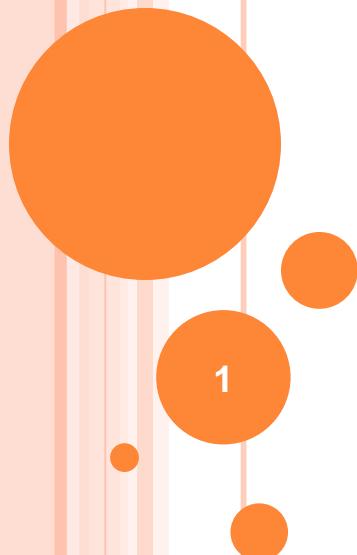


## **UNIT-III**

# **Convolutional Neural Networks**



# **CONTENTS**

- ❖ History of CNN
- ❖ Introduction about CNN
- ❖ Architecture of CNN
- ❖ Layers in CNN (1. Convolution Layer)
- ❖ Motivation
- ❖ Activation Functions

# HISTORY OF CNN

- CNN stands for **Convolutional Neural Networks**. It was first introduced by **Yann LeCun** who is a **post doctoral Computer Science Researcher**. CNNs were first developed and used around **the 1980s**. The most that a Convolutional Neural Network (CNN) could do at that time was **recognize handwritten digits**.
- It was mostly used in the **postal sector to read zip codes, pin codes, etc.** The important thing to remember about any deep learning model is that it requires a **large amount of data** to train and also **requires a lot of computing resources**.



- At that time also they used Back propagation algorithm used to train neural networks, was also computationally expensive at the time.
- This was a major drawback for CNNs at that period, and hence CNNs were only limited to the postal sectors and it failed to enter the world of machine learning.
- In 2012, Alex Krizhevsky realized that it was time to bring back the branch of deep learning that uses multi-layered neural networks. The availability of large sets of data, more specific ImageNet datasets with millions of labeled images, and an abundance of computing resources enabled researchers to revive CNNs.

# CONVOLUTIONAL NEURAL NETWORKS

- **CNN** stands for **Convolutional Neural Network**, which is a type of **deep learning model** commonly used in the **field of computer vision**. CNNs are particularly effective in tasks related to **image recognition and classification**.
- Convolution neural network (**also known as *ConvNet or CNN***) is a type of **feed-forward neural network** used in tasks like **image analysis**.
- This **CNN Algorithm** **take the image as an input** and learn the **various features of the image** through **filters**.



- It consists of 3 Layers. Those are
  - 1. Convolution Layer
  - 2. Pooling Layer
  - 3. Fully Connected Layer or Dense Layer
- Convolutional neural network (CNN) is a deep learning neural network designed for **processing structured arrays of data** such as **images**.
- A CNN is a **feed-forward neural network**, often with **up to 20 or 30 layers**.
- CNN represents the input data in the form of multidimensional arrays.

- It works well for a **large number of labeled data**. CNN extract the each and every portion of **input image**, which is known as receptive field. It **assigns weights for each neuron** based on the significant role of the receptive field.
- As soon as **we see an image**, **our brain starts categorizing it based on the colour, shape, etc..** Similar thing can be done through **machines even after a rigorous training**.
- But the **difficulty is there is a huge difference in what humans interpret and what machine does**.



- For a machine, the **image** is merely an array of pixels. There is a **unique pattern** included in **each object present in the image** and the computer tries to find out these patterns to get the **information about the image**.
- Machines can be trained giving tons of **images** to increase its ability to recognize the objects included in a **given input image**.
- Most of the **digital companies** have opted for **CNNs** for **image recognition**, some of these include **Google, Amazon, Instagram, Interest, Facebook, etc.**

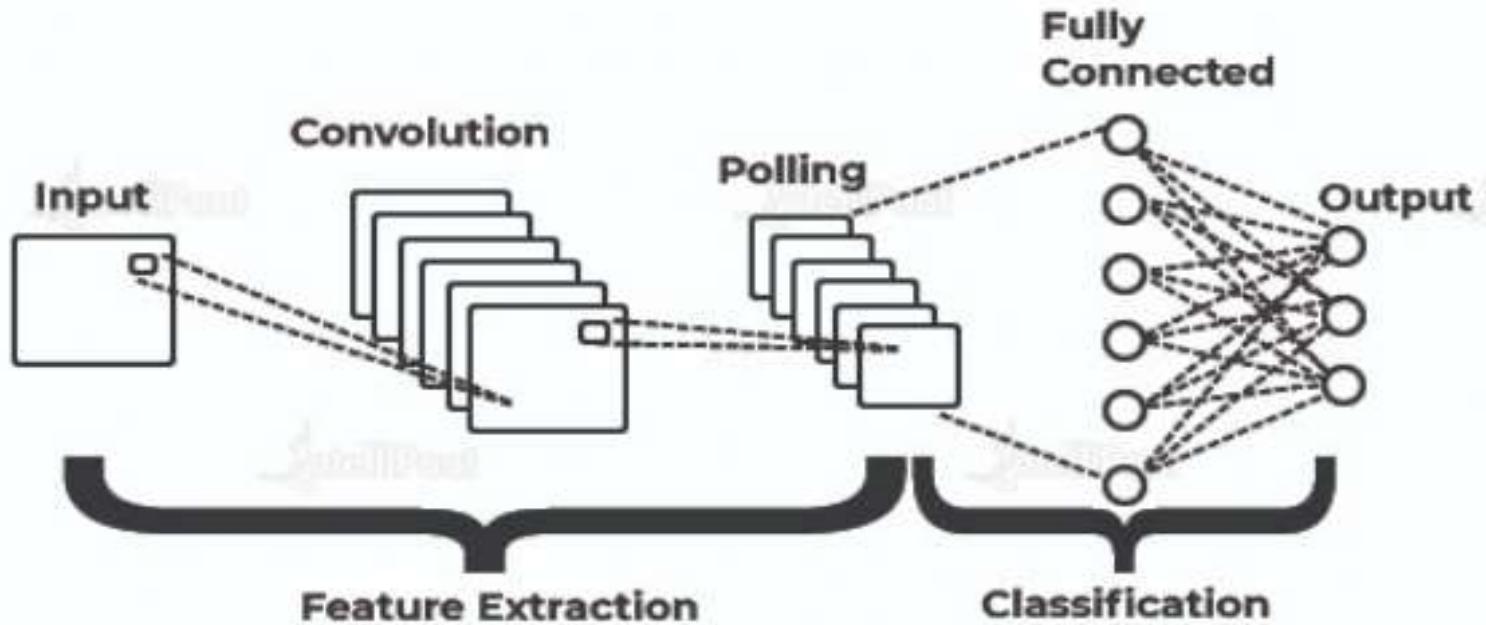


- Hence, we define **a convolutional neural network** as: "A neural network consisting of multiple convolutional layers which are used mainly **for image processing, classification, segmentation and other correlated data**".
- CNNs are designed to **solve problems with real time applications** of **image recognition and analysis**, including **image classification, drug discovery, and health risk assessments**. CNNs also help provide depth estimation for **self-driving cars**.



# ARCHITECTURE OF CNN

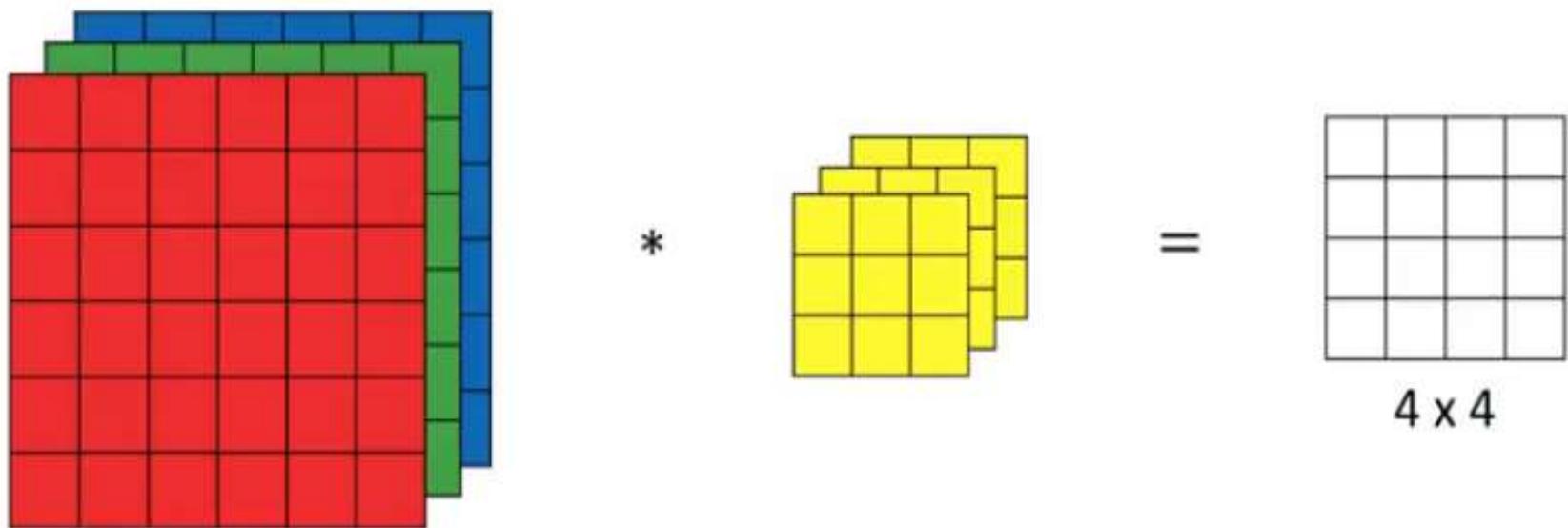
- CNN consists of **3 Layers**. Those are
  - **1. Convolution Layer**
  - **2. Pooling Layer**
  - **3. Fully Connected Layer or Dense Layer**



**CNN Architecture**

- CNN takes an **image as input**, which is classified and process under a certain category such as **dog, cat, lion, tiger, etc.**
- The computer sees an image as an **array of pixels** and **depends on the resolution of the image.**
- Based on image resolution, it will see as  **$h * w * d$** ,
  - where  **$h = \text{height}$**
  - **$w = \text{width}$**  and
  - **$d = \text{dimension.}$**
- For example, An **RGB image** is  **$6 * 6 * 3$**  array of the matrix, and the **gray scale image** is  **$4 * 4 * 1$**  array of the matrix.

- In CNN, **each input image** will pass through a sequence of **convolution layers** along **with pooling**, **fully connected layers**, **filters (Also known as kernels)**.
- After that, we will apply the **Activation function** to **classify an object with probabilistic values**.



## Types of CNNs:

- ✓ **1D CNN:** With these, the CNN kernel moves **in one direction**.  
1D CNNs are usually used on **time-series data**.
- ✓ **2D CNN:** These kinds of CNN kernels **move in two directions**.  
You'll see these used with **image labelling and processing**.
- ✓ **3D CNN:** This kind of CNN has a kernel that **moves in three directions**. With this type of CNN, researchers use them on **3D images like CT scans and MRIs**.
- ✓ In most cases, **you'll see 2D CNNs** because those are commonly associated with **image data**. Here are some of the applications  that you might see CNNs used for.

- Recognize images with little preprocessing
- Recognize different hand-writing
- Computer vision applications
- Used in banking to read digits on checks
- Used in postal services to read zip codes on an envelope



# LAYERS IN CNN

## 1. CONVOLUTION OPERATION

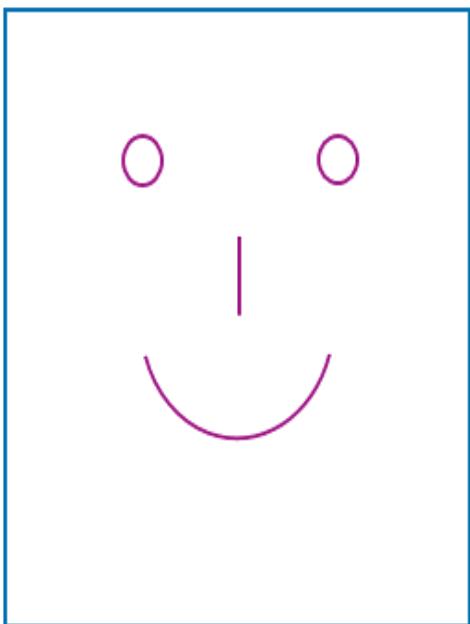
- 1. The Convolution Layer: Convolution operation focuses on extracting/preserving important features from the input.
- It means the network will learn specific patterns within the picture and will be able to recognize it everywhere in the picture.
- Convolution operation allows the network to detect horizontal and vertical edges of an image and then based on those edges build high-level features in the following layers of neural network.



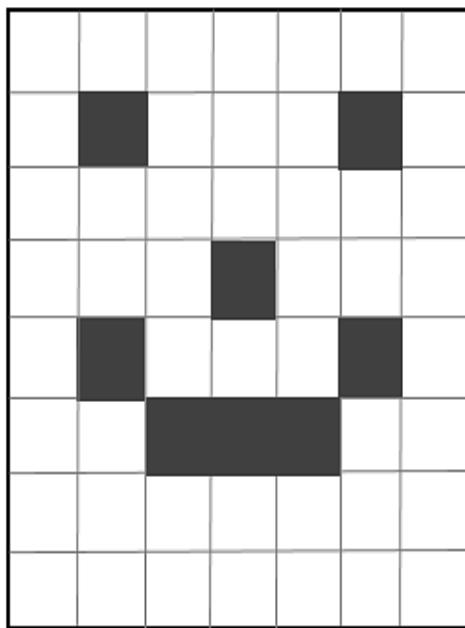
- Convolution operation uses three parameters:
  - Input image,
  - Feature detector and
  - Feature map.
- It uses the **mathematical operation** which takes **two inputs** such as **image matrix and a kernel or filter**.
- These **filters slide over the input data**, performing **element-wise multiplications** and **summations** to produce **feature maps**. Each filter specializes in **detecting a specific pattern** or **feature within the input**.

- Input matrix can be pixel values of a gray scale image whereas a filter is a relatively small matrix that detects edges by darkening areas of input image .
- There can be different types of filters depending upon what type of features we want to detect.
- Eg: Vertical, Horizontal, or Diagonal, etc.
- Input image is converted into binary 1 and 0. In this Image, how CNN recognizes an image: The convolution operation shown in the figure is known as the feature detector of a CNN.





Real Image



Represented in the form of  
black and white pixels

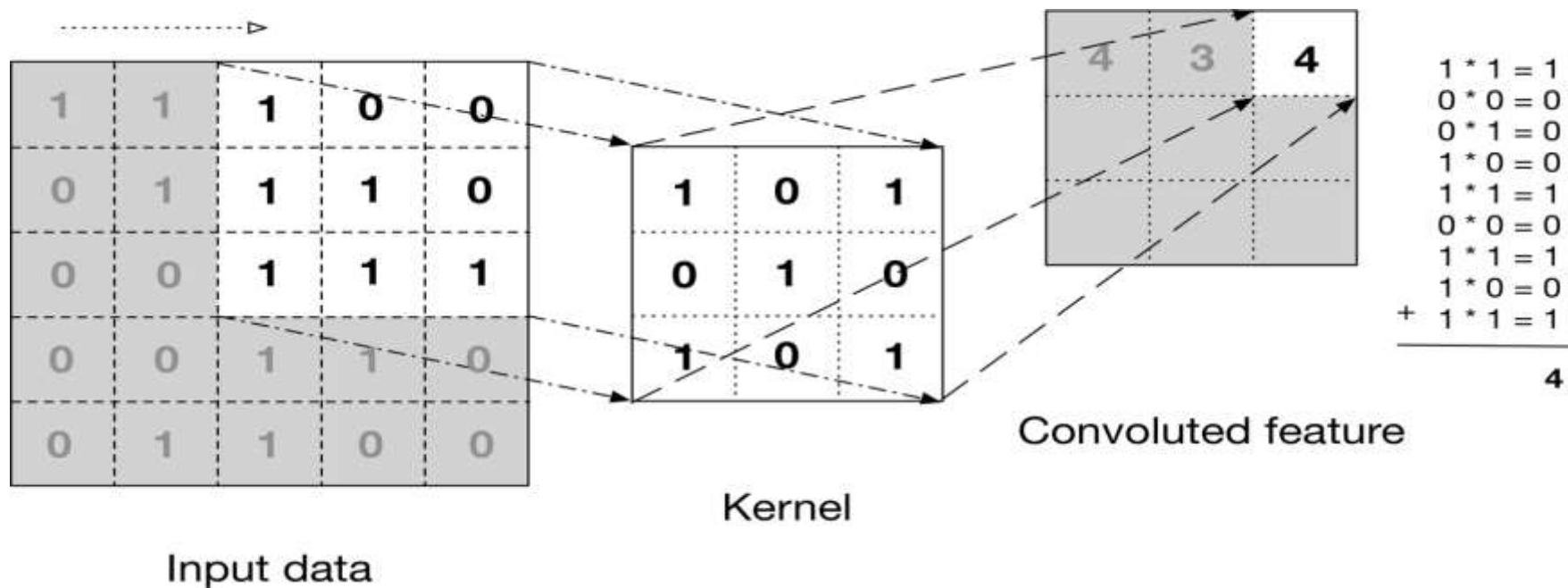


0	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	1	0	0	0	1	0	0
0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Image represented in the  
form of a matrix of numbers



- A feature detector, also known as a kernel or a filter, will traverse over the image's receptive fields, checking for the presence of the feature.
- For simplicity, let's stick with grayscale images as we try to understand how CNNs work.



- The input to a convolution can be raw data or a feature map output from another convolution.
- The **given image shows what a convolution is.** We take a **filter/kernel(3×3 matrix)** and **apply it to the input image** to get the **convolved feature.** This convolved feature is passed on to the next layer.

1	1	1	0	0
0	1	1	1	0
0	0	1 <sub>×1</sub>	1 <sub>×0</sub>	1 <sub>×1</sub>
0	0	1 <sub>×0</sub>	1 <sub>×1</sub>	0 <sub>×0</sub>
0	1	1 <sub>×1</sub>	0 <sub>×0</sub>	0 <sub>×1</sub>

Image

4	3	4
2	4	3
2	3	4

Convolved Feature



- Convoluting a **5x5x1 image** with a **3x3x1 kernel** to get a **3x3x1 convolved feature**
- Generally, an image can be considered as **a matrix** whose elements are numbers **between 0 and 255**. The size of image matrix is:  
**image height \* image width \* number of image channels.**
- A **gray scale image has 1 channel**, where a **colour image has 3 channels.**
- Let's pretend the input is a **color picture**, which is made up of a **3D matrix of pixels.**



- This implies the **input will have three dimensions**:
  - **height, width, and depth**, which match the **RGB color space of a picture**. Here we try to decompose RGB to a multidimensional layer and apply a filter to each layer.
- The dimension of the image matrix **is  $h \times w \times d$** .
- The dimension of the filter is  **$f_h \times f_w \times d$** .

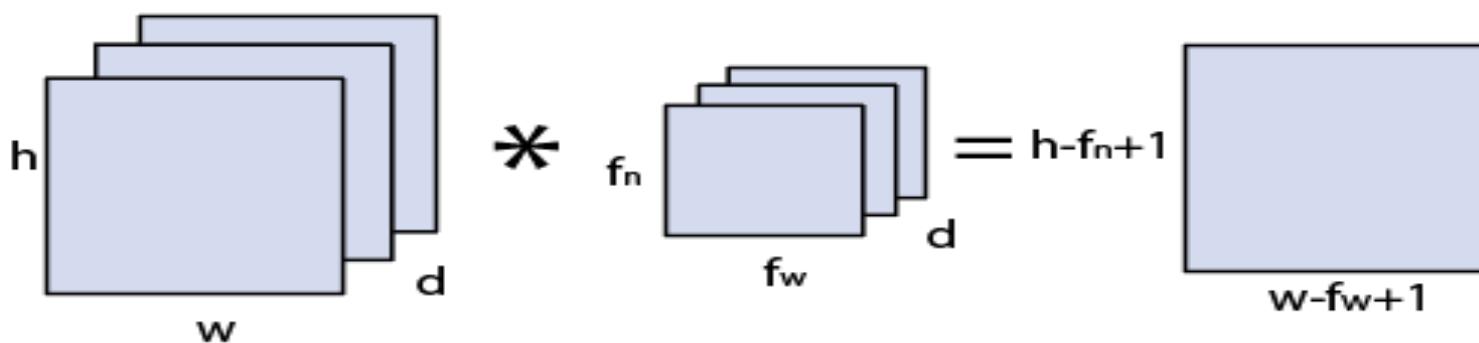


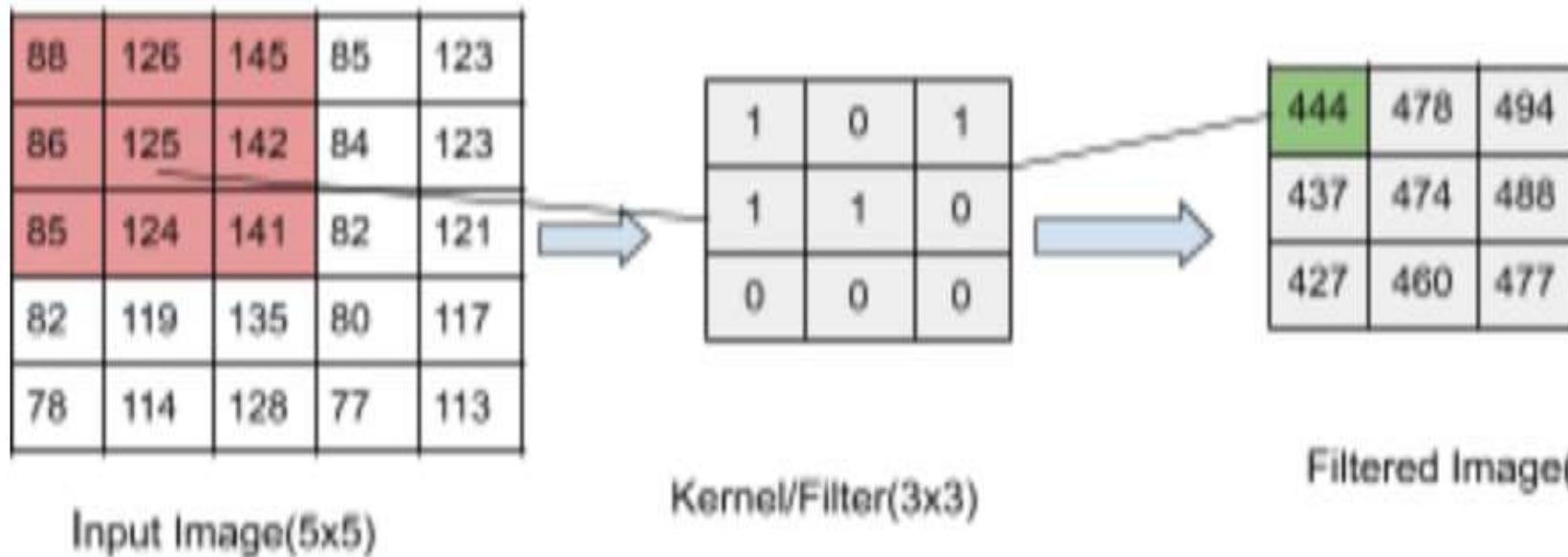
Image matrix multiplies kernel or filter matrix



- Here, Convolution is an **element-wise multiplication**.  
First, The computer will **scan a part of the image**, usually with a dimension of **3x3 and multiplies it to a filter**.
- The **output** of the element-wise multiplication is called a **feature map**. This **step is repeated until all the image is scanned**.
- Note that, after the convolution, the **size of the image is reduced**.



# CONTD..



$$\begin{aligned}
 & (88*1 + 126*0 + 145*1) + (86*1 + 125*1 + 142*0) + \\
 & (85*0 + 124*0 + 141*0) \\
 & = (88 + 145) + (86 + 125) \\
 & = 233 + 211 \\
 & = 444
 \end{aligned}$$

- █ Local Receptive Field
- █ Filtered pixel value

Input Slice (6 x 6)

20	24	11	12	16	19
19	17	20	23	15	9
21	40	25	13	14	8
9	18	8	6	11	22
31	3	7	9	17	23
20	12	3	11	19	30

Sobel Kernel

(3 x 3)

1	0	-1
2	0	-2
1	0	-1

Output (4 x 4)

3			

Sample calculation for first filter location

$$(20 \times 1) + (24 \times 0) + (11 \times -1) + \\ (19 \times 2) + (17 \times 0) + (20 \times -2) + \\ (21 \times 1) + (40 \times 0) + (25 \times -1) = 3$$

20	24	11	12	16	19
19	17	20	23	15	9
21	40	25	13	14	8
9	18	8	6	11	22
31	3	7	9	17	23
20	12	3	11	19	30

Sobel Kernel

(3 x 3)

1	0	-1
2	0	-2
1	0	-1

3	27		

Sample calculation for second filter location

$$(24 \times 1) + (11 \times 0) + (12 \times -1) + \\ (17 \times 2) + (20 \times 0) + (23 \times -2) + \\ (40 \times 1) + (25 \times 0) + (13 \times -1) = 27$$

- Different Types of Kernels or Filters: Convolution of a picture with totally **different filters** will perform operations like edge detection, blur and sharpen by applying filters.
- The **below example shows numerous convolution images once applying different types of filters (kernels).**

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

Vertical

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

Horizontal

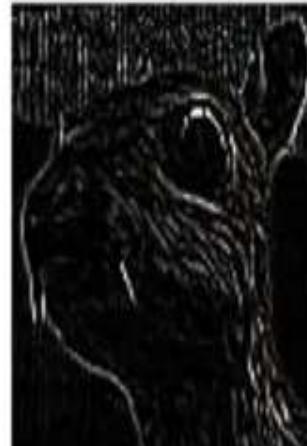
1	0	-1
2	0	-2
1	0	-1

Sobel filter

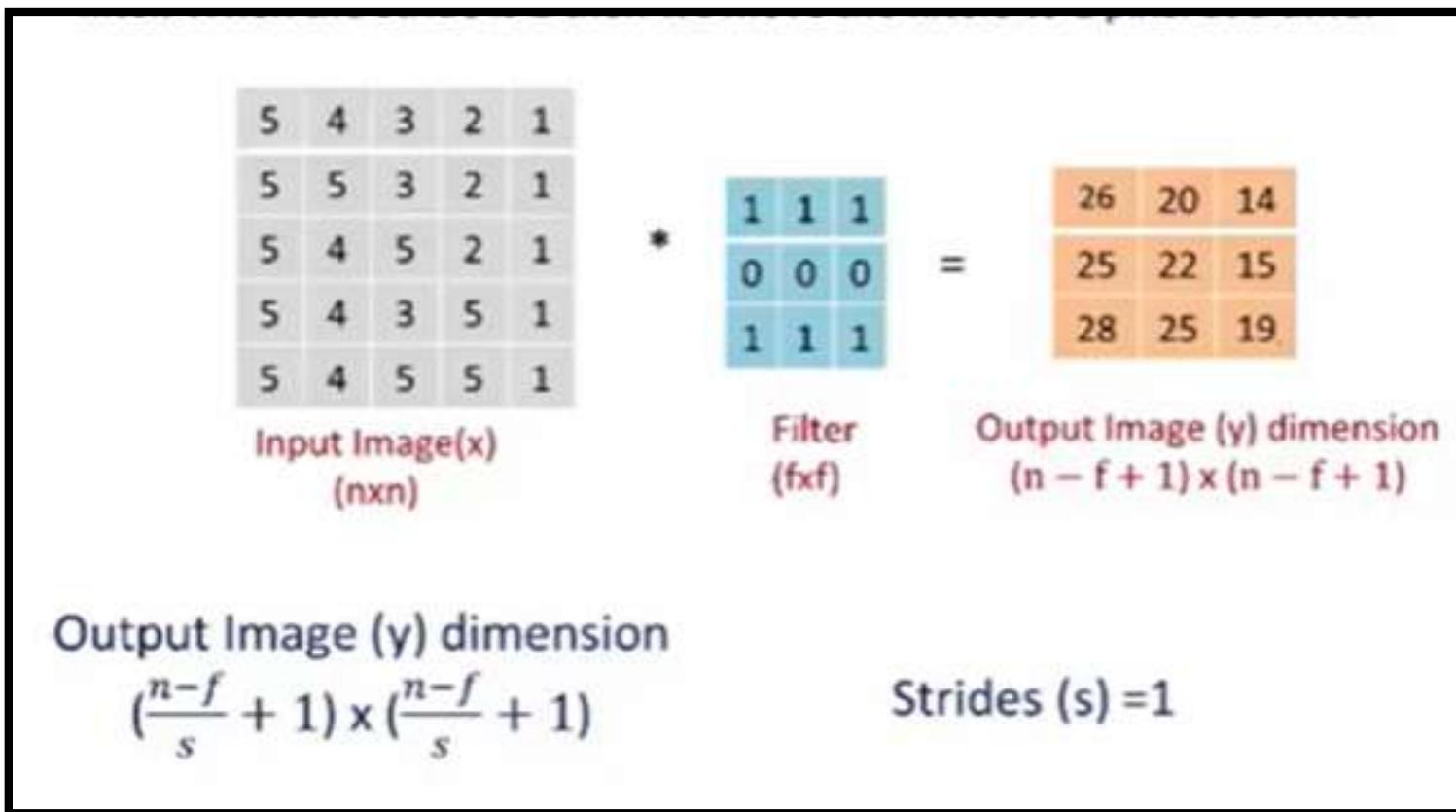
3	0	-3
10	0	-10
3	0	-3

Scharr filter

## CONTD..

<i>Original</i>	<i>Gaussian Blur</i>	<i>Sharpen</i>	<i>Edge Detection</i>
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$
			

- Important Parameters in CNN are:
- **1. Stride:** Stride determines how many pixels the kernel shifts over the input at a time.



Strides = 2

5	4	3	2	1
5	5	3	2	1
5	4	5	2	1
5	4	3	5	1
5	4	5	5	1

Input Image(x)

(n x n)

5 x 5

\*

1	1	1
0	0	0
1	1	1

Filter  
(f x f)

3 x 3

=

26	14
28	19

Output Image (y1)

$$(\frac{n-f}{s} + 1) \times (\frac{n-f}{s} + 1)$$

$$(\frac{5-3}{2} + 1) \times (\frac{5-3}{2} + 1)$$

2 x 2

Strides =3

$$\begin{array}{ccccc}
 5 & 4 & 3 & 2 & 1 \\
 5 & 5 & 3 & 2 & 1 \\
 5 & 4 & 5 & 2 & 1 \\
 5 & 4 & 3 & 5 & 1 \\
 5 & 4 & 5 & 5 & 1
 \end{array} * \begin{array}{ccc}
 1 & 1 & 1 \\
 0 & 0 & 0 \\
 1 & 1 & 1
 \end{array} = \boxed{26}$$

Input Image(x)      Filter  
 $(n \times n)$        $(f \times f)$   
 $5 \times 5$        $3 \times 3$

Output Image (y1)  
 $(\frac{n-f}{s} + 1) \times (\frac{n-f}{s} + 1)$   
 $1 \times 1$

When the convolution window continues to slide three columns to the right on the input, there is no output because the input element cannot fill the window (unless we add another column of padding).

**2. Padding:** Padding is a technique used to preserve the spatial dimensions of the input image after convolution operations on a feature map. Padding involves adding extra pixels around the border of the input feature map before convolution.



Strides =3

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	5	4	3	2	1	0	0
0	0	5	5	3	2	1	0	0
0	0	5	4	5	2	1	0	0
0	0	5	4	3	5	1	0	0
0	0	5	4	5	5	1	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

$$\begin{array}{c}
 * \\
 \text{Filter} \\
 (f \times f) \\
 3 \times 3
 \end{array}
 = 
 \begin{array}{c}
 \text{Output Image (y1)} \\
 (\frac{n+2p-f}{s} + 1) \times (\frac{n+2p-f}{s} + 1) \\
 3 \times 3
 \end{array}$$

**Input Image(x)**  
 $(n \times n)$  with  $p=2$   
 $5 \times 5$  with  $p=2$  ( $9 \times 9$ )

Dimension of the feature map as a function of the input image size(W), feature detector size(F), Stride(S) and Zero Padding on image(P) is

$$(W - F + 2P)/S + 1$$

Input image size W in our case is 5.

Feature detector or receptive field size is F, which in our case is 3

Stride (S) is 1, and the amount of zero padding used (P) on the image is 0.

so, our feature map dimension will  $(5 - 3 + 0)/1 + 1 = 3$ .

so feature map will a 3\*3 matrix with three channels(RGB).

# MOTIVATION

Convolution leverages **three important ideas** that can help improve a Deep learning system:

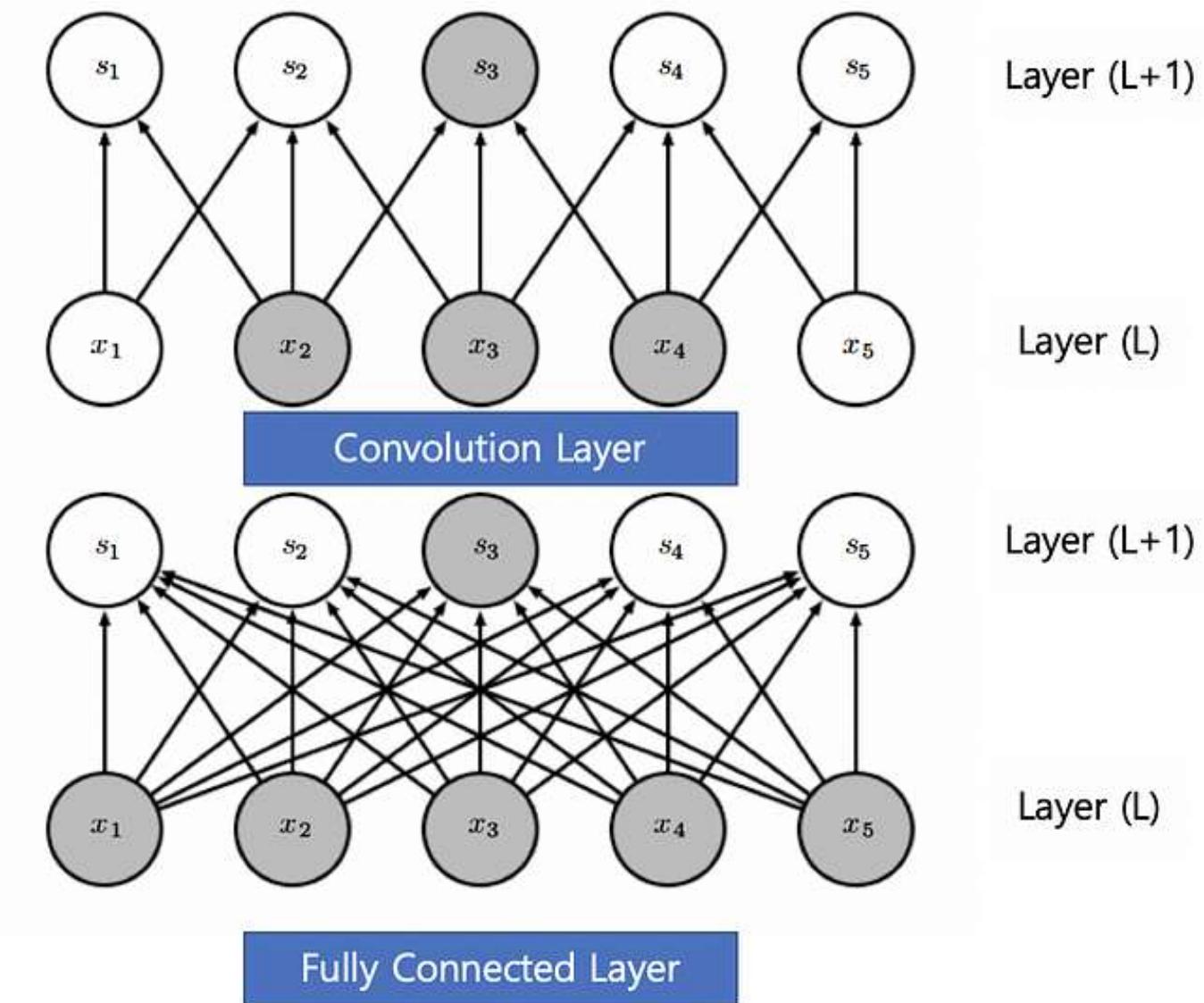
1. sparse interactions,
2. parameter sharing and
3. equivariant representations.

1. **Sparse Interactions:** In traditional Neural Networks, every output unit interacts with **every input unit**.

➤ Convolutional networks, however, typically have **sparse interactions**, by making **kernel smaller than input**.



- For example, when processing an image, the input image might have thousands or millions of pixels, but we can detect small, meaningful features such as edges with kernels that occupy only tens or hundreds of pixels.
- So that it finally Reduces memory requirements and Improves statistical efficiency
- These improvements in efficiency are usually quite large. If there are m inputs and n outputs, then matrix multiplication requires  $m \times n$  parameters and the algorithms used in practice have  $O(m \times n)$  runtime (per example).



- If we **limit the number of connections** each output may have to  $k$ , then the **sparsely connected** approach requires only  $k \times n$  parameters and  $O(k \times n)$  runtime.
- In **layers connected by convolution**, however, only **need 3 weights** to produce  $s_3$  as only **three hidden units  $x_2, x_3, x_4$**  are **contributing** to  $s_3$ .
- Compared to the  $s_3$  with **FC layer**, the  $s_3$  with **convolution** has fewer connections in the hidden units of the previous layer ( $L$ ), and this means it has a sparser connection than FC.



**2. Parameter Sharing:** Parameter sharing refers to using same parameter for more than one function in a model.

- In **convolutional neural net**, each member of kernel is used at **every position of input** i.e. parameters used to compute different output units are tied together (**all times their values are same**).
- Sparse interactions and **parameter sharing** combined can improve efficiency of a **linear function** for detecting edges in an image

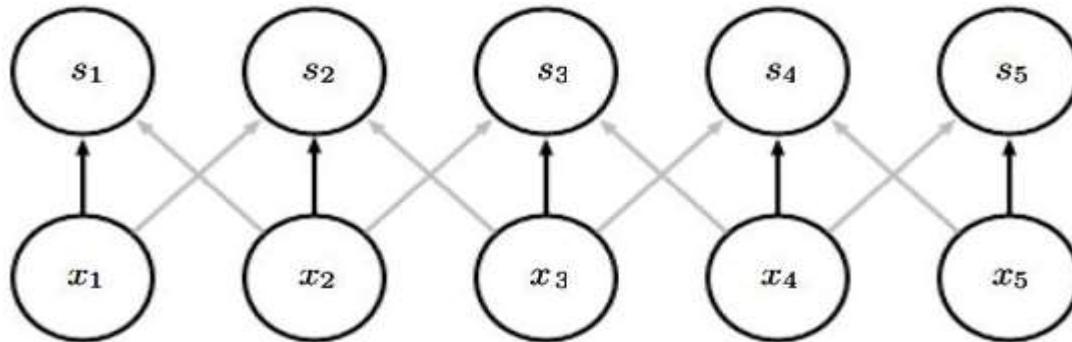


Fig. shows convolution shares the same parameters across all spatial locations.

**3. Equivariant Representation:** Convolution function is equivariant to translation. This means that shifting the input and applying convolution is equivalent to applying convolution to the input and shifting it.

*A function  $f$  is said to be equivariant to a function  $g$  if*

$$f(g(x)) = g(f(x))$$

*i.e. if input changes, the output changes in the same way.*

**Fig. shows convolution shares the same parameters across all spatial locations.**

Example of equivariance: With **2D images convolution** creates a map where certain features appear in the input.

If we move the object in the input, the representation will move the same amount in the output. It is useful to detect edges in first layer of convolutional network. Same edges appear every where in image, so it is practical to share parameters across entire image.



# ACTIVATION FUNCTION USED IN CNN

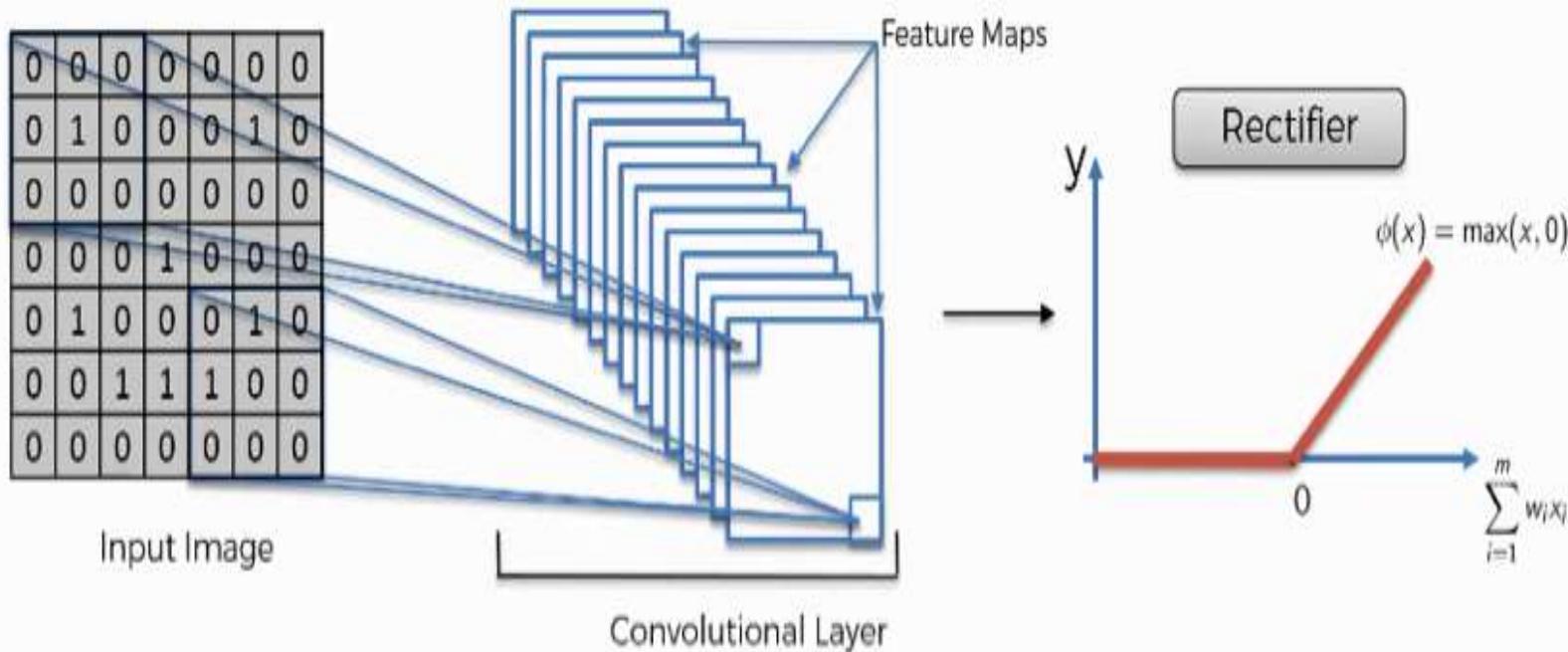
- The output volume of the Convolution layer is fed to an element wise activation function, commonly a Rectified-Linear Unit (ReLU).
- After each CONV layer in a CNN, we apply a nonlinear activation function, such as ReLU. The Rectified Linear Unit, or ReLU, is not a separate component of the convolutional neural networks' process.
- The ReLu layer will determine whether an input node will 'fire' given the input data.



# ACTIVATION FUNCTION USED IN CNN

- This 'firing' signals whether the convolution layer's filters have detected a visual feature.
- The dimensions of the volume are left unchanged.
- The **ReLU** (short for rectified linear units) layer commonly follows the **convolution layer**. i.e. the ReLU layer allows the convnet to account for situations in which the relationship between the pixel value inputs and the convnet output is not linear.





The purpose of applying **the rectifier function** is **to increase the non-linearity in our images**. The **ReLU function** takes **a value x** and returns **0** if **x is negative** and **x if x is positive**.

## Input

-249	-91	-37
250	-134	101
27	61	-153

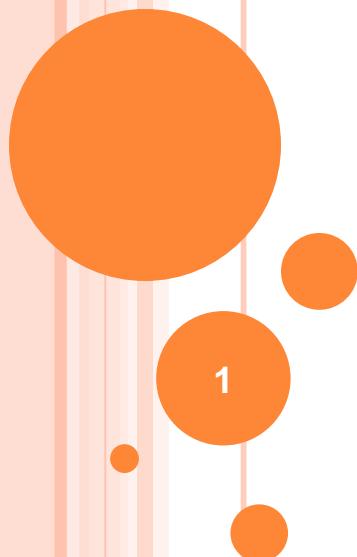
## ReLU

0	0	0
250	0	101
27	61	0

As you can see from the graph, the ReLU function is nonlinear. In this layer, the **ReLU function is applied to each point in the feature map.** The result is a feature map **without negative values.**

## **UNIT-III**

# **Convolutional Neural Networks**



# CONTENTS

## ❖ Pooling Layer

# POOLING LAYER

- Pooling layers are one of the building blocks of Convolutional Neural Networks.
- Where Convolutional layers extract features from images, Pooling layers consolidate the features learned by CNNs.
- we use a pooling function to modify the output of the layer further. Its purpose is to gradually shrink the representation's spatial dimension to minimize the number of parameters and computations in the network.
- Pooling layers are used to reduce the dimensions of the feature maps.

- i.e, The pooling layer summarizes the features **present in a region of the feature map** generated by a **convolution layer**. This makes the model **more robust to variations** in the position of the features in the input image.
- The **size of the pooling operation or filter** is **smaller than the size of the feature map**. This means that the pooling layer will **always reduce the size of each feature map by a factor of 2**,
- **Eg:** **Each Dimension is halved**, reducing the number of pixels or values in **each feature map** to **one quarter the size**.



- For example: a pooling layer applied **to a feature map** of  **$6 \times 6$  (36 pixels)** will result in an **output pooled feature map** of  **$3 \times 3$  (9 pixels)**.

The **pooling operation**, also called **subsampling** is used to reduce the **dimensionality** of feature maps from the **convolution operation**.

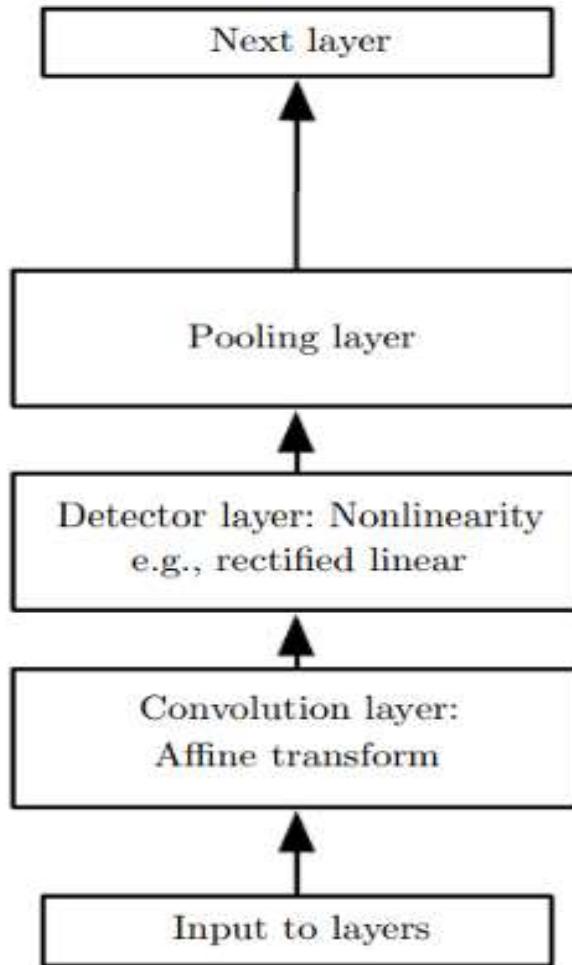
- Pooling layers, also known as **down sampling**, conducts **dimensionality reduction**, reducing the number of parameters in the input.



- Similar to the **convolutional layer**, the **pooling operation** sweeps a filter across the entire input, but the difference is that this **filter does not have any weights**. Instead, the kernel applies an aggregation function to the values within the receptive field, populating the output array.
- Types of Pooling Layer:
  - 1. Max Pooling
  - 2. Average Pooling
  - 3. Min Pooling



Simple layer terminology



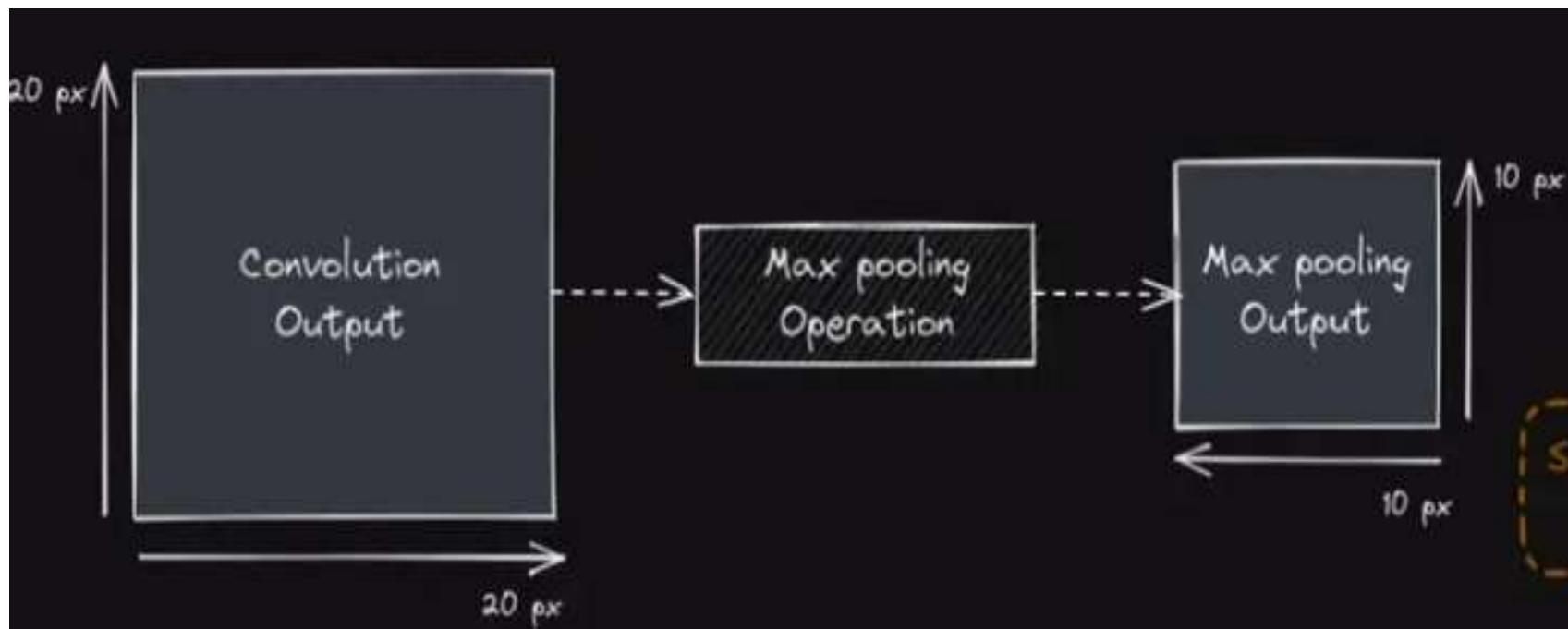
**Fig: Components of a typical Convolutional Neural Network**

**1. Max Pooling:** Finally, we can say it **selects the maximum valued element** from the **region captured by the filter** in **any feature map**.

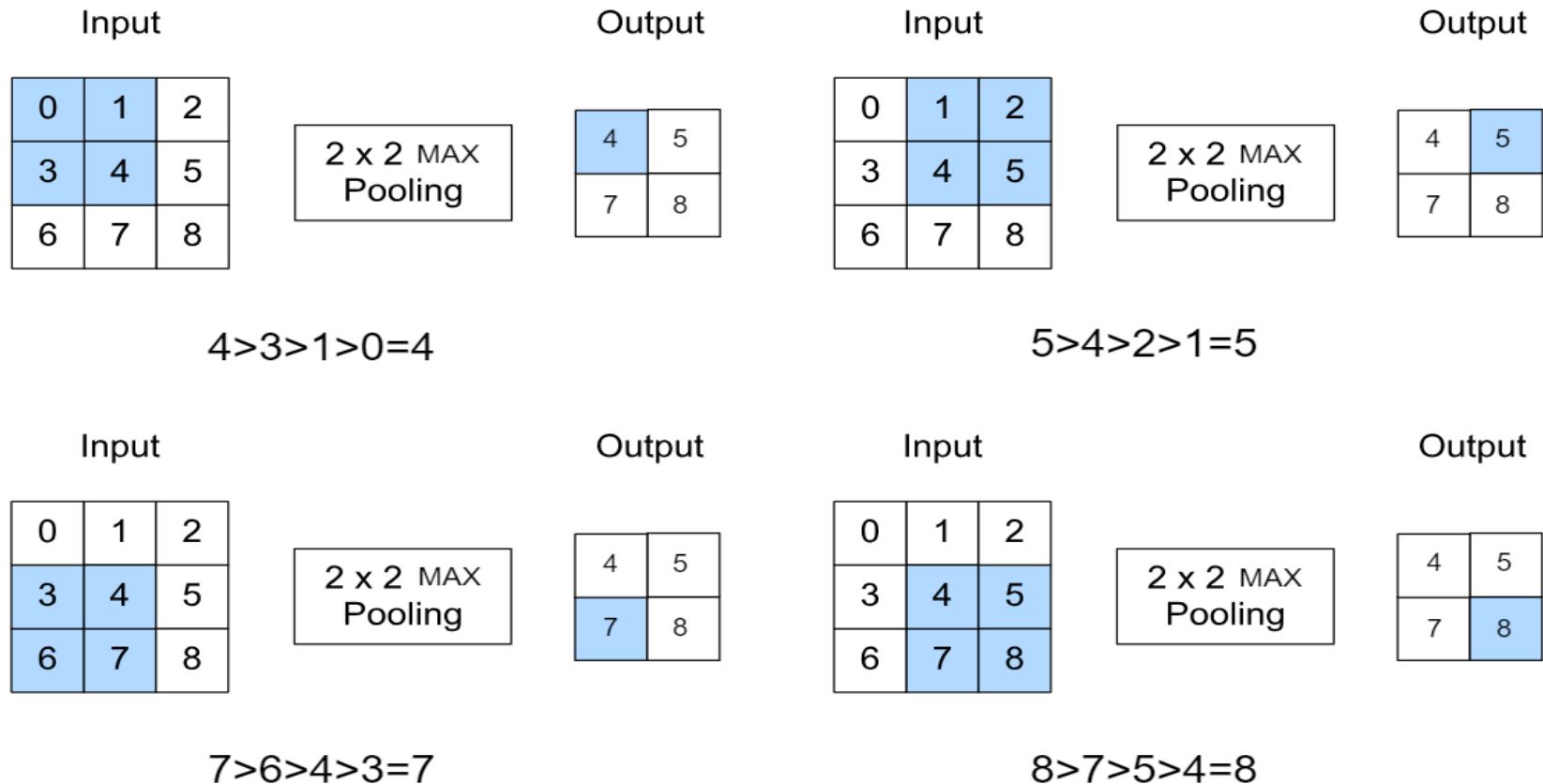
This works by selecting the **maximum value from every pool**. Max Pooling **retains the most prominent features** of the **feature map**, and the **returned image is sharper than the original image**.

- i.e, if we talk about **image processing** the **max-pooling** helps to extract the sharpest features on the image.
- The **Max Pooling layer** summarizes the features in a region represented by the **maximum value in that region**. Max Pooling is more suitable for images with a **dark background** as it will select brighter pixels in a region of the input image.

Kernel/filter is the size of a matrix that is applied over the complete input data. As the filter moves across the input, it selects the pixel with the maximum value to send to the output array.



For Eg:



- Mathematical Formulae: The pooling layer requires 2 hyperparameters
  1. Kernel/filter size F and
  2. Stride S.
- On applying the **pooling layer** over the input volume, output dimensions of output volume will be

$$((n-f)/s + 1) \times ((n-f)/s + 1)$$



## Max Pooling Input & Filter

$4 \times 4$  Input

4	3	8	5
9	1	3	6
6	3	5	3
2	5	5	2

$2 \times 2$  Filter



## Max Pooling Operation

$4 \times 4$  Input

4	3	8	5
9	1	3	6
6	3	5	3
2	5	5	2

Pool 1  
Output

9

## Max Pooling Output

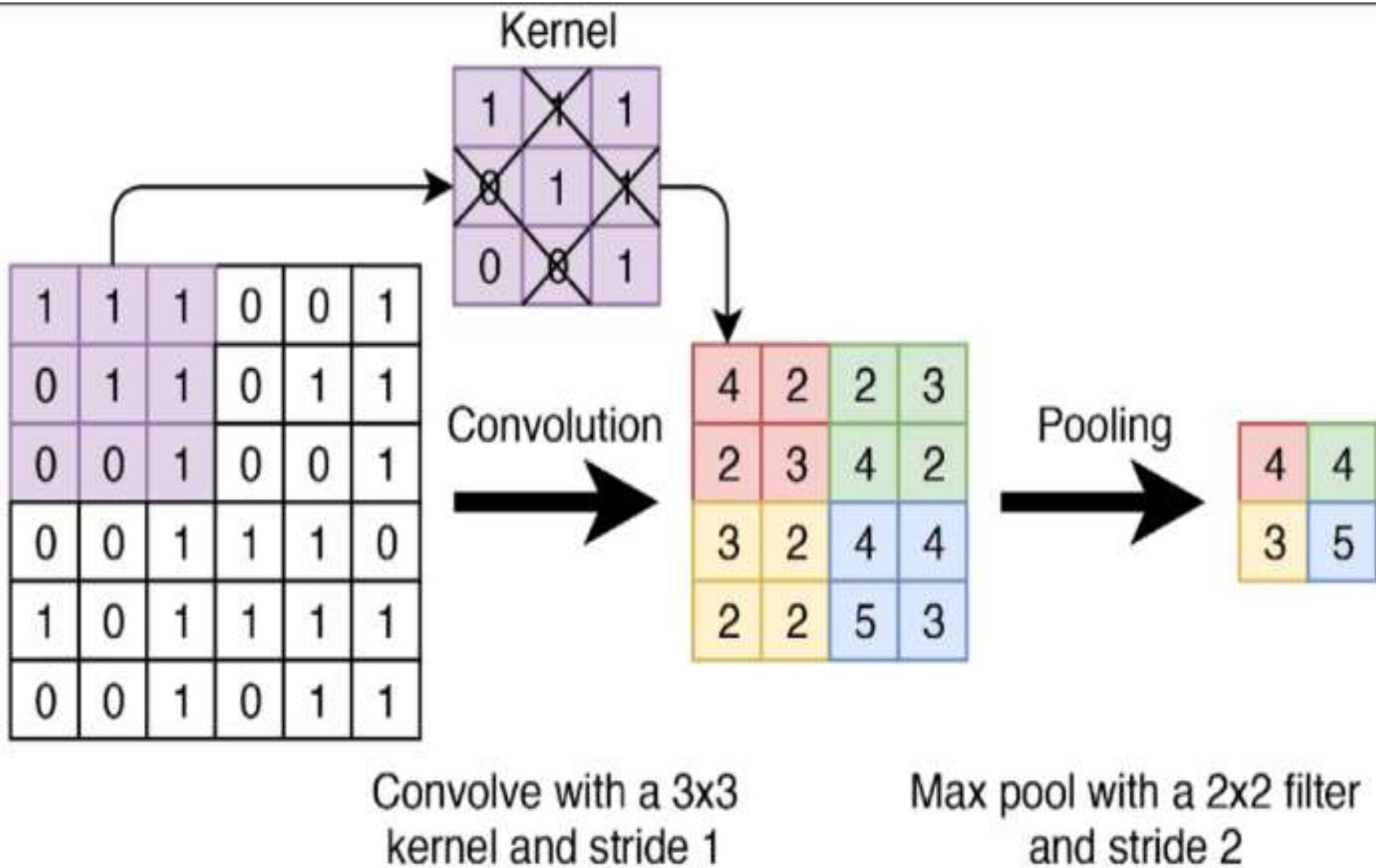
$4 \times 4$  Input

4	3	8	5
9	1	3	6
6	3	5	3
2	5	5	2

Final Output

9	8
6	5

For  $n \times n$  input and  $f \times f$  filter with stride  $s$ , the output dimensions are:  $\left(\frac{n-f}{s} + 1\right) \times \left(\frac{n-f}{s} + 1\right)$



- **2. Average pooling :** As the filter moves across the input, it calculates the average value within the receptive field to send to the output array.
- This **pooling layer** works by getting the **average of the pool**. Average pooling retains the **average values of features** of the feature map.
- Average pooling method **smooth's out the image** and hence the **sharp features may not be identified** when this pooling method is used.



- With average pooling, the harsh edges of a picture are **smoothed**, and this type of pooling layer can be used **when harsh edges can be ignored**.

Rectified feature map

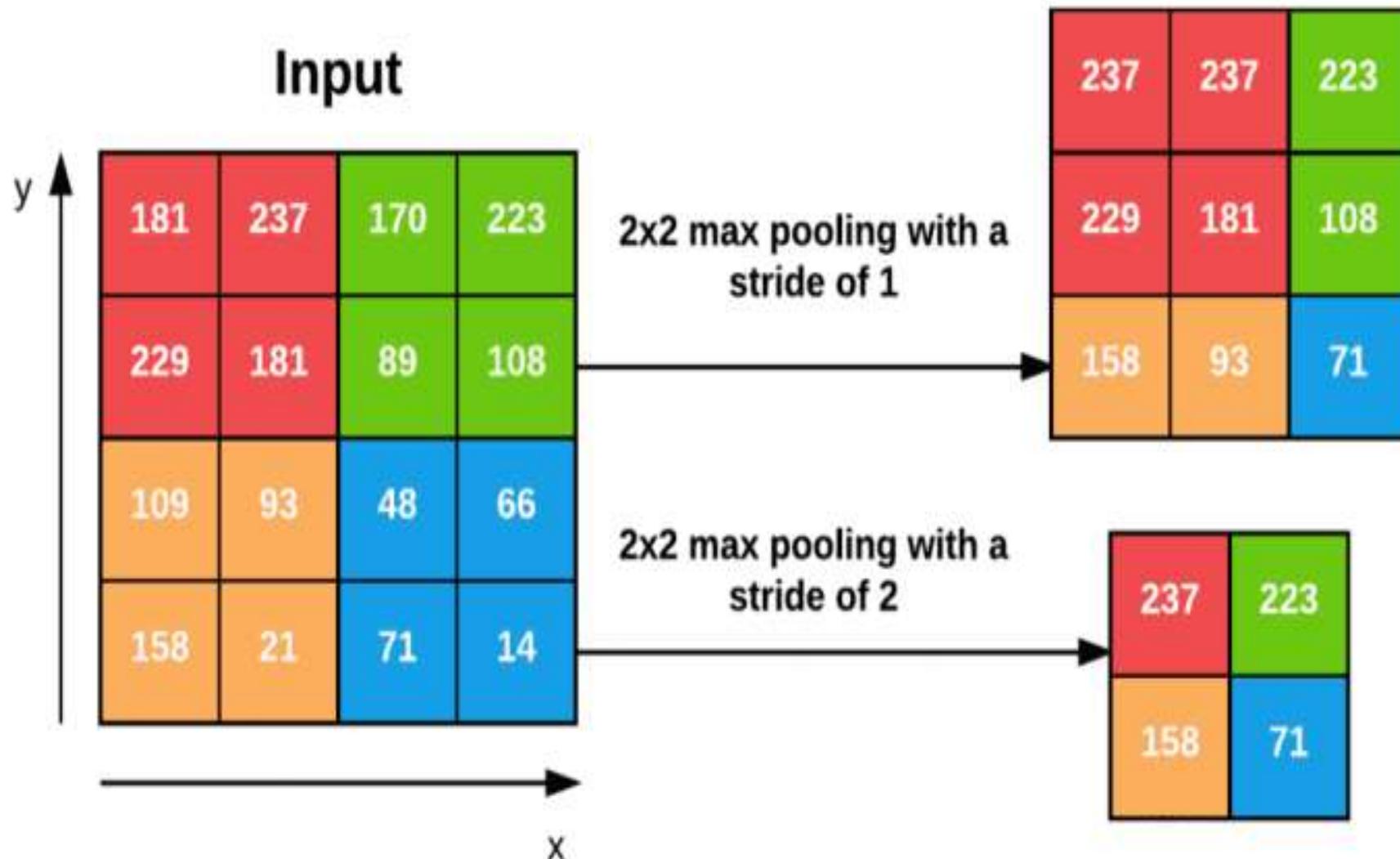
1	4	2	7
2	6	8	5
3	4	0	7
1	2	3	1

Average pooling with  $2 \times 2$  filters and stride 2

Pooled feature map

3.2	6.2
2.5	2.7

$$\text{Average}(3, 4, 1, 2) = 2.5$$



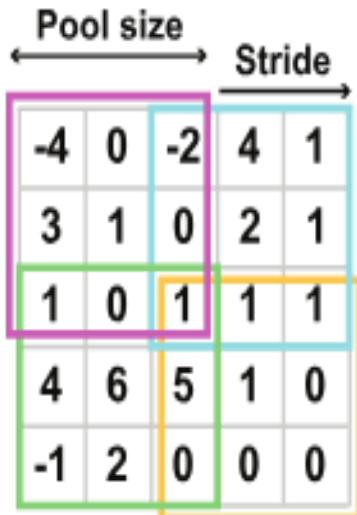
- **3. Min pooling :** In this type of pooling, the **summary of the features in a region** is represented by the **minimum value** in that region.
- It is mostly used when the image has **a light background** .

255	167	25	46
2	70	251	58
8	59	68	90
4	125	201	126

**Filter Size:**  
 $2 \times 2$   
→  
**Stride: 2**

2	25
4	68





Features

Max Pooling



Min Pooling

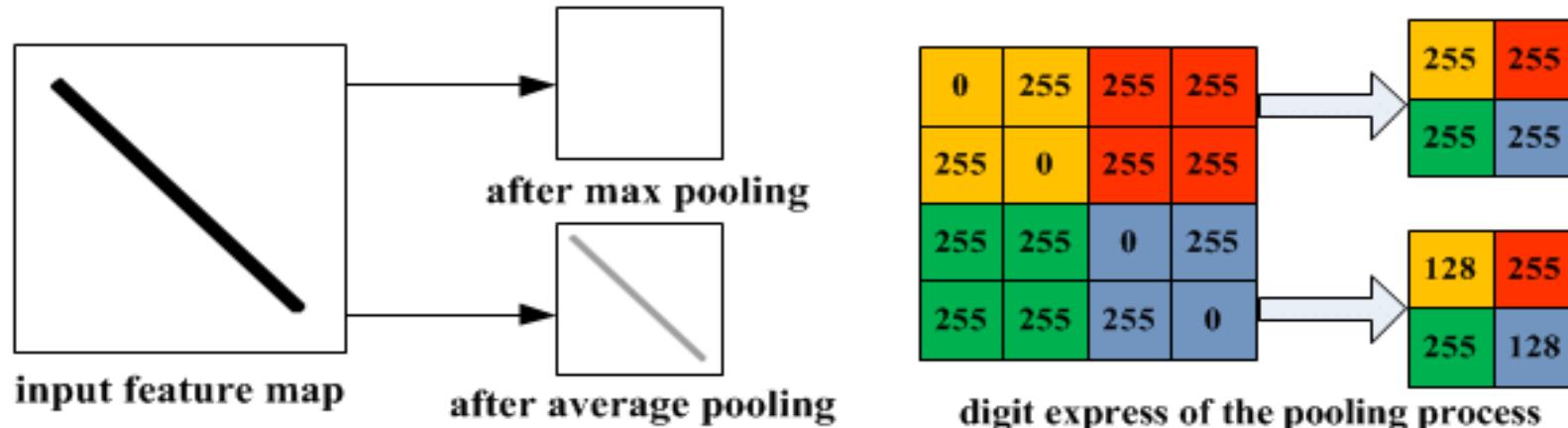


Average  
Pooling

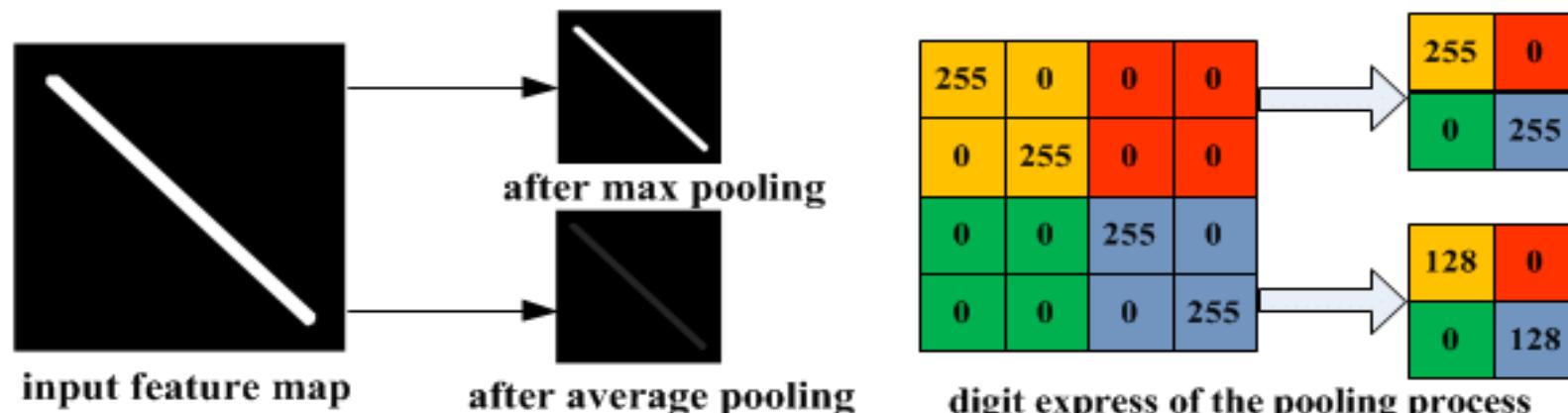


Output

## Some drawbacks in Selection of Pooling Layer:



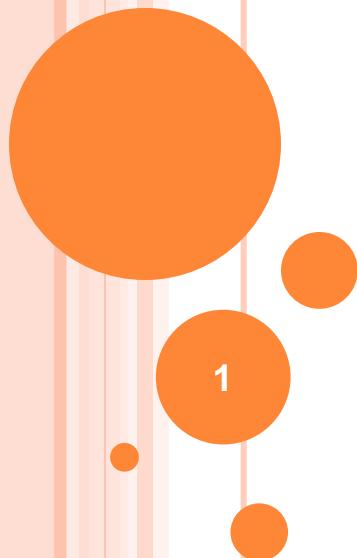
(a) Illustration of max pooling drawback



(b) Illustration of average pooling drawback

## **UNIT-III**

# **Convolutional Neural Networks**



# CONTENTS

- ❖ Variants of the Basic Convolution Function
- ❖ Structured Outputs
- ❖ Data Types
- ❖ Types of Convolutions

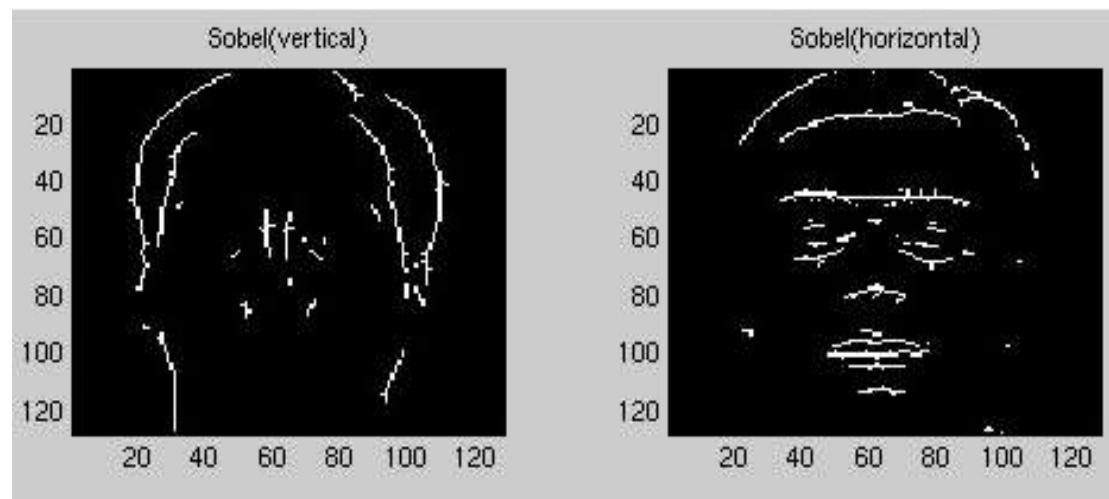
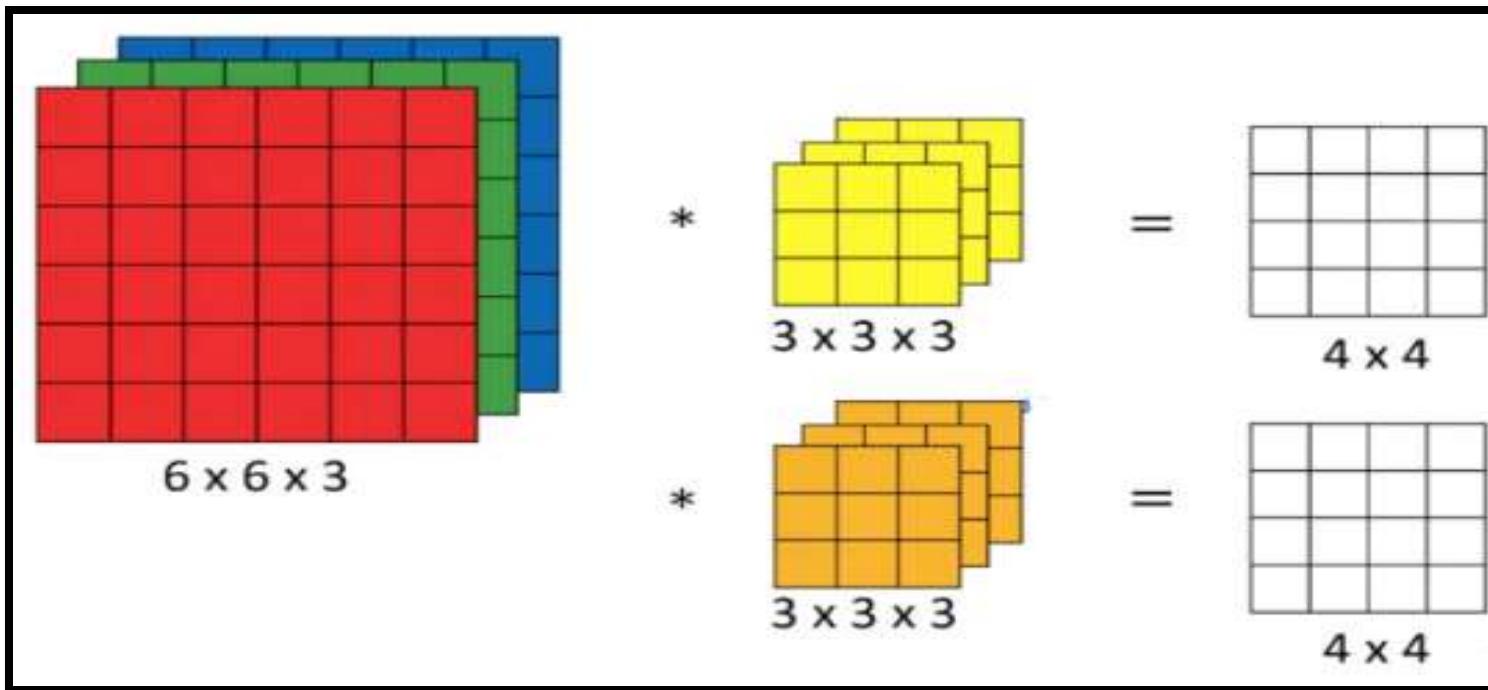
## VARIANTS OF THE BASIC CONVOLUTION FUNCTION

- In **practical implementations** of the **convolution operation**, certain modifications are made which **deviate** from standard discrete convolution operation -
  - 1. In general **a convolution layer** consists of **application of several different kernels to the input**. This allows the **extraction of several different features at all locations in the input**.
  - This means that in **each layer**, a **single kernel (filter) isn't applied**. **Multiple kernels (filters)**, are used as **different feature detectors**.



- **Convolution Operations with Multiple Filters:** Multiple filters can be used in a **convolution layer** to detect multiple features.
- Let's say the **first filter will detect vertical edges** and the **second filter will detect horizontal edges** from the image.
- If we use **multiple filters**, the **output dimension will change**.
- So, instead of having a **4 X 4 output** , we would have a **4 X 4 X 2 output (if we have used 2 filters)**
- The output of the layer then will have the **same number of channels** as the **number of filters in the layer**.

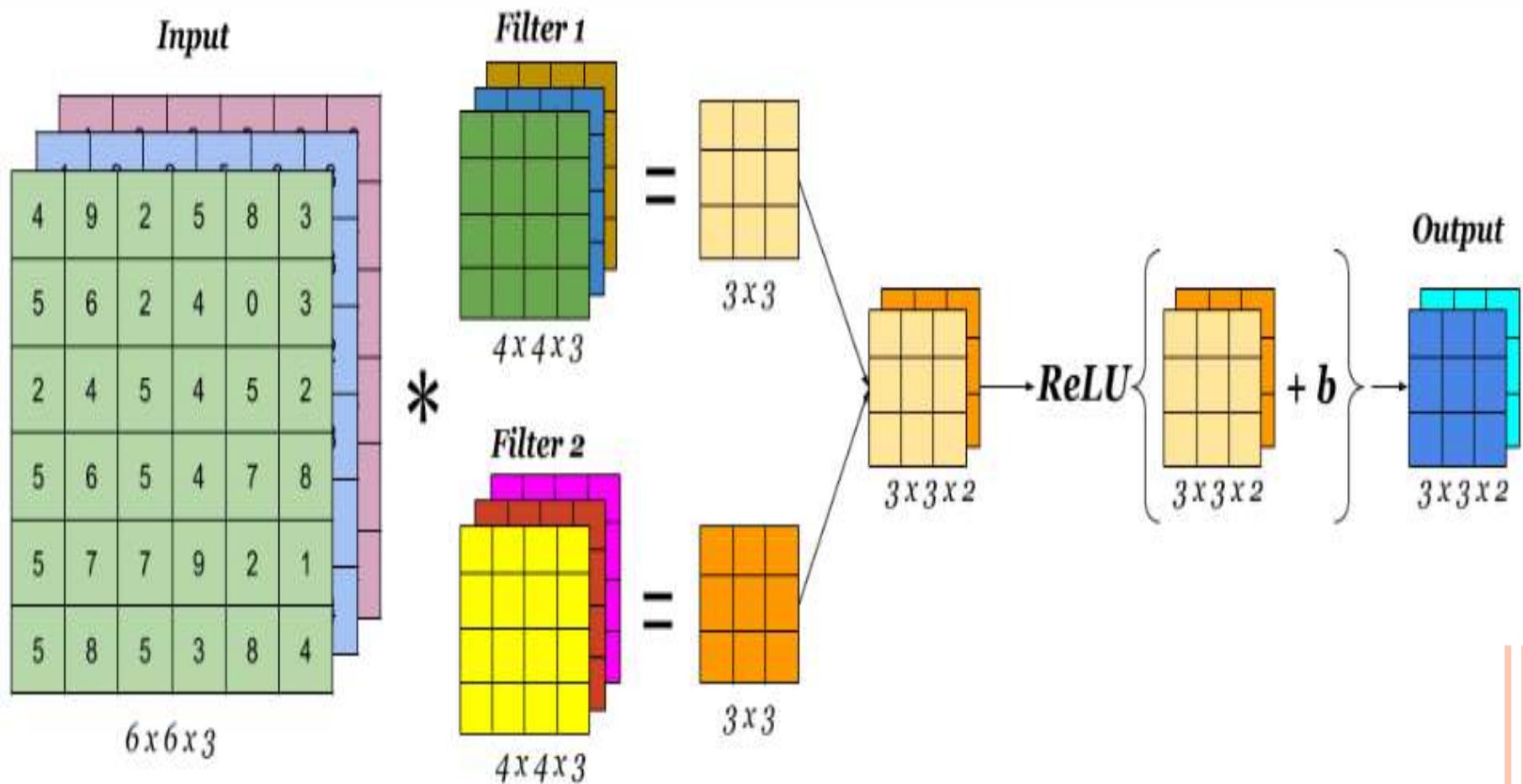




Generalized dimensions can be given as:

- **Input:**  $n \times n \times n_c$
- **Filter:**  $f \times f \times n_c$
- **Padding:**  $p$
- **Stride:**  $s$
- **Output:**  $[(n+2p-f)/s+1] \times [(n+2p-f)/s+1] \times n_c'$

Here,  $n_c$  is the number of channels in the input and filter, while  $n_c'$  is the number of filters.

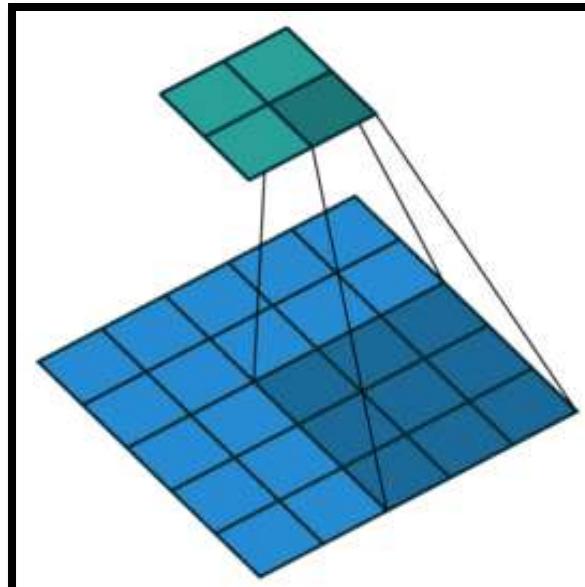
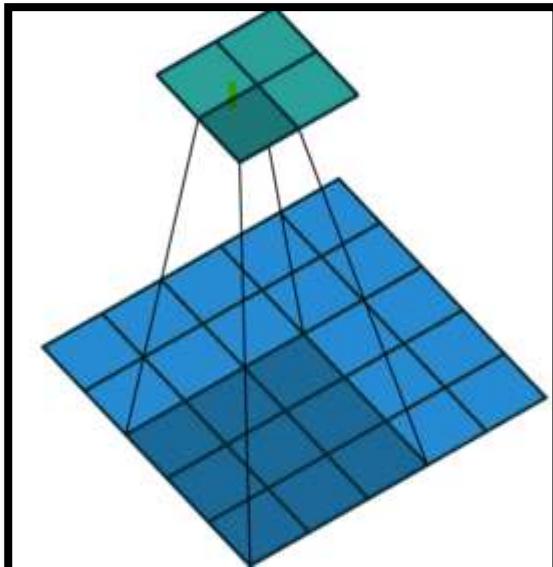
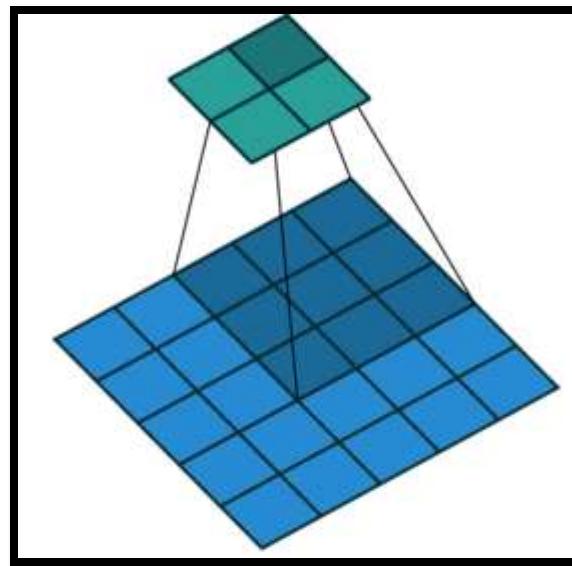
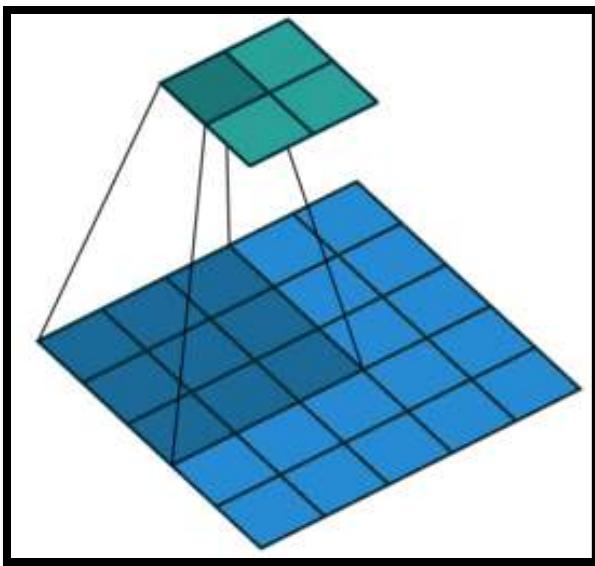


- 2. The input is generally **not real-valued** but instead **vector valued**.
- 3. In order to allow for **calculation of features** at **strided convolutions** can be used. strided convolution introduces a **stride variable** that **controls the step of the folder** as it moves over the **input**. The **effect of strided convolution** is the same as that of a **convolution followed by a down sampling stage**.
- **Down sampling** aims to **reduce the file size** and potentially simplify the image **by removing unnecessary details**.



- **Effects of Strides:** Stride is the number of pixels shifts over the input matrix.
- In order to allow for **calculation of features** at strided convolutions can be used.
- **Strides** can be used to **reduce the representation size**.
- Below is an example representing **2-D Convolution**, with **(3 \* 3) Kernel** and **Stride of 2 units**.



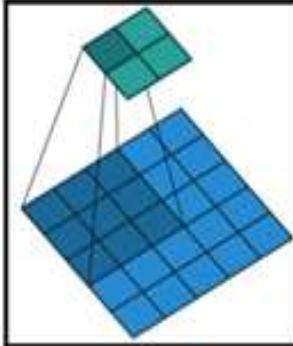
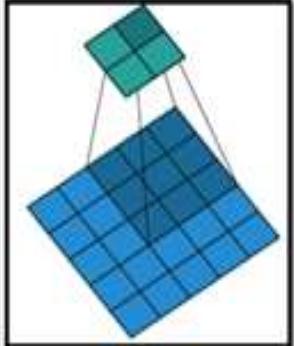
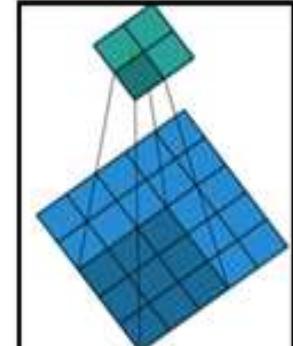
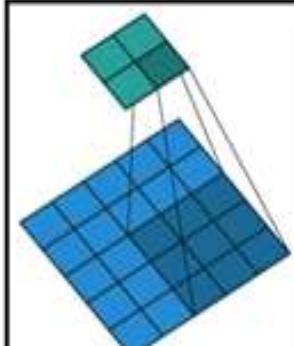


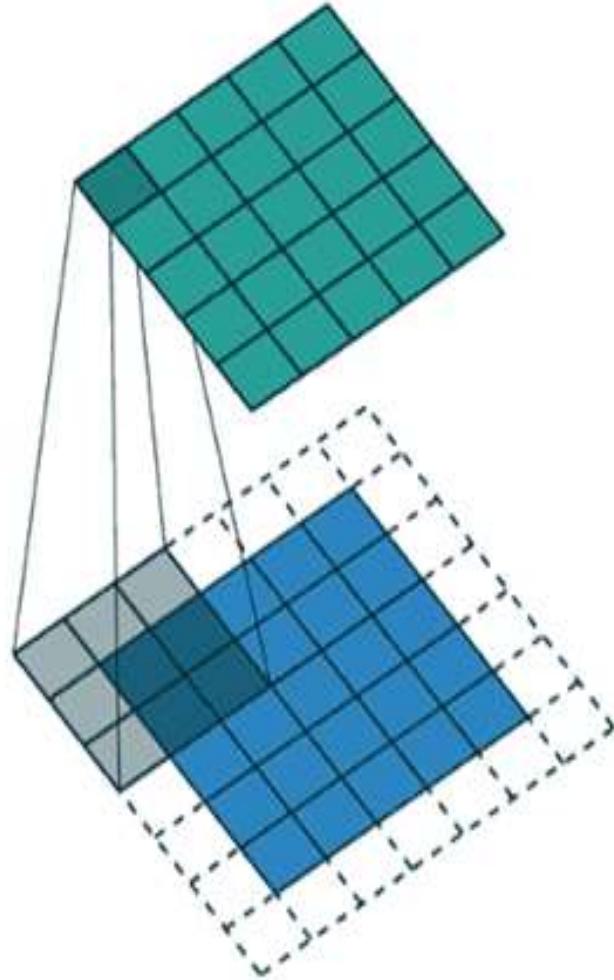
- **Effects of Zero Padding:** Convolution networks can implicitly zero pad the input **V**, to make it wider.
- The most common padding value is **zero-padding**, which involves **adding zeros** to the **borders of the input feature map**. This helps in **reducing the loss of information at the borders of the input feature map** and can improve the performance of the model.
- Zero padding the input allows to **control kernel width** and **size of output independently**.

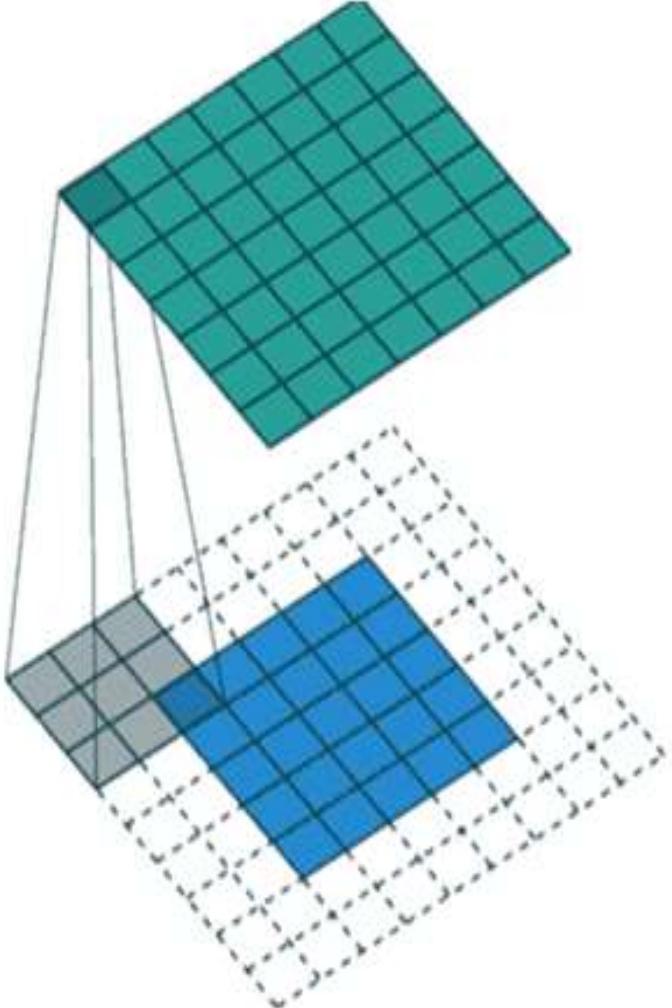


## ➤ Zero Padding Strategies: Three Common Zero Padding

Strategies are:

Zero Padding Type	Properties	Example
Valid Zero-Padding	<ul style="list-style-type: none"> <li>1. No zero padding is used.</li> <li>2. Output is computed only at places where entire kernel lies inside the input.</li> <li>3. Shrinkage <math>&gt; 0</math></li> <li>4. Limits #convolution layers to be used in network</li> <li>5. Input's width = <math>m</math>, Kernel's width = <math>k</math>, Width of Output = <math>m-k+1</math></li> </ul>	   

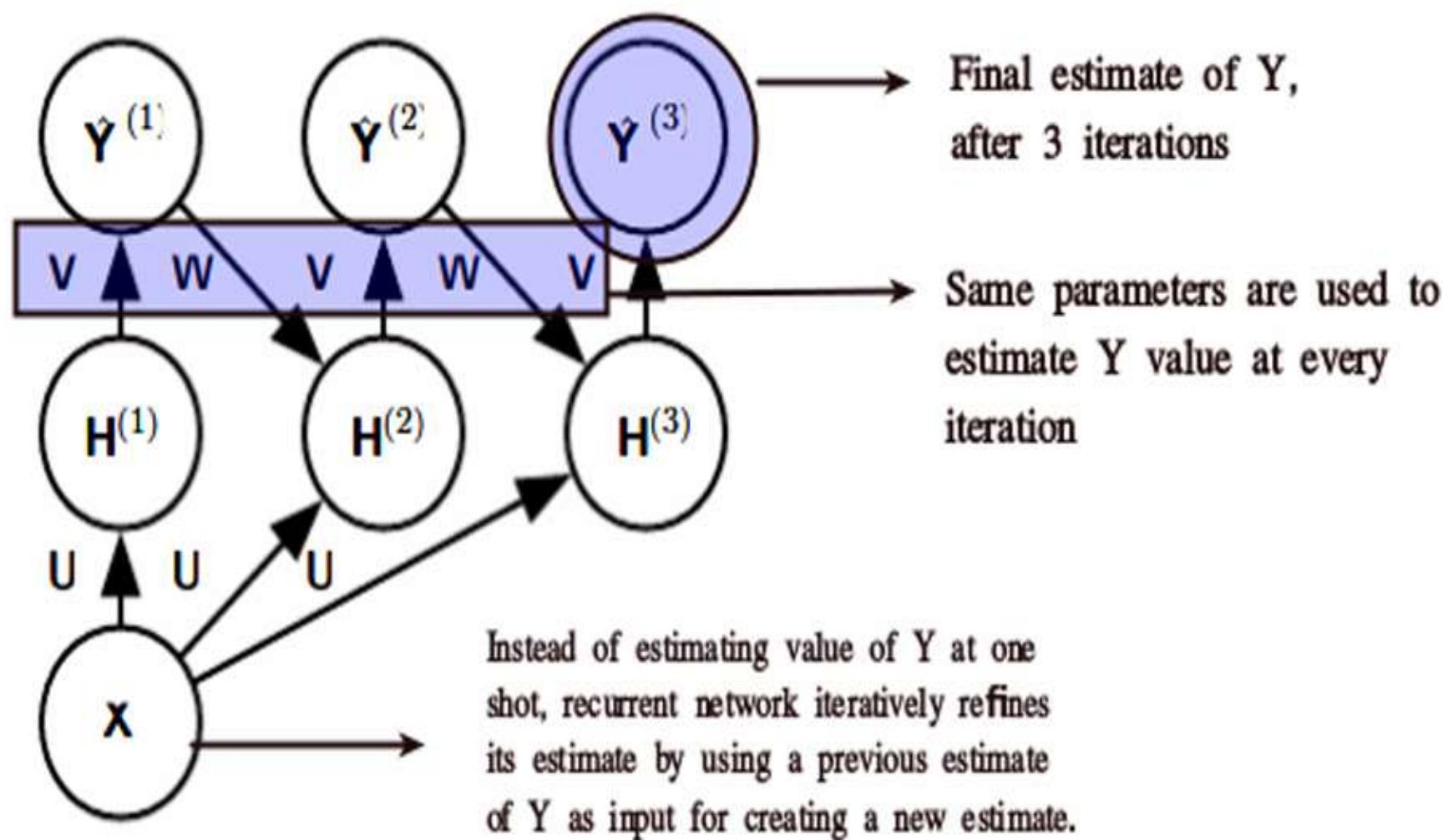
Zero Padding Type	Properties	Example
Same Zero-Padding	<p>1. Just enough <b>zero padding</b> is <b>added</b> to keep:</p> <p>1.a. <b>Size(Ouput) = Size(Input)</b></p> <p>2. Since the <b>#output units connected to border pixels</b> is less than that for centre pixels, it may <b>under-represent border pixels.</b></p> <p>3. Can <b>add as many convolution layers</b> as hardware can support</p> <p>5. Input's width = <math>m</math>, Kernel's width = <math>k</math>,</p> <p><b>Width of Output = <math>m</math></b></p>	

Zero Padding Type	Properties	Example
Strong Zero-Padding	<p>1. The input is padded by enough zeros such that <b>each input pixel is connected to same #output units.</b></p> <p>2. Allows us to make an <b>arbitrarily deep NN.</b></p> <p>3. Can <b>add as many convolution layers</b> as hardware can support</p> <p>4. Input's width = <math>m</math>, Kernel's width = <math>k</math>,</p> <p><b>Width of Output = <math>m + k - 1</math></b></p>	

# STRUCTURED OUTPUTS

- Convolutional networks can be trained to output high-dimensional structured output rather than just a classification score.
- To produce an output map as same size as input map, only same-padded convolutions can be stacked.
- The output of the first labelling stage can be refined successively by another convolutional model.
- If the models use tied parameters, this gives rise to a type of recursive model as shown below. ( $H^1, H^2, H^3$  share parameters or Hidden Layers)





Variable	Description
X	Input image tensor
Y	Output of Classification or Object Detection
H	Hidden representation
U	Kernel
V	Produce Estimated Labels
W	Output Y to provide the input to H

- The **input layer** receives the raw data, such as **an image**, and serves as the starting point for processing.
- Then the so-called **Hidden Layers**: here the Hidden Layers are **Convolutional Layer, Activation Function, Pooling Layer**.

- The term "**hidden layers**" in a CNN refers to the **intermediate layers** between the input and output layers. Although the layers are not directly observable from the input or output, you have the **authority to shape their structure and functionality** .
- By defining **the convolutional layers, activation functions, pooling layers**, you guide the network's ability to **extract meaningful features** and make accurate predictions.



# DATA TYPES

- The **data** used with a **convolutional network** usually consist of **several channels**, each channel being the observation of a **different quantity at some point in space or time**.
- When output is **variable sized**, **no extra design change** needs to be made.
- When output requires **fixed size** (classification), a **pooling stage** with **kernel size proportional to input size** needs to be used.



# DATA TYPES

Dimensions	Single channel
1D	<b>Raw audio</b> (single amplitude value per time point)
2D	<b>Audio spectrogram</b> (one FFT coefficient per time point per frequency)
3D	<b>CT scan</b> (one value per (x,y,z) tuple)

Dimensions	Multichannel
1D	<b>Skeleton animation data</b> (orientation of each joint)
2D	<b>Color image</b> (RGB triplet per (x,y) tuple)
3D	<b>Color video</b> (one RGB triplet per (x,y) tuple per time instant)

Different data types based on the number of spatial dimensions & channels

# **TYPES OF CONVOLUTIONS**

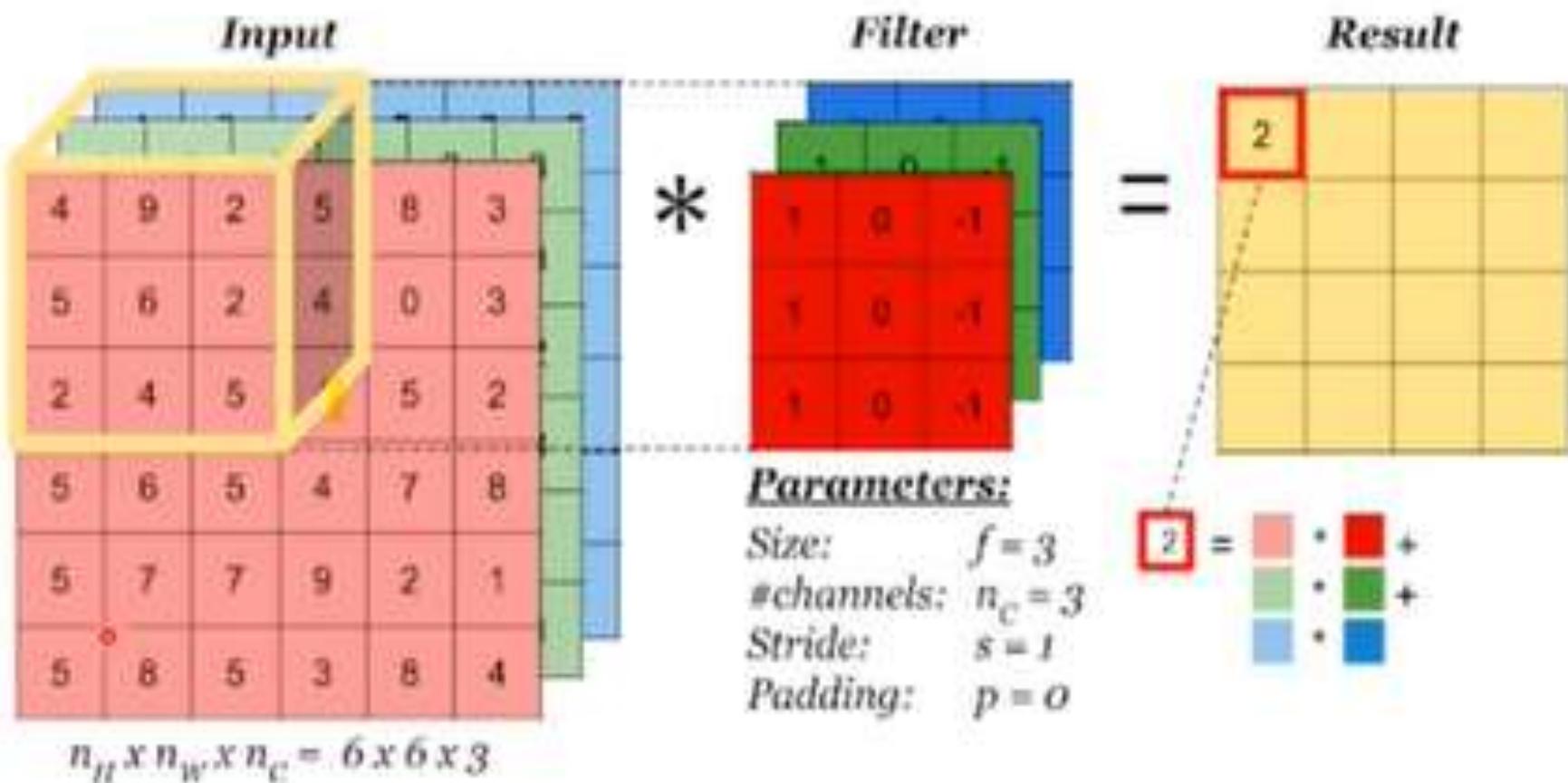
- There are mainly 3 Types of Convolutions are there. Those are:
  - 1. Locally Connected Layer / Un Shared Convolution
  - 2. Tiled Convolution
  - 3. Traditional Convolution
  - 4. Comparing Computational Times.

## 1. Unshared Convolution:

- ❖ The **connectivity graph of convolution operation** and **locally connected layer is the same**
- ❖ The only difference is that **parameter sharing is not performed.**

- ❖ i.e, **Each output unit** performs **a linear operation** on its neighbourhood but **parameters are not shared across output Units.**
- ❖ This allows models to **capture local connectivity** while allowing **different features** to be computed at **different spatial locations.**
- ❖ This however requires **much more parameters** than the **convolution operation.**





- ❖ In the Unshared Convolution, every connection has its own weight.

*i*, the output channel,  
*j*, the output row,  
*k*, the output column,  
*l*, the input channel,  
*m*, the row offset within the input, and  
*n*, the column offset within the input.

- ❖ The Linear Part of the Locally Connected Layer is given by

$$Z_{i,j,k} = \sum_{l,m,n} [V_{l,j+m-1,k+n-1} w_{i,j,k,l,m,n}]$$



- ❖ Advantages:

- ❖ 1. Reducing memory consumption
- ❖ 2. Increasing statistical efficiency
- ❖ 3. Reducing the amount of computation needed to perform forward and back-propagation.

- ❖ Disadvantages:

- ❖ 1. requires much more parameters than the convolution operation.



## 2. Tiled Convolution:

- ❖ It Offers a **compromise** b/w **unshared** and **traditional convolution**.  
i.e, Rather than learning a **separate set of weights** at every spatial location, we learn a **set of kernels** that **we rotate through** as we move through space.
- ❖ This means that **immediately neighboring locations** will **have different filters**, like in a locally connected layer, Makes **use of parameter sharing**.



- Cycle between shared parameter groups

Input  $I(:,:,)$

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

A set of filters (e.g. 2)

Filter  $K_0(:,:,)$

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

Filter  $K_1(:,:,)$

1	0	1
-1	1	1
0	0	-1

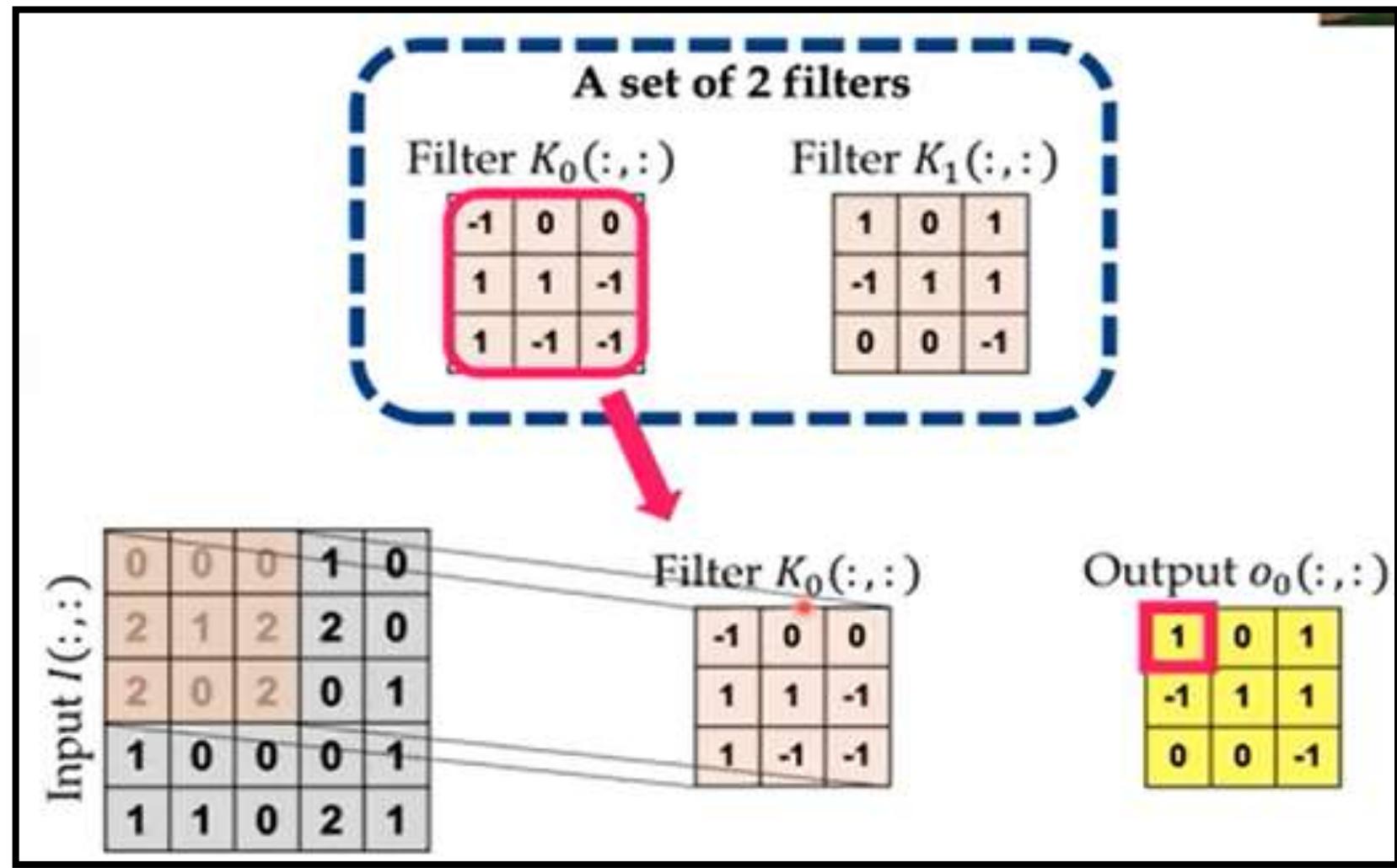
- In this Image consists of K0 and K1 Filters.
- Here we can use the **First Filter K0** with **First portion of the Input Image**. And **Second Filter K1** with **Second Portion of the Input Image** with **Stride -1**.
- Again we have to use K0 with 3<sup>rd</sup> Portion of the Image etc....

❖ Advantages:

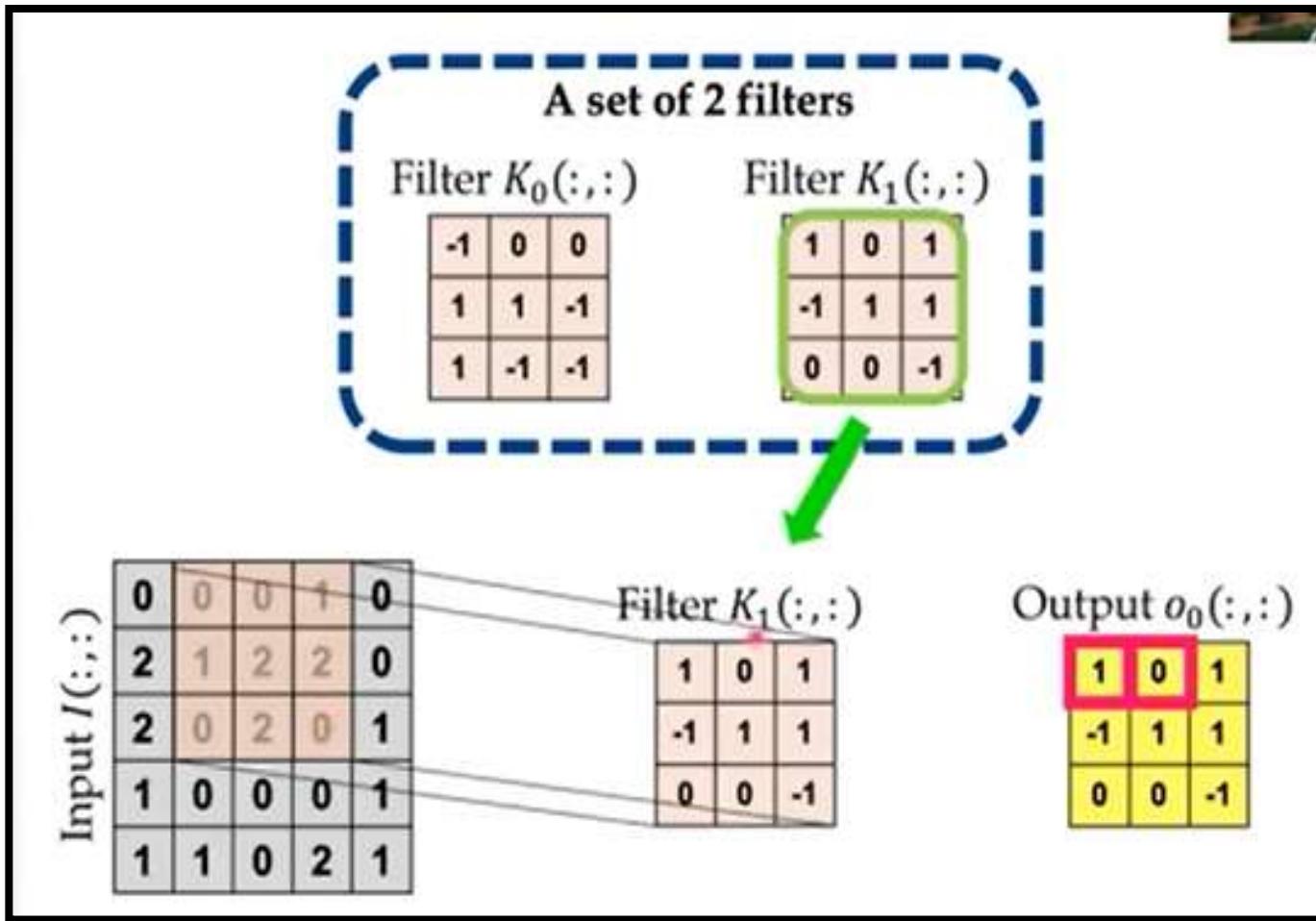
- ❖ 1. Reducing memory consumption



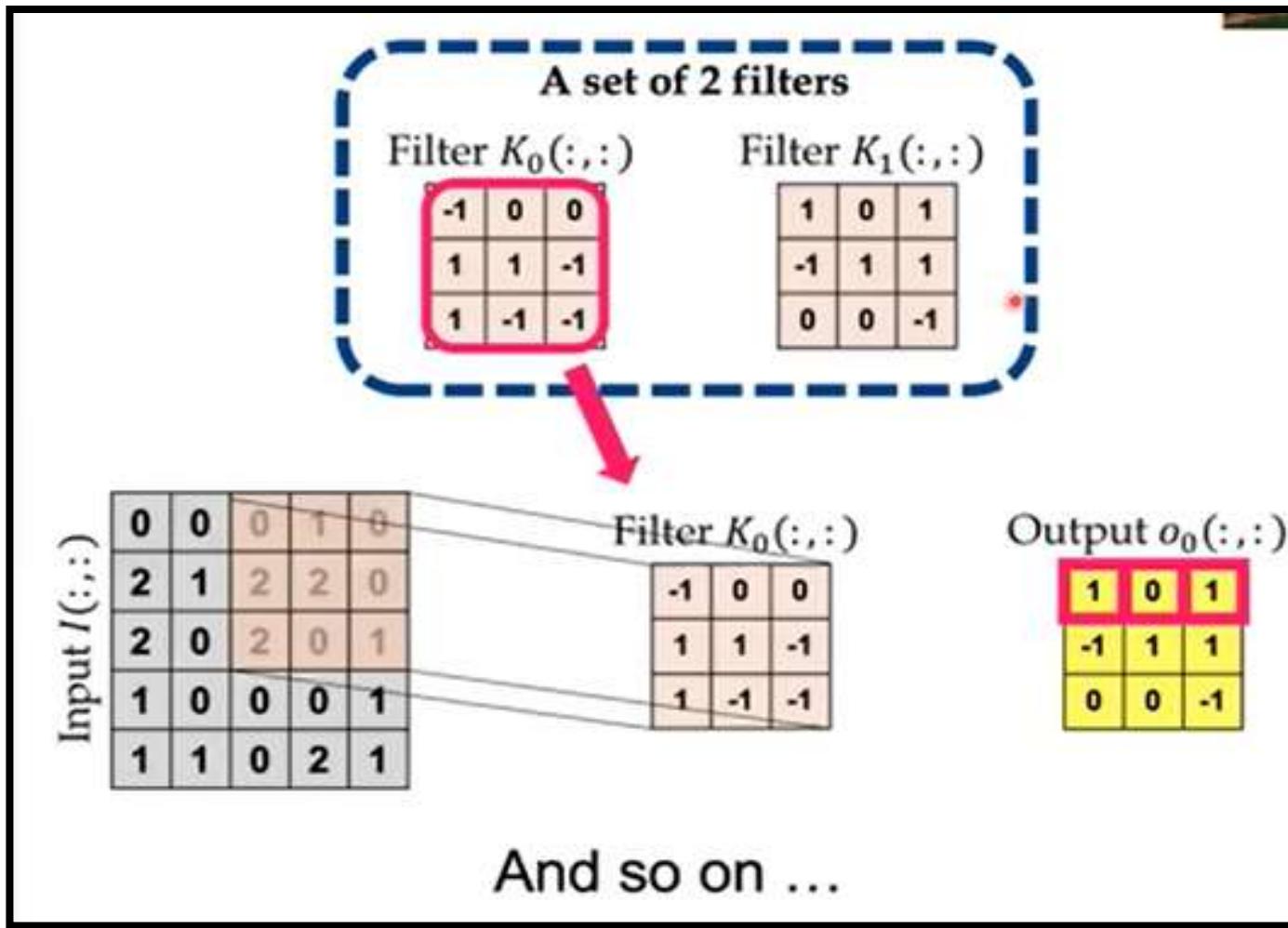
## K0 Filter with 1<sup>st</sup> Portion of the Image :



## K1 Filter with 2<sup>nd</sup> Portion of the Image :



Again K0 Filter with 3<sup>rd</sup> portion of the Image :



### 3. Traditional Convolution:

- ❖ Traditional convolution Equivalent to tiled convolution with  $t=1$ . i.e, there is **only one kernel** and it is **applied everywhere**.
- ❖ It looks like **Same connectivity** as **Unshared Convolution**. But here **we have to use Padding if Needed**.



#### 4. Comparing Computational Times:

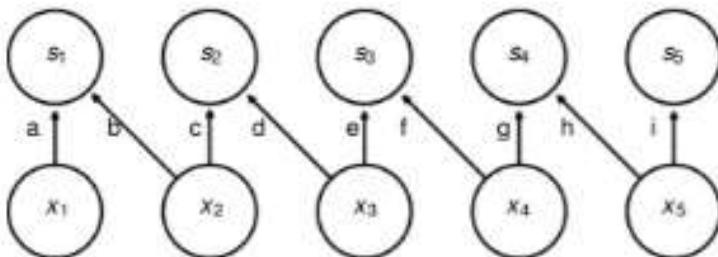
- ❖ The **parameter complexity** and **computation complexity** can be obtained as below.
  - ❖  $m$  = number of input units
  - ❖  $n$  = number of output units
  - ❖  $k$  = kernel size
  - ❖  $I$  = number of kernels in the set (for tiled convolution)

Type	Computations	Parameters
Fully connected	$O(mn)$	$O(mn)$
Locally connected	$O(kn)$	$O(kn)$
Tiled	$O(kn)$	$O(kl)$
Traditional	$O(kn)$	$O(k)$

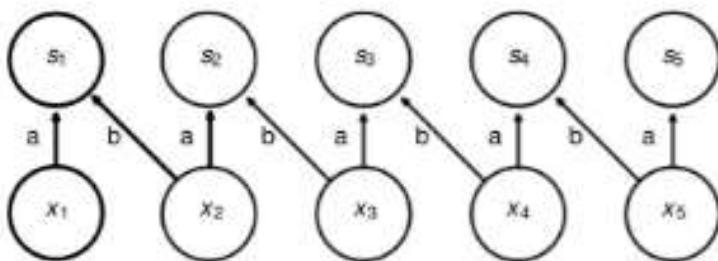


# OVER ALL TYPES OF CONVOLUTIONS

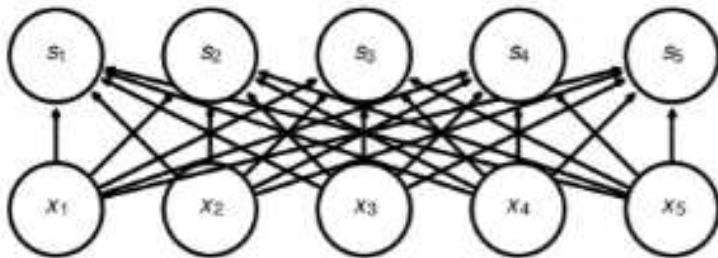
Unshared convolution



Local connection:  
like convolution,  
but no sharing



Convolution



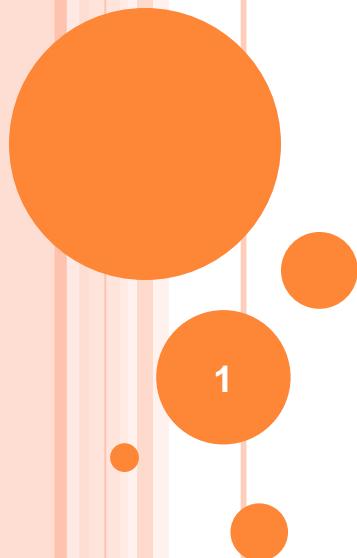
Fully connected



**THANK YOU!**

## **UNIT-III**

# **Convolutional Neural Networks**



# CONTENTS

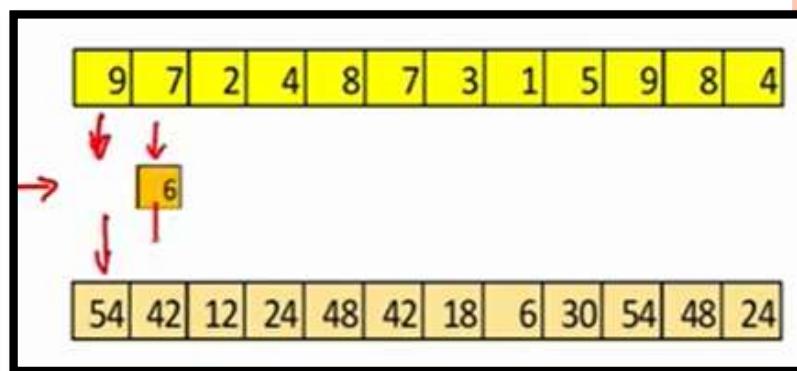
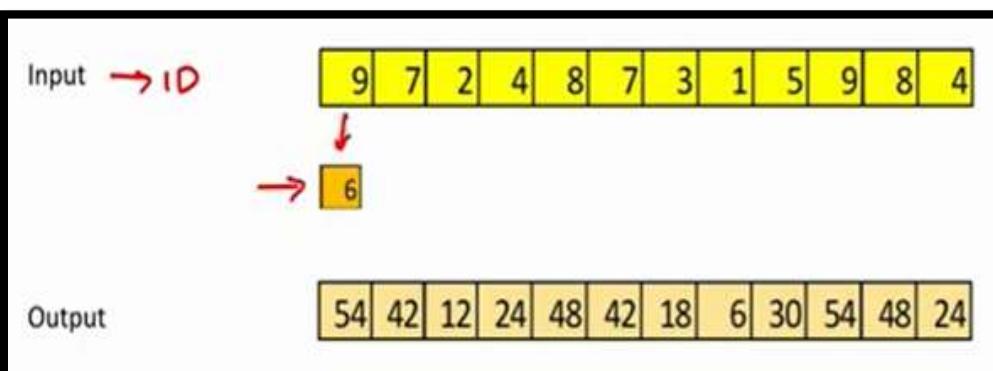
- ❖ Efficient Convolution Algorithms
- ❖ Random or Unsupervised Features

# EFFICIENT CONVOLUTION ALGORITHMS

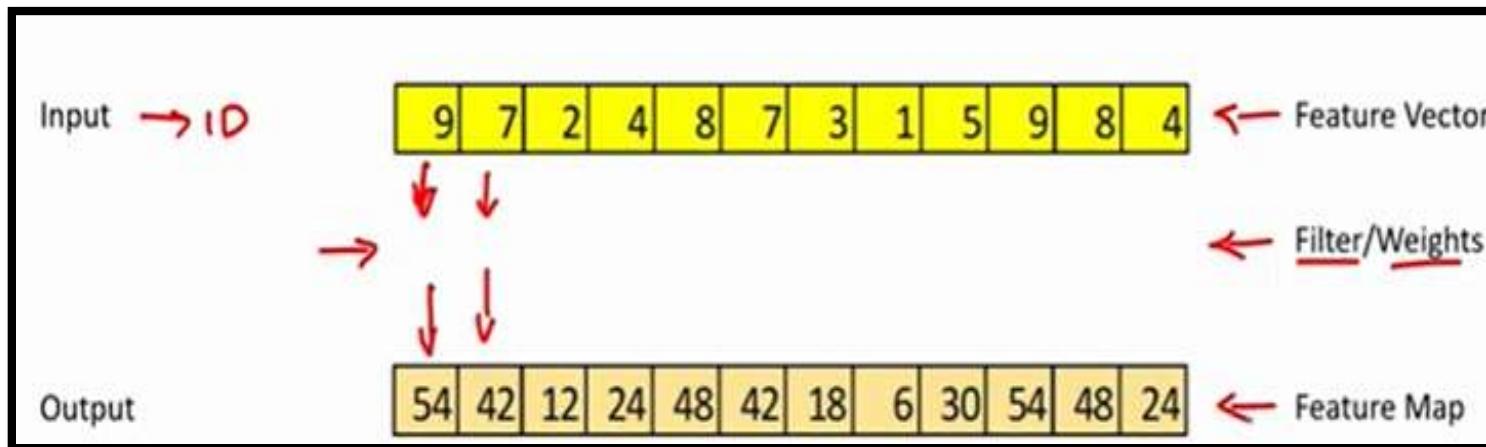
- In the **CNN Architecture**, There are **many Efficient Algorithms are used** based on Research occur in **Object Detection and Image Classification**.
- Here we will show the basic Convolution Algorithms are:
  - 1. One Dimensional (1D) Convolution Algorithm
  - 2. Two Dimensional(2D) Convolution Algorithm
  - 3. Fourier Transform



- 1. One Dimensional Convolution(ID): 1D CNN can perform **activity recognition task** from **accelerometer data**, such as if the person is standing, walking, jumping etc. In **1D convolution example** has **one input channel** and **one output channel**.
- In ID our Input is One Dimensional Array. In the given Example, we have taken the **Input Vector**, and simply multiplied with the **Number**. ( This is the simplest case of 1D Convolution)

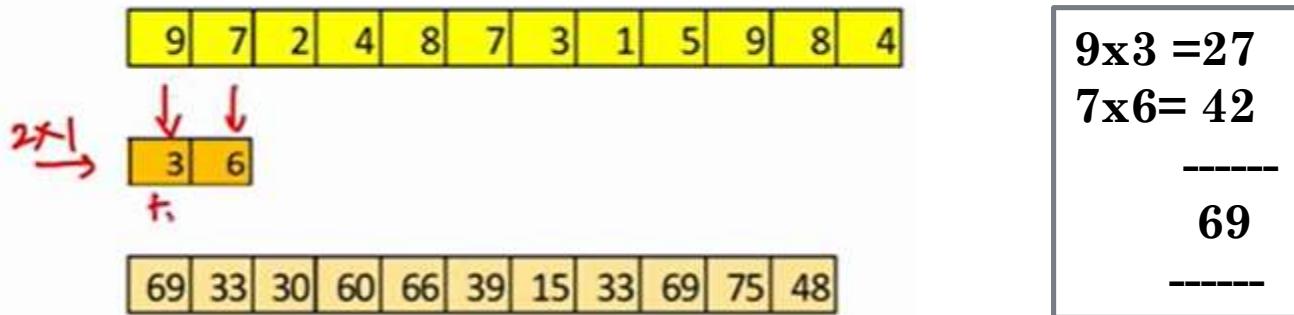


- But we can used this in Deep Learning , we have aware of Feature Vector, Filter or Weights, Feature Map.

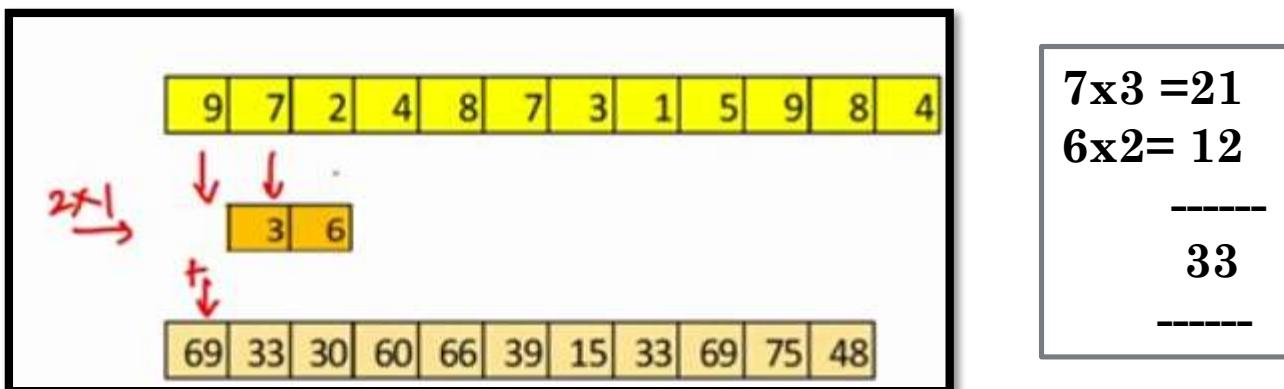


- Generally, Here the Input size is 12 and Filter size is 1 and Output Size is 12.
- But in Deep Learning all can represent by using **Matrix Form**. So, Input Size is  $12 \times 1$  and Filter Size  $1 \times 1$  and Output Size  $12 \times 1$ .

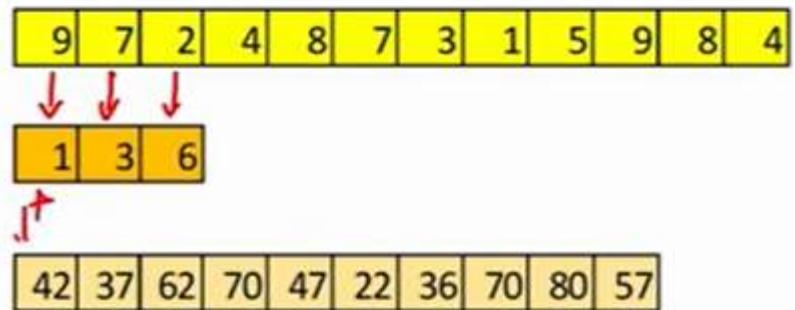
- Eg-3: Suppose here we can take the **Filter Size 3x 1**.



- Next, we can move this filter to next position (i.e, Right Direction)



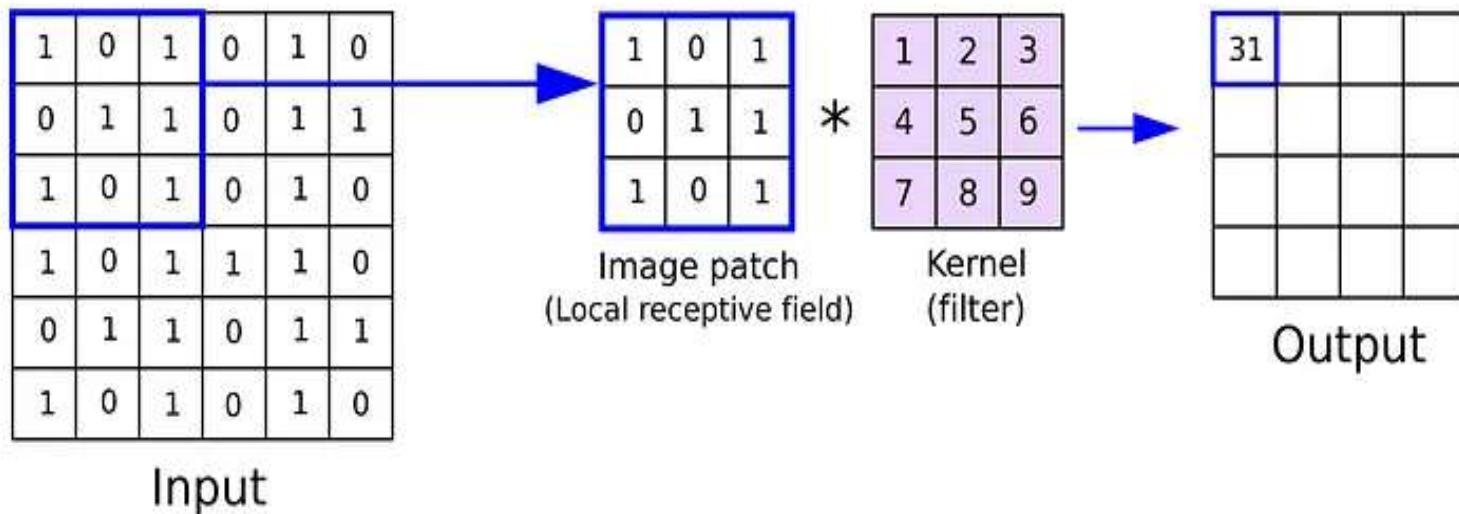
- Eg-2: Suppose here we can take the **Filter Size 3x 1**.



- So here, we can use padding due to loss of information.
- 2. Two Dimensional Convolution(2D): In 2D CNN, kernel moves in 2 directions. Input and output data of 2D CNN is 3 dimensional. Mostly used **on Image data**. Here **filter comes to play** which select some part of images and applies **dot product** on that.

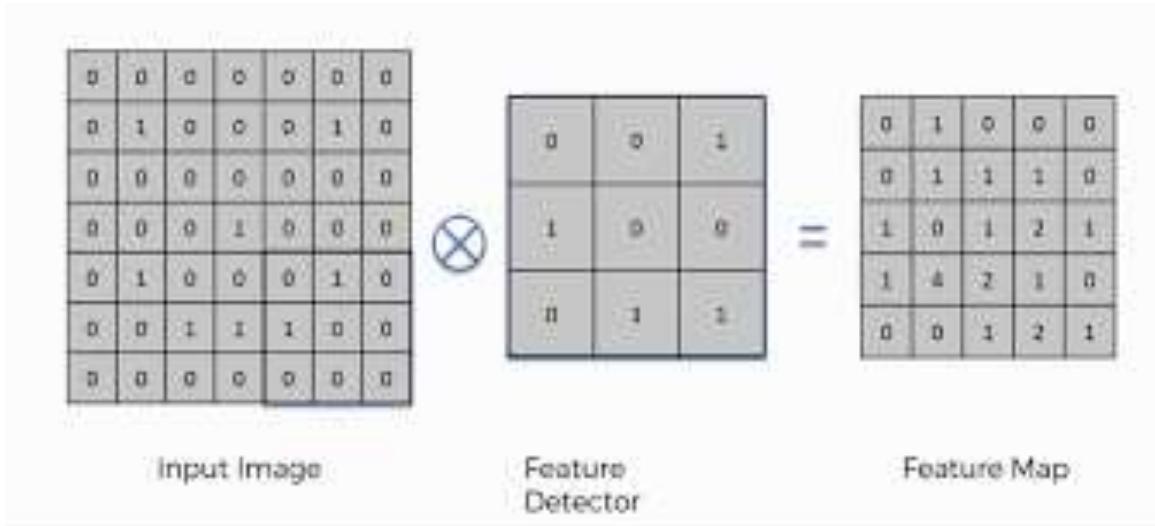


- A dot product is the **element-wise multiplication** between the **filter-sized patch of the input and filter**, which is then **summed**, always resulting in a single value. This filter has to be design to detect specific type of feature in image.



images is just a single value

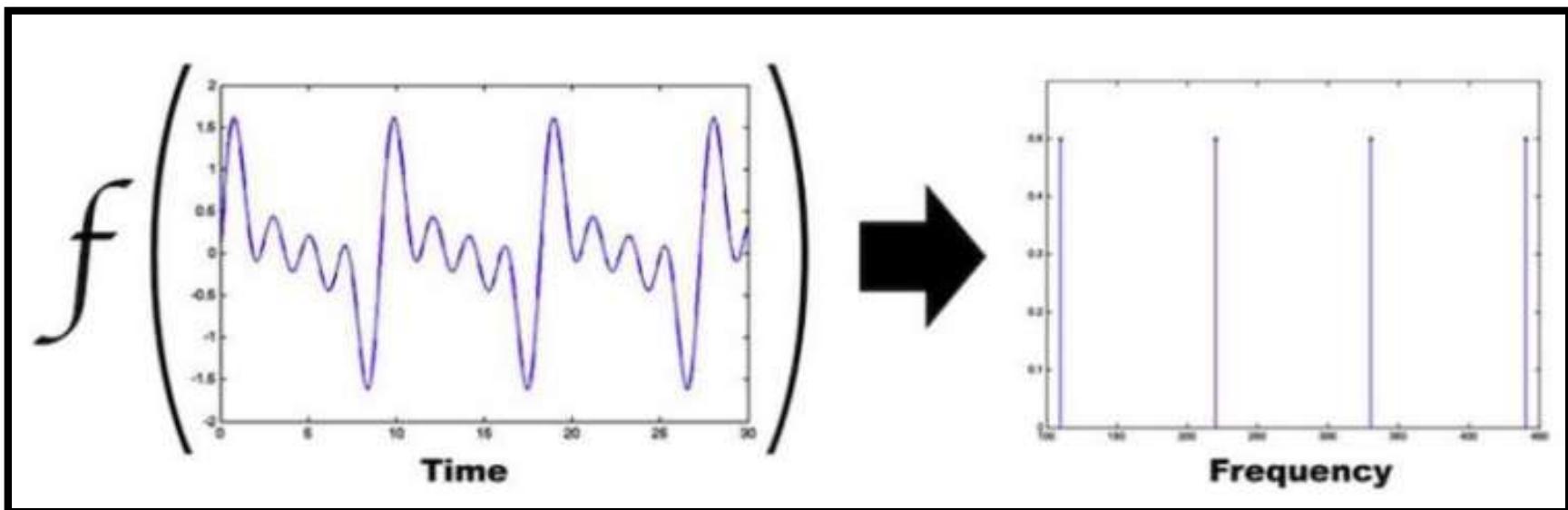
- After filter iterate through entire image(Input), we get feature map.



We can change the kernel or filter to detect horizontal or vertical lines as per needed. This **feature map** help us to **identify edges, vertical lines, horizontal lines, bends, etc**

- **3. Fourier Transform:** The Fourier Transform **is a tool** that **breaks a waveform (a function or signal)** into an alternate representation, characterized by sine and cosines.
- The Fourier transform is a **mathematical formula** that **transforms a signal** sampled in time or space to the same signal sampled in temporal or spatial frequency.
- According to the **convolution property**, the Fourier transform maps convolution to multiplication and also we can use **Seperable Kernels.**





- The **main advantage of Fourier analysis** is that very little information is lost from the signal during the transformation. The Fourier transform **maintains information on amplitude, harmonics, and uses all parts of the waveform** to translate the signal into the frequency domain.

- Seperable Kernels: Convolution is equivalent to converting both input and kernel to frequency domain using a Fourier transform, performing point-wise multiplication of two signals:

$$\mathcal{F}\{f \star g\} = \mathcal{F}\{f\} \times \mathcal{F}\{g\}$$

- Converting back to time domain using an inverse Fourier transform.

$$f \star g = \mathcal{F}^{-1}\{\mathcal{F}\{f \star g\}\} = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \times \mathcal{F}\{g\}\}$$



- When a **d-dimensional kernel** can be expressed as **outer product of d vectors, one vector per dimension**, the **kernel is called separable**.
- The **kernel also takes fewer parameters** to represent as vectors.

Kernel Type	Runtime complexity for <i>d</i> -dimensional kernel with <i>w</i> elements wide
Traditional Kernel	$O(w^d)$
Separable Kernel	$O(w \times d)$

An example application of the Fourier transform is determining the **constituent pitches in a musical waveform**.

# RANDOM OR UNSUPERVISED FEATURES

- Typically, the most expensive part of **conv network training** is **learning the features**.
- There are **3 basic strategies** for obtaining convolution kernels **without supervised training**.
- **1. Simply initialize randomly:** random filters work well in convolutional networks. Inexpensive way to choose the architecture of a convolutional network:
  - Evaluate the performance of several convolutional network architecture by **training only the last layer**.



- Take the **best of these architectures** and **train the entire architecture** using a **more expensive approach**.

## 2. Design them by hand

### 3. Unsupervised training of kernels may be performed;

- Eg: applying k-means clustering to image patches and using the centroids as convolutional kernels.
- Unsupervised pre training may offer regularization effect (not well established). It may also allow for training of larger CNNs because of reduced computation cost.



- Learning the features from unsupervised criterion allows them to be determined **separately from the classifier layer** at the **top of the architecture**.
- Another approach for CNN training is **greedy layer-wise pretraining** most **notably used in convolutional deep belief network**.
- For example, In the case of **multi-layer perceptrons**, starting with the **first layer**, **each layer is trained in isolation**. Once the **first layer is trained**, its **output is stored** and **used as input for training the next layer**, and so on.



- Instead of training an entire convolutional layer at a time, we can train a model of small patch, we can use the parameters from this patch-based to define the kernels of a convolutional layer.
- Today, most convolution networks are trained in a purely supervised fashion, using full forward and back-propagation through the entire network on each training iteration.
- To reduce the computational cost of training the CNN, we can use features not learned by supervised training.
- 1. Random initialization has been shown to create filters that features are selective and translation invariant. This can be used to inexpensively select the model architecture.

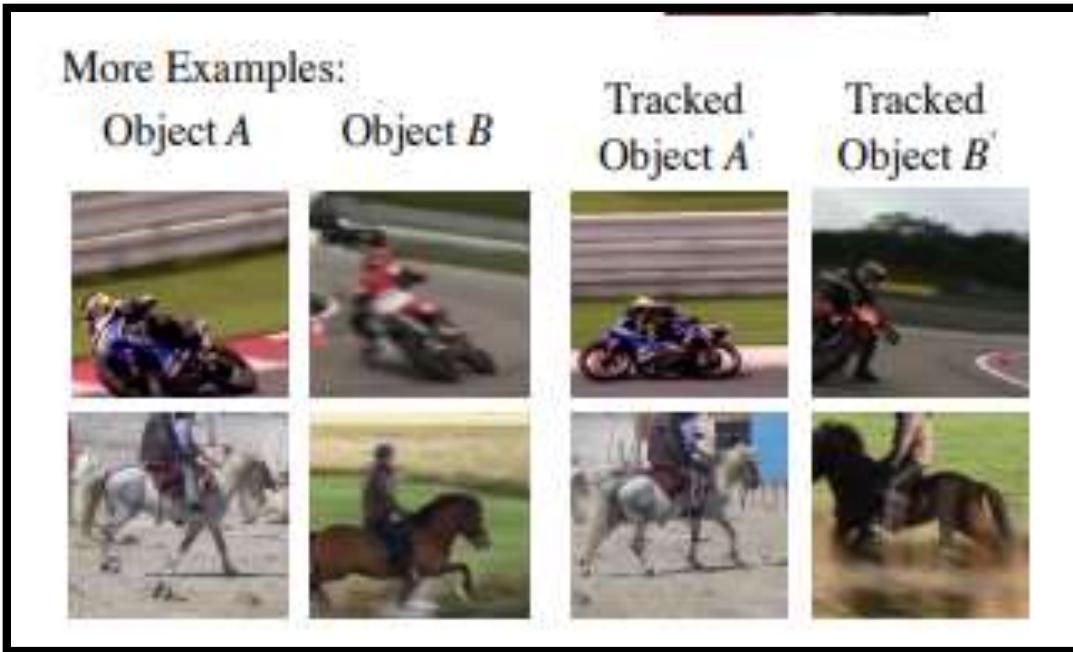
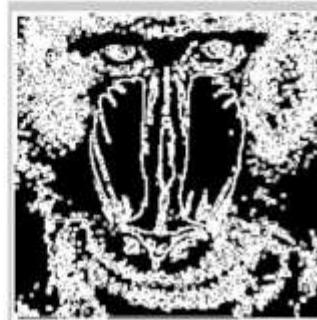
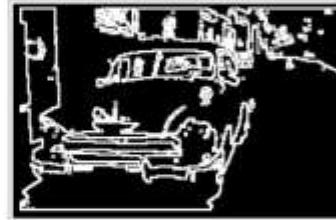
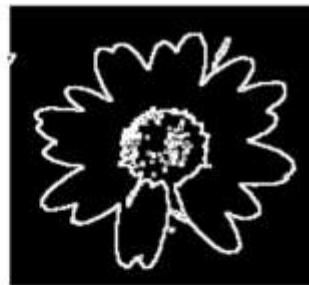
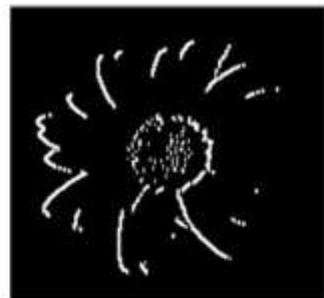


Fig: Transition Invariant

Randomly initialize several CNN architectures and just train the last classification layer. Once features are extracted, that model can be fully trained in a supervised manner.

- 2. Hand designed kernels may be used; e.g. to **detect edges** at different orientations and intensities.



- 3. Unsupervised training of kernels may be performed;
- Eg: applying k-means clustering to image patches and using the centroids as convolutional kernels.
- Unsupervised pre training may offer regularization effect (not well established). It may also allow for training of larger CNNs because of reduced computation cost.

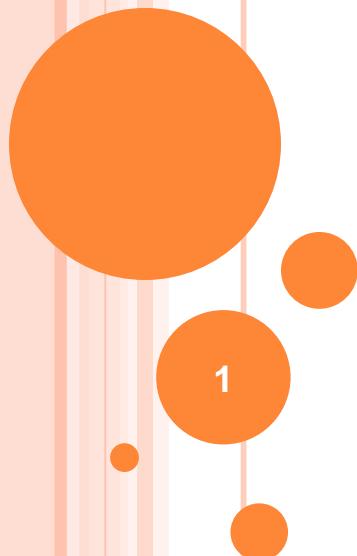




**THANK YOU!**

## **UNIT-III**

# **Convolutional Neural Networks**



# CONTENTS

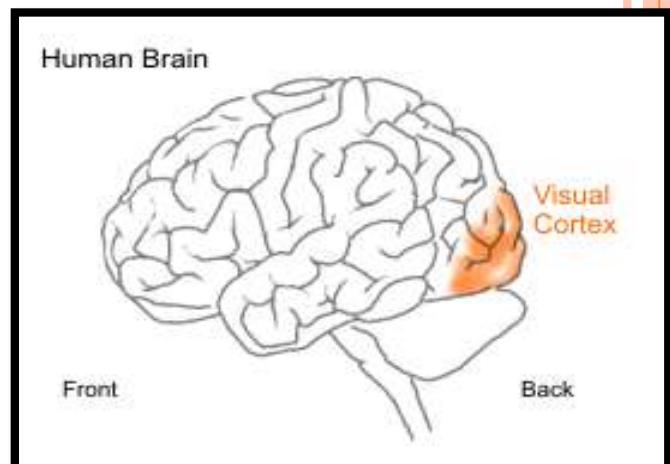
- ❖ The Neuro-scientific Basis for Convolutional Networks
- ❖ Fully Connected Layer

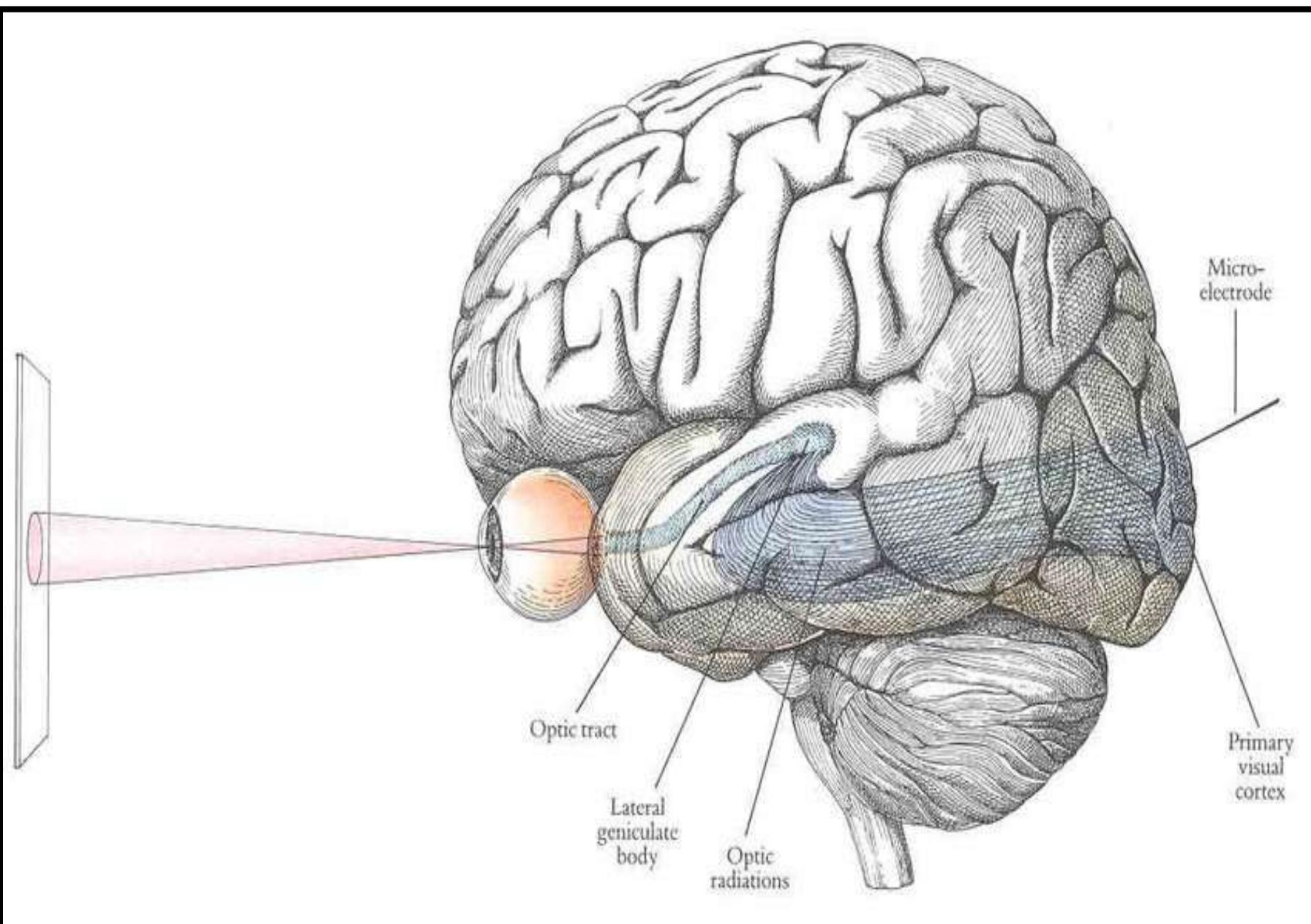
# THE NEURO-SCIENTIFIC BASIS FOR CONVOLUTIONAL NETWORKS

- The Human brain transforms visual inputs into information that is useful for semantic tasks like object recognition and scene interpretation etc.
- Q: How does the human brain transform visual information captured by the retina into information useful for semantic tasks like object recognition and scene interpretation?
- Ans: The history of convolutional networks begins with neuroscientific experiments long before the relevant computational models were developed.



- Recently, **Convolutional Neural Networks (ConvNets)** have been successfully used for transforming image pixels into **features** useful for **object recognition**.
- Neurophysiologists **David Hubel and Torsten Wiesel** observed how **neurons in the cat's brain responded to images** projected in precise locations on a screen in front of the cat. Their work **characterized** many aspects of brain function.
- **Q: How Image is processed by Organism?**

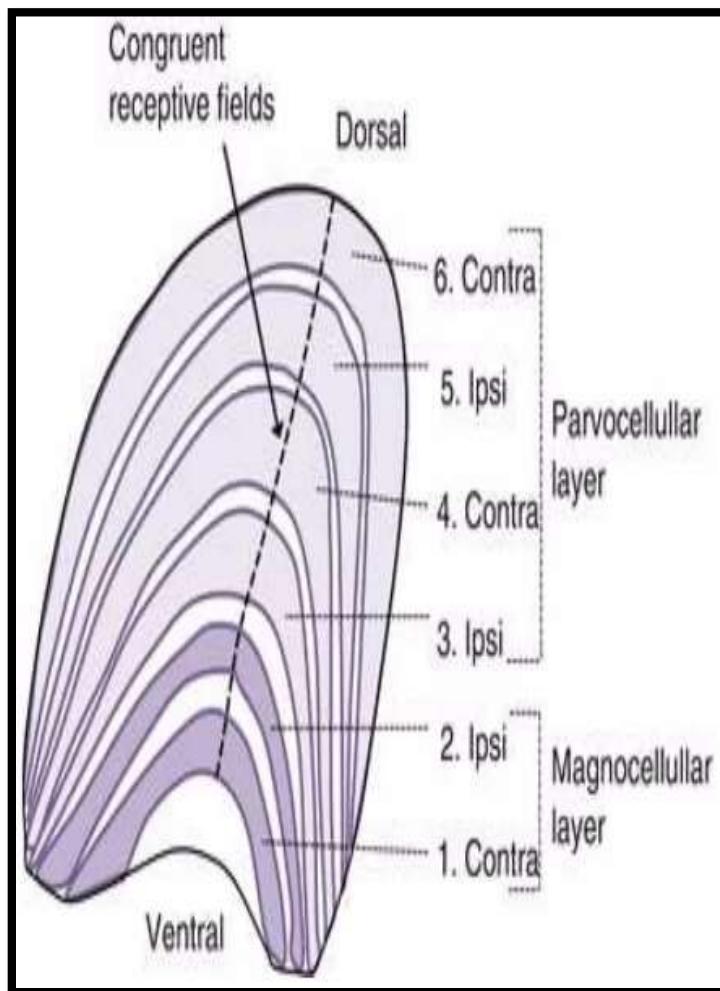




- To teach **an algorithm** how to **recognize objects in images**, data scientists use a specific type of **Artificial Neural Network**: a ***Convolutional Neural Network (CNN)***. Their name stems from one of the most important operations in the network: **convolution**.
- The **Visual Cortex** of the **brain** is a part of the **cerebral cortex** that **processes visual information**. **V1** is the **first area of the brain** that begins to perform significantly **advanced processing of visual input**.
- The **primary purpose** of the **Visual Cortex** called **V1** , is the main site of **input of signals** coming from the **retina** and **is to receive, segment, and integrate visual information**.



- Here the image passes through the optic nerve to a brain region called **Lateral Geniculate Body.(LGB)**.



- In **LGB**, It regulates the flow of **visual information**, ensuring that the most **important information is sent to the cortex**
- Is consists of **mainly 6 Layers in LGB**.
- In 6 Layers consisting of **neurons (Grey Matter)** and remaining are **White Matter**.
- **Layers 1&2** is mainly used for **Motion, Depth, Brightness**.
- **Layers 3 to 6** mainly used for **Color and dark images**.



- 1. The light entering the eye stimulates the retina. The image then passes through the optic nerve and a region of the brain called the **LGN (lateral geniculate nucleus)**
- 2. **V1 (primary visual cortex):** The image produced on the retina is transported to the **V1 with minimal processing.** The properties of V1 that have been replicated in CNNs are:
- Hubel and Weisel described two basic types of **visual neuron cells** in the **brain** that each act in a different way: **simple cells (S cells)** and **complex cells (C cells).**



- The **V1** response is **localized spatially**, i.e. the upper image stimulates the cells in the **upper region of V1 [localized kernel]**.
- The **Simple cells** respond maximally to specific **edge-like patterns** within their **receptive field**. In other words, The **simple cells activate**, when they **identify basic shapes** as lines in a fixed area and a specific angle.
- **V1** has **simple cells** whose activity is a **linear function** of the input in a small neighbourhood **[convolution]**.



- A **convolutional network layer** is designed to **capture** the properties of V1:
- **V1** is arranged in a **spatial map**. Here Space represents the **2D plane(x, y) in images**.
- It actually has a **two-dimensional structure** mirroring the structure of the image in the **retina**.
- **Convolutional networks** capture this property by having **their features** defined in terms of **two dimensional maps**.



- The **Complex cells** have **larger receptive fields** and **their output is not sensitive** to the **specific position in the field**. The complex cells continue to **respond to a certain stimulus**, even though its absolute position on the retina changes.
- **V1** has **complex cells** whose activity is **invariant to shifts** in the **position of the feature [pooling]** as well as **some changes** in **lighting**.
- **3.** There are several stages of **V1 like operations** [**stacking convolutional layers**].



- The **Neurons in the early visual cortex** are organized in a **hierarchical fashion**, where the first cells connected to the **cat's retinas** are **responsible for detecting simple patterns like edges and bars**, followed by later layers responding **to more complex patterns** by **combining the earlier neuronal activities**.
- Finally, **Convolutional Neural Network** may learn to **detect edges from raw pixels** in the **first layer**, then use the **edges to detect simple shapes** in the **second layer**, and then use these shapes to **higher-level features**, such as **facial shapes** in higher layers.



- There are many differences between convolutional networks and the mammalian vision system. Some of these differences are -
  - 1. The human eye is mostly very low resolution, except for a tiny patch called the fovea. Most convolutional networks receive large full resolution photographs as input.
  - 2. The human visual system is integrated with many other senses, such as hearing, and factors like our moods and thoughts. Convolutional networks so far are purely visual.



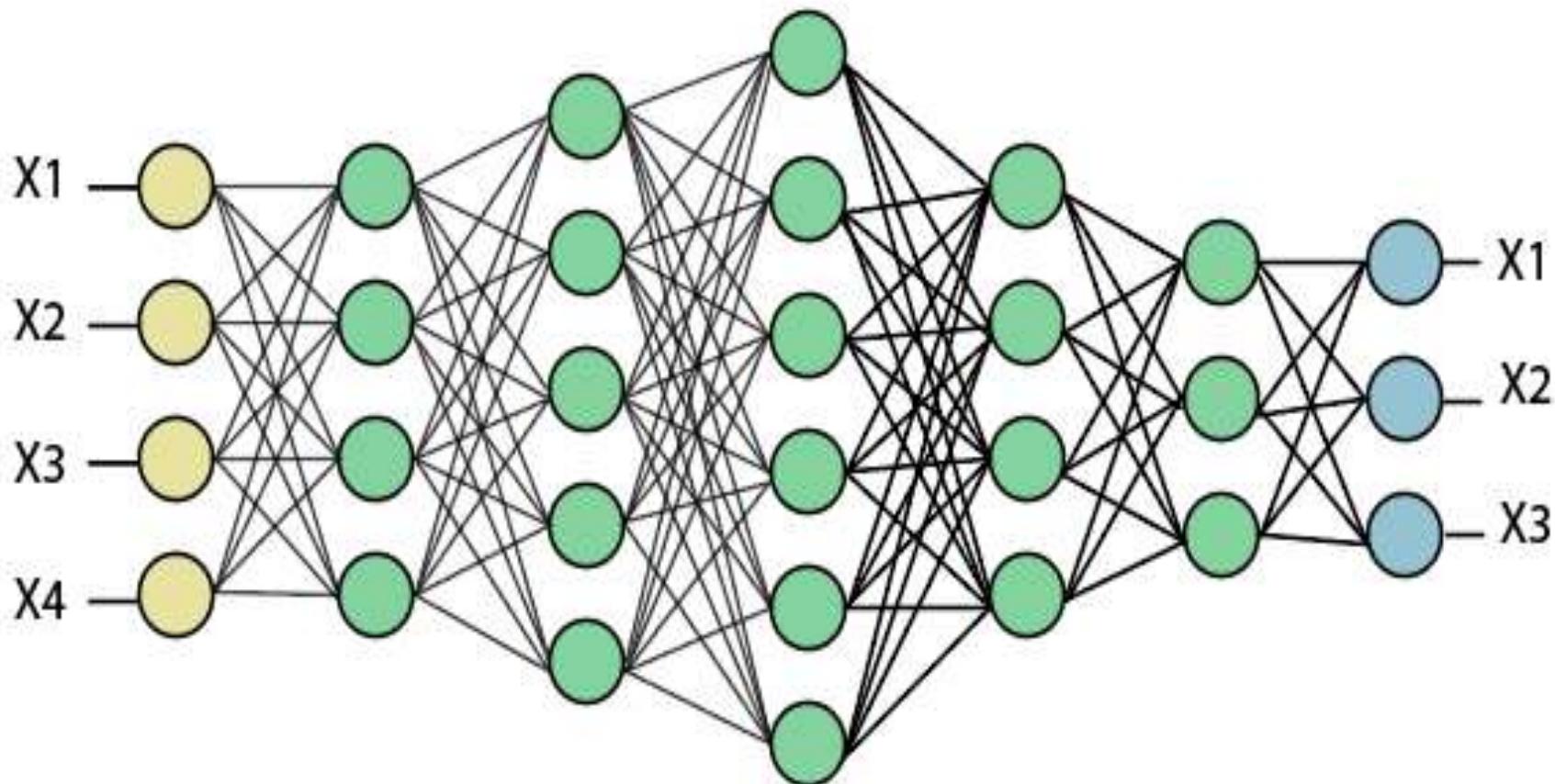
# LAST LAYER - FULLY CONNECTED LAYER

- **Fully Connected Layers:** The **last** layer in CNN is the fully connected layer.
- Fully connected means that **every output that's produced at the end of the last pooling layer** as an input to each node in this **fully connected layer**.
- Neurons in a fully connected layer have full connections to all the activations in the previous layer.
- This part is in principle the same as a regular Neural Network. Their activations can hence be computed with a matrix multiplication followed by a bias offset.

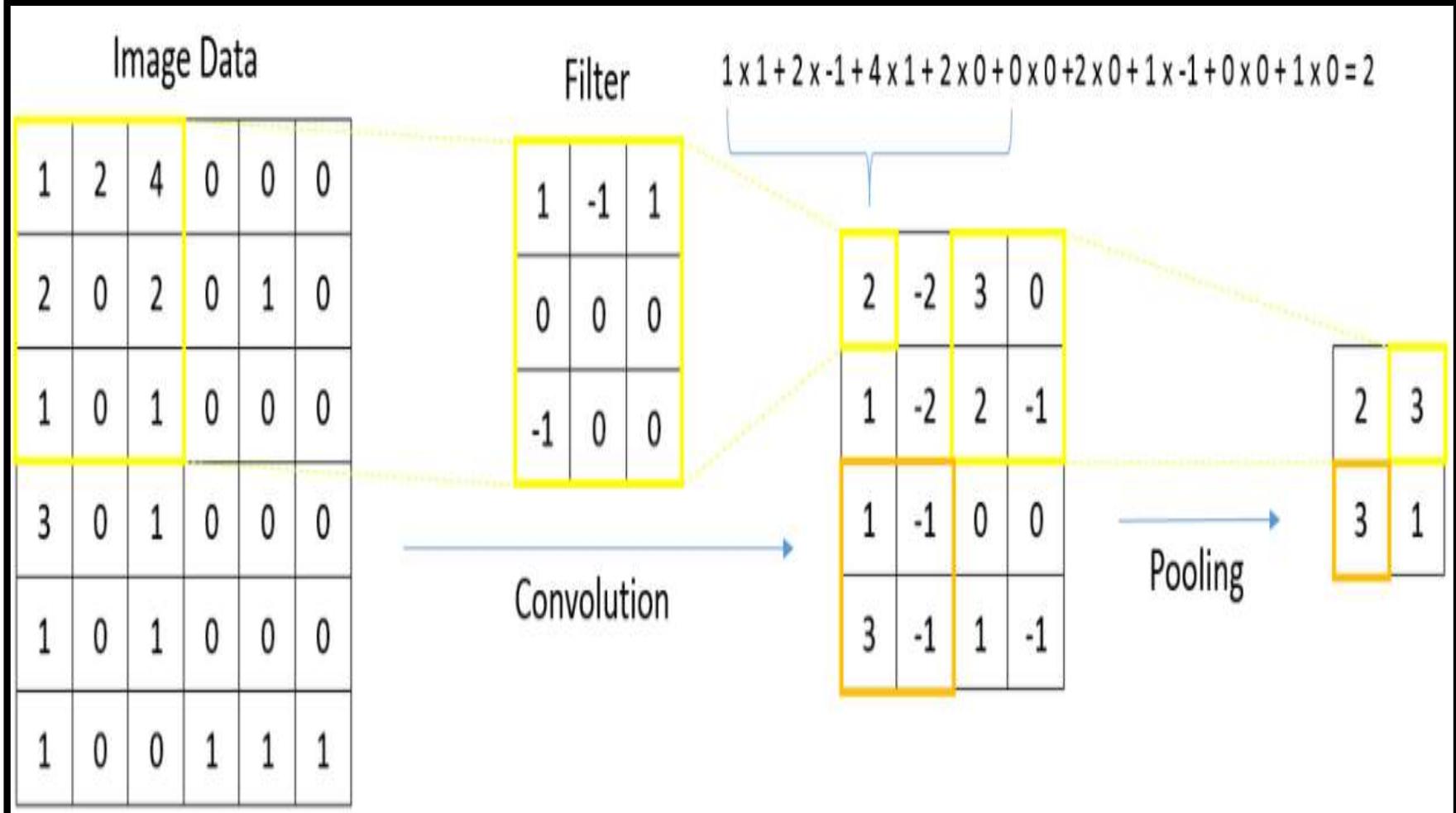
- However, these fully connected layers can only accept 1 Dimensional data; hence we have to first flatten a 3D into a 1D vector.
- The **role of the fully connected layer** is to **produce a list of class scores** and **perform classification** based on image features that have been **extracted** by the previous **convolutional and pooling layers**. So, the last **fully connected layer** will have as **many nodes as there are classes**.

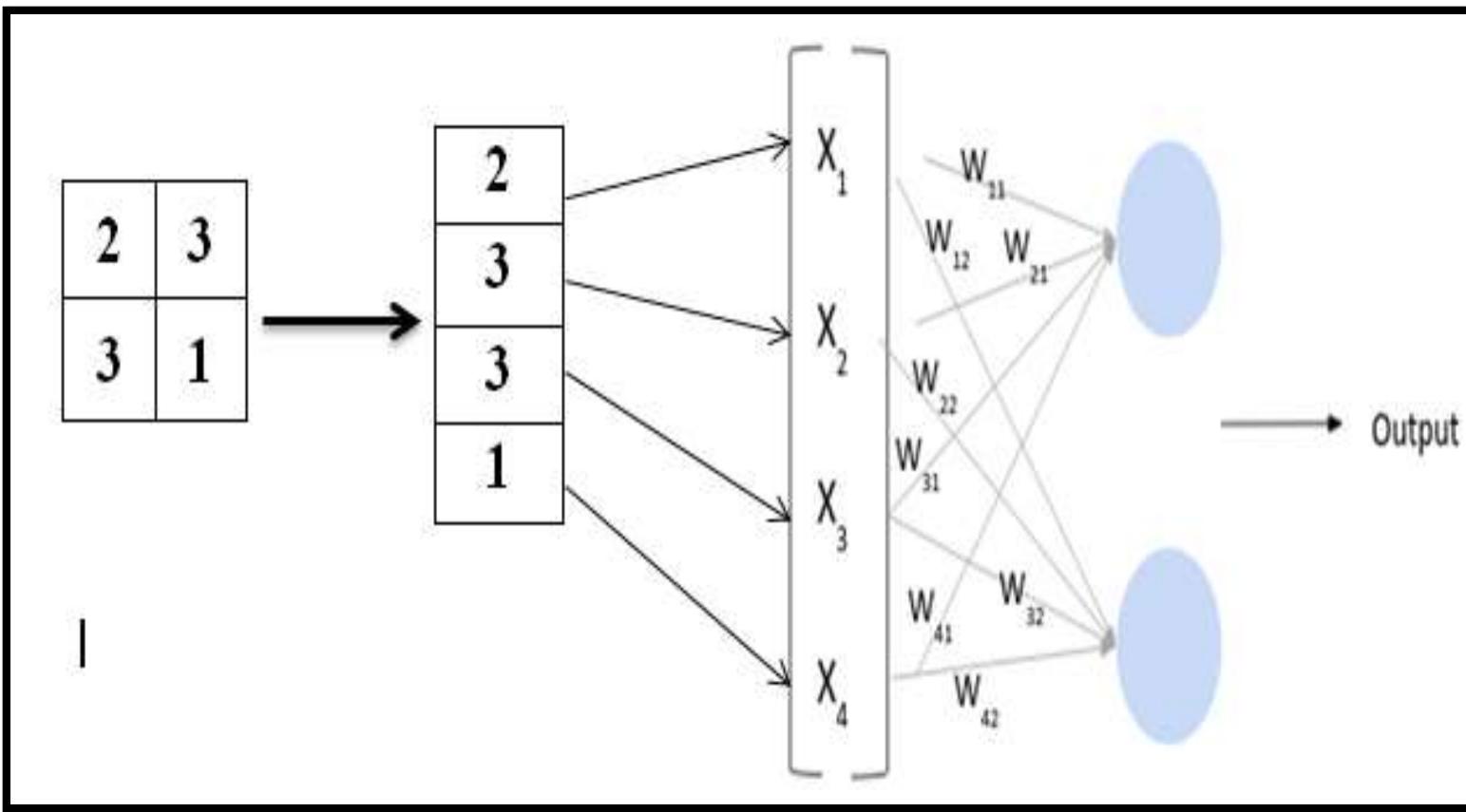


## Fully Connected Layer



# COMPLETE CONVOLUTIONAL NEURAL NETWORK EXAMPLE





- However, these fully connected layers can only accept 1 Dimensional data; hence we have to first flatten a 3D into a 1D vector.
- The **role of the fully connected layer** is to **produce a list of class scores** and **perform classification** based on image features that have been **extracted** by the previous **convolutional and pooling layers**. So, the last **fully connected layer** will have as **many nodes as there are classes**.





**THANK YOU!**