Generating Novel Cats Using Different Generative Adversarial Networks

Federico Ferlito (s2936860) Remo Sasso (s2965917) Leon Pater (s2380153) Jonathan Reid (s2905248)

University of Groningen, PO Box 72 9700 AB Groningen

1 Introduction

Generative Adversarial Network (GAN) [9] is a class of unsupervised machine learning systems in which two different neural networks compete in a game (such as a zero-sum game) in order to train each other. These two are called **generator** and **discriminator**. The former one has the goal to create as output samples with the same distribution of the training data, taking as input a vector from a random distribution. The latter one has to judge whether its input is a real image or a generated one, thus it will have two classes as output (real or fake). The training procedure for the generator is to maximize the probability of the discriminator to make mistakes, i.e. it has to fool the discriminator into believing that its generated images are real. Back-propagation is applied to both networks so that the generator produces better images, and the discriminator is better at recognizing synthetic images. Competition drives both to improve, until counterfeits created by the generator are indistinguishable for the discriminator.
In the context of Deep learning, many variants of GAN include convolutional layers to increase the capacity of these models for unsupervised tasks. In this report, we examine some of these implementations, which are described in Section 2.1. These include DCGAN [13], WGAN [7] and ProGAN [11]. The task for these networks is to learn to generate cats' faces, using a collection of open source data-sets, described in Section 2.2. In order to compare these models, we cannot use the objective function of the networks, since it one only tells us how well they're doing relatively to the opponent. Thus, to get more insights on the quality and diversity of the generated images, two different metrics are used, namely Inception Score (IS) [14] and Fréchet Inception Distance (FID) [10], which are discussed in detail in Section 2.3.

2 Methods

2.1 Architectures

Each of the aforementioned architecture is semi-optimized separately with respect to its hyper-parameters. This includes but is not limited to: the dropout rate, batch-size, momentum, iterations, and the learning-rate for both systems. This was accomplished by first studying parameter-combinations in state-of-the-art models, followed by tweaking intuitively based on obtained results. Finally, the average Fréchet Inception Distance (FID) and Inception Score (IS) was measured on the final models of each architecture-setting. A final 5-run average was taken because some models seem highly dependent on the initialization process, thus the average gives us a better indicator on the actual performance of the architecture. We will now specify the characteristics of each implemented architecture.

DCGAN

This is one of the most popular design for GAN, and it's considered one of its foundation pillar. It is mainly composed of convolutional layers, and it uses convolutional strides for the downsampling of

the discriminator, instead of using max pooling or average pooling. Further it uses transposed convolutions for upsampling of the generator instead of having fully connected layers. On top of that, batch normalization is used to help the gradient flow smoothly. For the activation functions, the authors [13] suggest to use ReLU units for the generator, except for its last layer, which should use $\tanh$ units, and LeakyReLU for all the layers of the discriminator. However, we have also tested the performance of different activation functions. The structure of the generator is shown in Figure 1.

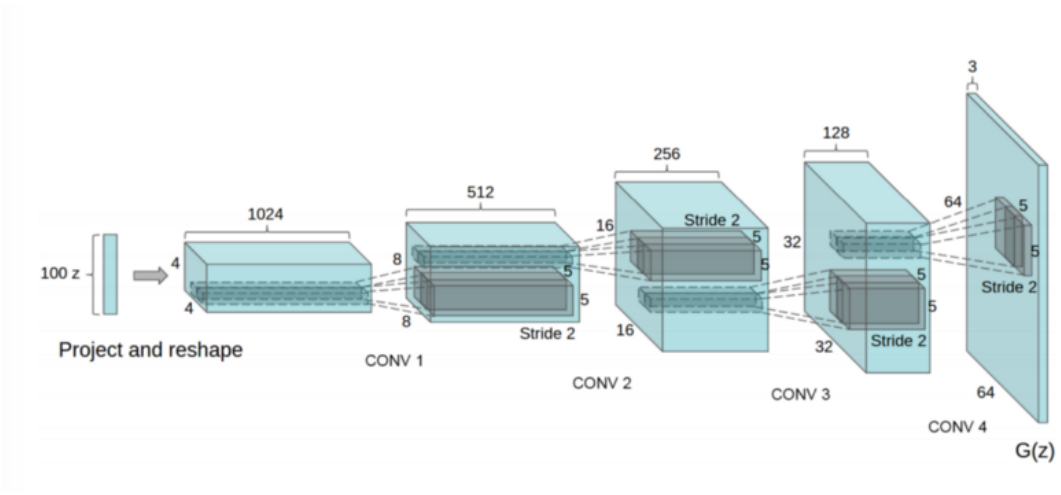


Figure 1: Architecture of the generator used by the authors of the DCGAN [13]

WGAN

The original version of GAN and other variants, like DCGAN, have the problem of optimizing the discriminator faster than the generator: this happens when the generated distribution Q is too far from the ground truth P, since the generator will be too bad and the discriminator cannot give useful information to it to improve. This leads to having a vanishing gradient problem when using binary cross entropy loss as cost function.
WGAN (Wasserstein GAN) introduces some changes, allowing the generator to learn even if is not performing well. First, after each gradient update of the discriminator, its weights are clipped to a fixed range $[-c, c]$, such as $[-0.01, 0.01]$ as suggested by the authors [7]. Second, a new loss function is derived from the Wasserstein distance, which does not include the logarithm in the computation anymore. The discriminator, in this context, estimates the Wasserstein metric between real and generated data distributions. Compared to the JS divergence used in vanilla GANS, this metric provides more meaningful values when the distributions of real and generated images are disjoint. This is because the new cost function allows to have a smooth gradient during the whole training process, if the parameter c is set correctly. The authors of the paper mentioned suggest that this allows for more stable training of the GANs.

ProGAN

The ProGAN (Progressive GANs) uses a different training methodology for generative adversarial networks with respect to those described above. Instead of starting out with full-sized images, the ProGAN instead starts the discriminator and generator off with low resolution images. As training progresses, the resolution is increased in steps. With each increase in resolution, a new layer is added, which should allow for modeling of the new features present in the higher resolution when compared to the lower resolution version. Starting at a low resolution allows for much faster training. The progressive build up of the layers gives the network more stability, as it can learn the rougher features from the low resolution images, which are then taken care of when the network gets to learning the finer details in the higher resolution images. The same settings that were used to generate the celebrity images were used, but then on images of cat faces. The process is illustrated in Figure 2.

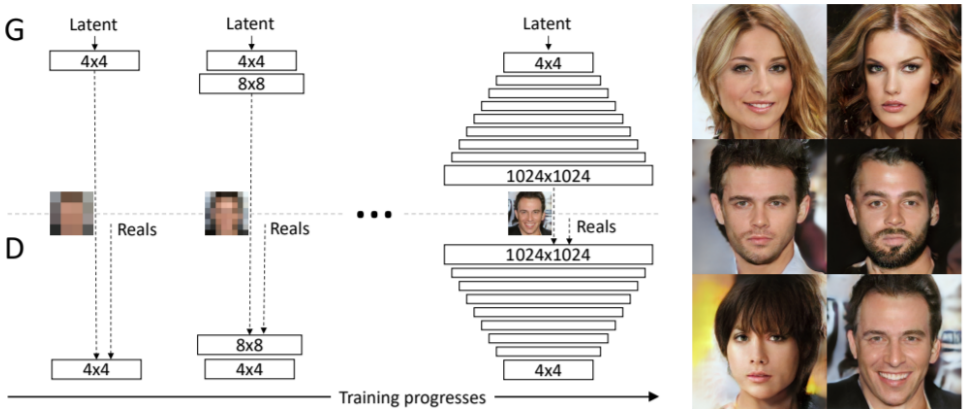


Figure 2: Figure describing the training process of a ProGAN from [11]. Discriminator and Generator both start off at low resolution, and increase in steps as training progresses. In this case, the network was trained on 1024x1024 images of celebrities. In our case, the images are 64x64 images of cat faces.

2.2 Data

For our task, we designed an ad-hoc dataset. We included images from 5 different open-source datasets, namely:

- Cats and Dogs Breeds Classification Oxford Dataset [2]
- Cute Cats and Dogs from Pixabay.com [5]
- Cat Dataset [1]
- Cats faces 64x64 (For generative models) [3]
- COCO Dataset [4]

The images were preprocessed with the openCV library [8], using a cat face detector [6], so that each image would contain only the face of the cat. This step was skipped for the last dataset (Cats faces 64x64), since the faces were already cropped there. After that, the false positive cats were removed by a visual inspection. The final number of used images comes roughly to a total of thirty thousand.

2.3 Evaluation Metrics

2.3.1 Inception Score (IS)

The Inception Score measures how much realistic the results of a GAN are, and correlate well with the human perception of image quality [14]. It measures at the same time if the produced image have variety (each image has a different class), and if each produced image contains clearly a specific class. If both the conditions are true, the score will be high, with the lowest score being zero. It takes its name from the Inception classifier, which is used to compute it. Its computation is pretty straightforward: we first take a sample of generated images, and we pass each image trough the classifier to get its probability distribution of classes. Optimally, we want each image containing a class with high accuracy. We then sum all these distributions to get a marginal distribution, which tells us about the variety of the generator. If there is variety of classes in the generated images, this distribution will be uniform.
To compute the Inception score, we compare each label distribution with the marginal distribution using the *Kullback-Leibler (KL) divergence*: we take the exponential of this value, and we average this for all the images. Since the KL divergence is high when the distributions are dissimilar, it will be high when each image has a distinct label, and the whole set has a marginal distribution covering a big range of labels. For two distributions P (real) and Q (generated), it is defined as:

$$D_{KL}(P|Q) = -\sum_{x \epsilon X} P(x) log\left(\frac{Q(x)}{P(x)}\right)$$

2.3.2 Fréchet Inception Distance (FID)

The FID [10] measures the realism of the generated images measuring the distance between the generated distribution and the true distribution of the dataset used for training. Lower scores are better, corresponding to more similarity between the real and fake samples. To compute it, we take the distance between the activation distributions of the Inception v3 pool3 layer [15] for both samples. Taking the Gaussian distribution of the activations with mean μ and covariance Σ, we compute the FID as:

$$FID(x, g) = ||\mu_x - \mu_g||_2^2 + Tr(\Sigma_x + \Sigma_g - 2(\Sigma_x \Sigma_g)^{\frac{1}{2}})$$

Where Tr is the sum of all the diagonal elements. With regards to the FID, lower is better, with a score of 0 meaning that the distributions are identical. In that case, there would be no difference between the real distribution and the generated distribution.

3 Results

In this section the performance of the various networks is reported in Table 1. The metrics used are described in detail in Section 2.3. Additionally, a small sample of the results of each network alongside a sample of the real images is given in Figure 3. The number of images used for calculating the final metrics was matched with the total number of images in the data set, roughly thirty thousand. For the FID, we list only the calculated distance. For the Inception Score, we list the mean plus or minus the standard deviation.

	FID Score	Inception Score
DCGAN	244.56	3.01 $\pm$0.06
WGAN	185.35	2.79 $\pm$0.027
ProGAN	12.77	2.69 $\pm$0.03

Table 1: Performance metrics for each of the trained GANs.

4 Discussion

The ProGAN seems to outperform the other two architectures by a great deal. This can be gleaned from the FID scores, as well as clearly from the sample results given in Figure 3. The DCGAN is then outperformed by the WGAN.

We currently only list the final score of each architecture, while the intermediate results are omitted. This is problematic because we are uncertain whether the models have fully converged yet or not; the model parameters could even oscillate, destabilize and never converge. Also, if the discriminator is too successful, the generator gradient vanishes and is unable to learn. In addition, with the current applied methods, we are unaware of mode collapse. Especially the DCGAN is vulnerable for this. Analyzing intermediate results can detect all of this happening. Unrolling the GAN by defining the generator objective with respect to an unrolled optimization of the discriminator can prevent these problems from occurring [12].

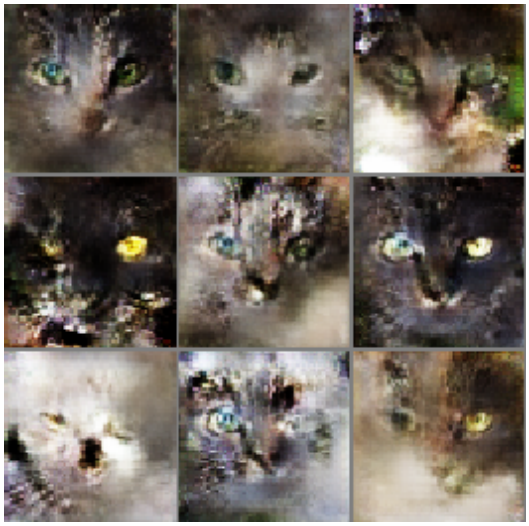
The WGAN and the DCGAN are highly vulnerable (though the latter generally more than the former) to the used parameter settings, and the performance can most certainly be increased by exploring more parameter combinations, though this is time consuming.

The ProGAN, however, is trained much more quickly, due to starting at lower resolutions, and seems to be much more robust. The FID scores for lower resolution outputs are obviously much worse, but the score stabilises and seems to vary only slightly as soon as the network reaches full resolution. The results obtained for this architecture appear to be much better than for the other two, both qualitatively, by simply looking at the generated images, further reinforced by the FID scores obtained. It is important to note that the ProGAN was pre-tuned for the celebrity face generation task, and parameter tuning wasn't explored enough to accurately determine their effects.

The Inception Score seems to have been a bad fit for this particular problem, or these particular data. A lower FID is better, but a higher IS is better. In our case, we have a low FID as well as a low IS. This

(a) Sample of real images used for training.

(b) Sample of images generated by the ProGAN.



(c) Sample of images generated by the DCGAN.

(d) Sample of images generated by the WGAN.

Figure 3: Samples of real images used for training and samples of the resulting best generated images per architecture.

may be because of our kind of data not being present in the Inception network, leading to bad scores. It could also be a result of the quality of the images. Whatever the case, the IS does not accurately reflect our performance, since it reflects the variety of classes being produced by the network (see Section 2.3 for a detailed explanation). Since the Inception network has only one class for cats in the output layer, this metric will not tell us useful information for this task. A solution may be to retrain the last layer of the network to classify different breeds of cats. The low FID and general good quality of the images for the ProGAN show us that the network performs well, despite the IS saying otherwise.
The data was still slightly dirty at the end, despite the automated cropping and further visual inspection. This was most likely due to the sheer volume of images used. In some cases, more of the cat was visible than we would have wanted, in some cases its entire body. In other cases, the cat was only a small part of the image, with the background taking up the most space. In yet other cases, there may have been, for example, a person holding the cat.
On top of that, the amount of data could have been larger. The current data set contains 29843 images. Some techniques for data augmentation could have been applied to extend it, like cropping or rotating the images. This would have led to richer outputs for the GANs. Another improvement that data augmentation would produce is the variety of images: right now the networks can only produce cats with the eyes in the same position and angle. Rotating the images of the training data would allow to produce cats in different orientations, but more research is needed. Some noise could have been added to the images to make the networks more robust. Despite not having taken any of these approaches, the ProGAN was still able to perform well.
The images used were very low in resolution (64x64). This was a somewhat conscious choice, as we had to take into account the time required to train each network. But it may have been interesting to see how higher resolution images fare on the networks, especially the ProGAN, as it may have benefited from the higher level of detail. The original paper (See Figure 2) boasts high quality 1024x1024 generated images of celebrity faces. Using a ProGAN on 64x64 images might be underutilizing its power.

5 Acknowledgements

We would like to thank the Center for Information Technology of the University of Groningen for their support and for providing access to the Peregrine high performance computing cluster.
We would also like to express our gratitude to all the cats being photographed, which made the completion of the project possible.

References

[1] Cat dataset. https://www.kaggle.com/crawford/cat-dataset. Accessed: 2019-05.

[2] Cats and dogs breeds classification oxford dataset. https://www.kaggle.com/zippyz/cats-and-dogs-breeds-classification-oxford-dataset. Accessed: 2019-05.

[3] Cats faces 64x64 (for generative models). https://www.kaggle.com/spandan2/cats-faces-64x64-for-generative-models/downloads/cats-faces-64x64-for-generative-models.zip/1. Accessed: 2019-05.

[4] Coco dataset. http://cocodataset.org/#home. Accessed: 2019-05.

[5] Cute cats and dogs from pixabay.com. https://www.kaggle.com/ppleskov/cute-cats-and-dogs-from-pixabaycom. Accessed: 2019-05.

[6] Opencv cat face detector. https://www.pyimagesearch.com/2016/06/20/detecting-cats-in-images-with-opencv/. Accessed: 2019-05.

[7] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[8] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[9] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[10] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6626–6637. Curran Associates, Inc., 2017.

[11] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. In *International Conference on Learning Representations*, 2018.

[12] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. *ArXiv*, abs/1611.02163, 2017.

[13] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. 11 2015.

[14] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training gans. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2234–2242. Curran Associates, Inc., 2016.

[15] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *ArXiv*, 2015.