

Python - Find dominant/most common color in an image

Asked 12 years, 4 months ago Modified 2 months ago Viewed 95k times



95



I'm looking for a way to find the most dominant color/tone in an image using python. Either the average shade or the most common out of RGB will do. I've looked at the Python Imaging library, and could not find anything relating to what I was looking for in their manual, and also briefly at VTK.

I did however find a PHP script which does what I need, [here](#) (login required to download). The script seems to resize the image to 150*150, to bring out the dominant colors. However, after that, I am fairly lost. I did consider writing something that would resize the image to a small size then check every other pixel or so for it's image, though I imagine this would be very inefficient (though implementing this idea as a C python module might be an idea).

However, after all of that, I am still stumped. So I turn to you, SO. Is there an easy, efficient way to find the dominant color in an image.

[python](#) [image](#) [image-processing](#) [colors](#)

Share Follow

asked Jul 13, 2010 at 22:05



Blue Peppers

3,598 2 21 23

I'm guessing it resizes the picture to let the rescaling algorithm do some of the averaging for you.

– [Skurmedel](#) Jul 13, 2010 at 22:09

11 Answers

Sorted by:

Highest score (default)



87



Here's code making use of [Pillow](#) and [Scipy's cluster package](#).

For simplicity I've hardcoded the filename as "image.jpg". Resizing the image is for speed: if you don't mind the wait, comment out the resize call. When run on this sample image,

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

[Sign up](#)





it usually says the dominant colour is #d8c865, which corresponds roughly to the bright yellowish area to the lower left of the two peppers. I say "usually" because the [clustering algorithm](#) used has a degree of randomness to it. There are various ways you could change this, but for your purposes it may suit well. (Check out the options on the `kmeans2()` variant if you need deterministic results.)

```
from __future__ import print_function
import binascii
import struct
from PIL import Image
import numpy as np
import scipy
import scipy.misc
import scipy.cluster

NUM_CLUSTERS = 5

print('reading image')
im = Image.open('image.jpg')
im = im.resize((150, 150))      # optional, to reduce time
ar = np.asarray(im)
shape = ar.shape
ar = ar.reshape(scipy.product(shape[:2]), shape[2]).astype(float)

print('finding clusters')
codes, dist = scipy.cluster.vq.kmeans(ar, NUM_CLUSTERS)
print('cluster centres:\n', codes)

vecs, dist = scipy.cluster.vq.vq(ar, codes)      # assign codes
counts, bins = scipy.histogram(vecs, len(codes)) # count occurrences

index_max = scipy.argmax(counts)                 # find most frequent
peak = codes[index_max]
colour = binascii.hexlify(bytearray(int(c) for c in peak)).decode('ascii')
print('most frequent is %s (%s)' % (peak, colour))
```

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up



Note: when I expand the number of clusters to find from 5 to 10 or 15, it frequently gave results that were greenish or bluish. Given the input image, those are reasonable results too... I can't tell which colour is really dominant in that image either, so I don't fault the algorithm!

Also a small bonus: save the reduced-size image with only the N most-frequent colours:

```
# bonus: save image using only the N most common colours
import imageio
c = ar.copy()
for i, code in enumerate(codes):
    c[scipy.r_[scipy.where(vecs==i)],:] = code
imageio.imwrite('clusters.png', c.reshape(*shape).astype(np.uint8))
print('saved clustered image')
```

Share Follow

edited Sep 20 at 1:04

answered Jul 14, 2010 at 7:14



[MattDMo](#)

98.7k 20 238 229



[Peter Hansen](#)

20.6k 4 48 72

- 3 Wow. That's great. Almost exactly what I was looking for. I did look at scipy, and had a feeling the answer was somewhere in there :P Thank you for your answer. – [Blue Peppers](#) Jul 14, 2010 at 11:38
- 2 I've edited/updated your code. Thanks for this compact and well working solution! – [Simon Steinberger](#) Dec 3, 2017 at 18:04
- 2 @SimonSteinberger Thanks for the edit, and I'm happy to hear it's still able to run and help someone 7 years later! It was a fun problem to work on. – [Peter Hansen](#) Dec 3, 2017 at 18:15
- 1 this has multiple issues with python 3.x. For example, (1) `.encode('hex')` is [no longer valid syntax](#), and (2) `from PIL import Image` [source](#) – [philshem](#) May 5, 2019 at 12:12
- 2 Thanks @philshem. I believe I've modified it to support 3.x as well now. Some changes done at the same time resolved deprecations and warnings that were reported on either 2.7 or 3.7 (but not necessarily both). – [Peter Hansen](#) May 6, 2019 at 14:18



Try [Color-thief](#). It is based on `Pillow` and works awesome.

49 Installation



```
pip install colorthief
```



Usage



```
from colorthief import ColorThief
color_thief = ColorThief('/path/to/imagefile')
# get the dominant color
dominant_color = color_thief.get_color(quality=1)
```

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up



```
palette = color_thief.get_palette(color_count=6)
```

Share Follow

edited Mar 3, 2021 at 10:30

answered Aug 26, 2018 at 9:01



Artem Bernatskyi

3,795 2 24 33

1 Fantastic module – Trect Jan 13, 2020 at 21:26

1 I am wondering if there is a difference in the correctness of the methods in Top voted, accepted answer and this? I understand the other answer is old and hence might have more votes. – vangap Jan 29 at 11:44

Python Imaging Library has method getcolors on Image objects:

20

im.getcolors() => a list of (count, color) tuples or None

I guess you can still try resizing the image before that and see if it performs any better.

Share Follow

answered Jul 13, 2010 at 23:19



zvone

17.4k 3 44 73

You can do this in many different ways. And you don't really need scipy and k-means since internally Pillow already does that for you when you either resize the image or reduce the image to a certain palette.

Solution 1: resize image down to 1 pixel.

```
def get_dominant_color(pil_img):
    img = pil_img.copy()
    img = img.convert("RGBA")
    img = img.resize((1, 1), resample=0)
    dominant_color = img.getpixel((0, 0))
    return dominant_color
```

Solution 2: reduce image colors to a palette

```
def get_dominant_color(pil_img, palette_size=16):
    # Resize image to speed up processing
    img = pil_img.copy()
    img.thumbnail((100, 100))
```

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up



```
# Find the color that occurs most often
palette = paletted.getpalette()
color_counts = sorted(paletted.getcolors(), reverse=True)
palette_index = color_counts[0][1]
dominant_color = palette[palette_index*3:palette_index*3+3]

return dominant_color
```

Both solutions give similar results. The latter solution gives you probably more accuracy since we keep the aspect ratio when resizing the image. Also you get more control since you can tweak the `palette_size`.

Share Follow

edited Apr 1 at 2:53



Community Bot
1 1

answered May 11, 2020 at 13:27



Pithikos
17.9k 15 110 132

1 This is also leaps and bounds faster than any of the scikit-learn/scipy images above. – [whiteXbread](#)
May 15, 2020 at 2:38

Works like a charm, and doesn't require any additional modules. Thank you so much! – [RealA10N](#) Feb 4, 2021 at 14:53

Thanks for the piece of code, also could you explain how to know what colour is this (Red, blue...). It would be great if you can provide some piece of code. – [Adithya Raj](#) Apr 4 at 12:56



7



It's not necessary to use k-means to find the dominant color as Peter suggests. This overcomplicates a simple problem. You're also restricting yourself by the amount of clusters you select so basically you need an idea of what you're looking at.

As you mentioned and as suggested by zvone, a quick solution to find the most common/dominant color is by using the [Pillow](#) library. We just need to sort the pixels by their count number.

```
from PIL import Image

def find_dominant_color(filename):
    #Resizing parameters
    width, height = 150, 150
    image = Image.open(filename)
    image = image.resize((width, height), resample = 0)
    #Get colors from image object
    pixels = image.getcolors(width * height)
    #Sort them by count number (first element of tuple)
    sorted_pixels = sorted(pixels, key=lambda t: t[0])
    #Get the most frequent color
    dominant_color = sorted_pixels[-1][1]
    return dominant_color
```

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up



The only problem is that the method `getcolors()` returns `None` when the amount of colors is more than 256. You can deal with it by resizing the original image.

In all, it might not be the most precise solution but it gets the job done.

Share Follow

edited Nov 4, 2019 at 9:22

answered Oct 18, 2018 at 17:03



[mobiuscreek](#)

371 4 5

You know what? You are returning the function itself, though you are assigning a value to it but it isn't a good idea – [Black Thunder](#) Oct 31, 2019 at 18:39

You're absolutely right and for that I have edited the name of the function! – [mobiuscreek](#) Nov 4, 2019 at 9:23

This is not very reliable. (1) you should use `thumbnail` instead of `resize` to avoid crop or stretch, (2) if you have an image with 2 white pixels and 100 different levels of blackish pixels, you will still get white. – [Pithikos](#) May 11, 2020 at 11:27

1 Agreed but I wanted to avoid the caveat of reducing the granularity when using predefined clusters or a palette. Depending on the use case this might not be desirable. – [mobiuscreek](#) Jul 10, 2020 at 15:51

If you're still looking for an answer, here's what worked for me, albeit not terribly efficient:

6

```
from PIL import Image

def compute_average_image_color(img):
    width, height = img.size

    r_total = 0
    g_total = 0
    b_total = 0

    count = 0
    for x in range(0, width):
        for y in range(0, height):
            r, g, b = img.getpixel((x,y))
            r_total += r
            g_total += g
            b_total += b
            count += 1

    return (r_total/count, g_total/count, b_total/count)

img = Image.open('image.png')
#img = img.resize((50,50)) # Small optimization
average_color = compute_average_image_color(img)
print(average_color)
```

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up



For png, you need to tweak this slightly to handle the fact that `img.getpixel` returns `r,g,b,a` (four values instead of three). Or it did for me anyway. – [rossdavidh](#) Sep 26, 2016 at 15:28

This weighs pixels unevenly. The final pixel touched contributes half the total value. The pixel before contributes half of that. Only the last 8 pixels will affect the average at all, in fact. – [Russell Borogove](#) Jan 22, 2017 at 13:52

You're right - silly mistake. Just edited the answer - let me know if that works. – [Tim S](#) Jan 23, 2017 at 19:00

11 This is not an answer to this question. Average color is not the dominant color in an image. – [Phani Rithvij](#) Aug 15, 2019 at 13:12

My solution

5

Here's my adaptation based on Peter Hansen's solution.

```
import scipy.cluster
import sklearn.cluster
import numpy
from PIL import Image

def dominant_colors(image): # PIL image input

    image = image.resize((150, 150)) # optional, to reduce time
    ar = numpy.asarray(image)
    shape = ar.shape
    ar = ar.reshape(numpy.product(shape[:2]), shape[2]).astype(float)

    kmeans = sklearn.cluster.MiniBatchKMeans(
        n_clusters=10,
        init="k-means++",
        max_iter=20,
        random_state=1000
    ).fit(ar)
    codes = kmeans.cluster_centers_

    vecs, _dist = scipy.cluster.vq.vq(ar, codes) # assign codes
    counts, _bins = numpy.histogram(vecs, len(codes)) # count occurrences

    colors = []
    for index in numpy.argsort(counts)[::-1]:
        colors.append(tuple([int(code) for code in codes[index]]))
    return colors # returns colors in order of dominance
```

What are the differences/improvements?

It's (subjectively) more accurate

It's using the `kmeans++` to pick initial cluster centers which gives better results. (`kmeans++` may not be the fastest way to pick cluster centers though)

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up



Using `sklearn.cluster.MinibatchKMeans` is significantly faster and gives very similar colors to the default KMeans algorithm. You can always try the slower `sklearn.cluster.KMeans` and compare the results and decide whether the tradeoff is worth it.

It's deterministic

I am using a `random_state` to get consistent output (I believe the original `scipy.cluster.vq.kmeans` also has a `seed` parameter). Before adding a random state I found that certain inputs could have significantly different outputs.

Benchmarks


I decided to very crudely benchmark a few solutions.

Method	Time (100 iterations)
Peter Hansen (kmeans)	58.85
Artem Bernatskyi (Color Thief)	61.29
Artem Bernatskyi (Color Thief palette)	15.69
Pithikos (PIL resize)	0.11
Pithikos (palette)	1.68
Mine (mini batch kmeans)	6.31

Share Follow

edited Apr 13 at 15:41

answered Oct 29, 2020 at 23:12

 **Jacob**
744 5 18


1 Thanks for adding to the solution Jacob! It's gratifying to see this fun question still helping people out. :) – [Peter Hansen](#) Jan 6 at 22:47

4

You could use PIL to repeatedly resize the image down by a factor of 2 in each dimension until it reaches 1x1. I don't know what algorithm PIL uses for downscaling by large factors, so going directly to 1x1 in a single resize might lose information. It might not be the most efficient, but it will give you the "average" color of the image.

Share Follow

answered Jul 13, 2010 at 23:46

 **Russell Borogove**
18k 3 41 49

4

that isn't "RGBA", then you need to apply an alpha mask to convert it to RGBA. You can do that pretty easily with:

```
if im.mode == 'P':
    im.putalpha(0)
```

Share Follow

answered Jan 27, 2011 at 3:52



[Samuel Clay](#)

1,242 1 13 24

2

Below is a c++ Qt based example to guess the predominant image color. You can use PyQt and translate the same to Python equivalent.

```
#include <Qt/QtGui>
#include <Qt/QtCore>
#include <QtGui/QApplication>

int main(int argc, char** argv)
{
    QApplication app(argc, argv);
    QPixmap pixmap("logo.png");
    QImage image = pixmap.toImage();
    QRgb col;
    QMap<QRgb,int> rgbcount;
    QRgb greatest = 0;

    int width = pixmap.width();
    int height = pixmap.height();

    int count = 0;
    for (int i = 0; i < width; ++i)
    {
        for (int j = 0; j < height; ++j)
        {
            col = image.pixel(i, j);
            if (rgbcount.contains(col)) {
                rgbcount[col] = rgbcount[col] + 1;
            }
            else {
                rgbcount[col] = 1;
            }

            if (rgbcount[col] > count) {
                greatest = col;
                count = rgbcount[col];
            }
        }
    }

    qDebug() << count << greatest;
```

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

[Sign up](#)



Share Follow

answered Jan 17, 2012 at 7:44



Ankur Gupta

2,284 4 27 40



This is a complete script with a function `compute_average_image_color()`.

1

Just copy and past it, and change the path of your image.



My image is `img_path='./dir/image001.png'`



```
#AVERAGE COLOR, MIN, MAX, STANDARD DEVIATION  
#SELECT ONLY NOT TRANSPARENT COLOR
```

```
from PIL import Image  
import sys  
import os  
import os.path  
from os import path  
import numpy as np  
import math  
  
def compute_average_image_color(img_path):  
  
    if not os.path.isfile(img_path):  
        print(path_inp_image, 'DONT EXISTS, EXIT')  
        sys.exit()  
  
    #load image  
    img = Image.open(img_path).convert('RGBA')  
    img = img.resize((50,50)) # Small optimization  
  
    #DEFINE SOME VARIABLES  
    width, height = img.size  
    r_total = 0  
    g_total = 0  
    b_total = 0  
    count = 0  
    red_list=[]  
    green_list=[]  
    blue_list=[]  
  
    #READ AND CHECK PIXEL BY PIXEL  
    for x in range(0, width):  
        for y in range(0, height):  
            r, g, b, alpha = img.getpixel((x,y))  
  
            if alpha !=0:  
                red_list.append(r)
```

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up



```
        r_total += r
        g_total += g
        b_total += b
        count += 1

#CALCULATE THE AVRANGE COLOR, MIN, MAX, ETC
average_color=(round(r_total/count), round(g_total/count),
round(b_total/count))
print(average_color)

red_list.sort()
green_list.sort()
blue_list.sort()

red_min_max=[]
green_min_max=[]
blue_min_max=[]

red_min_max.append(min(red_list))
red_min_max.append(max(red_list))
green_min_max.append(min(green_list))
green_min_max.append(max(green_list))
blue_min_max.append(min(blue_list))
blue_min_max.append(max(blue_list))

print('red_min_max: ', red_min_max)
print('green_min_max: ', green_min_max)
print('blue_min_max: ', blue_min_max)

#variance and standard devietion
red_stddev=round(math.sqrt(np.var(red_list)))
green_stddev=round(math.sqrt(np.var(green_list)))
blue_stddev=round(math.sqrt(np.var(blue_list)))

print('red_stddev: ', red_stddev)
print('green_stddev: ', green_stddev)
print('blue_stddev: ', blue_stddev)

img_path='./dir/image001.png'
compute_average_image_color(img_path)
```



449

4

7

Can you explain your code a little bit? (What libraries or modules you used if any and why). It would be nice for others to understand your research, the downsides and upsides of your code and alternatives. It's always better to add some explanation in order to provide context to readers. – [Maicon Mauricio](#)
Jul 1, 2021 at 21:11

Look better. This is a complete python script. There are 7-8 IMPORT instructions. And every line of code is commented. This is a script, so the user can copy and paste it. You have just change the name of the input image image001.png – [quine9997](#) Jul 1, 2021 at 21:31



Highly active question. Earn 10 reputation (not counting the [association bonus](#)) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.