IDC410
A course on Image Processing and Machine Learning
(Lecture 15)

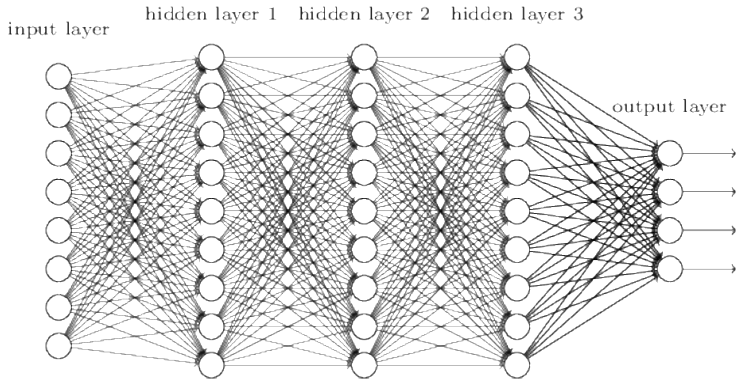
Shashikant Dugad,

IISER Mohali

Convolutional Neural Networks (CNN)

Convolutional Neural Networks

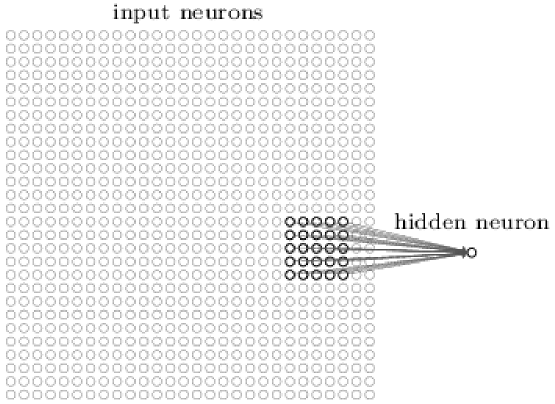
- **Fully connected network (FCN):**
  - The ***FCN*** does not take into account the ***spatial structure (features)*** of the images that may appear in ***ANY*** region of the image.
  - It treats input pixels which are far apart and close together on exactly the same footing!
  - It is rather a brut-force approach to learn the image without paying attention to the spatial features.
  - This results in large number of weights and biases making it computational challenging
  - Not a suitable approach for complex inputs



- Instead, build an architecture that will exploit built-in features present in the image
- ***Convolutional filters*** are best to extract spatial structure ***(features)*** in the image during training
- Use of filters significantly reduce the number of free parameters ***(weights and biases)*** making training faster

Convolutional Neural Networks

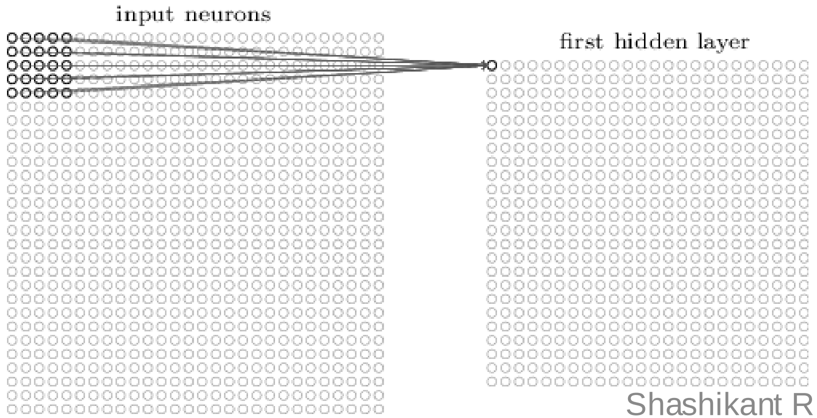
- **Convolutional Neural Network (CNN):** They are used in the most neural networks for image recognition and many other applications
- CNN uses three basic ideas: *a) Local receptive fields, b) shared weights/biases, and c) pooling*
- In the *FCN*, the inputs were depicted as a vertical line of neurons (1-dimensional input). In *CNN*, inputs are depicted *28×28 squared array* for a grey scale image *(2-dimensional input)* of neurons
- Unlike FCN, pixels in the input image are connected to only few neurons of an hidden layer representing localized regions of the input image.

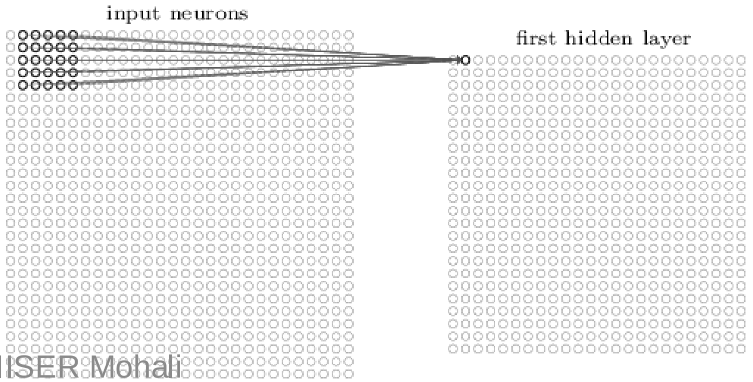


- For example, a *5×5 region*, corresponding to a *25 input pixels (NOT all input pixels)* are connected to particular hidden neuron.
- This region (*5x5 in figure*) in the input image is called the *local receptive field* for that particular connected hidden neuron.

Convolutional Neural Networks

- The value at the hidden neuron is obtained by applying ***5x5 filter*** on the inputs received from ***Local Receptive Field*** followed by the application of activation function on the ***output*** of the filter
- The ***output*** of the filter is nothing but weighted sum of inputs received from the ***Local Receptive Field plus the bias***.
- ***Weights (25) and a bias*** are stored in each hidden neuron of the hidden layer
- Slide the ***local receptive field*** across the entire input image. For each ***local receptive field***, there is a specific hidden neuron in the first hidden layer as shown below
- Size of first hidden layer with ***5x5 filter*** applied on ***28x28*** image ***(with stride=1)*** is ***24x24***





Convolutional Neural Networks

- ***Shared weights and biases:*** Each hidden neuron has a ***bias*** and ***5×5 weights*** connected to its corresponding ***local receptive field*** in the input image.

- In the CNN, ***SAME weights*** and ***bias*** are used for all the ***local receptive fields in the input image*** which means, the ***SAME weights*** and ***bias*** for each of the ***24x24*** neurons in the first hidden layer! Due to this, they are called ***Shared weights and biases***

- Such approach significantly reduces the ***number of unknown parameters*** while training

- Therefore, final value stored at the ***(j,k)th*** hidden neuron in the first hidden layer is given by:

$$z^{L}_{j,k} = \left(b + \sum_{l=0}^{4} \sum_{m=0}^{4} w_{lm} a^{L-1}_{j+l,k+m} \right) \qquad a^{L}_{j,k} = \sigma\left(z^{L}_{j,k}\right)$$

- w_{lm} is matrix element (weight) in the ***5x5 filter***. ***b*** is bias. $a^{L-1}_{j+l,k+m}$ represents input to the hidden layer (the pixel intensity in the input image), **σ** is the activation function. $a^{L}_{j,k}$ is the value stored for the ***(j,k)th*** position in the hidden layer

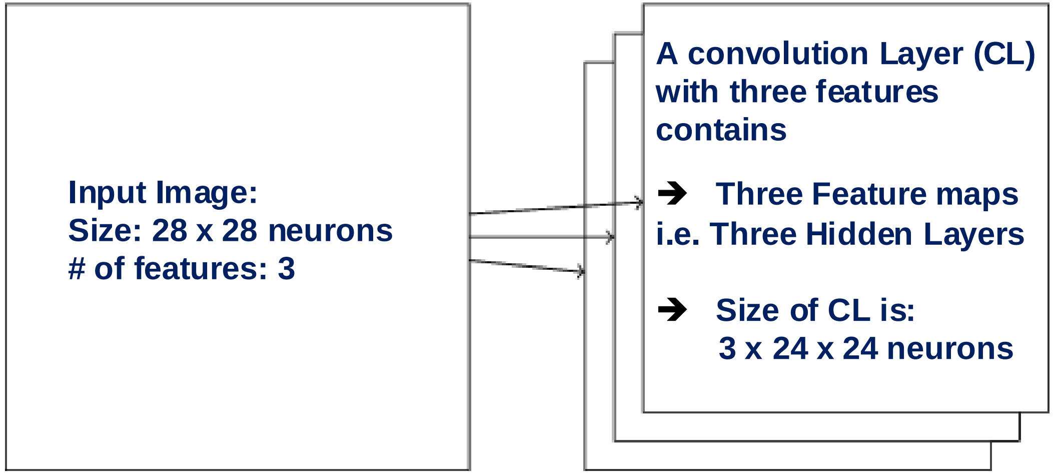
- Note: Convolutional filter OR the weight matrix w_{lm} and the ***bias b*** is same for all the neurons in the hidden layer!

Convolutional Neural Networks

- Since all the hidden neurons in a given layer shares exactly same weights and bias, the first hidden layer detects the presence of a particular ***feature*** along with ***its location*** in the input image!
- The shared weights and bias defines ***a kernel or filter***. The ***filter*** is often referred as the ***feature detector (or extractor)***. It is useful to have the same ***feature extractor*** applied across the entire image.
- CNN are well adapted to the translational invariance of features embedded anywhere in the images
- Therefore, the hidden layer can also be called as a ***feature map*** containing information on ***location*** of ***a particular feature*** in image
- However, there can be several features in the image which needs to be extracted. Therefore, there has to be several ***feature maps (hidden layers)*** for ***each of the features***
    - A given complete ***convolutional layer*** consists of several **different feature maps**

Convolutional Neural Networks

- In the example shown below, there are ***3 features*** in the image. Therefore, there are ***3 kernels (feature extractors)*** resulting into ***3 feature maps***, i.e., ***3 hidden layers***
- Each ***feature extractor (kernel/filter)*** is defined by a set of ***5×5 shared weights, and a single shared bias*** *resulting into* **3 hidden layers (feature maps).** The result is that the network can detect 3 different kinds of features



Input Image:
Size: 28 x 28 neurons
# of features: 3
A convolution Layer (CL) with three features contains
➔ Three Feature maps i.e. Three Hidden Layers
➔ Size of CL is: 3 x 24 x 24 neurons

A convolutional layer ***(CL)*** containing ***N features***, results into ***N hidden layers (feature maps).***

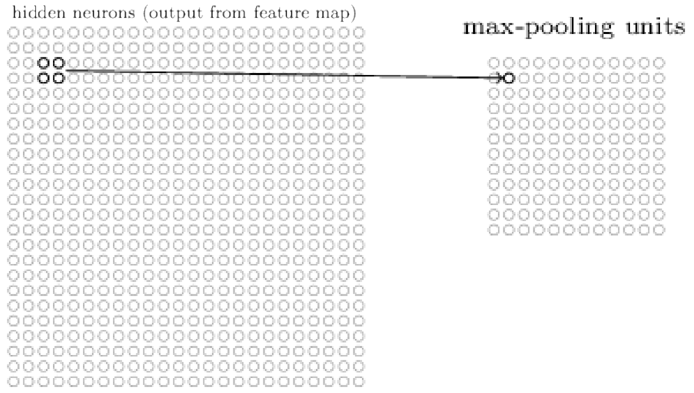
Each of the hidden layers (or feature map) of a given convolutional layer has the ***SAME*** size

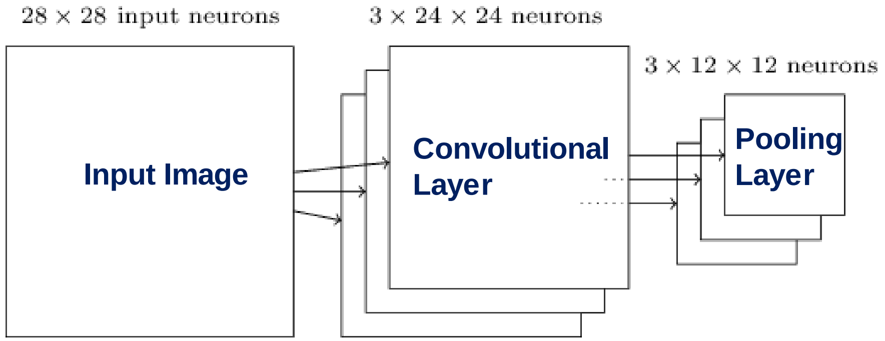
CL arrange neurons in 3-dimensions, so layers have width, height, and depth

Depth represents # of features

Pooling Layers

- CNN also contain *pooling layers; immediately after* the convolutional layers to simplify the information in the output *(feature map)* from the convolutional layer.
- The *pooling layer* takes each *feature map (hidden layer)* output from the convolutional layer and *prepares a condensed feature map*.
- For instance, each element (unit) in the pooling layer may summarize a region of (say) 2×2 neurons in the previous *hidden layer (feature map)* .
- *Max-pooling:* In max-pooling, a pooling unit simply outputs the maximum activation, say, in the 2×2 region of *feature map*



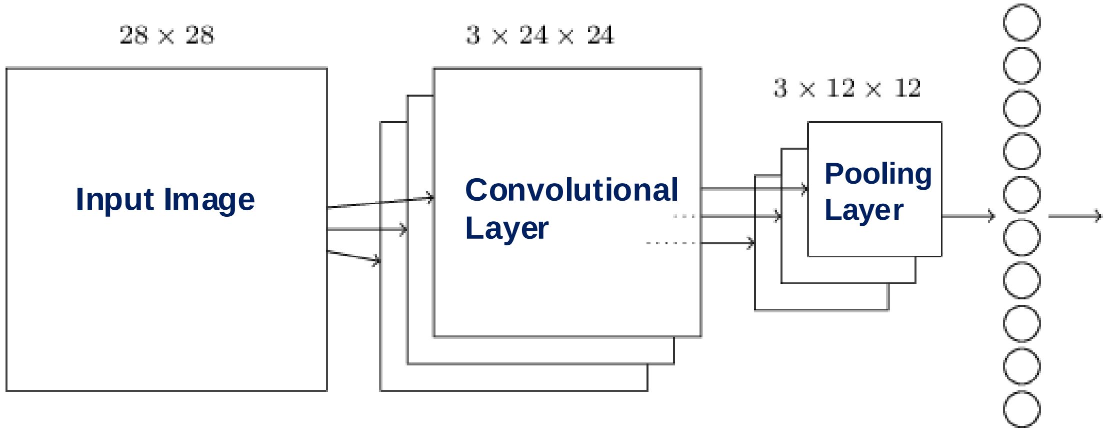


Pooling Layers

- The ***convolutional layer*** usually involves several ***feature maps***. Hence, the max-pooling is applied to each of the ***feature maps*** separately. So, if there were ***three feature maps***, in the convolutional layer then, it will result into ***three max-pooling layers***
- ***Positional information of the feature*** is somewhat ***compromised*** when using pooling layers
- ***L2 Pooling:*** Instead of taking the maximum activation of a 2×2 region of neurons, we take the square root of the sum of the squares of the activations in the 2×2 region. ***L2 pooling*** is a ***similar*** way of condensing information from the convolutional layer.

Complete Network with CNN

- ***Task:*** Identify a number between 0 to 9 (10 binary classes)
- The network begins with ***28×28 input neurons***, which are used to encode the pixel intensities for the MNIST image.
- The input image is followed by a convolutional layer using a ***5×5 local receptive field*** with ***3 feature maps*** followed by a ***max-pooling layer***, applied to ***2×2 regions***, across each of the ***3 feature maps*** resulting in a ***3 pooling layers*** of ***12×12*** size as discussed before.
- The final layer of connections in the network is a fully-connected layer. The final layer connects every neuron from the max-pooled layer to every one of the 10 output neurons.

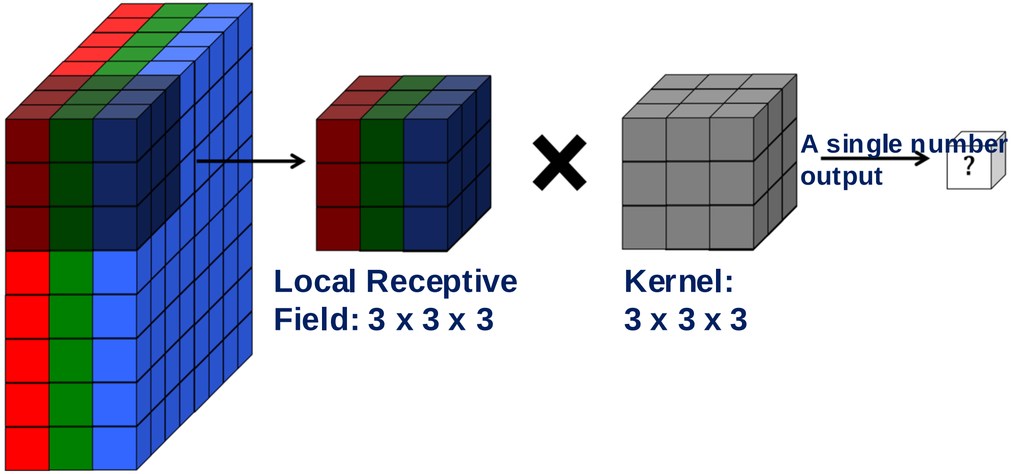


Volumetric Convolutional Filter

- Fully coloured (RGB) image can be viewed as a volume. Therefore, ***local receptive field*** also would be viewed as volume. Therefore the kernel also has to have a shape of volume.

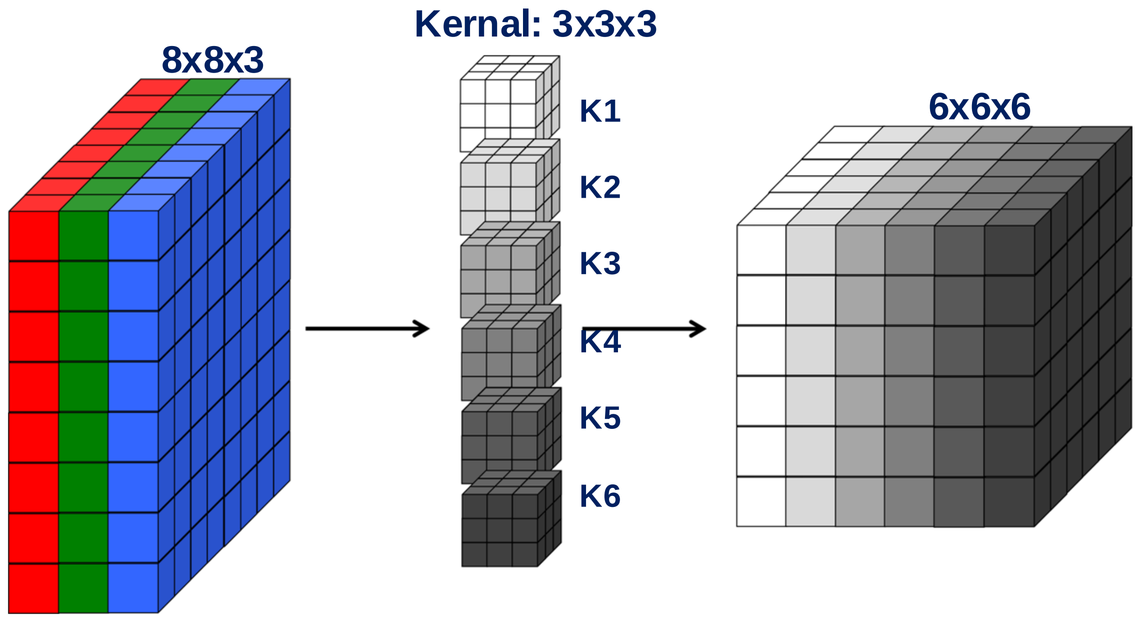
- ***Volumetric CL:*** For a coloured image, ***volumetric kernel of size nxnx3*** is applied on a ***volumetric local receptive field*** of the ***same depth (nxnx3)*** to obtain a ***single number output*** on which the ***activation function*** is applied.

- If ***ONLY*** one ***volumetric kernel (single feature)*** is applied on coloured image, then the CL will have ***ONLY one 2-dimensional hidden layer***

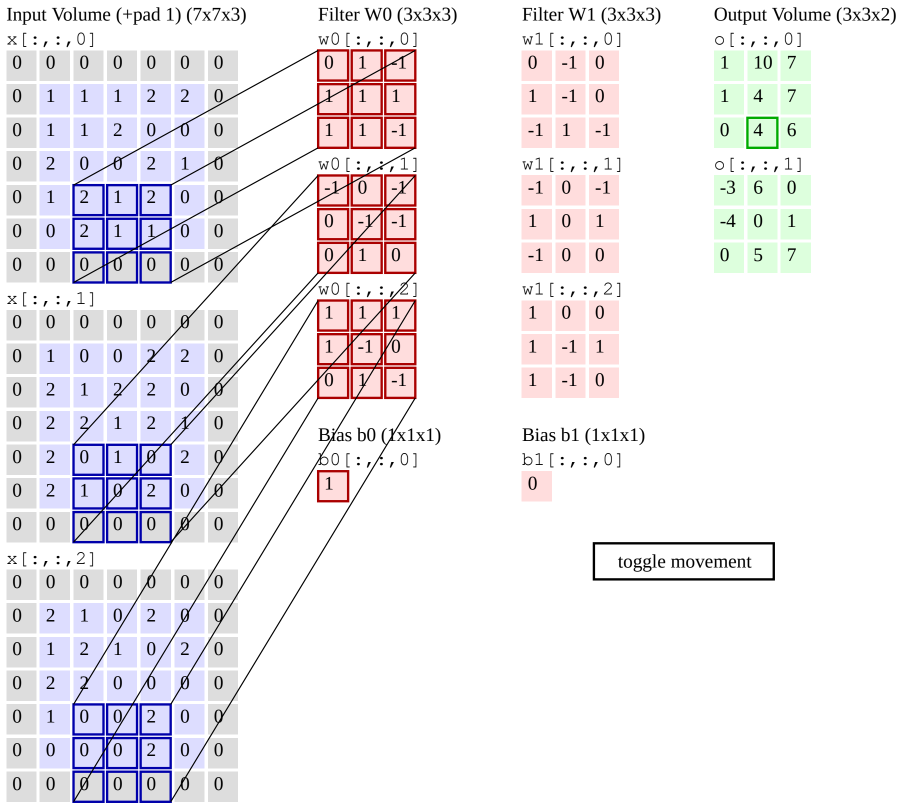


Multi-Feature Volumetric Convolutional Filter

- In figure below, ***6 volumetric kernels*** (K1, K2, …, K6) are applied on a coloured image resulting into ***6 (2-dimensional) feature maps*** in a convolutional layer
- The convolution layer therefore has a ***depth of 6 (i.e. 6; two-dimensional hidden layers)***



Example: Multi-Feature Volumetric Convolutional Filter



Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	1	1	1	2	2	0
0	1	1	2	0	0	0
0	2	0	0	2	1	0
0	1	2	1	2	0	0
0	0	2	1	1	0	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	1	0	0	2	2	0
0	2	1	2	2	0	0
0	2	2	1	2	1	0
0	2	0	1	0	2	0
0	2	1	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	0	2	0	0
0	1	2	1	0	2	0
0	2	2	0	0	0	0
0	1	0	0	2	0	0
0	0	0	0	2	0	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

0	1	-1
1	1	1
1	1	-1

w0[:,:,1]

-1	0	-1
0	-1	-1
0	1	0

w0[:,:,2]

1	1	1
1	-1	0
0	1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

0	-1	0
1	-1	0
-1	1	-1

w1[:,:,1]

-1	0	-1
1	0	1
-1	0	0

w1[:,:,2]

1	0	0
1	-1	1
1	-1	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

1	10	7
1	4	7
0	4	6

o[:,:,1]

-3	6	0
-4	0	1
0	5	7

toggle movement

Input Color Image: 5x5x3 (without padding)

Padding = P = 1

# of Features = # of Kernels = K = 2

Kernel Size = Filter Size = F = 3x3

Stride = S = 2

https://medium.com/biased-algorithms/batch-normalization-vs-layer-normalization-c44472883bf2

Batch (BN) and Layer (LN) Normalization

Batch Normalization (BN) layers

- **Batch Normalization is a supervised learning technique that converts interlayer outputs of neural network into a standard format, called normalizing.**

- **In machine learning, "batch normalization" refers to a technique used to stabilize and accelerate the training process of neural networks by normalizing the inputs to each layer within a mini-batch of data, essentially re-centering and re-scaling the activations to improve learning speed and generalization ability by reducing internal covariate shift.**

- **In the context of YOLOv3, "batch normalization" refers to a technique applied after convolutional layers that helps stabilize the training process by normalizing the output activations of each layer, essentially ensuring that the data entering subsequent layers has a consistent distribution, leading to faster and more reliable training of the object detection model; it essentially acts as a regularization method to improve the network's performance by mitigating "internal covariate shift" where the distribution of activations in a layer can significantly change due to updates in previous layers.**

- **This approach leads to faster learning rates since normalization ensures there’s no activation value that’s too high or too low, as well as allowing each layer to learn independently of the others**

Batch Normalization (BN)

- BN blends the input values across the mini-batch so that each layer gets a well-mixed input. In BN, each feature across the entire mini-batch is standardized to have a mean of zero and a standard deviation of one. Here's how it does this:
- **Normalization:** It calculates the mean and variance for each feature across the current mini-batch.
- **Scaling and Shifting:** After normalizing the data, it applies a scaling factor (gamma) and a shifting factor (beta). These are learnable parameters, meaning the network can optimize them during training for better performance.
- So, each time data passes through a layer, BN ensures it's appropriately normalized, much like a quality check. The formula for Batch Normalization looks like this:

$$\hat{x}^{\,i} = \frac{x^i - \mu_B}{\sqrt{\sigma_B^2 + \in}}\,\gamma + \beta$$

- x^i: Input value in i^{th} neuron in batch. μ_B and σ_B mean and variance of mini-batch. ε is a small constant added to avoid division by zero condition. β and Y are learnable parameters that scale and shift the normalized values.

Batch Normalization: Advantages and Limitations

Advantages

- ***Faster Convergence:*** **With inputs normalized, the network trains faster because each layer receives more stable data.**
- ***Regularization Effect:*** **By introducing a slight noise due to mini-batch variation, BN has a regularizing effect, reducing the need for other techniques like dropout.**
- ***Higher Learning Rates:*** **Since BN reduces internal covariate shift, you can afford to use higher learning rates, making the training process even more efficient.**

Limitations

- ***Dependent on Mini-Batch Size:*** **If your mini-batch size is too small, the statistics (mean and variance) might not be reliable, leading to suboptimal performance.**
- ***Not Suitable for Recurrent Networks:*** **In scenarios like RNNs, where the input sequence length varies, BN isn’t as effective due to the lack of consistent mini-batches.**

Layer Normalization (LN)

- **Instead of blending across the batch like BN, it focuses on each data point individually. It computes the mean and variance across the features within a layer for each data point.**

- **It ensures that each sample is treated independently, making it especially effective in models like RNNs or transformers where input sequences can vary in length**

$$\hat{x}^{\,i} = \frac{x^i - \mu_{LN}}{\sqrt{\sigma_{LN}^2 + \in}}\;\gamma + \beta$$

- ***Independence from Batch Size*: LN works regardless of how many samples you have in a batch, making it perfect for tasks where batch size is limited or varies.**

- ***Better for RNNs and Transformers*: If you're dealing with sequence data or NLP tasks, LN is often more effective than BN.**

- ***Effective in Sequence Modelling:* It stabilizes the learning process in models that deal with long-term dependencies.**

Layer Normalization (LN)

Limitations

- ***Less Effective in CNNs:*** In convolutional networks, where spatial invariance and batch-based learning are crucial, BN tends to outperform LN.
- ***Computational Cost:*** LN can be slightly more computationally intensive in some cases since it normalizes over a larger set of parameters

Use Case

- BN is recommended for FFN or CNN where you have large datasets and can afford a decent batch size
- LN is recommended for Recurrent Neural Networks (RNNs), Long Short-Term Memory networks (LSTMs), and transformer models used in Natural Language Processing (NLP).