1. Which text-to-image-model?

I chose the stable diffusion model. Stable diffusion is a type of diffusion model and is known to produce high-quality, realistic images through an iterative diffusion process. Stable diffusion is a latent diffusion model which first projects the original image to a latent space (using a variational autoencoder) and then applies the diffusion process on the latent images. Since the latent space is usually much smaller than the dimensionality of the original images, this makes the model much more memory efficient. Stable diffusion also creates an embedding for the text input and has the UNet conditional on the text embedding. This ensures that the images generated by the model will stay close to the prompts.

2. Process for fine-tuning

1) Decide on the fine-tuning method

I first researched commonly used diffusion fine-tuning methods and discovered DreamBooth. After reading a few articles and the original DreamBooth paper, I decided that this method will be good for our use case. The reason is that while our training images can be divided into four main classes: hoops, necklaces, rings, and studs, there are many styles within each class. E.g. for loops class, there are beaded, chunky, charlotte, croissant, dome loops etc. Similarly for the other three classes. However, for each sub-styles, we typically have less than 10 examples, which is quite small. One thing DreamBooth emphasizes is that it's capable of fine-tuning a diffusion model using just 3-5 images of a subject.

2) Prepare the training dataset

DreamBooth separates training images into instance and class. The instance folder contains images of new subjects/concepts that we want to teach the model through fine-tuning. Class images are to ensure the diversity of generated images and reduce language drift. Again, this naturally fits into our training set. I first divided all training images into the aforementioned four classes: hoops, necklace, ring, and studs. Then within each class, I group similar styles into one instance (based on the images themselves and the titles). I end up having 9 instances for the hoops class (beaded, chunky, Charlotte, croissant, dome, easy daily, rectangle, special, tube), 3 instances for the necklace class (chain, charm, pendant) and 6 instances for the ring class (band, bold, dome, signet, slim, stacker). For the studs class, I kept them as one instance since there are only seven pictures in total and they are all of different styles. The division process also ensured each instance would have a minimum of three pictures.

3) Start training

I followed this awesome implementation of DreamBooth to fine-tune the stable diffusion model. https://colab.research.google.com/github/ShivamShrirao/diffusers/blob/main/examples/dreambooth/DreamBooth_Stable_Diffusion.ipynb

The finetuning process is in general the same as training a stable diffusion model:

i. Project the original image to latent space with VAE
ii. Sample noises that will be added to latent images
iii. Sample a random timestep and pass it to the scheduler along with the sampled noise, the scheduler will add noise proportion to the sampled timestep to the latent image
iv. Generate text embedding for the prompts which U net will be conditioned upon
v. Predict the noise residual and compute loss (the difference between the predicted noise residual and the actual noise)
vi. Update optimizer, scheduler, and model parameters

I'll point out a few things that are unique to the DreamBooth implementation:

- Use two prompts: an instance prompt and a class prompt. For each instance prompt, it's also recommended to include a rare English token that serves as a unique identifier. For example, for a pair of beaded hoops:
    - class_prompt: photo of a pair of hoops
    - instance_prompt: photo of a pair of [BH] beaded hoops,
- Generate text embedding for both prompts and concatenate the two embeddings and pass it to the model
- By default, DreamBooth calculates a class-specific prior preservation loss, which aims for the fine-tuned model to preserve some of the prior semantics of the concepts we're introducing

3. Hyperparameter tuning

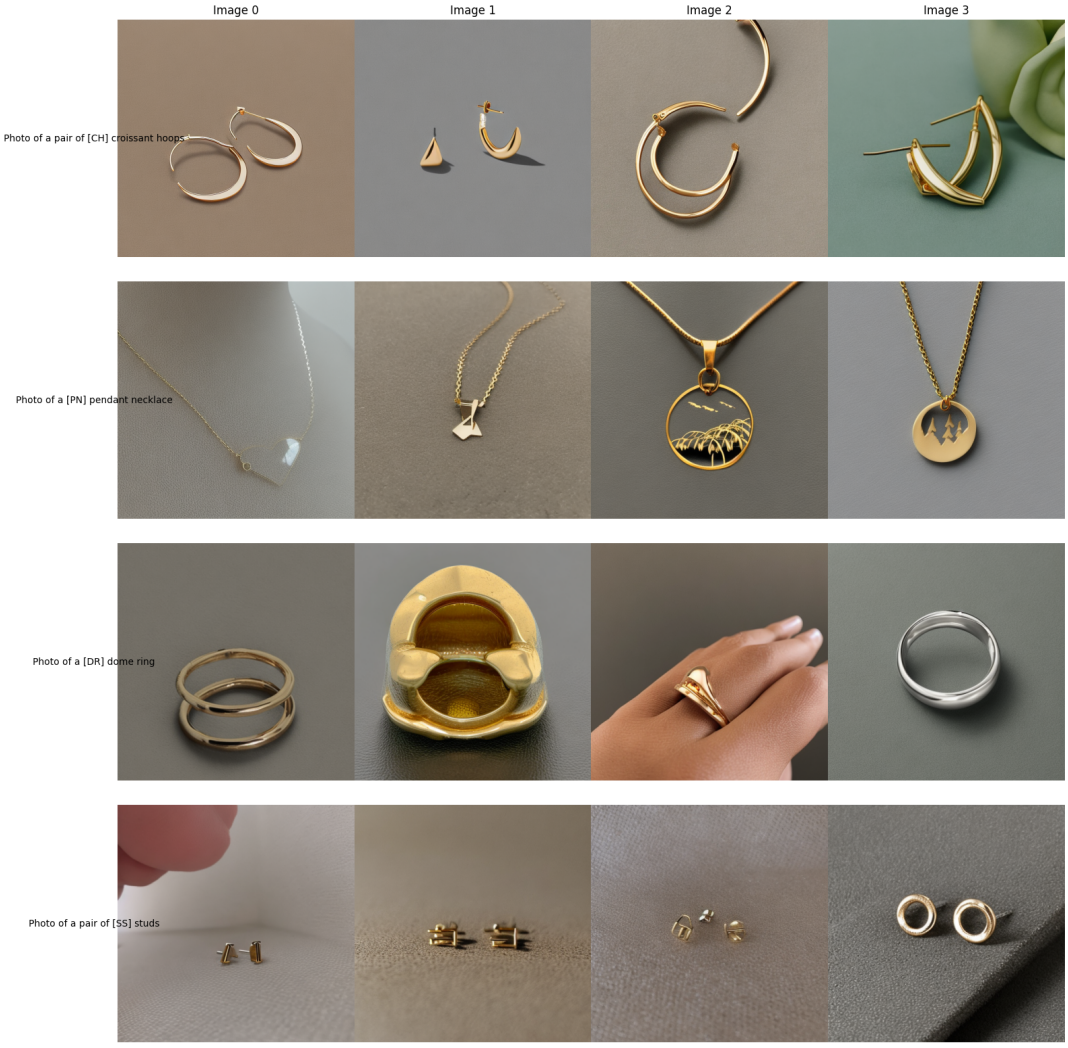
Before starting fine-tuning the hyperparameters of the model, I read this article https://huggingface.co/blog/dreambooth, which pointed out that DreamBooth is prone to overfitting and needs to find a balance between learning rate and training steps. I followed the articles' suggestion by starting with a small learning rate: 1e-6.

1) Train steps

I tried train steps = 100, 200, 300, 400. I observe that image quality improves significantly with increasing training steps. However, once the train step is greater than 300, the model starts to run into overfitting.

Images showing overfitting

The following images are trained with a learning rate set to 1e-6, train steps = 400, and no prior preservation loss. I suspect the model is overfitting as in some pictures the subject and context appear to be entangled. The second picture of the dome ring is also quite exaggerated.



2) Prior preservation loss

I also tried training with and without prior preservation loss. For this use case, I found that training with prior preservation loss does not make much difference and sometimes produces worse results. I suspect it's because the number of images for the new instances is very small (3-7) compared to the number of class images (200).

Images with prior preservation loss

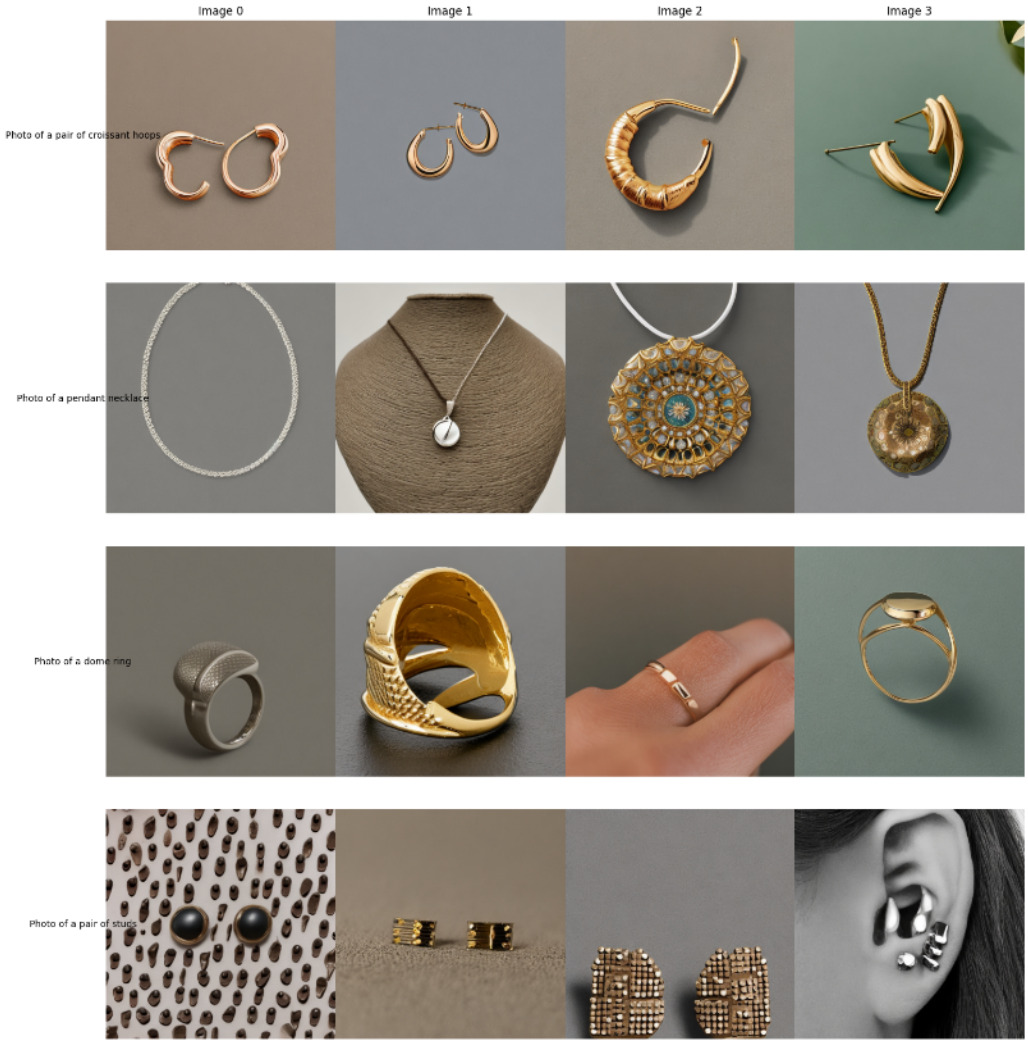
The following images are trained with a learning rate set to 1e-6, train steps = 200, and with prior preservation loss. Again in some pictures, the subject and context appear to be entangled. The model doesn't appear to have learned the concept of croissant loops very well and the second picture of the dome ring is also quite exaggerated.

3) Special token in the instance prompts

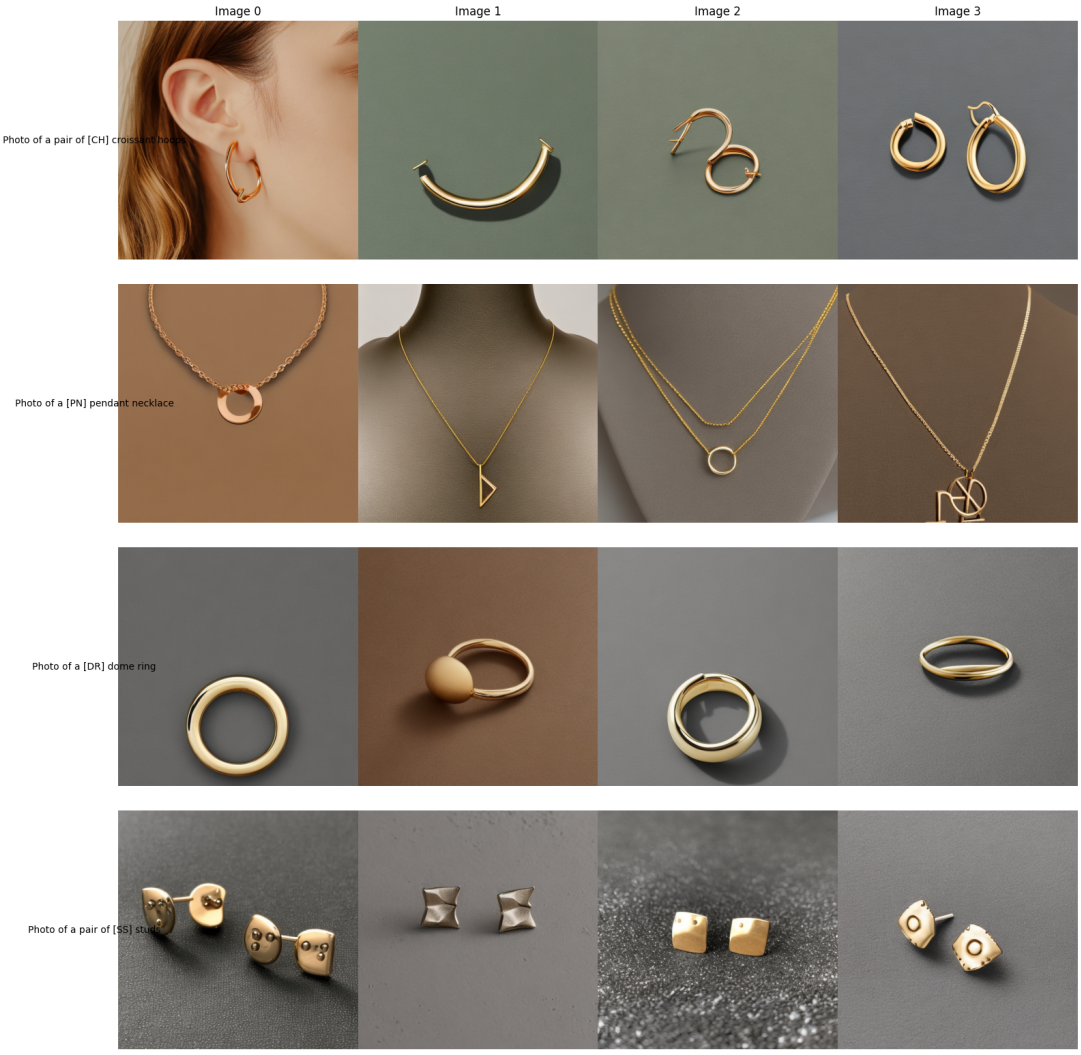
I also found that including the special token in the instance prompts does result in higher quality images as suggested by the original paper.

Images without special tokens in instance prompts

The following images are generated from a model trained with a learning rate set to 1e-6, train steps = 200, without prior preservation loss. However, at inference time I didn't include the special token in the prompts. Compared to images generated with prompts including special tokens, these image styles are a lot less consistent with the training images. Especially for pendant necklaces and studs.



Based on the image quality and cost of training, I decide on the following training parameters: learning rate = 1e-6, train steps = 200, no prior preservation loss, and including special tokens in the instance prompts. The following images were generated based on this setting.

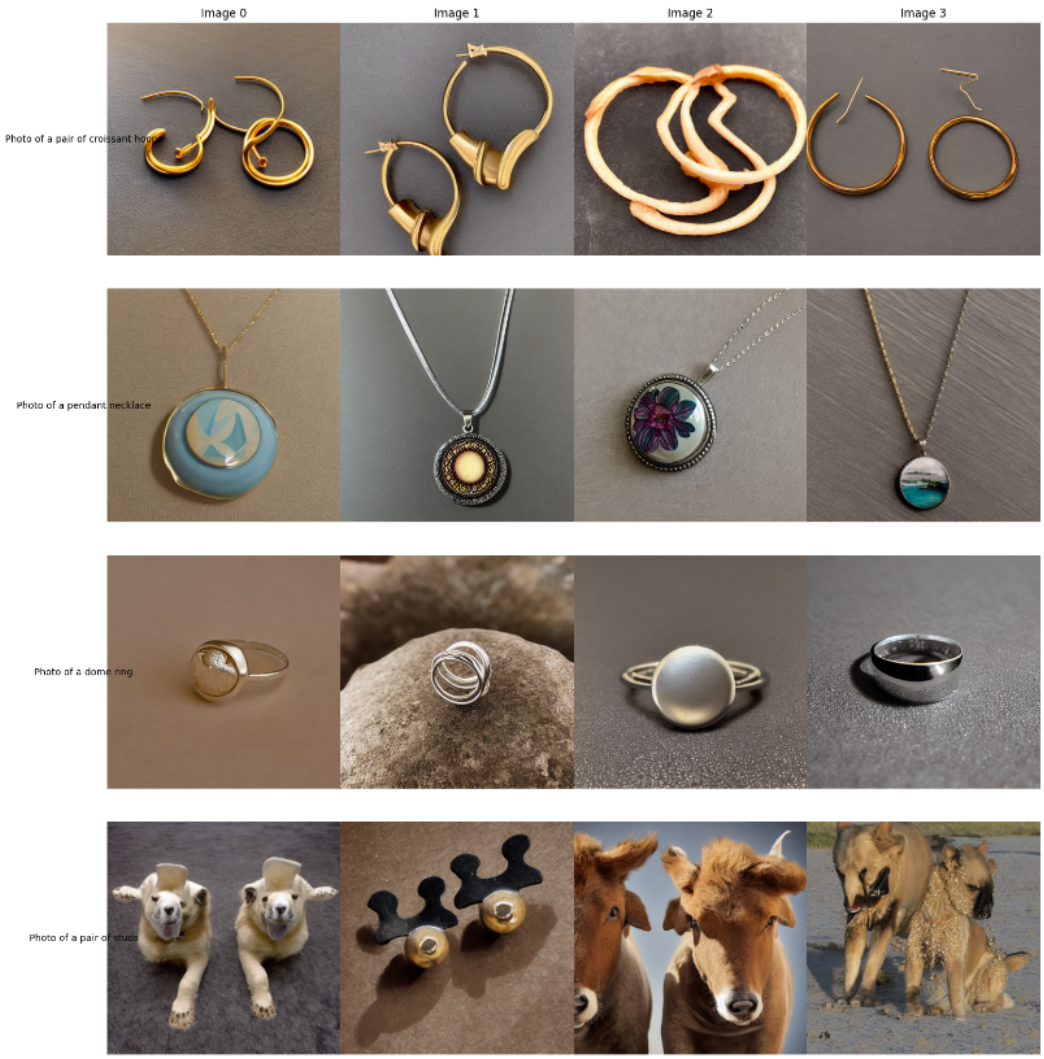


4. Qualitative evaluation and potential metrics

The quality of the generated images is quite high. When selecting an appropriate combination of hyperparameters, the model is capable of producing realistic images that preserve the style of the original training images.

For comparison, I generated images using a stable diffusion model without fine-tuning on the provided training images. In most cases, these images can correctly represent the given concepts. However, their style is not consistent with the training images at all. This inconsistency is particularly evident in the case of studs, where the unfine-tuned model produces confusing results due to the multiple meanings associated with the word.

Images without fine-tuning



Quantitative evaluation metrics

Evaluating and comparing the quality of text-to-image models quantitatively is a challenging problem. In general, good metrics should cover a few desirable properties of the generated images:

1) They should be realistic
2) They should be semantically close to the text prompts used to generate them
3) They should be diverse in their styles ideally

Some commons metrics for text-to-image tasks:

- Inception Score (IS): use a generated image to predict its label. The higher the score means the images are more diverse and meaningful.
- Frechet Inception Distance: calculates the distance between synthetic and real sample distribution. A low FID score indicates better similarity between the two distributions

The original DreamBooth paper proposed the following metrics which are also worth trying:

- CLIP-I: average pairwise cosine similarity between CLIP embeddings of generated and real images
- DINO: average pairwise cosine similarity between the ViT-S/16 DINO embeddings of generated and real images
- Prompt-fidelity: average cosine similarity between prompt and image CLIP embeddings

The first two will ensure that the generated images are realistic, while the last one ensures the generated images will stay close to the text prompts.

5. Model improvement and next steps:

1) Fine-tune stable diffusion with different noise scheduler. The DDPM scheduler used here is known to produce good results. However, it's worth trying other schedulers e.g. DDIM, PNDM, LMSDiscrete.
2) Fine-tune the stable diffusion model with Low-Rank Adaptation of Large Language Models (LoRA). LoRA has become a popular fine-tuning technique for accelerating training large models while consuming less memory. It has been shown that it can be combined with DreamBooth to achieve good results. (https://huggingface.co/docs/diffusers/training/lora#dreambooth)
3) Fine-tune the stable diffusion model with textual inversion and compare the results with DreamBooth. Textual inversion is another popular method for teaching a trained stable diffusion model new concepts. The trained weights with text inversion are also small and easy to share.
4) Explore other text-to-image models. I also want to perform the same exercise on DALL-E and Imagen, and look into the pros and cons of each model.

6. Computational requirements and cost for production:

One disadvantage of diffusion-based text-to-image models is that they are computationally expensive because of their iterative process. However, there are a few things we could do to save memory and accelerate training.

1) Load and run model weights in half-precision (float16), which will save half of the space
2) Use bitsandbytes 8-bit Adam optimizer, which allows training DreamBooth on a 16GB GPU
3) Enable memory-efficient attention offered by the xFormers package

With the abovementioned optimizations, I was able to fine-tune the stable diffusion model with ~800 images on Google Colab's NVIDIA Tesla T4 in a reasonable time.

It's hard to estimate the cost for production because it heavily depends on the type of infrastructure we choose. For example, doing inference on NVIDIA A100 is a lot faster than the Tesla T4 but also costs significantly more. The infrastructure we choose in turn depends on whether we are serving the model online or offline, and how much latency we are willing to tolerate if serving online. For serving models in production, hosted services such as Huggingface's inference endpoint are popular choices, since they are easy to use and reduce the heavy lifting of manual deployment and infrastructure management. They also offer a variety of GPU instances. https://huggingface.co/docs/inference-endpoints/pricing. I also found that start-up OctoML claims they can host large models much faster and cheaper than HuggingFace due to its in-house optimization. It might be worth looking into. https://octoml.ai/blog/how-to-run-stable-diffusion-3x-faster-at-one-fifth-the-cost/