Exercise 05 for MA-INF 2213 Computer Vision SS23
21.06.2023
Submission on 05.07.2023

Generative Diffusion Model. Figure 1 illustrates the forward and backward processes of a conditional diffusion model. The main idea is to train a neural denoising network that learns to transfer a standard normal distribution (not learned form data) into a target distribution (learned from the data) via a sequence T of denoising iterations. The denoising steps are further conditioned on a one-hot class label (denoted as c) to control what class to generate. During training, the target image (denoted as y_0) is mapped into a noisy image (e.g., y_t) and the model learns to predict the noise that brings y_t back into y_{t-1}. That is the training includes the forward process and minimizing the loss on the predicted noise. During inference, only the backward process is applied. In other words, when generating a new image, the trained network starts with a pure noisy image sampled from a normal distribution and one-hot class vector as a condition and iteratively removes the noise at various levels to generate a new image that belongs to the learned target distribution.

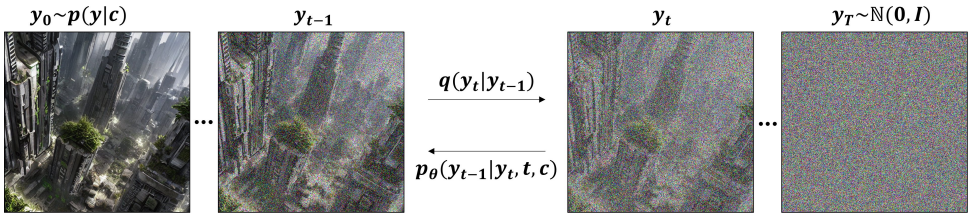
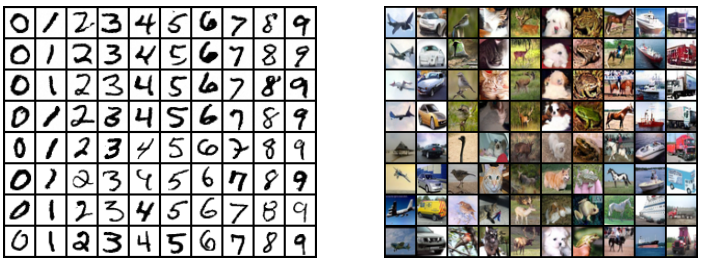


Figure 1: a simplified schematic overview of the diffusion process. The forward process $q(\mathbf{y}_t|\mathbf{y}_{t-1})$ (left to right) represents a Markovian chain which gradually adds Gaussian noise to the target image $(\mathbf{y}_0)$ until it becomes $(\mathbf{y}_t)$ after T steps. The inverse process $p_\theta(\mathbf{y}_{t-1}|\mathbf{y}_t, \mathbf{t}, \mathbf{c})$ (right to left) is parameterized by a neural denoising network that takes a random noise $\mathbf{y}_T \sim \mathcal{N}(0, I)$ and tries to generate $(\mathbf{y}_0)$ conditioned on the class label $(\mathbf{c})$. High level image features appear first and fine details appear last as t becomes smaller.

In this exercise, you will use a guided conditional diffusion model based on Denoising Diffusion Probabilistic Models (DDPM) [1]. In particular, you will implement a Classifier-free Diffusion Guidance [2] and train on two datasets, namely MNIST and CIFAR-10. The MNIST database contains 60k training grey images of handwritten digits with 10 classes $(0-9)$, each image has a spatial resolution of 28×28. While CIFAR-10 contains 50k training colored images of 10 classes and each image has a spatial resolution of 32×32.



(a) MNIST (b) CIFAR-10

Figure 2: generated samples by the guided diffusion model. First 4 rows are randomly generated samples by the model and the last 4 rows are real images randomly sampled from the training datasets.

You are provided with a project template that you are required to fill in with your ***own*** code. The template uses PyTorch Framework. You have to implement a small neural network based

on U-Net [3] that predicts the noise, fill in some missing lines of code for the diffusion process, train on MNIST and CIFAR-10 datasets, and answer the questions. More specifically:

1. **Define the U-Net model**
   Follow the implementation depicted in figures 3 and 4. The code for U-Net should be written in the file networks/Conditional_UNet.py.

   - Define the convolutional block with a residual connection as a class called *ConvBlock* which takes two parameters {input channels, output channels}. This block should contain two consecutive sub-blocks. The structure of each sub-block is the following:
     - Conv2D with kernel size of 3, stride of 1, and padding of 1
     - BatchNorm2D
     - GELU activation

     Check if the input and output channels parameters are equal. If they are equal apply the residual connection as (input tensor + output tensor of second sub-block). If this is not the case, apply the residual connection as (output tensor of the first sub-block + output tensor of the second sub-block). The final output after the residual connection should be normalized by $\sqrt{2}$. Define the forward pass of this class.
     (1 point)
   - Define a class called *Encoder_Block* which takes two parameters {input channels, output channels}. The encoder block contains two consecutive *ConvBlocks* (defined before) followed by a max pooling layer with kernel size and stride of 2. The output of the first *ConvBlock* should have number of channels equal to the parameter {output channel}. Define the forward pass of this class.
     (1 point)
   - Define a class called *Decoder_Block* which takes three parameters {input channels, output channels, transpose}. The decoder block has a transpose convolution with a kernel size and stride equal to the parameter {transpose} and followed by two consecutive *ConvBlocks*. Note that you can perform the concatenation with skip connection inside the decoder block. In this case the forward pass should take two arguments. Define the forward pass of this class.
     (1 point)
   - Define one class called *Embedding_Block* which takes three parameters {input channels, output channels, activation}. This block constitutes of two linear fully connected layers with an activation in between. The time embedding will take a sine function as an activation while the class embedding will take a GELU activation. The first linear layer increases the channels dimension while the second one keeps it. Define the forward pass of this class.
     (1 point)
   - Initialize the final block inside the *__init__* method of the class *UNet*. The structure of the final block is the following:
     - Conv2D with kernel size of 3, stride of 1, and padding of 1
     - GroupNorm with 8 groups
     - ReLU activation
     - Conv2D with kernel size of 3, stride of 1, and padding of 1 to map to the same number of channels as the input image

     The first Conv2D reduces the number of channels to the half.
     (1 point)

- Define the forward pass for the class *UNet*.
  (1.5 points)
- Do your own check under (if __name__ == '__main__':). For example, check the network with a dummy input for both CPU/GPU and print the number of parameters that require a gradient computation. For an architecture like figure 3 with intermediate channels of 32 and 10 classes, your model should have around 718k parameters.
  (0.5 points)

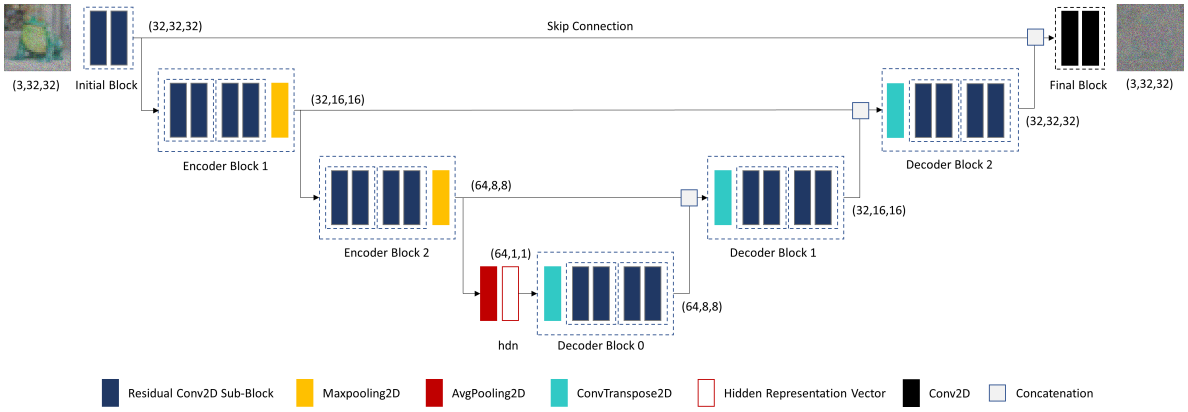


Figure 3: Overview of the U-Net model.

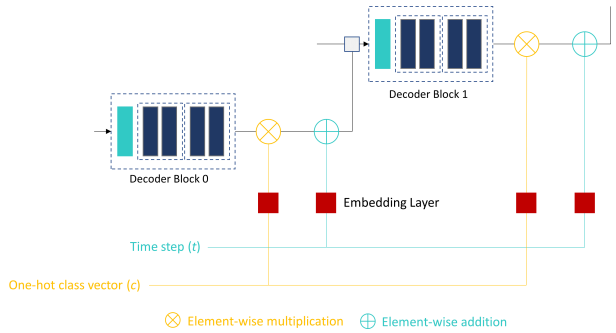


Figure 4: The U-Net model takes more information in the form of embedding. The image will be generated conditionally based on the embedding layers. There are two embedded input: the time step (t) and one-hot class vector (c).

2. Define the Denoising Diffusion Probabilistic Model (DDPM)

In diffusion models [1], the forward process or diffusion process is fixed to a Markov chain that gradually adds Gaussian noise to the data according to a variance schedule $\beta_1, \ldots, \beta_T$:

$$q(\mathbf{y}_{1:T}|\mathbf{y}_0) \coloneqq \prod_{t=1}^{T} q(\mathbf{y}_t|\mathbf{y}_{t-1}), \qquad q(\mathbf{y}_t|\mathbf{y}_{t-1}) \coloneqq \mathcal{N}(\mathbf{y}_t; \sqrt{1-\beta_t}\mathbf{y}_{t-1}, \beta_t\mathbf{I}) \tag{1}$$

Given this forward process, one can sample $\mathbf{y}_t$ at an arbitrary timestep t in a closed form solution: using the notation $\alpha_t \coloneqq 1 - \beta_t$ and $\bar{\alpha}_t \coloneqq \prod_{s=1}^{t} \alpha_s$, we have

$$q(\mathbf{y}_t|\mathbf{y}_0) = \mathcal{N}(\mathbf{y}_t; \sqrt{\bar{\alpha}_t}\mathbf{y}_0, (1-\bar{\alpha}_t)\mathbf{I}), \tag{2}$$

in simple terms:

$$\mathbf{y}_t = \sqrt{\bar{\alpha}_t}\mathbf{y}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{3}$$

In the inverse process, sampling $\mathbf{y}_{t-1} \sim p_\theta(\mathbf{y}_{t-1}|\mathbf{y}_t, \mathbf{t}, \mathbf{c})$ can be done with the following equation:

$$\mathbf{y}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{y}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{y}_t, \mathbf{t}, \mathbf{c}) \right) + \sqrt{\beta_t}\mathbf{z}, \tag{4}$$

where $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

This inverse process can be further enhanced via diffusion guidance [2] to control the trade-off between fidelity and diversity by linearly combine conditional and unconditional score estimates (predicted noises) based on the following equation:

$$\tilde{\boldsymbol{\epsilon}_\theta}(\mathbf{y}_t, \mathbf{t}, \mathbf{c}) = (1+w)\, \boldsymbol{\epsilon}_\theta(\mathbf{y}_t, \mathbf{t}, \mathbf{c}) \; - \; w \boldsymbol{\epsilon}_\theta(\mathbf{y}_t, \mathbf{t}), \tag{5}$$

where $\boldsymbol{\epsilon}_\theta(\mathbf{y}_t, \mathbf{t}, \mathbf{c})$ is the predicted noise by the neural network to map y_t into y_{t-1} conditioned on $\mathbf{c}$. To perform the unconditional generation, we mask the class label vector ($\mathbf{c} = \mathbf{0}$) randomly during training with a probability $p_{\text{unconditioned}} = 0.1$. For training, we will use the simplified loss function:

$$L(\theta) \coloneqq \mathbb{E}_{t, \mathbf{y}_0, \boldsymbol{\epsilon}} \left[\left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{y}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, \mathbf{t}, \mathbf{c}) \right\|^2 \right], \tag{6}$$

where t is uniform between 1 and T.

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

Require: $p_{\text{unconditioned}}$: probability of unconditional training
Require: $\mathbf{c}$: conditioning information for conditional sampling
Require: $\bar{\alpha}_1, \ldots, \bar{\alpha}_T$ from the variance schedule, where $\alpha_t \coloneqq 1 - \beta_t$ and $\bar{\alpha}_t \coloneqq \prod_{s=1}^{t} \alpha_s$
repeat
  $(\mathbf{y}_0, \mathbf{c}) \sim p(\mathbf{y}_0|\mathbf{c}, \mathbf{c})$ ▷ Sample data with conditioning from the dataset
  $t \sim \text{Uniform}(\{1, \ldots, T\})$ ▷ Sample timestep that determines the level of noise
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ ▷ Sample noise
  $\mathbf{c} \leftarrow \mathbf{0}$ with probability $p_{\text{unconditioned}}$ ▷ Randomly set c to zero with probability $p_{\text{unconditioned}}$ to discard conditioning to train unconditionally
  $\mathbf{y}_t = \sqrt{\bar{\alpha}_t}\mathbf{y}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}$ ▷ Perturb data to a specified noise level according to t
  Take gradient descent step on $\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{y}_t, \mathbf{t}, \mathbf{c}) \right\|^2$ ▷ Optimization of denoising model
until converged

Algorithm 2 Conditional sampling with classifier-free guidance

Require: w: guidance strength
Require: $\mathbf{c}$: conditioning information for conditional sampling
Require: $\beta_1, \ldots, \beta_T$ is the variance schedule
Require: $\bar{\alpha}_1, \ldots, \bar{\alpha}_T$ from the variance schedule, where $\alpha_t \coloneqq 1 - \beta_t$ and $\bar{\alpha}_t \coloneqq \prod_{s=1}^{t} \alpha_s$
$\mathbf{y}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ ▷ Sample noise
for $t = T, \ldots, 1$ **do**
  $\tilde{\boldsymbol{\epsilon}_\theta}(\mathbf{y}_t, \mathbf{t}, \mathbf{c}) = (1+w)\, \boldsymbol{\epsilon}_\theta(\mathbf{y}_t, \mathbf{t}, \mathbf{c}) \; - \; w \boldsymbol{\epsilon}_\theta(\mathbf{y}_t, \mathbf{t}, \mathbf{c} = \mathbf{0})$ ▷ Form the classifier-free guided score
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$ ▷ Sample extra noise
  $\mathbf{y}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{y}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \tilde{\boldsymbol{\epsilon}_\theta}(\mathbf{y}_t, \mathbf{t}, \mathbf{c}) \right) + \sqrt{\beta_t}\mathbf{z}$ ▷ Sampling step based on DDPM
end for
return $\mathbf{y}_0$

- Inside the file ddpm.py, you will find a class called *DDPM* which represents our diffusion model. The class has a method called *ddpm_schedule* that takes 3 arguments {beta1, beta2, n_timesteps} and returns a pre-computed schedule for DDPM

sampling, where {beta1} is β_1, {beta2} is β_T, and {n_timesteps} is the number of sampling steps T. Define the method *ddpm_schedule* in such a way that it creates $\beta_1, \ldots, \beta_T$ as a linearly increasing variance schedule and returns the schedule as a dictionary as it defined in the file.
(2 points)

- Define the forward pass for the class *DDPM* based on Algorithm 1.
  (2 points)
- Fill in the missing lines inside the method *sample* for the class *DDPM* with equations 4 and 5 following Algorithm 2.
  (1 point)

3. **Train on MNIST dataset using the file train_mnist.py**
   The script has predefined default hyper-parameters. We will use an Adam optimizer with a constant learning rate (change the training hyper-parameters as you think they would be suitable for your device and the task). The forward process variances are set to constants $\beta_1 = 10^{-4}$, $\beta_T = 0.02$. For testing, we will use three guidance weights ($w_1 = 0$, $w_2 = 1$, and $w_3 = 3$) and generate 4 samples for each class. Save the model and generate images like figure 2 for each w every 5 epochs. Save animated images every 5 epochs. After the training is done, visualize the progression of the training loss (do not visualize the smoothed loss but rather the mean loss for each epoch). The generated images and models should be saved in the folder log_mnist.
   (3 points)

4. **Train on CIFAR-10 dataset using the file train_cifar10.py**
   The script has predefined default hyper-parameters. We will use an Adam optimizer with a constant learning rate (change the training hyper-parameters as you think they would be suitable for your device and the task). The forward process variances are set to constants $\beta_1 = 10^{-4}$, $\beta_T = 0.02$. For testing, we will use two guidance weights ($w_1 = 1$, and $w_2 = 3$) and generate 4 samples for each class. Save the model and generate images like figure 2 for each w every 10 epochs. Save animated images every 20 epochs. After the training is done, visualize the progression of the training loss (do not visualize the smoothed loss but rather the mean loss for each epoch). The generated images and models should be saved in the folder log_cifar10. Do you need to change the capacity of the model? What hyper-parameters could improve the results?
   Note that your model may need more time to train on CIFAR-10 than MNIST dataset. If you want increase the interval for saving the output images.
   (3 points)

5. **Guidance.** What effect does the guidance weight w have on the generated images? Comment on the differences between the images generated with $w_1 = 0$, $w_2 = 1$, and $w_3 = 3$. In addition, discuss how you could guide the model to generate an image conditioned on a mixture of two or more classes.
   (1 point)

6. **Generative Diffusion.** Consider our implementation. What are the disadvantages of using a Classifier-free Diffusion model? What is the main drawback of using standard generative diffusion models?
   (1 point)

How to report on this assignment. Please exclude the data folder from your submission due to its size and include the following files in your solution:

- **networks/Conditional_UNet.py.** This should include the neural network model.

- **train_mnist.py.** This file includes the training and validating script for MNIST dataset.
- **train_cifar10.py.** This file includes the training and validating script for CIFAR-10 dataset.
- **utils.py.** Provided file includes helper functions for plotting.
- **ddpm.py.** File includes a class for training and sampling based on the denoising diffusion probabilistic process.
- **log_mnist/...** This folder should include the logs for training on the MNIST dataset.
- **log_cifar10/...** This folder should include the logs for training on the CIFAR-10 dataset.
- **text file** with your answers.

GPU Resources. If you do not have an adequate GPU on your machine, you could use the Bender - GPU Cluster by the HRZ at the University of Bonn (https://www.hpc.uni-bonn.de/en/systems/bender). As an alternative, you could also use Google Colab. This service provides an easy interface and free GPUs to train your models. In this case, please still use the reporting format with the files described above.

Questions If you have questions or find any ambiguities/mistakes, please contact (shams@iai.uni-bonn.de).

Important: Use Python 3.8 for your solution. You are **not** allowed to use any additional python modules beyond the ones imported in the template. Otherwise you won't get any points.

Grading: Submissions that generate runtime errors or produce obviously rubbish results (e.g. nans, inf or meaningless visual outputs) will receive at most 50% of the points.

Plagiarism: Plagiarism in any form is prohibited. If your solution contains code copied from any source (e.g. other students, or solutions from the web. This includes ChatGPT!), you will receive **0 points** for the entire exercise sheet.

Submission: You can complete the exercise in a group of two, but only one submission per group is allowed. Include a *README.txt* file with your group members into each solution. Points for solutions without a README file will only be given to the uploader.

References

[1] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[2] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

[3] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI: 18th International Conference, Munich, Germany, October 5-9, Proceedings, Part III 18*, pages 234–241. Springer, 2015.