Chapter 9. Advanced Techniques for Image Generation with Stable Diffusion

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 following 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 pretrained 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 multiply 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.

Figure 9-17. Outpainting in Img2Img

As you can see in Figure 9-16, with the extra castle being added to the sky, the potential for hallucination is high and the quality can be low. It often takes a lot of experimentation and iteration to get this process right. This is a similar technique to how early adopters of generative AI would add extra empty space on the sides of photos in photoshop, before inpainting them to match the rest of the image in Stable Diffusion. This technique is essentially just inpainting with extra steps, so all of the same advice listed above applies. This can be quicker than using the outpainting functionality in AUTOMATIC1111 because of the poor quality and limitations of not having a proper canvas.

ControlNet

Using prompting and Img2Img or base images it's possible to control the style of an image, but often the pose of people in the image, composition of the scene, or structure of the objects will differ greatly in the final image. ControlNet is an advanced way of conditioning input images for image generation models like Stable Diffusion. It allows you to gain more control over the final image generated through various techniques like edge detection, pose, depth, and many more. You upload an image you want to emulate, and use one of the pre-trained model options for processing the image to input alongside your prompt, resulting in a matching image composition with a different style (Figure 9-18, from the ControlNet paper).

Figure 9-18. ControlNet Stable Diffusion with Canny edge map

What's referred to as ControlNet is really a series of open-source models released following the paper "Adding Conditional Control to Text-to-Image Diffusion Models" (Zhang and Agrawala, 2023). While it is possible to code this in Python and build your own user interface for it, the quickest and easiest way to to get up and running is via the ControlNet extension for AUTOMATIC1111. As of the time of writing, not all ControlNet methods are available for SDXL, so we are using Stable Diffusion v1.5 (make sure you use a ControlNet model that matches the version of Stable Diffusion you're using). You can install the extension following these instructions:

1. Navigate to the Extensions tab and click the sub tab labelled Available
2. Click the *Load from* button.
3. In the Search box type `sd-webui-controlnet` to find the Extension.
4. Click Install in the Action column to the far right.
5. Web UI will now download the necessary files and install ControlNet on your local version of Stable Diffusion.

If you have trouble executing the preceeding steps you can try the following alternate method:

1. Navigate to the Extensions tab and click Install from URL sub tab.
2. In the URL field for the Git repository, paste the link to the extension: *https://github.com/Mikubill/sd-webui-controlnet*
3. Click Install.
4. WebUI will download and install the necessary files for ControlNet.

Now that you have ControlNet installed, restart AUTOMATIC1111 from your terminal or command line, or visit Settings and click "Apply and restart UI".

The extension will appear below the normal parameter options you get for Stable Diffusion, in an accordion tab (Figure 9-18). You first upload an image, then click enable before selecting the ControlNet preprocessor and model you wish to use. If your system has less than 6 GB of VRAM (Video Random Access Memory), you should check the Low VRAM box. Depending on the task at hand you might want to experiment with a number of models, and make adjustments to the parameters of those models, in order to see which gets results.

Figure 9-19. ControlNet extension interface in AUTOMATIC1111

Control Weight is analogous to prompt weight or influence, similar to putting words in brackets with a weighting `(prompt words: 1.2)`, but for the ControlNet input. The Starting Control Steps and Ending Control Steps are when in the diffusion process the ControlNet applies, by default from start to finish (0 to 1), akin to prompt editing / shifting such as `[prompt words::0.8]` (apply this part of the prompt from the beginning until 80% of the total steps are complete). Because the image diffuses from larger elements down to finer details, you can achieve different results by controlling where in that process the ControlNet applies; for example; removing the last 20% of steps (Ending Control Step = 0.8) may allow the model more creativity when filling in finer detail. The Preprocessor Resolution also helps maintain control here, determining how much fine detail there is in the intermediate image processing step. Some models have their own unique parameters, such as the Canny Low and High Thresholds, which determine what pixels constitute an *edge*. Finally the Control Mode determines how much the model follows the ControlNet input relative to your prompt.

When you first install ControlNet you won't have any models downloaded, so for them to populate in the dropdown you should install them by downloading them from the models page, and then dropping them in the *Models* > *ControlNet* folder. If you're unsure of which model to try, start with Canny edge detection as it is the most generally useful. Each model is relatively large (in the order of a few gigabytes) so only download the ones you plan to use. Below are examples from some of the more common models. All images in this section are generated with the `DPM++`

`SDE Karras` sampler, a CFG scale of 1.5, Control Mode set to Balanced, Resize Mode set to Crop and Resize (the uploaded image is cropped to match the dimensions of the generated image, 512 x 512), and 30 sampling steps, with the default settings for each ControlNet model. Version 1.5 of Stable Diffusion was used as not all of these ControlNet models are available for Stable Diffusion XL at the time of writing, but the techniques should be transferrable between models.

Canny edge detection creates simple, sharp pixel outlines around areas of high contract. It can be very detailed and give excellent results, but can also pick up unwanted noise and give too much control of the image to ControlNet. In images where there is a high degree of detail that needs to be transferred to a new image with a different style, Canny excels and should be used as the default option. For example, redrawing a city skyline in a specific style works very well with the Canny model, as we did with an image of New York City (by Robert Bye on Unsplash) in Figure 8-17:

Input:

```
new york city by studio ghibli
```

The output is shown in Figure 9-20.

Figure 9-20. ControlNet Canny

Sometimes in traditional `img2img` prompting, some elements of an image get confused or merged, because Stable Diffusion doesn't understand the depth of those objects in relation to each other. The Depth model creates a depth map estimation based on the image, which provides control over the composition and spatial position of image elements. If you're not familiar with depth maps, whiter areas are closer to the viewer, and blacker are further away. This can be seen in Figure 8-18, where an image of a band (by Hans Vivek on Unsplash) is turned into an image of soldiers with the same positions and depth of field:

Input:

```
us military unit on patrol in afghanistan
```

The output is shown in Figure 9-21.

Figure 9-21. ControlNet Depth

The Normal model creates a mapping estimation that functions as a 3D model of objects in the image. The colors red, green, and blue are used by 3D programs to determine how smooth or bumpy an object is, with each color corresponding to a direction (left/right, up/down, close/far). This is just an estimation, however, so it can have unintended consequences in some cases. This method tends to excel if you need more textures and lighting to be taken into consideration, but can sometimes offer too much detail in the case of faces, constraining the creativity of the output. In Figure 9-21 a woman playing a keyboard (by Soundtrap on Unsplash) is transported back in time to the Great Gatsby era:

Input:

```
woman playing piano at a great gatsby flapper party, 1920s,
symmetrical face
```

The output is shown in Figure 9-22.

Figure 9-22. ControlNet Normal

The OpenPose method creates a skeleton for a figure by determining its posture, hand placement and facial expression. For this model to work you typically need to have a human subject with the full body visible, though there are portrait options. It is very common practice to use multiple OpenPose skeletons and compose them together into a single image, if multiple people are required in the scene. Figure 9-23 transposes the Mona Lisa's pose onto an image of Rachel Weisz:

Input:

```
painting of Rachel Weisz
```

The output is shown in Figure 9-23.

Figure 9-23. ControlNet OpenPose

The MLSD (Mobile Line Segment Detection) technique is quite often used in architecture and interior design, as it's well suited to tracing straight lines. Straight lines tend only to appear in man-made objects, so it isn't well suited to nature scenes (though it might create an interesting effect). Man made objects like houses are well suited to this approach, as seen in the image of a modern apartment (by Collov Home Design on Unsplash) reimagined for the *Mad Men* era, in Figure 9-24:

Input:

```
1960s mad men style apartment
```

The output is shown in Figure 9-24.

Figure 9-24. ControlNet MLSD

The SoftEdge technique, also known as HED (Holistically-nested Edge Detection), is an alternative to Canny edge detection, creating smoother outlines around objects. It is very commonly used and provides good detail like Canny, but can be less noisy and deliver more aesthetically pleasing results. This method is great for stylizing and recoloring images, and it tends to allow for better manipulation of faces compared to Canny. Thanks to ControlNet, you don't need to enter too much of a detailed prompt of the overall image, and can just prompt for the change you want to see. Figure 9-25 shows a reimagining of Vermeer's Girl with a Pearl Earring, with Scarlett Johansson:

Input:

```
scarlett johansson, best quality, extremely detailed
```

Negative:

```
monochrome, lowres, bad anatomy, worst quality, low quality
```

The output is shown in Figure 9-25.

Figure 9-25. ControlNet SoftEdge

Another popular technique for architecture is segmentation, which divides the image into related areas or segments that are somethat related to one another. It is roughly analogous to using an image mask in Img2Img, except with better results. Segmentation can be used in situations where you require greater command over various objects within an image. One powerful use case is on outdoor scenes, which can vary for the time of day and surroundings, or even the era. For example, take a look at Figure 9-26, showing modern-day photograph of a castle (by Richard Clark on Unsplash), turned into a fantasy-style castle illustration:

Input:

```
A beautiful magical castle viewed from the outside, 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 9-26.

Figure 9-26. ControlNet Segmentation

One powerful feature is the ability to draw on a canvas and use that in ControlNet. You can also draw offline and take a picture to upload your image, but it can be quicker for simple images to click on the pencil emoji in the Stable Diffusion web UI, and draw with the provided brush. Even a simple scribble is often sufficient, and the edges don't have to be perfect, as shown in Figure 8-26:

Input:

```
the happy goldfish, illustrated children's book
```

The output is shown in Figure 9-27.

Figure 9-27. ControlNet Scribble

PROVIDE EXAMPLES

ControlNet gives an AI artist the ability to make an image that *looks like* another image in terms of composition, simply by providing an example image to emulate. This allows more control over visual consistency and more flexibility in making more sophisticated images.

Each of these ControlNet methods has its own preprocessor, and they must match the model for the image to make sense. For example, if you're using a Canny preprocessor, you should use a Canny model like `control_v11p_sd15_canny`. It's also important to choose a model that gives enough freedom for the task you're trying to accomplish; for example, an image of a cat with the SoftEdge model might perhaps have too much detail to be turned into a lion, and you might want to try something less fine-grained. As with all things Stable Diffusion, finding the exact combination of model and parameters requires experimentation, with new functionality and options proliferating all the time.

ControlNet supports being run with a simple prompt or even without a prompt at all. It will match the existing image you submit and ensure a high level of consistency. You can run a generic prompt like `a professional, detailed, high-quality image` and get a good version of the existing image. Most often however you'll be attempting to change certain aspects of the image and will want to input a full prompt, as in the examples above. The resulting image will match both the prompt and the ControlNet output, and you can experiment with adjusting the parameters available to see what gets results.

Segment Anything Model (SAM)

When working on an AI-generated image, it is often beneficial to be able to separate out a *mask* representing a specific person, object, or element. For example, dividing an image of a person from the background of the image would allow you to inpaint a new background behind that person. This can take a long time and lead to mistakes when using a brush tool, so it can be helpful to be able to automatically segment the image based on an AI model's interpretation of where the lines are.

The most popular and powerful model for doing this is *SAM*, which stands for Segment Anything Model, released open-source on GitHub by Meta. The model is trained on a dataset of 11 million images and 1.1 billion masks, and is able to infer where the image mask should be based on user input (clicking to add 1-3 dots to the image where masks should be) or it can automatically mask all the elements individually in an image. These masks can then be exported for use in inpainting, ControlNet, or as base images.

You can use SAM in the AUTOMATIC1111 interface using the sd-webui-segment-anything extension. Once AUTOMATIC1111 is installed and running, you can install the SAM extension following these instructions:

1. Navigate to the Extensions tab and click the sub tab labelled Available.
2. Click the *Load from* button.
3. In the Search box type in: `sd-webui-segment-anything` to find the Extension.
4. Click the Install in the Action column to the far right.
5. WebUI will now download the necessary files and install SAM on your local version of Stable Diffusion.

If you have trouble executing the preceding steps you can try the following alternate method:

1. Navigate to the Extensions tab and click the Install from URL sub tab.
2. In the URL field for the git repository, paste the link to the extension: *https://github.com/continue-revolution/sd-webui-segment-anything*.
3. Click Install.
4. WebUI will download and install the necessary files for SAM on your local version of Stable Diffusion.

You also need to download the actual SAM model weights, linked to from the repository. The 1.25 GB `sam_vit_l_0b3195.pth` is what's being used in this chapter. If you encounter issues with low VRAM (your computer freezes or lags), you should switch to smaller models. Move the model you downloaded into the *stable-diffusion-webui/sd-webui-segment-anything/models/sam* folder.

Now that you have SAM fully installed, restart AUTOMATIC1111 from your terminal or command line, or visit Settings and click "Apply and restart UI".

You should see the extension in the Img2Img tab, by scrolling down past the canvas and Seed parameter, in an accordion component alongside the ControlNet extension. Upload an image here (we used the photo for Figure 9-28 by Luca Baini on Unsplash) and click the image to select individual prompt points. These prompt points go along to SAM as user input, to help the model determine what should be segmented out from the image. You can click Preview to see what mask will be created, and iterately add or remove plot points until the mask is correct. There is a checkbox labelled *Preview automatically when add/remove points*, which updates the mask with each click. Often SAM gets it right with a single plot point, but if you are struggling you can also add negative plot points to parts of the image you don't want to mask by right clicking. Select the mask you want (Figure 9-28) from the three options provided (counting from 0 to 2):

Figure 9-28. Adding plot points

When your mask is ready, make sure the box for Copy to Inpaint Upload & img2img ControlNet Inpainting is checked, and click the Switch to Inpaint Upload button. You won't see anything happen visually, but when you switch to the Inpainting tab you should be able to generate your prompt with the mask generated by SAM. There is no need to upload the picture or mask to the Inpainting tab. You can also download your mask for later upload in the "Inpaint upload" tab. This method was unreliable during our testing, and there may be a better supported method for in-painting with SAM and Stable Diffusion made available.

DIVIDE LABOR

Generative models like Midjourney and Stable Diffusion are powerful, but they can't do everything. In training a separate image segmentation model, Meta has made it possible to generate more complex images by splitting out the elements of an image into different masks, which can be worked on separately before being aggregated together for the final product.

Dreambooth Fine-Tuning

The original Stable Diffusion model cost a reported $600,000 to train using a total of 150,000 GPU hours, so training your own foundational model is likely out of the question for most organizations. However, it is possible build on top of Stable Diffusion, using the Dreambooth technique, which was introduced in the paper "DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation" (Ruiz et al, 2022). Dreambooth allows you to fine-tune or train the model to understand a new concept it hasn't encountered yet in its training data. Not having to start from scratch to build a new model means significantly less time and resource: about 45 minutes to an hour on 1 GPU. Dreambooth actually updates the weights of the new model, which gives you a new 2 GB model file to use in AUTOMATIC1111 instead of the base Stable Diffusion model.

There are many Dreambooth based models available on websites like HuggingFace and Civitai. In order to use these models in AUTOMATIC1111, you simply download them and move them into the *stable-diffusion-webui/models/Stable-diffusion/* folder. Dreambooth models often have a specific word or token needed for triggering the style or subject, which must be included in the prompt. For example, the Inkpunk Diffusion model requires the word `nvinkpunk`. Note: the underlying base model here is v1.5 of Stable Diffusion, so reset your image size to 512 x 512:

Input:

```
skateboarding in times square nvinkpunk
```

The output is shown in Figure 9-29.

Figure 9-29. InkPunk skateboarder

DIVIDE LABOR

The mistake many people make with AI is assuming there's one model to rule them all. In reality there are many creative models out there, and often training on a specific task yields better results than the general foundational models. While the foundation models like Stable Diffusion XL are what most practicioners start with, commonly they begin to experiment with fine-tuning their own models on specific tasks, often based on smaller, more efficient models like v1.5.

The preferred method for training a Dreambooth model is Shivam Shriao's repository, which uses HuggingFace's `diffusers` library. What follows is an explanation of the code in Google Colab. Version 1.5 is used in this notebook, in this as it is a smaller model, and is able to be trained in a few hours in the Google Colab environment for free. A copy of this Python notebook is saved in the GitHub repository for the book for posterity, but it should be noted that it will only run on an NVidia GPU, not on a MacBook.

First the colab checks whether there is access to an Nvidia GPU. This is one good reason to run Dreambooth on Google Colab, because you are given access to the right resource to run the code without any configuration needed:

```
!nvidia-smi --query-gpu=name,memory.total, \
    memory.free --format=csv,noheader
```

Next the necessary libraries are installed, including the `diffusers` library from HuggingFace:

```
!wget -q https://github.com/ShivamShrirao/diffusers/raw/ \
    main/examples/dreambooth/train_dreambooth.py
!wget -q https://github.com/ShivamShrirao/diffusers/raw/ \
    main/scripts/convert_diffusers_to_original_stable_ \
    diffusion.py
%pip install -qq \
git+https://github.com/ShivamShrirao/diffusers
%pip install -q -U --pre triton
%pip install -q accelerate transformers ftfy \
bitsandbytes==0.35.0 gradio natsort safetensors xformers
```

Run the next cell to set the output directory of the model when it is finished running. It's recommended to save the model to Google Drive (even if temporarily) because you can more reliably download large files (4-5 GB) from there, than you can from the Google Colab filesystem. Ensure that you have selected the right base model from the HuggingFace hub `runwayml/stable-diffusion-v1-5` and choose a name for your token for the output directory (usually *ukj* or *zwx*, more on this later):

```
#@markdown If model weights should be saved directly in
#@markdown google drive (takes around 4-5 GB).
save_to_gdrive = False
if save_to_gdrive:
    from google.colab import drive
    drive.mount('/content/drive')

#@markdown Name/Path of the initial model.
MODEL_NAME = "runwayml/stable-diffusion-v1-5" \
    #@param {type:"string"}

#@markdown Enter the directory name to save model at.

OUTPUT_DIR = "stable_diffusion_weights/ukj" \
    #@param {type:"string"}
if save_to_gdrive:
    OUTPUT_DIR = "/content/drive/MyDrive/" + OUTPUT_DIR
else:
    OUTPUT_DIR = "/content/" + OUTPUT_DIR

print(f"[*] Weights will be saved at {OUTPUT_DIR}")

!mkdir -p $OUTPUT_DIR
```

Before training, you need to add the concepts you want to train on. In our experience, training on multiple concepts tends to harm performance, so typically we would train on only one subject or style. You can merge models later in the Checkpoint Merger tab of AUTOMATIC1111, although this gets into more advanced territory not covered in this book. The instance prompt includes the token you'll use in your prompt to trigger the model, and ideally it's a word that doesn't have any other meaning, like `zwx` or `ukj`. The class prompt is a starting point for the training, so if you're training a model of a specific person, you start from `photo of a person` to make the training more effective:

```
# You can also add multiple concepts here.
# Try tweaking `--max_train_steps` accordingly.

concepts_list = [
     {
         "instance_prompt":      "photo of ukj person",
         "class_prompt":         "photo of a person",
         "instance_data_dir":    "/content/data/ukj",
         "class_data_dir":       "/content/data/person"
     }
]

# `class_data_dir` contains regularization images
import json
import os
for c in concepts_list:
    os.makedirs(c["instance_data_dir"], exist_ok=True)

with open("concepts_list.json", "w") as f:
    json.dump(concepts_list, f, indent=4)
```

Next, we upload the images through Google Colab. Dreambooth can work with as few as five images, but typically it's recommended you use about 20-30 images, although some train with hundreds of images. One creative use case is to use the Consistent Characters method discussed in Chapter 8 to generate 20 different images of the same AI-generated character, and use those to train a Dreambooth model on. Alternatively you could upload 20 pictures of yourself to create an AI profile photo, or 20 pictures of a product your company sells to generate AI product photography. You can upload the files locally to the *instance_data_dir* in the Google Colab filesystem (which can be faster), or run the next cell to get an upload button:

```
import os
from google.colab import files
import shutil

for c in concepts_list:
    print(f"""Uploading instance images for
`{c['instance_prompt']}`""")
    uploaded = files.upload()
    for filename in uploaded.keys():
        dst_path = os.path.join(c['instance_data_dir'],
            filename)
        shutil.move(filename, dst_path)
```

Now the actual training begins! This code runs on the GPU and outputs the final weights when finished. Make sure to change `save_sample_prompt` before running to use the token you assigned, in this case `photo of ukj person`:

```
!python3 train_dreambooth.py \
  --pretrained_model_name_or_path=$MODEL_NAME \
  --pretrained_vae_name_or_path="stabilityai/sd-vae-ft-mse" \
  --output_dir=$OUTPUT_DIR \
  --revision="fp16" \
  --with_prior_preservation --prior_loss_weight=1.0 \
  --seed=1337 \
  --resolution=512 \
  --train_batch_size=1 \
  --train_text_encoder \
  --mixed_precision="fp16" \
  --use_8bit_adam \
  --gradient_accumulation_steps=1 \
  --learning_rate=1e-6 \
  --lr_scheduler="constant" \
  --lr_warmup_steps=0 \
  --num_class_images=50 \
  --sample_batch_size=4 \
  --max_train_steps=800 \
  --save_interval=10000 \
  --save_sample_prompt="photo of ukj person" \
  --concepts_list="concepts_list.json"
```

Now that the training is complete, the next two cells of code define the directory and then display a grid of images so you can see visually whether the model correctly understood your concept, and is now capable of generating useful images of your style of subject:

```
WEIGHTS_DIR = ""
if WEIGHTS_DIR == "":
    from natsort import natsorted
    from glob import glob
    import os
    WEIGHTS_DIR = natsorted(glob(OUTPUT_DIR + os.sep + \
        "*"))[-1]
print(f"[*] WEIGHTS_DIR={WEIGHTS_DIR}")

#@markdown Run to generate a grid of preview images from the last saved weights.
import os
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

weights_folder = OUTPUT_DIR
folders = sorted([f for f in os.listdir(weights_folder) \
    if f != "0"], key=lambda x: int(x))

row = len(folders)
col = len(os.listdir(os.path.join(weights_folder,
    folders[0], "samples")))
scale = 4
fig, axes = plt.subplots(row, col, figsize=(col*scale,
    row*scale), gridspec_kw={'hspace': 0, 'wspace': 0})

for i, folder in enumerate(folders):
    folder_path = os.path.join(weights_folder, folder)
    image_folder = os.path.join(folder_path, "samples")
    images = [f for f in os.listdir(image_folder)]
```

```
        for j, image in enumerate(images):
            if row == 1:
                currAxes = axes[j]
            else:
                currAxes = axes[i, j]
            if i == 0:
                currAxes.set_title(f"Image {j}")
            if j == 0:
                currAxes.text(-0.1, 0.5, folder, rotation=0,
                va='center', ha='center',
                transform=currAxes.transAxes)
            image_path = os.path.join(image_folder, image)
            img = mpimg.imread(image_path)
            currAxes.imshow(img, cmap='gray')
            currAxes.axis('off')

plt.tight_layout()
plt.savefig('grid.png', dpi=72)
```

Finally, you want to run the conversion process to get a *.ckpt* file, which is what you will use in AUTOMATIC1111:

```
#@markdown Run conversion.
ckpt_path = WEIGHTS_DIR + "/model.ckpt"

half_arg = ""
#@markdown Convert to fp16, takes half the space (2GB).
fp16 = True #@param {type: "boolean"}
if fp16:
    half_arg = "--half"
!python convert_diffusers_to_original_stable_diffusion.py \
    --model_path $WEIGHTS_DIR  --checkpoint_path \
    $ckpt_path $half_arg
print(f"[*] Converted ckpt saved at {ckpt_path}")
```

You can then visit the weights directory *stable_diffusion_weights/zwx* to find the model, and download it. If you are having issues downloading such a large file from the Google Colab filesystem, try checking the option to save to Google Drive before running the model, and download from there. We recommend renaming the model before dropping it into your *stable-diffusion-webui/models/Stable-diffusion/* folder, so you can tell what model it is when using it later:

Input:

```
a professional headshot of ukj person, standing with his
arms crossed and smiling at the camera with his arms
crossed, a character portrait, Adam Bruce Thomson, private
press, professional photo
```

The output is shown in Figure 9-30.

Figure 9-30. A Dreambooth model image of one of the authors

There is also an extension for training Dreambooth models via Automatic1111, based on Shivam Shriao's method. This extension can be installed in the same way as you installed ControlNet and Segment Anything in previous sections of this chapter. This tool is for advanced users as it exposes a significant number of features and settings for experimentation, many of which you need to be a machine learning expert to understand. To start learning what these parameters and settings mean so you can experiment with different options, check out the beginner's guide to training in the extension wiki. The benefit of using this method instead of Google Colab is that it runs locally on your computer, so you can leave it running without worrying it will time out and lose progress.

PROVIDE EXAMPLES

Dreambooth helps you personalize your experience with generative AI. You just need to supply 5 to 30 images that serve as examples of a concept, and less than an hour of training time, and you can have a fully personalized custom model.

There are other training and fine-tuning methods available besides Dreambooth, but this technique is currently the most commonly used. An older technique is Textual Inversion, which doesn't update the model weights, but instead approximates the right location for a token to represent your concept, though this tends to perform far worse than Dreambooth. One promising new technique is LoRA, from the paper "LoRA: Low-Rank Adaptation of Large Language Models" (Hu et al, 2021), also prevalent in the text-generation space with LLMs. This technique adds new layers into the model, and trains just those new layers to build a custom model without expending too much resource. There are also Hypernetworks, which train parameters that can then generate these new layers, as introduced by Kurumuz in the Medium article "NovelAI Improvements on Stable Diffusion". Both of these methods are experimental and only make up a small number of the models on Civitai at the time of writing (less than 10%), as well as having in general lower user ratings in terms of quality.

Stable Diffusion XL Refiner

The SDXL v1.0 model has 6.6 billion parameters, compared to 0.98 billion for the v1.5 model (Rombach et al, 2023). The increased firepower yields impressive results, and as such the model is starting to win over die-hard 1.5 enthusiasts. Part of the power of SDXL comes from the division of labor between the base model, which sets the global composition, and a refiner model (Figure 9-31) which adds finer details (optional).

Figure 9-31. Stable Diffusion XL base and refiner model

The underlying language model that infers meaning from your prompts is a combination of OpenClip (ViT-G/14) and OpenAI's CLIP ViT-L. Stable Diffusion v2 used OpenClip alone, and therefore prompts that worked on v1.5 were not as transferable: that problem has been largely solved with SDXL. Additionally, the SDXL model has been trained a more diverse set of image sizes, leading to better results when you need an image that isn't the standard square aspect ratio. Stablity AI's research indicates that users overwhelmingly prefer the XL model over v1.5 (Figure 9-32).

Figure 9-32. Relative performance preference

To make use of the refiner model, you must utilize the "Switch at" functionality in the AUTOMATIC1111 interface. This value controls at which step the pipeline switches to the refiner model. For example, switching at 0.6 with 30 steps means the base model will be used for the first 18 steps, and then it will switch to the refiner model for the final 12 steps (Figure 9-33).

Figure 9-33. Refiner - Switch at parameter

Common advice is to switch between 0.4 and 1.0 (a value of 1.0 will not switch, and only uses the base model), with 20-50 sampling steps for the best results. In our experience, switching at 0.6 with 30 sampling steps produces the highest quality image, but like all things Stable Diffusion, you must experiment to discover what gets the best results for your image. Setting the refiner to switch at 0.6 gives the output shown in Figure 9-35):

Input:

```
anime cat girl with pink hair and a cat ears outfit is posing for a picture
in front of a gaze, photorealistic, 1girl, a character portrait, floral print,
Alice Prin, sots art, official art, sunlight, wavy hair, looking at viewer
```

Negative:

```
disfigured, ugly, bad, immature, photo, amateur, overexposed, underexposed
```

The output is shown in Figure 9-34.

Figure 9-34. Anime cat girl with SDXL base model vs refiner at 0.6

DIVIDE LABOR

The architecture of SDXL is a perfect example of splitting a task into multiple jobs, and using the right model for the job. The base model sets the scene and guides the composition of the image, while the refiner increases fine detail.

One quality-of-life modification you can make is to install the aspect ratio selector extension, which can be loaded with image sizes or aspect ratios you use regularly, allowing 1-click setting of the correct size and aspect ratio for either model.

To install the extension, browse to the Extensions tab, go to “Install from URL”, paste in *https://github.com/alemelis/sd-webui-ar*, and click Install. Go to the extension folder *stable-diffusion-webui/extensions/sd-webui-ar* and add the following to the *resolutions.txt* file (or replace what’s there for cleanliness):

```
SD1:1, 512, 512 # 1:1 square
XL1:1, 1024, 1024 # 1:1 square
SD3:2, 768, 512 # 3:2 landscape
XL3:2, 1216, 832 # 3:2 landscape
SD9:16, 403, 716 # 9:16 portrait
XL9:16, 768, 1344 # 9:16 portrait
```

Clicking one of these preset buttons will automatically adjust the Width and Height accordingly. You may also replace the *aspect ratios.txt* with the following, allowing you to automatically calculate the aspect ratio based on the height value you have set in the webui and they’ll show in the webui interface (Figure 9-35):

```
Square 1:1, 1.0 # 1:1 ratio based on minimum dimension
Landscape 3:2, 3/2 # Set width based on 3:2 ratio to height
Portrait 9:16, 9/16 # Set width based on 9:16 ratio to height
```

Figure 9-35. Aspect ratios

Summary

In this chapter you have learned advanced techniques for image generation using Stable Diffusion, an open-source model. You have successfully installed Stable Diffusion and built an inference pipeline using the HuggingFace `diffusers` library. By following the step-by-step explanation, you have generated images based on prompts using the Stable Diffusion Inference model in Google Colab. Additionally, the chapter recommends exploring the open-source community and user interfaces like AUTOMATIC1111 for running Stable Diffusion with advanced features.

The chapter also introduces the concept of ControlNet, which allows for controlling the style of an image using prompting and base images, and SegmentAnything, a model for masking specific parts of an image. By applying these techniques, are now able to customize generated images to meet your specific needs. You also learned about techniques for personalization, specifically Dreambooth Fine-Tuning, allowing you to train a model to understand new concepts not encountered in its training data.

In the next chapter, you'll get the chance to put everything you've learned throughout this book into action. We'll be exploring how to build an AI blog post generator that produces both the blog text and an accompanying image. That final exciting chapter will take you through the process of creating an end-to-end system that generates high-quality blog posts based on user input, complete with custom illustrations in a consistent visual style. You'll learn how to optimize prompts, generate engaging titles, and create AI-generated images that match your desired style!