INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Ingeniería en Inteligencia Artificial

Practica 9: Segmentación de regiones usando agrupamiento y súper pixeles

Nombre del alumno: Torres López Marco Antonio

Nombre del profesor: Saul de la O Torres

Grupo: 5BM1

Unidad de aprendizaje: Visión artificial

Desarrollo

Algoritmo de SLIC (con código)

Paso 1.

Importamos nuestra imagen a procesar

```
String PATH = "c:/Users/MARCO/Documents/NetBeansProjects/";
String filePath = PATH + "profes.jpg";
Mat img = Imgcodecs.imread(filePath);
```

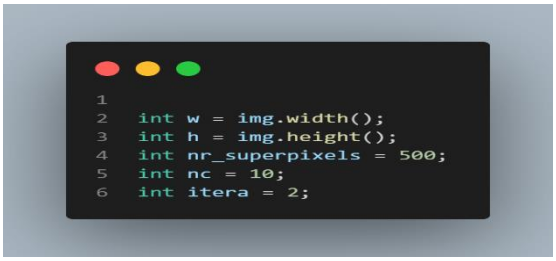
Paso 2.

Convertimos nuestra imagen RBG a Lab

```
Mat lab = new Mat();
Imgproc.cvtColor(img, lab, Imgproc.COLOR_BGR2Lab);
mostrarImagen("lab", lab);
```

Paso 3.

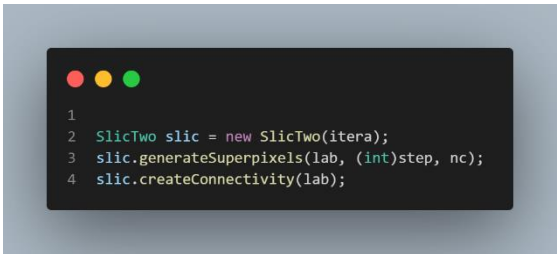
Proporcionamos algunos parámetros sobre la cantidad de super pixeles y los pesos que tendrán



```
int w = img.width();
int h = img.height();
int nr_superpixels = 500;
int nc = 10;
int itera = 2;
```

Paso 4.

Inicializaos nuestra función del algoritmo SLIC con generateSuperpixels



```
SlicTwo slic = new SlicTwo(itera);
slic.generateSuperpixels(lab, (int)step, nc);
slic.createConnectivity(lab);
```

Paso 4.1

Dentro de la función de generateSuperpixeles empieza inicializando los centros de los clusters, borra datos de anteriores iteraciones y se reinician, hará un ciclo For para cada centro de clustes: comparar los píxeles en una región de 2 x pasos por 2 x pasos, calcula la distancia para cada pixel con respecto a cada centro del cluster y se actualiza la asignación de grupos si el grupo minimiza la distancia.

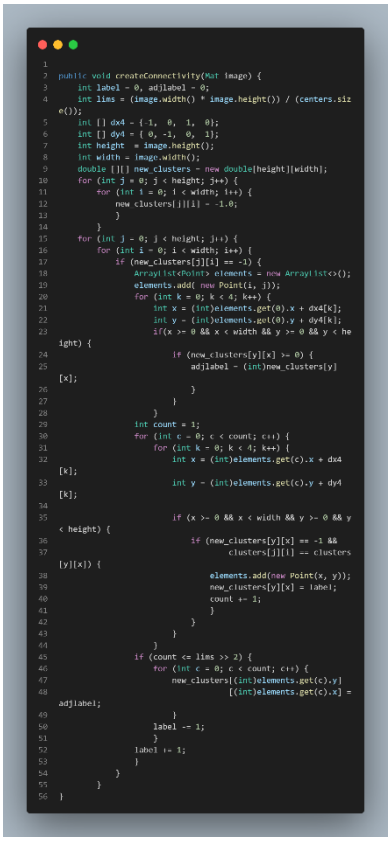
```
for (int j = 0; j < centers.size(); j++) {
            Centro centro = centers.get(j);
            int yIni = centro.getY()-step;
            int yFin = centro.getY()+step;
            int xIni = centro.getX()-step;
            int xFin = centro.getX()+step;
            for(int l=yIni; l<yFin; l++) {
                for( int k=xIni; k<xFin; k++) {
                    if (k >= 0 && k < image.width() &&
                            l >= 0 && l < image.height()) {
                        Scalar colour = new Scalar( image.get(l, k) );
                        double d = computeDistance(j, new Point(k,l), colour);
                        if (d < distances[l][k]) {
                            distances[l][k] = d;
                            clusters[l][k] = j;
                        }
                    }
                }
            }
```

repeat
  / Assignment */*
  for each cluster center C_k **do**
    for each pixel i in a $2S \times 2S$ region around C_k **do**
      Compute the distance D between C_k and i.
      if $D < d(i)$ **then**
        set $d(i) = D$
        set $l(i) = k$
      end if
    end for
    end for
    / Update */*
    Compute new cluster centers.
    Compute residual error E.
until $E \leq$ threshold

Paso 4.2

Después de terminar se borran los valores centrales, se calcula los nuevos centros de conglomerados y se normalizan los clústeres.

```
//Normalizar cada cluster
        for (int j = 0; j < centers.size(); j++) {
            Centro center = centers.get(j);
            double L = center.getL();
            L /= center_counts.get(j);
            center.setL(L);
            double a = center.getA();
            a /= center_counts.get(j);
            center.setA(a);
            double b = center.getB();
            b /= center_counts.get(j);
            center.setB(b);
            int y = center.getY();
            y /= center_counts.get(j);
            center.setY(y);
            int x = center.getX();
            x /= center_counts.get(j);
            center.setX(x);
            centers.remove(j);
            centers.add(j, center);
        }
    }
}
```



```
public void createConnectivity(Mat image) {
    int label = 0, adjlabel = 0;
    int lims = (image.width() * image.height()) / (centers.size());
    int [] dx4 = {-1,  0,  1,  0};
    int [] dy4 = { 0, -1,  0,  1};
    int height  = image.height();
    int width = image.width();
    double [][] new_clusters = new double[height][width];
    for (int j = 0; j < height; j++) {
        for (int i = 0; i < width; i++) {
            new_clusters[j][i] = -1.0;
        }
    }
    for (int j = 0; j < height; j++) {
        for (int i = 0; i < width; i++) {
            if (new_clusters[j][i] == -1) {
                ArrayList<Point> elements = new ArrayList<>();
                elements.add( new Point(i, j));
                for (int k = 0; k < 4; k++) {
                    int x = (int)elements.get(0).x + dx4[k];
                    int y = (int)elements.get(0).y + dy4[k];
                    if(x >= 0 && x < width && y >= 0 && y < height) {
                        if (new_clusters[y][x] >= 0) {
                            adjlabel = (int)new_clusters[y][x];
                        }
                    }
                }
                int count = 1;
                for (int c = 0; c < count; c++) {
                    for (int k = 0; k < 4; k++) {
                        int x = (int)elements.get(c).x + dx4[k];
                        int y = (int)elements.get(c).y + dy4[k];

                        if (x >= 0 && x < width && y >= 0 && y < height) {
                            if (new_clusters[y][x] == -1 &&
                                    clusters[j][i] == clusters[y][x]) {
                                elements.add(new Point(x, y));
                                new_clusters[y][x] = label;
                                count += 1;
                            }
                        }
                    }
                }
                if (count <= lims >> 2) {
                    for (int c = 0; c < count; c++) {
                        new_clusters[(int)elements.get(c).y]
                                    [(int)elements.get(c).x] = adjlabel;
                    }
                    label -= 1;
                }
                label += 1;
            }
        }
    }
}
```

Paso 5.

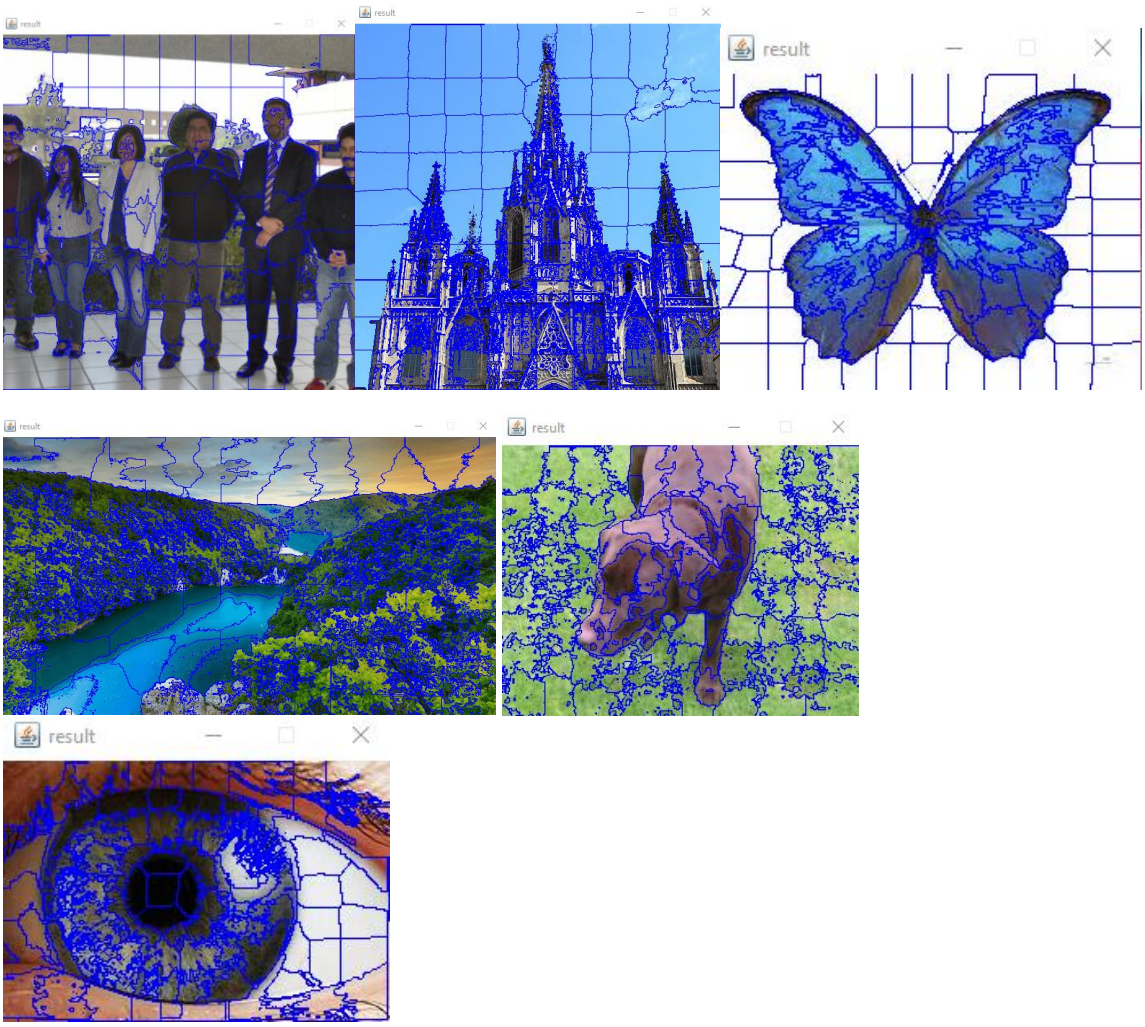
Después de terminar el proceso del algoritmo de SLIC se hará un etiquetado(conectividad) de pixeles, al cúmulo correspondiente se le asigna un valor al centro del cúmulo a los pixeles que resultaron ser parte de este de acuerdo la distancia medida.

Para los pixeles que fueron asignados con un valor del centro del super pixel y que estén fuera del área 2 pasos x 2 pasos abarca el cúmulo de clusters serán reetiquetados (forzar conectividad) asignándoles el valor del cúmulo mayor que se encuentre alrededor de su vecindad.

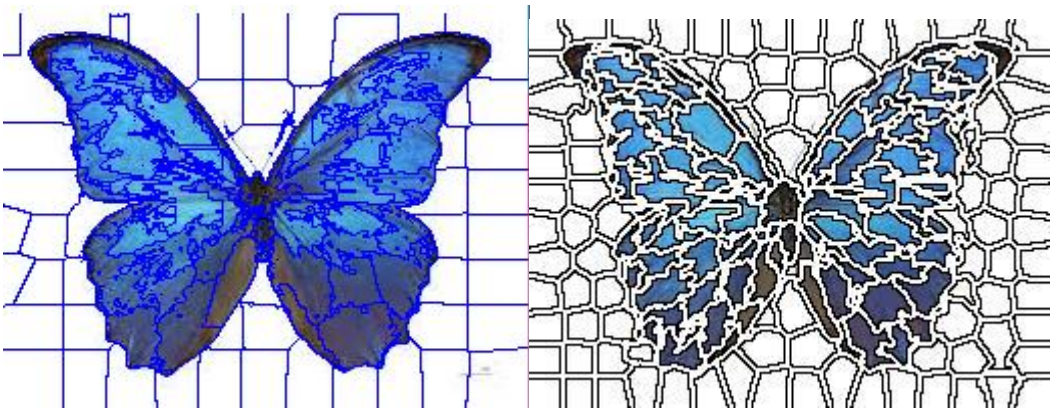
Resultados del algoritmo de SLIC

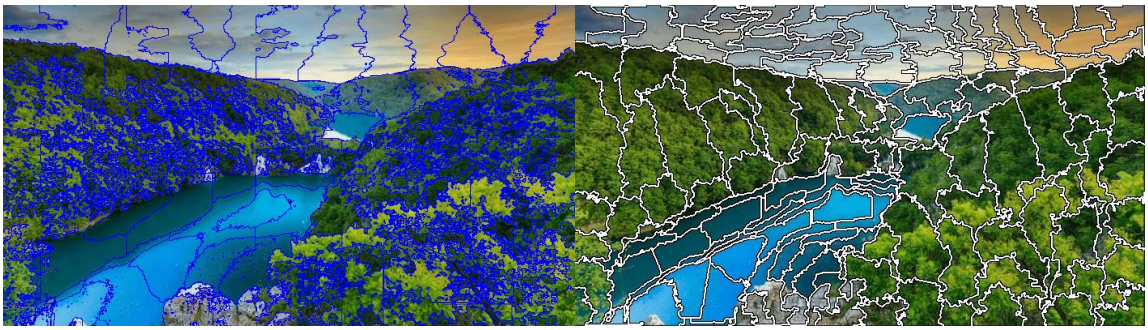
Imágenes de prueba:

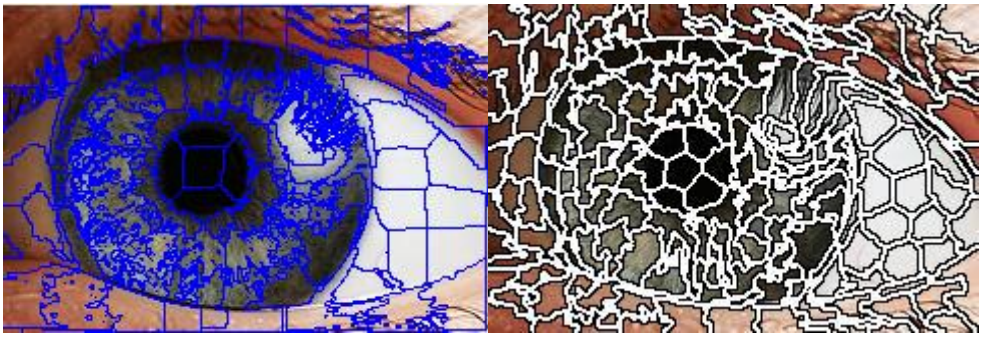
Imágenes en superpixel:



Comparacion entre programa de Java vs SLIC.exe





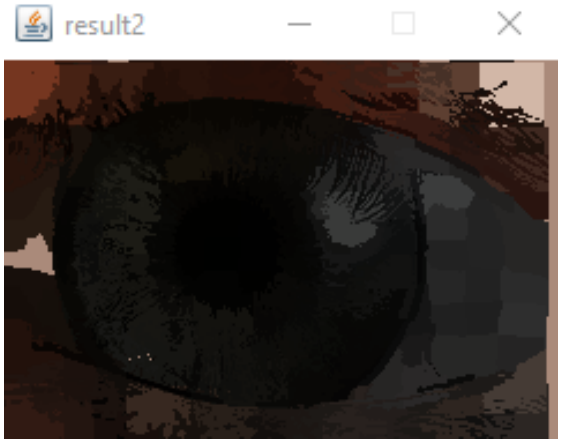


En lugar de poner los límites del super pixel trate de obtener los colores del pixel del centro del cluster (el cluster es un super pixel) e intercambie ese color para todos los pixeles que pertenecen al super pixel. ¿Qué obtuvo como resultado? repórtelos y concluya

```
public void colourWithClusterMeans(Mat image) {

        ArrayList<Scalar> colours = new ArrayList<>(centers.si
ze());

        //Cada color se le inserta el valor del centro
        for (Centro center : centers) {
            colours.add(new Scalar(center.getL(),center.getA
(),center.getB()));
        }

        //Se reunen los valores de color por grupo

        for (int i = 0; i < image.height(); i++) {
            for (int j = 0; j < image.width(); j++) {

                int index = (int)clusters[j][i];
                //System.out.println("index" + index );
                double [] colour = image.get(j, i);

                double a = colours.get(index).val[0];
                a += colour[0];
                //System.out.println(a);
                double b = colours.get(index).val[1];
                b += colour[1];
                double c = colours.get(index).val[2];
                c += colour[2];
                colours.set(index, new Scalar(a, b, c));
                }
            }

        // Divide el numero de pixeles por grupo para obtener
el color medio.
        for (int i = 0; i <colours.size(); i++) {

            double a = colours.get(i).val[0];
            a /= center counts.get(i);
            double b = colours.get(i).val[1];
            b /= center counts.get(i);
            double c = colours.get(i).val[2];
            c /= center counts.get(i);
            colours.set(i, new Scalar(a, b, c));
            }
        //Inserta el color a la nueva imagen
        for (int j = 0; j < image.height(); j++) {
            for (int i = 0; i < image.width(); i++) {
                Scalar mcolour = colours.get((int)clusters[j]
[i]);

                double [] dcolour = {
                    mcolour.val[0],
                    mcolour.val[1],
                    mcolour.val[2]
                    };
                image.put(i, i, dcolour);
                }
            }
    }
}
```

Solo agregamos a nuestro código como paso final una clase para convertir cada superpixel el color que pertenecen, primero cada color se le insterta el valor del centro, luego se reuncen los valores de color por grupo, se divide el numero de pixeles por grupo para tener una media en cada grupo y por ultimo se inserta el color.

Con este método obtuvimos una mejor visualización de los superpixeles y de como secciona cada centro del cluster a diferencia de solo seccionarlo con puros contornos. A continuación de mostraran algunas pruebas:



Conclusión

El algoritmo implementado resulta una alternativa novedosa para la segmentación de imágenes, podemos notar que el valor de K dependerá del tipo de imágenes que se estén trabajando en una determinada aplicación, considerando el número de objetos que existen en la imagen.

Los tiempos de procesamiento dependen totalmente de la velocidad del procesador de la computadora que ejecuta el algoritmo, y que, además, éste se encuentre libre de tareas mientras realiza exclusivamente la ejecución del programa. Aunque la diferencia del programa SLIC.exe lo hace perfectamente sin problemas con mejores resultados, pero la diferencia es que solo ofrece superpixeles con contornos.

Por ultimo podemos decir que el algoritmo nos permite compactar información relevante para después extraerla de manera sencilla, dependiendo de las características del superpixel.