CPSC 8430: Deep Learning – Homework 3

Sadegh Sadeghi Tabas

GitHub: https://github.com/sadeghitabas/CPSC8430-DeepLearning.git

I- CIFAR10 Dataset

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.

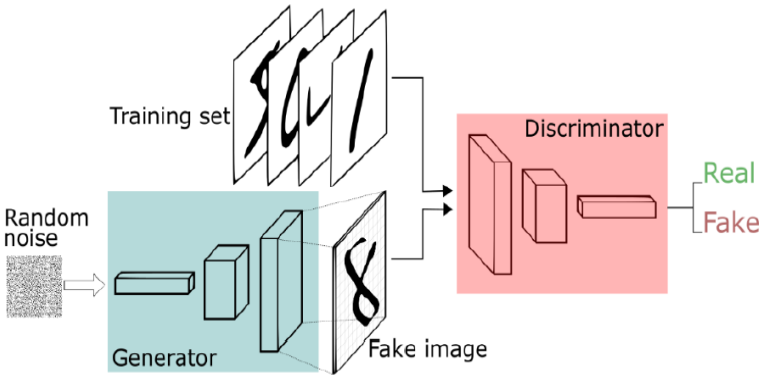
The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

II- GANs

A generative adversarial network (GAN) is a class of machine learning frameworks designed by Ian Goodfellow and his colleagues in 2014. Two neural networks contest with each other in a game (in the form of a zero-sum game, where one agent's gain is another agent's loss).

GAN neural networks, contain two networks called the generator and discriminator networks. The generator network produces new datapoint information, and the discriminator checks the accuracy. Discriminator, in other words 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.

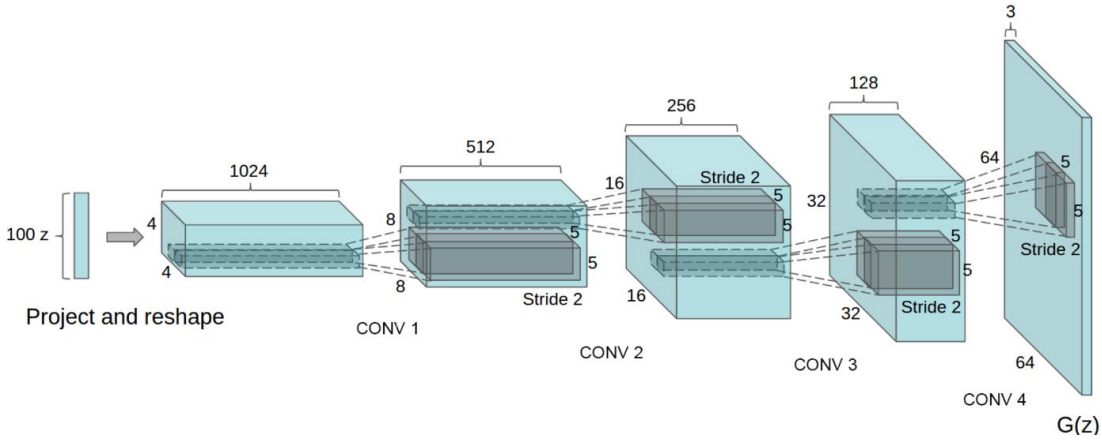


Given a training set, this technique learns to generate new data with the same statistics as the training set. For example, a GAN trained on photographs can generate new photographs that look at least superficially authentic to human observers, having many realistic characteristics. Though originally proposed as a form of generative model for unsupervised learning, GANs have also proven useful for semi-supervised learning, fully supervised learning, and reinforcement learning.

III- DCGAN

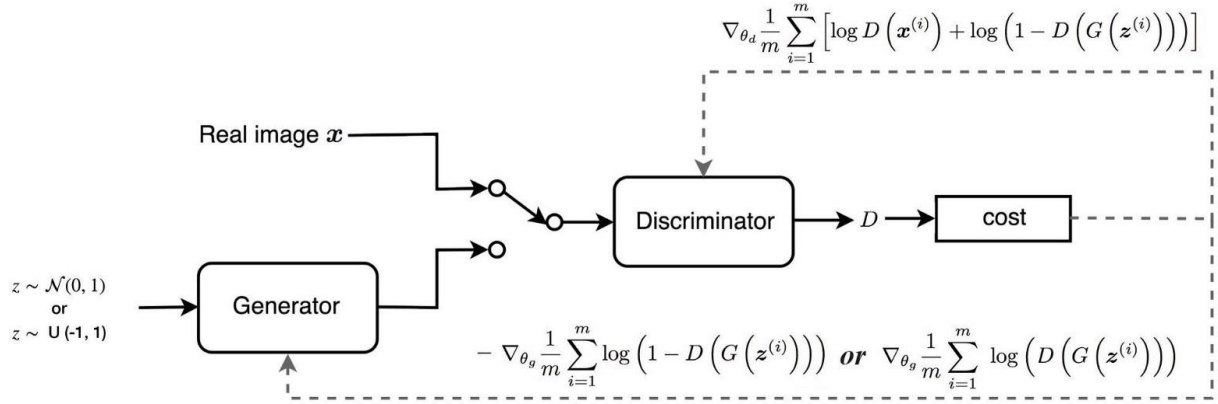
DCGAN is a class of CNNs called deep convolutional generative adversarial networks (DCGANs), that have certain architectural constraints, and demonstrate that they are a strong candidate for unsupervised learning. DCGAN is one of the GAN network developers that is so popular and successful. It consists

mostly of convolution layers without full pooling or fully attached layers. For the down sampling and up sampling, it uses coevolutionary 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 coevolutionary moves. Using up sampling transposed convolution. We also will have to remove fully linked layers. Using Batch normalization 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. The DCGAN structure used in this assignment is provided as follows



IV- WGAN

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. WGAN has a strong mathematical motivation, although only a few minor changes to the standard, deep coevolutionary, generative opponent network or DCGAN require in practice. Instead of sigmoid function, it uses the linear activation feature in the essential model's output sheet. It uses -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]). It also checks the essential model per iteration more than the generator. It's based on the variant of RMSProp with low learning and no momentum (for example 0.00005).



V- WGAN-GP

The proposed Wasserstein GAN (WGAN) makes progress toward stable training of GANs, but sometimes can still generate only low-quality samples or fail to converge. These problems which are often due to the use of weight clipping in WGAN to enforce a Lipschitz constraint on the critic, which can lead to undesired

behavior. WGAN-GP proposed by Gulrajani et. al. (2017) provided an alternative to clipping weights: penalize the norm of gradient of the critic with respect to its input.

VI- ACGAN

The ACGAN consists of a generator and a discriminator, as any GAN network does. However, in the ACGAN, every generated sample has a corresponding class label c ~ C(available classes) in addition to the noise z. This class label helps the model to synthesize the image based on the label passed. The generator G uses both the class label and the noise z to generate the image. The Generated images can be represented as A transposed convolutional layer is the same as a convolutional layer, but with added padding to the original input. As a result of this, when the convolution is applied with stride 1 and no padding, the height and width of the output is more than that of the input. With a stride of 2, we can perform up-sampling with a transposed convolutional layer — same as we can perform down-sampling with a convolutional layer of stride 2. The transposed convolutional layers of the generator is backed up with ReLU non-linearity.

The discriminator contains a set of 2d convolutional modules with leaky-ReLU non-linearity followed by linear layers and a softmax and sigmoid function for each of its outputs — detecting the class and the source of the model.

The loss function of the ACGAN is divided into two parts a) The log likelihood for the source being checked b) The log likelihood of the class being checked.

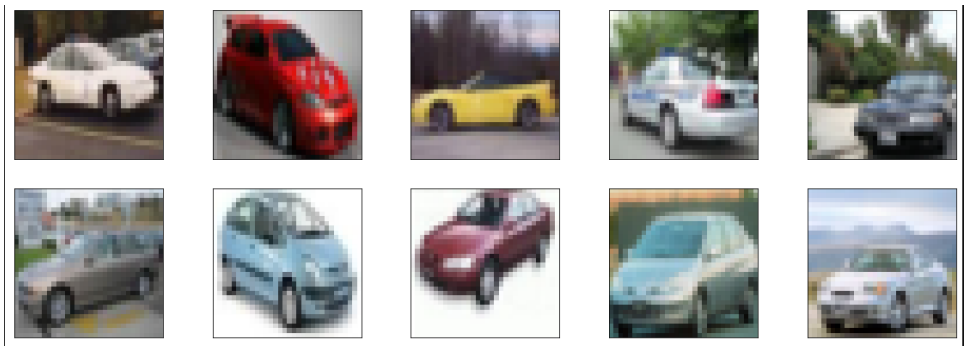
As is evident from the above loss functions, the generator and the discriminator 'fight' over this loss function. The generator and the discriminator both try to maximize the class-loss. The source-loss is however a min-max problem. The generator tries to minimize the source-loss and fool the discriminator. The discriminator on the other hand tries to maximize the source-loss and tries to prevent the generator from gaining an upper hand.

VII- Requirements:

- - Python 3
- - Tensorflow 1.15
- - CIFAR-10 Dataset

VIII- Implementation:

In order to implement various networks and evaluate their performance on generating fake images based on CIFAR10 dataset, first we downloaded and extracted the CIFAR10 dataset using the download link provided in the project description. As we mentioned above the CIFAR10 dataset contains 10 classes of images and in order to model any of those classes we need to extract that class and then train the GANs network based on the selected class dataset. In this assignment we selected the class number 2 which presents different images of cars. However, based on the code it is possible to easily change the class and evaluate the GANs network on the other classes of the CIFA10 dataset (just we need to specify the class number in the code). Below figure indicates a sample of car class in the CIFAR10 dataset.



The implementation procedure and steps are quite similar for all methods and the part we need to take care of that for different GANs networks is implementation of Generative and Discriminative networks. For any of the implemented GAN models we provided a jupyter notebook file which you can easily run it through palmetto jupyter hub. The only requirement that should be satisfied to be able to run the models is using tensorflow version 1.15 and I guess the other packages used in the codes are not very sensible to the version as I tested them on google colab as well. For more information regarding the generative and discriminative networks please refer to the codes provided in the github account (the github repo is publicly available and the link is provided in the top of the report).

IX- Results and Performance Assessment

The best 10 fake images generated by any implemented GAN is presented as follows:

DCGAN:

WGAN:

WGAN-GP:

ACGAN:

In order to compare the performance of different GAN model, first we shuffled the CIFAR10 dataset and extract 2560 random images. Next, we used the PID calculator package provided by TA to assess the performance of different m

odels. In order to use the PID calculator package we provided the two datasets containing 2560 random images extracted from CIFAR10 dataset and the 2560 fake images provided from each model. Finally, the two folders containing the different datasets for each model input to the pid.py code and the relevant PID score calculated. The results provided in the table below for each model.

Model	PID Score
DCGAN	36.21
WGAN	43.53
WGAN-GP	39.17
ACGAN	51.73

As we can see in the results the DCGAN model has the best PID score among all models. However, it may be because I spend most of my time to tune the DCGAN hyperparameters and did not put much efforts to tune other models.

References:

https://www.cs.toronto.edu/~kriz/cifar.html

https://en.wikipedia.org/wiki/Generative_adversarial_network

https://arxiv.org/abs/1511.06434

https://arxiv.org/abs/1701.07875

https://arxiv.org/abs/1704.00028

https://arxiv.org/abs/1610.09585

https://github.com/AliceAria/Performance-comparison-of-GAN-on-cifar-10