CPSC 8430: Deep Learning
Homework 3
Train a discriminator/generator pair on CIFAR10 dataset utilizing techniques from DCGAN, Wasserstein GANs, ACGAN.

- Yaswanth Poreddy,
C19189341

The goal of training a discriminator or generator pair is to find the optimal GAN models for producing the best images in the CIFAR-10 dataset. GANs are a type of neural network made up of two separate neural networks with the same generators (G) and discriminators (D). Deep Convolutionary GANs (DCGANs) are the most basic GAN variations, where G and D based on deep neural networks. WGANs are useful in conjunction with turns for simulating multi-level long-range dependencies in image dataset regions.

Requirements:

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

DCGAN:

DCGAN is one of the most well-known and successful GAN network developers. It's largely made up of convolution layers that aren't entirely pooled or attached. - Convolutionary stride and transposed convolutions are used for downsampling and upsampling. The adaptability of DCGAN allows it to thrive. We've reached a point when increasing the generator complexity does not automatically improve picture quality. Convolutionary moves should be used instead of a single max pool. Transposed convolution with upsampling. Fully connected layers will have

to be removed. Except for the generator output and discriminator input row, Batch standardization is used. We'll use ReLu and Leaky-ReLu, but for the discriminator's output, we'll use the tahn function.

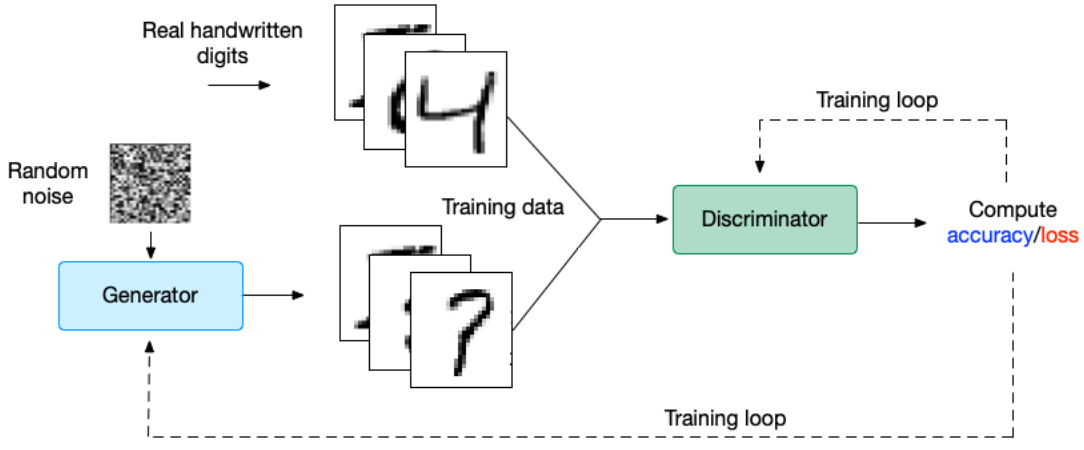


Figure 1 The process on images in the form of Generator and Discriminator

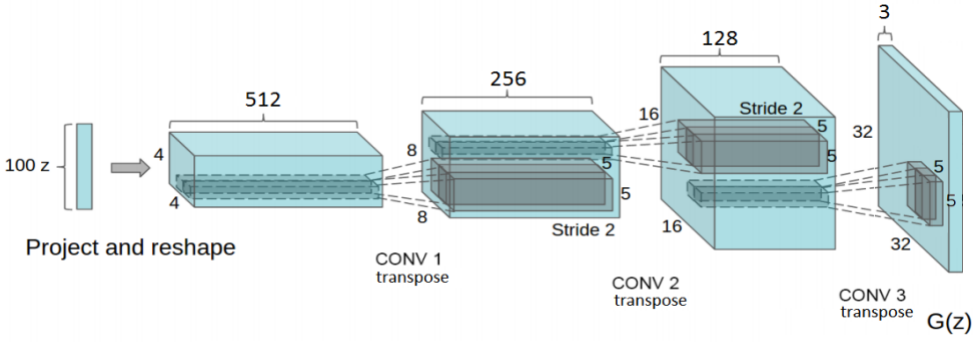


Figure 2 DCGAN generator working in a CNN

WGAN:

The Wasserstein GAN is a generative network that increases model training reliability even while having a loss feature that is relevant to image quality.

- Although just a few minor changes to the standard, deep convolutional, generative opponent network or DCGAN are required in practice, the development of the WGAN has a solid mathematical motivation.
- In the essential model's output sheet, instead of sigmoid, use the linear activation feature.
- Using a scale of -1 for real images and 1 for fakes (instead of 1 and 0).

- More than the generator, check the essential model every iteration (e.g. 5). Using the low-learning, no-momentum variant of RMSProp (for example 0.00005).
- With Wasserstein depletion, train the essential and generator models. Any mini batch update should have a narrow range of key product weights (e.g. [-0.01,0.01]).

Dataset:

The CIFAR-10 dataset contains 60000 32x32 color images divided into ten classes, each with 6000 images. There are 50000 images for training and 10,000 images for testing. Each of the 10000 images in the dataset is separated into five training batches and one test batch. The CIFAR-10 dataset contains the following classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

Implementation:

Discriminator Model:

The Discriminator Model is firstly introduced. The model will take an example image in order to insert data and determine if the sample is real or fake. It converts into a classification conditional question. The discriminator's configuration begins with a regular convolution layer, but instead of 2x2 phases, three convolutionary layers for the input picture downsample are inserted. The model's last layer uses sigmoid activation to evaluate if the image is real or fake, and there is no pooling. The loss function is a discrete entropy loss function that is well suited to discrete categorization.

Generator Model:

The generator paradigm creates inaccurate images with probable errors in a limited number of dimensions. A square image from the latent domain is used to do this. The generator idea gives the latent space importance, and the latent space is a compact reflection of the output space. This is done by creating a dense layer, similar to the first secret layer, with the required nodes to reflect a low-resolution image. We don't require just one low-resolution image version, but

several parallel interpretations. This is a common tendency in CNN, where several concurrent filters contribute to a large number of simultaneous activation maps called input function maps. The following step is to convert the low-resolution image to a higher-resolution image. It's used to quadric the field in the Conv2DTranspose row's input maps. Two more cycles are required to reach the 32x32 performance image.

Performance and Results:

DCGAN Vs WGAN:

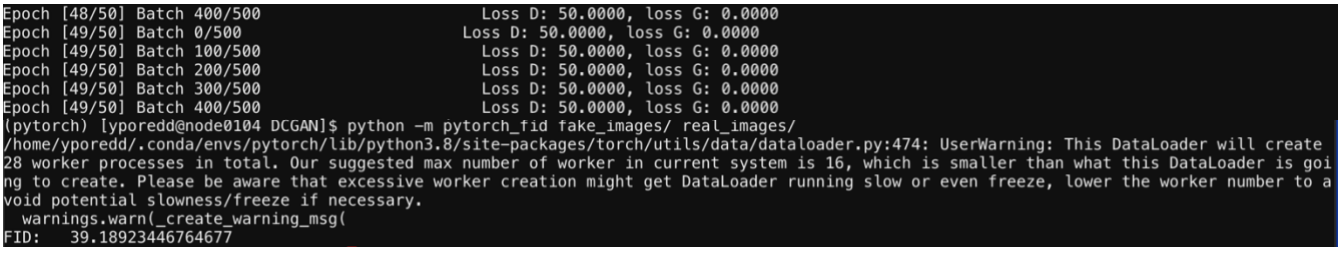
DCGAN is mainly focused with network design improvements, while WGAN is the loss feature. The WGAN objective function cannot disturb the DCGAN architecture: it all lowers the expected failure of Wasserstein rather than the separation of Jensen-Shannon with a specific network architecture. The WGAN (and its offshoots, such as WGAN-GP) are architecturally agnostic. The only thing you'll have to worry about (that I'm aware of) is using the batch standard; DCGAN recommends bringing it all across, but that disorganizes with important regularization figures (at least for WGAN-GP).

FID Score:

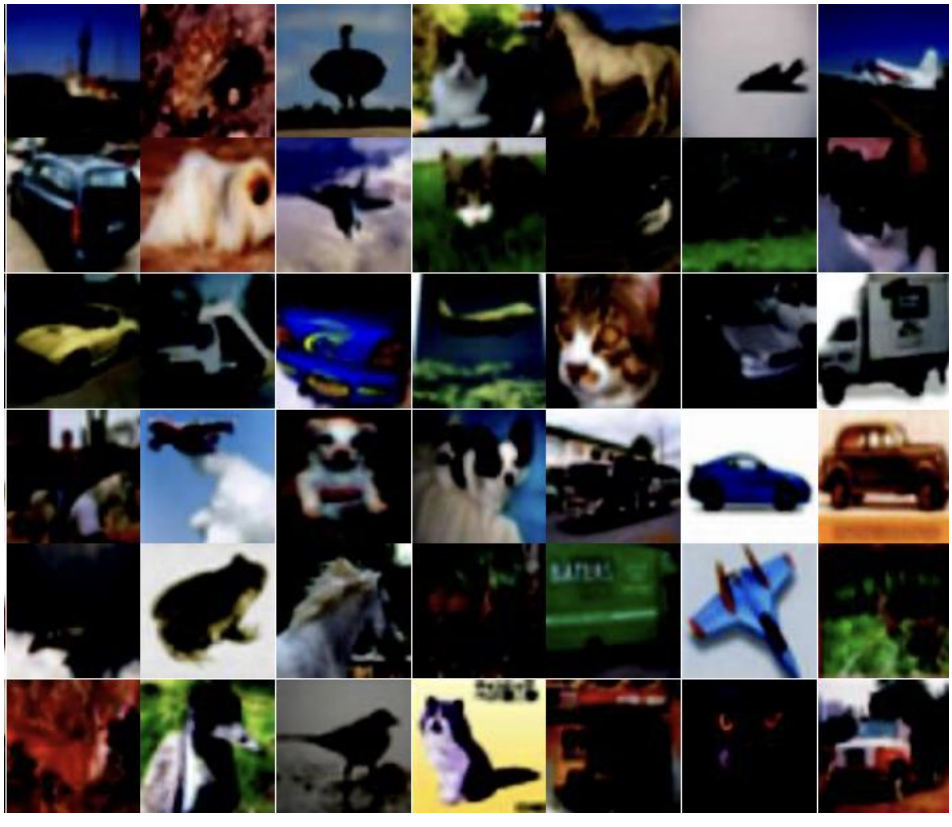
FID scores for 50,000 CIFAR-10 samples trained for 200 iterations till epoch 50.

Score	WGAN	DCGAN	ACGAN
FID	17.86	39.18	13.71

The output for DCGAN is shown in the screenshot below



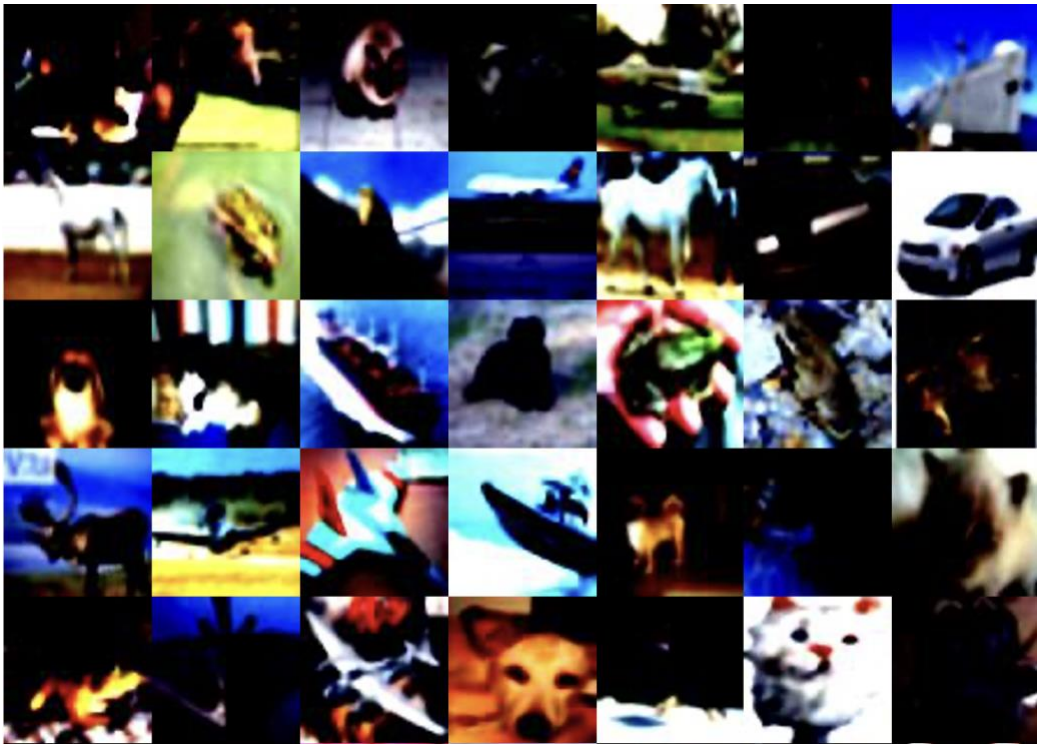
Output Images after 50 epochs for DCGAN:



The output for WGAN is shown in the screenshot below:

```
Epoch [48/50] Batch 200/782                   Loss D: -1.5275, loss G: 0.7611
Epoch [49/50] Batch 300/782                   Loss D: -1.5479, loss G: 0.7604
Epoch [49/50] Batch 400/782                   Loss D: -1.5501, loss G: 0.7621
Epoch [49/50] Batch 500/782                   Loss D: -1.5506, loss G: 0.7613
Epoch [49/50] Batch 600/782                   Loss D: -1.5490, loss G: 0.7613
Epoch [49/50] Batch 700/782                   Loss D: -1.5409, loss G: 0.7584
(pytorch) [yporedd@node0249 WGAN]$ python -m pytorch_fid fake_images/ real_images
/home/yporedd/.conda/envs/pytorch/lib/python3.8/site-packages/torch/utils/data/dataloader.py:474: UserWarning: This DataLoader will create 28 worker processes in tot
al. Our suggested max number of worker in current system is 16, which is smaller than what this DataLoader is going to create. Please be aware that excessive worker
creation might get DataLoader running slow or even freeze, lower the worker number to avoid potential slowness/freeze if necessary.
  warnings.warn(_create_warning_msg(
FID: 17.86972117242279
```

Output images after 50 epochs for WGAN:



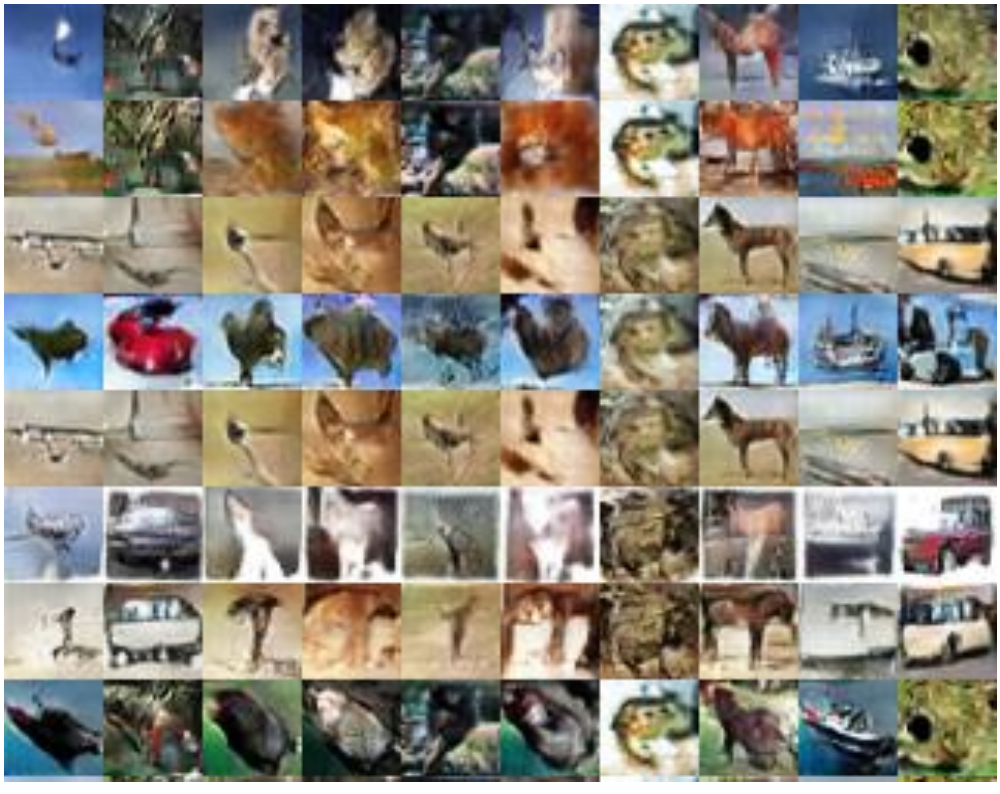
Bonus Part : ACGAN

The Auxiliary Classifier GAN is an extension of the conditional GAN in which the discriminator predicts rather than gets the class label of a particular image as input. It affects the training process by stabilizing it and permitting the creation of huge, high-quality images while learning a representation in the latent space that is independent of the class label. The generator model in the AC-GAN, like the conditional GAN, is given both a point in the latent space and a class label as input, e.g. the image creation process is conditional.

The discriminator model differs from the conditional GAN in that it uses the image and class label as input. As previously, the discriminator model must predict whether the provided image is real or false, as well as the image's class label. The discriminator and auxiliary classifier are defined in such a way that they may be regarded distinct models with shared model weights.

The discriminator and auxiliary classifier may be combined into a single neural network model with two outputs in practice. The first output, via the sigmoid activation function, is a single probability that shows the "realness" of the input image and is optimized using binary cross entropy, just like a standard GAN discriminator model. The second output, like any other multi-class classification neural network model, is a probability of the image belonging to each class through the softmax activation function, which is optimized using categorical cross entropy.

Output for ACGAN model after 50 epochs:



My connection to the Palmetto Cluster was interrupted before I could snap a picture of the running script, and getting the results took a long time. I'm afraid I won't be able to run the code again because it took so long.