UNIT-II

PART – I

BUILDING BLOCKS OF DEEP LEARNING

RESTRICTED BOLZMANN MACHINE

Topics :

- Restricted Boltzmann Machine
  - Types of RBM
  - Working Principle of RBM
  - Reconstruction of RBM
  - Training of RBM
  - Use cases of RBM
  - Application of RBM

Restricted Boltzmann Machine

- **Restricted Boltzmann Machine** is an **undirected graphical model** that plays a major role in **Deep Learning Framework** in recent times.
- It was initially introduced as ***Harmonium*** by ***Paul Smolensky*** *in* ***1986*** and it gained big popularity in recent years in the context of the ***Netflix Prize*** ***by using Collaborative Filtering.***

- Restricted Boltzmann Machine (RBM for versatile **feature extraction method.**

CONTD..

- It **is an algorithm** which is useful for

  i) **Dimensionality Reduction**

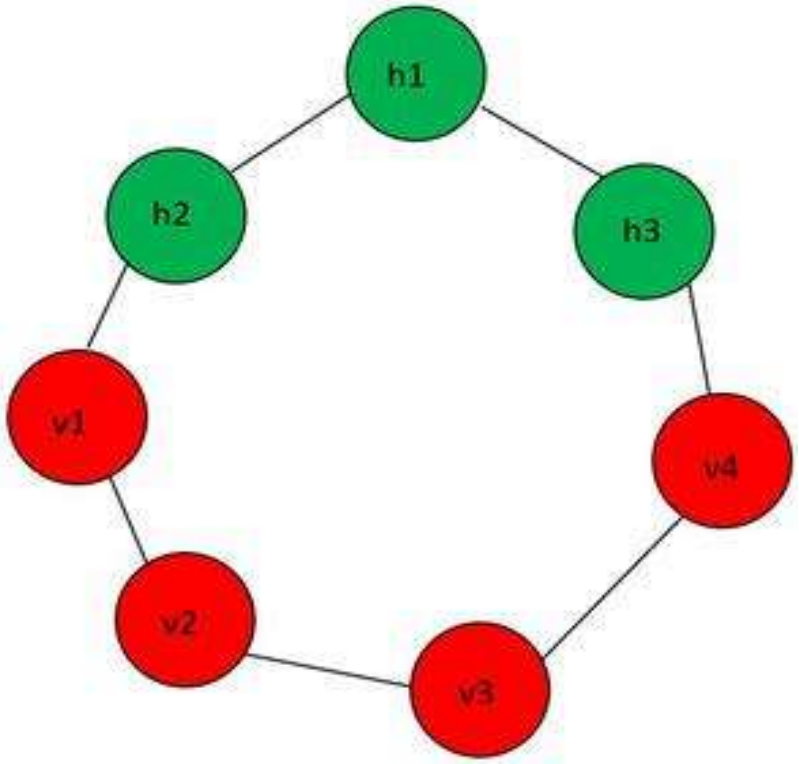
  ii) **Classification**

  iii)**Regression**

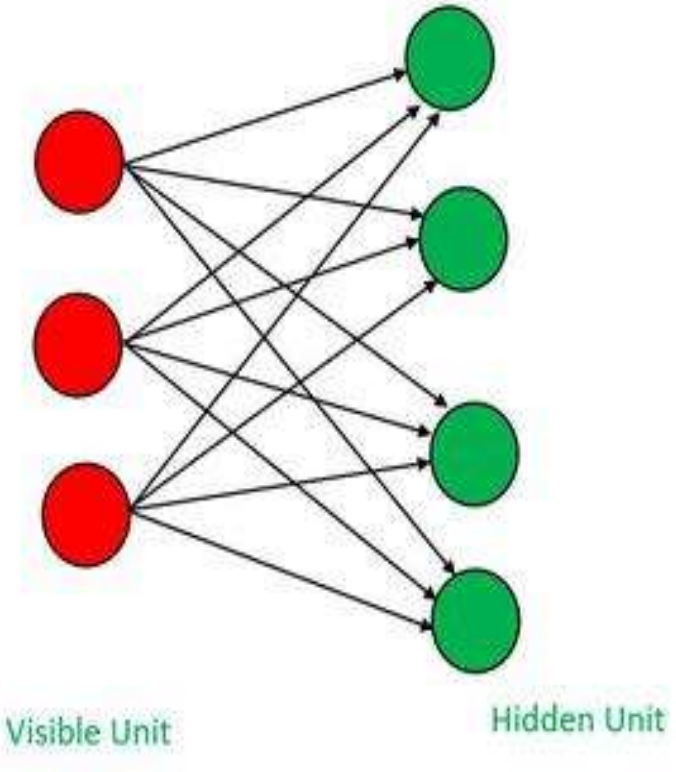
  iv) **Collaborative filtering, etc**.

- The Name suggests, The **"restricted"** part of the name "**Restricted Boltzmann Machines**" means
- **Restrict: No Visible unit is** connected to **any other Visible Unit**
- **No Hidden Unit is connected to any other Hidden Unit.**

Contd..



Boltzmann Machine



Restricted Boltzmann Machine

CONTD..

- The main difference between a **Boltzmann machine** and a **restricted Boltzmann machine** is that there is **no intra layer communication**, i.e, the nodes of the **same layer are not connected** which makes them **independent from each other**.

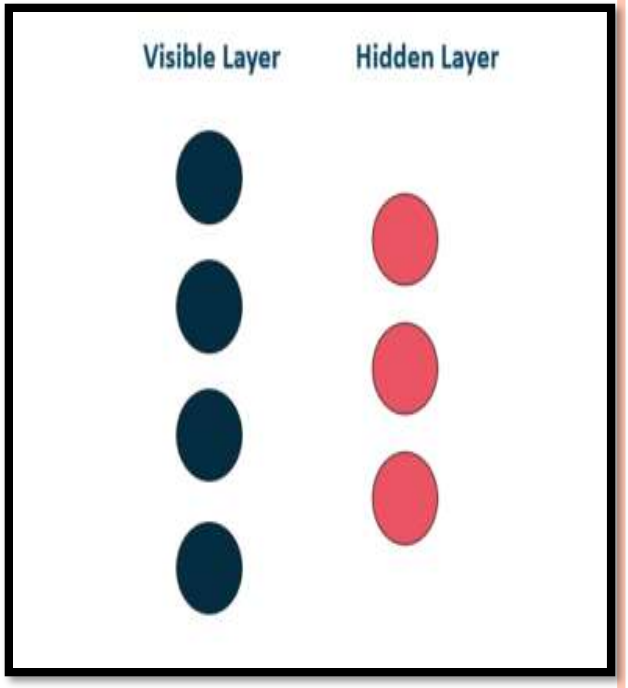
Types of RBM

- There are **mainly two types** of **Restricted Boltzmann Machine (RBM)** based on the **types of variables** they use:
- **Binary RBM:** In a **binary RBM**, the **input and hidden units are binary variables.** Binary RBMs are often used in modeling **binary data such as images or text.**
- **Gaussian RBM:** In a **Gaussian RBM**, the **input and hidden units are continuous variables** that follow a Gaussian distribution. Gaussian RBMs are often used in **modeling continuous data** such as **audio signals or sensor data.**

Working Principle of RBM

- **Layers:** Restricted Boltzmann Machines are **shallow**, **two-layer neural nets** that constitute the building blocks of ***deep-belief networks.***
- The **first Layer** of the RBM is called the **visible, or input layer,** and
- The **Second Layer** is the **Hidden Layer.**
- **Each circle** represents a **Neuron-like unit** called a **node.**

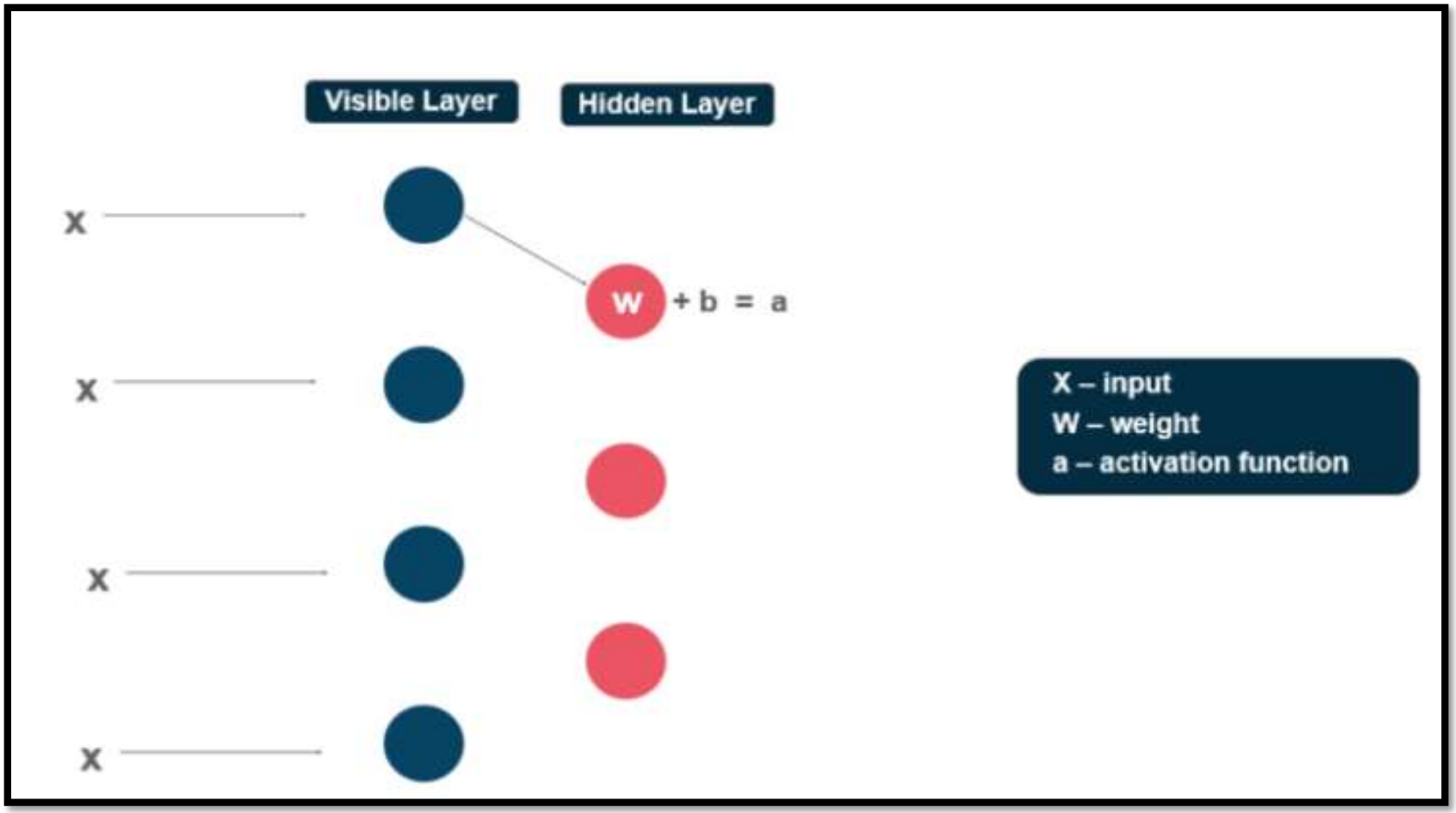
  The nodes are **connected to each other across layers,** but **no two nodes** of the **same layer are linked**.



CONTD..

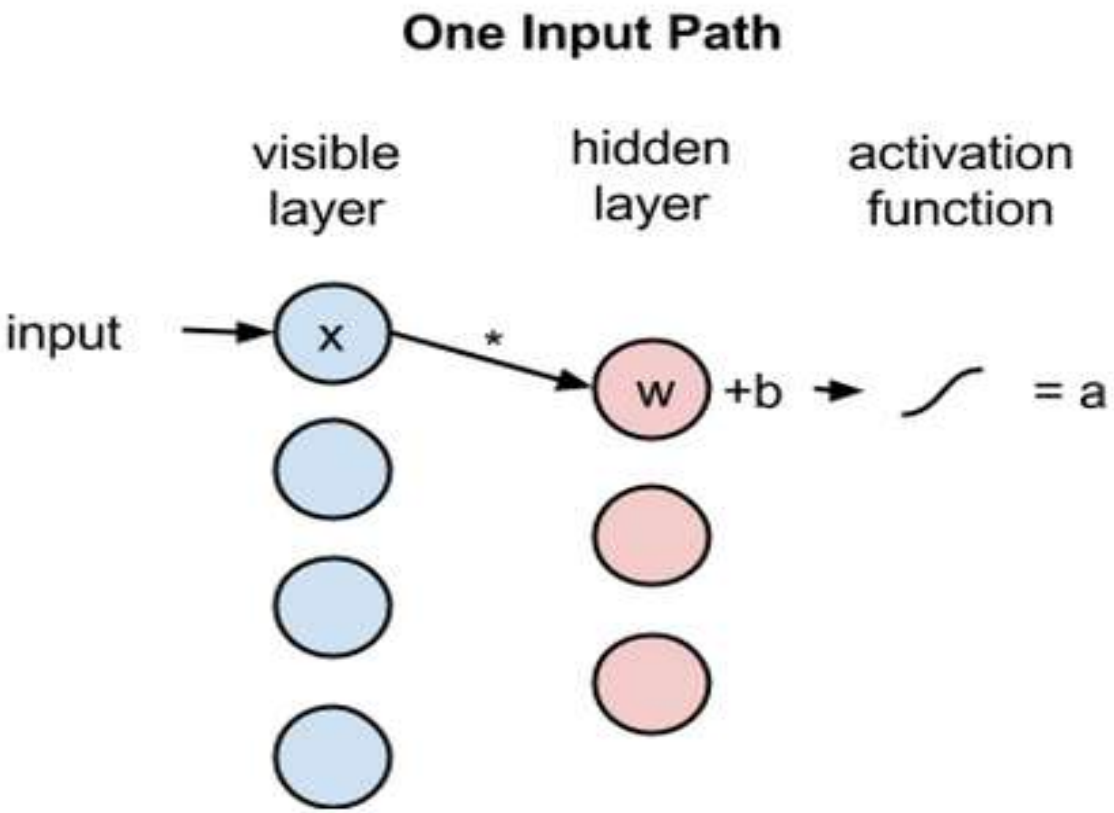
- Each visible node takes a **low-level feature** from **an item in the dataset** to be learned.
- **At node 1 of the hidden layer**, **x is multiplied** by a ***weight*** and added to a ***bias***.
- The result of those **two operations** is fed into an ***activation function***, which produces the **node's output,** or **the strength of the signal** passing through it, **given input x.**

CONTD..


Visible Layer
Hidden Layer
x
x
x
x
W
+ b = a
X – input
W – weight
a – activation function

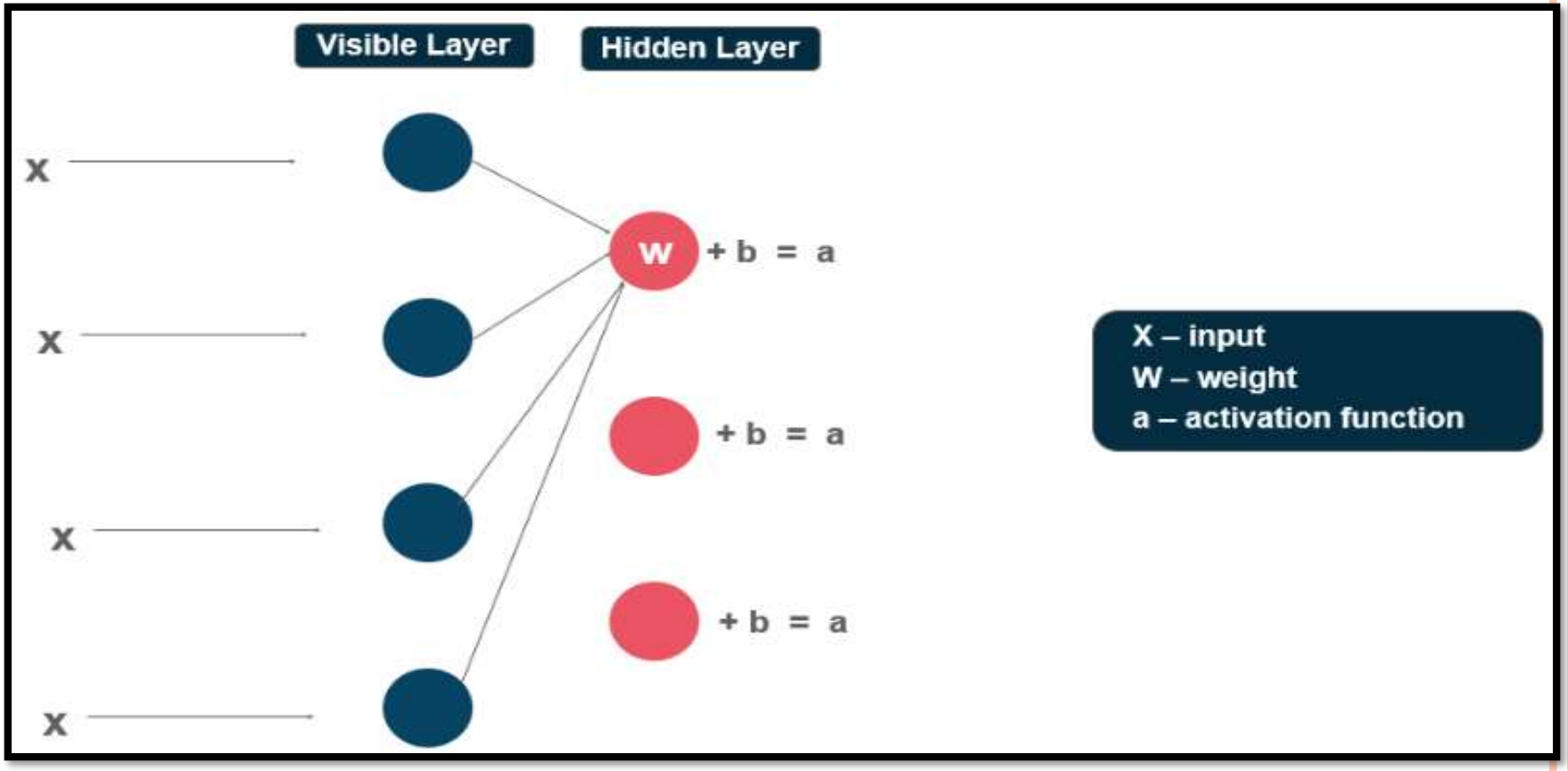
CONTD..

```
activation f((weight w * input x) + bias b ) = output a
```



CONTD..

- Next, let's look at **how several inputs** would **combine at one hidden node.**
- Each **x is multiplied by a separate weight,** the **products are summed, added to a bias,** and again the result is passed through **an activation function** to **produce the node's output.**



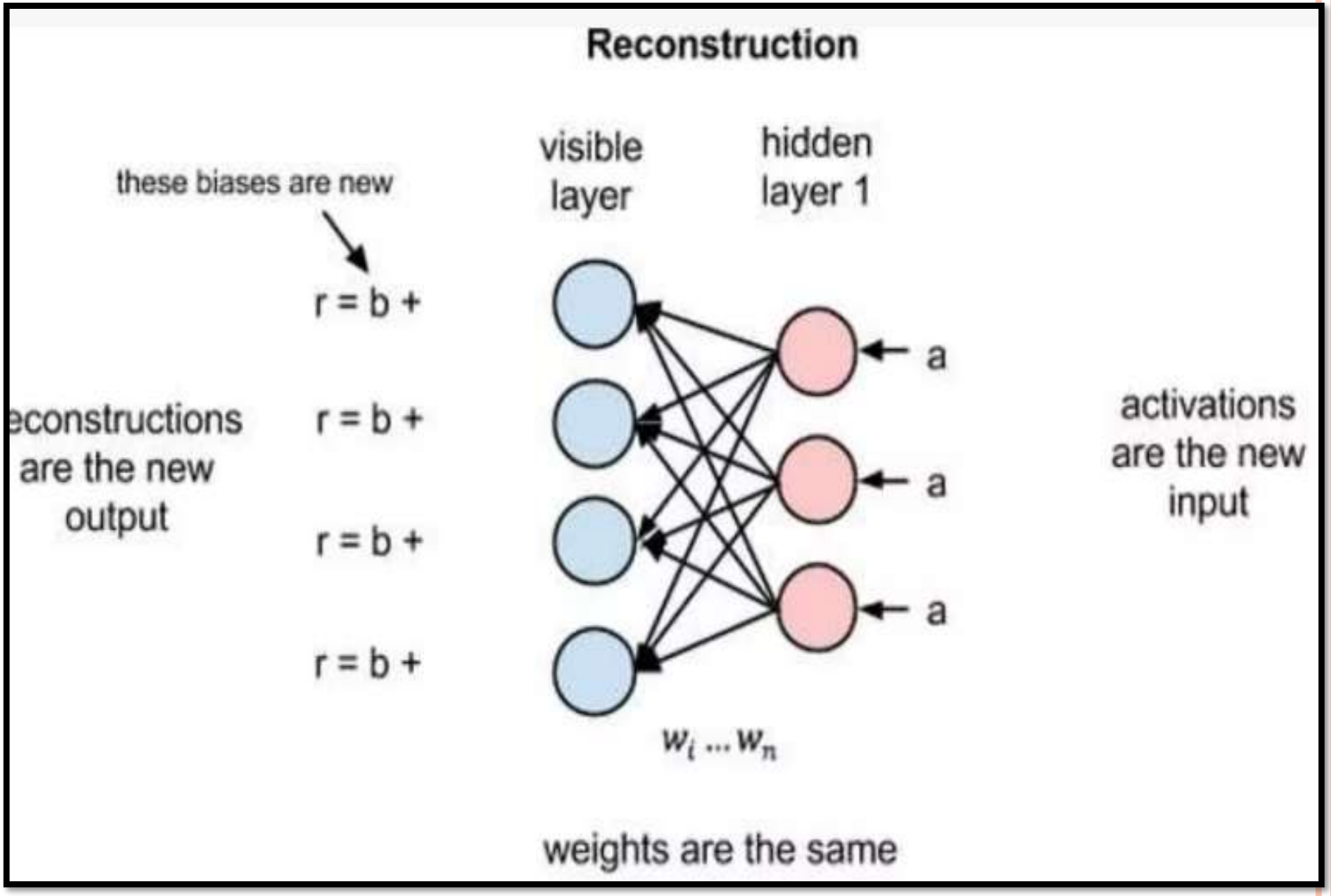
CONTD..

- At **each hidden node**, each **input x** is **multiplied by its respective weight w**. i.e, **a single input x** would have **three weights here**, making **12 weights altogether (4 input nodes x 3 hidden nodes).**
- The **weights between the two layers** will always ***form a matrix*** where the **rows are equal to the input nodes**, and the **columns are equal to the output nodes.**
- **Each hidden node receives the four inputs multiplied by their respective weights.**
- The **sum of those products** is **again added to a bias** (which forces at least some activations to happen), and the **result is passed through** the **activation Function** producing **one output for each hidden node**.

Reconstruction of RBM

- The Learning process consisting of **several Forward and Backward passes.**
- In the **Forward Pass,** RBM takes the **inputs and Translates them into a set of numbers** that ***encode the Inputs***
- In the **Backward Pass**, It takes the **set of numbers and translates them back** to ***form the Reconstructed Inputs.***
- In **Reconstruction**, the logic is pretty simple. You have the **activations,** which are the **inputs at this point** and then ***passed to the hidden layer and then to the input later***.
- After this, **new biases are obtained**, and **the reconstruction is the new output.**

CONTD..


Reconstruction
these biases are new
visible layer
hidden layer 1
r = b +
r = b +
r = b +
r = b +
econstructions are the new output
a
a
a
activations are the new input
$w_i \ldots w_n$
weights are the same

Training of Restricted Boltzmann Machine

- In RBM there are **Three Steps** through which the **entire RBM works:**
- **Step-1: FORWARD PASS:** In this phase, we take the **input layer** and using the concept of **weights and biased** we are going to **activate the hidden layer**. This process is said to be ***Feed Forward Pass***. In Feed Forward Pass we are identifying the **positive association and negative association.**

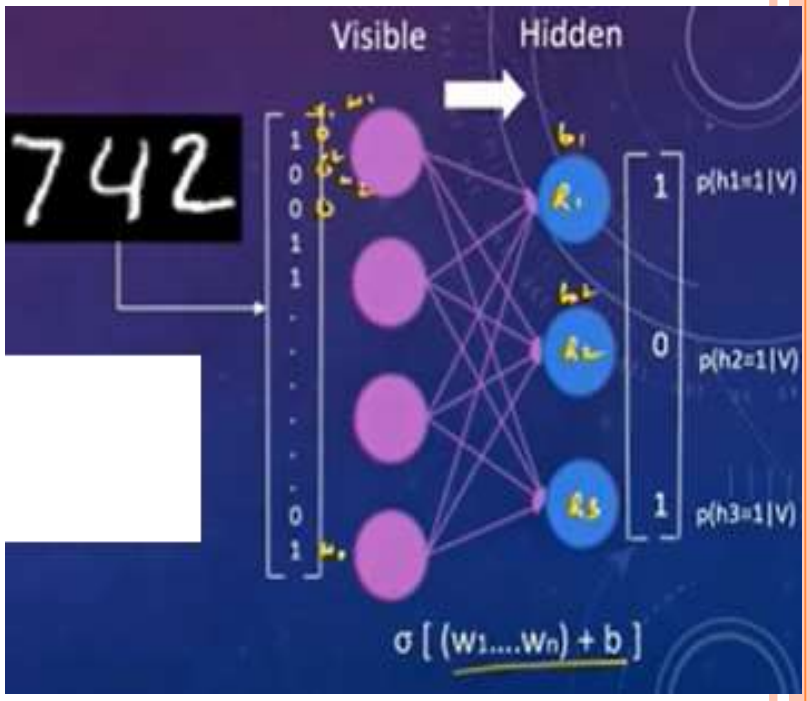
Feed Forward Equation:

Positive Association — When the association between the **visible unit and the hidden unit is positive.**

Negative Association — When the association between the **visible unit and the hidden unit is negative.**

CONTD..

- **For eg:** In **Forward Pass**, The **input image is converted into Binary Values.**
- Then the **vector input** is fed into the **network**, where:
  - Its **values are multiplied by weights** and **bias is added**
  - Then the result goes through the **Activation Function, Such as Sigmoid Function,**
  - Then it represents the **Probability of the node activation.**
  - i.e, it represents the which neuron **may or may not active.**



CONTD..

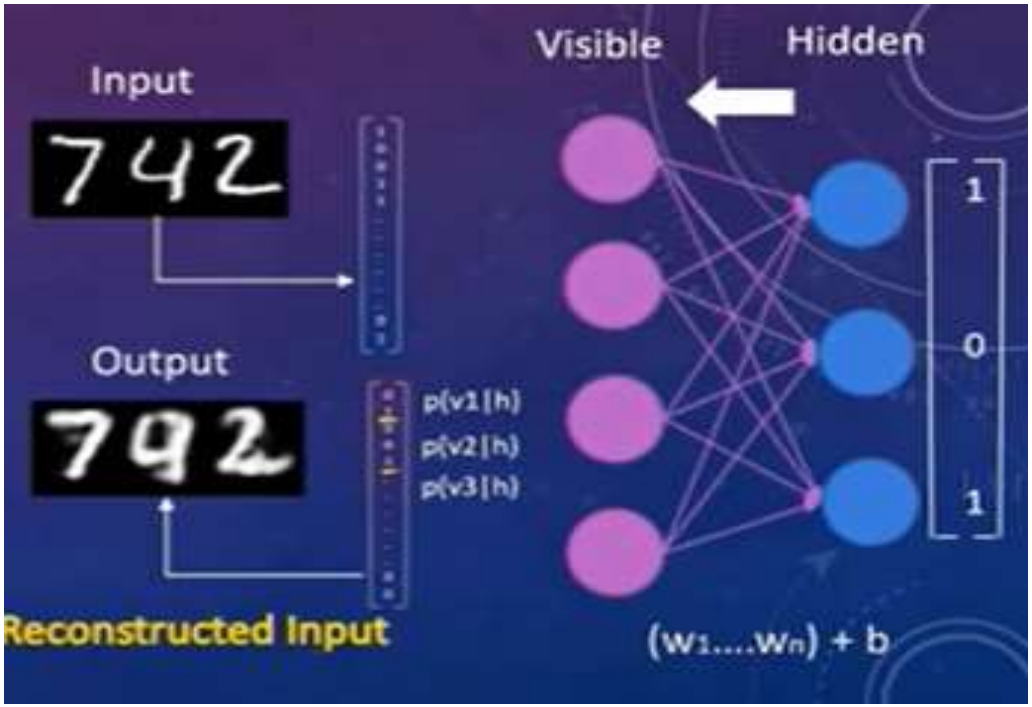
Step-2:BACK WARD PASS: As we don't have **any output layer**. Instead of calculating the **output layer**, we are **reconstructing the input layer through the activated hidden state.**

This process is said to be **Feed Backward Pass**. We are **just backtracking the input layer through the activated hidden neurons.**

After performing this we have **reconstructed Input through the activated hidden state.** So, we can **calculate the error and adjust weight in this way:**

CONTD..

- **For eg:** In **Backward Pass**, The **data that is passed backward.**
- It is also combined with the same weights and bias.
- Once the **information gets to the visible Layer** , the shape of the probability distribution of the **input values, sampling the distribution, the input is reconstructed.**



Step-3: Accessing Reconstruction Quality: Quality is compared by the original data.

The RBM calculates the error and adjust the weights and bias in order to minimize it.

Error = Reconstructed Input Layer-Actual Input layer

Adjust Weight = Input*error*learning rate (0.1)

INTERNAL TRAINING OF RESTRICTED BOLTZMANN MACHINE

- The training of the Restricted Boltzmann Machine **differs from the training of regular neural networks** via stochastic gradient descent.
- Here, **Visible Units** only talk to **Hidden Units** and
- **Hidden Units** talk to **Visible Units.**

HIDDEN	$z_i = \sum_j w_{ji} v_i + b_i$	$P(h_i = 1) = \frac{1}{1 + e^{-z_i}}$
VISIBLE	$y_i = \sum_j w_{ji} h_i + b_i$	$P(v_i = 1) = \frac{1}{1 + e^{-y_i}}$

CONTD..

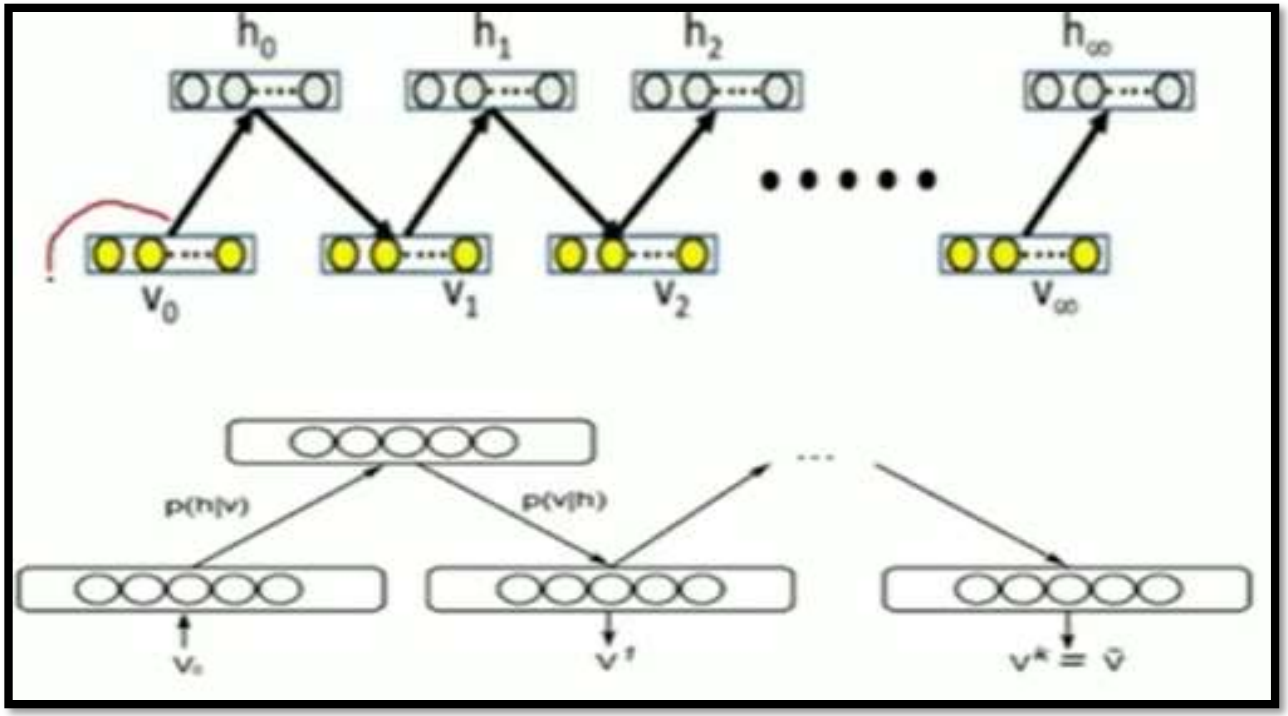
- **The Two main Training steps are:**

  1. Gibbs Sampling: The **first part of the training** is called **Gibbs Sampling.** Given an **input vector v** we **use p(h|v)for prediction** of the **hidden values h**.

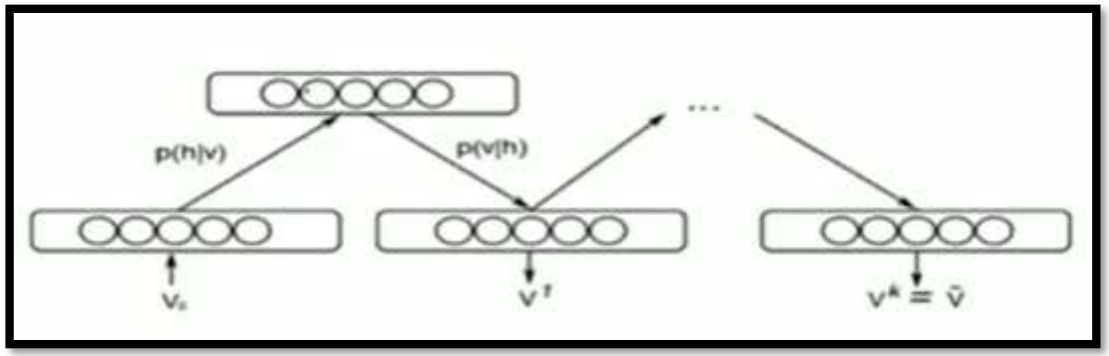
- Remember that we all know about **P(A/B).** Ie, we are fining the **probability of A given B.**
- In the same way, P(hj=1|V) is that given the input vector V, trying to calculates the values of the Hidden Vector.

$$p(h_i = 1 \mid \mathrm{v}) = \frac{1}{1+e(-(\mathrm{b}_j + W_j v_i))} = \sigma\left(\mathrm{b}_j + \sum_i v_i w_{ij}\right)$$

CONTD..



For **prediction of new input values v**. This **process is repeated k times.** After **k iterations**, we obtain **another input vector v_k** which was **recreated from original input values v_0.**



$$p(v_i = 1 \mid h) = \frac{1}{1+e(-(a_i+w_i h_j))} = \tilde{O}\,(a_i + \sum h_j w_{ij})$$

2. Contrastive Divergence step: The **update of the weight matrix** happens during the **Contrastive Divergence step**.

Vectors v_0 and v_k are used to **calculate the activation probabilities** for **hidden values h_0 and h_k :**

- The **difference between** the **outer products of those probabilities** with **input vectors v_0 and v_k results** in the **updated matrix :**

$$\Delta W = v_0) \otimes P(h_0 \mid v_0) - v_k \otimes P(h_k \mid v_k) - v_k)$$

- Using the **update matrix the new weights** can be **calculated with gradient ascent,** given by:

$$W_{new} = W_{old} + \triangle W$$

- Finally, **Contrastive divergence (CD) algorithm** is used to **train the RBM.** The algorithm performs **Gibbs sampling** and is used **inside a gradient descent procedure** (similar to the way **back propagation is used inside such a procedure** when training feed forward neural nets) to compute **weight update.**

USE CASES OF RBM

- **Pattern recognition :** RBM is used for **feature extraction** in **pattern recognition problems** where the challenge is to understand the **hand written text or a random pattern.**
- **Recommendation Engines** : RBM is widely used for **collaborating filtering techniques** where it is used to **predict what should be recommended to the end user.** so that the user enjoys using a **particular application or platform**.
  - **For eg : Movie Recommendation, Book Recommendation**

Applications of RBM

Hand Written Digit Recognition is a very common problem these days and is used in a variety of applications like **criminal evidence, office computerization, check verification, and data entry applications**.

It also comes with challenges like **different writing style, variations in shape and size as well as image noise**, which leads to changes in **numeral topology**.

In this a **hybrid RBM-CNN methodology** is used for **digit recognition.**

First, **features are extracted using RBM deep learning algorithms**. Then extracted features are fed to the **CNN deep learning algorithm for classification.**

CONTD..

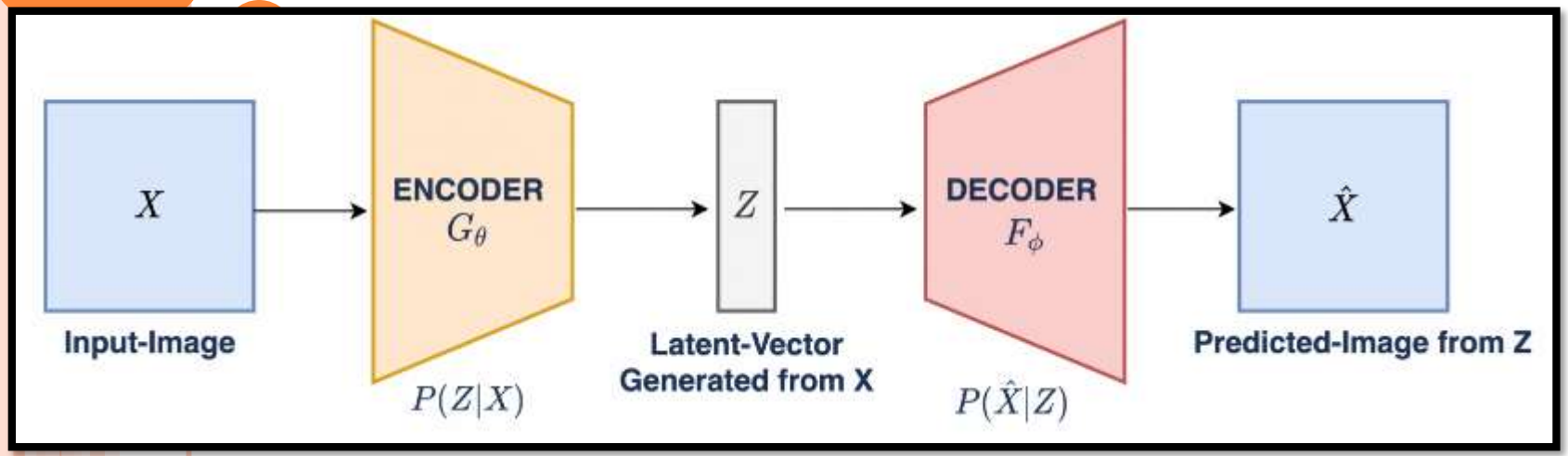
RBMs are highly capable for extracting features from input data. It is designed in such a way that it can extract the features from **large and complex datasets** by introducing **hidden units in an unsupervised manner.**

UNIT-II

PART – I

BUILDING BLOCKS OF DEEP LEARNING

AUTO ENCODERS



Contents

AUTOENCODERS

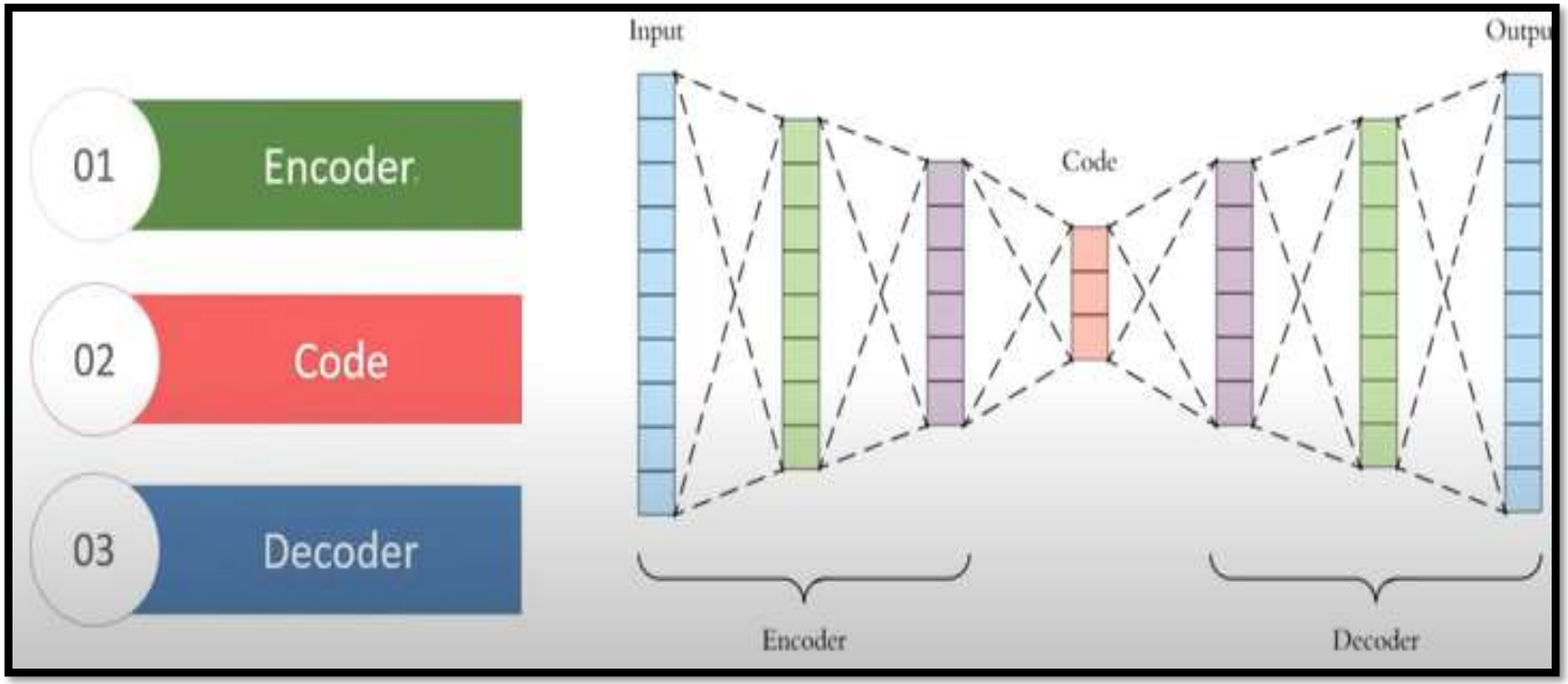
- **Autoencoders** fall under **unsupervised learning algorithms** as they learn the **compressed representation** of the data **automatically** from the **input data without labels.**
- **In Autoencoders Name,**
  - **AUTO** means =self
  - **ENCODERS** means- Convert into different form (reduced Dimensions)
- **Autoencoders (AEs)** are a type of **neural network architecture** that is able to find a ***compressed representation*** of the **input data** such as **image, video, text, speech, etc..**
- They became a popular solution for **reducing noisy data.**

CONTD..

- **Autoencoders** are used to reduce the size of our inputs into a smaller representation. If anyone needs the original data, they can reconstruct it from the compressed data.
- **Autoencoders** are a specific type of feed forward neural networks **trained to copy** its input to output.
- The **aim of an autoencoder** is to learn a lower-dimensional representation (encoding) for a higher-dimensional data, typically for dimensionality reduction, by training the network to capture the most important parts of the input image.

Architecture of Autoencoders

- Autoencoders consist of 3 parts:



An autoencoder replicates the data from the **input to the output** in an **unsupervised manner** and is therefore sometimes referred to as a **replicator neural network.**

CONTD..

- A generic way to **define an autoencoder** using a **mathematical notional** will be **f(x) = h**,
  - where **x is the input data** and **h is the latent variables** in the **information bottleneck**. This formula denotes the **encoder part** of the network
  - The basic **goal of an autoencoder** is to make the **output x as close to the input x as possible.**
  - Now it **cannot always be done if the number of neurons in the middle layer is much smaller than in the bottom or the top**.

CONTD..

- **Encoder:** An encoder is a **feed forward, fully connected neural network.**
- The **encoder** is the **first part of the autoencoder** and is responsible for t**ransforming the input data into a compressed representation**. This compressed representation is often referred to as the **latent space or bottleneck.**
- The encoder consists of **one or more layers of neurons,** typically implementing **non-linear activation functions like ReLU (Rectified Linear Unit)** to capture complex patterns in the input data.
  - First, the encoder takes the input and encodes it. For

CONTD..

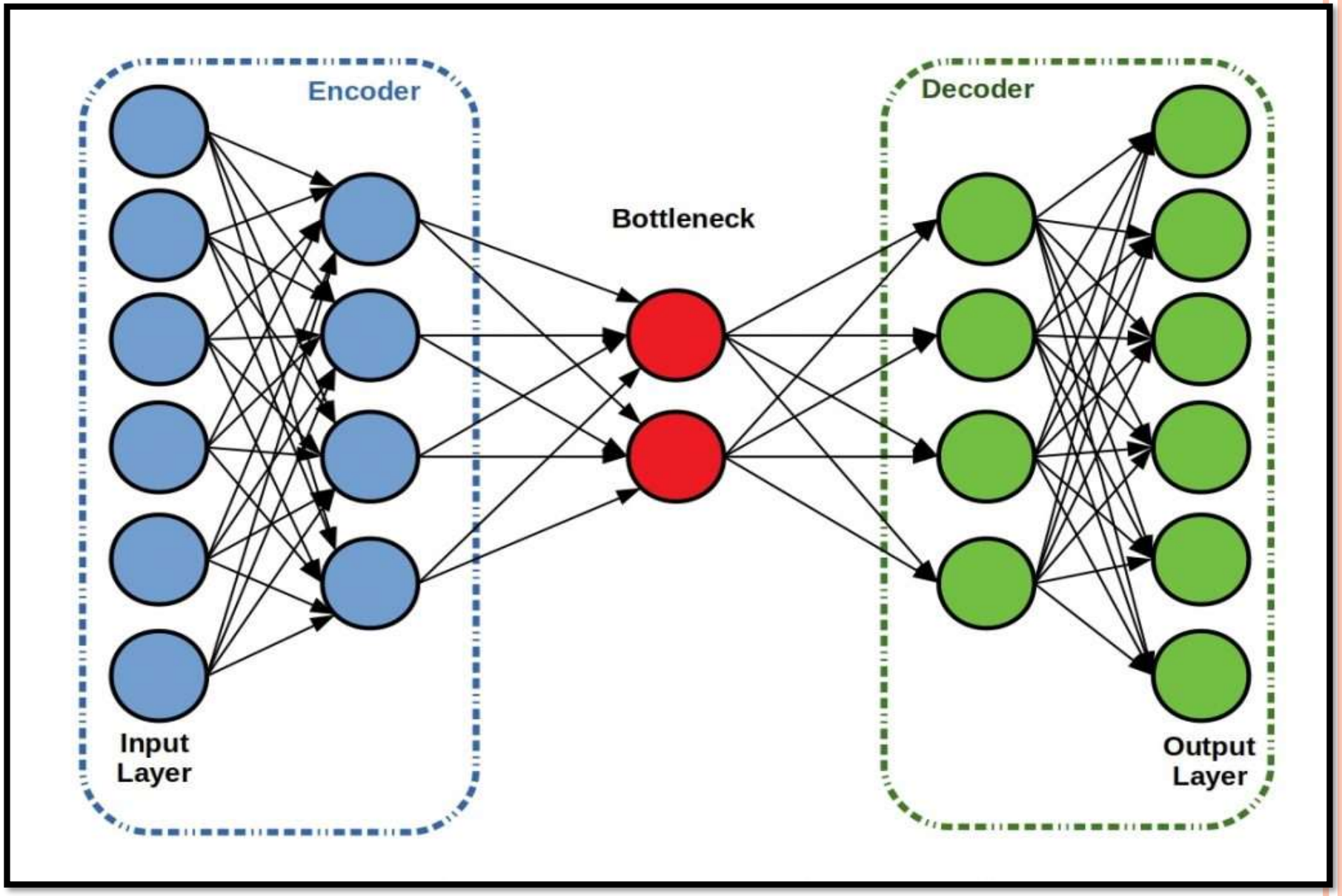
- First, the **encoder takes the input and encodes it**. For example, let the **input data be x**. Then, we can **define the encoded function as f(x).**

- **Code:** Code is the representation of **compressed input** which applies **to the decoder**. This part of the network is also refer as **bottleneck**. It **balances two factors** such as **which part of information to be taken** and **which part of information to be discarded.**

CONTD..

- **Decoder:** This layer decodes the encoded image back to the original dimension. The decoded image is a lossy reconstruction of the original image and it is reconstructed from the latent space representation.
- The layer between the encoder and decoder, ie. the code is also known as Bottleneck. The bottleneck layer is the layer in the middle of the autoencoder, where the compressed representation is stored.
- It has a smaller number of neurons compared to the input and output layers, effectively reducing the dimensionality of the data.

CONTD..



Mathematical Representation

- Essentially, we split the network into two segments, the encoder, and the decoder.

$$\phi : \mathcal{X} \rightarrow \mathcal{F}$$
$$\psi : \mathcal{F} \rightarrow \mathcal{X}$$
$$\phi, \psi = \underset{\phi,\psi}{\arg\min} \, ||X - (\psi \circ \phi)X||^2$$

- The encoder function, denoted by ϕ, maps the original data X, to a latent space F, which is present at the bottleneck.
- The decoder function, denoted by ψ, maps the latent space F at the bottleneck to the output.
- The output, in this case, is the same as the input function. Thus, we are basically trying to recreate the original image after some generalized non-linear compression.

CONTD..

- The encoding network can be represented by the standard neural network function passed through an activation function, where z is the latent dimension

$$\mathbf{z} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

- Similarly, the decoding network can be represented in the same fashion, but with different weight, bias, and potentially activation functions being used.

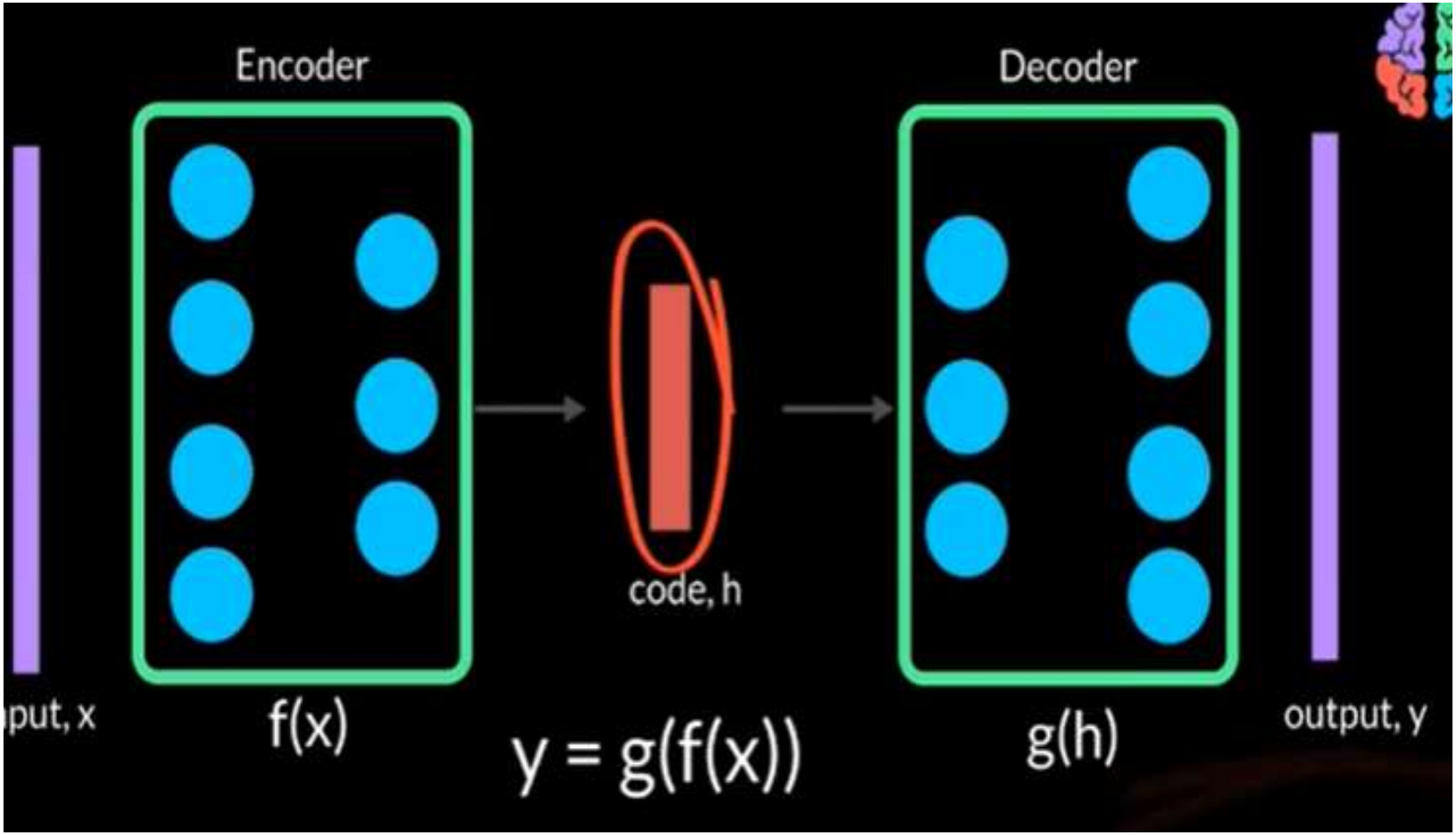
$$\mathbf{x}' = \sigma'(\mathbf{W}'\mathbf{z} + \mathbf{b}')$$

CONTD..

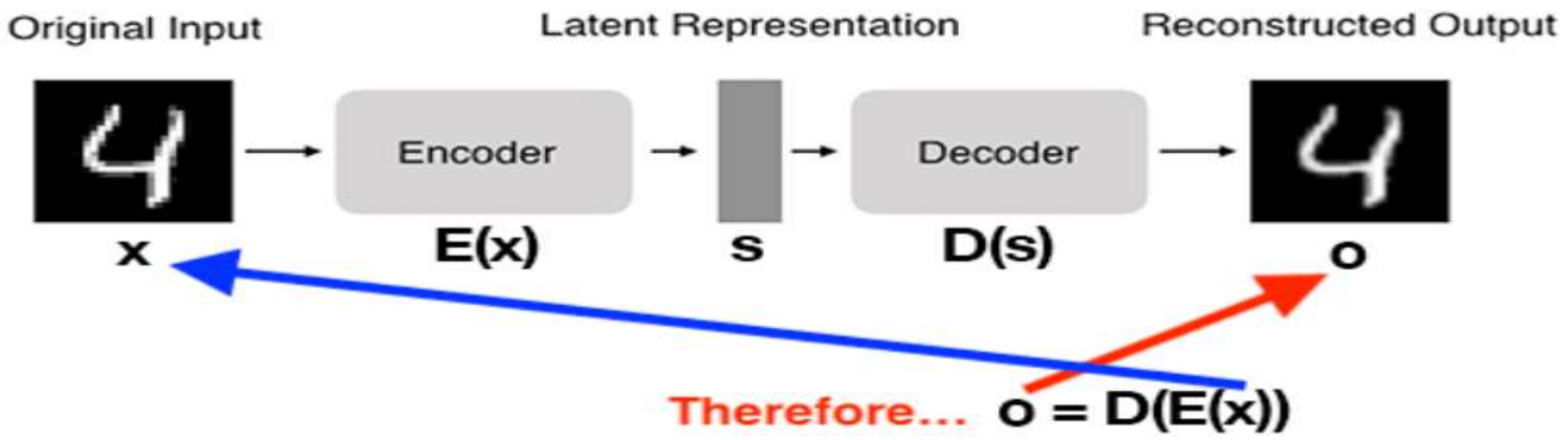
- The loss function can then be written in terms of these network functions, and it is this loss function that we will use to train the neural network through the standard back propagation procedure.

$$\mathcal{L}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|^2 = \|\mathbf{x} - \sigma'(\mathbf{W}'(\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})) + \mathbf{b}')\|^2$$

CONTD..



Example



In this example, you can observe the **following steps:**

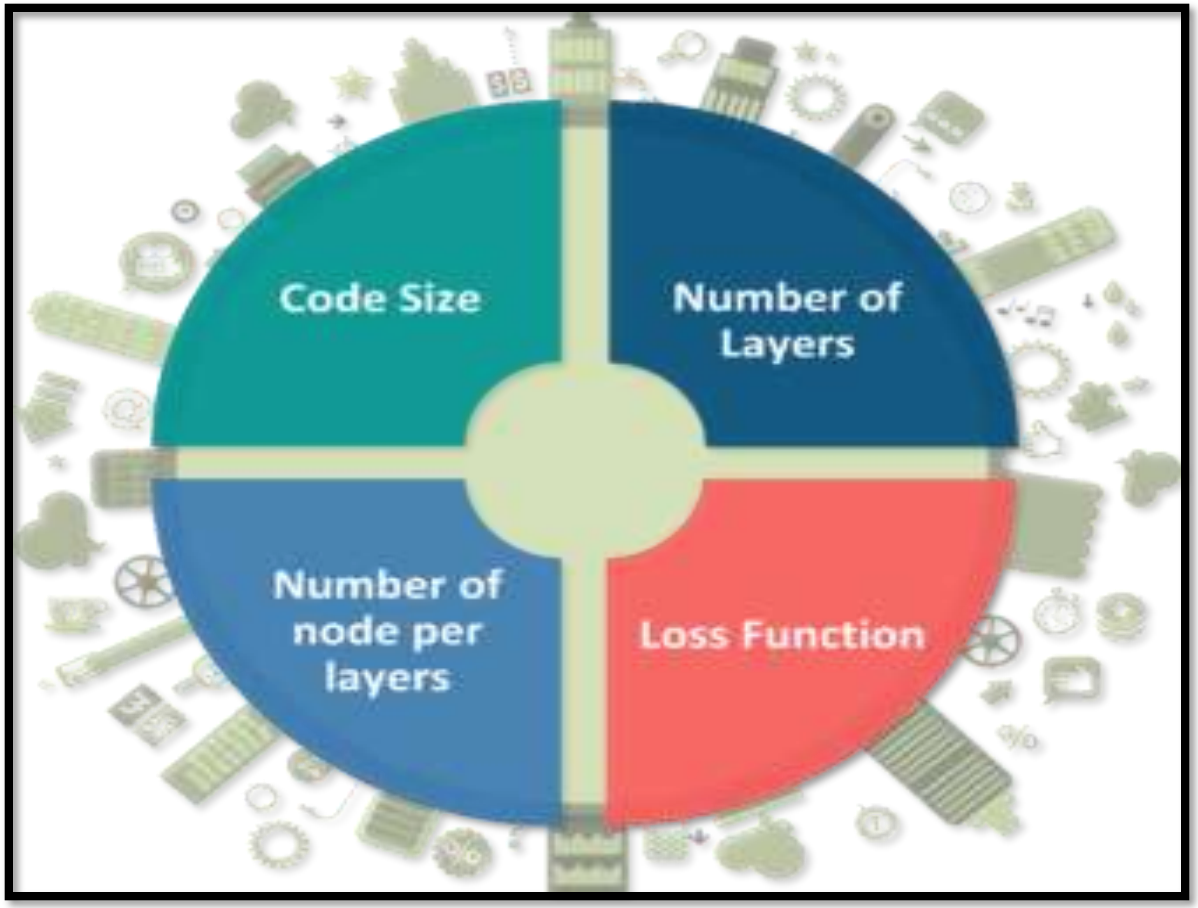
- The **input a digit** into the **autoencoder.**
- The **encoder sub network** generates a **latent representation of the digit 4** which is considerably **smaller in dimensionality than the input**.
- The **decoder sub network reconstructs the original digit** using the **latent representation.**

Properties of Autoencoders

- 1. Unsupervised Leaning: Autoenoders are considered as unsupervised learning technique. Since they **don't need explicit labels to train on.**
- 2. Data-specific: Autoencoders are only able to compress data similar to what they have been trained on.
- 3. Lossy: The decompressed outputs will be degraded compared to the original inputs.
- 4. Learned automatically from examples: It is easy to train specialized instances of the algorithm that will perform well on a **specific type of input.**

PARAMETERS OF AUTOENCODERS

here are **4 hyperparameters** that we need to set before **training an autoencoder**:



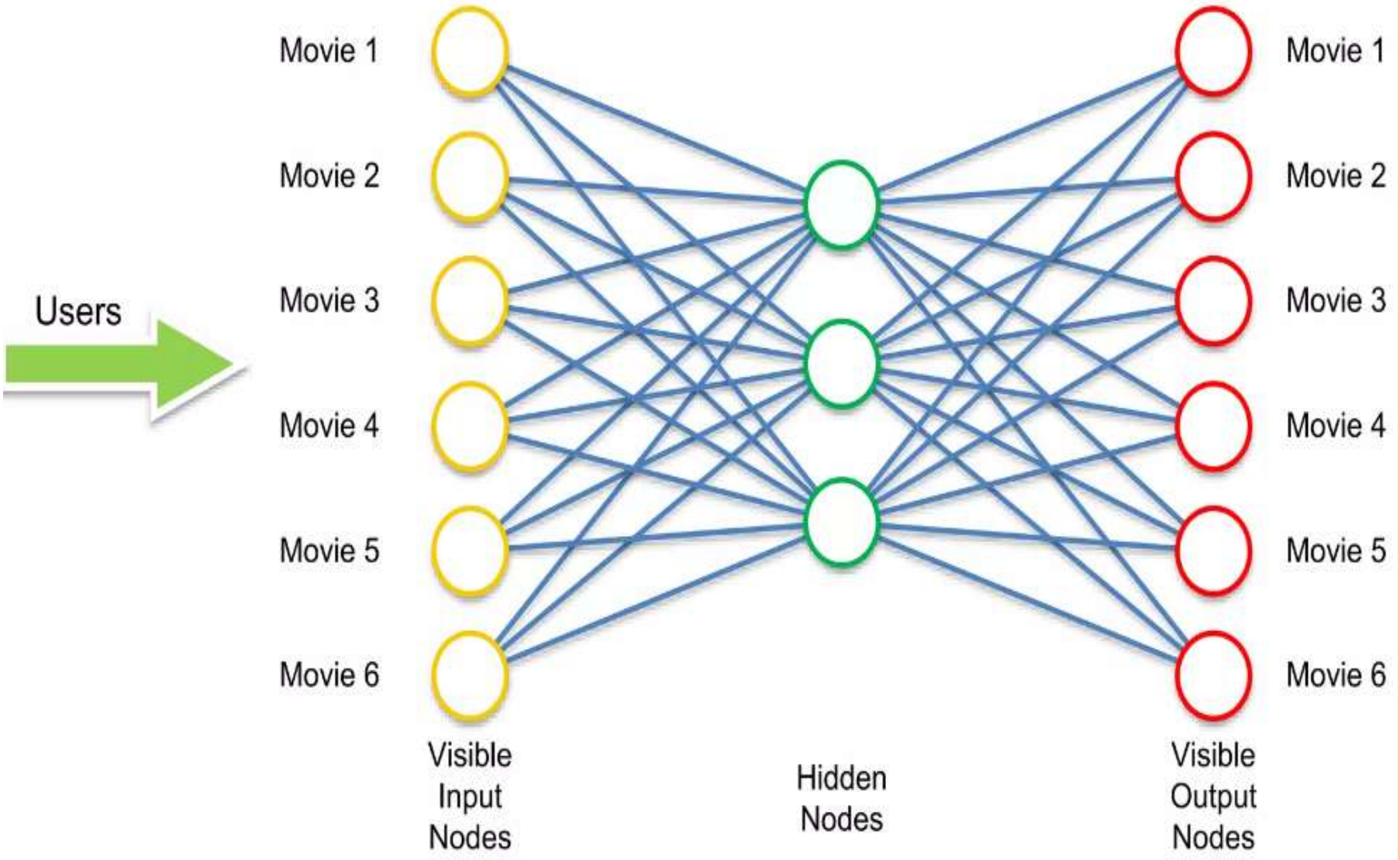
CONTD..

- Code size: It represents the number of nodes in the middle layer. Smaller size results in more compression.
- Number of layers: The autoencoder can consist of as many layers as we want.
- Number of nodes per layer: The number of nodes per layer decreases with each subsequent layer of the encoder, and increases back in the decoder. The decoder is symmetric to the encoder in terms of the layer structure.

CONTD..

- Loss function: We either use **mean squared error** or **binary cross-entropy.** If the input values are in the **range [0, 1]** then we typically use **cross-entropy**, otherwise, we use the **mean squared error.**

Training of Autoencoders



CONTD..

STEP 1: We start with an array where the lines (the observations) correspond to the users and the columns (the features) correspond to the movies. Each cell (u, i) contains the rating (from 1 to 5, 0 if no rating) of the movie i by the user u.

STEP 2: The first user goes into the network. The input vector $x = (r_1, r_2, \ldots, r_m)$ contains all its ratings for all the movies.

STEP 3: The input vector x is encoded into a vector z of lower dimensions by a mapping function f (e.g: sigmoid function):

$z = f(Wx + b)$ where W is the vector of input weights and b the bias

STEP 4: z is then decoded into the output vector y of same dimensions as x, aiming to replicate the input vector x.

STEP 5: The reconstruction error $d(x, y) = ||x-y||$ is computed. The goal is to minimize it.

STEP 6: Back-Propagation: from right to left, the error is back-propagated. The weights are updated according to how much they are responsible for the error. The learning rate decides by how much we update the weights.

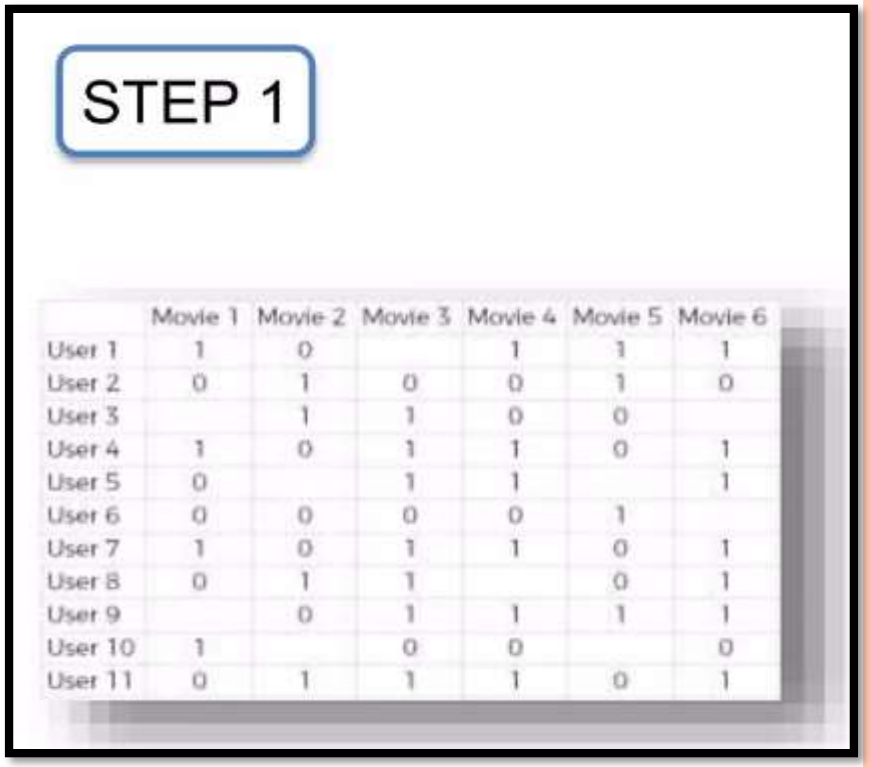
STEP 7: Repeat Steps 1 to 6 and update the weights after each observation (Reinforcement Learning). Or:
Repeat Steps 1 to 6 but update the weights only after a batch of observations (Batch Learning).

STEP 8: When the whole training set passed through the ANN, that makes an epoch. Redo more epochs.

CONTD..

Step-1: WE start **with an Array** where **the lines(the observations)** correspond to the **users** and the **columns(the Features)** corresponds to **the Movie.**

Each **Cell (u,i) contains** the **rating from 1 to 5 , 0 has no rating** of the **movie i by the user u**

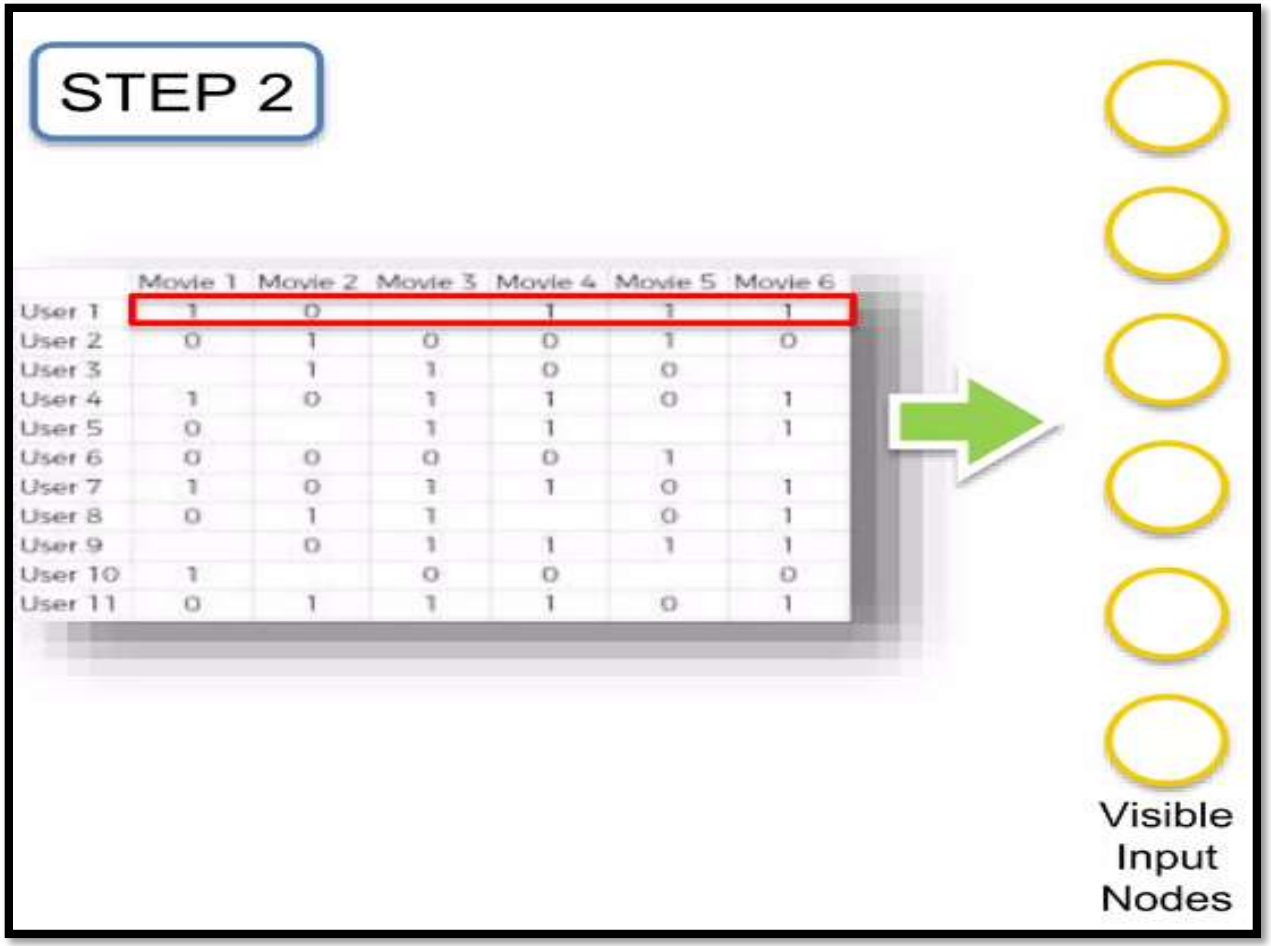


	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6
User 1	1	0		1	1	1
User 2	0	1	0	0	1	0
User 3		1	1	0	0	
User 4	1	0	1	1	0	1
User 5	0		1	1		1
User 6	0	0	0	0	1	
User 7	1	0	1	1	0	1
User 8	0	1	1		0	1
User 9		0	1	1	1	1
User 10	1		0	0		0
User 11	0	1	1	1	0	1

CONTD..

Step-2:

The first user goes into the network. The input vector $x = (r_1, r_2, \ldots, r_m)$ contains all its ratings for all the movies.

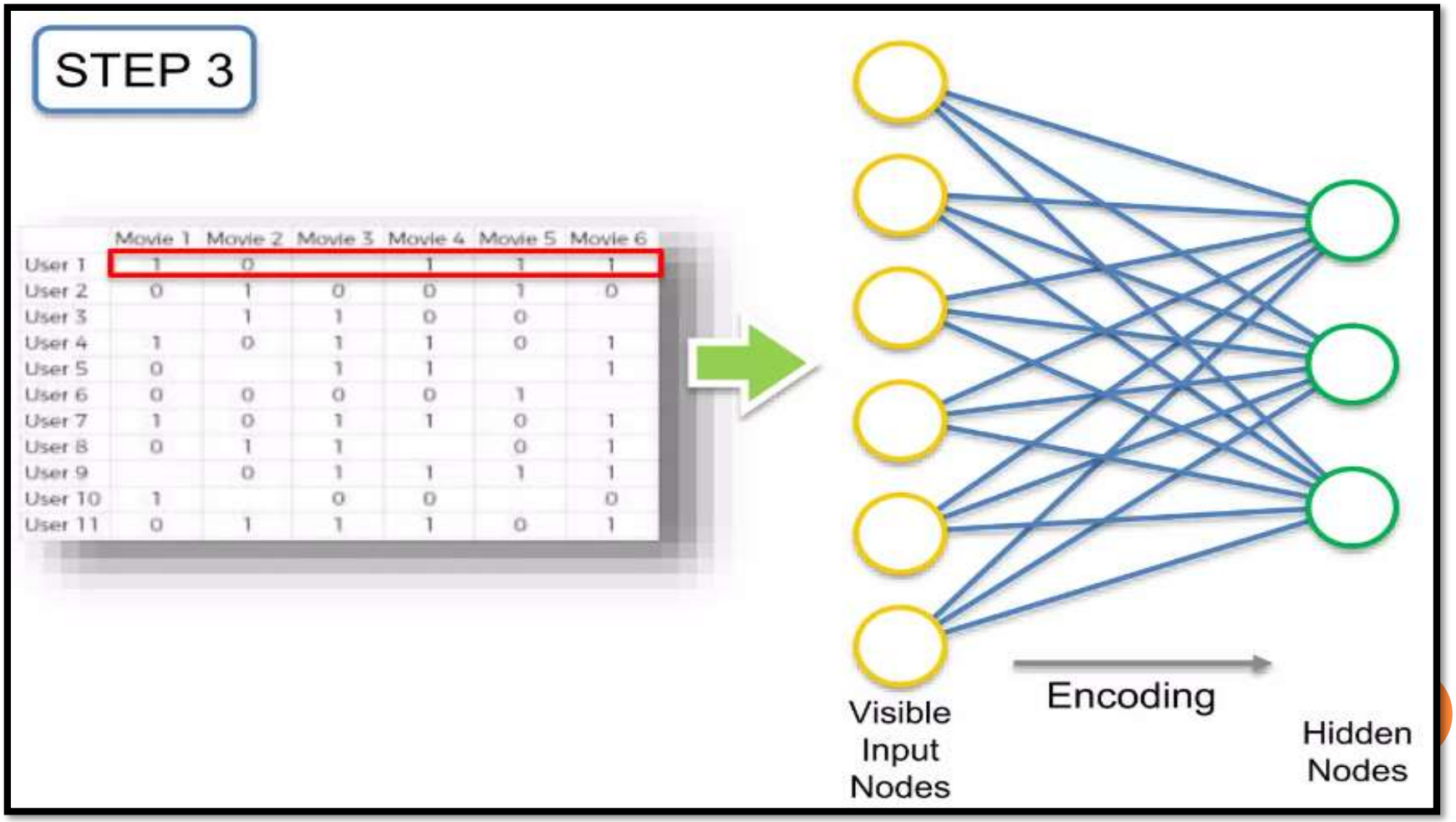


	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6
User 1	1	0		1	1	1
User 2	0	1	0	0	1	0
User 3		1	1	0	0	
User 4	1	0	1	1	0	1
User 5	0		1	1		1
User 6	0	0	0	0	1	
User 7	1	0	1	1	0	1
User 8	0	1	1		0	1
User 9		0	1	1	1	1
User 10	1		0	0		0
User 11	0	1	1	1	0	1

CONTD..

Step-3:

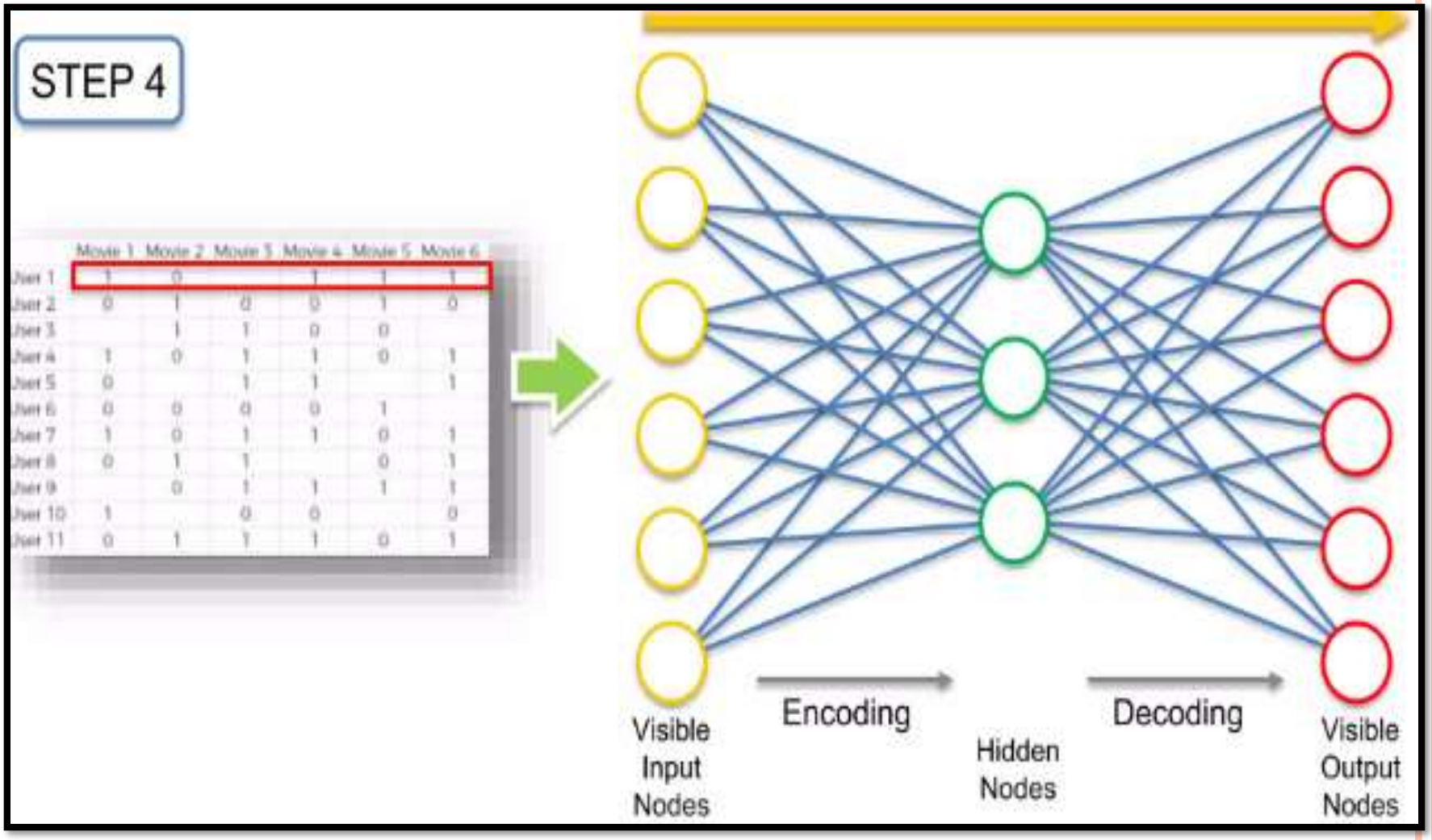
The input vector x is encoded into a vector z of lower dimensions by a mapping function f (e.g: sigmoid function): $z = f(Wx + b)$ where W is the vector of input weights and b the bias



CONTD..

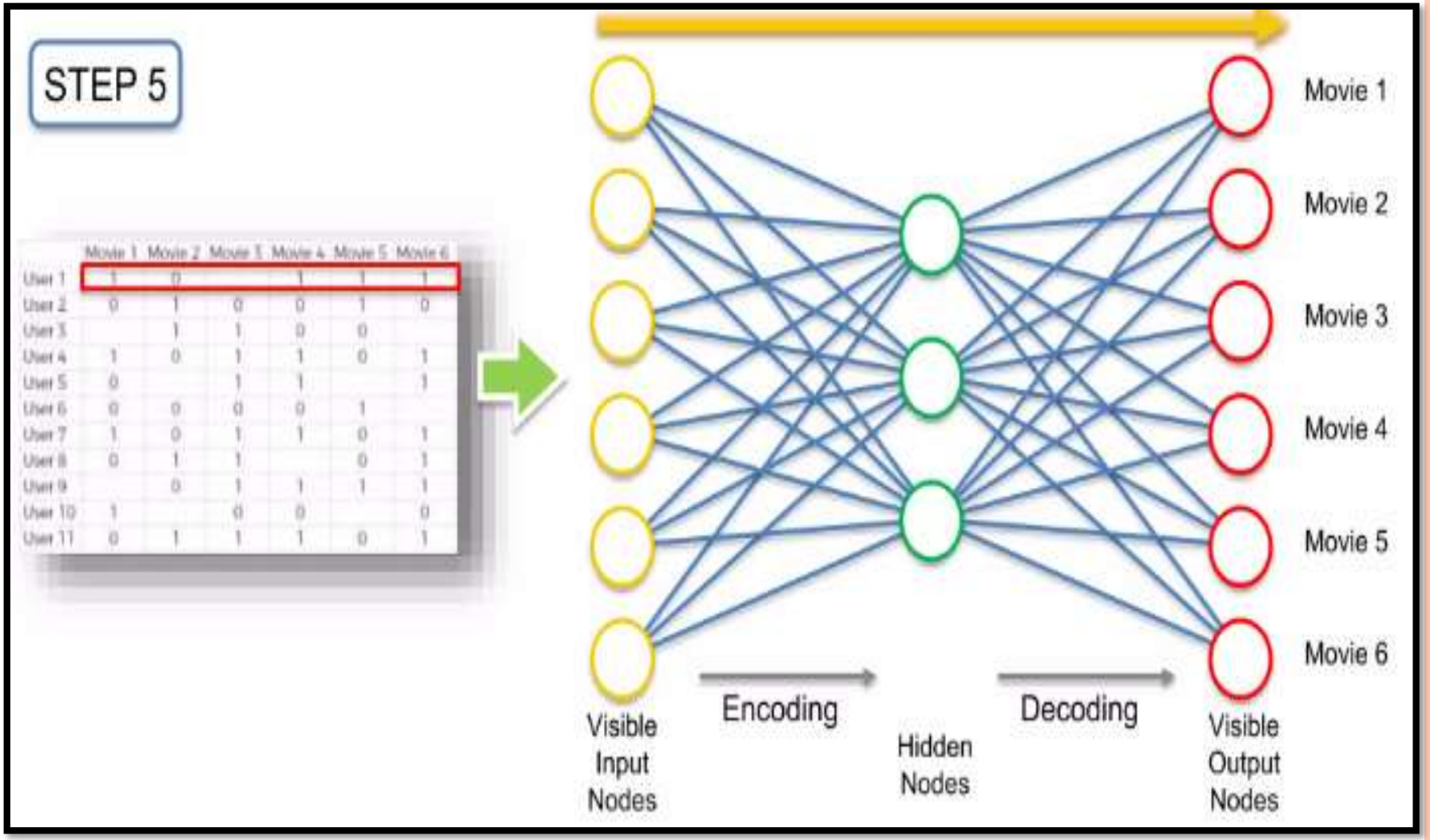
Step-4:

z is then decoded into the output vector y of same dimensions as x, aiming to replicate the input vector x.



CONTD..

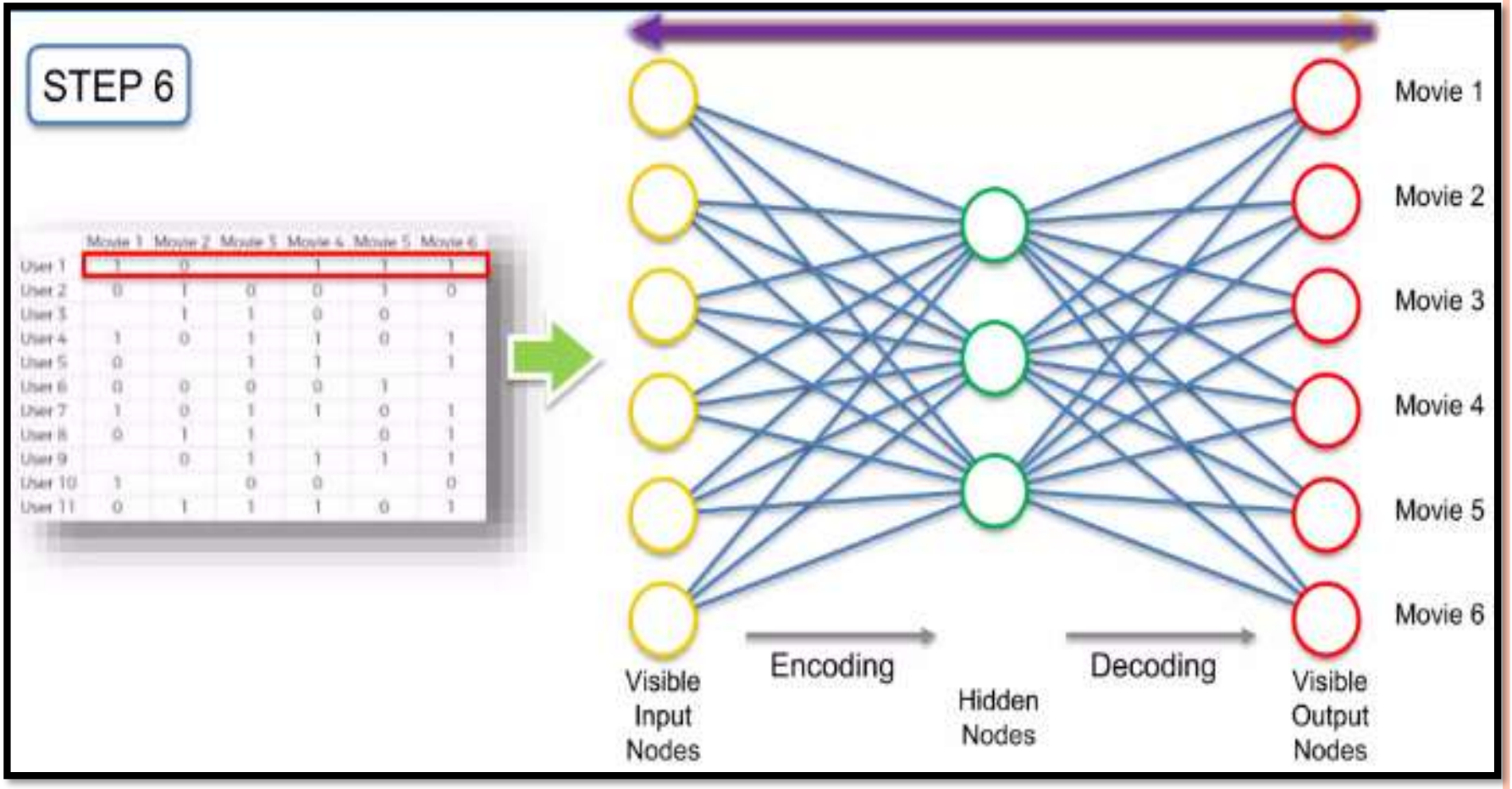
Step-5: The reconstruction error d(x, y) = ||x-y|| is computed. The goal is to minimize it.



CONTD..

Step-6:

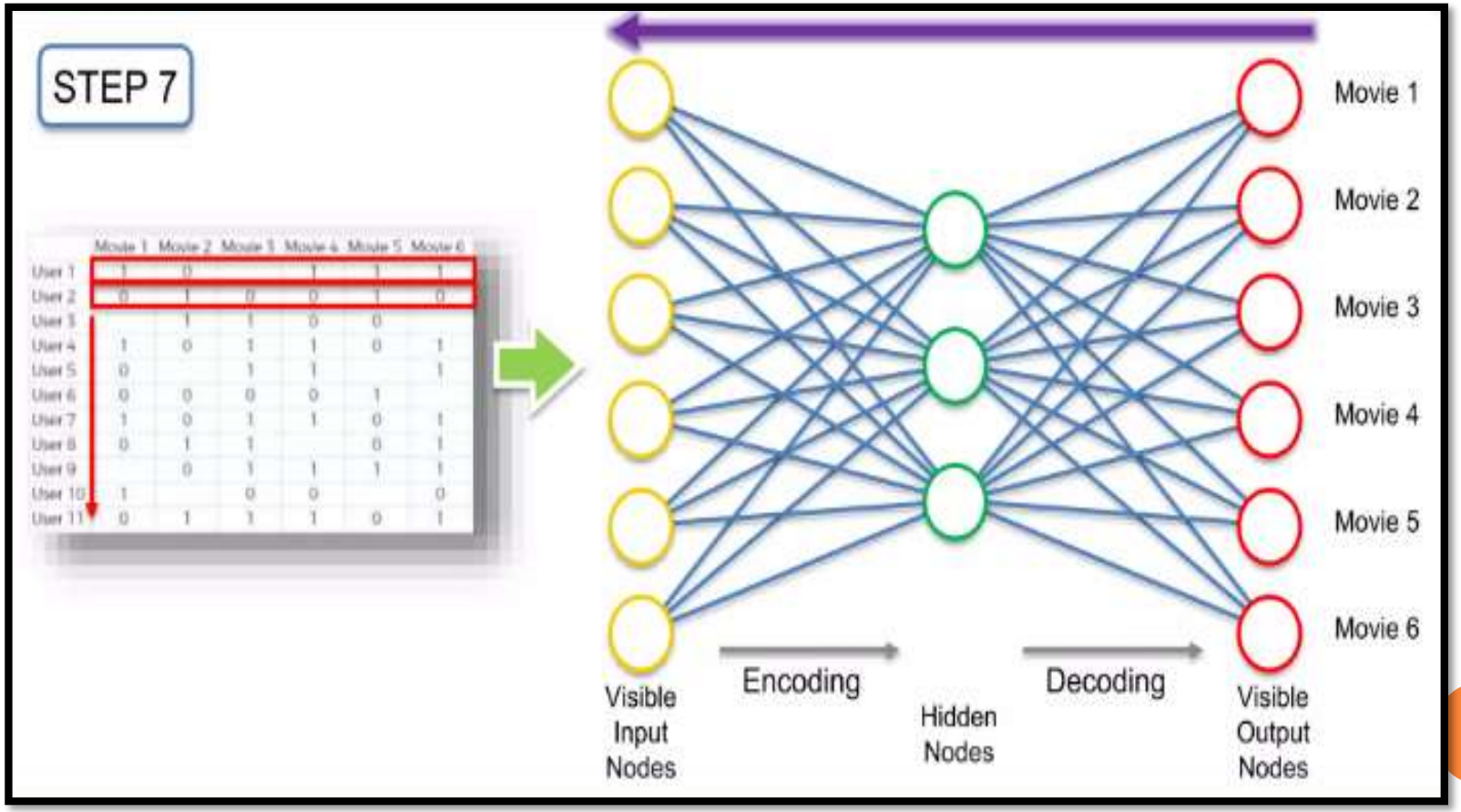
STEP 6: Back-Propagation: from right to left, the error is back-propagated. The weights are updated according to how much they are responsible for the error. The learning rate decides by how much we update the weights.



CONTD..

Step-7:

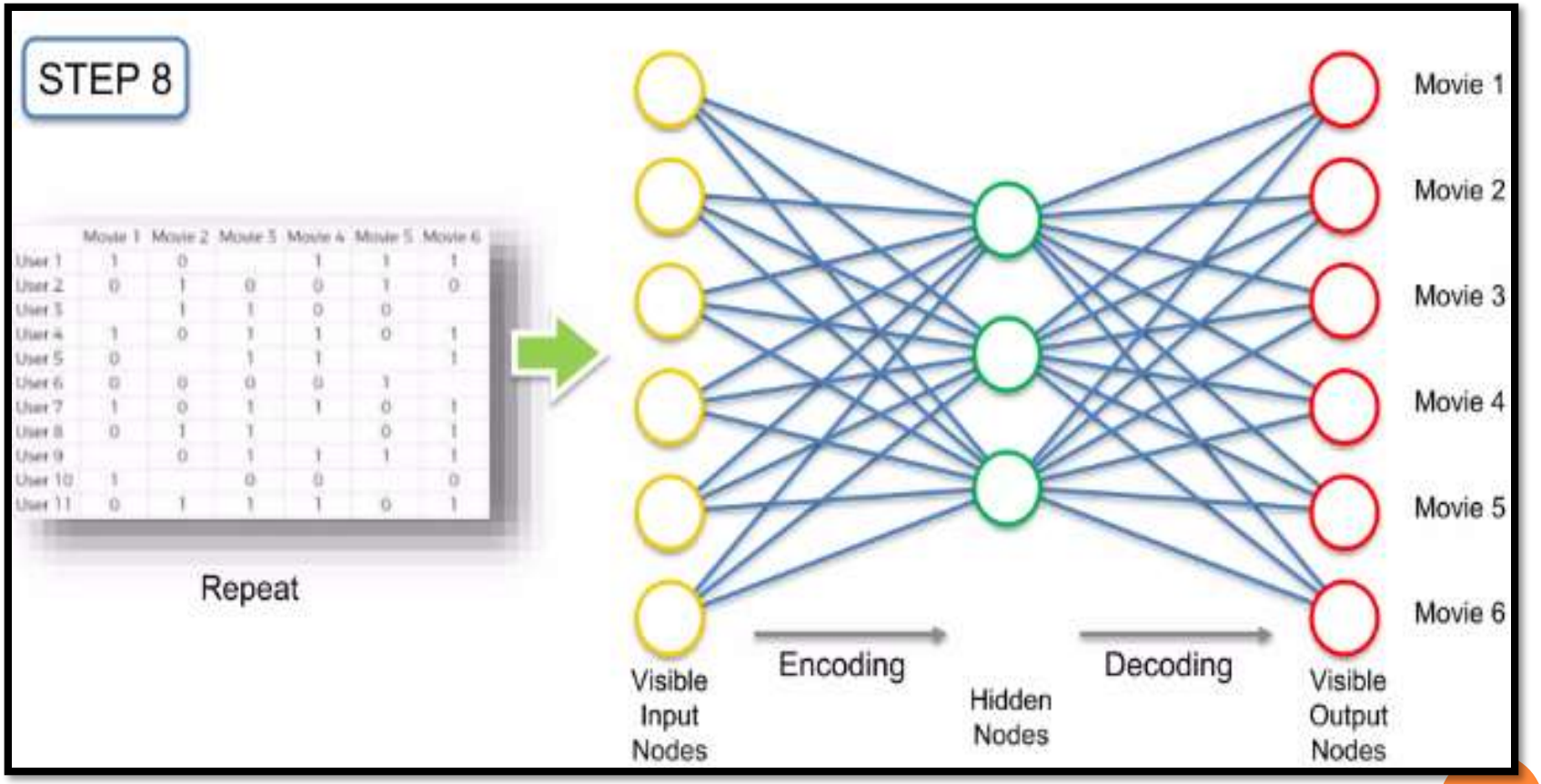
STEP 7: Repeat Steps 1 to 6 and update the weights after each observation (Reinforcement Learning). Or:
Repeat Steps 1 to 6 but update the weights only after a batch of observations (Batch Learning).



CONTD..

Step-8:

When the whole training set passed through the ANN, that makes an epoch. Redo more epochs.

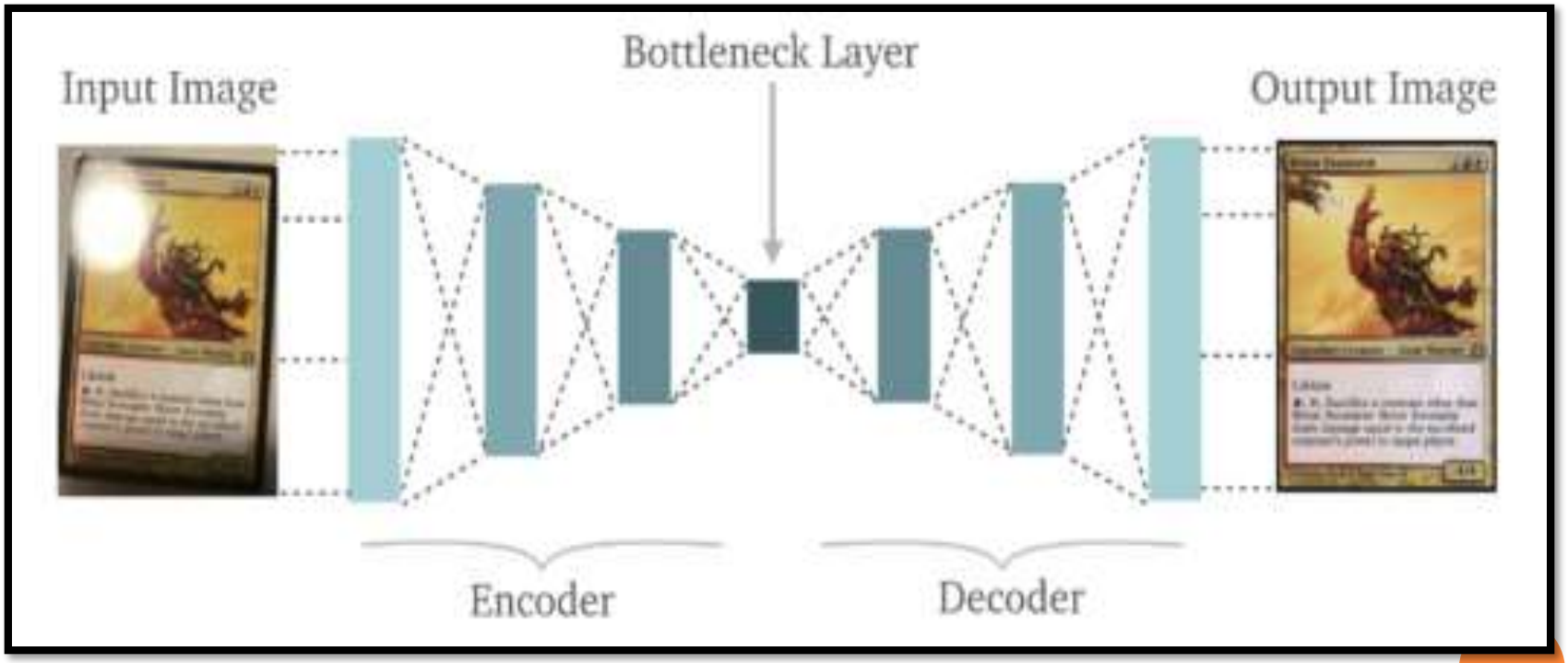


Applications of Autoencoders

- **1. Image Coloring:** Autoencoders convert **any black and white image into a colored image.** Depending on what is in the **picture** , it is **possible to tell what the colour should be**

CONTD..

- **2. Feature Variations:** Autoencoders **extract required features** from the **original input image** and then **generate noise free output image.**

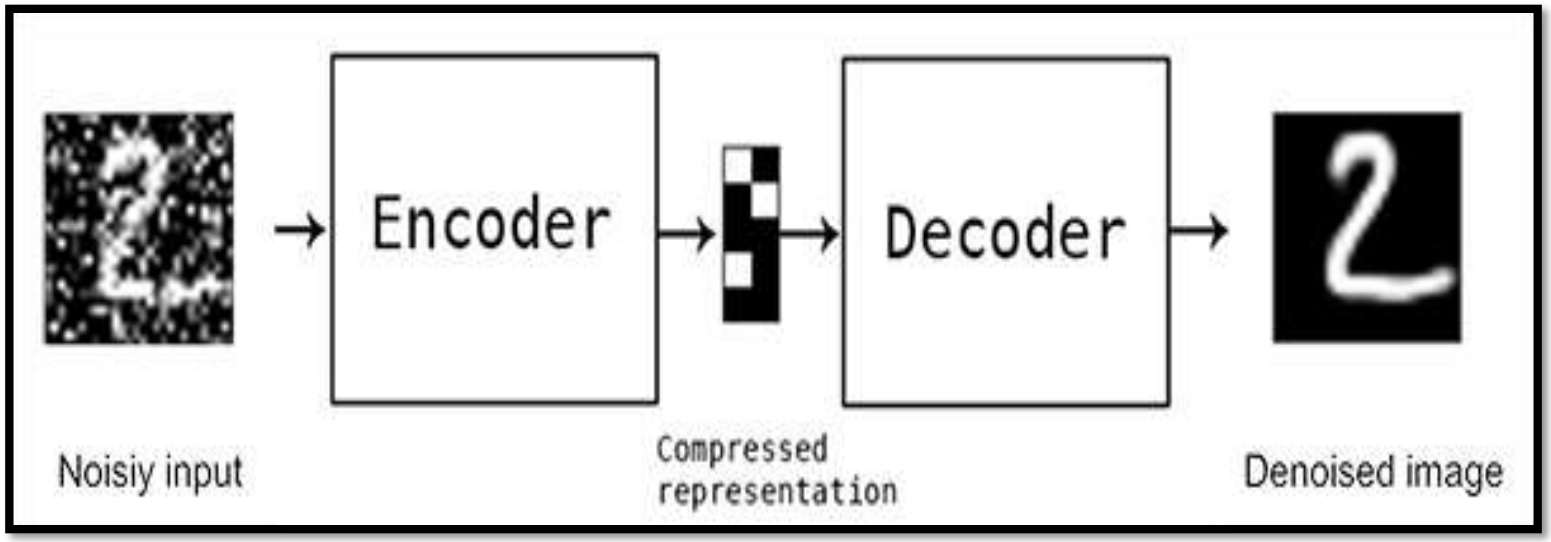


CONTD..

- **3. Dimensionality Reduction:** The **reconstructed image** is the **same as our input but reduced dimensions**. It helps in providing the **similar image with a reduced Pixel value.**

CONTD..

- **4. Denoising Image:** A denoising autoencoder is trained to **reproduce the clean image** from the **corrupted/noisy version.**

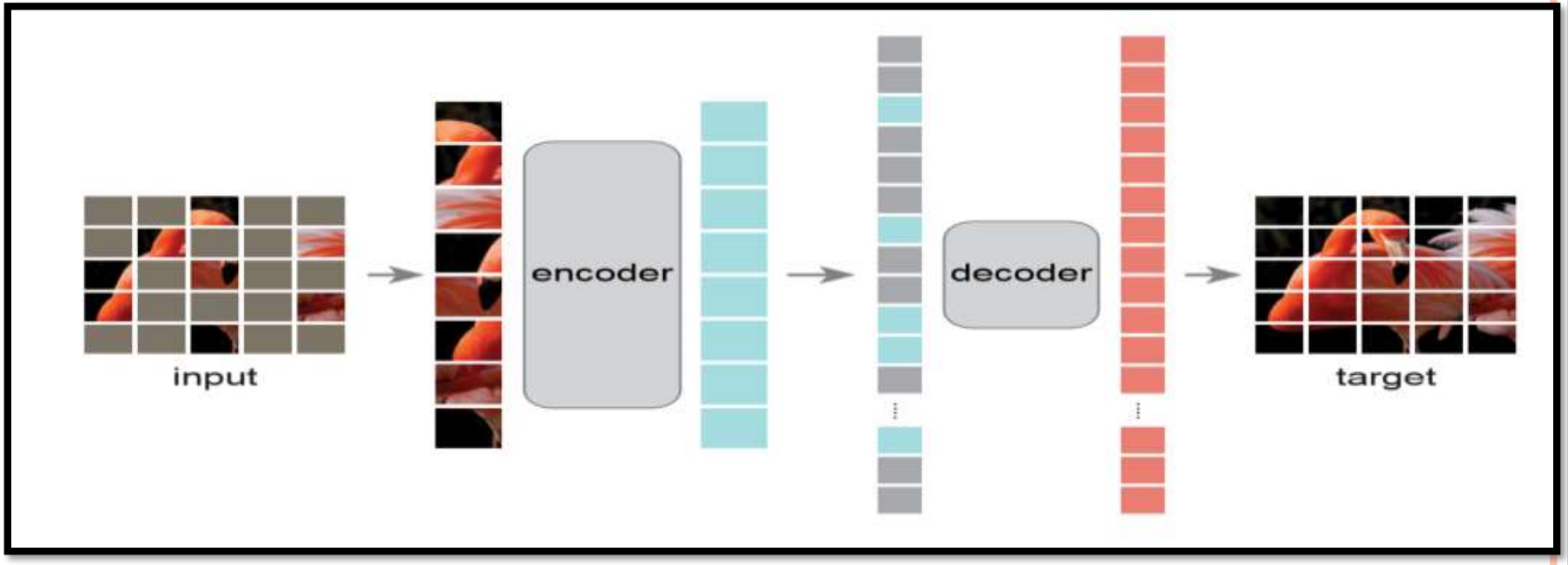


CONTD..

- **5. Watermark Removal:** It is also used for **removing watermarks from images** or to **remove any object** while **filming a video or a movie.**

How Autoencoder Works in Computer Vision

- Autoencoder models are commonly used for image processing tasks in computer vision.
- In this **use case**, the input is an image and the output is a reconstructed image. The model learns **to encode the image into a compressed representation**



Limitations of Autoencoders

- let's suppose we've trained an **autoencoder model** on a **large dataset of faces** with **a encoding dimension of 6**.
- An ideal autoencoder will learn **descriptive attributes** of faces **such as skin color, whether or not the person is wearing glasses, etc.** in an attempt to describe an observation in **some compressed representation.**

CONTD..

encoder

Smile: 0.99

Skin tone: 0.85

Gender: -0.73

Beard: 0.85

Glasses: 0.002

Hair color: 0.68

Latent attributes

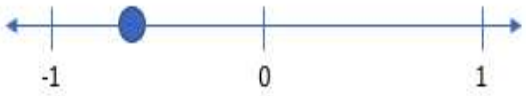
decoder

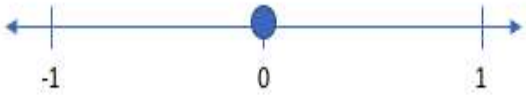
CONTD..

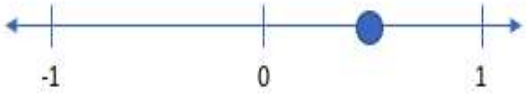
- In **the example above**, we've described the input image in terms of its **latent attributes using a single value** to **describe each attribute**.
- However, we may prefer to represent **each latent attribute** as **a range of possible values**.
- Using a variational autoencoder, we can describe ***latent attributes in probabilistic terms.***

CONTD..

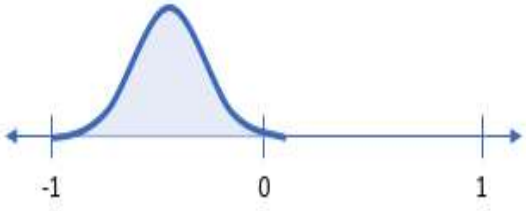
Smile (discrete value)

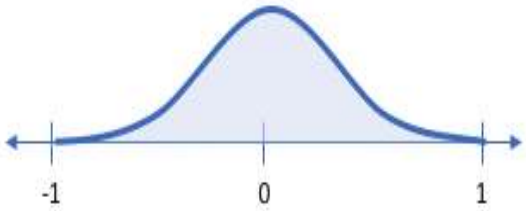




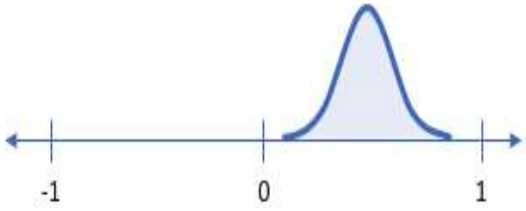


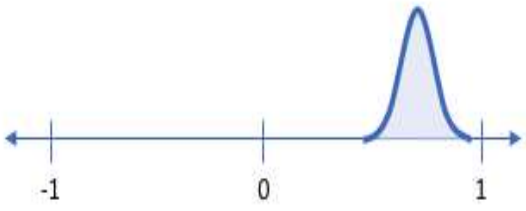
Smile (probability distribution)





VS.

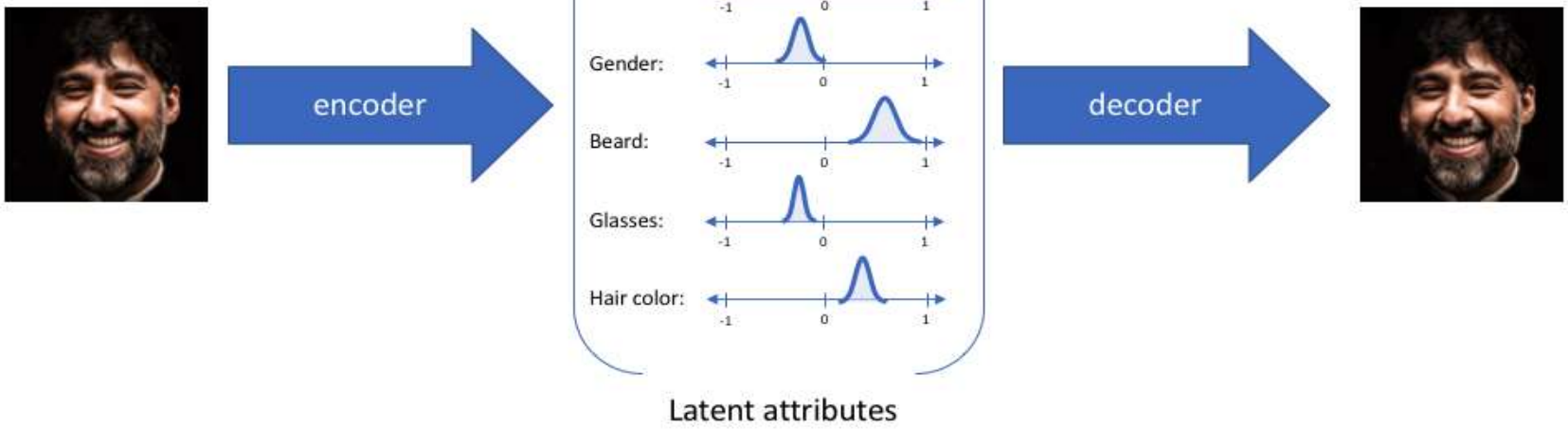




CONTD..

- With this approach, we will now represent ***each latent attribute*** for a **given input** as a ***probability distribution***.
- When **decoding from the latent state**, we will **randomly sample from each latent state distribution** to **generate a vector as input** for **our decoder model.**

CONTD..



Note: For **variational autoencoders**, the **encoder model** is sometimes referred to as the **recognition model** whereas the **decoder model** is sometimes referred to as the **generative model.**

Limitations of Autoencoder

- At this point, **a natural question** that comes in mind is "***what is the link between autoencoders and content generation?".***
- Here, once the **autoencoder has been trained**, we have **both an encoder and a decoder** but still **no real way to produce any new content.**
- At first sight, we could be **tempted to think that, if the latent space is regular enough** (well **"organized"** by the encoder during **the training process**)

CONTD..

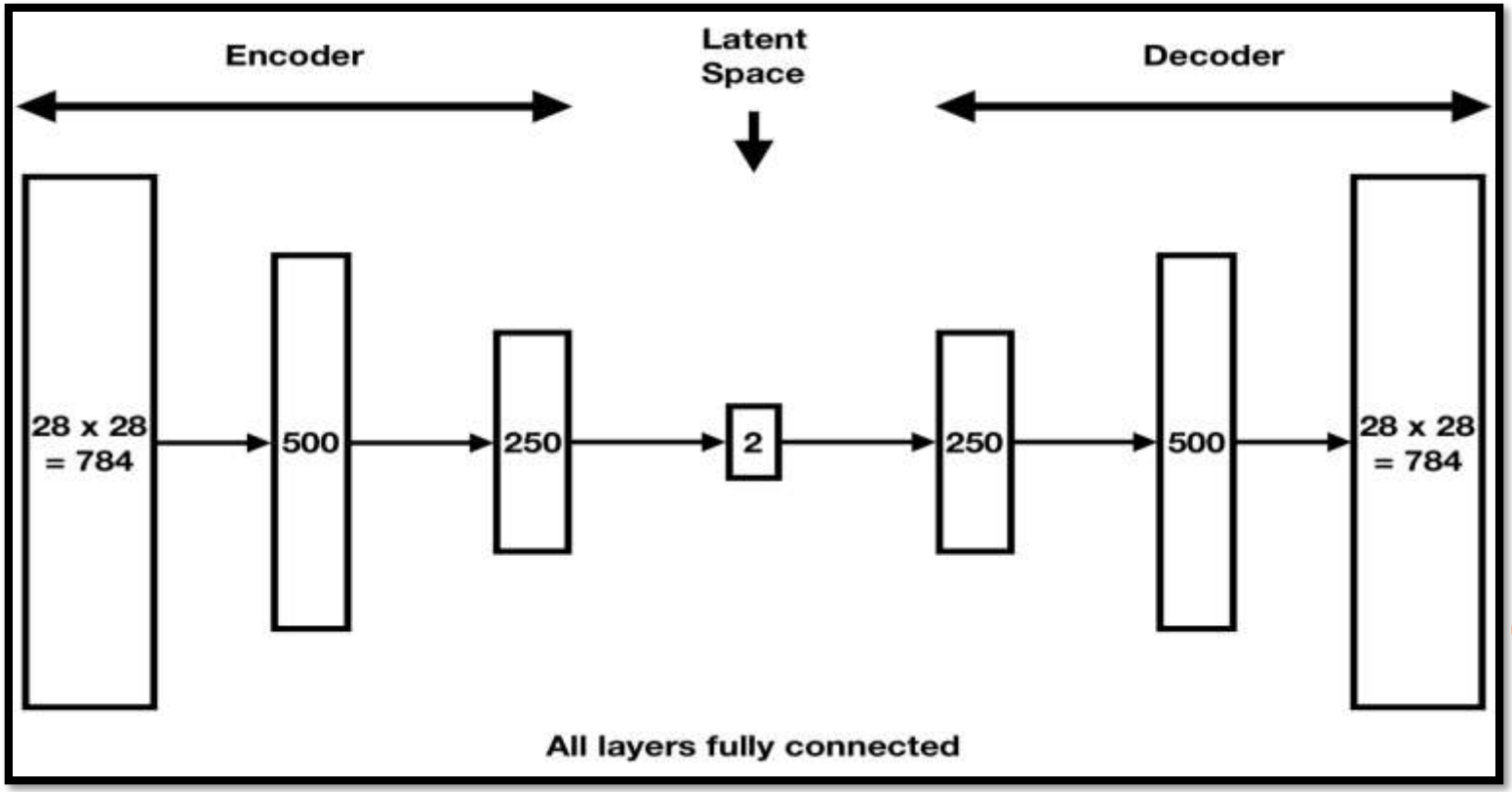
- we could take a point randomly from that latent space and decode it to get a new content.
- The decoder would then act more or less like the generator.
- But here we can't know how to get the decoder data.
- Suppose Autoencoder gives Good Results, OK well. But is does not gives the Good Results, there is a Problem.
- i.e, we have raised one question in our mind? How can Latent Space generate the samples.
- Lastly, autoencoders may not always generalize well to new, unseen data.

CONTD..

- This limitation is a challenge in applications where the model must perform well on data that is significantly different from the training set.
- Misunderstanding import variables
- Insufficient Training Data
- Imperfect decoding
- Generative Modeling

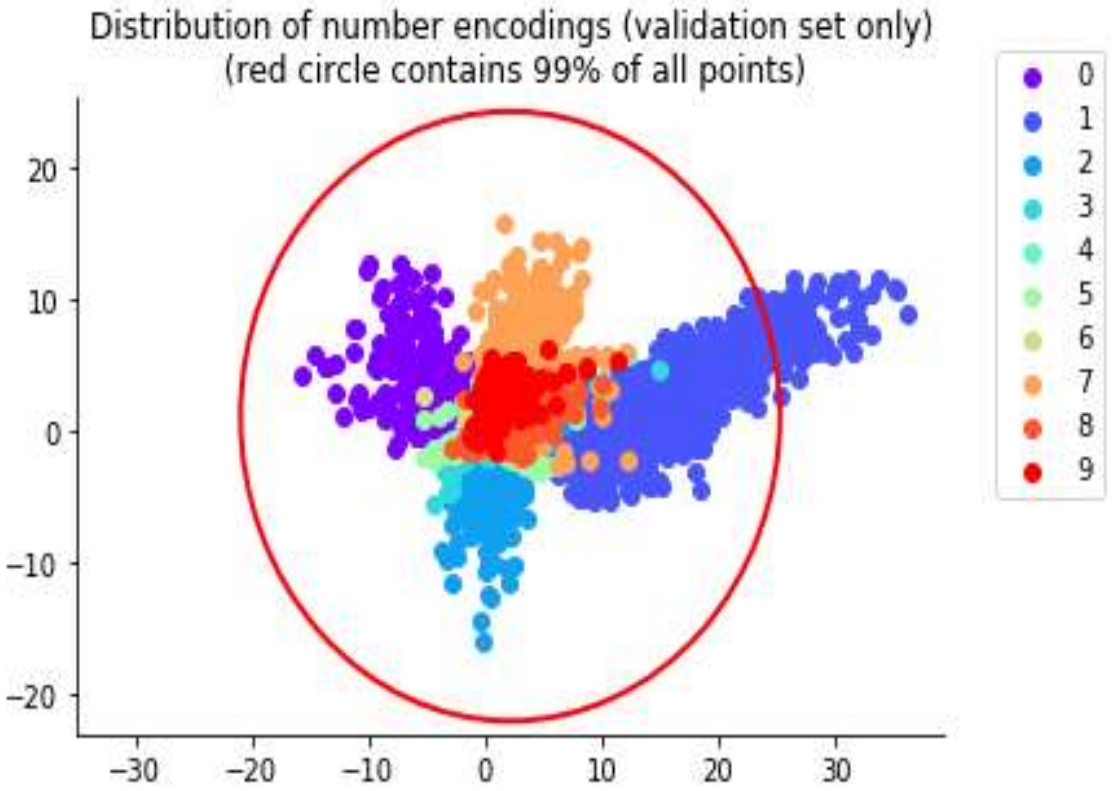
CONTD..

- **Another Real Time Example:** Using an autoencoder that compresses the famous MNIST 28 × 28 handwritten digits dataset. We use the following neural network architecture, which has a two-dimensional latent space:



CONTD..

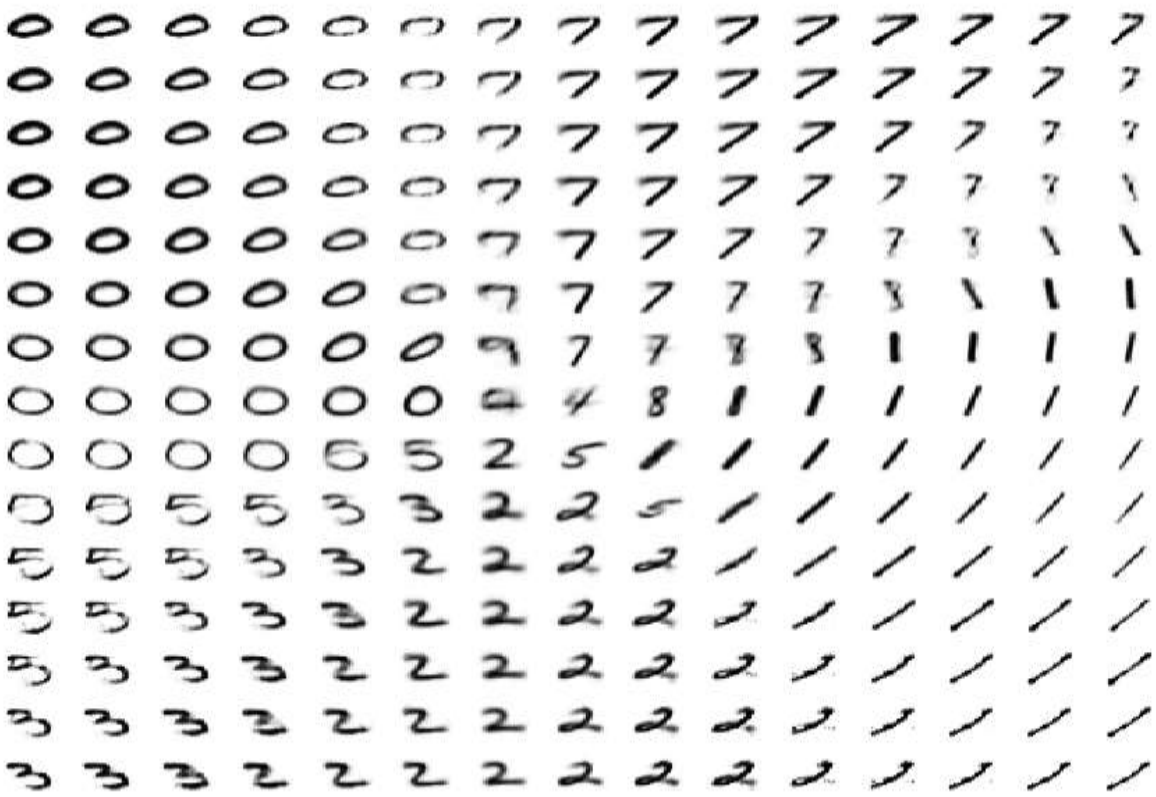
- After training our autoencoder (by minimizing its loss function via stochastic gradient descent), we find these encodings of the handwritten digits in our validation set, grouped by their labels 0–9:



CONTD..

- and **pass the vector at each lattice site through the decoder**, then we **generate** the following **"fake" handwritten digits:**

Numbers generated from latent variable values at sites of the square lattice inscribed in the red disk



CONTD..

- While **many images** have **good quality,** others are **blurry or incomplete** because **their sites are far from the point cloud.**
- **Drawback is :**
- The **quality of the generated handwritten digits** is **worse than in the two-dimensional case**.
- Because **two-dimensions is usually too small** to **capture the nuances of complicated data**, **this is not good news.**
- So we can introduced the advanced Topic Variational Autoencoder.

Thank You

UNIT-II

PART – I

BUILDING BLOCKS OF DEEP LEARNING

VARIATIONAL AUTO ENCODERS

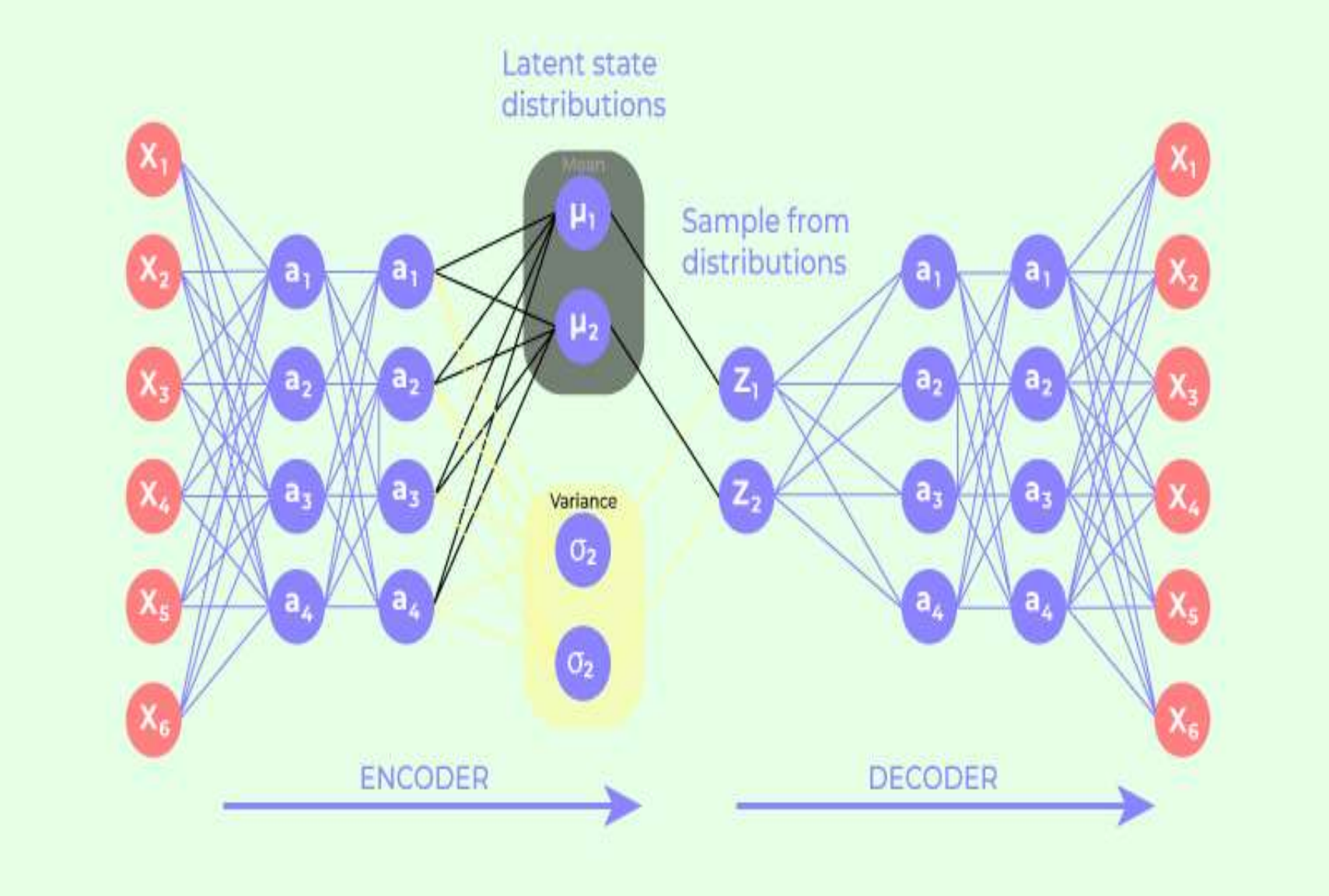
Variational Autoencoder

- A more recent type of autoencoder model is the Variational autoencoder (VAE) introduced by *Kingma and Welling* at Google introduced in the **year 2013.**
- The VAE is similar to compression and Denoising autoencoders in that they are all trained in an unsupervised manner to reconstruct inputs.
- Variational Autoencoders (VAEs) are generative models **explicitly designed** to capture the underlying probability distribution of a **given dataset** and generate novel samples.

Architecture of Variational Autoencoder

- We can fix these issues of Autoencoders, we can make two changes to the autoencoder. The result is the *"variational autoencoder."*
- However, the mechanisms that the VAEs use to perform training are quite different. In a compression/denoising autoencoder, activations are mapped to activations throughout the layers, as in a standard neural network; comparatively, a *VAE uses a probabilistic approach for the forward pass.*
- **First,** we map each point x in our dataset to a low-dimensional vector of means $\mu(x)$ and variances $\sigma(x)2$ for a diagonal multivariate Gaussian distribution.

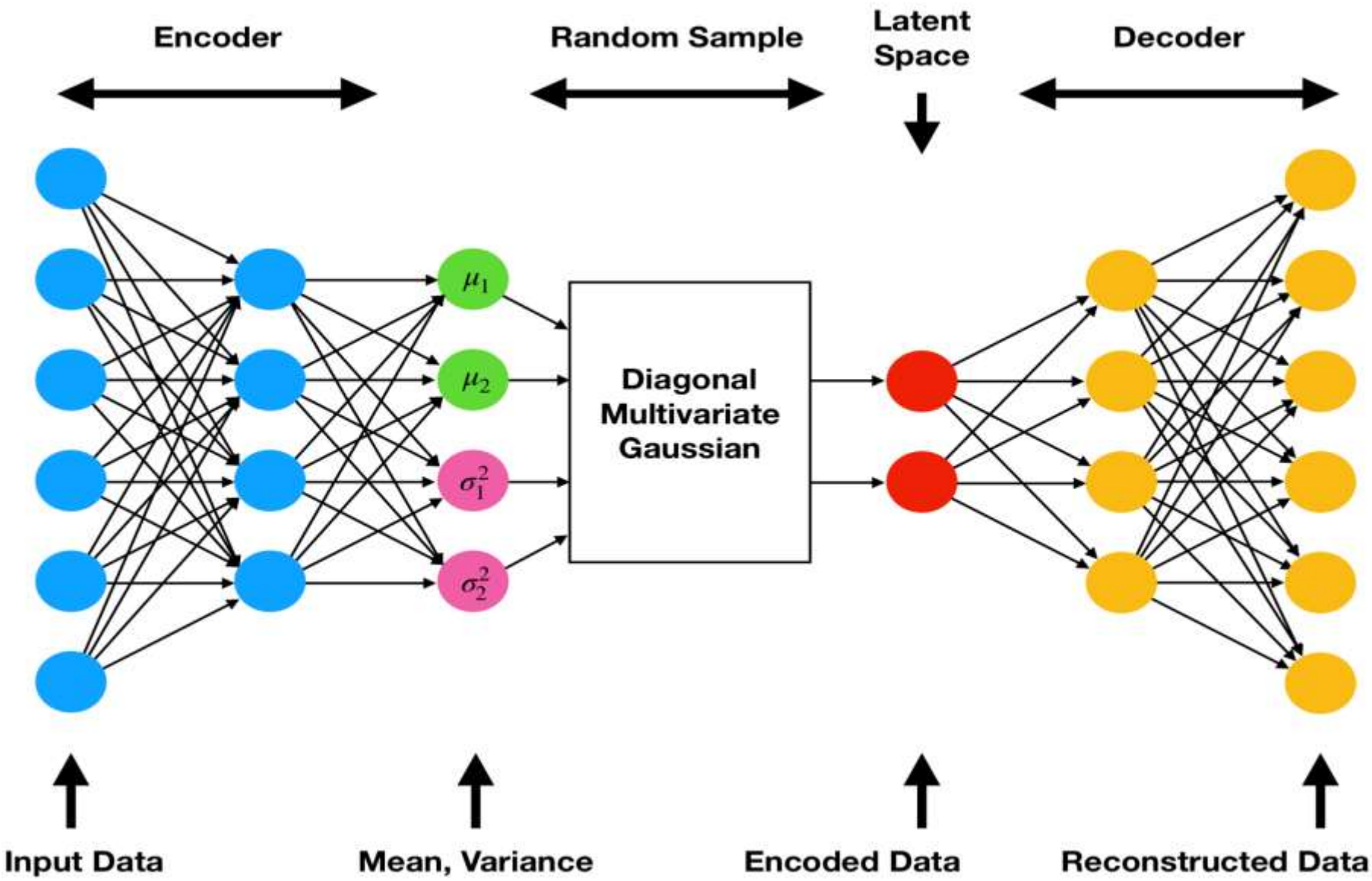
CONTD..


Latent state distributions
Mean
Sample from distributions
Variance
ENCODER
DECODER

CONTD..

- The **encoder-decoder architecture** lies at the heart of **Variational Autoencoders (VAEs),** distinguishing them from **traditional autoencoders.** The **encoder network takes raw input data and transforms it into a probability distribution within the latent space**.
- i.e, the **latent code generated by the encoder** is a **probabilistic encoding,** allowing the **VAE to express not just a single point in the latent space** but a **distribution of potential representations**.

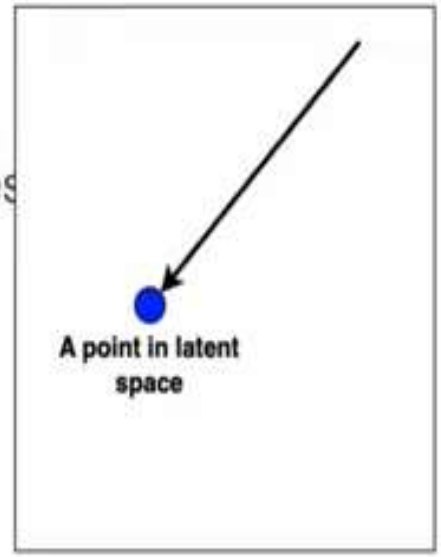
CONTD..


Encoder
Random Sample
Latent Space
Decoder
μ_1
μ_2
σ_1^2
σ_2^2
Diagonal Multivariate Gaussian
Input Data
Mean, Variance
Encoded Data
Reconstructed Data

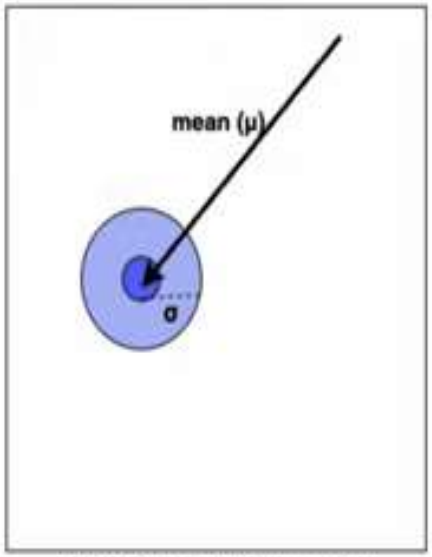
What is the use of two terms:

μ and σ?

- **Mean vector (μ)** determines the central point for encoding an input
- **Standard deviation (σ)** determines how much the encoding can deviate from that central point



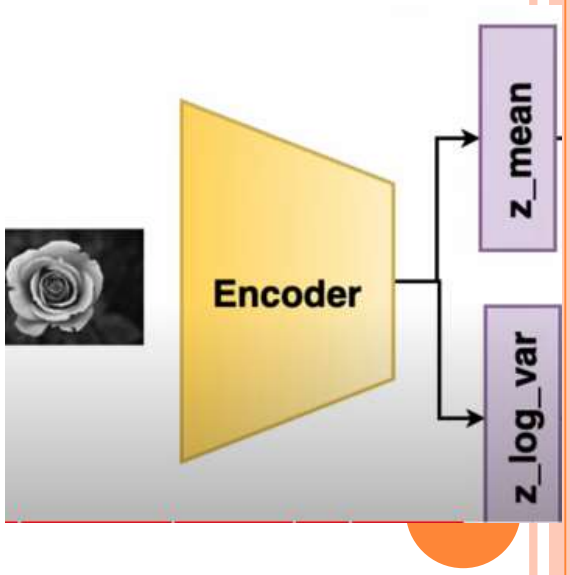
Vanilla Autoencoder



Variational Autoencoder
(μ and σ initialise a probability distribution)

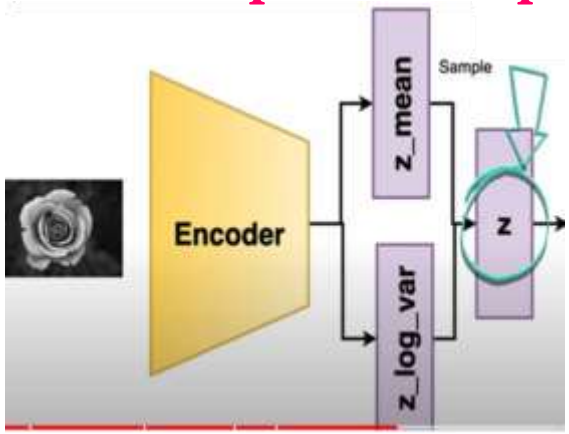
CONTD..

- First we can stats with **Encoder Part:** Here **variance values are Inherently Positive,** ie we can cover the **entire real number spectrum from (-∞ to + ∞).**
- Now this approach enables us to **employ the neural network** as the **encoder to transform input images** into **both Mean and Logarithmic Variance Vectors.**
- Here **Z-Mean** represents the **Mean Point.**
- Here **Z- log_Var** represents the **Logarithmic variance** of **each Dimension.**



CONTD..

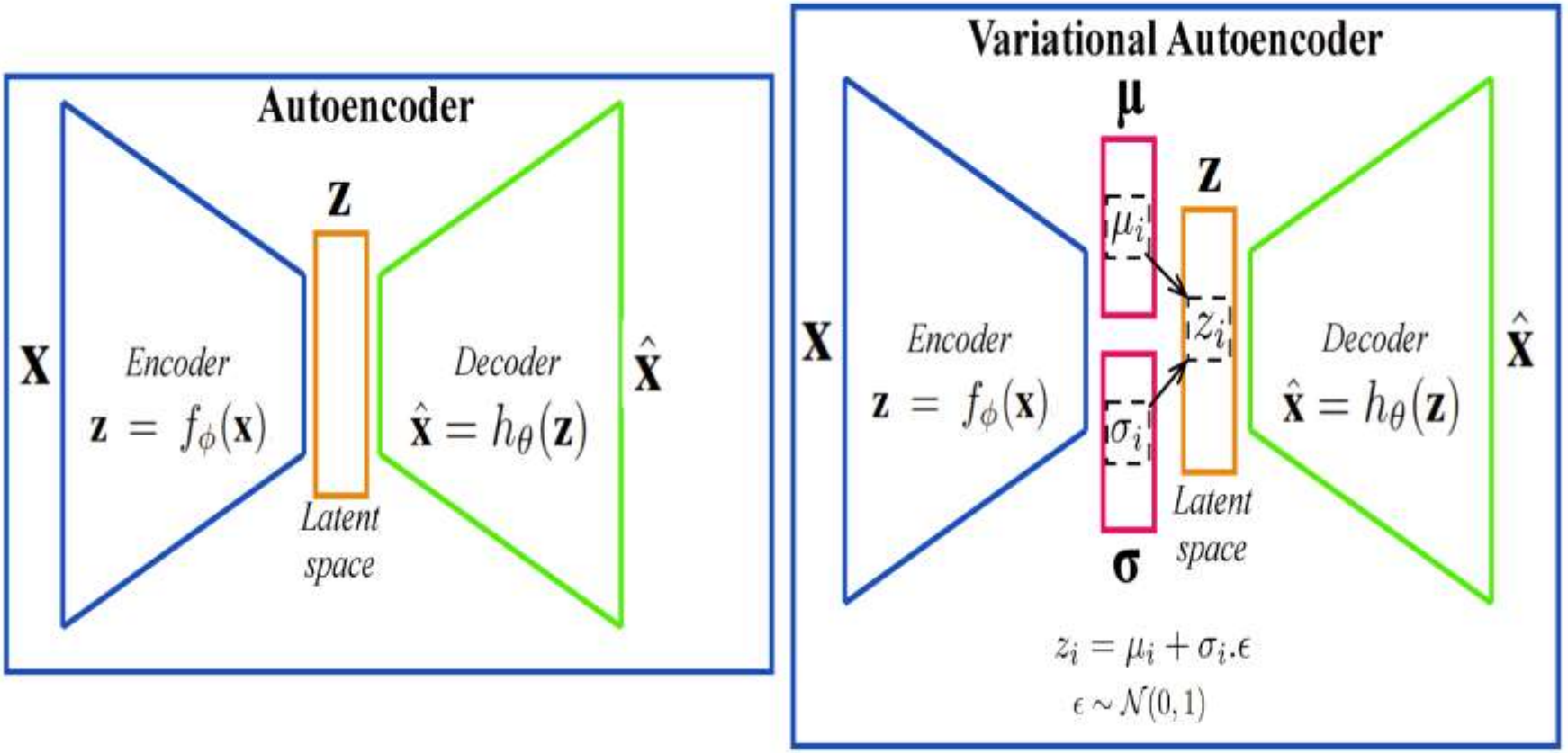
- Next, we generate Sample point Z, from the distribution defined by sample distribution. After that we can given to **Decoder.**
- **Suppose there is a loss in the expected output.**



- Variational autoencoder uses **KL-divergence as its loss function,** the **goal of this is to minimize the difference between a supposed distribution and original distribution of dataset.**

CONTD..

- Mathematically,



CONTD..

- **In reconstruction Phase**, instead of directly sampling from z_mean & Z_log_var , we use re-parameterized trick by the use of Back propagation Algorithm.
- In our example, you **approximate z using the decoder parameters** and **another parameter ε** as follows:

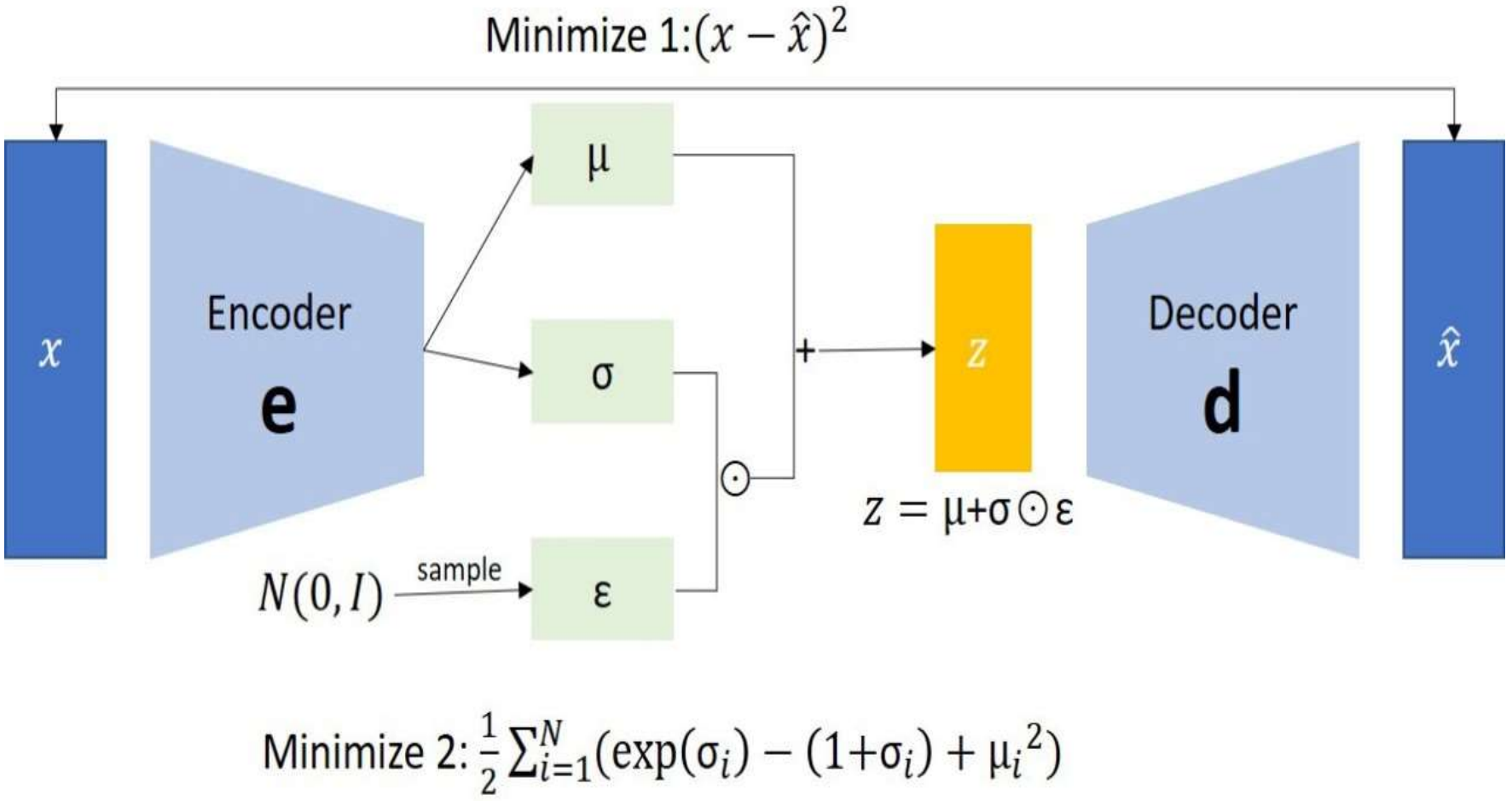
$$z_i = \mu_i + \sigma_i . \epsilon$$
$$\epsilon \sim \mathcal{N}(0,1)$$

- We start by Sampling epsilon from a standard normal distribution.
- where μ and σ represent the mean and Variance of a Gaussian distribution respectively. They can be derived from the decoder output.
- Here, we can add the Epsilon random value between (0,1).

CONTD..

- The Re parameterized Trick keeps all randomness with epsilon, ensuring deterministic gradients.
- These **deterministic gradients** are mainly used for back propagation through layer and effective neural network training.

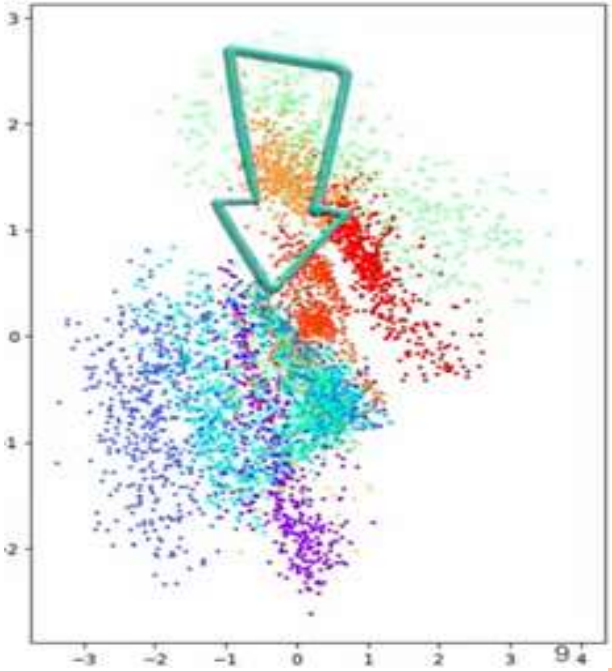
CONTD..



Minimize 2: $\frac{1}{2}\sum_{i=1}^{N}(\exp(\sigma_i) - (1+\sigma_i) + \mu_i{}^2)$

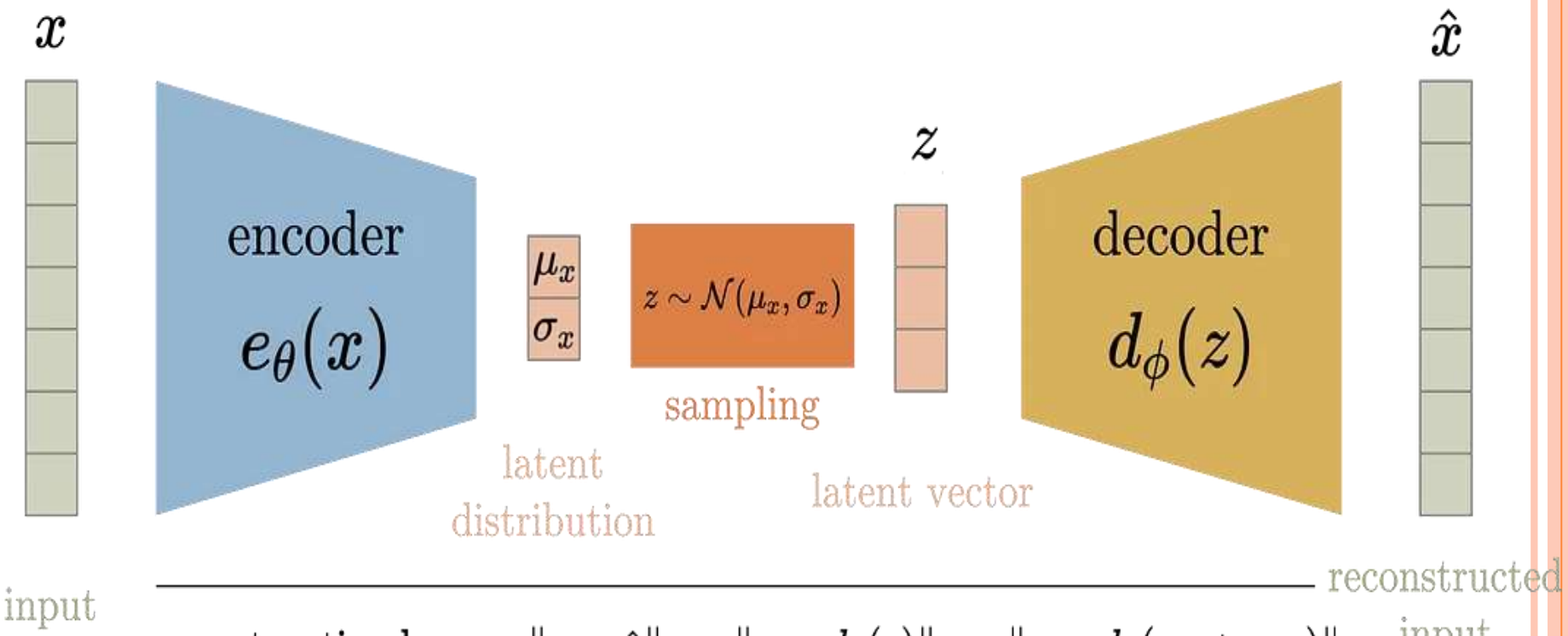
Why we need both Reconstruction and KL Loss

- When using only the **KL Loss** , the **Latent Space ends up with encodings** scattered randomly near its centers, without considering similarity among nearby encodings.
- When optimizing both the KL Loss and another Loss together, it creates a latent space where nearby encodings are similar, forming clusters on a local scale.



CONTD..

- Mathematically,



$$\text{reconstruction loss} = \|x - \hat{x}\|_2 = \|x - d_\phi(z)\|_2 = \|x - d_\phi(\mu_x + \sigma_x \epsilon)\|_2$$

$$\mu_x, \sigma_x = e_\theta(x), \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

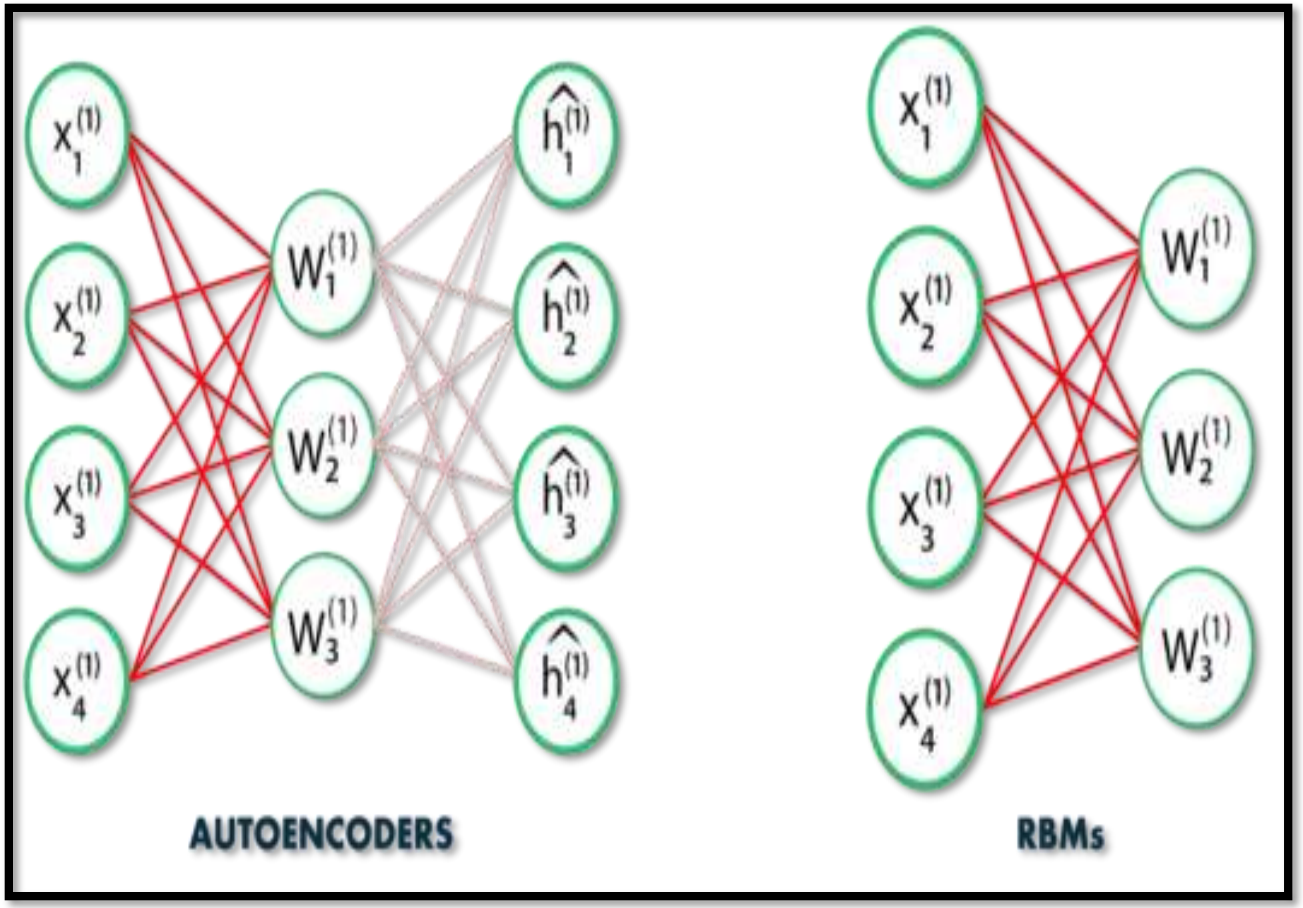
$$similarity\ loss = KL\ Divergence = D_{KL}(\mathcal{N}(\mu_x, \sigma_x) \parallel \mathcal{N}(\mathbf{0}, \mathbf{I}))$$

$$loss = reconstruction\ loss + similarity\ loss$$

Difference between RBN and Autoencoder

- Autoencoder is a simple 3-layer neural network where output units are directly connected back to input units. Typically, the number of hidden units is much less than the number of visible ones.
- The task of training is to minimize an error or reconstruction, i.e. find the most efficient compact representation for input data.
- Restricted Boltzmann Machines are, two-layer neural nets that constitute the building blocks of deep-belief networks. The first layer of the RBM is called the visible, or input layer, and the second is the hidden layer.

CONTD..



Difference between Autoencoder & VAE

Autoencoders and Variational Autoencoders

Similarities & Differences

Similarities	Differences
Both AE& VAE are **Neural Network Architectures** that are used for **Unsupervised Learning**	AE uses **Deterministic Mapping** to represent the Input Data, while **VAE uses Probabilistic Mapping**
Both AE& VAE consists of an **encoder and decoder network.** The Encoder maps the input data to a latent representation, and the decoder maps the latent representation back to the original data	VAE has **additional Loss term** that is used to regularize the **latent space called KL Divergence Loss.** This loss uses the **Gaussian Distribution.**
Both AE and VAE are used for **Dimensionality Reduction, Data Generation and Anomaly Detection**	VAE can be used to **generate new data samples by sampling** from the **latent space,** where AE cannot.

Difference between Autoencoder & VAE

Autoencoder	Variational Autoencoder
Used to **generate a compressed transformation** of **input in a latent space**	Enforces conditions on the latent variable to be the **unit norm**
The latent variable is **not regularized**	The latent variable in the **compressed form** is **mean and variance**
Picking a random latent variable will generate **garbage output**	A random value of latent variable generates **meaningful output at the decoder**
Latent variable has **deterministic values**	The input of the decoder is **stochastic and is sampled from a Gaussian with mean and variance** of the output of the encoder.
The **latent space lacks the generative capability**	The **latent space has generative capabilities**.

Thank You

UNIT-II

PART – II

MAJOR ARCHITECTURES OF DEEP NETWORKS

Unsupervised Pretrained Networks

Topics :

- Generative Adversarial Network
  - Architecture of GAN
  - Mathematical Notation
  - Loss Function GAN
  - Training Process of GAN
  - Types of GAN
  - Applications of GAN
  - Future generations of GAN
  - Differences b/w VAN and GAN

Unsupervised Pretrained Networks

- **Unsupervised pre-training** is a **Deep Learning technique** that leverages unlabeled data to learn a preliminary model, which can then be fine-tuned with a smaller amount of labeled data. This approach is particularly beneficial in scenarios where labeled data is scarce or expensive to obtain.
- **Unsupervised pre-training is a technique** involves **unlabeled data to learn features and data distribution:**

CONTD..

- 1. Pre-training : The model is trained on a large amount of unlabeled data using **unsupervised learning algorithms** like generative models or autoencoders. This phase helps the model learn the underlying data distribution and extract useful features.
- 2. Fine-tuning : The pre-trained model is then further trained on a smaller set of labeled data. The model adjusts its parameters to **better fit the specific task at hand.**

CONTD..

It also mainly follows Two Hypothesis in unsupervised Pretrained Networks:

1. Better optimization: Unsupervised pretraining puts the network in a region of parameter space run deeper than when starting with random parameters. In simple words, the network starts near a global minimum. In contrast to a local minimum, a global minimum means a lower training error.

2. Better regularization: Unsupervised pretraining puts the network in a region of parameter space in which training error is not necessarily better than when starting at random (or possibly worse)

CONTD..

but which systematically yields better generalization (lower test error).

Such behavior would be **indicative of a regularization effect.**

Based on the Unsupervised Pretrained Networks we can follow the GAN and Auto Encoders Algorithms.

GENERATIVE ADVERSARIAL NETWORKS

- To understand the term GAN let's break it into separate three separate Words. and each of them has its **separate meaning**, which is as follows:

1. Generative – To learn a generative model, which describes how data is generated in terms of a probabilistic model. In simple words, it explains how data is generated visually.

2. Adversarial – The training of the model is done in an adversarial setting. i.e, The word adversarial refers to the context of GANs, the generative result is compared with the actual images in the data set.

CONTD..

- **This mechanism known** as **a discriminator** and this is **used to apply a model that attempts to distinguish between real and fake images.**

3. Networks – Use **deep neural networks** as artificial intelligence (AI) **algorithms for training purposes.**

GANs are a type of neural network architecture that can **generate new data based on the patterns** learned from a **given dataset**.

This means that GANs can create **entirely new, realistic images, videos, and even audio clips** that have **never existed before.**

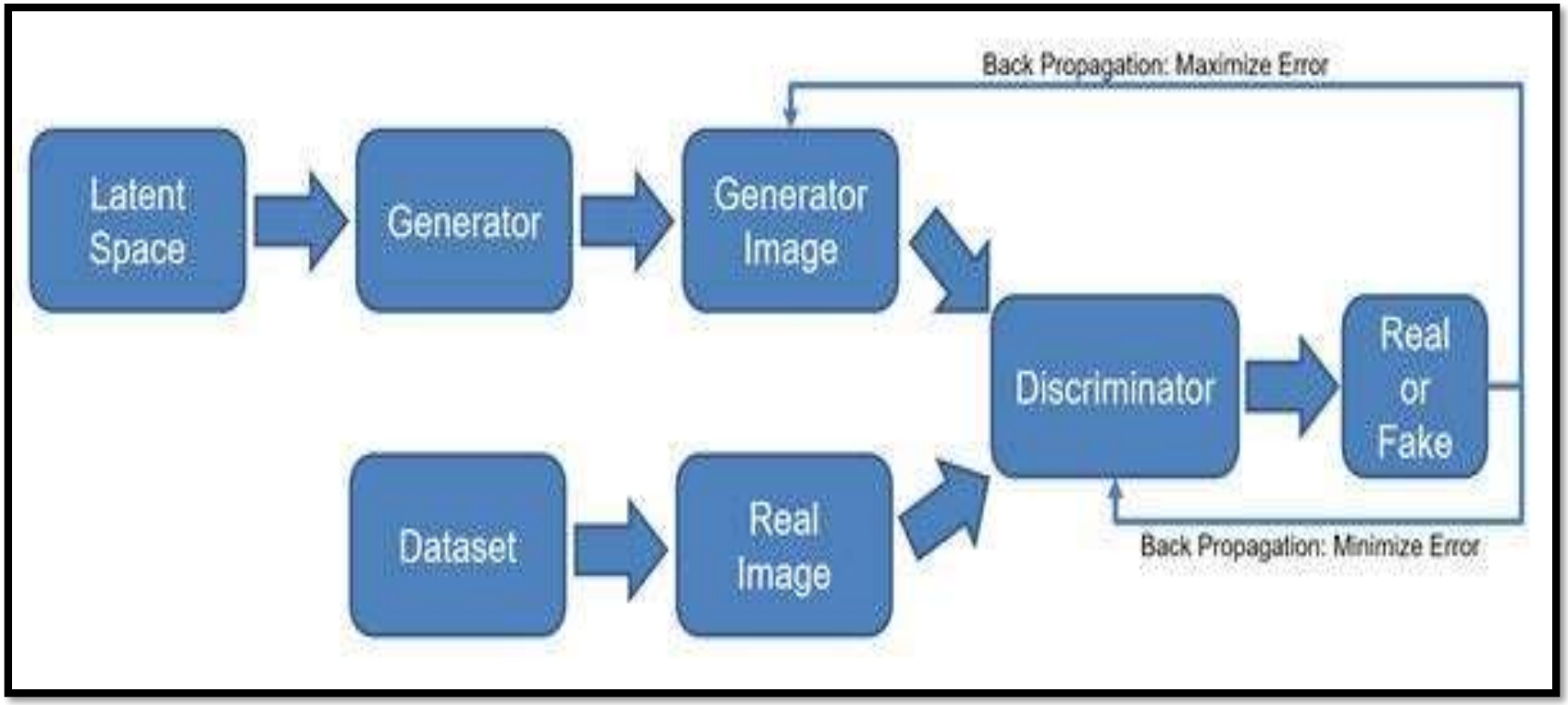
Architecture of GAN

- Generative Adversarial Networks (GANs) are a groundbreaking innovation in the field of Deep Learning, particularly within the domain of unsupervised learning.
- GANs are made up of two neural networks,
  - 1. Generator and
  - 2. Discriminator
- These **2 models** that automatically discover and learn the patterns in input data.

CONTD..

- They comprise two networks:
- The generator, which produces synthetic data. i.e, information that is artificially generated rather than produced by real-world events and
- The discriminator, which differentiates between real and generated data. This unique structure enables GANs to generate highly realistic and diverse outputs, from images to text.

CONTD..

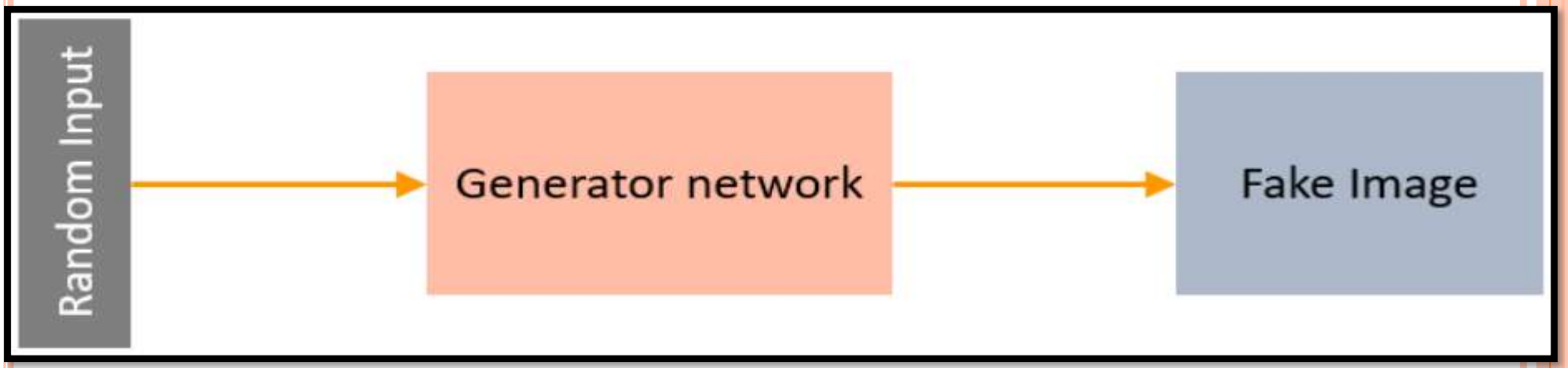


The GAN Network Process

CONTD..

The Generator: The generator network is responsible for generating new data that is similar to the training data. The generator network takes random noise as input and produces a generated output. The goal is to train the generator to produce outputs that are as close to the real data as possible.

- A Generator in GANs is a neural network that creates fake data to be trained on the discriminator.



CONTD..

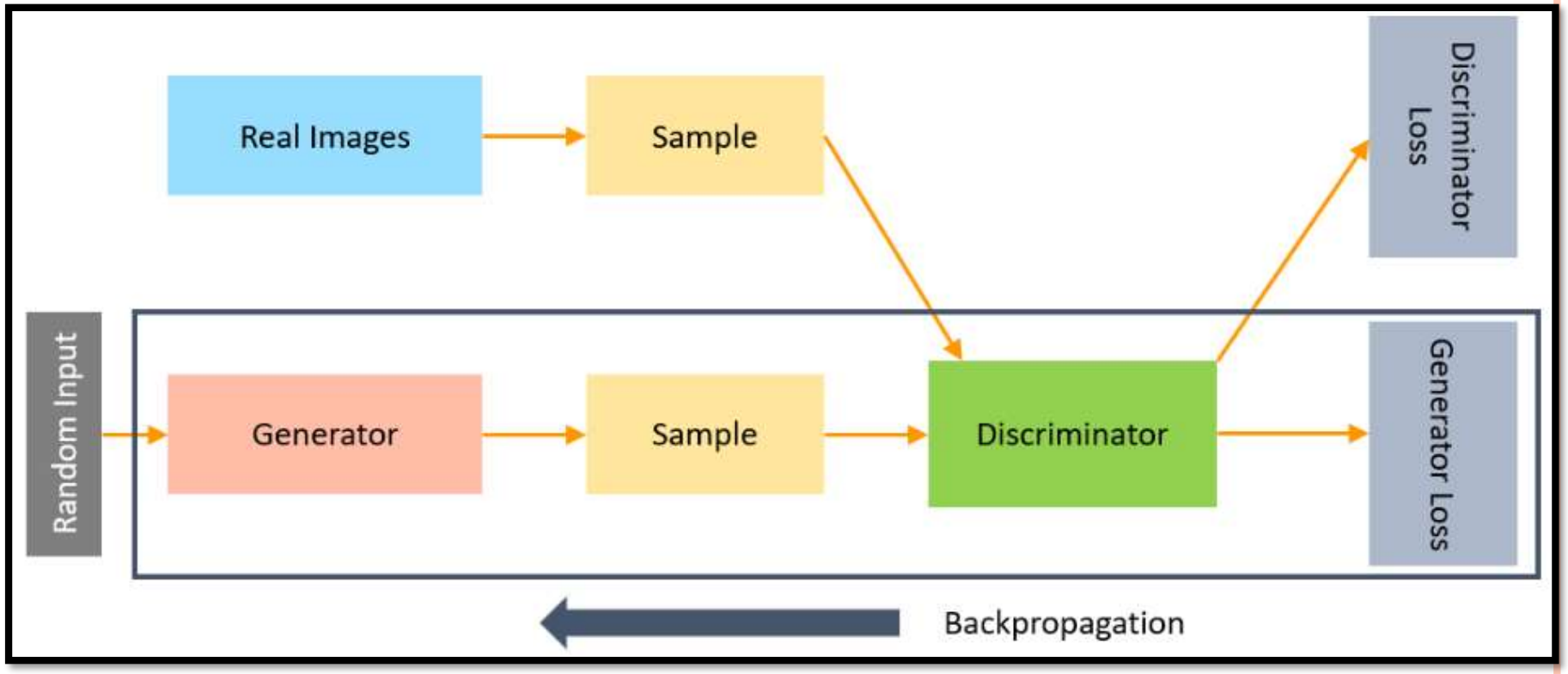
- The main aim of the Generator is to make the discriminator classify its output as real. The part of the GAN that trains the Generator includes:
  - Provide fake input or noise and get random noise to produce output based on the noise sample.
  - Predict generator output either real or fake using discriminator.
  - Calculate discriminator loss and perform back propagation.
  - Calculate gradients to update the weights of the generator.

CONTD..

- Back propagation of Generator: The generator modifies some data attributes by **adding noise (or random changes)** to certain attributes
- The generator passes the **modified data** to the discriminator
- The discriminator calculates the probability that the generated output belongs to the original dataset
- The discriminator gives some guidance to the generator to reduce the noise vector randomization in the next cycle
- The generator attempts to maximize the probability of mistake by the **discriminator**, but the discriminator attempts to minimize the probability of error.

CONTD..

- In training iterations, both the generator and discriminator iterating continuously until they reach an equilibrium state. In the equilibrium state, the discriminator can no longer recognize synthesized data. At this point, the training process is over.



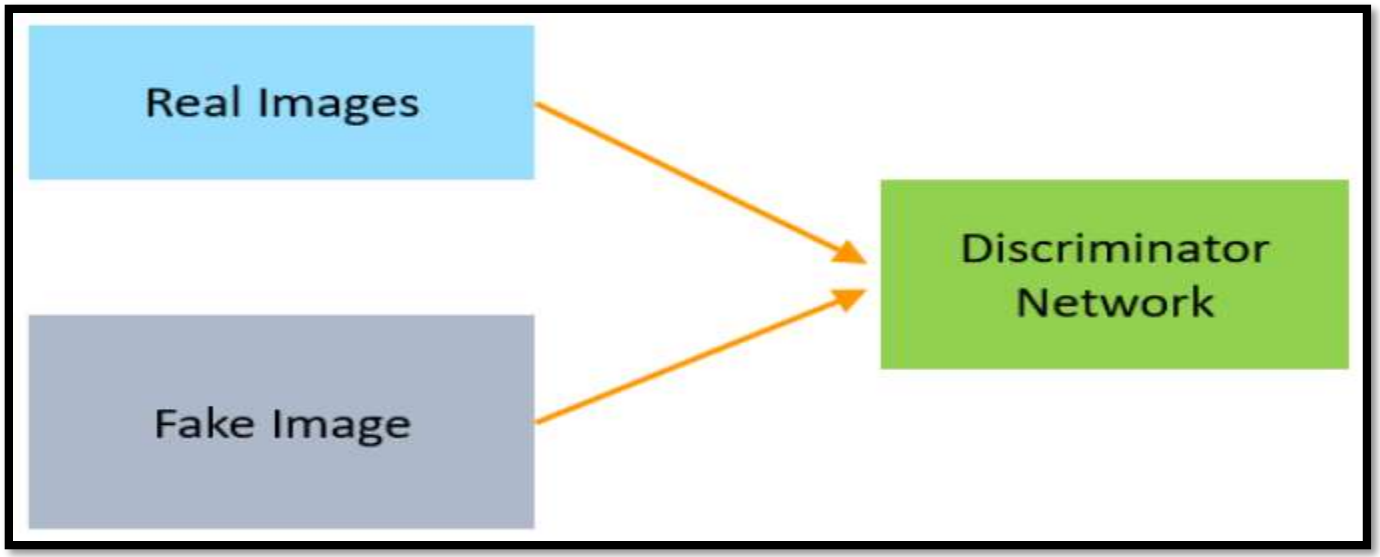
CONTD..

The Discriminator: In a GAN, the discriminator acts as a binary classifier whose main task is to differentiate between real data and data generated by the GAN's generator. The choice of network architecture for the discriminator depends on the type of data being classified.

- The Discriminator is a neural network that identifies real data from the fake data created by the Generator. The discriminator's training data comes from different two sources:
- The real data instances, such as real pictures of birds, humans, currency notes, etc., are used by the Discriminator as positive samples during training.

CONTD..

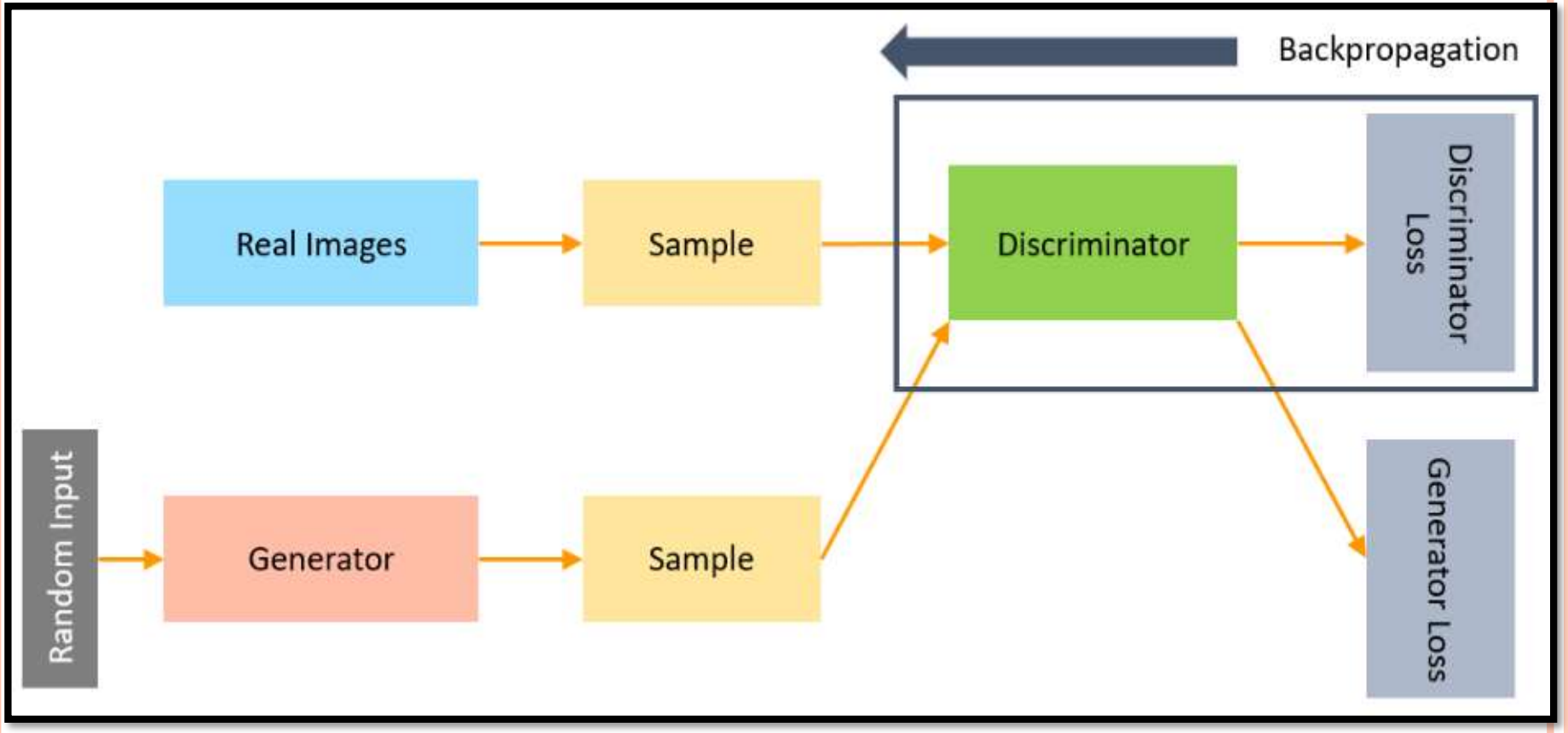
- The fake data instances created by the Generator are used as negative examples during the training process.



While training the discriminator, it connects to two loss functions. During discriminator training, the discriminator ignores the generator loss and just uses the discriminator loss.

CONTD..

- Next, The discriminator updates its weights through backpropagation from the discriminator loss through the discriminator network.



Mathematical notation

- We are basically training the Discriminator to maximize the probability of assigning correct labels to both real and generated data.

  We are also training the Generator to minimize the probability to get caught by the Discriminator, which is equivalent to minimizing log(1-D(G(z))).

- here the Discriminator is trying to minimize its reward V(D, G) and the Generator is trying to minimize the Discriminator's reward or in other words, maximize its loss.

MATHEMATICAL NOTATION

$$\min_G \max_D V(D,G)$$

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

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

x : Real data

z : Latent vector

$G(z)$: Fake data

$D(x)$: Discriminator's evaluation of real data

$D(G(z))$: Discriminator's evaluation of fake data

Loss Functions in GAN

- The training process for GANs involves minimizing a loss function that **quantifies** the difference between the generated and real data.
- There are two types of Loss Functions:
  - 1. Generator Loss
  - 2. Discriminator Loss

1. Generator Loss: The **objective of the generator in a GAN** is to produce synthetic samples that are realistic **enough to fool the discriminator**. The generator achieves this by minimizing its loss function J_G.

CONTD..

- The loss is minimized when the log probability is maximized, i.e., when the discriminator is highly likely to classify the generated samples as real. The following equation is given below:

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

The generator aims to minimize this loss, encouraging the production of samples that the **discriminator classifies as real** (logD(G(zi)), close to 1.

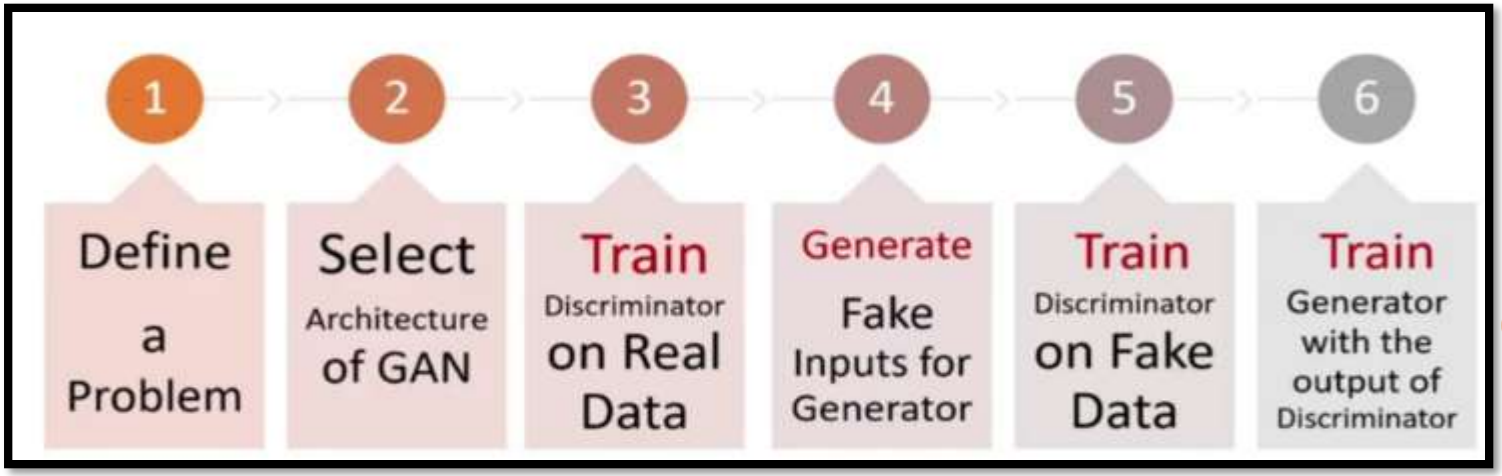
CONTD..

2. Discriminator Loss: The discriminator **reduces t**he **negative log likelihood** of correctly classifying both produced and real samples.

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

Training process of GAN

- Step 1: Define a Problem.
- Step 2: Select Architecture of GAN.
- Step 3: Train Discriminator on Real Dataset.
- Step 4: Train Generator.
- Step 5: Train Discriminator on Fake Data.
- Step 6: Train Generator with the output of Discriminator.



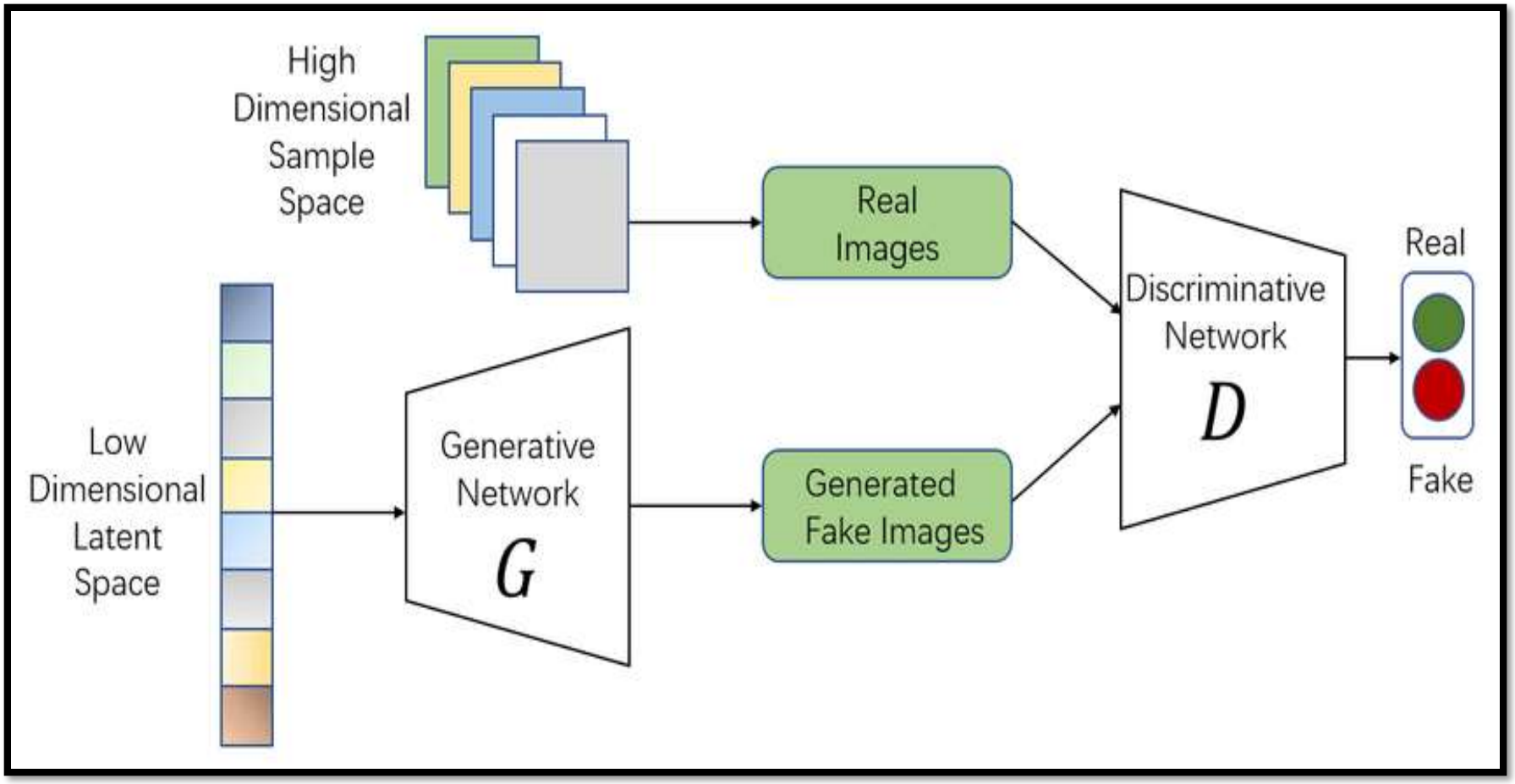
Types of GAN


Vanilla GAN
01
02
Conditional GAN (CGAN)
Deep Convolutional GAN (DCGAN)
03
04
CycleGAN
Generative Adversarial Text to Image Synthesis
05
06
StyleGAN
Super Resolution GAN (SRGAN)
07

CONTD..

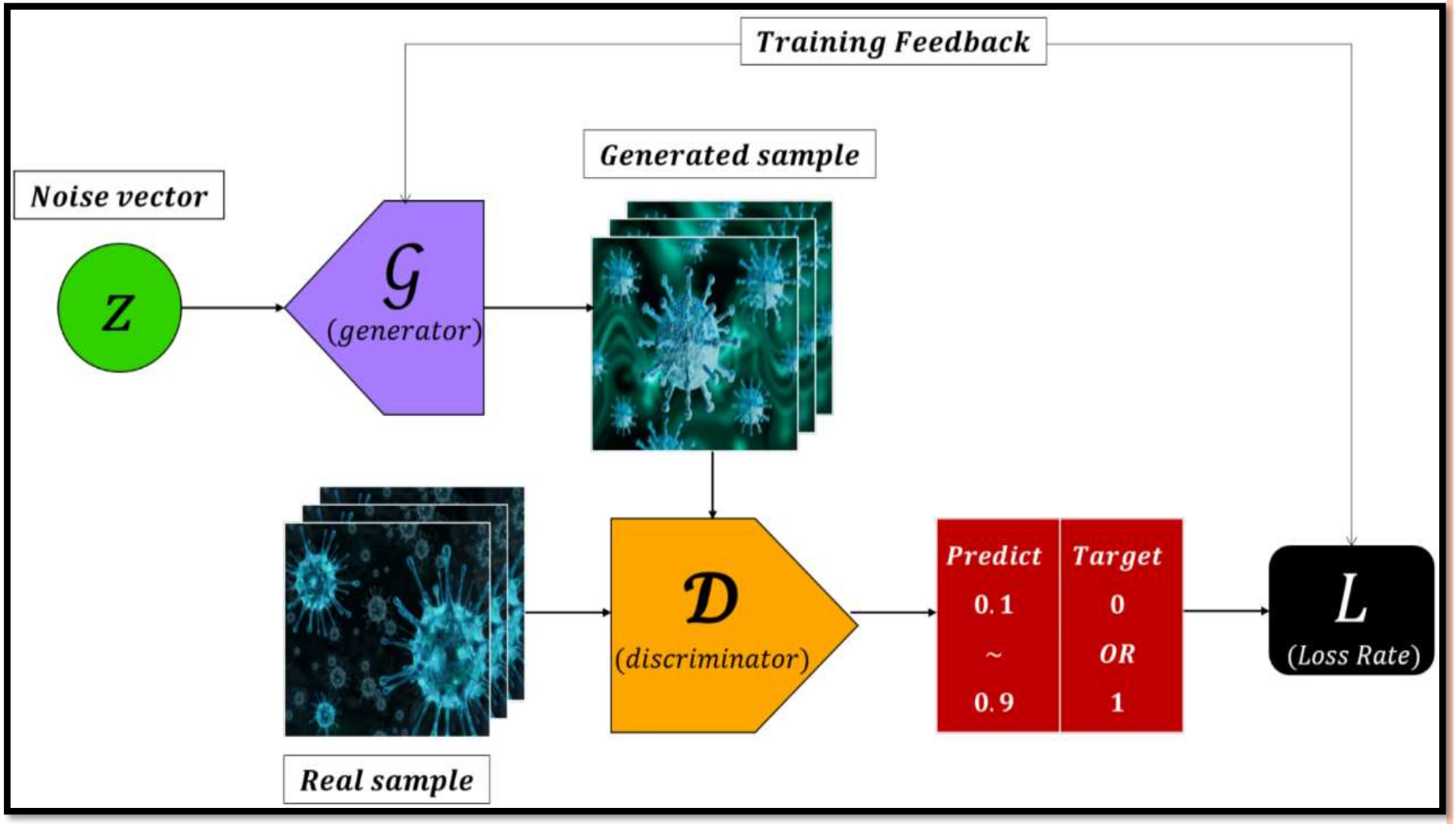
- 1. Vanilla GAN: This is the most basic form of GAN, where both the generator and discriminator are modeled using multi-layer perceptrons.
- The generator focuses on capturing data distribution, while the discriminator evaluates the probability that a given sample is real or synthetic.
- This Vanilla GAN always **tries to optimize** the **mathematical equation** using **stochastic gradient descent.**

CONTD..



ARCHITECTURE OF VANILLA GAN

CONTD..

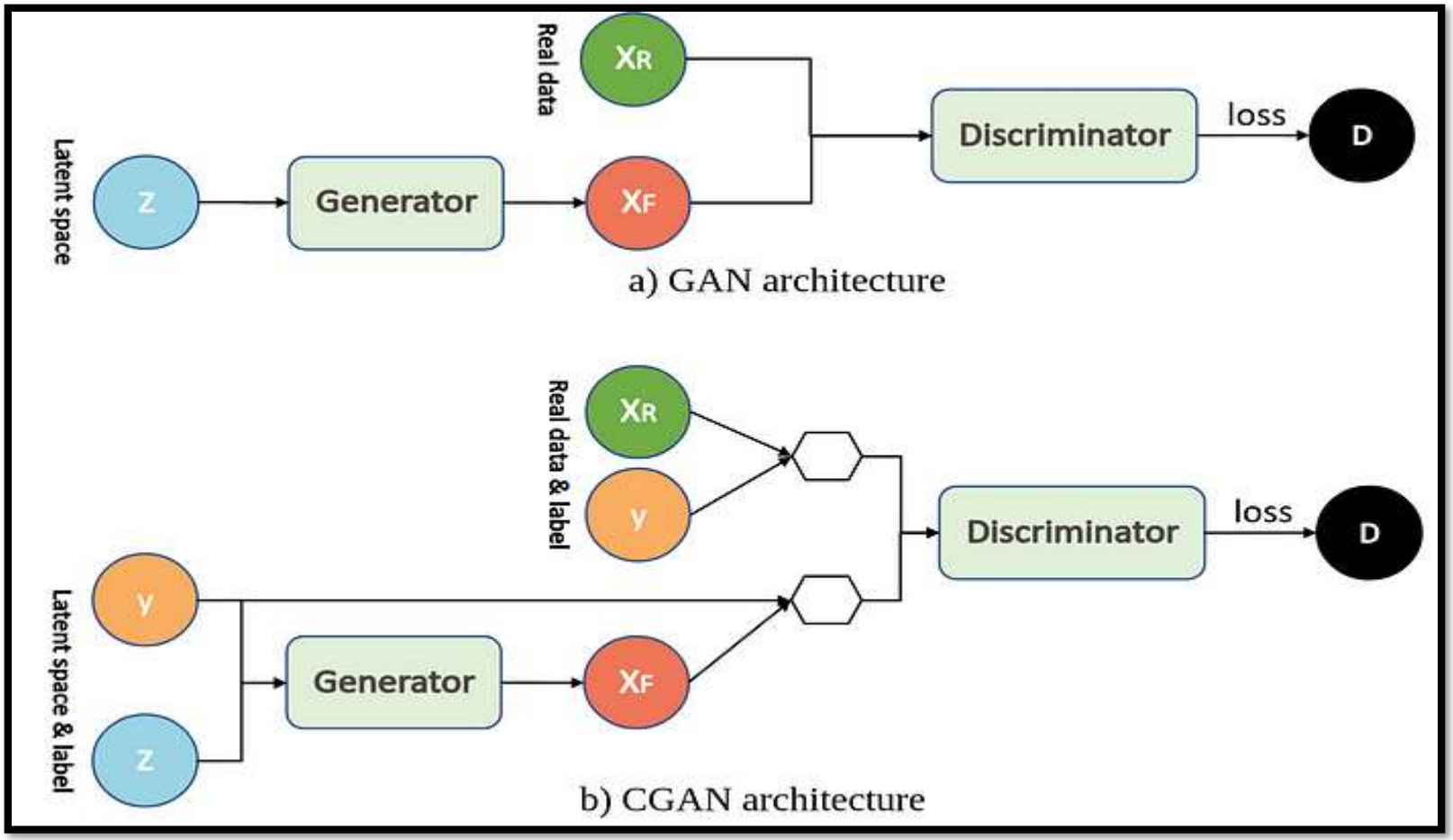


EXAMPLE OF VANILLA GAN

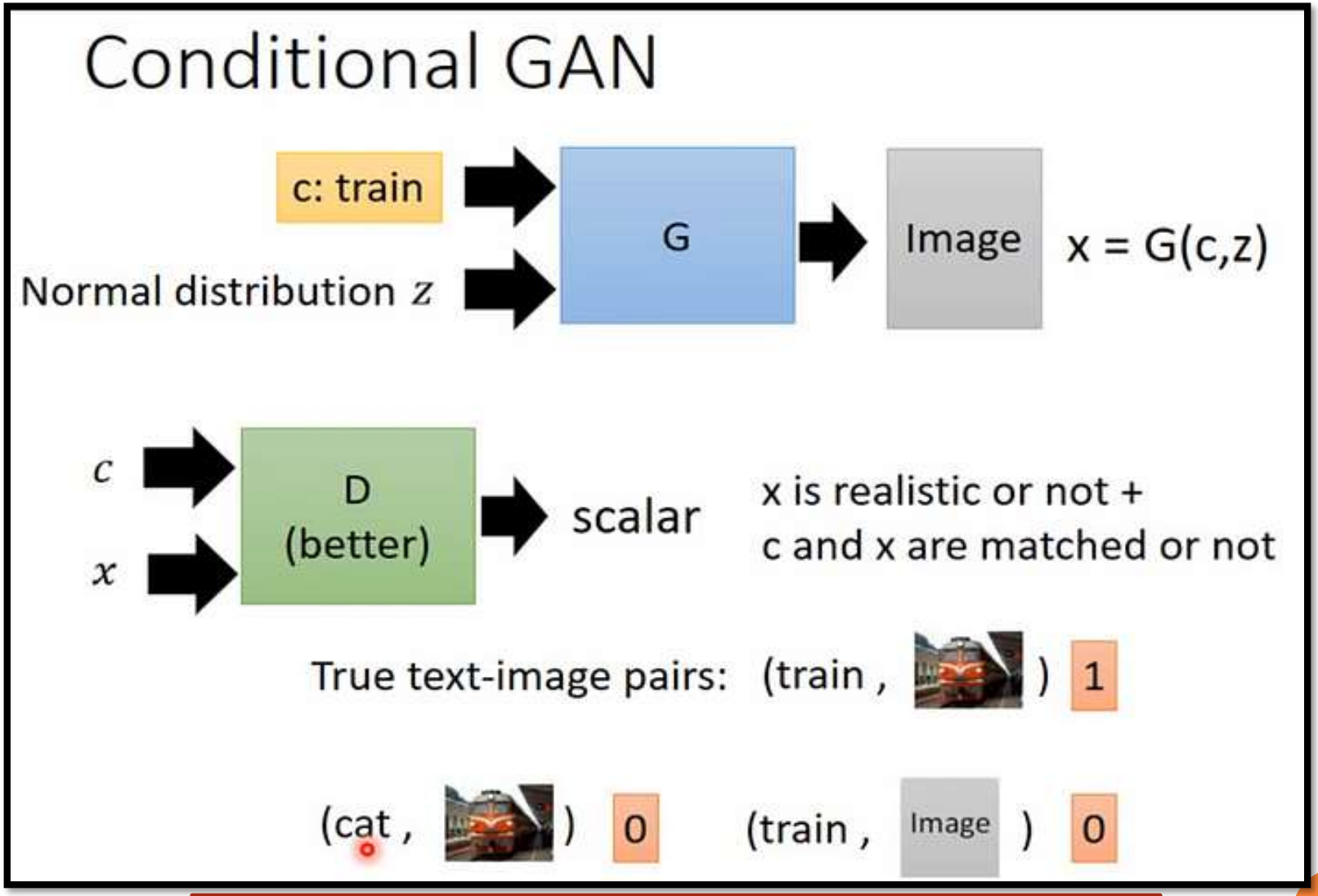
CONTD..

- **2. Conditional GAN:** CGAN can be described as a deep learning method in which some conditional parameters are put into place.
- Here, both networks receive additional information such as class labels, making the model conditional. This allows the generation of data that is more specific to the given condition.
- In CGAN, an additional parameter 'y' is added to the Generator for generating the corresponding data.
- Labels are also put into the input to the Discriminator in order for the Discriminator to help distinguish the real data from the fake generated data.

CONTD..



Architecture of Conditional GAN (CGAN)

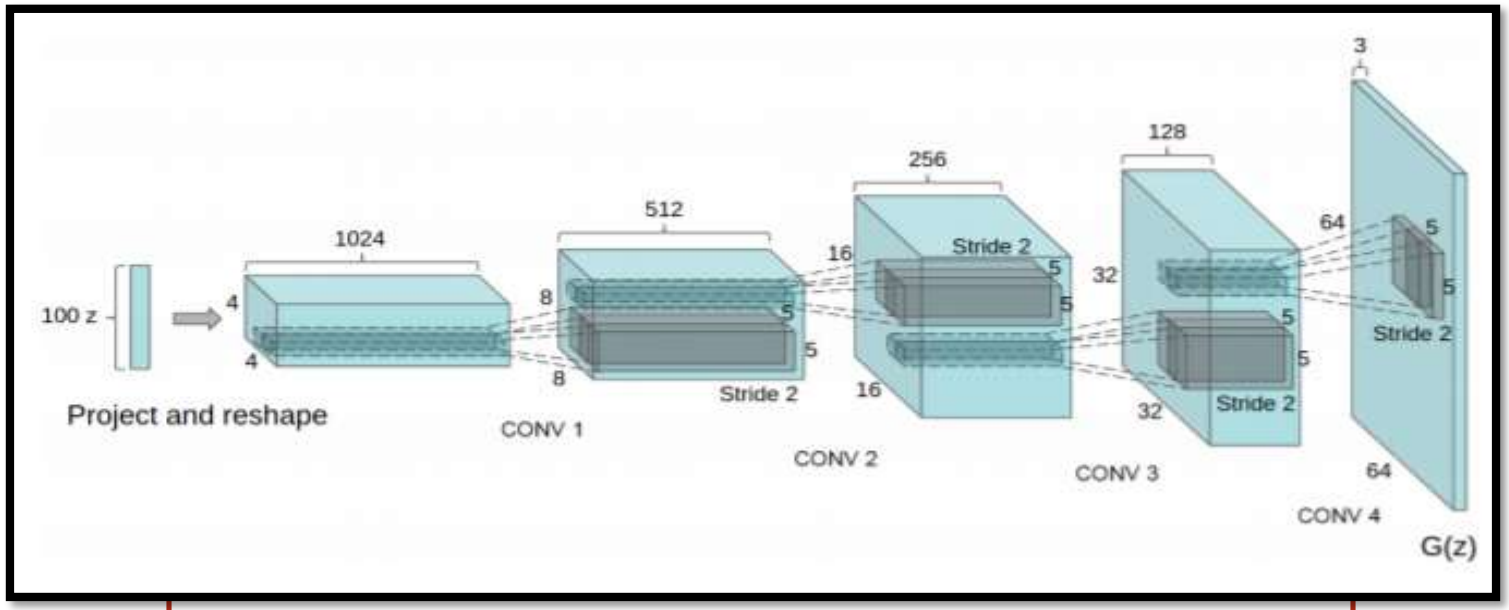


Example of Conditional GAN (CGAN)

CONTD..

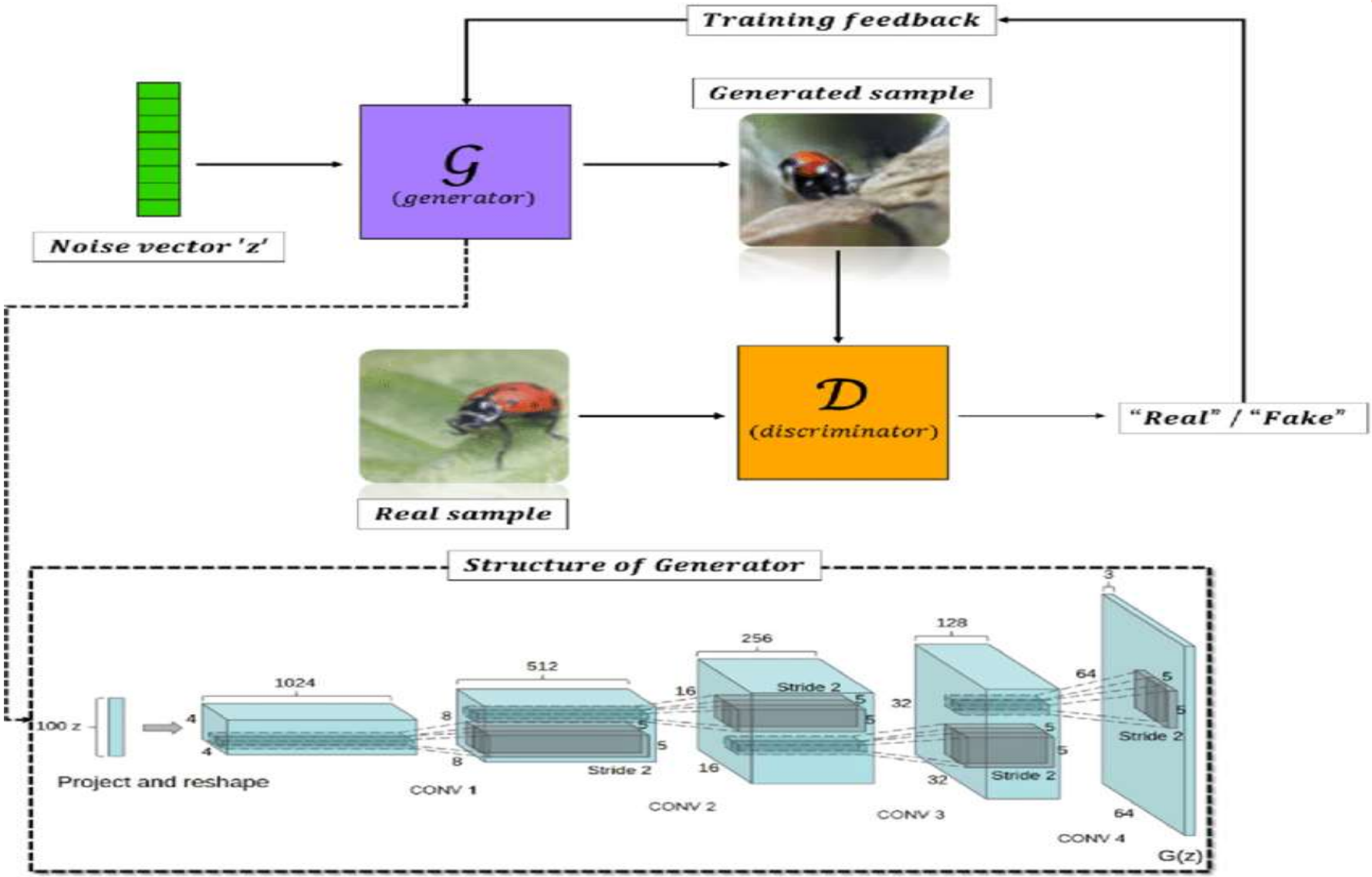
- 3. Deep Convolutional GAN (DCGAN): DCGANs employ convolutional neural networks, making them more effective for tasks that involve image data. They are known for generating high-quality, high-resolution images.
- Deep Convolutional GAN (DCGAN) was proposed by a researcher from MIT and Facebook AI research. It is widely used in many convolution-based generation-based techniques.
- It is composed of ConvNets in place of multi-layer perceptrons.
- DCGAN uses convolutional and convolutional-transpose layers in the generator and discriminator, respectively. It was proposed by Radford.

- Here the discriminator consists of strided convolution layers, and Relu as activation function.. The generator consists of convolutional-transpose layers, and ReLU activations. The output will be a 3x64x64 RGB image.

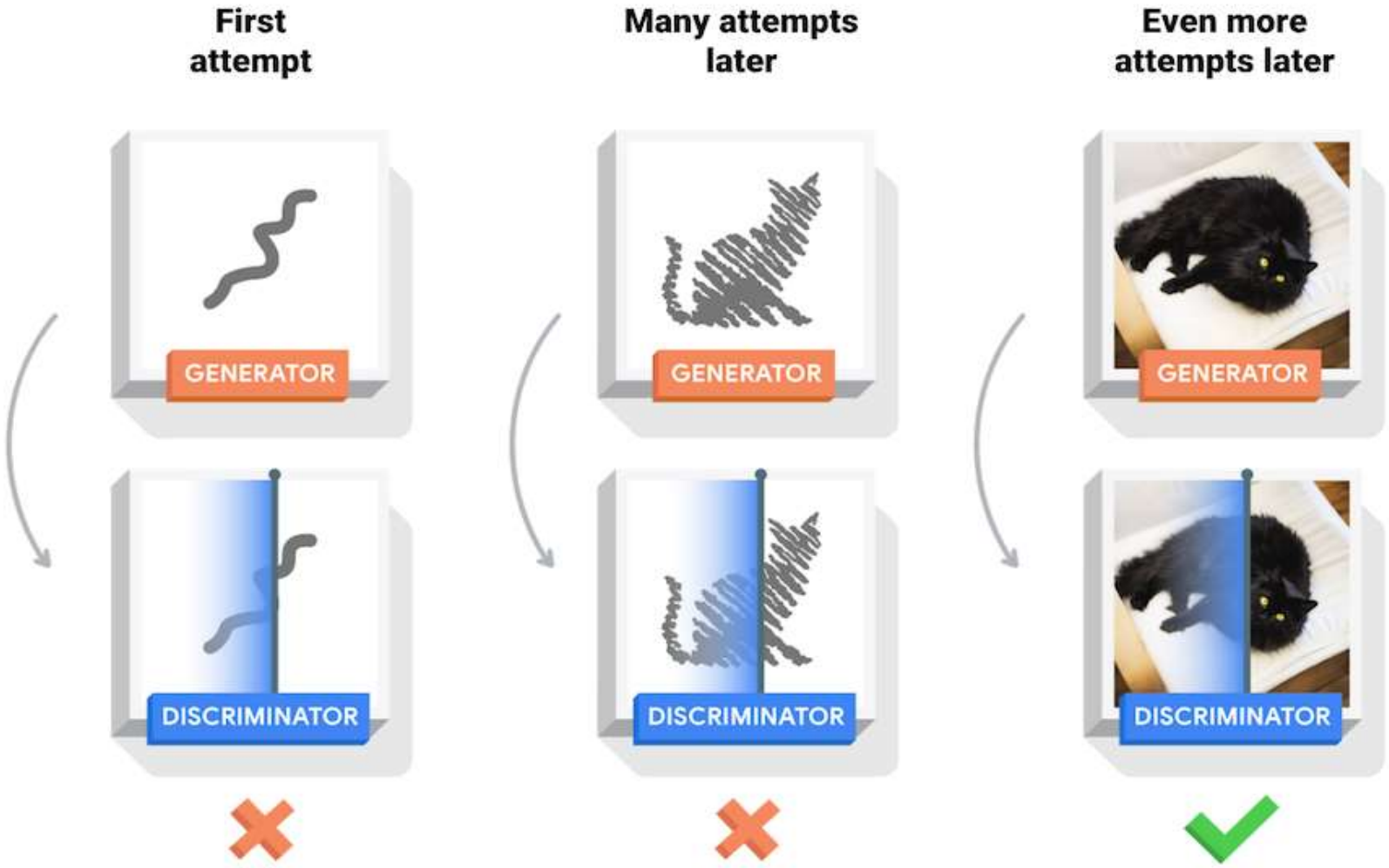


Architecture of Deep Convolution GAN (DCGAN)

CONTD..



Example of DCGAN (DCGAN)



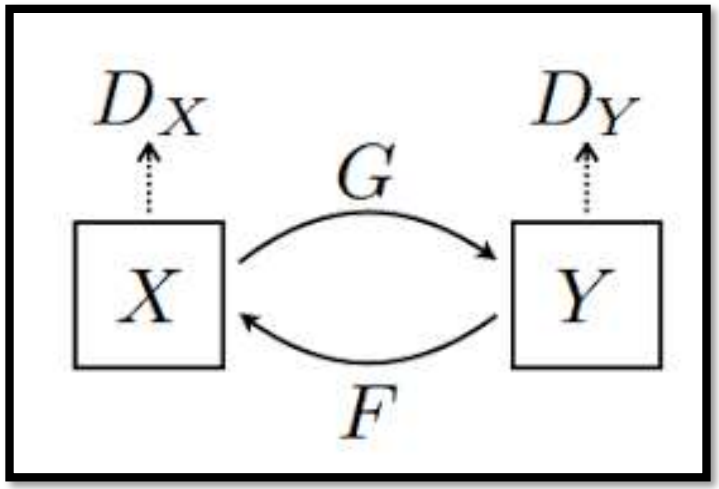
Example of DCGAN (DCGAN)

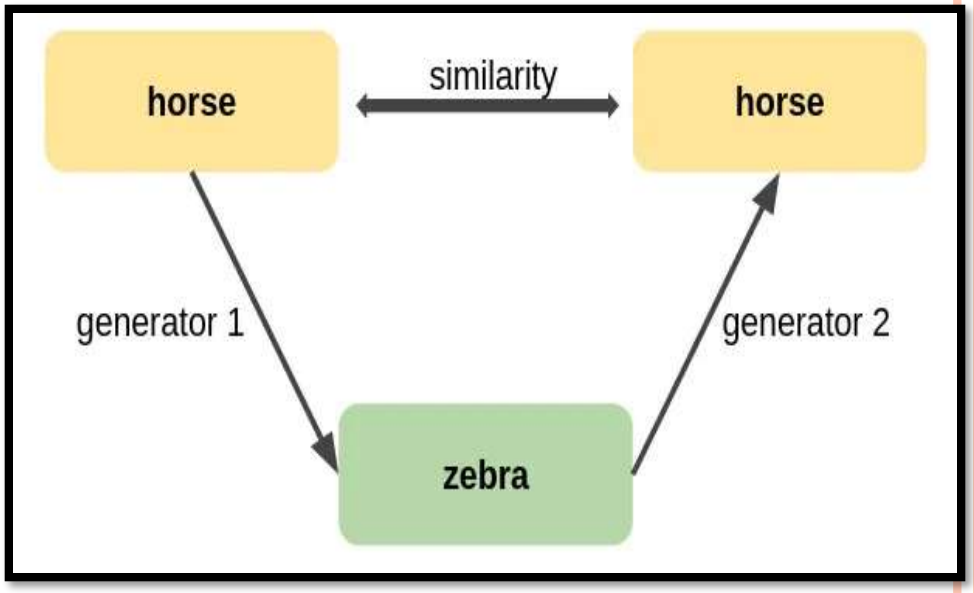
CONTD..

- 4. Cycle GAN: **Cycle GAN** is used to transfer characteristic of one image to another or can map the distribution of images to another.
- In **CycleGAN** we treat the problem as an image reconstruction problem. We first take an image input (x) and using the generator G to convert into the reconstructed image.
- Then we reverse this process from **reconstructed image to original image using a** generator F.
- Then we calculate the mean squared error loss between real and reconstructed image.

CONTD..

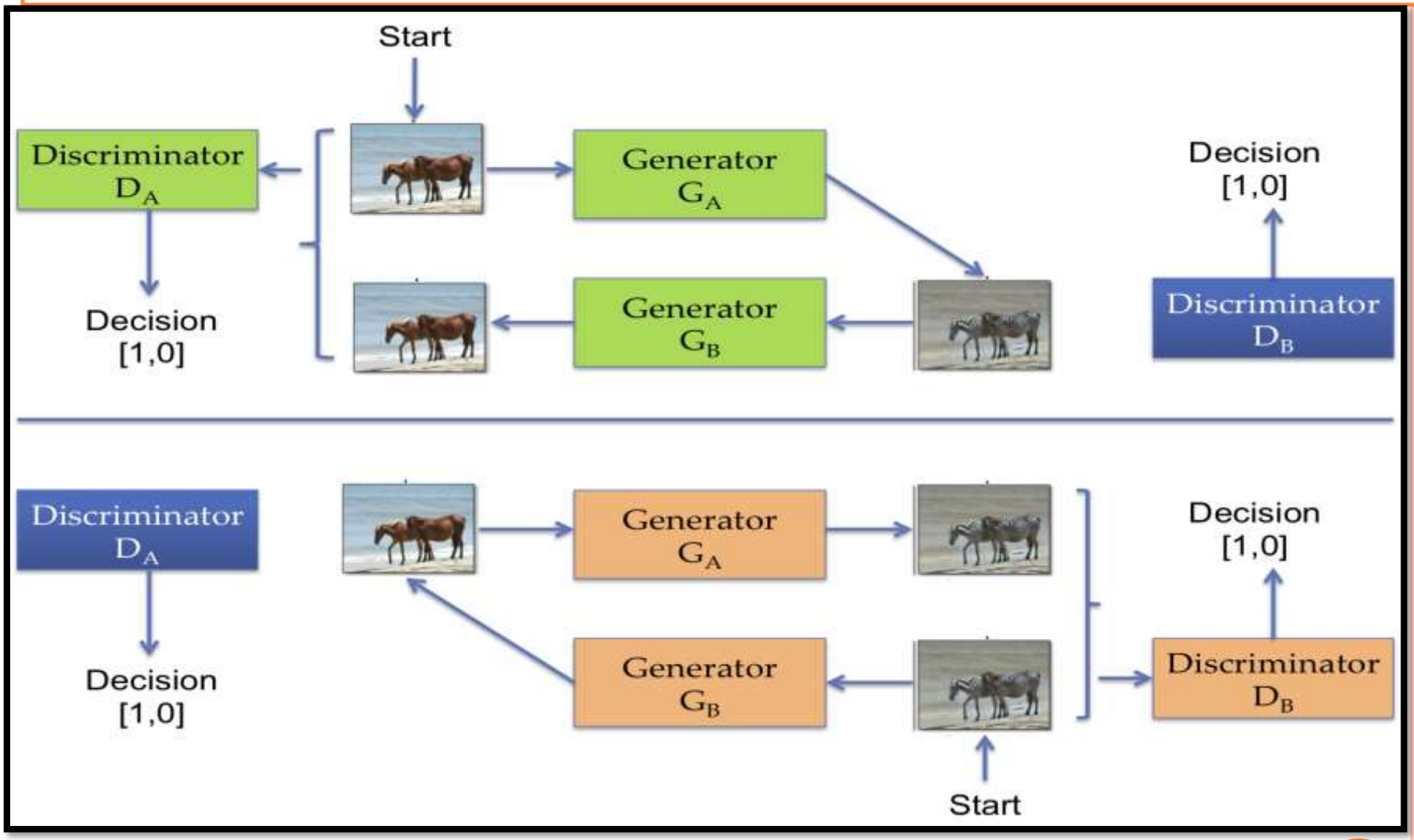
- The most important feature of this cycle_GAN is that it can do this image translation on an unpaired image where there is no relation exists between the input image and output image.





Architecture of Cycle GAN (CGAN)

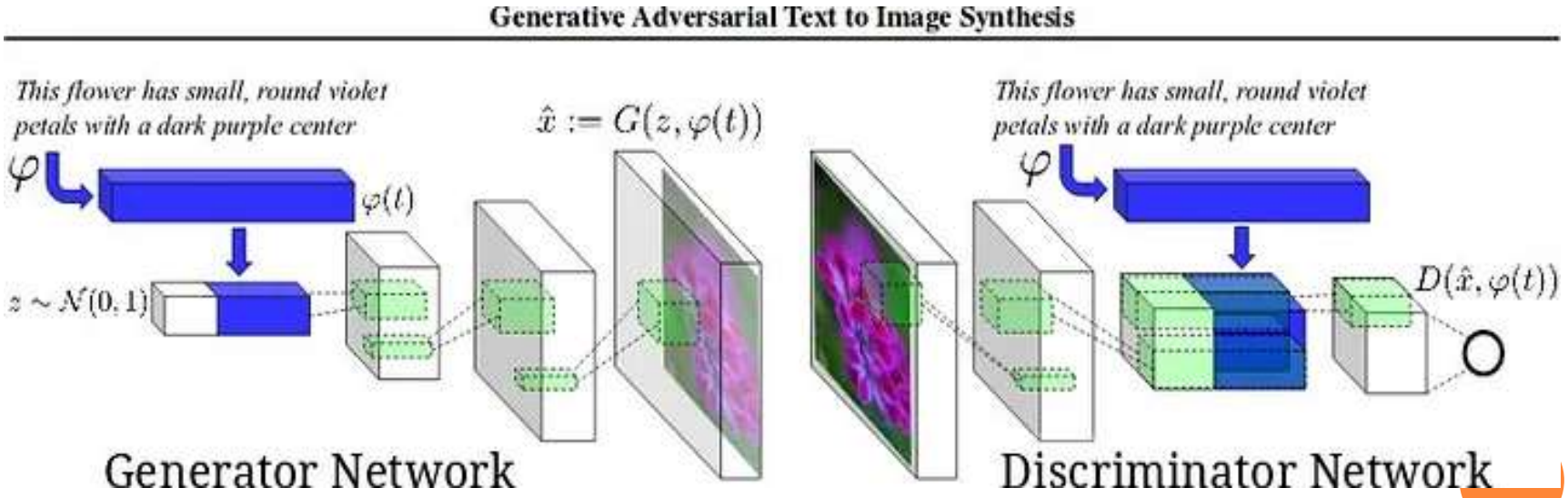
CONTD..



Architecture of Cycle GAN (CGAN)

CONTD..

- 5. Generative Adversarial Text to Image Synthesis: Text to image synthesis (T2I) is one of the most challenging and interesting tasks in the modern domain of Computer Vision. These GANs can generate images from textual descriptions, bridging the gap between natural language and visual data.

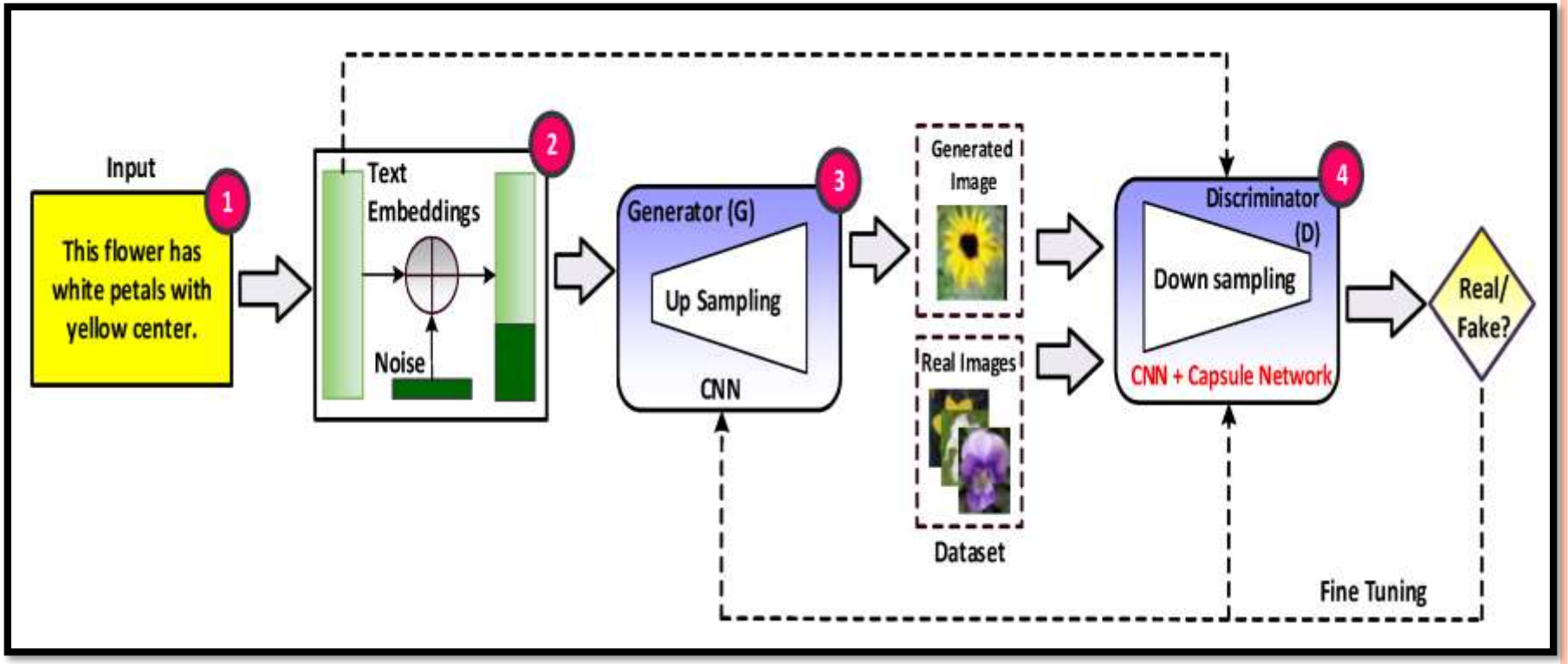


Architecture of Text to Image Synthesis

CONTD..

Example

CONTD..

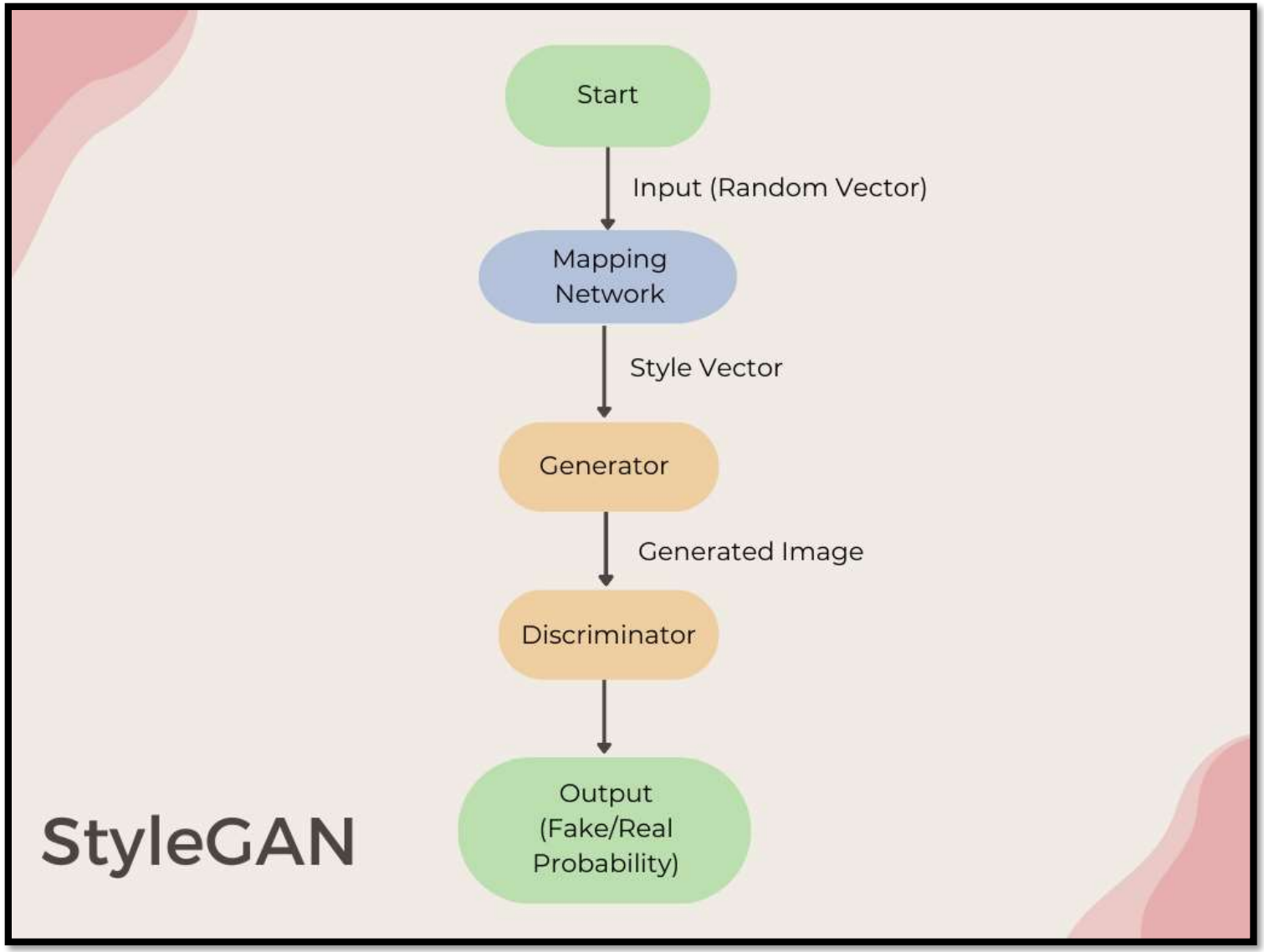


Example

CONTD..

- 6. Style GAN: **Style GAN** proposes a lot of changes in the generator part.
- StyleGAN was designed to create realistic images while manipulating and controlling certain features or styles of the image. These styles associated with the generated images could be features like color, texture, pose, etc.

CONTD..



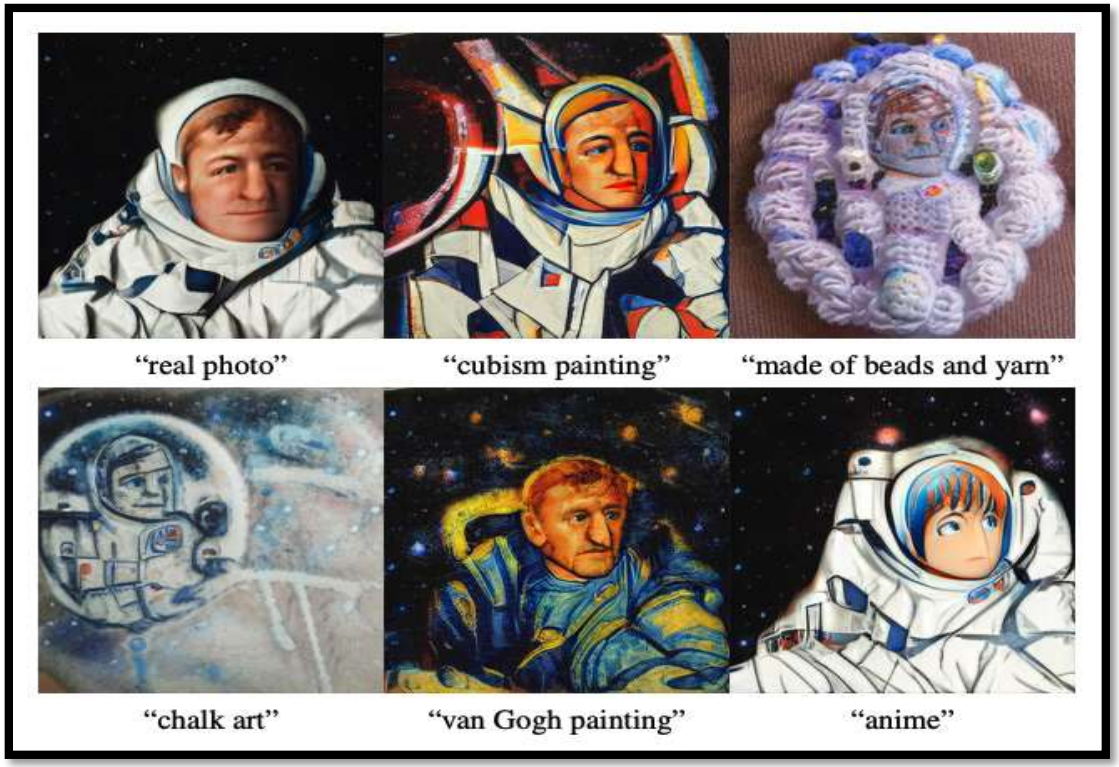
Architecture of Text to Image Synthesis

CONTD..

- First, StyleGAN takes a random vector as an input. This vector is mapped into a style vector representing different aspects of the image's style and appearance.
- The generator network then generates an image using the style vector.
- The discriminator network evaluates whether the generated image is real or fake (generated).

CONTD..

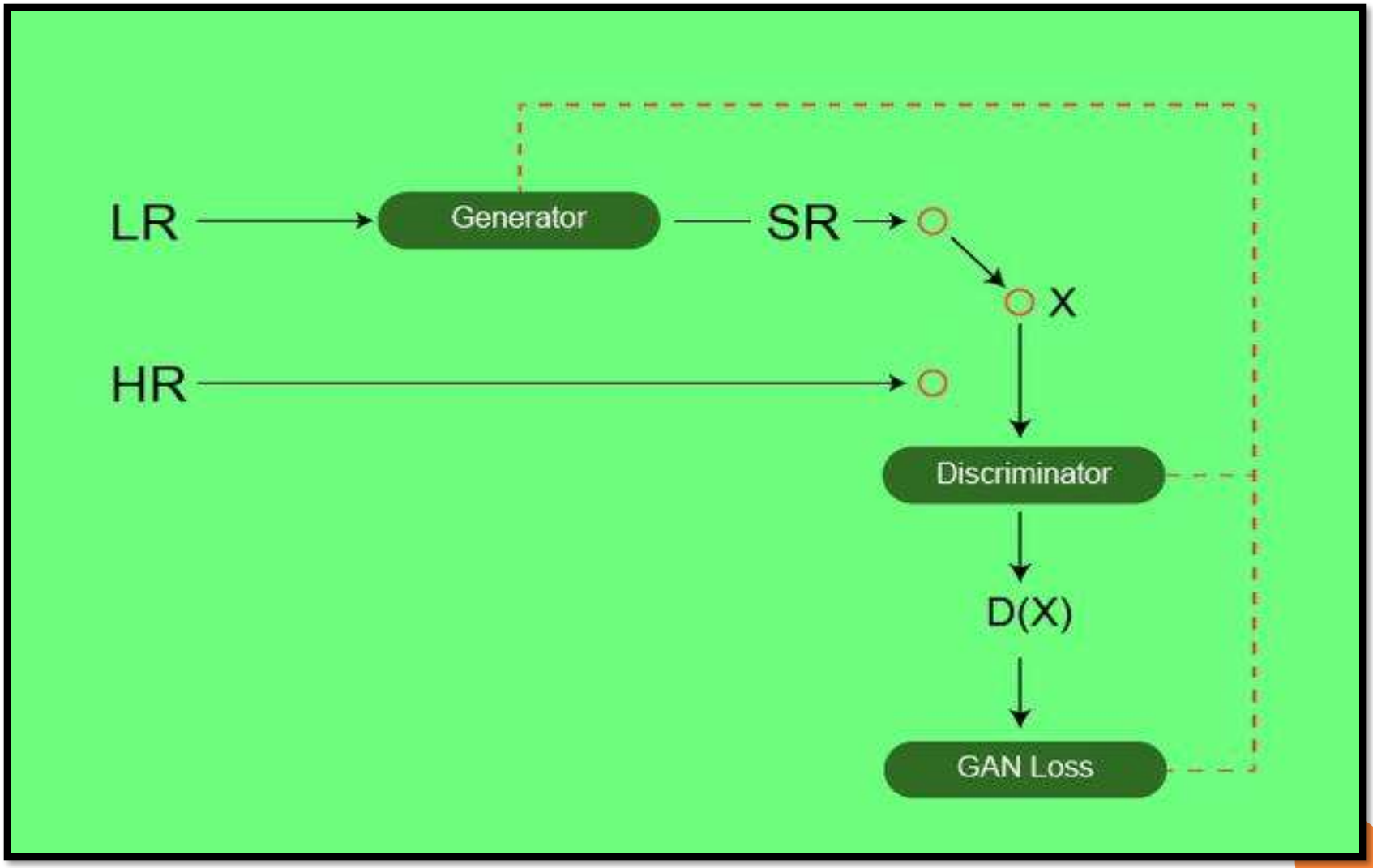

Content image
Style image
Output image


"real photo"
"cubism painting"
"made of beads and yarn"
"chalk art"
"van Gogh painting"
"anime"

CONTD..

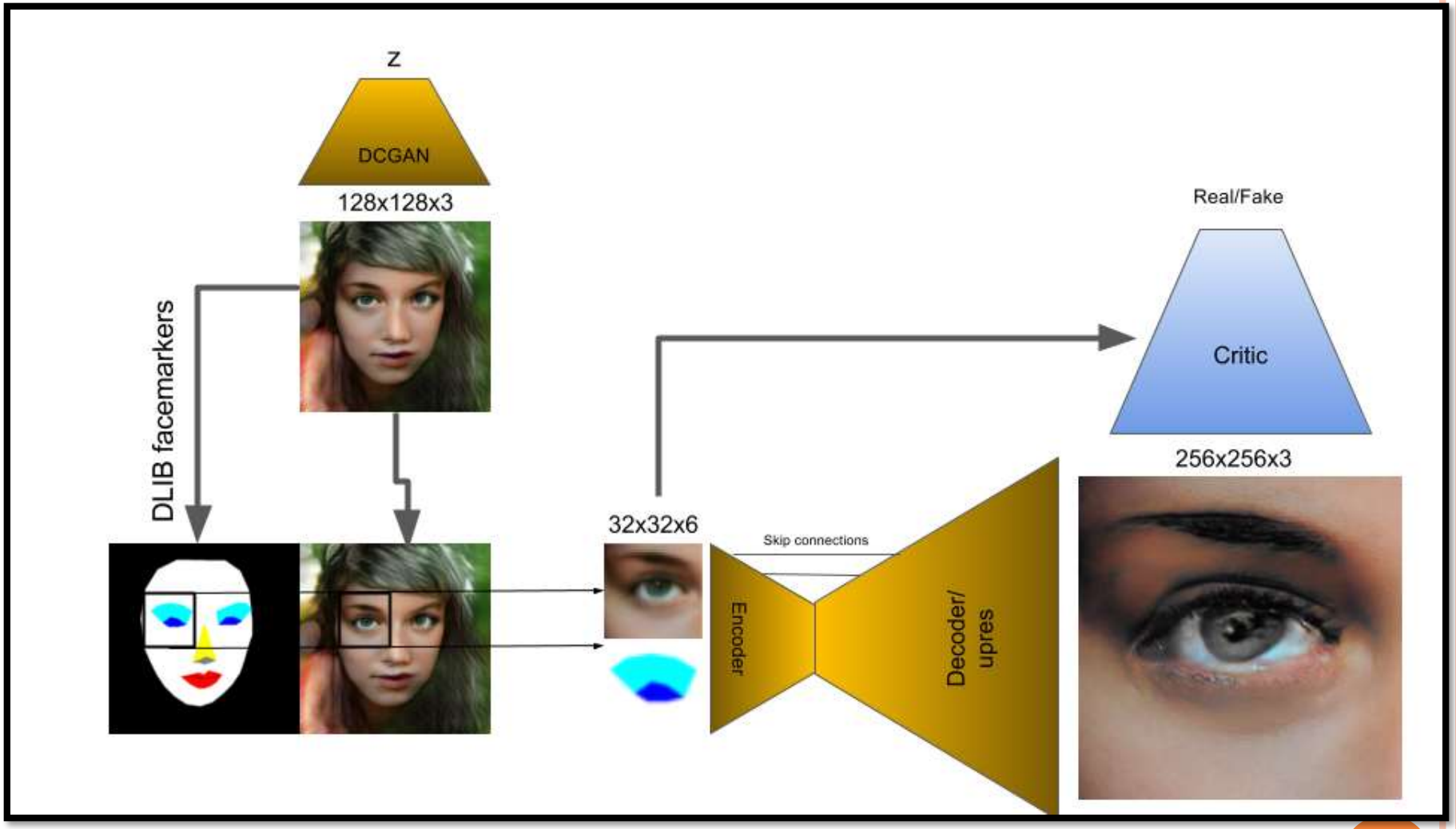
- 7. Super Resolution GAN (SRGAN): SRGAN enhances the resolution of images, turning low-resolution inputs into high-resolution outputs without losing detail.
- SRGAN was proposed by researchers at Twitter. The motive of this architecture is to recover finer textures from the image when we upscale it so that it's quality cannot be compromised.

CONTD..



Architecture of Super Resolution GAN

CONTD..



Example of Super Resolution GAN

Applications of GAN

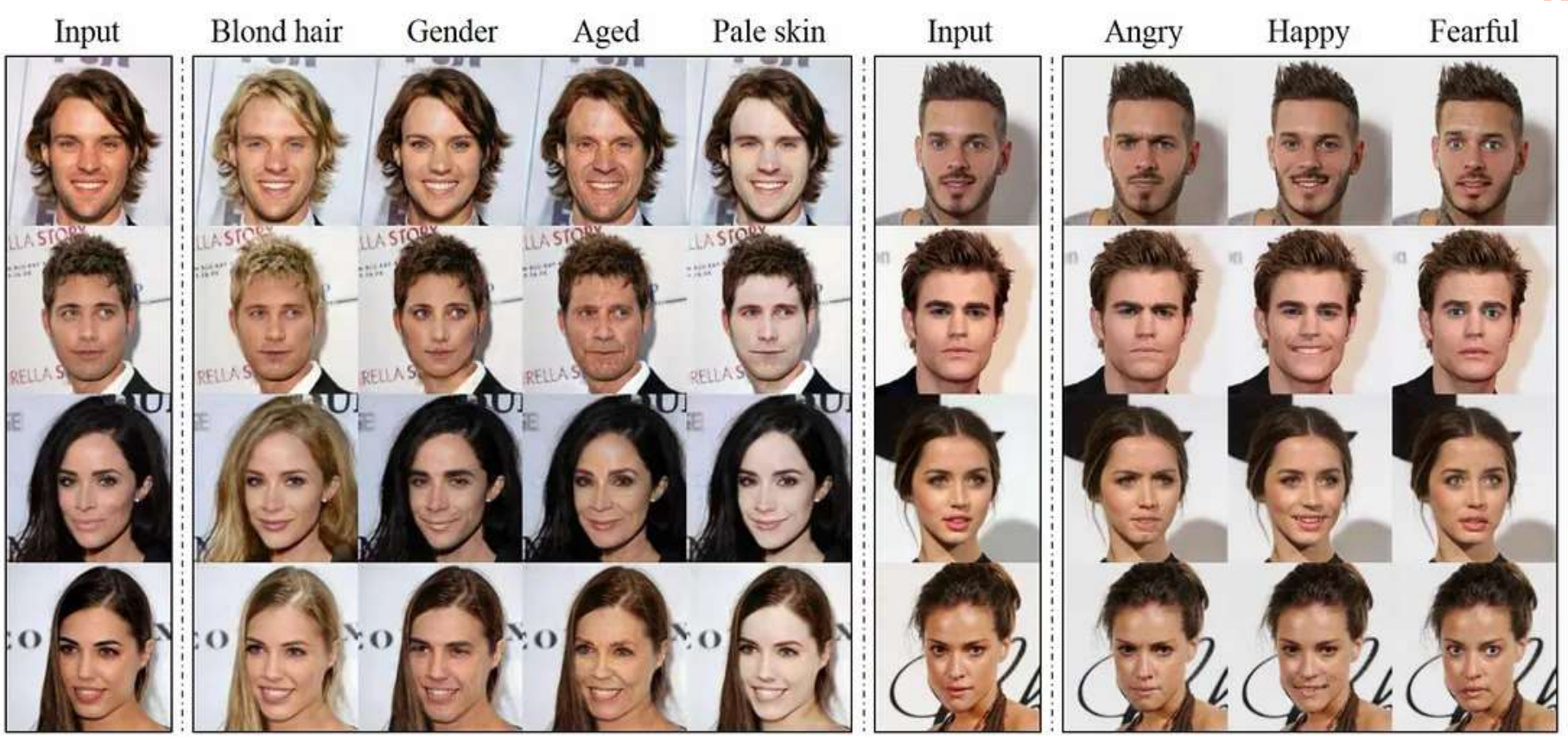
- NVIDIA research center has to develop these GAN Applications in real time.

- They generate high quality and photo realistic Images, Videos
- **1. Image to Image Translation:** GANs Can be in use for translating data from images. In image-to-image translations, GANs account for tasks such as:
  - Changing sketches to color photographs.
  - Converting satellite images to google maps.
  - Translation of photos from day to night and vice versa.
  - Translation of black and white photographs to color.

CONTD...

CONTD..

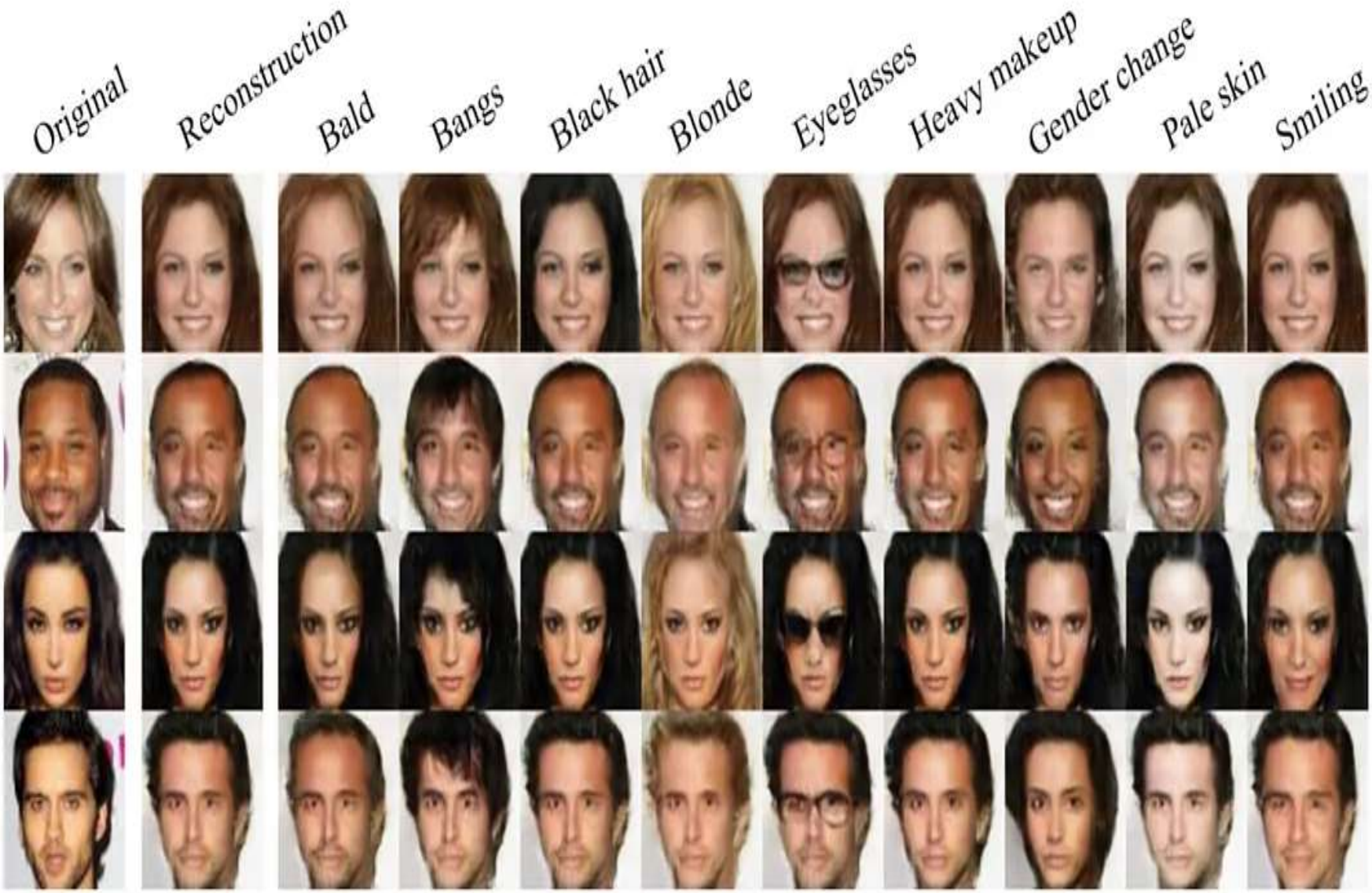
- StarGAN is an image-to-image translation for one domain to another. For example, given a happy face, we want to transform it into a fearful face.



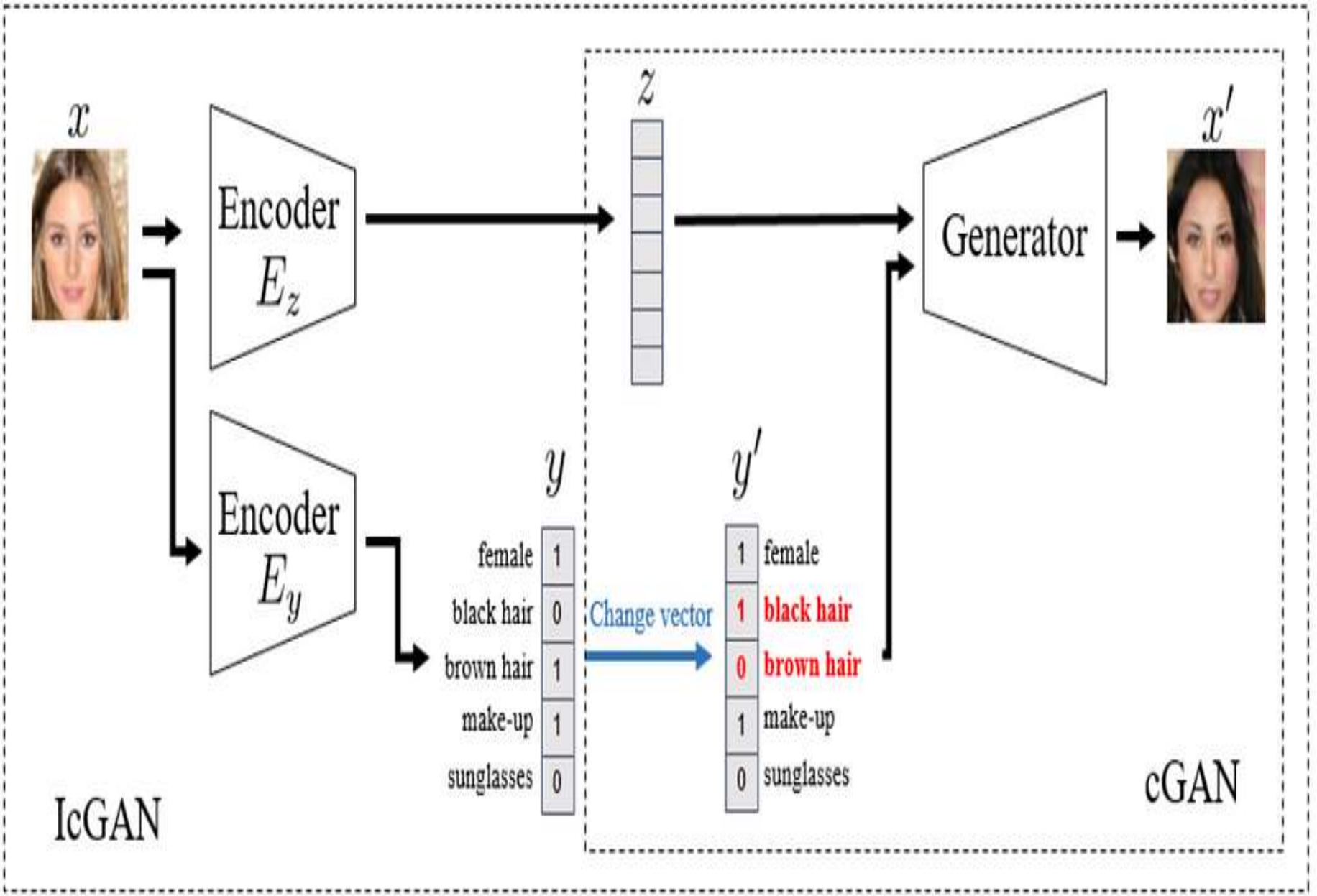
CONTD..

- **2. Image Editing:** Most image editing software these days don't give us much flexibility to make creative changes in pictures.
- **For example,** let's say you want to change the appearance of a 90-year-old person by changing his/her hairstyle. This can't be done by the current image editing tools out there. But guess what? Using GANs, **we can** reconstruct images **and** attempt to change the appearance drastically.

CONTD..


Original
Reconstruction
Bald
Bangs
Black hair
Blonde
Eyeglasses
Heavy makeup
Gender change
Pale skin
Smiling

CONTD..


x
Encoder
E_z
z
Generator
x′
Encoder
E_y
y
y′
female 1
black hair 0
brown hair 1
make-up 1
sunglasses 0
Change vector
1 female
1 black hair
0 brown hair
1 make-up
0 sunglasses
IcGAN
cGAN

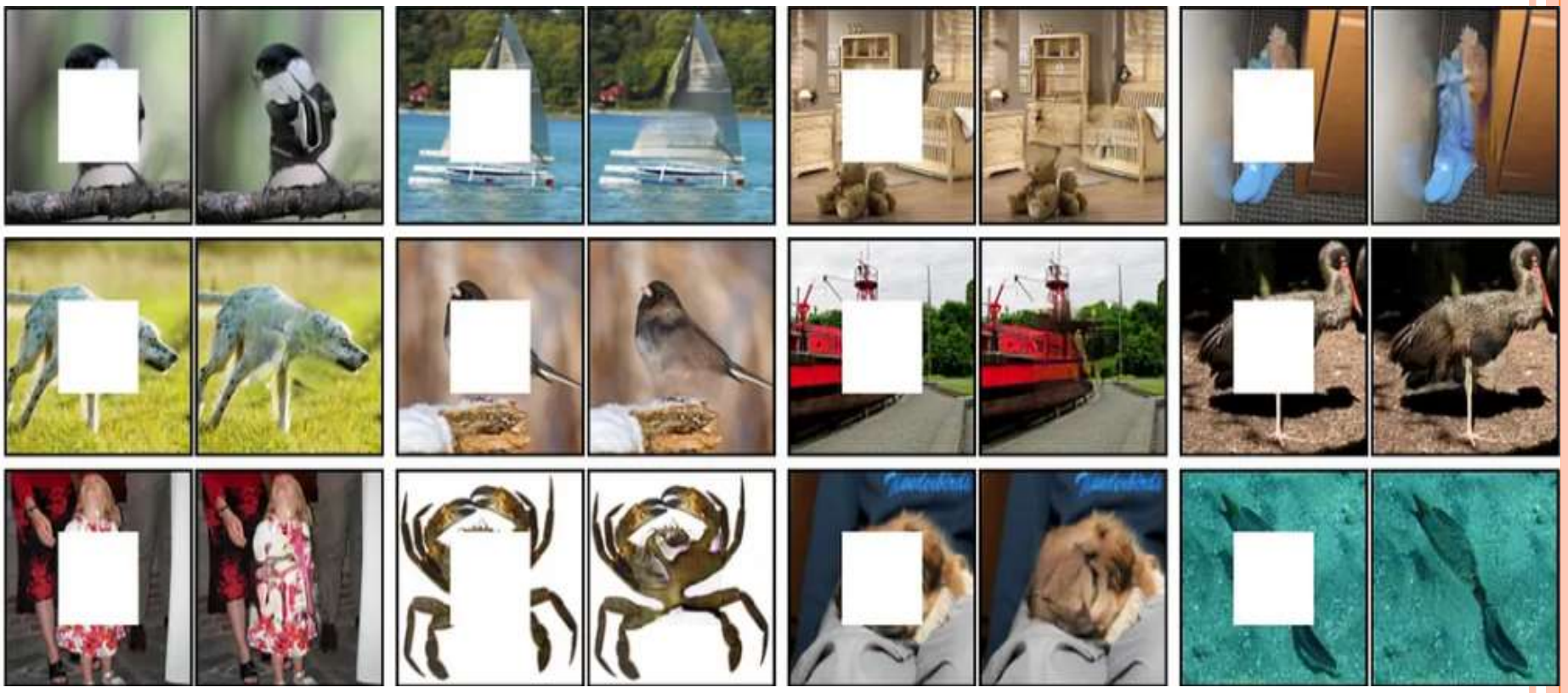
CONTD..

- **4. Face Synthesis:** Synthesis faces in different poses: With a single input image, we create faces in different viewing angles.
- For example, we can use this to transform images that will be easier for face recognition.

. Synthesis results under various illuminations. The first row is the synthesized image, the second row is the input.

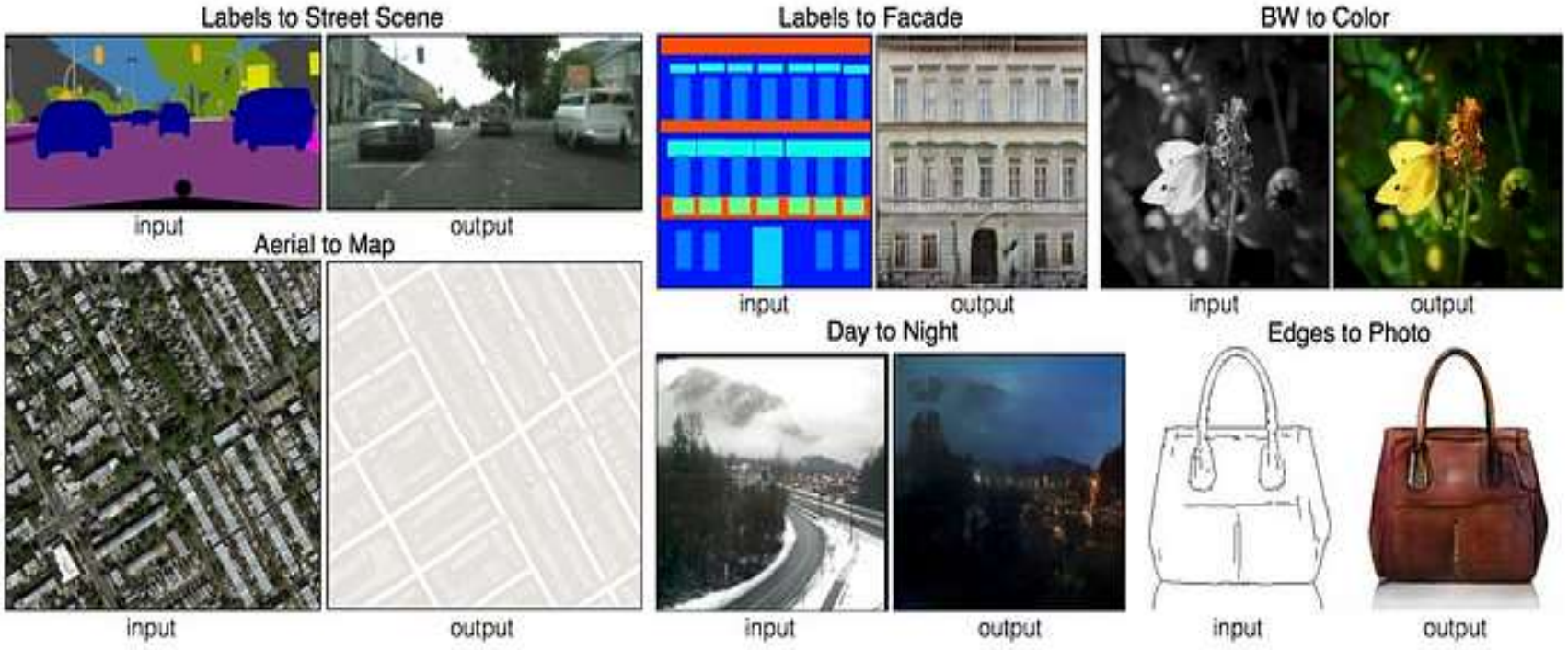
CONTD..

- **5. Image Painting: Repair images** have been an important subject decades ago. **GAN is used to repair images** and **fill the missing part** with **created "content".**



CONTD..

- **6. Pix to Pix:** Pix2Pix is an **image-to-image translation** that get quoted in **cross-domain GAN**'s frequently. For example, it converts a satellite image into a map (the bottom left).



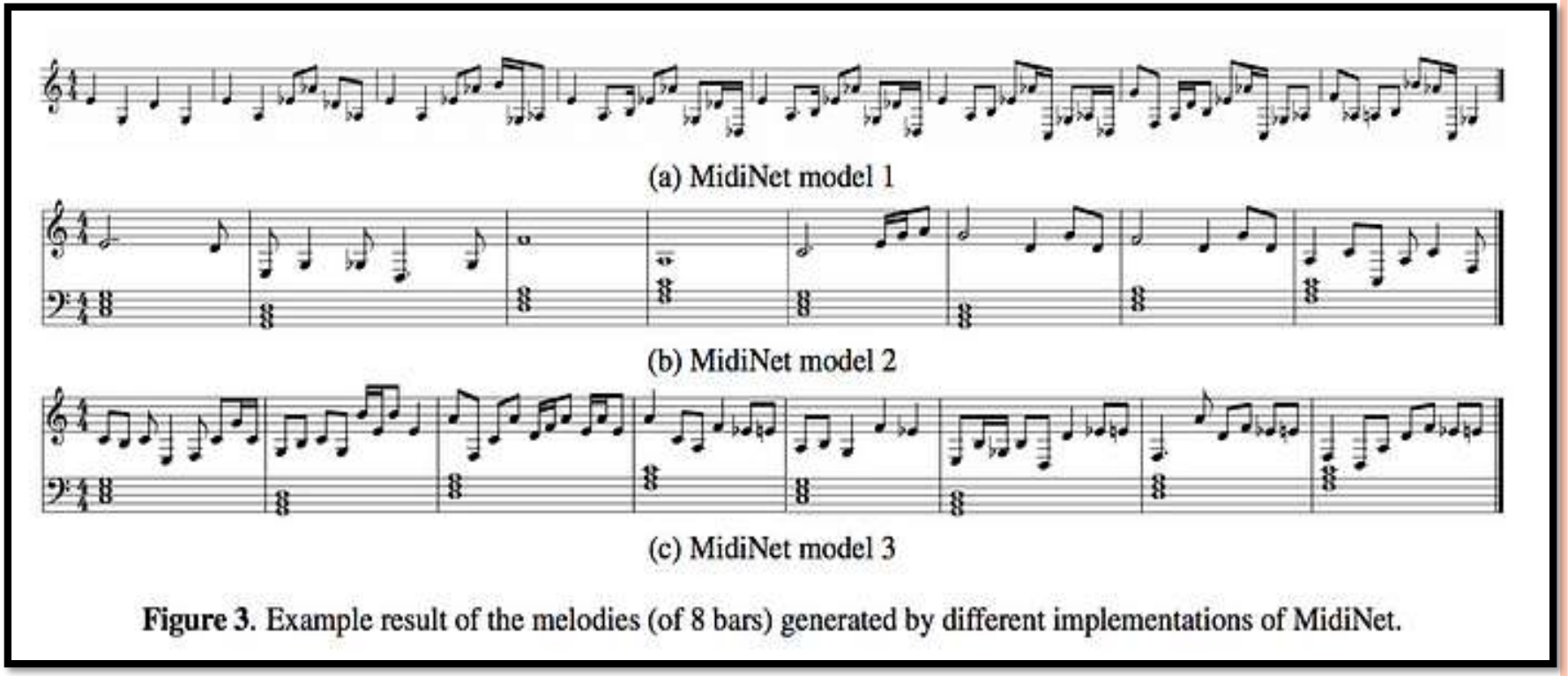
CONTD..

- **DeblurGAN** performs **motion deblurring.**

Figure 2: GoPro images [25] processed by DeblurGAN. Blurred – left, DeblurGAN – center, ground truth sharp – right.

CONTD..

- **7.Music Generation:** GAN can be applied to non-image domain, like composing music.



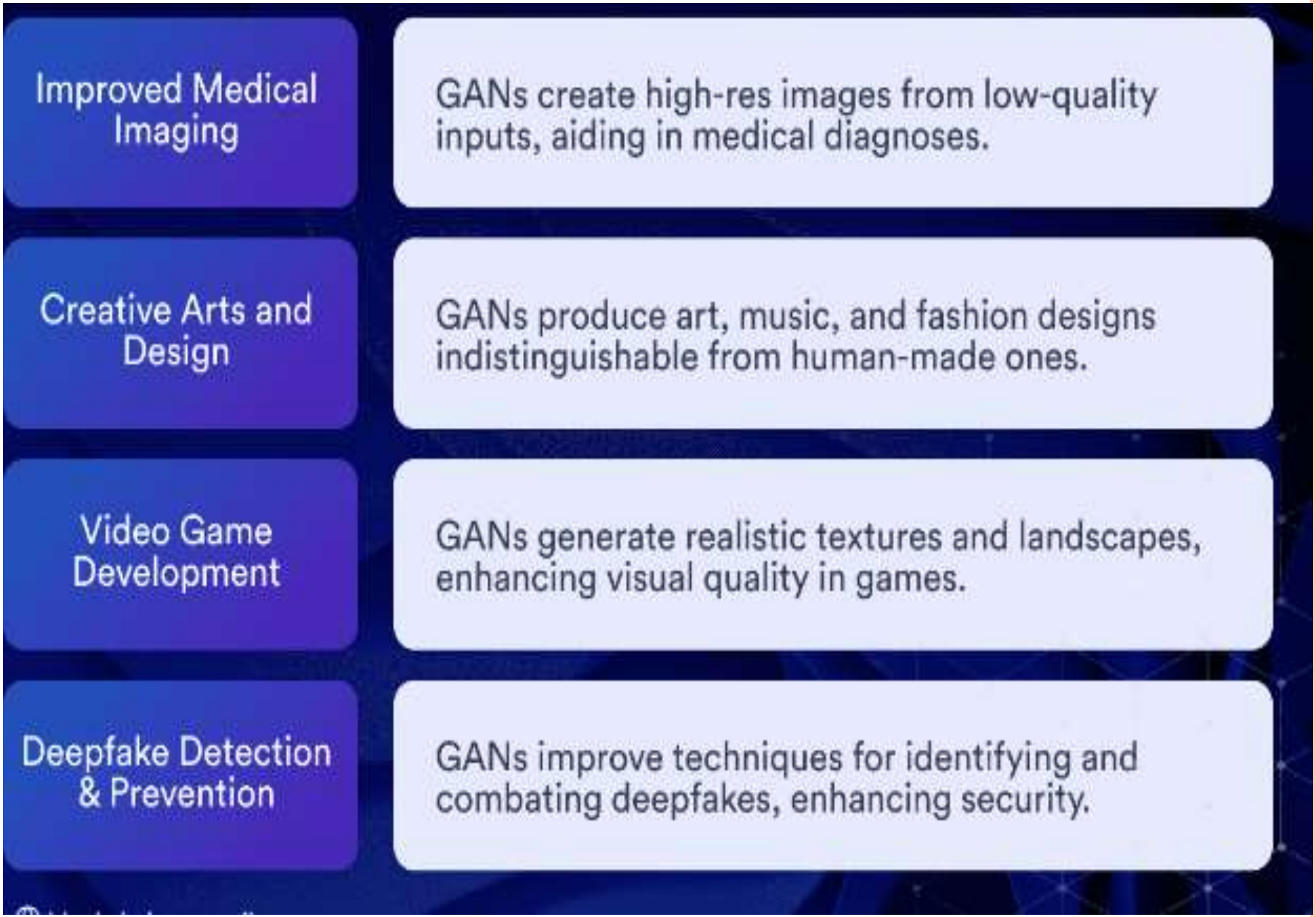
(a) MidiNet model 1

(b) MidiNet model 2

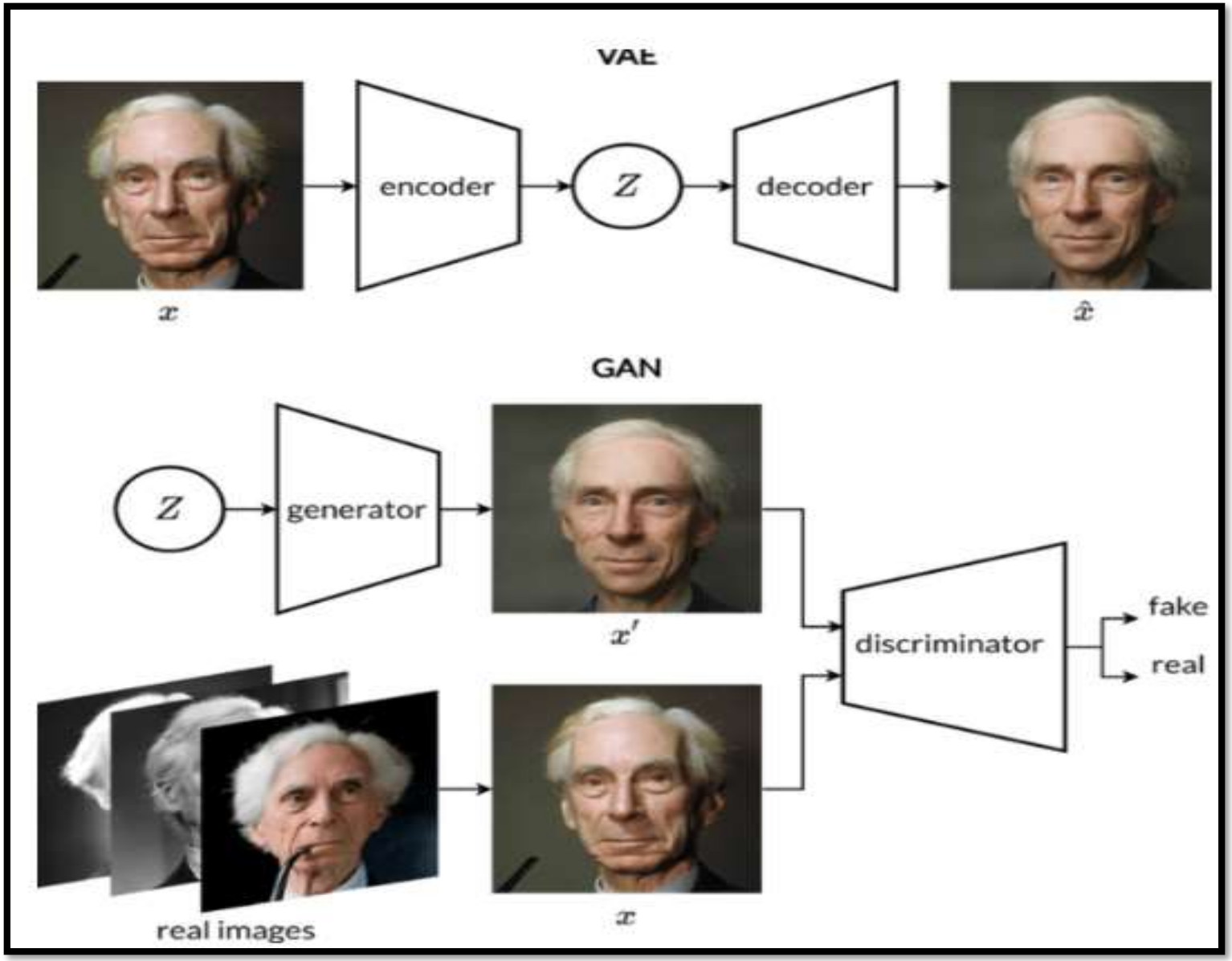
(c) MidiNet model 3

Figure 3. Example result of the melodies (of 8 bars) generated by different implementations of MidiNet.

Future Generations of GANs



Differences b/w VAN and GAN



CONTD..

Topics	Generative Adversarial Networks	Variational Autoencoder
Functionality	Composed of two models **(a generator and a discriminator)** that compete with each other. The **generator creates fake samples** and the discriminator attempts to distinguish between **real and fake samples.**	Composed of an **encoder and a decoder**. The **encoder maps inputs to a latent space,** and the **decoder maps points in the latent space back to the input space.**
Output Quality	Can **generate high-quality, realistic outputs.** Known for **generating images that are hard to distinguish from real ones**.	Generally produces **less sharp or slightly blurrier images compared to GANs**. However, this may depend on the specific implementation and problem domain.
Training Stability	Training GANs can be **challenging and unstable,** due to the **adversarial loss used in training.**	Generally **easier and more stable to train** because they use a **likelihood-based objective function.**

Thank You

UNIT-II

PART – II

MAJOR ARCHITECTURES OF DEEP NETWORKS

Deep Belief Networks

Deep Belief Networks

- We create *Deep Belief Networks (DBNs)* to address issues with classic neural networks in deep layered networks.
- For example – slow learning, becoming stuck in local minima owing to poor parameter selection, and requiring a large number of training datasets of these given input layer.
- A DBN is a deep-learning architecture introduced by Geoffrey Hinton in 2006. **Deep Belief Networks (DBNs)** are a type of deep learning architecture combining unsupervised learning principles and neural networks.
- **Deep Belief Networks (DBNs)** are sophisticated artificial neural networks used in the field of deep learning.

CONTD..

- They are designed to **discover and learn patterns within large sets** of data automatically.
- Imagine them as **multi-layered networks**, where **each layer is capable of making sense of the information received from the previous one,** gradually **building up a complex understanding of the overall data.**
- They are **composed of layers** of **Restricted Boltzmann Machines (RBMs),** which are **trained one at a time in an unsupervised manner.**

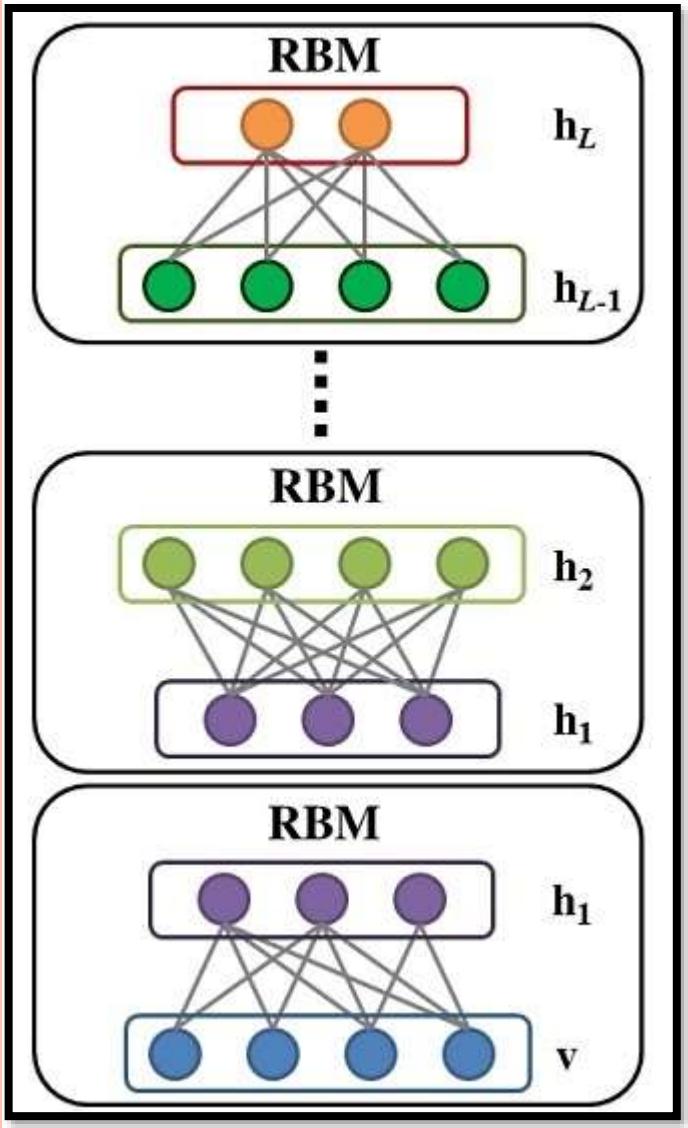
CONTD..

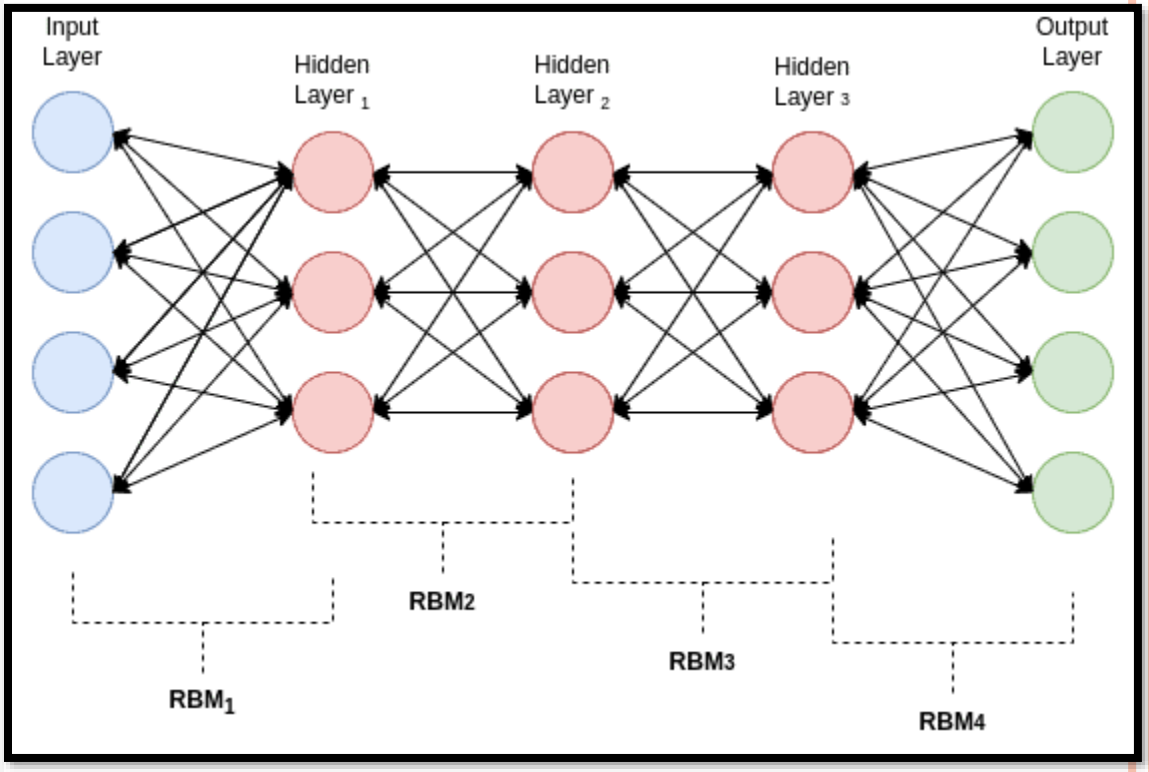
- **Several Restricted Boltzmann Machines (RBM)** can be **stacked** and **trained in a greedy manner** to **form Deep Belief Network architecture.**
- i.e, It is a **composition of Stack** of **Unsupervised Neural Network** such as **Restricted Bolzmann Machine(RBM).**
- **Each RBM** has a **Visible Layer (Input)** and **Hidden Layer (Output)**
- Here the **Hidden Layer in Stack1** is the **Visible Layer for the Stack2.etc.**

CONTD..

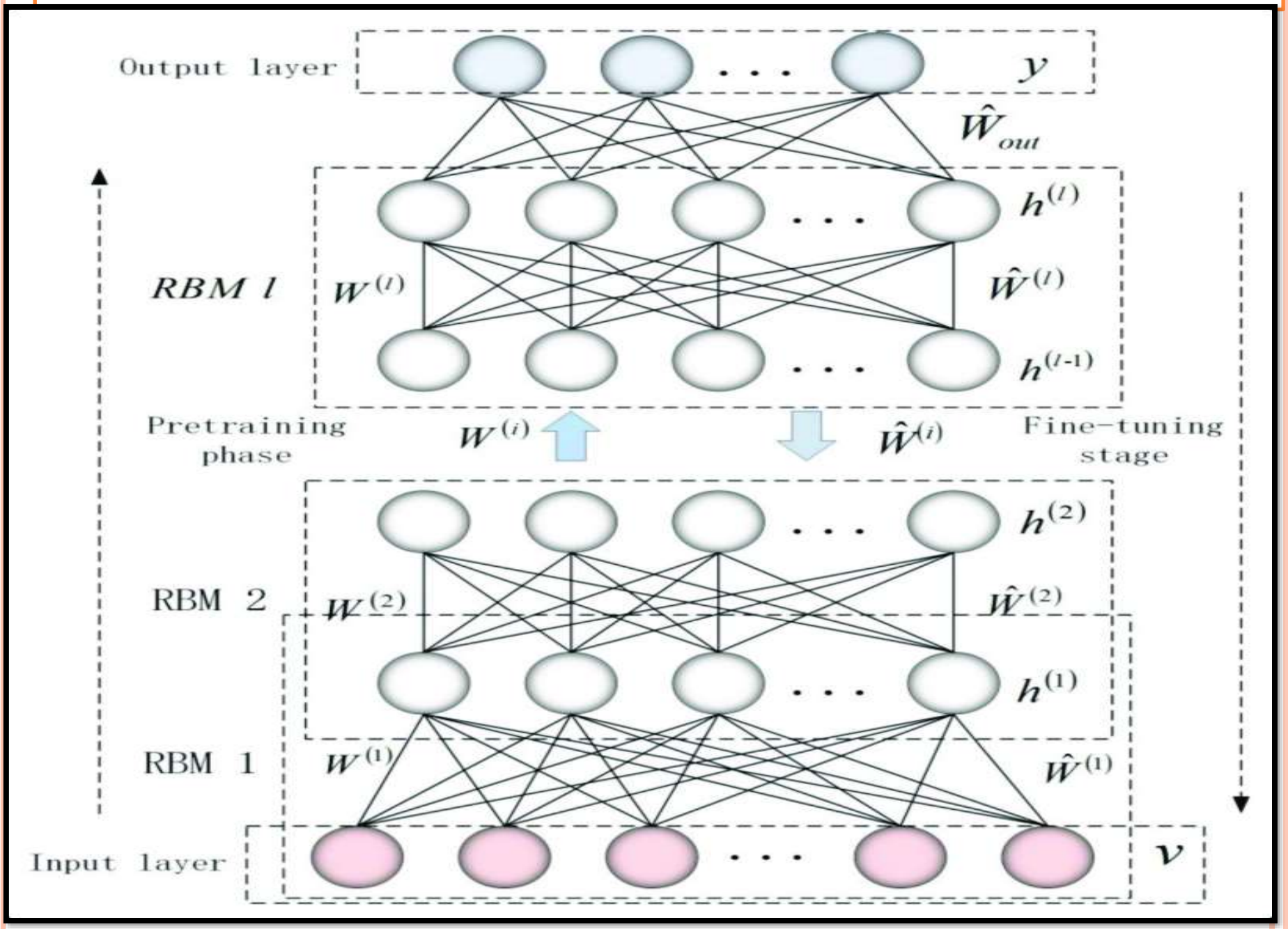
- The **output of one RBM** is used as the **input to the next RBM,** and the **final output** is used for **supervised learning tasks** such as **classification or regression**.
- Here, **Each RBM network is trained independently** with their **greedy method.**
- DBNs work similarly to **traditional multi-layer perceptrons (MLPs)** and offer certain benefits over them, including **faster training and better weight initialization.**

Structure of Deep Belief Network



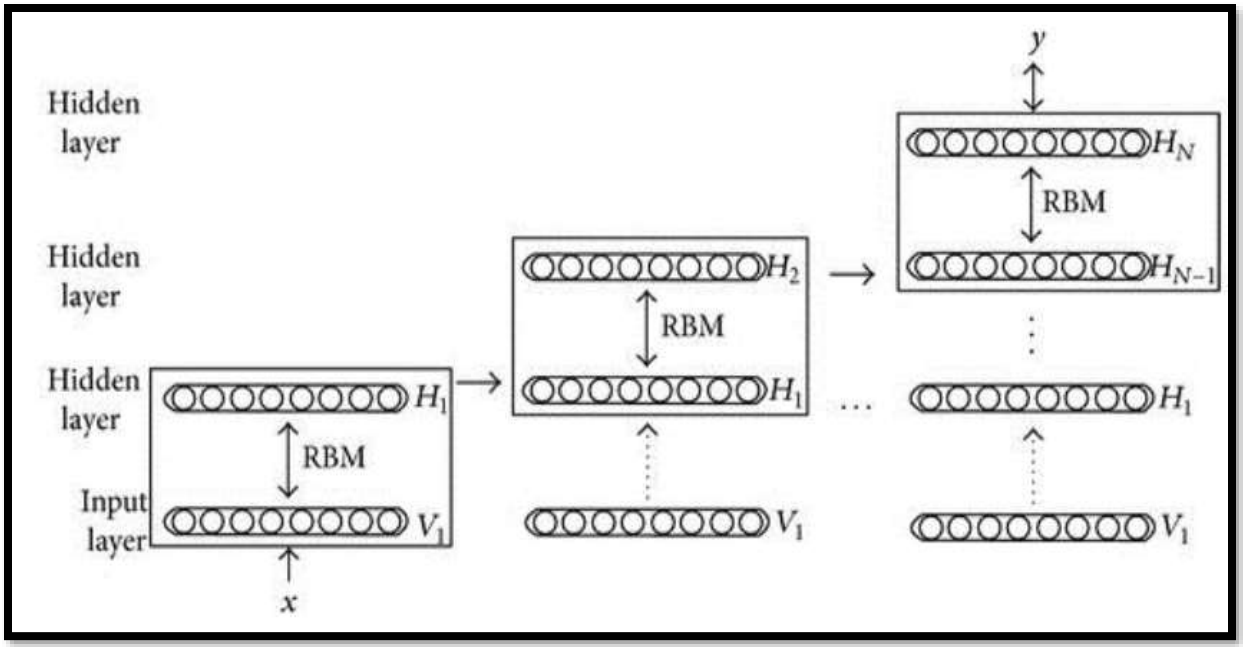


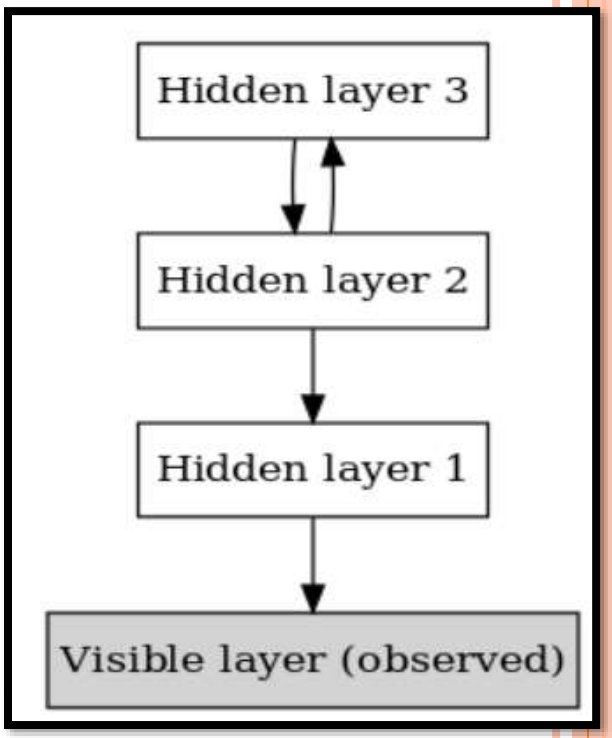
Structure of Deep Belief Network



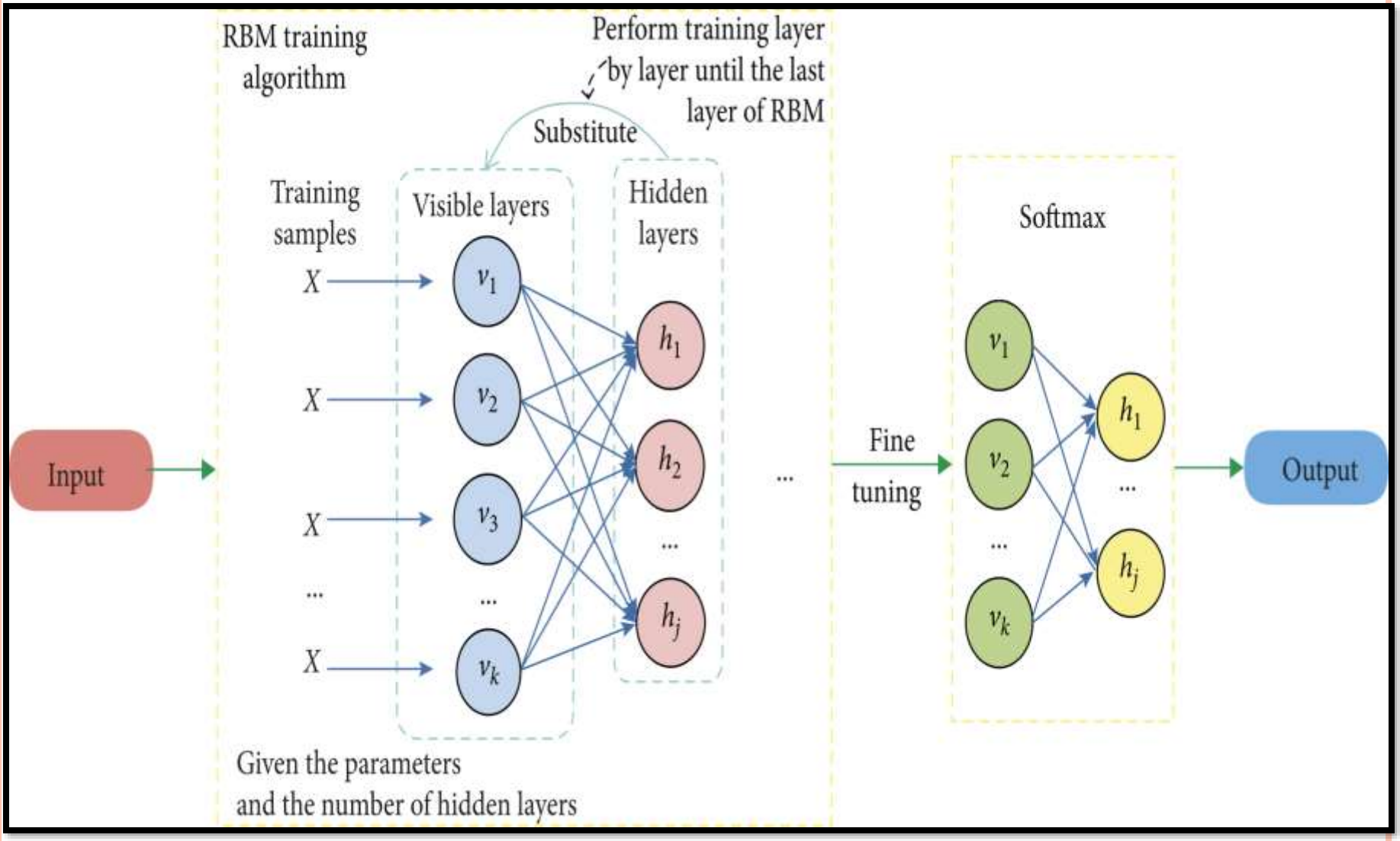
CONTD..

- In the DBN, we have a hierarchy of layers. The top two layers are the associative memory, and the bottom layer is the visible units. The arrows pointing towards the layer closest to the data point to relationships between all lower layers.





How Deep Belief Network Works



How Deep Belief Network Works

- The Deep Belief Network (DBN) algorithm consists **of two main steps:**
  - 1. Pre Training
  - 2. Fine Tuning
- 1. Pre Training: In the **pre-training phase**, the **network learns to represent the input data layer by layer**. **Each layer** is **trained independently as an RBM**, which allows the **network to learn complex data representations efficiently.**
- The **ensemble's first layer** (often called the input layer or bottom layer) **interacts directly with the raw data**, **learning its features** and **creating a latent representation during the process.**

CONTD..

- Each subsequent layer is then trained so the first's output becomes the next's input. This greedy layer-wise learning allows for efficient feature learning.
- We iterate this process multiple times, covering various data samples and updating weights after each pass. Finally, the last layer (output layer) outputs the network's prediction.

CONTD..

- 2. Fine Tuning: In the **fine-tuning phase**, the DBN adjusts its parameters for a specific task, like classification or regression.
- This is typically done using a technique known as backpropagation, where the network's performance on a task is evaluated, and the errors are used to update the network's parameters.
- This phase often involves supervised learning, where the network is trained with labelled data.

Algorithm

- The algorithm for training a DBN can be summarized as follows:
  - Initialize the network with random weights.
  - Start with the first layer and work your way to the last layer as you use unsupervised learning to train each layer of the network.
  - Fine-tune the entire network using supervised learning and back propagation.
  - Repeat steps 2 and 3 until the network has converged.

Mathematical Representation

- **Deep Belief Networks (DBNs)** employ several mathematical concepts, blending probability theory with neural network structures. At their core, they use Restricted Boltzmann Machines (RBMs) for layer-wise learning, which are based on probabilistic graphical models.
- 1. Energy-Based Model: Each RBM within a DBN is an energy-based model. For an RBM with visible units v and hidden units h, the energy function is defined as:

$$E(v,h) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_{i,j} v_j h_j w_{ij}$$

Here, ai and bj are bias terms, and wij represents the weights between units.

Mathematical Representation

- 2. Probability Distribution: The **probability of a given state** of the **RBM** is defined by the **Boltzmann distribution:**

$$P(v,h) = \frac{e^{-E(v,h)}}{Z}$$

where **Z is the partition function**, a **normalization factor** calculated as the **sum over all possible pairs of visible and hidden units**.

Benefits of DBN

- Deep belief networks use probabilistic modeling and a supervised learning approach to offer certain benefits over conventional neural networks. These include:
- Ability to handle large data using hidden units to extract underlying correlations.
- Faster training and better results. Achieving global minima due to better weights initialization.
- Model Interpretability: Like many deep learning models, DBNs can act as "black boxes," making it difficult to understand how they are making predictions or what features they have learned.

Applications of DBN

- Deep Belief Networks (DBNs) have been **applied** in a **variety of fields, including:**
- **Computer vision:** DBNs have been used for tasks like **recognizing objects in pictures** or **putting pictures into different groups.**
- **Speech recognition:** DBNs have been used for **speech recognition tasks,** such as **transcribing speech into text.**
- **Natural language processing:** DBNs have been used for natural languages processing tasks, such as sentiment analysis and text classification.

CONTD..

- **Recommender systems:** DBNs have been used in recommender systems, which give users suggestions based on what they like and how they act.
- **In bioinformatics,** DBNs have been used to **predict how proteins will interact with each other,** look at **how genes are expressed**, **and find new drugs.**
- **Financial analysis:** DBNs have been used to **predict the stock market** and **figure out how risky something is**.

KEY POINTS

- DBNs have been a crucial part of the **deep learning ecosystem.** Here's what you need to know about them:
- **Deep Belief Networks** are constructed by stacking multiple Restricted Boltzmann Machines.
- We train each RBM in the stack independently with greedy learning.
- Training DBNs consists of an unsupervised pre-training phase followed by supervised fine-tuning.

KEY POINTS

- DBNs understand latent data representations and can generate new data samples.
- DBNs are also employed in classification, motion capture, speech recognition, etc.
- The DBN architecture is not very popular today and has entirely been replaced by other Deep Learning algorithms like CNN and RNN.

Thank You

UNIT-II

PART – II

MAJOR ARCHITECTURES OF DEEP NETWORKS

Generative Adversarial Networks (GANs)

Topics :

- Generative Adversarial Network
  - Architecture of GAN
  - Mathematical Notation
  - Loss Function GAN
  - Training Process of GAN
  - Types of GAN
  - Applications of GAN
  - Future generations of GAN
  - Differences b/w VAN and GAN

Generative Adversarial Networks (GANs)

- Generative Adversarial Networks (GANs) are a powerful class of neural networks that are used for unsupervised learning. It was developed and introduced by Ian J. Goodfellow in 2014.
- Generative modeling generates unstructured data such as new images or text or Videos.
- GAN is a class of algorithmic Deep learning framework having two neural networks that connect and can analyze, capture and copy the variations within a dataset.

CONTD..

- To understand the term GAN let's break it into separate three separate Words. and each of them has its **separate meaning**, which is as follows:

1. Generative – To learn a generative model, which describes how data is generated in terms of a probabilistic model. In simple words, it explains how data is generated visually.

2. Adversarial – The training of the model is done in an adversarial setting. i.e, The word adversarial refers to the context of GANs, the generative result is compared with the actual images in the data set.

CONTD..

- **This mechanism known** as **a discriminator** and this is **used to apply a model that attempts to distinguish between real and fake images.**

3. Networks – Use **deep neural networks** as artificial intelligence (AI) **algorithms for training purposes.**

GANs are a type of neural network architecture that can **generate new data based on the patterns** learned from a **given dataset**.

This means that GANs can create **entirely new, realistic images, videos, and even audio clips** that have **never existed before.**

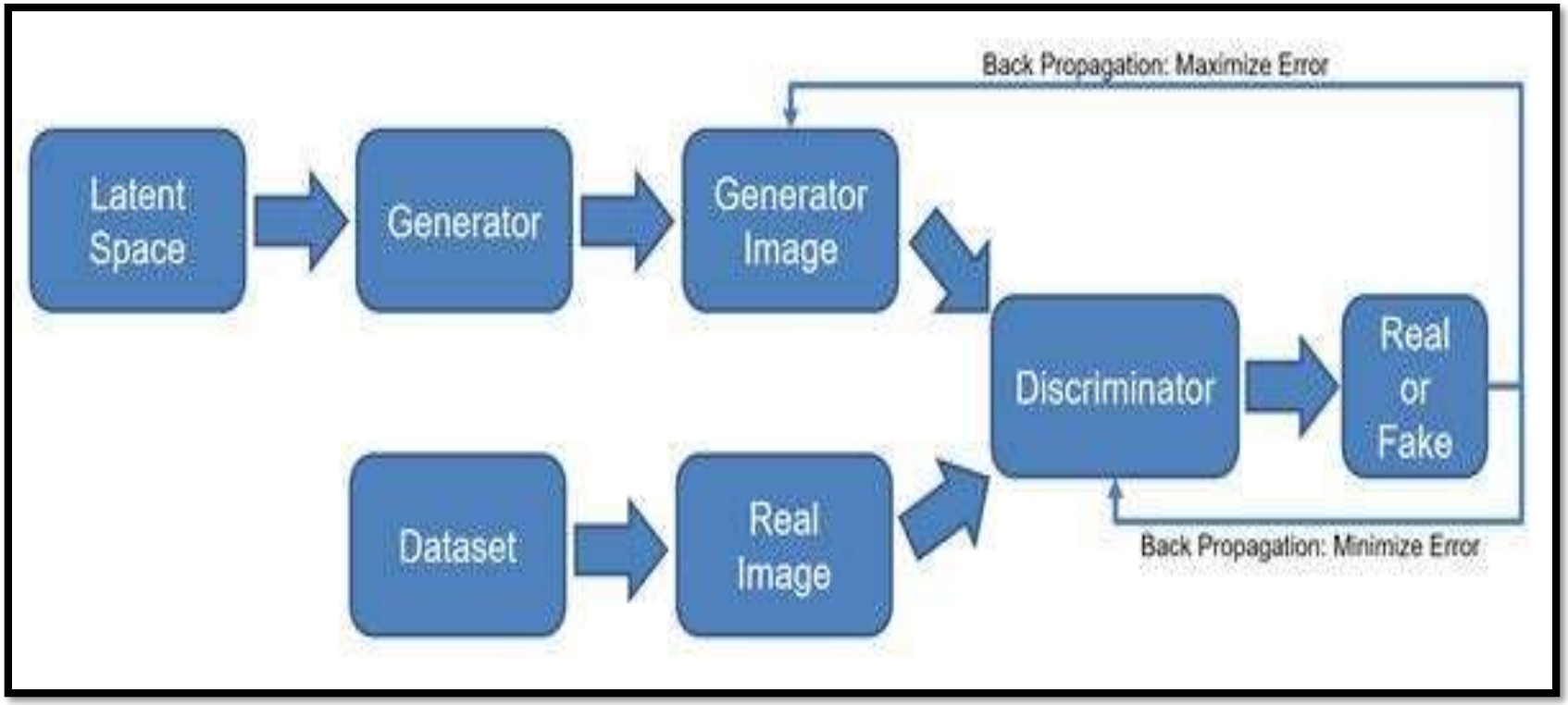
Architecture of GAN

- Generative Adversarial Networks (GANs) are a groundbreaking innovation in the field of Deep Learning, particularly within the domain of unsupervised learning.
- GANs are made up of two neural networks,
  - 1. Generator and
  - 2. Discriminator
- These **2 models** that automatically discover and learn the patterns in input data.

CONTD..

- They comprise two networks:
- The generator, which produces synthetic data. i.e, information that is artificially generated rather than produced by real-world events and
- The discriminator, which differentiates between real and generated data. This unique structure enables GANs to generate highly realistic and diverse outputs, from images to text.

CONTD..

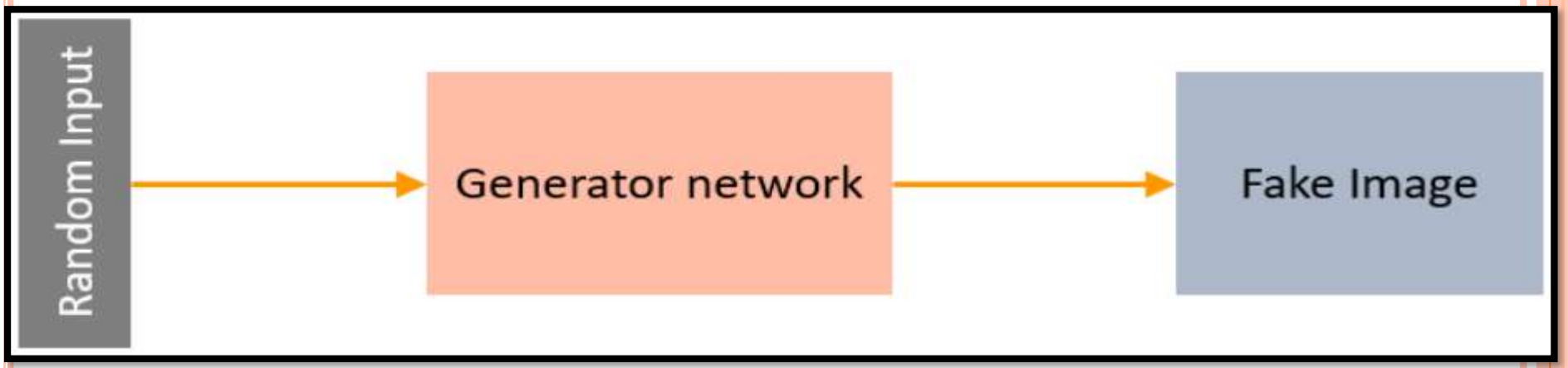


The GAN Network Process

CONTD..

The Generator: The generator network is responsible for generating new data that is similar to the training data. The generator network takes random noise as input and produces a generated output. The goal is to train the generator to produce outputs that are as close to the real data as possible.

- A Generator in GANs is a neural network that creates fake data to be trained on the discriminator.



CONTD..

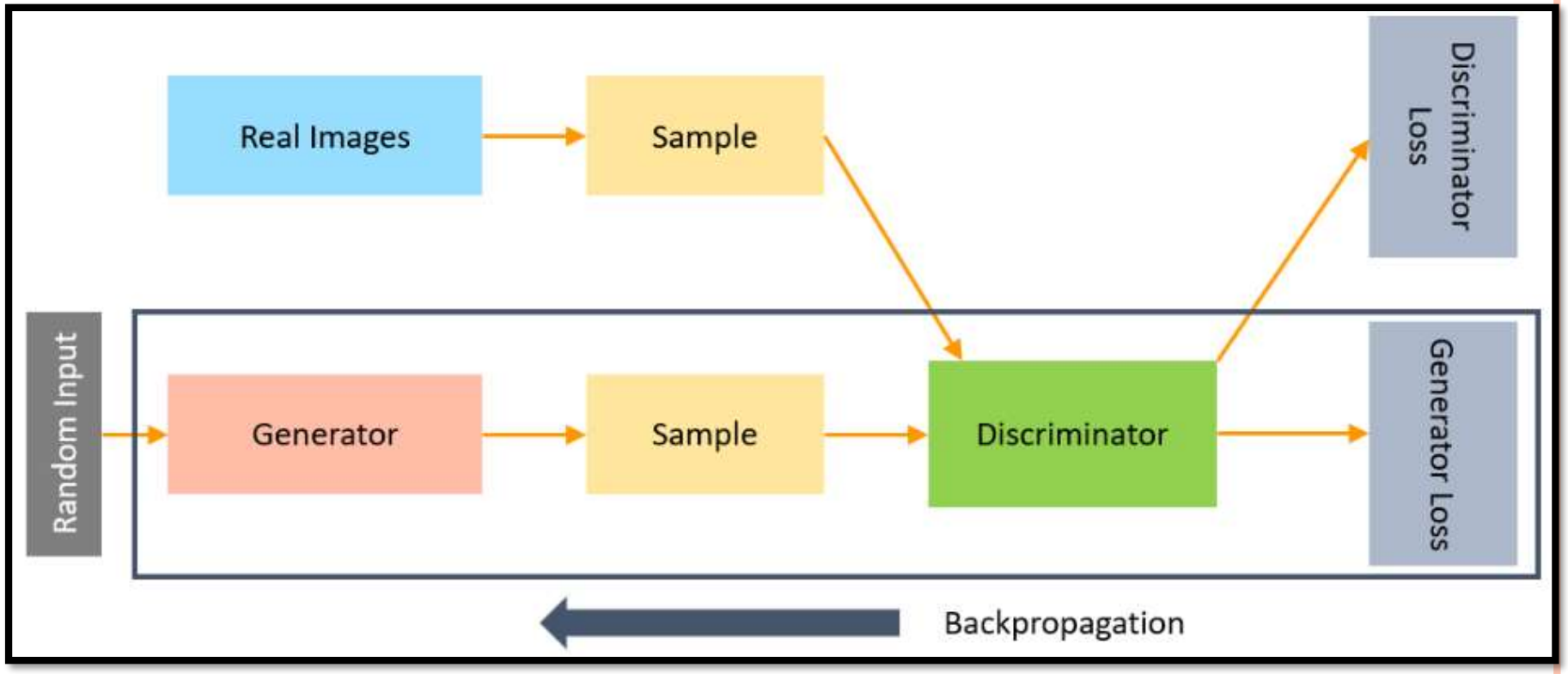
- The main aim of the Generator is to make the discriminator classify its output as real. The part of the GAN that trains the Generator includes:
  - Provide fake input or noise and get random noise to produce output based on the noise sample.
  - Predict generator output either real or fake using discriminator.
  - Calculate discriminator loss and perform back propagation.
  - Calculate gradients to update the weights of the generator.

CONTD..

- Back propagation of Generator: The generator modifies some data attributes by **adding noise (or random changes)** to certain attributes
- The generator passes the **modified data** to the discriminator
- The discriminator calculates the probability that the generated output belongs to the original dataset
- The discriminator gives some guidance to the generator to reduce the noise vector randomization in the next cycle
- The generator attempts to maximize the probability of mistake by the **discriminator**, but the discriminator attempts to minimize the probability of error.

CONTD..

- In training iterations, both the generator and discriminator iterating continuously until they reach an equilibrium state. In the equilibrium state, the discriminator can no longer recognize synthesized data. At this point, the training process is over.



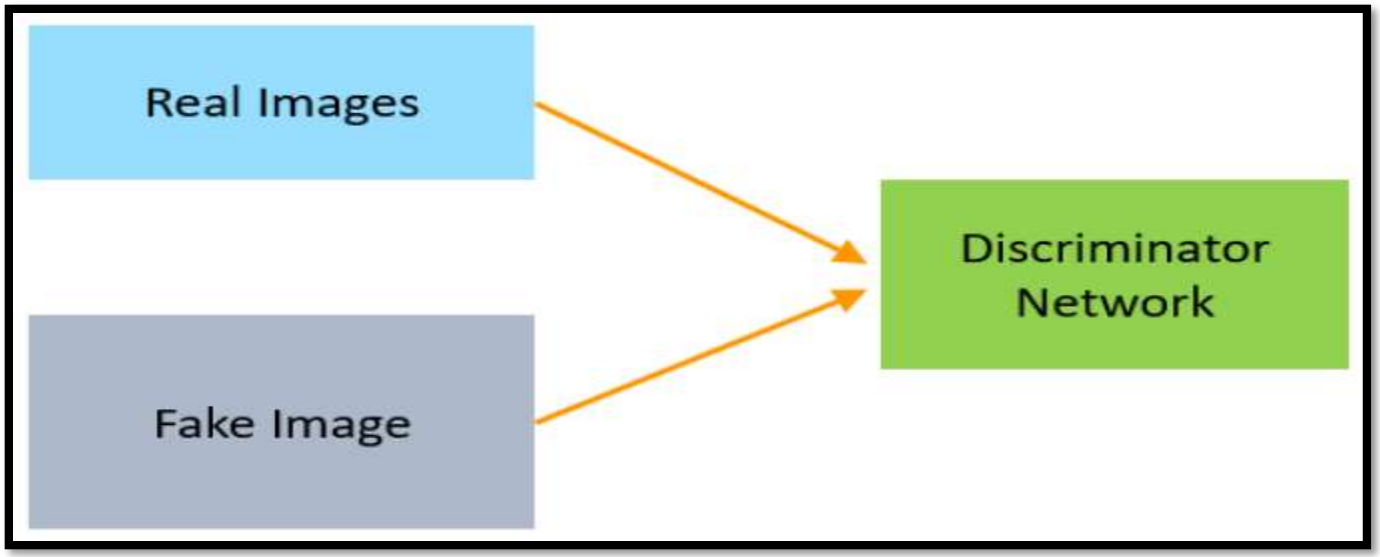
CONTD..

The Discriminator: In a GAN, the discriminator acts as a binary classifier whose main task is to differentiate between real data and data generated by the GAN's generator. The choice of network architecture for the discriminator depends on the type of data being classified.

- The Discriminator is a neural network that identifies real data from the fake data created by the Generator. The discriminator's training data comes from different two sources:
- The real data instances, such as real pictures of birds, humans, currency notes, etc., are used by the Discriminator as positive samples during training.

CONTD..

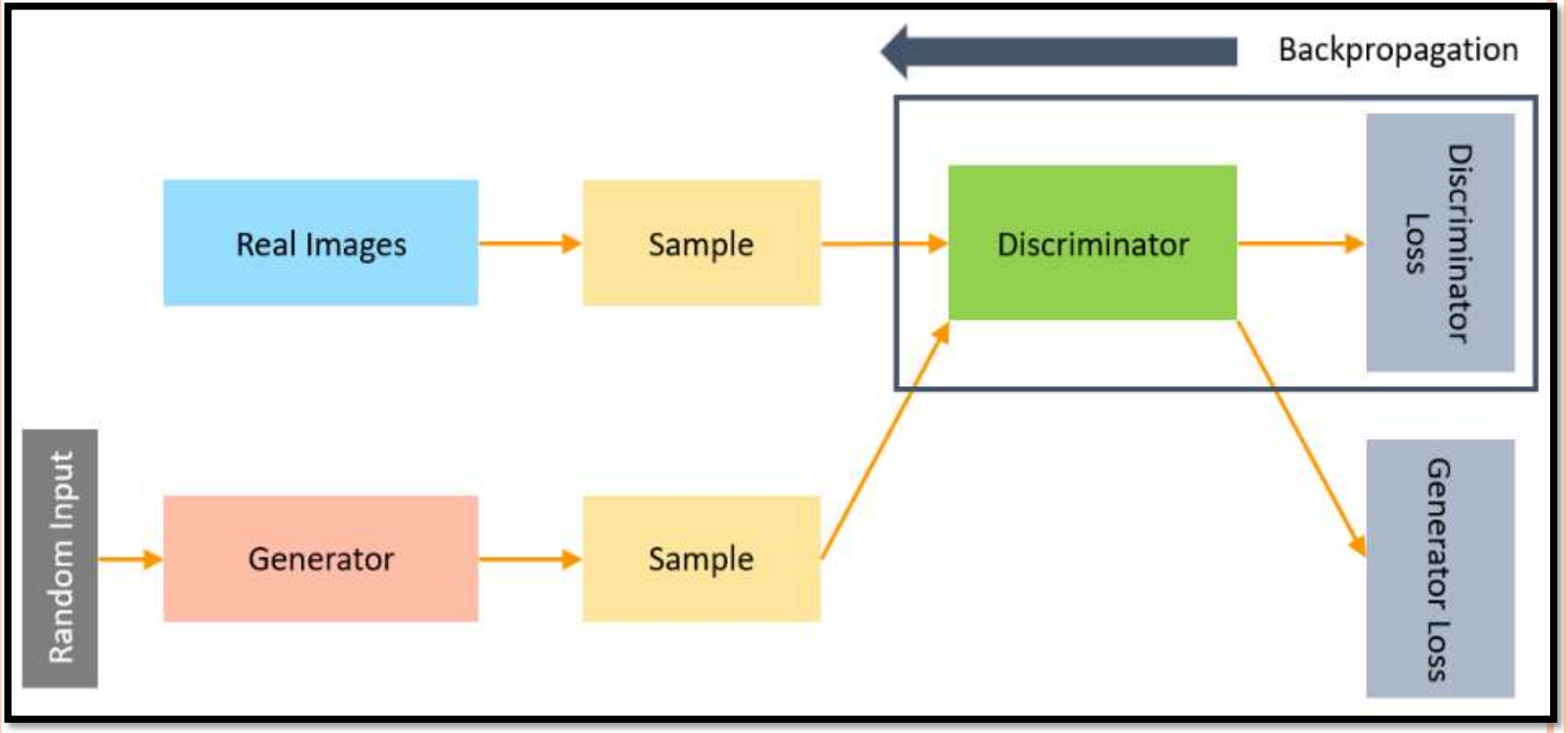
- The fake data instances created by the Generator are used as negative examples during the training process.



While training the discriminator, it connects to two loss functions. During discriminator training, the discriminator ignores the generator loss and just uses the discriminator loss.

CONTD..

- Next, The discriminator updates its weights through backpropagation from the discriminator loss through the discriminator network.



Mathematical notation

- We are basically training the Discriminator to maximize the probability of assigning correct labels to both real and generated data.

  We are also training the Generator to minimize the probability to get caught by the Discriminator, which is equivalent to minimizing log(1-D(G(z))).

- here the Discriminator is trying to minimize its reward V(D, G) and the Generator is trying to minimize the Discriminator's reward or in other words, maximize its loss.

MATHEMATICAL NOTATION

$$\min_G \max_D V(D,G)$$

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

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

x : Real data

z : Latent vector

$G(z)$: Fake data

$D(x)$: Discriminator's evaluation of real data

$D(G(z))$: Discriminator's evaluation of fake data

Loss Functions in GAN

- The training process for GANs involves minimizing a loss function that **quantifies** the difference between the generated and real data.
- There are two types of Loss Functions:
  - 1. Generator Loss
  - 2. Discriminator Loss

1. Generator Loss: The **objective of the generator in a GAN** is to produce synthetic samples that are realistic **enough to fool the discriminator**. The generator achieves this by minimizing its loss function J_G.

CONTD..

- The loss is minimized when the log probability is maximized, i.e., when the discriminator is highly likely to classify the generated samples as real. The following equation is given below:

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

The generator aims to minimize this loss, encouraging the production of samples that the **discriminator classifies as real** (logD(G(zi)), close to 1.

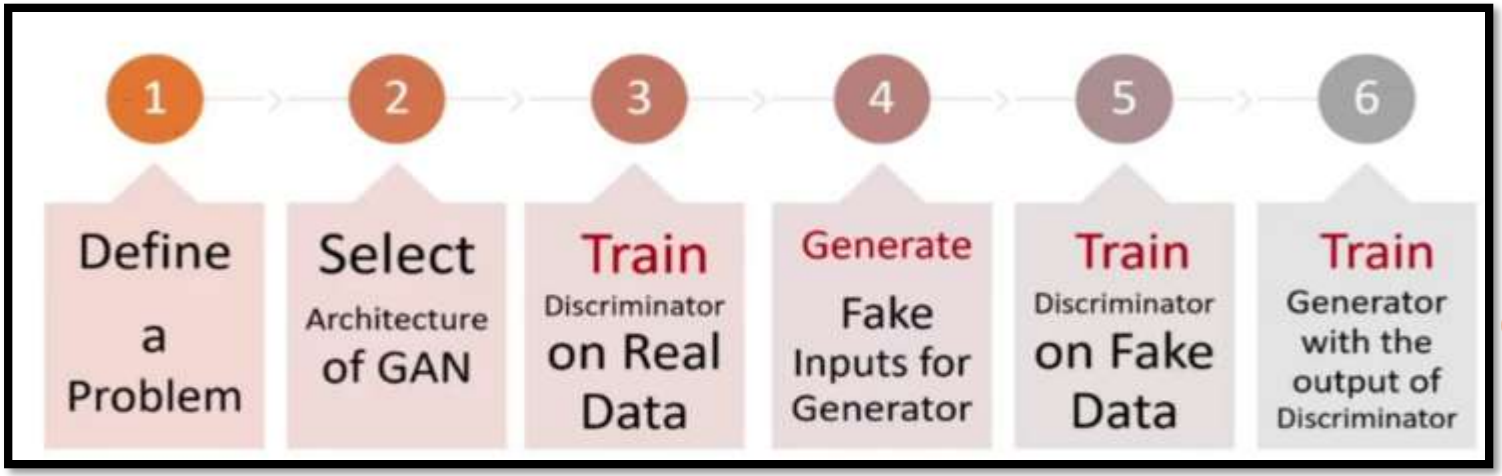
CONTD..

2. Discriminator Loss: The discriminator **reduces t**he **negative log likelihood** of correctly classifying both produced and real samples.

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

Training process of GAN

- Step 1: Define a Problem.
- Step 2: Select Architecture of GAN.
- Step 3: Train Discriminator on Real Dataset.
- Step 4: Train Generator.
- Step 5: Train Discriminator on Fake Data.
- Step 6: Train Generator with the output of Discriminator.



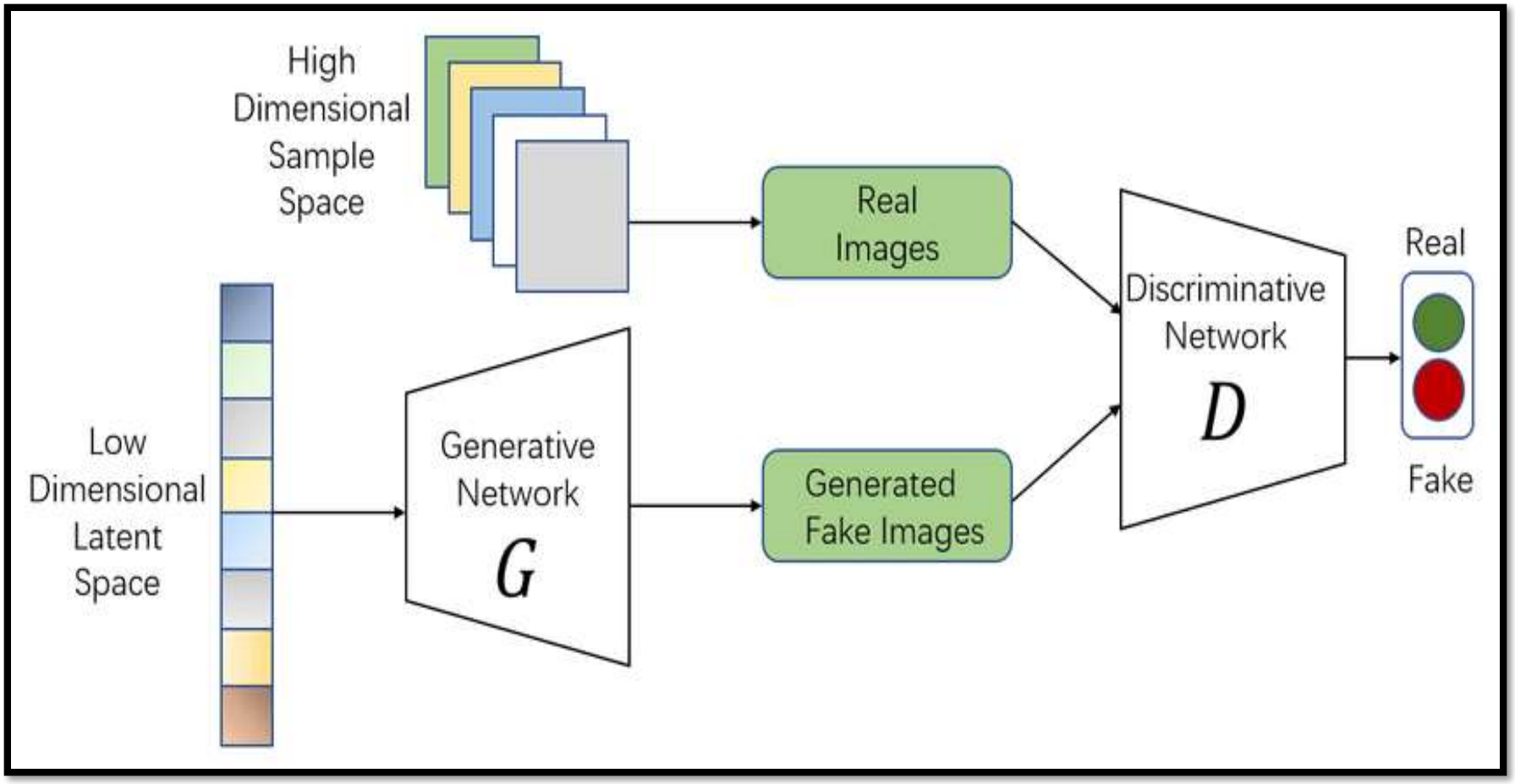
Types of GAN


Vanilla GAN
01
02
Conditional GAN (CGAN)
Deep Convolutional GAN (DCGAN)
03
04
CycleGAN
Generative Adversarial Text to Image Synthesis
05
06
StyleGAN
Super Resolution GAN (SRGAN)
07

CONTD..

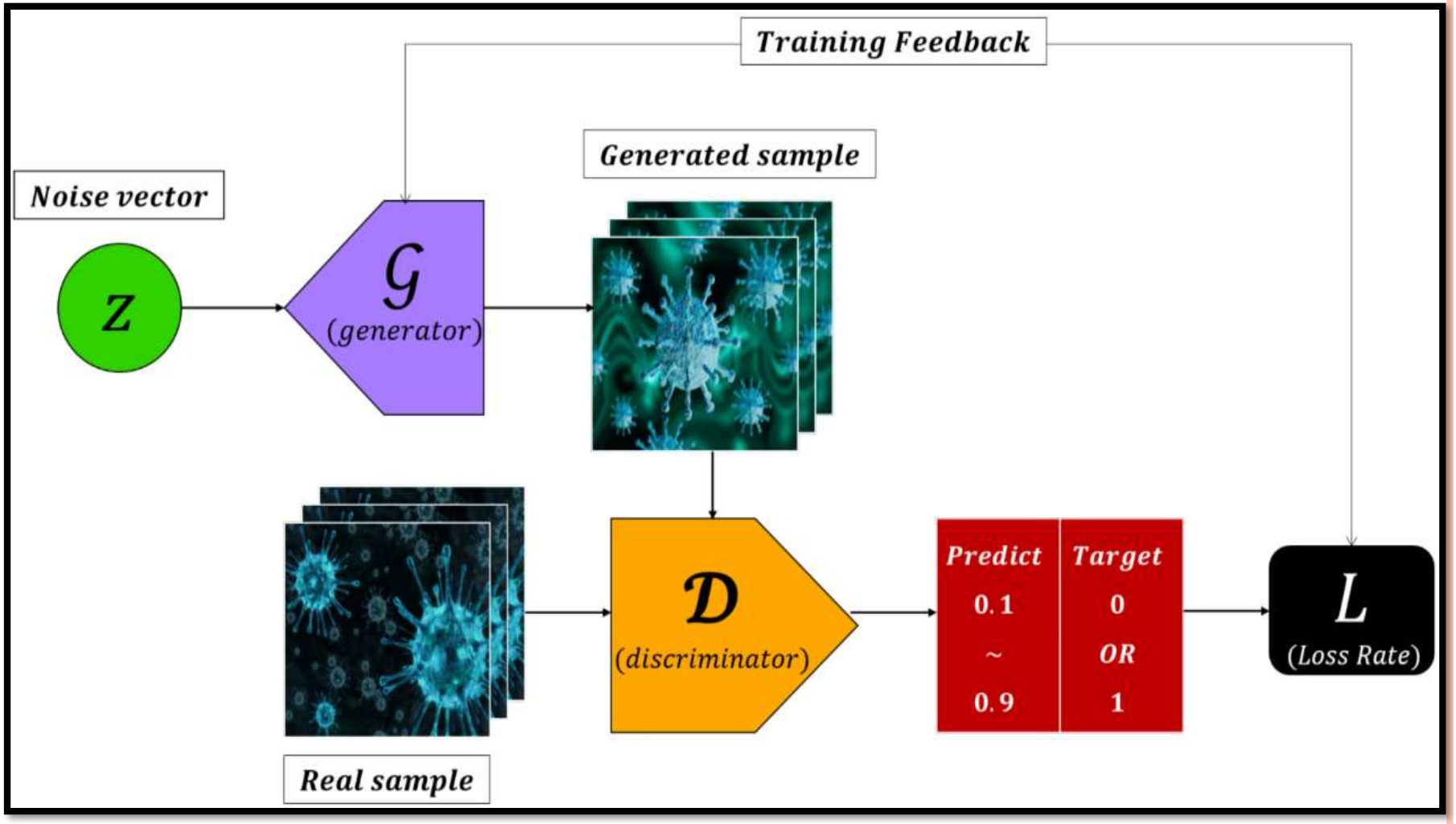
- 1. Vanilla GAN: This is the most basic form of GAN, where both the generator and discriminator are modeled using multi-layer perceptrons.
- The generator focuses on capturing data distribution, while the discriminator evaluates the probability that a given sample is real or synthetic.
- This Vanilla GAN always **tries to optimize** the **mathematical equation** using **stochastic gradient descent.**

CONTD..



ARCHITECTURE OF VANILLA GAN

CONTD..

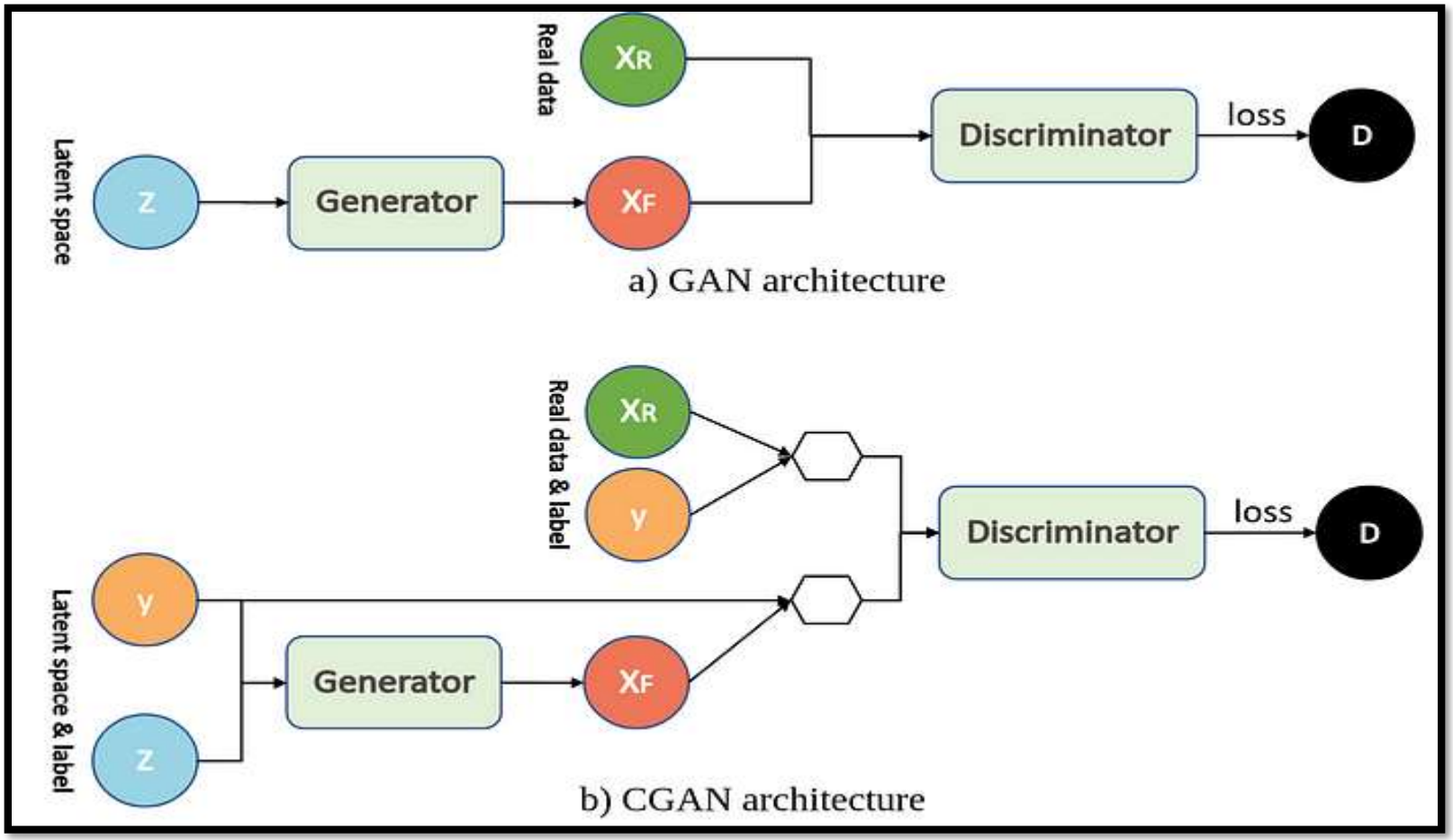


EXAMPLE OF VANILLA GAN

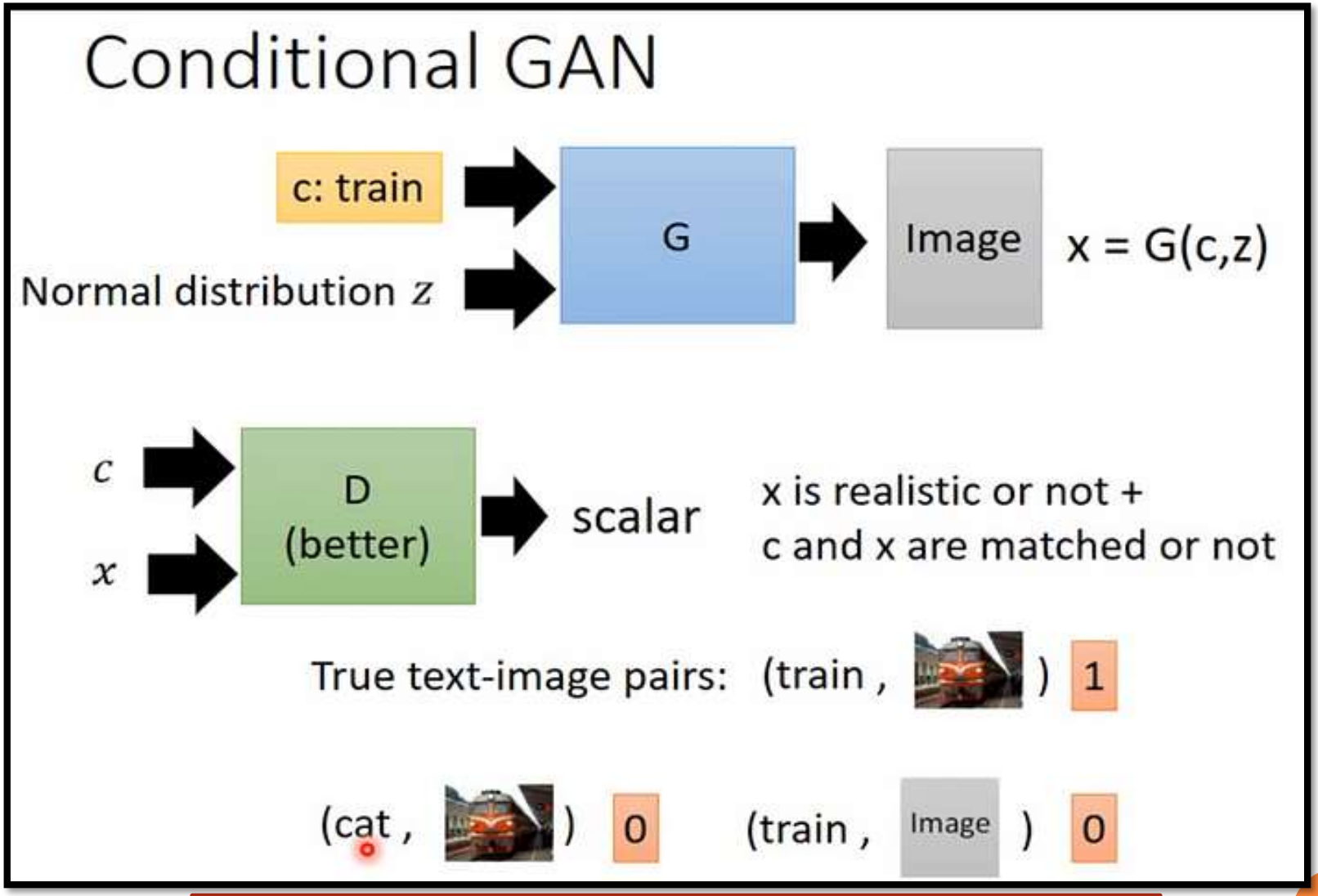
CONTD..

- **2. Conditional GAN:** CGAN can be described as a deep learning method in which some conditional parameters are put into place.
- Here, both networks receive additional information such as class labels, making the model conditional. This allows the generation of data that is more specific to the given condition.
- In CGAN, an additional parameter 'y' is added to the Generator for generating the corresponding data.
- Labels are also put into the input to the Discriminator in order for the Discriminator to help distinguish the real data from the fake generated data.

CONTD..



Architecture of Conditional GAN (CGAN)

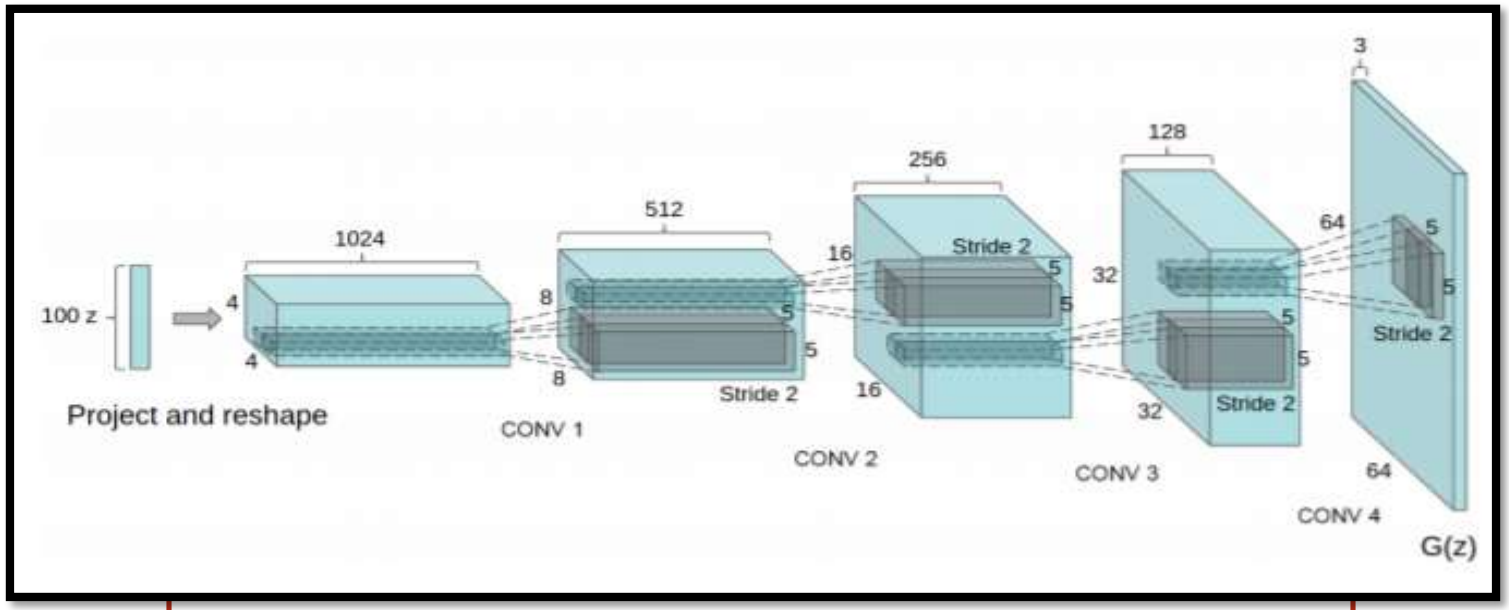


Example of Conditional GAN (CGAN)

CONTD..

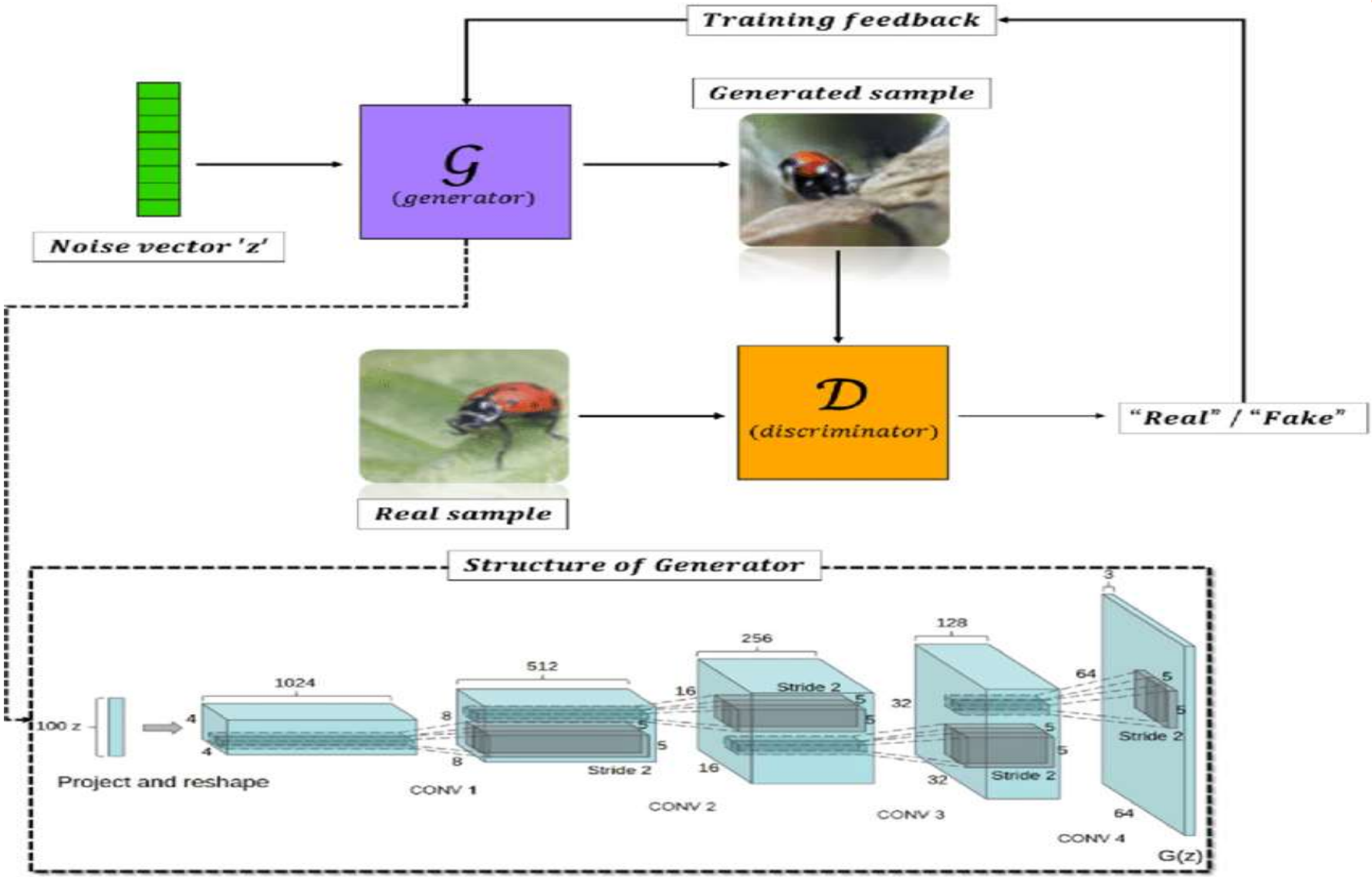
- 3. Deep Convolutional GAN (DCGAN): DCGANs employ convolutional neural networks, making them more effective for tasks that involve image data. They are known for generating high-quality, high-resolution images.
- Deep Convolutional GAN (DCGAN) was proposed by a researcher from MIT and Facebook AI research. It is widely used in many convolution-based generation-based techniques.
- It is composed of ConvNets in place of multi-layer perceptrons.
- DCGAN uses convolutional and convolutional-transpose layers in the generator and discriminator, respectively. It was proposed by Radford.

- Here the discriminator consists of strided convolution layers, and Relu as activation function.. The generator consists of convolutional-transpose layers, and ReLU activations. The output will be a 3x64x64 RGB image.

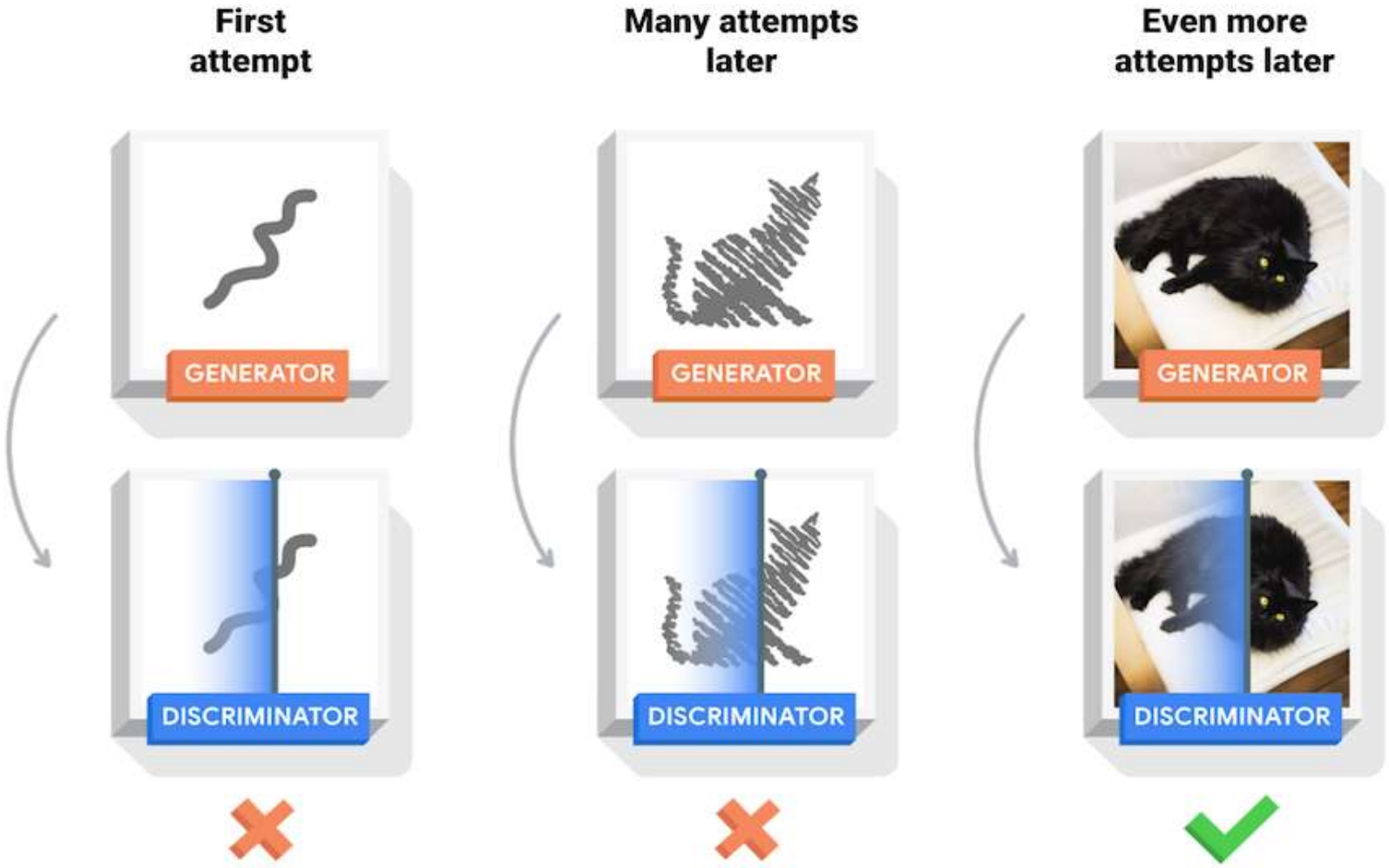


Architecture of Deep Convolution GAN (DCGAN)

CONTD..



Example of DCGAN (DCGAN)



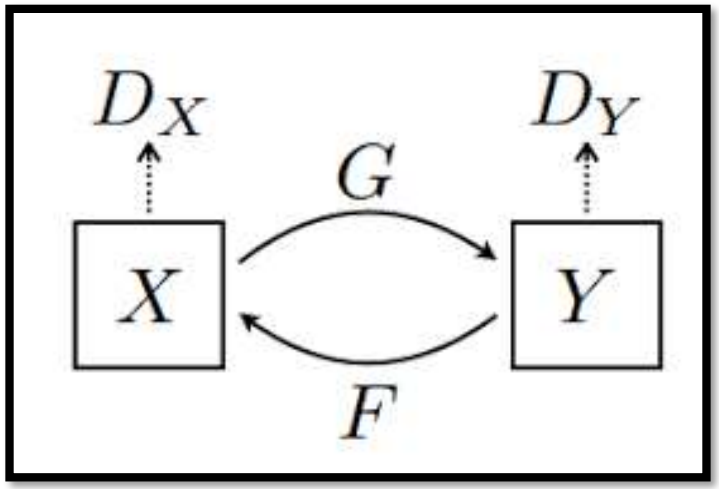
Example of DCGAN (DCGAN)

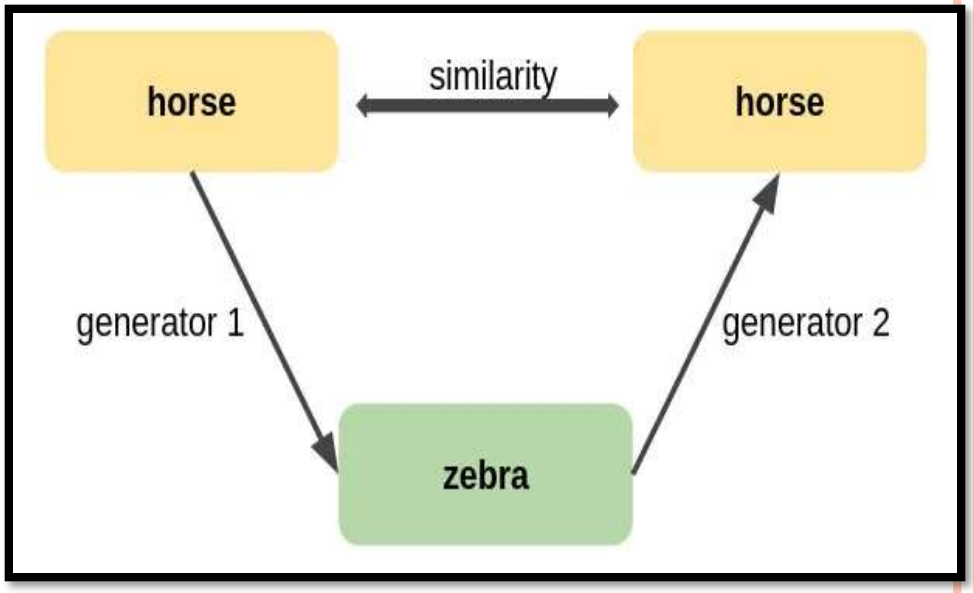
CONTD..

- 4. Cycle GAN: **Cycle GAN** is used to transfer characteristic of one image to another or can map the distribution of images to another.
- In **CycleGAN** we treat the problem as an image reconstruction problem. We first take an image input (x) and using the generator G to convert into the reconstructed image.
- Then we reverse this process from **reconstructed image to original image using a** generator F.
- Then we calculate the mean squared error loss between real and reconstructed image.

CONTD..

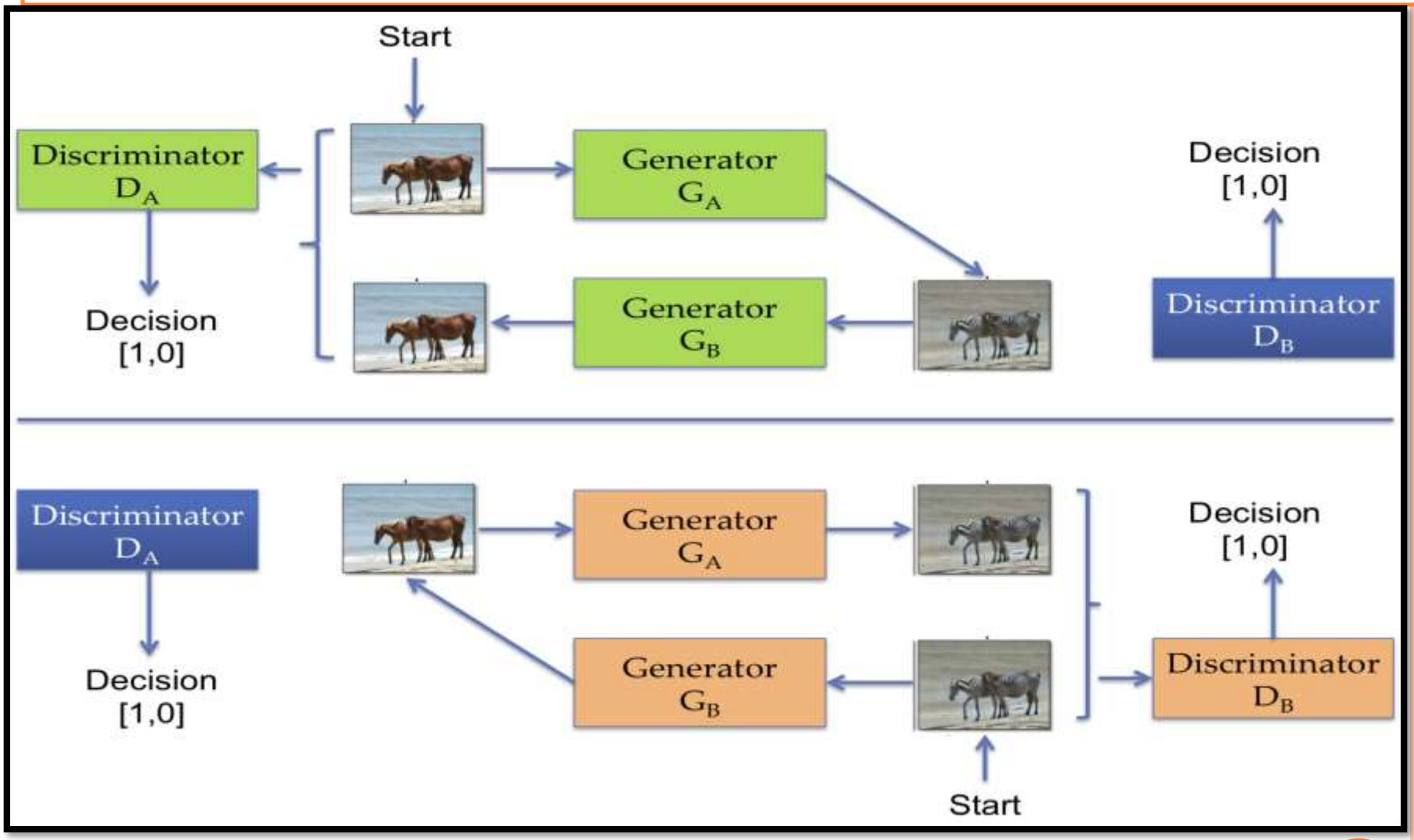
- The most important feature of this cycle_GAN is that it can do this image translation on an unpaired image where there is no relation exists between the input image and output image.





Architecture of Cycle GAN (CGAN)

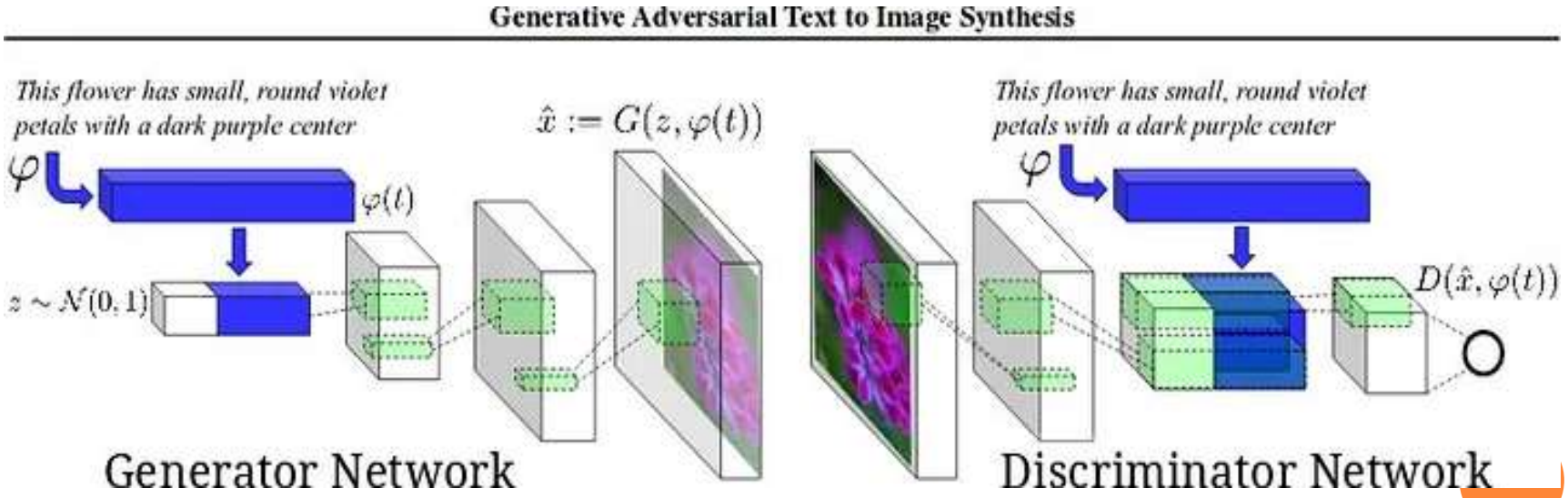
CONTD..



Architecture of Cycle GAN (CGAN)

CONTD..

- 5. Generative Adversarial Text to Image Synthesis: Text to image synthesis (T2I) is one of the most challenging and interesting tasks in the modern domain of Computer Vision. These GANs can generate images from textual descriptions, bridging the gap between natural language and visual data.

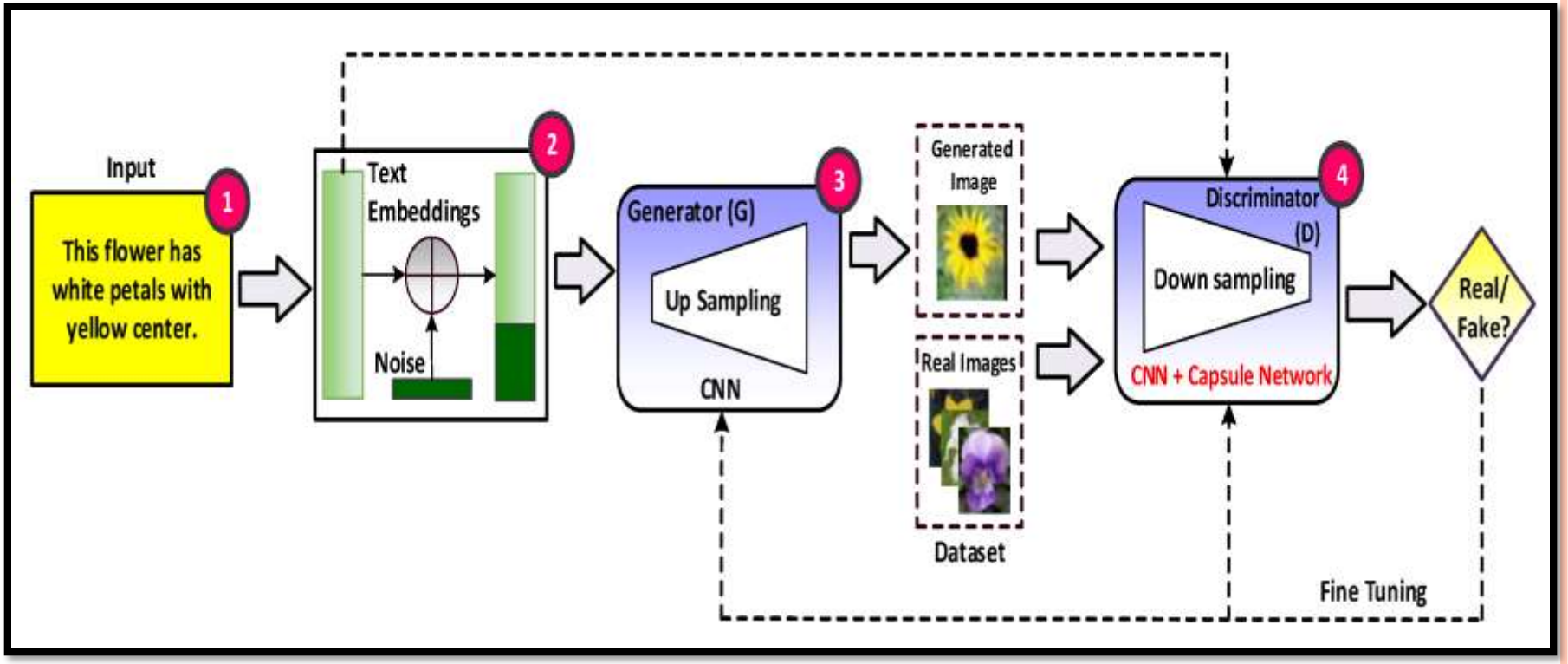


Architecture of Text to Image Synthesis

CONTD..

Example

CONTD..

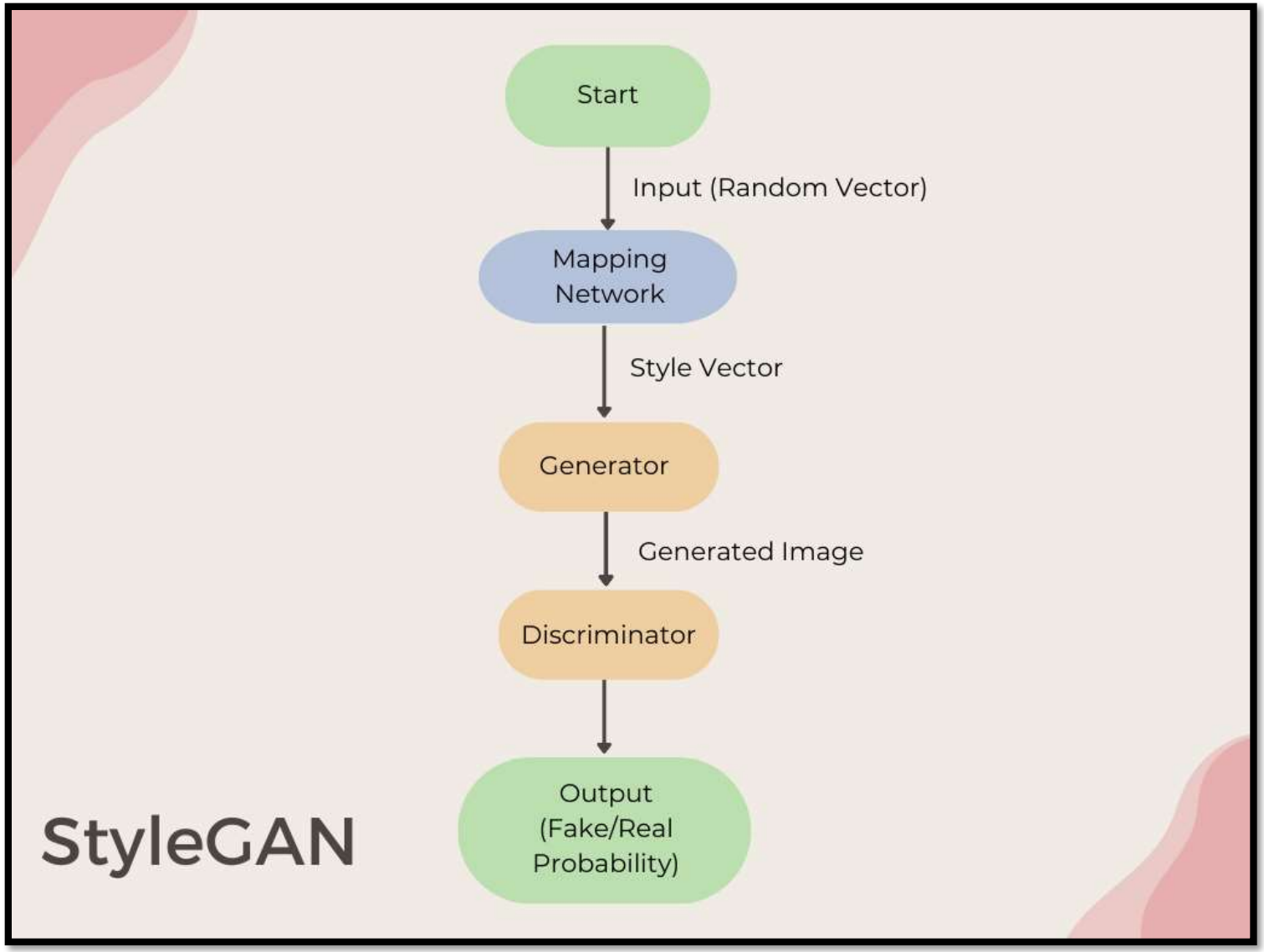


Example

CONTD..

- 6. Style GAN: **Style GAN** proposes a lot of changes in the generator part.
- StyleGAN was designed to create realistic images while manipulating and controlling certain features or styles of the image. These styles associated with the generated images could be features like color, texture, pose, etc.

CONTD..



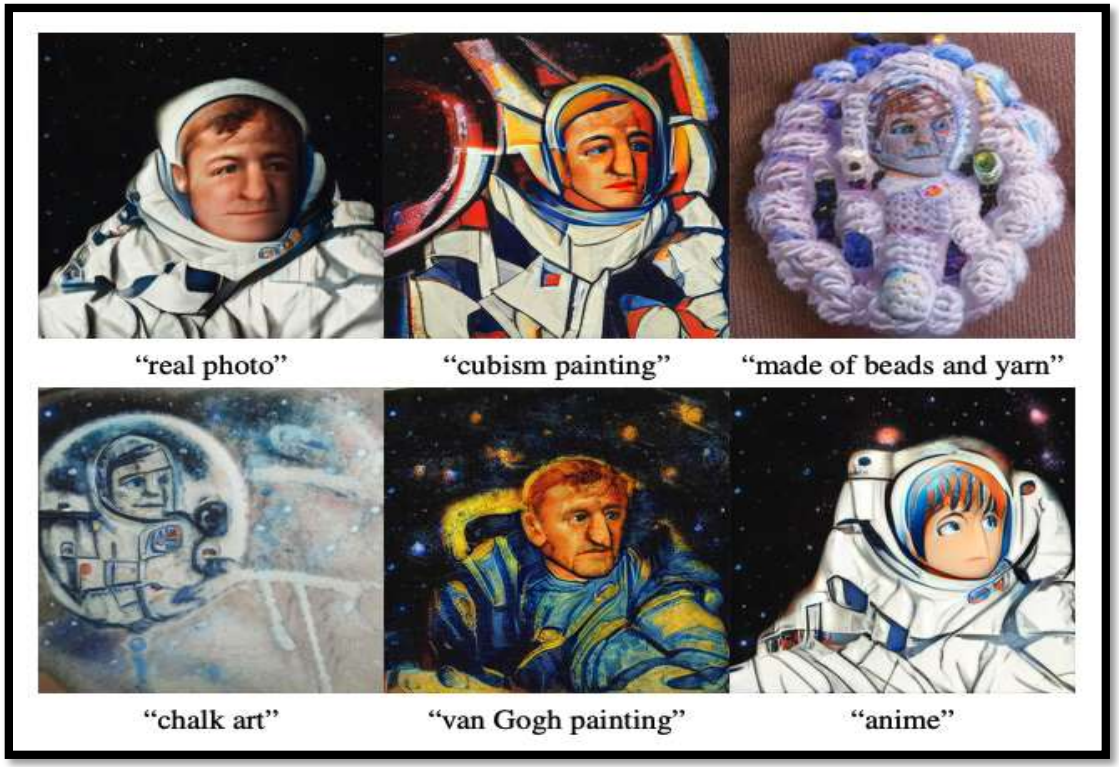
Architecture of Text to Image Synthesis

CONTD..

- First, StyleGAN takes a random vector as an input. This vector is mapped into a style vector representing different aspects of the image's style and appearance.
- The generator network then generates an image using the style vector.
- The discriminator network evaluates whether the generated image is real or fake (generated).

CONTD..

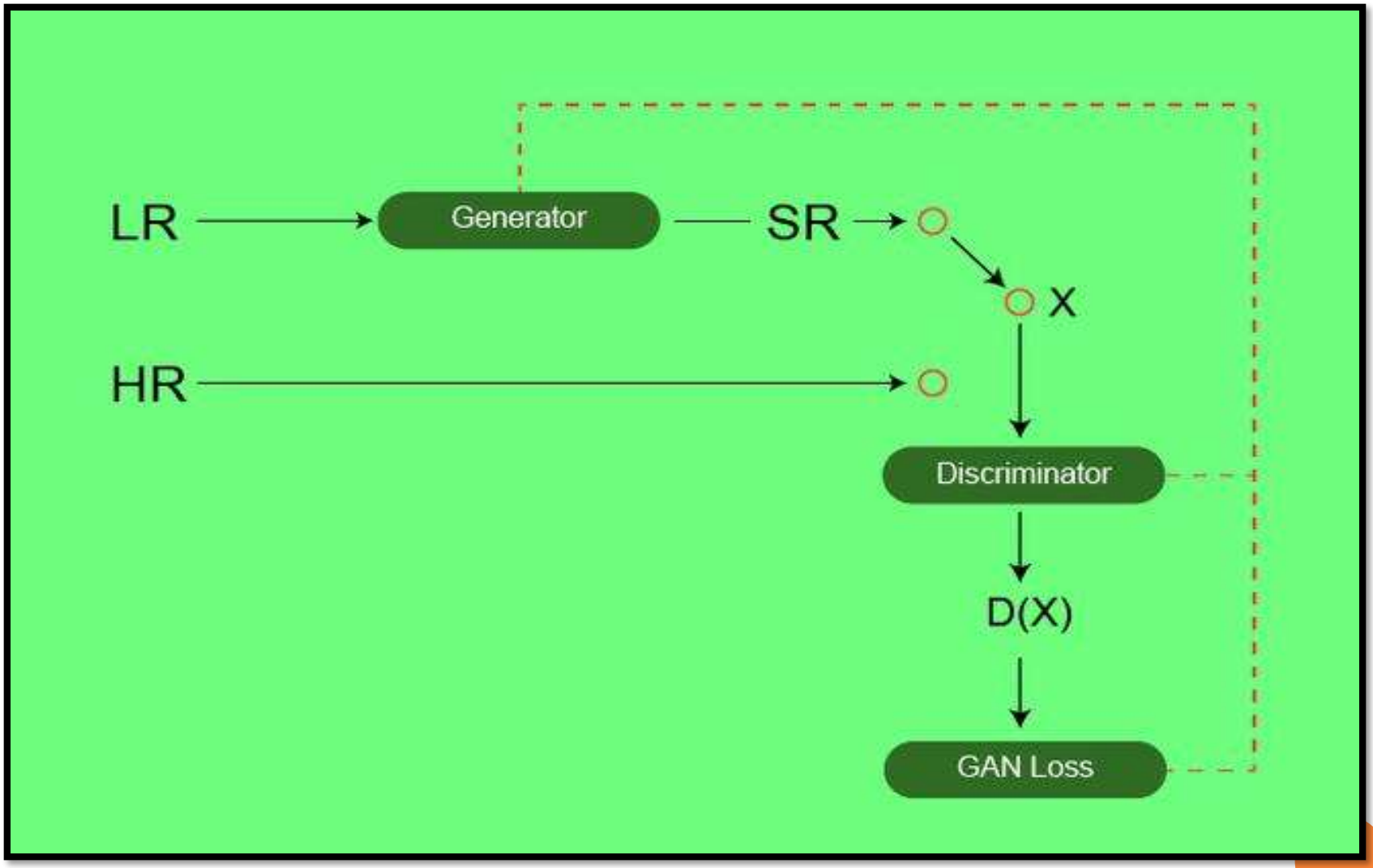

Content image
Style image
Output image


"real photo"
"cubism painting"
"made of beads and yarn"
"chalk art"
"van Gogh painting"
"anime"

CONTD..

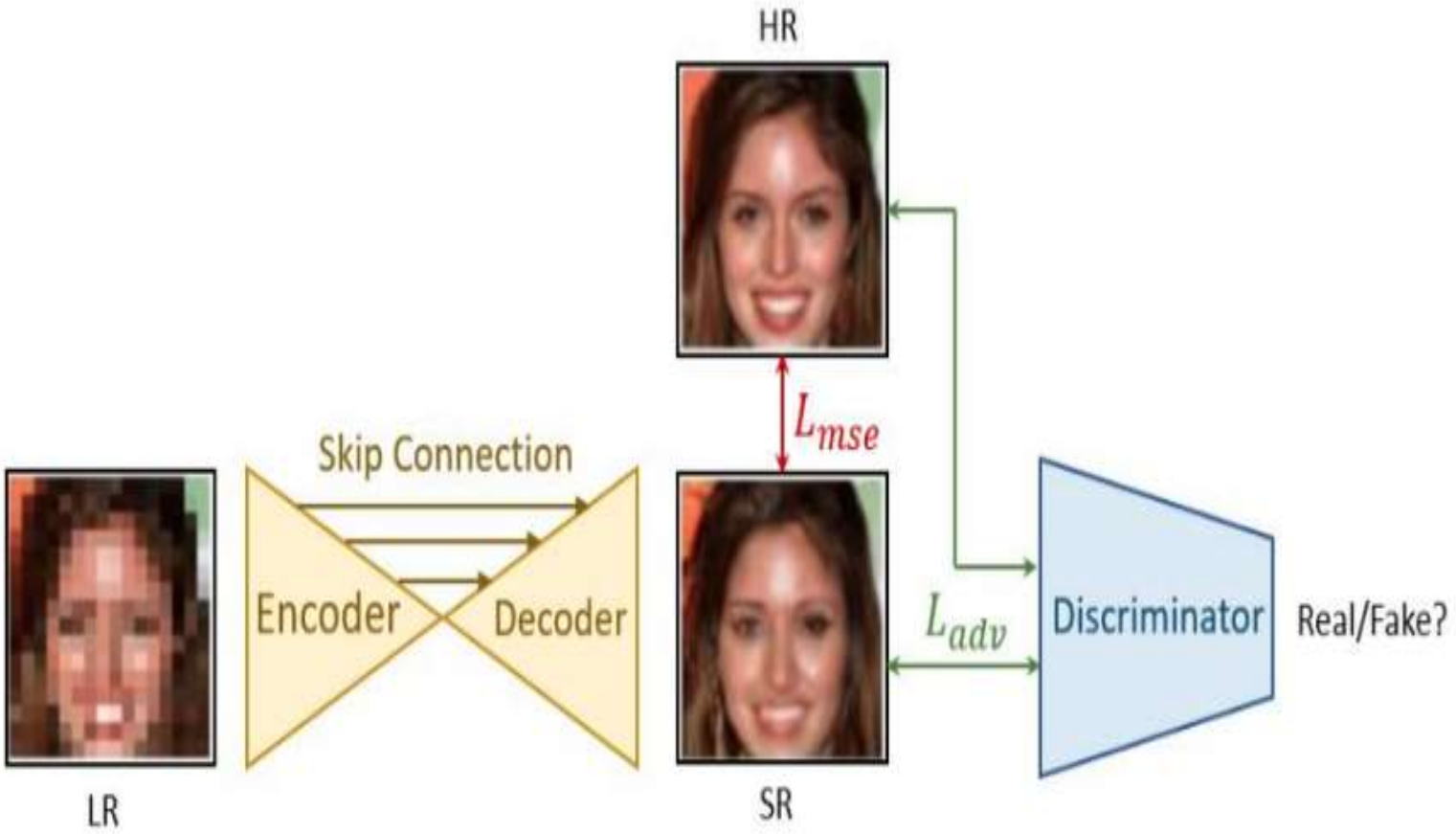
- 7. Super Resolution GAN (SRGAN): SRGAN enhances the resolution of images, turning low-resolution inputs into high-resolution outputs without losing detail.
- SRGAN was proposed by researchers at Twitter. The motive of this architecture is to recover finer textures from the image when we upscale it so that it's quality cannot be compromised.

CONTD..



Architecture of Super Resolution GAN

CONTD..



Example of Super Resolution GAN

Applications of GAN

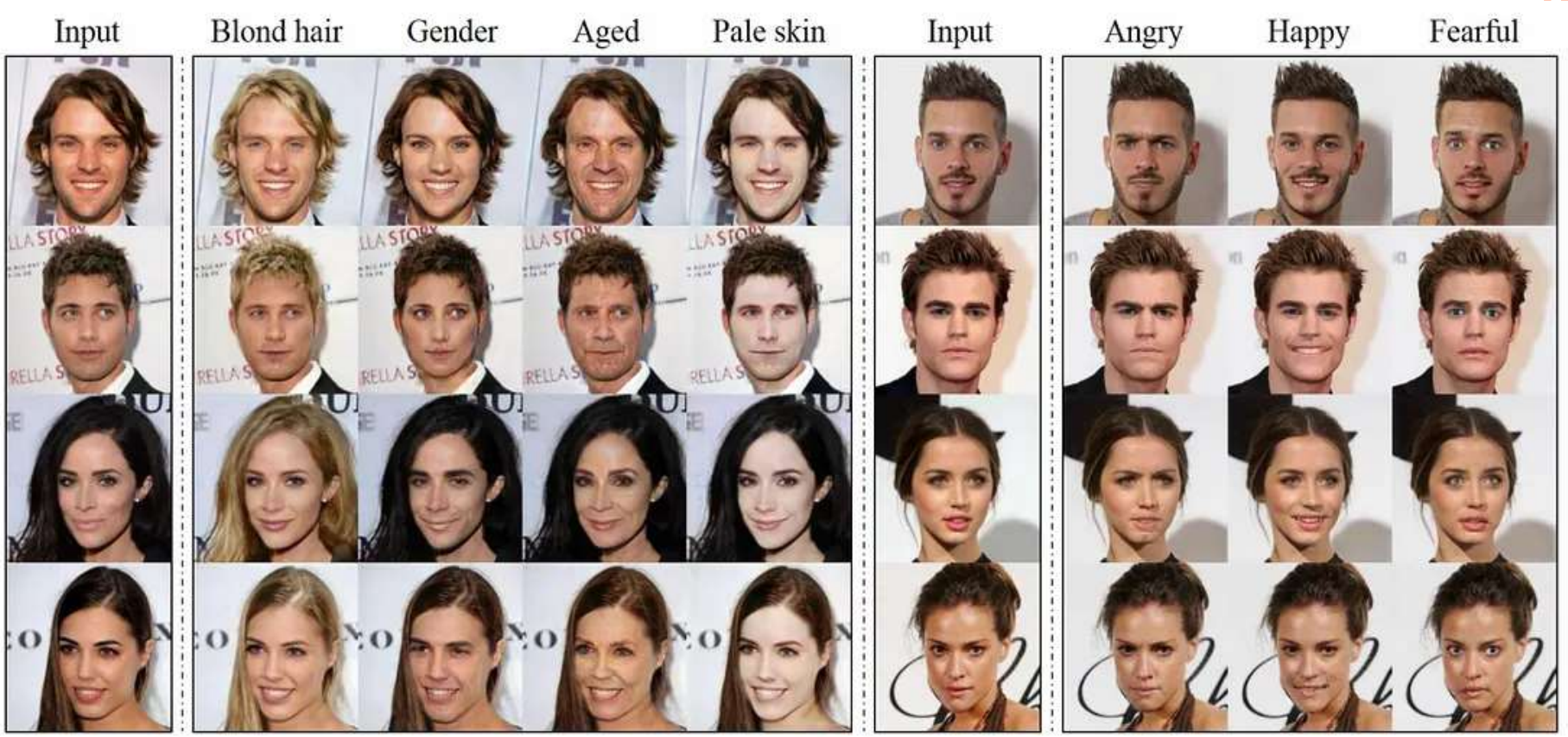
- NVIDIA research center has to develop these GAN Applications in real time.

- They generate high quality and photo realistic Images, Videos
- **1. Image to Image Translation:** GANs Can be in use for translating data from images. In image-to-image translations, GANs account for tasks such as:
  - Changing sketches to color photographs.
  - Converting satellite images to google maps.
  - Translation of photos from day to night and vice versa.
  - Translation of black and white photographs to color.

CONTD...

CONTD..

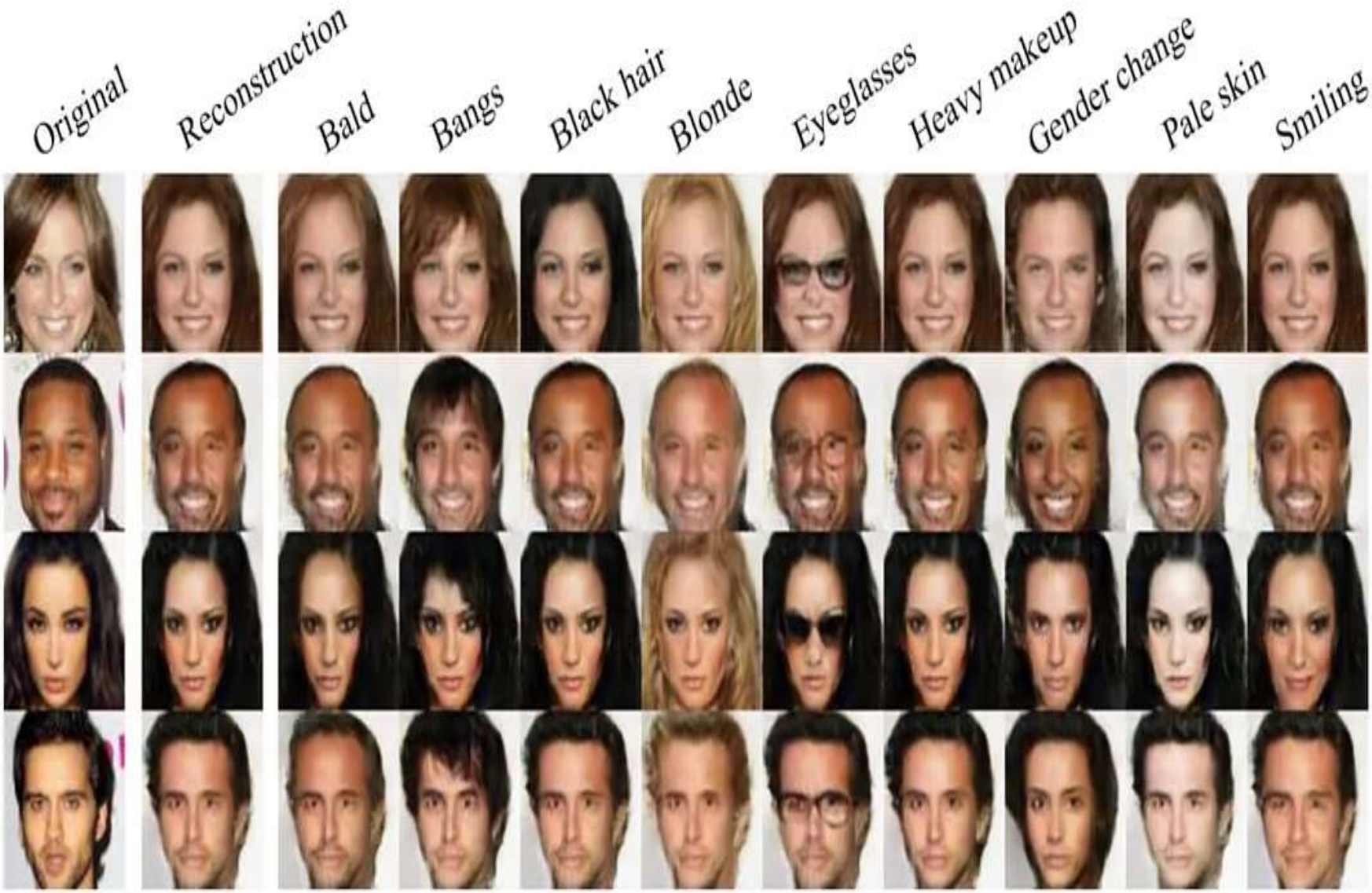
- StarGAN is an image-to-image translation for one domain to another. For example, given a happy face, we want to transform it into a fearful face.



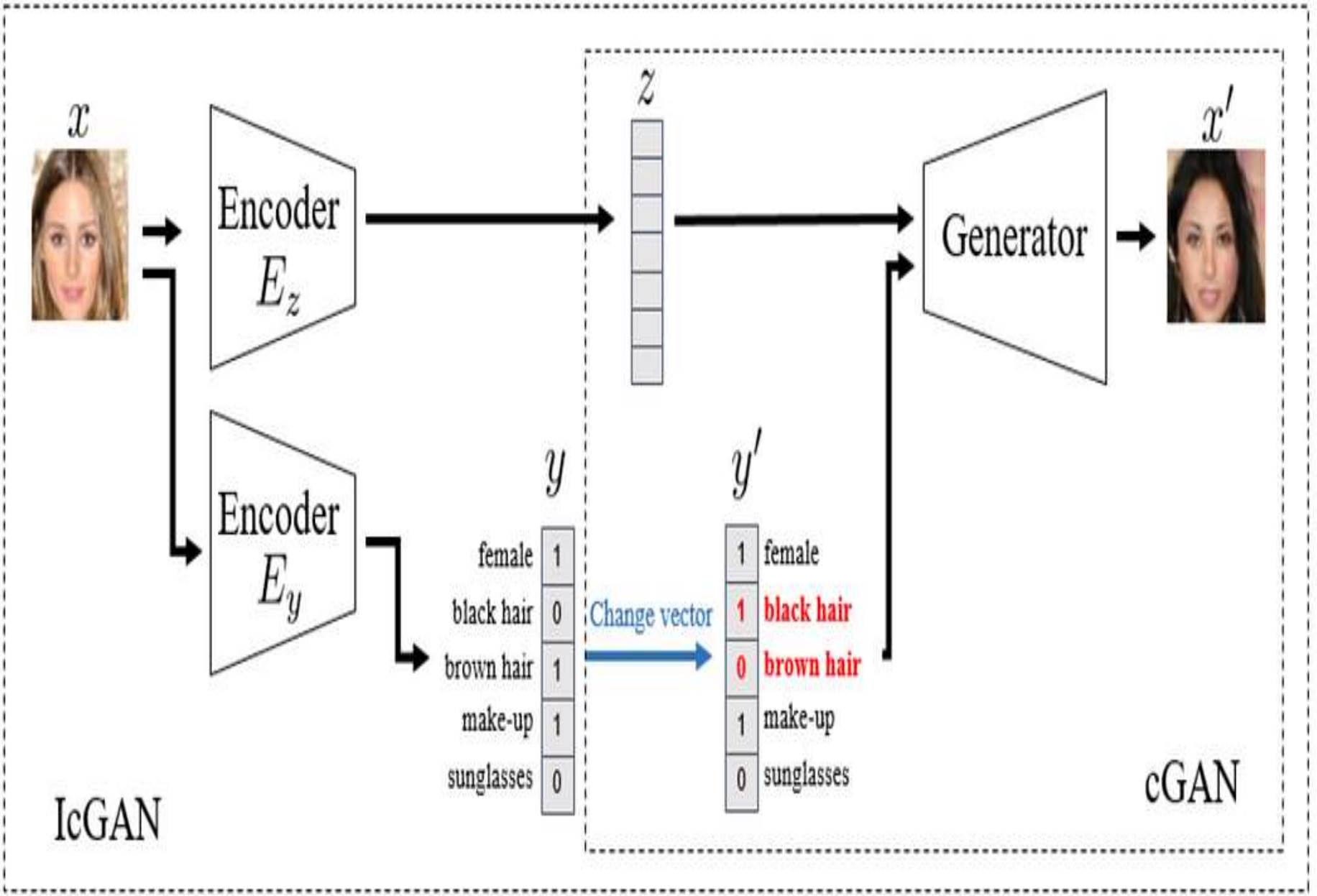
CONTD..

- **2. Image Editing:** Most image editing software these days don't give us much flexibility to make creative changes in pictures.
- **For example,** let's say you want to change the appearance of a 90-year-old person by changing his/her hairstyle. This can't be done by the current image editing tools out there. But guess what? Using GANs, **we can** reconstruct images **and** attempt to change the appearance drastically.

CONTD..


Original
Reconstruction
Bald
Bangs
Black hair
Blonde
Eyeglasses
Heavy makeup
Gender change
Pale skin
Smiling

CONTD..


x
Encoder
E_z
z
Generator
x′
Encoder
E_y
y
y′
female 1
black hair 0
brown hair 1
make-up 1
sunglasses 0
Change vector
1 female
1 black hair
0 brown hair
1 make-up
0 sunglasses
IcGAN
cGAN

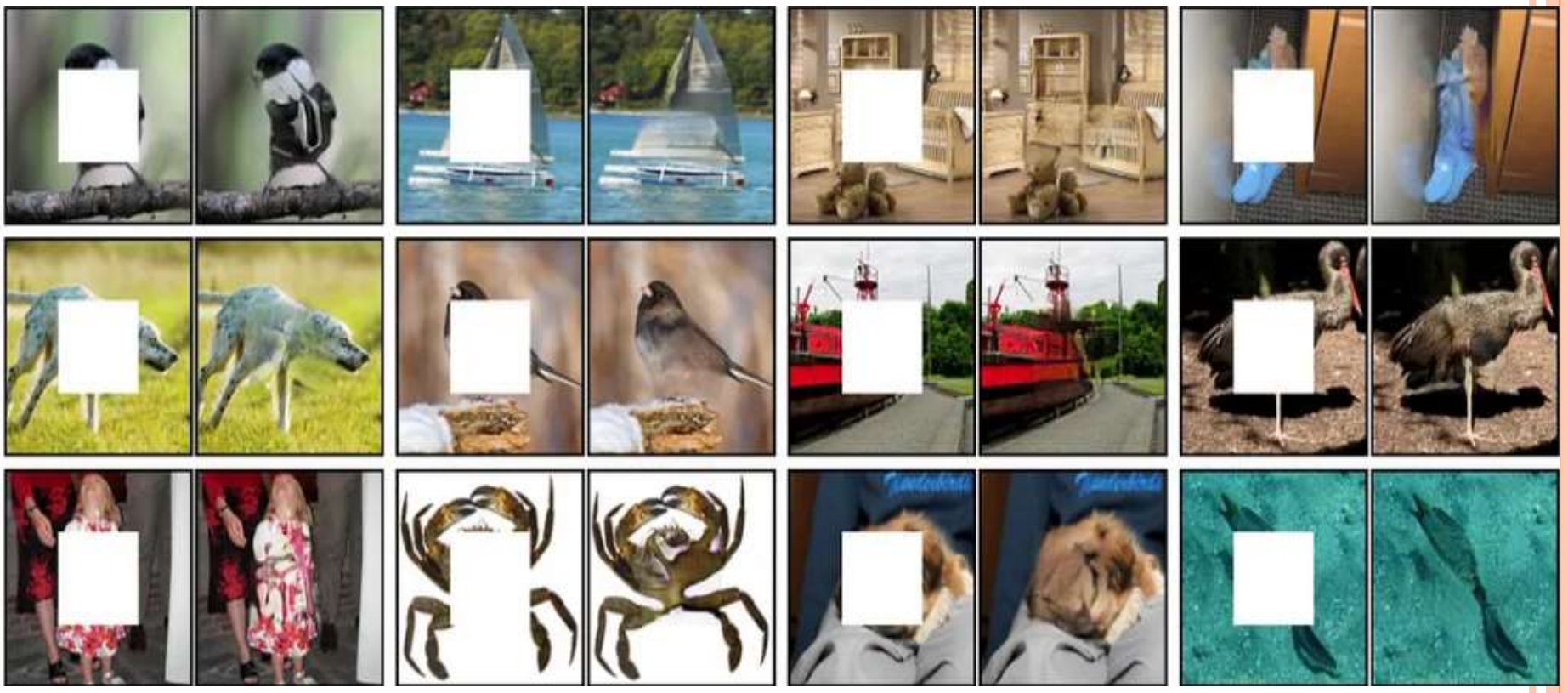
CONTD..

- **4. Face Synthesis:** Synthesis faces in different poses: With a single input image, we create faces in different viewing angles.
- For example, we can use this to transform images that will be easier for face recognition.

. Synthesis results under various illuminations. The first row is the synthesized image, the second row is the input.

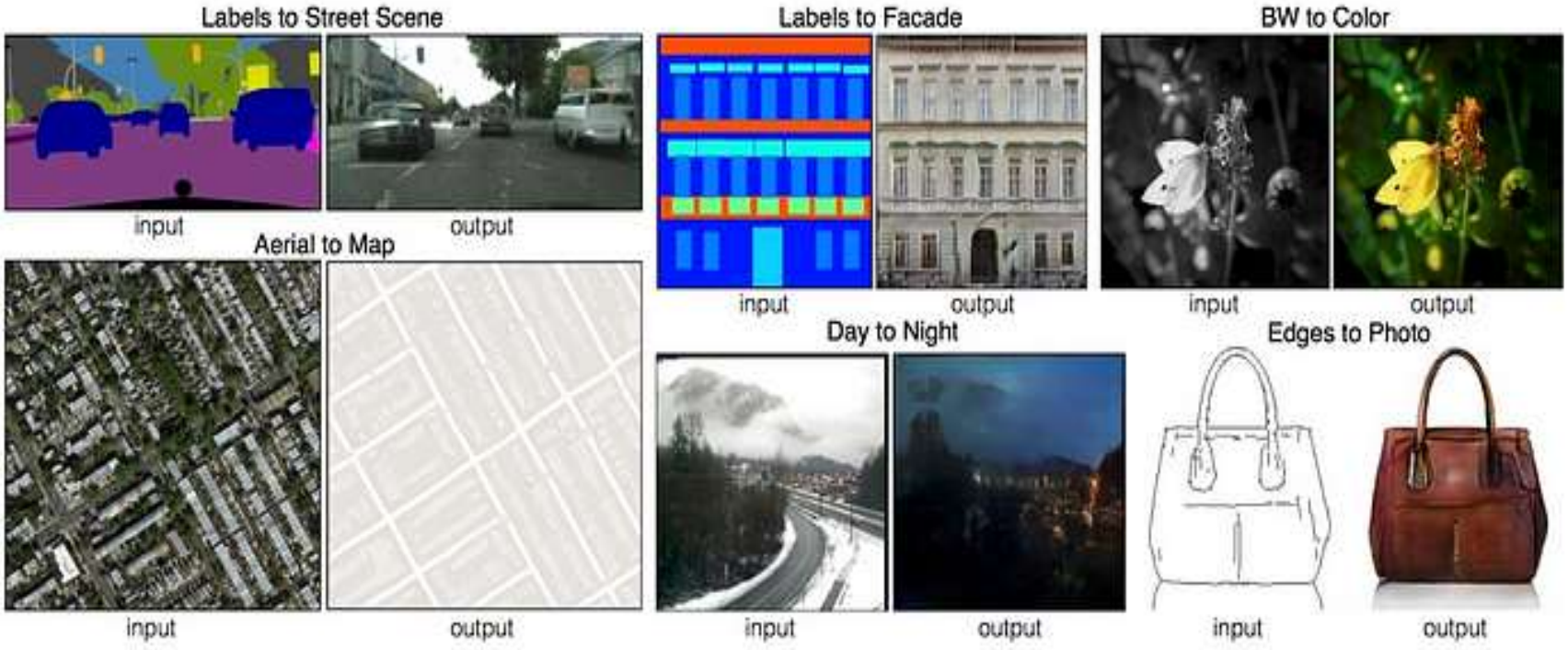
CONTD..

- **5. Image Painting: Repair images** have been an important subject decades ago. **GAN is used to repair images** and **fill the missing part** with **created "content".**



CONTD..

- **6. Pix to Pix:** Pix2Pix is an **image-to-image translation** that get quoted in **cross-domain GAN**'s frequently. For example, it converts a satellite image into a map (the bottom left).



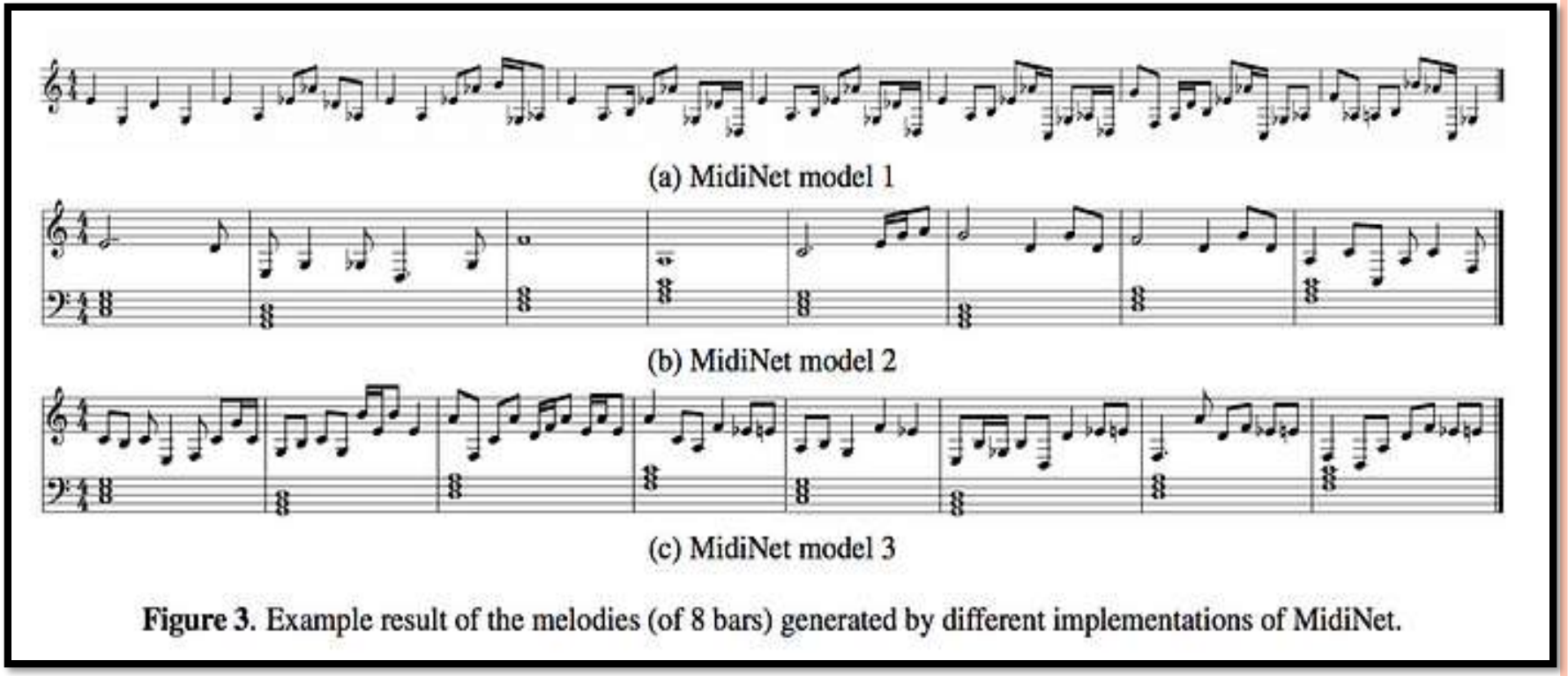
CONTD..

- **DeblurGAN** performs **motion deblurring.**

Figure 2: GoPro images [25] processed by DeblurGAN. Blurred – left, DeblurGAN – center, ground truth sharp – right.

CONTD..

- **7.Music Generation:** GAN can be applied to non-image domain, like composing music.



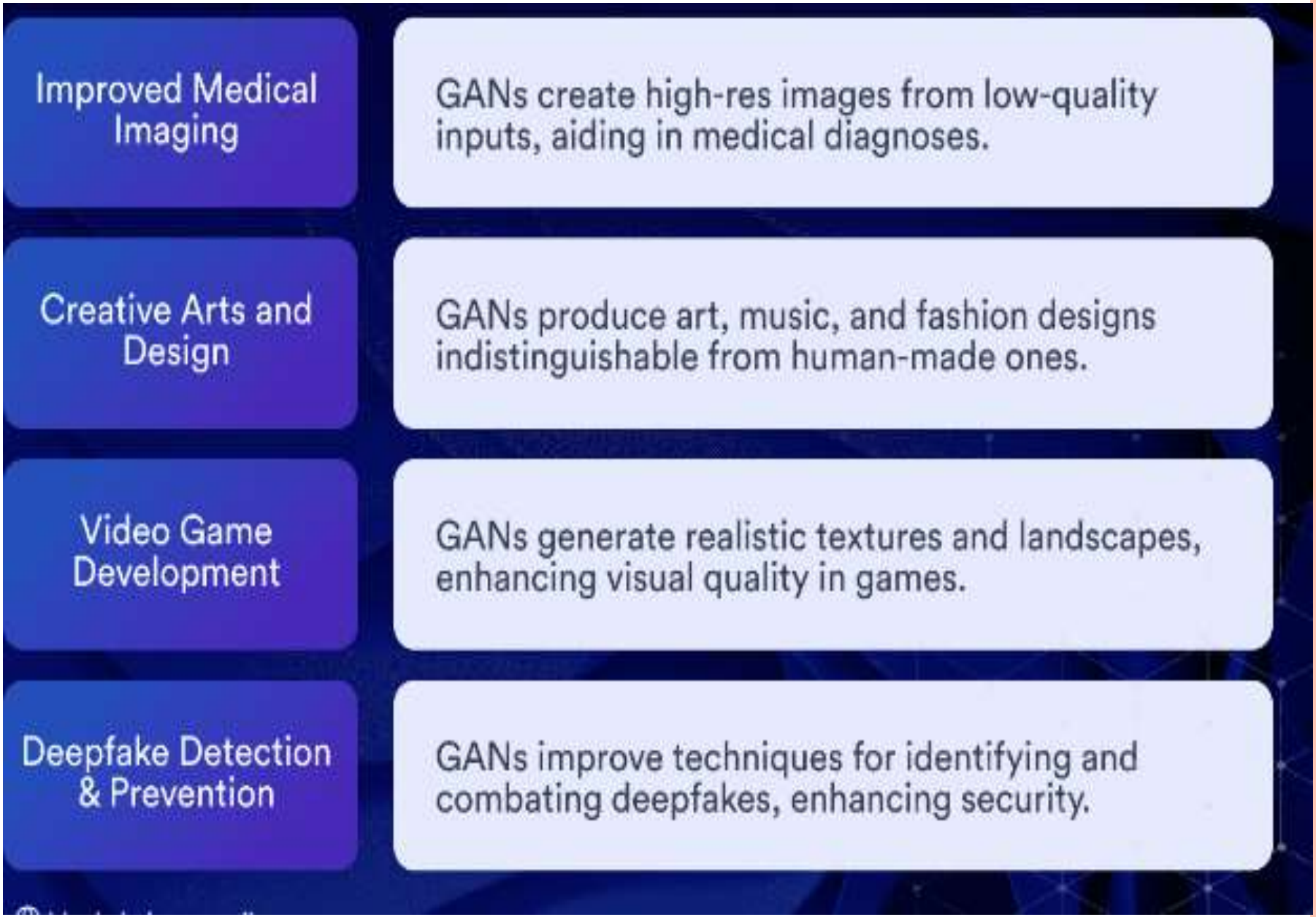
(a) MidiNet model 1

(b) MidiNet model 2

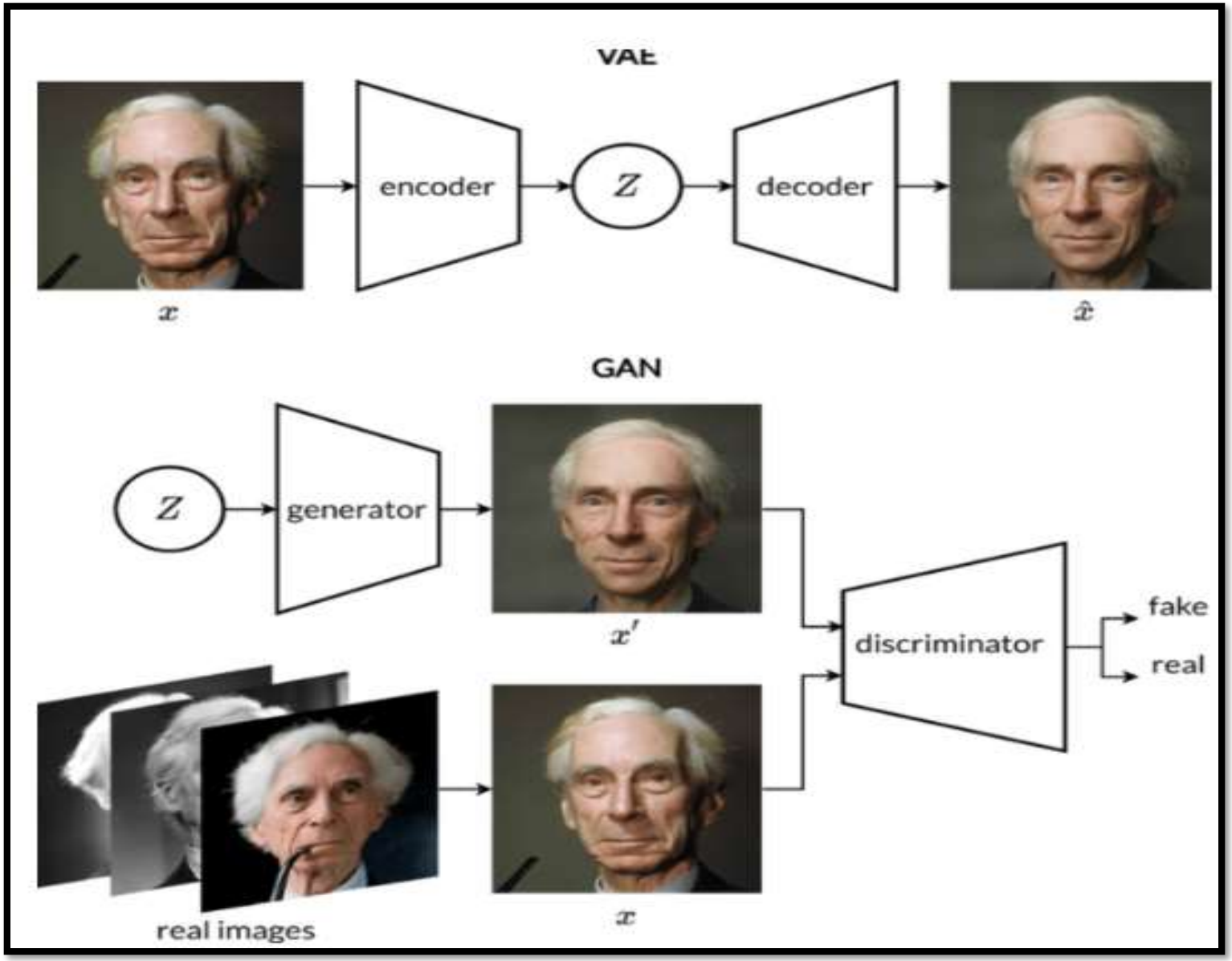
(c) MidiNet model 3

Figure 3. Example result of the melodies (of 8 bars) generated by different implementations of MidiNet.

Future Generations of GANs



Differences b/w VAN and GAN



CONTD..

Topics	Generative Adversarial Networks	Variational Autoencoder
Functionality	Composed of two models **(a generator and a discriminator)** that compete with each other. The **generator creates fake samples** and the discriminator attempts to distinguish between **real and fake samples.**	Composed of an **encoder and a decoder**. The **encoder maps inputs to a latent space,** and the **decoder maps points in the latent space back to the input space.**
Output Quality	Can **generate high-quality, realistic outputs.** Known for **generating images that are hard to distinguish from real ones**.	Generally produces **less sharp or slightly blurrier images compared to GANs**. However, this may depend on the specific implementation and problem domain.
Training Stability	Training GANs can be **challenging and unstable,** due to the **adversarial loss used in training.**	Generally **easier and more stable to train** because they use a **likelihood-based objective function.**

Thank You