Advanced Deep Learning

S. Vishali

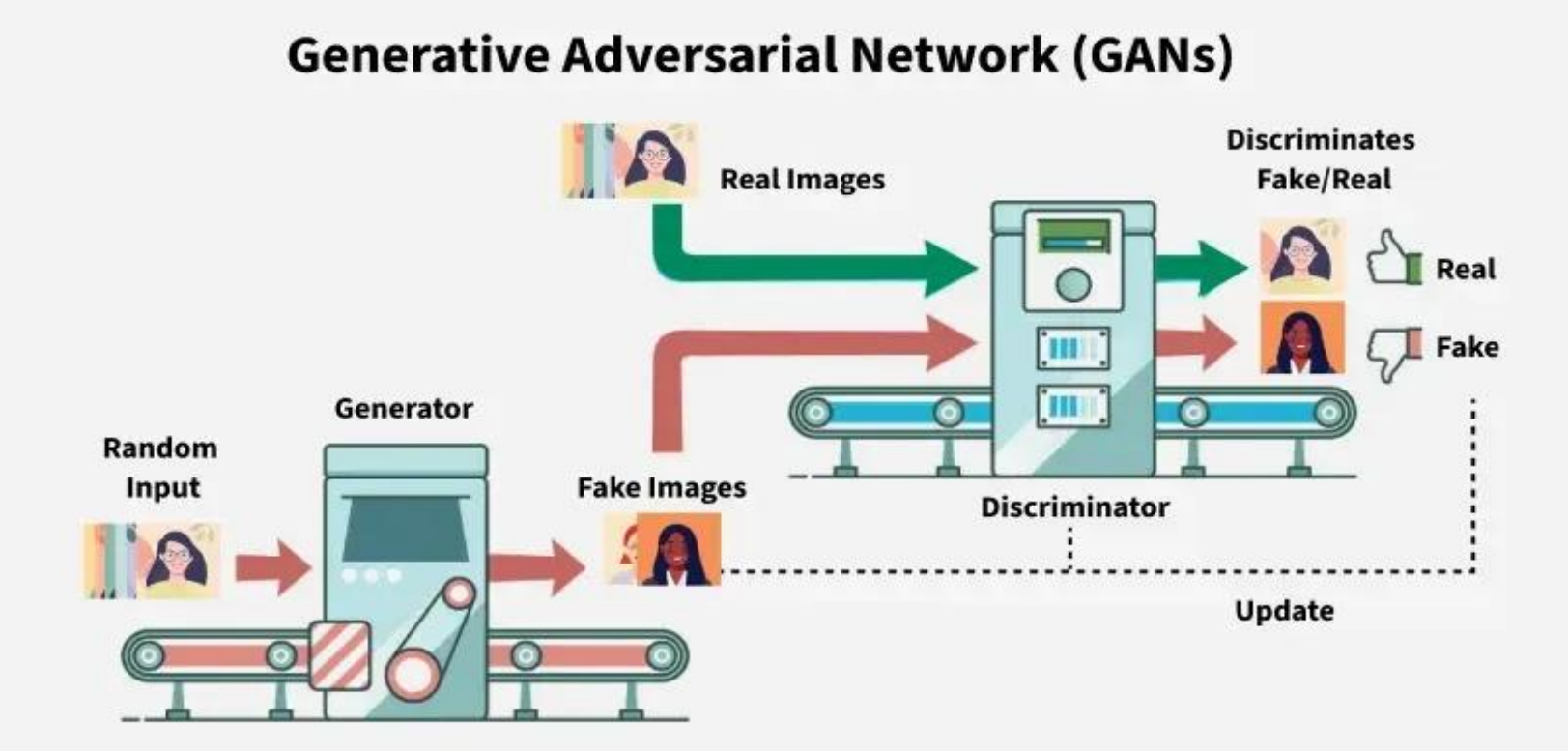
BITS Pilani

Pilani Campus

Generative Adversarial Network (GAN)

- Generative adversarial networks consist of an overall structure composed of two neural networks, one called the **generator** and the other called the **discriminator**.
- The role of the generator is to estimate the probability distribution of the real samples in order to provide generated samples resembling real data.
- The discriminator is trained to estimate the probability that a given sample came from the real data rather than being provided by the generator.
- These structures are called generative adversarial networks because the generator and discriminator are trained to compete with each other.
- The generator tries to get better at fooling the discriminator, while the discriminator tries to get better at identifying generated samples.

Example


Generative Adversarial Network (GANs)
Real Images
Discriminates
Fake/Real
Real
Fake
Generator
Random
Input
Fake Images
Discriminator
Update

Applications of GAN

- **Image Synthesis & Generation:** GANs generate realistic images, avatars and high-resolution visuals by learning patterns from training data. They are used in art, gaming and AI-driven design.
- **Image-to-Image Translation:** They can transform images between domains while preserving key features. Examples include converting day images to night, sketches to realistic images or changing artistic styles.
- **Text-to-Image Synthesis:** They create visuals from textual descriptions helps applications in AI-generated art, automated design and content creation.
- **Data Augmentation:** They generate synthetic data to improve machine learning models helps in making them more robust and generalizable in fields with limited labeled data.
- **High-Resolution Image Enhancement:** They upscale low-resolution images which helps in improving clarity for applications like medical imaging, satellite imagery and video enhancement.

Advantages of GAN

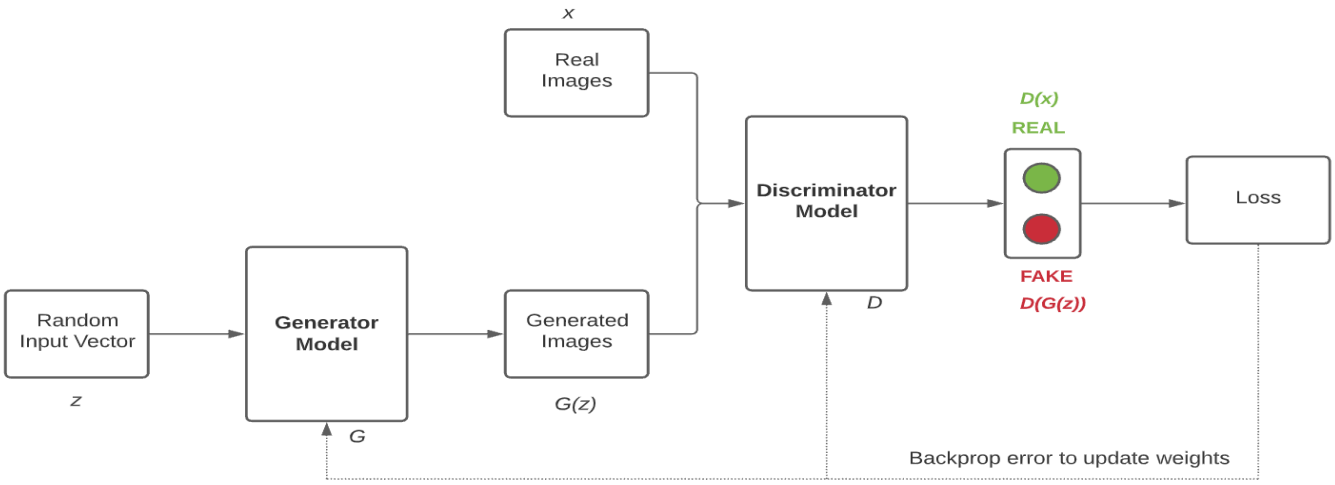
- **Synthetic Data Generation:** GANs produce new, synthetic data resembling real data distributions which is useful for augmentation, anomaly detection and creative tasks.
- **High-Quality Results**: They can generate photorealistic images, videos, music and other media with high quality.
- **Unsupervised Learning**: They don’t require labeled data helps in making them effective in scenarios where labeling is expensive or difficult.
- **Versatility**: They can be applied across many tasks including image synthesis, text-to-image generation, style transfer, anomaly detection and more.

Architecture of GAN

The Generator Model *G* takes a random input vector *z* as an input and generates the images *G(z)*.

These generated images along with the real images *x* from training data are then fed to the Discriminator Model *D*.

The Discriminator Model then classifies the images as real or fake. Then, we have to measure the loss and this loss has to be back propagated to update the weights of the Generator and the Discriminator.



- When we are training the Discriminator, we have to freeze the Generator and back propagate errors to only update the Discriminator.
- When we are training the Generator, we have to freeze the Discriminator and back propagate errors to only update the Generator.
- Thus, the Generator Model and the Discriminator Model getting better and better at each epoch.
- We have to stop training when it attains the Nash Equilibrium or $D(x) = 0.5$ *for all* x.
- In simple words, **when the generated images look almost like real images.**

1. Generator Model

The generator is a deep neural network that takes random noise as input to generate realistic data samples like images or text.

It learns the underlying data patterns by adjusting its internal parameters during training through backpropagation.

Its objective is to produce samples that the discriminator classifies as real.

Generator Loss Function: The generator tries to minimize this loss:

$$J_G = -\frac{1}{m}\Sigma_{i=1}^{m} log D(G(z_i))$$

Where,

J_G measure how well the generator is fooling the discriminator.

$G(z_i)$ is the generated sample from random noise z_i

$D\big(G(z_i)\big)$ is the discriminator's estimated probability that the generated sample is real.

The generator aims to maximize $D\big(G(z_i)\big)$ meaning it wants the discriminator to classify its fake data as real (probability close to 1).

2. Discriminator Model

The discriminator acts as a binary classifier helps in distinguishing between real and generated data.

It learns to improve its classification ability through training, refining its parameters to detect fake samples more accurately.

When dealing with image data, the discriminator uses **convolutional layers** or other relevant architectures which help to extract features and enhance the model's ability.

Discriminator Loss Function: The discriminator tries to minimize this loss:

$$J_D = -\frac{1}{m}\Sigma_{i=1}^{m} log\ D(x_i) - \frac{1}{m}\Sigma_{i=1}^{m} log(1 - D(G(z_i))$$

J_D measures how well the discriminator classifies real and fake samples.

x_i is a real data sample

$G(z_i)$ is a fake sample from the generator.

$D(x_i)$ is the discriminator's probability that x_i is real.

$D\big(G(x_i)\big)$ is the discriminator's probability that the fake sample is real.

MinMax Loss

GANs are trained using a MinMax Loss between the generator and discriminator.

Minimax is a kind of backtracking algorithm that is used in decision making and game theory to find the optimal move for a player, assuming that your opponent also plays optimally.

It is widely used in two player turn-based games such as Tic-Tac-Toe, Backgammon, Mancala, Chess, etc.
In Minimax the two players are called maximizer and minimizer. The **maximizer** tries to get the highest score possible while the **minimizer** tries to do the opposite and get the lowest score possible.

$$min_G\ max_D(G, D) = [\mathbb{E}_{x \sim p_{data}}[log\ D(x)] + \mathbb{E}_{z \sim p_z(z)}[log(1 - D(g(z)))]$$

where,

G is generator network and is D is the discriminator network

$p_{data}(x) = true\ data\ distribution$

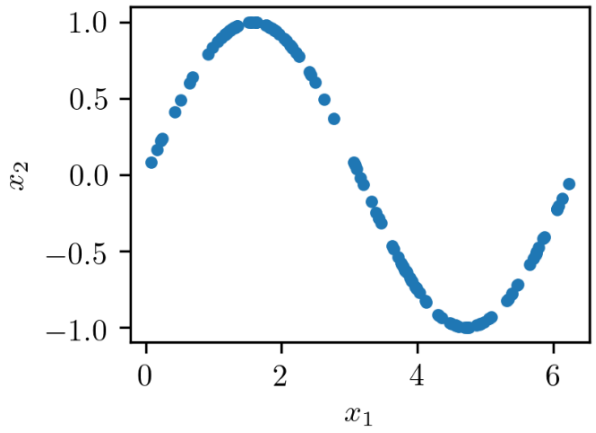
$p_z(z) = distribution\ of\ random\ \mathrm{noise}$

$D(x) = discriminator's\ estimate\ of\ real\ data$

$D\big(G(z)\big) = discriminator's\ estimate\ of\ generated\ data$

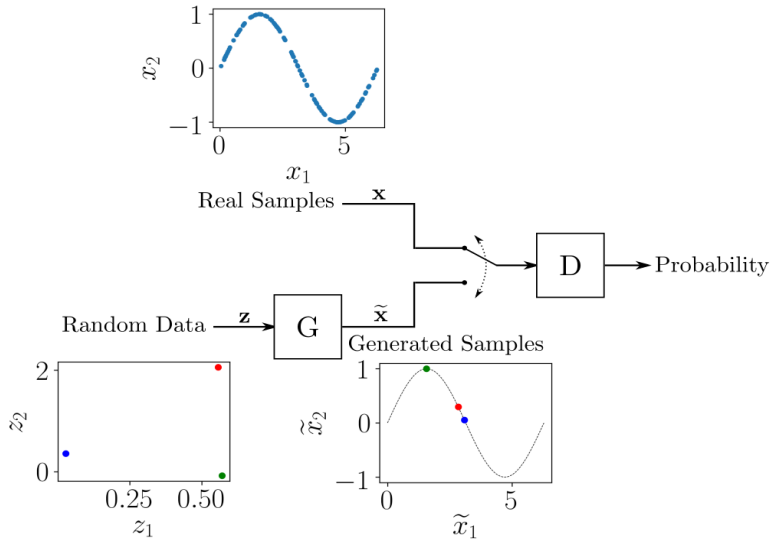
How GAN training works

consider a toy example with a dataset composed of two-dimensional samples (x_1, x_2), with x_1 in the interval from 0 to 2π and $x_2 = \sin(x_1)$, as illustrated in the following figure:



This dataset consists of points (x_1, x_2) located over a sine curve, having a very particular distribution.

The overall structure of a GAN to generate pairs $(\tilde{x}_1, \tilde{x}_2)$ resembling the samples of the dataset is shown in the following figure:



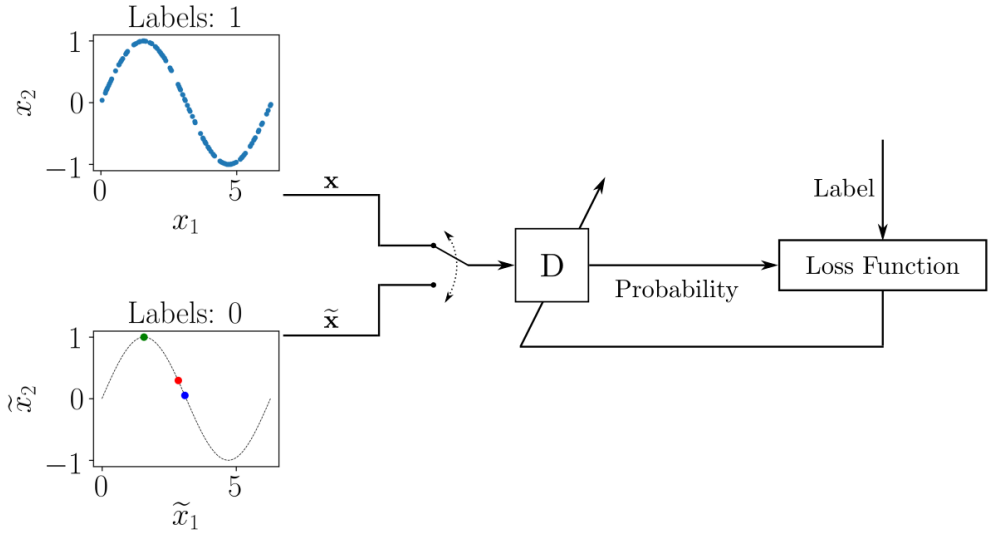
The generator *G* is fed with random data from a latent space, and its role is to generate data resembling the real samples.

In this example, it has a two-dimensional latent space, so that the generator is fed with random (z_1, z_2) pairs and is required to transform them so that they resemble the real samples.

1. The structure of the neural network *G* can be arbitrary, allowing us to use neural networks as a multilayer perceptron (MLP), a convolutional neural network (CNN), or any other structure as long as the dimensions of the input and output match the dimensions of the latent space and the real data.
2. The discriminator *D* is fed with either real samples from the training dataset or generated samples provided by *G*. Its role is to estimate the probability that the input belongs to the real dataset. The training is performed so that *D* outputs 1 when it's fed a real sample and 0 when it's fed a generated sample.
3. As with *G*, we can choose an arbitrary neural network structure for *D* as long as it respects the necessary input and output dimensions. In this example, the input is two-dimensional. For a binary discriminator, the output may be a scalar ranging from 0 to 1.

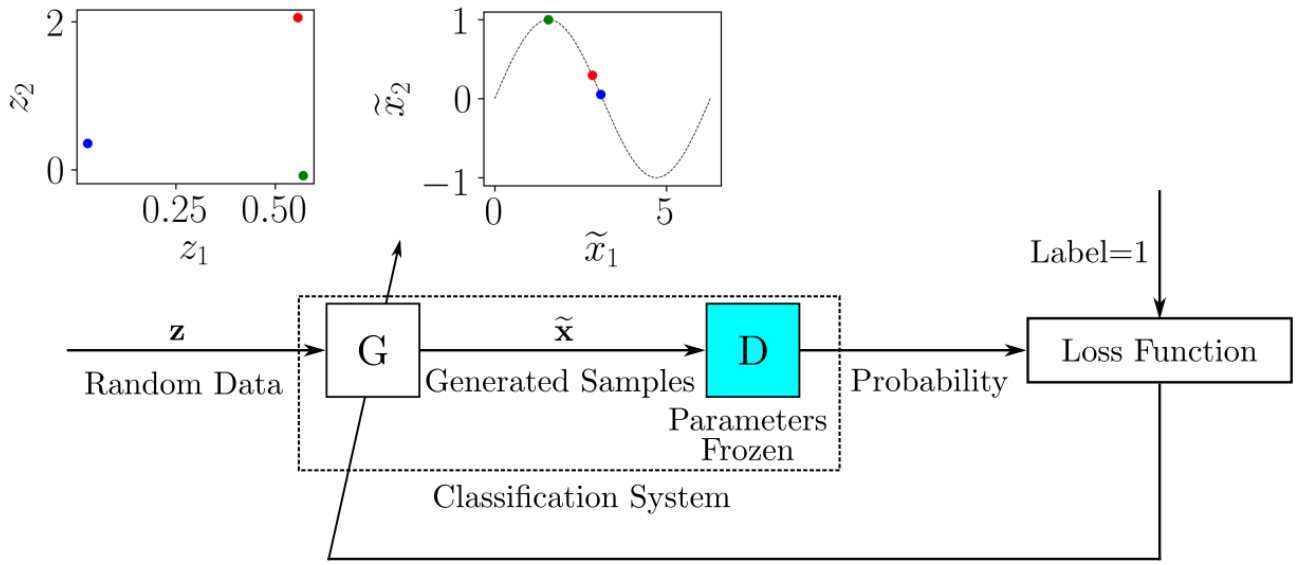
1. The GAN training process consists of a two-player minimax game in which *D* is adapted to minimize the discrimination error between real and generated samples, and *G* is adapted to maximize the probability of *D* making a mistake.
2. Although the dataset containing the real data isn't labeled, the training processes for *D* and *G* are performed in a supervised way. At each step in the training, *D* and *G* have their parameters updated. In fact, in the original GAN proposal, the parameters of *D* are updated *k* times, while the parameters of *G* are updated only once for each training step. However, to make the training simpler, you can consider *k* equal to 1.
3. To train *D*, at each iteration you label some real samples taken from the training data as 1 and some generated samples provided by *G* as 0.

We can use a conventional supervised training framework to update the parameters of *D* in order to minimize a loss function, as shown in the following scheme



For each batch of training data containing labeled real and generated samples, update the parameters of *D* to minimize a loss function. After the parameters of *D* are updated, train *G* to produce better generated samples.

1. The output of *G* is connected to *D*, whose parameters are kept frozen, as depicted here



When *G* does a good enough job to fool *D*, the output probability should be close to 1.

It could also use a conventional supervised training framework here: the dataset to train the classification system composed of *G* and *D* would be provided by random input samples, and the label associated with each input sample would be 1.

During training, as the parameters of *D* and *G* are updated, it's expected that the generated samples given by *G* will more closely resemble the real data, and *D* will have more trouble distinguishing between real and generated data.

Types of GANs

There are several types of GANs each designed for different purposes. Here are some important types:

1. Vanilla GAN

Vanilla GAN is the simplest type of GAN. It consists of:

A generator and a discriminator both are built using multi-layer perceptrons (MLPs).

The model optimizes its mathematical formulation using stochastic gradient descent (SGD).

While foundational, Vanilla GANs can face problems like:

Mode collapse: The generator produces limited types of outputs repeatedly.

Unstable training: The generator and discriminator may not improve smoothly.

2. Conditional GAN (CGAN)

Conditional GANs (CGANs) adds an additional conditional parameter to guide the generation process.

Instead of generating data randomly they allow the model to produce specific types of outputs.

Working of CGANs:

A conditional variable (y) is fed into both the generator and the discriminator.

This ensures that the generator creates data corresponding to the given condition (e.g generating images of specific objects).

The discriminator also receives the labels to help distinguish between real and fake data.

Example: Instead of generating any random image, CGAN can generate a specific object like a dog or a cat based on the label.

Architecture and Working of CGANs

Conditional GANs extend the basic GAN framework by conditioning both the generator and discriminator on additional information.

This conditioning helps to direct the generation process helps in making it more controlled and focused.

1. Generator in CGANs: The generator creates synthetic data such as images, text or videos. It takes two inputs:

Random Noise (z): A vector of random values that adds diversity to generated outputs.

Conditioning Information (y): Extra data like labels or context that guides what the generator produces for example a class label such as "cat" or "dog".

The generator combines the noise and the conditioning information to produce realistic data that matches the given condition.

For example, if the condition y is "cat" the generator will create an image of a cat.

2. Discriminator in CGANs: The discriminator is a binary classifier that decides whether input data is real or fake. It also receives two inputs:

Real Data (x): Actual samples from the dataset.

Conditioning Information (y): The same condition given to the generator.

Using both the real/fake data and the condition, the discriminator learns to judge if the data is genuine and if it matches the condition.

For example, if the input is an image labeled "cat" the discriminator verifies whether it truly looks like a real cat.

3. Interaction Between Generator and Discriminator: The generator and discriminator train together through adversarial training:

The generator tries to create fake data based on noise (z) and condition (y) that can fool the discriminator.

The discriminator attempts to correctly classify real vs. fake data considering the condition (y).

The goal of the adversarial process is:

Generator: Produce data that the discriminator believes is real.

Discriminator: Accurately distinguish between real and fake data.

4. Loss Function and Training: Training is guided by a loss function that balances the generator and discriminator:

$$min_G max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[logD(x|y)] + \mathbb{E}_{z \sim p_z}(z)[log(1 - D(G(z \mid y)))]$$

The first term encourages the discriminator to classify real samples correctly.

The second term pushes the generator to produce samples that the discriminator classifies as real.

Here $\mathbb{E}$ represents the expected value p_{data} is the real data distribution and p_z is the prior noise distribution.

As training progresses both the generator and discriminator improve. This adversarial process results in the generator producing more realistic data conditioned on the input information.

3. Deep Convolutional GAN (DCGAN)

Deep Convolutional GANs (DCGANs) are among the most popular types of GANs used for image generation.

They are important because they:

Uses Convolutional Neural Networks (CNNs) instead of simple multi-layer perceptrons (MLPs).

Max pooling layers are replaced with convolutional stride helps in making the model more efficient.

Fully connected layers are removed, which allows for better spatial understanding of images.

DCGANs are successful because they generate high-quality, realistic images.

DCGANs are introduced to reduce the problem of mode collapse.

Mode collapse occurs when the generator got biased towards a few outputs and can't able to produce outputs of every variation from the dataset.

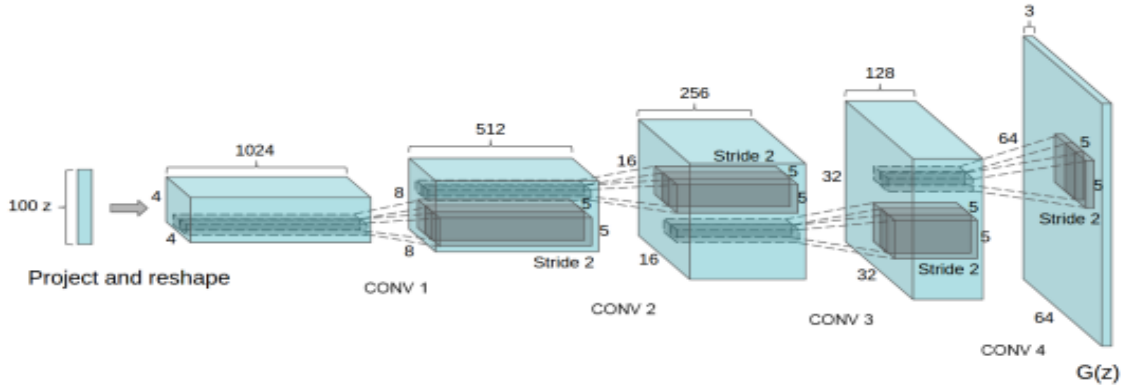
For Example- MNIST digits dataset (digits from 0 to 9)

we want the generator should generate all type of digits but sometimes our generator got biased towards two to three digits and produce them only.

Because of that the discriminator also got optimized towards that particular digits only, and this state is known as mode collapse.

But this problem can be overcome by using DCGANs.

Architecture:



The generator of the DCGAN architecture takes 100 uniform generated values using normal distribution as an input.

First, it changes the dimension to 4x4x1024 and performed a fractionally stridden convolution 4 times with a stride of 1/2 (this means every time when applied, it doubles the image dimension while reducing the number of output channels).

The generated output has dimensions of (64, 64, 3). There are some architectural changes proposed in the generator such as the removal of all fully connected layers, and the use of Batch Normalization which helps in stabilizing training.

The role of the discriminator here is to determine that the image comes from either a real dataset or a generator. The discriminator can be simply designed similar to a convolution neural network that performs an image classification task

4. Laplacian Pyramid GAN (LAPGAN)

Laplacian Pyramid GAN (LAPGAN) is designed to generate ultra-high-quality images by using a multi-resolution approach.

Working of LAPGAN:

Uses multiple generator-discriminator pairs at different levels of the Laplacian pyramid.

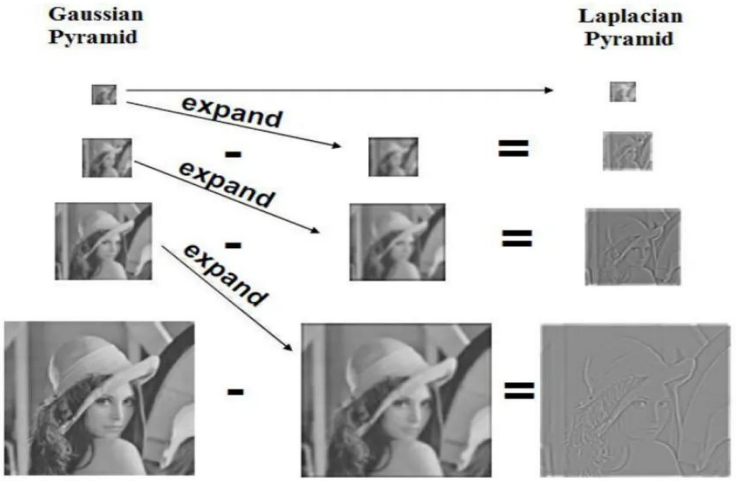
Images are first down sampled at each layer of the pyramid and upscaled again using Conditional GANs (CGANs).

This process allows the image to gradually refine details and helps in reducing noise and improving clarity.

Due to its ability to generate highly detailed images, LAPGAN is considered a superior approach for photorealistic image generation.

LAPGAN combines the CGAN model with a Laplacian pyramid representation.

The Laplacian pyramid is a linear invertible image representation consisting of a set of band-pass images, spaced an octave apart, plus a low-frequency residual.



d(.): **Downsampling** operation, d(*I*), with image *I* of size *j*×*j* got a new image of size *j*/2×*j*/2.

u(.): **Upsampling** operation, u(*I*), with image *I* of size *j*×*j* got a new image of size 2*j*×2*j*.

A Gaussian pyramid *G*(*I*) = [*I*_0, *I*_1, …, *I*_*K*] is built, where *I*_0 = *I* and I_*k* is *k* repeated applications of *d*(.) to *I*. *K* is the number of levels in the pyramid, selected so that the final level has very small spatial extent (≤8×8 pixels).

The coefficients *h_k* at each level *k* of the Laplacian pyramid *L*(*I*) are constructed by taking **the difference between adjacent levels in the Gaussian pyramid, upsampling the smaller one with u(.)**:

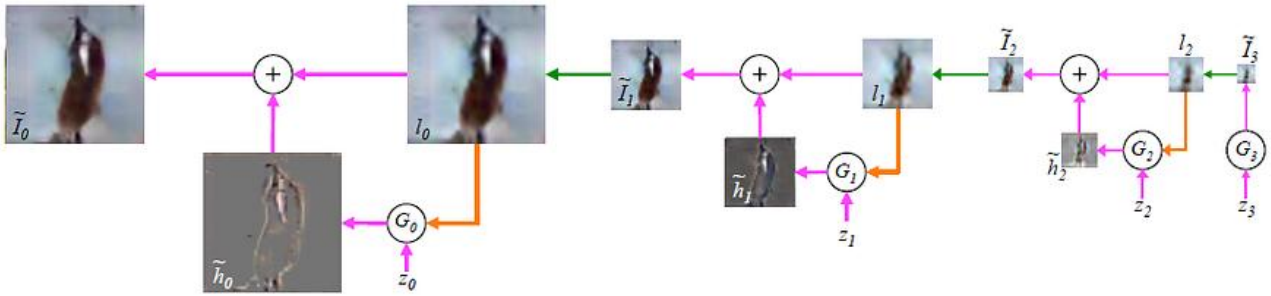
$$h_k = \mathcal{L}_k(I) = \mathcal{G}_k(I) - u(\mathcal{G}_{k+1}(I)) = I_k - u(I_{k+1})$$

Thus, reconstruction from a Laplacian pyramid coefficients [*h*_1, …, *h_K*] is performed using the backward recurrence:

$$I_k = u(I_{k+1}) + h_k$$

By repeating, we can get back to the full resolution image.

Generative Network



First, starts by:

$$\tilde{I}_{K+1} = 0$$

Thus, the image at *K* is only generated by using noise vector *z_K*:

$$\tilde{I}_K = G_K(z_K).$$

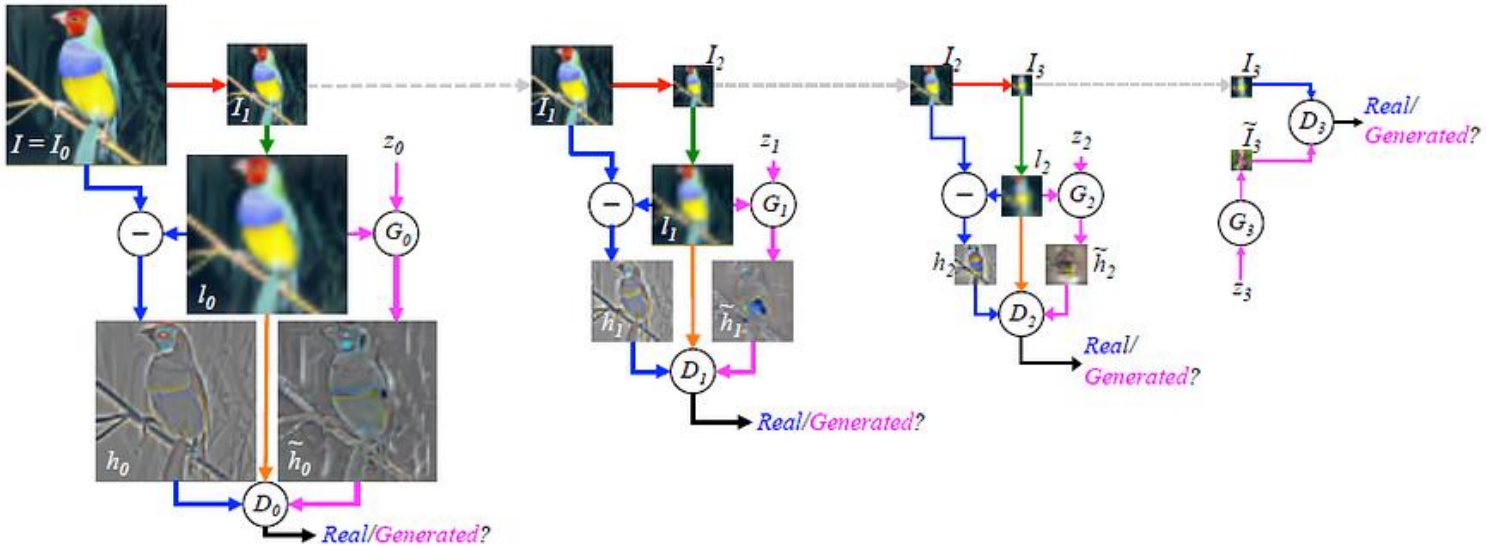
After that, models at all levels take an upsampled version of the current image ~*I_k*+1 as a conditioning variable, in addition to the noise vector *z_k*, as shown above to output ~*h_k*:

$$\tilde{h}_k = G_k(z_k, u(I_{k+1}))$$

Specifically, the above figure shows a pyramid with *K* = 3 using 4 generative models to sample a 64×64 image.

The generative models {*G*_0, …,*G_K*} are trained using the CGAN approach at each level of the pyramid, where *G_k* is CNN.

Discriminative Network



D_k takes as input *h_k* or *~h_k*, and predicts if the image is real or generated.

Using LAPGAN, the generation can be broken into successive refinements which is the key idea in LAPGAN, that can focus on making each step plausible.

5. Super Resolution GAN (SRGAN)

Super-Resolution GAN (SRGAN) is designed to increase the resolution of low-quality images while preserving details.

Working of SRGAN:

Uses a deep neural network combined with an adversarial loss function.

Enhances low-resolution images by adding finer details helps in making them appear sharper and more realistic.

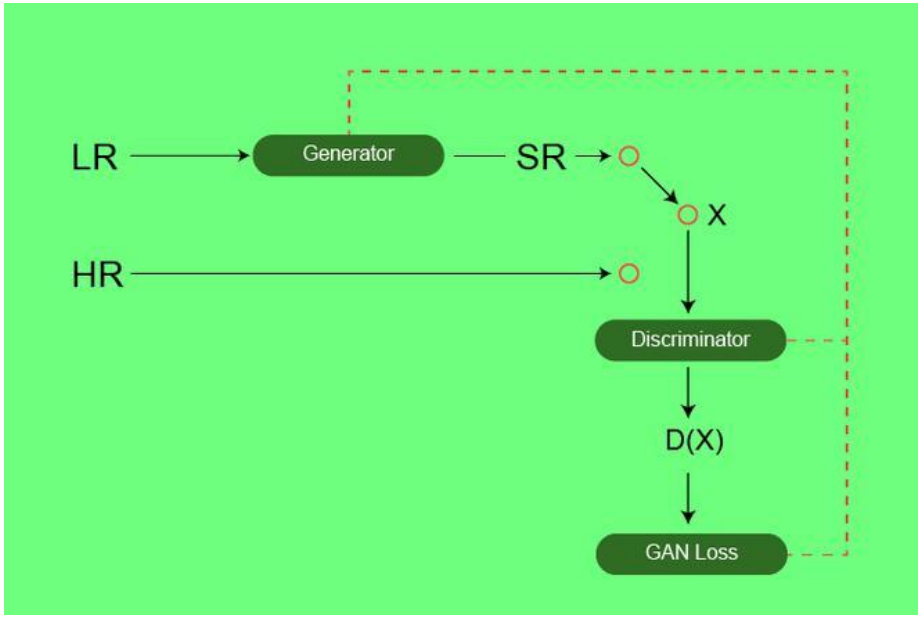
Helps to reduce common image upscaling errors such as blurriness and pixelation.

SRGAN was proposed by researchers at Twitter. The motive of this architecture is to recover finer textures from the image when we upscale it so that it's quality cannot be compromised.

There are other methods such as Bilinear Interpolation that can be used to perform this task but they suffer from image information loss and smoothing.

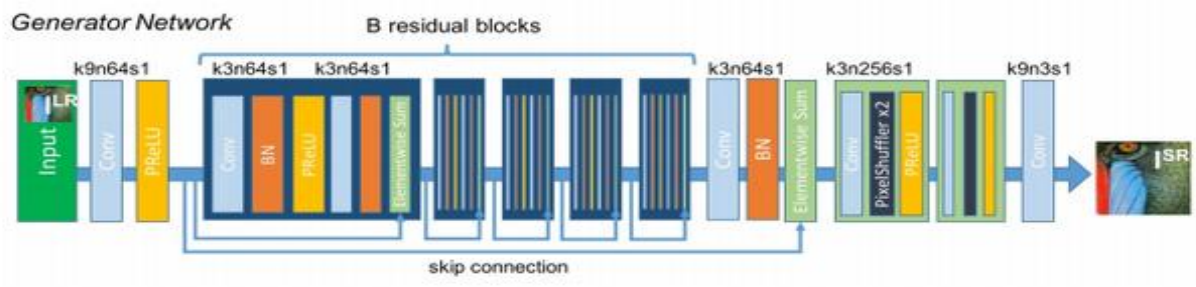
In this paper, the authors proposed two architectures the one without GAN (SRResNet) and one with GAN (SRGAN). It is concluded that SRGAN has better accuracy and generate image more pleasing to eyes as compared to SRGAN.

Architecture: Similar to GAN architectures, the Super Resolution GAN also contains two parts Generator and Discriminator where generator produces some data based on the probability distribution and discriminator tries to guess weather data coming from input dataset or generator. Generator than tries to optimize the generated data so that it can fool the discriminator. Below are the generator and discriminator architectural details:



Generator Architecture:

There are B residual blocks (16), originated by ResNet. Within the residual block, two convolutional layers are used, with small 3×3 kernels and 64 feature maps followed by batch-normalization layers and ParametricReLU as the activation function.



The resolution of the input image is increased with two trained sub-pixel convolution layers.

This generator architecture also uses parametric ReLU as an activation function which instead of using a fixed value for a parameter of the rectifier (alpha) like LeakyReLU. It adaptively learns the parameters of rectifier and improves the accuracy at negligible extra computational cost

The generator architecture contains residual network instead of deep convolution networks because residual networks are easy to train and allows them to be substantially deeper in order to generate better results. This is because the residual network used a type of connections called skip connections.

Discriminator Architecture:

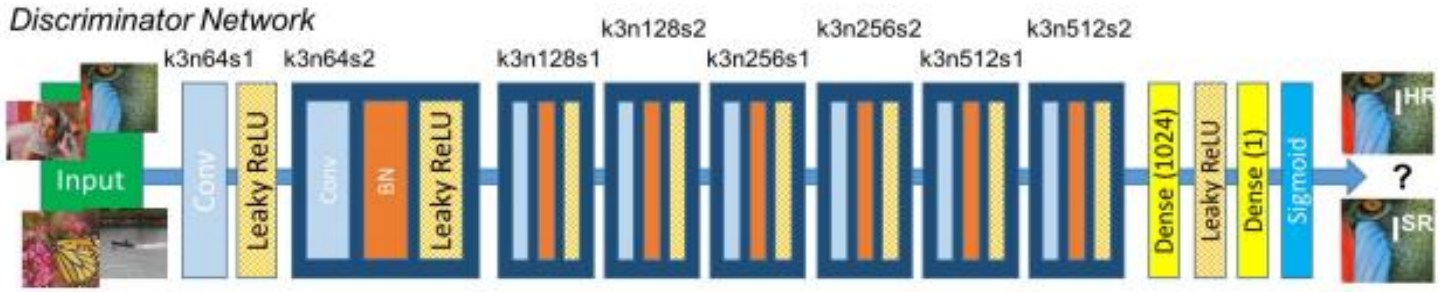
The task of the discriminator is to discriminate between real HR images and generated SR images.

The discriminator architecture used in this paper is similar to DC- GAN architecture with LeakyReLU as activation.

The network contains eight convolutional layers with of 3×3 filter kernels, increasing by a factor of 2 from 64 to 512 kernels.

Strided convolutions are used to reduce the image resolution each time the number of features is doubled.

The resulting 512 feature maps are followed by two dense layers and a leakyReLU applied between and a final sigmoid activation function to obtain a probability for sample classification



Loss Function:

The SRGAN uses perpetual loss function (L_{SR}) which is the weighted sum of two loss components : content loss and adversarial loss. This loss is very important for the performance of the generator architecture:

Content Loss: We use two types of content loss in this paper : pixelwise MSE loss for the SRResnet architecture, which is most common MSE loss for image Super Resolution. However MSE loss does not able to deal with high frequency content in the image that resulted in producing overly smooth images. Therefore the authors of the paper decided to use loss of different VGG layers. This VGG loss is based on the ReLU activation layers of the pre-trained 19 layer VGG network.

$$l^{SR}_{MSE} = \frac{1}{r^2WH}\sum_{x=1}^{rW}\sum_{y=1}^{rH}(I^{HR}_{x,y} - G_{\theta_G}(I^{LR})_{x,y})^2$$

$$l^{SR}_{VGG/i.j} = \frac{1}{W_{i,j}H_{i,j}}\sum_{x=1}^{W_{i,j}}\sum_{y=1}^{H_{i,j}}(\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}(G_{\theta_G}(I^{LR}))_{x,y})^2$$

Adversarial Loss: The Adversarial loss is the loss function that forces the generator to image more similar to high resolution image by using a discriminator that is trained to differentiate between high resolution and super resolution images.

$$l_{Gen}^{SR} = \sum_{n=1}^{N} -\log D_{\theta_D}(G_{\theta_G}(I^{LR}))$$

Therefore, total content loss of this architecture will be

$$l^{SR} = \underbrace{\underbrace{l_{X}^{SR}}_{\text{content loss}} + \underbrace{10^{-3} l_{Gen}^{SR}}_{\text{adversarial loss}}}_{\text{perceptual loss (for VGG based content losses)}}$$

Thank you