

Images and image processing

8 minutes

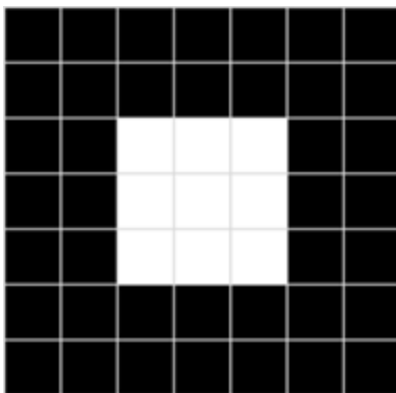
Before we can explore image processing and other computer vision capabilities, it's useful to consider what an image actually *is* in the context of data for a computer program.

Images as pixel arrays

To a computer, an image is an array of numeric *pixel* values. For example, consider the following array:

```
0  0  0  0  0  0  0
0  0  0  0  0  0  0
0  0 255 255 255 0  0
0  0 255 255 255 0  0
0  0 255 255 255 0  0
0  0  0  0  0  0  0
0  0  0  0  0  0  0
```

The array consists of seven rows and seven columns, representing the pixel values for a 7x7 pixel image (which is known as the image's *resolution*). Each pixel has a value between 0 (black) and 255 (white); with values between these bounds representing shades of gray. The image represented by this array looks similar to the following (magnified) image:



The array of pixel values for this image is two-dimensional (representing rows and columns, or *x* and *y* coordinates) and defines a single rectangle of pixel values. A single layer of pixel values like

this represents a grayscale image. In reality, most digital images are multidimensional and consist of three layers (known as *channels*) that represent red, green, and blue (RGB) color hues. For example, we could represent a color image by defining three channels of pixel values that create the same square shape as the previous grayscale example:

Red:

150	150	150	150	150	150	150
150	150	150	150	150	150	150
150	150	255	255	255	150	150
150	150	255	255	255	150	150
150	150	255	255	255	150	150
150	150	150	150	150	150	150
150	150	150	150	150	150	150

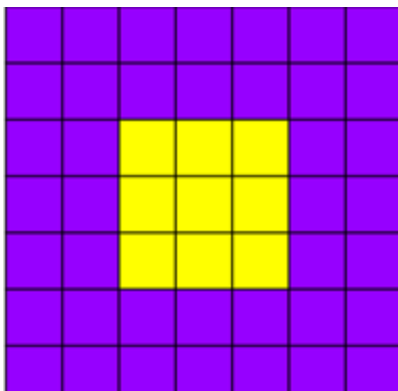
Green:

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	255	255	255	0	0
0	0	255	255	255	0	0
0	0	255	255	255	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Blue:

255	255	255	255	255	255	255
255	255	255	255	255	255	255
255	255	0	0	0	255	255
255	255	0	0	0	255	255
255	255	0	0	0	255	255
255	255	255	255	255	255	255
255	255	255	255	255	255	255

Here's the resulting image:



The purple squares are represented by the combination:

```
Red: 150  
Green: 0  
Blue: 255
```

The yellow squares in the center are represented by the combination:

```
Red: 255  
Green: 255  
Blue: 0
```

Using filters to process images

A common way to perform image processing tasks is to apply *filters* that modify the pixel values of the image to create a visual effect. A filter is defined by one or more arrays of pixel values, called filter *kernels*. For example, you could define filter with a 3x3 kernel as shown in this example:

```
-1 -1 -1  
-1 8 -1  
-1 -1 -1
```

The kernel is then *convolved* across the image, calculating a weighted sum for each 3x3 patch of pixels and assigning the result to a new image. It's easier to understand how the filtering works by exploring a step-by-step example.

Let's start with the grayscale image we explored previously:

```
0 0 0 0 0 0 0  
0 0 0 0 0 0 0  
0 0 255 255 255 0 0  
0 0 255 255 255 0 0  
0 0 255 255 255 0 0
```

0	0	0	0	0	0	0
0	0	0	0	0	0	0

First, we apply the filter kernel to the top left patch of the image, multiplying each pixel value by the corresponding weight value in the kernel and adding the results:

$$\begin{aligned} &(0 \times -1) + (0 \times -1) + (0 \times -1) + \\ &(0 \times -1) + (0 \times 8) + (0 \times -1) + \\ &(0 \times -1) + (0 \times -1) + (255 \times -1) = -255 \end{aligned}$$

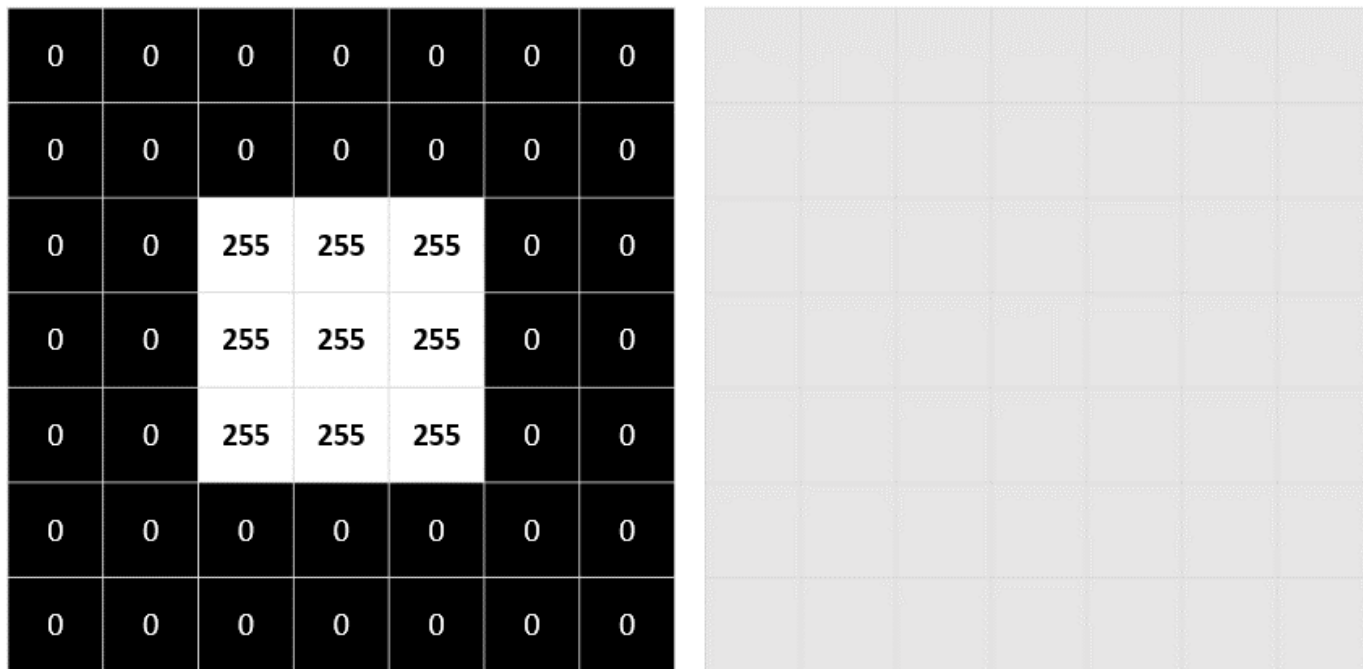
The result (-255) becomes the first value in a new array. Then we move the filter kernel along one pixel to the right and repeat the operation:

$$\begin{aligned} &(0 \times -1) + (0 \times -1) + (0 \times -1) + \\ &(0 \times -1) + (0 \times 8) + (0 \times -1) + \\ &(0 \times -1) + (255 \times -1) + (255 \times -1) = -510 \end{aligned}$$

Again, the result is added to the new array, which now contains two values:

-255 -510

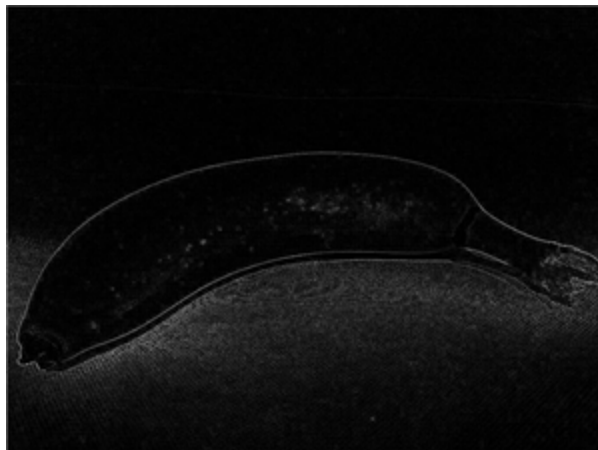
The process is repeated until the filter has been convolved across the entire image, as shown in this animation:



The filter is convolved across the image, calculating a new array of values. Some of the values might be outside of the 0 to 255 pixel value range, so the values are adjusted to fit into that range. Because of the shape of the filter, the outside edge of pixels isn't calculated, so a padding value (usually 0) is applied. The resulting array represents a new image in which the filter has transformed the original image. In this case, the filter has had the effect of highlighting the *edges* of shapes in the image.

To see the effect of the filter more clearly, here's an example of the same filter applied to a real image:

[Expand table](#)

Original Image**Filtered Image**

Because the filter is convolved across the image, this kind of image manipulation is often referred to as *convolutional filtering*. The filter used in this example is a particular type of filter (called a *laplace* filter) that highlights the edges on objects in an image. There are many other kinds of filter that you can use to create blurring, sharpening, color inversion, and other effects.

Next unit: Machine learning for computer vision

[Continue >](#)