Major Module M2: Application of Machine Learning
Coursework Assignment
Word count: 2897 words

C. Grivot

(Dated: 28 March 2024)

I. TRAINING A DIFFUSION MODEL

A. Part 1(a)

The provided notebook "coursework starter.ipynb" is a simplified comprehensive implementation of a diffusion model, specifically the Denoising Diffusion Probabilistic Models (DDPM) (Ho, Jain, and Abbeel (2020)), applied to generate realistic digit images from the MNIST dataset using PyTorch.

1. Model Overview

The diffusion model is a type of generative model that learns to generate data by gradually denoising a random noise distribution. The process is divided into two main phases: the forward process, which adds noise to the data over a series of steps until it becomes pure noise, and the reverse process or sampling, which learns to denoise back to the original data (Weng (2021)).

The diffusion process is modelled mathematically as a Markov chain, where each step adds a small, controlled amount of Gaussian noise to the data, following a predefined schedule determined by beta parameters. The forward diffusion process can be described by a sequence of conditional probabilities, each representing the distribution of the data at time $t+1$ given its state at time t. The noise schedule is for controlling the diffusion process. It defines how noise is added at each timestep of the forward process. The code from the starter notebook computes a linear noise schedule using parameters β_1 and β_2 to generate β_t and α_t, representing the variance of the noise added at each step and the cumulative product of $1-\beta_t$, respectively. The forward process is based on the algorithm 18.1, and the sampling is based on the algorithm 18.2 of the Prince (2023) textbook:

Algorithm1 Diffusion model training

Require: Training data x
Ensure: Model parameters ϕ_t
  repeat
    for $i \in \mathcal{B}$ **do**
      $t \sim \text{Uniform}[1, \ldots, T]$
      $e \sim \mathcal{N}(0, \mathbf{I})$
      $\mathcal{L}_i = \left\| g_t\left(\sqrt{\alpha_t}x_i + \sqrt{1-\alpha_t}e, \phi_t\right) - e \right\|^2$
    end for
    Accumulate losses for batch and take gradient step
  until converged

A simple Convolutional Neural Network (CNN) is used to estimate the reverse diffusion process. It does not directly generate the data at each step but instead predicts the noise that was added at a particular step of the forward process. By accurately estimating this noise, the model can effectively "denoise" the data, step by step, in the reverse diffusion process. The CNN consists of a series of 'CNNBlock' modules, each comprising a convolutional layer followed by Layer Normalisation and a GELU activation function. The model

Algorithm2 Sampling

Require: Model, g_t^θ, ϕ_t
Ensure: Sample, x
  $z_T \sim \mathcal{N}(0, \mathbf{I})$
  for $t = T, \ldots, 2$ **do**
    $\tilde{z}_{t-1} = \frac{1}{\sqrt{1-\beta_t}} \left(z_t - \frac{\beta_t}{\sqrt{1-\alpha_t^t-\beta_t}} g_t^\theta \left(z_t, \phi_t \right) \right)$
    $e \sim \mathcal{N}(0, \mathbf{I})$
    $z_{t-1} = \tilde{z}_{t-1} + \sigma_t e$
  end for
  $x = \frac{1}{\sqrt{1-\beta_1}} \left(z_1 - \frac{\beta_1}{\sqrt{1-\alpha_1}} g_1^\theta \left(z_1, \phi_1 \right) \right)$

also includes a mechanism to incorporate time information into the network via sinusoidal time embedding, which is important for the model to learn the denoising process at different noise levels.

2. Training Algorithm

At each training step, the model takes a noised image, tries to denoise it, and then the loss is calculated as the MSE between the model's output and the actual noise, using the model's prediction of the added noise.

Fist the algorithm loads and preprocess the MNIST dataset, initialise the CNN model and DDPM module, and set up the optimiser, in this case we are using the ADAM optimiser. For each epoch, the training algorithm iterates over the dataset in batches: - For each batch, compute the model's loss by comparing the predicted noise to the actual noise added to the original data. - Perform backpropagation and update the model's weights. By minimising this loss across all steps and all images in the training dataset, the model learns to effectively reverse the diffusion process. Eventually, it becomes capable of generating clear images starting from pure noise.

After each epoch, it generates samples by reversing the diffusion process starting from pure noise. This involves iteratively applying the model to denoise the data over the reverse sequence of noise levels. The generated digit images and the model's state are saved periodically, 16 images per epoch, for visualisation.

B. Part 1(b)

1. Loss Curve Analysis

In the loss curve against epoch count (Figure **??**), it is shown a steep decline at the beginning, this a could mean that the model quickly learns to estimate the noise added to the data during the early stages of training. The curve flattens as training progresses, the model begins to converge and makes smaller incremental improvements.

2. FID Score Analysis

The Frechet Inception Distance (FID) scores are a measure of the quality of the samples generated by the model, comparing the distribution of generated images to the distribution of real images. Lower FID scores indicate better quality and similarity to the target distribution. We have FID scores computed by different functions, including Torchmetrics and two other functions implemented which are "My function" and " Bansal *et al.* (2022) functions".

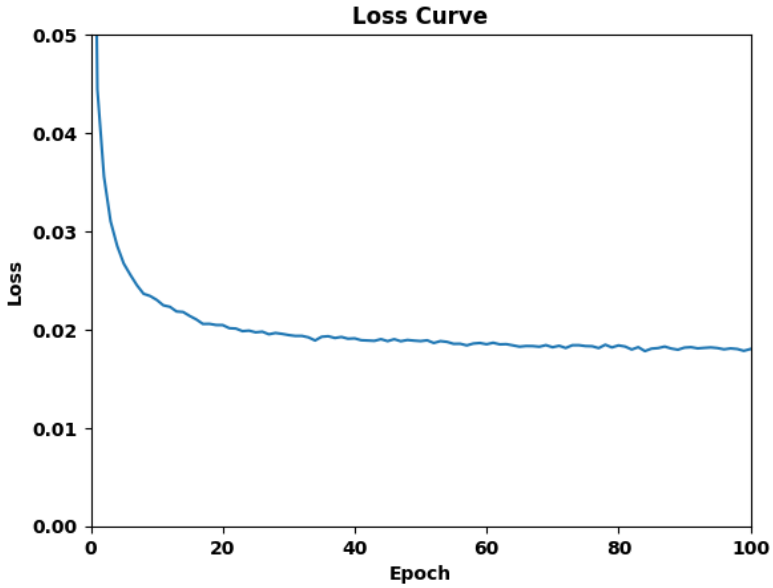

FIG. 1. Loss curve against epoch count of starter model

When calculating the score with Torchmetrics' function, we get a negative FID score. This value obtained seems erroneous since FID scores should not be negative. This might indicate a possible implementation error or computational anomaly. This could be due to the high number of features set in the parameters, here we have chosen 2048, which indicates the inceptionv3 feature layer to choose. Hence, recalculations are needed with a lower value among: 64, 192 and 768. The scores from the other two functions are positive but differ substantially, suggesting that the implementation details of the FID calculation might vary, leading to different assessments of the sample quality. Bansal *et al.* (2022) functions functions were used so that the different models could be compared to those in the paper.

Figure 2 shows the FID score calculated using Bansal *et al.* (2022) functions as it changes over each epoch during the training of the diffusion model on the MNIST dataset. We can infer that the FID score fluctuates considerably across epochs. This might be due to the stochastic nature of the training process, changes in the batch data across epochs, or due to the model sometimes generating better or worse samples at different training stages. Despite the fluctuations, the overall trend shows a slight direction towards improvement as the epochs increase. We see a downward trend as the model improves in generating samples closer to the real data distribution. However, the FID scores vary between approximately 62.5 and 80. This range is quite broad, indicating a significant variance in the quality of generated images across epochs. Ideally, the range of variation would decrease as the model becomes more consistent in generating high-quality samples, which is not the case. In some cases, an initial rapid improvement followed by high variability in FID scores can be a sign of overfitting. There is no clear point at which the FID scores converge to a stable value, additional training, early stopping around 70, or further hyperparameter tuning could potentially lead to better and more stable performance.

C. Part 1(c): Model Training and Analysis

For the hyperparameters Set 2, the plan was to increase the depth and width of the CNN (e.g., more layers and channels), this way the model has a larger capacity for learning complex patterns. However, it might also be prone to overfitting or require more data and

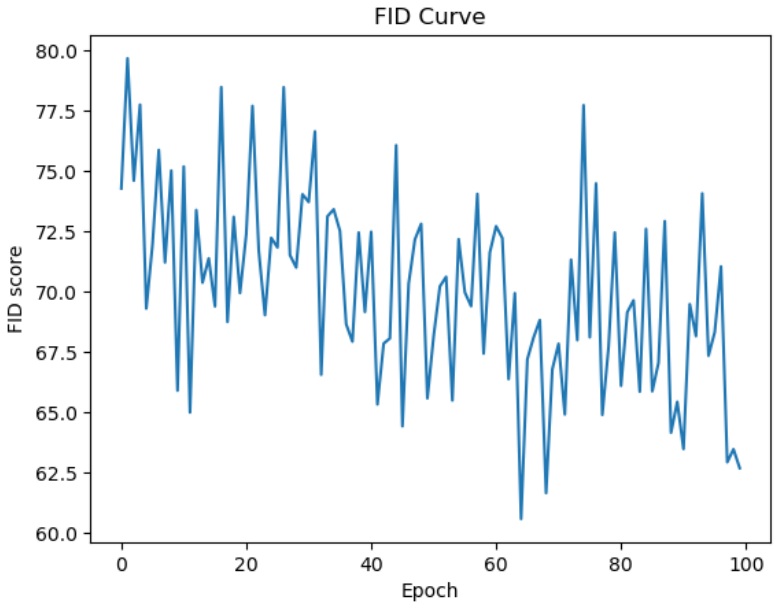

FIG. 2. FID Curve representing quality of samples across training epochs of the starter model

training time to converge. The change from GELU to LeakyReLU was motivated by the fact that the LeakyReLU allows a small gradient when the unit is not active, potentially mitigating the dying ReLU problem, but it also behaves differently in terms of data flow within the network. A lower learning rate was chosen, which might slow down training but can lead to more stable convergence over time.

For the hyperparameters set 3, the network structure aims for a balance between the starter and hyperparameters Set 2, which might offer good performance without the cost of significant computation. ELU can help mitigate the vanishing gradient problem and generally offers better performance for deeper networks. However, it is computationally more intensive than ReLU and its variants. The choice of a more gradual noise schedule and a moderate learning rate is intended to provide a stable training dynamic, avoiding both the potential instability of an aggressive noise schedule and the slow convergence of a very low learning rate.

Standard Model with Hyperparameters Set 3:

- Using Your Own Function: FID Score: 69.34
- Using Bansal et al. (2022) Functions: FID Score: 79.10

1. Qualitative Sample Analysis

To evaluate the samples, we must look at digit recognition, MNIST dataset characteristics, variability and background noise.

High-quality generated images should be easily recognisable as one of the ten digits (0-9), the generated images of set 3 present this quality, where we can easily recognise the digits that are not shaded or disappearing into the background (Figure 4). As for set 2 we can't tell apart digits 0 from 6, we can also observe some anomalies where we can't recognise if the digits are 1 or 7 giving us images that are ambiguous or hard to classify (3).

Both samples are consistent with the MNIST dataset characteristics, such as the typical thickness of the strokes which is present, the range of digit sizes. However, the background-foreground contrast is better seen in the samples of set 2.

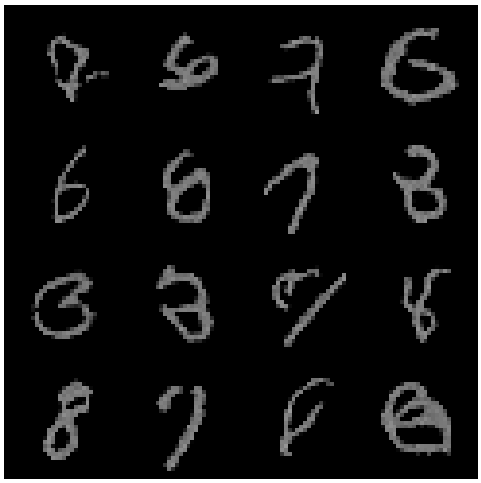

FIG. 3. Samples from the trained model with the second set of hyper-parameters

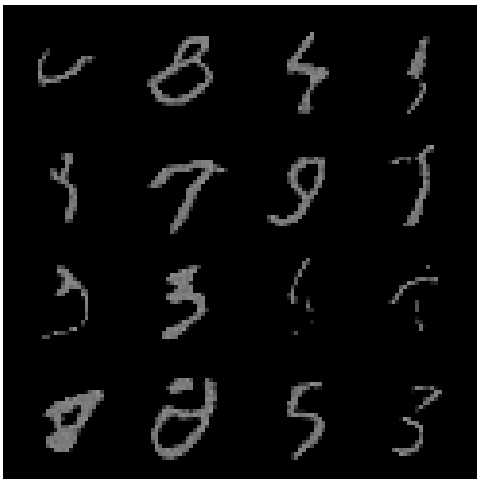

FIG. 4. Samples from the trained model with the third set of hyper-parameters

Since the MNIST dataset includes a wide range of handwriting styles. High-quality generated images should reflect this variability, capturing different styles and variations in digit formation, which we can see from the different types of "7" in set 2 and digit "4" in set 3.

The background of MNIST images is typically clean and uniform, set 2 clearly has a similar clean background. However, in set 3, the digits seem to blend into the background.

II. CUSTOM DEGRADATION

A. Part 2(a)

The initial approach involved incorporating structural disruptions, such as cutout masks and elastic transformations, along with contrast alterations, into the diffusion process to study their effects. However, introducing elastic transformations during the forward process hindered the model's ability to generate recognisable digits after training for 100 epochs. Implementing local contrast variations also presented coding challenges. It was decided to first proceeded with a diffusion model combining custom degradation and Gaussian noise.

The cutout mask algorithm was applied to each image, with the number of masks based on the diffusion step's intensity. The location of each mask was random, and its size varied with intensity. Masks were applied by zeroing out pixel values within their region, introducing occlusions to challenge the model's training and potentially enhance its generalisation capabilities.

Contrast adjustment was tied to the diffusion step's intensity, setting the range of contrast change with a scaling factor, "alpha t". For every image, we adjusted pixel intensities around the mean value to simulate both local and global contrast variations. This was particularly relevant for the MNIST dataset's black-and-white digit images, testing the model's adaptability to different contrast levels.

1. Custom degradation without Gaussian noise added

For creativity, in this implementation, the Gaussian noise addition is replaced with solely custom degradation. Which, at first, was a degradation strategy based on transforming

Algorithm3 Apply Cutout Masks

Require: Image tensor x with shape (B, C, H, W), Diffusion step intensity *intensity*
Ensure: Image tensor x with cutout masks applied
  $num\_masks \leftarrow \text{int}(1 + intensity \times 5)$ {The number of masks increases with intensity}
  $mask\_size \leftarrow \text{int}(3 + intensity \times H/4)$ {Size of mask increases with intensity}
  for each image in batch x **do**
    for $i \leftarrow 1$ to num_masks **do**
      $mask\_center_x \sim \text{Uniform}[0, W)$
      $mask\_center_y \sim \text{Uniform}[0, H)$
      $left \leftarrow \max(mask\_center_x - mask\_size//2, 0)$
      $right \leftarrow \min(mask\_center_x + mask\_size//2 + 1, W)$
      $top \leftarrow \max(mask\_center_y - mask\_size//2, 0)$
      $bottom \leftarrow \min(mask\_center_y + mask\_size//2 + 1, H)$
      for all channels c in C **do**
        $x[:, c, top : bottom, left : right] \leftarrow 0$ {Apply mask to region}
      end for
    end for
  end for
  return x

Algorithm4 Adjust Image Contrast

Require: Image tensor x with shape (B, C, H, W), Diffusion step intensity *intensity*
Ensure: Image tensor x with adjusted contrast
  $min\_contrast \leftarrow 0.5 - intensity \times 0.5$ {Minimum contrast decreases with intensity}
  $max\_contrast \leftarrow 1.5 + intensity \times 0.5$ {Maximum contrast increases with intensity}
  $contrast\_levels \sim \text{Uniform}[min\_contrast, max\_contrast]$ {Random contrast level for each image}
  for each image in batch x **do**
    $means \leftarrow \text{mean}(x, \text{across spatial dimensions})$ {Compute mean intensity per image}
    $contrast\_level \leftarrow$ sample from $contrast\_levels$ {Sample random contrast level}
    $x \leftarrow (x - means) \times contrast\_level + means$ {Apply contrast adjustment}
  end for
  return x

MNIST digits into QR codes. However, during the training process, it encountered significant computational challenges, primarily due to the time-intensive nature of generating and applying QR codes to each image in the batch. This approach proved to be impractical for large-scale training, leading to reconsider the strategy.

Shifting to a more computationally efficient approach, it was decided to use pseudo-QR codes as the basis of the degradation strategy. Pseudo-QR codes are binary patterns that mimic the appearance of QR codes without encoding any specific data (Figure5). By randomly generating these patterns, we were able to create visually similar degradation effects while significantly reducing the computational overhead associated with QR code generation and application.

The process involves overlaying these pseudo-QR patterns onto the original MNIST digits with varying intensity levels. The intensity of the pattern is controlled by a predefined schedule, which determines the extent of the degradation at each diffusion step. Initially, the intensity is low, resulting in minimal degradation and preserving most of the original digit's features. As the diffusion process progresses, the intensity increases, leading to a gradual transformation of the digit into a pseudo-QR pattern. This was done by setting sigma schedule with a non-linear one, adopting a sigmoidal shape that introduces a more gradual transition of intensities.

The forward method calculates the loss between the original images and the predictions made by the CNN, instead of the noise as in the original method. This loss is used to train the model, guiding the optimisation of both the CNN and the parameters governing the degradation process.

Two key modifications were introduced to the original training method to enhance train-

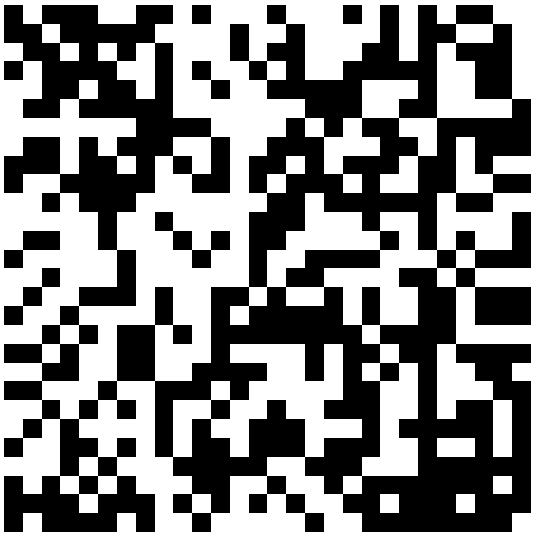

FIG. 5. Randomly generated Pseudo-QR code used as degradation

ing efficiency. Firstly, implementing a learning rate scheduler that decreases the learning rate over time. This modification allows for a smoother progression of the pseudo-QR code degradation, providing the model with a gentler learning curve during the early stages of training without overfitting to noisy, intermediate states. Secondly, weight decay is incorporated. The weight decay imposes a constraint on the magnitude of the model parameters, encouraging simpler models. Together, these adjustments refine the model's ability to generalise and reconstruct the original data from its degraded state, yielding improved performance on sampling.

To ensure the reversibility of the degradation process, we employed a sampling algorithm inspired by Algorithm 2 of the paper "Cold Diffusion: Inverting Arbitrary Image Transforms Without Noise" by Bansal *et al.* (2022). This algorithm is designed to invert arbitrary image transformations without relying on noise, making it well-suited for our application. By iteratively refining the predictions and adjusting the intensity of the pseudo-QR pattern, the sampling algorithm enables the recovery of the original digit from its degraded state.

Algorithm5 Reverse Process Using Pseudo QR Code

Require: Model gt, intensity schedule σ_t
Ensure: Reconstructed sample z_0
  Initialize z_T with a batch of pseudo QR-coded random images
  for $t = T, \ldots, 1$ **do**
    if $t > 1$ **then**
      $\sigma_t \leftarrow$ intensity at time step t from σ_t schedule
      $\sigma_{t-1} \leftarrow$ intensity at time step $t-1$ from σ_t schedule
      $x_0^{pred} \leftarrow gt(z_t, \frac{t}{T})$ {Predict the original image from z_t}
      $z_t \leftarrow z_t - \text{apply\_qr\_transformation}(x_0^{pred}, \sigma_t) + \text{apply\_qr\_transformation}(x_0^{pred}, \sigma_{t-1})$
    else
      $z_0 \leftarrow gt(z_t, \frac{t}{T})$ {Final reconstruction}
    end if
  end for

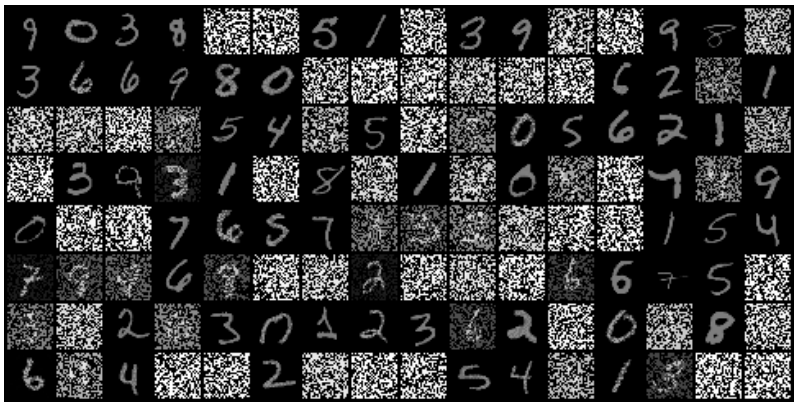

FIG. 6. Degraded Images at different time-steps

B. Part 2(b)

In the first custom method, the diffusion model was trained with an additional custom degradation on top of the standard Gaussian noise, but maintained the original sampling process of adding Gaussian noise at each step of the reverse diffusion, as per the DDPM protocol. This approach allows us to evaluate the model's ability to clean complex noise patterns. Although the model is exposed to both Gaussian and custom noise during training, sampling with Gaussian noise ensures the model's effectiveness in recovering the clean image. The custom degradation is applied to assess the model's proficiency in denoising realistically corrupted images, and the loss quantifies the model's success in reversing the totality of introduced noise.

1. Analysis of Final FID Scores

The final FID scores are interpreted as follows:
Mask+Contrast:

- Using my own function: 76.29757039478
- Using Bansal *et al.* (2022) functions: 83.31222372020312

Contrast Only:

- Using my own function: 68.43941096897
- Using Bansal *et al.* (2022) functions: 78.30155370263714

The Contrast Only condition has resulted in lower (better) FID scores compared to the Mask+Contrast condition across both functions. This shows that the model with only contrast perturbations is more capable of generating images that are closer to the real data distribution. The use of Mask and Contrast perturbations may have introduced complexities, thus slightly reducing the quality of generated samples as reflected by the higher FID scores and can be seen via the samples (Figure 7).

2. Custom degradation without Gaussian noise added

Initially, the loss of this model decreases rapidly as it learns from the gross errors it makes at the beginning of training (Figure 5). After the initial epochs, the rate of loss

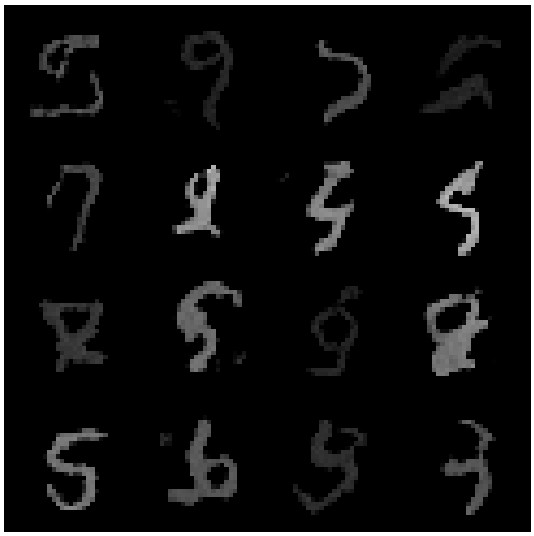

FIG. 7. Result of the mix ddpm

reduction slows down, which suggests that the model is starting to converge and fine-tune its parameters for more nuanced features of the data. Then the curve flattens out towards the later epochs, indicating that the model is reaching a point where further training does not significantly reduce the loss; normally, this is often considered as a good time to stop training to avoid overfitting. During the training on the notebook, the ReduceLROnPlateau scheduler was added, it is a scheduler that reduces the learning rate when a metric has stopped improving, in this case: loss. The FID scores couldn't be calculated as the difference in the distribution between the real images an dthe predictions were too big, hence, giving out complex numbers. Using the final FID score via the created custom fid function, it got 5e+94, which is really not great.

Each individual output of the final sampling exhibits varying degrees of structural coherence and fidelity relative to the MNIST aesthetic and dataset archetype. However, the granular quality of some images within the grid indicates an intermediate stage of detail synthesis. Furthermore, the presence of residual artefacts and varying contrast levels across the samples implies an asymmetrical convergence in the model's parameter space, with some instances reflecting premature halting or the potential need for further iterations to achieve optimal visual clarity since this was stopped at 100 epochs. Unfortunately, it seems that this model can't produce any digits, we can guess a few digits from some of these outputs but nothing concrete (Figure X), for the exception of 1 or 2 per batch samples as seen in Figure 8.

FIG. 8. Best samples from QR degradation ddpm model

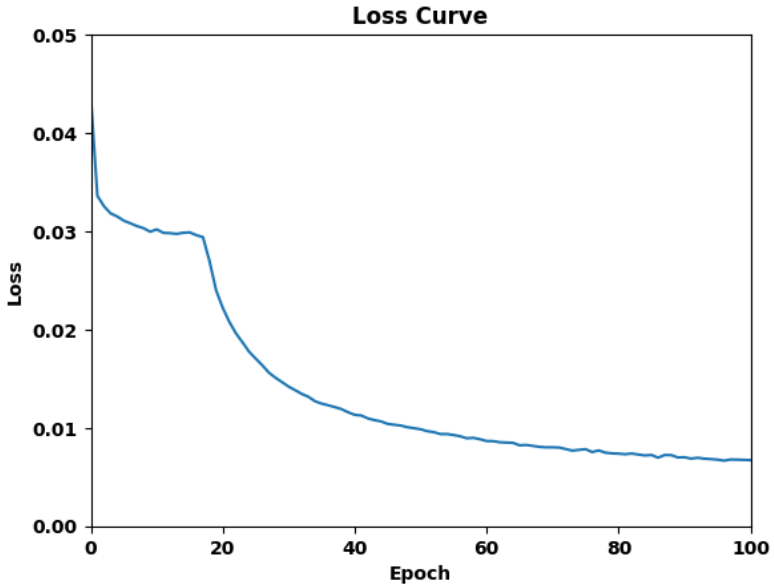

FIG. 9. Loss curve for the pure custom degradation based on pseudo QR codes

C. Part 2(c)

While Gaussian noise degradation is a common and simple approach, it may not capture the complexity of real-world degradation scenarios such as structured noise or patterns. In contrast, the custom degradation strategy, such as using a pseudo QR code pattern, introduces a structured and more representative degradation pattern. However, the custom degradation model has a harder time than the standard model representing the digits due to the complex and structured nature of the degradation introduced. While this can pose a challenge for the model, it also provides an opportunity to learn more robust features that can generalise better to diverse degradation conditions.

As for the mix model, based on the FID scores, it appears that the custom degradation strategies, particularly the Contrast Only Mixed with Gaussian Noise, perform better than the standard Gaussian noise degradation strategy. The lower FID scores suggest that the samples generated by your custom strategies are closer to the real data distribution, indicating higher fidelity.

Appendix A: Custom Degradation DDPM

This repository contains a custom degradation strategy for training diffusion models (DDPM) using PyTorch. The custom degradation strategy replaces the standard Gaussian noise degradation with a pseudo QR code transformation. The goal is to explore the effectiveness of this alternative degradation strategy in generating high-fidelity images, particularly focusing on datasets like MNIST.

1. Table of contents

1. Introduction[1]

[1] `#introduction`

2. Requirements[2]
3. Setup[3]
4. Usage[4]
5. Credits[5]

2. Introduction

Diffusion models are a class of generative models that learn a data distribution by modelling the diffusion process of a data sample. This repository explores a custom degradation strategy using pseudo QR codes instead of the traditional Gaussian noise. The aim is to explore whether this alternative degradation method can improve the fidelity of generated images.

3. Requirements

- Python 3.9 or later
- PyTorch
- torchvision
- NumPy

4. Setup

a. Using Conda

Create and activate the Conda environment:
`bash conda env create --name <env_name> -f environment.yml conda activate <env_name>`
Install dependencies if not using the Conda environment:
`bash pip install -r requirements.txt`

5. Usage

To train the DDPM model with the pure custom degradation strategy, run:
`bash python run_pure.py`
To train the DDPM model with the MIX custom degradation strategy, run:
`bash python run_mix.py`
To train the DDPM model with the gaussian standard degradation strategy with all different set of hyperparameters, run:
`bash python run_gaussian.py`
Ensure that you have the required datasets and adjust hyperparameters as needed in the scripts.

a. Documentation

- Auto-generated documentation using Doxygen. Run `doxygen` in the `/docs` folder to generate.

---

[2] `#requirements`
[3] `#setup`
[4] `#usage`
[5] `#credits`

- This README serves as the front page for the documentation.

b. Structure

The src directory contains the implementation of the DDPM and utility functions, while the notebooks directory provides Jupyter notebooks for experimentation and analysis. The data directory stores the MNIST dataset, and the papers directory includes relevant research papers. You can find the weights produced by the training on notebooks in the directory called "notebooks\notebooks_weights"

6. Credits

- The implementation of the diffusion model and training pipeline is based on the original DDPM paper (Ho et al. 2020).

7. Appendix: Use of Auto-generation Tools

Throughout the development of this project, ChatGPT by OpenAI was utilized for various purposes including code generation, prototyping, debugging assistance, and conceptual explanations. Below, I outline the instances of ChatGPT's use, detailing the nature of the prompts provided, the context in which the output was employed, and any modifications made to the generated content.

a. Code Generation and Prototyping

- **Prompts Submitted**: Queries were made to generate Python code snippets for loading and processing DICOM images, understanding a 3D CNN model in PyTorch, if it was needed to modify the 2D CNN model, and writing custom dataset classes for PyTorch DataLoader.
- **Usage**: Generated code was used as a foundation for the project's data preprocessing pipeline and model implementation. This included loading DICOM images, converting them to tensors, and defining the neural network architecture.
- **Modifications**: The code provided by ChatGPT was adapted to fit the specific requirements of the dataset and project objectives, on its own it didn't function with the provided dataset and context. This involved adjusting data loading mechanisms from my part to handle the dataset's unique structure, and optimizing performance.

b. Debugging Assistance

- **Prompts Submitted**: Assistance was requested for debugging issues related to Docker file, Custom Dataset, DataLoader, including errors with variable image sizes and tensor stacking.
- **Usage**: Suggestions from ChatGPT were employed to resolve runtime errors and improve the data loading process.
- **Modifications**: Debugging advice was integrated with existing code, with adjustments made to accommodate the specific data formats and processing goals.

c. Drafting and Proofreading

- **Prompts Submitted**: Requests were made for drafting sections of the README file and technical documentation, as well as for proofreading and suggesting alternative wordings.
- **Usage**: The output was utilised to enhance the clarity and completeness of project documentation.
- **Modifications**: Generated text was revised to better align with the coursework's scope, terminology, and presentation style.

d. Co-Pilot

- **Usage**: GitHub Copilot was used for code suggestion, completion, and documentation.

REFERENCES

Bansal, A., Borgnia, E., Chu, H., Li, J. S., Kazemi, H., Huang, F., Goldblum, M., Geiping, J., and Goldstein, T., "Cold diffusion: inverting arbitrary image transforms without noise," arXiv (Cornell University) (2022), 10.48550/arxiv.2208.09392.

Ho, J., Jain, A. N., and Abbeel, P., "Denoising diffusion probabilistic models," arXiv (Cornell University) (2020).

Prince, S. J., *Understanding Deep learning* (MIT Press, 2023).

Weng, L., "What are diffusion models?" lilianweng.github.io (2021).