1.7. Binarización de imágenes en escala de grises

Como usted sabe una imagen digital está compuesta de pixeles que si la imagen es en color pues el píxel es RGB y los valores de los tres componentes están entre 0 y 255, así una imagen en niveles de gris tiene una paleta de colores que comienza con los componentes R=0, G=0, B=0 este implicaría negro y todos se incrementan en una unidad para dar un total de 255 tonalidades de gris, es decir, los componentes llegarán a R=255, G=255 y B=255 lo que implica tener un blanco. Para binarizar una imagen se supone que solo tenga dos valores, como su nombre lo implica binario, pero podemos tomar al cero y al uno (0, 1), eso es binario, pero si hiciéramos eso, con la paleta de niveles de gris, la imagen resultante estaría muy oscura, por lo que el valor de uno (1) se deberá tomar como 255, es decir, solo tomaremos dos niveles de gris negro y blanco. Aún queda la pregunta ¿Cómo decidir que pixeles son convertidos en negro (R=0, G=0, B=0) y cuales de los pixeles se convierten en blanco (R=255, G=255, ¿B=255)?

Para contestar la pregunta anterior, existe un atributo para juzgar que pixeles se convierten en negro y que pixeles se convierten en blancos, ese atributo se conoce como threshold (en inglés) cuya traducción podría ser límite, sujetador, umbral, etc. Este valor se selecciona arbitrariamente de tal manera que la información contenida en su imagen resultante sea inteligible para el usuario, es decir que los objetos que están contenidos en la imagen estén ahí y no desaparezcan dejando una imagen muy oscura o una imagen muy blanqueada. Lo anterior se logra aplicando la siguiente ecuación

$$f_b(x,y) = \begin{cases} 0 & si\ f_g(x,y) \leq umbral \\ 255 & si\ f_g(x,y) > umbral \end{cases} \tag{1.62}$$

Donde $f_b(x,y)$ es la imagen binarizada y $f_g(x,y)$ es la imagen en niveles de gris, por lo tanto, si se quiere binarizar una imagen se puede ver el proceso en la figura 1.27.

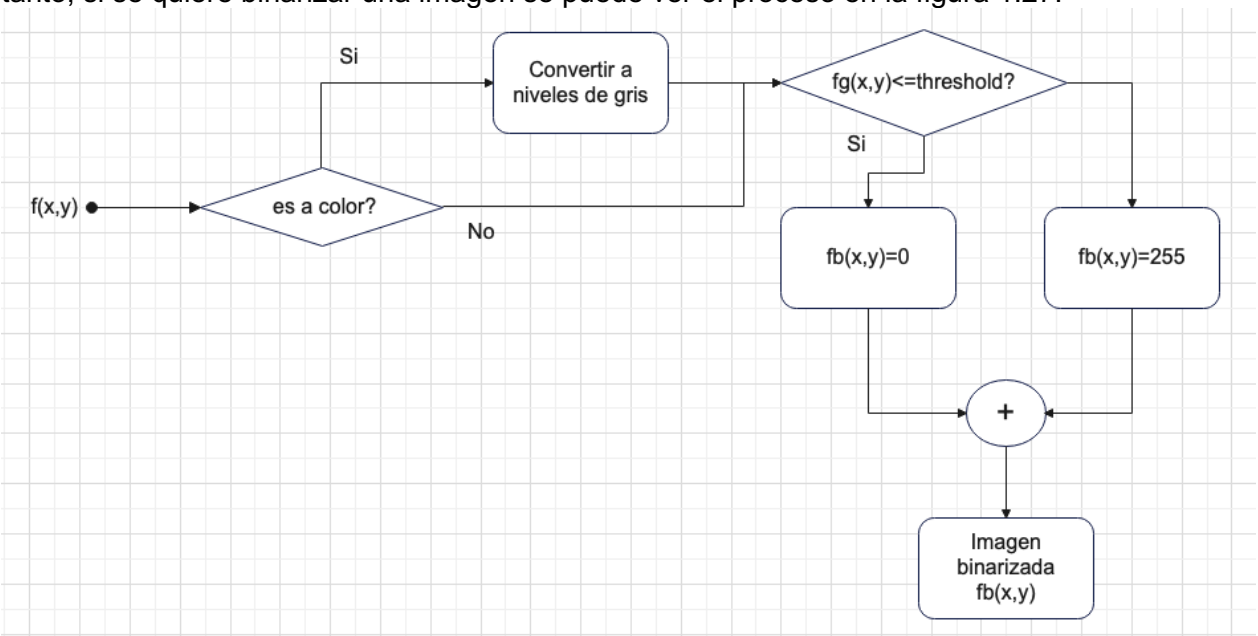


Figura 1.27 Diagrama a bloques de la binarización de una imagen

Para realizar este procesamiento de la imagen vamos a emplear la clase ImagenBinaria que se muestra en la figura 1.28, en la figura 1.29 se muestra la clase con la que se visualiza el

proceso de binarizado FrameImagenBinaria y en la figura 1.30 el resultado final. La clase para binarizar recibe en su constructor una matriz de enteros que contiene los pixeles de la imagen, tiene un método binarizarImagen() al cual se le envía un entero que es el thresholding para que el método realiza bien su trabajo, en este método en las líneas 50, 51 y 52 usted puede emplear cualquiera de ellas, ya que con ellas se extraen cualquiera de los componentes rojo, verde y azul respectivamente, pero como se trata de una imagen en niveles de gris, todos ellos tienen el mismo valor, por ello puede utilizar el que sea. Por último cuenta con un método getImagen() que devuelve un objeto de tipo Image para visualizar en pantalla el resultado de la imagen binarizada.

En la clase FrameImagenBinaria se emplean tres paneles de imagen para mostrar la imagen en color, la imagen en niveles de gris y la imagen binarizada.

```
1  package unidaduno;
2
3  import java.awt.Color;
4  import java.awt.Image;
5  import java.awt.image.MemoryImageSource;
6  import javax.swing.JFrame;
7
8  /**
9   *
10  * @author sdelaot
11  */
12 public class ImagenBinaria {
13     /**
14      * Color negro
15      */
16     private final int NEGRO = 0;
17     /**
18      * Color clanco
19      */
20     private final int BLANCO = 255;
21     /**
22      * La imagen en niveles de gris
23      */
25     private int [][] imagenGris;
26     /**
27      * La imagen binaria
28      */
29     private int [][] imagenBinaria;
30     /**
31      * El constructor
32      *
33      * @param imagenGris recibe la imagen en niveles de gris
34      */
35     public ImagenBinaria(int [][] imagenGris) {
36         this.imagenGris = imagenGris;
     }
```

```
37      /**
38       * Metodo para binarizar la imagen
39       *
40       * @param threshold el limite para saber que pixeles son cero y cuales son
41       * doscientos cincuenta y cinco.
42       */
43      public void binarizarImagen(int threshold) {
44          int alto = imagenGris.length;
45          int ancho = imagenGris[0].length;
46          imagenBinaria = new int[alto][ancho];
47          Color color;
48          int pixel;
49          for(int y=0; y<alto; y++) {
50              for(int x=0; x<ancho; x++) {
51                  pixel = (imagenGris[y][x] & 0x00ff0000) >> 16;
52                  //pixel = (imagenGris[y][x] & 0x0000ff00) >> 8;
53                  //pixel = imagenGris[y][x] & 0x000000ff;
54                  //System.out.println("GRIS " + pixel);
55                  if(pixel<=threshold) {
56                      imagenBinaria[y][x] = NEGRO;
57                      }
58                  else {
59                      imagenBinaria[y][x] = BLANCO;
60                      }
61                  }
62              }
63      }
64      /**
65       * Metodo que devuelve la imagen binaria convertida en objeto Image
66       *
67       * @return devuelve un objeto Image
68       */
69      public Image getImagenBinaria() {
            Image imagen = null;
71          int alto = imagenGris.length;
72          int ancho = imagenGris[0].length;
73          Color color;
74          int pixel;
75          for(int y=0; y<alto; y++) {
76              for(int x=0; x<ancho; x++) {
77                  pixel = imagenBinaria[y][x];
78                  color = new Color(pixel, pixel, pixel);
79                  imagenBinaria[y][x] = color.getRGB();
80                  }
81              }
82          JFrame frameTmp = new JFrame();
83          imagen = frameTmp.createImage(new MemoryImageSource(ancho,
84                  alto, convertirInt2DAInt1D(imagenBinaria, ancho, alto),
85                  0, ancho));
86          return imagen;
87      }
```

```java
    /**
     * Convierte un buffer de tipo double y de dos dimensiones de imagen a uno
     * de una dimension de tipo entero.
     *
     * @param matriz la matriz a convertir
     * @param ancho ancho de la imagen en pixeles
     * @param alto alto de la imagen en pixeles
     *
     * @return el vector de la image convertida
     */
    public int [] convertirInt2DAInt1D(int[][] matriz, int ancho, int alto) {
        int index = 0;
        int [] bufferInt = null;
        try {
            bufferInt = new int[ancho * alto];
            for(int y = 0; y < alto; y++) {
                for(int x=0; x < ancho; x++) {
                    bufferInt[index++] = matriz[y][x];
                    }
                }
        } catch (NegativeArraySizeException e) {
            System.out.println(" Error alto, ancho o ambos negativos"
                + " en  convierteInt2DAInt1D( double [][] ) "
                + e);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println(" Error desbordamiento en bufferInt"
                + " en  convierteInt2DAInt1D( double [][] ) "
                + e);
        } catch (NullPointerException e) {
            System.out.println(" Error bufferInt nulo"
                + " en  convierteInt2DAInt1D( double [][] ) "
                + e);
        }
        return bufferInt;
    }
}
```

Figura 1.28 Clase ImagenBinaria.

```java
package vista;

import control.ControlBinario;
import java.awt.Container;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import javax.swing.JFrame;

/**
 *
 * @author sdelaot
 */
public class FrameImagenBinaria extends JFrame {
    private PanelDeImagen panelRGB;
    private PanelDeImagen panelGris;
    private PanelDeImagen panelBinario;
    private ControlBinario controlBinario;
    public FrameImagenBinaria(String nombreArchivo) {
        super("Visor de imagen");
        initComponents(nombreArchivo);
    }
    private void initComponents(String nombreArchivo) {
        Container contenedor = this.getContentPane();
        contenedor.setLayout(null);
        controlBinario = new ControlBinario(nombreArchivo, 4);
        panelRGB = new PanelDeImagen(controlBinario.getImagen(5));
        panelRGB.setBounds(0, 0,
                           controlBinario.getAncho(),
                           controlBinario.getAlto());
        contenedor.add(panelRGB);
        panelGris = new PanelDeImagen(controlBinario.getImagen(4));
        panelGris.setBounds(controlBinario.getAncho(), 0,
                           controlBinario.getAncho(),
                           controlBinario.getAlto());
        contenedor.add(panelGris);
        panelBinario = new PanelDeImagen(controlBinario.getImagenBinaria(128));
        panelBinario.setBounds(controlBinario.getAncho()*2, 0,
                           controlBinario.getAncho(),
                           controlBinario.getAlto());
        contenedor.add(panelBinario);
        this.setSize(controlBinario.getAncho()*3, controlBinario.getAlto()+40);
        this.setVisible(true);
        this.addWindowListener(new SalidaFrame());
    }
    private class SalidaFrame extends Object implements WindowListener {
        @Override
        public void windowOpened(WindowEvent e) { }
        @Override
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
        @Override
        public void windowClosed(WindowEvent e) {
            System.exit(0);
        }
        @Override
        public void windowIconified(WindowEvent e) { }
        @Override
        public void windowDeiconified(WindowEvent e) { }
        @Override
        public void windowActivated(WindowEvent e) { }
        @Override
        public void windowDeactivated(WindowEvent e) { }
    }
}
```

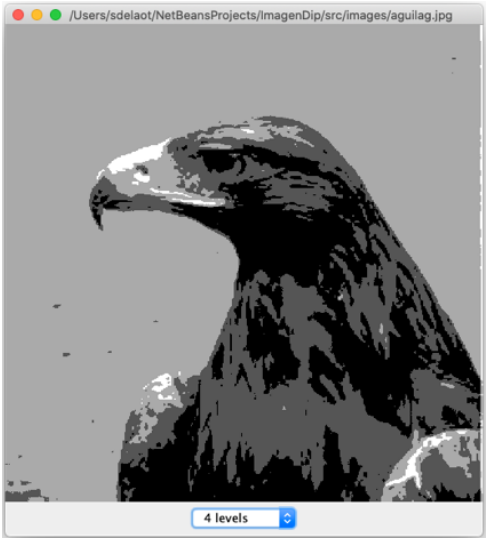
Figura 1.29 Clase FrameImagenBinaria.

Figura 1.30 Proceso de binarizado de una imagen en color.

Es habitual digitalizar los valores de la función imagen, f(x, y), además de su coordenada espacial. Este proceso de cuantificación consiste en reemplazar una variación continua en f(x, y) con un conjunto discreto de niveles de cuantificación. La precisión con la que las variaciones en f(x, y) están representados y vienen determinados por el número de niveles de cuantificación que utilizamos; cuanto más niveles que usamos, mejor será la aproximación. Convencionalmente, un conjunto de *n* niveles de cuantificación comprende los números enteros 0, 1, 2, ..., *n* - 1. 0 y *n* - 1 generalmente se muestran o se imprimen en blanco y negro, respectivamente, con niveles prestados en varios tonos de gris. Por lo tanto, los niveles de cuantificación son comúnmente denominados niveles de gris. El término colectivo para todos los niveles de gris, que van del negro al blanco, es una escala de grises. Para un procesamiento conveniente y eficiente por computadora, el número de niveles de gris, *n*, es generalmente una potencia entera de dos. Podemos escribir

$$n = 2^b \tag{1.63}$$

donde *b* representa el número de bits utilizados para la cuantificación, b es típicamente ocho (8), dándonos imágenes con doscientos cincuenta y seis (256) posibles niveles de gris que van desde 0 (negro) a 255 (blanco), esto se muestra en la figura 1.31.



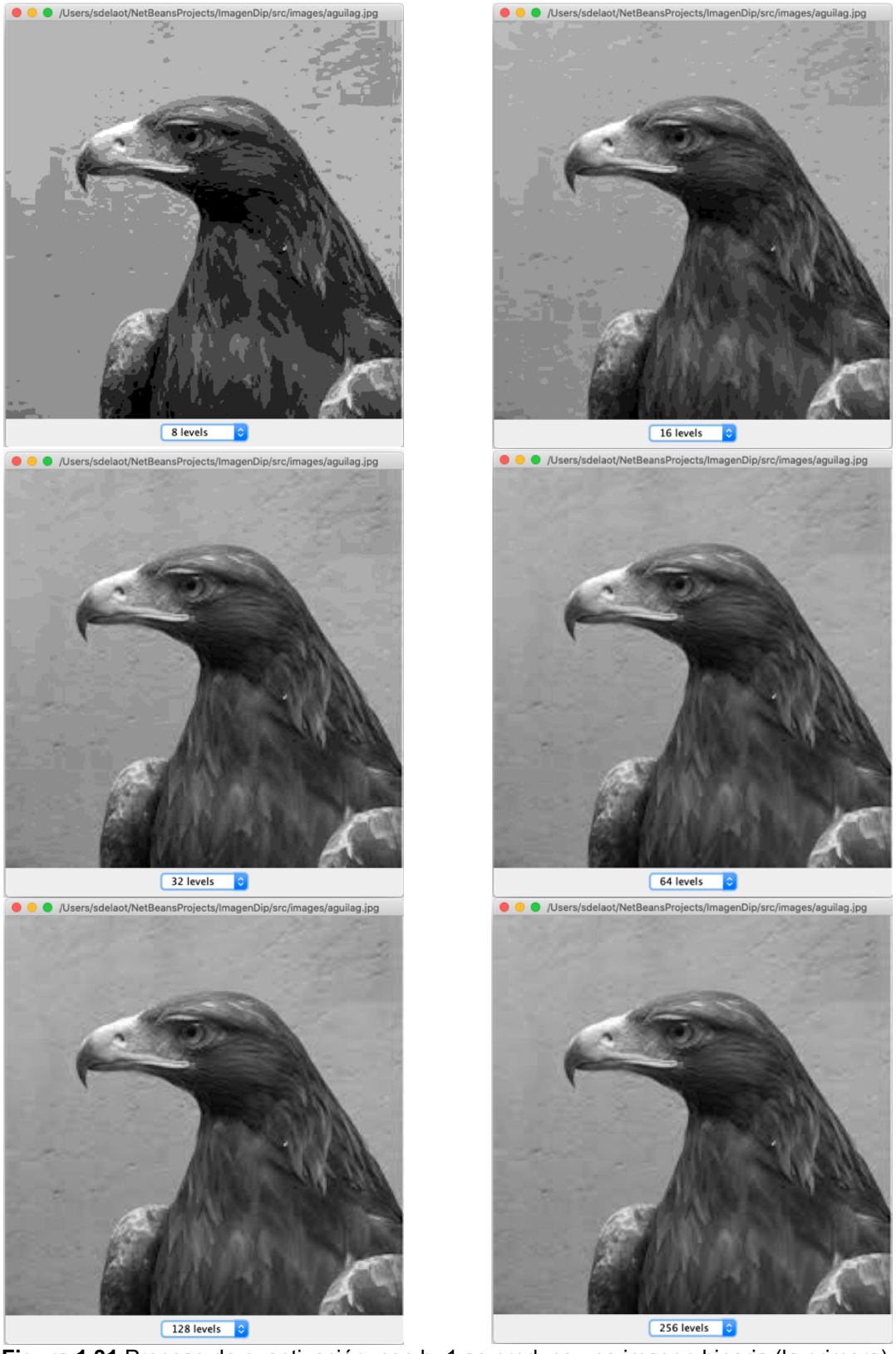
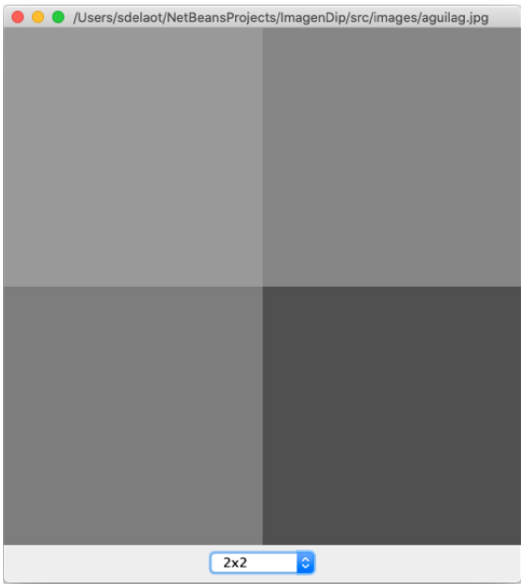


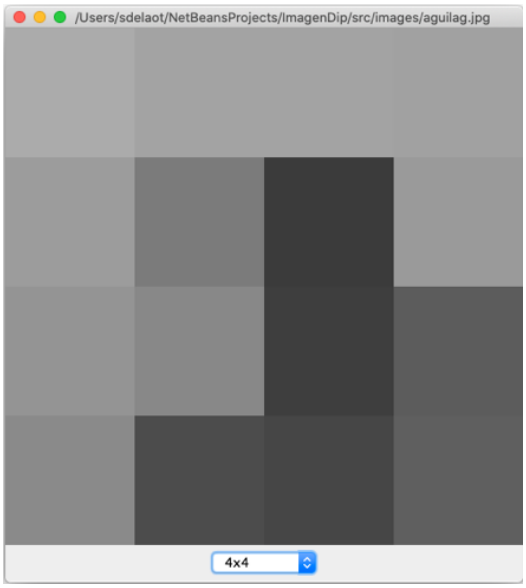
Figura 1.31 Proceso de cuantización, con b=1 se produce una imagen binaria (la primera).

Resolución Espacial

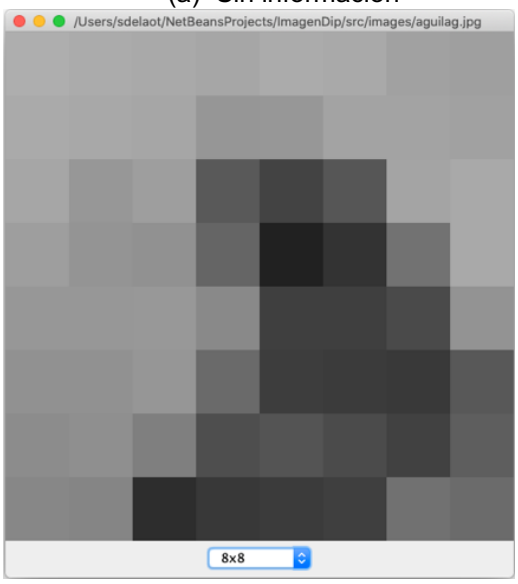
Se puede mostrar imágenes a escala o cambiando la resolución espacial. La resolución espacial de una imagen es el tamaño de los píxeles de la imagen, es decir, el área de visualización representada por un solo píxel de la imagen. Los patrones secos producen imágenes de alta resolución que contienen múltiples píxeles para un área, donde cada píxel representa el área funcional más pequeña; por otro lado, las imágenes de baja resolución tienen píxeles más pequeños, siendo cada punto una parte más grande de la imagen. El procesamiento espacial depende de cuánta información útil se puede extraer de la imagen. La Figura 1.32 muestra esta sección junto con un diagrama que muestra las distintas opciones.



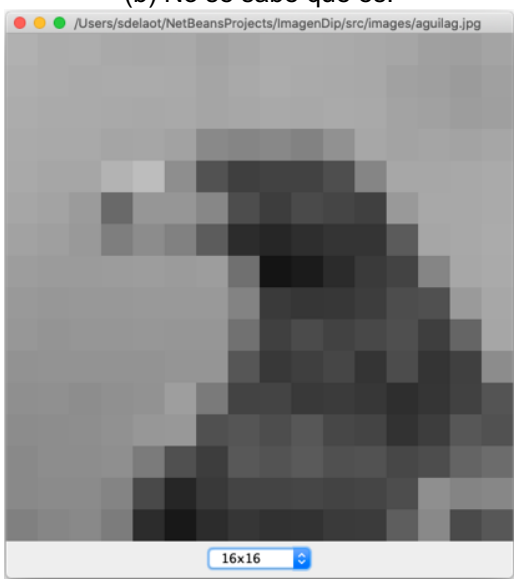
(a) Sin información



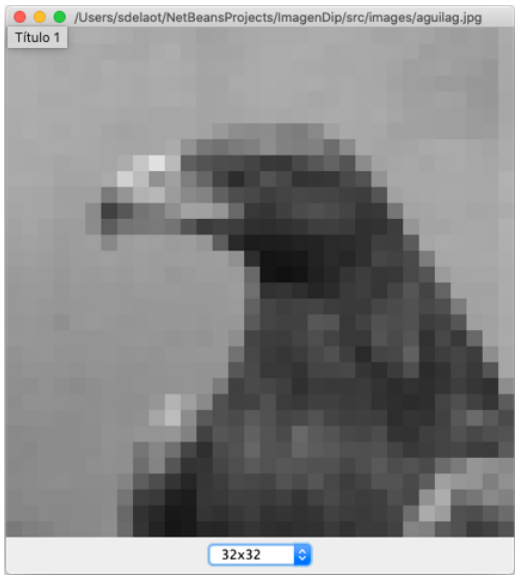
(b) No se sabe que es.



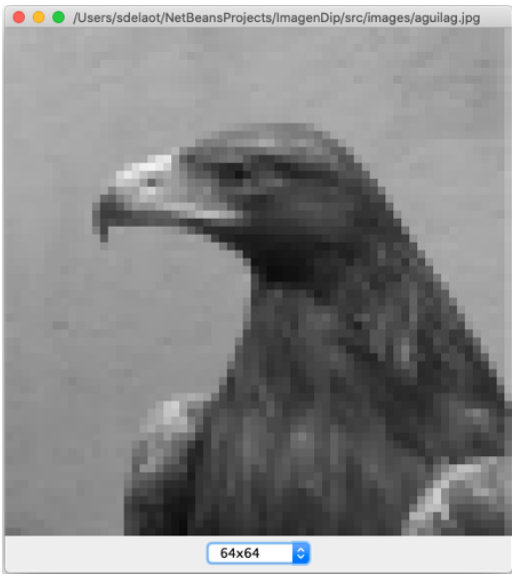
(c) Aun sin información



(d) Va apareciendo la figura



(e) Mayor información



(f) Ya hay más información.

(g) Doble de información

(h) Información completa

Figura 1.32 Resolución de una imagen representada por su número de pixeles.