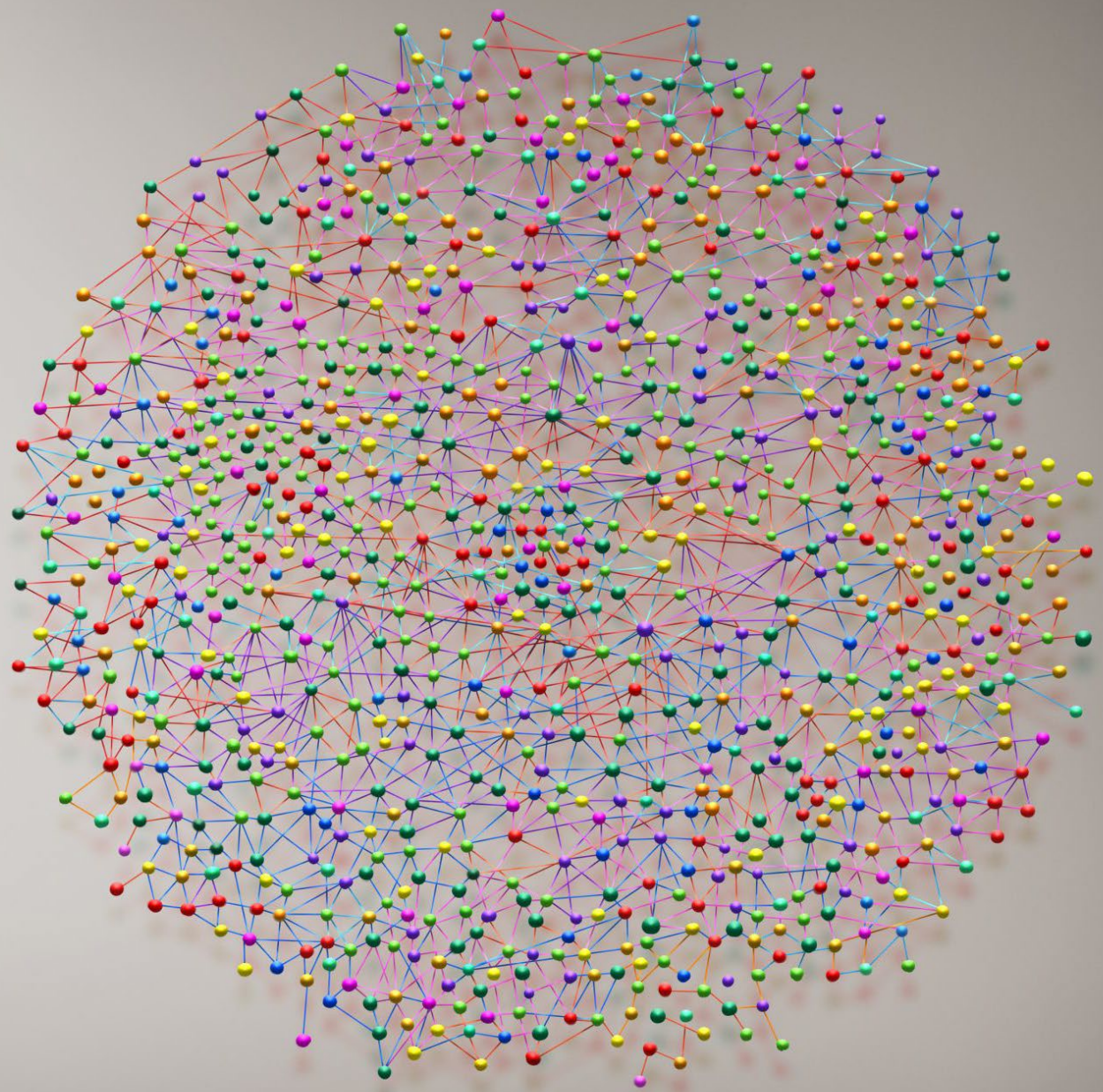
Image Generation with Generative Adversarial Network

Saurabh Parkar

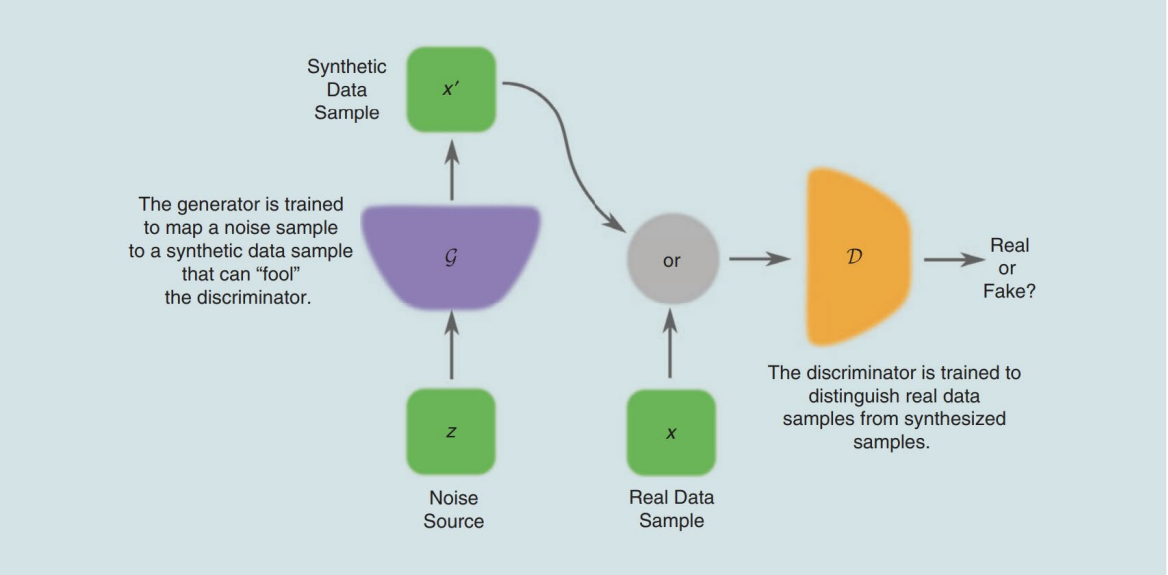
CWID: 20021678



Contents

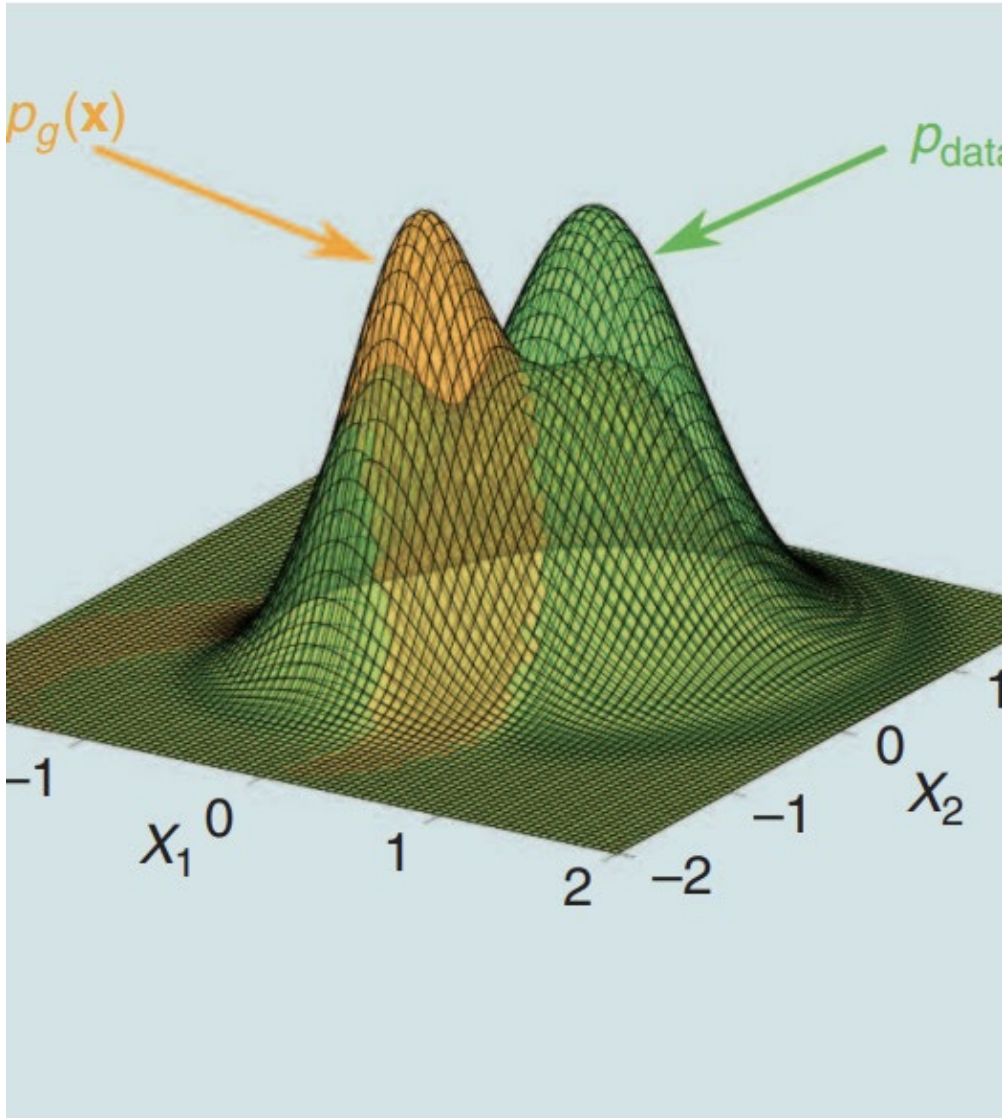
Introduction: What are Generative Adversarial Network(GANS)?

Generative adversarial networks (GANs) provide a way to learn deep representations without extensively annotated training data. They achieve this by deriving backpropagation signals through a competitive process involving a pair of networks. The representations that can be learned by GANs may be used in a variety of applications, including image synthesis, semantic image editing, style transfer, image superresolution, and classification.



Introduction: How do GANS Work?

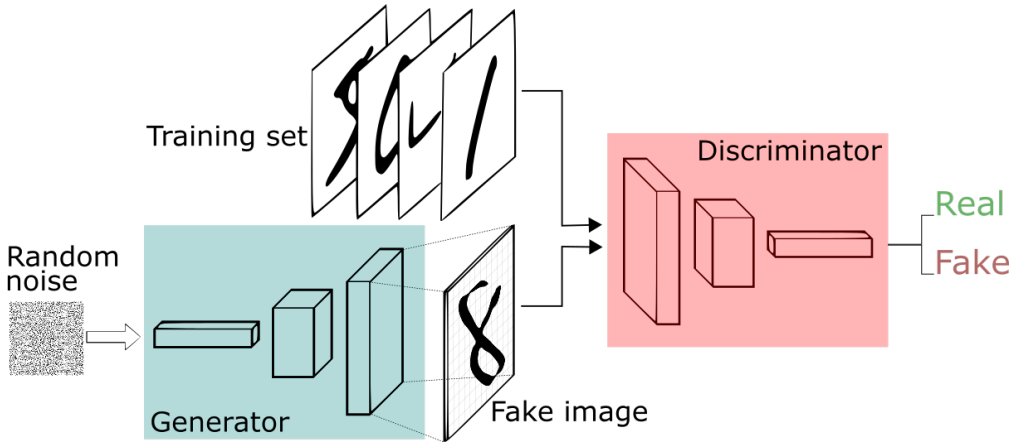
- In generative modeling, training examples x are drawn from an unknown distribution $p_{data}(x)$. The goal of a generative modeling algorithm is to learn a $p_{model}(x)$ that approximates $p_{data}(x)$ as closely as possible.
- A straightforward way to learn an approximation of p_{data} is to explicitly write a function $p_{model}(x; \theta)$ controlled by parameters θ and search for the value of the parameters that makes p_{data} and p_{model} as similar as possible.
- In particular, the most popular approach to generative modeling is probably maximum likelihood estimation, consisting of minimizing the Kullback-Leibler divergence between p_{data} and p_{model} . The common approach of estimating the mean parameter of a Gaussian distribution by taking the mean of a set of observations is one example of maximum likelihood estimation.



Introduction: How Do GANS Work?

The discriminator determines if each instance of data that it analyzes is actually a part of the training data set, whereas the generator, creates new data instances, that are evaluated for authenticity.

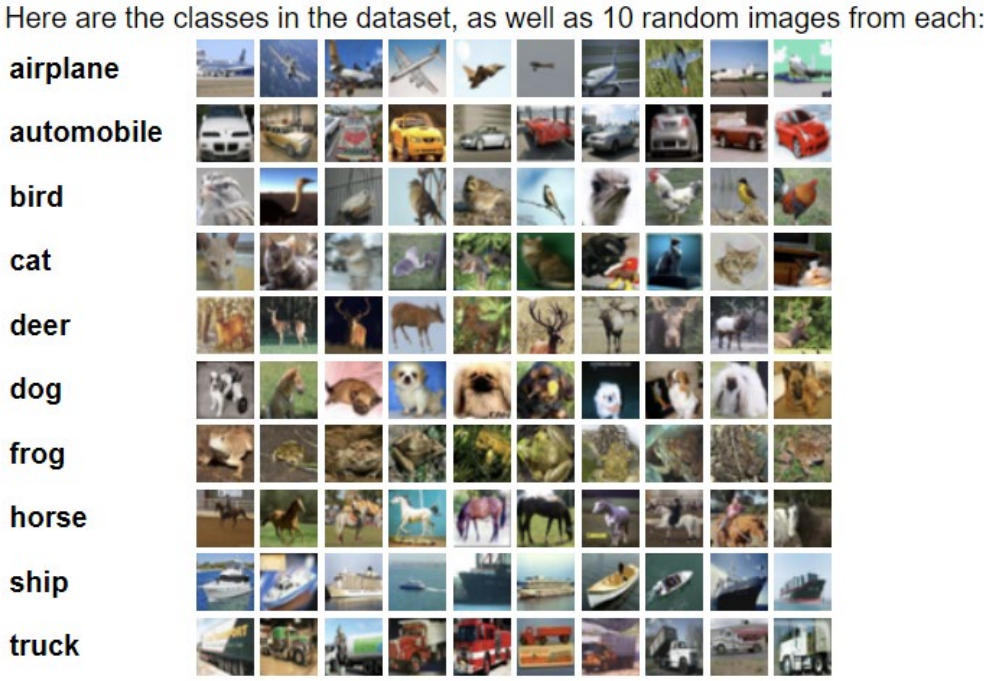
The cost of each participant determines how the algorithm will proceed. Using the Minimax game, where the generator value is equal to the discriminator cost minus one, is the simplest way to define the cost.



Data Set: The CIFAR-10 dataset

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images.



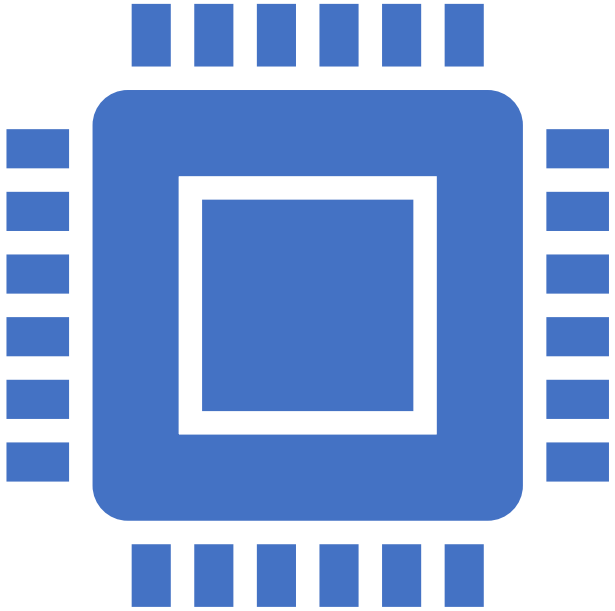
The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

```
# Setting some hyperparameters
batchSize = 64 # the size of the batch.
imageSize = 64 # the size of the generated images (64x64).

# Creating the transformations (scaling, tensor conversion, normalization) to apply to the input images.
transform = transforms.Compose([transforms.Resize(imageSize), transforms.ToTensor(), transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),])

# Loading the dataset
dataset = dset.CIFAR10(root = './data', download = True, transform = transform)
dataloader = torch.utils.data.DataLoader(dataset, batch_size = batchSize, shuffle = True, num_workers = 2)
```

Data Preprocessing

Images in CIFAR-10 are of size 32x32

For our experiment, we rescale them to 64x64 and divide the dataset in batch sizes of 64 using the transforms and dataloader methods of Pytorch



Requirements

Hardware:

CPU: Intel i5 12th generation or higher, AMD Ryzen 5 5000s or higher

GPU: Nvidia CUDA Compatible GPU with minimum Pascal Architecture

CUDA and cuDNN: CUDA compute support of 8.6 and cuDNN support of 11.8 with Video Memory atleast 8GB

Memory: 16GB and Higher

Software:

Programming Language: Python 3.10

IDE: VSCode, Jupyter or Anaconda

Modules: PyTorch, numpy, matplotlib

Model: Generator

The GAN model consists of two Neural Network Component, generator and Discriminator

Generator (G):

The generator is a sequential neural network composed of several layers.

- The generator (G) has a total of **5 layers.**
- **Initial layer:** ConvTranspose2d with 100 input channels, 512 output channels, a kernel size of (4, 4), and stride of (1, 1). It performs transposed convolution.
- Batch normalization is applied to the output of the first convolutional layer with 512 channels.
- **ReLU** activation is applied after batch normalization.
- Three more layers follow, each consisting of **ConvTranspose2d, BatchNorm2d, and ReLU layers**, progressively reducing the number of channels from 512 to 256, 256 to 128, and 128 to 64.
- The final layer is a **ConvTranspose2d with 64 input channels, 3 output channels (representing RGB channels)**, a kernel size of (4, 4), stride of (2, 2), and Tanh activation function.

```
G(
  (main): Sequential(
    (0): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace=True)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
)
```

Model: Discriminator

Discriminator (D):

The discriminator is also a sequential neural network.

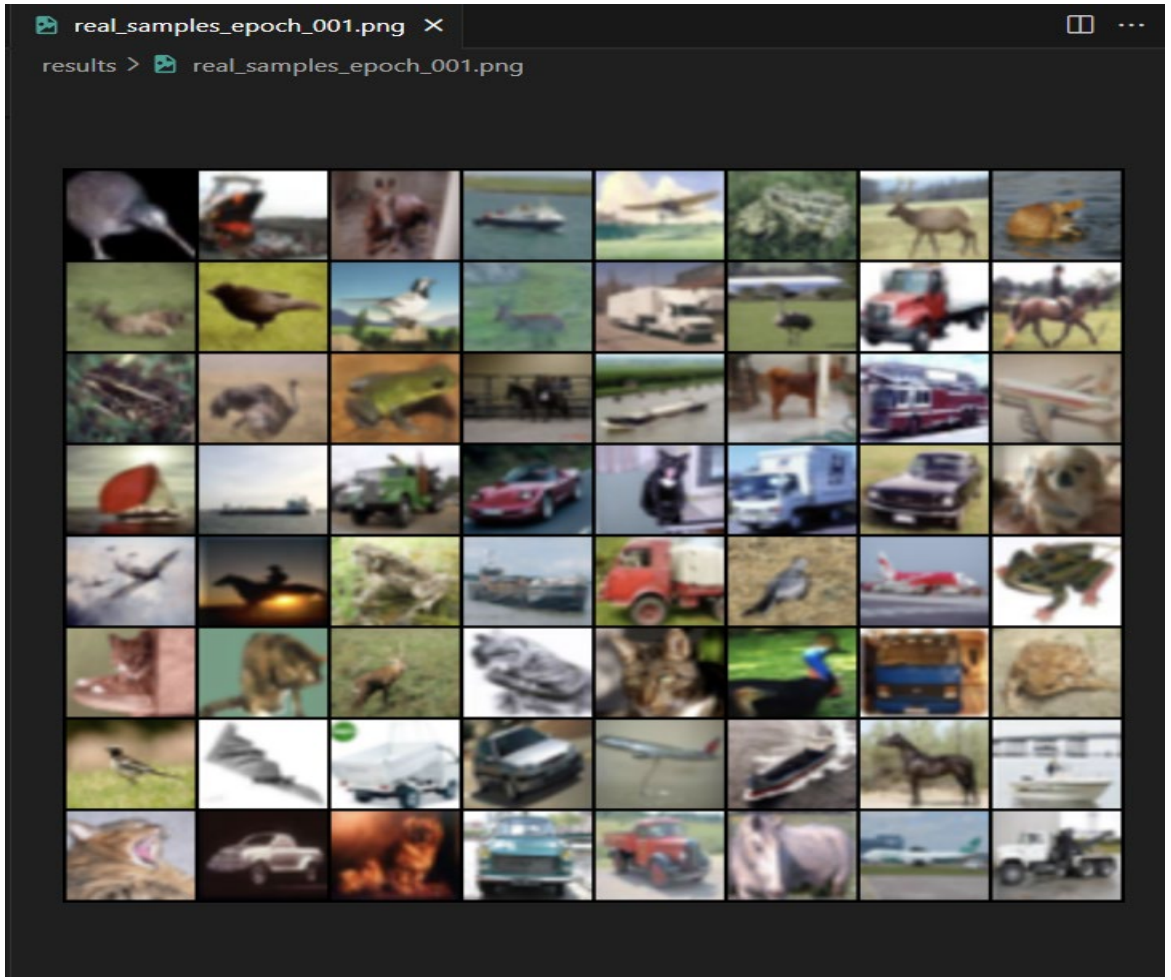
- The discriminator (D) has a total of 5 layers.
- **Initial layer:** Conv2d with 3 input channels, 64 output channels, a kernel size of (4, 4), and stride of (2, 2). LeakyReLU activation is applied.
- Three more layers follow, each consisting of **Conv2d, BatchNorm2d, and LeakyReLU layers**, progressively increasing the number of channels from 64 to 128, 128 to 256, and 256 to 512.
- The **final layer is a Conv2d** with 512 input channels and 1 output channel, representing the binary classification result (real or fake). Sigmoid activation is applied after this convolutional layer.

In summary, the generator takes random noise as input and generates synthetic images, while the discriminator aims to distinguish between real and generated images. The GAN training process involves the generator trying to improve its ability to generate realistic images, and the discriminator trying to improve its ability to differentiate between real and fake images. This adversarial training loop leads to the generation of increasingly realistic images by the generator.

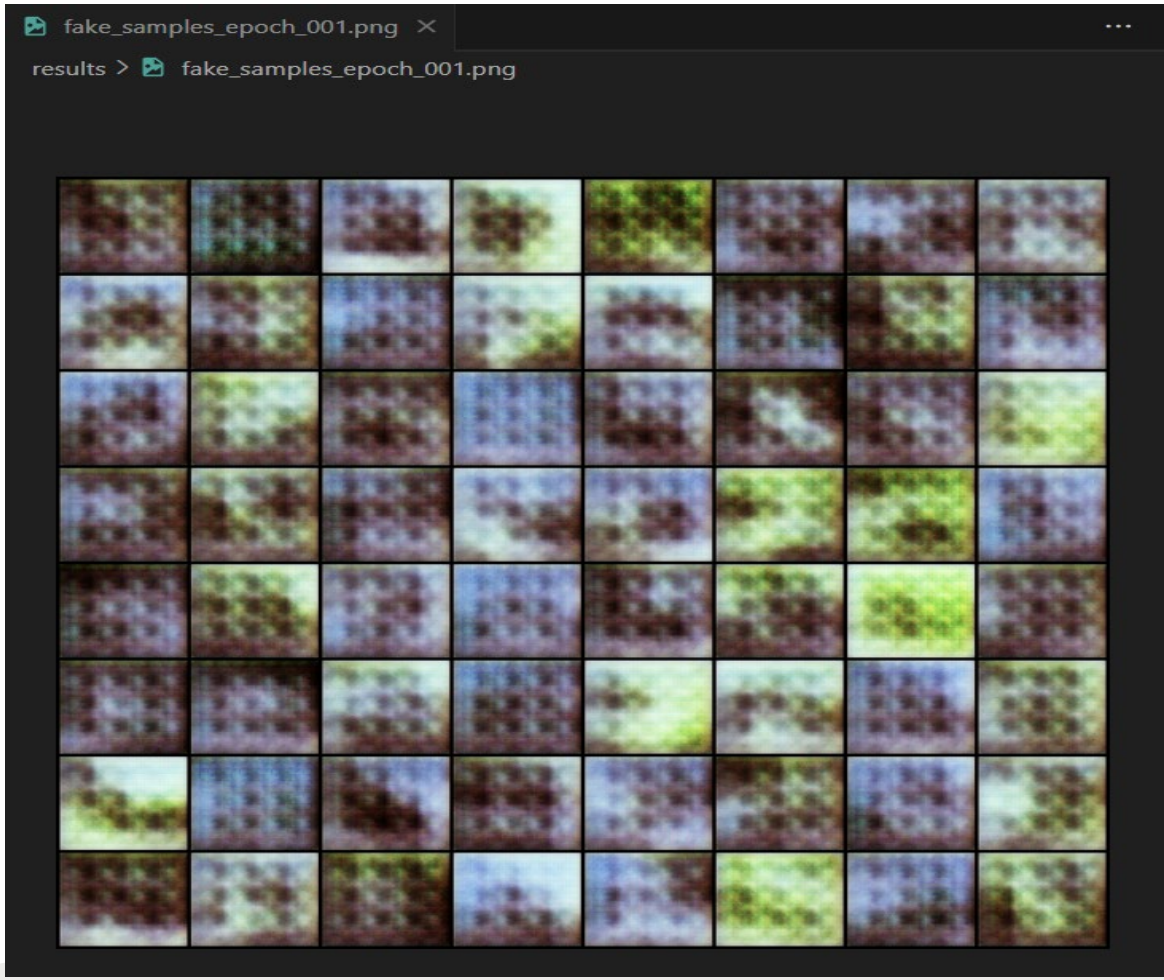
```
D(
  (main): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, 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)
    (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (12): Sigmoid()
  )
)
```

Results: Initial Epoch 1/30

Real Image Sample

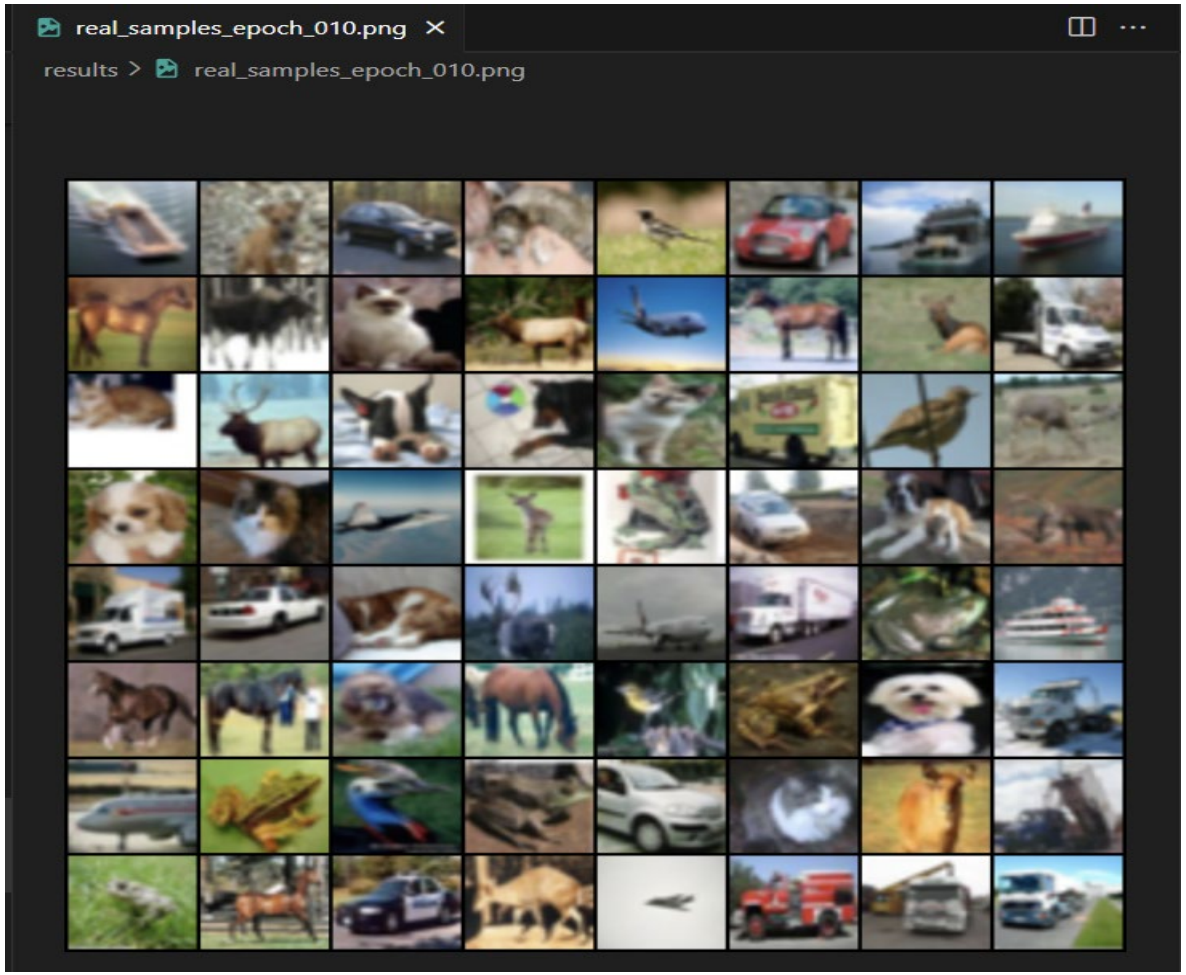


Generated Image

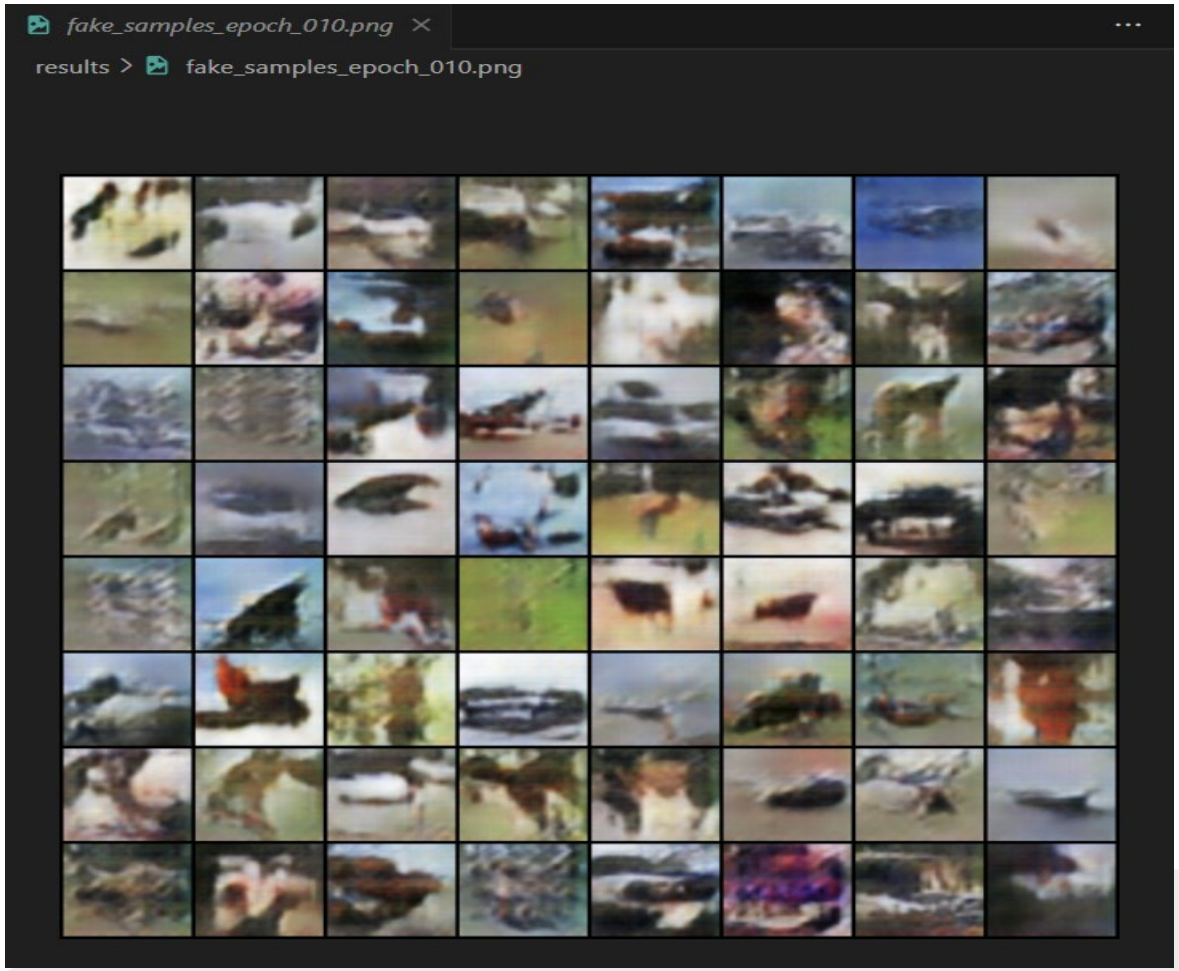


Results: Epoch 10/30

Real Image Sample

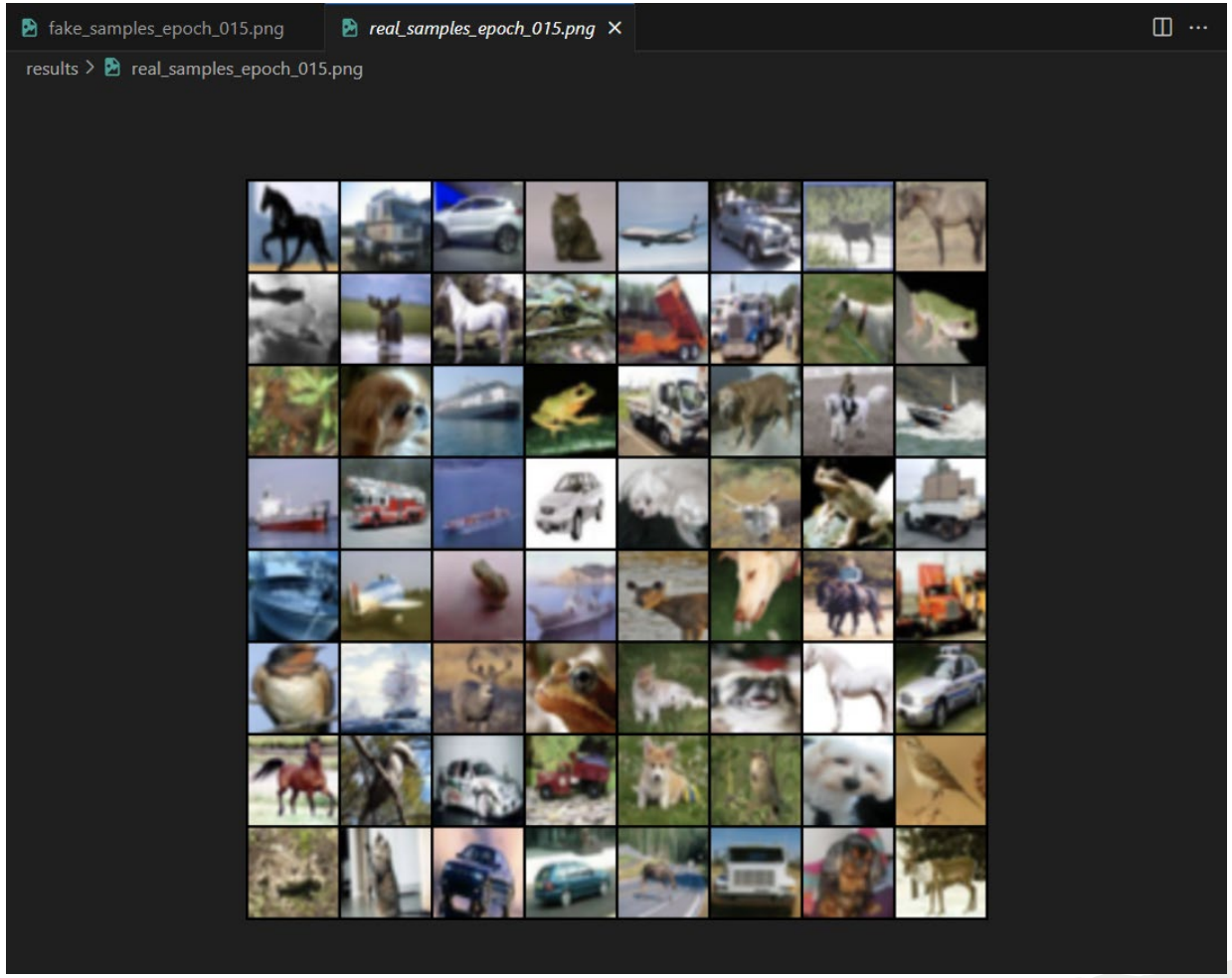


Generated Image

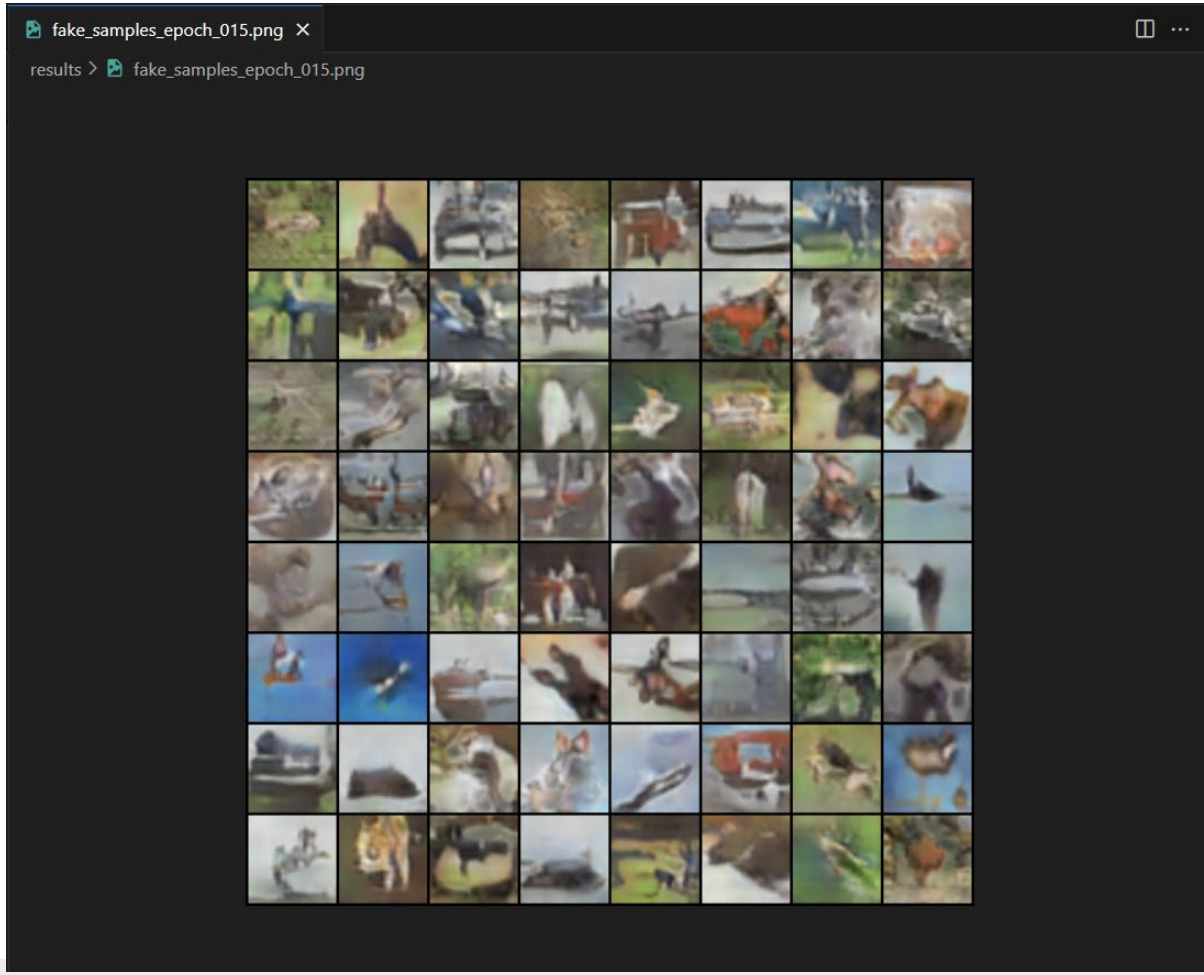


Results: Epoch 15/30

Real Image Sample

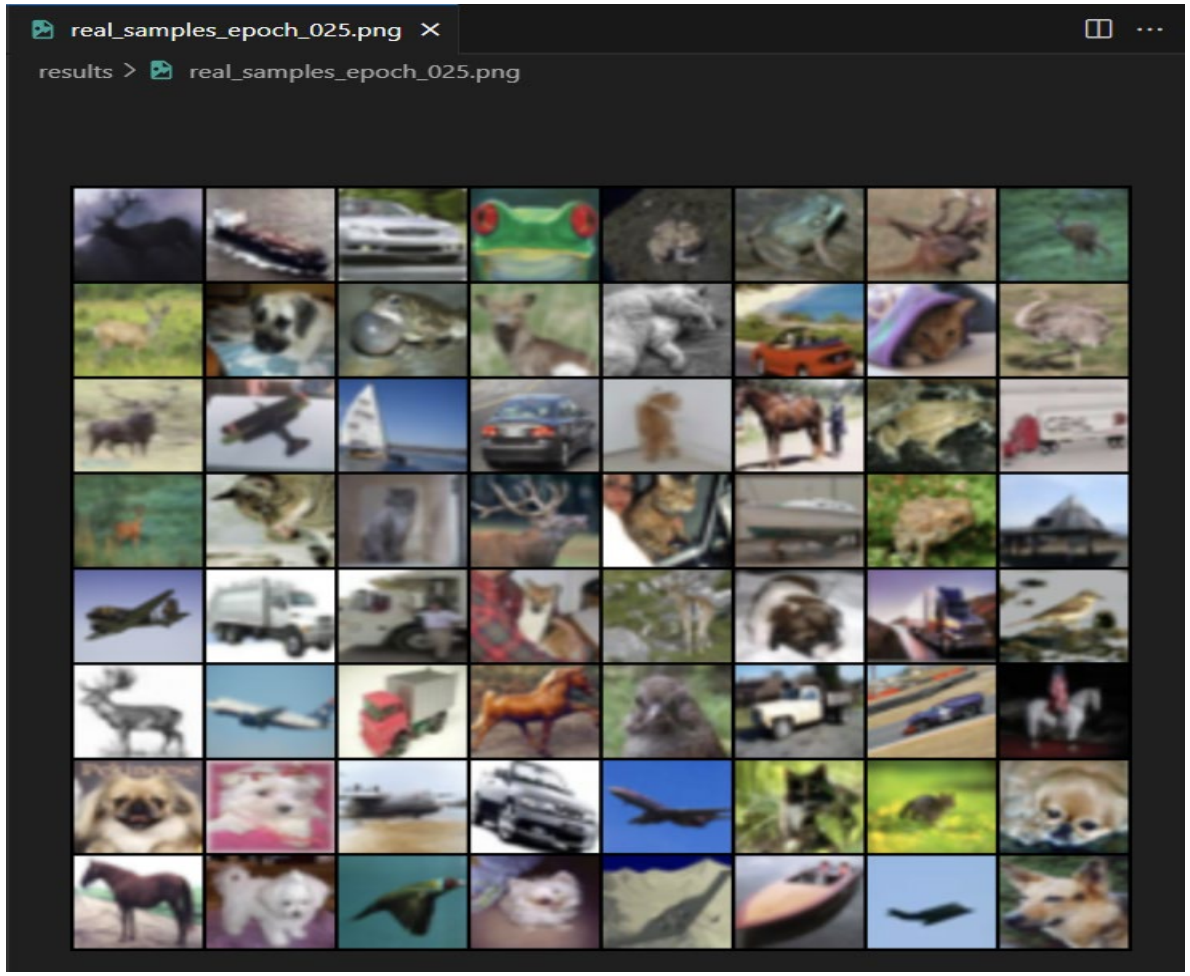


Generated Image

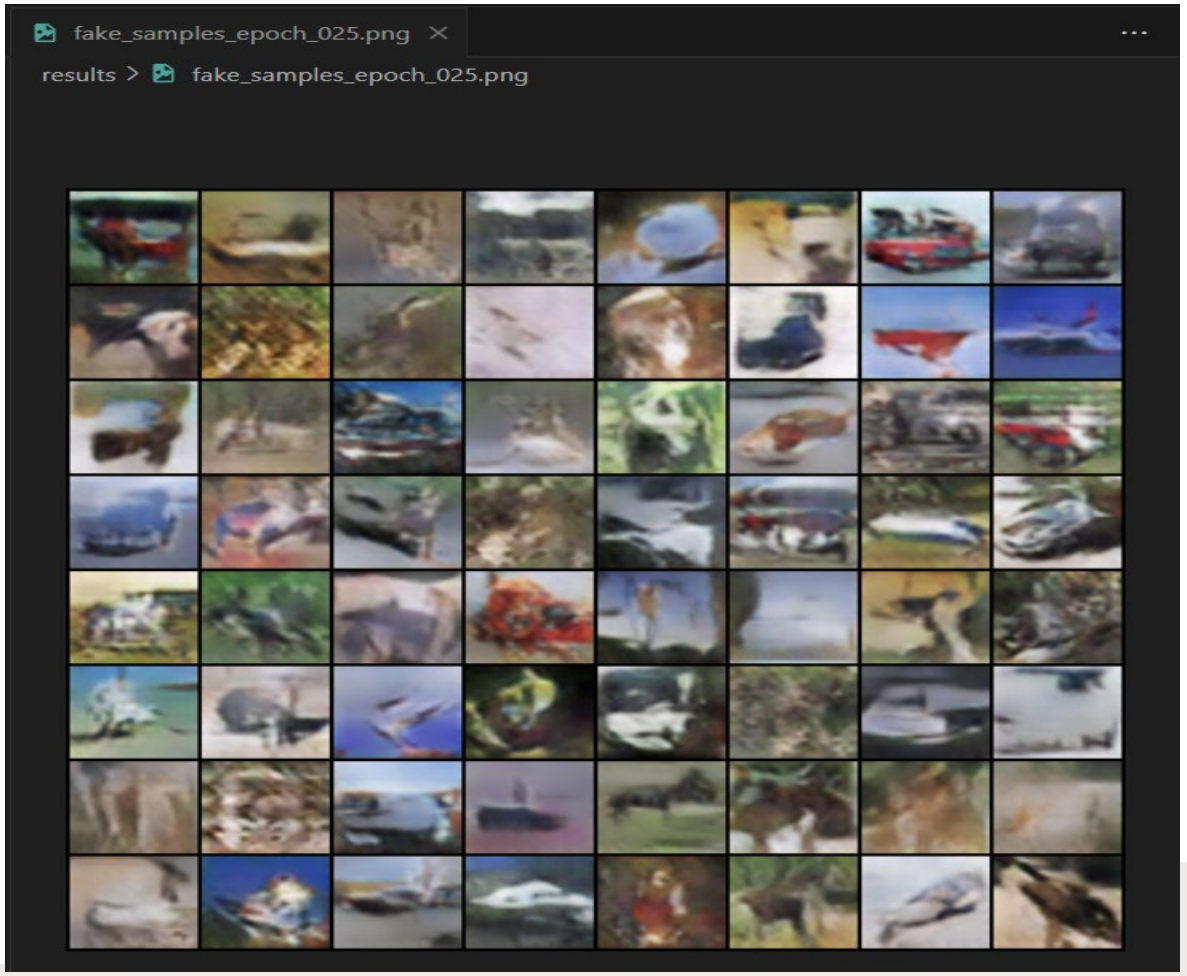


Results: Epoch 25/30

Real Image Sample

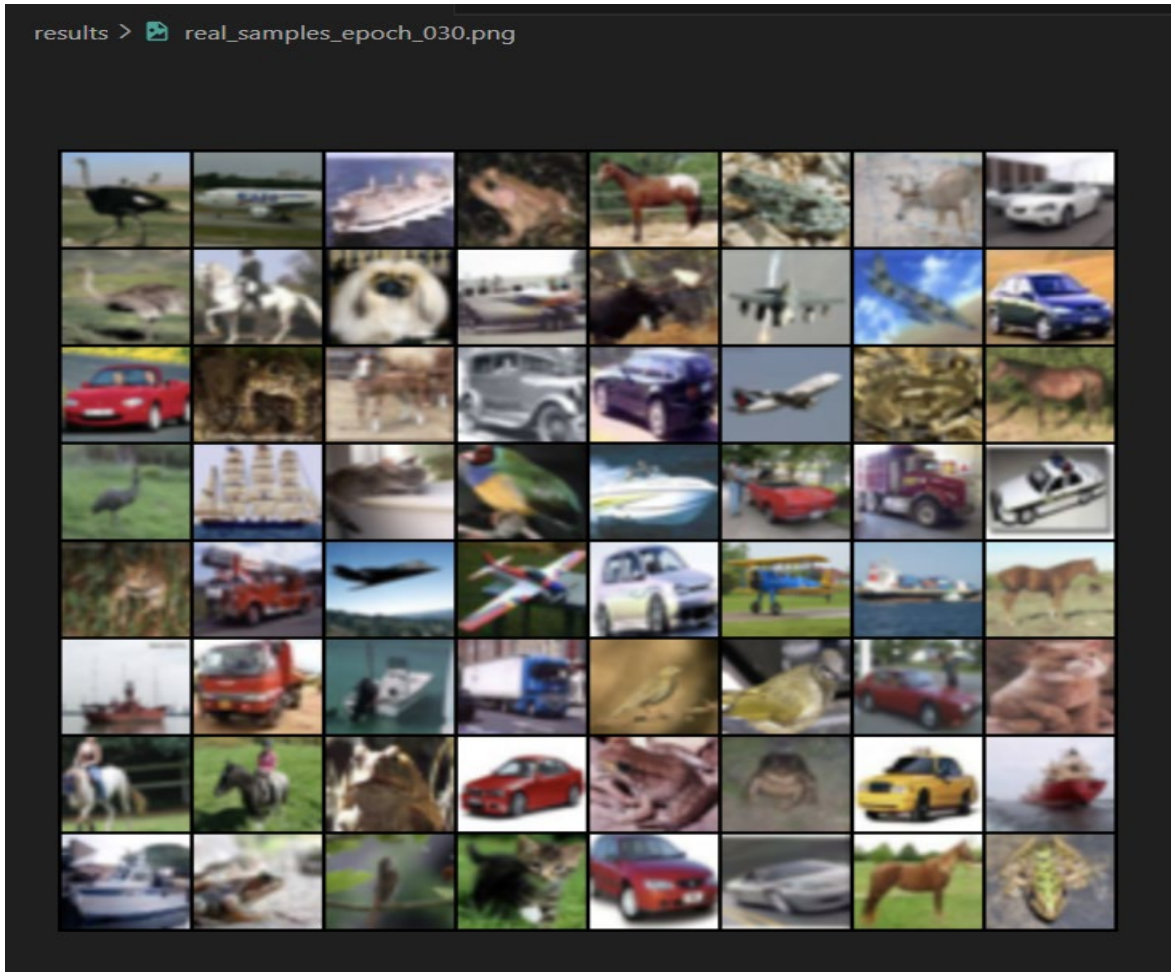


Generated Image

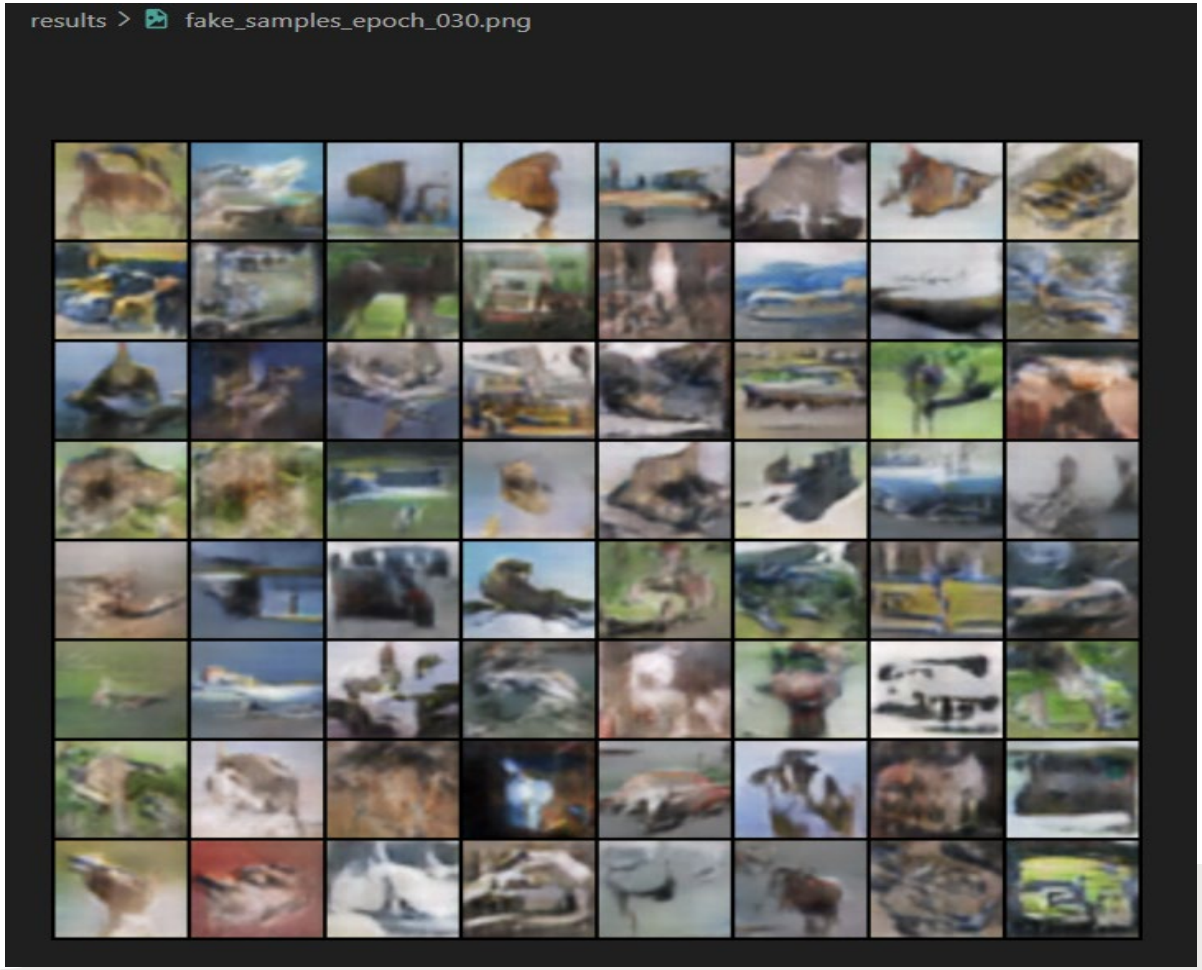


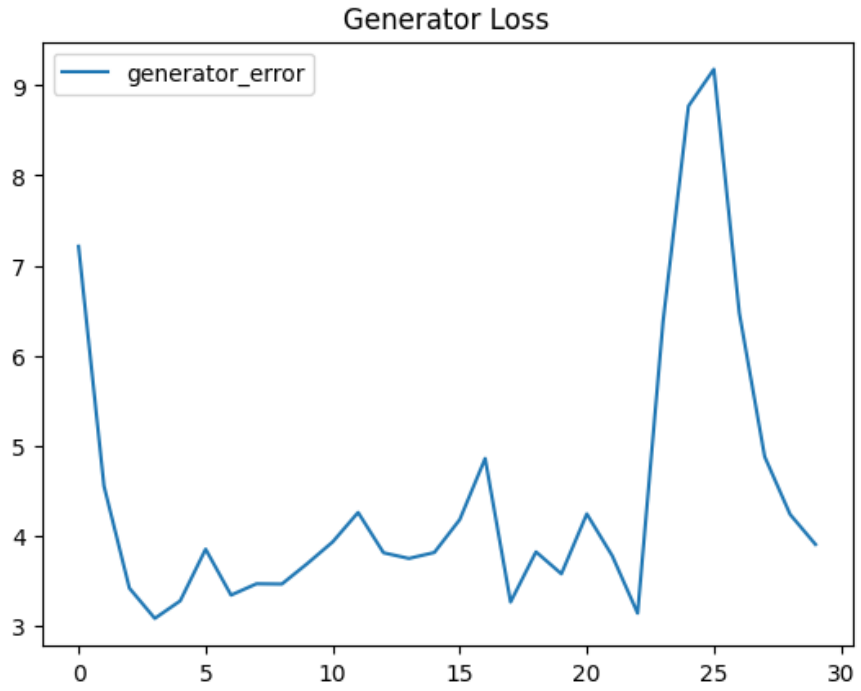
Results: Epoch 30/30

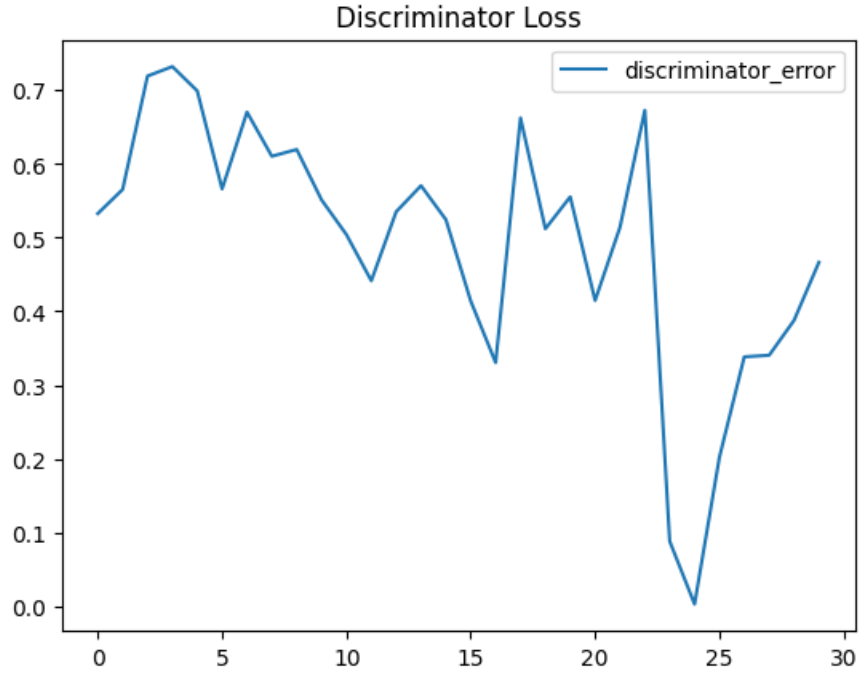
Real Image Sample



Generated Image







Evaluation

- After training for 30 epochs we get a final loss of 3.9% for Generator and 0.46% for discriminator.
- This suggests that the generator can still be refined to get Better results with more iterations and/or data

Limitations of GAN's

- **Data Quality and Quantity:**
  Limited Datasets: GANs require large and diverse datasets for effective training. Obtaining high-quality datasets can be challenging, especially for niche or specialized domains.
- **Data Imbalance:** If the dataset is imbalanced or lacks diversity, the generated samples may not accurately represent the entire data distribution.
- **Training Instability:**
  - **Mode Collapse:** GANs are prone to mode collapse, where the generator learns to generate a limited set of samples, ignoring the full diversity of the target distribution.
  - **Convergence Issues:** GANs might struggle to converge to a stable state, leading to oscillations in the training process.
- **Hyperparameter Sensitivity:**
  GANs are sensitive to the choice of hyperparameters, and finding the right set of parameters for a specific task can be a time-consuming process.
- **Compute Requirements:**
  Training GANs demands significant computational resources, including high-end GPUs or TPUs. This can be a barrier for smaller research labs or individual researchers with limited access to such resources.

Conclusions: Future Scope

GANS are still a emerging field with lots of future implementation. This model can be further used to:

Transfer Learning Exploration:

Investigate transfer learning techniques to enable the GAN model to adapt seamlessly to new datasets and domains, ensuring versatility and improved performance.

Synthetic Data Generation:

Investigate the use of the GAN model in generating synthetic data for training and testing in fields like computer vision, virtual reality, and autonomous systems, reducing the dependence on large, real-world datasets.

Image Style Transfer:

GAN’s can be further developed to mimic a specific form of art-style and create newer arts.

Explore the integration of style transfer techniques to allow users to customize the artistic style of generated images, enhancing the creative possibilities of the GAN model.

High-Resolution Image Generation:

Explore techniques to enhance the GAN's ability to generate high-resolution images or upscale lower resolution images to higher, catering to applications where fine details are crucial, such as medical imaging or digital art.

References

1. Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B. and Bharath, A.A., 2018. Generative adversarial networks: An overview. *IEEE signal processing magazine*, *35*(1), pp.53-65.
2. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y., 2014. Generative adversarial nets. *Advances in neural information processing systems*, *27*.
3. Hong, Yongjun, Uiwon Hwang, Jaeyoon Yoo, and Sungroh Yoon. "How generative adversarial networks and their variants work: An overview." *ACM Computing Surveys (CSUR)* 52, no. 1 (2019): 1-43.