

**Department of Artificial Intelligence**

**College of Computer Science and Information Technology**

1. **Objectives**

This lab is designed to achieve the following goals:

1. Get Started with Filters and convolutions.
2. Get familiar with Noise and Denoising.
3. **Introduction**

## Filters

## We can use Filters to modify or enhance an image’s features. You’ll be pretty familiar with filters if you’ve ever played around with images on social media platforms.

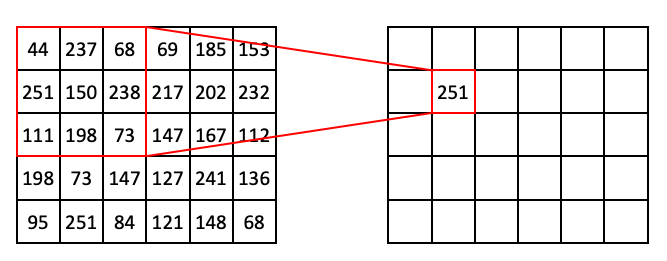
We can use filters for various purposes, such as smoothing and sharpening the image, removing noise, highlighting features and edges in the image, etc.

When we apply a filter on an image, every pixel value is replaced by a new value generated using surrounding pixel values.

**Filtering** is a fundamental operation in image processing in which a new image is created from an input image by following some algorithm or mathematical operation. Most of the filters you will encounter follow a basic pattern of:

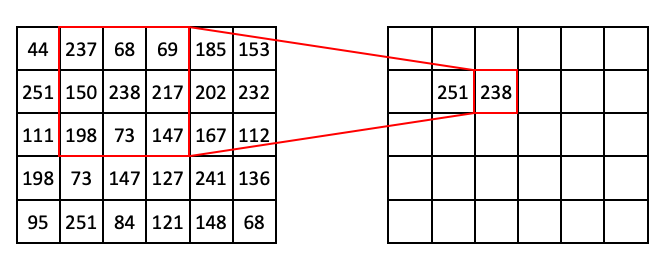
1. Collect all pixel values in the **neighborhood** surrounding a particular target pixel in the input image.
2. Reduce the neighborhood of pixels values to a single value via some operation (e.g., mean, median, dot product, etc.)
3. In the output image, populate the pixel that corresponds to the target pixel in the input image with the value from Step 2.
4. Repeat Steps 1 through 3 for all pixels in the input image.

To see this in action, consider a 3 x 3 Max Filter applied to an array with shape (5, 6):

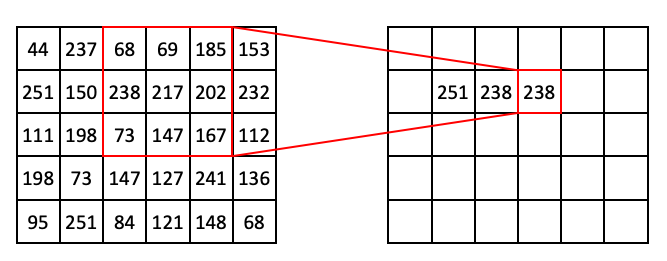
[](https://github.com/TheJacksonLaboratory/Basic_skimageJAX/blob/master/lessons/slides/Filters_Slide12.png)

In the above example, the target pixel is located at (1, 1) and contains the value 150. In the 3 x 3 neighborhood surrounding that target pixel, the values are (44, 237, 68, 251, 150, 238, 111, 198, 73). The maximum value in that list is 251, so position (1, 1) of the output array is filled in with 251.

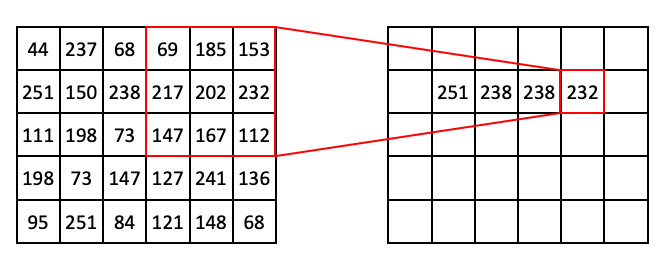
If we slide the 3 x 3 window one step to the right, we are now targeting (1, 2), which has the value of 238 in our input array. This also happens to be the maximum value in the neighborhood:

[](https://github.com/TheJacksonLaboratory/Basic_skimageJAX/blob/master/lessons/slides/Filters_Slide13.png)

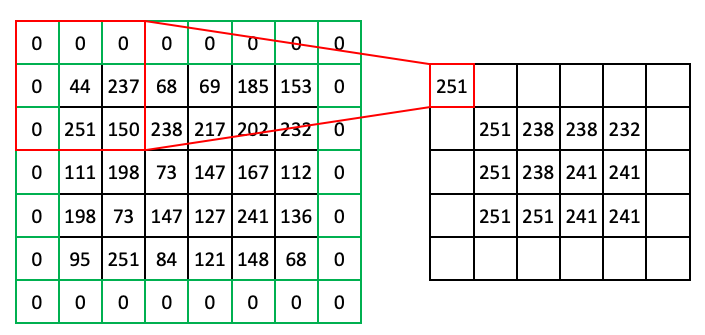
If we extend this process further, we can begin to fill out the row:

[](https://github.com/TheJacksonLaboratory/Basic_skimageJAX/blob/master/lessons/slides/Filters_Slide14.png)

Then:

[](https://github.com/TheJacksonLaboratory/Basic_skimageJAX/blob/master/lessons/slides/Filters_Slide15.png)

At this point you might notice that, if we use a 3 x 3 window, we can't target pixels on the edge of the image. To get around this problem, we can use a technique called **padding**, where we add extra pixels to the input image. Here is an example of zero padding:

[](https://github.com/TheJacksonLaboratory/Basic_skimageJAX/blob/master/lessons/slides/Filters_Slide16.png)

You can imagine other types of padding. Another common pad is 'edge' or 'extend' where the edge pixels just repeat as far as needed to fill the neighborhood.

*Note: Not all filters strictly require padding, but it's a generally useful concept to know, so it is shown here.*

Here we went over an example of a Max filter, but you'll see many types of these window-based filters including but not limited to:

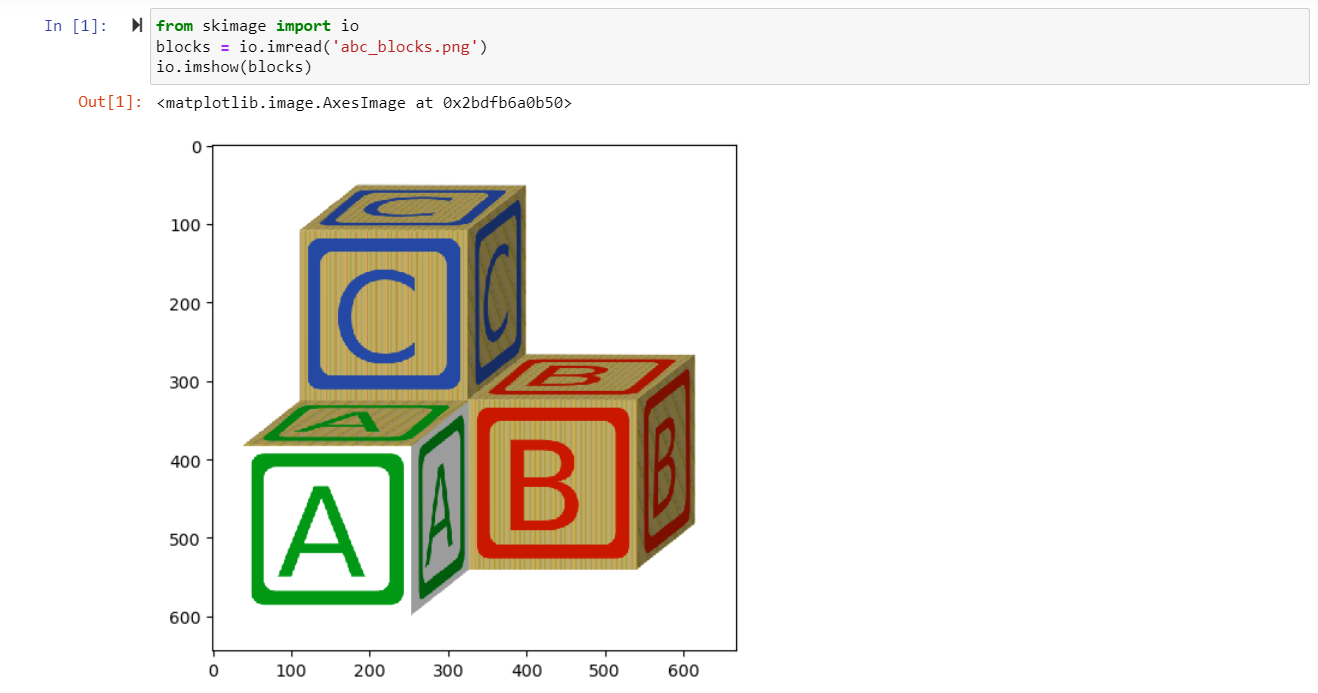
* Rank filters such as max, min, and median
* Filters based on descriptive statistics such as mean and standard deviation
* Convolutional filters (more on these later)

**Filters have a lot of practical uses in image processing, such as:**

* Smoothing out noise (despeckling)
* Expanding light or dark areas
* Highlighting regions of interest such as edges
* Calculating features for machine learning

## Filtering images with scikit-image

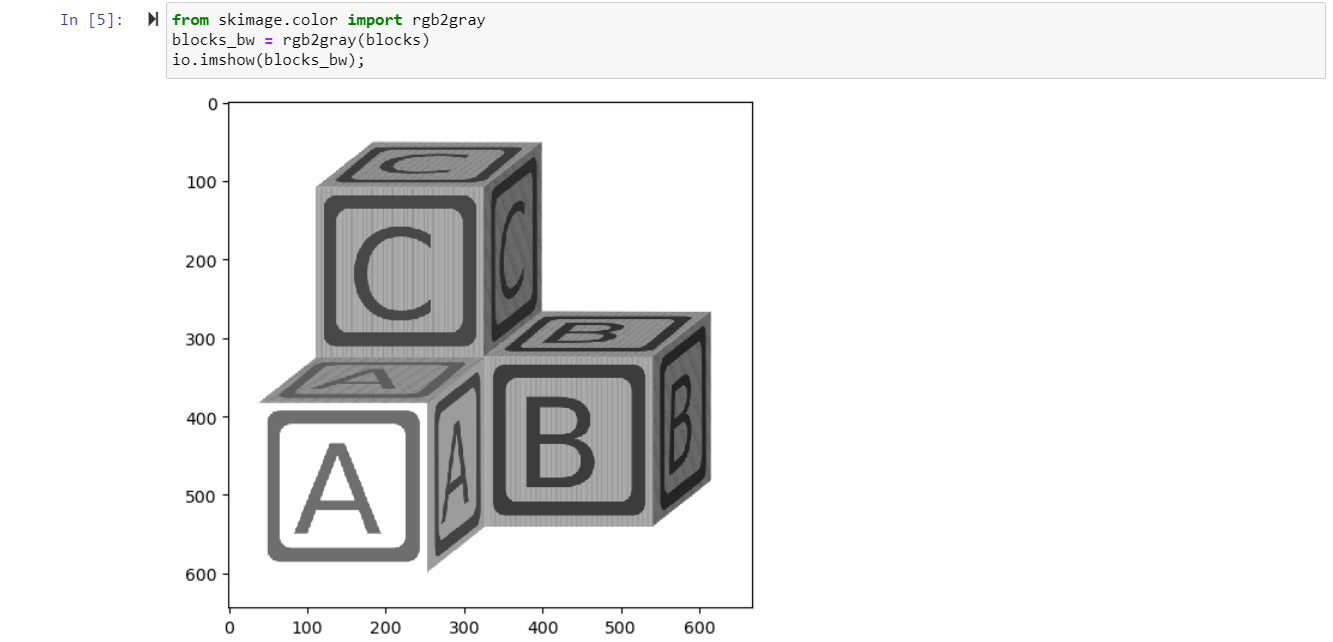
To begin, we'll start with a grayscale and inverted version of the ABC blocks image.



check some image attributes



convert the image to grayscale



A screenshot of a computer

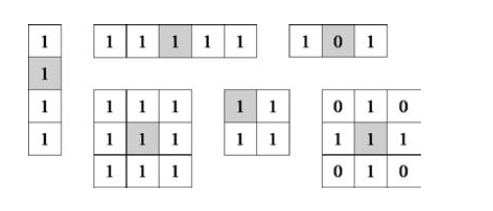
Description automatically generated 

Now let's take a look at some rank filters.

First, we need to indicate the shape of the neighborhood around each pixel. This is just an array of ones and zeros, where ones indicate pixels to be included in the neighborhood and zeros indicate excluded pixels. This neighborhood indicator is often referred to as a **structuring element**, abbreviated in the scikit-image API as selem.

Structuring elements and neighborhoods

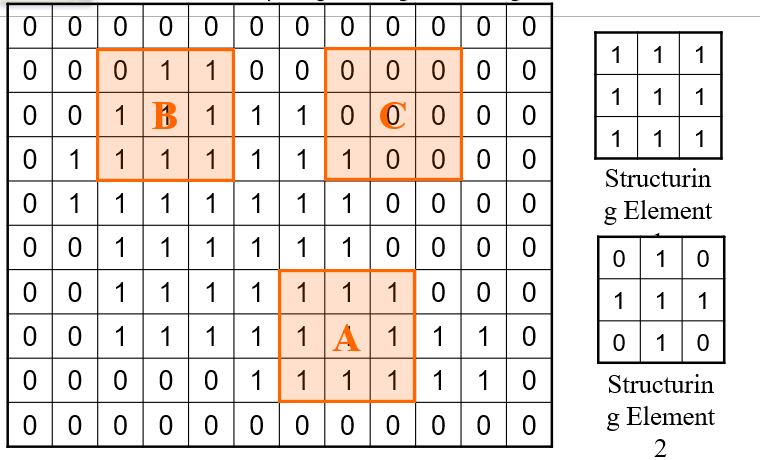
* A structuring element is a rectangular array of pixels containing the values either 1 or 0.



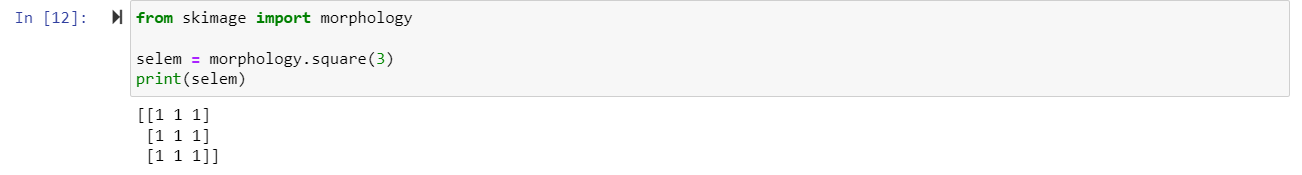
A diagram of a diagram

Description automatically generated

Structuring elements can be any size and make any shape. However, for simplicity we will use rectangular structuring elements with their origin at the middle pixel.



In our examples in the beginning of this lesson, we used the shape (3, 3). We can create that fairly easily.

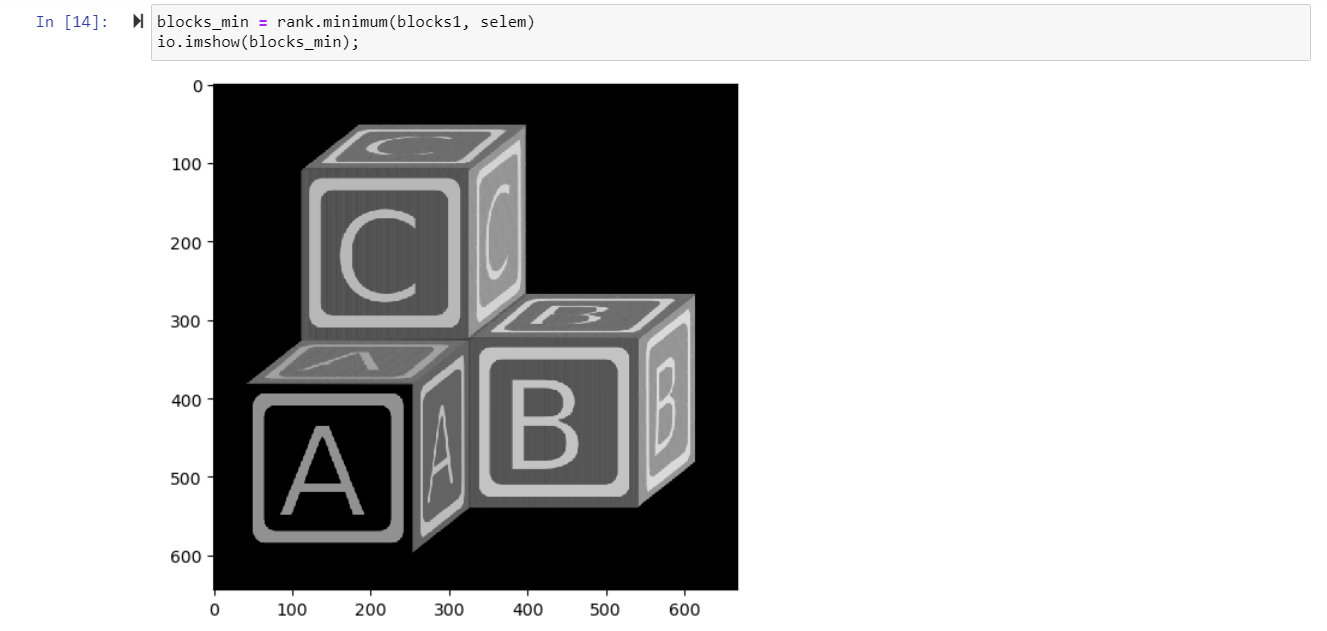


Using that selem, we can apply a Max Filter:

A screenshot of a computer

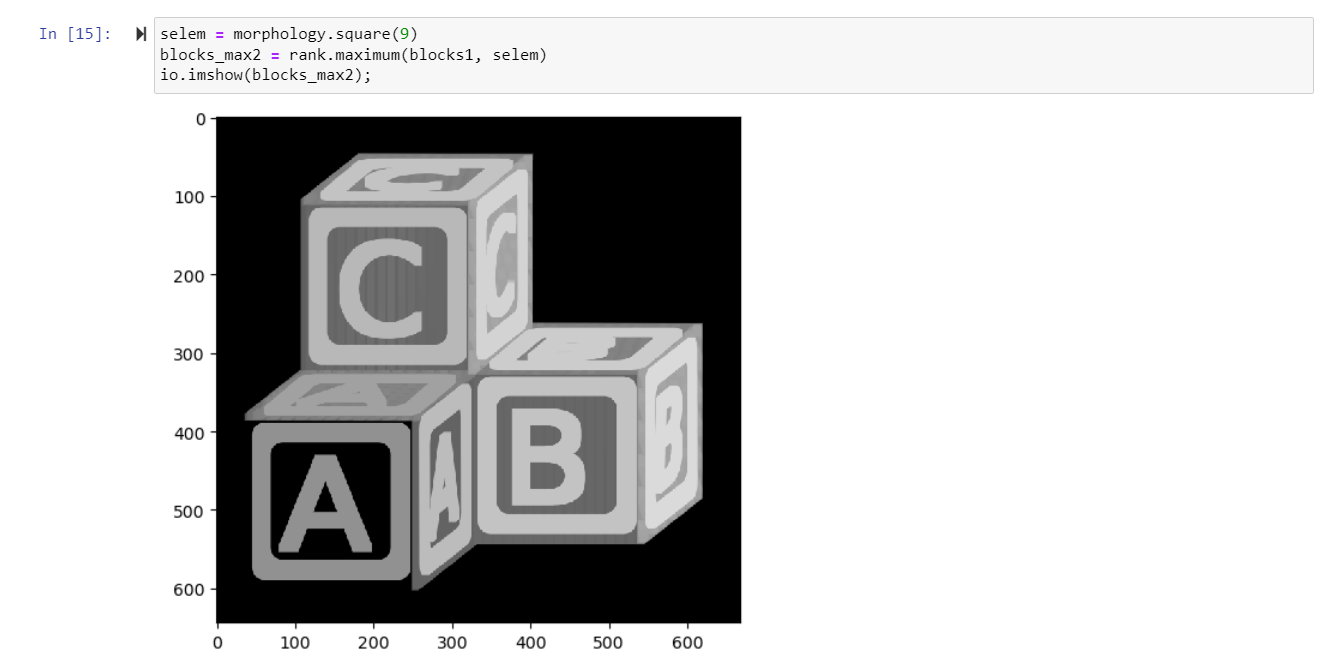
Description automatically generated

Here you can see that the Max Filter expanded the bright regions of the image into the dark regions. A Min filter does the opposite:



You can see that, compared to the original image, the bright areas have receded.

Now watch what happens when you use a larger structuring element (9 x 9) with these operations:



A screenshot of a computer

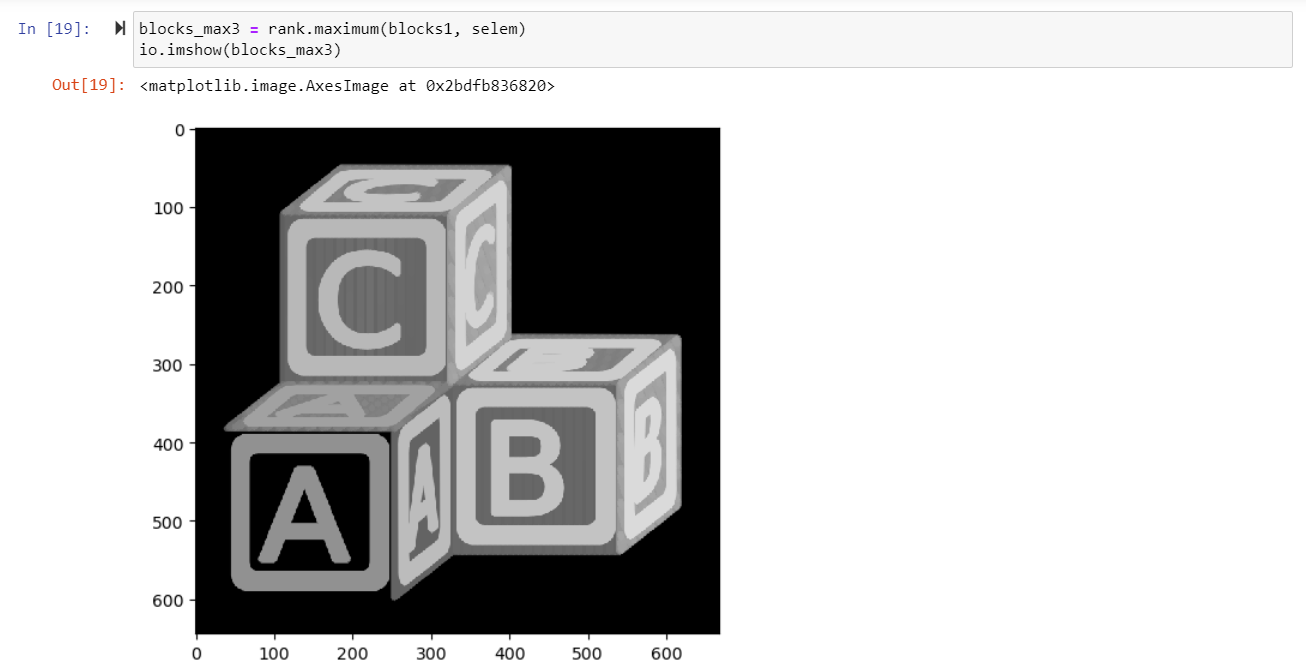
Description automatically generated

Structuring elements don't necessarily have to be square. For example, here is a disk-shaped structuring element.

A white rectangular object with a white background

Description automatically generated with medium confidence For this structuring element, rather than taking all of the pixels in a 9 x 9 window, only the pixels that correspond to a 1 in the element would be considered.

Watch how a round element changes the effect of the max filter:

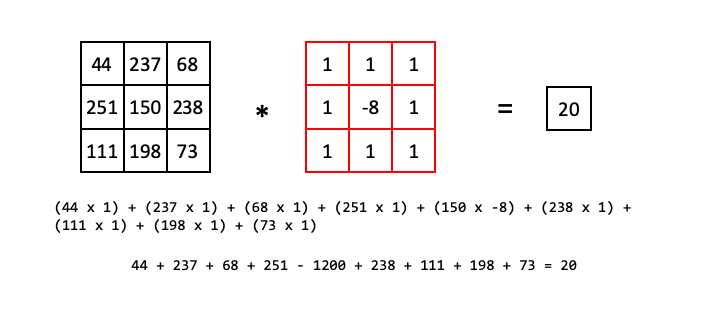


**Convolutional Filters**

Filters based on [convolution](https://en.wikipedia.org/wiki/Convolution) are fundamental not only to image processing, but other aspects of [signal processing](https://en.wikipedia.org/wiki/Signal_processing) and [machine learning](https://en.wikipedia.org/wiki/Convolutional_neural_network) as well. It may be worth reading up on the mathematical basis of convolution to better understand what it entails. Here we will discuss only the specific case pertaining to image analysis.

The basic concept of a convolutional filter is similar to the filters described above. Output image pixels are derived from a mathematical operation on a neighborhood surrounding the corresponding input image pixel. But the nature of the operation is quite different from the previous examples. Each step of a convolution entails the calculation of the [dot product](https://en.wikipedia.org/wiki/Dot_product) between the pixel values in a region of input image and corresponding values in a kernel of equal shape. The choice of the values of the kernel determines the effect of the filter.

Consider the following example, showing only one step of a convolutional filter. We take all of the pixel values of some region of input image, (44, 237, 68, 251, 150, 238, 111, 198, 73) and then the corresponding pixel values in our kernel (1, 1, 1, 1, -8, 1, 1, 1, 1) and calculate the dot product:

[](https://github.com/TheJacksonLaboratory/Basic_skimageJAX/blob/master/lessons/slides/Convolution_Slide1.png)

So the pixel in the output image that corresponds to the 150 value in the input image will have the value 20.

With that in mind, let's look at the case of a simple edge detector applied to a simple edge. In the input image below, there is a single vertical edge separating a dark region (all 0) from a bright region (all 255).

The kernel we will use for edge detection is called a [Prewitt](https://en.wikipedia.org/wiki/Prewitt_operator) operator and looks like:

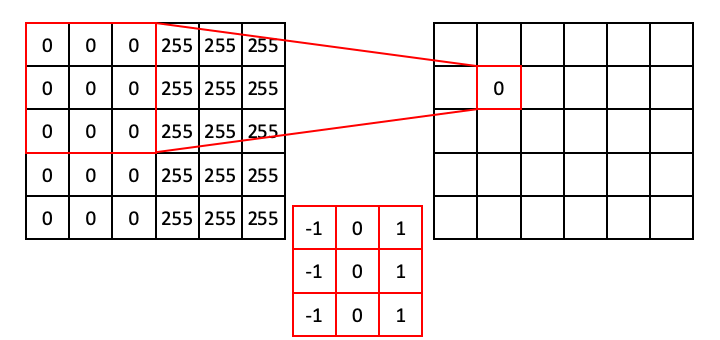
1 0 -1

1 0 -1

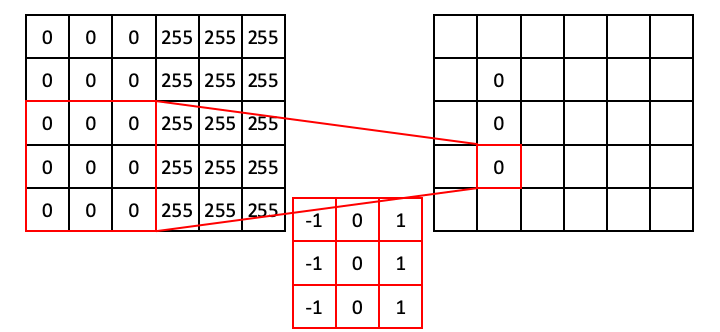
1 0 -1

It is important to note here that when we calculate the dot product with this operator, we first flip it along both the horizonal and vertical axes before matching up corresponding values. This has to do with the mathematical basis of the convolution function. In the following examples below, we show the kernel flipped so that it is easier to match up corresponding pixel values.

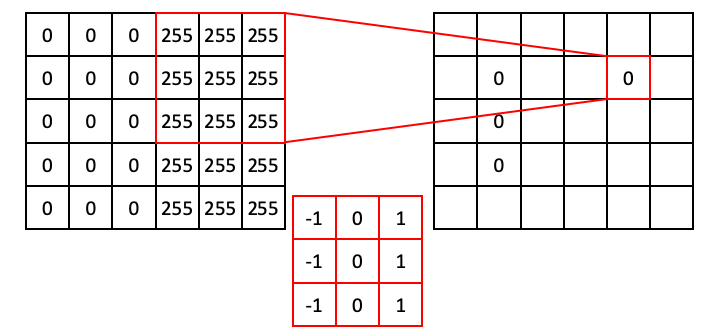
For regions of all 0, the dot product will simply be 0.

[](https://github.com/TheJacksonLaboratory/Basic_skimageJAX/blob/master/lessons/slides/Convolution_Slide2.png)

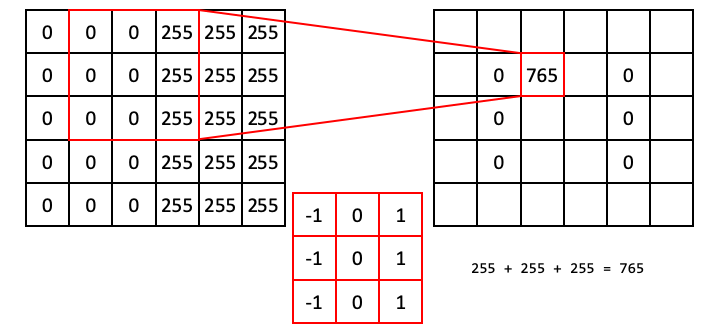
Moving down the column:

[](https://github.com/TheJacksonLaboratory/Basic_skimageJAX/blob/master/lessons/slides/Convolution_Slide3.png)

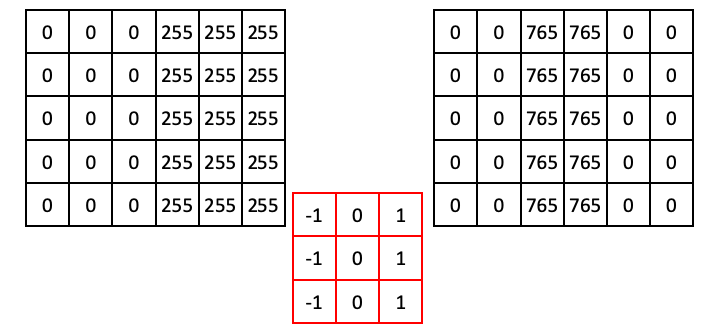
Similarly, for regions where the values are all 255, the dot product will also be zero (because the positive values resulting from the ones will exactly cancel out the values resulting from the negative ones).

[](https://github.com/TheJacksonLaboratory/Basic_skimageJAX/blob/master/lessons/slides/Convolution_Slide4.png)

However, the situation is different when the operator straddles an edge. In the next example, the ones align with 255s, whereas the negative ones align with zeros. This results in a dot product of 765.

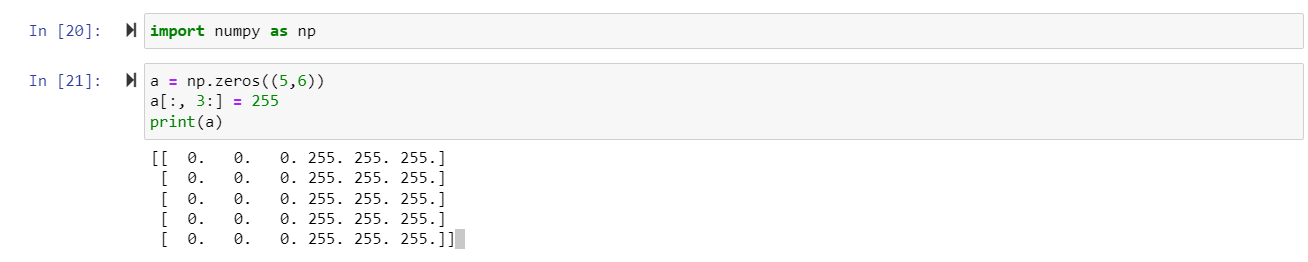
[](https://github.com/TheJacksonLaboratory/Basic_skimageJAX/blob/master/lessons/slides/Convolution_Slide5.png)

Following this operation all the way through (with [edge padding](https://scikit-image.org/docs/dev/api/skimage.util.html#skimage.util.pad)) produces an output array that shows signal only over the edge in the original image (i.e., a region of high horizontal [gradient](https://en.wikipedia.org/wiki/Image_gradient)), and regions of the input image that are constant result in zeros:

[](https://github.com/TheJacksonLaboratory/Basic_skimageJAX/blob/master/lessons/slides/Convolution_Slide6.png)

Now that we understand how convolutional filters work, we can try some in scikit-image. We'll first try the vertical Prewitt filter example we just showed.

We first use some numpy to create the example array:



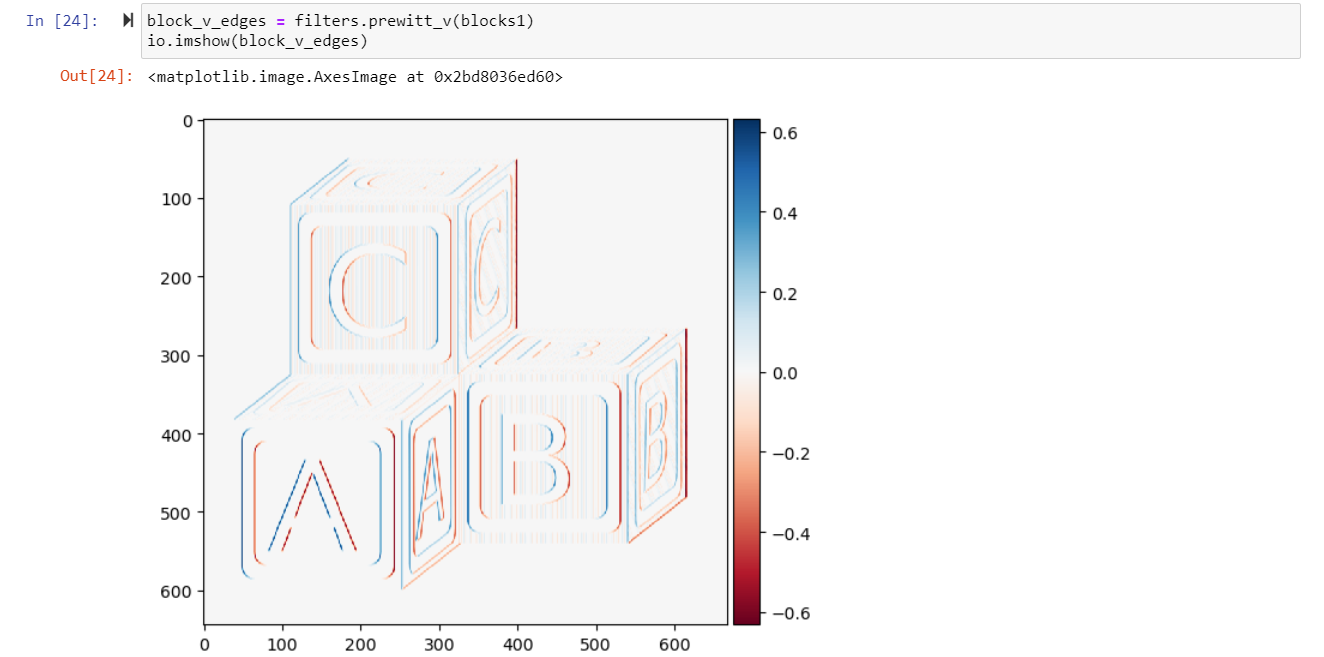
Now for the filter:

A white rectangular object with a black border

Description automatically generated with medium confidence

You'll notice that this output is different from our example. This is because scikit-image knows that we started with a uint8 image, and 765 is out of the range of values that are acceptable for that data type. The pixel intensities were thus rescaled, athough the output array is still float64. This is a good example of why you always want to explore the behavior of code before you use it in your analysis!

To better appreciate the utility of this filter, we can apply it to our block image:



You can see that our edge detector did a pretty good job of detecting edges with a vertical orientation. The floating point values are positive where the edge is a transition from a relatively darker area to a lighter area, and negative when transitioning from light to dark.

We can similarly find all horizontal edge:

A screenshot of a graph

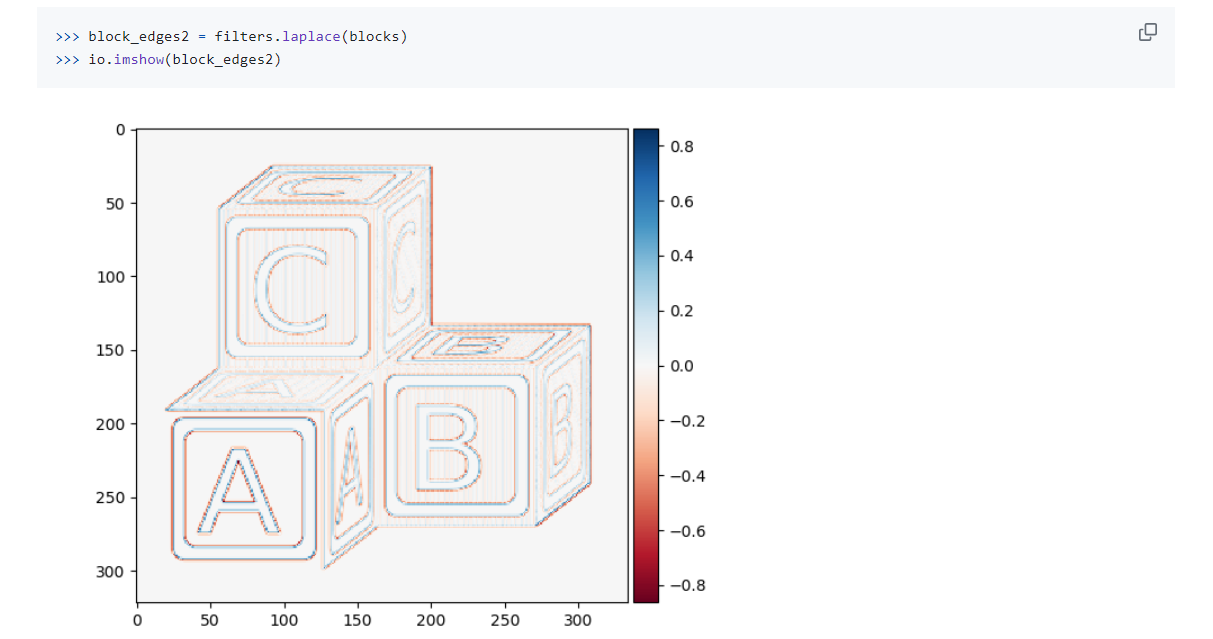
Description automatically generated

Or all edges:



Note here that the color scheme is different - there are no negative values because filters.prewitt does not capture orientation information.

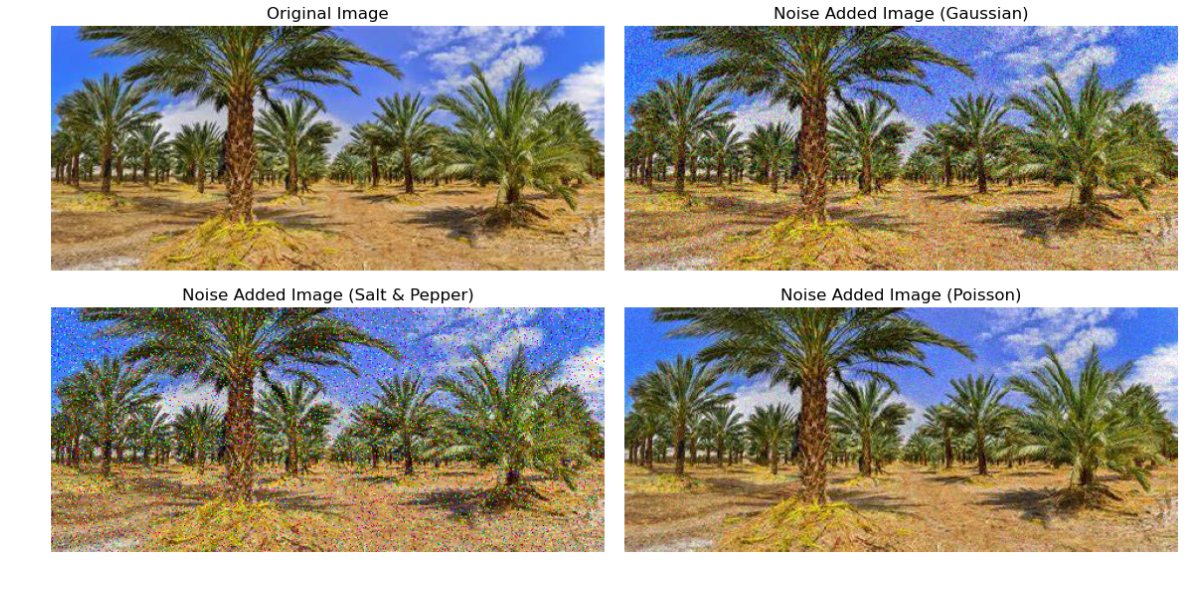
Another nice edge filter is the [Laplacian](https://en.wikipedia.org/wiki/Laplace_operator) filter:



1. **Noise and Denoising**

We can add noise to an image using skimage. Here, we will add random noise to a landscape image using Gaussian, Salt & Pepper and Poisson distributions.

****

****

# **Removing Noise from an Image:**

We learned how to add noise to an image. Let’s learn how to get rid of it too. The Gaussian smoothing filter mentioned above can denoise an image, but it also blurs the edges. Here, we will use two techniques to denoise an image: **Total Variation (TV)** and **Bilateral**. TV has the ability to remove noise from the image without blurring its edges, however this comes at the expense of contrast loss. Bilateral filter is also great at reducing the noise while preserving edges, but it may produce cartoonish looking images. In the following, we will use the Gaussian noise added image from the above and apply TV and Bilateral filters to it.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated