Laboratory Report - GANs

Course: SGN-26006 Advanced Signal Processing

Assignment no. 7: GANs

Authors: Adam Ligocki, Amir Salah

Released: 11th of December 2018

Table of Contents

1. Task

Implement the vanilla and deep convolutional Generative Adversarial Network (GAN) using predefined TensorFlow based templates. Compare the pros and cons of these techniques.

2. Solution

We have created 3 different solutions. One is for vanilla GAN based on fully connected neural network layers. The second implementation is using deep convolutional GAN (DCGAN) techniques. These two implementations are based on TensorFlow framework. The third implementation is using Keras framework with TensorFlow Backend and also implements DCGAN scheme.

Architecture

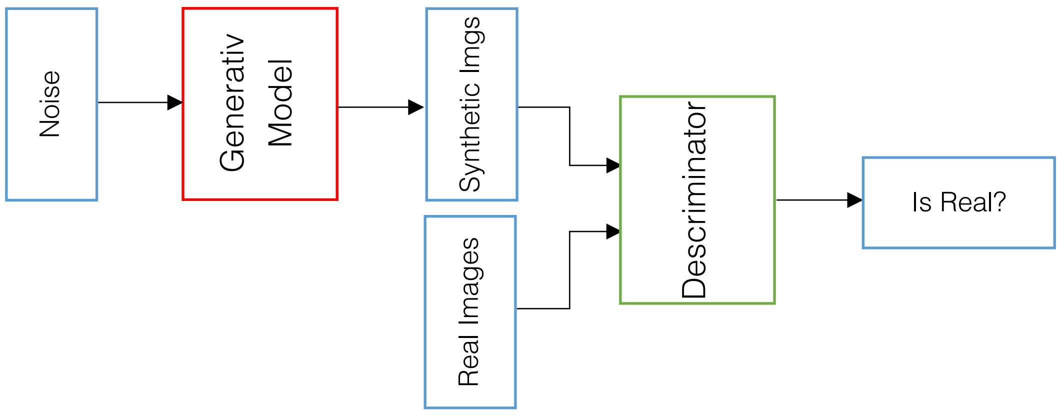

Figure 1 - General scheme of Generative Adversarial Network.

Figure 1 shows basic idea of GAN. The generative model maps input noise vector into an output image of the same size as the real images. The discriminator job here is to distinguish whether the image is real or synthetic.

In this process, the generator network and the discriminator compete against each other. The generator tries to fools the discriminator and discriminator is rewarded for correctly labeling valid or synthetically generated images. Because of this competition, both models have to have approximately the same complexity of architecture. In other case the more sophisticated the model gets, the better the opposing network learns. At some point in time though, the equilibrium state of the learning process will be reached and the less sophisticated model stops improving.

Vanilla GAN [1] neural networks have been designed as a two fully connected layers for both models. The hidden layer contains 128 neurons for both models. The generator input noise is a vector of 100 uniformly generated numbers in range <-1,1>.

TensorFlow based DCGAN [2] generative model begins by one fully connected layer followed by two 2D transpose convolutional layers with batch normalization and leaky Relu activation function. After that the dropout with probability of 0.5 is inserted and entire model is finished by single kernel 2D transpose convolutional layer with Tanh activation function. The discriminator is common convolutional network with 3 convolutional layers with leaky Relu function on output and batch normalization. In this case max pooling layers are not used. In step of this, every convolutional layer has kernel of the size of 5 and strides parameter equals to 2 which helps to reduce size of following layers.

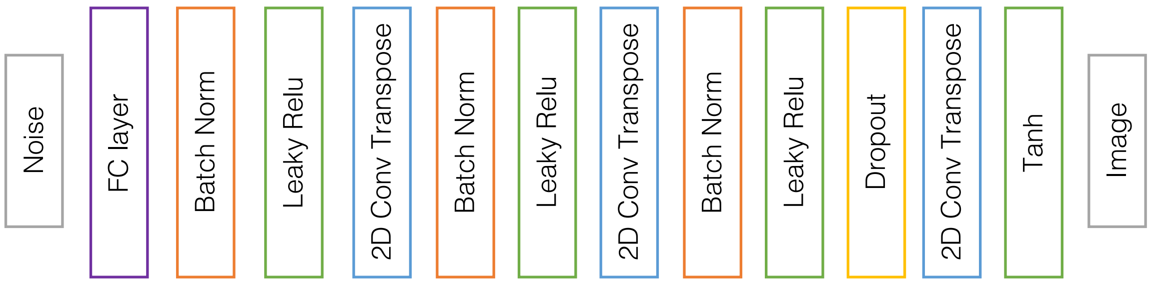

Figure 2 - Architecture of TensorFlow based DCGAN generator implementation.

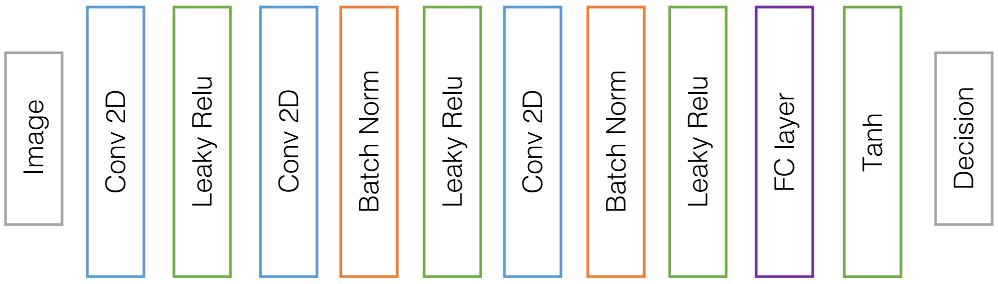

Figure 3 - Architecture of TensorFlow based DCGAN discriminator implementation.

Keras base DCGAN is designed similarly to previous DCGAN scheme. The generator maps 100 length input noise vector into synthetic images by fully connected input layer followed by two cells where each contains two 2D convolutional layers with dropout and leaky Relu activation function and one up sampling layer. Finally flatted into single fully connected layer.

Dataset

For learning we have used the MNIST dataset, which contains 60 000 labeled images of hand written digits. Every image is single channel and has size of 28x28px.

Figure 4 - Example of MNIST images

All the images are normalized into <-1,1> range [3].

Learning

The learning process is divided into two phases. The first one is the discriminator learning. The generator generates batch of images labeled as a synthetical ones and this batch is expanded by real images labeled so. This set of images is used for single run training of the discriminator to recognize these two categories of input data. During this time the weights of generator are frozen.

In the second phase the generator learns to generate the images that will fool the discriminator. Generator gets reward for doing so. In this phase the weights of discriminator are frozen.

These two phases keeps on repeating until the generator is able to create images of required qualities.

3. Results

Vanilla GAN

The output of this GAN are relatively well generated images of all types of digits. The disadvantage of this technique is that for fully connected layers of neural network

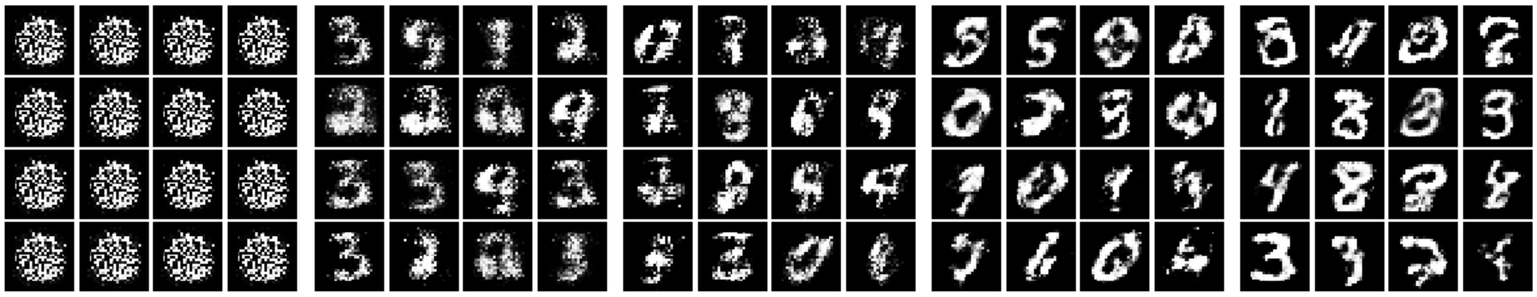

Figure 5 - Results of learning phase after 1, 4000, 10000, 20000 and 100000 epochs

there are no space-geometrical constrains, which means that each neuron has the same influence on all inputs of next layer respectively all pixels of output image. This cause noise, we can see on the output images.

DCGAN

The results of this implementation bring images with better noise quality. The shape of digits is approximately the same but big disadvantage is that the GAN tends to stack on generating single digit for all input noise variants.

To avoid this effect some techniques of augmentations could be used, or we could try to balance the ration of all digit types in learning dataset.

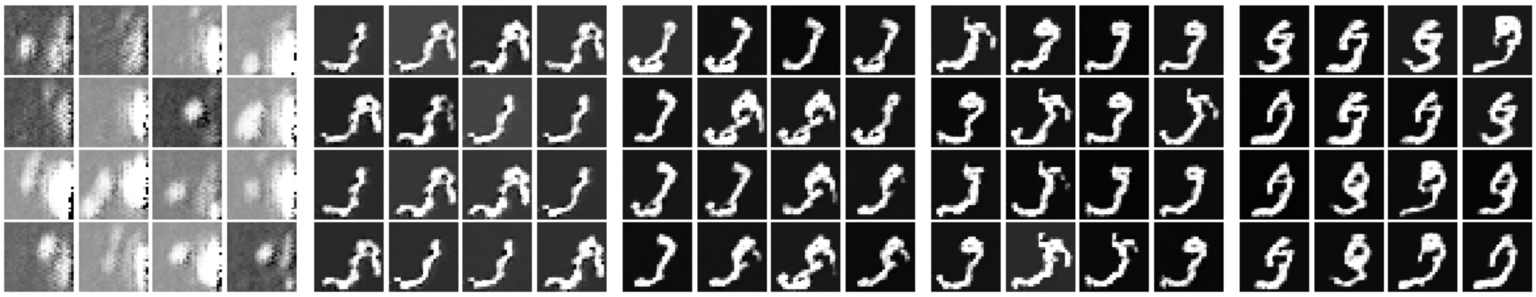

Figure 6 - Results of learning phase after 1, 4000, 10000, 20000 and 100000 epochs

DCGAN (Keras)

Best results have been reached by our own implementation of 5-layer generator-discriminator GAN scheme. The synthetical output images are noise less and also they are geometrically consistent. Also, all the digits are distributed equally.

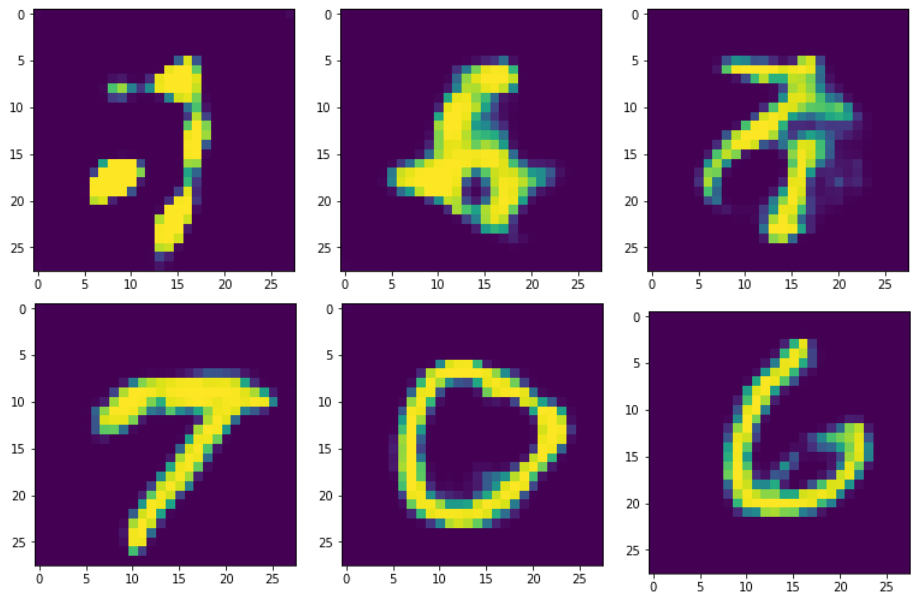

Figure 7 - Learning results after 1, 5,10, 20, 30 and 50 epochs.

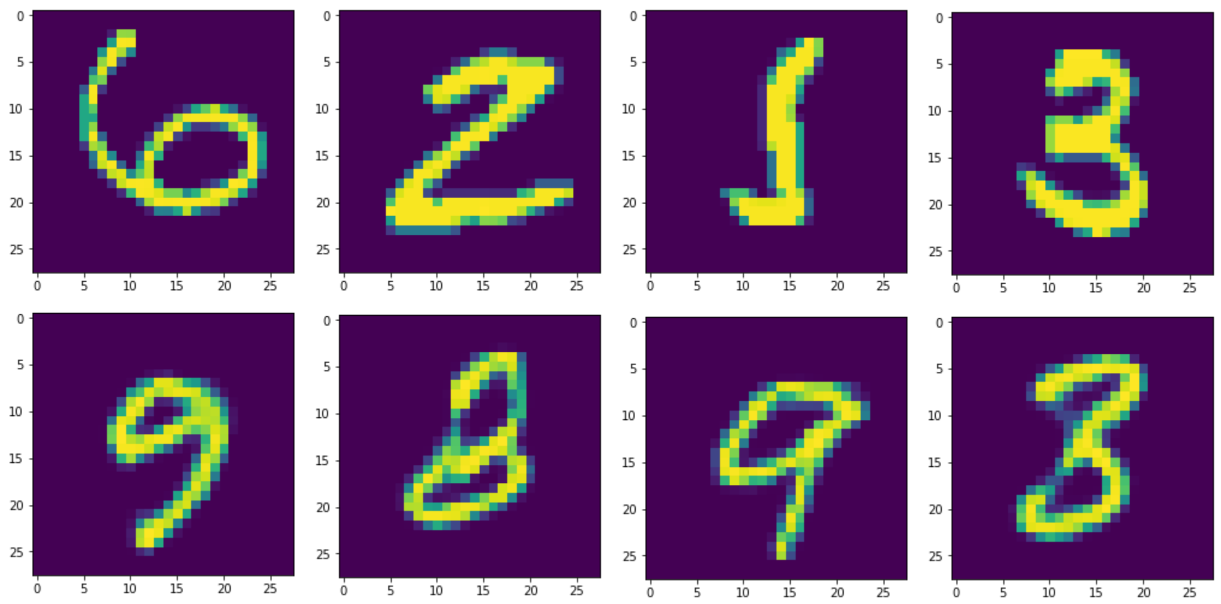

Figure 8 – Comparison of real and synthetic images. We are leaving these images unlabeled as a puzzle for the reader. One row is handwritten, and the other is generated by GAN network.

4. Conclusion

We implemented 3 different solutions by taking advantage of GANs properties. Single vanilla GAN implementation with fully connected neural networks, and two deep CNN based models.

Two of these implementations has been pretrained prior to the implementation and one of them was built entirely from scratch.

The results of vanilla GAN looked humanely readable. GAN represents wide range of different digits. However, the output images have high amount of noise.

The first DCGAN implementation generated clearer images but the implementation was limited to generating only single digit.

The best result has been achieved while using the self-implemented DCGAN as displayed on Figure 8.

5. References

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. In Int. Conf. on Neural Information Processing Systems (NIPS), proceedings, 2014.

[2] A. Radford, L. Metz, and S. Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In Int. Conf. on Learning Representa- tions (ICLR), proceedings, 2016.

[3] How to Train a GAN? Tips and tricks to make GANs work. *GitHub* [online]. [cit. 2018-12-11]. Dostupné z: https://github.com/soumith/ganhacks