Edge Detection and Kernels

Pre-work: Computer Vision

Agenda

- Convolution and Edge Detection
- Filters and Kernels

Convolution and Edge Detection

Edge Detection

- As we previously discussed, **the Convolution operation** is the fundamental building block of Convolutional Neural Networks.

- Let's try to understand how the Convolution process works, with the help of the Edge Detection concept that Convolution filters are able to perform.

- Mathematically in terms of pixel intensity values, **edges** are image regions where there is a **sudden, sharp change in the brightness of the pixels**. This can be observed by a large change in the pixel intensity values among neighboring pixels in some direction.

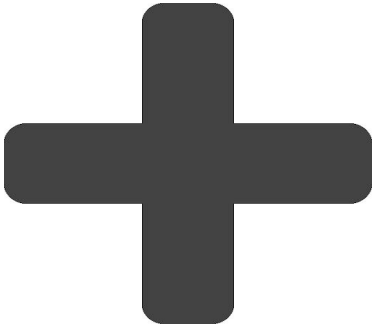
For example:

This image of a Plus (**+**) sign has edges that can be detected by Convolution.

Edge Detection

- **Let us first convert this image into its grayscale version**, so that we have a 2D matrix to work with for us to analyze the pixel intensity values of the image.

- The information about edges in images is usually preserved during grayscale conversion for most of the images we will work with, so we don't have to worry about color information loss for now.



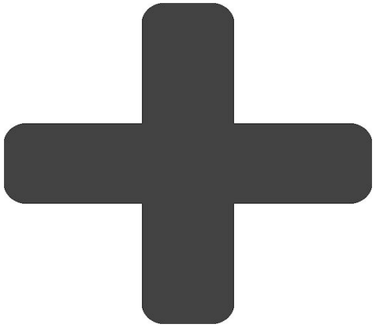
255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	117	61	67	67	67	67	61	117	255	255	255	255	255	255	255	255
116	116	116	116	116	116	116	116	80	65	67	67	67	67	65	80	116	116	116	116	116	116	116	116
61	61	61	61	61	61	61	61	65	67	67	67	67	67	67	65	61	61	61	61	61	61	61	61
67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
61	61	61	61	61	61	61	61	65	67	67	67	67	67	67	65	61	61	61	61	61	61	61	61
116	116	116	116	116	116	116	116	80	65	67	67	67	67	65	80	116	116	116	116	116	116	116	116
255	255	255	255	255	255	255	255	117	61	67	67	67	67	61	117	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255

The grayscale version of the image, converted into a matrix of pixel intensity values.

Edge Detection

- We observe that **there is a clear gradient or change in pixel intensity values** either side of the lines drawn inside this matrix. (Ex: 255 on one side and 116 and 61 on the other)
- **These gradient lines also mirror the '+' shape of the edges of the image**, so we know that they are a mathematical representation of what we visually perceive to be edges inside the image.

- **This gradient of pixel intensity values** is hence **the mathematical meaning of an edge** inside an image.



255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	117	61	67	67	67	67	61	117	255	255	255	255	255	255	255	255
116	116	116	116	116	116	116	116	80	65	67	67	67	67	65	80	116	116	116	116	116	116	116	116
61	61	61	61	61	61	61	61	65	67	67	67	67	67	67	65	61	61	61	61	61	61	61	61
67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
61	61	61	61	61	61	61	61	65	67	67	67	67	67	67	65	61	61	61	61	61	61	61	61
116	116	116	116	116	116	116	116	80	65	67	67	67	67	65	80	116	116	116	116	116	116	116	116
255	255	255	255	255	255	255	255	117	61	67	67	67	67	61	117	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	116	61	67	67	67	67	61	116	255	255	255	255	255	255	255	255

The grayscale version of the image, converted into a matrix of pixel intensity values.

The importance of Edges

- **So why are edges important?** Let's try to understand this with an example:
- On the left below, we have a grayscale image of a Ferrari 456 GT, a famous sports car from the nineties.
- On the right is another version of the same image, but with purely the edges of the image of the car on the left, and with very little information retained about the background.
- **Can we still identify the presence of the car** despite this loss of information? Yes, we can!

The importance of Edges

- What this means, is that **we can identify objects** in an image based **purely on the information about their edges**, even by completely ignoring their background or other minute details of the image that are not important in detecting that object.
  Therein lies **the importance of edges in image prediction tasks.**

- Edge detection is one of the primary tasks in object detection and object recognition.

- A lot of information about an image is found within its edges. So detecting edges and enhancing those areas should be our focus in image processing for the purpose of image classification tasks.

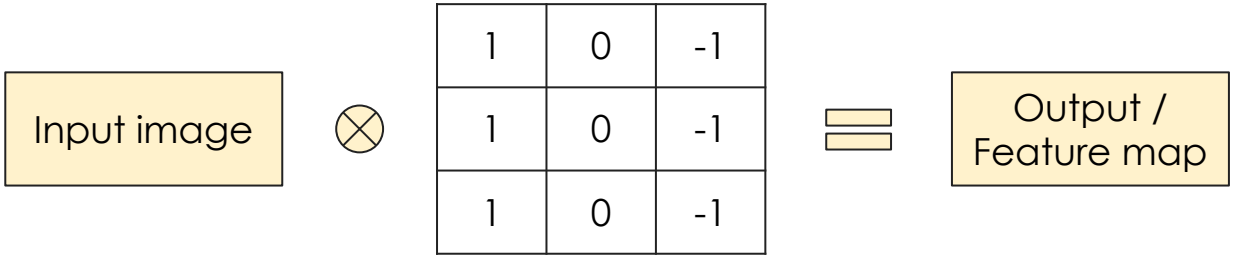
How do we detect edges?

- One way to mathematically detect edges in an image is to perform convolution operations on a input image with a small array of numbers, also called a **Filter / Kernel**.
- The term Kernel refers to a 2D array, while the term Filter generally refers to multiple kernels stacked together, to operate across a whole N-dimensional array. However the terms **Kernel and Filter are often used interchangeably** across the industry.

- There are essentially three steps involved in the convolution operation:
    i. **Taking in the Input Image** as a Pixelmap
    ii. **Applying a Filter / Kernel** on the Input Image Pixelmap
    iii. Obtaining the **Output Feature map**

How do we detect edges?

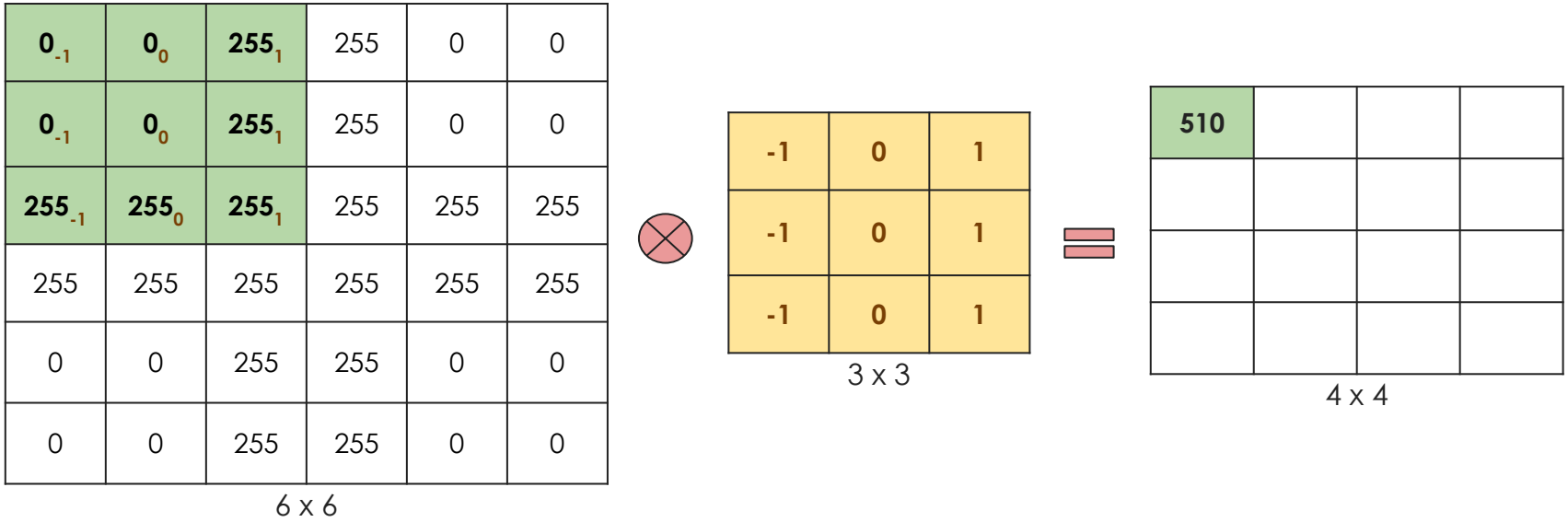
- The below visual is a representation of this three-stage convolution process.
- In the process, an input image (taken in the form of its pixelmap) is **convolved** with a kernel or filter, in order to obtain an output feature map.



- The above filter containing 1s, 0s and -1s, is an example of a **vertical edge detector**, as the pattern of numbers in the filter is vertically consistent (top to bottom) and will hence likely detect vertical edges inside the image.

- Let us now try to understand the mathematical working of convolution kernels, in order to understand the intuition behind how kernel edge detectors work.

Convolution operation



(0 X -1) + (0 X 0) + (255 X 1) + (0 X -1) + (0 X 0) + (255 X 1) + (255 X -1) + (255 X 0) + (255 X 1) = 510

- **The filter is first superimposed over the first corresponding 3 x 3 region** of the image, taking the products of the corresponding elements from the image and the filter, and then adding all these products to get a final sum.
- In other words, we take **the sum of the element-wise products of the input region and the filter**, to get the final number **510**.

Convolution operation

0	0_{-1}	255_{0}	255_{1}	0	0
0	0_{-1}	255_{0}	255_{1}	0	0
255	255_{-1}	255_{0}	255_{1}	255	255
255	255	255	255	255	255
0	0	255	255	0	0
0	0	255	255	0	0

6 x 6

⊗

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

3 x 3

=

510	510		

4 x 4

(0 X -1) + (255 X 0) + (255 X 1) + (0 X -1) + (255 X 0) + (255 X 1) + (255 X -1) + (255 X 0) + (255 X 1) = 510

- The filter then **slides** over the input image array (with a default **stride** of 1), with a new sum of element-wise products being computed from the new region of the image over which the filter is now superimposed.
- This new sum of element-wise products again gives us the number 510, which **goes into the second element of the output feature map.**

Convolution operation

0	0	255_{-1}	255_{0}	0_{1}	0
0	0	255_{-1}	255_{0}	0_{1}	0
255	255	255_{-1}	255_{0}	255_{1}	255
255	255	255	255	255	255
0	0	255	255	0	0
0	0	255	255	0	0

6 x 6

⊗

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

3 x 3

=

510	510	-510	

4 x 4

(255 X -1) + (255 X 0) + (0 X 1) + (255 X -1) + (255 X 0) + (0 X 1) + (255 X -1) + (255 X 0) + (255 X 1) = -510

- In the same way, the filter slides over the rest of the row in the input image array, and for each slide, the corresponding sum of element-wise products is computed and added to the relevant cell in the output feature map.

Convolution operation

0	0	255	255_{-1}	0_{0}	0_{1}
0	0	255	255_{-1}	0_{0}	0_{1}
255	255	255	255_{-1}	255_{0}	255_{1}
255	255	255	255	255	255
0	0	255	255	0	0
0	0	255	255	0	0

6 x 6

⊗

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

3 x 3

=

510	510	-510	-510

4 x 4

(255 X -1) + (0 X 0) + (0 X 1) + (255 X -1) + (0 X 0) + (0 X 1) + (255 X -1) + (255 X 0) + (255 X 1) = -510

- In the same way, the filter slides over the rest of the row in the input image array, and for each slide, the corresponding sum of element-wise products is computed and added to the relevant cell in the output feature map.
- As seen above, with a 6 x 6 image and a 3 x 3 filter, we reach the end of the row after the 4th sliding position.

Convolution operation

0	0	255	255	0	0
0_{-1}	0_{0}	255_{1}	255	0	0
255_{-1}	255_{0}	255_{1}	255	255	255
255_{-1}	255_{0}	255_{1}	255	255	255
0	0	255	255	0	0
0	0	255	255	0	0

6 x 6

⊗

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

3 x 3

=

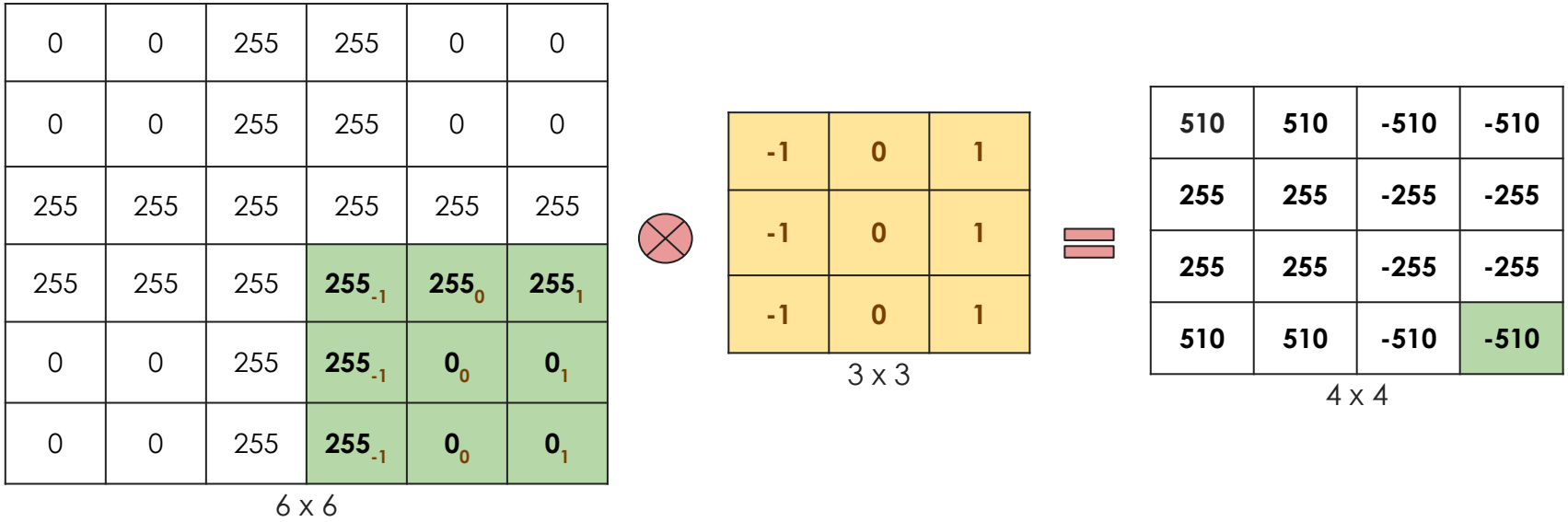
510	510	-510	-510
255			

4 x 4

(0 X -1) + (0 X 0) + (255 X 1) + (255 X -1) + (255 X 0) + (255 X 1) + (255 X -1) + (255 X 0) + (255 X 1) = 255

- Once the filter has completed the first row, **the same sliding mechanism moves in the column direction to the next row**. The same sliding of the filter is then carried out from the start to the end of the next row.
- The sum of element-wise products is computed in the same fashion, to give a new number **255**, which goes into the first cell of the next row in the output.

Convolution operation



(255 X -1) + (255 X 0) + (255 X 1) + (255 X -1) + (0 X 0) + (0 X 1) + (255 X -1) + (0 X 0) + (0 X 1) = -510

- This **sliding mechanism continues** till the last row and last column, and the sum of element-wise products goes to the right cell in the output matrix.
- In general, for an **n x n** input image, and an **f x f** filter size, the output feature map will have dimensions of **(n-f+1) x (n-f+1)**.
- That is how we have an output dimension of (6-3+1) x (6-3+1) or **4 x 4** here.

Intuition behind Convolutions

- Convolutions can be used for various kinds of feature extraction tasks, depending only on the pixel pattern of the filter.

-1	**0**	**1**
-1	**0**	**1**
-1	**0**	**1**

The convolution filter used in the previous example, for instance, has a vertical pattern of -1s, 0s and 1s.

It is also known in the industry as the **Prewitt Filter.**

- The above convolution filter would be considered a type of **vertical edge detector**, capable of identifying vertical edges in the image.

- The reason for this is due to the nature of the convolution operation, which uses a sum of element-wise products. The implication of that is **those regions of the image which have similar vertical patterns to those of the filter**, **get amplified** in the element-wise product, and hence in the output matrix.

Intuition behind Convolutions

- In other words, convolution filters are adept at **identifying those regions of the image which have pixel patterns that match the pixel pattern of the filter** itself.

-1	**0**	**1**
-1	**0**	**1**
-1	**0**	**1**

- That is why the above filter would be considered a type of vertical edge detector.

  It has a vertical pattern of pixels (-1s, 0s and 1s) which represent a left-to-right increasing gradient. Any regions in the input image that have a similar pattern to this would be identified by this filter, and amplified numerically with a higher output value for those regions.

- One other point of consideration is that **filters with odd-valued dimensions**

  (such as 3 x 3 Filters) **are preferred**, due to the presence of a central pixel.

Filters and Kernels

- Convolution filters can hence be used for various image manipulation tasks, such as edge detection, sharpening or blurring.

- As an example, let us discuss three convolution filters which have gained popularity over the years:

    1. **The Prewitt filter**
    2. **The Sobel filter**
    3. **The Laplacian filter**

The Prewitt filter

- **The Prewitt filter** (named after Judith M.S. Prewitt) has mostly been used in research for **detecting horizontal and vertical edges in an image.**

- This filter has three properties:
  - The sum of the elements in the filter is zero.
  - The elements present in the filter should have opposite signs.
  - The higher the magnitude of the positive and negative numbers, the higher the chances of detecting an edge.

- There are actually **two variations** of the filter, for detecting edges in **both the horizontal and the vertical directions.**

Horizontal Prewitt filter

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

Vertical Prewitt filter

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

The Prewitt filter

Original image

Combination of outputs from horizontal and vertical Prewitt filters

The Sobel filter

- **The Sobel filter** was conceptualized by Irwin Sobel and Gary Feldman from the Stanford Artificial Intelligence Laboratory (SAIL) in the 1960s.

- It is similar to the Prewitt Filter, with the **only change being in the central values of the positive and negative numbers**, which are now +2 and -2 instead of +1 and -1.

- This filter can give better intensities in the regions containing edges, in comparison to the results from the Prewitt Filter.

Horizontal Sobel filter

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

Vertical Sobel filter

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

The Sobel filter

Original image

Vertical Sobel filter output

Horizontal Sobel filter output

Combination of outputs

The Laplacian filter

- **The Laplacian filter** (named after Pierre-Simon Laplace, the famous French scientist with several contributions in mathematical and statistical theory) is a second-order derivative filter that tries to highlight those regions where pixel intensities change abruptly.

- The implication of being a second-order filter is that unlike the Prewitt and Sobel Filters (which need a kernel in the horizontal and vertical directions each), **the Laplacian filter only needs one kernel**, which can detect both kinds of edges.

- One other thing to keep in mind is that **Laplacian Filters are very sensitive to noise.** So in many cases, the image may need to be smoothed out before applying the Filter.

Laplacian filter variant 1

0	1	0
1	-4	1
0	1	0

Laplacian filter variant 2

-1	-1	-1
-1	8	-1
-1	-1	1

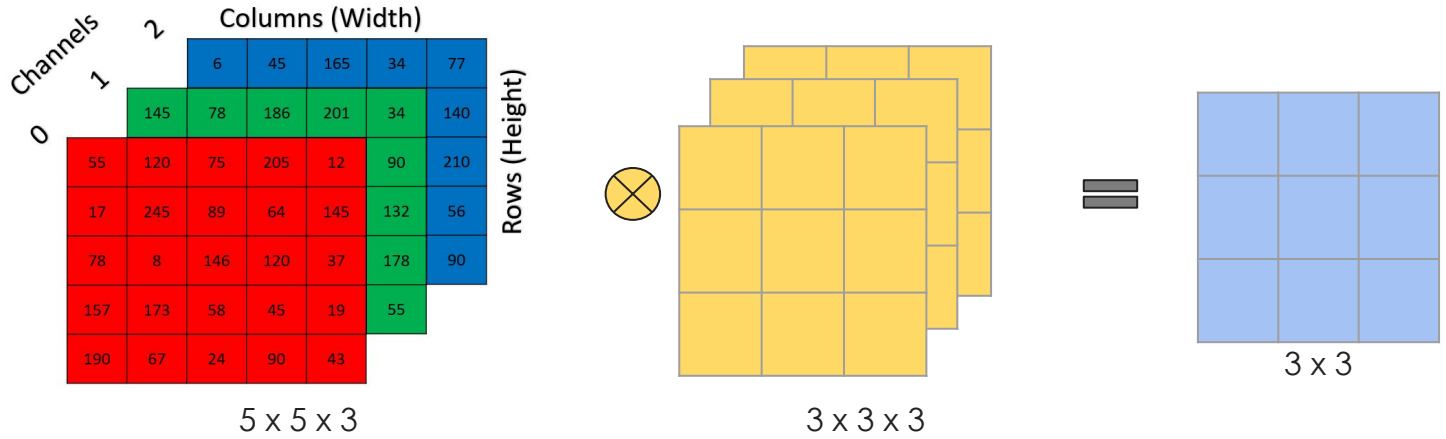
The Laplacian filter

Original image

Laplacian filter output

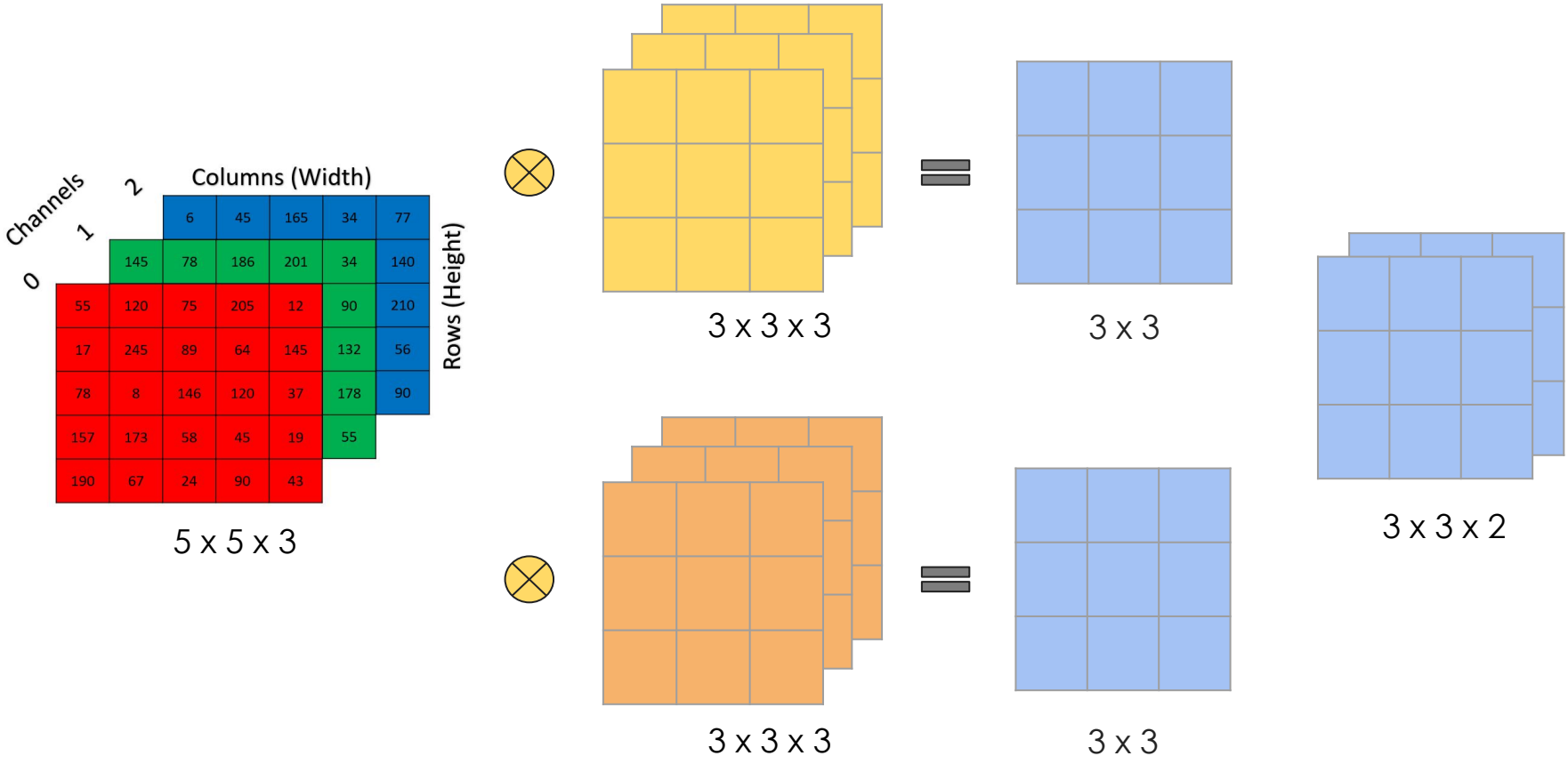
Convolution on RGB images

- When performing the convolution operation on RGB images as opposed to grayscale, **the number of channels in the filter must equal the number of channels in the image (3).**



- To compute the output of this convolution operation for example, we take the 3×3×3 filter, and first place it in the upper-left most position.
- Notice that **the 3×3×3 filter has 27 numbers.**
- So we take each of these 27 numbers, multiply them with the corresponding first 27 numbers (9 each - 3 x 3 from the red, green and blue channels each), and **add up those products to get a final sum.** That sum goes into the relevant cell in the output.

Multiple filters



Convolutional Neural Networks (CNNs)

- However, the Computer Vision industry came to understand that such **handcrafted filters would not easily generalize** across all kinds of images.

- **There are a lot of complex patterns that need to be detected** in many kinds of images, and it is far from trivial to think of all the expected patterns in an image and design handcrafted filters that could try and capture all of those patterns. Such an endeavor (similar to feature engineering) is ideally something we would want to automate in some way to help the model scale and generalize well to different tasks.

- The idea of Convolutional Neural Networks is that while the dimensions of the convolution filters might remain a hyperparameter (which we give to the model), **the values of the convolution filters themselves, could become learnable parameters.** These are values which the model would learn by itself through the iterative cycle of forward propagation and backpropagation, to find the filter values that work best for that predictive task.

W1	W2	W3
W4	W5	W6
W7	W8	W9

Rather than us handcrafting the values of the Ws in the filter, in a CNN, we merely give the dimensions of the filter, but **ask the CNN to learn the best values of Ws for the task by itself.**

Convolutional Neural Networks (CNNs)

- **This turns out to be the magic of Convolutional Neural Networks (CNNs)**, and this is one of the main reasons why they have enjoyed great success in image prediction tasks across a wider variety of image-related data than was previously possible.

- So the way convolutional filters really work in CNNs is by the Neural Network itself learning the best values / parameters of the filter - **this eliminates the need for hand-coding values** and the developer having to choose good filter values for image prediction tasks.

- In the coming module, we will learn more about some other architectural building blocks of a CNN, before understanding their end-to-end structure and getting a sense for all the image manipulation operations happening in the various layers of the CNN.

W1	W2	W3
W4	W5	W6
W7	W8	W9

Rather than us handcrafting the values of the Ws in the filter, in a CNN, we merely give the dimensions of the filter, but **ask the CNN to learn the best values of Ws for the task by itself.**

Thank You