The Andrew & Erna Viterbi Faculty of Electrical Engineering Technion — Israel Institute of Technology

Electronics
Computers
Communications

VISL

Vision and Image
Sciences Laboratory

Project B final paper

Winter 2021

Cloud synthesis using GANs

Jad Taya

Matti baer

Supervised by:

Adi Vainiger

Submitted November 2021

Table of Contents

Table of figures

Abbreviations

CNN	-	Convolutional Neural Network
GAN	-	Generative Adversarial Network
LWC	-	Liquid Water Content
MSE	-	Mean Squared Error
NZE	-	Non-Zero Elements
NZP	-	Non-Zero Penalty
WGAN	-	Wasserstein Generative Adversarial Network
WGAN-GP	-	Wasserstein Generative Adversarial Network – Gradient Penalty

Abstract

Despite having a profound effect on our lives, clouds remain one of the natural phenomena that scientists struggle to understand. We might be able to better predict their behavior and weather with a better understanding of their inner structure. Research at the Hybrid Imaging Lab at the Technion focuses on developing an algorithm that can provide information on the inside structure of clouds, based on images taken from different angles. This method is known as computed tomography[1] (CT). In order to develop this algorithm, many cloud samples are necessary, which can be problematic because physics-based cloud samples (containing their internal structure) are hard to simulate. This project attempts to synthesize cloud samples using a new method, namely GANs.

1. Introduction

Current methods of acquiring cloud samples involve simulating a cloud field using a physical simulator, the field changes with time and samples can be taken at different times. Physical simulations take a considerable amount of time, this makes producing a large number of samples quite the difficult task.

Recent years have seen a surge of interest in generative networks. This method entails feeding random vectors to a neural network, which outputs the desired sample as a result. GANs[2] are a powerful class of generative models characterized by two networks - a generator network generates synthetic data given some noise and a discriminator network distinguishes between the generator's output and the real data. GANs can produce very visually appealing samples at a rapid rate, but they can be difficult to train.

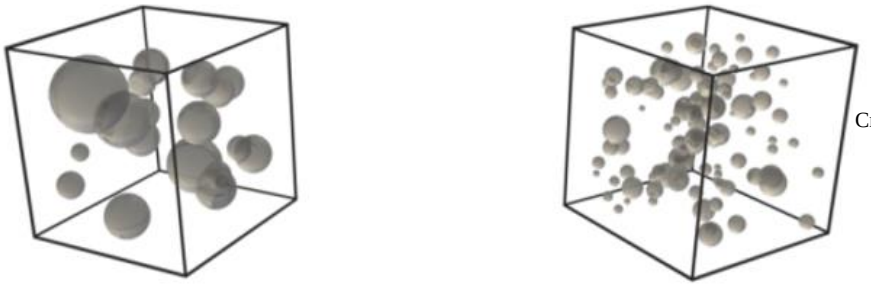
In this paper we attempt to train different GAN models to generate 2D slices of clouds samples.

2. Theoretical background

2.1 Clouds

As most of us already know clouds consist mostly of water droplets and air, but in order to create usable data that can later be plugged into an algorithm we must describe this inner structure numerically. In order to do this, we divide our 3D space into cubes with identical volume which we call Voxels (3D version of a pixel), and attribute microphysical properties to each one:

1. density – the number of water droplets per cubed meter in the voxel, denoted- $n[m^{-3}]$
2. extinction cross section – the average area of a water droplet it the voxel, denoted - $\sigma[m^2]$
3. cross section variance – the variance of the areas of water droplets in the voxel, denoted – $v_e[m^2]$



Credit: Aviad Levis

Figure 1: Water droplets in a voxel

Besides the microphysical parameters there are optical parameters as well, these parameters describe how light interacts with the voxel and can be calculated directly using the microphysical parameters. The reason these optical parameters interest us is because computed tomography analyzes the light that goes through the cloud in order to infer its inner structure, therefore the parameters they can infer are optical. The most important optical parameter needed for the computed tomography is the extinction coefficient $\beta[m^{-1}] = \sigma \cdot n$, which is the extinction of the light per meter traveling through the voxel.

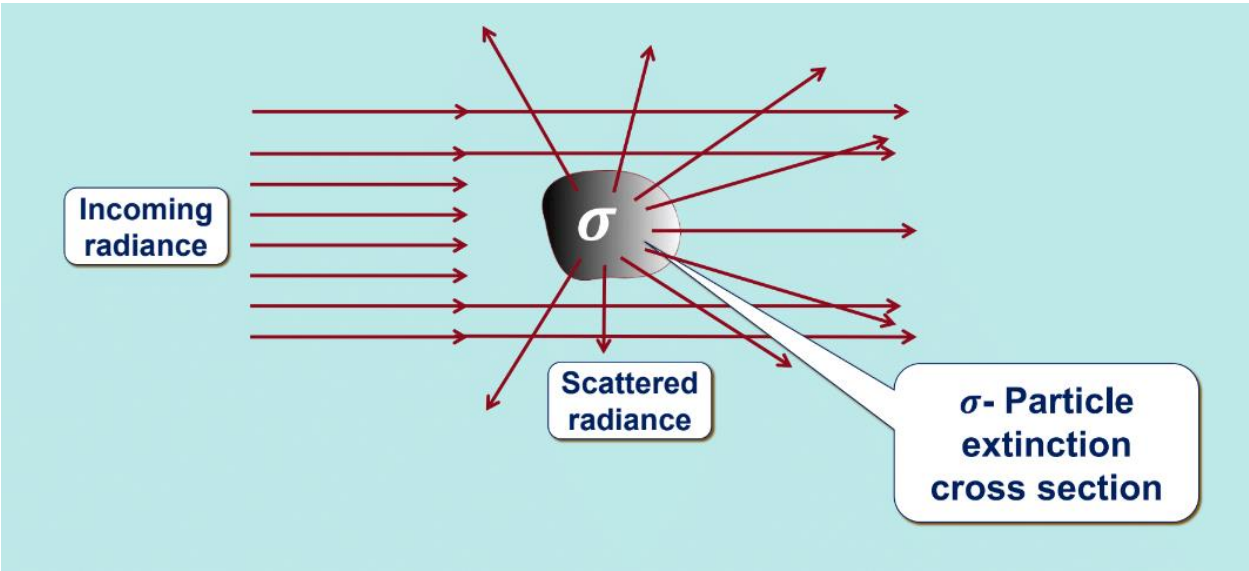


Figure 2: Extinction cross section and scattering radiance

Credit: Adi Vainiger

One of the important qualities of clouds is the connection between the LWC (liquid water content) and the height[3]. This means that the areas on the higher parts of the cloud will have greater water density (n) then the lower parts, which in turn means the extinction coefficient

becomes greater the higher we are in the cloud as well. This is caused by the physical process that drives the formation of the clouds and can be mathematically described by: $\beta \propto (z - z_0)^{\frac{2}{3}}$ (where z is the height and z_0 is the height of the base of the cloud). When synthesizing clouds, we would like to see this physical quality remain in the generated clouds as well. So later, when we prepared the training data, we made sure to slice the 3D clouds into 2D ones along the XZ and YZ axis only.

2.2 Texture synthesis

Having assumed that clouds are some sort of texture, we decided to study the latest trends in texture synthesis first:

2.2.1 Texture synthesis using Gram matrices

In this article[4] they use a standard VGG-19 CNN to try and "imitate" the texture of a given image. This is done by using the feature maps that are created at each layer of the network, the idea is that the auto correlation of the feature maps of a given image hold the information of its texture. In order to do this the input image is forward passed in a pre-trained VGG-19 CNN, which creates several feature maps at each layer denoted F_{ik}^L (this is the k'th element of the i'th feature map in layer L). Then the auto correlations between the maps of each layer are calculated and put into matrix elements $G_{ij}^L = \sum_k F_{ik}^L F_{jk}^L$ (Gram matrices). After that, white noise is plugged into the network and its Gram matrices are calculated, then the input is gradually changed using gradient decent on the following loss: $E_L = \sum_{ij} (G_{ij}^L - \widehat{G_{ij}^L})^2$ for each layer and a total loss of $L = \sum_L w_L \cdot E_L$ (where w_L are weights given to each layer). Meaning at each step $\vec{x} = \vec{x} - \alpha \frac{\partial L}{\partial \vec{x}}$ (where $\vec{x}$ is the input image), convergence happens when the input image has similar auto correlations to the original image. This process can be visualized in the following figure.

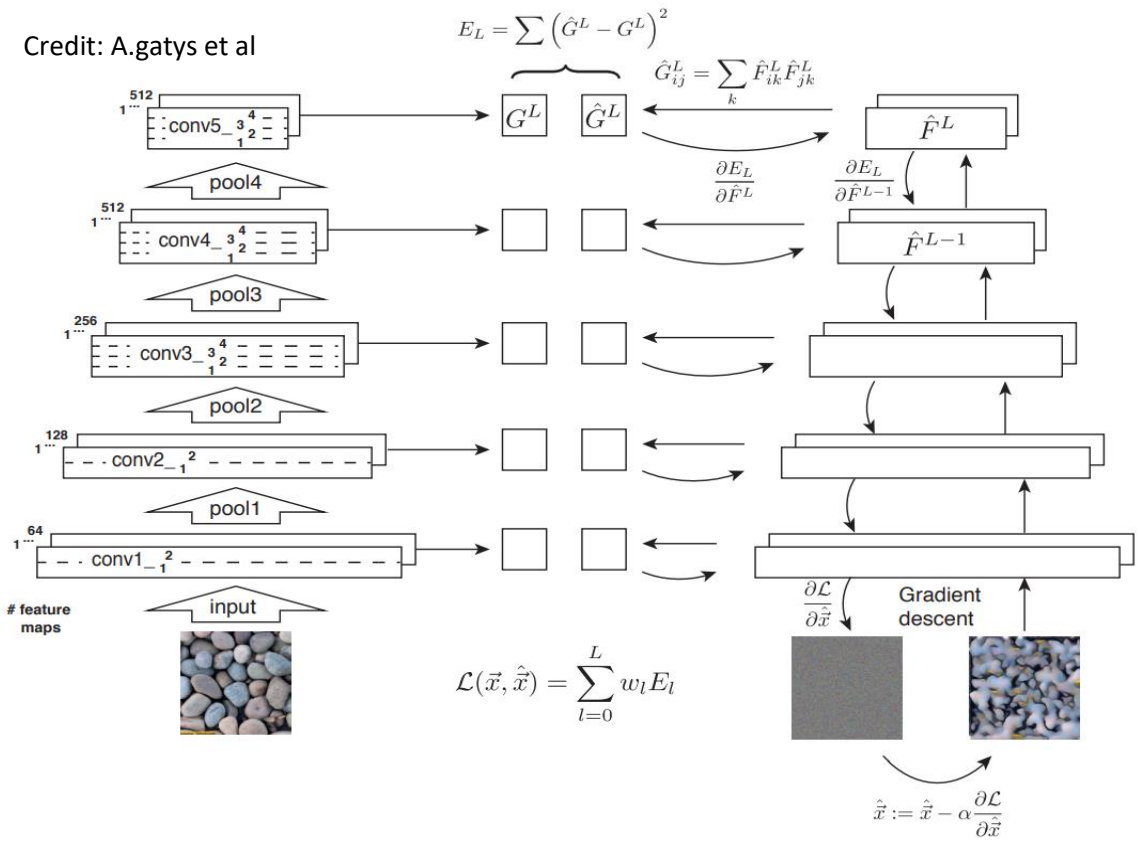


Figure 3: Schematic layout of the gram matrix algorithm

2.2.2 Spatial GANs

In this article[5] they take an ordinary DCGAN and make one minor change to it, instead of one latent vector input to the generator it is possible to insert an array of latent vectors into the generator (see Figure 4) resulting in a larger output image. This has two useful features, the first being the ability to generate different size images using the same architecture and the second is the ability to add or remove layers from our network and stay with the same size of output image by adjusting the latent vector size accordingly. The perspective field, the number of output pixels influenced by a single pixel in the input vector, increases the more layers a network has, so the size of the perspective field can also be controlled this way. In the article they show that if the perspective field is too small the network cannot learn high level textures since pixels in the output image that are far away from each other won't be influenced by the same pixels.

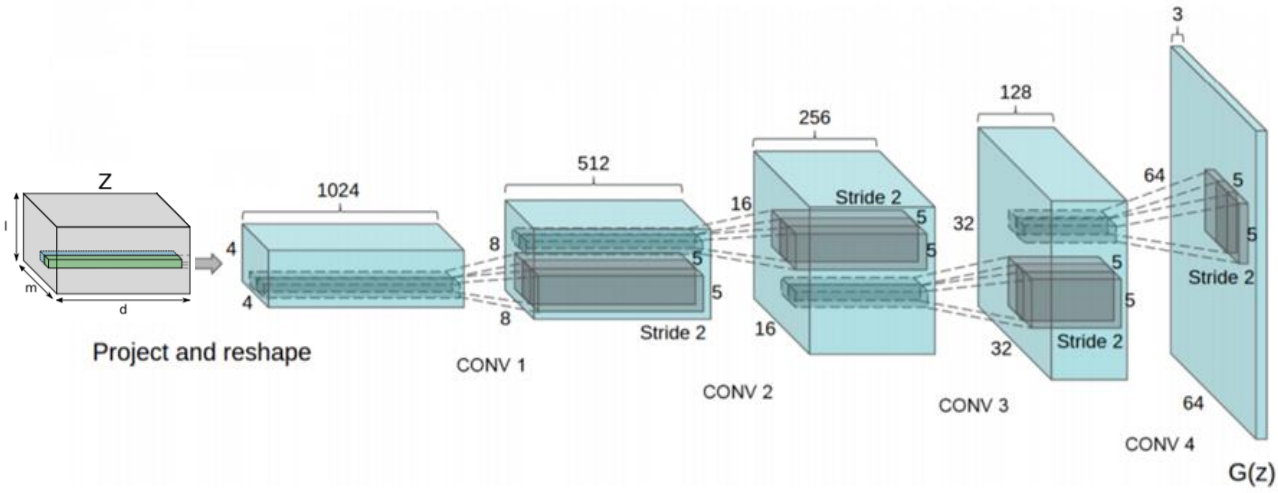


Figure 4: Spatial GAN architecture

2.2.3 Non-stationary texture synthesis by adversarial expansion

In this article[6] they aim to expand a given image that contains a non-stationary texture. A non-stationary texture is a texture that doesn't have cyclicality, which typically makes it harder to synthesize. In order to do this the input image is cropped and plugged into the generator of a GAN which outputs an expanded image with the same size as the original uncropped image, then the discriminator receives the larger image and calculates its loss. The main difference from an ordinary GAN is that the generators loss is a combination of a few different losses: $L_{genrator} = L_{adv} + \gamma L_1 + \theta L_{style}$. L_{adv} is the ordinary GAN loss, L_1 is the L_1distance between the original image and the synthesized expansion & the L_{style} loss is the Gram matrices loss that we saw in the first article. The main idea we took from this article was the ability to add more losses to the default generator loss to influence the generator's output if we see that it can't fully learn some things by itself. The outline of this article's architecture can be seen in Figure 5.

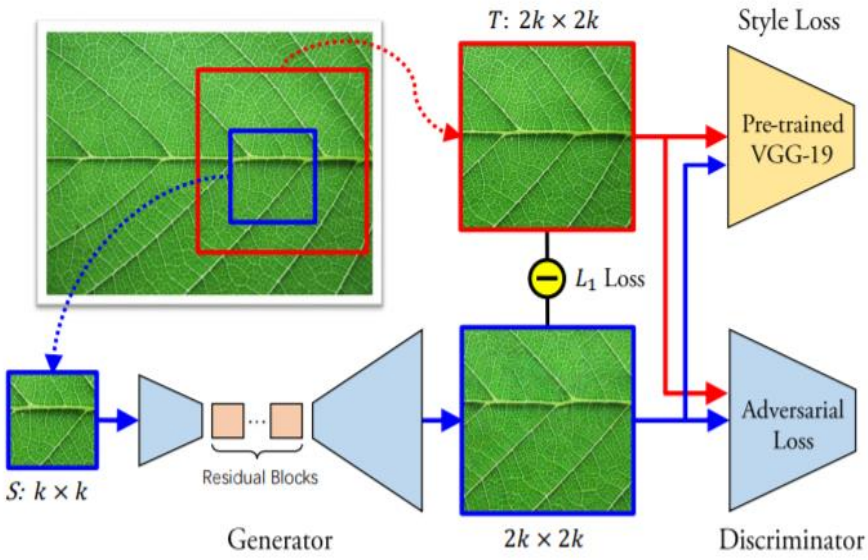


Figure 5: Adversarial expansion schematics

2.3 Generative adversarial networks

A GAN (generative adversarial network) is a set of two neural networks that work together in order to synthesize data that resembles the training data.
The way this works is that there are two neural networks, one is called the generator and the other one is called the discriminator, the purpose of the generator is to generate data that resembles the training data while the purpose of the discriminator is to receive data (either from the generator or the training set) and decide whether its real (training set) or fake (from the generator).
A GAN's networks are designed to self-adapt over time, so by the end of the training, the generator produces data that is similar to the training set.

2.3.1 Mathematical formulation - DCGAN

Suppose our training set comes from a distribution $P(x)$ our purpose is to generate data from the same distribution. In order to do this, we plug in a vector of gaussian white noise Z to the generator network and denote the output $G(Z)$. After this we plug in to the discriminator data from the training set or data from the generator and denote the output $D(X)$ or $D(G(z))$ consecutively. And finally using the loss function:

(1) $$E_{X\sim P(X)}\left[log\left(D(X)\right)\right] + E_{Z\sim P(Z)}\left[log\left(1 - D\left(G(Z)\right)\right)\right]$$

We train the networks, where the discriminator adapts its wights in order to maximize this function and the generator adapts its wights in order to minimize it. It can be shown that if D & G can be any function, the G that minimizes this expression:

(2) $$min_G max_D \left(E_{X\sim P(X)}[log(D(X))] + E_{Z\sim P(Z)}\left[log\left(1 - D(G(Z))\right)\right]\right)$$

Will maintain - $G(Z) \sim P(X)$

Which means we succeeded in generating data from $P(X)$. Obviously, G & D cannot be any function but with a big enough network we can get close.

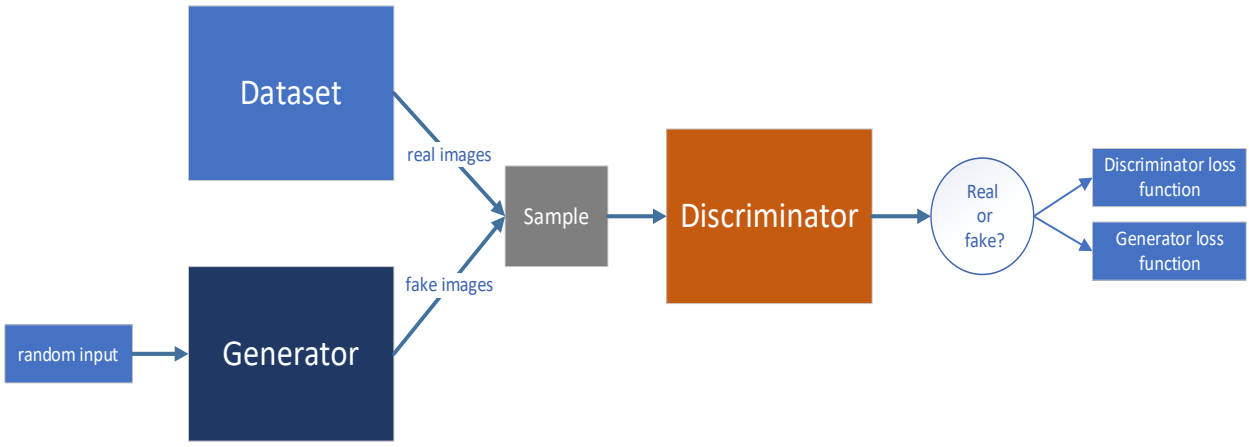


Figure 6: GAN outline

2.3.2 Vanishing gradient:

For the Generator to minimize the above function it must only minimize the second term: $E_{Z\sim P(Z)}\left[log\left(1 - D(G(Z))\right)\right]$, since it has no affect on the first term. The issue with this is that the gradient of this term is very small at the beginning of the training and this doesn't allow the generator to adapt its wights properly. In order to solve this issue, we instead train the Generator to maximize $E_{Z\sim P(Z)}\left[log\left(D(G(Z))\right)\right]$, which yields the same results but avoids the issue of the vanishing gradient.

2.3.3 GAN algorithm:

At first, we decide on a batch size n, number of epochs t & learning rate r.

We start with random discriminator wights ω & generator wights θ.

While $(epoch \leq t)$:

Sample n real images → $\{x_i\}_1^n$

Create n white noise vectors → $\{z_i\}_1^n$

Plug the white noise vectors into the generator → $\{G(z_i)\}_1^n$

Plug the real and fake images into the discriminator

→ $\{D(x_i)\}_1^n, \{D(G(z_i))\}_1^n$

Calculate the discriminators loss:

→ $DL = \frac{1}{n}\sum_1^n Log\big(D(x_i)\big) + Log(1 - D\big(G(z_i)\big)$

Update the discriminators wights: $\omega = \omega + r \cdot optimizer(\nabla_\omega DL, \omega)$

Plug new white noise vectors into the generator → $\{G(z_i)\}_1^n$

Calculate the Generator loss: → $GL = \frac{1}{n}\sum_1^n Log(D\big(G(z_i)\big)$

Update the Generators wights: $\theta = \theta + r \cdot optimizer(\nabla_\theta GL, \theta)$

End while

2.3.4 Mathematical formulation - Wasserstein GAN (WGAN)

WGAN[7] is a form of GAN that uses the earth mover distance (EM) in order to try and converge $P(G(Z)) \to P(x)$.

The EM distance is defined by:

(3) $$W\big(P_r, P_g\big) = \inf_{\gamma \in \Pi(P_r,\ P_g)} E_{(x,y)\sim\gamma}\left[\|x - y\|\right]$$

Where $\Pi(P_r,\ P_g)$ denotes the set of all distributions $\gamma(x, y)$ whose marginals are P_r *and* P_g . Intuitively $\gamma(x, y)$ indicates how much "mass" must be transported from x to y in order to transform the distributions P_r *into the distributions* P_g. The EM distance then is the "cost" of the optimal transport plan.

The Kantorovich-Rubinstein duality tells us that:

(4) $$W(P_r, P_g) = \sup_{\|f\|_L \leq 1} E_{x \sim P_r}[f(x)] - E_{x \sim P_g}[f(x)]$$

Where $\|f\|_L \leq 1$ denotes all 1- Lipschitz functions.

And so, to minimize the Wasserstein distance between the training data and generated data in our GAN we need to train a generator that minimizes it:

(5) $$\min_{G(z)} \{W(P_R, P_{G(z)})\} = \min_{G(z)} \left\{ \max_{\|D\|_L \leq 1} \left(\mathrm{E}_{x \sim P_R}[D(x)] - \mathrm{E}_{x \sim P_{G(z)}}[D(x)] \right) \right\}$$

The idea behind the WGAN is to get the generator to bring this distance to 0, which brings us back to a similar min max problem like the original GAN. Using NN we can write out this distance: $W(P_r, P_g) = \frac{1}{n}\sum_{i=1}^{n} D(x_i) - D(G(z_i))$. but in order to make sure the discriminators function is a 1-Lipschitz the authors of the original WGAN paper chose to use weight clipping during the training.

2.3.5 Wasserstein GAN – Gradient Penalty (WGAN-GP)

Weight clipping leads to optimization difficulties[8], so a different way to make sure the discriminator is 1-Lipschitz is proposed in the paper "Improved training of Wasserstein GANs", namely gradient penalty:

(6) $$\mathrm{E}_{x \sim P_R}[D(x)] - \mathrm{E}_{x \sim P_{G(z)}}[D(x)] - \lambda \mathrm{E}_{x \sim P_u}\left[\left(\|\nabla D(x)\|_2 - 1 \right)^2 \right]$$

The gradient penalty is calculated on elements that are on a straight line between both distributions, the real and fake ones. This is done by blending elements from both distributions, this is done using the following equation:

(7) $$P_u = \varepsilon \cdot P_R + (1 - \varepsilon) \cdot P_{G(z)}$$
$$\varepsilon \sim U[0,1]$$

2.3.6 WGGAN-GP algorithm

For each batch:

- Sample n real images $\{x_i\}_1^n$
- Generate n fake images $\{G(z_i)\}_1^n$
- Sample n random numbers $\{\epsilon_i \sim U[0,1]\}_1^n$
- *Calculate n blended images* $\{y_i = \epsilon_i \cdot x_i + (1 - \epsilon_i) \cdot G(z_i)\}_1^n$
- for t=1,…,m
    - Plug the real and fake images into the discriminator & Calculate the discriminators loss:
    - $DL = \frac{1}{n} \sum_1^n D(x_i) - D\big(G(z_i)\big) + \lambda(\|\nabla_\omega D(y_i)\| - 1)^2$
    - Update the discriminators wights: $\omega = \omega + r \cdot opt(\nabla_\omega DL, \omega)$
- Generate new fake images and calculate the generator loss:

  $GL = \frac{1}{n} \sum_1^n Log(D\big(G(z_i)\big)$
- Update the generators wights: $\theta = \theta + r \cdot opt(\nabla_\theta GL, \theta)$
- End

3. Database

At the start of the project we made a decision to generate 2D "images" where each pixels value represents the extinction coefficient $\beta[m^{-1}]$. Obviously, the end goal is to generate 3D clouds with all 3 microphysical parameters, but in the scope of this project we decided to focus on obtaining good results for a simpler version. And hopefully our ideas can then be used on more complex versions.

In order to train our GAN we needed to create a training set of 2D "images" of β. Initially we received a small data set of 3D clouds 32x32x32 obtaining the extinction coefficient in each voxel, we sliced these 3D images in to 2D slices along the Z-X & Z-Y axis in order to create our training set. The reason we didn't slice along the X-Y axis as well was because the height of the voxel has physical importance as we mentioned earlier.

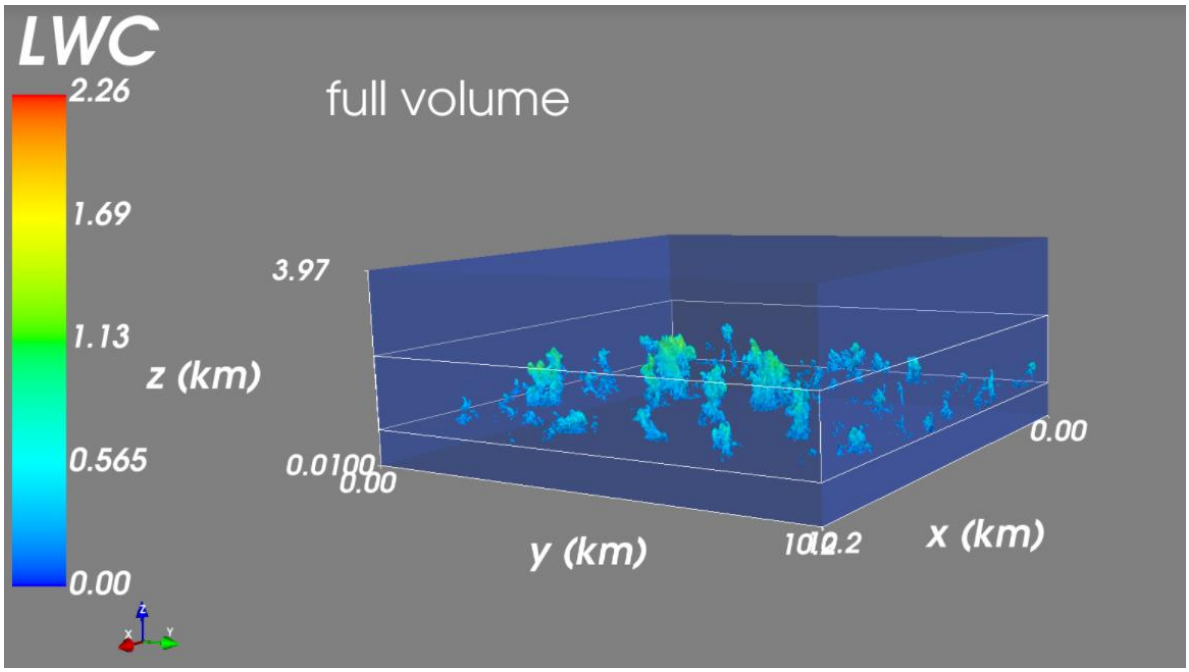


Figure 7: Cloud field generated by physical simulator

After training the GAN on this small data set, we realized we need a larger data set in order to achieve better results, which led us to extract the training set from the physical simulator directly. The physical simulator generates large 3D cloud fields that contain all 3 microphysical parameters (see Figure 7), the first step was to run a script that uses the Pyshdom library[9] to calculate the optical parameters from the microphysical parameters, so the script takes the field and calculates the extinction coefficient in each voxel returning a cloud field with the optical parameter alone. After that we manually cut out each cloud from the field to obtain single clouds (see Figure 8), making sure the base height is identical to all clouds. And at last, slicing the clouds

into 2D images as we did with the first batch (see Figure 9). We needed to make sure all the images were the same size so we zero padded our images to fit the largest one, ultimately creating training set of 2D images size 130x80, we chose this size because 99% of the images fit into these dimensions, the last 1% were discarded as they were anomalies with much larger dimensions.

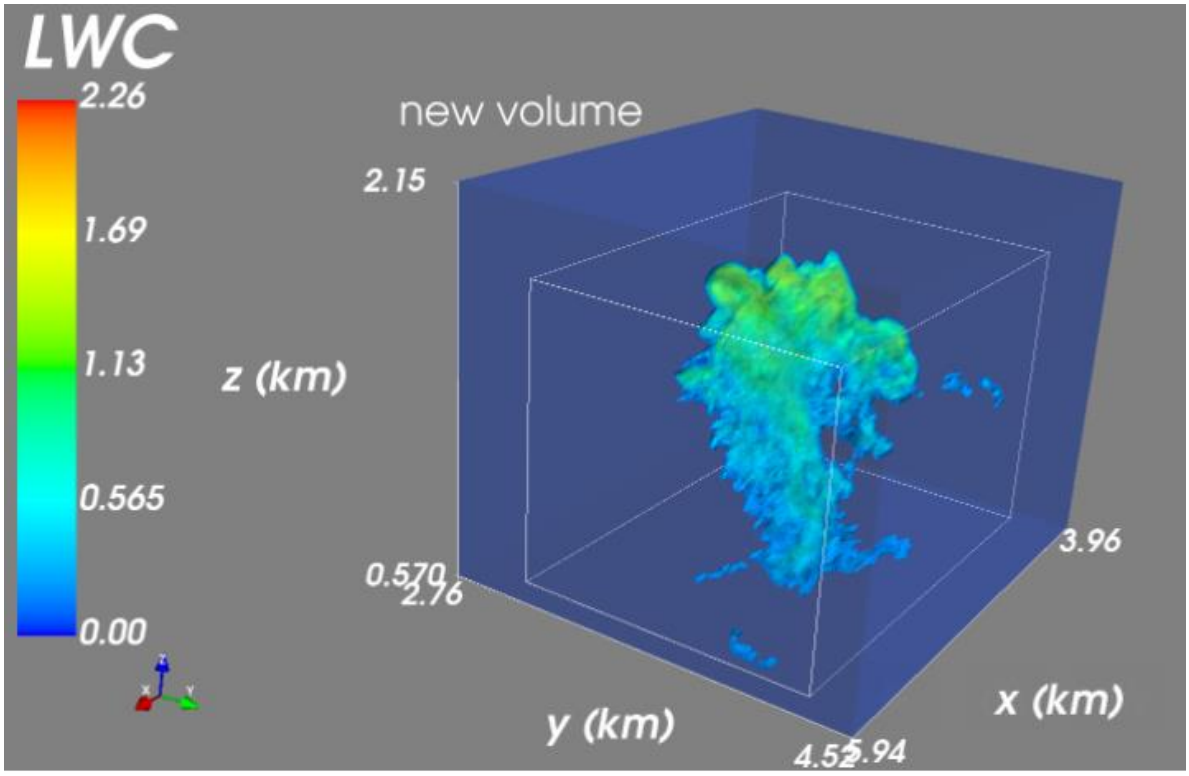


Figure 8: Single cloud cropped out of field

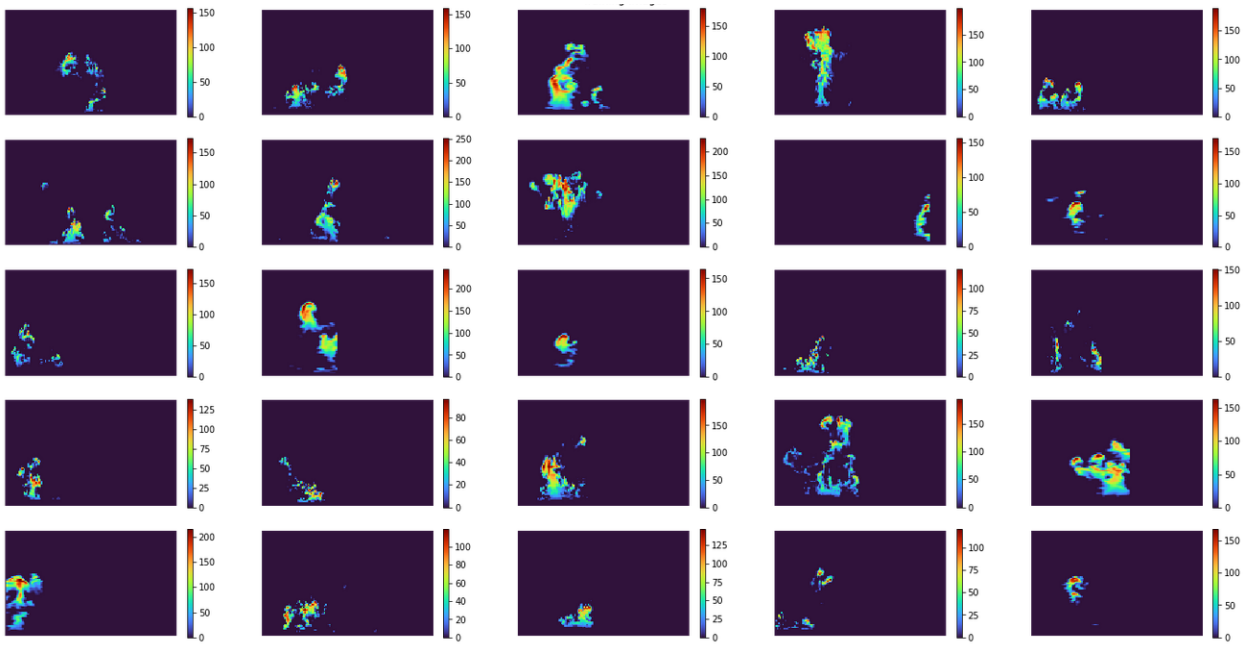


Figure 9: 2D cloud slices cut from simulated cloud field

4. The suggested models

4.1 DCGAN

The first model we tried was the DCGAN[10], this GAN uses the same algorithm as the basic GAN but the architecture of the network is a bit different. Instead of having fully connected NN as the discriminator and generator we use CNN (convolutional neural network), meaning in each layer the image goes through a 2D convolution with the weights (filters). this architecture has been proven to be better for images that have spatial qualities.

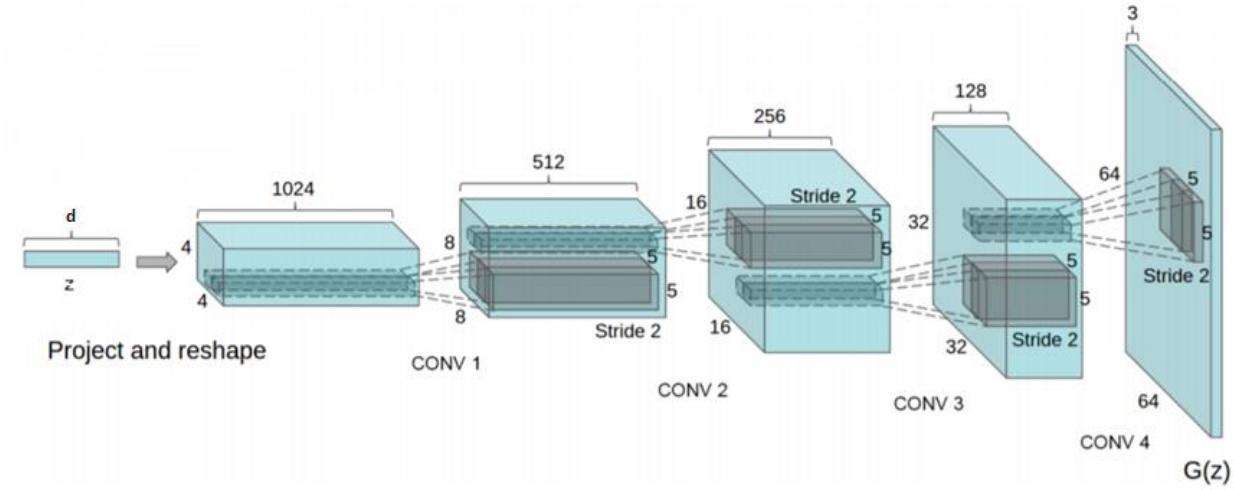


Figure 10: DC GAN architecture

4.2 Improved DCGAN

After testing our DCGAN we realized we must make a few improvements. The first was to reduce the number of learned parameters (weights) from ~46 mil to ~12 mil. this helps the training in two ways: reducing the training time and avoiding overfitting. The second improvement was to add an MSE loss to the loss function in the following way:

$$loss\ G = \log\big(D(G(z))\big) + \alpha \cdot MSE\left(mean\big(G(z)\big), mean(x)\right) \tag{8}$$

This term is meant to normalize our generated data making sure that it is in the typical range of values as the GT database. The final network had the architecture we can see in Figure 10 with the following depths in each layer: [16, 32, 64, 128, 256].

4.2 WGAN

For the WGAN the same architecture was used with slight changes to the generators depth resulting in the following network depths:

Gdepth = [512, 256, 128, 64, 32]

Ddepth = [16, 32, 64, 128, 256]

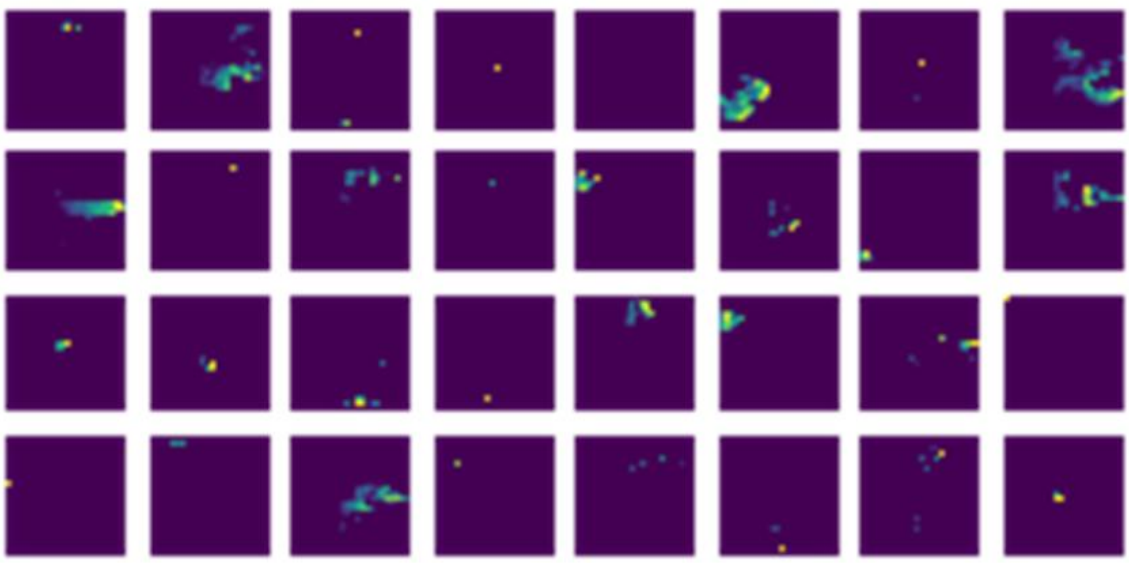
The algorithm was obviously changed as well to fit the WGAN algorithm with the loss function seen in eq (6).

5. Results

5.1 DCGAN

Initially we received a small data set of images size 32x32, we used the to train the DCGAN and received the following results:

Original Data set (32x32):



(Credit: Yael Sde Chen)

Figure 11: 2D slices of simulator generated clouds

Generated Data (32x32):

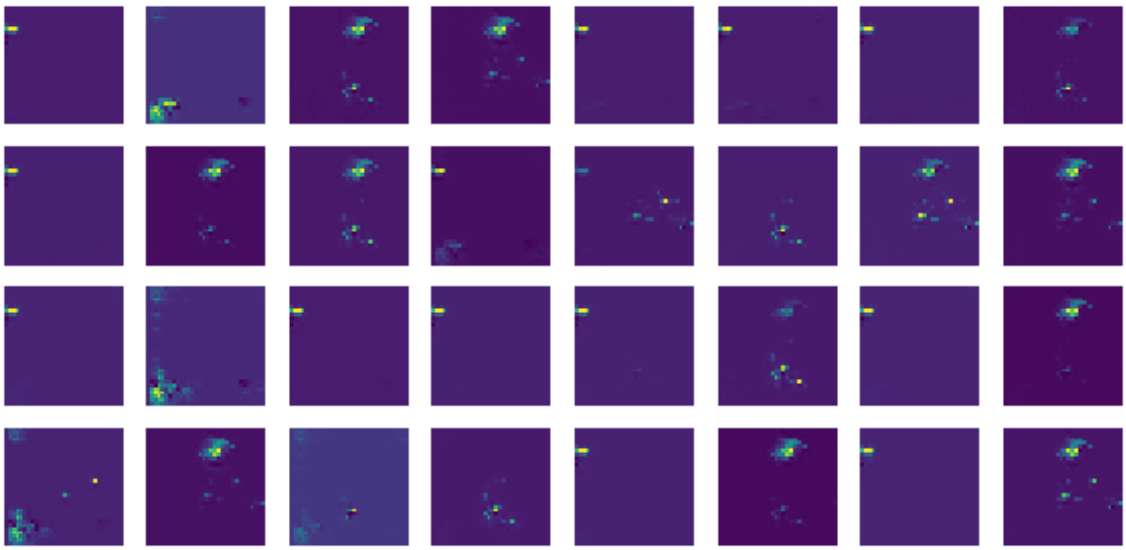


Figure 12: 2D clouds generated from initial DCGAN

Looking at the generated images we can see they are quite repetitive (the same image is generated many times), this is a common issue that occurs when training GANs with a small data set. We concluded that in order the Generate a high verity of realistic images we must have a larger data set to work with.

In order to create a larger data set, we manually cropped clouds out of larger cloud fields generated by the physical simulator, eventually creating a data set of clouds size 130x80.

Training the GAN with these clouds yielded the following results:

Original Data set (130x80):

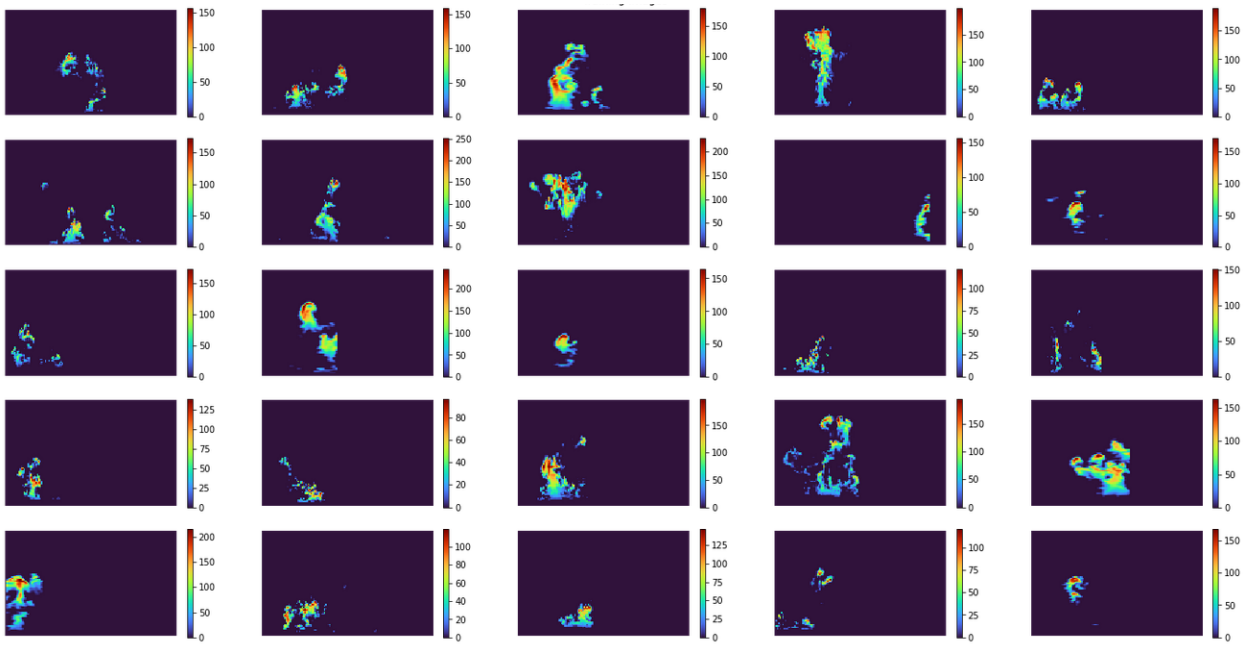


Figure 13: 2D cloud slices cut from simulated cloud

Generated Data set (130x80):

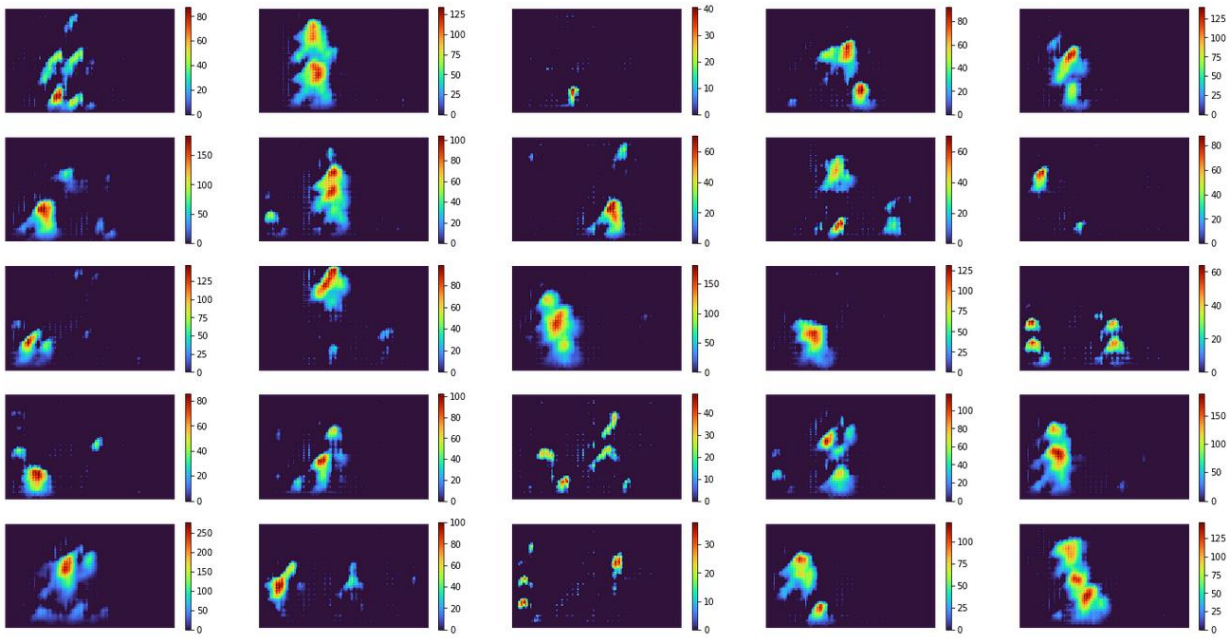


Figure 14: 2D cloud slices generated from first DCGAN with large data set

After observing these results, it seemed we were moving in the right direction, but we were still unsatisfied with the quality of the generated images. In order to try to generate higher quality images we added MSE loss (see (8)) to the generator loss function with an alpha parameter of 1 and significantly lowered the number of parameters in our network.

Generated Data set (130x80):

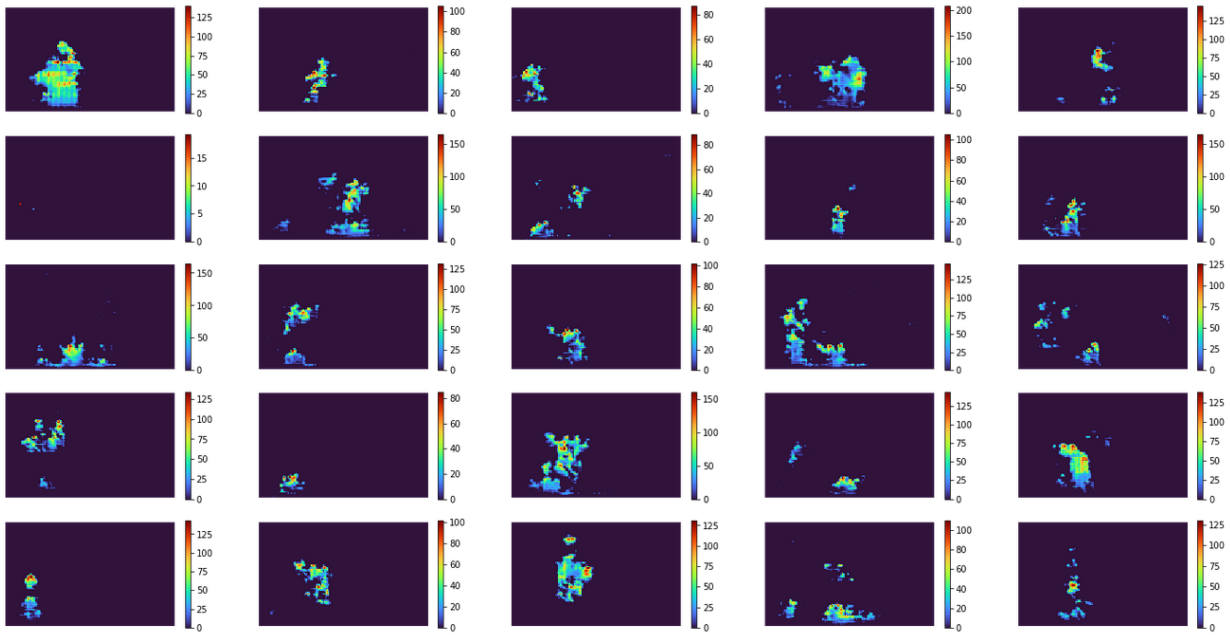


Figure 15: 2D cloud slices generated from improved DCGAN

These images have a much higher resemblance to the original data, but we still wanted to try another method to see if we can generate more accurate images.

5.2 WGAN

In order to try and generate images that are more photo realistic we decided to use WGAN with the following loss function:

(9)
$$loss\ D = D\big(G(z)\big) - D(x)$$
$$loss\ G = -D\big(G(z)\big)$$

We received the following results:

Generated Data set (130x80):

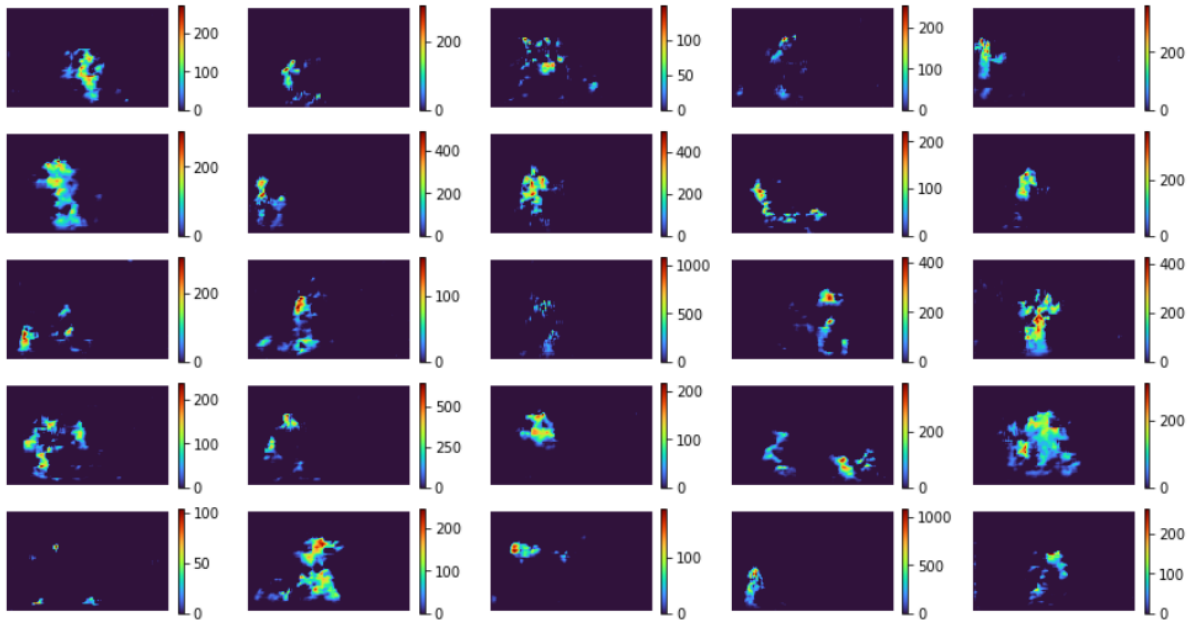


Figure 16: 2D cloud slices generated from first WGAN

These images are more photo realistic than the DCGAN, but in order to assess how good the generated images are the visual quality is not enough. Since we are trying to generate data that resembles physical clouds, we cannot judge our results purely on how photo realistic they are.

In order to try and measure the accuracy of our generated images we collected statistics of our generated data and compared it to the original data:

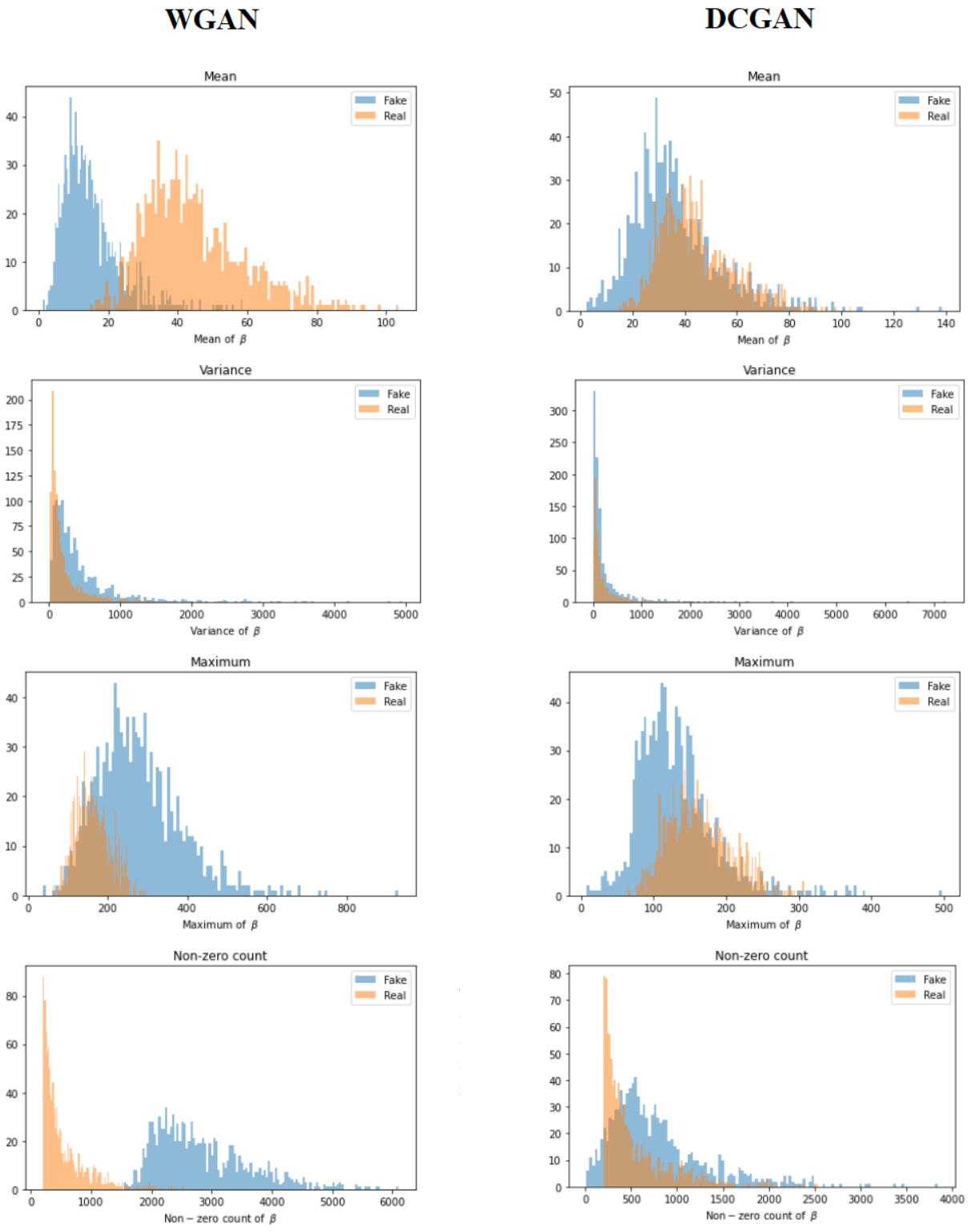


Figure 17: Beta statistics comparing real and fake data for WGAN & DCGAN

Looking at these statistics we can see that even though the WGAN achieves images that are more photorealistic, the statistics do not match that of the original data, we can see that the graphs in

the bottom left aren't aligned at all, most of the real clouds have a count of non-zero elements between 200 and 2000, while most of the fake clouds have a count between 1700 and 5000. We also see that the maximum and mean graphs are mis-aligned.

Moving forward we decided to synthesise clouds using WGAN-GP because of the advantages it has over standard GANs (like DCGAN), seemingly only having one disadvantage which is the slower training time. However, neither DCGAN nor WGAN-GP could accurately capture the physical properties of clouds when generating images. Adding constraints on things such as the number of nonzero elements in an image has helped bring the generated clouds closer to the ground truth.

We noticed that the graph for the nonzero elements in the training images looked like a gamma distribution shifted to the right 201, as that is the lower bound:

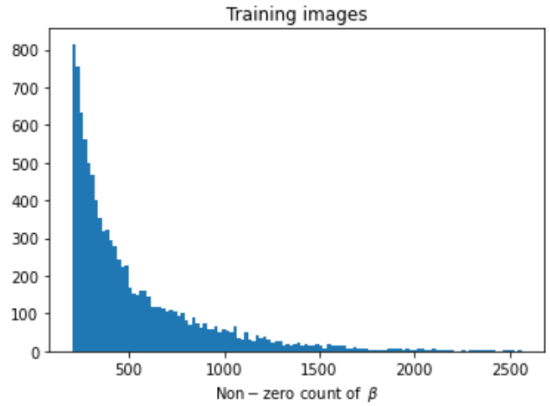


Figure 18: Number of non-zero optical density elements histogram for the training set

We shifted the histogram to the left 201 places (201 counts) so that we can find its gamma distribution parameters:

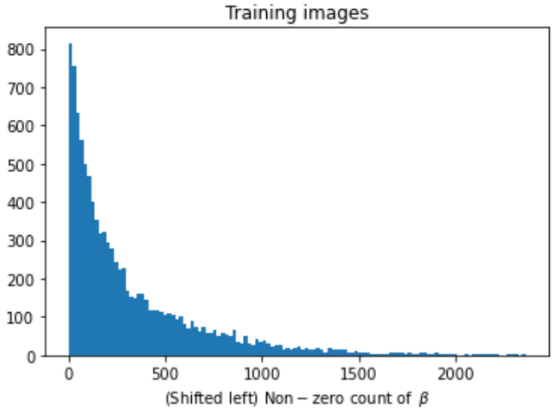


Figure 19: Figure 15 shifted left to start at 0

We find the parameters of the gamma distribution that fits the data (data being the non-zero elements count), from now on we will call that distribution P.

Using P, we will now add a loss element to the generator's loss, the goal of this element is to punish generated clouds that are less likely to exist according to the distribution P, for example the loss will increase for clouds that have a count of non-zero elements outside the range [201, 2563]. Clouds will not incur any NZP (Non-Zero Penalty) when they have 201 elements, the penalty will increase the further away we get from 201.

Using the loss functions:

(10)
$$loss\ D = D\big(G(z)\big) - D(x) + \lambda \cdot GP(G(z), x)$$
$$loss\ G = -D\big(G(z)\big) + \theta \cdot NZP(G(z))$$

Where:

(11)
$$NZP(x) = \begin{cases} 1 & f(x) < 201 \\ 0.2 - P(f(x) - 201) & 2563 > f(x) \geq 201 \quad f(x) \\ 1 & 2563 \leq f(x) \end{cases}$$

$$f(x) = number\ of\ non\ zero\ elements\ in\ x$$

201 and 2563 are respectively the upper and lower boundaries for the number of NZE in all the images in the training set.

We received the following results using the parameters $\lambda = 10$ and $\theta = 20000$:

Generated Data set (130x80):

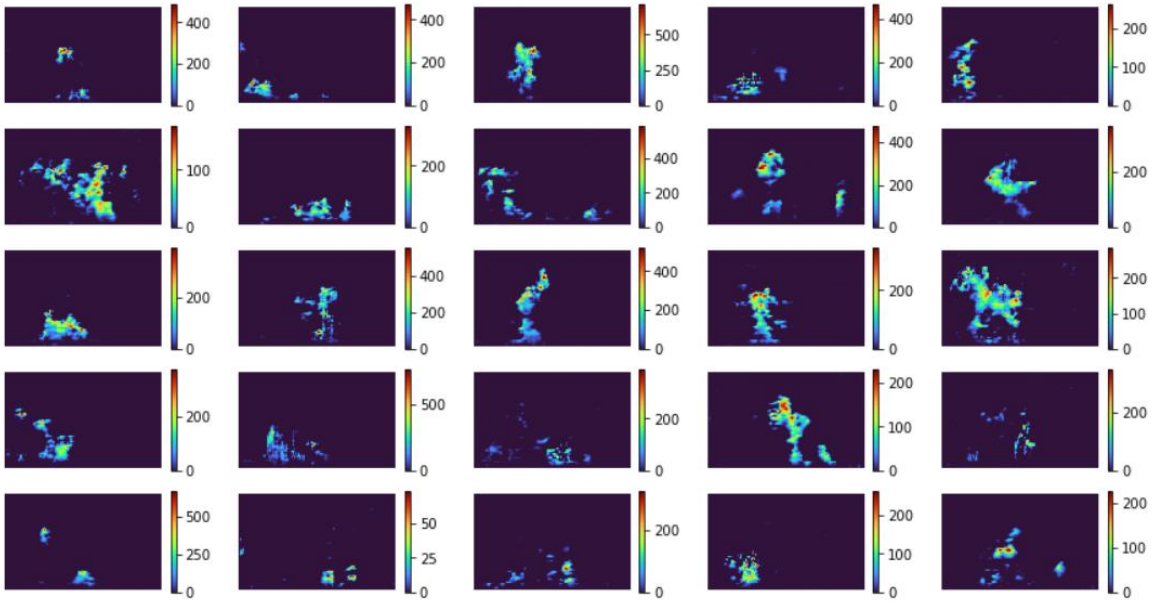


Figure 20: 2D cloud slices generated by WGAN with NZE loss addition

Generated Data statistics:

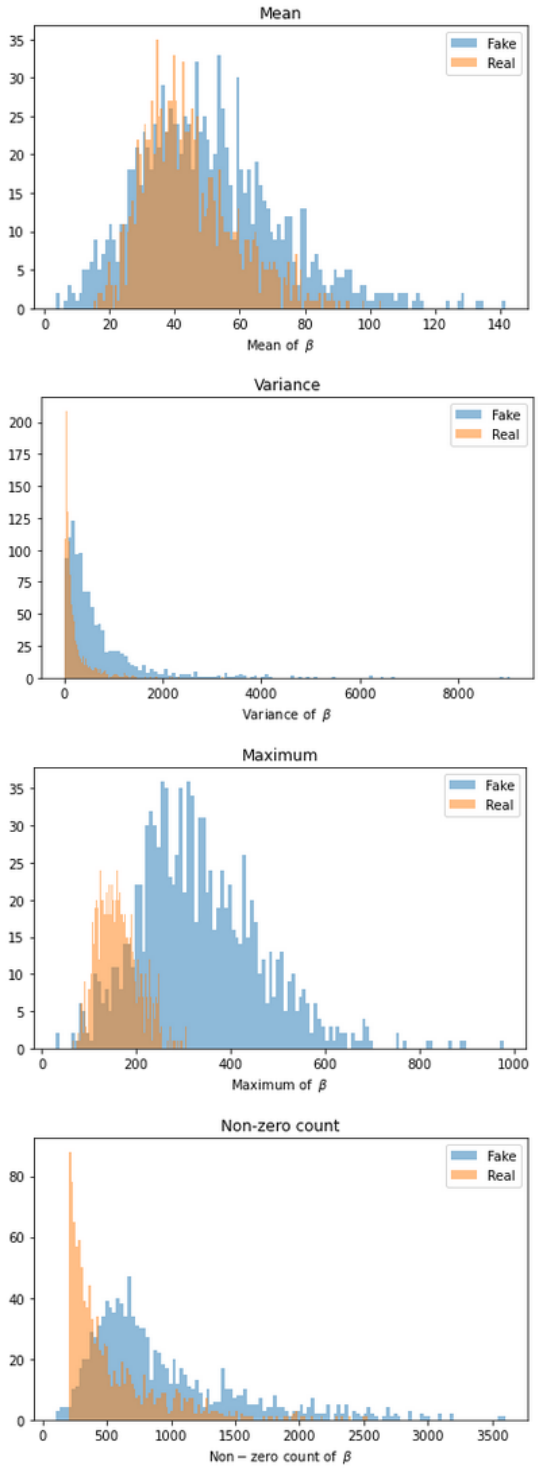


Figure 21: Beta statistics comparing 1024 real and fake clouds for WGAN with NZE loss

As we can see adding the NZP loss significantly improved the statistics of the generated data. Both the graphs for the non-zero count and the mean of the betas align much more closely now,

but we can see that the graphs for the maximums are still quite different, the range for the maximums of the fake data is much larger.

5.3 Learning time

We trained all the networks on a GeForce GTX 1070 GPU.
The time it took for the different GANs to converge was:
DCGAN 3-4 hours
WGAN 6-10 hours
The reason WGANs take longer to converge could be attributed to many things, every batch, WGAN trains the discriminator n time (5 in our case) compared to the one time the DCGAN does it. WGAN uses image blending which means costly math on matrices. WGAN also uses the random function a lot more which could increase the time quite a bit.

6. Summary & conclusions

In this project we learned about the physical properties and physical models of clouds – how to best prepare a dataset of such models to make sure it's physically consistent by taking into account the height of the voxels when slicing the clouds. We took some inspiration from previous works about texture synthesis, such as adding an extra loss element to the original generator or discriminator losses to improve certain elements of the produced samples. We found that without an extra loss element, the WGANs we trained focused more on producing images that were photorealistic and as a result, they produced images that were not physically consistent with the training images.
Synthesizing clouds using GANs has potential, if it succeeds then it could be a massive improvement over current methods (namely physical simulators), as the current methods take a long time to produce samples, while GANs can do so in merely seconds (in our experiment it takes only fractions of a second to produce one sample).
We started with a DCGAN that learned to generate faces of people, we adjusted its architecture to accommodate the clouds images and tuned its hyperparameters and its loss functions (adding MSE loss) to improve the results. We then moved to WGANs and WGAN-GPs and added loss on the number of non-zero elements.
DCGANs seemed to produce samples that are closer statistically (in regard to the extinction coefficient) to the ground truth without much tuning needed, when compared to WGANs.
WGAN learns slower and needs more supervision to produce results that are more physically consistent but has greater stability while learning and has the advantage of more photorealistic results.

7. Future Works

Recently, new articles claim that GANs such as DCGAN and WGAN can produce photorealistic images, but struggle when it comes to physical consistency. They propose a new type of GAN called PIGAN (physics informed GAN), these GANs incorporate physics into the loss of either the generator or the discriminator to try and make the GAN learn the physical attributes of the training data[11], we suggest looking further into expanding PIGANs into cloud synthesis.
To get more varied results and to improve the training we also suggest using training data from different physical simulations (different cloud fields), this can enrich the training data and further improve the results.

Acknowledgements

We thank Johanan Erez, Eli Appelboim, Ina Talmon & Daniel Yagodin from The Vision and Image Sciences Laboratory (VISL) as well as Adi Vainiger, Yael Sde Chen, Omer Shubi & Ido Czerninski from the Hybrid Imaging Lab at The Andrew & Erna Viterbi Faculty of Electrical and Computer Engineering for helpful comments and advice as well as help with technical issues.

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 810370: CloudCT).

References

[1] Levis, A., Schechner, Y.Y. and Davis, A.B. Multiple-scattering microphysics tomography. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 6740-6749), 2017.

[2] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Advances in Neural Information Processing Systems 27, pages 2672–2680. Curran Associates, Inc., 2014.

[3] Loeub, T., Levis, A., Holodovsky, V. and Schechner, Y.Y. Monotonicity prior for cloud tomography. In Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVIII 16 (pp. 283-299). Springer International Publishing, 2020.

[4] L. A. Gatys, A. S. Ecker, and M. Bethge. Texture synthesis using convolutional neural networks. In Advances in Neural Information Processing Systems (NIPS), pages 262–270, 2015.

[5] N. Jetchev, U. Bergmann, and R. Vollgraf, ‘‘Texture synthesis with spatial generative adversarial networks,’’ in Proc. Adv. Neural Inf. Process. Syst., 2017, pp. 1–11.

[6] Yang Zhou, Zhen Zhu, Xiang Bai, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. Non-stationary texture synthesis by adversarial expansion. ACM Transactions on Graphics (TOG), 37(4):49:1–49:13, 2018.

[7] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein generative adversarial networks,” in International conference on machine learning. PMLR, 2017, pp. 214–223.

[8] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, “Improved training of wasserstein gans,” arXiv preprint arXiv:1704.00028, 2017.

[9] Aviad Levis, PYSHDOM. https://github.com/aviadlevis/pyshdom

[10] Nathan Inkawhich, DCGAN Tutorial. https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

[11] L. Yang, D. Zhang, and G. E. Karniadakis, “Physics-informed generative adversarial networks for stochastic differential equations,” arXiv preprint arXiv:1811.02033, 2018