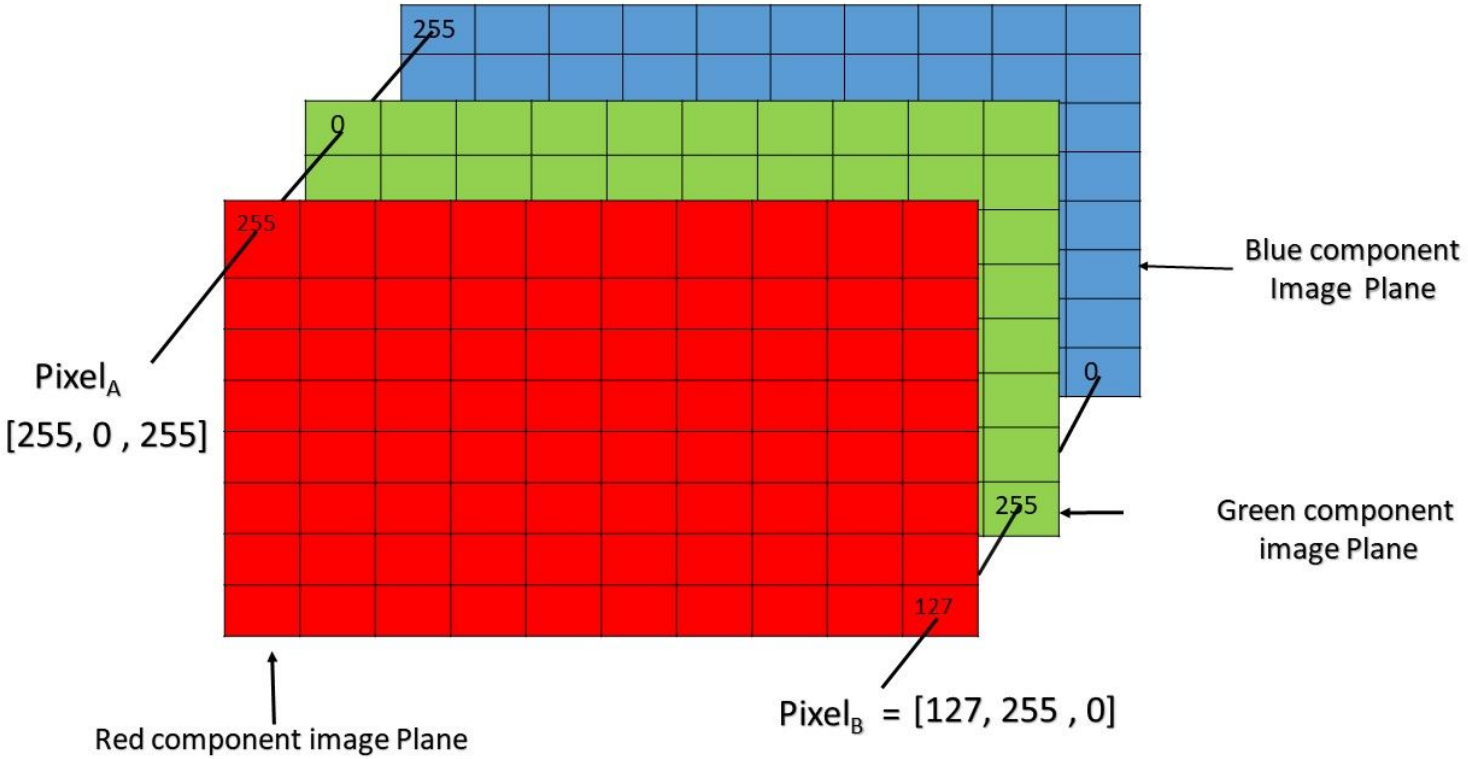
Images

Libraries

- matplotlib - has a module for working with images
- PIL - has number of image processing functions
- imageio - good for working with different formats
- cv2 - OpenCV library for computer vision

RGB



Pixel of an RGB image are formed from the corresponding pixel of the three component images

Opening image

```
import matplotlib.pyplot as plt



path = 'lenna.png'

img = plt.imread(path)
```

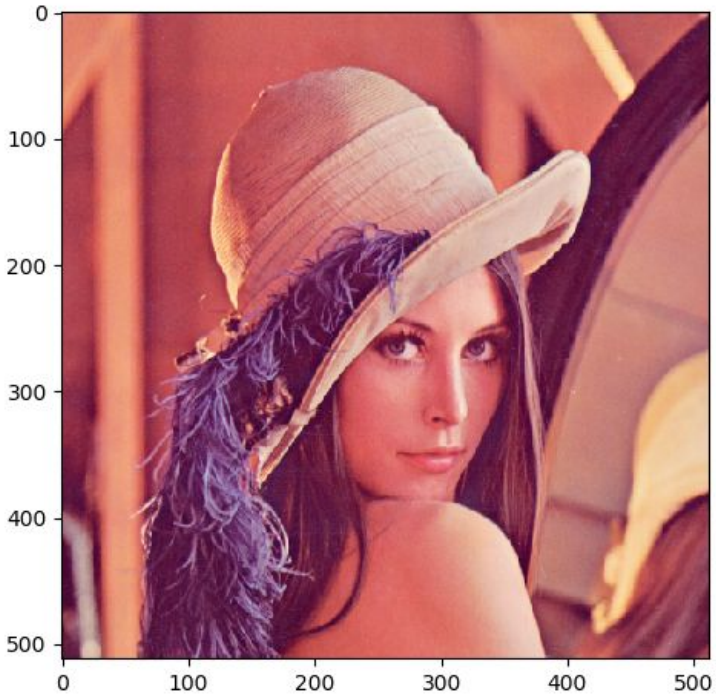
img value

```
print(img.shape)
(512, 512, 3)

print(img)
[[[0.8862745  0.5372549  0.49019608]    1 pixel
  [0.8862745  0.5372549  0.49019608]
  [0.8745098  0.5372549  0.52156866]
  ...
  [0.9019608  0.5803922  0.47843137]
  [0.8666667  0.50980395 0.43137255]
  [0.78431374 0.3882353  0.3529412 ]]   1 row
  ...
```

Visualize image

```
plt.imshow(img)
plt.show()
```



What's wrong with this approach

```
plt.imshow(img)
plt.axis('off')
plt.show()
```

Image cropping

As you can see color image is just a 3d np.array. You can manipulate it in any way, e.g. get part of the matrix

```
plt.imshow(img[200:300, 200:300])
plt.axis('off')
plt.show()
```

Separate colors

```
img[:, :, 1:] = 0
plt.imshow(img)
plt.axis('off')
plt.show()

img = plt.imread(path)
img[:, :, [0, 2]] = 0
plt.axis('off')
plt.show()

img = plt.imread(path)
img[:, :, :2] = 0
plt.imshow(img)
plt.axis('off')
plt.show()
```

```
img = plt.imread(path)
img[:, :, 0] = 0
plt.imshow(img)
plt.axis('off')
plt.show()

img = plt.imread(path)
img[:, :, 1] = 0
plt.imshow(img)
plt.axis('off')
plt.show()

img = plt.imread(path)
img[:, :, 2] = 0
plt.imshow(img)
plt.axis('off')
plt.show()
```

```
img = plt.imread(path)
img = 1 - img
plt.imshow(img)
plt.axis('off')
plt.show()
```

Grayscale conversion

```
img = plt.imread(path)

plt.imshow(img.mean(2))
plt.axis('off')
plt.show()
```

```
print(data.shape)
(512, 512)

print(data)
[[0.6379085  0.6379085 ... 0.60261434 0.5084967 ]
 [0.6379085  0.6379085 ... 0.60261434 0.5084967 ]
 ...
]
```

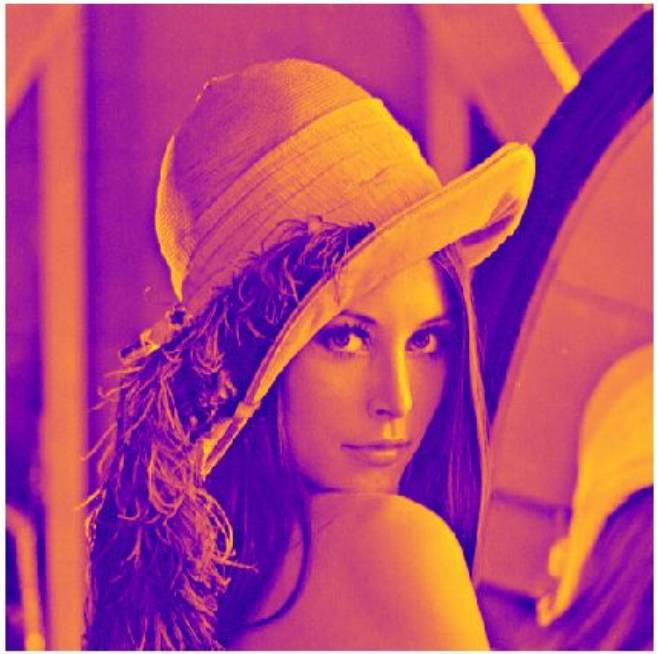
Normal grayscale conversion)

```
img = plt.imread(path)

plt.imshow(img.mean(2))
plt.axis('off', cmap='gray')
plt.show()
```

Even better versions

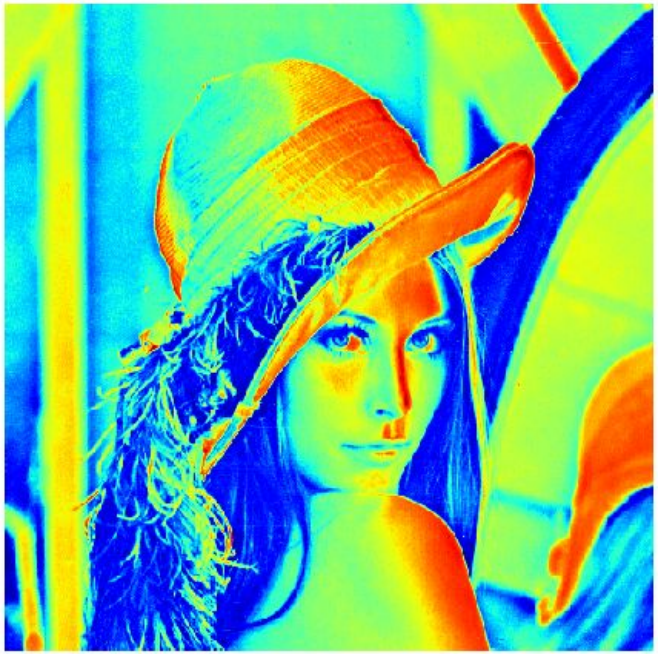
plasma



inferno

Even better versions

jet



terrain

Time for ~~pillage~~ PIL

```
from PIL import Image
import numpy as np


img = Image.open(path).convert('L')
data = np.array(img)

img.show()
# Almost the same as this
# plt.imshow(img)
# plt.axis('off')
# plt.show()
```

```
print(data.shape)
(512, 512)

print(data)
[[162 162 162 ... 169 154 128]
 [162 162 162 ... 169 154 128]
 [162 162 162 ... 169 154 128]
 ...
 [ 42  42  49 ... 104 100  98]
 [ 43  43  54 ... 103 105 107]
 [ 43  43  54 ... 103 105 107]]
```

Convolution

Cool mathematical operation with widespread use in convolutional networks

part of image		kernel		result of multiplication	summation
[[1 2 3] [0 2 4] [1 5 3]]	X	[[0 1 0] [0 1 0] [0 1 0]]	=	[[0 2 0] [0 2 0] [0 5 0]]	9

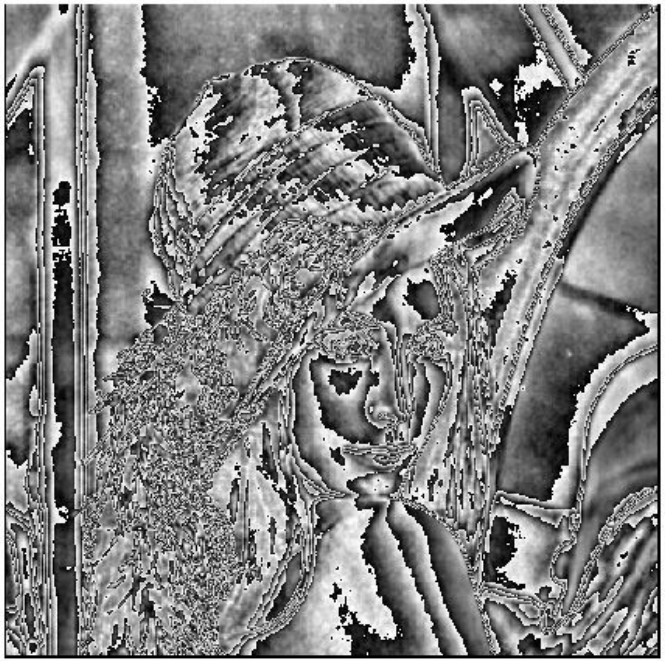
```
kernel = np.ones((3, 3))
print(kernel
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]

modified = np.zeros_like(data)
print(modified)
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

```
for row in range(1, data.shape[0] - 2):
    for col in range(1, data.shape[1] - 2):
        modified[row, col] = (data[row:row + 3, col:col + 3]
                              * kernel).sum()

plt.imshow(modified, cmap='gray')
plt.axis('off')
plt.show()
```

Looks quite strange...

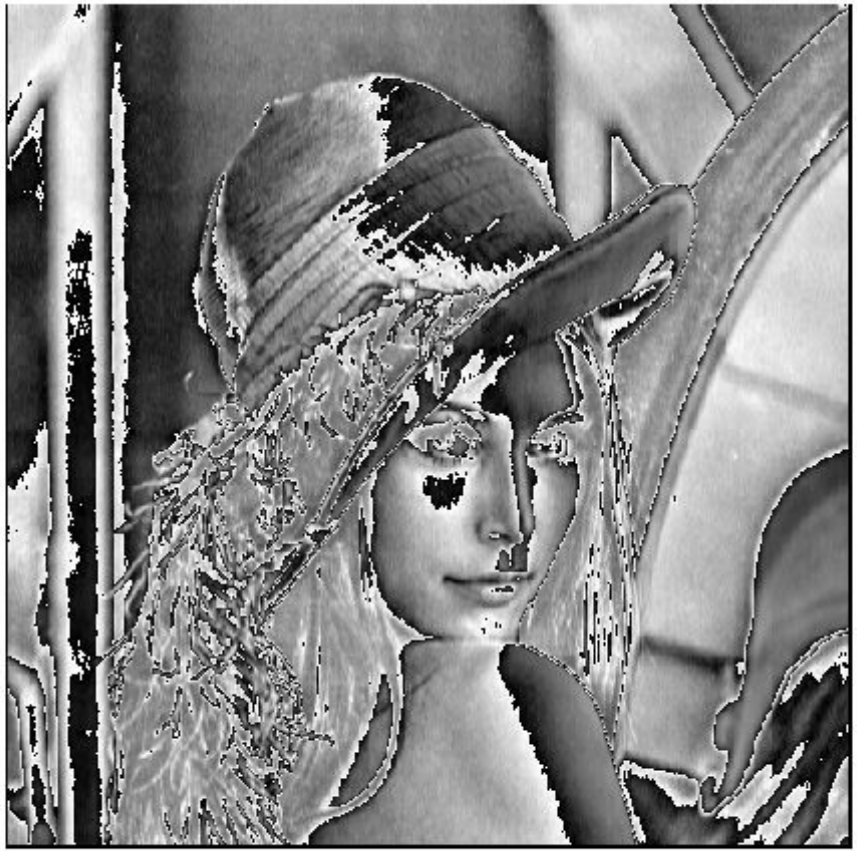


```
def convolve(img, kernel):
    """
    Make primitive convolution of img with kernel, assuming
     stride is equal to 1
    :param img: 2d np.array
    :param kernel: 2d 3 x 3 np.array
    :return:
    """
    # Assuming kernel is 3 x 3 and stride equals to 1
    modified = np.zeros_like(img)
    for row in range(1, img.shape[0] - 2):
        for col in range(1, img.shape[1] - 2):
            modified[row, col] = (img[row:row + 3, col:col + 3]
                                  * kernel).sum()

    return modified
```

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


modified = convolve(data, kernel)
plt.imshow(modified, cmap='gray')
plt.axis('off')
plt.show()
```



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

modified = convolve(data, kernel)
plt.imshow(modified, cmap='gray')
plt.axis('off')
plt.show()
```

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


modified = convolve(data, kernel)
print(modified.max(), modified.min())
plt.imshow(modified, cmap='gray')
plt.axis('off')
plt.show()
```

```
def convolve(img, kernel):
    """
    "Convolution" of img with kernel, assuming stride is equal
    to 1
    :param img: 2d np.array
    :param kernel: 2d 3 x 3 np.array
    :return:
    """
    # Assuming kernel is 3 x 3 and stride equals to 1
    modified = np.zeros_like(img)
    for row in range(1, img.shape[0] - 2):
        for col in range(1, img.shape[1] - 2):
            modified[row, col] = (img[row:row + 3, col:col + 3]
                                  * kernel).mean()

    return modified
```

```
kernel = np.ones((3, 3))

modified = convolve(data, kernel)
plt.imshow(modified, cmap='gray')
plt.axis('off')
plt.show()
```

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


modified = convolve(data, kernel)
plt.imshow(modified, cmap='gray')
plt.axis('off')
plt.show()
```

PIL way

```
from PIL import Image, ImageFilter


im = Image.fromarray(data)

im1 = im.filter(ImageFilter.BLUR)
im1.show()

im2 = im.filter(ImageFilter.CONTOUR)
im2.show()
```