Convolutional Neural Network (CNN)

Let’s Talk About
Images | Pixels | Videos | Frame

Computer Vision Fundamentals

Computer vision is a field of artificial intelligence that enables computers and systems to derive meaningful information from **digital images**, **videos**, and **other visual inputs**, and to act or make recommendations based on that information.

The key aspects of computer vision, summarized as main points:

- ❖ **Image Recognition:** Identifying objects, people, and other elements within images.
- ❖ **Object Detection:** Recognizing and locating objects within an image using bounding boxes or other markers.
- ❖ **Image Segmentation:** Dividing an image into parts to simplify the analysis, often used in applications like medical imaging.
- ❖ **Pattern Recognition:** Recognizing patterns in visual data, such as shapes or movements.
- ❖ **Scene Reconstruction:** Reconstructing a 3D scene from images, used in augmented reality and robotics.
- ❖ **Video Tracking:** Tracking objects or individuals across a video sequence.
- ❖ **Image Restoration:** Restoring or enhancing the quality of degraded images.

Image Concepts

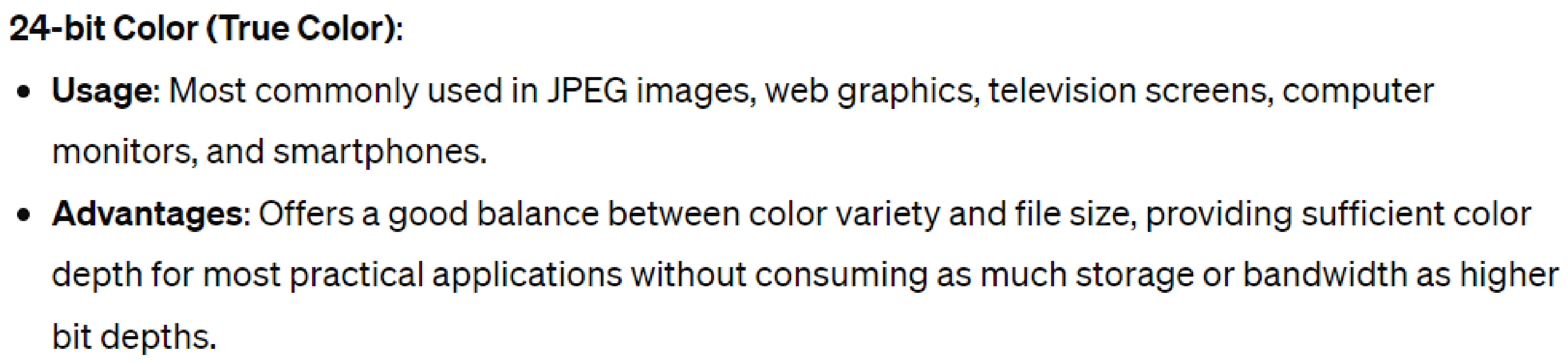
Image: In deep learning and computer vision, an image is a **digital representation of visual information**—like a photograph or a frame from a video—formatted in a way that can be processed by algorithms. The total number of pixels that represent the image is 1024x768 = 7,86,432. From **black at the lowest value (0)** to **white** at the **highest value (255)**, with shades of gray in between proportional to the value.

Image Size: 1024×768

Image Size: 1024×768

Pixel Concepts

A pixel, short for "picture element," is the **smallest unit of information in an image** or display. Each pixel represents a single point in the image. Pixels are arranged in a **grid pattern**. When viewed together at the proper distance, they form the complete image that our eyes perceive. An image with a resolution of 1920x1080 has 2,073,600 pixels.



24-bit Color (True Color):

- **Usage:** Most commonly used in JPEG images, web graphics, television screens, computer monitors, and smartphones.
- **Advantages:** Offers a good balance between color variety and file size, providing sufficient color depth for most practical applications without consuming as much storage or bandwidth as higher bit depths.

The term "True Color" is used to describe a color depth that allows for a representation of 24-bit color information. This configuration provides a total of 16,777,216 color variations. In True Color, each color component (Red, Green, and Blue) is allocated 8 bits, which allows each component 256 different intensity levels, leading to 256×256×256=16,777,216 possible colors.

Pixel Concepts

A pixel, short for "picture element," is the **smallest unit of information in an image** or display. Each pixel represents a single point in the image. Pixels are arranged in a **grid pattern**. When viewed together at the proper distance, they form the complete image that our eyes perceive. An image with a resolution of 1920x1080 has 2,073,600 pixels.

8-bit Grayscale:
For tasks where color information **might not add significant value**, converting images to **8-bit grayscale** can reduce computational requirements and streamline the learning process. This is common in tasks like text recognition, certain types of medical imaging, or when training efficiency is a priority.

Pixel Concepts

Color Representation:

- **8-bit Grayscale**: These images contain only shades of gray, meaning that each pixel carries information only about intensity or luminance, without any color data. The values range from 0 (black) to 255 (white), providing a total of 256 different shades of gray.

- **24-bit True Color**: True Color images use three color channels (Red, Green, and Blue), with each channel represented by 8 bits. This setup allows each channel to display 256 different intensity levels, combining to offer over 16 million possible colors (256 x 256 x 256).

Data Complexity and Size:

- **8-bit Grayscale**: In an 8-bit grayscale image, there is only one channel. These images are simpler and require less storage space and bandwidth because they consist of a single channel. An 8-bit grayscale image uses one byte per pixel.

- **24-bit True Color**: In a 24-bit color image, there is a total of three channels. These images are more complex, requiring three times the data for each pixel compared to grayscale images (since there are three channels, each using one byte). This makes them larger and more resource-intensive to process and store.

Pixel Concepts

1-bit (Monochrome):

- **Value Range**: 0-1
- **Colors**: Black and White (2 colors)

2-bit color:

- **Value Range**: 0-3
- **Colors**: 4 different colors or shades

3-bit color:

- **Value Range**: 0-7
- **Colors**: 8 different colors or shades

4-bit color:

- **Value Range**: 0-15
- **Colors**: 16 different colors or shades (often seen in early computer graphics)

Pixel Concepts

8-bit color:

- **Value Range**: 0-255
- **Colors**: 256 different colors or shades (common for GIFs and 8-bit grayscale images)

16-bit color (High Color):

- **Value Range**: 0-65535
- **Colors**: 65,536 different colors (used in older or lower-end display technologies)

24-bit color (True Color):

- **Value Range per Channel**: 0-255
- **Total Colors**: 16,777,216 colors (standard for JPEGs, web images, TVs, and monitors)

32-bit color:

- **Value Range per Channel**: 0-255 (24 bits for RGB and 8 bits for alpha)
- **Total Colors**: 16,777,216 colors, plus 256 levels of transparency

Pixel Concepts

48-bit color (used in professional photography):

- **Value Range per Channel:** 0-65535
- **Total Colors:** This effectively allows trillions of different color nuances (important in high-end photography and printing).

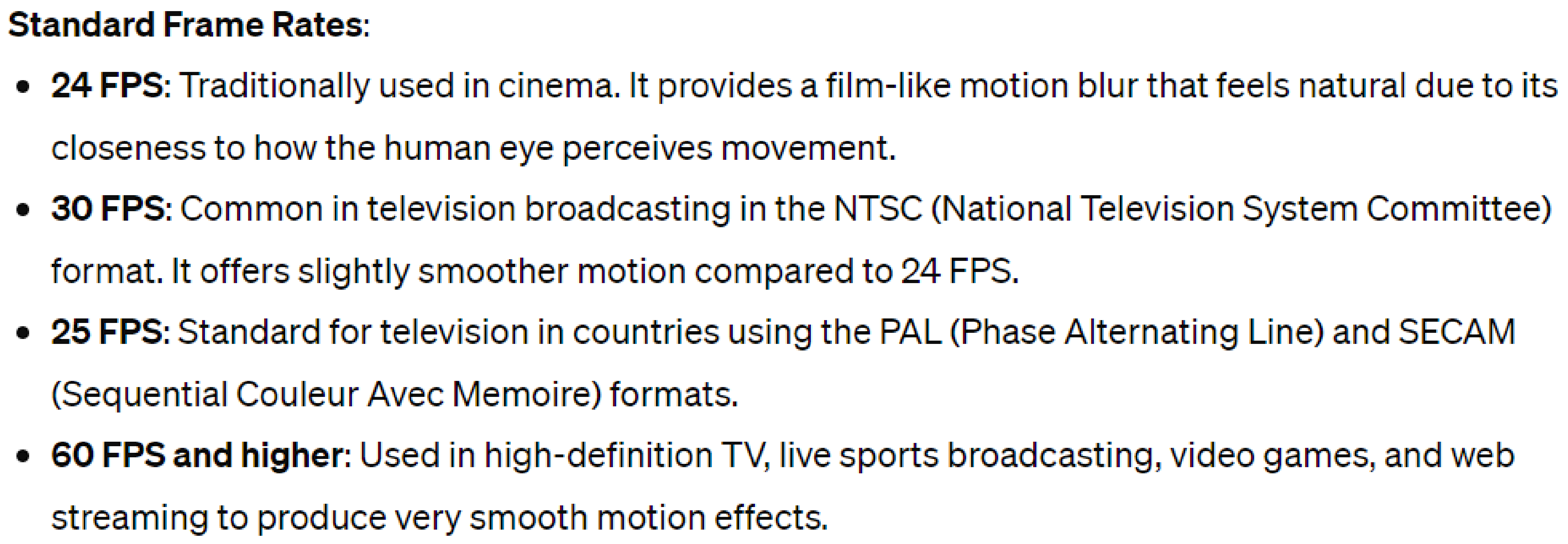
64-bit color (HDR and advanced graphics):

- **Value Range per Channel:** 0-65535 for RGB and alpha
- **Total Colors:** This provides deep color with high dynamic range capabilities.

Working with Videos

❖ **Frame:** A "frame" in video and animation refers to a single still image in a sequence of images that constitute a video or animated content. When multiple frames are displayed rapidly, they create the illusion of motion.

❖ **Frame Rate:** It is measured in frames per second (FPS) and indicates the frequency at which these frames are displayed or projected.



Standard Frame Rates:

- **24 FPS**: Traditionally used in cinema. It provides a film-like motion blur that feels natural due to its closeness to how the human eye perceives movement.
- **30 FPS**: Common in television broadcasting in the NTSC (National Television System Committee) format. It offers slightly smoother motion compared to 24 FPS.
- **25 FPS**: Standard for television in countries using the PAL (Phase Alternating Line) and SECAM (Sequential Couleur Avec Memoire) formats.
- **60 FPS and higher**: Used in high-definition TV, live sports broadcasting, video games, and web streaming to produce very smooth motion effects.

Working with Videos

1. Object Detection and Tracking:

- **Moderate to High FPS**: For tasks like real-time object tracking in video feeds or sports analytics, a higher frame rate (30-60 FPS) can be beneficial. It provides more temporal information and smoother transitions, which can improve the accuracy and robustness of tracking algorithms.

2. Action Recognition:

- **Variable FPS**: The necessary frame rate can vary depending on the speed of the actions being analyzed. Faster actions might require higher frame rates to capture relevant motion details effectively. Typically, 30 FPS is a good starting point, but more dynamic scenes might benefit from up to 60 FPS.

Working with Videos

3. Behavior Analysis:

- **Lower to Moderate FPS**: For general behavior analysis, such as monitoring crowd movements or analyzing consumer behavior in stores, lower frame rates (15-30 FPS) might be sufficient. These applications often prioritize broader trends over split-second details.

4. Surveillance:

- **Lower FPS Sufficient:** In surveillance, especially in scenarios with less dynamic scenes, lower frame rates (5-15 FPS) are often adequate. This helps save on storage and processing while still capturing enough temporal information for activity detection and person identification.

5. Medical Imaging and Analysis:

- **Specific FPS Requirements:** Certain medical applications, such as analyzing heart rate or other quick physiological changes, may require very specific frame rates defined by the speed of the physiological events.

Working with Videos

General Recommendations:

- **Start with Standard Video FPS:** Starting with a standard video frame rate like 30 FPS is often a safe choice. It provides a good balance between capturing motion and managing data and computational loads.
- **Experiment and Optimize:** It can be beneficial to experiment with different frame rates to find the best balance for the specific task, considering both performance metrics and computational efficiency.

Convolutional Neural Network (CNN)

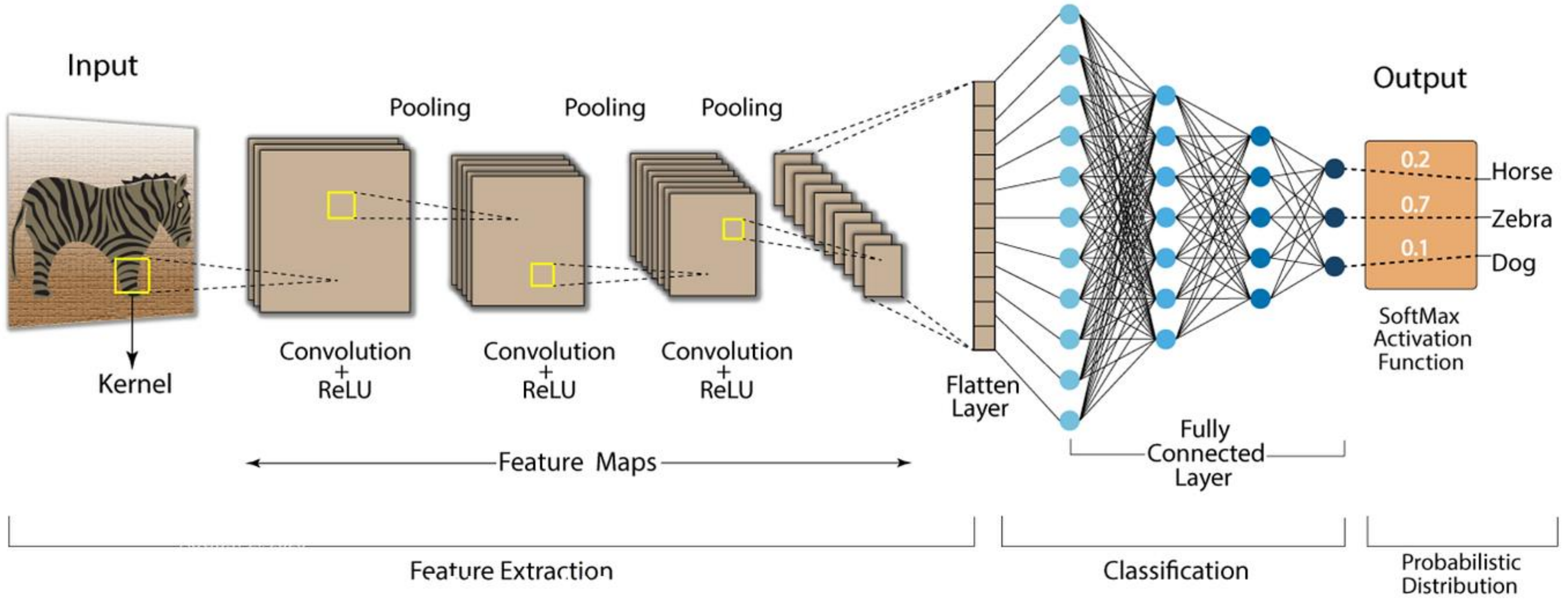


Fig: CNN Architecture

Convolutional Neural Network (CNN)

- **Input:** The process begins with an input image, which in this case is a **zebra.**

- **Padding**: Before applying the convolution operation, padding might be added around the border of the input image. Padding is used to **preserve the original size** of the image after convolution.
  - **Zero Padding (most common)**: Adds zeros around the border of the image.
  - **Reflective or Replication Padding**: Copies the edge values or reflects them around the border.

- **Convolution Layer:** To create feature maps, small matrices called **kernels** or filters are applied to the input image. These filters detect features such as edges and textures by sliding across the image. Each application of the filter results in a new feature map. (Next Slide)

- **ReLU Activation:** After convolution, the feature maps are passed through a Rectified Linear Unit (ReLU) activation function to introduce non-linearity, making the network capable of learning more complex patterns. It does this by turning all negative values to zero.

- **Pooling Layer:** After the ReLU activation, a pooling layer (typically max pooling) simplifies the output by reducing its dimensions but retaining the most important information. It selects the maximum value from a group of pixels in a feature map.

More About Convolution Layer in CNN

Suppose, your input image has **three** channels (such as an RGB image), and you use a single filter, you **do not** get three separate feature maps for each channel. Instead, you get a **single feature map** from the convolution operation. This is how it works:

- **Multi-channel Filter:** Each filter in a convolution layer is three-dimensional when working with multi-channel images like RGB images. The depth of the filter matches the number of channels in the input image. So, for an RGB image, each filter has three layers—one for each channel (Red, Green, Blue).

- **Convolution Operation:** During the convolution operation, each layer of the filter is applied to the corresponding channel of the image. This means the first layer of the filter processes the Red channel, the second layer processes the Green channel, and the third layer processes the Blue channel.

- **Summation:** The results of these three convolutions (one per channel) are then summed up to produce a single output value. This summation results in a single feature map per filter, even though the filter is applied to multiple channels.

Convolutional Neural Network (CNN)

- **Flatten Layer:** After several convolution and pooling layers, high-level reasoning in the neural network occurs. The data is flattened into a single vector to prepare for the fully connected layer, which needs a flat input. (Next Slide)

- **Fully Connected Layer:** This layer is a deep neural network that uses the flattened data from the previous layers. Every neuron in this layer is connected to all the activations in the previous layer, and it's where the actual classification process begins to take shape.

- **Output Layer with SoftMax Function:** The last step in the process involves the SoftMax activation function, which is applied in the output layer. This function converts the output into a probability distribution, representing the likelihood of the input image belonging to one of the predefined classes (like horse, zebra, or dog).

- **Classification Output:** The output is a probabilistic distribution where each class (type of animal in this case) is assigned a probability score, indicating the confidence of the network in predicting each class.

More About Flatten Layer in CNN

Suppose, you will provide a **20x20** 2D data array to a **flatten layer**, it will convert this 2D array into a 1D vector as 400 individual elements.

How does it work when you use 100 neurons for a 400-element 1D input vector?

- **All Connections:** Each of the neurons is connected to every element of the input vector. For example, 100 neurons would each connect to all 400 elements from the input.

- **Weighted Sum and Bias:** Every connection has a weight. Each neuron computes a sum of the input elements multiplied by these weights, plus a bias term. This integrates information from all 400 points.

Mathematically: $\sum_{j=1}^{400} w_{ij} \cdot x_j + b_i$, where b_i is the bias for neuron i.

- **Activation Function:** This sum is then processed through an activation function, adding non-linearity and allowing the network to learn complex patterns.

- **The output of the Layer:** The output is a new vector, the size of which equals the number of neurons (for example 100). This output represents a condensed form of the input, capturing essential features.

Convolutional Neural Network (CNN)

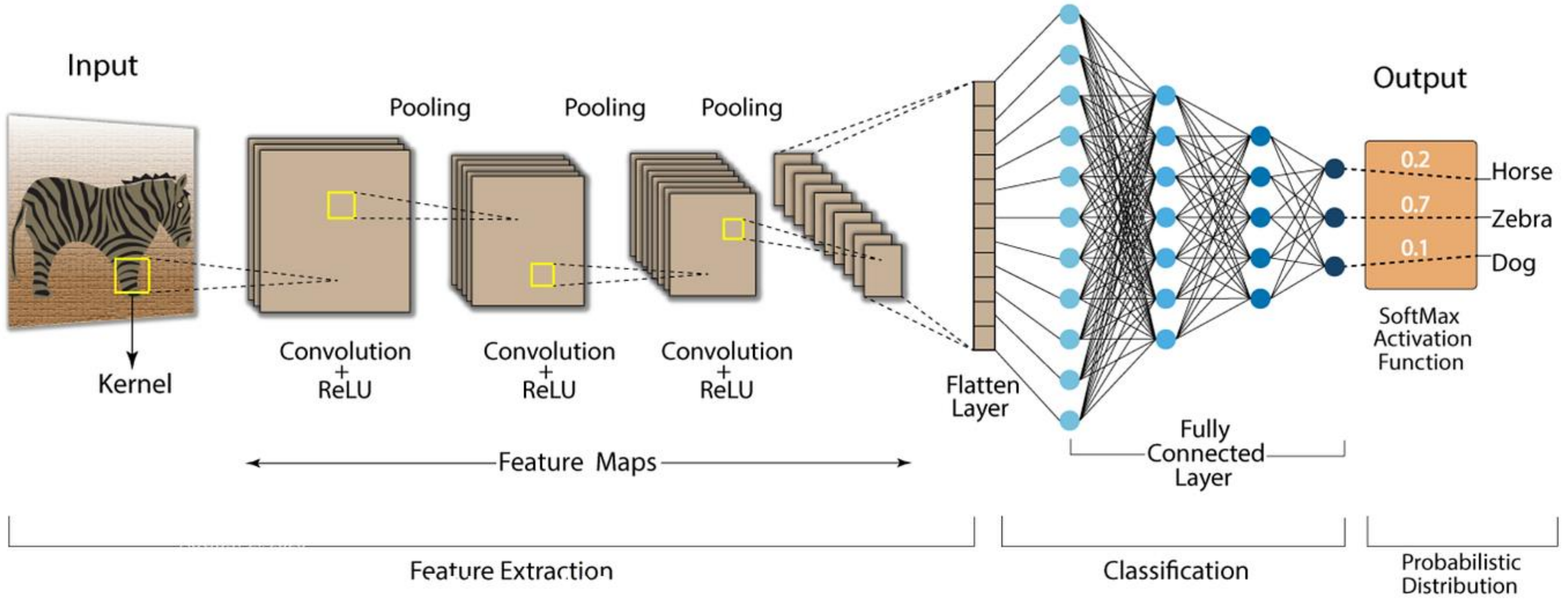
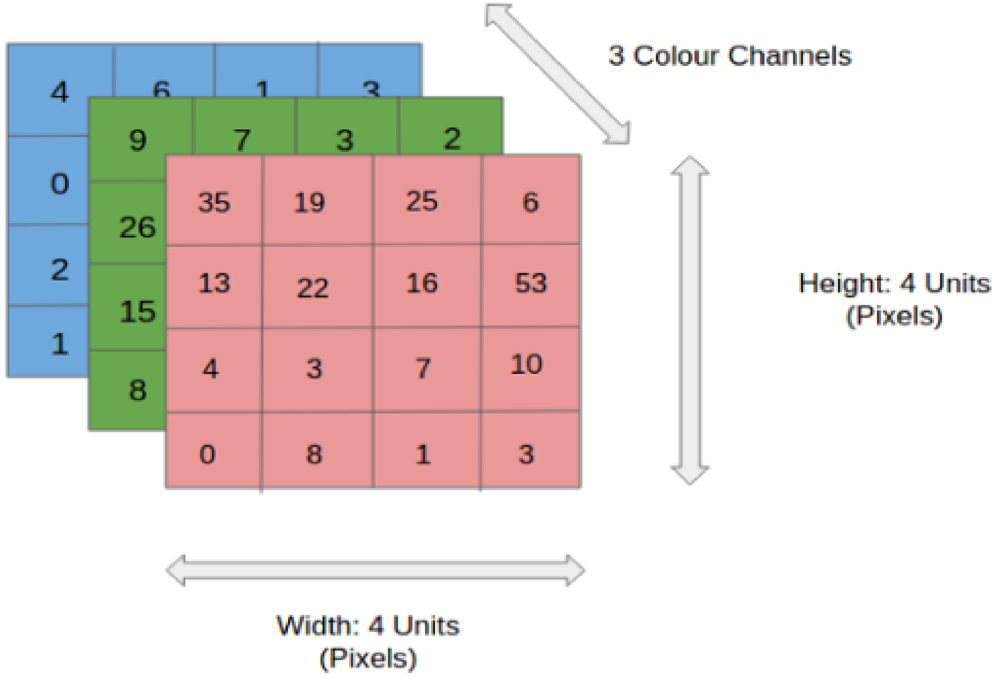


Fig: CNN Architecture

Input Layer

Fig: RGB Image

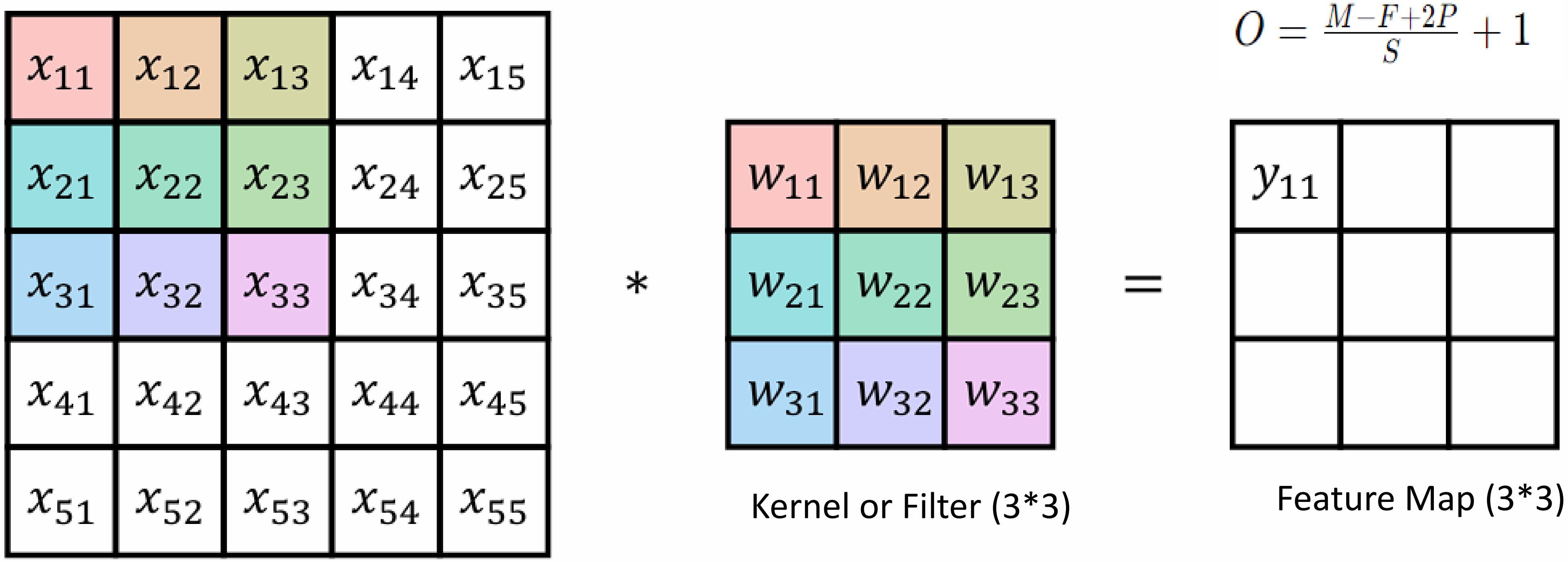


Convolutional Neural Network (CNN)

The output computation now only depends on a subset of inputs:

$$y_{11} = w_{11}x_{11} + w_{12}x_{12} + w_{13}x_{13} + w_{21}x_{21} + w_{22}x_{22} + w_{23}x_{23} + w_{31}x_{31} + w_{32}x_{32} + w_{33}x_{33}$$

$$O = \frac{M-F+2P}{S} + 1$$



x_{11} x_{12} x_{13} x_{14} x_{15}
x_{21} x_{22} x_{23} x_{24} x_{25}
x_{31} x_{32} x_{33} x_{34} x_{35}
x_{41} x_{42} x_{43} x_{44} x_{45}
x_{51} x_{52} x_{53} x_{54} x_{55}
*
w_{11} w_{12} w_{13}
w_{21} w_{22} w_{23}
w_{31} w_{32} w_{33}
=
y_{11}

Kernel or Filter (3*3)

Feature Map (3*3)

Input Image (5*5)

Convolutional Neural Network (CNN)

The output computation now only depends on a subset of inputs:

$$y_{12} = w_{11}x_{12} + w_{12}x_{13} + w_{13}x_{14} + w_{21}x_{22} + w_{22}x_{23} + w_{23}x_{24} + w_{31}x_{32} + w_{32}x_{33} + w_{33}x_{34}$$

$$O = \frac{M - F + 2P}{S} + 1$$

$$\begin{array}{|c|c|c|c|c|}\hline x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ \hline x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \\ \hline x_{31} & x_{32} & x_{33} & x_{34} & x_{35} \\ \hline x_{41} & x_{42} & x_{43} & x_{44} & x_{45} \\ \hline x_{51} & x_{52} & x_{53} & x_{54} & x_{55} \\ \hline \end{array} \; * \; \begin{array}{|c|c|c|}\hline w_{11} & w_{12} & w_{13} \\ \hline w_{21} & w_{22} & w_{23} \\ \hline w_{31} & w_{32} & w_{33} \\ \hline \end{array} \; = \; \begin{array}{|c|c|c|}\hline y_{11} & y_{12} & \phantom{y_{11}} \\ \hline & & \\ \hline & & \\ \hline \end{array}$$

Input Image (5*5)

Kernel or Filter (3*3)

Feature Map (3*3)

Convolutional Neural Network (CNN)

The output computation now only depends on a subset of inputs:

$$y_{13} = w_{11}x_{13} + w_{12}x_{14} + w_{13}x_{15} + w_{21}x_{23} + w_{22}x_{24} + w_{23}x_{25} + w_{31}x_{33} + w_{32}x_{34} + w_{33}x_{35}$$

$$O = \frac{M-F+2P}{S} + 1$$

$$\begin{array}{|c|c|c|c|c|}\hline x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ \hline x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \\ \hline x_{31} & x_{32} & x_{33} & x_{34} & x_{35} \\ \hline x_{41} & x_{42} & x_{43} & x_{44} & x_{45} \\ \hline x_{51} & x_{52} & x_{53} & x_{54} & x_{55} \\ \hline \end{array} \ast \begin{array}{|c|c|c|}\hline w_{11} & w_{12} & w_{13} \\ \hline w_{21} & w_{22} & w_{23} \\ \hline w_{31} & w_{32} & w_{33} \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline y_{11} & y_{12} & y_{13} \\ \hline & & \\ \hline & & \\ \hline \end{array}$$

Input Image (5*5)

Kernel or Filter (3*3)

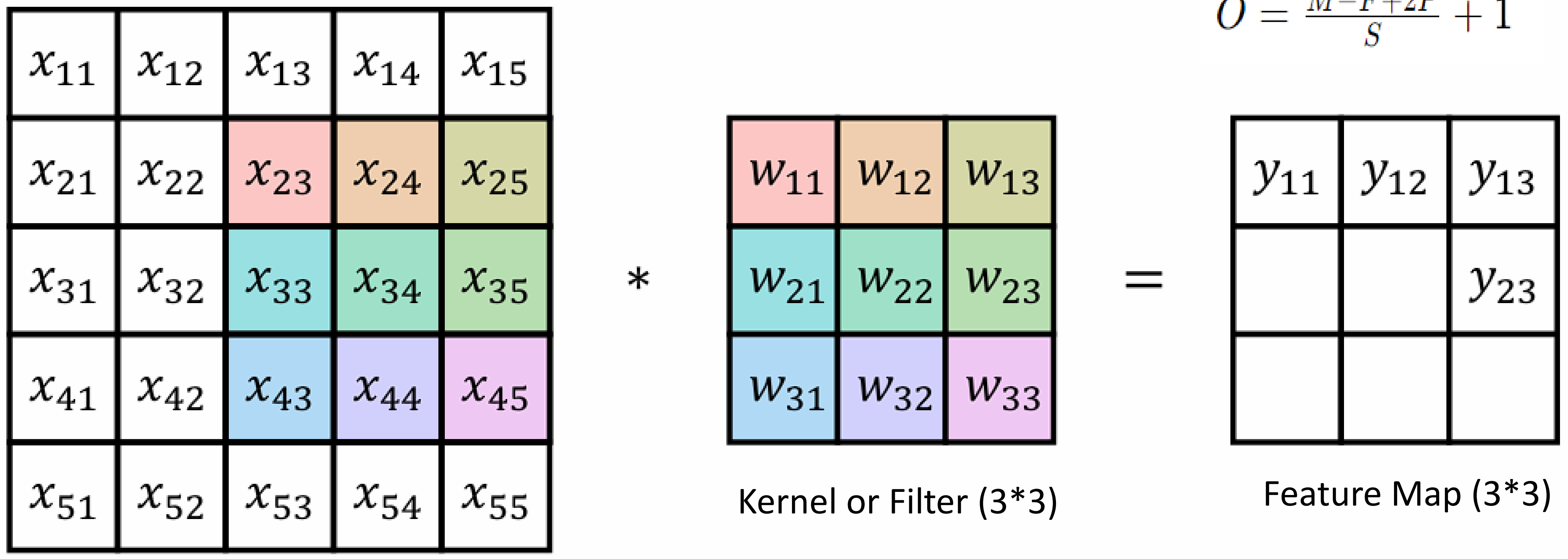
Feature Map (3*3)

Convolutional Neural Network (CNN)

The output computation now only depends on a subset of inputs:

$$y_{23} = w_{11}x_{23} + w_{12}x_{24} + w_{13}x_{25} + w_{21}x_{33} + w_{22}x_{34} + w_{23}x_{35} + w_{31}x_{43} + w_{32}x_{44} + w_{33}x_{45}$$

$$O = \frac{M-F+2P}{S} + 1$$



x_{11} x_{12} x_{13} x_{14} x_{15}
x_{21} x_{22} x_{23} x_{24} x_{25}
x_{31} x_{32} x_{33} x_{34} x_{35}
x_{41} x_{42} x_{43} x_{44} x_{45}
x_{51} x_{52} x_{53} x_{54} x_{55}
*
w_{11} w_{12} w_{13}
w_{21} w_{22} w_{23}
w_{31} w_{32} w_{33}
=
y_{11} y_{12} y_{13}
y_{23}

Input Image (5*5)

Kernel or Filter (3*3)

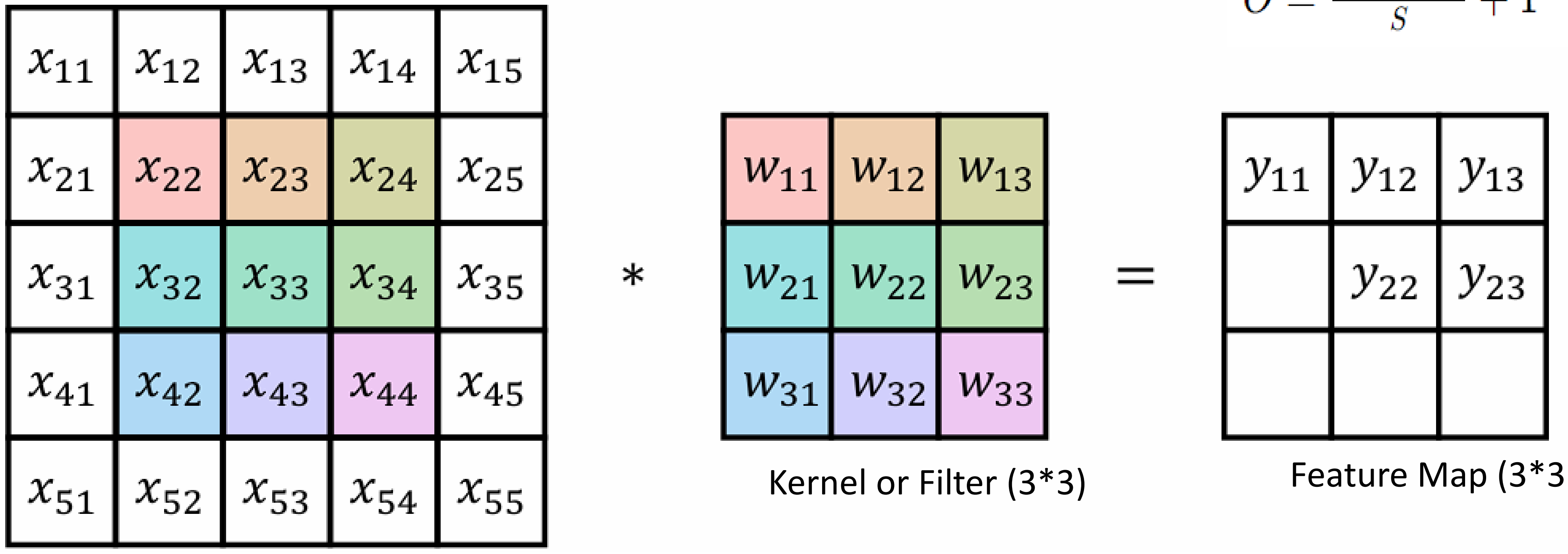
Feature Map (3*3)

Convolutional Neural Network (CNN)

The output computation now only depends on a subset of inputs:

$$y_{22} = w_{11}x_{22} + w_{12}x_{23} + w_{13}x_{24} + w_{21}x_{32} + w_{22}x_{33} + w_{23}x_{34} + w_{31}x_{42} + w_{32}x_{43} + w_{33}x_{44}$$

$$O = \frac{M-F+2P}{S} + 1$$



x_{11} x_{12} x_{13} x_{14} x_{15}
x_{21} x_{22} x_{23} x_{24} x_{25}
x_{31} x_{32} x_{33} x_{34} x_{35}
x_{41} x_{42} x_{43} x_{44} x_{45}
x_{51} x_{52} x_{53} x_{54} x_{55}
*
w_{11} w_{12} w_{13}
w_{21} w_{22} w_{23}
w_{31} w_{32} w_{33}
=
y_{11} y_{12} y_{13}
y_{22} y_{23}

Input Image (5*5)

Kernel or Filter (3*3)

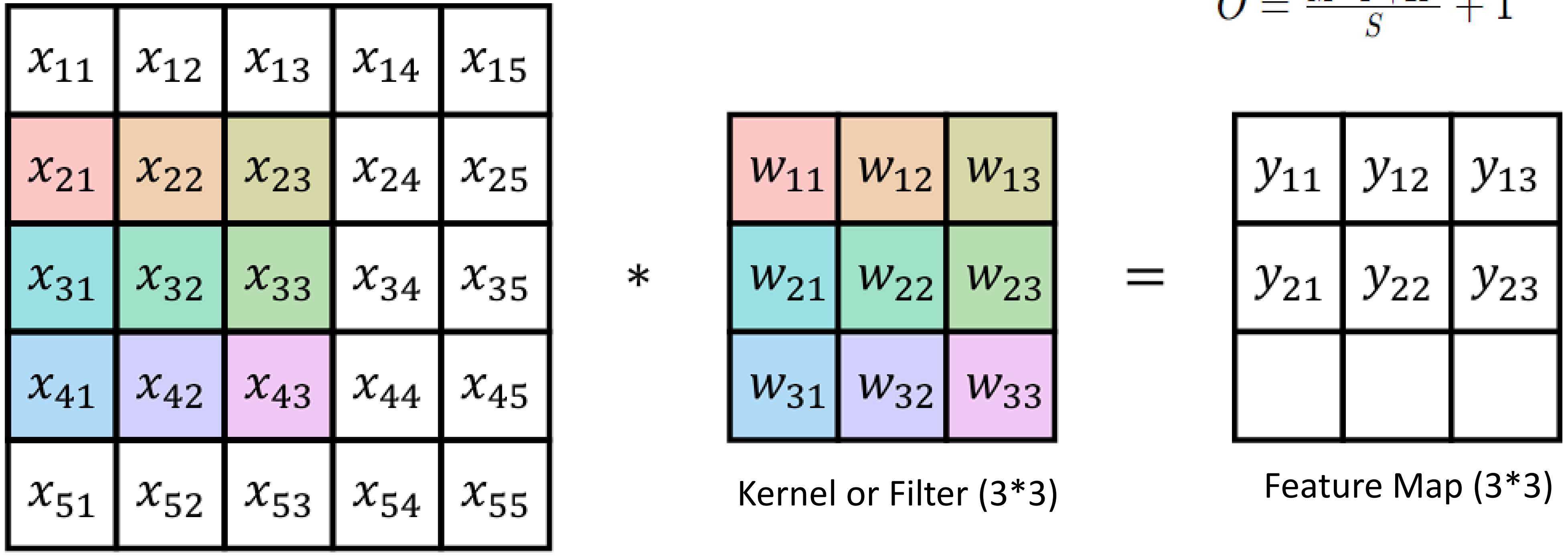
Feature Map (3*3)

Convolutional Neural Network (CNN)

The output computation now only depends on a subset of inputs:

$$y_{21} = w_{11}x_{21} + w_{12}x_{22} + w_{13}x_{23} + w_{21}x_{31} + w_{22}x_{32} + w_{23}x_{33} + w_{31}x_{41} + w_{32}x_{42} + w_{33}x_{43}$$

$$O = \frac{M-F+2P}{S} + 1$$



x_{11} x_{12} x_{13} x_{14} x_{15}
x_{21} x_{22} x_{23} x_{24} x_{25}
x_{31} x_{32} x_{33} x_{34} x_{35}
x_{41} x_{42} x_{43} x_{44} x_{45}
x_{51} x_{52} x_{53} x_{54} x_{55}
*
w_{11} w_{12} w_{13}
w_{21} w_{22} w_{23}
w_{31} w_{32} w_{33}
=
y_{11} y_{12} y_{13}
y_{21} y_{22} y_{23}

Kernel or Filter (3*3)

Feature Map (3*3)

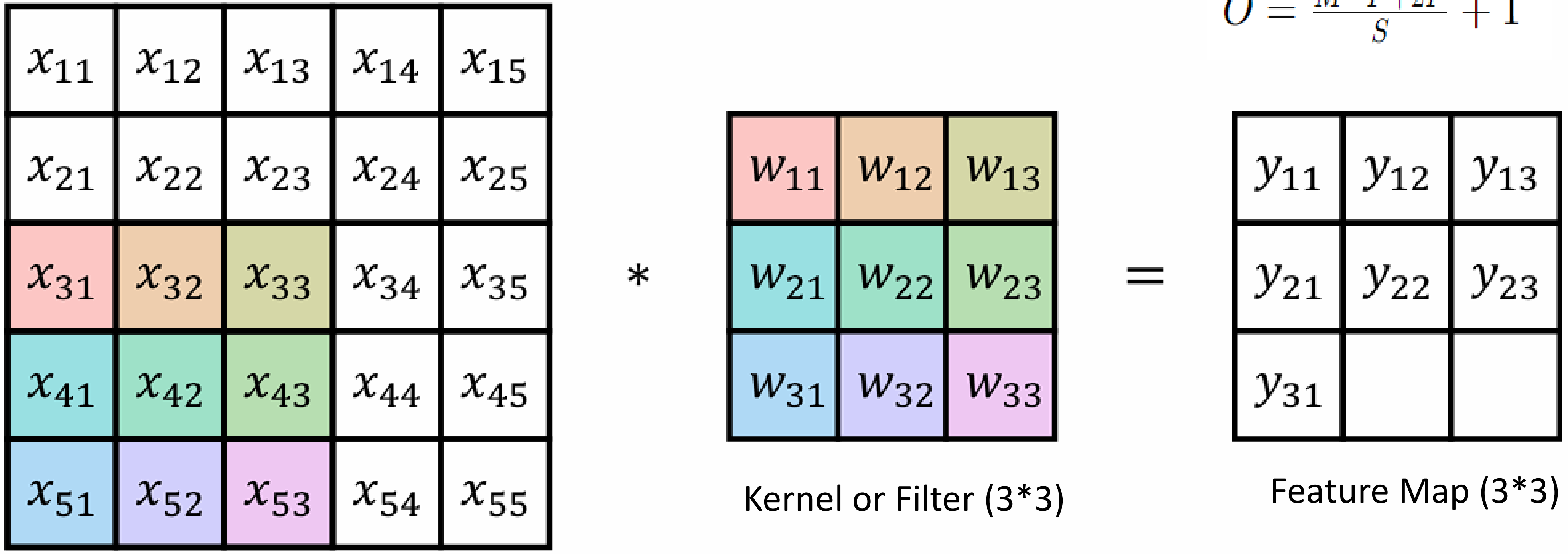
Input Image (5*5)

Convolutional Neural Network (CNN)

The output computation now only depends on a subset of inputs:

$$y_{31} = w_{11}x_{31} + w_{12}x_{32} + w_{13}x_{33} + w_{21}x_{41} + w_{22}x_{42} + w_{23}x_{43} + w_{31}x_{51} + w_{32}x_{52} + w_{33}x_{53}$$

$$O = \frac{M-F+2P}{S} + 1$$


x_{11} x_{12} x_{13} x_{14} x_{15}
x_{21} x_{22} x_{23} x_{24} x_{25}
x_{31} x_{32} x_{33} x_{34} x_{35}
x_{41} x_{42} x_{43} x_{44} x_{45}
x_{51} x_{52} x_{53} x_{54} x_{55}
*
w_{11} w_{12} w_{13}
w_{21} w_{22} w_{23}
w_{31} w_{32} w_{33}
=
y_{11} y_{12} y_{13}
y_{21} y_{22} y_{23}
y_{31}
Kernel or Filter (3*3)
Feature Map (3*3)

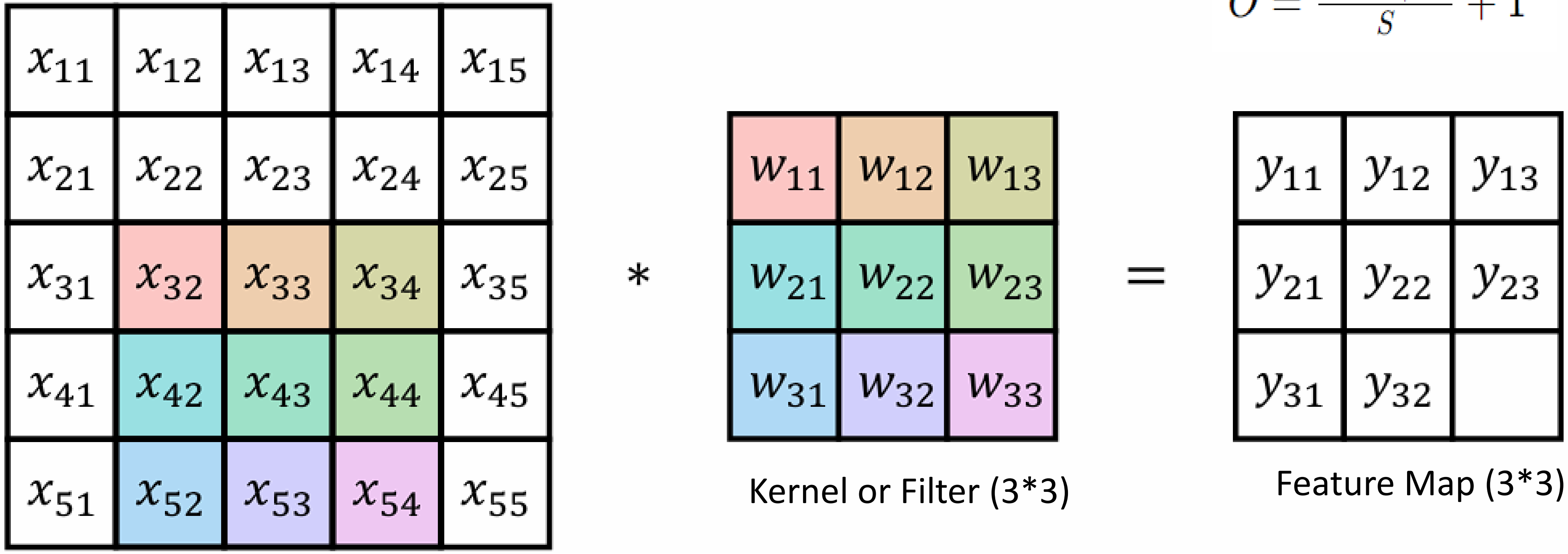
Input Image (5*5)

Convolutional Neural Network (CNN)

The output computation now only depends on a subset of inputs:

$$y_{32} = w_{11}x_{32} + w_{12}x_{33} + w_{13}x_{34} + w_{21}x_{42} + w_{22}x_{43} + w_{23}x_{44} + w_{31}x_{52} + w_{32}x_{53} + w_{33}x_{54}$$

$$O = \frac{M-F+2P}{S} + 1$$


x_{11} x_{12} x_{13} x_{14} x_{15}
x_{21} x_{22} x_{23} x_{24} x_{25}
x_{31} x_{32} x_{33} x_{34} x_{35}
x_{41} x_{42} x_{43} x_{44} x_{45}
x_{51} x_{52} x_{53} x_{54} x_{55}
$*$
w_{11} w_{12} w_{13}
w_{21} w_{22} w_{23}
w_{31} w_{32} w_{33}
$=$
y_{11} y_{12} y_{13}
y_{21} y_{22} y_{23}
y_{31} y_{32}

Input Image (5*5)

Kernel or Filter (3*3)

Feature Map (3*3)

Convolutional Neural Network (CNN)

The output computation now only depends on a subset of inputs:

$$y_{33} = w_{11}x_{33} + w_{12}x_{34} + w_{13}x_{35} + w_{21}x_{43} + w_{22}x_{44} + w_{23}x_{45} + w_{31}x_{53} + w_{32}x_{54} + w_{33}x_{55}$$

$$O = \frac{M-F+2P}{S} + 1$$

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \\ x_{31} & x_{32} & x_{33} & x_{34} & x_{35} \\ x_{41} & x_{42} & x_{43} & x_{44} & x_{45} \\ x_{51} & x_{52} & x_{53} & x_{54} & x_{55} \end{bmatrix} * \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} = \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ y_{31} & y_{32} & y_{33} \end{bmatrix}$$

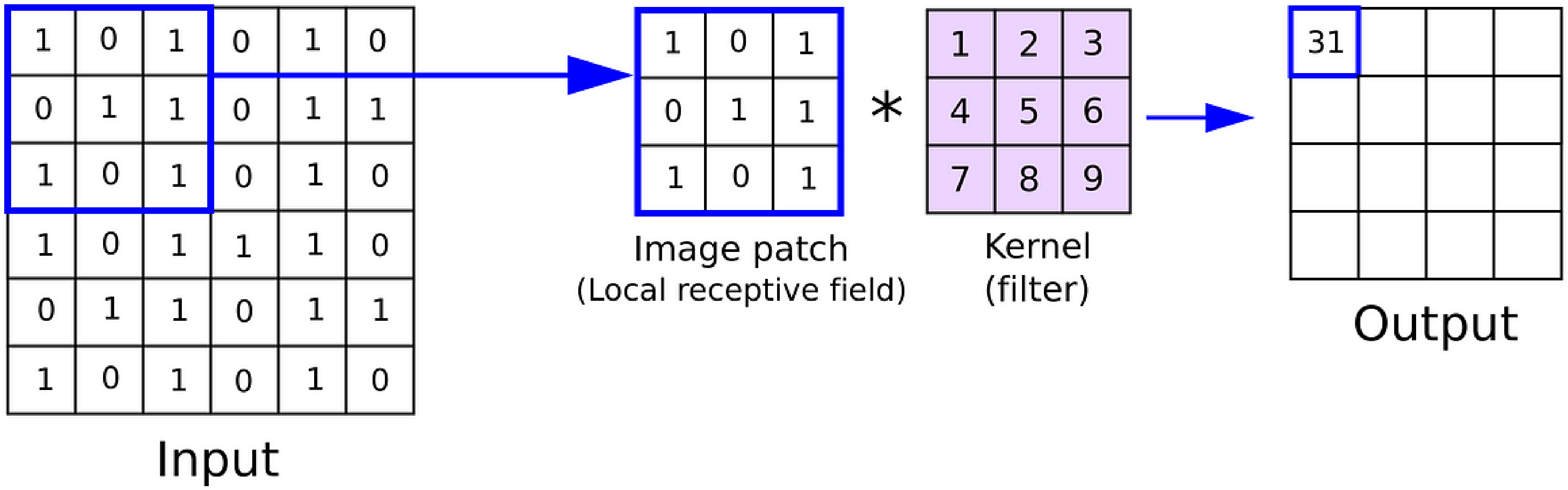
Input Image (5*5)

Kernel or Filter (3*3)

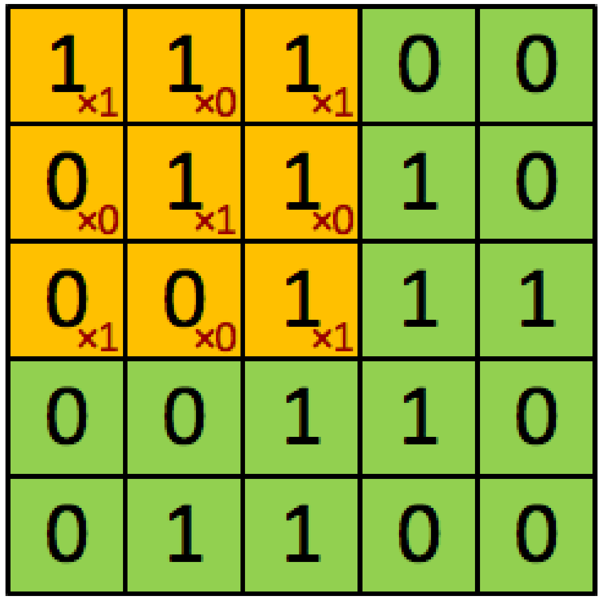
Feature Map (3*3)

Convolutional Neural Network (CNN)

$$O = \frac{M - F + 2P}{S} + 1$$

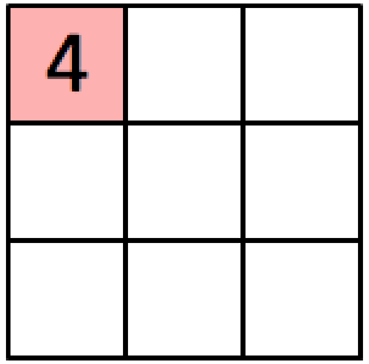


Convolutional Neural Network (CNN)



Image

$$O = \frac{M-F+2P}{S} + 1$$



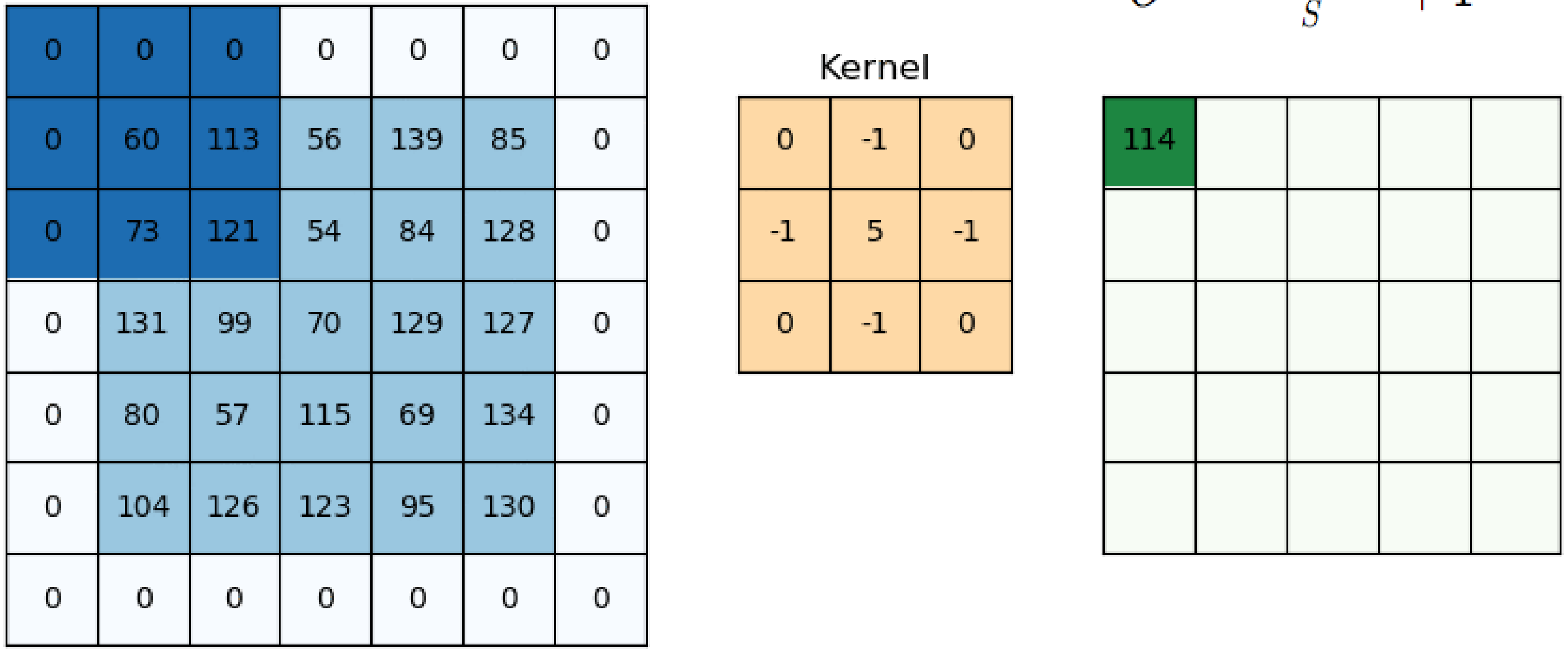
Convolved Feature

1	0	1
0	1	0
1	0	1

Filter (3*3)

Convolutional Neural Network (CNN)

$$O = \frac{M - F + 2P}{S} + 1$$



0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

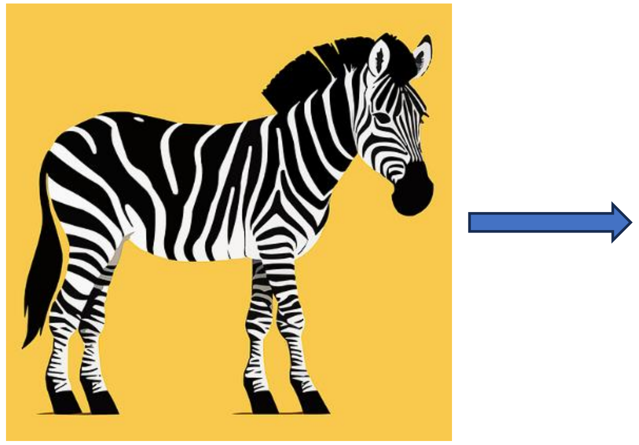
Kernel

0	-1	0
-1	5	-1
0	-1	0

114				

Fig: Zero-Padding in CNN

Convolutional Neural Network (CNN)

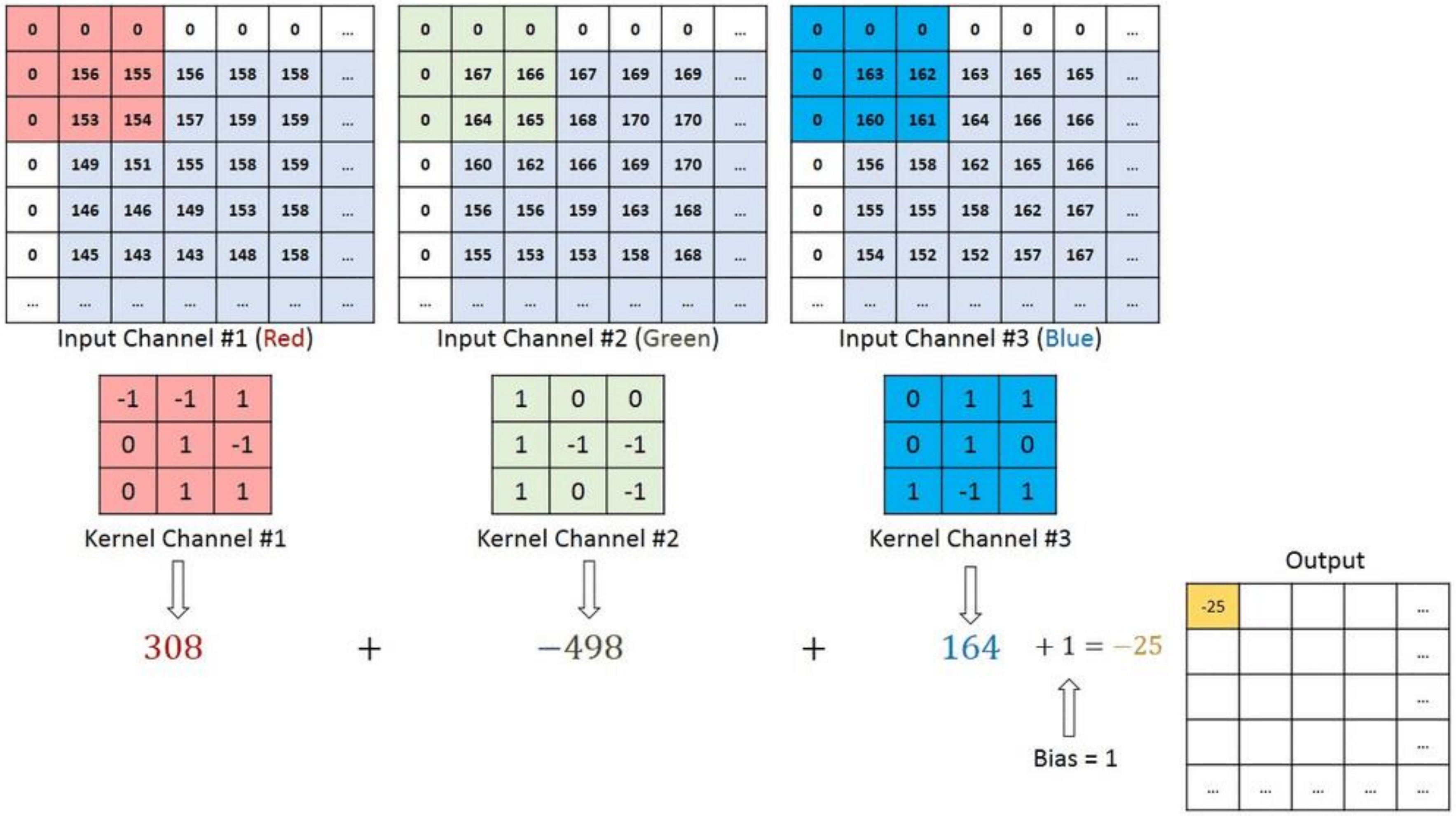


$$O = \frac{M-F+2P}{S} + 1$$

$$\begin{array}{ccccccc}
\text{Input} & & \text{Kernel} & & \text{Bias} & & \text{Output} \\
\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} & \star & \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} & + & \boxed{5} & = & \begin{bmatrix} 11 & 12 \\ 15 & 16 \end{bmatrix}
\end{array}$$

$\star$ Cross-Correlation

Convolutional Neural Network (CNN)



Input Channel #1 (Red)

0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Kernel Channel #1

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

Kernel Channel #2

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

Kernel Channel #3

0	1	1
0	1	0
1	-1	1

$$308 + (-498) + 164 + 1 = -25$$

Output

-25				...
				...
				...
				...
...	...	...	...	...

Convolutional Neural Network (CNN)

- **Prepare Input:** The input matrix includes pixel values or feature values arranged in a grid.
- **Position the Kernel:** The kernel (or filter) is a smaller matrix that you slide over the input matrix. Start at the **top-left** corner of the input and move the kernel over every valid position.
- **Perform Element-wise Multiplication:** For each position of the kernel, multiply each element of the kernel with the corresponding element of the input matrix that it covers.
- **Sum the Results:** After multiplying, sum up all the products obtained in the previous step to get a single output.
- **Adding Bias:** Add a bias term to the scalar value from the previous step. The bias is a single value that is learned during the training of the network. Adding a bias can help the model learn better by adjusting the output.
- **Generate Output Matrix:** The scalar result from each position of the kernel (after adding the bias) forms one element of the output matrix (or feature map).
- **Repeat for Multiple Filters:** If the CNN uses multiple filters, repeat steps 2-7 for each filter to produce multiple output matrices. Each filter can detect different features in the input, such as edges, textures, or other patterns.

Convolutional Neural Network (CNN)

Calculating Output Dimensions:

To find the dimensions of the output feature map (width and height), you can use the following formulas:

1. **Output Height (OH):**

$$OH = \left\lfloor \frac{H + 2P - F}{S} + 1 \right\rfloor$$

2. **Output Width (OW):**

$$OW = \left\lfloor \frac{W + 2P - F}{S} + 1 \right\rfloor$$

3. **Output Depth (OD):**
   - The output depth is equal to the number of filters K used in the convolutional layer.

Here,

W: The width of the input volume.

H: The height of the input volume.

D: The depth of the input volume (number of input channels).

K: The number of filters.

F: The spatial size (height and width) of the filter/kernel.

P: The amount of padding applied to the width and height of the input.

S: The stride, which is the step size the filter moves across the input.

Convolutional Neural Network (CNN)

Suppose you have an input volume of size 32×32×3 (width x height x depth), and you apply a convolutional layer with the following parameters:

- Filter size: 5×5
- Padding: 1 (applied to all sides)
- Stride: 1
- Number of filters: 10

Using the formulas, you would calculate the output dimensions as follows:

- **Output Height:**

$$OH = \left\lfloor \frac{32 + 2 \times 1 - 5}{1} + 1 \right\rfloor = 29$$

Convolutional Neural Network (CNN)

- **Output Height:**

$$OH = \left\lfloor \frac{32 + 2 \times 1 - 5}{1} + 1 \right\rfloor = 29$$

- **Output Width:**

$$OW = \left\lfloor \frac{32 + 2 \times 1 - 5}{1} + 1 \right\rfloor = 29$$

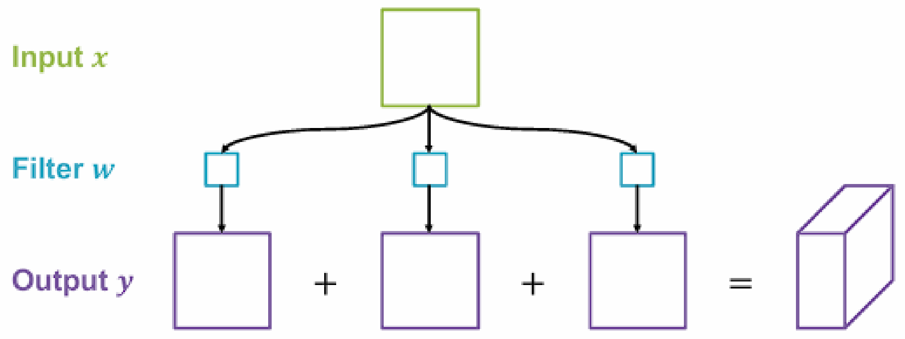
- **Output Depth:** 10 (since there are 10 filters)

So, the output feature map would have dimensions 29×29×10.

Convolutional Neural Network (CNN)

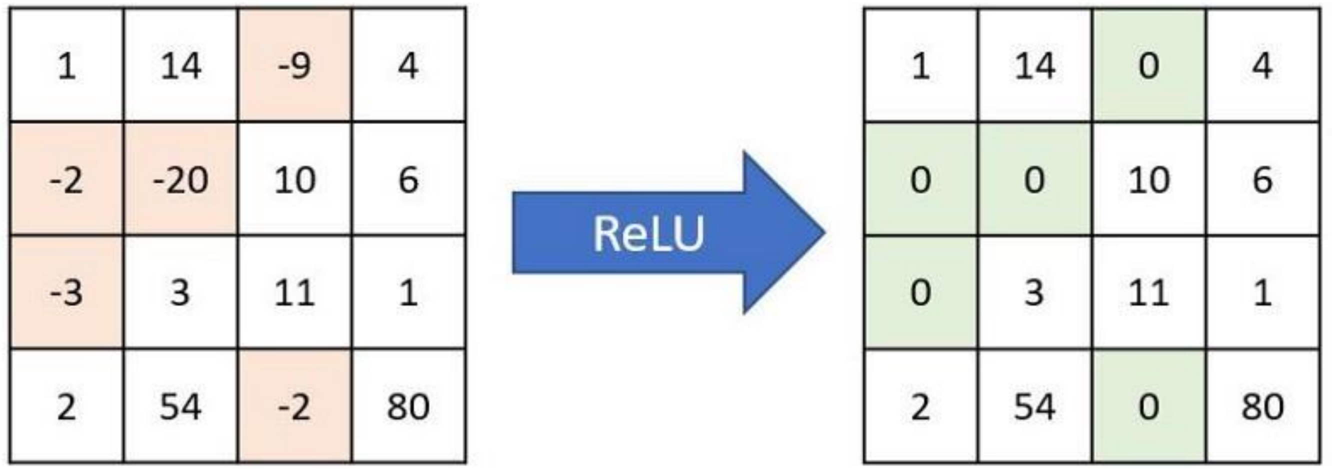
In a convolutional layer, multiple filters can be used in parallel, which result in multiple convolutional operations in parallel on the same input.

The different output are concatenated to create a multi-channel feature map.



Convolutional layer

Convolutional Neural Network (CNN)



1	14	-9	4
-2	-20	10	6
-3	3	11	1
2	54	-2	80

1	14	0	4
0	0	10	6
0	3	11	1
2	54	0	80

Fig: Feature Map (**Before** ReLU)

Fig: Feature Map (**After** ReLU)

Pooling Layer

Convolutional Neural Network (CNN)

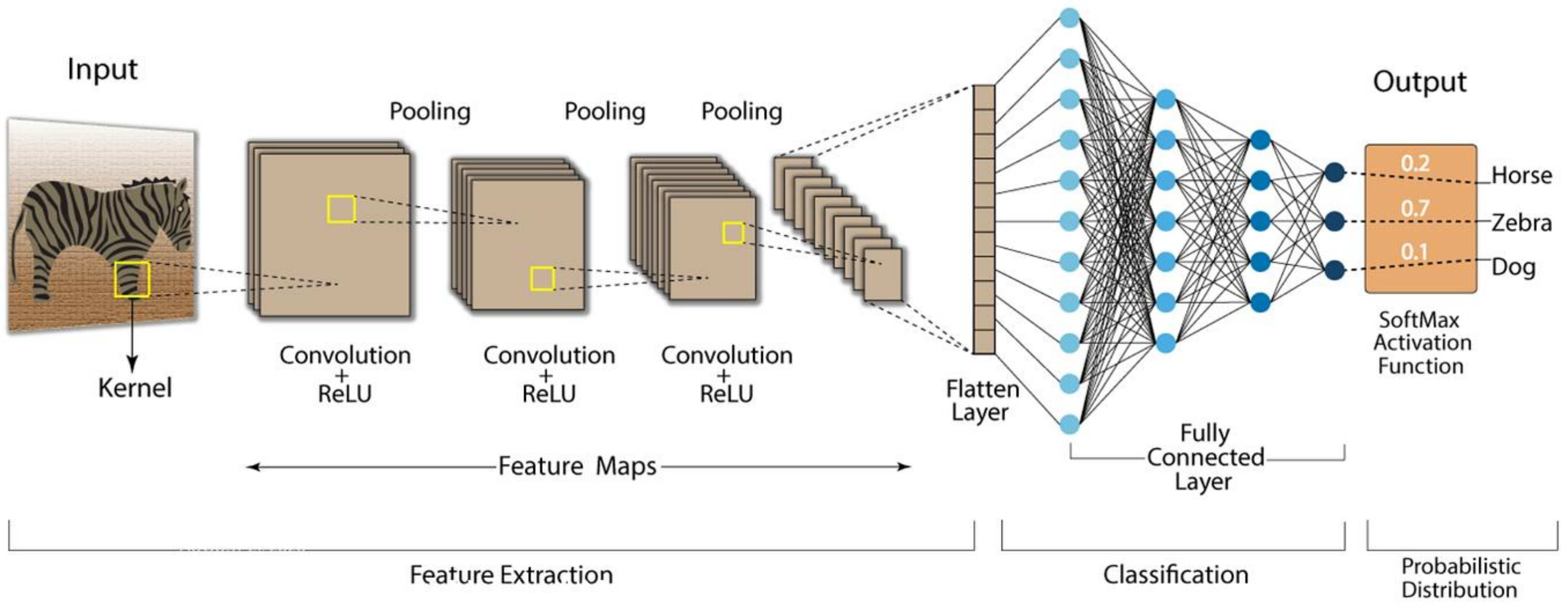


Fig: CNN Architecture

CNN: Pooling Layer

In a convolutional neural network (CNN), the pooling layer plays a crucial role in **reducing** the spatial dimensions (width and height) of the input volume for the subsequent layers.

Pooling layers in convolutional neural networks (CNNs) are important for three main reasons:

- ❖ **Efficiency:** They reduce the spatial dimensions of feature maps, lowering computational costs and the number of parameters.
- ❖ **Feature Invariance:** Pooling provides robustness to minor variations in the input, helping the network to recognize features regardless of their position.
- ❖ **Generalization:** By simplifying the features, pooling helps prevent overfitting, enhancing the model's ability to perform well on new, unseen data.

CNN: Pooling Layer

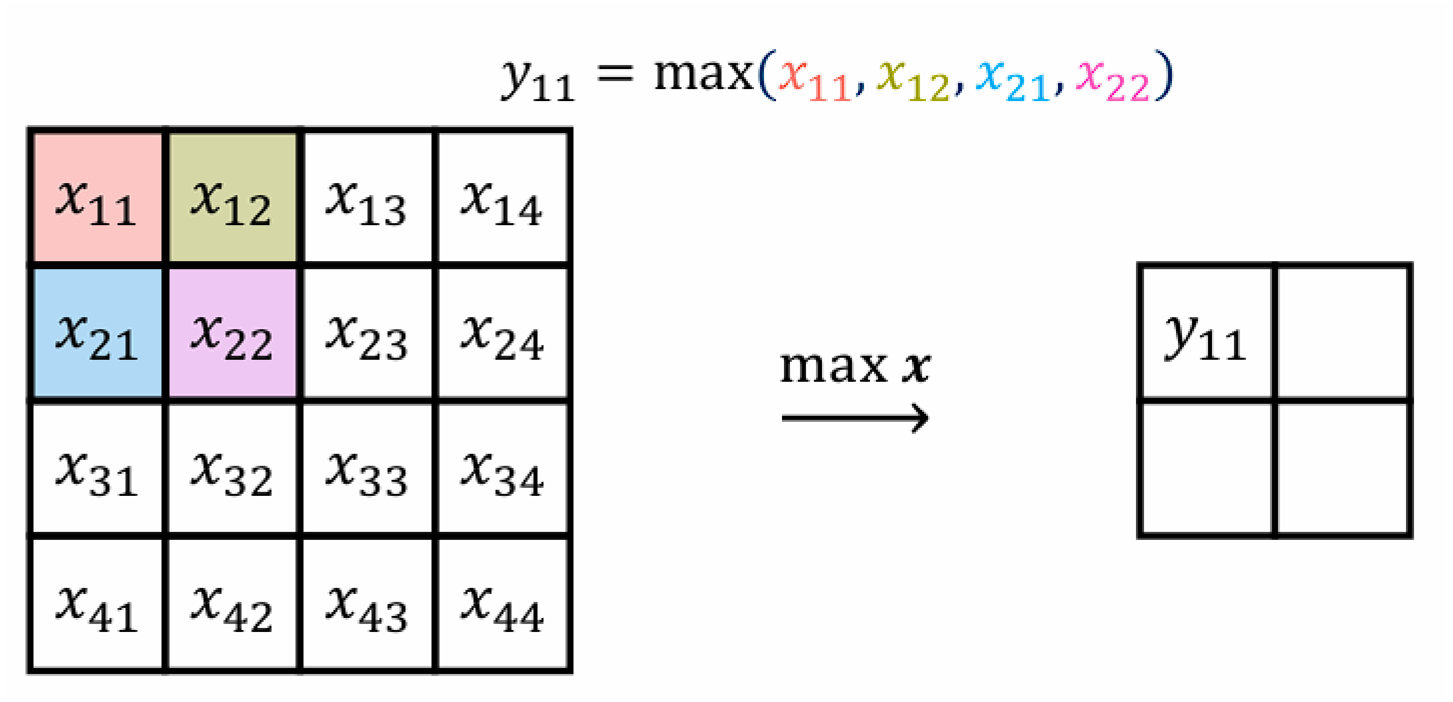
In a convolutional neural network (CNN), the pooling layer plays a crucial role in **reducing** the spatial dimensions (width and height) of the input volume for the subsequent layers.

The most common types of pooling are:

- **Max Pooling:** Outputs the maximum value from each patch of the feature map.
- **Average Pooling:** Outputs the average value from each patch.

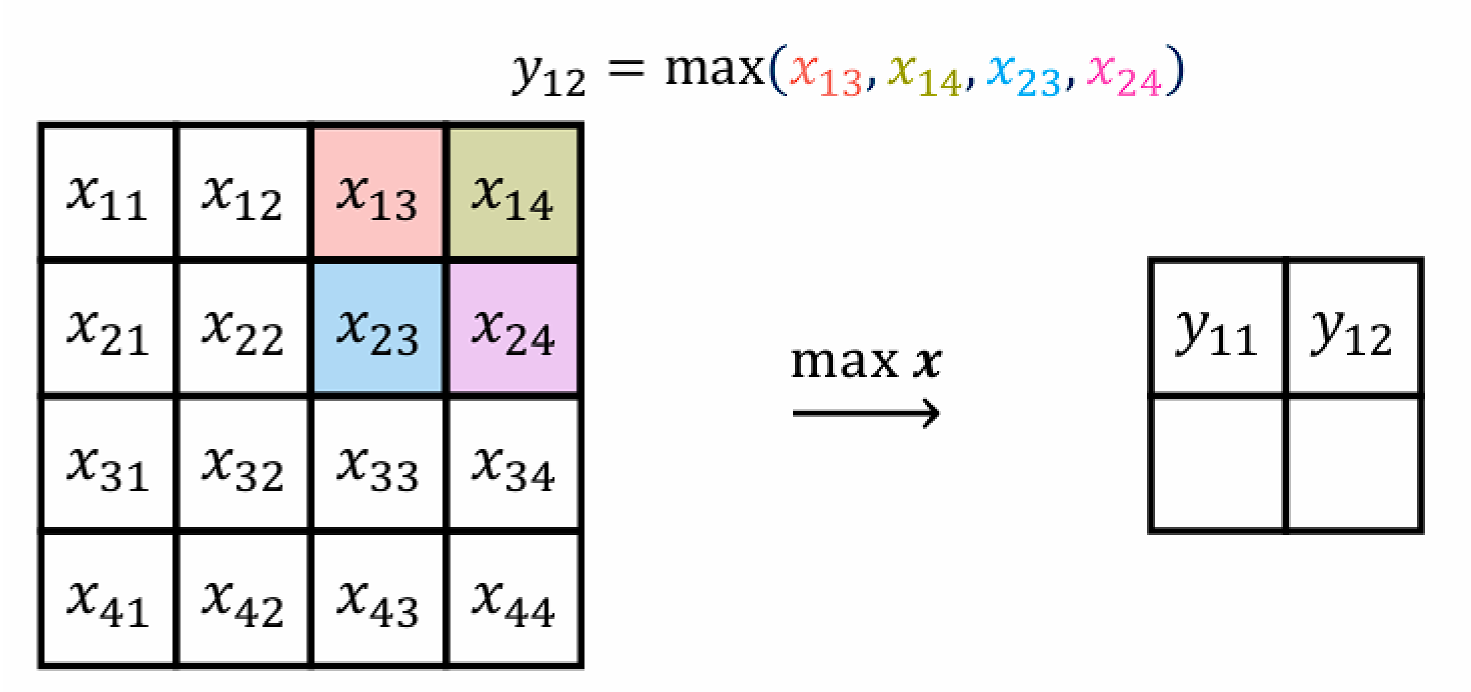
CNN: Pooling Layer

The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.


$y_{11} = \max(x_{11}, x_{12}, x_{21}, x_{22})$
x_{11} x_{12} x_{13} x_{14}
x_{21} x_{22} x_{23} x_{24}
x_{31} x_{32} x_{33} x_{34}
x_{41} x_{42} x_{43} x_{44}
$\xrightarrow{\max \boldsymbol{x}}$
y_{11}

CNN: Pooling Layer

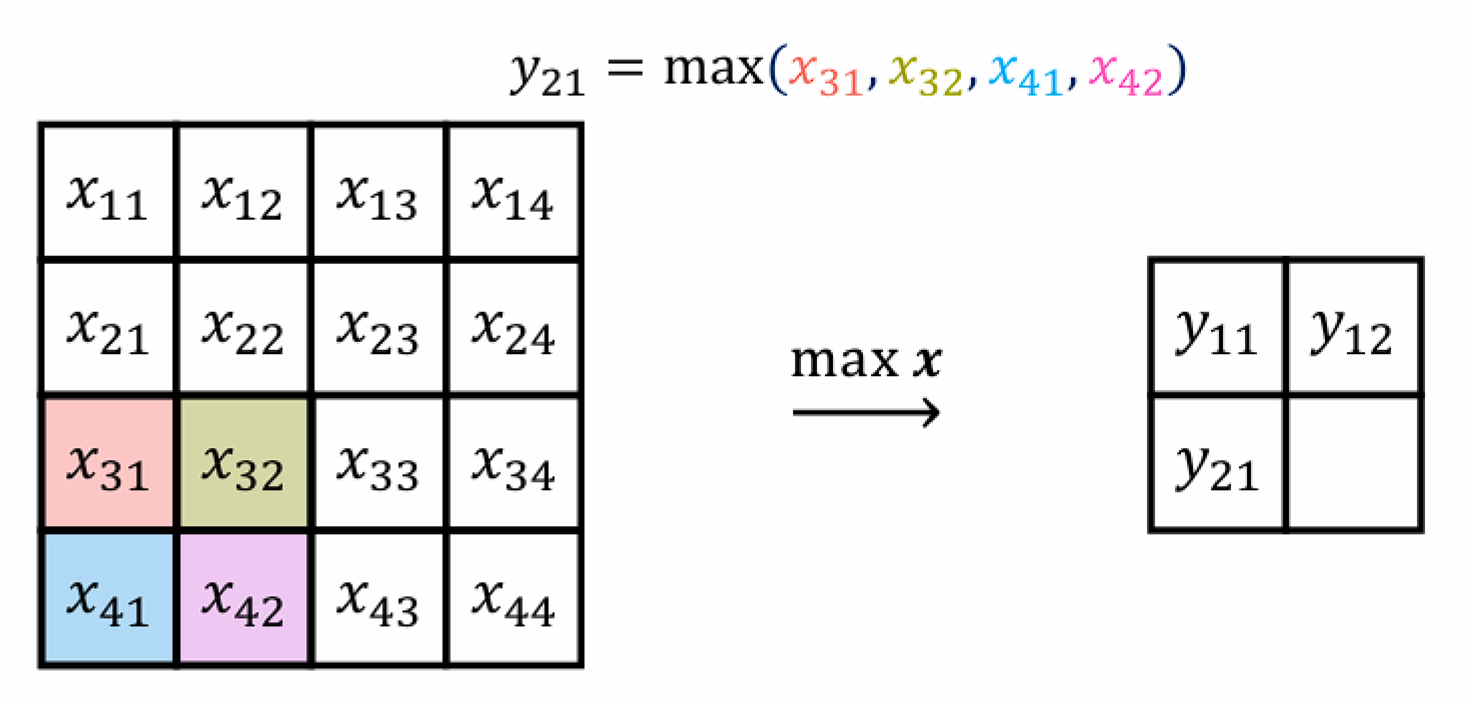
The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.



$y_{12} = \max(x_{13}, x_{14}, x_{23}, x_{24})$
x_{11} x_{12} x_{13} x_{14}
x_{21} x_{22} x_{23} x_{24}
x_{31} x_{32} x_{33} x_{34}
x_{41} x_{42} x_{43} x_{44}
$\max \boldsymbol{x}$
y_{11} y_{12}

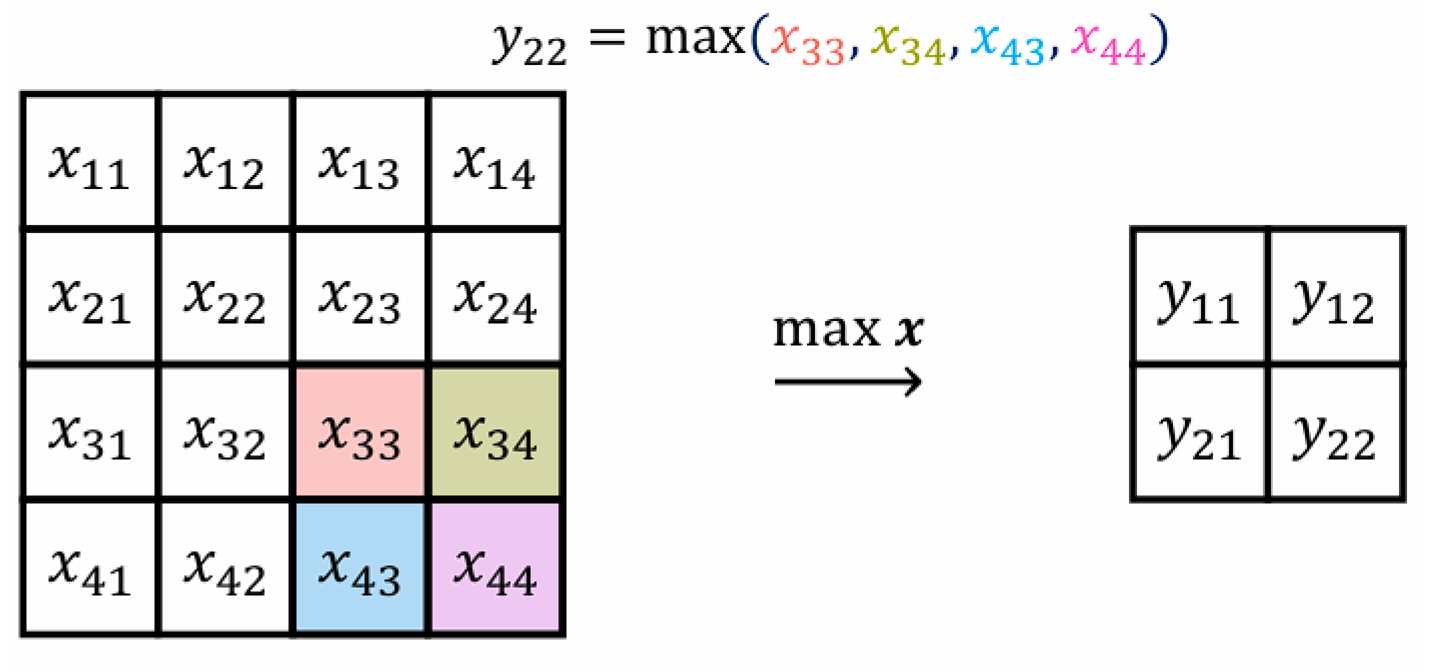
CNN: Pooling Layer

The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.


$y_{21} = \max(x_{31}, x_{32}, x_{41}, x_{42})$
x_{11} x_{12} x_{13} x_{14}
x_{21} x_{22} x_{23} x_{24}
x_{31} x_{32} x_{33} x_{34}
x_{41} x_{42} x_{43} x_{44}
$\xrightarrow{\max \boldsymbol{x}}$
y_{11} y_{12}
y_{21}

CNN: Pooling Layer

The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.



$y_{22} = \max(x_{33}, x_{34}, x_{43}, x_{44})$
x_{11} x_{12} x_{13} x_{14}
x_{21} x_{22} x_{23} x_{24}
x_{31} x_{32} x_{33} x_{34}
x_{41} x_{42} x_{43} x_{44}
$\max \boldsymbol{x}$
y_{11} y_{12}
y_{21} y_{22}

CNN: Pooling Layer

The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.

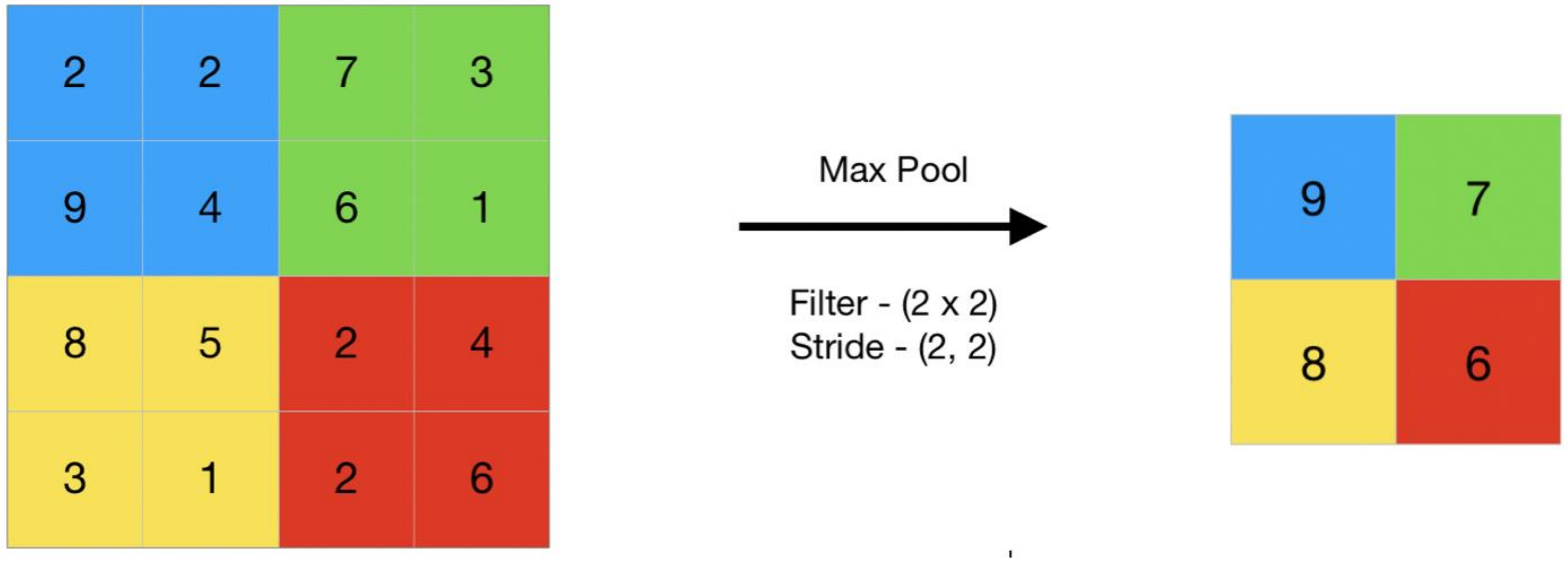


Fig: Max Pooling in CNN

CNN: Pooling Layer

The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.

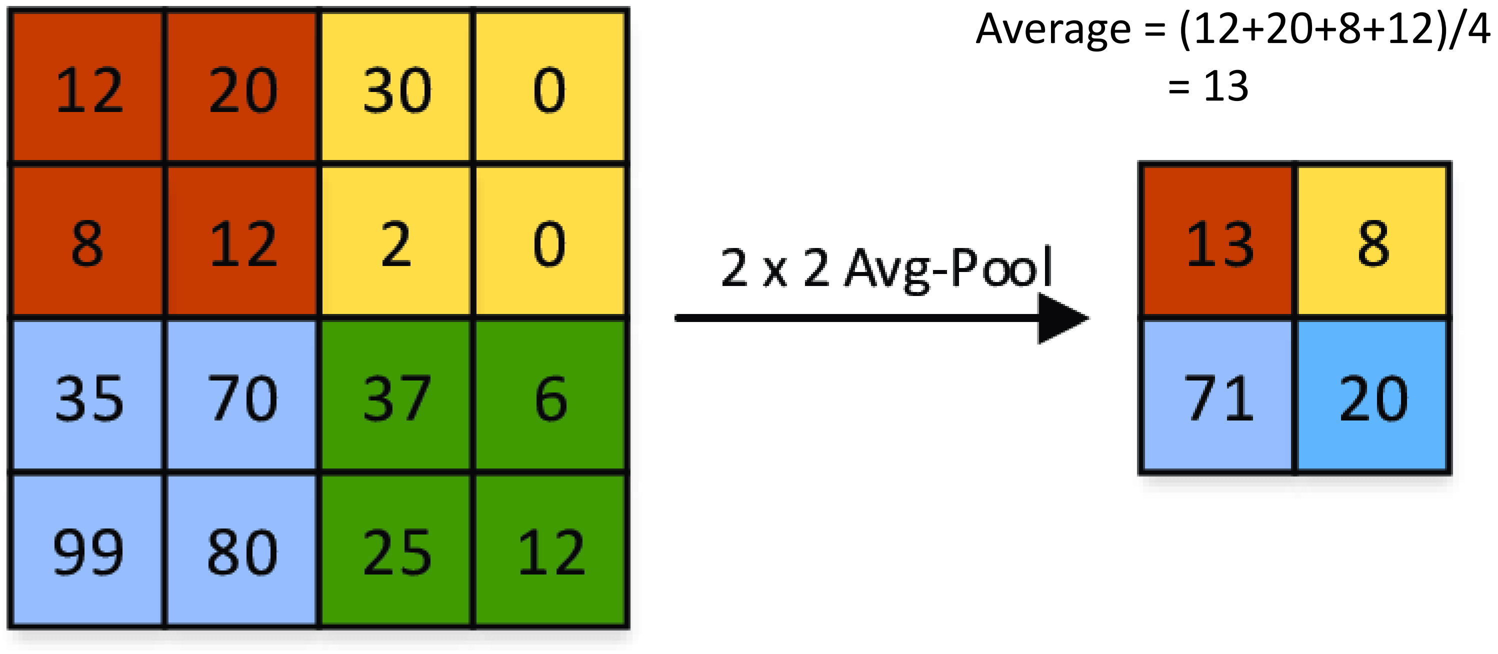


Fig: Average Pooling in CNN

Convolutional Neural Network (CNN)

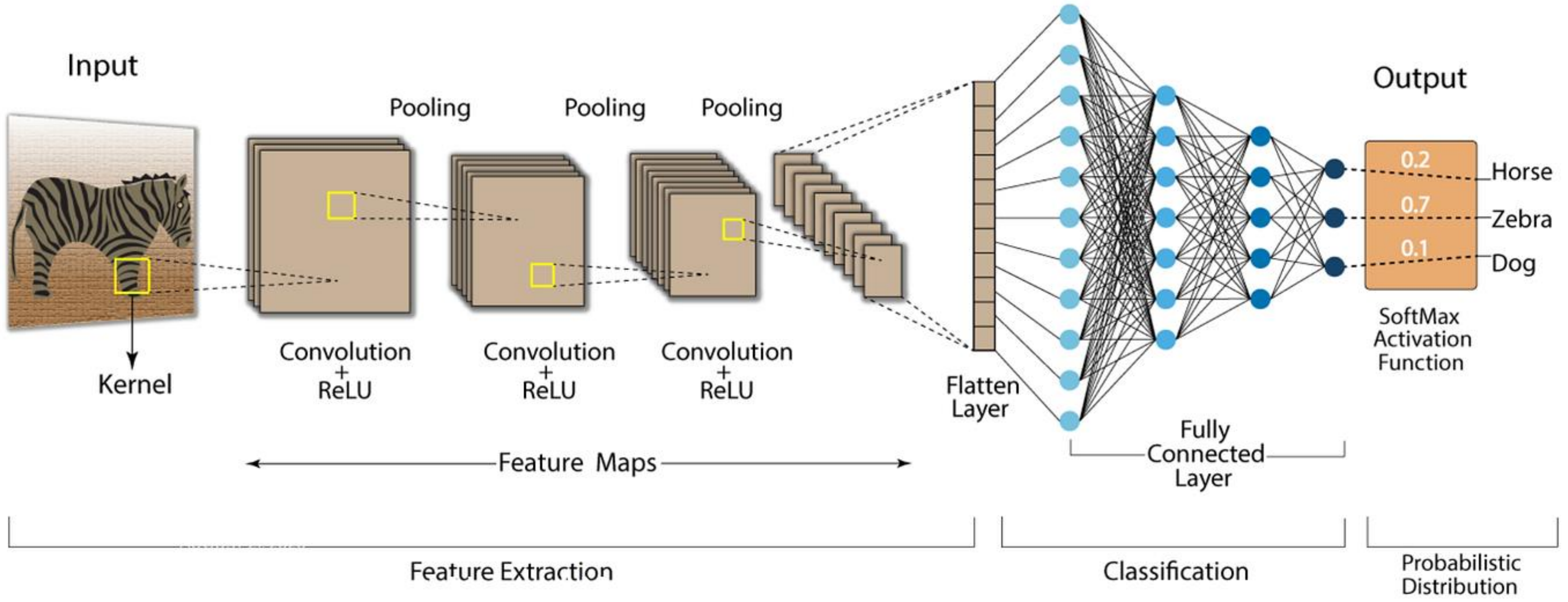
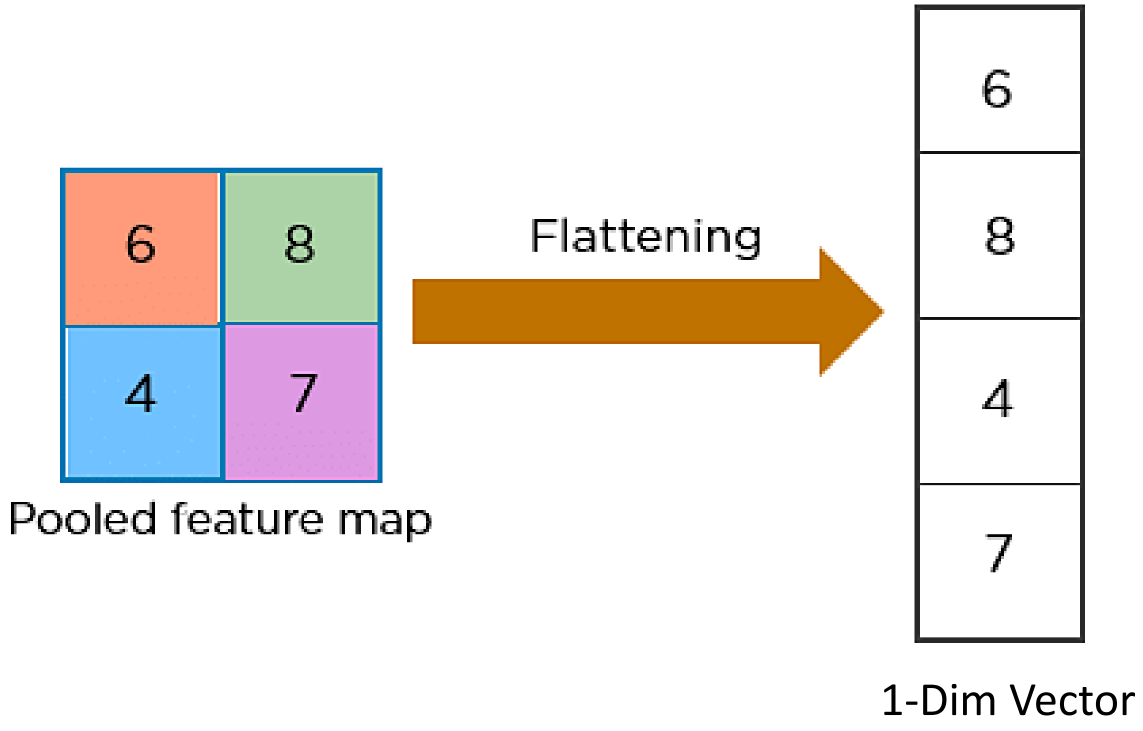


Fig: CNN Architecture

CNN: Flatten Layer

The flatten layer typically appears after the convolutional and pooling layers in convolutional neural network (CNN) architectures. The main **responsibility** of the flatten layer in a convolutional neural network (CNN) is to **reshape the multi-dimensional output** from the preceding convolutional or pooling layers into a **single, one-dimensional vector**.



CNN: Dense Layer

CNN: Flatten Layer

Question: If a convolutional neural network has a flatten layer that receives a **10x10 dimensional feature map**, resulting in **100** flattened values, how are these values processed in the subsequent fully connected layer? Specifically, does each neuron in the next layer receive all 100 values, or does each value get assigned to a different neuron?

CNN: Flatten Layer

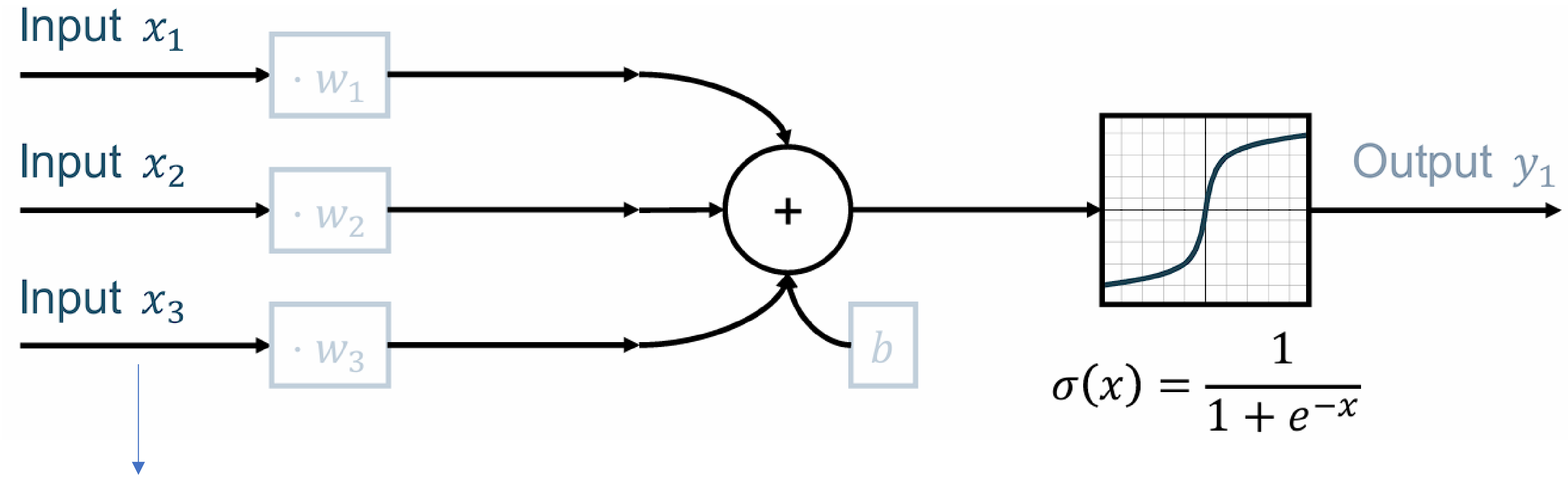
Question: If a convolutional neural network has a flatten layer that receives a **10x10 dimensional feature map**, resulting in 100 flattened values, how are these values processed in the subsequent fully connected layer? Specifically, does each neuron in the next layer receive all 100 values, or does each value get assigned to a different neuron?

Answer: In a fully connected layer that follows a flatten layer in a convolutional neural network, **each neuron in the fully connected layer receives all 100 values** from the flattened 10x10 dimensional feature map. The fully connected layer operates such that each neuron is connected to every single input from the previous layer (in this case, the 100 flattened values).

Deep Learning & AI

Artificial Neural Networks (ANNs)

$$y_1 = \sigma(w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + b) = \sigma(\sum_i w_i \cdot x_i + b)$$



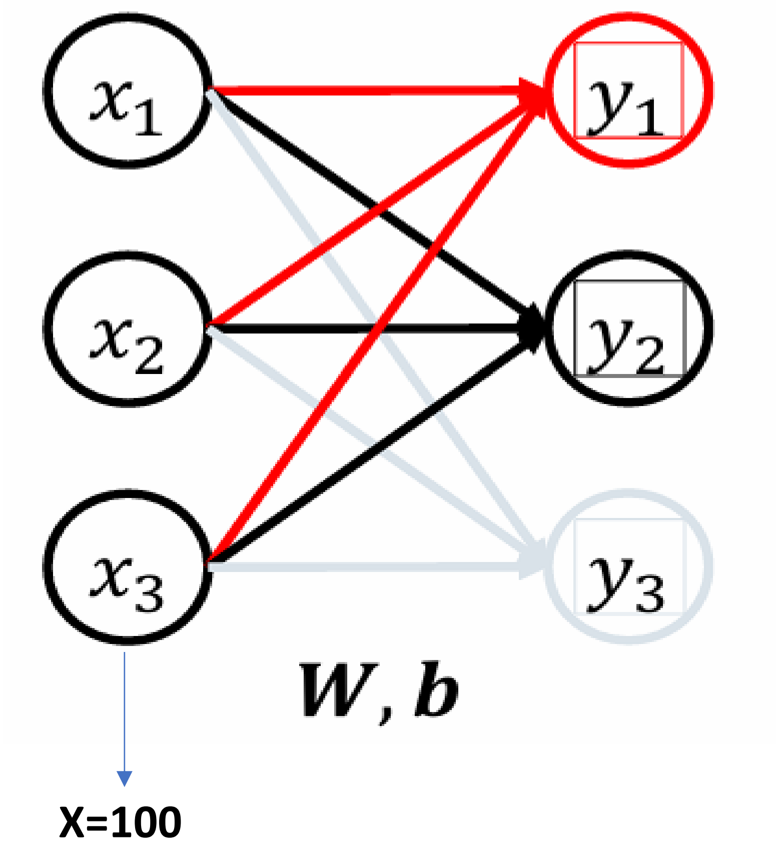
X=100

Fig: Artificial Neuron (Single Layer Perceptron)

Deep Learning & AI

Artificial Neural Networks (ANNs)

We can combine multiple perceptrons to create a layer.



We can thus rewrite the three computations as:

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

Or in a more simplified form:

$$y = \sigma(W \cdot x + b)$$

Convolutional Neural Network (CNN)

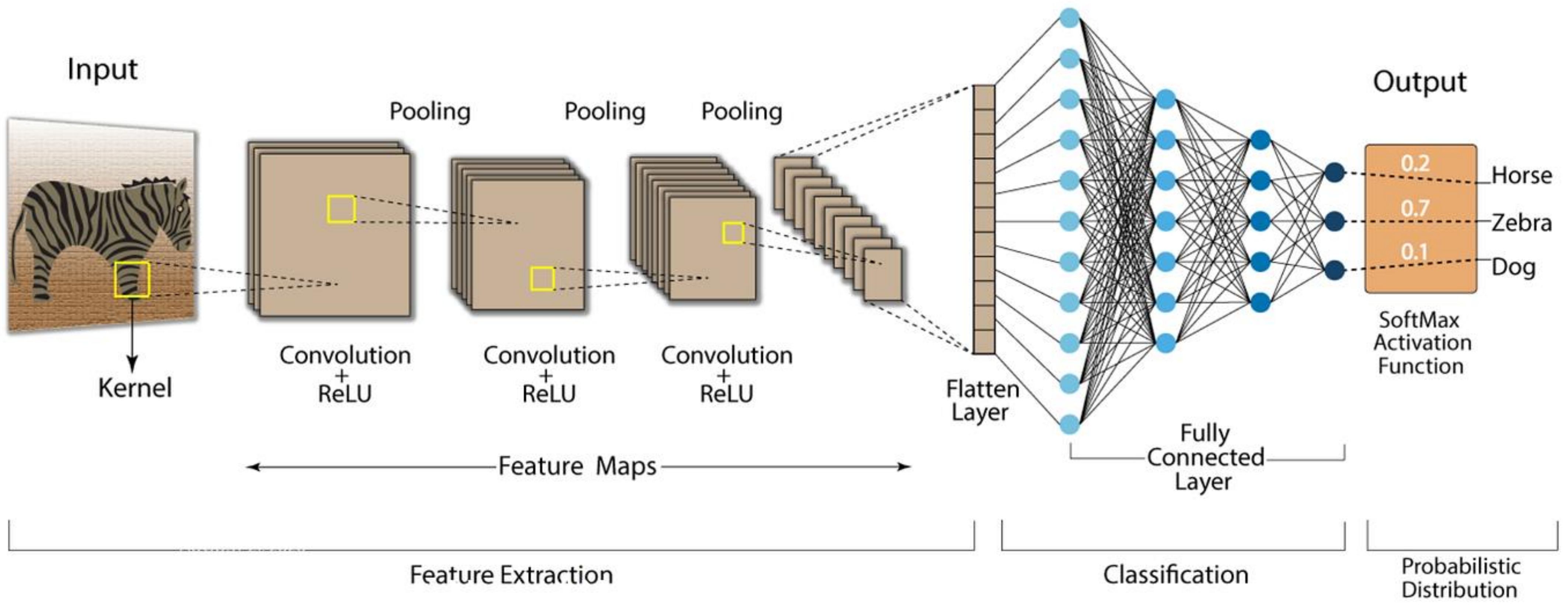


Fig: CNN Architecture

Convolutional Neural Network (CNN)

Output Dimensions | Parameters

CNN: Dimensions

Output dimensions describe the **size of the data tensor that results from a layer** within a CNN. This typically includes:

- **Width** and **Height**: These dimensions can change based on the type of layer (convolutional, pooling), the kernel size, the stride, and the padding used.
- **Depth** (or Channels): This dimension often changes in convolutional layers depending on the number of filters used. It stays the same through pooling layers unless pooling is done separately across channels.

CNN: Dimensions

Question: Consider the input tensor of dimensions 10×10×10 where the last dimension is the channel dimension. The following operations are applied:

1. 3×3 convolution (40 channels) with stride 1 and padding 1 for each dimension.
2. ReLU activation.
3. 3×3 max pooling with stride 1 and padding 1 for each dimension.
4. 3×3 convolution (20 channels) with stride 1 and padding 1 for each dimension.
5. ReLU activation.
6. 2×2 max pooling with stride 2 and padding 1 for each dimension.

What are the dimensions of the output tensor after each operation? Report the name of the operation and the output dimensions as width x height x channels. There are 6 operations in total.

CNN: Dimensions

$$\text{Output Dimension} = \left\lfloor \frac{\text{Input Dimension} - \text{Kernel Size} + 2 \times \text{Padding}}{\text{Stride}} \right\rfloor + 1$$

Dimensions After Each Operation:

1. **After 3×3 Convolution:** $10 \times 10 \times 40$
2. **After ReLU Activation:** $10 \times 10 \times 40$
3. **After 3×3 Max Pooling:** $10 \times 10 \times 40$
4. **After 3×3 Convolution:** $10 \times 10 \times 20$
5. **After ReLU Activation:** $10 \times 10 \times 20$
6. **After 2×2 Max Pooling:** $6 \times 6 \times 20$

CNN: Dimensions

Question: Consider the input tensor of dimensions **10×10×10** where the last dimension is the channel dimension. The following operations are applied:

1. 3×3 convolution (40 channels) with stride 1 and padding 1 for each dimension.
2. ReLU activation.
3. 3×3 max pooling with stride 1 and padding 1 for each dimension.
4. 3×3 convolution (20 channels) with stride 1 and padding 1 for each dimension.
5. ReLU activation.
6. 2×2 max pooling with stride 2 and padding 1 for each dimension.

What are the dimensions of the output tensor after each operation? Report the name of the operation and the output dimensions as width x height x channels. There are 6 operations in total.

CNN: Dimensions

1. 3×3 Convolution (40 channels) with stride 1 and padding 1

- **Kernel Size**: 3×3
- **Stride**: 1
- **Padding**: 1
- **Input Dimensions**: $10 \times 10 \times 10$

$$\text{Output Width} = \left\lfloor \frac{10-3+2\times1}{1} \right\rfloor + 1 = 10$$
$$\text{Output Height} = \left\lfloor \frac{10-3+2\times1}{1} \right\rfloor + 1 = 10$$
$$\text{Output Channels} = 40$$

Dimension: 10 x 10 x 40

CNN: Dimensions

Question: Consider the input tensor of dimensions **10×10×10** where the last dimension is the channel dimension. The following operations are applied:

1. 3×3 convolution (40 channels) with stride 1 and padding 1 for each dimension.
2. **ReLU activation.**
3. **3×3 max pooling with stride 1 and padding 1 for each dimension.**
4. 3×3 convolution (20 channels) with stride 1 and padding 1 for each dimension.
5. ReLU activation.
6. 2×2 max pooling with stride 2 and padding 1 for each dimension.

What are the dimensions of the output tensor after each operation? Report the name of the operation and the output dimensions as width x height x channels. There are 6 operations in total.

CNN: Dimensions

2. ReLU Activation

- Does not change dimensions.
- **Output Dimensions:** $10 \times 10 \times 40$

3. 3×3 Max Pooling with stride 1 and padding 1

- **Kernel Size:** 3×3
- **Stride:** 1
- **Padding:** 1

$$\text{Output Width} = \left\lfloor \frac{10-3+2\times 1}{1} \right\rfloor + 1 = 10$$
$$\text{Output Height} = \left\lfloor \frac{10-3+2\times 1}{1} \right\rfloor + 1 = 10$$
$\text{Output Channels} = 40$ (Channel dimension remains unchanged in pooling)

Dimension: 10 x 10 x 40

CNN: Dimensions

Question: Consider the input tensor of dimensions **10×10×10** where the last dimension is the channel dimension. The following operations are applied:

1. 3×3 convolution (40 channels) with stride 1 and padding 1 for each dimension.
2. ReLU activation.
3. 3×3 max pooling with stride 1 and padding 1 for each dimension.
4. **3×3 convolution (20 channels) with stride 1 and padding 1 for each dimension.**
5. ReLU activation.
6. 2×2 max pooling with stride 2 and padding 1 for each dimension.

What are the dimensions of the output tensor after each operation? Report the name of the operation and the output dimensions as width x height x channels. There are 6 operations in total.

CNN: Dimensions

4. 3×3 Convolution (20 channels) with stride 1 and padding 1

- **Kernel Size:** 3×3
- **Stride:** 1
- **Padding:** 1
- **Input Dimensions:** $10 \times 10 \times 40$

$$\text{Output Width} = \left\lfloor \frac{10-3+2\times1}{1} \right\rfloor + 1 = 10$$
$$\text{Output Height} = \left\lfloor \frac{10-3+2\times1}{1} \right\rfloor + 1 = 10$$
$$\text{Output Channels} = 20$$

Dimension: 10 x 10 x 20

CNN: Dimensions

Question: Consider the input tensor of dimensions **10×10×10** where the last dimension is the channel dimension. The following operations are applied:

1. 3×3 convolution (40 channels) with stride 1 and padding 1 for each dimension.
2. ReLU activation.
3. 3×3 max pooling with stride 1 and padding 1 for each dimension.
4. 3×3 convolution (20 channels) with stride 1 and padding 1 for each dimension.
5. **ReLU activation.**
6. **2×2 max pooling with stride 2 and padding 1 for each dimension.**

What are the dimensions of the output tensor after each operation? Report the name of the operation and the output dimensions as width x height x channels. There are 6 operations in total.

CNN: Dimensions

5. ReLU Activation

- Does not change dimensions.
- **Output Dimensions**: $10 \times 10 \times 20$

6. 2×2 Max Pooling with stride 2 and padding 1

- **Kernel Size**: 2×2
- **Stride**: 2
- **Padding**: 1

$$\text{Output Width} = \left\lfloor \frac{10-2+2\times1}{2} \right\rfloor + 1 = 6$$
$$\text{Output Height} = \left\lfloor \frac{10-2+2\times1}{2} \right\rfloor + 1 = 6$$
$\text{Output Channels} = 20$ (Channel dimension remains unchanged in pooling)

Dimension: 6 x 6 x 20

CNN: Counting the Parameters

Parameters reflect the model's learning capacity and complexity. More parameters can mean a more powerful model, but also one that is more prone to overfitting and is computationally more expensive to train and run.

Note: In typical Convolutional Neural Network (CNN) architectures, the **convolutional layers** are primarily responsible for adding parameters, which are the learnable weights and biases of the model.

CNN: Parameter Count

Question: Consider the input tensor of dimensions **10×10×10** where the last dimension is the channel dimension. The following operations are applied:

1. **3×3 convolution (40 channels) with stride 1 and padding 1 for each dimension.**
2. ReLU activation.
3. 3×3 max pooling with stride 1 and padding 1 for each dimension.
4. 3×3 convolution (20 channels) with stride 1 and padding 1 for each dimension.
5. ReLU activation.
6. 2×2 max pooling with stride 2 and padding 1 for each dimension.

What is the total parameter count?

CNN: Parameter Count

$$\text{Number of Parameters} = (\text{Kernel Height} \times \text{Kernel Width} \times \text{Input Channels} + 1) \times \text{Output Channels}$$

First 3x3 Convolution (40 channels):

- **Kernel Height**: 3
- **Kernel Width**: 3
- **Input Channels:** 10 (initial input channels)
- **Output Channels**: 40

$$\text{Parameters} = (3 \times 3 \times 10 + 1) \times 40 = (90 + 1) \times 40 = 3640$$

CNN: Parameter Count

Question: Consider the input tensor of dimensions **10×10×10** where the last dimension is the channel dimension. The following operations are applied:

1. 3×3 convolution (40 channels) with stride 1 and padding 1 for each dimension.
2. ReLU activation.
3. 3×3 max pooling with stride 1 and padding 1 for each dimension.
4. **3×3 convolution (20 channels) with stride 1 and padding 1 for each dimension.**
5. ReLU activation.
6. 2×2 max pooling with stride 2 and padding 1 for each dimension.

What is the total parameter count?

CNN: Parameter Count

$$\text{Number of Parameters} = (\text{Kernel Height} \times \text{Kernel Width} \times \text{Input Channels} + 1) \times \text{Output Channels}$$

Second 3x3 Convolution (20 channels):

- The input to this layer is the output of the first max pooling layer, which maintains 40 channels.
- **Output Channels**: 20

$$\text{Parameters} = (3 \times 3 \times 40 + 1) \times 20 = (360 + 1) \times 20 = 7220$$

CNN: Parameter Count

Total Parameters

Now, to find the total number of parameters in the CNN:

$$\text{Total Parameters} = 3640(\text{First Conv}) + 7220(\text{Second Conv}) = 10860$$

Thus, the CNN has a total of 10,860 parameters, solely from the convolutional layers as pooling layers and ReLU activations do not add any learnable parameters.

CNN: Parameter Count

Question: Consider the input tensor of dimensions **10×10×10** where the last dimension is the channel dimension. The following operations are applied:

1. 3×3 convolution (40 channels) with stride 1 and padding 1 for each dimension.
2. ReLU activation.
3. 3×3 max pooling with stride 1 and padding 1 for each dimension.
4. 3×3 convolution (20 channels) with stride 1 and padding 1 for each dimension.
5. ReLU activation.
6. 2×2 max pooling with stride 2 and padding 1 for each dimension.

What is the total parameter count?

CNN: Parameter Count

If you are adding another convolutional layer with 10 output channels following the previous convolutional layer that had 20 output channels, you would again use the formula to calculate the parameters. This layer also uses 3x3 kernels:

Parameters for the Third Convolutional Layer:

- **Kernel Size:** 3×3
- **Input Channels:** 20 (output of the second convolution layer)
- **Output Channels:** 10
- **Bias:** 1 bias per output channel (10 in total)

Formula and Calculation:

$$\text{Number of Parameters} = (\text{Kernel Height} \times \text{Kernel Width} \times \text{Input Channels} + 1) \times \text{Output Channels}$$

$$\text{Number of Parameters} = (3 \times 3 \times 20 + 1) \times 10$$

$$\text{Number of Parameters} = (180 + 1) \times 10$$

$$\text{Number of Parameters} = 181 \times 10$$

$$\text{Number of Parameters} = 1810$$

Thank you!