How to find the average colour of an image in Python with OpenCV?

Asked 3 years, 7 months ago   Active 1 month ago   Viewed 63k times

Highly active question. You have enough reputation to answer or unprotect this question.

36

28

I tried this code:

```
import cv2
image = cv2.imread("sample.jpg")
pixel = image[200, 550]
print pixel
```

But I am getting error as:

> 'Nonetype' no attributes error **getitem**

This error is getting displayed after executing the third line of code.

python   opencv   numpy   image-processing   scikit-image   Edit tags

edited Mar 31 '17 at 10:36
Tonechas
10.2k 9 35 62

asked Mar 30 '17 at 7:17
Nikhith Tayambhath
369 1 4 3

see stackoverflow.com/a/47859322/1497139 – Wolfgang Fahl Nov 16 '19 at 8:52

7 Answers

Active | Oldest | Votes

93

How to fix the error

There are two potential causes for this error to happen:

1. The file name is misspelled.
2. The image file is not in the current working directory.

To fix this issue you should make sure the filename is correctly spelled (do case sensitive check just in case) and the image file is in the current working directory (there are two options here: you could either change the current working directory in your IDE or specify the full path of the file).

Average colour vs. dominant colour

Then to calculate the "average colour" you have to decide what you mean by that. In a grayscale image it is simply the mean of gray levels across the image. Colours are usually represented through 3-dimensional vectors whilst gray levels are scalars.

The average colour is the sum of all pixels divided by the number of pixels. However, this approach may yield a colour different to the most prominent visual color. What you might really want is **dominant color** rather than average colour.

Implementation

Let's go through the code slowly. We start by importing the necessary modules and reading the image:

```
import cv2
import numpy as np
from skimage import io

img = io.imread('https://i.stack.imgur.com/DNM65.png')[:, :, :-1]
```

Then we can calculate the mean of each chromatic channel following a method analog to the one proposed by @Ruan B.:

```
average = img.mean(axis=0).mean(axis=0)
```

Next we apply k-means clustering to create a palette with the most representative colours of the image (in this toy example `n_colors` was set to `5`).

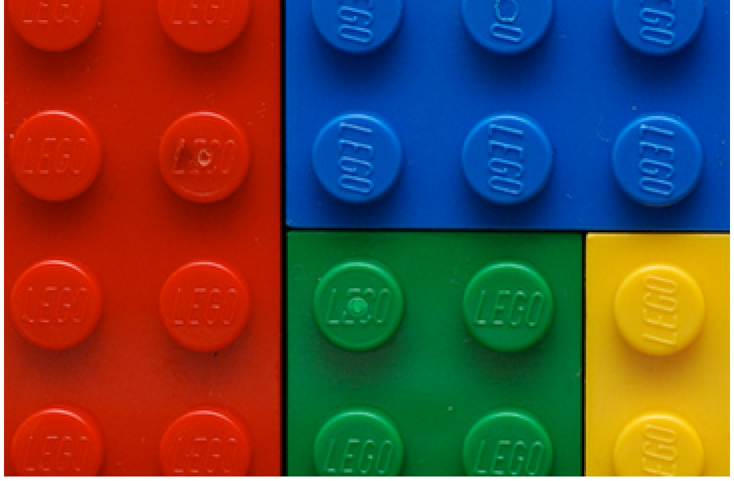
```
pixels = np.float32(img.reshape(-1, 3))

n_colors = 5
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 200, .1)
flags = cv2.KMEANS_RANDOM_CENTERS

_, labels, palette = cv2.kmeans(pixels, n_colors, None, criteria, 10, flags)
_, counts = np.unique(labels, return_counts=True)
```

And finally the dominant colour is the palette colour which occurs most frequently on the quantized image:

```
dominant = palette[np.argmax(counts)]
```

Comparison of results

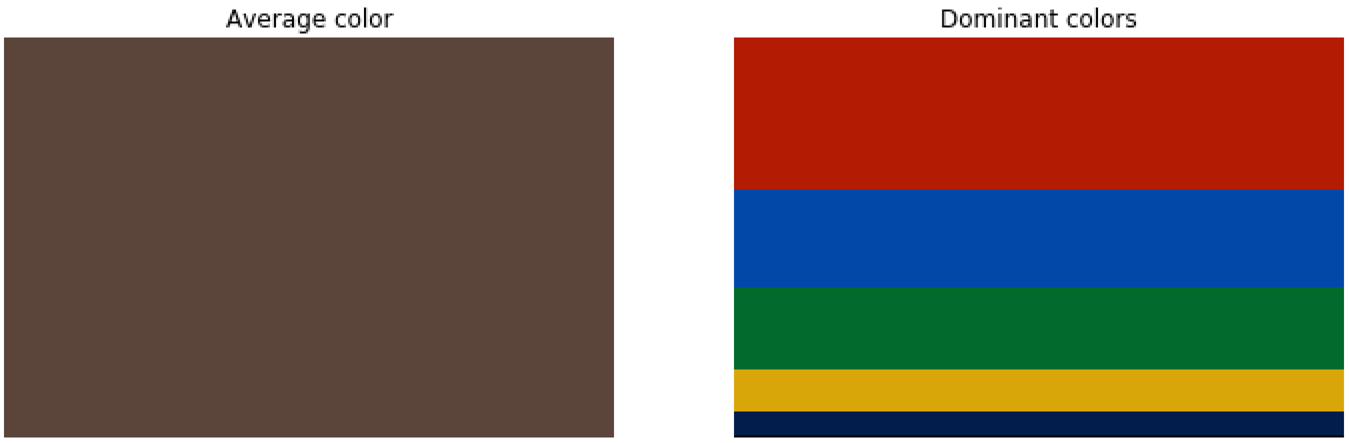
To illustrate the differences between both approaches I've used the following sample image:



The obtained values for the average colour, i.e. a colour whose components are the means of the three chromatic channels, and the dominant colour calculated throug k-means clustering are rather different:

```
In [30]: average
Out[30]: array([91.63179156, 69.30190754, 58.11971896])

In [31]: dominant
Out[31]: array([179.3999   ,  27.341282,   2.294441], dtype=float32)
```

Let's see how those colours look to better understand the differences between both approaches. On the left part of the figure below it is displayed the average colour. It clearly emerges that the calculated average colour does not properly describe the colour content of the original image. In fact, there's no a single pixel with that colour in the original image. The right part of the figure shows the five most representative colours sorted from top to bottom in descending order of importance (occurrence frequency). This palette makes it evident that the dominant color is the red, which is consistent with the fact that the largest region of uniform colour in the original image corresponds to the red Lego piece.



This is the code used to generate the figure above:

```
import matplotlib.pyplot as plt

avg_patch = np.ones(shape=img.shape, dtype=np.uint8)*np.uint8(average)

indices = np.argsort(counts)[::-1]
freqs = np.cumsum(np.hstack([[0], counts[indices]/float(counts.sum())]))
rows = np.int_(img.shape[0]*freqs)

dom_patch = np.zeros(shape=img.shape, dtype=np.uint8)
```

```
for i in range(len(rows) - 1):
    dom_patch[rows[i]:rows[i + 1], :, :] += np.uint8(palette[indices[i]])

fig, (ax0, ax1) = plt.subplots(1, 2, figsize=(12,6))
ax0.imshow(avg_patch)
ax0.set_title('Average color')
ax0.axis('off')
ax1.imshow(dom_patch)
ax1.set_title('Dominant colors')
ax1.axis('off')
plt.show(fig)
```

TL;DR answer

In summary, despite the calculation of the average colour - as proposed in @Ruan B.'s answer - is correct, the yielded result may not adequately represent the colour content of the image. A more sensible approach is that of determining the dominant colour through vector quantization (clustering).

edited Oct 4 at 10:18
Community ♦
1 1

answered Mar 30 '17 at 7:27
Tonechas
10.2k 9 35 62

2 Note: with cv2(2.4.8) , need change `cv2.kmeans(pixels, n_colors, None, criteria, 10, flags)` to `cv2.kmeans(pixels, n_colors, criteria, 10, flags)` . Doc: docs.opencv.org/2.4/modules/core/doc/clustering.html – Mithril Jul 10 '17 at 4:36

1 Why can't we just take the average R value, average G value and average B value (across all pixels) and check if the image is more Red, Green or Blue? – Vishal Sep 2 '17 at 17:37

@Tonechas In the calculation of the dominant_color, you said that you calculate it as the most frequent colour appearing in the quantized image, but you never use this variable. If we remove the quantized variable, the scripts works in the same way. Is that a mistake or I'm missing something? – nachmr Nov 12 '18 at 12:34

1 @nachmr: `dominant_color` was used in the interactive session and in the code that generates the last figure to show the differences between both approaches. I edited my answer to include that code and make this point clearer. – Tonechas Nov 12 '18 at 18:07

6 This answer is quite OK, except where it says "it makes no sense to average vectors." Of course it does. It is perfectly valid to compute the average of a set of vectors, and the result is meaningful as the average of the input vectors. For vectors in RGB space, the result is the average RGB value, the average color, and perfectly correct mathematically. "Separating the image into its chromatic components and taking the average of each component is a possible way to go." Yes, and mathematically equivalent to computing the average vector. – Cris Luengo Nov 12 '18 at 18:38

3

If you put the image into OpenCV's BGR format, you can run this code that puts each pixel into one of four classifications:

blue-green-red-gray

In the code that follows we process the image used by Tonechas,

colored lego pieces

PROGRAM

```
import cv2 as cv
import numpy as np
```

```
from imageio import imread

image = imread('https://i.stack.imgur.com/DNM65.png')
img  = cv.cvtColor(np.array(image), cv.COLOR_RGB2BGR)
rows, cols, _ = img.shape

color_B = 0
color_G = 0
color_R = 0
color_N = 0 # neutral/gray color

for i in range(rows):
    for j in range(cols):
        k = img[i,j]
        if k[0] > k[1] and k[0] > k[2]:
            color_B = color_B + 1
            continue
        if k[1] > k[0] and k[1] > k[2]:
            color_G = color_G + 1
            continue
        if k[2] > k[0] and k[2] > k[1]:
            color_R = color_R + 1
            continue
        color_N = color_N + 1

pix_total = rows * cols
print('Blue:', color_B/pix_total, 'Green:', color_G/pix_total, 'Red:',
color_R/pix_total, 'Gray:',  color_N/pix_total)
```

OUTPUT

```
Blue: 0.2978447577378059 Green: 0.21166979188369564 Red: 0.48950158575827024 Gray:
0.0009838646202282567
```

answered Nov 11 '19 at 19:13

CopyPasteIt
416 6 18

17

Another approach using K-Means Clustering to determine the dominant colors in an image with `sklearn.cluster.KMeans()`

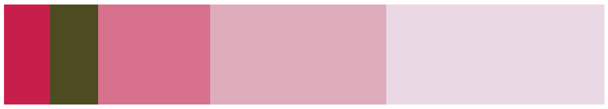
Input image

Results

With `n_clusters=5` , here are the most dominant colors and percentage distribution

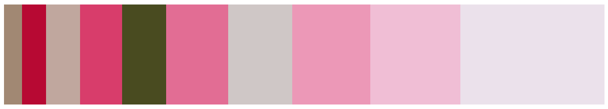
```
[76.35563647 75.38689122 34.00842057] 7.92%
[200.99049989  31.2085501   77.19445073] 7.94%
[215.62791291 113.68567694 141.34945328] 18.85%
[223.31013152 172.76629675 188.26878339] 29.26%
[234.03101989 217.20047979 229.2345317 ] 36.03%
```

Visualization of each color cluster



Similarity with `n_clusters=10` ,

```
[161.94723762 137.44656853 116.16306634] 3.13%
[183.0756441    9.40398442  50.99925105] 4.01%
[193.50888866 168.40201684 160.42104169] 5.78%
[216.75372674  60.50807092 107.10928817] 6.82%
[73.18055782 75.55977818 32.16962975] 7.36%
[226.25900564 108.79652434 147.49787087] 10.44%
[207.83209569 199.96071651 199.48047163] 10.61%
[236.01218943 151.70521203 182.89174295] 12.86%
[240.20499237 189.87659523 213.13580544] 14.99%
[235.54419627 225.01404087 235.29930545] 24.01%
```



```python
import cv2, numpy as np
from sklearn.cluster import KMeans

def visualize_colors(cluster, centroids):
    # Get the number of different clusters, create histogram, and normalize
    labels = np.arange(0, len(np.unique(cluster.labels_)) + 1)
    (hist, _) = np.histogram(cluster.labels_, bins = labels)
    hist = hist.astype("float")
    hist /= hist.sum()

    # Create frequency rect and iterate through each cluster's color and percentage
    rect = np.zeros((50, 300, 3), dtype=np.uint8)
    colors = sorted([(percent, color) for (percent, color) in zip(hist, centroids)])
    start = 0
    for (percent, color) in colors:
        print(color, "{:0.2f}%".format(percent * 100))
        end = start + (percent * 300)
        cv2.rectangle(rect, (int(start), 0), (int(end), 50), \
                      color.astype("uint8").tolist(), -1)
        start = end
    return rect

# Load image and convert to a list of pixels
image = cv2.imread('1.png')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
reshape = image.reshape((image.shape[0] * image.shape[1], 3))

# Find and display most dominant colors
cluster = KMeans(n_clusters=5).fit(reshape)
visualize = visualize_colors(cluster, cluster.cluster_centers_)
visualize = cv2.cvtColor(visualize, cv2.COLOR_RGB2BGR)
```

```
cv2.imshow('visualize', visualize)
cv2.waitKey()
```

edited Oct 5 '19 at 2:16

answered Oct 1 '19 at 2:47
nathancy
22.2k 10 52 67

very neat ! Thank you – PolarBear10 May 6 at 12:31

-1

This post is hidden. It was deleted last year by Bhargav Rao ♦.

Here's a nice article explaining the differences between averaging, using color frequency and k-means clustering with sample images and results.

answered May 6 '19 at 1:54
Pratik Khadloya
10.9k 9 67 90

A link to a solution is welcome, but please ensure your answer is useful without it: add context around the link so your fellow users will have some idea what it is and why it's there, then quote the most relevant part of the page you're linking to in case the target page is unavailable. Answers that are little more than a link may be deleted. – Machavity ♦ May 6 '19 at 2:40

comments disabled on deleted / locked posts / reviews

-1

This answer is hidden. This answer was deleted via review 2 years ago by Jonathan Eustace, vitr, SilverNak, Primusa.

```
average_color=cv2.mean(my_image)
```

answered Apr 22 '18 at 2:55
user9680476
1

2 We like to see context and explanation around answers. Would you edit your post to explain why this would help? Have a look at Tonechas's answer to see the kinds of answer we prefer here - they don't need to be *that* substantial, but one-liners tend not to be received well. – halfer Apr 22 '18 at 11:07

comments disabled on deleted / locked posts / reviews

-2

This answer is hidden. This answer was deleted via review 2 years ago by Joe McMahon, drhagen, niko, wp78de.

if you are clustering using k means you may want to convert your color space in to LAB. RGB colors are great for computers but the LAB color space more accurately represents how people see colors. so two colors that are close in value in the LAB space are more similar to the human eye.

answered Nov 30 '17 at 22:27
user9036316
1

This does not provide an answer to the question. Once you have sufficient reputation you will be able to comment on any post; instead, provide answers that don't require clarification from the asker. - From Review – niko Dec 1 '17 at 0:47

comments disabled on deleted / locked posts / reviews

36

I was able to get the average color by using the following:

```
import cv2
import numpy
myimg = cv2.imread('image.jpg')
avg_color_per_row = numpy.average(myimg, axis=0)
avg_color = numpy.average(avg_color_per_row, axis=0)
print(avg_color)
```

Result:

```
[ 197.53434769  217.88439451  209.63799938]
```

Great Resource which I referenced

answered Mar 30 '17 at 8:19
Ruan B.
361 2 3

1 Thank you for this short and very efficient code. It returns [blue, green, red] instead of [red, green, blue] order – Mohsen Haddadi Dec 1 '17 at 7:08

2 @MohsenHaddadi its apparent , because cv2 reads image in BGR format. – Nasir Shah Nov 16 '18 at 13:14

I am getting this error, do you know why? ImportError: numpy.core.multiarray failed to import Traceback (most recent call last): File "/Users/luca/Desktop/example.py", line 1, in <module> import cv2 ImportError: numpy.core.multiarray failed to import – Luca Perico Nov 16 '18 at 14:39

@LucaPerico reinistall opencv python wheel to fix this – iratzhash Nov 8 '19 at 14:35