CS180 Project 5 -- Ryan Nader -- Part A + B

github: https://github.com/berkeleybear22ryan/CS180_Project5

website: https://berkeleybear22ryan.github.io/CS180_Project5/

for `PART A` ...

IMPORTANT: for all of `PART A` I used the `seed=180`

this is all from the file `180_proj5A_starter.ipynb` and just following the instructions on this

Part 0: Setup -- done

for this part there are 3 prompts that were used for the corresponding images each with varying amounts of `num_inference_steps`

Here we use the DeepFloyd IF diffusion model. DeepFloyd is a two stage model trained by Stability AI. The first stage produces images of size 64×64 and the second stage takes the outputs of the first stage and generates images of size 256×256

prompts are ...

1. `an oil painting of a snowy mountain village`
2. `a man wearing a hat`
3. `a rocket ship`

and each has `num_inference_steps=10` or `num_inference_steps=50` or `num_inference_steps=200`

here are the ones associated with `an oil painting of a snowy mountain village`

`num_inference_steps=10`

`num_inference_steps=50`

`num_inference_steps=200`

here are the ones associated with `a man wearing a hat`

`num_inference_steps=10`

`num_inference_steps=50`

`num_inference_steps=200`

here are the ones associated with `a rocket ship`

`num_inference_steps=10`

`num_inference_steps=50`

`num_inference_steps=200`

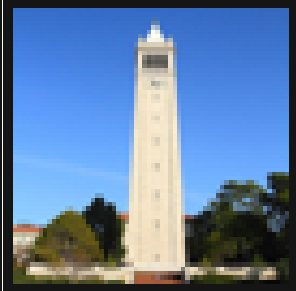
Answer to briefly reflect on the quality of the outputs and their relationships to the text prompts.

Increasing `num_inference_steps` tends to significantly improves the image quality, detail, and alignment with the text prompt. At lower steps, the images are vague and underdeveloped, while higher steps lead to more refined and accurate outputs.

Part 1: Sampling Loops

Part 1.1: Implementing the forward process -- done

so we are working with the image ...



now we use the formula: $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\,\epsilon$ where $\epsilon \sim N(0, 1)$ to get the noised image x_t

here they want deliverables ...

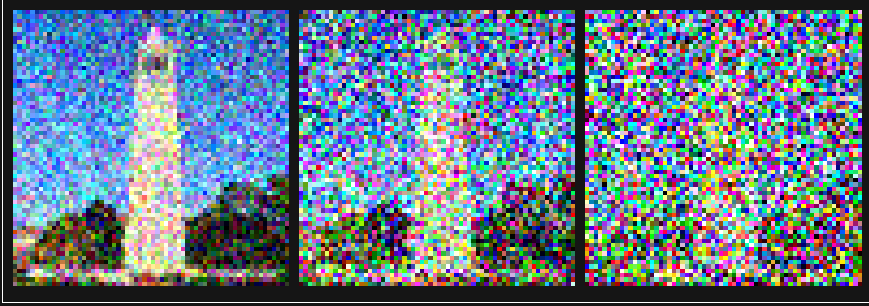
Implement the im_noisy = forward(im, t) function

Show the test image at noise level [250, 500, 750]

I implemented the function and include the results below

We have a test image (64x64x3 image of the Campanile) and apply noise to it (lower timestep is closer to original image and higher timestep is closer to pure noise)

`t=250`, `t=500`, `t=750`



Part 1.2: Classical Denoising -- done

so we are working with the image ...

here they want deliverables ...

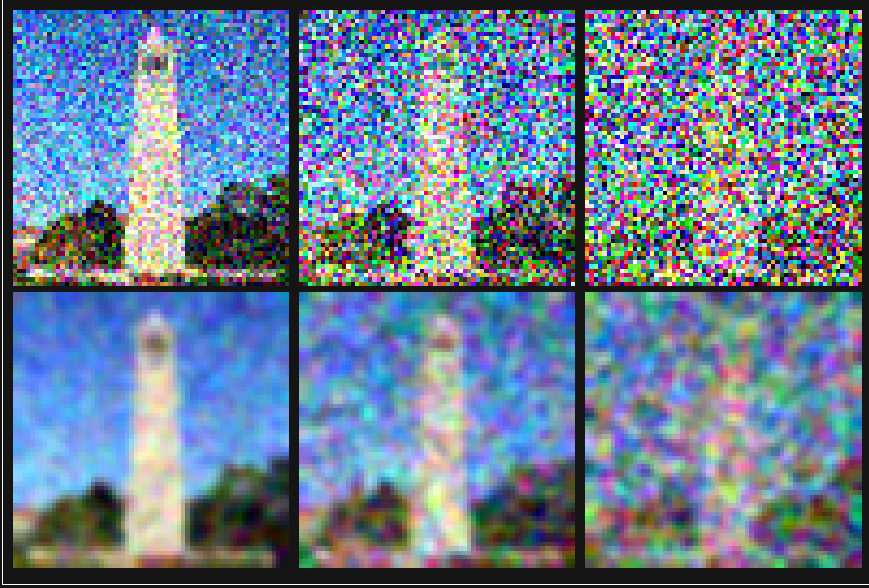
For each of the 3 noisy test images from the previous part, show your best Gaussian-denoised version side by side.

results below ...

here we use Gaussian blurring, to eliminate the noise from the image and we get substandard results:

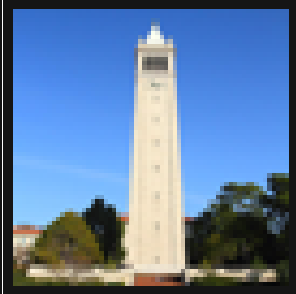
We have a test image (64x64x3 image of the Campanile) and apply noise to it (lower timestep is closer to original image and higher timestep is closer to pure noise) along with the classic denoise attempt for each one below

`t=250` , `t=500` , `t=750`



Part 1.3: Implementing One Step Denoising -- done

so we are working with the image ...



here they want deliverables ...

For the 3 noisy images from 1.2 (t = [250, 500, 750]):

Using the UNet, denoise the image by estimating the noise.

Estimate the noise in the new noisy image, by passing it through stage_1.unet

Remove the noise from the noisy image to obtain an estimate of the original image.

Visualize the original image, the noisy image, and the estimate of the original image

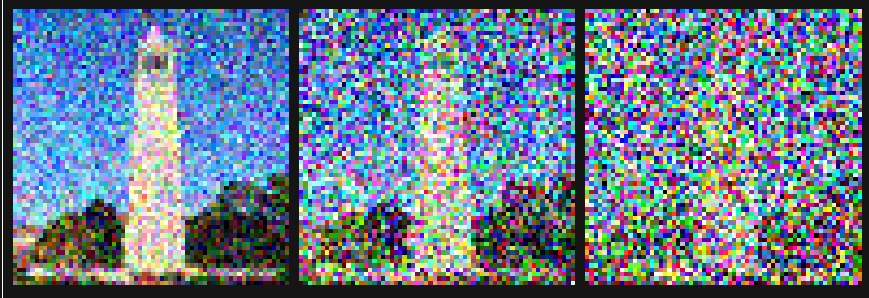
we can write the equation for x_0 as $\hat{x_0} = \frac{x_t - \sqrt{1-\bar{\alpha}_t}\epsilon}{\sqrt{\bar{\alpha}_t}}$

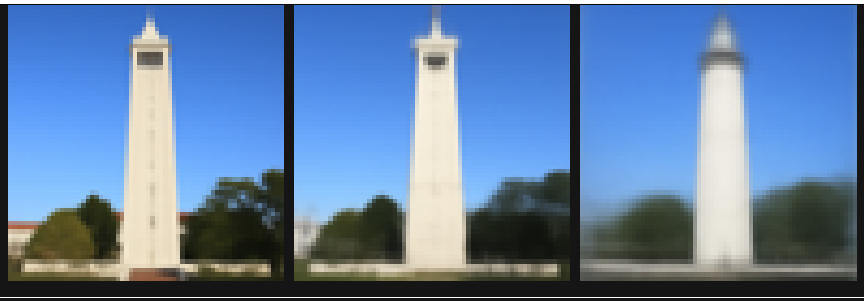
results below ...

here we use Gaussian blurring, to eliminate the noise from the image and we get substandard results:

We have a test image (64x64x3 image of the Campanile) and apply noise to it (lower timestep is closer to original image and higher timestep is closer to pure noise) along with the one step denoise attempt for each one below

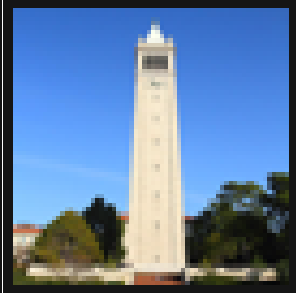
`t=250` , `t=500` , `t=750`





Part 1.4: Implementing Iterative Denoising -- done

so we are working with the image ...



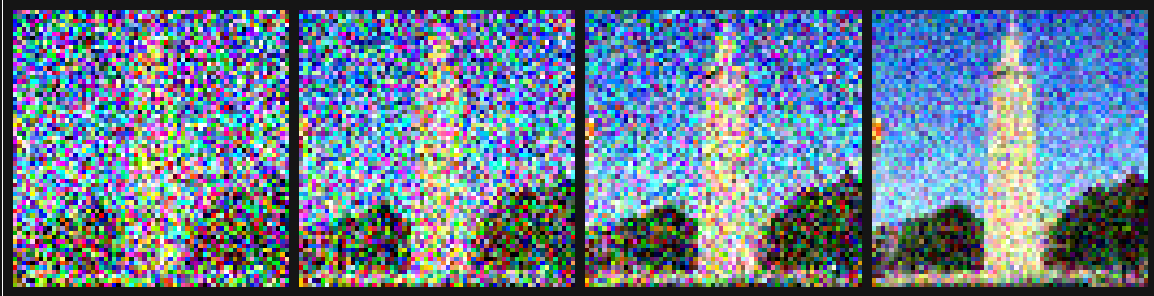
here we are iteratively denoise towards x_0 rather than just doing it in one step.

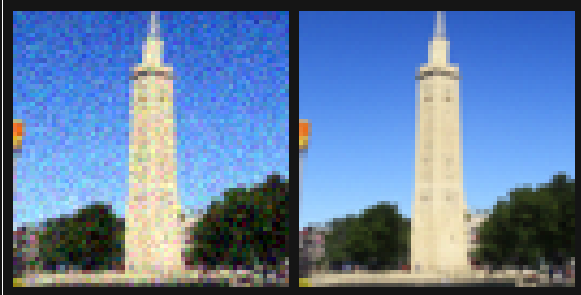
We start with a noisy image and then use the following formula to get the image at the previous timestep which should be less noisy each time.

thus we have the formula ... $x_{t'} = \frac{\sqrt{\bar{\alpha}_t}\beta_t}{1-\bar{\alpha}_t}x_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t'})}{1-\bar{\alpha}_t}x_t + v_\sigma$

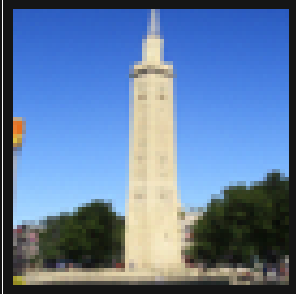
now the iterative results look like this ...

this meets the requirement of Show the noisy image every 5th loop of denoising (it should gradually become less noisy)





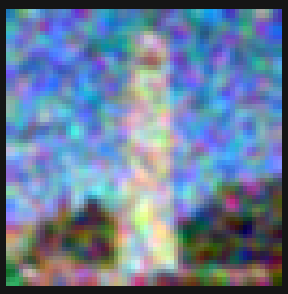
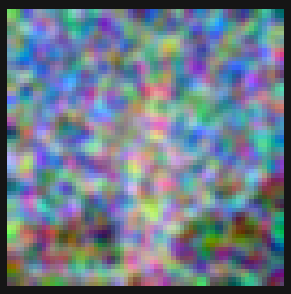
and now will meet the requirement to show the final predicted clean image, using iterative denoising



and now will ... Show the predicted clean image using only a single denoising step, as was done in the previous part. This should look much worse.



and now will show the predicted clean image using gaussian blurring, as was done in part 1.2.

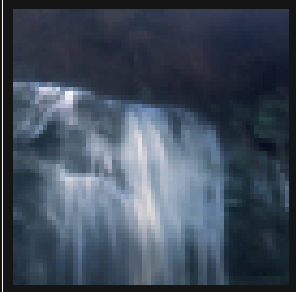
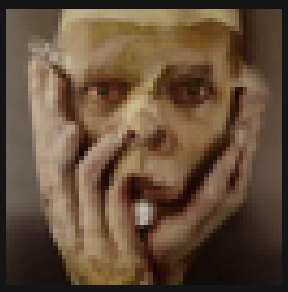




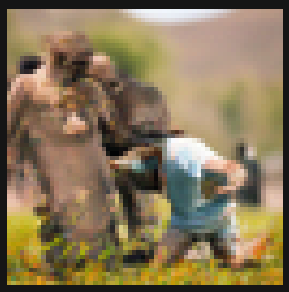
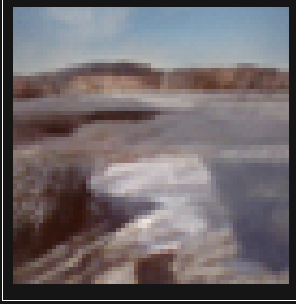
and as can be seen fron this the iterative denoising one is the best so far

Part 1.5: Diffusion Model Sampling -- done

Here the deliverables are to show 5 sampled images

this is iterative denoising from pure, random noise to generate images based on prompt `a high quality photo`

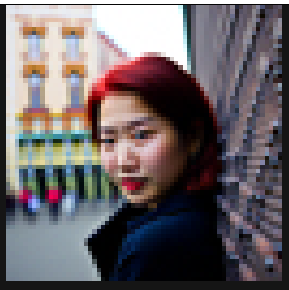
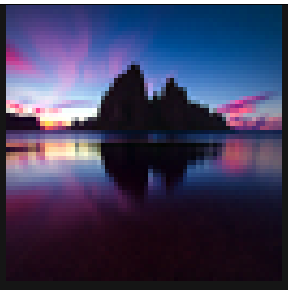
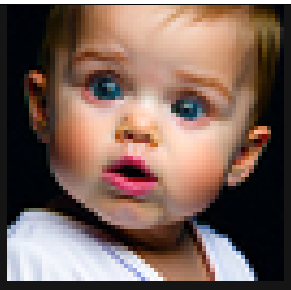
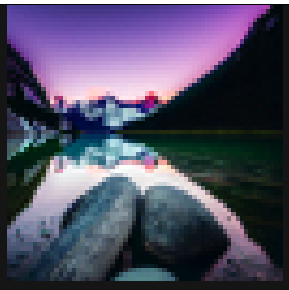
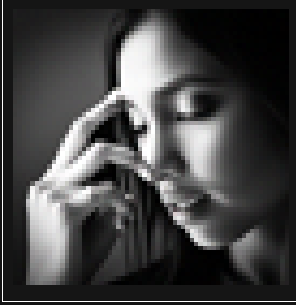
Part 1.6: Classifier Free Guidance -- done

now for this part they want ...

Implement the `iterative_denoise_cfg` function

Show 5 images of `a high quality photo` with a CFG scale of $\gamma = 7$

and we now have new noise estimate of $\epsilon = \epsilon_u + \gamma(\epsilon_c - \epsilon_u)$

Part 1.7: Image-to-image Translation -- done

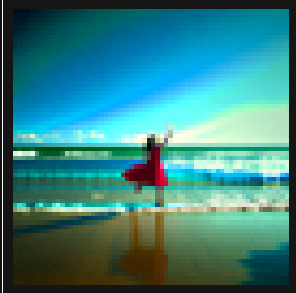
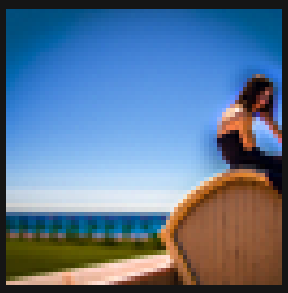
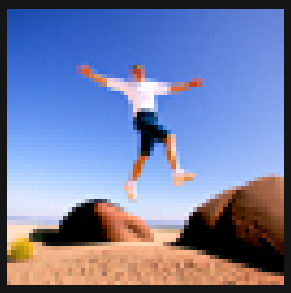
Here they want Edits of the test image, using the given prompt at noise levels [1, 3, 5, 7, 10, 20] with text prompt "a high quality photo"

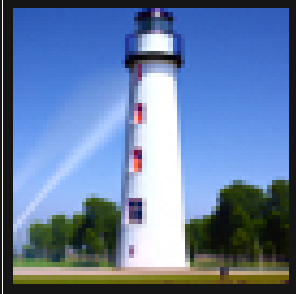
and ... Edits of 2 of your own test images, using the same procedure.

so here we have the images ...

test image

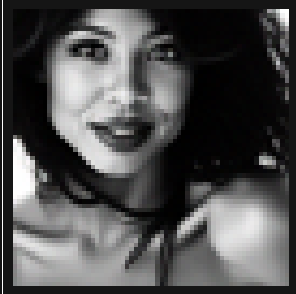
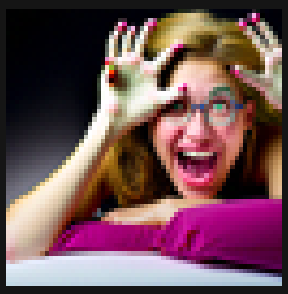
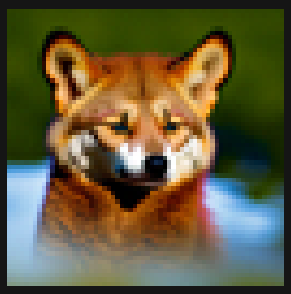
then the 6 images at 1,3,5,7,10,20 are ...

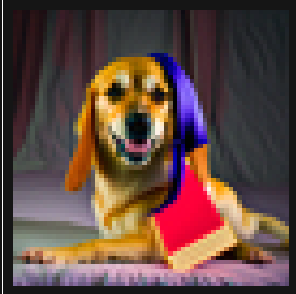







custom image 1

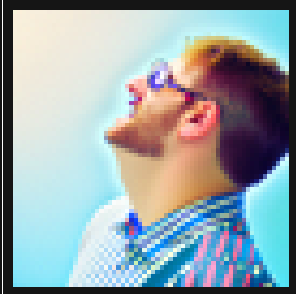
then the 6 images at 1,3,5,7,10,20 are ...

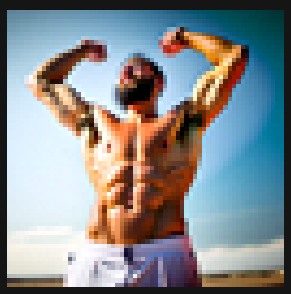




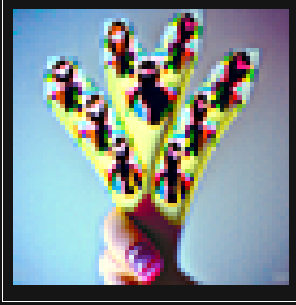
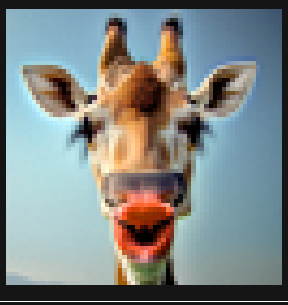



custom image 2

then the 6 images at 1,3,5,7,10,20 are ...





Part 1.7.1: Editing Hand-Drawn and Web Images -- done

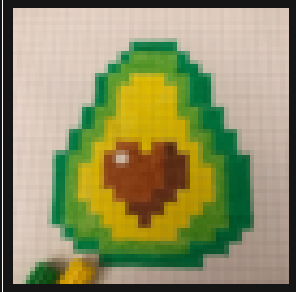
so for this part they want ...

1 image from the web of your choice, edited using the above method for noise levels [1, 3, 5, 7, 10, 20] (and whatever additional noise levels you want)

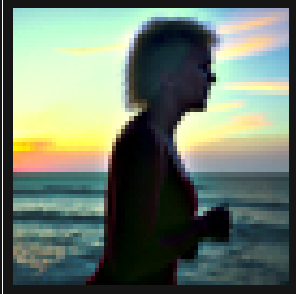
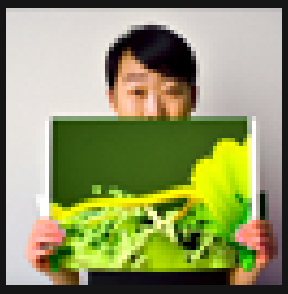
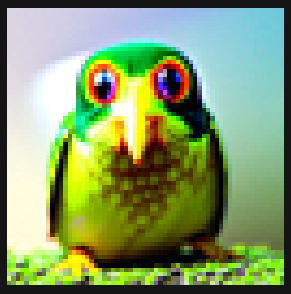
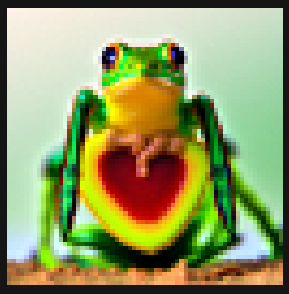
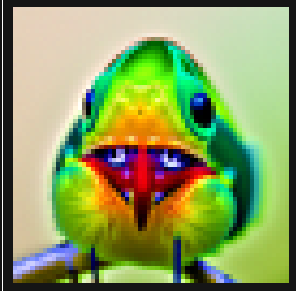
2 hand drawn images, edited using the above method for noise levels [1, 3, 5, 7, 10, 20] (and whatever additional noise levels you want)

I will provide them in the order original, then below that will be side by side 1,3,5,7,10,20

now the reference image they give is as follows ...

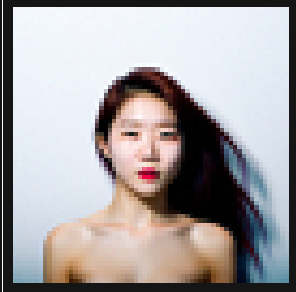
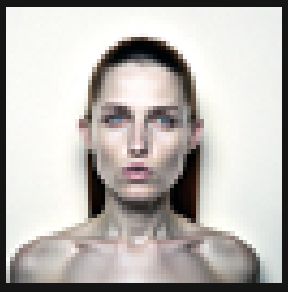
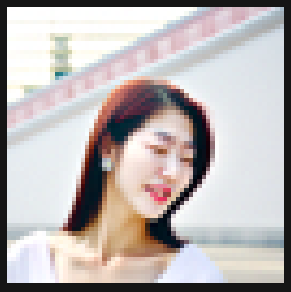
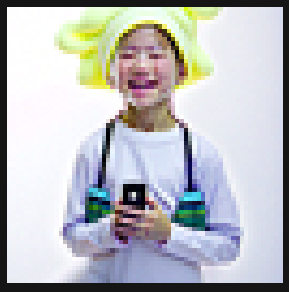


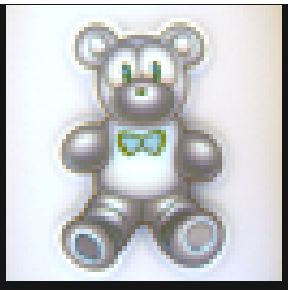
and the generated images are ...

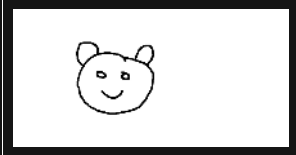

a web drawn image is as follows ...

and the generated images are ...

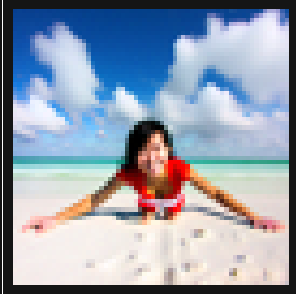





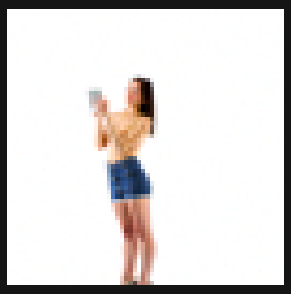
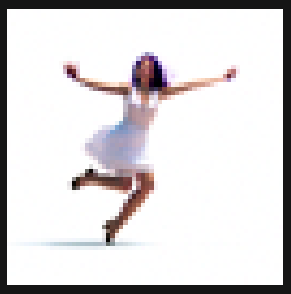
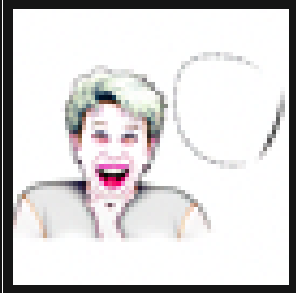
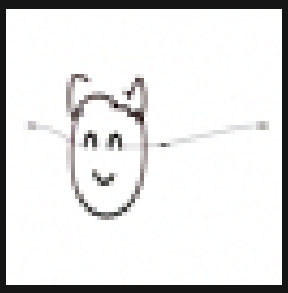



first hand drawn image using drawing board is as follows ...

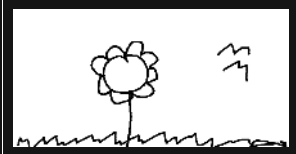


and the generated images are ...

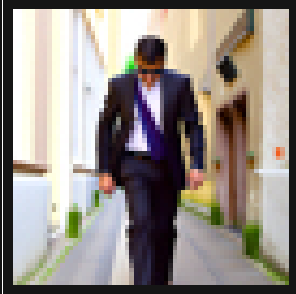


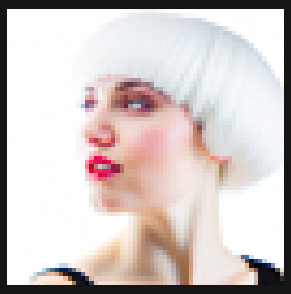





second hand drawn image using drawing board is as follows ...

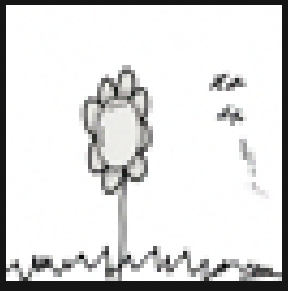


and the generated images are ...



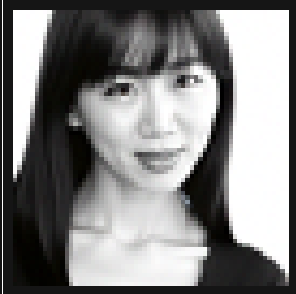
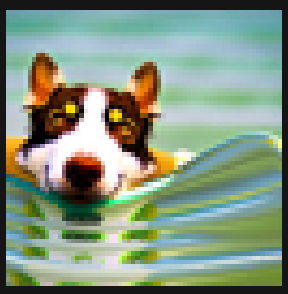


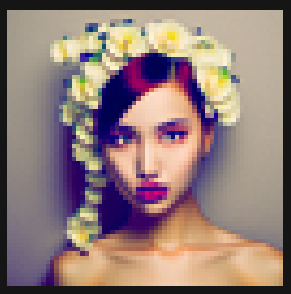


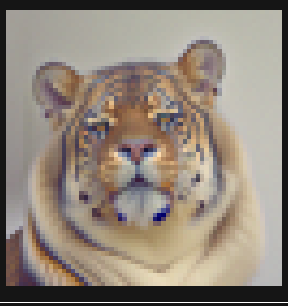


a drawing that I drew on paper is as follows

and the generated images are ...





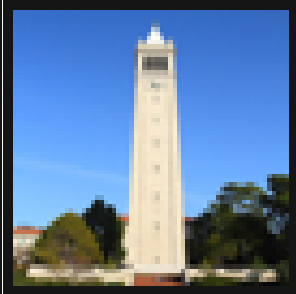
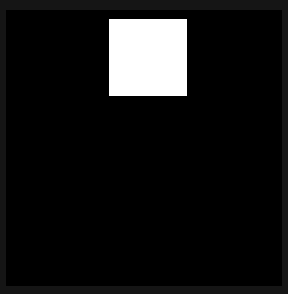
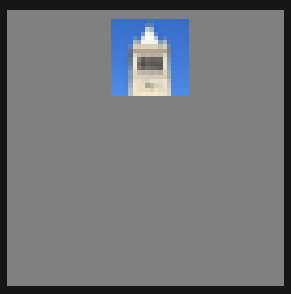
Part 1.7.2: Inpainting -- done

so for this part they want ...

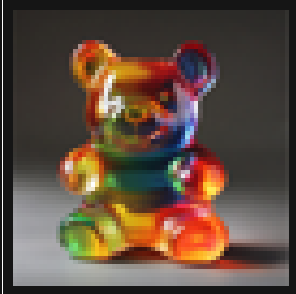
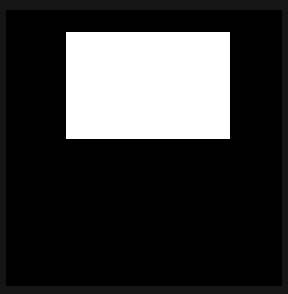
A properly implemented inpaint function

I will give them in the order original image, mask, to replace, inpainted

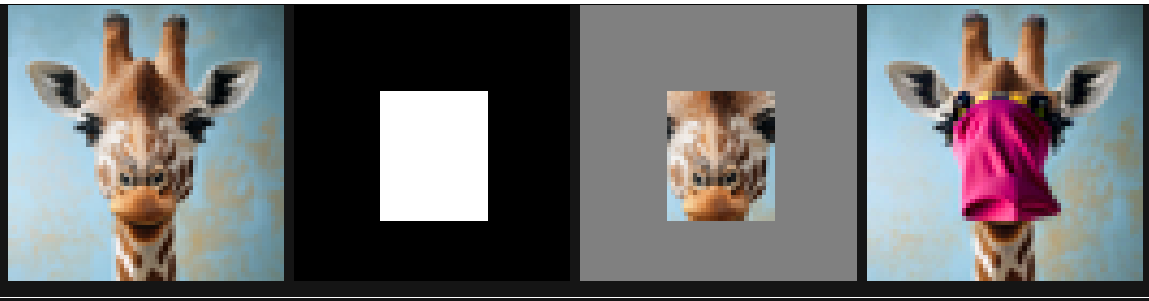
The test image inpainted (feel free to use your own mask)


1/2 of your own images edited (with different mask)

2/2 of your own images edited (with different mask)



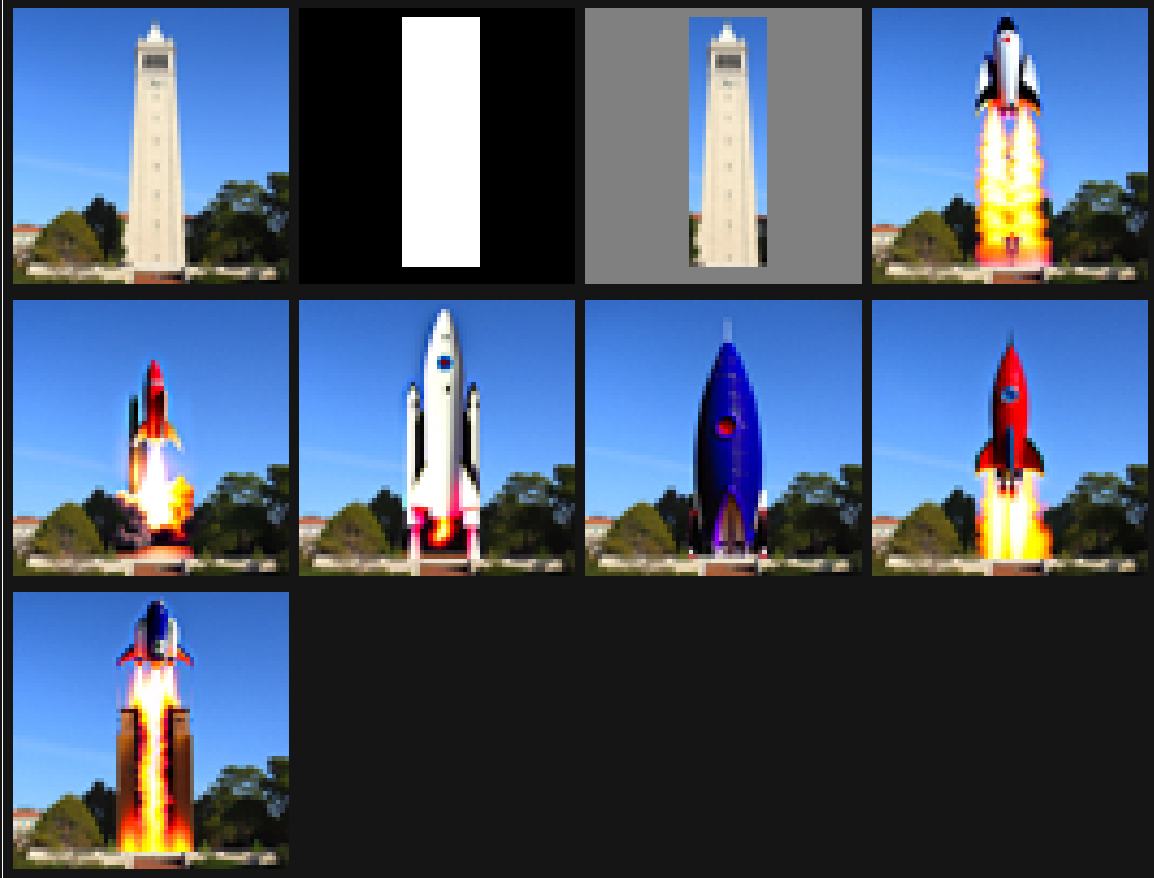
Part 1.7.3: Text-Conditioned Image-to-image Translation -- done

so for this part they want ...

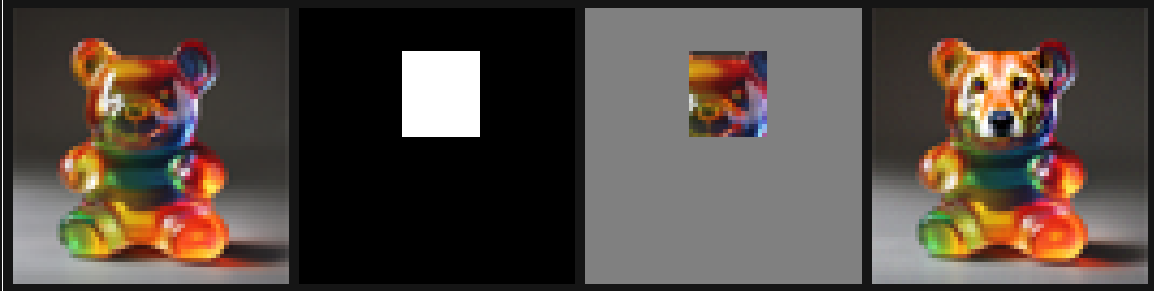
all of them will be in the order image, mask, to_replace, 1, 3, 5, 7, 10, 20

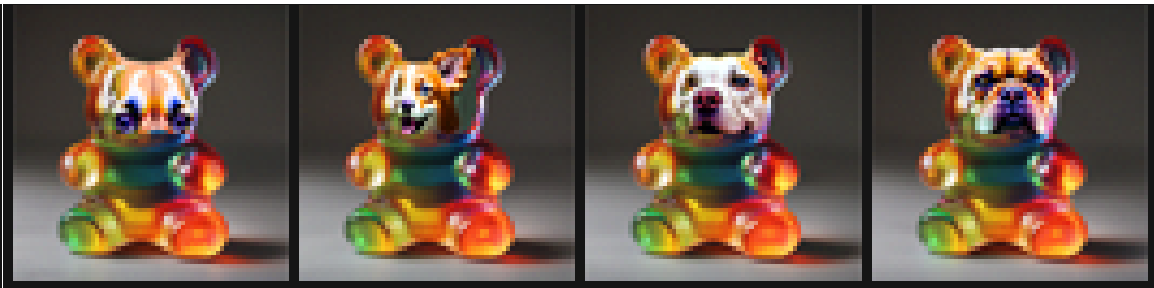
Edits of the test image, using the given prompt at noise levels [1, 3, 5, 7, 10, 20]

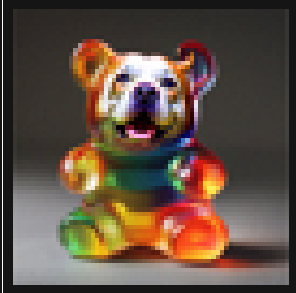
here the is prompt `a rocket ship`



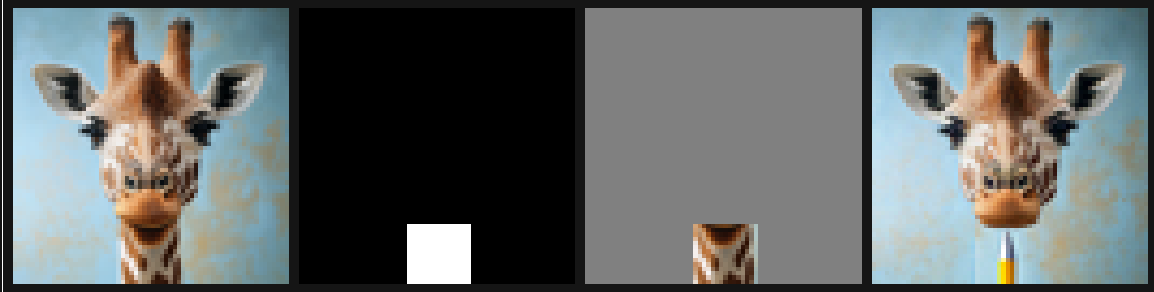
here the is prompt on another image `a photo of a dog`

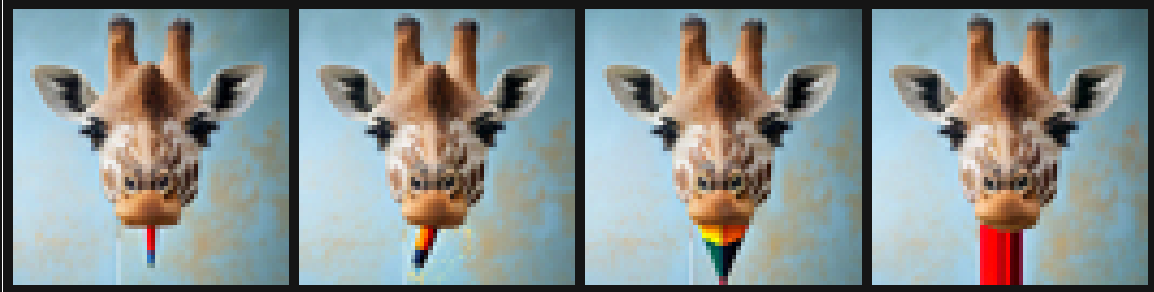


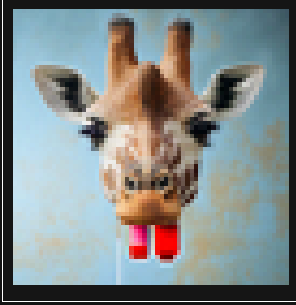




here the is prompt on another image `a pencil`







Part 1.8: Visual Anagrams -- done

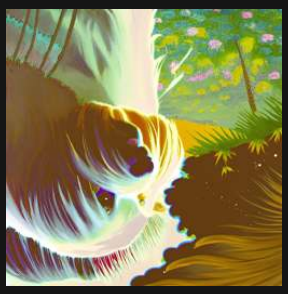
Here we want to create an image that looks like one prompt from one direction and a different prompt upside down.

problem wants ...

Correctly implemented `visual_anagrams` function ... good

A visual anagram where on one orientation `"an oil painting of people around a campfire"` is displayed and, when flipped, `"an oil painting of an old man"` is displayed.


here we have prompt1 = "a photo of a dog" and prompt2 = "a lithograph of waterfalls"

here we have prompt1 = "a photo of a man" and prompt2 = "a lithograph of a skull"


Part 1.10: Hybrid Images ... should be 1.9 I think they misnumbered it -- done

here we want to create a hybrid image that looks like one prompt from a close distance and another prompt from a far distance

Note: for this the distances are pretty far apart for it to work as well based on just using the methods in this topic as the randomness makes it difficult to get correct, also really depends on the combos some work a lot better then others i.e. ones that share a similar object border

for the deliverables they want ...

Correctly implemented make_hybrids function ... done

An image that looks like a skull from far away but a waterfall from close up

1/2 more hybrid images of your choosing.

prompt1 = 'a photo of a man' close AND prompt2 = 'a photo of a hipster barista' far

2/2 more hybrid images of your choosing.

prompt1 = 'an oil painting of an old man' close AND prompt2 = 'an oil painting of people around a campfire' far

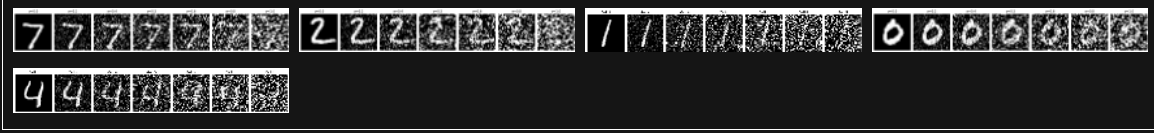
now onto part B of the project ... Diffusion Models from Scratch

In part B you will train your own diffusion model on MNIST.

Part 1: Deliverables -- done

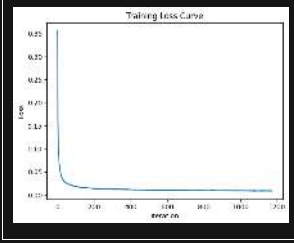
Deliverable 1

A visualization of the noising process using $\sigma = [0.0, 0.2, 0.4, 0.5, 0.6, 0.8, 1.0]$.



Deliverable 2

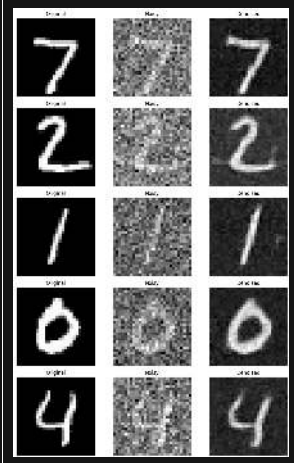
A training loss curve plot every few iterations during the whole training process



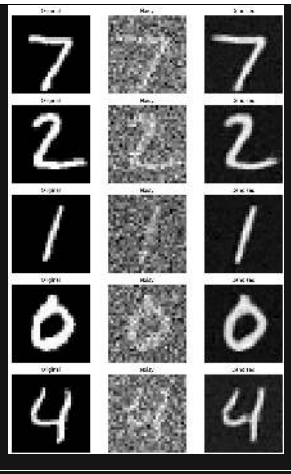
Deliverable 3

Sample results on the test set after the first and the 5-th epoch (staff solution takes ~7 minutes for 5 epochs on a Colab T4 GPU)

Denoising results after 1 epoch:



Denoising results after 5 epochs:



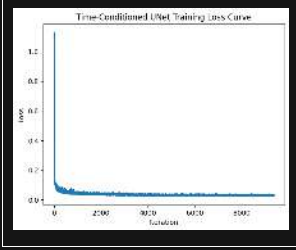
Deliverable 4

Sample results on the test set with out-of-distribution noise levels after the model is trained. Keep the same image and vary $\sigma = [0.0, 0.2, 0.4, 0.5, 0.6, 0.8, 1.0]$.

Part 2: Deliverables -- Group 1 -- done

Deliverable 1 -- P2

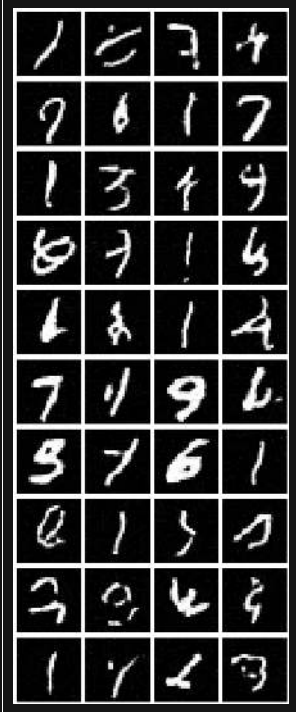
A training loss curve plot for the time-conditioned UNet over the whole training process (figure 10).



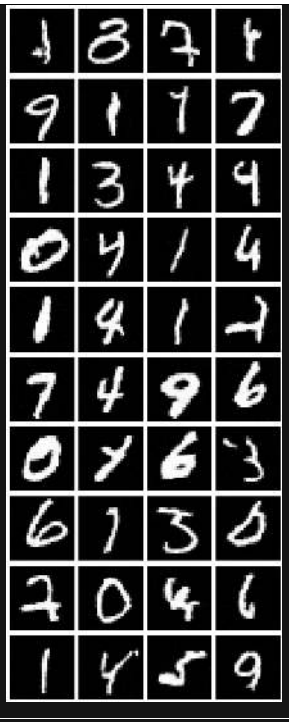
Deliverable 2 -- P2

Sampling results for the time-conditioned UNet for 5 and 20 epochs.

Samples after 5 epochs:



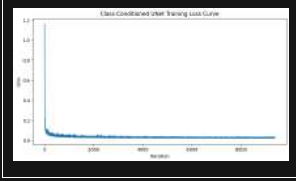
Samples after 20 epochs:



Part 2: Deliverables -- Group 2 -- done

D1

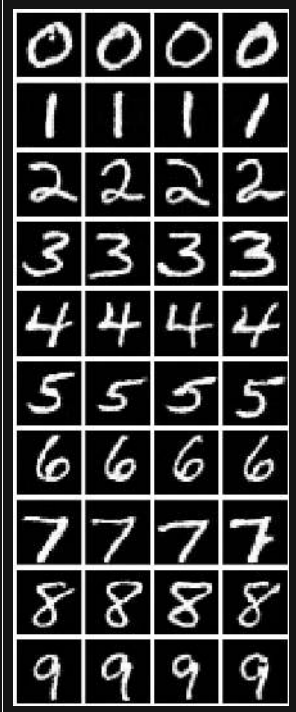
A training loss curve plot for the class-conditioned UNet over the whole training process.



D2

Sampling results for the class-conditioned UNet for 5 and 20 epochs. Generate 4 instances of each digit as shown above.

Class-Conditioned Samples after 5 epochs:



Class-Conditioned Samples after 20 epochs: