Creating A Mosaic of Several Tiny Images Design Report

Jinhang Zhu

École Centrale de Nantes, France

jinhang.zhu@eleves.ec-nantes.fr

Abstract

A mosaic is an image which is decomposed into small rectangular patches, which is replaced by a host of images that are similar to the original image patches separately [1]. In image processing, to create a mosaic, the source image needs to be split into small pieces, which are compared to each image of a large set of images, based on a standard which assesses the similarity. In order to quantify the similarity of two images, methods including calculating the Euclidean distance between two images [2], calculating the Hamming distance via Hashing algorithm [3], implement image specification [10], etc. In this project, luminance differences of three channels and the Euclidean distances of three channels are computed, which, coupled with a certain threshold of the luminance differences, decides the most similar patch for each tile of the original image.

Keywords: image processing, mosaic, Euclidean distance

1 Introduction

1.1 Overview of the algorithm

In order to obtain the most similar image from the image package to substitute the pad of the original image, an algorithm is designed to implement this function.

Generally, the idea is to get an image which has the most similar luminance differences in blue, green and red channels between the patches and the corresponding tiles of the original image. Moreover, the Euclidean distances of the histograms in blue, green and red channels between the patches and the corresponding tiles of the original image are computed. Then all patches from the image package are iterated to find the patch whose luminance differences and Euclidean distances of histograms are all the lowest BUT MAYBE NOT the lowest separately in each channel. Meanwhile, only those patches whose luminance differences are lower than a threshold will be chosen. The threshold is chosen as 50 in this project.

Note that:

(1) Threshold can be modified higher if there is no patch in the package that satisfies the threshold.
(2) The database of this project has 466 images.

1.2 Uniqueness

1.2.1 Color features are well preserved

Instead of calculating a single histogram and the Euclidean distance between the input patch and the tile of the original image, all Euclidean distances in the three channels are obtained to make a decision. In contrast, single Euclidean distance only implies the difference in the gray value distribution between two images. Covering all three channels can use more color features to find the similarity.

1.2.2 Luminance difference avoids wrong colors

If only Euclidean distance is used to decide, there may be a situation where the program choose an unsuitable image. For example, the tile of the original image has red tone, which means the value of the red channel is relatively greater than those of the other two. As has been mentioned above, the Euclidean distance satisfy that three of them are all lower than those of the previous image but may not be the lowest in each channel. Different iterating order among the resource images leads to choosing different patch. It is noticeable that there is no image whose Euclidean distances are all the lowest in three channels than any others.

Therefore, it may lead to a situation where the chosen patch is similar to the tile in red channel but are obviously different in the other two channels. Provided that the value in the blue channel of the patch is greater than its red channel value. Because of this, the patch turns out to be blue rather than red. For instance, the right image of figure 1.2 illustrates the color inconsistency if there is no luminance check.

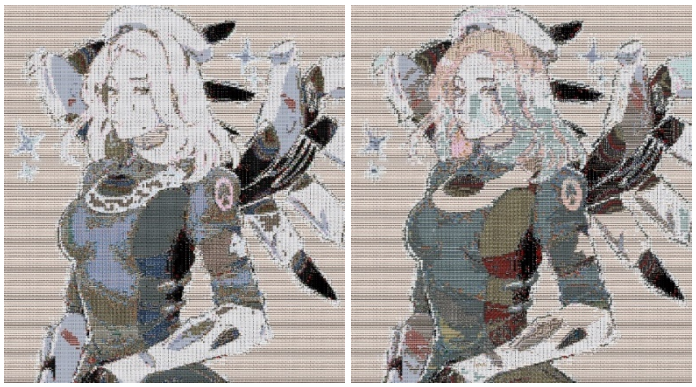

Figure 1.2 Luminance check vs No luminance check

In order to avert this exception, an auxiliary criterion should be used to make sure the patch and the tile perform similarly in each channel. An easy way to solving the problem is to use average value, which is called luminance in image processing. The lowest luminance differences in three channels or even luminance differences below a certain value can help prevent wrong colors.

1.2 Comparison between algorithms

The image tested is as figure 1.3.

Figure 1.3 Original image

Figure 1.4 Luminance, Euclidean distance of images

database of 102 images, duration time: 92.7959 s

1.2.1 Single luminance difference, Euclidean distance between images

In this case, luminance difference and Euclidean distance between images are used to choose the most similar patch. The output image is as figure 1.4. Obviously, although the process is fast, the image is like a grayscale image and is in shortage of color features.

1.2.2 Single luminance difference, Euclidean distance between histograms

In this case, luminance difference and Euclidean distance between histograms are used to choose the most similar patch. The output image is as figure 1.5. The output is more colorful, but colors of some elements are changed such as the white background.

Figure 1.5 Luminance, Euclidean distance of histograms database of 102 images, duration time: 195.8768 s

Figure 1.6 Image specification database of 102 images, duration time: 803.2086 s

1.2.3 Image specification

This is a solution by which the resource images are modified to be similar to the original image. Generally, image specification or image matching is used to specify the histogram of the patch to the that of the tile. The result is shown as figure 1.6. The output image well preserves the color features of the original image. However, the duration time is too long.

1.2.4 Single Luminance difference, Euclidean distances between histograms

In this case, luminance difference and Euclidean distance between histograms in channels are used to choose the most similar patch. See figure 1.7. The color features are more abundant but, in some areas, colors are changed, such as the red armband.

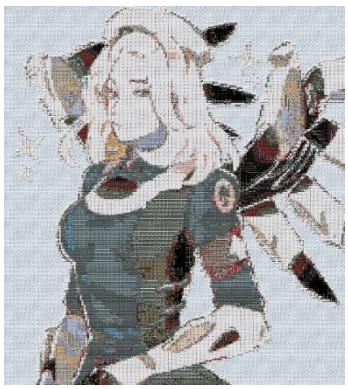

Figure 1.7 Single luminance database of 102 images, duration time: 182.8161 s

Figure 1.8 3 Channels Luminance database of 102 images, duration time: 247.5845 s

1.2.5 Channels' Luminance difference, Euclidean distances between histograms

In this case, luminance differences and Euclidean distances between histograms in channels are used to choose the most similar patch. See figure 1.8. Significantly, the output is much better than the previous ones, especially in terms of the performance in color features.

2 Implementation

2.1 Imported Packages

2.1.1 re

Regular expression operations [4]. This module offers some well-arranged expression operations. *re.search* scan through the filename looking for the first location where the extension '.jpg' matches. If there is no match, then it returns None. *re.I* is the flag which means both capital or lowercase letters can be searched.

This line of code requires a '.jpg' extension of the image filename.

2.1.2 os

Miscellaneous operating system interfaces [5]. *os.listdir* returns a lost of names of the entries in the directory *img_dir. os.path.join* creates the entry's fullname.

2.1.3 multiprocessing

Process-bases "threading" interface [6]. It supports spawning processes using an API similar to the threading module. *mp.JoinableQueue()* uses *JoinableQueue* class to create a new queue which additionally has *task_done()* and *join(). mp.process()* create 5 processes which run simultaneously. Every process takes the queue and uses the objects in the queue to function. The specifications of *mp.process()* has a target, usually a function, which will be called every time when a process runs. The args provide the input arguments for the invoked function *find_closest_image(). Daemon* decides that the process will not have a child process if terminated. *p.start()* starts a process. *join_queue_put()* puts objects into the queue. *Join_queue.join()* makes processes join the queue. *q.task_done()* is necessary for *JoinableQueue* class, and it terminates a process at a time. *RawArray* returns a *ctypes* array allocated from shared memory, which is quite useful when there are shared materials that the processes all use. The size is *len(resource_images.flatten())*, which means all information of the images are saved in memory in the shape of flattened array. *np.frombuffer* gets the array from the memory buffer and is copied to resource_images via *np.copyto()* [13].

2.1.4 matplotlib.pyplot

Python 2D plotting library [7]. *pyplot.py* is used to show the original image and the mosaic in a figure.

2.1.5 scipy.spatial.distance

Computing distance matrix from arrays [8]. *euclidean()* computes the euclidean distance between two 1-D arrays. It is actually the norm of the element-wise subtraction of two 1-D arrays.

2.1.6 tqdm

Can show a progress meter via wrapping the iteration [9].

2.1.7 time

To get the processing time using *time.time()*. The subtraction of two time points makes the duration time.

2.2 Python implementation

2.2.1 Read and resize the original image

cv2.imread(infile) reads the image whose path is infile. *get_set_shape()* returns a fixed shape at [32, 32], which is the shape of patches or tiles that will compose a mosaic.

Then the original image is resized so that the height and width of it are the multiple of the *set_height* and *set_width*, in which case the original can be composed of integer number of tiles instead of a float number.

Suppose the original image size is [762, 477]. The *RATIO* is 10. Then the height will be rounded to int(762/32) * 32 * 10 = 24 * 32 * 10 = 32 * 240. The width will be rounded to int(477/32) * 32 * 10 = 15 * 32 * 10 = 32 * 150. Therefore, 240 * 150 patches are collected to make a mosaic.

2.2.2 Load resource images

Firstly a list is created, each time a new image is read from the directory, it will be appended to the end of the list. After reading all images, the list is changed to *numpy.array* type and all values be rounded to unsigned integer 8-bit.

2.2.3 Create space in shared memory

Save the resource images and the empty output image into the shared memory so that all processes can access the memory, read information from resource images and modify values of the output image.

2.2.4 Start multi processes

Create a queue and several processes (5 processes are used in the project). Each process will invoke the function *find_closest_image()* once to modify a file in the output image at a time when the queue provides a new set of the position of the patch and the pad from the resized original image at the same position.

2.2.5 Find the closest patch

Firstly, the resource images are recovered to their original shape from the memory buffer. Then the set of the position and the resized original image's pad at the position is obtained from the queue for future use. Then use *len(pad.shape)* to judge if the output image is grayscale.

2.2.5.1 When the output image is non-grayscale

Define the minimum values of the Euclidean distances of histograms in b, g, r channels and the minimum values of the luminance differences in b, g, r channels.

Iterate the images in the resource. For each image, calculate the Euclidean distances of histograms and the luminance differences between the image and the pad of the original image.

Then a threshold condition is designed to make sure only images whose channels' luminance performances are close to those of the pad will be the participants into the selection.

When all Euclidean distances and luminance differences are lower than the minimum values, the minimum values will be updated, and the by-now most similar patch will be updated as this image.

After the iterating of all images, the result image is recovered to its original shape from the buffer, and the tile at the same position will be substituted with the found most similar patch.

2.2.5.2 When the output image is grayscale

Use single luminance difference and single Euclidean distance between the image and the pad of the original image. The rest process is generally the same as above.

2.2.6 Write and show

Recover the result image to its original shape and write it as a '.jpg' image file in the directory. If necessary, the original image and the mosaic can be displayed using *matplotlib.pyplot* in a figure for comparison.

Note that image read from *cv2.imread()* is in BGR arrangement while the *plt.imshow()* will only display images in RGB. Thus, a color conversion is needed to make the plotting function well [11].

2.3 DIP Concepts in the codes

2.3.1 Luminance

Luminance is the average value for the gray values of the image. If the image is colored, it will have three luminance values in channels.

2.3.2 Histogram

Histogram is a graph (line chart or bar chart) which illustrates the distribution percentages of each level of gray values. The horizonal coordinates are levels of gray values (if the image is 8-bit, there will be 2^8 = 256 levels). The vertical coordinates are the probabilities of each level, often range from 0 to 1.

In order to get the histogram in a channel. Firstly, a 256-level array is created, which has indices range from 0 to 255. Iterate the image pixels, if the gray value is g, then the count for the value g will plus 1. That is *array[g] = array[g] +1*.

After iterating all pixels, an array composed of counts for all levels is accomplished [12]. Then it will be divided by the size (height* width) of the image to make the counts values scaled down to [0, 1]. Then the Euclidean distance can be computed using two histograms of the two images. *Euclidean distance = norm(histogram1-histogram2)*

3 Conclusion

This is design is not the best due to its relatively long processing time compared to that of the design, in which only Euclidean distance of images is used. But the duration time is shorter than that of the design which implements the function using Hashing algorithm or specification. Another test is made to make a mosaic using 10 resource images

without multiprocessing and it costs processing time: 98.6782 s. Fortunately, multiprocessing can reduce the time more than 5 times in the project because the tasks or the processes of the calculation run in parallel mode while the design without multiprocessing run in serial mode. Once a task is finished, a new task can be addressed regardless of other running tasks. Hashing algorithm is also tested but it costs also much longer than this design. Therefore, both two solutions are not used despite that they produce good mosaic.

Admittedly, it is not realistic that the self-made function like *hamming_distance(str1, str2)* and *tf.matching(source, dest)* are undoubtedly better than those from existing packages like *scipy.spatial.distance. hamming(u, v[, w])*. They are usually highly optimized and have endured a host of tests before the release.

Thanks to the professor Diana Mateus Lamus for choosing this impressive project. It generally improved my understanding of image processing and Python language. Specifically, I have an insight into multiprocessing, which is significantly useful and helpful in future project involving a great amount of iteration or calculation. Besides, I entered a new world of similar image recognition. There are many ways to finding a similar image such as Hamming distance, Euclidean distance, object matching, etc. It is enjoyable and exciting for me to be exposed to new knowledges.

References

[1] Szul, Piotr, and Tomasz Bednarz. "Productivity frameworks in big data image processing computations-creating photographic mosaics with Hadoop and Scalding." Procedia computer science 29 (2014): 2306-2314.

[2] Zhongqiang, Shen. mosaic/main.py at master pythonml/mosaic. 12 Sep. 2018, https://github.com/pythonml/mosaic/blob/master/main.py.

[3] leeafay. Using Hamming distance to do image recognition. 09 Aug. 2018, https://blog.csdn.net/leeafay/article/details/53907394

[4] Python Software Foundation. re – Regular expression operations. 27 March. 2019, https://docs.python.org/3/library/re.html#re.I

[5] Python Software Foundation. os – Miscellaneous operating system interface. 27 March. 2019, https://docs.python.org/3/library/os.html

[6] Python Software Foundation. multiprocessing – Process-based “threading” interface. 18 March. 2019, https://docs.python.org/2/library/multiprocessing.html

[7] John Hunter, Darren Dale, Eric Firing, Michael Droettboom and the Matplotlib development team. Matplotlib: Python plotting. 18 March. 2019, https://matplotlib.org/

[8] The SciPy community. Distance computations. 10 Feb. 2019, https://docs.scipy.org/doc/scipy/reference/spatial.distance.html

[9] tqdm. A fast, extensible progress bar for Python and CLI. 17 Feb. 2019, https://github.com/tqdm/tqdm

[10] Jinhang Zhu. myimpy`PyPI. 13 Mar. 2019, https://pypi.org/project/myimpy/#description

[11] laks. Difference between plt.show and cv2.imshow. 11 Aug. 2016, https://stackoverflow.com/questions/38598118/difference-between-plt-show-and-cv2-imshow

[12] OpenCV. OpenCV: Histograms. 18 Dec. 2015, https://docs.opencv.org/3.1.0/d6/dc7/group__imgproc__hist.html#ga4b2b5fd75503ff9e6844cc4dcdaed35d

[13] Mianzhi Wang. On Sharing Large Arrays When Using Python’s Multiprocessing. 7 Mar. 2018, https://research.wmz.ninja/articles/2018/03/on-sharing-large-arrays-when-using-pythons-multiprocessing.html