Deep Learning - Assignment 2 Report

Bart van der Woude (s3498891)
Thijs Lukkien (s3978389)
Sophie van Dijke (s3567826)

April 15, 2024

1 Introduction

In recent years, deep generative models such as Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), and diffusion models have garnered significant attention by producing increasingly high-quality results in image and audio generation [1, 2, 3]. In particular, diffusion models have innovated the field of deep generative models by leveraging non-equilibrium statistical physics to define a method of systematically destroying structure in data through a forward diffusion process and employing a reverse diffusion process to reconstruct the original data [4]. This systematic process of noising and denoising data proved to be a flexible and tractable method of modeling highly complex data, resulting in significant gains in computation time and complexity [4]. In the 2020 paper 'Denoising Diffusion Probabilistic Models' [5], Ho et. al. adapted the diffusion model process, being the first to obtain high-quality output. The final result of this paper delineated a model capable of generating images from pure noise in a series of denoising steps learned through training.

1.1 Goal of this project

Our goal was to implement the denoising diffusion probabilistic model proposed by Ho et. al [5]. on a new dataset. This involved data (pre)processing of a new dataset (Fruits-360 [6]) and designing a custom U-Net architecture to fit the task requirements. We will evaluate the performance of our implementation using the Structural Similarity Index Measurement (SSIM) [7].

2 Diffusion Models

Before expanding on the methods of our project, we will give a basic intuition on the steps of a denoising diffusion probabilistic model. The process of a diffusion model can be globally distinguished into two steps: a forward noising process and a reverse denoising process.

Forward Process. The forward process consists of a parameterized Markov chain which incrementally adds Gaussian noise to an image until it consists of pure noise, effectively destroying the image information. This process can be represented by the approximate posterior $q(x_{1:T}|x_0)$ where x_0 is the original image at time step $t = 0$, and x_T is the destroyed image at time step $t = T$. The steps of the Markov chain are parameterized by the noise scheduler β_t.

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

Reverse Process. The reverse process then learns the transitions of the Markov chain to implement an incremental denoising process, reconstructing the original image. The reverse process can be represented by the joint distribution $p_\theta(x_{0:T})$ whereby all the noised time steps from 0 to T are predicted from x_T. Figure 1 demonstrates the reverse Markovian denoising process from x_T to x_0.

$$p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right) = \mathcal{N}\left(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta\left(\mathbf{x}_t, t\right), \boldsymbol{\Sigma}_\theta\left(\mathbf{x}_t, t\right)\right) \tag{2}$$

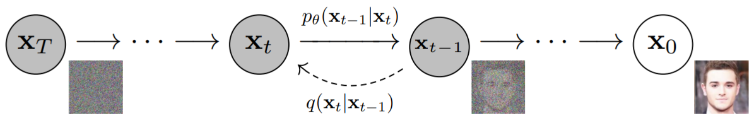

$\mathbf{x}_T$
$\mathbf{x}_t$
$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$
$\mathbf{x}_{t-1}$
$q(\mathbf{x}_t|\mathbf{x}_{t-1})$
$\mathbf{x}_0$

Figure 1: The noise-adding Markov Chain, as shown in the original paper [5]. The forward process is depicted from right to left, and the reverse process is depicted from left to right.

3 Methods

Our implementation can be found on the dedicated GitHub page.

3.1 Model architecture

The overarching structure of our model complies with the U-Net architecture proposed by Ronneberger et. al. [8], consisting of an encoder and decoder. Different compared to the original U-Net, our implementation uses a self-attention block as a bridge between the encoder and decoder. The encoder comprises two convolutional layers followed by three blocks containing downsampling and convolution operations. The decoder equally comprises three blocks containing upsampling and convolution operations. Skip connections are applied between corresponding blocks of the encoder and decoder. Finally, the last layer of the decoder is a convolutional layer that restores the original number of channel dimensions. Figure 2 demonstrates the basic architecture as described.

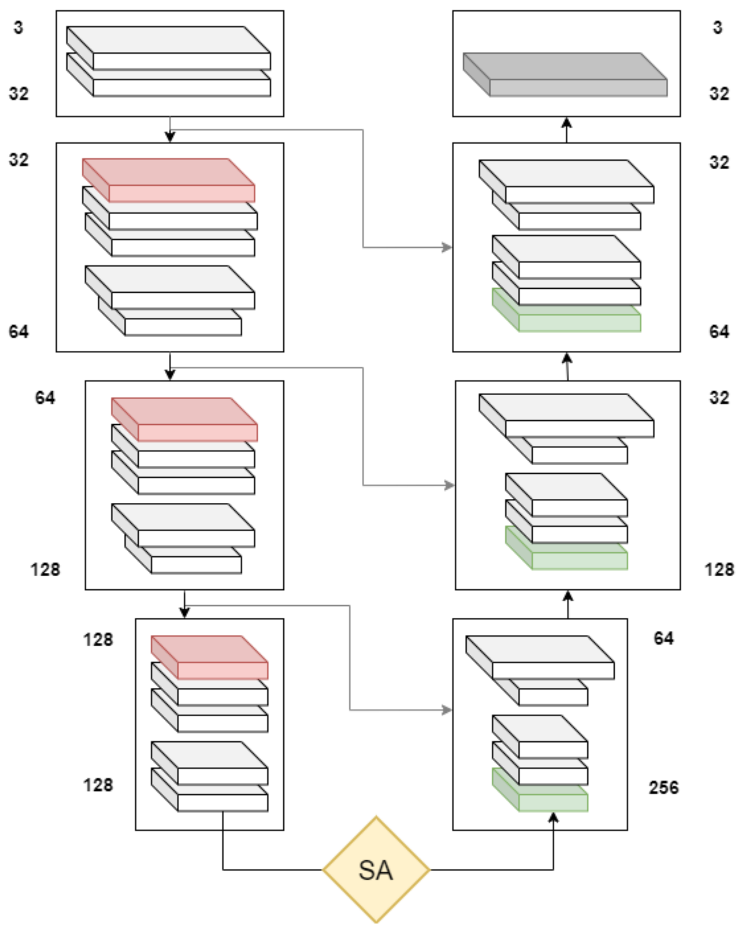


Figure 2: The architecture of the U-Net for the reverse diffusion process. Red squares indicate 2x2 maxpooling. White squares indicate a convolution operation with a filter size of 3. Green squares indicate bilinear interpolation by a factor of 2. The number of input and output channels is specified next to the blocks. Grey arrows in the middle indicate residual connections where concatenation occurs.

For further detail on this model architecture; the images are first fed through the forward noising process as defined in Section 2. The noised images comprise the input to the U-Net that we adapted from Awjuliani on Github [9]. In the encoder, or 'downward' pass, of the U-Net, the input image first undergoes a double convolution with group normalization and GELU activation, increasing the number of channels to 32. Then, the output is passed to three consecutive downsampling convolution blocks. Each block first applies a 2x2 maxpooling kernel, followed by two consecutive double convolutions. Furthermore, at each block, a positional encoding is generated and a residual connection is implemented. Thus, at the end of the encoder, the input image has undergone 3 2x2 maxpooling kernels reducing the spatial dimensions and 7 double convolutions, increasing the number of feature map channels to 128.

The output of the decoder is then passed to the self-attention mechanism adapted from the implementation by Awjuliani on Github [9], serving as the bottleneck for this U-Net. Next, the decoder, or the 'upward' pass, of the U-Net comprises 3 upsampling convolution blocks. Each block upsamples the input using bilinear interpolation with a scale factor of 2. Following the bilinear interpolation, the spatial dimensions of the output match the spatial dimensions of the input to the corresponding block in the decoder, and thus the two are concatenated. The concatenated tensor is then passed through two double convolutions decreasing the number of feature map channels. Additionally, a positional encoding is generated at each block and a group normalization is performed at each double convolution. The final step in the decoder is a 1x1 convolution, where the output has the same number of channels as the original input (3 in this case).

3.2 Model Implementation

This model was created using the PyTorch library. The self-attention mechanism implemented in this project was created by Awjuliani on Github [9], and the modules utilized in the U-Net steps were created by Milesial on Github [10]. The loss function implemented for the training of this model was a mean-squared error between the predicted noise and actual noise at time step t. The optimizer used was Adam [11] with a learning rate of 0.0002 and a weight decay of $1e^{-6}$. Appendix A contains our implementation for the forward noising process, derived from the equations derived by Ho et. al. [5]. In [5], the forward process was calculated in one step by taking the cumulative product of α_t where $\alpha_t = 1 - \beta_t$ for all time steps of β_t and applying the re-parameterization trick to formulate the equation:

$$q(x_t|x_0) = N(x_t; \sqrt{\bar{\alpha_t}}x_0, (1 - \bar{\alpha_t})I) \tag{3}$$

Early stopping was implemented, where training was stopped when validation loss did not increase over 5 consecutive epochs.

3.3 Data Preprocessing

The Fruits-360 dataset [6] was downloaded from Kaggle and preprocessed according to Appendix B. The images were first converted to type float32 and scaled to range [0 1]. Afterward, pixel values were normalized using a mean and standard deviation of 0.5 for all three channels to ensure values were in the range [-1 1]. The Fruits-360 dataset came with a given train and test split. Training data was further divided into 5 folds, with one fold designated as the validation set, during training.

3.4 Evaluation metric

The ability of a trained model to reconstruct noised images back to the original ground truth image will be evaluated using the Structural Similarity Index Measurement (SSIM) [7].

SSIM outputs a value in the range [-1, 1], where 1 represents perfect similarity between two input images. SSIM compares patches of the two input images on luminance, contrast and structure. The final output is the average of all compared image patches. SSIM was initially introduced to provide a more humanly intuitive measurement for image similarity, rather than existing pixel-based evaluation methods. [7]

SSIM can be calculated using Equation 4. x and y represent image patches to be compared, μ_x and μ_y represent average values of the image patches, σ_x^2 and σ_y^2 are the variances of the image patches, C_1 and C_2 are constants to avoid weak denominators.

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{4}$$

4 Results

The model was trained using the Habrók High Performance Computer cluster on the Fruits-360 data for 40 epochs and 5 folds of cross-validation, taking approximately 7.5 hours to complete. Figure 3 contains the plotted training and validation losses by epoch across the 5 folds. The second fold terminated at 21 epochs and the third fold terminated at 33 epochs due to early stopping; early stopping was induced when the validation loss did not decrease for five consecutive epochs.

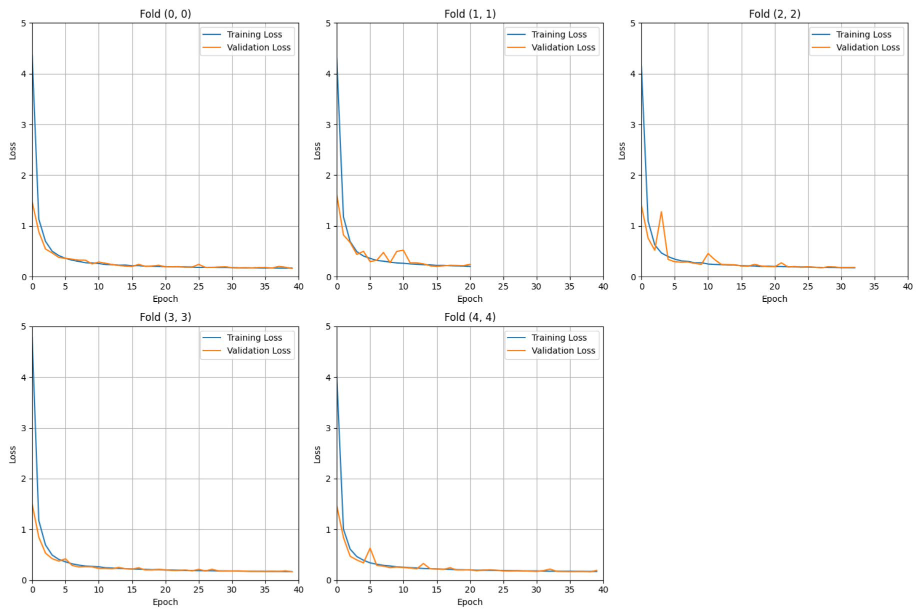


Figure 3: The training and validation loss over 40 epochs for each of the 5 folds in the cross-validation process.

Figure 4 visualizes model performance by demonstrating the ground truth of a sample image (coconut) in the top row compared to the forward and reverse processes in the middle and bottom rows, respectively. Note that these reconstructions were single-shot noise predictions. Initially, removing the predicted noise from noised inputs at high time-steps, i.e. large amounts of added noise, resulted in very dark predicted images. This was corrected by incorporating a time-step-dependent scaler into the output. This scaler was applied in the calculation: $output = scaler * (noised\_image - predicted\_noise)$, where the computation of the scaler is shown in Equation 5. Note that this scaler was found through trial and error. The original literature used $\frac{1}{\sqrt{\alpha}}$ as a scaler, however, that did not prove sufficient for our implementation as images stayed too dark. Thus, we scale the scaler further by a time-step exponential term.

$$scaler = ((\frac{1}{\sqrt{\alpha}} - 1) * 8(\frac{t}{50})^2) + 1 \tag{5}$$

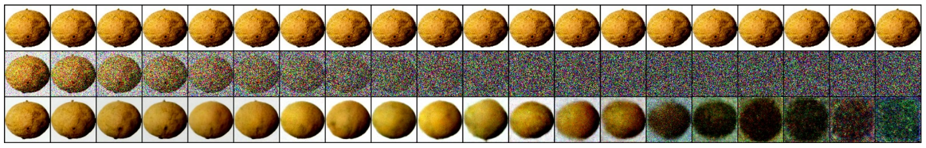


Figure 4: Noising and denoising steps of a sample image of a coconut. The top row indicates the ground truth image, the middle row exhibits the forward process, and the bottom row shows the images that remain after subtracting the predicted noise from the reverse process.

All models were evaluated on the test set using the Structural Similarity Index Measurement (SSIM). Output, as seen in the bottom row of Figure 4 was calculated by subtracting the predicted noise from the noised input image. Again, the output was scaled by the scaler mentioned in Equation 5. Table 1 shows the SSIM scores for different time-steps and models.

t	Fold 0	Fold 1	Fold 2	Fold 3	Fold 4
50	0.697 ± 0.013	0.694 ± 0.011	0.701 ± 0.013	0.703 ± 0.016	$\mathbf{0.704 \pm 0.012}$
100	0.688 ± 0.011	0.678 ± 0.010	$\mathbf{0.692 \pm 0.009}$	0.685 ± 0.011	0.682 ± 0.012
150	0.672 ± 0.012	0.654 ± 0.018	0.671 ± 0.011	$\mathbf{0.681 \pm 0.014}$	0.671 ± 0.014
200	$\mathbf{0.670 \pm 0.017}$	0.646 ± 0.014	0.664 ± 0.012	0.666 ± 0.016	0.664 ± 0.013
250	0.652 ± 0.011	0.629 ± 0.014	0.653 ± 0.011	$\mathbf{0.657 \pm 0.014}$	0.653 ± 0.008
300	$\mathbf{0.640 \pm 0.012}$	0.604 ± 0.011	0.630 ± 0.015	0.630 ± 0.009	0.638 ± 0.011
350	$\mathbf{0.626 \pm 0.011}$	0.556 ± 0.011	0.608 ± 0.012	0.619 ± 0.015	0.624 ± 0.011
400	$\mathbf{0.606 \pm 0.011}$	0.553 ± 0.011	0.580 ± 0.011	0.591 ± 0.015	0.596 ± 0.012
450	0.568 ± 0.015	0.496 ± 0.012	0.537 ± 0.018	$\mathbf{0.572 \pm 0.014}$	0.557 ± 0.011
500	$\mathbf{0.531 \pm 0.009}$	0.426 ± 0.013	0.497 ± 0.013	0.500 ± 0.014	0.519 ± 0.012
550	$\mathbf{0.481 \pm 0.015}$	0.359 ± 0.012	0.414 ± 0.011	0.473 ± 0.012	0.440 ± 0.007
600	$\mathbf{0.397 \pm 0.011}$	0.274 ± 0.005	0.328 ± 0.009	0.384 ± 0.010	0.373 ± 0.010
650	0.307 ± 0.007	0.181 ± 0.005	0.257 ± 0.009	$\mathbf{0.316 \pm 0.009}$	0.286 ± 0.009
700	$\mathbf{0.224 \pm 0.005}$	0.120 ± 0.003	0.167 ± 0.006	0.223 ± 0.007	0.203 ± 0.007
750	0.127 ± 0.004	0.070 ± 0.002	0.103 ± 0.004	$\mathbf{0.139 \pm 0.004}$	0.114 ± 0.003
800	0.078 ± 0.002	0.040 ± 0.001	0.053 ± 0.002	$\mathbf{0.080 \pm 0.003}$	0.061 ± 0.002
850	$\mathbf{0.048 \pm 0.002}$	0.025 ± 0.001	0.034 ± 0.001	0.041 ± 0.002	0.040 ± 0.002
900	$\mathbf{0.028 \pm 0.001}$	0.017 ± 0.000	0.024 ± 0.001	0.027 ± 0.001	0.027 ± 0.001
950	0.020 ± 0.001	0.017 ± 0.000	0.017 ± 0.001	$\mathbf{0.020 \pm 0.000}$	0.018 ± 0.000
1000	0.015 ± 0.000	$\mathbf{0.016 \pm 0.000}$	$\mathbf{0.016 \pm 0.000}$	$\mathbf{0.016 \pm 0.000}$	$\mathbf{0.016 \pm 0.000}$

Table 1: Average SSIM scores for all models, i.e. folds, evaluated on the test set. Values in bold indicate the best score over all models for each time-step t.

Aside from reconstructing images from the test set, several attempts were made to generate images. Figure 5 shows our best attempt at image generation. In this form of image generation, a zero-initialized image had Gaussian noise added according to the forward process. The model then predicts the noise in the image. This prediction is removed from the noised image. The next

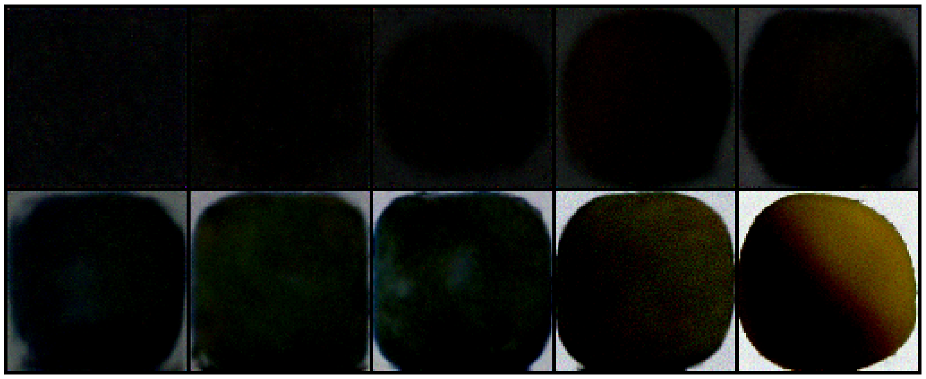


Figure 5: Results of image generation; removing a portion of noise iteratively from t=1000 to t=0. Rows show progression through timesteps starting at t=900 to t=0 with increments of 100 time-steps. Note that the actual generation used increments of 1.

iteration again adds Gaussian noise, but at a lower time-step, which the model aims to remove. The results can be seen in the aforementioned figure. Note that this is not the conventional way of generation. Conventional methods were fruitless and this was the best result acquired. In the analysis, we discuss the other generative methods we tried.

5 Analysis

The first point of analysis concerns the training and validation losses; Figure 3 shows the training and validation losses by epoch for each of the 5 folds in cross-validation. At first glance, overfitting does not appear to be present as both the training and validation losses exhibit similar, almost identical, convergence. Similarly, underfitting does not appear to be present because both losses converge close to zero. However, these plots revealed an error made in the model training and cross-validation procedure as the two losses appear to be nearly identical in each fold. We identified this problem as one of data leakage; in the Fruits-360 dataset, the training data is created by photographing the same fruit from different angles (e.g., a photo of the same coconut taken from different angles). The splits for the cross-validation were executed without regard for this augmentation, leading to leakage of the training data into the validation set and, therefore, low validation losses. The early stopping present in folds 2 and 3 are thus erroneous since although optimal validation loss was achieved before the allotted amount of epochs, it did not represent an accurate indication of generalization ability. Therefore, it is not possible to discern if we indeed elucidated the optimal model.

Next, Figure 4 contains the forward and reverse processes compared to a sample ground truth image. The second row demonstrates the progression of the forward process on a sample image, indicating the successful linear addition of random Gaussian noise. However, based on the image progression, the linear schedule of adding Gaussian noise appears to destroy the signal quality very early on. This leads to a redundancy in the latter half where the input is destroyed and possibly a reduction in learning capacity because the signal is destroyed too soon. The bottom row shows the reverse process results with increasingly noisier inputs until the last image where the input is pure noise. The image reconstruction exhibits quite high fidelity in the lower time steps, indicating that the model is fairly robust to noise up to a certain extent. This is corroborated in Table 1 demonstrating SSIM scores of each model over incremented time steps. The models produce the most similar outputs at the lower time steps with the best performance being 0.704 ± 0.012 at time step $t = 50$ by the Fold 4 model. Across all models, the SSIM scores decrease with higher time steps, indicating that the model struggled to perform accurate denoising at higher noise levels. At the last time step $t = 1000$, the Fold 0 model outputs an SSIM score of 0.015 ± 0.000, and Fold 1-4 models output an SSIM score of 0.016 ± 0.000. Thus, it appears that the model performs reconstruction from lower noise quite well, achieving an average SSIM score of approximately 0.7 overall trained models, however, the model struggles to reconstruct images from larger amounts of noise. This may contribute to the problems we encountered in generating images from noise.

The result of our attempt at image generation was not satisfactory. Initially, our implementation was closely linked to the method from the original paper. In this method, Gaussian noise is added, afterward a small ratio (*noise_scaler*) of the predicted noise is removed from the noised image and that output is slightly scaled (*scaler*). As a last step, a small amount of noise is added back. *noise_scaler* is a value close to 0 and *scaler* is a value slightly above 1. However, using this method resulted in very quick convergence to black output. Different variations of the *noise_scaler* and *scaler* were attempted but with little success. Using the *scaler* from Equation 5 slightly improved the generative process, however, not to a satisfactory degree. Only after using our model to remove all the noise it predicted and then adding new noise back for the next time-step in the iteration, was the result seen in Figure 5 generated. It is unclear why the generative process is such a hurdle. There is a good chance the faulty generative process is linked to why the custom *scaler* from Equation 5 was necessary. Perhaps a portion of the forward process was wrong, where it created Gaussian noise that was shifted positively to a degree where it enforces the model to predict more positive noise. Resulting in darker images when this positive noise is removed.

References

[1] R. Gozalo-Brizuela and E. C. Garrido-Merchan, "Chatgpt is not all you need. a state of the art review of large generative ai models," 2023.

[2] S. Bond-Taylor, A. Leach, Y. Long, and C. G. Willcocks, "Deep generative modelling: A comparative review of vaes, gans, normalizing flows, energy-based and autoregressive models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 11, pp. 7327–7347, 2022.

[3] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2022.

[4] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," *CoRR*, vol. abs/1503.03585, 2015. [Online]. Available: http://arxiv.org/abs/1503.03585

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

[6] H. Mureșan and M. Oltean, "Fruit recognition from images using deep learning," *Acta Universitatis Sapientiae, Informatica*, vol. 10, pp. 26–42, 06 2018.

[7] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

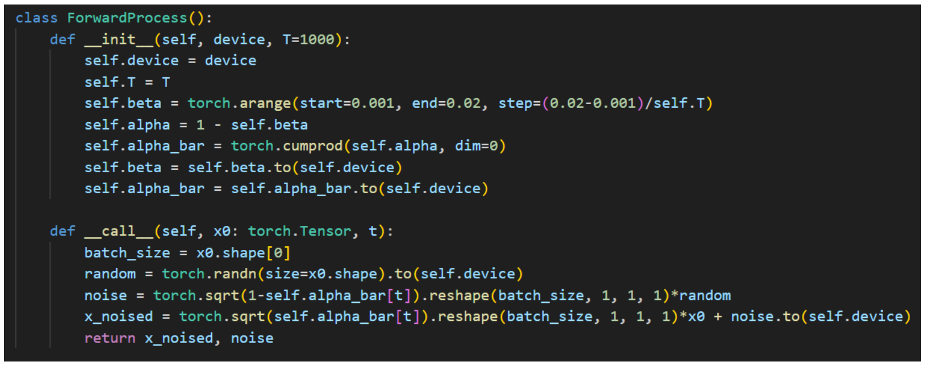
[8] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015.

[9] A. Juliani, "pytorch-diffusion," https://github.com/awjuliani/pytorch-diffusion, accessed: April, 2023.

[10] milesial, "Pytorch-unet," https://github.com/milesial/Pytorch-UNet, accessed: April, 2023.

[11] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

A Forward Process

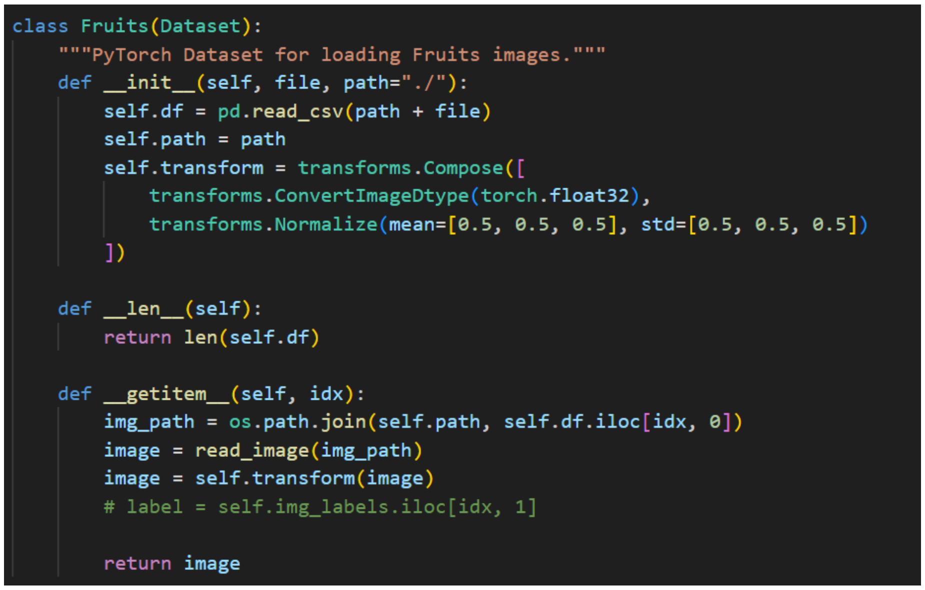


```python
class ForwardProcess():
    def __init__(self, device, T=1000):
        self.device = device
        self.T = T
        self.beta = torch.arange(start=0.001, end=0.02, step=(0.02-0.001)/self.T)
        self.alpha = 1 - self.beta
        self.alpha_bar = torch.cumprod(self.alpha, dim=0)
        self.beta = self.beta.to(self.device)
        self.alpha_bar = self.alpha_bar.to(self.device)

    def __call__(self, x0: torch.Tensor, t):
        batch_size = x0.shape[0]
        random = torch.randn(size=x0.shape).to(self.device)
        noise = torch.sqrt(1-self.alpha_bar[t]).reshape(batch_size, 1, 1, 1)*random
        x_noised = torch.sqrt(self.alpha_bar[t]).reshape(batch_size, 1, 1, 1)*x0 + noise.to(self.device)
        return x_noised, noise
```

Figure 6: The forward noising process as calculated by Equation 3.

B Data Preprocessing



```python
class Fruits(Dataset):
    """PyTorch Dataset for loading Fruits images."""
    def __init__(self, file, path="./"):
        self.df = pd.read_csv(path + file)
        self.path = path
        self.transform = transforms.Compose([
            transforms.ConvertImageDtype(torch.float32),
            transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
        ])

    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
        img_path = os.path.join(self.path, self.df.iloc[idx, 0])
        image = read_image(img_path)
        image = self.transform(image)
        # label = self.img_labels.iloc[idx, 1]

        return image
```

Figure 7: Pytorch Dataset for loading and preprocessing Fruits-360 data.