Let's explore how convolutions work by creating a basic convolution on a 2D Grey Scale image. Fir 'ascent' image from scipy. It's a nice, built-in picture with lots of angles and lines.

```
import cv2
import numpy as np
from scipy import misc
i = misc.ascent()

```

Next, we can use the pyplot library to draw the image so we know what it looks like.

```
import matplotlib.pyplot as plt
plt.grid(False)
plt.gray()
plt.axis('off')
plt.imshow(i)
plt.show()
```

The image is stored as a numpy array, so we can create the transformed image by just copying tha image so we can loop over it later.

```
i_transformed = np.copy(i)
size_x = i_transformed.shape[0]
size_y = i_transformed.shape[1]
```

Now we can create a filter as a 3x3 array.

```
#@title Default title text
# This filter detects edges nicely
# It creates a convolution that only passes through sharp edges and straight
# lines.

#Experiment with different values for fun effects.
#filter = [ [0, 1, 0], [1, -4, 1], [0, 1, 0]]

# A couple more filters to try for fun!
```

```
filter = [ [-1, -2, -1], [0, 0, 0], [1, 2, 1]]
#filter = [ [-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]

# If all the digits in the filter don't add up to 0 or 1, you
# should probably do a weight to get it to do so
# so, for example, if your weights are 1,1,1 1,2,1 1,1,1
# They add up to 10, so you would set a weight of .1 if you want to normalize
weight  = 1
```

--NORMAL--

Now let's create a convolution. We will iterate over the image, leaving a 1 pixel margin, and multiply pixel by the value defined in the filter.

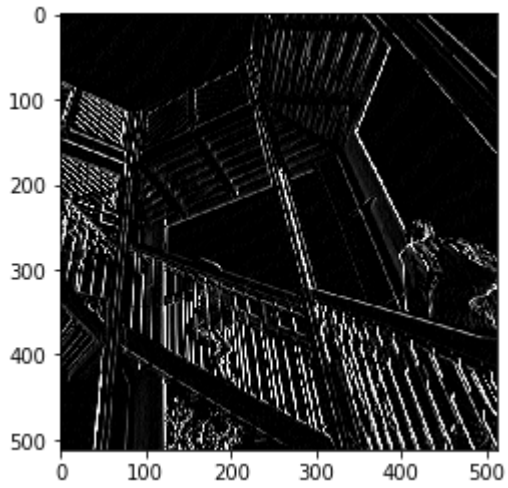
i.e. the current pixel's neighbor above it and to the left will be multiplied by the top left item in the fi the weight, and then ensure the result is in the range 0-255

Finally we'll load the new value into the transformed image.

```
for x in range(1,size_x-1):
  for y in range(1,size_y-1):
      convolution = 0.0
      convolution = convolution + (i[x - 1, y-1] * filter[0][0])
      convolution = convolution + (i[x, y-1] * filter[0][1])
      convolution = convolution + (i[x + 1, y-1] * filter[0][2])
      convolution = convolution + (i[x-1, y] * filter[1][0])
      convolution = convolution + (i[x, y] * filter[1][1])
      convolution = convolution + (i[x+1, y] * filter[1][2])
      convolution = convolution + (i[x-1, y+1] * filter[2][0])
      convolution = convolution + (i[x, y+1] * filter[2][1])
      convolution = convolution + (i[x+1, y+1] * filter[2][2])
      convolution = convolution * weight
      if(convolution<0):
        convolution=0
      if(convolution>255):
        convolution=255
      i_transformed[x, y] = convolution
```

Now we can plot the image to see the effect of the convolution!

```
# Plot the image. Note the size of the axes -- they are 512 by 512
plt.gray()
plt.grid(False)
plt.imshow(i_transformed)
#plt.axis('off')
plt.show()
```

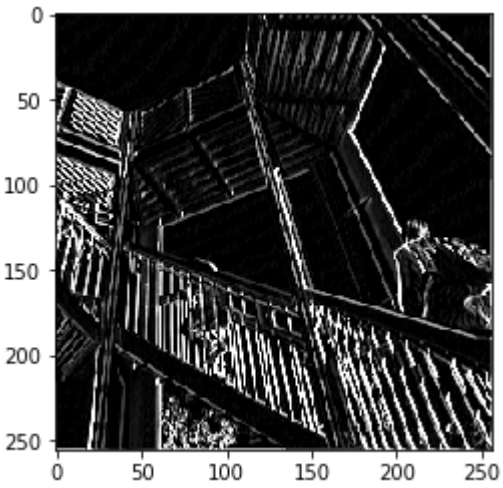



This code will show a (2, 2) pooling. The idea here is to iterate over the image, and look at the pixe beneath, and right-beneath. Take the largest of them and load it into the new image. Thus the new the dimensions on X and Y being halved by this process. You'll see that the features get maintaine

```
new_x = int(size_x/2)
new_y = int(size_y/2)
newImage = np.zeros((new_x, new_y))
for x in range(0, size_x, 2):
  for y in range(0, size_y, 2):
    pixels = []
    pixels.append(i_transformed[x, y])
    pixels.append(i_transformed[x+1, y])
    pixels.append(i_transformed[x, y+1])
    pixels.append(i_transformed[x+1, y+1])
    newImage[int(x/2),int(y/2)] = max(pixels)

# Plot the image. Note the size of the axes -- now 256 pixels instead of 512
plt.gray()
plt.grid(False)
plt.imshow(newImage)
#plt.axis('off')
plt.show()


```


0
50
100
150
200
250
0
50
100
150
200
250