
ATTEMPT 1 of 2

I decided to start my experimentation for this task on Modal because of good experiences in the past working with the service, and more importantly, a fantastic tutorial for fine tuning stable diffusion on Modal for an icon generator here:

https://modal.com/blog/fine-tuning-stable-diffusion

I curated my dataset based on the provided XML files and deployed it here:

https://huggingface.co/datasets/Shivamkak/kanjiCaptions

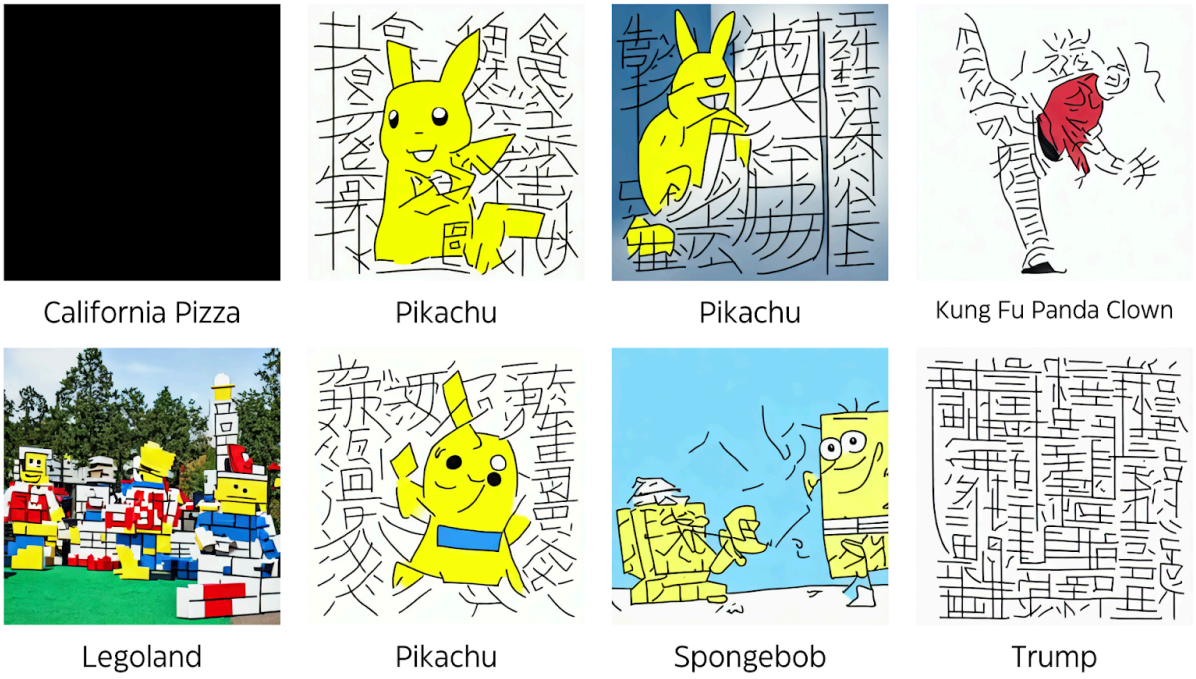
Please let me know if you would like to see my data parsing code. I will not include it in the appendix for this section as it was fairly straight forward, and I want to avoid making the submission overly long. For my published dataset, I followed the data preparation format requested by the Modal tutorial, ie a metadata.csv file in the following format:

file_name	text
亜_kanji_04e9c.png	"Image of a Japanese kanji character 亜 in plain font. The character means: Asia, rank next, come after, -ous."
唖_kanji_05516.png	"Image of a Japanese kanji character 唖 in plain font. The character means: mute, dumb."
娃_kanji_05a03.png	“Image of a Japanese kanji character 娃 in plain font. The character means: beautiful.”

ATTEMPT 1 of 2 - RESULTS

Initial generations were trained off of the base model `runwayml/stable-diffusion-v1-5`.

I curated a sample of prompts and resultant kanji character generation below, and I will compare to these across my various attempts on this project. You may notice 3 of the 8 slots are Pikachu … this is a feature of my testing, not a flaw. It was triply important that my model learned adequate kanji character generation for "Pikachu".



[Full Fine-Tuning of runwayml/stable-diffusion-v1-5 on Kanji Dataset]

Then, I added the following style prompting:

```
# System prompt to enforce black and white kanji style
style_prefix = "A minimal, black and white kanji character. 5 STROKES MAXIMUM. Pure black strokes on pure white background. Single character only, centered. Minimalist, thick brush strokes in the style of Japanese calligraphy. "

# Negative prompt to prevent unwanted elements
negative_prompt = "color, colors, colorful, grey, grayscale, detailed, complex, ornate, decorative, artistic, multiple characters, background, texture, 3d, shading, gradient"
```

And I re-deployed to generate the results below. This version is still live on my Modal Gradio app here:

https://shivamkak19--instantkanji-sandbox-fastapi-app.modal.run/

[Full Fine-Tuning of runwayml/stable-diffusion-v1-5 on Kanji Dataset with Style Prompting]

Never in my life have I seen a Kanji character with that many strokes. Maybe I haven't seen enough Kanji. It suffices to say that these results are hardly satisfactory. And I might add, the results without the style prompting were less accurate, but far more entertaining :)

ATTEMPT 2 of 2

This time around I decided to try a LORA fine tuning on top of SD3.5 Large, available here:

https://huggingface.co/stabilityai/stable-diffusion-3.5-large

I also could have done a full fine-tuning, but my intuition was that LORA would be sufficient for the purposes of kanji character generation. I am extremely curious how results would differ between a full finetuning and LORA of SD3.5 Large, as I suspect results would be nearly identical due to the large size of the Kanji Dataset I am using for LORA.

However, to save on my compute resources (I spent $450 on compute across all tasks in this problem set :..)), I haven't had the chance to try it out yet.

I followed the official SD3.5 fine-tuning tutorial from Stability.ai's notion site: https://stabilityai.notion.site/Stable-Diffusion-3-5-fine-tuning-guide-11a61cdcd1968027a15bdbd7c40be8c6#12461cdcd19680788a23c650dab26b93

I trained the model using `SimpleTuner`, and I trained using an **A100 (40 GB SXM4) on Lambda Labs.** I set max training steps for 36000, following the equation below:

- my Kanji Dataset has 6413 entries
- I do not want any repeats
- I found a batch size of 6 do be appropriate for my compute resource
- 30 epochs

$$\text{Max training steps} = \left(\frac{\text{Number of samples} \times \text{Repeats}}{\text{Batch size}} \right) \times \text{Epochs}$$

[Equation originally from SD3.5 fine-tuning tutorial linked directly above]

ATTEMPT 2 of 2 - RESULTS

I ran a total of two training runs on SD3.5 Large, the first with `lr=1.05e-3`, and the second with `lr=9.5e-4`. In both cases I used a cosine learning rate scheduler. My hypothesis was that a larger learning rate would yield more extreme cases of exploding gradients but would converge to more satisfactory results. Since this fine-tuning task is for a very niche style compared to

---

SD3.5 base generation, a more aggressive learning rate likely helps enforce style patterns from the training data set. I couple this with a `lora_rank=768` and `lora_alpha=768` in both experimental runs to allow for a stronger capacity to learn style patterns.

Below, I have curated exemplary checkpoints for both training runs, as well as a screenshot of the validation image at all checkpoints (available for download in full resolution at given links).

`validation prompt="mountains"` | checkpoint_image_strip_mountain.png

[Base] [9400 Steps] [12000 Steps]

[Validation Images for LORA Fine-Tuning of SD3.5 Large on Kanji Dataset | LR = 1.05e-3]

`validation prompt="mountains"` | kanji_lora2_checkpoint_image_strip_mountain.png

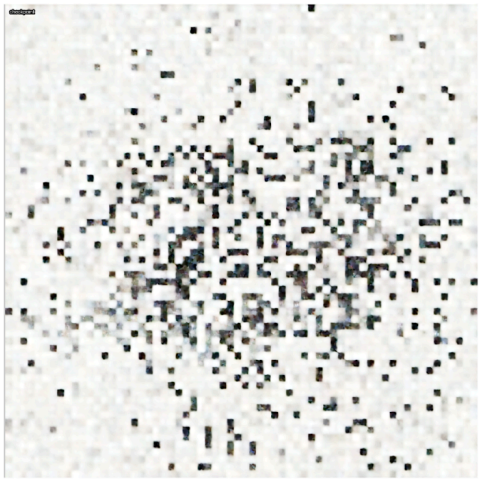
[Base] [5400 Steps] [9600 Steps]

[Validation Images for LORA Fine-Tuning of SD3.5 Large on Kanji Dataset | LR = 9.5e-4]

Both learning rates experienced phases of exploding gradients in accordance with the cosine learning rate scheduler.

Exploding Gradients

[LR = 1.05e-3; 7600 steps]



[LR = 9.5e-4; 7800 steps]

However, I found that the exploding gradients were more extreme in the case of `lr=9.5e-4`. Heuristically, it appeared that at times the phases of exploding gradients for `lr=9.5e-4` were exploding off of the SD3.5 base rather than the LORA fine-tune. I did not see this trend repeat in the case of `lr=1.05e-3`. This may very well be due to the fact that the lower learning rate is transferring styles from the kanji dataset less aggressively.

Exploding Gradients - Extreme Case

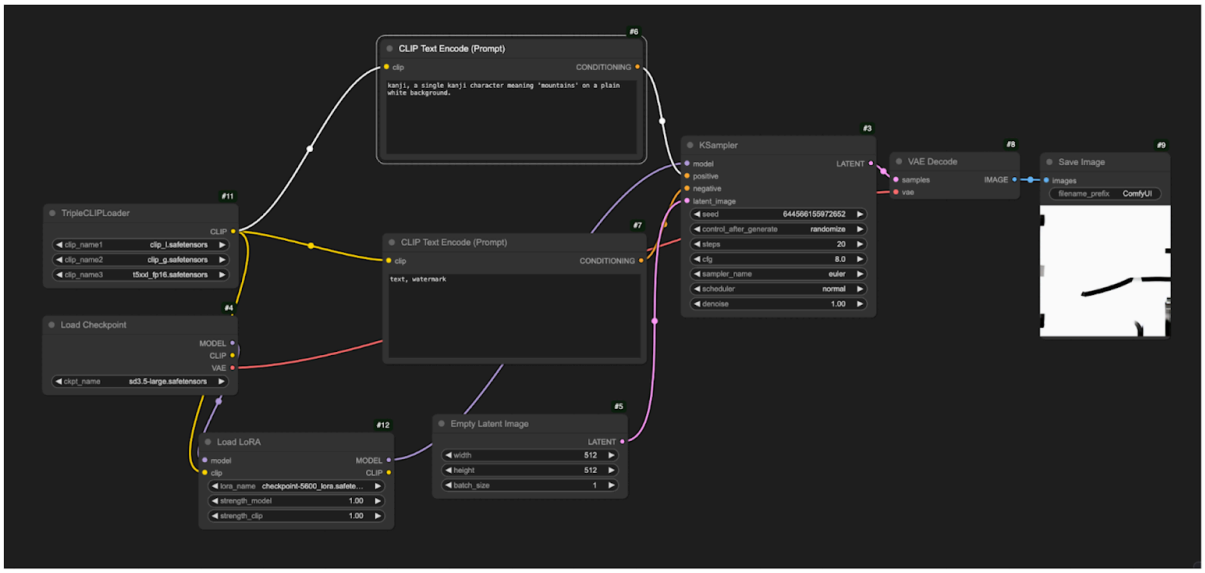
[LR = 9.5e-4; 7800 steps]

After evaluating the validation benchmarks above for both training runs, I decided to proceed with the `lr=1.05e-3`. Further, based off of visual inspection, I decided to use my safetensor checkpoint at 14400 steps for freestyle generation in ComfyUI.

I published this checkpoint to Hugginface, and it is currently publicly available here: https://huggingface.co/Shivamkak/sd35L_kanji_lora

In the default ComfyUI setup, text encoders are not loaded properly. Specifically, I needed to pass the text encoders to a TripleClipLoader. I found the fix in this resolved GitHub issue: https://github.com/comfyanonymous/ComfyUI/issues/5388

Additionally, I followed this guide to locate the correct text encoders: https://comfyanonymous.github.io/ComfyUI_examples/sd3/



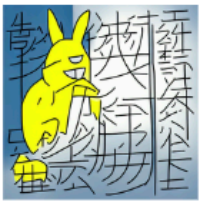
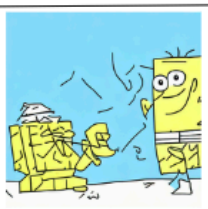
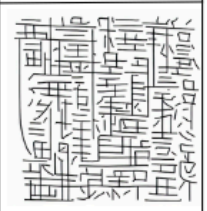
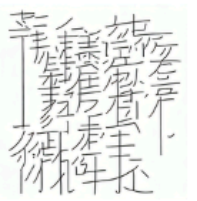
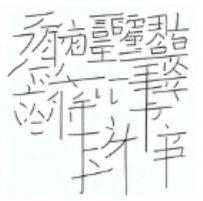
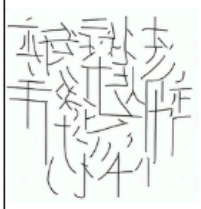
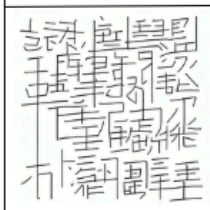
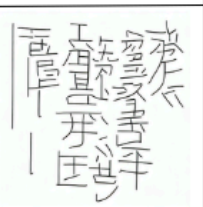
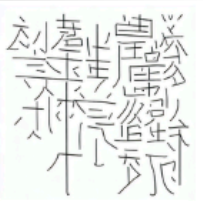
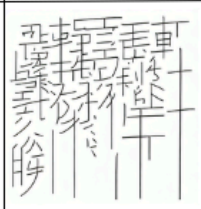
[ComfyUI Workflow for Loading LORA Checkpoints on Top of SD3.5 Large]

Using the checkpoint at 14400 steps for the `lr=1.05e-3` run, I made the following generations:

`prompt="Kung Fu Panda Clown"` | kanji_lora1_checkpoint_image_strip_kungfu.png

`prompt="Middle-Eastern Uncle who feeds shawarma to local monkeys"` | kanji_lora1_checkpoint_image_strip_middle_east_uncle_shawarma.png

Below, I have included a final summary of results based on my important `Triple-Pikachu & Miscellaneous Creatures Benchmark`:

	 California Pizza	 Pikachu	 Pikachu	 Kung Fu Panda Clown
SD1.5 Full Fine-Tune	 Legoland	 Pikachu	 Spongebob	 Trump
	 California Pizza	 Pikachu	 Pikachu	 Kung Fu Panda Clown
SD1.5 Full Fine-Tune + Style Prompting	 Legoland	 Pikachu	 Spongebob	 Trump
	 California Pizza	 Pikachu	 Pikachu	 Kung Fu Panda Clown
SD3.5 LORA Fine-Tune [LR = 1.05e-4]	 Legoland	 Pikachu	 Spongebob	 Trump