MinImagen: A minimal implementation of Imagen Text-to-Image Model

Prateek Sridhar Kumar, Manish Singh Saini

{ps4146, mss9238}@nyu.edu
NYU Fall 2022 Deep Learning Project
https://github.com/manish1022iiti/MinImagen

Introduction

The development of Text-to-Image deep learning models, began in the mid-2010s. With the development of Generative Adversarial Networks (GAN)[8] in 2014, these models have gained the ability to generate high-quality synthetic images among being able to generate human-like language and other things. GANs however capture less diversity than state-of-the-art likelihood-based models and are also often difficult to train and collapse without carefully selected hyperparameters and regularizers, making them difficult to scale and apply to new domains. Diffusion models are a class of likelihood-based models, which have been shown to produce high quality images while providing easy scalability, a stationary training objective, and distribution coverage[6].

In the past year, state-of-the-art text-to-image models have been introduced, based on diffusion models specifically, namely DALL-E 2 and Imagen. Imagen, which was released by Google, can generate high-quality, high-resolution images given a caption describing a scene regardless of the real-world logicality or plausibility of the scene, going beyond the previous capabilities of caption-based image generation.

In this project, we explore MinImagen[4], a minimal implementation of the Imagen[7] Text-to-image model capturing the salient features of Google's implementation. We first reproduce the model and familiarize ourselves with the key elements of the Imagen model and the role the individual elements play in its architecture. We then tweak the model and train various configurations to achieve meaningful results using Google's Conceptual Captions dataset and a movie/anime posters-plot dataset containing images and the plot as text captions.

Architecture

The MinImagen model mainly includes the predominant features from the Imagen model, which are:

- Diffusion model
- U-Net Noise prediction model (Base and Super U-Net)
- Text encoder
- Super-Resolution models (Super U-Net)

The first step in the training process involves using the diffusion model to progressively add Gaussian noise to an image in a series of timesteps. This is the forward diffusion step.

These noised images at the various timesteps are used to train the U-Net Noise prediction model, the objective of which is to predict the noise component of an image at any given timestep. This de-noising process occurs progressively as well, as the model is trained to predict and reduce the noise to obtain the image from the previous timestep in comparison to the noised image provided as input.

The U-Net noise prediction model in MinImagen is based off of the "Diffusion Models Beat GANs on Image Synthesis" paper[6] published in 2021 by OpenAI. Some changes were made to this U-Net architecture however, which include:

- Removing the global attention layer.
- Replacing the attention layers with transformer encoders.
- Placing the transformer encoders at the end of the sequence at each layer rather than in between the residual blocks in order to allow for a variable number of residual blocks.

This U-Net is a conditional model, which means it depends on captions as text inputs as well. It also depends on the timestep information as the U-Net needs information on how much noise is to be removed from the image, as the same U-Net is iteratively used for all the timesteps. Caption conditioning vectors were generated using a pre-trained T5 text encoder, and Time conditioning vectors were generated using the timestep information. These vectors were concatenated and used as input for the Noise prediction model.

To generate an image, MinImagen uses the T5 text encoder to obtain a representative encoding of the caption text given as input, and an image containing purely Gaussian Noise as well as the text encoding is fed to the Noise prediction model. After some iterations of the U-Net Noise prediction model (as many iterations as number of timesteps used during training), an image is generated by removing the predicted noise component from the image. MinImagen feeds the generated image to two super-resolution models in succession, along with the text encoding as input, to upscale the image to higher resolutions.

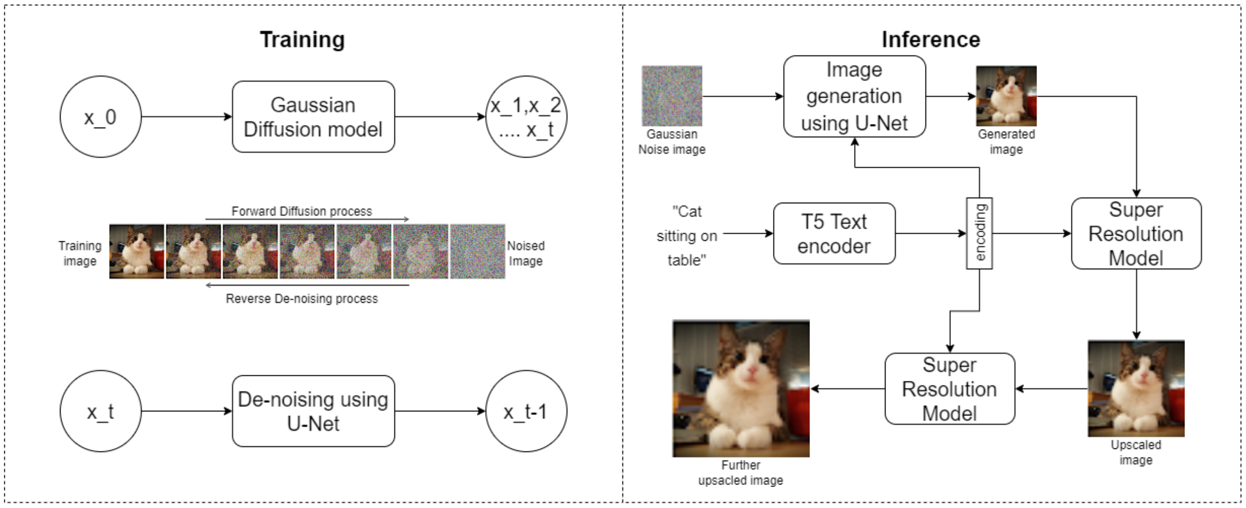

Fig. 1: Architecture of the MinImagen model.

The Super Resolution model is a U-Net which is trained on down-scaled images to generate images in higher resolution when compared to the input. MinImagen also adds some noise to the low-resolution down-scaled images for noise conditioning augmentation.

Implementation and Results

Dataset

We trained and tested variations of MinImagen on one or more of these datasets:

1. Google's Conceptual Captions dataset[1]
2. Dataset of movie posters and plot text gathered from The Movie Database API[2]
3. Google Search Images of Anime posters, with captions from google search results

Environment Setup

We used three computer environments to run different variations of our models to achieve faster results while running in parallel, one of them being a local laptop, with the other two being Compute Instances on the Google Cloud[3]. For convenience and ease of reference, let's name them as follows:

- M1: 8 vCPU, 30GB RAM, 1x NVidia V100 GPU
- M2: 12 vCPU, 65GB RAM, 1x NVidia A100 40GB GPU
- M3: 8 vCPU, 16GB RAM

Model Setup

Listed below are some of the model parameter configurations we used throughout our analysis:

- Train, Validation and Test Split - We used the standard training validation split with a ratio of 0.9. However, we haven't procedurally tested our model(s) on a test set, but rather we tested randomly on a bunch of inputs (captions), as discussed more in the Details and Analysis section.
- Optimizer and Learning Rate - We used the Adam optimizer with a learning rate of 0.0001.
- Loss Function - We used the L2 loss metric to compute loss between actual noise in a noised image and predicted noise by the model.
- Batch Size - We used two different batch sizes for various test architectures: 2 and 8 images/captions per batch, depending upon the total number of image/caption pairs the model is being trained on. Typically, we used a batch size of 2 with fewer training samples and 8 with a greater number of training samples (for operational convenience).
- Epochs - Typically, we ran most of our models for 200 epochs. For models that showed interesting or encouraging results, we ran the model for 800-2000 epochs.
- Timesteps - We used a fixed number of timesteps of 1000 for noising/de-noising the images with both the U-Net Noise Prediction Model and the Super Resolution U-Net model.
- Max number of words - The maximum number of words allowed in an input caption was kept to 64.
- Image Side length - Side length of the square output images was set to 128.
- Text Encoder - As mentioned previously in the report, MinImagen uses T5 encoder for computing text embeddings for the captions. For our analysis, we specifically used T5's "Base" encoder.

U-Net As mentioned earlier, the architecture mainly consists of two U-Nets. Each U-Net has multiple U-Net Layers, with each layer having the below components:

- Multiple Residual blocks, with each block having multiple Residual sub-blocks (each sub-block containing Convolutional layer, Batch-Normalization layer, activation layer) and a channel dimension.
- An optional self attention layer.
- An optional cross attention layer b/w image and conditioning vectors.

As an example, a U-Net with residual blocks as (1, 2, 4), initial channel dimension of 32 and dimension multiplier as (1, 4, 8) would represent three layers in the U-Net, with:

- First layer having 1 residual block and 32 channels
- Second layer having 2 residual blocks and $32 * 4 = 128$ channels
- Third layer having 4 residual blocks and $32 * 8 = 256$ channels.

Going forward in our analysis, we will represent a U-Net by specifying the initial channel dimension and the dimension multiplier.

Details and Analysis

We started out by replicating the MinImagen[4] project locally on M3, which in itself is a large codebase given the architectural details discussed in previous sections. We were successfully able to replicate the architecture[5] and do a simple train/test run (using Conceptual Captions dataset) to generate and image given an input caption. This very basic test run produced the below image (regardless of the caption) which is basically noise (as expected).

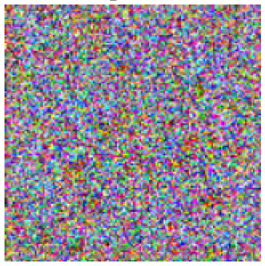
Fig. 2: Output from basic test run.

As a next step, we spent time understanding the various components and implementations in the codebase for Diffusion model, U-net and Imagen itself. As a result of our initial analysis (of code and theory) and test runs, we identified a few key model configurations that we played around with in all further experiments in this project:

- dim (no. of channels) at the greatest spatial resolution in the U-net
- no. of epochs
- dim_mults (no. of channels multiplier for each layer of the U-net)

Our next objective for the project was to tweak these identified parameters in the MinImagen code to realize interesting results.

Note: Throughout our analysis, we use very few images (16-1000) for training our models. This is because even a minimal Imagen architecture requires enormous hardware resources (time and computation) to train and test, even on M1 and M2, which was our primary constraint for this project. Therefore, all our analysis will be on a dataset of this size. Additionally, any input captions we used to test generate an image on a trained model came from the training/validation set, as our main goal in this project is to realize and achieve results which the Imagen architecture offers, within our constraint resources. Therefore, to realize this as a proof of concept, we test on training captions in the hope of achieving the best possible image generation in the results.

Our very first analysis was on a model with default U-net parameters from the Imagen paper[7], that starts out with 512 channels and then subsequent multipliers of (1, 2, 3, 4) for the Base U-net, with 2.3B parameters. This run was unsuccessful since the default configurations from the paper required a lot more hardware resources (on the scale of M2) to even initialize the model, let alone train it. Therefore, we switched to the reverse approach of starting out smaller, and then enlarging the model to the extent possible given our constraints on time and resources.

Our smallest model was with 8 initial channels and multipliers of (1, 2) in the subsequent layers of Base U-net, with 264k parameters. We trained this model for 100 Conceptual Captions captions/images for 100 and 200 epochs; and tested on an input caption (taken from the validation set) and got the below results. Not surprisingly, we still don't see any meaningful results, however, we noted that the image produced after 200 epochs seems to have more features than the one with 100, which was encouraging.

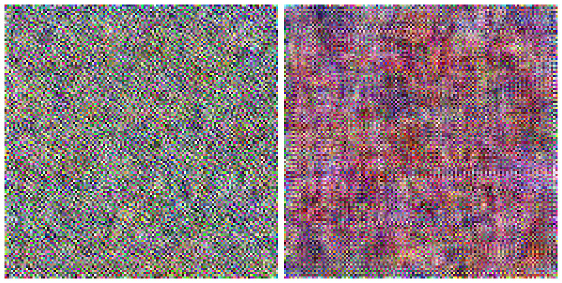
Fig. 3: Output with 8 initial channels for 100 and 200 epochs, respectively.

Next, we tested a model starting with 256 channels in the U-net and multipliers of (1, 2, 4), with 396M parameters. We ran this model on a Conceptual Captions dataset of 100 images for 100 and 200 epochs and achieved below results (captions taken from the training set). These results definitely look more feature rich than the previous model (with the output from 200 epochs being slightly better than 100), although nothing too impressive so far.

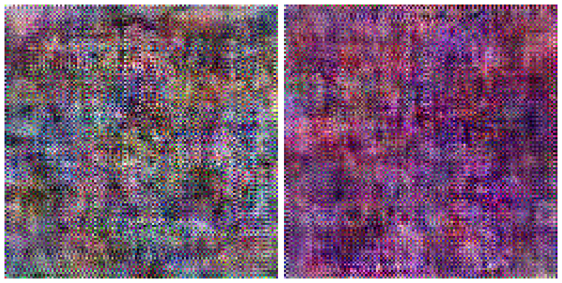
Fig. 4: Output with 256 initial channels for 100 and 200 epochs, respectively.

From the previous two tests, we learned that increasing the number of epochs as well as the dimensions is having a positive impact on the generated quality of the image, which was also expected. Having this information validated, we decided to test a model with 256 initial channels; multipliers (1, 2, 4) (same as the previous test); and run it for 800 epochs. We made two changes this time, however.

- We ran the model on a training dataset containing animated images/captions
- We ran it for fewer images (specifically, 16 images).

Our first change was based on the intuition that simple animated characters' images will have less complex features and hence would be comparatively easier to learn, which is suitable for our constraints.

We did the second change since we were running the model for a lot more epochs and running it for say 100 images, would require a much more time than we could afford. To give an estimate, this model run took ∼8 hours on M1 for training and ∼2 more hours for inference on three input captions (taken from training dataset).

The below generated images look extremely promising as we can clearly observe more complex features in the images now.

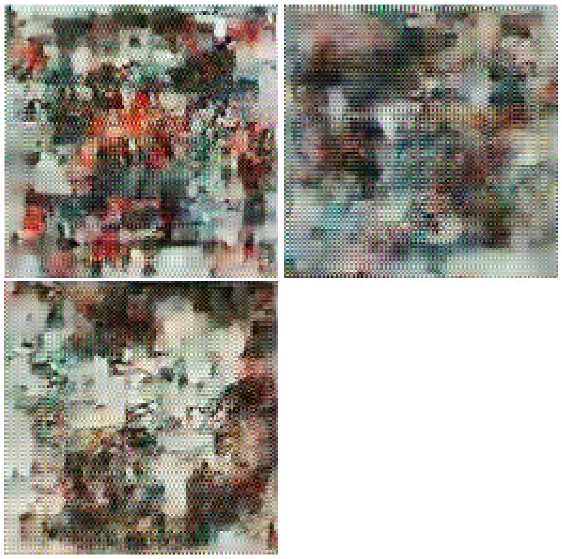
Fig. 5: Outputs for three input captions on the model with 256 initial channels, trained on animated images and run for 800 epochs.

Motivated by the above generated images, we continued running the same model (256 dim, (1, 2, 4) multiplier) for up to 1500 epochs and below are the original training images and their corresponding generated images.

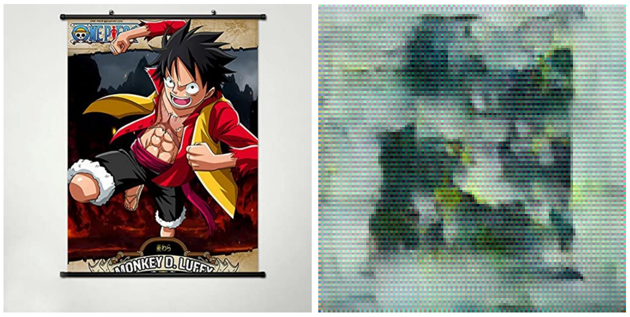

Fig. 6: Original image and inferred image for the caption "I don't want to rule anything. Being King of the Pirates means being more free than anyone."

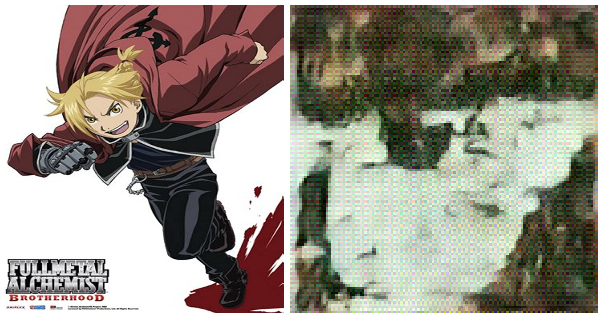

Fig. 7: Original image and inferred image for the caption "There's no such thing as a painless lesson. They just don't exist. Sacrifices are necessary"

As we can observe above, the model is beginning to capture the high level features of the training images now, like outlines, shapes, and colors.

Hence, from our implementation and analysis, we can safely assume that if we continue for more epochs and potentially train on more images, we should start seeing impressive text-to-image generation results. A validation of this assumption is also shown in the below generated images over a timeline of higher epochs.

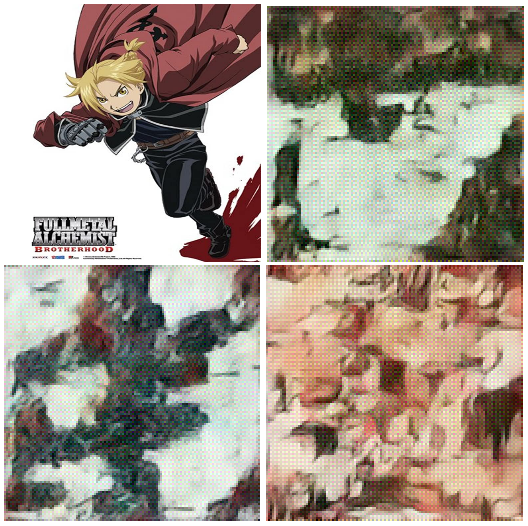

Fig. 8: (left to right, top to bottom) Training image; Inferred image (1500 epochs); Inferred image (1800 epochs);

Inferred image (2000 epochs); for the caption: “There’s no such thing as a painless lesson. They just don’t exist. Sacrifices are necessary”

We can clearly see above that the model is starting to capture finer details of the actual training images like faces (we can spot an anime girl face in the bottom-left of the last image), skin color, hair, although they seem jumbled up as of now, which should get better as we train more. Also note that, given an input caption, the model is not necessarily trying to generate an image that looks exactly the same as the original training image for that caption. This is because the model learns on several image-caption (or text embedding) pairs that can be overlapping (common words, image features, etc) and performs image synthesis for a given caption based on its overall learnings on the entire training dataset.

Conclusion

In this project, inspired by the latest state-of-the-art developments in the field of Text-to-image synthesis, we set out to explore MinImagen[4], a minimal implementation of Imagen[7], with two goals in mind:

- replicate the MinImagen implementation and in the process understand how actual Imagen works (including U-nets, diffusion models, text encoders)
- tweak the MinImagen model to generate movie posters based on text input of movie plots.

While we were able to achieve our first goal successfully, we pivoted on the second goal. Instead of tweaking MinImagen for movie poster generation, we decided to train it on a small subset of anime characters and their taglines/quotes. The reasoning for this pivot was that we were not getting any meaningful results for movie posters in a reasonable number of epochs (given the constraints on time and resources). We hypothesized that pivoting to a use case (anime character images) which has comparatively fewer features to learn should yield good results faster. While we did obtain some promising results (as discussed), they were far from what a well trained MinImagen model should be capable of in an ideal environment.

Therefore, we still have a lot more to explore and experiment with respect to Imagen and text-to-image synthesis in general. A few ideas that we wanted to try but could not include training on a larger dataset of images and for more epochs; using a deeper model with more parameters and a bigger T5 encoder (Imagen paper[7] clearly highlights the better quality of results obtained using a bigger text encoder); tweaking the super-resolution U-Net image generator (something which is a central component in the Imagen[7] architecture and which was left unexplored in this work); and testing our models on various use cases to understand the applicability of a model configuration in various domains.

References

1. Google’s Conceptual Captions. Online. Available: https://ai.google.com/research/ConceptualCaptions/
2. The Movie Database (TMDB) API. Online. Available: https://developers.themoviedb.org/3/getting-started/introduction
3. Google Cloud Virtual Machine Instances. Online. Available: https://cloud.google.com/compute/docs/instances
4. MinImagen - Build Your Own Imagen Text-to-Image Model. Online. Available: https://www.assemblyai.com/blog/minimagen-build-your-own-imagen-text-to-image-model/
5. AssemblyAI-Examples. MinImagen. Online. Available: https://github.com/AssemblyAI-Examples/MinImagen
6. Prafulla Dhariwal, Alex Nichol. Diffusion Models Beat GANs on Image Synthesis. Online. Available: https://arxiv.org/abs/2105.05233
7. Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S. Sara Mahdavi, Rapha Gontijo Lopes, Tim Salimans, Jonathan Ho, David J Fleet, Mohammad Norouzi. Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding. Online. Available: https://arxiv.org/abs/2205.11487
8. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. Generative Adversarial Networks. Online. Available: https://arxiv.org/abs/1406.2661