Homework 4

Shiva Kalyan Sunder Diwakaruni

CPSC 8430 – DEEP LEARNING

Github link: https://github.com/kalyan1998/CPSC-8430-001---DeepLearning/tree/main/HW4

Introduction

This report examines three significant Generative Adversarial Network (GAN) models: Deep Convolutional GAN (DCGAN), Wasserstein GAN (WGAN), and Auxiliary Classifier GAN (ACGAN). GANs have significantly impacted machine learning by producing new, realistic data instances. This comparative analysis focuses on understanding their architectural nuances, performance metrics, and the quality of generated images, aiming to identify the optimal GAN model for practical applications.

Background Theory

GANs Overview: Generative Adversarial Networks (GANs) involve two networks, the generator and discriminator, competing in a zero-sum game to improve data generation quality. The generator creates data, and the discriminator evaluates its authenticity, progressively refining the output.

DCGAN: Deep Convolutional GAN (DCGAN) enhances GANs by integrating convolutional neural networks that stabilize training and improve output quality through architectural tweaks like strided convolutions and batch normalization.

WGAN: Wasserstein GAN (WGAN) modifies the GAN cost function using Wasserstein distance, which ameliorates training stability and mitigates common issues like mode collapse, facilitating smoother and more reliable training dynamics.

ACGAN: Auxiliary Classifier GAN (ACGAN) adds a classification task to the discriminator, enhancing its ability to judge and influence the generator to create diverse, category-specific images, thus improving the quality and variation of the generated data.

DATASET

The CIFAR-10 collection is utilized here, containing 60,000 32x32 resolution colored images across 10 unique categories, encompassing a broad array of objects. This compilation is organized into two sets: 50,000 images for training and 10,000 for testing purposes. Categories span a diverse group including vehicles, animals and other distinct classes with no overlap.

FID Calculation

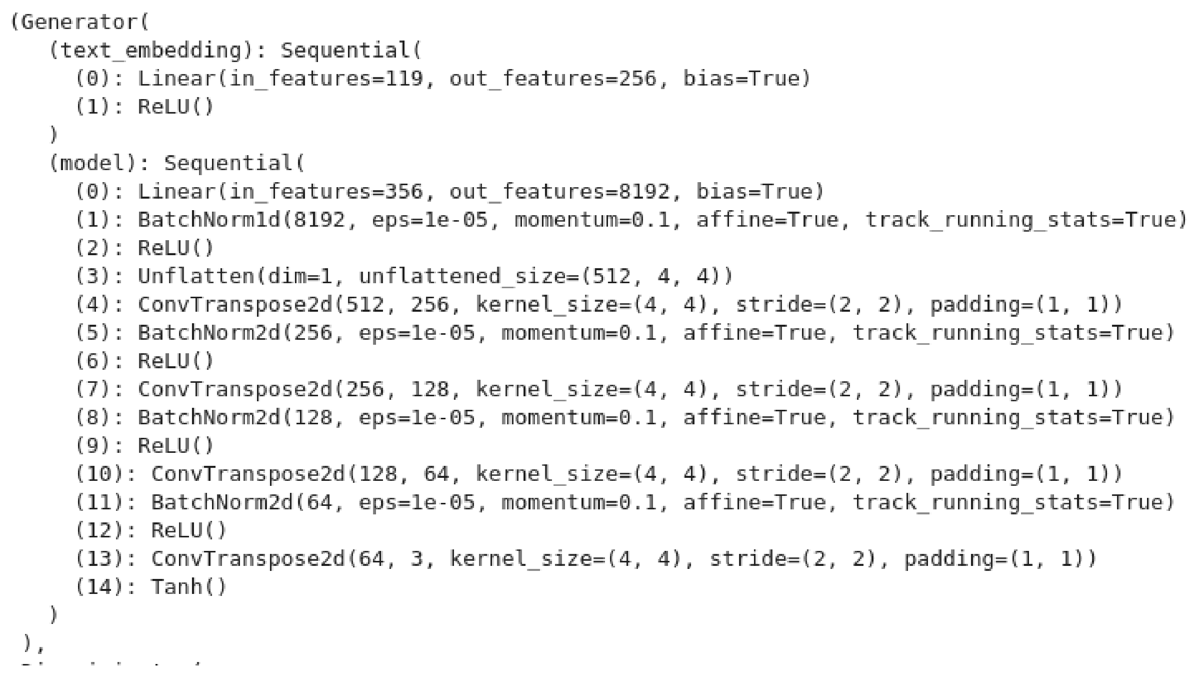
The script sets up the InceptionV3 model to extract features from images and calculate the Fréchet Inception Distance (FID) score, comparing the statistical similarity between features of real and generated images to evaluate the quality of generative models. It computes this metric using the mean and covariance of the InceptionV3 features for both image sets.

DCGAN

Model Architecture

Generator

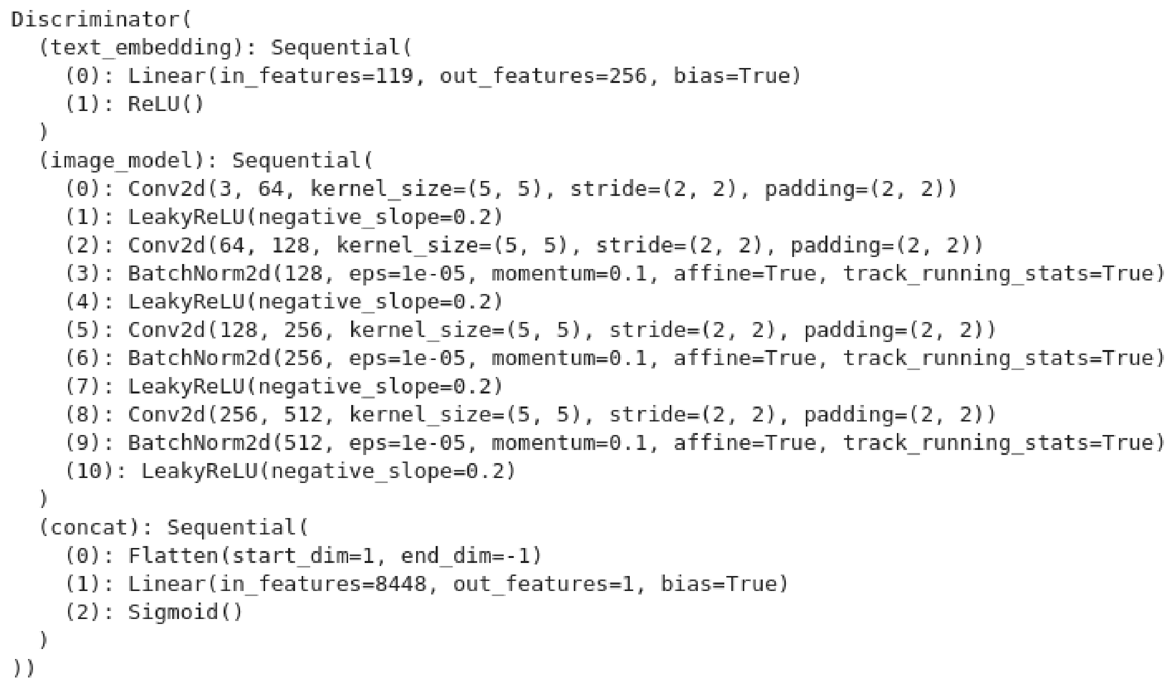
The Generator of the GAN begins with processing the text input through a linear layer followed by a ReLU activation to create a text embedding. This embedding is then merged with a noise vector generated from a Gaussian distribution. The combined vector is subsequently fed into a sequence of transpose convolutional layers, each enhanced with batch normalization and ReLU activation, to gradually upscale the feature map. This process culminates in the generation of a three-channel RGB image, with pixel values normalized between -1 and 1 using a Tanh activation function. This normalization aligns with the preprocessing of the training images, ensuring consistency in data handling.



```
(Generator(
   (text_embedding): Sequential(
     (0): Linear(in_features=119, out_features=256, bias=True)
     (1): ReLU()
   )
   (model): Sequential(
     (0): Linear(in_features=356, out_features=8192, bias=True)
     (1): BatchNorm1d(8192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
     (2): ReLU()
     (3): Unflatten(dim=1, unflattened_size=(512, 4, 4))
     (4): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
     (5): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
     (6): ReLU()
     (7): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
     (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
     (9): ReLU()
     (10): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
     (11): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
     (12): ReLU()
     (13): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
     (14): Tanh()
   )
 ),
```

Discriminator

The Discriminator receives both an image and a corresponding text description as inputs. It processes the image through a series of convolutional layers equipped with LeakyReLU activation and batch normalization to derive complex feature maps. Parallelly, it embeds the text input in a manner akin to the Generator. The embedded text is then reshaped and combined with the image features. This combined vector passes through a linear layer with a sigmoid activation to produce a probability score. This score quantifies the likelihood of the input image being authentic as per the accompanying text, effectively assessing the coherence between the generated image and its description.



```
Discriminator(
  (text_embedding): Sequential(
    (0): Linear(in_features=119, out_features=256, bias=True)
    (1): ReLU()
  )
  (image_model): Sequential(
    (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
    (1): LeakyReLU(negative_slope=0.2)
    (2): Conv2d(64, 128, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2)
    (5): Conv2d(128, 256, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
    (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2)
    (8): Conv2d(256, 512, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
    (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2)
  )
  (concat): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=8448, out_features=1, bias=True)
    (2): Sigmoid()
  )
))
```

Hyperparameters

Optimizer: Adam optimizer with a lr = 0.0002.

Batch Size: models are trained with a batch size of 128, balancing computational efficiency and model stability.

Noise Dimension: The noise vector has a dimension of 100, providing sufficient randomness to generate diverse images.

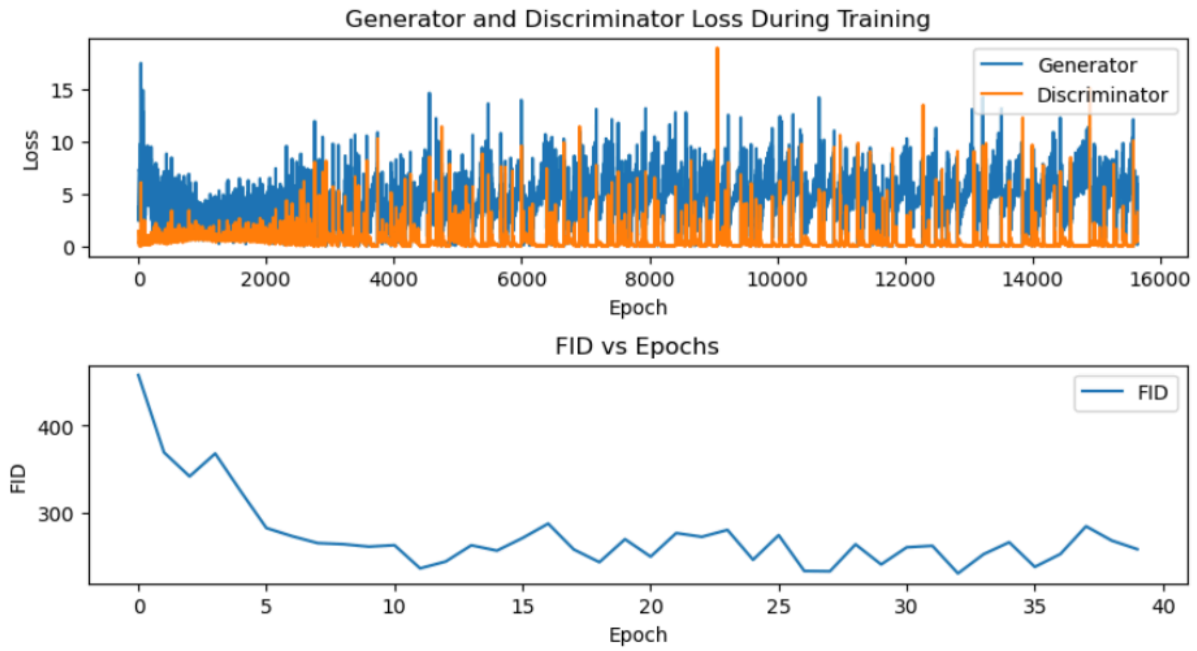
Text Embedding Dimension: Text inputs are embedded into a 256-dimensional space to capture a wide array of textual features.

Training and Results

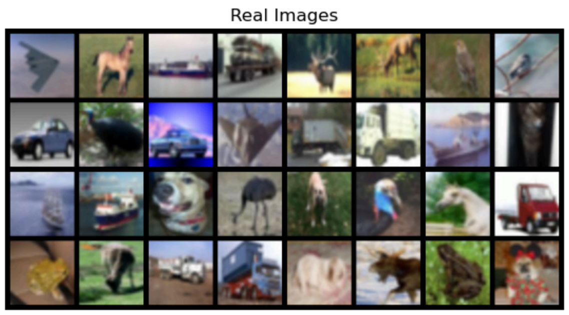
The training process is visualized through two main graphs:

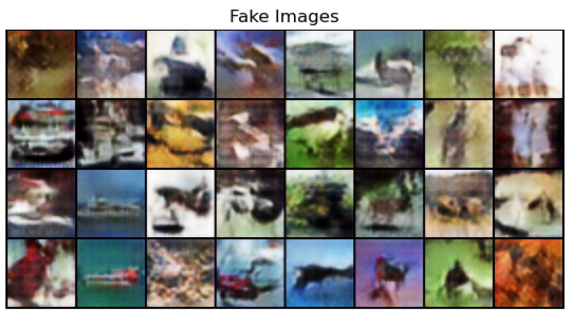
Loss Graph: Displays the generator and discriminator losses over epochs. This graph is crucial for monitoring the adversarial training dynamics, ensuring both networks learn effectively without overpowering each other.

FID (Fréchet Inception Distance) Graph: Tracks the FID score across epochs, which measures the similarity between the generated images and real images, thus providing a quantitative measure of image quality.



GENERATED IMAGES



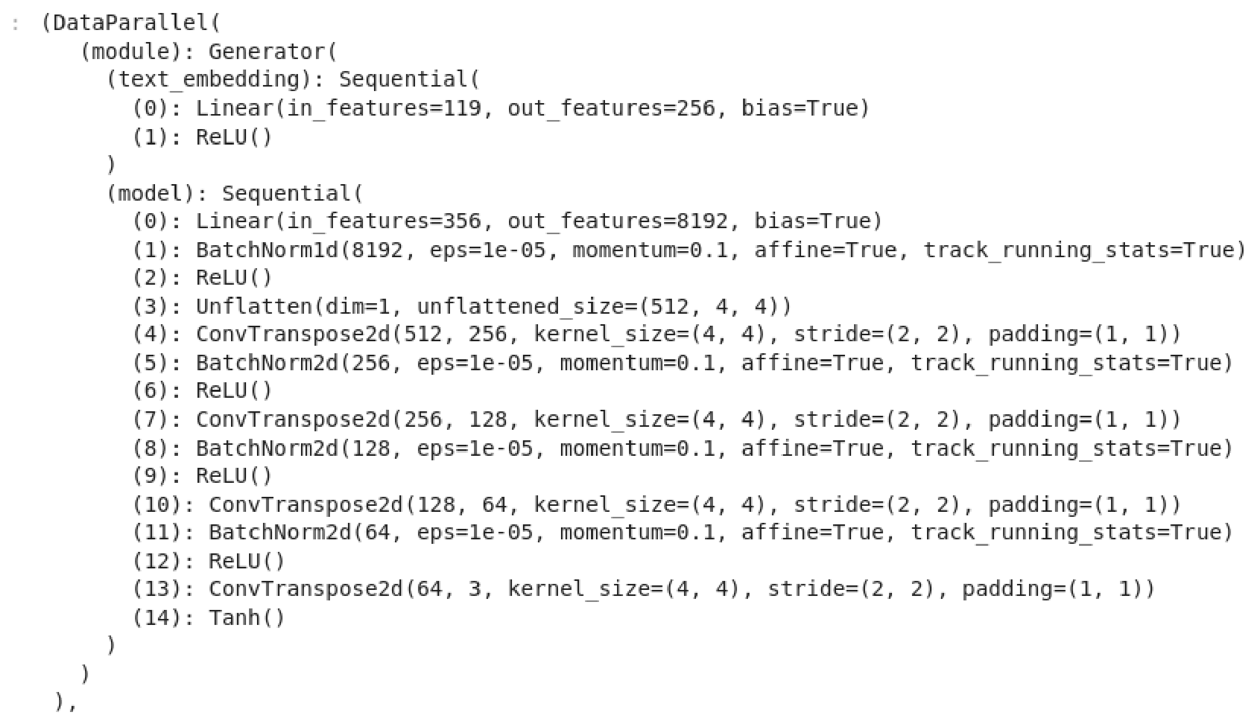


Model Architecture of WGAN

The Wasserstein GAN (WGAN) model implemented here has two primary components: the Generator and the Critic.

Generator

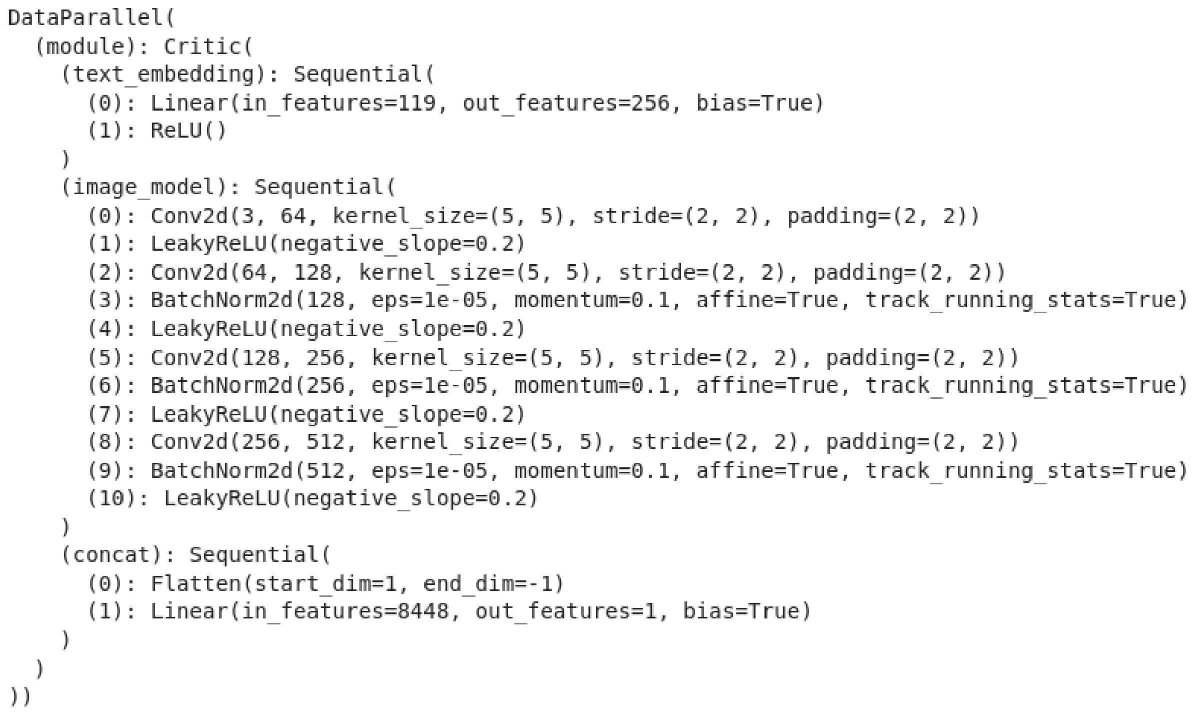
This transforms a text embedding and a noise vector into a visual representation. Initially, transforming the text input into an embedded representation using a linear layer followed by a ReLU activation function. This text embedding is concatenated with a latent noise vector. The merged vector undergoes processing via a series of transposed convolutional layers, each succeeded by batch normalization and activation with a Rectified Linear Unit function.that upsample noise and embed the text information into an image. The final layer outputs a three-channel RGB image with pixel values normalized between -1 and 1 via a Tanh activation function.



```
: (DataParallel(
     (module): Generator(
       (text_embedding): Sequential(
         (0): Linear(in_features=119, out_features=256, bias=True)
         (1): ReLU()
       )
       (model): Sequential(
         (0): Linear(in_features=356, out_features=8192, bias=True)
         (1): BatchNorm1d(8192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
         (2): ReLU()
         (3): Unflatten(dim=1, unflattened_size=(512, 4, 4))
         (4): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
         (5): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
         (6): ReLU()
         (7): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
         (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
         (9): ReLU()
         (10): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
         (11): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
         (12): ReLU()
         (13): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
         (14): Tanh()
       )
     )
   ),
```

Critic

In WGAN, the discriminator is replaced by a critic that does not output probabilities but scores representing the "realness" of the input data. The critic takes an image and text input, embeds the text using the same architecture as the generator's embedding, and processes the image through convolutional layers with LeakyReLU activation and batch normalization. These two representations are then flattened and concatenated before being passed through a linear layer to produce a scalar score.



```
DataParallel(
  (module): Critic(
    (text_embedding): Sequential(
      (0): Linear(in_features=119, out_features=256, bias=True)
      (1): ReLU()
    )
    (image_model): Sequential(
      (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
      (1): LeakyReLU(negative_slope=0.2)
      (2): Conv2d(64, 128, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
      (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (4): LeakyReLU(negative_slope=0.2)
      (5): Conv2d(128, 256, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
      (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (7): LeakyReLU(negative_slope=0.2)
      (8): Conv2d(256, 512, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
      (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (10): LeakyReLU(negative_slope=0.2)
    )
    (concat): Sequential(
      (0): Flatten(start_dim=1, end_dim=-1)
      (1): Linear(in_features=8448, out_features=1, bias=True)
    )
  )
))
```

Hyperparameters and Initialization

- **Learning Rate (Generator and Critic)**: 0.0002
- **Critic (Discriminator) Iterations per Generator Iteration**: 5
- **Batch Size**: 128
- **Gradient Penalty Weight** : 10
- **Noise Dimension**: 100
- **Epochs**: 40

Gradient Penalty

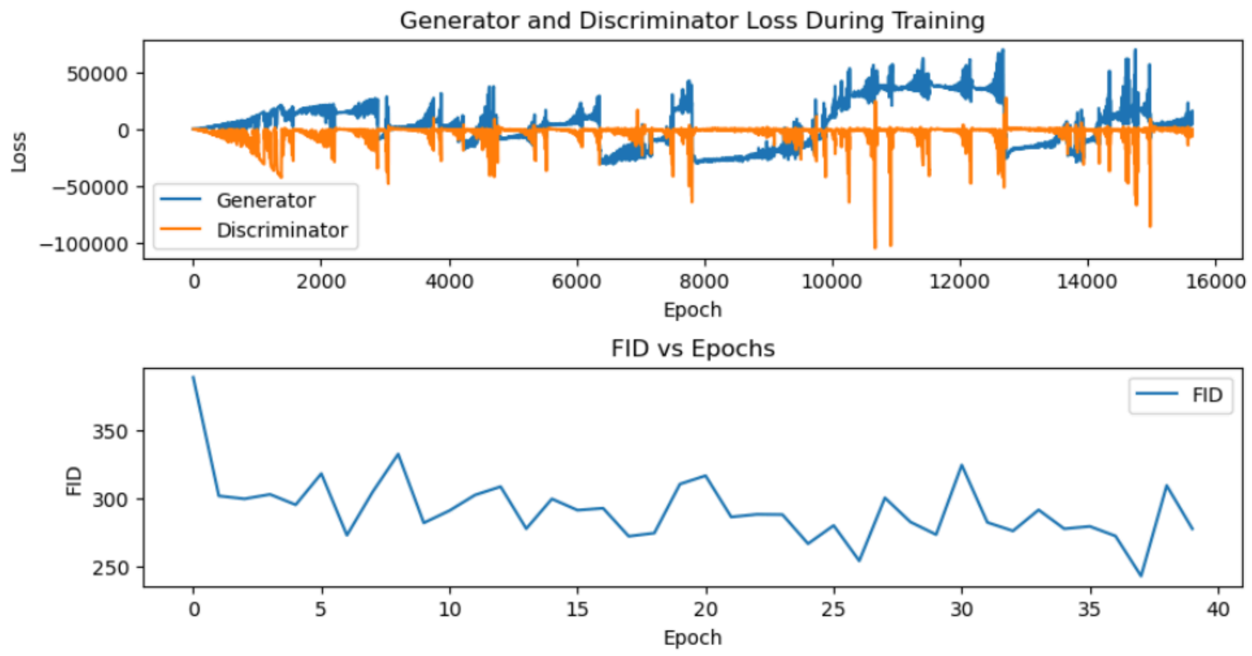
The gradient penalty is an essential feature of the WGAN, serving as a regulatory factor that maintains the Lipschitz condition for the critic. This is calculated by blending real and synthetic samples and then measuring the critic's score gradients relative to the blend. If these gradients stray from the target norm of 1, the penalty comes into effect, promoting uniformity in the critic's gradient norms.

Training and Results

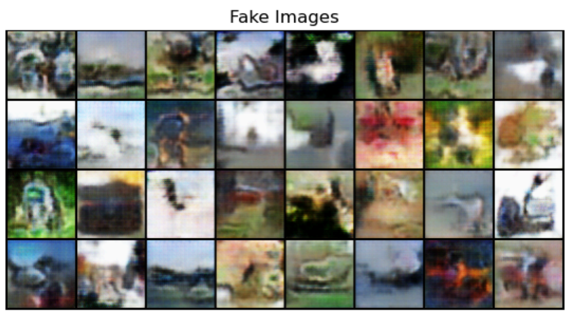
Graphs

During training, various metrics are plotted to monitor the learning process:

- **Discriminator Loss**: Shows how well the discriminator learns to differentiate real from fake images.
- **Generator Loss**: Represents the generator's performance in creating realistic images.
- **FID vs EPOCHS**



GENERATED IMAGES

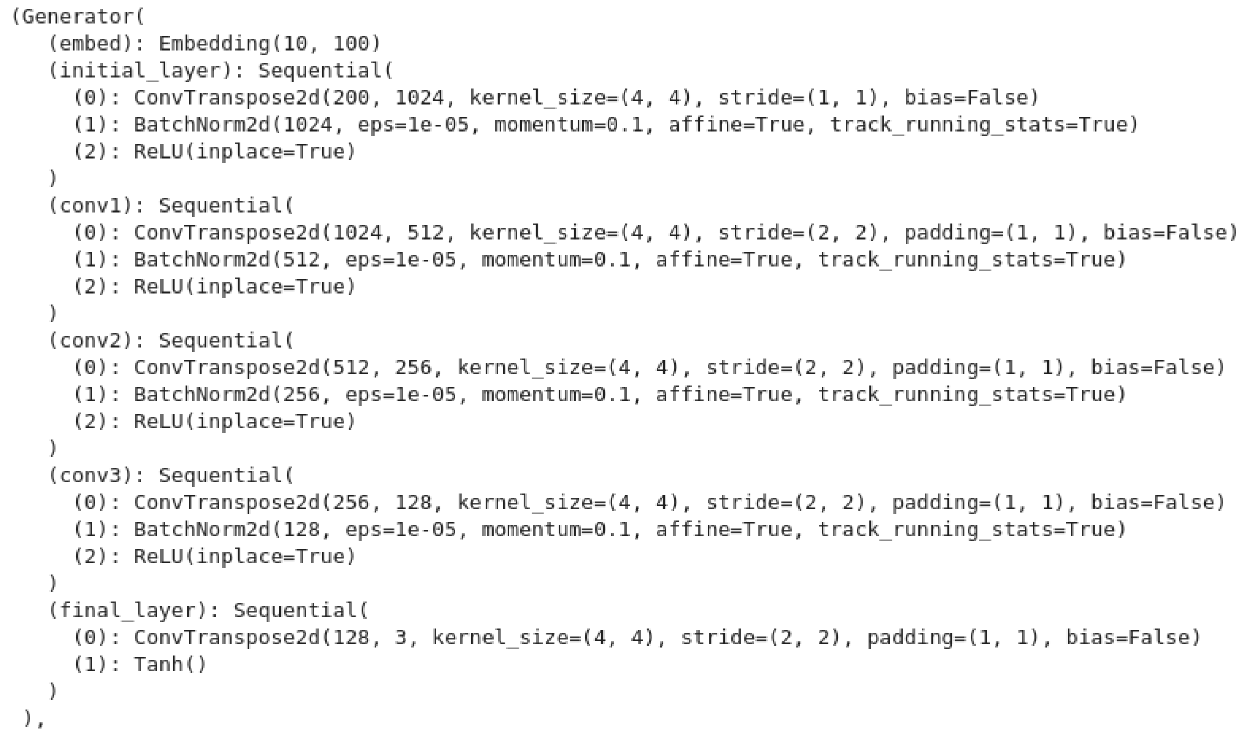


ACGAN

Model Architecture

Generator

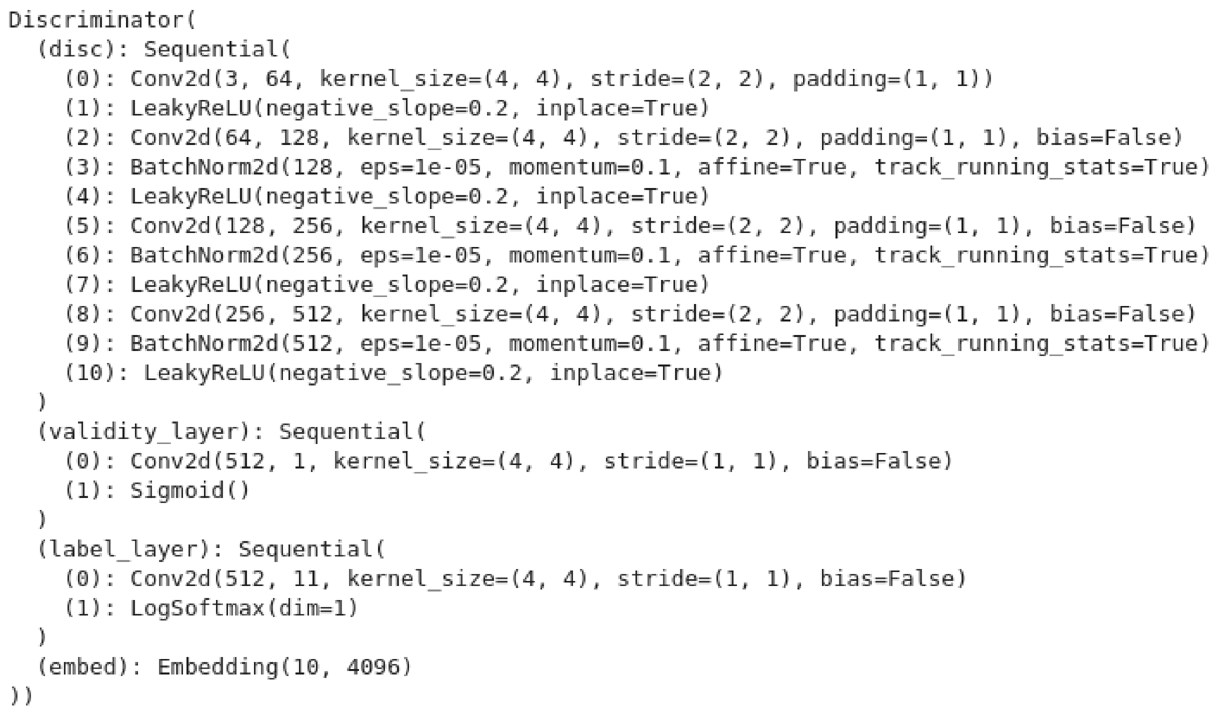
The Generator receives a noise vector, drawn from a normal distribution, along with class labels as input. These labels are then transformed into a continuous vector representation. These two inputs are then concatenated and passed through a series of transpose convolutional layers, which upscale the noise vector to generate an image of the desired size. The architecture uses batch normalization and ReLU activation functions between layers, culminating in a Tanh activation that outputs a three-channel RGB image.



```
(Generator(
   (embed): Embedding(10, 100)
   (initial_layer): Sequential(
     (0): ConvTranspose2d(200, 1024, kernel_size=(4, 4), stride=(1, 1), bias=False)
     (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
     (2): ReLU(inplace=True)
   )
   (conv1): Sequential(
     (0): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
     (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
     (2): ReLU(inplace=True)
   )
   (conv2): Sequential(
     (0): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
     (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
     (2): ReLU(inplace=True)
   )
   (conv3): Sequential(
     (0): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
     (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
     (2): ReLU(inplace=True)
   )
   (final_layer): Sequential(
     (0): ConvTranspose2d(128, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
     (1): Tanh()
   )
 ),
```

Discriminator

The Discriminator accepts an image and outputs both a validity score, representing the likelihood of the image being real rather than fake, and class predictions. The image is processed through a series of convolutional layers with LeakyReLU activation and batch normalization. The model then splits into two heads: one for the validity score using a sigmoid activation function and the other for class predictions using a log-softmax function.



```
Discriminator(
  (disc): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (validity_layer): Sequential(
    (0): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): Sigmoid()
  )
  (label_layer): Sequential(
    (0): Conv2d(512, 11, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): LogSoftmax(dim=1)
  )
  (embed): Embedding(10, 4096)
))
```

Hyperparameters

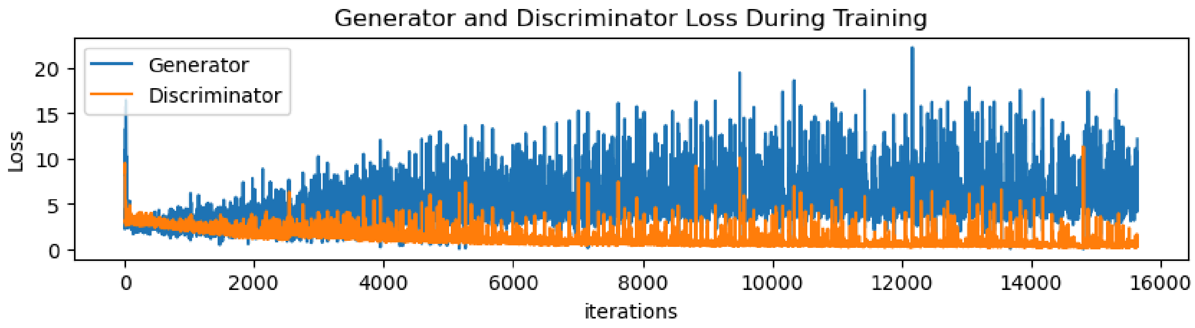
- **Number of Classes (num_classes)**: 10, corresponding to CIFAR-10 dataset classes.
- **Noise Dimension (nz)**: 100, size of the latent noise vector.
- **Generator Feature Maps (ngf)**: 64
- **Discriminator Feature Maps (ndf)**: 64.
- **Number of Channels (nc)**: 3, for RGB images.
- **Batch Size**: 128, determines the number of samples processed before the model is updated.
- **Learning Rate (lr)**: 0.0002, sets the step size at each iteration of the optimizer.
- **Beta1**: 0.5, the exponential decay rate for the first moment estimates in Adam optimizer.
- **Epochs**: 40, the number of passes through the entire dataset.
- **Image Size (img_size)**: 64

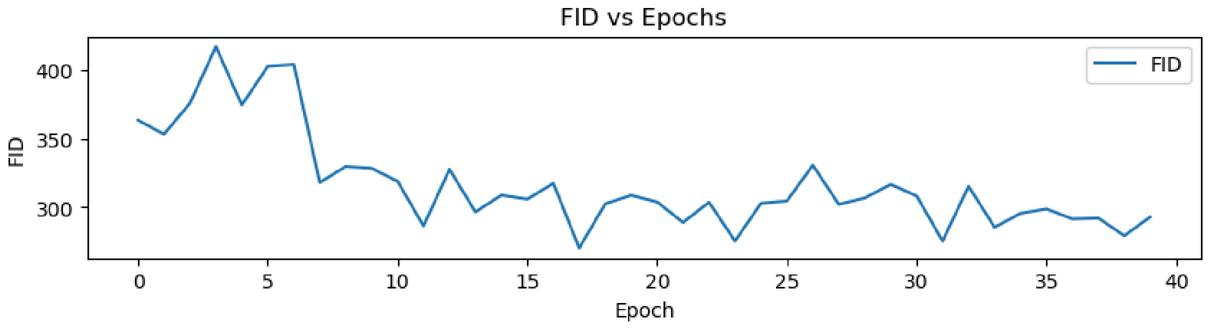
Training Dynamics and Visualization

The training process involves alternating updates to the Discriminator and Generator, with losses for both models tracked over time. The Generator loss indicates how well it is fooling the Discriminator, while the Discriminator loss indicates how well it distinguishes between real and fake images. Moreover, FID scores are computed at regular intervals to evaluate the quality of the produced images in comparison to authentic images.

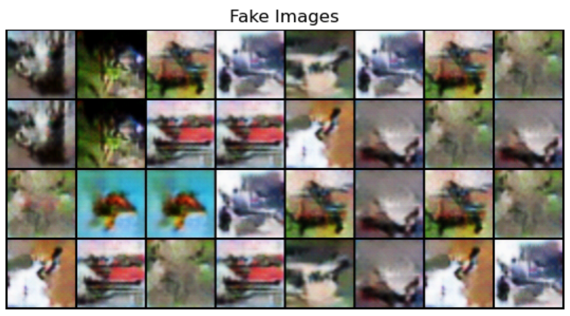
Graphs and Results

- **Generator and Discriminator Loss Graph**: This displays the adversarial losses over iterations, providing insights into the training dynamics and whether the models are converging towards a solution.
- **FID Scores Graph**: The FID scores over epochs give an objective measure of the similarity between generated and real images, with lower scores indicating better quality and diversity.



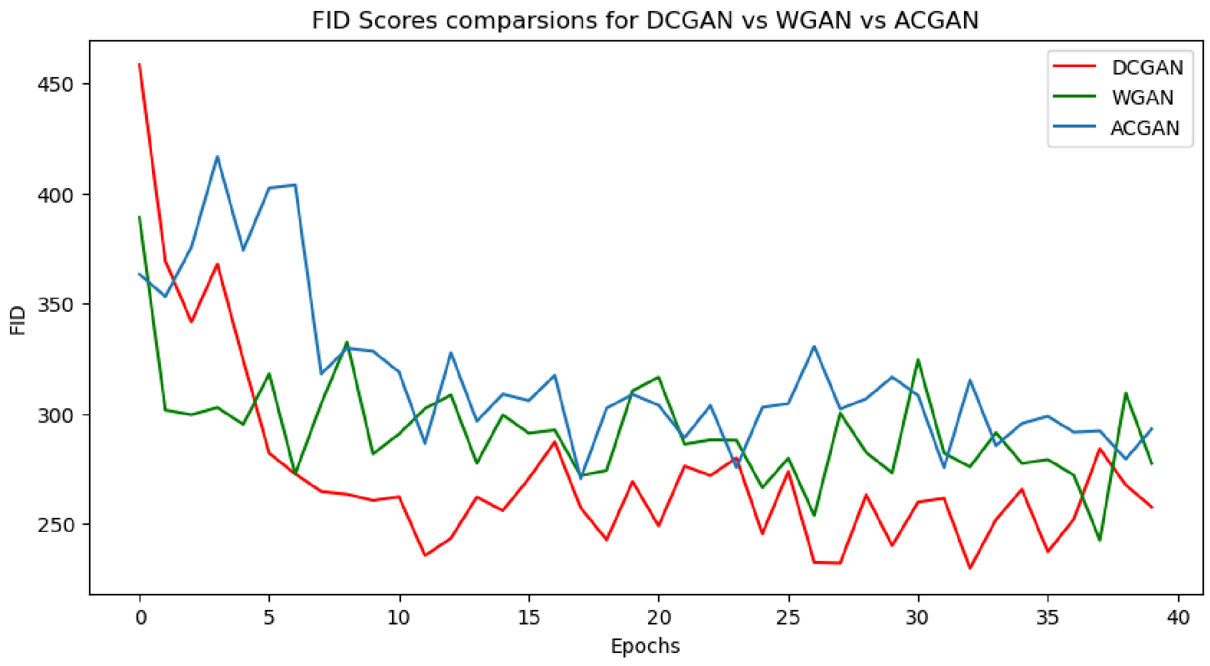


<u>**GENERATED IMAGES**</u>



COMPARISON ANALYSIS

Below graph displays FID scores for DCGAN, WGAN, and ACGAN models over 40 epochs. Initially, DCGAN has higher FID scores, indicating lower image quality, but it improves quickly. WGAN demonstrates the most stable and consistent decrease in FID, reflecting steady image quality enhancement. ACGAN performs well, particularly in the early epochs. By the 40th epoch, all models converge with similar FID scores, yet WGAN maintains a slight lead, suggesting it generates the highest quality images throughout training. Overall, WGAN seems to offer the best performance in terms of image quality stability and improvement.


FID Scores comparsions for DCGAN vs WGAN vs ACGAN
DCGAN
WGAN
ACGAN
FID
Epochs