# Generating realistic fantasy planets with Generative Adversarial Networks

Christian Burmester [a], Maximilan Kalcher [b]
University of Osnabrück, Osnabrück, Germany
[a]jburmester@uos.de
[b]mkalcher@uos.de

31.08.2022

**Abstract**

We introduce a Generative Adversarial Network (GAN) that can produce images of planet-like objects. Our model is inspired by recent advances in GAN training, and Arjovsky, Chintala, and Bottou (2017)'s paper on Wasserstein GANs (WGAN). We trained our model with a stretched dataset that is based on images of celestial bodies of our solar system. Those include 13 planets and dwarf planets and 18 moons. Using a loss function comprising of the WGAN loss and the Structural Similarity Index (SSIM), our model managed to build on top of previous works of GAN training in the context of astronomy. It is able to generate realistic celestial objects with proper textures, sources of light and coloring. Our work results can be accessed via a web-app integration.

**Keywords:** GAN; WGAN; AI; machine learning; deep learning; SSIM; planets; astronomy; universe

**GitHub** https://github.com/avocardio/ganiverse

**Web-App** https://theseplanetsdonotexist.com

**Parent website** https://space.ml

## 1 Introduction

**Context and goal of this project** This work was produced as part of a study seminar at the University of Osnabrück. The seminar, called "Projects in Deep Computer Vision", was supervised by Axel Schaffland and Ulf Krumnack.

Inspired by recent successes of GANs in Computer Vision, paired with our fascination for these types of Deep Learning models and our universe, we deployed an improved GAN to investigate the potential of generating never seen, realistic celestial bodies. We detected that there is a sparsity of modern approaches and implementations of such networks in the field of astronomy. Despite some previous work, projects with appealing and interactive applications are missing. In a web article, Nasa (2022) confirmed the existence of 5,000 exoplanets. These are planets that lie outside of our solar system and their existence is documented in various lists and databases. However, there does not exist any appealing visualizations of so far detected exoplanets. The goal of our project was to find out whether or not GANs were able to produce images of fantasy exoplanets, based on the celestial bodies of our solar system that we have real and good images of. Finally, we hope to further bridge the intersection between Deep Learning and astronomy.

This work makes use of the following methods:

1. We use WGAN parts for stable training, and
2. the Structural Similarity Index (SSIM) for an additional loss component.

**Generative Adversarial Networks** A GAN is a generative Deep Learning model that was first introduced by Goodfellow et al. (2014). Specifically, it is a type of network that can be utilized to produce an output that is similar to an existing dataset of, for example, images. A GAN comprises of a generating sub-model, the generator $G$, and a discriminating sub-model, the discriminator $D$. In every epoch, $D$ is presented with either a real sample stemming from the original data distribution or with a fake sample stemming from a fake data distribution of the generator. During training, the generator seeks to learn the original distribution by receiving feedback for its suggestions in every epoch. Since there is no intervention by the developers except their influence on the original dataset, it is an unsupervised learning task. Research often refers to the learning process as zero-sum game as both sub-models try to converge. $D$ attempts to decrease the probability of it rating fake images as being real. $G$ tries to increase this probability. Feedback is given in the form of gradient optimization. The nash equilibrium is reached when $D$ cannot decrease the probability of a fake image being rated as real and $G$ can no longer increase this probability.

# 2 Related work

## 2.1 GANs for image generation

Since the beginning of generative network history, GANs have been implemented with many different objectives, architectures, and training schemes. One of the most common implementations of GANs is the DCGAN from Radford, Metz, and Chintala (2015), which was the first attempt to train a GAN using a more stable architecture. DCGANs are implemented using a deep convolutional architecture, meaning the entire network, generator and discriminator, are deep convolutional neural networks. DCGANs are trained using a method called batch normalization, where the activations at each layer in the generator or discriminator are normalized to have zero mean and unit variance. This normalization method helps improve training stability. The novel architecture also included the ReLu activation in the generator, except for the output which uses TanH and LeakyReLu in the discriminator layers. This bounded activation helped the model to saturate the color space more efficiently. The newly optimized model for image generation proved to work very well in generating realistic images.

## 2.2 State-of-the-art: GANs in astronomy

In the field of astronomy, researchers have long been looking for a way to generate realistic images of celestial bodies in order to study their formation and evolution. However, this is a very difficult task because of the vast amount of data that needs to be collected and processed.

Generative Networks were notably first used specifically in the context of astronomy in the work of Schawinski, Zhang, Zhang, Fowler, and Santhanam (2017), who used GANs to recover astrophysical objects such as galaxies from random or systematic noise[1] due to conventional deconvolution techniques being limited in their ability to recover features. They trained on 4,500 images of nearby galaxies and used as inputs a set of image pairs: one high definition image of a galaxy and an artificially degraded image. The GAN then learned to recover the noisy image. The outputs of the layers were normalized by clipping, a method usually used in the context of WGAN training, to avoid exploding gradients. The training time was around 200 hours on an NVIDIA Titan X GPU. Overall, they achieved very good results and the GAN-created images look very similar to the original. The model was not, however, used for generating new images.

Another more recent paper from Coccomini, Messina, Gennaro, and Falchi (2021) was successful in generating new images of celestial objects. The paper used a lightweight Style-GAN-like architecture, partially due to data limitations, but also to train for a shorter period of time. The generated images were either 128x128 or 256x256 pixels in size, and included all kinds of celestial objects. The training was done on a single NVIDIA Tesla T4 GPU and took 3 days. Notably, besides using the Frechet Inception Distance (FID), used widely to assess the quality of the GAN output, they also used the Structural Similarity Index (SSIM) to compare real and generated images. Their model achieved promising results, but due to the small output size, the generated images were not as sharp and less realistic looking as one could hope. As a result, these pictures may be difficult to be used for scientific purposes.

# 3 Dataset

The images we trained our network with were scraped from different sources. With a great collection of images of celestial bodies of our solar system available on their website, NASA formed the main source. All files and their respective sources can be found in our Github repository. We selected a set of 31 celestial objects that consisted of 13 planets and 18 moons[2]. We scraped these images manually from the websites and scaled them for further preprocessing.

[1] This includes background light, instrument noise, telescope accuracy, etc.

[2] We purposely selected those moons that gave the base dataset a good variety in terms of colours, surface structures, craters, etc.

Planets and dwarf planets:

1. Ceres
2. Earth
3. Eris
4. Haumea
5. Jupiter
6. Makemake
7. Mars
8. Mercury
9. Neptune
10. Pluto
11. Saturn
12. Uranus
13. Venus

Moons:

1. Callisto (moon of Jupiter)
2. Charon (moon of Pluto)
3. Enceladus (moon of Saturn)
4. Europa (moon of Jupiter)
5. Ganymede (moon of Jupiter)
6. Iapetus (moon of Saturn)
7. Io (moon of Jupiter)
8. Mimas (moon of Saturn)
9. Miranda (moon of Uranus)
10. Moon (moon of Earth)
11. Oberon (moon of Uranus)
12. Phobos (moon of Mars)
13. Rhea (moon of Saturn)
14. Tethys I (moon of Saturn)
15. Titan (moon of Saturn)
16. Titania (moon of Uranus)
17. Triton (moon of Neptune)
18. Umbriel (moon of Uranus)

# 4 Methodology

## 4.1 Data preprocessing

In a recent study by Kamenshchikov and Krauledat (2018) on optimised GAN training it has been shown that around 50,000 data points is an ideal dataset size for a classic GAN. When the GAN is given a very small dataset, there is a risk of overfitting the discriminator during training phase according to Karras et al. (2020). The discriminator has no difficulty in telling the difference between real and fake, which results in sparse feedback for the generator and a collapse and diverge of training. We started with an initial dataset of 160 images, consisting of 5-6 images per object. We thus had to rely on data augmentation to stretch our dataset and overcome this risk. All of our data augmentation techniques had invertible transformations, and although we did not match the suggested 50,000 images, already with our final dataset size of 5,000 images, the model had shown decent results. All images are resized to 500x500 pixels, normalized, shuffled, cast to float32 and batched (16).

**Data Augmentation** We utilized conventional data augmentation techniques to add variations of the 160 original images to our dataset. Our goal was to have a more heterogeneous dataset to map onto the diversity of exoplanets outside our solar system as suggested by Ballmer and Noack (2021). Given the almost perfect round shape of planets and moons (apart from Humea), classic geometrical operations, such as rotation, flipping top to bottom, and flipping left to right did not yield much variety. Our data augmentation pipeline therefore included colour conversion, blurring, detail filter, enhancing, and increase of contrast, sharpness and brightness. We allowed the deployment of data augmentation to change the original data distribution slightly but not dramatically. The reason here was to not restrict our view too much to what objects exist within our solar system and, at the same time, ensure that the generator would still be able to learn the original data distribution. Data augmentation scaled the dataset by a factor of 10, resulting in 1,600 images.

**Mix-up** The mix-up of images is a data augmentation technique, that was introduced by Zhang, Cisse, Dauphin, and Lopez-Paz (2017) and, inter alia, used by Liu et al. (2022) for their novel ConvNeXt model. The promise and objective of this technique is to avoid memorization and sensitivity to adversarial examples, improve generalization, and stabilize GAN training, according to the authors. In Mix-up, two randomly chosen data samples are blended to construct a new, synthetic data sample. The proportion of each of the two original images for the new image, is given by a probability. For our data augmentation we chose 0.5 in order to have an equal share in the new image. Using Mix-up, we stretched the dataset to a final of 5,000 (500x500 pixel) images for training.

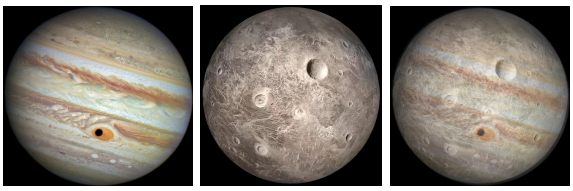

Figure 1: Left: Jupiter; Middle: Ceres (dwarf planet); Right: resulting Mix-up image

## 4.2 Model architecture

Our model architecture is effectively a WGAN from Arjovsky et al. (2017), a simple, yet powerful GAN architecture that uses a gradient penalty, as well as additional modifications that we believe helped to improve training stability. It consists of a generator $G$ and a critic $C$ (instead of the usual discriminator $D$). Both $G$ and $C$ are fully convolutional neural networks, except for $G$'s input layer.

**Generator** The generator consists of a single densely packed input layer of size 65,536, which is reshaped to 16x16x256, followed by 6 up-sampling (deconvolution) layers with batch normalization (BN) and a LeakyReLU activation each, and by 2 down-sampling (convolution) layers to achieve the 500x500 pixel output images. The dense input layer gets a noise vector $z$, sampled from a normal distribution with the dimension $dim_z = 100$. The final convolutional layer uses a TanH activation, a stride of 1 and 3 5x5 filters in order to create a 3-channel output image. The TanH activation ensures the output pixel values range in $[-1; 1]$. $G$ was initialized with the RMSProp optimizer using a learning rate of $1e^{-4}$ with a decay of $1e^{-5}$.

**Critic** The critic gets a generated image of size 500x500 as an input and uses 5 down-sampling (convolution) layers with LeakyReLU activations and dropout layers with a drop rate of 0.1 between them, followed by a single dense layer with a linear activation. This linearity is used to prevent our gradients from vanishing, as described in Arjovsky et al. (2017). $C$ was also initialized with the RMSProp optimizer using a learning rate of $1e^{-4}$ with a decay of $1e^{-5}$.

**Gradient penalty** Gradient penalty is a technique used to train a WGAN to enforce the Lipschitz constraint and to prevent the critic from becoming too powerful or collapsing. It is based on the idea that two images that are close in the latent space should be close in the image space, i.e. their gradients should be close. The objective is to enforce the critic to be locally Lipschitz continuous[3], therefore to have a gradient norm of 1, using a penalty that is added to the loss function. The gradient norm is calculated using images $\hat{x} = \epsilon * x + (1 - \epsilon) * x'$, where $x$ and $x'$ are two real images and $\epsilon$ is the random noise vector, sampled from a uniform distribution. The computed gradient penalty value is then passed on as an addition to the critic loss. Additionally, we manually clip all weights in both the generator and critic to be in between $[-0.5; 0.5]$ for each training iteration.

**SSIM** The Structural Similarity Index is a metric used to compare two images to find out how similar they are. It takes into account both the luminance and the structural similarity, i.e. how similar the two images are in terms of their pixels and their shapes (Wang, Bovik, Sheikh, and Simoncelli (2004)). This is important in our case, since our objective is to generate images that look "natural", i.e. that have a realistic shape. The SSIM outputs a score between 0 and 1, and the higher the score, the more similar the two images are. We import the metric using the *skimage* package, but wrap it with a custom function, where a number of previously generated images are all compared with the current $G$ output, and the mean value taken from all the SSIM scores is returned:

$$ssimloss(\alpha, \beta) = \frac{1}{|\alpha|} \sum_{i=1}^{|\alpha|} SSIM(\gamma, \gamma_i) * \beta \tag{1}$$

Where $\alpha$ is the number of previous images, $\beta$ is an internal scaling factor and $\gamma$ is either the current generator output image, or $n - \alpha$ previously generated saved images. The resulting value from this function, in the range $[0; 1]$, is added to 1 and then used as a scaling factor for the generator loss. During training, this loss suppresses mode collapse and encourages more output variability, due to the structural similarity between images after each training step decreasing.

### Loss functions

1. In order to train our generator, we use the mean of our generated images multiplied by our custom SSIM loss as the objective function:

$$\mathcal{L}_G = E_{z \sim p_z(z)}[G(z)] * (1 + ssimloss(3, 1)) \tag{2}$$

   This means that the SSIM-based loss takes into account the last $\alpha = 3$ generated images with a scaling factor of $\beta = 1$. In a good training scenario, the first part of the loss function is just scaled by 1.

2. For our critic, we use the standard Wasserstein loss, which is the mean absolute difference between the generated and the real image samples. We coupled this with 1.5 times the gradient penalty value:

$$\mathcal{L}_C = E_{z \sim p_z(z)}[C(G(z))] - E_{x \sim p_r(x)}[C(x)] + (1.5 * GP) \tag{3}$$

   Where $GP$ is the gradient penalty, and $p_z(z)$ and $p_r(x)$ are the distributions of the random noise vector and real image samples, respectively.

[3] Also locally linear. This type of continuity is important for ensuring that the WGAN training process converges.

### 4.3 Training

Within the training loop, we first sample a batch (16) of noise from a normal distribution $p_z(z)$ with mean 0 and variance 1, and use $G$ to generate images from the noise. $C$ is called with both real and generated images from the batch. We then calculate the loss for $C$ and $G$. The total loss is backpropagated to update the weights of all parameters, which are then clipped to $[-0.5; 0.5]$. For every training iteration, $C$ is called 3 times, and $G$ is called once.

We trained our model for 500 epochs on an NVIDIA RTX 3090 for about 18 hours using the Python library Keras and Tensorflow as the backend. During training, we saved the ($G$) model weights every 50 epochs. After 50 epochs, the generated images began to show a round structure, and after about 200 epochs, the generated images became visually similar to real images of planets. We monitored the training in real time using a custom integration of the Telegram API in our training loop, which can be found in our repository. After training, we converted the Tensorflow model to TensorflowJS for use in our web application.

## 5 Results

### 5.1 Fantasy planets

In early training (see Fig. 1), the model successfully figures out the shape of the original images.

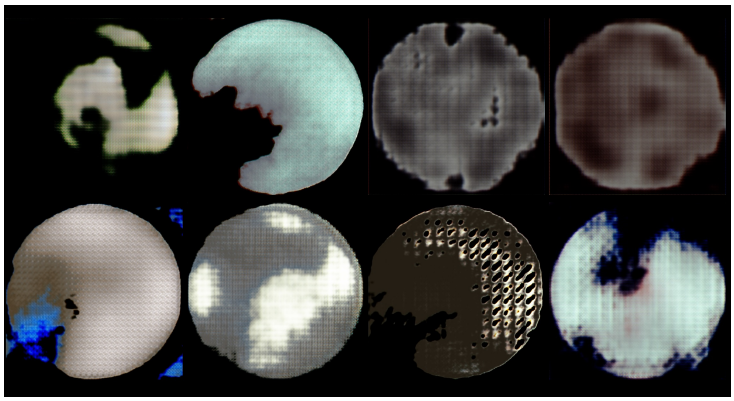

Figure 2: Generator outputs in early training phase

In the later training phase (see Fig. 2), the shape is fixed and the model achieves appealing results. Although of fictive nature, familiar structures such as craters or oceans can be spotted in the images. Each object seems to have a different source of light and colouring.

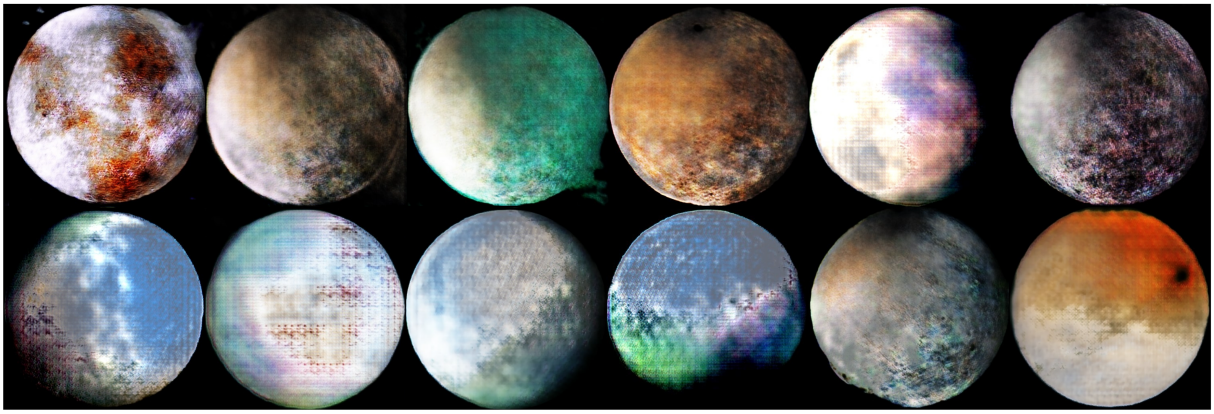

Figure 3: Generator outputs in later training phase

After training, our generator can be sampled from a normal distribution $p_z(z)$ to generate new images. A few examples of generated images from our web interface can be seen in Fig. 3. The mean of this z-value can be specified to fix the sampling process, while the variance is adjusted to 0.2 for image consistency. It can also be left to the model to set a random value for z for a complete randomized process and notably the best results.

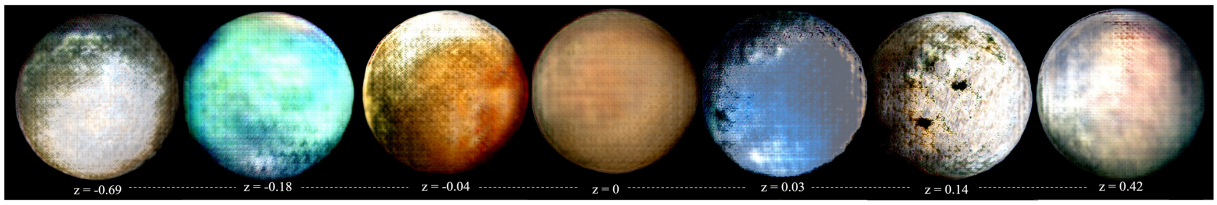


Figure 4: Generator outputs with a fixed z-distribution mean; z-means ascending from left to right (-0.69, -0.18, -0.04, 0, 0.03, 0.14, 0.42)

## 5.2 Surprises and challenges

**Mode collapse** The main issue we had while training the GAN was the mode collapse. After trying a classic DCGAN (without WGAN-parts), we realized that although the generated images after each training step differed a lot, once the model concluded training, the generated images became identical to each other.

This well-known challenge in GAN training was first detected by Metz, Poole, Pfau, and Sohl-Dickstein (2016) and results in the generator producing the same output during training or after training, or in other words, the generator only learns to produce a single or a few modes from the real data distribution. Although the generator may be able to trick the discriminator for a few epochs, it does not internalize a latent space that maps onto the original data distribution. Instead, it maps several different or all z-values of the original data distribution onto the same generated data point or mode of mixture. The loss function of regular GANs attempt to quantify the similarity between the generated and real data distributions, but cannot quantify the diversity of the generated samples.

This issue is usually solved by using a Wasserstein GAN, which has a new loss function that encourages the generator to learn the entire data distribution. This also constrains its discriminator to be Lipschitz-continuous. Gradient Penalty is also used to enforce this constraint and has proven to be highly effective in preventing mode collapse. In our case, we did not only upgrade our model to have WGAN parts, but also implemented a custom loss function based on the SSIM, as explained in the sections above, to completely try to suppress the mode collapse.

**Artifacts** Artifacts are another common GAN problem that can appear as repetitive patterns, like in some of our cases, or as blurring or other visual defects. Odena, Dumoulin, and Olah (2016) explain this phenomenon by the fact that the kernel size of a layer is not dividable by its stride, effectively causing an overlap in pixels in certain parts of the image, or also, when using large strides in the final generator layers. Although the broader implications are still unclear, gradient artifacts due to strided convolutions in the discriminator could also be the problem.

Even though our generator upsampling or deconvolution layers use at most strides of 2, and our final layers all have a strides of 1, we were still not able to avoid some patterns in our generated images. Some solutions proposed in the literature are to either resist the artifacts using a new loss function component, or leaving out deconvolution layers altogether and instead adding nearest-neighbor upsampling layers to the generator, followed by convolutional layers only, similar to some super-resolution models. But this was not something we tried.

We decided to suppress these artifacts by applying a post-processing filter to smooth out the image. The filter is the bilateralFilter[4] from the *OpenCV* package for reducing unwanted noise while keeping the edges sharp. The filter was not applied in the previous section.

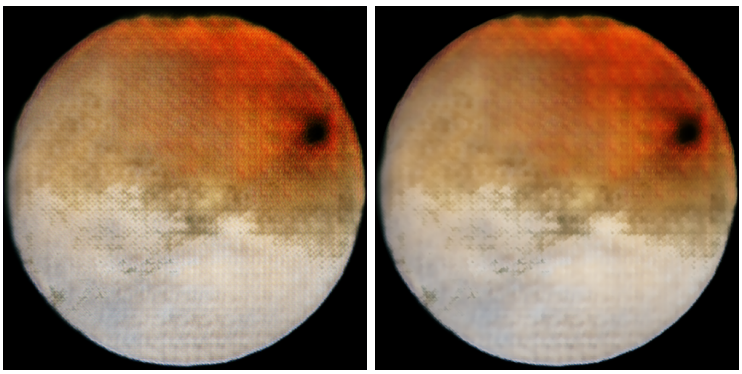

Figure 5: Left: Generator output; Right: Generator output after using bilateralFilter

[4]Using the following parameters: cv2.bilateralFilter(image,8,25,25).

# 6 Conclusion

We conclude that the WGAN architecture we used was able to capture the essence of real planets, i.e. their shape, lighting, color, and to produce images that are visually realistic and appealing. We also conclude that our improved model was able to successfully train without mode collapse. We believe that the SSIM-based loss contributed to this result, although we did not compare this custom loss with other losses. Overall, for our short training time, the results are very promising. But of course, one can try to train the model much longer, and with more data, in order to obtain even better results.

Another interesting approach would be to pass smaller generated images through a subsequent custom trained up-scaling model, therefore not relying on the generator to generate high quality images, but to only focus on output variability. As for the model, we would strongly suggest using a uniform distribution to sample from, rather than a normal. Specially in the context of interactive sampling, this would lead to a wider and more flexible range of fixed distribution output images.

Future work may also consider exploring newer architecture types, i.e. word-to-image diffusion models, as these are becoming increasingly popular in the field of AI. All in all, we believe that the intersection of Deep Learning and astronomy still has a lot to offer, and that there is a great potential for further applications within exoplanet research and beyond.

# References


Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein gan. Retrieved from https://arxiv.org/abs/1701.07875 DOI: 10.48550/ARXIV.1701.07875

Ballmer, M. D., & Noack, L. (2021). *The diversity of exoplanets: From interior dynamics to surface expressions.* arXiv. Retrieved from https://arxiv.org/abs/2108.08385 DOI: 10.48550/ARXIV.2108.08385

Coccomini, D., Messina, N., Gennaro, C., & Falchi, F. (2021). Generative adversarial networks for astronomical images generation. *CoRR*, *abs/2111.11578*. Retrieved from https://arxiv.org/abs/2111.11578

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... Bengio, Y. (2014). Generative adversarial networks. Retrieved from https://arxiv.org/abs/1406.2661 DOI: 10.48550/ARXIV.1406.2661

Kamenshchikov, I., & Krauledat, M. (2018, November). Effects of Dataset properties on the training of GANs. *arXiv e-prints*, arXiv:1811.02850.

Karras, T., Aittala, M., Hellsten, J., Laine, S., Lehtinen, J., & Aila, T. (2020). *Training generative adversarial networks with limited data.* arXiv. Retrieved from https://arxiv.org/abs/2006.06676 DOI: 10.48550/ARXIV.2006.06676

Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., & Xie, S. (2022). *A convnet for the 2020s.* arXiv. Retrieved from https://arxiv.org/abs/2201.03545 DOI: 10.48550/ARXIV.2201.03545

Metz, L., Poole, B., Pfau, D., & Sohl-Dickstein, J. (2016). *Unrolled generative adversarial networks.* arXiv. Retrieved from https://arxiv.org/abs/1611.02163 DOI: 10.48550/ARXIV.1611.02163

Nasa. (2022). *The count of confirmed exoplanets just ticked past the 5,000 mark, representing a 30-year journey of discovery led by nasa space telescopes.* (https://exoplanets.nasa.gov/news/1702/cosmic-milestone-nasa-confirms-5000-exoplanets/,)

Odena, A., Dumoulin, V., & Olah, C. (2016). Deconvolution and checkerboard artifacts. *Distill*. Retrieved from http://distill.pub/2016/deconv-checkerboard DOI: 10.23915/distill.00003

Radford, A., Metz, L., & Chintala, S. (2015). *Unsupervised representation learning with deep convolutional generative adversarial networks.* arXiv. Retrieved from https://arxiv.org/abs/1511.06434 DOI: 10.48550/ARXIV.1511.06434

Schawinski, K., Zhang, C., Zhang, H., Fowler, L., & Santhanam, G. K. (2017, 01). Generative adversarial networks recover features in astrophysical images of galaxies beyond the deconvolution limit. *Monthly Notices of the Royal Astronomical Society: Letters*, *467*(1), L110-L114. Retrieved from https://doi.org/10.1093/mnrasl/slx008 DOI: 10.1093/mnrasl/slx008

Wang, Z., Bovik, A., Sheikh, H., & Simoncelli, E. (2004). Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, *13*(4), 600-612. DOI: 10.1109/TIP.2003.819861

Zhang, H., Cisse, M., Dauphin, Y. N., & Lopez-Paz, D. (2017). *mixup: Beyond empirical risk minimization.* arXiv. Retrieved from https://arxiv.org/abs/1710.09412 DOI: 10.48550/ARXIV.1710.09412