Blur Filter

February 20, 2018

1 Blurring Filter

1.1 Will Koehrsen wjk68

1.1.1 Gaussian Method

The Gaussian method of blurring also called Gaussian smoothing, reduces noise in an image by assigning a weighted average of surrounding pixel values to each pixel in an image. It is often used as a pre-processing step such as prior to detecting edges to reduce the amount of noise/detail in an image.

Gaussian blurring works by convolving an image with a Gaussian kernel such as

`[[1, 4, 6, 4, 1], [2, 8, 12, 8, 2], [6, 24, 36, 24, 6], [2, 8, 12, 8, 2], [1, 4, 6, 4, 1]]`

and then normalizing the result by the sum of the weights in the filter. The resulting value is thus a weighted average of surrounding pixels values and is assigned to the center pixel to reduce the gradients in the image. The kernel controls the amount of blurring: a larger kernel convolved over the image produces more blurring because it smooths out the pixel values over a larger area. Other kernels, such as a simple box

'[[1 1 1] [1 1 1] [1 1 1]]

'

can also be used for blurring.

The basic process is as follows:

1. Create a filter with larger filters resulting in more blurring.
2. Separate the image into three color channels
3. For each channel, iterate through the rows and columns of the image
4. At each step, element-wise multiply the filter times the patch
5. Take the weighted average of the patch and assign the pixel value to all pixels in the patch
6. Normalize the multiplication results by the sum of weights in the kernel and convert to integers between 0 and 255 for image plotting.
7. Visualize Images
8. Try different filters to adjust the level of blurring

In [3]:
```
# numpy for arrays
import numpy as np

# matplotlib for plotting images
import matplotlib.pyplot as plt
```

```
from matplotlib.image import imread

# PIL for loading images
from PIL import Image
```

1.2 Implementation

```
In [4]: def apply_blur_filter(blur_filter, image_path):

            # Load in the image
            image = Image.open(image_path)

            # Crop to correct size
            image = image.crop(box=(0, 0,
                              int(image.size[0] / blur_filter.shape[0]) * blur_filter.shape[0
                              int(image.size[1] / blur_filter.shape[1]) * blur_filter.shape[1

            im_array = np.array(image)

            # Horizontal and vertical moves, using a stride of filter shape
            h_moves = int(im_array.shape[1] / blur_filter.shape[1])
            v_moves = int(im_array.shape[0] / blur_filter.shape[0])

            new_image = np.zeros(shape = im_array.shape)

            k = np.sum(blur_filter)

            # Iterate through 3 color channels
            for i in range(im_array.shape[2]):
                # Extract the layer and create a new layer to fill in
                layer = im_array[:, :, i]
                new_layer = np.zeros(shape = layer.shape, dtype='uint8')

                # Left and right bounds are determined by columns
                l_border = 0
                r_border = blur_filter.shape[1]


                # Iterate through the number of horizontal and vertical moves
                for h in range(h_moves):
                    # Top and bottom bounds are determined by rows
                    b_border = 0
                    t_border = blur_filter.shape[0]
                    for v in range(v_moves):
                        patch = layer[b_border:t_border, l_border:r_border]

                        # Take the element-wise product of the patch and the filter
                        product = np.multiply(patch, blur_filter)
```

```
                # Find the weighted average of the patch
                product = np.sum(product) / k
                new_layer[b_border:t_border, l_border:r_border] = product

                b_border = t_border
                t_border = t_border + blur_filter.shape[0]

            l_border = r_border
            r_border = r_border + blur_filter.shape[1]


        new_image[:, :, i] = 255 * ( (new_layer - np.min(new_layer)) /
                                    (np.max(new_layer) - np.min(new_layer)) )


    # Convert to correct type for plotting
    new_image = new_image.astype('uint8')

    plt.imshow(image); plt.title('Original Image'); plt.axis('off')
    plt.show()

    plt.imshow(new_image); plt.title('Blurred Image'); plt.axis('off')
    plt.show()

    return new_image
```

1.3 Results

```
In [5]: blur_filter = np.array([[1, 4, 6, 4, 1],
                                [2, 8, 12, 8, 2],
                                [6, 24, 36, 24, 6],
                                [2, 8, 12, 8, 2],
                                [1, 4, 6, 4, 1]])

        blurred_image = apply_blur_filter(blur_filter, 'images/president-barack-obama.jpg')
```

Original Image

Blurred Image

```
In [6]: blur_filter = np.array([[1, 2, 4, 8, 4, 2, 1],
                                [2, 4, 16, 32, 16, 4, 2],
                                [4, 16, 32, 64, 32, 16, 4],
                                [2, 4, 16, 32, 16, 4, 2],
                                [1, 2, 4, 8, 4, 2, 1]])

        blurred_image = apply_blur_filter(blur_filter, 'images/president-barack-obama.jpg')
```

Original Image

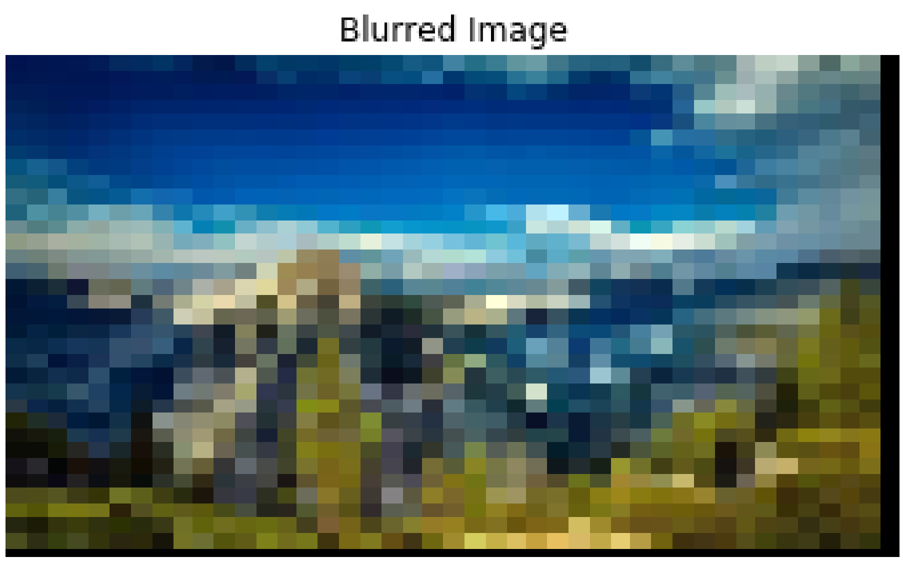
Blurred Image

```
In [7]: blur_filter = np.array([[0, 0, 0, 0, 0],
                                [0, 0, 0, 0, 0],
                                [16, 32, 64, 32, 16],
                                [0, 0, 0, 0, 0],
                                [0, 0, 0, 0, 0]])

        blurred_image = apply_blur_filter(blur_filter, 'images/president-barack-obama.jpg')
```

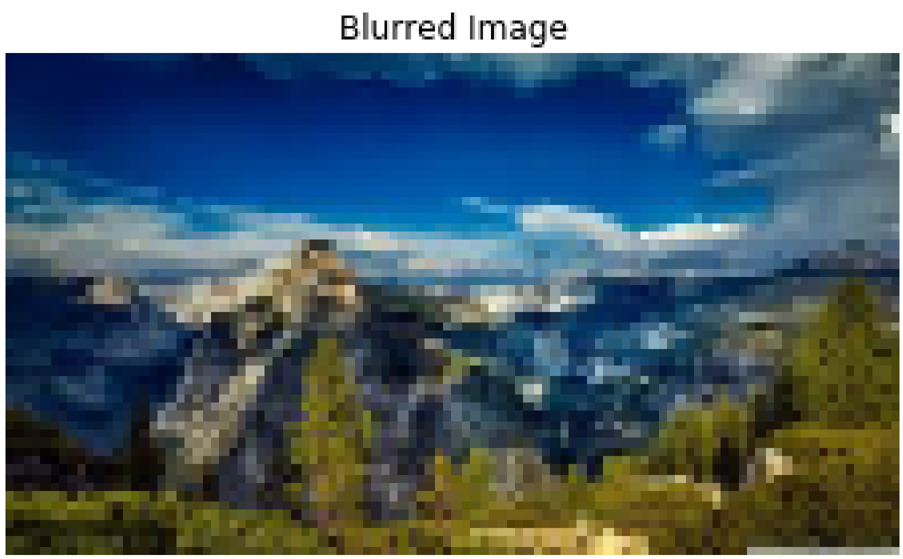
Original Image

Blurred Image

```
In [8]: blur_filter = np.array([[1, 2, 4, 8, 4, 2, 1],
                                [2, 4, 16, 32, 16, 4, 2],
                                [4, 16, 32, 64, 32, 16, 4],
                                [2, 4, 16, 32, 16, 4, 2],
                                [1, 2, 4, 8, 4, 2, 1]])

        blurred_image = apply_blur_filter(blur_filter, 'images/mountains.jpg')
```




```
In [9]: blur_filter = np.array([[1, 2, 1],
                                [2, 4, 2],
                                [1, 2, 1]])

        blurred_image = apply_blur_filter(blur_filter, 'images/mountains.jpg')
```



1.4 Conclusions

We see that by changing the parameters of the blur filter we can control the amount of smoothing in the image. A blur filter is a useful pre-processing step because it reduces the amount of noise in an image which can be helpful for processes such as edge detection and feature recognition.