Algorithm 1 - Training, and Algorithm 2 - Sampling in the DDPM paper use 1 indexing for timesteps. However, you should use 0 indexing (index start from 0).

Part 2.1 - Noise Scheduler [6 points]

2.1.1 - Forward Process (Adding Noise) [3 points] [.5 writeup points]

During the forward process, we take an input and gradually add Gaussian noise to the data according to a variance schedule. This is described in section 2, background, of the DDPM paper. You must read that section to complete this part of the assignment.

TODO: Complete initialization function in `DiffusionModels/noise_scheduler.py`

TODO: Implement `add_noise` function in `part2-DiffusionModel/noise_scheduler.py`

Hint: Pay attention to equation 4

Question: what does $q(x_t|x_o)$ represent for a diffusion model? (read the paper)

Answer: the (Gaussian) noise generation process for any t conditioned on x_0. That is the distribution used for the noising (forward) process.

In [ ]:
```
# test initialization
unit_tester.test_noise_scheduler_init()
# test add noise
unit_tester.test_add_noise()
```

```
NoiseScheduler() initialization test case passed!
add_noise() test case passed!
```

2.1.2 - Reverse Process (Removing Noise) [2.5 point]

When taking a step during the backwards process, we have access to a partially noised sample x_t, the denoising timestep t, and our model's noise prediction. We remove the predicted noise to get a slightly less noisy sample x_{t-1}. The completely uncorrupted sample is recovered at x_0. This process is described in detail in section 3 of the DDPM paper, please read this section before completing this part of the assignment.

TODO: Implement `denoise_step` function in `DiffusionModels/noise_scheduler.py` (do not implement the thresholding part yet)

Hint: Pay attention to algorithm 2 - Sampling

In [ ]:
```
# test denoise step
unit_tester.test_denoise_step()
```

```
denoise_step() test case passed!
```

Part 2.2 - Constructing the Diffusion Model [7

generation. In practice, we train the unconditional noise prediction net at the same time as the conditional noise prediction net. Section 3.2 of the CFG paper describes this process.

We expect you to use 0s as the null tokens.

TODO: Implement classifier free guidance section of `compute_loss_on_batch` function in `DiffusionModels/diffusion_model.py`

Hint: This should only be a few lines, use self.p_uncond

In [ ]:
```
# test classifier free guidance training
unit_tester.test_compute_loss_with_cfg()
```

```
compute_loss_with_cfg() test case passed!
```

2.2.3 Sample Generation (with classifier free guidance) [3.5 points]

Before we can actually use our diffusion model, we need to write the code for generating a sample. To generate an output, we draw a sample x_T from random gaussian noise. We then itteratively denoise the random output for T denoising timesteps until we get our uncorrupted output x_0. You will be implementing sampling with classifier-free guidance. Do not worry about replicating algorithm 2 of the CFG paper, we only expect you to calculate the updated noise prediction from equation 6 of the CFG paper:

$$\bar{\epsilon_\theta}(x_t, t, c) = (1 + w) * \epsilon_\theta(x_t, t, c) - w * \epsilon_\theta(x_t, t, \text{empty})$$

where ϵ_θ is the predicted noise from your noise prediction network and w is the guidance weight

TODO: Implement `generate_sample` function in `DiffusionModels/diffusion_model.py`

Hint: Use your noise scheduler and refer to Algorithm 2 - Sampling of the DDPM paper
Hint: Dont forget that this is a conditional model and to include classifier free guidance

In [ ]:
```
# test generate sample
unit_tester.test_generate_sample()
```

```
generate_sample_step() test case passed!
generate_sample_step() with guidance test case passed!
```

In [ ]:
```
## Trying to test threshold
unit_tester.test_threshold_denoise_step()
```

```
threshold_denoise_step() test case passed!
```

Part 2.3 - Diffusion Model Applications (Computer Vision) [8 points]

Diffusion models are mostly known in popular culture because of image generation

In parts 2.1 and 2.2 you implemented diffusion model training and sample generation, so the majority of the work for generating images is done! The computer_vision/image_gen.py file handles the rest of the logic for image generation. Your first task is to complete the initialization function, this is only a few lines of code.

TODO: Complete initialization function in `DiffusionModels/computer_vision/image_gen.py`

In [ ]:
```
generator = ImageGenerator()
```

After initializing the image generator, take a look through the rest of the file to understand how the data is loaded and how the images are generated. Your next task is to perform data augmentation on the CIFAR dataset (for the sake of training time we do not require you to actually add augmenations and increase the size of your dataset, but go through this exersize with us anyway). This is a common technique in the computer vision world as it allows us to increase the size of our dataset and train the model to be invariant to certain perturbations. You may have done this before when training a classification model, but now we are training a generator. The types of augmentations that we want our generator to be invariant to are not necessariliy the same as for a classifier.

Question: *Why might we want a classifier to be invariant to different augmentations then a generator? Name 2 augmentations that we might perform on a classification dataset that we don't want to perform on a dataset for a generative model. Name 2 augmentations that you want to perform on a dataset for a generative model (its okay if you want also want to do the same augmentations on a classification dataset).*

Your Answer Here: We want classifiers need to be robust to minor variations (out-of-distribution data, but somewhat bounded) and perform the class classifcation correctly; hence, be invariant to those augmentations. On the other hand, we want generators to be more "creative" and give us diverse outputs.

- Two augmentations that we might perform on a classification dataset that we don't want to perform on a dataset for a generative model:
  - Addition of blurr only in the background, not the objects (class specific occlusion)
  - Addition of noise that preserve only the labels (label preserving noise injection)
- Two augmentations that you want to perform on a dataset for a generative model:
  - Addition of blurr (classifiers might want)
  - Addition of flips and rotations (classifiers might want)

OPTIONAL: Implement two augmentations in the `load_dataset` function of `DiffusionModels/computer_vision/image_gen.py` , (these should be the augmentations you described above). Increasing the dataset size will slow down your training, so we do not require you to actually implement these augmentations.

will produce num_samples generations of each class.

In [ ]:
```
image = generator.generate_images(guidance=0, num_samples=8)
plot_image(image)
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
Cell In[16], line 1
----> 1 image = generator.generate_images(guidance=0, num_samples=8)
      2 plot_image(image)

File ~/CS7643/Assignments/hw4_code_student_version/part2_DiffusionModels/computer_vision/image_gen.py:119, in ImageGenerator.generate_images(self, guidance, num_samples, threshold, class_label)
    113     all_labels = [class_label] * num_samples * 3
    115 # The size of this conditioning tensor is different than what you would expect given the function signature of generate_sample
    116 # If you are looking at this before implementing generate_sample, the first dimension in this cond tensor is still the batch size
    117 # The generate_sample function signature reflects the shape of the conditioning tensor for the 1D prediction network (part 2.4)
    118 # If none of this makes sense to you, that is fine, you do not need to worry about any of this
--> 119 cond = torch.tensor(all_labels).to(self.policy.device)
    121 # generate the image
    122 image = self.policy.generate_sample(cond, guidance_weight = guidance, threshold=threshold)

RuntimeError: CUDA error: device-side assert triggered
CUDA kernel errors might be asynchronously reported at some other API call, so the stacktrace below might be incorrect.
For debugging consider passing CUDA_LAUNCH_BLOCKING=1.
Compile with `TORCH_USE_CUDA_DSA` to enable device-side assertions.
```

Question: What do you notice about the quality of unguided image generation? How effectively is the model learning each class? This is very early in the training process, but do you notice any features that the model is learning for each class?

Your Answer Here: Unfortunatelly, I am running into CUDA errors, the most frequent being `RuntimeError: cuDNN error: CUDNN_STATUS_INTERNAL_ERROR`, therefore I can not see the images nor draw conclusions about it. Below are some of the things I have tried, but it is also important to note that I did try many different numbers for `self.condition_dim`, both in the int format and tuple of ints.

I am running locally, in VSCode and already tried restarting the kernel, closing and reopening the program, and even restarting my machine. Unfortunatelly I don't have time to dig any further in the web or talk about this in OH. I made a post on piazza about this, apparently some people had the same issue.

Now, we will evaluate image generation with different guidance weights. To do this, we dont actually need to retrain the model at all, this guidance can be done purely at inference time. Great! Explain why we dont have to retrain for image guidance.

Your Answer Here: Unfortunatelly, I am running into CUDA errors, the most frequent being `RuntimeError: cuDNN error: CUDNN_STATUS_INTERNAL_ERROR`, therefore I can not see the images nor draw conclusions about it. Below are some of the things I have tried, but it is also important to note that I did try many different numbers for `self.condition_dim`, both in the int format and tuple of ints.

I am running locally, in VSCode and already tried restarting the kernel, closing and reopening the program, and even restarting my machine. Unfortunatelly I don't have time to dig any further in the web or talk about this in OH. I made a post on piazza about this, apparently some people had the same issue.

In [ ]:
```
image = generator.generate_images(guidance=0.5, num_samples=8)
plot_image(image)
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
Cell In[18], line 1
----> 1 image = generator.generate_images(guidance=0.5, num_samples=8)
      2 plot_image(image)

File ~/CS7643/Assignments/hw4_code_student_version/part2_DiffusionModels/computer_vision/image_gen.py:119, in ImageGenerator.generate_images(self, guidance, num_samples, threshold, class_label)
    113     all_labels = [class_label] * num_samples * 3
    115 # The size of this conditioning tensor is different than what you would expect given the function signature of generate_sample
    116 # If you are looking at this before implementing generate_sample, the first dimension in this cond tensor is still the batch size
    117 # The generate_sample function signature reflects the shape of the conditioning tensor for the 1D prediction network (part 2.4)
    118 # If none of this makes sense to you, that is fine, you do not need to worry about any of this
--> 119 cond = torch.tensor(all_labels).to(self.policy.device)
    121 # generate the image
    122 image = self.policy.generate_sample(cond, guidance_weight = guidance, threshold=threshold)

RuntimeError: CUDA error: device-side assert triggered
CUDA kernel errors might be asynchronously reported at some other API call, so the stacktrace below might be incorrect.
For debugging consider passing CUDA_LAUNCH_BLOCKING=1.
Compile with `TORCH_USE_CUDA_DSA` to enable device-side assertions.
```

In [ ]:
```
image = generator.generate_images(guidance=1, num_samples=8)
plot_image(image)
```

[-1, 1]? Additionally, why is thresholding particularly useful when dealing with large guidance weights? Will thresholding have an effect even if we don't have a large guidance weight?

Your Answer Here: For the coding answer, you can see that I run this above, and it works (I'll try to link that page in Gradescope).

For high quality, we need to play with the values of the bounds for thresholding, so I see it as hyperparameters: it might have a good or a bad effect depending on the values you choose. If we do constrain it to [-1, 1] though, we might make the noise prediction outside what is within the true distribution that we are trying to learn, and maybe also lose data (whatever is outside that interval) in the process.

Although, when dealing with large guidance weights, thresholding will do some clipping to outputs that are probably not wanted, which is a heuristic to deal with outputs that are non-representative of the distribution we are learning.

Even if we don't have a large guidance weight, threshold can still have an effect of preventing the generation of pixel values that are off (too dark or too bright), and/or outliers of the image distribution ("artifacts").

Disclosure: Unfortunatelly, I am running into CUDA errors, the most frequent being `RuntimeError: cuDNN error: CUDNN_STATUS_INTERNAL_ERROR` , therefore I can not see the images nor draw conclusions about it. Below are some of the things I have tried, but it is also important to note that I did try many different numbers for `self.condition_dim` , both in the int format and tuple of ints.

I am running locally, in VSCode and already tried restarting the kernel, closing and reopening the program, and even restarting my machine. Unfortunatelly I don't have time to dig any further in the web or talk about this in OH. I made a post on piazza about this, apparently some people had the same issue.

The answer above is what makes sense to me + some research that I did.

So we need to compute x_{t-1} such that we estimate x_0 as an intermediate step. This way we can threshold our estimate of x_0.
This is shown in equations (6) and (7) of the DDPM paper. The equation for computing an estimate of x_0 is shown in equation (15). Make sure to threshold the prediction of x_0!

TODO: Implement denoising with thresholding in the `denoise_step` function in `DiffusionModels/noise_scheduler.py`

In [ ]:
```
# test denoise step with thresholding
unit_tester.test_threshold_denoise_step()
```

Now that you have completed the thresholding, evaluate how this changes image generation with different guidance weights.

In [ ]:
```
image = generator.generate_images(guidance=0, num_samples=8, threshold=Tr
plot_image(image)
```

In [ ]:
```
image = generator.generate_images(guidance=0.5, num_samples=8, threshold=
plot_image(image)
```

In [ ]:
```
image = generator.generate_images(guidance=1, num_samples=8, threshold=Tr
plot_image(image)
```

In [ ]:
```
image = generator.generate_images(guidance=2, num_samples=8, threshold=Tr
plot_image(image)
```

In [ ]:
```
image = generator.generate_images(guidance=5, num_samples=8, threshold=Tr
plot_image(image)
```

Question: How do the results after thresholding compare to the results before thresholding? With thresholding and high guidance weight, which class specific features are most prominant? Which set of parameters gave you the best output?

Your Answer Here: I guess with thresholding it would perform better.

Unfortunatelly, I am running into CUDA errors, the most frequent being `RuntimeError: cuDNN error: CUDNN_STATUS_INTERNAL_ERROR`, therefore I can not see the images nor draw conclusions about it. Below are some of the things I have tried, but it is also important to note that I did try many different numbers for `self.condition_dim`, both in the int format and tuple of ints.

I am running locally, in VSCode and already tried restarting the kernel, closing and reopening the program, and even restarting my machine. Unfortunatelly I don't have time to dig any further in the web or talk about this in OH. I made a post on piazza about this, apparently some people had the same issue.

TODO: Experiment with different training times and guidance weights! Try and get the best image possible (or show something else interesting). You need at least two more generations, feel free to do more are welcome though.

In [ ]:
```
# room for experimentation
```

Collect Submission

Run the following cell to collect `assignment3_part2_submission.zip`. You will submit this to HW3 Code - Part 2 on gradescope. Make sure to also export a PDF of this jupyter notebook and attach that to the end of your theory section. This PDF must show your answers to all the questions in the document, please leave in all the photos that you generate as well. You will not be given credit for anything that is not visible to us in this PDF.

In [ ]:
```
%%bash collect_submission.sh
```

Contributers