Chapter 9. Advanced Techniques for Image Generation with Stable Diffusion

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the authors' raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 9th chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the author at ccollins@oreilly.com.

Most work with AI images only requires simple prompt engineering techniques, but there are more powerful tools available when you need more creative control over your output, or want to train custom models for specific tasks. These more complex abilities often requires more technical ability and structured thinking as part of the workflow of creating the final image.

All images in this chapter are generated by Stable Diffusion XL unless otherwise noted, as in the sections relying on extensions such as ControlNet, where more methods are supported with the older v1.5 model. The techniques discussed were devised to be transferrable to any future or alternative model. We make extensive use of AUTOMATIC1111's Stable Diffusion WebUI, and have provided detailed setup instructions that were current as of the time of writing, but please consult the official repository for up-to-date instructions, and to diagnose any issues you encounter.

Running Stable Diffusion

Stable Diffusion is an open-source image generation model, so you can run it locally on your computer for free, if you have an Nvidia or AMD GPU, or Apple Silicon, as powers the M1, M2, or M3 Macs. It was common to run the first popular version (1.4) of Stable Diffusion in a Google Colab notebook, which provides access to a free GPU in the cloud (though you may need to upgrade to a paid account if Google limits the free tier). Visit the Google Colab website if you haven't used it before, or to find the latest information on limits. A copy of this Python notebook is saved in the GitHub repository for this book, but you should upload it to Google Drive and run it in Google Colab to avoid setup issues.

Installing Stable Diffusion can be done via the HuggingFace diffusers libary, alongside a handful of dependencies. In the Google Colab the fol-

lowing code installs the necessary dependencies (you would drop the exclamation marks *!* if installing locally rather than in a Jupyter Notebook or Google Colab):

```
!pip install diffusers==0.11.1
!pip install transformers scipy ftfy accelerate
```

In order to download and use the model, you first build an inference pipeline (what runs when we use the model):

```
# create an inference pipeline
import torch
from diffusers import StableDiffusionPipeline

pipe = StableDiffusionPipeline.from_pretrained(
    "CompVis/stable-diffusion-v1-4",
    torch_dtype=torch.float16)

pipe = pipe.to("cuda")
```

Let's break down the script line by line:

1. `import torch` : This line is importing the torch library, also known as PyTorch. PyTorch is an open-source machine learning library, used for applications such as computer vision and natural language processing.
2. `from diffusers import StableDiffusionPipeline` : Here the script is importing the `StableDiffusionPipeline` class from the `diffusers` library. This specific class is probably a pipeline for using diffusion models, of which Stable Diffusion is the most popular example.
3. `pipe = StableDiffusionPipeline.from_pretrained("CompVis/stable-diffusion-v1-4", torch_dtype=torch.float16)` : This is creating an instance of the `StableDiffusionPipeline` class with pre-trained weights. The method `from_pretrained` loads the weights of a pre-trained model - in this case, the model is `CompVis/stable-diffusion-v1-4` . The `torch_dtype=torch.float16` argument specifies that the data type used in the model should be float16, which is a half-precision floating point format. Using float16 can speed up model computation and reduce memory usage (necessary to stay within the Google Colab free tier limits).
4. `pipe = pipe.to("cuda")` : Finally, this line moves the pipe model to the GPU. The string `"cuda"` refers to CUDA, a parallel computing platform and application programming interface (API) model created by Nvidia. By doing this, all computations performed by the pipe model will be executed on the GPU, which can be significantly faster than running them on a CPU for large-scale models and data.

Now that we have our pipe, we can pass in a prompt and other parameters for the model, like a random seed (change this to get a different image each time), the number of inference steps (more steps takes time but results in a higher quality image), and the guidance scale (how closely the image matches the prompt):

```
# run inference on a prompt
prompt = "a photograph of an astronaut riding a horse"

generator = torch.Generator("cuda").manual_seed(1024)

image = pipe(prompt, num_inference_steps=50,
    guidance_scale=7, generator=generator
    ).images[0] # image here is in PIL format

# Now to display an image you can either save it such as:
image.save(f"astronaut_rides_horse.png")

# If you're in a google colab you can directly display:
image
```

The output is shown in Figure 9-1.

Figure 9-1. Photograph of an astronaut riding a horse

Let's walk through this script to explain what it does:

1. `prompt = "a photograph of an astronaut riding a horse"`: This is the prompt that will be passed into the model to guide the generation of an image.
2. `generator = torch.Generator("cuda").manual_seed(1024)`: In this line, a PyTorch Generator is created and assigned to the generator variable. The Generator is initialized with `"cuda"`, which means that it will be using a GPU for computations. The `manual_seed(1024)` function is used to set the random seed for generating random numbers, ensuring that the results are reproducible. If you run this code with the same model you should get the exact same image.
3. `image = pipe(prompt, num_inference_steps=50, guidance_scale=7, generator=generator).images[0]`: This line runs the pipe model on the prompt to generate an image. The `num_inference_steps` argument is set to 50, meaning that the model will perform 50 steps of inference. The `guidance_scale` argument is set to 7, which adjusts how strongly the prompt guides the

generated image (higher values tend to get grainy and less diverse). The generator argument passes in the random number generator created earlier. The result is an array of generated images, and `images[0]` selects the first image from this array.

4. `image.save(f"astronaut_rides_horse.png")`: This line saves the generated image to a file.
5. `image`: Finally, this line of code will display the image if the code is running in an environment like a Jupyter notebook or Google Colab. This happens because these environments automatically display the result of the last line of code in a code cell if it is not assigned to a variable.

It's powerful to be able to run an open-source model locally or in the cloud, and customize it to meet your needs. However, custom coding your own inference pipelines and building a user interface on top is likely overkill unless you are an extremely advanced user with deep machine learning knowledge, or your intention is to build your own AI image generation product. Stablity AI, the company funding development of Stable Diffusion, has a hosted web interface called Dream Studio (Figure 9-2), which is similar to the DALL-E playground, also operating on a credit system and offering advanced functionality such as inpainting:

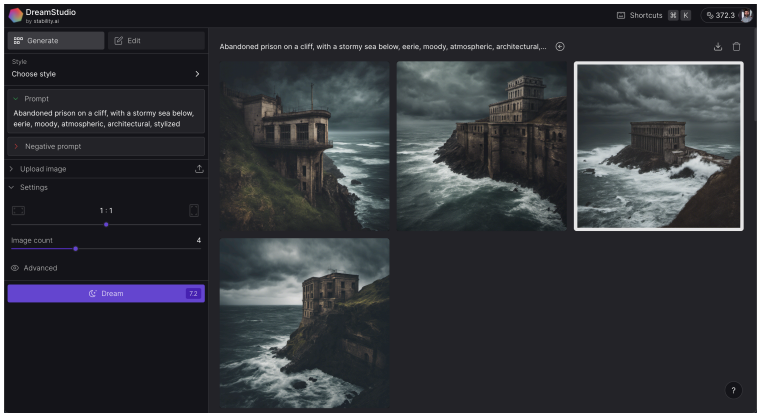

Figure 9-2. Stability AI Dream Studio

Like DALL-E, Dream Studio offers access via API, which can be convenient for building AI image applications or running programmatic scripts for generating lots of images, without the encumberance of hosting and running your own Stable Diffusion model. Visit *https://beta.dreamstudio.ai/account* once you have created an account to get your API key, and top up with credits (at time of writing, 1,000 credits cost $10 and can generate approximately 5,000 images). The following code is included in the GitHub repository for the book:

```
import os
import base64
import requests
from IPython.display import Image

engine_id = "stable-diffusion-xl-1024-v1-0"
api_host = os.getenv('API_HOST', 'https://api.stability.ai')
api_key = os.getenv("STABILITY_API_KEY")

image_description = "computers being tied together"
prompt = f"""an illustration of {image_description}. in the
style of corporate memphis, white background, professional,
clean lines, warm pastel colors"""
```

```
response = requests.post(
    f"{api_host}/v1/generation/{engine_id}/text-to-image",
    headers={
        "Content-Type": "application/json",
        "Accept": "application/json",
        "Authorization": f"Bearer {api_key}"
    },
    json={
        "text_prompts": [
            {
                "text": prompt,
            }
        ],
        "cfg_scale": 7,
        "height": 1024,
        "width": 1024,
        "samples": 1,
        "steps": 30,
    },
)

if response.status_code != 200:
    raise Exception(
        "Non-200 response: " + str(response.text))

data = response.json()

image_paths = []

# if there's no /out folder, create it
if not os.path.exists("./out"):
    os.makedirs("./out")

for i, image in enumerate(data["artifacts"]):
    filename = f"./out/image-{i}.png"
    with open(filename, "wb") as f:
        f.write(base64.b64decode(image["base64"]))

    image_paths.append(filename)

# display the first image
Image(filename=image_paths[0])
```

The output is shown in Figure 9-3.

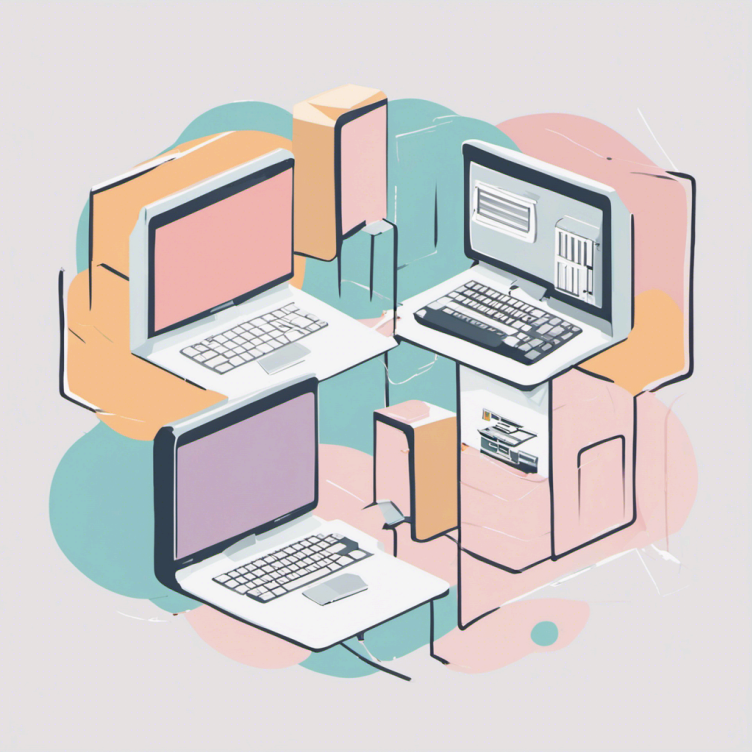

Figure 9-3. Corporate Memphis illustration from the Dream Studio API

Let's break down this code step by step:

1. First, set up the required environment variables:
    - `engine_id`: This refers to a specific model version at `stability.ai`.
    - `api_host`: It retrieves the API host URL from environment variables. If not set, it defaults to `'https://api.stability.ai'`.
    - `api_key`: Retrieves the API key from environment variables.
2. The `prompt`: Defines how the image should look, including the style and colors.
3. A `POST` request is made to the URL derived from `api_host` and `engine_id`.
    - The headers for the request are set to accept and send JSON data, and include an authorization header with the `api_key`.
    - The JSON body of the request specifies the prompt (description of the image), the desired scale of the image, its dimensions, the number of samples, and the number of steps.
4. If the status code of the response is not 200 (indicating a successful request), an exception is raised with the response text to indicate something went wrong. Otherwise, the response is parsed into JSON format.
5. If there isn't a directory named *out*, one is created. For each artifact (image) in the response, the code does the following:
    - Sets a filename path.
    - Decodes the base64-encoded image data from the response.
    - Writes the decoded image data to a file.
    - Appends the file's path to the `image_paths` list.
    - This is typically where you would save the image to Google Cloud Storage or Amazon Simple Storage Service (S3), in order to display later in your application.
6. The first image from the `image_paths` list (the only one, in this case) is displayed (only in Jupyter Notebooks or Google Colab) using

the `Image` class from `IPython.display`.

The downside of using Stability AI's service is a lack of control over customization. One of the great benefits of Stable Diffusion being open-source is the ability to modify almost any aspect of the model, and make use of community-built advanced functionality. In addition, there is no guarantee that functions or features you rely on for your scripts today will still be there in the future, as Stability AI strives to live up to the expectations of their investors, legal team, and corporate customers. For example, the popular (and more permissive) version 1.5 model has been deprecated in favor of the new Stable Diffusion 2.0 and XL models, causing problems for those who had finely-tuned their workflows, parameters, and prompts to work with v1.5.

AUTOMATIC1111 Web User Interface

Heavy users of Stable Diffusion typically recommend the AUTOMATIC1111 (pronounced "automatic eleven eleven") web user interface, because it is feature-rich and comes with multiple extensions built by Stable Diffusion power users. This project is the gateway to taking advantage of the best aspect of Stable Diffusion: the vibrant open-source community who have dedicated countless hours to integrating advanced functionality to the tool. Advancd users may also wish to explore ComfyUI, as it supports more advanced worklows and increased flexibility (including image-to-video), but we deemed this too complex for the majority of use cases, which can easily be handled by AUTOMATIC1111.

You can use the normal text-to-image Stable Diffusion model, but also run image-to-image (similar to the base image feature in Midjourney), as well as upscaling finished images for higher quality, and inpainting (as is offered by DALL-E). It's even possible to train and run custom models within this interface, and there are thousands of models shared publicly in communities such as HuggingFace and Civitai.

WARNING

Some custom open-source models are NSFW (Not Safe For Work), so be careful when browsing websites like Civitai.

Running Stable Diffusion locally with AUTOMATIC1111 requires some technical setup, and it's best to look for an up-to-date guide in the AUTOMATIC1111 Wiki:

- Install and run on NVidia GPUs
- Install and run on AMD GPUs
- Install and run on Apple Silicon

Installation generally involves ensuring you have Git and Python installed (as well as any other dependencies), and downloading Stable Diffusion, as well as the Automatic1111 code to your local computer. The images in this chapter use the XL 1.0 version of Stable Diffusion, though many still use the older version 1.5 as it is considered more permissive and has a wealth of custom community-trained models. The techniques work the same across models, though the results and quality will differ: it's commonly believed that removing NSFW images from the training

data for version 2.0 led to worse performance at generating (even non-explicit) images of realistic human figures (though this seems largely corrected in the XL version).

As the model is open-source, you can get SDXL v1.0 on your local computer by visiting the model page on HuggingFace for the base and refiner models, and downloading the `.safetensors` files from the Files and versions tab. This format is safer than the previous *.ckpt* file format, as it does not execute code on your computer when running.

- Base model - *sd_xl_base_1.0.safetensors*
- Refiner model - *sd_xl_refiner_1.0.safetensors*

These models take time to download, so start downloading them now and later you will place them in your *models/Stable-diffusion* folder once you have installed the AUTOMATIC111 interface. If you wish to use the older v1.5 Stable Diffusuion model, download the *v1-5-pruned-emaonly.ckpt* file from HuggingFace, and move that into the models folder where you placed the base and refiner models.

Once you have everything installed, the web interface is accessed by running a script that launches the application locally, which will show up as a web address in your browser. As one example, here are the current instructions (at time of writing) for Windows, with a computer that has an Nvidia GPU:

1. Install Python 3.10.6 (ticking Add to PATH), and git
2. Open the command prompt from search bar, and type `git clone https://github.com/AUTOMATIC1111/stable-diffusion-webui`
3. Remember to move the `sd_xl_base_1.0.safetensors` and `sd_xl_refiner_1.0.safetensors` models into the `stable-diffusion-webui/models/Stable-diffusion` folder
4. Double-click *webui-user.bat* file and visit the address the interface is running on (usually `http://127.0.0.1:7860`). For Mac or Linux, you would run `bash webui.sh` in the terminal.

From this interface, shown in Figure 9-4 taken (from the official GitHub repository), you can enter your prompt (top left, under the txt2img tab) and click generate to get your image.

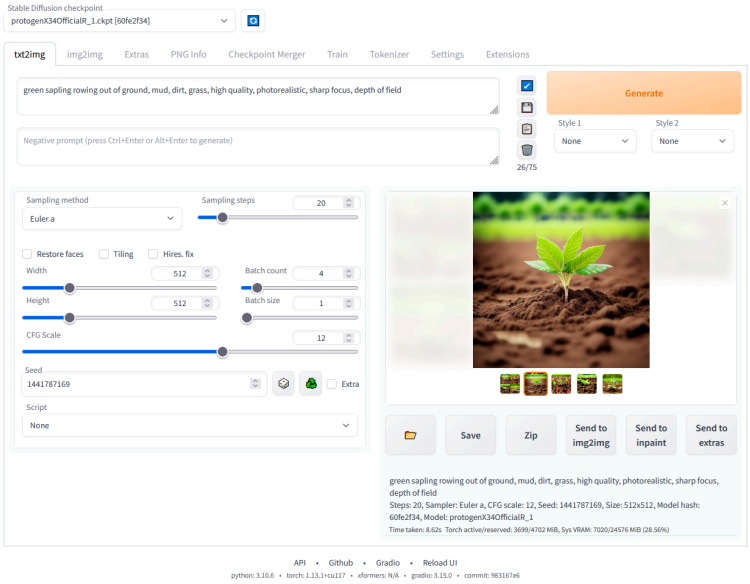

Figure 9-4. Stable Diffusion Web UI

If you run into an error or if you downloaded AUTOMATIC1111 web UI a while ago and need to update it, you can enter the *stable-diffusion-webui* folder in your terminal and run `git pull`. If you are running into errors, you may reset your implementation (move any files you want to save first) by running `git checkout -f master` in the *stable-diffusion-webui* folder.

WARNING

Resetting AUTOMATIC1111 this way will delete any files in the folder, along with any customizations. We recommend you make a local copy in a different folder for recovery.

The box immediately below the prompt input is where you can add negative prompts to remove concepts from an image and ensure they don't show up (see Chapter 8 for more on negative prompts). Underneath you'll find a number of settings including the Seed (set to -1 for a new image each time), number of Sampling (inference) Steps, Batch Count (number of generations to run one after another), and Batch Size (number of images processed in each batch at the cost of higher VRAM needed). When images are generated you can download them from the interface directly, send them to various tabs with the buttons below, or visit the *stable-diffusion-webui/outputs* folder where they are organized by method (`text2img`, `img2img`) and date.

```
stable-diffusion-webui/
    outputs/
        txt2img-images/
            2023-10-05/
                your_image.png
```

When you run the AUTOMATIC1111 web UI any models you downloaded will appear in the Stable Diffusion Checkpoint dropdown menu at the top. Select the base model and enter your prompt as well as adjusting your settings as normal. Make sure you set the image size to 1024x1024. For now, set the "Switch at" parameter under Refiner to `1` to run only the base model, as in Figure 9-5.

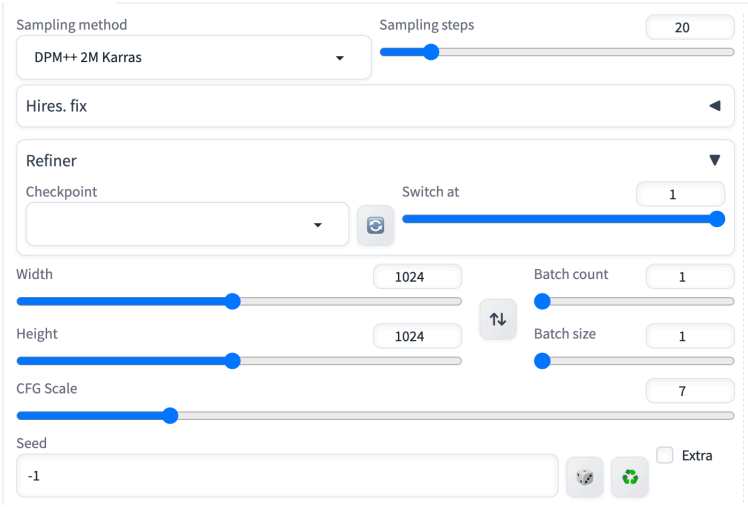

Figure 9-5. Standard Settings for SDXL

The sampling methods available are relatively complex and technical to explain, but the tradeoffs are generally between speed, quality, and randomness. `Euler` is the simplest sampler, and `DDIM` was the first designed specifically for Diffusion models. The sampling methods that have an *a* in the name, for example `Euler a`, are ancestral samplers, which inject noise into the image as part of the process. This gives less reproduceable results as the image does not converge (there is some randomness to the image each time you run the model). The `DPM++ 2M Karras` or `UniPC` sampler running for 20 - 30 steps are excellent choices for robust, stable, and reproduceable images. For higher-quality but slower and more random images, try the `DPM++ SDE Karras` or `DDIM` samplers with 10-15 steps.

Another important parameter is the CFG Scale (Classifier Free Guidance - the same as the `guidance_scale` introduced in the Stable Diffusion Inference Google Colab example). As a rule of thumb, here are common values for CFG Scale and what they equate to:

- *1*: Mostly ignore the prompt.
- *3*: Feel free to be creative.
- *7*: A good balance between the prompt and creativity.
- *15*: Adhere to the prompt.
- *30*: Strictly follow the prompt.

You can change the size of the image generated with Height and Width, as well as the number of images using Batch Count. The checkbox Highres fix uses an upscaler to generate a larger high resolution image (more on this later), the Restore faces checkbox uses a face restoration model (by default `Codeformer`) to fix the defects in human faces that often occur with Stable Diffusion, and the Tiling checkbox creates an image that can be tiled in a repeating pattern. There's also the ability to save and insert styles which are just prompts you want to reuse regularly. There are many powerful features in the different tabs, as well as community-built extensions you can add, with more added as they become available.

AUTOMATIC1111 supports prompt weights, or weighted terms, much like Midjourney (covered in Chapter 8). The way you access them is slightly different, as instead of separating by double colons like in Midjourney, you use parentheses. For example, `(pirate)` would emphasize pirate features by 10% or 1.1, and double parentheses `((pirate))` would mul-

tiply it again, so the weight would be 1.1 x 1.1 = 1.21. You can also control the weights precisely by inputing your own number in the form of (keyword: factor), for example `(pirate: 1.5)`, for the model to pay 50% more attention to those tokens:

Input:

```
Marilyn Monroe as a (pirate:1.5) on a desert island, detailed clothing,
by Stanley Artgerm Lau and Alphonse Mucha
```

Negative:

```
racy, nudity, cleavage
```

The output is shown in Figure 9-6.

Figure 9-6. Marilyn Monroe pirate

Square brackets `[pirate]` work the same way but in reverse, de-emphasising a term in a prompt by 10%. So for example, `[hat]` would be the same as a weight of 0.9, or `(hat:0.9)`. Note this is not the same as a negative prompt, because the term will still be present in the generation of the image, just dialed down. Prompt weights work in the negative prompt box as well, acting to more aggressively remove that concept from the image, or reduce their effects. This can be used to ensure unwanted elements or styles don't appear, when a negative prompt isn't enough.

GIVE DIRECTION

Providing more or less emphasis on specific words or sections of a prompt can give you more fine-grained control over what the model pays attention to.

A more advanced technique used by power users of AUTOMATIC1111 is *prompt editing*, also known as *prompt switching*. During the diffusion process the early steps move from random noise to a fuzzy outline of the general shapes expected to be in the image, before the final details are filled in. Prompt editing allows you to pass a different prompt to the early or later steps in the diffusion process, giving you more creative control. The syntax is `[from:to:when]`, where `from` is your starting prompt, `to` is your finishing prompt, and `when` is when to make the switch, denoted in number of steps or a decimal representing a percentage. The

prompt `[emma watson: amber heard: 0.5]` would start generating an image of Emma Watson, before switching halfway to generating an image of Amber Heard on top of the last frame, finishing with a mixture of the two actresses. This is a useful trick for creating images of people that look attractive and vaguely familiar, without being recongizeable as any specific celebrity, and therefore may be seen as more ethical and legally sound than simply copying a celebrity's likeness (seek your own legal counsel):

Input:

```
vogue fashion shoot of [emma watson: amber heard: 0.5],
highly realistic, high resolution, highly detailed,
dramatic, 8k
```

The output is shown in Figure 9-7.

Figure 9-7. Emma Watson and Amber Heard mixed

PROVIDING DIRECTION

Prompt editing is an advanced technique which gets deep into the actual workings of the diffusion model. Interfering with what layers respond to what concepts can lead to very creative results if you know what you're doing, and are willing to undergo enough trial and error.

If you want the model to alternate between two concepts, the syntax is `[emma watson | amber heard]`, which will make the switch at every step, ending with a more blended mixture. There are many advanced uses of prompt editing, though it is seen as something of a dark art. In some cases experts report being able to get around difficult generations, for example starting by generating something easy for the model to generate, before switching to what is really needed in the final details phase. In practice we have found limited use out of this technique, but you should experiment and see what you can discover.

Img2Img

The AUTOMATIC1111 web UI supports `Img2Img` (Figure 9-8), which is the functional equivalent to Midjourney's ability to submit an image along with the prompt. It grants you more control over the style and composition of your resulting image, by uploading an image for the model to use as guidance. To get good results with `Img2Img` try using `Euler` sampling, 50 sampling steps, and a higher than usual CFG scale of 20 to 30:

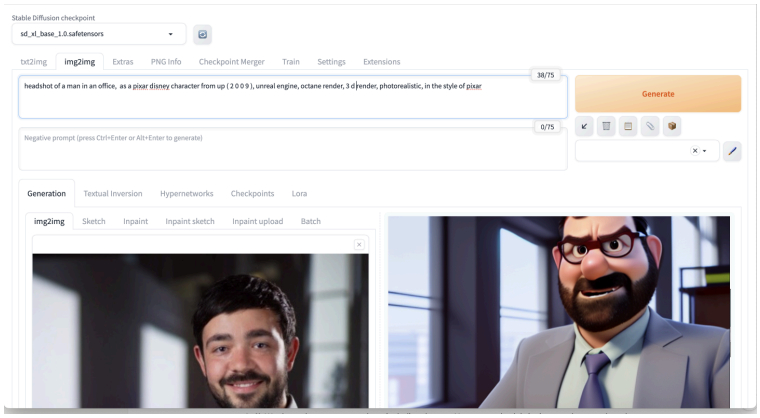

Figure 9-8. Img2Img

The parameters are the same as the normal `Text2Image` mode with the addition of *Denoising Strength*, which controls how much random noise is added to your base image before running the generation process. A value of 0 will add zero noise, so your output will look exactly like your input, and a value of 1 will completely replace your input with noise (functionally the same as using `Text2Image`). Often you need to experiment with different combinations of values for Denoising Strength, CFG Scale, and Seed alongside the words in your prompt. The following example in Figure 9-9 creates a character in Pixar style just for fun: we wouldn't recommend using protected IP in your prompt for commercial use:

Input:

```
headshot of a man in an office,  as a pixar disney character
from up ( 2 0 0 9 ), unreal engine, octane render, 3 d
render, photorealistic, in the style of pixar
```

The output is shown in Figure 9-9.

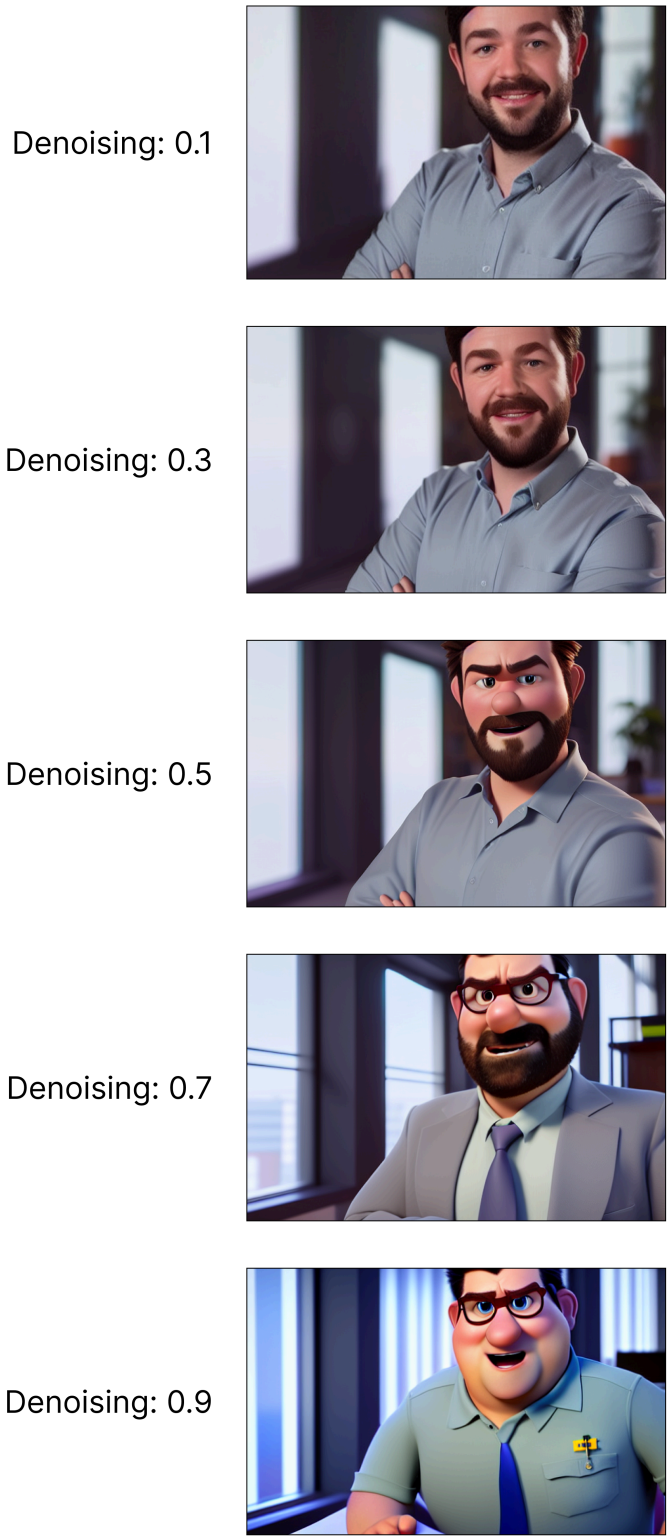

Figure 9-9. Denoising Strength

If you want to test many different values for a parameter in AUTOMATIC1111 and generate a grid as is shown in Figure 9-9, that is supported in the Script dropdown at the bottom, where you can select X/Y/Z Plot and choose up to three parameters to generate multiple values for. For example, you may try also adjusting the CFG scale to see how it interacts with Denoising. Figure 9-10 shows how to select multiple values for the Denoising strength parameter. When you click the Generate button a grid of images will be made, and you can find each individual image

that populates the grid in your Output folder under the method (i.e. `Text2Image`, or `Img2Img`) and today's date:

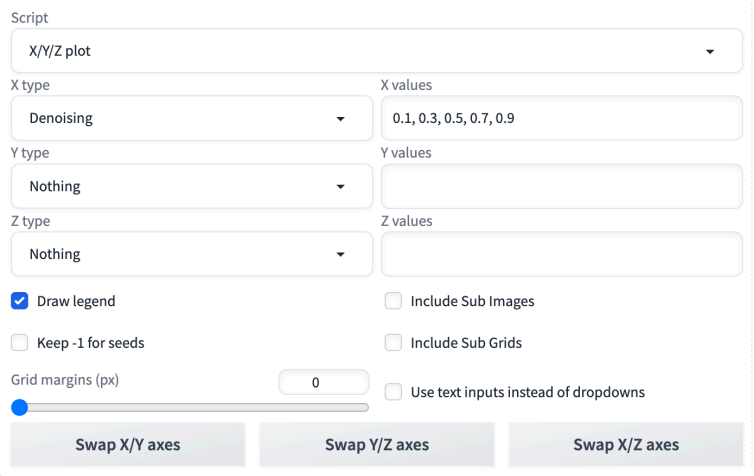

Figure 9-10. X/Y/Z Plot of Denoising parameter

EVALUATE QUALITY

Generating a grid of many different parameter combinations or values is one of the powerful advantages of running Stable Diffusion locally. Although it may take time to generate lots of images, there's no better way to visually identify exactly what a parameter does, and where the sweet spot is in terms of quality.

If you forgot what settings or prompt you used to generate an image, AUTOMATIC1111 saves this as metadata on every image generated. You can visit the PNG Info tab (Figure 9-11) to read that metadata whenever needed. This also works with images you get from other users of the web interface, but only if they have posted the image on a website that doesn't strip out this metadata:

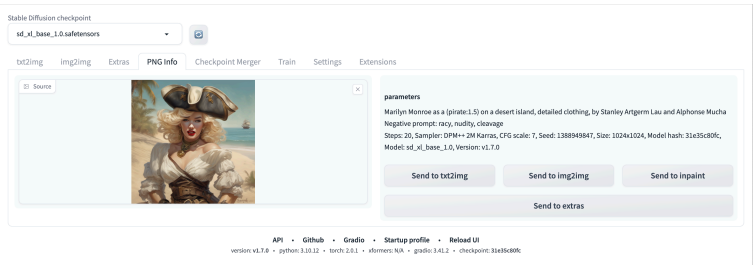

Figure 9-11. PNG Info tab

The Resize Mode options are there to determine what happens when you upload an image that doesn't match the dimensions of your base image, for example going from 1000 x 500 to 512 x 512, either stretching the aspect ratio to fit with Just Resize, cropping a part of the image in the right aspect ratio with Crop and Resize, assing noise to pad out the image with Resize and fill or generating an image in the new dimensions Just resize (latent upscale).

Upscaling Images

There's also the ability to upscale images to higher resolution in AUTOMATIC1111's `Img2Img` tab, just like you can in Midjourney, but with more control. Upload your image and add a generic prompt like

`highly detailed` in the prompt box. This is necessary because the upscaler works by breaking the image into tiles, expanding so there are gaps between the tiles, then *filling in* the gaps using the prompt and context of the surrounding pixels. Go down to Scripts at the bottom and select the SD Upscale script, then choose an upscaler (Figure 9-12).

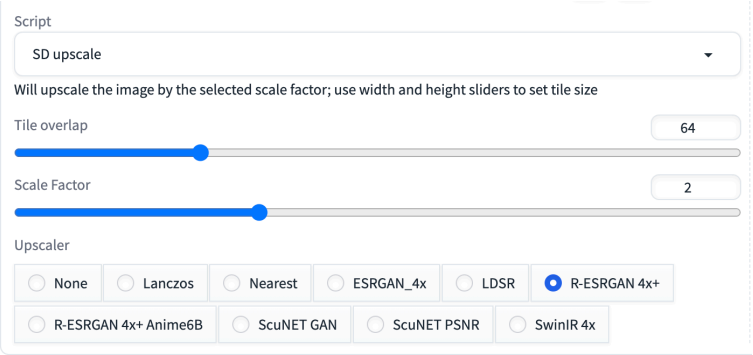

Figure 9-12. SD Upscale interface

Typically we have found the `R-ESRGAN 4x+` upscaler as a good default, but this can sometimes give a cartoonish quality as can be seen in Figure 9-12 with the grass. There are more models available to test if you aren't getting good results. When you download a new model (a *.pth* file) you just need to place it in the *ESRGAN* folder and restart the web interface for them to show up (in your terminal). You can also get good results with upscaling by modifying the prompt, particularly if you are losing some detail or the style is changing too much. However, it is not advised to use your original prompt, as that would have the strange effect of Inpainting the same image in each tile. To show a wider quality difference we have used the v1.5 model to generate the original image (SDXL creates images that are 4x larger, and at a higher quality, so upscaling is less needed):

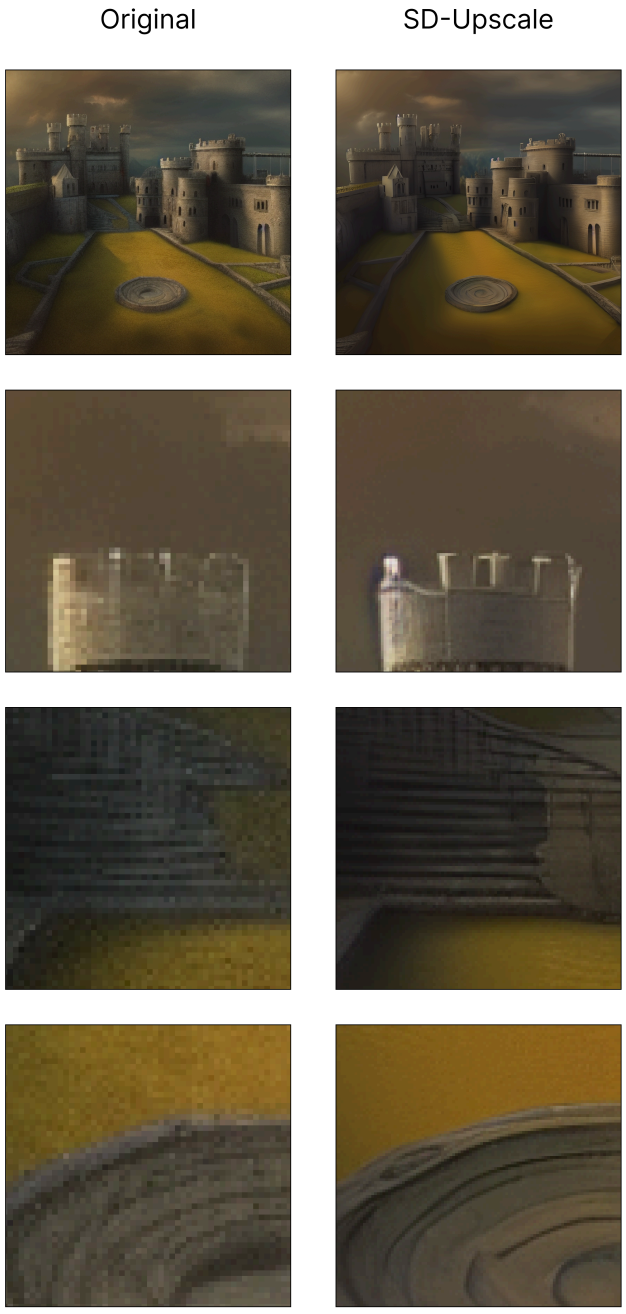

Figure 9-13. Upscaled image comparison

SPECIFY FORMAT

If you're going to use the images you generate in the real world, often you can't just use a square 512 x 512 image in low resolution. Using upscaling you can generate an image in any size and whatever the required resolution.

As with all things Stable Diffusion it helps to experiment, but for good results we recommend a high number of steps (150-200+), a CFG scale of 8-15, and a Denoising strength of 0.1 to 0.2 to keep the base image intact. You can click Generate to get the resulting upscaled image (512x512 becomes 1024x1024), and then you can either download the higher resolution image or click Send to Img2Img and click generate again to double the size of the image again. The process can take a significant amount of

time due to the multiple tile generations and large number of sampling steps, approximately 10-30 minutes on a M2 Macbook Air.

Interrogate CLIP

In the Img2Img tab the CLIP embeddings model (which is also used by Stable Diffusion) is implemented in the Interrogate CLIP button (in some versions shown as a paperclip), which allows you to reverse-engineer the prompt from an image, similar to Midjourney's Describe feature, covered in Chapter 8. Once you click the button and the script has run, the prompt will appear in your prompt box (Figure 9-14):

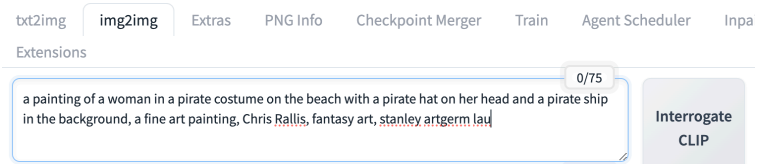

Figure 9-14. Interrogate CLIP

Output:

```
a painting of a woman in a pirate costume on the beach with a pirate hat on her head and a pira
```

SD Inpainting and Outpainting

Img2Img also supports Inpainting and Outpainting, and provides a simple canvas tool for creating the mask. To use Inpainting or Outpainting, click the Inpaint sub-tab in the Img2Img tab and upload your image. It's optionally recommended to use a specific Inpainting model for better results, which you can install by downloading the *sd-v1-5-inpainting.ckpt* file and moving it into your *Models* > *Stable-Diffusion* folder. Restart the interface and the model should appear in the top left dropdown. The canvas allows you to use a brush to remove parts of the image just like in DALL-E (see Chapter 8), which is adjustable in size for fine-grained control. In Figure 9-15 the center of a stone circle in the middle of a castle courtyard has been removed.

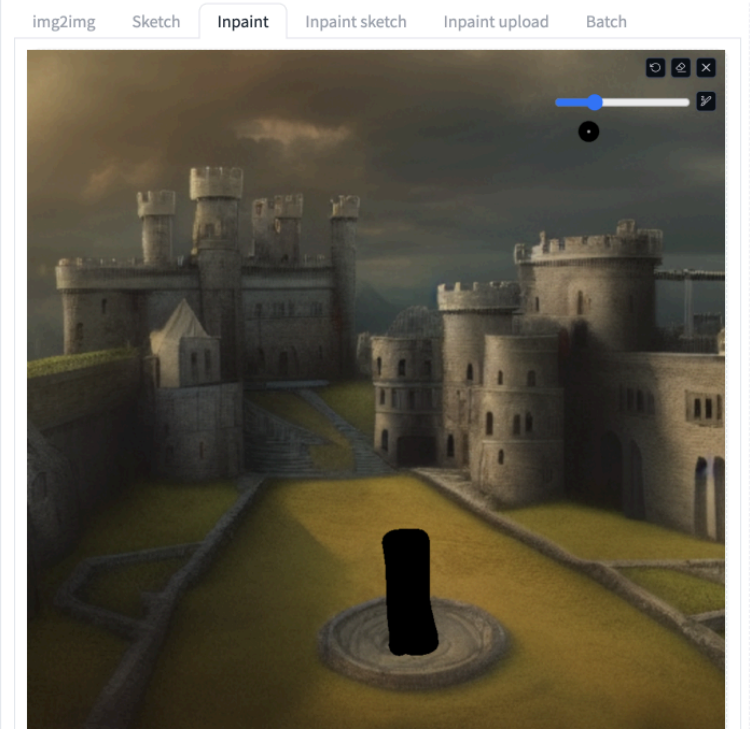

Figure 9-15. Inpainting canvas in Img2Img

The advice typically given for DALL-E, which also supports Inpainting, is to use your prompt to describe the entire image, not just the inpainted area. This is a good default and should be tried first. Make sure Inpaint area is set to *Whole picture* rather than *Only masked*, or it'll try and fit the whole scene in the masked area (don't worry, even if you select *Whole picture*, it will only paint in your masked area). It can also help to carry over your Seed from the original image if it was AI generated. However, adding to or changing the prompt to include specifics about the region you want modified or fixed tends to get better results in our experience. At the very least you should change the subject of the prompt; for example, in Figure 9-15 the prompt changed from `castle` to `statue` because that's what we wanted to appear in the courtyard. You can also try only prompting for the infilled region, though that risks getting an image that isn't globally consistent in style:

Input:

```
statue of a king, texture, intricate, details, highly
detailed, masterpiece, architecture, building, trending on
artstation, focus, sharp focus, concept art, digital
painting, fantasy, sunny, day, midday, in the style of
high fantasy art
```

The output is shown in Figure 9-16.

Figure 9-16. Inpainting to add a statue to an image

PROVIDING DIRECTION

Inpainting is so powerful because it gives you control. The ability to isolate an individual part of an image and give detailed directions on how to fix it gives you a more efficient workflow, without affecting the rest of the image.

If it's a small adjustment to the inpainted area, use Original as the masked content option and use a Denoising Strength of 0.2 to 0.4. If you're totally replacing an element of the image you may need the Latent Noise option and as high as 0.8 for Denoising Strength, though any time you get above 0.4 you start to see globally inconsistent elements and hallucinations in the image, so it can take time to iterate towards something that works. The Fill option is also useful as it matches the colors of the surrounding area. If you're getting ugly seams at the edge of the Inpainting area you can increase the Mask Blur, but typically the default of 4 works well. Inpainting is an iterative process. We recommend working on fixing one issue or artifact at a time, applying it as many times as you want, experimenting with different parameters, until you're satisfied with the final image.

Outpainting doesn't work the same as in DALL-E (see Chapter 8), which has the ability to add new generation frames to an infinite canvas. Instead in AUTOMATIC1111, outpainting is implemented by scrolling down to the Script dropdown and selecting "Poor man's outpainting". You need to set the Resize mode to Resize and fill in the Img2Img Inpaint tab, and set a relatively high Denoising Strength to make this work. This extension allows you to expand the pixels on different sides of the image, while setting the Masked Content and Mask Blur parameters as usual for these gaps on the side to be inpainted:

The output is shown in Figure 9-17.