

HOMework 2

GENERATIVE MODELS OF IMAGES *

10-423/623/723 GENERATIVE AI
<http://423.mlcourse.org>

OUT: Feb. 9, 2026
DUE: Feb. 21, 2026
TAs: Bhavesh, Raj, Yuchen

Instructions

- **Collaboration Policy:** Please read the collaboration policy in the syllabus.
- **Late Submission Policy:** See the late submission policy in the syllabus.
- **Submitting your work:** You will use Gradescope to submit answers to all questions and code.
 - **Written:** You will submit your completed homework as a PDF to Gradescope. Please use the provided template. Submissions can be handwritten, but must be clearly legible; otherwise, you will not be awarded marks. Alternatively, submissions can be written in $\text{\LaTeX}$. Each answer should be within the box provided. If you do not follow the template, your assignment may not be graded correctly by our AI assisted grader and there will be a **2% penalty** (e.g., if the homework is out of 100 points, 2 points will be deducted from your final score).
 - **Programming:** You will submit your code for programming questions to Gradescope. We will examine your code by hand and may award marks for its submission.
- **Materials:** The data that you will need in order to complete this assignment is posted along with the writeup and template on the course website.

*Compiled on Tuesday 10th February, 2026 at 18:10

Question	Points
$\LaTeX$ Template Alignment	0
Convolutional Neural Networks	8
Encoder-only Transformers	4
Generative Adversarial Network (GAN)	5
Variational Autoencoders	6
Understanding Diffusion Models	14
Programming: Diffusion Models	21
Code Upload	0
Collaboration Questions	2
Total:	60

1 $\text{\LaTeX}$ Template Alignment (0 points)

1.1. (0 points) **Select one:** Did you use $\text{\LaTeX}$ for the entire written portion of this homework?

☐ Yes

☐ No

1.2. (0 points) **Select one:** I have ensured that my final submission is aligned with the original template given to me in the handout file and that I haven't deleted or resized any items or made any other modifications which will result in a misaligned template. I understand that incorrectly responding yes to this question will result in a penalty equivalent to 2% of the points on this assignment.

Note: Failing to answer this question will not exempt you from the 2% misalignment penalty.

☐ Yes

2 Convolutional Neural Networks (8 points)

- 2.1. Suppose we define a convolution layer that takes as input a 3D tensor $\mathbf{x} \in \mathbb{R}^{C \times N_h \times N_w}$ which is indexed as $x_{c,i,j}$ where c selects the pixel channel, i selects the row of the pixel, and j selects the column of the pixel. The convolution parameters $\boldsymbol{\theta} \in \mathbb{R}^{C \times K_h \times K_w}$ are indexed in the same way. $\theta_0 \in \mathbb{R}$ is the intercept/bias parameter. The output 3D tensor has just a single channel, $\mathbf{y} \in \mathbb{R}^{1 \times N_h \times N_w}$.

$$y_{1,h,w} = \theta_0 + \sum_{c=1}^C \sum_{i=1}^{K_h} \sum_{j=1}^{K_w} \theta_{c,i,j} x_{c,m,n}, \quad (1)$$

$$\text{where } m = h - \left\lfloor \frac{K_h}{2} \right\rfloor + (i - 1) \text{ and } n = w - \left\lfloor \frac{K_w}{2} \right\rfloor + (j - 1) \quad (2)$$

$$\forall h \in \{1, \dots, N_h\}, w \in \{1, \dots, N_w\} \quad (3)$$

where we have written m and n as functions of w, K_h, K_w, i, j for notational convenience.

- 2.1.a. (3 points) **Short answer:** The valid indices for $x_{c,m,n}$ are $c \in \{1, \dots, C\}$, $m \in \{1, \dots, N_h\}$, $n \in \{1, \dots, N_w\}$. So, as defined, this convolution layer indexes into some values $x_{c,m,n}$ that do not exist! Let's call these non-existent pixels "hallucinated pixels" and assume they take value 0. How many *columns* of hallucinated pixels are needed on the left p_l and the right p_r ? How many *rows* of hallucinated pixels are needed on the top p_t and the bottom p_b ? Report your answer by defining p_l, p_r, p_t, p_b . Express each quantity using only **floor** functions.

Left padding p_l	Right padding p_r
Top padding p_t	Bottom padding p_b

- 2.1.b. (3 points) **Short answer:** Now suppose we create a new input 3D tensor $\mathbf{x}' \in \mathbb{R}^{C \times (N_h + p_t + p_b) \times (N_w + p_l + p_r)}$ by explicitly adding p_l columns on the left of $\mathbf{x}$, p_r columns on the right, p_t rows on top, and p_b rows on bottom—all the newly added columns/rows have value 0. These rows/columns are called *padding*. Define a new convolution layer by rewriting Equations (1),(2),(3) so that the input is $\mathbf{x}'$, the resultant output tensor $\mathbf{y}'$ still has shape $\mathbb{R}^{1 \times N_h \times N_w}$, and we only index into valid positions of $\mathbf{x}'$. The values of $\mathbf{y}'$ should be the same as those that would have been in $\mathbf{y}$ if hallucinated pixels were allowed in our original formulation.

2.1.c. **Conceptual Question:** U-Net is a convolutional neural network architecture widely used for image segmentation.

2.1.c.i. (1 point) What is the role of skip connections in U-Net, and why are they important for segmentation tasks?

2.1.c.ii. (1 point) How does the spatial resolution of feature maps change as they pass through the encoder and decoder?

3 Encoder-only Transformers (4 points)

- 3.1. (2 points) **Drawing:** Suppose we feed a sentence $w_1, \dots, w_N$ of length N into a *decoder-only* Transformer model (aka. Transformer LM), which defines a distribution $p(w_1, \dots, w_N)$. Draw a directed graphical model representing this probability distribution. Your drawing must include exactly N nodes, one node for each word w_n in the sentence. You may use ... to indicate omitted lines or nodes from your drawing. Optionally, feel free to define N to help you with your answer.

- 3.2. (2 points) Suppose we feed a sentence $w_1, \dots, w_N$ of length N into an *encoder-only* Transformer model, to obtain one output layer embedding $\mathbf{h}_n \in \mathbb{R}^D$ per word w_n . We then compute a score vector $\mathbf{s}_n$ per word w_n as follows:

$$\mathbf{s}_n = \exp(\mathbf{W}\mathbf{h}_n + \mathbf{b}), \quad \forall n \in \{1, \dots, N\}$$

where $\mathbf{W} \in \mathbb{R}^{V \times D}$ and $\mathbf{b} \in \mathbb{R}^V$, and V is the size of your output vocabulary. Assume your output vocabulary is the set of possible part-of-speech tags for the words in the input language, e.g. for English input, the parts of speech are nouns, verbs, adjectives, etc. Each tag is represented by an integer $1, \dots, V$.

Compare the types of sequence distributions that can be modeled by encoder-only Transformers versus decoder-only Transformers. Specifically:

- How do these models define probability distributions over sequences?
- What kinds of applications are each best suited for?

4 Generative Adversarial Network (GAN) (5 points)

- 4.1. Lora the Llama wants to define a GAN-inspired model for inpainting grayscale images of fellow llamas. She wants her images to have height N_h and width N_w .

Each original image is represented as a matrix:

$$\mathbf{x} \in (0, 1)^{N_h \times N_w},$$

where each element is a scalar between 0 and 1 (exclusive).

The image is accompanied by a binary pixel mask:

$$\mathbf{m} \in \{0, 1\}^{N_h \times N_w},$$

where:

$$m_{ij} = \begin{cases} 1, & \text{if the pixel should be masked out,} \\ 0, & \text{if the pixel should be left intact.} \end{cases}$$

Lora's model is a variant of U-Net that takes as input the image $\mathbf{x}$ and the mask $\mathbf{m}$, and returns a reconstructed image $\mathbf{x}'$ where masked pixels are filled in by the model:

$$\mathbf{x}' = g_\theta(\mathbf{x}, \mathbf{m}).$$

She decides to train her model using two loss functions in combination. Help Lora formulate parts of her model below.

```
def TrainModel( $\mathcal{D}$ ,  $g_\theta$ ,  $\eta$ ,  $E$ ,  $d_\phi$  = None) :
    #  $\mathcal{D}$ : dataset of images  $\mathbf{x}$ 
    #  $g_\theta$ : Lora's model with parameters  $\theta$ 
    #  $\eta$ : learning rate,  $E$ : number of epochs
    #  $d_\phi$ : (optional) discriminator with parameters  $\phi$ 

    for epoch in range(1, E + 1) :
        for  $x$  in  $\mathcal{D}$  :
            # 0) generate a random mask
             $m \sim$  binary matrix generator

            # 1) compute the output
             $x' =$  (a)

            # 2) compute loss
            if  $d_\phi$  = None:
                 $\ell_{mse}(\theta) =$  (b) # MSE loss
            else:
                 $\ell_{gan}(\phi, \theta) =$  (c) # GAN-style loss

            # 3) update the model (we assume the update of  $d_\phi$  is
            # handled properly)
             $\theta = \theta - \eta \cdot \nabla_\theta \ell$ 

    return  $\theta$ 
```

4.1.a. (1 point) Write math to fill in blank **(a)**

Hint: You may find an expression involving $(1 - m)$, x , and $g_\theta(\cdot)$ useful.

4.1.b. (1 point) Define a squared error loss $\ell_{mse}(\theta)$ for one example $(\mathbf{x}, \mathbf{m})$ to fill in blank **(b)**

Hint: Ensure that pixels outside of the masked area do not affect the loss.

4.1.c. (2 points) Given a discriminator $d_\phi(x)$ that outputs the probability that x is a real image (not generated by g_θ). Write a GAN-style objective function to fill in blank **(c)**.

Hint: Use your expression of x' from question [4.1.a.](#)

4.1.d. (1 point) Describe one possible disadvantage of training with only $\ell_{mse}(\theta)$ as compared to using this combined training approach.

5 Variational Autoencoders (6 points)

Variational Autoencoders (VAEs) are generative models that learn to encode data into a latent space and then decode it back to reconstruct the original data.

Unlike standard autoencoders, VAEs impose a probabilistic structure on the latent space by learning distributions rather than deterministic mappings.

The key components of a VAE are: (1) an **encoder** that maps input data $\mathbf{x}$ to parameters of a latent distribution $q_\phi(\mathbf{z}|\mathbf{x})$, (2) a **decoder** that maps latent samples $\mathbf{z}$ to a reconstruction distribution $p_\theta(\mathbf{x}|\mathbf{z})$, and (3) a **prior** distribution $p(\mathbf{z})$ (typically standard normal).

The VAE is trained by maximizing the Evidence Lower BOund (ELBO), which consists of a reconstruction term and a regularization term (KL divergence between the approximate posterior and prior).

5.1. (2 points) **Short answer:** Consider the reconstruction term of a VAE,

$$\mathcal{R}(\phi, \theta) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x} | \mathbf{z})].$$

We estimate this expectation using a single Monte Carlo sample

$$\mathbf{z} \sim q_\phi(\mathbf{z} | \mathbf{x}), \quad \widehat{\mathcal{R}} = \log p_\theta(\mathbf{x} | \mathbf{z}),$$

and treat the sampling operation “ $\mathbf{z} \sim q_\phi(\mathbf{z} | \mathbf{x})$ ” as a black-box node in the computation graph.

Assume the following parameterization:

Encoder: $q_\phi(\mathbf{z} | \mathbf{x}) = \mathcal{N}(\mu_\phi(\mathbf{x}), \sigma_\phi^2(\mathbf{x}))$ where $\mu_\phi(\mathbf{x}) = w_1\mathbf{x} + b_1$, $\sigma_\phi^2(\mathbf{x}) = \exp(w_2\mathbf{x} + b_2)$

Decoder: $p_\theta(\mathbf{x} | \mathbf{z}) = \mathcal{N}(\mu_\theta(\mathbf{z}), \sigma^2)$ where $\mu_\theta(\mathbf{z}) = w_3\mathbf{z} + b_3$, $\sigma^2 = 0.1$

The parameters are $\phi = \{w_1, b_1, w_2, b_2\}$ and $\theta = \{w_3, b_3\}$. The prior is $p(\mathbf{z}) = \mathcal{N}(0, 1)$.

(a) Under this computation graph (black-box sampling), which of the parameters $w_1, b_1, w_2, b_2, w_3, b_3$ can receive gradients from $\widehat{\mathcal{R}}$?

(b) Now suppose we instead use the reparameterization trick,

$$\mathbf{z} = \mu_\phi(\mathbf{x}) + \sigma_\phi(\mathbf{x}) \odot \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, I).$$

Under this computation graph, which of the parameters $w_1, b_1, w_2, b_2, w_3, b_3$ receive gradients from $\widehat{\mathcal{R}}$?

- 5.2. (3 points) Let's consider how we might learn a representation for videos with a VAE. Suppose we have a batch of N videos $x_i|_{i=1}^N$, with each $x_i \in \mathbb{R}^{3 \times F \times H \times W}$. We will keep the same architecture and parameters as 5.1, and use Mean Squared Error (MSE) as our reconstruction loss as opposed to trying to calculate $\log p_\theta(x|z)$. For the MSE loss, normalize each sample's loss by number of pixels in the sample.

Let the outputs of the decoder be $\tilde{x}_i|_{i=1}^N$ with each $\tilde{x}_i \in \mathbb{R}^{3 \times F \times H \times W}$, and let the outputs of the encoder be $(\mu_\phi(x_i), \sigma_\phi^2(x_i))|_{i=1}^N$, with $\mu_\phi(x_i) \in \mathbb{R}$, $\sigma_\phi^2(x_i) \in \mathbb{R}^+$.

Derive the exact numerical loss for this single batch of VAE inputs/outputs in terms of $x_i, \tilde{x}_i, \sigma_\phi^2, \mu_\phi$, and constants. Assume that we multiply the sum of the individual losses by $\frac{1}{\text{batch size}}$, and assume that we multiply the KL term by some weight factor β .

Hint: The KL divergence between Gaussians is:

$$D_{KL}(\mathcal{N}(\mu_0, \sigma_0^2) \parallel \mathcal{N}(\mu_1, \sigma_1^2)) = \frac{1}{2} \left(\log \frac{\sigma_1^2}{\sigma_0^2} + \frac{\sigma_0^2 + (\mu_0 - \mu_1)^2}{\sigma_1^2} - 1 \right)$$

- 5.3. (1 point) Consider a VAE where we gradually increase the weight of the KL divergence term in the ELBO loss function (i.e., increase the β parameter in β -VAE where the loss is $\mathcal{L} = \text{Reconstruction Loss} + \beta \cdot \text{KL Divergence}$). Assume a standard normal prior, $\mathcal{N}(0, I)$. As β gradually increases from 0, what happens to the learned latent representations?
- ☐ The distribution $q_\phi(z|x)$ approaches $\mathcal{N}(0, I)$ but reconstructions become worse
 - ☐ The distribution $q_\phi(z|x)$ goes farther from $\mathcal{N}(0, I)$ and reconstructions become better
 - ☐ The distribution $q_\phi(z|x)$ approaches $\mathcal{N}(0, I)$ and reconstructions become better
 - ☐ The distribution $q_\phi(z|x)$ goes farther from $\mathcal{N}(0, I)$ and reconstructions become worse

6 Understanding Diffusion Models (14 points)

Introduction

Diffusion models have transformed data generation, demonstrating remarkable success in text-conditioned image generation. In this section, we explore the variational interpretation of diffusion models

Given observed samples $\mathbf{x}$ from a target distribution, the goal of a generative model is to learn an approximation of the true data distribution, $p(\mathbf{x})$. Once learned, this model allows us to generate new samples at will. In many situations, we can imagine the data $\mathbf{x}$ we see as coming from a latent representation $\mathbf{z}$ responsible for capturing abstract properties that we can't directly observe. Mathematically, we can imagine the latent variables and the data we observe as modeled by a joint distribution $p(\mathbf{x}, \mathbf{z})$.

Directly computing and maximizing the likelihood $p(\mathbf{x})$ is difficult, so instead, we maximize a lower bound:

$$\log p(\mathbf{x}) \geq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z} | \mathbf{x})} \right]. \quad (4)$$

$q_\phi(\mathbf{z} | \mathbf{x})$ is called an encoder and can be any distribution with parameters ϕ .

Diffusion Probabilistic Models (Ho et al., 2020) can be viewed as a sequence, of length T , of latent variables which all have the same dimensionality with the data:

$$p(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \times \prod_{t=1}^T p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t), \quad (5)$$

where $p(\mathbf{x}_0)$ is the data we observe, $\mathbf{x}_{1:T}$ are the latent variables of the model, and $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Equation (5) is also called the reverse process of the diffusion model.

The encoder or forward process of a diffusion model is:

$$q(\mathbf{x}_{1:T} | \mathbf{x}_0) = \prod_{t=1}^T q(\mathbf{x}_t | \mathbf{x}_{t-1}). \quad (6)$$

The mean and variance of the encoder of a diffusion model are predefined. Therefore, the encoder of Equation 6 does not have any learnable parameters ϕ .

The ELBO (Equation 4) for the diffusion model described by Equations 5, 6 becomes:

$$\log p(\mathbf{x}) \geq \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T} | \mathbf{x}_0)} \right]. \quad (7)$$

To keep track of the various expressions introduced, here's a table of their English interpretations.

Symbol	Description
$p(\mathbf{x})$	True data distribution
$q_\phi(\mathbf{z} \mid \mathbf{x})$	Encoder distribution (variational approximation)
$p_\theta(\mathbf{x}, \mathbf{z})$	Joint model distribution
$p(\mathbf{x}_{0:T})$	Joint distribution of the full Markov chain in the diffusion model
$p(\mathbf{x}_T)$	Prior distribution (typically standard Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$)
$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$	Reverse (denoising) process in diffusion models
$q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)$	Forward process of the diffusion model
$q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$	Forward transition probability
$p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)$	First-step reconstruction probability
$\mathcal{L}_t$	KL divergence penalty enforcing accurate reverse process
$\mathbb{E}_{q(\cdot)}$	Expectation under distribution q
$\mathcal{N}(\mathbf{0}, \mathbf{I})$	Isotropic Gaussian prior

Table 1: Glossary of Terms and Symbols

ELBO Surgery

6.1. (5 points) Show that we can break down Equation (7) as:

$$\log p(\mathbf{x}) \geq \mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)}[\log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)] - \sum_{t=1}^{T-1} \underbrace{\mathbb{E}_{q(\mathbf{x}_{t-1}, \mathbf{x}_t, \mathbf{x}_{t+1}|\mathbf{x}_0)} \left[\log \frac{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})}{p_\theta(\mathbf{x}_t \mid \mathbf{x}_{t+1})} \right]}_{\mathcal{L}_t} + C, \quad (8)$$

where C is a constant term that does not depend on θ .

Hints:

1. Start from the ELBO expression:

$$\mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \right].$$

2. Expand both the forward process $q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)$ and the reverse model $p_\theta(\mathbf{x}_{0:T})$.

3. Separate terms into R (reconstruction term) + C + L_T .

4. To simplify the expectations, recall:

$$\mathbb{E}_{q(a,b,c)}[\log q(b, c)] = \mathbb{E}_{q(b,c)}[\log q(b, c)],$$

which lets you marginalize out a , an irrelevant variable.

5. Use the Markov property: given $\mathbf{x}_t$, the past ($\mathbf{x}_{<t}$) and future ($\mathbf{x}_{>t}$) are conditionally independent. This lets you reduce expectations down to only the triples $(\mathbf{x}_{t-1}, \mathbf{x}_t, \mathbf{x}_{t+1})$.
6. Check carefully which parts depend on θ and which do not. This justifies treating the prior term as a constant C .

Note: Proofs are scary! Often done with sounds of frustration (e.g., Lora the Llama will be like “Ahhhh”). With the above high-level guidelines, we hope to make it easier for you, and **we expect your proof to be primarily equations with some brief verbal explanations if needed.**

- 6.2. (2 points) Can you explain in words what is the effect of the term $\mathcal{L}_t$ on the reverse process $p_\theta(\mathbf{x}_t \mid \mathbf{x}_{t+1})$ of the diffusion model when we try to maximize the ELBO in Equation (8) and why? When is the term $\mathcal{L}_t$ maximized?

Image Diffusion

- 6.3. (2 points) Assume the encoder of the diffusion model at step t is given by:

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{\alpha_t} \mathbf{x}_{t-1}, (1 - \alpha_t) \mathbf{I}), \alpha_t > 0. \quad (9)$$

Describe a way to obtain a sample of the diffusion process at timestep $t = \tau$. Also, state the time complexity of your algorithm as a function of τ .

- 6.4. (3 points) **Reparameterization trick.** The reparameterization trick is a powerful mathematical tool that allows us to generate samples of any Gaussian distribution $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \sigma^2 \mathbf{I})$ by sampling the standard normal distribution using the following transformation:

$$\mathbf{x} = \boldsymbol{\mu} + \sigma \boldsymbol{\epsilon}, \text{ where } \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \quad (10)$$

Use the reparameterization trick (Equation 10) to show that we can write the encoder of Equation 9 as:

$$q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t} \mathbf{x}_0, (1 - \bar{\alpha}_t) \mathbf{I}), \text{ where } \bar{\alpha}_t = \prod_{i=1}^t \alpha_i. \quad (11)$$

- 6.5. (2 points) Describe a way to obtain a sample of the forward diffusion process at timestep $t = \tau$ using the formulation of Equation 11. Also, state the time complexity of your algorithm as a function of τ .

7 Programming: Diffusion Models (21 points)

Introduction

In this section, you will dive into the practical aspects of implementing the diffusion model we saw in Problem 6. Throughout this programming assignment, you will gain hands-on experience into state-of-the-art techniques for image generation and denoising tasks.

It's worth noting that, due to limited computing resources, the dataset provided for this exercise is only a subset of the original dataset. Therefore, the quality of the generated images may not meet expectations. Nevertheless, your experimentation with DDPM will offer valuable insights into its capabilities and potential for broader applications in machine learning and data analysis. Upon completion, you'll have acquired practical experience in building and leveraging DDPMs, opening doors to a deeper understanding of diffusion models.

Dataset

The dataset for this homework is the Animal Faces-HQ dataset ([AFHQ](#)), consisting of 15,000 images at 36×36 resolution. The dataset includes three domains of cat, dog, and wildlife, and in our assignment you **only need to use cat** images to reduce the computation complexity.

Starter Code

The main structure of the files is organized as follows:

```
hw2/  
  data/  
  diffusion.py  
  test_diffusion.py  
  main.py  
  requirements.txt  
  run_in_colab.ipynb  
  run_in_kaggle.ipynb  
  trainer.py  
  unet.py  
  utils.py
```

Here is what you will find in each file:

1. `data`: Contains the AFHQ dataset.
2. `diffusion.py`: Constructs the diffusion model, including the forward process, backward process, and scheduler, which you will implement.
3. `main.py`: Serves as the main entry point for training and evaluating your diffusion model IF you are running locally or on AWS. You won't need this file if you are running on Colab or Kaggle. Append flags to this command to adjust the diffusion model's configuration.
4. `requirements.txt`: Lists the packages that need to be installed for this assignment.
5. `run_in_colab/kaggle.ipynb`: Provides command lines to train and evaluate your diffusion model in Google Colab or Kaggle.
6. `trainer.py`: Provides code for training and evaluating the diffusion model.

7. `unet.py`: Contains code for the U-Net network, which aims to model the denoising function for the diffusion model.
8. `utils.py`: Helper functions to simplify the process of training or evaluating your diffusion model.
9. `test_diffusion.py`: You can use this file to test your implementations of *diffusion.py* and *unet.py*. Run it using ``python test_diffusion.py``

For detailed instructions on how to use Google Colab / Kaggle see Homework 1 Handout.

(Hint: *diffusion.py* and *unet.py* are the **only** files you need to modify. Locations in the code where changes ought to be made are marked with a **TODO**.)

Parameters

In table 2,3 and 4, you will find all of the parameters which can be configured in the starter code. You can set these parameters in functions `train_diffusion` or `visualize_diffusion` as seen in `run_in_colab/kaggle.py`.

Description	Parameter	Default Value
Directory from which to load data	<code>data_path</code>	(See starter notebook)
Number of iterations to train the model	<code>train_steps</code>	(See handout below)
Enable FID calculation	<code>fid</code>	(See handout below)
Frequency of periodic save, sample and (optionally) FID calculation	<code>save_and_sample_every</code>	(See handout below)

Table 2: Useful parameters for `run_in_colab/kaggle.ipynb`

Description	Parameter	Default Value
Dataloader worker threads	<code>dataloader_workers</code>	16
Directory where the model is stored	<code>save_folder</code>	<code>./results/</code>
Path of a trained model	<code>load_path</code>	<code>./results/model.pt</code>

Table 3: Additional parameters for `run_in_colab/kaggle.ipynb`. You likely won't need to change these.

Description	Parameter	Default Value
Model image size	<code>image_size</code>	32
Model batch size	<code>batch_size</code>	32
Data domain of AFHQ dataset	<code>data_class</code>	cat
Number of steps of diffusion process, T	<code>time_steps</code>	50
Number of output channels of the first layer in U-Net	<code>unet_dim</code>	16
Learning rate in training	<code>learning_rate</code>	1e-3
U-Net architecture	<code>unet_dim_mults</code>	[1, 2, 4, 8]

Table 4: Additional parameters for `run_in_colab/kaggle.ipynb`. These won't need to be changed from default values for this homework.

Diffusion

In this problem, you will implement Denoising Diffusion Probabilistic Models (DDPM) (Ho et al., 2020), in the `Diffusion` class in `diffusion.py`.

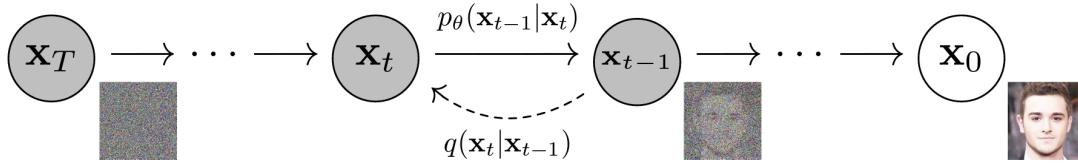


Figure 1: The Markov chain of forward (reverse) diffusion process of generating a sample by slowly adding (removing) noise.

Forward Process (Noise $\leftarrow$ Image): In this problem, $\mathbf{x}_0 \sim q(\mathbf{x})$ corresponds to the pixels of the image. As we saw in Problem 6, the *forward diffusion* process sequentially applies a small amount of Gaussian noise to the data sample $\mathbf{x}_0$ for T steps, producing a sequence of noisy samples $\mathbf{x}_1, \dots, \mathbf{x}_T$. In Equation 9, we derived the diffusion step:

$$q(\mathbf{x}_t | \mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{\alpha_t} \mathbf{x}_{t-1}, 1 - \alpha_t \mathbf{I}), \quad (12)$$

where $\mathbf{x}_t$ is the image after t diffusion steps, $\mathbf{I}$ is the identity matrix. The step sizes are controlled by a variance schedule $\{\alpha_t \in (0, 1)\}_{t=0}^{T-1}$ such that the data sample $\mathbf{x}_0$ gradually loses its distinguishable features as step t becomes larger. This is shown in Fig. 1.

In Problem 6.4, we used the reparameterization trick to sample $\mathbf{x}_t$ directly from $\mathbf{x}_0$:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon, \text{ where } \epsilon \sim \mathcal{N}(0, \mathbf{I}). \quad (13)$$

Noise Schedule: In this assignment, we use the improved cosine-based variance schedule of (Nichol & Dhariwal, 2021):

$$\alpha_t = \text{clip}\left(\frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}, 0.001, 1\right), \bar{\alpha}_t = \frac{f(t)}{f(0)}, \quad (14)$$

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

and we set $s = 0.008$ to prevent α_t from becoming too large when close to $t = 0$.

Reverse Process (Noise $\rightarrow$ Image): Ho et al. (2020) proved that the ELBO for a diffusion model can be rewritten as:

$$\log p(\mathbf{x}) \geq \mathbb{E}_{q(\mathbf{x}_1 | \mathbf{x}_0)} [\log p_\theta(\mathbf{x}_0 | \mathbf{x}_1)] - \sum_{t=2}^T \underbrace{\mathbb{E}_{q(\mathbf{x}_t | \mathbf{x}_0)} [D_{KL}(q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0), p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t))]}_{\mathcal{L}'_t} + C, \quad (15)$$

(see Appendix A in their paper linked above). The reverse model $p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)$ as shown in Fig. 1 is trained to maximize the lower bound of Equation 15. Note that the forward process $q(\mathbf{x}_t | \mathbf{x}_{t-1})$ does not contain any trainable parameters.

The distributions $q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0)$ inside the $\mathcal{L}'_t$ terms of Equation 15 can act as a “ground-truth signal”, since they define how to denoise a noisy image $\mathbf{x}_t$ with access to what the final, completely denoised

image $\mathbf{x}_0$ should be. Using the Markov properties of a diffusion model, it is possible to show that the distribution $q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0)$ decomposes as

$$q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0) = \frac{q(\mathbf{x}_t | \mathbf{x}_{t-1})q(\mathbf{x}_{t-1} | \mathbf{x}_0)}{q(\mathbf{x}_t | \mathbf{x}_0)}. \quad (16)$$

Conveniently, we have already derived the three terms of Equation 16. In particular, $q(\mathbf{x}_t | \mathbf{x}_{t-1})$ is given by Equation 9. From Problem 6.4, we also know that:

$$q(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1 - \bar{\alpha}_t)\mathbf{I}), \text{ where } \bar{\alpha}_t = \prod_{i=1}^t \alpha_i. \quad (17)$$

This derivation can be modified to also yield the Gaussian parameterization describing $q(\mathbf{x}_{t-1} | \mathbf{x}_0)$. After tedious numerical combinations to combine the three Gaussian terms in Equation 16, we obtain:

$$\begin{aligned} q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0) &= \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\boldsymbol{\Sigma}}_t), \text{ where:} \\ \tilde{\boldsymbol{\mu}}_t &= \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} \mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}(1 - \alpha_t)}{1 - \bar{\alpha}_t} \mathbf{x}_0, \\ \tilde{\boldsymbol{\Sigma}}_t &= \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} (1 - \alpha_t) \mathbf{I} = \sigma_t^2 \mathbf{I}. \end{aligned} \quad (18)$$

We have therefore shown that at each step, $q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0)$ is normally distributed, with mean $\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0)$ that is a function of $\mathbf{x}_t$ and $\mathbf{x}_0$, and variance $\tilde{\boldsymbol{\Sigma}}_t$ as a function of $\bar{\alpha}_t$ coefficients. In order to match approximately the denoising transition step $p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)$ to ground-truth denoising transition step $q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0)$ as closely as possible, we can also model it as a Gaussian. Furthermore, since $\tilde{\boldsymbol{\Sigma}}_t$ is a priori known during training, we can immediately construct the variance of the approximate denoising transition step to also be $\tilde{\boldsymbol{\Sigma}}_t$. Therefore, to define $p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)$, we only need to find its mean $\boldsymbol{\mu}_\theta(\mathbf{x}_t, t)$. $\boldsymbol{\mu}_\theta(\mathbf{x}_t, t)$ is implemented by a neural network that only takes $\mathbf{x}_t$ as an input, and not $\mathbf{x}_0$. This is because $p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)$ is conditioned only on $\mathbf{x}_t$ and not $\mathbf{x}_0$.

Under these assumptions, one can show that minimizing the KL divergence in Equation 15 boils down to learning a neural network to predict the original ground truth image $\mathbf{x}_0$ from an arbitrarily noisified version of it $\mathbf{x}_t$.

Going one step further, one can show that this is equivalent to training a neural network $\epsilon_\theta(\mathbf{x}_t, t)$ that learns to predict the source noise ϵ that determines $\mathbf{x}_t$ from $\mathbf{x}_0$. This can be understood by rearranging the terms in Equation 13:

$$\mathbf{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}} (\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t} \epsilon). \quad (19)$$

Reverse Process Model: The reverse process model ϵ_θ is defined in `unet.py` and is an implementation of a CNN called U-Net, as illustrated in Fig. 2. U-Net's role here is to model the denoising function at each step of the reverse diffusion process. The architecture's ability to handle details at multiple scales and its effectiveness in capturing both local and global features make it well-suited for the task of denoising in diffusion models. By predicting the noise that was added at each step of the forward diffusion process, the U-Net helps to gradually reconstruct the data sample from noise. You are **required** to implement the `forward` function within the `UNet` class.

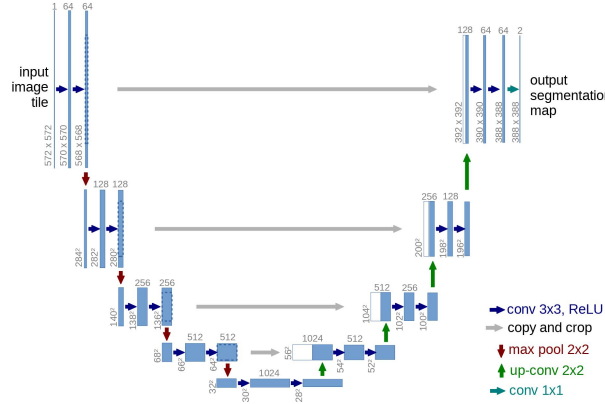


Figure 2: The structure of U-Net.

Training: The training algorithm is described in Alg. 1. We utilize a minibatch of data to train our reverse process model, denoted as ϵ_θ , which estimates the noise introduced during the forward diffusion process. You are **required** to implement the function `q_sample`, `p_losses`, and `forward` within the `Diffusion` class. The `p_losses` function defines the training loss using the L_1 loss. Additionally, we set a noise scheduler and pre-define some coefficients in the `__init__` function for efficient reuse, so you should also fill in the blanks there.

Algorithm 1 Training

- 1: **repeat**
 - 2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
 - 3: $t \sim \text{Uniform}(\{0, \dots, T-1\})$
 - 4: $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
 - 5: $\mathbf{x}_t \leftarrow \sqrt{\alpha_t} \mathbf{x}_0 + \sqrt{1 - \alpha_t} \epsilon$ ▷ forward diffusion process
 - 6: Take optimizer step on L_1 loss, $\nabla_\theta \|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\|_1$
 - 7: **until** converged
-

Sampling: The sampling algorithm is described in Alg. 2. The real implementation considers a mini-batch of samples, and use `gather_timestep_coeff` function to extract coefficients for batched operation. You are **required** to implement function `p_sample`, `p_sample_loop`, and `sample` in the `Diffusion` class that defines the reverse diffusion process to generate images.

Algorithm 2 Sampling

```

1:  $\mathbf{x}_{T-1} \sim \mathcal{N}(\mathbf{0}, I)$ 
2: for  $t = T - 1, \dots, 0$  do
3:    $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, I)$  if  $t > 0$ , else  $\mathbf{z} = \mathbf{0}$ 
4:    $\boldsymbol{\epsilon}_t \leftarrow \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$  ▷ predicted noise
5:    $\hat{\mathbf{x}}_0 \leftarrow \frac{1}{\sqrt{\bar{\alpha}_t}} (\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}_t)$  ▷ estimated  $\hat{\mathbf{x}}_0$ 
6:    $\hat{\mathbf{x}}_0 \leftarrow \text{clamp}(\hat{\mathbf{x}}_0, -1, 1)$  ▷ rectify  $\hat{\mathbf{x}}_0$ 
7:    $\tilde{\boldsymbol{\mu}}_t \leftarrow \frac{\sqrt{\bar{\alpha}_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} \mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}(1 - \alpha_t)}{1 - \bar{\alpha}_t} \hat{\mathbf{x}}_0$  ▷ posterior mean of  $x_{t-1}$ 
8:    $\sigma_t^2 \leftarrow \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} (1 - \alpha_t)$  ▷ posterior variance of  $x_{t-1}$ 
9:    $\mathbf{x}_{t-1} \leftarrow \tilde{\boldsymbol{\mu}}_t + \sigma_t \mathbf{z}$  ▷ reverse diffusion process
return  $\mathbf{x}_0$ 

```

Evaluation: To gauge the improvement in generative prowess throughout the training process, calculate the Fréchet Inception Distance (FID) between the training dataset and the generated samples from the current model. FID serves as a crucial metric in assessing the quality of generated data, providing a quantitative measure that goes beyond traditional visual inspection.

FID is a widely adopted metric in the realm of generative models, offering a robust evaluation of the dissimilarity between the true data distribution and the generated distribution. By incorporating both the mean and covariance of feature representations extracted from a pre-trained neural network, FID captures nuanced differences and similarities, offering valuable insights into the fidelity of generated samples.

We use `clean-fid` package to easily compute the FID score between resized training images and generated images.

Diffusion Empirical Questions

Clarification: The code to generate the following figures are **already provided**, you can get figures in wandb once you complete the diffusion part.

- 7.1. (4 points) **Training:** Plot the training loss of your Diffusion model above over 1,000 training steps with the recommended parameters in the above command line. Your model should be generating blurry cats at this point, similar to the image below. Recommended parameters: `train_steps=1000, save_and_sample_every=100, fid=False`.

[Expected runtime on Colab T4: 5-10 minutes]

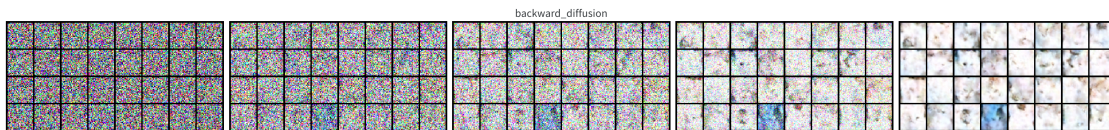


Figure 3: The Backward Diffusion Process, after just 1000 steps.



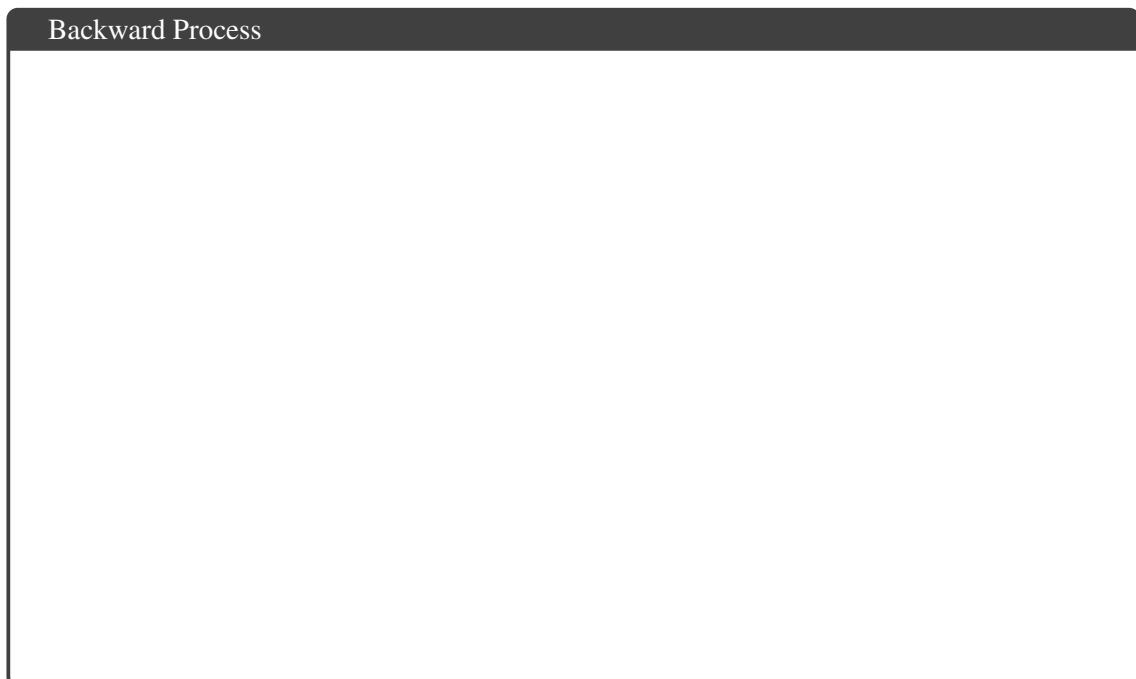
- 7.2. (4 points) **Training:** During training time, the starter code uses the ‘compute_fid’ function from ‘clean-fid’ to compute the FID value between training samples and generated samples. Get the FID value every 100 training steps and plot it over 1,000 training steps with the recommended parameters in the above command line. Recommended parameters: `train_steps=1000, save_and_sample_every=100, fid=True`.

[Expected runtime on Colab T4: 15-60 minutes]



- 7.3. (5 points) **Visualization:** Train the model for a full 10,000 iterations and show the images generated in the last sample batch. The images should be a substantial improvement over training with fewer iterations. Recommended parameters: `train_steps=10000, save_and_sample_every=1000, fid=False`

[Expected runtime on Colab T4: 2 hours]



- 7.4. (4 points) **Visualization:** Use the trained model after 10,000 steps to illustrate the forward diffusion process on the initial batch of the training dataset at key time intervals: 0%, 25%, 50%, 75%, and 99% of the total timesteps. The resulting figure should resemble the provided sample, though the images will vary due to inherent randomness.

Hint: you can find this figure on Colab after calling `visualize_diffusion`

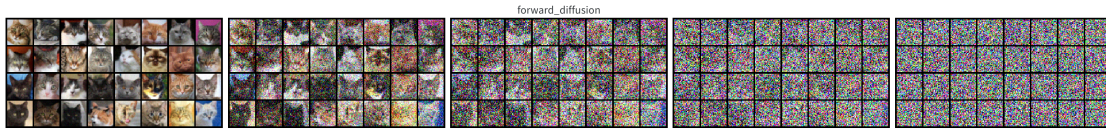
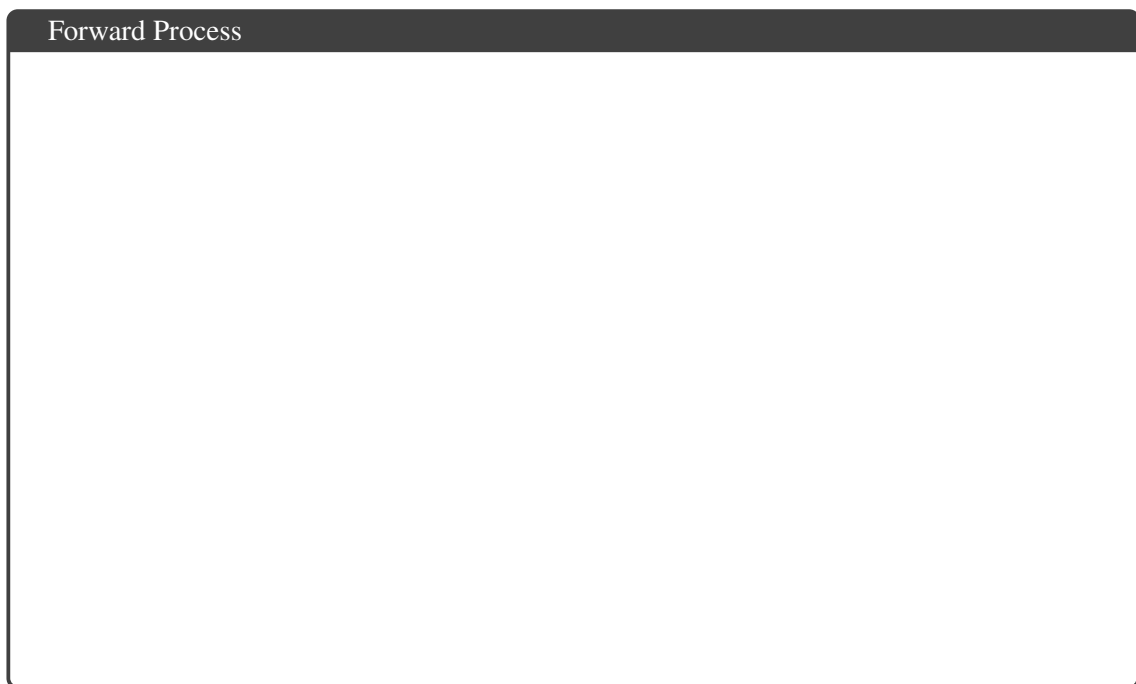


Figure 4: Sample Figure of the Forward Diffusion Process.



- 7.5. (4 points) **Visualization:** Use the trained model after 10,000 steps to visualize the backward diffusion process. Input the noise images generated from the preceding forward process (i.e., the image from the last timestep in the forward process) to the diffusion model. Utilize these images to generate visualizations of the backward diffusion process at key intervals: 0%, 25%, 50%, 75%, and 99% of the total timesteps. The resulting figure should resemble the provided sample, though the images will vary due to inherent randomness.

Hint: you can find this figure on Colab after calling `visualize_diffusion`

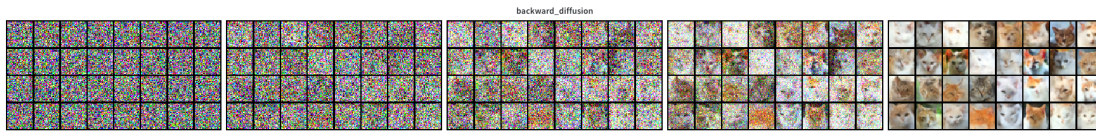
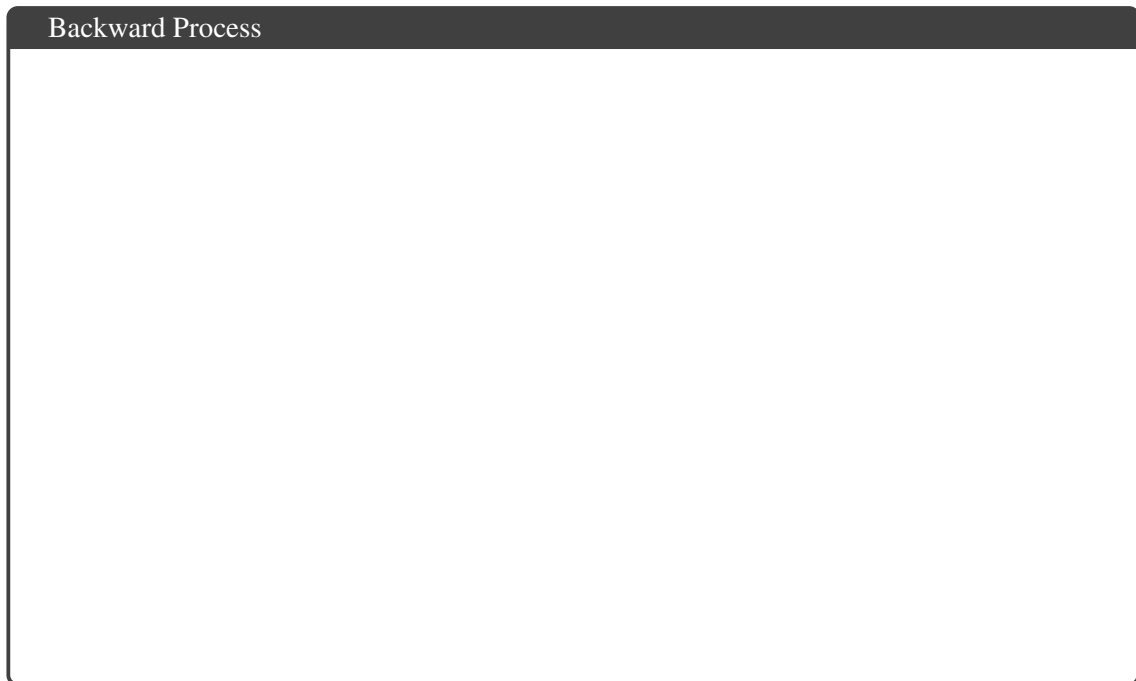


Figure 5: Sample Figure of the Backward Diffusion Process.



8 Code Upload (0 points)

8.1. (0 points) Did you upload your code to the appropriate programming slot on Gradescope?

Hint: The correct answer is ‘yes’.

☐ Yes

☐ No

For this homework, you should upload only `diffusion.py`.

9 Collaboration Questions (2 points)

After you have completed all other components of this assignment, report your answers to these questions regarding the collaboration policy. Details of the policy can be found in the syllabus.

- 9.1. (1 point) Did you collaborate with anyone on this assignment? If so, list their name or Andrew ID and which problems you worked together on.

- 9.2. (1 point) Did you find or come across code that implements any part of this assignment? If so, include full details.