CODE > PYTHON

Image Enhancement in Python

by Abder-Rahman Ali 5 Jan 2018

Length: Medium Languages: English

Python

Let me start this tutorial by taking some theoretical jargon out of your way. When we talk about *image enhancement*, this basically means that we want a new version of the image that is more suitable than the original one.

For instance, when you scan a document, the output image might have a lower quality than the original input image. We thus need a way to improve the quality of output images so they can be visually more expressive for the viewer, and this is where image enhancement comes into play. When we enhance an image, what we are doing is sharpening the image features such as its contrast and edges.

It is important to note that image enhancement does not increase the information content of the image, but rather increases the dynamic range of the chosen features, eventually increasing the image's quality. So here we actually don't know what the output image would look like, but we should be able to tell (subjectively) whether there were any improvements or not, like observing more details in the output image, for instance.

Image enhancement is usually used as a preprocessing step in the fundamental steps involved in digital image processing (i.e. segmentation, representation). There are many techniques for image enhancement, but I will be covering two techniques in this tutorial: *image inverse* and *power law transformation*. We'll have a look at how we can implement them in Python. So, let's get started!

Image Inverse

As you might have guessed from the title of this section (which can also be referred to as *image negation*), image inverse aims to transform the dark intensities in the input image to bright intensities in the output image, and bright intensities in the input image to dark intensities in the output image. In other words, the dark areas become lighter, and the light areas become darker.

Say that `I(i,j)` refers to the intensity value of the pixel located at `(i,j)`. To clarify a bit here, the intensity values in the grayscale image fall in the range `[0,255]`, and `(i,j)` refers to the row and column values, respectively. When we apply the image inverse operator on a grayscale image, the output pixel `O(i,j)` value will be:

```
O(i,j) = 255 - I(i,j)
```

Nowadays, most of our images are color images. Those images contain three channels, *red*, *green*, and *blue*, referred to as `RGB` images. In this case, as opposed to the above formula, we need to subtract the intensity of *each* channel from 255. So the output image will have the following values at pixel `(i,j)`:

```
O_R(i,j) = 255 - R(i,j)
O_G(i,j) = 255 - G(i,j)
O-B)i,j) = 255 - B(i,j)
```

After this introduction, let's see how we can implement the image inverse operator in Python. I would like to mention that for the sake of simplicity, I will run the operator on a grayscale image. But I will give you some thoughts about applying the operator on a color image, and I will leave the full program for you as an exercise.

The first thing you need to do for a color image is extract each pixel channel (i.e. RGB) intensity value. For this purpose, you can use the Python Imaging Library (PIL). Go ahead and download a sample baboon image from baboon.png. The size of the image is `500x500`. Let's say you want to extract the red, green, and blue intensity values located at the pixel location `(325, 432)`. This can be done as follows:

```
from PIL import Image

im = Image.open('baboon.png')
print im.getpixel((325,432))
```

Based on the documentation, what the method `getpixel()` does is:

> *Returns the pixel value at a given position.*

After running the above script, you will notice that you only get the following result: `138`! But where are the three channels' (RGB) intensity values? The issue seems to be with the `mode` of the image being read. Check the mode by running the following statement:

```
print im.mode
```

You will get the output `P`, meaning that the image was read in a palette mode. One thing you can do is convert the image to RGB mode before returning the intensity values of the different channels. To do that, you can use the `convert()` method, as follows:

```
rgb_im = im.convert('RGB')
```

In this case, you would get the following value returned: `(180, 168, 178)`. This means that the intensity values for the red, green, and blue channels are 180, 168, and 178, respectively.

To put together everything we have described so far, the Python script which would return the RGB values of an image looks as follows:

```
from PIL import Image

im = Image.open('baboon.png')
rgb_im = im.convert('RGB')
print rgb_im.getpixel((325,432))
```

There is one point left before you move forward to the image inverse operator. The above example shows how to retrieve the RGB value of *one* pixel only, but when performing the inverse operator, you need to perform that on *all* the pixels.

To print out all the intensity values for the different channels of each pixel, you can do the following:

```
from PIL import Image

im = Image.open('baboon.png')
rgb_im = im.convert('RGB')
width, height = im.size

for w in range(width):
    for h in range(height):
        print rgb_im.getpixel((w,h))
```

At this point, I will leave it as an exercise for you to figure out how to apply the image inverse operator on all the color image channels (i.e. RGB) of each pixel.

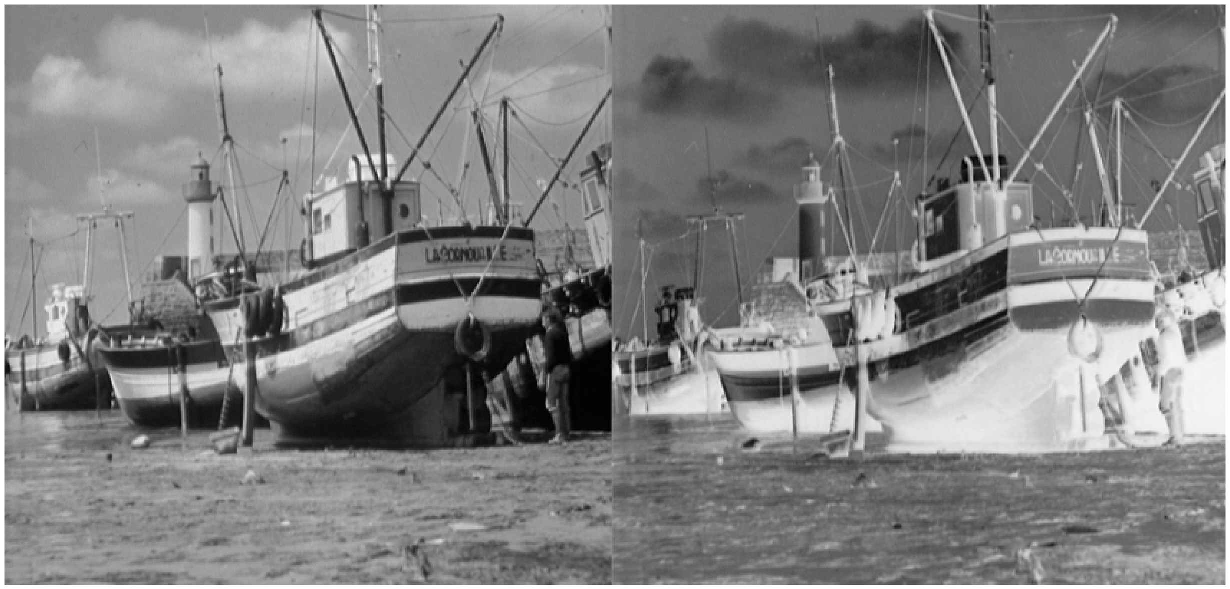
Let's have a look at an example that applies the image inverse operator on a grayscale image. Go ahead and download boat.tiff, which will serve as our test image in this section. This is what it looks like:

I'm going to use the `scipy` library for this task. The Python script for applying the image inverse operator on the above image should look as follows:

```python
import scipy.misc
from scipy import misc
from scipy.misc.pilutil import Image

im = Image.open('boat.tiff')
im_array = scipy.misc.fromimage(im)
im_inverse = 255 - im_array
im_result = scipy.misc.toimage(im_inverse)
misc.imsave('result.tiff',im_result)
```

The first thing we did after reading the image is to convert it to an ndarray in order to apply the image inverse operator on it. After applying the operator, we simply convert the ndarray back to an image and save that image as `result.tiff`. The figure below displays the result of applying image inverse to the above image (the original image is on the left, and the result of applying the image inverse operator is on the right):



Notice that some features of the image became clearer after applying the operator. Look, for instance, at the clouds and the lighthouse in the right image.

Power Law Transformation

This operator, also called *gamma correction*, is another operator we can use to enhance an image. Let's see the operator's equation. At the pixel `(i,j)`, the operator looks as follows:

```
p(i,j) = kI(i,j)^gamma
```

`I(i,j)` is the intensity value at the image location `(i,j)`; and `k` and `gamma` are positive constants. I will not go into mathematical details here, but I believe that you can find thorough explanations of this topic in image processing books. However, it is important to note that in most cases, `k=1`, so we will mainly be changing the value of gamma. The above equation can thus be reduced to:

```
p(i,j) = I(i,j)^gamma
```

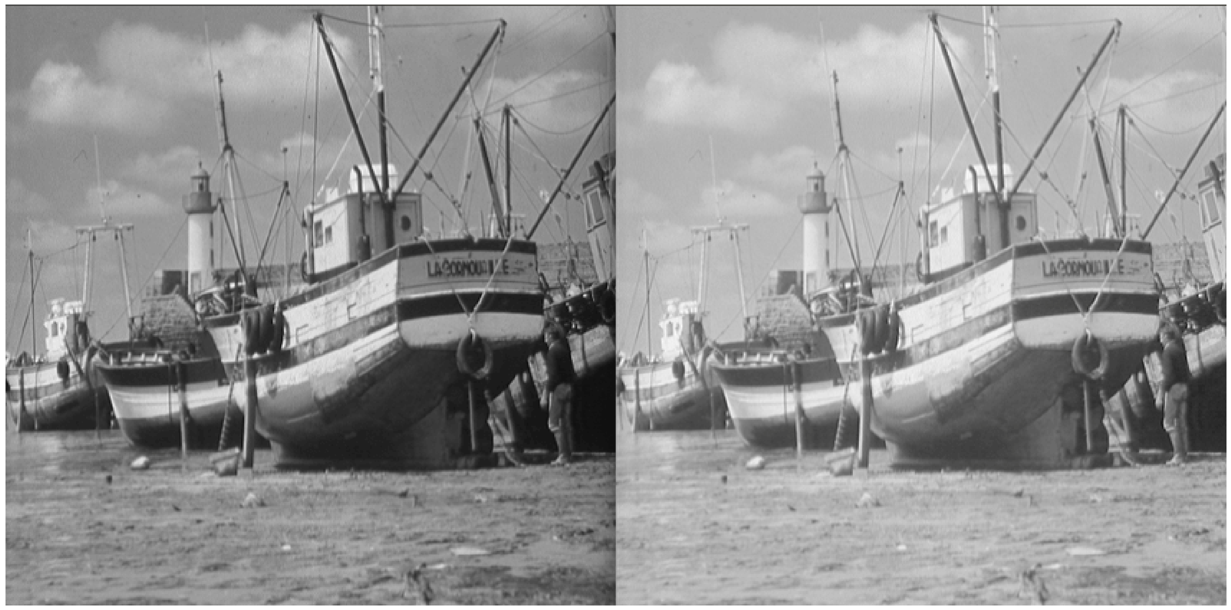
I'm going to use the `OpenCV` and `NumPy` libraries here. You can kindly check my tutorial Introducing NumPy should you need to learn more about the library. Our test image will again be boat.tiff (go ahead and download it).

The Python script to perform the Power Law Transformation operator looks as follows:

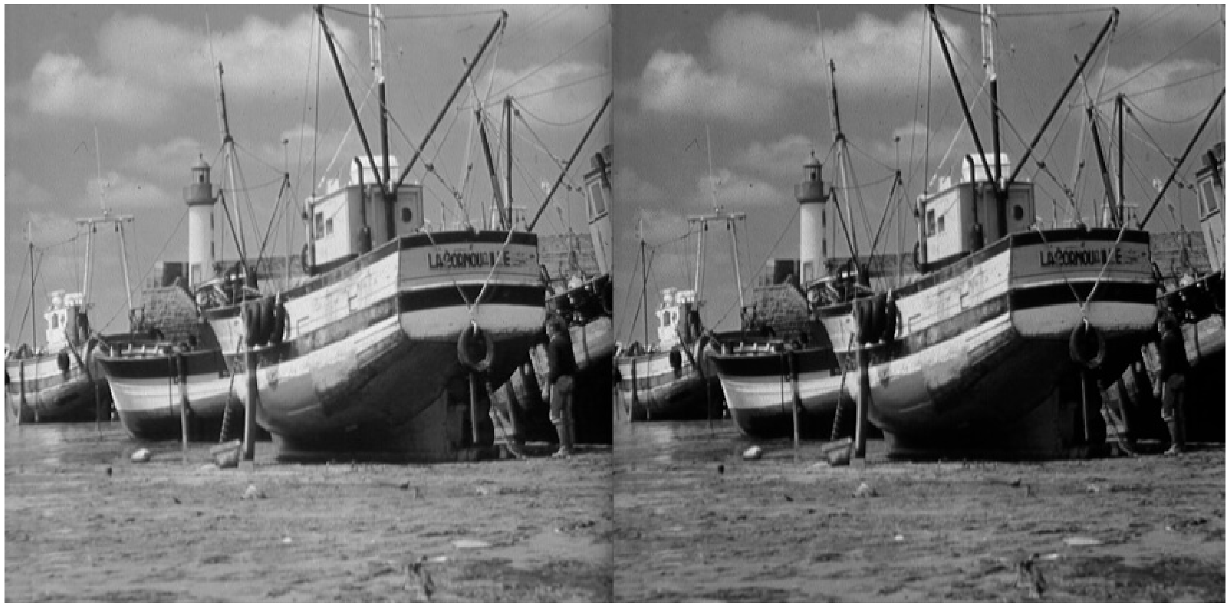
```python
import cv2
import numpy as np

im = cv2.imread('boat.tiff')
im = im/255.0
im_power_law_transformation = cv2.pow(im,0.6)
cv2.imshow('Original Image',im)
cv2.imshow('Power Law Transformation',im_power_law_transformation)
cv2.waitKey(0)
```

Notice that the gamma value we chose is `0.6`. The figure below shows the original image and the result of applying the Power Law Transformation operator on that image (the left image shows the original image, and the right image shows the result after applying the power law transformation operator).



The result above was when `gamma = 0.6`. Let's see what happens when we increase gamma to `1.5`, for instance:



Notice that as we increase the value of gamma, the image becomes darker, and vice versa.

One might be asking what the use of the power law transformation could be. In fact, the different devices used for image acquisition, printing, and display respond according to the power law transformation operator. This is due to the fact that the human brain uses gamma correction to process an image. For instance, gamma correction is considered important when we want an image to be displayed correctly (the best image contrast is displayed in all the images) on a computer monitor or television screens.

Advertisement

Conclusion

In this tutorial, you have learned how to enhance images using Python. You have seen how to highlight features using the image inverse operator, and how the power law transformation is considered a crucial operator for displaying images correctly on computer monitors and television screens.

Furthermore, don't hesitate to see what we have available for sale and for study in the Envato Market, and please ask any questions and provide your valuable feedback using the feed below.

Advertisement

Abder-Rahman Ali

Check out my FREE eBook How I Became Productive: 12 Proven Factors to Productivity.

abderhasan

FEED LIKE FOLLOW

Advertisement