PyImageSearch Gurus Course

1.5: Kernels

Take a second to refresh your memory on **Module 1.2** on Image Basics (https://gurus.pyimagesearch.com/lessons/image-basics/).

Remember how we discussed that an image is simply a *matrix*?

Well if we think of an image as a ***big*** matrix, then we can think of a *kernel* or *convolutional matrix* as a ***tiny*** matrix that is used for blurring, sharpening, edge detection, and other image processing functions.

Essentially, this *tiny* kernel sits on top of the *big* image and slides from left to right and up to down, applying a mathematical operation at each *(x, y)*-coordinate in the original image. Again, by applying kernels to images we are able to blur and sharpen them, similar to if we were editing an image in Photoshop.

We can also use convolution to extract features from images and build very powerful deep learning systems. But we'll cover that in **Module 8** (https://gurus.pyimagesearch.com/topic/introduction-to-neural-networks/) of this course.

In the rest of this lesson we'll perform a high-level overview of kernels while constructing a solid foundation that we can build upon. This foundation will be crucial when we get to topics such as **Smoothing and blurring** (https://gurus.pyimagesearch.com/lessons/smoothing-and-blurring/), **Morphological operations** (https://gurus.pyimagesearch.com/lessons/morphological-operations/), and **Edge detection** (https://gurus.pyimagesearch.com/lessons/gradients-and-edge-detection/) — all of these image processing methods are rooted in the usage of kernels

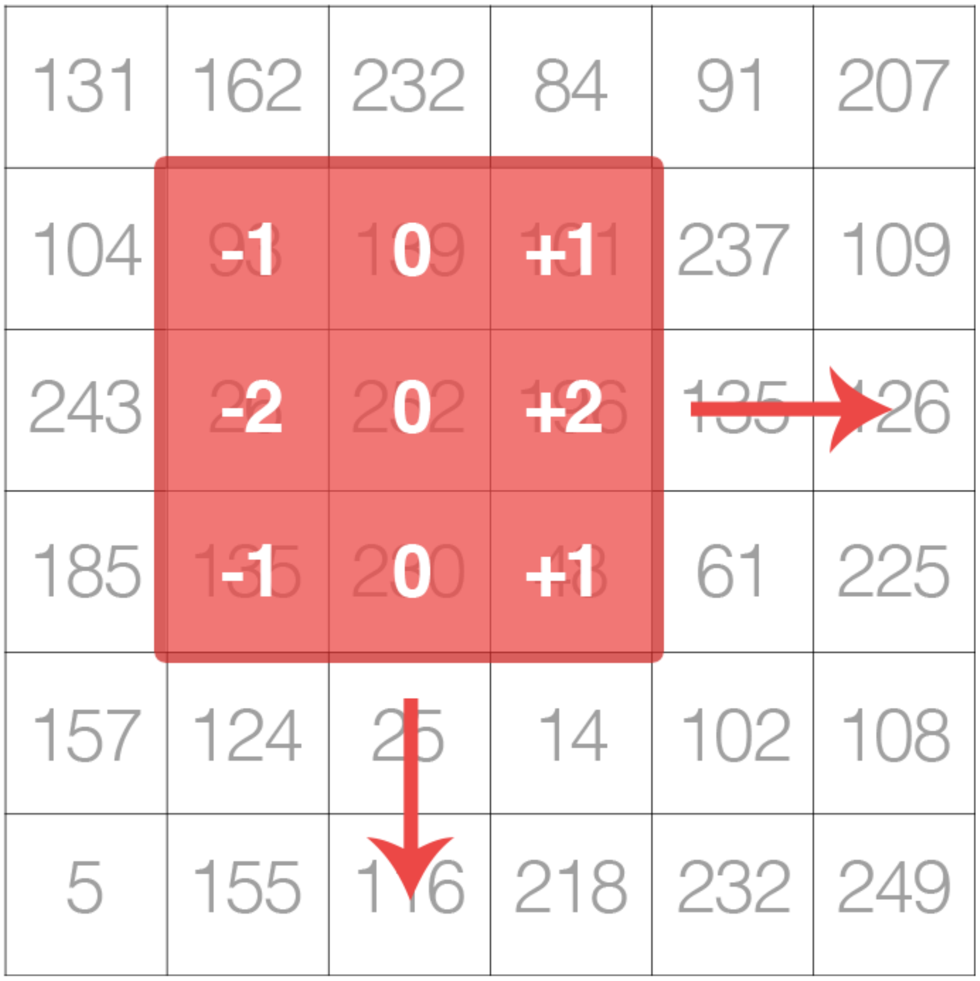
Objective:

By the end of this lesson you should have:

1. A high level understanding of what kernels are and what they can be used to accomplish.
2. An understanding of the term *convolution* and how we convolve images with kernels.

Image Kernels

So again, let's think of an image as a ***big*** matrix and a kernel as ***tiny*** matrix (at least in respect to the original "big matrix" image):



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/04/kernel_sliding.png)

FIGURE 1: A KERNEL IS A SMALL MATRIX THAT SLIDES ACROSS FROM LEFT-TO-RIGHT AND TOP-TO-BOTTOM OF A LARGER IMAGE. AT EACH PIXEL IN THE INPUT IMAGE, THE NEIGHBORHOOD OF THE IMAGE IS CONVOLVED WITH THE KERNEL AND THE OUTPUT STORED.

Feedback

As you can see from the above figure, we are sliding this kernel from left-to-right and top-to-bottom along the original image. At each *(x, y)*-coordinate of the original image we stop and examine the neighborhood of image pixels located at the ***center*** of the image kernel.

We can take this neighborhood of pixels, *convolve* them with the kernel, and we get a single output value. This output value is then stored in the output image at the same *(x, y)*-coordinate as the center of the kernel.

If this sounds a little confusing, don't worry — we'll be reviewing an example later on in this lesson that will make everything crystal clear.

But before we dive into an example, let's first take a look at what a kernel looks like:

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

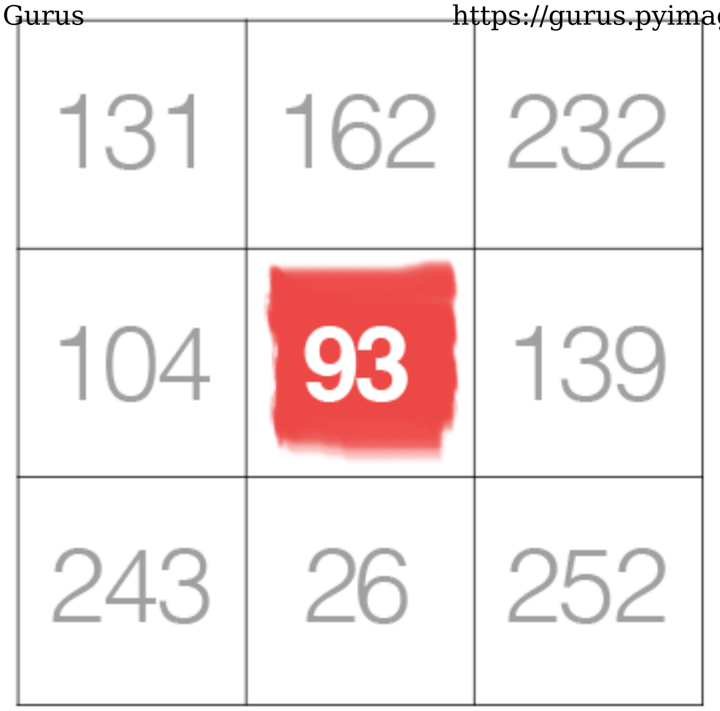
Challenge yourself: *What do you think this kernel does? It seems to be taking an average of some sort. What could we use this kernel for?*

Above we have defined a square 3×3 kernel. Kernels can be an arbitrary size of $M \times N$ pixels, provided that both *M* and *N* are ***odd integers***.

Feedback

Why do both *M* and *N* need to be odd?

Take a look at our introduction to kernels above — the kernel must have a ***center*** *(x, y)*-coordinate. In a 3×3 kernel, the center is located at *(1, 1)*, assuming a zero-index array of course:

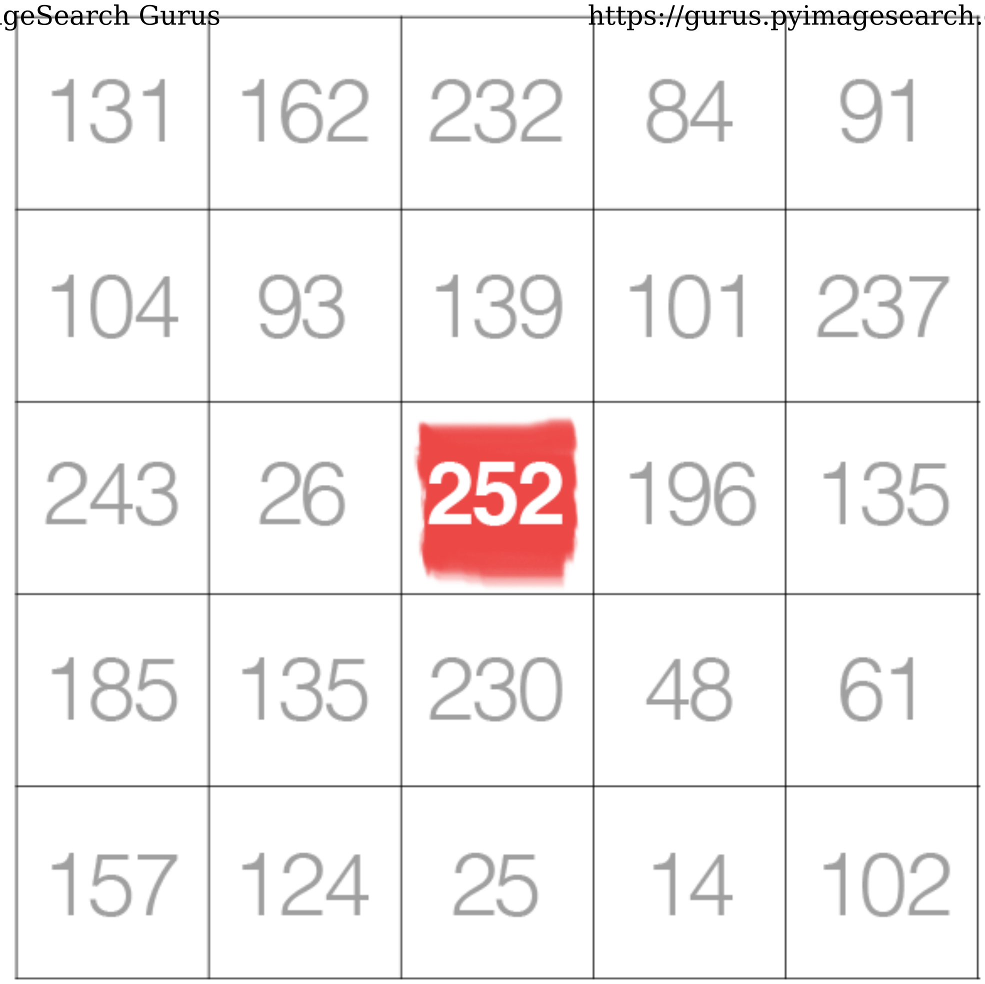


(https://gurus.pyimagesearch.com/wp-content/uploads/2015/04/kernel_3x3.png)

FIGURE 2: THE CENTER PIXEL OF A *3 X 3* KERNEL IS LOCATED AT COORDINATE (1, 1).

As we can see, the center pixel value of *93* located at coordinate *(1, 1)* is highlighted in red.

For a 5×5 kernel, the center is located at *(2, 2)*:



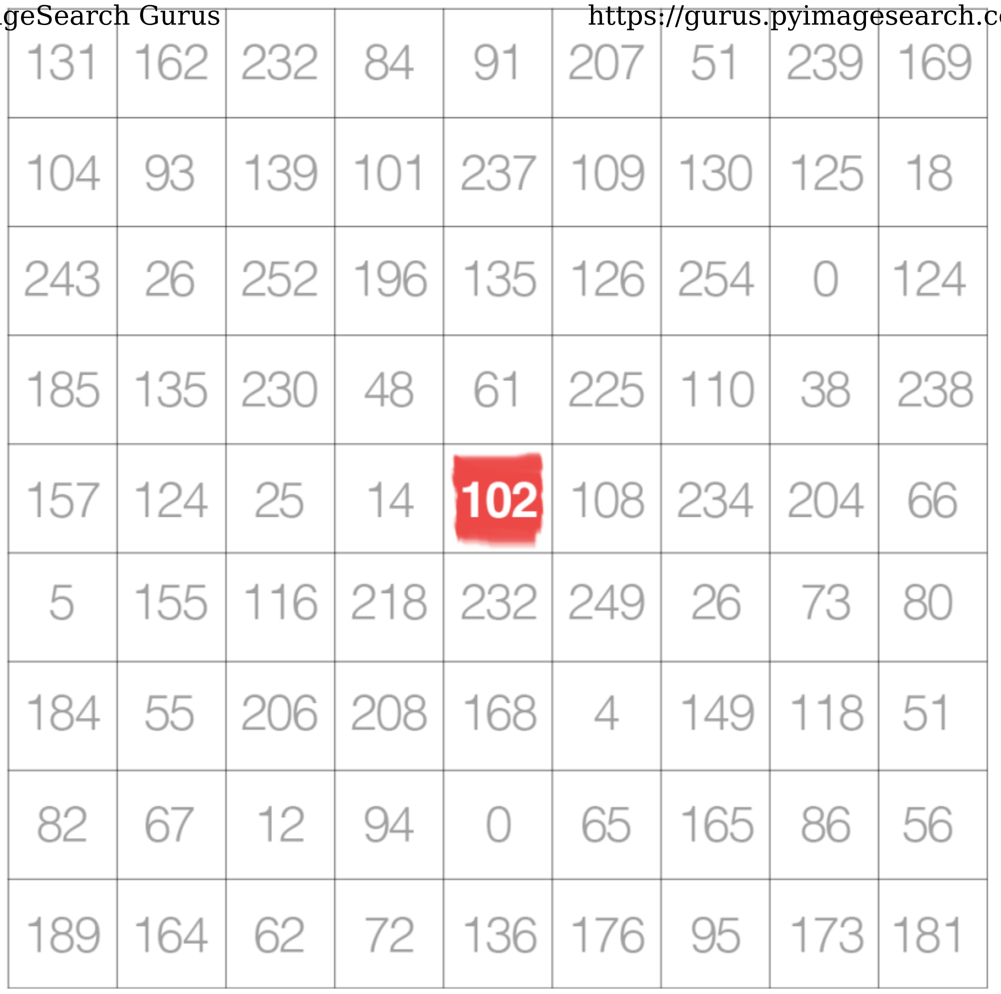
131	162	232	84	91
104	93	139	101	237
243	26	252	196	135
185	135	230	48	61
157	124	25	14	102

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/04/kernel_5x5.png)

FIGURE 3: A VISUALIZATION OF A *5 X 5 KERNEL*, WITH THE CENTER PIXEL LOCATED AT *(2, 2)*.

Feedback

And for a 9×9 kernel, the center is located at *(4, 4)*:



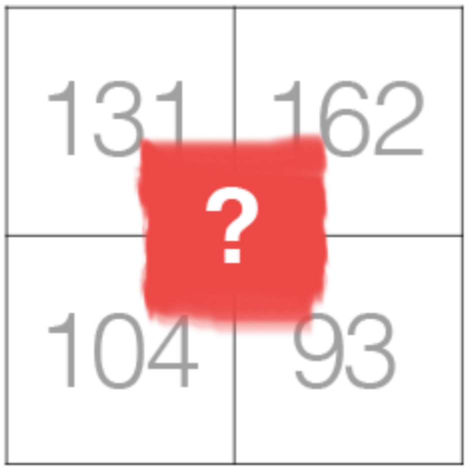
131	162	232	84	91	207	51	239	169
104	93	139	101	237	109	130	125	18
243	26	252	196	135	126	254	0	124
185	135	230	48	61	225	110	38	238
157	124	25	14	102	108	234	204	66
5	155	116	218	232	249	26	73	80
184	55	206	208	168	4	149	118	51
82	67	12	94	0	65	165	86	56
189	164	62	72	136	176	95	173	181

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/04/kernel_9x9.png)

FIGURE 4: THE CENTER PIXEL, HIGHLIGHTED IN RED, OF A *9 X 9* KERNEL IS LOCATED AT COORDINATE *(4, 4)*.

Thus the pattern continues.

But what if we defined a kernel of size 2×2:



131	162
104	93

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/04/kernel_2x2.png)

FIGURE 5: WHAT IS THE CENTER COORDINATE OF A KERNEL OF SIZE (2, 2)?

What is the center coordinate? Again, assuming a zero-index array, the center coordinate would actually be located at *(0.5, 0.5)*. But as we know, without any type of interpolation, there is no such thing as pixel location *(0.5, 0.5)* in an image. This is exactly why we use ***odd*** kernel sizes — to always ensure there is a valid *(x, y)*-coordinate at the center of the kernel.

Convolution

Now that we have discussed the basics of a kernel, let's talk about the mathematical term *convolution*.

In image processing, convolution requires three components, as we described in **Figure 1** of this lesson:

1. An input image.
2. A kernel matrix that we are going to apply to the input image.
3. An output image to store the output of the input image convolved with the kernel.

Feedback

Convolution itself is very easy. All we need to do is:

1. Select an *(x, y)*-coordinate from the original image.
2. Place the **center** of the kernel at this *(x, y)* coordinate.
3. Multiply each kernel value by the corresponding input image pixel value — and then take the sum of all multiplication operations. (More simply put, we're taking the element-wise multiplication of the input image region and the kernel, then summing the values of all these multiplications into a single value. The sum of these multiplications is called the **kernel output**.)
4. Use the same *(x, y)*-coordinate from **Step 1**, but this time store the kernel output in the same *(x, y)*-location as the output image.

Here is an example of convolving (which is normally denoted mathematically as the $\star$ operator) a 3×3 region of an image with a 3×3 kernel:

$$O_{i,j} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \star \begin{bmatrix} 93 & 139 & 101 \\ 26 & 252 & 196 \\ 135 & 230 & 18 \end{bmatrix} = \begin{bmatrix} -1 \times 93 & 0 \times 139 & 1 \times 101 \\ -2 \times 26 & 0 \times 252 & 2 \times 196 \\ -1 \times 135 & 0 \times 230 & 1 \times 18 \end{bmatrix}$$

Therefore,

$$O_{i,j} = \sum \begin{bmatrix} -93 & 0 & 101 \\ -52 & 0 & 392 \\ -135 & 0 & 18 \end{bmatrix} = 231$$

After applying this convolution, we would set the pixel located at the coordinate *(i, j)* of the output image *O* to $O_{i,j} = 231$.

Again, all we are doing is multiplying each of the elements together and taking the sum.

That's all there is to it! Convolution is simply the sum of element-wise matrix multiplication between the kernel and the neighborhood that the kernel covers of the input image.

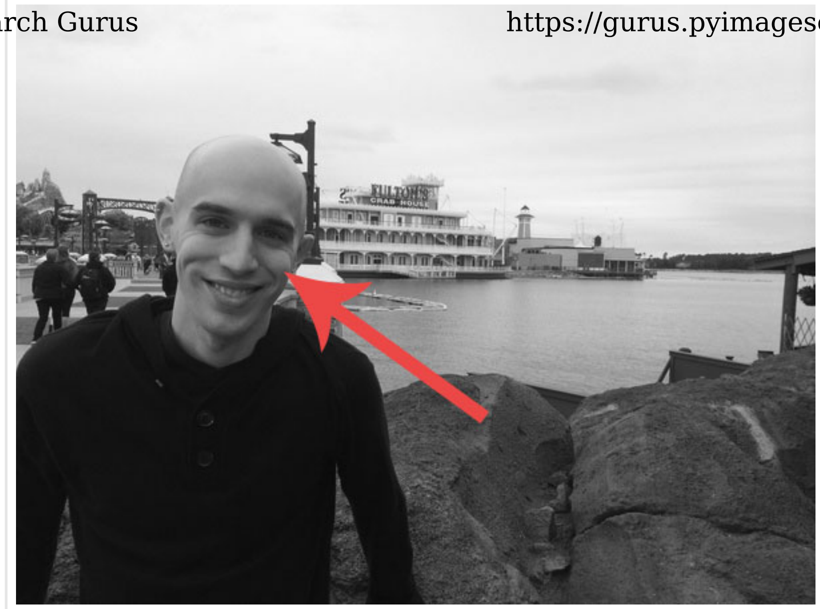
Example Image Convolution

While the example above was easy to understand, let's actually apply our convolution to real image:

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/04/kernel_florida_trip_grayscale.jpg)

FIGURE 6: OUR GRAYSCALE INPUT IMAGE THAT WE ARE GOING TO SMOOTH AND BLUR USING A KERNEL.

In the figure above we have our input image. Our goal here is to *blur* and *smooth* this image. Specifically, let's look at this pixel located at *(X, Y)* and extract the 3×3 region surrounding it:



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/04/kernels_arrow_selection.jpg)

FIGURE 7: LET'S EXAMINE THE *3 X 3* PIXEL NEIGHBORHOOD OF THE PIXEL LOCATED AT *(200, 200)*.

The pixel intensities of this 3×3 region are:

$$K = \begin{bmatrix} 85 & 84 & 81 \\ 80 & 80 & 78 \\ 76 & 77 & 75 \end{bmatrix}$$

So given this 3×3 region, how are we going to go about blurring it? By using kernels and convolution of course!

Let's define our image kernel just like we did at the top of this lesson:

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

This kernel is actually used to ***average*** all pixels together inside the 3×3 area. We'll discuss this in more detail in the **Smoothing and blurring** (https://gurus.pyimagesearch.com/lessons/smoothing-and-blurring/) lesson, but for the time being understand that weighting each element of the image equally and then taking the sum is exactly the same as taking the average — thus we can "average" pixels together in a 3×3 region and smooth the image.

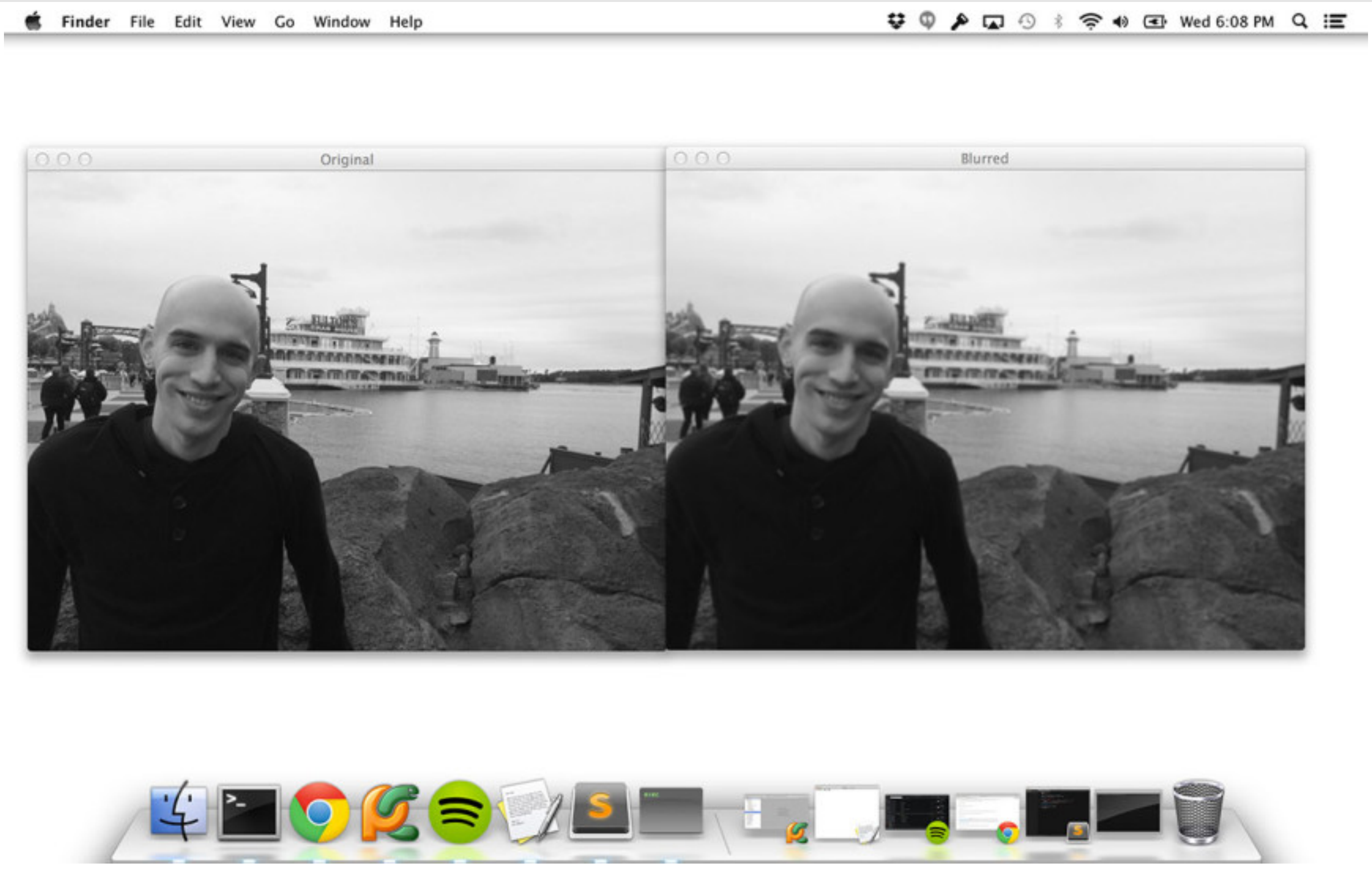
Alright, now that we have our image region and our kernel, let's perform the convolution:

$$O_{200,200} = \sum(\begin{bmatrix} 85 & 84 & 81 \\ 80 & 80 & 78 \\ 76 & 77 & 75 \end{bmatrix} \star \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}) \approx 79$$

Notice how we are left with a value of *79*. We would then take this value of *79* and store it in our output image at the same *(200, 200)*-coordinate.

Finally, to blur the entire image, we would slide our kernel from left-to-right and top-to-bottom, and process ***all*** *(x, y)*-coordinates in the original image in the same manner.

After processing all pixels in the input image, we would have an output image that looks like this:

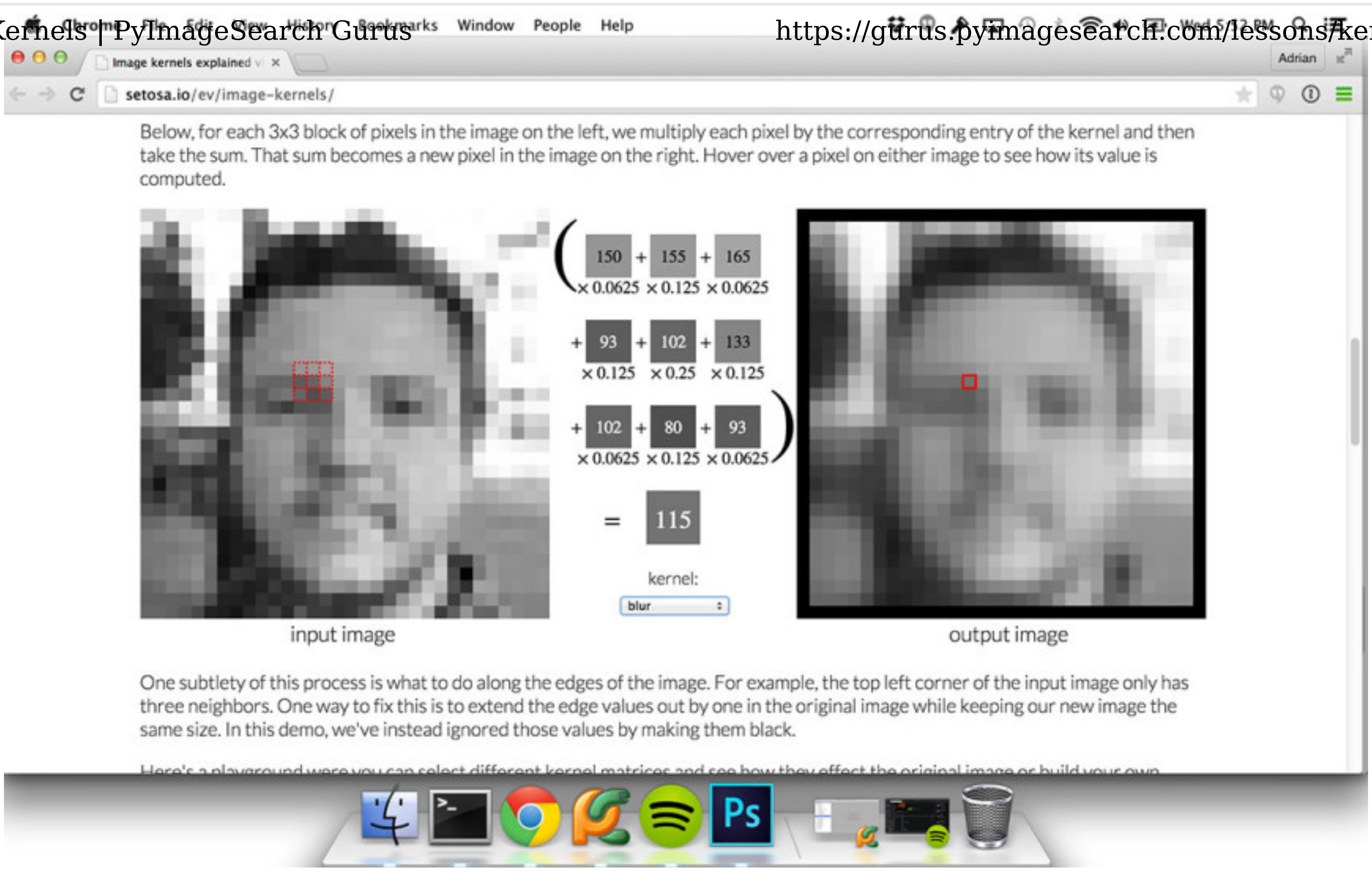


(https://gurus.pyimagesearch.com/wp-content/uploads/2015/04/kernel_blur_applied.jpg)

FIGURE 8: AFTER APPLYING OUR *3 X 3* AVERAGING KERNEL TO OUR INPUT IMAGE *(LEFT)*, WE LEFT WITH A SMOOTHED AND BLURRED VERSION OF THE IMAGE *(RIGHT)*.

Notice how our output image on the *right* looks more blurred and smoothed than the original image on the *left* — this is all from applying our averaging kernel. Specifically, the details on both my face and the boat in the background have been "smoothed" over.

We can also use the kernel visualization tool over at setosa.io (http://setosa.io/ev/image-kernels/) to further cement our understanding of kernels:



Below, for each 3x3 block of pixels in the image on the left, we multiply each pixel by the corresponding entry of the kernel and then take the sum. That sum becomes a new pixel in the image on the right. Hover over a pixel on either image to see how its value is computed.

One subtlety of this process is what to do along the edges of the image. For example, the top left corner of the input image only has three neighbors. One way to fix this is to extend the edge values out by one in the original image while keeping our new image the same size. In this demo, we've instead ignored those values by making them black.

Here's a playground were you can select different kernel matrices and see how they effect the original image or build your own.

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/04/kernel_setosa_example.jpg)

FIGURE 9: USING THE KERNEL VISUALIZATION TOOL OVER AT SETOSA.IO (HTTP://SETOSA.IO/EV/IMAGE-KERNELS/) TO VISUALIZE APPLYING A WEIGHTED GAUSSIAN BLUR TO AN IMAGE.

Feedback

On the *left* we have our original input image, where the red square indicates the location of the image kernel, and each individual cell inside the kernel represents an input image pixel.

In the *center* we have our actual convolution calculation. This time we are applying a Gaussian blur, where pixels further from the center of the kernel contribute less to the overall average. But take notice of the actual calculation — all we are doing is multiplying the input pixels with the kernel values and summing them all together. That's it!

Finally, the output of the convolution is stored in the output image on the *right* — notice how the image is now significantly more smoothed and blurred.

Summary

In this lesson we introduced the concepts of *image kernels* and *convolution*.

If we think of an image as a big matrix, then an image kernel is just a tiny matrix that sits on top of our image. This kernel slides from left-to-right and top-to-bottom, computing the sum of element-wise multiplications between the input image and kernel along the way — we call this value the *kernel output*. The kernel output is then stored in an output image with the same *(x, y)*-coordinate as the input image.

If you're still a little confused about kernels, that's quite understandable. This lesson was simply meant to give you a brief introduction to the concept of kernels. Once we get into topics such as morphological operations, smoothing and blurring, and edge detection, we'll be examining many examples of kernels — and these examples will help solidify your knowledge.

In the meantime, I suggest you play around with the awesome kernel visualization tool over at setosa.io (http://setosa.io/ev/image-kernels/) — this tool is incredibly useful in visualizing the output of kernel operations and can be used to further solidify your understanding of image kernels.

Quizzes		Status
1	Kernels Quiz (https://gurus.pyimagesearch.com/quizzes/kernels-quiz/)	

← Previous Lesson (https://gurus.pyimagesearch.com/lessons/basic-image-processing/) Next Lesson → (https://gurus.pyimagesearch.com/lessons/morphological-operations/)

Feedback

Course Progress

Ready to continue the course?

Click the button below to **continue your journey to computer vision guru**.

I'm ready, let's go! (/pyimagesearch-gurus-course/)

Resources & Links

- PyImageSearch Gurus Community (https://community.pyimagesearch.com/)
- PyImageSearch Virtual Machine (https://gurus.pyimagesearch.com/pyimagesearch-virtual-machine/)
- Setting up your own Python + OpenCV environment (https://gurus.pyimagesearch.com/setting-up-your-python-opencv-development-environment/)

- Course Syllabus & Content Release Schedule (https://gurus.pyimagesearch.com/course-syllabus-content-release-schedule/)
- Member Perks & Discounts (https://gurus.pyimagesearch.com/pyimagesearch-gurus-discounts-perks/)
- Your Achievements (https://gurus.pyimagesearch.com/achievements/)
- Official OpenCV documentation (http://docs.opencv.org/index.html)

Your Account

- Account Info (https://gurus.pyimagesearch.com/account/)
- Support (https://gurus.pyimagesearch.com/contact/)
- Logout (https://gurus.pyimagesearch.com/wp-login.php?action=logout&redirect_to=https%3A%2F%2Fgurus.pyimagesearch.com%2F&_wpnonce=5736b21cae)

Search

Feedback