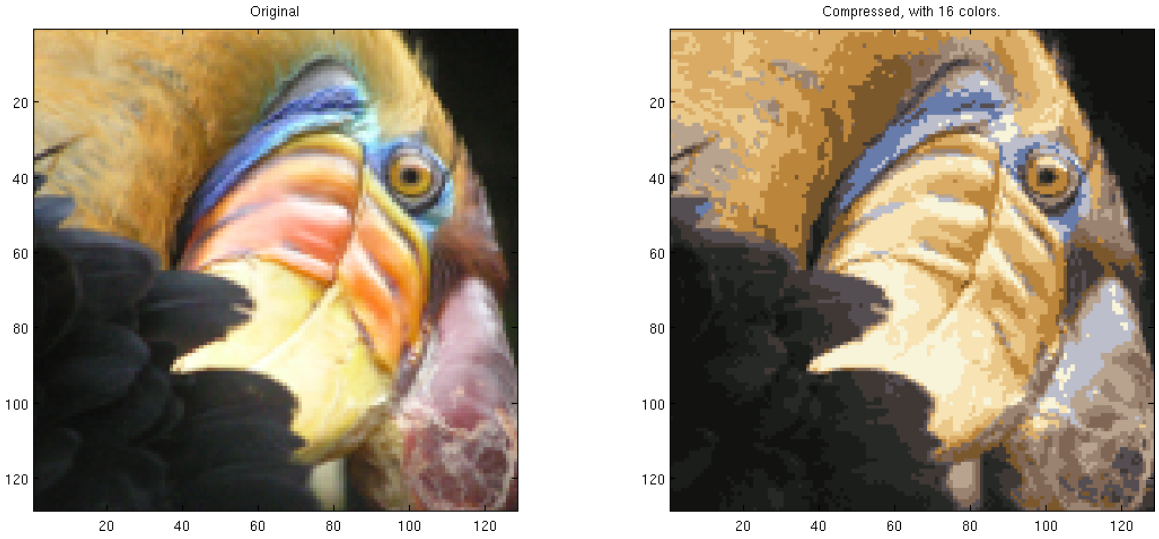
Implementation 3: Image Compression

In this part, you will use your previously written KMeans code for a useful real-world problem. In many RGB encodings each pixel is represented by 24 bits, 8 bits for each one of the main colors (Red, Green, Blue) ranging from 0 to 255, and therefore each pixel can have more than 16 million colors.
In this part, you will use K-means algorithm to reduce the number of colors to 16 (or 256) so that the color of each pixel can be represented by only 4 bits (or 8 bits). In fact, you only need to store the RGB values of the 16 (or 256) selected colors in an array, and for each pixel in the image you now need to only store the index of the color in the array. Since reducing the number of the colors result in lower quality for the image, we use K-means to find the 16 (or 256) colors that best group pixels in the image.

You can use the following python code to load, show an image inside a Jupyter notebook:

```
%matplotlib inline
import matplotlib.pyplot as plt
from matplotlib import image
img = image.imread('image.png')
plt.imshow(img)
plt.show()
```

Use the given code to load image.png. Note that by loading the image the variable img will be set to a 3D numpy array with 800 columns and 800 rows corresponding to the location of each pixel. The third dimension of this numpy array will hold the values of Red, Green, and Blue of the color.
In order to use the K-means Algorithm, that you've already implemented, on the image, first you need to reshape this 3D matrix into a 2D one in which every row represents a pixel.



The original image has 800x800 pixels and with 24 bits per pixel the image requires at least 800x800x24 bits to be stored. By reducing the number of colors to 16 and 256 the image only requires about 800x800x4 and 800x800x8 bits respectively to be stored.
By this point hopefully you have learnt the intuition behind the clustering and grasped a hands-on knowledge on how to use clustering for real world challenges.

In this section do the following steps:

A) Set K equal to 16 and run your KMeans algorithm on the image data.

B) Replace the RGB value of each pixel with the RGB value of the center of its cluster.

C) Set K equal to 256 and repeat the previous steps.

D) Show the result images alongside the original image, write compressed image to disk and compare its file size with the original image.