CPSC 8810: Deep Learning

Homework 3 Report

Namita Vagdevi Cherukuru – C32671672 (ncheruk@clemson.edu)

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

The basic idea of training a discriminator or the generator pair is to determine the best GAN models to generate best images in CIFAR-10 dataset. GANs are a kind of neural networks composed of 2 independent, neural networks, which are consistent with generators (G) and discriminators (D). DCGAN is the basic GAN variants of Deep Convolutionary GANs, where G and D are based on a deep neural network. WGANs are complementary to turns and help the simulation of multi-level long-range dependencies in the image dataset regions.

1. Requirements:

- Python 3
- Tensorflow 1.15
- CIFAR-10 Dataset
- DCGAN and WGAN models

2. GANs

- One of the GAN neural networks, called the generator, produces new datapoint information, and the other is an accuracy checked by the discriminator.
- Discriminator, in other terms makes the determination whether or not any instance of knowledge that is analyzed belongs to the specific training dataset.
- The generated image and details from the real ground reality dataset are applied to the discriminator.

- The discriminator takes all true and false pictures and returns odds of 0 or 1 with 1.
- The discriminator is with the fundamental reality of the images in the feedback loop.

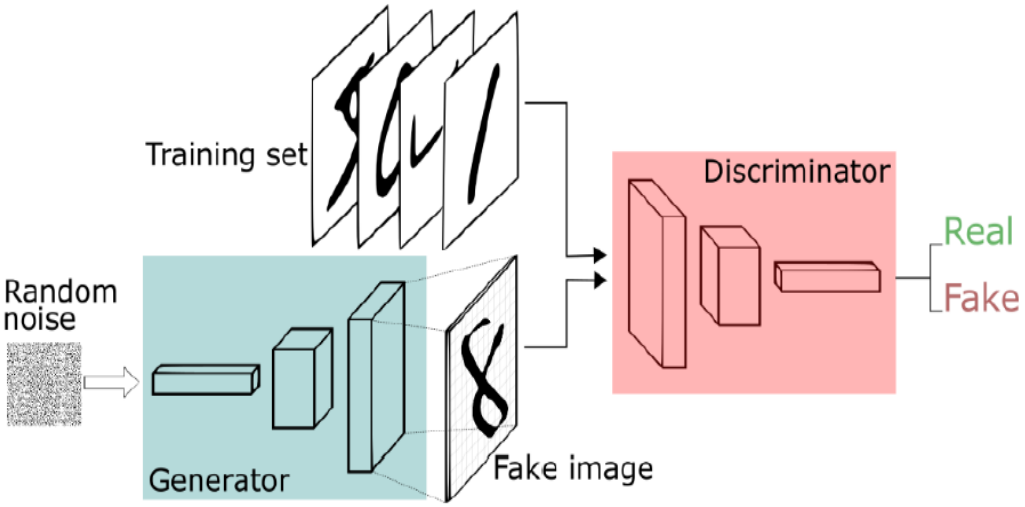


Figure 1: The process on images in the form of Generator and Discriminator

2.1 DCGAN

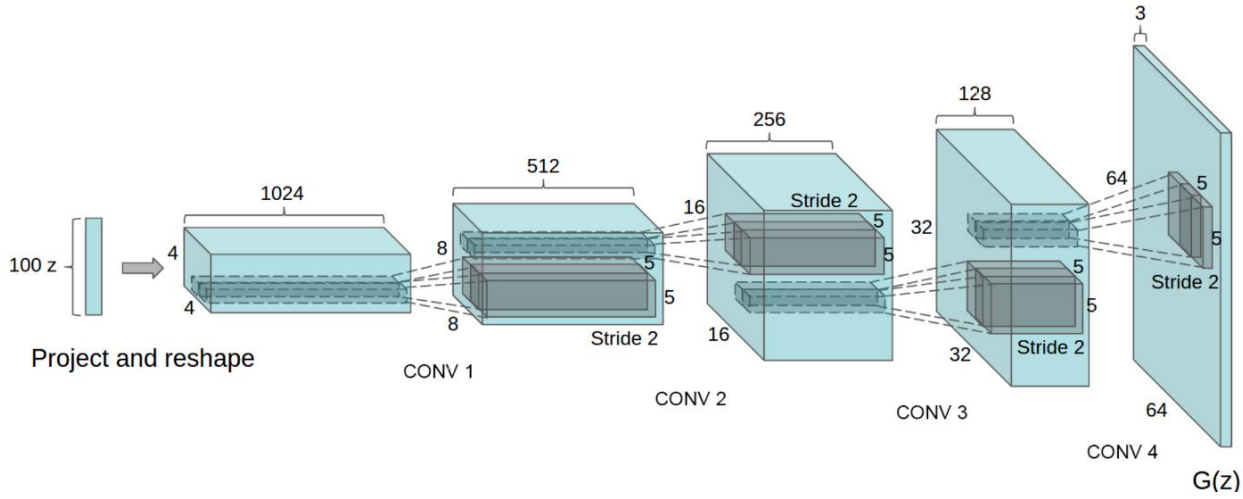


Figure 2: DCGAN generator working in a CNN

- DCGAN is one of the GAN network developers that is popular and successful. It consists mostly of convolution layers without full pooling or fully attached layers.
- For the downsampling and upsampling it uses convolutionary stride and transposed convolutions.

- DCGAN's versatility makes it thrive. We hit a certain limit, which does not automatically boost picture quality by growing the generator complexity.
- Replace single max pool with convolutionary moves. Using upsampling transposed convolution.
- We will have to remove fully linked layers. Using Batch standardization except for the generator output and the discriminator input row.
- We will be using ReLu and LeakyReLu but with the tahn function for the output of the discriminator.

2.2 W-GAN

- 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).

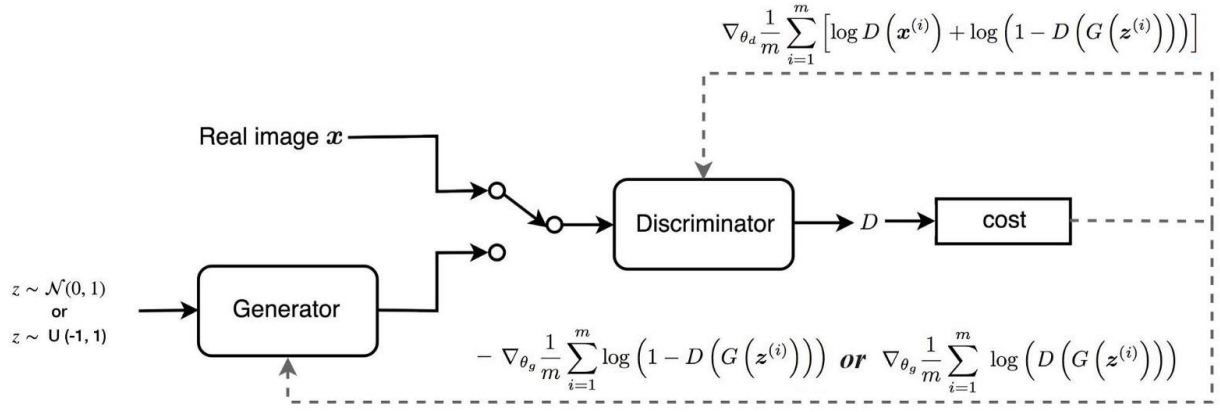


Figure 3: Working of WGAN with loss functions and Generator/Discriminator model

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.

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_119 (Conv2D)          (None, 32, 32, 64)        1792
_________________________________________________________________
leaky_re_lu_37 (LeakyReLU)   (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_120 (Conv2D)          (None, 16, 16, 128)       73856
_________________________________________________________________
leaky_re_lu_38 (LeakyReLU)   (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_121 (Conv2D)          (None, 8, 8, 128)         147584
_________________________________________________________________
leaky_re_lu_39 (LeakyReLU)   (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_122 (Conv2D)          (None, 4, 4, 256)         295168
_________________________________________________________________
leaky_re_lu_40 (LeakyReLU)   (None, 4, 4, 256)         0
_________________________________________________________________
flatten_6 (Flatten)          (None, 4096)              0
_________________________________________________________________
dropout_6 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_10 (Dense)             (None, 1)                 4097
=================================================================
Total params: 522,497
Trainable params: 522,497
Non-trainable params: 0
_________________________________________________________________
```

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 quadrick the field in the input maps of the Conv2DTranspose row. Two more cycles to hit the performance image of 32x32.

4. Performance and Results (Inception / 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 Inception and FID Score

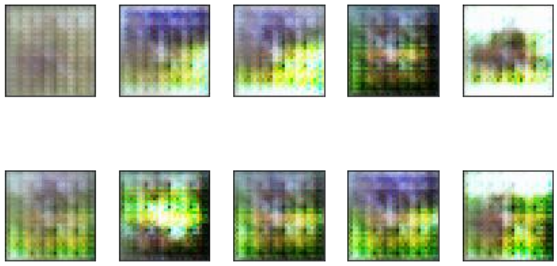
Inception and FID scores for 50,000 samples of CIFAR-10 which were trained until epoch 100 for 200 iterations.

Scores type	WGAN	DCGAN
Inception Score	9.18	6.35
FID (5k)	24.2	48.3
FID	15.9	42.6
GAN - train	25.7	4.9
GAN - test	39.3	2.4

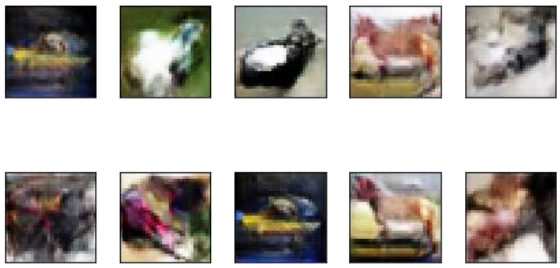
5. Output Images

5.1 WGAN

Epoch 0

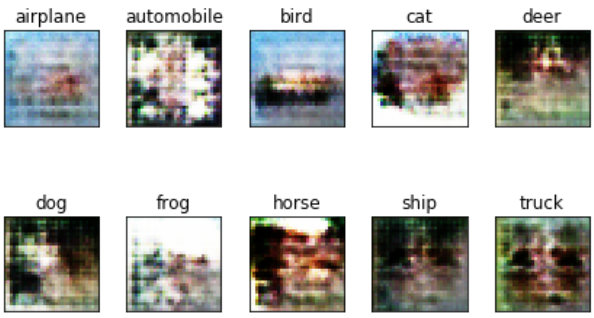


Epoch 100



5.2 DCGAN

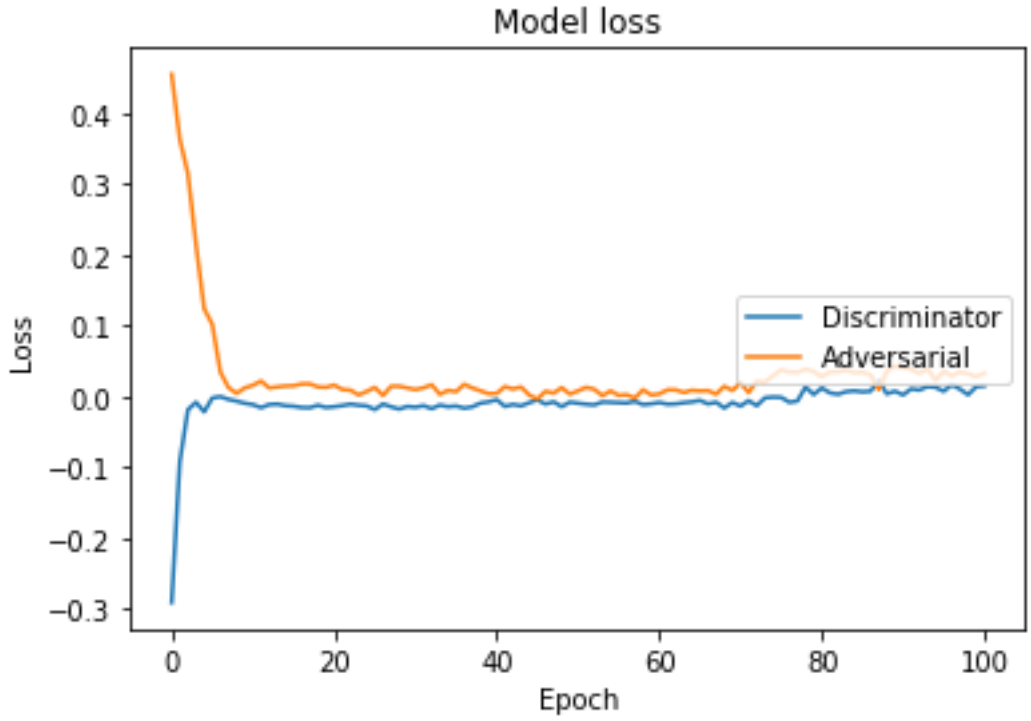
Epoch 0



Epoch 100

6. Loss Function

6.1 WGAN



6.2 DCGAN