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.

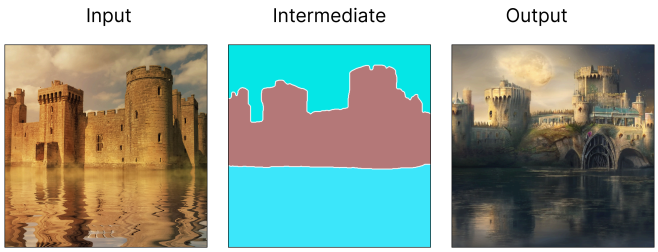

Figure 9-26. ControlNet Segmentation

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.

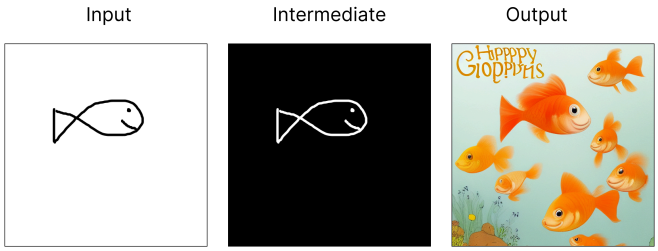

Figure 9-27. ControlNet Scribble

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 bil-

lion 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):

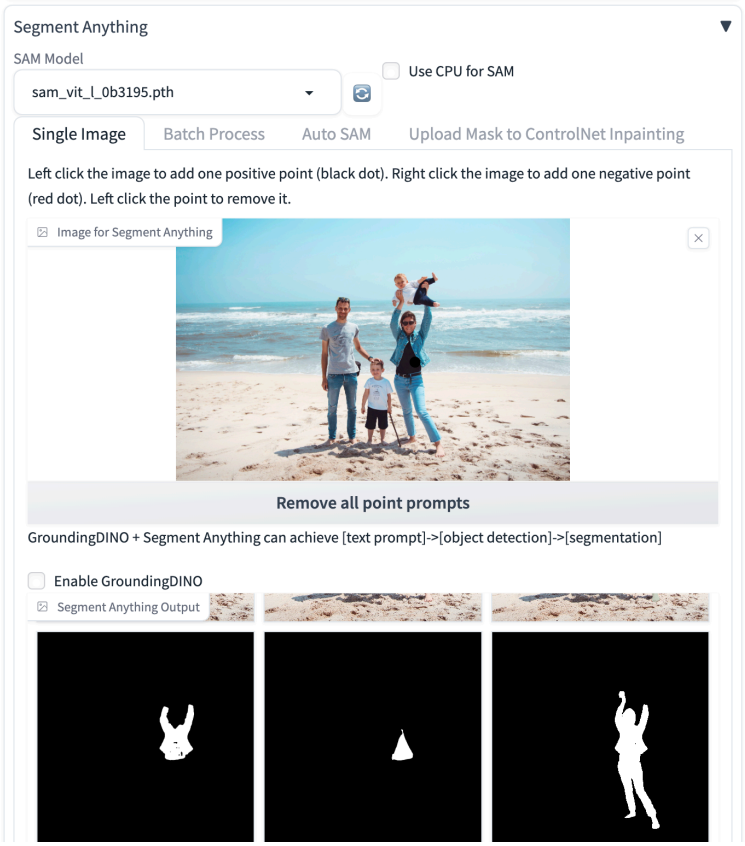

Figure 9-28. Adding plot points

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 inpainting 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 up-

dates 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 pos-

terity, 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).

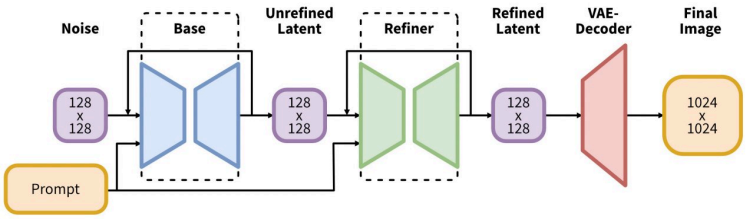

Figure 9-31. Stable Diffusion XL base and refiner model

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).

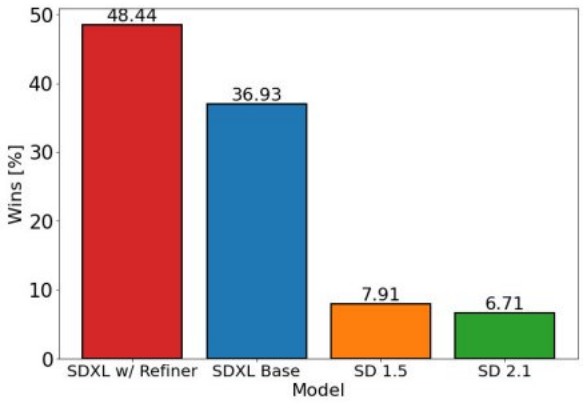

Figure 9-32. Relative performance preference

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).

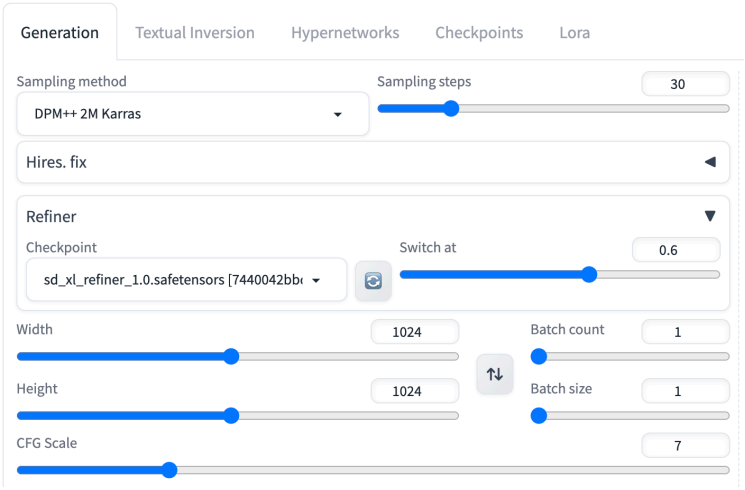

Figure 9-33. Refiner - Switch at parameter

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.

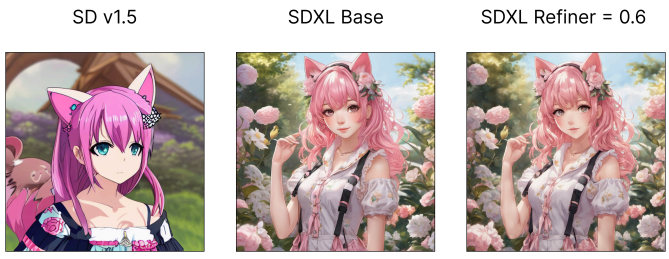

Figure 9-34. Anime cat girl with SDXL base model vs refiner at 0.6

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 personal-

ization, 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!