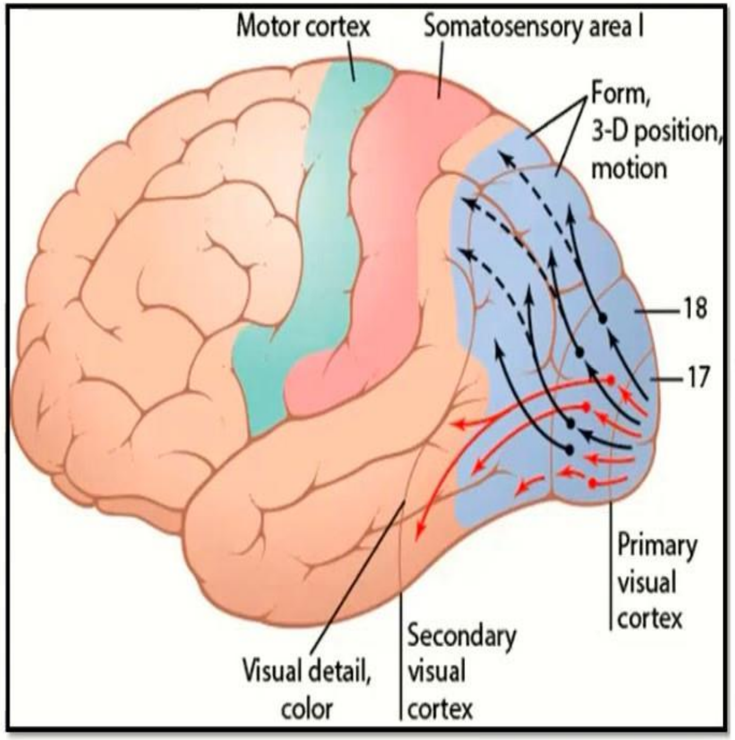
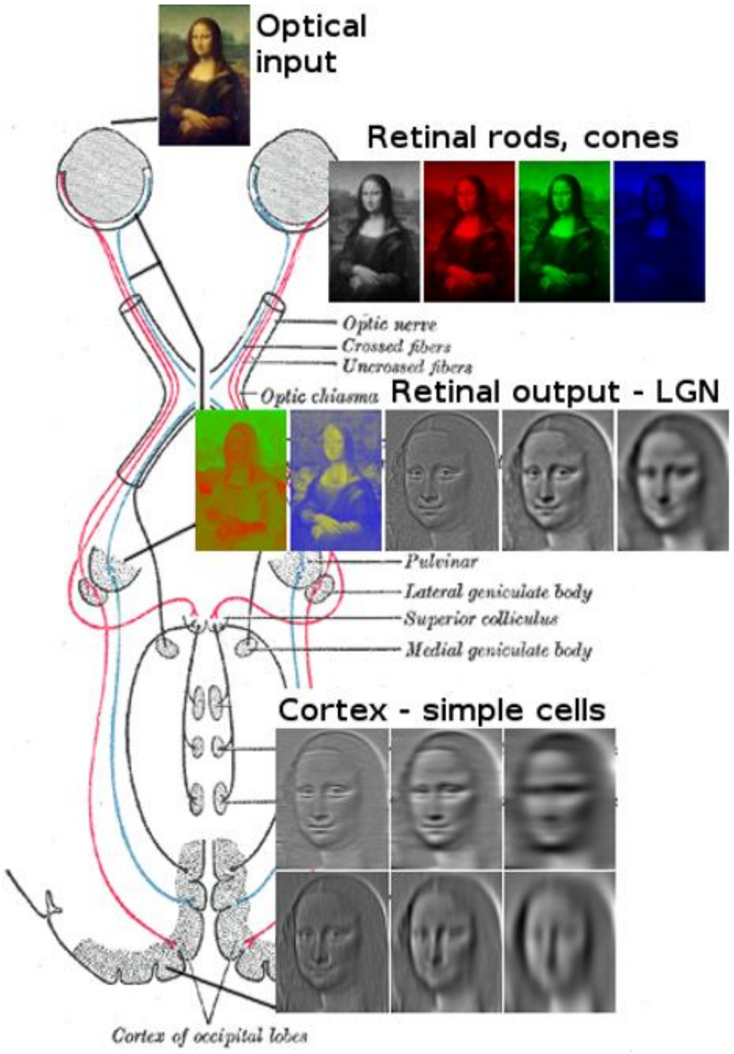
From Pixels to Patterns The Magic of Convolutional Neural Networks

Dr. Muhammad Sajjad

R.A: Kaleem Ullah





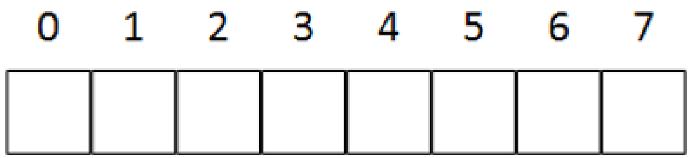
Our Brains are amazing at Vision

- The **visual cortex** (located in the occipital lobe) is the primary cortical region of the brain that receives, integrates, and processes visual information relayed from the retinas

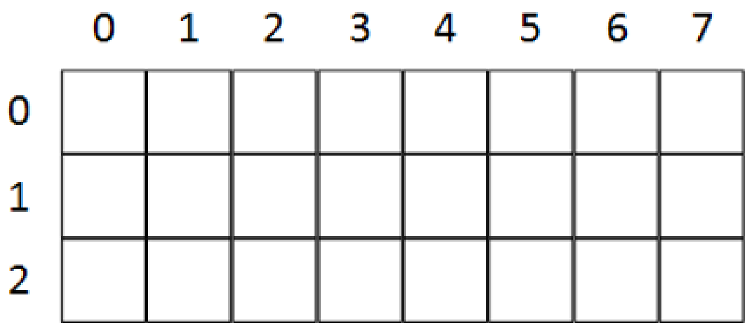
Digital Images Format

Images are stored in Multi-Dimensional Arrays

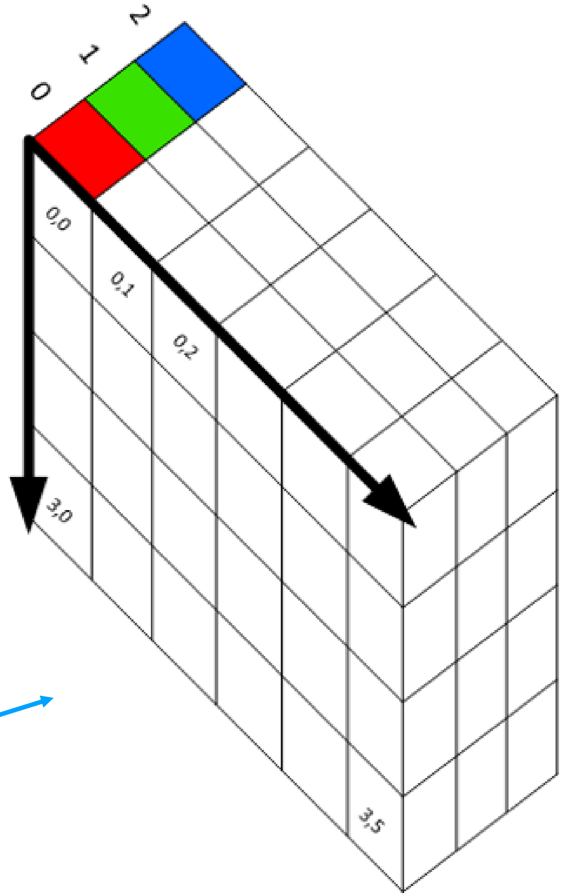
- A 1-Dimensional array looks like this



- A 2-Dimensional array looks like this



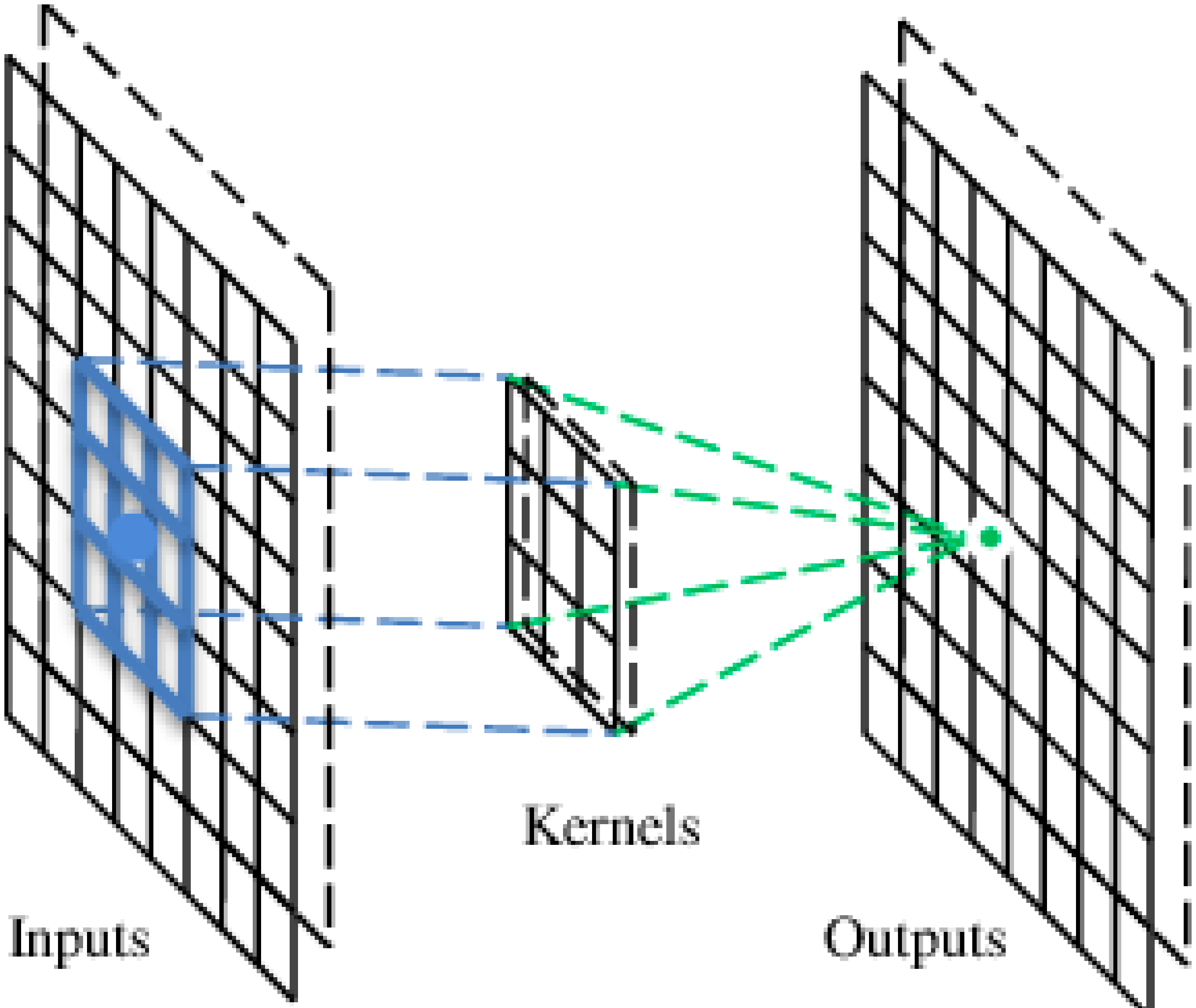
- A **3-Dimensional** array looks like this



Convolutional Neural Networks

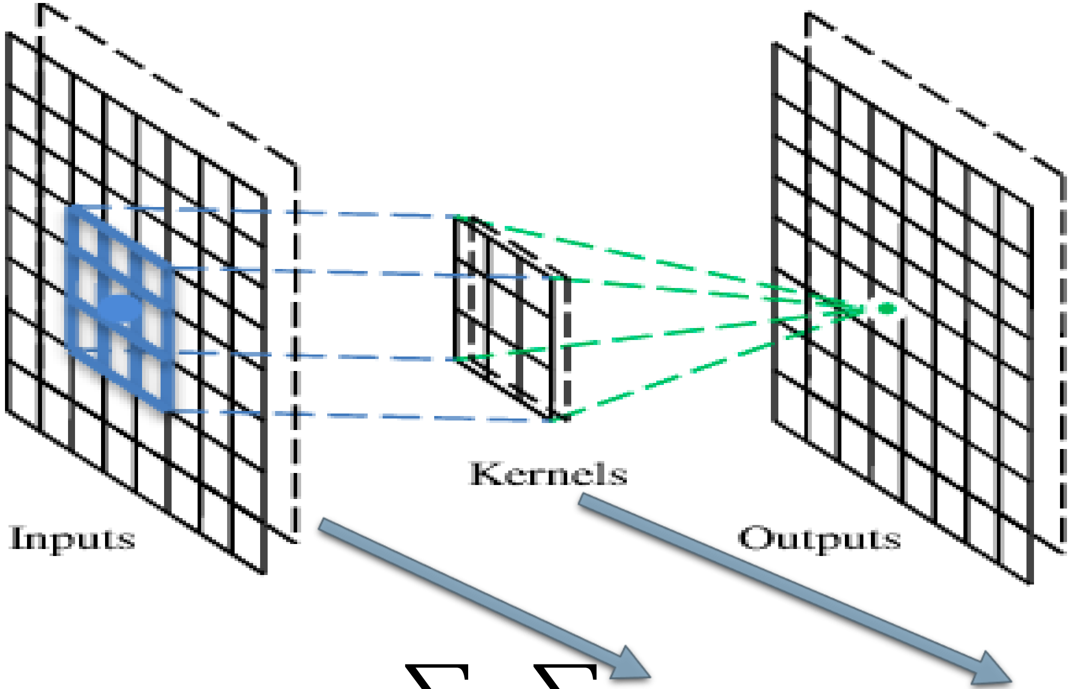
A convolutional layer

- A CNN is a neural network with some convolutional layers
- (and some other layers). A convolutional layer has a number
- of filters that does convolutional operation.



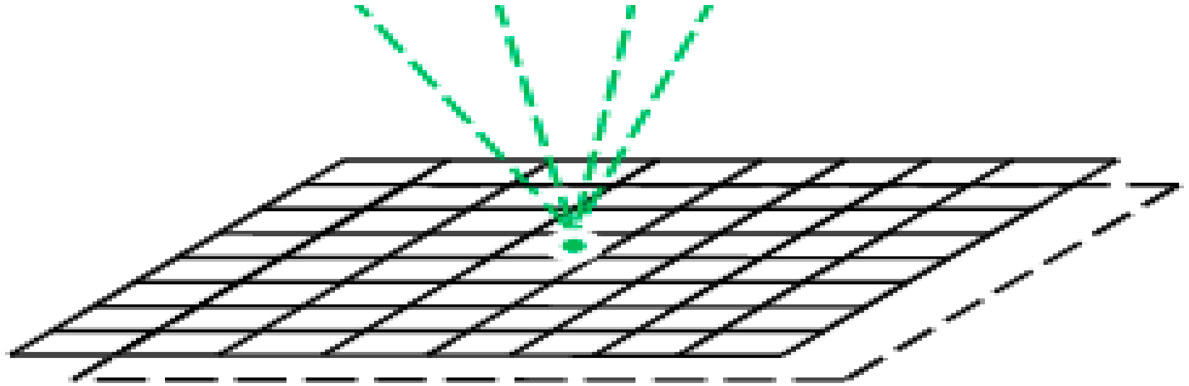
A convolutional layer

- In CNNs, the convolution operation is the core building block
- computing dot products between the filter and local regions of image
- filter’s weights extracts features such as edges, textures or patterns.



I is input Image
K kernel
m,n are filter indices

$$(I * K)(i,j) = \sum_m \sum_n I(i-m, j-n).K(m,n)$$



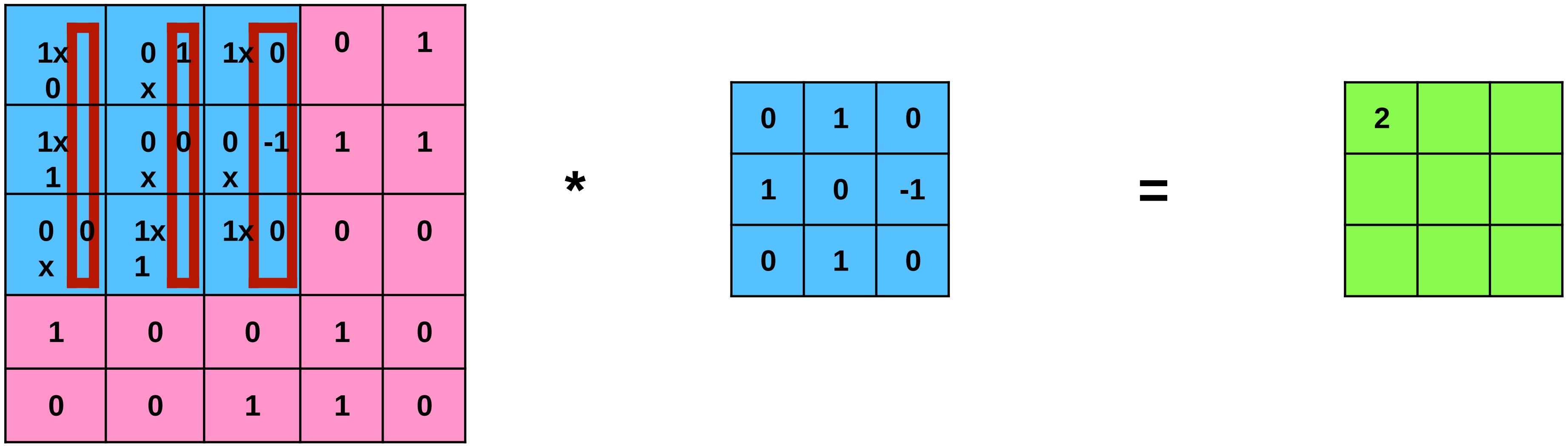
Example of a Convolution Operation

$$
\begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}
*
\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}
=
\begin{bmatrix} 2 & 1 & -1 \\ -1 & 1 & 3 \\ 2 & 1 & 1 \end{bmatrix}
$$

Input Image — Filter or Kernel — Output or Feature Map

Example of a Convolution Operation

$$(1 \times 0) + (0 \times 1) + (1 \times 0) + (1 \times 1) + (0 \times 0) + (0 \times -1) + (0 \times 0) + (1 \times 1) + (1 \times 0) = 2$$



Input Image

Filter or Kernel

Output or Feature Map

Example of a Convolution Operation

$$(0 \times 0) + (1 \times 1) + (0 \times 0) + (0 \times 1) + (0 \times 0) + (1 \times -1) + (1 \times 0) + (1 \times 1) + (0 \times 0) = 1$$

1	0x0	1x1	0x0	1
1	0x1	0x0	1x-1	1
0	1x0	1x1	0x0	0
1	0	0	1	0
0	0	1	1	0

Input Image

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	1	

Output or Feature Map

Example of a Convolution Operation

$$
\begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}
*
\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}
=
\begin{bmatrix} 2 & 1 & -1 \\ \; & \; & \; \\ \; & \; & \; \end{bmatrix}
$$

Input Image | Filter or Kernel | Output or Feature Map

Example of a Convolution Operation

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

Input Image

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	1	-1
-1		

Output or Feature Map

Example of a Convolution Operation

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

Input Image

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	1	-1
-1	1	

Output or Feature Map

Example of a Convolution Operation

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

Input Image

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	1	-1
-1	1	3

Output or Feature Map

Example of a Convolution Operation

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

Input Image

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	1	-1
-1	1	3
2		

Output or Feature Map

Example of a Convolution Operation

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 1 & -1 \\ -1 & 1 & 3 \\ 2 & 1 & \end{bmatrix}$$

Input Image — Filter or Kernel — Output or Feature Map

Example of a Convolution Operation

$$
\begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}
*
\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}
=
\begin{bmatrix} 2 & 1 & -1 \\ -1 & 1 & 3 \\ 2 & 1 & 1 \end{bmatrix}
$$

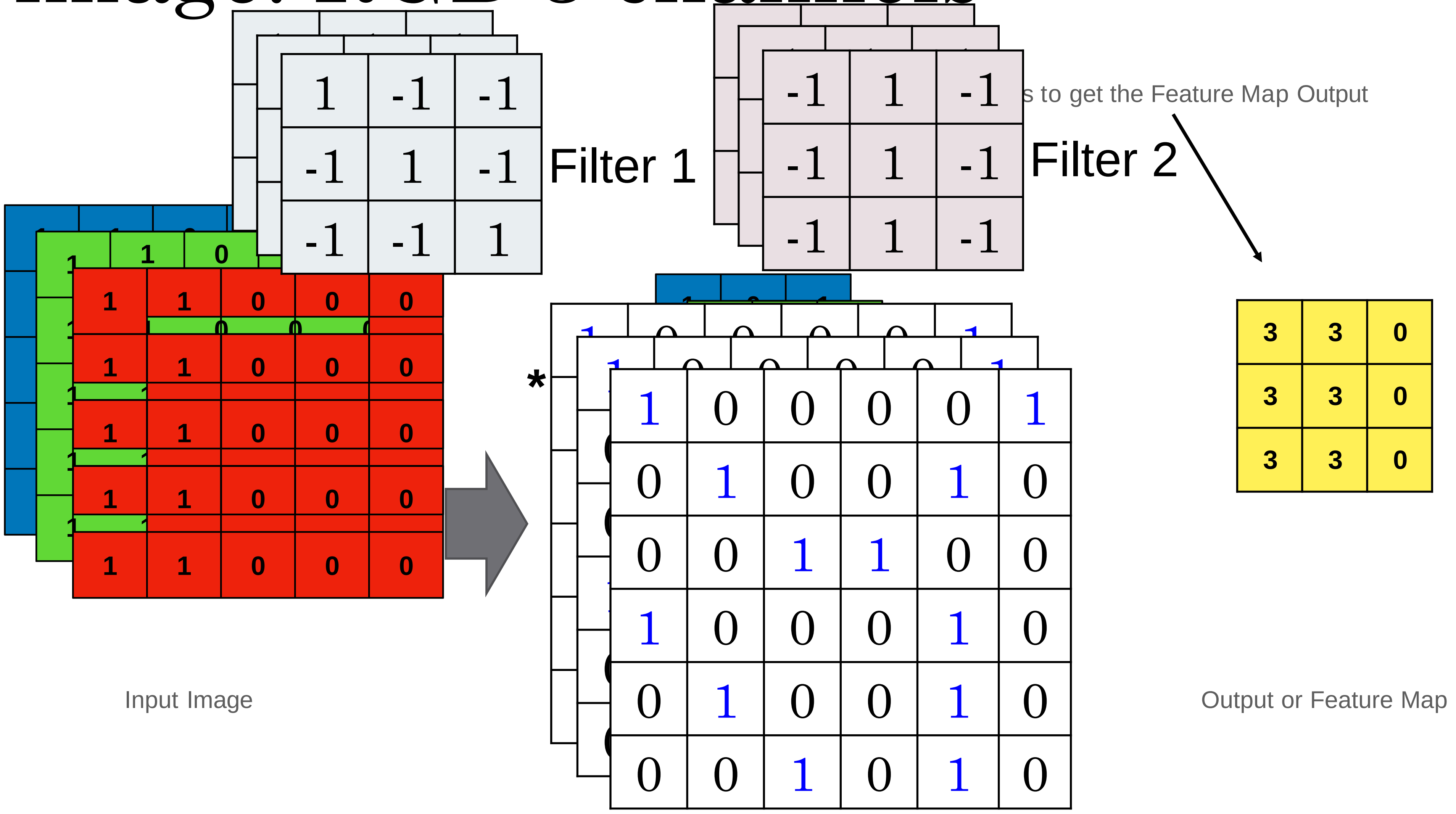
Input Image

Filter or Kernel

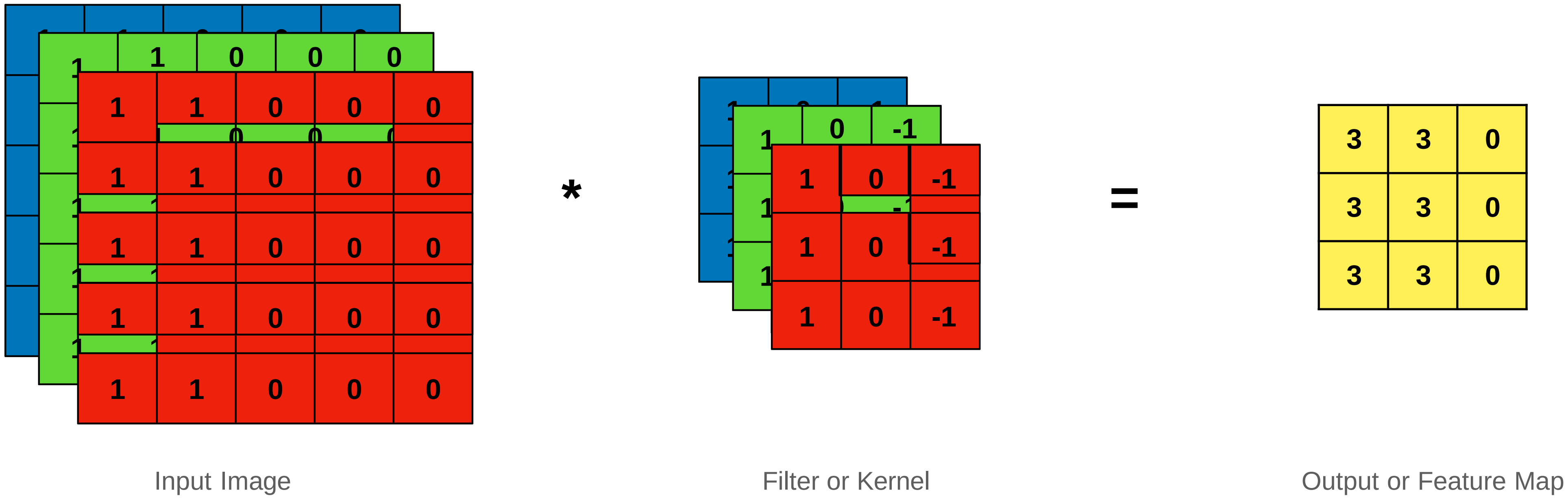
Output or Feature Map

Convolution Operations on Color Images

Color image: RGB 3 channels

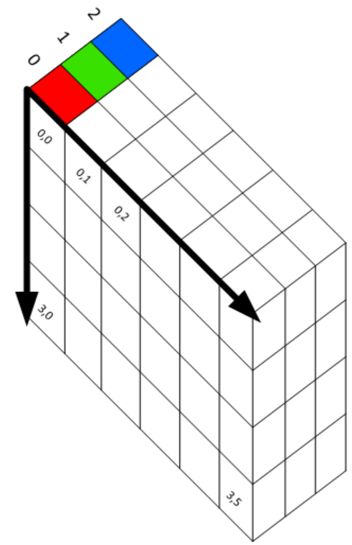


Advantages of Having a Filter For Each Colour



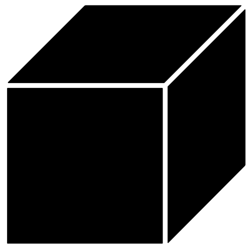
- We can detect features that are specific to a colour

Considered 3D Volumes



Input Image
5 x 5 x **3**

*



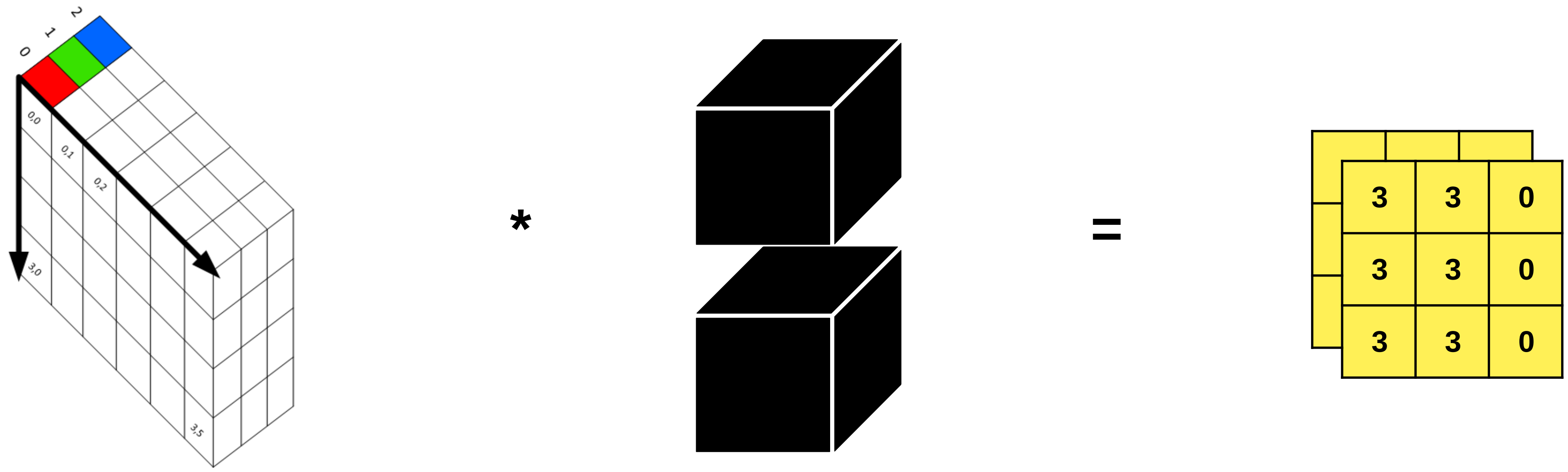
Filter or Kernel
3 x 3 x **3**

=

3	3	0
3	3	0
3	3	0

Output or Feature Map
3 x 3

How Multiple Filters Affect Our Output



Input Image
5 x 5 x 3

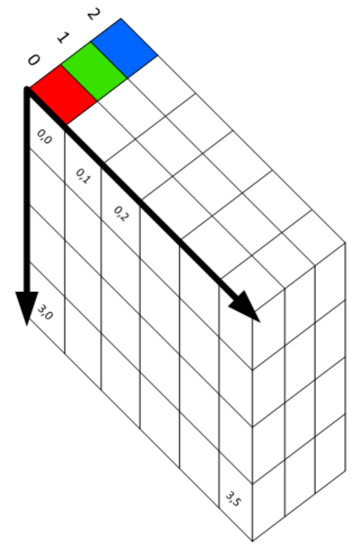
2 Filters or Kernels
3 x 3 x 3 x 2

Output or Feature Map
3 x 3 x 2

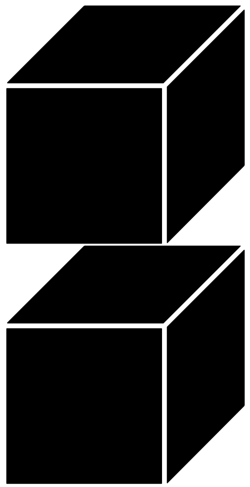
Calculating Output Size for 3D Conv Volumes

$$(n \times n \times n_c) * (f \times f \times n_c) = (n - f + 1) \times (n - f + 1) \times n_f$$

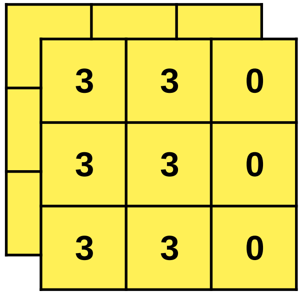
$$(5 \times 5 \times 3) * (3 \times 3 \times 3) = 3 \times 3 \times 2$$



*



=



Input Image
5 x 5 x **3**

2 Filters or Kernels
3 x 3 x **3** x **2**

Output or Feature Map
3 x 3 x **2**

Consecutive Conv layers would keep shrinking the output

Can we preserve our image size?

We’ve added a 1 pixel pad of zeros (zero padding) around our input

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
*
\begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & -1 \\
0 & 1 & 0
\end{bmatrix}
=
$$

7 x 7

$n \times n$

3 x 3

$f \times f$

Padding

Let's Perform our Convolution with the Padding

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
*
\begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & -1 \\
0 & 1 & 0
\end{bmatrix}
=
$$

7 x 7   3 x 3

$n \times n$ $f \times f$

Padding

Let's Perform our Convolution with the Padding

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} =$$

7 x 7   3 x 3

$n \times n$ $f \times f$

Padding

Let’s Perform our Convolution with the Padding

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
*
\begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & -1 \\
0 & 1 & 0
\end{bmatrix}
=
$$

7 x 7   3 x 3

$n \times n$ $f \times f$

Padding

Let's Perform our Convolution with the Padding

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

*

0	1	0
1	0	-1
0	1	0

=

7 x 7

$n \times n$

3 x 3

$f \times f$

Padding

Let's Perform our Convolution with the Padding

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

*

0	1	0
1	0	-1
0	1	0

=

7 x 7

$n \times n$

3 x 3

$f \times f$

Padding

Let’s Perform our Convolution with the Padding

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
*
\begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & -1 \\
0 & 1 & 0
\end{bmatrix}
=
$$

7 x 7   3 x 3

$n \times n$ $f \times f$

Padding

Let's Perform our Convolution with the Padding

Feature Map Size = $n - f + 1 = m$

Feature Map Size = $7 - 3 + 1 = 5$

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

7 x 7

$n \times n$

*

0	1	0
1	0	-1
0	1	0

3 x 3

$f \times f$

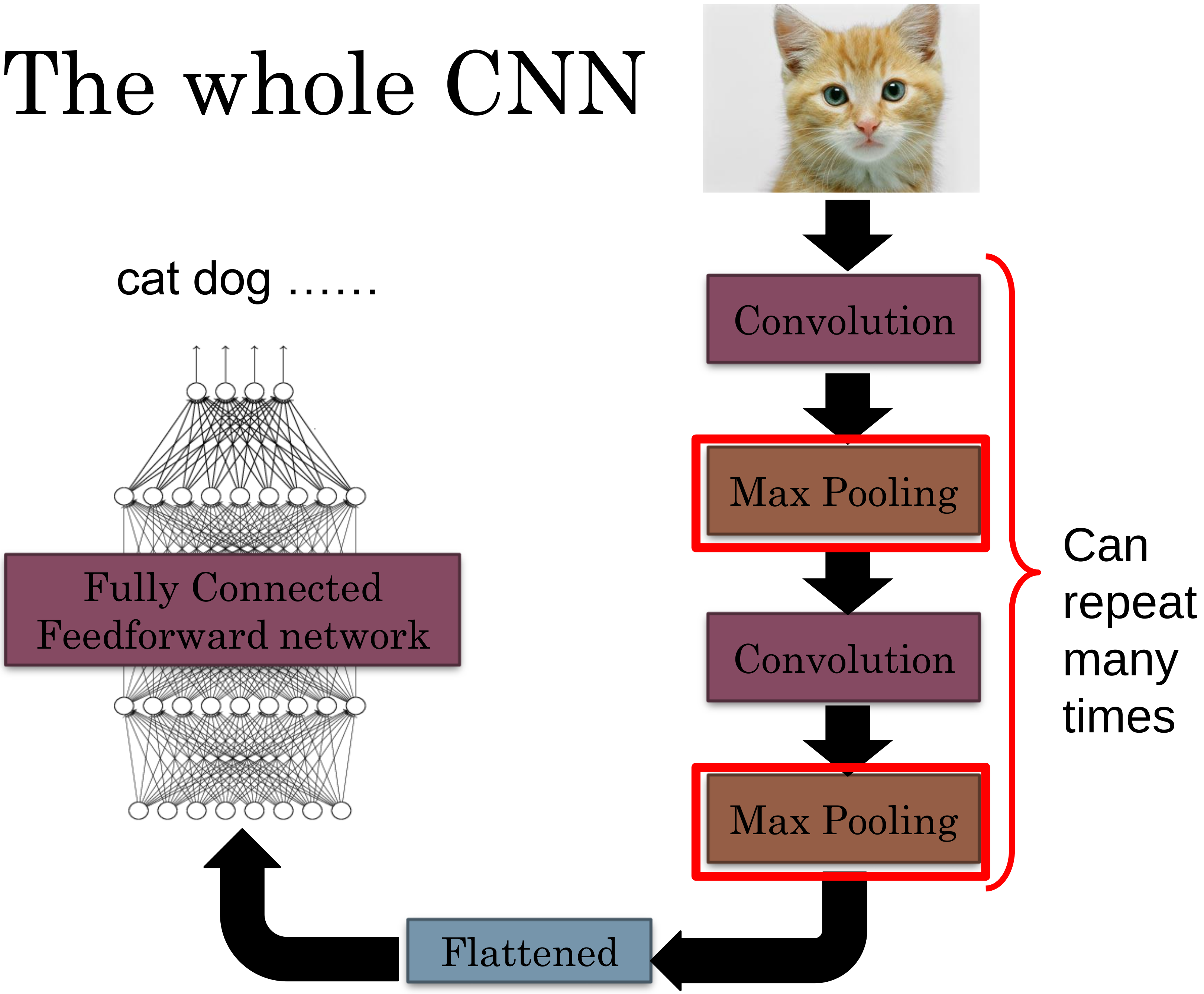
=

2	1	-1	2	2
-1	1	3	2	1
2	1	1	1	2
1	1	1	0	2
2	0	2	3	1

5 x 5

$m \times m$

The whole CNN



Example of Max Pooling

Stride = 2

Kernel = 2x2

4	123	1	34
56	99	222	253
45	122	165	12
21	187	133	124

MaxPool Operation →

123	167
187	165

Why Pooling

- Subsampling pixels will not change the object

bird

Subsampling

bird

We can subsample the pixels to make image smaller

fewer parameters to characterize the image

What a Stride of 1 Looks Like

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

Input Image

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2		

Output or Feature Map

What a Stride of 1 Looks Like

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

Input Image

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	1	

Output or Feature Map

What a Stride of 1 Looks Like

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

Input Image

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	1	-1

Output or Feature Map

What a Stride of 1 Looks Like

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

Input Image

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	1	-1
-1		

Output or Feature Map

What a Stride of 1 Looks Like

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

Input Image

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	1	-1
-1	1	

Output or Feature Map

What a Stride of 1 Looks Like

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 1 & -1 \\ -1 & 1 & 3 \\ & & \end{bmatrix}$$

Input Image

Filter or Kernel

Output or Feature Map

What a Stride of 1 Looks Like

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

Input Image

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	1	-1
-1	1	3
2		

Output or Feature Map

What a Stride of 1 Looks Like

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

Input Image

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	1	-1
-1	1	3
2	1	

Output or Feature Map

What a Stride of 1 Looks Like

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

Input Image

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	1	-1
-1	1	3
2	1	1

Output or Feature Map

What about a Stride of 2?

What a Stride of 2 Looks Like

We start off in the same position

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

Input Image

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	

Output or Feature Map

What a Stride of 2 Looks Like

Now we jump two spots to the left

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

Input Image

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	-1

Output or Feature Map

What a Stride of 2 Looks Like

Now we go down, but by also 2 spots

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

*

0	1	0
1	0	-1
0	1	0

=

2	-1
2	

Input Image

Filter or Kernel

Output or Feature Map

What a Stride of 2 Looks Like

Now we jump two spots to the left

$$
\begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}
*
\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}
=
\begin{bmatrix} 2 & -1 \\ 2 & 1 \end{bmatrix}
$$

Input Image

Filter or Kernel

Output or Feature Map

Stride Observations

- A larger Stride produced a **smaller** Feature Map output
- Larger Stride has **less overlap**
- We can use stride to **control the size of the Feature Map output**

Calculating Output Size

Using Stride and Padding

Stride = 2
Padding = 0

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & -1 \\ 2 & 1 \end{bmatrix}$$

Input Image
5 x 5

Filter or Kernel
3 x 3

$$(n \times n) * (f \times f) = \left(\frac{n + 2p - f}{s} + 1\right) \times \left(\frac{n + 2p - f}{s} + 1\right) = \left(\frac{5 + (2 \times 0) - 3}{2} + 1\right) \times \left(\frac{5 + (2 \times 0) - 3}{2} + 1\right) = 2 \times 2$$

Calculating Output Size

Using Stride and Padding

Stride = 1
Padding = 0

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

Input Image
5 x 5

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel
3 x 3

=

2	-1	1
2	1	0
1	-1	1

$$(n \times n) * (f \times f) = \left(\frac{n + 2p - f}{s} + 1\right) \times \left(\frac{n + 2p - f}{s} + 1\right) = \left(\frac{5 + (2 \times 0) - 3}{1} + 1\right) \times \left(\frac{5 + (2 \times 0) - 3}{1} + 1\right) = 3 \times 3$$

Calculating Output Size

When using Stride & Padding

Stride = 1
Padding = 1

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

Input Image
7 x 7

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel
3 x 3

=

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

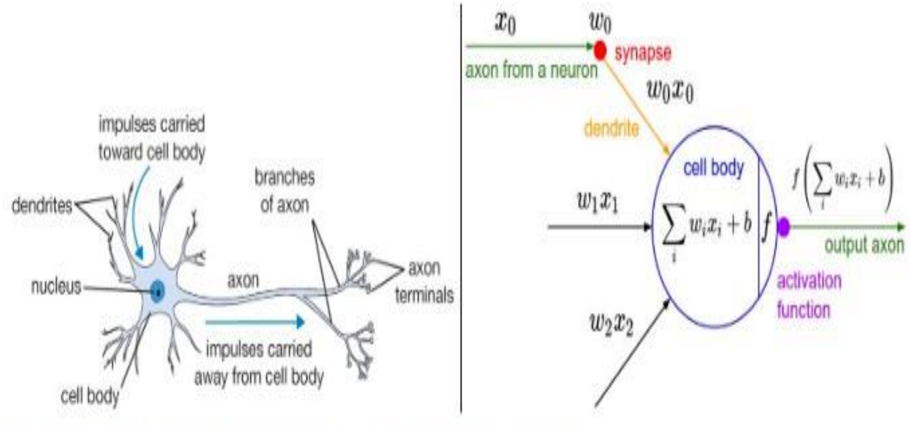
$$(n \times n) * (f \times f) = \left(\frac{n + 2p - f}{s} + 1\right) \times \left(\frac{n + 2p - f}{s} + 1\right) = \left(\frac{5 + (2 \times 1) - 3}{1} + 1\right) \times \left(\frac{5 + (2 \times 1) - 3}{1} + 1\right) = 5 \times 5$$

Note if we get non integer value for our output size, we found it down to the nearest integer

Purpose of Activation Functions

To enable the learning of **complex patterns** in our data

- Biological neurons fire (activate) on certain inputs, these are then fed into other neurons
- Introduces **non-linearity** to our network
- This allows a non-linear decision boundary via non-linear combinations of the weight and inputs



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

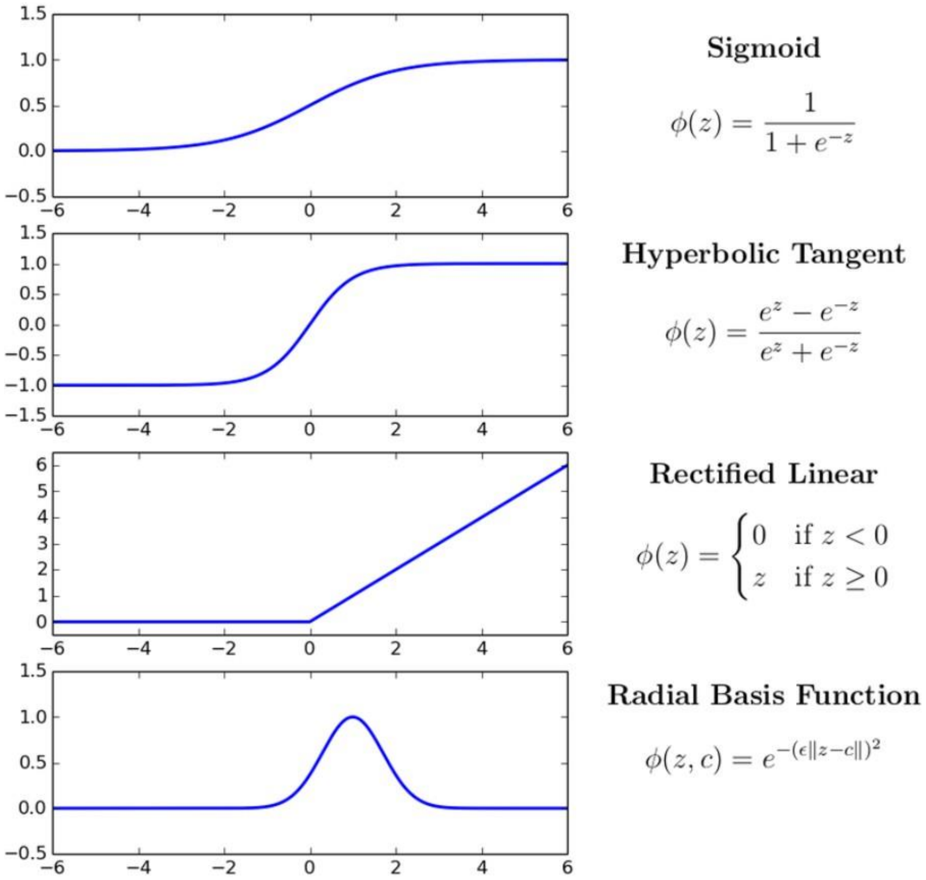
Stanford's CS231 Course

Types of Activation Functions

There are several activation functions we can use in our CNN. However, Rectified Linear Units (ReLU) have become the activation function of choice for CNNs.

ReLU is advantageous in CNN Training:

- Simple Computation (fast to train)
- Does not saturate



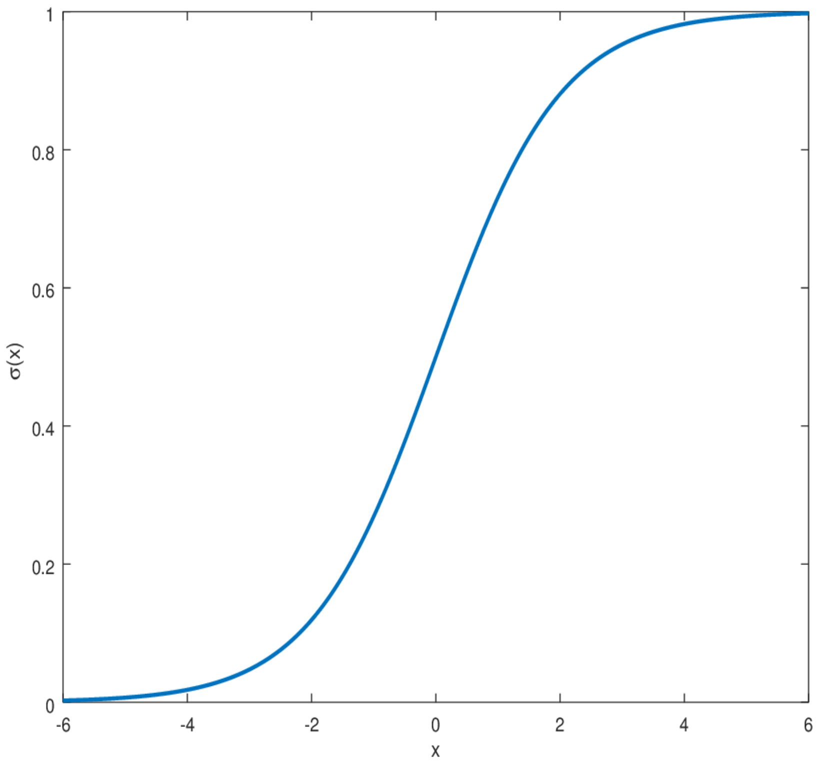
Sigmoid Function (Logistic)

- **Explanation**: The sigmoid function squashes the input values between 0 and 1, making it useful in binary classification problems where we need to produce probabilities.

- **Formula**:

$$f(x) = \frac{1}{1 + e^{-x}}$$

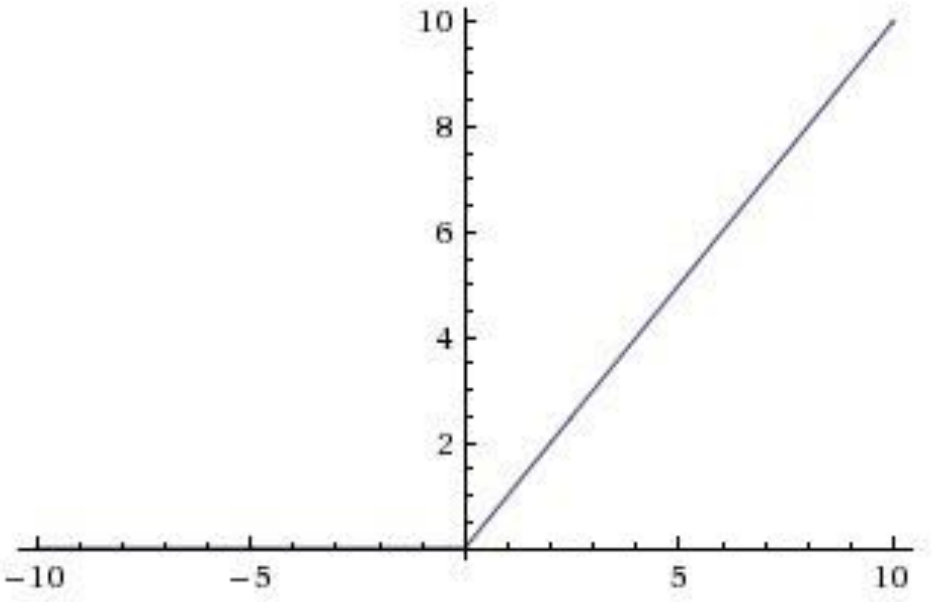
- **Usage**: Commonly used in the output layer for binary classification tasks, where it transforms the network's raw output into probabilities.



The ReLU Operation

- Change all negative values to 0
- Leave all positive Values alone

$$f(x) = max(0,x)$$



Applying the ReLU Activation

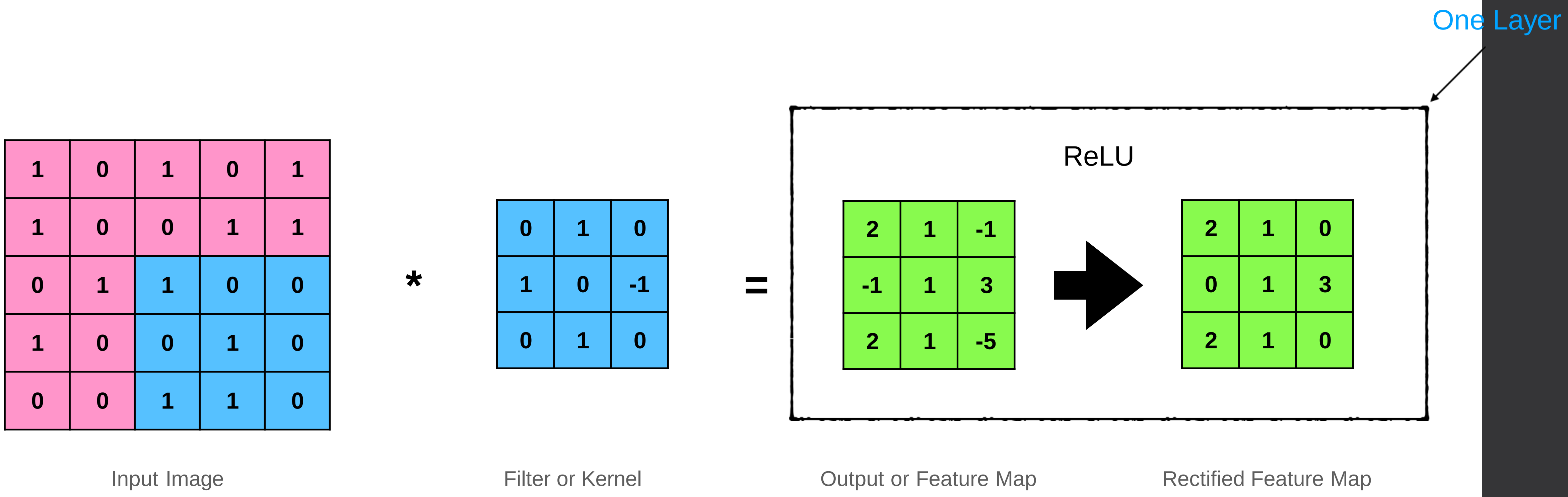
$$\begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 1 & -1 \\ -1 & 1 & 3 \\ 2 & 1 & -5 \end{bmatrix} \xrightarrow{\text{ReLU}} \begin{bmatrix} 2 & 1 & 0 \\ 0 & 1 & 3 \\ 2 & 1 & 0 \end{bmatrix}$$

Input Image

Filter or Kernel

Output or Feature Map

Applying the ReLU Activation



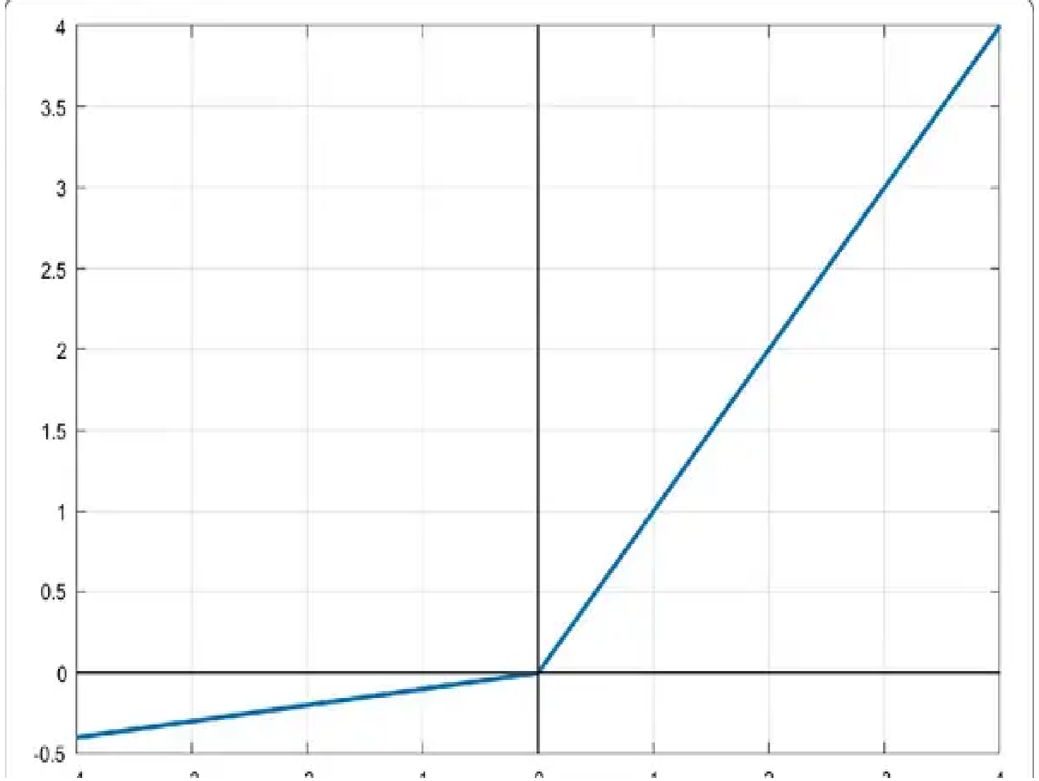
Leaky ReLU

- Dying ReLU → neurons using the ReLU activation function become inactive during training and never recover.
- Leaky ReLU is similar to ReLU but has a small slope for negative inputs, which helps mitigate the "dying ReLU" problem.
- **Formula**:

$$f(x) = max\ (0.1x, x)$$

- **Usage**: Leaky ReLU addresses the "dying ReLU" problem by allowing a small gradient for negative inputs, which can be beneficial in training deep networks.

Solution



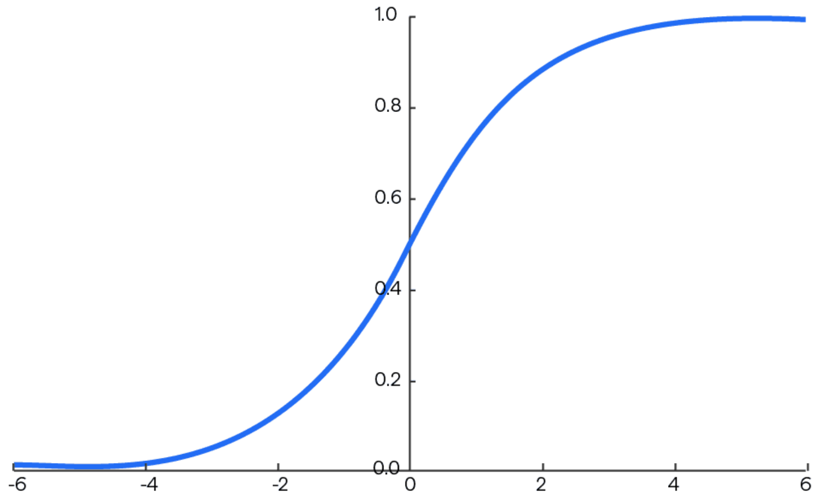
Softmax Function

- **Explanation**: The softmax function is commonly used in the output layer of neural networks for multi-class classification problems. It converts raw scores (logits) into probabilities, ensuring that the sum of the probabilities for all classes is equal to 1.

- **Formula**:

$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

- **Usage**: Softmax transforms the final layer activations into a probability distribution, allowing the model to make predictions about the likelihood of each class.

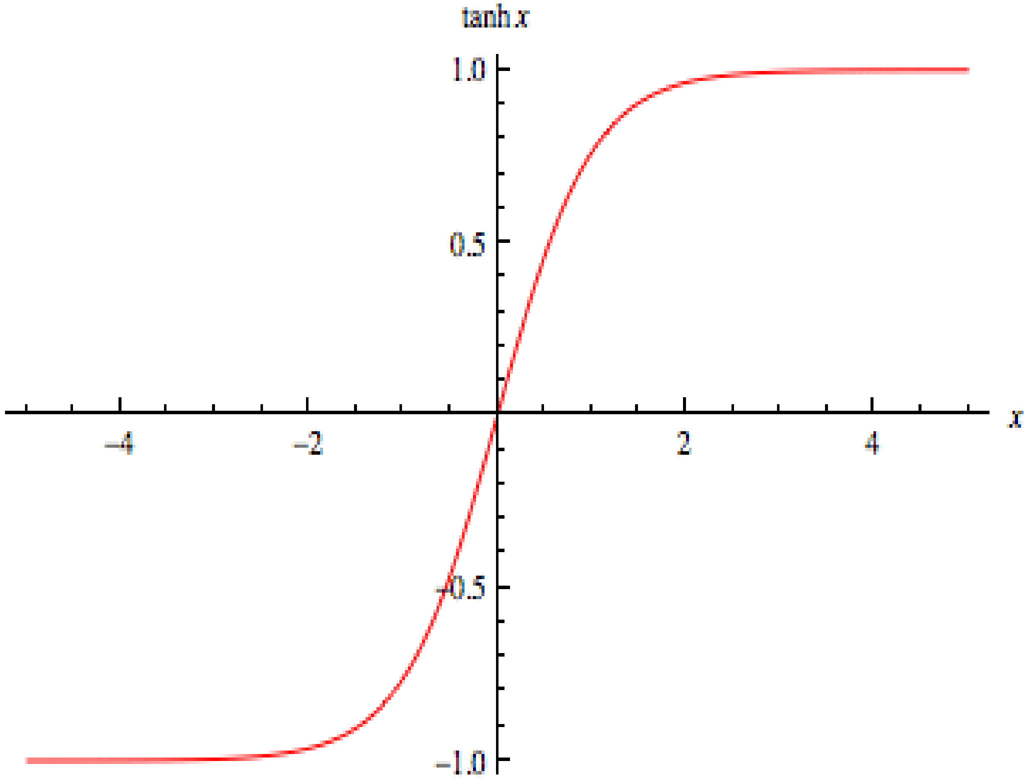


Hyperbolic Tangent Function (Tanh)

- **Explanation**: Tanh function squashes the input values between -1 and 1, which can be useful for normalization and also in classification tasks.
- **Formula**:

$$f(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$$

- **Usage**: Similar to other activations function, tanh is used in the hidden layers of neural networks. Specifically useful in GANs and GenAI.



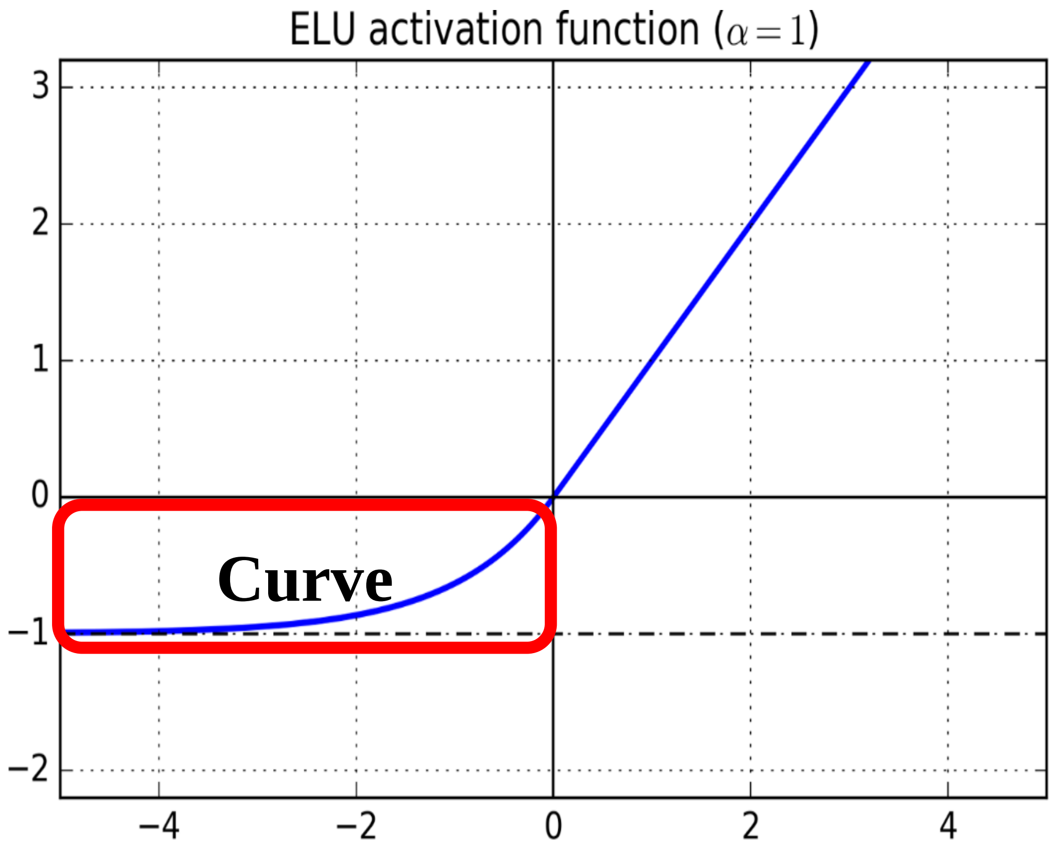
Exponential Linear Unit (ELU)

- **Explanation**: ELU is similar to ReLU for positive inputs but allows negative values with a smooth curve, aiming to address some limitations of ReLU.

- **Formula**

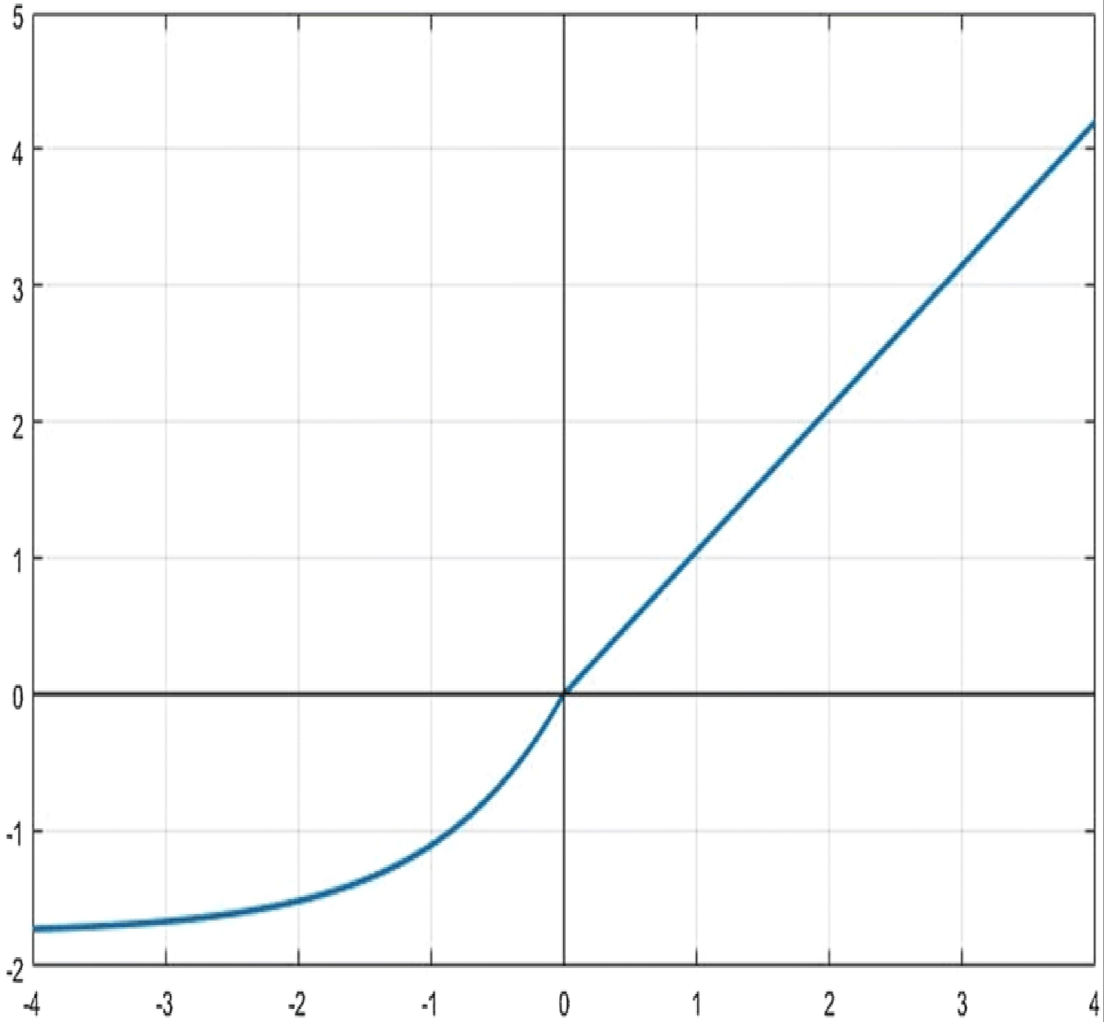
$$\mathrm{elu}(x) = \begin{cases} x, & x > 0 \\ \alpha\,(\exp(x) - 1), & x \leq 0 \end{cases}$$

- **Usage**: ELU mitigates the limitations of ReLU by handling negative inputs with a smooth curve, which can improve the robustness of deep networks.



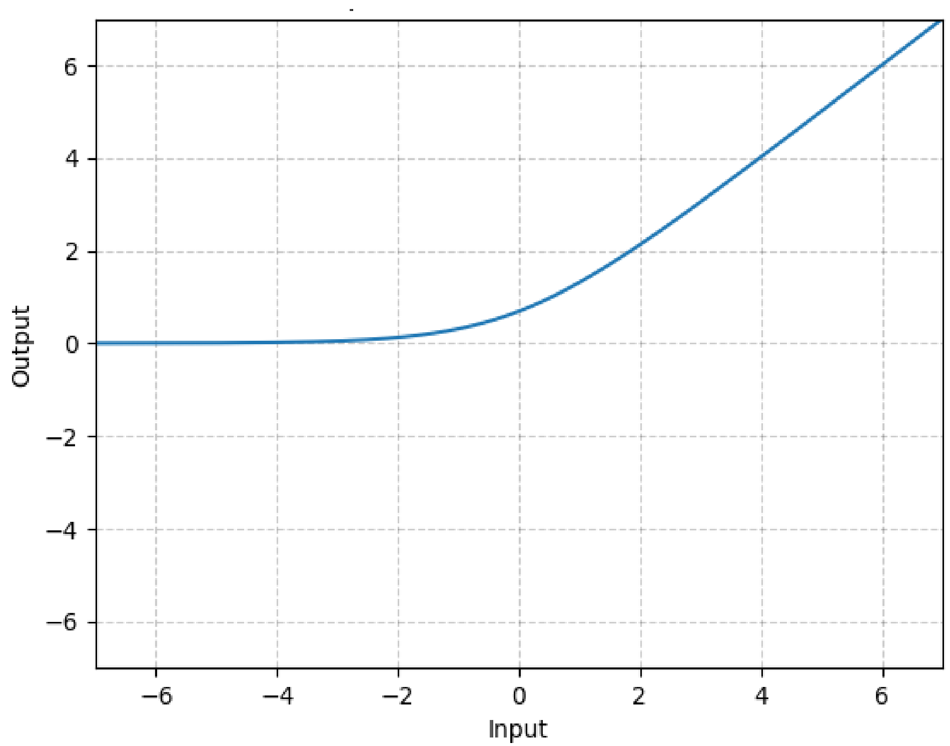
Scaled Exponential Linear Unit (SELU)

- **Explanation**: SELU is designed to maintain the mean and variance of the activations close to 0 and 1 respectively, aiding convergence.
- $$f(\alpha, x) = \lambda \begin{cases} \alpha\left(e^{x} - 1\right) & for\ x < 0 \\ x & for\ x \geqslant 0 \end{cases}$$
- **Usage**: SELU is designed to maintain the stability of activations throughout the network, often leading to better convergence and performance in deep architectures.



Softplus Function

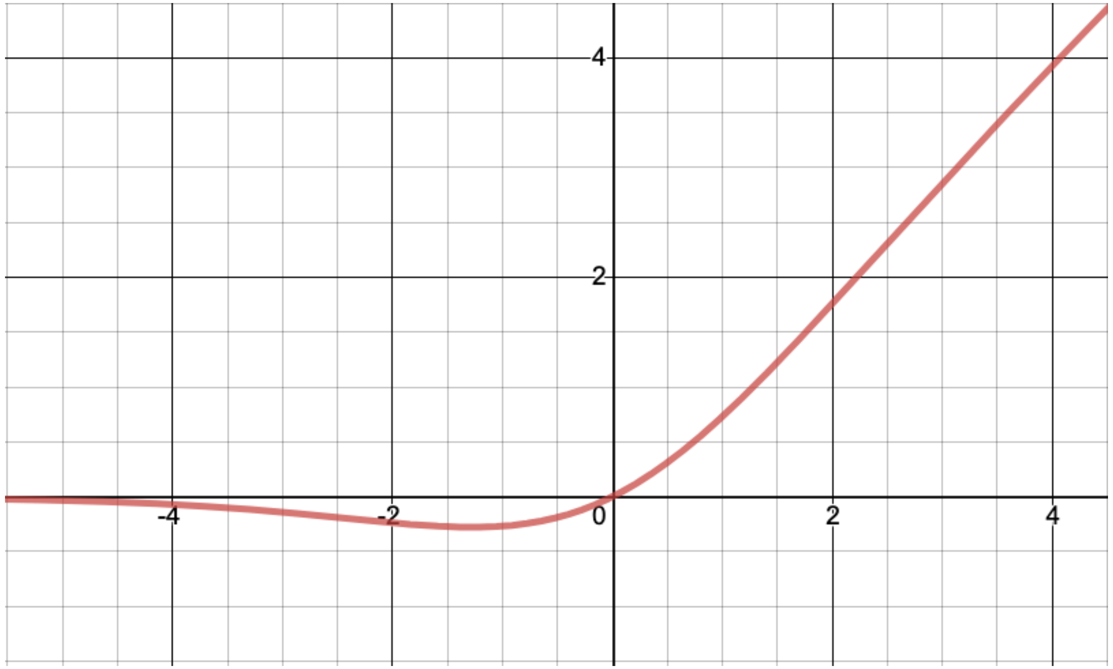
- **Explanation**: Softplus is a smooth version of ReLU and can be used as an alternative activation function in some cases.
- **Formula**
- $f(x) = In(1 + e^{x})$



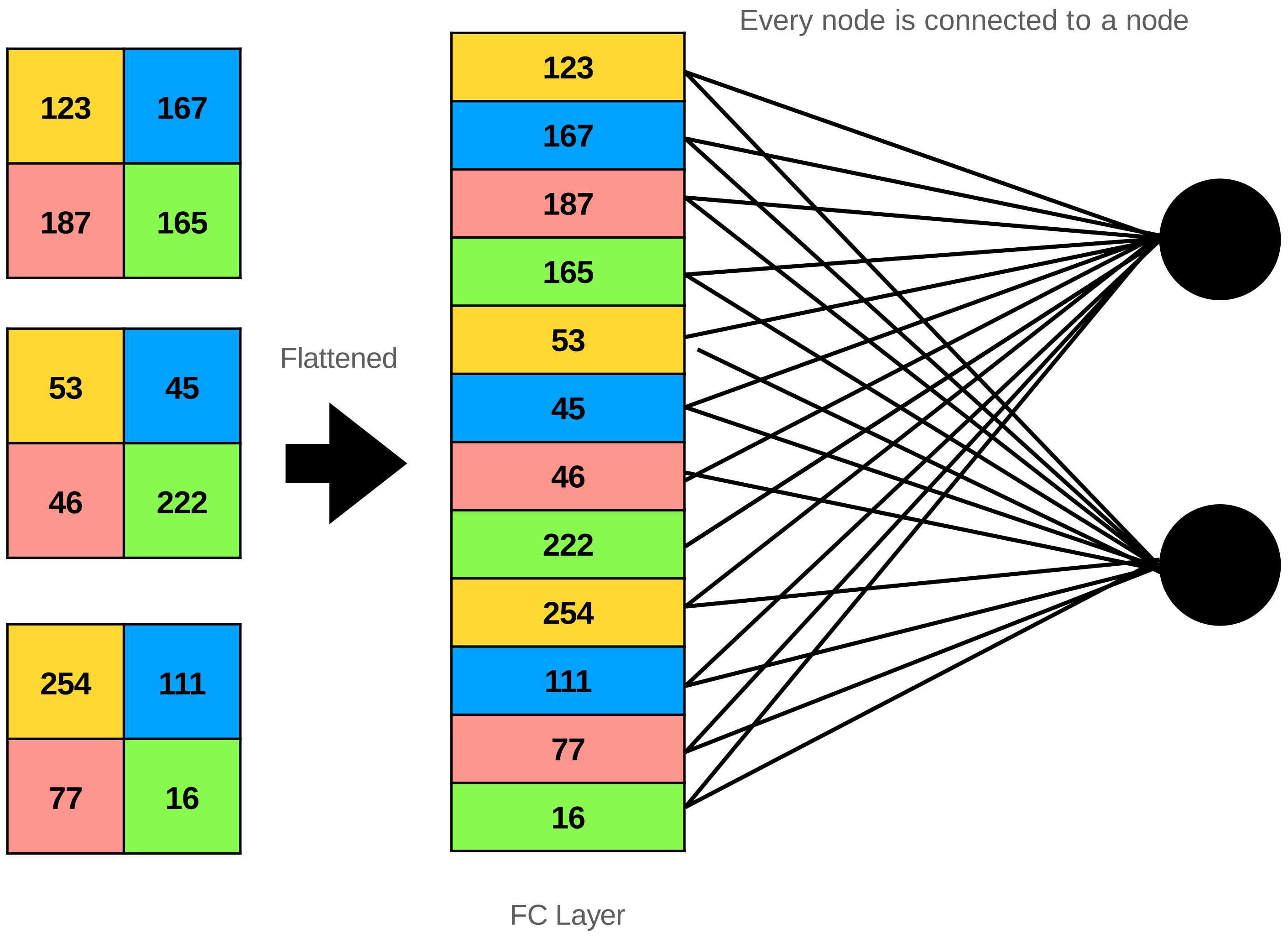
- **Usage**: Softplus is a smooth approximation of ReLU and can be used in scenarios where a differentiable activation function is required.

Swish Function

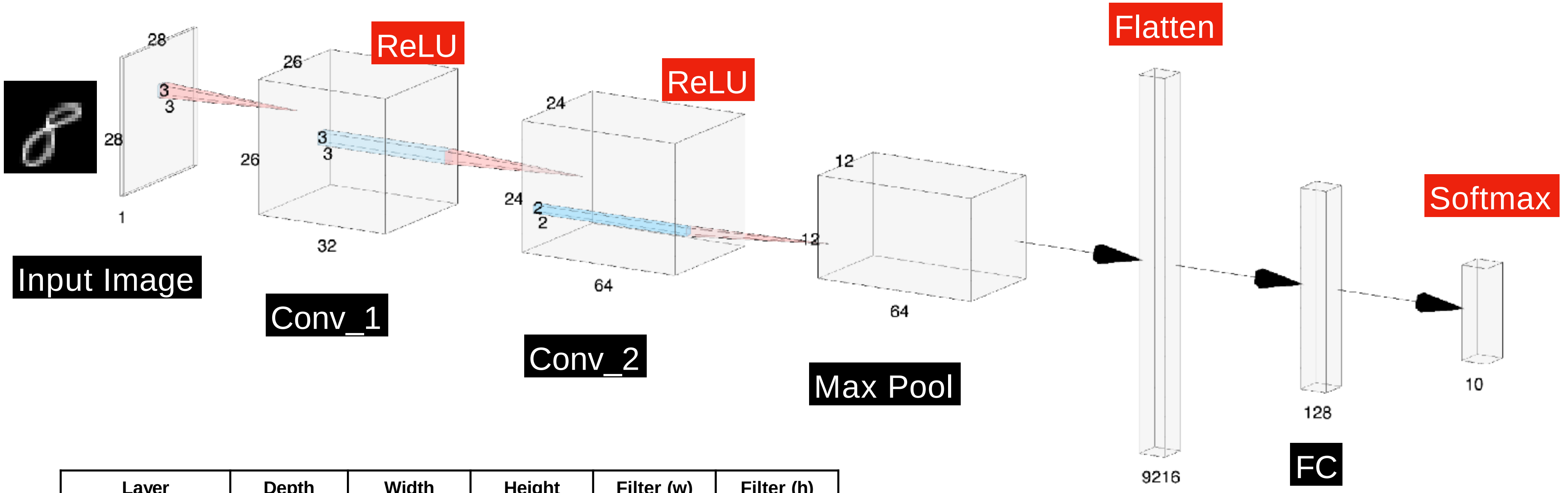
- **Explanation**: Swish function is a recently proposed activation function that tends to perform better than ReLU in certain scenarios.
- **Formula**
- $f(x) = x \cdot \text{sigmoid}(x)$
- **Usage**: Swish is an alternative to ReLU, offering potentially better performance, especially in large-scale datasets and deeper networks.



FC Layer - The Max Pool Layer is Flattened



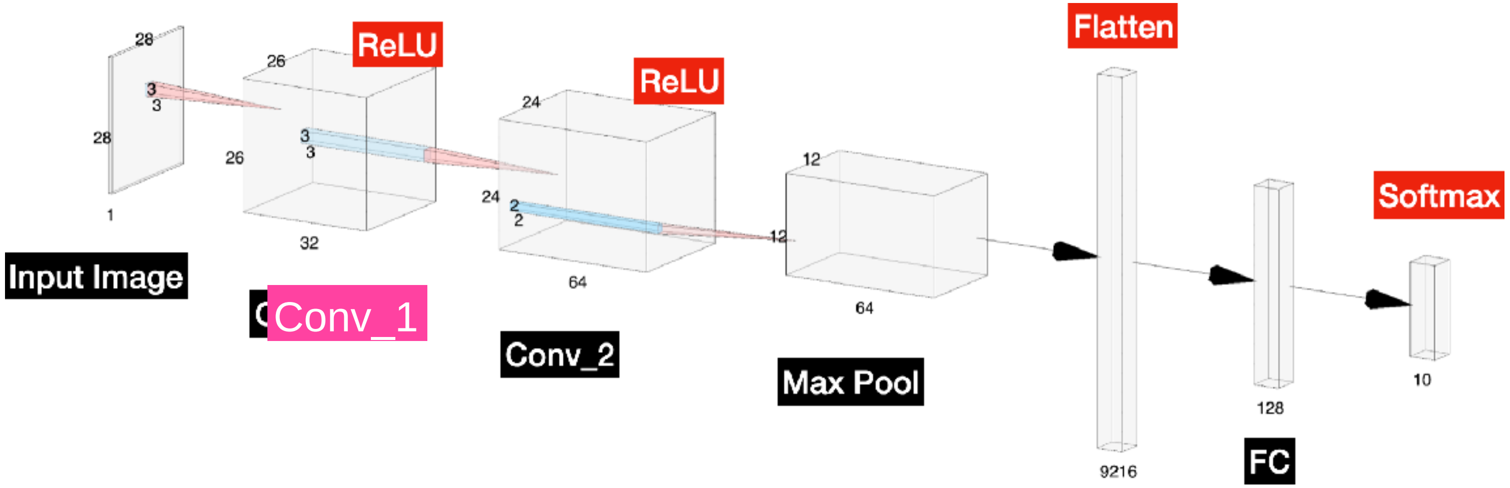
Another Representation



Layer	Depth	Width	Height	Filter (w)	Filter (h)
Input	1	28	28		
Conv_1	32	26	26	3	3
Conv_2	64	24	24	3	3
Max Pool	64	12	12	2	2
Flatten	9216	1	1		
Fully Connected	128	1	1		
Output	10	1	1		

Notes:
- We choose 32 & 64 Filters or Kernels for Conv_1 & Conv_2
- We choose to
- The Feature Maps are shown as Conv_1 and Conv_2
- Stride is 1
- Padding is 0 (not used)
- Max Pool Stride is 2

Calculating the Output Size of Conv_1



Notes:

- We choose 32 Filters or Kernels for Conv_1
- The Feature Maps are shown as Conv_1 & Conv_2
- Stride is 1
- Padding is 0 (not used)
- Max Pool Stride is 2

$$(n \times n) * (f \times f) = \left(\frac{n + 2p - f}{s} + 1\right) \times \left(\frac{n + 2p - f}{s} + 1\right) = \left(\frac{28 + (2 \times 0) - 3}{1} + 1\right) \times \left(\frac{28 + (2 \times 0) - 3}{1} + 1\right) = \text{26 x 26}$$

Where

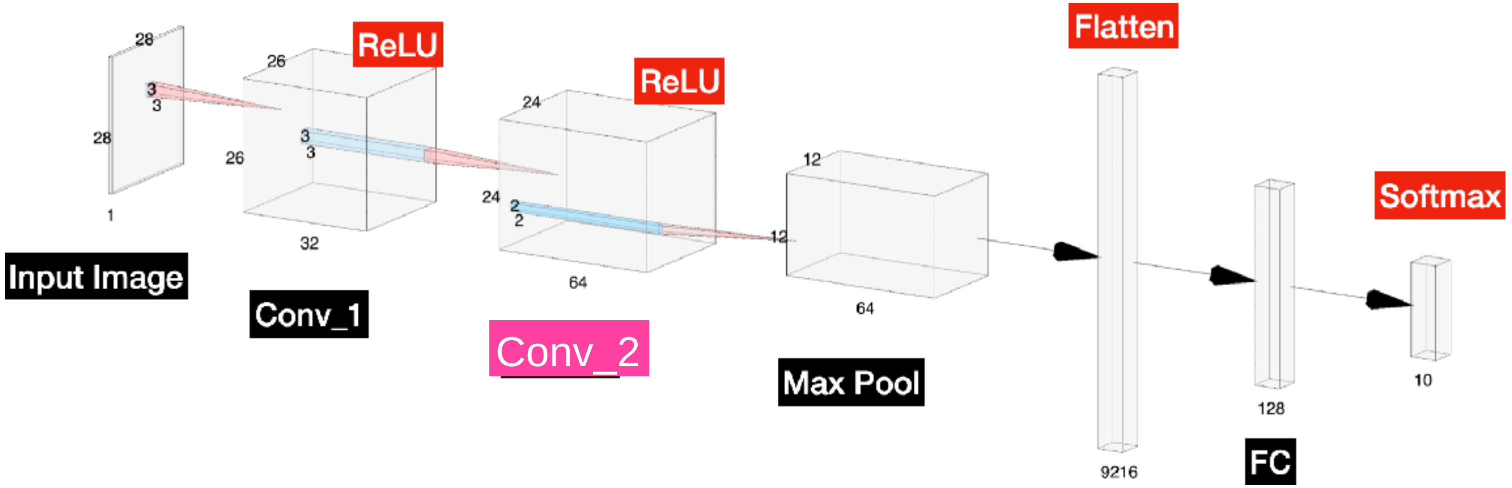
$n = 28$

$f = 3$

$s = 1$

$p = 0$

Calculating the Output Size of Conv_2



Notes:

- We choose 64 Filters or Kernels for Conv_2
- The Feature Maps are shown as Conv_1 & Conv_2
- Stride is 1
- Padding is 0 (not used)
- Max Pool Stride is 2

$$(n \times n) * (f \times f) = \left(\frac{n + 2p - f}{s} + 1\right) \times \left(\frac{n + 2p - f}{s} + 1\right) = \left(\frac{26 + (2 \times 0) - 3}{1} + 1\right) \times \left(\frac{26 + (2 \times 0) - 3}{1} + 1\right) = \text{24 x 24}$$

Where

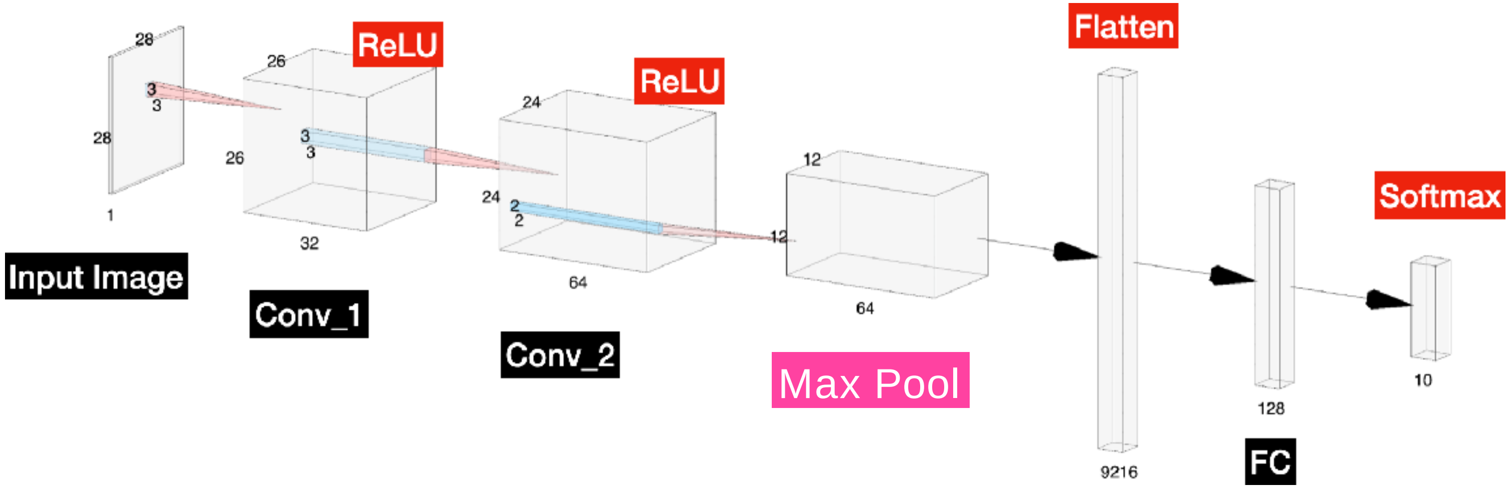
$n = 26$

$f = 3$

$s = 1$

$p = 0$

Calculating the Output Size of the Max Pool Layer



Notes:

- We choose 64 Filters or Kernels for Conv_2
- The Feature Maps are shown as Conv_1 & Conv_2
- Stride is 1
- Padding is 0 (not used)
- Max Pool Stride is 2

$$(n \times n) * (f \times f) = (\frac{n + 2p - f}{s} + 1) \times (\frac{n + 2p - f}{s} + 1) = (\frac{24 + (2 \times 0) - 2}{2} + 1) \times (\frac{24 + (2 \times 0) - 2}{2} + 1) = 12 \times 12$$

12 x 12

Where

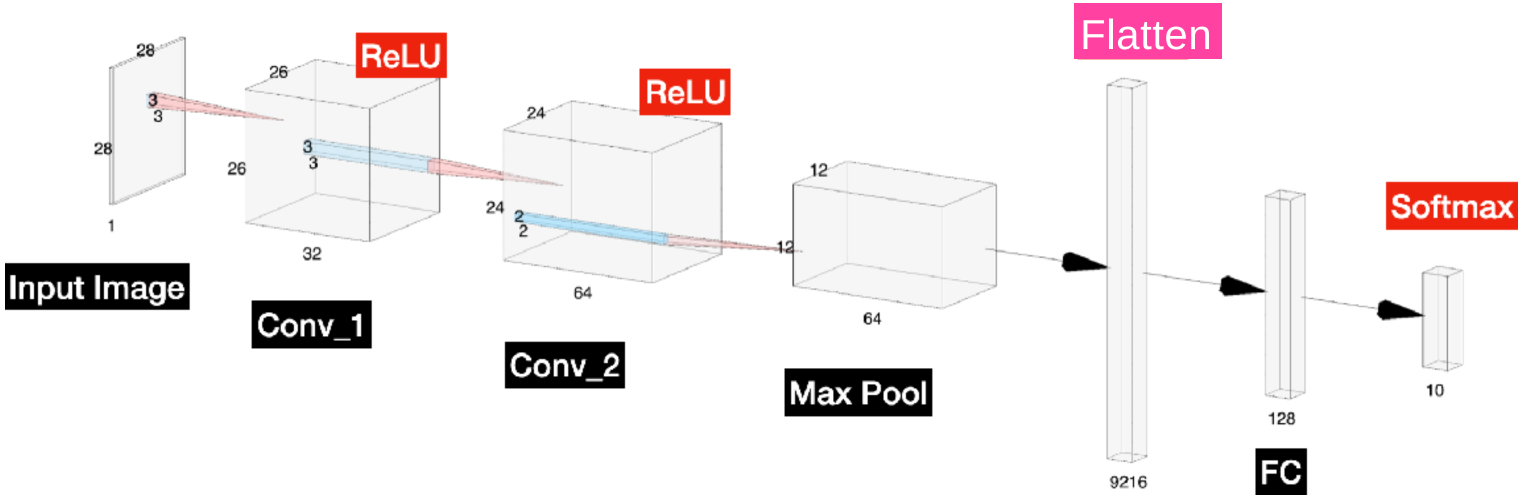
$n = 24$

$f = 2$

$s = 2$

$p = 0$

Calculating the Output Size of Flattened Layer



Notes:

- We choose 64 Filters or Kernels for Conv_2
- The Feature Maps are shown as Conv_1 & Conv_2
- Stride is 1
- Padding is 0 (not used)
- Max Pool Stride is 2

$$12 \times 12 \times 64 = 9216$$

Where

$n = 12$