Machine Learning Guru

(../../../index.php)

Machine Learning and Computer Vision tutorials using open source packages.

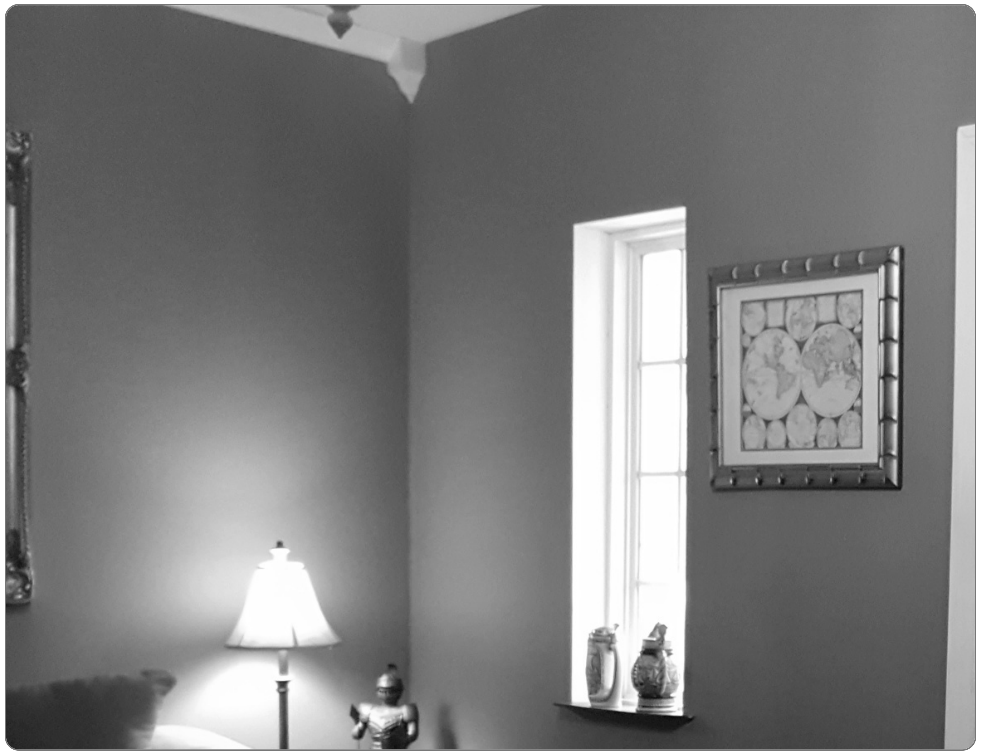
Sections

Image Filtering

A comprehensive tutorial towards 2D convolution and image filtering (The first step to understand Convolutional Neural Networks (CNNs))

Introduction

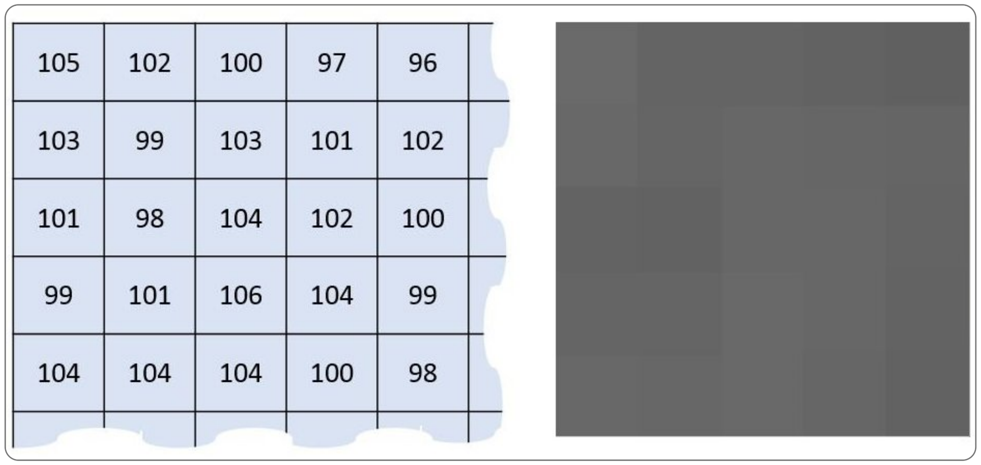
Convolution is one of the most important operations in signal and image processing. It could operate in 1D (e.g. speech processing), 2D (e.g. image processing) or 3D (video processing). In this post, we discuss convolution in 2D spatial which is mostly used in image processing for feature extraction and is also the core block of Convolutional Neural Networks (CNNs). Generally, we can consider an image as a matrix whose elements are numbers between 0 and 255. The size of this matrix is (image height) x (image width) x (# of image channels). A grayscale image has 1 channel where a color image has 3 channels (for an RGB). In this tutorial we are going to work on a grayscale image shown in Figure 1 and apply different convolution kernels on it.



(../../../_images/topics/computer_vision/basics/convolution/image.jpg)

Figure 1: The original grayscale image

If we zoom on the very top-left corner of the image, we can see the pixels of the image. You can see the pixels on the top-left corner of the image (first five rows and five columns) and their corresponding values in Figure 2.



(../../../_images/topics/computer_vision/basics/convolution/7.jpg)

Figure 2: The first 5 columns and rows of the image in Figure 1

You can load and plot the image as a **Numpy** array using **skimage** library in python:

Load and plot an image

Note: *skimage* load grayscale images in [0-1] scale instead of [0-255].

python Code **Output**

```
from skimage import io, viewer
img = io.imread('image.jpg', as_grey=True)  # load the image as grayscale
print 'image matrix size: ', img.shape      # print the size of image
print '\n First 5 columns and rows of the image matrix: \n', img[:5,:5]*255
viewer.ImageViewer(img).show()              # plot the image
```

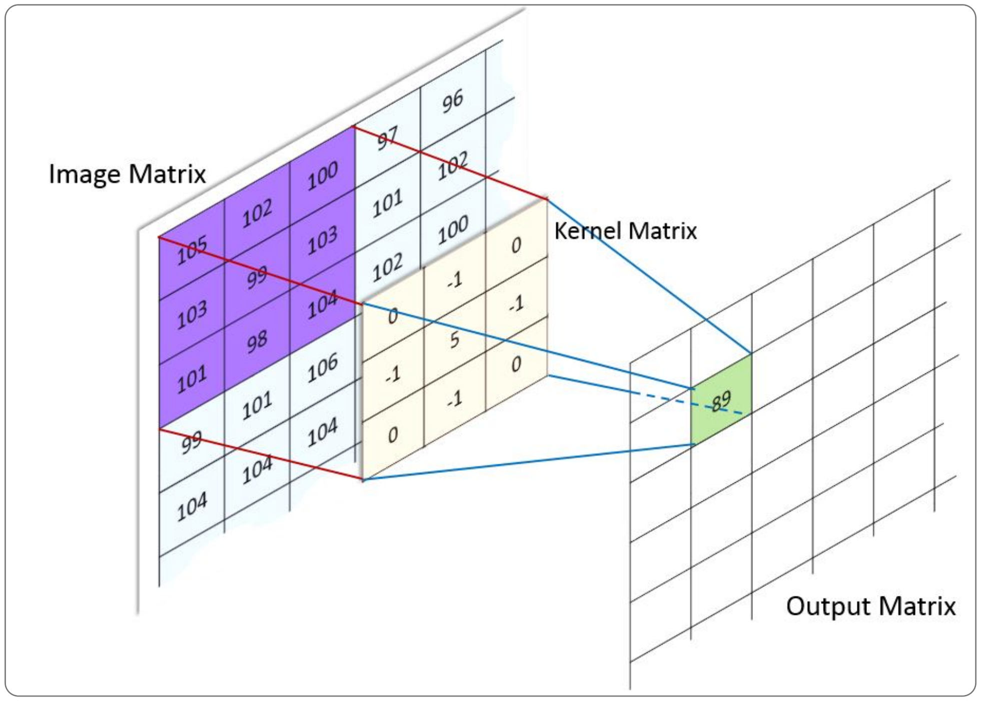
Convolution

Each convolution operation has a kernel which could be a any matrix smaller than the original image in height and width. Each kernel is useful for a spesific task, such as sharpening, blurring, edge detection, and more. Let's start with the sharpening kernel which is defined as:

$$Kernel = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

You can find a list of most common kernels here (https://en.wikipedia.org/wiki/Kernel_(image_processing)). As previously mentioned, each kernel has a specific task to do and the sharpen kernel accentuate edges but with the cost of adding noise to those area of the image which colors are changing gradually. The output of image convolution is calculated as follows:

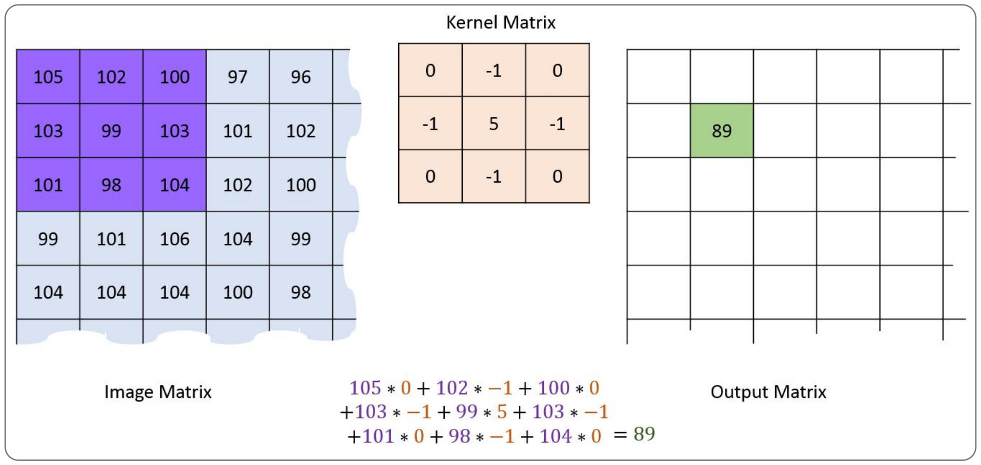
- Flip the kernel both horizontally and vertically. As our selected kernel is symetric, the flipped kernel is equal to the original.
- Put the first element of the kernel at every pixel of the image (element of the image matrix). Then each element of the kernel will stand on top of an element of the image matrix.



(../../../_images/topics/computer_vision/basics/convolution/1.JPG)

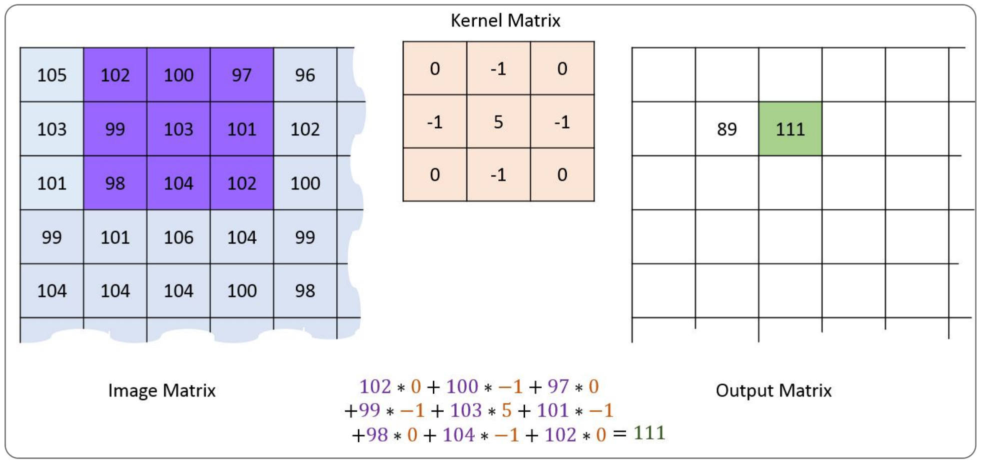
Figure 3: To calculate the value of convolution output at pixel (2,2), center the kernel at the same pixel position on the image matrix

- Multiply each element of the kernel with its corresponding element of the image matrix (the one which is overlapped with it)
- Sum up all product outputs and put the result at the same position in the output matrix as the center of kernel in image matrix.



$105*0+102*-1+100*0$
$+103*-1+99*5+103*-1$
$+101*0+98*-1+104*0 = 89$

(../../../_images/topics/computer_vision/basics/convolution/3.JPG)

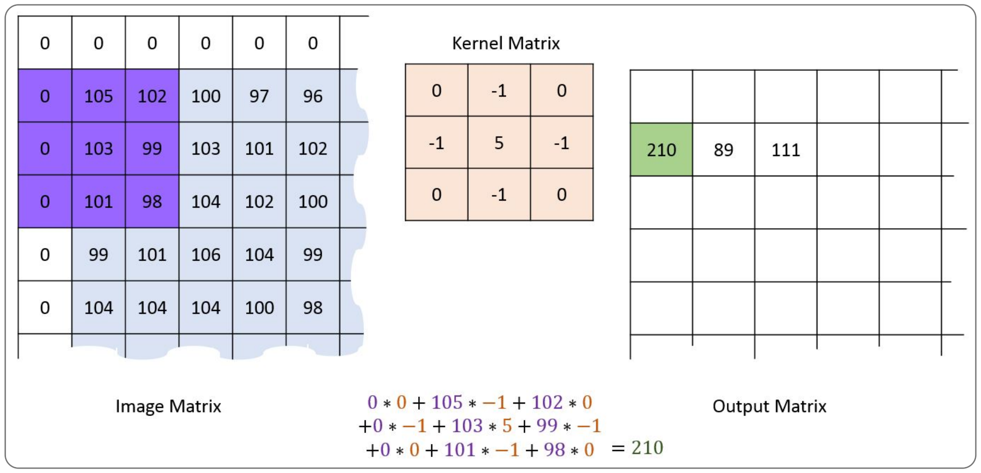


$102*0+100*-1+97*0$
$+99*-1+103*5+101*-1$
$+98*0+104*-1+102*0 = 111$

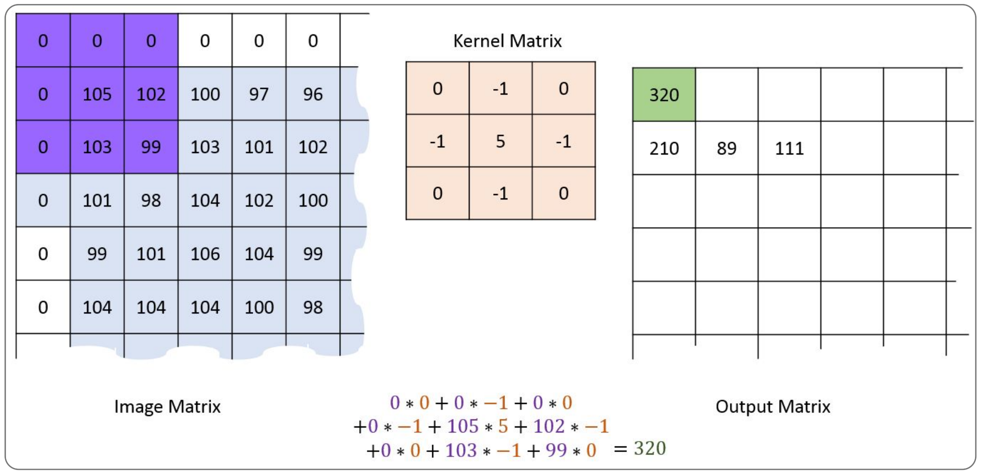
(../../../_images/topics/computer_vision/basics/convolution/4.JPG)

Figure 4: Convolution calculation

- For the pixels on the border of image matrix, some elements of the kernel might stands out of the image matrix and therefore does not have any corresponding element from the image matrix. In this case, you can eliminate the convolution operation for these position which end up an output matrix smaller than the input (image matrix) or we can apply padding to the input matrix (based on the size of the kernel we might need one or more pixels padding, in our example we just need 1 pixel padding):



(../../../_images/topics/computer_vision/basics/convolution/5.JPG)



(../../../_images/topics/computer_vision/basics/convolution/6.JPG)

Figure 5: Convolution calculation on borders

As you can see in Figure 5, the output of convolution might violate the input range of [0-255]. Even though the python packages would take care of it by considering the maximum value of the image as the pure white (correspond to 255 in [0-255] scale) and the minimum value as the pure black (correspond to 0 in [0-255] scale), the values of the convolution output (filtered image) specially along the edges of the image (which are calculated based on the added zero padding) can cause a low contrast filtered image. In this post, to overcome this loss of contrast issue, we use Histogram

Equalization technique. However, you might be able to end up with a better contrast neglecting the zero padding. The following python code convolves an image with the sharpen kernel and plots the result:

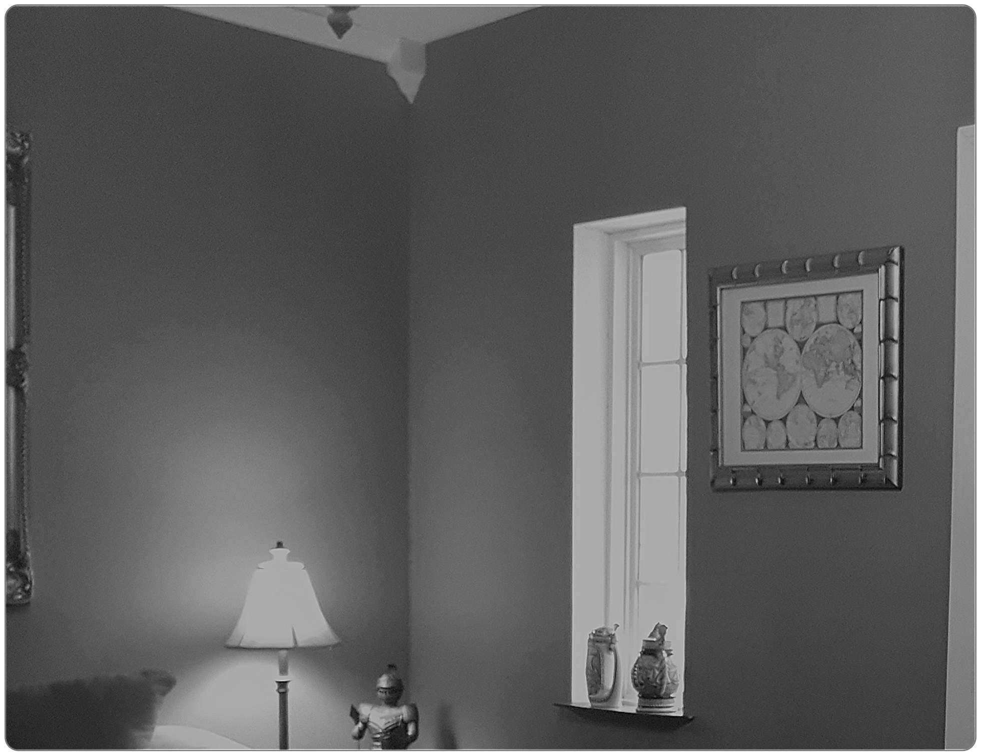
Convolve the sharpen kernel with an image

python Code Output

```
from skimage import io, color
import matplotlib.pyplot as plt
import numpy as np
from skimage import exposure
import pylab

def convolve2d(image, kernel):
    # This function which takes an image and a kernel
    # and returns the convolution of them
     # Args:
     #   image: a numpy array of size [image_height, image_width].
     #   kernel: a numpy array of size [kernel_height, kernel_width].
     # Returns:
     #   a numpy array of size [image_height, image_width] (convolution output).

     kernel = np.flipud(np.fliplr(kernel))    # Flip the kernel
     output = np.zeros_like(image)            # convolution output
     # Add zero padding to the input image
     image_padded = np.zeros((image.shape[0] + 2, image.shape[1] + 2))
     image_padded[1:-1, 1:-1] = image
     for x in range(image.shape[1]):     # Loop over every pixel of the image
         for y in range(image.shape[0]):
             # element-wise multiplication of the kernel and the image
             output[y,x]=(kernel*image_padded[y:y+3,x:x+3]).sum()
     return output

img = io.imread('image.png')    # Load the image
img = color.rgb2gray(img)       # Convert the image to grayscale (1 channel)

# Adjust the contrast of the image by applying Histogram Equalization
image_equalized = exposure.equalize_adapthist(img/np.max(np.abs(img)), clip_limit=0.03)
plt.imshow(image_equalized, cmap=plt.cm.gray)
plt.axis('off')
plt.show()

# Convolve the sharpen kernel and the image
kernel = np.array([[0,-1,0],[-1,5,-1],[0,-1,0]])
image_sharpen = convolve2d(img,kernel)
print '\n First 5 columns and rows of the image_sharpen matrix: \n', image_sharpen[:5,:5]*255
```

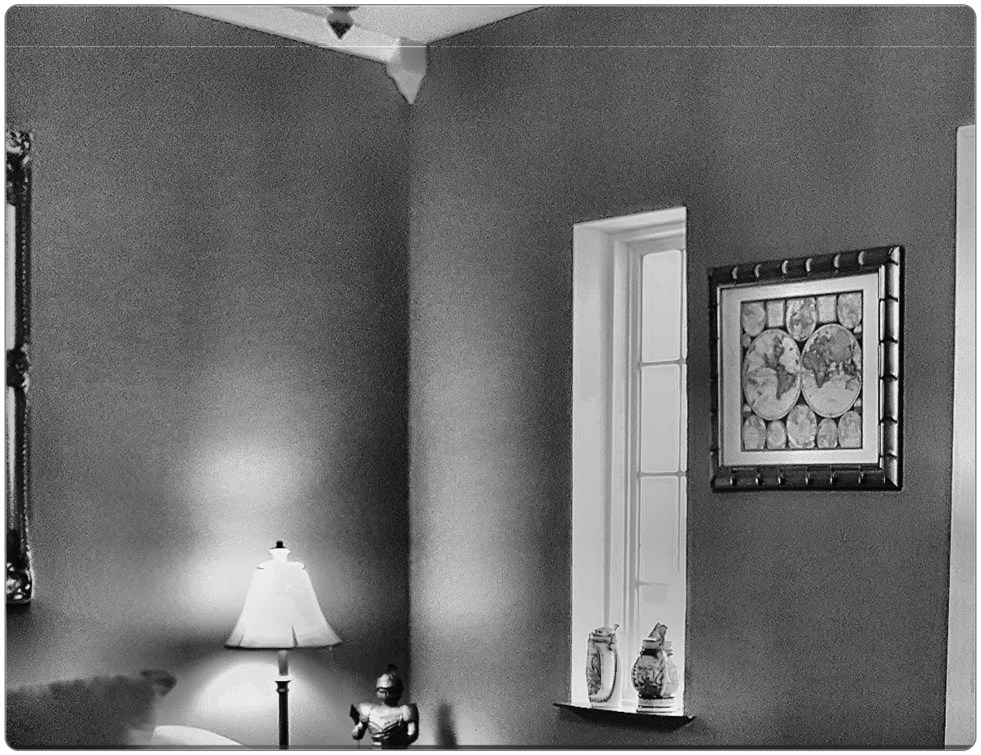
```

# Plot the filtered image
plt.imshow(image_sharpen, cmap=plt.cm.gray)
plt.axis('off')
plt.show()

# Adjust the contrast of the filtered image by applying Histogram Equalization
image_sharpen_equalized = exposure.equalize_adapthist(image_sharpen/np.max(np.abs(image_sharpen)), cli
plt.imshow(image_sharpen_equalized, cmap=plt.cm.gray)
plt.axis('off')
plt.show()
```

and you can see the filtered image after applying **sharpen** filter in Figure 6 and the filtered image after Histogram Equalization in Figure 7.



(../../../_images/topics/computer_vision/basics/convolution/sharpen.jpg)

Figure 6: Sharpened image



(../../../_images/topics/computer_vision/basics/convolution/sharpen_eq.jpg)

Figure 7: Sharpened image after Histogram Equalization

So far, we have been using our own convolution function which was not written to be efficient. Hopefully, you can easily find well written functions for 1D, 2D, and 3D convolutions in most of the python packages which are related to machine learning and image processing. Here is our previous code but using Scipy or OpenCV built-in functions.

Convolve the sharpen kernel with an image using python packages

Scipy OpenCV

```
import numpy as np
import scipy
from skimage import io, color
from skimage import exposure
```

```
import matplotlib.pyplot as plt

img = io.imread('image.png')    # Load the image
img = color.rgb2gray(img)       # Convert the image to grayscale (1 channel)

kernel = np.array([[0,-1,0],[-1,5,-1],[0,-1,0]])

# you can use 'valid' instead of 'same', then it will not add zero padding
image_sharpen = scipy.signal.convolve2d(img, kernel, 'same')
print '\n First 5 columns and rows of the image_sharpen matrix: \n', image_sharpen[:5,:5]*255

# Adjust the contrast of the filtered image by applying Histogram Equalization
image_sharpen_equalized = exposure.equalize_adapthist(image_sharpen/np.max(np.abs(image_sharpen)), cli
plt.imshow(image_sharpen_equalized, cmap=plt.cm.gray)
plt.axis('off')
plt.show()
```

More Filters

There are many other filters which are really useful in image processing and computer vision. One of the most important one is edge detection. Edge detection aims to identify pixels of an image at which the brightness changes drastically. Let's apply one of the simplest edge detection filters to our image and see the result. Here is the kernel:

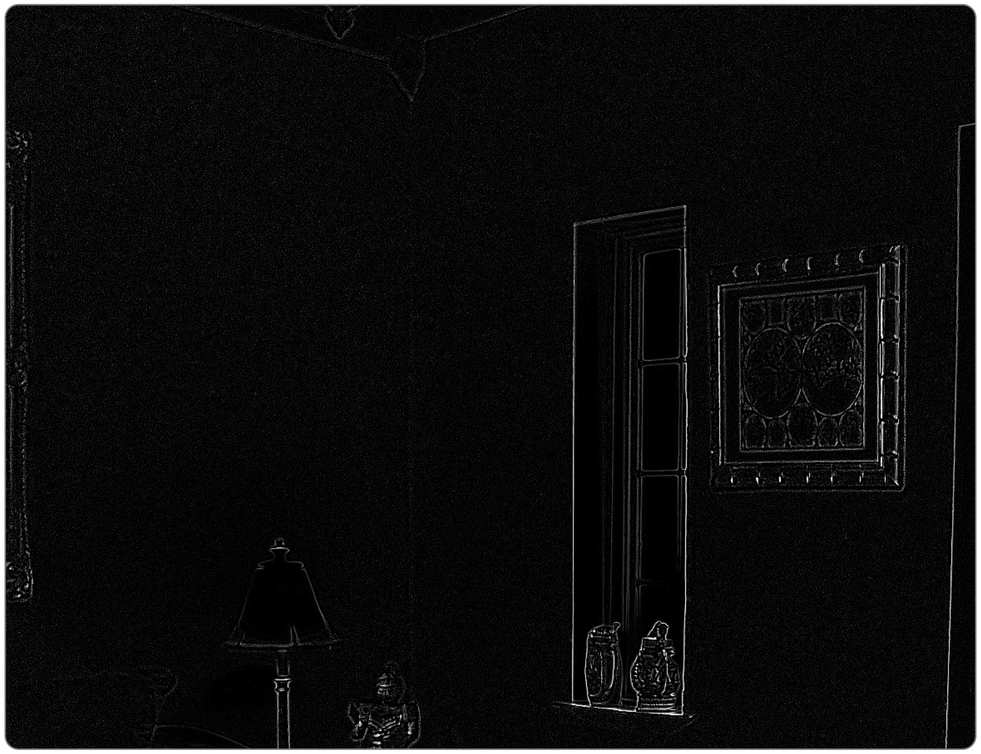
$$Kernel = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

and here is the python code:

Convolve an edge detection kernel with an image

```
import numpy as np
import scipy.signal
import matplotlib.pyplot as plt
from skimage import io, color
from skimage import exposure

img = io.imread('image.png')    # Load the image
img = color.rgb2gray(img)       # Convert the image to grayscale (1 channel)

 kernel = np.array([[-1,-1,-1],[-1,8,-1],[-1,-1,-1]])
 # we use 'valid' which means we do not add zero padding to our image
 edges = scipy.signal.convolve2d(img, kernel, 'valid')
 print '\n First 5 columns and rows of the image_sharpen matrix: \n', image_sharpen[:5,:5]*255

 # Adjust the contrast of the filtered image by applying Histogram Equalization
 edges_equalized = exposure.equalize_adapthist(edges/np.max(np.abs(edges)), clip_limit=0.03)
 plt.imshow(edges_equalized, cmap=plt.cm.gray)    # plot the edges_clipped
 plt.axis('off')
 plt.show()
```

and here is what you will see when you run the code:



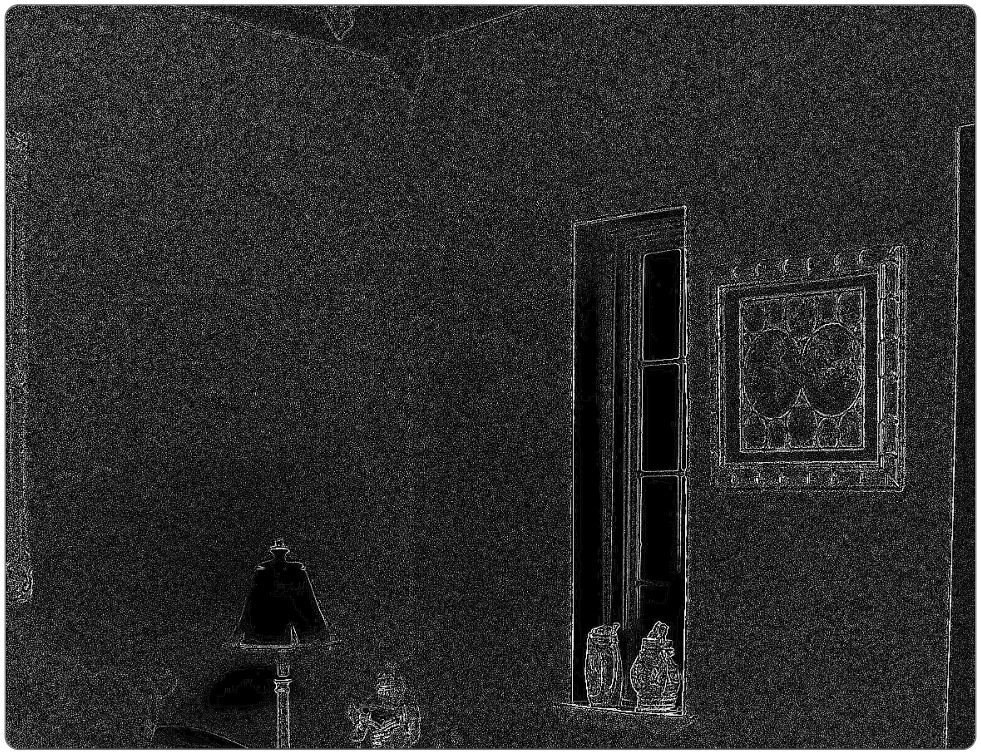
(../../../_images/topics/computer_vision/basics/convolution/edges.jpg)

Figure 7: Filtered image

What about if we apply the edge detection kernel to the output of sharpen filter? Let's have a look at it:

Apply sharpen and edge detection filters back to back

```
import numpy as np
import scipy.signal
import matplotlib.pyplot as plt
from skimage import io, color
from skimage import exposure

img = io.imread('image.png')    # Load the image
img = color.rgb2gray(img)       # Convert the image to grayscale (1 channel)

# apply sharpen filter to the original image
sharpen_kernel = np.array([[0,-1,0],[-1,5,-1],[0,-1,0]])
image_sharpen = scipy.signal.convolve2d(img, sharpen_kernel, 'valid')

edge_kernel = np.array([[-1,-1,-1],[-1,8,-1],[-1,-1,-1]])
edges = scipy.signal.convolve2d(image_sharpen, edge_kernel, 'valid')

# Adjust the contrast of the filtered image by applying Histogram Equalization
edges_equalized = exposure.equalize_adapthist(edges/np.max(np.abs(edges)), clip_limit=0.03)

plt.imshow(edges_equalized, cmap=plt.cm.gray)    # plot the edges_clipped
plt.axis('off')
plt.show()
```



(../../../_images/topics/computer_vision/basics/convolution/edge2.jpg)

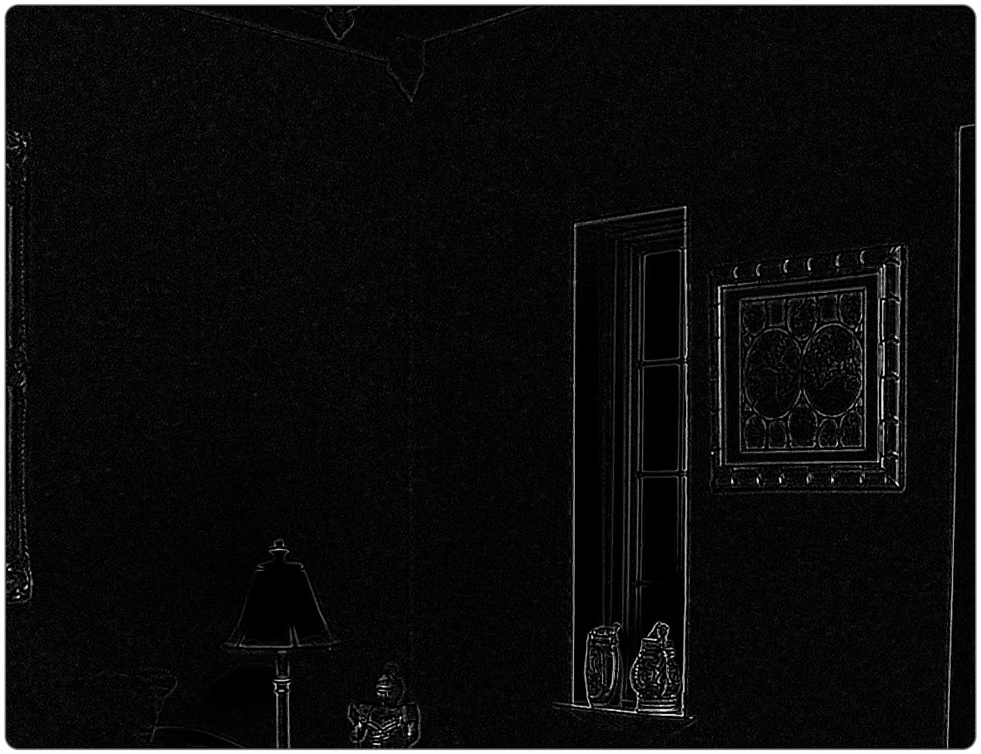
Figure 8: Filtered image

As we mentioned before, sharpen filter bolds the edges but with the cost of adding noise to the image. You can clearly see these effects comparing Figure 8 and Figure 7. Now it's time to apply a filter to the noisy image and reduce the noise. Blur filter could be a smart choise:

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

Apply blur filter to denoise an image

```
import numpy as np
import scipy.signal
import matplotlib.pyplot as plt
from skimage import io, color
from skimage import exposure

img = io.imread('image.png')    # Load the image
img = color.rgb2gray(img)       # Convert the image to grayscale (1 channel)

 # apply sharpen filter to the original image
 sharpen_kernel = np.array([[0,-1,0],[-1,5,-1],[0,-1,0]])
 image_sharpen = scipy.signal.convolve2d(img, sharpen_kernel, 'valid')

 # apply edge detection filter to the sharpen image
 edge_kernel = np.array([[-1,-1,-1],[-1,8,-1],[-1,-1,-1]])
 edges = scipy.signal.convolve2d(image_sharpen, edge_kernel, 'valid')

 # apply blur filter to the edge detection filtered image
 blur_kernel = np.array([[1,1,1],[1,1,1],[1,1,1]])/9.0;
 denoised = scipy.signal.convolve2d(edges, blur_kernel, 'valid')

 # Adjust the contrast of the filtered image by applying Histogram Equalization
 denoised_equalized = exposure.equalize_adapthist(denoised/np.max(np.abs(denoised)), clip_limit=0.03)

 plt.imshow(denoised_equalized, cmap=plt.cm.gray)    # plot the denoised_clipped
 plt.axis('off')
 plt.show()
```



(../../../_images/topics/computer_vision/basics/convolution/blur.jpg)

Figure 9: Denoised Image

Go Top

15 Comments Machine Learning Guru 1 Login

Recommend 8 Tweet Share Sort by Newest

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Михаил Бахрах • 24 days ago

Really only one topic across the web who in a simple way explained convolution logic for person withour ML skills

• Reply • Share ›

llgxvi • 6 months ago

amazing

Related Posts:

(https://twitter.com/M_L_Guru) (https://github.com/Machinelearninguru) (https://www.linkedin.com/groups/12030461)