Generative Modeling of high-resolution Cosmology Images

Till Schnabel, Sven Kellenberger, Hannes Pfammatter, Michelle Woon
Group: Galaxy Crusaders, Department of Computer Science, ETH Zurich, Switzerland

***Abstract*—This paper explores various machine learning models for generating sparse, high-resolution cosmology images. These models are described and the motivation for their usage is explained.**

The contribution of this paper is twofold. Beside the design of models for cosmology image generation, a similarity function which is part of our data is learned.

The generative models are compared to a simple baseline model and evaluated by estimating the values of the similarity function. We found that extracting the most important features from the images and designing generative models for these features only produced the best results. Our approximation of the similarity function achieved the 2^{nd} best result on both public and private Kaggle test data sets.

I. Introduction

Generative modeling is not a new area of research. However, with the recent advent of deep generative models the research interest in this area has increased drastically.

In this project, we study the generation of cosmology images. As our cosmology images are sparse, high-dimensional data, this has shown to be an interesting problem suitable for the application of mathematical methods and machine learning models.

We split our project into two parts. First, we designed multiple machine learning models for cosmology image generation. Our starting point was a deep convolutional generative adversarial network (DCGAN) capable of generating cosmology images with high resolution. As our model was not able to capture important features such as stars with enough detail, we decided to focus on generating these features only. For the generation of stars we designed a variational autoencoder (VAE), a DCGAN and used an existing PixelCNN. Our final solution consists of a conditional DCGAN (cDCGAN) that is able to generate several classes of stars that were found by clustering.

Second, we designed a CNN and used an off-the-shelf random forest regression model to learn the similarity function which is part of our data. Finally, we evaluated our generative models by estimating the similarity scores of the generated images.

As our models are quite diverse, we refer to section II for the discussion of related work.

II. Models and Methods

This section briefly describes the models and methods which were used for this project and explains the motivation for their usage.

A. Approximation of the similarity function

1) Random forest regression baseline: Random forests were introduced in 1995 by Ho [1] and further developed in 1999 by Breiman [2]. In this work, they were used as an off-the-shelf baseline model for regression. By improving on this baseline we achieved our best result for approximating the similarity function. For training, histograms of image properties were used as input features. We started with a histogram of pixel intensities and consider this our baseline model. By using histograms of different image properties, we were able to improve the accuracy of the regression as measured by the mean absolute error (MAE) by a factor of approximately 3. The results are shown in Table V and detailed in the following paragraphs.

In 2007 Bosch et al. [3] used random forests for image classification. They captured information on shape by computing histograms of oriented gradients (HOG) of edges inside image regions. We computed a similar histogram for the entire image and combined it with the histogram of pixel intensities which improved the accuracy of the regression.

What worked best was a histogram of the power spectrum of the images. To capture the power spectrum in a histogram, we used the fast Fourier transform (FFT) and applied a log transformation. We added histograms of regions of interest (ROI) where we filtered out low frequencies, high frequencies or certain orientations, but were not able to substantially improve the results.

2) CNN: We used a convolutional neural network (CNN) with a deep architecture as our second model for approximating the similarity function. In this subsection, the architecture of the CNN is introduced and analyzed. To speed up computations, most experiments were performed on images with dimensions 125×125 and 250×250. Our most important findings were:

- Batch normalization (Ioffe and Szegedy [4]) stabilized and accelerated training.
- A scaled sigmoid activation function at the output of the network deteriorated accuracy.
- Raising the number of convolutions increased the receptive field and thus improved accuracy.
- Due to the sparsity of the images, fewer parameters prevented the network from overfitting.
- We did not find residual connections (He et al. [5]) to be beneficial.
- Dropout (Srivastava et al. [6]) prevented the network

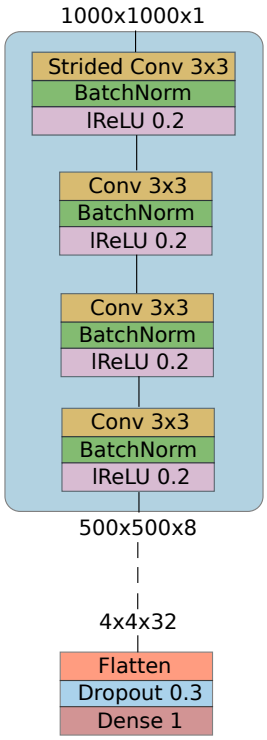

Figure 1. Illustration of the CNN architecture.

from early overfitting.

- Using the log-transformed power spectrum of the images as input to the network improved accuracy

Based on these findings, our final architecture is shown in figure 1. It consists of 8 stacked convolutional blocks where only the first one is shown in figure 1. The number of feature maps is increased by a factor of 2 in each stacked convolutional block until a maximum amount of 32 feature maps is reached. The padding layer used for mapping the spatial resolution from 125×125 to 128×128 is omitted. The network contains a total amount of 228 680 trainable parameters. For training, we augmented the data using horizontal and vertical flips with probability 0.5 as well as random shifts in height and width by up to $20\,\%$. The model was trained for 140 epochs. We achieved mean absolute errors (MAE) of 0.168 and 0.188 on the public and on the private Kaggle test data set.

B. Cosmology Image Generation

1) Adhoc Generator (AG) baseline: The cosmology images are, in essence, white stars on a black background, as such we were able to use a simple tiling method for our easy baseline. Using the available labeled data, we detect stars in all the cosmology images by finding contours in the image. Doing this also allows us to find the minimum and maximum amount of stars in a cosmology image.

Each image to generate first starts out as a black destination image. Then, for each image to generate, a random number between the minimum and maximum amount of stars is selected. For each star to place into the image, a random source cosmology image is chosen and then a random star from that source image is taken. That star is then placed on a random spot in the destination image.

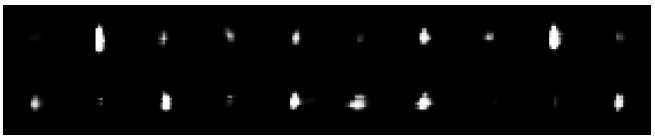

Figure 2. Random selection of stars generated by the large DCGAN.

2) Large DCGAN: The deep convolutional generative adversarial network (DCGAN) was introduced in 2015 by Radford et al. [7]. It is a generative adversarial network (GAN) where both generator and discriminator are convolutional neural networks (CNNs) with architectural constraints which stabilize training. We used a large DCGAN to generate entire cosmology images.

Our implementation is based on on a reference implementation on the TensorFlow website [8] as well as on the original paper and implementation. As proposed by Odena et al. [9], nearest neighbor interpolation and convolution were used instead of transposed convolution.

For training, all labeled images and all scored images with similarity score greater or equal to 2.61 were used. The pixel intensities were normalized to the range $[-1\ ,\ 1]$ and the images were padded to dimensions 1024×1024 to simplify upsampling and strided convolutions.

As shown in Figure 2, the large DCGAN did not manage to capture the shape of important features in detail. Table III shows the estimated similarity scores for these cosmology images.

3) VAE on stars: In our second approach, we focused on the generation of the most important features only. We used a variational autoencoder (VAE) to generate images of stars and placed them within a black background image.

The VAE was introduced in 2014 by Kingma and Welling [10]. It allows the encoding of data points to low-dimensional latent representations $\sim \mathcal{N}(0, I)$ and to generate new data points by decoding arbitrary latent vectors $\sim \mathcal{N}(0, I)$. This is achieved by learning to map each data point x to a Gaussian distribution $q_\phi(z|x)$ determined by μ and σ and each latent vector z to a Bernoulli distribution $p_\theta(x|z)$ determined by p, which is done by maximizing the evidence lower bound (ELBO):

$$-D_{KL}(q_\phi(z|x)||\mathcal{N}(0, I)) + \mathbb{E}_{q_\phi(z|x)}(\log(p_\theta(x|z)))$$

Our implementation approximates the expectation with the binary cross-entropy loss function. For the Kullback-Leibler divergence, the analytic expression is computed.

Our implementation is based on the original paper as well as on the tutorial on variational autoencoders by Doersch

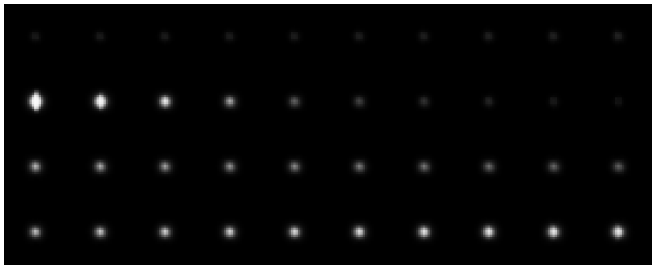

Figure 3. Linear interpolation between vectors in the latent space of the star variational autoencoder.

[11] and on a reference implementation on the TensorFlow website [12].

For training, only the labeled images are used. Using the approach described in the subsection on the adhoc generator, stars are extracted and centered inside images of size 28x28. The pixel intensities are normalized to the range $[0\ ,\ 1]$.

Because stars are shaped in a similar fashion, a multilayer perceptron (MLP) with a single hidden layer of size 500 is used for both the probabilistic encoder $q_\phi(z|x)$ and the probabilistic decoder $p_\theta(x|z)$. The parameters of the MLPs are denoted by ϕ and θ. The latent dimension is set to 16.

Figure 3 shows linear interpolations between latent vectors to demonstrate that the model works well. To create cosmology images, generated star images are distributed randomly inside an image with black background. The number of stars per image is normally distributed and estimated from the labeled images. Table III shows the estimated similarity scores for these images.

4) cDCGAN on stars: While our results from the DCGAN do follow the pattern from the cosmology data, the stars generally lack the necessary precise detail to make them look realistic. Hence, we also explored a combination of the straightforward adhoc and the DCGAN approach. We trained a smaller DCGAN on our data set of extracted stars. For the generation of the high-resolution cosmology images, the generator was fed some random latent code and the resulting star images were placed on a random spot in a black background image similar to the adhoc method.

While this approach already produced quite convincing results, we decided to go one step further. We found that there were different kinds of stars inside the cosmology images; some very bright, some higher than broad, some very dark, some completely round. However, considering our large data set of about $15\,000$ star images, most of them looked very similar, meaning they could potentially be divided into some small amount of individual classes. We trained a simple deep convolutional autoencoder (DCAE) on the star images for 400 epochs. We then applied k-means clustering on each image's latent code produced by the DCAE's encoder. In this way, the stars were separated into five distinct classes. We did not have to worry about overfitting the DCAE, because it was only used on the data it was trained on.

Afterwards, we used the clustered data to train a conditional DCGAN (cDCGAN). Beside the image and latent code, the generator and discriminator were also fed the class label belonging to the star. To prevent the cDCGAN from overfitting, we generated complete images every 5 epochs and included the best RF and CNN classifier models to score them. Since this would not have been an objective measurement for the scores, we could not simply distribute the stars at random in a black background image. Hence, we measured the number of occurrences of each star class per cosmology image and approximated it with a normal distribution bound to unsigned integers, i.e. excluding fractional and negative amounts of stars. We assumed the positioning of a star to be uniformly at random. In order for the final result to be deterministic, we then used a prototype cDCGAN model to distribute stars following the specified distributions of number and positioning. Each time a set of 100 images was produced, the best RF and CNN models scored them. We repeated this process 2000 times and saved those random numbers that produced the highest score. With all the numbers saved, the next cDCGAN could then be trained with a deterministic validation procedure.

The architecture of the cDCGAN was also adopted from the reference implementation on the TensorFlow website [8]. No adjustments to the resolution had to be made, only the conditional property had to be added. The architecture of the conditional generator is shown in Figure 4. It uses a total amount of $3\,212\,480$ trainable parameters. Figure 5 shows samples of the five different star classes. The discriminator uses $269\,313$ trainable parameters. Its architecture is illustrated in Figure 6. The number of parameters is quite high and could have been reduced. However, since the generation of such small images is very stable, no architectural experiments were conducted. The cDCGAN was trained for 185 epochs, and about half the training time was spent on validation.

5) PixelCNN: For the PixelCNN, we adopted the code from Gupta [13], which follows the work from van den Oord et al. [14]. We evaluated a conditional and an unconditional setting. The results looked strictly worse than those of the (c)DCGAN in both settings, so we did not bother to include the code any further into our existing project.

III. Results

The evaluation for our similarity scorer approximators is pretty straightforward. All the models were evaluated on Kaggle. During the training of the CNN, we also calculated the MAE of the model on the validation set that was kept separate from the training set. These results are visible in Table I. While the RF models were also initially trained on a training set and were validated locally with the rest of the data from the scored data set, the models used for the approximation of the query set were trained on the

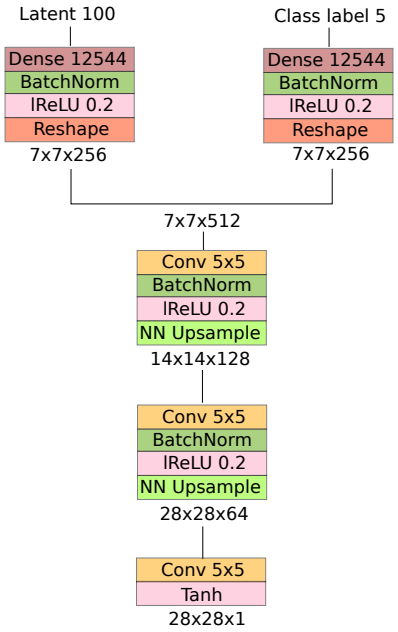

Figure 4. Illustration of the architecture of the conditional generator.

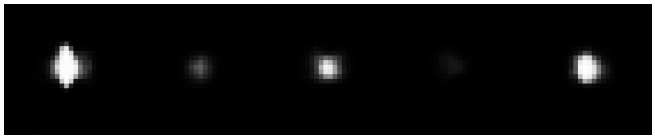

Figure 5. Samples of the five different star classes generated by the cDCGAN.

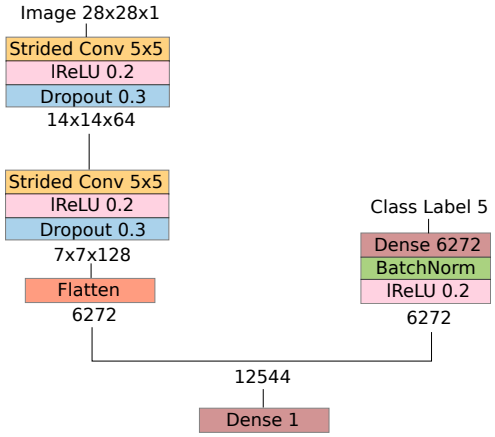

Figure 6. Illustration of the architecture of the conditional discriminator.

Model	MAE (loc.)	STD (loc.)	MAE (pub.)	MAE (priv.)
RF (baseline)	-	-	0.258	0.287
CNN	0.174	0.279	0.168	0.188
RF (best)	-	-	0.095	0.105

Table I
MEAN ABSOLUTE ERROR (MAE) AND STANDARD DEVIATION OF THE ABSOLUTE ERROR (STD) OF OUR MODELS APPROXIMATING THE SIMILARITY FUNCTION.

Approximator	Training time	Scoring time
RF (baseline)	1.5 h	2 min
CNN	24 h	1 min
RF (best)	1.5 h	2 min

Table II
TRAINING AND SCORING TIME FOR OUR VARIOUS SIMILARITY FUNCTION APPROXIMATORS (INCLUDING LOADING AND PREPROCESSING DATA).

Model	RF MSS	RF STD	CNN MSS	CNN STD
AG (baseline)	1.718	0.910	1.450	0.697
large DCGAN	1.069	0.717	1.154	0.773
VAE	1.690	0.555	2.031	0.662
cDCGAN	3.018	1.220	2.746	1.048

Table III
MEAN SIMILARITY SCORE (MSS) AND STANDARD DEVIATION OF THE SIMILARITY SCORE (STD) OF THE IMAGES GENERATED BY OUR MODELS ESTIMATED BY OUR CNN AND OUR BEST RANDOM FOREST REGRESSION MODEL (RF).

whole data set. As such, we do not have local scores for the RF models in Table I. We have also included training and scoring time for these models in Table II. We built all our neural networks with the Keras (Chollet et al. [15]) API using TensorFlow (Abadi et al. [16]) backend and trained them on the Leonhard cluster, using the NVIDIA Tesla V100-SXM2 32 GB GPU. One can clearly see that our best RF model outperforms the CNN in both score and performance time.

To evaluate our generators we used our two best similarity function approximators, i.e. our best RF model and CNN, to calculate the mean similarity score of the images. We chose not to train and use a conventional classifier to evaluate the generated images, as the Kaggle competition states that the

Generator	Training time	Generation time
AG	-	<1 min
large DCGAN	2.5 h	<1 min
VAE	2 min	<1 min
cDCGAN	30 min	<1 min

Table IV
TRAINING AND GENERATION TIME FOR OUR VARIOUS GENERATORS.

	# features	MAE (pub.)	MAE (priv.)
Pixel intensity (baseline)	32	0.258	0.287
Pixel intensity + oriented gradients	$32 + 36$	0.183	0.203
Power spectrum	48	0.107	0.124
Power spectrum + ROI	$48 + 48$	0.095	0.105

Table V
MEAN ABSOLUTE ERROR (MAE) FOR RANDOM FOREST REGRESSION USING DIFFERENT FEATURES.

images will be scored according to their mean similarity to a prototypical cosmology image. As such we decided to use the same evaluation process. These results are visible in Table III. The time needed for training and generating images for the various models is displayed in Table IV.

IV. Discussion

Table IV shows that besides the highest MSS, the cD-CGAN also reaches the highest STD. This is because we try to approximate the star distribution with "good" random numbers by choosing those giving the highest MSS. Instead, an additional neural network could have learnt to produce the "perfect" set of cosmology images by learning the star distribution on its own. The trained CNN regression model could have been used as a loss function. However, because of gradient computations, the RF model could not have been used.

Present classification is mostly dominated by CNNs. Still, our RF model performed significantly better than our CNN. This is due to the fact that the image are very sparse, since they are mostly black and only contain a few small stars, and the range of images is very small, i.e. they all mostly look the same (beside the ones that are definitely not cosmology images). In general, an algorithm that scores an image according to its histograms will not be able to achieve as high an accuracy as a well-built neural network.

V. Summary

By selectively extracting the most important features from images and learning to generate only these, we achieved generated images superior than if the generative model learns to generate the image as a whole. This is judged by our similarity scorers. "superior than if" sounds strange, should probably be something with "superior to"...

For the similarity scorers of cosmology images, we have found that preprocessing in the form of the power spectrum and the use of ROI histograms improved the approximation by a significant amount. Lastly, despite being more complex, a CNN does not always necessarily outperform a simpler model such as a RF.

References

[1] T. K. Ho, "Random decision forests," in *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1)*, 1995, pp. 278–. [Online]. Available: http://dl.acm.org/citation.cfm?id=844379.844681

[2] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[3] A. Bosch, A. Zisserman, and X. Munoz, "Image classification using random forests and ferns," in *2007 IEEE 11th International Conference on Computer Vision*, Oct 2007, pp. 1–8.

[4] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[6] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[7] A. Radford, L. Metz, and S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," *arXiv e-prints*, 2015. [Online]. Available: https://arxiv.org/abs/1511.06434v1

[8] TensorFlow, "Deep Convolutional Generative Adversarial Network." [Online]. Available: https://www.tensorflow.org/beta/tutorials/generative/dcgan

[9] A. Odena, V. Dumoulin, and C. Olah, "Deconvolution and checkerboard artifacts," *Distill*, 2016. [Online]. Available: http://distill.pub/2016/deconv-checkerboard

[10] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations*, 2014. [Online]. Available: http://arxiv.org/abs/1312.6114

[11] C. Doersch, "Tutorial on variational autoencoders," *arXiv preprint*, 2016. [Online]. Available: https://arxiv.org/abs/1606.05908

[12] TensorFlow, "Convolutional Variational Autoencoder." [Online]. Available: https://www.tensorflow.org/beta/tutorials/generative/cvae

[13] A. Gupta, "Image Generation with Gated PixelCNN Decoders," 2016. [Online]. Available: https://github.com/anantzoid/Conditional-PixelCNN-decoder

[14] A. van den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu, "Conditional image generation with pixelcnn decoders," *CoRR*, vol. abs/1606.05328, 2016. [Online]. Available: http://arxiv.org/abs/1606.05328

[15] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[16] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/