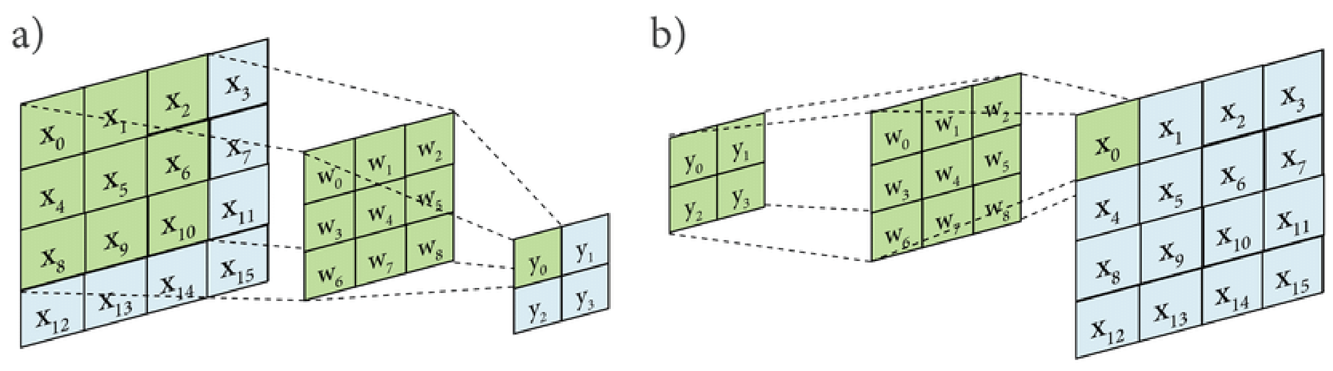
2D Convolution using Python & NumPy

Samrat Sahoo Follow
Jun 17, 2020 · 5 min read


a)
b)
x0 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15
w0 w1 w2 w3 w4 w5 w6 w7 w8
y0 y1 y2 y3

2D Convolutions are instrumental when creating convolutional neural networks or just for general image processing filters such as blurring, sharpening, edge detection, and many more. They are based on the idea of using a kernel and iterating through an input image to create an output image. If you are new to convolutions I would high reccommend the playlist by deeplearning.ai on convolutional neural networks.

In this article we will be implementing a 2D Convolution and then applying an edge detection kernel to an image using the 2D Convolution.

Imports

For this implementation of a 2D Convolution we will need 2 libraries:

```
import cv2
import numpy as np
```

OpenCV will be used to pre-process the image while NumPy will be used to implement the actual convolution.

Pre-process Image

In order to get the best results with a 2D convolution, it is generally recommended that you process the image in grayscale. To do this we can write a method. Let's start with:

```
def processImage(image):
```

This method will have 1 parameter which will be the image file name. You will want to make sure your image is stored in the same directory as the python file, else you may have to specify the full path. To read the contents and turn it to grayscale, we can add the following lines of code:

```
    image = cv2.imread(image)
    image = cv2.cvtColor(src=image, code=cv2.COLOR_BGR2GRAY)
    return image
```

When reading images with OpenCV, the default mode is BGR and not RGB, so we will want to specify the code parameter as BGR2GRAY, allowing us to turn the BGR image into a grayscaled image. We will then return the new image.

Full processImage Method:

```python
def processImage(image):
  image = cv2.imread(image)
  image = cv2.cvtColor(src=image, code=cv2.COLOR_BGR2GRAY)
  return image
```

convolution.py hosted with ♡ by GitHub view raw

2D Convolution

To start the 2D Convolution method, we will have the following method header:

```
def convolve2D(image, kernel, padding=0, strides=1):
```

Such that the image and kernel are specified by the user and the default padding around the image is 0 and default stride is 1.

The next thing that we must do is apply cross correlation to our kernel and this can be done using NumPy very easily through just flipping the matrix horizontally then vertically. This looks like:

```
kernel = np.flipud(np.fliplr(kernel))
```

We then need to compute the matrix size of our outputted image. This can very simply be done through the formula:

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in}:	number of input features
n_{out}:	number of output features
k:	convolution kernel size
p:	convolution padding size
s:	convolution stride size

This must be implemented in each dimension (x, y). To start, we must gather the x and y size of the image and kernel. This can be done through:

```
xKernShape = kernel.shape[0]
yKernShape = kernel.shape[1]
xImgShape = image.shape[0]
yImgShape = image.shape[1]
```

We can then apply the size formula for each output dimension:

```
xOutput = int(((xImgShape — xKernShape + 2 * padding) / strides) + 1)

 yOutput = int(((yImgShape — yKernShape + 2 * padding) / strides) +
1)
```

Then we can create a fresh matrix with the deduced dimensions:

```
output = np.zeros((xOutput, yOutput))
```

This method specifically relies on padding being even on each side. First we want to check if the padding is 0 and if it is we do not want to apply unnecessary operations in order to avoid errors. So we start off with the following conditional statement:

```
if padding != 0:
```

We then create a fresh array of zeroes with the padded dimensions. This can be done through:

```
imagePadded = np.zeros((image.shape[0] + padding*2, image.shape[1] +
padding*2))
```

Note: We multiply the padding by 2 because we are applying even padding on all sides so a padding of 1 would increase the dimension of the padded image by 2.

We then replace the inner portion of the padded image with the actual image:

```
imagePadded[int(padding):int(-1 * padding), int(padding):int(-1 *
padding)] = image
```

If there is no padding we have an else statement to make the padded image equal to the image:

```
else:
    imagePadded = image
```

Now we get to the core of the convolution. We must iterate through the image and apply element wise multiplication and then sum it and set it equal to the respective element in the output array. To start, we can write our first loop:

```
for y in range(image.shape[1]):
```

This will be used it iterate through all of y dimension elements. We then have a break statement:

```
if y > image.shape[1] — yKernShape:
    break
```

This statement allows us to check if we are at the end of the image in the y direction. It will exit the complete convolution once we to reach the very bottom right of the image matrix.

We then have a conditional statement to take account of strides:

```
if y % strides == 0:
```

This will make sure that the step size is equivalent to the specified stride amount.

We then have a loop that iterates through each element in the x dimension:

```
for x in range(image.shape[0]):
```

The next thing we check for is if the kernel is at the very right of the image. If it is then it will break out of the x loop and then move down in the y direction and restart the convolution process.

```
if x > image.shape[0] — xKernShape:
    break
```

Finally, we have the main convolution operator that applies a convolution, sums the elements, and appends it to the output matrix:

```
try:
   if x % strides == 0:
      output[x, y] = (kernel * imagePadded[x: x + xKernShape, y: y + 
yKernShape]).sum()

except:
   break
```

Finally we return the output!

```
return output
```

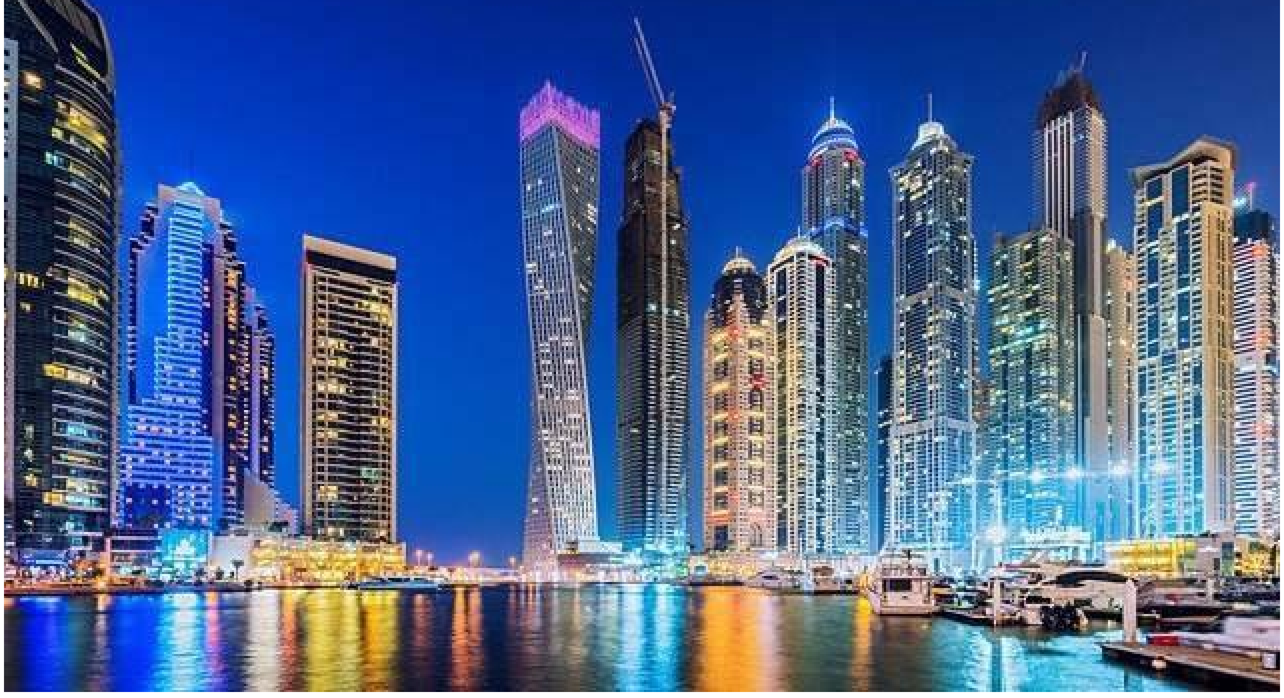
The complete convolution method looks like this:

```python
def convolve2D(image, kernel, padding=0, strides=1):
    # Cross Correlation
    kernel = np.flipud(np.fliplr(kernel))

    # Gather Shapes of Kernel + Image + Padding
    xKernShape = kernel.shape[0]
    yKernShape = kernel.shape[1]
    xImgShape = image.shape[0]
    yImgShape = image.shape[1]

    # Shape of Output Convolution
    xOutput = int(((xImgShape - xKernShape + 2 * padding) / strides) + 1)
    yOutput = int(((yImgShape - yKernShape + 2 * padding) / strides) + 1)
    output = np.zeros((xOutput, yOutput))

    # Apply Equal Padding to All Sides
    if padding != 0:
        imagePadded = np.zeros((image.shape[0] + padding*2, image.shape[1] + padding*2))
        imagePadded[int(padding):int(-1 * padding), int(padding):int(-1 * padding)] = image
        print(imagePadded)
    else:
        imagePadded = image

    # Iterate through image
    for y in range(image.shape[1]):
        # Exit Convolution
        if y > image.shape[1] - yKernShape:
            break
        # Only Convolve if y has gone down by the specified Strides
        if y % strides == 0:
            for x in range(image.shape[0]):
                # Go to next row once kernel is out of bounds
                if x > image.shape[0] - xKernShape:
                    break
                try:
                    # Only Convolve if x has moved by the specified Strides
                    if x % strides == 0:
                        output[x, y] = (kernel * imagePadded[x: x + xKernShape, y: y + yKernShap
                except:
                    break

    return output
```

convolution.py hosted with ♡ by GitHub view raw

Testing the 2D Convolution:

I decided to apply an edge detection kernel to my 2D Convolution. This was my original image:



The kernel I used was:

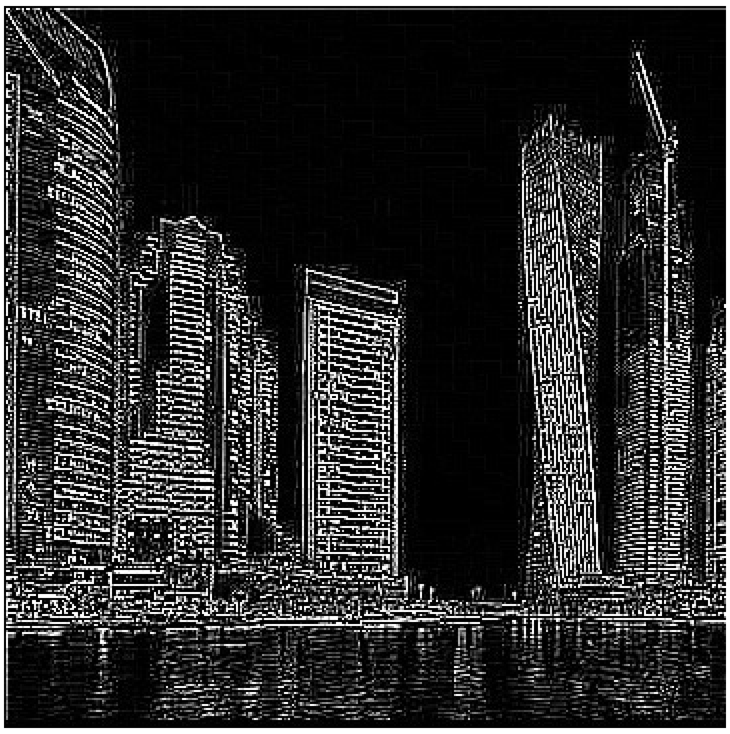
```
kernel = np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]])
```

My complete main method looked like:

```python
if __name__ == '__main__':
    # Grayscale Image
    image = processImage('Image.jpeg')

    # Edge Detection Kernel
    kernel = np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]])

    # Convolve and Save Output
    output = convolve2D(image, kernel, padding=2)
    cv2.imwrite('2DConvolved.jpg', output)
```

convolution.py hosted with ♡ by GitHub view raw

Upon applying the convolution, I received the following edges:



I think its safe to say it worked pretty well! My complete code can be found here on Github. If you enjoyed this or found this helpful please give it a clap and give me a follow! Thank you for reading!

Machine Learning Deep Learning Computer Science Numpy Convolutional Network