Average Hashing for Perceptual Image Similarity in Mobile Phone Application

Sam Farisa Chaerul Haviana and Dedy Kurniadi
Sultan Agung Islamic University
Jl.Raya Kaligawe Km.4, Semarang, Central Java, +6224-6583584
e-mail: samfch@unissula.ac.id

Abstract

Common problem occurs in almost all mobile devices was duplicated data or files. Such as duplicated images that often happen by events like capturing perceptually similar photos by the user, or images that shared several times in messaging applications chat groups. This common problem can be solved by manually search and remove the duplicated images one by one by the users, but better solutions is by building automated application that search perceptually similar images then provide the results to the users. We study and implement Average Hashing and Hamming distance for perceptual image similarity into application under mobile phone platform to realize the solution for the problem. The results was very promising in speed and accuracy for finding perceptually similar images under limited resources device like mobile phone.

Keywords: perceptual image similarity, average hash, mobile phone

1. Introduction

Digital images is one of the most common file found in mobile phone storage nowadays. Its existence could be consciously kept by the user or by application programs that utilize them without user knowledge. Habits of taking picture and using messaging applications, like WhatsApp, to share images or another multimedia files, could deliver users into storage runs out problem. Another problem is that not all saved images by those process are unique. Some or many images could be the same images or perceptually similar. Such as pictures taken several times in the same situation, or almost similar poses, tend to produce perceptually similar images. This could be solved, for example, by choosing only the best images to save storage spaces. In messaging application like WhatsApp, sometimes some people in a chat group shares the same images for several times, this also could lead the users into storage problem.

Users could manually search and find those perceptually similar images for later move them to another storage or just remove them. But this could be very time consumed action that user has to take to find those images one by one. A better way is that we could build an application that search and compare those perceptually similar images then give the result to the user. This way need particular set of methods that could calculate the images similarity, perceptually, by finding their features or characteristics then measure the distance for each image to determine which images are perceptually similar. This is commonly called by perceptual image search or perceptual image similarity.

To build the character or feature of an image that could recognized by machine, some approach have been studied. Some of them build the images color histogram to build the characteristic, some other build the hash value of the image that represent its characteristic. Many research like [1]–[6], build hash fingerprint so that in some applications like in [3], [6]–[8], and many others use this hash to check the image integrity for security purpose. In perceptual image hash, the approach of hashes application is slightly different than the hash used in security (like MD5 and SHA). In security like cryptography, the resulting hash of the image is sensitive to its content, even change on a bit of image could result a very different hash, but in perceptual image hash, the resulting hash is only slightly different on identical
images. This gives us the chance to calculate the distance of similarity of the images. Table 1 shows the difference between hash for cryptography (MD5) and perceptual image hash.

<table>
<thead>
<tr>
<th>Image</th>
<th>Cryptography Hash (MD5) Value</th>
<th>Perceptual Hash Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat.jpg</td>
<td>0E2A296EAF42C28899 11224531A6DF8</td>
<td>11111001010001011000</td>
</tr>
<tr>
<td>cat-with-dot.jpg</td>
<td>928F22030B9A02DCC1 C207099201B220</td>
<td>000101100100010001011</td>
</tr>
</tbody>
</table>

As you see in Table 1, those identical images (cat.jpg and cat-with-dot.jpg) result in a very different hash value on MD5 hash, but a slightly different hash value on perceptual hash. Value in bold of perceptual hash is the only difference of resulting hash from those two identical images. This shows us that the perceptual hash still maintains the correlation of data source and the hash value so we could calculate the degree of similarity of the image.

Research in perceptual image similarity has become an interesting area at least in the last decade. One of the reasons is to find techniques that can make machines recognize the perceptually similar images just like human do with their perception. In [9], show some important factors in assessment of image similarity based on human and machine perception. One of determinant factor according to [9] is image color. Color become key to judge image perceptually. This means that image character could be built by its color composition. In some Content Based Image Retrieval (CBIR) researches, image character was built by extracting color histogram of the image. Histogram approach and its variation shows success in many applications or systems using it [10]. However the usage of histogram like in [10] is to build the index of image characteristic for later usage in similarity measure. This means that this approach requiring a pre-processing step, which does not quite suitable for application in limited computing capabilities devices like mobile phone.

Another solution approaches have been proposed, one of them is in [11], by building the hash of the image which is a perceptual hash. On the resulting hash, then similarity measurement methods can be used to find the distance of two hashes that represent two images, so it can be determined if two images perceptually different or not [11]. In [11] also studied the benchmark of three different perceptual hash methods which is Discrete Cosine Transform (DCT) based, Marr-Hildreth based, radial variance based, and block mean value based method. The result show the block mean value based method is the best in speed [11]. There is also a categorization of perceptual image hashing like stated in [12]. Four different category of image hashing method briefly described in [12]. First, Statistic-based schemes, used in [1], [2], [7]. This category building hash by calculating statistics in spatial domain like mean, variance, higher moment of image block and histogram. Second, Relation-based schemes, used in [8], [13]. This category building the hash by using some invariant relation of DCT or Discrete Wavelet Transform (DWT) coefficient. Third, Coarse-representation-based schemes, used in [3]–[6], this category use coarse information on whole image, like significant wavelet coefficient distribution and Fourier transform coefficient in low frequency. Fourth, Low level feature-based schemes, used in [14], [15]. This category building hash by detecting prominent feature point of image. This category doing DCT and DWT transformation on the original image the directly use its coefficient to build the hash.

Image hashing method variation continues to grow, like stated in [16], because of its application in security and image similarity in various platform. The main challenge is the difficulties to find method that has good performance in low computing capabilities device and with acceptable result of accuracy. The method we used in the application of this study is similar.
with the one used in [17], but we utilize simpler perceptual hashing method combined with hamming distance function for similarity measurement in order to get lower computing cost in mobile device.

2. Research Method

The method used in this study is one of the simplest method to build the hash of an image. We have tried to implement the simplest steps of process in order to get minimal computing cost while still maintaining the correlation of the hash to the image. The main idea of this method is to get the average value of pixel color of processed image then utilize it to build the hash value. Unlike the block mean value based method, this method we use does not divide the image into blocks to build the hash, but use the entire pixel value to build the hash. This study implement average hashing in four main steps:

1. Resize image dimension
This step is reducing image dimension into 8x8 pixel from any dimension without considering image aspect ratio. This step is intended to remove high frequency of image color and detail. The result of this step is 64 pixels image which represent the color composition and distribution of the original image. By reducing dimension, the cost to process the next step will also be reduced.

2. Reducing image colors or Grayscaling
This step is intended to reduce the Red, Green, Blue (RGB) color value of the image into a single greyscale color value of each pixel. The result is 64 pixel image with 0 to 255 value in each pixel.

3. Calculate average color
We could calculate the average color of all pixels by dividing the sum of all pixel value resulting from the last step with 64. The order to read and sum the value does not matter. The average value later used to build the hash in the next step.

4. Build hash
The hash was built by comparing the value of each pixel of the image resulting from the grayscaling step with the image average color value. The value greater than or equal to the average value will be represented as 1 in the hash. The smaller value than the average will be represented as 0 in the hash. So that for 64 pixel of image, we get 64 bit of binary hash value. This binary representation later could be represented as hexadecimal or decimal value depending on its application. In this study we keep using the binary value of the hash.

Next process in to calculate the similarity of the images using the hash by comparing them using similarity measurement method. We utilize Hamming Distance as this method works good in binary data. In the end, we represent the distance into percentage of image similarity. The application’s process flow we build in this study shown in Figure 1.

![Figure 1. Flowchart of Perceptual Image Search Application](image)
We use linear search to find all the images in the specified path than build the hash and
calculate the distance in runtime. We did not utilize database engine or file storage to save the
hashes as those will need additional storage space in the device. This does not fit with our goal
to save more space by finding perceptually similar images. The other reason why we don’t
utilize database or file storage for the hashes is that we don’t make a preprocessing step to
save hashes of the images as the image files in the device is dynamically changed even in
small time interval, for example like the image files used by messaging applications. This could
be time and resources consuming thing if we do a preprocessing step to save and index the
hashes.

In the application we built for this study, we use an image as the query image. This
query image will be compared to all images found in the specified path. Then using a specified
threshold value to get the most similar images. All images that have similarity percentage above
the threshold are considered as perceptually similar images with query image. The images are
exactly the same if they have the same hash or in other word they have 100% similarity
percentage. In this application we are focusing to find all perceptually similar images. The
process after the images found is part of development of the application, for example to remove
or to move the images found into another storage location will be considered based on the need
of the users.

3. Results and Analysis

The application we built for this study runs under Android mobile phone platform with
specification of 2Gb of RAMs, Cortex-A7 Quad-core 1.6 GHz CPU, 13 MP primary back camera
and 6.0.1 Marshmallow Android OS. This specification was common specification for mobile
phone released the last few years. We have tested the application using selectively chosen
images we get from http://www.gettyimages.com and http://www.kapanlagi.com. For the test we
categorize the images based on five category, that is Person, People, Building, Landscape and
Objects. This category was selected based on the category that commonly saved or taken by
the users using their mobile phone camera. Figure 2 shows the whole images used in this test.

Due to the limited availability of images in every category that perceptually similar in
source sites, the number of images we could get for every category are different between 6 to
11 images. The total image we could collect is 42 images for all category. The image sizes are
varies between 42 Kb to 246 Kb. The dimension of the images also varies. In addition to the
main image sets, we also test the application using previously found images in DCIM folder and
WhatsApp Media folder in the device we use. This two other category was intended to test the
application for more numbers and sizes of images. DCIM and WhatsApp Media folder are
commonly used by camera application and WhatsApp messaging application to save the images in real usage of mobile phone. The way we test is by choosing a random query image for every image category then run the application search function. The results are recorded and analyzed.

What we want to achieve from the test is the ability of implemented methods we built to find perceptually similar image in acceptable running time under limited resource. We also analyze the method fitness for all image categories, so as the most suitable threshold value for the method. Last, we observing the application running behavior to make sure that the implemented method is running well. Figure 3 show screenshot of application running under testing device.

![Application screenshot under testing device](image)

Figure 3. Application screenshot under testing device

The result of the test discussed in the following discussion.

Table 2 show the results obtained from main image set test. Percentage of Accuracy value on Table 2 was calculated using:

\[
\text{Accuracy} = \frac{\text{number of images found} - \text{error images}}{\text{number of images in category}} \times 100\%
\]

*number of images found* was the images that successfully displayed from the search results which have similarity percentage above the specified threshold value. *Error images* are the images that displayed in the search results but does not came from the same category with query image. *Number of images in category* is the number of images in category where the query image belong.

<table>
<thead>
<tr>
<th>Image category</th>
<th>Total image in category</th>
<th>Accuracy within Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>95</td>
</tr>
<tr>
<td>Person</td>
<td>10</td>
<td>80%</td>
</tr>
<tr>
<td>People</td>
<td>9</td>
<td>11%</td>
</tr>
<tr>
<td>Landscape</td>
<td>11</td>
<td>18%</td>
</tr>
<tr>
<td>Building</td>
<td>6</td>
<td>17%</td>
</tr>
<tr>
<td>Objects</td>
<td>6</td>
<td>17%</td>
</tr>
</tbody>
</table>
From Table 2 we found that Person category has the highest accuracy in almost all threshold value. This is because the images in Person category have the simplest color composition and with high color contrast. From this we obtain that Average Hashing is suitable for the type of image that has simple color composition. This is also confirmed by the result in Objects category that show the lowest accuracy of all, where object category has the most complex color composition. The threshold value, from the result in Table 2, we found that value around 75 to 80 percent was the best value in this study. This threshold value show more accurate results.

Another test was to test running time of application in more image set. All three image sets were tested, and the results shown in Table 3.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Average running time (seconds)</th>
<th>Number of Images</th>
<th>Average Size of Images (kb)</th>
<th>Average processing time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main Image Test Files</td>
<td>1.616</td>
<td>42</td>
<td>145</td>
<td>0.0385</td>
</tr>
<tr>
<td>DCIM Camera Images</td>
<td>38.2432</td>
<td>44</td>
<td>2.656</td>
<td>0.8692</td>
</tr>
<tr>
<td>WhatsApp Images</td>
<td>34.5788</td>
<td>555</td>
<td>132</td>
<td>0.0623</td>
</tr>
</tbody>
</table>

From the results in Table 3 we obtain that the method we implement is fast enough running under the test device. The test against relatively big image size, above 2Mb, only need less than one second in average to process. On smaller images, show even more promising processing time, only less than 0.1 second in average. This time was achieved by dividing average running time with the number of files. Of course this running time was directly proportional to the number and image size.

4. Conclusion

The method we implement was working good and quite promising. Show really good results on images with simple color composition, this is shown in Table 2. But this method also show less accurate result on images that have complex color composition, this means images with complex color content was relatively difficult to perceptually compared using our method, this is also shown in Table 2. The best threshold value was around 75 to 80 percent similarity according to the results also on Table 2. More threshold value resulting less accuracy. In term of running time, the implementation of our method was very promising, like shown in Table 3. The application running under limited resources device runs very well and at acceptable running time even can be considered that this implementation is running fast.

References


