Member-only story

How to Fine-tune Stable Diffusion using LoRA

Ng Wai Foong · Follow

8 min read · Feb 21

Personalized generated images with custom datasets

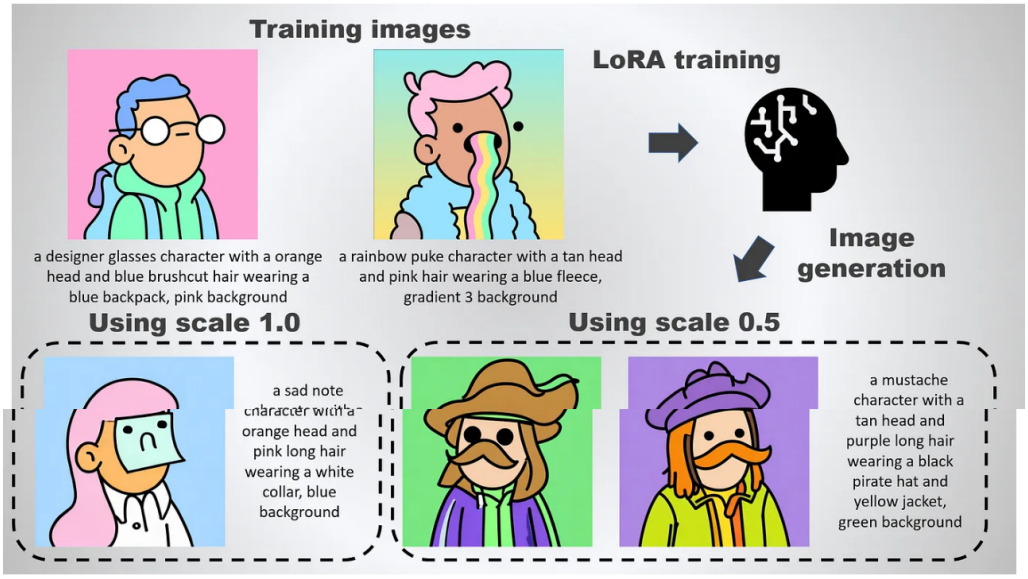


Image by the author

Previously, I have covered the following articles on fine-tuning the Stable Diffusion model to generate personalized images:

- How to Fine-tune Stable Diffusion using Textual Inversion
- How to Fine-tune Stable Diffusion using Dreambooth
- The Beginner's Guide to Unconditional Image Generation Using Diffusers

By default, doing a full fledged fine-tuning requires about 24 to 30GB VRAM.

However, with the introduction of Low-Rank Adaption of Large Language Models (LoRA), it is now possible to do fine-tuning with consumer GPUs.

> *Based on a local experiment, a single process training with batch size of 2 can be done on a single 12GB GPU (10GB without `xformers`, 6GB with `xformers`).*

LoRA offers the following benefits:

- less likely to have catastrophic forgetting as the previous pre-trained weights are kept frozen
- LoRA weights have fewer parameters than the original model and can be easily portable
- allow control to which extent the model is adapted toward new training images (supports interpolation)

This tutorial is strictly based on the `diffusers` package. Training and inference will be done using the `StableDiffusionPipeline` class directly. Model conversion is required for checkpoints that are trained using other repositories or web UI.

Let's proceed to the next section for the setup and installation.

. . .

Setup

Before that, it is highly recommended to create a new virtual environment.

Python packages

Activate the virtual environment and run the following command to install the dependencies:

```
pip install accelerate torchvision transformers datasets ftfy tensorboard
```

Next, install the `diffusers` package as follows:

```
pip install diffusers
```

For the latest development version of `diffusers`, kindly install it using the following command:

```
pip install git+https://github.com/huggingface/diffusers
```

Accelerate

Next, configure `accelerate` by running the following command at the terminal:

```
accelerate config
```

Use the following configuration to train on a local machine with mixed precision (single GPU):

```
----------------------------------------------------------------------------------
In which compute environment are you running?
This machine
----------------------------------------------------------------------------------
Which type of machine are you using?
No distributed training

Do you want to run your training on CPU only (even if a GPU is available)? [yes/
no

Do you wish to optimize your script with torch dynamo?[yes/NO]:
no

Do you want to use DeepSpeed? [yes/NO]:
no

What GPU(s) (by id) should be used for training on this machine as a comma-seper
all
----------------------------------------------------------------------------------
Do you wish to use FP16 or BF16 (mixed precision)?
fp16

accelerate configuration saved at /home/wfng/.cache/huggingface/accelerate/defau
```

For multi-GPU setup, either choose the multi-GPU option

```
Which type of machine are you using?
Multi-GPU
```

or override the existing configuration by passing in the following arguments when training:

```
accelerate launch --multi_gpu --gpu_ids="1,2" --num_processes=2 train.py \
    ...
```

- `multi_gpu` — training will be done using multiple GPUs
- `gpu_ids` — determine which GPU to be used for training (separated by comma)
- `num_processes` — number of processes for parallel training. Set it to a value of 2 to use 2 GPUs.

xformers (optional)

The `xformers` package helps to improve the inference speed. As of version 0.0.16, there are pip wheels support for PyTorch 1.13.1.

Pip install (win/linux)

For those with `torch==1.13.1`, simply run the following command to install `xformers`:

```
pip install -U xformers
```

Conda (linux)

For conda users, the installation only supports either `torch==1.12.1` or `torch==1.13.1`

```
conda install xformers
```

Building from source

For the other use cases, consider building `xformers` directly from source:

```
# (Optional) Makes the build much faster
pip install ninja
# Set TORCH_CUDA_ARCH_LIST if running and building on different GPU types
pip install -v -U
git+https://github.com/facebookresearch/xformers.git@main#egg=xformers
# (this can take dozens of minutes)
```

. . .

Datasets

There most common ways to prepare the training datasets are as follows:

- upload or use datasets on HuggingFace Hub that comes with the `image` and `text` keys
- a folder containing the `metadata.jsonl` and all the relevant training images

Datasets on HuggingFace Hub

Head over to HuggingFace Hub and locate any image datasets that comes with captions. Use the unique datasets name as the `dataset_name` argument. On the first run, it will download the files to the `.cache` folder automatically. On the subsequent run, the training script will reuse the cache version of the datasets for training.

Have a look at the following command which uses the `lambdalabs/pokemon-blip-captions` datasets for training:

```
accelerate launch train_text_to_image_lora.py \
  --dataset_name="lambdalabs/pokemon-blip-captions" \
  ...
```

> The datasets above comes with *Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)* license.

Custom images in a folder

Create a new file called `metadata.jsonl` and fill it using the following syntax:

```
{"file_name": "images/xxx.png", "text": "a drawing of a dog with red eyes"}
{"file_name": "images/xxy.png", "text": "a drawing of a cat sleeping"}
```

```
{"file_name": "images/train/xxz.png", "text": "a drawing of a pink rabbit"}
```

Each line consists of a dictionary that represents the metadata for a single image:

- `file_name`: the corresponding file path of the image
- `text`: the caption for the image

The training images can be located at any directory as long as it match the `file_name` value in the `metadata.jsonl` file.

Have a look at the following folder structure as reference:

```
data/metadata.jsonl
data/images/xxx.png
data/images/xxy.png
...
data/images/train/xxz.png
data/images/val/yyz.png
```

The training script will locate the `metadata.jsonl` file and load the corresponding training images based on its content. Simply set the `train_data_dir` argument to the base folder. For example:

```
accelerate launch train_text_to_image_lora.py \
    --train_data_dir="data" \
    ...
```

. . .

Training

Access the `train text to image lora.py` training script from the official repository and save it locally in the working directory.

The script is based on based on the latest development version. If you are using an older version of `diffusers`, it will report an error due to non-matching version. Locate the `check_min_version` function in the script and comment it out as follows:

```
...
# check_min_version("0.13.0.dev0")
```

Datasets on HuggingFace Hub

Run the following command to start the training on datasets that are available on the HuggingFace Hub:

```
accelerate launch train_text_to_image_lora.py \
  --pretrained_model_name_or_path=runwayml/stable-diffusion-v1-5 \
  --dataset_name="lambdalabs/pokemon-blip-captions" \
  --resolution=512 --center_crop --random_flip \
  --train_batch_size=1 \
  --num_train_epochs=100 \
  --learning_rate=1e-04 --lr_scheduler="constant" --lr_warmup_steps=0 \
  --seed=42 \
  --output_dir="output" \
  --validation_prompt="a drawing of a pink rabbit"
```

> Replace the `dataset_name` argument with the unique datasets name that are available on the HuggingFace Hub.
> Replace the *`pretrained_model_name_or_path`* argument with the desired Stable Diffusion model.

- `resolution` — The resolution for input images, all the images in the train/validation datasets will be resized to this. Higher resolution requires higher memory during training. For example, set it to 256 to train a model that generates 256 x 256 images.
- `train_batch_size` — Batch size (per device) for the training data loader. Reduce the batch size to prevent Out-of-Memory error during training.
- `num_train_epochs` — The number of training epochs. Default to 100.
- `checkpointing_steps` — Save a checkpoint of the training state every X updates. These checkpoints are only suitable for resuming. Default to 500. Set it to a higher value to reduce the number of checkpoints being saved.

Add the `enable_xformers_memory_efficient_attention` argument if xformers is installed. This will reduce memory and speed up the training process.

```
accelerate launch train_text_to_image_lora.py \
  ...
  --enable_xformers_memory_efficient_attention
```

Custom images in a folder

Make sure to have a `metadata.jsonl` file in the training folder. Have a look at the following directory structure as reference:

```
|- data (folder)
|  |- metadata.jsonl
|  |- xxx.png
|  |- xxy.png
|  |- ...
|- train_text_to_image_lora.py
```

> The `data` folder contains the `metadata.jsonl` file and all the relevant training images for this tutorial.

For training a conditional image generation model with LoRA using custom datasets, use the `train_data_dir` argument instead. For example:

```
accelerate launch train_text_to_image_lora.py \
  --pretrained_model_name_or_path=runwayml/stable-diffusion-v1-5 \
  --train_data_dir="data" \
  --resolution=512 --center_crop --random_flip \
  --train_batch_size=1 \
  --num_train_epochs=100 \
  --learning_rate=1e-04 --lr_scheduler="constant" --lr_warmup_steps=0 \
  --seed=42 \
  --output_dir="output" \
  --validation_prompt="a drawing of a pink rabbit"
```

> *During the first run, it will download the Stable Diffusion model and save it locally in the `cache` folder. In the subsequent run, it will reuse the same cache data.*

Tensorboard

By default, the script will only save LoRA weights once at the end of the training.

```
|- output
|  |- checkpoint-5000
|  |- checkpoint-10000
```

```
|  |- checkpoint-10000
|  |- checkpoint-15000
|  |- checkpoint-20000
|  |- logs
|- data
|- train_text_to_image_lora.py
```

It is a good idea to utilize the tensorboard package for monitoring the training. Open a new terminal and run the following command:

```
tensorboard --logdir output
```

The following will be displayed on the terminal:

```
Serving TensorBoard on localhost; to expose to the network, use a proxy or
pass --bind_all
TensorBoard 2.12.0 at http://localhost:6006/ (Press CTRL+C to quit)
```

Open a new tab in a browser and access the TensorBoard page at the following URL:

```
http://localhost:6006
```

You should see the training metrics such as `train_loss`.

Resume from checkpoint

To resume from an existing checkpoints, use the resume_from_checkpoint argument and set it to the desired checkpoints:

```
accelerate launch train_text_to_image_lora.py \
  ...
  --resume_from_checkpoint="output/checkpoint-20000"
```

Set the value to `latest` to automatically select the last available checkpoint:

```
accelerate launch train_text_to_image_lora.py \
  ...
  --resume_from_checkpoint="latest"
```

. . .

Inference

Once the training is completed, it will generate a small LoRA weights called `pytorch_lora_weights.bin` at the output directory.

Create a new file called `inference.py` and append the following code inside it:

```
from diffusers import StableDiffusionPipeline
import torch

device = "cuda"

# load model
model_path = "./output/pytorch_lora_weights.bin"
pipe = StableDiffusionPipeline.from_pretrained(
    "runwayml/stable-diffusion-v1-5",
    torch_dtype=torch.float16,
    safety_checker=None,
    feature_extractor=None,
    requires_safety_checker=False
)

# load lora weights
pipe.unet.load_attn_procs(model_path)
# set to use GPU for inference
pipe.to(device)

# generate image
prompt = "a drawing of a white rabbit"
image = pipeline(prompt, num_inference_steps=30).images[0]
# save image
image.save("image.png")
```

Then, run the following command to generate images with the newly trained LoRA weights:

```
python inference.py
```

Deterministic generation

By default, the generated images will be different on each run. In order to reproduce the same generated images, create a new `torch.Generator`

instance with the desired seed and pass it as input parameter to the pipeline object:

```
...

# create a generator with seed 42 for reproducibility
generator = torch.Generator(device=device).manual_seed(42)

prompt = "a drawing of a white rabbit"
image = pipeline(prompt, num_inference_steps=30, generator=generator).images[0]
# save image
image.save("image.png")
```

> *The example above used 42 as the seed. Modify it to other values to obtain different images.*

Cross attention arguments

The `StableDiffusionPipeline` class accepts an additional parameter called `cross_attention_kwargs`, which can be used to interpolate the inference output.

Simply pass in a dictionary containing a `scale` key with a value between 0 and 1. For example:

```
...

image = pipeline(
    prompt,
    num_inference_steps=30,
    generator=generator,
    cross_attention_kwargs={"scale": 1.0}
).images[0]
# save image
image.save("image.png")
```

> *A value of 0 means that the LoRA weights will not be used. On the other hand, a value of 1 means that only the LoRA fine-tuned weights will be used. Values between 0 and 1 will interpolate between the two weights.*

. . .

Conclusion

Let's recap some of the learning points for this article.

It started off with a brief introduction on the advantages of using LoRA for fine-tuning Stable Diffusion models.

The article continued with the setup and installation processes via `pip install`. Also, manual configuration is required to setup the `accelerate` module properly.

Next, it covered how to prepare the datasets. The training script supports datasets that are available on the HuggingFace Hub or custom datasets in a local folder.

Then, it moved on to the training process. It explained in detail on some of the useful training arguments, how to monitor the training via `tensorboard`, and how to resume from an existing checkpoint.

Lastly, it highlighted model inference using the newly trained LoRA model for conditional image generation.

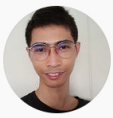
Thanks for reading this piece. Feel free to check out my other articles. Have a great day ahead!

. . .

References

1. Github — diffusers
2. HuggingFace — LoRA Support in Diffusers

Machine Learning | Stable Diffusion | Lora | Text To Image Generation | Diffusers

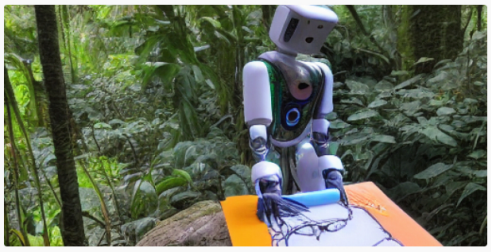


Written by Ng Wai Foong

4.1K Followers

Senior AI Engineer@Yoozoo | Content Writer #NLP #datascience #programming #machinelearning | Linkedin: https://www.linkedin.com/in/wai-foong-ng-694619185/

More from Ng Wai Foong



Ng Wai Foong in Towards Data Science

How to Fine-tune Stable Diffusion using Textual Inversion

Personalized generated images with custom styles or objects

· 7 min read · Aug 31, 2022

301 11

Ng Wai Foong in Better Programming

The Beginner's Guide to Pydantic

A Python package to parse and validate data

· 7 min read · Aug 11, 2020

731 2

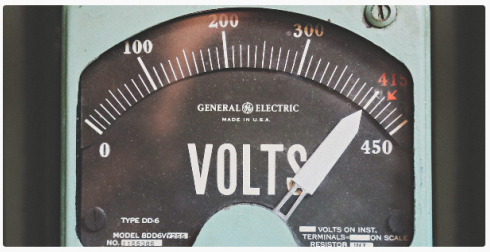
Ng Wai Foong in Better Programming

How to Send an Email With Attachments in Python

Utilizing 'smtplib' to send images and text emails

· 5 min read · Mar 24, 2020

270 4



Ng Wai Foong in Better Programming

How to Optimize Stable Diffusion for Image Generation

Faster inference and lower memory consumption

· 7 min read · Feb 3

61

See all from Ng Wai Foong

Recommended from Medium

Ng Wai Foong in Better Programming

The Beginner's Guide to Unconditional Image Generation...

Explore and generate unique and imaginative images based on existing datasets

✨ · 8 min read · Feb 1

82 1

LUCA TOPRAK in Generative AI

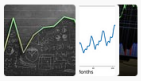
Introducing ControlNet: A Game-Changer for Stable Diffusion in...

The field of artificial intelligence-generated images is constantly changing, with new too...

✨ · 4 min read · Mar 27

105

Lists



Predictive Modeling w/ Python

18 stories · 125 saves

Practical Guides to Machine Learning

10 stories · 135 saves

Natural Language Processing

402 stories · 53 saves

The New Chatbots: ChatGPT, Bard, and Beyond

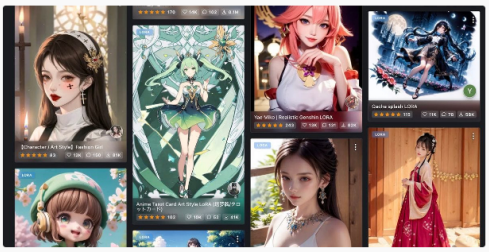
13 stories · 48 saves

Umberto Grando

Stable Diffusion Ultimate Guide pt. 1: Setup

Learn how to setup the Stable Diffusion UI on your machine and start generating pictures...

✨ · 10 min read · Mar 11



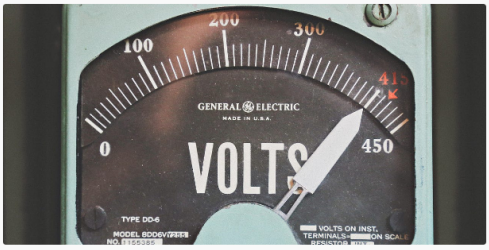
Edmond Yip in StableDiffusion

How to use LoRA Model in Civital?

This tutorial also tell you how to setup preview of Extra Networks

✨ · 3 min read · Jun 5

78 1



Ng Wai Foong in Better Programming

How to Optimize Stable Diffusion for Image Generation

Faster inference and lower memory consumption

· 7 min read · Feb 3

61

Mason McGough in Towards Data Science

Stable Diffusion as an API

Remove people from photos with a Stable Diffusion microservice

· 12 min read · Feb 4

81

See more recommendations