LIGN 167

December 9, 2022

Final Project Writeup

The versatility of image generation programs powered by large language models has led to the recent development of many varied and customized uses for such programs. In this project, we attempted to finetune one such image generation program, Stable Diffusion, by improving its capability to adjust specific sections of an image, which could be useful for image editing in commercial and artistic contexts. In particular, we wanted to see if Stable Diffusion could be used to focus adjustments to the foreground or background of an image while keeping the rest relatively unchanged. This would be an adaptation and combination of text-guided inpainting and image-to-image generation tools, similar programs which have both already been successfully implemented in Stable Diffusion. Inpainting with Stable Diffusion allows users to erase parts of an image and then fill in the spaces with content generated from a text prompt. However, this process is not very automatic: it requires the user to manually select the target areas of the image, and the output often involves lots of manual revision on the user's end because of the potential for the inpainted area to look isolated or disconnected from the rest of the image. On the other hand, Img2img with Stable Diffusion allows users to generate a new similar image from an uploaded image and a text prompt. In contrast with inpainting, this tool is very automated and produces cohesive final images, but is quite destructive of the initial image. It doesn't allow for specificity or control in the parts of the image that are being changed, and it will often result in an output image that differs greatly from the original image.

Navigating this interplay between automation and control, our goal was to engineer a tool that would have the automation and cohesion of Img2img generation while maintaining inpainting's ability to focus modifications on a target portion of the original image. With this in mind, we decided to create a tool that creates an image from three inputs: the initial image to be modified, a description of the initial image, and a description of the goal (target) image to be generated. We opted to use OpenAI's GPT-3 and API 4 AI's "Background Removal" tool to identify the region of the original image (i.e., foreground or background) that should be altered based on the image descriptions. This information would then be used to generate a custom noise pattern to input into a modified Stable Diffusion Img2img pipeline, which would generate the final image according to the target image description.

To implement this system, we broke down the pipeline into three fundamental steps. For the initial phase of our system, we used GPT-3 to identify whether to mask out the foreground or background to optimally modify the image. We framed this process as a classification problem; given a pair of {initial image description, target image description}, we wanted GPT-3 to select one of the following classes: foreground, background, or unidentified (for situations where the input descriptions were unclear). With this framing, we focused on a zero-shot prompting technique, hoping to draw on GPT-3's conceptual understanding of scenes to identify foreground and background rather than forcing it to re-learn this dichotomy with flat, prompted examples. We then combined the initial and target image descriptions with our engineered prompt and generated a completion from GPT-3 using the OpenAI python library. In the case that GPT-3 was unable to identify whether the foreground or background should be changed, this step raised an error, giving the user a chance to re-consider their image descriptions or manually set a default behavior. We left the handling of this option open because the best course of action may be very context-dependent.

Next, as described above, we used an online, API-accessible background removal tool to generate a mask over the input image. Like with GPT-3, the heavy lifting for the computational aspect of this task (i.e. edge detection and subject identification) was mostly handled in that external tool, which we accessed using an API, so this part of our project is relatively light on code. After some modifications, the resulting mask image identifies pixels in the region of the image to be modified as black and the rest of the image (where noise will not be applied) as white.

Finally, we implemented a modified version of the Stable Diffusion Img2img pipeline. As described by Huggingface, the standard Stable Diffusion pipeline works by generating Gaussian noise into a 64x64 “latent space” (i.e., a compressed vector representation of an image). It takes this random noise and the input prompt, and iteratively changes—or “denoises”—the random noise toward an image that closely matches the text prompt. The final step of the image generation takes these denoised latent vectors and converts them into an image using a VAE decoder. For the purposes of our project, we focused on altering the initial noise pattern that Stable Diffusion starts from, with the idea that denoising latent vectors that have information from the initial image encoded into them should produce a final image that is similar to the initial image.

Similar to the standard Stable Diffusion pipeline, our modified Stable Diffusion pipeline involved adding noise to the initial image and then running the de-noising function to iteratively remove noise from the image, using the target image description to guide the process. We implemented a lot of this process on our end, but we drew heavily from the standard pipeline for structure and guidance with the more Stable Diffusion-specific steps, like denoising, which we were not interested in modifying. To this end, we used Stable Diffusion's pre-trained K-LMS Scheduler (and accompanying documentation) to coordinate a lot of the pipeline steps and process the actual de-noising. We also used the Variational Autoencoder (VAE) and U-Net encoder-decoder models from Stable Diffusion for handling image latent vectors. Using these pre-established tools for latent vector handling and denoising allowed us to reduce the number of moving pieces in our project and focus specifically on the effect of our custom noise function, which was our primary element of study for this project.

The first step in this modified pipeline was to generate a random noise pattern and apply it to the initial image using the previously-generated mask. We used torch.randn to generate random latent vectors to represent the original noise pattern. After converting the noise vectors to an image using a VAE encoder-decoder model (as done in this Google Colab, which explores using Stable Diffusion for image compression), we applied our foreground-background mask to the noise image and combined the initial image with the masked noise image using the built-in add_noise function in the K-LMS Scheduler. This noise-applied image was then re-combined with the initial, noise-free image, using the Python Image Library (PIL) composite method, to allow for an adjustable amount of transparency in the noise. We tested a variety of different methods for generating the original noise pattern, like using, using the PIL effect_noise function to generate a gaussian noise image, and randomly generating each individual pixel in the masked image, but we found that the latent vector + scheduler.add_noise method allowed us the largest amount of control over the noise strength, and it resulted in the best performance on the de-noising step. This method is likely most compatible with the de-noising algorithm because it leverages the same scheduler, which was all trained together.

After applying our noise function to the initial image, we used the K-LMS scheduler to iteratively de-noise the image. As stated above, our implementation of this process drew heavily from the normal Stable Diffusion's de-noising process, as documented in this Google Colab by Huggingface and the accompanying blog post. We also directly referenced the code for Stable Diffusion's actual Img2img implementation, as posted on GitHub. Essentially, this process involved using the U-Net to predict how the current image could have been produced (i.e. by adding noise to the target image) and then using the scheduler to step backward (i.e. inferring a hypothetical "previous" version of the image, with less noise) towards the target image. The direction of these steps was guided by text embeddings generated by OpenAI's CLIPTextModel, which Stable Diffusion was trained on top of. This process resulted in a set of latent vectors that can be converted into the final output image using the VAE.

For our dataset, we were focusing on a similar use case as Stable Diffusion—i.e., pairs of images and text descriptions. For testing and developing our tool, we manually gathered public domain images from Wikimedia Commons, using the image titles/captions as the descriptions, rather than scraping data broadly from the web. This manual approach was easy to perform and kept our input images in the public domain, but it did involve some level of cherry-picking in our data. For example, we designed some of the ending modification prompts to specifically be possible within our system, focusing on image modifications that would primarily affect one region of the image that could be easily described as "foreground" or "background."

During our informal prompt engineering process for the GPT-3 element of our system, we used some manually generated image prompts to test GPT-3's performance against a human baseline. We did not formally assess GPT-3's performance on this task, but, as described above, we did build in some error handling options in the case that GPT-3 could not satisfactorily identify whether the foreground or background of the image should be changed.

During the development of our modified Stable Diffusion pipeline, we evaluated our model's output subjectively, focusing on three main characteristics: general cohesiveness of the final image, visual similarity to the initial image (where relevant), and conceptual similarity to the target image description. We used this evaluation approach to compare the noise function that we developed (using the initial image and regionally-masked noise) to a variety of other noise functions. Specifically, we compared our noise strategy to plain random gaussian noise as well as a few arbitrary noise masking techniques, like setting a rectangle at the top corner of the noise pattern to all 0, without incorporating any information from the initial image. This allowed us to focus on the specific effect of the noise pattern on image generation while setting the rest of the pipeline as constant.

Through these experiments, we found that our custom noise function performed quite poorly. Information is not preserved perfectly between encoding and decoding, and performing this process on the noise pattern (as we did on our first attempt at adding noise) appeared to transform it in a way that interfered with the denoising step. Running the de-noising process on our custom noise pattern would either result in an image containing apparently-random blobs of color in the noise-applied section and an overly-exposed version of the initial image in the non-noisy section, or it would iteratively smooth out the initial image until the result showed a few simple shapes/bands of color. Although this did evidently preserve some visual information from the initial image, the final images were extremely incoherent, and any influence of the target image description on this output was unclear because the de-noising failed so severely.

We did find that the effect of the text prompt on the final image was a little more evident as we increased the opacity of the noise to 95% (i.e. weighting the noise much more heavily than the initial image), but the resulting images were still very blurry and retained virtually no visual similarity to the initial images.

In contrast, the arbitrary noise masking techniques, which altered the random noise pattern but did not add any information from the initial image, resulted in somewhat vague but completely comprehensible images. For example, the background would sometimes be more simple, often involving smooth color gradients, but the subject was generally clear. Of course, these resulting images bore no similarity to the initial image because they had no information from it, but the resulting images were clearly related to the target prompt.

To evaluate our entire model more broadly, we planned to do a more rigorous evaluation, involving surveys covering a variety of question types, like having respondents match the output image to the text prompt, rate the similarity between the output image and the input image, and rate the aesthetic qualities of our output. For this final evaluation, we planned to compare images generated through our system to images generated from Stable Diffusion (using just the target image description) and Stable Diffusion Img2img (using the initial image and target description). This would have allowed us to put our tool's performance in the context of the pre-existing tools that we hoped to compete with. However, since our system did not produce any coherently transformed images, we found that this level of more formal evaluation would be redundant to our previous findings.

On the whole, our tool was unsuccessful in accomplishing its purpose of automatically generating images with an altered background/foreground. The experiments we conducted to determine why our tool wasn’t working were inconclusive, as changing the noise pattern fed into Stable Diffusion altered the output somewhat unpredictably. The easiest conclusion to make from our observations, then, is that Stable Diffusion simply “doesn’t work” when you give it prior image information—more particularly, that its weights as is are unfit to perform this task of generating an image from a meaningful (non-Gaussian?) noise pattern. Perhaps Stable Diffusion would need to be retrained or fine-tuned so as to optimize its model parameters toward this specific task of using modified noise patterns.

Regardless of why exactly Stable Diffusion failed to generate images in the way that we wanted it to, the guiding motivation for our project was really about exploring the effect of adjustments to Stable Diffusion’s noise pattern on its synthesized image output. In this regard, it is clear that Stable Diffusion is quite sensitive to changes in its initial noise pattern. Altering the noise pattern in any way noticeably worsened the model’s performance, causing the generated outputs to be less clear, less accurate, or sometimes completely nonsensical. Furthermore, different alterations affected the model differently. Stable Diffusion performed the worst by far when we fed it noise patterns that included image information; however, it was still mostly successful when we altered the noise more procedurally (e.g. zeroing a section or proportion of the noise). These results seem to suggest that the amount of entropy or information that the noise pattern contains plays a vital role in Stable Diffusion's performance, but this is a difficult quality to define and quantify, and more dedicated testing would be required to see exactly how the model's behavior changes with respect to this factor.

A potential avenue for further development for our system would be to take a closer look into the mechanics behind inpainting and Img2img and how they preserve information from the initial image. In our preliminary research, we were unable to determine how inpainting draws information from the surrounding portions of the image or how img2img is able to preserve some information from the initial image when adding noise.

Additionally, to further explore Stable Diffusion noise patterns, we would encourage further experiments into other noise patterns to see if they can be manipulated in a predictable and useful manner, for this selective image modification task as well as other applications, like multi-image synthesis and even computer vision tasks, like edge detection.