Gitlink: https://github.com/anvithayerneni/DCGAN-WGAN-and-ACGAN-Implementations

DCGAN Configuration:

Epoch Count: 15
Learning Rate: 0.0002

Generator Specifications:

- A noise vector is transformed into a tensor through a transposed convolution layer during the process.
- It comprises three layers that upscale the image while reducing the channel count, each with a kernel size of 4, stride 2 and padding1. These layers have tensors as their inputs.
- Finally, there is an ending sequence which brings about an upscaling to an image of 64x64 pixels and channels_img determines the number of channels.
- Lastly, pixel values are normalized ranging from -1 to 1 by means of Tanh activation function applied on the final-layer output

Discriminator Architecture:

- Commence with a convolutional layer that takes an input image and generates an output tensor with dimensions (N x features_d x 32 x 32). The size of the kernel is 4x4, the stride is equal to 2 and there is one cell of padding. Here features_d is the number of features contained in this first layer.

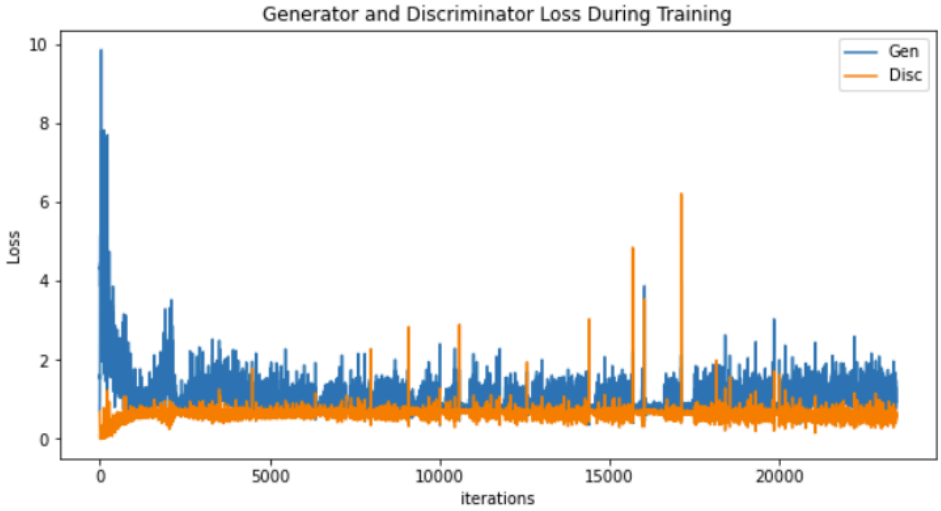
- Then a LeakyReLU function activates the output tensor obtained from the first convolution layer.

- Through the Dnet function, three convolution layers are organized into residual blocks. By taking parameters such as input channels, output channels, kernel size, stride and padding, this function sets up a sequential block. Each block contains a convolutional layer followed by a LeakyReLU activation function.

- Lastly, the final convolution layer generates a tensor with size (N x 1 x 1 x 1). This tensor gets flattened thereafter resulting in shape (N x 1).

- A sigmoid activation function processes the flattened tensor to produce a real valued scalar between zero and one. In other words, it shows how sure or doubtful the discriminator feels about whether that image can be real or not.

Loss:



WGAN Configuration:

Learning rate: 0.0002
Noise dimension: 128
Training epochs: 15
Weight clipping limit: 0.01

Generator Architecture:

1. Start with a Noise Vector: The generator begins with a random noise vector, which is a simple list of random numbers. This vector is the "seed" from which the image grows.

2. First Transposed Convolution: This vector is input into the first transposed convolutional layer. A transposed convolution is somewhat like the reverse of a normal convolution in image processing. Instead of reducing the size of the image, it increases it. The spatial size of the output is doubled compared to the input.

3. Batch Normalization and ReLU Activation:

Batch Normalization: After the transposed convolution, batch normalization is applied. This technique normalizes the output of the previous layer, reducing internal covariate shift and helping the network train faster and more effectively.

ReLU Activation: Next, a ReLU (Rectified Linear Unit) activation function is used. This function keeps only positive values and discards negative values (it replaces negative values with zero), introducing non-linearity to the process, which helps learn complex patterns.

4. Repeat the Process: Steps 2 and 3 are repeated in sequence for a total of four blocks. Each block continues to double the spatial size of its inputs through transposed convolution, and apply batch normalization and ReLU activation.

5. Final Transposed Convolution: After the sequential blocks, a final transposed convolution is used. This layer specifically uses a kernel size of 4, a stride of 2, and padding of 1. Its main role is to adjust the dimensions of the output to exactly 64 x 64 pixels, which is the size of the final image.

6. Tanh Activation Function: Finally, the tanh (hyperbolic tangent) function is applied. This activation function scales the pixel values to be between -1 and 1. It is often used in GANs because it helps the model to train more stable, as it centers the data.

By sequentially increasing the resolution through each block and applying non-linearities and normalization, the generator transforms a simple noise vector into a complex, 64x64 pixel image that can represent things like photos, depending on what the GAN has been trained on.

Discriminator Structure:

Initial Convolutional Layer:

- Kernel Size: 4
- Stride: 2
- Padding : 1
- Effect : Halves the spatial dimensions of the input and alters feature channels.

Activation :

- Type : LeakyReLU
- Negative Slope Coefficient : 0.2

Subsequent Layer Blocks (3 blocks) :

- Components per Block :
- Convolutional layer
- Instance Normalization
- LeakyReLU activation

Behavior :

- Doubles the feature count

- Halves the image size

Instance Normalization :

- Purpose : Normalizes layer outputs to stabilize training.

Final Convolutional Layer :

- Kernel Size : 4
- Stride : 2
- Padding : None
- Output : Collapses to a single scalar value representing the image's authenticity assessment.

Performance Evaluation:

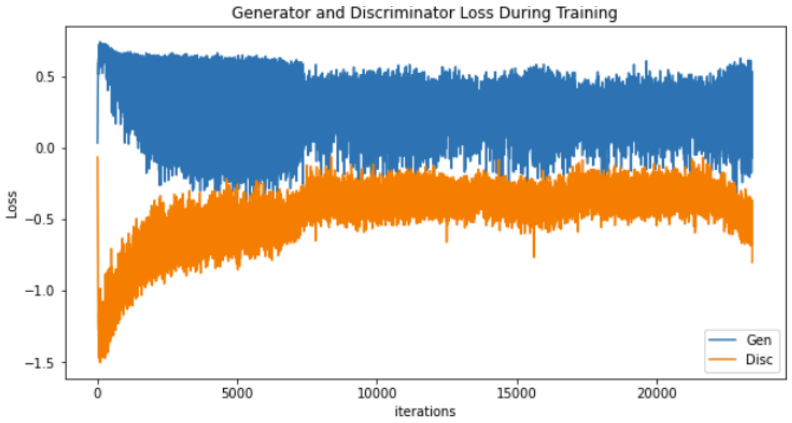
1) Inception Score Analysis:

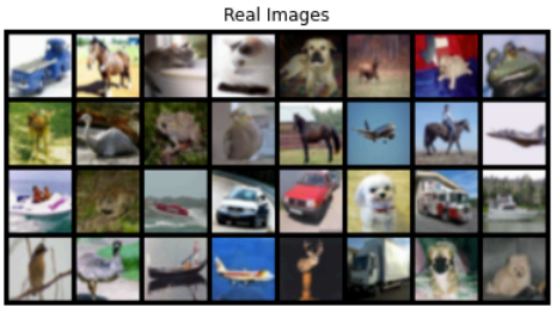
The Inception Score is a measure used to evaluate the quality and diversity of generated images. This score assesses how well an Inception model can distinguish the images and how varied they are from each other. Both DGCAN and WGAN models achieved an Inception Score of 0.3678794503211975, showing that they perform similarly in terms of image quality and diversity.

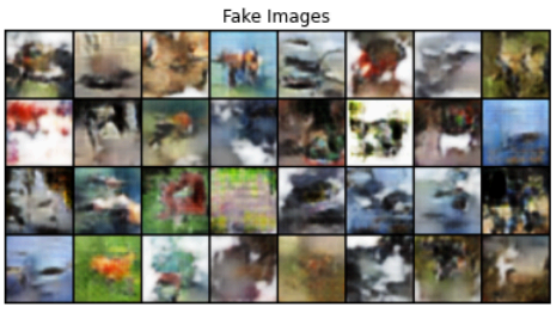
2) Frechet Inception Distance (FID) Assessment:

The FID is useful to measure the similarity between two groups of images by using an Inception network to extract features from the images.These features are then used to calculate the Fréchet distance between the Gaussian distributions representing the sets. A lower FID score indicates a closer similarity between the image sets. The calculation of the FID is computationally intensive, involving large matrix operations and dealing with extensive collections of images and a feature set of 2048 elements. The process took around 3 hours and was not completed due to the demanding nature of the computations.

Loss:







ACGAN Configuration:

The ACGAN model operates over 20 epochs and is structured as follows:

Generator Architecture:
1. Transposed Convolution Layers: There are five layers of transposed convolution. Each layer increases the number of filters and decreases the kernel size as it progresses. This helps in gradually upsampling the input representation to a larger spatial dimension.

2. Batch Normalization and Activation Functions: Following each transposed convolution layer, batch normalization is applied to stabilize the learning and improve the training dynamics. The Rectified Linear Unit (ReLU) activation function is used for all layers except the final one to introduce non-linearity, helping the model learn more complex patterns.

3. Final Layer Activation: The last transposed convolution layer uses a hyperbolic tangent (Tanh) activation function. This is typical for generating images as it normalizes the output pixel values to the range [-1, 1], which is standard for image data in many neural networks.

4. Embedding Layer: An embedding layer is used to transform a categorical label into a continuous, dense 100-dimensional vector. This helps the generator to condition the image generation based on specific input labels.

5. Noise Vector Combination: The 100-dimensional label vector is combined element-wise with a noise vector (typically sampled from a normal distribution). This combination introduces randomness into the image generation process, which is crucial for generating diverse images.

6. Input Tensor Reshaping: The combined vector from the previous step is reshaped into a four-dimensional tensor with dimensions (batch_size, 100, 1, 1). This reshaping is necessary to match the input requirements of the transposed convolution layers.

7. Image Generation: The reshaped tensor is fed into the series of transposed convolution layers. As it passes through these layers, it is transformed into a full-size image, which is the final output of the generator.

This structured approach allows the neural network to learn how to generate images that are conditioned on specific input labels, leveraging both the learned representations in the embedding layer and the variability introduced by the noise vector.

Discriminator structure:
To convert your description of a neural network architecture into more concise, bullet-point style information, here's how it can be laid out:

1. Initial Layer:
   - Number of Input Channels: 3
   - Number of Output Channels: 64
   - Convolution Parameters: Kernel size of 4x4, stride 2, padding 1

2. Second Layer:
   - Number of Input Channels: 64
   - Number of Output Channels: 128
   - Convolution Parameters: Kernel size of 4x4, stride 2, padding 1

3. Third Layer:
   - Number of Input Channels: 128
   - Number of Output Channels: 256
   - Convolution Parameters: Kernel size of 4x4, stride 2, padding 1

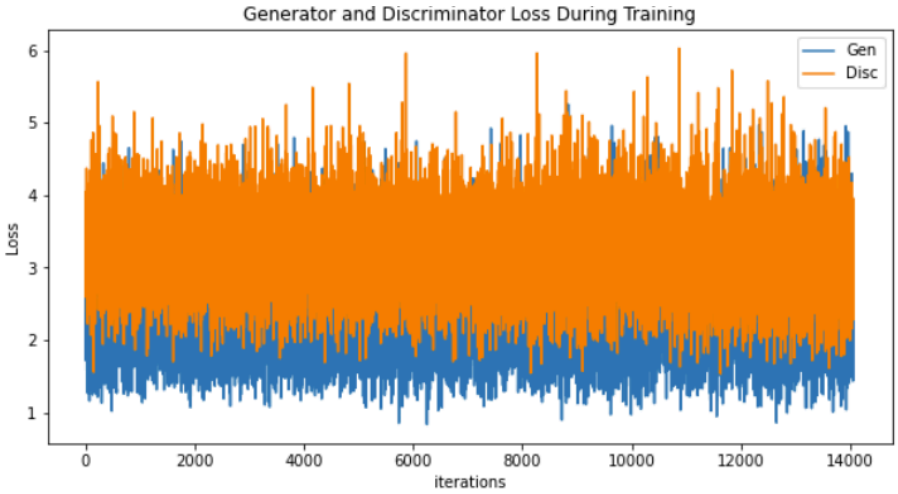
4. Fourth Layer:
   - Number of Input Channels: 256
   - Number of Output Channels: 512
   - Convolution Parameters: Kernel size of 4x4, stride 2, padding 1

5. Branching into Two Fully Connected Layers

   Branch 1: Authenticity Check
   - Convolution: 4x4 kernel, stride of 1, no padding
   - Activation Function: Sigmoid
   - Output: 1 channel, flattened into a one dimensional tensor

   Branch 2: Label Prediction
   - Convolution: 4x4 kernel, stride of 1, no padding
   - Activation Function: Log-Softmax
   - Output: 11 channels, structured into a two-dimensional tensor with 11 columns

Loss:


Generator and Discriminator Loss During Training
Gen
Disc
Loss
iterations
0
2000
4000
6000
8000
10000
12000
14000
1
2
3
4
5
6