CPSC-8430: Deep Learning - Homework 4

The main aim of this assignment is to understand how Generative Adversarial Networks (GANs) can be a great technique to generate new images based on the patterns and structures of data and also implement the extended versions of GANs and observe their performance.

Github Link: https://github.com/laxman2405/DeepLearning_HW4/tree/main

Introduction

This report summarizes the implementations of extended versions of different GANs. A GAN is a technique in deep learning architecture that aims to generate new data from the given dataset most likely images. It consists of Generator for creating fake data and Discriminator that compares to differentiate real and fake data. In the sections below, I have implemented the extended versions of GANs namely Deep Convolutional GAN, Wasserstein GAN and Auxiliary Classifier GAN.

As for the dataset, I have chosen the CIFAR-10 dataset which is very much used for image classification and generation. It contains 60,000 color images (Training set - 50,000 and Testing set - 10,000) with each image having 32x32 pixels of resolution, spanning across 10 different classes.

Common Calculations for all GANs

The quality of generated images can be identified by Frechet Inception Distance (FID) score. This score compares the generated images with the real images and calculates the score. Here, I created a file **calculate_fid.ipynb,** to compute the FID score. The calcualte_fid_score() takes in real and fake images and extracts the high-level features using a pre-trained **InceptionV3FeatureExtractor** class. Then, mean and covariance for these features are computed and Frechet distance is calculated between the two. A low FID score represents, the generated image is closer to the real image.

Overview of GANs

1. Deep Convolutional GAN (DCGAN)

This technique extends the GAN architecture by utilizing convolutional neural networks in both Generator and Discriminator. It uses different convolutional layers to improve the stability and produce higher quality images. Both neural networks use batch normalization while training, while the generator uses ReLU activation function and discriminator uses LeakyReLU.

2. Wasserstein GAN (WGAN)

This technique was introduced to address the problem in the traditional GAN model of generators becoming collapsed while producing smaller outputs. Here, instead of cross entropy loss, the loss is computed using Wasserian distance which provides a more robust way to find differences between real and fake images.

3. Auxiliary Classifier GAN (ACGAN)
This technique adds an extra layer of auxiliary classifier in discriminator to the existing GAN to generate images based on conditions. With discriminator generating class labels, ACGAN can send these to the generator, so that it can produce class specific data, which inturn improves the overall data quality.

Architectures of GANs
1. DCGAN

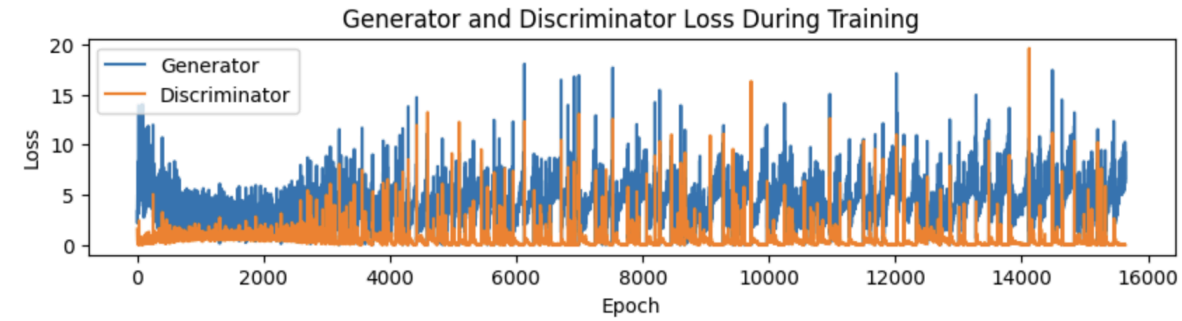
- **Generator:** For this, created a Generator class that takes text and noise as input parameters. Firstly, it processes the input text through a fully connected layer and the produced text features are combined with the noise input and passed through multiple transposed convolutional layers using batch normalization and ReLU activation function to improve image quality. A pixel value in the range of -1 and 1 is produced using Tanh function to achieve the image scalability.

- **Discriminator:** Created separate class for discriminator to distinguish between real and fake images. Firstly, text processing is done similar to a generator using a fully connected layer, but for image processing it is passed through multiple convolutional layers to extract the features and uses Leaky ReLU function along with batch normalization. Lastly, the processed images and text features are then combined using a sigmoid function to give a probability score of whether the image is real or fake.

Training
For training, a batch size of **128** is used, with **Adam** Optimizer for both generator and discriminator. A fixed **noise** dimension of **100** along with **256**-dimensional space for text embedding dimension is taken and trained over **40 epochs** with a learning ratio of **0.0002.** Below are the two generated graphs after training.

Loss Graph between generator and discriminator over epochs shows the adversarial training dynamics between two components and it's essential to identify if the training is stable or training has any issues like sudden spikes.

FID Score Vs Epoch Graph shows the FID score obtained over the training epochs. As we see from the graph, the score gradually reduced compared to the beginning score. This indicates an effective training of the model as a low score represents a better image quality.



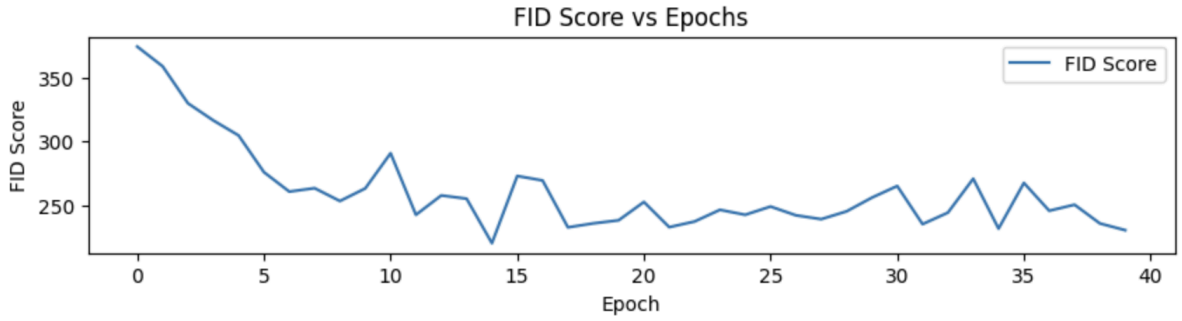
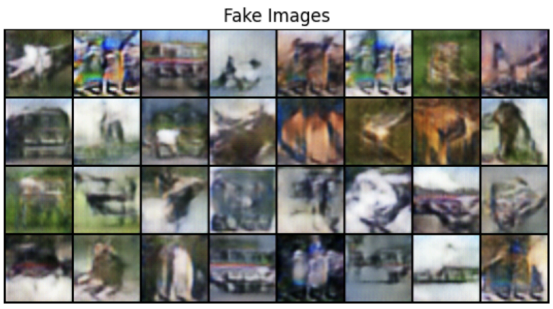


Image Results



2. WGAN

- **Generator:** This class takes in text_input and noise as the input parameters and generates the most possible realistic images. First, processes the input text to extract features from it and concatenates with the random noise vector. So, this is the condition that we apply if we use WGAN and then this single vector is passed through a series of convolutional layers and apply batch normalization with ReLU activation function after each layer to introduce non-linearity. As for scaling, we have used the Tahn activation function and generated the final output layer.

- **Critic:** This class evaluates the realness of the image based on the text and image input that was given. Processing the text is similar to that of a generator, and downsampling of image input is done by passing through a series of convolutional layers using stride, padding, Leaky ReLU function and batch normalization. Then, the processed features are combined with flattened image features to measure the image realness and outputs the final score obtained.

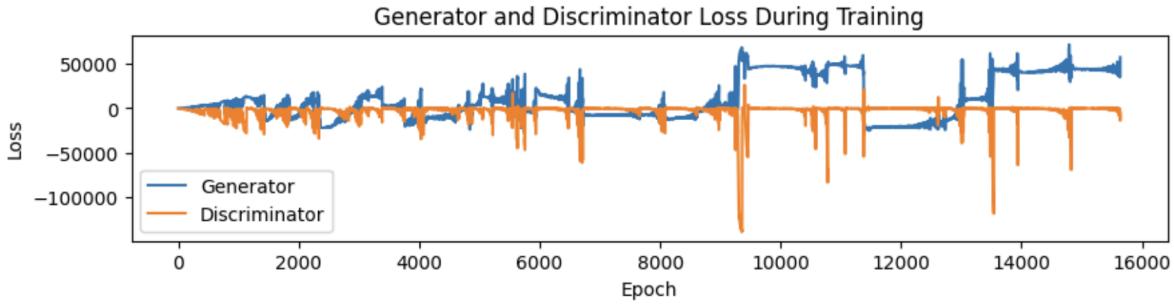
Gradient Penalty: WGAN doesn't use regular loss functions, but computes gradient penalty for the samples and input to ensure Lipschitz continuity through the training to achieve stabilization. First, generate the interpolated samples between images, find the gradient of the critic's output and make sure the gradient norm is close to 1. This value is then multiplied with a penalty constant and added to the loss function.

Training

For training, a batch size of **128** is used, with **Adam** Optimizer for both generator and critic. A fixed **noise** dimension of **100,** Critic iterating over **5 times** for each generator iteration along with **gradient penalty weight of 10** is taken and trained over **40 epochs** with a learning ratio of **0.0002.** Below are the two generated graphs after training.

Loss Graph shows the alternating patterns of loss between generator and critic indicating that training is resulting as more balanced as it progresses.

FID Score Vs Epoch shows decrease in scores as the training goes on indicating improvement in image quality and diversity.



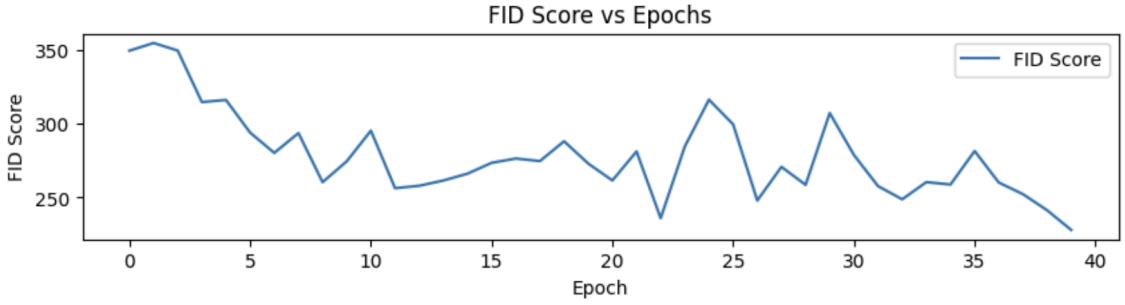
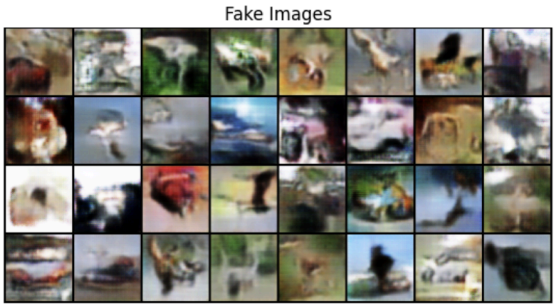


Image Results



3. ACGAN

- **Generator:** Created a Generator class that initially maps given class labels to vectors of given embedded size. The concatenated noise and embed layers are then given to the initial transposed convolutional layer for upsampling. Each inner layer is doubling the spatial resolution and reducing the number of feature maps by applying ReLU activation function at each layer. To scale pixel values that range from -1 to 1, Tanh function is also used at the final output layer. Lastly, it produces a 64x64 image with 3 channels.

- **Discriminator:** It finds the image realness by sequential downsampling of the input image with stride as 2, and kernel size as 4 using LeakyReLU and batch normalization. Also, real_fake_layer outputs a single value indicating the image is real or not and class_label uses LogSoftmax to predict the class label the image belongs to.

Training

Here, a batch size of **128** is used, with **Adam** Optimizer having **0.5** exponential decay rate for both generator and discriminator. A fixed **noise** dimension of **100,** number of classes as **10, 64** feature maps for both the components, **3 channels** for rgb images**,** and image size of **64** is taken and trained over **40 epochs** with a learning ratio of **0.0002.** Here, we compute loss at each iteration for both components using **BCELoss** and **NLLLoss.** Below are the two generated graphs after training.

Loss Graph shows the adversarial training pattern between generator and discriminator. Though the loss was high for the generator initially, it gradually decreased and for the discriminator the loss remained lower than the generator. This shows how the adversarial balance is maintained throughout training.

FID Score Vs Epoch shows similarity between real and generated images. As the score is reduced over epochs, it's an indication that the generator is learning from the process and generating better quality images.

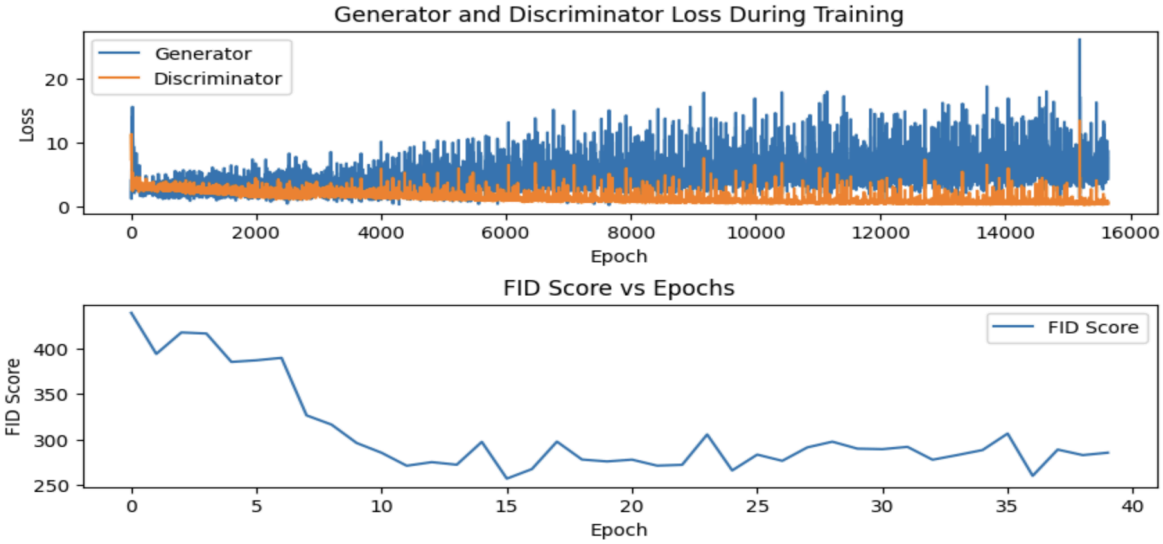


Image Results

FID Scores Comparison between DCGAN, WGAN and ACGAN

If we observer the below FID scores comparison graphs between DCGAN, WGAN and ACGAN, we can see that **DCGAN** started off with a high FID score indicating less quality images, but as the training progresses, we see the score is reduced to a much lower value, indicating increase in image quality. **WGAN,** on the other hand, started with a lower score compared to DCGAN and almost reached the same FID score as DCGAN at the end of training. Though WGAN had an increase in FID scores, in the 20-30 epoch, it eventually reduced its score attaining higher image quality. Lastly, **ACGAN** started with a very high FID score compared to others and it did have ups and downs in the score, but if we see at the end of 40th epoch, it had a FID score of around 290, which is greater than DCGAN and WGAN.

Overall, both **DCGAN and WGAN** seem to be performing well. But, between them, we can say **DCGAN** offers best performance because it maintained a consistent lower FID score compared to WGAN and ended with less score resulting in much better image quality and performance.