CPSC 8430 Homework 2

DCGAN Design and Training Methods:

For the DCGAN a simple approach was taken where the DCGAN architecture was directly implemented in Pytorch. The focus of implementing the DCGAN was to achieve a baseline performance of an FID score of 35. After achieving the baseline the focus was shifted to increasing FID score based on hyperparameters different from the original parameters suggested in the DCGAN paper.

During training, the most influential hyperparameters were epochs, latent vector size, batch size, and learning rate. In total the best FID score achieved from the DCGAN was from **520 epochs**, a **latent vector size of 200**, a **batch size of 128**, and a **learning rate of 0.0002**. The training method used to further decrease the DCGAN FID score was the utilization of saved states in the network followed by many repeated training sessions of that saved state. The reason this helped to decrease the FID score is because DCGANs are unstable when training for large epochs. This training method allows the researcher to retrain from a previous saved state in case the DCGAN collapses resulting in incremental improvements in the FID score. Another very helpful observation is that a latent vector of size 200 seemed to consistently produce better results than the suggested vector size of 100 from the DCGAN paper.

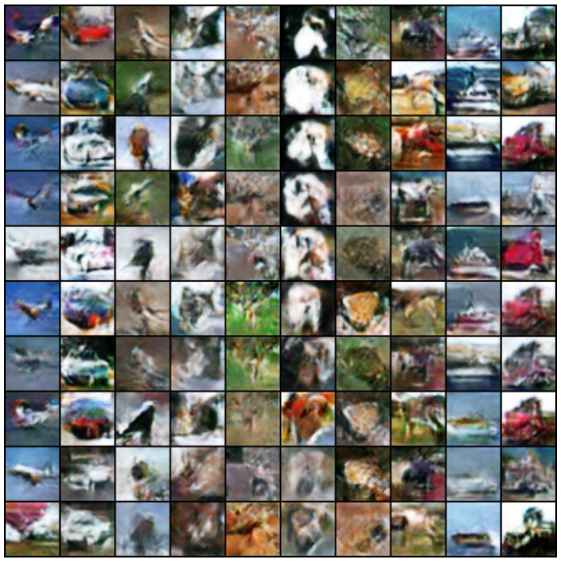
WGAN Design and Training Methods:

Since WGAN is just a loss calculation of a GAN network the WGAN implementation was created on top of the existing DCGAN network from the prior section. The hyperparameters chosen were the same as the WGAN paper. Two new technique that was attempted in the WGAN implementation was gradient clipping. Gradient clipping helps to solve the problem where a gradient can explode to very large numbers. The model was able to take advantage of this because without gradient clipping the model's Wasserstein loss would jump to well past -2000. Large loss led to unstable results and "smeared" images from the generator of the network.

The only difference between the WGAN paper's hyperparameters and mine were a latent vector of size 200. I also found that the WGAN was much more stable and could be run for an arbitrary number of epochs without the gradient vanishing or exploding. In theory this should produce a much better FID results after training the network for a large number of epochs, but this was not the case. The WGAN image quality/realism seemed to improve but the FID score bounced anywhere from the high 30s to the just under 100. Once possible explanation is Wasserstein loss is not well suited for the model architecture of DCGAN.

ACGAN Design and Training Methods:

Lastly was the ACGAN, I tried to implement ACGAN using the DCGAN architecture but there were some fundamental problems with the data dimensions, so I opted to use the architecture purposed in the ACGAN paper. The ACGAN ran into similar problems the WGAN encountered where no matter how many epochs were used the FID score was not reduced. The network did however create very distinct classes when manually observed. Below is an example of the ACGAN generating all classes from the CIFAR10 dataset.

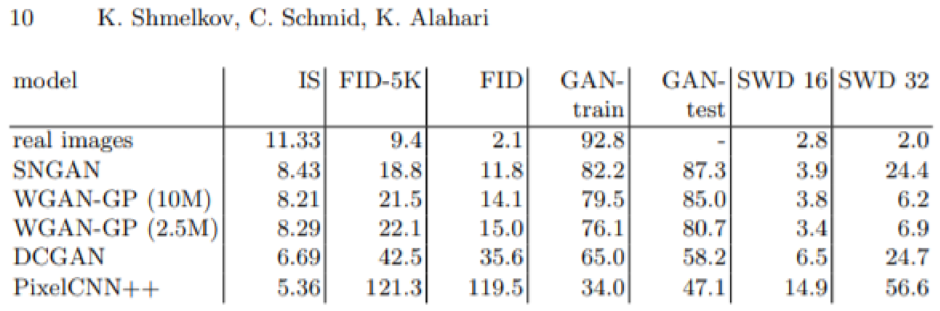


GAN Differences and Performance Evaluation:

By far the best performance was from the DCGAN with the hyperparameters mentioned before, the DCGAN achieved a score of **31.53**. Next up was the WGAN which achieved a score of **34.87**. And finally, the ACGAN achieved a score of **63.17**.

The performance gaps are obvious. A few possible explanations for the score difference are that these networks perform better when observed under different metrics. For instance, the DCGAN may perform better on FID but the images it produces are not very structured with no clear class the generator is trying to simulate. As for WGAN, it may not work well with DCGAN and might be better served being implemented on the architecture purposed in the WGAN paper. For the ACGAN score I am unsure as to why the FID score is not good, but the quality of the images is subjectively the best. The ACGAN images have very clear class boundaries with very interpretable images.

Below is a table of baseline GAN FID scores that I used to compare my GAN networks. From this table I can conclude that my DCGAN has overall good performance.



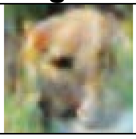
10 K. Shmelkov, C. Schmid, K. Alahari

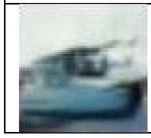
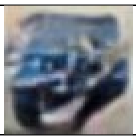
model	IS	FID-5K	FID	GAN-train	GAN-test	SWD 16	SWD 32
real images	11.33	9.4	2.1	92.8	-	2.8	2.0
SNGAN	8.43	18.8	11.8	82.2	87.3	3.9	24.4
WGAN-GP (10M)	8.21	21.5	14.1	79.5	85.0	3.8	6.2
WGAN-GP (2.5M)	8.29	22.1	15.0	76.1	80.7	3.4	6.9
DCGAN	6.69	42.5	35.6	65.0	58.2	6.5	24.7
PixelCNN++	5.36	121.3	119.5	34.0	47.1	14.9	56.6

Table from “How good is my GAN?” https://arxiv.org/abs/1807.09499

CPSC 8430 Homework 2

Top 10 GAN Produced Pictures:

Bird	**Horse**	**Dog**	**Car**	**Car**
				

Boat	**Truck**	**Truck**	**Boat**	**Plane**
				

Program Usage:

To generate images from any of the three GANs discussed in this report you can either individually run each testXGAN.py program or run the test script included in the repository. This program was run and tested on the palmetto cluster using an nvidia V100 GPU. The outputs from each program will end up in ./realX/ and ./fakeX/ directories where X is AC, DC, or W.

All FID tests were run using this command…

```
python3 -m pytorch_fid --device cuda:0 ./fakeX/ ./realX/
```

where X is AC, DC, or W.