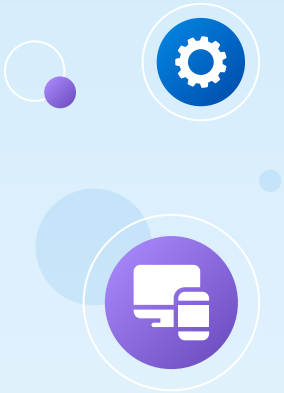
GAN-101

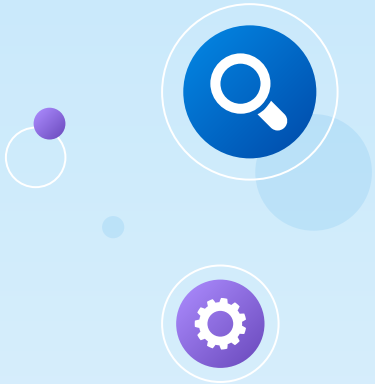
Done by:
Goh Rui Zhuo (2222329) & Toh Kien Yu (2222291)

Table of contents



01

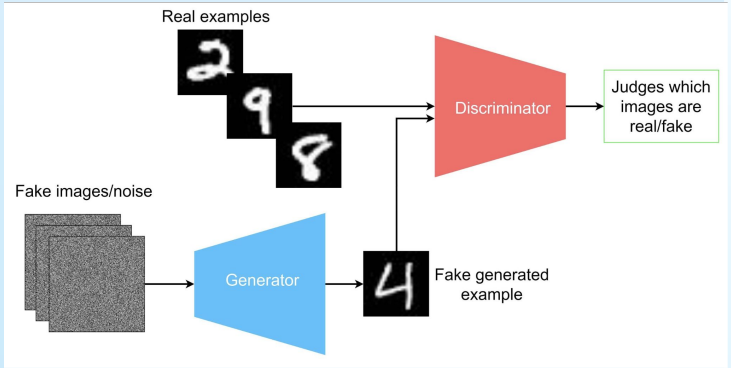
Background Research & Feature Engineering



Background Research

What is GAN?

- Gan represents Generative Adversarial Networks which is all about creating here. Generally it is harder compared to other deep learning fields.
- The main focus of GAN is to generate data be it images or videos or music from scratch and generally what does GAN consist of?
- GAN are algorithmic architectures that uses 2 neural networks, putting one against the other. This is to generate new, synthetic instances of data that can be pass as real data.
- Generally, widely use in image generation, voice generation etc.
- GAN also work with competition of two different parts



Background Research

Generator

1. For this, you can input random images, known as random noise. This random images can be anything.
2. Through the generator, it will generate a sample that hopefully ends up like the real dataset if the generator and discriminator and discriminator both are trained good enough
3. Output is sometimes referred to as latent space or a latent vector

Discriminator

1. For discriminator, input the real images from actual data that needs to be generated like CIFAR10
2. Also, you input the output of generator into the discriminator
3. Convolutional layer of discriminator is the normal convolution we are used to.
4. This are downscaled to the input that is suitable for classification.

How can we optimise as a whole?

After data is passed through both generator and discriminator, optimisation with backpropagation begins, the goal is to keep improving at a level pace for both models to be good at its role, equilibrium point is exactly when the discriminator is leaning 50% to both sides, meaning that both images could either be real or fake, where the generator model tries to minimize the probability that the discriminator will predict the generator's output as fake. On the other side, the discriminator tries to maximize the probability that it will correctly classify both real and fake image

Feature Engineering

1. Convert Data Type

```
x_train = x_train.astype('float32')
```

This line converts the data type of x_train elements to float32. It's a standard practice to use floating-point numbers in computations involving neural networks due to their capability to represent decimal values, which are crucial for the granularity of learning.

2. Scale The Data

```
x_train /= (255/2)
```

Here, the pixel values of the images (assuming `x_train` contains image data) are scaled. Usually, image data comes in the format of integers ranging from 0 to 255 (8-bit representation). This operation scales the data to a new range. Dividing by `(255/2)` scales the data to a range of [0, 2].

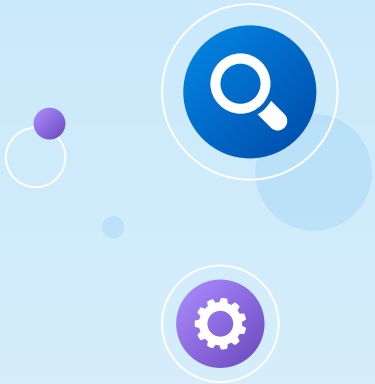
3. Shift The Data

```
x_train -= 1
```

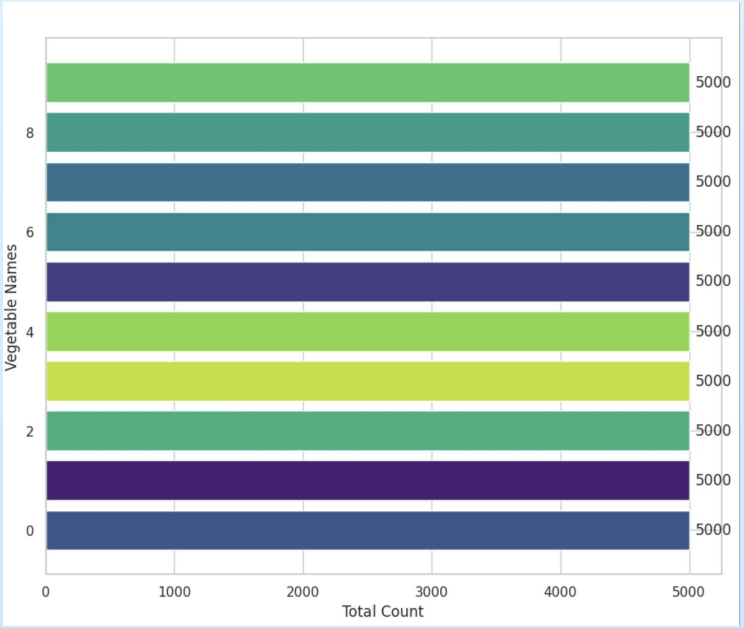
This operation shifts the data range from [0, 2] to [-1, 1]. Shifting data to be centered around zero (mean = 0) can significantly improve the convergence during training, as it ensures that the input features have similar scales.

02

Exploratory Data Analysis



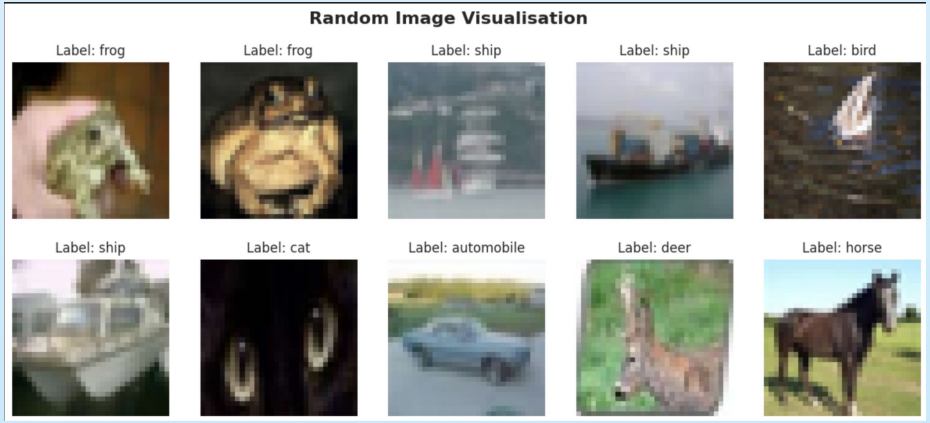
Exploratory Data Analysis



Random Image Visualisation

Process of picking random images and visualise it.

- This provide us a way to inspect our dataset
- To ensure that images are of good quality

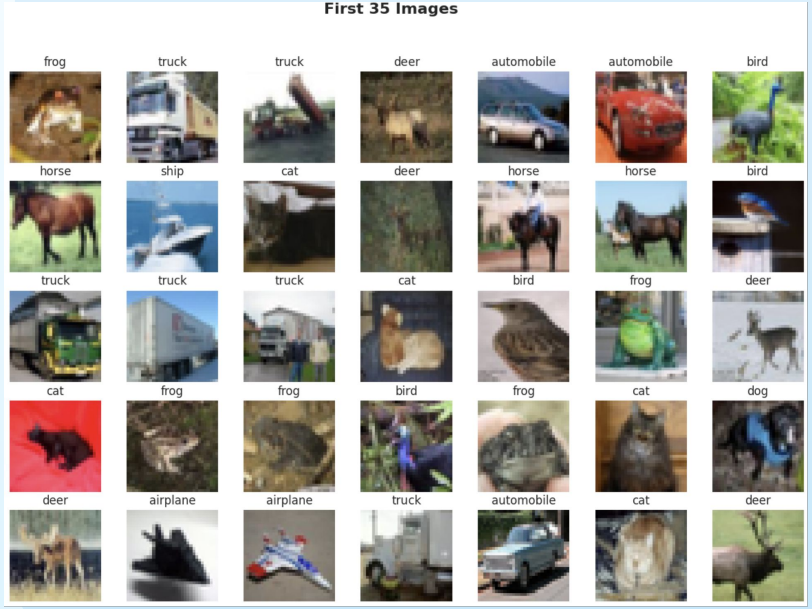


Dataset are of pretty good quality

Exploratory Data Analysis

35 Image visualisation

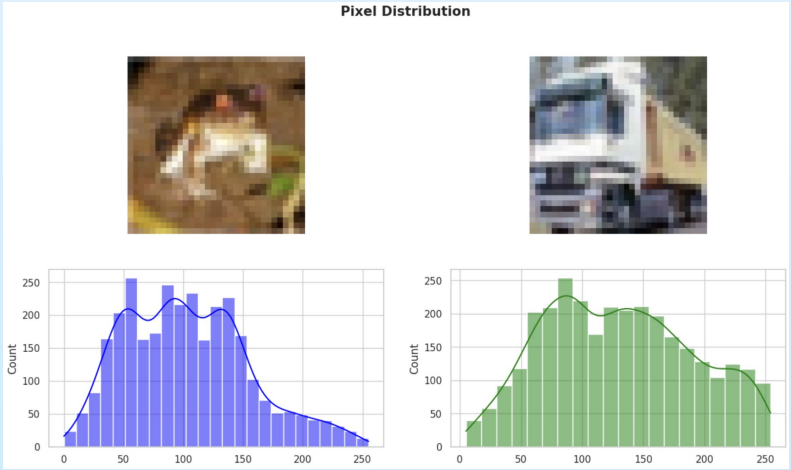
Similar to above random visualisation, this is to
- Ensure the images are of good quality
- Check first few images here



Pixel Distribution

To determine which methods to use for scaling of dataset

1. Check for the distribution of the first to images



Things Observed

First images
- Significant variance in colour intensity
- Contains peaks at 150
- Wide variation of colour intensity

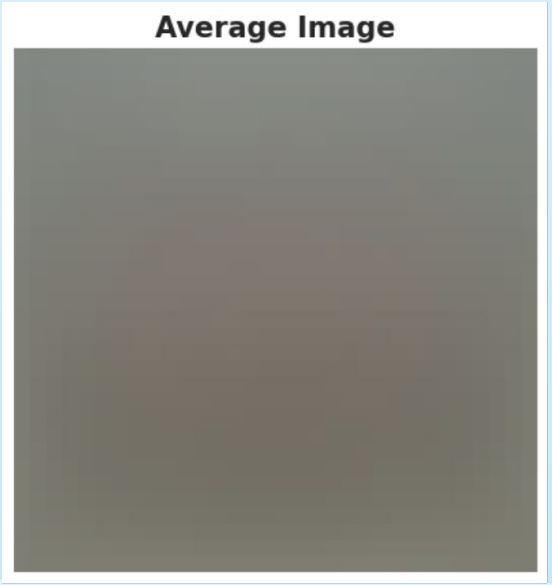
Second Images
- Smaller number of green beans on a light background
- Histogram for this image shows a significant peak around the 200 intensity level, which corresponds to the light background

Exploratory Data Analysis

Mean Pixel Distribution

Checking the mean pixel and standard deviation.

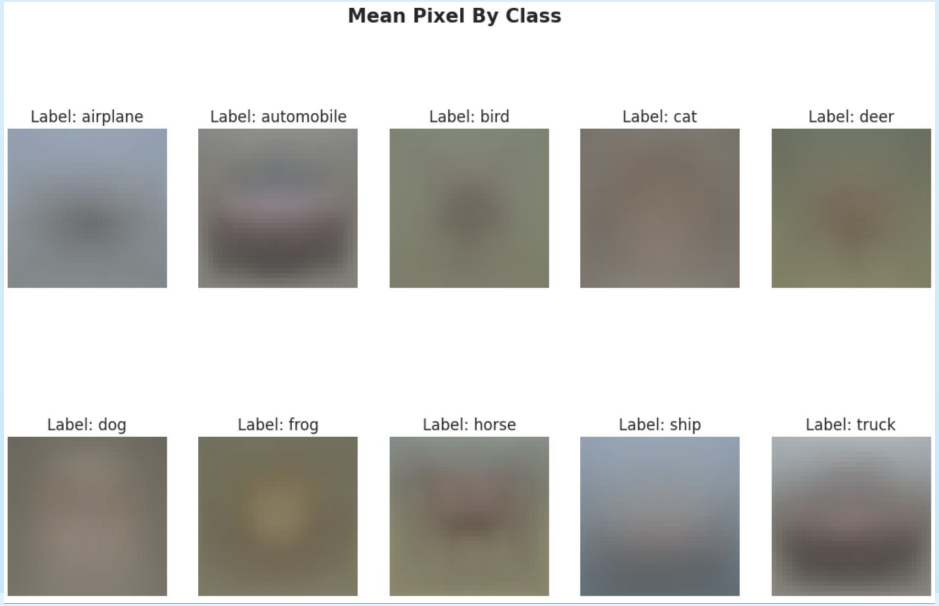
Whole Dataset



Things observed

Overall, the mean image is not as clear here.

By Class



03

Application Of GAN + GAN Architecture

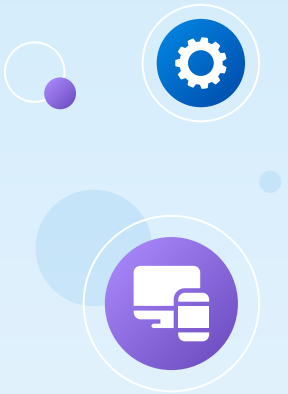
Application Of GAN

01 DCGAN

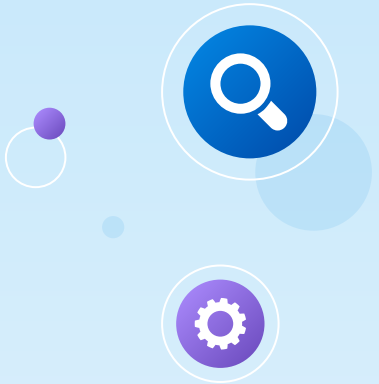
02 CGAN

03 WGAN

04 SAGAN



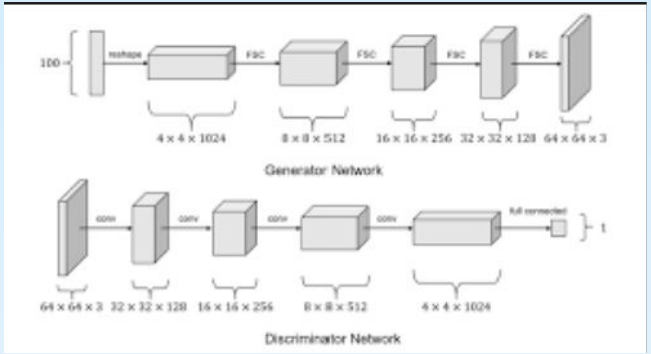
DCGAN



DCGAN

Key Ideas

- Uses convolutional and convolutional-transpose layers in the generator and discriminator, respectively
- Discriminator consists of strided convolution layers, batch normalization layers, and LeakyRelu as activation function
- Propose certain arc guidelines to stabalise the training of GAN
- The changes is generally on the model architecture itself rather than the training step here
- One of the most early methods
- Weights can be use as random noise too



DCGAN (Model 1)

First Model

Generator

Key Ideas

1. General approach is to create a generator that utilises upsample to see if results in a good model

Model Arch

1. **Input and Reshape Layers**:

- **Dense Layer**:
  - **Units**: 256 * 4 * 4
  - **Input**: `latent_dim` (the dimension of the latent space).
- **LeakyReLU**:
  - **Alpha**: 0.2
- **Reshape**:
  - **New Shape**: (4, 4, 256)

2. **Convolutional Layers and Upsampling**:

- **UpSampling2D** (3 times):
- **Conv2D and LeakyReLU Pairs** (3 times):
  - **Filters**: 128 (for each Conv2D layer)
  - **Kernel Size**: (3, 3) (for each Conv2D layer)
  - **Padding**: 'same' (keeps the spatial dimensions constant after the operation)
  - **Activation (LeakyReLU)**: Alpha set to 0.2

3. **Output Layer**:

- **Conv2D**:
  - **Filters**: 3 (representing RGB channels)
  - **Kernel Size**: (3, 3)
  - **Activation**: 'tanh'
  - **Padding**: 'same'

Discriminator

Key Ideas

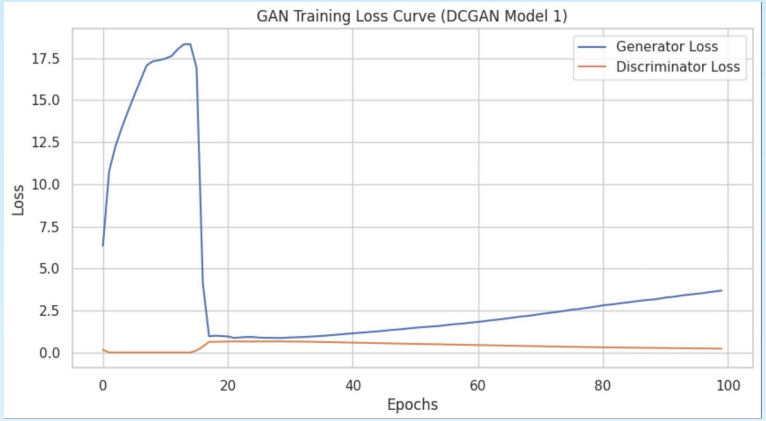
1. **Input Layer**:

- **Conv2D**:
  - **Filters**: 64
  - **Kernel Size**: (3, 3)
  - **Padding**: 'same'
  - **Input Shape**: `in_shape` (the shape of the input images).
- **LeakyReLU**:
  - **Alpha**: 0.2, allows a small gradient when the unit is inactive, promoting gradient flow during backpropagation.

2. **Convolutional Layers**:
   - Sequentially adding sets of Conv2D and LeakyReLU layers, increasing the number of filters and reducing the spatial dimensions of the feature maps using strides.
   - **First Set**:
     - **Conv2D**:
       - **Filters**: 128
       - **Kernel Size**: (3, 3)
       - **Strides**: (2, 2), reduces the spatial dimensions by half.
       - **Padding**: 'same'
     - **LeakyReLU**: Alpha set to 0.2
   - **Second Set**:
     - **Conv2D**:
       - **Filters**: 128
       - **Kernel Size**: (3, 3)
       - **Strides**: (2, 2)
       - **Padding**: 'same'
     - **LeakyReLU**: Alpha set to 0.2
   - **Third Set**:
     - **Conv2D**:
       - **Filters**: 256
       - **Kernel Size**: (3, 3)
       - **Strides**: (2, 2)
       - **Padding**: 'same'
     - **LeakyReLU**: Alpha set to 0.2
3. **Flattening and Dense Output Layer**:
   - **Flatten**: Converts the 3D feature maps to a 1D feature vector.
   - **Dropout**:
     - **Rate**: 0.4, randomly sets a fraction of input units to 0, helping to prevent overfitting.
   - **Dense**:
     - **Units**: 1, the output layer with a single neuron for binary classification.
     - **Activation**: 'sigmoid'

DCGAN (Model 1)

	Model	FID	KID	d_loss	g_loss
0	Baseline DCGAN 1	53.235623	0.039666	0.229398	3.687913



Generally, we can tell most of the animal images like deer, dog but cars and ships images are not produced

Discriminator loss

- For the discriminator loss, it starts relatively low and rises quickly and continues to increase steadily as the epochs increases

Generator loss

- For the generator loss, it increases with a spike and drops and proceeded to increase slowly

- FID Score of 53
- KID Score is 0.04

Both are not too good, let us see if we can improve

DCGAN (Model 2)

Second Model

Generator

Key Difference

1. Change upsamling to conv transpose here
2. Reducing the number of layers

Key Ideas

1. **Input and Dense Layer**:
   - **Dense Layer**:
     - **Units**: The layer has 128 * 8 * 8 units.
     - **Input**: Takes an input of dimension `latent_dim` representing the latent space.
     - **Kernel Initializer**: Utilizes `RandomNormal` with a standard deviation of `GAUSS_SD` for weight initialization.
2. **Reshape Layer**:
   - **New Shape**: Reshapes the output of the Dense layer into an 8x8x128 tensor
3. **Normalization and Activation**:
   - **BatchNormalization**:
     - **Momentum**: Utilizes a momentum factor of `MOMENTUM`.
   - **LeakyReLU**:
     - **Alpha**: Set to `ALPHA`.
4. **Transposed Convolution Layers**:
   - **Conv2DTranspose (First Layer)**:
     - **Filters**: 128
     - **Kernel Size**: 5x5
     - **Strides**: 2
     - **Padding**: 'same'
   - **Conv2DTranspose (Second Layer)**:
     - **Filters**: 3 (corresponding to RGB channels)
     - **Kernel Size**: 5x5
     - **Strides**: 2
     - **Padding**: 'same'
     - **Activation**: 'tanh'

Discriminator

Key Ideas

1. Addition of random weight
2. Adding of batch normlisation

Model Ideas

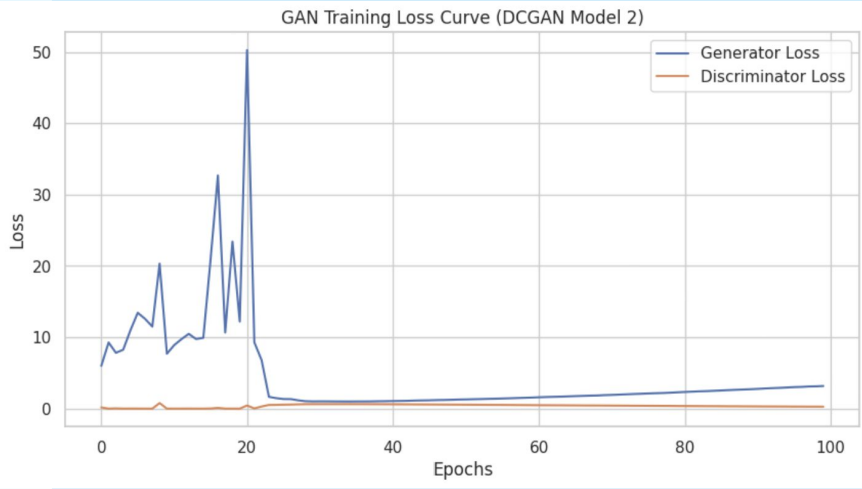
1 **Input and Convolutional Layers**:

- **Conv2D** (First Layer):
  - **Filters**: 64
  - **Kernel Size**: (5, 5), captures the spatial features.
  - **Strides**: 2
  - **Padding**: 'same'
  - **Kernel Initializer**: `RandomNormal(stddev=GAUSS_SD)`
  - **Input Shape**: `in_shape` (the shape of the input images).
- **LeakyReLU**:
  - **Alpha**: `ALPHA`
- **Conv2D** (Second Layer):
  - **Filters**: 128
  - **Kernel Size**: (5, 5)
  - **Strides**: 2
  - **Padding**: 'same'
- **BatchNormalization**:
  - **Momentum**: `MOMENTUM`
- **LeakyReLU**:
  - **Alpha**: `ALPHA`

3. **Flattening and Output Layer**:
   - **Flatten**:
   - **Dropout**:
     - **Rate**: 0.4,
   - **Dense**:
     - **Units**: 1, the output layer for binary classification.
     - **Activation**: 'sigmoid'

DCGAN (Model 2)

	Model	FID	KID	d_loss	g_loss
0	Baseline DCGAN 1	53.235623	0.039666	0.229398	3.687913
1	Baseline DCGAN 2	54.338184	0.041610	0.275021	3.177531



Similar to above, we can tell most of the animal images like deer, dog but cars and ships images are not produced

Discriminator loss

- For the discriminator loss, it starts relatively low and mains around there

Generator loss

- For the generator loss, it increases with a spike and drops and proceeded to increase slowly

- FID Score of 54
- KID Score is 0.04

It did not improve from the previous model, let us keep trying

DCGAN (Model 3)

Third Model

Key Ideas

1. Usage of `LeakyReLU` instead of `PReLU`.
2. Introduction of `UpSampling2D` layers for upsampling.

Model Ideas

1. **Input and Dense Layer**:
   - **Dense Layer**:
     - **Units**:128 * 8 * 8 units
     - **Input**: Accepts an input of dimension `latent_dim`
2. **Activation and Reshape Layer**:
   - **LeakyReLU**:
     - **Alpha**: 0.2
   - **Reshape**:
     - **New Shape**: Reshapes the output of the Dense layer into an 8x8x128 t
3. **Upsampling and Convolution**:
   - **Upsampling to 16x16**:
     - **UpSampling2D**: scaling from 8x8 to 16x16.
     - **Conv2D**:
       - **Filters**: 128
       - **Kernel Size**: (3,3), used to refine the upscaled features.
       - **Padding**: 'same', ensures the spatial dimensions remain constant.
     - **LeakyReLU**:
       - **Alpha**: 0.2
   - **Upsampling to 32x32**:
     - **UpSampling2D**: scaling from 16x16 to 32x32.
     - **Conv2D**:
       - **Filters**: 3 (corresponding to RGB channels, final layer to generate the image)
       - **Kernel Size**: (3,3)
       - **Padding**: 'same'
       - **Activation**: 'tanh', in the range [-1, 1].

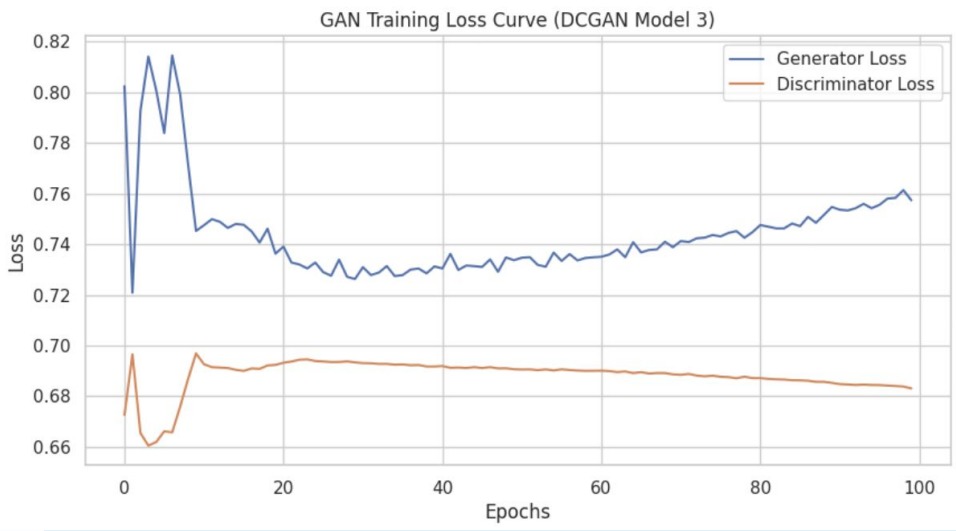
Discriminator

Key Ideas

1. **Input Layer and Convolutional Layers**:
   - **Conv2D (First Layer)**:
     - **Filters**: 64
     - **Kernel Size**: (3,3), captures spatial features.
     - **Strides**: (2,2),
     - **Padding**: 'same'
     - **Input Shape**: `in_shape` (the shape of the input images).
   - **LeakyReLU**:
     - **Alpha**: 0.2
   - **Conv2D (Second Layer)**:
     - **Filters**: 128
     - **Kernel Size**: (3,3)
     - **Strides**: (2,2), reduces the spatial dimensions to half
     - **Padding**: 'same'
   - **LeakyReLU**:
     - **Alpha**: 0.2
2. **Flattening and Output Layer**:
   - **Flatten**: Transforms the 2D feature maps into a 1D feature vector, preparing it for the dense output layer.
   - **Dense**:
     - **Units**: 1, the output layer for binary classification.
     - **Activation**: 'sigmoid'

DCGAN (Model 3)

	Model	FID	KID	d_loss	g_loss
0	Baseline DCGAN 1	53.235623	0.039666	0.229398	3.687913
1	Baseline DCGAN 2	54.338184	0.041610	0.275021	3.177531
2	Baseline DCGAN 3	47.789308	0.030753	0.683071	0.757343



Generally, we can tell most of the animal images like deer, dog and cars and ships are able to be seen

Discriminator loss

- For the discriminator loss, it starts relatively low and rises quickly and continues to increase steadily as the epochs increases

Generator loss

- For the generator loss, it increases with a spike and drops and proceeded to increase slowly

- FID Score of 47
- KID Score is 0.03

It improved further compared to the previous two models

DCGAN (Model 4)

Fourth Model

Generator

Key Difference

1. Change of relu function to prelu

Key Ideas

1. **Input and Dense Layer**:
   - **Dense Layer**:
     - **Units**: The layer has 4 * 4 * 512 units
     - **Input**: Takes an input of dimension `latent_dim` representing the latent space
     - **Kernel Initializer**: Utilizes `RandomNormal` with a standard deviation of 0.02
2. **Reshape Layer**:
   - **New Shape**: Reshapes the output of the Dense layer into an 4x4x512 tensor.
3. **Normalization and Activation**:
   - **BatchNormalization**:
   - **PreLu**:
4. **Transposed Convolution Layers**:
   - **Conv2DTranspose (First Layer)**:
     - **Filters**: 256
     - **Kernel Size**: 5x5
     - **Strides**: 2
     - **Padding**: 'same'
   - **Conv2DTranspose (Second Layer)**:
     - **Filters**: 128
     - **Kernel Size**: 5x5
     - **Strides**: 2
     - **Padding**: 'same'
   - **Conv2DTranspose (Third Layer)**:
     - **Filters**: 3 (corresponding to RGB channels)
     - **Kernel Size**: 5x5
     - **Strides**: 2
     - **Padding**: 'same'
     - **Activation**: 'tanh'

Discriminator

Key Changes

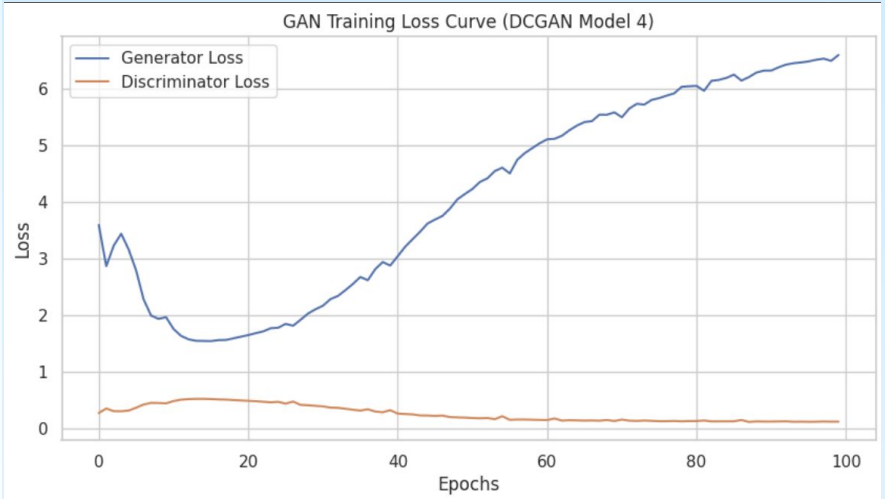
1. Use of Spectral Normalisation
   - **Roles**: Usual normalisation will stop if the derivatives of the discriminator turn out to be 0
   - Prevent this by normalising the weight matrix instead of normalising the function

Key Ideas

1. **Model Initialization**:
   - **Weights Initialization**: Uses `RandomNormal(mean=0, stddev=0.02)` for initializing the weights of the layers.
   - **Dropout Rate**: 0.3
2. **Input Layer**:
   - **Input Shape**: (32, 32, 3), accepts RGB images of size 32x32 pixels.
3. **Convolutional and Regularization Layers**:
   - **Conv2D with SpectralNormalization (Four Layers)**:
     - **Filters**: Increasing number of filters with each layer (64, 128, 256, 512)
     - **Kernel Size**: (4, 4) for all layers.
     - **Strides**: (2, 2)
     - **Padding**: 'same'
     - **Kernel Initializer**: `weights_init`
   - **LeakyReLU (after each Conv2D)**:
     - **Alpha**: 0.2
   - **Dropout (after each LeakyReLU)**:
     - **Rate**: `dropout_rate`, added after each LeakyReLU layer for regularization.
4. **Flattening and Output Layer**:
   - **Flatten**: Transforms the 3D feature maps into a 1D feature vector, preparing it for the dense output layer.
   - **Dense**:
     - **Units**: 1, the output layer for binary classification.
     - **Activation**: 'sigmoid'

DCGAN (Model 4)

	Model	FID	KID	d_loss	g_loss
0	Baseline DCGAN 1	53.235623	0.039666	0.229398	3.687913
1	Baseline DCGAN 2	54.338184	0.041610	0.275021	3.177531
2	Baseline DCGAN 3	47.789308	0.030753	0.683071	0.757343
3	Baseline DCGAN 4	61.771299	0.047135	0.123955	6.587285



Generally, we can tell most of the animal images like deer, dog but cars and ships images are not produced

Discriminator loss

- For the discriminator loss, it starts relatively low and rises quickly and continues to increase steadily as the epochs increases

Generator loss

- For the generator loss, it increases with a spike and drops and proceeded to increase slowly

- FID Score of 62
- KID Score is 0.047

It is not too good, let us see if we can improve

DCGAN (Model 5)

Fifth Model

Generator

Key Difference

1. Addition of separate upsample block to set the use of batch normlisation, activation depending on the situation

Key Ideas

1. **Input and Dense Layer**:
   - **Input**: 128
   - **Dense Layer**:
     - **Units**: 2 * 2 * 512
     - **Use Bias**: False
2. **Activation and Reshape Layer**:
   - **LeakyReLU**:
     - **Alpha**: 0.2
   - **Reshape**:
     - **New Shape**: Reshapes the output of the Dense layer into a 2x2x512 tensor, preparing it for the convolutional operations.
3. **Upsampling Blocks**:
   - The model uses a custom `upsample_block` function for upsampling, which includes:
     - **Upsampling**:
       - **Size**: Doubles the dimensions of the feature maps.
     - **Conv2D**:
       - **Filters**:(128, 64, 64, and 3 for the final block).
       - **Kernel Size**: (3, 3) for all blocks.
       - **Strides**: (1, 1) for all blocks.
       - **Padding**: 'same'
       - **Use Bias**: False.
     - **BatchNormalization**: Applied if `use_bn` is True.
     - **Activation**: LeakyReLU with alpha 0.2 for intermediate blocks, and tanh for the final block.
     - **Dropout**: Applied if `use_dropout` is True, with a rate of `drop_value`.

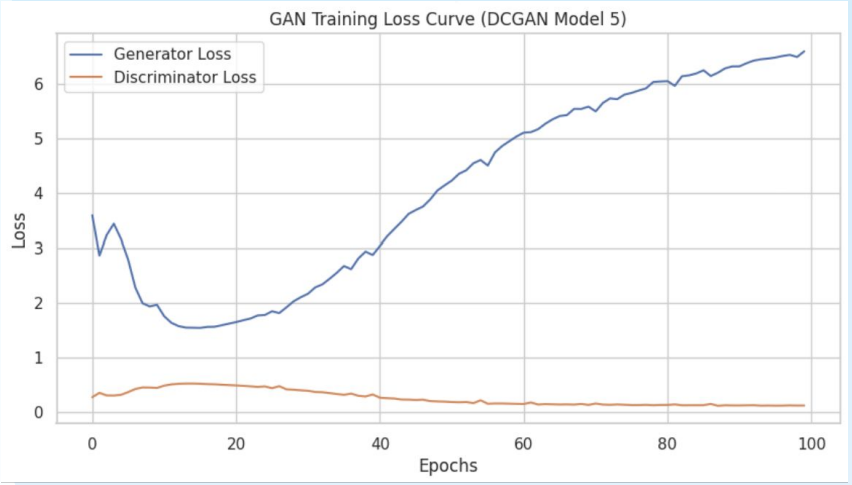
Discriminator

Key Ideas

1. **Input Layer and Convolutional Layers**:
   - **Conv2D (First Layer)**:
     - **Filters**: 64
     - **Kernel Size**: (3,3), captures spatial features.
     - **Strides**: (2,2),
     - **Padding**: 'same'
     - **Input Shape**: `in_shape` (the shape of the input images).
   - **LeakyReLU**:
     - **Alpha**: 0.2
   - **Conv2D (Second Layer)**:
     - **Filters**: 128
     - **Kernel Size**: (3,3)
     - **Strides**: (2,2), reduces the spatial dimensions to half
     - **Padding**: 'same'
   - **LeakyReLU**:
     - **Alpha**: 0.2
2. **Flattening and Output Layer**:
   - **Flatten**: Transforms the 2D feature maps into a 1D feature vector, preparing it for the dense output layer.
   - **Dense**:
     - **Units**: 1, the output layer for binary classification.
     - **Activation**: 'sigmoid'

DCGAN (Model 5)

	Model	FID	KID	d_loss	g_loss
0	Baseline DCGAN 1	53.235623	0.039666	0.229398	3.687913
1	Baseline DCGAN 2	54.338184	0.041610	0.275021	3.177531
2	Baseline DCGAN 3	47.789308	0.030753	0.683071	0.757343
3	Baseline DCGAN 4	61.771299	0.047135	0.123955	6.587285
4	Baseline DCGAN 5	62.499499	0.047779	0.123955	6.587285



Generally, we can tell most of the animal images like deer, dog but cars and ships images are not produced

Discriminator loss

- For the discriminator loss, it starts relatively low and rises quickly and continues to increase steadily as the epochs increases

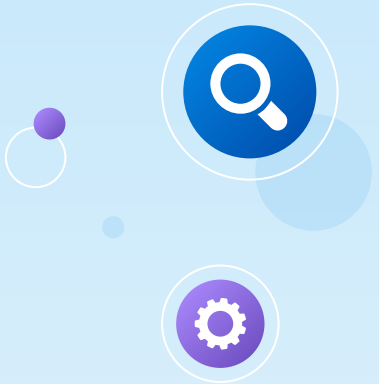
Generator loss

- For the generator loss, it increases with a spike and drops and proceeded to increase slowly

- FID Score of 62
- KID Score is 0.04

Both are not too good, showing this did not work

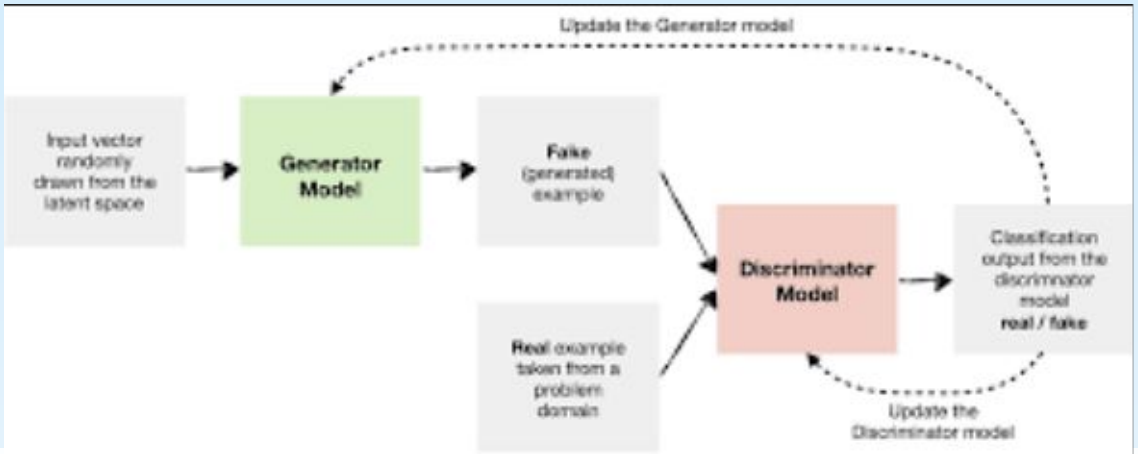
CGAN



CGAN

Key Ideas

- Extended to include conditional information, typically in the form of labels or tags. This inclusion of additional information allows the generated data to be controlled and directed towards a specific domain or characteristic. CGANs are widely used in various applications such as image synthesis, semantic image editing, and data augmentation
- Additional conditional information (c), often in the form of labels or data from other modalities
- Generates data conditioned on specific information for generator
- Discriminator judeges the data basd on autheticity of data and correctness



CGAN (Testing Model)

Testing Model (From GAN)

Generator

Key Ideas

1. **Input and Reshape Layers**:

- **Dense Layer**:
  - **Units**: 256 * 4 * 4
  - **Input**: `latent_dim` (the dimension of the latent space).
- **LeakyReLU**:
  - **Alpha**: 0.2
- **Reshape**:
  - **New Shape**: (4, 4, 256)

2. **Convolutional Layers and Upsampling**:

- **UpSampling2D** (3 times):
- **Conv2D and LeakyReLU Pairs** (3 times):
  - **Filters**: 128 (for each Conv2D layer)
  - **Kernel Size**: (3, 3) (for each Conv2D layer)
  - **Padding**: 'same' (keeps the spatial dimensions constant after the operation)
  - **Activation (LeakyReLU)**: Alpha set to 0.2

3. **Output Layer**:

- **Conv2D**:
  - **Filters**: 3 (representing RGB channels)
  - **Kernel Size**: (3, 3)
  - **Activation**: 'tanh'
  - **Padding**: 'same'

Discriminator

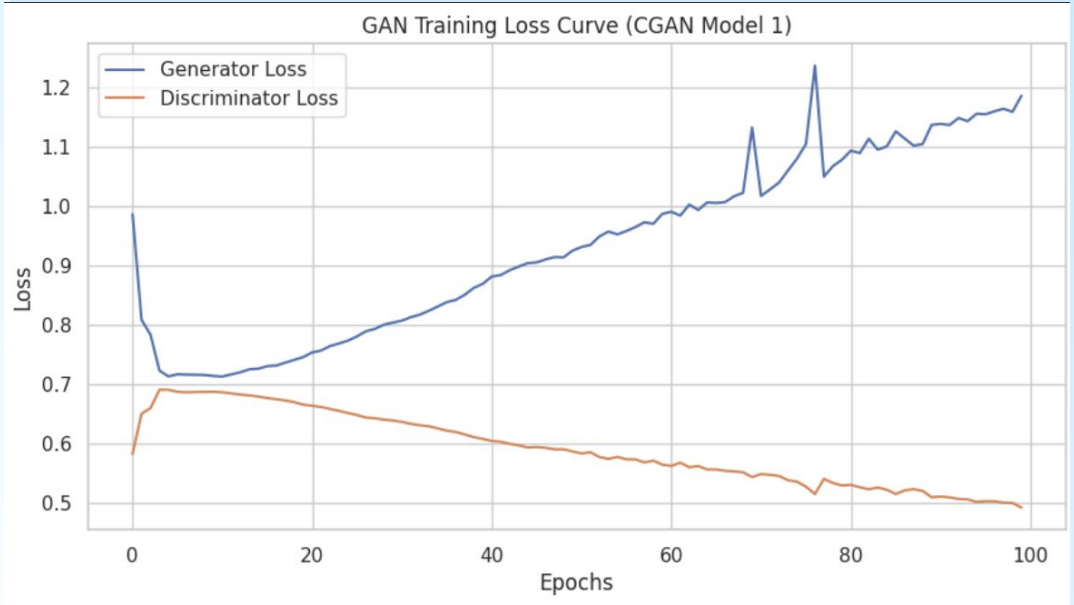
Key Ideas

1. **Input Layer**:

- **Conv2D**:
  - **Filters**: 64
  - **Kernel Size**: (3, 3)
  - **Padding**: 'same'
  - **Input Shape**: `in_shape` (the shape of the input images).
- **LeakyReLU**:
  - **Alpha**: 0.2, allows a small gradient when the unit is inactive, promoting gradient flow during backpropagation.

2. **Convolutional Layers**:
   - Sequentially adding sets of Conv2D and LeakyReLU layers, increasing the number of filters and reducing the spatial dimensions of the feature maps using strides.
   - **First Set**:
     - **Conv2D**:
       - **Filters**: 128
       - **Kernel Size**: (3, 3)
       - **Strides**: (2, 2), reduces the spatial dimensions by half.
       - **Padding**: 'same'
     - **LeakyReLU**: Alpha set to 0.2
   - **Second Set**:
     - **Conv2D**:
       - **Filters**: 128
       - **Kernel Size**: (3, 3)
       - **Strides**: (2, 2)
       - **Padding**: 'same'
     - **LeakyReLU**: Alpha set to 0.2
3. **Flattening and Dense Output Layer**:
   - **Flatten**: Converts the 3D feature maps to a 1D feature vector.
   - **Dropout**:
     - **Rate**: 0.4, randomly sets a fraction of input units to 0, helping to prevent overfitting.
   - **Dense**:
     - **Units**: 1, the output layer with a single neuron for binary classification.
     - **Activation**: 'sigmoid'

CGAN (Testing Model)



1000 images provided similar idea compared to comparing it with a classifier but model really collapsed

Discriminator loss

- For the discriminator loss, it starts relatively low and rises quickly and continues to decrease steadily as the epochs increases

Generator loss

- For the generator loss, it drops and proceeded to increase slowly

- FID Score of 55
- KID Score is 0.04

Both are not too good

CGAN (Model 1)

First Model

Generator

Key Ideas

1. **Input and Dense Layer**:
   - **Dense Layer**:
     - **Units**: 2 * 2 * 512
     - **Input**: `noise` dimension + 10 (for the conditional input).
     - **Activation**: 'relu'
     - **Kernel Initializer**: Random Normal (mean=0, stddev=0.02)
2. **Batch Normalization and Activation (PReLU)**:
   - **BatchNormalization**: Momentum set to 0.8
   - **PReLU**: Parametric ReLU used after each BatchNormalization layer
3. **Reshape Layer**:
   - **Reshape**: Converts the Dense layer's output to be compatible with the Conv2DTranspose layers, reshaping to (2, 2, 512).
4. **Conv2DTranspose Layers**:
   - **1st Conv2DTranspose**:
     - **Filters**: 256
     - **Kernel Size**: 4
     - **Strides**: 2
     - **Padding**: 'same'
     - **Kernel Initializer**: Random Normal (mean=0, stddev=0.02)
   - **2nd Conv2DTranspose**:
     - **Filters**: 128
     - **Kernel Size**: 4
     - **Strides**: 2
     - **Padding**: 'same'
     - **Kernel Initializer**: Random Normal (mean=0, stddev=0.02)
   - **3rd Conv2DTranspose**:
     - **Filters**: 64
     - **Kernel Size**: 4
     - **Strides**: 2
     - **Padding**: 'same'
     - **Kernel Initializer**: Random Normal (mean=0, stddev=0.02)
5. **Output Layer**:
   - **Conv2DTranspose**:
     - **Filters**: 3 (for generating RGB images)
     - **Kernel Size**: 4
     - **Strides**: 2
     - **Padding**: 'same'
     - **Activation**: 'tanh'
     - **Name**: 'Output_Layer'
6. **Merging Input (Noise and Condition)**:
   - **Noise Input**: Input layer for noise vector
   - **Condition Input**: Input layer for conditions
   - **Merged Input**: Concatenation of Noise and Condition inputs

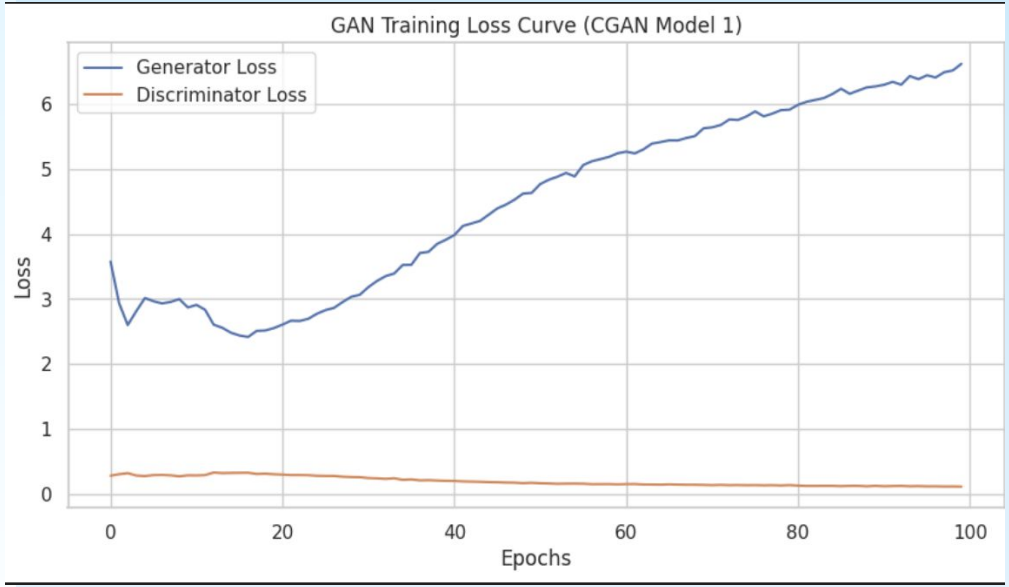
Discriminator

Key Ideas

1. **Input Layers**:
   - **Image Input**: Input layer for the image with shape `image_size` (32, 32, 3).
   - **Conditions Input**: Input layer for the conditions with shape (10,).
2. **Convolutional Layers**:
   - **1st Conv2D Layer**:
     - **Filters**: 64
     - **Kernel Size**: 4
     - **Strides**: 2
     - **Padding**: 'same'
     - **Kernel Initializer**: Random Normal (mean=0, stddev=0.02)
     - Followed by BatchNormalization (momentum=0.8) and LeakyReLU (alpha=0.2)
   - **2nd Conv2D Layer**:
     - **Filters**: 128
     - **Kernel Size**: 4
     - **Strides**: 2
     - **Padding**: 'same'
     - **Kernel Initializer**: Random Normal (mean=0, stddev=0.02)
     - Followed by BatchNormalization (momentum=0.8) and LeakyReLU (alpha=0.2)
   - **3rd Conv2D Layer**:
     - **Filters**: 256
     - **Kernel Size**: 4
     - **Strides**: 2
     - **Padding**: 'same'
     - **Kernel Initializer**: Random Normal (mean=0, stddev=0.02)
     - Followed by BatchNormalization (momentum=0.8) and LeakyReLU (alpha=0.2)
   - **4th Conv2D Layer**:
     - **Filters**: 512
     - **Kernel Size**: 4
     - **Strides**: 2
     - **Padding**: 'same'
     - **Kernel Initializer**: Random Normal (mean=0, stddev=0.02)
     - Followed by BatchNormalization (momentum=0.8) and LeakyReLU (alpha=0.2)
3. **Global Average Pooling**:
   - **GlobalAveragePooling2D**: Reduces the spatial dimensions to a vector.
4. **Merging Layer (Features and Conditions)**:
   - **Concatenate**: Merging the output of the GlobalAveragePooling2D layer with the conditions input.
5. **Dense Layers**:
   - **1st Dense Layer**:
     - **Units**: 512
     - **Activation**: 'relu'
   - **Output Layer (Dense)**:
     - **Units**: 1 (for binary classification, real or fake)
     - **Activation**: 'sigmoid'
     - **Name**: 'Output_Layer'

CGAN (Model 1)

	Model	FID	KID	d_loss	g_loss
0	Baseline CGAN 1	36.450094	0.020338	0.110001	6.623448



1000 images provided similar idea compared to comparing it with a classifier

Discriminator loss

- For the discriminator loss, it starts relatively low and mains around there

Generator loss

- For the generator loss, it increases with a drops then proceeded to increase slowly

- FID Score of 36
- KID Score is 0.02

It did improve and currently the best model

CGAN (Model 2)

Second Model

Generator

Key Differences

1. Use of Spectral Normalization to stabilize the training of the generator.

Key Ideas

1. **Input and Dense Layer**:
   - **Dense Layer**:
     - **Units**: The layer has 22512 units.
     - **Input**: Takes an input of dimension `noise + 10` (noise dimension plus condition vector).
     - **Kernel Initializer**: Utilizes `RandomNormal` with a mean of 0 and a standard deviation of 0.02 for weight initialization.
   - **BatchNormalization**:
     - **Momentum**: Utilizes a momentum factor of 0.8.
   - **PReLU Activation**: Advanced activation function that allows the model to learn the most beneficial activation pattern.
2. **Transposed Convolution Layers**:
   - **First Conv2DTranspose Layer**:
     - **Filters**: 256.
     - **Kernel Size**: 4x4.
     - **Strides**: 2.
     - **Padding**: 'same'.
     - **Spectral Normalization**: Applied for stabilizing training.
     - **BatchNormalization and PReLU Activation** as above.
   - **Second Conv2DTranspose Layer**:
     - Same as the first layer but with 128 filters.
   - **Third Conv2DTranspose Layer**:
     - Same as above but with 64 filters.

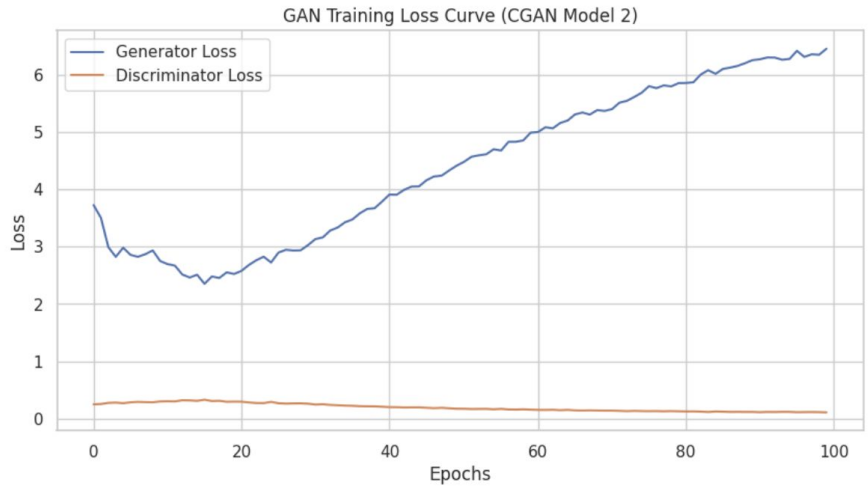
Discriminator

Key Ideas

1. **Input Layers**:
   - **Image Input**: Receives input of shape `image_size`, representing the real or generated images.
   - **Conditions Input**: Receives a 10-dimensional condition vector.
2. **Convolutional Layers**:
   - Sequentially apply convolutional layers with increasing filter sizes (64, 128, 256, 512).
   - **Kernel Size**: 4x4 for all convolutional layers.
   - **Strides**: 2 for all convolutional layers, reducing the spatial dimensions by half each time.
   - **Padding**: 'same', ensuring the output dimensions are only affected by the strides.
   - **BatchNormalization**: Applied after each convolutional layer with a momentum of 0.8.
   - **LeakyReLU Activation**: Utilizes an alpha value of 0.2 for allowing a small gradient when the unit is inactive, promoting gradient flow during training.
3. **GlobalAveragePooling2D Layer**:
   - Reduces the spatial dimensions to a single 512-dimensional vector per image.
4. **Merged Layer**:
   - **Concatenate**: Merges the feature representation from the GlobalAveragePooling2D layer with the condition vector.

CGAN (Model 2)

	Model	FID	KID	d_loss	g_loss
0	Baseline CGAN 1	36.450094	0.020338	0.110001	6.623448
1	Baseline CGAN 2	38.609907	0.023634	0.108867	6.457820



No model collapse

Discriminator loss

- For the discriminator loss, it starts relatively low and mains around there

Generator loss

- For the generator loss, it starts off with drops then proceeded to increase slowly

- FID Score of 39
- KID Score is 0.02

It did not improved further compared to the previous two models

CGAN (Model 3)

Third Model

Generator

Key Difference

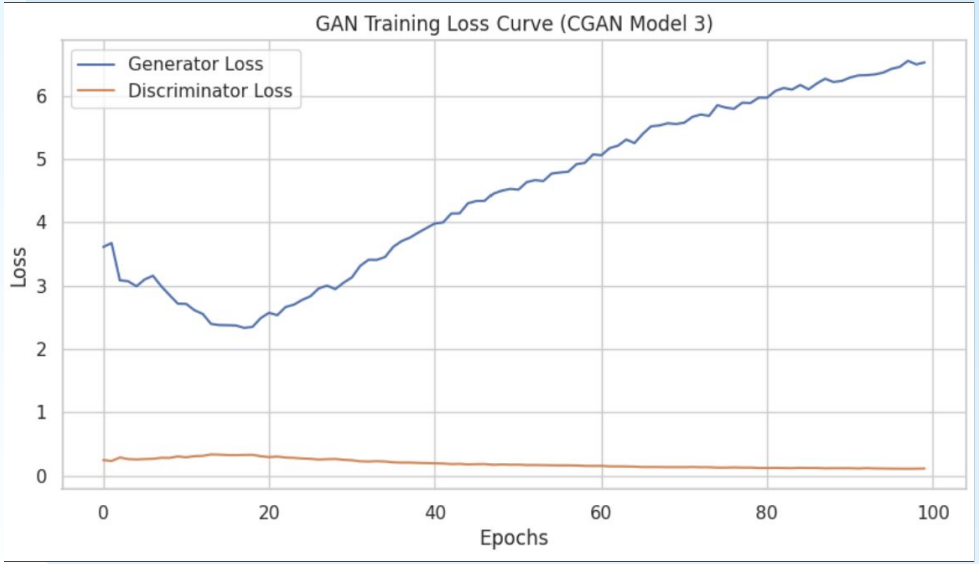
1. Added skip conntection

Key Ideas

1. **Input and Dense Layer**:
   - **Dense Layer**:
     - **Units**: The layer has 2 * 2 * 512 units
     - **Input**: Takes an input of dimension `latent_dim` representing the latent space
     - **Kernel Initializer**: Utilizes `RandomNormal` with a standard deviation of 0.02
2. **Reshape Layer**:
   - **New Shape**: Reshapes the output of the Dense layer into an 2x2x512 tensor.
3. **Normalization and Activation**:
   - **Spectral Normalization**
   - **BatchNormalization**:
   - **PreLu**:
4. **Transposed Convolution Layers**:
   - **Conv2DTranspose (First Layer)**:
     - **Filters**: 256
     - **Kernel Size**: 5x5
     - **Strides**: 2
     - **Padding**: 'same'
   - **Conv2DTranspose (Second Layer)**:
     - **Filters**: 128
     - **Kernel Size**: 5x5
     - **Strides**: 2
     - **Padding**: 'same'
   - **Conv2DTranspose (Third Layer)**:
     - **Filters**: 3 (corresponding to RGB channels)
     - **Kernel Size**: 5x5
     - **Strides**: 2
     - **Padding**: 'same'
     - **Activation**: 'tanh'

CGAN (Model 3)

	Model	FID	KID	d_loss	g_loss
0	Baseline CGAN 1	36.450094	0.020338	0.110001	6.623448
1	Baseline CGAN 2	38.609907	0.023634	0.108867	6.457820
2	Baseline CGAN 3	36.428541	0.018707	0.113654	6.526493



Discriminator loss

- For the discriminator loss, it starts relatively low and remains stagnant with a slight decrease as epoch increases.

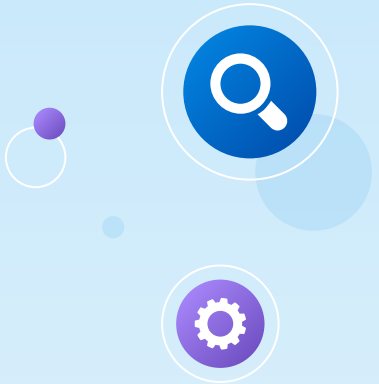
Generator loss

- For the generator loss, it drops then proceeded to increase slowly

- FID Score of 36
- KID Score is 0.018

It improved further compared to the previous two models

WGAN



WGAN

Key Ideas

1. Wasserstein Distance:
    - The Wasserstein distance provides a more stable and meaningful measurement of the distance between two distributions and less likely to provide vanishing gradients, which are a significant issue in traditional GANs
2. Critic instead of Discriminator:
    - In WGAN, the discriminator (now called a critic) doesn't classify inputs as real or fake. Instead, it scores the inputs based on their "realness" without being confined to [0,1] through sigmoid activation. This change helps in providing more useful gradients to the generator throughout the training process. Weight Clipping:
3. No Logarithm in Loss Function:
    - The WGAN loss function doesn't use logarithms, addressing the vanishing gradient problem by providing smoother and more meaningful gradients.

WGAN (Model 1)

First Model

Generator

Key Ideas

1. **Input and Reshape Layers**:

- **Dense Layer**:
  - **Units**: 256 * 4 * 4
  - **Input**: `latent_dim` (the dimension of the latent space).
- **LeakyReLU**:
  - **Alpha**: 0.2
- **Reshape**:
  - **New Shape**: (4, 4, 256)

2. **Convolutional Layers and Upsampling**:

- **UpSampling2D** (3 times):
- **Conv2D and LeakyReLU Pairs** (3 times):
  - **Filters**: 128 (for each Conv2D layer)
  - **Kernel Size**: (3, 3) (for each Conv2D layer)
  - **Padding**: 'same' (keeps the spatial dimensions constant after the operation)
  - **Activation (LeakyReLU)**: Alpha set to 0.2

3. **Output Layer**:

- **Conv2D**:
  - **Filters**: 3 (representing RGB channels)
  - **Kernel Size**: (3, 3)
  - **Activation**: 'tanh'
  - **Padding**: 'same'

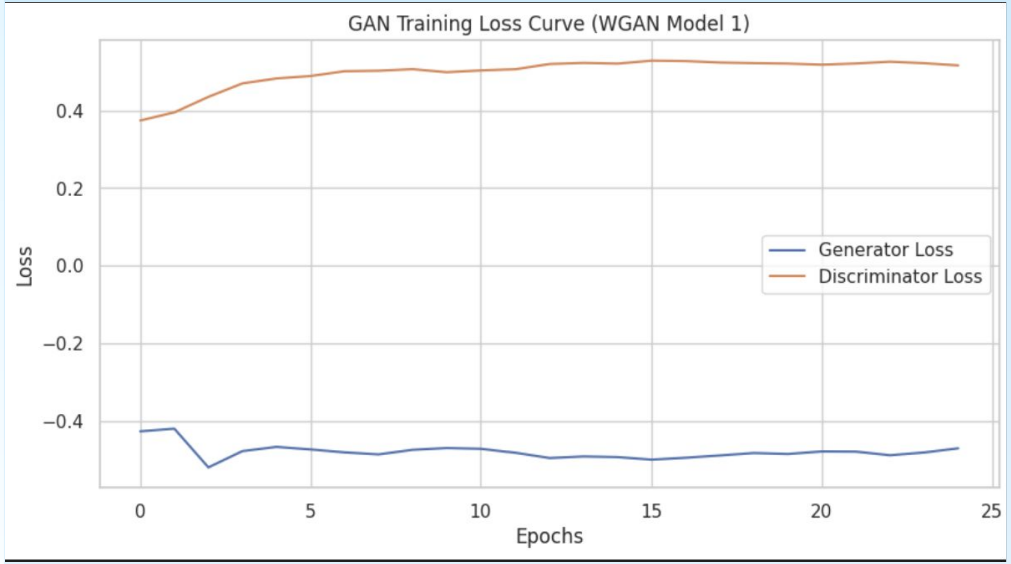
Discriminator

Key Ideas (DCGAN3)

1. **Input Layer and Convolutional Layers**:
   - **Conv2D (First Layer)**:
     - **Filters**: 64
     - **Kernel Size**: (3,3), captures spatial features.
     - **Strides**: (2,2),
     - **Padding**: 'same'
     - **Input Shape**: `in_shape` (the shape of the input images).
   - **LeakyReLU**:
     - **Alpha**: 0.2
   - **Conv2D (Second Layer)**:
     - **Filters**: 128
     - **Kernel Size**: (3,3)
     - **Strides**: (2,2), reduces the spatial dimensions to half
     - **Padding**: 'same'
   - **LeakyReLU**:
     - **Alpha**: 0.2
2. **Flattening and Output Layer**:
   - **Flatten**: Transforms the 2D feature maps into a 1D feature vector, preparing it for the dense output layer.
   - **Dense**:
     - **Units**: 1, the output layer for binary classification.
     - **Activation**: 'sigmoid'

WGAN (Model 1)

	Model	FID	KID	d_loss	g_loss
0	Baseline WGAN 1	129.074924	0.085398	0.515937	-0.469762



- 1000 images provided showing actually model collapse happened

Discriminator loss

- For the discriminator loss, it maintains pretty constant here.

Generator loss

- For the generator loss, it maintains pretty stable here

- FID Score of 129
- KID Score is 0.08

Worse than DCGAN here

WGAN (Model 2)

Second Model

Generator

Key Difference

1. Change upsamling to conv transpose here

Key Ideas

1. **Input and Dense Layer**:
   - **Dense Layer**:
     - **Units**: The layer has 128 * 8 * 8 units.
     - **Input**: Takes an input of dimension `latent_dim` representing the latent space.
     - **Kernel Initializer**: Utilizes `RandomNormal` with a standard deviation of `GAUSS_SD` for weight initialization.
2. **Reshape Layer**:
   - **New Shape**: Reshapes the output of the Dense layer into an 8x8x128 tensor
3. **Normalization and Activation**:
   - **BatchNormalization**:
     - **Momentum**: Utilizes a momentum factor of `MOMENTUM`.
   - **LeakyReLU**:
     - **Alpha**: Set to `ALPHA`.
4. **Transposed Convolution Layers**:
   - **Conv2DTranspose (First Layer)**:
     - **Filters**: 128
     - **Kernel Size**: 5x5
     - **Strides**: 2
     - **Padding**: 'same'
   - **Conv2DTranspose (Second Layer)**:
     - **Filters**: 3 (corresponding to RGB channels)
     - **Kernel Size**: 5x5
     - **Strides**: 2
     - **Padding**: 'same'
     - **Activation**: 'tanh'

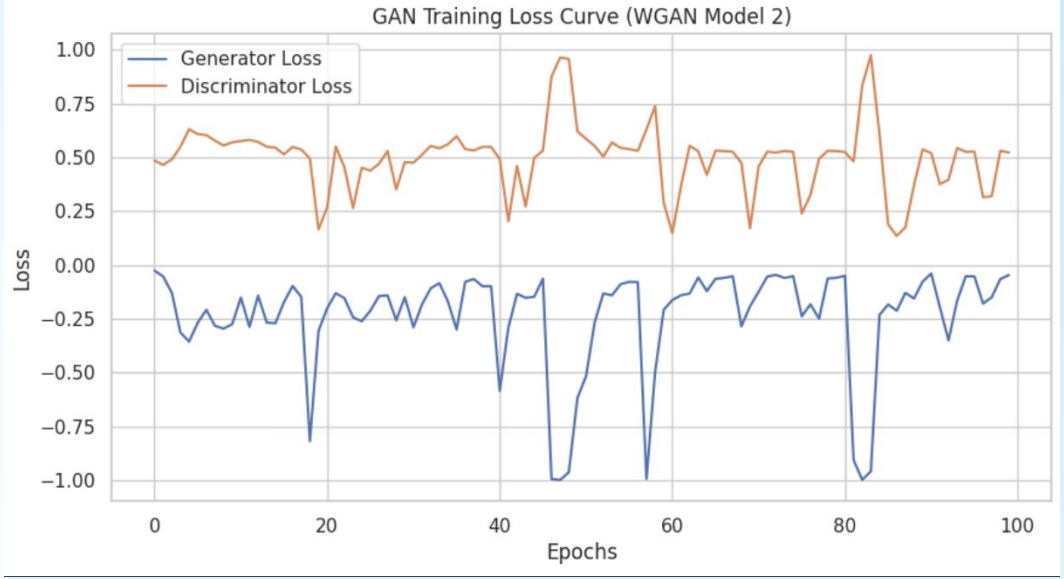
Discriminator

Key Ideas (DCGAN4)

1. **Model Initialization**:
   - **Weights Initialization**: Uses `RandomNormal(mean=0, stddev=0.02)` for initializing the weights of the layers.
   - **Dropout Rate**: 0.3
2. **Input Layer**:
   - **Input Shape**: (32, 32, 3), accepts RGB images of size 32x32 pixels.
3. **Convolutional and Regularization Layers**:
   - **Conv2D with SpectralNormalization (Four Layers)**:
     - **Filters**: Increasing number of filters with each layer (64, 128, 256, 512)
     - **Kernel Size**: (4, 4) for all layers.
     - **Strides**: (2, 2)
     - **Padding**: 'same'
     - **Kernel Initializer**: `weights_init`
   - **LeakyReLU (after each Conv2D)**:
     - **Alpha**: 0.2
   - **Dropout (after each LeakyReLU)**:
     - **Rate**: `dropout_rate`, added after each LeakyReLU layer for regularization.
4. **Flattening and Output Layer**:
   - **Flatten**: Transforms the 3D feature maps into a 1D feature vector, preparing it for the dense output layer.
   - **Dense**:
     - **Units**: 1, the output layer for binary classification.
     - **Activation**: 'sigmoid'

WGAN (Model 2)

	Model	FID	KID	d_loss	g_loss
0	Baseline WGAN 1	129.074924	0.085398	0.515937	-0.469762
1	Baseline WGAN 2	182.955915	0.190047	0.523048	-0.046282



Discriminator loss

- For the discriminator loss, it fluctuates

Generator loss

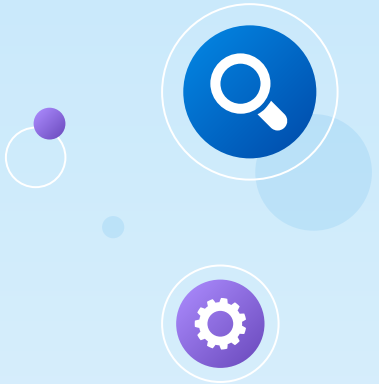
- For the generator loss, it fluctutates

- FID Score of 182
- KID Score is 0.19

It is not too good, worse than DCGAN

Overall, WGAN did not improve our score and not producing good images

SADCGAN



SADCGAN

Key Ideas

- Self attention, allows model to weigh the significance of different parts of input data differently.
  It enables the model to focus on relavant parts of the image when generating new images Below shows how it generally looks like
- Allows model to consider entire spatial extent of image, therefore helping it to udnerstand and generate images with complex compositions
- Able to incooperate to both the generator and discriminator

SADCGAN (Model 1)

First Model

Generator

Key Difference

1. Change upsamling to conv transpose here

Key Ideas

1. **Input and Dense Layer**:
    - **Dense Layer**:
        - **Units**: The layer has 128 * 8 * 8 units.
        - **Input**: Takes an input of dimension `latent_dim` representing the latent space.
        - **Kernel Initializer**: Utilizes `RandomNormal` with a standard deviation of `GAUSS_SD` for weight initialization.
2. **Reshape Layer**:
    - **New Shape**: Reshapes the output of the Dense layer into an 8x8x128 tensor
3. **Normalization and Activation**:
    - **BatchNormalization**:
        - **Momentum**: Utilizes a momentum factor of `MOMENTUM`.
    - **LeakyReLU**:
        - **Alpha**: Set to `ALPHA`.
4. **Transposed Convolution Layers**:
    - **Conv2DTranspose (First Layer)**:
        - **Filters**: 128
        - **Kernel Size**: 5x5
        - **Strides**: 2
        - **Padding**: 'same'
    - **Conv2DTranspose (Second Layer)**:
        - **Filters**: 3 (corresponding to RGB channels)
        - **Kernel Size**: 5x5
        - **Strides**: 2
        - **Padding**: 'same'
        - **Activation**: 'tanh'

Discriminator

Key Ideas

1. Addition of self attention layer

Model Ideas

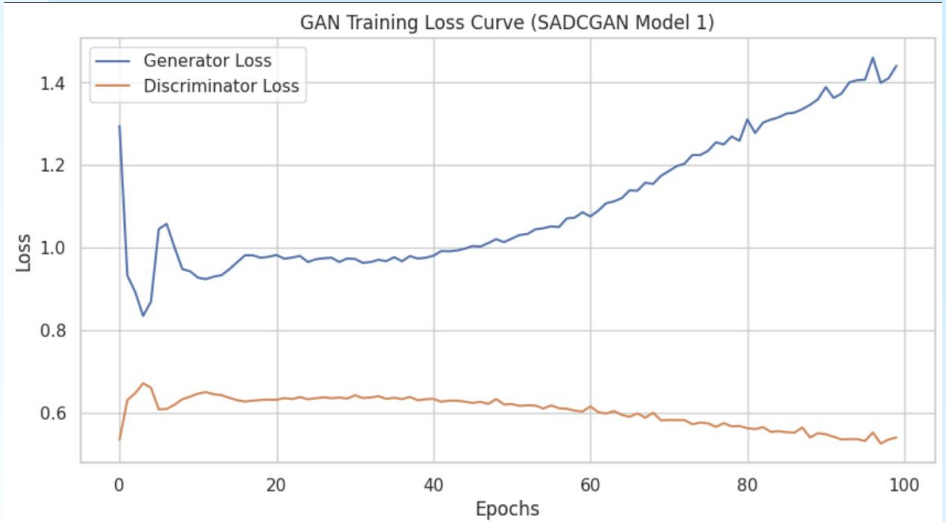
1 **Input and Convolutional Layers**:

- **Conv2D** (First Layer):
    - **Filters**: 64
    - **Kernel Size**: (5, 5), captures the spatial features.
    - **Strides**: 2
    - **Padding**: 'same'
    - **Kernel Initializer**: `RandomNormal(stddev=GAUSS_SD)`
    - **Input Shape**: `in_shape` (the shape of the input images).
- **LeakyReLU**:
    - **Alpha**: `ALPHA`
- **Conv2D** (Second Layer):
    - **Filters**: 128
    - **Kernel Size**: (5, 5)
    - **Strides**: 2
    - **Padding**: 'same'
- **BatchNormalization**:
    - **Momentum**: `MOMENTUM`
- **LeakyReLU**:
    - **Alpha**: `ALPHA`

3. **Flattening and Output Layer**:
    - **Flatten**:
    - **Dropout**:
        - **Rate**: 0.4,
    - **Dense**:
        - **Units**: 1, the output layer for binary classification.
        - **Activation**: 'sigmoid'

SADCGAN (Model 1)

	Model	FID	KID	d_loss	g_loss
0	Baseline SADCGAN 1	50.283339	0.03671	0.53996	1.440854



Similar to above, we can tell most of the animal images like deer,with some other classes also seen

Discriminator loss

- For the discriminator loss, it starts relatively low and mains around there

Generator loss

- For the generator loss, it increases with a spike and drops and proceeded to increase slowly

- FID Score of 50
- KID Score is 0.04

Did have a minor improvement over baseline dcgan 2

SADCGAN (Model 2)

Second Model

Generator

Key Difference

1. Self attention layer added to both discriminator and generator

Key Ideas

1. **Input and Dense Layer**:
   - **Dense Layer**:
     - **Units**: The layer has 4 * 4 * 512 units
     - **Input**: Takes an input of dimension `latent_dim` representing the latent space
     - **Kernel Initializer**: Utilizes `RandomNormal` with a standard deviation of 0.02
2. **Reshape Layer**:
   - **New Shape**: Reshapes the output of the Dense layer into an 4x4x512 tensor.
3. **Normalization and Activation**:
   - **BatchNormalization**:
   - **PreLu**:
4. **Transposed Convolution Layers**:
   - **Conv2DTranspose (First Layer)**:
     - **Filters**: 256
     - **Kernel Size**: 5x5
     - **Strides**: 2
     - **Padding**: 'same'
   - **Conv2DTranspose (Second Layer)**:
     - **Filters**: 128
     - **Kernel Size**: 5x5
     - **Strides**: 2
     - **Padding**: 'same'
   - **Conv2DTranspose (Third Layer)**:
     - **Filters**: 3 (corresponding to RGB channels)
     - **Kernel Size**: 5x5
     - **Strides**: 2
     - **Padding**: 'same'
     - **Activation**: 'tanh'

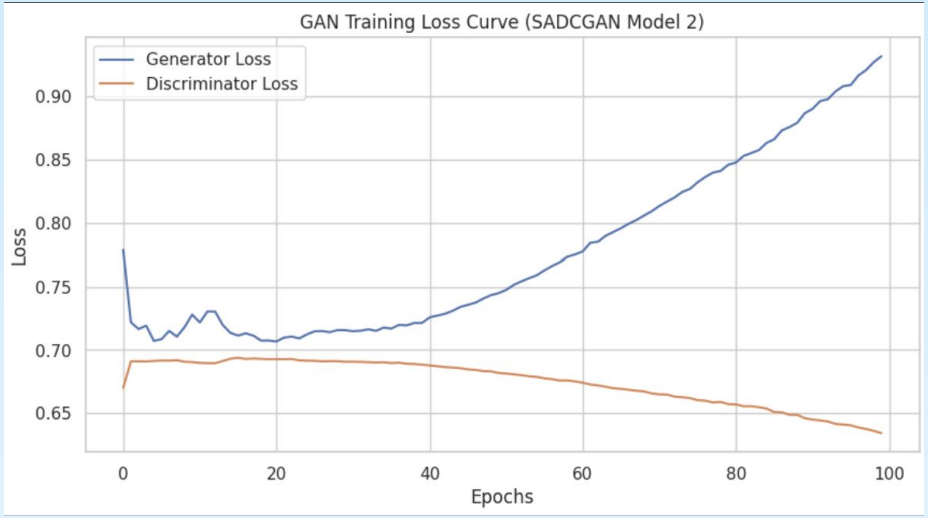
Discriminator

Key Ideas

1. **Input Layer and Convolutional Layers**:
   - **Conv2D (First Layer)**:
     - **Filters**: 64
     - **Kernel Size**: (3,3), captures spatial features.
     - **Strides**: (2,2),
     - **Padding**: 'same'
     - **Input Shape**: `in_shape` (the shape of the input images).
   - **LeakyReLU**:
     - **Alpha**: 0.2
   - **Conv2D (Second Layer)**:
     - **Filters**: 128
     - **Kernel Size**: (3,3)
     - **Strides**: (2,2), reduces the spatial dimensions to half
     - **Padding**: 'same'
   - **LeakyReLU**:
     - **Alpha**: 0.2
2. **Flattening and Output Layer**:
   - **Flatten**: Transforms the 2D feature maps into a 1D feature vector, preparing it for the dense output layer.
   - **Dense**:
     - **Units**: 1, the output layer for binary classification.
     - **Activation**: 'sigmoid'

SADCGAN (Model 2)

	Model	FID	KID	d_loss	g_loss
0	Baseline SADCGAN 1	50.283339	0.036710	0.539960	1.440854
1	Baseline SADCGAN 2	49.595077	0.032582	0.634288	0.931837



Discriminator loss

- For the discriminator loss, it starts relatively low and rises quickly and continues to decrease steadily as the epochs increases

Generator loss

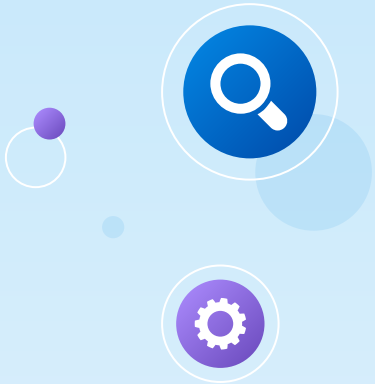
- For the generator loss, it drops and proceeded to increase slowly

- FID Score of 49
- KID Score is 0.047

It did not improve too much.

04

Evaluation



Evaluation Metrics

KID (Kernel Inception Distance)

1. Metrics used for evaluting the quality of images generated by GAN. This measures the similarity between two sets of images by comparing the statistics of deep features. These uses the pre trained Inception model

$$MMD^2 = \frac{1}{m^2}\sum_{i=1}^{m}\sum_{j=1}^{m}K(x_i, x_j) + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}K(y_i, y_j) - \frac{2}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}K(x_i, y_j)$$

- Compute the kernel metrics using kernel function
- Calculate the mean Maximum Mean Discrepancy
- Final average KID values computed over entire dataset

FID

- Metrics that compares the statistics of generated images to real images in the feature space of a deep learning model (Inception model) and calculate the distance betwen the feature vectors of the real and generated images, lower FID implies that better quality of generated images

$$FID = ||\mu_x - \mu_y||^2 + Tr(\Sigma_x + \Sigma_y - 2(\Sigma_x\Sigma_y)^{1/2})$$

- Includes the Euclidean distance between mean vectors of real and generated images
- Tr denotes the trace of matrix

$$(\Sigma_x\Sigma_y)^{1/2}$$

Denotes the sqaure root of the product of the covariance matrices which is computed using methods eigen decomposition

Evaluation Metrics

Visual Approach

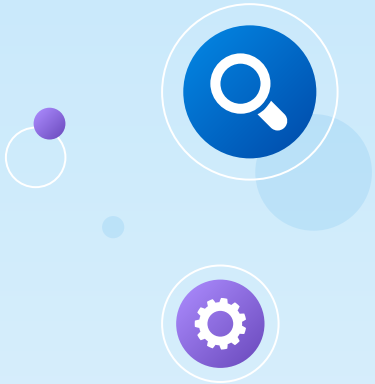
1. Even with metrics above, human approach is required to be able to see and tell what the images is
2. Below function is to generate 1000 images to be able to view the diversity and how well the image is representing the data

```python
def generate_1000(generator, num_iterations=10, saveDisk=False):
    for iteration in range(num_iterations):
        latent_z = np.random.normal(size=(100, 128))
        imgs = generator.predict(latent_z)
        fig, axes = plt.subplots(10, 10, figsize=(30, 30))
        fig.patch.set_facecolor('#abdbe3')
        fig.subplots_adjust(left=0.05, right=0.95, bottom=0.05, top=0.95, hspace=0.1, wspace=0.1)

        for idx, ax in enumerate(axes.flatten()):
            ax.imshow((imgs[idx] + 1) / 2, interpolation='nearest')
            ax.axis('off')

        if saveDisk:
            plt.savefig(f'generated_images_set_{iteration + 1}.png', bbox_inches='tight', pad_inches=0.5)  # Adjust padding
        else:
            plt.show()

        plt.close(fig)
```

05

Model Improvement

We will further improve CGAN model 1 and DCGAN model 3

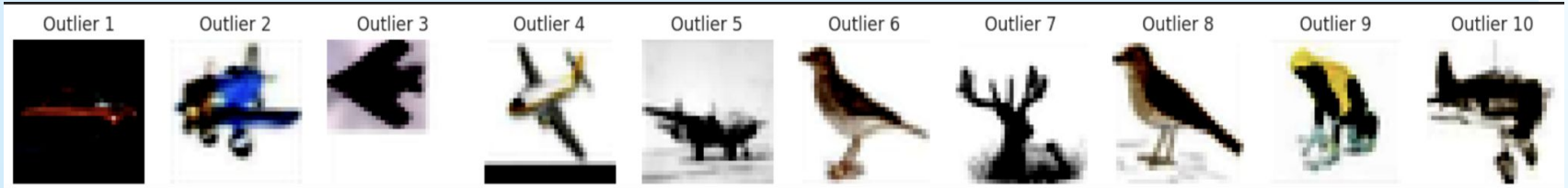


Model Improvement – Outlier Testing

1. Outlier testing in GANs is crucial for ensuring the generative model produces high-quality and diverse outputs, identifying instances where the model deviates significantly from the expected data distribution.
2. By integrating outlier detection mechanisms, researchers can monitor and improve the stability of GAN training, mitigating common issues like mode collapse, where the model generates limited varieties of outputs.
3. Techniques like statistical outlier detection, distance metrics in latent space, or comparison against a validation set are commonly used to identify anomalies in the data generated by GANs.
4. Incorporating outlier testing into the GAN framework not only enhances the reliability of the model but also provides valuable insights into the model's learning dynamics and potential areas for improvement
5. This is done with the use of auto encoder

- Auto Encoder are a type of deep-learning architectures that are used to learn how to compress and decompress data faithfully.
- The compression layers are referred to as the encoding layers, and the decompression layers are referred to as the decoding layer

Model Improvement – Outlier Testing



Outlier image seems to be mix with half clear and half unclear

Model Improvement – Sharpening Image

1. Sharpening images in GANs is a post-processing step aimed at enhancing the clarity and detail of generated images, often addressing common issues like blurriness or lack of texture definition
2. This technique involves applying image processing filters, such as unsharp masking or high-pass filters, to emphasize edges and fine details, making the generated images more realistic and visually appealing
3. Image sharpening can be particularly beneficial in applications where GANs are used for super-resolution tasks, converting low-resolution images to high-resolution counterparts while maintaining high-quality details
4. Incorporating sharpening techniques into GAN frameworks can also aid in improving the perceptual quality of generated images, making them more convincing to the human eye and useful for practical applications

Model Improvement – Sharpening Image

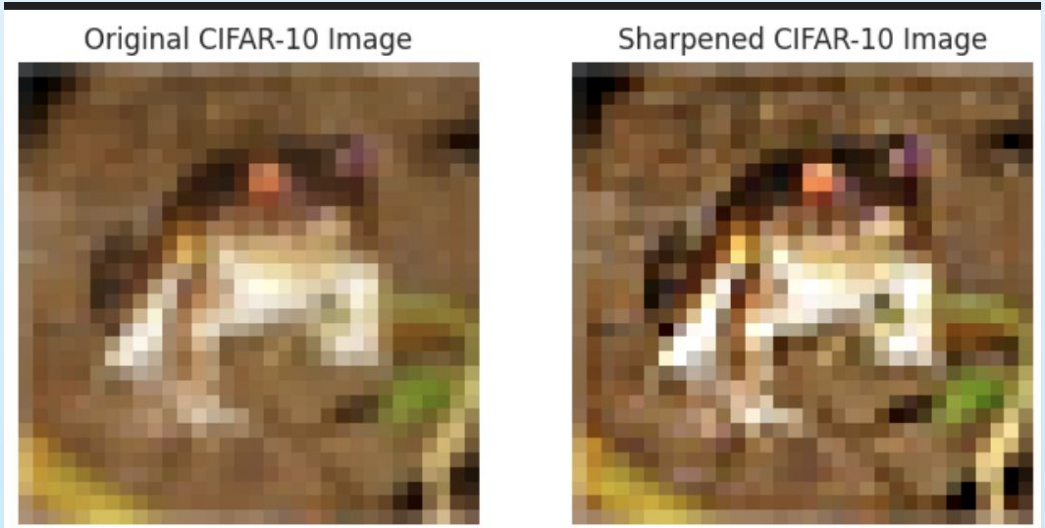


Image are sharpened even clearer here

Model Improvement – Data Augmentation

1. Data augmentation in GANs involves artificially expanding the dataset by generating altered versions of the training images, utilizing techniques such as rotation, width and height shifts, and horizontal flipping, as seen by ImageDataGenerator

2. By applying transformations like a 15-degree rotation range, 10% width and height shift range, and enabling horizontal flips, `ImageDataGenerator` enhances the diversity of the training data, helping the model generalize better and reducing overfitting

3. This augmentation process makes the GAN model more robust to variations in input data, effectively teaching the model to recognize and generate images that maintain their identity despite alterations in orientation, position, and perspective

4. Incorporating such data augmentation strategies is particularly useful in scenarios where the dataset is limited, as it artificially expands the dataset size, providing the GAN with more varied examples to learn from

Model Improvement – CutMix

1. The idea of cutmix is to base on the labels mix two different images.
2. This helps with the discriminator such that the goal is to be able to learn more with less through this process
3. However, this could result in generated images being unrealistic with cutout portion

Model Improvement – OverSampling

1. The idea here is to oversample the dataset that we have
2. The goal is to generate more dataset to be able to have more data to put into training for gan.
3. Resample from sklearn was used to oversample the data here
4. The process was first done on the train data

Model Improvement – OverSampling

1. The idea here is to oversample the dataset that we have
2. The goal is to generate more dataset to be able to have more data to put into training for gan.
3. Resample from sklearn was used to oversample the data here
4. The process was first done on the train data

Model Improvement – Cosine and Exponential Decay

Cosine Decay Scheduler

1. Cosine decay reduces the learning rate in cosine wave pattern, slowing down at the beginning and end of training.
2. The idea is to start with an initial learning rate and then reduce it which at the start descends slowly and more steeply in the middle.
3. This approach is suitable for training neural networks as it combines both rapid and slow decay of learning rate

Exponential Decay Scheduler

The idea here is to oversample the dataset that we have

1. Exponential decay reduces the learning rate exponentially where the rate is decreasing at a exponential rate
2. Very useful in cases where one wants to rapidly decrease the learning rate
3. Learning rate is a fraction of the previous learning rate here

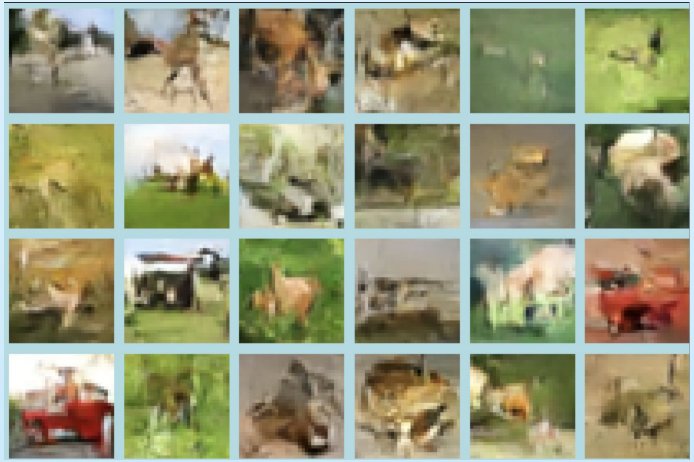
Scores After Model Improvement

	Model	FID	KID	d_loss	g_loss
0	Non-Outlier DCGAN 3	46.075019	2.945318e-02	0.686637	0.746695
1	Sharpened Image DCGAN 3	680163.777727	3.370699e+07	0.684277	0.750914
2	Improved Data Generator DCGAN 3	700665.890898	3.764938e+07	0.689118	0.726457
3	CutMix DCGAN 3	51.564493	3.489228e-02	0.686637	0.746695
4	Oversample DCGAN 3	64.013066	4.585502e-02	0.647558	0.915615
5	CosineDecay DCGAN 3	46.440363	2.945601e-02	0.685025	0.751231
6	ExponentialDecay DCGAN 3	49.036579	3.086610e-02	0.683575	0.761838

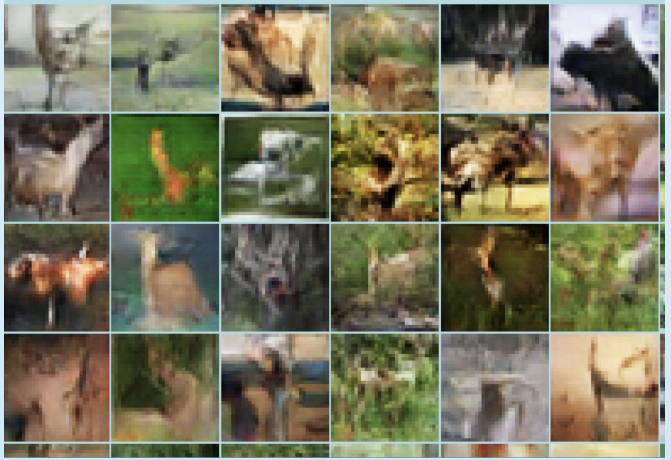
	Model	FID	KID	d_loss	g_loss
0	Non-Outlier CGAN 1	35.389534	2.167156e-02	0.105153	6.727473
1	Sharpened Image CGAN 1	680924.469492	3.372896e+07	0.105472	6.766756
2	CutMix Image CGAN 1	91.519452	7.159536e-02	0.109868	4.574687
3	Oversample Image CGAN 1	32.972425	2.073221e-02	0.109868	4.574687
4	CosineDecay CGAN 1	32.555592	1.869066e-02	0.090373	7.033971
5	ExponentialDecay CGAN 1	32.555592	1.869066e-02	0.108475	6.636795

Stable Diffusion VS CGAN 1

Stable Diffusion



CGAN 1

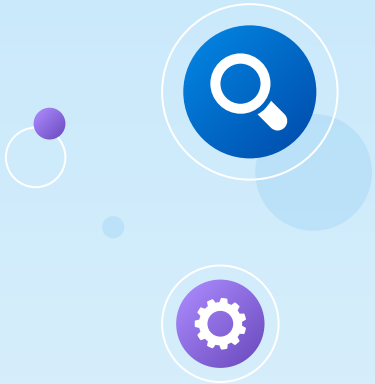


OVERALL: CGAN Produces better image overall but stable diffusion potential is huge as this is just a baseline basic model here

06

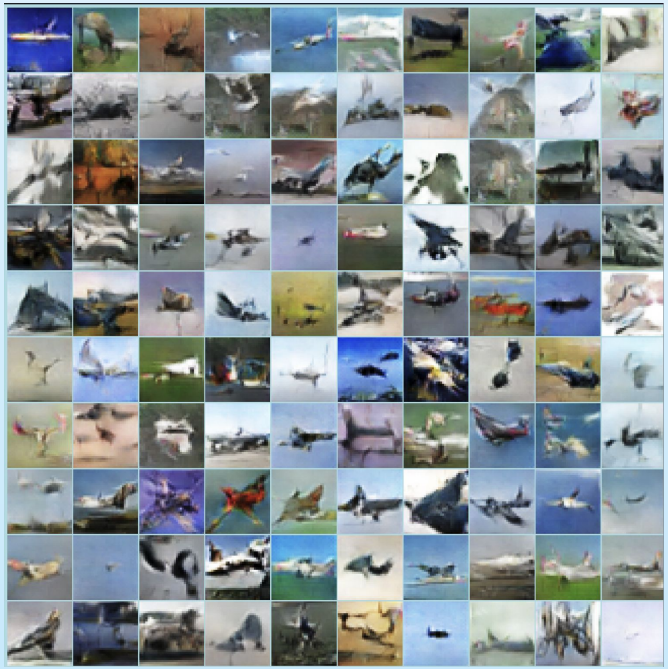
Final Model

For our final model, we will choose CGAN 1 (Exponential Decay).

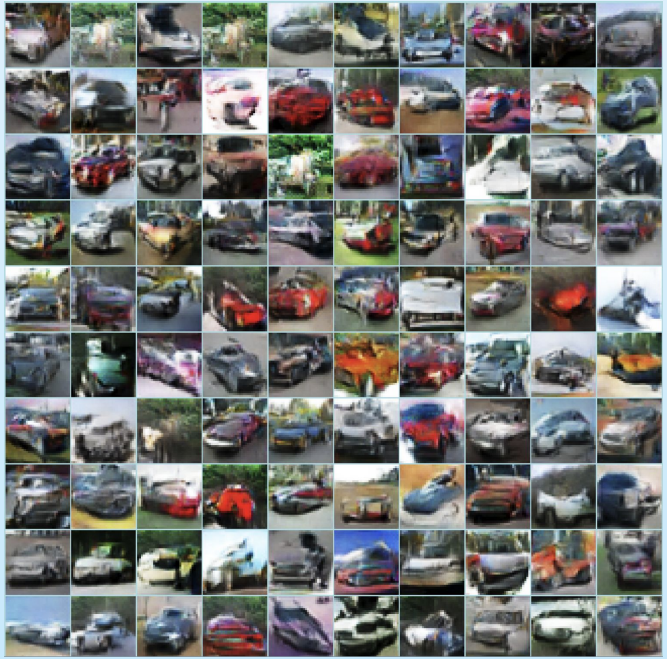


Final Images

Airplane



Automobile



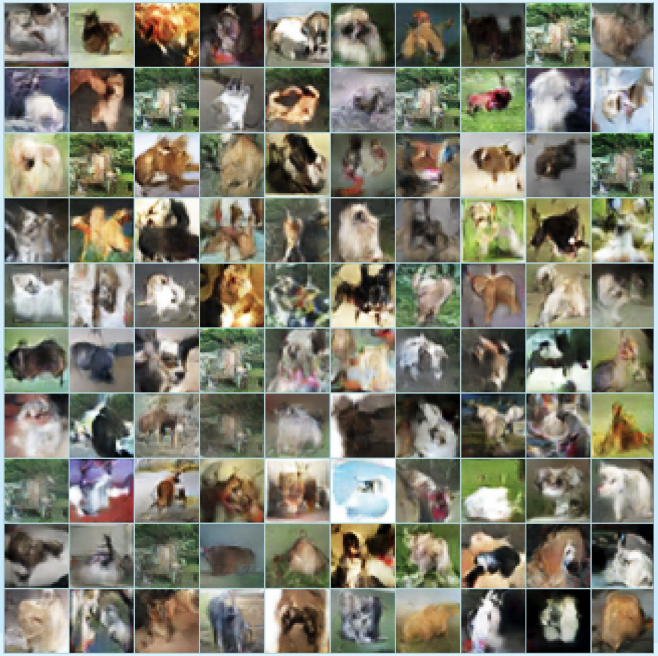
Final Images

Bird

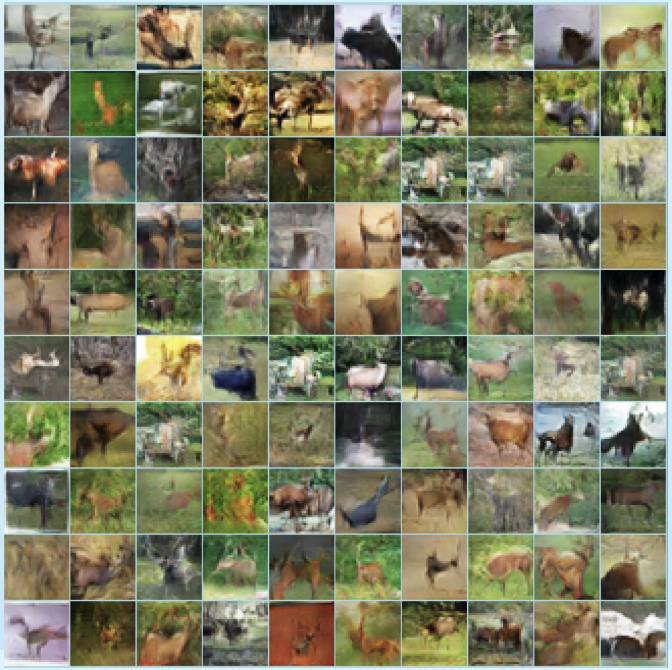
Cat

Final Images

Dog

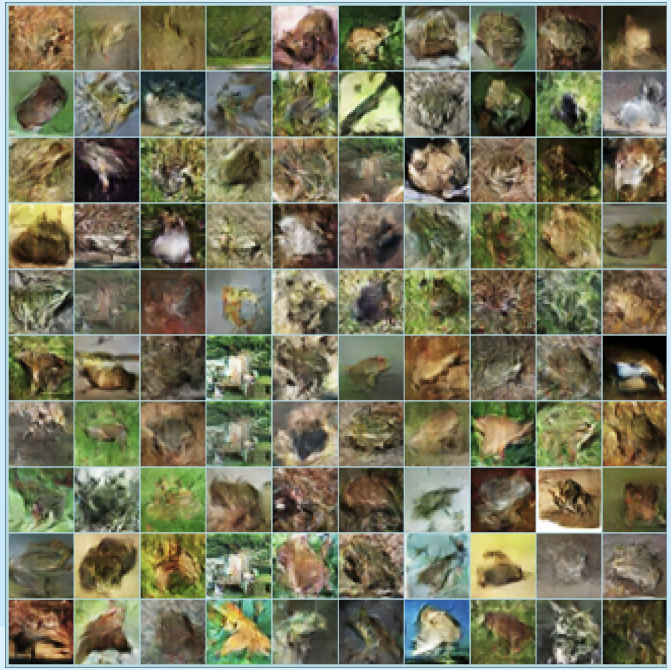


Deer



Final Images

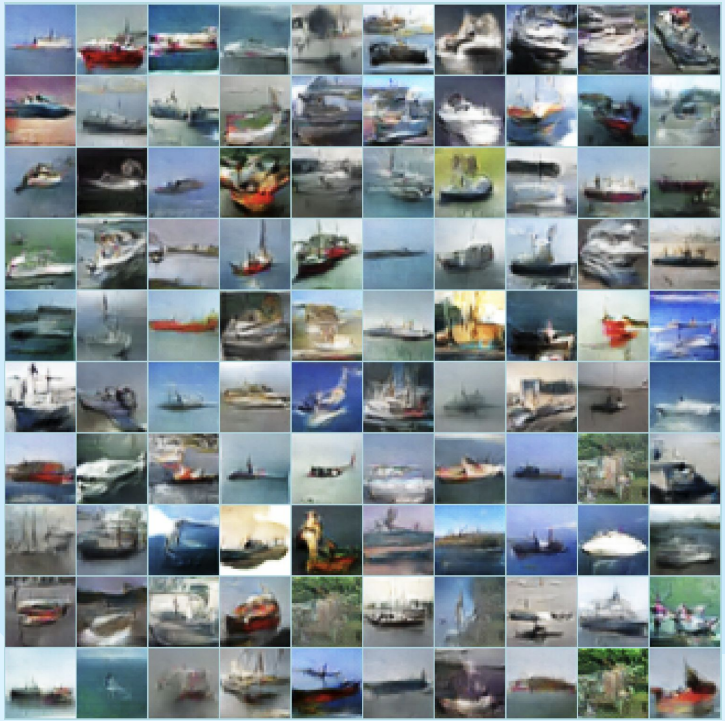
Frog



Horse

Final Images

Ship



Truck

Thank You!

CREDITS: This presentation template was created by **Slidesgo**, and includes icons by **Flaticon** and infographics & images by **Freepik**