optimise is the mean-squared error between the actual noise at timestep t, ϵ, and the predicted noise $\epsilon_\theta(x_t,t)$, multiplied by a factor:

$$\mathbb{E}_{x_0,\boldsymbol{\epsilon}}\left[\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\overline{\alpha}_t)}\left\|\epsilon - \epsilon_\theta(x_t,t)\right\|^2\right] \tag{8}$$

However, the authors found out that removing the scaling factor was beneficial to sample quality, and also easier to implement and therefore the final used loss function is simply $\left\|\epsilon - \epsilon_\theta(x_t,t)\right\|^2$.
To conclude the theoretical background of the reverse diffusion process, we can finally observe that to remove noise at a certain timestep t, from a given image x_t, we can iterate through $p_\theta(x_{t-1}|x_t)$, whose formula can be obtained from the reparameterization trick for the gaussian distribution:

$$\begin{aligned} p_\theta(x_{t-1}|x_t) &= \mathcal{N}(x_{t-1}; \boldsymbol{\mu}_\theta(x_t,t), \beta_t\mathbf{I}) \\ &= \boldsymbol{\mu}_\theta(x_t,t) + \sqrt{\beta_t}\epsilon \\ &= \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\overline{\alpha}_t}}\boldsymbol{\epsilon}_\theta(x_t,t)\right) + \sqrt{\beta_t}\epsilon \end{aligned} \tag{9}$$

C. Algorithms

To implement the denoising diffusion probabilistic models, there are two algorithms that we need, the first one being, the training algorithm for the neural network, and the second one being a sampling algorithm, that takes pure noise images and generates new ones based on the learned $p_\theta(x_{t-1}|x_t)$. The training algorithm is the following:

Algorithm 1 Training

```
1: repeat
2:   $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3:   $t \sim \mathrm{Uniform}(\{1, \dots, T\})$
4:   $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5:   Take gradient descent step on
         $\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$
6: until converged
```

Fig. 2: Training Algorithm

This algorithm simply samples a image x_0 from our dataset, samples t from a uniform distribution and ϵ from a standard normal distribution, and does gradient descent on our objective function until it converges. The sample algorithm is the following:

Algorithm 2 Sampling

```
1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: for $t = T, \dots, 1$ do
3:   $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4:   $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: end for
6: return $\mathbf{x}_0$
```

Fig. 3: Sampling Algorithm

This algorithm starts by generating a pure noise image, using a multivariate normal distribution. Then it reverses the diffusion in T timesteps, applying the formula resulting from [9], with an exception for the last timestep since we do not want to add noise to our final image. The sampling algorithm allows us to create new images based on the dataset that was used for training.

D. Improvements

Since the original DDPM paper [2], several improvements have been researched over the years. Some improved the sampling efficiency [3], [4], allowing faster generation of images, others focus on simple modifications to DDPM structure that can achieve competitive log-likelihoods while maintaining high sample quality [5]. The improvement that we chose to focus and implement, was Classifier-Free Diffusion Guidance (CFG) [6], introduced by J. Ho, that was also part of the team that introduced DDPMs [2], and T. Salimans.
The idea behind CFG is to improve image generation if we have access to class labels in our dataset. Nowadays, models are trained to generate a big variety of images, ranging an extremely large set of possibilities and therefore, it is very important that we can not only ask the model to generate a certain class of images but also to differentiate images while training, since not doing so can lead to posterior collapse, mixing information from different classes, and ignoring conditional information.
To address these problems, the authors show that by jointly training a conditional and an unconditional diffusion model, we can combine the resulting conditional and unconditional error estimates, by interpolating both values, and obtain a decent trade-off between sample quality and diversity of generation. The authors also show that rather than sampling in the direction of the gradient of an image classifier, Classifier-Free Guidance mixes the score estimates of a conditional and unconditional diffusion model. We can adjust the mixing weight to attain a FID/IS trade-off, that assess the distance between the feature representations of real and generated images, and quality and diversity of the generated images, respectively. The updated algorithm for training is the following:

Algorithm 1 Joint training a diffusion model with classifier-free guidance

```
Require: $p_{\text{uncond}}$: probability of unconditional training
1: repeat
2:     $(\mathbf{x}, \mathbf{c}) \sim p(\mathbf{x}, \mathbf{c})$                ▷ Sample data with conditioning from the dataset
3:     $\mathbf{c} \leftarrow \varnothing$ with probability $p_{\text{uncond}}$  ▷ Randomly discard conditioning to train unconditionally
4:     $\lambda \sim p(\lambda)$                                ▷ Sample log SNR value
5:     $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
6:     $\mathbf{z}_\lambda = \alpha_\lambda \mathbf{x} + \sigma_\lambda \boldsymbol{\epsilon}$                   ▷ Corrupt data to the sampled log SNR value
7:     Take gradient step on $\nabla_\theta \left\| \boldsymbol{\epsilon}_\theta(\mathbf{z}_\lambda, \mathbf{c}) - \boldsymbol{\epsilon} \right\|^2$   ▷ Optimization of denoising model
8: until converged
```

Fig. 4: Training Algorithm

We can see that the algorithm is pretty similar to the previous training algorithm 2, the main difference being that we also have access to the class $\mathbf{c}$ and that with probability p_{uncond} we remove the class to train the unconditional model. For our work we set $p_{\text{uncond}} = 0.1$ so that only 10% of the times, the model trains unconditionally. Lastly, the sampling algorithm is the following:

Algorithm 2 Conditional sampling with classifier-free guidance

```
Require: $w$: guidance strength
Require: $\mathbf{c}$: conditioning information for conditional sampling
Require: $\lambda_1, \dots, \lambda_T$: increasing log SNR sequence with $\lambda_1 = \lambda_{\min}$, $\lambda_T = \lambda_{\max}$
1: $\mathbf{z}_1 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: for $t = 1, \dots, T$ do
       ▷ Form the classifier-free guided score at log SNR $\lambda_t$
3:     $\tilde{\boldsymbol{\epsilon}}_t = (1+w)\boldsymbol{\epsilon}_\theta(\mathbf{z}_t, \mathbf{c}) - w\boldsymbol{\epsilon}_\theta(\mathbf{z}_t)$
       ▷ Sampling step (could be replaced by another sampler, e.g. DDIM)
4:     $\tilde{\mathbf{x}}_t = (\mathbf{z}_t - \sigma_{\lambda_t}\tilde{\boldsymbol{\epsilon}}_t)/\alpha_{\lambda_t}$
5:     $\mathbf{z}_{t+1} \sim \mathcal{N}(\tilde{\boldsymbol{\mu}}_{\lambda_{t+1}|\lambda_t}(\mathbf{z}_t, \tilde{\mathbf{x}}_t), (\tilde{\sigma}^2_{\lambda_{t+1}|\lambda_t})^{1-v}(\sigma^2_{\lambda_t|\lambda_{t+1}})^v)$ if $t < T$ else $\mathbf{z}_{t+1} = \tilde{\mathbf{x}}_t$
6: end for
7: return $\mathbf{z}_{T+1}$
```

Fig. 5: Sampling Algorithm

The main difference between this algorithm and the previous sampling algorithm is that we give conditioning information for sampling, $\mathbf{c}$, so we can choose what class of images we want to generate, and that we interpolate between the predicted conditional noise estimates, $\epsilon_\theta(\mathbf{z}_t, \mathbf{c})$ and unconditional noise estimates $\epsilon_\theta(\mathbf{z}_t)$, by doing an interpolation with parameter w, called the guidance strength, obtaining a final predicted noise equal to:

$$\tilde{\epsilon}_t = (1+w)\epsilon_\theta(\mathbf{z}_t, \mathbf{c}) - w\epsilon_\theta(\mathbf{z}_t) \tag{10}$$

To conclude, it is important to notice that in the above algorithms, the authors add noise to the data using a log SNR (Signal-to-noise) sequence of λ_i. However, the previously mentioned linear variance schedule also presents good results and we will stay with it for our second model.

III. Datasets

In our study two datasets were used to train the DDPMs, both of these datasets being subsets of the well known CIFAR-10 dataset [7]. CIFAR-10 consists of 32×32px images categorized into 10 classes, each class containing 6000 images. The first dataset used was composed exclusively by the automobile category of the aforementioned dataset, the goal being feeding our models with only one type of image. With the second dataset we introduced the models to different categories. This dataset consisted of the following categories of CIFAR-10: airplane, automobile, cat, horse, ship. This choice was made, instead of using the full dataset, to reduce drastically the training time and complexity while still exposing the model to multiple categories of images.
This way we end up with two datasets to train and test our models with, one with just one category of images and another with five.

IV. Implementation

For both the simple model and the improved model, we chose to implement forward diffusion with a linear noise variance schedule, β_t. The variance schedule ranged linearly from $\beta_0 = 0.0001$, to $\beta_T = 0.02$, the same values used by the authors of the original DDPM paper [2]. However, while the authors chose $T = 1000$ steps, we use $T = 500$, to reduce the computational cost of the backward process, allowing us to do less iterations over the noisy images while sampling, but also maintaining it big enough so we wouldn't have a considerable reduction in sampling quality.
The backward process is what differs most between implementations. We wanted to maintain a similar structure to the original DDPM and therefore we also used a U-Net architecture [8] for our neural network. We can see in the following example an U-Net architecture:

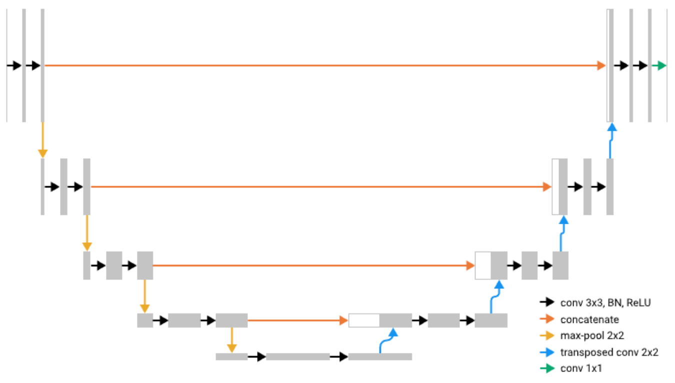

Fig. 6: Example U-Net

The U-Net has an U-like structure and consists of 3 parts, an encoder, a bottleneck, and a decoder.
Our encoder is the first part of the structure, it starts by applying a Double-Convolution block that consists of 2 sets of convolution layers, followed by group normalization layers and a GeLU activation function, only for the first convolution. To conclude the encoder, we have 3 Down-sample blocks, each of them followed by a self-attention layer, as recommended in the original DDPM paper. The Down-sample blocks start by applying Max Pooling, and then 2 Double-Convolution blocks, where the first convolution is residual. The objective of the encoder is to double the number of feature channels at each step, while reducing the spatial dimension of the features. The bottleneck simply apply 3 Double-convolution maintaining the same dimensions.
The decoder consists of 3 Up-sample blocks, each of them followed by a single self-attention layer. The Up-sample blocks do the reverse of the Down-sample ones, they apply up-sampling layers followed by 2 Double-convolutions, with the objective of doing ”deconvolution”, increasing the spatial dimensions and reducing the feature channels. It is important to notice that in the Up-sample blocks there are also skip-connections, the Up-sample blocks receive the result from the symmetric Down-sample blocks. Also, all of our down-sample and up-sample blocks receive a sinusoidal positional embedding of the diffusion timestep t adding it to their output, so the network can have a better sense of how much noise is present.
To conclude, the neural network ends in a simple convolution with a kernel-size equal to 1, to obtain a matrix whose values are the predicted noise for each of the given image pixels. The difference between the simple model and the improved model is that throughout the improved network we also do embedding with linear layers on the images classes, adding the embedding to the positional embedding of the timestep t, so that the network can know which type of image is being used for training, or evaluation. Our neural network is a simplification and is based on the one done by Outlier [9], having a total of 23 million parameters, compared to the total of 32 million of the original DDPM architecture. We chose to simplify it since our model was already very computational expensive and we did not have the computational power to train the full model within a reasonable time frame. To train each of the models we used a batch size equal to 64, a learning rate equal to 0.001, and a total of 500 epochs, using the Adam optimizer. As we have said in the theoretical background, the loss function is the mean-squared error between the predicted noise and the real noise added to the image. It is also important to notice that with a probability equal to 50% we would apply a horizontal flip on the given image, since in the original implementation [2] the authors tried both approaches, with and without flips, and found out that when applied random horizontal flips, the sampling quality improved.

V. Results and Discussion

As the central result of our work, and the output of our models, we have Figure 8, a mosaic of images generated by our improved multi-class model (model 2).

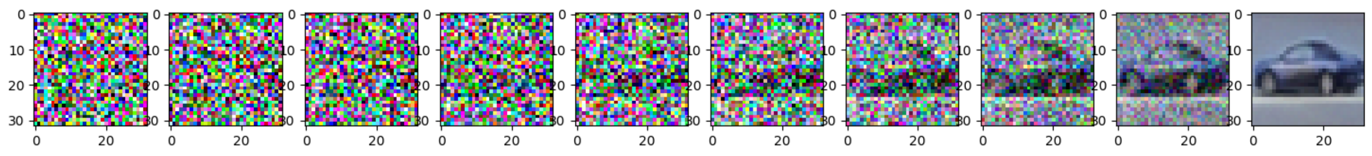
Fig. 7: Reverse diffusion process

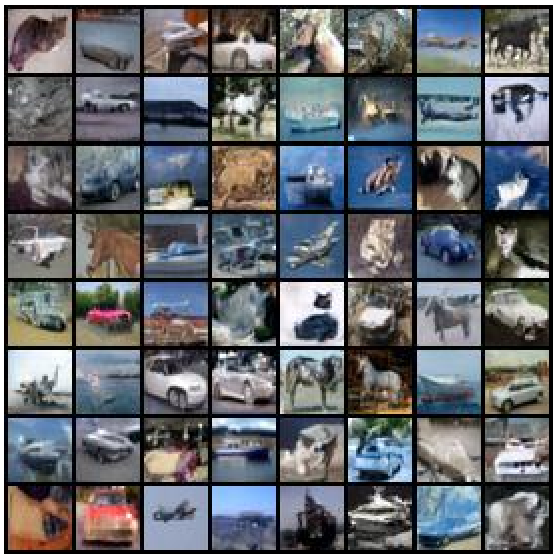
Fig. 8: Improved Model 2 - Samples

In Figure 7 we can observe the reverse diffusion process that the model applies to Gaussian noise to generate an image.
Now more acquainted with the frameworks used in our analysis, let's dive deeper into the results section of this article. First let's introduce the metrics used to measure the performance of the aforementioned models: the Fréchet Inception Distance (FID) and the Inception score (IS).
The Fréchet Inception Distance is used to assess the quality of the generated images and is computed using the following formula:

$$FID = \|\mu - \mu_w\|^2 + \operatorname{tr}\left(\Sigma + \Sigma_w - 2\left(\Sigma\Sigma_w\right)^{\frac{1}{2}}\right) \tag{11}$$

where $\mathcal{N}(\mu, \Sigma)$ is the multivariate normal distribution estimated from Inception v3 [10] features calculated on real life images and $\mathcal{N}(\mu_w, \Sigma_w)$ is the multivariate normal distribution estimated from Inception v3 features calculated on generated (fake) images. The metric was originally proposed in [10].
The Inception score measures how realistic are the images by evaluating their variety and quality. Its formula is:

$$IS = \exp\left(\mathbb{E}_x KL(p(y \mid x) \| p(y))\right) \tag{12}$$

where $KL(p(y \mid x)\|p(y))$ is the KL divergence between the conditional distribution $p(y \mid x)$ and the marginal distribution $p(y)$. Both the conditional and marginal distribution are calculated from features extracted from the images. The score is calculated on random splits of the images such that both the mean and standard deviation of the score are returned. The metric was originally proposed in inception [11].
The ideal model would minimize FID and maximize IS.
It is worth noting that the accuracy and reliability of these metrics is heavily affected by sample size, and due to computational power constraints we only generated samples of $N = 500$ images to run our tests, so results may be affected by this fact.
Now with the performance metrics established, let's explore the results in terms of performance and some of their patterns.

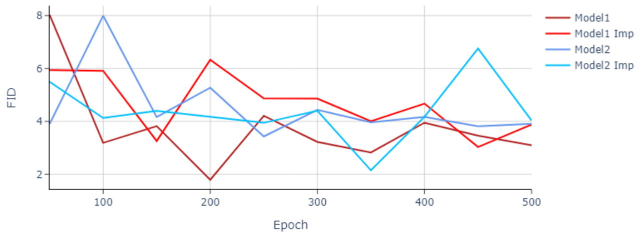

Fig. 9: FID over epochs

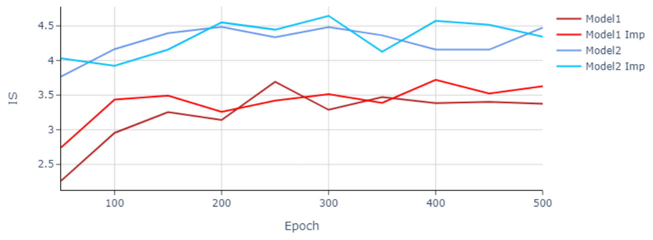

Fig. 10: IS over epochs

In Figures 9 and 10 we can observe the evolution of the metrics throughout the training epochs. We can see that both metrics trend in the desired direction (FID down and IS up) although the FID results are much noisier than the IS ones. In the IS results the expected patterns emerge beautifully, model 2 always produces better scores than model 1 (because it generates more classes of images) and the score increases as the number of epochs gets bigger. A more unfortunate result from these plots is that the improved models show no clear advantage over the non-improved ones, but as mentioned before this could stem from the small samples sizes, since

there might not be enough images to capture the difference between the models.
Our improved models incorporated class guidance so we also analyzed how the guidance level affected the performance of our models.

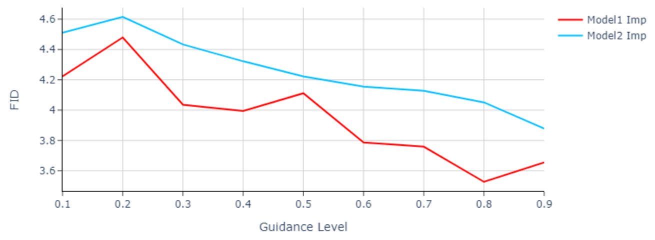

Fig. 11: FID for different guidance levels

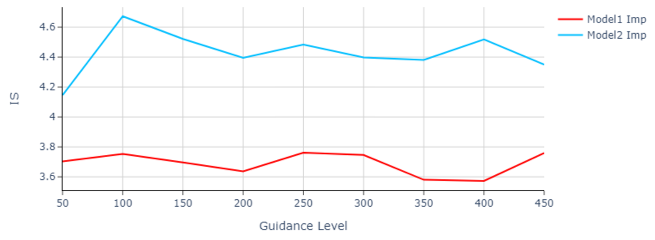

Fig. 12: IS for different guidance levels

From Figure 12 we can conclude that, in our experiments, the guidance level had little to no impact in the models' performance in terms of Inception score. But on the other hand, we are able to see, from Figure 11, that the guidance level heavily impacts FID, producing better and better results as it rises.

To finalize this results section, we will proceed to the analysis of the stability and convergence of loss during the models' training.

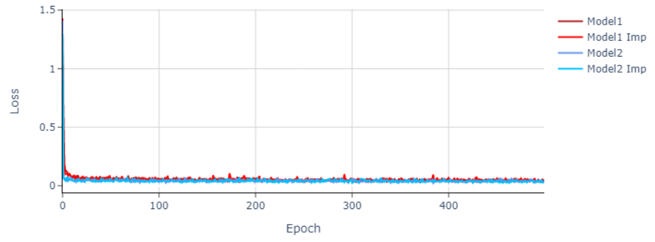

Fig. 13: Loss over training

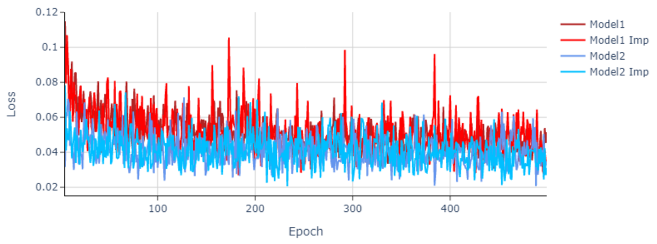

Fig. 14: Loss over training (epoch 5 onwards)

In Figure 13 we can see the evolution of loss during training, but since the reduction of loss is so drastic over the first few epochs, we are left with a large peak at the beginning followed by an apparent constant behavior. This is why we present Figure 14 that shows exactly the same as the previous but it excludes the first five epochs' values. In Figure 14 we can see patterns in the evolution of loss much more clearly. We see that for all models it slowly decreases until around epoch 100 and present a much stabler behavior afterwards. Each model and its improved version are very similar in terms of the progression of this metric and we also see that on average losses are lower for models 2 than for models 1.

VI. Conclusion and Future Work

In summary, this article explores Denoising Diffusion Probabilistic Models (DDPMs). From their theoretical background to their implementation. We also implement an improved version of the initial model, adding class guidance. From an relative perspective the results (generated images) were quite satisfactory, having in account the heavy computational power constraints faced. As mentioned in the results section, this lack of computational resources led to small generated samples that could have affected the performance metrics. Despite this, looking again at the generated images we conclude that DDPMs are quite powerful as an image generation framework and produced acceptable images, even while heavily constrained.
As for future work there are some clear improvements that can be done to help with both performance and computational/time cost. Instead of using a linear schedule to add noise to images in the forward process, a cosine schedule could be used, leading to a smoother transition from the original image to Gaussian noise and, consequently, smoother training and better results in terms of the generated images [5]. Another improvement would be a faster sampling algorithm, which would decrease the computational/time cost of sampling. Sampling was one of the biggest bottlenecks of this project in terms of results generation so fixing it could have a major impact in the efficiency of producing them. Lastly, we would like to point out that the experiment should be replicated using a larger sample, at least matching the sample sizes in [2] of $N = 50000$.

VII. References

[1] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," 2015.

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

[3] A. Jolicoeur-Martineau, K. Li, R. Piché-Taillefer, T. Kachman, and I. Mitliagkas, "Gotta go fast when generating data with score-based models," 2021.

[4] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, "Score-based generative modeling through stochastic differential equations," 2021.

[5] A. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," 2021.

[6] J. Ho and T. Salimans, "Classifier-free diffusion guidance," 2022.

[7] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)." [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

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

[9] Outlier, "Diffusion Models — PyTorch Implementation." [Online]. Available: https://www.youtube.com/watch?v=TBCRlnwJtZU

[10] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," 2015.

[11] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," 2016.

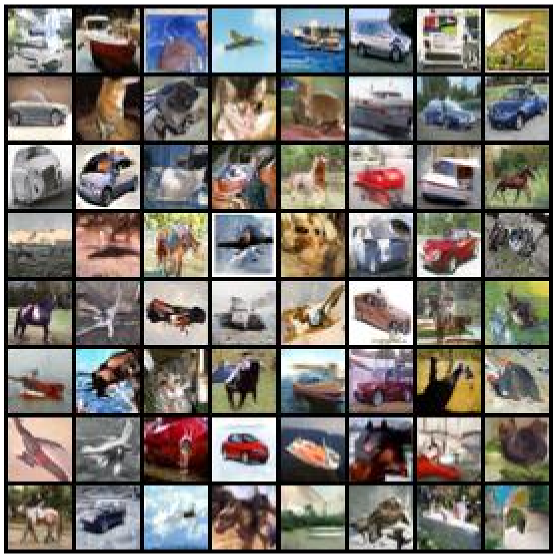
Fig. 16: Model 2 - Samples

Appendix

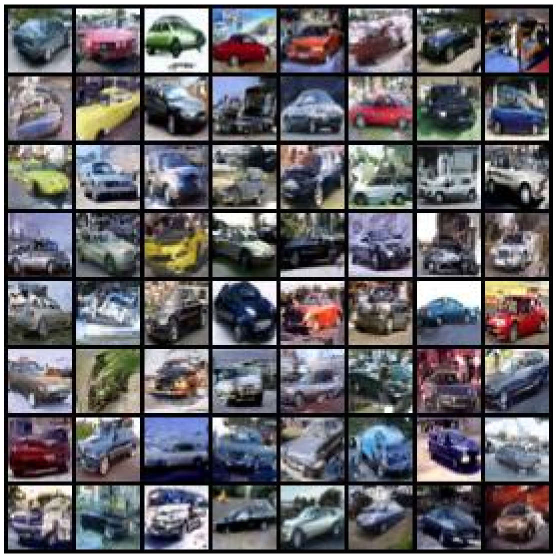
Fig. 15: Model 1 - Samples

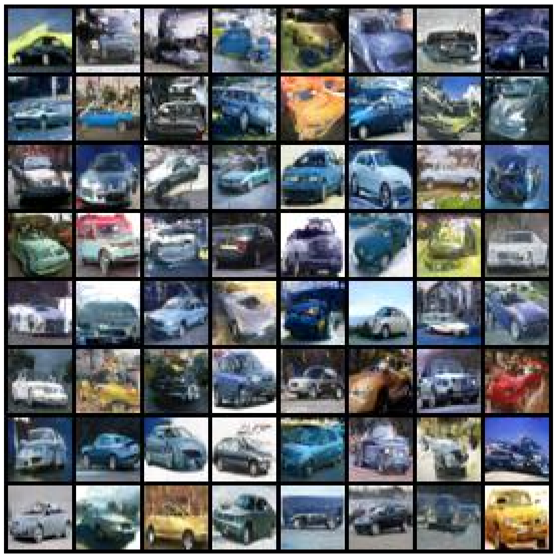
Fig. 17: Improved Model 1 - Samples