

Color Identification using Machine Learning

Submitted by

Shubham Jitendra Shinde

Abstract

The project was to develop a machine learning model to extract colors from the logos. I have used a machine learning algorithm, KMeans, to extract colors from a given image with OpenCV2 for image manipulation, applied KMeans to identify the major colors and then plot the information using Matplotlib. I have identified the majority 8 colors that exist in the image. I have learned about the new libraries and their functionality, how we can use the different libraries and connect them.

Table of Content

1. Problem Statement
2. Libraries used
3. Workflow of the program
4. Model Specifications
5. Test Cases

1. Problem Statement :

Take an image to extract the appropriate colour scheme from the logo.

Refer to the following specifications.

- Visit good looking and renowned websites like Facebook, LinkedIn, etc.
- Download logo image file.
- Extract 3 colours from the website. (Primary, secondary and background). Research about colour palettes for more information.
- Train a neural net to take the logo as input and output 3 colours in hex values.

2. Libraries Used :

a) OpenCV -

OpenCV-Python is a library of Python bindings designed to solve computer vision problems.

OpenCV-Python makes use of Numpy, which is a highly optimized library for numerical operations with a MATLAB-style syntax. All the OpenCV array structures are converted to and from Numpy arrays. This also makes it easier to integrate with other libraries that use Numpy such as SciPy and Matplotlib.

b) Matplotlib.pyplot -

Matplotlib.pyplot is a collection of functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

In matplotlib.pyplot various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area, and the plotting functions are directed to the current axes (please note that "axes" here and in most places in the documentation refers to the *axes* part of a figure and not the strict mathematical term for more than one axis).

c) Scikit-learn -

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistent interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.

Clustering of unlabeled data can be performed with the module `sklearn.cluster`.

Each clustering algorithm comes in two variants: a class that implements the `fit` method to learn the clusters on train data, and a function that, given train data, returns an array of integer labels corresponding to the different clusters. For the class, the labels over the training data can be found in the `labels` attribute.

d) Collection -

Collections in Python are containers that are used to store collections of data, for example, list, dict, set, tuple etc. These are built-in collections. Several modules have been developed that provide additional data structures to store collections of data. One such module is the Python collections module.

Counter is a subclass of dictionary object. The `Counter()` function in the collections module takes an iterable or a mapping as the argument and returns a Dictionary. In this dictionary, a key is an element in the iterable or the mapping and value is the number of times that element exists in the iterable or the mapping. You have to import the Counter class before you can create a counter instance.

e) Time -

The Python time module provides many ways of representing time in code, such as objects, numbers, and strings. It also provides functionality other than representing time, like waiting during code execution and measuring the efficiency of your code.

3.Workflow of the program :

To read any image, we use the method `cv2.imread()` and specify the complete path of the image which gets imported into the notebook as a Numpy array. We can then plot it using the pyplot's method `imshow()`.

The shape of the array is (x, y, 3). The first two values match the pixels of the image. Third value is set to 3 as each pixel is represented as a combination of three colors, Red, Blue and Green.

The color of the image looks a bit off. This is because, by default, OpenCV reads images in the sequence Blue Green Red (BGR). Thus, to view the actual image we need to convert the rendering to Red Green Blue (RGB).

The method `cvtColor` allows us to convert the image rendering to a different color space. To move from BGR color space to RGB, we use the method `cv2.COLOR_BGR2RGB`.

We can also resize the image to a given dimension. We use the method `resize` provided by `cv2`. The first argument is the image we want to resize, and the second argument is the width and height defined within parentheses.

Color Identification :

RGB to Hex Conversion

We'd first define a function that will convert RGB to hex so that we can use them as labels for our pie chart.

On reading the color which is in RGB space, we return a string. `{:02x}` simply displays the hex value for the respective color.

Read image in RGB color space

Next, we define a method that will help us get an image into Python in the RGB space.

We supply the path of the image as the argument . First, we read the file using `imread` and then change its color space before returning it.

Get colors from an image

We now define the complete code as a method that we can call to extract the top colors from the image and display them as a pie chart. I've named the method as `get_colors` and it takes 3 arguments:

1. `image`: The image whose colors we wish to extract.
2. `number_of_colors`: Total colors we want to extract.
3. `show_chart`: A boolean that decides whether we show the pie chart or not.

Method

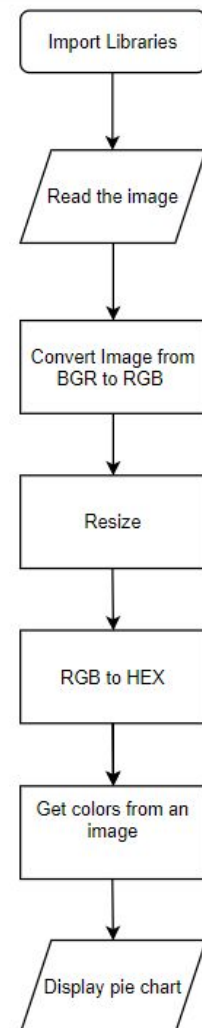
First, we resize the image to the size 600 x 400. It is not required to resize it to a smaller size but we do so to lessen the pixels which will reduce the time needed to extract the colors from the image. `KMeans` expects the input to be of two dimensions, so we use Numpy's `reshape` function to reshape the image data.

KMeans algorithm creates clusters based on the supplied count of clusters. In our case, it will form clusters of colors and these clusters will be our top colors. We then fit and predict on the same image to extract the prediction into the variable labels.

We use Counter to get count of all labels. To find the colors, we use `clf.cluster_centers_`. The `ordered_colors` iterates over the keys present in count, and then divides each value by 255. We could have directly divided each value by 255 but that would have disrupted the order.

Next, we get the hex and rgb colors. As we divided each color by 255 before, we now multiply it by 255 again while finding the colors. If `show_chart` is True, we plot a pie chart with each pie chart portion defined using `count.values()`, labels as `hex_colors` and colors as `ordered_colors`. We finally return the `rgb_colors` which we'll use at a later stage.

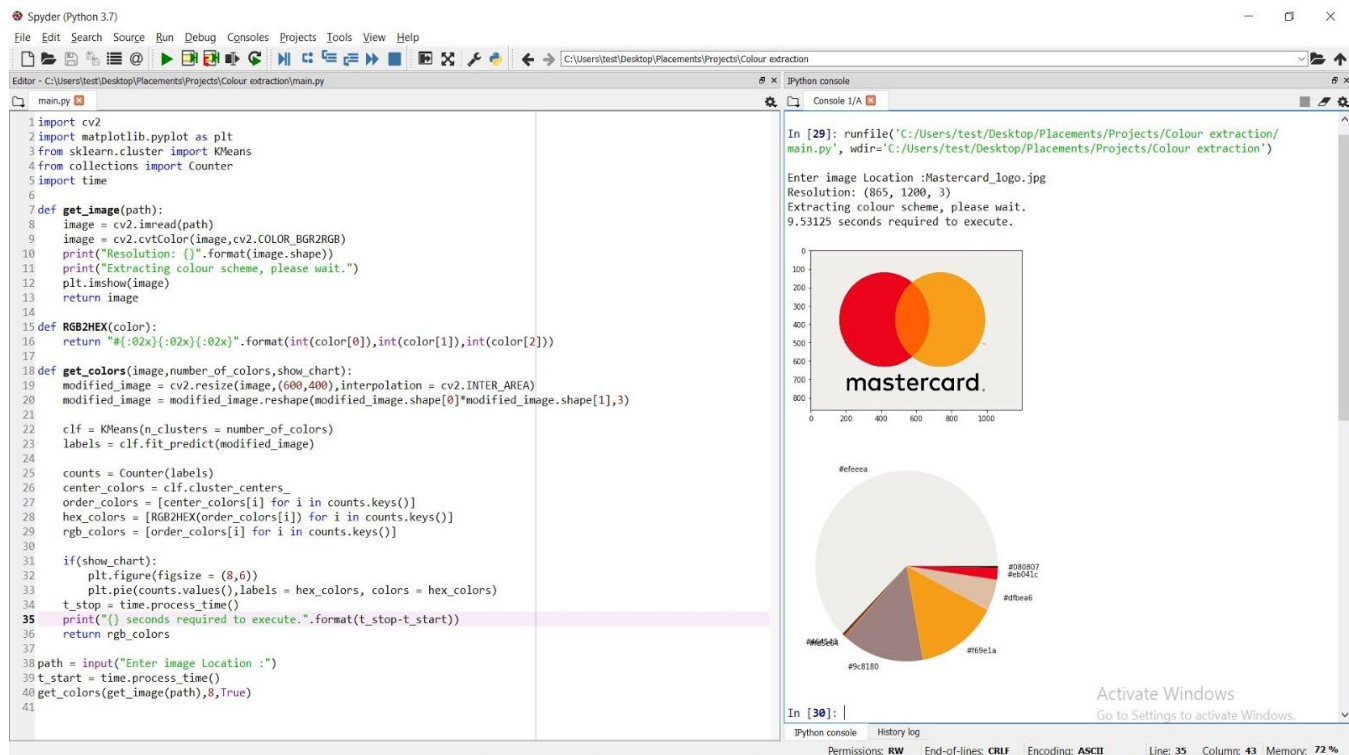
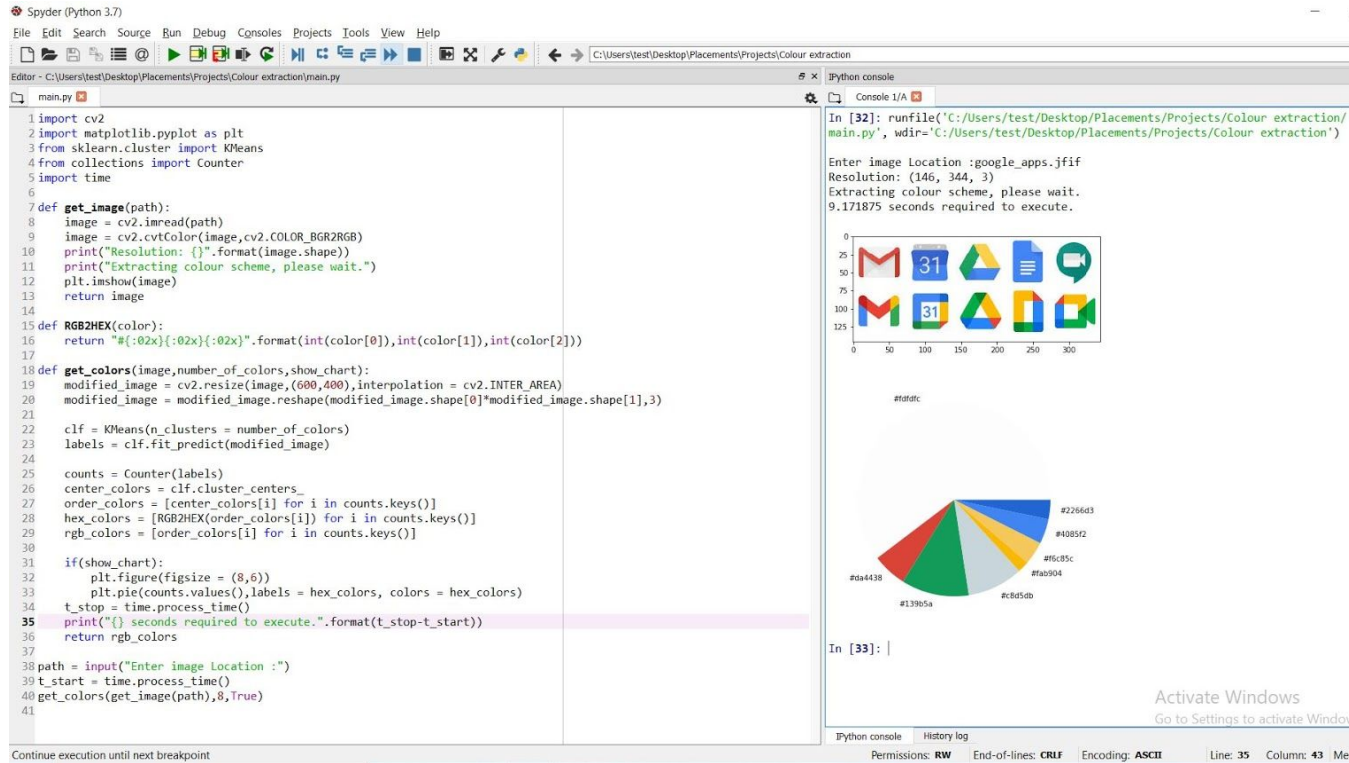
Call this method as `get_colors(get_image('sample_image.jpg'), 8, True)` and our pie chart appears with the top 8 colors of the image.



4. Model Specifications :

This model which goes by the name Color Identification by machine learning, extracts the different colours from any given image. The pie chart provided represents the percentages of different colours present in an image attached.

5. Test Cases :



Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\test\Desktop\Placements\Projects\Colour extraction

Editor - C:\Users\test\Desktop\Placements\Projects\Colour extraction\main.py

```

1 import cv2
2 import matplotlib.pyplot as plt
3 from sklearn.cluster import KMeans
4 from collections import Counter
5 import time
6
7 def get_image(path):
8     image = cv2.imread(path)
9     image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
10    print("Resolution: {}".format(image.shape))
11    print("Extracting colour scheme, please wait.")
12    plt.imshow(image)
13    return image
14
15 def RGB2HEX(color):
16    return "#{:02x}{:02x}{:02x}".format(int(color[0]), int(color[1]), int(color[2]))
17
18 def get_colors(image, number_of_colors, show_chart):
19    modified_image = cv2.resize(image, (600, 400), interpolation = cv2.INTER_AREA)
20    modified_image = modified_image.reshape(modified_image.shape[0]*modified_image.shape[1], 3)
21
22    clf = KMeans(n_clusters = number_of_colors)
23    labels = clf.fit_predict(modified_image)
24
25    counts = Counter(labels)
26    center_colors = clf.cluster_centers_
27    order_colors = [center_colors[i] for i in counts.keys()]
28    hex_colors = [RGB2HEX(center_colors[i]) for i in counts.keys()]
29    rgb_colors = [order_colors[i] for i in counts.keys()]
30
31    if (show_chart):
32        plt.figure(figsize = (8, 6))
33        plt.pie(counts.values(), labels = hex_colors, colors = hex_colors)
34        t_stop = time.process_time()
35    print("{} seconds required to execute.".format(t_stop-t_start))
36    return rgb_colors
37
38 path = input("Enter image Location :")
39 t_start = time.process_time()
40 get_colors(get_image(path), 8, True)
41

```

Python console

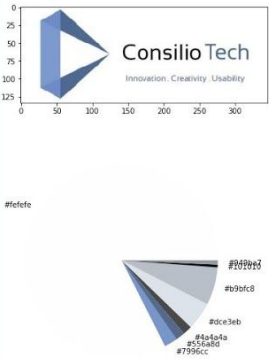
Console 1/A

```

In [27]: runfile('C:/Users/test/Desktop/Placements/Projects/Colour extraction/main.py', wdir='C:/Users/test/Desktop/Placements/Projects/Colour extraction')

Enter image Location :consilio_tech-logo.jpg
Resolution: (134, 346, 3)
Extracting colour scheme, please wait.
10.234375 seconds required to execute.

```



#f0f0e4
#b9b9c8
#d9d9e4
#a9a9d9
#7979cc
#7979cc
#a9a9d9
#f0f0e4

In [28]:

Activate Windows
Go to Settings to activate Windows.

Python console History log

Permissions: RW End-of-lines: CRLF Encoding: ASCII Line: 35 Column: 43 Memory: 73 %