Face Detection using Haar Cascades

1. Haar Cascades

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, “Rapid Object Detection using a Boosted Cascade of Simple Features” published in Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2001). It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images.

OpenCV already contains many pre-trained classifiers for face, eyes, smiles, etc. Those XML files are stored in

/usr/local/Cellar/opencv/4.5.3_2/share/opencv4/haarcascades

folder as shown in Figure 1.

Note: In the latest version, "4.5.3_2" may be another number.

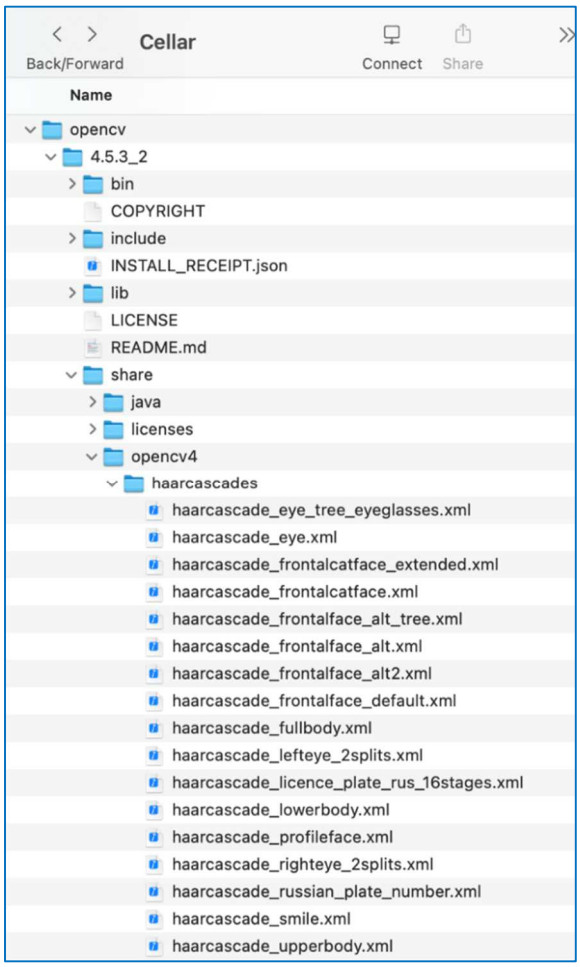


Figure 1. Classifiers provided by OpenCV4.

2. Face Detection

The **CascadeClassifier** class of the package **org.opencv.objdetect** is used to load the classifier file. The constructor and the methods of CascadeClassifier class are shown in Table 1 and Table 2, respectively. Table 3 shows the constructor and the main method of the **MatOfRect** class.

Table 1. The constructors of the CascadeClassifier class.

Constructor of CascadeClassifier
CascadeClassifier() Makes a object.
CascadeClassifier(java.lang.String *filename*) Loads a classifier from a file ***filename***.

Table 2. The main methods provided by the CascadeClassifier class.

Method & Description of CascadeClassifier
void detectMultiScale(Mat *image*, MatOfRect *objects*) Detects objects of different sizes in the input image. The detected objects are returned as a list of rectangles. ***image***: Matrix containing an image where objects are detected. ***objects***: Vector of rectangles where each rectangle contains the detected object.
boolean load(java.lang.String *filename*) Loads a classifier from a file ***filename***.

Table 3. The constructor and the main method of the MatOfRect

Constructor of MatOfRect
MatOfRect() Makes a object.
Method & Description of MatOfRect
Rect[] toArray() Converts into Rect type array.

Figure 2 shows the program "Cascade1.java" for detecting face area. Figure 3 shows the result of "Cascade1.java". The two faces are found.

The following variables are declared in the field (lines 17 to 22).

- ✓ BufferedImage variable **bimage1**
- ✓ Mat variable **src**
- ✓ String variable **filename** for the name of the file “imageB.jpg” to be loaded
- ✓ String variable **path** for the path to the folder where the classifiers are located
  In the latest version of OpenCV, "4.5.3_2" has been changed to another number.
- ✓ String variable **cascade** that is assigned the name of the xml file indicating the classifier
- ✓ Rect array **faceArea** that is for records face areas
  See Chapter 6 of "Introduction to OpenCV" (The thirteenth day) for how to get the value from the Rect class.

The native library of OpenCV should be loaded using the LoadLibrary method of System class to use OpenCV package (line 25).

In the paint method, the size of "imageB.jpg" is 640x480, so it will be displayed in the same size as the loaded image. The yellow rectangles indicating the face area are displayed. In the for statement of lines 39 to 42, yellow rectangles are drawn for all items **rect** of **faceArea**.

Cascade1.java

```
1  /*
2   * Face detection using haarcascade
3   * by Hitoshi Ogawa on January 8th, 2022
4   */
5  import java.awt.Color;
6  import java.awt.Graphics;
7  import java.awt.image.BufferedImage;
8  import javax.swing.JFrame;
9  import org.opencv.core.Core;
10 import org.opencv.core.Mat;
11 import org.opencv.core.MatOfRect;
12 import org.opencv.core.Rect;
13 import org.opencv.imgcodecs.Imgcodecs;
14 import org.opencv.objdetect.CascadeClassifier;
15
16 public class Cascade1 extends JFrame {
17     BufferedImage bimage1;
18     Mat src;
19     String filename = "imageB.jpg";
20     String path ="/usr/local/Cellar/opencv/4.5.3_2/share/opencv4/haarcascades/";
21     String cascade = "haarcascade_frontalface_default.xml";
22     Rect[] faceArea = null;
23
24     public Cascade1() {
25         System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
26         setSize(660, 520);
27         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
28         setVisible(true);
29         detect();
30         repaint();
31     }
32
```

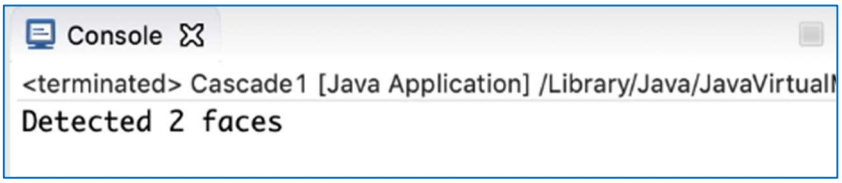
Figure 2. The program “Cascade1.java” for detecting face area.

```
33      public void paint(Graphics g){
34          if(bimage1 != null){
35              g.drawImage(bimage1, 10, 30, this);
36          }
37          if(faceArea != null) {
38              g.setColor(Color.yellow);
39              for(Rect rect : faceArea) {
40                  g.drawRect(rect.x + 10, rect.y + 30, rect.width, rect.height);
41              }
42          }
43      }
44
45      void detect(){
46          src = new Mat();
47          src = Imgcodecs.imread(filename);
48          bimage1 = matToBufferedImage(src);
49
50          CascadeClassifier faceDetector = new CascadeClassifier(path + cascade);
51          MatOfRect faceDetections = new MatOfRect();
52          faceDetector.detectMultiScale(src, faceDetections);
53          System.out.println("Detected "
54                              + faceDetections.toArray().length + " faces");
55          faceArea = faceDetections.toArray();
56      }
57
58
59      public static void main(String[] args) {
60          new Cascade1();
61      }
62
63      public static BufferedImage matToBufferedImage(Mat matrix) {
64          int cols = matrix.cols();
65          int rows = matrix.rows();
66          int elemSize = (int)matrix.elemSize();
67          byte[] data = new byte[cols * rows * elemSize];
68          int type;
69
70          matrix.get(0, 0, data);
71
72          switch (matrix.channels()) {
73          case 1:
74              type = BufferedImage.TYPE_BYTE_GRAY;
75              break;
76          case 3:
77              type = BufferedImage.TYPE_3BYTE_BGR;
78              byte b;
79              for(int i=0; i<data.length; i=i+3) {
80                  b = data[i];
81                  data[i] = data[i+2];
82                  data[i+2] = b;
83              }
84              break;
85          default:
86              return null;
87          }
88          BufferedImage out = new BufferedImage(cols, rows, type);
89          out.getRaster().setDataElements(0, 0, cols, rows, data);
90          return out;
91      }
92  }
```

Figure 2. The program “Cascade1.java” for detecting face area (Continue).

In the detect method, face areas are detected. The loaded image is assigned to the variable **src** (line 47), and **src** is converted to BufferedImage using the matToBufferedImage method (line 48). The variable **faceDetector** is substituted for the object of the CascadeClassifier class using the xml file "haarcascade_frontalface_default.xml" (line 50). The variable **faceDetections** is substituted for the object of MatOfRect class to store the detected faces (line 51). The faces in the image are detected using detectMultiScale method of CascadeClassifier class (line 52). Finally, Rect type array faceArea is obtained using toArray method of MatOfRect class (line 55).

Figure 3 shows the result of “Cascade1.java”. The two faces are found.



(a)

(b)

Figure 3. The result of “Cascade1.java”.
(a) The output to console, (b) Image display.

3. Effectiveness of Classifiers

In Cascade1, "haarcascade_frontalface_default.xml" is used as a classifier. However, OpenCV provides 17 classifiers. The program “Cascade2.java” is introduced to investigate the effectiveness of the classifier on the person’s face. Figure 4 shows the program source of “Cascade2.java”. Figure 5 (a) shows the numbers of extracted regions in the order in which the classifiers were applied. Figure 5 (b) shows the result of “Cascade2.java”.

In the field, there are prepared four xml files “cascade*N*” and arrays “faceArea*N*” of Rects that record facial regions acquired by them (*N* = 1, 2, 3, 4). The cascade1 and cascade2 are classifiers for eye region extraction. Since the rectangles are drawn with the colors shown in Table 4, the areas recognized by the classifiers are compared.

Table 4. Color used for each classifier

Color	Classifier
Color.***yellow***	haarcascade_eye_tree_eyeglasses.xml
Color.***red***	haarcascade_eye.xml
Color.***green***	haarcascade_frontalcatface_extended.xml
Color.***blue***	haarcascade_frontalcatface.xml

Cascade2.java

```
1  /*
2   * Comparing four cascades for face detection
3   * by Hitoshi Ogawa on January 8th, 2022
4   */
5  import java.awt.Color;
6  import java.awt.Graphics;
7  import java.awt.image.BufferedImage;
8  import javax.swing.JFrame;
9  import org.opencv.core.Core;
10 import org.opencv.core.Mat;
11 import org.opencv.core.MatOfRect;
12 import org.opencv.core.Rect;
13 import org.opencv.imgcodecs.Imgcodecs;
14 import org.opencv.imgproc.Imgproc;
15 import org.opencv.objdetect.CascadeClassifier;
16
17 public class Cascade2 extends JFrame {
18     BufferedImage bimage1;
19     Mat src;
20     String filename = "imageB.jpg";
21     String path ="/usr/local/Cellar/opencv/4.5.3_2/share/Opencv4/haarcascades/";
22     String cascade1 = "haarcascade_eye_tree_eyeglasses.xml";
23     Rect[] faceArea1 = null;
24     String cascade2 = "haarcascade_eye.xml";
25     Rect[] faceArea2 = null;
26     String cascade3 = "haarcascade_frontalcatface_extended.xml";
27     Rect[] faceArea3 = null;
28     String cascade4 = "haarcascade_frontalcatface.xml";
29     Rect[] faceArea4 = null;
30
31     public Cascade2() {
32         System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
33         setSize(660, 520);
34         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
35         setVisible(true);
36         detect();
37         repaint();
38     }
```

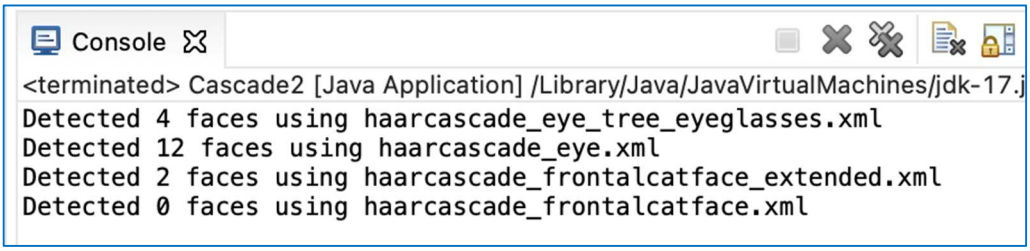
Figure 4. The program “Cascade2.java” for evaluating classifiers.

```
39
40      public void paint(Graphics g){
41          if(bimage1 != null){
42              g.drawImage(bimage1,  10,  30, this);
43          }
44          if(faceArea1 != null) {
45              g.setColor(Color.yellow);
46              for(Rect rect : faceArea1) {
47                  g.drawRect(rect.x + 10, rect.y + 30, rect.width, rect.height);
48              }
49          }
50          if(faceArea2 != null) {
51              g.setColor(Color.red);
52              for(Rect rect : faceArea2) {
53                  g.drawRect(rect.x + 10, rect.y + 30, rect.width, rect.height);
54              }
55          }
56          if(faceArea3 != null) {
57              g.setColor(Color.green);
58              for(Rect rect : faceArea3) {
59                  g.drawRect(rect.x + 10, rect.y + 30, rect.width, rect.height);
60              }
61          }
62          if(faceArea4 != null) {
63              g.setColor(Color.blue);
64              for(Rect rect : faceArea4) {
65                  g.drawRect(rect.x + 10, rect.y + 30, rect.width, rect.height);
66              }
67          }
68      }
69
70      void detect(){
71          src = new Mat();
72          src = Imgcodecs.imread(filename);
73          bimage1 = matToBufferedImage(src);
74
75          faceArea1 = getArea(cascade1);
76          faceArea2 = getArea(cascade2);
77          faceArea3 = getArea(cascade3);
78          faceArea4 = getArea(cascade4);
79      }
80
81      Rect[] getArea(String str) {
82          Rect[] area = null;
83          CascadeClassifier faceDetector = new CascadeClassifier(path + str);
84          MatOfRect faceDetections = new MatOfRect();
85          faceDetector.detectMultiScale(src, faceDetections);
86          System.out.println("Detected "
87                  + faceDetections.toArray().length + " faces using " + str);
88          area = faceDetections.toArray();
89          return area;
90      }
91
92      public static void main(String[] args) {
93          new Cascade2();
94      }
95
96      public static BufferedImage matToBufferedImage(Mat matrix) {
```

/* Omitted */

```
124     }
125 }
```

Figure 4. The program “Cascade2.java” for evaluating classifiers (Continue).



(a)

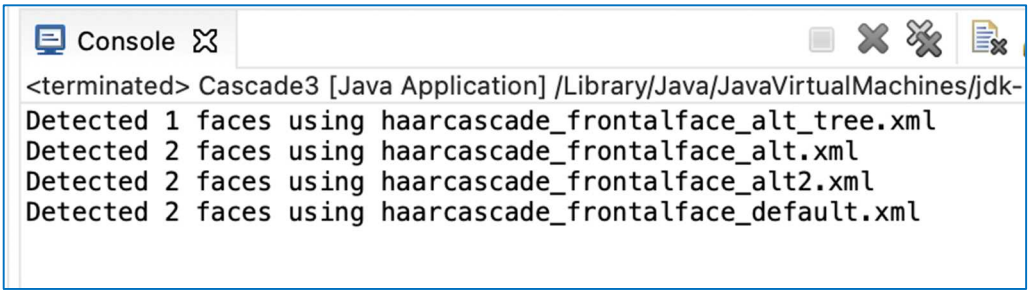
(b)

Figure 5. The result of "Cascade2.java".
(a) The output to console, (b) Image display.

Figure 6 (a) shows the extracted number. Figure 6 (b) shows the results in the case of using the classifier shown in Table 5.

Table 5. Color used for each classifier

Color	Classifier
Color.***yellow***	haarcascade_frontalface_alt_tree.xml
Color.***red***	haarcascade_frontalface_alt.xml
Color.***green***	haarcascade_frontalface_alt2.xml
Color.***blue***	haarcascade_frontalface_default.xml



(a)

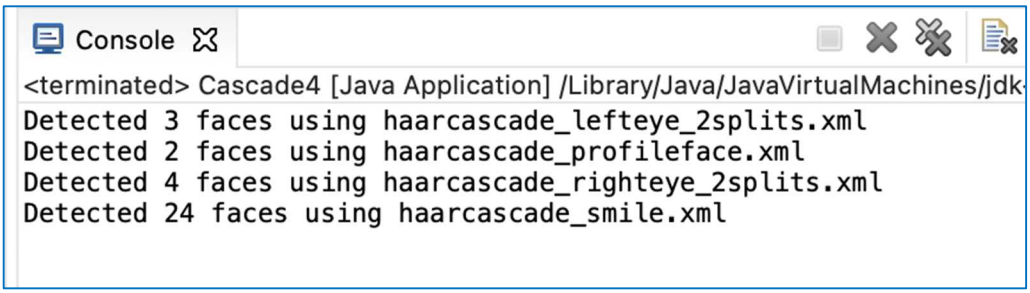
(b)

Figure 6. The result of face detection using Table 5.
(a) The output to console, (b) Image display.

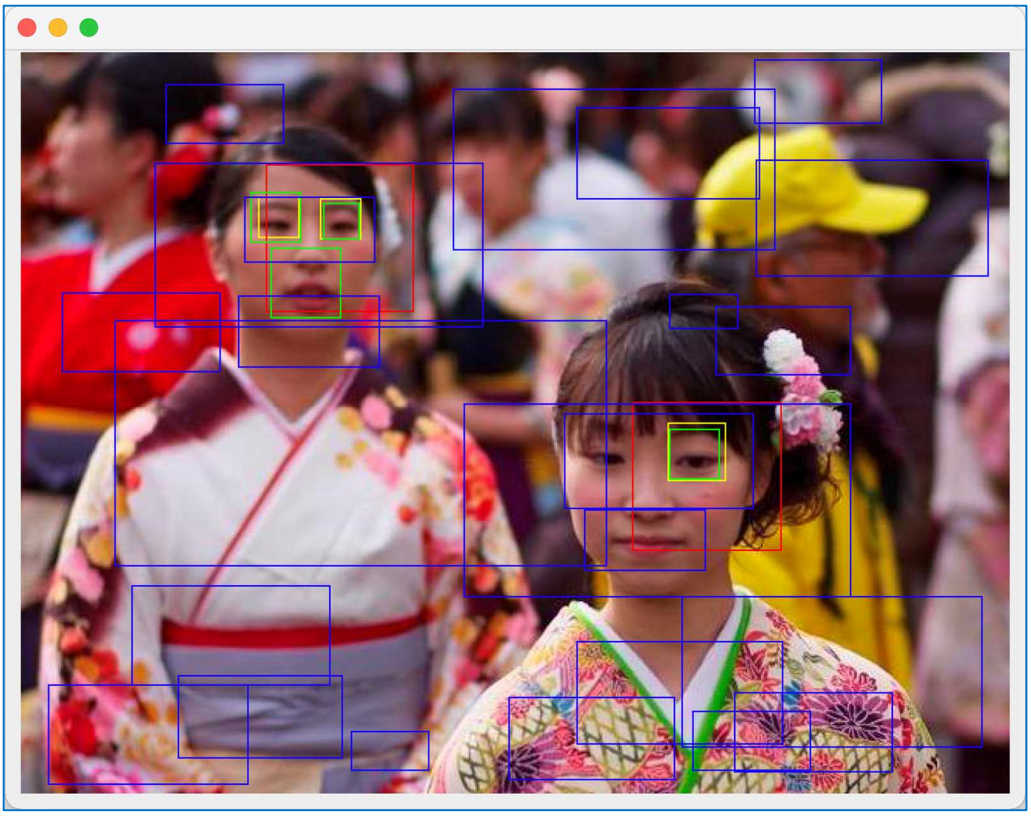
Figure 7 (a) shows the extracted number. Figure 7 (b) shows the results in the case of using the classifier shown in Table 6.

Table 6. Color used for each classifier

Color	Classifier
Color.***yellow***	haarcascade_lefteye_2splits.xml
Color.***red***	haarcascade_profileface.xml
Color.***green***	haarcascade_righteye_2splits.xml
Color.***blue***	haarcascade_smile.xml



(a)



(b)

Figure 7. The result of face detection using Table 6.
(a) The output to console, (b) Image display.

Comparing the regions extracted by each cascade, the following four cascades seem to be usable for extracting facial regions.

- ✓ haarcascade_frontalcatface_extended.xml
- ✓ haarcascade_frontalface_alt.xml
- ✓ haarcascade_frontalface_alt2.xml
- ✓ haarcascade_frontalface_default.xml

4. Extraction of Face Area

Figure 8 shows a program “FaceExtraction.java” to cut out and display the first identified face area.

Program “FaceExtraction.java” is almost the same as “CascadeN.java” (N = 1, 2, 3, 4), but there are the following differences.

- ✓ BufferedImage array “bimage” is prepared for recording faces in line 18.
- ✓ Mat variable “part” is for face extraction in line 19.
- ✓ The method “paint” is extended with the display of the array bimage in lines 44 to 46.
- ✓ The method “detect” is added the following function.
  - ➢ The maximum number of faces to display is five in lines 57 o 59.
  - ➢ The face images are cut out from the original image and recorded in the bimage array in lines 61 to 64.

FaceExtraction.java

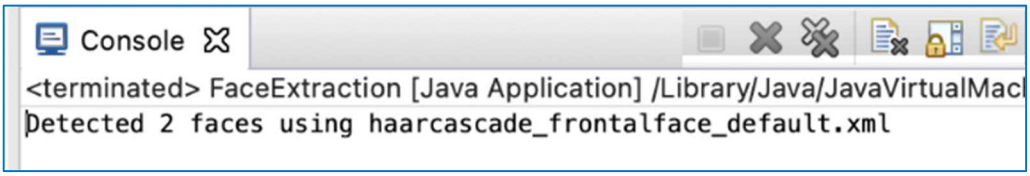
```
1  /*
2   * Extraction of face area
3   * by Hitoshi Ogawa on January 8th, 2022
4   */
5  import java.awt.Color;
6  import java.awt.Graphics;
7  import java.awt.image.BufferedImage;
8  import javax.swing.JFrame;
9  import org.opencv.core.Core;
10 import org.opencv.core.Mat;
11 import org.opencv.core.MatOfRect;
12 import org.opencv.core.Rect;
13 import org.opencv.imgcodecs.Imgcodecs;
14 import org.opencv.objdetect.CascadeClassifier;
15
16 public class FaceExtraction extends JFrame {
17     BufferedImage bimage1;
18     BufferedImage bimage[];
19     Mat src, part;
20     String filename = "imageB.jpg";
21     String path ="/usr/local/Cellar/opencv/4.5.3_2/share/Opencv4/haarcascades/";
22     String cascade = "haarcascade_frontalface_default.xml";
23     Rect[] faceArea = null;
24     int count = 0;
25
26     public FaceExtraction() {
27         System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
28         setSize(660, 670);
29         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
30         setVisible(true);
31         detect();
32         repaint();
33     }
34
35     public void paint(Graphics g){
36         if(bimage1 != null){
37             g.drawImage(bimage1,  10,  30, this);
38         }
39         if(faceArea != null) {
40             g.setColor(Color.yellow);
41             for(Rect rect : faceArea) {
42                 g.drawRect(rect.x + 10, rect.y + 30, rect.width, rect.height);
43             }
44             for(int i = 0; i < count; i++) {
45                 g.drawImage(bimage[i], 10 + 150 * i, 520, this);
46             }
47         }
48     }
49
```

Figure 8. The program “FaceExtraction.java” for displaying face area.

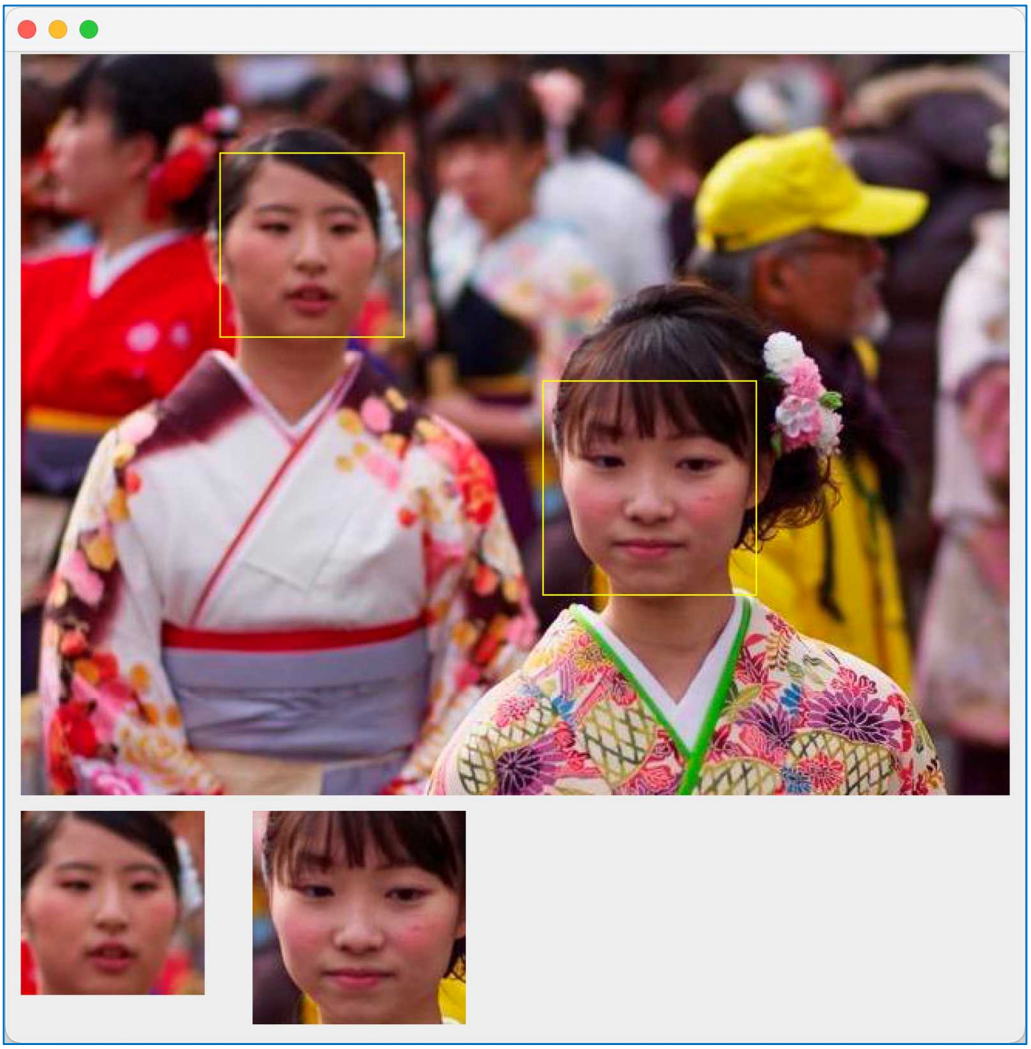
```
50      void detect(){
51          src = new Mat();
52          src = Imgcodecs.imread(filename);
53          bimage1 = matToBufferedImage(src);
54          faceArea = getArea(cascade);
55          if(faceArea != null) {
56              count = faceArea.length;
57              if(count > 5) {
58                  count = 5;
59              }
60              bimage = new BufferedImage[count];
61              for(int i = 0; i < count; i++) {
62                  part = new Mat(src, faceArea[i]);
63                  bimage[i] = matToBufferedImage(part);
64              }
65          }
66      }
67
68      Rect[] getArea(String str) {
69          Rect[] area = null;
70          CascadeClassifier faceDetector = new CascadeClassifier(path + str);
71          MatOfRect faceDetections = new MatOfRect();
72          faceDetector.detectMultiScale(src, faceDetections);
73          System.out.println("Detected "
74                  + faceDetections.toArray().length + " faces using " + str);
75          area = faceDetections.toArray();
76          return area;
77      }
78
79      public static void main(String[] args) {
80          new FaceExtraction();
81      }
82
83      public static BufferedImage matToBufferedImage(Mat matrix) {

                                /* Omitted */

111     }
112 }
```

Figure 8. The program “FaceExtraction.java” for displaying face area. (continue).

Figure 9 shows the result of “FaceExtraction.java”. The face images are displayed below the original image.



(a)



(b)

Figure 9. The result of “FaceExtraction.java”.

(a) The output to console,

(b) Execution example of “FaceExtraction.java”.