HOMEWORK 2
GENERATIVE MODELS OF IMAGES [*]

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

OUT: Sep. 23, 2024
DUE: Oct. 07, 2024
TAs: Shreeya, Aravind, Jacob, Katie

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 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.

Question	Points
Convolutional Neural Networks	6
Encoder-only Transformers	8
Generative Adversarial Network (GAN)	5
Understanding Diffusion Models	14
Programming: Diffusion Models	21
Code Upload	0
Collaboration Questions	2
Total:	56

[*]Compiled on Tuesday 24th September, 2024 at 16:50

1 Convolutional Neural Networks (6 points)

1.1. 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, $\mathbf{y} \in \mathbb{R}^{1 \times N_h \times N_w}$.

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

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

$$\forall h \in \{1, \ldots, N_h\}, w \in \{1, \ldots, N_w\} \tag{3}$$

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

1.1.a. (3 points) **Short answer:** The valid indices for $x_{c,m,n}$ are $c \in \{1, \ldots, C\}$, $m \in \{1, \ldots, N_h\}$, $n \in \{1, \ldots, N_w\}$. 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 need on the top p_t and the bottom p_b? Report your answer by defining p_l, p_r, p_t, p_b.

1.1.b. (3 points) **Short answer:** Now suppose we create a new input 3D tensor $\mathbf{x}' \in \mathbb{R}^{C \times (N_h + p_b + p_t) \times (N_w + p_l + p_r)}$ 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 Encoder-only Transformers (8 points)

2.1. (2 points) **Drawing:** Suppose we feed a sentence $w_1, \ldots, w_N$ of length N into a *decoder-only* Transformer model (aka. Transformer LM), which defines a distribution $p(w_1, \ldots, w_N)$. 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.

2.2. Suppose we feed a sentence $w_1, \ldots, w_N$ 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:

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

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 $1, \ldots, V$.

2.2.a. (3 points) Use the score vectors $\mathbf{s}_t$ to define a conditional probability distribution over a sequence of part-of-speech tags $t_1, \ldots, t_N$ given the words:

$$p(t_1, \ldots, t_N \mid w_1, \ldots, w_N) = \cdots$$

The distribution you define must be *globally* normalized, not locally normalized.

2.2.b. (3 points) **Drawing:** Draw the corresponding factor graph for this probability distribution. Your drawing must include exactly $2N$ nodes, one node for each word w_n in the sentence and one for each tag t_n.

3 Generative Adversarial Network (GAN) (5 points)

3.1. Suppose we want to define a GAN-inspired model for inpainting grayscale images. Each original image is a matrix $\mathbf{x} \in (0,1)^{N_h \times N_w}$ with scalars 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 $m_{ij} = 1$ indicates that the pixel should be masked out and $m_{ij} = 0$ indicates the pixel should be left intact. Our model is a variant of U-Net that takes in an input image $\mathbf{x}$ and and a mask $\mathbf{m}$ and returns a reconstructed image $\mathbf{x}'$ with the masked pixels filled in by the model, $\mathbf{x}' = g_\theta(\mathbf{x}, \mathbf{m})$. You decide to train your model by using two loss functions in combination.

3.1.a. (1 point) **Short answer:** Define a mathematical expression for $\mathbf{x}'$ that will use the model g_θ and a mask $\mathbf{m}$ (as defined above) to create an in-filled image, ensuring that the masked-out pixels are never visible to g_θ.

3.1.b. (1 point) **Short answer:** Next, using the term you wrote in part a define a squared error loss $\ell_{mse}(\theta)$ for one example $(\mathbf{x}, \mathbf{m})$ that, when minimized, encourages the masked out pixels of the original image to match the reconstructed pixels output by the model. Ensure that pixels outside of the masked area do not affect the loss.

3.1.c. (2 points) **Short answer:** Now suppose we have a discriminator $d_\phi(\mathbf{x})$ that returns the probability that $\mathbf{x}$ is a real image and not from the generator g_θ. Using your answer from 3.1a, give a GAN-style objective function that to train the generator so that it is effective at inpainting, and the discriminator so that it is effective at distinguishing real from inpainted images.

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

4 Understanding Diffusion Models (14 points)

Introduction

Diffusion models are revolutionizing how we generate data, achieving impressive results in text-conditioned image generation. In this section, we will study the variational interpretation of diffusion models.

Given observed samples $p(\mathbf{x})$ from a distribution of interest, the goal of a generative model is to learn to model its true data distribution. Once learned, we can generate new samples from our approximate model at will. In many situations, we can imagine the data $\mathbf{x}$ we see as coming from a latent representation $\boldsymbol{z}$ 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 $p(\mathbf{x}, \boldsymbol{z})$. Directly computing and maximizing the likelihood $p(\mathbf{x})$ is difficult. Instead, we maximize a lower bound:

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

$q_\phi(\boldsymbol{z} \mid \mathbf{x})$ is called 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:

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

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

The encoder or forward process of a diffusion model is:

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

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:

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

ELBO surgery

4.1. (5 points) Prove that we can break down Equation (7) as:

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

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

4.2. (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 this term maximized?

Image Diffusion

4.3. (2 points) Assume the encoder of the diffusion model at step t is given by:

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

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.

4.4. (3 points) **Reparameterization trick.** The reparameterization trick is a powerful mathematical tool that allows us to generate samples of any Gaussian distribution $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \sigma^2 \boldsymbol{I})$ by sampling the standard normal normal distribution using the following transformation:

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

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

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

4.5. (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.

5 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 4. 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 high-quality images at 512×512 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
    main.py
    requirements.txt
    run_in_cloud.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. (Hint: This is the **only** file you need to modify. Locations in the code where changes ought to be made are marked with a **TODO**.)
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_cloud.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 simply the process of training or evaluating your diffusion model.

Parameters

In table 1,2 and 3, 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_cloud.py`.

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

Table 1: Useful parameters for `run_in_cloud.py`

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

Table 2: Additional parameters for `run_in_cloud.py`. You likely won't need to change these

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

Table 3: Additional parameters for `run_in_cloud.ipynb`. These won't need to be changed from default values for this homework.

Google Colab

Colab provides a free T4 GPU for code execution, albeit with a time limitation that may result in slower training. In the event of GPU depletion on Colab, options include waiting for GPU recovery, switching Google accounts, purchasing additional GPU resources, switching to Kaggle, or switching to a cloud provider (such as GCP or AWS).

To download the AFHQ dataset on Colab, run the commands below. Ensure to prepend "!" before the commands below when working on Colab. If you mount your drive, you should only need to run this once. See the `run_in_cloud.ipynb` file for more details

```
mkdir -p ./data
wget -N https://cmu.box.com/shared/static/5
   iydd9zn3ednl27rcq0qud2scq7839ym -O ./data/afhq_v2.zip
unzip -q ./data/afhq_v2.zip -d ./data
```

Kaggle

Kaggle provides 30 hours of free T4 or V100 GPU runtime per week. This is more than sufficient to complete this homework. See the HW2 recitation for more details on how to set up this homework to be run on Kaggle.

Diffusion

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

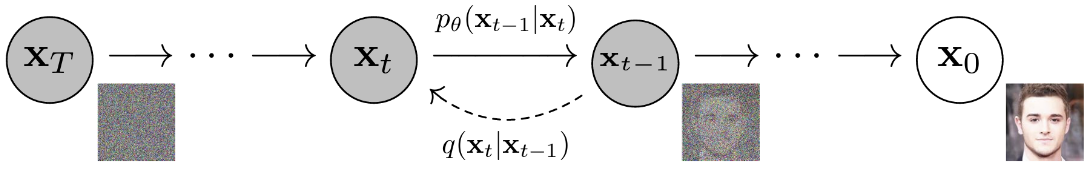
$\mathbf{x}_T$
$\mathbf{x}_t$
$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$
$\mathbf{x}_{t-1}$
$q(\mathbf{x}_t|\mathbf{x}_{t-1})$
$\mathbf{x}_0$

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

Forward Process (Noise ← Image): In this problem, $\mathbf{x}_0 \sim q(\mathbf{x})$ corresponds to the pixels of the image. As we saw in Problem 4, 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:

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

where $\mathbf{x}_t$ is the image after t diffusion steps, $\boldsymbol{I}$ is the identity matrix. The step sizes are controlled by a variance schedule $\{\alpha_t \in (0,1)\}_{t=1}^T$ 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 4.4, we used the reparameterization trick to sample $\mathbf{x}_t$ directly from $\mathbf{x}_0$:

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

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

$$\begin{aligned} \alpha_t &= \text{clip}\left(\frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}, 0.001, 1\right), \bar{\alpha}_t = \frac{f(t)}{f(0)}, \\ \text{where } f(t) &= \cos\left(\frac{t/T + s}{1+s} \cdot \frac{\pi}{2}\right)^2, \end{aligned} \tag{14}$$

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:

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

(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 $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ 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 $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ decomposes as

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

Conveniently, we have already derived the three terms of Equation 16. In particular, $q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$ is given by Equation 9. From Problem 4.4, we also know that:

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

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

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

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 equilavent to training a neural network $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ that learns to predict the source noise $\boldsymbol{\epsilon}$ that determines $\mathbf{x}_t$ from $\mathbf{x}_0$. This can be understood by rearranging the terms in Equation 13:

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

Reverse Processs Model: The reverse process model $\boldsymbol{\epsilon}_\theta$ 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.

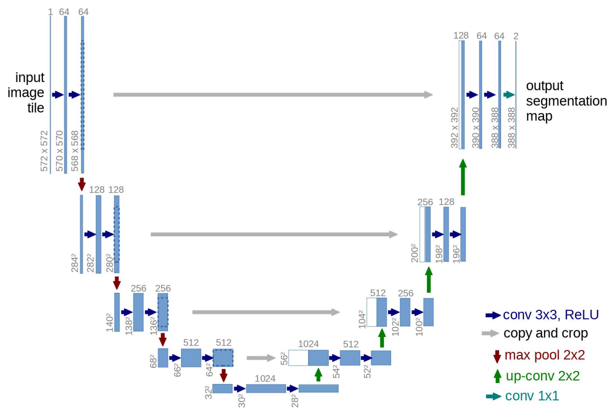

Figure 2: The structure of U-Net.

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 `forward`, `q_sample` and `p_loss` within the `Diffusion` class. The `p_loss` 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:     $t \sim \text{Uniform}(\{1, ..., T\})$
4:     $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
5:     $\mathbf{x}_t \leftarrow \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}$        ▷ forward diffusion process
6:     Take optimizer step on $L_1$ loss, $\nabla_\theta \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|_1$
7: until converged
```

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

Algorithm 2 Sampling

```
1: $\mathbf{x}_T \sim \mathcal{N}(0, I)$
2: for $t = T, ..., 1$ do
3:     $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ if $t > 1$, else $\mathbf{z} = 0$
4:     $\boldsymbol{\epsilon}_t \leftarrow \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$        ▷ predicted noise
5:     $\hat{\mathbf{x}}_0 \leftarrow \frac{1}{\sqrt{\bar{\alpha}_t}}\left(\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}_t\right)$        ▷ estimated $\hat{\mathbf{x}}_0$
6:     $\hat{\mathbf{x}}_0 \leftarrow \textit{clamp}(\hat{\mathbf{x}}_0, -1, 1)$        ▷ rectify $\hat{\mathbf{x}}_0$
7:     $\tilde{\boldsymbol{\mu}}_t \leftarrow \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)}{1-\bar{\alpha}_t}\hat{\mathbf{x}}_0$        ▷ 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.

5.1. (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]

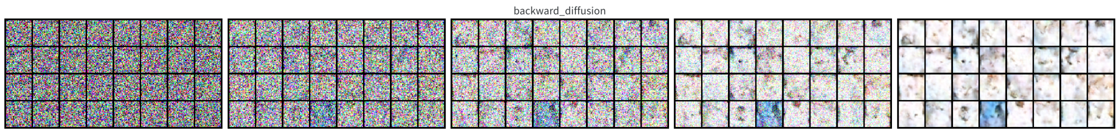

Figure 3: The Backward Diffusion Process, after just 1000 steps.

Training Loss

5.2. (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]

FID

5.3. (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]

Backward Process

5.4. (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`

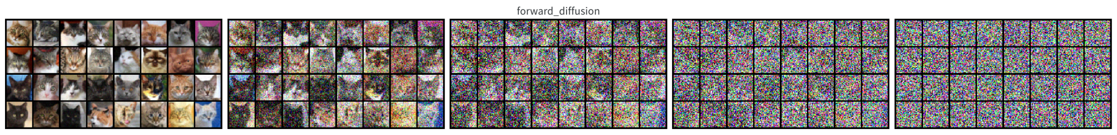

Figure 4: Sample Figure of the Forward Diffusion Process.

Forward Process

5.5. (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`

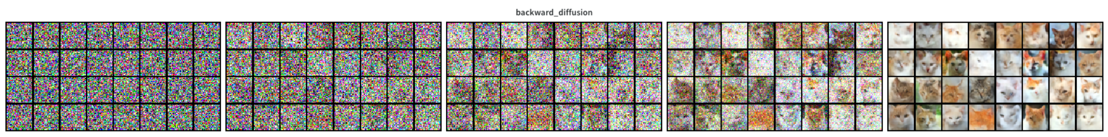

Figure 5: Sample Figure of the Backward Diffusion Process.

Backward Process

6 Code Upload (0 points)

6.1. (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 all the code files that contain your new and/or changed code. Files of type `.py` and `.ipynb` are both fine.

7 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.

7.1. (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.

7.2. (1 point) Did you find or come across code that implements any part of this assignment? If so, include full details.