VOL. 3, NO. 4, April 2012 ISSN 2079-8407
Journal of Emerging Trends in Computing and Information Sciences

http://www.cisjournal.org

# An Enhancement on Content-Based Image Retrieval using Color and Texture Features

[1] Tamer Mehyar, [2] Jalal Omer Atoum

[1] Department of Computer Science, King Hussien Scholl of Information Technology
[2] Professor, Department of Computer Science, King Hussien Scholl of Information Technology

[1] tamer_mehyar@hotmail.com,[2] atoum@psut.edu.jo

## ABSTRACT

As the digital technology advances, especially the data storage and image capturing technologies, more digital images are being created and stored digitally. This led to the creation of large numbers of digital image libraries. Hence, the need for intuitive and effective image storage, indexing, classification and retrieval mechanisms rises. In this paper, an enhancement on the use of color and texture visual features in Content-Based Image Retrieval (CBIR) is proposed by adding a new color feature called Average Color Dominance which tries to enhance color description using the dominant colors of an image. The proposed methodology was compared with the work of Kavitha et al [1] and has shown an increase in the average precision from 40.4% to 45.06%.

**Keywords:** *Image retrieval, CBIR, Color and Texture Features, Gray-Level Co-occurrences.*

## 1. INTRODUCTION

Digital image storage is replacing most alternative image storage methods. This led to the creation of large number of huge digital libraries or digital image databases which are also assisted by the advances in communications technology. Image retrieval is one of the most important services or functions that all digital image systems must support. To do this, two main approaches had been developed: textual image retrieval and content-based image retrieval.

Textual image retrieval depends on attaching textual description, captioning or metadata with every image stored digitally. Then traditional database queries are used to retrieve images containing the query keywords in their metadata. However, in order to be able to use the image metadata, all digital images must be annotated with the metadata. Manual annotation requires people to have a look at each image and annotate it accordingly. However, this process is very time-consuming, laborious and expensive [10].

The other method for image retrieval is the content-based image retrieval (CBIR). CBIR is the process of retrieving images from a database or library of digital images according to the visual content of the images. In other words, it is the retrieving of images that have similar content of colors, textures or shapes.

The earliest use of the term CBIR in the literature seems to have been used by the authors in [5] to describe their experiments of automatic retrieval of images from a database by color and shape features. This term has since been widely used to describe the process of retrieving desired images from a large collection of images on the basis of visual features (such as color, texture and shape) that can be automatically extracted from the images themselves.

Textual image search is not CBIR, it is the retrieval of images by image content. The image content is a set of visual features that describe an image. Extracting an image's visual features is not an easy process and before we extract visual features, we must have some knowledge about what kind of features we need. Several CBIR research papers [5, and 9] had agreed that image features are basically color, texture and shape features ordered in their specificity. Color is the least specific feature in that two colors may have the same Red component value but have completely different colors. Textures features are more specific but shapes/objects features are the most specific since they look for specific shapes or objects in an image.

The extraction of visual features from an image is one of the fundamental operations of CBIR. "Visual features are properties of an image that are extracted using image processing, pattern recognition, and computer vision methods" [4].

A visual feature is a visual descriptor or property which can be a numeric value or a set of values (a matrix or a 3-dimensional vector for example). The features used must not be too specific. Similar images from the same image categories are needed to be retrieved.

Color is by far the most common visual feature used in CBIR, primarily because of the simplicity of extracting color information from images [3]. A digital image in this context is a set of pixels. Each pixel represents a color. Colors can be represented using different color spaces depending on the standards used by the researcher or depending on the application such as Red-Green-Blue (RGB) and Hue-Saturation-Value (HSV).

Texture still doesn't have a standard definition. Different research papers define it differently. Texture is an image or a part of image containing repeating patterns. Figure 1, illustrates an example of the concept of repeating patterns.

Quantifying or measuring texture is also not a simple and direct process. Again, no standard process of

extracting texture values has been set. But researchers have tried to do so using different approaches such as co-occurrence matrices or Gabor filters [1, 2, 4, 6, and 8].

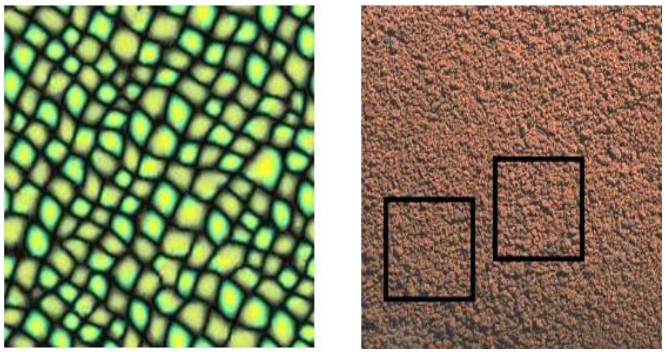


**Fig. 1:** Texture: Repeating patterns

A co-occurrence matrix or Gray-Level Co-occurrences Matrix (GLCM) is a matrix created from information about each pixel and its neighbor pixels *d* blocks away, where d is the distance between the current pixel and the neighboring pixels. Texture features are then extracted from the statistics of this matrix. Four GLCM texture features are commonly used which are given later in this section.

GLCM contains information about the positions of pixels having similar gray level values. A GLCM is a 2-dimensional array, *P*, in which both the rows and the columns represent a set of possible image values. A GLCM element $P_d[i,j]$ is defined by first specifying a displacement vector *d* and counting all pairs of pixels separated by *d* having gray levels *i* and *j*.

Furthermore, GLCM is one of the most known texture analysis methods. In order to estimate the similarity between different gray level co-occurrence matrices, the authors in [8] had proposed 14 different statistical features extracted from GLCM. To reduce the computational complexity, only some of these features are selected in this paper, specifically: Energy, Entropy, Contrast and Inverse Difference.

Energy, as defined in eq. 1, is a measure of textural uniformity of an image. Energy reaches its highest value when gray level distribution has a constant form [1, 6, and 8].

$$\sum_i\sum_j P^2_d(i,j) \qquad (1)$$

Entropy, as defined in eq. 2, measures the disorder of an image and it achieves its largest value when all elements in *P* matrix are equal. When the image is not texturally uniform many GLCM elements have very small values, which imply that entropy is very large [1, 6, and 8].

$$\sum_i\sum_j P_d(i,j) \log P_d(i,j) \qquad (2)$$

Contrast, as defined in eq. 3, is the difference moment of *P* matrix and it measures the amount of local variations in an image [6].

$$\sum_i\sum_j (i-j)^2 P_d(i,j) \qquad (3)$$

Inverse Difference, as defined in eq. 4, measures the image homogeneity. This parameter achieves its largest value when most of the occurrences in GLCM are concentrated near the main diagonal [1, 6, and 8].

$$\sum_i\sum_j (P_d(i,j))/ (ABS(i-j)^2),\ i \neq j \qquad (4)$$

Using these numerical values, texture could be quantified and used in various statistical or algorithmic ways. One of the most important advantages of quantifying texture is that textures of images can be compared against each other much easier.

Section two of this paper presents the related work, section three presents our proposed methodology that improves on the existing CBIR mechanisms and algorithms, section four presents the experimentation results, section five presents the conclusion and finally section six presents the future work of this paper.

## 2. RELATED WORK

The GLCM for texture features had been used in the context of Rock categorization [6]. Factories using rocks must choose good rocks with certain standard qualities. However, this usage lacks the applicability to give it the ability to be used in other applications or contexts where the wide range of images should be somehow similar or have similar image structures.

An efficient image retrieval technique which uses local color and texture features had been presented in [1]. Images were partitioned into sub-blocks of equal sizes as a first step. Colors of each sub-block were extracted by quantifying the HSV color space into non-equal intervals. Texture of each sub-block was obtained using the gray-level co-occurrence matrix (GLCM). A one to one matching scheme was used to compare the query and the target image. Euclidean distance was used in retrieving similar images. This algorithm lacks the generality in which similar images could be retrieved.

Another algorithm of CBIR in which color, texture and shape features are used as the visual features had been presented in [7]. This algorithm had used a novel framework for combining all the three features i.e. color, texture and shape information. It had achieved higher retrieval efficiency using the image and its complement. The image and its complement are partitioned into non-overlapping tiles of equal size. The features drawn from conditional co-occurrence histograms between the image tiles and corresponding complement tiles, in RGB color space, serve as local descriptors of color and texture. This local information is captured for two resolutions and two grid layouts that provide different details of the same

image. Also, this algorithm lacks the generality in which similar images could be retrieved.

A different approach of CBIR in which only texture features are used had been presented in [2]. The textures features used were extracted using Gabor filter (wavelet). Gabor wavelet is a widely adopted methodology for extracting texture features. It had been proved to be very useful too. Texture features are found by calculating the mean and the variation of the Gabor filtered image. Since Gabor filter (wavelet) provides a powerful way of extracting texture features, this algorithm had provided good results, but in certain categories. Although texture features extracted using Gabor filter (wavelet) was a success, it still is too generic as a CBIR visual feature. Using texture features only is not enough for large database of images of different categories. It may suffice for certain categories but not all.

## 3. PROPOSED METHODOLOGY

As mentioned earlier, the main visual features to recognize and identify images are colors, textures and shapes. Colors and textures are specific enough to identify images with similar features but also not too specific to retrieve exact or very similar images. Retrieving exact or very similar images is not very applicable in this application since different images from different places with different objects could belong to the same category. For example, Figure 2 shows two images that belong to the same category (i.e. beaches). If objects or shapes features are used, they would not be counted as similar since they do not have any similarity in shapes and in objects. The main similarity between them is the blue color and the ocean texture in addition to the top sky texture.

**Fig. 2:** Two images from the same category have similar color and texture features but different shape features

Images of the same category should have very similar colors and similar textures (such as clouds, skies, grass, desert, monuments textures). Therefore, in this paper colors and textures features will be used and not shape features.

### 3.1 IMAGE PRATITIONING

It has been proved that to obtain better results, each image is divided into six blocks, more accurately two rows and three columns with equal sizes [1]. The color and texture values are calculated for each block independently from other blocks and stored separately for each block. All Images are resized to a standard size of 256x384 so that each block's size is 128x128. Therefore, in this paper, images are resized to 160x240 to obtain better results. Image partitioning is demonstrated as shown in Figure 3.

### 3.2 COLOR FEATURES EXTRACTION

There are many different color spaces and standards that have being used, the most common one is the RGB or the CMYK color spaces. However, for this application HSV is perceived to be the best, in addition to a proposed color feature named Average Color Dominance (ACD) value to be explained later [1].


**Fig. 3:** Six 80x80 sub-blocks

### 3.3 HSV COLOR FEATURE

For efficient processing and reducing the range of values for computation, the HSV values are quantized to

non-equal bins (8x3x3 bins) as done by the authors in [10] i.e. Hue: 8 bins; Saturation: 3 bins; Value: 3 bins as defined below:

$$H=\begin{cases}0 \text{ if } h\in[316,20]\\ 1 \text{ if } h\in[21,40]\\ 2 \text{ if } h\in[41,75]\\ 3 \text{ if } h\in[76,155]\\ 4 \text{ if } h\in[156,190]\\ 5 \text{ if } h\in[191,270]\\ 6 \text{ if } h\in[271,295]\\ 7 \text{ if } h\in[296,315]\end{cases} \quad S=\begin{cases}0 \text{ if } s\in[0,0.2)\\ 1 \text{ if } s\in[0.2,0.7)\\ 2 \text{ if } s\in[0.7,1)\end{cases} \quad V=\begin{cases}0 \text{ if } v\in[0,0.2)\\ 1 \text{ if } v\in[0.2,0.7)\\ 2 \text{ if } v\in[0.7,1)\end{cases} \tag{5}$$

In accordance with the quantization level above in equation 5, 3-dimensional feature vectors for different values of HSV with different weights are combined to form a 1-dimensional feature vector named G [1].

To combine the three values into one combined color feature vector, the authors in [1] had used the following equation:

$$G = (9 * H) + (3 * S) + V \tag{6}$$

Using this method, the 3-dimensional vector of HSV is quantized into a 1-dimensional vector for the 72 (8x3x3) of main colors. This vector is stored as the HSV value of each image block [1].

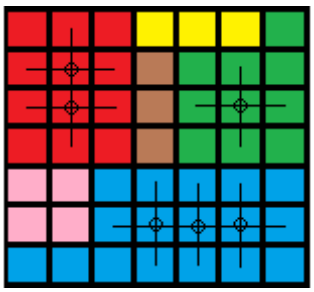
### 3.4 ACD COLOR FEATURE

It has been noticed that the retrieved images had quite similar textures but sometimes different colors. Some of the top images had more than one block with very different colors. It seems that the HSV color feature is weak or the algorithm is lacking color features. The authors in [3] had discussed image retrieval and indexing methods. They had presented color discretization or quantization in which the color palette of the image is reduced to a specified number of colors. Although, eleven colors palette provided an image with a reduced color quality but less variations in color, so it was easier to extract a dominant color from the image.

A dominant color can be described as the most used color (i.e. the color with most pixels exhibit). However, in this case many pixels could be counted when they are actually not connected to any region. For example, a red pixel surrounded by blue pixels would not be really perceived by the human eye. So counting this red pixel would not be logical and would result in an inaccurate output. To solve this issue, only pixels that have the same color as its 4-neighbor pixels were counted. This would reduce the chance of counting invalid pixels that could be noise.

The first step to calculate the ACD vector is to quantize the image into eleven colors or reduce the image's color palette to eleven colors palette as discussed by the authors in [3]. In this paper, the Aforge's Median Cut quantizer will be used to quantize the images.

The next step is the color counting process. For each pixels that has a color similar to its 4-neighbors colors, the value of this color key in a separate results array *A* is incremented by 1, where *A* is a 2-dimensional array of 11 rows (number of colors) and two columns (the color value and its frequency). But as mentioned previously, a pixel's colors is only counted when it has the same color as all of its 4-neighbors colors as demonstrated in Figure 4.a. When this process is done, the outcome is an array of the colors used in the images and their counts as shown in Figure 4.b.



| | |
|---|---|
| Red | 2 |
| Yellow | 0 |
| Green | 1 |
| Brown | 0 |
| Pink | 0 |
| Blue | 3 |

**Fig. 4:** a) A sample image. b) its 2-dimentional array of colors

From this array *A*, we know the colors that dominate the image. Here, another issue rises: if two images have the same dominant colors, would they be similar? The answer is No. However, when the HSV color features and the texture features are included, the outcome is improved. While experimenting, and proved later in the experimentation results, it was found that it is much better to take the average of the top three most used colors rather than the first one only since usually more than one color may dominate an image or an image block. In Figure 4.b, the top three colors are blue, red and green. The average of the top three colors is stored in the database as a color feature along with the HSV color feature.

Each color is represented as a 3-dimensional vector i.e. one Red value, one Green value and one Blue value, which is what's known as the RGB color space. So to be able to take the averages of the top three colors, the average value of R, G and B is taken and is considered as a color value. Usually this action is not logical since different colors may result with a similar average, however, in this case it is more reasonable since the image is first quantized into eleven colors. The eleven colors are usually not very similar to one another so would have different RGB values.

The ACD extraction algorithm as illustrated in Figure 5, starts with preprocessing the image by first resizing it into a preset size of 160x240. Then the image is partitioned into 6 blocks. Each block will be quantizes by the median cut quantizer. The quantized image will then get its colors that have similar 4-neighbor pixels colors counted and the average of the top 3 colors are taken to end up with the average color dominance (ACD) value.

We now have two color feature values: HSV and ACD value, which will be stored in the database for each block of each image.

*1-Resize image into 160x240*

*2-Partition image into 6 blocks*

*3-Loop: for each image block Do:*

*3.1-Quantize block using Aforge's Median Cut Quantizer into eleven colors*

*3.2-Count the number of times each color exists with a color similar to its 4-neighbor*

*pixel colors*

*3.3-Quantize the 3-dimensional RGB vectors into 1-dimensional vectors equal to the*

*average of the R,G and B components = (R + G + B) / 3*

*3.4-Calculate the average of the top 3 most counted colors (RGB average values)*

*3.5-Store this RGB average value (ACD value) into the database for this image block*

*4- End Loop*

**Fig. 5:** ACD Extraction Algorithm

### 3.5 TEXTURE FEATURE EXTRACTION

Texture is not a simple feature and still is not properly defined. So proper texture features have not been defined, however, different methods had been tried to gain information about the texture of an image using different aspects. In this paper, Gray-Level Co-occurrence Matrix (GLCM) is created to supply some texture information. From this GLCM, certain numeric values can be extracted to supply different types of information about the image texture.

GLCM is a matrix created from the information about each pixel and its neighbors *d* blocks away. Texture features are then extracted from the statistics of this matrix. Four GLCM texture features are commonly used which are given later in this section.

The GLCM matrix *Pd* has a dimension of *n x n*, where *n* is the number of gray levels in the image.

A sample image that is made up of only three gray-level values is showed in Figure 6.a. Figure 6.b, shows the relationship that will be used to extract the Pd matrix. Figure 6.c, shows the number of times each two color combinations occurred in the image with a relationship as the spatial separation shown in Figure 6.b. This process is further explained in the algorithm presented in Figure 7.

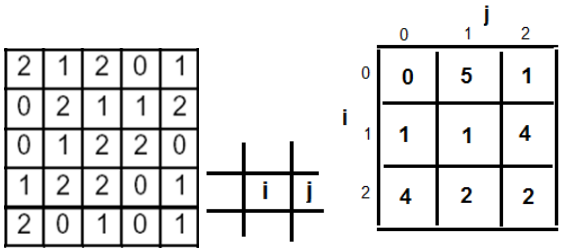



**Fig. 6:** a) Sample Image's gray-levels. b) Spatial separation between pixels i and j. c) Sample Co-occurrence Matrix generated from a using b relationship.

Let's take an example by showing how the value 4 in row 2 and column 0 was obtained in Figure 6.c. This is done by finding out the number of times 2 (value of the row position) was found on the left of 0 (value of the column position) as the spatial separation positioning shows. It can be seen in the sample image that at positions (0,2), (2,3), (3,2) and (4,0) the value 2 is found right next to a 0 on its right. The number of times this happens in this sample image is 4.

Normalizing the GLCM makes the values lie between 0 and 1 and allows them to be represented as probabilities which help in statistics and further texture calculations and analysis.

To normalize the GLCM matrix in Figure 6.c, the number of different colors that the GLCM matrix must used is 20. So each value in the GLCM matrix is divided by 20. So the value at (0,0) becomes 0 (0 divided by 20), the value at (0,1) becomes 0.25 (5 divided by 20) and so on.

We now have a 2-dimensional matrix containing the probabilities of gray-level co-occurrences of an image. But such a matrix is not very useful and it is very difficult to store and utilize in image similarity. Therefore, numeric features are extracted from this matrix to provide different texture meanings or measures. The numerical measures used in this study are described as follows [6]:

Contrast:

- $\sum_i\sum_j (i-j)^2 P_d(i,j)$

- Energy:
  - $\sum_i\sum_j P^2_d(i,j)$
- Entropy:
  - $\sum_i\sum_j P_d(i,j) \log P_d(i,j)$
- Inverse Difference
  - $\sum_i\sum_j (P_d(i,j))/ (ABS(i - j)^2), i \neq j$

The above four independent numerical texture features are extracted for each block from the extracted GLCM and stored as the texture features.

*1- Quantize the original image's gray-level Histogram to a known number of bins e.g. 3 bins*

*2- Create a 3x3 matrix: M(i,j)*

*3- Loop: for each i, j pair in the matrix M(i,j):*

*3.1- Count the number of pairs in the original image according to the relationship shown in figure 7 b).*

*3.2- Store the pairs count into the GLCM at position (i, j)*

*4- End Loop*

*5- Normalize the GLCM M(i,j) by dividing all its values by the number of pixel pair (in figure 7: 20)*

**Fig. 7:** GLCM Generation Algorithm

### 3.6 IMAGE RETRIEVAL

After populating the database, all images should have their color and texture features for each of their blocks stored in the database. The actual application of this study is the image retrieval that is based on content similarity. Similarity in this context is represented by the color and texture features of each image. So we now have to retrieve images from the database sorted by how close they are to a source image.

To do this we have to be able to compare the data extracted from each image. Each image has information for its 6 blocks which must be compared. Corresponding blocks for the source image and a stored image are compared against each other i.e. the first block of the stored image is compared with the first block of the source image, the second block of the stored image is compared with the second block of the source image and so on. This is illustrated in Figure 8.

z

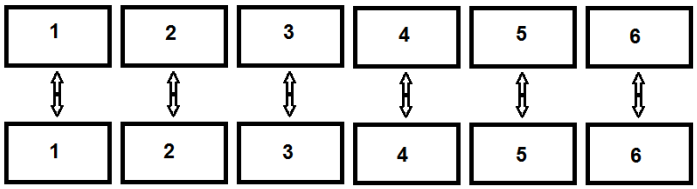



**Fig. 8:** Comparisons of Corresponding blocks

### 3.7 IMAGE MATCHING

As it has been mentioned earlier, matching is done using a 1-1 relationship between corresponding image blocks. Matching is done by finding the Euclidean distance for each feature (color or texture) as shown in Figure 9.

Similarity to the source image is measured by how closed the visual features of each image are to the source image using Euclidean distance. For each image block six lists with all images in the database are created for the six visual features (HSV, ACD, Energy, Entropy, Contrast and Inverse Difference). Each list will be ordered by how close a visual feature is to the source image. Considering the HSV list, the most similar image would have an index of 0 in the list while the least similar would have an index of 999. The same applies for all the other features.

In order to combine the six lists into one list that is ordered by the images' similarity, the sum of the six indices of those six lists are used to correspond to this similarity. Images which are actually similar will have relatively small indices while images that are not so similar will have large indicies. Therefore, a new list is created that sorts all images by the sum of indices of the six features' lists.

For example, suppose that block#1 of image#34 exists at position15 in list1, at position 55 in list2, at position 60 in list3, at position 19 in list4, at position 150 in list5 and at position 345 in list6. Taking the sum of indices for block#1 of image#34 gives a value of 15 + 55 + 60 + 19 +150 +345 = 644. This value is calculated for each block of each image ending up with an ordered list of how close block#1 of each image is to block#1 of the source image.

At this point we have an ordered list for each block of each image showing how similar images are to the source image. The next step is to get a proper final ordering of all images. This is done by taking the sum of indices of each image from the six list of each block. Ordering images by their sum of indices would result in one list of 1000 images ordered by similarity to the source image.

Finally, we have one list ordered by the total similarity of all images to the source image. From this list we can take the top 20 images, for example, as the result of the content-based image retrieval for the provided source image.

1-*Provide source image*

*2-Preprocess source image (resizing image into 160x240)*

*3-Partition image into 6 blocks*

*4-Loop: For each block Do*

*4.1-Compute HSV, ACD, and GLCM texture values (orig_HSV, orig_ACD, orig_GLCM_Contrast, orig_GLCM_Energy, orig_GLCM_Entropy, orig_GLCM_InvDiff)*

*5-End Loop*

*6-Loop: For each image in the database Do*

*6.1-Loop: For each block b in current image Do*

*6.1.1-Compute Euclidean distance*

*diff_HSVb# = current_HSVb# – orig_HSVb#,*

*diff_ ACDb# = current_ ACDb# – orig_ACDb#,*

*etc...*

*6.1.2-Create six lists of all database images ordered by the Euclidean distance computed for each of the HSV, ACD, Energy, Entropy, Contrast and Inverse Difference features and for the current block*

*6.1.3- Order each of these lists by their list values i.e. diff_HSV, diff_ACD etc...*

*6.2-End Loop*

*7-End Loop*

*8-Create six new lists with all images (lstImageBlock#)*

*9-Loop: For each image i Do:*

*9.1-Loop: For each image block b Do:*

*9.1.1- lstImageBlockb#[i].value = List(dif_HSVb#)[i].Index + List(diff_ACDb#)[i].Index + List(diff_Contrastb#)[i].Index + etc...*

*-//i.e. image1's index in List(diff_HSV1) is 10 and in List(diff_ACD1) is 33 and in List(diff_Contrast1) is 15 etc.. So.lstImageBlock1[image1].value = 10 + 33 + 15 + ..... (for block 1)*

*9.2-Order lstImageBlock#b by its values (the sum of indices)*

*9.3-End Loop*

*10-End Loop*

*11-Create a new list of all images (lstImages)*

*12-Loop: For each image i Do:*

*12.1- lstImages[i].value = lstImageBlock1[i].Index + lstImageBlock2[i].Index + etc..*

*13-End Loop*

*14-Order lstImages by its values (the sum of indices)*

**Fig. 9:** Image Matching Algorithm

## 4. EXPERIMENTATIONS AND RESULTS

Various CBIR methodologies had been proposed. The *Kavitha* methodology proposed in [1] is based on CBIR methodology that uses the HSV color space as the color feature and the GLCM's Energy, Entropy, Contrast and Inverse Difference as the texture features. Therefore in this section, our proposed methodology will be compared with the *Kavitha* methodology. The comparison will be carried out by comparing the precisions and recall of both methodologies.

In order to do so, the *Kavitha* methodology was implemented in the same environment as the proposed methodology on the same benchmark image database. Table 1 and Figure 10 show the results (precision and recall) of this experiment.

We can notice from Table 1 and Figure 10, that the average precision in the proposed methodology is higher than the *Kavitha* methodology. This means that the proposed methodology resulted in better output and accuracy. However, we can notice that not all categories were improved in the proposed methodology. Some categories' precision and recall results remained almost the same and some others had lower precisions and recalls values. This is due to the fact that different categories have different color properties i.e. some categories may have a very clear dominant color or a set of colors while some other categories may not. In other words, some categories may have most of its 11 colors as dominant i.e. most of the colors are evenly distributed around the image or image block.

**Table 1:** Precision and Recall results

| **Image Category** | **HSV + GLCM *Kavitha* methodology** | | **HSV + GLCM + ACD [proposed methodology]** | |
|---|---|---|---|---|
| | **Precision** | **Recall** | **Precision** | **Recall** |
| Africa | 29.1% | 5.82% | 31.0% | 6.2% |
| Beaches | 27.9% | 5.58% | 37.0% | 7.4% |
| Monuments | 21.5% | 4.3% | 23.7% | 4.74% |
| Buses | 36.9% | 7.38% | 31.4% | 6.28% |
| Dinosaurs | 90.55% | 18.11% | 95.2% | 19.04% |
| Elephants | 37.7% | 7.54% | 37.9% | 7.58% |
| Flowers | 58.6% | 11.72% | 70.1% | 14.02% |
| Horses | 59.3% | 11.86% | 66.2% | 13.24% |
| Mountains | 26.9% | 5.38% | 32.4% | 6.48% |
| Foods | 19.9% | 3.98% | 25.7% | 5.14% |
| **Average** | **40.8%** | **8.16%** | **45.06%** | **9.012 %** |

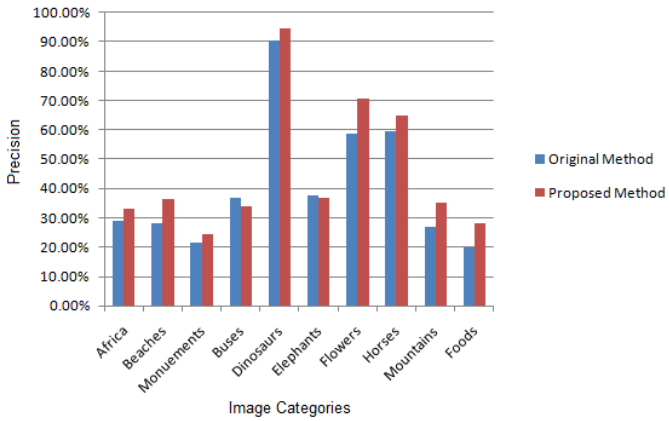



**Fig. 10:** Comparison between the precisions of *Kavitha* Methodology and the proposed methodologies.

Based on the results obtained from the previous experiments done, the following could be concluded:

- The proposed methodology had obtained better precision than the *Kavitha* methodology. The *Kavitha* methodology obtained an average of 40.8% of precision while the proposed methodology obtained an average of 45.7% of precision.
- Calculating the ACD value from the average of the top 3 most dominating colors obtained better results than using only the top color alone
- Calculating the ACD using the value of *d* as 1 (1 pixels away from the current pixel or in other word the 4-neigbor pixels) retrieved a higher precision than using a larger value of *d* such as 3
- Working on images of sizes 160x240 resulted in better results than images with sizes of 384x256. This is due to resizing images into smaller sizes and reducing the amount of noise.

Hence, the proposed methodology has proven to be better than the *Kavitha* methodology. The precision of the proposed methodology is higher than the precision of the *Kavitha* methodology. However, there are some weaknesses in the proposed methodology, mainly the following:

- Images with sizes less than 160x240 would be scaled up to 160x240 which will not allow the same effect of scaling down images to take place. Therefore, images with smaller sizes must be dealt with differently.
- With the increase in the number of output images to 40 and 60 images, the precision has decreased by 18% and 27% respectively. The image matching part is where the core of this application lies. Improvements on the matching algorithm could provide a better overall results.

## 5. CONCLUSION

The purpose of this paper was to improve the accuracy (precision) of a CBIR application by allowing the system to retrieve more images similar to the source image. The proposed methodology adds a new color features to the already implemented HSV color features and GLCM texture features. The new color features is the Average Color Dominance value (ACD) which represents the average of the top dominating colors.

The proposed methodology was tested and experimented on a benchmark database of images. The proposed methodology had increased the average precision from an average of 40.4% (*Kavitha* methodology) to an average of 45.7% for precision. However, there are still some drawbacks in the image matching algorithm that could be improved to provide better results in general and prevent the precision to decrease when increase the number of output images.

## 6. FUTURE WORK

This CBIR application has some drawbacks and weaknesses that could be improved. Some of these drawbacks that could be improved in the future are the following:

- One of the main and core improvements is the Image Matching Algorithm. The system gets all images and sorts them by how close they are to the source image. However, the sorting algorithm (image matching algorithm) could be further improved to obtain better precision.
- It was found better to scale down images from 384x256 to 160x240. However images with sizes smaller than 160x240 would be scaled up and so would not obtain similar results. In addition, very large images would lose lots of visual information when scaled down to 160x240. Therefore it would be better to preprocess images in a certain way so that it will provide the same effect as scaling images down.

## REFERENCES

[1] C. Kavitha, B. Prabhakara, and A. Govardhan CH," Image Retrieval Based on Color and Texture Features of the Image Sub-Blocks". *International Journal of Computer Applications* (0975 – 8887), Vol. 15– No.7, February 2011.

[2] D. Zhang, A. Wong, M. Indrawan, and G. Lu, "Content-based Image Retrieval Using Gabor Texture Features", *IEEE IEEE Transactions PAMI*, pp. 13-15, 2000.

[3] H. Muller, N. Michoux, D. Bandon, and A. Geissbuhler. "A Review of Content-Based Image Retrieval Systems in Medical Applications – Clinical Benefits and Future Directions", *International Journal of Medical Informatics*, 73(1):1–23, Feb 2004.

[4] J. M. Zachary, An Information Theoretic Approach To Content Based Image Retrieval, Louisiana State University and Agricultural and Mechanical College, PhD. Thesis, 45-63, 2000.

[5] K. Hirata, and T. Kato, "Query by visual example – content-based image retrieval", in EDBT'92, *Third International Conference on Extending Database Technology*, pp. 56-71, 1992.

[6] M. Partio, B. Cramariuc, M. Gabbouj, A. Visa, "Rock texture retrieval using gray level co-occurrence matrix", *Proc. of 5th Nordic Signal Procesing Symposium*, On board Hurtigrutenm/S Trollfjord, Norway, October 4-7, 2002.

[7] P. S. Hiremath, Jagadeesh Pujari, "Content Based Image Retrieval based on Color, Texture and Shape features using Image and its complement", *International Journal of Computer Science and Security*, Vol. 1 No. 4, pp 25-35, 2007

[8] R. M. Haralick, K. Shanmugam, and I. Dinstein, "Textural Features for Image Classification", *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-3, No. 6, pp.610-621. November 1973.

[9] T. Kato, "Database Architecture for Content-Based Image Retrieval", in *Image Storage and Retrieval Systems*, (Jambardino, A A and Niblack, W R , eds), Proc SPIE 1662,112-123, 1992.

[10] http://en.wikipedia.org/wiki/Image_retrieval.