CS156 Final: GAN Cityscapes

Oluwakorede Akande

December (Fall) 2021

1. Problem Definition

Artificial intelligence (AI) is becoming an increasingly prevalent tool in many industries in today's society. The fashion industry is one such industry that has been impacted by AI, with fashion designers increasingly using AI tools and devices to create fashion models, fabric designs, and clothing (Dennis, 2020).

In this project, we explore the potential of an AI model to generate fashionable clothing, focusing specifically on cityscape graphic t-shirts of the kind shown below.

Figure 1: Cityscape T-shirt Images

In particular, we focus on the graphic element of the t-shirts and evaluate the ability of an AI model to create new and realistic cityscapes (based on existing cityscape images) to serve as graphics for t-shirts.

2. Solution Specification

For the image/fashion generation task at hand, *Generative Adversarial Networks (GANs)* are a widely applied technique (Ping et al., 2019). As such, a GAN model is selected for implementation in this project.

Generally speaking, a *GAN* consists of two neural networks, a *generator* and a *discriminator*. The *generator* takes as input a random vector and aims to generate a fake sample that looks like a real one from the original dataset. On the other hand, the *discriminator* is trained with both real and fake samples and predicts whether or not a given sample is real. This prediction result is then back-propagated through both networks to optimize the parameters (Wang & Xiao, 2021).

Fig 2 situates the implemented GAN model (see *Generator* and *Discriminator Networks*) in the context of the entire workflow. Each component of the system is discussed in detail below.

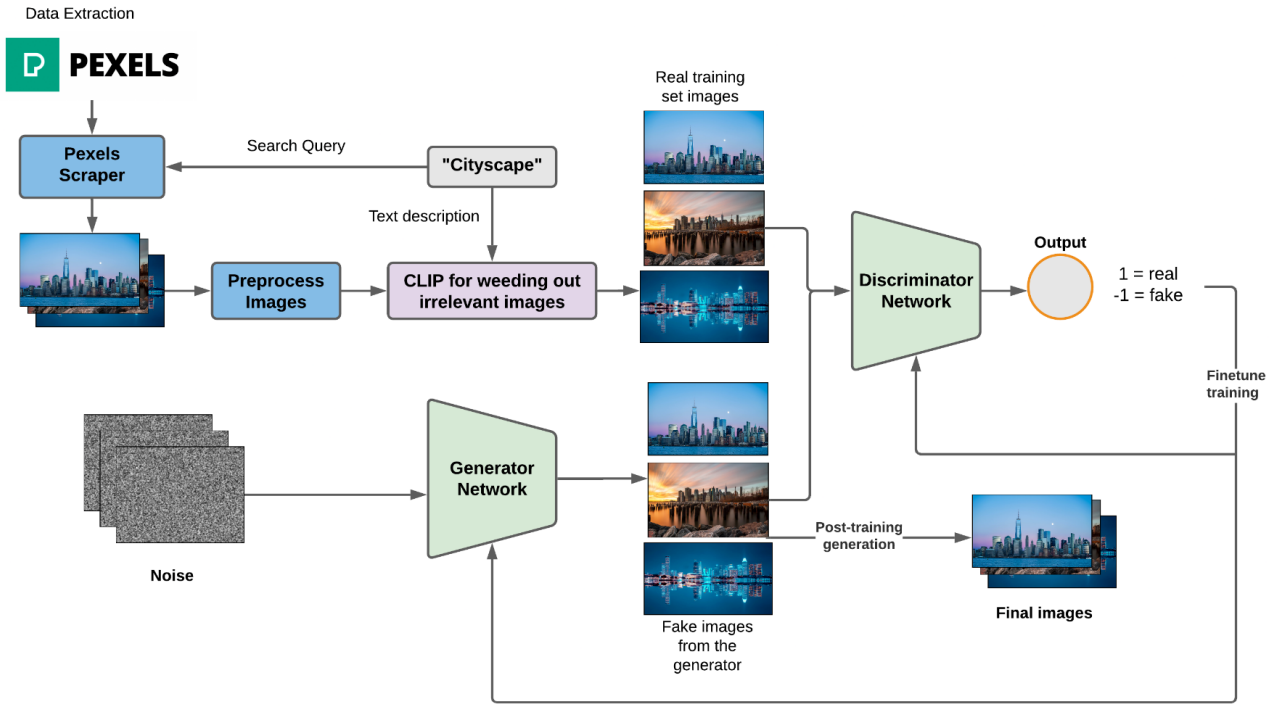

Figure 2: Project System Architecture

a. Data Extraction

Pexels[1] was utilized as the data source for cityscape images. *Pexels* was deemed an especially good choice because it offers free stock images that can be used for any purpose and provides a free and easy to use API with satisfactory quotas.

To specifically obtain the images via the API, a publicly available *Pexels web scraper*[2] was utilized. I simply provided my API key, a search query (in this case, 'cityscape'), the directory to store the images in, a range of pages to extract images from, and the number of images to extract per page (see *Fig 3*).

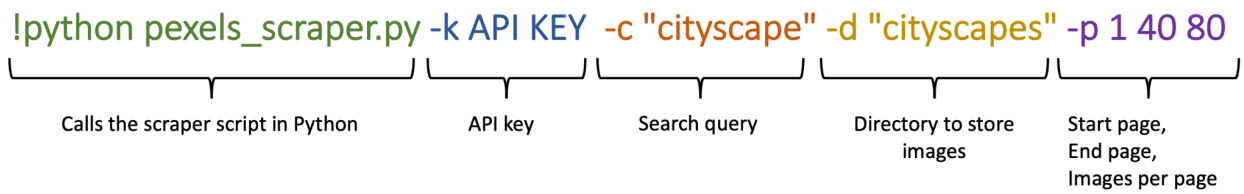

Figure 3: Pexels Data Extraction Function Call

[1] https://www.pexels.com/
[2] https://github.com/ToshY/pexels-scraper

The final extracted dataset had 7983 images[3], a few of which are displayed below in *Fig 4*:

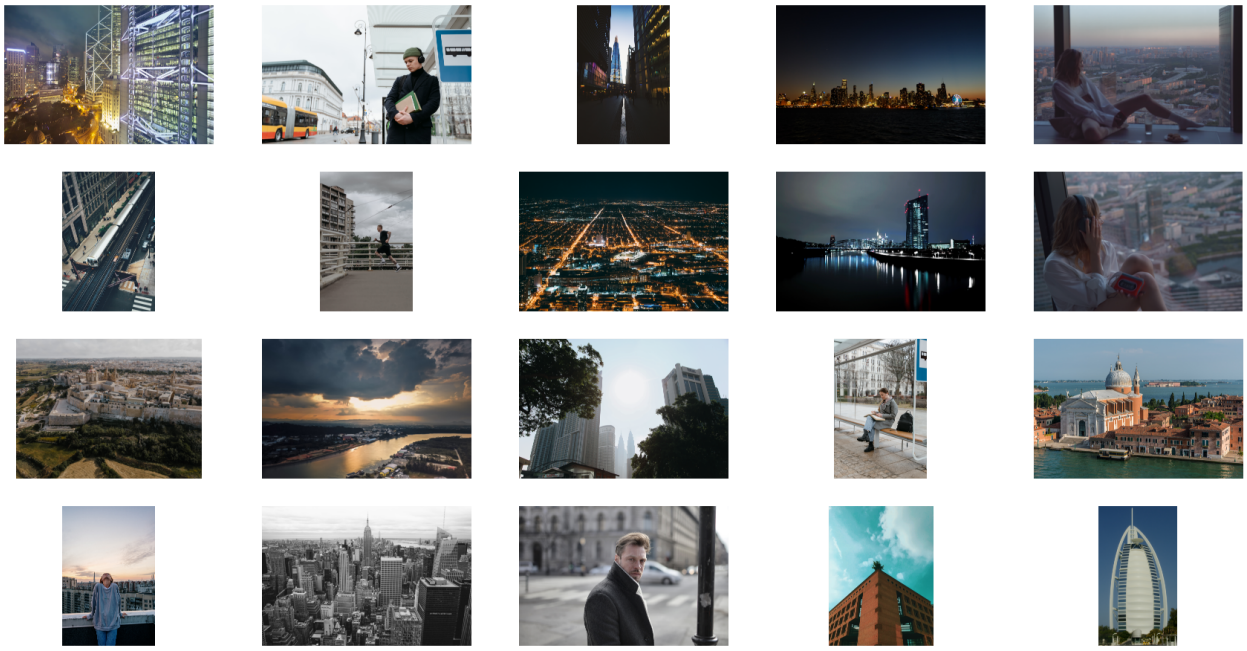

Figure 4: Cityscape Images (Source: Pexels, n.d.).

b. Image Processing

Given the images had varying aspect ratios (see *Fig 4*) and most state-of-the-art GANs require standard size input images (i.e. square images) (Connah et al., 2020), the aspect ratios of the images were adjusted.

In particular, the *center rectangle* approach developed by Gonsalves (2021) was utilized in resizing the images as it results in images that are free of distortion. This technique requires first determining the median aspect ratio of all the original images (which for the cityscape dataset was 1.5:1). Subsequently, each image is cropped into that median aspect ratio and a center rectangle is pulled out. Finally, the cropped images are squeezed

[3] The full (uncleaned) dataset can be accessed here: https://drive.google.com/drive/folders/11Q0nR4cl3LagBcQsLAfjS1as6_wBlbRW?usp=sharing

horizontally into a square format with a resolution of 1024 x 1024 pixels.

Fig 5 shows an example of the resizing applied on a cityscape image.

Figure 5: Image resizing using center rectangle technique. Left: The original cityscape image. Right: The square image after resizing with the center rectangle method.

c. Data cleaning and curation

By the nature of search on the *Pexels* platform and the fact that uploaders can tag anything in their images, there is a high chance that some images extracted via the API are not good representations of cityscapes (see *Fig 4* for evidence).

Instead of manually filtering out irrelevant images–a process that would be very time-consuming and repetitive–I employed a neural network developed by OpenAI called *CLIP (Contrastive Language–Image Pre-training)*[4] to carry out this filtration process.

[4] https://openai.com/blog/clip/

CLIP is a multi-modal vision and language model that can determine image-text similarity. To do so, it uses a vision transformer (ViT) to get visual features from the image and a causal language model to get the text features from the inputted text. Both the text and visual features are then projected to a latent space with identical dimensions. The dot product between the projected image and text features is then used as a similarity score (Huggingface, 2017).

In this scenario, CLIP was used to determine the similarity between all the extracted images and the word 'cityscape.' Next, images were dropped using a manually (and visually) determined similarity threshold of 0.21. Specifically, 0.21 was selected as the threshold because, based on observation, it was found to capture a high number of true positives and a few false positives for cityscape images.

Fig 6 shows a sample of images that fell below the 0.21 threshold[5]. These images will be

[5] The full dataset of removed images can be accessed here:
https://drive.google.com/drive/folders/1PPFbd1ZFh3rKbhhvK7bqOdbgk-Ac4T9x?usp=sharing

removed from the dataset.

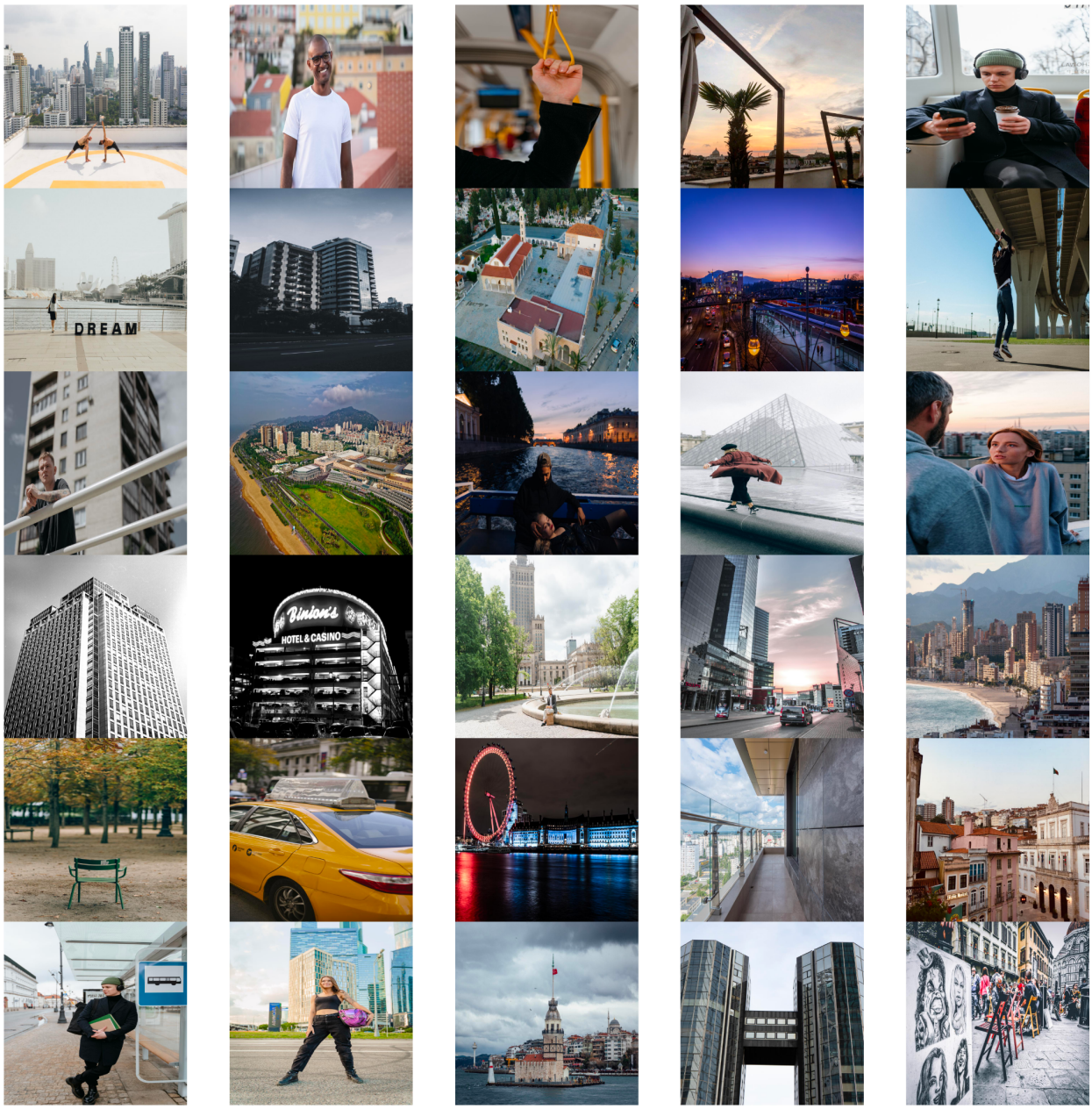

Figure 6: Sample of images with low similarity to 'cityscape' according to CLIP. Here a threshold of 0.21 was utilized

After removing all images (about 2000) below the threshold, we are left with more appropriate cityscape images[6] as shown in the random sample in *Fig 7*:

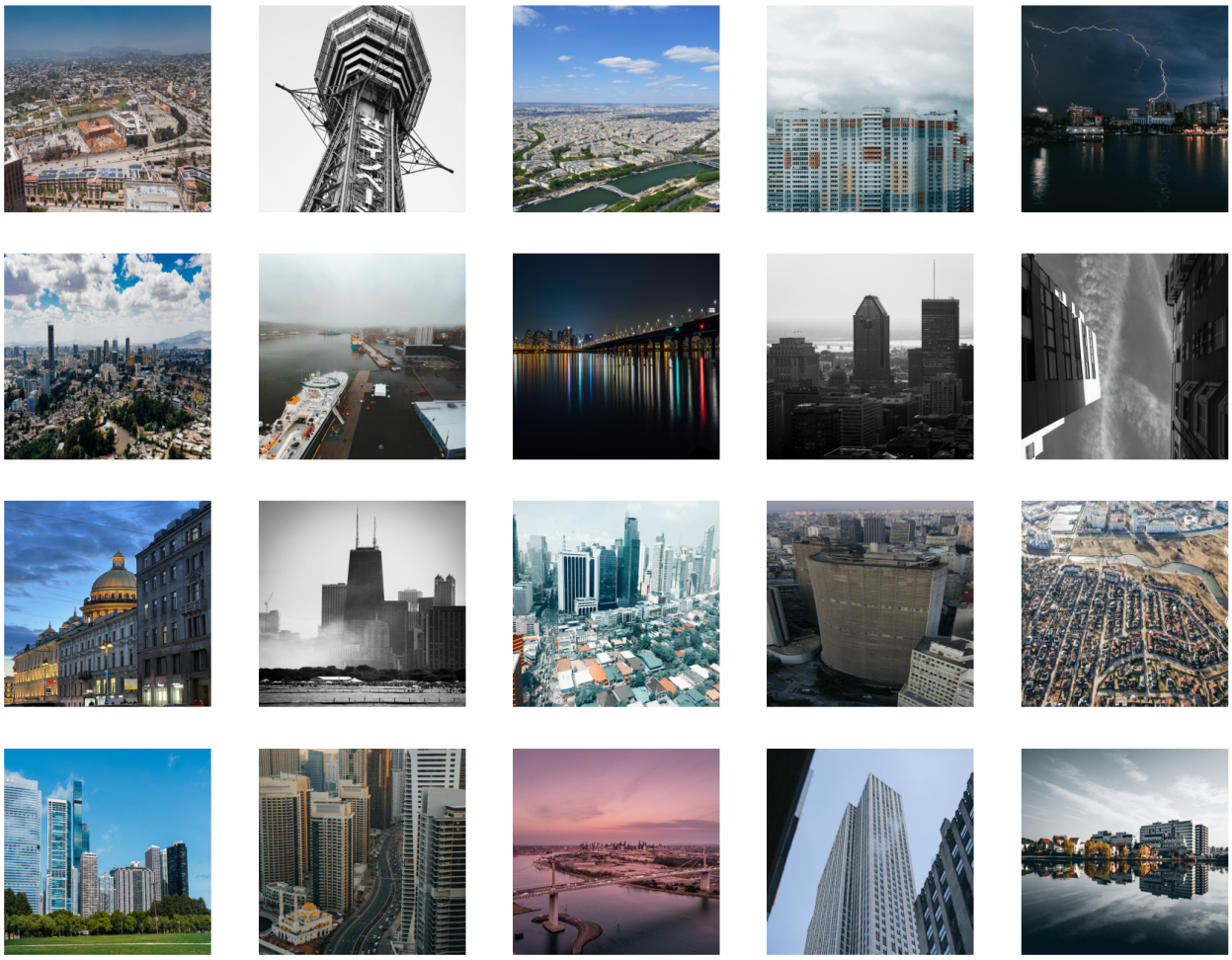

Figure 7: Random sample of images left after cleaning out images with low similarity to 'cityscape' according to CLIP

[6] The cleaned and resized dataset can be accessed here: https://drive.google.com/drive/folders/1whRIEPYK50FQfveDO6RYmp_9jtFU-UWZ?usp=sharing

d. Modeling: Generative Adversarial Network[7]

In settling on the type of GAN model to train and the programming environment that would enable this training, various obstacles and constraints were considered.

In relation to my existing resources, the following in particular, posed huge challenges:

i. **Memory size or disk space:** My laptop (MacBook Pro 13 inch 2018) has a measly memory of 250GB, of which just 25GB is free. This already rules out the possibility of running this project locally as the image dataset on its own would require far more space than that.

ii. **Limited GPU:** A GPU is essential for running GANs. Given my laptop has a single GPU, allocating it entirely to this project would effectively render my laptop useless for other tasks.

Additionally, due to the project deadline and my set budget, all implemented solutions were constrained by the following:

iii. **Time constraints:** The programming environment and the selected GAN model, jointly should be capable of providing meaningful results (both in terms of output images and in terms of metric results) easily and consistently, days before the project deadline. Additionally, the implemented solution should ideally provide good solutions in as little time as possible.

iv. **Money constraints:** This constraint is more relevant to the programming environment, especially cloud-hosted environments which typically require

[7] **#constraints:** I outlined my approach to evaluating and selecting a programming environment and GAN model to utilize for the project, situating it in the context of my laptop's capabilities and the project's deadline. This involved identifying obstacles and constraints to the task at hand, given my (then) current situation and identifying existing tools that could satisfy/overcome those obstacles and constraints.

payment to enjoy powerful GPUs and large RAM. Selected solutions should be effective yet affordable.

v. **Dataset size:** Training GANs can require upwards of 100,000 images. However, there exist methods that enable results with 10 to 20 times less than this amount (Johnson, 2020). Although more data can always be gathered, I consider my dataset size (described in sections above) a constraint in this scenario as I must allocate enough time to actually build and train the model. The more time spent collecting data, the more time must be allocated to cleaning that data, and in turn, the less time I have for the actual modeling. Hence, I pause on data collection and look to GAN models that can provide great results with limited data.

Google Colab

Taking all these into account, I settled on Google Colab as the programming environment as it is clearly superior to the available resources on my local machine. Google Colab works in tandem with Google Drive, on which I have a significant amount of storage space (via my Minerva account), thus solving the problem of limited memory. Additionally, Google Colab offers an affordable Pro plan for about $10 which also provides access to a large RAM and a GPU.

Lightweight-GAN

For the GAN modeling, a lightweight-GAN is selected as the model of choice. This model has been shown to learn on high-resolution images, with low computational cost and few training samples (Liu et al., 2021) thus (potentially) satisfying our constraints of a small dataset and limited time. In particular, Liu et al (2021) report lightweight-GAN training on

samples in a fraction of the time used by the state-of-the-art model, StyleGAN2 while still yielding consistently good performance, even with less than 100 training samples.

The specific implementation of Lightweight-GAN in this project was mediated through a Python package written in Pytorch[8]. This package was used to train on the cityscape data, with the hyperparameters being fine-tuned to improve performance.

Below I discuss metric results for the different runs of the Light-weight GAN:

3. Testing and Analysis[9][10]

a. Model Evaluation

To evaluate the quality of the generated samples by the GAN, the popular *Fréchet inception distance (FID)* metric was utilized. According to Heusel et al. (2018), this metric correlates with human evaluation of visual quality and is more robust to noise than Inception Scores.

In particular, the process of computing the FID involves first embedding the generated samples into a feature space given by (a specific layer) of Inception Net. Consequently, the embedding layer is viewed as a continuous multivariate Gaussian, whose mean and covariance are estimated both for the generated data and the real data. The Fréchet distance between these two Gaussians is then used to quantify the quality of the samples, with a lower FID indicating better-quality images (Heusel et al., 2018).

[8] https://github.com/lucidrains/lightweight-gan
[9] #modelmetrics
[10] #neuralnetworks: Firstly, I identified the opportunity to leverage the CLIP neural network to filter images, reducing the need to manually filter out the images myself. Secondly, I justified and implemented a Lightweight-GAN to generate cityscape images. Additionally, I conducted a grid search on the hyperparameters to suggest initial hyperparameters that might be favourable for the GAN in optimizing its FID.

The FID computation was embedded within the Lightweight-GAN package and could be passed as a parameter to compute after a series of timesteps. Additionally, I utilized the pytorch-fid[11] package to compute the score outside of Lightweight-GAN.

Run 1: Modeling without Attention Layers

Although the default number of training steps for the Lightweight-GAN implementation is 150,000, I set the number of steps to 10,000 to get a sense of how the model performs in the short run. Additionally, I enabled automatic mixed precision as the author of the package, Phil Wang (2021), reported it speeding up the modeling process by 33% while saving up to 40% memory. Most notably, I excluded the attention mechanism from the model as individuals in the issues section reported improved visual quality on small datasets after attention was removed. The entire run lasted about 16 hours and produced pretty poor results (see Table 1 and *Fig 8*).

Run 2: Modeling with Attention Layers

Following the results from the first run (see Table 1 and Fig 8), it was evident that the GAN was not trained for long enough. As such, in the second run, I doubled the number of steps to give the model a longer period of time to learn unique and key features in the images. Asides this, the major changes I made were adding colour manipulation as an augmentation type (to the default of translation and cutout) and including the attention layers previously excluded in Run 1. The inclusion of the attention mechanism was deemed especially important as attention helps capture long-range dependencies and the overall structures of images (Welander et al., 2018).

[11] https://github.com/mseitzer/pytorch-fid

The results from Table 1 in fact suggest (although confounded by the inclusion of colour augmentation) that the addition of attention does improve FID. We note that the FID value obtained at 5000 steps for the second run was better than the best FID value obtained across the 10000 steps from Run 1.

Figure 9 displays the fake images generated after 21,000 steps from the second run.

Model Run	**FID (5k steps)**	**FID (10k steps)**	**FID (15k steps)**	**FID (20k steps)**
Run 1 (Without attention)	229.08	198.29	–	–
Run 2 (With attention)	**157.13**	**135.05**	**137.74**	**119.49**

Table 1: FID results for the two GAN training runs

Figure 8: Fake images of cityscapes generated from the model at 10000 steps from Run 1.

These were manually determined to be the closest visually to cityscapes.

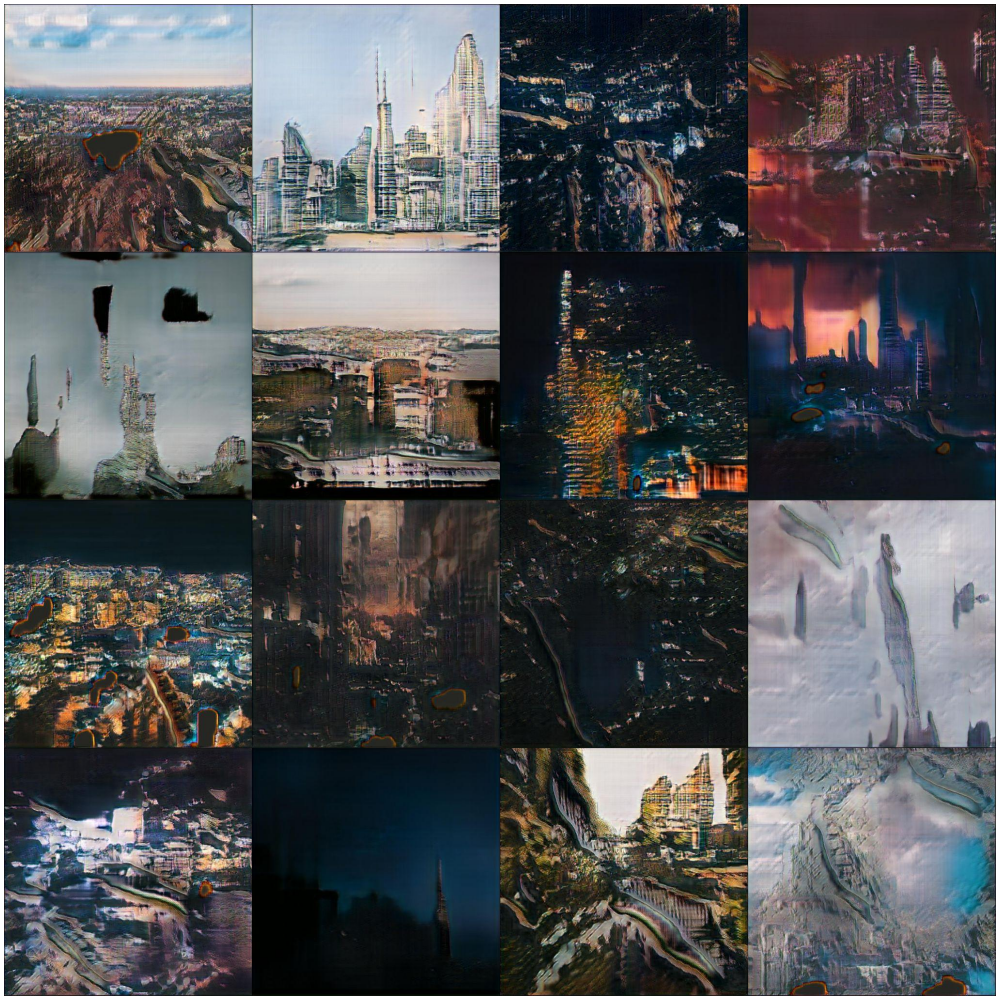

Figure 9: Fake images of cityscapes generated from Run 2. These were manually determined to be the closest visually to cityscapes.

b. Analysis

Despite the second run performing better than the first run, the results from both runs are quite poor. This is evidenced by the high FID values and the quality of images in Figures 8 and 9. Some potential reasons for the poor results are offered below:

i. The model simply did not train long enough. As mentioned above, the default number of steps was set at 150,000. We ran a very tiny fraction of this. With

increased training, we should expect the fake images generated to get better and better.

ii. High variance and noise in the input images: A second look at the dataset highlighted a number of images that are poor representations of cityscapes. These could potentially have influenced training. A way to address this could be to perform multiple runs of CLIP on the dataset to filter as many weeds out as possible.

iii. Poor initial hyperparameters: This is an especially important point as (Lucic et al., 2018) found that most models can reach similar scores as state-of-the-art models with enough hyperparameter optimization.

In line with the last point, I conducted a grid search on a few crucial hyperparameters for the Lightweight-GAN, computing the FID after 100 steps. This might help us figure out promising initial parameters for the model. The results are displayed in Table 2 below:

batch_size	learning rate	attention layers	disc_output_size	FID
10	0.0002	[32]	1	408.8257596
10	0.0002	[32]	5	403.3730726
10	0.0002	[]	1	442.2516765
10	0.0002	[]	5	424.140485
10	0.001	[32]	1	270.0307548
10	0.001	[32]	5	317.6589357
10	0.001	[]	1	285.8042333
10	0.001	[]	5	352.5651928
16	0.0002	[32]	1	379.5133043
16	0.0002	[32]	5	403.5309535
16	0.0002	[]	1	412.1824794
16	0.0002	[]	5	388.5250646
16	0.001	[32]	1	288.7485818
16	0.001	[32]	5	272.2296175
16	0.001	0	1	325.5064751

16	0.001	0	5	297.1486431

Table 2: Hyperparameter Search Results. The best FID obtained after 100 steps is highlighted in blue

From Table 2 above, we note that configurations that had a learning rate of 0.0002, as against 0.001 always had worse FID scores after 100 steps. This is not surprising because it represents a slower/lower learning rate. It is important to note that too high a learning rate could result in the model converging to a suboptimal solution

From Table 2, we also note that the inclusion of attention to a configuration always decreased the FID score, again corroborating the finding from the second run.

Note: 100 steps is an extremely small duration but I was constrained on time and couldn't run the hyperparameter optimization for longer. I, however, felt hyperparameter tuning was an interesting and important area to look at when dealing with GANs, hence, my decision to include it.

4. Conclusion & Next Steps

GANs are a powerful tool that has been utilized in a myriad of ways in the fashion industry. Although the GAN trained for cityscape generation did not produce highly realistic cityscapes, the images generated are quite hyperrealistic and arguably more artistic. As is often the case with GANs, the model will benefit from longer training and some more rigorous hyperparameter optimization. Attention in particular, seems to contribute positively to improvement in FID. Finally, the dataset could be enlarged and further cleaned to improve

the performance of the model.

5. References

Connah, K., David, G., & Hoon. (2020). *Anysize GAN: A solution to the image-warping problem*. https://arxiv.org/pdf/2003.03233v1.pdf

Dennis, C. (2020). AI-Generated Fashion Designs: Who or What Owns the Goods? AI-Generated Fashion Designs: Who or What Owns the Goods? *Fordham Intellectual Property, Media and Entertainment Law Fordham Intellectual Property, Media and Entertainment Law Journal Journal*, *30*. https://ir.lawnet.fordham.edu/cgi/viewcontent.cgi?article=1747&context=iplj

Gonsalves, R. A. (2021, April). *GANscapes: Create Impressionist Paintings with AI | Towards Data Science*. Medium; Towards Data Science. https://towardsdatascience.com/ganscapes-using-ai-to-create-new-impressionist-paintings-d6af1cf94c56

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., & Hochreiter, S. (n.d.). *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*. https://arxiv.org/pdf/1706.08500.pdf

Huggingface. (2017). *CLIP*. Huggingface.co. https://huggingface.co/docs/transformers/model_doc/clip

Johnson, K. (2020, December 7). *Nvidia researchers devise method for training GANs with less data*. VentureBeat; VentureBeat. https://venturebeat.com/2020/12/07/nvidia-researchers-devise-method-for-training-gans-

with-less-data/#:~:text=Training%20GANs%20can%20require%20upwards,to%2020%20times%20less%20data.

Liu, B., Zhu, Y., Song, K., & Elgammal, A. (n.d.). *Published as a conference paper at ICLR 2021 TOWARDS FASTER AND STABILIZED GAN TRAINING FOR HIGH-FIDELITY FEW-SHOT IMAGE SYNTHESIS*. Retrieved December 17, 2021, from https://openreview.net/pdf?id=1Fqg133qRaI

Luce, L. (2011). *Artificial Intelligence for Fashion*. Google Books. https://books.google.de/books?hl=en&lr=&id=ZRF-DwAAQBAJ&oi=fnd&pg=PR5&dq=artificial+intelligence+fashion+industry&ots=rKrv21AUi3&sig=biXC_TQBPKXD55L9m_yfaUmbv00&redir_esc=y#v=onepage&q=artificial%20intelligence%20fashion%20industry&f=false

Lucic, M., Kurach, K., Google, M., Bousquet, B., & Gelly, S. (2018). *Are GANs Created Equal? A Large-Scale Study*. https://arxiv.org/pdf/1711.10337.pdf

Ping, Q., Wu, B., Ding, W., & Yuan, J. (2019). *Fashion-AttGAN: Attribute-Aware Fashion Editing with Multi-Objective GAN*. https://openaccess.thecvf.com/content_CVPRW_2019/papers/FFSS-USAD/Ping_Fashion-AttGAN_Attribute-Aware_Fashion_Editing_With_Multi-Objective_GAN_CVPRW_2019_paper.pdf

Wang, C., & Xiao, Z. (2021). Lychee Surface Defect Detection Based on Deep Convolutional Neural Networks with GAN-Based Data Augmentation. *Agronomy*, *11*(8), 1500. https://doi.org/10.3390/agronomy11081500

Welander, P., Karlsson, S., & Eklund, A. (2018). *GENERATIVE ADVERSARIAL NETWORKS FOR IMAGE-TO-IMAGE TRANSLATION ON MULTI-CONTRAST MR IMAGES -A COMPARISON OF CYCLEGAN AND UNIT*. https://arxiv.org/pdf/1806.07777.pdf

Pexels (n.d.). Cityscape. https://www.pexels.com/search/cityscape/

6. Appendices

- **Google Drive/Colab Folder (Main Folder)**
- **Code**
  a. Pexels Data Extraction code
  b. Image preprocessing (resizing)
  c. Image filtering (using CLIP)
  d. Modeling: First run without attention mechanism
  e. Modeling: Second run with attention mechanism
  f. Hyperparameter search