The Visualized Color Quantization by K-means in Processing and Its Performance in Machine Learning

Senyuan Liu [a*]
[a*]Dept. of Statistics and Applied Probability, University of California, Santa Barbara
Goleta, California, CA USA 93106
[*] Corresponding author: senyuan@ucsb.edu

Abstract

The forms of data could be varied in many other ways but not limited to numerical value, for example images. Therefore, it arouses my interests to visualize several color quantization methods to extract color from image. In this project, images will be processed using color quantization algorithm based on K-Means in RGB and K-means in HSB to reduce the number of colors in the image, realized in Processing 4.0.3. I will then introduce and implement three methods that compare image similarity in python to compare the performance of RGB and HSB. After selecting the color space which has better accuracy, I will use Res-Net18 image classification machine learning model to test the performance of K-means in color quantization by comparing the training time and training results. This project uses Natural_Image Dataset from Kaggle, which contains 6,899 images from 8 distinct classes compiled from various sources. The classes include airplane, car, cat, dog, flower, fruit, motorbike, and person.

1. Introduction

Definition 1.1

Color space: A color space is a specific organization of colors. It specifies a mapping between numeric values and specific colors, which are called "primaries" in that color space.

Definition 1.2

RGB: RGB color space is an additive color space: it adds different amounts of red, green and blue light together to create different colors. In a 24-bit image, each component is expressed as a number from 0 to 255.

Definition 1.3

HSB: The HSB color model uses hue (H), saturation (S), and brightness (B) as components for defining color. HSB is also known as HSV (with the components hue, saturation, and value). Hue describes the pigment of a color and is expressed in degrees to represent the location on the standard color wheel.

Color quantization is an important operation with many applications in graphics and image processing. The algorithm is to select the most representative color and reduce the redundant color in the image as much as possible. To be more specific, it is to compress the image by replacing multiple colors to one similar color. In this project, we will apply two color space: one is Red-Green-Blue(RGB), the other one is Hue-Saturation-Brightness(HSB). Using the K-Means algorithm to perform unsupervised clustering on these pixels with specific colors, color quantization can be realized. The use of K-Means for color quantization of images can reduce the number of colors in those images, so that images can be reproduced well in lower performance

computer equipment. At the same time, color quantization reduces the size of images and improves the efficiency of image processing as well as image analysis.

2. K-means in RGB and HSB color space (Realized in Processing)

Definition 2.1

K-means clustering: *k*-means clustering is a method of vector quantization, originally from single processing, that aims to partition *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster.

K-means clustering will group similar colors together into 'k' clusters of different colors (RGB values & HSB values). Therefore, each cluster centroid is the representative of the color vector in RGB or HSB color space of its respective cluster. All of this part is done in Processing 4.0.3. In Processing, I mainly use the ‘round()’ built-in function and other algorithms to realize color quantization and K-means methods by clustering similar colors. For example, in the HSB color space method, I use:

```
ic=color(image.pixels[x+y*image.width]);
h=round(hue(ic));
s=round(saturation(ic));
b=round(brightness(ic));
```

And in RGB color space method, I use:

```
color ic=color(image.pixels[x+y*image.width]);
 r=round(red(ic));
 g=round(green(ic));
 b=round(blue(ic));
```

To visualize the color quantization process, I extract the colors and make each color as a circle. The portion each color takes up in the image decides the size of the circle. The size of the circle and the portion it takes is proportional.

I use the famous painting “The Starry Night” as an example, as shown below with different “k” values. These results are produced by HSB color space and can be found in the folder named “hsb”. The very left side of the images represent the images after processed by HSB. The center of the region (small box) presents the color quantization in a visual way. And the right side is the original images for us to easily compare.

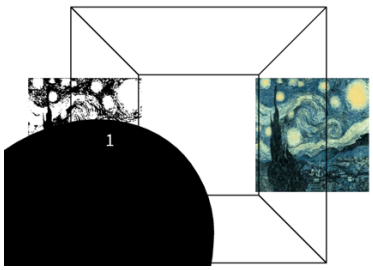

Figure 2.1 : K=1

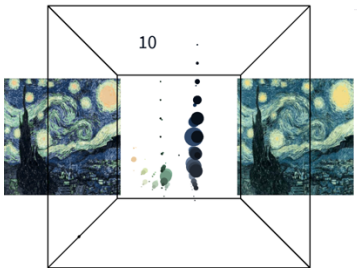

Figure 2.2: K=10

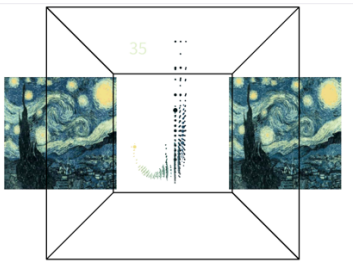

Figure 2.3: K = 35

Figure 1-3 shows the color quantization process corresponding to the different K values. Take Figure 1 as an example, the K value is 1, which means the total number of colors in this image is 1*1*1 = 1. Therefore, we can only see one color which is black in this picture. Since the black takes up about 60% percent in the whole picture, the size of the black circle to the size of the white in the small box is about 6 : 4. The visualization changes when I set the K value to 10. Theoretically, there will be a presentation of 10*10*10 = 100 circles of colors in the box. Some of them can barely see because they take up too few space in the images. The same reason also applies for Figure 3 when K comes to 35.

The following images (Figures4-6) are another example which is an image with less colors. This may look more directly. I also would like to mention that my code has many functions. You can put as many images as you want to process and simply press “N” to switch to next image. You also need to press “Space Bar” to get the color quantization results (stuff in the middle small box). You are able to increase and decrease the K values by pressing “-” and “Shift” + “+”.

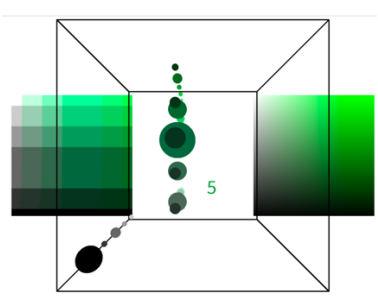

Figure 2.4: K = 5

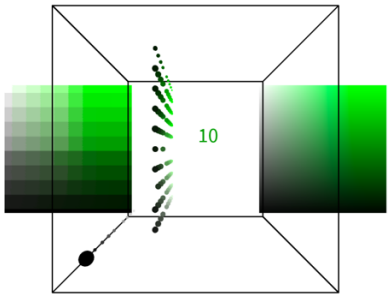

Figure 2.5: K = 10

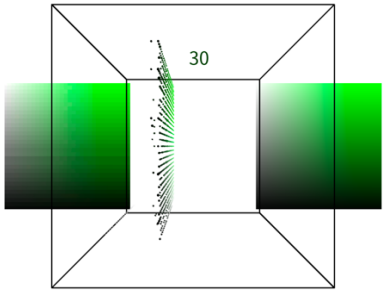

Figure 2.6: K=30

The same procedures and process also apply on the K-means in RGB code. Those who are interested in that could also check that part in the folder named “rgb”. All the operations are the same with HSB. There will be visual difference because the performance of the two methods is different. I will show how they are different numerically later. To avoid repetition, I skip showing RGB part.

These two programs implement the single image color quantization by K-means in RGB and HSB color space. However, to process a large dataset which includes 6899 images and output the processed images, I need a program which generates all the images in one time. This is what leads to the second program, named “saving program”. It can be found in the folder named “output”. The procedure is to put the input data along with the program in the same folder and change the path name inside the function. At the end, the program will produce a folder which include all the images that are being processed, which is named “output_data”.

3. K-means methods comparison by Hash algorithms (Realized in Anaconda Python)

To compare the K-means methods, we need to compare their output results which is to compare the image similarity. There are totally three ways to realize the image difference: Average Hash Algorithm (aHash), Difference Hash Algorithm (dHash), and Perceptual Hash Algorithm (pHash). Before moving to explain the three methods, I would like to mention some basic definitions that will be involved in the methods.

Definition 3.1

Hamming distance: Hamming distance is the distance between two equal-length strings of symbols is the number of positions at which the corresponding symbols are different.

Take string 1010 and 1110 as an example, the Hamming distance is 1, because the only difference between these two strings is the first position (starting from 0). First string's first position is 0 while the second string's is 1. This leads to Hamming distance is 1.

Definition 3.2

Definition- Discrete Cosine Transform (DCT): DCT expresses a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies. It transfers the pixel domain into the frequency domain.

DCT helps separate the image into parts of differing importance (with respect to the image's visual quality)

I list the steps and they correspond to my code:

1. Average Hash Algorithm (aHash)
    a. Reduce the image scale to 8*8 (so keep the basic structure, reduce the details)
    b. Grayscale the image (0-255)
    c. Calculate the average of all the pixels values

d. Compare: The pixel value greater than the average is marked as 1 and the opposite is 0 for a total of 64 bits.

e. Produce hash value (two strings with 1 and 0)

f. Calculate the Hamming distance

2. Difference Hash Algorithm (dHash)

    a. Reduce the image scale to 8*9

    b&c…

    d. Compare: A pixel value greater than the latter is denoted as 1, and the opposite is denoted as 0. This line is not compared to the next line, each line is 9 pixels, 8 differences, there are 8 lines, total 64 bits

3. Perceptual Hash Algorithm (pHash)

    a. Grayscale the image

    b. Calculate the DCT (32*32 matrix)

    c. Reduce the DCT to 8*8 matrix

    d. Calculate the average of all the pixels values

    e. Same as others' next step: compare, produce, calculate Hamming distance

I implement these three methods and compare them in two datasets: output_data_RGB and output_data_HSB, by using OpenCV and PIL packages in Anaconda. Figure 3.7 below is the test result of first airplane image from input dataset class airplane. I also plot a graph presenting the

relationship between two images in order to see their differences more clearly, as shown in Figure3.8. The X-axis of the image refers to the pixel change between 0 and 255 of the image, and the Y-axis refers to the ratio of 0 to 255 pixels in the column. We can obviously see that the changing trend of IMG1 and IMG2 histogram is consistent with the coincidence state.

```
1111111111111111101111111101010000000000010110010100100000000000
1111111111111111101111111101011000000000010110011100100000000000
Average Hash value:  2
1001101110010110101110001100110111110010000011100110100001010011
1011101110110111011110111001101110100100000111001101000110100 11
Difference Hash value:  7
1001001110000100001000000000000000000000000111000100000000 1000000
1001001110000100001000000000000000000000000111000100000000 1000000
Perceptual hash value:  0
The performance value of RGB method:  3.0
```

Figure 3.7: Test run image airplane 1

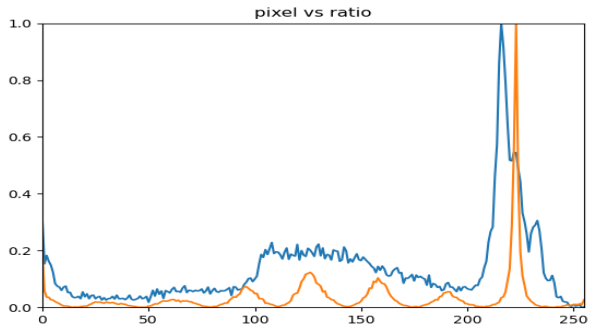

Figure 3.8

```
Compare Airplane Output Class with airplane original images. Similarity:
In aHash method:  24.988950276243095
In dHash method:  27.99171270718232
In pHash method:  19.33839779005525
The performance of RGB method:  24.106353591160225
```

Figure 3.9: RGB performance

```
Compare Airplane Output Class with airplane original images. Similarity:
In aHash method:  5.466850828729282
In dHash method:  12.802486187845304
In pHash method:  2.3729281767955803
The performance of HSB method:  6.8807550644567215
```

Figure 3.10: HSB performance

It is very clear that K-means in HSB color space is the one which has better accuracy and performance by comparing the Figure 3.9 and Figure 3.10. The smaller the hash value is, the less difference between original and processed image. Therefore, I choose HSB dataset (output_data_HSB) as the training dataset to move on to the Machine Learning part.

4. Res-Net Image Classification Model (Realized in python)

Each machine learning model needs a training dataset when we set up to train. My training dataset is the images after processed by color quantization of K-means. In order to select a

dataset with best accuracy, I select the dataset that processed by K-means in RGB which has better accuracy and performance.

I select to use Res-Net18 Image Classification pre-trained model.

Definition 4.1

Res-Net18: ResNet (Residual Neural Network) was proposed by Microsoft Research and successfully trained 152 layers of Neural Network by using ResNet Unit.ResNet-18 is a convolutional neural network that is 18 layers deep.

I will compare the performance of K-means algorithm in two ways. The first is to compare the accuracy of using the processed dataset (output_data_RGB) as the training dataset and of using the original dataset as the training dataset. The second is to compare the training time and prediction time of using the processed dataset and original dataset.

First, here are the results of training accuracy, test accuracy and loss.

```
Epoch  0/15
Loss is :0.0267, Train Accuracy is:0.4310, Test Accuracy is:0.7374
Epoch  1/15
Loss is :0.0106, Train Accuracy is:0.8087, Test Accuracy is:0.8562
Epoch  2/15
Loss is :0.0070, Train Accuracy is:0.8798, Test Accuracy is:0.8945
Epoch  3/15
Loss is :0.0055, Train Accuracy is:0.9039, Test Accuracy is:0.9217
Epoch  4/15
Loss is :0.0047, Train Accuracy is:0.9163, Test Accuracy is:0.9241
Epoch  5/15
Loss is :0.0042, Train Accuracy is:0.9211, Test Accuracy is:0.9333
Epoch  6/15
Loss is :0.0038, Train Accuracy is:0.9308, Test Accuracy is:0.9357
Epoch  7/15
Loss is :0.0032, Train Accuracy is:0.9407, Test Accuracy is:0.9386
Epoch  8/15
Loss is :0.0033, Train Accuracy is:0.9351, Test Accuracy is:0.9420
Epoch  9/15
Loss is :0.0031, Train Accuracy is:0.9405, Test Accuracy is:0.9438
Epoch  10/15
Loss is :0.0027, Train Accuracy is:0.9492, Test Accuracy is:0.9455
Epoch  11/15
Loss is :0.0026, Train Accuracy is:0.9515, Test Accuracy is:0.9513
Epoch  12/15
Loss is :0.0026, Train Accuracy is:0.9526, Test Accuracy is:0.9554
Epoch  13/15
Loss is :0.0025, Train Accuracy is:0.9478, Test Accuracy is:0.9542
Epoch  14/15
Loss is :0.0024, Train Accuracy is:0.9559, Test Accuracy is:0.9513
```

Figure 4.11: New dataset numerical value

```
Out[15]: <matplotlib.legend.Legend at 0x7fd051f32cd0>
```

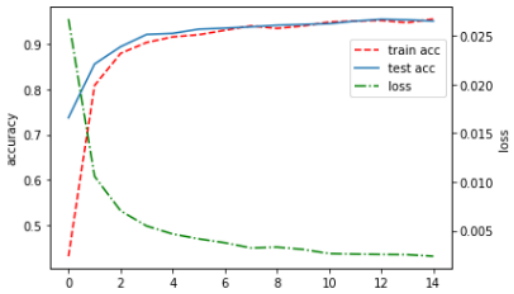

Figure 4.12: New data visualization comparison

```
Epoch  0/15
Loss is :0.0207, Train Accuracy is:0.5853, Test Accuracy is:0.8074
Epoch  1/15
Loss is :0.0086, Train Accuracy is:0.8517, Test Accuracy is:0.8915
Epoch  2/15
Loss is :0.0061, Train Accuracy is:0.8962, Test Accuracy is:0.9118
Epoch  3/15
Loss is :0.0049, Train Accuracy is:0.9155, Test Accuracy is:0.9072
Epoch  4/15
Loss is :0.0041, Train Accuracy is:0.9302, Test Accuracy is:0.9304
Epoch  5/15
Loss is :0.0037, Train Accuracy is:0.9339, Test Accuracy is:0.9437
Epoch  6/15
Loss is :0.0033, Train Accuracy is:0.9372, Test Accuracy is:0.9310
Epoch  7/15
Loss is :0.0029, Train Accuracy is:0.9484, Test Accuracy is:0.9472
Epoch  8/15
Loss is :0.0027, Train Accuracy is:0.9526, Test Accuracy is:0.9571
Epoch  9/15
Loss is :0.0028, Train Accuracy is:0.9468, Test Accuracy is:0.9472
Epoch  10/15
Loss is :0.0024, Train Accuracy is:0.9519, Test Accuracy is:0.9484
Epoch  11/15
Loss is :0.0025, Train Accuracy is:0.9532, Test Accuracy is:0.9559
Epoch  12/15
Loss is :0.0023, Train Accuracy is:0.9549, Test Accuracy is:0.9542
Epoch  13/15
Loss is :0.0022, Train Accuracy is:0.9561, Test Accuracy is:0.9600
Epoch  14/15
Loss is :0.0022, Train Accuracy is:0.9546, Test Accuracy is:0.9611
```

Figure 4.13:Original dataset numerical value

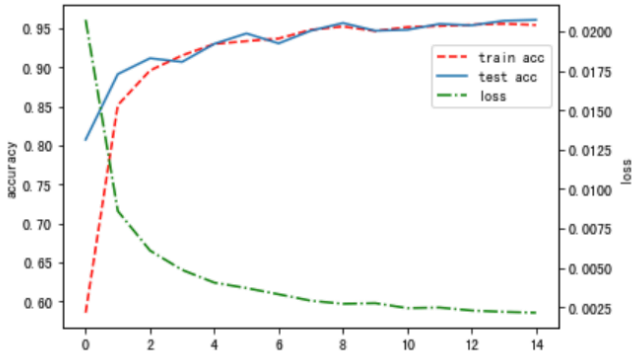

Figure 4.14: Original data visualization comparison

Figure 4.11 shows the numerical value of 15 times test accuracy, training accuracy and loss of processed dataset. To visualize the results, I instead plot them in the same graph, as shown in Figure 4.12. Figure 4.13 and 4.14 correspond respectively to numerical value and visualization for original dataset.

We can clearly see that both datasets perform in high performance. The test accuracy and training accuracy are both high and close to each other. The loss is also very low for both.

Secondly, the training time and running time of the processed datasets is obviously shorter than the original dataset. One of the reasons would be the size of the file. Processed images' size is smaller than the original dataset which leads to the short training time. This dataset contains only 7000 images. If one dataset contains infinity images for a machine learning model to train, it would save a significant amount of time.

Therefore, it is confident to say that both datasets are reliable and thus K-means algorithm performs very well in color quantization.

5. Conclusion

In this project, I explore the feasibility of realizing K-means of Color quantization in Processing 4.0.3, and the performance of K-means algorithms in Res-Net18 Image Classification model. The results show that K-means algorithm is one very good method to do color quantization and at the same time without accuracy and performance loss. It is expected that people could apply K-means of Color quantization in more areas where need image compress and etc.

Bibliography

"400 bad request." 400 Bad Request. n.d. https://iopscience.iop.org/article/10.1088/1742-6596/1213/4/042012/pdf.

"Resnet-18 convolutional neural network - MATLAB resnet18." MathWorks - Makers of MATLAB and Simulink - MATLAB & Simulink. n.d. https://www.mathworks.com/help/deeplearning/ref/resnet18.html.

Springer. n.d. https://link.springer.com/content/pdf/10.1007%252F978-3-540-85920-8_91.pdf.

Senyuan (Russell) Liu is a senior from Beijing, China, who is majoring in Financial Mathematics and Statistics at the UC Santa Barbara. He is expected to graduate in 2023. He currently is one of the research scholars in SEEDS program (Student Engagement in Data Science) and in Dynamo Lab(prof.Ambuj Singh). Afterwards, he hopes to pursue a master degree and work in Financial Engineering or Data science area.

Supervisor/Professor email: George Legrady (glegrady@ucsb.edu)