

# Generative Adversarial Networks

By

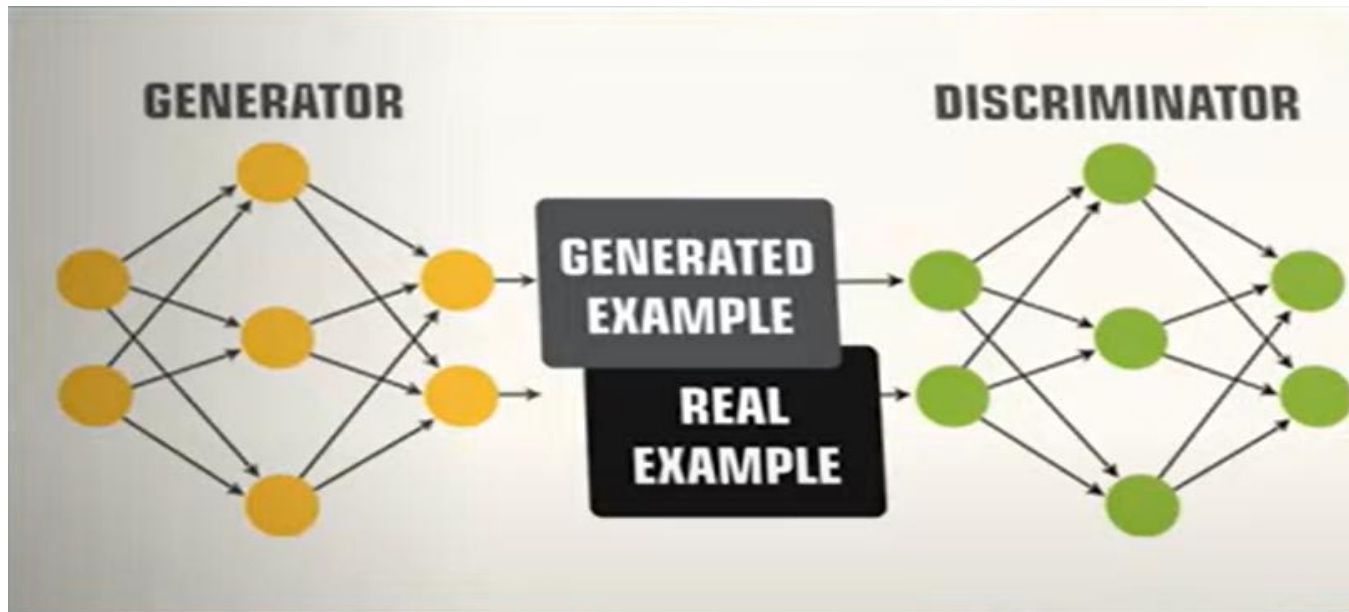
Prof(Dr) Premanand P Ghadekar



<https://realpython.com/generative-adversarial-networks/>

# Outline

- ❖ What are Generative Adversarial Networks
- ❖ Generator
- ❖ Discriminator
- ❖ How GANs work
- ❖ Types of GANs
- ❖ Application of GANs



# What is a GAN?

---

## ❑ What is a GAN?

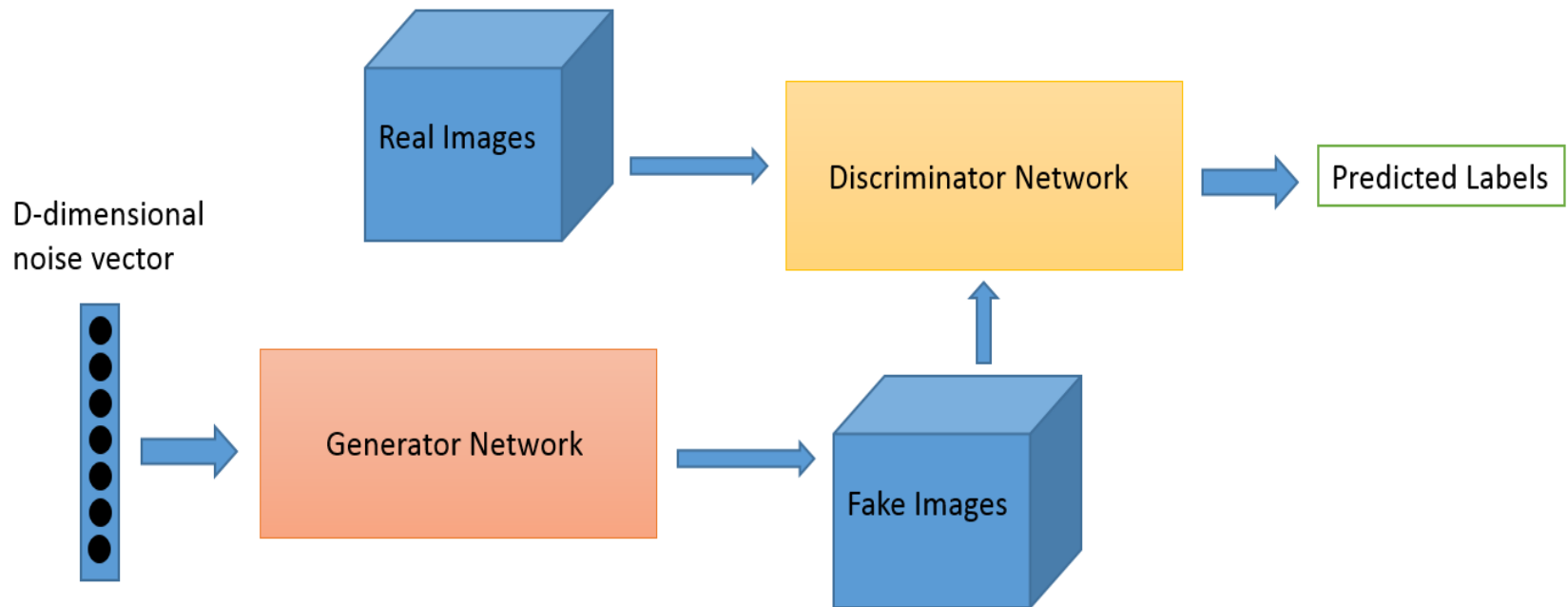
Generative Adversarial Network is an architecture for training a generative model. It was developed and introduced by **Ian J. Goodfellow in 2014.**

## ❑ The architecture is comprised of two models:

- Generator
  - Discriminator
- 
- GANs are neural networks that learn to create duplicate data similar to a known input data.
  - GAN has two components with competing objectives that work through opposite goals.
  - The generator tries to produce perfect data and the discriminator tries to find the defection in the data generated by the generator.

# What is a GAN?

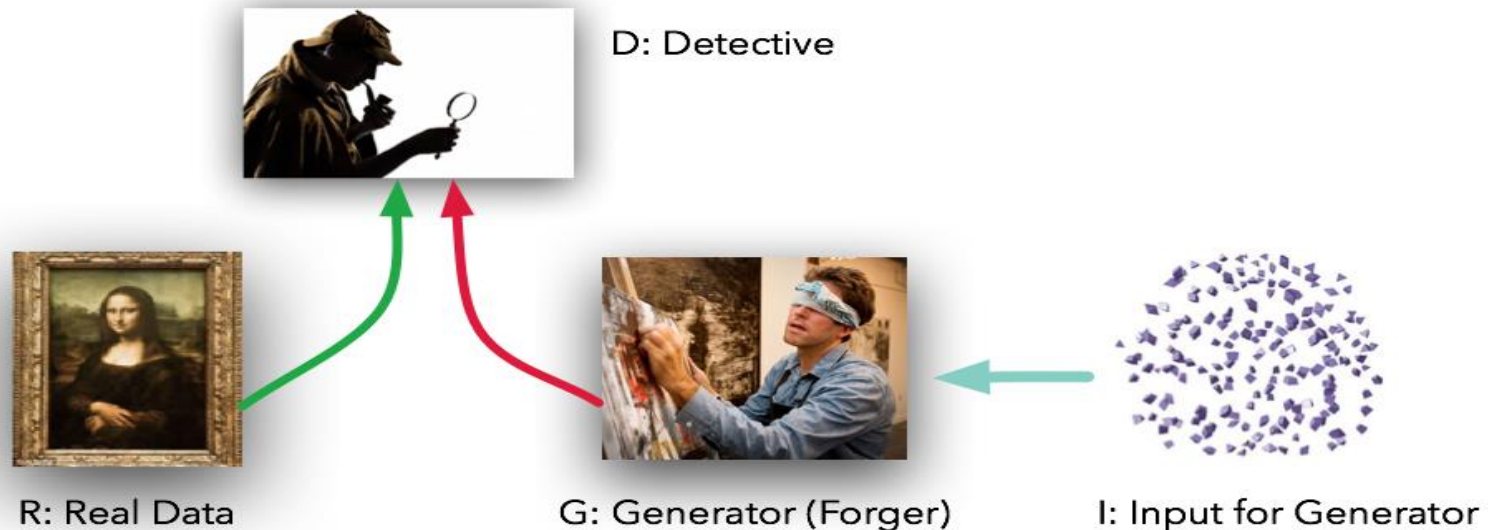
- Generative adversarial networks consist of two models: a **generative model** and a **discriminative model**.



<https://iq.opengenus.org/beginners-guide-to-generative-adversarial-networks/>

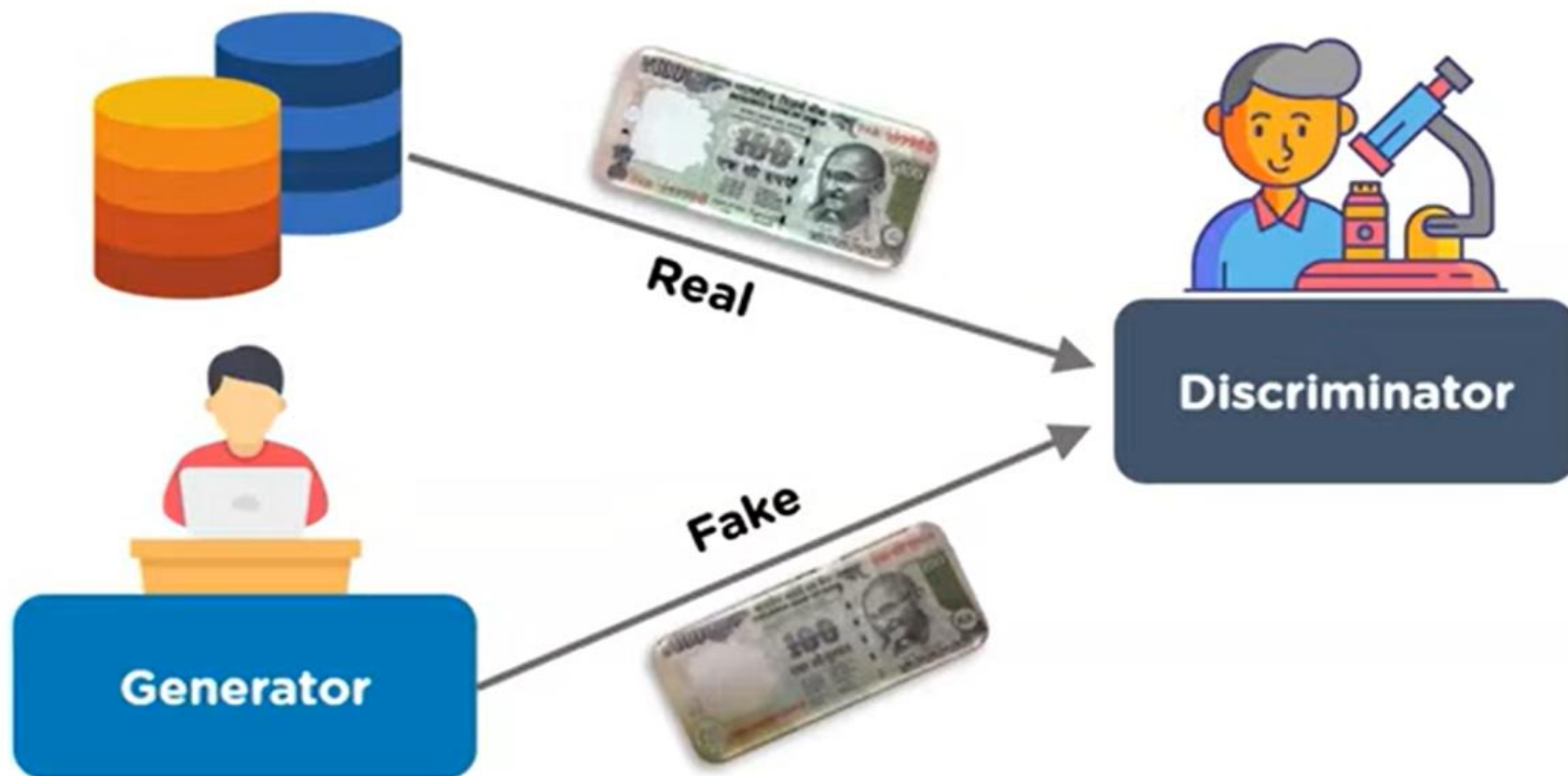
# What is a GAN?

- ❑ **The discriminator model is a classifier** that has the task of determining whether a given image looks natural (an image from the dataset) or looks like it has been artificially created.
- ❑ This is basically a **binary classifier** that will take the form of a **normal convolutional neural network (CNN)**.
- ❑ The generator model takes random input values and transforms them into images through a deep convolutional neural network.



# What are Generative Adversarial Networks?

**Generative Adversarial Networks** consist of two models that **compete** to analyze, capture and copy the variations within a dataset.

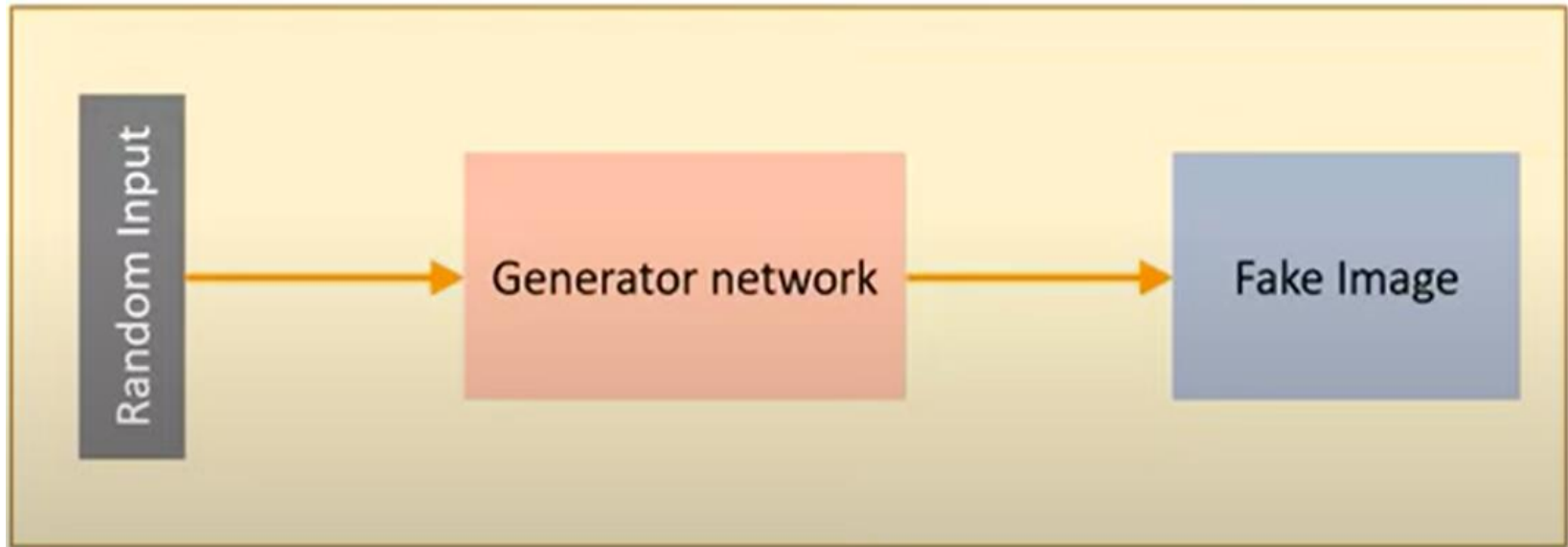


<https://www.simplilearn.com/tutorials/deep-learning-tutorial/generative-adversarial-networks-gans>

# Generator

---

**Generator Network** that takes a sample and generates a sample of data.  
The Generator in GAN learns to create fake data by incorporating feedback from the Discriminator.

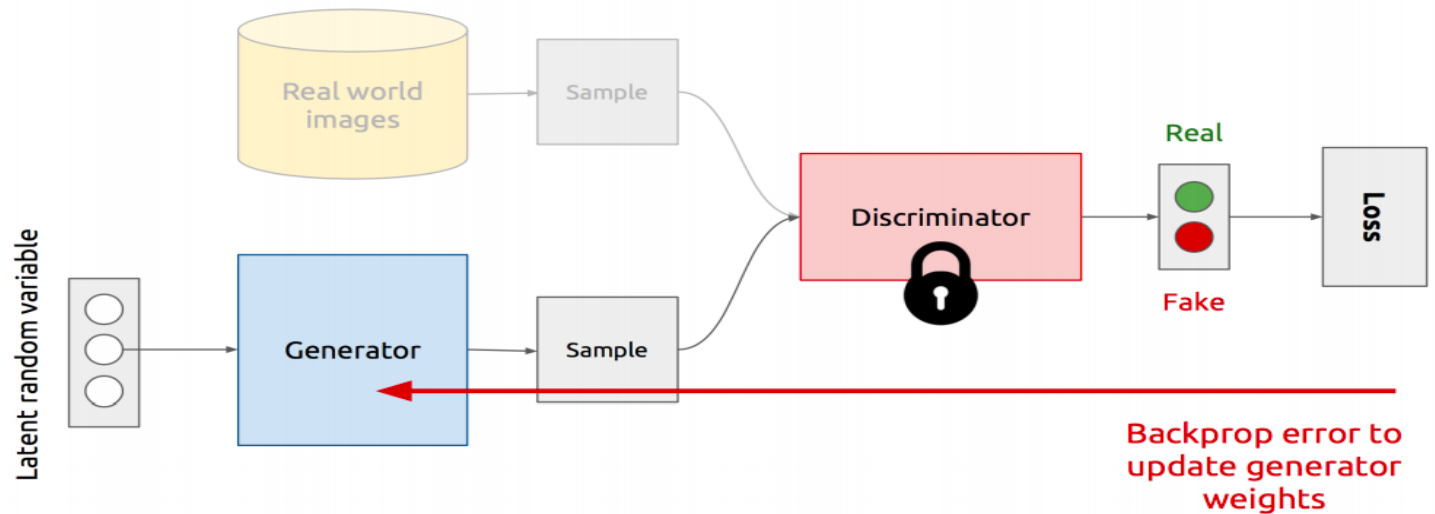


<https://www.simplilearn.com/tutorials/deep-learning-tutorial/generative-adversarial-networks-gans>

# Generator Training

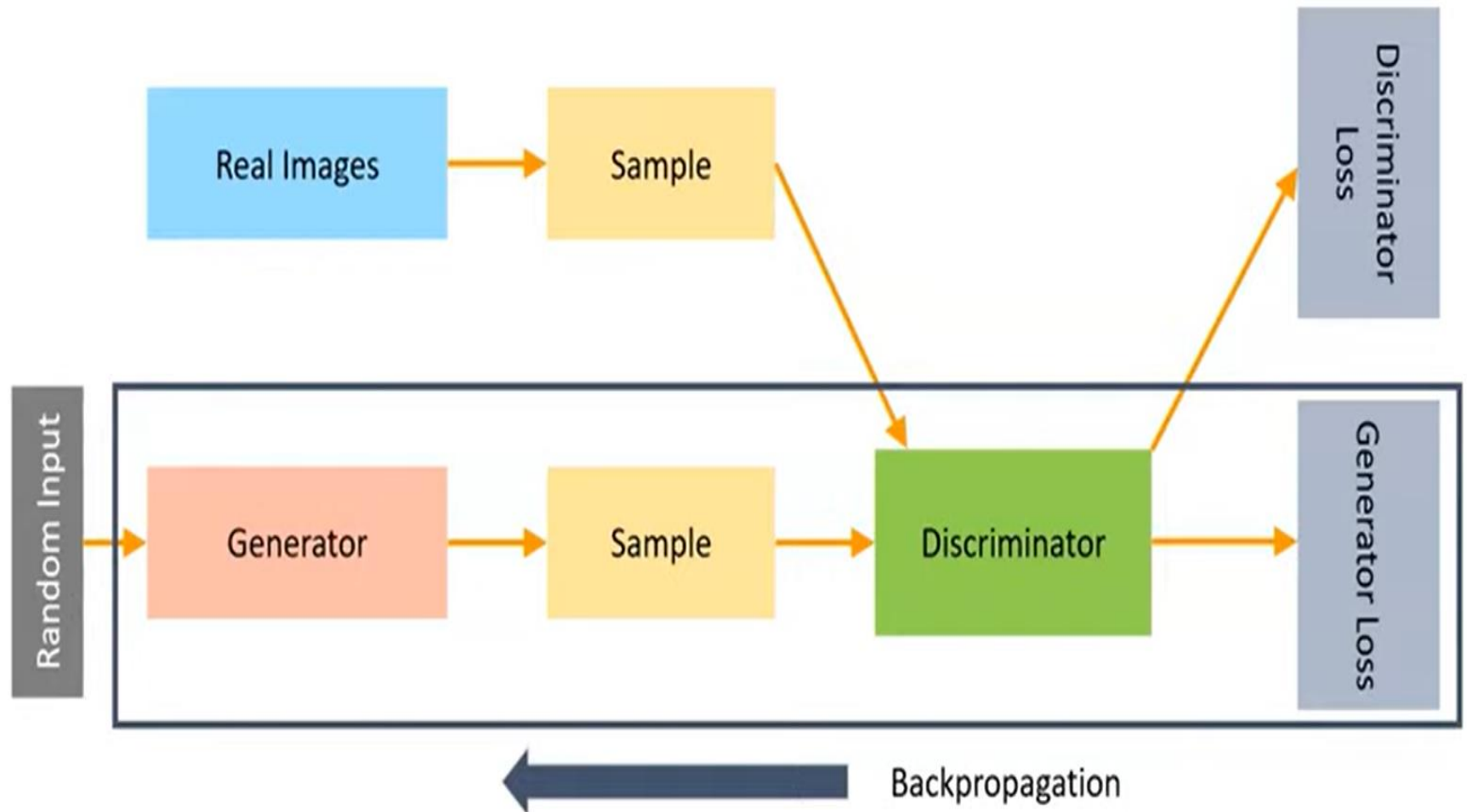
- The Generator is trained while the Discriminator is idle.
- After the Discriminator is trained by the generated fake data of the Generator, we can use its predictions to update the weights for Generator and make it smarter at fooling the Discriminator.

## Training Generator





# Generator Training

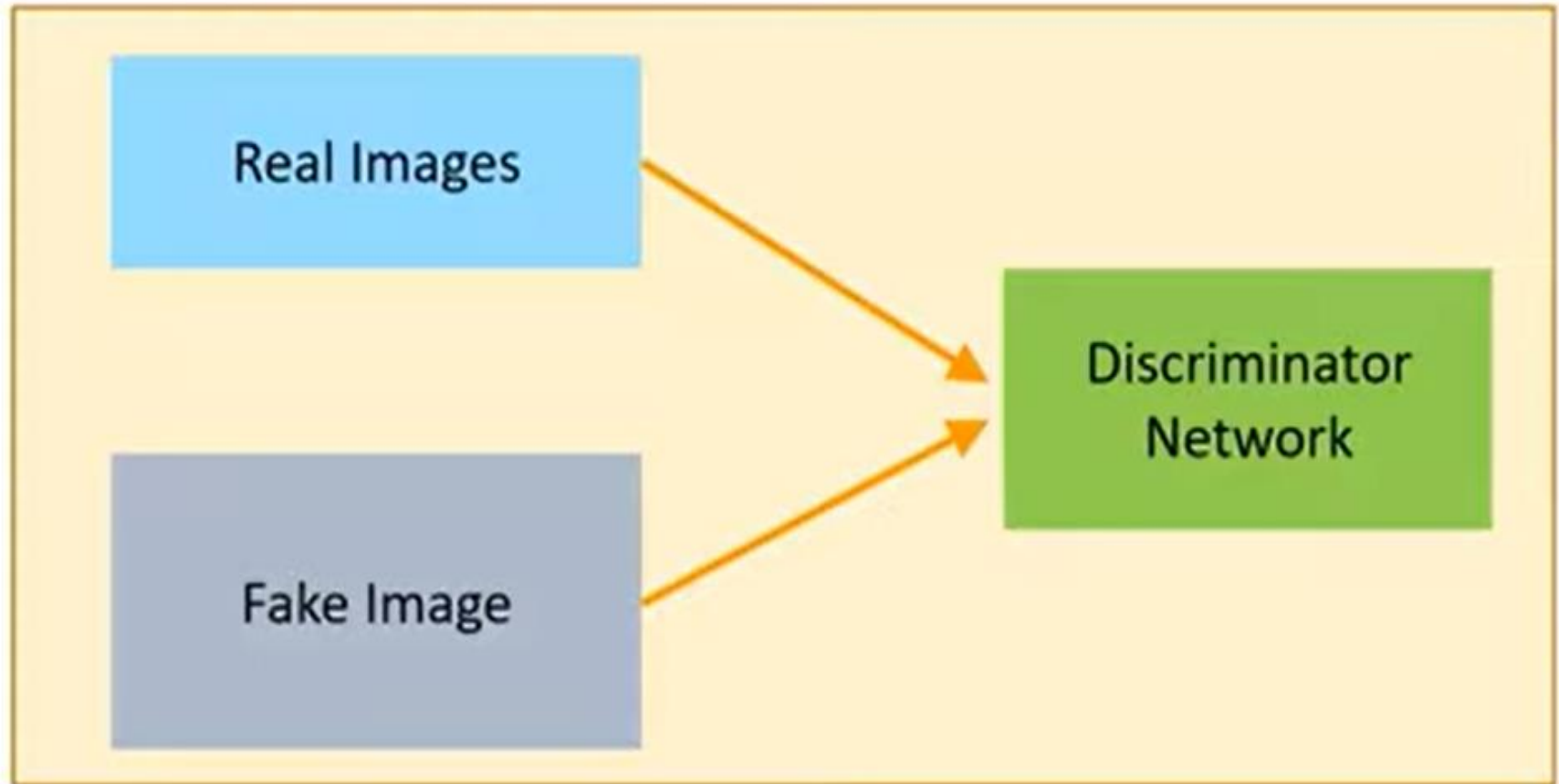


<https://www.simplilearn.com/tutorials/deep-learning-tutorial/generative-adversarial-networks-gans>

# Discriminator

---

The Discriminator in GAN is a classifier that identifies real data from the fake data created by the Generator.

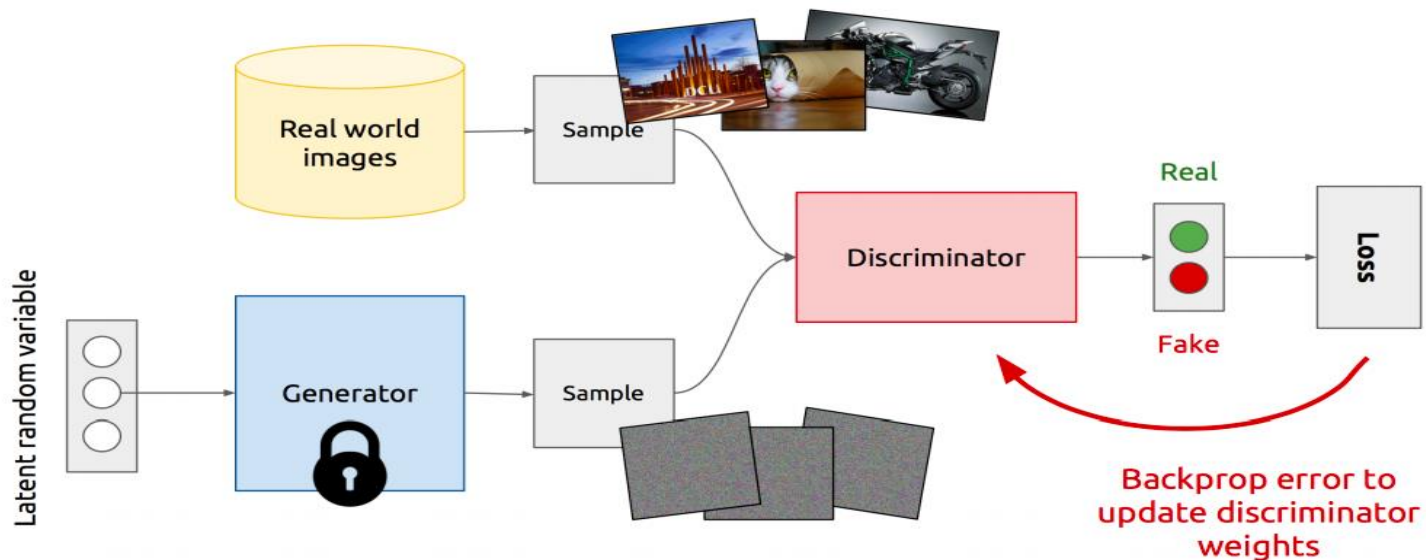


<https://www.simplilearn.com/tutorials/deep-learning-tutorial/generative-adversarial-networks-gans>

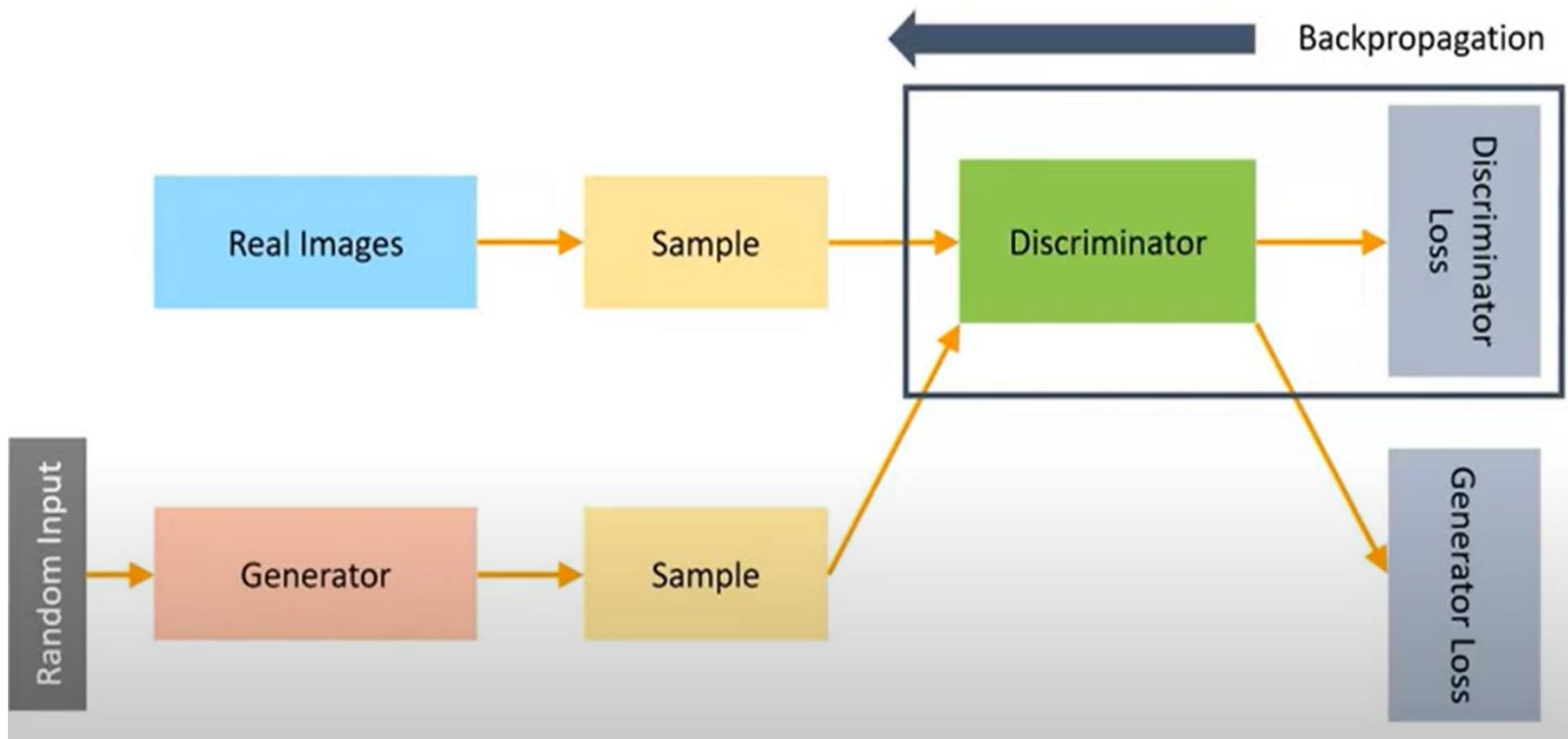
# Discriminator Training

- The **Discriminator is trained** while the **Generator is idle**. In this phase, the network is **only forward propagated, and no back-propagation is done**.
- The Discriminator is trained on both the real data and the fake generated data from Generator for n epochs and see if it can correctly predict them as real and fake respectively.

## Training Discriminator



# Discriminator Training



<https://www.simplilearn.com/tutorials/deep-learning-tutorial/generative-adversarial-networks-gans>

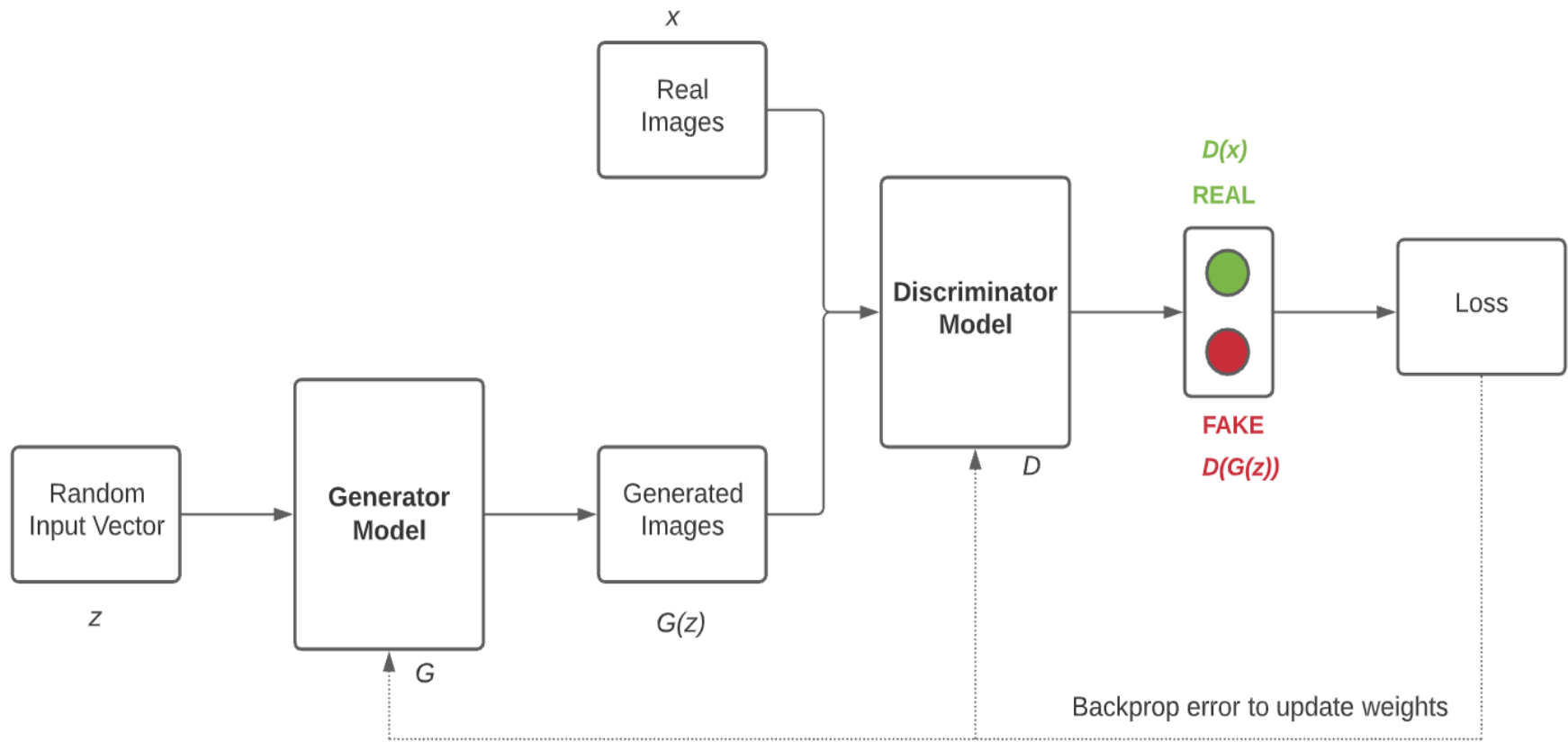
# How GANs Work?

## Steps for training GAN-

1. Define the Problem
2. Choose the architecture of GAN
3. Train Discriminator on Real Data
4. Generate Fake input for Generator
5. Train Discriminator on Fake data
6. Train Generator with the output of Discriminator.



# How GANs Work?



# How GANs Work?

---

- 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.

# How GANs Work?

---

Thus, the Generator Model and the Discriminator Model getting better and better at each epoch.

**We have to stop training when it attains the  $D(x) = 0.5$  for all  $x$ .** In simple words, when the generated images look almost like real images.



# Derivation of the Loss Function

---

The **loss function** described in the original paper by **Ian Goodfellow et al.** can be derived from the formula of binary cross-entropy loss. The **binary cross-entropy loss** can be written as

$$L(\hat{y}, y) = [y \cdot \log \hat{y} + (1 - y) \cdot \log(1 - \hat{y})]$$

Original data

Reconstructed data

# Discriminator Loss

---

$$L(\hat{y}, y) = [y \cdot \log \hat{y} + (1 - y) \cdot \log(1 - \hat{y})]$$

Original data

Reconstructed data

While training discriminator, the label of data coming from  $P_{data}(x)$  is  $y = 1$  (real data) and  $\hat{y} = D(x)$ . Substituting this in above loss function we get,

$$L(D(x), 1) = \log(D(x)) \quad (1)$$

and for data coming from generator, the label is  $y = 0$  (fake data) and  $\hat{y} = D(G(z))$ . So in this case,

$$L(D(G(z)), 0) = \log(1 - D(G(z))) \quad (2)$$

Now, **the objective of the discriminator is to correctly classify the fake and real dataset.** For this, **equations (1) and (2) should be maximized** and **final loss function for the discriminator** can be given as,

$$L^{(D)} = \max[\log(D(x)) + \log(1 - D(G(z)))] \quad (3)$$

# Generator Loss

---

Here, the generator is competing against discriminator. So, it will try to minimize the equation (3) and loss function is given as,

$$L^{(G)} = \min[\log(D(x)) + \log(1 - D(G(z)))] \quad (4)$$

# Combined Loss Function-MinMax Loss Function

---

We can combine equations (3) and (4) and write as,

$$L = \min_G \max_D [\log(D(x)) + \log(1 - D(G(z)))] \quad (5)$$

The above loss function is valid only for a single data point, to consider entire dataset we need to take the expectation of the above equation as

$$\min_G \max_D V(D, G) = \min_G \max_D (E_{x \sim P_{data}(x)} [\log D(x)] + E_{z \sim P_z(z)} [\log(1 - D(G(z)))] ) \quad (6)$$

which is the same equation as described in the original paper by Goodfellow et al.

# MinMax Loss Function

---

**The mathematical formula for working on GANs can be represented as:-**

It is designed to quantify the difference between the two probability distributions.

$$V(D, G) = E_{x \sim P_{data}(x)} [\log D(x)] + E_{z \sim p(z)} [\log(1 - D(G(z)))]$$

**Where,**

G = Generator

D = Discriminator

Pdata(x) = Distribution of real data

p(z) = Distribution of Generator

X = Sample from Pdata(x)

z = Sample from p(z)

D(x) = Discriminator Network

G(z) = Generator Network

# Loss Function

---

The loss function tells how good your model is in predictions.

If the **model predictions are closer to the actual values the Loss will be minimum** and **if the predictions are totally away from the original values the loss value will be the maximum.**

$$\text{Loss} = \text{abs}(Y_{\text{pred}} - Y_{\text{actual}})$$

# What is Binary Cross Entropy Or Logs Loss?

---

**Binary Cross Entropy** compares each of the predicted probabilities to actual class output which can be either 0 or 1.

It then calculates the score that penalizes the probabilities based on the distance from the expected value.

**That means how close or far from the actual value.**

# Predicted Probabilities

---

ID	Actual	Predicted probabilities
ID6	1	0.94
ID1	1	0.90
ID7	1	0.78
ID8	0	0.56
ID2	0	0.51
ID3	1	0.47
ID4	1	0.32
ID5	0	0.10

ID: It represents a unique instance.

Actual: It is the class the object originally belongs to.

Predicted probabilities.: The is output given by the model that tells, the probability object belongs to class 1.



## Corrected Probabilities

ID	Actual	Predicted probabilities	Corrected Probabilities
ID6	1	0.94	0.94
ID1	1	0.90	0.90
ID7	1	0.78	0.78
ID8	0	<b>0.56</b>	<b>0.44</b>
ID2	0	<b>0.51</b>	<b>0.49</b>
ID3	1	0.47	0.47
ID4	1	0.32	0.32
ID5	0	<b>0.10</b>	<b>0.90</b>

As shown in the above image, ID6 originally belongs to class 1 hence its predicted probability and corrected probability is the same i.e 0.94.

On the other hand, the observation ID8 is from class 0. In this case, the predicted probability i.e. the chances that ID8 belongs to class 1 is 0.56 whereas, the corrected probability means the chances that **ID8 belongs to class 0 is (1- predicted\_probability) is 0.44.**

## Log(Corrected Probabilities)

ID	Actual	Predicted probabilities	Corrected Probabilities	Log
ID6	1	0.94	0.94	-0.0268721464
ID1	1	0.90	0.90	-0.0457574906
ID7	1	0.78	0.78	-0.1079053973
ID8	0	0.56	0.44	-0.3565473235
ID2	0	0.51	0.49	-0.30980392
ID3	1	0.47	0.47	-0.3279021421
ID4	1	0.32	0.32	-0.4948500217
ID5	0	0.10	0.90	-0.0457574906

We will calculate the log value for each of the corrected probabilities.

The reason behind using the log value is, **the log value offers less penalty for small differences between predicted probability and corrected probability. When the difference is large the penalty will be higher.**

# **Types of Generative Adversarial Networks (GANs)**

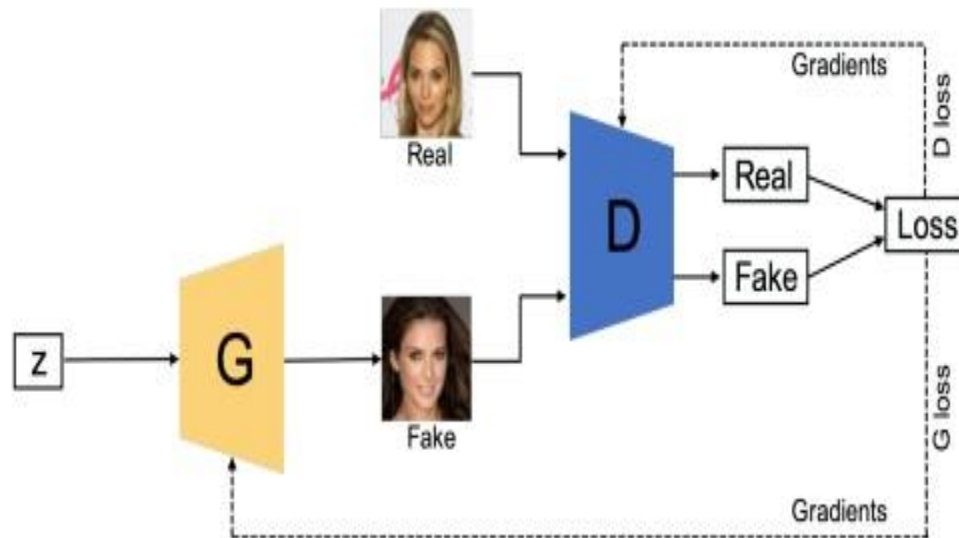
---

**The Different Types of Generative Adversarial Networks (GANs) are:**

- Vanilla GAN
- Conditional Gan (CGAN)
- Deep Convolutional GAN (DCGAN)
- CycleGAN
- Generative Adversarial Text to Image Synthesis
- Style GAN
- Super Resolution GAN (SRGAN)
- 3D GAN
- SpecGAN
- WaveGAN
- Stack GAN etc.

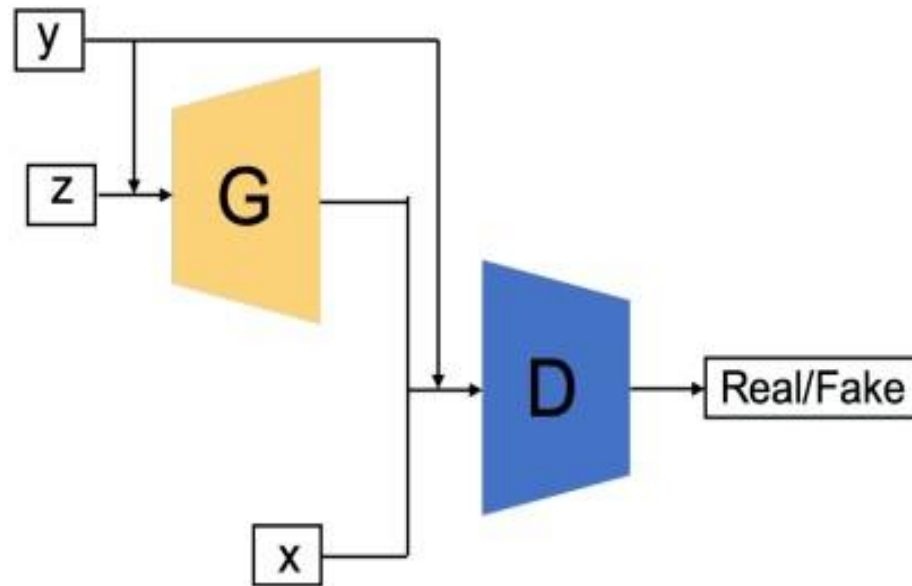
# Vanilla GAN

- The **Vanilla GAN is the simplest type of GAN** made up of the generator and discriminator, where the classification and generation of images is done by the generator and discriminator internally **with the use of multi layer perceptrons**.
- The generator captures the data distribution meanwhile, the discriminator tries to find the probability of the input belonging to a certain class.
- Finally, the feedback is sent to both the generator and discriminator after calculating the loss function , and hence the effort to minimize the loss comes into picture.



# Conditional Gan (cGAN)

- In this GAN, the generator and discriminator both are **provided with additional information that could be a class label or any modal data.**
- As the name suggests the **additional information helps the discriminator in finding the conditional probability instead of the joint probability.**



<https://iq.opengenus.org/types-of-gans/>

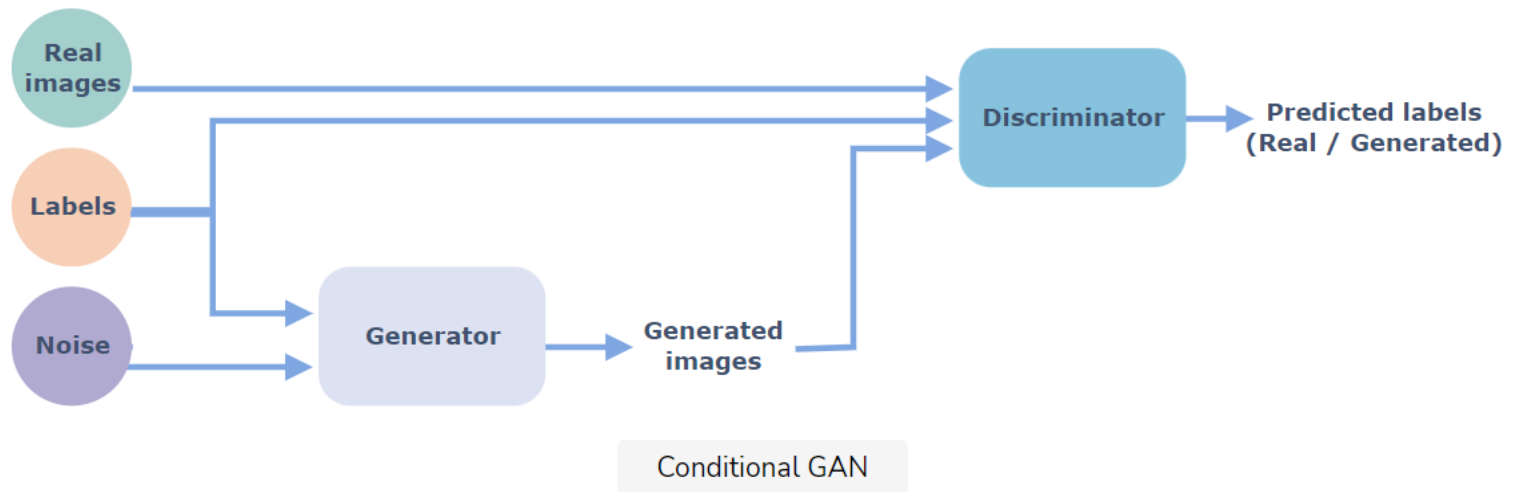
The loss function of the conditional GAN is as below

$$\min_G \max_D \mathbb{E}_{\mathbf{x} \sim p_r} \log[D(\mathbf{x}|\mathbf{y})] + \mathbb{E}_{\mathbf{z} \sim p_z} \log[1 - D(G(\mathbf{z}|\mathbf{y}))].$$

# Conditional Gan (cGAN)

**Suppose you train a GAN on hand-written digits (MNIST images).**

- We normally cannot control what specific images the generator will produce.
- In other words, there is no way you can request a particular digit image from the generator.
- This is where the cGANs come in as we can add **an additional input layer of one-hot-encoded image labels**.
- **This additional layer guides the generator in terms of which image to produce.**
- The input to the additional layer can be a feature vector derived either an image that encodes the class or a set of specific characteristics we expect from the image.



# Conditional Gan (cGAN)-Advantages

---

❑ **By providing additional information, we get two benefits**

- 1. Convergence will be faster.** Even the random distribution that the fake images follow will have some pattern.
- 2. You can control the output of the generator at test time by giving the label for the image you want to generate.**

❑ **Applications**

- Image-to-image translation
- Text-to-image synthesis
- Video generation
- Convolutional face generation etc.

# Deep Convolutional GAN(DCGAN)

---

- **Deep Convolutional GAN (DCGAN)** was proposed by a researcher from **MIT and Facebook AI research**.
- It is widely used in many convolution-based generation techniques.
- The focus of this paper was to **make training GANs stable**.
- Hence, they **proposed some architectural changes**.
- It mainly **composes of convolution layers without max pooling & fully connected layers**.
- It uses **Convolutional Stride and Transposed Convolution for the downsampling and the upsampling**.



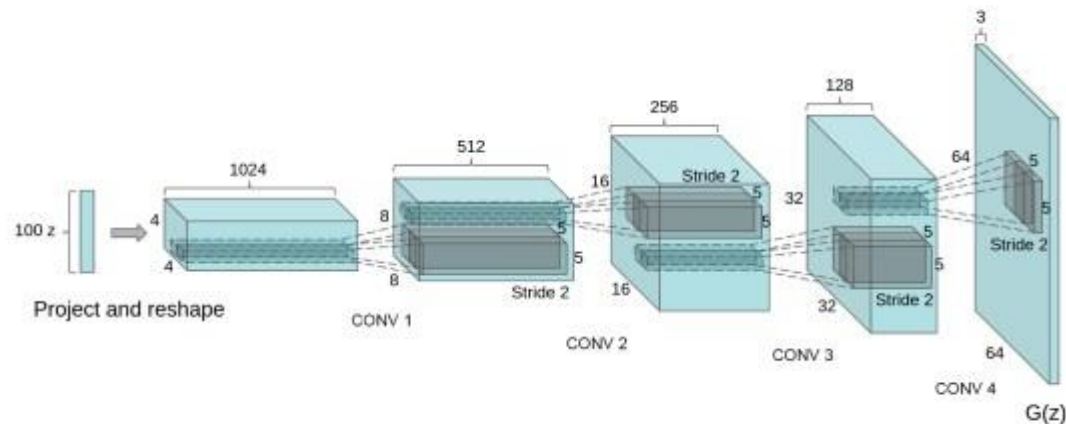
# Deep Convolutional GAN(DCGAN)

---

- DCGAN uses convolutional and convolutional-transpose layers in the generator and discriminator, respectively.
- It was proposed by Radford et. al. in the paper [Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks](#).
- The discriminator consists of strided convolution layers, batch normalization layers, and LeakyRelu as activation function.
- It takes a 3x64x64 input image.
- The generator consists of convolutional-transpose layers, batch normalization layers, and ReLU activations.
- The output will be a 3x64x64 RGB image.
- The transposed Convolutional Layer is also known as the Deconvolutional layer.
- Transposed convolution doesn't reverse the standard convolution by values, rather by dimensions only.

# Deep Convolutional GAN (DCGAN)

- This is the **First GAN where the generator used deep convolutional network** , hence generating high resolution and quality images to be differentiated.
- **ReLU activation** is used in Generator all layers except last one where **Tanh** activation is used, meanwhile **in Discriminator** all layers uses the **Leaky-ReLu activation function**.
- **Adam optimizer** is used to update network weights iterative based in training data with a Learning Rate of 0.0002.



<https://iq.opengenus.org/types-of-gans/>

The above figure shows the architecture of generator of the GAN. The input generated is of 64 X 64 resolution.

# Need for DCGANs:

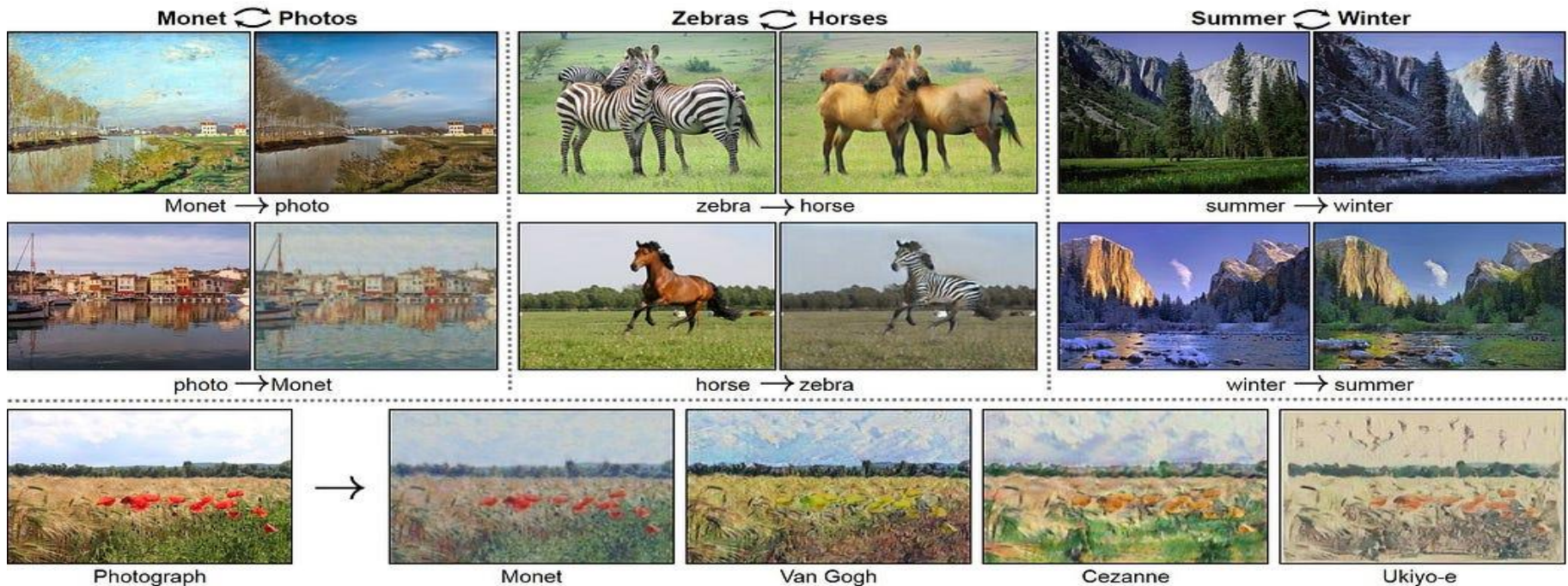
---

- **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- take the case of 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.

# CycleGAN

- This GAN is made for Image-to-Image Translations, meaning one image to be mapped with another image.

A mapping function  
Is used to convert  
Image into image



# CycleGAN-Architecture

---

**CycleGAN** is a Generative Adversarial Network (GAN) that **uses two generators and two discriminators.**

We call one generator  $G$ , and it converts images from the  $X$  domain to the  $Y$  domain. The other generator is called  $F$ , and converts images from  $Y$  to  $X$ .

$$G : X \longrightarrow Y$$

$$F : Y \longrightarrow X$$

Both  $G$  and  $F$  are generators that take an image from one domain and translate it to another.  $G$  maps from  $X$  to  $Y$ , whereas  $F$  goes in the opposite direction, mapping  $Y$  to  $X$ .

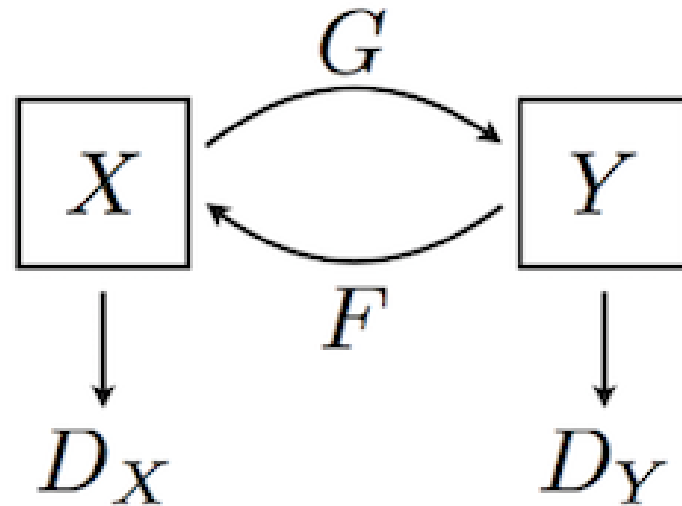
Each generator has a corresponding discriminator, which attempts to tell apart its synthesized images from real ones.

**Dy: Distinguishes  $y$  from  $G(x)$**

**Dx: Distinguishes  $x$  from  $F(y)$**

**One discriminator provides adversarial training for  $G$ , and the other does the same for  $F$ .**

# CycleGAN-Architecture



## It consists of :

- Two mappings  $\mathbf{G} : X \rightarrow Y$  and  $\mathbf{F} : Y \rightarrow X$
- Corresponding Adversarial discriminators  $\mathbf{D}_x$  and  $\mathbf{D}_y$

**Role of G:**  $G$  is trying to **translate**  $X$  into outputs, which are fed through  $D_y$  to check whether they are real or fake according to Domain  $Y$

**Role of F :**  $F$  is trying to **translate**  $Y$  into outputs, which are fed through  $D_x$  to check if they are indistinguishable from Domain  $X$



# Generative Adversarial Text to Image Synthesis

- In this the GANs are capable of finding an image from the dataset that is closest to the text description and generate similar images.

this small bird has a pink breast and crown, and black primaries and secondaries.



this magnificent fellow is almost all black with a red crest, and white cheek patch.



the flower has petals that are bright pinkish purple with white stigma



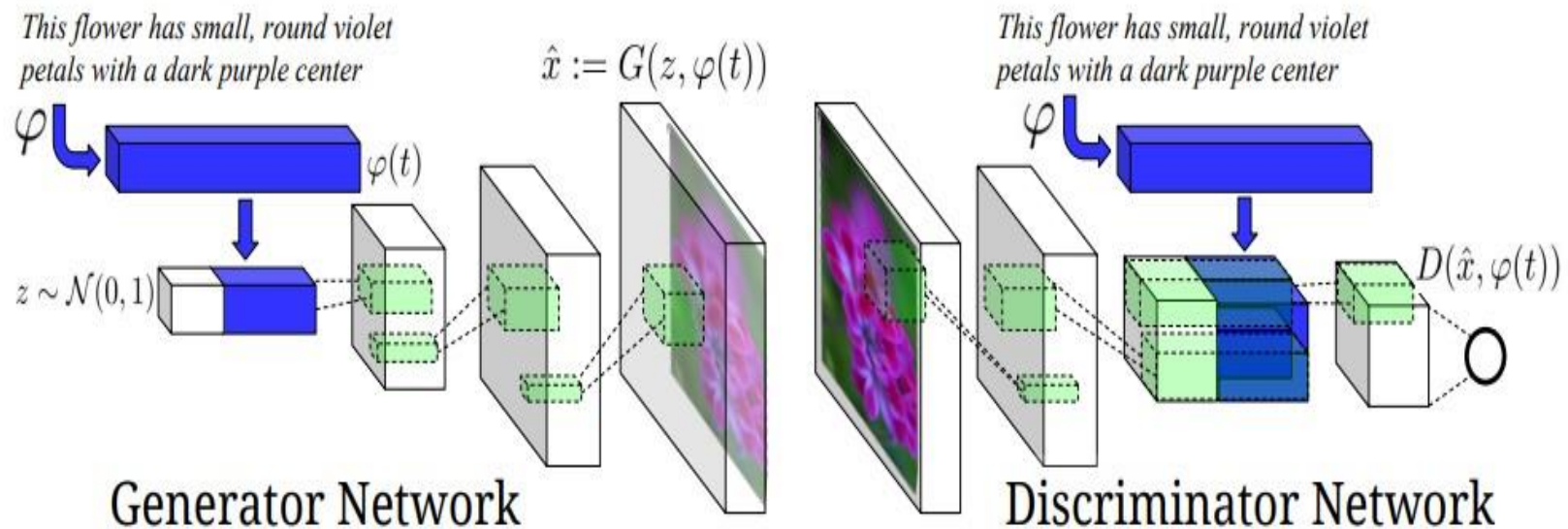
this white and yellow flower have thin white petals and a round yellow stamen



<https://iq.opengenus.org/types-of-gans/>

# The GAN architecture in this case is given below:

- As, the generator network generates based on the description and the differentiation is done by the discriminator based on the features mentioned in text description.



<https://iq.opengenus.org/types-of-gans/>

**A deep convolutional generative adversarial network (DC-GAN) conditioned on text features.**



# Style GAN

- Other GANs focused on improving the discriminator in this case we improve the generator. **This GAN generates by taking a reference picture.**



# Style GAN

---

## ❑ There are three main Components of StyleGAN

1. Progressive Growing,
2. Noise Mapping Network,
3. Adaptive Instance Normalization.

# Style GAN

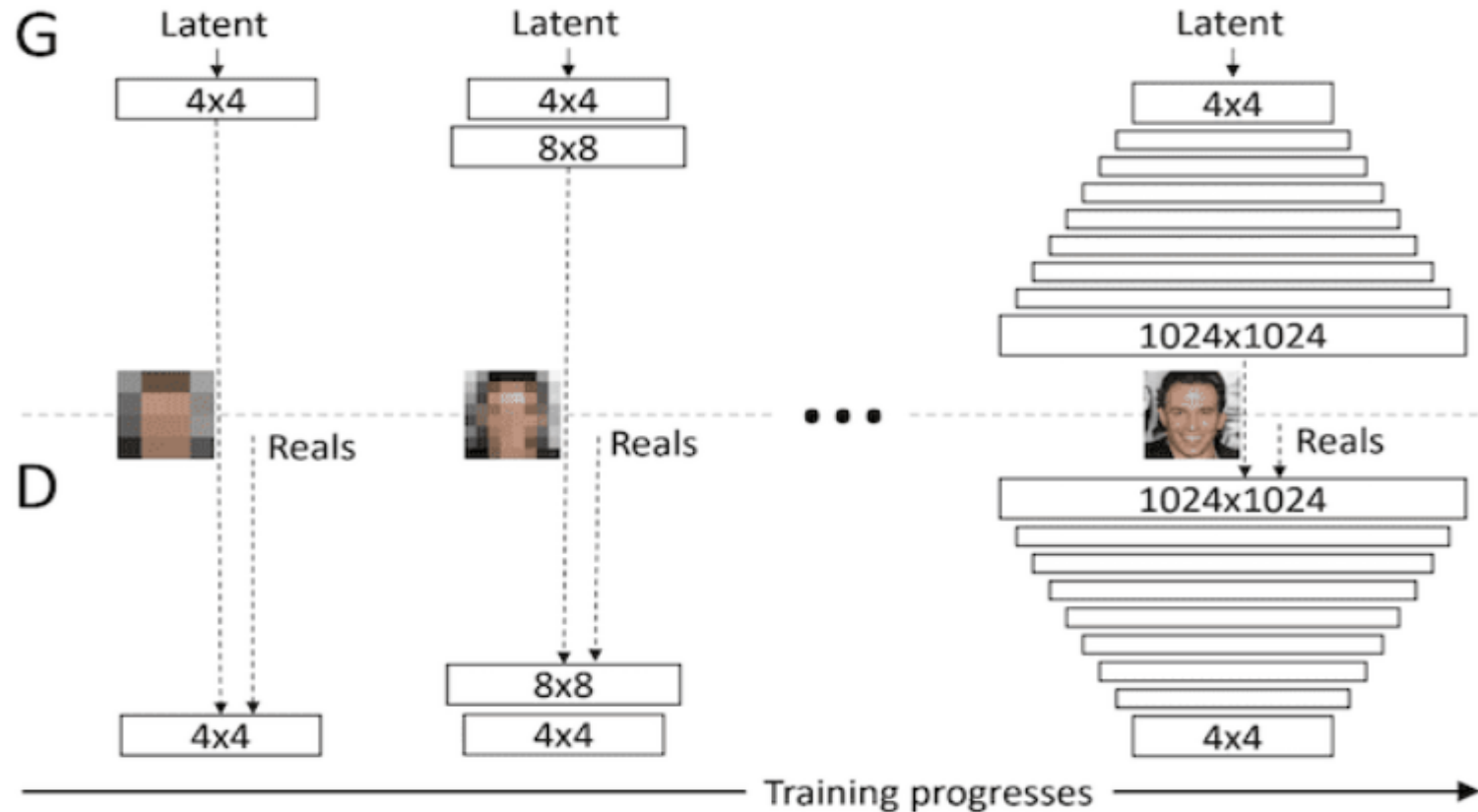
---

## 1. Progressive Growing

- This means that both models start with small images, in this case,  $4 \times 4$  images. The models are fit until stable, then both discriminator and generator are expanded to double the width and height (quadruple the area), e.g.  $8 \times 8$ .
- A new block is added to each model to support the larger image size, which is faded in slowly over training.
- Once faded-in, the models are again trained until the desired target image size is met, such as  $1024 \times 1024$ .
- The progressive growing GAN **uses nearest neighbor layers for upsampling instead of transpose convolutional layers** that are common in other generator models.

# Style GAN

## 1. Progressive Growing



Example of Progressively Adding Layers to Generator and Discriminator Models.  
Taken from: Progressive Growing of GANs for Improved Quality, Stability, and Variation.

# Style GAN

---

## 2. Mapping Network and AdaIN

- A standalone mapping network is used that takes a randomly sampled point from the latent space as input and generates a style vector. (Latent space is a simpler, hidden representation of a data point.)
- The mapping network is comprised of eight fully connected layers, e.g. it is a standard deep neural network.
- The style vector is then transformed and incorporated into each block of the generator model after the convolutional layers via an operation called adaptive instance normalization or AdaIN.
- The **AdaIN layers** involve first standardizing the output of feature map to a standard Gaussian, then adding the style vector as a bias term.

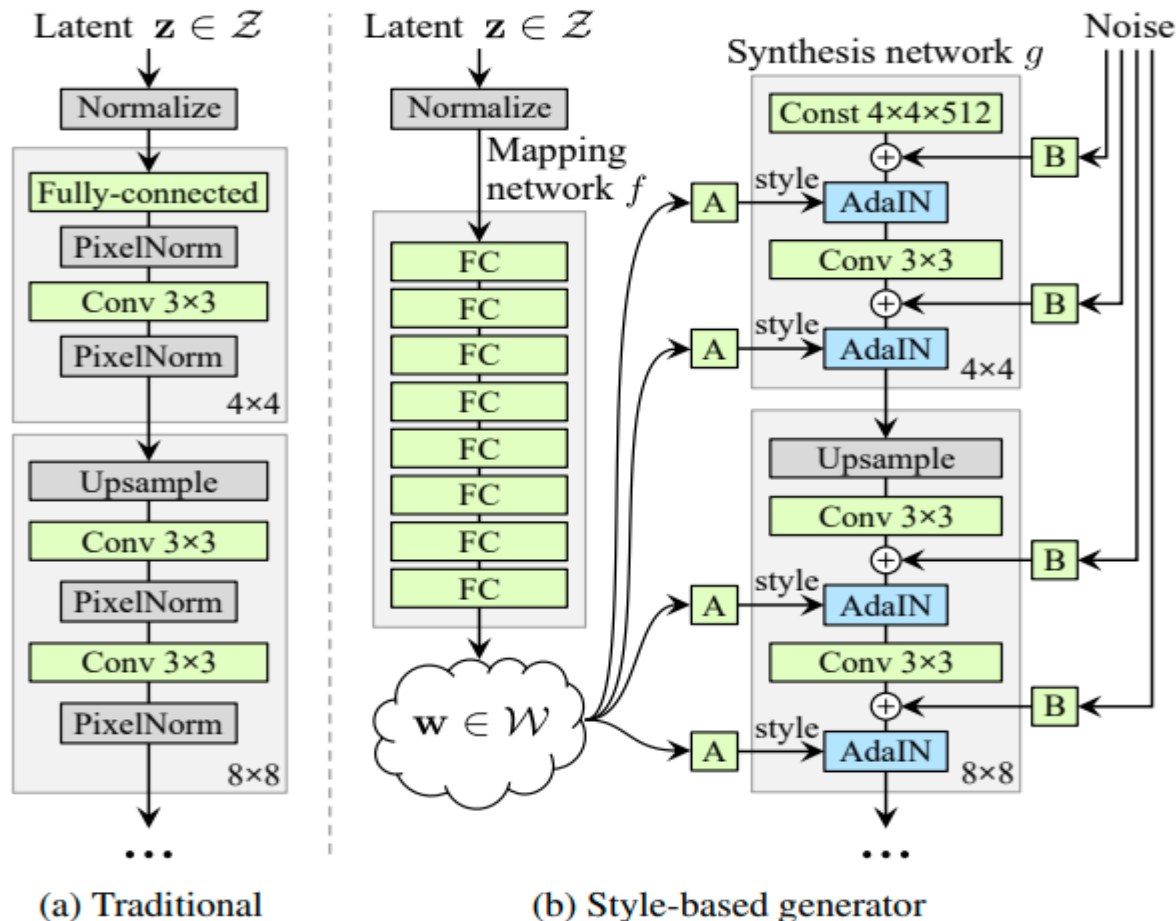
# Style GAN

---

- StyleGAN produces the **simulated image sequentially**, originating from a simple resolution and **enlarging to a huge resolution (1024×1024).**  
**(Progressive Growing GANs)**
- By transforming the input of each level individually, it examines the **visual features that are manifested in that level, from standard features (pose, face shape) to minute details (hair color)**, without altering other levels.
- The resulting model is proficient in producing **impressively photorealistic high-quality photos of faces.**
- It **grants control over the characteristic of the created image at different specification levels by changing the style vectors and noise.**
- **The noise is** affixed to whole feature maps that **enable the model to understand the style in a fine-grained, per-pixel manner.**

# Style GAN

- As you can see the figure below, Style GAN architecture consists of a Mapping network that maps the input to an intermediate Latent space.
- Further the intermediate is processed using the AdaIN after each layer, there are approximately 18 convolutional layers.



# Style GAN

---

- The Style GAN uses the AdaIN or the Adaptive Instance Normalization which is defined as

$$\text{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i},$$

Here,  $\mathbf{x}_i$  is the feature map, which is tweaked by using the component  $\mathbf{y}$ .

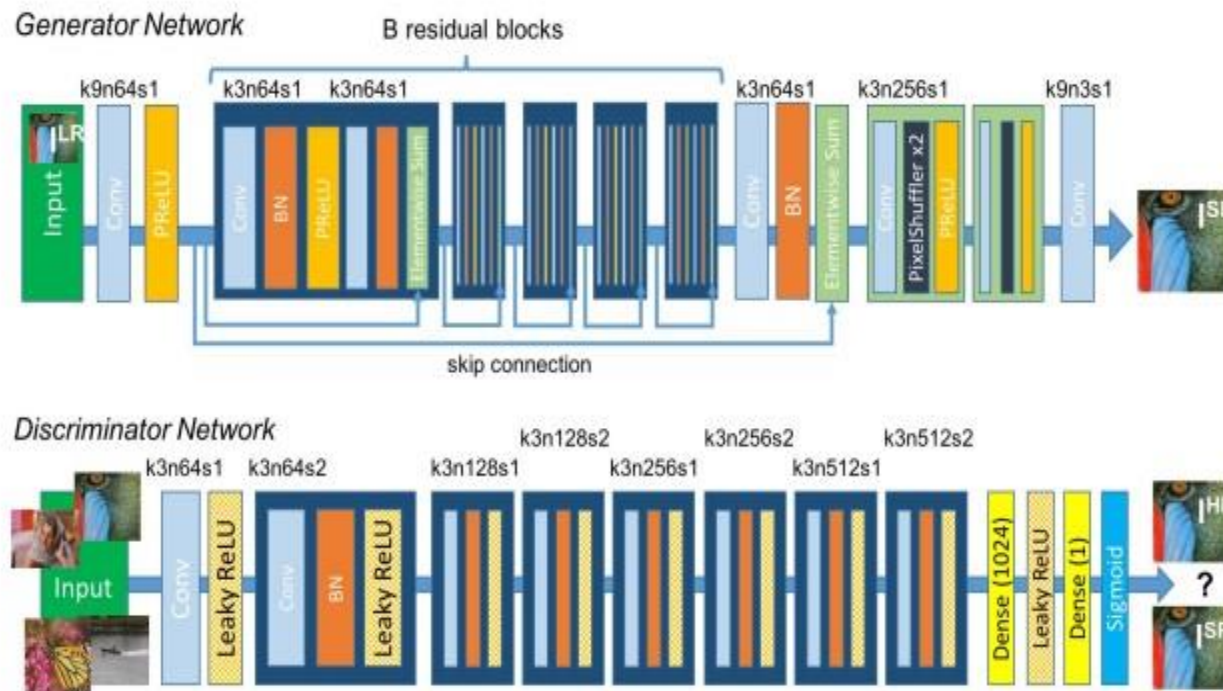
- **A separate convolutional layer's input is normalized with adaptive instance normalization (AdaIN) operation** using the latent vector “**style**” **embeddings**.
- Lastly, added noise is included in the network to generate more stochastic details in the images.
- This noise is just a single-channel picture consisting of uncorrelated Gaussian noise.
- Before each AdaIN operation, noise is supplied at a specific convolutional layer.
- Additionally, there is a scaling part for the noise, which is decided per feature.

## [Example](#)



# Super Resolution GAN (SRGAN)

- The main purpose of this type of GAN is to make a low-resolution picture into a more detailed picture.
- This is one of the most researched problems in Computer vision. The architecture of the SRGAN is given below



# Super Resolution GAN (SRGAN)

---

- As given in the above figure we observe that the Generator network and Discriminator both make use of Convolutional layers, the Generator make use of the Parametric ReLU as the activation function whereas the Discriminator uses the Leaky ReLU.
- The Generator takes the input data and finds the average of all possible solutions that is pixelwise, when formulated looks like,

$$\hat{\theta}_G = \arg \min_{\theta_G} \frac{1}{N} \sum_{n=1}^N l^{SR}(G_{\theta_G}(I_n^{LR}), I_n^{HR})$$

here,  $I^{HR}$  represents high resolution image , whereas  $I^{LR}$  represents lower resolution image and  $l^{SR}$  represents loss and  $\theta_g$  represents the weight and biases at layer L. Now the perceptual loss is given as

# Super Resolution GAN (SRGAN)

---

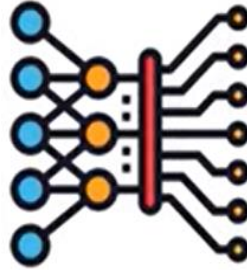
$$l^{SR} = \underbrace{l_X^{SR}}_{\text{content loss}} + \underbrace{10^{-3}l_{Gen}^{SR}}_{\text{adversarial loss}}$$

perceptual loss (for VGG based content losses)

Finally, the content loss is summed with adversarial loss and Mean Squared Error is taken to find the best pixel wise solution.

# Vanilla GANs & DCGANs

---



## Vanilla GANs

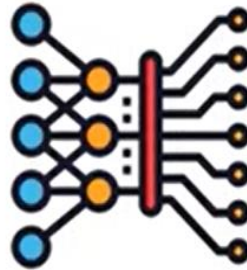
**Simplest type of GAN where the Generator and Discriminator are simple multi-layer perceptrons**

## Deep Convolutional GANs (DCGANs)

**DCGANs comprise of ConvNets and are more stable and generate higher quality images**

# CGANs & SRGANs

---



Conditional GANs  
(CGANs)

**CGANs use extra label information to generate better results**

Super Resolution GAN  
(SRGAN)

**SRGANs generate a photorealistic high-resolution image when given a low-resolution image**

# Applications of GANs

---

1. Firstly, **GANs can be used as data augmentation techniques** where the generator generates new images taking the training dataset and producing multiple images by applying some changes.
2. **GANs improve the classification techniques** by training the discriminator on a very large scale of data that are real as well as fake.
3. **GANs are also used to improve the resolution of any input image.**
4. Just **like filters used in Snapchat**, a filter could be applied to see what a place might look like if **in summer, winter, spring or autumn and many more conditions** could be applied.
5. **GANs are used to convert semantics into images** and better understand the visualizations done by the machine etc.

# Applications of GANs

---

## 1. Generating Cartoon Characters



**Using DCGANs, you can create faces of anime and Pokemon characters**

# Applications of GANs

---

## 2. Generating Human Faces



**GANs can be trained on the images of humans to generate realistic faces**



# Applications of GANs

## 3. Text to Image Translation

Bird with a black head, yellow body and a short beak



GANs can build realistic images from textual descriptions of simple objects like birds

A Flower With **Red** Petals And **Green** Leaves



# Applications of GANs

---

## 4. 3-D object Generation-3-D GAN



**GANs can generate 3-D models  
using 2-D pictures of objects  
from multiple perspectives**

# Applications of GANs

## 5. Prediction of Next Frame in A Video



# Applications of GANs

---

## 6. Image to Image Translation-Cycle GAN



**Real**



**Generated**



**Reconstructed**

# Applications of GANs

---

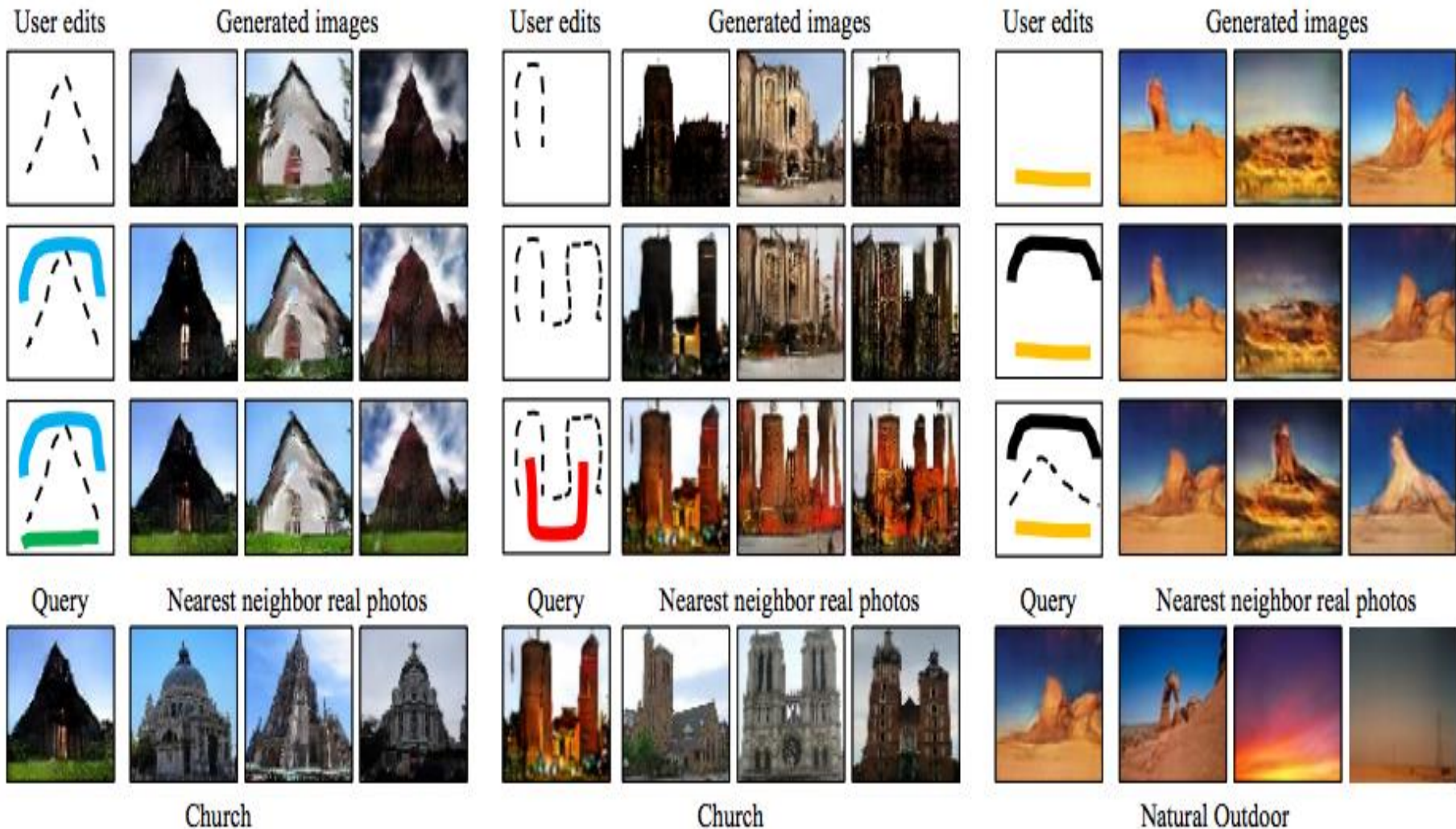
## 7. Enhancing the Resolution of an Image using Super Resolution GANs





# Applications of GANs

## 8. Interactive Image Generation



# Challenges Faced by GANs

---

Problem of stability between generator and discriminator

Problem to determine positioning of the objects

Problem in understanding the perspective

Problem in understanding global objects

**Deep Convolutional GAN and recent GANs are more advanced. We can overcome problems using it**

