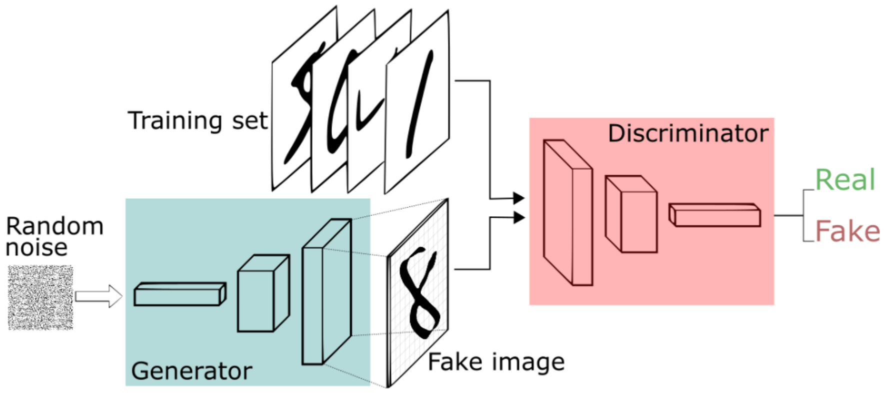
# CPSC 8430: Deep Learning Homework 3 (Mary Aiyetigbo)

https://github.com/dimky01/8430-Deep-Learning/tree/main/homework3

This homework aims to train GAN networks on the CIFAR10 dataset using techniques from DCGAN, Wasserstein GANs, and ACGAN.

## I. INTRODUCTION

Generative Adversarial Networks, known as GAN, is an unsupervised learning algorithm that involves learning patterns in input data to generate new data examples [1]. GAN uses two neural networks, Generator and Discriminator, playing adversarial with each other. The generator network generates new data instances while the discriminator network evaluates them [2]. The purpose of the generator is to fool the discriminator such that the discriminator cannot distinguish between the images generated by the generator and the actual data. The generator learns to map from a latent space to a data distribution of interest, and the discriminator evaluates if the generated data belongs to the actual training dataset [3]. Below is the architecture of the GAN network.



Source: [4]

## II. DATASET

The dataset used for homework is the CIFAR-10 dataset [5], consisting of RGB images belonging to 10 classes. There are 50,000 training samples and 10,000 test samples.

For this homework, both the training and testing images were used to have enough images for the training process. Each model was tested with a total of 60,000 images. The following preprocessing was done on the images

- The images were resized to 64x64
- The images were normalized to a range of **-1 and 1** using a mean of 0.5 and a standard deviation of 0.5. Normalization was done so that the image input to the discriminator would be consistent with that of the generator's output since the activation function of the generator network's output layer is Tanh.

## III. IMPLEMENTATIONS

### 1. DCGAN: Deep Convolutional Generative Adversarial Network

DCGAN eliminates fully connected layers and uses a convolutional stride for the discriminator network and convolutional-transpose for the generator network [6]. The generator consists of:

- Convolutional- transpose stride for up-sampling the latent vector
- Batch Normalization layers
- ReLu activation function
- An output of a 3x64x64 image with Tanh activation

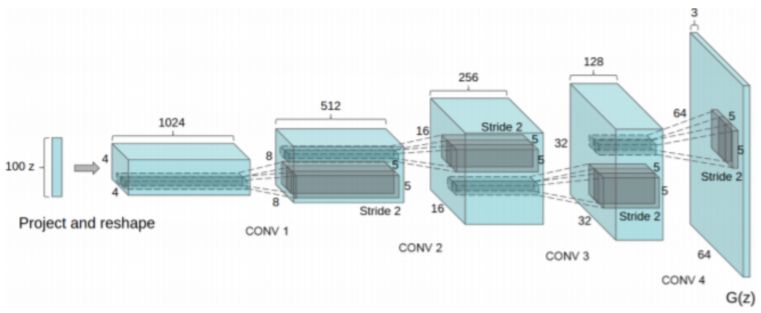


Figure 1: Architecture of the generator model with Convolutional Transpose layers

The discriminator network consists of;

- An input of a 3x64x64 image
- convolutional stride for down-sampling images
- Batch Normalization layers
- LeakyReLU activation function was used except for the output layer, which uses the sigmoid activation function.

The following hyperparameters were used for both networks

- Training Batch size: 128
- Learning rate: 2e-4
- Latent vector dimension: 100
- Optimizer: Adam with beta1=0.5 and beta2=0.999
- Loss function: BCELoss
- Weight initialization using a normal distribution with a standard deviation of 0.02

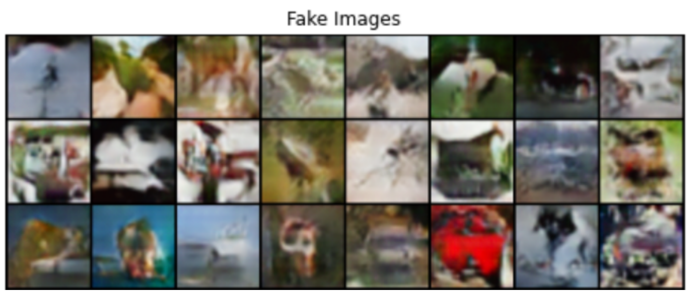


Figure 2: Sample images generated with DCGAN

## 2. WGAN: Wasserstein Generative Adversarial Network

WGAN is an improvement on the GAN networks, and the primary purpose is to stabilize training. GAN uses the Jensen-Shannon divergence method to measure the distance between the probability distribution of the generative and distributive models. However, the JS divergence has unstable training issues while working with gradients. Hence, the use of Wasserstein distance to stabilize training [8]. The Discriminator network in WGAN is referred to as Critic, which aims to maximize the distance between the real data and the generated data using Wasserstein distance. The WGAN model has the following features:

- Sigmoid function at the output layer of the critic model was replaced with a linear activation
- Weight Clipping was used on the parameters of the Critic model
- It uses a small learning rate
- Uses RMSProp for optimization with no momentum
- It might take a longer time to train if the weight clipping is large

The network structure of the WGAN was similar to that of DCGAN; however, for the Critic model, instead of the sigmoid function at the output layer, it uses linear activation as the output is no longer a probability. The critic model/discriminator was trained 7 times for each epoch, while the generator was trained once during training. Weight clipping was applied on the critic model for each training iteration

The following hyperparameters were used for both networks

- Training Batch size: 64
- Learning rate: 5e-5
- Latent vector dimension: 100
- Optimizer: RMSProp with Momentum = 0
- Weight Clipping = 0.5
- Critic Iteration = 7
- Wasserstein Loss = mean of predicted and true
- Weight initialization using a normal distribution with a standard deviation of 0.02

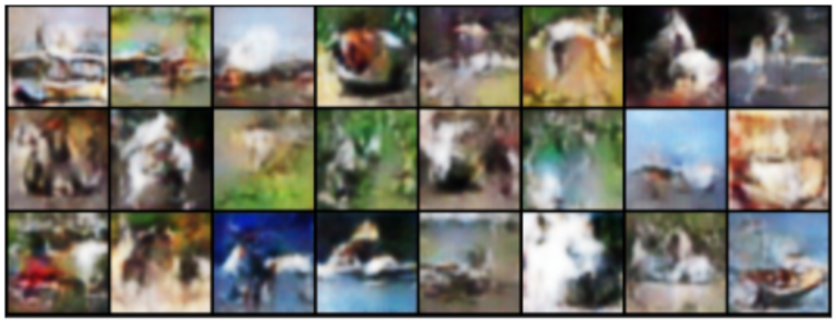


Figure 3: Sample images generated with WGAN

## 3. WGAN-GP: Wasserstein Generative Adversarial Network with Gradient Penalty

WGAN-GP is a slight improvement in the training mechanism of WGAN. The weight clipping method in WGAN was found not always to work [8] because when the weight clipping is too large, the model takes a long time to train as the critic model took much time adjusting to the expected weights, the model might not converge. When weight clipping is small, it leads to vanishing gradients. Hence, WGAN-GP proposed by [9] tends to mitigate these issues with the use of gradient penalty where the norm of the gradient of the Critic with respect to its input is penalized

The following hyperparameters were used for both networks

- **Training Batch size:** 64
- **Learning rate:** 1e-4
- **Latent vector dimension:** 100
- **Optimizer:** Adam with beta1=0.5 and beta2=0.999
- **Critic Iteration** = 7
- **LAMBDA** for Gradient Penalty = 10
- **Wasserstein Loss** = mean of predicted and true
- **Weight initialization** using a normal distribution with a standard deviation of 0.02

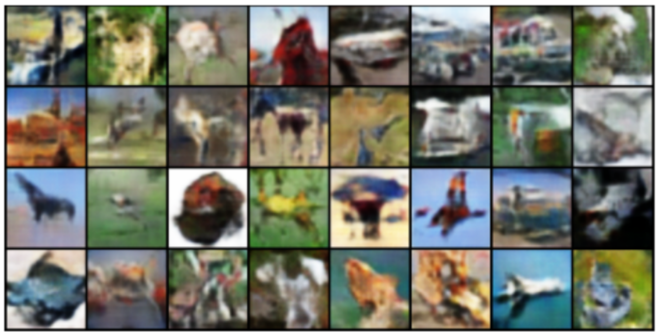


Figure 4: Sample images generated with WGAN-GP

## 4. ACGAN: Auxillary Conditional Generative Adversarial Network

For ACGAN, the input to the generator is latent noise and its label, and the output is a fake image that belongs to the input class label. For the discriminator, the input is an image, and the model predicts the probability that a given image is real and its class label [9]. The class label helps the model synthesize the image based on the label passed. The network structure of the discriminator is similar to DCGAN, except that the ACGAN has two outputs, the probability of the image and the class label. Also, the network architecture of the generator is the same as DCGAN, but it takes latent noise and labels as input. Embedding was used to represent the latent vector before being passed into the model.

ACGAN computes two losses;

- Binary cross-entropy loss for the image
- The log-likelihood of the class of the image

The following hyperparameters were used for both networks

- Training Batch size: 64
- Learning rate: 2e-4
- Latent vector dimension: 100
- Embedding size: 100
- Optimizer: Adam with beta1=0.5 and beta2=0.999
- Critic Iteration = 7
- LAMBDA for Gradient Penalty = 10
- Image Probability Loss = BCELoss
- Label/class loss = negative log-likelihood loss (NLL_Loss)
- Weight initialization using a normal distribution with a standard deviation of 0.02

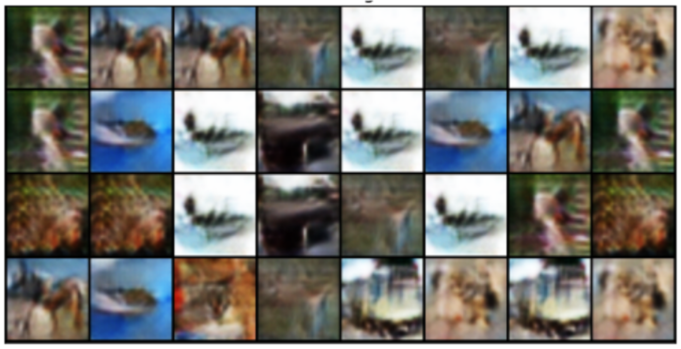


Figure 5: Sample images generated with ACGAN

## IV. PERFORMANCE ASSESSMENT

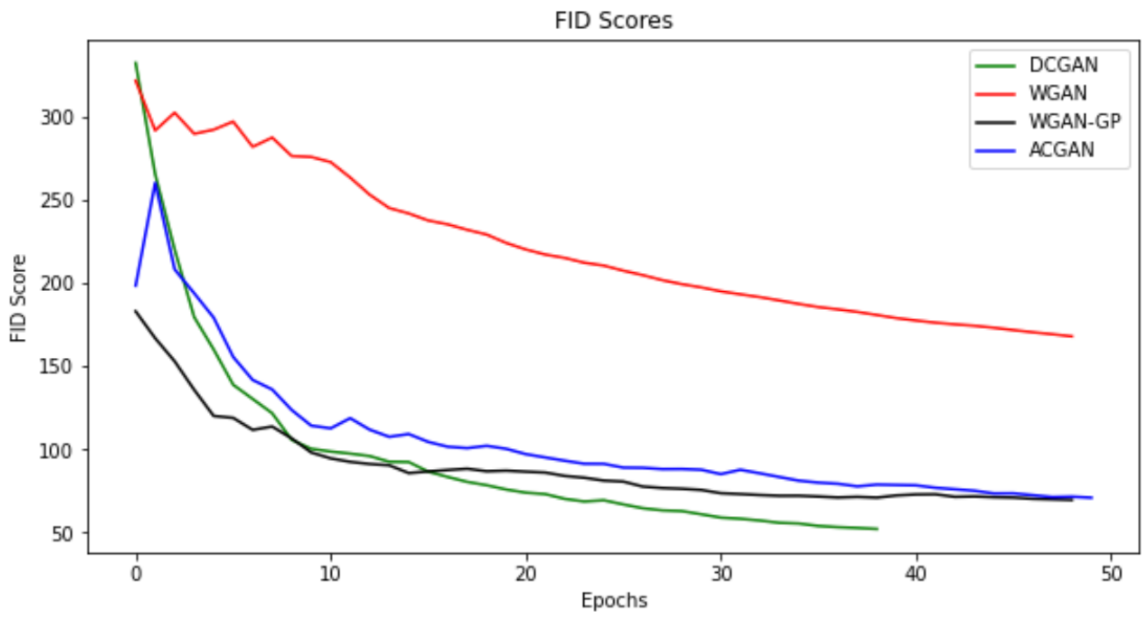
Frechet Inception Distance (FID) was used to evaluate the performance of the implemented GAN models (DCGAN, WGAN, WGAN-GP, and ACGAN). FID is a metric that calculates the distance between the feature vectors generated for the real and generated images. Using the Inceptionv3 model, feature vectors of the real images and generated images were extracted, and the FID score was used to evaluate the similarities between the images. Lower scores indicate that the images are identical. The FID code used for evaluating these models was adopted from [10] and [11].

The last batch of training data and images generated by the Generator models were saved to file to compute the FID score for every epoch. After training is completed, FID is calculated between these groups of images. Below is a table showing the FID score for each model.

| Models | FID SCORE |
|---|---|
| DCGAN | 52 |
| WGAN | 168 |
| WGAN-GP | 69.6 |
| AC-GAN | 70.9 |

From the image above, DCGAN has the lowest FID Score of **52**, which means that based on Frechet Inception Distance, DCGAN performed the best. However, from the quality of the generated images, I would WGAN-GP had the best performance because it was easier to recognize what class the images generated belong to. ACGAN also generated clear images, as shown in Figure 5 above. Images of ships and horses were visible in the image screenshot. ACGAN also had the second-highest FID score of **70.9**.

The plot below also shows the FID score graph plotted for each model at every epoch



## References

1. https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/
2. https://en.wikipedia.org/wiki/Generative_adversarial_network
3. https://wiki.pathmind.com/generative-adversarial-network-gan
4. https://www.freecodecamp.org/news/an-intuitive-introduction-to-generative-adversarial-networks-gans-7a2264a81394
5. CIFAR 10: https://www.cs.toronto.edu/~kriz/cifar.html
6. https://arxiv.org/pdf/1511.06434.pdf
7. https://blog.paperspace.com/wgans/
8. WGAN-GP: https://arxiv.org/pdf/1704.00028.pdf
9. https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781788629416/5/ch05lvl1sec31/auxiliary-classifier-gan-acgan
10. https://github.com/mseitzer/pytorch-fid
11. https://www.kaggle.com/code/ibtesama/gan-in-pytorch-with-fid/notebook