# How can I quantify difference between two images?

Asked 16 years, 2 months ago Modified 1 year, 10 months ago

Viewed 270k times

230

Here's what I would like to do:

I'm taking pictures with a webcam at regular intervals. Sort of like a time lapse thing. However, if nothing has really changed, that is, the picture pretty much *looks* the same, I don't want to store the latest snapshot.

I imagine there's some way of quantifying the difference, and I would have to empirically determine a threshold.

I'm looking for simplicity rather than perfection. I'm using python.

**python** **image-processing** **background-subtraction** **image-comparison** **timelapse**

Share

Improve this question

Follow

edited Jun 4, 2012 at 20:47

Jav_Rock

**22.2k** ● 21 ● 124 ● 165

asked Oct 10, 2008 at 2:39

carrier

**33k** ● 23 ● 79 ● 100

Related: stackoverflow.com/questions/25977/… – Anoyz Jun 9, 2015 at 13:01

25 Answers

Sorted by: Highest score (default)

315

## General idea

Option 1: Load both images as arrays (`scipy.misc.imread`) and calculate an element-wise (pixel-by-pixel) difference. Calculate the norm of the difference.

Option 2: Load both images. Calculate some feature vector for each of them (like a histogram). Calculate distance between feature vectors rather than images.

However, there are some decisions to make first.

## Questions

You should answer these questions first:

- Are images of the same shape and dimension?

  If not, you may need to resize or crop them. PIL library will help to do it in Python.

  If they are taken with the same settings and the same device, they are probably the same.

- Are images well-aligned?

  If not, you may want to run cross-correlation first, to find the best alignment first. SciPy has functions to do it.

  If the camera and the scene are still, the images are likely to be well-aligned.

- Is exposure of the images always the same? (Is lightness/contrast the same?)

  If not, you may want to normalize images.

  But be careful, in some situations this may do more wrong than good. For example, a single bright pixel on a dark background will make the normalized image very different.

- Is color information important?

  If you want to notice color changes, you will have a vector of color values per point, rather than a scalar value as in gray-scale image. You need more attention when writing such code.

- Are there distinct edges in the image? Are they likely to move?

If yes, you can apply edge detection algorithm first (e.g. calculate gradient with Sobel or Prewitt transform, apply some threshold), then compare edges on the first image to edges on the second.

- Is there noise in the image?

  All sensors pollute the image with some amount of noise. Low-cost sensors have more noise. You may wish to apply some noise reduction before you compare images. Blur is the most simple (but not the best) approach here.

- What kind of changes do you want to notice?

  This may affect the choice of norm to use for the difference between images.

  Consider using Manhattan norm (the sum of the absolute values) or zero norm (the number of elements not equal to zero) to measure how much the image has changed. The former will tell you how much the image is off, the latter will tell only how many pixels differ.

## Example

I assume your images are well-aligned, the same size and shape, possibly with different exposure. For simplicity, I convert them to grayscale even if they are color (RGB) images.

You will need these imports:

```
import sys

from scipy.misc import imread
from scipy.linalg import norm
from scipy import sum, average
```

Main function, read two images, convert to grayscale, compare and print results:

```
def main():
    file1, file2 = sys.argv[1:1+2]
    # read images as 2D arrays (convert to grayscale f
    img1 = to_grayscale(imread(file1).astype(float))
    img2 = to_grayscale(imread(file2).astype(float))
    # compare
    n_m, n_0 = compare_images(img1, img2)
    print "Manhattan norm:", n_m, "/ per pixel:", n_m/
    print "Zero norm:", n_0, "/ per pixel:", n_0*1.0/i
```

How to compare. `img1` and `img2` are 2D SciPy arrays here:

```
def compare_images(img1, img2):
    # normalize to compensate for exposure difference,
    # consider disabling it
    img1 = normalize(img1)
    img2 = normalize(img2)
    # calculate the difference and its norms
    diff = img1 - img2  # elementwise for scipy arrays
    m_norm = sum(abs(diff))  # Manhattan norm
    z_norm = norm(diff.ravel(), 0)  # Zero norm
    return (m_norm, z_norm)
```

If the file is a color image, `imread` returns a 3D array, average RGB channels (the last array axis) to obtain

intensity. No need to do it for grayscale images (e.g. `.pgm`):

```
def to_grayscale(arr):
    "If arr is a color image (3D array), convert it to
    if len(arr.shape) == 3:
        return average(arr, -1)  # average over the la
    else:
        return arr
```

Normalization is trivial, you may choose to normalize to [0,1] instead of [0,255]. `arr` is a SciPy array here, so all operations are element-wise:

```
def normalize(arr):
    rng = arr.max()-arr.min()
    amin = arr.min()
    return (arr-amin)*255/rng
```

Run the `main` function:

```
if __name__ == "__main__":
    main()
```

Now you can put this all in a script and run against two images. If we compare image to itself, there is no difference:

```
$ python compare.py one.jpg one.jpg
Manhattan norm: 0.0 / per pixel: 0.0
Zero norm: 0 / per pixel: 0.0
```

If we blur the image and compare to the original, there is some difference:

```
$ python compare.py one.jpg one-blurred.jpg
Manhattan norm: 92605183.67 / per pixel: 13.4210411116
Zero norm: 6900000 / per pixel: 1.0
```

P.S. Entire compare.py script.

## Update: relevant techniques

As the question is about a video sequence, where frames are likely to be almost the same, and you look for something unusual, I'd like to mention some alternative approaches which may be relevant:

- background subtraction and segmentation (to detect foreground objects)
- sparse optical flow (to detect motion)
- comparing histograms or some other statistics instead of images

I strongly recommend taking a look at “Learning OpenCV” book, Chapters 9 (Image parts and segmentation) and 10 (Tracking and motion). The former teaches to use Background subtraction method, the latter gives some info on optical flow methods. All methods are implemented in OpenCV library. If you use Python, I suggest to use OpenCV ≥ 2.3, and its `cv2` Python module.

The most simple version of the background subtraction:

- learn the average value μ and standard deviation σ for every pixel of the background
- compare current pixel values to the range of (μ-2σ,μ+2σ) or (μ-σ,μ+σ)

More advanced versions make take into account time series for every pixel and handle non-static scenes (like moving trees or grass).

The idea of optical flow is to take two or more frames, and assign velocity vector to every pixel (dense optical flow) or to some of them (sparse optical flow). To estimate sparse optical flow, you may use Lucas-Kanade method (it is also implemented in OpenCV). Obviously, if there is a lot of flow (high average over max values of the velocity field), then something is moving in the frame, and subsequent images are more different.

Comparing histograms may help to detect sudden changes between consecutive frames. This approach was used in Courbon et al, 2010:

> *Similarity of consecutive frames.* The distance between two consecutive frames is measured. If it is too high, it means that the second frame is corrupted and thus the image is eliminated. The Kullback–Leibler distance, or mutual entropy, on the histograms of the two frames:

$$d(p,q) = \sum_i p(i) \log \frac{p(i)}{q(i)}$$

where *p* and *q* are the histograms of the frames is used. The threshold is fixed on 0.2.

Share Improve this answer Follow

edited Apr 21, 2015 at 17:15

answered Oct 14, 2010 at 15:43

sastanin

**41.5k** ● 14 ● 105 ● 130

I get a `RuntimeWarning: invalid value encountered in double_scalars` on line 44 ( `return (arr-amin)*255/rng` ) and a `ValueError: array must not contain infs or NaNs` on line 30 ( `z_norm = norm(diff.ravel(), 0)` ) – BioGeek Aug 26, 2015 at 9:47

@BioGeek that is if `rng` equals zero. Just add a check and set `rng = 1` – haisi Apr 24, 2018 at 8:20

88

**A simple solution:**

Encode the image as a **jpeg** and look for a substantial change in **filesize**.

I've implemented something similar with video thumbnails, and had a lot of success and scalability.

edited Oct 10, 2008 at 4:44

Share Improve this answer

Follow

answered Oct 10, 2008 at 2:46

keparo

34k • 13 • 63 • 66

6 This is a very easy, simple solution and is much better than any pixel-wise comparison. If there's a little bit of noise in your webcam's image or if the image is shifted by even one pixel, then a direct comparison will pick up all these meaningless changes. A more robust approach would be to compute the discrete cosine transformation and then compare the images in the frequency domain. Using JPEG compression like this gets you most of the benefits without diving into Fourier theory. – AndrewF Oct 14, 2010 at 17:12

2 Like it. Although other solutions works too, this has a great advantage for a common situation: what if you dont want to save the "base" image? just save the filesize as a hash and then compare just numbers with substraction. In my case I have 4 images, one of them is very simillar and the others 3 are absolutely different. Just scale to the same dimensions, to jpg and substract. Really Nice.
– Diego Andrés Díaz Espinoza Jul 1, 2018 at 2:09

Thanks, worked well for me :-D I used it to extract slices of a quickly scrolled document captured via a videoconference call. Using suggested difference detection applied to adjacent frames I selected key frames were there was a visible content jump. After selecting around 50 images from over 2000 frames of video I could reconstruct the presented document. – Glushiator Aug 23, 2022 at 5:12

In case it's useful, you can encode an image with opencv without saving it with `cv2.imencode('jpg', img)`
– PangolinPaws Sep 15, 2022 at 14:24

71

You can compare two images using functions from PIL.

```
import Image
import ImageChops

im1 = Image.open("splash.png")
im2 = Image.open("splash2.png")

diff = ImageChops.difference(im2, im1)
```

The diff object is an image in which every pixel is the result of the subtraction of the color values of that pixel in the second image from the first image. Using the diff image you can do several things. The simplest one is the `diff.getbbox()` function. It will tell you the minimal rectangle that contains all the changes between your two images.

You can probably implement approximations of the other stuff mentioned here using functions from PIL as well.

Share Improve this answer

Follow

answered Oct 13, 2008 at 6:50

elifiner

7,565 ● 8 ● 42 ● 48

2 I want to save the difference image . means the diff object which hold the difference of images. ow do I save it ? – Sagar Feb 21, 2014 at 6:30

2 @Anthony you can call save() on diff object specifying the image name. like this : diff.save("diff.png") it will save difference image for you. – Sagar Jun 26, 2015 at 8:08

2 NOTE: This has changed the import and it will not work anymore, at least tested in python `3.10`, you have to use `from PIL import Image, ImageChops` – nck Feb 3, 2022 at 14:21

23

Two popular and relatively simple methods are: (a) the Euclidean distance already suggested, or (b) normalized cross-correlation. Normalized cross-correlation tends to be noticeably more robust to lighting changes than simple cross-correlation. Wikipedia gives a formula for the normalized cross-correlation. More sophisticated methods exist too, but they require quite a bit more work.

Using numpy-like syntax,

```
dist_euclidean = sqrt(sum((i1 - i2)^2)) / i1.size

dist_manhattan = sum(abs(i1 - i2)) / i1.size

dist_ncc = sum( (i1 - mean(i1)) * (i2 - mean(i2)) ) / (
  (i1.size - 1) * stdev(i1) * stdev(i2) )
```

assuming that `i1` and `i2` are 2D grayscale image arrays.

Share Improve this answer

Follow

answered Oct 10, 2008 at 3:47

Mr Fooz

112k ● 7 ● 76 ● 103

3 Image cross-correlation functions are built into SciPy (docs.scipy.org/doc/scipy/reference/generated/…), and a fast

version using the FFT is available in stsci python (stsci.edu/resources/software_hardware/pyraf/stsci_python) – endolith Sep 17, 2009 at 16:16

16

A trivial thing to try:

Resample both images to small thumbnails (e.g. 64 x 64) and compare the thumbnails pixel-by-pixel with a certain threshold. If the original images are almost the same, the resampled thumbnails will be very similar or even exactly the same. This method takes care of noise that can occur especially in low-light scenes. It may even be better if you go grayscale.

Share Improve this answer Follow

edited Oct 10, 2008 at 3:49

answered Oct 10, 2008 at 3:37

Ates Goral

**140k** ● 27 ● 141 ● 191

but how would you compare the pixels? – carrier Oct 10, 2008 at 15:13

Once you have the thumbnails, you can simply compare the pixels one by one. You would calculate the "distance" of the RGB values, if you're working in colour or just the difference between the gray tones if you're in grayscale. – Ates Goral Oct 15, 2008 at 3:45

1 "compare the pixels one by one". What does that mean? Should the test fail if ONE of the 64^2 pixel-per-pixel tests fails? – Federico A. Ramponi Oct 15, 2008 at 13:25

What I meant by "compare the thumbnails pixel-by-pixel with a certain threshold" is to come up with a fuzzy algorithm to compare the pixels. If the calculated difference (depends on your fuzzy algorithm) exceeds a certain threshold, the images are "not the same". – Ates Goral Oct 15, 2008 at 16:31

1 Very simple example, without the "fuzzy algorithm": parallel loop through every pixel (compare pixel# *n* of image#1 to pixel# *n* of image#2), and add the difference in value to a variable – mk12 Nov 8, 2009 at 1:00

11

Another nice, simple way to measure the similarity between two images:

```
import sys
from skimage.measure import compare_ssim
from skimage.transform import resize
from scipy.ndimage import imread

# get two images - resize both to 1024 x 1024
img_a = resize(imread(sys.argv[1]), (2**10, 2**10))
img_b = resize(imread(sys.argv[2]), (2**10, 2**10))

# score: {-1:1} measure of the structural similarity b
score, diff = compare_ssim(img_a, img_b, full=True)
print(score)
```

If others are interested in a more powerful way to compare image similarity, I put together a tutorial and web app for measuring and visualizing similar images using Tensorflow.

Share Improve this answer Follow

edited Mar 30, 2018 at 16:29

answered Mar 30, 2018 at 13:00

duhaime

27.6k • 21 • 191 • 242

4 Yes, `skimage` is really nice to use for this application. I use `from skimage.measure import compare_ssim, compare_mse` a lot. skimage.measure docs. – ximiki Oct 11, 2018 at 15:10

2 This code is deprecated. `compare_ssim` is now `structural_similarity`. Use `from skimage.metrics import structural_similarity` and `from imageio import imread`. – shredEngineer Dec 2, 2022 at 13:29

9

I had a similar problem at work, I was rewriting our image transform endpoint and I wanted to check that the new version was producing the same or nearly the same output as the old version. So I wrote this:

https://github.com/nicolashahn/diffimg

Which operates on images of the same size, and at a per-pixel level, measures the difference in values at each channel: R, G, B(, A), takes the average difference of those channels, and then averages the difference over all pixels, and returns a ratio.

For example, with a 10x10 image of white pixels, and the same image but one pixel has changed to red, the

difference at that pixel is 1/3 or 0.33... (RGB 0,0,0 vs 255,0,0) and at all other pixels is 0. With 100 pixels total, 0.33.../100 = a ~0.33% difference in image.

I believe this would work perfectly for OP's project (I realize this is a very old post now, but posting for future StackOverflowers who also want to compare images in python).

Share Improve this answer

Follow

answered May 27, 2018 at 15:27

nicolashahn
539 ● 7 ● 20

---

7

I am addressing specifically the question of how to compute if they are "different enough". I assume you can figure out how to subtract the pixels one by one.

First, I would take a bunch of images with *nothing* changing, and find out the maximum amount that any pixel changes just because of variations in the capture, noise in the imaging system, JPEG compression artifacts, and moment-to-moment changes in lighting. Perhaps you'll find that 1 or 2 bit differences are to be expected even when nothing moves.

Then for the "real" test, you want a criterion like this:

- same if up to P pixels differ by no more than E.

So, perhaps, if E = 0.02, P = 1000, that would mean (approximately) that it would be "different" if any single

pixel changes by more than ~5 units (assuming 8-bit images), or if more than 1000 pixels had any errors at all.

This is intended mainly as a good "triage" technique to quickly identify images that are close enough to not need further examination. The images that "fail" may then more to a more elaborate/expensive technique that wouldn't have false positives if the camera shook bit, for example, or was more robust to lighting changes.

I run an open source project, OpenImageIO, that contains a utility called "idiff" that compares differences with thresholds like this (even more elaborate, actually). Even if you don't want to use this software, you may want to look at the source to see how we did it. It's used commercially quite a bit and this thresholding technique was developed so that we could have a test suite for rendering and image processing software, with "reference images" that might have small differences from platform-to-platform or as we made minor tweaks to tha algorithms, so we wanted a "match within tolerance" operation.



edited Oct 14, 2010 at 16:10

answered Oct 14, 2010 at 16:04

Larry Gritz
13.6k ● 7 ● 43 ● 42

5

Most of the answers given won't deal with lighting levels.

I would first normalize the image to a standard light level before doing the comparison.

Share Improve this answer

Follow

answered Oct 10, 2008 at 3:55

community wiki
Loren Pechtel

If you're taking periodic images and diffing adjacent pairs, you can probably afford to keep the first one after someone turns on the lights. – walkytalky Oct 15, 2010 at 11:03

4

Have you seen the Algorithm for finding similar images question? Check it out to see suggestions.

I would suggest a wavelet transformation of your frames (I've written a C extension for that using Haar transformation); then, comparing the indexes of the largest (proportionally) wavelet factors between the two pictures, you should get a numerical similarity approximation.

Share Improve this answer

Follow

edited May 23, 2017 at 12:34

Community Bot
1 • 1

answered Oct 10, 2008 at 3:13

tzot
95.8k • 30 • 149 • 208

3

I had the same problem and wrote a simple python module which compares two same-size images using pillow's ImageChops to create a black/white diff image and sums up the histogram values.

You can get either this score directly, or a percentage value compared to a full black vs. white diff.

It also contains a simple is_equal function, with the possibility to supply a fuzzy-threshold under (and including) the image passes as equal.

The approach is not very elaborate, but maybe is of use for other out there struggling with the same issue.

https://pypi.python.org/pypi/imgcompare/

Share Improve this answer Follow

answered Nov 21, 2016 at 0:23

datenhahn
141 • 3

3

I apologize if this is too late to reply, but since I've been doing something similar I thought I could contribute somehow.

Maybe with OpenCV you could use template matching. Assuming you're using a webcam as you said:

1. Simplify the images (thresholding maybe?)
2. Apply template matching and check the max_val with minMaxLoc

Tip: max_val (or min_val depending on the method used) will give you numbers, large numbers. To get the difference in percentage, use template matching with the same image -- the result will be your 100%.

Pseudo code to exemplify:

```
previous_screenshot = ...
current_screenshot = ...

# simplify both images somehow

# get the 100% corresponding value
res = matchTemplate(previous_screenshot, previous_scre
_, hundred_p_val, _, _ = minMaxLoc(res)

# hundred_p_val is now the 100%

res = matchTemplate(previous_screenshot, current_scree
_, max_val, _, _ = minMaxLoc(res)

difference_percentage = max_val / hundred_p_val

# the tolerance is now up to you
```

Hope it helps.

Share Improve this answer

Follow

answered Jun 15, 2018 at 16:08

zanfranceschi
51 • 4

2

you can compute the histogram of both the images and then calculate the Bhattacharyya Coefficient, this is a very fast algorithm and I have used it to detect shot changes in a cricket video (in C using openCV)

Share Improve this answer

Follow

edited Feb 24, 2011 at 20:44

endolith

**26.7k** ● 35 ● 135 ● 202

answered Feb 24, 2011 at 6:00

vishalv2050

**863** ● 1 ● 10 ● 19

Could you calculate the coefficient on the images themselves? – endolith Feb 24, 2011 at 20:35

You will have to calculate the histograms for the images (with the bin size of the histogram as per requirements).
– vishalv2050 Mar 18, 2011 at 5:51

2

```
import os
from PIL import Image
from PIL import ImageFile
import imagehash

#just use to the size diferent picture
def compare_image(img_file1, img_file2):
    if img_file1 == img_file2:
        return True
    fp1 = open(img_file1, 'rb')
    fp2 = open(img_file2, 'rb')

    img1 = Image.open(fp1)
    img2 = Image.open(fp2)
```

```
    ImageFile.LOAD_TRUNCATED_IMAGES = True
    b = img1 == img2

    fp1.close()
    fp2.close()

    return b




#through picturu hash to compare
def get_hash_dict(dir):
    hash_dict = {}
    image_quantity = 0
    for _, _, files in os.walk(dir):
        for i, fileName in enumerate(files):
            with open(dir + fileName, 'rb') as fp:
                hash_dict[dir + fileName] =
imagehash.average_hash(Image.open(fp))
                image_quantity += 1

    return hash_dict, image_quantity

def compare_image_with_hash(image_file_name_1, image_f
    """
    max_dif: The maximum hash difference is allowed, t
accurate, the minimum is 0.
    recommend to use
    """
    ImageFile.LOAD_TRUNCATED_IMAGES = True
    hash_1 = None
    hash_2 = None
    with open(image_file_name_1, 'rb') as fp:
        hash_1 = imagehash.average_hash(Image.open(fp)
    with open(image_file_name_2, 'rb') as fp:
        hash_2 = imagehash.average_hash(Image.open(fp)
    dif = hash_1 - hash_2
    if dif < 0:
        dif = -dif
    if dif <= max_dif:
        return True
    else:
```

```
        return False


def compare_image_dir_with_hash(dir_1, dir_2, max_dif=
    """
    max_dif: The maximum hash difference is allowed, t
accurate, the minimum is 0.

    """
    ImageFile.LOAD_TRUNCATED_IMAGES = True
    hash_dict_1, image_quantity_1 = get_hash_dict(dir_
    hash_dict_2, image_quantity_2 = get_hash_dict(dir_

    if image_quantity_1 > image_quantity_2:
        tmp = image_quantity_1
        image_quantity_1 = image_quantity_2
        image_quantity_2 = tmp

        tmp = hash_dict_1
        hash_dict_1 = hash_dict_2
        hash_dict_2 = tmp

    result_dict = {}

    for k in hash_dict_1.keys():
        result_dict[k] = None

    for dif_i in range(0, max_dif + 1):
        have_none = False

        for k_1 in result_dict.keys():
            if result_dict.get(k_1) is None:
                have_none = True

        if not have_none:
            return result_dict

        for k_1, v_1 in hash_dict_1.items():
            for k_2, v_2 in hash_dict_2.items():
                sub = (v_1 - v_2)
                if sub < 0:
                    sub = -sub
                if sub == dif_i and result_dict.get(k_
                    result_dict[k_1] = k_2
```

```
                    break
    return result_dict


def main():
    print(compare_image('image1\\815.jpg', 'image2\\5.
    print(compare_image_with_hash('image1\\815.jpg', '
    r = compare_image_dir_with_hash('image1\\', 'image
    for k in r.keys():
        print(k, r.get(k))


if __name__ == '__main__':
    main()
```

- output:

  False

  True

  image2\5.jpg image1\815.jpg

  image2\6.jpg image1\819.jpg

  image2\7.jpg image1\900.jpg

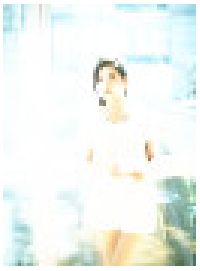
  image2\8.jpg image1\998.jpg

  image2\9.jpg image1\1012.jpg

- the example pictures:

- 815.jpg

- 5.jpg



Share Improve this answer

Follow

edited Jun 26, 2020 at 15:20

林果皞

**7,773** ● 4 ● 58 ● 74

answered May 22, 2017 at 7:04

admin

**169** ● 1 ● 6

---

1

Earth movers distance might be exactly what you need. It might be *abit* heavy to implement in real time though.

Share Improve this answer

Follow

answered Oct 10, 2008 at 2:48

shoosh

**78.8k** ● 57 ● 213 ● 331

I don't really feel this answer adresses nicely : "I'm looking for simplicity rather than perfection. I'm using python." – PilouPili Oct 28, 2019 at 11:04

I think as this question thread gets a lot of traffic and the title which draws in most viewers is about how to quantify the difference between two images, it has value here.
– Daniel Oram Oct 28, 2019 at 11:24

1

What about calculating the Manhattan Distance of the two images. That gives you n*n values. Then you could do something like an row average to reduce to n values and a function over that to get one single value.

Share Improve this answer

Follow

answered Oct 10, 2008 at 2:53

Tobias

**5,108** ● 8 ● 37 ● 41

1

I think you could simply compute the euclidean distance (i.e. sqrt(sum of squares of differences, pixel by pixel)) between the luminance of the two images, and consider them equal if this falls under some empirical threshold. And you would better do it wrapping a C function.

Share Improve this answer

Follow

edited Oct 10, 2008 at 4:08

answered Oct 10, 2008 at 2:48

Federico A. Ramponi
**47k** ● 30 ● 111 ● 133

1

I have been having a lot of luck with jpg images taken with the same camera on a tripod by (1) simplifying greatly (like going from 3000 pixels wide to 100 pixels wide or even fewer) (2) flattening each jpg array into a single vector (3) pairwise correlating sequential images with a simple correlate algorithm to get correlation coefficient (4) squaring correlation coefficient to get r-square (i.e fraction of variability in one image explained by variation in the next) (5) generally in my application if r-square < 0.9, I say the two images are different and something happened in between.

This is robust and fast in my implementation (Mathematica 7)

It's worth playing around with the part of the image you are interested in and focussing on that by cropping all images to that little area, otherwise a distant-from-the-camera but important change will be missed.

I don't know how to use Python, but am sure it does correlations, too, no?

Share Improve this answer

Follow

answered Feb 19, 2011 at 21:28

Roman Dial

11 ● 1

1

Check out how Haar Wavelets are implemented by [isk-daemon](). You could use it's imgdb C++ code to calculate the difference between images on-the-fly:

isk-daemon is an open source database server capable of adding content-based (visual) image searching to any image related website or software.

This technology allows users of any image-related website or software to sketch on a widget which image they want to find and have the website reply to them the most similar images or simply request for more similar photos at each image detail page.

Share Improve this answer

Follow

answered Jan 16, 2012 at 21:11



Ricardo Cabral

785 ● 1 ● 8 ● 17

1

A somewhat more principled approach is to use a global descriptor to compare images, such as GIST or CENTRIST. A hash function, as described here, also provides a similar solution.

Share Improve this answer

Follow

answered Nov 29, 2017 at 0:53

Felix Goldberg

421 ● 4 ● 21

1

There's a simple and fast solution using numpy by calculating mean squared error:

```
before = np.array(get_picture())
while True:
    now = np.array(get_picture())
    MSE = np.mean((now - before)**2)

    if  MSE > threshold:
        break

    before = now
```

Share Improve this answer

Follow

answered Jul 18, 2019 at 16:26

arianvc

101 • 10

1

Here is a function I wrote, which takes 2 images (filepaths) as arguments and returns the average difference between the two images' pixels' components. This worked pretty well for me to determine visually "equal" images (when they're not `==` equal).

*(I found 8 to be a good limit to determine if images are essentially the same.)*

*(Images must have the same dimensions if you add no preprocessing to this.)*

```
from PIL import Image

def imagesDifference( imageA, imageB ):
    A = list(Image.open(imageA, r'r').convert(r'RGB').
    B = list(Image.open(imageB, r'r').convert(r'RGB').
    if (len(A) != len(B)): return -1
    diff = []
    for i in range(0, len(A)):
        diff += [abs(A[i][0] - B[i][0]), abs(A[i][1] -
```

```
B[i][2])]
    return (sum(diff) / len(diff))
```

Share Improve this answer Follow

edited Feb 27, 2021 at 23:01

answered Feb 27, 2021 at 19:19

Pedro V. G.
353 ● 1 ● 4 ● 14

---

0

There are many metrics out there for evaluating whether two images look like/how much they look like.

I will not go into any code here, because I think it should be a scientific problem, other than a technical problem.

Generally, the question is related to human's perception on images, so each algorithm has its support on human visual system traits.

Classic approaches are:

Visible differences predictor: an algorithm for the assessment of image fidelity (https://www.spiedigitallibrary.org/conference-proceedings-of-spie/1666/0000/Visible-differences-predictor--an-algorithm-for-the-assessment-of/10.1117/12.135952.short?SSO=1)

Image Quality Assessment: From Error Visibility to Structural Similarity

(http://www.cns.nyu.edu/pub/lcv/wang03-reprint.pdf)

FSIM: A Feature Similarity Index for Image Quality Assessment (https://www4.comp.polyu.edu.hk/~cslzhang/IQA/TIP_IQA_FSIM.pdf)

Among them, SSIM (Image Quality Assessment: From Error Visibility to Structural Similarity ) is the easiest to calculate and its overhead is also small, as reported in another paper "Image Quality Assessment Based on Gradient Similarity" (https://www.semanticscholar.org/paper/Image-Quality-Assessment-Based-on-Gradient-Liu-Lin/2b819bef80c02d5d4cb56f27b202535e119df988).

There are many more other approaches. Take a look at Google Scholar and search for something like "visual difference", "image quality assessment", etc, if you are interested/really care about the art.

Share Improve this answer

Follow

answered May 7, 2019 at 15:57

cyfex

55 ● 10

---

0

Use SSIM to measure the Structural Similarity Index Measure between 2 images.

Share Improve this answer

Follow

answered Sep 9, 2021 at 12:43

Saqib Shakeel

685 ● 2 ● 9 ● 18

Check out this quite useful python package if someone may need to check image quality metric. project sewar

0

Share Improve this answer

Follow

answered Feb 10, 2023 at 10:35

mustafa can nacak
171 ● 1 ● 2 ● 5