Convolution

- Convolution is a mathematical operation that combines two functions to produce a third function, representing how one function modifies the other.
- In the context of neural networks, particularly convolutional neural networks (CNNs), convolution is primarily used for processing data with a grid-like topology, such as images.

Convolution

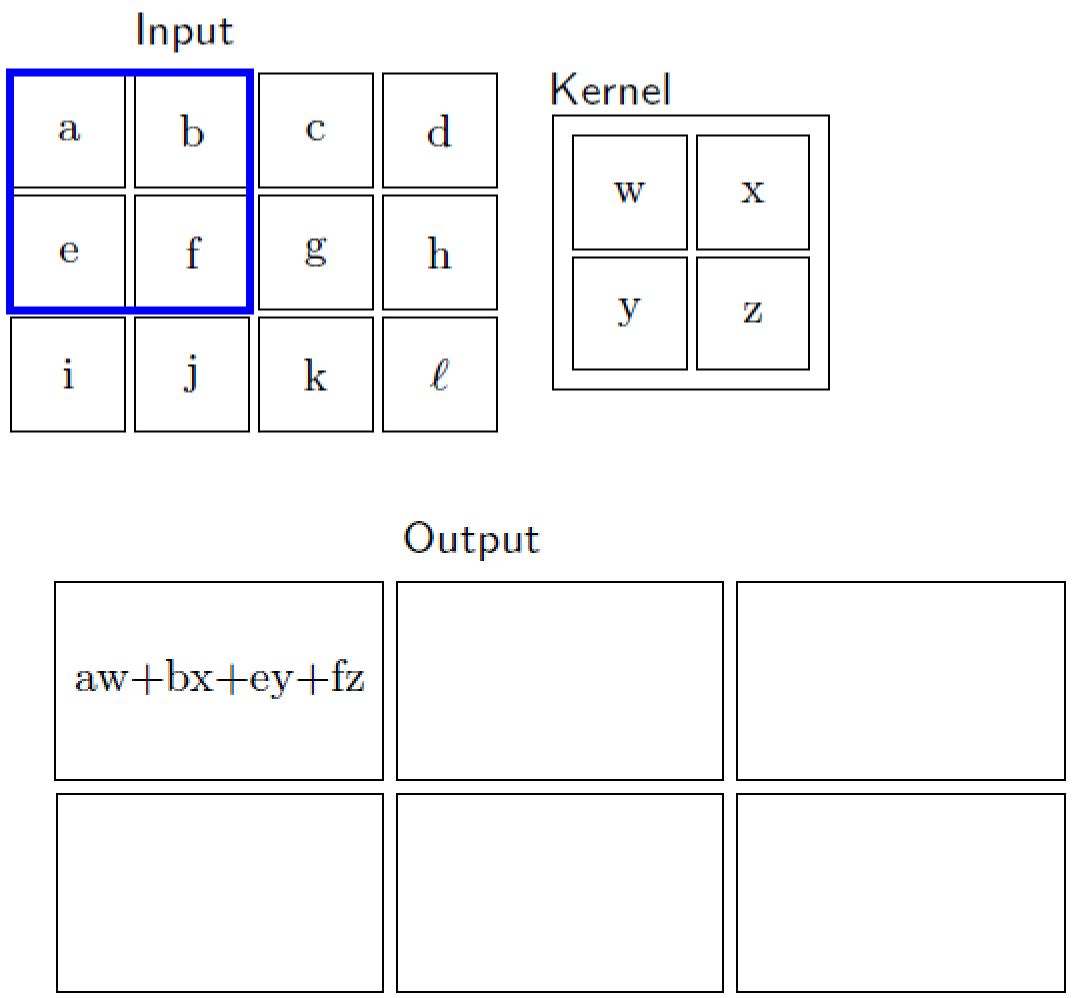
- We may think the image as a 2D matrix
- We may use a 2D filter of size m×n for convolution.
- The convolution operation applied on image and filter gives the modification of the image by the filter.
- This convolution operation works as follows:

$$S_{ij} = (I * K)_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i+a,j+b} K_{a,b}$$

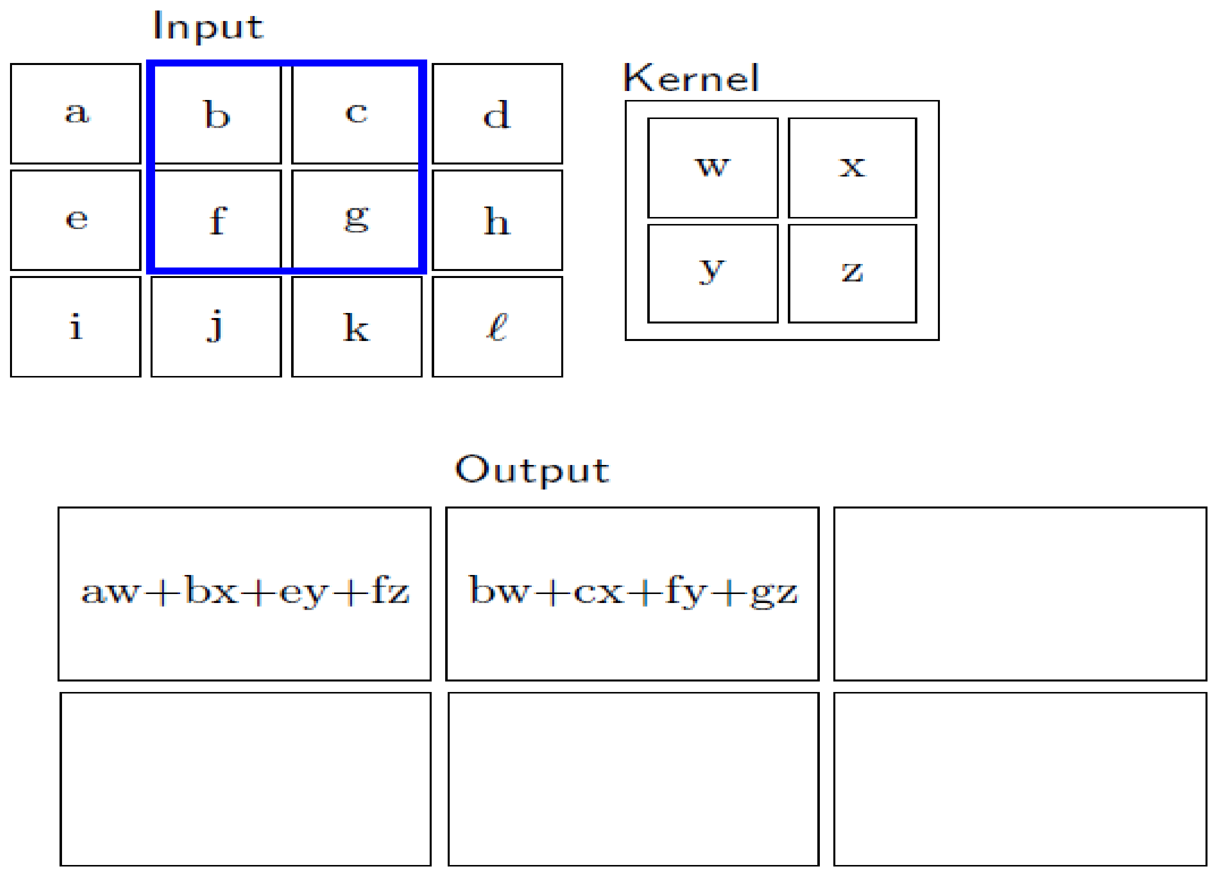
Key Aspects of Convolution

- **Kernel/Filter**:
  - A small matrix (e.g., 3x3 or 5x5) that slides over the input data (e.g., an image). The kernel is responsible for extracting features like edges, textures, or patterns.
- **Sliding Window**:
  - The kernel moves across the input image in small steps (called strides), performing the convolution operation at each position.
- **Dot Product**:
  - At each position, the kernel performs an element-wise multiplication with the portion of the input it covers, and the results are summed up to produce a single output value. This process captures local patterns in the data.
- **Output Feature Map**:
  - The result of applying the convolution operation across the entire input creates a new matrix called a feature map, which highlights the features such as edges, shapes, and textures detected by the kernel.

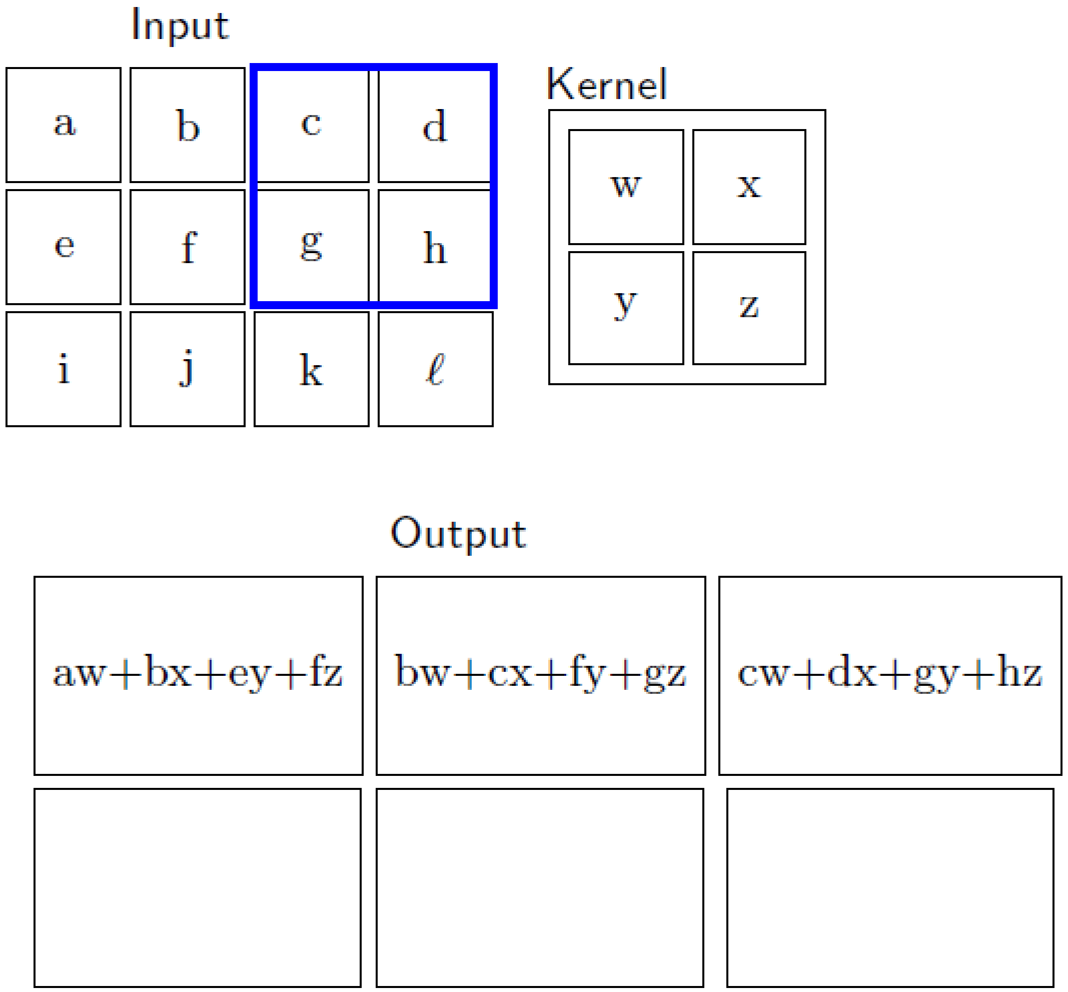
Convolution



Convolution



Convolution



Input

a	b	c	d
e	f	g	h
i	j	k	ℓ

Kernel

w	x
y	z

Output

aw+bx+ey+fz	bw+cx+fy+gz	cw+dx+gy+hz

Convolution

Input

a	b	c	d
e	f	g	h
i	j	k	ℓ

Kernel

w	x
y	z

Output

aw+bx+ey+fz	bw+cx+fy+gz	cw+dx+gy+hz
ew+fx+iy+jz		

Convolution

Input

a	b	c	d
e	f	g	h
i	j	k	ℓ

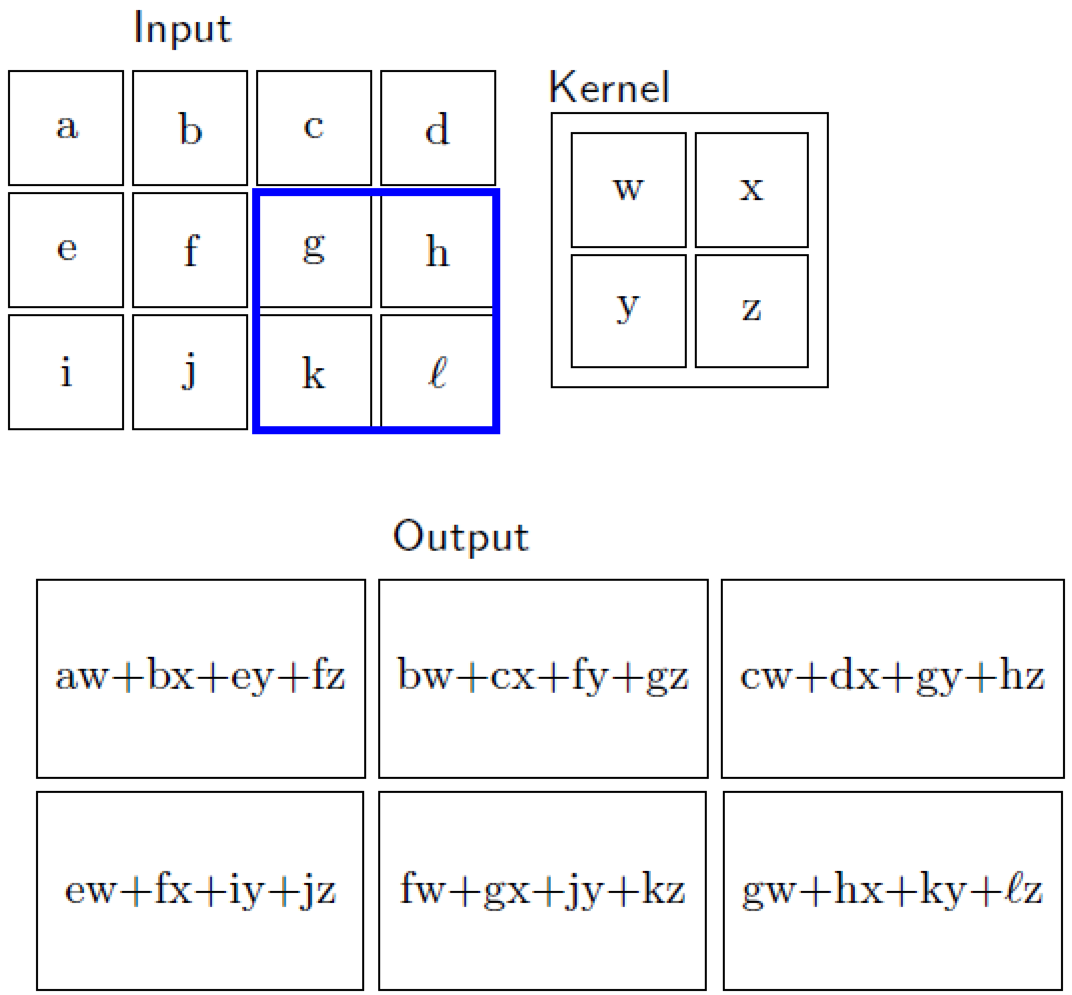
Kernel

w	x
y	z

Output

aw+bx+ey+fz	bw+cx+fy+gz	cw+dx+gy+hz
ew+fx+iy+jz	fw+gx+jy+kz	

Convolution



- From an image of size 3× 4, we obtain a feature map of size 2 × 3
- To get the feature map of same size, we need to add one extra row and one extra column in the image, called padding

Padding

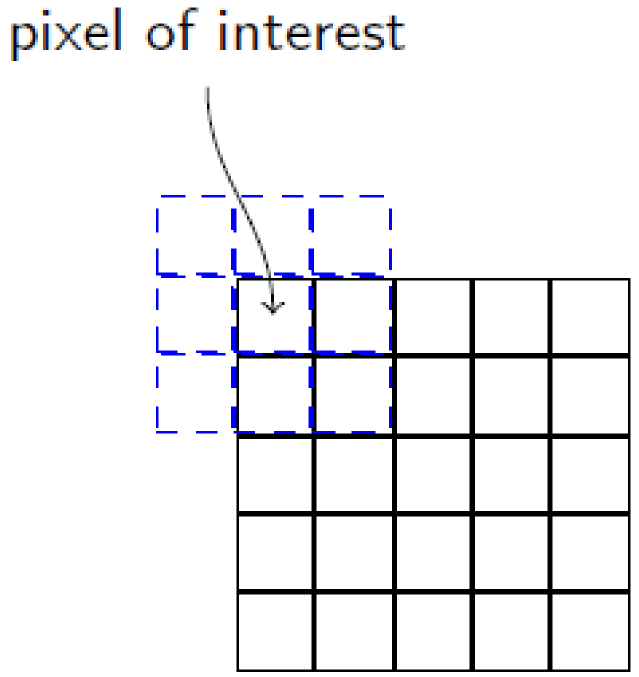
- If top left one is the pixel of interest, then we must have to place the filter as shown in the figure. In this case, padding is necessary.

1. Valid Padding:

- No padding is applied. The convolution operation is only applied to valid positions where the kernel fits within the input dimensions.
- **Effect**: Reduces the spatial dimensions of the output. For example, a 5×5 input with a 3×3 kernel will produce a 3×3 output.

2. **Same Padding**:

- Padding is applied such that the output dimensions are the same as the input dimensions. This is achieved by adding a sufficient number of zeroes around the input.
- **Effect**: Preserves the spatial dimensions. For example, a 5×5 input with a 3×3 kernel will produce a 5×5 output.



Padding

- Padding is a technique used in convolutional neural networks (CNNs) and other neural network architectures to manage the spatial dimensions of input data during the convolution operation.
- Padding involves adding extra pixels (usually zeros) around the edges of the input data (e.g., images) before applying the convolution operation.
- This extra border of pixels helps in controlling the output size and maintaining important features at the edges of the input.
- Choosing the right type of padding depends on the specific requirements of the neural network architecture and the task at hand.

When is Padding Required?

1. **Preserving Dimensions**: When we want the output size to match the input size, especially in tasks like segmentation or when stacking multiple layers of convolutions.
2. **Edge Features**: To ensure that features near the edges of the input are effectively captured. Without padding, convolutions near the edges can miss relevant information.
3. **Reducing Spatial Shrinkage**: In deeper networks, successive convolutions without padding can lead to significant reductions in spatial dimensions, which can make it difficult to maintain meaningful representations.
4. **Improving Network Performance**: Padding can help in stabilizing the learning process and improving the overall performance of the model by providing more context around the edges of the input data.

Kernel / Filter

- In a convolutional neural network (CNN), a kernel, also known as a filter, is a small matrix used to scan over the input data (such as an image) to detect features.
- **Size**: Kernels are usually smaller than the input data. Common sizes are 3x3, 5x5, or 7x7 pixels.
- **Feature Detection**: Different kernels can be trained to detect various features such as edges, textures, or patterns. Early layers in a CNN typically learn simple features, while deeper layers learn more complex representations.

Kernel / Filter

- **Stride**: This parameter determines how many pixels the kernel moves after each operation. A larger stride results in a smaller output size.
- **Padding**: Sometimes, zeroes are added around the input data to control the spatial size of the output. This helps in preserving the dimensions and preventing loss of important information at the borders.
- **Multiple Kernels**: Each convolutional layer can have multiple kernels, allowing the layer to learn a variety of features from the input.
- Overall, kernels are fundamental to how CNNs process data, enabling them to learn hierarchical representations effectively.
- Different types of kernels can serve various purposes depending on the specific application.

Edge Detection Kernel

- **Sobel Kernel**: Used to detect edges in horizontal and vertical directions.
  - Horizontal:

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

  - Vertical:

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

- It computes gradients in horizontal and vertical direction

- **Canny Edge Detector**: A multi-stage algorithm that includes several steps to detect edges more robustly.

Canny Edge Detector

- The Canny edge detector doesn't rely on a single kernel like the Sobel operator.
- Instead, it uses a series of steps, including the application of Gaussian filters for smoothing, followed by gradient computation, non-maximum suppression, and hysteresis thresholding.
- However, the gradient computation often involves using derivatives, and the commonly used kernels for this step are:

1. **Gaussian Kernel**: Used for smoothing the image to reduce noise before edge detection.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Canny Edge Detector

2. **Gradient Kernels**: After smoothing, the gradient is calculated using derivative kernels, which can be similar to Sobel kernels:

   - **Horizontal Gradient Kernel**:

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

   - **Vertical Gradient Kernel**:

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

- In summary, while Canny edge detection doesn’t use a single kernel as in other methods, it employs Gaussian smoothing and gradient kernels to effectively identify edges.

Blurring Kernels

- Example: Gaussian Kernel, Average/Mean Kernel

1. **Gaussian Kernel :**

- A Gaussian kernel is a type of filter used in image processing, particularly for smoothing or blurring images.
- It applies the Gaussian function, which helps reduce noise and detail in an image by averaging the pixel values based on their distance from a central pixel. The Gaussian function is defined as:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where, σ (Sigma) controls the width of the Gaussian curve. A larger sigma results in more blurring, while a smaller sigma retains more detail.

- **Kernel Size**: The kernel is typically a square matrix (e.g., 3x3, 5x5, or 7x7) derived from the Gaussian function. The size is often chosen to be odd to have a center pixel.

Blurring Kernels: Gaussian Kernel

- A 3 × 3 Gaussian Kernel: $\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ normalized $\begin{bmatrix} 1/16 & 2/16 & 1/16 \\ 2/16 & 4/16 & 2/16 \\ 1/16 & 2/16 & 1/16 \end{bmatrix}$

- A 5 × 5 Gaussian Kernel: $\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$ $\begin{bmatrix} \frac{1}{256} & \frac{4}{256} & \frac{6}{256} & \frac{4}{256} & \frac{1}{256} \\ \frac{4}{256} & \frac{16}{256} & \frac{24}{256} & \frac{16}{256} & \frac{4}{256} \\ \frac{6}{256} & \frac{24}{256} & \frac{36}{256} & \frac{24}{256} & \frac{6}{256} \\ \frac{4}{256} & \frac{16}{256} & \frac{24}{256} & \frac{16}{256} & \frac{4}{256} \\ \frac{1}{256} & \frac{4}{256} & \frac{6}{256} & \frac{4}{256} & \frac{1}{256} \end{bmatrix}$

Blurring Kernels: Average Kernel

- An average kernel, also known as a box filter or mean filter, is a simple smoothing technique used in image processing.
- It works by replacing each pixel's value with the average of the pixel values in its neighborhood, resulting in a blur effect.
- This helps reduce noise and blur the image.
- **Kernel Size**: Average kernels can be of various sizes, commonly 3x3, 5x5, or 7x7. The size determines the area over which the averaging is performed.
- **Normalization**: The values in the kernel are typically equal and sum to 1, ensuring that the output image has a similar brightness to the input image.

Blurring Kernels: Average Kernel

- A 3 × 3 Gaussian Kernel: $\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$

- A 5 × 5 Gaussian Kernel: $\begin{bmatrix} \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} \\ \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} \\ \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} \\ \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} \\ \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} \end{bmatrix}$

Sharpening Kernel

- A sharpening kernel is used in image processing to enhance the edges and fine details in an image.
- It works by emphasizing the differences between adjacent pixel values, effectively making the image appear sharper.
- **Applications:**

  1. Detail Enhancement: Useful for emphasizing edges and textures in images.

  2. Image Preprocessing: Often applied before other processing tasks to improve feature detection.

  3. Artistic Effects: Can create a stylized look in images by enhancing outlines and details.

- The most common sharpening kernels include the Laplacian kernel and the unsharp mask kernel.

Sharpening Kernel

1. **Laplacian Kernel**:

- The second derivative measures the rate of change of the gradient. In image processing, this means detecting areas where there is a rapid change in intensity, which corresponds to edges.
- The Laplacian operator calculates the second derivative in both the x and y directions. When applied to an image, it helps identify regions of rapid intensity change (edges) by emphasizing the areas where the intensity changes sharply.

Mathematical Formulation: The Laplacian of an image $I(x, y)$ can be expressed as:

$$\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Here, ∇^2 denotes the Laplacian operator.

Laplacian Kernel:

1. A typical 3x3 Laplacian kernel looks like this: $$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

- This kernel emphasizes edges by subtracting the average of the neighboring pixels from the center pixel.
- When the Laplacian kernel is applied to an image, it produces a result where edges are highlighted, and flat regions are suppressed, effectively making the edges more pronounced.
- The result can sometimes be combined with the original image for further enhancement, a technique often used in unsharp masking.

Unsharp Mask Kernel

- **Unsharp Masking Process:**

(i) Gaussian Blur: First, a Gaussian blur is applied to the original image to create a blurred version. This helps reduce noise and fine details.

(ii) Difference Calculation: The blurred image is then subtracted from the original image to highlight the edges.

(iii) Combination: The result is added back to the original image, often with a scaling factor to control the degree of sharpening.

- A common kernel used in this technique is:
- This kernel effectively enhances edges by applying a weighted average where the center pixel gets more weight than its neighbors.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Sharpening Kernels

Difference:

(i) In **Laplacian Kernel, the** center value is **4** (detecting edges by subtracting the average). But in **Unsharp Mask Kernel,** the center value is **5**, which increases the emphasis on the center pixel more than in the Laplacian kernel.

(ii) Laplacian kernel detects edges based on the rate of change, focusing on curvature rather than enhancing pixel values. But Unsharp Mask enhances edges through a combination of the original pixel value and its neighbors, effectively sharpening the image.

(iii) While both techniques are used for edge detection and enhancement, they operate on different principles. The unsharp mask is not a second derivative but a method to accentuate edges by emphasizing differences in pixel intensity.

Type of Kernel

4. Embeddings Kernels:

- **Feature Extraction Kernels**: In CNNs, these kernels are learned during training and can capture complex features at various levels of abstraction (e.g., edges, textures, shapes).

5. **Dilated Kernels**

- **Dilated Convolution**: Dilated kernels, also known as atrous convolution kernels, are used in convolutional neural networks to increase the receptive field without increasing the number of parameters or the amount of computation. This allows the network to capture larger context while maintaining spatial resolution.

Type of Kernel

- A standard 3x3 kernel can be dilated in various ways. For instance, with a dilation rate of 2, the kernel is applied with gaps between its elements:
- **Standard 3x3 Kernel** $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ **Dilated 3x3 Kernel with Dilation Rate = 2** $\begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}$
- **Dilation Rate:** The number "2" indicates that there is one pixel of space between the kernel elements in both horizontal and vertical directions.
- **Application:** This dilated kernel can cover a larger area of the input while still producing output of the same size as a regular convolutional layer, making it useful for tasks like segmentation and dense prediction.
- **Benefits**

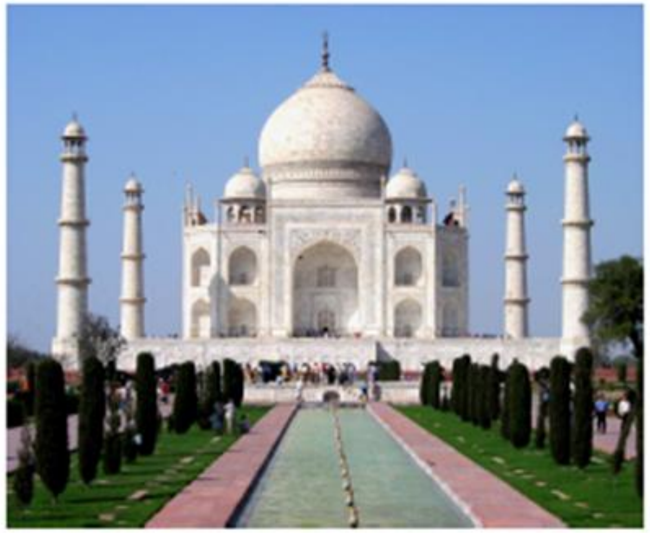
  1. Larger Receptive Field: Allows the network to capture wider context without downsampling.

  2. Preserves Spatial Resolution: Useful in applications where maintaining the original resolution is important, like in semantic segmentation.

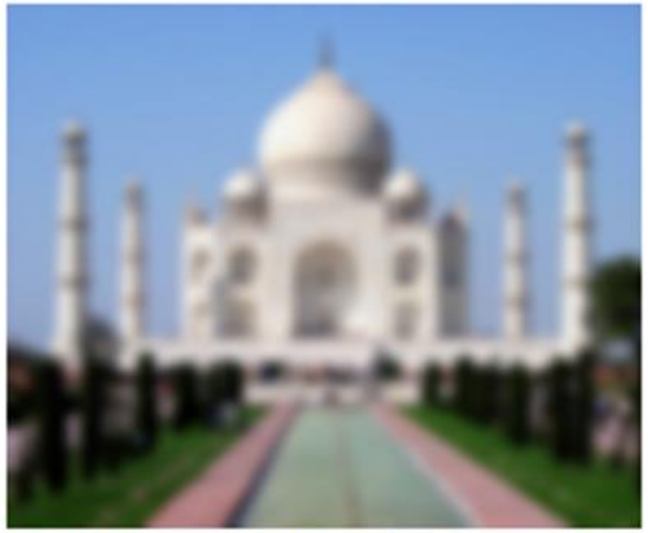
6. **Custom Kernels**

- Users can design custom kernels tailored for specific tasks or datasets, such as detecting specific patterns, textures, or other features relevant to their application.

Convolution



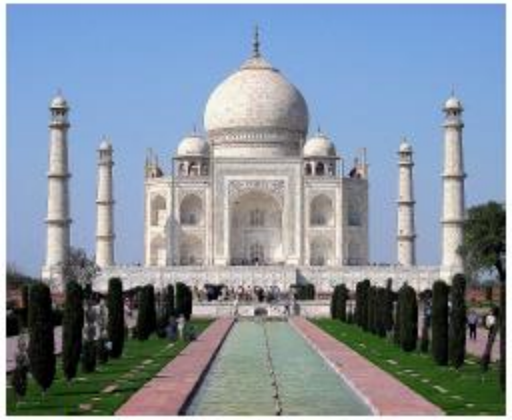
$$* \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} =$$



blurs the image

Convolution

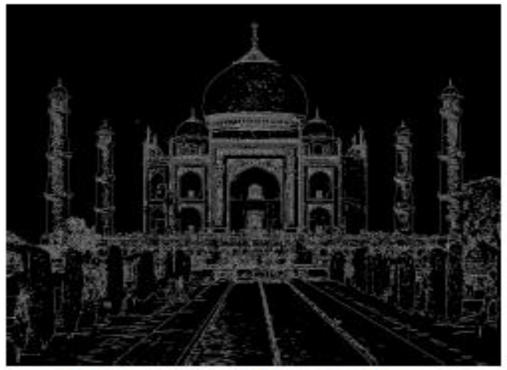
$$* \quad \begin{matrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{matrix} \quad =$$



sharpens the image

Convolution

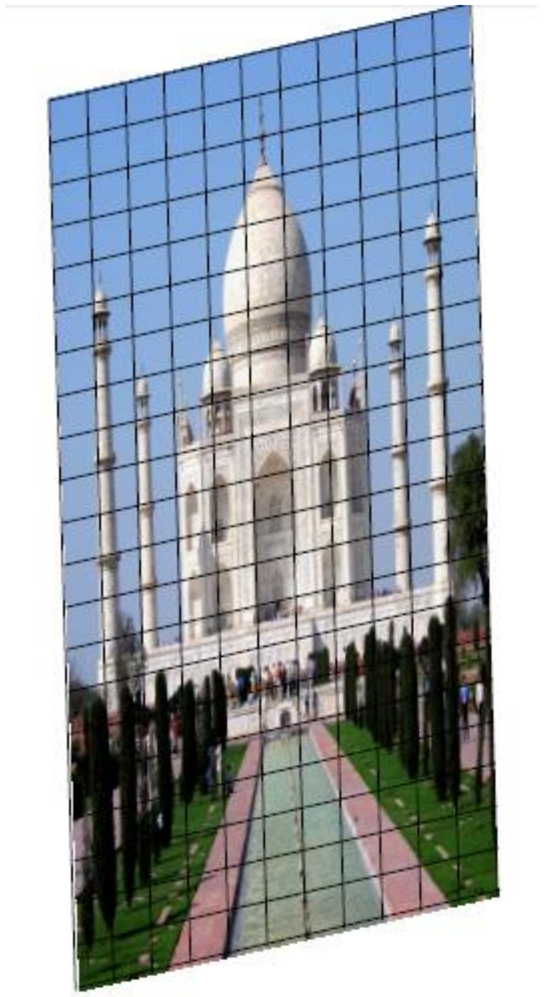
$$* \begin{matrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{matrix} =$$



detects the edges

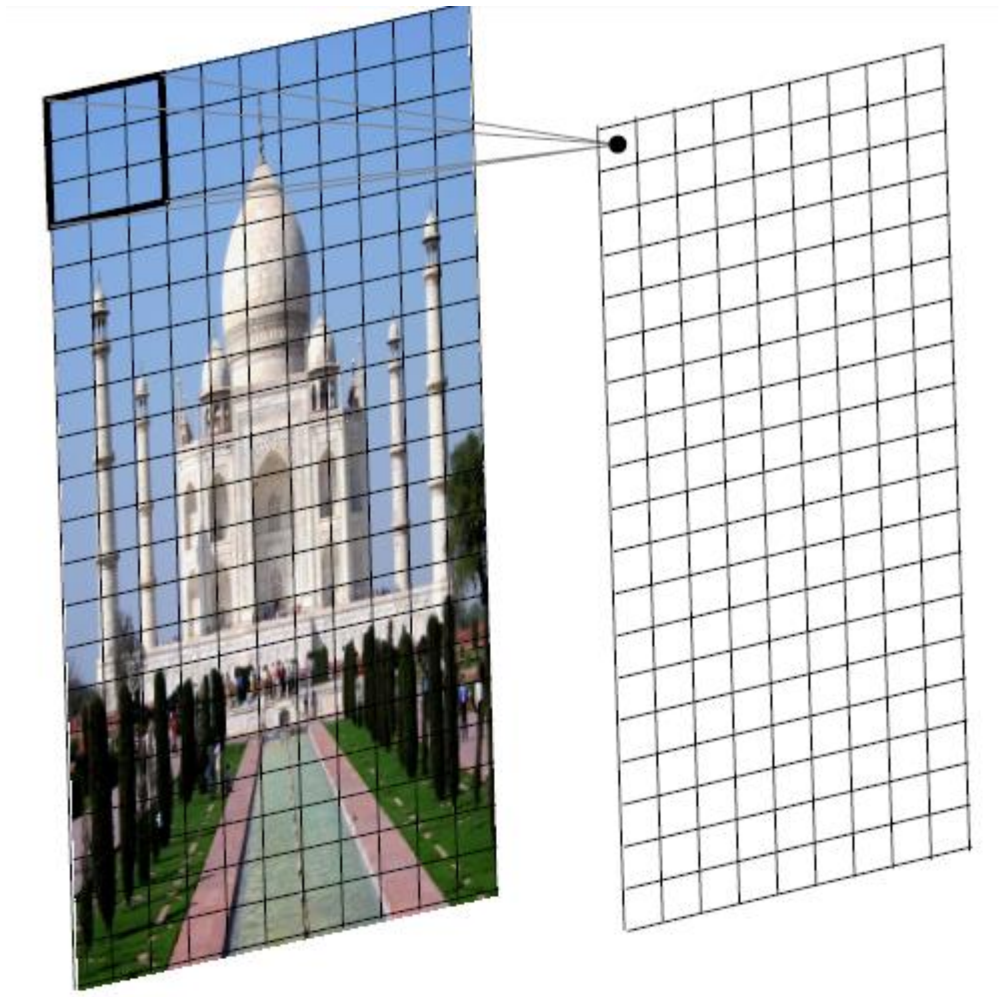
- It can be used for edge detection and sharpening. It has a form similar to a high-pass filter.
- **Edge Detection:** This kernel emphasizes the center pixel while considering its surrounding pixels. The negative center value effectively highlights areas where there is a significant change in intensity, making it useful for detecting edges.
- **High-Pass Filter**: By subtracting a weighted average of the neighboring pixels from the center pixel, it removes low-frequency components (smooth areas) and enhances high-frequency components (edges).
- **Sharpening Effect**: When convolved with an image, it can create a sharpening effect, making edges more pronounced.

Convolution



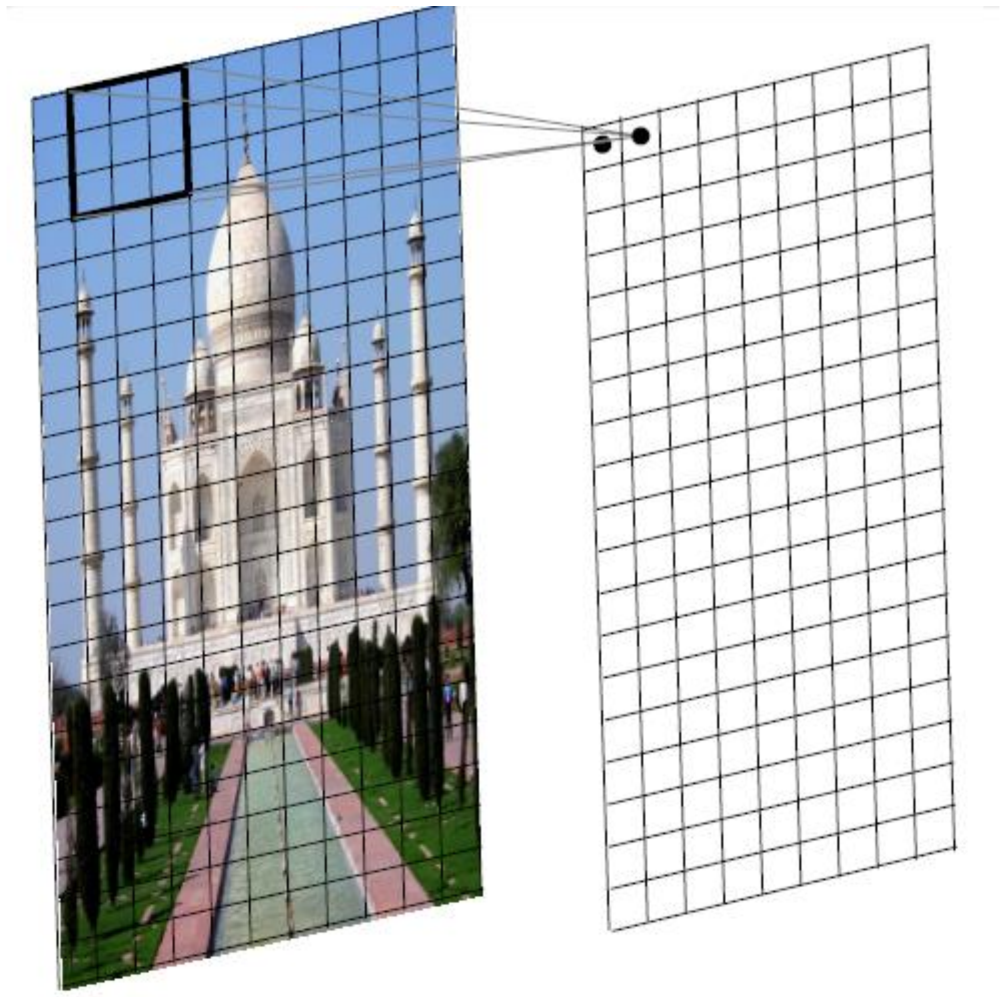
- We just slide the kernel over the input image

Convolution



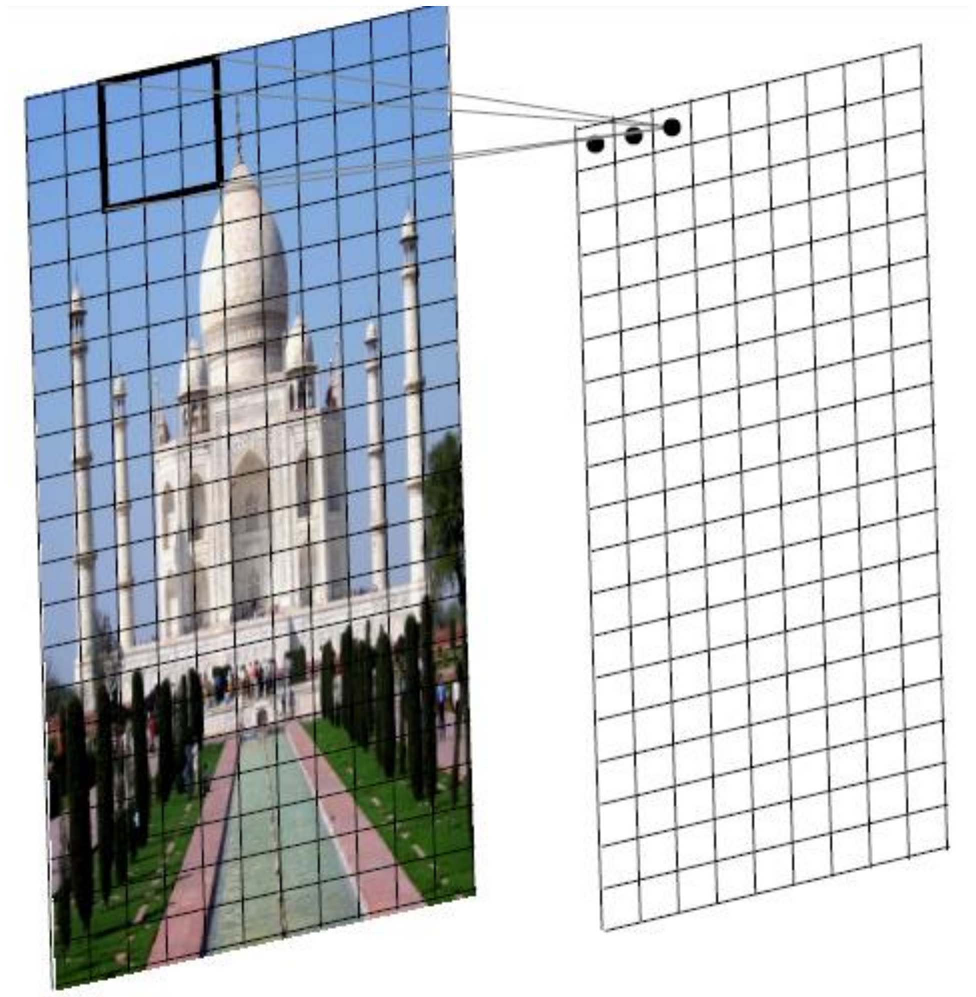
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output

Convolution



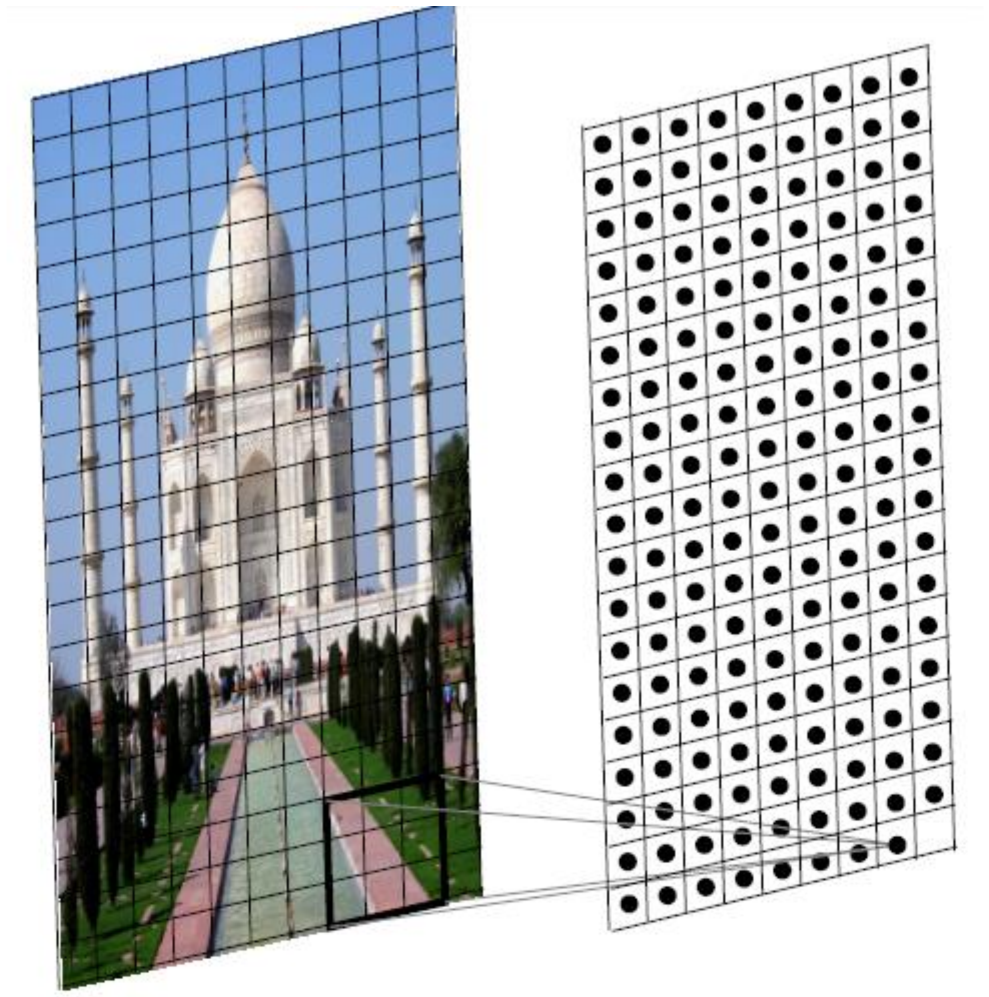
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output

Convolution



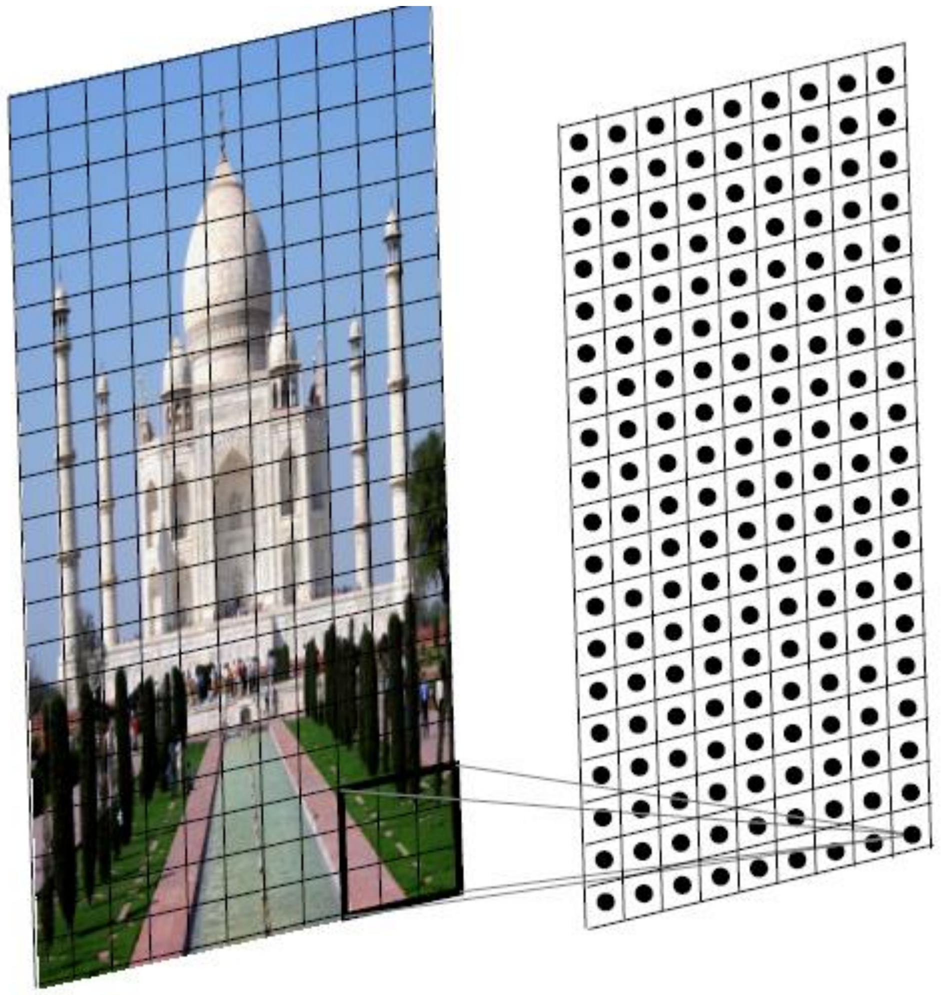
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output

Convolution



- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output

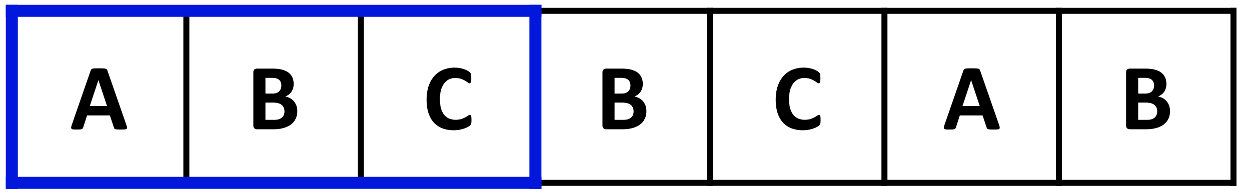
Convolution



- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output
- The resulting output is called a feature map.
- We can use multiple filters to get multiple feature maps.

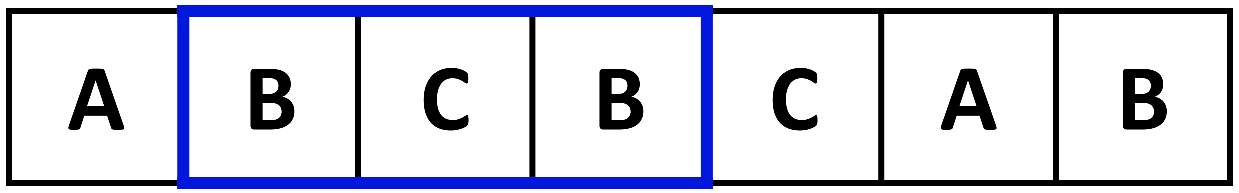
Convolution in 1-D

- In the 1D case, we slide a one dimensional filter over a one dimensional input



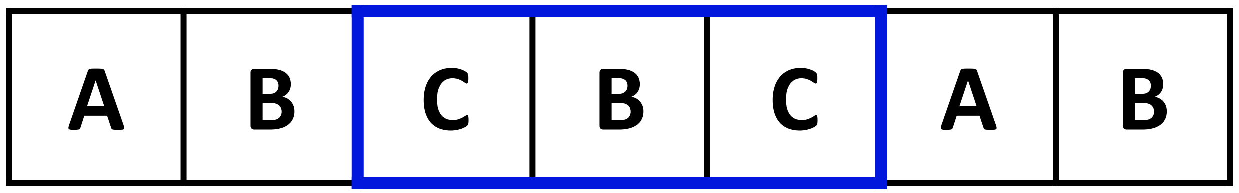
Convolution

- In the 1D case, we slide a one dimensional filter over a one dimensional input



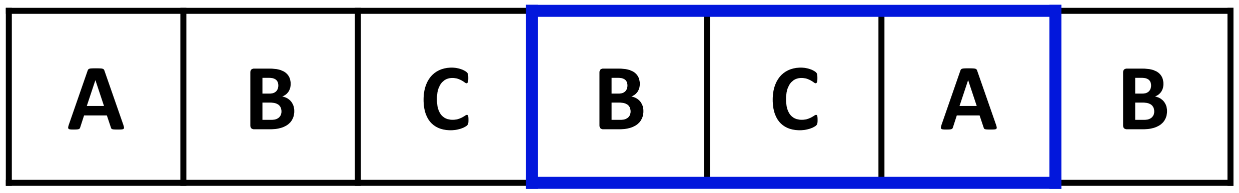
Convolution

- In the 1D case, we slide a one dimensional filter over a one dimensional input



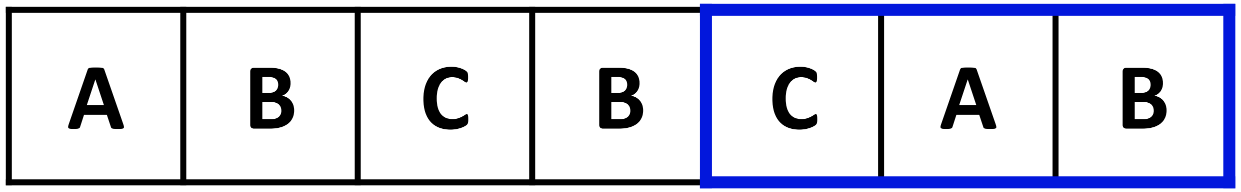
Convolution

- In the 1D case, we slide a one dimensional filter over a one dimensional input

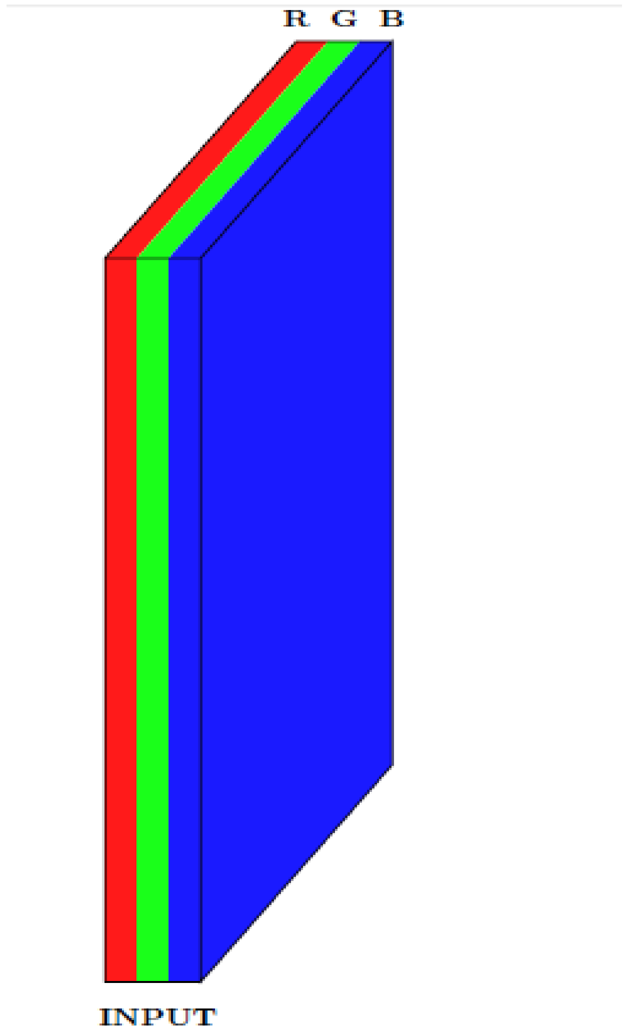


Convolution

- In the 1D case, we slide a one dimensional filter over a one dimensional input

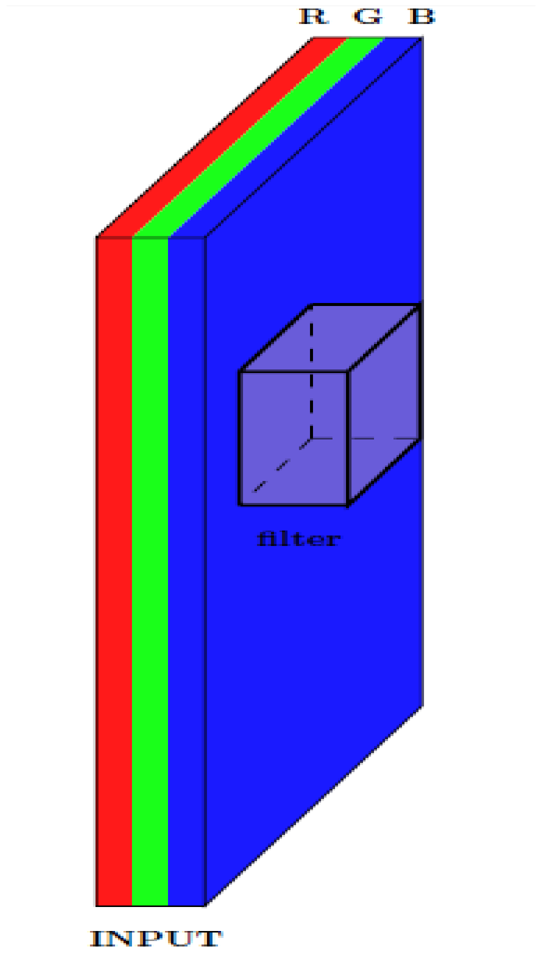


Convolution on 3-D



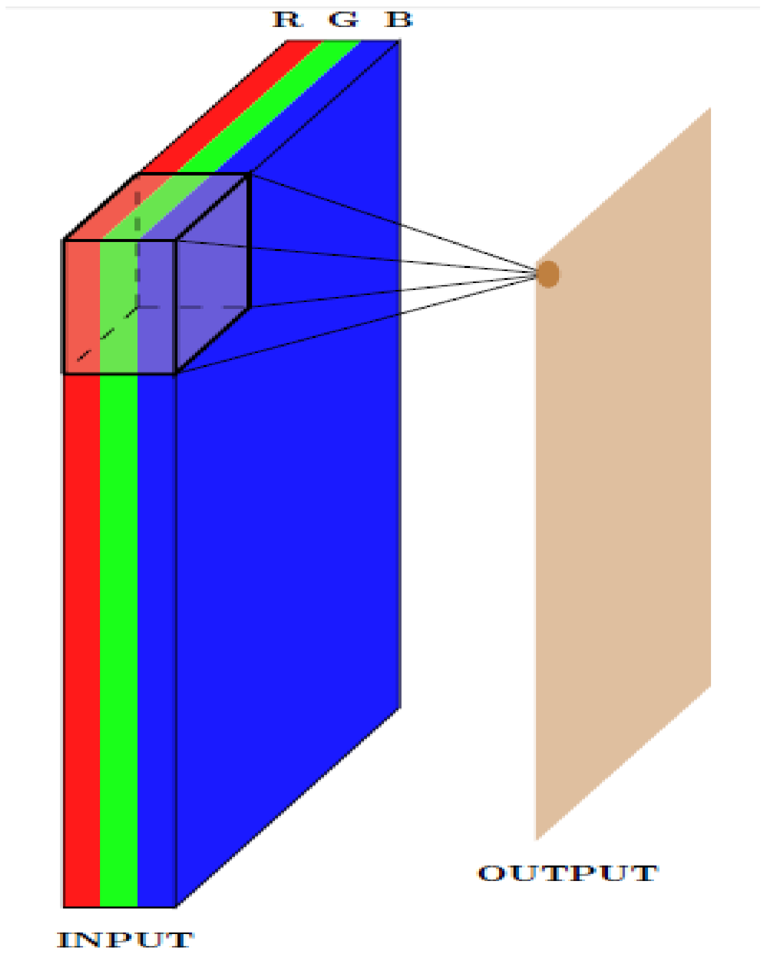
- What would a 3D filter look like?

Convolution on 3-D



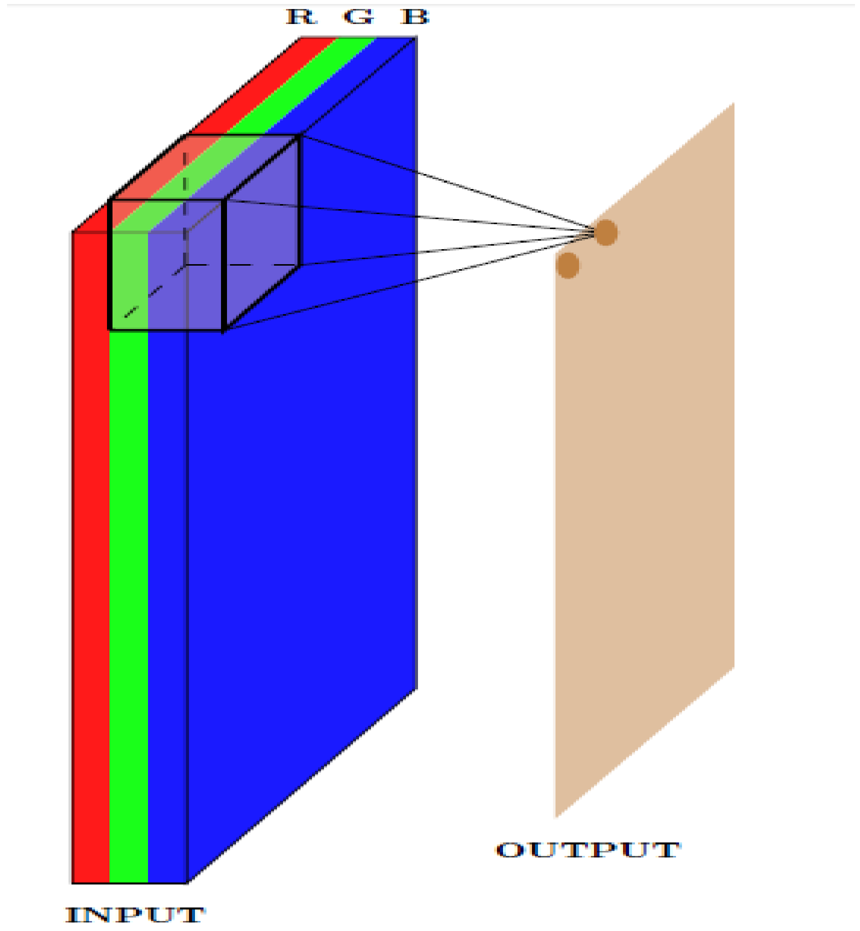
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume

Convolution on 3-D



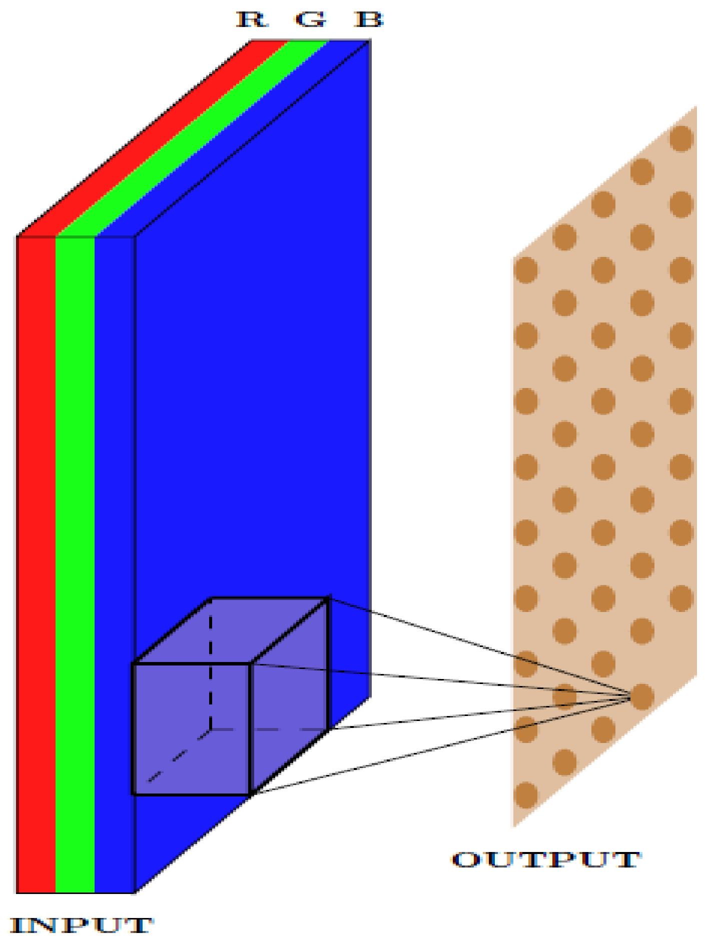
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation

Convolution on 3-D



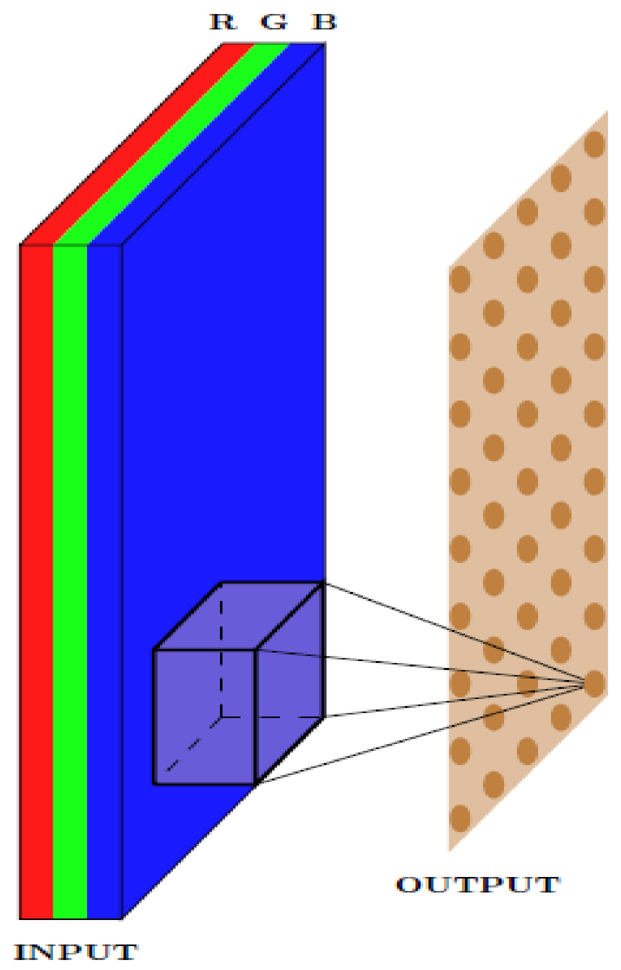
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation

Convolution on 3-D



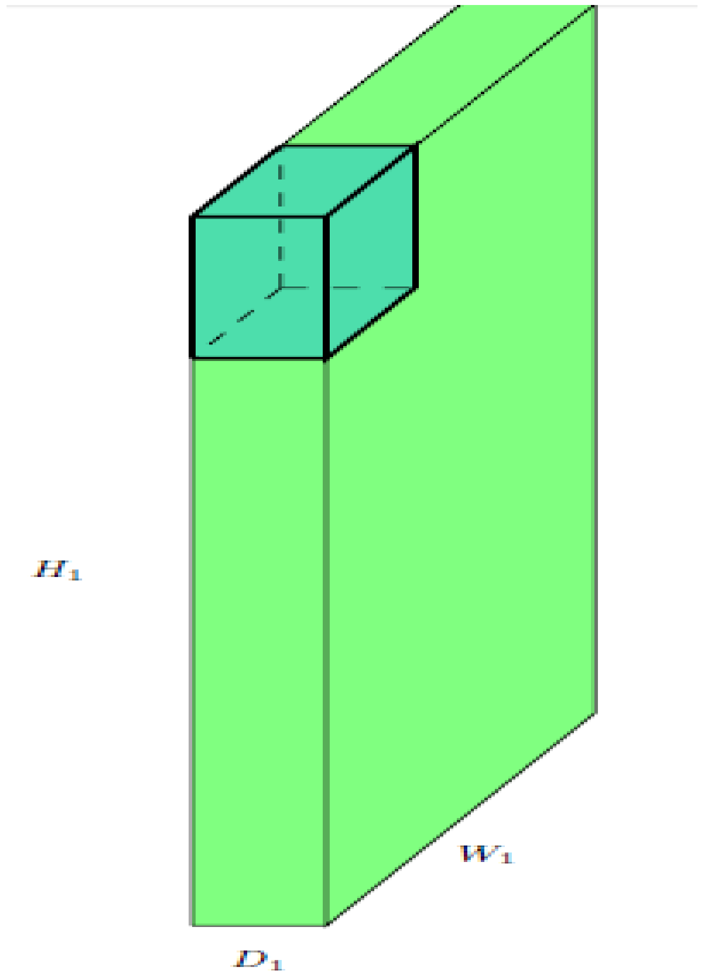
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation

Convolution on 3-D



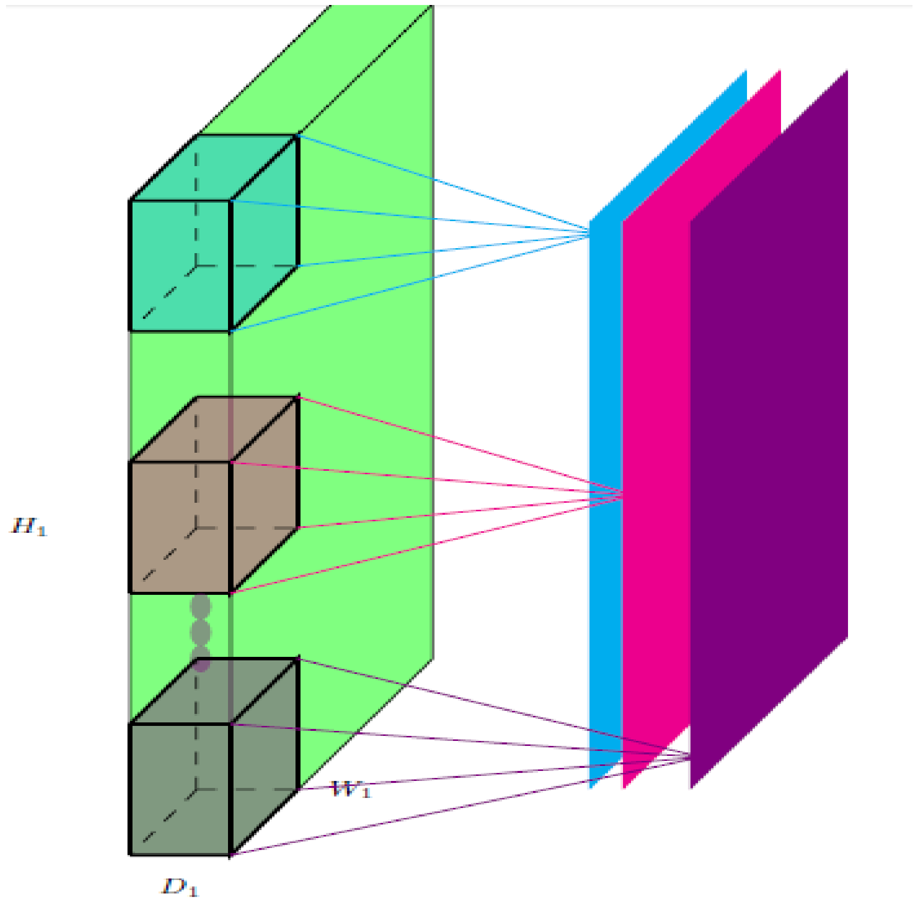
- In effect, we are doing a 2D convolution operation on a 3D input
- Because the filter moves along the height and the width but not along the depth
- As a result the output will be 2D (only width and height, no depth)
- Once again we can apply multiple filters to get multiple feature maps

Convolution on 3-D



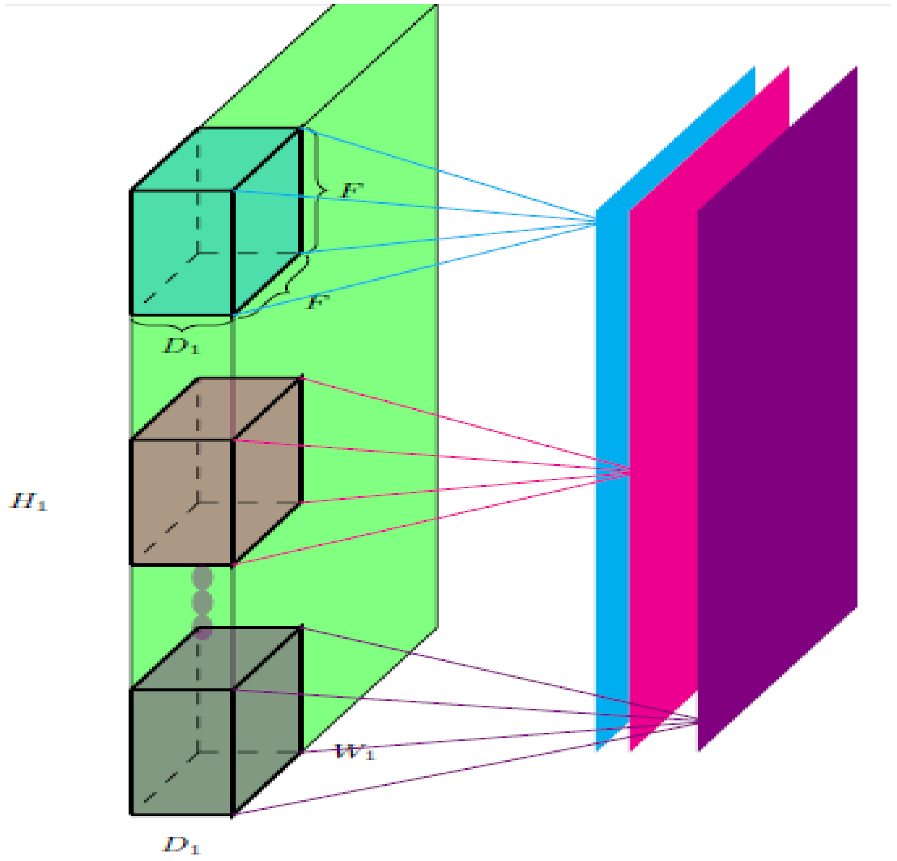
- Width (W_1), Height (H_1) and Depth (D_1) of the original input

Convolution on 3-D



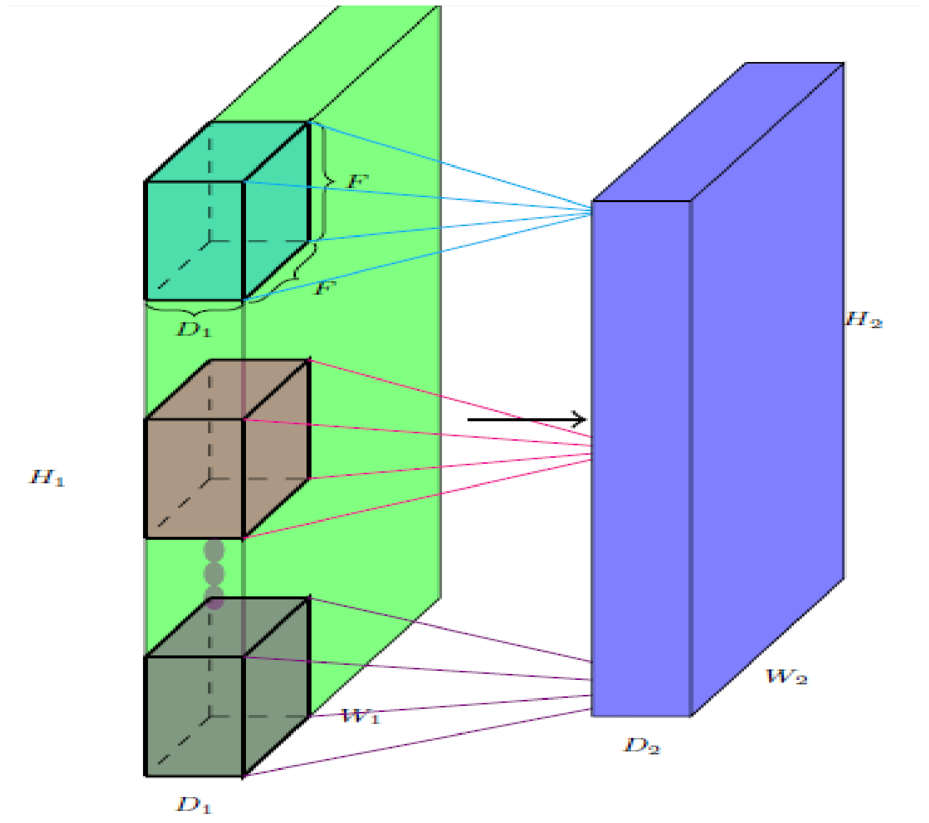
- Width (W_1), Height (H_1) and Depth (D_1) of the original input
- The number of filters K

Convolution on 3-D



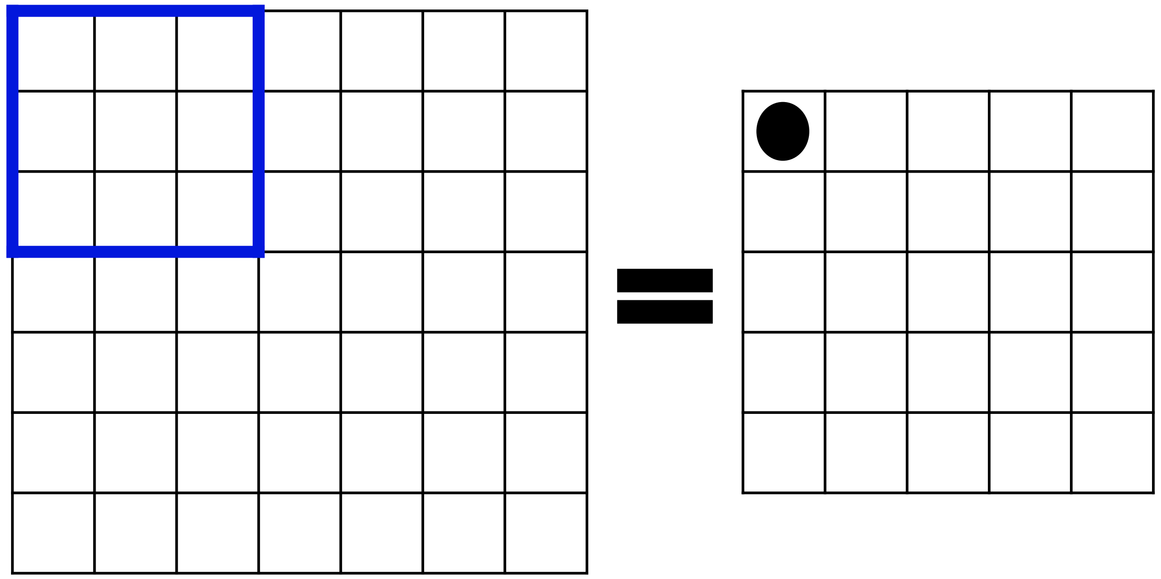
- Width (W_1), Height (H_1) and Depth (D_1) of the original input
- The number of filters K
- The spatial extent (F) of each filter (the depth of each filter is same as the depth of each input)
- So, RGB image is W1 × H1 × D1 and kernel is F × F × D1

Convolution on 3-D



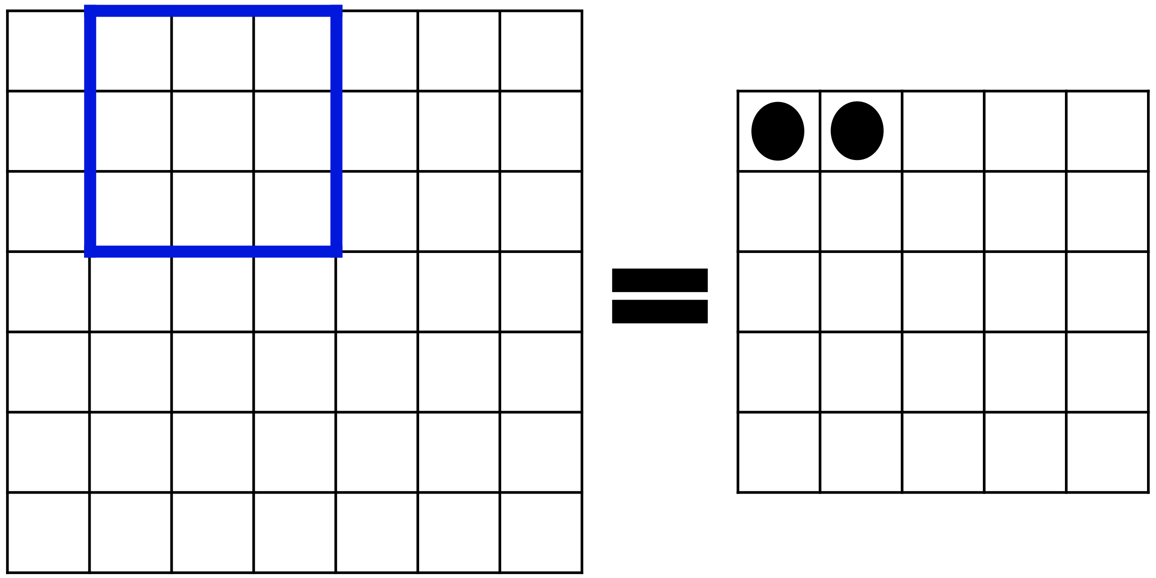
- Width (W_1), Height (H_1) and Depth (D_1) of the original input
- The number of filters K
- The spatial extent (F) of each filter (the depth of each filter is same as the depth of each input)
- So, RGB image is W1 × H1 × D1 and kernel is F × F × D1
- The output is W_2 x H_2 x D_2 (we will soon see a formula for computing W_2, H_2 and D_2)

Calculation of Output Dimension



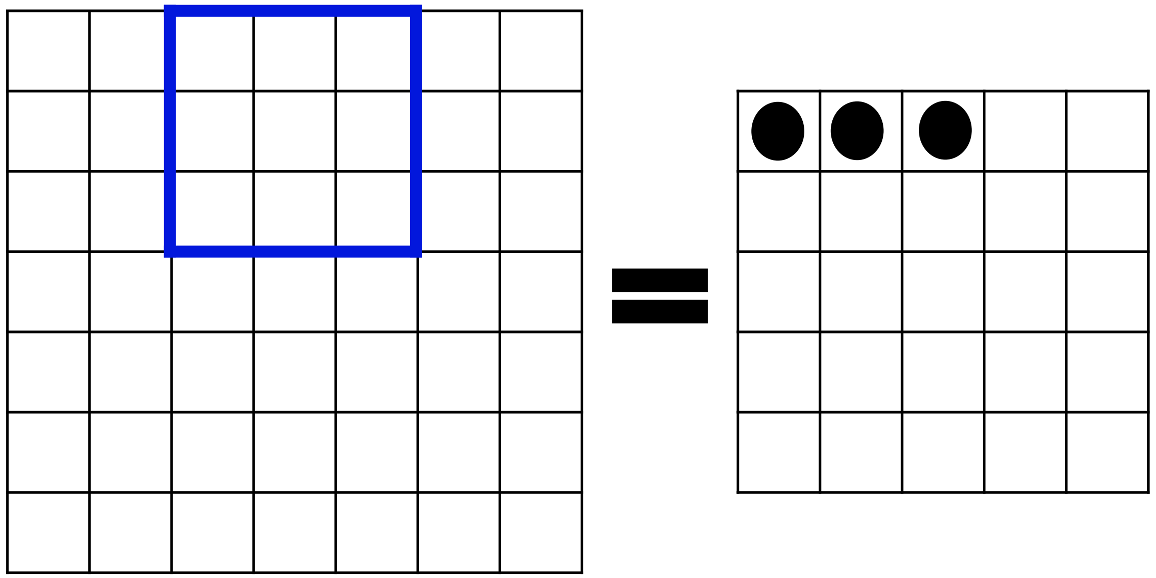
- Let us compute the dimension (W_2, H_2) of the output

Calculation of Output Dimension



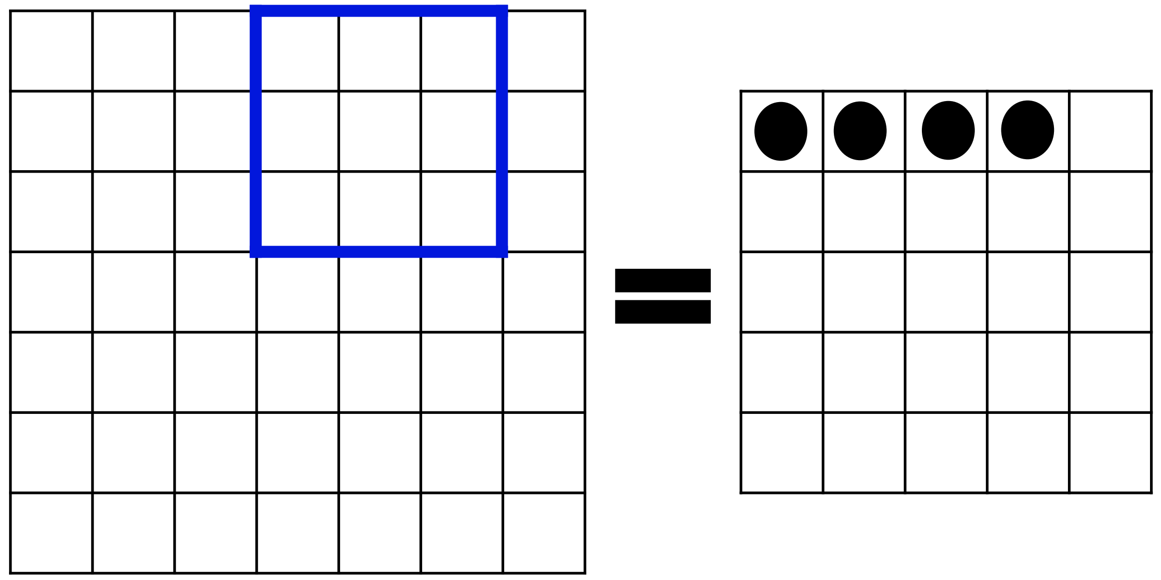
- Let us compute the dimension (W_2, H_2) of the output

Calculation of Output Dimension



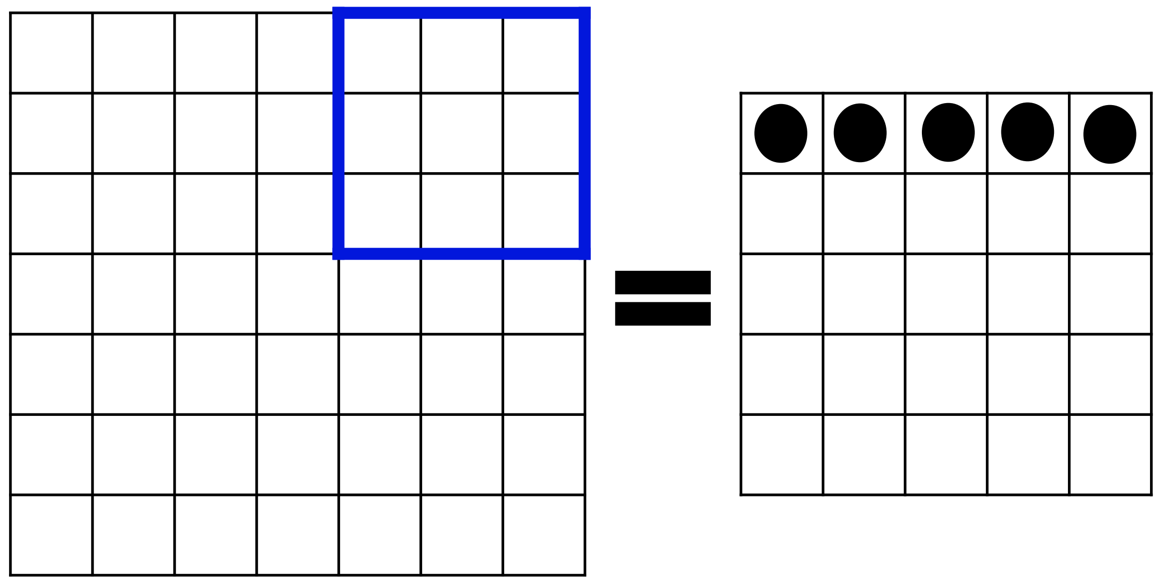
- Let us compute the dimension (W_2, H_2) of the output

Calculation of Output Dimension



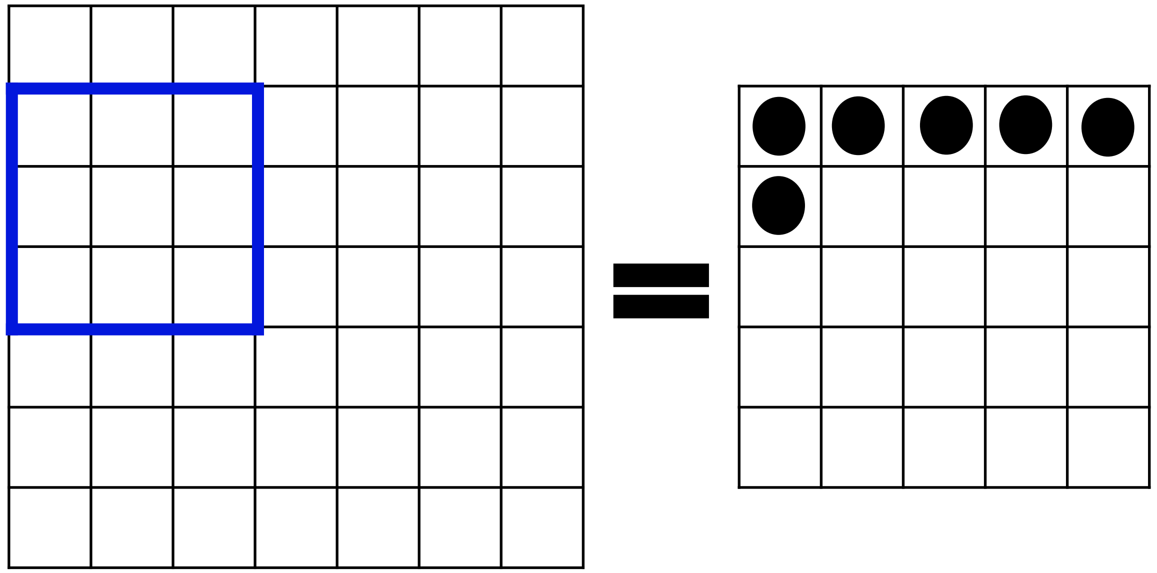
- Let us compute the dimension (W_2, H_2) of the output

Calculation of Output Dimension



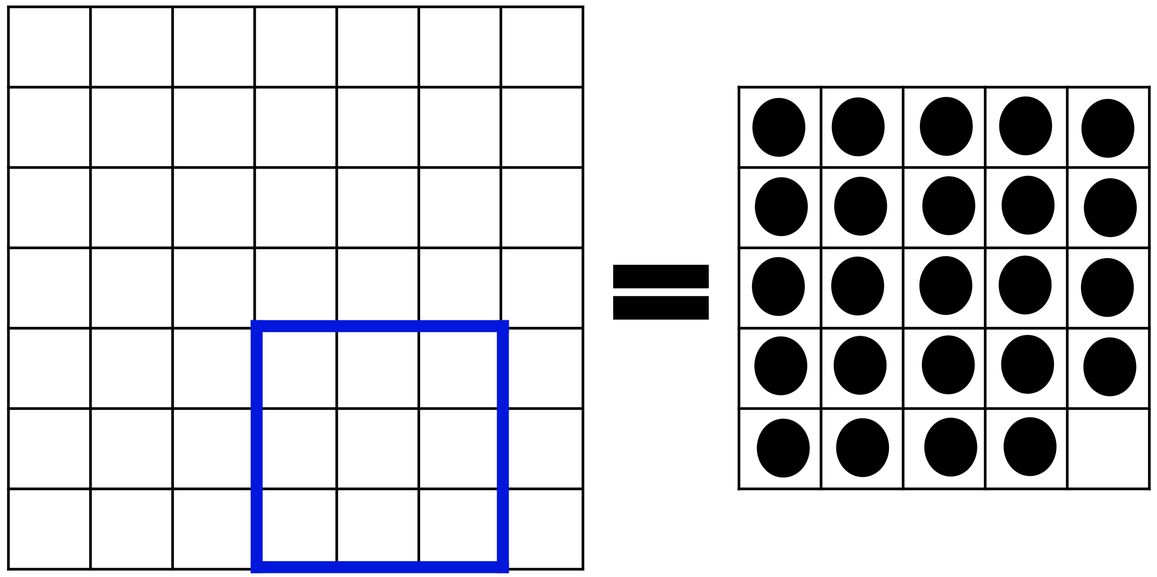
- Let us compute the dimension (W_2, H_2) of the output

Calculation of Output Dimension



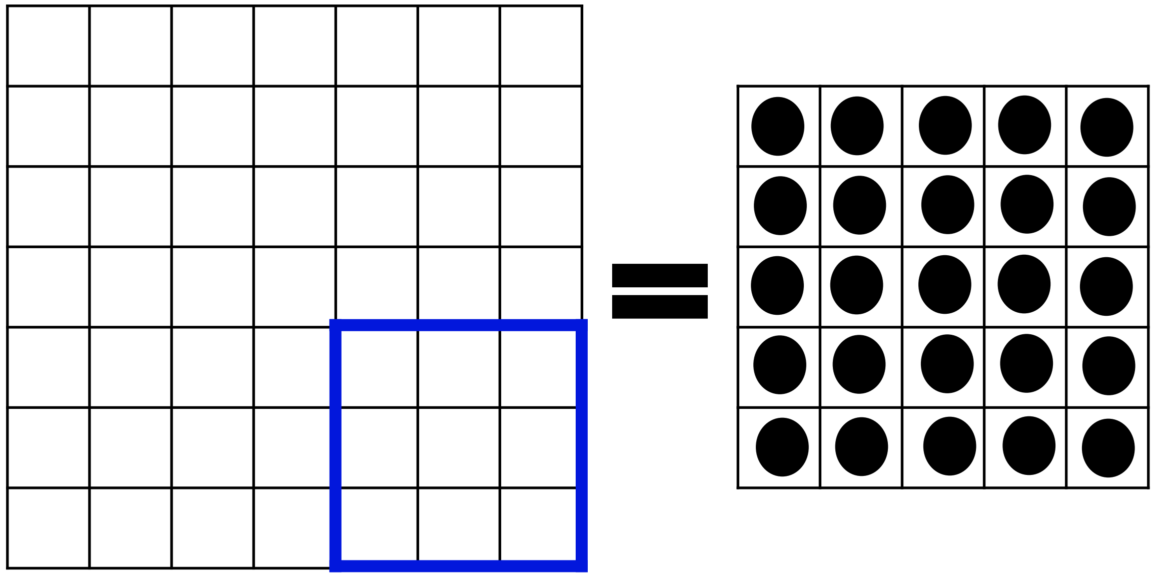
- Let us compute the dimension (W_2, H_2) of the output

Calculation of Output Dimension



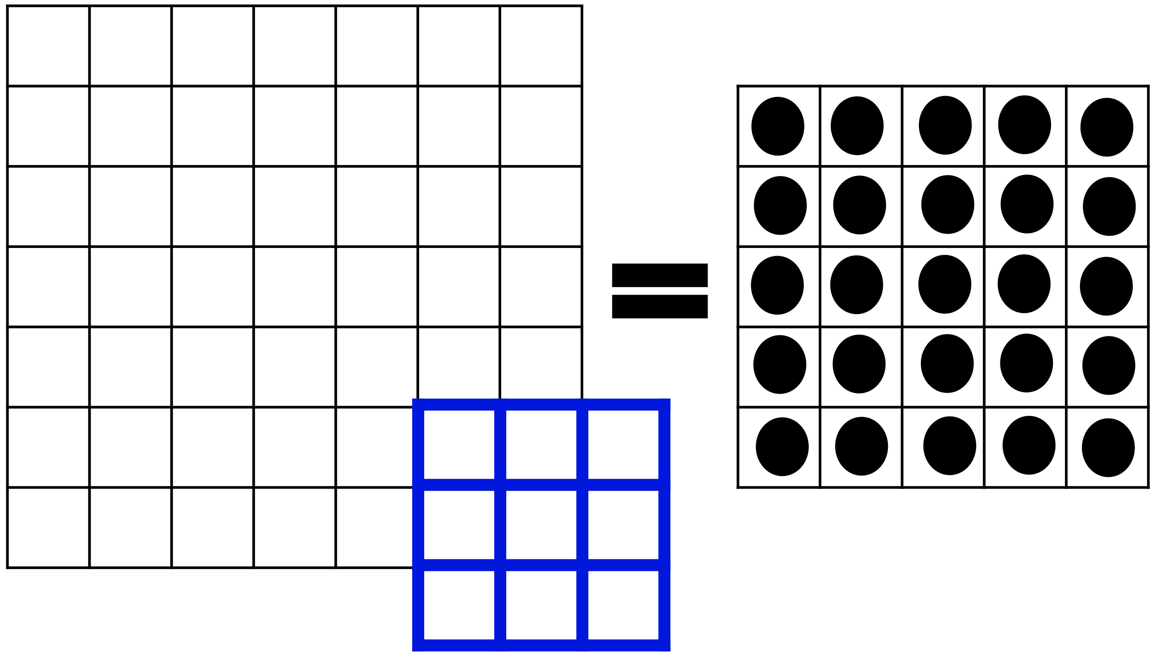
- Let us compute the dimension (W_2, H_2) of the output

Calculation of Output Dimension



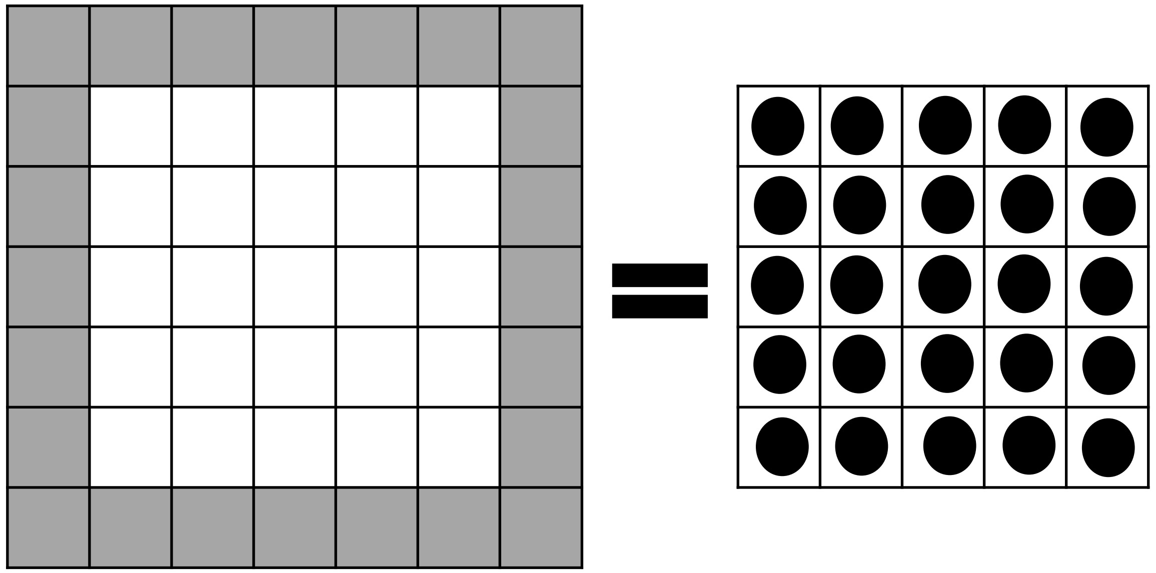
- Let us compute the dimension (W_2, H_2) of the output

Calculation of Output Dimension



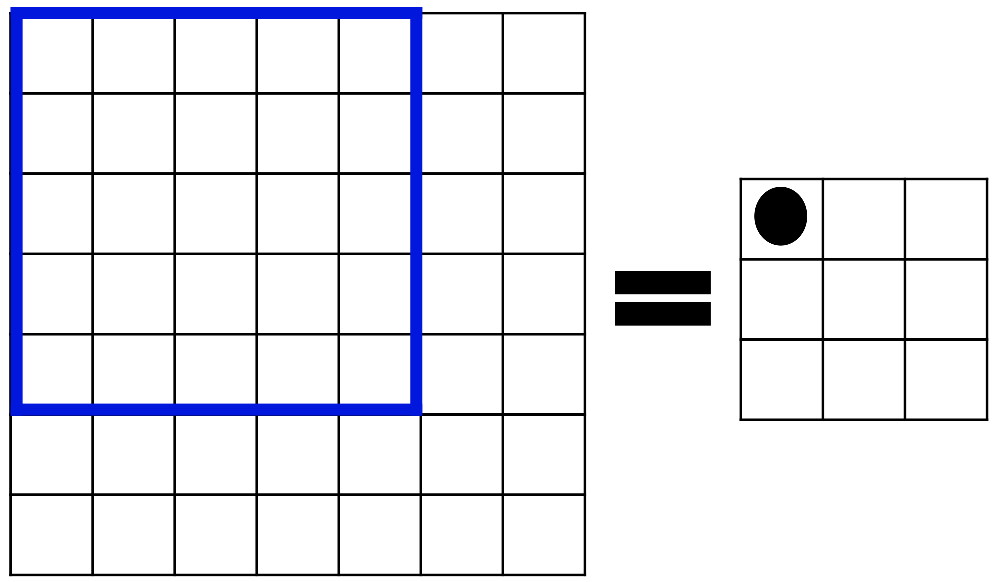
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary

Calculation of Output Dimension



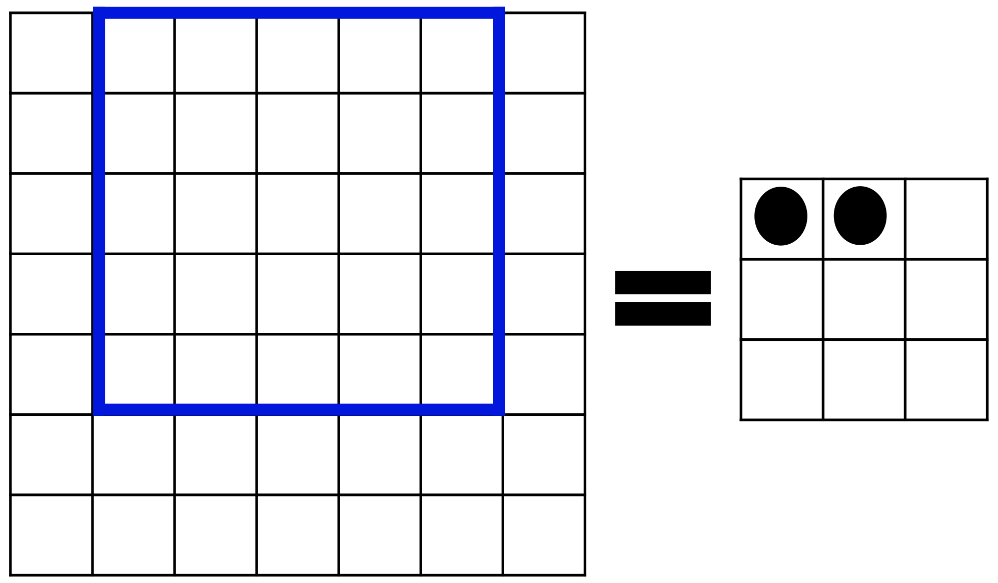
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input

Calculation of Output Dimension



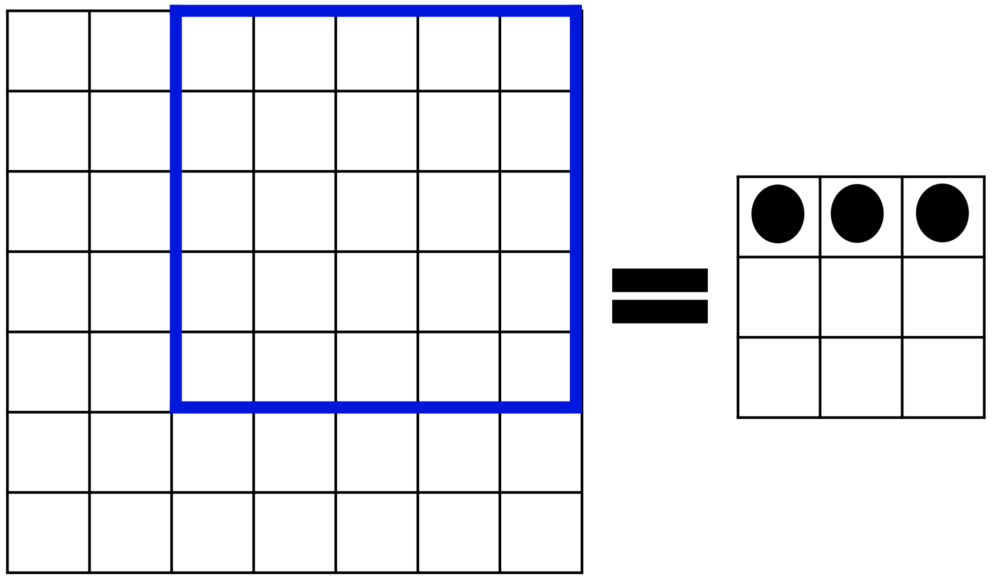
- As the size of the kernel increases, this becomes true for even more pixels
- For example, let's consider a 5 x 5 kernel

Calculation of Output Dimension



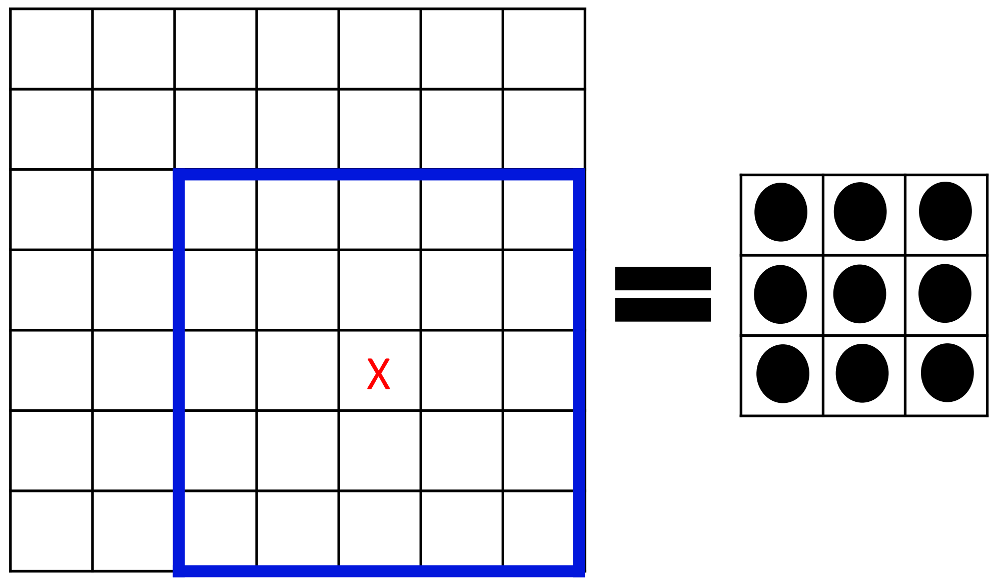
- As the size of the kernel increases, this becomes true for even more pixels
- For example, let's consider a 5 x 5 kernel

Calculation of Output Dimension



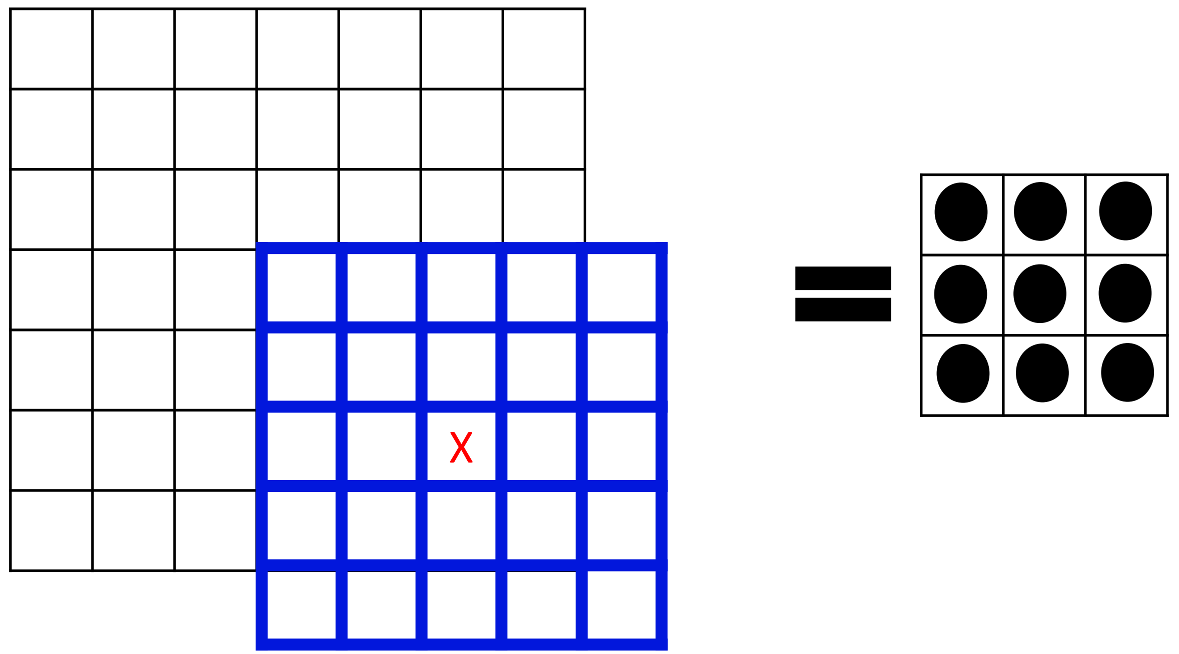
- As the size of the kernel increases, this becomes true for even more pixels
- For example, let's consider a 5 x 5 kernel

Calculation of Output Dimension



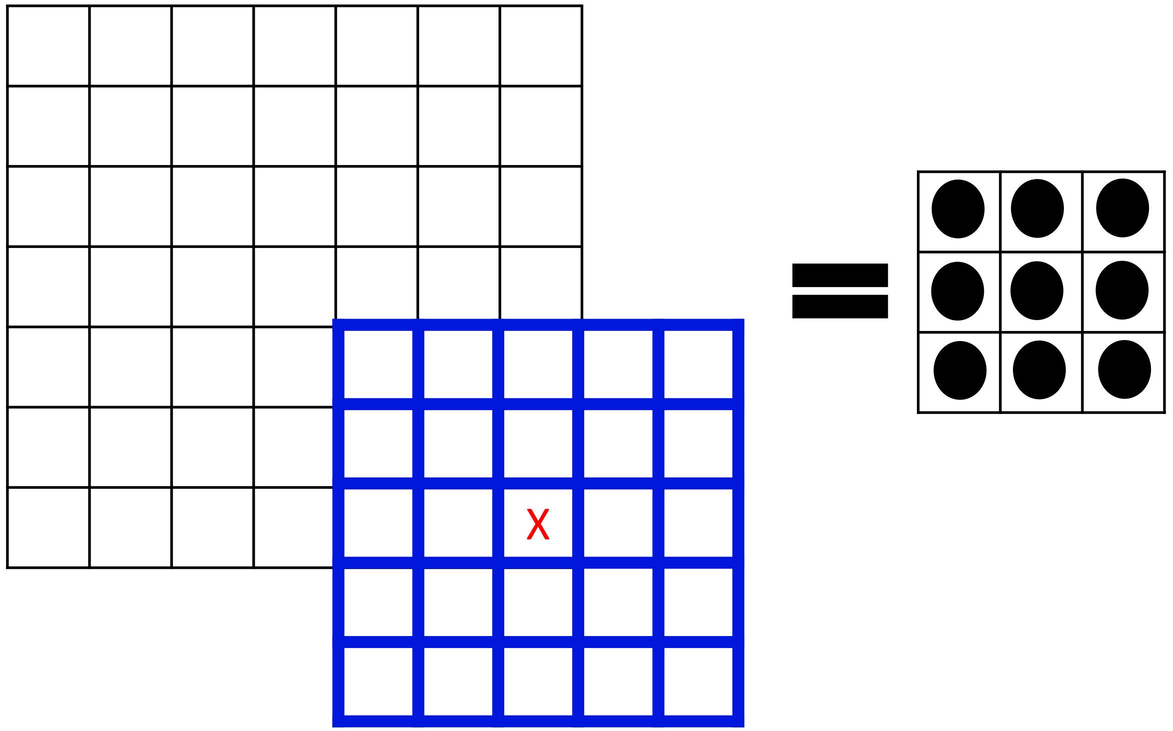
- As the size of the kernel increases, this becomes true for even more pixels
- For example, let's consider a 5 x 5 kernel

Calculation of Output Dimension



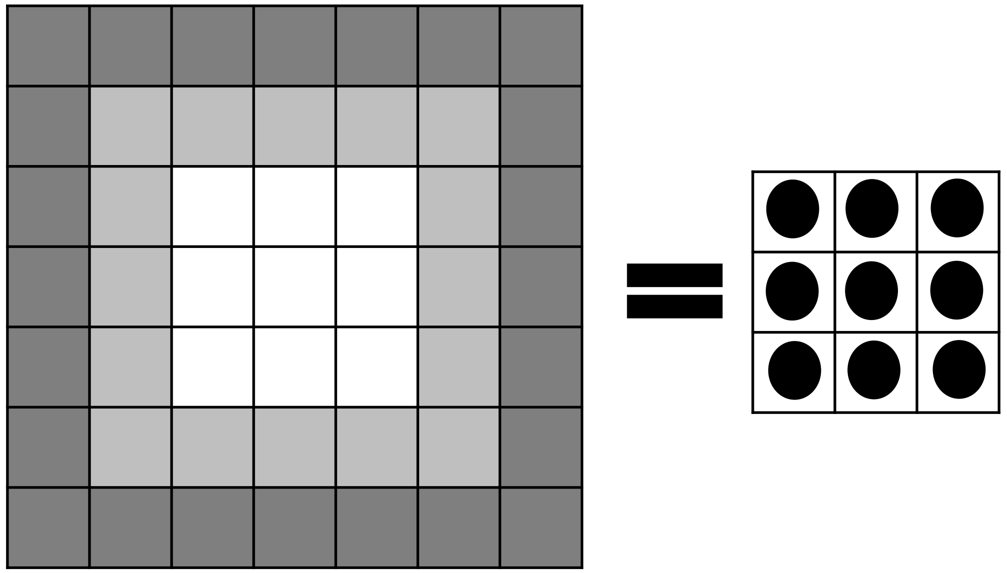
- As the size of the kernel increases, this becomes true for even more pixels
- For example, let's consider a 5 x 5 kernel

Calculation of Output Dimension



- As the size of the kernel increases, this becomes true for even more pixels
- For example, let's consider a 5 x 5 kernel

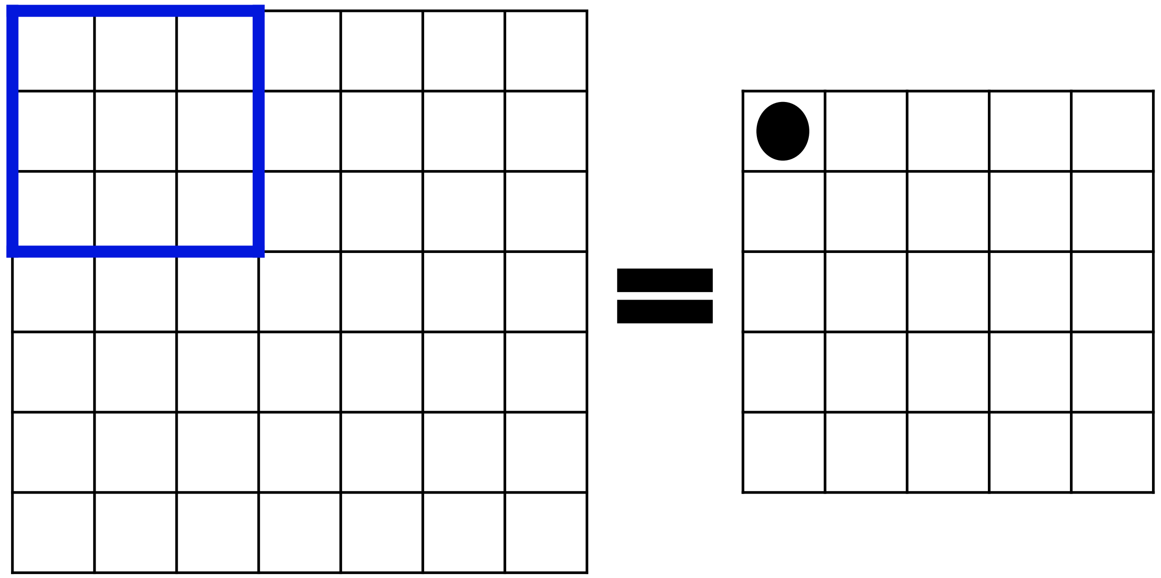
Calculation of Output Dimension



- As the size of the kernel increases, this becomes true for even more pixels
- For example, let's consider a 5 x 5 kernel
- In general;

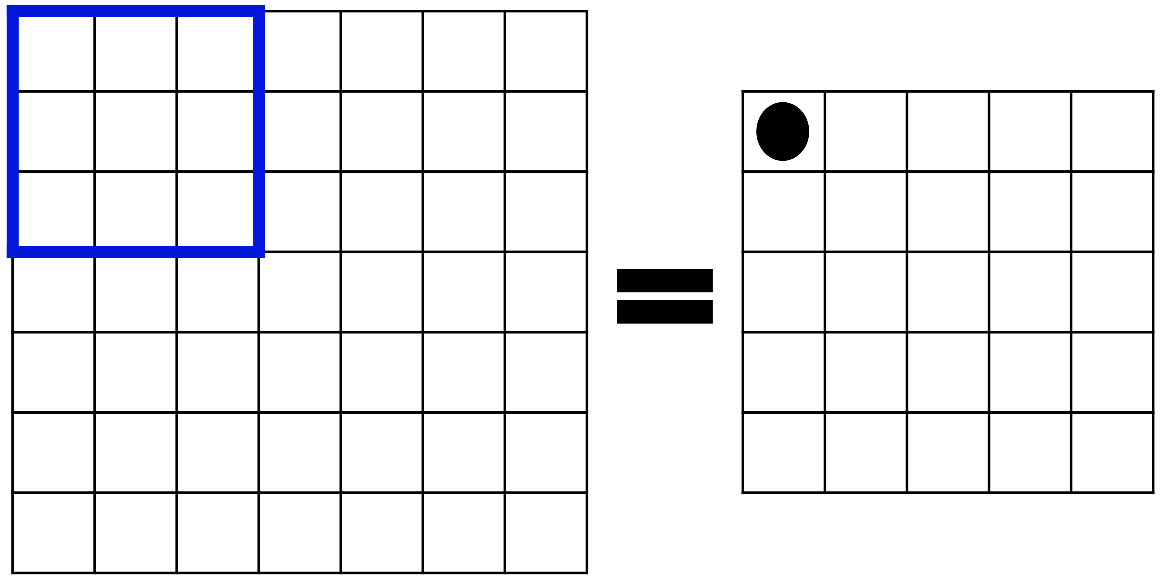
$$W_2 = W_1 - F + 1$$
$$H_2 = H_1 - F + 1$$

Padding



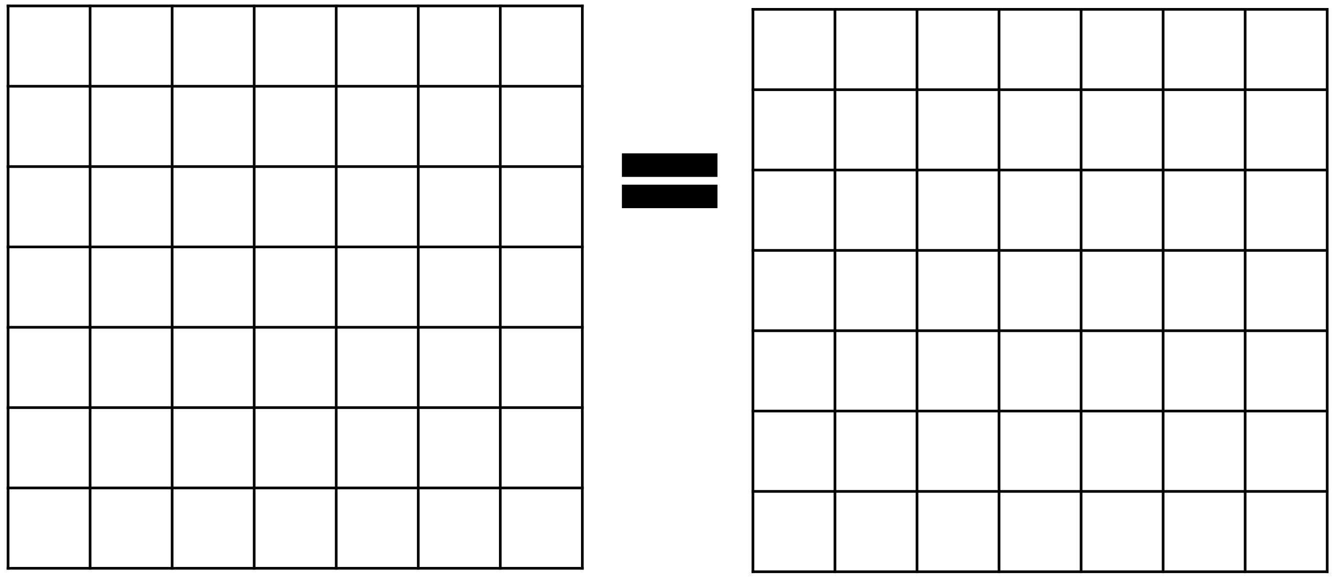
- What if we want the output to be of same size as the input?

Padding



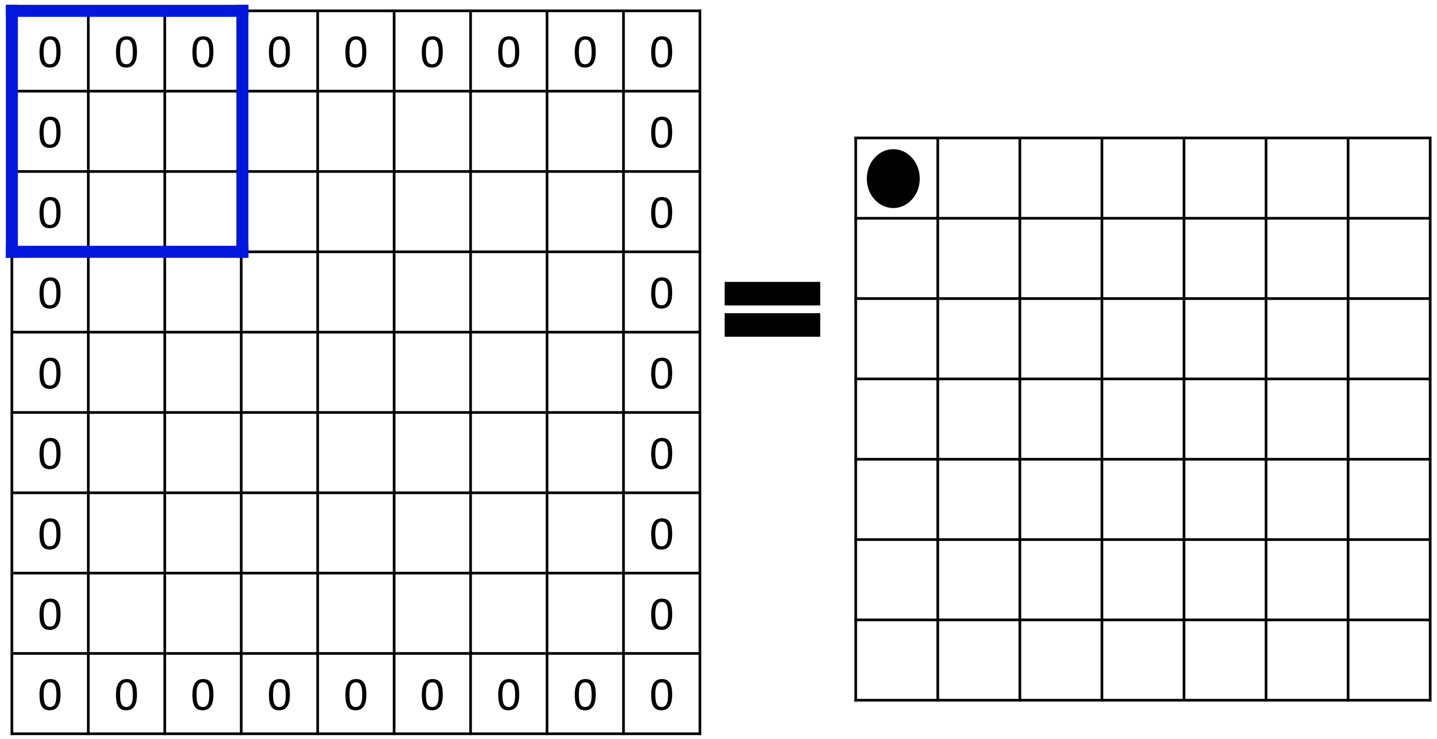
- What if we want the output to be of same size as the input?
- We can use something known as **padding**

Padding



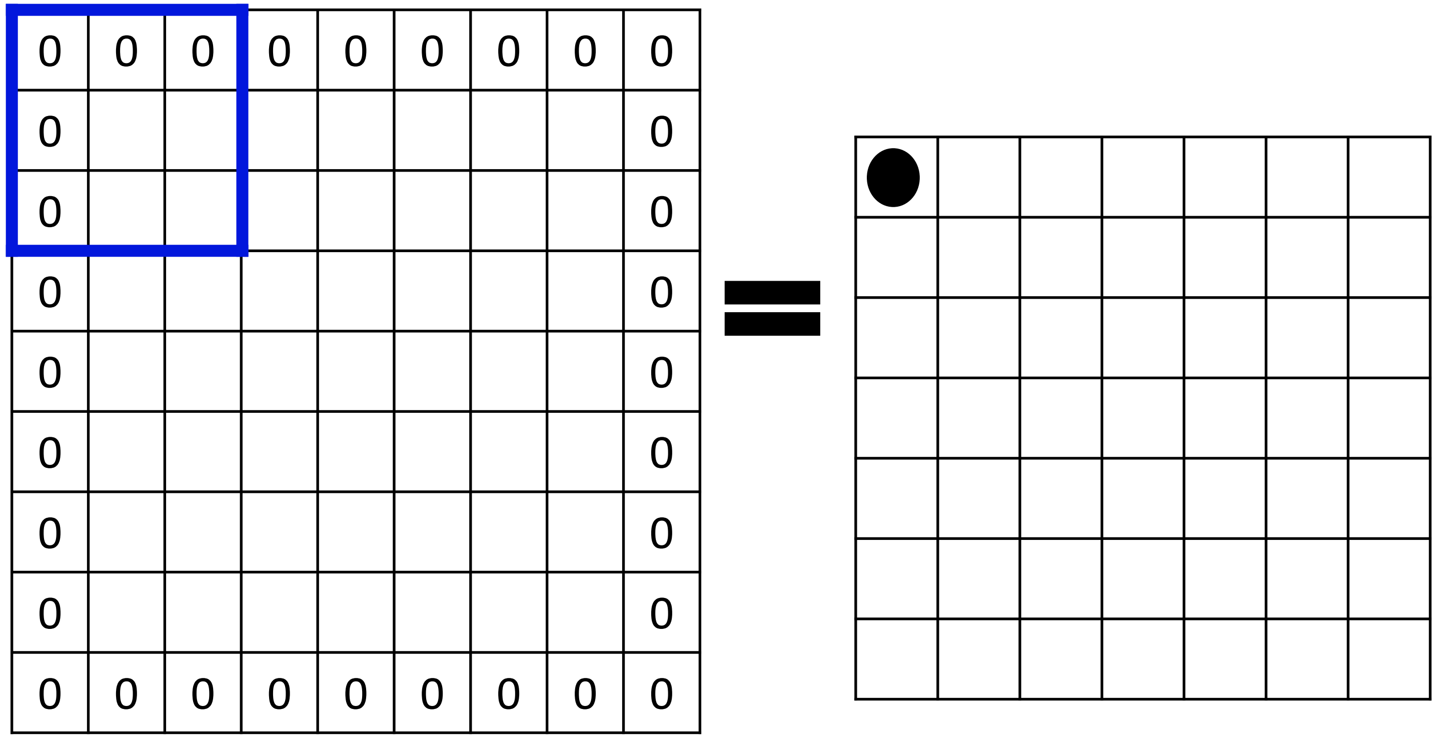
- Pad the inputs with appropriate number of ‘**0**’ inputs so that you can now apply the kernel at the corners

Padding



- Let us use pad P = 1 with a 3 x 3 kernel
- This means we will add one row of ‘**0**’ inputs at the top and bottom, and add one column of ‘0’ inputs at the left and right.

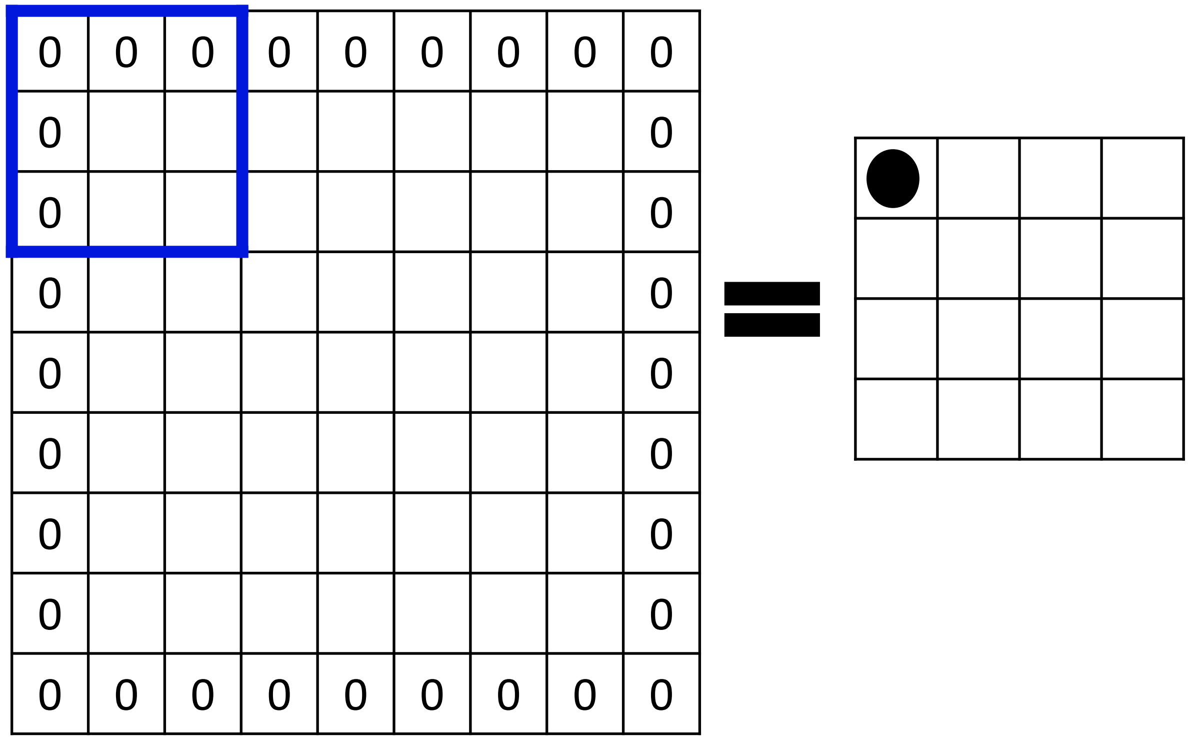
Padding



- Now we have;

$$W_2 = W_1 - F + 2P + 1$$
$$H_2 = H_1 - F + 2P + 1$$

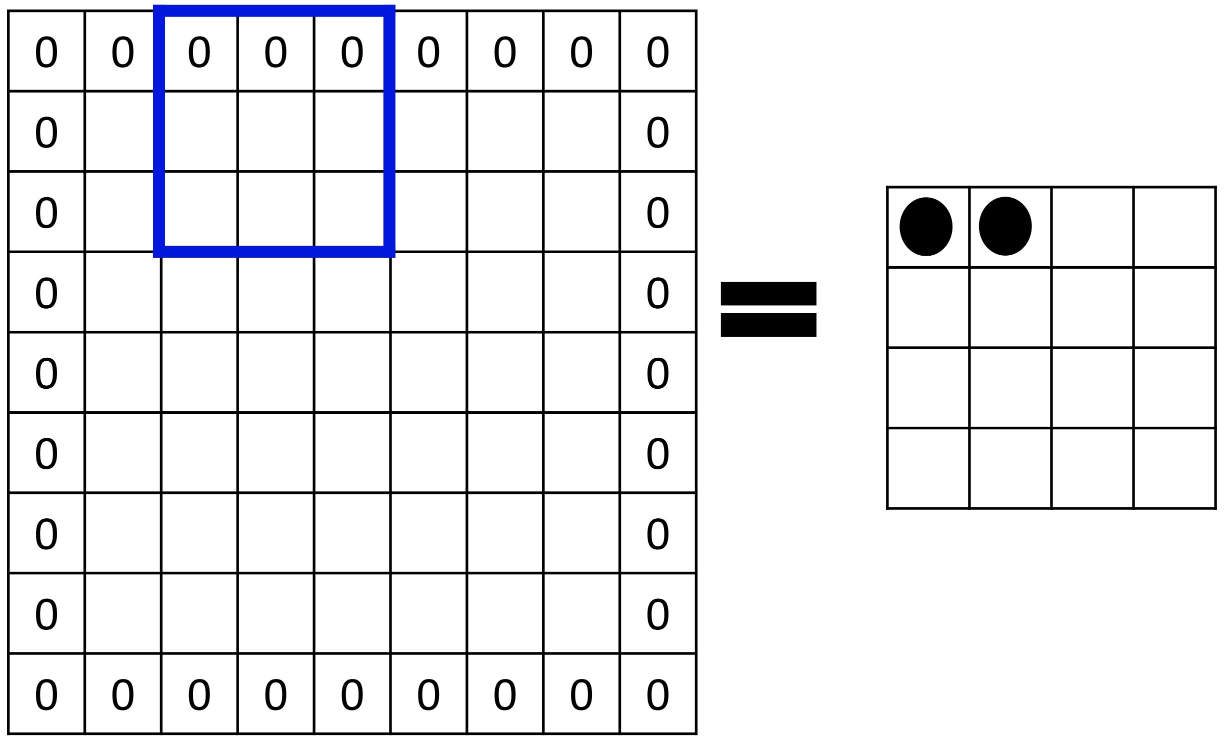
Stride



- What does the stride S do?

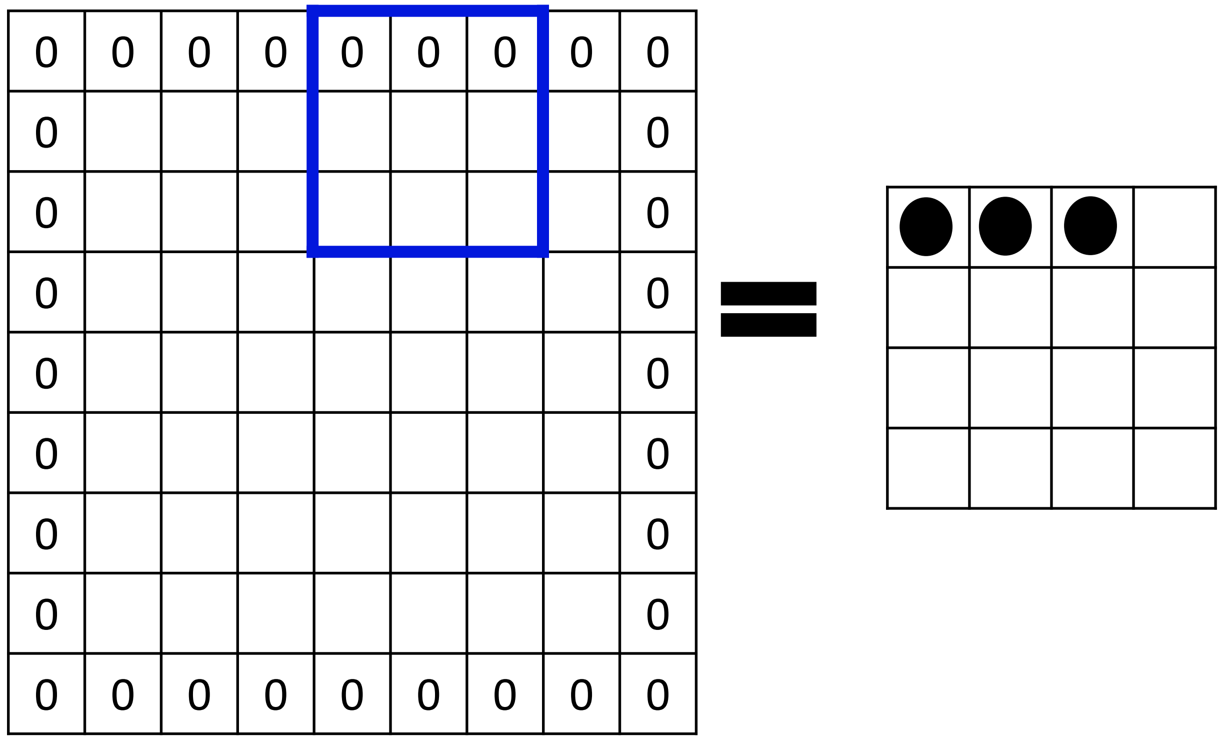
• **Stride**: This parameter determines how many pixels the kernel moves after each operation.
• A larger stride results in a smaller output size.

Stride



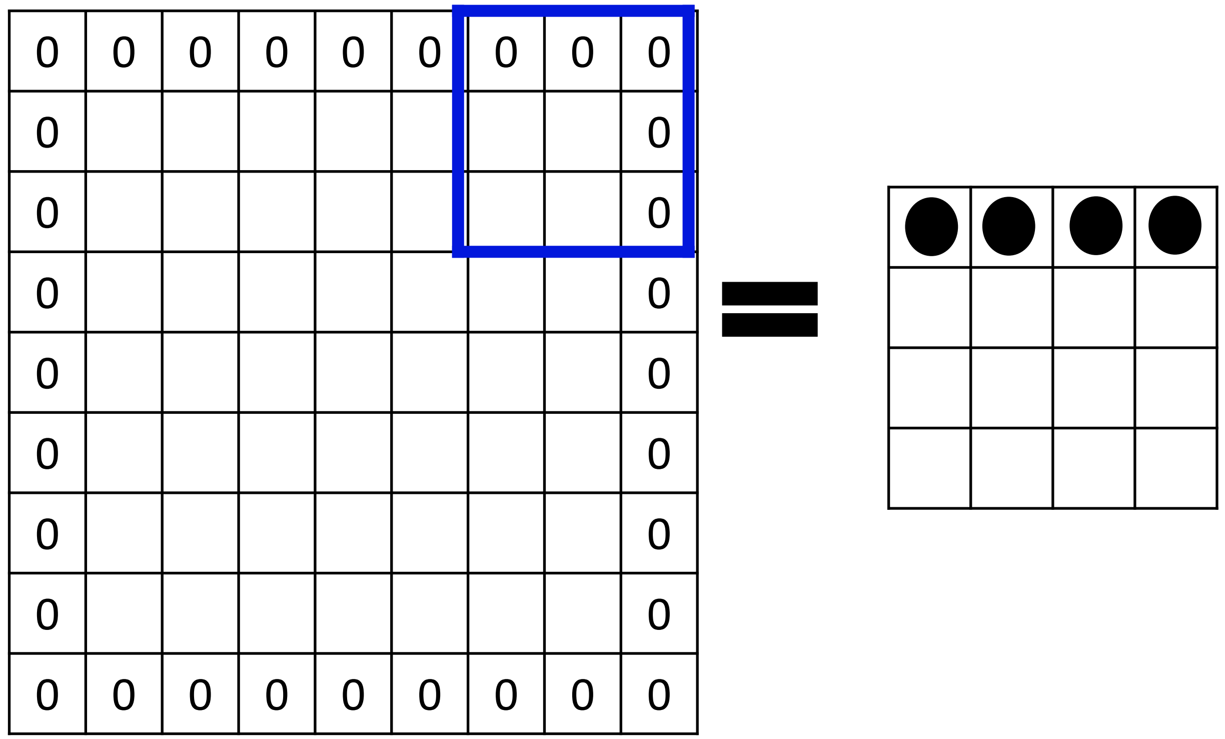
- What does the stride S do?
- S=2 means the kernel moves 2 pixels after each operation.
- It defines the intervals at which the filter is applied (here S = 2)

Stride



- What does the stride S do?
- It defines the intervals at which the filter is applied (here S = 2)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

Stride



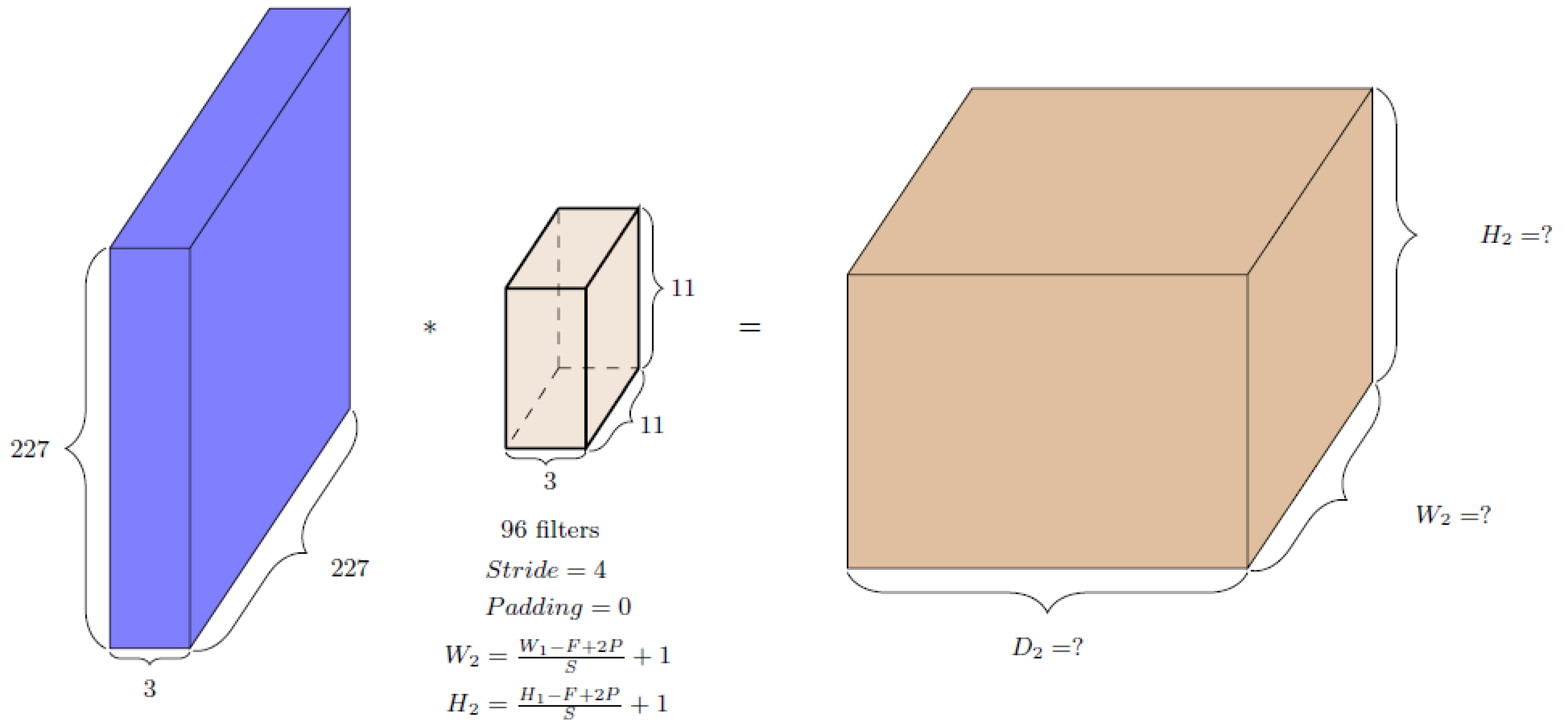
- Now we have;

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

- Since the output size must be an integer, we typically floor the result.

Numerical Example



96 filters

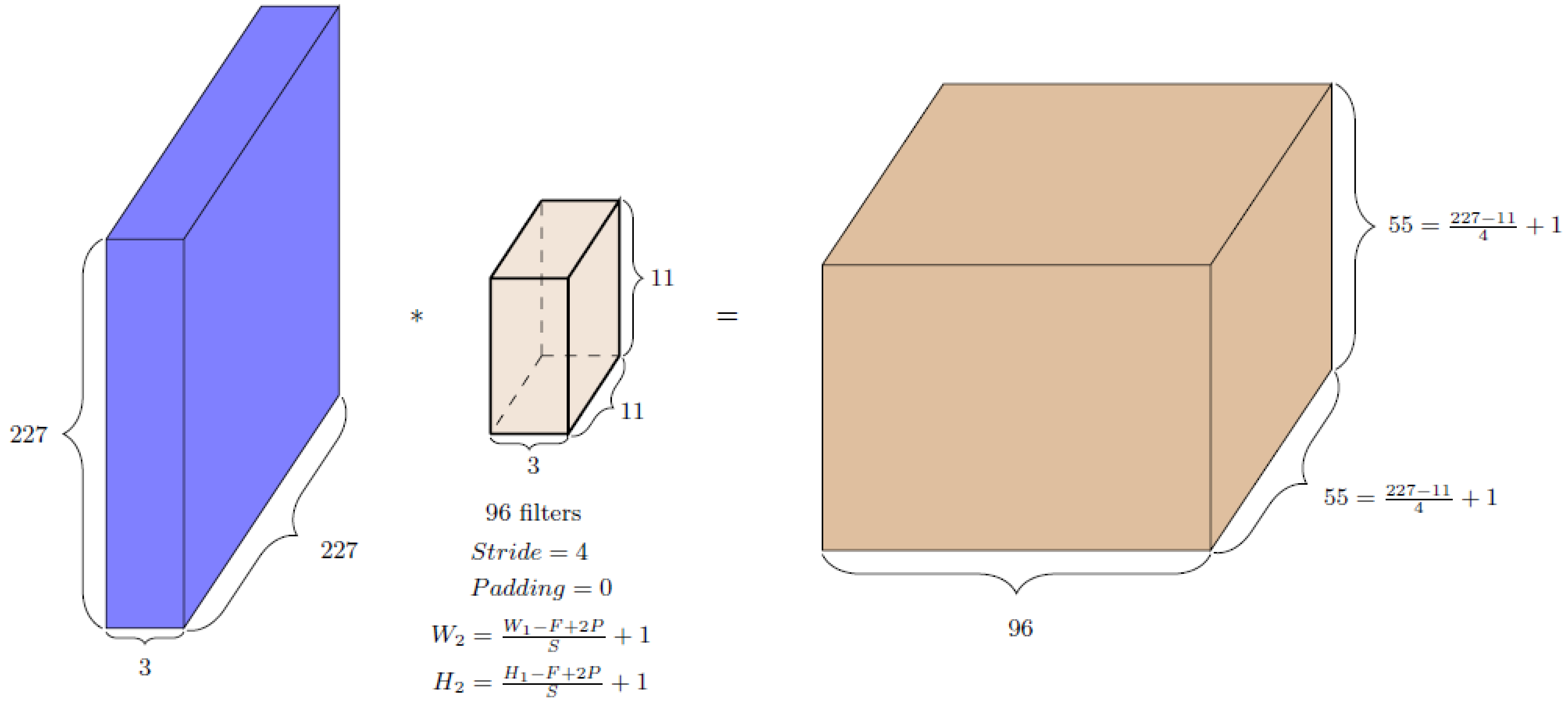
$Stride = 4$

$Padding = 0$

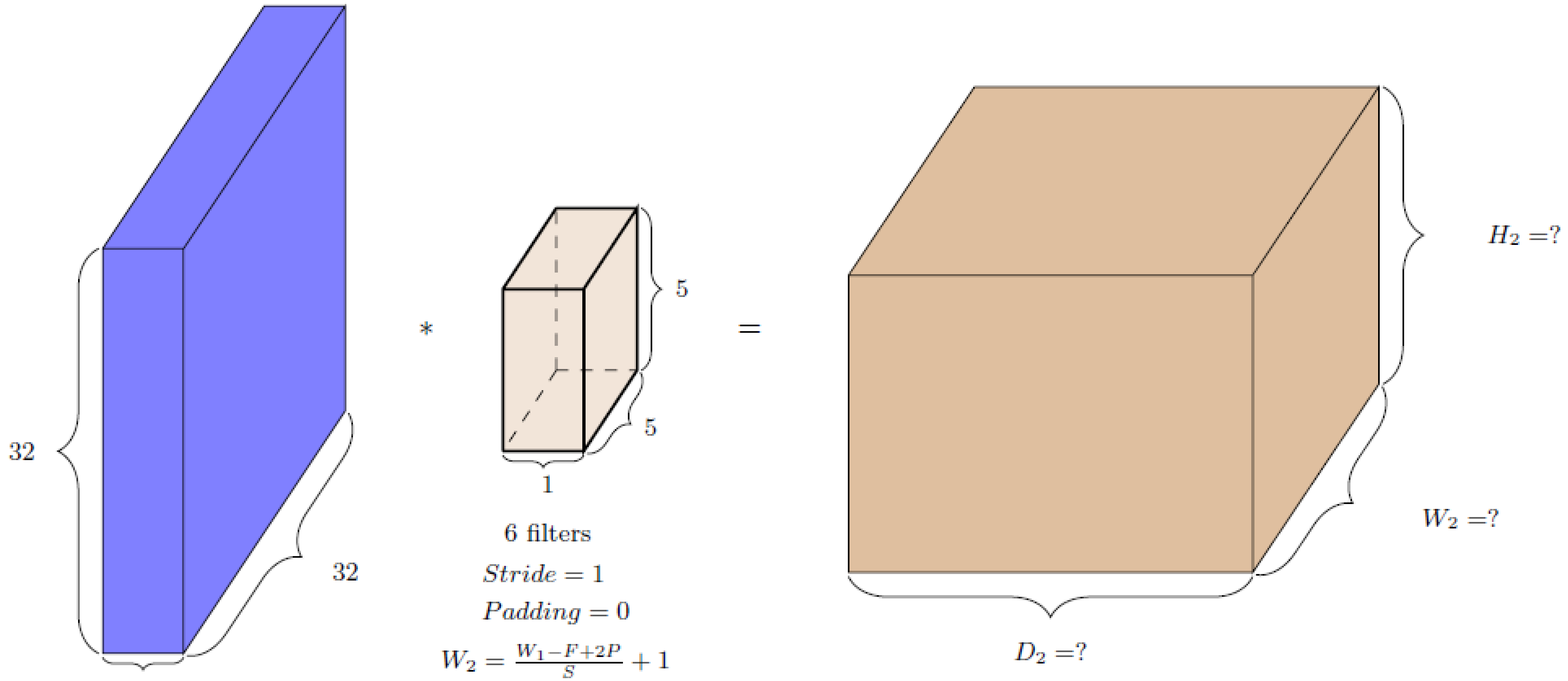
$W_2 = \frac{W_1 - F + 2P}{S} + 1$

$H_2 = \frac{H_1 - F + 2P}{S} + 1$

Numerical Example


227
227
3
*
11
11
3
96 filters
$Stride = 4$
$Padding = 0$
$W_2 = \frac{W_1 - F + 2P}{S} + 1$
$H_2 = \frac{H_1 - F + 2P}{S} + 1$
=
$55 = \frac{227-11}{4} + 1$
$55 = \frac{227-11}{4} + 1$
96

Numerical Example



6 filters

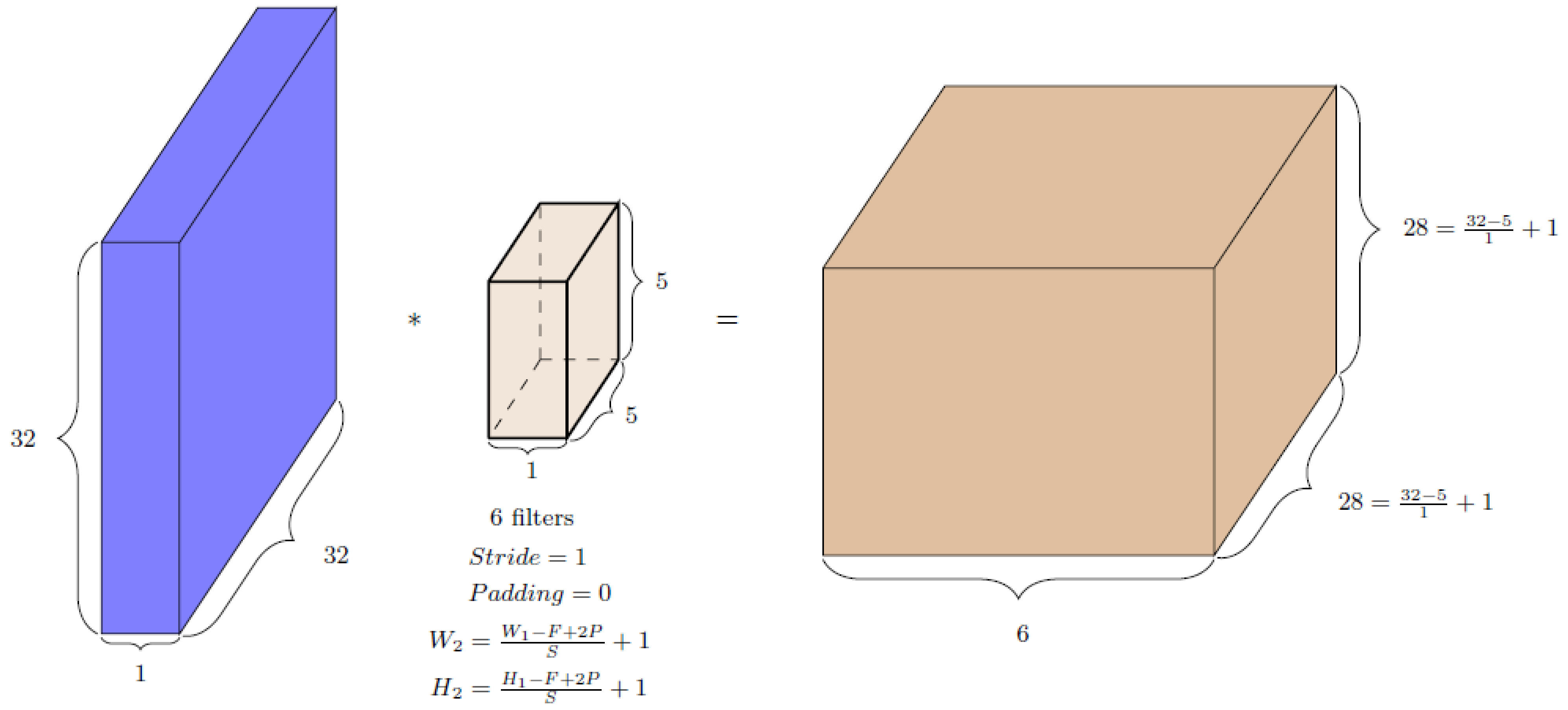
$Stride = 1$

$Padding = 0$

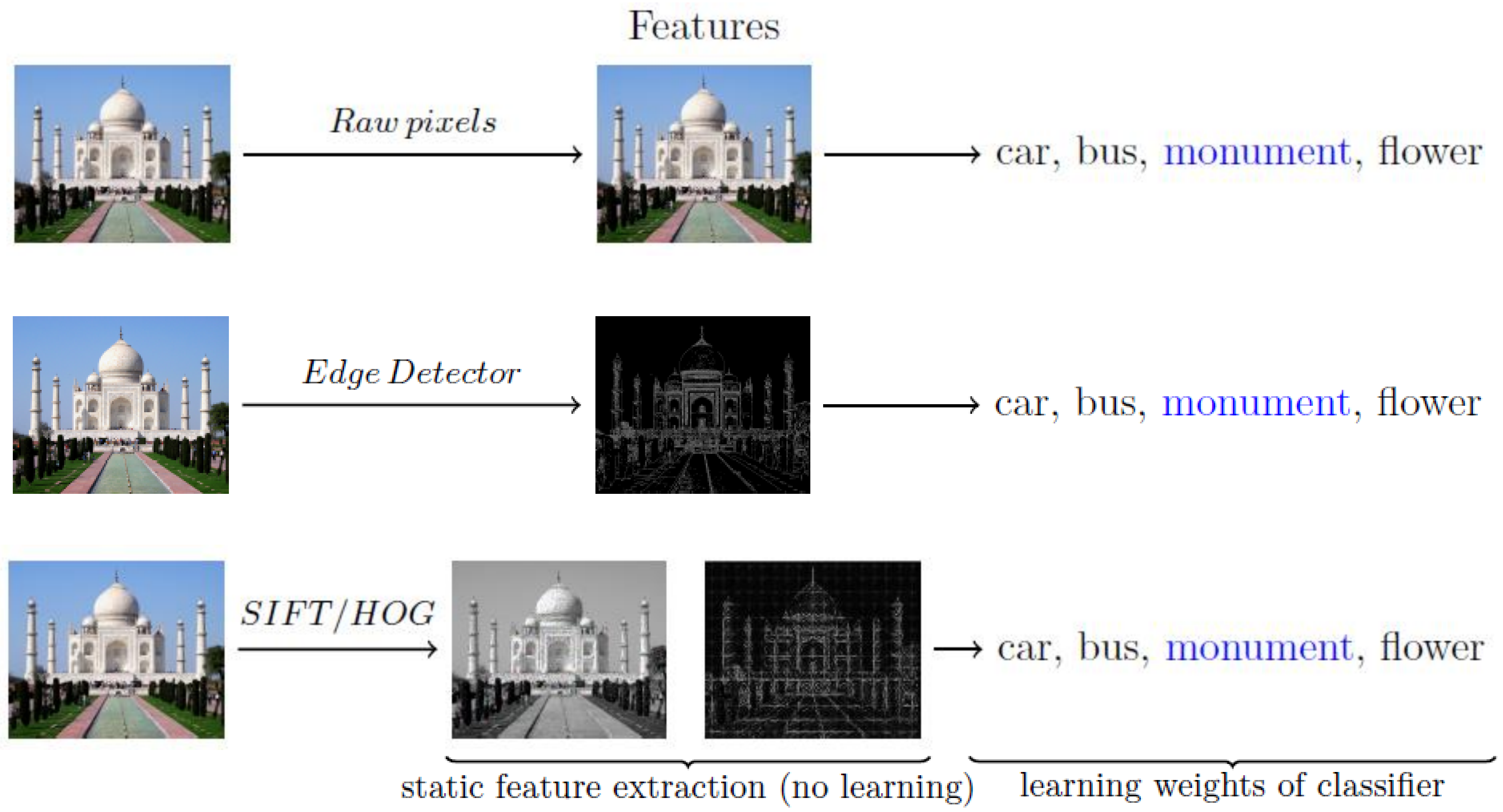
$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

Numerical Example


32
32
1
*
5
5
1
6 filters
$Stride = 1$
$Padding = 0$
$W_2 = \frac{W_1 - F + 2P}{S} + 1$
$H_2 = \frac{H_1 - F + 2P}{S} + 1$
=
$28 = \frac{32-5}{1} + 1$
$28 = \frac{32-5}{1} + 1$
6

Convolutional Neural Networks


Features
Raw pixels
car, bus, monument, flower
Edge Detector
car, bus, monument, flower
SIFT/HOG
car, bus, monument, flower
static feature extraction (no learning)
learning weights of classifier

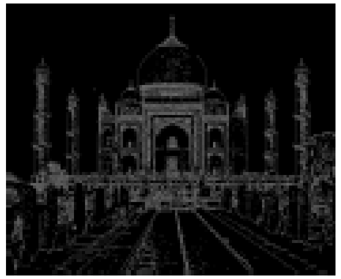
Convolutional Neural Networks

Input

Features

Classifier

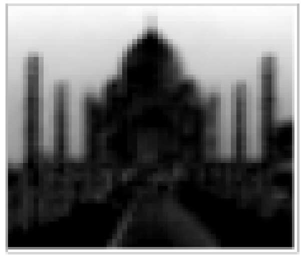
$\longrightarrow$



$\longrightarrow$ car, bus, monument, flower

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & -8 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$

$\longrightarrow$

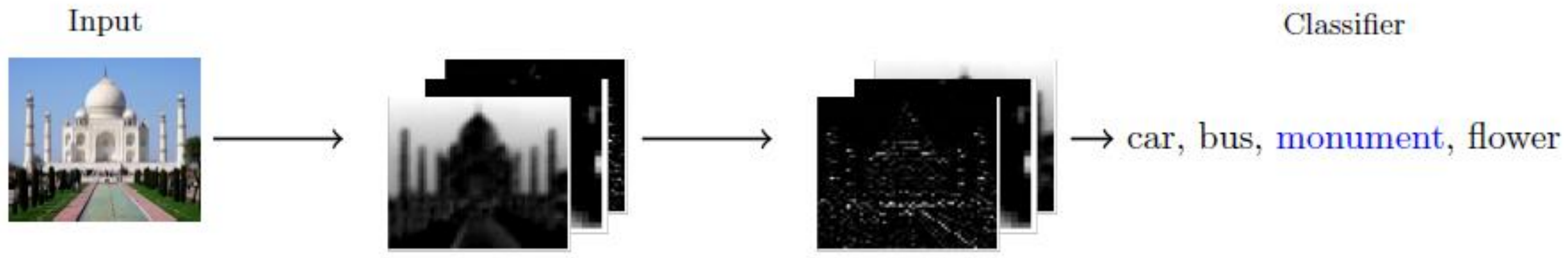


$\longrightarrow$ car, bus, monument, flower

$$\begin{matrix} -1.21[illegible]e\text{-}03 & 3.23[illegible]e\text{-}03 & \cdots & \cdots & -3.06[illegible]e\text{-}02 \\ -1.52[illegible]e\text{-}03 & 2.36[illegible]e\text{-}03 & \cdots & \cdots & -1.19[illegible]e\text{-}02 \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ -8.35[illegible]e\text{-}04 & -5.14[illegible]e\text{-}03 & \cdots & \cdots & -9.00[illegible]e\text{-}03 \end{matrix}$$

$\longleftarrow$ Learn these weights

Convolutional Neural Networks



- We can learn multiple layers of meaningful kernels / filters in addition to learning the weights of the classier
- Simply by treating these kernels as parameters and learning them in addition to the weights of the classfier (using back propagation)

Convolutional Neural Networks

- **Random Initialization**: When a CNN model is first created, the kernel weights are typically initialized randomly, often with small values.
- These random kernels will not necessarily be useful initially, but they will evolve during training. They are **learned through training** based on the data via backpropagation.
- Common initialization techniques include **Xavier initialization** or **He initialization**, designed to ensure that the starting weights allow the network to learn effectively.
- **Why Random?** Starting with random kernels prevents the network from having any bias toward specific patterns and ensures that it can learn a wide variety of features from the data.
- Predefined kernels are generally **not used** in deep learning, as learned kernels are more effective for complex tasks.

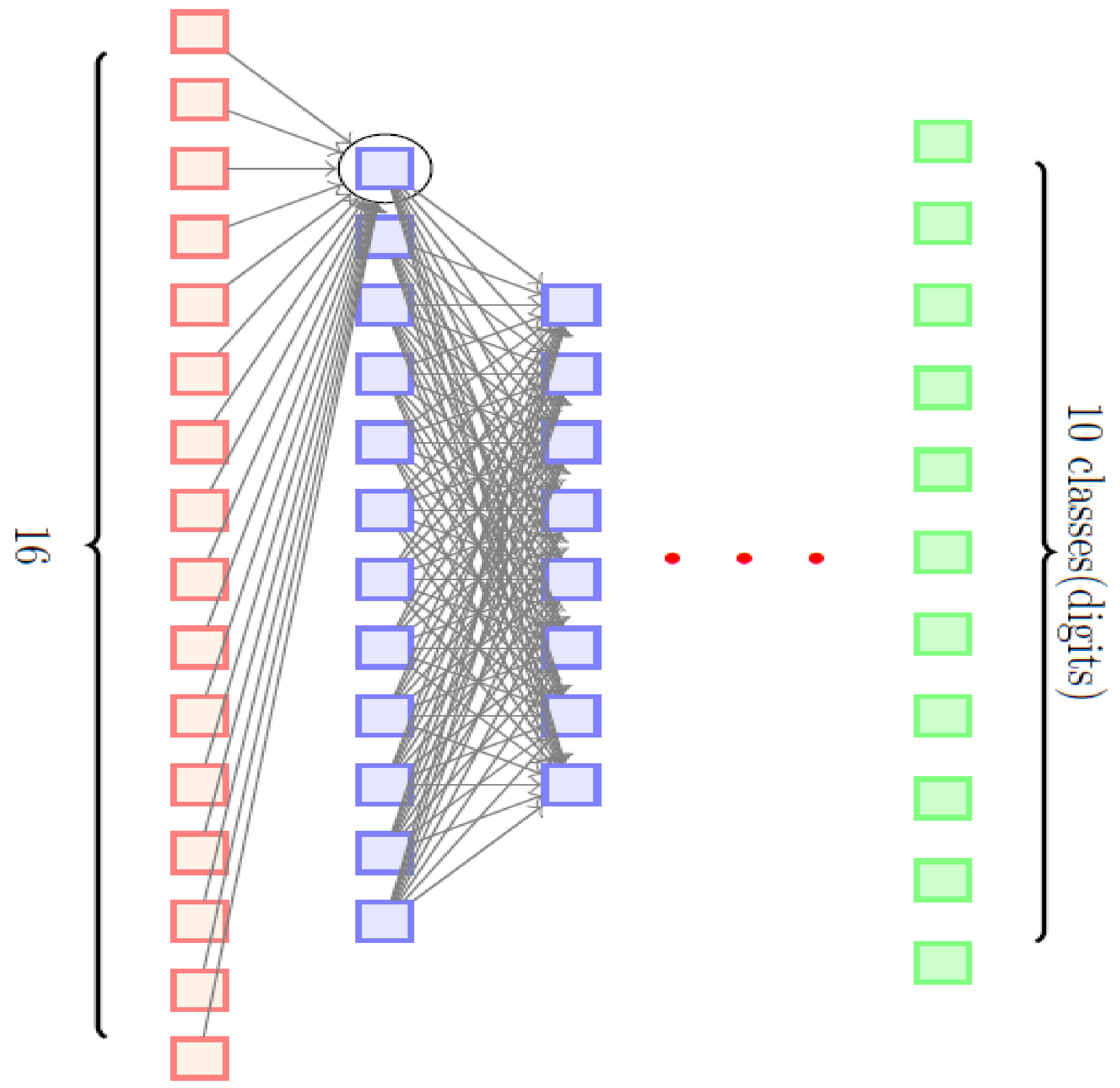
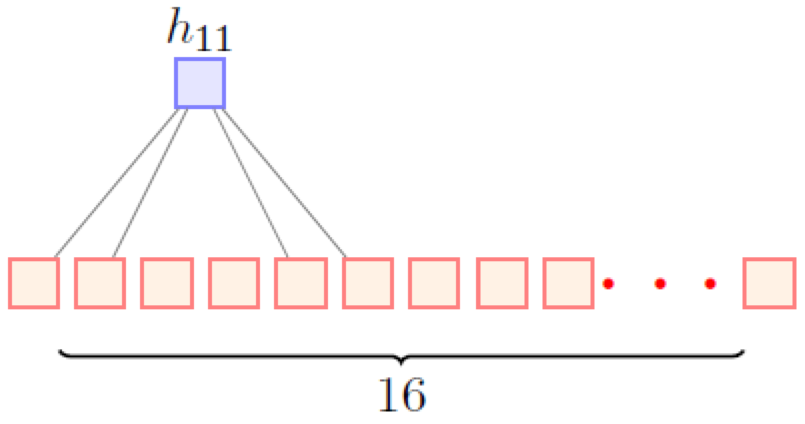
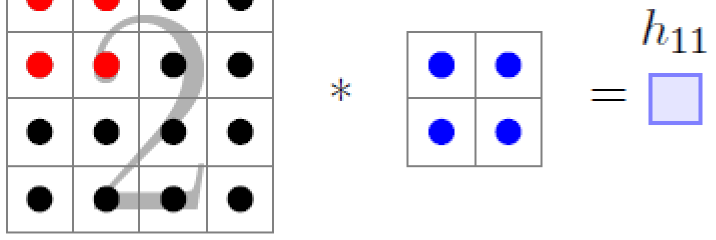


Image classification

- This is what a regular feed-forward neural network will look like
- There are many dense connections here
- For example all the 16 input neurons are contributing to the computation of h_{11}
- Contrast this to what happens in the case of convolution

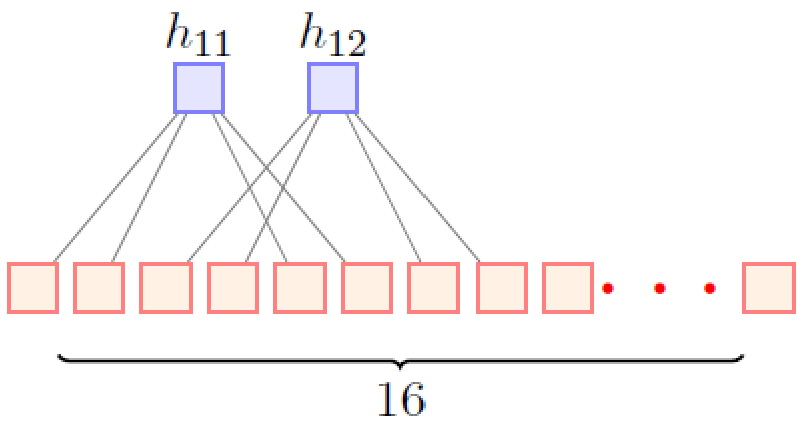
Sparsity of connections

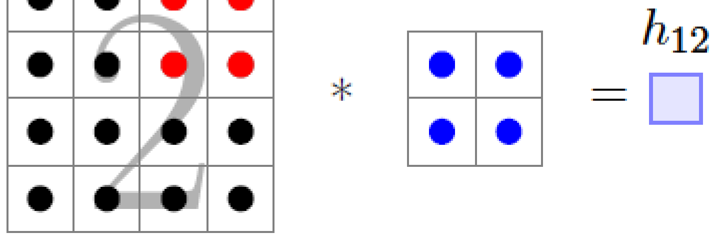




- The image of size 4×4 is considered as a vector [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
- Only a few local neurons participate in the computation of h_{11}
- For example, only pixels 1, 2, 5, 6 contribute to h_{11}

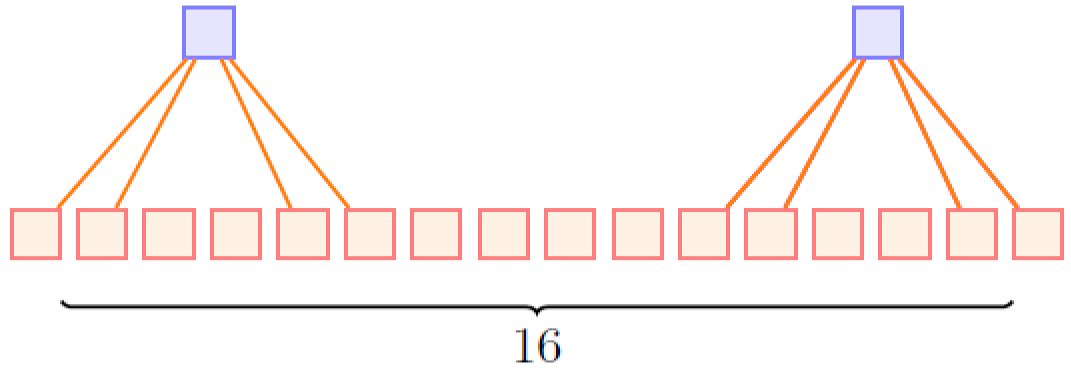
Sparsity of connections

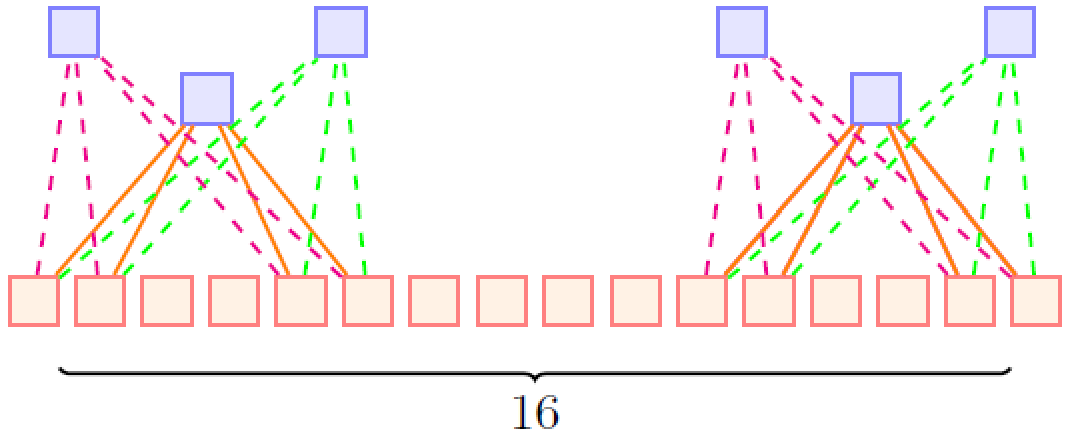




- Similarly, only pixels 3, 4, 7, 8 contribute to h_{12}
- The connections are much sparser
- We are taking advantage of the structure of the image (interactions between neighboring pixels are more interesting)
- This **sparse connectivity** reduces the number of parameters in the model

Weight sharing





- We are not learning a single kernel
- We can have many such kernels but the kernels will be shared by all locations in the image
- This is called “**weight sharing**"

Advantages of sparsity of connection and weight sharing in CNN

- CNNs offer several advantages over traditional ANNs due to their use of sparsity of connection and weight sharing.

1. **Reduced Parameters**: In CNNs, weight sharing means that the same filter (or kernel) is applied across different regions of the input. This significantly reduces the number of parameters compared to fully connected layers in ANNs, which require a unique weight for every connection.
2. **Computational Efficiency**: Because of the reduced number of parameters, CNNs require less memory and computational resources, making them faster to train and easier to deploy, especially for large datasets and high-dimensional inputs like images.
3. **Translation Invariance**: The sparse connections allow CNNs to learn features that are translation invariant. This means that the model can recognize patterns regardless of their position in the input, which is particularly useful in image processing.

Advantages Contd..

4. **Hierarchical Feature Learning**: CNNs can learn hierarchical representations of data, where lower layers capture simple features (like edges) and higher layers capture more complex patterns (like shapes and objects). This hierarchy is facilitated by the localized connections and weight sharing.
5. **Better Generalization**: The reduction in the number of parameters and the emphasis on shared features can lead to better generalization on unseen data, as the model is less likely to overfit compared to a fully connected ANN with many parameters.
6. **Robustness to Noise and Variability**: The ability to learn from local patterns and use shared weights makes CNNs more robust to noise and variations in the input data, such as changes in lighting, scale, and rotation.

Overall, these advantages make CNNs particularly well-suited for tasks like image recognition, object detection, and other applications involving spatial data.

Pooling Layer

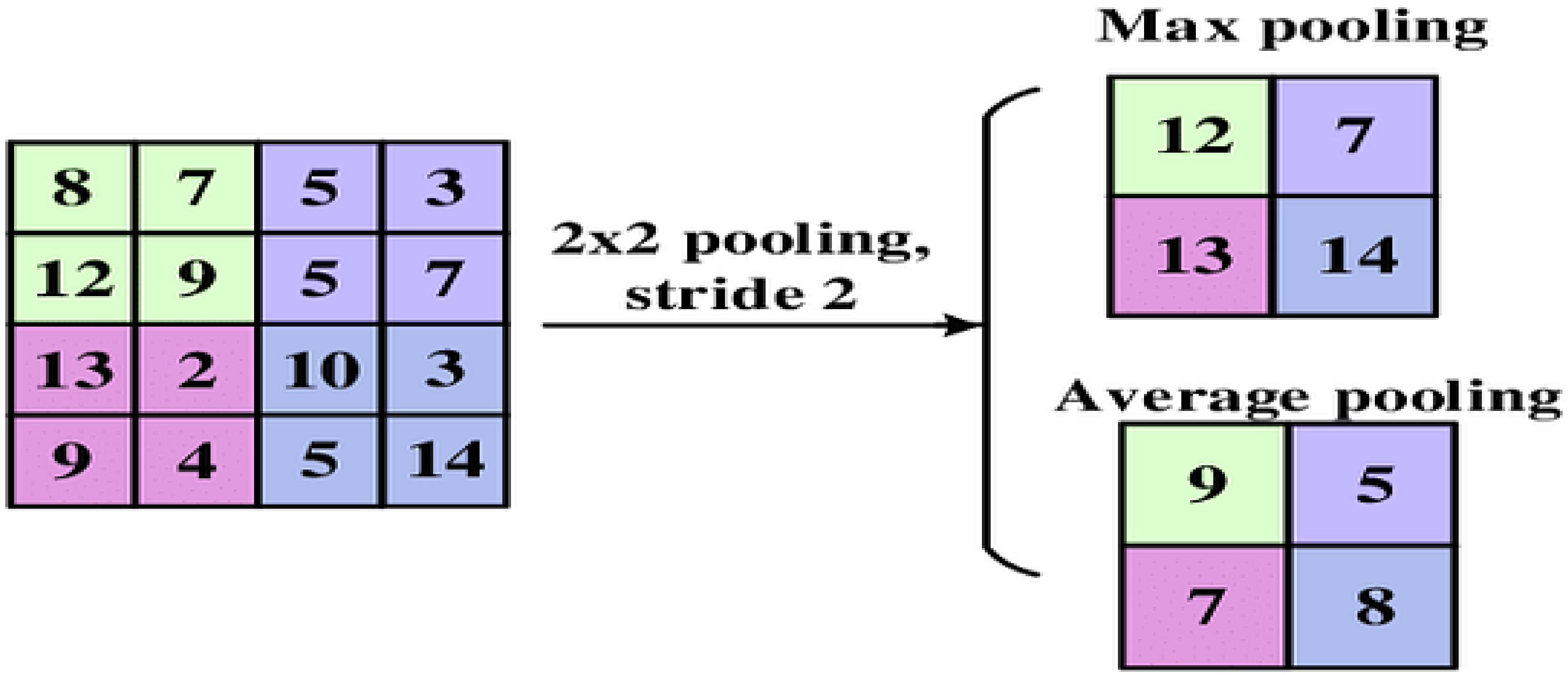
Pooling Layer: Reduce the spatial dimensions of the feature maps, which decreases the number of parameters and computation in the network.

- **Types:**

(i) **Max Pooling:** Takes the maximum value from a defined window. It retains the most prominent features, making it effective for detecting edges and textures.

(ii) **Average Pooling:** Takes the average value from a defined window. It provides a smoother feature map and can be less sensitive to noise than max pooling. It is sometimes in cases where we want to retain more contextual information.

Pooling



8
7
5
3
12
9
5
7
13
2
10
3
9
4
5
14
2x2 pooling, stride 2
Max pooling
12
7
13
14
Average pooling
9
5
7
8

Pooling

(iii) Global Average Pooling

- It takes the average of the entire feature map, producing a single output for each feature map. It is helpful for reducing the size of feature maps and can be effective in classification tasks. It is generally used in the final layers of CNN architectures before the output layer.

(iv) Global Max Pooling

- It takes the maximum value from the entire feature map. Similar to global average pooling it is helpful for reducing the size of feature maps but it retains the most significant feature. It is also used before the output layer, especially when spatial resolution is less important.

Pooling

- Pooling operations like max pooling and average pooling generally reduce spatial resolution.
- However, the specific impact on spatial resolution depends on how the pooling is configured (e.g., window size and stride).
- **Pooling Parameters**

(i) Window Size: The size of the pooling filter (e.g., 2x2, 3x3).

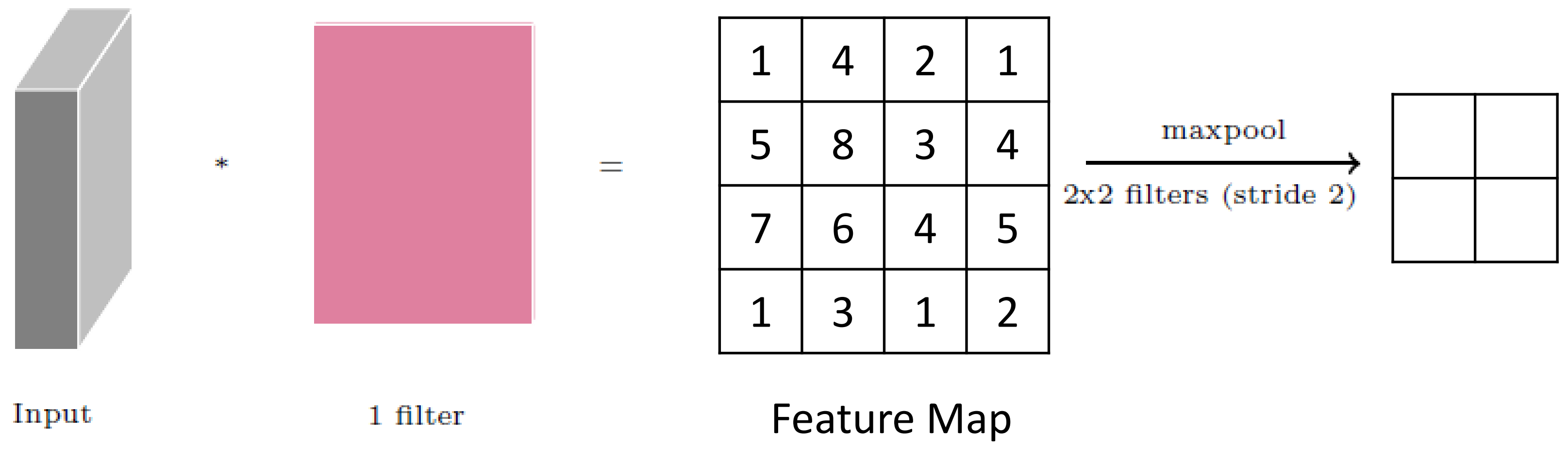
(ii) Stride: The number of pixels the filter moves after each operation. A larger stride reduces the output size more significantly.

(iii) Padding: Adding extra pixels around the input feature map before pooling. Typically, pooling layers don’t use padding, but it can be useful in certain architectures.

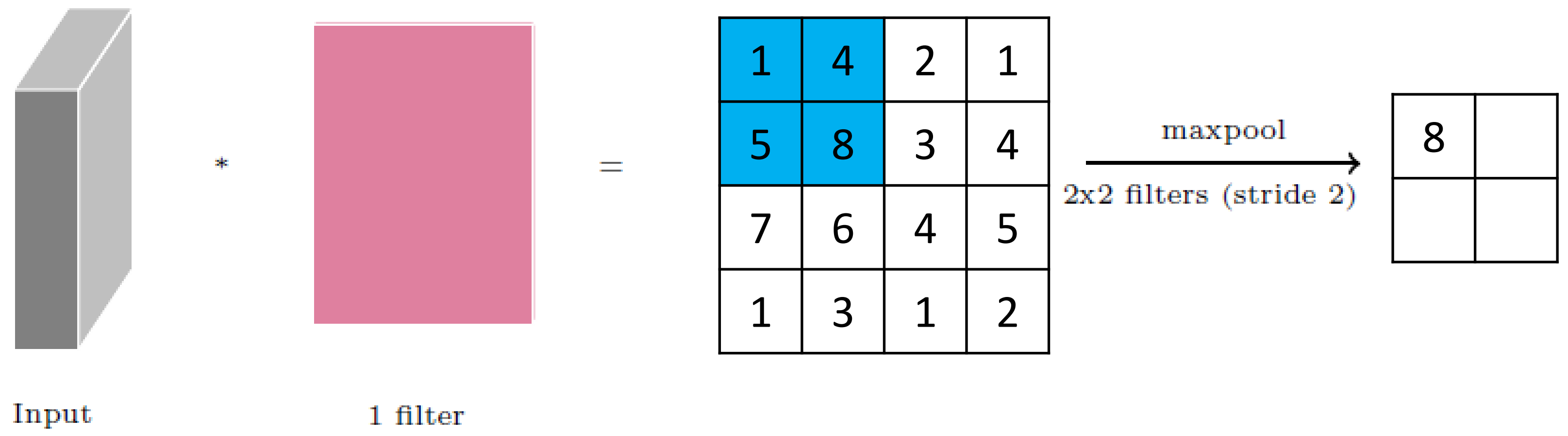
Pooling

- Instead of pooling, some architectures use convolutions with strides greater than one to achieve downsampling while learning features.
- Pooling can lead to a loss of spatial information, which might be critical for some tasks (e.g., image segmentation).
- The choice between max pooling, average pooling, or other pooling techniques often depends on the specific task and desired outcomes.

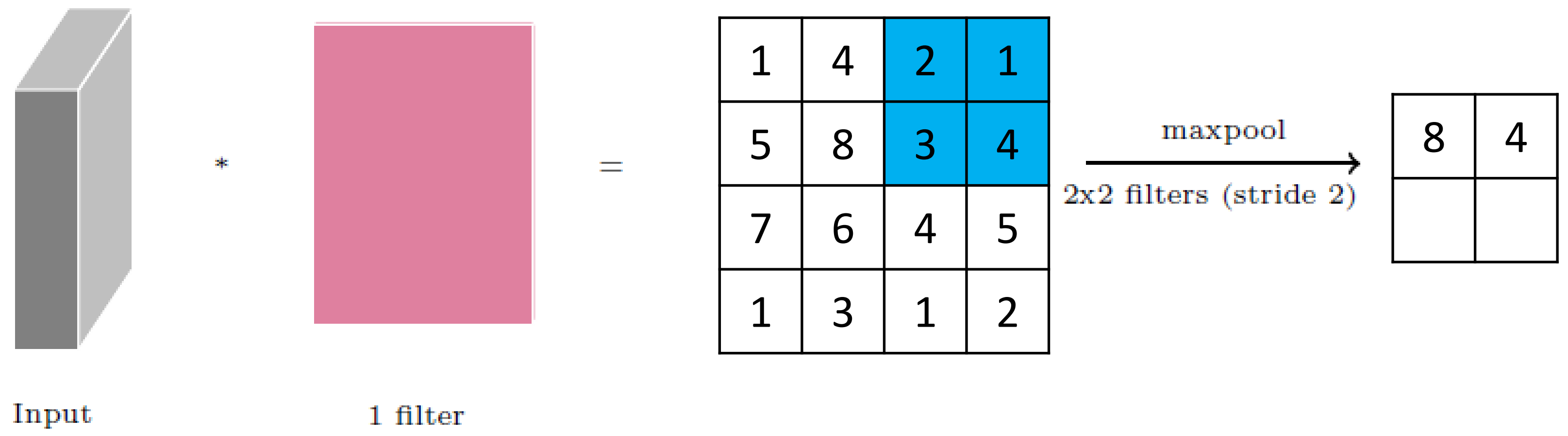
Pooling layer



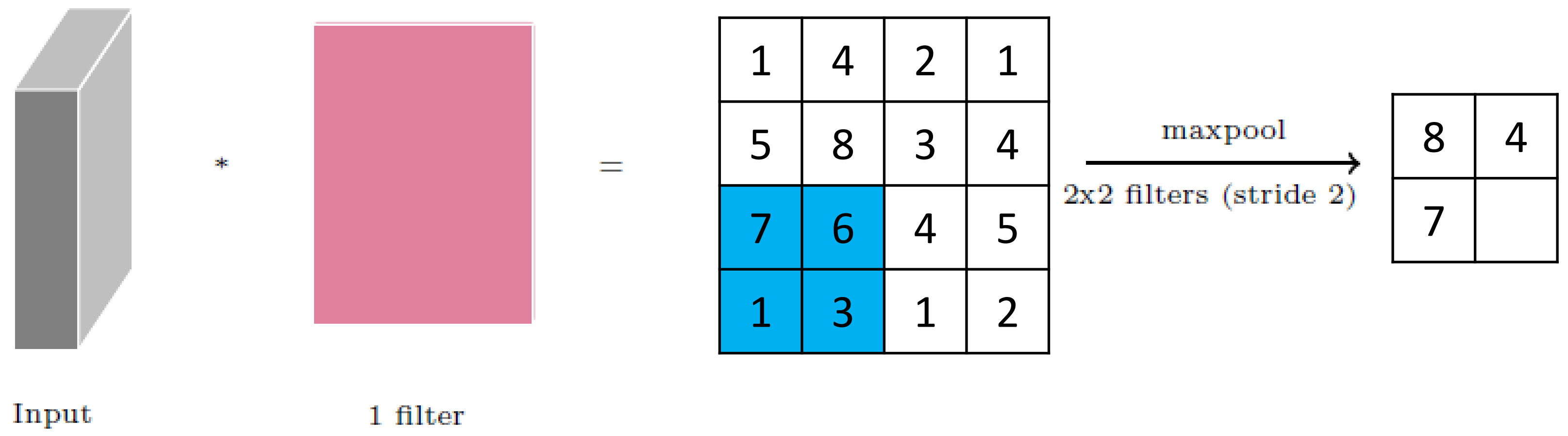
Pooling layer



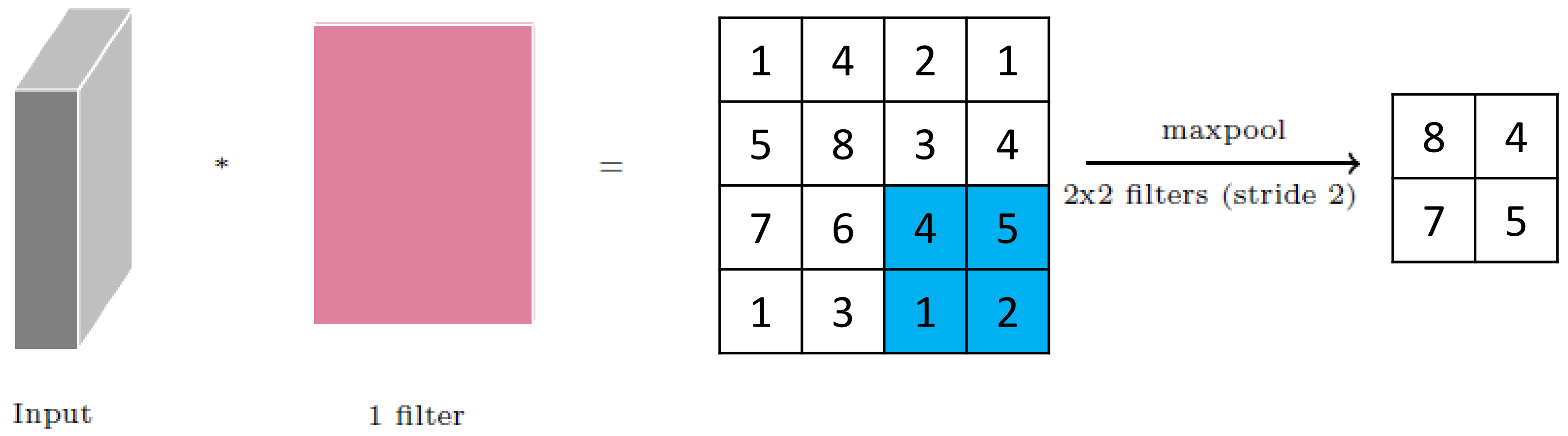
Pooling layer



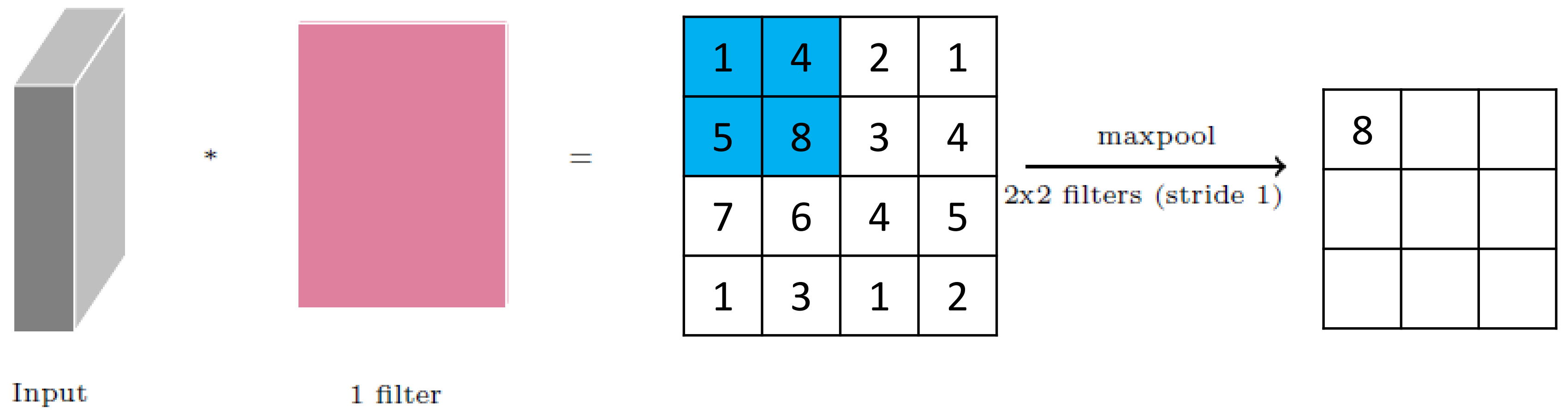
Pooling layer



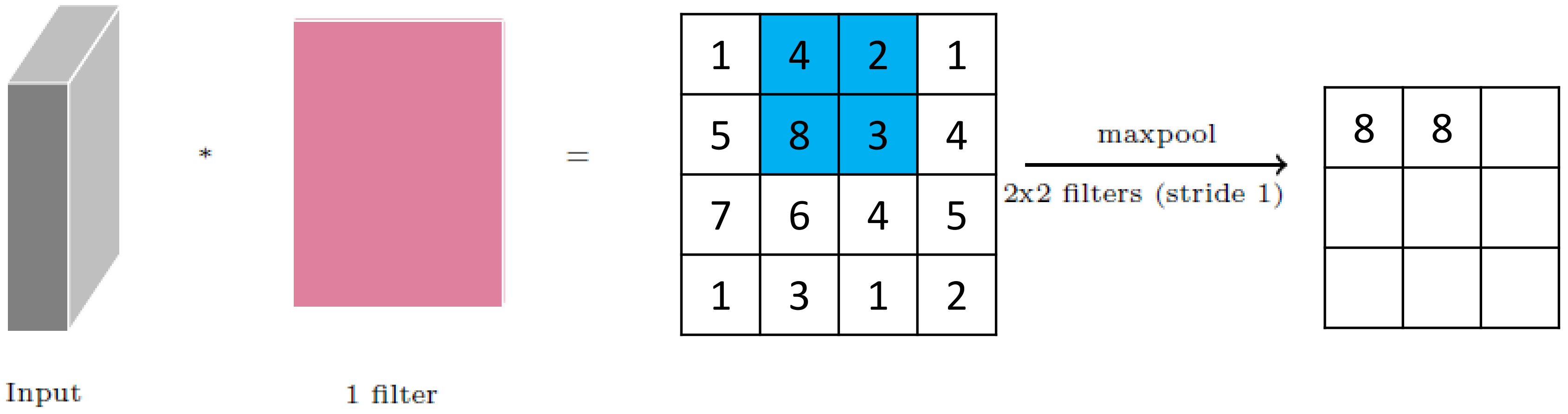
Pooling layer



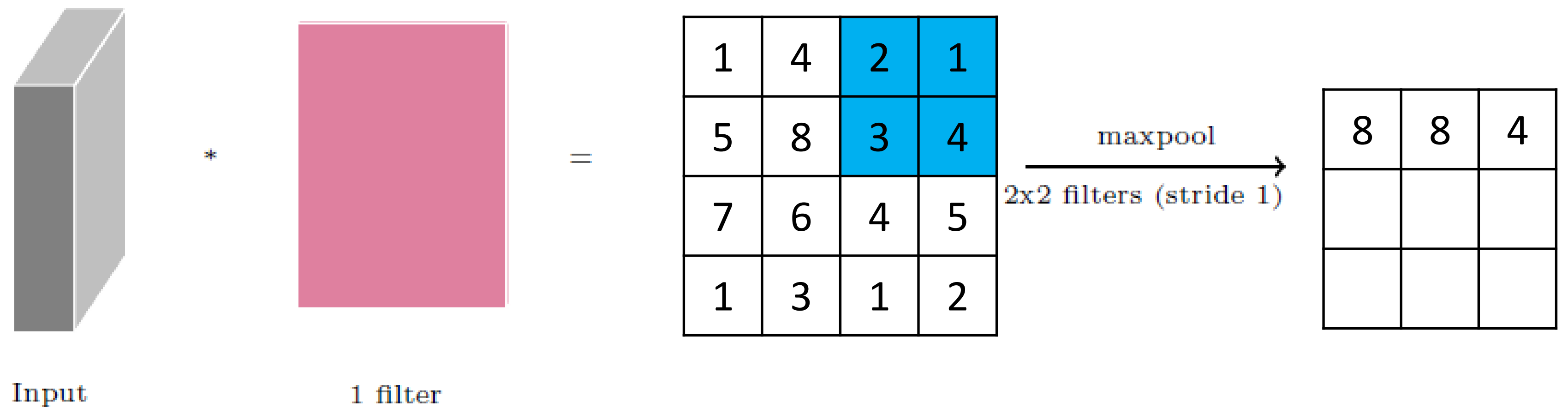
Pooling layer



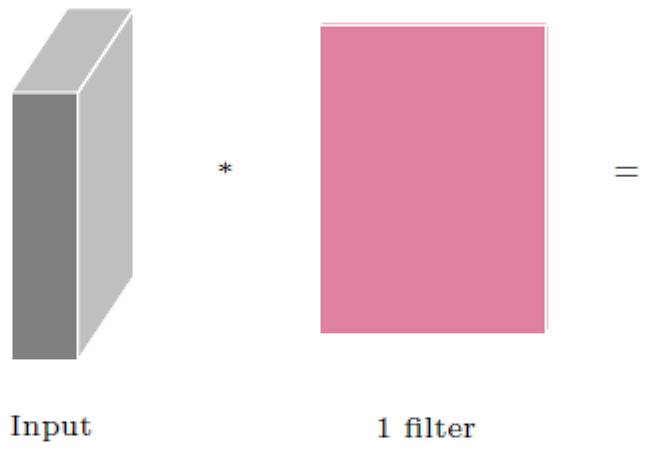
Pooling layer



Pooling layer



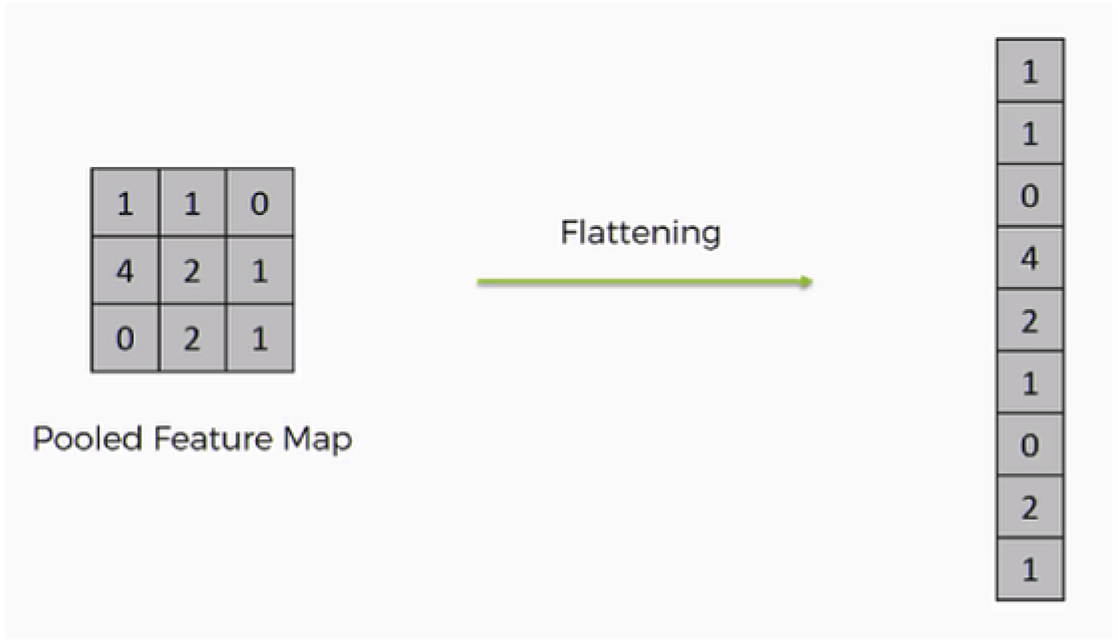
Pooling layer



$$\begin{array}{|c|c|c|c|}\hline 1 & 4 & 2 & 1 \\ \hline 5 & 8 & 3 & 4 \\ \hline 7 & 6 & 4 & 5 \\ \hline 1 & 3 & 1 & 2 \\ \hline \end{array} \xrightarrow[\text{2x2 filters (stride 1)}]{\text{maxpool}} \begin{array}{|c|c|c|}\hline 8 & 8 & 4 \\ \hline 8 & 8 & 5 \\ \hline 7 & 6 & 5 \\ \hline \end{array}$$

Flatten

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



Thank You