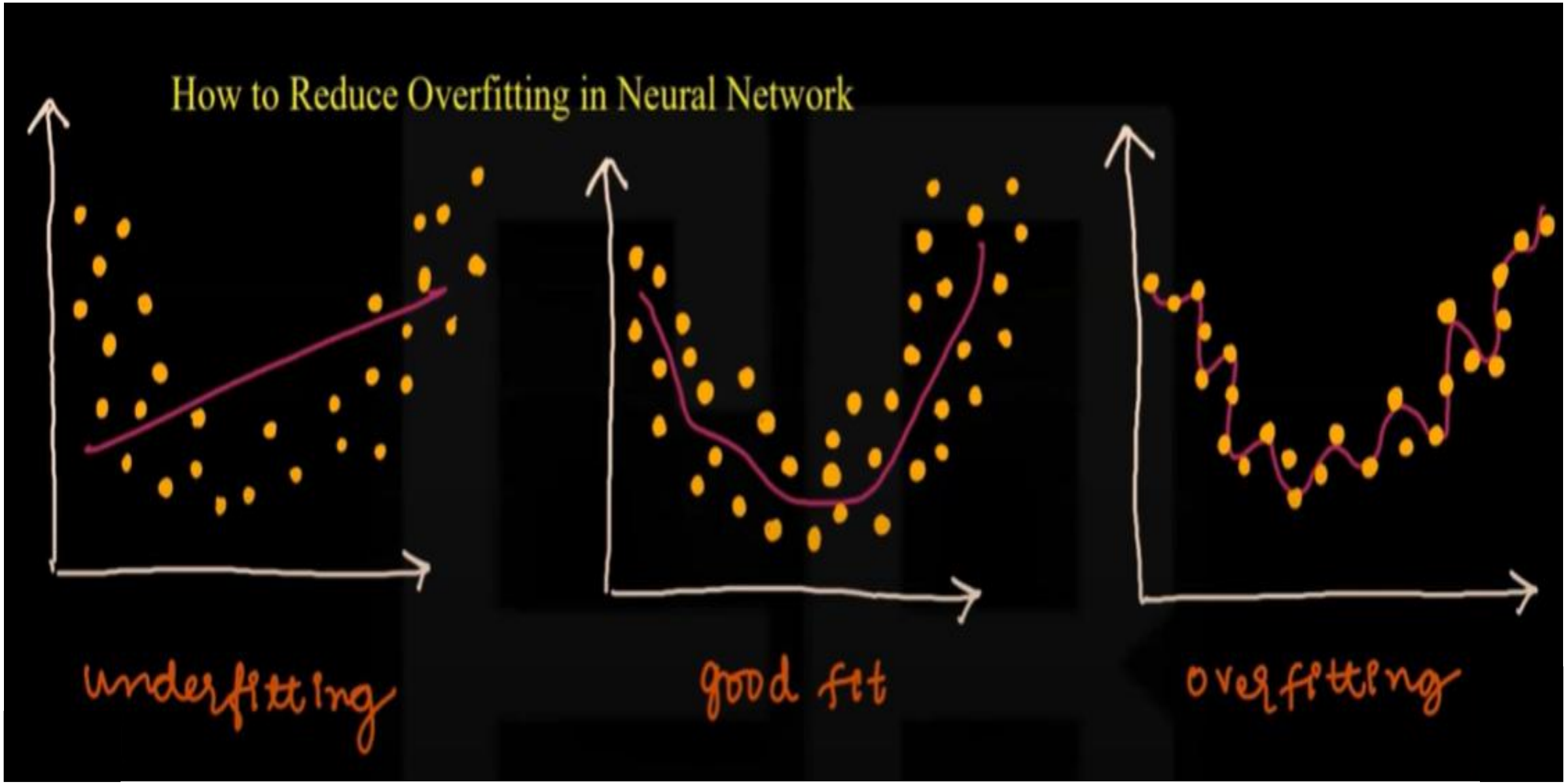
DEEP LEARNING


How to Reduce Overfitting in Neural Network
underfitting
good fit
overfitting

Overfitting: Overfitting in machine learning occurs when a model learns the training data too precisely, capturing noise and failing to generalize to new data. It results in high training accuracy but poor performance on unseen data, highlighting the need for regularization techniques to promote generalization.

Regularization:

- Description: Introduce regularization terms (L1 or L2) into the loss function to penalize large weights. This discourages overly complex models.
- Implementation: Commonly applied with techniques like L1 regularization (Lasso) or L2 regularization (Ridge) during model training.

Data Augmentation:

- Description: Generate additional trainirg samples by applying random transformations (rotation, scaling, flipping) to existing data. This increases diversity and helps the model generalize better.
- Implementation: Commonly used in computer vision tasks, like image classification, to artificially expand the dataset.

Dropout:

- Description: During training, randomly deactivate a proportion of neurons in each layer, preventing the model from relying too heavily on specific features. This enhances generalization.
- Implementation: Implemented as a layer in neural networks, typically by specifying a dropout rate.

Simpler Models:

- Description: Choose a less complex model with fewer parameters or shallower architectures. This reduces the risk of fitting noise in the data.
- Implementation: Consider using simpler models, such as linear models or shallow decision trees, depending on the problem.

Early Stopping:

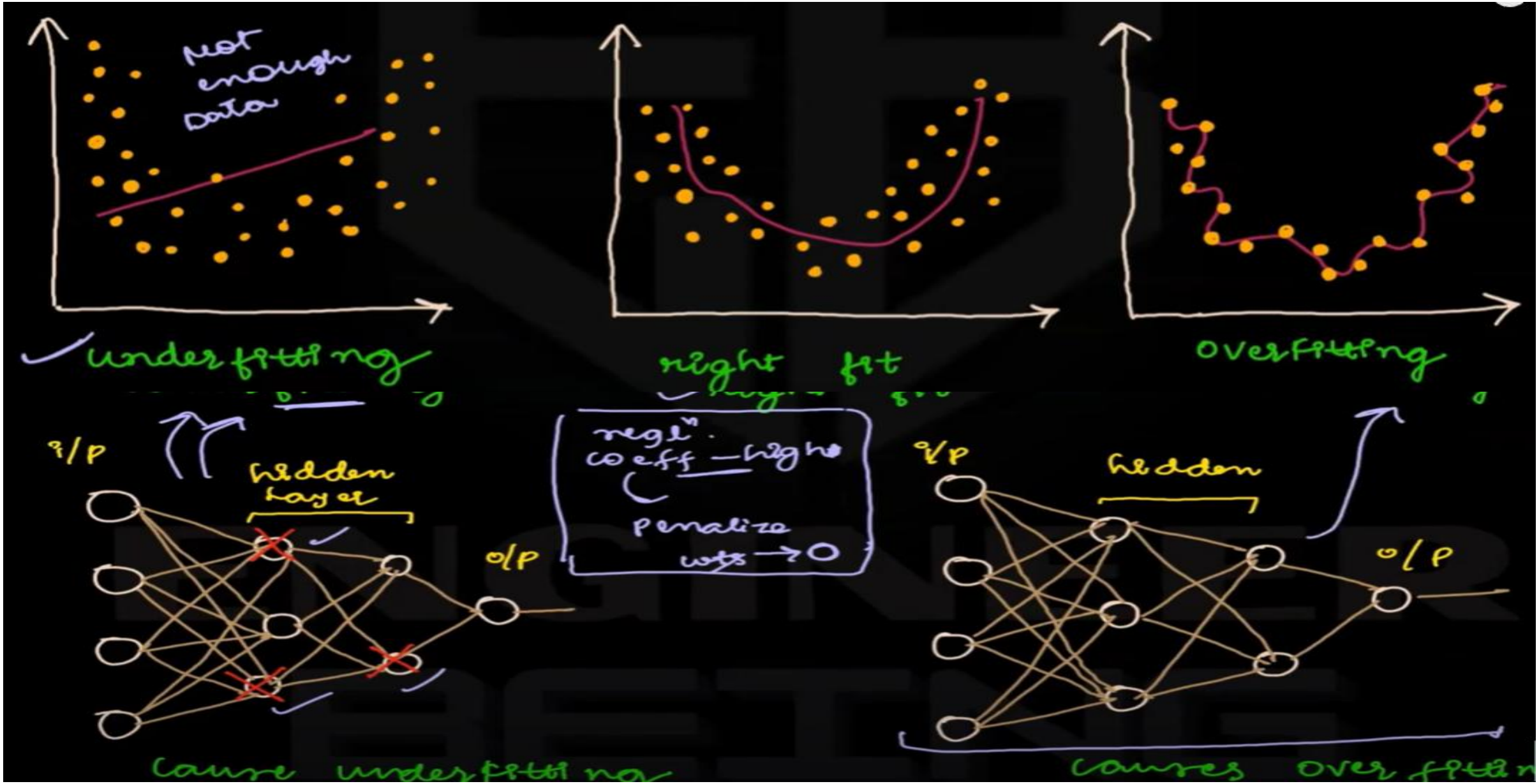
- Description: Monitor the performance of a validation set during training. Stop training when the performance on the validation set starts to degrade, preventing overfitting.
- Implementation: Use a patience parameter to determine the number of epochs without improvement before stopping.

Feature Selection:

- Description: Choose a subset of relevant features and exclude irrelevant or redundant ones. Reducing the number of features can simplify the model.
- Implementation: Various feature selection techniques, like recursive feature elimination or feature importance from tree-based models, can be employed.
- These strategies collectively aim to balance model complexity and generalization, addressing overfitting issues in machine learning models. The choice of specific techniques depends on the nature of the problem and the characteristics of the dataset.

Regularization

- Regularization is a technique used in machine learning and deep learning to prevent overfitting and improve the generalization performance of a model. It involves adding a penalty term to the loss function during training.
- In deep learning, it penalizes the weight matrices of the nodes.
- Assume that our regularization coefficient is so high that some of the weight matrices are nearly equal to zero.


Not enough Data
Underfitting
Right fit
Overfitting
i/p
Hidden layer
o/p
regl. coeff — high
Penalize wts → 0
i/p
Hidden
o/p
Cause underfitting
Causes overfitting

- This will result in a much simpler linear network and slight underfitting of the training data.
- Such a large value of the regularization coefficient is not that useful. We need to optimize the value of regularization coefficient in order to obtain a well-fitted model.

L2 & L1 regularization

L1 and L2 are the most common types of regularization. These update the general cost function by adding another term known as the regularization term.

Cost function = Loss (say, binary cross entropy) + Regularization term

Due to the addition of this regularization term, the values of weight matrices decrease because it assumes that a neural network with smaller weight matrices leads to simpler models. Therefore, it will also reduce overfitting to quite an extent.
However, this regularization term differs in L1 and L2.

In L1, we have:

In this, we penalize the absolute value of the weights. Unlike L2, the weights may be reduced to zero here. Hence, it is very useful when we are trying to compress our model. Otherwise, we usually prefer L2 over it.

$$\text{Cost} = \underset{\text{fun}^n.}{\text{loss}} + \lambda \sum_{i=1}^{m} |w_i|$$

m – no. of features.

In L2, we have:

Here, lambda is the regularization parameter. It is the hyperparameter whose value is optimized for better results. L2 regularization is also known as weight decay as it forces the weights to decay towards zero (but not exactly zero).

$$\text{Cost} = \underset{\text{fun}^n.}{\text{loss}} + \lambda \sum_{i=1}^{m} |w_i^2|$$

DEEP LEARNING

GAN (Generative Adversarial Network)

Generated Data

Discriminator

Fake

Real

Real Data

As training process progress the generator gets closer to producing output that can fool the discriminator

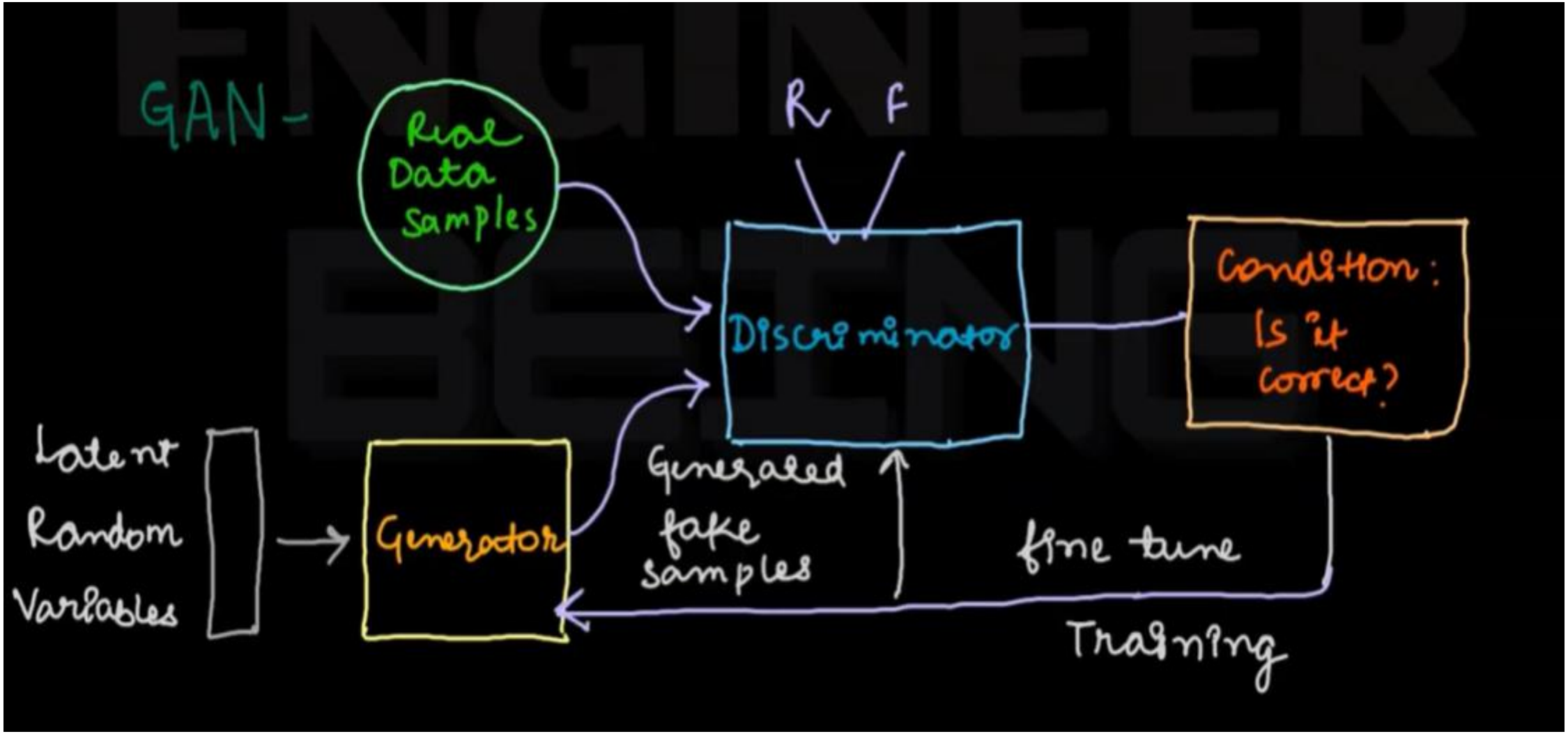
fake

Real

Finally, if generated training goes well the discriminator gets worst at telling the difference between real and fake, it starts to classify fake data as real, its accuracy decreases

Real

Real


GAN -
Real Data samples
R F
Discriminator
Condition: Is it correct?
Latent Random Variables
Generator
Generated fake samples
fine tune
Training

How does a GAN work?

A Generative Adversarial Network (GAN) is a framework made up of two neural networks that have undergone simultaneous adversarial training—a discriminator and a generator. The discriminator separates generated data from real data, while the generator produces synthetic data that attempts to imitate real data. Training makes the generator more adept at producing realistic samples in an effort to trick the discriminator, which strengthens the generator's discriminating abilities. GANs are an effective tool for producing realistic, high-quality outputs in a variety of fields, including text and image generation, because of this back-and-forth competition, which results in the creation of increasingly convincing and indistinguishable synthetic data.

Different Types of GAN Models

- Vanilla GAN: This is the simplest type of GAN. Here, the Generator and the Discriminator are simple multi-layer perceptrons. In vanilla GAN, the algorithm is really simple, it tries to optimize the mathematical equation using stochastic gradient descent.

- Conditional GAN (CGAN): CGAN can be described as a deep learning method in which some conditional parameters are put into place. 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.

- Deep Convolutional GAN (DCGAN): DCGAN is one of the most popular and also the most successful implementations of GAN. It is composed of ConvNets in place of multi-layer perceptrons. The ConvNets are implemented without max pooling, which is in fact replaced by convolutional stride. Also, the layers are not fully connected.

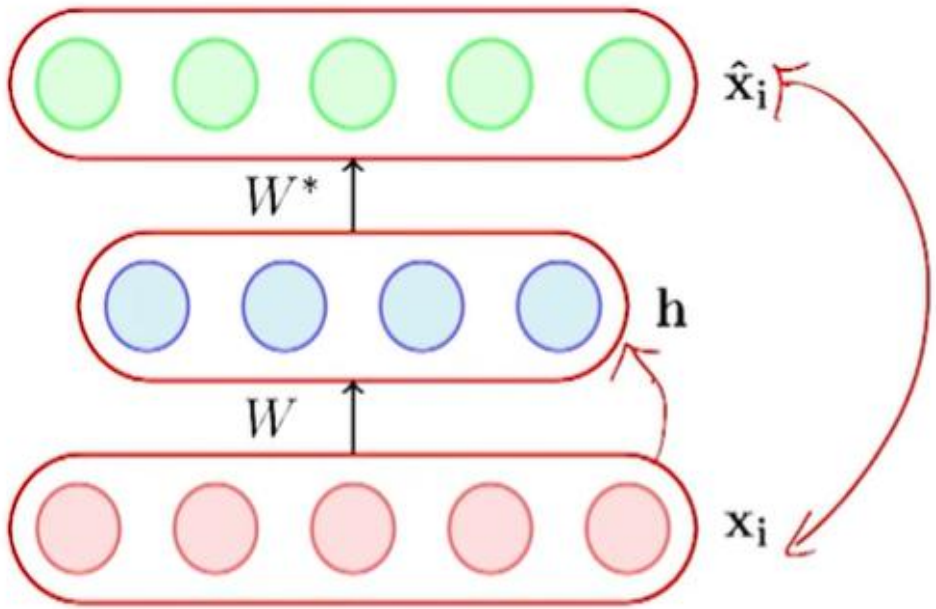
Laplacian Pyramid GAN (LAPGAN): The Laplacian pyramid is a linear invertible image representation consisting of a set of band-pass images, spaced an octave apart, plus a low-frequency residual. This approach uses multiple numbers of Generator and Discriminator networks and different levels of the Laplacian Pyramid. This approach is mainly used because it produces very high-quality images. The image is down-sampled at first at

each layer of the pyramid and then it is again up-scaled at each layer in a backward pass where the image acquires some noise from the Conditional GAN at these layers until it reaches its original size.

- Super Resolution GAN (SRGAN): SRGAN as the name suggests is a way of designing a GAN in which a deep neural network is used along with an adversarial network in order to produce higher-resolution images. This type of GAN is particularly useful in optimally up-scaling native low-resolution images to enhance their details minimizing errors while doing so.

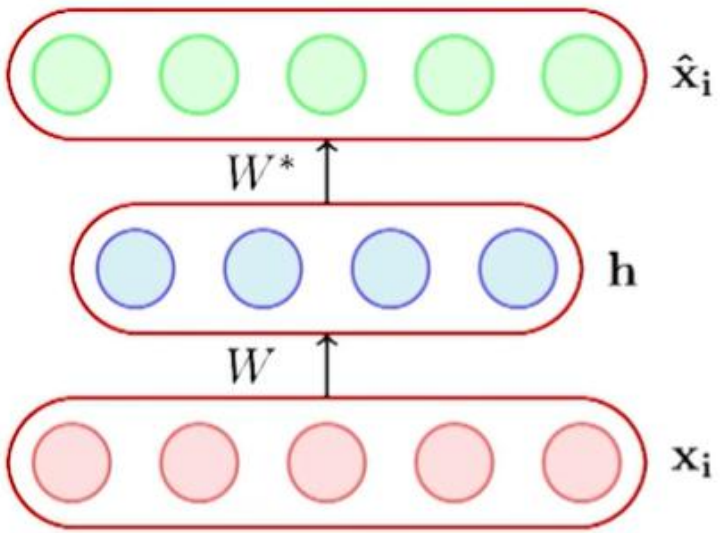
Autoencoders

An autoencoder is a type of neural network architecture used in deep learning for unsupervised learning and dimensionality reduction tasks. It is primarily designed to learn efficient representations of data by encoding it into a lower-dimensional latent space and then decoding it back to the original data space. Autoencoders are widely used for various applications, including data compression, denoising, feature learning, and anomaly detection.



$$\mathbf{h} = g(W\mathbf{x_i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

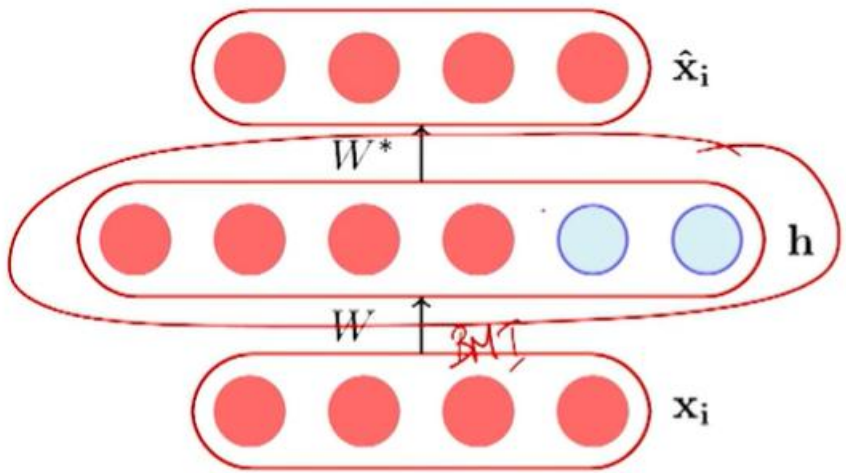
- An autoencoder is a special type of feed forward neural network which does the following
- Encodes its input $\mathbf{x_i}$ into a hidden representation $\mathbf{h}$
- Decodes the input again from this hidden representation
- The model is trained to minimize a certain loss function which will ensure that $\hat{\mathbf{x}}_\mathbf{i}$ is close to $\mathbf{x_i}$ (we will see some such loss functions soon)



$$\mathbf{h} = g(W\mathbf{x_i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

An autoencoder where $\dim(\mathbf{h}) < \dim(\mathbf{x_i})$ is called an under complete autoencoder

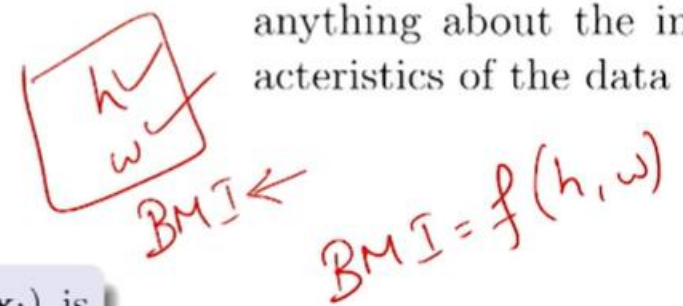
- Let us consider the case where $\dim(\mathbf{h}) < \dim(\mathbf{x_i})$
- If we are still able to reconstruct $\hat{\mathbf{x}}_\mathbf{i}$ perfectly from $\mathbf{h}$, then what does it say about $\mathbf{h}$?
- $\mathbf{h}$ is a loss-free encoding of $\mathbf{x_i}$. It captures all the important characteristics of $\mathbf{x_i}$
- Do you see an analogy with PCA?

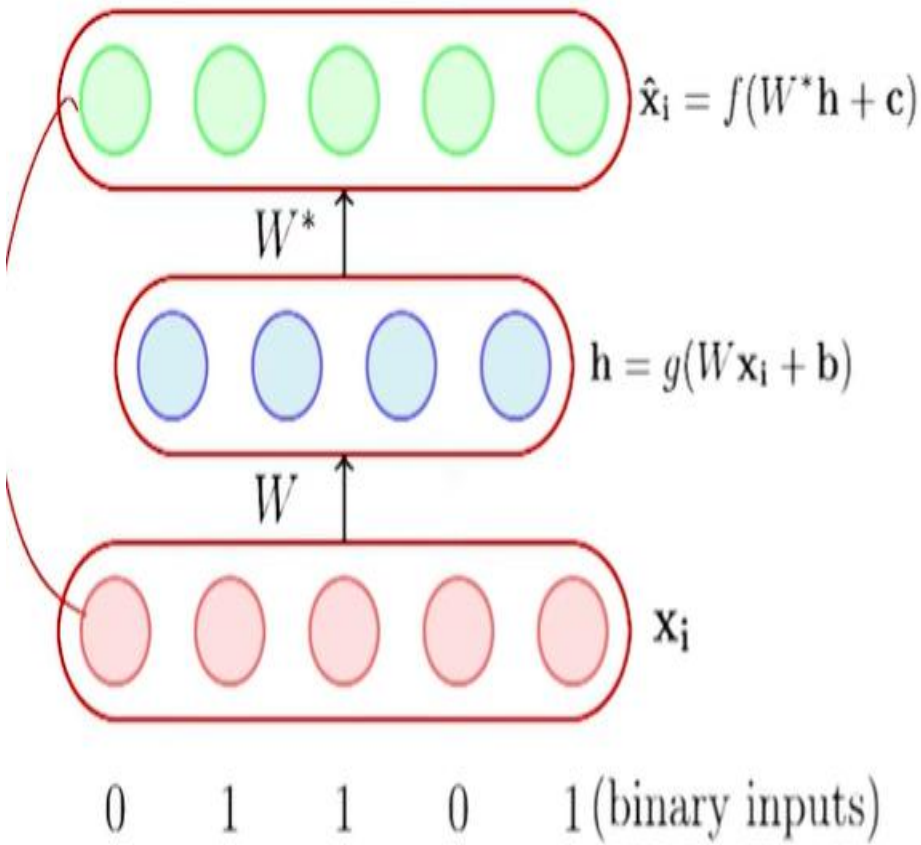


$$\mathbf{h} = g(W\mathbf{x_i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

An autoencoder where $\dim(\mathbf{h}) \geq \dim(\mathbf{x_i})$ is called an over complete autoencoder

- Let us consider the case when $\dim(\mathbf{h}) \geq \dim(\mathbf{x_i})$
- In such a case the autoencoder could learn a trivial encoding by simply copying $\mathbf{x_i}$ into $\mathbf{h}$ and then copying $\mathbf{h}$ into $\hat{\mathbf{x}}_\mathbf{i}$
- Such an identity encoding is useless in practice as it does not really tell us anything about the important characteristics of the data





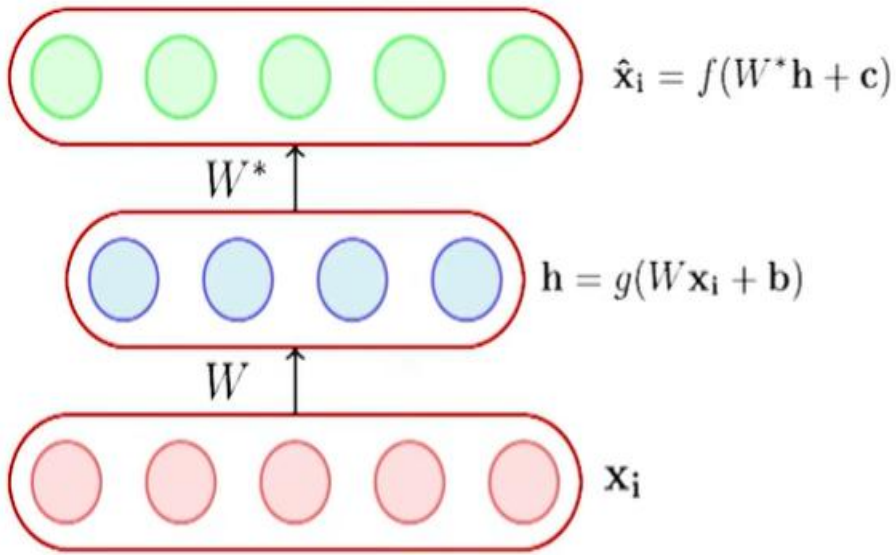
- Suppose all our inputs are binary (each $x_{ij} \in \{0, 1\}$)
- Which of the following functions would be most apt for the decoder?

$$\hat{\mathbf{x}}_i = \tanh(W^*\mathbf{h} + \mathbf{c})$$

$$\hat{\mathbf{x}}_i = W^*\mathbf{h} + \mathbf{c}$$

$$\hat{\mathbf{x}}_i = \mathit{logistic}(W^*\mathbf{h} + \mathbf{c})$$

- Logistic as it naturally restricts all outputs to be between 0 and 1

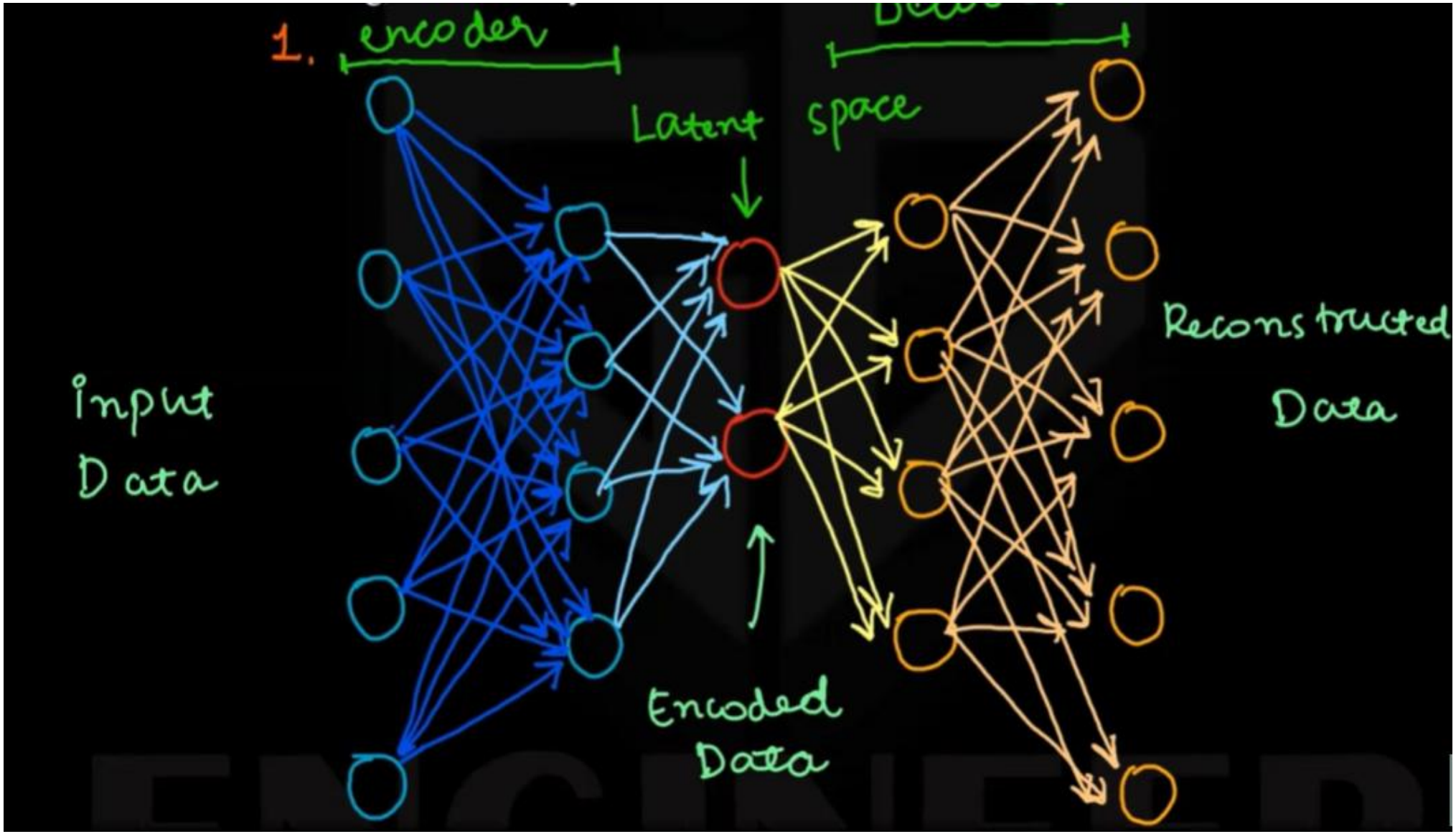


Again, g is typically chosen as the sigmoid function

- Suppose all our inputs are real (each $x_{ij} \in \mathbb{R}$)
- Which of the following functions would be most apt for the decoder?

$$\hat{\mathbf{x}}_i = \tanh(W^*\mathbf{h} + \mathbf{c})$$
$$\hat{\mathbf{x}}_i = W^*\mathbf{h} + \mathbf{c}$$
$$\hat{\mathbf{x}}_i = \text{logistic}(W^*\mathbf{h} + \mathbf{c})$$

- What will logistic and tanh do?
- They will restrict the reconstructed $\hat{\mathbf{x}}_i$ to lie between [0,1] or [-1,1] whereas we want $\hat{\mathbf{x}}_i \in \mathbb{R}^n$


1. encoder
Latent space
Input Data
Encoded Data
Reconstructed Data

1. Architecture: Autoencoders consist of two main parts: an encoder and a decoder.
    - Encoder: The encoder network takes the input data and maps it to a lower-dimensional representation, often called the encoding or latent space. This encoding is typically a compressed representation of the input data. The encoder can consist of one or more hidden layers and uses activation functions like ReLU (Rectified Linear Unit).
    - Decoder: The decoder network takes the encoded representation and attempts to reconstruct the original input data from it. Like the encoder, the decoder can also have one or more hidden layers and uses activation functions.

Objective: The main objective of an autoencoder is to minimize the reconstruction error, which measures how well the decoder can reconstruct the input data from the encoding. Common loss functions used for this purpose include mean squared error (MSE) or binary cross-entropy, depending on the type of data (continuous or binary) and the specific task.

Training: Autoencoders are trained in an unsupervised manner, which means they don't require labeled data. The training process involves feeding input data into the encoder, encoding it into a lower-dimensional representation, and then decoding it back to the original data. The difference between the input and the reconstructed output is used to compute the loss, which is minimized during training using optimization techniques like gradient descent.

Variations of Autoencoders: There are several variations of autoencoders, each designed for specific tasks or data types. Some common types include:

- Denoising Autoencoder: Trained to remove noise from data by corrupting the input and reconstructing the clean data.
- Sparse Autoencoder: Encourages the network to learn sparse representations, which can be useful for feature learning and dimensionality reduction.
- Variational Autoencoder (VAE): Introduces probabilistic elements in the latent space, allowing for the generation of new data samples and improved data compression.

- Convolutional Autoencoder: Specifically designed for image data, using convolutional layers in the encoder and decoder.
- Recurrent Autoencoder: Suitable for sequential data, such as time series, by using recurrent layers in the architecture.

Now, if we consider the numerical aspect of an autoencoder, here's how it works:

1. Input Data: Autoencoders take numerical data as input. This data could be anything from images (pixel values), audio (amplitude values over time), text (word embeddings), or any other form of numerical representation.

2. Encoder: The encoder network receives this numerical data and maps it to a lower-dimensional numerical representation. For example, in the case of image data, the encoder might transform pixel values into a reduced set of numerical features that capture essential information.

3. Decoder: The decoder takes the reduced numerical representation and attempts to reconstruct the original numerical data. It does so by mapping the reduced representation back to the original numerical format.

4. Loss Function: During training, the difference between the input numerical data and the reconstructed data is measured using a numerical loss function. The model then adjusts its weights and biases through numerical optimization techniques to minimize this loss.

In summary, autoencoders are a versatile tool in deep learning for learning compact representations of data and have numerous applications in various domains, working with both numerical and non-numerical data.

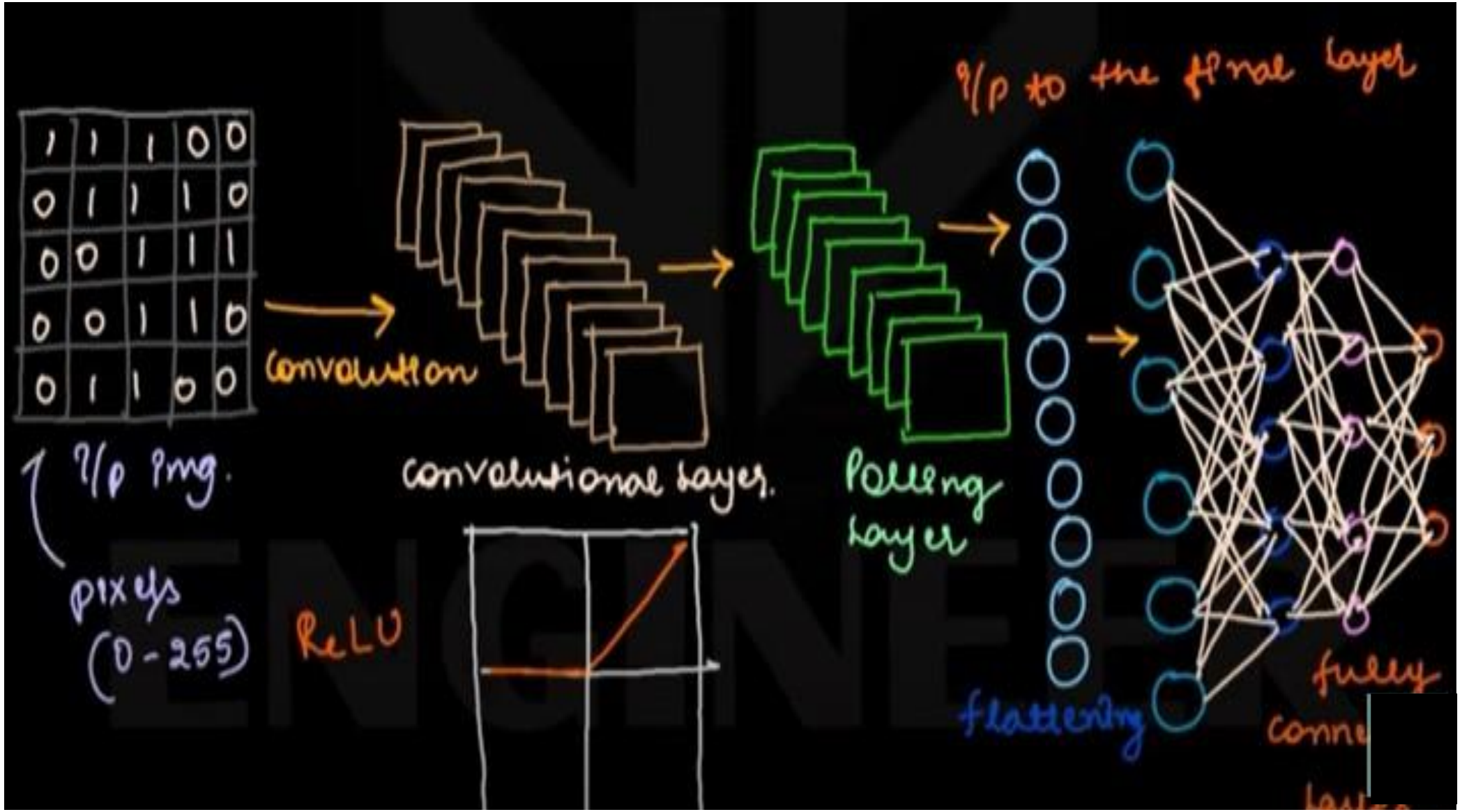
CNN (Convolutional neural Network)

Convolutional Neural Networks (CNNs) are specially designed to work with images. An image consists of pixels. In deep learning, images are represented as arrays of pixel values.

There are four main types of layers in a CNN:

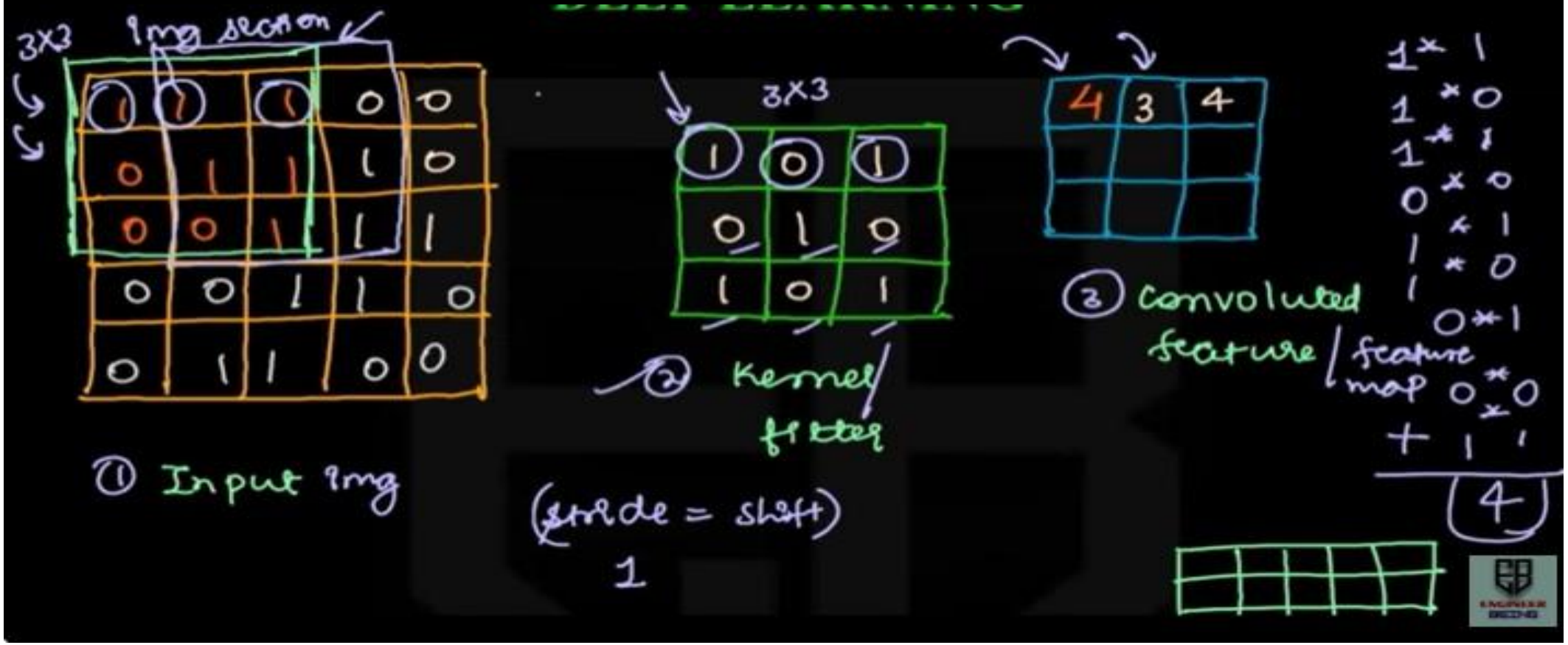
- Convolutional layers
- ReLU
- Pooling layers
- Fully connected (dense) layers.

There are four main types of operations in a CNN:
Convolution operation, Pooling operation, Flatten operation and Classification (or other relevant) operation.


1 1 1 0 0
0 1 1 1 0
0 0 1 1 1
0 0 1 1 0
0 1 1 0 0
convolution
convolutional layer.
pooling layer
o/p to the final layer
i/p img.
pixels
(0-255)
ReLU
flattening
fully
connected
layer

1. Convolutional layers and convolution operation: The first layer in a CNN is a convolutional layer. It takes the images as input and begins to process.

There are three elements in the convolutional layer: Input image, Filters and Feature map.



Filter: This is also called Kernel or Feature Detector.

Image section: The size of the image section should be equal to the size of the filter(s) we choose. The number of image sections depends on the Stride.

Feature map: The feature map stores the outputs of different convolution operations between different image sections and the filter(s).

The number of steps (pixels) that we shift the filter over the input image is called Stride.

ReLU Layer (Rectified Linear Unit):

- Function: The ReLU layer introduces non-linearity to the network by applying the Rectified Linear Unit activation function.
- Operations: ReLU activation sets all negative values in the feature map to zero and leaves positive values unchanged: $f(x)=\max(0,x)$.
- Advantages: ReLU helps in overcoming the vanishing gradient problem and accelerates convergence during training.

The Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data by reducing the dimensions.

There are two types of pooling operations.

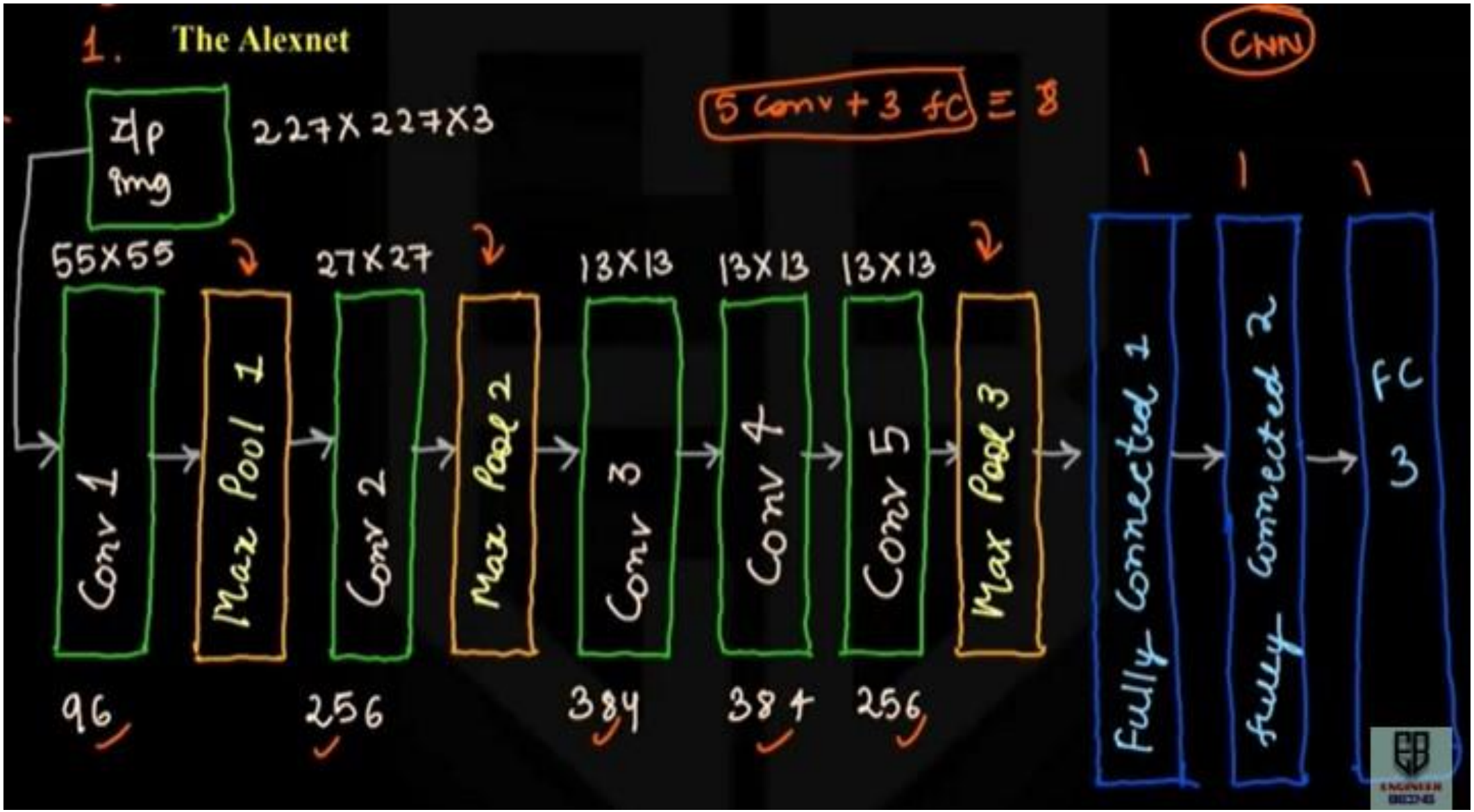
- Max pooling: Get the maximum value in the area where the filter is applied.
- Average pooling: Get the average of the values in the area where the filter is applied.

Flattening

- The next step in the process is called flattening. Flattening is used to convert all the resultant 2-Dimensional arrays from pooled feature maps into a single long continuous linear vector.
- The flattened matrix is fed as input to the fully connected layer to classify the image.

Fully Connected Layer:

- Function: The fully connected layer is a traditional neural network layer where each neuron is connected to every neuron in the previous and subsequent layers.
- Operations: The output of the previous layers is flattened into a vector and connected to the neurons of the fully connected layer. This layer learns complex patterns and relationships in the data.
- Parameters: The number of neurons in the fully connected layer is a crucial parameter that affects the model's capacity.


1. The Alexnet
CNN
Ip img
227X227X3
5 conv + 3 fc ≡ 8
55X55
27X27
13X13
13X13
13X13
Conv 1
Max Pool 1
Conv 2
Max Pool 2
Conv 3
Conv 4
Conv 5
Max Pool 3
Fully connected 1
fully connected 2
FC 3
1
1
1
96
256
384
384
256

The AlexNet model was proposed in 2012 in the research paper named ImageNet Classification with Deep Convolutional Neural Network by Alex Krizhevsky and his colleagues.

AlexNet has eight layers with learnable parameters.

The model has five layers with a combination of max pooling followed by 3 fully connected layers.

The fully connected layers use Relu activation except the output layer

They found out that using the Relu as an activation function accelerated the speed of the training process by almost six times.

They also used dropout layers, which prevented the model from overfitting.

The model is trained on the ImageNet dataset. The Imagenet dataset has around 14 million images across 1000 classes.

The input to this model is the images of size 227X227X3.

The first convolution layer with 96 filters of size 11X11 with stride 4

The activation function used in this layer is relu. The output feature map is 55X55X96.

Next, we have the first Max Pooling layer of size 3X3 and stride 2.

Next the filter size is reduced to 5X5 and 256 such filters are added

The stride value is 1 and padding 2. The activation function used is again relu. The output size we get is 27X27X256

Next we have a max-pooling layer of size 3X3 with stride 2. The resulting feature map size is 13X13X256

The third convolution operation with 384 filters of size 3X3 stride 1 and also padding 1is done next. In this stage the activation function used is relu. The output feature map is shaped 13X13X384.

Then the fourth convolution operation with 384 filters of size 3X3. The stride value along with the padding is 1. The output size remains unchanged as 13X13X384.

After this, we have the final convolution layer of size 3X3 with 256 such filters. The stride and padding are set to 1, also the activation function is relu. The resulting feature map is of shape 13X13X256.

If we look at the architecture now, the number of filters is increasing as we are going deeper. Hence more features are extracted as we move deeper into the architecture. Also, the filter size is reducing, which means a decrease in the feature map shape.