Department of Computer Science
Technical University of Cluj-Napoca

Image Processing
Content-Based Image Retrieval

Name: Paul-Adrian Kovacs
Group: 30232
Email: kovacs_paul_adrian@yahoo.com

March 2023

Contents

1 Introduction

1.1 Background

With the increasing availability of digital images, there is a growing need for efficient and accurate methods to search and retrieve images based on their content. Content-Based Image Retrieval (CBIR) is one approach that uses features such as color, texture, and shape to find similar images. However, there are still challenges in developing CBIR systems that can handle large datasets and provide accurate results.

1.2 Problem Statement

The main problem that this project seeks to address is the limited performance of CBIR systems when dealing with large-scale image datasets. Existing CBIR methods may not be able to accurately and efficiently search through a large number of images based on their content due to the high computational cost of feature extraction and matching algorithms, as well as the difficulty of dealing with variations in image quality and noise.

1.3 Significance

Developing an image retrieval system that accurately retrieves images based on their visual features has significant practical applications in fields such as medical imaging, surveillance, and data mining. The proposed system has the potential to save time and improve efficiency by enabling users to easily find similar images in large datasets.

1.4 Objectives

The specific objectives of this project are to (1) develop an image retrieval system that uses color histograms and Pearson correlation coefficient, (2) evaluate the performance of the system on a large dataset of color images, and (3) propose improvements to the system to enhance its accuracy and scalability.

1.5 Relevance

This project is highly relevant to the field of computer science, specifically in the area of image processing and retrieval. The results of this project will be useful to researchers and industry professionals who are working on CBIR systems and related applications.

1.6 Scope

This project will focus specifically on the development and evaluation of an image retrieval system using color histograms and Pearson correlation coefficient. The system will use a set of images, given by the user at initialization, as a database and will allow users to select an input image to find the most similar images in the database. The system will also calculate the histograms on the Hue channel of the HSI color space for the selected images and display them along with their correlation coefficients. The project will not cover other image processing techniques.

2 Bibliographic Study

2.1 CBIR - Content-based image retrieval

Content-based image retrieval is an approach in which similar images are retrieved for a particular query image depending on the image content similarity. One of the challenges of CBIR is to determine the similarity between color images, which may vary in hue, saturation and brightness. A common approach to measure color similarity is to use color histograms, which represent the distribution of colors in an image.

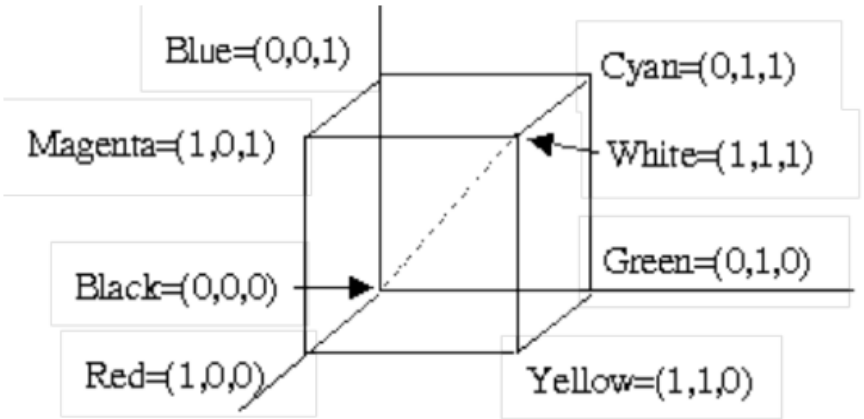
Color is one of the key visual features that can be used to determine the similarity between images. In content-based image retrieval, color similarity between images can be analyzed by examining various aspects of the color distribution in the images, such as color histograms, color correlograms, and color coherence vectors. These features can be compared using various distance measures, such as Euclidean distance, Manhattan distance, or cosine similarity, to determine the similarity between the color images.

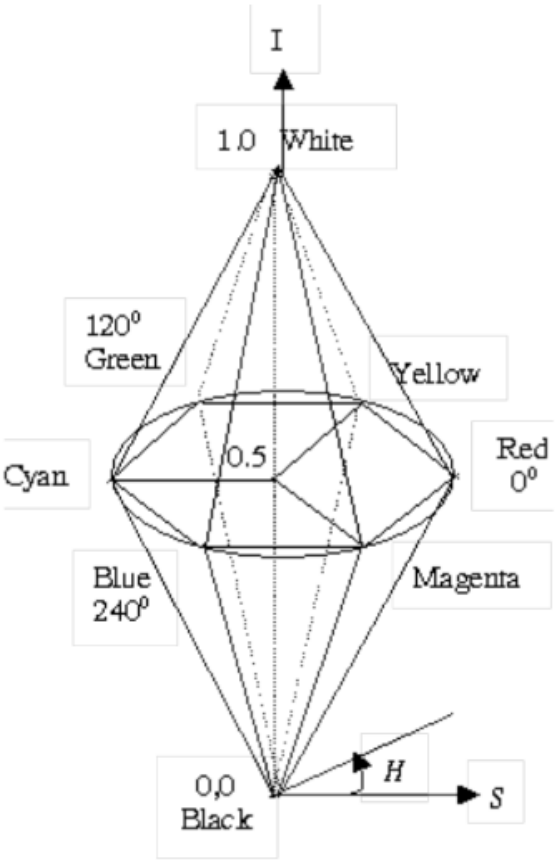
2.2 Color spaces[1]

A color space is a system that describes how colors can be represented numerically. HSI (hue, saturation, and intensity) provides intuitive color representation and manipulation. Therefore it will be beneficial to convert our images from BGR to HSI:

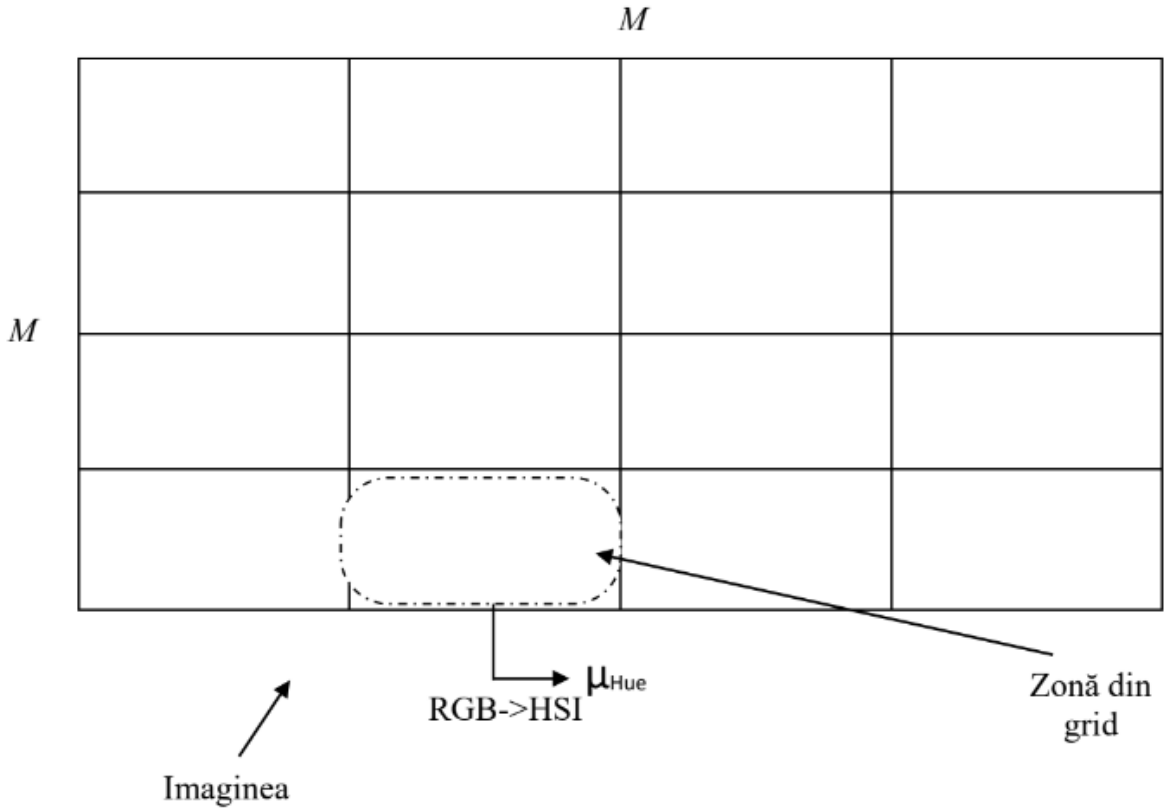
$$I = \frac{R+G+B}{3}; S = \left(I \neq 0 ? 1 - \frac{\min(R,G,B)}{I} : 0\right)$$

$$h = \left(S \neq 0 ? \operatorname{acos} \frac{(R-G)+(R-B)}{2\sqrt{(R-G)^2+(R-B)(G-B)}} : 0\right); H = (B > G ? 360° - h : h)$$





2.3 Determining similarity based on color distribution[2]



The image will be divided into an M*M grid to increase accuracy.

On each grid zone, an RGB -> HSI conversion is performed and the average Hue value ($\mu_{Hue} \in [0, 2\pi)$) is calculated for that zone, ignoring points for which S = 0. This value will be stored in a vector V of size M*M. If all points in a grid zone have S = 0, then the average value is meaningless, and the corresponding element in V will be assigned the value -1, indicating that it should be ignored in subsequent calculations.

The resulting vector V obtained by traversing all zones will be the image representation vector.

If we want to see the similarity between this image and a new image, the corresponding V vector for the new image will be calculated. The degree of differentiation between the two images is given by the formula:

$$D = \frac{1}{C} * \sqrt{\sum_{i=1}^{M*M} (min(|V_1[i] - V_2[i]|, 2\pi - |V_1[i] - V_2[i]|)^2,\ if\ V_1[i] \geq 0\ and\ V_2[i] \geq 0)}$$

where:
V_1, V_2 - the arrays corespoding to the 2 images
$C = \sqrt{N}\pi$ - normalization constant such that D has values in the range [0,1]
V_1, V_2 - number of pairs i which met $V_1[i] \geq 0$

The degree of similarity between the two images is given by the formula:

$$S = 1 - D, if N > 0 or S = 0, if N = 0$$

If a percentage representation is used, S will be multiplied by 100. A similarity threshold T (in percentage) can be chosen:

$$S > T => the\ images\ are\ similar$$

$$S <= T => the\ images\ are\ not\ similar.$$

2.4 Gaussian filter[3]

Gaussian noise removal must be performed using a filter with adequate shape and size, correlated to the amount of the Gaussian noise that corrupts the image. The filter size w of such a filter is usually 6σ (for example, for a Gaussian noise with $\sigma = 0.8 => w = 4.8 \approx 5$ Constructing the elements of such a kernel/Gaussian filter G will be performed using the following equation:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x-x_0)^2+(y-y_0)^2}{2\sigma^2}}$$

where:
X_0, Y_0 - are the coordinates of the central column and row of the kernel
σ - standard deviation

A more efficient method to remove the noise is using 2x1 dimensional gaussian filters since:

$$G(x,y) = G_x(x) * G_y(y)$$

$$G_x(x) = \frac{1}{2\pi\sigma} e^{-\frac{(x-x_0)^2}{2\sigma^2}}$$

$$G_y(y) = \frac{1}{2\pi\sigma} e^{-\frac{(y-y_0)^2}{2\sigma^2}}$$

2.5 Histograms[4]

Many applications use the HSI color model. Machine vision uses HSI color space in identifying the color of different objects. Image processing applications, such as histogram operations, intensity transformations, and convolutions, operate on only an image's intensity. These operations are performed much easier on an image in the HSI color space. For the HSI is modeled with cylindrical coordinates. The hue (H) is represented as the angle, varying from 0o to 360o. Saturation (S) corresponds to the radius, varying from 0 to 1. Intensity (I) varies along the z axis with 0 being black and 1 being white.

2.6 Pearson correlation coefficient[5]

In statistics, the Pearson correlation coefficient is a measure of linear correlation between two sets of data. It is the ratio between the covariance of two variables and the product of their standard deviations; thus, it is essentially a normalized measurement of the covariance, such that the result always has a value between -1 and 1. As with covariance itself, the measure can only reflect a linear correlation of variables, and ignores many other types of relationships or correlations.

$$\rho_{X,Y} = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y}$$

where

- cov is the covariance
- σ_X is the standard deviation of X
- σ_Y is the standard deviation of Y.

The formula for ρ can be expressed in terms of mean and expectation. Since

$$\operatorname{cov}(X,Y) = \mathbb{E}[(X - \mu_X)(Y - \mu_Y)],$$

the formula for ρ can also be written as

$$\rho_{X,Y} = \frac{\mathbb{E}[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y}$$

where

- σ_Y and σ_X are defined as above
- μ_X is the mean of X
- μ_Y is the mean of Y
- $\mathbb{E}$ is the expectation.

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

where

- n is sample size
- x_i, y_i are the individual sample points indexed with *i*
- $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ (the sample mean); and analogously for $\bar{y}$.

Rearranging gives us this formula for r_{xy}:

$$r_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - \left(\sum x_i\right)^2}\ \sqrt{n\sum y_i^2 - \left(\sum y_i\right)^2}}.$$

3 Implementation

1. We compute the Gauss kernel to use it multiple times later
2. We process the images from the database and save the relevant data to allow us multiple searches without requiring redundant reprocessing
2.1. Process and save the vector of predominant hues for every image, converted to hsi, in the database
2.2. For every image, we reduce noise using 5x5 Gaussian filter, then process and save the histogram for the hue channel of every image converted to hsi
3. We do the 2 steps from above for the user selected image
4. Then compare the predominant hues vector to get an initial ranking for image similarity 5. For the 'noValues' most similar images we compute the Pearson correlation coefficient and sort the images based on it to get the final ranking

Listing 1: Main Funtion

```
double sigma = 0.8;
int w = 5; // ~ sigma * 6
pair<vector<double>, vector<double>> gaussKernel = gaussBidimensionalKernel(sigma, w);

int count = 1;

FileGetter fg(folderName, "bmp");
while (fg.getNextAbsFile(fname))
{
   cout << "Processing image " << count++ << '\n';
   Mat src = imread(fname, IMREAD_COLOR);

   Mat hsi = BGRtoHSI(src);
   vector<double> dominantHues = getDominantHuesVector(hsi, gridSize);

   Mat gauss = gaussBidimensionalFilter(src, gaussKernel);
   hsi = BGRtoHSI(gauss);
   vector<double> hist = histogram(hsi);

   imagedb.emplace_back(image(src, fg.getFoundFileName(), dominantHues, hist));
}

while (openFileDlg(fname))
{
   Mat src = imread(fname, IMREAD_COLOR);

   Mat hsi = BGRtoHSI(src);
   vector<double> dominantHues = getDominantHuesVector(hsi, gridSize);

   cout << "Processing similarity... ";
   vector<pair<double, image>> similarity = getMostSimilar(noValues, gridSize, dominantHues, imagedb);
   cout << "done\n";

   Mat gauss = gaussBidimensionalFilter(src, gaussKernel);
   hsi = BGRtoHSI(gauss);
   vector<double> srcHistogram = histogram(hsi);

   vector<double> pearsonCoefficients;
   for (pair<double, image> img : similarity)
   {
      double pearson = pearsonCorrelationCoefficient(srcHistogram, img.second.hist);
      pearsonCoefficients.push_back(pearson);
   }

   for (int i = 0; i < 3; i++)
   {
```

```
        for (int j = 0; j < pearsonCoefficients.size() - i - 1; j++)
        {
            if (pearsonCoefficients[j] < pearsonCoefficients[j + 1])
            {
                iter_swap(pearsonCoefficients.begin() + j, pearsonCoefficients.begin() + j + 1);
                iter_swap(similarity.begin() + j, similarity.begin() + j + 1);
            }
        }
    }
}
```

3.1 Conversion to HSI color space

Listing 2: BGR to HSI

```
for (int i = 0; i < height; i++)
{
    for (int j = 0; j < width; j++)
    {
        int pixel = (i * width + j) * 3;

        int R, G, B;
        double r, g, b;
        double H, S, I;

        R = imgData[pixel + 2];
        G = imgData[pixel + 1];
        B = imgData[pixel];

        r = (double)R / 255;
        g = (double)G / 255;
        b = (double)B / 255;

        I = (r + g + b) / 3;

        S = (I == 0 || (R == G && G == B)) ? 0 : (1 - min3(r, g, b) / I);

        double numerator = (r - g) + (r - b);
        double denominator = 2 * sqrt((r - g) * (r - g) + (r - b) * (g - b));

        H = (S == 0) ? 0 : acos( numerator / denominator );
        H *= 180 / PI;
        H = (b > g) ? 360 - H : H;
        hsiData[pixel + 2] = I;
        hsiData[pixel + 1] = S;
        hsiData[pixel] = H;
    }
}
```

3.2 Converting images to vectors of predominant hues

We split the image into a grid of size 'gridSize' * 'gridSize'. If we can't split the image evenly, the remainder is added to the first area

Listing 3: Vectors of predominant hues

```
int width = img.cols / gridSize;
int height = img.rows / gridSize;
int startWidth = img.cols - (gridSize - 1) * width;
int startHeight = img.rows - (gridSize - 1) * height;
```

```
int offset_x = 0;
int offset_y = 0;

for (int i = 0; i < gridSize; i++)
{
   offset_x = 0;
   for (int j = 0; j < gridSize; j++)
   {
      Rect roi;
      roi.x = offset_x;
      roi.y = offset_y;
      roi.width = (j > 0) ? width : startWidth;
      roi.height = (i > 0) ? height : startHeight;
      offset_x += roi.width;

      Mat crop = img(roi);
      double dhv = getDominantHueValue(crop);

      dominantHues.push_back(dhv);
   }
   offset_y += (i > 0) ? height : startHeight;
}
```

Listing 4: Dominant hue value

```
double avgHue = -1;

for (int i = 0; i < height; i++)
{
   for (int j = 0; j < width; j++)
   {
      int pixel = (i * width + j) * 3;

      if (hsiImg.at<Vec3d>(i, j)[1] != 0) // S != 0
      {
         avgHue += hsiImg.at<Vec3d>(i, j)[0]; // += H
      }
   }
}

if (avgHue != -1)
{
   avgHue += 1;
   avgHue /= width * height;
}
```

3.3 Finding the most similar images

Listing 5: Most similar images

```
for (image img : imagedb)
{
   int N = 0;
   double SUM = 0;
   for (int i = 0; i < gridSize * gridSize; i++)
   {
      if (dominantHues[i] != -1 && img.hues[i] != -1)
      {
         N++;
```

```
            double aux = abs(dominantHues[i] - img.hues[i]);
            SUM += pow(min(aux, 2 * 180 - aux), 2);
        }
    }

    if (N > 0)
    {
        double C = sqrt(N) * 180;
        double D = 1.0 * sqrt(SUM) / C;
        similarity.push_back({ 1 - D, img });
    }
    else
    {
        similarity.push_back({ 0, img });
    }
}

// bubble 'noValues' times to push the most similar images to the end
for (int i = 0; i < noValues; i++)
{
    for (int j = 0; j < similarity.size() - i - 1; j++)
    {
        if (similarity[j].first > similarity[j + 1].first)
        {
            iter_swap(similarity.begin() + j, similarity.begin() + j + 1);
        }
    }
}

vector<pair<double, image>> result;
int i = similarity.size() - noValues;
i = (i > 0) ? i : 0;
while (i < similarity.size())
{
    result.push_back(similarity[i++]);
}
```

3.4 Gaussian filter

Listing 6: Gaussian kernel

```
double sigma22 = 2 * sigma * sigma;
int x0, y0;
x0 = y0 = w / 2;
pair<vector<double>, vector<double>> gauss;
gauss.first = vector<double>(w);
gauss.second = vector<double>(w);

for (int i = 0; i < w; i++)
{
    double xx = (i - x0) * (i - x0);
    double yy = (i - y0) * (i - y0);

    gauss.first[i] = (1.0 / sqrt(PI * 2) / sigma) * exp(-xx / sigma22);
    gauss.second[i] = (1.0 / sqrt(PI * 2) / sigma) * exp(-yy / sigma22);
}

double c = 0;
for (int i = 0; i < w; i++)
    c += gauss.first[i];
for (int i = 0; i < w; i++)
```

```
   gauss.first[i] / c;

c = 0;
for (int i = 0; i < w; i++)
   c += gauss.second[i];
for (int i = 0; i < w; i++)
   gauss.second[i] / c;
```

Listing 7: Gaussian filter

```
int w = kernel.first.size();
int k = w / 2;

for (int i = 0; i < height; i++)
{
   for (int j = 0; j < width; j++)
   {
      int pixel = (i * width + j) * 3;
      if (i < k || i >= height - k || j < k || j >= width - k)
      {
         auxDstData[pixel] = srcData[pixel];
         auxDstData[pixel + 1] = srcData[pixel + 1];
         auxDstData[pixel + 2] = srcData[pixel + 2];
      }
      else
      {
         int sum = 0, sum1 = 0, sum2 = 0;
         for (int u = 0; u < w; u++)
         {
            int pixelK = (i * width + (j - k + u)) * 3;
            sum += (kernel.first[u] * srcData[pixelK]);
            sum1 += (kernel.first[u] * srcData[pixelK + 1]);
            sum2 += (kernel.first[u] * srcData[pixelK + 2]);
         }
         auxDstData[pixel] = sum;
         auxDstData[pixel + 1] = sum1;
         auxDstData[pixel + 2] = sum2;
      }
   }
}

for (int i = 0; i < height; i++)
{
   for (int j = 0; j < width; j++)
   {
      int pixel = (i * width + j) * 3;
      if (i < k || i >= height - k || j < k || j >= width - k)
      {
         dstData[pixel] = auxDstData[pixel];
         dstData[pixel + 1] = auxDstData[pixel + 1];
         dstData[pixel + 2] = auxDstData[pixel + 2];
      }
      else
      {
         int sum = 0, sum1 = 0, sum2 = 0;
         for (int u = 0; u < w; u++)
         {
            int pixelK = ((i - k + u) * width + j) * 3;
            sum += (kernel.second[u] * srcData[pixelK]);
            sum1 += (kernel.second[u] * srcData[pixelK + 1]);
            sum2 += (kernel.second[u] * srcData[pixelK + 2]);
```

```
            }
            dstData[pixel] = sum;
            dstData[pixel + 1] = sum1;
            dstData[pixel + 2] = sum2;
        }
    }
}
```

3.5 Pearson correlation coefficient

Listing 8: Pearson correlation coefficient

```
double coefficient = 0;

double sumX = 0, sumY = 0;
double sumX2 = 0, sumY2 = 0;
double sumXY = 0;
int n = x.size();

for (int i = 0; i < n; i++)
{
   sumX += x[i];
   sumY += y[i];
   sumXY += x[i] * y[i];
   sumX2 += x[i] * x[i];
   sumY2 += y[i] * y[i];
}

coefficient = (double)(n * sumXY - sumX * sumY)
        / (sqrt(n * sumX2 - sumX * sumX) * sqrt(n * sumY2 - sumY * sumY));
```

4 User guide

1. Open the project in Visual Studio 2019
2. Run the project
3. Input 1 to start
4. Input the grid size that will be used to get the predominant hues vector
5. Select the folder containing the images for the image database
6. Select the image to search with
7. Press any key to go back to step 6
8. Close the image dialog to go back to step 3

5 Results

The results of the image search using Pearson correlation coefficient indicate the level of similarity between the query image and the images in the image database. The values for the three most similar images are given as Similaritate 1, Similaritate 2, and Similaritate 3.

All the results show that the most similar image to be itself as expected, and indicated by the similarity score of 1.

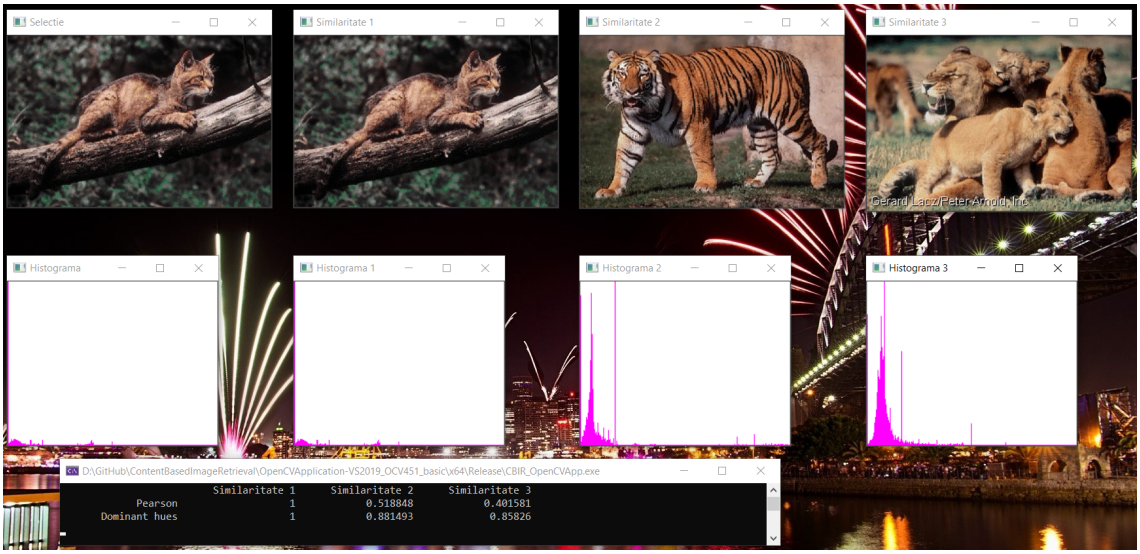


Figure 1: "Cat"

The second most similar image is a picture of a tiger, with a similarity score of 0.518848. This suggests that the query image and the tiger image share some common color features, which make them visually similar.

The third most similar image is a picture of some lions, with a similarity score of 0.401581. This indicates that the lion image has some color features in common with the query image, although to a lesser extent than the tiger image.

Overall, the results suggest that the image retrieval system is able to identify visually similar images based on their color features.

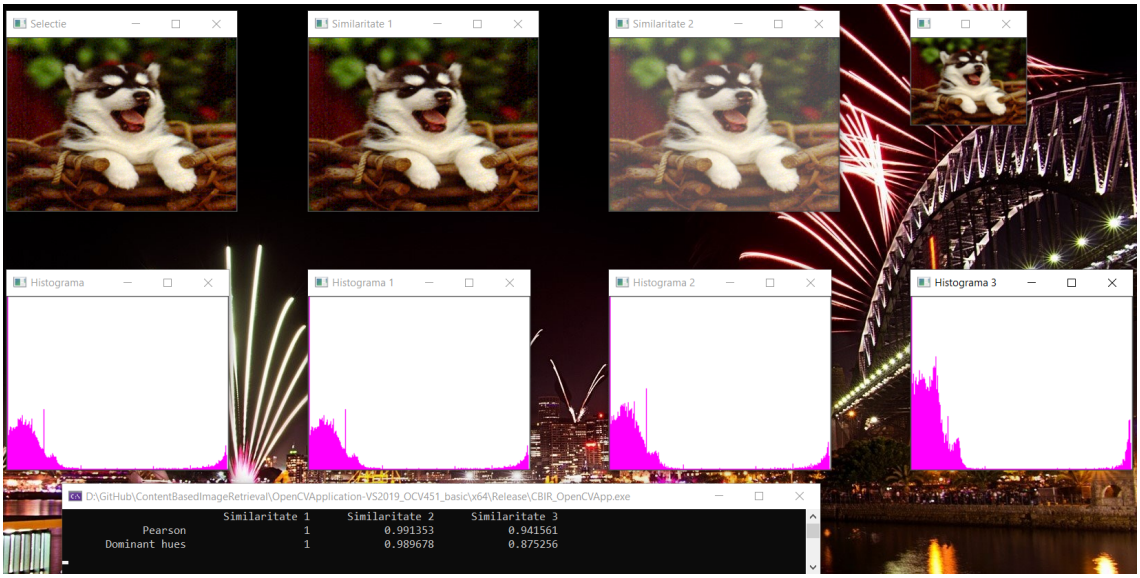


Figure 2: "Hund"

The second most similar image is the same dog image with some noise added, with a similarity score of 0.991353 and the third most similar image is the same dog image resized to be smaller, with a similarity score of 0.941561. This suggests that the image retrieval system is able to identify visually similar images even when they are slightly modified by adding noise or resized.

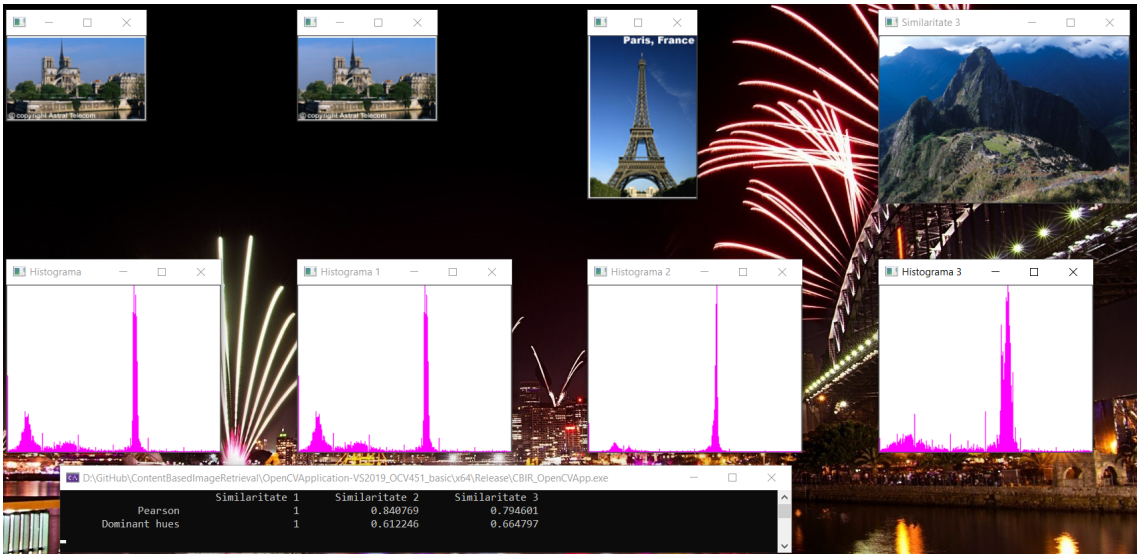


Figure 3: "Notre Dame"

The results of the image search using Pearson correlation coefficient for images with the same colored object and background but different sizes indicate the level of similarity between the query image and the images in the image database.

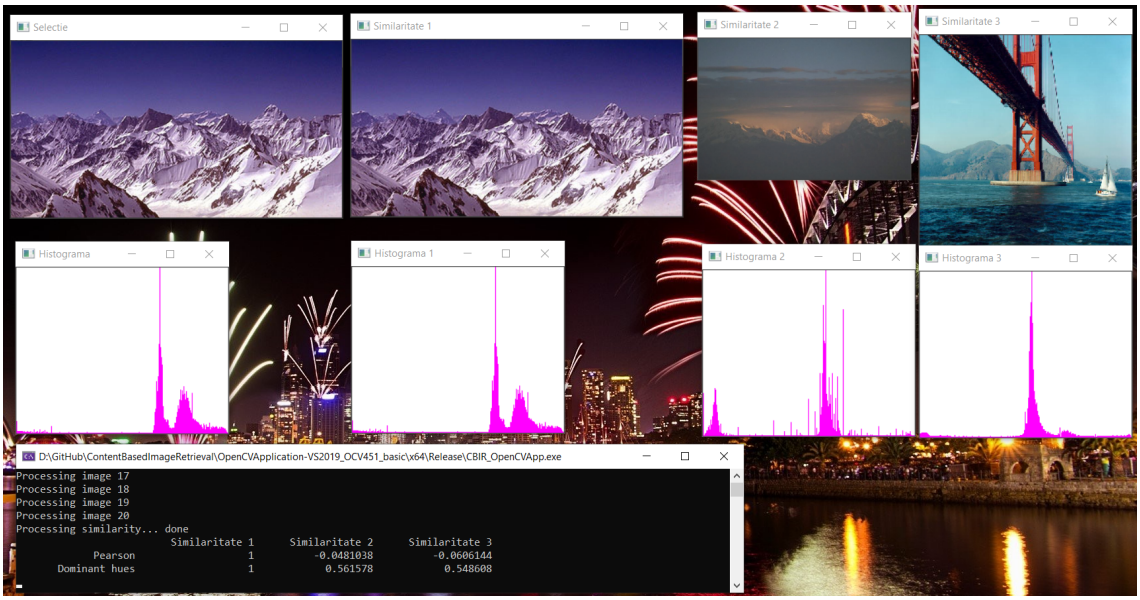


Figure 4: "Alps"

The results of the image search with no other similar images indicates that the image retrieval system accurately identifies that there are no visually similar images.

6 Conclusions

The content-based image retrieval system developed in this project successfully retrieved the most similar images based on color. The system used a grid-based approach to extract the predominant hues vector from the input image, which was then compared with the database images to find the most similar matches.

The project was implemented using Visual Studio 2019 and involved the user inputting the grid size, selecting the image database folder, and choosing the image to search with. The system was able to process the input images quickly and accurately, making it a useful tool for various applications such as image search and retrieval. Also by using some additional memory we managed to avoid redundant some reprocessing.

Overall, the project demonstrated the effectiveness of color-based image retrieval techniques and the potential of such systems in the field of image processing. Future work could involve expanding the system to incorporate other image features such as texture and shape to further improve the retrieval performance.

It should be noted that different results may be obtained when splitting the image into grids with different methods. The choice of grid size and shape, as well as the method used to calculate the predominant hues vector, can affect the retrieval performance of the system and using a grid size larger than the image size is not ideal.

References

[1] https://drive.google.com/file/d/1n-kH2kb8LKC8hTaiCUWGsaMBhnS-dj74/view *accessed on March 23, 2023*

[2] https://drive.google.com/file/d/1d1_VbqRfohH9AArnJ93tFQyTNYgtk-KJ/view *accessed on March 23, 2023*

[3] https://users.utcluj.ro/ rdanescu/pi_c09.pdf *accessed on March 23, 2023*

[4] https://biblioteca.utcluj.ro/files/carti-online-cu-coperta/625-8.pdf *accessed on March 23, 2023*

[5] https://en.wikipedia.org/wiki/Pearson_correlation_coefficient *accessed on March 23, 2023*

Listings