Imagenes

April 7, 2014

1 Imagenes

1.1 Introducción

En estas notas se crearan usando ipython 2.0 con la opción notebook, junto con las librerias scipy y numpy para realizar operaciones sobre las imagenes, la libreria matplotlib para visualizar las imagenes. Es importante notar que para manipular imagenes en python una de las librerias más usada es PIL, sin embargo, esta no se actuliza frecuentemente y solo funciona con python 2.7, por lo que se usara PILLOW que es un fork de PIL y si funciona con python 3.

Acontiuación se da un ejemplo de como cargar una imagen en python.

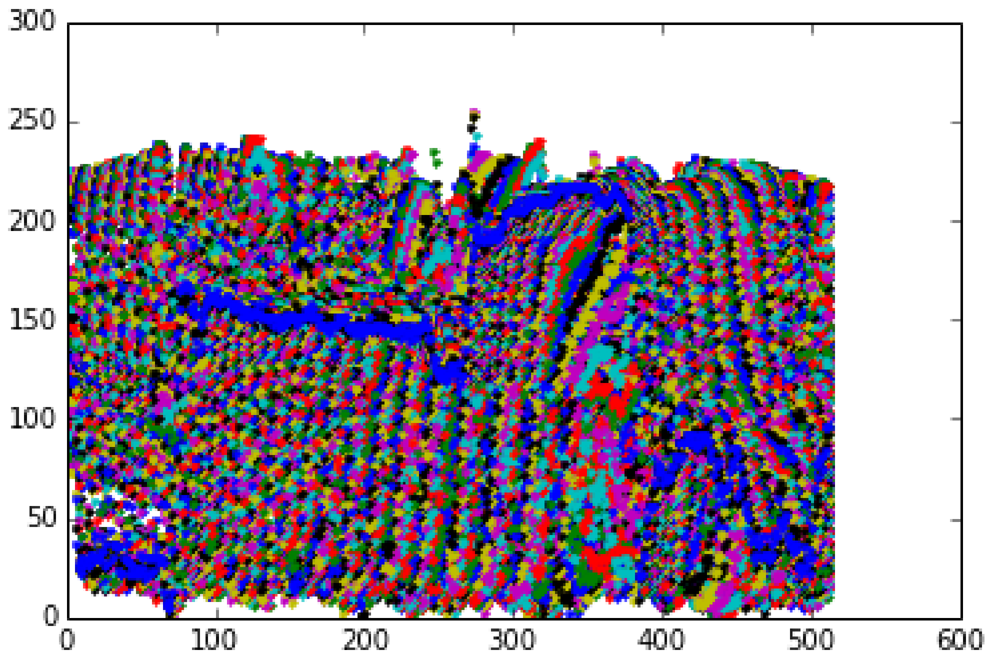
```
In [2]: from scipy import misc
        l = misc.lena()
        #misc.imsave('lena.png', l) # Con esta instrucción se graba la imagen a disco, es usada por PIL
        l.shape, l.dtype
        import matplotlib.pyplot as plt
        import matplotlib.image as mpimg
        import matplotlib.cm as cm
        import numpy as np

        #Esta isntrucción le dice a ipython que la imagenes las ponga dentro del mismo notebook
        # y no como una ventana independiente
        %matplotlib inline
        plt.figure(figsize=(14,10), dpi=300)
        plt.imshow(l,cmap=cm.Greys_r) #Es necesario avisar que se usara una imagen escala de grices en p
        plt.show()
```


```
In [7]: from PIL import Image
        im = Image.open('lena.png')
        im2=im.convert('RGB') #Otro metodo es abrir la imagen es usando PIL convertir la imagen a RGB
        plt.figure(figsize=(14,10), dpi=300)
        plt.imshow(im2)
        plt.show()
```


```
In [8]: im2.size
```

```
Out[8]: (512, 512)
```

```
In [14]: im3=np.asarray(im2)
         im3.shape
         plt.plot(im3[:,:,1],'.')
         plt.show()
```



1.2 Interpolación

Interpolaron is a basic tool used extensively in tasks such as zooming, shrinking, rotating, and geometric corrections. It is introduce interpolation and apply it to image resizing (shrinking and zooming), which are basically image resampling methods.

Fundamentally, interpolation is the process of using known data to estímate valúes at unknown locations. We begin the discussion of this topic with a simple example. Suppose that an image of size 500 x 500 pixels has to be enlarged 1.5 times to 750 x 750 pixels. A simple way to visualize zooming is to create an imaginary 750 X 750 grid with the same pixel spacing as the original, and then shrink it so that it fits exactly over the original image. Obviously, the pixel spacing in the shrunken 750 X 750 grid will be less than the pixel spacing in the original image. To perform intensity-level assignment for any point in the overlay, we look for its closest pixel ín the original image and assign the in tensity of that pixel to the new pixel in the 750 X 750 grid. When we are finished assigning intensities to all the points in the overlay grid, we expand it to the original specified size to obtain the zoomed image.

The method just discussed is called ***nearest neighbor interpolation*** because it assigns to each new location the intensity oí its nearest ncighbor in the original image. This approach is simple but, it has the tendenecy to produce undesirable artifacts, such as severe distortion oí straight edges. For this reason, it is used infrequently in practice. A more suiíable approach is bilinear interpolation, in which we use the íour nearest neighbors to estimate the intensity at a given location. Let (x, y) denote the coordínales oí the location to which we want to assign an intensity valué (think of it as a point of the grid described previously), and let v(x, y) denote that intensity valué. For bilinear interpolation, the assigned value is obtained using the equation

$$v(x, y) = ax + by + cxy + d \tag{1}$$

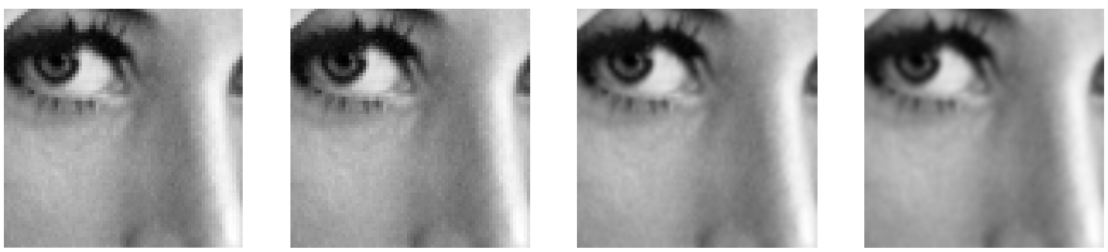
where the four coefficients are determined from the four equations in four unknowns that can be written using the four nearest neighbors oí point (x, y).

The next level of complexity is bicubic interpolation, which involves the six- teen nearest neighbors of a point.The intensity valué assigned to point (x, y) is obtained using the equation

$$v(x,y) = \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij} x^i y^j \tag{2}$$

where the sixteen coeífficients are determined from the sixteen equations in sixteen unknowns that can be written using the sixteen nearest neighbors of point (x , y).

In [4]:
```
plt.figure(figsize=(24,20), dpi=300)
plt.subplot(141)
plt.imshow(l[250:320, 250:320], cmap=plt.cm.gray, interpolation='none')
plt.axis('off')
plt.subplot(142)
plt.imshow(l[250:320, 250:320], cmap=plt.cm.gray, interpolation='nearest')
plt.axis('off')
plt.subplot(143)
plt.imshow(l[250:320, 250:320], cmap=plt.cm.gray, interpolation='bilinear')
plt.axis('off')
plt.subplot(144)
plt.imshow(l[250:320, 250:320], cmap=plt.cm.gray, interpolation='bicubic')
plt.axis('off')
plt.show()
```



It is possible to use more neighbors in interpolation, and there are more complex techniques, such as using splines and wavelets, that in some instances can yield better results than the methods just discussed.

1.3 Neighbors od a Pixel

A pixel p at coordinates (x, y) has four horizontal and vertical neighbors whose coordinates are given by

$$(x+1, y), (x-1, y), (x, y+1), (x, y-1)$$

1.3.1 Transformaciones

- Escalamiento

$$x = C_x V \tag{3}$$
$$y = C_y w \tag{4}$$

- Rotación

$$x = v\cos(\theta) + w\sin(\theta) \tag{5}$$
$$y = v\sin(\theta) + w\cos(\theta) \tag{6}$$

- Translación

$$
\begin{aligned}
x &= \quad v + t_x &\quad (7)\\
y &= \quad w + t_x &\quad (8)
\end{aligned}
$$