Instituto Politécnico Nacional
Escuela Superior de Cómputo

Práctica 2

Generador del Histograma

Procesamiento Digital de Imágenes
Grupo 4BV1

Alumno: Padilla García Andrea Miranda

Objetivo: Realizar un programa en Java para generar los histogramas de imágenes en escala de grises y RGB

Introducción

Para contextualizar, definamos algunos conceptos.

Un histograma es una representación gráfica del brillo o la luminosidad de cada píxel de la fotografía (bins). Lo que se visualiza en él refiere a la cantidad de pixeles que hay de cada todo, mientras más alto sea, más pixeles de ese tono encontramos en la imagen. La información se distribuye a la largo del histograma del siguiente modo:

Izquierda	Centro	Derecha
Sombras	Medios tonos	Luces
Negro absoluto		Blanco puro

El histograma de una imagen se puede definir como una transformación de dos dimensiones a una dimensión, se representa por:

$$h_i = T[f(x, y)L]$$

Donde $i =$ valor de la intensidad del píxel y $L - 1 =$ límite.

El histograma es una herramienta valiosa utilizada para:

- Conocer el perfil de una imagen
- Obtener una visión de la composición de la imagen
- Proveer de información sobre el contraste de la imagen
- Conocer la distribución de intensidad de una imagen

Ahora bien, el modelo de color YIQ es una codificación para convertir la señal blanco y negro a color y viceversa.

Otro modelo es el HSI que define los colores en función de tres valores muy importantes: matiz (estado puro del color sin blanco o negro agregados), saturación (concentración de color en el objeto) y brillo (que tan claro u oscuro es un color).

Otro modelo de color que emplearemos, además del RGB visto la práctica anterior, es el HSV que es un modelo definido por el tono, saturación y luminosidad, lo que lo hace más a la percepción del color humano.

Por otro lado, debemos aplicar ciertas propiedades a la imagen, entre ellas están:

- **Media:** es el valor medio de los niveles de intensidad, brinda información sobre el brillo global de la imagen. Se calcula como: $\mu = \sum_{i=0}^{255} i \cdot P(i)$
- **Varianza:** este valor indica qué tanto se dispersan los valores alrededor de la media, brinda información sobre el contraste, se calcula como: $\sigma^2 = \sum_{i=0}^{L-1} (i - \mu)^2 \cdot P(i)$

- **Asimetría:** se calcula sobre la media en la distribución de los niveles de gris, se calcula como: $a = \sum_{i=0}^{L-1}(i-\mu)^3 \cdot P(i)$
- **Energía:** brinda información sobre la distribución de las intensidades, se calcula como: $E = \sum_{i=0}^{L-1} P^2(i)$
- **Entropía:** brinda información sobre la distribución de los niveles de intensidad, su cálculo se realiza con: $e = -\sum_{i=0}^{L-1} P(i)\, log_2[P(i)]$

Además de lo anterior, hay que calcular la probabilidad de ocurrencia de cada uno de los niveles de intensidad que aparecen en la imagen, para calcularla se emplea la fórmula:

$$P(i) = \frac{N(i)}{T_P}$$

Donde:

$P(i)$ es la probabilidad de aparición del píxel (i)

$N(i)$ es el numero total de apariciones del píxel

T_P es el total de píxeles en la imagen (ancho x alto)

Y, por último, se debe calcular la densidad de probabilidad, para ello, se realiza la sumatoria de $P(i)$ cuyo resultado nos debe dar 1.

Desarrollo

Para realizar lo solicitado en la práctica, inicialmente hay que extraer los canales RGB y hacer la conversión a niveles de gris, para ello, decidí utilizar Y de YIQ para hacerlo se multiplican los valores RGB por ciertos valores determinados, como se muestra en el código:

```
private void calcularHistogramas() {
        int width = image.getWidth();  //ancho
        int height = image.getHeight();  //alto
        totalPixels = width * height;
        imageGray = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
        BufferedImage imageR = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
        BufferedImage imageG = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
        BufferedImage imageB = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);

        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                Color c = new Color(image.getRGB(x, y));
                int r = c.getRed();
                int g = c.getGreen();
                int b = c.getBlue();
                histR[r]++;
                histG[g]++;
                histB[b]++;
                int Y = (int)(0.299*r + 0.587*g + 0.114*b);  //obtener Y de YIQ para gris
                if (Y < 0) Y = 0; if (Y > 255) Y = 255;   // limitar a 0 o 255 si se
                                                            desbordan los valores
                histGray[Y]++;
                int grayRGB = (Y<<16)|(Y<<8)|Y;
                imageGray.setRGB(x, y, grayRGB);
                imageR.setRGB(x, y, new Color(r,0,0).getRGB());
                imageG.setRGB(x, y, new Color(0,g,0).getRGB());
                imageB.setRGB(x, y, new Color(0,0,b).getRGB());
            }
        }
        // reescalar todas las imágenes
        int scaledWidth = Math.min(200, image.getWidth());
        int scaledHeight = (int) ((double) scaledWidth / image.getWidth() *
        image.getHeight());
        scaledImageGray = scaleImage(imageGray, scaledWidth, scaledHeight);
        scaledImageR = scaleImage(imageR, scaledWidth, scaledHeight);
        scaledImageG = scaleImage(imageG, scaledWidth, scaledHeight);
        scaledImageB = scaleImage(imageB, scaledWidth, scaledHeight);
    }
```

Reescalamos las imágenes para que se logran visualizar en la pantalla.

Para mostrar las imágenes mostrar las imágenes de cada canal, se emplea:

```
// Panel de imágenes
JPanel multiImagePanel = new JPanel(new GridLayout(1, 5, 10, 10));
multiImagePanel.add(new JLabel(new ImageIcon(scaledImage)));     //imagen original
multiImagePanel.add(new JLabel(new ImageIcon(scaledImageGray))); //gris
multiImagePanel.add(new JLabel(new ImageIcon(scaledImageR)));    //canal R
multiImagePanel.add(new JLabel(new ImageIcon(scaledImageG)));    //canal G
multiImagePanel.add(new JLabel(new ImageIcon(scaledImageB)));    //canal B
```

En el entendido de que el primer histograma calculado y mostrado es una tabla de frecuencias que indica cuántos píxeles hay de cada tonalidad, se implementa en calcularHistogramas, donde se recorren los píxeles de la imagen y guarda en un arreglo las frecuencias por tonalidad.

Para mostrar los histogramas:

```
  private void mostrarHistogramasRGB() {
      histogramPanel.removeAll();
      histogramPanel.setLayout(new GridLayout(1, 3, 10, 10));
      histogramPanel.add(new GraphPanel(histR, "Histograma Rojo", Color.RED, mediaR, true));
      histogramPanel.add(new GraphPanel(histG, "Histograma Verde", Color.GREEN, mediaG,
  true));
      histogramPanel.add(new GraphPanel(histB, "Histograma Azul", Color.BLUE, mediaB, true));
      histogramPanel.revalidate();
      histogramPanel.repaint();
  }

  private void mostrarHistogramasGris() {
      histogramPanel.removeAll();
      histogramPanel.setLayout(new GridLayout(1, 1));
      histogramPanel.add(new GraphPanel(histGray, "Histograma Gris", Color.GRAY, mediaGray,
  true));
      histogramPanel.revalidate();
      histogramPanel.repaint();
  }
```

Para las propiedades de la imagen, en código se implementan:

```
private double calcMedia(double[] prob) {
        double m=0;
        for (int i=0;i<256;i++) m += i*prob[i];
        return m;
    }
    private double calcVarianza(double[] prob,double m) {
        double v=0;
        for (int i=0;i<256;i++) v+= Math.pow(i-m,2)*prob[i]; return v;
    }
    private double calcAsimetria(double[] prob,double m) {
        double a=0; for (int i=0;i<256;i++) a+= Math.pow(i-m,3)*prob[i]; return a;
    }
    private double calcEnergia(double[] prob) {
        double e=0; for (int i=0;i<256;i++) e+= Math.pow(prob[i],2); return e;
    }
    private double calcEntropia(double[] prob) {
        double h=0; for (int i=0;i<256;i++) if (prob[i]>0) h+=
        prob[i]*(Math.log(prob[i])/Math.log(2));
        return -h;
    }
```

Para crear las propiedades de cada canal:

```
private void calcularPropiedades() {
        mediaR = calcMedia(probR);
        mediaG = calcMedia(probG);
        mediaB = calcMedia(probB);
        mediaGray = calcMedia(probGray);

        varR = calcVarianza(probR, mediaR);
        varG = calcVarianza(probG, mediaG);
        varB = calcVarianza(probB, mediaB);
        varGray = calcVarianza(probGray, mediaGray);

        asimGray = calcAsimetria(probGray, mediaGray);

        energiaR = calcEnergia(probR);
        energiaG = calcEnergia(probG);
        energiaB = calcEnergia(probB);
        energiaGray = calcEnergia(probGray);

        entropiaR = calcEntropia(probR);
        entropiaG = calcEntropia(probG);
        entropiaB = calcEntropia(probB);
        entropiaGray = calcEntropia(probGray);
    }
```

La probabilidad y la densidad de probabilidad se implementan de la siguiente manera:

```
private void calcularProbabilidades() {
   for (int i = 0; i < 256; i++) {
        probR[i] = (double) histR[i] / totalPixels;
        probG[i] = (double) histG[i] / totalPixels;
        probB[i] = (double) histB[i] / totalPixels;
        probGray[i] = (double) histGray[i] / totalPixels;
    }
}

private void calcularDensidadProbabilidad() {
    dpR[0] = probR[0];
    dpG[0] = probG[0];
    dpB[0] = probB[0];
    dpGray[0] = probGray[0];
    for (int i = 1; i < 256; i++) {
       dpR[i] = dpR[i-1] + probR[i];
       dpG[i] = dpG[i-1] + probG[i];
       dpB[i] = dpB[i-1] + probB[i];
       dpGray[i] = dpGray[i-1] + probGray[i];
     }
}
```

```
//mostrar propiedades
    private void mostrarPropiedadesRGB() {
        propPanel.removeAll();
        propPanel.setLayout(new GridLayout(2, 2, 10, 10));
        propPanel.add(new BarChartPanel(new double[]{varR, varG, varB}, "Varianza (RGB)",
        new String[]{"R","G","B"}));
        propPanel.add(new BarChartPanel(new double[]{energiaR, energiaG, energiaB},
        "Energía (RGB)", new String[]{"R","G","B"}));
        propPanel.add(new BarChartPanel(new double[]{entropiaR, entropiaG, entropiaB},
       "Entropía (RGB)", new String[]{"R","G","B"}));
        propPanel.revalidate();
        propPanel.repaint();
    }
    private void mostrarPropiedadesGris() {
        propPanel.removeAll();
        propPanel.setLayout(new GridLayout(2, 2, 10, 10));
        propPanel.add(new BarChartPanel(new double[]{varGray}, "Varianza (Gris)", new
        String[]{"Gray"}));
        propPanel.add(new BarChartPanel(new double[]{asimGray}, "Asimetría (Gris)", new
        String[]{"Gray"}));
        propPanel.add(new BarChartPanel(new double[]{energiaGray}, "Energía (Gris)", new
        String[]{"Gray"}));
        propPanel.add(new BarChartPanel(new double[]{entropiaGray}, "Entropía (Gris)", new
        String[]{"Gray"}));
        propPanel.revalidate();
        propPanel.repaint();
    }

//mostrar probabilidades
    private void mostrarProbabilidadesRGB() {
        probPanel.removeAll();
        probPanel.setLayout(new GridLayout(1, 3, 10, 10));
        probPanel.add(new GraphPanel(probR, "Probabilidad Rojo", Color.RED, mediaR, true));
        probPanel.add(new GraphPanel(probG, "Probabilidad Verde", Color.GREEN, mediaG,
        true));
        probPanel.add(new GraphPanel(probB, "Probabilidad Azul", Color.BLUE, mediaB,
        true));
        probPanel.revalidate();
        probPanel.repaint();
    }
    private void mostrarProbabilidadesGris() {
        probPanel.removeAll();
        probPanel.setLayout(new GridLayout(1, 1));
        probPanel.add(new GraphPanel(probGray, "Probabilidad Gris", Color.GRAY, mediaGray,
        true));
        probPanel.revalidate();
        probPanel.repaint();
    }
```

```
//mostrar densidades
private void mostrarDensidadRGB() {
    dpPanel.removeAll();
    dpPanel.setLayout(new GridLayout(1, 3, 10, 10));
    dpPanel.add(new GraphPanel(dpR, "Densidad Probabilidad Rojo", Color.RED, mediaR,
    false));
    dpPanel.add(new GraphPanel(dpG, "Densidad Probabilidad Verde", Color.GREEN, mediaG,
    false));
    dpPanel.add(new GraphPanel(dpB, "Densidad Probabilidad Azul", Color.BLUE, mediaB,
    false));
    dpPanel.revalidate();
    dpPanel.repaint();
}
private void mostrarDensidadGris() {
    dpPanel.removeAll();
    dpPanel.setLayout(new GridLayout(1, 1));
    dpPanel.add(new GraphPanel(dpGray, "Densidad Probabilidad Gris", Color.GRAY,
    mediaGray, false));
    dpPanel.revalidate();
    dpPanel.repaint();
}
```

Para realizar la interfaz gráfica, se implementan pestañas para separar donde mostrar los histogramas, la probabilidad, la densidad y las propiedades, se realiza:

```
class GraphPanel extends JPanel {
        private double[] data;
        private String title;
        private Color color;
        private double media;
        private boolean verMedia;

        public GraphPanel(int[] data,String title,Color color,double media,boolean
        verMedia){
            this.data=new double[data.length];
            for(int i=0;i<data.length;i++) this.data[i]=data[i];
            this.title=title; this.color=color; this.media=media; this.verMedia=verMedia;
        }
        public GraphPanel(double[] data,String title,Color color,double media,boolean
        verMedia){
            this.data=data; this.title=title; this.color=color; this.media=media;
            this.verMedia=verMedia;
        }

        @Override
        protected void paintComponent(Graphics g){
            super.paintComponent(g);
            Graphics2D g2=(Graphics2D)g;
            int width=getWidth(), height=getHeight(), margin=50;
            double max=0; for(double v:data) if(v>max) max=v;
            int graphWidth=width-2*margin, graphHeight=height-2*margin;
            g2.setColor(Color.BLACK);
            g2.drawLine(margin,height-margin,width-margin,height-margin);
            g2.drawLine(margin,margin,margin,height-margin);

            // Eje X
            for(int i=0;i<=255;i+=51){
                int x=margin+(i*graphWidth/256);
                g2.drawLine(x,height-margin,x,height-margin+5);
                g2.drawString(Integer.toString(i),x-10,height-margin+20);
            }

            // Eje Y (0, mitad, máximo)
            g2.drawString("0", margin-25,height-margin);
            g2.drawString(String.format("%.2f",max/2), margin-40,
            height-margin-(graphHeight/2));
            g2.drawString(String.format("%.2f",max), margin-40, margin+10);

            g2.drawString(title,width/2-50,margin-15);
```

```
        // Curva
        g2.setColor(new Color(color.getRed(),color.getGreen(),color.getBlue(),100));
        Polygon poly=new Polygon();
        poly.addPoint(margin,height-margin);
        for(int i=0;i<256;i++){
            int x=margin+(i*graphWidth/256);
            int y=height-margin-(int)((data[i]/max)*graphHeight);
            poly.addPoint(x,y);
        }
        poly.addPoint(margin+graphWidth,height-margin);
        g2.fillPolygon(poly);

        g2.setColor(color);
        for(int i=0;i<255;i++){
            int x1=margin+(i*graphWidth/256);
            int y1=height-margin-(int)((data[i]/max)*graphHeight);
            int x2=margin+((i+1)*graphWidth/256);
            int y2=height-margin-(int)((data[i+1]/max)*graphHeight);
            g2.drawLine(x1,y1,x2,y2);
        }
        //linea de la media
        if(verMedia){
            int Media=margin+(int)((media*graphWidth)/256);
            g2.setColor(Color.MAGENTA);
            g2.drawLine(Media,margin,Media,height-margin);
            g2.drawString("Media="+String.format("%.2f",media),Media+5,margin+15);
        }
    }
}
```

```
class BarChartPanel extends JPanel {
        private double[] values;
        private String title;
        private String[] labels;

        public BarChartPanel(double[] values,String title,String[] labels){
            this.values=values; this.title=title; this.labels=labels;
        }

        @Override
        protected void paintComponent(Graphics g){
            super.paintComponent(g);
            Graphics2D g2=(Graphics2D)g;
            int width=getWidth(), height=getHeight(), margin=50;
            double maxAbs=0;
            for(double v:values) maxAbs = Math.max(maxAbs, Math.abs(v));
            int graphWidth=width-2*margin, graphHeight=height-2*margin;
            int barWidth=graphWidth/values.length-20;
            g2.setColor(Color.BLACK);
            g2.drawLine(margin,height-margin,width-margin,height-margin);
            g2.drawLine(margin,margin,margin,height-margin);
            g2.drawString(title,width/2-40,margin-15);


          // Eje Y (0, mitad, máximo)
          g2.drawString("0", margin-25,height-margin);
          g2.drawString(String.format("%.2f",maxAbs/2), margin-40,
          height-margin-(graphHeight/2));
          g2.drawString(String.format("%.2f",maxAbs), margin-40, margin+10);

          Color[] barColors={Color.RED,Color.GREEN,Color.BLUE,Color.GRAY};
          for(int i=0;i<values.length;i++){
              int barHeight=(int)((Math.abs(values[i])/maxAbs)*graphHeight);
              int x=margin+i*(barWidth+20);
              int y=height-margin-barHeight;
              g2.setColor(barColors[i%barColors.length]);
              g2.fillRect(x,y,barWidth,barHeight);
              g2.setColor(Color.BLACK);
              g2.drawRect(x,y,barWidth,barHeight);
              g2.drawString(labels[i],x+barWidth/2-10,height-margin+20);
              g2.drawString(String.format("%.2f",values[i]),x+barWidth/2-15,y-5);
          }
      }
  }
```

Conclusión

Puedo concluir que realizar esta práctica fue sencillo hablando de implementar las propiedades mencionadas previamente, así como la obtención del histograma (que yo creería sería más lioso), lo que más tiempo me ha llevado fue acomodar las pestañas de un modo amigable y entendible. No está de más añadir que viendo los histogramas de distintas imágenes que subí como ejemplo para probar el programa logré entender la importancia de tener un grafico para describir las propiedades de la imagen, en especial si se sabe interpretar nos brinda información sobre la composición de la imagen.

Referencias

Fosado, L. H. (s. f.). *Modelos de color YIQ y HSI*. Scribd. https://es.scribd.com/document/233380295/Modelos-de-Color-Yiq-y-Hsi

León, N. (s. f.). *Histograma RGB: Cuando el Histograma Convencional no es 100% Fiable*. Dzoom. https://www.dzoom.org.es/histograma-rgb/

Trost, S. (s. f.). *Modelos de color: HSV*. https://es.sttmedia.com/modelo-de-color-hsv

El histograma y estadísticas de imagen. https://drive.google.com/file/d/1v8YvdXr9UFH9sOQZPu2GQJyb0wk8igNm/view