Unsupervised Representation Learning with DCGAN - A Students' Project

Andreas Peintner, Josef Gugglberger, and Tobias Kupek

Abstract

The project is based on work from Radford et al. 2016. In this students' project, we summarize the main contributions from the paper and confirm some of the results with our own re-implementation. Additionally, experiments on the size of the utilized dataset and investigations on the output and input dimensions of the model are performed.

1 Introduction

The paper from Radford et al. 2016 addresses generative methods to produce novel samples, such as images, that are hard to distinguish from real-world samples. Examples for the use of such generation processes are speech synthesis, image-to-image translation, and image inpainting.

Previous solutions are dealing with performance issues, the lack of a latent representation, the stability of the generation process, small output resolutions, and a limited variation. Most of the approaches in the literature address some of these issues but have at least one major downside. The presented paper is based on generative adversarial networks (GAN) and focuses on a stable generation of images.

The work has shown major advances in the generation of stables artificial images that are hard to distinguish from real-world samples. The authors present five guidelines and demonstrate them in a model architecture to generate RGB images with a resolution of $64 \times 64 \times 3$ pixels.

Additionally, experiments on the latent space are shown, in which the input representation can be altered to show targeted features in the resulting outputs.

In our students' project, we take a deep dive into the presented solution. We explore the architecture on our own and run additional experiments to further explore the properties of DCGANs.

The rest of the paper is organized as follows: In Section 2 we study the existing work and explain the most important concepts and advances that have been made by the authors towards stable GANs. In Section 3 we re-implement the solution, confirm some of the presented results, and

demonstrate the flexibility of the solution on a completely new dataset. Section 4 investigates the impact of different sizes of the used training dataset, presents an improved architecture with doubled output dimensions, and explores the impacts of smaller and larger input vectors.

2 Background

GANs are deep neural networks that consist of a generator and a discriminator network. The generator module learns to create images that are increasingly hard to distinguish from real-world samples. Within this process, the discriminator module will learn to separate between the fake and real samples. Both modules are tied together and will work as adversaries during training, each trying to optimize their own objectives.

Figure 1 shows such an architecture, where a random input is upscaled to an image of the target dimensions. On the right-hand side, an input of the same dimension is fed to a neural network and convolved to a binary output. The output indicates if the image is classified as a *real* image from the dataset or a *fake* image produced by the generator.

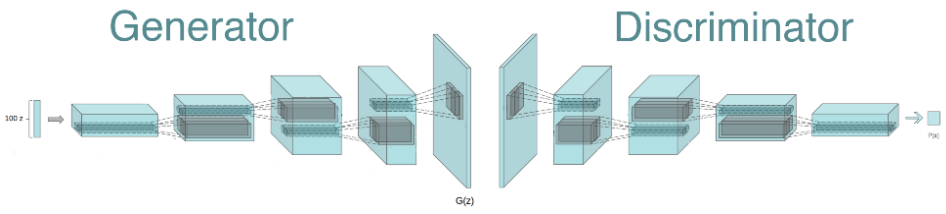


Figure 1: Visualized DCGAN architecture with generator and discriminator

While the generator starts with random input noise, the discriminator gets real images from the training dataset, and fake images from the generator and tries to differentiate between those. Through this procedure, the generator and discriminator work against each other and gradually improve in their tasks. The training loss that is collected from the generator and discriminator output is propagated back through the complete model and adjusts weights from all layers.

However, prior work has shown that building an architecture like this is not trivial Goodfellow et al. 2014 and the solutions suffer from noisy, incomprehensible, or wobbly outputs. The presented DCGAN architecture takes one step ahead to generate stable images without too many artifacts.

3 Setup

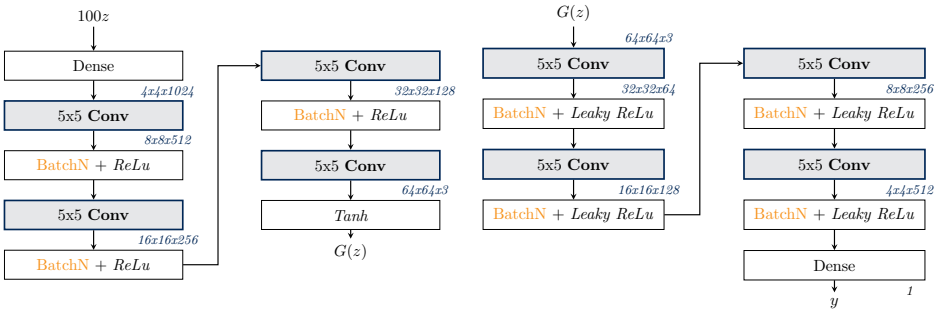
We now describe our implementation of the DCGAN and in particular the used parameters for training. To confirm the correctness of our model, some of the results from the original paper are reconstructed.

3.1 Reimplementation

Our reimplementation is based on the same architecture presented by Radford et al. which is shown in Figure 2.

The generator takes a random vector of size 100 as an input and performs a step-wise up-convolution to reach a final output dimension of $64 \times 64 \times 3$. The intermediate layers consist of four strided-convolutional layers that are followed by a BatchNormalization layer (Ioffe and Szegedy 2015) and a *ReLu* activation function. The last convolutional layer is followed by a *tanh* activation to produce the final fake-image output.

The discriminator takes an input of the same dimension as the generator output and performs a step-wise down-convolution to reach a binary output. The strided convolution is performed in four layers that are each followed by a BatchNormalization layer and a *LeakyReLu* activation function.



(a) Generator

(b) Discriminator

Figure 2: Layers of the DCGAN generator and discriminator with fixed dimensions

For both parts of the network, the Adam optimizer Kingma and Ba 2015 is used with tuned hyperparameters. As stated in the original paper, we use a lowered learning rate of 0.0002 and a reduced β_1 parameter with a value of 0.5. We can confirm that these adjustments indeed led to more stable results.

3.2 Reproducing Results

To demonstrate the correctness of our reimplementation, we reproduce some of the prior results. In particular, we train our model on the LSUN bedroom

dataset (Yu et al. 2015) to generate new bedroom scenes.

While the paper utilizes the full dataset, consisting of around 3 million samples, we are working on a subset with 100.000 samples ($\approx 3\%$). This is due to a restriction of computational resources. We compensate for the lack of samples by $30\times$ more epochs used for training and, therefore, achieve the same amount of training steps. A deeper analysis of the required amount of training samples to achieve comparable results is shown in Section 4.1.

(a) Original results (b) Our results

Figure 3: Samples after *five* and *150* training epochs accordingly on the LSUN bedroom dataset

To confirm that our model learns strong representations from the dataset, we show an interpolation between a series of nine random points in the input vector following the experiment "6.3.1 Forgetting to draw certain objects" (Radford et al. 2016, p. 6). Figure 4 demonstrates that the space learned from our model has smooth transitions and, as in the original experiment, every image plausibly looks like a bedroom.

Figure 4: Reproduction of the experiment to interpolate two input vectors to show a smooth transition between two scenes.

3.3 Training on a new Dataset

To prove that the solution is independent of the used dataset and can be adapted to many other image domains, we train the architecture on a completely new dataset.

In order to deal with a completely new type of images, we chose an image set consisting of 12.000 Pokemon images (*Monster GANs: create monsters for your game* 2020). We note that for training, no adjustments to the model architecture have been made.

Figure 5 shows 16 samples of the resulting model after training for 1-million steps (equivalent to 90 epochs on the chosen dataset). We notice, that the model learned to create small Pokemon-like monsters that are clearly distinguished from the background. Additionally, a vast variety of colors and shapes exists in the outputs.

The example of the Pokemon dataset shows the flexibility of the model architecture. With a sufficient amount of training samples, it can learn representations even from different image domains.

Figure 5: Samples after training the DCGAN model for 1-million steps with the Pokemon dataset

4 Experiments

In the following, we present three of our own experiments, concerning the size of the training datasets, increasing the output dimensions of the model, and exploring the effects of altering the size of the input vector.

4.1 Dataset Sizes

Radford et al. 2016 demonstrates most of their results using the LSUN bedroom dataset (Yu et al. 2015).

For this, they train their architecture on the full-size unsupervised dataset, including $\approx$ 3 million sample images from bedrooms.

When comparing to other commonly used scene-recognition datasets as Places205 (Zhou, Lapedriza, et al. 2014), this seems a lot. Places205 has around 10.000 to 100.000 samples per class, which is only around 3% of the used LSUN set.

In our first experiment, we are questioning the required size of the training dataset. The training samples are used by the discriminator to learn real representations of the scene and distinguish between real and faked images. In theory, the representation could be learned by a smaller amount of images as well, as it is shown in the literature (Zhou, Khosla, et al. 2016).

The explore this field, we will train four different models with reduced datasets of size 100, 1.000, 10.000, and 100.000. To have a comparable amount of time for the model to learn representations, the number of epochs has to be scaled up respectively. Therefore each model is trained for 10-million steps.

(a) 100 samples, 100.000 epochs (b) 1.000 samples, 10.000 epochs

(c) 10.000 samples, 1.000 epochs (d) 100.000 samples, 100 epochs

Figure 6: Results of the training with different sized of the dataset, while maintaining the same amount of training steps

For each trained model, Figure 6 shows 16 randomly generated samples. It is easy to see that the model trained on 100 samples (see Figure 6a) has not sufficiently learned from the training images. The outputs are blurry, unstable, and most important, show an extremely low variance in colors and shapes.

Figure (see Figure 6b) shows outputs of the model trained on 1.000 samples. While a clear improvement can be recognized in comparison to the first model, the images still lack clearly distinguishable features and general stability.

The following two models trained on 10.000 and 100.000 samples (see Figure 6c and 6d) show outputs that are comparable to the original results of the paper. Surprisingly the improvements from samples in Figure 6c to samples in Figure 6d are not very significant. It seems that in our test, the training stagnates on a dataset size in the range of 10.000 and 100.000 and no major improvements on the image quality can be made by adding more

training samples.

Although the image quality does not increase, generated outputs should still gain more variance, when a larger dataset is fed into the network.

4.2 Output Dimensions

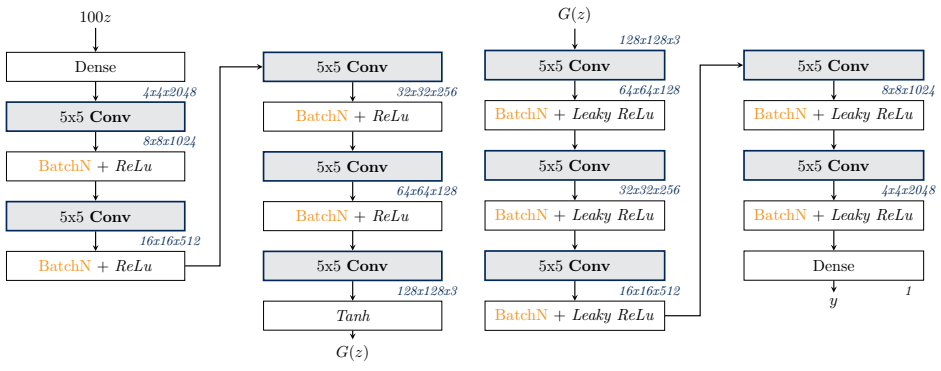
A major restriction of the architecture shown by Radford et al. 2016 is the fixed size of the generated image, which is limited to $64 \times 64 \times 3$. The paper does not mention if it is possible to increase these dimensions.

In our experiment we adjust the presented architecture to generate images with twice the size, resulting in a generator output dimension of $128 \times 128 \times 3$. Respectively the discriminator has to be adjusted to handle inputs of the new size.

The new generator architecture builds upon the basic setup, introduced in Section 3. After the last convolution layer and before the *tanh* activation function, we introduce a new block, consisting of a BachNormalization layer followed by a *ReLu* activation function and an additional convolution layer. As in the original architecture, the final output is produced by the *tanh* activation. To reach the required output dimensions, the amounts of channels of the previous convolution layers have to be doubled.

On the side of the discriminator, the first convolution layer is adjusted to directly start with 128 channels. The channels of the ongoing convolutional layers are doubled, respectively. Before the final Dense layer, we introduce an additional convolutional layer to downsample the input once more before the binary output is given.

The adjusted architecture with the additional layers is shown in Figure 7.



(a) Generator (b) Discriminator

Figure 7: Layers of the DCGAN generator and discriminator with increased output dimension

We note, that the new architecture consists of a total of $72,922,240$ trainable parameters for the generator, which is around $4\times$ more than the

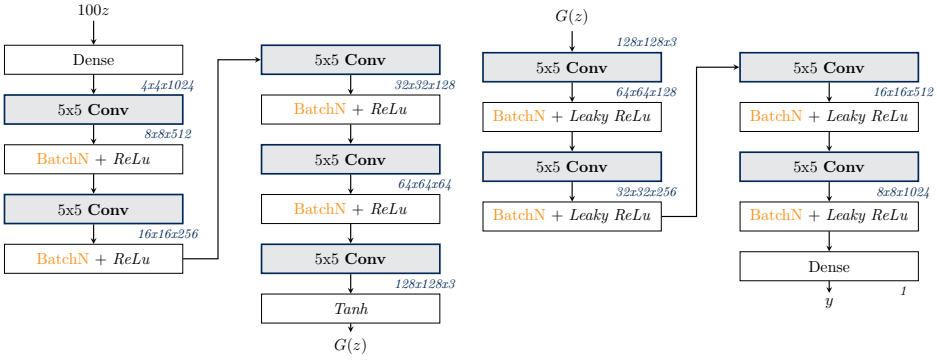
original DCGAN model. This leads to a more complex training procedure, that is clearly noticeable in practice.

With the same hardware setup, we measured 28× more time required for each training step (averaged over several thousand steps).

While we can still observe comparable outputs after the first epochs, the massively increased training time makes the DCGAN method impractical for output dimensions of $128 \times 128 \times 3$ or higher.

We, therefore, suggest another architecture to generate images of a larger dimension. The new architecture only extends the generator part by an additional layer. The previous intermediate convolutional layers of the generator are not scaled. On the discriminator side, we only adjust the channels of the original layers to be able to handle the new input size.

The exact layer definitions of the new architecture are shown in Figure 8 Our new model has $19,053,120$ trainable parameters for the generator, which is now only 1% more than the parameters of the original DCGAN model.



(a) Generator (b) Discriminator

Figure 8: Layers of the DCGAN generator and discriminator with increased output dimension without channel adjustment

With the adjusted setup, a 5× higher training time is observed compared to the original architecture. This is much more practical than the first attempt and can, therefore, easier be used to generate outputs with a higher resolution.

Figure 9 shows 16 samples from our adjusted model after 1.8 million training steps. One can observe that the output has comparable stability and a similar amount of fragments as the samples generated with a lower dimension.

Although we were, with some obstacles, able to generate images with twice the resolution, we still see the scalability to high-resolution images as a major limitation of the DCGAN architecture. The gained training

Figure 9: Samples from our adjusted architecture (see Figure 8) with an output dimension of $128 \times 128 \times 3$

overhead for doubling the dimensions is definitely not negligible and leads to bigger challenges when aiming for higher resolutions.

4.3 Input Dimensions

Besides the fixed output dimension, which is explored in the previous section, Radford et al. 2016 uses an input vector of size 100 to feed a random input to the generator. During training, this z-vector develops a latent space and represents various features of the generated outputs.

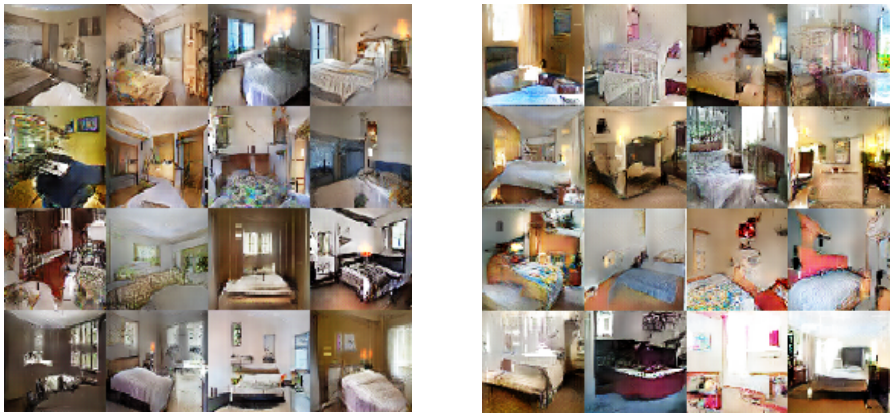
The effects of altering the values of the latent space are already explored in several experiments of the original paper. Additionally, we explore if the input dimension 100 is well chosen to represent the features. In particular, we show the effects of a noticeable smaller and larger input space.

For this, we train two new variants of the architecture, one with a reduced input vector of size 20 and one with an increased input vector of size 500. To visualize the effects of the two vectors, we look at some of the generated samples and repeat the experiment on the latent space from Section 3.2.

At first, we note that we could not measure any differences in training performance. Both models require the same amount of time to develop a similar set of features and comparable stability of the generated outputs.

Figure 10 shows 16 sample outputs generated from both models. Considering the variance in colors and features, no visual differences can be obtained. Both generators generate windows, beds, and furniture with different colorization.

As the variance of the inputs does not seem to be affected by a smaller dimension, we suppose the variance in inputs is now shifted towards the



(a) $|z| = 20$ (b) $|z| = 500$

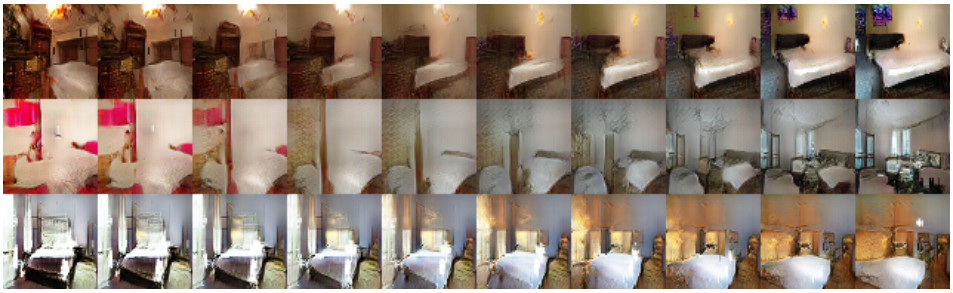
Figure 10: Samples after *four* million training steps with different dimensions for the input vector

absolute values of the input. In other words, the same adjustment of an input value should lead to a bigger visible effect in the generated output.

To give an indication for this, we repeat the experiment of Section 3.2 where two input vectors are interpolated in ten steps.

Three results of each model are shown in Figure 11. We observe a visible difference in the transition between the left and the right bedroom. We notice several steps of the model with a smaller input (see Figure 11a) that are unexpected. For example in the transition in the second row from samples three to four, The red left side disappears completely, while all transitions of the model with a larger space (see 11b) seem much smoother.

To quantify this effect, more experiments on the latent space and in particular measurements of the output variance are required, that are not part of this student's project.



(a) $|z| = 20$

(b) $|z| = 500$

Figure 11: Interpolation of input vectors on models with different input sizes

5 Conclusion

In our project, we made an in-depth analysis of the paper "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks" by Radford et al. 2016.

We summarized the important concepts and presented the general setup. Additionally, we re-implemented the solution and re-evaluated some of the original experiments to demonstrate the correct behavior of our code. We extended the demonstration to a completely new dataset, which indicates that DCGANs can learn arbitrary image representation.

Further on, we have shown several experiments that support and extend the claims of the paper by supplementary analyses on the size of the dataset and the input and output dimensions of the generator.

We demonstrate that stable DCGANs can be trained with a highly reduced set of training images. Our model with only 0.3% of the original training samples can generate comparable results. Regarding the output dimension of the images, we explore the impact of doubling the layer sizes and propose an adjusted architecture to generate images with twice the resolution. Experiments on the dimensions of the input vector show that the size is not necessarily fixed by $|z| = 100$, but other dimensions can be considered as well.

By far, our experiments are not sufficient to eventually give a proof for

the shown results, but they serve as a first indication and hopefully are an inspiration to future work in the field.

References

Goodfellow, Ian J., Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio (2014). "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems, Montreal, Quebec, Canada*. Ed. by Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, pp. 2672–2680. URL: http://papers.nips.cc/paper/5423-generative-adversarial-nets.

Ioffe, Sergey and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the 32nd International Conference on Machine Learning, ICML, Lille, France*. Ed. by Francis R. Bach and David M. Blei. Vol. 37. JMLR Workshop and Conference Proceedings. JMLR.org, pp. 448–456. URL: http://proceedings.mlr.press/v37/ioffe15.html.

Keras.io Python Deep Learning Library (2020). https://keras.io/. Accessed: 2020-05-09.

Kingma, Diederik P. and Jimmy Ba (2015). "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR, San Diego, CA, USA*. Ed. by Yoshua Bengio and Yann LeCun. URL: http://arxiv.org/abs/1412.6980.

Monster GANs: create monsters for your game (2020). https://medium.com/@yvanscher/using-gans-to-create-monsters-for-your-game-c1a3ece2f0a0. Accessed: 2020-05-15.

Radford, Alec, Luke Metz, and Soumith Chintala (2016). "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". In: *4th International Conference on Learning Representations, ICLR, San Juan, Puerto Rico*. Ed. by Yoshua Bengio and Yann LeCun. URL: http://arxiv.org/abs/1511.06434.

Tensorflow.org Machine Learning Framework (2020). https://www.tensorflow.org/. Accessed: 2020-05-09.

Yu, Fisher, Yinda Zhang, Shuran Song, Ari Seff, and Jianxiong Xiao (2015). "LSUN: Construction of a Large-scale Image Dataset using Deep Learning with Humans in the Loop". In: *CoRR* abs/1506.03365. arXiv: 1506.03365. URL: http://arxiv.org/abs/1506.03365.

Zhou, Bolei, Aditya Khosla, Àgata Lapedriza, Aude Oliva, and Antonio Torralba (2016). "Learning Deep Features for Discriminative Localization". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR, Las Vegas, NV, USA*. IEEE Computer Society, pp. 2921–2929. DOI: 10.1109/CVPR.2016.319. URL: https://doi.org/10.1109/CVPR.2016.319.

Zhou, Bolei, Àgata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva (2014). "Learning Deep Features for Scene Recognition using Places

Database". In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems, Montreal, Quebec, Canada*. Ed. by Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, pp. 487–495. URL: http://papers.nips.cc/paper/5349-learning-deep-features-for-scene-recognition-using-places-database.

Appendix

Code

Our reimplementation of the DCGAN model is based on Tensorflow 2.0 (*Tensorflow.org Machine Learning Framework* 2020) and uses the Keras functional API (*Keras.io Python Deep Learning Library* 2020). The code including additional Python scripts for loading the datasets and visualizing the results is publicly available at
https://github.com/moejoe95/theDCGANProject.