REGULARIZED GENERATIVE ADVERSARIAL NETWORK

GABRIELE DI CERBO, ALI HIRSA, AHMAD SHAYAAN

Abstract. We propose a framework for generating samples from a probability distribution that differs from the probability distribution of the training set. We use an adversarial process that simultaneously trains three networks, a generator and two discriminators. We refer to this new model as regularized generative adversarial network (RegGAN). We evaluate RegGAN on a synthetic dataset composed of gray scale images and we further show that it can be used to learn some pre-specified notions in topology (basic topology properties).

Contents

1. Introduction

In recent years, adversarial models have proven themselves to be extremely valuable in learning and generating samples from a given probability distribution. What is interesting about generative adversarial networks (`GANs`) is that they are capable of mimicking any non-parametric distribution. On the other hand, it is fairly common that we are interested in generating samples from a probability distribution that differs from the training set. We propose a method that allows us to use generative models to generate samples from a probability distribution even though we do not have samples of it in the training dataset. The key idea is to use a pre-trained network to drive the loss function in the learning process of a `GAN`.

Our main contributions are:

- We propose and evaluate a new architecture (`RegGAN`) for a generative adversarial network which is able to generate samples from a target distribution which does not appear in the training set.
- We show that these methods can be used as data augmentation technique to improve the performance of one of the discriminators.
- We extensively discuss how to use convolutional neural networks (`CNNs`) to learn discontinuous functions and use them in the loss function of a `GAN`, avoiding differentiability issues.
- We show that our model is able to learn basic topology properties of two dimensional sets.

2. Generative Adversarial Networks

Given training data

$$\text{training data} \sim p_{data}(x) \tag{1}$$

we wish to generate new samples from the same distribution. The goal is to find $p_{model}(x)$ in order to sample from it. In generative models we learn $p_{model}(x)$ similar to $p_{data}(x)$ which turns out to be a maximum likelihood problem

$$\theta^\star = \arg\max_\theta \mathbb{E}_{x\sim p_{data}} (\log p_{model}(x|\theta)) \tag{2}$$

Should be rephrased. The work in generative model sounds ambigious

The work in generative models can be categorized as follows (a) explicit density, (b) implicit density. In explicit density we assume some parametric form for density and utilize Markov techniques to be able to track distribution or update our distribution as more data is processed. `MCMC` techniques are an example. In implicit density case would not be possible to come up with a parametric from and we assume some non-parametric from that we are trying to learn.

`GANs` are designed to avoid using `Markov` chains because of high computational cost of `Markov` chains. Another advantage relative to `Boltzmann` machines is that the generator function has much fewer restrictions (there are only a few probability distributions that admit `Markov` chain sampling). Goodfellow et al. (2014) introduced it in a paper titled Generative Adversarial Networks. They are deep neural networks that contain two networks, competing with one another, that is where the name is coming from, used in unsupervised machine learning.

A new framework for estimating generative models through an adversarial process where they simultaneously train two models: a generative model that captures the data distribution and a discriminative model that estimates the probability that a sample came from the training data rather than the model being trained in (a).

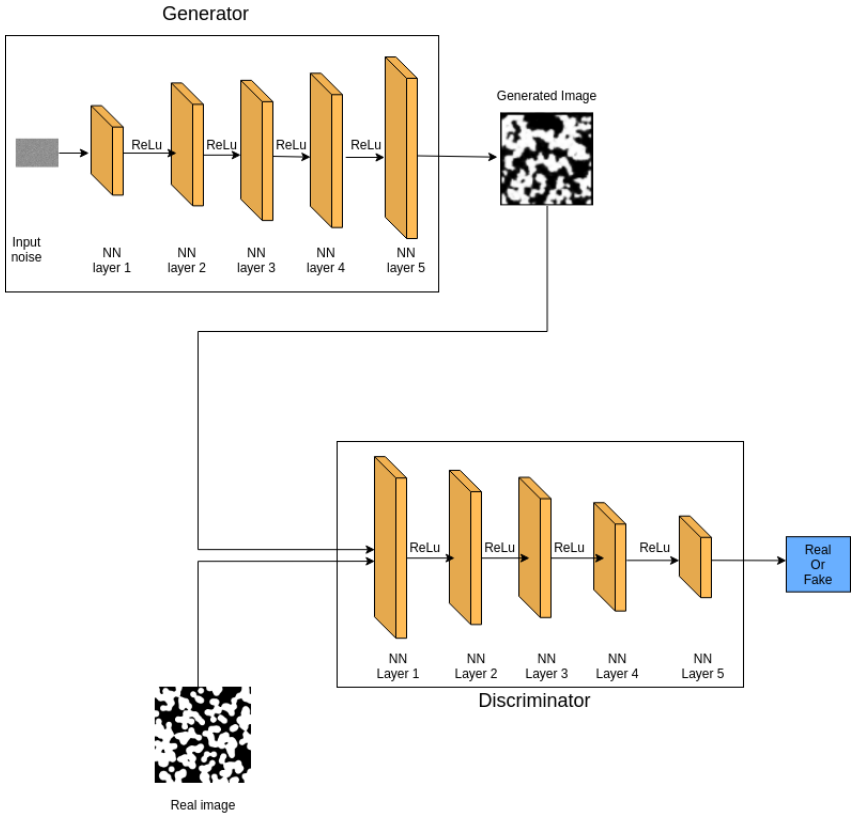


FIGURE 1. GAN architecture

The type of training in a GAN is set as min-max game (game theory) with the value function $V(G, D)$:

$$\min_G \max_D V(G, D) = \mathbb{E}_{x \sim P_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim P_z(z)} [\log(1 - D(G(z)))] \tag{3}$$

that means generator G tries harder to fool discriminator D and discriminator D becomes more and more cautious not getting fooled by the generator G

Shouldn't this be referenced. How did we com to this conclusion?

What makes GANs very interesting and appealing in that they can learn to copy and imitate any distribution of data. In the beginning GANs were used to improve images, make high-quality pictures and anime characters, nowadays they can be taught to create things amazingly similar to our surroundings. However, a vanilla GAN is simplistic and not able to learn the high-dimensional distribution especially in computer vision.

2.1. **Deep Convolutional Generative Adversarial Networks.** elaborate on evolution of GAN to DCGAN that is similar to feedforward neural net to convolutional neural net

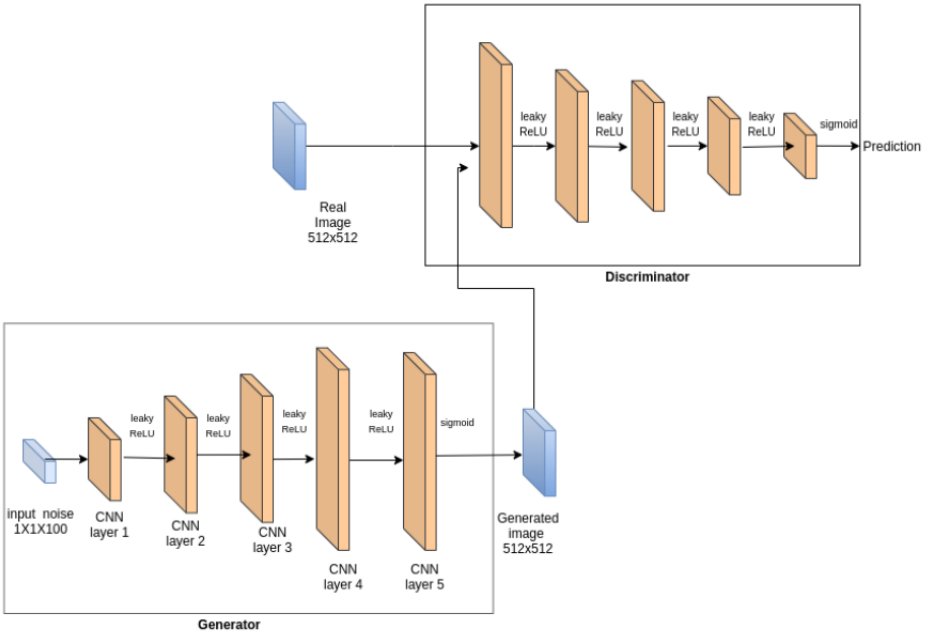


FIGURE 2. DCGAN architecture

Vanilla GANs are not capable of capturing the complexity of images and would be natural to introduce convolution networks into GANs, that is what is done in DCGAN. They bridge the gap between the success of CNNs for supervised learning and unsupervised learning in a GAN. They introduce a class of CNNs called deep convolutional generative adversarial networks (DCGANs), that have certain architectural constraints, and demonstrate that they are a strong candidate for unsupervised learning.

In our work, we first implemented a *Deep Convolutional Generative Adversarial* (DCGAN) network to generate the images with connected components. The architecture is shown in Figure 2. This network was able to generate images, however it did not capture the connected components in the image.

2.2. **Conditional Deep Convolutional Generative Adversarial Networks.** In general, if we are interested in any regularization, one typically add an explicit penalty function to the original loss function as follows

$$\widetilde{L}(\Theta) = L(\Theta) - \lambda \times score(\Theta) \tag{4}$$

where the score function measures certain characteristics about the object under consideration, for example the ratio of the biggest connected component to the entire area in an image. In learning, the explicit penalty does not work, the score function has to be incorporated into

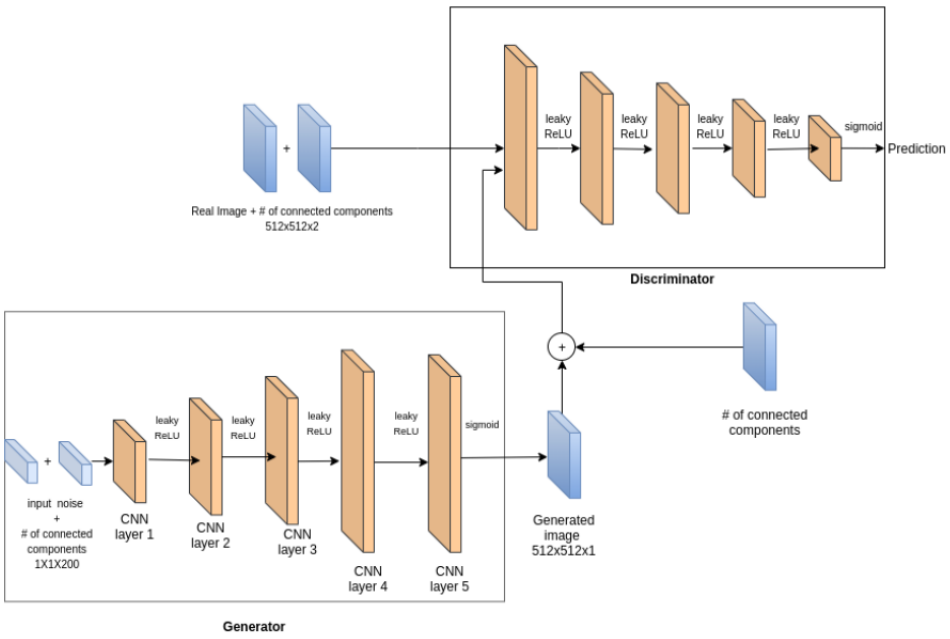


FIGURE 3. Conditional DCGAN architecture

learning. However the score function is not differentiable as will be shown shortly. The model will not able to learn and the best it can do is to generate black images to minimize the score.

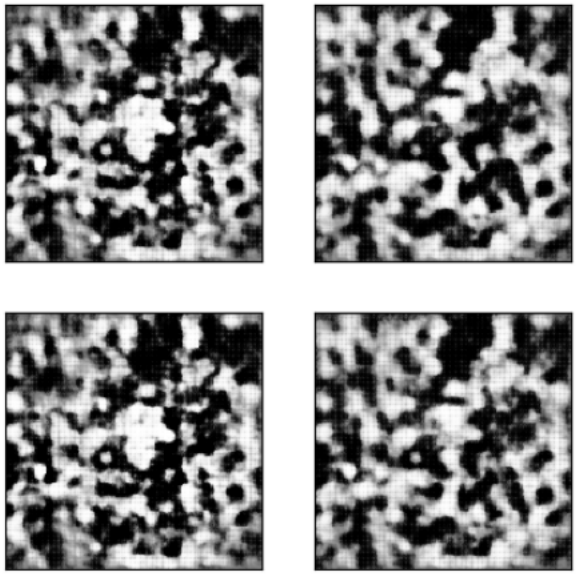
We tried to use a Conditional Deep Convolutional Generative Adversarial Networks (CDCGAN) to generate the images. CDCGAN are an extension of the vanilla DCGAN in which both the discriminator and the generator are conditioned on some extra information. The conditioning of the model on the extra information gives us control on the type of data that we want to generate. We condition the network on the number of connected components that we want in the generated image. The high level architecture of the CDCGAN is shown in figure 3.

CDCGAN did not work in our applications and for that reason we add the second discriminator to learn the score function and include it in the learning to be able to generate images with connected components. However the CDCGAN was not able to sufficiently capture the structures of the images and generated images as shown below.

3. Loss function in RegGAN

The new loss function is given by

$$\begin{aligned}(5)\quad &\min_{G}\max_{D_1,D_2} V(G,D_1,D_2)\\ &= \mathbb{E}_x\left[\log(D_1(x))\right] + \mathbb{E}_z \log(1-D_1(G(z))) + \mathbb{E}_z \log(1-D_2(G(z)))\end{aligned}$$



Epoch 22

We attempted to use Generative Adversarial Networks to generate images with a given number of connected components. We tried various different architectures, regularization schemes, and training paradigms to achieve the task. We also created a synthetic data set to train our network. We created collections of "blobs" ranging between 11-18 in number in every image

Why did we pre-train the score network? Could we have trained the entire model in one go?

4. Related work

There are a lot of papers doing similar things, we would need to provide a reasonable account of them here. I will add a few of them soon.

5. Dataset and topology

We use a synthetic dataset composed of 10k gray scale images. The images are generated by drawing a random number of pure black circles with a Gaussian blur of random intensity. This produces blob like pictures like the following:

For a given picture, we define the number of connected components of it to be the number of connected components of the region in the two dimensional space produced by the non white pixels with the topology defined by the Euclidean distance. For the purpose of our application, we are interested in generating images in the same style of the dataset

with only one connected components. On the other hand, our dataset has been generated in such a way that the images have a number of connected components between 15 and 20.

5.1. **Score function.** The number of connected components is a useful topological invariant of a region but it is not a very flexible invariant. For this reason, we define a function that measures how far a region is from being connected. Since images are presented as a collection of gray scale pixels, or equivalently a square matrix with entries between 0 and 1, the function below depends on the choice of a threshold α.

Let M be a $n \times n$ matrix with entries $0 \leq a_{ij} \leq 1$ and fix a real number $0 < \alpha < 1$. Let $\bar{M}$ be the matrix with entry $\bar{a}_{ij}$ defined by the following

$$\bar{a}_{ij} = \begin{cases} 1 & \text{if } a_{ij} \geq \alpha, \\ 0 & \text{if } a_{ij} < \alpha. \end{cases}$$

Let M_o be the largest connected component of $\bar{M}$. Here we defined a connected component to be the matrix composed by all entries with value 1 that share some common side with at least one other element of the same component. The largest connected component is the one that contains the largest number of 1's. Note that it is not necessarily unique. If we represents pixels as squares of fixed side length in the Euclidean space, M_o corresponds to the largest connected component of the region defined by the pixels with value 1 under the Euclidean topology.

For a given $n \times x$ matrix $M = (a_{ij})$ we define $\|M\| = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}$. For a matrix with entries 0 or 1, it corresponds to the usual Euclidean norm and it computes the area of the region defined by the pixels with value 1.

We are now ready to define the score function $s : \mathbb{R}^{n \times n} \to \mathbb{R}$ as

$$s(M) = \frac{\|M_o\|}{\|\bar{M}\|}.$$

Note that $0 \leq s(M) \leq 1$ and $s(M) = 1$ if and only if M has a unique connected component. The above definition depends on a choice of α and for the rest of this paper we will assume that $\alpha = 0.5$.

One of the main technical problems encounter in this paper is the fact that s is not a continuous function. It is easier to imagine the behavior of the function s acting on regions of the plane. If our region is composed by two disconnected disks of equal area then s has value 0.5 there. On the other hand, if we let the disks come closer and closer, s will have constant value 0.5 until the disks touch and s will jump to the value 1.

Since s is not a differentiable function, it cannot be used in combination with gradient descent while training our model.

To overcome this problem we use a convolutional neural network (CNN) to learn the score function. As noted before, s is not a differentiable function and a CNN will not perform well if we just try to learn the function s. The main idea here is to bin together images in the dataset with similar score function. More precisely, we create 11 labels corresponding to the values obtained by applying $.round()$ to $10s(M)$. For example, as we are working with torch tensors, $.round()$ return 0 for all values between 0 and 0.499 and 1 for all values between 0.5 and 1.499.

6. RegGAN

6.1. **Model architecture.** The RegGAN[1] architecture consists of two discriminator and a single generator. The second classifier is used to simulate the score function, which was designed by us. The first discriminator is used to differentiate between the images generated by the network and the ones from the dataset. The dataset is composed by images of size 64×64 which will determine the number of convolutional layers of the networks. The architecture in shown in Figure 4.

6.2. **Classifier.** This network is composed by 4 convolutional layers, 2 max pool layers and 3 linear layers. We pre-trained it on the dataset as a classifier of the images where the labels are assigned by the score function s as explained before. We use cross entropy loss to train it. Around 15k iterations we get close to 80% accuracy.

We pre-train this network to learn the score function. This is done so that the second discriminator has a good starting point for the actual training of the network.

During pre-training we feed in the images from the data set to the network. The outputs from the network are then compared to the actual scores given by the score function.

We then save the trained network and load it when we train the generator.

6.3. **Discriminator.** This network is composed by 5 convolutional layers. We trained against the generator using BCEWithLogicLoss[] which combines a sigmoid layer to a criterion that measures the binary

[1]number of publications on deep learning applications is enormous. Considering that the initial aim of this study was to able to produce artist patterns with connected components. We first thought to call it ArtGAN, but recognized the name is taken. Considering that our architecture has two discriminators would have been natural to call it D2GAN or DDGAN, but those two names were taken as well. We thought of YAGAN which stands for Yet Another GAN based on YACC, Yet Another Compiler Compiler, but thought it would not reflect the nature of the proposed architecture. Knowing that the second dicriminator plays the role of a regularizer implicitly we name it RegGAN for regularized GAN.

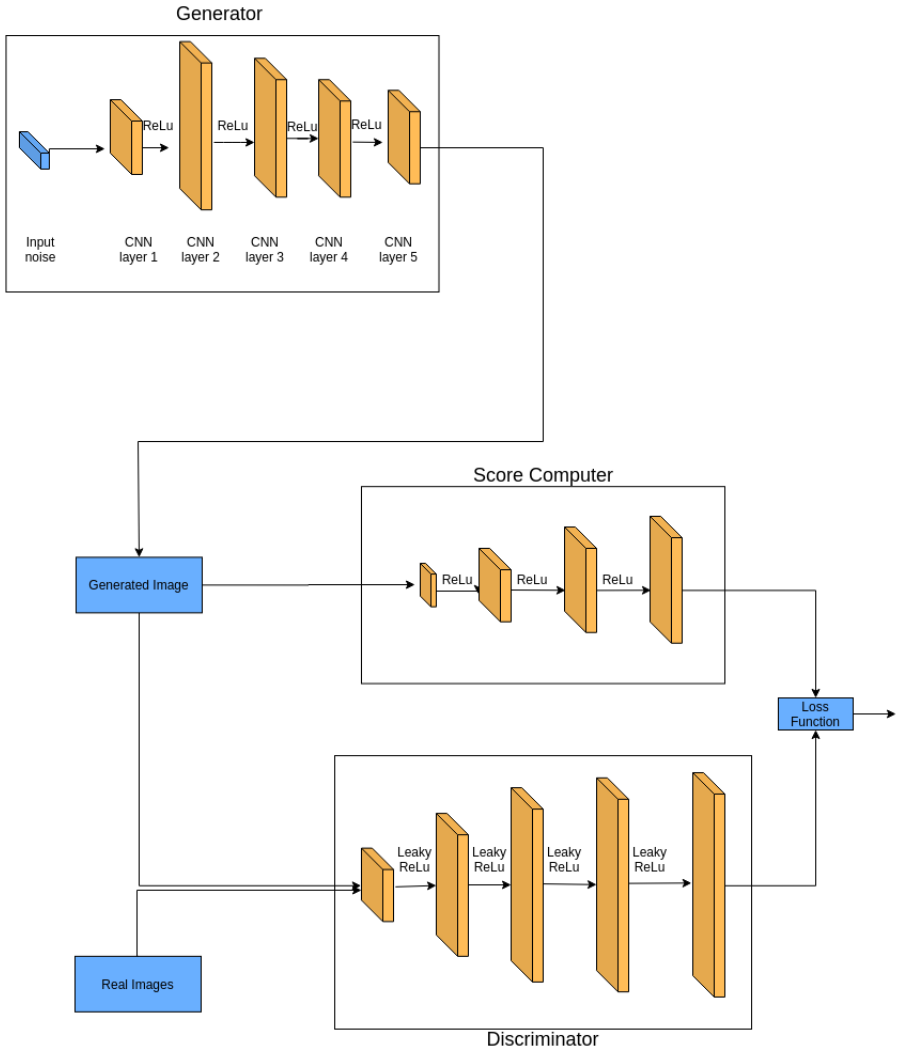


FIGURE 4. RegGAN architecture

cross entropy in a single class. In various experiments, it proves to be more numerically stable than binary cross entropy (BCE).

6.4. **Generator.** Similarly to the discriminator the generator is composed by 5 convolutional layers. We train the generator in two steps during each epoch: first we train it against the discriminator in the usual way we train a DCGAN. Then we train again against the classifier. We train the generator to maximize the value of the classifier on the generated images. This pushes the score function of the generated images to converge to 1, which forces the production of only images with a single connected component.

We feed in noise to the generator and get outputs as images. These images are then fed into both the discriminators to compute the score and to compare it to the images of the actual data set.

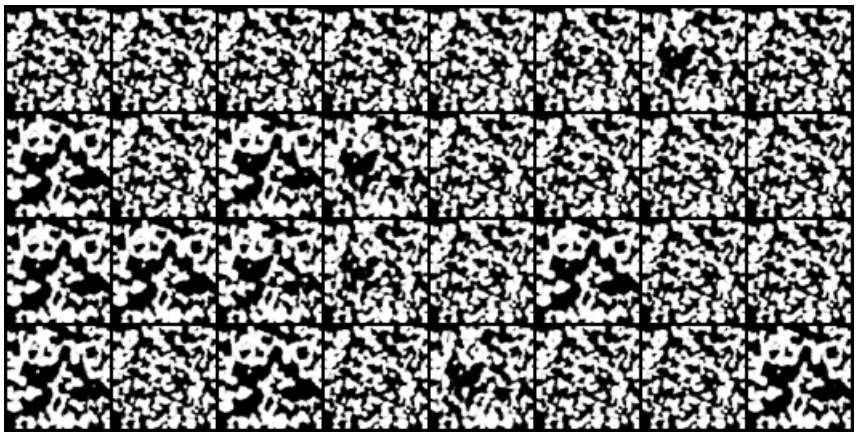


FIGURE 5. sample images generated by RegGAN

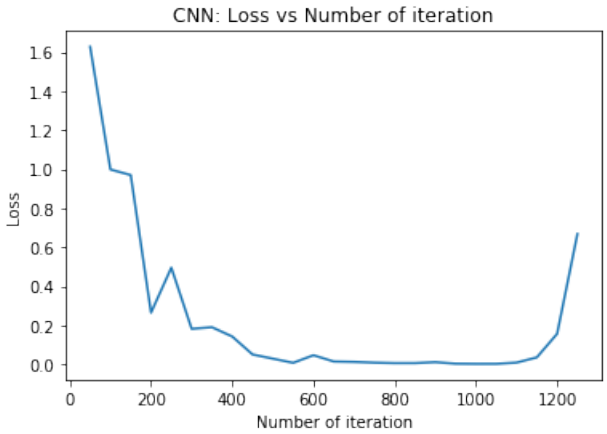


FIGURE 6. Results of regGAN

We have two ways in which we back-propagate. In the first one we freeze the weights of the second discriminator and the gradient is only propagated through the the generator and the first discriminator. In the second method we pass the gradient through the second discriminator as well.

A sample of the images generated by the network can be seen in Figure 5.

The outputs from the network are then compared to the actual scores given by the score function. The results from the training of the network are shown in Figure 6.

7. Empirical validation

We compare our results against a DCGAN to show that our architecture is able to generate images with very high score and a low number of connected components.

We trained a DCGAN and RegGAN on the same dataset with the same number of iterations. Here are the results compared:

8. Future work

Maybe something in 3D? We could do something similar with 3D images and try to apply to medical research. Also, topology in 3D is much harder so it should be interesting.

References

[GAN14] Goodfellow, Ian J., Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron C., and Bengio, Yoshua. Generative adversarial nets. NIPS, 2014.

[DCGAN16] Radford, Alec, Metz, Luke, and Chintala, Soumith Uusupervised representation learning with deep convolutional generative adversarial networks. ICLR 2016

[ArtGAN17] ArtGAN: Wei Ren Tan, Chee Seng Chan, Hernan E. Aguirre, Kiyoshi Tanaka. Artwork Synthesis with Conditional Categorical GANs. https://arxiv.org/pdf/1702.03410.pdf 2017

[D2GAN17] Tu Dinh Nguyen, Trung Le, Hung Vu, Dinh Phung. Dual Discriminator Generative Adversarial Nets. https://arxiv.org/pdf/1709.03831.pdf 2017

Department of Mathematics, Princeton University, Princeton NJ 08540, USA

E-mail address: dicerbo@math.princeton.edu

Industrial Engineering and Operation Research, Columbia University, New York NY 10027, USA

E-mail address: ali.hirsa@columbia.edu

Industrial Engineering and Operation Research, Columbia University, New York NY 10027, USA

E-mail address: ahmad.shayaan@columbia.edu