CLEMSON UNIVERSITY

S2101-CPSC-8430 Deep Learning - 001 - 20696

HOMEWORK-3

Generator/Discriminator Training using DCGAN, WGAN techniques

Submitted By:

Manish Enishetty

To:

Dr. Feng Luo

Abstract

This Report explains the performance comparison between DCGAN and WGAN implemented on CIFAR-10 dataset.

Problem Statement

Train a discriminator/generator pair on CIFAR10 dataset utilizing techniques from DCGAN and Wasserstein GANs

Introduction

GAN (Generative Adversarial Network) are algorithmic architectures that use two networks opposing against each other to generate synthetic new instances of data that will be similar like the real data. The generative network creates new data while the discriminative network will evaluate the data. Here, the generative network adapts to map from the latent space to the data distribution of interest, on the other hand, the discriminative network differentiates the data produced by the generative network from the true data distribution. The goal of the generative network is to generate novel data such that the discriminative network cannot determine between the data generated by the generative network and the true data of the dataset.

1. Requirements:

    A. CIFAR-10 Dataset
    B. DCGAN Model
    C. WGAN model
    D. Python 3, PyTorch.

A) Dataset: CIFAR-10

The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images.

The classes in the CIFAR-10 dataset are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck.

B) *DCGAN: Deep Convolutional Generative Adversarial Network*

DCGANs composes of mainly convolution layers without max pooling and fully connected layers. It uses transposed convolution and convolutional stride for up-sampling and down-sampling of the images. As DCGANs are hard to train, so there are architecture guidelines for stable training of the network.

- Use convolutional stride instead of max pooling
- For up-sampling use transposed convolution
- Eliminate fully connected layers.
- Use Batch normalization for all the layers except the input layer of the discriminator and the output layer for the generator.
- Use Leaky-ReLU in the generator except for the output layer where tanh activation function is used.
- Use Leaky-ReLU in the discriminator except for the output layer where sigmoid activation function is used.

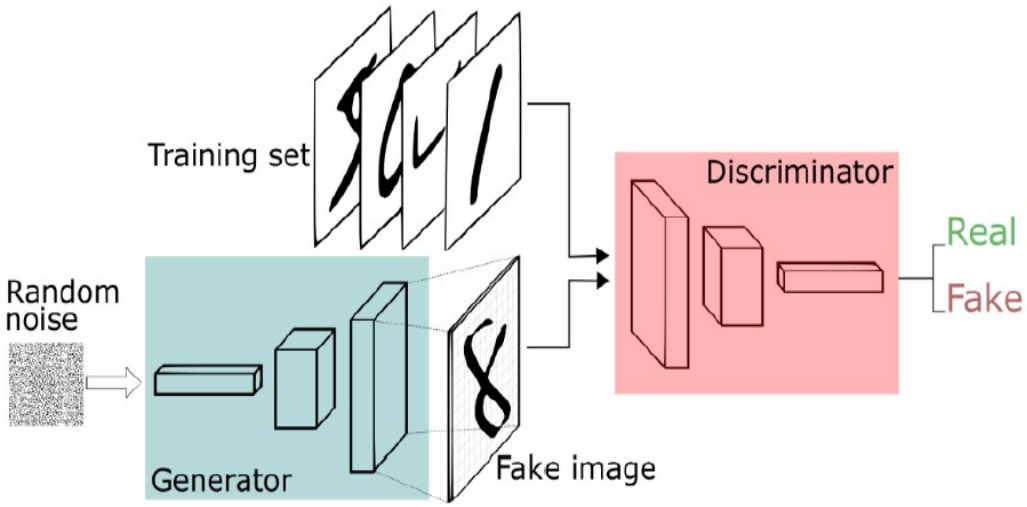


Figure 1: The process on images in the form of Generator and Discriminator

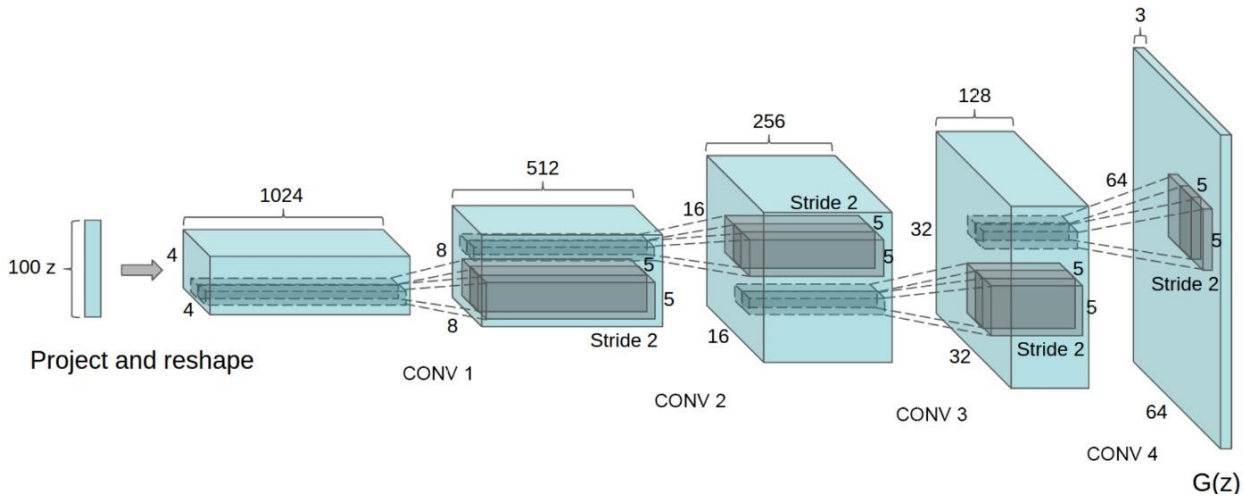


Figure 2: DCGAN generator working in a CNN

C) WGAN: Wasserstein Generative Adversarial Network

The Wasserstein GAN expands into the generative network, both improving reliability during model training and having a loss feature that is related to image quality.

- The development of the WGAN has a strong mathematical motivation, although only a few minor changes to the standard, deep convolutionary, generative opponent network or DCGAN require in practice.
- Instead of sigmoid, using the linear activation feature in the essential model's output sheet.
- Using -1 for actual pictures and 1 for bogus pictures (instead of 1 and 0).
- Train the vital and generator models with Wasserstein depletion. Limit vital product weight for any mini batch upgrade to a small range (e.g. [0.01,0.01]).
- Check the essential model per iteration more than the generator (e.g. 5). Using the variant of RMSProp with low learning and no momentum (for example 0.00005).

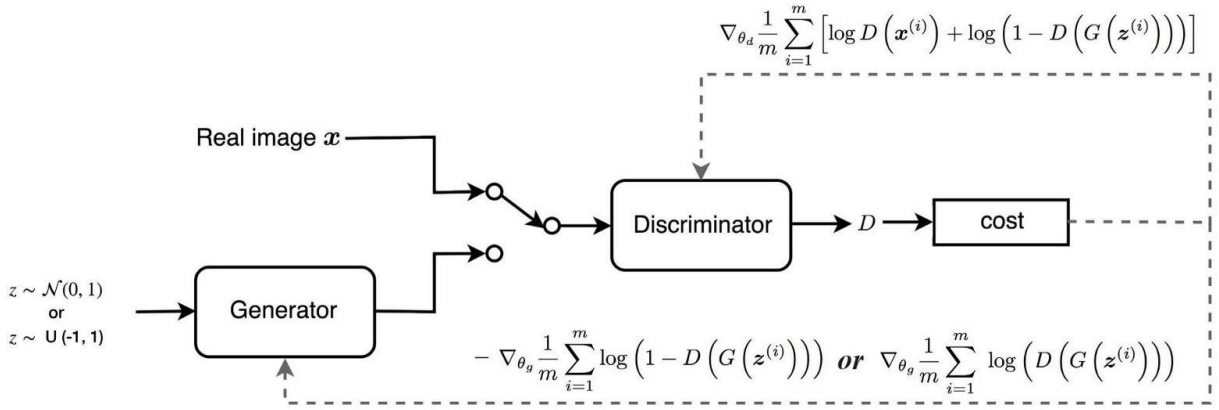


Figure 3: Working of WGAN with loss functions and Generator/Discriminator model

3. Implementation

3.1 Discriminator model

The Discriminator Model is first described. In order to insert data and determine if the sample is real or false, the model will take an example picture. It transforms into a conditional question of classification. The configuration of the discriminator begins with a regular convolution layer and instead with 2x2 phases are introduced three convolutionary layers for the input picture downsample. In the final layer, the model has the sigmoid activation feature to determine whether the picture is valid or false and consists of no pooling. The loss function is a discrete entropy loss function that is ideal for discrete classification. Here is the Discriminator Model Description.

3.2 Generator model

The generator paradigm generates incorrect images with possible images with artifacts in limited dimensions. This is achieved by pointing a square color picture from the latent domain. The concept of the generator assigns significance to the latent space and the latent space is a compact reflection of the output space. This is achieved by making a dense layer like the first secret layer with appropriate nodes to reflect a picture with low resolution. We do not need only one image version in low resolution, but many parallel interpretations.

This is a trend in CNN with several concurrent filters, contributing to many simultaneous activation maps known as input function maps. The next move is to update the low-resolution picture to a higher-resolution image version. It is used to quadric the field in the input maps of the Conv2DTranspose row. Two more cycles to hit the performance image of 32x32.

4. Performance and Results (FID Score)

4.1 DCGAN vs. WGAN

DCGAN is more concerned with improvements to the network design, while WGAN is the loss feature. You can't interrupt the DCGAN architecture with the WGAN objective function: it all decreases the estimated failure of Wasserstein instead of the separation of Jensen Shannon with a specific network architecture. The WGAN (or its follow ups, for example WGAN-GP), is architecturally agnostic. The only aspect (which I know of) you will look after is to use the batch standard; DCGAN advises to bring it all over, however (for WGAN-GP at least) it disorganizes up with vital regularization figures.

4.2 FID Score

FID scores for 50,000 samples of CIFAR-10 which were trained until epoch 100.

Score	WGAN	DCGAN	ACGAN
FID	15.9	42.38	12.7

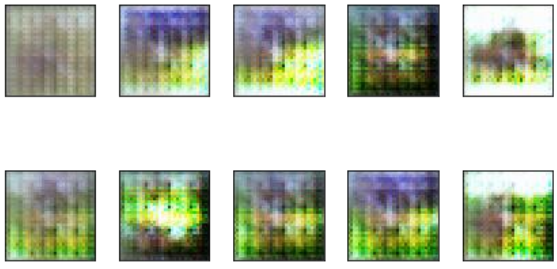
This is the output for DC GAN :

```
Epoch [99/100] Batch 100/391                    Loss D: 0.1676, loss G: 3.7033
Epoch [99/100] Batch 200/391                    Loss D: 0.1555, loss G: 4.3193
Epoch [99/100] Batch 300/391                    Loss D: 0.1487, loss G: 3.8571
(pytorch_env) [menishe@node0006 dataset]$ python -m pytorch_fid fake_images/ real_images/
FID:   42.3889689580584
```

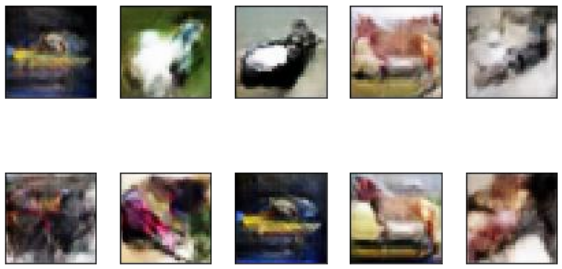
5. Output Images

5.1WGAN

Epoch 0

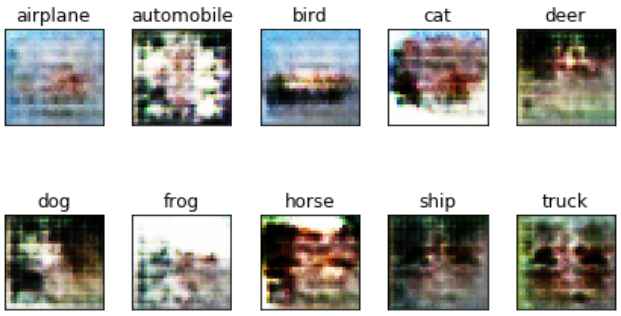


Epoch 100



5.2 DCGAN

Epoch 0



Epoch 100

Results after 100 EPOCHS(10 best pics)

Bonus Part- ACGAN

The Auxiliary Classifier GAN is an extension of the conditional GAN that changes the discriminator to predict the class label of a given image rather than receive it as input. It has the influence of stabilizing the training process and allowing the generation of large high-quality images whereas learning a representation in the latent space that is independent of the class label.

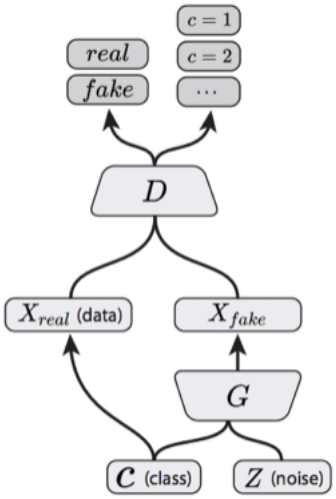


Fig: ACGAN Model

ACGAN was introduced by Augustus Odenl, from Google Brain in the 2016 paper titled "Conditional Image Synthesis with Auxiliary Classifier GANs."

As with the conditional GAN, the generator model in the AC-GAN is provided both with a point in the latent space and the class label as input, e.g. the image generation process is conditional.

The main difference is in the discriminator model is unlike the conditional GAN that is provided with the image and class label as input. The discriminator model must then predict whether the given image is real or fake as before and must also predict the class label of the image.

The architecture is described in such a way that the discriminator and auxiliary classifier may be considered separate models that share model weights. In practice, the discriminator and auxiliary classifier can be implemented as a single neural network model with two outputs.

The first output is a single probability via the sigmoid activation function that indicates the "*realness*" of the input image and is optimized using binary cross entropy like a normal GAN discriminator model.

The second output is a probability of the image belonging to each class via the softmax activation function, like any given multi-class classification neural network model, and is optimized using categorical cross entropy.

Results for ACGAN Model:

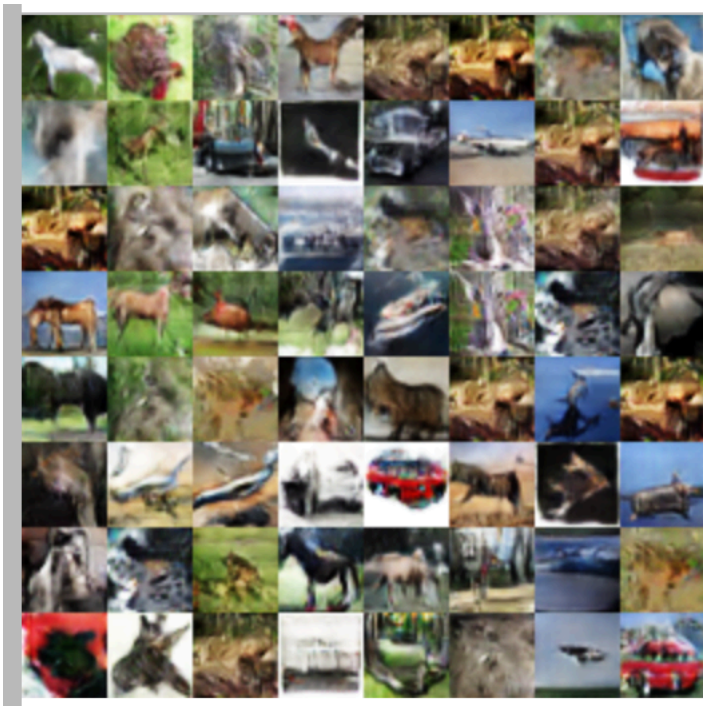


Fig: ACGAN output after 50 EPOCHS

menishe(× | ACGAN.p × | main.py × | menishe(× | ACGAN_t × | ACGAN_t × | ACGAN_t × | utils.py × | inception × | m

```
Epoch: [41] [  81/ 937] time: 1222.1115, d_loss: 0.50603181, g_loss: 6.35006809, q_loss: 0.22413442
Epoch: [41] [ 581/ 937] time: 1236.6612, d_loss: 0.65400803, g_loss: 3.22593594, q_loss: 0.30092230
Epoch: [42] [ 144/ 937] time: 1251.2234, d_loss: 0.45096850, g_loss: 4.73541927, q_loss: 0.29326567
Epoch: [42] [ 644/ 937] time: 1265.7205, d_loss: 0.78180957, g_loss: 6.07752132, q_loss: 0.20944621
Epoch: [43] [ 207/ 937] time: 1280.1930, d_loss: 0.50695652, g_loss: 3.59898233, q_loss: 0.24441463
Epoch: [43] [ 707/ 937] time: 1294.6402, d_loss: 0.43134302, g_loss: 3.80162334, q_loss: 0.33846071
Epoch: [44] [ 270/ 937] time: 1309.1671, d_loss: 0.21215165, g_loss: 4.87053347, q_loss: 0.17149404
Epoch: [44] [ 770/ 937] time: 1323.7914, d_loss: 0.39773554, g_loss: 3.89454818, q_loss: 0.18642852
Epoch: [45] [ 333/ 937] time: 1338.3266, d_loss: 0.46134466, g_loss: 3.89621067, q_loss: 0.15290976
Epoch: [45] [ 833/ 937] time: 1352.8885, d_loss: 0.84376895, g_loss: 2.58300805, q_loss: 0.31694523

 4.856570065238843 0.14652305195384216

Epoch: [46] [ 396/ 937] time: 1377.5850, d_loss: 0.26131558, g_loss: 4.06837273, q_loss: 0.35754368
Epoch: [46] [ 896/ 937] time: 1392.1384, d_loss: 0.61720210, g_loss: 6.68030882, q_loss: 0.11361665
Epoch: [47] [ 459/ 937] time: 1406.7361, d_loss: 0.26957494, g_loss: 4.89649725, q_loss: 0.13679664
Epoch: [48] [  22/ 937] time: 1421.3133, d_loss: 0.22192979, g_loss: 5.45210838, q_loss: 0.35978937
Epoch: [48] [ 522/ 937] time: 1435.8459, d_loss: 0.35978782, g_loss: 3.59306383, q_loss: 0.24984702
Epoch: [49] [  85/ 937] time: 1450.3563, d_loss: 0.97547185, g_loss: 9.40154839, q_loss: 0.26081944
Epoch: [49] [ 585/ 937] time: 1464.9869, d_loss: 0.58408451, g_loss: 7.81673670, q_loss: 0.12507066
Epoch: [50] [ 148/ 937] time: 1479.5664, d_loss: 0.15631814, g_loss: 4.86164856, q_loss: 0.20774125
Epoch: [50] [ 648/ 937] time: 1494.1793, d_loss: 0.27106148, g_loss: 6.24809170, q_loss: 0.15814511

 5.08803151190708 0.11451172737143163

Epoch: [51] [ 211/ 937] time: 1518.7415, d_loss: 0.18433848, g_loss: 4.73287868, q_loss: 0.23943068
Epoch: [51] [ 711/ 937] time: 1533.3397, d_loss: 0.57527798, g_loss: 7.42532158, q_loss: 0.28302139
Epoch: [52] [ 274/ 937] time: 1547.9455, d_loss: 0.15609622, g_loss: 5.48298168, q_loss: 0.28742716
```