YILDIZ TEKNİK ÜNİVERSİTESİ
ELEKTRİK – ELEKTRONİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

GÖRÜNTÜ İŞLEME ÖDEV 3

Sıddık Said AYDOĞAN
11011052

Doç. Dr. Mine Elif KARSLIGİL

İSTANBUL, 2015

YÖNTEM

Gerçekleştirilen ödevde, Local Binary Pattern(LBP) kullanılarak doku benzerlikleri belirlenmiş ve test edilen resimlerle en benzer 3 tanesi getirilmiştir. Ödev 3 alt parçadan oluşmaktadır.

1. Modül: LBP hesaplaması yapılır ve Histogram alınır. Ödevin ana konusunu oluşturan kısımdır. Burada resim matrisi alınarak kodlama gerçekleştirilir. Binary kodlamasında 2 ve daha az değişime sahipler ayrı ayrı sayılır. Diğerleri aynıymış gibi sayılır.

   LBP Kodlama sol üstten başlayarak saat yönünde yapılır. Yarı kodlama olarak verilecek olursa:

   ```
   for i =1 to imageHeight -1
           for j=1 to imageWidth -1
                   pixel = Image[i,j]
                   if image[i-1,j-1] > pixel then yonler[0]=1 else yönler[0]=0
                   if image[i-1,j] > pixel then yonler[0]=1 else yönler[0]=0

           ...
   ```

   En fazla 58 tane 2 veya daha az değişime sahip kodlama çıkacaktır. HashMap tutularak çıkan kodların tekrarlarında birer artırılması sağlanmıştır. Bu sayede oluşan LBP değerlerinin histogramı alınmış olur. 9 (00001001) için 1-0 ve 0-1 geçişi 2 den fazla olduğu için zaten 0 olacaktı burası 58 in dışında kalanların sayısı için kullanılmıştır.

2. Modül: Eğitim amacıyla yapılmıştır. TRAIN klasörü altındaki resimleri okuyarak LBP hesaplaması yaparak oluşan histogramları liste olarak tutar.
3. Modül: Test amacıyla yapılmıştır. TEST klasörü altındaki resimlerin 1. Modülde belirtilen şekilde LBP histogramını alarak 2.Modül den elde edilmiş histogram listesiyle L1 (Manhattan) uzaklık hesaplaması yapılır. Bu hesaplama yapılırken:

   ```
   for j=1 to trainImageCount
           for i=1 to 59
                   distance += trainImage[i]->histogram[i] – testHistogram[i]
   ```

   şeklinde uzaklık hesaplaması yapılarak en az farka sahip 3 eğitim resmi gösterilir.

UYGULAMA

TÜR	*BAŞARI YÜZDESİ*
Ağaç Kabuğu	%85
Çakıl	%95
Duvar	%90
Tuğla	%70
Kadife	%85
Ekoseli	%80
Döşeme	%90

SONUÇ

Sonuç olarak LBP ile doku benzerliği karşılaştırıldığında belirlenen 7 tür için ortalama %85 başarı elde ettiği görülmüştür. Resimler özellikle belirli bir yöne doğru yönelimli şekiller içerdiğinde başarı arttığı görülmüştür.

Resimler 4 parçaya bölünüp LBP alındığında da sonuçların değişmediği görülmüştür.

KAYNAK KOD

Test CLASS

```
package goruntuIslemeProje;
import static org.opencv.core.Core.*;
import java.util.List;
public class Test {
	public static void main(String[] args) {
		System.loadLibrary(NATIVE_LIBRARY_NAME);
		String workingDir = System.getProperty("user.dir");

		Operation o = new Operation();
		List<String> trainFileNames = o.getFileNames(workingDir + "\\TRAIN\\");
		List<int[]> trainData = o.Train(workingDir + "\\TRAIN\\",
trainFileNames);
		o.Test(workingDir + "\\TEST\\", trainData, trainFileNames);
	}
}
```

Operation CLASS

```
package goruntuIslemeProje;

import static org.opencv.highgui.Highgui.CV_LOAD_IMAGE_GRAYSCALE;
import static org.opencv.highgui.Highgui.imread;

import java.io.File;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;

import org.opencv.core.Mat;

public class Operation {

	public int[] getHistogram(int[][] resim, int width, int height) {

		Map<Integer, Integer> histMap = new HashMap<Integer, Integer>();

		int degisim = 0;
		int[] histogram = new int[256];
		int cokDegisen = 0;
```

```
int pixel = 0;
int[] yonler = new int[8];

for (int i = 1; i < height - 1; i++) {

        for (int j = 1; j < width - 1; j++) {

                degisim = 0;
                pixel = resim[i][j];

                if (resim[i - 1][j - 1] > pixel) {
                        yonler[0] = 1;
                } else {
                        yonler[0] = 0;
                }

                if (resim[i - 1][j] > pixel) {
                        yonler[1] = 1;
                } else {
                        yonler[1] = 0;
                }

                if (resim[i - 1][j + 1] > pixel) {

                        yonler[2] = 1;
                } else {
                        yonler[2] = 0;
                }

                if (resim[i][j + 1] > pixel) {
                        yonler[3] = 1;
                } else {
                        yonler[3] = 0;
                }

                if (resim[i + 1][j + 1] > pixel) {
                        yonler[4] = 1;
                } else {
                        yonler[4] = 0;
                }

                if (resim[i + 1][j] > pixel) {
                        yonler[5] = 1;
                } else {
                        yonler[5] = 0;
                }

                if (resim[i + 1][j - 1] > pixel) {
                        yonler[6] = 1;
```

```
                                } else {
                                        yonler[6] = 0;
                                }

                                if (resim[i][j - 1] > pixel) {
                                        yonler[7] = 1;
                                } else {
                                        yonler[7] = 0;
                                }

                                for (int k = 0; k < 7; k++) {
                                        if (yonler[k] != yonler[k + 1]) {
                                                degisim++;
                                        }
                                }

                                if (degisim < 3) {
                                        int point = yonler[0] * 128 + yonler[1] * 64 +
yonler[2] * 32 + yonler[3] * 16 + yonler[4] * 8 + yonler[5] * 4 + yonler[6] * 2 + yonler[7]
* 1;

                                        // varsa bir artır
                                        if (histMap.containsKey(point))
                                                histMap.put(point, ((int) histMap.get(point))
+ 1);
                                        else
                                                histMap.put(point, 1);

                                } else {
                                        cokDegisen++;
                                }
                        }
                }

                if (histMap.size() > 58)
                        System.out.println("Bi bokluk olabilir debug ediver!");

                for (Entry<Integer, Integer> entry : histMap.entrySet()) {

                        histogram[entry.getKey()] = entry.getValue();
                }

                // 9 zaten 0 olacaktı oraya koyuldu
                histogram[9] = cokDegisen;

                return histogram;

        }
```

```
	public List<String> getFileNames(String directory) {

		List<String> fileNames = new ArrayList<String>();

		File folder = new File(directory);

		File[] listOfFiles = folder.listFiles();

		for (int i = 0; i < listOfFiles.length; i++) {
			if (listOfFiles[i].isFile() &&
listOfFiles[i].getName().endsWith(".jpg")) {
				fileNames.add(listOfFiles[i].getName());
			}
		}

		return fileNames;
	}

	public List<int[]> Train(String directory, List<String> fileNames) {

		int height = 0;
		int width = 0;
		int[][] resim = null;

		List<int[]> histogramlar = new ArrayList<int[]>();

		for (int i = 0; i < fileNames.size(); i++) {
			Mat img = imread(directory + fileNames.get(i),
CV_LOAD_IMAGE_GRAYSCALE);

			height = img.height();
			width = img.width();

			resim = new int[height][width];

			for (int j = 0; j < height; j++) {

				for (int k = 0; k < width; k++) {
					double[] bgrPixel = img.get(j, k);
					resim[j][k] = (int) bgrPixel[0];

				}
			}

			histogramlar.add(getHistogram(resim, width, height));
		}

		return histogramlar;
	}
```

```
	public void Test(String directory, List<int[]> histogramlar, List<String>
trainFileNames) {
		List<String> fileNames = getFileNames(directory);

		int height = 0;
		int width = 0;
		int[][] resim = null;

		for (int r = 0; r < fileNames.size(); r++) {

			Mat img = imread(directory + fileNames.get(r),
CV_LOAD_IMAGE_GRAYSCALE);

			height = img.height();
			width = img.width();

			resim = new int[img.height()][img.width()];

			for (int j = 0; j < img.height(); j++) {

				for (int k = 0; k < img.width(); k++) {
					double[] bgrPixel = img.get(j, k);

					resim[j][k] = (int) bgrPixel[0];
				}
			}

			int fark = 0;

			int[] histogram = getHistogram(resim, width, height);

			Map<String, Integer> farklar = new HashMap<String, Integer>();

			for (int i = 0; i < histogramlar.size(); i++) {
				fark = 0;

				for (int j = 0; j < 256; j++) {
					if (histogram[j] == 0)
						continue;

					fark += Math.abs(histogramlar.get(i)[j] -
histogram[j]);
				}

				farklar.put(String.valueOf(i), fark);

			}
```

```
                        Map<String, Integer> sorted = sortMap(farklar);

                        System.out.print(fileNames.get(r) + " -> ");
                        int i = 0;
                        for (Entry<String, Integer> entry : sorted.entrySet()) {

                                System.out.print(" " +
trainFileNames.get(Integer.parseInt(entry.getKey())));
                                i++;
                                if (i > 2)
                                        break;
                        }

                        System.out.println();
                }

        }

        private Map<String, Integer> sortMap(Map<String, Integer> unsortMap) {

                List<Map.Entry<String, Integer>> list = new
LinkedList<Map.Entry<String, Integer>>(unsortMap.entrySet());

                Collections.sort(list, new Comparator<Map.Entry<String, Integer>>() {
                        public int compare(Map.Entry<String, Integer> o1,
Map.Entry<String, Integer> o2) {
                                return (o1.getValue()).compareTo(o2.getValue());
                        }
                });

                Map<String, Integer> sortedMap = new LinkedHashMap<String,
Integer>();
                for (Iterator<Map.Entry<String, Integer>> it = list.iterator(); it.hasNext();) {
                        Map.Entry<String, Integer> entry = it.next();
                        sortedMap.put(entry.getKey(), entry.getValue());
                }
                return sortedMap;
        }

}
```