ComicGAN: GENERATING COMIC BOOK COVERS USING GENERATIVE ADVERSARIAL NETWORKS

Daniel Yang, Gagan Kaushik, Kristen Erlon
Computer Engineering Students at the University of Texas at Austin

https://comicgan.herokuapp.com
https://github.com/gagank1/Self-Attention-GAN
https://github.com/danielyang2000/Web-scraper-for-comicvine.com

ABSTRACT

In this paper, we describe the process we went through to build a website that generates new comic book covers using generative adversarial networks (GANs). We scraped *comicvine.org* for 100,000 cover images to train the GAN. Our website dynamically generates samples and offers viewers a choice between four different models.

1. INTRODUCTION AND MOTIVATION

This project is an exploration of comic book art through the use of GANs. Comic cover art is absolutely unlike any other art form. The bold colors, the artistic fonts- all so unique. Being avid comic readers, the team wanted to see how image processing would handle such a unique medium. More specifically, would a computer program be able to recreate it?

GANs are known to be good at synthesizing faces since they are doing fine interpolation of large datasets. In addition, faces are perfect since the human eye, mouth, nose, etc.. are all relatively predictable and well-defined. On the other hand, comic covers are far less well-defined and much more diverse due to different characters, titles, and art styles appearing on each cover. The team wanted to push GANs' capability to see how well it can generate a new cover from old ones. Will it just be a montage of random colored regions with alien texts at the top, or will it show clearly generated characters just like human faces?

This is an exploration into both the technical and the print side. Through the lens of GANs, we were able to see what consistencies exist on the comic covers and realize patterns in style and form.

2. METHODS

Our main goal for this project was to display generated comic book covers on our website. To do this we split the project into three parts, where each member focused on one part of the pipeline. Daniel focused on scraping comic covers as well as augmenting the data by flipping each cover horizontally and getting each cover to the same shape. Gagan worked on model selection, GAN training, and full-stack development. Lastly, Kristen focused on front-end development, front-end to back-end connection, and image quality analysis.

Although each of the three parts are very different, they are highly interactive with each other. We constantly had to check with each other to see if our work was compatible with one another's. This check let us see if each member was doing his/her work with good quality as well as learning what he/she was doing. Because of this, we all had a good understanding of each part of the project.

2.1. Web scraping

In order for our GAN to work, we need data to train it. We initially decided that about 100,000 comic book covers are needed for training. We have a dataset with 100,000 cover images of size 128x128 and another with 100,000 cover images of size 256x256 to see which one gives better quality when training our GAN. If they are not enough, then we would find or scrape more comic book covers.

2.1.1. Finding resources

The biggest issue we encountered here was finding the comic book data. We first looked around the internet to

see if there are existing comic book datasets that we can use to train the model right away, but we couldn't find any. Then we tried to find existing code that scrapes comic book covers from popular comics websites such as *comics.org, comicvine.com, coverbrowser.com, etc…* We did find a few but they are outdated and could not be used, since the website they are trying to scrape changes its API every now and then. In the end, we found a repository created by GitHub user "miri64" that works well [1].

Miri64's repository scrapes data from *comicvine.org* using its API resources, which provide full access to their structured-wiki content data in JSON formats [2]. The user just needs to sign up for an API key in order to scrape the website. Miri64's repository can be used in a variety of ways such as getting the url of the issue image (which is the cover image of that issue), getting a list of issues from a particular volume, getting a list of volumes from a particular publisher, etc…

2.1.2. Scraping data

We decided to scrape only Marvel and DC comic book covers to reduce the amount of different characters and art styles our GAN had to deal with. However, just Marvel and DC comic covers were not enough, so we decided to scrape only 50,000 covers from them, then augment the other 50,000 by horizontally flipping each cover image. Since each cover has their characters facing different directions, horizontally flipping the cover does not have many negative effects.

All of our web scraping code was written in Python on Jupyter Notebook by Daniel. The link to the web scraping code is posted at the top of page one.

Using miri64's repository, the list of volumes published by Marvel was scraped. Then the list of issues for each volume was scraped. Finally for each issue, the url of its cover image was scraped, which can be received by using the request module in Python. Then we wanted a copy of it to have size 128x128 and another with size 256x256. We first tried resizing the original scraped image right away with the PIL resizing function, but the resized ones were blurry. We later came up with a better solution in that for each cover image that we scraped, we cropped the largest square starting from the middle top first, then resized it to 128x128 and another copy of it to 256x256. Since most covers have their title and characters

Figure 1: The top two images are the original and cropped. Bottom four shows the normal and flipped image of each size (middle two are size 128x128 and bottom two are size 256x256)

in the top square, cropping the top largest square in the cover does not lose much information. These two cropped and resized images are then horizontally flipped using PIL transpose function to produce two more images. Lastly, our two 128x128 size and two 256x256 size images are then saved to two folders in our laptop directory based on their image size. Figure 1 shows an example of what the four saved images look like.

After finishing scraping all Marvel comic covers, we were 14,000 away from the desired 100,000 cover images. We then scraped the rest of the covers from a few Batman volumes and a few Superman volumes, using the same image processing method for cropping and flipping as before.

In addition, *comicvine.org* has a rate limit on how much people can scrape per hour. To avoid exceeding the

rate limit, we used the time module to let the program sleep for 6.5 seconds after scraping 8 covers (since each scraped cover is horizontally flipped , that's a total of 16 128x128 size images and 16 256x256 size images). We chose this time based on trial and error. This is the fastest we can do without reaching the rate limit and crashing the program, which will slow down the scraping process.

2.2. GAN Development

GANs are a relatively new set of generative models that can be used to create artificial images among many other applications. A GAN consists of two separate neural networks that are trained against each other: the generator and the discriminator. The discriminator's goal is to decipher fake images from real ones. The generator's goal is to fool the discriminator as much as possible.

2.2.1. Choice of Architecture

The most popular GAN architecture for images seemed to be DCGAN, with over 10,800 citations on their paper [3]. DCGAN was state of the art for a while, but its shortcomings became apparent. DCGAN performs well on relatively uniform, smooth images. Examples include beaches and peoples' faces. However, it doesn't perform as well on highly structured images with global dependencies. This is because the convolutional operator only captures local relationships, and although the relationships are propagated through the network, they are often not strong enough by the time they get to the later layers.

Zhang et al. developed Self-Attention Generative Adversarial Network, or SAGAN as a solution to the lack of global dependencies in convolutional nets [4]. SAGAN uses self-attention layers which capture the relationships between a set of feature maps and themselves.

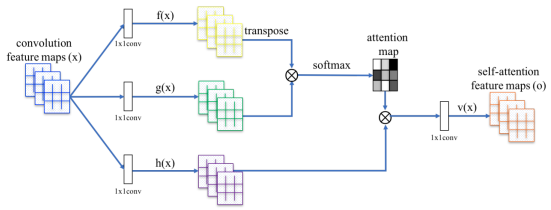

Figure 2: Self-attention layer architecture as laid out in the SAGAN paper

We chose to use SAGAN because comic book covers are highly structured and have complex relationships between different areas of the image. For example, there are always titles at the top and characters in the center.

2.2.2. Training

We started out by forking *heykeetae*'s "Self-Attention-GAN" repository on Github, a PyTorch implementation of SAGAN. This was missing the ability to use custom datasets and non-standard image sizes. We augmented the repository by adding functions to support custom datasets and a dynamic architecture that varies with image size, as suggested by existing Github issues. We used TACC to accelerate training times, as it would be intractable. We chose to use the 128x128 dataset due to long training times. We used 4x Nvidia Geforce GTX 1080 Tis and trained for ~20 epochs (30,000 iterations). Hyperparameters were left the same as those in the paper, except imsize (=128) and batchsize (=64). Training took 8-9 hours to complete.

2.2.3. Improvement

After the first training run, we were seeing mode collapse and instability/non convergence. Mode collapse is when the generator learns to generate one set of images really well and continues to exploit the discriminator on this set. Eventually, the discriminator catches on and starts to win consistently. Then the generator moves onto another set of images and continues the cycle. This leads to unstable training and a very limited set of generated images.

One of the strategies commonly used to prevent this is to handicap the discriminator to allow the generator to learn better representations of the data. We do this by adding random gaussian noise before convolutional and dense layers in the discriminator only [5]. We implemented this into our fork of the repository as a PyTorch module that we added to the layers. We also had to modify the spectral normalization layer to fit in the gaussian noise. Once this was done, we trained the GAN again from scratch on the whole dataset.

2.3. Website Development

The team created a website using HTML, CSS, Javascript, and Python. The purpose of this website was to learn website development techniques as well as showcase our work. One of the team members, Kristen, has a personal interest in graphic design and website design, so she used this as an opportunity to delve into

learning website development techniques, from the backend to frontend connection to server deployment.

After analyzing our program needs, we decided HTML and CSS would be best suited to write the majority of the site in. Javascript was then added to make requests to the Python backend, which ran the actual simulation. By combining these four, the team was able to create an interactive and aesthetically designed website.

2.3.1. HTML Usage

The HTML portion makes up the majority of the website's code. Everything that is visible from pages, menu, images, headers and the larger text portions are all directly coded in HTML.

We used starter code provided by W3 Schools [6] to provide a bare bones structure for the website, as well as libraries they made to simplify actions like centering items, adding page breaks, etc. Their tutorials and language documentation were very helpful.

The page was sectioned into five main parts: the top bar, image analysis page, demo, "About" page, and footer. The top bar includes our title, date and drop down menu. The drop down menu would move the user to specific parts of the page depending on which selection they make. The top bar stays fixed on the top of the page when scrolling through the site. The image analysis portion includes a header, paragraph, an original comic book cover image, and then two sections of subdivisions of three each including image, header, and descriptor paragraph that analyzes the GAN results. We used w3-center and w3-third imported from a W3 library [7] to separate the layout into three columns. This was then repeated to create what can be seen as a 2x3 grid. The demo portion uses javascript and python, further detailed in 2.3.3, to simulate the python program we wrote and adds an interactive element to the page. The "About" section replicates formatting techniques described above in a new order. Finally, the footer shows our names and acknowledgements.

2.3.2. CSS Usage

CSS was used to create a consistent website in regards to formatting and functionality. While the code is shorter, the effects of using this were substantial. CSS code is not necessary for creating a functional website. However, it allows for page elements like headings, backgrounds, and fonts to be stylized to fit [8].

This is where we standardized formatting like making the paragraph sections all Verdana font and 20pt font size. We were able to center all headers in this code as well. Finally, we changed the background of all pages to bright yellow. We chose yellow because it is a very prominent color found in earlier eras of comics. The HTML file implemented the CSS portion by importing the CSS file as a stylesheet.

2.3.3. Backend Development

The backend was developed using Flask, a Python web application framework. Flask was the straightforward choice as it is easy to connect with other Python modules. We developed a ComicGenerator class that loads pre-trained weights that were exported during training. It has a method that generates new images when called.

The flask server serves the static web page and also has one specialized endpoint. */imgen* takes an argument *genNum* which allows the user to select one of four generators to use to create new comic book covers. It generates an image using the requested generator and serves it to the front-end to be displayed. We deployed the website onto a Heroku pipeline.

3. RESULTS

Our results with the original model had many interesting features, despite not being realistic to the human eye.

Figure 3: Sample images after 30,000 iterations on the original SAGAN model

SAGAN learned that comic book covers nearly always have large blocks of text at the top of the page, and that there are vaguely humanoid objects near the center of the page. We can see a few different variations of this in the image above. However, there is clearly mode collapse occurring as these images are not a good representation of the original data distribution.

Our results from our augmented SAGAN model with Gaussian noise showed more variability between images in the same class, although we saw the same mode collapse as before. This makes sense since random noise will disrupt the "traditional" training process.

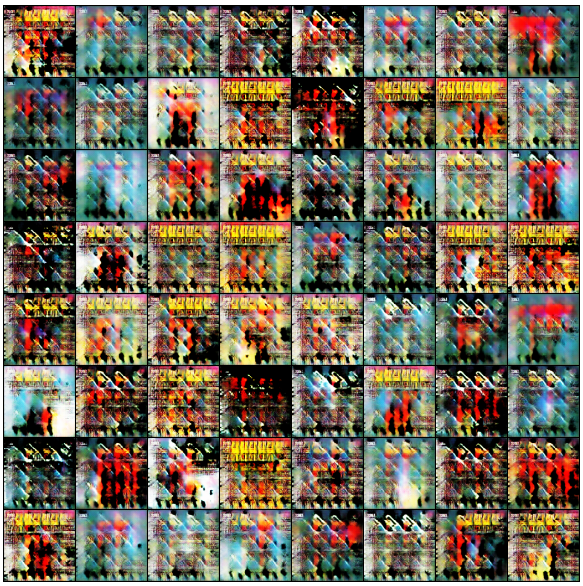

Figure 4: Sample images after ~15,000 iterations on SAGAN+Gaussian noise

Here in Figure 4, we see the characters in the images in many different positions, and we see the yellow title clearly in most images. This is likely an iteration where the GAN caught onto Iron Man comics.

However, not all iterations had such good samples even with Gaussian noise added as represented in Figure 5.

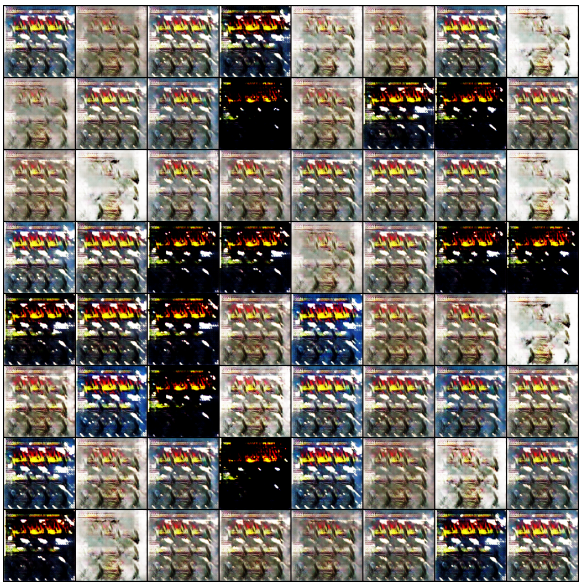

Figure 5: Nearly all the images in this iteration look alike and have the exact same objects in the center

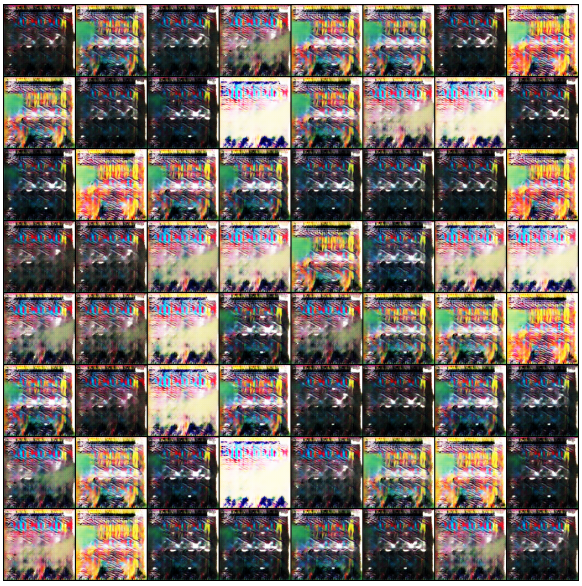

Figure 6: Very similar titles in all the images, but good variability in the central objects

Overall, we see that GANs are fairly powerful networks that can pick up on high level patterns as well as certain details, even given tough and limited data to work with.

4. FUTURE WORK

There are many avenues for future work with this project. We would benefit greatly from a more robust dataset if we had more time to scrape carefully. Our dataset was mostly Marvel images with some DC, and half our images were horizontally flipped to cut down on scraping time. If we had a 500k+ sample dataset with diverse, unique comic book covers, we would likely see an increase in realism and reduction in mode collapse.

We could also leverage the larger dataset to use a larger, more powerful model following the same SAGAN ideas. We could add more self-attention layers and perhaps skip connections to improve performance.

If we still see mode collapse after those two improvements, we could try more strategies such as one-sided label smoothing and unrolling the GAN [9].

5. CONCLUSION

Overall, this has been a wonderful experience for the team. Each member encountered numerous challenges and learned a lot from their work.

While the results were not quite what we were expecting, they are still very interesting to view and analyze. In the beginning, almost every generated cover was similar to the ones shown in Figure 3. After some improvements on mode collapsing, different variability of characters and objects started to appear as shown in Figure 4,5, and more on our website. We started to see patterns that pointed out similarities on the covers we didn't even notice originally, i.e. the price/logo square in the left corner of every comic.

From what we have seen, GANs do not produce very realistic comic book covers. Although we see some structures in the generated covers, they still contain random colored regions overall and the models tend towards mode collapse. Comics truly are art and this just goes to show that nothing can replace the vision and skills of an actual artist.

6. REFERENCES

[1] *pycomicvine*. (2013), Martin Lenders. [Online]. Available: https://github.com/miri64/pycomicvine

[2] "Comic Vine is the largest comic book wiki in the universe". Comic Vine https://comicvine.gamespot.com/

[3] A. Radford, L. Metz, en S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", arXiv [cs.LG]. 2016.

[4] H. Zhang, I. Goodfellow, D. Metaxas, en A. Odena, "Self-Attention Generative Adversarial Networks", arXiv [stat.ML]. 2019.

[5] F. Huszar, "Instance noise: A trick for stabilising Gan Training," inFERENCe, 21-Oct-2016. [Online]. Available: https://www.inference.vc/instance-noise-a-trick-for-stabilising-gan-training/. [Accessed: 08-Dec-2021].

[6] *W3 Schools*: HTML. [Online]. https://www.w3schools.com/html/default.asp

[7] *W3 Schools*: Responsive Classes. [Online]. https://www.w3schools.com/w3css/w3css_responsive.asp

[8] *W3 Schools*: CSS. [Online]. https://www.w3schools.com/cssref/default.asp

[9] "Common problems [with] generative adversarial networks," Google. [Online]. Available: https://developers.google.com/machine-learning/gan/problems. [Accessed: 08-Dec-2021].