Memoria proyecto de prácticas de Biometría

Proyecto de reconocimiento de huellas dactilares

Biometría y Seguridad de Sistemas 2019-2020

Autor: Gonzalo Bueno Rodríguez
DNI: 80105317-C

Descripción de los pasos y algoritmos utilizados

1.- Convertir imagen RGB a matriz de grises

Para poder trabajar con la imagen que será un archivo en nuestro equipo primero debemos pasarlo a BufferedImage leyéndola con la función de lectura de ImageIO de Java, previamente le hemos pedido al usuario el nombre de la imagen o la ruta completa, con esto llamamos a nuestro método de lectura para pasarlo a BufferedImage y una vez pasado llamamos a nuestro método de pasar BufferedImage a una matriz de enteros con nuestra nueva BufferedImage para poder recorrerla pixel a pixel, calcular su nivel de gris a partir de cada componente RGB y el color negro con una operación lógica AND y calculando la media en nivelGris para asignarlo a ese píxel en concreto.

2.- Realizar el histograma de la imagen en escala de grises

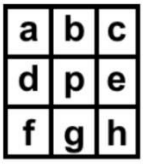
Una vez transformada la imagen a matriz en escala de grises procedemos a mejorar el contraste mediante un histograma, que calcula la frecuencia relativa con la cual aparece cada nivel de gris en la matriz y tras esto realiza una ecualización para conseguir un histograma con una distribución uniforme, de esta forma obtendremos una imagen más contrastada.

3.- Convertir imagen grises a una matriz de Blanco y Negro

Una vez hemos ecualizado la matriz procedemos a binarizarla, es decir pasarla a blanco y negro, 0 o 1 en cada pixel, esto lo haremos mediante un algoritmo de umbralización, que consiste en lo siguiente:
Dado un umbral, que por defecto lo hemos puesto a 50, recorremos todos los píxeles de la matriz, cada uno lo comparamos con nuestro umbral, si está por debajo, ese píxel lo ponemos a negro, si está por encima, lo ponemos a blanco y una vez terminado devolvemos la matriz pasada a blanco y negro.

4.- Eliminación del ruido binario

Con la imagen ya binarizada podemos encontrarnos con que se ha generado ruido, como pueden ser contornos irregulares, pequeños huecos, etc. En definitiva, queda una imagen con irregularidades, y para filtrar estas irregularidades usamos dos filtros binarios en una ventana de vecindad de 3x3:



Vecindad de 3x3

En el primer filtrado rellenamos los huecos en las zonas más oscuras y los cortes y muescas en segmentos de lados rectos, para ello recorremos la matriz y en cada iteración guardamos todos los puntos alrededor del centro p en línea recta, es decir, arriba, abajo, izquierda y derecha, con estos puntos hacemos la siguiente operación:

```
p | b & g & (d | e) | d & e & (b | g)
```

Una vez obtenido el resultado se lo asignamos a ese pixel.
En el segundo filtrado elimina los unos aislados y las pequeñas protuberancias a lo largo de segmentos de lados rectos, para

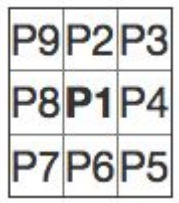
filtrar procedemos de la misma forma que en el primer filtrado, pero esta vez aplicamos esta fórmula y tenemos en cuenta todos los puntos en vecindad 3x3, es decir, las esquinas de la figura las tendremos en cuenta, los puntos en diagonal al centro p

```
p & ((a | b | d) & (e | g | h) | (b | c | e) & (d | f | g))
```

Una vez hemos aplicado los dos filtros (se encuentran en métodos separados) devolvemos la imagen ya filtrada.

5.- Adelgazamiento de la imagen

Adelgazar la imagen consiste en quedarse con su esqueleto, es decir, lo que consideraríamos la estructura básica de la misma, se consigue a base de erosionar las partes más anchas para solo quedarnos con la estructura mínima para representar la imagen.



El algoritmo para adelgazar la imagen usado es el algoritmo de Zhang-Shuen, este algoritmo trabaja por sub-iteraciones, en la primera sub-iteración se realiza un recorrido de los píxeles de la misma, para cada píxel se comprueba si es blanco, que en realidad sería negro ya que se ha invertido la imagen, si lo es, guardamos todos los puntos próximos a él en una ventana de 3x3, similar al algoritmo de filtrado binario, una vez obtenidos todos los puntos comprobamos el número de cambios “01” que hay, es decir, comprobamos si un punto es 0 y su contiguo es 1, si es así, aumentamos un contador.

Una vez comprobados todos los puntos se suman los puntos blancos encontrados(sumamos todos, solo los 1 aumentarán el contador) y se comprueba la condición:

```
(Bp1 >= 2 && Bp1 <= 6) && Ap1 == 1 && (P2 * P4 * P6) == 0 && (P4 * P6 * P8) == 0)
```

Si esta se cumple, ese píxel se anota en una matriz auxiliar para posteriormente cambiarlo de color, ya que no es esencial para la estructura.

Una vez terminada la primera sub-iteración invertimos los píxeles marcados y los borramos de la matriz auxiliar, una vez hecho esto comienza la segunda sub-iteración, realiza las mismas acciones que la primera, pero la condición cambiará esta vez, será la siguiente:

```
(Bp1 >= 2 && Bp1 <= 6) && Ap1 == 1 && (P2 * P4 * P8) == 0 && (P2 * P6 * P8) == 0)
```

En el momento en que ninguna de las 2 condiciones se cumpla, terminará la ejecución y se restaurará la imagen, ya adelgazada, a blanco y negro original, es decir, fondo blanco y estructura negra, para su mejor visualización y devolveremos esta matriz.

6.- Convertir imagen grises/ByN a matriz RGB

Las imágenes generadas tras cada función/algoritmo son matrices de enteros, si queremos visualizar el resultado como una archivo de imagen en el explorador de archivos, que es la mejor forma de comprobar visualmente la aplicación de estos algoritmos, primero debemos pasar la matriz de nuevo a BufferedImage rgb, para ello se implementa un método que se encarga de ello, convertirImagenARGB() después pasamos esta BufferedImage a otro método que se encarga de pasarlo a un archivo .jpg con ImageIO.write para crear un archivo en el explorador con el resultado.

Clases y módulos implementados

En mi proyecto tan solo hay una clase, `FingerPrintImage`, en ella se encuentran todas las funciones necesarias para realizar las operaciones anteriormente descritas, no he visto necesidad de ampliar a más clases, son unas 500 líneas de código en total.

A continuación explicaré brevemente todos los métodos de la clase FingerPrintImage.java en orden, el funcionamiento y la lógica de los algoritmos ya está explicado en el primer apartado:

```
private void seleccionarArchivo(String imagenSeleccionada)
```

Este método se usa para seleccionar un archivo y pasarlo a BufferedImage utilizando la función ImageIO.read de Java, le pasamos el nombre de la imagen o la ruta como un String, y se nos devuelve la imagen como BufferedImage que guardamos en imagenOriginal, un atributo de la clase.
También obtenemos aquí el tamaño de la imagen original, con getWidth para el ancho y getHeight para el alto, los guardamos en atributos de la clase para usarlos en otras funciones, de esta forma no tendremos que implementar nuestras funciones de obtención de ancho y alto en FingerPrintImage y además no tendremos que obtener esto en cada creación de una nueva matriz ya que lo tendremos guardado en 2 variables.

```
private void pasarImagenAArchivo(int[][] imagenEntrada, String nombreArchivo, int modo)
```

En este método convertimos una imagen de tipo int a un archivo con extensión .jpg para su visualización en el explorador de archivos.
Para hacer esto creamos un nuevo BufferedImage con el tamaño del original, tras esto convertimos la imagen a RGB con otro método que describiré más adelante, tras ello usamos ImageIO.write para crear el nuevo archivo .jpg, a este le pasaremos el parámetro nombreArchivo con el nombre del nuevo fichero en String

```
private int[][] convertirImagenAGrises(BufferedImage imagenEntrada)
```

Este método convierte una imagen de tipo BufferedImage a una matriz de enteros en escala de grises, también es el método que nos convierte de BufferedImage a matriz para usarla con el resto de métodos de procesamiento de imagen, su funcionalidad ya está descrita en el primer punto, recorre la imagen original y extrae sus componentes rgb para cada píxel de la misma, una vez obtenidos hace AND con el negro y el resultado de estas 3 las suma y divide entre 3 para obtener la media de gris de ese píxel, una vez obtenido pone ese pixel a el nivel de gris obtenido.

```
private BufferedImage convertirImagenARGB(int[][] imagenEntrada, int modo)
```

Pasa de una matriz de enteros ya sea sea en escala de grises o blanco y negro a tipo BufferedImage, si el atributo modo es 0, la imagen debe ser en blanco y negro, si es 1 será en escala de grises.

Se recorre la matriz y para cada pixel se obtiene el mismo, si el modo es 0, se multiplica por 255, tras esto se calcula su RGB y se aplica al BufferedImage creado anteriormente llamado imagenRGB para dejar ese pixel con el valor RGB calculado, al terminar devolvemos este BufferedImage.

```
private int[][] calcularHistograma(int[][] imagenEntrada)
```

Se encarga de calcular el histograma de distribución uniforme de la matriz de entrada, es decir, la ecualiza, la explicación de este algoritmo ya está en el primer punto, se recorre la matriz de entrada y guardamos en un histograma de 256 posiciones las ocurrencias de cada nivel de gris de la imagen, tras esto construimos la lookup table recorriendo el histograma anteriormente creado con los niveles de gris obtenidos y guardando en la LUT para ese valor del histograma el acumulado de ocurrencias de ese nivel de gris multiplicado por 255 y dividido por el tamaño de la imagen.
Para terminar para cada pixel de la imagen obtenemos de nuevo su valor de gris e indexamos la tabla LUT con él para obtener su nuevo valor de gris de la misma, y devolvemos la imagen ecualizada.

```
private int[][] convertirImagenAByN(int[][] imagenEntrada, int umbral)
```

Pasa la matriz en escala de grises a una matriz en blanco y negro, es decir la binarizamos, cada uno de sus píxeles será un 0 si es negro y un 1 si es blanco.
Recorremos la matriz y para cada pixel comprobamos si está por encima o por debajo de un nivel de gris establecido, es decir, usamos un algoritmo de umbralización, en mi caso he establecido el umbral en 50, por debajo de 50 se pondrá a negro y si está por encima se pondrá a 1, tras ello devolvemos la imagen en blanco y negro

```
private int[][] filtroBinario1(int[][] imagenEntrada)
```

Aplica el filtro binario 1 descrito en el guión de la práctica, el algoritmo está explicado en el primer punto.
Se recorre la matriz guardando los puntos adyacentes al centro en línea recta, arriba, abajo, izquierda y derecha, para esos puntos calculamos un valor entero con la fórmula siguiente:

```
p | b & g & (d | e) | d & e & (b | g)
```

Y tras ello aplicamos este valor a ese píxel, seguidamente devolvemos la imagen filtrada.

```
private int[][] filtroBinario2(int[][] imagenEntrada)
```

Aplica el filtro binario 2, similar al primero pero teniendo en cuenta las esquinas, es decir, los puntos en diagonal desde el centro p, y la fórmula para calcular el valor es la siguiente:

```
p & ((a | b | d) & (e | g | h) | (b | c | e) & (d | f | g))
```

Una vez obtenido el valor lo asignamos a el pixel consultado en esa iteración.

```
private int[][] adelgazamientoZhangSuen(int[][] imagenEntrada)
```

Se encarga de adelgazar la imagen haciendo uso del algoritmo de Zhang-Suen, su lógica está explicada en el primer punto.

Al empezar se invierte la imagen, tras ello comienza la primera sub-iteración, en ella se recorre la matriz y se guardan los puntos adyacentes a cada punto blanco(recordemos que

se ha invertido la imagen previamente) en una ventana de vecindad de 3x3, una vez obtenidos se comprueba el número de cambios de tipo 01 entre los puntos contiguos, si hay cambio, se aumenta un contador, al final se suman en otra variable todos los puntos, los que sean blancos agregarán 1 a la suma, una vez terminado esto se comprueba la condición:

```
(Bp1 >= 2 && Bp1 <= 6) && Ap1 == 1 && (P2 * P4 * P6) == 0 && (P4 * P6 * P8) == 0)
```

Si se cumple esta condición, ese píxel que está siendo consultado en ese momento se pone a 1 en una matriz auxiliar para anotar los que hay que invertir, y ponemos un booleano a true que se consultará en el bucle do while

Reiniciamos el contador de cambios una vez anotado el cambio

Cuando termina la primera sub-iteración se aplican los cambios anotados para ello recorremos la matriz auxiliar consultando aquellos puntos anotados a cambiar, cuando encontremos uno lo invertimos, una vez terminada comienza la segunda sub-iteración, su funcionamiento es el mismo, con un cambio en la condición marcado en negrita:

```
(Bp1 >= 2 && Bp1 <= 6) && Ap1 == 1 && (P2 * P4 * P8) == 0 && (P2 * P6 * P8) == 0)
```

y tras terminar esta también se aplican los cambios y se consulta el booleano para saber si se ha cambiado a alguno, cada vez que comience este bucle se pondrá a false y cambiará a true en el momento en que encuentre algún píxel a invertir, cuando ninguna de las 2 sub-iteraciones encuentre alguno a cambiar se terminará el algoritmo y volverá a invertir la matriz para devolverla a la configuración original, fondo blanco y esqueleto de la imagen negro, se devuelve la imagen adelgazada al terminar.

```
public int menuSeleccion()
```

Este método sirve para mostrar el menú por consola además de recibir la opción seleccionada por el usuario usando un Scanner, consta de 8 opciones contando el 0 que es la opción que termina el programa, en el primer paso se pide al usuario la imagen sobre la cual desea realizar las operaciones, los pasos 2 a 6 se corresponden con los algoritmos descritos en el primer punto y la última opción realiza todos los pasos con la imagen seleccionada hasta adelgazarla.

```
public void procesoPrincipal()
```

Este es el método principal que se ejecuta al iniciar el programa, llama al método anterior, menuSeleccion() para mostrar el menú de opciones y dependiendo de la opción elegida por el usuario mediante un switch ejecuta la función elegida.

Para cada operación de transformación que se realice sobre la imagen se mostrará un mensaje informando que se ha creado un nuevo archivo con el nombre de la transformación realizada y se creará este archivo que estará disponible para la visualización en el directorio en el cual se ejecute el programa.

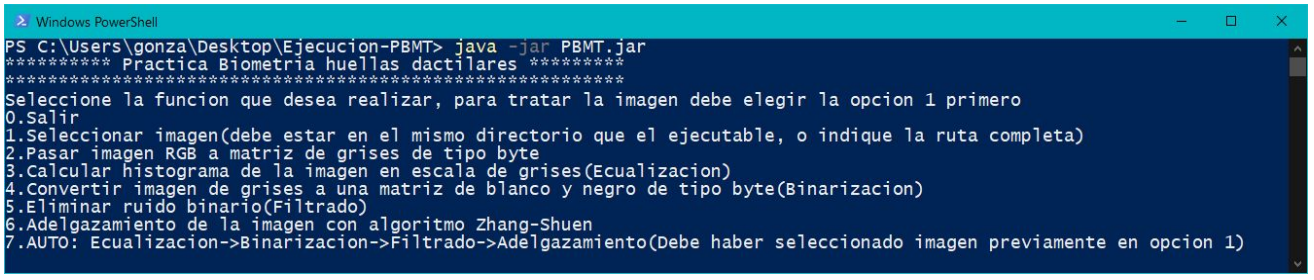
Está en un bucle mientras no se elija la opción 0 del menú que es salir, una vez pulsada el programa termina.

Ejemplo de ejecución

Realizaremos un ejemplo de ejecución de la práctica usando el ejecutable .jar de la misma y usando una imagen ubicada en el mismo directorio que el ejecutable por comodidad al elegir la imagen sobre la que queremos trabajar.
Usaré el PowerShell de Windows, pero también se puede ejecutar con cmd o cualquier otro terminal con el que se ejecute un programa en Java por comandos.

Imagen original usada, obtenida del campus virtual de la asignatura:

Nada más ejecutarla se nos presenta el menú principal:



Para comenzar debemos elegir una imagen con la opción 1, elegimos esta opción y escribimos por consola el nombre de la imagen, en mi caso imagen.jpg:

Una vez seleccionada la imagen aplicamos la primera transformación, la opción 2, para pasarla a una matriz en escala de grises, tras esto se generará una nueva imagen, imagenEscalaGrises.jpg que podremos encontrar en el directorio dónde estamos ejecutando el programa

```
Windows PowerShell
************************************************************
Seleccione la funcion que desea realizar, para tratar la imagen debe elegir la opcion 1 primero
0.Salir
1.Seleccionar imagen(debe estar en el mismo directorio que el ejecutable, o indique la ruta completa)
2.Pasar imagen RGB a matriz de grises de tipo byte
3.Calcular histograma de la imagen en escala de grises(Ecualizacion)
4.Convertir imagen de grises a una matriz de blanco y negro de tipo byte(Binarizacion)
5.Eliminar ruido binario(Filtrado)
6.Adelgazamiento de la imagen con algoritmo Zhang-Shuen
7.AUTO: Ecualizacion->Binarizacion->Filtrado->Adelgazamiento(Debe haber seleccionado imagen previamente en opcion 1)
2

imagenEscalaGrises.jpg generada

************************************************************
```

La siguiente opción, 3, es la del cálculo del histograma, es decir, la ecualización de la imagen, generará un nuevo archivo, imagenHistograma.jpg:

```
Windows PowerShell
************************************************************
Seleccione la funcion que desea realizar, para tratar la imagen debe elegir la opcion 1 primero
0.Salir
1.Seleccionar imagen(debe estar en el mismo directorio que el ejecutable, o indique la ruta completa)
2.Pasar imagen RGB a matriz de grises de tipo byte
3.Calcular histograma de la imagen en escala de grises(Ecualizacion)
4.Convertir imagen de grises a una matriz de blanco y negro de tipo byte(Binarizacion)
5.Eliminar ruido binario(Filtrado)
6.Adelgazamiento de la imagen con algoritmo Zhang-Shuen
7.AUTO: Ecualizacion->Binarizacion->Filtrado->Adelgazamiento(Debe haber seleccionado imagen previamente en opcion 1)
3

imagenHistograma.jpg generada

************************************************************
```

La siguiente opción, 4, es convertir la imagen en una imagen en blanco y negro, es decir, binarizar la imagen:

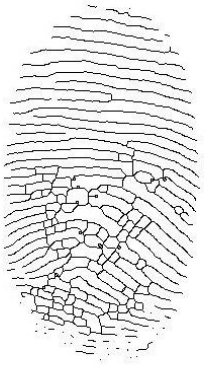
```
Windows PowerShell
************************************************************
Seleccione la funcion que desea realizar, para tratar la imagen debe elegir la opcion 1 primero
0.Salir
1.Seleccionar imagen(debe estar en el mismo directorio que el ejecutable, o indique la ruta completa)
2.Pasar imagen RGB a matriz de grises de tipo byte
3.Calcular histograma de la imagen en escala de grises(Ecualizacion)
4.Convertir imagen de grises a una matriz de blanco y negro de tipo byte(Binarizacion)
5.Eliminar ruido binario(Filtrado)
6.Adelgazamiento de la imagen con algoritmo Zhang-Shuen
7.AUTO: Ecualizacion->Binarizacion->Filtrado->Adelgazamiento(Debe haber seleccionado imagen previamente en opcion 1)
4

imagenByN.jpg generada

************************************************************
```

El siguiente paso es el filtrado para eliminar el ruido binario, la opción 5:

```
Windows PowerShell
************************************************************
Seleccione la funcion que desea realizar, para tratar la imagen debe elegir la opcion 1 primero
0.Salir
1.Seleccionar imagen(debe estar en el mismo directorio que el ejecutable, o indique la ruta completa)
2.Pasar imagen RGB a matriz de grises de tipo byte
3.Calcular histograma de la imagen en escala de grises(Ecualizacion)
4.Convertir imagen de grises a una matriz de blanco y negro de tipo byte(Binarizacion)
5.Eliminar ruido binario(Filtrado)
6.Adelgazamiento de la imagen con algoritmo Zhang-Shuen
7.AUTO: Ecualizacion->Binarizacion->Filtrado->Adelgazamiento(Debe haber seleccionado imagen previamente en opcion 1)
5

imagenFiltrada.jpg generada

************************************************************
```

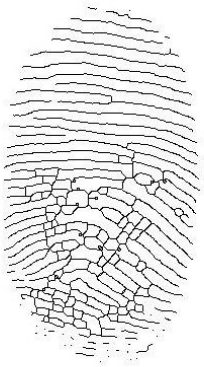
El último paso, el 6, realiza el adelgazamiento haciendo uso del algoritmo de Zhang-Suen:

```
Windows PowerShell
************************************************************
Seleccione la funcion que desea realizar, para tratar la imagen debe elegir la opcion 1 primero
0.Salir
1.Seleccionar imagen(debe estar en el mismo directorio que el ejecutable, o indique la ruta completa)
2.Pasar imagen RGB a matriz de grises de tipo byte
3.Calcular histograma de la imagen en escala de grises(Ecualizacion)
4.Convertir imagen de grises a una matriz de blanco y negro de tipo byte(Binarizacion)
5.Eliminar ruido binario(Filtrado)
6.Adelgazamiento de la imagen con algoritmo Zhang-Shuen
7.AUTO: Ecualizacion->Binarizacion->Filtrado->Adelgazamiento(Debe haber seleccionado imagen previamente en opcion 1)
6

imagenAdelgazadaZS.jpg generada

************************************************************
```

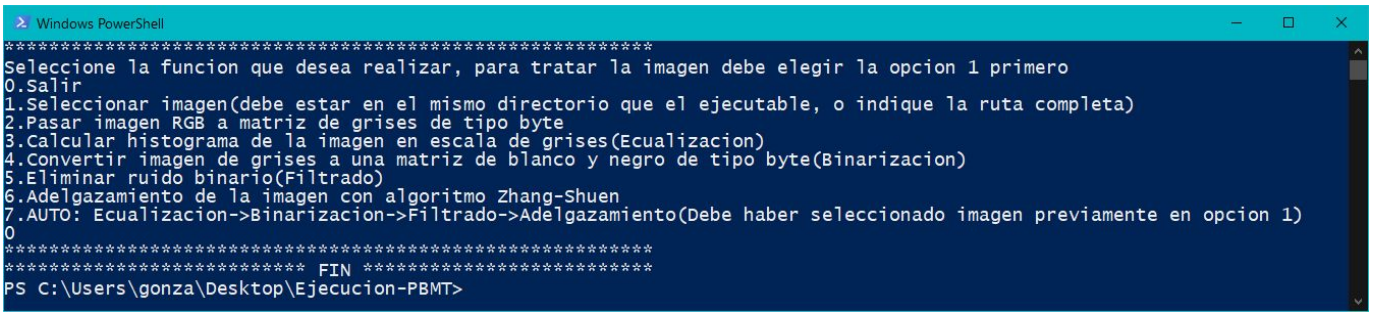


Por último tenemos la opción 7, que es una transformación automática, una vez elegida la imagen realiza la ecualización, binarización, filtrado y adelgazamiento de la misma y genera una única imagen de salida:

```
Windows PowerShell
************************************************************
Seleccione la funcion que desea realizar, para tratar la imagen debe elegir la opcion 1 primero
0.Salir
1.Seleccionar imagen(debe estar en el mismo directorio que el ejecutable, o indique la ruta completa)
2.Pasar imagen RGB a matriz de grises de tipo byte
3.Calcular histograma de la imagen en escala de grises(Ecualizacion)
4.Convertir imagen de grises a una matriz de blanco y negro de tipo byte(Binarizacion)
5.Eliminar ruido binario(Filtrado)
6.Adelgazamiento de la imagen con algoritmo Zhang-Shuen
7.AUTO: Ecualizacion->Binarizacion->Filtrado->Adelgazamiento(Debe haber seleccionado imagen previamente en opcion 1)
7

imagenAUTO.jpg generada

************************************************************
```



La opción 0 termina el programa y muestra un mensaje de fin:



Tendremos disponibles en el directorio en el cual ejecutamos la práctica todas las imágenes generadas, la imagenAUTO.jpg solo será generada si se elige la opción 7, será la única generada en este caso, la final que será la imagen tras el adelgazado, al ejecutar todos los pasos y al final el 7 como en el ejemplo de ejecución obtenemos estos archivos:

Conclusiones

Como conclusión podemos ver que con esta serie de pasos somos capaces de preparar una imagen de una huella dactilar, a falta de extraer las minutias, para después compararla con la base de datos de huellas y mediante un algoritmo de Matching identificar a una persona por un rasgo biométrico como es la huella dactilar que es algo que todo el mundo posee y se usa cada vez más como por ejemplo en dispositivos electrónicos como teléfonos móviles y tablets o para acceder a sitios donde el acceso es limitado.

La práctica me ha parecido muy interesante y gracias al guión que se nos dió se podía seguir perfectamente a falta de algunos conocimientos como tratamiento de imágenes en Java y operaciones lógicas para píxeles con hexadecimales, gracias a esta práctica he aprendido los pasos para un método de identificación muy usado hoy en día y además he ganado conocimientos sobre tratamiento de componentes RGB de una imagen, tratamiento de imágenes en Java, tratamiento de píxeles, filtrado de ruido, creación de una lookup table y operaciones lógicas a partir de píxeles de una imagen, todos estos conocimientos me resultarán útiles en el futuro.