Generation of Real Looking Images Using GANs

BOHNSTEDT Timo, B.Sc.
Friedrich Alexander University
timo.bohnstedt@fau.de

***Abstract*—Automatic image generation is a complex task with many applications in several domains such as policing (e.g. generating portraits from the description), styling and entertainment. In this project, advanced versions of Generative Adversarial Networks (GANs) are used to generate real looking images. A conditional GAN (cGAN) is used as a prototype, followed by an evaluation with visual examination, k-nearest-Neighbours (kNN) and Fréchet Inception Distance (FID). The evaluation indicates that a cGAN can generate realistic images of handwritten digits. Whereas, the discussion shows that more work must be done to create realistic images from datasets which contains larger and more complex images.**

***Index Terms*—Machine Learning in the Industry 4.0, EDA, Data Analytics, FAU, Python, GANs, cGANs**

1. Introduction

Generative Adversarial Networks (GANs) have been getting a lot of attention recently. They are used for different tasks, such as image-to-image translation where images can be transferred from one domain to another, text to image translation and super-resolution. GANs are introduced as an unsupervised learning model, trained on data without labels. The model learns to map noise to images reassembling examples from the training set.

There exist several challenges to model distributions in a stable and discriminative way. The two main issues are a slow training process and mode collapse.

Optimizing a GAN can be treated as optimizing the JS-divergence. This leads to the problem of vanishing gradients in early training if the original dataset and the generated dataset are very different. To encounter this problem, Goodfellow et al. suggested to use another loss function. That means optimizing a GAN can no longer be treated as optimizing the JS-divergence, and it shrinks the probability of suffering from vanishing gradients. The solution works in practice despite a scientific discourse. This is why, in this paper, we focus on mode collapse.

Mode collapse means a lack of image diversity. Instead of reproducing all classes, one or more modes can be observed in the generated data set. This means that one or more classes occur most frequently in the data set. It would be desirable if there is no mode, i.e. no class that occurs more often in relation to all other classes.

There are versions of GANs that can use additional information as input, such as conditional GANs (cGANS). Research suggests to use cGANS to get a higher chance that mode collapse does not appear [1], [2].

Deep networks require a large amount of examples for successful training. Because data acquisition is usually quite time-consuming, we want to reduce this effort by generating real world images from noise. As such we have the following two research questions:

- *Is it possible to generate real-looking images?*
- *Is it possible to introduce enough inter-class variance for training?*

This paper focuses on answering both questions.

2. Methods

GANs are a framework for estimating generative models via an adversarial process, in which two models get trained simultaneously. A generative model G that captures the data distribution and a discriminative model D that estimates the probability that a sample is drawn from the training data rather than from G [3]. Simplified, the generator tries to produce data that comes from some probability distribution.

Figure 1 highlights by example how the forward step works.[1].

Mathematically this problem can be described as solving a two-player min-max game. The problem is divided into three parts:

1) Maximize the chance to recognize real images as real.
2) Maximize the chance to recognize fake images as fake.
3) Optimize G such that it fools D the most.

The first two steps can be summarized with the following equation where $D(x)$ outputs a value indicating the chance that x is a real image [5]:

$$\underset{D}{max}\, V(D) = \mathbb{E}_{x\sim p_{data}(x)}[log\, D(x)] + \mathbb{E}_{z\sim p_z(z)}[log(1 - D(G(z)))] \tag{1}$$

Therefore x stands for the generated images, V is the value function of D and z is noise which is transformed by the network into real-looking images. Furthermore, p_{data} means that we sample x from the data set of the original images and p_z means we sample noise z from a distribution which generates random noise.

Note that the first addend is to recognize real images better and the second addend is to recognize fake images better. So, the objective function of G can be described as follows:

$$\underset{G}{min}\, V(G) = \mathbb{E}_{z\sim p_z(z)}[log(1 - D(G(z)))] \tag{2}$$

[1]The image is retrieved from the TensorFlow website. It is part of an intuitive example of how to interpret GANs [4]

By using gradient decent as an optimization technique and using the chain rule to get the derivates of the objective functions one step of particular training involves the following steps [5]:

- Sample a mini batch of m noise vectors $\{z^{(1)}, \dots, z^{(m)}\}$.
- Sample a mini batch of m m training samples $\{x^{(1)}, \dots, x^{(m)}\}$.
- Update D by doing one gradient descent step on its loss function:

$$J_D = -\frac{1}{m}\sum_{i=1}^{m}\Big[log\, D(x^{(i)}) + log(1 - D(G(z^{(i)})))\Big] \tag{3}$$

- Update G by doing one gradient descent step on its loss function:

$$J_G = -\frac{1}{m}\sum_{i=1}^{m}\Big[log\, D(G(z^{(i)}))\Big] \tag{4}$$

The cGAN can be constructed by merely feeding the labels y into both the generator and discriminator [6].

The generator G and the discriminator D are now conditioned with additional information y. So the steps of the training are now as follows [1]:

- Sample a mini batch of m noise vectors $\{z^{(1)}, \dots, z^{(m)}\}$.
- Sample a mini batch of m m training samples $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$.
- Update D by doing one gradient descent step on its loss function:

$$J_D = -\frac{1}{m}\sum_{i=1}^{m}\Big[log\, D(x^{(i)}, y^{(i)}) + log(1 - D(G(z^{(i)}, y^{(i)}), y^{(i)}))\Big] \tag{5}$$

- Update G by doing one gradient descent step on its loss function:

$$J_G = -\frac{1}{m}\sum_{i=1}^{m}\Big[log\, D(G(z^{(i)}, y^{(i)}), y^{(i)})\Big] \tag{6}$$

We use a dataset of handwritten digits as training data, which is produced by the Mixed National Institute of Standards and Technology (MNIST). This dataset is a good fit because of its sheer size, allowing GANs to work efficiently. This dataset contains 50,000 training images and 10,000 testing images, formatted as 28 × 28 pixel monochrome images. Figure 2 shows the images contained in this dataset.

The overall procedure is depicted in Figure 3. The left column depicts the generator network G, the right column the discriminator network D.

First, the noise and the labels are passed to G. The different layers of the network perform dense and reshape operations before the vectors are concatenated. Finally, G produces images at the same size as the original images. Afterwards, a real image and the generated image get passed to D, which uses dense and reshape operations as well. Finally, D *decides*

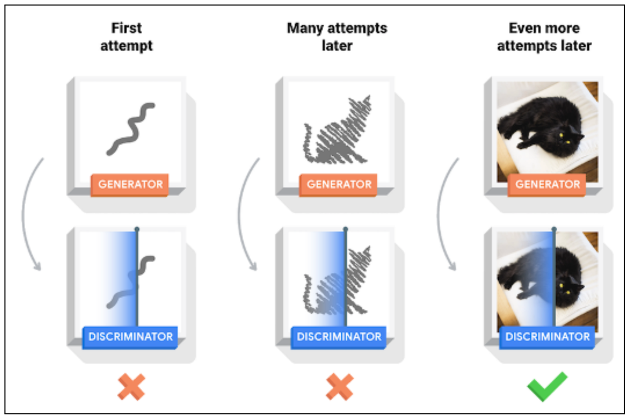

Fig. 1. Two models are trained simultaneously by an adversarial process. A generator (”the artist”) learns to create images that look real, while a discriminator (”the art critic”) learns to tell real images apart from fakes [4].

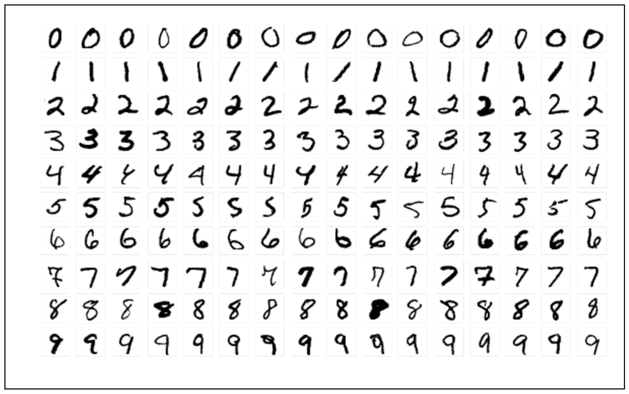

Fig. 2. A few examples of the so-called MNIST dataset which contains 60,000 images of handwritten digits.

whether or not this is a real or a generated image, as can be observed on the bottom right in Figure 3.

Precisely, the generator of the DCGAN uses upsampling layers to produce an image from random noise. It starts with a dense layer that takes this seed as input, then G upsamples serval times until it reaches the desired image size. The Network uses a leaky ReLU activation function for each layer, except the output layer which uses tanh. The generator network of the DCGAN is a CNN-based image classifier. It uses two convolutional layers, followed by a dropout layer. D uses the leaky ReLU activation function as well.

The generator network of the cDCGAN prepares the input and the labels by bringing the data into the desired form. Therefore it uses two dense, and two reshape layers each, before it gets concatenated. Then the combined input will be upsampled as in the generator network of our DCGAN.

The discriminator model of the cDCGAN is also a CNN-based image classifier. It uses two convolutional layers and two pooling layers for the generated input images . For the labels G uses two dense layers. After the two inputs get through the mentioned layers, they will be concatenated as well.

We use a training size of 50,000 images, a batch-size of 128 images, 20 epochs and we use the Adam optimizer.

GAN training faces some problems when it comes to generating real-looking images. Therefore, a conditional Deep

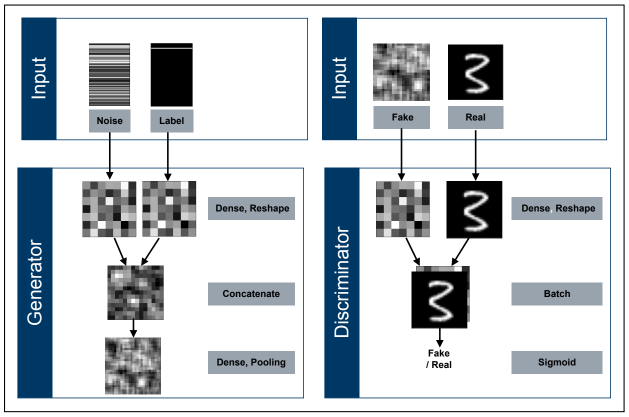

Fig. 3. Two distinct inputs get fed into the generator to create images. Different dense, reshape operations and finally the concatenation of the images transforms the images such that the model can be trained by using gradient decent as optimization technique.

Convolutional GAN (cDCGAN) offers a more stable training process. The generator of the DCGAN uses the transposed convolution technique to perform up-sampling of 2D images, such that the cDCGAN is considered as a better fit when it comes to generating real-looking images. Figure 4 highlights the data-flow and the gradients used for the backpropagation [5]. After performing on a particular step of gradient decent, a maximum error is used to adjust the weights within the layers of $\boldsymbol{G}$. Therefore a minimum error is used to adjust the weights within the layers of $\boldsymbol{D}$.

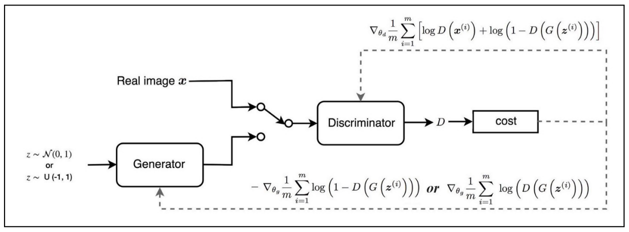
$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right]$
Real image $\boldsymbol{x}$
Discriminator
D
cost
$z \sim \mathcal{N}(0,1)$ or $z \sim$ U (-1, 1)
Generator
$-\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$ or $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$

Fig. 4. The Figure summarizes the data-flow and the gradients used for the backpropagation [5].

$\boldsymbol{G}$ is implemented such that $-\boldsymbol{log}\,\boldsymbol{D}(\boldsymbol{G}(\boldsymbol{z}))$ instead of $\boldsymbol{log}(1 - \boldsymbol{D}(\boldsymbol{G}(\boldsymbol{z})))$ is minimized as suggested in the original GAN paper [5]. It ensures stronger gradients for $\boldsymbol{G}$ early in training while keeping the same fixed point dynamics of $\boldsymbol{G}$ and $\boldsymbol{D}$, as shown in the value function (Equation 2).

The performance of $\boldsymbol{G}$ compared to $\boldsymbol{D}$ is usually poor in the early stages of the training process. That means $\boldsymbol{D}$ can easily distinguish fake examples from real images. Therefore the learning of $\boldsymbol{D}$ is implemented such that it learns slower, which is done by running more than one training step for $\boldsymbol{G}$ for every step of $\boldsymbol{D}$ and lowering the learning rate of $\boldsymbol{D}$.

GANs and cGANs stipulate a high-level framework with a lot of freedom in designing its various components. The prototype which will be used in order to find answers to the research question uses a so-called conditional deep convolutional neural network (cDCGAN) and uses the hyperparameters suggested by Isola et al. [1]. To set a baseline within the evaluation a DCGAN will be implemented as well.

3. Results

Great care must be taken when it comes to presenting and evaluating the results of any GAN. The objective functions presented in the methods section can not tell necessarily the quality of the image. Furthermore, there is not an exact definition of good quality. The objective functions for G and D are relative measurements. Their results indicate the performance with respect to each other. For this reason, we consider a visual examination, followed by quantitative measurements. Furthermore, quantitative measurements, such as the inception score and the Fréchet Inception Distance (FID), can be combined with a qualitative evaluation approach to provide a robust evaluation [7], [1].

When it comes to the evaluation of the results visual examination, k Nearest Neighbour (kNN) and the FID is used.

FID is a metric that calculates the distance between feature vectors calculated for real and generated images.

The score summarizes how similar the two groups are in terms of statistics on computer vision features of the raw images calculated using a CNN-based model which can be used for image classification. Lower scores indicate the two groups of images are more similar, or have more similar statistics, with a perfect score being 0.0, meaning that the two groups of images are identical. [8].

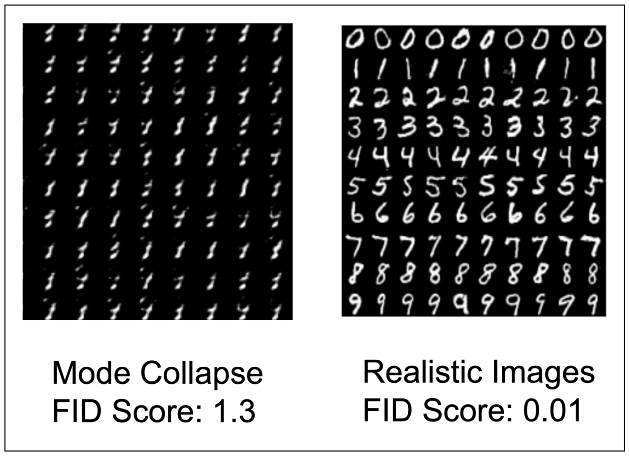

Fig. 5. Images generated by the DCGAN (left) and the cDCGAN (right) and their corresponding FID-scores show that the images generated by DCGAN are more realistic.

Figure 5 highlights the result of the visual examination. The images produced by the cDCGAN (right) look quite real, whereas the images from the DCGAN have just one mode (left). The FID score for the DCGAN and the cDCGAN is also shown in Figure 5. The score indicates that the results from the visual examination are correct because the score of the DCGAN generated images is proportionally higher in comparison with the cDCGAN generated images. Figure 6 highlights the results of the kNN approach. In the dataset generated by the DCGAN, just on mode appears if we search for the nearest neighbour in the original dataset. Whereas if we use the dataset generated by the cDCGAN, the nearest neighbours have no mode.

Figure 7 shows that the loss fluctuates more within the cDCGAN training in comparison to the DCGAN training pro-

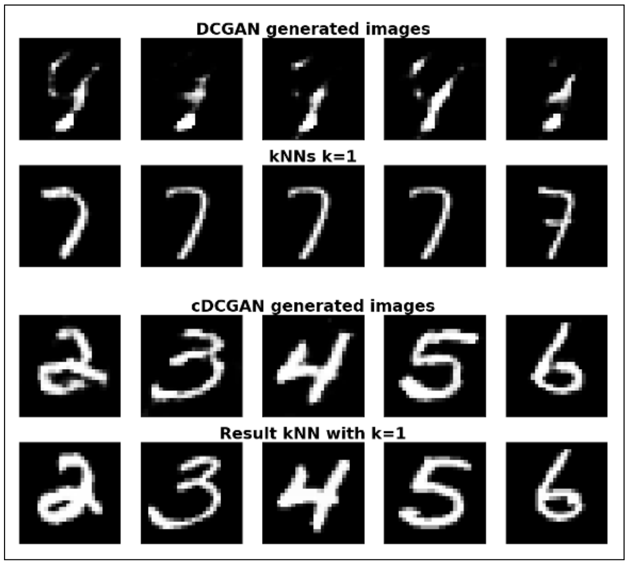

Fig. 6. The evaluation with the kNN shows that just one digit in the original dataset is close to the generated images.

cedure. At the same time, the loss curves from the cDCGAN decrease more within the final epochs. The GAN trained for 3.32 seconds per epoch on average, whereas the cDCGAN trained for 53.02 seconds per epoch on average. Note that the overall loss for cDCGAN is lower than cDCGAN. Secondly, it could be that with much longer training, CDCGAN could have similar loss, but with much longer training time. Furthermore, the loss of the generator and the discriminator decrease in the cDCGAN training procedure compared to the CGAN.

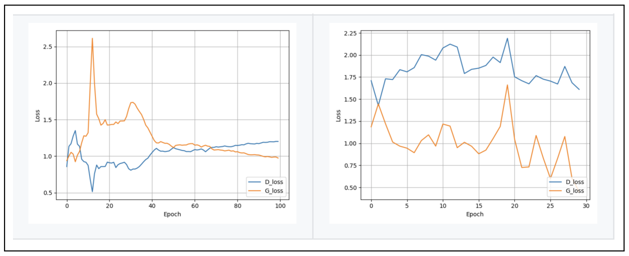

Fig. 7. The loss of $\boldsymbol{G}$ and $\boldsymbol{D}$ compared for a GAN (left) and a cDCGAN (right) architecture.

4. Discussion

As shown above, cGANs have performed remarkably well on the task of generating realistic images. However, as the complexity rises the results are getting surprisingly worse. For example, if the dataset contains photographs from objects on a conveyor belt, the generated images are not realistic at all. It is not even necessary to go towards visual examination because the results are so obvious. An example can be seen in Figure 8. Note two more dense layers and additional training time is added to ensure that the training process can be started.

Salimans et al. suggest an iterating process in order to find the most suitable values for the following hyperparameters: [2]:

- The discriminator's optimizer.

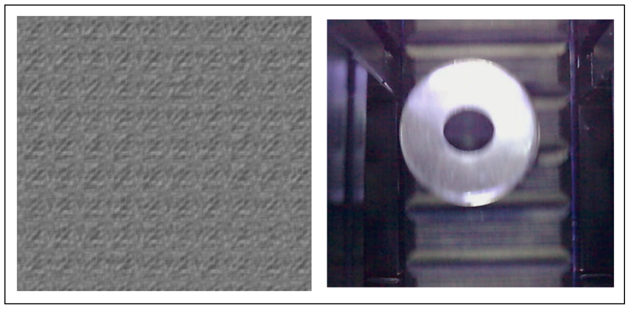

Fig. 8. The image shows the result (left) while using another dataset (right).

- The learning rate.
- Dropout probability.
- Batch size.

Therefore Goodfellow suggested to try different loss functions in order to significantly speed up the training process and thus to decrease the loss in a shorter amount of time [9].

Unrolled GANs are another method to prevent a GAN and a cDCGAN for mode s collapse [9]. It lowers the chance that the generator is overfitted for a specific discriminator by defining the generator objective with respect to an unrolled optimization of the discriminator.

Figure 8 shows that the performance of the cDCGAN can be strongly affected by the training set. All improvements mentioned above could help to improve the results of a future scenario. Within this scenario, we could try to use another dataset with larger images and the mentioned approaches to improve the training procedure of a DCGAN and cDCGAN.

5. Conclusion

Image generation conditioned on labels is a complex task that requires a model to learn not only the general data distribution for images, but also label embeddings for each individual class. It can be concluded from the results that well-designed cDCGANs can perform admirably well on this very complex task. Provided high-quality input data, these cGANs can be trained so that the combination of a conditional and the noise vector ($\boldsymbol{z}$) actually can generate images of handwritten digits. It is planned to expand this work in two directions. First, improving the training process by tuning the hyperparameters and secondly supporting any combination of training images and training labels.

References

[1] P. Isola, J. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," Tech. Rep., 2016. [Online]. Available: http://arxiv.org/abs/1611.07004

[2] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," Tech. Rep., 2016. [Online]. Available: http://arxiv.org/abs/1606.03498

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[4] 2020. [Online]. Available: https://www.tensorflow.org/tutorials/generative/dcgan?hl=en

[5] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," Tech. Rep., Jun. 2014. [Online]. Available: https://arxiv.org/abs/1406.2661

[6] M. Mirza and S. Osindero, "Conditional generative adversarial nets," Tech. Rep., 2014. [Online]. Available: http://arxiv.org/abs/1411.1784

[7] A. Borji, "Pros and cons of GAN evaluation measures," Tech. Rep., 2018. [Online]. Available: http://arxiv.org/abs/1802.03446

[8] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, G. Klambauer, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a nash equilibrium," Tech. Rep., 2017. [Online]. Available: http://arxiv.org/abs/1706.08500

[9] I. J. Goodfellow, "NIPS 2016 tutorial: Generative adversarial networks," *CoRR*, vol. abs/1701.00160, 2017. [Online]. Available: http://arxiv.org/abs/1701.00160