Photo by Rayia Soderberg on Unsplash

Image color identification with Machine Learning and Image Processing, using Python

Identify the colors of your image with few lines of code

1. The image

The first step is to get an image. I've used this one:

Photo by Daniela Cuevas on Unsplash

The great advantage of this algorithm is that, as you will see, it is pretty robust and it gives satisfying output quite independently from the input image, so you can choose your own photo.

2. The libraries

Let' s invoke some demons. You will need the **Sklearn** libraries for the Machine Learning part, **Numpy** for the vector transformations**, Pandas** for the final summary and some Image Processing typical libraries **(cv2, skimage, matplotlib.pyplot, …)**

```
plt.rcParams['axes.grid']=False
plt.rcParams['lines.linewidth'] = 2
```

Hosted on Jovian View File

3. The Machine Learning part

This great article **gives us a really good hint**. In fact, the main idea is that it is possible to use the image as a (N_rows X N_columns X N_channels) vector. **Considering this vector, it is possible to apply the K Means algorithm and identify k clusters, that will be our colors.**

This is super interesting for several reasons. The first one is that it does not require any specific training on a huge set of images. The second one is that you can increase the number of clusters (and, thus, the number of colors), choosing a smaller or higher amount of tones.

In order to do that, you will need these functions:

```
def get_image(image_path):
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    return image
def RGB2HEX(color):
    return "#{:02x}{:02x}{:02x}".format(int(color[0]), int(color[1]),
```

Hosted on Jovian View File

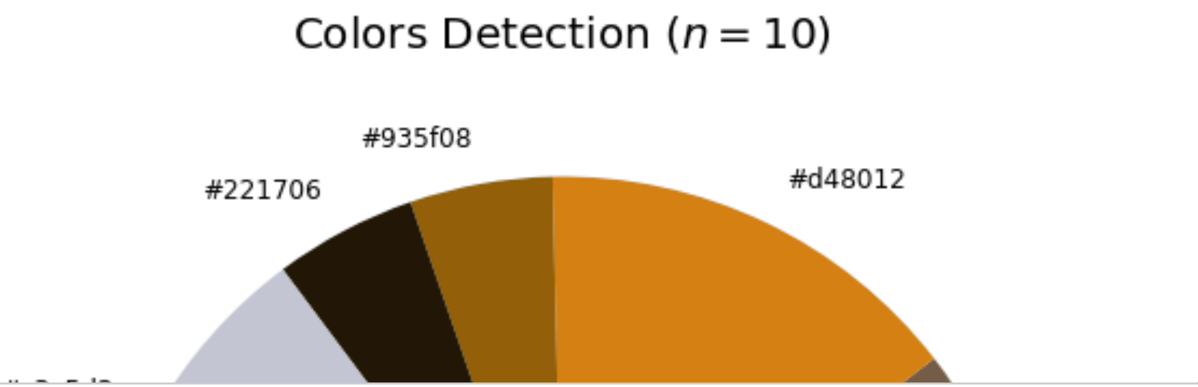
And these commands:

```
image = get_image('inputimage.jpg')
number_of_colors = 10
modified_image = image.reshape(image.shape[0]*image.shape[1], 3)
clf = KMeans(n_clusters = number_of_colors)
```

And here they are.

```
plt.title('Colors Detection ($n=10$)', fontsize=20)
plt.pie(counts.values(), labels = hex_colors, colors = hex_colors)
```

```
([<matplotlib.patches.Wedge at 0x7fb646f70700>,
  <matplotlib.patches.Wedge at 0x7fb646f70be0>,
  <matplotlib.patches.Wedge at 0x7fb646f7d0a0>,
  <matplotlib.patches.Wedge at 0x7fb646f7d520>,
  <matplotlib.patches.Wedge at 0x7fb646f7d9d0>,
  <matplotlib.patches.Wedge at 0x7fb646f7de50>,
  <matplotlib.patches.Wedge at 0x7fb646f89310>,
  <matplotlib.patches.Wedge at 0x7fb646f89790>,
  <matplotlib.patches.Wedge at 0x7fb646f89c10>,
  <matplotlib.patches.Wedge at 0x7fb64ca550d0>],
 [Text(1.0409393894711987, 0.355591320826772, '#735d49'),
  Text(0.4733826961743657, 0.992929414894477, '#d48012'),
  Text(-0.1939242723907806, 1.0827711561441349, '#935f08'),
  Text(-0.5141445017872571, 0.9724481638020268, '#221706'),
  Text(-0.9563095690753566, 0.5435733695600862, '#c3c5d2'),
  Text(-1.0757545930443362, -0.22967815643638176, '#d6b095'),
  Text(-0.7241104492344375, -0.8280483423747079, '#4c3613'),
  Text(0.004558222007513601, -1.0999905556922434, '#ebd6cd'),
  Text(0.7603768005262967, -0.7948755381953784, '#fbcea4'),
  Text(1.0730665090661293, -0.241926160471808, '#a98c75')])
```



Hosted on Jovian View File

In particular, it is useful to adopt the rgb encoding of colors (rgb_colors list)… but we'll get there later.

Let me tell you, this is not orthodox. In fact, you will have negative pixels. But it works, and if it works, don't touch it :)

The first step is technical, and **it is based on converting the RGB in integer values.**

```
for i in range(len(rgb_colors)):
    rgb_colors[i] = rgb_colors[i].astype(int)
```

Hosted on Jovian View File

Then, if we want to identify the colors of the image, **the idea is to break this image into smaller squares**. In this case, I've chosen the dimension of each square to be N_rows/10 X N_columns/10, thus obtaining 100 squares. These squares are obtained by using the following function.

```
def square_maker():
    inp_img = image
    h = int(inp_img.shape[0])
    step_h = int(h/10)
    w = int(inp_img.shape[1])
    step_w = int(w/10)
    X = np.arange(0,h+step_h,step_h)
    Y =np.arange(0,w+step_w,step_w)
    squares = [inp_img[0:step_h,0:step_w]]
    for i in range(0,len(X)-1):
        for j in range(0,len(Y)-1):
            squares.append(inp_img[X[i]:X[i+1],Y[j]:Y[j+1]])
```

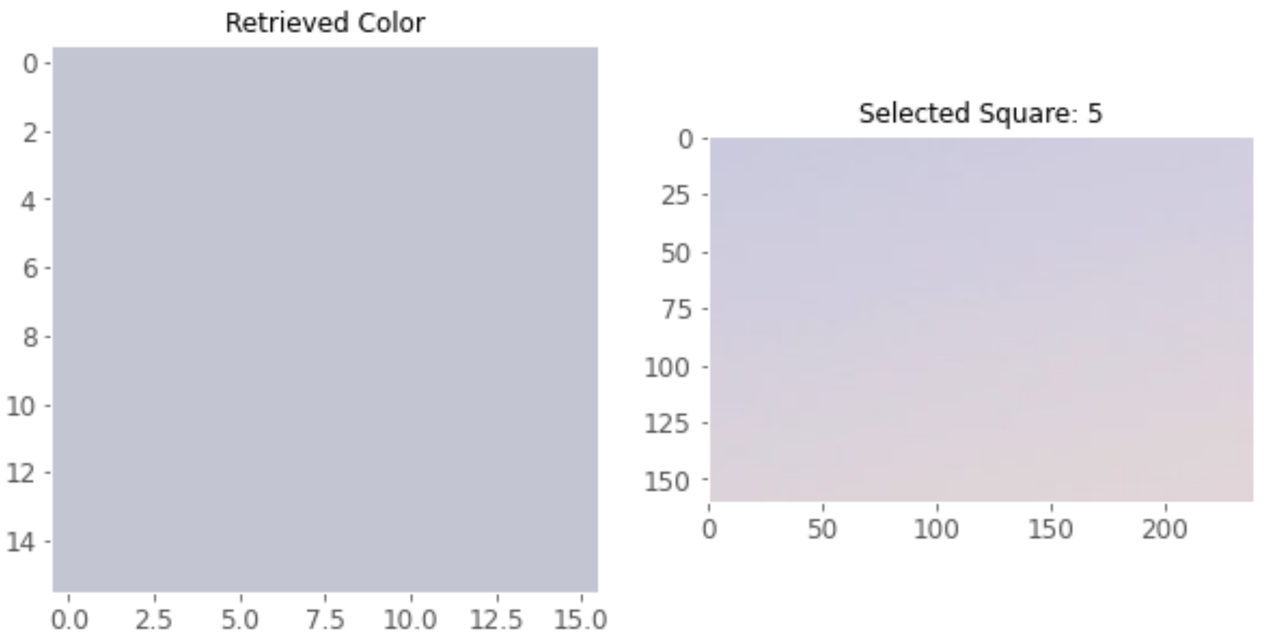
Hosted on Jovian View File

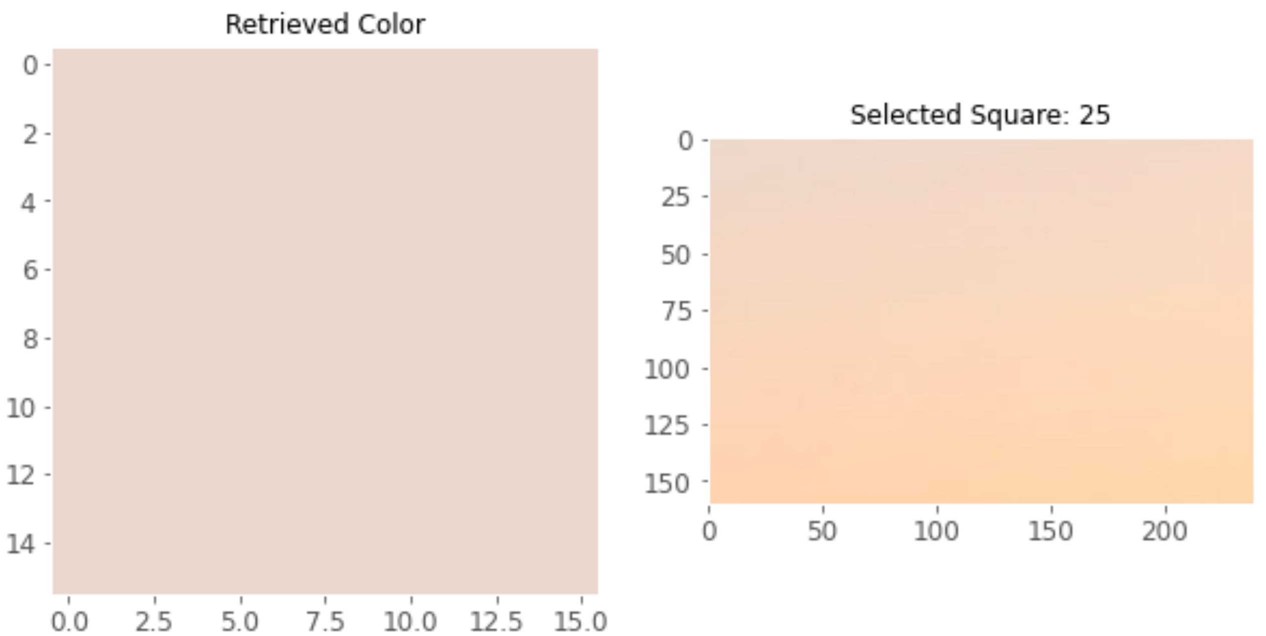
Of course, the squares are not monochromatic. This means that each square will have multiple colors. Nonetheless, the average distance between the image is the indicator that we need: **we pick the color**

```
def best_color_plot(selected_slice):
    plt.subplot(1,2,1)
    plt.title('Retrieved Color')
    plt.imshow((np.zeros((16,16,3))+ rgb_colors[color_computing(image)
    plt.subplot(1,2,2)
    plt.title('Selected Square: '+ str(selected_slice))
    plt.imshow(square_maker()[selected_slice])
```

Hosted on Jovian View File

Let me show you:





Retrieved Color

```
        sorted_results = sorted_results.append(d,ignore_index=True)
        k=k+1
    sorted_results['Square Number'] = sorted_results['Square Number'].
```

Hosted on Jovian View File

And this is it:

```
summary_df = build_summary()
```

```
summary_df.head()
```

	Square Number	#735d49	#d48012	#935f08	#221706	#c3c5d2	#d6b095	#4c3613	#eb
0	0	11.988112	12.349544	13.313494	21.308538	1.791643	4.784741	17.631673	4.37
1	1	12.068758	11.961283	13.457969	21.096211	1.926994	4.949398	17.497237	4.45
2	2	12.968671	11.809121	14.236116	21.881670	0.738481	4.572419	18.365533	3.19
3	3	13.357885	11.463896	14.596250	22.066389	0.713439	4.393230	18.630924	2.62

Hosted on Jovian View File

> *P.S. The number under each column are the percentage of the color for that specific square*

Conclusions

We are terrified by the idea that we live in a world with machines that "can see". And while this idea may be alarming, as it is comprehensible to be, at the same time I can't help but thinking that it is extremely fascinating. I like to think that we are somehow creator of new worlds, and a new nature… or maybe I'm just too tired.

If you liked the article and you want to know more about Machine