Cognitive Science Student Journal
Osnabrück University

Generating Fantasy Planets with Generative Adversarial Networks

Christian Burmester,
Maximilian Kalcher

Abstract

We introduce a Generative Adversarial Network (GAN) that can produce images of planet-like objects. We trained our model with a stretched dataset that is based on images of celestial bodies of the solar system. Those include 13 planets and dwarf planets and 18 moons. Using a loss function comprising of the Wasserstein-GAN (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-looking celestial objects with proper textures, sources of light and rich coloring. Our work results can be accessed via a web app integration.

1 Introduction

Inspired by recent successes of Generative Adversarial Networks (GANs) and our fascination for the universe, we investigated the potential of GANs for generating imaginative but realistic-looking celestial bodies. In a web article, Brennan (2022) confirmed the existence of 5,000 exoplanets. These are planets that lie outside of our solar system. Their existence is documented in various lists and databases. However, there do not exist any appealing visualizations of so far detected exoplanets. The goal of our project was to find out whether GANs are able to produce the visuals of imaginative exoplanets based on real images of the celestial bodies in the solar system. It is to be noted that this work does not take into account any other features of the celestial bodies of the solar system (e.g., temperature, density, etc.). It solely considers images to envision exoplanets that may potentially be in our universe. Finally, we hope to bridge the intersection between Deep Learning and astronomy. It is to be noted that the project was finalized in July 2022.

This work makes use of the following methods:

1. The Mix-up preprocessing function to generate a larger dataset,
2. WGAN parts for stable training, and
3. The Structural Similarity Index (SSIM) for an additional loss component.

Generative Adversarial Networks A GAN is a generative Deep Learning model that was 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, G 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. The learning process can be seen as a zero-sum game as both sub-models try to converge (Farnia & Ozdaglar, 2020). 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.

GANs in astronomy Generative networks were first notably used in the context of astronomy in the work of Schawinski et al. (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. The GAN-created images looked very similar to the original; however, the model was not deployable for generating new images.

Another more recent paper by Coccomini et al. (2021) was successful in generating new images of celestial objects. The paper used a lightweight GAN, 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 three days. Notably, besides using the Frechet Inception Distance (FID), widely used 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 and quality of training images, the generated images were not as sharp and less realistic-looking than one could hope.

2 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[2]. We selected a set of 31 celestial objects that consisted of 13 planets and 18 moons[3]. We scraped these images manually from the websites and scaled them for further preprocessing.

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).

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

[2]Please refer to https://github.com/avocardio/ganiverse.

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

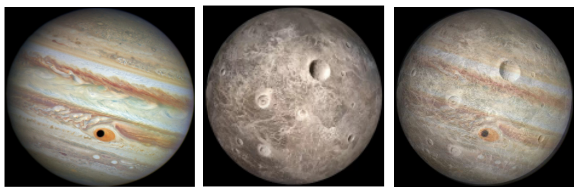

Figure 1: Left: Jupiter; Middle: Ceres (dwarf planet); Right: resulting Mix-up image

3 Methodology

The following section discusses the methodology used in our work. We first outline the data preprocessing steps used. Then we describe the model architecture and the training process.

3.1 Data preprocessing

In a recent study by Kamenshchikov and Krauledat (2018) on optimized GAN training, it has been shown that around 50,000 data points is an ideal dataset size for a classic GAN. According to Karras et al. (2020), about "$10^5 - 10^6$ images [are] required to train a modern high-quality, high-resolution GAN". Arjovsky and Bottou (2017) and Y. Wang et al. (2018) reported that there is a risk of overfitting the discriminator during training when datasets are too small. 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. In our work, we started with an initial dataset of 160 images, consisting of 5-6 images per object. We had to rely on data augmentation to stretch our dataset and overcome the risk mentioned above. All of our data augmentation techniques had invertible transformations, and although we did not match the suggested 50,000 or 100,000 images, training with a dataset size of 5,000 images already showed acceptable results. All images are resized to either 500x500 or 256x256 pixels, normalized, shuffled, cast to float32 and batched (batch size of 16).

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 perfectly 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 color 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 for this 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.

The Mix-up of images is a data augmentation technique that was introduced by Zhang et al. (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 of 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. Figure 1 shows a Mix-up example. For our data augmentation, we chose 0.5 in order to have an equal share in the new image. We ended up with a final dataset of 5,000 images for training.

3.2 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 one and three 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 WGAN literature. 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[4], i.e., 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 ϵ 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 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 (Z. Wang et al., 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, created by van der Walt et al. (2014), 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 α is the number of previous images, β is an internal scaling factor and γ is either the current generator output image, or $n - \alpha$ previously generated saved images. The resulting value

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

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:

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

   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:

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

   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.

3.3 Training

Within the training loop, we first sample a batch (batch size of 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 three times and G is called once.

We trained our first model on 500x500 pixel images for 500 epochs on an NVIDIA RTX 3090 for about 18 hours using the Python library Keras and Tensorflow as the backend. For our second model, we used 256x256 pixel images on the same hardware and configuration. 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.

4 Results

After training both models, we analyzed the generated imaginary planets. In this section, we discuss the observed results and examine our model outputs for unwanted artifacts.

4.1 First model

Our first model had a size of about 4 million parameters. As can be seen in Figure 2, the shape of the objects was round and started showing realistic-looking results during the later training phase. Although of fictive nature, familiar structures such as craters or oceans can be spotted in the images. Each object seemed to have a different source of light and coloring.
After training, the generator of the model can be utilized to generate new images by sampling from a normal distribution $p_z(z)$. A few examples of generated images from our web interface can be seen in Figure 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 completely randomized process and, notably, the best results.

4.2 Second model

Our second and smaller model has about 1.8 million parameters. This model still managed to contain high output variance and surprisingly less visible artifacts than the first model. We did notice, however, that when trying to direct the model by using the normal distribution $p_z(z)$ with a tight variance as described above, the outputs across any mean distribution [0,1] were mostly the same, with only slight changes in grey and brown coloring. The best results, again, can be seen using a completely randomized distribution of z. Figure 4 shows examples of the generator output after training.

4.3 Artifacts

Next to overfitting, 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 et al. (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. This can also occur 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 two, and our final layers all have strides of one, we were still not able to avoid some patterns in our generated images.

5 Conclusion

We conclude that the WGAN architecture we used was able to capture the essence in images of the real objects, i.e., their shape, lighting and color. Both our models were able to produce photo-realistic images that are visually appealing and suitable for artistic or other experimental use cases. For example, they may support communicating and visualizing the research of exoplanets. We noticed that our improved model was able to train successfully 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 and limited data, the results were very promising. Our smaller (1.8m parameter) model showed encouraging results despite its size, leading to extremely fast inference times for the mobile version of the web app.

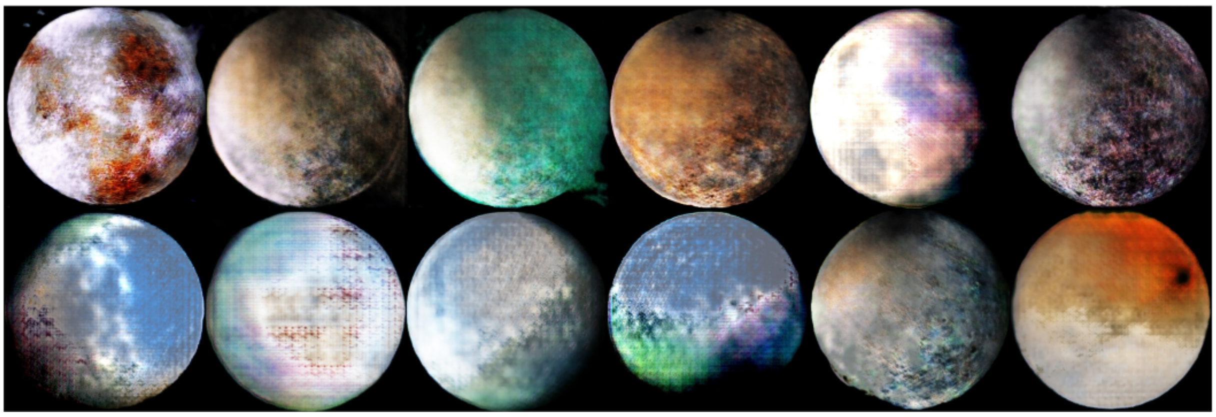

Figure 2: 4m generator model outputs in later training phase.

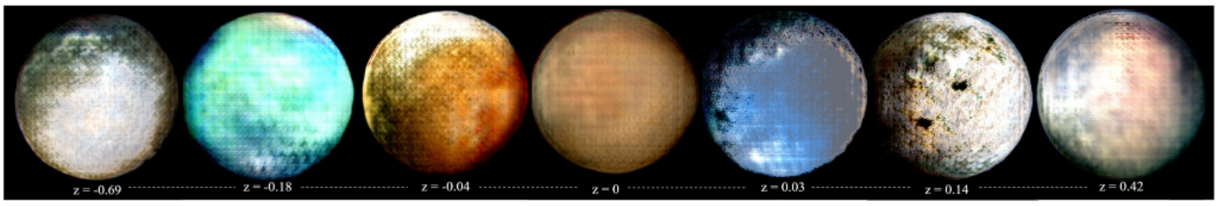

Figure 3: Generator outputs with a fixed z-distribution mean; z-means ascending from left to right (-0.69, -0.18, -0.04, 0, 0.03, 0.14, 0.42)

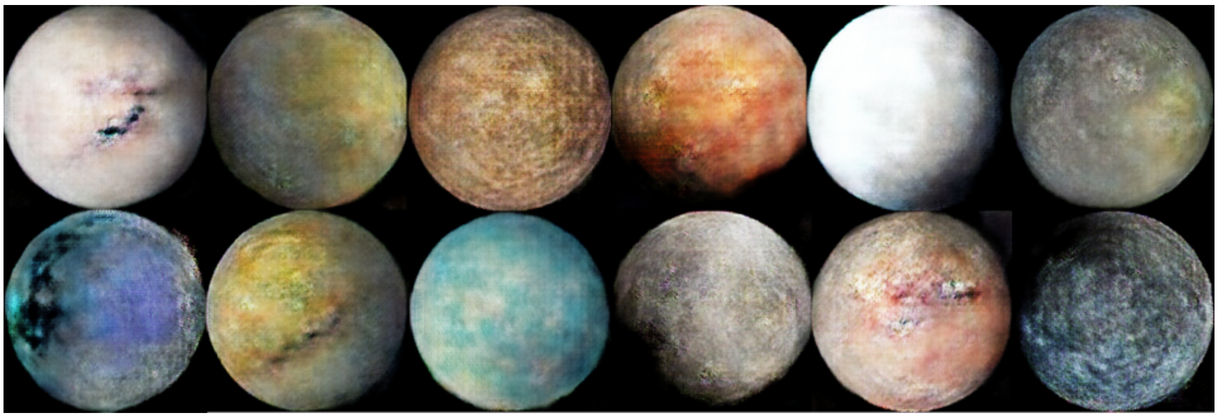

Figure 4: 1.8m generator model outputs in later training phase.

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 one. Especially in the context of interactive sampling, this would lead to a wider and more flexible range of fixed distribution output images.

By now, other techniques, like diffusion models by Ho et al. (2020) and Rombach et al. (2021), have shown much better results in image synthesis (Dhariwal & Nichol, 2021). With this work, we intended to show that GANs still have their application for specific use cases. The almost instant inference time provides the ability to implement such models in apps or website environments. We believe that the intersection of Deep Learning and astronomy still has a lot to offer and that there is great potential for further applications within exoplanet research and beyond.

References

Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein GAN. *arXiv*. https://doi.org/10.48550/arXiv.1701.07875

Arjovsky, M., & Bottou, L. (2017). Towards principled methods for training generative adversarial networks. *ArXiv*. https://doi.org/10.48550/arXiv.1701.04862

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

Brennan, P. (2022, March 21). *Cosmic milestone: Nasa confirms 5,000 exoplanets*. NASA. https://exoplanets.nasa.gov/news/1702/cosmic-milestone-nasa-confirms-5000-exoplanets/

Coccomini, D., Messina, N., Gennaro, C., & Falchi, F. (2021). Generative adversarial networks for astronomical images generation. *arXiv*. https://doi.org/10.48550/arXiv.2111.11578

Dhariwal, P., & Nichol, A. (2021). Diffusion models beat GANs on image synthesis. *ArXiv*. https://doi.org/10.48550/arXiv.2105.05233

Farnia, F., & Ozdaglar, A. E. (2020). GANs may have no nash equilibria. *ArXiv*. https://doi.org/10.48550/arXiv.2002.09124

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial networks. *arXiv*. https://doi.org/10.48550/arXiv.1406.2661

Ho, J., Jain, A., & Abbeel, P. (2020). Denoising diffusion probabilistic models. *arXiv*. https://doi.org/10.48550/arXiv.2006.11239

Kamenshchikov, I., & Krauledat, M. (2018). Effects of dataset properties on the training of GANs. *arXiv*. https://doi.org/10.48550/arXiv.1811.02850

Karras, T., Aittala, M., Hellsten, J., Laine, S., Lehtinen, J., & Aila, T. (2020). Training generative adversarial networks with limited data. *arXiv*. https://doi.org/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*. https://doi.org/10.48550/arxiv.2201.03545

Odena, A., Dumoulin, V., & Olah, C. (2016). Deconvolution and checkerboard artifacts. *Distill*. https://doi.org/10.23915/distill.00003

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., & Björn, O. (2021). High-resolution image synthesis with latent diffusion models. *arXiv*. https://doi.org/10.48550/arXiv.2112.10752

Schawinski, K., Zhang, C., Zhang, H., Fowler, L., & Santhanam, G. K. (2017). 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. https://doi.org/10.1093/mnrasl/slx008

van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., Yu, T., & the scikit-image contributors. (2014). Scikit-image: Image processing in Python. *PeerJ*, *2*, e453. https://doi.org/10.7717/peerj.453

Wang, Y., Wu, C., Herranz, L., van de Weijer, J., Gonzalez-Garcia, A., & Raducanu, B. (2018). Transferring GANs: Generating images from limited data. *ArXiv*. https://doi.org/10.48550/arXiv.1805.01677

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. https://doi.org/10.1109/TIP.2003.819861

Zhang, H., Cisse, M., Dauphin, Y. N., & Lopez-Paz, D. (2017). Mixup: Beyond empirical risk minimization. *arXiv*. https://doi.org/10.48550/arXiv.1710.09412

About the Journal

The 'Cognitive Science Student Journal' aims at giving its readers an insight into current research and cutting-edge topics at our institute from a student perspective as well as students a platform to publish their work. Its editorial board consists of seminar participants and instructors of the Institute of Cognitive Science.

Cognitive Science is taught as an interdisciplinary research field at University Osnabrück, investigating cognition and the mind as a joint research effort of Artificial Intelligence, Neuroscience, Computational Linguistics, Psychology, Neuroinformatics, and Philosophy of Mind.

The journal can be accessed via:
http://cogsci-journal.uni-osnabrueck.de

Find us on social media:
https://www.instagram.com/cogscistudentjournal/
https://www.linkedin.com/company/cognitive-science-student-journal/

Cognitive Science Student Journal 2023, Number 8