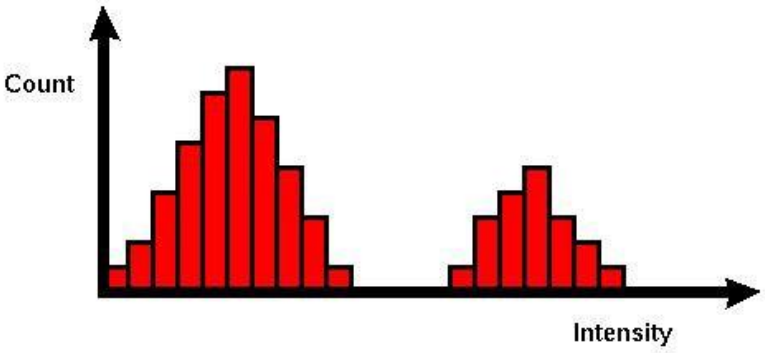
Image Histogram

Image processing is one of the rapidly growing technologies of our time and it has become an integral part of the engineering and computer science disciplines. Among its many subsets, techniques such as median filter, contrast stretching, histogram equalization, negative image transformation, and power-law transformation are considered to be the most prominent. In this tutorial, we will focus on the histogram equalization for digital image enhancement using Python.

What is a Histogram of An Image?

Histogram of an image is the graphical interpretation of the image's pixel intensity values. It can be interpreted as the data structure that stores the frequencies of all the pixel intensity levels in the image.

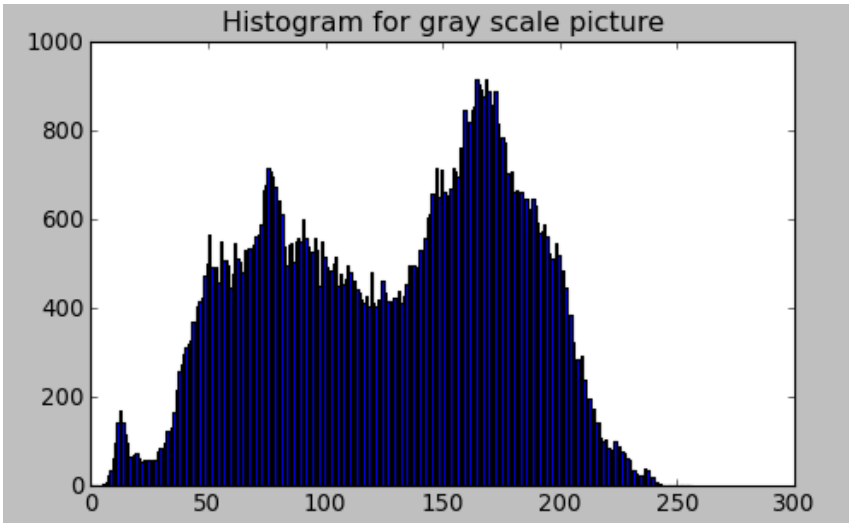


Histogram of an Image

As we can see in the image above, the X-axis represents the pixel intensity levels of the image. The intensity level usually ranges from 0 to 255. For a gray-scale image, there is only one histogram, whereas an RGB colored image will have three 2-D histograms — one for each color. The Y-axis of the histogram indicates the frequency or the number of pixels that have specific intensity values.

Here is a simple Python code for just loading the image:

```
import cv2
import numpy as np

inputImg = 'SunsetGoldenGate.jpg'
gray_img = cv2.imread(inputImg, cv2.IMREAD_GRAYSCALE)
cv2.imshow('GoldenGate',gray_img)

while True:
    k = cv2.waitKey(0) & 0xFF
    if k == 27: break             # ESC key to exit
cv2.destroyAllWindows()
```



We will need two Python libraries: NumPy for numerical calculation and OpenCV for image I/O. The easiest way to install these libraries is via Python package installer ***pip***. Enter the following commands on your terminal and you are set,

```
$ pip install numpy

$ pip install opencv-python
```

The code for histogram looks like this:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

inputImg = 'SunsetGoldenGate.jpg'
gray_img = cv2.imread(inputImg, cv2.IMREAD_GRAYSCALE)
cv2.imshow('GoldenGate',gray_img)
hist = cv2.calcHist([gray_img],[0],None,[256],[0,256])
```

```
plt.hist(gray_img.ravel(),256,[0,256])
plt.title('Histogram for gray scale picture')
plt.show()

while True:
    k = cv2.waitKey(0) & 0xFF
    if k == 27: break             # ESC key to exit
cv2.destroyAllWindows()
```

Note: This is how ravel() works, and it's equivalent of reshape(-1).

```
>>> x = np.array([[1, 2, 3], [4, 5, 6]])
>>> print np.ravel(x)
[1 2 3 4 5 6]
>>> x.reshape(-1)
array([1, 2, 3, 4, 5, 6])
```

Histogram Terminology

Before using that function, we need to understand some terminologies related with histograms.

1. **bins** :The histogram above shows the number of pixels for every pixel value, from 0 to 255. In fact, we used 256 values (bins) to show the above histogram. It could be 8, 16, 32 etc. OpenCV uses **histSize** to refer to **bins**.
2. **dims** : It is the number of parameters for which we collect the data. In our case, we collect data based on intensity value. So, in our case, it is **1**.
3. **range** : It is the range of intensity values we want to measure. Normally, it is [0,256], ie all intensity values.

calcHist()

OpenCV comes with an in-built **cv2.calcHist()** function for histogram. So, it's time to look into the specific parameters related to the **cv2.calcHist()** function.

```
cv2.calcHist(images, channels, mask, histSize, ranges[, hist[,
accumulate]])
```

In the code, we used:

```
hist = cv2.calcHist([gray_img],[0],None,[256],[0,256])
```

The parameters are:

1. **images**: source image of type uint8 or float32. it should be given in as a list, ie, **[gray_img]**.

2. **channels**: it is also given in as a list []. It the index of channel for which we calculate histogram. For example, if input is grayscale image, its value is **[0]**. For color image, you can pass [0],[1] or [2] to calculate histogram of blue,green or red channel, respectively.
3. **mask**: mask image. To find histogram of full image, it is set as **None**. However, if we want to get histogram of specific region of image, we should create a mask image for that and give it as mask.
4. **histSize**: this represents our BIN count. Need to be given in []. For full scale, we pass **[256]**.
5. **ranges**: Normally, it is **[0,256]**.

NumPy - np.histogram()

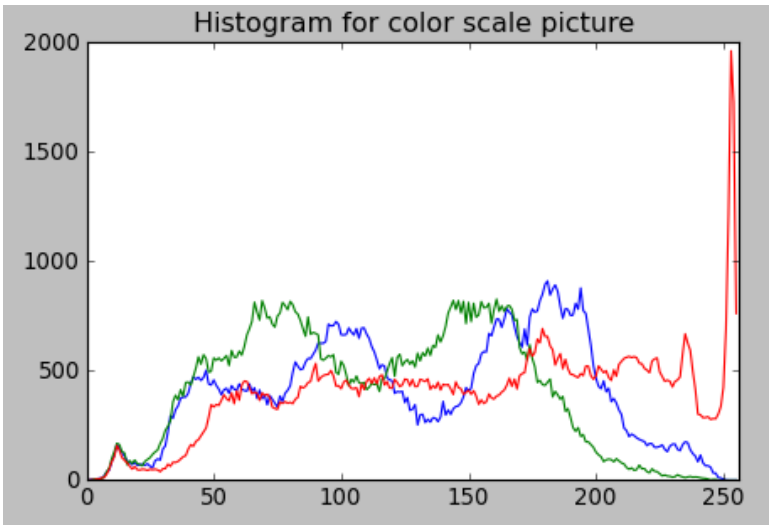
NumPy also provides us a function for histogram, np.histogram(). So, we can use NumPy fucntion instead of OpenCV function:

Other parts of the code remain untouched, and it gives us the same histogram.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

inputImg = 'SunsetGoldenGate.jpg'
gray_img = cv2.imread(inputImg, cv2.IMREAD_GRAYSCALE)
cv2.imshow('GoldenGate',gray_img)
#hist = cv2.calcHist([gray_img],[0],None,[256],[0,256])
hist,bins = np.histogram(gray_img,256,[0,256])

plt.hist(gray_img.ravel(),256,[0,256])
plt.title('Histogram for gray scale picture')
plt.show()

while True:
    k = cv2.waitKey(0) & 0xFF
    if k == 27: break             # ESC key to exit
cv2.destroyAllWindows()
```

Histogram for color image

Let's draw RGB histogram:



The code:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

inputImg = 'GoldenGateSunset.png'
img = cv2.imread(inputImg, -1)
cv2.imshow('GoldenGate',img)

color = ('b','g','r')
for channel,col in enumerate(color):
    histr = cv2.calcHist([img],[channel],None,[256],[0,256])
    plt.plot(histr,color = col)
    plt.xlim([0,256])
```

```
plt.title('Histogram for color scale picture')
plt.show()

while True:
    k = cv2.waitKey(0) & 0xFF
    if k == 27: break             # ESC key to exit
cv2.destroyAllWindows()
```

Histogram Equalization

Remember when you saw that low-quality image and felt a bit disappointed? It wasn't clear enough, and the details were a bit fuzzy. What if you could enhance that image to a better version? Wouldn't that be great? Fortunately, there's a way to do that, using Python!

One of the methods we can use to enhance an image is histogram equalization, which in particular enhances the contrast of the image. Almost all camera systems actually use histogram equalization to make our pictures look better, and at the end of the tutorial we will discover why this is so.

In the next section, we will delve deeper into what is meant by histogram equalization and what happens to the image when applying the method, and then we'll see how we can implement the method in Python.

We will use pout.jpg as a demo image in our tutorial here. The image looks as follows:

Let's take a look at how we can access the pixel values of the image, referred to as intensities. The small Python script below can be used to do just that (here we have used the OpenCV library to perform the image processing tasks):

```
import cv2

inputImg = 'pout.jpg'
img = cv2.imread(inputImg)
img_shape = img.shape
height = img_shape[0]
width = img_shape[1]

for row in range(width):
    for column in range(height):
        print (img[column][row])
```

What we are doing here is reading the input image (pout.jpg), and then investigating the shape (size) of the image. img_shape will return: (1031, 850, 3). This means that our image is of height (number of columns) 1031, and of width (number of rows) 850, and has 3 channels (RGB). Notice that the first parameter in the result is the height, and the second parameter is the width. Finally, we loop through the rows and columns and print out the different pixel values (intensities) at each row/column pair.

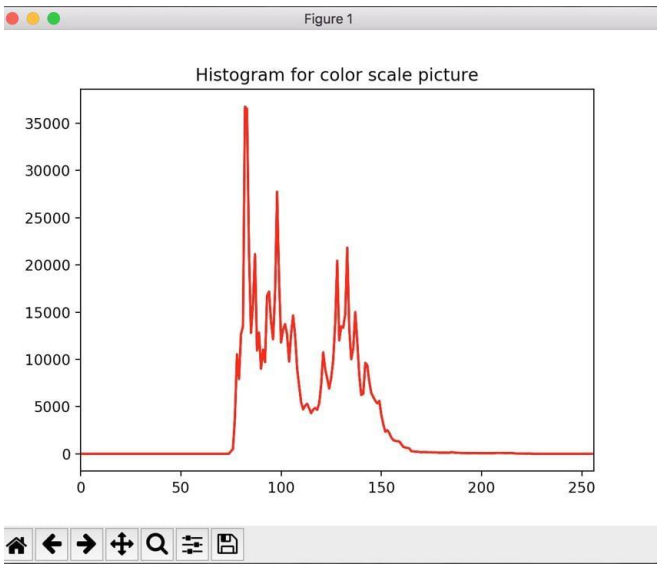
One sample of the output is: [137 137 137]. Yes, I know, you were expecting one value as a result for the pixel intensity. We actually have the value of the pixel intensity here, but what the output is showing us are the results of the red, green, and blue (RGB) channels. Please be aware, however, that in OpenCV the order is BGR, as this is how OpenCV loads the image. Thus, the above sample result contains the value 137 for each channel, in the order of B, G, and R, respectively.

The reason for the introduction is that histogram equalization is actually about the modification of pixel intensities for the sake of improving the image's contrast. Thus, our main work here will be at the pixel intensity level.

At this point, you might be wondering what a histogram is. Although sometimes the term might be a bit confusing, it is actually a very simple concept. The histogram is simply a diagram that depicts the number of pixels in an image at each intensity value found in that image.

Since our pixels have three values, one for each of the BGR channels, one way to draw the histogram is to have three histograms, one for each channel, where the x-axis will have the different pixel values (intensities), and the y-axis will show how many times (frequency) that particular pixel value appeared among the different pixel values. For instance, the red channel histogram can have a pixel value of 137 on the x-axis, and the y-axis can show how many pixels had this value for the red channel—say, for instance, 86. So the way we read that is by saying that the pixel value for the red channel of 137 showed up in 86 pixels, or has repeated 86 times in our image.

Using the code discussed in the Image Histogram section above to draw the histogram for our image, we get the following:



The histogram is actually for the red, green, and blue channels. Let's take a small sample of the output you would get from the previous code, as shown below. This shows that the channel values seem to always be the same, and the different three lines drawn will thus have the same values and will be drawn on top of each other, appearing as only one line.

```
[94 94 94]
[95 95 95]
[97 97 97]
[99 99 99]
[100 100 100]
[101 101 101]
[101 101 101]
[101 101 101]
[100 100 100]
[98 98 98]
[95 95 95]
[93 93 93]
```

What the histogram equalization method will do for the above histogram is that it will transform the intensity values in a way that will make the histogram look flatter in the resulting image. In other words, histogram equalization is a method that adjusts image intensities in order to enhance the contrast of the image.

The above histogram looks a bit concentrated towards the middle of the figure, and what histogram equalization will do is distribute the pixel intensity values further to get a more flattened histogram.

It's sufficient about histogram equalization to discuss here, as we don't want to get more mathematical in this tutorial, especially since it is more about the implementation of the method in Python. So now let's dive in to the implementation.

Histogram Equalization in Python

In this section, we shall implement the histogram equalization method in Python. We will use the above image (pout.jpg) in our experiments. Let's go through the process step by step. The first thing we need to do is import the OpenCV and NumPy libraries, as follows:

```
import cv2
import numpy
```

After that, we simply need to read our image, pout.jpg:

```
inputImg = 'pout.jpg'
img = cv2.imread(inputImg)
```

The good news is that OpenCV provides us with a function through which we can apply histogram equalization on an image, namely equalizeHist(). It is straightforward to apply this function on a grayscale image as the method actually equalizes the histogram of a grayscale image, but in our case we have three channels (RGB) for each pixel and we cannot apply histogram equalization on the three channels in a separate manner.

A nice solution was suggested in the book *Python: Real World Machine Learning*, which is to convert our image to the YUV color space, equalize the Y channel, and finally convert the result to RGB. So the first thing we do is convert our image to YUV. This can be done using the cvtColor() method, which converts the image from one space color to another, as follows:

```
img_to_yuv = cv2.cvtColor(img,cv2.COLOR_BGR2YUV)
```

Notice that we use BGR instead of RGB here, since OpenCV (as mentioned before) loads the images in BGR format. We now apply the histogram equalization method on the Y channel using the equalizeHist() method:

```
img_to_yuv[:,:,0] = cv2.equalizeHist(img_to_yuv[:,:,0])
```

Finally, we convert the Y channel to RGB (BGR in OpenCV), as follows:

```
hist_equalization_result = cv2.cvtColor(img_to_yuv, cv2.COLOR_YUV2BGR)
```

Congratulations! You have now applied histogram equalization to the image. In the next subsection, we will put all the code together and show how our image will look like after applying histogram equalization.

Putting It All Together

Let's put everything we have learned together. The Python script for applying histogram equalization on pout.jpg looks as follows:

```
import cv2
```

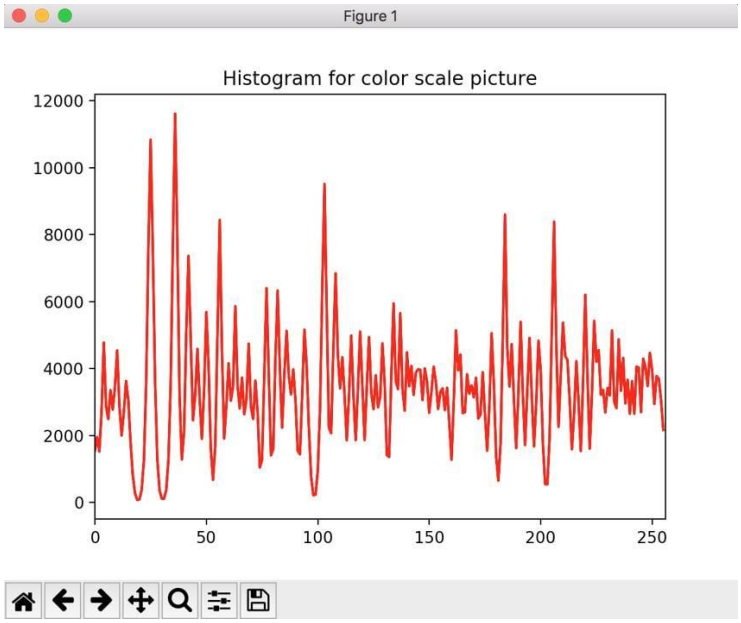
```
import numpy

inputImg = 'pout.jpg'
img = cv2.imread(inputImg)
img_to_yuv = cv2.cvtColor(img,cv2.COLOR_BGR2YUV)
img_to_yuv[:,:,0] = cv2.equalizeHist(img_to_yuv[:,:,0])
hist_equalization_result = cv2.cvtColor(img_to_yuv,
cv2.COLOR_YUV2BGR)

outputImg = 'result.jpg'
cv2.imwrite(outputImg, hist_equalization_result)
```

The output of the above script is the following image:

To notice the difference better, I will put the two images beside each other (left: original image; right: result of histogram equalization):

Did you notice the difference? The right image looks much clearer than the original image. No wonder why almost all imaging systems perform histogram equalization.

Before we wrap up, let's see what the histogram of our result looks like:



If you compare the histogram of the resulting image with the histogram of the original image, you will notice that the histogram of the resulting image is flatter than the histogram of the original image, and this is exactly what the histogram equalization method does.

Conclusion

In this tutorial, we saw how we can enhance the contrast of an image using a method called histogram equalization, and how it is easy to implement using Python and OpenCV.

The result was very interesting as it was much clearer than the original image, and the histogram of the result was flatter than the histogram of the original image, showing a better distribution of pixel intensity values across the image.