GÖRÜNTÜ İŞLEME PROJE RAPORU

Yöntem:

İlk adımda daha önceden hazırlanmış olan bölütleme algoritması kullanılarak, segmentasyon yapıldı. Her bir segmentin piksel sayısı hesaplanarak alanı 300 pikselden düşük olan segmentler -1 değeri ile işaretlenerek göz ardı edildi. Threshold değeri her bir resimde en fazla 3 sınıf olacağı düşünülerek deneme yoluyla elde edilmiştir. Her bir segmentin histogramı alınarak normalizasyon yapılmış ve alanı threshold değerinden büyük olan segmentlerin histogramları ve resim ismi bir dosyaya yazılmıştır. Her resim için aynı işlem tekrarlanmıştır.

Test kısmında her bir resim dosyadan okunarak segmentlerine ayrılmış ve histogramı alınmıştır. Resmin her bir segmenti, train setteki her resmin her segmenti ile karşılaştırılarak birbirine en çok benzeyen segmentler bulunmuştur. Bu sayede aynı resim olup farklı boyutlarda olan resimlerin üst üste çakıştırılarak karşılaştırılması sağlanmıştır. Örneğin iki resim sadece lokasyon farklılığıyla ya da açı farlılığıyla birbirinden ayrılabilir. Bu resimler aslında çok benzerdir. Bu durumda segmentlerin çakıştırılması yöntemi kullanıldığı için bir sorun oluşturmayacaktır.

Kaynak Kod:

```
//////////////////////////////////HEADER DOSYASI//////////////////////////////

#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <iostream>
#include <math.h>
#include <time.h>
#include <fstream>
#include <list>


using namespace cv;
using namespace std;
int** allocate_matrix(int width, int height);
int* randomize_clusters(int k, int* arrH);
int** cluster_pixels(int* means, int k, int** H_mat, int width, int height);
int calculate_min_dist(int* mean, int pixel, int k);
//void label_connected_components(int** clusters, int width, int height, Mat hsv, Mat
image, int** H_mat,String line);

void Tokenize(const string& str,
	vector<string>& tokens,
	const string& delimiters = " ")
{
	// Skip delimiters at beginning.
	string::size_type lastPos = str.find_first_not_of(delimiters, 0);
	// Find first "non-delimiter".
	string::size_type pos = str.find_first_of(delimiters, lastPos);

	while (string::npos != pos || string::npos != lastPos)
	{
		// Found a token, add it to the vector.
		tokens.push_back(str.substr(lastPos, pos - lastPos));
		// Skip delimiters.  Note the "not_of"
		lastPos = str.find_first_not_of(delimiters, pos);
		// Find next "non-delimiter"
		pos = str.find_first_of(delimiters, lastPos);
	}
```

```
}

void test_function(int** clusters, int width, int height, int** H_mat, String line)
{
	int label = 1;
	int** labeled_mat = allocate_matrix(width, height);

	for (int i = 0; i < width; i++)
		for (int j = 0; j < height; j++)
			labeled_mat[i][j] = -1;

	labeled_mat[0][0] = 0;

	for (int i = 0; i < width; i++)
	{
		for (int j = 0; j < height; j++)
		{
			if (i == 0 && j == 0)
			{
			}
			else if (i == 0) // ilk satırda sadece soldaki piksele bakılır.
			{
				if (clusters[i][j - 1] == clusters[i][j])
					labeled_mat[i][j] = labeled_mat[i][j - 1];
				else
				{
					labeled_mat[i][j] = label;
					label++;
				}
			}
			else if (j == 0) //ilk sütunda yukarıdaki ve sağ çaprazdaki
piksellere bakılır.
			{
				if (clusters[i][j] == clusters[i - 1][j])
					labeled_mat[i][j] = labeled_mat[i - 1][j];
				else if (clusters[i][j] == clusters[i - 1][j + 1])
					labeled_mat[i][j] = labeled_mat[i - 1][j + 1];
				else
				{
					labeled_mat[i][j] = label;
					label++;
				}
			}
			else if (j == height - 1) // son sütunda sağ yukarı çaprazındaki,
yukarıdaki ve soldaki piksellere bakılır.
			{
				if (clusters[i][j] == clusters[i - 1][j])
					labeled_mat[i][j] = labeled_mat[i - 1][j];
				else if (clusters[i][j] == clusters[i - 1][j - 1])
					labeled_mat[i][j] = labeled_mat[i - 1][j - 1];
				else if (clusters[i][j] == clusters[i][j - 1])
					labeled_mat[i][j] = labeled_mat[i][j - 1];
				else
				{
					labeled_mat[i][j] = label;
					label++;
				}
			}
			else
			{ // sağ ve sol yukarı çaprazdaki sınıflar pikselle aynıysa üçünü
de aynı etiketle etiketle ve bağlı pikselleri de
```

```
                        if (clusters[i][j] == clusters[i - 1][j + 1] &&
clusters[i][j] == clusters[i - 1][j - 1])
                        {
                                for (int x = 0; x <= i - 1; x++)
                                {
                                        for (int y = 0; y <= j + 1; y++)
                                                if (labeled_mat[i - 1][j + 1] ==
labeled_mat[x][y])
                                                        labeled_mat[x][y] =
labeled_mat[i - 1][j - 1];
                                }
                                labeled_mat[i][j] = labeled_mat[i - 1][j - 1];
                        }
                        // soldaki ve sağ yukarı çaprazdaki sınıflar pikselle
aynıysa üçünü de aynı etiketle etiketle ve bağlı pikselleri de
                        else if (clusters[i][j] == clusters[i - 1][j + 1] &&
clusters[i][j] == clusters[i][j - 1])
                        {
                                for (int x = 0; x <= i - 1; x++)
                                {
                                        for (int y = 0; y <= j + 1; y++)
                                                if (labeled_mat[i - 1][j + 1] ==
labeled_mat[x][y])
                                                        labeled_mat[x][y] =
labeled_mat[i][j - 1];
                                }
                                labeled_mat[i][j] = labeled_mat[i][j - 1];
                        }
                        else if (clusters[i][j] == clusters[i - 1][j + 1])

                                labeled_mat[i][j] = labeled_mat[i - 1][j + 1];

                        else if (clusters[i][j] == clusters[i - 1][j])

                                labeled_mat[i][j] = labeled_mat[i - 1][j];

                        else if (clusters[i][j] == clusters[i - 1][j - 1])

                                labeled_mat[i][j] = labeled_mat[i - 1][j - 1];

                        else if (clusters[i][j] == clusters[i][j - 1])

                                labeled_mat[i][j] = labeled_mat[i][j - 1];

                        else
                        {
                                labeled_mat[i][j] = label;
                                label++;
                        }
                }

        }
    }


    // her segmentin alanının bulunması
    int* segments = new int[label];
    int thr_val = 300; // en fazla 3 sınıf oldugu düşünülerek threshold degeri
seçilmiştir.
    for (int i = 0; i < label; i++)
        segments[i] = -1;
```

```
	for (int i = 0; i < width; i++)
		for (int j = 0; j < height; j++)
			segments[labeled_mat[i][j]]++;

	// segmentlerin onemli olanlarının tespiti
	for (int i = 0; i < label; i++)
		if (segments[i] < thr_val)
			segments[i] = -1;

	// segment sayısı kadar satır ve Hue değerleri için 180 sütunluk yer açılması
	double** hist_of_segments = new double*[label];
	for (int i = 0; i < label; ++i)
		hist_of_segments[i] = new double[180];

	for (int i = 0; i < label; i++)
		for (int j = 0; j < 180; j++)
			hist_of_segments[i][j] = 0;

	int* total_pixel = new int[label]; // normalizasyon için her segmentin toplam piksel sayısının tutulması.
	for (int i = 0; i < label; i++)
		total_pixel[i] = 0;

	// Segmentlere göre sınıflandırılarak, piksellerin Hue değerlerinin histogramının alınması
	for (int i = 0; i < width; i++)
		for (int j = 0; j < height; j++)
		{
			//Vec3b c = hsv.at<Vec3b>(Point(j, i));
			//cout << "c:" << c.val[0]<< " ";
			hist_of_segments[labeled_mat[i][j]][H_mat[i][j]]++; // Her segmentteki piksellerin Hue degerlerinin histogramının alınması
			total_pixel[labeled_mat[i][j]]++;
		}


	for (int i = 0; i < label; i++)
	{
		for (int j = 0; j < 180; j++)
		{
			if (total_pixel[i] != 0)
				hist_of_segments[i][j] /= total_pixel[i]; // Histogram değerlerinin normalize edilmesi
			//cout << hist_of_segments[i][j] << " ";
		}
	}

		list<string> results;

		ifstream reader;
		reader.open("C:/Users/mukad/Desktop/CBIR/TRAIN/train.txt");
		if (!reader.is_open()) {
			cout << "ERROR: Failed to open " << endl;
			cout << "Now exiting \n";
			exit(-1);
		}

		int c = 0;
		int result[30];
		String resultss[100][100];
		String** res = new String*[label];
		for (int i = 0; i < label; ++i)
```

```
                        res[i] = new String[50];
                String s = "";

                for (String line; getline(reader, line);)
                {


                        double min = 10000;
                        vector<string> tokens; // her bir resmin adını ve segmentlerin
histogramlarını tutan dizi.
                        Tokenize(line, tokens);
                        int counter = 1;
                        double train_segment[180];

                        while (counter < tokens.size())
                        {
                                for (int i = 0; i < 180; i++)
                                {
                                        train_segment[i] = std::stod(tokens[counter]);
                                        counter++;
                                }
                                cv::Mat trainSet = cv::Mat(1, 4, CV_8UC1, train_segment);

                                for (int i = 0; i < label; i++)
                                {
                                        if (segments[i] != -1)
                                        {
                                                cv::Mat testSet = cv::Mat(1, 4, CV_8UC1,
segments[i]);
                                                min = compareHist(trainSet, testSet,
CV_COMP_BHATTACHARYYA);
                                        }
                                }

                                s = tokens[0];
                                result[c] = min;

                        }



                        }
                cout << s << endl;
                cout << "min: " << endl;
}

int** cluster_pixels(int* means, int k, int** H_mat, int width, int height)
{
        int c = 0;
        int* new_means = (int*)malloc(k*sizeof(int));
        int** clusters_cont = allocate_matrix(k, width*height); // sınıflardaki hue
değerlerinin tutulduğu matris.
        int** location = allocate_matrix(width*height, 3); // tüm piksellerin
koordinatlarını ve ait oldukları sınıfları tutan matris.
        int** clusters = allocate_matrix(width, height); // piksellerin ait oldukları
sınıfları tutan matris. return de .
        int* counter = (int*)malloc(k*sizeof(int)); // sınıfların eleman sayısını tutan
dizi.
        for (int i = 0; i < k; i++)
                counter[i] = 0;
        for (int i = 0; i < k; i++)
                for (int j = 0; j < width*height; j++)
```

```
                        clusters_cont[i][j] = -1;

        for (int x = 0; x < width; x++)
        {
                for (int y = 0; y < height; y++)
                {
                        int index_cluster = calculate_min_dist(means, H_mat[x][y], k); //
pikselin ait olduğu sınıfın indisinin bulunması.
                        clusters_cont[index_cluster][counter[index_cluster]] =
H_mat[x][y]; // sınıflara ait Hue değerlerini tutan matrise elemanın eklenmesi.
                        // pikselin x,y koordinat değerini ve ait olduğu sınıfı tutan
matrisin güncellenmesi.
                        location[c][0] = x;
                        location[c][1] = y;
                        location[c][2] = index_cluster;
                        counter[index_cluster]++;
                        c++;
                }
        }
        // yeni mean değerlerinin hesaplanması.
        for (int i = 0; i < k; i++)
        {
                int sum = 0;
                for (int j = 0; j < counter[i]; j++)
                {
                        sum += clusters_cont[i][j];
                }
                new_means[i] = sum / (counter[i] + 1);
        }
        int flag = 0;
        // sınıflar değişmeyinceye kadar clustering e devam et.
        for (int i = 0; i < k; i++)
        {   // eski ve yeni mean değerlerini karşılaştır.
                if (means[i] != new_means[i])
                        flag = 1;
        }
        if (flag == 1)
                cluster_pixels(new_means, k, H_mat, width, height);

        // her bir pikselin ait olduğu sınıfı gösteren dizinin oluşturulması.
        for (int i = 0; i < width*height; i++)

        {
                clusters[location[i][0]][location[i][1]] = location[i][2];
        }


        //for (int j = 0; j < width; j++)
        //{
        //      for (int i = 0; i < height; i++)
        //      {
        //              //if (clusters[j][i] == -1)
        //              cout << clusters[i][j] << endl;

        //      }

        //      cout << endl;
        //}
        return clusters;
}
```

```
//void label_connected_components(int** clusters, int width, int height, Mat hsv, Mat
image, int** H_mat,String line)

void label_connected_components(int** clusters, int width, int height, int** H_mat,
String line)
{
	int label = 1;
	int** labeled_mat = allocate_matrix(width, height);

	for (int i = 0; i < width; i++)
		for (int j = 0; j < height; j++)
			labeled_mat[i][j] = -1;

	labeled_mat[0][0] = 0;

	for (int i = 0; i < width; i++)
	{
		for (int j = 0; j < height; j++)
		{
			if (i == 0 && j == 0)
			{
			}
			else if (i == 0) // ilk satırda sadece soldaki piksele bakılır.
			{
				if (clusters[i][j - 1] == clusters[i][j])
					labeled_mat[i][j] = labeled_mat[i][j - 1];
				else
				{
					labeled_mat[i][j] = label;
					label++;
				}
			}
			else if (j == 0) //ilk sütunda yukarıdaki ve sağ çaprazdaki
piksellere bakılır.
			{
				if (clusters[i][j] == clusters[i - 1][j])
					labeled_mat[i][j] = labeled_mat[i - 1][j];
				else if (clusters[i][j] == clusters[i - 1][j + 1])
					labeled_mat[i][j] = labeled_mat[i - 1][j + 1];
				else
				{
					labeled_mat[i][j] = label;
					label++;
				}
			}
			else if (j == height - 1) // son sütunda sağ yukarı çaprazındaki,
yukarıdaki ve soldaki piksellere bakılır.
			{
				if (clusters[i][j] == clusters[i - 1][j])
					labeled_mat[i][j] = labeled_mat[i - 1][j];
				else if (clusters[i][j] == clusters[i - 1][j - 1])
					labeled_mat[i][j] = labeled_mat[i - 1][j - 1];
				else if (clusters[i][j] == clusters[i][j - 1])
					labeled_mat[i][j] = labeled_mat[i][j - 1];
				else
				{
					labeled_mat[i][j] = label;
					label++;
				}
			}
			else
```

```
                    { // sağ ve sol yukarı çaprazdaki sınıflar pikselle aynıysa üçünü
de aynı etiketle etiketle ve bağlı pikselleri de
                        if (clusters[i][j] == clusters[i - 1][j + 1] &&
clusters[i][j] == clusters[i - 1][j - 1])
                        {
                            for (int x = 0; x <= i - 1; x++)
                            {
                                for (int y = 0; y <= j + 1; y++)
                                    if (labeled_mat[i - 1][j + 1] ==
labeled_mat[x][y])
                                        labeled_mat[x][y] =
labeled_mat[i - 1][j - 1];
                            }
                            labeled_mat[i][j] = labeled_mat[i - 1][j - 1];
                        }
                        // soldaki ve sağ yukarı çaprazdaki sınıflar pikselle
aynıysa üçünü de aynı etiketle etiketle ve bağlı pikselleri de
                        else if (clusters[i][j] == clusters[i - 1][j + 1] &&
clusters[i][j] == clusters[i][j - 1])
                        {
                            for (int x = 0; x <= i - 1; x++)
                            {
                                for (int y = 0; y <= j + 1; y++)
                                    if (labeled_mat[i - 1][j + 1] ==
labeled_mat[x][y])
                                        labeled_mat[x][y] =
labeled_mat[i][j - 1];
                            }
                            labeled_mat[i][j] = labeled_mat[i][j - 1];
                        }
                        else if (clusters[i][j] == clusters[i - 1][j + 1])

                            labeled_mat[i][j] = labeled_mat[i - 1][j + 1];

                        else if (clusters[i][j] == clusters[i - 1][j])

                            labeled_mat[i][j] = labeled_mat[i - 1][j];

                        else if (clusters[i][j] == clusters[i - 1][j - 1])

                            labeled_mat[i][j] = labeled_mat[i - 1][j - 1];

                        else if (clusters[i][j] == clusters[i][j - 1])

                            labeled_mat[i][j] = labeled_mat[i][j - 1];

                        else
                        {
                            labeled_mat[i][j] = label;
                            label++;
                        }
                    }

            }
        }

        //Vec3b co;
        //cvtColor(image, hsv, CV_BGR2HSV);

        //for (int i = 0; i < width; i++)
        //    for (int j = 0; j < height; j++)
        //    {
```

```
	//			Vec3b c;
	//			c.val[0] = ((labeled_mat[i][j] * 31) % 180); // 360 100 100
olmuyor
	//			co = hsv.at<Vec3b>(Point(j, i));
	//			c.val[1] = co.val[1];
	//			c.val[2] = co.val[2];
	//			//c.val[1] = (labeled_mat[i][j]* 31) % 255;
	//			//c.val[2] = (labeled_mat[i][j]* 31) % 255;
	//			hsv.at<Vec3b>(Point(j, i)) = c;
	//		}

	//cvtColor(hsv, image, CV_HSV2BGR);
	//imshow("Display windowww", image);

	// her segmentin alanının bulunması
	int* segments = new int[label];
	int thr_val = 300; // en fazla 3 sınıf oldugu düşünülerek threshold degeri
seçilmiştir.
	for (int i = 0; i < label; i++)
		segments[i] = -1;

	for (int i = 0; i < width; i++)
		for (int j = 0; j < height; j++)
			segments[labeled_mat[i][j]]++;

	// segmentlerin onemli olanlarının tespiti
	for (int i = 0; i < label; i++)
		if (segments[i] < thr_val)
			segments[i] = -1;

	// segment sayısı kadar satır ve Hue değerleri için 180 sütunluk yer açılması
	double** hist_of_segments = new double*[label];
	for (int i = 0; i < label; ++i)
		hist_of_segments[i] = new double[180];

	for (int i = 0; i < label; i++)
		for (int j = 0; j < 180; j++)
			hist_of_segments[i][j] = 0;

	int* total_pixel = new int[label]; // normalizasyon için her segmentin toplam
piksel sayısının tutulması.
	for (int i = 0; i < label; i++)
		total_pixel[i] = 0;

	// Segmentlere göre sınıflandırılarak, piksellerin Hue değerlerinin
histogramının alınması
	for (int i = 0; i < width; i++)
		for (int j = 0; j < height; j++)
		{
			//Vec3b c = hsv.at<Vec3b>(Point(j, i));
			//cout << "c:" << c.val[0]<< " ";
			hist_of_segments[labeled_mat[i][j]][H_mat[i][j]]++; // Her
segmentteki piksellerin Hue degerlerinin histogramının alınması
			total_pixel[labeled_mat[i][j]]++;
		}

	//for (int i = 0; i < label; i++)
	//	for (int j = 0; j < 180; j++)
	//	{
	//		cout << hist_of_segments[i][j] << " ";
	//		//cout << total_pixel[i] << " : label ";
	//	}
```

```
	for (int i = 0; i < label; i++)
	{
		for (int j = 0; j < 180; j++)
		{
			if (total_pixel[i] != 0)
				hist_of_segments[i][j] /= total_pixel[i]; // Histogram
değerlerinin normalize edilmesi
			//cout << hist_of_segments[i][j] << " ";
		}
		//cout << endl;
	}

	ofstream writer;
	writer.open("C:/Users/mukad/Desktop/CBIR/TRAIN/train.txt", ios::app); // Append
mode , ios::app
	writer << "\n" << line << " ";
	for (int i = 0; i < label; i++)
	{
		if (segments[i] != -1) // thresholdu aşan segmentlerin histogramını
dosyaya yazma
		{

			for (int j = 0; j < 180; j++)
			{
				writer << hist_of_segments[i][j] << " ";
			}

			//writer << "\n";
		}


	}



	writer.close(); // Closing the file

	free(labeled_mat);

}

int calculate_min_dist(int* means, int pixel, int k)
{
	int min_dist = 1000;
	int index_cluster = 1000;
	int man_dist = 0;
	for (int i = 0; i < k; i++)
	{// manhattan distance hesaplaması
		man_dist = abs(means[i] - pixel);
		if (min_dist > man_dist)
		{
			min_dist = man_dist;
			index_cluster = i;
		}
	}
	return index_cluster;
}
int* randomize_clusters(int k, int* arrH)
{
	srand(time(NULL));
	int count = 0;
	int* means = (int*)malloc(k*sizeof(int));
```

```
	while (count < k)
	{
		int pixel = rand() % 180;
		int i = 0;
		while (i < count)
		{
			while (means[i] == pixel || arrH[pixel] == 0)
				pixel = rand() % 180;
			i++;
		}
		means[count] = pixel;
		count++;
	}
	/*for (int i = 0; i < k; i++)
	cout << means[i] << " " << endl;*/
	return means;
}


int** allocate_matrix(int width, int height)
{
	int** matrix = (int**)malloc(width * sizeof(int*));
	if (matrix == NULL)
	{
		cout << "Memory Allocation(rows) Error!!!";
		exit(-1);
	}
	for (int i = 0; i < width; i++)
	{
		matrix[i] = (int*)malloc(height * sizeof(int));
		if (matrix[i] == NULL)
		{
			cout << "Memory Allocation(cols) Error!!!";
			exit(-1);
		}
	}

	return matrix;
}

//////////////////////////////////SOURCE DOSYASI////////////////////////////////

#include "Header.h"


int main(int argc, char** argv)
{
	int* arrH = (int*)malloc(180 * sizeof(int)); // Hue değerlerinin kaç tane
pikselde olduğunu tutan dizi.
	int* means = new int();
	int** clusters = NULL;
	Mat image;
	Mat  hsv;
	int k=3;
	int** H_mat; // Hue değerlerini tutan matris.
	String s = "muk";
	cout << s << endl;
	ifstream reader;

	// Training İşlemi
	reader.open("C:/Users/mukad/Desktop/CBIR/TRAIN/resimler.txt");
	if (!reader.is_open()) {
```

```
                cout << "ERROR: Failed to open "  << endl;
                cout << "Now exiting \n";
                exit(-1);
        }

        for (String line; getline(reader, line);)
        {
                image = imread(line, CV_LOAD_IMAGE_COLOR);
//"C:/Users/mukad/Desktop/CBIR/TRAIN/053.bmp"

                if (!image.data)                              // Görüntüyü bulup
bulamadığını kontrol et
                {
                        cout << "Could not open or find the image" << std::endl;
                        return -1;
                }
                // 1. adım: RGB resmi HSV formatlı resme dönüştürme.
                cvtColor(image, hsv, CV_BGR2HSV);


                // Hue değerlerini tutacak matris için yer açma.
                H_mat = allocate_matrix(hsv.rows, hsv.cols);

                for (int i = 0; i < 180; i++)
                        arrH[i] = 0;

                for (int i = 0; i < hsv.rows; i++)
                        for (int j = 0; j < hsv.cols; j++)
                        {
                                Vec3b color = hsv.at<Vec3b>(Point(j, i));
                                H_mat[i][j] = color.val[0]; //hue
                                arrH[color.val[0]]++; //hue değerine göre histogram
değerlerini elde etme.
                        }

                //cout << "sinif (k) sayisini giriniz: " << endl;
                //cin >> k;

                means = randomize_clusters(k, arrH);
                clusters = cluster_pixels(means, k, H_mat, hsv.rows, hsv.cols);
                label_connected_components(clusters, hsv.rows, hsv.cols, H_mat,line);

                free(H_mat);
        }

        // Test işlemi
        reader.open("C:/Users/mukad/Desktop/CBIR/TEST/resimler.txt");
        if (!reader.is_open()) {
                cout << "ERROR: Failed to open " << endl;
                cout << "Now exiting \n";
                exit(-1);
        }

        for (String line2; getline(reader, line2);)
        {
                image = imread(line2, CV_LOAD_IMAGE_COLOR);
//"C:/Users/mukad/Desktop/CBIR/TRAIN/053.bmp"

                if (!image.data)                              // Görüntüyü bulup
bulamadığını kontrol et
                {
                        cout << "Could not open or find the image" << std::endl;
```

```
                return -1;
            }
            // 1. adım: RGB resmi HSV formatlı resme dönüştürme.
            cvtColor(image, hsv, CV_BGR2HSV);


            // Hue değerlerini tutacak matris için yer açma.
            H_mat = allocate_matrix(hsv.rows, hsv.cols);

            for (int i = 0; i < 180; i++)
                arrH[i] = 0;

            for (int i = 0; i < hsv.rows; i++)
                for (int j = 0; j < hsv.cols; j++)
                {
                    Vec3b color = hsv.at<Vec3b>(Point(j, i));
                    H_mat[i][j] = color.val[0]; //hue
                    arrH[color.val[0]]++; //hue değerine göre histogram
değerlerini elde etme.
                }

            //cout << "sinif (k) sayisini giriniz: " << endl;
            //cin >> k;

            means = randomize_clusters(k, arrH);
            clusters = cluster_pixels(means, k, H_mat, hsv.rows, hsv.cols);
            test_function(clusters, hsv.rows, hsv.cols, H_mat, line2);

            free(H_mat);

        }

        free(arrH);

        free(means);
        free(clusters);

        waitKey(0);
        return 0;
}

_
```