



Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Universidad Pública de Navarra

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

PROCESADO DE SEÑALES MULTIMEDIA

Clasificador de llaves con Qt y OpenCV en C++

Jesús Arellano Usón
Víctor Labayen Guembe

Enero 2020

Índice general

1. Introducción	2
1.0.1. Notación	3
2. Herramientas	4
3. Sistema de clasificación	6
3.1. Sistema inicial	6
3.1.1. Captura de imágenes	6
3.1.2. Segmentación	7
3.1.3. Extracción de características	14
3.1.4. Clasificación	15
3.2. Desarrollo de mejoras	17
3.2.1. Watershed	17
3.2.2. Escalado de las características	20
3.2.3. SVM	23
3.2.4. Caracterización del cifrado	25
3.2.5. Rellenado de huecos	28
3.2.6. Filtrado de llaves interferentes	31
3.2.7. Ajustes de las condiciones de captura de la cámara	32
3.2.8. Limpieza de bordes	33
3.2.9. CNN: Red neuronal convolucional	35
3.3. Sistema final	39
3.3.1. Captura de imágenes	39
3.3.2. Segmentación	39
3.3.3. Extracción de características	40
3.3.4. Clasificación	40
3.3.5. Resultados	40
4. Aspectos de mejora	42
Referencias	43
Índice de figuras	44
Índice de tablas	45

1. Introducción

El objetivo del presente proyecto es diseñar, implementar y validar un sistema automático que permita reconocer y distinguir diferentes tipos de llaves. Para ello se ha dispuesto de diferentes modelos, los cuales son mostrados a continuación:



Figura 1.1: Tipos de llaves disponibles para la realización del clasificador automático.

El sistema automático debe ser capaz de identificar y reconocer la llave presentada de entre las mostradas en la figura 1.1. Para ello se debe implementar un sistema de visión artificial trabajando en el espectro visible y empleando como elemento de captura una de las *webcams* disponibles en el laboratorio.

El proyecto a realizar debe disponer de una interfaz gráfica que permita interactuar con el elemento de captación, visualizar el proceso interno de captura, procesado y segmentación de las imágenes extraídas, detección de las llaves presentes en las mismas, extracción de las características consideradas como pertinentes, entrenamiento del algoritmo de clasificación y la implementación final del sistema automático que permita representar sobre las imágenes captadas las llaves identificadas y el resultado de su clasificación.

Se ha dispuesto de un total de 6 tipos de llave diferentes, tal y como se ha mostrado en la figura 1.1. El proyecto debe abarcar la captura de imágenes para todos los tipos de llave considerados, la segmentación de dichas imágenes para identificar la región de la imagen en la que se encuentra la llave (o varias si las hubiese), la obtención de las características necesarias para su correcta identificación, el entrenamiento del clasificador seleccionado y finalmente la clasificación de nuevas imágenes ya capturadas o a partir de vídeo en tiempo real. Las condiciones en las que se realiza la captura de imágenes pueden variar. De este modo, cabe la posibilidad de que las fotos sean realizadas a diferentes alturas, con distinto fondo o en diferentes condiciones de iluminación. El entorno y las condiciones concretas de captura del presente proyecto serán expuestas posteriormente en la sección 3.1.1.

Para la realización del sistema automático de reconocimiento y clasificación de llaves se ha optado por el lenguaje de programación C++, debiendo por tanto implementarse las partes de procesado, segmentación, clasificación e interfaz gráfica del usuario. Para ello se ha dispuesto de las librerías OpenCv y Qt. Si bien, también han sido empleados otros lenguajes como MATLAB o Python como complemento y ayuda al prototipado, diseño y desarrollo del proyecto.

1.0.1. Notación

Con la finalidad de poder referenciar y particularizar algunas de las partes de las llaves sujetas a estudio y facilitar así la explicación, a lo largo de la memoria se va seguir la notación presente en la figura 1.2.

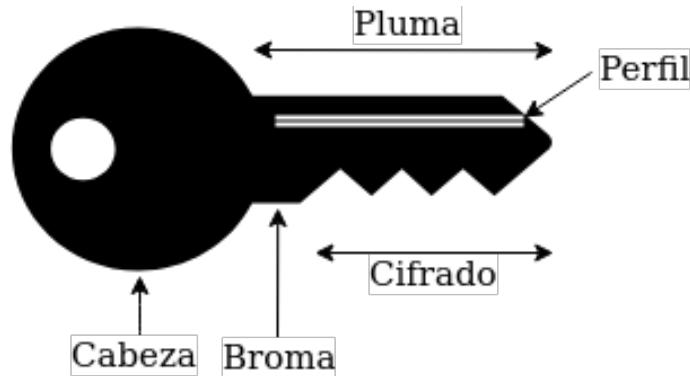


Figura 1.2: Notación seguida para la identificación de las diferentes partes de una llave.

2. Herramientas

El sistema se ha implementado en C++ utilizando Qt [1], para la creación de la interfaz gráfica, y OpenCV [2], para la lectura y procesado de imágenes, así como la clasificación de llaves utilizando algoritmos de *Machine Learning*. A su vez, se ha hecho uso tanto de MATLAB [3], para el procesado de imágenes, como de la librería Scikit-learn [4] de Python, que implementa gran cantidad de algoritmos de *Machine Learning*, con el fin de facilitar el prototipado y diseño del sistema de clasificación. Adicionalmente, se ha utilizado Keras [5], una librería de código abierto de redes neuronales implementada en Python, para el diseño y entrenamiento de una CNN. Esta es exportada mediante Tensorflow [6] permitiendo su posterior utilización en C++ mediante el módulo de *Deep learning* de OpenCV [7].

Para la identificación y segmentación de las imágenes se han utilizado diversos algoritmos y métodos vistos en la teoría de la asignatura, de entre los cuales podemos destacar los siguientes:

- Transformaciones en potencias
- Apertura
- Cierre morfológico
- Erosión morfológica
- Watershed
- Transformación en distancia
- OTSU
- Filtro de media piramidal por desplazamiento
- Operaciones lógicas
- Detección de regiones conexas
- Limpieza de bordes

Para la extracción de características no se ha utilizado ningún algoritmo concreto. Todos los descriptores utilizados han sido propuestos específicamente para la resolución del problema en cuestión e implementados directamente utilizando C++. Sin embargo, se han utilizado ciertas figuras de mérito vistas en teoría para evaluar el nivel de separabilidad que aportan las características calculadas. Podemos destacar los siguientes:

- Histograma
- Índices J de separabilidad

Por otro lado, podemos mencionar los algoritmos de clasificación y *clustering* utilizados. Todos ellos han sido vistos en la teoría de la asignatura y podemos destacar los siguientes:

- KMeans
- SVM
- Redes neuronales convolucionales (CNN)

Finalmente, para la evaluación del sistema se han utilizado los siguientes parámetros, estrategias y figuras de mérito:

- Validación cruzada aleatoria (*Cross-validation*)

- Validación cruzada dejando uno fuera (*Leave-one-out Cross-validation*)
- Matrices de confusión (*Confusion Matrix*)
- Métricas de precisión, sensibilidad y f1-score

3. Sistema de clasificación

En este capítulo se va a proceder a explicar el sistema de clasificación propuesto inicialmente, separándolo en 4 partes fácilmente diferenciables: Captura de imágenes, segmentación, extracción de características y clasificación. Posteriormente se explicarán los problemas detectados y las mejoras introducidas en el sistema para tratar de darles solución. Finalmente, se hará un breve resumen del estado final del sistema basándose en la propuesta inicial y los cambios introducidos.

La captura de imágenes ha sido implementada en C++ utilizando OpenCV y se explicará en el apartado 3.1.1. La segmentación se ha desarrollado utilizando MATLAB y Python para facilitar el diseño y, posteriormente, se ha implementado en C++ mediante el uso de funciones de OpenCV. Por el contrario, la extracción de características ha sido íntegramente desarrollada e implementada en C++. Finalmente, la clasificación ha sido desarrollada utilizando las librerías Scikit-learn y Keras de Python, facilitando el diseño y evaluación de distintos clasificadores. Posteriormente, cuando los resultados de clasificación alcanzaron las expectativas deseadas, los clasificadores fueron también implementados en C++ para realizar la clasificación de nuevas imágenes.

En la figura 3.1 se puede ver un diagrama de alto nivel que muestra el flujo de diseño seguido con los bloques mencionados anteriormente.

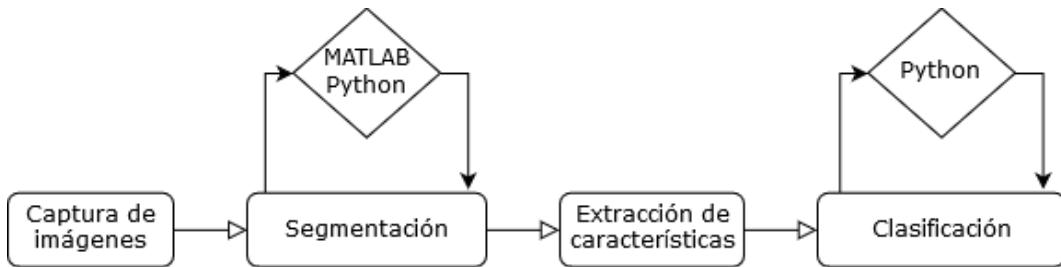


Figura 3.1: Flujo de diseño del sistema.

3.1. Sistema inicial

3.1.1. Captura de imágenes

Para la clasificación se debe disponer de un *dataset* con numerosas imágenes de las diferentes llaves disponibles. Tal y como se ha explicado con anterioridad, el sistema de visión implementado emplea una *webcam* como elemento de captura. Para generar la base de datos requerida, se debieron acotar las condiciones de contorno. Como punto de partida se decidió realizar las capturas en las mejores condiciones posibles. Esto implica evitar aquellos elementos o fenómenos que posteriormente pueden complicar el procesado, segmentación o clasificación de las imágenes.

Durante el proceso de generación del *dataset*, la *webcam* empleada fue fijada con ayuda de un trípode a la mesa del laboratorio formando un plano cenital con ella. De este modo, se buscaba que las llaves se dispusiese lo más perpendicular al plano de captura. La cámara fue situada a una altura de 21 cm respecto al plano de la mesa. De este modo, se buscó que el total de las llaves disponibles pudiesen entrar en el plano de una manera uniforme si que tuviesen que estar forzosamente juntas o el área total sin elementos a detectar fuera excesivamente grande.

Para la elección del fondo se optó por una superficie lo más uniforme posible (evitando así posibles elementos o ruido de fondo) y que propiciará el mayor contraste con los elementos a detectar. De este

modo, la segmentación a realizar posteriormente sería más sencilla. Es por esto por lo que el fondo finalmente escogido fue una funda de ordenador de color negro tejida en neopreno, frente a otras opciones probadas como una cartulina mate del mismo color o la propia mesa del laboratorio.

Debido a que las llaves reflejan la luz se optó por tratar de uniformizar la luz incidente sobre el plano de captura. El objetivo fue evitar posibles problemas en la posterior segmentación y umbralización. Se determinó que idealmente la intensidad luminosa incidente sobre la superficie de la llave debía ser lo más uniforme posible. Para ello, se optó por apagar la iluminación del laboratorio y emplear la mesa más alejada a las ventanas que dan al exterior del edificio.

Como punto de partida y de acuerdo con el enunciado del proyecto, se realizaron 20 capturas diferentes por llave, variando la posición dentro del plano de captura y volteando las llaves en varias ocasiones para obtener la mayor variedad posible en el conjunto de las imágenes. Adicionalmente, se obtuvieron 20 fotografías más del conjunto total de las llaves, así como imágenes del fondo en ausencia de cualquier objeto por si fuera necesario realizar un procesado posterior para eliminar gradientes de iluminación.

En la tabla 3.1 se muestra la cantidad de imágenes finalmente capturadas para cada tipo de llave, así como para el conjunto de todas ellas de forma simultánea.

Llave	Cantidad de imágenes
1	20
2	21
3	21
4	21
5	21
6	22
Mix	17

Tabla 3.1: Composición del *dataset* capturado.

3.1.2. Segmentación

Tal y como se ha indicado con anterioridad en la sección 1 : 'Introducción', el sistema de visión artificial debe identificar la presencia de llaves en las imágenes a procesar para posteriormente poder extraer características pertinentes que ayuden al clasificador a determinar el tipo de llave en cuestión. Para ello es necesario segmentar todos aquellos elementos presentes en la imagen que puedan ser considerados como una llave, determinar su posición, área y orientación para poder aislar cada uno de ellos de manera discreta y facilitar así su posterior procesado.

En primer lugar se representaron las imágenes capturadas mediante la *webcam*, empleando para ello diferentes modelos de color con la intención de determinar cuál de ellos aportaba mayor cantidad de información de interés.

En la figura 3.2, es posible apreciar el modelo de representación RGB. En ella se ha representado la imagen original junto con los planos R, G y B de manera independiente. La diferencia entre los diferentes planos es mínima y no hay ninguno en el que la información aportada sea más relevante que en la imagen original.



(a) Imagen original.



(b) Plano R.



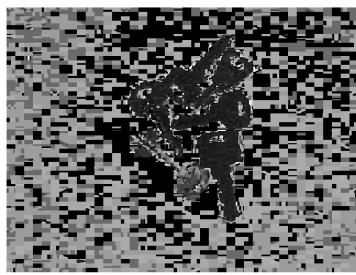
(c) Plano G.



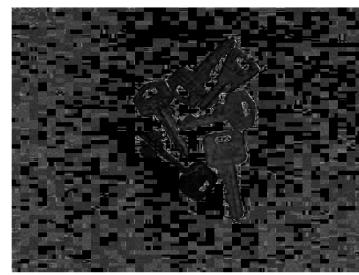
(d) Plano B.

Figura 3.2: Representación de imagen capturada empleando el modelo *RGB*.

Por otro lado, en la figura 3.3, es posible apreciar el modo de representación *HSV*. En ella se han representado los planos H, S, V de manera independiente. Los planos H y S aportan información que puede dificultar la interpretación de la imagen ya que no propician la diferenciación de las llaves respecto al fondo de manera clara, además de transformarse este en un elemento más ruidoso. El plano V por su parte es muy similar a la imagen original.



(a) Plano H.



(b) Plano S.



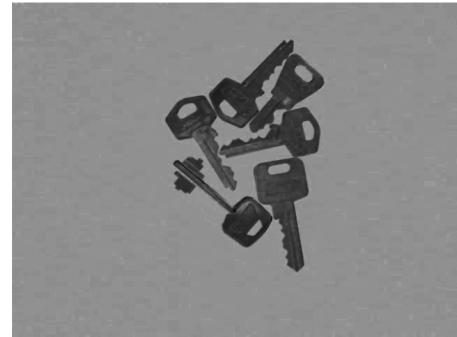
(c) Plano V.

Figura 3.3: Representación de imagen capturada empleando el modelo *HSV*.

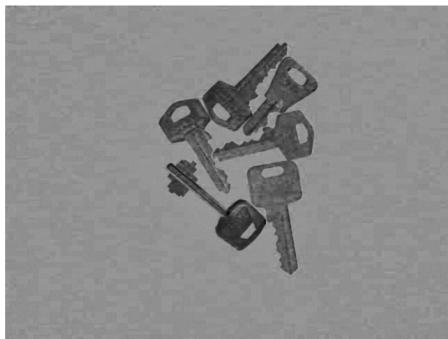
Finalmente, en la figura 3.4, se ha optado por visualizar los planos C, M, Y y K del modelo de representación *CMYK*. En esta ocasión el plano C y en menor medida el M aportan un buen contraste entre las llaves y el fondo. Sin embargo, se ha considerado como punto de partida el plano V del modelo de representación *HSV* (Ver figura 3.3) debido a que permite diferenciar mejor los diferentes elementos del plano de captura.



(a) Plano C.



(b) Plano M.



(c) Plano Y.



(d) Plano K.

Figura 3.4: Representación de imagen capturada empleando el modelo CMYK.

Ateniendo a la figura 1.1 se previó que, si bien algunas de las llaves podrían ser diferencias con parámetros como el área o el centroide, algunas de las llaves tienen un alto grado de similitud (Ver llaves tipo 1, 2 y 3) por lo que pueden presentar magnitudes similares. Es por esto por lo que se consideró de interés que durante la segmentación se preservara en el mayor grado posible la información aportada por el cifrado de la llave (Ver figura 1.2), debido a que posteriormente, en el proceso de extracción de características, podría resultar de interés. Por tanto, con el objetivo de conservar los bordes, en primer lugar se obtuvo la derivada de primer orden de la imagen, aportada por el gradiente. Para ello, se han empleado dos máscaras de *Sobel*, una para la dirección X y otra para la dirección Y. Las máscaras empleadas son de una dimensión 3x3 y son las siguientes:

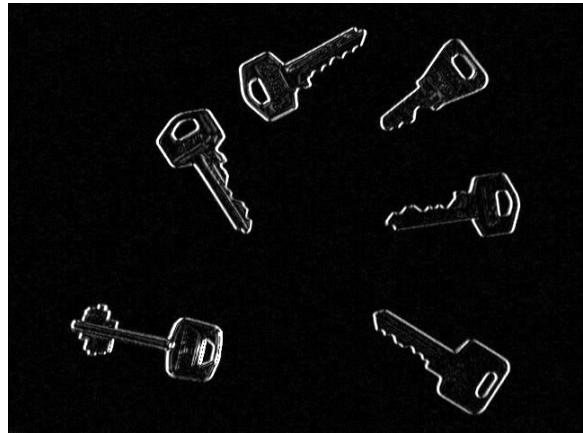
$$\text{Dirección } X : \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{Dirección } Y : \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Una vez computado el filtro de realce sobre la imagen utilizando las máscaras especificadas, se ha calculado el valor absoluto de las mismas y se ha sumado sobre la imagen original, obteniendo así una versión realzada de la captura a segmentar.

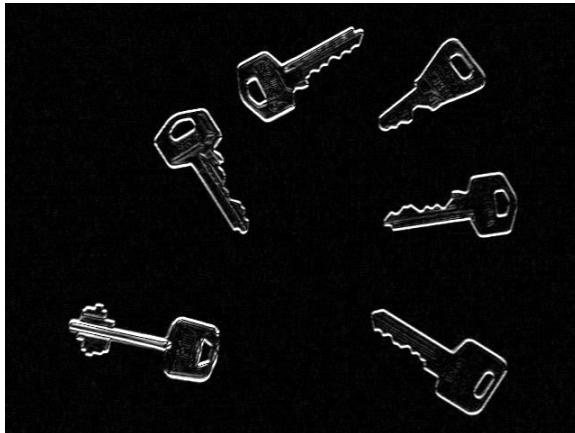
El resultado del filtro de realce de primer orden y la imagen realzada pueden ser visualizadas en la figura 3.5. Cabe mencionar que el filtro de gradiente no ha sido sometido a ningún tipo de escalado y que el interpolado empleado para la implementación en *OpenCv* es la configurada por defecto, correspondiente con un interpolado en el que los píxeles son espejados de acuerdo a la siguiente expresión: *gfedcb|abcdefgh|gfedcba* (donde los límites de la imagen están definidos por el separador |).



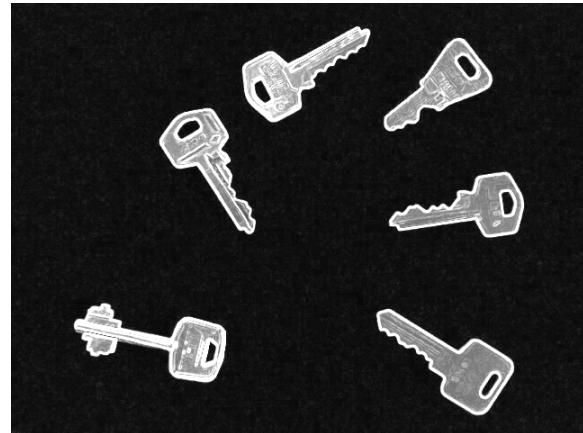
(a) Imagen original.



(b) Valor absoluto del filtrado en la dirección X.



(c) Valor absoluto del filtrado en la dirección Y.



(d) Relace de la imagen original.

Figura 3.5: Proceso de aplicación del gradiente y realce de la imagen origina

Tal y como se puede apreciar en la figura, la versión realizada de la imagen original permite apreciar con mejor detalle la totalidad de la pluma de la llave, conservando el cifrado, la broma y el perfil además de preservar íntegramente la cabeza.

Las llaves presentes en la imagen realizada presentan un alto contraste respecto al fondo negro que ha permanecido prácticamente inalterado a la salida del filtro de tipo gradiente. Debido a esta condición de alto contraste, se optó por tratar de binarizar la imagen a partir de este estado. Para ello, se implementó una umbralización global óptima empleando el método OTSU. El resultado de dicha binarización puede ser apreciada en la Figura 3.6 donde el umbral determinado con el método OTSU es de 105.00.

Una vez realizada la binarización de las imágenes se constató que en alguna de ellas, debido a la variación en la iluminación sobre las llaves y su carácter reflectivo, algunas de las regiones internas de los elementos quedaban sin rellenar. A priori esto no supone un problema ya que se puede ser solucionado con un relleno de huecos siempre y cuando el contorno de la llave sea cerrado. Sin embargo, cabe la posibilidad de que se generen contornos no cerrados cuando la zona que no se rellena esté próxima al borde del elemento. Es por esto por lo que se optó por realizar un cierre morfológico que permita terminar el contorno de la llave. Para evitar perder el carácter abrupto del cifrado se ha empleado un elemento estructurante de tamaño pequeño, concretamente un círculo de radio 3. En la figura 3.7 se puede apreciar como, además de solucionar algunos puntos discretos que se han confundido con el fondo durante la binarización, el contorno de la llave ha terminado de definirse.

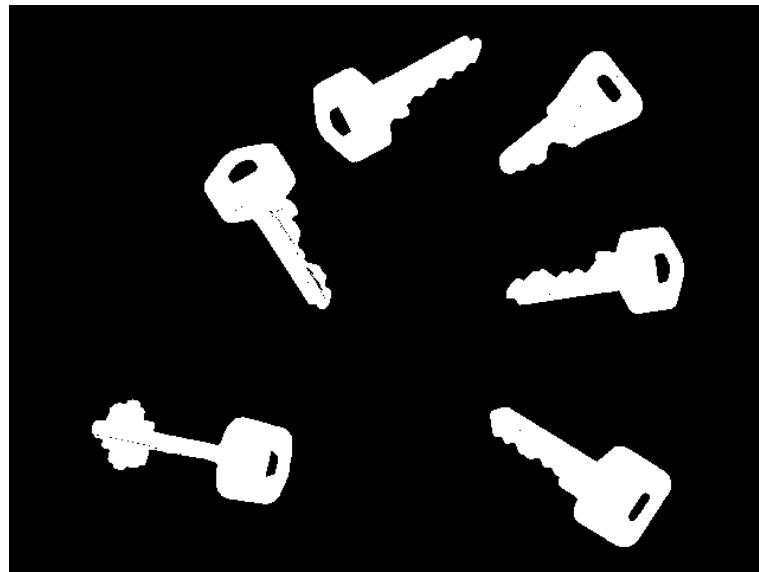
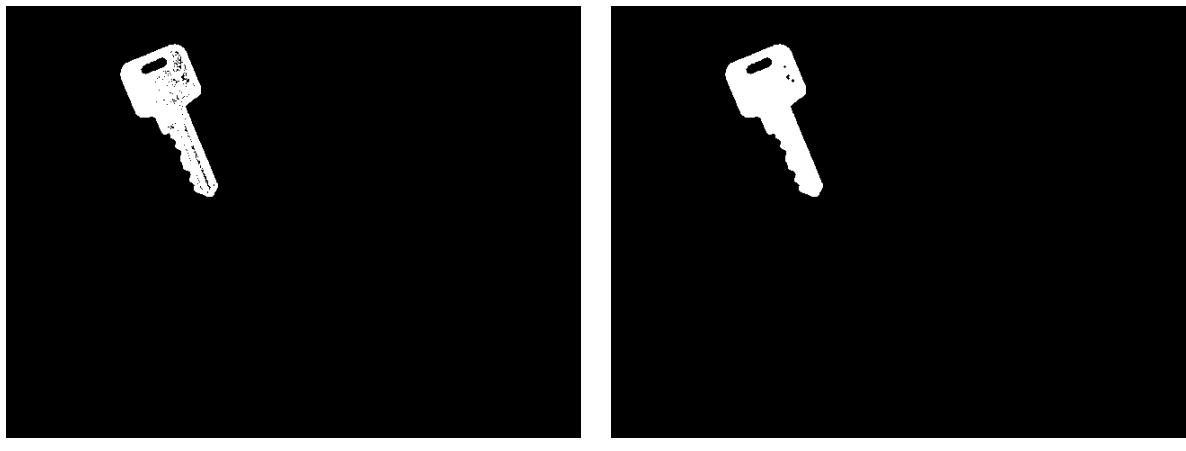


Figura 3.6: Resultado de la binarización generada por el método OTSU.

Una vez se dispone de la imagen sujeta a análisis correctamente binarizada es posible identificar las regiones conexas presentes. De este modo y con ayuda de la función de OpenCV `findcontours` se obtienen los contornos exteriores de todos aquellos elementos en estado lógico ‘1’. Para ello se ha empleado un método de aproximación simple en el que se emplea la altura, anchura y diagonal de los elementos encontrados para definir 4 puntos que lo contengan. De este modo, es posible obtener el rectángulo de área mínima que contiene a las llaves detectadas. Gracias a las funcionalidades de OpenCV se puede representar dicho rectángulo con la intención de evaluar el algoritmo diseñado para la detección de llaves, como se puede observar en la figura 3.8. La función `drawcontours` permite, a la hora de dibujar el contorno, llenar el interior del mismo con un determinado color gracias a la propiedad `color`. De este modo, se obtiene una implementación rápida del llenado de huecos.



(a) Binarización aportada por OTSU.

(b) Resultado del cierre.

Figura 3.7: Cierre de la binarización proporcionada por el método OTSU.

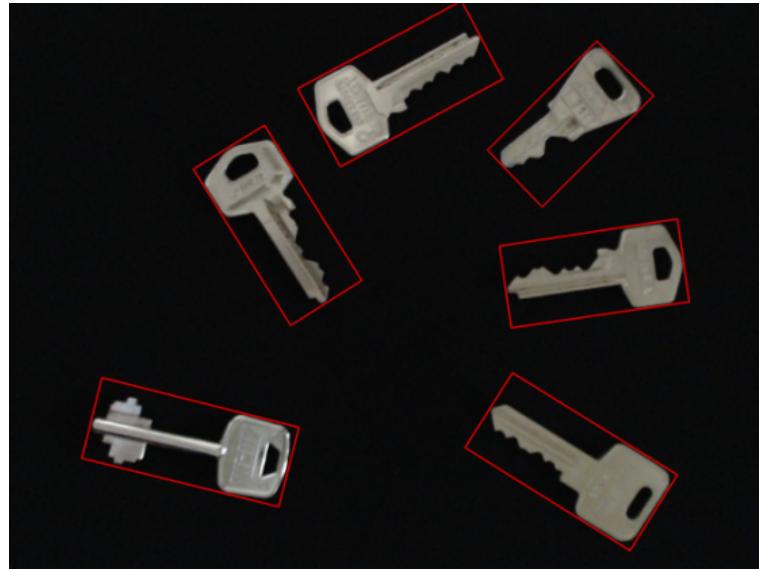


Figura 3.8: Representación del resultado de detección de regiones conexas

A lo largo del proceso de prototipado se constató que el algoritmo de segmentación desarrollado presentaba algunos problemas debido a que la presencia de otros elementos o sombras acentuadas provocadas por el perfil de las llaves generaban regiones conexas que eran detectadas como elementos diferentes. Es por esto por lo que, para acotar en mayor medida el número de regiones detectadas se implementó un primer filtro por área. Dado que el área es del mismo orden para todos los tipos de llaves (Ver tabla 3.2) se optó por eliminar aquellas regiones cuya área fuese inferior a 0.25 veces el área de la mayor región conexa detectada. Adicionalmente, frente a fondos ruidosos, la segmentación preservaba algunos elementos del fondo que debían ser procesados igualmente y que ralentizaban el funcionamiento del programa. Para evitar esa carga computacional extra relacionada con los elementos sin interés, se implementó una filtrado adicional por área en el que cualquier región detectada menor a 50 píxeles no sería tratada. Se escogió 50 píxeles dado que se constató que en una imagen de esa resolución no era posible ver de que llave se trataba.

Una vez se ha identificado y localizado las diferentes llaves presentes en las imágenes capturadas se deben obtener regiones de interés sobre la imagen original para poder procesar y extraer características de manera individual de cada una de ellas. Es por esto por lo que para facilitar el análisis posterior se optó por orientar todas las llaves localizadas de la misma manera, concretamente en la dirección horizontal.

El proceso de reorientación de las llaves requiere girar la imagen respecto a un punto central. La matriz que aloja la información relativa a la imagen puede ser transformada para girar todo el contenido con una dirección y ángulo concretos. Sin embargo, si el proceso se realiza sobre la propia matriz y respecto al punto central, cabe la posibilidad de que se pierdan píxeles de interés. Esta casuística puede apreciarse en la figura 3.9. Es por esto por lo que es necesario recrecer la imagen para no perder información de los objetos segmentados. Para ello se ha tenido en cuenta que la mayor dimensión posible que puede abarcar la nueva matriz en el giro se corresponde con la diagonal de la imagen original definida como $\sqrt{\text{Número columnas}^2 + \text{Número filas}^2}$.

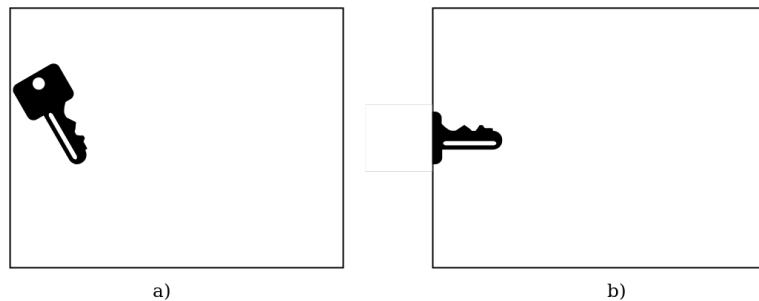


Figura 3.9: (a) Imagen original. (b) Resultado del giro sin recrecimiento.

El recrecimiento consiste en definir una matriz nueva de acuerdo a las nuevas dimensiones máximas calculadas a partir de la diagonal y copiar el contenido de la captura original compartiendo el mismo punto central. Es decir, se debe copiar el contenido de la matriz original en la recrecida a partir del punto (OffsetX , OffsetY) o lo que es lo mismo ($\frac{\text{diagonal} - \text{Número columnas}}{2}, \frac{\text{diagonal} - \text{Número filas}}{2}$). El procedimiento seguido ha sido representado en la figura 3.10.

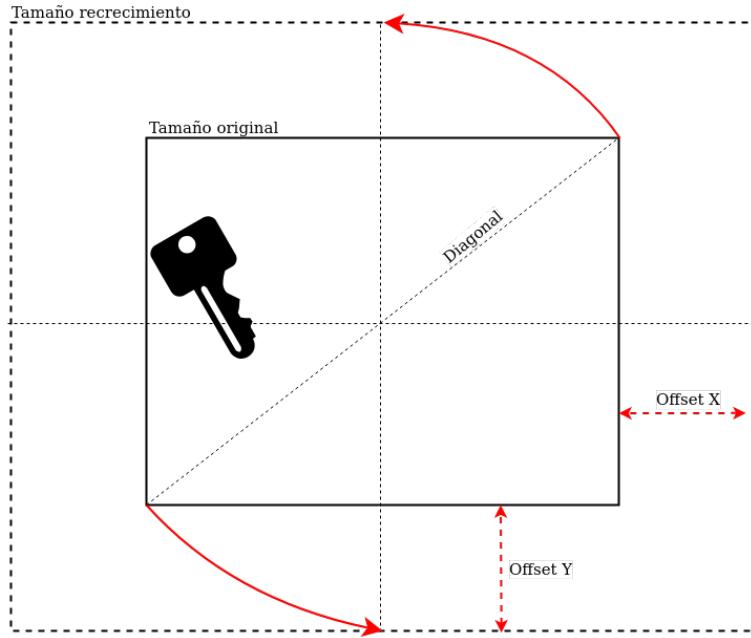


Figura 3.10: Recrecimiento de la imagen original.

El rectángulo de área mínima que contiene a las diferentes regiones conexas presentes en la imagen y detectadas por OpenCv permite acceder al ángulo con el que están giradas respecto a la horizontal. Para ello, se definen 4 puntos correspondientes con los vértices del rectángulo. El punto 0 siempre es el definido por el vértice inferior y a partir de él se definen el resto en sentido horario. Según esta notación (Ver figura 3.11), el ángulo del rectángulo de área mínima que contiene a una región conexa queda definido por el existente entre el vértice 0 y 3. Por otro lado, la anchura del rectángulo de área mínima que contiene a la región conexa queda definida siempre entre los vértices 1 y 2 (o 0 y 3) y la altura como la distancia entre los vértices 0 y 1 (o 2 y 3).

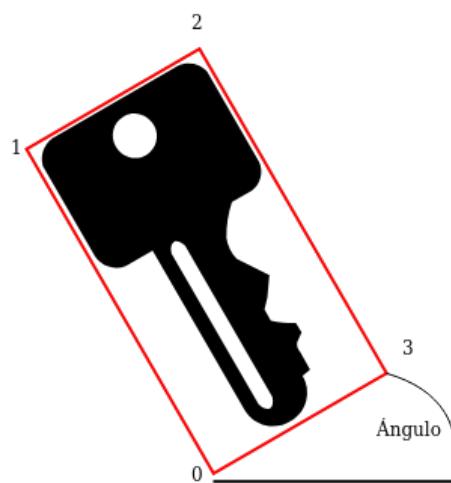


Figura 3.11: Ángulo definido por el rectángulo contenedor de una región conexa.

Para poder orientar todas las llaves de la misma manera, se determinó que la dirección en la que se

iba hacer era la más larga, la correspondiente a la pluma. El ángulo aportado por OpenCv puede tener un valor comprendido entre 0 y -90 grados. Es por esto por lo que es necesario saber en qué dirección está la llave. En caso de que la anchura sea mayor que la altura, el giro necesario corresponderá con una suma del ángulo actual más 180, mientras que en el caso contrario, solamente habrá que añadir 90 grados al ángulo para completar el giro. (En este último caso además se deberá modificar la información relativa a la región conexa para especificar que los valores de anchura y altura se han intercambiado).

Una vez se ha rotado la imagen y teniendo en cuenta la nueva posición de la región conexa es posible recortar la llave para poder extraer las características consideradas como pertinentes. Un ejemplo del estado inicial y final de la segmentación puede ser visualizado en la figura 3.12.



Figura 3.12: Resultado del proceso de segmentación

3.1.3. Extracción de características

Inicialmente, hemos tratado de caracterizar las distintas llaves a partir de la imagen segmentada, binarizada, recortada y rotada para posicionarla en sentido horizontal, tal y como se obtienen del proceso de segmentación. Un ejemplo es la imagen de la figura 3.12, mostrada anteriormente.

Como conjunto de características se han extraído el alto y el ancho de la llave, calculándose como la máxima distancia en dos píxeles de la llave binarizada, en sentido vertical y horizontal respectivamente.

A su vez, se ha extraído el área de la llave, calculado como la cantidad de píxeles resultantes del proceso de binarización.

Finalmente, se ha obtenido el centroide de la llave. Para calcularlo se suman los valores de las coordenadas x e y para todos los píxeles pertenecientes a la llave y posteriormente se dividen ambas sumas por el número total de píxeles considerados.

Dado el hecho de que llaves pueden presentar su cabeza tanto en el lado izquierdo como derecho de la imagen, los puntos del centroide obtenido se transforman a la menor distancia del borde de la imagen, obteniendo así un conjunto de valores invariantes respecto a la posición de la cabeza de la llave tras su rotación. A su vez, dichos valores del centroide se normalizan respecto al ancho y alto de la imagen recortada respectivamente, obteniendo por tanto un conjunto de dos valores comprendidos entre 0 y 0.5. Posteriormente, estos valores se dividen entre 0.5 para obtener valores acotados entre 0 y 1. Estas tres transformaciones pueden aplicarse mediante las expresiones (3.1) y (3.2), donde W y H son el ancho y alto de la imagen utilizada. A su vez, podemos observar en la figura 3.13 el resultado de estas transformaciones para el valor horizontal (c_x).

$$c_x = \begin{cases} c_x / (W \cdot 0,5) & \text{si } c_x \leq W/2 \\ (1 - c_x) / (W \cdot 0,5) & \text{si } c_x > W/2 \end{cases} \quad (3.1)$$

$$c_y = \begin{cases} c_y / (H \cdot 0,5) & \text{si } c_y \leq H/2 \\ (1 - c_y) / (H \cdot 0,5) & \text{si } c_y > H/2 \end{cases} \quad (3.2)$$

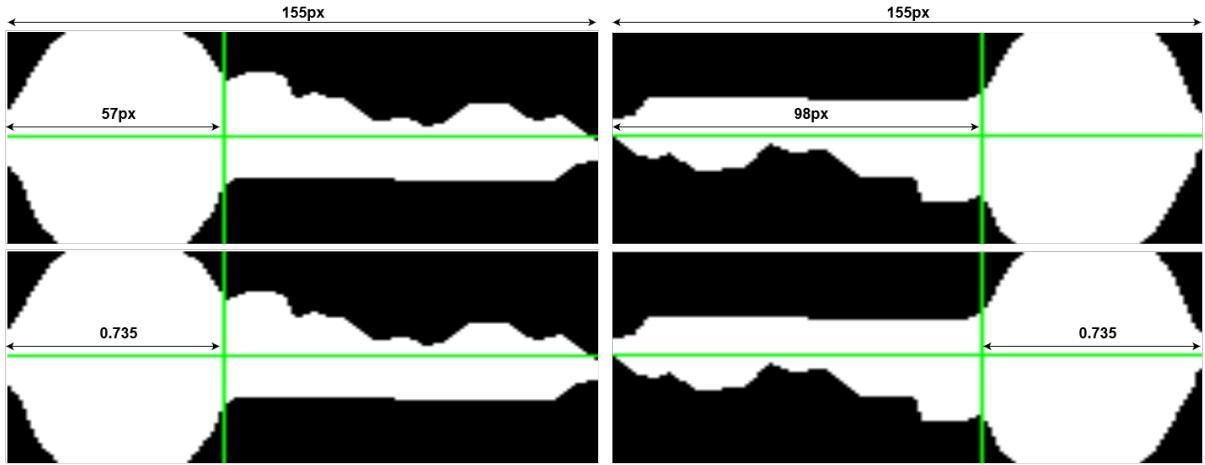


Figura 3.13: Resultado de la normalización del centroide

Para agilizar el entrenamiento, se procesan todas las imágenes del *dataset* para extraer las características mencionadas. Estas se almacenan junto con la clase correspondiente en un fichero de texto en formato **csv**, al que posteriormente se acudirá para realizar el entrenamiento y la evaluación. Podemos ver algunas de las líneas del fichero en la tabla 3.2.

Llave	Ancho	Alto	Area	Centro _x	Centro _y
1	72	153	6082	0.751	0.923
2	73	152	5963	0.742	0.990
3	70	157	6026	0.755	0.985
4	79	171	8124	0.742	0.988
5	65	130	5269	0.758	0.920
6	69	169	6589	0.841	0.951

Tabla 3.2: Ejemplo características extraídas

3.1.4. Clasificación

En primera instancia se propuso utilizar **KMeans** para tratar de clasificar las llaves, pues fue considerada como una buena opción dado que se conoce a priori el número de clases presentes en el *dataset*. Para realizar la clasificación se establecerá una relación entre clusters y clases, de forma que cada cluster representará la clase que más muestras de entrenamiento aporte a la población del mismo. Por tanto, se utilizó **KMeans** con $K = 6$ y distancias euclídeas.

Para realizar el entrenamiento y la evaluación se optó por utilizar el método de validación cruzada utilizando un 70 % de los datos para realizar el entrenamiento. Para reducir la dependencia con la aleatoriedad de este método, debido al tamaño reducido del *dataset* empleado (aproximadamente 20 imágenes por clase), se repitió el procedimiento de dividir el conjunto de datos, entrenar y evaluar un total de 100 veces.

Para realizar la evaluación del clasificador se utilizarán las siguientes métricas: precisión, sensibilidad y *f1-score*. En el contexto de clasificación con múltiples clases, para una clase dada, la precisión se puede calcular como la división entre la cantidad de observaciones correctamente clasificadas y el total de observaciones clasificadas como la clase considerada. Respecto a la matriz de confusión, para la clase '*c*', es la división entre el valor en la intersección de la fila y columna correspondientes a '*c*' y la suma de toda la columna. De forma similar, la sensibilidad es la división entre la intersección de fila y columna de una clase y la suma de toda la fila. Finalmente, el valor *f1-score* es una métrica ampliamente utilizada que recoge los valores de precisión y sensibilidad. Se puede calcular a partir de estos valores mediante la expresión (3.3).

$$\text{f1-score} = 2 * \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (3.3)$$

En la figura 3.14 puede observarse la matriz de confusión resultante del proceso de evaluación. Adicionalmente podemos observar los valores de precisión, sensibilidad y *f1-score* para cada clase, así como la media de todas ellas, en la tabla 3.3.

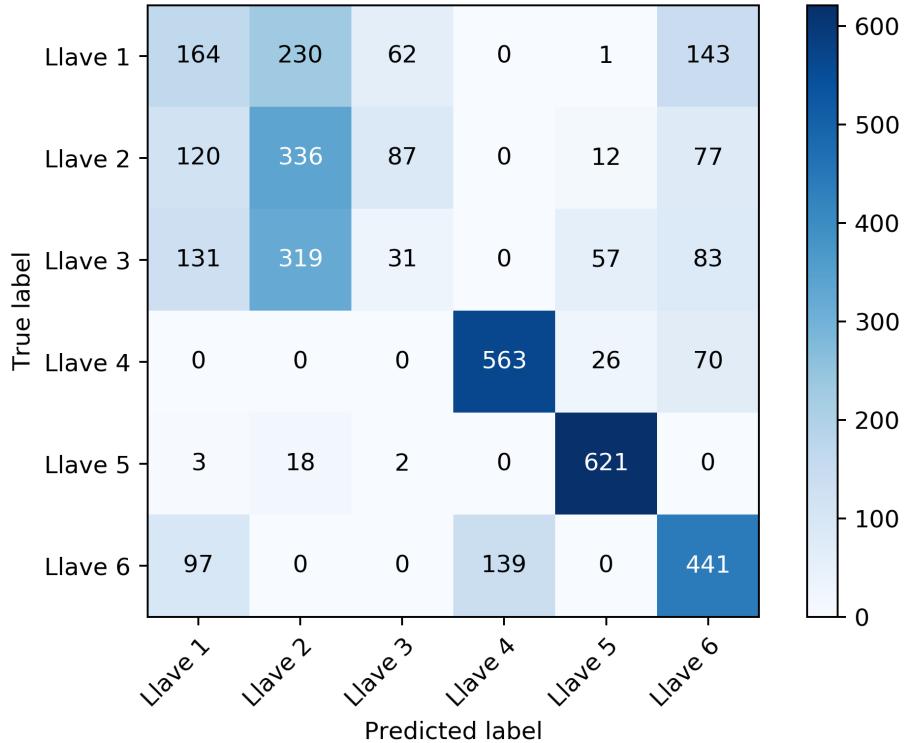


Figura 3.14: Matriz de confusión de la clasificación con KMeans.

Llave	Precisión	Sensibilidad	f1-score
1	0.3184	0.2733	0.2942
2	0.3721	0.5316	0.4378
3	0.1703	0.0499	0.0772
4	0.8020	0.8543	0.8273
5	0.8661	0.9643	0.9126
6	0.5418	0.6514	0.5915
Media	0.5118	0.5542	0.5234

Tabla 3.3: Resultados de la clasificación con KMeans.

Podemos observar como las llaves 4 y 5 son clasificadas particularmente bien con las características consideradas. Sin embargo, la llave 6, a pesar de su clara diferencia a la vista, no obtiene los mismos resultados que las dos anteriores. Finalmente, las llaves 1, 2 y 3, como era de esperar dada su similitud, resultan difícilmente separables con las características utilizadas. El problema subyacente en los errores de clasificación se debe principalmente al hecho de que las características extraídas no aportan información sobre la distribución del cifrado de las distintas llaves. Sin embargo, los problemas de la presente clasificación se explicarán con profundidad en la sección '*Desarrollo de mejoras*', donde se analizarán los distintos problemas y se propondrán las soluciones correspondientes.

3.2. Desarrollo de mejoras

3.2.1. Watershed

Problemas y propuestas

Uno de los objetivos del proyecto es poder detectar la presencia de llaves en el plano de captura para poder extraer las características que faciliten su diferenciación respecto a otros tipos. El entrenamiento se ha realizado empleado imágenes en las que solamente aparece un único tipo de llave y se sabe a priori a que clase pertenece. Sin embargo, para realizar las pruebas en tiempo real se contempla que el número de llaves presentes sea mayor que uno. A la hora de realizar las primeras pruebas se detectaron algunos problemas relacionados con la proximidad de las llaves. Esto es debido a que la hora de umbralizar y segmentar los objetos muy próximos se interpretan como una única región conexa. Un ejemplo de este fenómeno puede ser visualizado en la figura 3.15.

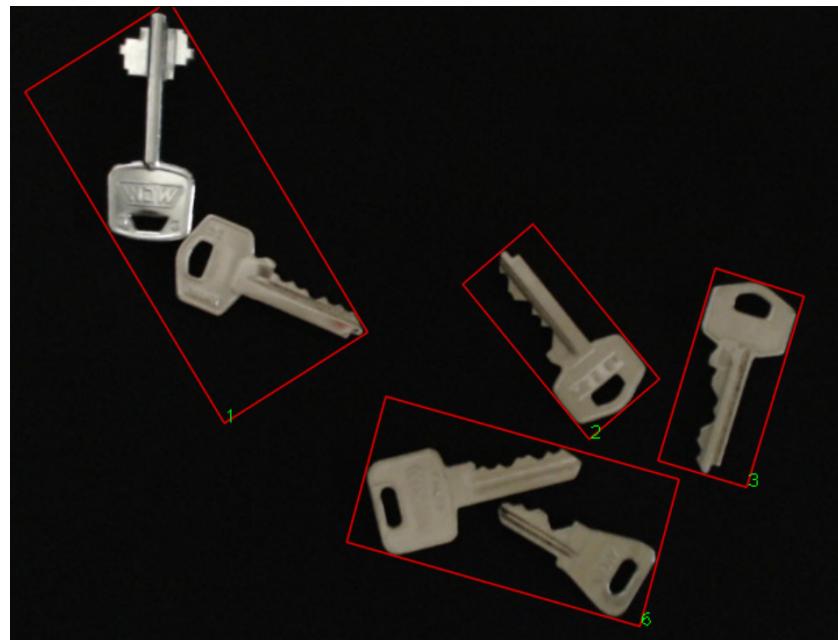
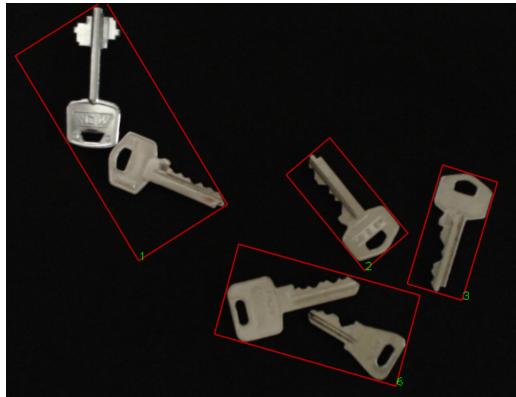


Figura 3.15: Unión de regiones conexas debida a la proximidad.

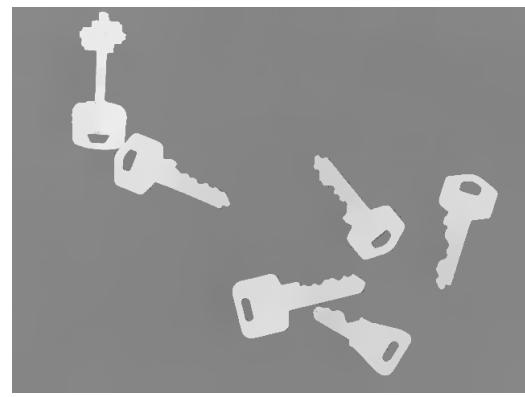
Con la intención de resolver esta problemática se decidió implementar el algoritmo de segmentación de *watershed* conjuntamente con transformaciones en distancia para poder diferenciar elementos muy próximos entre si. Para ello se empleó la librería OpenCv, pues permite emplear este algoritmo utilizando para ello marcadores que determinan los puntos de la imagen en los que comenzar el crecimiento.

Implementación

Para ello es necesario marcar todas las regiones de nuestro objeto en función de tres criterios; aquellas regiones que se conoce con certeza que pertenecen al primer plano, aquellas que pertenecen al fondo y aquellas que no se sabe con seguridad si forman parte alguna de las dos anteriores. Debido a que el algoritmo de *watershed* corre el riesgo de generar mucha sobresegmentación, el primer objetivo planteado es el de uniformizar la superficie de la llave para evitar gradientes de iluminación o variaciones abruptas en la coloración de la imagen. Es por esto por lo que tras realizar una transformación en potencia de 0.2 que permite resaltar la imagen respecto al fondo, se emplea un filtro de media piramidal por desplazamiento que aplica una homogeneización local. El resultado de dicho proceso se ha representado en la figura 3.16.



(a) Imagen original.



(b) Filtro de media por desplazamiento local.

Figura 3.16: Homogeneización local empleando filtro de media y transformación en potencias.

Una vez se homogeneizado la superficie se ha empleado el método de binarización de OTSU para poder realizar las operaciones morfológicas necesarias. El resultado se ha representado en la figura 3.17.

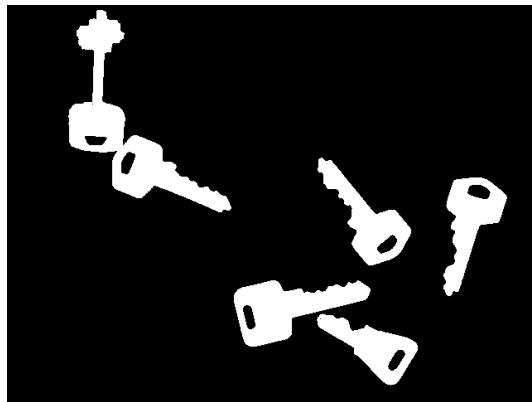


Figura 3.17: Binarización empleando el método OTSU a la salida del proceso de homogeneización.

El primer objetivo es poder detectar el fondo de las llaves. Para ello, primero se realiza una erosión con la finalidad de eliminar los píxeles residuales de la imagen. Seguidamente se realizan dos aperturas consecutivas empleando para ello un elemento estructurante definido como una matriz 3×3 . El resultado puede visualizarse en la figura 3.18.

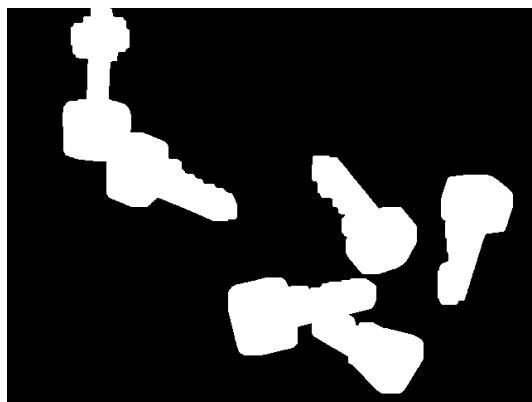


Figura 3.18: Detección del fondo de las llaves.

Para la detección del primer plano se ha realizado una trasformación en distancia que permitirá obtener una estructura más definida de las llaves. Seguidamente, con el objetivo de preservar exclusivamente una región conexa por cabeza, se realiza una apertura empleando una circunferencia de radio 33 como elemento

estructurante. Como es posible que algunos de los elementos más finos no se pierdan en dicha apertura, se aplica una umbralización empleando para ello un valor mínimo de 0.1. Este proceso se ha representado en la figura 3.19.

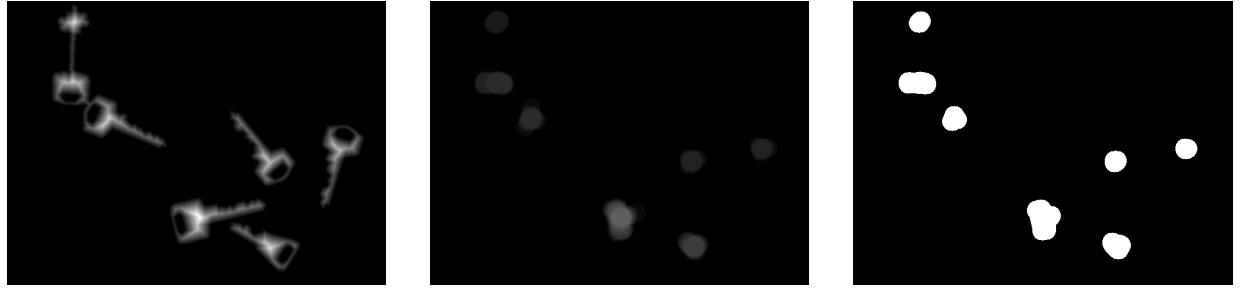


Figura 3.19: Detección del primer plano de las llaves.

Una vez se tienen determinadas las regiones que son consideradas como fondo de la llave y como primer plano de la misma, se debe obtener la diferencia entre ambas para poder delimitar una región de indecisión y determinar dentro de ella un marcador a partir del cual recrecer las cuencas.

Una vez se dispone de los marcadores se procede a aplicar el algoritmo de *watershed* sobre la imagen homogeneizada y a dibujar los contornos detectados sobre ella. Algunos de los casos resueltos satisfactoriamente por la implementación del algoritmo pueden apreciarse en la figura 3.20.

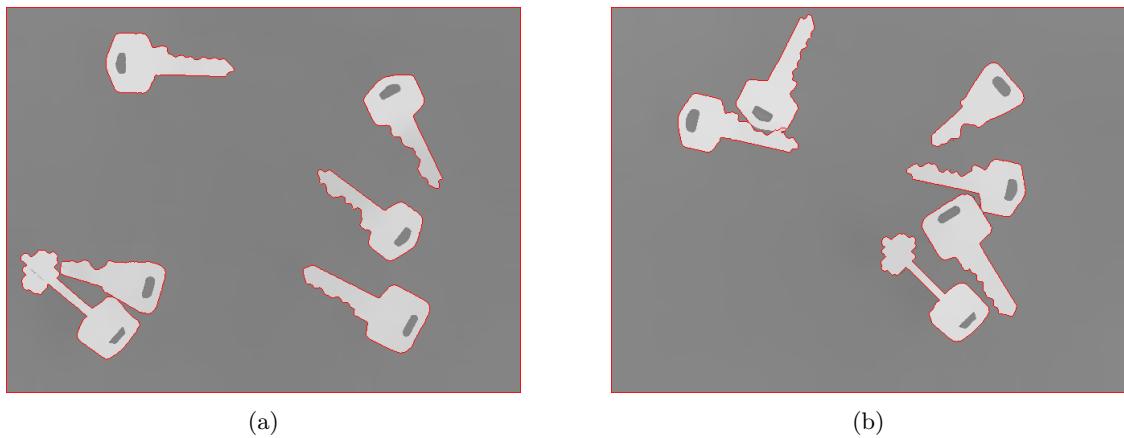
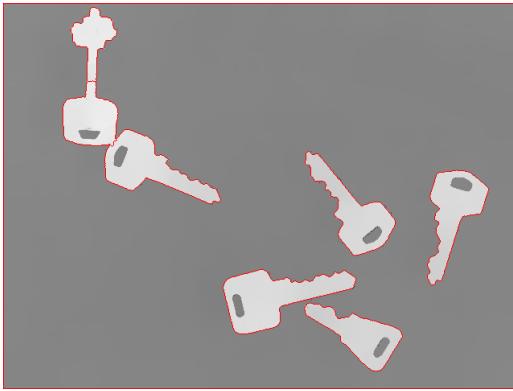


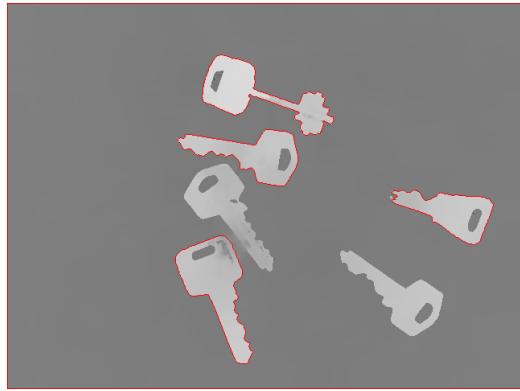
Figura 3.20: Detección de bordes realizada por *watershed* y transformación en distancia.

Resultados

Una vez implementada la segmentación por *watershed* fueron detectados algunos problemas asociados al planteamiento empleado. Algunos de ellos han sido representados en la figura 3.21. En el caso (a), se está produciendo una sobresegmentación de la llave 6. Esto es debido a que la apertura realizada sobre la transformación en distancia seguida de la umbralización, no ha sido suficiente para eliminar la pluma. La llave 6 presenta una pluma cuya área y forma se asemeja a su propia cabeza por lo que en algunas ocasiones no se dispone de un único marcador por llave y es sobresegmentada. Por otro lado, la elección de un circulo de radio 33 como elemento estructurante buscaba, precisamente, terminar con los marcadores de la pluma de la llave de tipo 6. Sin embargo, en algunos casos, esta apertura es tan agresiva que termina con los marcadores de otras llaves impidiendo que estas sean detectadas como se puede apreciar en el caso (b).



(a)



(b)

Figura 3.21: Problemas asociados a la detección de bordes realizada por *watershed* y transformación en distancia.

Conclusión

A la vista de los resultados expuestos se ha concluido que la nueva segmentación implementada supone un compromiso. Pese a que es capaz de resolver satisfactoriamente algunas casuísticas de proximidad compleja, el riesgo de sobresegmentación debido a la idiosincrasia de la llave tipo 6 es alto. Es por esto por lo que, para garantizar una mayor estabilidad del segmentador, y por tanto, del clasificador, se ha preferido preservar el modelo anterior sin la implementación del algoritmo de *watershed*, renunciando así a la posibilidad de detectar llaves muy próximas y asumiendo el riesgo de que estas sean identificadas como un único objeto en el plano de captura.

3.2.2. Escalado de las características

Problemas y propuestas

En primer lugar, a la vista de los resultados de la figura 3.14 y la tabla 3.3, se propuso afrontar los errores de clasificación en la llave 6. Para poder entender la causa de estos errores debemos empezar por evaluar la separabilidad de esta llave mediante las características mencionadas en el apartado 3.1.3 para concluir si el problema se debe afrontar con la propuesta de nuevas características o si el problema viene dado por elementos posteriores del sistema. Para permitir este análisis se ha hecho uso de la representación del histograma de las distintas características para cada una de las llaves, como podemos ver en la figura 3.22.

Podemos observar como la llave 6 resulta fácilmente separable mediante el valor de la coordenada x del centroide (centro_x). Por otra parte, observamos como el área también representa una característica importante para identificar esta llave. Finalmente, el ancho de la llave también aporta mucha información pues únicamente comparte el espacio de características con la llave 4, que resulta ser la opuesta en la característica mencionada en primer lugar.

Por tanto, mediante el uso de estas características el clasificador debería ser capaz de separar adecuadamente las muestras de las distintas llaves. Sin embargo, podemos observar como el rango de variación entre la coordenada x del centroide y las otras dos características difiere en varios órdenes de magnitud. A su vez, la anchura de la llave presenta variaciones de menor orden que el área.

Dado que estamos usando KMeans con distancia euclídeas, es muy posible que la clasificación este siendo condicionada principalmente por el área de la llave, al ser la que presenta mayor variabilidad debido a su magnitud y, por tanto, mayores distancias entre muestras. Si observamos el histograma de esta característica podemos ver como la llave 6 se solapa con las llaves 1, 3, 2 y 4, de mayor a menor medida, generando posibles errores en la clasificación. Si observamos de nuevo la figura 3.14, podemos ver como la columna de la llave 6 muestra exactamente este orden de confusión con las llaves citadas, verificando esta problemática.

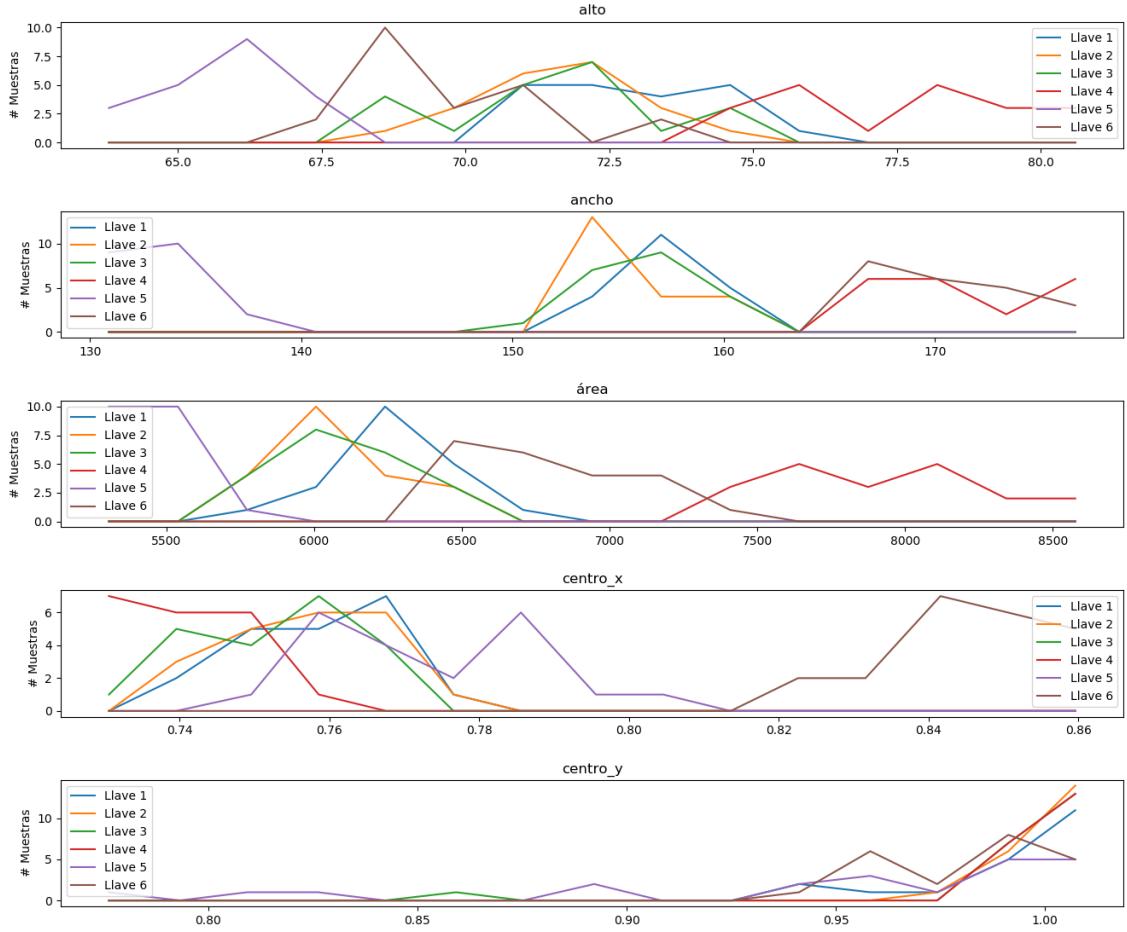


Figura 3.22: Histograma de las características utilizadas.

Para solucionar este problema debemos escalar las distintas características de forma que sean comparables en magnitud, tal y como se ha visto en la teoría de la asignatura. Por tanto, en nuestro caso concreto, se ha optado por realizar la estandarización de las distintas características.

Implementación

Para realizar la implementación de un estandarizador en C++ se ha procedido a calcular las medias y desviaciones típicas de cada una de las características a lo largo de todo el *dataset*, que ya se habían guardado anteriormente en un fichero de texto. Estos valores son guardados nuevamente en un fichero de texto en formato **csv**, permitiendo utilizarlos posteriormente para la estandarización de las características de nuevas imágenes, sin tener que volver a procesar todos los datos.

Resultados

Una vez realizada la estandarización de los datos podemos comparar las nuevas dimensiones de las características, mostradas en la figura 3.23, frente a las anteriores, mostradas en la figura 3.22. Al ser el estandarizado una transformación lineal, podemos ver como únicamente se modifica el rango y no la forma de los respectivos histogramas.

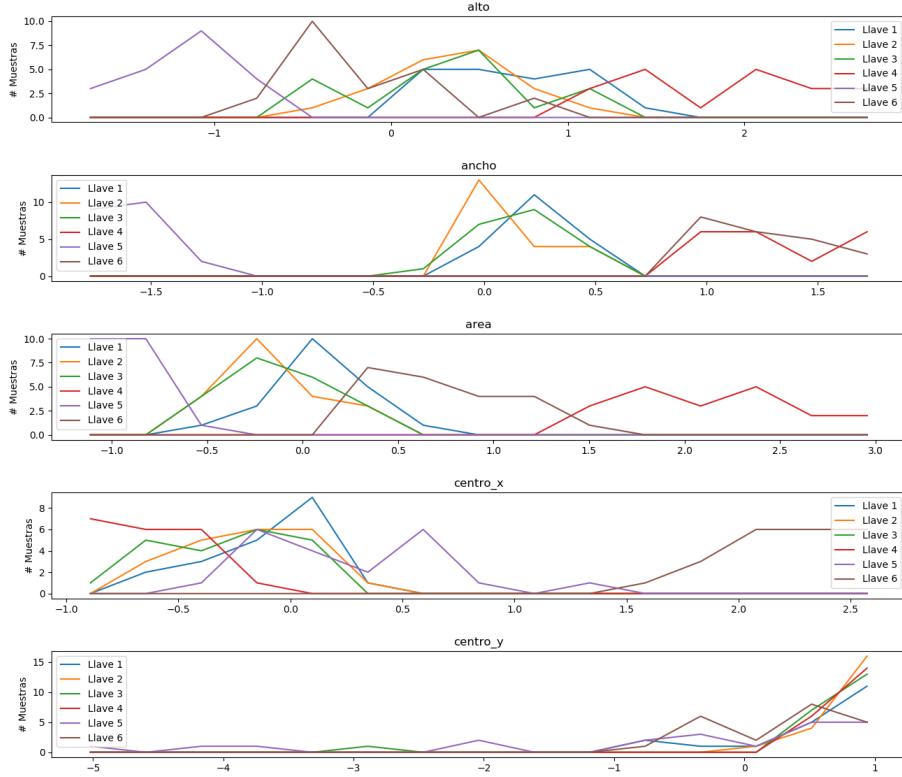


Figura 3.23: Histograma de las características estandarizadas.

Por tanto, una vez solucionada esta problemática podemos proceder a repetir el entrenamiento y evaluación del clasificador. En la figura 3.24 y la tabla 3.4 podemos ver los resultados obtenidos al introducir el escalado de características en el sistema de clasificación.

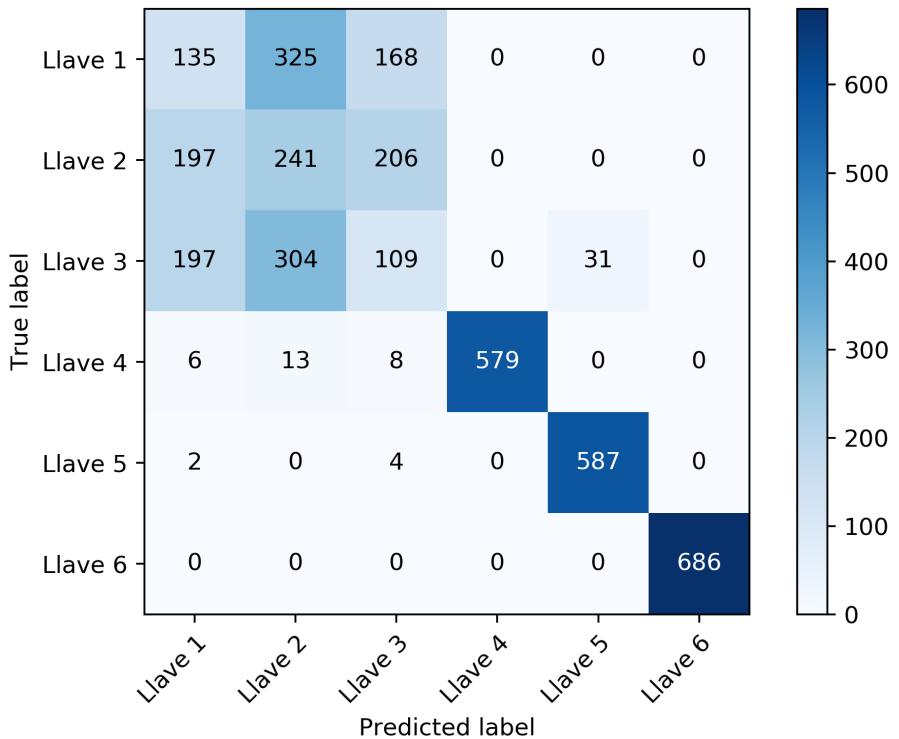


Figura 3.24: Matriz de confusión de la clasificación con KMeans utilizando estandarizado.

Llave	Precisión	Sensibilidad	f1-score
1	0.2514	0.2150	0.2318
2	0.2729	0.3742	0.3157
3	0.2202	0.1700	0.1919
4	1.0000	0.9554	0.9772
5	0.9498	0.9899	0.9694
6	1.0000	1.0000	1.0000
Media	0.6157	0.6174	0.6143

Tabla 3.4: Resultados de la clasificación con KMeans utilizando estandarizado.

Podemos ver como, gracias a los cambios introducidos, obtenemos resultados perfectos para la llave 6. A su vez, a pesar de que la cantidad de muestras de evaluación para cada clase resulta modificada al repetir la división aleatoria entre datos de entrenamiento y test, se puede observar como también mejoran los resultados en las llaves 4 y 5. Podemos medir la diferencia en el rendimiento de estas clases mediante la diferencia de los valores de las tablas 3.3 y 3.4. Estas diferencias se muestran para las llaves 4, 5 y 6 en la tabla 3.5. Podemos observar un gran incremento para todos los aspectos de la llave 6, un incremento considerable para la llave 4, y uno menor, aunque no despreciable, para la llave 5.

Llave	Precisión	Sensibilidad	f1-score
4	0.1980	0.1011	0.1499
5	0.0837	0.0256	0.0569
6	0.4582	0.3486	0.4085

Tabla 3.5: Diferencia entre las tablas 3.3 y 3.4 para las llaves 4, 5 y 6.

Conclusión

Como se ha podido comprobar, esta modificación aporta importantes incrementos en la precisión y sensibilidad del clasificador, especialmente para la llave 6, cuyos problemas pretendíamos solucionar en un principio. Por tanto, esta modificación ha sido incluida en el sistema de clasificación propuesto.

3.2.3. SVM

Problemas y propuestas

Debido a los problemas en la clasificación de las llaves 1, 2 y 3, que en ninguna de las clases resulta la llave adecuada la más frecuente en las predicciones para dicha clase, al hecho de que KMeans no hace uso de la información que aporta tener las llaves con sus respectivas etiquetas, así como a la linealidad de este método, se decidió probar SVM para ver si se conseguían mejorar los resultados.

Resultados

Como puede verse en la figura 3.25 y la tabla 3.6, conseguimos mantener los valores de sensibilidad para las llaves 4, 5 y 6 mientras los de precisión caen ligeramente. Sin embargo, obtenemos clasificaciones mucho más apropiadas para las llaves 1, 2 y 3. Esta vez podemos ver como para las predicciones de una clase la llave más común es precisamente la de dicha clase, a diferencia de con el modelo anterior en las figuras 3.14 y 3.24.

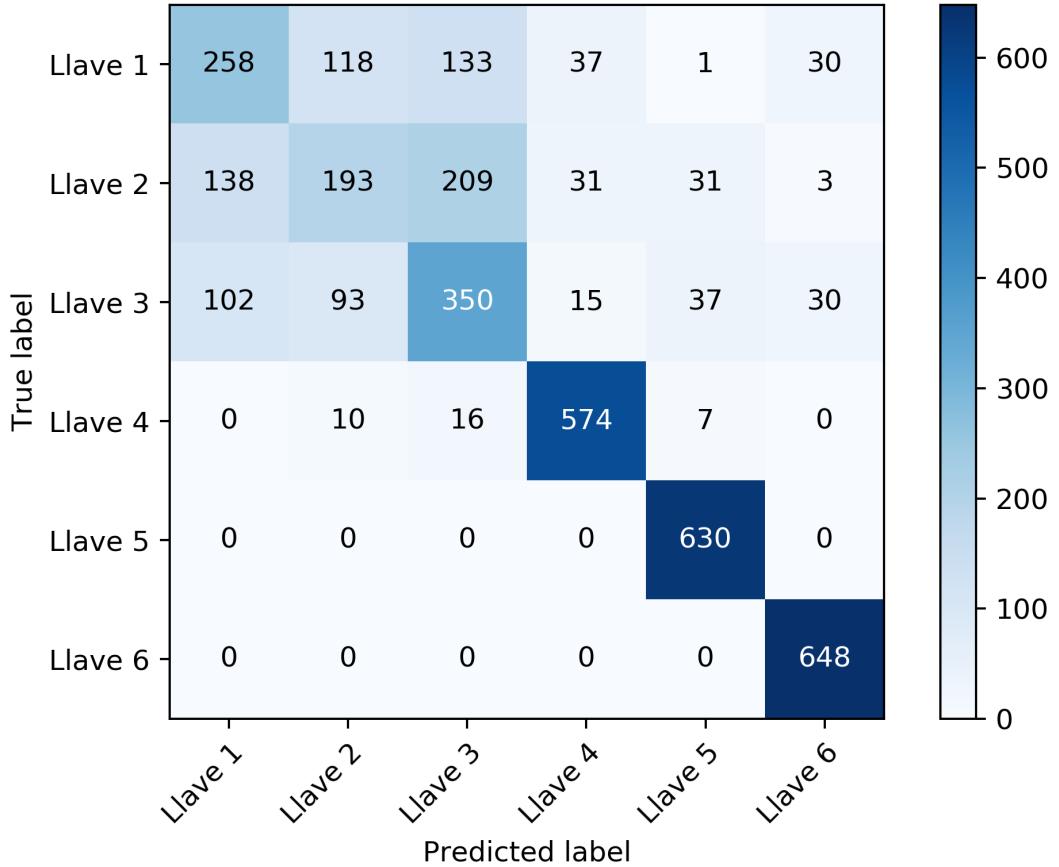


Figura 3.25: Matriz de confusión de la clasificación con SVM.

A su vez, en la tabla 3.6 se pueden observar métricas de estos resultados que son invariantes al reparto aleatorio de los datos utilizados, permitiendo un análisis más cuantitativo. Podemos ver como obtenemos valores de precisión de entre el 46 % y 52 % para las llaves 1, 2 y 3, muy distantes del rango entre el 22 % y 28 % obtenido en la tabla 3.4.

Llave	Precisión	Sensibilidad	f1-score
1	0.5181	0.4471	0.4800
2	0.4662	0.3190	0.3788
3	0.4944	0.5582	0.5243
4	0.8737	0.9456	0.9082
5	0.8924	1.0000	0.9431
6	0.9114	1.0000	0.9536
Media	0.6927	0.7117	0.6980

Tabla 3.6: Resultados de la clasificación con SVM.

De forma similar a la que se utilizó en la sección anterior, podemos resaltar las implicaciones de estos cambios realizando la diferencia entre las tablas 3.4 y 3.6, generando así la tabla 3.7. Aquí se aprecia claramente un incremento en las precisiones de las llaves 1, 2 y 3 a costa de un decremento, de menor nivel, para las llaves 4, 5 y 6. Sin embargo, la sensibilidad resulta incrementada para las llaves 1 y 3 en un 23 % y 38 % respectivamente a costa de un decremento del 5 % en la llave 2, mientras que las llaves 4, 5 y 6 apenas sufren modificaciones.

Llave	Precisión	Sensibilidad	f1-score
1	0.2667	0.2322	0.2482
2	0.1933	-0.0552	0.0632
3	0.2741	0.3882	0.3324
4	-0.1263	-0.0098	-0.0690
5	-0.0575	0.0101	-0.0263
6	-0.0886	0.0000	-0.0464
Media	0.0769	0.0942	0.0837

Tabla 3.7: Diferencia entre las tablas 3.4 y 3.6.

Conclusión

A la vista de los resultados, se puede argumentar que gracias al uso de la información que aporta tener las muestras etiquetadas se ha podido utilizar SVM para mejorar los resultados de la clasificación. Por tanto, en este punto se tomó la decisión de sustituir KMeans por SVM.

3.2.4. Caracterización del cifrado

Problemas y propuestas

A la vista de los resultados mostrados en la figura 3.25 y la tabla 3.6 se puede observar que los problemas de clasificación restantes son principalmente debidos a la confusión entre las llaves 1, 2 y 3. Esta confusión puede atribuirse a la escasa diferenciabilidad ofrecida por las características propuestas en la sección 3.1.3: '*Extracción de características*', lo cual puede comprobarse mediante el histograma mostrado en la figura 3.23.

Para tratar de abordar esta problemática se decidió extraer un nuevo conjunto de características que permitan diferenciar el cifrado de las distintas llaves. Para ello, se va a proceder a dividir la imagen en varias columnas de igual anchura para evaluar la cantidad de píxeles de las llaves en cada una de ellas. De esta forma, se pretende obtener una lista de valores que permita caracterizar la altura y posición de los distintos dientes presentes en la llave. Posteriormente, con la finalidad de permitir que estas métricas sean invariantes para imágenes obtenidas a diferentes distancias de la llave, se procede a normalizar la cantidad de píxeles en cada columna con el área de la misma.

Aprovechando la idea de obtener características invariantes frente a la distancia de captura de la imagen, se procede a normalizar el ancho y alto de la llave respecto al área total de la misma. A su vez, se elimina el área del conjunto de características utilizadas por el clasificador.

Implementación

El primer paso para realizar la implementación de esta propuesta consiste en, dada una imagen ya segmentada como la mostrada en la figura 3.12, calcular el ancho de la columna, en píxeles, para obtener el número de columnas deseado. Para ello se divide el ancho de la imagen entre la cantidad de columnas que se quieren generar redondeando el valor obtenido, si fuese necesario, al entero superior más cercano. Siendo N el número de columnas deseado, W el ancho de la imagen y C_w la anchura resultante de la columna, este procedimiento se puede realizar aplicando la expresión (3.4):

$$C_w = \left\lceil \frac{W}{N} \right\rceil \quad (3.4)$$

A continuación, para permitir que este conjunto de características sea a su vez invariante a la posición de la cabeza de la llave (derecha o izquierda), deberemos considerar el valor de la coordenada x del centroide para generar las columnas en una u otra dirección, calculando este valor sin aplicar las transformaciones de 3.1.3 referentes a esta invariabilidad (Tomando únicamente el valor inicial y dividiendo entre el ancho). Tomando la decisión de que las primeras columnas sean siempre las de la cabeza de la llave, evaluaremos si este valor es mayor que 0.5. En tal caso, supondremos que la cabeza de la llave está situada en la parte derecha de la imagen (ver figura 3.13), y, en consecuencia, generaremos las columnas

de derecha a izquierda.

En este punto se procede a recorrer la imagen contando la cantidad de píxeles pertenecientes a la llave en cada columna. Posteriormente, mediante el alto de la imagen y el ancho de la columna, se procede a calcular el área de dicha columna. Una vez obtenido este área se normaliza el valor obtenido durante el recuento haciendo que el nuevo valor sea invariante a la distancia a la que se obtenga la imagen. Podemos ver un ejemplo gráfico del resultado de este procedimiento en la figura 3.26.

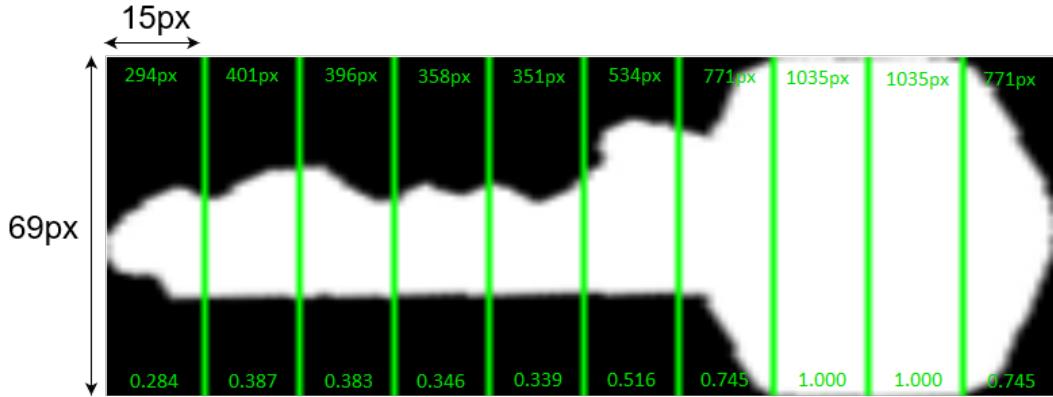


Figura 3.26: Ejemplo de caracterización del cifrado de una llave.

Resultados

Con el objetivo de obtener el mejor rendimiento posible para este nuevo conjunto de características deberemos encontrar cuál es el número óptimo de columnas a utilizar para caracterizar las llaves. Para ello haremos uso de los índices de separabilidad J_1 , J_2 y J_3 vistos en la teoría de la asignatura.

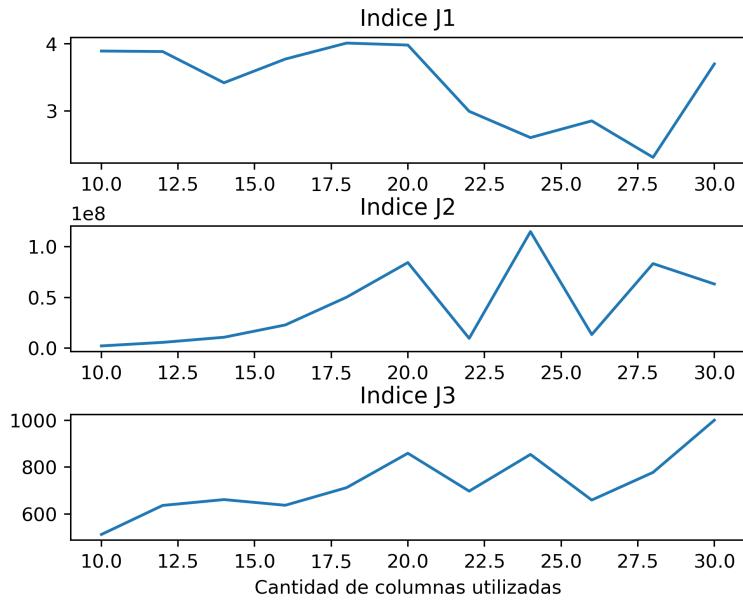


Figura 3.27: Índices de separabilidad en función del número de columnas utilizadas.

En la figura 3.27 se han representado estos índices en función del número de columnas utilizadas, considerando desde 10 hasta 30 columnas en incrementos de 2. En esta gráfica podemos ver como se obtienen los valores más elevados para $N = 20$ y $N = 30$. Con el fin de tratar de mantener la dimensionalidad del espacio de características al mínimo, mientras se maximiza la separabilidad de las muestras del *dataset*, se seleccionado $N = 20$ para las posteriores evaluaciones.

De cara a observar el rendimiento en la clasificación utilizando este nuevo conjunto de características se hará uso de las mismas figuras de mérito utilizadas en las secciones anteriores. En la figura 3.28 podemos observar la matriz de confusión resultante, en la que se observa claramente que se ha conseguido mejorar en gran medida los resultados anteriores, mostrados en la figura 3.25. Se puede observar como las llaves 1, 2 y 3 siguen siendo erróneamente clasificadas al ser confundidas entre sí, pero a diferencia del caso anterior, ahora estos casos son minoritarios.

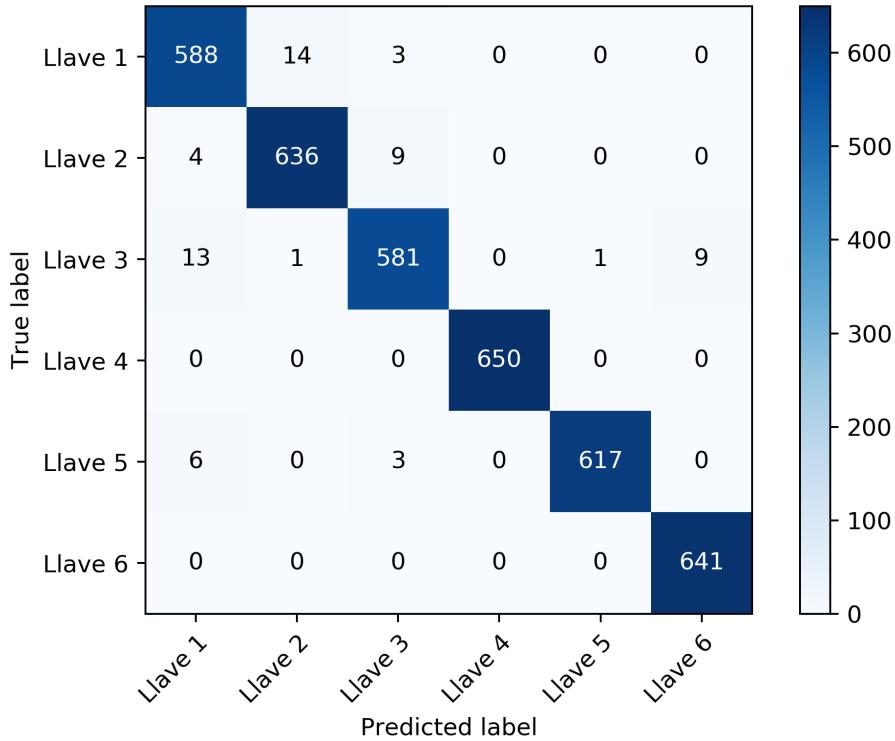


Figura 3.28: Matriz de confusión utilizando 20 columnas para la caracterización del cifrado.

Nuevamente, podemos observar los resultados de la clasificación mediante la tabla 3.8. En este caso destaca que se ha conseguido superar el 95 % tanto de precisión como de sensibilidad para todas las clases, mientras que anteriormente las llaves 1, 2 y 3 rondaban el 50 % (véase la tabla 3.6).

Llave	Precisión	Sensibilidad	f1-score
1	0.9624	0.9719	0.9671
2	0.9770	0.9800	0.9785
3	0.9748	0.9603	0.9675
4	1.0000	1.0000	1.0000
5	0.9984	0.9856	0.9920
6	0.9862	1.0000	0.9930
Media	0.9831	0.9830	0.9830

Tabla 3.8: Resultados de la clasificación utilizando 20 columnas para la caracterización del cifrado.

Adicionalmente, como en apartados anteriores, podemos observar la diferencia entre los resultados de clasificación actuales y previos a las modificaciones propuestas en este apartado. Estas se muestran en la tabla 3.9 y permiten observar que las modificaciones propuestas permiten incrementar los resultados para tanto precisión como sensibilidad en todas las clases, exceptuando la sensibilidad de la llave 5, para la cual anteriormente habíamos obtenido un valor de 1.000. Destacan, como era de esperar, las llaves 1, 2 y 3 con incrementos comprendidos entre el 40 % y 67 %, permitiendo obtener una clasificación de alta fiabilidad para este tipo de llaves.

Llave	Precisión	Sensibilidad	f1-score
1	0.4443	0.5248	0.4871
2	0.5108	0.6610	0.5997
3	0.4805	0.4021	0.4432
4	0.1263	0.0544	0.0918
5	0.1060	-0.0144	0.0488
6	0.0748	0.0000	0.0394
Media	0.2904	0.2713	0.2850

Tabla 3.9: Diferencia entre las tablas 3.8 y 3.6.

Conclusión

A la vista de los resultados expuestos es obvia la decisión de sustituir las características utilizadas hasta el momento por este nuevo conjunto. A su vez debemos destacar que, teóricamente, además de permitir una clasificación de mayor fiabilidad para todas las llaves, este conjunto de características es invariante frente a la distancia a la cual se captura la imagen, y por tanto también a la resolución de la cámara empleada en la captura.

3.2.5. Rellenado de huecos

Problemas y propuestas

En la sección "Segmentación" se ha realizado un relleno de huecos empleado una de las propiedades aportadas por la función *drawcountours*. Sin embargo, en el procesado de todo el *dataset* se detectaron algunos problemas asociados a una detección de contornos externos excesiva tal y como se puede apreciar en la figura 3.29.

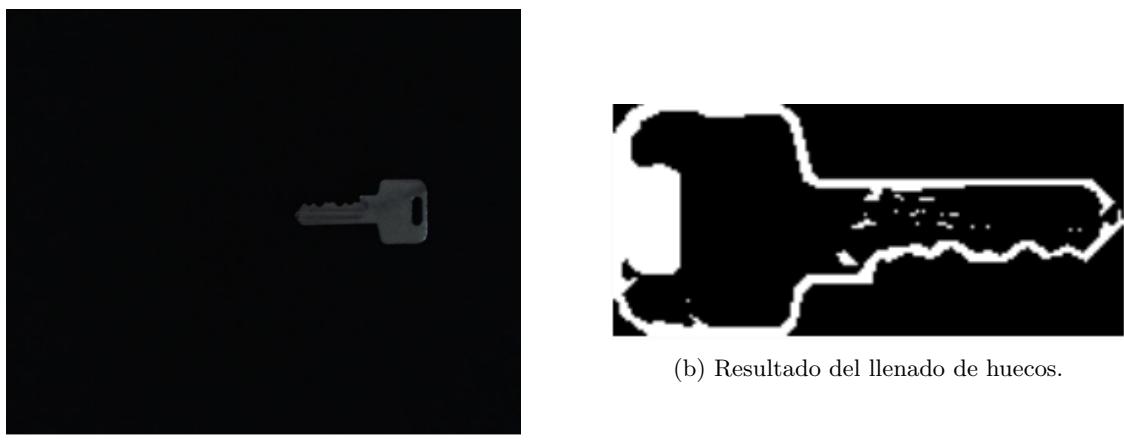
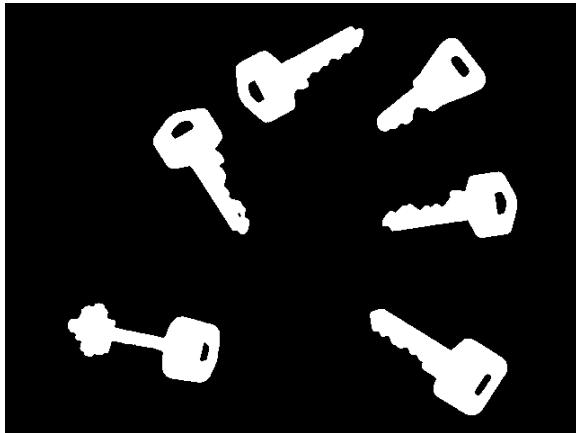


Figura 3.29: Problemática del llenado de huecos aportado por *drawcountours*.

Para resolver este problema, se ha optado por implementar un método diferente que consiga uniformizar el resultado del llenado de huecos para la totalidad de las llaves.

Implementación

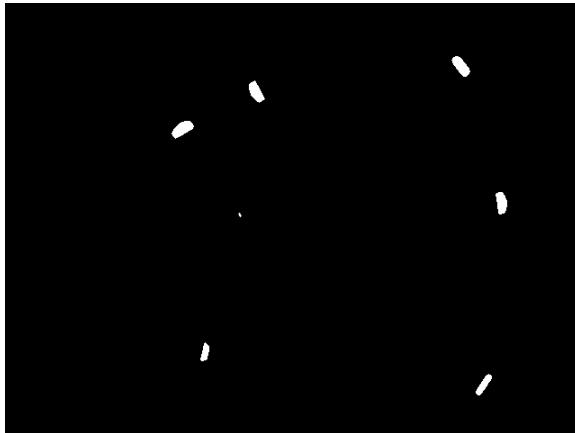
Para ello, se ha empleado la función de OpenCV *FloodFill* que permite rellenar los huecos dentro de una región conexa a partir de una semilla. Como es necesaria especificar una semilla, se ha optado por llenar la imagen a la salida del cierre a partir del punto (0,0) de la matriz y obtener la representación complementaria. De este modo, si se realiza la operación lógica **OR** con la salida del cierre se obtendrá el llenado de huecos necesario. El proceso se ha representado en la figura 3.30.



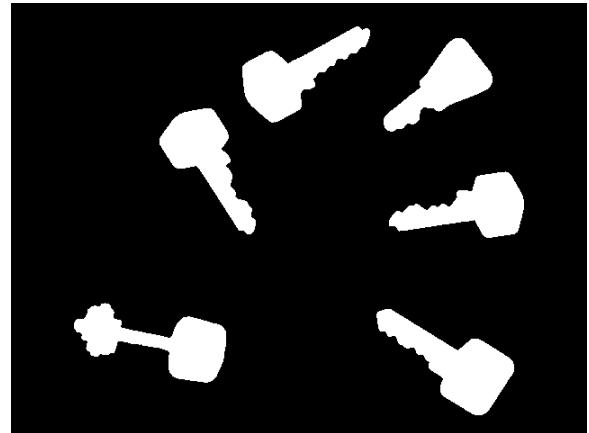
(a) Salida del cierre



(b) Imagen salida de la función *FloodFill*.



(c) Imagen complementaria de b)



(d) Resultado de la operación OR.

Figura 3.30: Implementación del relleno de huecos.

Resultados

Anteriormente, en la figura 3.27 se han representado los índices de separabilidad en función del número de columnas utilizadas, considerando desde 10 hasta 30 columnas en incrementos de 2 empleando el llenado de huecos aportado por *drawcountours*. Para poder constatar que la nueva implementación se trata de una mejora para la totalidad del dataset y no para algunas imágenes en particular, se han obtenido los índices de separabilidad J_1 , J_2 y J_3 (Ver figura 3.31) y la diferencia con los mismos índices del llenado de huecos previo (Ver figura 3.32).

Tal y como se puede apreciar, la diferencia en todos los casos es positiva por lo que el hecho de mejorar el llenado de huecos permite estabilizar el número de píxeles contados por columnas y por tanto, aumentar la separabilidad entre clases. Por otro lado, es posible apreciar como el número de columnas que permite una mayor separabilidad con la nueva implementación sigue siendo 20, del mismo modo que ocurría para el llenado de huecos aportado por *drawcountours*.

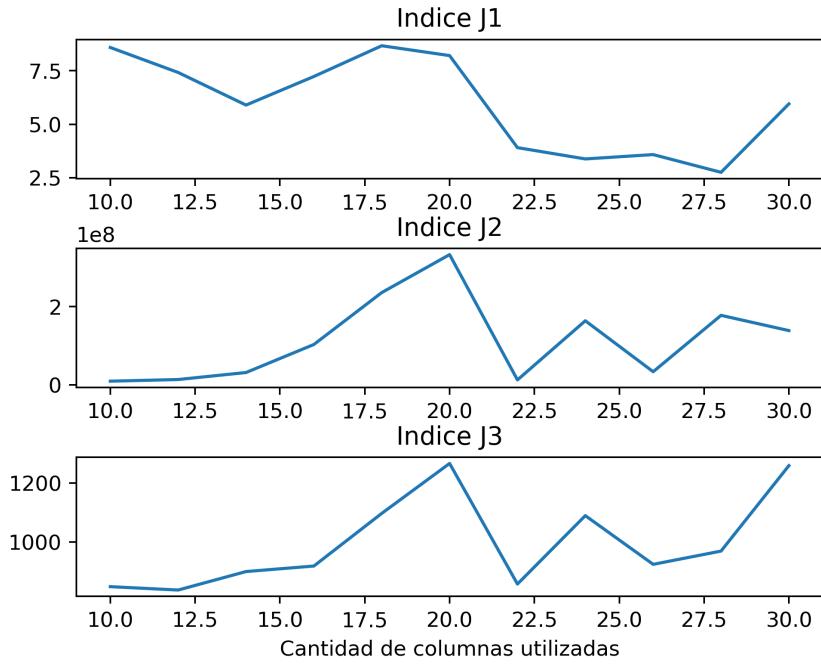


Figura 3.31: Índices de separabilidad en función del número de columnas empleando el nuevo rellenero de huecos.

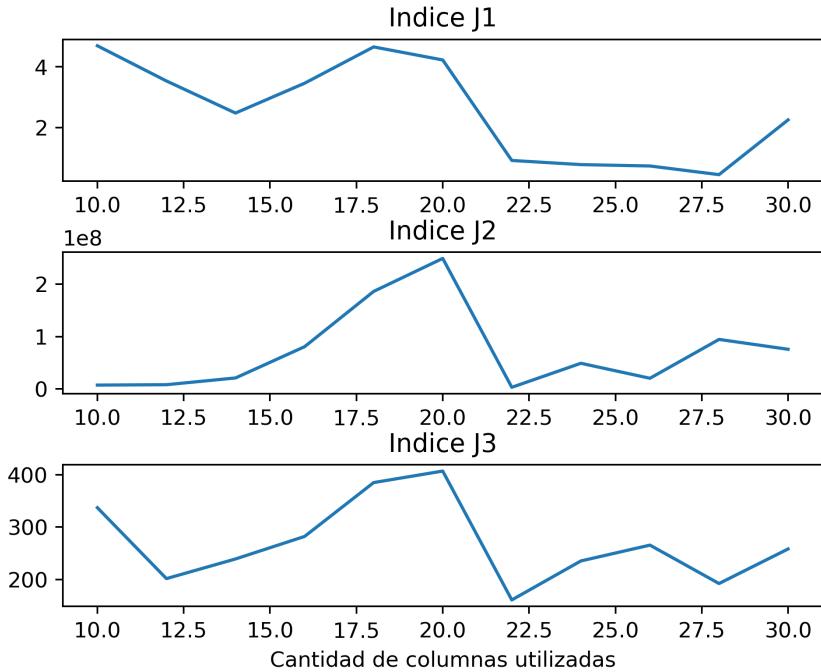


Figura 3.32: Diferencia entre los índices de separabilidad en función del número de columnas empleado el nuevo rellenador de huecos y el aportado por *drawcountours*.

Conclusión

A la vista de los resultados expuestos en la figura 3.31, se ha optado por adaptar el nuevo cambio en el rellenero de huecos del segmentador en favor de una mayor separabilidad de las características empleadas en el clasificador y homogeneizado al mismo tiempo la segmentación obtenida para los diferentes tipos de llaves.

3.2.6. Filtrado de llaves interferentes

Problemas y propuestas

Debido a que el número de llaves que pueden aparecer en el plano de captura puede ser mayor que uno, cabe la posibilidad de que, una vez se ha detectado una región conexa, rotado y recortado, aparezca algún residuo de otra llave en la región de interés obtenida. Este fenómeno ha sido representado en la figura 3.33. Debido a que se ha caracterizado el cifrado de la llave cuantificando el número de píxeles que aparecen en una división por columnas (Ver figura 3.26) este fenómeno puede alterar la fiabilidad de la característica obtenida. Es por esto por lo que se ha implementado un filtrado de llaves interferentes para las regiones de interés.

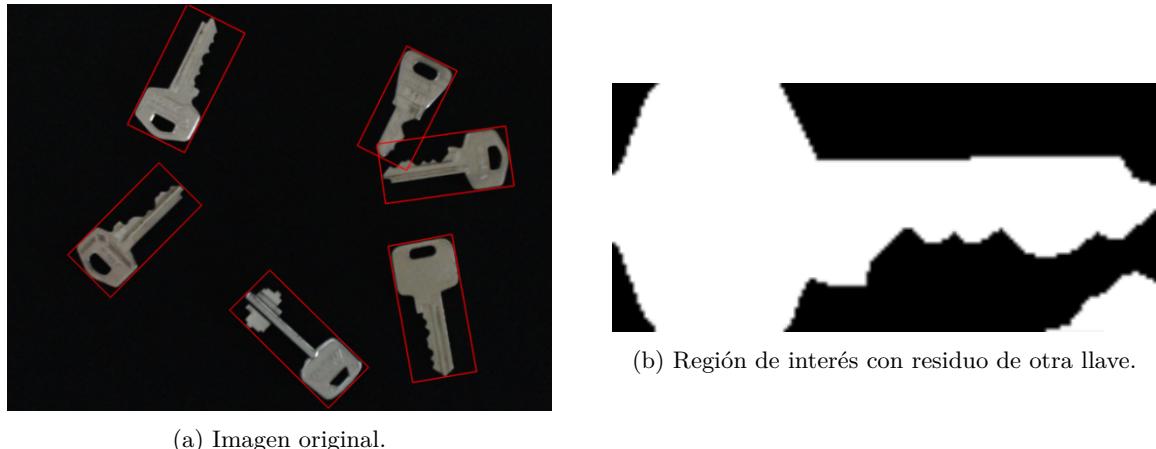


Figura 3.33: Problemática de las llaves interferentes.

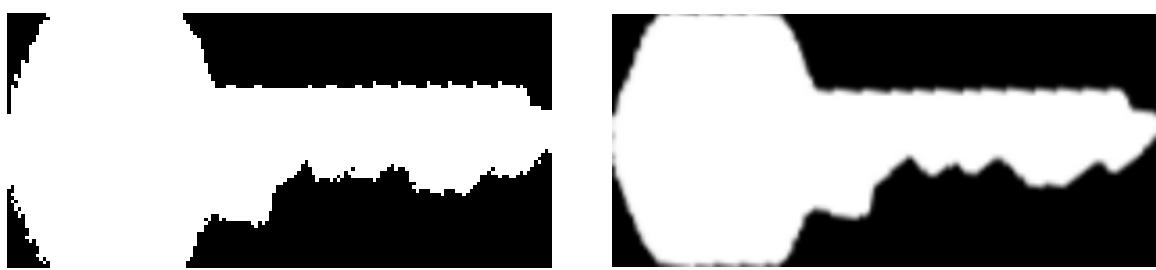
Implementación

Para ello, se ha empleado la función de `OpenCV connectedComponentsWithStats` que permite detectar regiones conexas y obtener características de las mismas además de una matriz de las mismas dimensiones que la imagen original pero *marcada* con diferentes números, pudiendo diferenciar de este modo las diferentes regiones encontradas.

El planteamiento empleado consiste en filtrar todas aquellas regiones conexas detectadas en la región de interés que no sean la de área máxima. Es por esto por lo que es necesaria obtener el marcador que identifica a la mayor región conexa para, con ayuda de la sentencia `compare`, eliminar el resto.

Resultados

En la primera implementación se constató que el método de filtrado de `OpenCV` introducía ruido en la región de interés filtrada. Para resolver este problema, se optó por realizar una operación lógica de tipo AND con la imagen original. El proceso ha sido representado en la figura 3.34.



(a) Ruido introducido por el filtrado de regiones conexas. (b) Resultado de la operación lógica AND.

Figura 3.34: Resultado final del filtrado de llaves interferentes.

Conclusión

A la vista de las figuras expuestas se ha optado por incluir el filtrado de llaves interferentes debido a que permite eliminar regiones de píxeles que aportarían ruido a la hora de obtener las columnas que definen el cifrado, favoreciendo de este modo, la clasificación y robustez del sistema final.

3.2.7. Ajustes de las condiciones de captura de la cámara

Problemas y propuestas

Durante la realización de las primeras pruebas de contexto se constató que la segmentación implementada era sensible a las condiciones de iluminación de la estancia. Cambios en las sombras o reflejos producidos por las llaves o en su superficie propiciaban una umbralización diferente que dificultaba la correcta clasificación del tipo de llave. Al variar el resultado de la segmentación, el rectángulo de área mínima que contiene a las regiones conexas también lo hace. De este modo, las propiedades obtenidas relativas al número de píxeles por columnas varía de un fotograma a otro. Es por esto por lo que se optó por añadir a la interfaz gráfica controles manuales que permitiesen ajustar las condiciones de captura de acuerdo con la iluminación de la estancia.

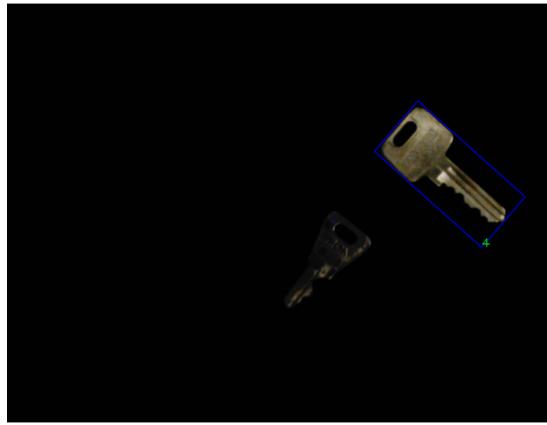
Implementación

Concretamente se ha permitido al usuario ajustar el brillo, el contraste y el enfoque de la cámara manualmente pudiendo configurar su valor desde 0 hasta 255 en intervalos de 5. Para ello se ha requerido el uso de tres *sliders*. Esta nueva funcionalidad ha permitido estabilizar los resultados aportados por el sistema de clasificación automática al propiciar una segmentación más homogénea.

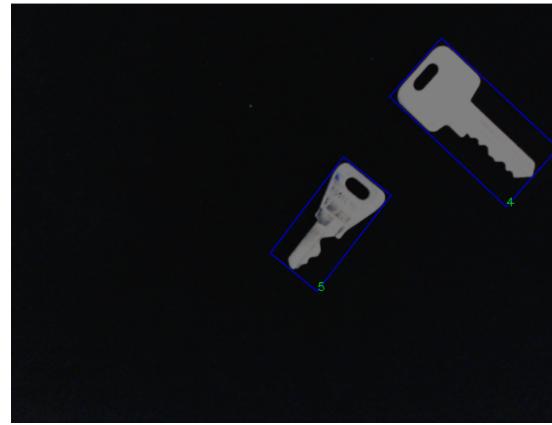
Resultados

El hecho de poder ajustar manualmente los parámetros de la cámara permite asegurar que la segmentación realizada por el sistema de visión artificial es satisfactoria favoreciendo así la detección de todos los elementos presentes en el plano de captura.

En la figura 3.35 es posible apreciar como en el caso (a) las condiciones de iluminación de la estancia en la que fue capturada la imagen eran pobres y el segmentador no era capaz de detectar satisfactoriamente la llave de tipo 5 presente en el plano de captura. Sin embargo, gracias a los ajustes de enfoque, brillo y contraste, en el caso (b) es posible detectar y clasificar adecuadamente la misma llave de acuerdo a las condiciones del entorno.



(a) Llave 5 no detectada.



(b) Resultado del ajuste de las condiciones de contorno.

Figura 3.35: Detección favorecida por el ajuste de las condiciones de contorno.

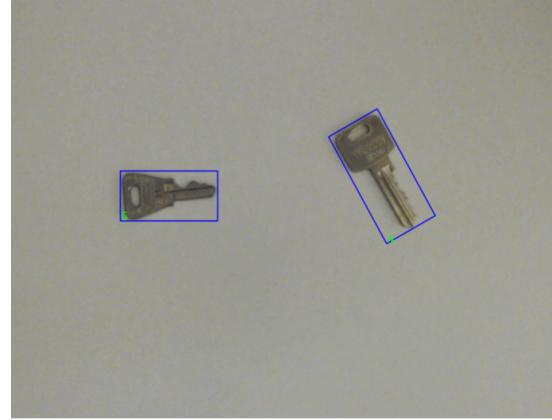
Resultados: Análisis con fondo blanco

Una vez se implementó el ajuste manual de algunas de las propiedades de la cámara se decidió evaluar el comportamiento del sistema en tiempo real variando el fondo empleado en el plano de captura. Concretamente, en lugar de emplear el fondo negro planteado inicialmente, se optó por utilizar el fondo

blanco de las mesas del laboratorio. En un primer momento, si los ajustes de la cámara son ajustados por defecto por la misma, las llaves se pierden en el fondo por el filtrado de área y ninguna región conexa es detectada. Sin embargo, si se realiza un ajuste de brillo, enfoque y contraste buscando una mayor diferencia entre el fondo del plano de captura y la superficie de las llaves, la segmentación mejora y los elementos de interés son identificados. Un ejemplo de este fenómeno ha sido representado en la figura 3.36. Sin embargo, cabe mencionar que a la vista de los resultados experimentales, la clasificación es más inestable respecto aquella que es realizada empleando un fondo negro.



(a) Imagen con ajuste automático.



(b) Resultado sobre el ajuste de fondo.

Figura 3.36: Detección favorecida por el ajuste de las condiciones de contorno.

3.2.8. Limpieza de bordes

Problemas y propuestas

Para poder aportar más datos al clasificador y favorecer así su estabilidad, en colaboración con el resto de grupos de la asignatura se conformó un *dataset* común con el conjunto de todas las imágenes capturadas. A la hora de evaluar el comportamiento del clasificador con el resto de imágenes se constató que algunas presentaban peculiaridades que comprometían el correcto comportamiento del segmentador. Un caso particular ha sido representado en la figura 3.37. Tal y como se puede apreciar la imagen capturada esta formada por diferentes fondos; Un fondo negro, un fondo blanco y una porción en el borde inferior constituido por las anillas del cuaderno.



Figura 3.37: Imagen con varios fondos.

La binarización obtenida de la imagen presentada en la figura 3.37 puede visualizarse en la figura 3.38. Tal y como se puede apreciar, además de las correspondientes a las llaves y al ruido generado por las manchas del cuaderno, se ha generado una región en blanco correspondiente a la mesa y otra al anillado. Debido al filtrado por área basado en 0.25 veces el de la mayor región conexa encontrada en el plano de

captura, las llaves serían eliminadas y por lo tanto no sujetas a la clasificación. Para evitar este tipo de casuísticas se optó por implementar una limpieza de bordes.

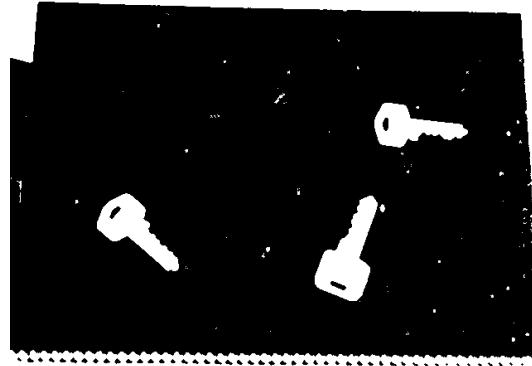


Figura 3.38: Binarización de imagen con varios fondos.

Implementación

Para la limpieza de bordes se ha generado un marco de anchura 1 píxel en los extremos de la imagen que recorre todos los laterales de la captura. De este modo, el nuevo marco entrará en contacto con todas las componentes conexas que estén en el borde de la imagen. Con ayuda de la sentencia *floodfill* (cuya función es llenar regiones conexas con un determinado color) y especificando una semilla sobre el píxel (0,0) es posible llenar el marco y todas las regiones conexas con las que ha entrado en contacto, de color negro, colapsándolas así con el fondo.

Resultados

El resultado de la limpieza de bordes aplicada sobre la imagen binarizada presentada en la figura 3.38 puede apreciarse en la figura 3.39. Con la nueva imagen es posible realizar una detección de regiones conexas que, debido a la eliminación de los bordes evitarán que se pierdan las llaves. Los elementos que aparecen en el fondo junto a las llaves serán eliminados por el filtrado de área explicado con anterioridad.

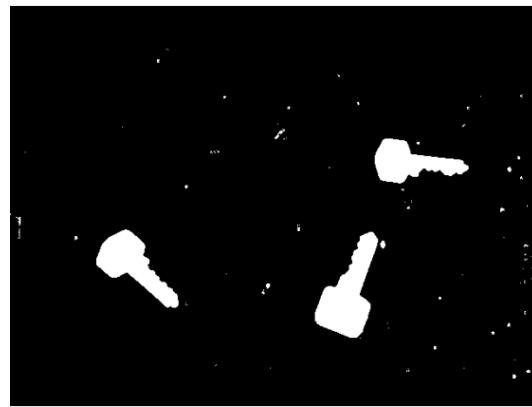


Figura 3.39: Resultado de la limpieza de bordes aplicada sobre la imagen binarizada (Ver 3.38).

Por otro lado, la limpieza de bordes ha permitido solventar satisfactoriamente la segmentación de otras casuísticas como la presentada en la figura 3.40 donde, además de la variación en el color del fondo comentada con anterioridad, está presente una mano que tras el proceso de binarización y filtrado por área evitaría la detección de las llaves.

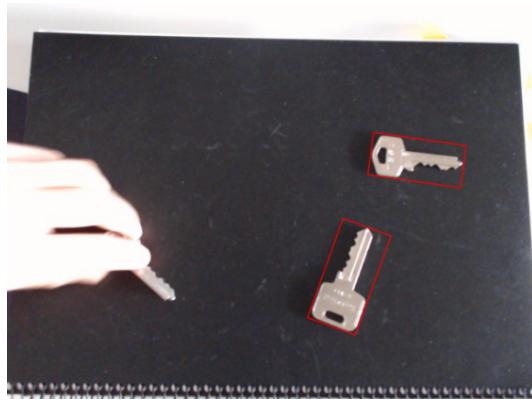


Figura 3.40: Detección de llaves en entorno complejo con presencia de manos.

Conclusión

La implementación de la limpieza de bordes ha permitido la generalización del segmentador para diferentes casuísticas y entornos complejos. Es por esto por lo que se ha considerado una implementación beneficiosa para el clasificador final.

Por otro lado, se debe tener en cuenta que cualquier llave de interés que este en contacto con el borde a la hora de la capturar las imágenes será filtrada por la limpieza de bordes y no será clasificada. Este fenómeno ha sido representado en la figura 3.41

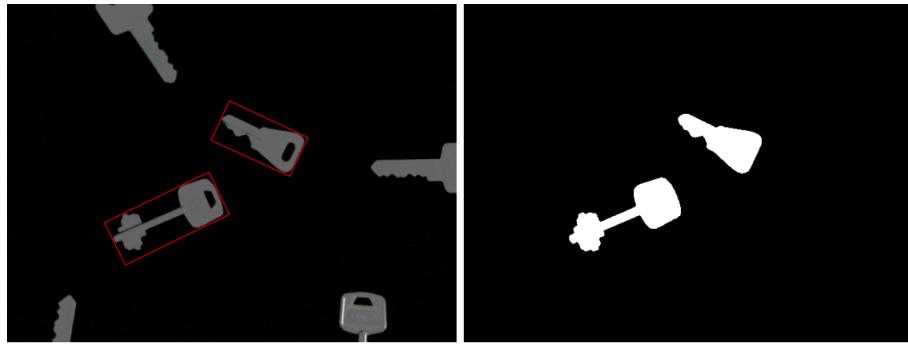


Figura 3.41: Compromiso con la posición de las llaves en el plano de captura.

3.2.9. CNN: Red neuronal convolucional

Problemas y propuestas

Debido al potencial y actual popularidad de las redes neuronales convolucionales para la resolución de problemas de visión artificial se decidió probar el rendimiento de estas en la resolución del problema de clasificación en cuestión. A su vez, cabe destacar que mediante el uso de este tipo de redes no sería necesaria la caracterización de las distintas llaves mediante un proceso de extracción de características, pues este tipo de redes se alimentan directamente con las imágenes, pudiendo ser estas en blanco y negro o incluso en color. Por lo tanto, en caso de que se quisieran añadir nuevas llaves al sistema de clasificación o incluso cambiar el tipo de elemento de clasificación de llaves a otro conjunto de objeto, los ajustes en el código del programa serían mínimos.

Debido a la complejidad interna de este tipo de redes neuronales se ha considerado conveniente incrementar la cantidad de fotos utilizadas para entrenamiento, evaluación y test de la misma. Para ello se ha optado por utilizar, de las fotos capturas por el resto de compañeros de clase, aquellas que por sus condiciones de iluminación y contorno sobrevivían a la fase de segmentación. A su vez, se procedió a la captura de un segundo conjunto de imágenes de entrenamiento, aportando aproximadamente 150 nuevas imágenes para cada llave y utilizando el mismo plano de captura que en el primer conjunto pero variando

las condiciones de iluminación a lo largo de las distintas imágenes. De esta forma, se consiguió aumentar el conjunto de datos de entrenamiento tal y como se muestra en la tabla 3.10.

Conjunto de datos	Llave 1	Llave 2	Llave 3	Llave 4	Llave 5	Llave 6	Total
Original	20	21	21	21	21	22	126
Compañeros	82	89	91	89	82	86	519
Nuevo	141	159	146	130	153	149	878
Total	243	269	258	240	256	257	1523

Tabla 3.10: Composición del nuevo *dataset* ampliado.

Implementación

Puesto que no se ha conseguido encontrar la funcionalidad de entrenamiento en el módulo de redes neuronales de **OpenCV**, se ha optado por realizar la implementación de la red neuronal utilizando **Keras**, una librería de redes neuronales de código abierto hecha en **Python** que utiliza, entre otras, **Tensorflow** como *backend*. Para la red neuronal se han utilizado capas convolucionales (*Convolutional layer*), *Average pooling*, *Global average pooling* y completamente conectadas (*Fully connected layer* o *Dense Layer*). La arquitectura empleada puede verse en la figura 3.42.

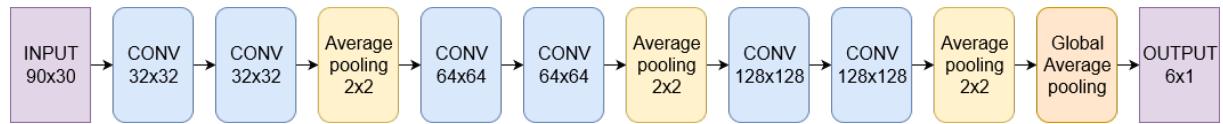


Figura 3.42: Arquitectura de la CNN.

En primer lugar se hace necesario exportar las imágenes una vez han sido segmentadas para poder utilizarlas desde **Python**. Para ello, se implementó en **C++** la funcionalidad de segmentar y guardar de nuevo la imagen recortada como **png**. Dado que se han realizado las pruebas tanto con imágenes umbralizadas como en **RGB** se implementaron ambas opciones.

El modelo se construye y entrena utilizando **Keras**. Para el entrenamiento se utilizaron las imágenes extraídas realizando una separación del 50 % para entrenamiento, 30 % para evaluación y 20 % para test, manteniendo en cada conjunto la proporción de muestras por clase del conjunto original. En primer lugar, la imagen debe ser redimensionada a un tamaño constante para poder alimentar la red neuronal. Se ha escogido la resolución de 90x30, pues no resulta en un tamaño excesivo y a su vez permite diferenciar las llaves visualmente, por lo que aporta información suficiente. A su vez, para incrementar la cantidad de datos de entrenamiento y evaluación, se propuso realizar la rotación en 180° y volteado por el eje horizontal, consiguiendo 4 imágenes distintas por cada imagen original. Se puede ver un ejemplo en la figura 3.43.

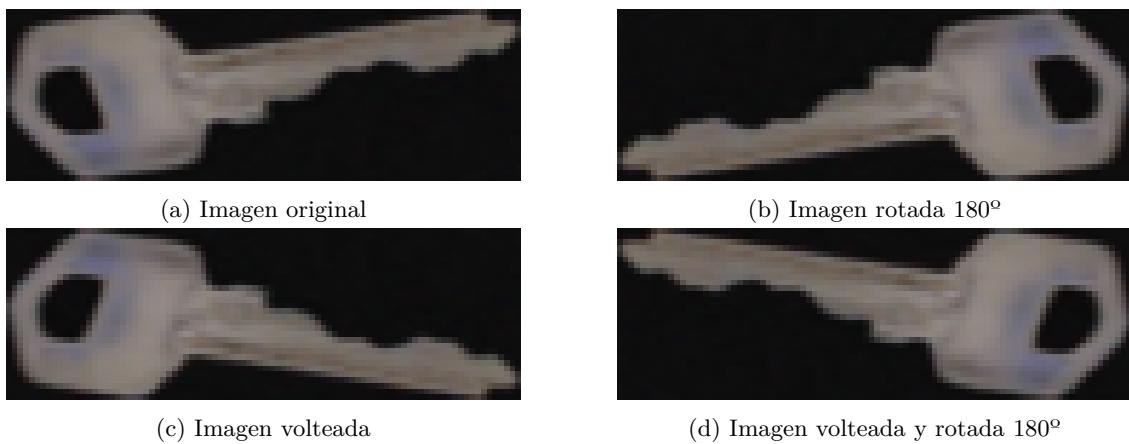


Figura 3.43: Ejemplo de obtención de 4 imágenes a partir de 1.

Una vez se ha entrenado el modelo se exporta en formato `h5` y posteriormente se convierte a formato `pb` [8]. Una vez se dispone del modelo entrenado en formato `pb` se puede proceder a importarlo desde C++ con `OpenCv` y utilizarlo para realizar predicciones.

Cabe destacar que la diferencia durante la construcción de la red neuronal, para los casos de imágenes umbralizadas e imágenes a color, es únicamente la dimensionalidad del *input* de la red. En el caso de imágenes umbralizadas esta es de $1 \times 90 \times 30$, mientras que en RGB aumenta a $3 \times 90 \times 30$. A su vez, en `OpenCv` se puede utilizar la función `blobFromImages` para construir una matriz de $N \times C \times W \times H$, donde N es el número de imágenes, C el número de canales de color y W y H el ancho y alto respectivamente. El resultado de esta función se utilizará como *input* de la red para realizar predicciones de varias imágenes simultáneamente. Sin embargo, debido a que existe una alteración en el orden de las dimensiones de la capa de entrada entre `Keras` y `OpenCV`, cuando se construye la red se establece el *input* como $N \times H \times W \times C$, de forma que tras dicha alteración quedan en orden $N \times C \times W \times H$, facilitando posteriormente el proceso de predicción.

Resultados

En primer lugar, se ha procedido a evaluar si resulta adecuado generar múltiples imágenes a partir de cada imagen original, como se ha mostrado en la figura 3.43. Para ello, en la figura 3.44 se muestra el progreso del entrenamiento a lo largo de 60 épocas, para los casos de imágenes umbralizadas y en RGB con los datos originales y con incremento de imágenes.

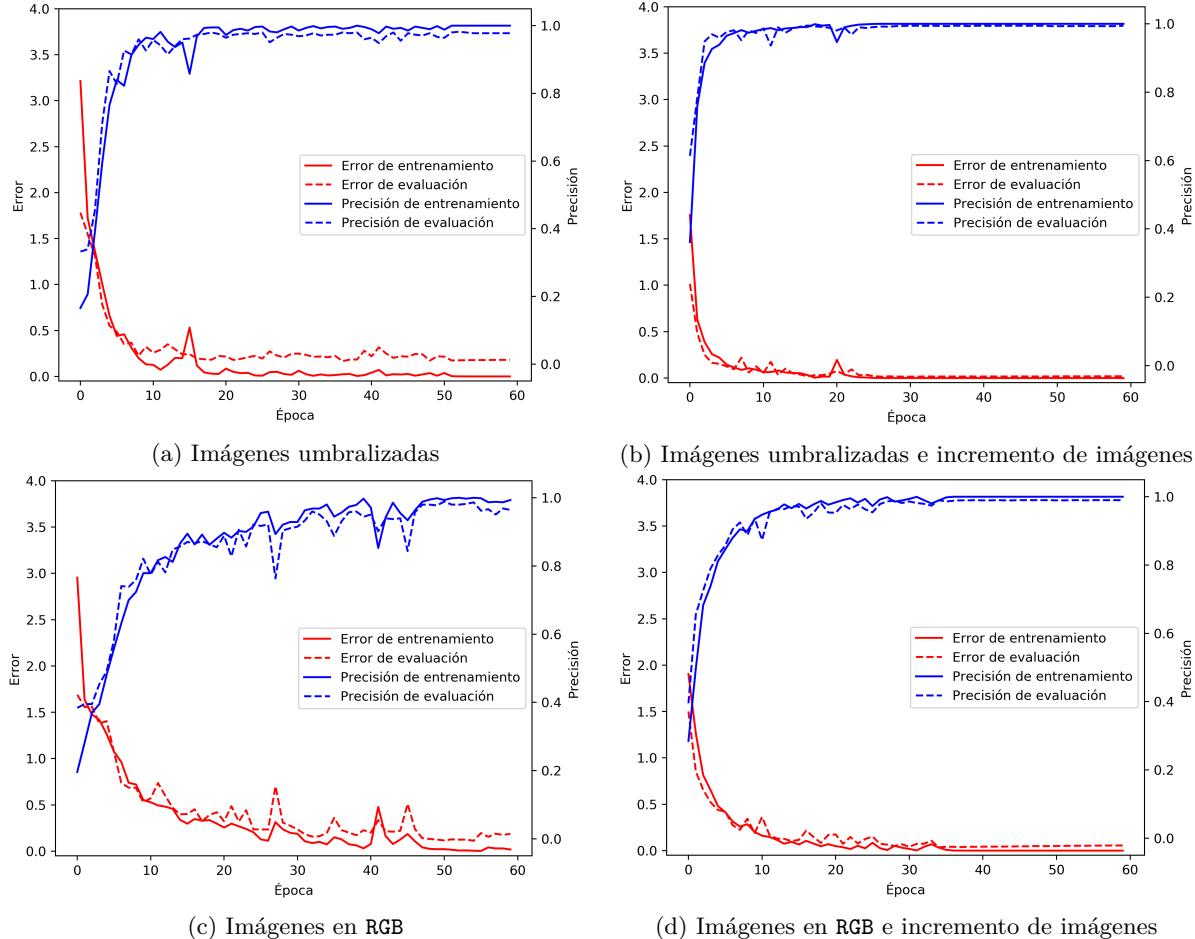


Figura 3.44: Proceso de entrenamiento de la CNN.

En primer lugar se puede observar que cuando se hace uso del incremento de imágenes la red requiere de un menor número de épocas para estabilizarse en el proceso de entrenamiento, dado que se le aplica una cantidad de datos 4 veces mayor. A su vez, también vemos como en el inicio del entrenamiento el error cometido resulta de menor magnitud y en el final presenta una menor diferencia en error y precisión entre

entrenamiento y evaluación. Por tanto, parece resultar conveniente aplicar esta técnica de incremento de imágenes.

En segundo lugar, cabe analizar la diferencia de comportamiento cuando se utilizan imágenes umbralizadas o en color. A la vista de la figura 3.44, se puede observar que el uso de imágenes en RGB supone una mayor complejidad para la red neuronal, pues presenta mayor variabilidad y requiere de mayor cantidad de épocas para estabilizarse. Esto es debido al incremento de la dimensionalidad de la matriz de datos de entrada junto con la escasa información que aporta el color en este problema de clasificación concreto.

En la tabla 3.11 se muestran los resultados de la clasificación para ambos casos, mostrando que el uso de los planos de color no solo no genera mejores resultados sino que quedan por debajo de los obtenidos con imágenes umbralizadas, a pesar de los buenos resultados obtenidos en ambos casos. A su vez, cabe destacar que el uso de imágenes umbralizadas supone una menor carga computacional, favoreciendo la clasificación en tiempo real.

Llave	Imágenes umbralizadas			Imágenes RGB		
	Precisión	Sensibilidad	f1-score	Precisión	Sensibilidad	f1-score
1	0.9800	1.0000	0.9899	0.9796	0.9796	0.9796
2	1.0000	0.9815	0.9907	1.0000	0.9630	0.9811
3	1.0000	0.9804	0.9901	0.9615	0.9804	0.9709
4	0.9792	1.0000	0.9895	0.9792	1.0000	0.9895
5	1.0000	1.0000	1.0000	0.9800	0.9608	0.9703
6	1.0000	1.0000	1.0000	0.9808	1.0000	0.9903
Media	0.9932	0.9936	0.9934	0.9802	0.9806	0.9803

Tabla 3.11: Resultados de la clasificación con imágenes umbralizadas y en RGB.

Finalmente, se representa en la figura 3.45 la matriz de confusión resultante para la red con imágenes umbralizadas y uso de la técnica de incremento de imágenes. Destacar que esta técnica se ha utilizado únicamente para ampliar el conjunto de datos de entrenamiento y evaluación mientras que el de test se compone únicamente de imágenes originales. Se observa que se obtienen muy buenos resultados, como se podía prever a partir de la tabla 3.11

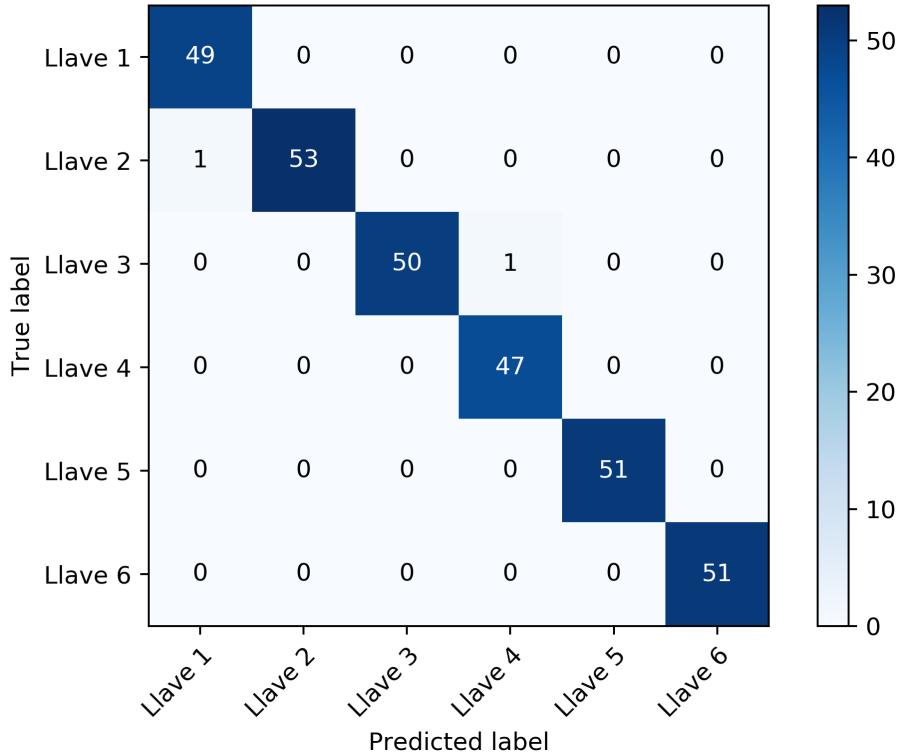


Figura 3.45: Matriz de confusión de la CNN con imágenes umbralizadas.

Conclusión

A la vista de los resultados obtenidos y de la sencillez que ofrecería este clasificador para futuras incorporaciones de nuevos tipos de llaves, así como la posibilidad de incorporar la información de los planos de color si estos resultasen relevantes, se ha procedido a implementar la red neuronal en el sistema de clasificación. Sin embargo, dado que los resultados obtenidos anteriormente por SVM con las características extraídas para caracterizar el cifrado de la llave (véase la figura 3.28) son suficientemente buenos y que a su vez estos posiblemente se consigan mejorar cuando se aplique el conjunto de datos incrementado mostrado en la tabla 3.10, junto con la notable diferencia observada en el tiempo requerido para realizar la clasificación en el caso de la CNN, se ha optado por, en lugar de sustituir el clasificador anterior, incorporar la posibilidad de seleccionar cual de ambos quiere utilizarse.

3.3. Sistema final

3.3.1. Captura de imágenes

Para la generación del *dataset* se han mantenido las condiciones de contorno expuestas en la sección 3.1.1 : 'Captura de imágenes'. Las imágenes han sido obtenidas en las mismas circunstancias de altura (21 cm) e iluminación para generar el conjunto de datos del entrenamiento. El fondo empleado corresponde con una funda de ordenador de color negro tejida en neopreno que permite realizar el contraste del mismo respecto a las llaves a detectar y clasificar.

En favor de una mayor estabilidad e independencia del clasificador a las condiciones de iluminación de la estancia, la versión final del sistema de visión artificial permite ajustar el contraste, brillo y enfoque de la cámara a la hora de obtener generar un nuevo *dataset* o clasificar en tiempo real nuevas imágenes. Esta funcionalidad ha permitido independizar el clasificador del fondo negro planteado inicialmente permitiendo su correcto funcionamiento en distintos entornos como el fondo blanco de las mesas del laboratorio. (Ver figura 3.36).

3.3.2. Segmentación

El estado final del segmentador ha sido representado en la figura 3.46. Tal y como se puede apreciar, respecto al planteamiento inicial se ha implementado una limpieza de bordes, un filtrado de llaves interferentes y se ha mejorado el llenado de huecos planteado en la etapa de prototipado. Por otro lado, la mejora propuesta para una segmentación de elementos próximos entre sí empleando *watershed* ha sido descartada debido a la sobresegmentación experimentada.

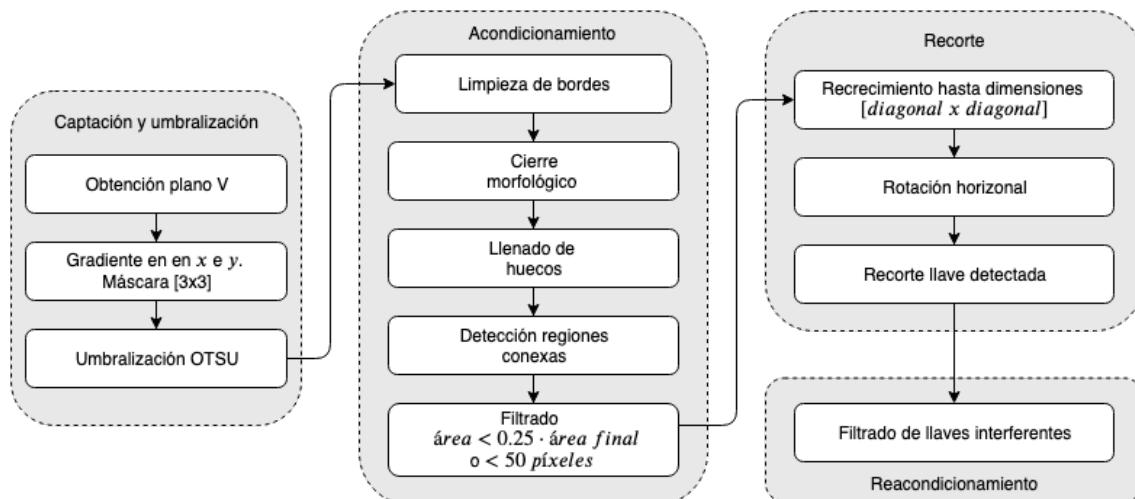


Figura 3.46: Diagrama de bloques del proceso de segmentación.

3.3.3. Extracción de características

En el estado final del sistema se extraen un total de 24 características, todas ellas invariantes a la distancia y resolución de la imagen capturada. Estas se obtienen a partir de la imagen recortada y umbralizada, y son las siguientes:

- Altura normalizada respecto al área total
- Anchura normalizada respecto al área total
- Coordenada x del centroide normalizada (Véase la sección 3.1.3 : 'Extracción de características')
- Coordenada y del centroide normalizada (Véase la sección 3.1.3 : 'Extracción de características')
- 20 columnas de caracterización del cifrado (Véase la sección 3.2.4 : 'Caracterización del cifrado')

A todas ellas se les aplica posteriormente un estandarizado para evitar que la variación en las magnitudes entre las distintas características suponga un problema para el clasificador, tal y como se expuso en la sección 3.2.2 : 'Escalado de las características'.

3.3.4. Clasificación

El estado final del sistema consta de dos opciones para realizar la clasificación de las llaves. En primer lugar se puede utilizar **SVM** a partir de las características mencionadas en la sección anterior. En segundo lugar puede utilizarse una red neuronal convolucional. Esta se entrena utilizando las imágenes recortadas y umbralizadas aplicando rotaciones y volteados para incrementar el *dataset* disponible, tal como se expuso en la sección 3.2.9 : 'CNN: Red neuronal convolucional.'

3.3.5. Resultados

Finalmente se procede a mostrar los resultados finales obtenidos para las clasificaciones con **SVM** y la **CNN**. En ambos se ha utilizado el *dataset* de la tabla 3.10. En las figuras 3.47a y 3.47b se muestran las matrices de confusión normalizadas para **SVM** y la **CNN** respectivamente. En el caso de **SVM**, se puede apreciar una mejora respecto a los resultados mostrados en la figura 3.28 y la tabla 3.8 debido al incremento en la cantidad de datos. A su vez, a diferencia de los resultados mostrados anteriormente, estos se han obtenido utilizando la técnica de validación cruzada dejando uno fuera (*Leave-one-out Cross-validation*). Respecto a la matriz de confusión de la red neuronal, los resultados son los mismos que los mostrados anteriormente en la figura 3.45, con la única diferencia de que se muestra la matriz normalizada para facilitar la comparación con **SVM**. Puede observarse como **SVM** muestra valores en una mayor cantidad de posiciones que la red neuronal, sin embargo, el error total acumulado es menor en este caso que para la **CNN**. A su vez, destacar que la carga computacional asociada a la red neuronal es significativamente mayor, por lo que la predicción a tiempo real ofrecerá menor velocidad de clasificación.

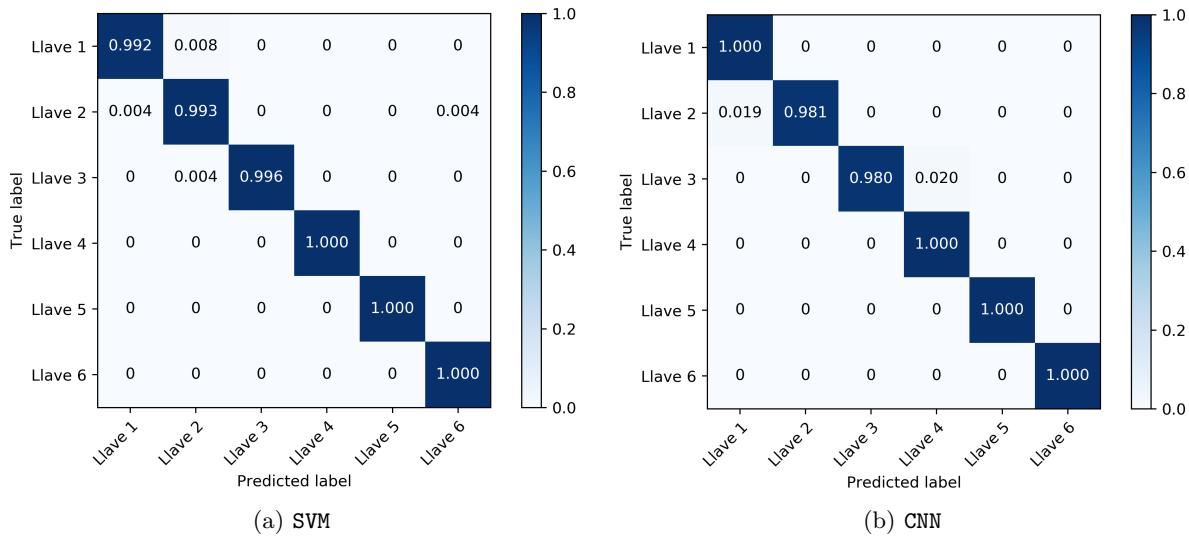


Figura 3.47: Matriz de confusión final de cada clasificador

En la tabla 3.12 se muestran los resultados de clasificación de ambos sistemas. Hay que destacar que en ambos casos se pueden observar unas sensibilidades del 100 % para las llaves 4, 5 y 6. Al igual que se ha mencionado anteriormente, se observa como **SVM** presenta una mayor cantidad de llaves con errores en la clasificación, pero sin embargo, estos errores son de menor frecuencia. Este hecho se refleja directamente en los resultados medios de ambos clasificadores que, a pesar de ser muy similares, son ligeramente superiores para la clasificación con **SVM**.

Llave	SVM			CNN		
	Precisión	Sensibilidad	f1-score	Precisión	Sensibilidad	f1-score
1	0.9959	0.9918	0.9938	0.9800	1.0000	0.9899
2	0.9889	0.9926	0.9907	1.0000	0.9815	0.9907
3	1.0000	0.9961	0.9981	1.0000	0.9804	0.9901
4	1.0000	1.0000	1.0000	0.9792	1.0000	0.9895
5	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
6	0.9961	1.0000	0.9981	1.0000	1.0000	1.0000
Media	0.9968	0.9967	0.9968	0.9932	0.9936	0.9934

Tabla 3.12: Resultados de la clasificación de cada clasificador.

4. Aspectos de mejora

La implementación final del sistema de visión artificial, si bien presenta resultados satisfactorios a la hora de realizar la clasificación, es susceptible a ser mejorada.

Por un lado, el problema de diferenciación de objetos muy próximos a la hora de realizar la segmentación no ha podido ser resuelto empleando el algoritmo de *watershed*. La sobresegmentación presentada sobre la llave de tipo 6 ha impedido resolver algunos escenarios satisfactoriamente. Las operaciones morfológicas realizadas sobre la transformación en distancia de los planos, ha impedido obtener un único marcador por llave. Cabe la posibilidad de que reorientando el planteamiento, sea posible obtener un único marcador a partir de los agujeros en las cabezas de las llaves. Se han realizado algunas pruebas pero por motivos de tiempo no se ha podido profundizar lo suficiente. Podría ser objeto de interés trabajar en esta línea para poder discernir objetos muy próximos.

Por otro lado, la red convolucional del sistema final ha sido entrenada empleando exclusivamente las imágenes binarizadas una vez las llaves presentes en estas han sido detectadas y recortadas sobre el plano de captura. Podría resultar de interés, terminar de verificar los beneficios de la red neuronal empleando imágenes de las llaves sobre distintos fondos, utilizando imágenes a color para evitar tener que realizar la umbralización de la imagen, pudiendo así asegurar una independencia de la clasificación a las condiciones del plano de captura.

A su vez, aprovechando el hecho de que se han implementado dos clasificadores independientes, podría utilizarse la salida de ambos de forma coordinada para incrementar la fiabilidad de las predicciones realizadas. Cabría por tanto añadir también un tipo de llave "desconocida" para cuando los clasificadores no lleguen a una misma solución, así como entrenar ambos con distintas llaves no consideradas permitiendo esta clase ser una de las posibles salidas de los propios clasificadores.

Finalmente, la aplicación final es susceptible a fallos y errores dependientes del usuario. Sería conveniente realizar las comprobaciones oportunas para evitar este tipo de errores durante la ejecución de la aplicación, evitando así su cierre inesperado. A su vez, hay varios aspectos de eficiencia computacional que requerirían una revisión para permitir a la aplicación ejecutarse más rápidamente. Estos aspectos podrían evaluarse y corregirse permitiendo así mejorar la velocidad de clasificación del sistema.

Referencias

- [1] Qt, <https://www.qt.io/>
- [2] OpenCV, <https://opencv.org/>
- [3] MATLAB, <https://www.mathworks.com/products/matlab.html>
- [4] Scikit-learn, <https://scikit-learn.org/stable/>
- [5] Keras, <https://keras.io/>
- [6] Tensorflow, <https://www.tensorflow.org/>
- [7] OpenCV, *Deep Neural Network Module*, https://docs.opencv.org/3.4/d6/d0f/group__dnn.html
- [8] Stack Overflow, Export h5 to pb,
<https://stackoverflow.com/questions/45466020/how-to-export-keras-h5-to-tensorflow-pb>

Índice de figuras

1.1.	Tipos de llaves disponibles para la realización del clasificador automático.	2
1.2.	Notación seguida para la identificación de las diferentes partes de una llave.	3
3.1.	Flujo de diseño del sistema.	6
3.2.	Representación de imagen capturada empleando el modelo <i>RGB</i>	8
3.3.	Representación de imagen capturada empleando el modelo <i>HSV</i>	8
3.4.	Representación de imagen capturada empleando el modelo <i>CMYK</i>	9
3.5.	Proceso de aplicación del gradiente y realce de la imagen original	10
3.6.	Resultado de la binarización generada por el método <i>OTSU</i>	11
3.7.	Cierre de la binarización proporcionada por el método <i>OTSU</i>	11
3.8.	Representación del resultado de detección de regiones conexas	12
3.9.	(a) Imagen original. (b) Resultado del giro sin recrecimiento.	12
3.10.	Recrcimiento de la imagen original.	13
3.11.	Ángulo definido por el rectángulo contenedor de una región conexa.	13
3.12.	Resultado del proceso de segmentación	14
3.13.	Resultado de la normalización del centroide	15
3.14.	Matriz de confusión de la clasificación con <i>KMeans</i>	16
3.15.	Unión de regiones conexas debida a la proximidad.	17
3.16.	Homogeneizació n local empleando filtro de media y transformación en potencias.	18
3.17.	Binarización empleando el método <i>OTSU</i> a la salida del proceso de homogeneización.	18
3.18.	Detección del fondo de las llaves.	18
3.19.	Detección del primer plano de las llaves.	19
3.20.	Detección de bordes realizada por <i>watershed</i> y transformación en distancia.	19
3.21.	Problemas asociados a la detección de bordes realizada por <i>watershed</i> y transformación en distancia.	20
3.22.	Histograma de las características utilizadas.	21
3.23.	Histograma de las características estandarizadas.	22
3.24.	Matriz de confusión de la clasificación con <i>KMeans</i> utilizando estandarizado.	22
3.25.	Matriz de confusión de la clasificación con <i>SVM</i>	24
3.26.	Ejemplo de caracterización del cifrado de una llave.	26
3.27.	Índices de separabilidad en función del número de columnas utilizadas.	26
3.28.	Matriz de confusión utilizando 20 columnas para la caracterización del cifrado.	27
3.29.	Problemática del llenado de huecos aportado por <i>drawcontours</i>	28
3.30.	Implementación del relleno de huecos.	29
3.31.	Índices de separabilidad en función del número de columnas empleando el nuevo relleno de huecos.	30
3.32.	Diferencia entre los índices de separabilidad en función del número de columnas empleado el nuevo relleno de huecos y el aportado por <i>drawcontours</i>	30
3.33.	Problemática de las llaves interferentes.	31
3.34.	Resultado final del filtrado de llaves interferentes.	31
3.35.	Detección favorecida por el ajuste de las condiciones de contorno.	32
3.36.	Detección favorecida por el ajuste de las condiciones de contorno.	33
3.37.	Imagen con varios fondos.	33
3.38.	Binarización de imagen con varios fondos.	34
3.39.	Resultado de la limpieza de bordes aplicada sobre la imagen binarizada (Ver 3.38).	34
3.40.	Detección de llaves en entorno complejo con presencia de manos.	35
3.41.	Compromiso con la posición de las llaves en el plano de captura.	35
3.42.	Arquitectura de la <i>CNN</i>	36
3.43.	Ejemplo de obtención de 4 imágenes a partir de 1.	36
3.44.	Proceso de entrenamiento de la <i>CNN</i>	37
3.45.	Matriz de confusión de la <i>CNN</i> con imágenes umbralizadas.	38

3.46. Diagrama de bloques del proceso de segmentación.	39
3.47. Matriz de confusión final de cada clasificador	40

Índice de tablas

3.1. Composición del <i>dataset</i> capturado.	7
3.2. Ejemplo características extraídas	15
3.3. Resultados de la clasificación con KMeans.	16
3.4. Resultados de la clasificación con KMeans utilizando estandarizado.	23
3.5. Diferencia entre las tablas 3.3 y 3.4 para las llaves 4, 5 y 6.	23
3.6. Resultados de la clasificación con SVM.	24
3.7. Diferencia entre las tablas 3.4 y 3.6.	25
3.8. Resultados de la clasificación utilizando 20 columnas para la caracterización del cifrado.	27
3.9. Diferencia entre las tablas 3.8 y 3.6.	28
3.10. Composición del nuevo <i>dataset</i> ampliado.	36
3.11. Resultados de la clasificación con imágenes umbralizadas y en RGB.	38
3.12. Resultados de la clasificación de cada clasificador.	41