Görüntü

İşleme

Ödev 2

k-Means yöntemi ile renkli resmi bölütleme

Sardor HAZRATOV

14011901

Yöntem: k-Means Clustering algoritmasının genel yapısı aşağıdaki gibidir:
a)Rastgele resimden k tane pixel seçilir. Seçilen pixeller oluşturacağımız sınıflar için merkez değerleri olacaktır;
b)Resim üzerinden her pixel (bir önceki adımda) seçilmiş merkezlerin RGB değerlerine göre distance hesaplanır. O pixel hangi sınıfın merkezine yakınsa onu sınıfın etiketi veya numarası ile etiketlendirilir;
c)Tüm pixeller sınıflara atandıktan sonra her sınıfın RGB değerlerine göre ortalaması hesaplanır ve hesaplanan ortalamaya göre sınıfın yeni merkezi oluşturulur;
d)Belirli bir epsilon değerine göre sınıf merkezlerinin arasındaki distance büyük ise veya iterasyon sayısına göre (b) adımdan tekrarlanır;
e)Sınıf içerisindeki tüm pixeller sınıf merkezin rengine boyanır;

Kod içerisindeki ana adımlar:

Resim içerisinden *n* cluster sayısı kadar random pixel seçilir;
Seçilen pixelin RGB değerleri *kCenters[i]* sınıf merkezleri dizisine atanır;

```
for (int i = 0; i < n; i++){
	int c = rand() % cols;
	int r = rand() % rows;
	Vec3b intensity = image.at<Vec3b>(r, c);
	kCenters[i].r = intensity.val[2];
	kCenters[i].g = intensity.val[1];
	kCenters[i].b = intensity.val[0];
}
```

---

stop bayrak değişkeni *false* iken ve iterasyon sayısı 1000 altındayken her pixeli en yakın olduğu sınıfı belirlenir ve *classLabel[i][j]* diye bir etiket matrisinde etiketlenir;
calculateAvgOfClasses() sınıf içerisindeki pixellerin renginin ortalaması alınır ve yeni sınıf merkezi belirlenir;
stop bayrak değişkeni *isNew()* fonksiyonunda sınıf merkezlerin bir önceki değerleri arasındaki distance göre değişir, distance *eps'*den küçük ise *true* olur;

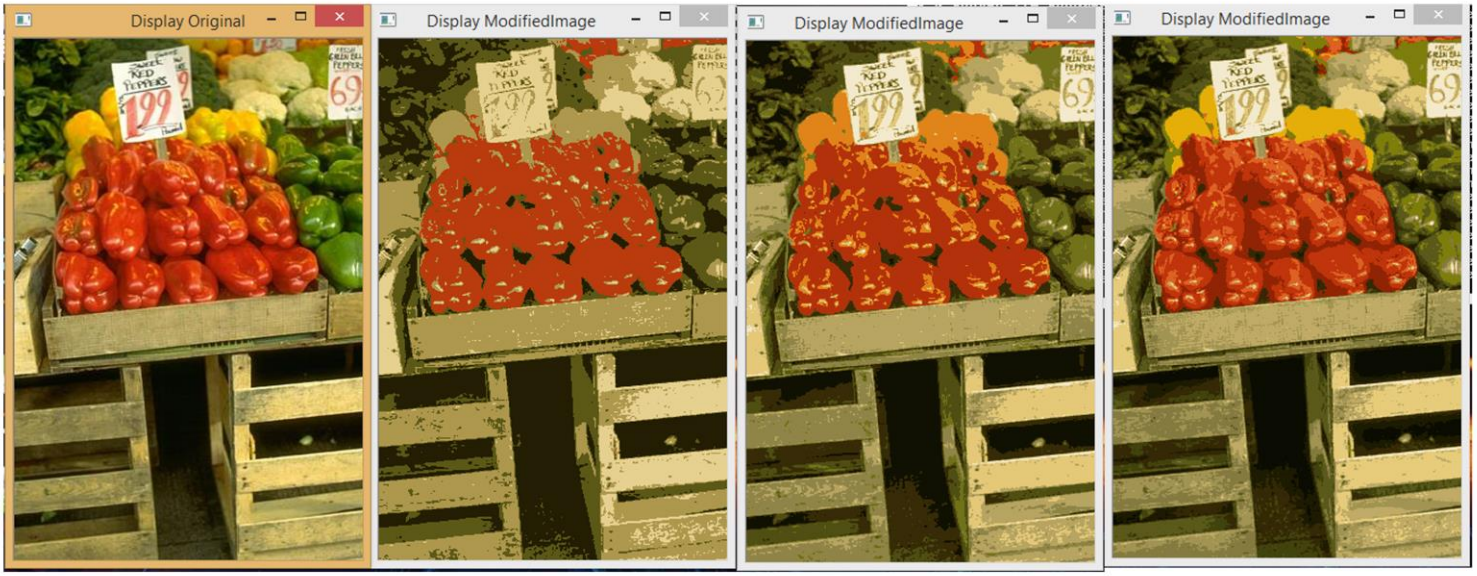
```
while (!stop && iteration<1000){
		for (int i = 0; i < rows; i++){
			for (int j = 0; j < cols; j++){
				intensity = image.at<Vec3b>(i, j);
			blue = intensity.val[0];
			green = intensity.val[1];
			red = intensity.val[2];
			classLabel[i][j] = calculateDistance(kCenters, red, green, blue, n);
			}/*end of for j*/
		}/*end of for i*/
		for (int k = 0; k < n; k++){
			kCentersOld[k] = kCenters[k];
		}
		calculateAvgOfClasses(classLabel, image, kCenters, n);
		stop = isNew(n, kCenters, kCentersOld, eps);
		iteration++;
	}/*end while*/
```

classLabel[i][j]'de sınıfa göre etiketlenmiş pixeller sınıf merkezlerinin rengine boyanır;

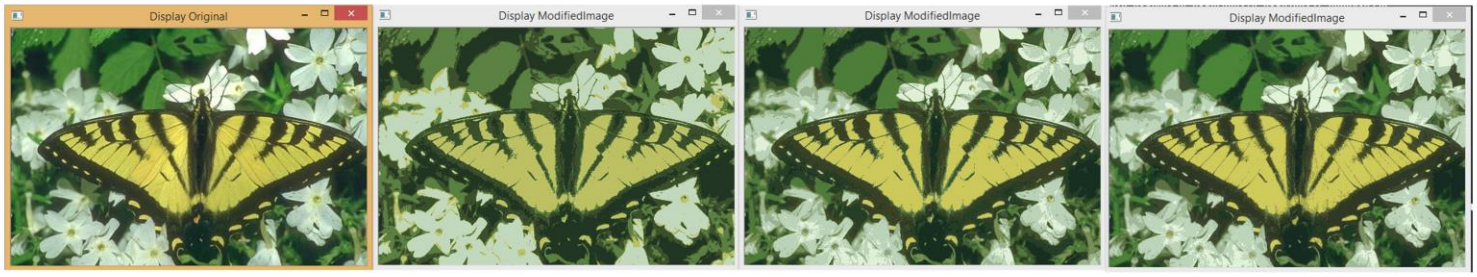
```
Mat imageModified = image;
	for (int k = 0; k < n; k++){
		for (int i = 0; i < rows; i++){
			for (int j = 0; j < cols; j++){
				if (classLabel[i][j] == k){
					intensity.val[0] = kCenters[k].b;
					intensity.val[1] = kCenters[k].g;
					intensity.val[2] = kCenters[k].r;
					imageModified.at<Vec3b>(i, j) = intensity;
				}
			}
		}
	}
```

Uygulama:

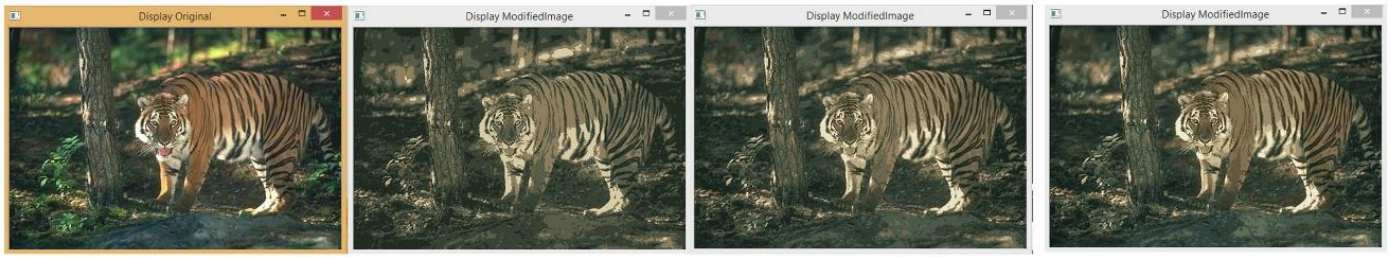
Örnek resimler:

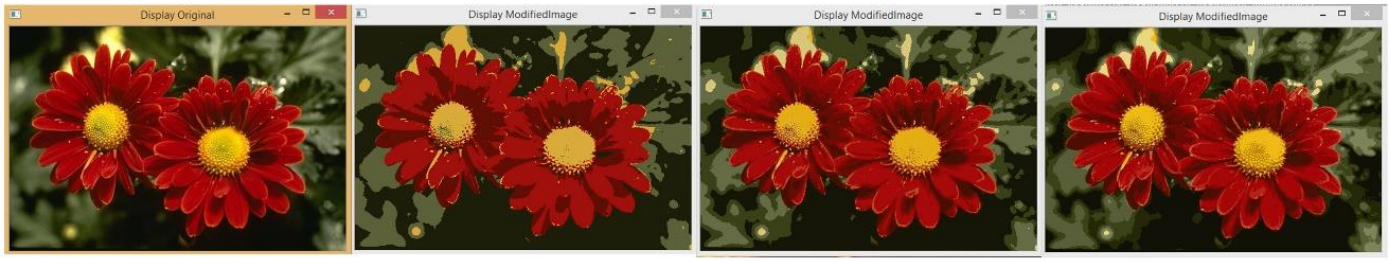


Orjinal Resim k=5 k=10 k=15

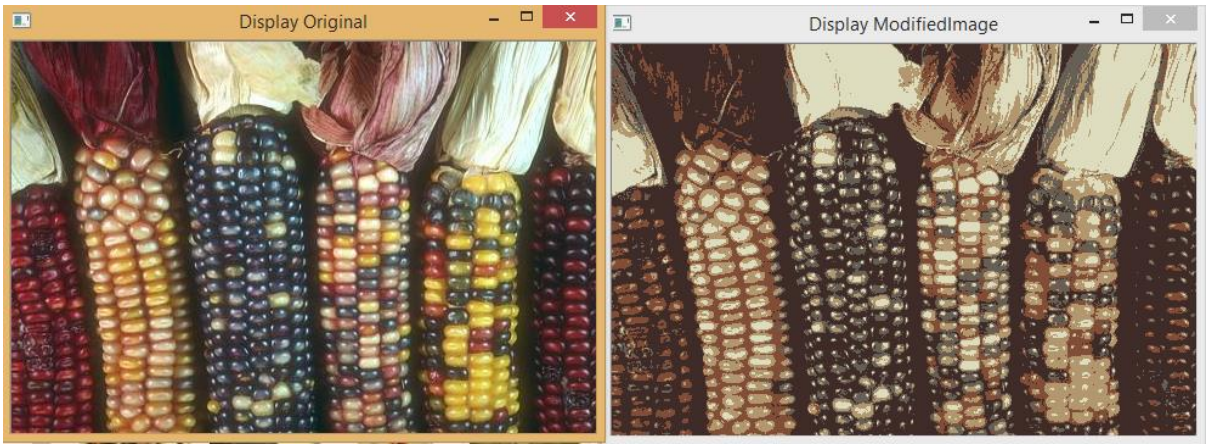


Orjinal Resim k=5 k=10 k=15

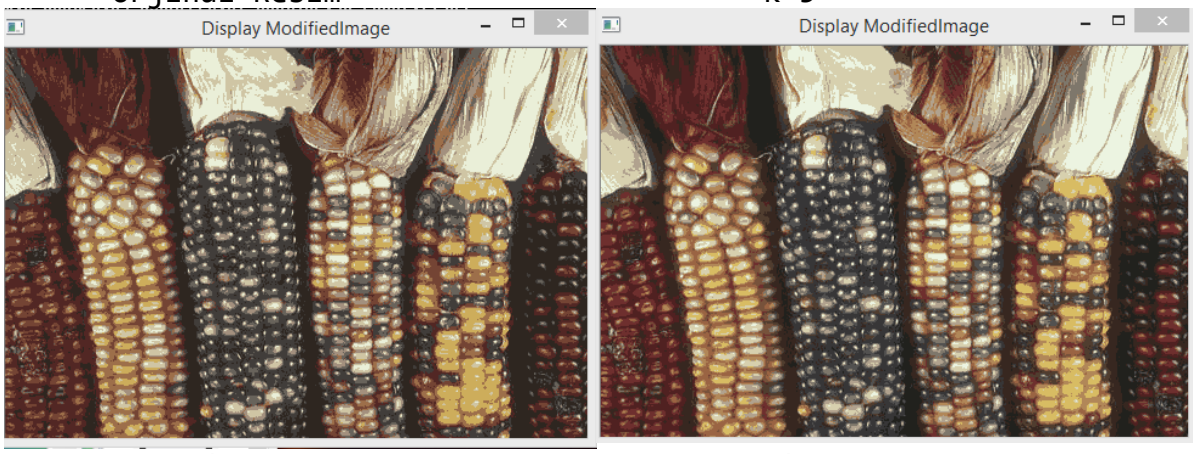




Orjinal Resim k=5 k=10 k=15



Orjinal Resim k=5



k=10 k=15

Program kodu:

```
#include "stdafx.h"
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2\opencv.hpp>
#include <iostream>
#include <math.h>
#include <stdlib.h>

using namespace cv;
using namespace std;

typedef struct{
	uchar r;
	uchar g;
	uchar b;
}RGBcolor;

void calculateAvgOfClasses(int **, Mat, RGBcolor *, int);
bool isNew(int, RGBcolor *, RGBcolor *, float);
int calculateDistance(RGBcolor *, uchar, uchar, uchar, int);


int main(int argc, char** argv)
{
	if (argc != 2)
	{
		cout << " Usage: display_image ImageToLoadAndDisplay" << endl;
		return -1;
	}

	Mat image;
	image = imread(argv[1], CV_LOAD_IMAGE_COLOR);   // Read the file
	if (!image.data)                              // Check for invalid input
	{
		cout << "Could not open or find the image" << std::endl;
		return -1;
	}
	namedWindow("Display Original", WINDOW_AUTOSIZE);// Create a window for
display.
	imshow("Display Original", image);

	int rows = image.rows;
	int cols = image.cols;
	int n = 4;
	float eps = 0.005;
	bool stop;
	RGBcolor *kCenters, *kCentersOld;
	int **classLabel;

	cout << "\nEnter the k value:";
	cin >> n;
	cout << "\nEnter the eps value:";
	cin >> eps;

	if ((kCenters = (RGBcolor*)malloc(sizeof(RGBcolor)*n)) == NULL){
		cout << "\nCould not allocate memory" << std::endl;
		return -1;
	}
	if ((kCentersOld = (RGBcolor*)malloc(sizeof(RGBcolor)*n)) == NULL){
		cout << "\nCould not allocate memory" << std::endl;
		return -1;
```

```
	}

	classLabel = (int**)malloc(sizeof(int*)*rows);
	for (int i = 0; i < rows; i++){
		classLabel[i] = (int*)malloc(sizeof(int)*cols);
	}
	if (classLabel == NULL){
		cout << "\nCould not allocate memory" << std::endl;
		return -1;
	}

	for (int i = 0; i < n; i++){
		int c = rand() % cols;
		int r = rand() % rows;
		Vec3b intensity = image.at<Vec3b>(r, c);
		kCenters[i].r = intensity.val[2];
		kCenters[i].g = intensity.val[1];
		kCenters[i].b = intensity.val[0];
	}

	stop = false;
	Vec3b intensity;
	uchar blue;
	uchar green;
	uchar red;
	int iteration = 0;
	while (!stop && iteration<1000){
		for (int i = 0; i < rows; i++){
			for (int j = 0; j < cols; j++){
				intensity = image.at<Vec3b>(i, j);
				blue = intensity.val[0];
				green = intensity.val[1];
				red = intensity.val[2];
				classLabel[i][j] = calculateDistance(kCenters, red, green,
blue, n);
			}/*end of for j*/
		}/*end of for i*/
		for (int k = 0; k < n; k++){
			kCentersOld[k] = kCenters[k];
		}
		calculateAvgOfClasses(classLabel, image, kCenters, n);
		stop = isNew(n, kCenters, kCentersOld, eps);
		iteration++;
	}/*end while*/

	Mat imageModified = image;
	for (int k = 0; k < n; k++){
		for (int i = 0; i < rows; i++){
			for (int j = 0; j < cols; j++){
				if (classLabel[i][j] == k){
					intensity.val[0] = kCenters[k].b;
					intensity.val[1] = kCenters[k].g;
					intensity.val[2] = kCenters[k].r;
					imageModified.at<Vec3b>(i, j) = intensity;
				}
			}
		}
	}

	namedWindow("Display ModifiedImage", WINDOW_AUTOSIZE);// Create a window for
display.
```

```
	imshow("Display ModifiedImage", imageModified);	// Show our image inside
it.

	waitKey(0);					// Wait for a keystroke in
the window
	return 0;
}

void calculateAvgOfClasses(int *classLabel[], Mat image, RGBcolor kCenters[], int n){
	long avgRed, avgGreen, avgBlue;
	long count;
	uchar red, green, blue;
	Vec3b intensity;
	int cols = image.cols;
	int rows = image.rows;
	for (int k = 0; k < n; k++){
		count = 1;
		avgRed = 0;
		avgGreen = 0;
		avgBlue = 0;
		for (int i = 0; i < rows; i++){
			for (int j = 0; j < cols; j++){
				if (classLabel[i][j] == k){
					intensity = image.at<Vec3b>(i, j);
					blue = intensity.val[0];
					green = intensity.val[1];
					red = intensity.val[2];
					avgRed += red;
					avgGreen += green;
					avgBlue += blue;
					count++;
				}
			}
		}
		avgRed /= count;
		avgGreen /= count;
		avgBlue /= count;
		//printf("k:%d avgRed:%d avgGreen:%d avgBlue:%d count:%d\n", k, avgRed,
avgGreen, avgBlue, count);
		kCenters[k].r = avgRed;
		kCenters[k].g = avgGreen;
		kCenters[k].b = avgBlue;
	}
}

bool isNew(int n, RGBcolor kCenters[], RGBcolor kCentersOld[], float eps){
	int r, g, b;
	int r2, g2, b2;
	float dist;
	for (int i = 0; i < n; i++){
		r = kCentersOld[i].r;
		g = kCentersOld[i].g;
		b = kCentersOld[i].b;
		r2 = kCenters[i].r;
		g2 = kCenters[i].g;
		b2 = kCenters[i].b;
		dist = sqrt((r - r2)*(r - r2) + (g - g2)*(g - g2) + (b - b2)*(b - b2));
		if (dist>eps){
			return false;
		}
	}
	return true;
```

```
}

int calculateDistance(RGBcolor kCenters[], uchar red, uchar green, uchar blue, int n){
	int redDistance;
	int greenDistance;
	int blueDistance;
	float *sampleDistance;
	if ((sampleDistance = (float*)malloc(sizeof(float)*n)) == NULL){
		printf("Cannot allocate memory!\n");
		return -1;
	}

	for (int i = 0; i < n; i++){
		redDistance = (red - kCenters[i].r)*(red - kCenters[i].r);
		greenDistance = (green - kCenters[i].g)*(green - kCenters[i].g);
		blueDistance = (blue - kCenters[i].b)*(blue - kCenters[i].b);
		sampleDistance[i] = sqrt(redDistance + greenDistance + blueDistance);
	}
	int min = 0;
	for (int i = 1; i < n; i++){
		if (sampleDistance[i] < sampleDistance[min]){
			min = i;
		}
	}
	return min;
	}
```