06-blurring

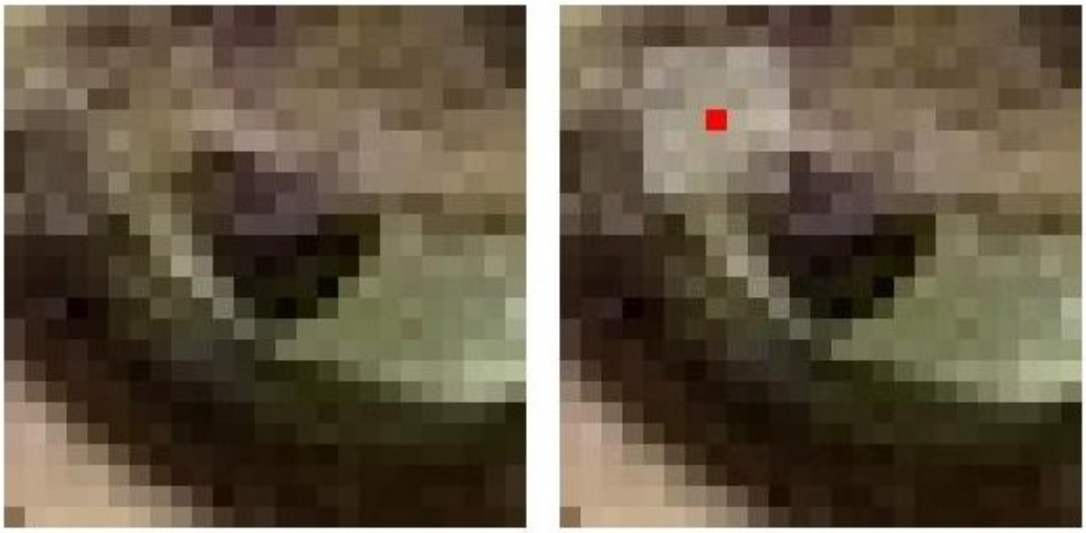
January 14, 2020

1 Gaussian Blur

In this episode, we will learn how to use skimage functions to blur images. When we blur an image, we make the color transition from one side of an edge in the image to another smooth rather than sudden. The effect is to average out rapid changes in pixel intensity. The blur, or smoothing, of an image removes "outlier" pixels that may be noise in the image. Blurring is an example of applying a low-pass filter to an image. In computer vision, the term "low-pass filter" applies to removing noise from an image while leaving the majority of the image intact. A blur is a very common operation we need to perform before other tasks such as edge detection. There are several different blurring functions in the skimage.filters module, so we will focus on just one here, the Gaussian blur.

Consider this image of a cat, in particular the area of the image outlined by the white square.

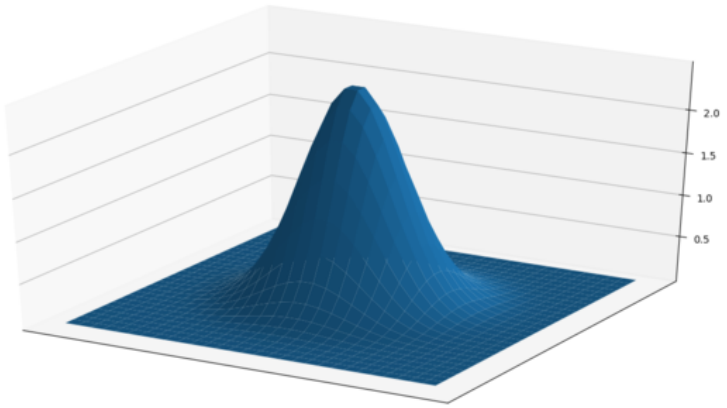
Now, we zoom in on the area of the cat's eye, as shown in the left-hand image below. When we apply a blur filter, we consider each pixel in the image, one at a time. In this example, the pixel we are applying the filter to is highlighted in red, as shown in the right-hand image.



In a blur, we consider a rectangular group of pixels surrounding the pixel to filter. This group of pixels, called the *kernel*, moves along with the pixel that is being filtered. So that the filter pixel is always in the center of the kernel, the width and height of the kernel must be odd. In the example shown above, the kernel is square, with a dimension of seven pixels.

To apply this filter to the current pixel, a weighted average of the the color values of the pixels in the kernel is calculated. In a Gaussian blur, the pixels nearest the center of the kernel are given more weight than those far away from the center. This averaging is done on a channel-by-channel basis, and the average channel values become the new value for the filtered pixel. Larger kernels have more values factored into the average, and this implies that a larger kernel will blur the image more than a smaller kernel.

To get an idea of how this works, consider this plot of the two-dimensional Gaussian function:



Imagine that plot overlaid over the kernel for the Gaussian blur filter. The height of the plot corresponds to the weight given to the underlying pixel in the kernel. I.e., the pixels close to the center become more important to the filtered pixel color than the pixels close to the edge of the kernel. The shape of the Gaussian function is controlled via its standard deviation, or sigma. A large sigma value results in a flatter shape, while a smaller sigma value results in a more pronounced peak. The mathematics involved in the Gaussian blur filter are not quite that simple, but this explanation gives you the basic idea.

To illustrate the blur process, consider the blue channel color values from the seven-by-seven kernel illustrated above:

```
68  82 71 62 100  98  61
90  67 74 78  91  85  77
50  53 78 82  72  95 100
87  89 83 86 100 116 128
89 108 86 78  92  75 100
90  83 89 73  68  29  18
77 102 70 57  30  30  50
```

The filter is going to determine the new blue channel value for the center pixel – the one that currently has the value 86. The filter calculates a weighted average of all the blue channel values in the kernel, {76, 83, 81, . . . , 39, 53, 68}, giving higher weight to the pixels near the center of the kernel. This weighted average would be the new value for the center pixel. The same process would be used to determine the green and red channel values, and then the kernel would be moved over to apply the filter to the next pixel in the image.

Something different needs to happen for pixels near the edge of the image, since the kernel for the filter may be partially off the image. For example, what happens when the filter is applied to the upper-left pixel of the image? Here are the blue channel pixel values for the upper-left pixel of the cat image, again assuming a seven-by-seven kernel:

```
x   x   x   x   x   x   x
x   x   x   x   x   x   x
x   x   x   x   x   x   x
x   x   x   4   5   9   2
x   x   x   5   3   6   7
x   x   x   6   5   7   8
x   x   x   5   4   5   3
```

The upper-left pixel is the one with value 4. Since the pixel is at the upper-left corner. there are no pixels underneath much of the kernel; here, this is represented by x's. So, what does the filter do in that situation?

The default mode is to fill in the *nearest* pixel value from the image. For each of the missing x's the image value closest to the x is used:

```
x   x   x   4   x   x   x
x   x   x   4   x   x   x
x   x   x   4   x   x   x
4   4   4   4   5   9   2
x   x   x   5   3   6   7
x   x   x   6   5   7   8
```

```
x   x   x   5   4   5   3
```

Another strategy to fill those missing values is to *reflect* the pixels that are in the image to fill in for the pixels that are missing from the kernel. If we fill in a few of the missing pixels, you will see how this works:

```
x   x   x   5   x   x   x
x   x   x   6   x   x   x
x   x   x   5   x   x   x
2   9   5   4   5   9   2
x   x   x   5   3   6   7
x   x   x   6   5   7   8
x   x   x   5   4   5   3
```

A similar process would be used to fill in all of the other missing pixels from the kernel. Other *border modes* are available; you can learn more about them in the skimage documentation.

This animation shows how the blur kernel moves along in the original image in order to calculate the color channel values for the blurred image.

skimage has built-in functions to perform blurring for us, so we do not have to perform all of these mathematical operations ourselves.

1.0.1 Gaussian Blur with skimage

[ ]:
```
import skimage
from skimage import filters

import matplotlib.pyplot as plt
# 'magic' to display plots in the jupyter notebook
%matplotlib inline

original = skimage.io.imread('../fig/06-gaussian-original.png')

plt.figure()
plt.imshow(original)

# apply the filter

blurred = filters.gaussian(original, sigma = 10)
plt.figure()
plt.imshow(blurred)
```

1.0.2 Exercise: Experimenting with sigma values

Use different Kernel widths and observe the effect of the filter on the image. Assess by eye the size of the different features in the image (lines, text, blue circle) and try to link the sigma values you use with the effect on each of these features.

Hint: You can use the slice operator to manually 'zoom in' onto the different features of the image and assess their size in pixels respectively, e.g. plt.imshow(original[:10,:20]) let's you inspect the widths of the lines.

[ ]:
```
%load ../exercises/06-GaussianBlur.py
```

[ ]:
```
# sigma to blur out the lines
blurred25 = filters.gaussian(original, sigma = 25)
plt.imshow(blurred25)
```

[ ]:
```
# sigma to blur out the text
blurred50 = filters.gaussian(original, sigma = 50)
plt.imshow(blurred50)
```

[ ]:
```
# sigma to blur out the circle, note the filter-edge effects in top corners!
blurred200 = filters.gaussian(original, sigma = 200)
plt.imshow(blurred200)
```

[ ]:
```
# show Gaussian Kernel width (and mention isotropicity)
# at ../fig/06-gaussian-kernel-width.png
```

1.0.3 Other methods of blurring

The Gaussian blur is a way to apply a low-pass filter in skimage. It is often used to remove Gaussian (i. e., random) noise from the image. For other kinds of noise, e.g. "salt and pepper" or "static" noise, a median filter is typically used. See the skimage.filter documentation for a list of available filters.

1.0.4 Exercise: Blurring Bacteria Colonies

As we move further into the workshop, we will see that in order to complete the colony-counting morphometric challenge at the end, we will need to read the bacteria colony images as grayscale, and blur them, before moving on to the tasks of actually counting the colonies.

- Create a Python function to read one of the colony images as grayscale, and then apply a Gaussian blur to the image. You should also provide the sigma for the blur as a second function argument.
- Do not alter the original image, use skimage.io.imsave(...) to save your results locally.
- As a reminder, the images are located in ../data/coloniesX.tif

Hint: You can use string methods to automatically define an output file name given the input file name. Maybe do a 'dry run' first, meaning that instead of saving the blurred image to disk, just display the result and print the output file name. If that works as expected, save the blurred image to your hard drive using `skimage.io.imsave`.

[ ]:
```
import os
def blur_image(file_name, sigma):

    orig = skimage.io.imread(os.path.join('../data/', file_name), as_gray = True)

    blurred = filters.gaussian(orig, sigma)

    orig_name = file_name.split('.')[0] # chop of the '.tif'
    out_name = orig_name + '_blurred' + '.tif'

    output_path = os.path.join("./", out_name)

    print(f'writing to {output_path}')

    skimage.io.imsave(output_path, blurred)

    return blurred
```

[ ]:
```
b = blur_image('colonies01.tif', 6)
plt.imshow(b, cmap = 'gray')
```