UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA
UNIVERSIDAD COMPLUTENSE DE MADRID

MÁSTER EN INGENIERÍA DE SISTEMAS Y CONTROL

Proyecto de Fin de Máster

TÍTULO

APLICACIÓN PARA DETECCIÓN DE MATRÍCULAS DE VEHÍCULOS Y RECONOCIMIENTO DE LOS CARACTERES DE LA MATRÍCULA EN IMÁGENES EN MOVIMIENTO CON OPENCV

BERTA GARRALETA ARRESE

Dirigido por: GONZALO PAJARES MARTINSANZ

Curso: 21/22
Convocatoria: Junio/22

UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA
UNIVERSIDAD COMPLUTENSE DE MADRID

MASTER EN INGENIERIA DE SISTEMAS Y CONTROL

Proyecto de Fin de Master

APLICACIÓN PARA DETECCIÓN DE MATRÍCULAS DE VEHICULOS Y RECONOCIMIENTO DE LOS CARACTERES DE LA MATRÍCULA EN IMÁGENES EN MOVIMIENTO CON OPENCV

.....

PROYECTO DE FIN DE MASTER
DE MODALIDAD ESPECÍFICA

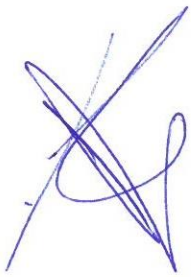
BERTA GARRALETA ARRESE

Dirigido por: GONZALO PAJARES MARTINSANZ

Autorización

Autorizamos a la Universidad Complutense y a la UNED a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firmado: BERTA GARRALETA ARRESE



Firma del alumno

RESUMEN

En el presente proyecto se desarrolla una aplicación para detectar y segmentar la placa de un vehículo a partir de una imagen extraída de un video obtenido mediante la cámara de un dispositivo móvil, por ejemplo, un Smartphone.

Para ello, se utilizan diversas técnicas de procesamiento digital de imágenes explicadas en la referencia [1] que permiten mejorar la calidad de la imagen, y facilitar la extracción de contornos válidos susceptibles de ser una placa de vehículos basándonos en las características propias de una matrícula [2]. De esta forma se realiza una propuesta de regiones con alta potencialidad de contener la matrícula. Tras este proceso se procede a aplicar una segmentación de más alto nivel para segmentar la placa en cada uno de sus dígitos y obtener un texto que pueda ser integrado en otras aplicaciones.

Para la interpretación de cada segmento se utilizan técnicas de aprendizaje, como es el algoritmo K-NN y las redes neuronales basadas en características Haar y en los momentos Hu.

Todo ello en conjunto e integrado, genera un sistema de reconocimiento automático y válido que permite reconocer la matrícula de un vehículo partiendo de una imagen en movimiento.

La aplicación Android [3] se desarrolla en JAVA [4], implementándose mediante Android Studio [5] y Apache Netbeans [6] respectivamente, y utiliza la librería OpenCV [7] como herramienta principal para el análisis y tratamiento de imágenes. Esta librería proporciona diversas funciones, necesarias para el procesamiento de las imágenes utilizadas en este trabajo, además de incluir funciones específicas para el aprendizaje automático.

PALABRAS CLAVE

ANDROID: sistema operativo para dispositivos móviles.

ANDROID STUDIO: entorno de desarrollo para dispositivos con sistema operativo Android.

OPENCV: librería open-source para tratamiento de imágenes.

JAVA: lenguaje de programación.

Sistema ANPR: reconocimiento automático del número de matrícula.

REDES NEUONALES: modelo computacional simplificado que emula el modo en que el cerebro humano procesa la información.

HAAR: Redes neuronales basadas en características Haar.

APRENDIZAJE AUTOMÁTICO: Rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas que permitan que las computadoras aprendan.

TENSORFLOW-KERAS: Frameworks de alto nivel para experimentación rápida, programado en Phyton.

PHYTON: lenguaje de programación de alto nivel.

TITLE

SOFTWARE FOR DETECTING VEHICLE REGISTRATIONS AND READING THE LICENSE PLATE IN VIDEO

ABSTRACT

This project develops an application for detecting and segmenting a vehicle's number plate from an image extracted from a video, obtained from the camera of a mobile device, for example, a smartphone.

To do this, different digital image processing techniques, explained in reference [1] are used to improve the quality of the image, and facilitate the extraction of valid contours likely to be a vehicle license plate based on their characteristics [2], and later to be able to segment the plate in each of its digits and obtain a text that can be integrated into other applications.

In this way, a proposal of regions with a high potential to contain the number plate. After this process, a higher-level segmentation is applied to extract the number of the plate through the digits, obtaining a text that can be integrated in other applications.

For the interpretation of each segment (character), learning techniques are applied, such as the K-NN algorithm and neural networks based on Haar characteristics and on Hu moments.

All this together, and conveniently integrated generates an automatic and valid recognition system that allows to recognize the license plate of a vehicle starting from a moving image.

The Android [3] and JAVA [4] application is implemented in Android Studio [5] and Apache Netbeans [6] respectively and uses the OpenCV library [7] as the main tool for image analysis and treatment. This library implements many functions necessary for image processing, and includes specific functions for machine learning

.

Índice

Índice de figuras

Índice de tablas

Índice de máscaras

Índice de formulas

PRELIMINARES

La segmentación de imágenes para extraer información relevante, identificando los elementos básicos incluidos en la imagen constituye un objetivo clave en visión por computador, donde el reconocimiento de objetos es un aspecto esencial, que desemboca en lo que se conoce como segmentación.

La detección de los caracteres que componen una matrícula es el paso decisivo para la identificación del vehículo correspondiente a través de esta. En este proceso la localización y ubicación de la matrícula resulta esencial para el reconocimiento de los caracteres que la componen.

En base a lo anterior, en este trabajo se proponen dos módulos de procesamiento bien separados secuencialmente, a saber: detección de la propia placa y reconocimiento de sus caracteres. El primero es responsable de la localización de la región en la imagen donde se ubica la placa de la matrícula, de forma que una vez detectada, se procede al recorte de la región que la contiene. Esta sub-imagen convenientemente recortada constituye la entrada al módulo de reconocimiento de caracteres propiamente dicho, cuya misión consiste en aislar los caracteres para su reconocimiento, teniendo el contexto de la matrícula, según el país de utilización de la misma.

Ambos módulos se integran de forma secuencial, de forma que el primero realiza un proceso, cuyo resultado sirve de base al segundo. Esto significa que los resultados del primero son decisivos para un buen rendimiento del segundo.

Más específicamente, el proceso de detección de la placa aplica métodos basados en la detección de contornos que definen la placa y considera aspectos derivados de su proyección vertical para la localización indicada previamente. La idea subyacente es aprovechar la geometría rectangular o cuadrada de la propia placa, que la caracteriza, según el tipo de vehículo.

En el capítulo 1 se plantea y describe el problema a resolver, junto con los requisitos y planteamientos necesarios para tal fin.

En el capítulo 2 se describe la captura de la imagen procedente del video y el algoritmo de detección de movimiento.

En el capítulo 3 se describen los procesos preliminares para el tratamiento de la imagen antes de abordar procesos más complejos. Se estudian los conceptos necesarios para tal fin.

En el capítulo 4 se describen los métodos para la detección de la región donde se ubica la placa de la matrícula.

En el capítulo 5 se describe la implementación relativa a la segmentación una vez localizada la región de la imagen que la define, junto con los fundamentos teóricos necesarios para tal fin.

En el capítulo 6 se evalúan los resultados obtenidos, así como las conclusiones pertinentes y sus posibles ampliaciones.

1. INTRODUCCIÓN Y PLANTEAMIENTO DEL PROYECTO

Hoy en día, con el auge y desarrollo futuro de las ciudades inteligentes las técnicas inteligentes de reconocimiento de matrículas basadas en visión por computador están llamadas a jugar un papel esencial en el control de vehículos por diversas razones, entre las que destacan las de seguridad como acceso a determinados espacios públicos o privados, así como el control del flujo de tráfico por la identificación del tipo de vehículos que circulan, cuya finalidad última es una mejor gestión de cara a la sostenibilidad. Una de las mejores formas de lograr estos objetivos es la identificación de matrículas, que identifican de forma unívoca cada tipo de vehículo.

El problema a resolver, que constituye la hipótesis de partida, consiste en obtener la región de una placa de matrícula, a partir de la cual se extraen los caracteres que la componen y que en conjunto definen unívocamente el vehículo correspondiente.

Se utilizan distintas técnicas para obtención del valor correspondiente a dicho digito segmentado [1]. Técnicas de clasificación como el algoritmo KNN, técnicas de OCR (*Optical Character Recognition*), técnicas de aprendizaje como redes neuronales en base a las características Haar de una imagen o en base a los momentos HU, en una matrícula española en base a unos atributos diferenciadores. Justificación y situación actual.

El reconocimiento de placas de matrículas puede utilizase en diferentes aplicaciones, destacando las siguientes como ejemplos relevantes:

- Control de acceso a parkings o peajes.
- Control del flujo de tráfico en circulación.
- Utilización como mecanismo de seguridad.

Existen diversas soluciones comerciales en el mercado que realizan este tipo de reconocimiento normalmente en cámaras estáticas de gran calidad. [8]. A su vez existe gran cantidad de proyectos, tesis, artículos relacionados con el tema. Cabe citar a este respecto la tesis en [9].

La diferencia con dichos proyectos estriba en la realización sobre sistema Android en tiempo real, bajo cualquier tipo de móvil lo que hace que el proyecto sea fácil de utilizar y con coste de desarrollo y computacional casi nulo.

También es importante el grado de comunicación de la aplicación con el usuario que permite interactuar.

1.1. Objetivos del proyecto

El objetivo general y los objetivos específicos de este proyecto son los que se describen a continuación.

1.1.1. Objetivo general

Como objetivo general se plantea desarrollar un sistema que permita detectar y segmentar la placa de matrícula de vehículos a partir de una imagen sacada de un video obtenido de la cámara de un dispositivo móvil, por ejemplo, un Smartphone.

1.1.2. Objetivos específicos

Los objetivos específicos del proyecto son los siguientes:

- Leer la imagen previamente obtenida por medio de la cámara integrada en un dispositivo móvil.
- Procesar la imagen para extraer contornos válidos para ser susceptibles de ser una placa de vehículo.
- Procesar y analizar los contornos interiores de la placa para validar o no la placa.

- Segmentar la placa válida en cada uno de sus dígitos.
- Recopilar las propiedades características de estos dígitos.
- Interpretación de cada segmento mediante el algoritmo KNN, o redes neuronales.
- Guardado de la matrícula en una base de datos con las características de fecha, hora y el fotograma completo donde está recogida dicha matrícula.
- Realizar evaluación y comparación de los resultados obtenidos.

1.2. Planificación y estructura del proyecto

La planificación sigue una estructura típica de Especificación de requisitos, Descripción general del proyecto, Análisis de requisitos, Modelado de casos de uso, Análisis de Visión artificial, Diseño del proyecto, Codificación e implementación, Pruebas, Conclusiones y Documentación.

Dentro del Análisis de Visión artificial se sigue el esquema general mostrado en la figura 1.

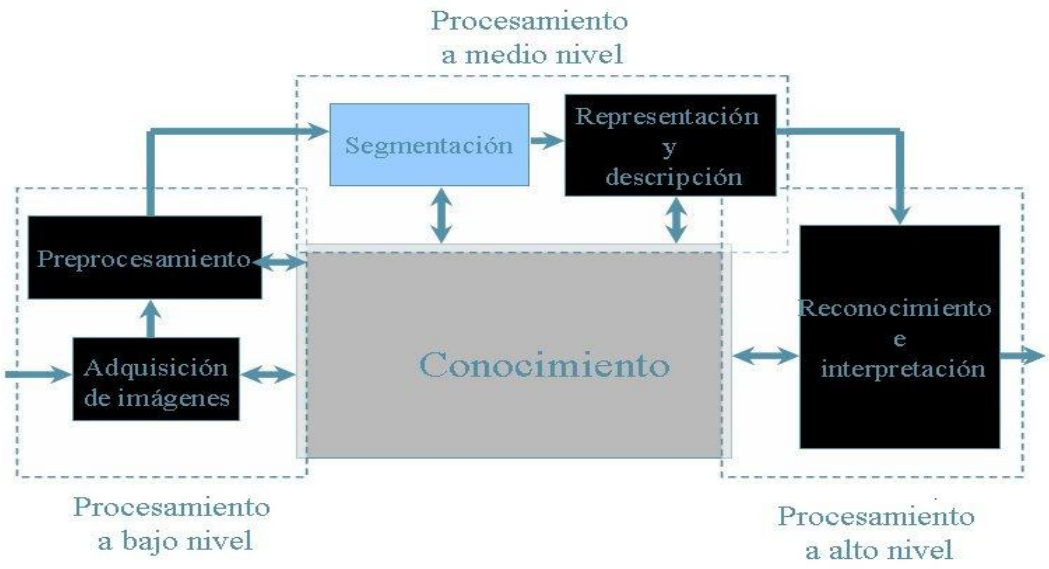


Figura 1: Esquema general de visión artificial

1.3. Ámbito

La aplicación consistirá principalmente en una aplicación móvil para dispositivos con sistema operativo Android y mediante la cual se realizarán todas y cada una de las distintas acciones posibles asociadas a la detección y reconocimiento de caracteres de una matrícula española de vehículos.

1.4. Perfil de usuario

Solo existe un perfil de usuario que no necesita registrarse para su uso. Como posible ampliación cabe mencionar la creación de un usuario registrado administrador que pueda cambiar la configuración del registro de resultados.

1.5. Referencias clave

A continuación, se enumeran las referencias base, que constituyen la base y apoyo fundamental del proyecto.

- Para la especificación de requisitos se utiliza la IEEE Std. 830-1998: ESPECIFICACIÓN DE LOS REQUISITOS DEL SOFTWARE. [10]
- Para el Análisis de métodos de procesamiento de imágenes y segmentación se utilizan:
- **Cruz, Gonzalo Pajares y Jesús M. de la.** Visión por computador: imágenes digitales y aplicaciones; 2ª edición. s.l. : RA-MA, 2007.
- **Chen, Jun Wei Hsieh Shih Hao Yu and Yung-Sheng.** Morphology-based-License Plate Detection from Complex Scenes Proceedings of the Complex Scenes. Quebec : s.n., 2002. 10.1109/ICPR.2002.1047823.

1.6. Modelo de desarrollo

El modelo elegido para el desarrollo del programa será evolutivo, en la cual enlazamos especificación, desarrollo y validación.

Como metodología, la elegida es SCRUM, que es ágil y flexible, proporciona gran valor al cliente y a los principios de inspección continua, adaptación, innovación y autogestión.

1.7. Hardware y software necesario

El hardware necesario para la creación del proyecto ha sido un ordenador con Windows mínimo 7, 4GB de RAM, 128 Gb de Disco Duro. En cuanto al software, por la lentitud de compilación del Android Studio, las pruebas se han realizado programando sobre Java 8 con Netbeans 8.2 y una vez realizado cada función se ha traspasado a Android Studio Chipmunk 2021. Se ha utilizado la librería OPENCV versión 4.5.2.

La librería OpenCV (Open Computer Vision Library) fue creada en el año 2000 por Intel® Corporation. Es una librería Open Source para la visión artificial o visión por computadora escrita en C y C++. Corre bajo Linux, Windows y Mac OS X. También se puede utilizar con lenguajes como Python, Ruby, Matlab, Java entre otros. Proporciona varios paquetes de alto nivel para el desarrollo de aplicaciones de visión.

1.8. Estudio de costes y viabilidad

Los costes de equipo son básicos dado que se puede realizar en cualquier ordenador con sistema operativo Windows y que el software utilizado es gratuito. El único coste relevante es la mano de obra de la autora de la aplicación.

2. CAPTURA DE IMAGEN Y DETECCIÓN DE MOVIMIENTO

2.1. Características de las cámaras de un dispositivo móvil.

Los móviles que hoy día salen al mercado pueden tener dos, tres, cuatro o hasta cinco cámaras. La cámara principal estándar solo tiene una distancia focal fija y por lo tanto un único ángulo de visión por eso las cámaras adicionales podrán ampliar el ángulo de visión (gran angular) o reducir el ángulo de visión y acercar el zoom a los sujetos (teleobjetivo).

La cámara principal de un teléfono inteligente suele tener un ángulo de visión moderadamente amplio de unos 60° y más. La apertura puede variar desde f/1.5 (apertura grande) en algunos modelos hasta f/2.8 (apertura pequeña) en teléfonos más baratos.

Los teléfonos inteligentes con teleobjetivos adicionales pueden acercarse o hacer más zoom, lo que a menudo se usa como sinónimo de tener un ángulo de visión más pequeño. Mientras que la mayoría de las cámaras de teléfonos inteligentes permiten hacer zoom al recortar la imagen y ampliar el resultado (zoom digital), los teléfonos inteligentes con lentes separadas lo logran de manera óptica. La segunda cámara tiene una distancia focal más larga o un sensor más pequeño, o a veces ambos.

En contraste con las cámaras teleobjetivo, las lentes de gran angular adicionales permiten capturar más contenido de la imagen desde una cierta perspectiva al ampliar el ángulo de visión.

Las características mínimas que debemos buscar en nuestra cámara son:

- **Resolución del sensor:** es la cantidad de píxeles del sensor. Más de 12 Mpx son innecesarios y no mejoran la calidad de la imagen.

- **Estabilizador óptico (OIS):** El estabilizador óptico de imagen que es capaz de contrarrestar los temblores de la mano. Esto es especialmente útil en entornos más oscuros.
- **Estabilización electrónica (EIS)** para la grabación de video: Al contrario de OIS, la estabilización de la imagen no se realiza de forma electromecánica, sino en el software. La desventaja es que esto reduce el ángulo de visión del modo de video en comparación con la configuración no EIS.
- **Grabar con más de 30 fps**: Otra muestra de calidad es que el móvil puede grabar con más de 30fps o imágenes por segundo.

Estas características harán que la calidad de nuestra imagen sobre la que trabajar varié mucho entre un dispositivo y otro.

2.2. Adquisición de una imagen desde vídeo

La adquisición de un fotograma desde un video es un elemento básico de cualquier librería de trabajo con imágenes.

En este caso se trabaja con el paquete *Videoio* de OpenCV, donde encontramos la clase *VideoCapture* que nos permite capturar video desde un archivo, desde la cámara web o la cámara del video. Una vez que la cámara esté activada y leyendo los fotogramas, estos se pasan a la aplicación. En Android Studio se convierte en *cameraBridgeViewBase.*

Hay que tener en cuenta que la cámara de un móvil normal, en general, suele grabar a 30 fps, (30 fotogramas por segundo). Normalmente no va a ser necesario realizar nuestro procesamiento con todos los fotogramas. Por lo que se crea una función que detecta el suficiente movimiento entre fotogramas para considerar que ha habido una modificación suficiente para realizar el procesamiento del siguiente fotograma.

2.3. Algoritmo para detectar un movimiento suficiente

Las posibles situaciones de grabación que se han contemplado en este proyecto incluyen grabación estática y grabación en movimiento. Los coches también se pueden encontrar estáticos o en movimiento.

a) *Analizando el movimiento*

El peor caso sería aquel en el que el usuario se encontrase estático y los coches en movimiento. Analizando esta situación, considerando una velocidad máxima de 120 km/hora, transformando esta velocidad corresponde a 33,333 m/s que pasándolo a fotogramas son 1,11 m/fotogramas. En esta situación se determina que se debe analizar cada fotograma.

Por el contrario, en el mejor caso el usuario se encuentra caminando a una velocidad media de 5 km/hora y los coches estáticos. Transformando las unidades obtenemos 0,05 m/fotograma. En esta situación no tiene sentido analizar cada fotograma.

Según este análisis debemos buscar un procedimiento para determinar cada cuántos fotogramas se debe analizar la imagen. Para ello existen gran cantidad de técnicas de detección de movimientos cuyo marco teórico está desarrollado en los capítulos 12 y 13 del libro de referencia [1].

b) Analizando dichos métodos me decanto por la técnica de substracción de imágenes.

Esta técnica consiste en capturar una secuencia de imágenes. En este caso, dos imágenes. Una imagen corresponde al último fotograma analizado y la otra imagen corresponde al fotograma actual leído de la cámara. Sobre ellas se aplican unas pequeñas transformaciones como es, conversión a escala de grises, reducción de la resolución de las imágenes y comparación de imágenes mediante diferencias de píxeles.

Cuando la imagen resultante de la diferencia sea lo suficientemente significativa se envía el fotograma actual al proceso de análisis y se actualiza la última imagen analizada. Se ha considerado una diferencia superior al 2%.

En función de estas diferencias se puede determinar el tipo de movimiento que realiza el usuario.

En el anexo 1 se especifican las distintas funcionalidades del código utilizado.

3. PRE-PROCESAMIENTO DE LA IMAGEN

En esta sección se describen los distintos preprocesamientos realizados a la imagen antes de la búsqueda de contornos y regiones de interés. En el anexo 2 se proporcionan detalles sobre algunos conceptos elementales para el tratamiento de imágenes.

Una vez que la imagen está digitalizada es posible mejorar la apariencia visual de la misma, o transformarla de tal forma que facilite su análisis computacional para una aplicación específica. Este es el objetivo de las técnicas de realce trabajadas en este capítulo. Es importante destacar el término específico, ya que el resultado de estos métodos depende en gran medida de la imagen que se quiere mejorar. Este procesamiento siempre será mejor contra más información se tenga de las imágenes. Puede ser de utilidad en algún caso interactuar con el usuario para pedir dicha información, aunque siempre el programa debe trabajar de modo autónomo.

El marco teórico para este estudio se desarrolla ampliamente en el capítulo 4 del libro de González y Woods [11]. También se utiliza los capítulos 2 al 6 del libro de Pajares y Cruz. [1]

Los métodos de realzado de imágenes pueden realizarse tanto en el dominio espacial como en el de la frecuencia. Los del primer grupo se caracterizan por operar directamente sobre los píxeles de la imagen, mientras que los del segundo grupo modifican una transformada de esta, (Fourier, DCT, wavelet, etc.) para obtener los resultados.

El procesamiento sobre el dominio espacial es una función tal que dada una imagen de entrada produce una imagen de salida. Estas operaciones pueden ser vistas como:

- **Operaciones puntuales de transformación:** Son transformaciones de la imagen que no involucran vecindarios. Ejemplos de estos operadores incluyen

ajustes de brillo y contraste, así como corrección de color y transformaciones geométricas.

- **Operaciones basadas en los píxeles vecinos**: operaciones con ayuda de una máscara o *kernel*, las operaciones son efectuadas sobre un píxel tomando en cuenta sus píxeles vecinos. Son útiles cuando se busca reducir ruido, detectar bordes o generar algún tipo de efecto artístico.

3.1. Transformación de Color

Para el procesamiento de imágenes debemos recurrir a los llamados espacios de color, que son representaciones basadas en conjuntos de fórmulas matemáticas que permiten describir los colores y descomponerlos en distintos canales. Los espacios más utilizados son el RGB (Red, Green, Blue), el CMYK (impresoras), también el sistema tridimensional HSI (tono, saturación e intensidad), el sistema XYZ (como referencia para definir los colores que percibe el ojo humano) y el más completo el LAB (identifica cada color mediante sus valores "*a*" y "*b*" y su brillo *L*.)

3.1.1. Proceso aplicado

En este proyecto podemos necesitar la imagen a color (RGB) tomada por la cámara del dispositivo móvil tanto para guardarla como posible comparación manual o en su caso para poder realizar alguna transformación basada en el color azul existente en las matrículas.

Para la mayoría de los procesamientos utilizaremos imágenes en escala de grises, en las cuales solo tenemos un canal donde se guarda la intensidad (el blanco) de cada píxel. Esta transformación se realiza mediante la siguiente función.

$$\boldsymbol{Y = 0,299R + 0,587G + 0,114B} \qquad \textbf{(1)}$$

Donde R el valor de la componente Roja, G el valor de la componente Verde y B el valor de la componente azul.

3.2. Ajuste de iluminación

La iluminación es un factor muy importante a tener en cuenta para el análisis de la imagen.

Dos operaciones puntuales comúnmente utilizadas son *la multiplicación* y la *adición* con una constante mediante la siguiente función:

$$\boldsymbol{g(x) = \alpha f(x) + \beta} \quad \textbf{(2)}$$

Los parámetros α y β a menudo se denominan parámetros de *ganancia* y sesgo*;* a veces se dice que estos parámetros controlan el *contraste* y el *brillo* respectivamente.

Se puede pensar en f(x) como los píxeles de la imagen de origen y g(x) como los píxeles de la imagen de salida. Entonces, más convenientemente podemos escribir la expresión como: $\boxed{g(i,j) = \alpha f(i,j) + \beta}$ donde i y j indican que el píxel se encuentra en la fila *i-ésima* y la columna *j-ésima.*

Actuaremos sobre los tres valores (R,G,B) y accederemos a ellos separadamente para realizar la transformación.

También se puede utilizar la siguiente transformación no lineal entre la imagen de entrada y de salida.

$$\boldsymbol{O = \left(\frac{I}{255}\right)^{\gamma} \cdot 255} \quad \textbf{(3)}$$

Al no ser una transformación lineal, el efecto no será el mismo para todos los píxeles y depende de la imagen de entrada.

Como se puede ver en la figura 2, Cuando γ<1, las regiones oscuras originales serán más brillantes y el histograma se desplazará a la derecha, mientras que será al contrario con γ>1.

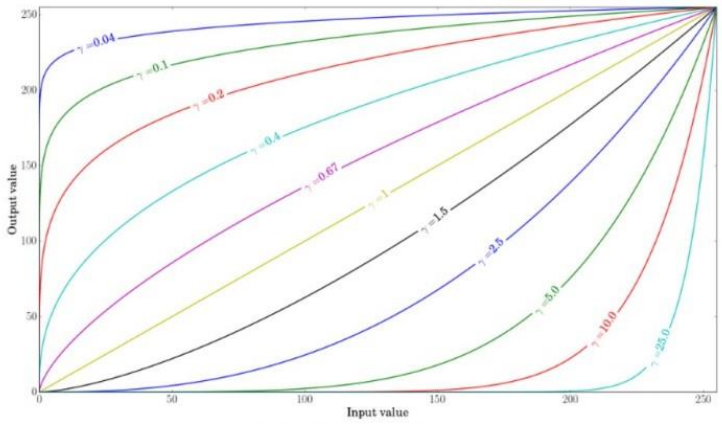


Figura 2: Trama para diferentes valores de gamma

Se utiliza esta transformación en la opción manual permitiendo que el usuario pueda modificar el valor gamma mediante una barra de desplazamiento y que vea los resultados en pantalla.

En cambio, para la transformación automática se realiza un estudio del histograma de la imagen.

El histograma es una función discreta que representa el número de píxeles en la imagen en función de los niveles de intensidad *g* o lo que es lo mismo la frecuencia con la que se repiten determinados valores.

Corresponde a la siguiente función:

$$\boldsymbol{P(g) = \frac{N(g)}{M}} \quad \textbf{(4)}$$

Donde la probabilidad de ocurrencia es *P(g)*, *M* es el número de píxel en la imagen y *N(g)* es el número de píxeles en el nivel de intensidad *g*.

El histograma tiene unas propiedades estadísticas como son la media, varianza, asimetría y entropía que nos dan información sobre el brillo y contraste de la imagen.

Un histograma con una distribución concentrada en una zona presenta un contraste muy bajo, mientras que con una amplia distribución de los niveles de grises tiene un alto contraste. Si están concentrados en la parte baja del rango es una imagen oscura y si están concentrados en la parte alta es una imagen brillante.

Podemos ver estas características en una de las imágenes, figura 3, que es muy oscura.

Figura 3: Histograma

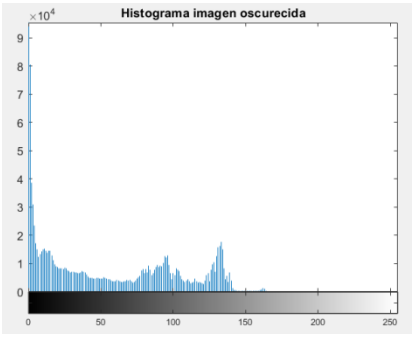


Figura 4: Imagen original oscura

Se observa en la figura 4, cómo el histograma está desplazado hacia el lado izquierdo. Calculando su media se obtiene un valor de 4,4, que corresponde a unos valores muy oscuros. Hay que tener en cuenta que la media debería estar alrededor de 128 (la mitad de los niveles de intensidad normales en un rango de 0 a 255).

Basándonos en esta información podemos calcular el parámetro gamma necesario para mejorar la imagen, despejando de la fórmula $O = \left(\frac{I}{255}\right)^{\gamma} \cdot 255$ el valor γ e introduciendo la media como valor de entrada (I) y poniendo 128 como valor de salida adecuado (*O*), figura 5.

Figura 5: Imagen después de aplicar el γ calculado basado en la media de la imagen

También se puede utilizar la ecualización del histograma de una imagen que es una transformación que pretende obtener para una imagen un histograma con una distribución uniforme. Es decir, que exista el mismo número de píxeles para cada nivel de gris del histograma de una imagen monocroma, figura 6.

Figura 6: Imagen obtenida después de aplicar una ecualización del histograma de la imagen

Muchas veces esta transformación no es adecuada debido a que el histograma puede no estar confinado a una región en particular y puede haber partes de la imagen que queden quemadas. Para resolver este problema, se utiliza la ecualización de histograma adaptativo. En este caso, la imagen se divide en pequeños bloques. Luego, cada uno de estos bloques se ecualiza considerándolo una imagen completa.

Entonces, en un área pequeña, el histograma se limitaría a una región pequeña (a menos que haya ruido). Si hay ruido, este se amplificará. Para evitar esto, se aplica la limitación de contraste. Si cualquier BIN del histograma está por encima del límite de contraste especificado (por defecto 40 en OpenCV), esos píxeles se recortan y se distribuyen uniformemente a otros BINS antes de aplicar la ecualización de histograma.

Después de la ecualización, para eliminar artefactos en los bordes de los mosaicos por la división de la imagen en los mencionados bloques, se aplica la interpolación bilineal, figura 7.

Figura 7: Imagen obtenida con ecualización adaptativa del histograma con tamaño de bloque 8 y con umbral de 40

3.2.1. Proceso aplicado

Después de analizar los resultados obtenidos con los distintos métodos en 50 imágenes de referencia, la mejor opción es utilizar una ecualización del histograma para el trabajo con la imagen completa cuando inicialmente se está buscando la placa. Y en el proceso sobre la placa para identificar correctamente los caracteres se utiliza una ecualización adaptativa con tamaño de bloque 8 y umbral 40.

Los resultados con el cálculo de gamma no han resultado adecuados en imágenes donde hay gran cantidad de focos o en imágenes donde la luz de los focos satura la

imagen. Habría que utilizar un cálculo de gamma subdividiendo en secciones la imagen. Circunstancia ésta, que se incluye como propuesta de mejora.

3.3. Eliminación de ruidos

En toda imagen digital no ideal es posible encontrar ruidos. El ruido es el resultado de errores en el proceso de adquisición de imágenes que dan como resultado valores de píxeles que no reflejan la intensidad real de la escena. Este ruido, en nuestro caso, vendrá de la adquisición de la imagen y de los procesos anteriores realizados.

Para eliminar el ruido se puede utilizar determinados filtros lineales como el promedio, filtros gaussianos y otros no lineales como la mediana. Estas operaciones son operadores de vecindad, o lo que es lo mismo, el valor resultante de un píxel se calcula como una función de su vecindad. Son invariantes al desplazamiento lo que permite que el comportamiento sea independiente de la posición del píxel. Su linealidad e invariancia permiten que se representen con una convolución.

Un filtro lineal se representa como:

$$\boldsymbol{g(i,j) = \sum_{k,l} f(i+k, j+l) h(k,l)} \quad (5)$$

Donde *h* se denomina núcleo o máscara del filtro y los valores *k*, *l* representan la vecindad del píxel (*i*,*j*).

A nivel práctico representamos la operación convolución como *g=f*h*, que se caracteriza por reemplazar cada píxel con la media de los píxeles vecinos multiplicado por unos pesos que se denominan *kernel* o máscara.

La librería utilizada en este proyecto dispone de varios tipos de filtros:

Filtro promediado: Se obtiene una imagen suavizada cuya intensidad resulta ser el promedio de los valores de intensidad de los píxeles incluidos en el entorno de

vecindad predefinido. Suaviza la imagen, eliminado las diferencias. Un efecto colateral es que desdibuja contornos. Puede ser reducido utilizando el filtro de la mediana.

$$f(i,j) = \frac{1}{P}\sum_{m,n\,\epsilon\,S} g(m,n) \quad \textbf{(6)}$$

Donde P es el número de puntos de vecindad y S, el conjunto de coordenadas de los puntos de la vecindad.

Mediante la función *Blur* y la función *boxFilter* del OpenCV podemos realizar este tipo de filtro y configurar el tamaño de los vecinos y mover el ancho con respecto al centro del *kernel*. La diferencia entre estas dos funciones solo es que en *boxFilter* no se normaliza el filtro, figura 8 y 9.

Figura 8: Imagen resultado de aplicación de la función *boxfilter* con tamaño de matriz de 3x3

Figura 9: Imagen resultado de aplicación de la función *blur* con tamaño de matriz de 3x3

Filtro de mediana (*medianBlur*): toma la mediana de todos los píxeles debajo del área del *kernel* y el elemento central se reemplaza con este valor mediano. Esto es muy eficaz contra el ruido de sal y pimienta en una imagen. Reduce el ruido de forma eficaz. El tamaño de su núcleo debe ser un entero impar positivo. Es no lineal, figura 10.

Figura 10: Imagen resultante de aplicación de la función *medianblur* con tamaño 3

Filtro gaussiano (GaussianBlur): sustituimos el kernel por un kernel gaussiano que da más peso al centro y menos a los bordes. Un filtro gaussiano toma la vecindad alrededor del píxel y encuentra su promedio ponderado gaussiano Este filtro gaussiano es solo una función del espacio, es decir, todos los píxeles cercanos se consideran durante el filtrado. No considera si los píxeles tienen casi la misma intensidad. No tiene en cuenta si un píxel es un píxel de borde o no. Para mejorar esto utilizamos un filtro bilateral. Computacionalmente es más lento que otros filtros. La fórmula del filtro gaussiano es:

$$G(x,y) = \frac{1}{2\pi\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}} \qquad (7)$$

Esta función se discretiza mediante una máscara, cuyo resultado se muestra en la figura 11.

Figura 11: Imagen resultante de aplicación de la función GaussianBlur con tamaño 3 y sigma 1

.

Filtro bilateral (bilateralFilter): ES muy eficaz en eliminación de ruido manteniendo los bordes nítidos. Pero es más lento que los demás filtros. Utiliza una función gaussiana de diferencia de intensidad, que asegura que solo aquellos píxeles con intensidades similares al píxel central se consideren para desenfoque. Por lo tanto, conserva los bordes, ya que los píxeles en los bordes tendrán una gran variación de intensidad, figura 12.

Figura 12: Imagen resultante de aplicación de la función bilateralFilter con tamaño 3 y sigma 1

3.3.1. Proceso aplicado

Analizando los distintos filtros, no se distingue visualmente gran variación en la zona de la matrícula que es la que nos interesa. Aunque realmente una modificación de 2 o 3 píxeles pueden hacer que posteriormente no se detecte un contorno perfectamente.

Para cubrir todas las opciones se realizan dos pasadas por cada fotograma, uno realizando un filtro de mediana con tamaño 3 que en general es bastante eficiente para el ruido de pimienta para la imagen completa y en otra pasada un filtro bilateral con tamaño 3 y con sigma 5.

3.4. Enfoque y afilado

Para poder realizar una segmentación adecuada, la imagen bajo procesamiento debe estar lo más enfocada y afilada posible.

Existen varias técnicas para realizar este tipo de funciones. Se pueden utilizar filtros de paso alto que conservan las altas frecuencias y atenúan las bajas frecuencias. También mediante convolución con determinadas máscaras conseguimos que los bordes sean más marcados, porque acentúan los puntos que más se diferencian de los vecinos, aumentando por lo tanto las altas frecuencias. Aunque también aumenta el ruido de alta frecuencia. Y también se pueden utilizan otras técnicas basadas en el gradiente y el laplaciano para detectar las zonas de la imagen con mayor variedad de intensidad. Se comentan en el siguiente apartado de detección de bordes.

Las máscaras más utilizadas para ello son:

- **Kernel de realce de bordes:** Si hacemos una convolución con el siguiente *kernel* obtenemos la figura 13. Observándose una pérdida de información.

$$\begin{bmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Máscara 1: Realce de bordes

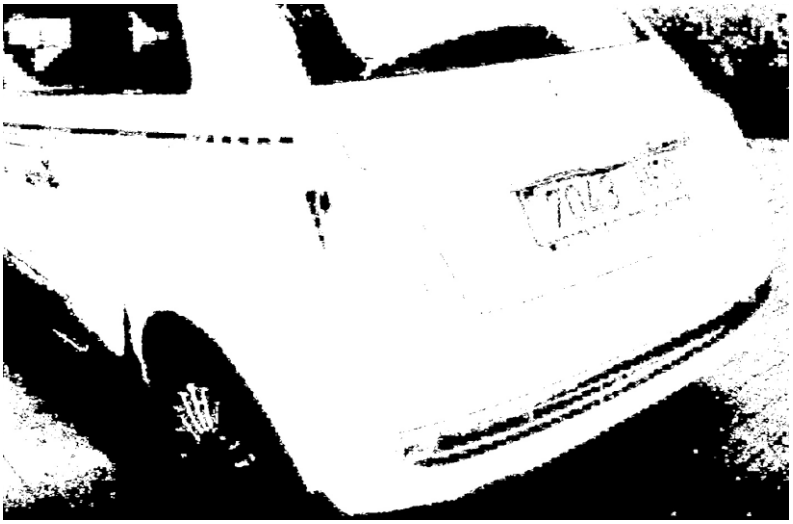


Figura 13: Imagen resultante de aplicar un filtro mediante una máscara de realce de bordes

- **Kernel de enfoque**: Si se aplica una convolución con el siguiente kernel obtenemos una imagen con los bordes más enfocados, figura 14.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Máscara 2: Enfoque

Figura 14: Imagen resultante de aplicar un filtro mediante una máscara de enfoque

- **Kernel de repujado de bordes**: Si se realiza una convolución con el siguiente *kernel* obtenemos una imagen con los bordes más repujados como se puede ver en la siguiente imagen de la figura 15. Deberíamos añadir esta imagen a la original para obtener mejores resultados.

$$\begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

Máscara 3: Repujado de bordes

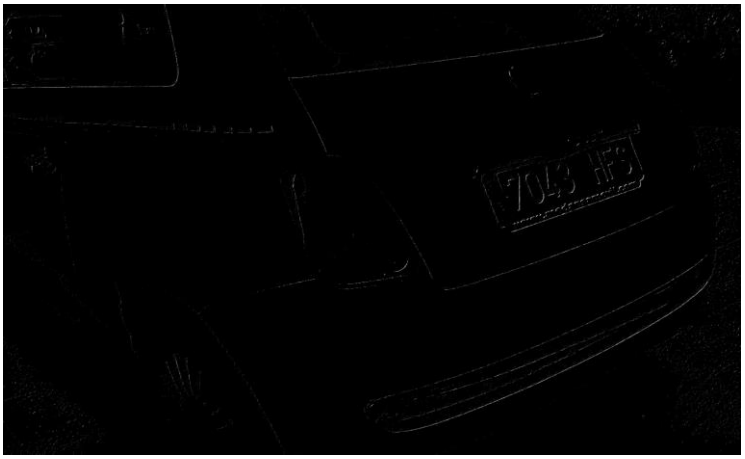


Figura 15: Imagen resultante de aplicar un filtro mediante una máscara de repujado

- **Filtro de tipo Sharpen (afilado):** Si se aplica una convolución con el siguiente *kernel* obtenemos una imagen con los bordes más afilados como se puede ver en la imagen de la figura 16.

$$\begin{bmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

Máscara 4: Sharpen

Figura 16: Imagen resultante de aplicar un filtro de tipo Sharpen

- **Filtro de máscara Sobel:** Aplicando una convolución con el siguiente kernel obtenemos una imagen con los bordes muy marcados como se puede ver en la imagen de la figura 17.

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Máscara 5: Sobel

Figura 17: Imagen resultante de aplicar una máscara Sobel

- Existen otras máscaras también utilizadas para el enfoque:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 5 & -1 \\ -1 & -1 & -1 \end{bmatrix} \qquad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & 0 \end{bmatrix} \qquad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Máscara 6: Otras máscaras de enfoque

- **Algoritmo de máscara de enfoque**: Muy utilizado en programas de diseño gráfico tipo Gimp [12]. Se basan en el concepto de "enmascaramiento difuminado" (*unsharp-masking*). Operan restando a la imagen una parte de su correspondiente paso bajo.

$$\boldsymbol{H_{unsharp-masking} = A * Original - H_{paso\ bajo} = (A-1) * Original + Original - H_{paso\ bajo}} \\ \boldsymbol{= (A-1) * Original + H_{paso\ alto}}$$

(8)

Tal y como se puede observar en la ecuación anterior, el resultado es equiparable a un filtrado paso-alto al que se le añade parte de la imagen original, lo que le devuelve, parcialmente, las componentes de baja frecuencia perdidas. Es por esto que estas técnicas también se conocen por "énfasis de las altas frecuencias"' o high-boost. El grado de mejora, por tanto, dependerá del parámetro *A*, definido en la ecuación (8).

En la mayoría de los programas de diseño gráfico se utiliza un **filtro de suavizado gaussiano (como paso bajo)** y se resta la versión suavizada de la imagen original (de forma ponderada para que los valores de un área constante permanezcan constantes), figura 18.

Figura 18: Imagen resultante después de aplicar un suavizado gaussiano y restarla a la imagen original de forma ponderada

3.4.1. Proceso aplicado

Aunque el algoritmo de máscara de enfoque es muy potente, tiene un gasto computacional elevado por lo que se opta por un filtrado de tipo Sharpen para el trabajo con la imagen completa y el algoritmo de mascara de enfoque para la zona de la matrícula.

3.5. Detección de bordes

La finalidad de estos filtros es resaltar los bordes de la imagen y destacar los detalles finos de la misma; entendiendo el término borde como un cambio abrupto en el nivel de intensidad.

Si bien dentro de este grupo se pueden englobar cualquier configuración equivalente al paso-alto en frecuencia comentadas anteriormente, es habitual utilizar el gradiente y el laplaciano para detectar las zonas de la imagen con mayor variación de intensidad.

- **Operadores discretos del gradiente:**

La detección de bordes mediante estos métodos se basa en el operador gradiente

$$\nabla f_c(x,y) = \frac{\partial f_c(x,y)}{\partial x}\vec{x} + \frac{\partial f_c(x,y)}{\partial y}\vec{y} \qquad \textbf{(9)}$$

Al trabajar con imágenes discretas, estos operadores se caracterizan por ser una aproximación a la primera derivada y tener un fuerte carácter direccional.

Existe una gran cantidad de máscaras que permiten aproximar las derivadas parciales en el ámbito discreto, pero todas tienen en común que sus coeficientes tienen valores tanto negativos como positivos, y que la suma de todos los coeficientes da lugar a cero, para que la respuesta del filtro en frecuencias espaciales sea nula.

Las versiones más conocidas son los gradientes de Roberts, Prewitt y Sobel. Todas ellas poseen dos versiones según la dirección en la que se quiere evaluar el gradiente.

- **Sobel:**

Suaviza la imagen, minimizando la aparición de falsos bordes. La utilizamos para detectar bordes horizontales y verticales principalmente. Aproximamos los gradientes mediante las siguientes formulas:

$$G_x = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7) \qquad \textbf{(10)}$$

$$\boldsymbol{G_y = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)}$$

Donde los distintos valores de z son los niveles de gris de los píxeles solapados por las siguientes máscaras, figura 19.

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Máscara 7: Máscara para obtener Gx y Gy en el punto central de la región

Figura 19: Imagen resultante de aplicar el algoritmo Sobel con orden de derivada 1

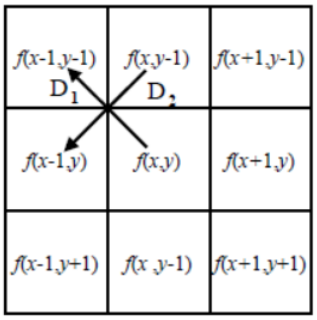
- **Prewitt**: Similar al Sobel, pero se diferencia en los coeficientes de las máscaras, figura 20.

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Máscara 8: Máscaras de Prewitt

Figura 20: Imagen resultante de aplicar Prewitt extendido

- **Roberts**: Marca los puntos sin informar de orientación. Opera según las dos diagonales perpendiculares.



$$D_1 = f(x,y) - f(x-1,y-1)$$

$$\boldsymbol{D_2 = f(x,y-1) - f(x-1,y)} \qquad \textbf{(11)}$$

$$\boldsymbol{R = \sqrt{D_1^2 + D_2^2}\ lo\ aproximamos\ a\ R = |D_1| + |D_2|}$$

En la práctica se utilizan las máscaras del gradiente en las direcciones de 45º y de 135º.

- **Operador Laplaciano:**

Es un operador de Segunda derivada. Cuya función:

$$\boldsymbol{\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}} \qquad \textbf{(12)}$$

Cuando la segunda derivada vale cero implica que la función es constante o cambia linealmente su amplitud. El cambio de signo de la función resultante indica que en este lugar existe un cruce por cero y por tanto indica la presencia de un borde, figura 21.

Figura 21: Imagen resultante de aplicar el operador Laplaciano

- **Algoritmo de Canny:**

Se fundamenta en la teoría de operadores primera y segunda derivada. Extrae bordes y cierra los contornos. Se desglosa en 3 módulos:

- **Obtención del gradiente:** Calcula la magnitud y ángulo del gradiente en cada píxel suavizando primero la imagen con un filtro gaussiano discreto. Las aristas serán perpendiculares al vector gradiente.

1. **Histéresis de umbral a la supresión no máxima:** Como la salida puede contener máximos locales creados por el ruido. Eliminamos estos máximos mediante la histéresis de umbral, que reduce falsos contornos y elimina uniones en Y en los segmentos que confluyen en un punto. Se centra en establecer dos umbrales, uno máximo y otro mínimo. Esto le ayudará a determinar si un píxel forma parte de un borde o no. Pueden darse 3 casos:
    - Si la intensidad de su gradiente es mayor que el umbral máximo, el píxel se considera parte del borde.
    - Un píxel se considera que no es borde si su valor es menor que el umbral mínimo,
    - Si está entre el máximo y el mínimo, será parte del borde si está conectado con un píxel que forma ya parte del borde.

- **Cierre de contornos abiertos:** Se utiliza el algoritmo de Deriche y Cocquerez para llevar a cabo el cierre de bordes.

En OpenCV es de los más utilizados por ser computacionalmente asequible.

3.5.1. Proceso aplicado

Inicialmente se optó por utilizar el algoritmo de Canny. No obstante, si en los ensayos se ve que es necesario ayudarle, se puede aplicar con anterioridad a Canny, los filtros Sobel o Prewitt. Trabaja sobre imágenes binarias o en escala de grises por lo que no hace falta realizar la umbralización de la imagen antes de aplicar este tipo de algoritmo. Si bien la umbralización se ve en el apartado 7 del capítulo 3, por utilizarse dentro de los posibles rectángulos asociados a placas para determinar proporciones de color y para detectar correctamente los dígitos de la matrícula.

Los resultados dependen mucho de los procesos previos realizados a la imagen, como puede ser eliminación de ruidos, ajuste de iluminación u otro tipo de suavizado. Además, siempre hay que tener en cuenta el coste computacional dado que el sistema debe trabajar sobre video en tiempo real.

3.6. Transformaciones morfológicas

La morfología matemática se basa en operaciones de teoría de conjuntos. En el caso de imágenes binarias, los conjuntos tratados son subconjuntos de Z^2 y en el de las imágenes en escala de grises, se trata de conjuntos de puntos con coordenadas en Z^3.

Las operaciones morfológicas simplifican imágenes y conservan las principales características de forma de los objetos.

Un sistema de operadores de este tipo y su composición permite que las formas subyacentes sean identificadas y reconstruidas de manera morfología óptima a partir de sus formas distorsionadas y ruidosas.

La morfología matemática se puede usar, entre otros, con los siguientes objetivos:

– Pre-procesamiento de imágenes (supresión de ruidos, simplificación de formas).

– Destacar la estructura de los objetos (extraer el esqueleto, detección de objetos, envolvente convexa, ampliación, reducción, etc.)

– Descripción de objetos (área, perímetro, etc.)

Una transformación morfologica viene dada por la relación de la imagen con otro pequeño conjunto de puntos *B* llamado elemento estructural. *B* se expresa respecto a un origen local *O* (punto representativo). En la siguiente imagen se puede ver cuatro ejemplos de elementos estructurales.

$$\begin{bmatrix}1 & 1 & 1\\ 1 & *1 & 1\\ 1 & 1 & 1\end{bmatrix}\quad \begin{bmatrix}0 & 1 & 0\\ 1 & *1 & 1\\ 0 & 1 & 0\end{bmatrix}\quad \begin{bmatrix}0 & 0 & 0\\ 1 & *0 & 1\\ 0 & 0 & 0\end{bmatrix}\quad \begin{bmatrix}1 & *1 & 1\end{bmatrix}$$

Figura 22: Elementos estructurales

La transformada *ϕ(X)* aplicada a la imagen *X* significa que el elemento estructural *B* se desplaza por toda la imagen. El resultado entre la imagen *X* y el elemento estructural *B* se almacena en el píxel actual de la imagen. Transformaciones típicas morfológicas son:

Dilatación

Combina dos conjuntos utilizando la adición de vectores. La dilatación es el conjunto de puntos de todas las posibles adiciones vectoriales de pares de elementos, uno de cada conjunto *X* y *B*.

$$\boldsymbol{X \oplus B = \{ d \in E^2 : d = x + b \quad \forall x \in X \; y \; b \in B\}} \qquad \textbf{(13)}$$

Esta dilatación es isotrópica (se comporta igual en todas las direcciones). A veces se denomina rellenado o crecimiento.

Por ejemplo, con un elemento estructura *3x3* isótropo, seria cambiar todos los píxeles del fondo que son vecinos del objeto. Tienen propiedades interesantes como: conmutativa, asociativa, invariante a la traslación, transformación creciente, figura 23

Figura 23: Imagen resultante de aplicar una dilatación con elemento estructural B identidad de 3x3

Erosión

La erosión combina dos conjuntos utilizando la substracción de vectores. Es dual con la dilatación. (No son invertibles).

$$\boldsymbol{X} \otimes \boldsymbol{B} = \{\, \boldsymbol{d} \in \boldsymbol{E}^2 : \boldsymbol{d} + \boldsymbol{b} \in \boldsymbol{X} \quad \forall\, \boldsymbol{b} \in \boldsymbol{B} \} \qquad \textbf{(14)}$$

Gráficamente, se va recorriendo la imagen, donde nos encontremos un 1, situando el origen del elemento estructural sobre ese 1, si todos los 1 del elemento estructural coinciden con el 1 de la imagen, entonces marcamos el píxel de la imagen donde está el origen del elemento estructural con 1, figura 24.

Figura 24: Imagen resultante de aplicar una erosión con elemento estructural B identidad de 3x3

Apertura y cierre

La erosión seguida de una dilatación crea una transformación morfológica llamada APERTURA. Si una imagen permanece invariable por apertura con elemento estructural *B* se dice que es abierta en *B*, figura 25.

Figura 25: Imagen resultante de aplicar una apertura con elemento estructural B identidad de 3x3

$$\boldsymbol{X} \circ \boldsymbol{B} = (\boldsymbol{X} \otimes \boldsymbol{B}) \oplus \boldsymbol{B} \qquad \textbf{(15)}$$

La dilatación seguida de una erosión crea una transformación morfológica llamada CIERRE. Y si es invariable por cierre con *B* se dice que es cerrada con respecto a *B*, figura 26.

$$\boldsymbol{X} \circ \boldsymbol{B} = (\boldsymbol{X} \oplus \boldsymbol{B}) \otimes \boldsymbol{B} \qquad \textbf{(16)}$$

Figura 26: Imagen resultante de aplicar un cierre con elemento estructural B identidad de 3x3

Se utilizan para eliminar detalles específicos de la imagen más pequeños que el elemento estructural. La forma global del contenido de la imagen no se distorsiona.

El cierre conecta objetos que están próximos entre sí, rellena pequeños huecos y suaviza el contorno del objeto rellenando pequeños valles mientras que la apertura produce el efecto contrario.

Apertura y cierre son invariantes a la traslación, son transformaciones duales, utilizadas iterativamente son idempotentes (la re-aplicación no cambia el resultado previo).

Morfología Gradiente

El efecto comentado anteriormente del operador gradiente, basado en el cálculo discreto de las derivadas parciales, resalta los elementos de alta frecuencia de la imagen, y de forma particular los contornos de la misma. Este efecto se puede conseguir sustrayendo de la dilatación el resultado de la erosión, figura 27. Véase la página 94 del libro Conceptos y métodos en Visión [13].

Figura 27: Imagen resultante de aplicar una transformación morfológica de tipo Gradiente

Morfología TopCross

El elemento estructural es en forma de cruz. Este tipo de morfología es muy útil después de aplicar un Canny para reforzar las esquinas, figura 28.

Figura 28: Imagen resultante de aplicar una transformación morfológica de tipo Cross después de haber aplicado un Canny

Morfología TopHat

Transformación suma, que mejora el contraste y resalta las formas lineales de la imagen, figura 29.

Figura 29: Imagen resultante de aplicar una transformación morfológica de tipo TopHat

3.6.1. Proceso aplicado

Se distingue dos procesos diferentes. Uno realizado sobre toda la imagen y otro realizado sobre la zona posible de matrícula.

- Sobre toda la imagen se realiza una erosión con un elemento estructural de 3x3 para remarcar los contornos. Posteriormente se realiza el proceso Canny, comentado anteriormente en detección de bordes, para que detecte los bordes y

a estos bordes se les realiza una transformación morfológica con elemento estructural de *3x3* de Cross para que ayude a detectar mejor los elementos en forma de cruz (esquinas).

- Sobre la zona posible de matrícula se aplica sobre la imagen original sin ruido una erosión, un proceso Canny y un cierre de contornos antes de realizar la umbralización y la posterior detección de bordes.

3.7. Umbralización

En el procesamiento de imágenes, la segmentación es un proceso fundamental, ya que permite diferenciar los diferentes objetos que la componen y etiquetar cada uno de ellos para su posterior aplicación de un determinado tratamiento.

Existen varios tipos de segmentación las cuales se clasifican de acuerdo a las características entre píxeles de una imagen. Esta segmentación se clasifica basada en:

- **Discontinuidad**: Se propone la división de la imagen según cambios abruptos del nivel de gris.
- **Similitud**: Comparar grupos de píxeles considerando una región a aquel grupo de píxeles que tienen propiedades similares.
- **Umbralización**: Un método básico para diferenciar un objeto del fondo de la imagen es mediante binarización (en realidad se puede considerar una medida de similitud).

La umbralización es una técnica de segmentación rápida ya que tiene un costo computacional bajo. Esta técnica toma como punto de partida un histograma de la imagen. Se trata de convertir una imagen de niveles de grises o color en una imagen binaria de tal forma que los objetos de interés se etiqueten con un valor distinto de

los píxeles del fondo. Dentro de la umbralización tenemos los siguientes tipos de umbralización:

Umbralización global

Consiste en segmentar una imagen donde los objetos (regiones de píxeles) contienen niveles de gris dentro del rango de valores y el fondo tiene píxeles con valores en otro rango disjunto. Cuando los niveles de intensidad de los objetos y del fondo son suficientemente distintos, es posible segmentar la imagen usando un umbral global aplicable a toda la imagen.

Umbralización múltiple

Esta técnica consiste en la elección de múltiples valores de umbral dentro del proceso permitiendo separar los distintos objetos dentro de una escena cuyos niveles de gris difieran el resultado no será ahora una imagen binaria, sino que los diferentes objetos (regiones) tendrán etiquetas diferentes como lo muestra la figura 30.

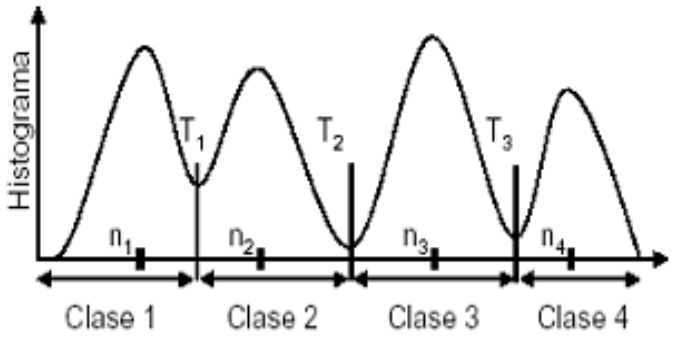


Figura 30: Multiumbralización

Umbralización local

La imagen original se divide en sub-imágenes y se encuentra un umbral para cada una de ellas por alguno de los métodos de umbralización global.

Umbralización adaptativa

En las otras técnicas el umbral o umbrales se consideran fijos independientemente de las características locales de la imagen considerada.

Sin embargo, en muchas imágenes donde la iluminación no es uniforme o en aquellas donde los objetos sean muy pequeños con respecto al fondo puede ocurrir que píxeles de un mismo objeto a segmentar tengan niveles de gris diferentes. La umbralización adaptativa consigue que el valor del umbral varíe como una función de las características locales de una imagen. La imagen se divide en sub-imágenes y para cada una de ellas se calcula un umbral. Cada sub-imagen se procesará según su propio umbral. El histograma de una imagen no tiene en cuenta la información espacial sino solamente la distribución de grises en la imagen por ello dos imágenes diferentes pueden tener el mismo histograma, ello hace que los métodos basados en la búsqueda de umbrales mediante análisis de histograma resulten limitados en algunos problemas reales.

Existen distintos algoritmos para realizar estos tipos de umbralización. Se comentan posteriormente los algoritmos implementados en OpenCV y se analiza los resultados y costes computacionales.

Umbralización global

Transforma una imagen en escala de grises a una imagen binaria mediante la fórmula 17, figura 31 y 32.

$$\boldsymbol{dst}(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} \boldsymbol{maxValor} & \boldsymbol{src}(\boldsymbol{x},\boldsymbol{y}) > \boldsymbol{umbral} \\ \mathbf{0} & \boldsymbol{demas\ casos} \end{cases} \qquad \textbf{(17)}$$

Figura 32: Imagen resultante de aplicar una umbralización con umbral 100 y maxValor 255

Figura 31: Imagen resultante de aplicar una umbralización con umbral 50 y maxValor 255

La ecuación (18) permite transformar en una función binaria inversa los resultados de la imagen mostrados en la figura 33.

$$dst(x,y) = \begin{cases} 0 & en\ los\ demas\ casos \\ maxValor & src(x,y) > umbral \end{cases} \quad \textbf{(18)}$$

Figura 33: Imagen resultante de aplicar una umbralización inversa con umbral 100 y maxValor 255

Utilizando el algoritmo del Triángulo

El método del triángulo construye una línea entre el pico y el extremo más lejano del histograma. El umbral es el punto de máxima distancia entre la línea y el histograma. Esta implementación utiliza una estimación robusta (el valor predeterminado es 1% y 99%) de los extremos del histograma, figura 34.

Figura 34: Imagen resultante de aplicar una umbralización mediante el algoritmo del Triángulo

Utilizando el algoritmo de Otsu

Dada una imagen con *L* niveles de intensidad y asumiendo que el umbral buscado es *T*, las probabilidades acumuladas hasta *T* y desde *T* hasta *L* son:

$$w_1(t) = \sum_{z=1}^{T} P(z) \quad \text{y} \quad w_2(t) = \sum_{z=T+1}^{L} P(z)$$

Medias y varianzas asociadas

$$\mu_1(t) = \sum_{z=1}^{T} zP(z) \quad \text{y} \quad \mu_2(t) = \sum_{z=T+1}^{L} zP(z)$$

$$\sigma_1^2(t) = \sum_{z=1}^{T} (z-\mu_1(t))^2 \frac{P(z)}{w_1(t)} \quad \textbf{y} \quad \sigma_2^2(t) = \sum_{z=T+1}^{L} (z-\mu_2(t))^2 \frac{P(z)}{w_2(t)}$$

(19)

Varianza ponderada

$$\sigma_w^2(t) = w_1(t)\sigma_1^2(t) + w_2(t)\sigma_2^2(t)$$

Con este método automático nos evitamos tener que elegir el umbral. Se elige el umbral como la mínima varianza ponderada, figura 35.

Figura 35: Imagen resultante de aplicar una umbralización mediante el algoritmo de Otsu

Se puede combinar una umbralización binaria con un algoritmo del Triángulo o con el algoritmo de Otsu, figura 36.

Figura 36: Imagen resultante de aplicar una umbralización mediante el algoritmo de Otsu + binario con umbral 100

En algunos casos puede ser interesante aplicar este procesamiento sobre imágenes convertidas en espacio de color HSV y aplicar esta umbralización sobre *H* para hacer una umbralización por color.

Analizando los resultados, tanto en el algoritmo de Otsu como en el del Triángulo que determina automáticamente el umbral se observa cómo en el caso de Otsu marca la sombra de la luz de la matrícula como negro y en el caso del Triángulo la zona de las estrellas y el número 7 no se ven correctamente. Esto es debido a que el cálculo del umbral lo realiza basándose en el cálculo de toda la imagen. Sería interesante realizar

una umbralización local. Se analiza las opciones de umbral adaptativo en el siguiente apartado.

Umbral adaptativo

Transforma una imagen en escala de grises a una imagen binaria mediante la siguiente fórmula:

$$dst(x,y) = \begin{cases} maxValor & src(x,y) > T(x,y) \\ 0 & demas\ casos \end{cases} \quad \textbf{(20)}$$

Donde *T(x,y)* es una función de umbral que se puede elegir entre dos métodos:

- ADAPTIVE_THRESH_MEAN_C: implementa la media de los valores de los vecinos (el tamaño se puede configurar) menos el valor *C*, figura 37.

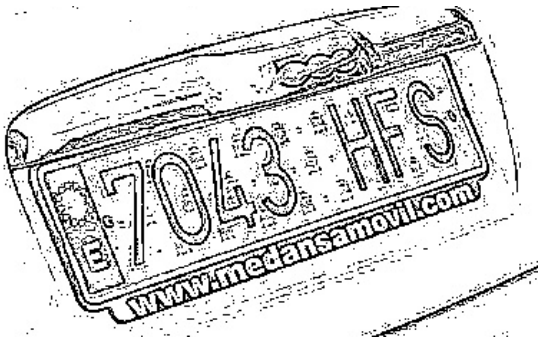


Figura 37: Imagen resultante de aplicar un umbral adaptativo con tamaño de bloque de vecinos de 3, un C de 3 y función de umbral de promedio

- ADAPTIVE_THRESH_GAUSSIAN_C: realiza una suma ponderada (correlación cruzada con una ventana gaussiana) de los valores de los vecinos menos *C*, distintos valores de vecinos se muestran en las figuras 38, 39 y 40.

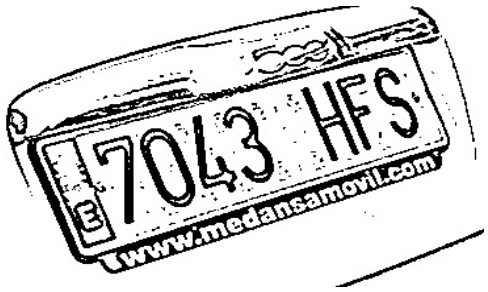


Figura 38: Imagen resultante de aplicar un umbral adaptativo con tamaño de bloque de vecinos de 11, un C de 7 y función de umbral gaussiana

Figura 39: Imagen resultante de aplicar un umbral adaptativo con tamaño de bloque de vecinos de 21, un C de 7 y función de umbral gaussiana

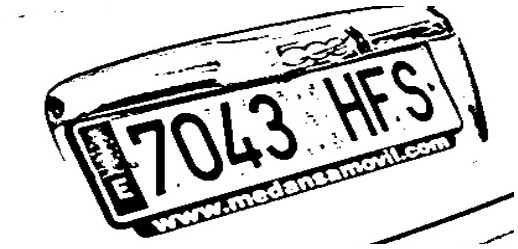


Figura 40: Imagen resultante de aplicar un umbral adaptativo con tamaño de bloque de vecinos de 31, un C de 7 y función de umbral gaussiana

En función del valor del tamaño de los vecinos a tener en cuenta se puede conseguir eliminar los elementos del fondo de la matrícula, pero también se pierde calidad en los detalles.

También permite la transformación en una imagen binaria inversa donde:

$$dst(x,y) = \begin{cases} 0 & en\ los\ demas\ casos \\ maxValor & src(x,y) > T(x,y) \end{cases} \quad \textbf{(21)}$$

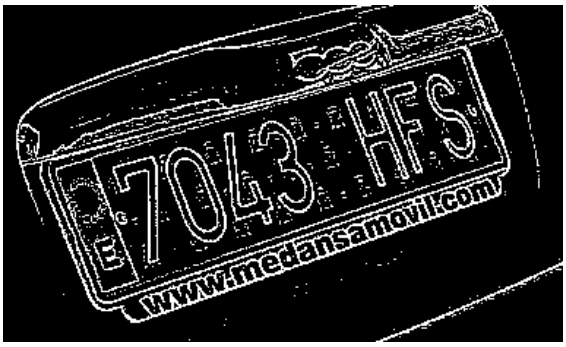


Figura 41: Imagen resultante de aplicar un umbral adaptativo binario inverso con tamaño de bloque de vecinos de 3, un C de 7 y función de umbral promedio

En nuestro caso interesa detectar tanto los dígitos como la propia placa en la que se encuentran los dígitos. Lo más importante inicialmente sería distinguir el rectángulo correspondiente a la placa. Siempre se podría volver a aplicar los pasos de pre-procesamiento anteriormente realizados ya concretamente sobre esta zona.

La mayoría de las matrículas están remarcadas por una zona gris o negra que se distingue visualmente bastante bien, figura 42.

Figura 42: Ejemplo de matrículas actuales

Pero existen casos como en las figuras 43,44 y 45, en las que el coche es blanco y el contorno de la matrícula también es casi blanco, por lo que el contorno no se remarca destacadamente, como sería deseable.

Figura 43: Imagen original de coche blanco con matrícula remarcada en gris

Figura 44: Imagen resultante de aplicar un umbral adaptativo con tamaño de bloque de vecinos de 21, un C de 7 y función de umbral gaussiana

Imagen 45: Imagen resultante de aplicar un umbral adaptativo con tamaño de bloque de vecinos de 31, un C de 7 y función de umbral gaussiana

Para poder comprobar realmente los resultados debemos pasarles después la función de detectar contornos como se ve en la parte de debajo de las imágenes de las figuras 44 y 45.

En estos casos resulta muy difícil el encontrar un contorno viable, aunque las letras han sido detectadas correctamente. En el capítulo 4 se comenta cómo determinar una zona posible de matrícula.

3.7.1. Proceso aplicado

Los resultados de la umbralización, como puede deducirse de lo comentado anteriormente en este apartado, dependen mucho de la resolución de la imagen y de las transformaciones realizadas anteriormente. La umbralización global requiere de un conocimiento de la imagen para determinar los valores umbral adecuados, ese conocimiento en el caso que nos ocupa, aunque sí se conoce que el fondo de la matrícula es blanco y los números negros.

Como se ha visto en los ejemplos en las matrículas donde el contorno es muy similar al color del coche es difícil determinar el valor de umbral adecuado que permita que

se vea correctamente en estos casos y que no genere muchos manchones en negro asociados a las sombras en otros casos, por eso será más adecuado una umbralización adaptativa, aunque también se genera bastante ruido en determinadas imágenes con bastante calidad.

Para cubrir las posibles variaciones se realizan dos pasadas por el fotograma, una aplicando una umbralización global con umbral 100 y valor máximo de 255 y con método binario más el algoritmo de Otsu y otra con una umbralización adaptativa con tamaño de bloque de 21 y con *C* de 7.

4. DETECCIÓN DE LA MATRÍCULA

En este capítulo se describe el procedimiento seguido para la detección de la placa. En el primer apartado se explican las características de las placas en uso españolas. En el segundo apartado la extracción de contornos, en el tercer apartado los métodos utilizados para la extracción de candidatos posibles basados en las características de las letras y de las placas. En el cuarto apartado se explica la rectificación de las letras y la división en elementos inferiores si fuera necesario. Y, finalmente en el quinto apartado los métodos comparados para determinar las letras de la matrícula.

4.1. Características de la placa

Las características de las placas españolas actuales con la que se trabaja son las siguientes, recogidas del BOE 223 del 16 de septiembre del 2000 [14]:

Tenemos 4 tipos de matrículas actualmente y dentro de cada tipo se distingue entre numeración actual que corresponde a cuatro dígitos (empezando en el 0000 y acabando en el 9999) y tres letras (empezando en BBB y acabando en ZZZ). De las tres letras posibles se han eliminado las vocales y las letras Ñ, Q, CH y LL. Y también existe numeración con siglas de provincia, en las que tenemos una o dos letras para la provincia, cuatro números y dos letras.

Los distintos tipos de matrícula se muestran en las figuras 46 al 49:

- Ordinaria larga:

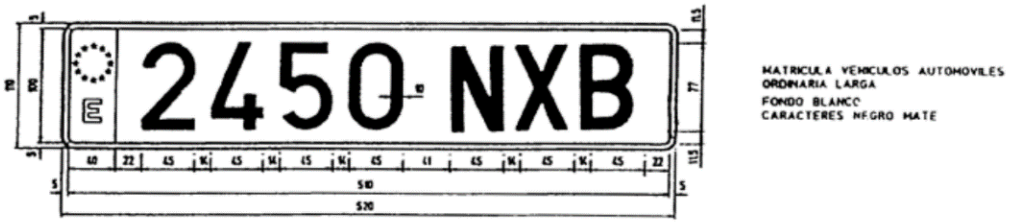


Figura 46: Dimensiones de las matrículas ordinarias largas

- Ordinaria alta:

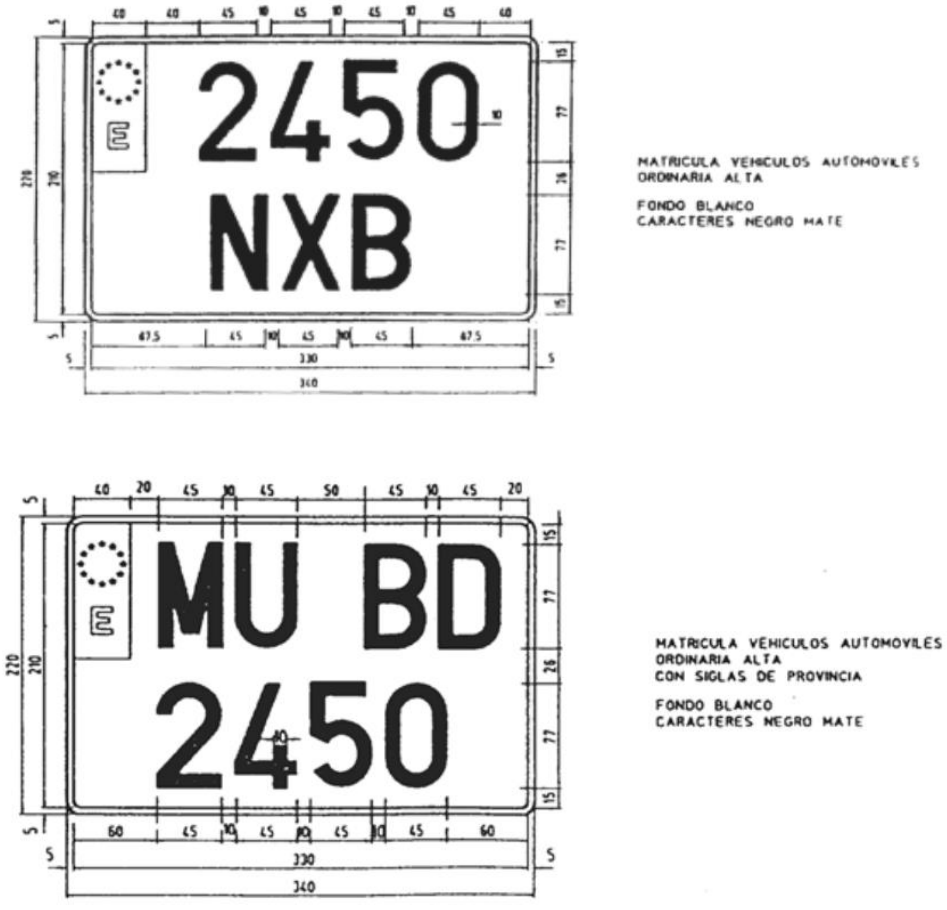


Figura 47: Dimensiones de las matrículas ordinarias altas

- Ordinaria larga delantera:

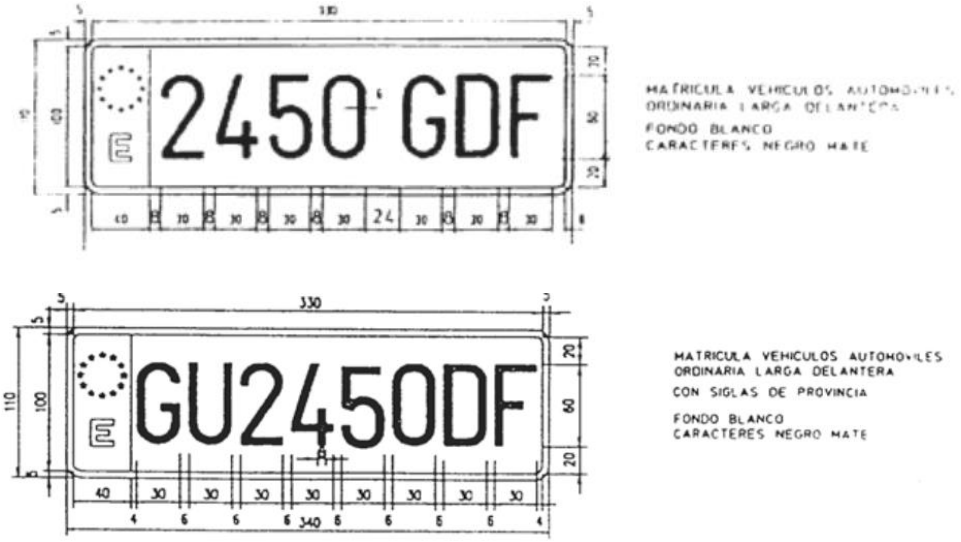


Figura 48: Dimensiones de las matrículas ordinarias largas delanteras

- Motocicleta ordinaria:

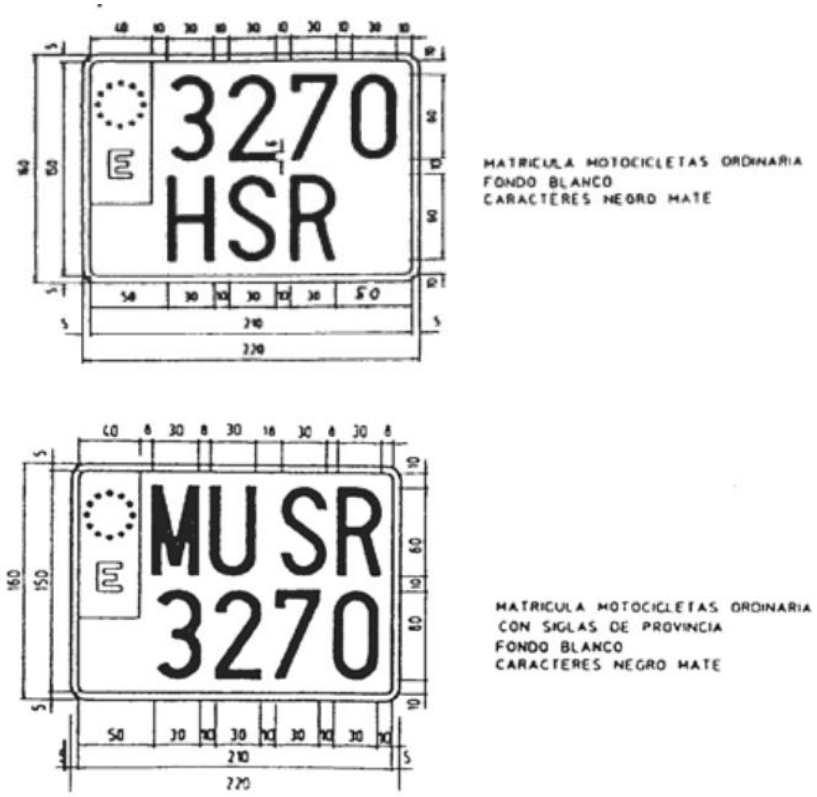


Figura 49 Dimensiones de las matrículas ordinarias de motocicletas

En la tabla 1 se recopilan las medidas de las matrículas oficiales.

TIPO PLACA MATRÍCULA	Medidas (mm)		Caracteres (mm)			Separación entre caracteres (mm)	Espacio entre grupos de caracteres (mm)	Separación a bordes (mm)						Color		Tolerancia entre caracteres, grupo de caracteres y distancia a bordes
	Total	Superficie reflectante	Anchura	Altura	Grueso trazo			Horizontales	Entre líneas	Verticales línea superior		Verticales línea inferior		Fondo	Caracteres	
										Izq.	Der.	Izq.	Der.			
Ordinaria larga (1)	520 × 110	510 × 100	45	77	10	14	41	11,5	–	62	22	–	–	B	N	
Ordinaria alta (1)	340 × 220	330 × 210	45	77	10	10	–	15	26	80	40	87,5	87,5	B	N	
Ordinaria larga delantera (2)	340 × 110	330 × 100	30	60	6	8	24	20	–	48	8	–	–	B	N	
Motocicletas ordinaria	220 × 160	210 × 150	30	60	6	10	–	10	10	50	10	50	50	B	N	

Tabla 1: Dimensiones de las matrículas actuales españolas

Otra característica muy importante, es que todas las matrículas, tendrán dentro del contorno o a su izquierda, una zona azul con las propiedades definidas en la figura 50. Esta característica resulta útil en imágenes en las que no se distingue bien el contorno o para eliminar este contorno de la lista de contornos de letra antes de aplicar la identificación de los mismos.

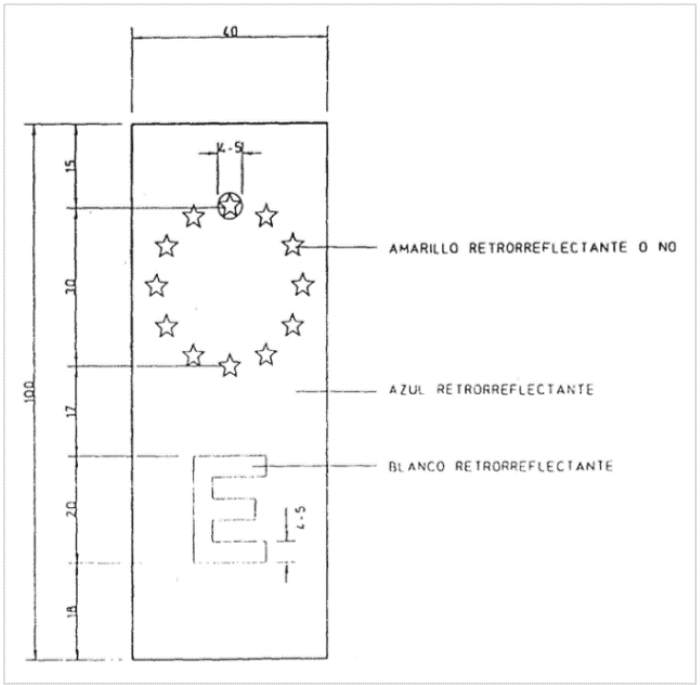


Figura 50: Dimensiones y colores del símbolo comunitario

4.2. Extracción de contornos

Considerando el enfoque estadístico, la extracción de regiones se realiza con la función de OpenCV de encontrar contornos, ***findContours***, que se describe a continuación.

La función recupera los contornos de la imagen binaria mediante el algoritmo de Suzuki85. Devuelve el número de contornos recuperados. Los contornos podemos transformarlos a un vector de puntos.

Como parámetro debemos indicar el tipo de jerarquía en la búsqueda de contornos y el tipo de Aproximación (ninguna, simple, aproximación TC89_L1 o TC89_KCOS).

En función del tipo de jerarquía indicada nos devuelve el número del contorno siguiente, el contorno previo, el primer hijo y el padre (contorno exterior). Indica con NULL (-1) si no se detectan los contornos.

Esta jerarquía nos ayudará para identificar que contornos son interiores de otro, como el caso del 0 en la figura 51.

Figura 51: Imágenes resultantes después de la aplicación de la función para encontrar contornos

Analizando los resultados obtenidos en el paso anterior se observa que realmente es difícil muchas veces de identificar un rectángulo perfecto, bien porque las líneas están partidas, o por ruido u otros factores a veces se unen con otros contornos no deseados.

Para solventar la problemática anterior, se intenta realizar una aproximación poligonal de contornos mediante la función ***approxPolyDP***. Esta función nos permite aproximar una forma de contorno a otra forma con menos número de vértices dependiendo de la precisión especificada. Es una implementación del algoritmo Douglas-Peucker [15]. La precisión es la máxima distancia entre la curva original y su aproximada. Este método se aparece como solución en muchos de los artículos y tesis revisadas sobre el tema.

Se recomienda el 5% de la longitud del contorno, figura 52, pero los resultados no son adecuados teniendo en cuenta el tamaño de una matrícula. Finalmente se opta por utilizar el 1% de la longitud del contorno, figura 53.

Figura 53: Imagen resultante de aplicar una aproximación poligonal con precisión de un 1% de la longitud del contorno

Figura 52: Imagen resultante de aplicar una aproximación poligonal con precisión de un 5% de la longitud del contorno

Realmente no se aprecia una gran mejoría en aproximar la curva en casos complejos, no se consigue obtener una curva de 4 lados que correspondería a un rectángulo, pero sí se ha conseguido que las líneas sean menos sinuosas. Finalmente, en el presente trabajo se opta por esta aproximación sobre todo en los caracteres para eliminar pequeñas imperfecciones.

Otra posible solución es utilizar la función ***convexHull***, que nos crea la curva convexa del contorno, figura 54.

Imagen 54: Imagen resultante de aplicar la función convexHull para obtener la curva convexa

Los resultados no son satisfactorios en casos difíciles como es la imagen anterior, por lo que en estos casos se busca el rectángulo rotado que cubre el contorno mediante la función de ***minAreaRect*** y se considera como base para aplicar el siguiente proceso, aunque no haya obtenido una aproximación adecuada, figura 55.

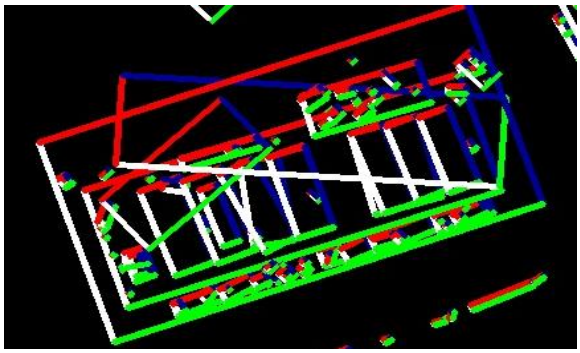


Figura 55: Imagen que representa los distintos rectángulos rotados mediante la función minAreaRect que cubren los contornos

4.3. Extracción de candidatos

Se denota *R* como la región de la placa extraída anteriormente con el tamaño W*x*H*,* donde *W* y *H* son el ancho (*width*) y alto (*height*) de *R*, respectivamente.

Para eliminar regiones poco probables o difíciles de identificar los criterios son los siguientes aplicados sucesivamente:

- **Criterio 1**: El primer criterio es la longitud de la placa. Si no es lo suficientemente grande, los caracteres no podrán ser identificados. Se considera que *W* debe ser mayor a 60 para que realmente se puedan identificar posteriormente los caracteres. Este resultado de la característica de firma se obtiene tras la inspección realizada sobre 50 imágenes de placas.
- **Criterio 2**: Relación entre *W* y *H* de la placa, tabla 2.

Tipo de matrícula	Relación anchura/altura
Ordinaria larga	4,72727273
Ordinaria alta	1,54545455
Ordinaria larga delantera	3,09090909
Motocicletas	1,375

Tabla 2: Relación anchura/altura de las matrículas

Dado las diferencias existentes por la perspectiva se elige entre 0.9 y 5. Con estos dos criterios se pasa de 2500 contornos a 50 contornos a evaluar.

- **Criterio 3:** La densidad de la región con letras posibles (zona negra) podemos considerarla como la división del área posible de letras, respecto al área del rectángulo de la placa.

$$\boldsymbol{Densidad_{letra}} = \frac{\sum_{letras} \boldsymbol{Area}}{\boldsymbol{w \cdot h}} \quad \textbf{(22)}$$

$$\boldsymbol{Densidad_{blanco}} = \frac{\sum_{blancas} \boldsymbol{Area}}{\boldsymbol{w \cdot h}} \quad \textbf{(23)}$$

Analizando las características de densidad de color en 50 imágenes, se determina que el valor de relación *densidad_letra* debe ser mayor que el 8%**.** También se determina la densidad del fondo blanco sobre el área de la placa. Este valor debe ser superior al 70%.

Con este criterio se reduce el proceso de detección de 50 contornos a 4 o 5 contornos.

- **Criterio 4**: Analizamos las características de los contornos interiores del rectángulo en base al conocimiento que tenemos de las letras.
    - La altura de los caracteres debe ser similar.
    - La anchura de los caracteres debe ser similar.
    - Los centros de los caracteres están en la misma línea o en dos líneas más o menos paralelas, como es el caso de motocicletas.

  Con estas características se puede crear un conjunto de reglas para determinar si es o no una placa, en base a los contornos interiores:

    - **Regla 1:** Si encontramos 6 o más contornos con altura y anchura similar. Hay que tener en cuenta que nos podemos encontrar una matrícula antigua con una sola letra identificativa de la provincia y una sola letra de matrícula después de los 4 números.

    - **Regla 2:** La altura de estos contornos debe ser mayor que la mitad de la altura de la placa en el caso de matrículas ordinarias y en el caso de matrículas altas o de motocicletas debe ser mayor a la cuarta parte de la altura de la placa. Como la región obtenida puede no ser muy precisa, en la práctica se permite que sea mayor que el 40% de la altura de la placa.

- **Regla 3:** La anchura de estos contornos debe ser inferior a la anchura de la placa dividido entre el número mínimo de dígitos posibles en una línea (en el caso de matrículas largas son 6, y en el caso de matrículas altas o motocicletas son 3).

- **Regla 4:** La anchura de los contornos debe estar entre el 30% y el 90% de la altura de estos contornos. La relación entre altura y anchura debe estar entre 1.5 y 3.

La aplicación de estas reglas es un proceso laborioso no tanto por los conceptos de visión artificial sino por el proceso matemático asociado con las siguientes circunstancias:

- Se deben identificar los contornos que cumplan las reglas 2, 3 y 4, en el caso de que existan contornos interiores (contorno interior del 0 por ejemplo) ya que como no cumplen los criterios se habrán eliminado, y debemos incorporarlo posteriormente a nuestro objeto letra.
- Se debe comprobar si existe otro contorno cuyo centro sea muy similar y cuyo tamaño sea también similar a uno de los seleccionados, y en este caso no se unen a nuestra selección.
- Comprobar que los centros de los contornos se encuentren en una línea similar, tanto si la placa es ordinaria larga como en dos líneas si es ordinaria alta o motocicleta.
- La comprobación de que los contornos tenga una altura y anchura similar.
- La comprobación de que un contorno sea exterior y no participe en el cómputo como posible de los contornos interiores del mismo.
- La comprobación de si un contorno es correcto, pero no se ha partido en dos por una alguna pequeña imperfección.

Si se cumplen estos cuatro criterios se dice que la región *R* es un candidato posible. Con esto se pasa al siguiente apartado, que sería el procesado de la placa candidata.

4.4. Procesado de la placa

Estos pasos son realizados para la utilización del enfoque estadístico y aplicación del clasificador KNN. En el caso del clasificador HAAR y las redes neuronales no se realiza tratamiento, sino que se entrena con las zonas de imágenes preprocesadas anteriormente y con zonas de imágenes sin procesar.

Primer paso: Se ordenan los contornos candidatos a letras en función de la posición de su centro en el eje de abscisas *X*, cuando hablemos de una matrícula larga o larga ordinaria. Si es una matrícula alta o de motocicleta, se tiene en cuenta tanto la posición del centro en el eje de abscisas *X* como en el eje de coordenadas *Y*.

Segundo paso: Se añade de los contornos interiores que han sido eliminados en el proceso anterior.

Tercer paso: Se comprueba de existencia de una zona azul al principio de los contornos candidatos para ser letras. Si fuera así se elimina de la lista de contornos.

Cuarto paso: Se crea una imagen correspondiente a la letra, en la cual se rellena el contorno exterior y los contornos interiores con la función ***drawContours*** con la opción de relleno, posteriormente se realizar una substracción de imágenes para obtener correctamente la letra, como se puede ver en la figura 56.

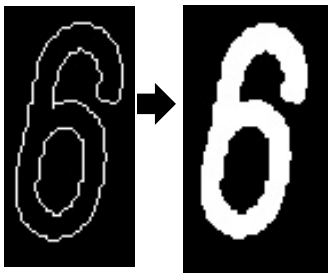


Figura 56: Imagen resultante de rellenar los contornos y realizar una substracción de imágenes

Quinto paso: Una vez encontrado el conjunto de candidatos a letras, se procede a realizar una transformada afín sobre los contornos candidatos que nos permitan enderezar dichos contornos.

La función de OpenCV que nos ayuda para la corrección es ***warpAffine,*** en la que se debe pasar la matriz Afín que se genera mediante la función ***getRotationMatrix2D*** indicando el centro de giro, el ángulo de giro y la escala, figura 57.

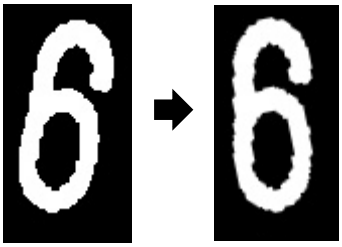


Imagen 57: Imagen resultante de aplicar una transformada afín

Sexto paso: Consistente en el cálculo de propiedades. Para ello calculamos los momentos normalizamos de los contornos, las propiedades de compatibilidad, redondez y compacidad. Cálculo de la envolvente convexa y las propiedades y momentos de dichas envolventes. Estos valores se introducen en el vector de atributos.

Una vez preparado todos los datos se pasa al siguiente proceso que se materializa en el algoritmo de clasificación.

5. DETERMINACIÓN DE LA CORRESPONDENCIA DE DÍGITOS. RECONOCIMIENTO DE PATRONES

El reconocimiento de patrones es la disciplina científica cuyo objetivo es la clasificación de objetos en un cierto número de categorías o clases.

Históricamente los dos enfoques más utilizados han sido el estadístico (bajo la teoría de la decisión) y el sintáctico (o estructural). Recientemente el desarrollo de redes neuronales ha proporcionado un nuevo enfoque. El aumento en capacidad de almacenamiento ha propiciado el reconocimiento basado en la apariencia.

El enfoque estadístico y el de redes neuronales utilizan patrones de los que se extraen propiedades de naturaleza cuantitativa, mientras que el enfoque sintáctico se fundamente en las relaciones geométricas asociadas a las formas de los objetos y el enfoque basado en la apariencia considera distintas formas de vista de los mismos.

En el apartado 5.1 y 5.2 se describe el enfoque estadístico creando un clasificador KNN para la clasificación de los dígitos de la matrícula. Y en los apartados 5.3 y 5.4 se analizan los métodos de clasificación HAAR y las redes neuronales como procedimientos para la detección en el primero caso tanto de la matrícula como de los dígitos y en el segundo caso de la clasificación de dígitos de la matrícula.

En el apartado 5.5 se analiza otros métodos utilizados comúnmente como son los OCR y MATCH.

5.1. ENFOQUE ESTADISTICO

Se debe considerar cuáles de las propiedades de los contornos pueden ser útiles para hacer un agrupamiento y clasificación de los mismos. Estas propiedades o atributos pueden ser propiedades métricas, topológicas o basadas en las irregularidades.

5.1.1. Propiedades métricas

Existen varias propiedades que pueden ser interesantes para crear los atributos de las clases, tales como perímetro, compatibilidad, centroide, elongación o razón de aspecto, ejes mayor y menor, redondez, compacidad y orientación de la región.

De estas propiedades algunas pueden ser importantes para la diferenciación entre caracteres y otras no.

Por ejemplo, para calcular la elongación o razón de aspecto circunscribimos un rectángulo a la figura y calculamos la razón de aspecto del rectángulo circunscrito. Al cociente de las longitudes de estos ejes, le llamamos excentricidad de la región. En nuestro caso esta propiedad no es significativa, porque todas las letras realmente tienen una excentricidad similar. Aunque se ha utilizado para eliminar contornos no válidos. La orientación tampoco resulta útil dado que las regiones después de los procesos anteriores aparecen rectas.

Se comprueban otras propiedades tales como el centroide, la compatibilidad, redondez y compacidad entre las imágenes base para la clasificación y las imágenes reales obtenidas tras el procesamiento anterior en la tabla 3.

$$\boldsymbol{Centroide} \quad \bar{x} = \frac{1}{A}\sum_i x_i \quad \bar{y} = \frac{1}{A}\sum_i y_i \tag{24}$$

$$\boldsymbol{Compatibilidad} = \frac{P^2}{A} \tag{25}$$

$$\boldsymbol{Redondez} = \frac{A}{(eje\ mayor)^2} \tag{26}$$

$$\boldsymbol{Compacidad} = \frac{\sqrt{A}}{eje\ mayor} \tag{27}$$

Imagen de referencia	Compatibilidad Redondez Compacidad	Nuestra imagen	Compatibilidad Redondez Compacidad
	15.429767803632906 0.4623698018638891 0.6799777951256122		17.472345044720857 0.3728190930330118 0.6105891360260284
	33.20282494128282 0.18664965986394558 0.4320296978958108		29.014461623091094 0.1951530612244898 0.4417613170304636
	56.75510382861027 0.27420663907150394 0.523647437758941		53.27803521259523 0.32037639335239554 0.5660180150422737
	51.40994989977154 0.3031632653061225 0.5506026383029076		53.01436385766427 0.22633943959566205 0.4757514472869862
	52.60761328183836 0.23991735537190081 0.48981359247360706		53.158353984236044 0.2176825189487819 0.46656459247223414
	53.7382062980276 0.24681438194951708 0.4968041686112518		48.68512620086944 0.203110704118906 0.45067804929783967

Tabla 3: Comparación de compatibilidad, redondez y compacidad

En base a estas características podemos considerar estos parámetros para realizar un agrupamiento inicial, aunque se observa cómo los resultados para los números 2,3 y 4 por ejemplo, son resultados muy parecidos. Se deben buscar otras propiedades que nos puedan clasificar mejor, para ello se indaga dentro de las propiedades topológicas.

5.1.2. Propiedades topológicas

Se consideran algunas de las propiedades topológicas, invariantes a ciertas deformaciones de las figuras en la imagen. Las propiedades no pueden involucrar nociones de distancia ni de forma directa ni indirecta.

Una de las descripciones topológicas más usadas son el número de sus componentes conexas. Una componente conexa de un conjunto es un subconjunto de dimensión máxima tal que cualquiera dos de sus puntos puede unirse por una curva continua pertenecientemente enteramente al subconjunto. Una segunda propiedad es el número de huecos en la figura.

Formalmente, el número de huecos en una figura es uno menos que el número de componentes conexas en el complemento de la figura. Siendo *C* el número de componentes conexas de una figura y *H* el número de huecos, definimos el número de Euler *E = C- H* que también es una propiedad topológica.

Dado que la mayoría de los caracteres se caracterizan por tener solo de una a tres componentes conexas y los huecos varían entre 0 y 2. No existe una variación significativa como para tenerlo en cuenta en este proyecto.

5.1.3. Propiedades basadas en las irregularidades

La envoltura convexa ("*convex hull*") *H* de un conjunto arbitrario *S* es el conjunto convexo más pequeño conteniendo *S*. Si *S* es convexo se cumple que *H = S*. Si *S* tiene solamente una componente conexa, entonces *H* puede verse como el conjunto encerrado por una goma elástica alrededor del perímetro de *S*. El conjunto diferencia *H-S* se denomina deficiencia convexa *D* del conjunto *S*.

Mediante la función ***convexHull*** encontramos la lista de puntos de la envoltura convexa usando el algoritmo de Sklansky's. En la siguiente tabla 4 vemos el contorno con su envoltura convexa marcada en rojo.

Podemos encontrar los defectos entre nuestra envoltura y nuestro contorno, mediante la función **convexityDefects** que nos permite obtener para cada contorno el punto inicial, el punto final, y el punto más cercano entre la envoltura y el contorno.

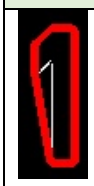
Contorno y cobertura convexa						
						

Tabla 4: Contornos y cobertura convexa

5.1.4. Momentos invariantes de HU

Otra propiedad importante para seleccionarla para el algoritmo de clasificación, son los momentos invariantes de una región.

Cuando una región se da en términos de sus puntos interiores, podemos describirlas mediante un conjunto de momentos que son invariantes a estos efectos.

Tomamos *f*(*x*,*y*) como la intensidad del punto (*x*,*y*) en una región. El momento de orden (*p*+*q*) para la región se define como:

$$\boldsymbol{m_{pq}} = \sum_x \sum_y \boldsymbol{x^p y^q f(x,y)} \qquad \textbf{(28)}$$

Donde el sumatorio se toma sobre todas las coordenadas espaciales (*x*,*y*) de puntos de la región.

El momento central de orden (*p*+*q*) es:

$$\boldsymbol{\mu_{pq}} = \sum_x \sum_y \boldsymbol{(x-\bar{x})^p (y-\bar{y})^q f(x,y)} \qquad \textbf{(29)}$$

Donde $\bar{x}$ y $\bar{y}$ son las siguientes medias:

$$\bar{x} = \frac{m_{10}}{m_{00}} \quad \bar{y} = \frac{m_{01}}{m_{00}} \qquad \textbf{(30)}$$

Los momentos centrales normalizados de orden (*p*+*q*) se definen como:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}} \; donde \; \gamma = \frac{p+q}{2} + 1 \; para (p+q) = 2, 3 \ldots . \; \textbf{(31)}$$

El siguiente conjunto de momentos invariantes se pueden obtener usando únicamente los momentos centrales normalizados de orden 2 y 3.

$$\phi_1 = \eta_{20} + \eta_{02}$$

$$\phi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$$

$$\phi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$$

$$\phi_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2$$

$$\phi_6 = (\eta_{20} - \eta_{02})\left[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})$$

$$\phi_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})\left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2\right] + (3\eta_{12} - \eta_{03})(\eta_{21} + \eta_{03})\left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right]$$

$$\phi_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})\left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2\right] + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})\left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right]$$

(32)

Para que todos los momentos contribuyan por igual en la función de discriminación y por tanto estén dentro del mismo orden de magnitud, es necesario proceder a una normalización.

$$\phi'_n = abs(Ln(abs(\phi_n)) \qquad \textbf{(33)}$$

Este conjunto es invariante a la traslación, rotación y escala, por lo que podemos utilizarlos para comparar contornos.

La función que nos permite obtener inicialmente los momentos centrales Mpq a partir del contorno es ***Imgproc.moments*** a la cual pasamos un contorno e indicamos como opción si la imagen es binaria.

Mediante esta función podemos obtener el centroide, media, varianza y otros datos de interés.

Posteriormente calculamos los momentos HU mediante la función **HuMoments.** De esta función obtenemos directamente todos los momentos invariantes que debemos pasar a una variable para poder trabajar con ellos. Realizamos la normalización mediante la ecuación (31).

En la tabla 5 comparamos los momentos de los contornos de la imagen de referencia con los momentos de los contornos de las imágenes adquiridas, dando idea de su comportamiento, que es extensible al resto de imágenes analizadas.

Imagen	Momentos Hu de la imagen	Momentos Hu de nuestra imagen normalizados
7	hm1 0.5054309978111237 hm2 0.11160904869309524 hm3 0.06663726520039907 hm4 0.018621912107506838 hm5 5.845575896573881E-4 hm6 0.005808144344844961 Hm7 -2.97675813761527E-4	hm1 0.68234375268541 hm2 2.19275315084250 hm3 2.70849132037502 hm4 3.98341632134721 hm5 7.44465525383491 hm6 5.14849414900590
7	hm1 0.38142376740312217 hm2 0.06969953310720053 hm3 0.022169665342426332 hm4 0.006106308660527794 hm5 6.3377824762856E-5 hm6 0.0014332304038798362 Hm7 -3.2108767668045625E-5	hm1 0.9638442715457458 hm2 2.663561659843539 hm3 3.8090303505320646 hm4 5.098432835582225 hm5 9.66639652485127 hm6 6.547824358775907
0	hm1 0,248377846 hm2 0,035340606 hm3 2,090199E-06 hm4 1,796720E-07 hm5 -1,063457E-13 hm6 -3,254746E-08 hm7 2,853275E-14	hm1 1.4170656310340712 hm2 3.4177044049783465 hm3 16.718019329805355 hm4 18.699087028071776 hm5 36.410659605872546 hm6 20.45089269858368 hm7 1.4170656310340712

	hm1 0.30286037055705245 hm2 0.06568070578643902 hm3 5.542507870927075E-6 hm4 5.948747321961258E-7 hm5 -9.0351556266128E-13 hm6 -1.34741042728909E-7 hm7 -5.9196559410054E-13	hm1 1.0365297605365964 hm2 2.308644426391836 hm3 12.330903837443406 hm4 12.698523847805816 hm5 25.213785389426732 hm6 13.855426450510237 hm7 28.621830274020933
	hm1 0,396412159 hm2 0,048201196 hm3 0,039427302 hm4 0,009127011 hm5 1,59E-04 hm6 0,001609644 hm7 -6,75E-05	hm1 0.9253008034546255 hm2 3.032371436197307 hm3 3.2332967509510584 hm4 4.696517072910915 hm5 8.743820981037985 hm6 6.431742426544116 hm7 0.9253008034546255
	hm1 0.3656739502512653 hm2 0.0657790668270378 hm3 0.020386729252357117 hm4 0.006261726178729748 hm5 6.847999534851512E-5 hm6 0.0013985954908886625 hm7 -1.7770377372883713E-5	hm1 1.1355638181392207 hm2 3.0670369834927245 hm3 4.29883253690366 hm4 5.594506028289946 hm5 10.60098683369545 hm6 7.318503442667926 hm7 1.1355638181392207

Tabla 5: Momento Hu

Es preciso señalar que las pequeñas variaciones en la imagen debido a la perspectiva, al procesamiento realizado anteriormente, y otras características hacen que no se puedan comparar directamente. Pero sí que nos sirven, junto a las propiedades métricas descritas anteriormente, para generar nuestro clasificador. Se ve que tiene mucho más sentido utilizar los valores de los momentos normalizados. Con tal propósito se utilizan los 6 primeros momentos porque el momento 7 en algunas imágenes no proporcionan valores adecuados al normalizarlos.

También nos interesa tener los momentos invariantes y las características métricas de la envoltura convexa para posteriormente aplicar el algoritmo de Clasificación. Estos resultados se muestran en la tabla 6 con algunos ejemplos ilustrativos.

Imagen	Momentos Hu normalizados de la envoltura convexa de referencia		Momentos Hu normalizados de la envoltura convexa real	
	hm1 1.4649005188799975 hm2 3.8414636577529664 hm3 5.711783213116601 hm4 7.559446269578358 hm5 14.211996406250845 hm6 9.53635481999451 hm7 15.896105942561023		hm1 1.48689318480805 hm2 3.853196789596131 hm3 6.108223558259431 hm4 7.848570961469865 hm5 14.827959631991995 hm6 9.780135061387568 hm7 17.93908111572806
	hm1 1.5790056839369828 hm2 4.384277781006042 hm3 6.14599062895486 hm4 8.203646180907958 hm5 15.385539734368946 hm6 10.428258875764199 hm7 1.5790056839369828		hm1 1.50339465181403 hm2 3.894183391717087 hm3 6.2680146096186276 hm4 7.9573908129161515 hm5 15.090414909299065 hm6 9.981401749605219 hm7 1.50339465181403
	hm1 1.6536642902446872 hm2 4.636776383920248 hm3 9.184762409909007 hm4 13.506113071017378 hm5 26.087344859686596 hm6 16.41709509413789 hm7 24.89566630989111		hm1 1.6144583072370018 hm2 4.344231450523178 hm3 9.35587598137378 hm4 12.31897290350491 hm5 23.25046715939889 hm6 14.557185867902211 hm7 24.037980490329893
	hm1 1.6660258454717327 hm2 4.596977545094809 hm3 13.2417419128696 hm4 16.03073502576972 hm5 18.408717550573485 hm6 35.96759758641393 hm7 33.260374485869924		hm1 1.6067523107552275 hm2 4.2281796734468715 hm3 11.825451329448118 hm4 13.913602215556933 hm5 27.513793006132488 hm6 17.250064961519556 hm7 26.915064928057518

Tabla 6: Momentos Hu envoltura convexa

Como se puede analizar a partir de los resultados obtenidos, podría ser suficiente para la clasificación tener en cuenta la envoltura convexa y no tener en cuenta los propios momentos de los contornos originales. En el caso de números es bastante cierto, pero en el caso de las letras, no es así. Aunque podríamos darle más peso a estos momentos para resolver este inconveniente.

5.2. DISEÑO DEL CLASIFICADOR

Dentro de los algoritmos que se puede utilizar para la clasificación de muestras, elige el método conocido como k-NN [16], que es un algoritmo basado en instancia de tipo supervisado de Machine Learning.

Es supervisado porque tenemos etiquetado nuestro conjunto de datos de entrenamiento, con la clase o resultado esperado dada "una fila" de datos.

Es basado en instancia porque nuestro algoritmo no aprende explícitamente un modelo (como por ejemplo en Regresión Logística o árboles de decisión). En cambio, memoriza las instancias de entrenamiento que son usadas como "base de conocimiento" para la fase de predicción.

Puede usarse tanto para clasificar nuevas muestras como para predecir. Es un método sencillo, que sirve para clasificar valores buscando los puntos de datos "más similares" (por cercanía en función de su similitud basada en una función de distancia) aprendidos en la etapa de entrenamiento y haciendo conjeturas de nuevos puntos basado en esta clasificación.

Como **pros** tiene sobre todo que es sencillo de aprender e implementar. Tiene como **contras** que utiliza todo el conjunto para entrenar "cada punto" y por eso requiere un uso de mucha memoria y recursos de procesamiento (CPU). Por estas razones k-NN tiende a funcionar mejor en conjunto de datos pequeños y sin una cantidad enorme de atributos.

5.2.1. Descripción del algoritmo K-NN

1. Calcular la distancia (euclídea) entre el contorno a clasificar y el resto de contornos del conjunto de entrenamiento.

2. Seleccionar los “k” elementos más cercanos (con menor distancia, según la función que se use).
3. Realizar una “votación de mayoría” entre los k puntos: los de una clase/etiqueta que dominen, decidirán su clasificación final.

Destacar que K-NN es muy sensible a:

- La **variable *k***, de modo que con valores distintos de *k* podemos obtener resultados también muy distintos. Es interesante elegir valores impares de *k* para desempatar (si las características que utilizamos son pares). No será lo mismo tomar para decidir 3 valores que 13. Esto no quiere decir que necesariamente tomar más puntos implique mejorar la precisión. Lo que es seguro es que cuantos más “puntos k”, más tardará el algoritmo en procesar y darnos respuesta.

El caso de *k=1* se llama Algoritmo del Vecino más cercano.

- La **métrica** de similitud utilizada, puesto que esta influirá, fuertemente, en las relaciones de cercanía que se irán estableciendo en el proceso de construcción del algoritmo. La métrica de distancia puede llegar a contener pesos que nos ayudarán a calibrar el algoritmo de clasificación, convirtiéndola, de hecho, en una métrica personalizada. Las formas más populares de “medir la cercanía” entre puntos son la **distancia Euclidiana**. La normalización de datos también puede ayudar.

$$\boldsymbol{d(x_i, x_j) = \sqrt{\sum_{r=1}^{p}(x_{ri} - x_{rj})^2}} \qquad \textbf{(34)}$$

5.2.2. Elección de atributos y de k

En nuestro caso las clases son el alfabeto completo incluido los números (25 letras + 9 números) y las características a analizar son las siguientes:

- **Compatibilidad:** La variación entre unas y otras es suficiente como para tenerla en cuenta.
- **Redondez**: es muy similar en todas las letras, excepto en casos concretos como es el 1, la J, la L y la T. Esta propiedad se utiliza para descartar entre los *k* candidatos.
- **Compacidad**: también es muy similar entre los diferentes caracteres, por lo que se descarta.
- **6 momentos invariantes normalizados**
- **Compatibilidad del contorno convexo:** valores muy similares entre 14 y 25, razón por la que se descarta.
- **Redondez del contorno convexo**: valores muy similares, todos entre 0.3 y 0.7, que se descarta.
- **Compacidad del contorno convexo:** valores muy similares todos entre 0.5 y 0.7, también se descarte
- **6 momentos invariantes normalizados del contorno convexo**
- **Características propias de las letras** como son el porcentaje en la parte superior, porcentaje en la parte inferior, porcentaje en la parte izquierda y porcentaje en la parte derecha. También, porcentaje en el primer cuadrante, segundo cuadrante, tercer cuadrante y cuarto cuadrante.

Analizando los resultados se opta por utilizar inicialmente los datos correspondientes a los contornos convexos, los momentos invariantes, compatibilidad, redondez y compacidad que son menos susceptibles de pequeñas variaciones de perspectiva o grosor, entre otras, y utilizar los demás atributos para la selección dentro de los *k* elementos más parecidos.

Los pasos realizados son los siguientes:

1. Selección de los *k* (7) cuyos datos del contorno convexo sea similar.
2. Dentro de estos *k* elementos se descartan, en función de los datos del contorno, aquellos cuya compatibilidad sea muy superior o muy inferior y lo mismo con respecto a sus momentos invariantes.
3. Si existen uno o varios contornos interiores se descartan los que no cumplan este criterio.
4. Entre los que quedan se comprueba píxel a píxel, y se selecciona el que mayor número de coincidencias tenga.

Al plantear como objetivo el desarrollo de una aplicación en tiempo real en un teléfono móvil o Tablet en los que la memoria es limitada, los datos de entrenamiento de cada clase son solo los valores de referencia, y no se añaden los nuevos contornos identificados en el grupo de entrenamiento.

Una posible ampliación consistiría en añadir los resultados obtenidos como datos de entrenamiento. Habría que analizar qué cantidad de datos sería viable manejar para un dispositivo móvil.

Por eso, a la hora de seleccionar dentro de los *k*, dado que los valores que tenemos para comparar son muy pocos, es necesario añadir alguna otra característica a comparar.

Veamos un caso de letras parecidas que nos pueden dar lugar a equivocación, tabla 7. Por ejemplo, entre la B y D, hay que tener en cuenta que lo que se está comparando es el contorno exterior.

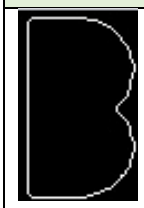
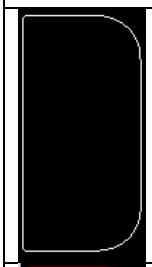
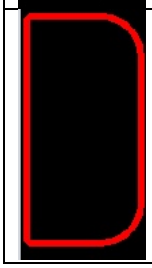
Imagen	Compatibilidad/Redondez/ Compacidad	Momentos invariantes
	compatibilidad 17.228448136703793 redondez 0.5194080904223677 compacidad 0.7206997227849943	1.6500705885565168 4.6598956452500255 9.109855778160096 12.072785130598817 NaN NaN NaN
	compatibilidad convexo 15.212929027403346 redondez convexo 0.5379982153480072 compacidad convexo 0.7334836162778328	1.6841519015249913 4.874277926104495 8.575600731321243 11.350570500433449 NaN NaN 25.884686639244556
	compatibilidad 16.90460158082713 redondez 0.4789507882767347 compacidad 0.692062705451417	1.5848813793893464 4.202665254579443 9.771250075116388 12.2586050371967 NaN NaN 27.507590503069633
	compatibilidad convexo 16.498874440309777 redondez convexo 0.4798399507540782 compacidad convexo 0.6927048078034959	1.5844182510623668 4.201174839928258 9.86103724432525 12.343103292549673 NaN NaN 28.402942572565617

Tabla 7: Comparación atributos

Se ve que los dos valores de referencia son muy similares y el algoritmo puede elegir cualquiera de los dos. En estos casos es muy interesante que la votación sea en base a otro parámetro como puede ser el número de contornos interiores. También es interesante esta opción para separar los resultados del 6 y de la G.

Los resultados en general son bastante aceptables, aunque en determinadas imágenes con picos en los contornos no los detecta correctamente. Por esta razón es conveniente entrenar, al menos mínimamente, el algoritmo para que los resultados sean mucho más correctos.

En tiempo de ejecución, dado que no se dispone un número elevado de valores con los que comparar no existe problema. Por esta razón, se prueban y analizan otros métodos para comparar los resultados, concretamente el clasificador de Haar y las Redes neuronales.

5.3. CLASIFICADOR HAAR

Los clasificadores Haar Cascade son una forma eficaz de detección de objetos. Este método fue propuesto por Paul Viola y Michael Jones en sus artículos [17] y [18] en el que se utilizan muchas imágenes positivas y negativas para entrenar al clasificador.

Se caracteriza por la introducción de una nueva representación de imagen, llamada Imagen integral que permite que las características utilizadas por el detector se calculen más rápidamente que en otras representaciones. Las características de Haar se muestran en la figura 58.

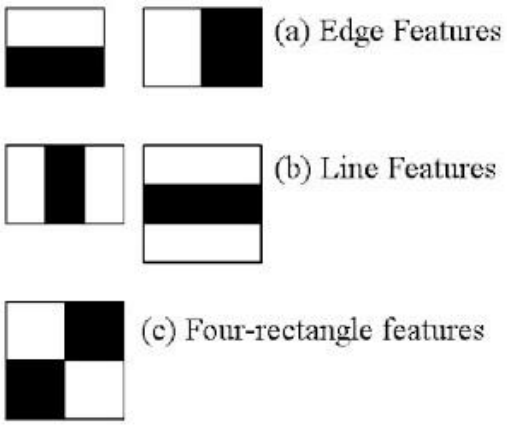


Figura 58: Características Haar

Estas características, se pueden asimilar al *kernel* convolucional. Cada característica es un valor único obtenido al restar la suma de píxeles debajo del rectángulo blanco de la suma de píxeles debajo del rectángulo negro. Todos los tamaños y ubicaciones posibles de cada núcleo se utilizan para calcular un elevado número de características, por ejemplo, una ventana de 24x24 da como resultado más de 160000

características. Para cada cálculo de características, se necesita encontrar la suma de los píxeles debajo de los rectángulos blanco y negro. Para solucionar esto, se introduce el concepto de imagen integral. Por muy grande que sea su imagen, reduce los cálculos para un píxel dado a una operación que involucra solo cuatro píxeles.

Pero entre todas estas características que se calculan, la mayoría de ellas son irrelevantes. Por ejemplo, considerando la imagen de la figura 59, se desea detectar la región de los ojos.

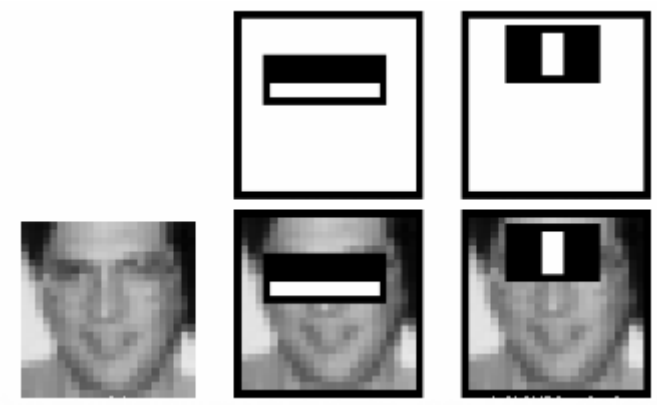


Figura 59: Características Haar para regiones de ojos

La fila superior muestra dos buenas características para encontrar los ojos. La primera característica seleccionada parece centrarse en la propiedad de que la región de los ojos suele ser más oscura que la región de la nariz y las mejillas. La segunda característica seleccionada se basa en la propiedad de que los ojos son más oscuros que el puente de la nariz. En este caso, las mismas ventanas aplicadas a las mejillas o cualquier otro lugar son irrelevantes.

El seleccionar las mejores funciones dentro de todas las posibles características lo realiza el algoritmo de aprendizaje, basado en AdaBoost, que selecciona una pequeña cantidad de características visuales críticas de un conjunto más grande y produce clasificadores extremadamente eficientes [19].

El algoritmo, aplica todas y cada una de las funciones en todas las imágenes de entrenamiento. Para cada característica, encuentra el mejor umbral que clasificará los objetos en positivos y negativos. Obviamente, habrá errores o clasificaciones erróneas. Se seleccionan las características con una tasa de error mínima, lo que significa que son las características que clasifican con mayor precisión las imágenes. A cada imagen se le da el mismo peso inicialmente. Después de cada clasificación, se aumentan los pesos de las imágenes mal clasificadas. Luego se realiza el mismo proceso, luego se calculan nuevas tasas de error y también nuevos pesos. El proceso continúa hasta que se logra la precisión requerida o la tasa de error deseable o bien se encuentra el número requerido de características.

El clasificador final da como resultado una suma ponderada de estos clasificadores débiles. Un clasificador se llama débil porque por sí solo no puede clasificar la imagen, pero junto con otras formas se convierte en un clasificador fuerte. El articulo comentado anteriormente [19] asegura que incluso 200 funciones brindan detección con una precisión del 95%. Su configuración final tendrá alrededor de 6000 características. Con esto se consigue reducir de más de 160 000 funciones a 6000 funciones para una ventana 24x24 píxeles.

También el clasificador Haar se caracteriza por el método para combinar clasificadores cada vez más complejos en una cascada que permite que las regiones de fondo de la imagen se descarten rápidamente mientras se necesita más computación en regiones similares a objetos prometedores.

En lugar de aplicar las 6000 funciones en una ventana, las funciones se agrupan en diferentes etapas de clasificadores y se aplican una por una. Normalmente, las primeras etapas contendrán muchas menos funciones. Si una ventana falla en la primera etapa, se desecha y no se considera las características restantes. Si pasa, se aplica la segunda etapa de características y continúa el proceso. La ventana que pasa

por todas las etapas es una región frontal. Según los autores, en su artículo [19], en promedio se evalúan 10 características de más de 6000 en cada sub-ventana.

La cascada se puede ver como un mecanismo específico de foco de atención que, a diferencia de los enfoques anteriores, proporciona garantías estadísticas de la poca probabilidad que las regiones descartadas contengan el objeto de interés. En el dominio de la detección de rostros, el sistema produce índices de detección comparables a los mejores sistemas descritos anteriormente.

OpenCV ofrece este tipo de clasificadores dentro de su implementación. Es compatible tanto con la utilización de características wavelet HAAR como características LBP (*Local Binary Patterns,* patrones binarios locales). Las características LBP producen una precisión de números enteros en contraste con las características de HAAR, cuya precisión es de coma flotante, por lo que tanto el entrenamiento como la detección con LBP son varias veces más rápidas que el realizado con características HAAR. En cuanto a la calidad de detección entre LBP y HAAR, depende principalmente de los datos de entrenamiento utilizados y los parámetros de entrenamiento seleccionados. Es posible entrenar un clasificador basado en LBP que proporcione casi la misma calidad que uno basado en HAAR, dentro de un porcentaje del tiempo de entrenamiento empleado.

Trabajar con un clasificador de cascada Haar incluye dos etapas principales: la etapa de entrenamiento y la etapa de detección.

5.3.1. Preparación de los datos de entrenamiento

Para entrenar este tipo de clasificadores, se necesita un conjunto de muestras positivas (que contengan objetos reales que desea detectar) y un conjunto de imágenes negativas (que contengan todo lo que no se desea detectar). El conjunto de muestras negativas debe prepararse manualmente, mientras que el conjunto de muestras positivas se crea utilizando la aplicación *opencv_createsamples*.

Las muestras negativas se toman de imágenes arbitrarias que no contienen los objetos que se desea detectar. Las imágenes descritas pueden ser de diferentes tamaños. Sin embargo, cada imagen debe ser igual o mayor que el tamaño deseado de la ventana de entrenamiento *(que corresponde a las dimensiones del modelo, siendo la mayoría de las veces el tamaño promedio de su objeto)*, porque estas imágenes se utilizan para submuestrear una imagen negativa dada en varias imágenes. El conjunto de muestras negativas se usará para indicar el paso de aprendizaje automático, impulsando en este caso, lo que no debe buscar, cuando intente encontrar sus objetos de interés.

Las muestras positivas son creadas por la aplicación *opencv_createsamples*. Con esta herramienta se recortan, se cambia el tamaño y se pone en formato binario necesario de OpenCV las imágenes que corresponden a las matrículas o a los dígitos a buscar. Se debe especificar además el ancho y el alto de las muestras de salida.

5.3.2. Entrenamiento

Para el entrenamiento se utiliza la herramienta de OpenCV llamada *opencv_traincascade.*

Los parámetros básicos que se deben configurar son, además del número de muestras positivas y negativas y el tamaño de las muestras de salida, el número de etapas, tamaño del *buffer* para el uso de la memoria RAM, número de hilos de computación, y la ratio con la cual se determina la precisión con la que el modelo debe seguir aprendiendo y cuándo detenerse. Una buena pauta es entrenar con ratio como máximo de 10e-5, para garantizar que el modelo no se sobreajuste con sus datos de entrenamiento.

Como parámetros para la cascada se utiliza los tipos de etapa, el ancho y alto de las muestras de entrenamiento que debe ser igual al utilizado en la preparación de las

muestras de entrenamiento, y el tipo de característica, si elegimos HAAR o LPB. Y como parámetros del clasificador se debe indicar:

- Tipo de clasificador:
  - **DAB - AdaBoost discreto**: se adapta basándose en los errores de las etiquetas de clase predicha
  - **RAB - AdaBoost real:** utiliza las probabilidades de clase predichas.
  - **LB – LogitBoost:** representa una aplicación de técnicas de regresión logística establecidas al método AdaBoost.
  - **GAB - AdaBoost suave**: presenta un tamaño de paso limitado, que se elige para minimizar una función cuadrática y no se aplica ningún otro coeficiente.

Las explicaciones detalladas sobre dichos algoritmos se describen en la referencia [20]. En la figura 60 se muestra una comparación a nivel de errores para una clasificación binaria donde la función *Y* a obtener es una función conocida no lineal de 10 características de entrada.

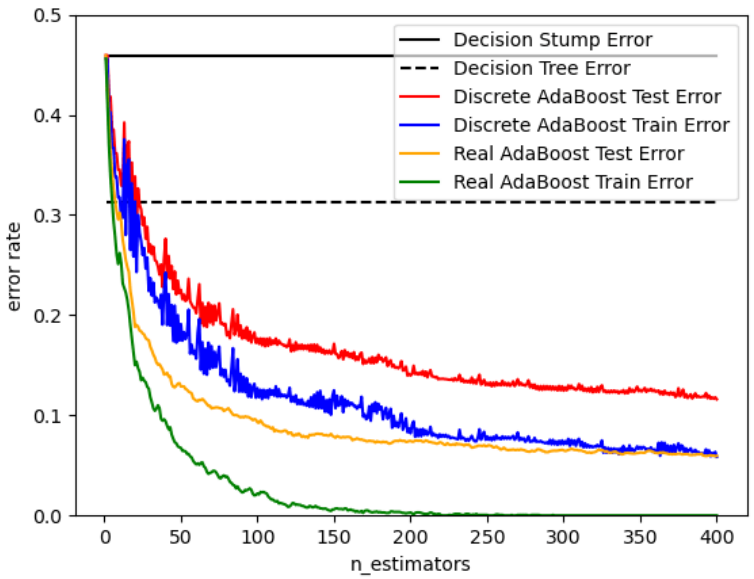


Figura 60: Comparación entre distintos algoritmos AdaBoot

El método requiere la definición de los siguientes parámetros,

- **Tasa de acierto mínimo (minHitRate)**: la tasa de acierto mínimo deseada para cada etapa del clasificador. La tasa general de acierto se puede estimar como: $Tasa\ general\ aciertos = tasa\ aciertos\ minima * num\_etapas$

(35)

- **Tasa máxima de falsas alarmas (maxFalseAlarmRate):** Tasa máxima deseada de falsas alarmas para cada etapa del clasificador. La tasa general de falsas alarmas se puede estimar como:

$$\boldsymbol{T}asa\ general\ falsas\ alarmas = tasa\ maxima\ falsas\ alarmas * num\_etapas$$

(36)

- **Ratio de recorte (weightTrimRate)**: Especifica si se debe utilizar recorte y su peso. Una opción aceptable es 0,95.
- **Profundidad máxima de los árboles débiles (maxDepth)**: Profundidad máxima del árbol débil. Una opción decente es 1, ese es el caso de los tocones.
- **maxWeakCount <max_weak_tree_count>:** Recuento máximo de árboles débiles para cada etapa de la cascada. El clasificador potenciado (etapa) tendrá tantos árboles débiles *(<=maxWeakCount),* como sea necesario para lograr el objetivo dado (*maxFalseAlarmRate*).

Como parámetros sobre las características tenemos el modo (*mode*) que selecciona el tipo de conjunto de características de Haar utilizado en el entrenamiento. Dentro de las opciones *<BASIC (default) | CORE | ALL>:* BÁSICO usa solo funciones verticales, mientras que TODO usa el conjunto completo de funciones verticales y giradas 45 grados.

5.3.3. Configuración y pruebas del clasificador para matrículas

Se realizan dos enfoques o tipos de clasificador. Un clasificador que detecte matrículas y otros clasificadores que detecten números y letras dentro del rectángulo detectado como matrícula.

Inicialmente se realiza un primer entrenamiento con 500 imágenes de matrículas sin preocuparse del tamaño de dichas matrículas y poniendo como tamaño del clasificador anchura 70 y anchura 15. Los resultados obtenidos no son del todo satisfactorios. En la imagen de la figura 61 se observa cómo reconoce solo 2 de las 4 matrículas existentes.

Figura 61: Resultados clasificador tamaño 70x15

Analizando artículos relacionados [21], se identifica el hecho de que dan gran importancia al tamaño de las imágenes positivas para el entrenamiento y posterior clasificador. Por lo que se estudian los tamaños de las distintos tipos de matrícula y las posiciones más comunes. En las matrículas ordinarias frontales se distinguen dos tipos de tamaño proporcionales uno a anchura 52 y altura 11 y otro a anchura 80 y altura 20, figuras 62 y 63.

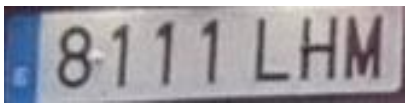


Figura 62: Formato 52x11

Figura 63: Formato 80x20

En imágenes no frontales, se encuentran gran cantidad de posiciones con orientación izquierda o derecha. De esta forma se identifican 3 grupos con las relaciones de anchura y altura, que se muestra en la figura 64.

Izquierda formato 30x20

Izquierda formato 50x20

Izquierda formato 70x20

Derecha formato 30x20

Derecha formato 50x20

Derecha formato 70x20

Figura 64: Imágenes no frontales

Para matrículas de moto y ordinarias altas, figuras 65, 66 y 67.

Figura 67: formato 18x16

Figura 66: Formato derecha 18x16

Figura 65: Izquierda Formato 22x16

Para crear la base de datos de imágenes positivas se debe generar como mínimo un conjunto de 300 imágenes para cada tipo, con el fin de conseguir una correcta clasificación. Dado el trabajo tan laborioso de sacar las fotografías e ir recontando las zonas correspondientes y la falta de tiempo, dado que no existen, hasta donde se ha podido llegar, bases de datos gratuitas con características similares a las utilizadas en

este proyecto se decide trabajar sobre imágenes frontales, aunque se almacenan las demás imágenes seleccionadas para posteriores ampliaciones.

Las imágenes del conjunto de datos incluyen diferentes tamaños, aunque siempre proporcional al tamaño del clasificador, diferente iluminación y enfoque.

Para las imágenes negativas se utilizan 500 imágenes, las mismas para todos los clasificadores de matrículas. Según los artículos analizados [22], los resultados son mejores si incluyen elementos que nos podemos encontrar en un fotograma y no con elementos muy dispares, figura 68.

IMÁGENES NEGATIVAS

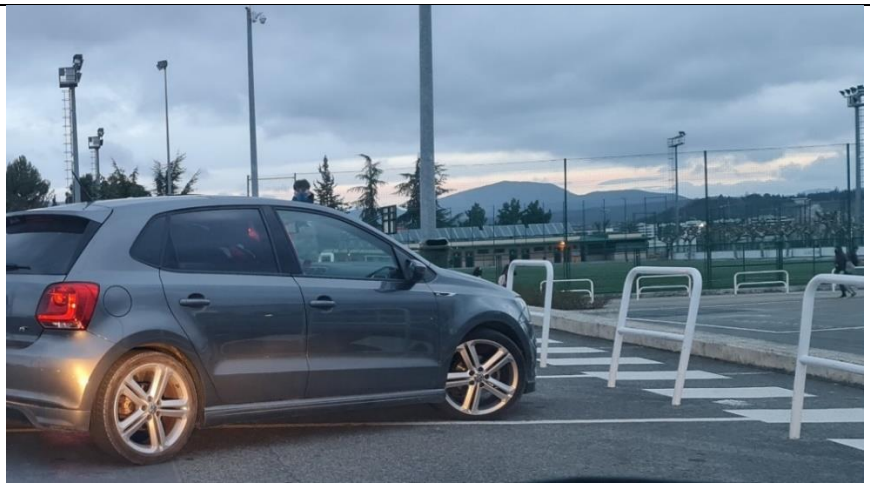



Figura 68: Imágenes negativas

La configuración utilizada es la siguiente, figura 69:

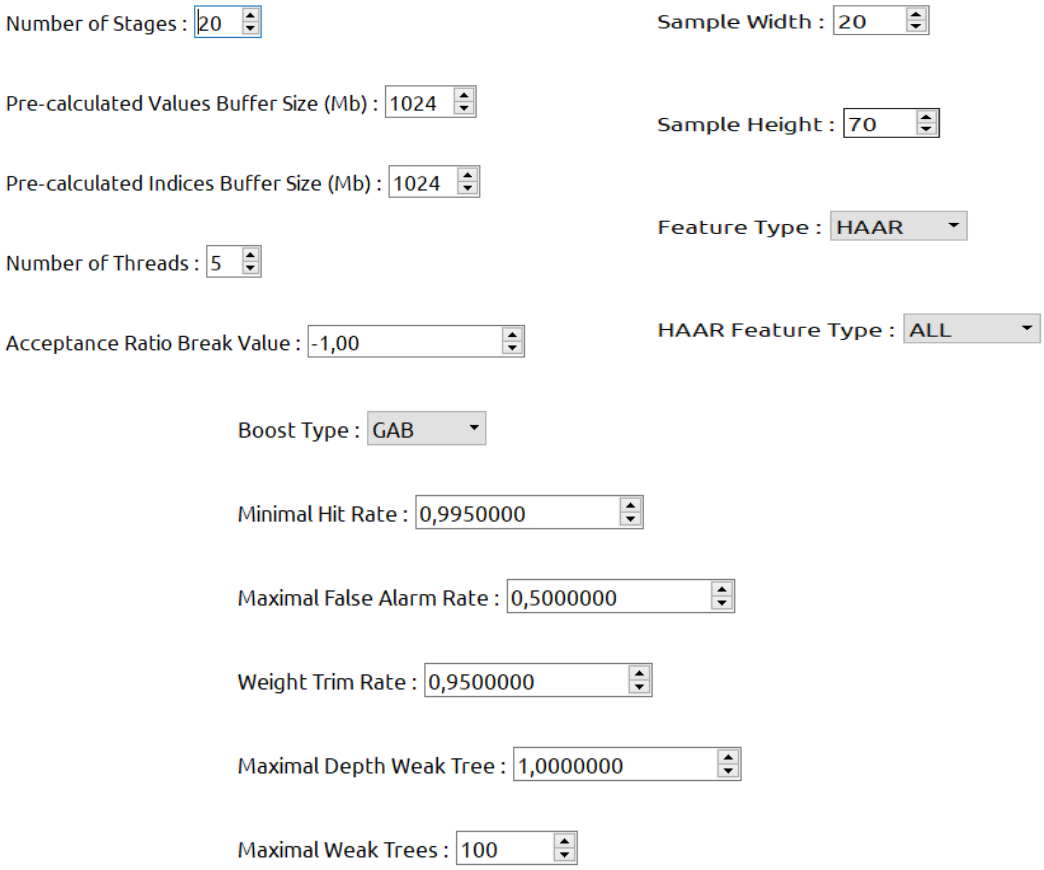


Figura 69: Ajustes del clasificador Haar

Se realiza entrenamiento con métodos HAAR y LBP, modificando el tipo de características HAAR básicas a todas y también los algoritmos Boost. Cada entrenamiento con el ordenador mencionado en el apartado 1.8, requiere de alrededor de 80 minutos, en la figura 70 se analizan los tiempos de entrenamiento con distintas configuraciones para el clasificador de imágenes frontales 52x11. Destacar el margen de error en el análisis de tiempo dado que el sistema no especifica tiempo de entrenamiento.

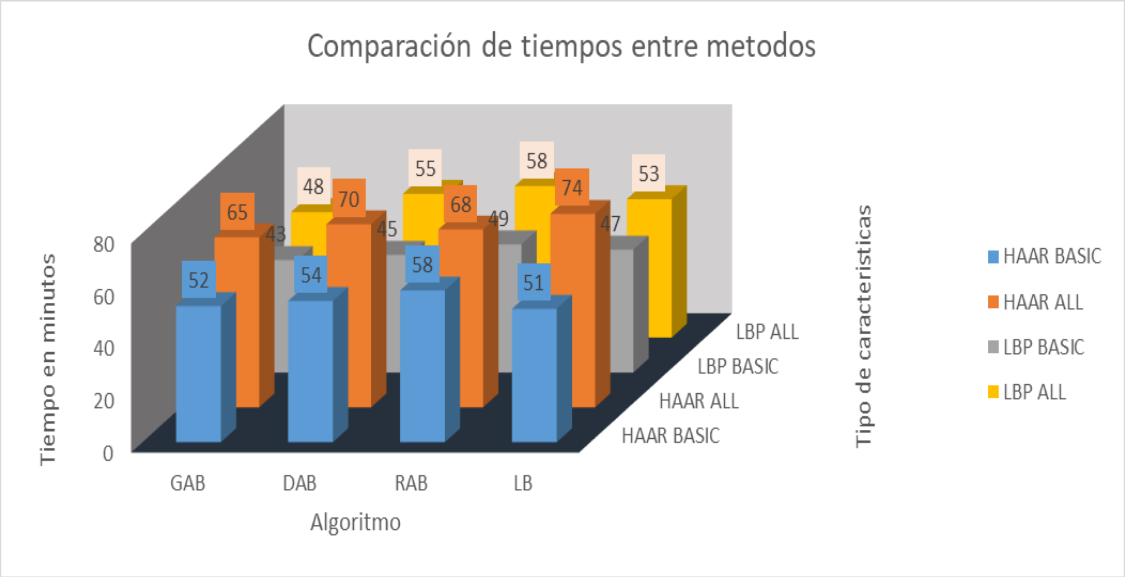


Figura 70: Comparativa de tiempos para distintas configuraciones del Clasificador Haar

Sobre 20 imágenes en las que las matrículas se encuentran en diferentes escalas la proporción de aciertos encontrados se pueden ver en la figura 71. Se distingue una pequeña diferencia entre el método Haar y el método LBP, mientras que no se distingue una diferencia significativa entre los distintos algoritmos.

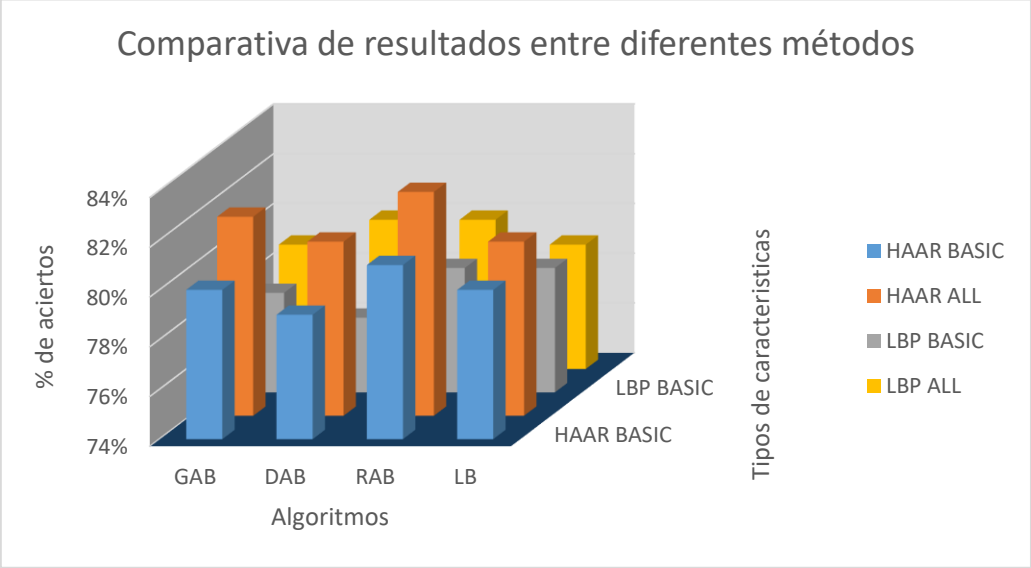


Imagen 71: Comparativa de aciertos para distintas configuraciones del Clasificador Haar

5.3.4. Configuración y pruebas del clasificador para los dígitos y letras

Para las letras de las matrículas se realiza el mismo proceso para cada uno de los números y de las letras que pueden aparecer dentro de una matrícula, tabla 8.

Tamaño de las letras y números								
	Anchura	3		Anchura	5		Anchura	3
	Altura	6		Altura	8		Altura	8

Tabla 8: Tamaño de letras y números

Después se crea la base de datos para los dos clasificadores de cada número o letra frontales y se deja el formato 3x8, que es el encontrado en imágenes muy laterales para una posterior ampliación. En las imágenes positivas se genera un conjunto de 200 imágenes para cada tipo. Las imágenes incluyen diferentes tamaños, aunque siempre proporcional al tamaño del clasificador, diferente iluminación y enfoque.

Para las imágenes negativas se utilizan 700 imágenes, corresponden a imágenes de matrículas que no contienen el número o letra a buscar. En la tabla 9 se muestran varias imágenes de matrículas que no contienen el número 0.

EJEMPLO DE IMÁGENES NEGATIVAS PARA EL CLASIFICADOR DEL NÚMERO 0	
	
	

Tabla 9: Imágenes negativas para el clasificador del número 0

La configuración es similar a la realizada en el apartado anterior, exceptuando el tamaño de las imágenes.

Realizo pruebas similares a la configuración. Destacar el hecho de que el tiempo de entrenamiento es superior al entrenamiento con la matricula completa, aun cuando las imágenes son de menor dimensión, dado que corresponden a zonas concretas y las imágenes negativas también son de dimensión inferior.

En la figura 72 se analizan los tiempos de entrenamiento para la letra B, con la misma configuración comentada anteriormente.

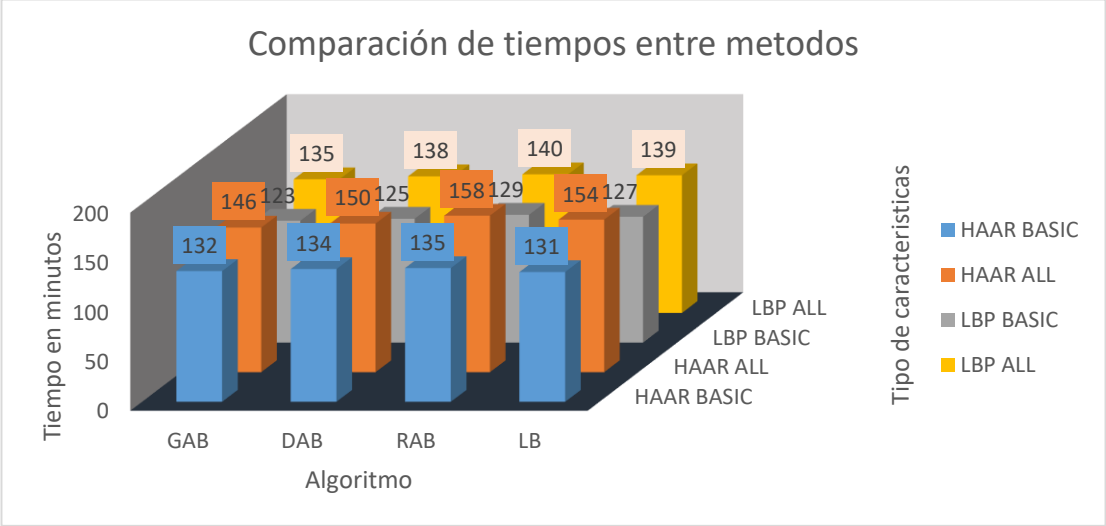


Figura 72: Comparativa de tiempo entre métodos para el entrenamiento de la letra B

En la figura 73 se compara la tasa de aciertos de los diferentes métodos.

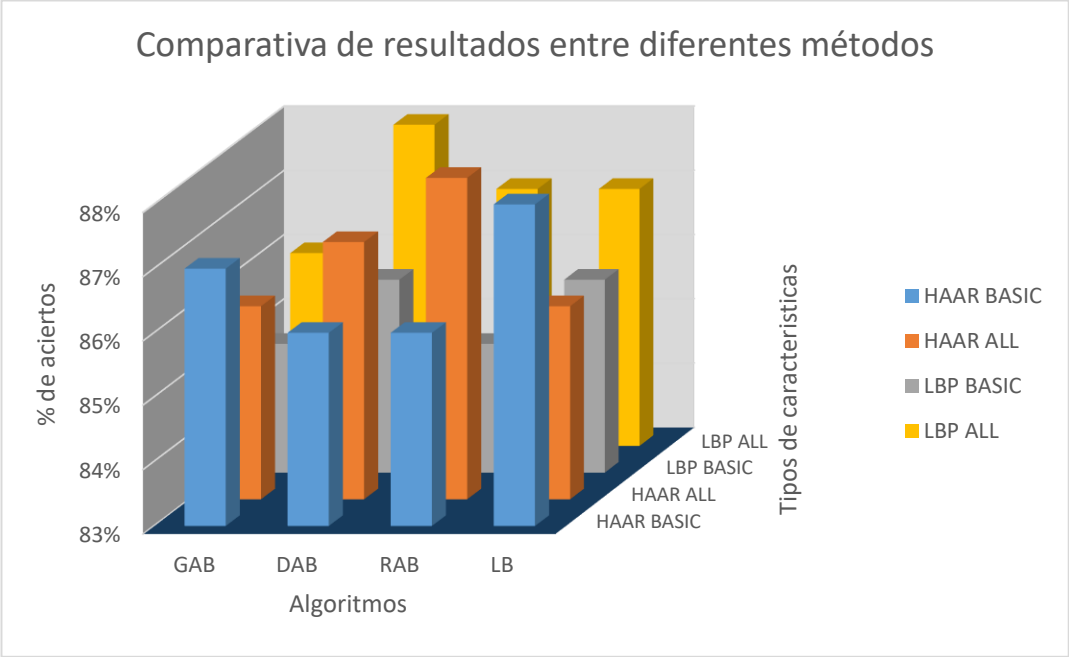


Figura 73: Comparativa de aciertos para distintas configuraciones del Clasificador Haar para la letra B.

5.4. REDES NEURONALES

Las Redes Neuronales Artificiales (Artificial Neural Network, ANN/NN) son un enfoque conexionista de la Inteligencia Artificial (IA), vagamente inspirado en sus homólogos biológicos.

La información de entrada atraviesa la ANN (donde se somete a diversas operaciones) produciendo unos valores de salida.

La ANN es una estructura de grafo dirigida donde los nodos son Neuronas Artificiales que consiste en un conjunto de unidades llamadas neuronas artificiales conectadas entre sí para transmitir señales por medio de enlaces. La salida de la neurona la determina una función de activación, que modifica el valor resultado o impone un límite que se debe sobrepasar para propagar el valor a otra neurona. Los enlaces multiplican el valor de salida de la neurona anterior por un valor de peso. Estos pesos pueden incrementar o inhibir el estado de activación de las neuronas.

El modelo más simple es el Perceptrón [23] y uno de los modelos más conocidos son los perceptrones multicapa, cuya estructura se puede apreciar en la figura 74. Se implementa una capa de entrada, varias capas ocultas (*hidden layers*) que van procesando características (*features*) sobre las que operan otras capas y una capa de salida.

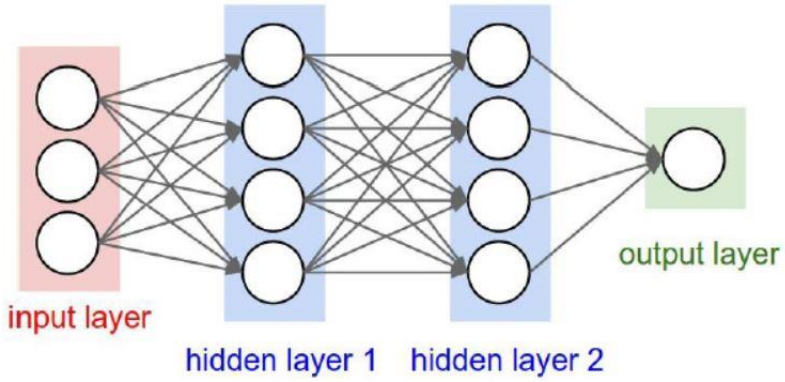


Figura 74: Estructura red neuronal multicapa

Una ANN aprende utilizando distintas técnicas de Aprendizaje supervisado, no supervisado o por refuerzo. Para el aprendizaje supervisado necesitamos saber la clasificación y el resultado a obtener, como es este caso. Normalmente, en su aprendizaje se intenta minimizar una función de pérdida (*loss*) que evalúa la red en su totalidad. El aprendizaje actualiza los valores de los pesos de los enlaces buscando reducir el valor de la función de pérdida.

Las redes neuronales se pueden clasificar en base a su estructura:

- **Monocapa**: la red neuronal está formada por una única capa de neuronas que recibe valores del exterior de la red, las procesa y genera unos valores de salida que entrega como resultado. Las neuronas de la capa no se interconectan entre ellas.
- **Multicapa:** es el tipo de red neuronal más extendido. Consta de una capa de entrada que recibe valores del exterior, una serie de capas intermedias u ocultas que van procesando la información y una capa de salida que entrega los resultados de la red neuronal. Las neuronas de una capa se interconectan con todas las neuronas de la capa siguiente, pero no entre ellas.
- **Convolucional:** son similares a las redes neuronales multicapa, pero la diferencia radica en que, en estas, las neuronas de cada capa no se interconectan con todas las de la capa siguiente, sino con un subconjunto de estas, y no se interconectan entre las neuronas de la misma capa. En este caso se habla de especialización de las neuronas (grupos) y se reduce la cantidad de unidades necesarias y la complejidad de los sistemas que utilizan este tipo de redes.
- **Concurrentes:** en estas redes las neuronas no se organizan en capas, sino que las neuronas están interconectadas entre ellas de manera no estructurada en capas. Esto permite que estas redes tengan memoria, es decir, que la

información generada en iteraciones anteriores afecte al resultado del procesamiento en un tiempo futuro.

Explicaciones más detalladas sobre redes neuronales se encuentran en el libro de la referencia [24].

5.4.1. Elección de la red neuronal

Hoy en día existen varias plataformas y librerías que realizan Deep Learning. Muy popular es TensorFlow y Keras de Google [25], que se puede ejecutar distribuido, está disponible para distintas plataformas y soporta otras técnicas de aprendizaje máquina. Existe una versión para Android llamada TensorFlow Lite que nos permite integrar esta plataforma en nuestro dispositivo.

Otra plataforma muy interesante para trabajar directamente sobre Java es Neuroph [26] que incluye Neuroph Studio Gui, que es un IDE de red neuronal Java basado en la plataforma NetBeans, libre y muy visual.

Se prueba esta última plataforma con un conjunto de entrenamiento de 6500 imágenes de tamaño 15x24 píxeles, no siendo un número excesivamente grande el sistema se queda bloqueado y no se han obtenido resultados de ningún tipo. Por lo que, se elige el Framework TensorFlow y Keras.

En el anexo 3 se presenta el código completo utilizado, y en el siguiente apartado se especifican los aspectos más relevantes de dicho código.

5.4.2. Configuración y pruebas de la red neuronal

Se cargan 6456 imágenes divididas en 31 categorías correspondientes a las letras y números existentes en una matrícula española, incluyendo la zona azul.

Las imágenes son de tamaño 15x24 píxeles y el tamaño del lote utilizado es de 32. Se dividen las imágenes en un grupo de entrenamiento correspondiente a un 80% y un

grupo de validación correspondiente a un 20% del total de imágenes, ya que es una proporción ampliamente utilizada [14].

Los valores de intensidad de los canales RGB se encuentran comprendidos entre [0, 255]. Esto no es ideal para una red neuronal, por lo que se estandariza (normalización) a valores entre [0, 1] usando *tf.keras.layers.Rescaling*.

Se entrena un modelo secuencial con distintas configuraciones, en la referencia [27] se pueden analizar las distintas capas que se pueden añadir al modelo propuesto.

- **Modelo 1:** Capa de entrada *Flatten*. Capa oculta conectada completamente con 128 neuronas activadas mediante una función de activación *ReLU*. Y capa de salida densa con 31 clases y activación *softmax*, figura 75.

```
[ ]  num_classes = len(class_names)
     model = tf.keras.Sequential([
       tf.keras.layers.Rescaling(1./255),
       tf.keras.layers.Flatten(input_shape=(15, 24)),
       tf.keras.layers.Dense(128, activation='relu'),
       tf.keras.layers.Dense(num_classes, activation='softmax')
     ])
```

Figura 75: Modelo secuencial básico de red neuronal.

Resultados con distintas fases, figura 76,77 y 78. Distintas ejecuciones dan resultados ligeramente diferentes.

```
Epoch 19/20
86/86 [==============================] - 0s 4ms/step - loss: 0.2331 - accuracy: 0.9412 - val_loss: 0.2535 - val_accuracy: 0.9346
Epoch 20/20
86/86 [==============================] - 0s 4ms/step - loss: 0.2260 - accuracy: 0.9431 - val_loss: 0.2486 - val_accuracy: 0.9376
```

Figura 76: Resultados modelo secuencial básico con 20 fases de entrenamiento

```
Epoch 39/40
86/86 [==============================] - 0s 4ms/step - loss: 0.1147 - accuracy: 0.9696 - val_loss: 0.2040 - val_accuracy: 0.9467
Epoch 40/40
86/86 [==============================] - 0s 4ms/step - loss: 0.1137 - accuracy: 0.9702 - val_loss: 0.2030 - val_accuracy: 0.9467
```

Figura 77: Resultados modelo secuencial básico con 40 fases de entrenamiento

```
Epoch 59/60
86/86 [==============================] - 0s 4ms/step - loss: 0.0584 - accuracy: 0.9846 - val_loss: 0.2474 - val_accuracy: 0.9414
Epoch 60/60
86/86 [==============================] - 0s 4ms/step - loss: 0.0587 - accuracy: 0.9848 - val_loss: 0.2431 - val_accuracy: 0.9414
```

Figura 78: Resultados modelo secuencial básico con 60 fases de entrenamiento

- **Modelo 2**: 3 capas convolucionales (*tf.keras.layers.Conv2D*) con una capa *Max pooling* (*tf.keras.layers.MaxPooling2D*) entre cada una de ellas y una capa densa totalmente conectada de 128 neuronas y con activación '*ReLU*'. Capa de salida densa de 31 clases**,** figura 79**.**

```
num_classes = len(class_names)

model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

Figura 79: Modelo de red neuronal convolucional

Los resultados obtenidos en distintas fases se muestran en las figuras 80, 81 y 82. Conviene recalcar que distintas ejecuciones dan resultados ligeramente diferentes.

```
Epoch 9/10
167/167 [==============================] - 1s 4ms/step - loss: 0.1056 - accuracy: 0.9688 - val_loss: 0.2135 - val_accuracy: 0.9339
Epoch 10/10
167/167 [==============================] - 1s 4ms/step - loss: 0.0859 - accuracy: 0.9762 - val_loss: 0.1916 - val_accuracy: 0.9452
```

Figura 80: Modelo convolucional con 10 fases de entrenamiento

```
Epoch 19/20
167/167 [==============================] - 1s 4ms/step - loss: 0.0191 - accuracy: 0.9942 - val_loss: 0.1344 - val_accuracy: 0.9699
Epoch 20/20
167/167 [==============================] - 1s 4ms/step - loss: 0.0072 - accuracy: 0.9989 - val_loss: 0.1309 - val_accuracy: 0.9722
```

Figura 81: Modelo convolucional con 20 fases de entrenamiento

```
Epoch 39/40
167/167 [==============================] - 1s 4ms/step - loss: 1.1404e-05 - accuracy: 1.0000 - val_loss: 0.1957 - val_accuracy: 0.9752
Epoch 40/40
167/167 [==============================] - 1s 4ms/step - loss: 1.0728e-05 - accuracy: 1.0000 - val_loss: 0.1996 - val_accuracy: 0.9760
```

Figura 82: Modelo convolucional con 40 fases de entrenamiento

A continuación, se analizan las pérdidas y la exactitud para el modelo convolucional con 20 fases de entrenamiento, figura 83. La exactitud se encuentra entre los valores 0,97 y 0,98 en el conjunto de validación, siendo la diferencia entre el entrenamiento

y validación aceptable, pero las pérdidas se encuentran en valores de 0.125 aproximadamente. Aun siendo estos valores aceptables, se propone como mejora el ajuste de la red neuronal para la obtención de pérdidas inferiores.

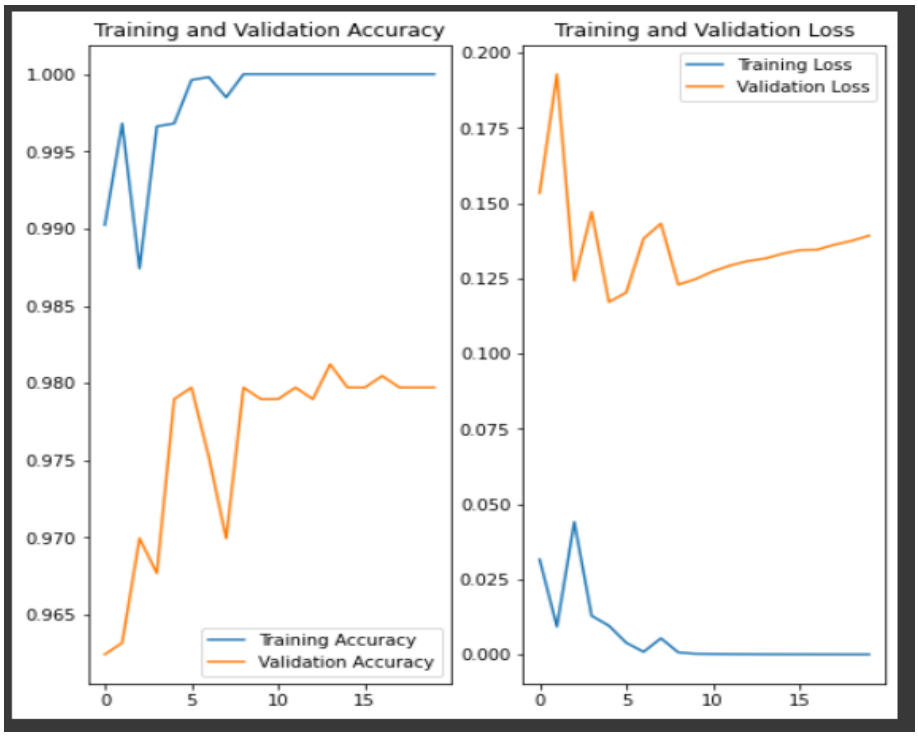


Figura 83: Análisis de exactitud y perdidas de la red neuronal convolucional.

El siguiente paso es la exportación a formato TFLite. En la figura 84, se analiza el proceso a alto nivel de exportación a dicho formato. Y en la figura 85 se muestra el código necesario para ello.

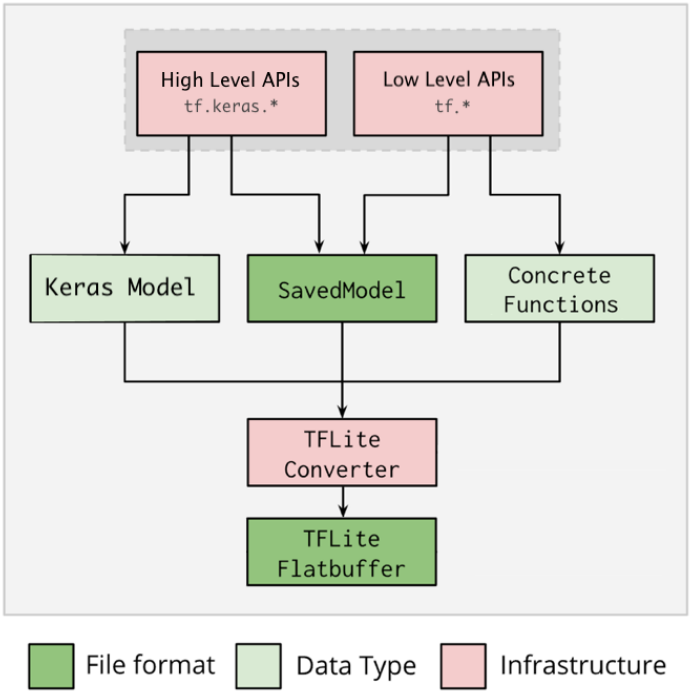


Figura 84: Proceso de exportación a formato TfLite para Android.

Guardar el modelo

```
# Convertir el modelo
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Salvarlo en disco
with open('model.tflite', 'wb') as f:
  f.write(tflite_model)
```

Figura 85: Código para guardado de modelo en formato Tflite

Por último, se debe importar el modelo en nuestro proyecto y realizar la predicción. En el siguiente código, figura 86, se analiza la predicción, codificada directamente en Phyton.

```python
#Subimos una imagen escalada en formato
imagen_letra= 'letra_desconocida.jpg'

img = tf.keras.utils.load_img(
    imagen_letra, target_size=(img_height, img_width)
)
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "Esta imagen pertenece a la clase {} con un porcentaje de confianza de un {:.2f}."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

Figura 86: Código de predicción.

Como ejemplo ilustrativo, en la tabla 10 se pueden comprobar los resultados obtenidos para dos matrículas con diferentes tamaños y orientación.

Predicción	
	Esta imagen pertenece a la clase 15x24num5 con un porcentaje de confianza de un 100.00.
	Esta imagen pertenece a la clase 15x24num1 con un porcentaje de confianza de un 100.00.
	Esta imagen pertenece a la clase 15x24num3 con un porcentaje de confianza de un 100.00.
	Esta imagen pertenece a la clase 15x24num2 con un porcentaje de confianza de un 100.00.
	Esta imagen pertenece a la clase 15x24letraG con un porcentaje de confianza de un 100.00.
	Esta imagen pertenece a la clase 15x24letraW con un porcentaje de confianza de un 100.00.
	Esta imagen pertenece a la clase 15x24letraK con un porcentaje de confianza de un 100.00.
Predicción	
	Esta imagen pertenece a la clase 15x24num3 con un porcentaje de confianza de un 100.00.

	Esta imagen pertenece a la clase 15x24num2 con un porcentaje de confianza de un 100.00.
	Esta imagen pertenece a la clase 15x24num5 con un porcentaje de confianza de un 100.00.
	Esta imagen pertenece a la clase 15x24num0 con un porcentaje de confianza de un 100.00.
	Esta imagen pertenece a la clase 15x24letraF con un porcentaje de confianza de un 100.00.
	Esta imagen pertenece a la clase 15x24letraF con un porcentaje de confianza de un 100.00.
	Esta imagen pertenece a la clase 15x24letraB con un porcentaje de confianza de un 100.00.

Tabla 10: Resultados predicción redes neuronales

Los resultados obtenidos en la predicción han sido muy satisfactorios, e incluso no ha sido necesario escalar la imagen. En las 20 imágenes utilizadas para la validación, el único resultado incorrecto obtenido se puede ver en la tabla 11.

	Esta imagen pertenece a la clase 15x24letraK con un porcentaje de confianza de un 81.41.

Tabla 11: Resultado incorrecto predicción

5.5. OTROS MÉTODOS

En distintas referencias analizadas [28], y productos del mercado [29], lo mas común es la utilización de un OCR, o la comparación (MATCH) con imágenes de muestra.

5.5.1. OCR

El reconocimiento óptico de caracteres, es un proceso dirigido a la digitalización de textos, los cuales se identifican automáticamente a partir de una imagen de símbolos o caracteres que pertenecen a un determinado alfabeto, para luego almacenarlos en forma de datos procesados.

El proceso consta de cuatro etapas:

1. Binarización o caracterización.
2. Fragmentación o segmentación de la imagen.
3. Adelgazamiento de los componentes.
4. Comparación con patrones.

Realmente, ya se han realizado los tres primeros pasos en los procesos anteriores. Existen varios OCR que trabajan sobre JAVA como es el caso de la librería JavaANPR [30] que además es específica para la detección de matrículas y Tess4J [31] que es una librería Java open-source con licencia Apache, que realiza la detección mediante inteligencia artificial, y redes neuronales, y actúa como Wrapper JNA para la librería OCR open-source Tesseract [32].

El coste computacional de un OCR realmente es muy alto en una aplicación en tiempo real y los resultados no siempre son satisfactorios como a primera vista puede parecer. Vemos varios ejemplos ilustrativos para aclaración de algunos aspectos relacionados, figuras 87 y 88.

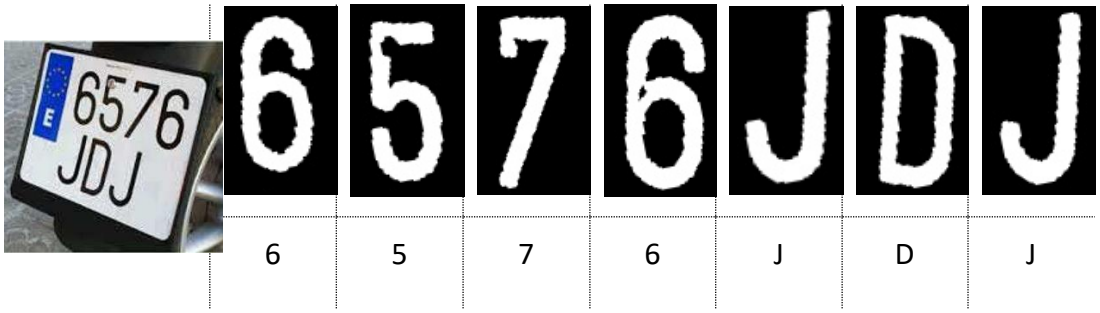


Figura 87: OCR de matrícula de moto resolución 640 x366

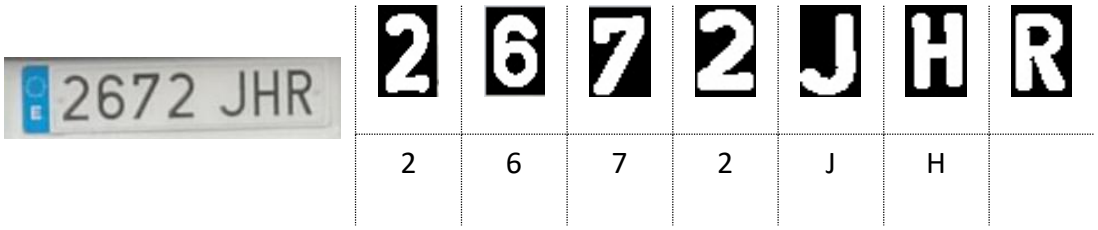


Figura 88: OCR de matrícula con resolución 230 x60

La aplicación del OCR sobre una matrícula de tamaño relativamente pequeño permite comprobar que no detecta todos los contornos de forma satisfactoria.

Para solventar este problema, se procede a realizar una aproximación de los contornos a una curva poligonal mediante la función ***approxPolyD***. Con la aproximación poligonal y con los parámetros comentados anteriormente, a pesar de que visualmente no se nota gran diferencia, sí que es cierto que los OCR distinguen mejor este tipo de imagen. No obstante, aunque los resultados son aceptables, el tiempo de ejecución se incrementa considerablemente.

5.5.2. MATCH

Otra opción sería la comparación con el alfabeto una vez que las dos imágenes tengan un tamaño proporcional.

En OpenCV existe una función llamada ***MatchTemplate*** que nos permite comparar dos imágenes.

El proceso consiste en ir recorriendo la imagen original (*I*) píxel a píxel y en cada localización, se calcula la métrica de similitud y se representa cuánto de similar es esa zona de la imagen original con la imagen que queremos encontrar (*T*). En una matriz *R* de métricas se recogen los resultados. Existen diferentes métricas al respecto, a saber, figura79:

1. **method=TM_SQDIFF**

$$R(x,y) = \sum_{x',y'} (T(x',y') - I(x+x',y+y'))^2$$

2. **method=TM_SQDIFF_NORMED**

$$R(x,y) = \frac{\sum_{x',y'} (T(x',y') - I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

3. **method=TM_CCORR**

$$R(x,y) = \sum_{x',y'} (T(x',y') \cdot I(x+x',y+y'))$$

4. **method=TM_CCORR_NORMED**

$$R(x,y) = \frac{\sum_{x',y'} (T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

5. **method=TM_CCOEFF**

$$R(x,y) = \sum_{x',y'} (T'(x',y') \cdot I'(x+x',y+y'))$$

where

$$T'(x',y') = T(x',y') - 1/(w \cdot h) \cdot \sum_{x'',y''} T(x'',y'')$$
$$I'(x+x',y+y') = I(x+x',y+y') - 1/(w \cdot h) \cdot \sum_{x'',y''} I(x+x'',y+y'')$$

Figura 89: Fuente Tutorial OpenCV

En la práctica se localiza el valor más alto o más bajo (dependiendo del método elegido usado) en la matriz *R* mediante la función **minMaxLoc**().

Este método no permite encontrar la mejor similitud, porque con cualquiera de los métodos probados encuentra el mejor punto con todas las imágenes de referencia a comparar, si bien ese punto no está colocado correctamente. Como se puede observar en la tabla 10, en la que se compara una imagen patrón con dos imágenes

de referencia, en los dos casos indica un punto como mejor resultado. Además, se puede ver cómo este punto ni siquiera es el más adecuado. Esta es la razón por la que se desecha este método.

Imagen localizada por la aplicación desarrollada	Imagen de referencia a buscar	Otra posible imagen a buscar
		
	Resultado	Resultado
		

Tabla 12: Resultados de MatchTemplate

6. CONCLUSIONES Y PROPUESTAS DE MEJORA

Durante los distintos capítulos se han ido analizando las opciones posibles, mostrando los resultados obtenidos en las diferentes pruebas realizadas y justificando con ello la elección tomada.

Por esto, en este capítulo solo se describen las conclusiones, problemas encontrados y se enumeran varias propuestas de mejora.

6.1 Conclusiones

En este proyecto se ha desarrollado una aplicación Android de lectura y reconocimiento de matrículas en movimiento mediante la combinación de varias técnicas de Visión Artificial, utilizando la librería OpenCV y aplicando técnicas avanzadas de Redes neuronales como inferencia para la tarea de detección y clasificación de la matrícula y de los dígitos de la misma.

El aspecto más importante a destacar en la realización de este proyecto ha sido tanto el aprendizaje conceptual en Visión Artificial y Redes neuronales, como el aprendizaje de las fases a seguir en un proyecto completo desde cero y la estructura necesaria para el desarrollo y finalización de un proyecto con éxito.

Estos conocimientos se pueden utilizar en cualquier otra aplicación que se tenga que desarrollar en un futuro. El desarrollo ha requerido mucho tiempo, esfuerzo y dedicación; debido básicamente a que se está demostrando prácticamente y de manera conjunta los conocimientos adquiridos durante este máster, en varias materias. Sumado a esto, se encuentra la gran cantidad de datos necesarios para poder realizar un buen entrenamiento del clasificador. Simplemente nombrar que se trabaja sobre 6500 imágenes divididas en 31 categorías correspondientes a letras y números, para lo cual se han tenido que realizar más de 5000 fotos con su correspondiente edición posterior. Hay que añadir a este trabajo la instalación y configuración correcta de las distintas librerías necesarias para el funcionamiento, y

el estudio de las librerías OpenCV, TensorFlow así como el entorno de desarrollo Android Studio.

Por otro lado, el hecho de tener que desarrollar la aplicación de forma individual, sin un código inicial o referencia ha requerido la consulta de muchas fuentes de información tales como libros o páginas web para solventar diversos problemas o cuestiones surgidos a lo largo del desarrollo.

Se ha intentado desarrollar el proyecto de manera profesional y procurando en todo momento realizar una codificación clara añadiendo comentarios y creando un anexo de funciones que facilitase a cualquier programador interesado la realización de una aplicación similar. Se ha intentado que siguiera totalmente una programación basada en objetos.

Como conclusión final desde el punto de vista técnico, comentar que, tras probar distintos métodos de Visión Artificial y Reconocimiento de Patrones durante el desarrollo del proyecto, los mejores resultados se han obtenido uniendo la detección de la zona de la matrícula mediante el proceso de detección de la zona donde se ubica, comentado en el capítulo 4, y la detección de los caracteres junto con la inferencia obtenida mediante el entrenamiento realizado en Redes Neuronales.

Aunque siempre existen muchos aspectos que podrían ser mejorados, se considera que los objetivos, marcados inicialmente, se han cumplido en gran medida. Los resultados obtenidos así lo confirman.

6.2 Propuestas de mejora

Ciertamente, a pesar de lo indicado previamente respecto de la consecución de los objetivos y de los resultados obtenidos, siempre cabe la posibilidad de realizar mejoras técnicas relativas al procesamiento de imágenes, que permitan un mejor rendimiento de la aplicación, algunas de ellas son las siguientes:

- **Modificaciones en ajustes de iluminación:** Cálculo de gamma local subdividiendo la imagen en secciones.

- **Extracción de bordes:** Utilización de otros métodos para extracción de bordes, como Harris, Sobel, y etiquetado. Uso de la transformada Hough para la obtención de líneas y posteriormente calcular los parámetros correspondientes para la transformación de perspectiva.

- **Segmentación mediante crecimiento de regiones**: Probar la técnica de segmentación de crecimiento de regiones para la determinación de los caracteres dentro de la posible placa a detectar.

- **Segmentación mediante el algoritmo de Watershed:** Utilizar la segmentación de imágenes basada en marcadores utilizando algoritmos basados en el concepto de cuencas hidrográficas (Watershed).

- **Aplicación del operador laplaciano** (derivada de segundo orden) para la creación de dos matrices con las que realizar la detección de bordes mediante el método de Canny.

- **Rectificación de contornos**: Para disponer de los caracteres de la matrícula en función de las proyecciones de dichos contornos, que es lo que se obtiene realmente debido a la transformación de perspectiva en la captura de las imágenes.

- **Comparación de caracteres:** Realizar modificaciones en el Algoritmo KNN añadiendo los valores obtenidos a los valores de entrenamiento. Probar a reducir atributos y comprobar la viabilidad del mismo. Probar a crear pesos

en los atributos. Simplificar las búsquedas en función de los resultados del primer carácter identificado, por ejemplo, si es un número los tres siguientes solo pueden ser números.

- **Red neuronal:** Ajuste de la red neuronal para la obtención de una mayor exactitud y menores pérdidas. Pruebas con otros tipos modelos con distintas capas e incluso creación de una red recurrente.

- **Datos de entrenamiento:** Para una mayor profesionalidad se requiere un número mucho más amplio de imágenes tanto para la detección de la matrícula como de los dígitos de la misma. Además, se debe ampliar el conjunto de muestras de entrenamiento incluyendo imágenes de matrículas con formatos inclinados.

- **Comprobación entre fotogramas:** para solventar errores en matrículas incompletas, como los casos comentados anteriormente, se podría añadir la comprobación entre fotogramas de tal manera que si en varios fotogramas sale la misma matrícula eliminar la matrícula anterior de la lista que era similar.

- **Conexión con base datos:** conectar con una base de datos donde guardar la placa recortada, fecha y hora y el número y caracteres encontrados en la identificación de la matrícula.

BIBLIOGRAFÍA

[1] G. y. D. l. C. J. M. Pajares, Visión por computador: imágenes digitales y aplicaciones; 2ª edición, RA-MA, 2007.

[2] J. W. H. S. H. Y. a. Y.-S. Chen, «Morphology-based-License Plate Detection from Complex Scenes Proceedings of the Complex Scenes,» Quebec, 2002.

[3] «Android,» [En línea]. Available: https://www.android.com/intl/es_es/.

[4] ORACLE, «JAVA,» [En línea]. Available: https://www.java.com/es/.

[5] «Android Studio,» [En línea]. Available: https://developer.android.com/studio.

[6] Apache, «Netbeans,» 8.3. [En línea]. Available: https://netbeans.apache.org/download/index.html.

[7] «OpenCV,» [En línea]. Available: https://opencv.org/.

[8] «Neurollabs,» [En línea]. Available: https://www.neurallabs.net/es/soluciones/control-accesos.

[9] L. E. Toledo, «Recconocimiento automático de matrículas de Automóvil,» 2005.

[10] «IEEE Std 830,» 1998. [En línea]. Available: https://standards.ieee.org/standard/830-1998.html.

[11] G. &. Woods, Digital Image processing, 2002.

[12] «Gimp,» [En línea]. Available: http://www.gimp.org.es/.

[13] E. Alegre y G. D. l. E. A. Pajares, Conceptos y métodos en Visión por Computador, 2016.

[14] «BOE 223 del 16 de Septiembre del 2020,» [En línea]. Available: https://www.boe.es/buscar/doc.php?id=BOE-A-2000-16805.

[15] «Ramer–Douglas–Peucker algorithm,» [En línea]. Available: https://en.wikipedia.org/wiki/Ramer%E2%80%93Douglas%E2%80%93Peucker_algorithm.

[16] H. P. S. D. Duda R.O, Pattern Classification, John Wiley & Sons, 2000.

[17] M. J. Paul Viola, «Rapid Object Detection using a Boosted Cascade of Simple Features,» 2001. [En línea]. Available: https://www.researchgate.net/publication/3940582_Rapid_Object_Detection_using_a_Boosted_Cascade_of_Simple_Features.

[18] P. V. a. M. Jones, «Rapid Object Detection using a Boosted Cascade of Simple Features,» 2001.

[19] Y. F. a. R. E. Schapire, «A decision-theoretic generalization of on-line learning and an application to boosting,» nº 23-37, 1995.

[20] «hmong.es,» [En línea]. Available: https://hmong.es/wiki/AdaBoost. [Último acceso: 02 04 2022].

[21] A. Ahmadi, «Cascade Trainer Gui,» [En línea]. Available: https://amin-ahmadi.com/cascade-trainer-gui/.

[22] D. o. C. S. t. U. o. A. Mahdi Rezaei, «Creating a Cascade of Haar-Like Classifiers: Step by Step».

[23] F. Rosenblatt, «The perceptron and recognizig automaton,» 1957.

[24] P. H. E. B. Gonzalo Pajares, Aprendizaje profundo, Alpha, 2021.

[25] «Tensor Flow,» [En línea]. Available: https://www.tensorflow.org/.

[26] «Neuroph,» Version 2.98. [En línea]. Available: http://neuroph.sourceforge.net/index.html.

[27] Tensorflow, «Capas de tensorflow,» 2.9. [En línea]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers.

[28] A. Rosebrock, «https://pyimagesearch.com/2018/08/20/opencv-text-detection-east-text-detector/,» [En línea]. Available: https://pyimagesearch.com/2018/08/20/opencv-text-detection-east-text-detector/.

[29] Innova, «https://www.innovagroupbcn.com/software-ocr/,» [En línea].

[30] «JavaANPR,» [En línea]. Available: http://javaanpr.sourceforge.net/.

[31] «Tess4j,» [En línea]. Available: http://tess4j.sourceforge.net/.

[32] «Tesseract,» [En línea]. Available: https://github.com/tesseract-ocr.

[33] «Tutorial OpenCV para Java,» [En línea]. Available: https://www.tutorialspoint.com/opencv/index.htm.

[34] L. A. Zadeh, «ACIMIENTO Y EVOLUCIÓN DE LA LÓGICA BORROSA, EL SOFT COMPUTING Y LA COMPUTACIÓN CON PALABRAS:UN PUNTO DE VISTA PERSONAL,» *PST,* vol. 8, nº 2, pp. 421-429, 1996.

LISTA DE SIGLAS, ABREVIATURAS Y ACRONIMOS

ANDROID: sistema operativo para dispositivos móviles.

ANDROID STUDIO: entorno de desarrollo para dispositivos con sistema operativo Android.

ANPR: reconocimiento automático del número de matrícula.

OCR: Optical Character Recognition.

MATCH: Comparación pixel a pixel entre imágenes.

KNN: algoritmo basado en instancia de tipo supervisado de Machine Learning.

HAAR: Redes neuronales basadas en características Haar.

HU: Momentos invariantes de una región.

PERCEPTRON: Neurona artificial o unidad básica de inferencia.

PHYTON: lenguaje de programación de alto nivel.

TESSERACT: Motor de reconocimiento óptimo de caracteres para varios sistemas operativos.

DNN: Deep neuronal networks

OPENCV: librería open-source para tratamiento de imágenes.

JAVA: lenguaje de programación.

TENSORFLOW-KERAS: Frameworks de alto nivel para experimentación rápida, programado en Phyton.

ML Kit: Librería de aprendizaje automático de Google para desarrollador móvil.

ANEXO 1: FUNCIONES OPENCV

Código de las principales funciones en orden de uso

ADQUISICIÓN DE VIDEO Y FOTOGRAMAS	
CLASE	DESCRIPCIÓN
VideoCapture	Utilizamos la clase VideoCapture para crear una instancia y abrir la captura de video. VideoCapture video = new VideoCapture (0); Las opciones para creación de este objeto son: • ***Index***: introducir el id de la cámara (por defecto 0) • ***Nombre de archivo***: nombre y ruta del video a abrir. video.read(imagenDeVideo); Leemos el fotograma correspondiente y debemos guardarlo en una imagen que en OpenCV corresponde a un Mat. Para más información véase la página oficial de OpencV.
DETECCIÓN DEL MOVIMIENTO	
FUNCIÓN	DESCRIPCIÓN
Absdiff	La diferencia absoluta entre las imágenes se realiza mediante la función: Core.absdiff(Mat imageninicial, Mat imagensiguiente, Mat resultado); Como parámetros introducimos dos imágenes y la diferencia absoluta nos lo guarda en resultado.
TRANSFORMACIÓN A ESPACIO DE COLOR ESCALA DE GRISES	
FUNCIÓN	DESCRIPCIÓN

<table>
<tr><td>cvtColor</td><td colspan="2">Imgproc.cvtColor(Mat origen, Mat destino, int código);
La función cvtColor nos permite transformar la imagen origen a otra imagen cambiando su tipo mediante el código.
Existen cantidades de tipos, los más importantes son:</td></tr>
<tr><td></td><td>Imgproc.COLOR_BGR2GRAY</td><td>Escala de grises</td></tr>
<tr><td></td><td>Imgproc.COLOR_BGR2HLS</td><td>Convertir de RGB a HLS</td></tr>
<tr><td colspan="3">AJUSTE DE BRILLO Y CONTRASTE</td></tr>
<tr><td colspan="3">No existe función especial para brillo y contraste se trabaja directamente con los píxeles de la imagen. Primero debemos calcular el histograma de la imagen y en función de ello podemos modificar. Una solución si no se tiene conocimiento de la imagen es realizar una ecualización del histograma o utilizar los resultados del histograma calculado el gamma como se explica en la sección 3.2.</td></tr>
<tr><td>FUNCIÓN</td><td colspan="2">DESCRIPCIÓN</td></tr>
<tr><td>calcHist</td><td colspan="2">Imgproc.calcHist(Planos, new MatOfInt(0), Mask, iHist, new MatOfInt(histSize), histRange, accumulate);
Cálculo del histograma donde:
• Planos: son una lista de imágenes en la que representamos los canales de la imagen. (por ejemplo, R,G,B). Se realiza mediante la función SPLIT
• MatOfInt(0): nos indica el número del plano sobre el que actuar.
• Mask: máscara utilizada sobre el plano. Se puede dejar vacia. (new Mat())
• iHist: La imagen donde se guardará el histograma
• histSize: indicamos el tamaño del histograma (256)
• histRange: rango del histograma (entre 0 y 255) para cada dimensión.
• accumulate: si queremos que nuestros bins tenga el mismo tamaño</td></tr>
<tr><td>equalizeHist</td><td colspan="2">Imgproc.equalizeHist(Mat origen, Mat destino);
Nos realiza una ecualización de la imagen origen y nos la introduce en la imagen destino, siguiendo el siguiente proceso:</td></tr>
</table>

	1. Calcular H histograma de src. 2. Normalizar el histograma de modo que la suma de los cubos de histograma es de 255. 3. Calcular integral del histograma: (I) H '= sum0 ≤ j ≤ iH (j) 4. Transformar la imagen utilizando H 'como una tabla de búsqueda: DST (x y) = H' (src (x y)) 5. El algoritmo normaliza el brillo y aumenta el contraste de la imagen.
ELIMINACIÓN DE RUIDOS	
FUNCIÓN	DESCRIPCIÓN
FILTRO BOXFILTER boxFilter	Imgproc.boxFilter(Mat imagen, Mat destino, 0, new Size(3,3)); Los parámetros son la imagen origen, el destino y el tamaño del kernel utilizado para la convolución con la imagen. Cada píxel destino será la media de sus vecinos (todos contribuyen por igual).
FILTRO BLUR blur	Imgproc.blur(Mat imagen, Mat destino, new Size(3,3), new Point(-1,-1); Los parámetros son la imagen origen, el destino y el tamaño del kernel utilizado para la convolución con la imagen. Además, podemos indicar el punto sobre el que se realiza. (-1,-1) es el centro del kernel.
FILTRO DE LA MEDIANA medianBlur	Imgproc.medianBlur(Mat imagen,Mat destino, int tamaño); Los parámetros son la imagen origen, el destino y el tamaño del kernel utilizado para la convolución con la imagen. Reemplaza cada píxel por la mediana de sus píxeles vecinos.
FILTRO GAUSSIANO GaussianBlur	Imgproc.GaussianBlur(imagen, destino, new Size(3,3), sigma); Los parámetros son la imagen origen, el destino y el tamaño del kernel utilizado para la convolución con la imagen. Este kernel es una aproximación a una función gaussiana en la

	que debemos indicar la sigma. El peso de los vecinos decrece en función de la distancia entre ellos.
FILTRO BILATERAL bilateralFilter	Imgproc.bilateralFilter(imagen, destino, d, sigmacolor, sigmaespacio); Considera a los vecinos, pero asignándoles pesos. Estos pesos tienen dos componentes, la primera se utiliza para el suavizado gaussiano y la segunda lleva la cuenta de la diferencia de intensidad entre los vecinos y el píxel evaluado. Los parámetros son: • imagen original • imagen destino (mismo tipo que la original) • diámetro de cada píxel vecino que es usado para filtrar. • Sigmacolor: el sigma aplicado en el espacio de color. • SigmaSpace: el sigma en el espacio de cordenadas.
[Véase los tutoriales de Suavizado de OpenCV]	
ENFOQUE Y AFILADO	
Función de convolución filter2D	Imgproc.filter2D(Mat imagen, Mat destino, int profundidad, kernel); Los parámetros son la imagen original, la profundidad de la imagen original y la máscara a utilizar. El resultado lo realiza en la imagen destino. Las distintas máscaras están comentadas en la sección de Enfoque y afilado
TRANSFORMACIONES MORFOLOGICAS	
FUNCIÓN	DESCRIPCIÓN
Erode	Imgproc.erode(Mat origen, Mat destino, Mat kernel); Donde los parámetros son la imagen origen y el kernel y el resultado se introduce en la imagen destino.
Dilate	Imgproc.dilate(Mat origen, Mat destino, Mat kernel);

	Donde los parámetros son la imagen origen y el kernel y el resultado se introduce en la imagen destino.
Cierre	Imgproc.morphologyEx(Mat origen, Mat destino, Imgproc.MORPH_CLOSE, Mat kernel); Donde los parámetros son la imagen origen y el kernel y el tipo de operación y el resultado se introduce en la imagen destino.
Apertura	Imgproc.morphologyEx(Mat origen, Mat destino, Imgproc.MORPH_OPEN, Mat kernel); Donde los parámetros son la imagen origen y el kernel y el tipo de operación y el resultado se introduce en la imagen destino.
TopCross	Imgproc.morphologyEx(Mat origen, Mat destino, Imgproc.MORPH_OPEN, Mat kernel); Donde los parámetros son la imagen origen y el kernel y el tipo de operación y el resultado se introduce en la imagen destino.
DETECCIÓN DE BORDES	
FUNCIÓN	DESCRIPCIÓN
Canny	Imgproc.Canny(Mat origen, Mat destino, int umbralinicial, int umbralfinal,tipo); Los parámetros son el origen, el umbra inicial y final y el tipo de umbralización.
UMBRALIZACIÓN	
FUNCIÓN	DESCRIPCIÓN
Threshold	Imgproc.threshold(Mat origen, Mat destino, int valor_umbral, int valor_max, int thresholdType); Los parámetros de entrada son la imagen origen el valor umbral, valor máximo y el tipo de umbral que realizamos. La salida es la imagen destino
Umbralización adaptativa adaptiveThreshold	Imgproc.adaptiveThreshold(Mat imagen, Mat destino, int maxValue, int tipoalgoritmo, int tipo_umbral, int blocksize, int C); Los parámetros de entrada son la imagen origen el valor máximo y el tipo de umbral que realizamos, tipo de algoritmo

	a utilizar, tamaño del bloque de vecinos y el tamaño de C. La salida es la imagen destino.
TRABAJO CON CONTORNOS	
FUNCIÓN	DESCRIPCIÓN
Encontrar contornos findContours	Imgproc.findContours(Mat imagen, List <MatOfPoint> contornos, Mat jerarquia, int tipo_jerarquia, int aproximacion); Los parámetros de entrada son la imagen, el tipo de jerarquía y el algoritmo de aproximación. Y los parámetros de salida son una lista de contornos y una matriz jerarquía con la relación con los demás contornos.
Aproximación de polilíneas aproxyPolDP	Imgproc.approxPolyDP(MatOfPoint2f contorno, MatOfPoint2f aproximado, double aproximacion, boolean cerrado); Los parámetros de entrada son el contorno, el valor de aproximación y si el contorno es cerrado. Y los parámetros de salida son el contorno aproximado.
Posición de un punto respecto a un contorno pointPolygonTest	double distancia=Imgproc.pointPolygonTest(MatOfPoint2f contorno, Point punto, Boolean distancia); Nos indica si un punto se encuentra dentro de un contorno, si indicamos que nos de la distancia, el resultado será la distancia al punto más cercano. Será positivo si está dentro, negativo si esta fuera y 0 si está en el borde.
Convexo convexHull	Imgproc.convexHull(MatOfPoint contorno, MatOfInt convexo); Se introduce el contorno como parámetro de salida y nos devuelve el convexo. Se debe convertir en un contorno MatOfPoint.
Momentos moments	Moments momentos = Imgproc.moments(MatOfPoint contorno, boolean binario); Calculamos los momentos de un contorno, indicamos si es una imagen binaria.
Momentos HU HuMoments	Imgproc.HuMoments(Moments momentos, Mat hu); El parámetro de entrada es la lista de momentos y la salida es una matriz con los datos de los momentos HU.

	Los normalizamos mediante la función Math.abs(Math.log(dato));
Área contourArea	Imgproc.contourArea(MatOfPoint contorno, Boolean cerrado); Nos devuelve un double con el área del contorno indicado, podemos indicar si el contorno se encuentra cerrado o no.
Rectángulo BoundingRect	Rect rectangulo=Imgproc.boundingRect(MatOfPoint contorno); Nos devuelve un objeto rectángulo que circunscribe el contorno.
Longitud ArcLength	double longitud=Imgproc.arcLength (MatOfPoint2F contorno, Boolean cerrado); Nos devuelve la longitud del perímetro del contorno. Indicamos si el contorno está cerrado o no.
Rectángulo Rotado minAreaRect	RotatedRect rectg=Imgproc.minAreaRect(MatOfPoint2f c2f); Nos devuelve el mínimo rectángulo rotado que cubre totalmente el contorno.
TRANSFORMACIONES GEOMETRICAS	
FUNCIÓN	DESCRIPCIÓN
Matriz afín getRotationMatriz2D	Mat afin = Imgproc.getRotationMatrix2D(Point centro, double angulo, escala); Nos devuelve una matriz afin. Como parámetros de entrada se debe introducir el centro, ángulo de giro y la escala.
Aplicar transformada afín warpAffine	Imgproc.warpAffine(Mat imagen, Mat destino, Mat afin, int destino.size()); Nos crea una matriz destino aplicando la matriz afín a nuestra imagen. Debemos indicar también el tamaño de la imagen destino.
Escalar resize	Imgproc.resize(Mat imagen, Mat destino, Size sz,int fx, int fy,int tipo_interpolacion); Nos crea una matriz destino aplicando a la matriz imagen una interpolación (se puede configurar varias interpolaciones)

	para que el destino tenga el tamaño sz que le indicamos. O también podemos indicar la escala en x y la escala en y .
DIBUJO DE ELEMENTOS	
FUNCIÓN	DESCRIPCIÓN
Dibujo de contornos drawContours	Imgproc.drawContours(Mat imagen, List <MatOfPoint> contornos, contornos, int idcontorno, Scalar color, int grosor); En la imagen se dibujan los contornos con el color marcado y el grosor seleccionado. Mediante el id_contorno se puede indicar el índice del contorno o -1 para todos los contornos. Si en grosor se pone -1 los contornos se rellenan del color marcado.
Dibujo de líneas drawLines	Imgproc.line(Mat imagen, Point punto1, Point punto, Scalar color, int grosor); Se dibuja una línea entre el punto1 y el punto2 con el color indicado y con el grosor indicado.
TRABAJO CON TEXTOS	
FUNCIÓN	DESCRIPCIÓN
Poner texto Imgproc.putText	Imgproc.putText (Mat imagen, String texto, Point origen, int fontFace, int fontScale, Scalar color, int thickness) Los parámetros son: • Imagen: el objeto Mat que representa la imagen. • Texto: un String con el texto • Origen: un punto con las coordenadas de la esquina inferior izquierda donde se situará el texto • fontFace: un entero para indicar el tipo de letra (dentro de las contenidas por el Sistema) • fontScale: un entero para indicar la escala • color: un escalar que representa el color del texto (RGB) • thickness: grosor de la línea
TRABAJO CON CLASIFICADOR HAAR Y REDES NEURONALES	
FUNCIÓN	DESCRIPCIÓN
Creación de modelo createsamples	Aplicación auxiliar para la creación del modelo de entrenamiento.

	• vec <vec_file_name>: Nombre del archivo de salida que contiene las muestras positivas para el entrenamiento. • img <image_file_name>: imagen del objeto de origen • bg <background_file_name>: archivo de descripción de fondo • num <number_of_samples>: Número de muestras positivas a generar. • bgcolor <background_color>: color de fondo • inv: si se especifica, los colores se invertirán. • maxidev <max_intensity_deviation>: Desviación de intensidad máxima de píxeles en muestras de primer plano. • w <sample_width>: Ancho (en píxeles) de las muestras de salida. • -h <sample_height>: Altura (en píxeles) de las muestras de salida.
Entrenamiento de modelo *traincascade*	Aplicación auxiliar para entrenamiento del modelo. • data <cascade_dir_name> : donde se debe almacenar el clasificador entrenado • -vec <vec_file_name>: archivo vec con muestras positivas • bg <background_file_name>: Este es el archivo que contiene las imágenes de muestra negativas. • -numPos <number_of_positive_samples>: Número de muestras positivas utilizadas en el entrenamiento para cada etapa del clasificador. • -numNeg <number_of_negative_samples>: Número de muestras negativas utilizadas en el entrenamiento para cada etapa del clasificador. • -numStages <number_of_stages>: Número de etapas en cascada que se entrenarán. • -precalcValBufSize: Tamaño del búfer para valores de características precalculados • -numThreads <max_number_of_threads>: número máximo de subprocesos para usar durante el entrenamiento- • -acceptanceRatioBreakValue <break_value>: este argumento se usa para determinar la precisión con la que su modelo debe seguir aprendiendo y cuándo detenerse-

	• -featureType<{HAAR(predeterminado), LBP}> : Tipo de funciones: HAAR: funciones similares a Haar, LBP: patrones binarios locales. • -w <sampleWidth>: Ancho de las muestras de entrenamiento (en píxeles). • -h <sampleHeight>: Altura de las muestras de entrenamiento (en píxeles). Parámetros del clasificador potenciado: • -bt <{DAB, RAB, LB, GAB(predeterminado)}> : Tipo de clasificadores potenciados: DAB (AdaBoost discreto), RAB(AdaBoost real), LB (LogitBoost, GAB), AdaBoost suave. • -minHitRate <min_hit_rate>: Tasa de acierto mínima deseada para cada etapa del clasificador. • -maxFalseAlarmRate <max_false_alarm_rate>: Tasa máxima deseada de falsas alarmas para cada etapa del clasificador. • -maxDepth <max_ depth_of_weak_tree>: Profundidad máxima de un árbol débil. • -maxWeakCount <max_weak_tree_count>: Recuento máximo de árboles débiles para cada etapa de cascada. Parámetros de características similares a Haar: • -modo <BÁSICO (predeterminado)| ALL>: Selecciona el tipo de conjunto de características de Haar utilizado en el entrenamiento.
Detección de objetos detectMultiScale	detectMultiScale(Mat image, MatOfRect objects, double scaleFactor, int minNeighbors) Nos detecta los rectángulos de diferentes tamaños en la imagen de entrada. Parámetros: • image: Matriz del tipo CV_8U que contiene una imagen donde se detectan objetos. • Objects: Vector de rectángulos donde cada rectángulo contiene el objeto detectado, los rectángulos pueden estar parcialmente fuera de la imagen original. • scaleFactor: parámetro que especifica cuánto se reduce el tamaño de la imagen en cada escala de imagen.

	• minNeighbors: parámetro que especifica cuántos vecinos debe tener cada rectángulo candidato para conservarlo.
Lectura módelo TensorFlow readNetFrom TensorFlow	Cargar un modelo entrenado con TensorFlow. Parámetros: • bufferModel: búfer que contiene el contenido del archivo pb • bufferConfig: búfer que contiene el contenido del archivo pbtxt
Convertir a imagen de 4 dimensiones blobFromImage	Mat blobFromImage(Mat image, double scalefactor, Size size) Crea un blob de 4 dimensiones a partir de una imagen. Opcionalmente, cambia el tamaño y recorta la imagen desde el centro, resta los valores medios, escala los valores por factor de escala, intercambia los canales azul y rojo. Parámetros: • image: imagen de entrada • tamaño: tamaño espacial para que la imagen de salida esté en orden RGB si la imagen tiene un orden BGR • scalefactor: factor de escala
Pasar el blob a la red Net.setInput	net.setInput(Mat blob) Se pasa a la red el blob creado anteriormente.
Ejecución de la red Net.forward	Net.forward(List<Mat> result, List <String> outBlobNames) Ejecuta la red hacia adelante para calcular la salida de la capa con el nombre outputName por defecto. Si la capa de salida tuviese otro nombre, hay que indicárselo como parámetro en la lista. Devuelve una lista de candidatos indicando la posición x e y del centro y la altura y anchura del candidato.

ANEXO 2: GLOSARIO

Píxel

Elemento más pequeño en que puede dividirse una imagen digital la superficie real que representa cada uno de ellos define los objetos o detalles más pequeños que pueden observarse en una imagen. El cual es utilizado por las cámaras por la limitación en la visión humana que lo percibe como un conjunto y no como unidades independientes

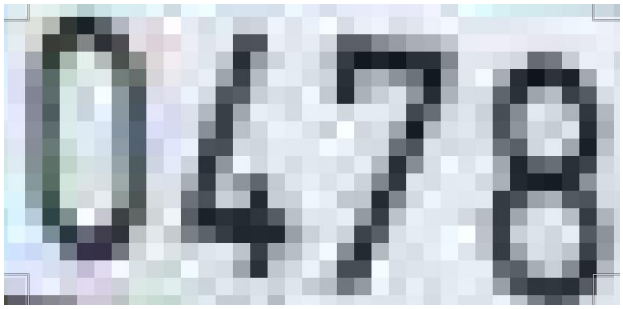


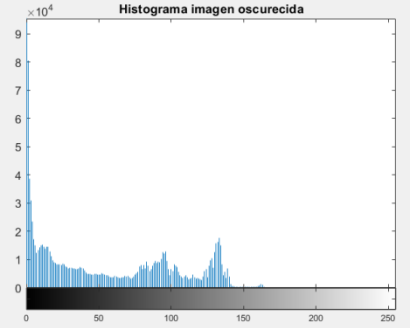
Imagen 90: Pixel

Histograma

Un histograma es un gráfico que muestra la distribución de los colores o tonos de un color en una imagen según su luminosidad. En el histograma, el eje horizontal indica la luminosidad o intensidad, a la vista de su representación gráfica se determina que la acumulación de valores más a la izquierda la imagen es más oscura y más a la derecha, más luminosa. El eje vertical indica la cantidad de píxeles con esa luminosidad. Un pico del histograma en el lado izquierdo indica un gran número de píxeles oscuros o negros (posiblemente una foto subexpuesta) mientras que un pico en la parte derecha indica un gran número de luminosos o blancos (posiblemente una

foto sobreexpuesta). Por este razonamiento un histograma uniforme (sin picos) en todos los tonos es probable que indique que la imagen está debidamente expuesta.

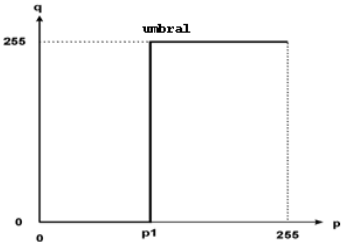
El histograma adjunto al texto muestra que la distribución de intensidades no cubre todo el rango posible de los 256 niveles. También expresa una distribución multimodal que corresponde a los distintos tipos de objetos que aparecen en la imagen: fondo de la regla números grabados en la misma y fondo de la imagen.



Nivel de gris: Luminosidad o intensidad del píxel que va del valor 0 (negro) al 255 (blanco). El tono de gris de cada píxel se puede obtener, bien asignándole un valor de intensidad en el rango 0 a 255 o bien como porcentajes de tinta negra (0% es igual a blanco y 100% es igual a negro). Las imágenes producidas con escáneres en blanco y negro o en escala de grises se visualizan normalmente en el modo escala de grises. Este modo maneja un solo canal (el negro) para trabajar con imágenes monocromáticas de 256 tonos de gris entre el blanco y el negro

Umbralización

También conocida como *thresholding* es una técnica de segmentación de imágenes en la que cada píxel pertenece obligatoriamente a un segmento y sólo uno. Si el nivel de gris del píxel es menor o igual al umbral especificado se pone a cero si es menor se pone al máximo valor (normalmente 255). En función del valor umbral que se escoja el tamaño de los objetos irá oscilando



$$dst(x,y) = \begin{cases} maxValor & src(x,y) > umbral \\ 0 & demas\ casos \end{cases}$$

Segmentación

La segmentación consiste en dividir una imagen en varias regiones (grupos de píxeles) denominadas segmentos. Más concretamente, la segmentación es un proceso de clasificación por píxel que asigna una categoría a cada píxel de la imagen analizada.

***Kernel* o Máscara**

En procesamiento de imagen, un núcleo, *kernel*, matriz de convolución o máscara es una matriz de dimensión mucho menor que la imagen original que se utiliza con distintos propósitos, tales como para desenfoque, enfoque, realce, detección de bordes y más. Esto se logra realizando una operación de convolución entre un núcleo y una imagen. Dependiendo de esta matriz se obtienen diferentes resultados.

Convolución

La convolución en un operador matemático que transforma dos funciones *f* y *g* en una tercera función que en cierto sentido representa la magnitud en la que se superponen *f* y una versión trasladada e invertida de *g*. La función de convolución se expresa por el símbolo *.

En matrices se considera la convolución como la multiplicación de una matriz elemento a elemento por otra que se llama "*kernel*" y sumando el resultado. El filtro matriz de convolución usa una primera matriz que es la imagen tratada. La imagen es

una colección bidimensional de píxeles en coordenadas rectangulares. El filtro examina, sucesivamente, cada píxel de la imagen. Para cada uno de ellos, que llamaremos “píxeles iniciales”, se multiplica el valor de este píxel y el valor de los 8 circundantes (si el *kernel* es de 3x3) por el valor correspondiente del *kernel*. Entonces se añade el resultado, y el píxel inicial se regula en este valor resultante final.

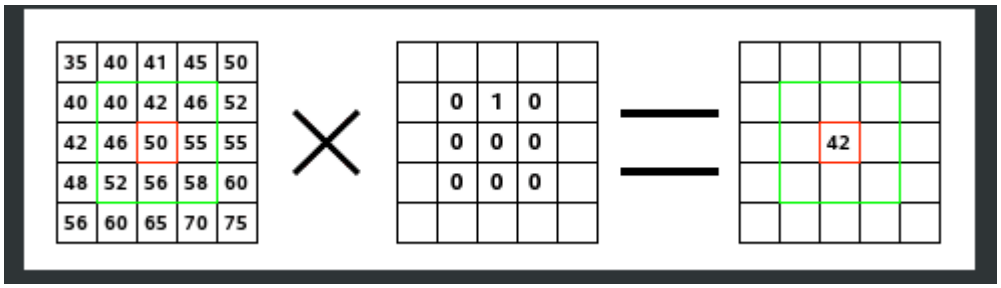


Imagen 91: Convolución

Fuente: Manual de Gimp

Momentos

Un momento es una medida para extraer propiedades de regiones en una imagen, resultando ser cierto promedio ponderado particular de las intensidades de los píxeles de la región; o también una función de tales momentos.

Los momentos de imagen son útiles para describir objetos por sus propiedades luego de una segmentación. Algunas propiedades simples de la imagen, tales como centroide, área y orientación, se obtienen a partir de esos momentos.

Los momentos centrales son invariantes a la traslación, los momentos normales invariantes también a la escala y los momentos de Hu invariantes a traslación, rotación y escala.

Algoritmo KNN

Es un algoritmo basado en instancia de tipo supervisado dentro del paradigma de Machine Learning. El algoritmo clasifica cada dato nuevo en el grupo que corresponda, según tenga *k* vecinos más cerca de un grupo o de otro. Es decir, calcula la distancia del elemento nuevo a cada uno de los existentes, y ordena dichas distancias de menor a mayor para ir seleccionando el grupo al que pertenecer. Este grupo será, por tanto, el de mayor frecuencia con menores distancias.

K-NN no genera un modelo fruto del aprendizaje con datos de entrenamiento, sino que el aprendizaje sucede en el mismo momento en el que se prueban los datos de test. A este tipo de algoritmos se les llama Lazy learning methods.

Clasificador HAAR

Los clasificadores Haar Cascade son una forma eficaz de detección de objetos. Este método fue propuesto por Viola y Jones [17] y [18] en el que se utilizan muchas imágenes positivas y negativas para entrenar al clasificador.

En el apartado 3 del capítulo 5 de este proyecto se explica con más detalle.

Redes neuronales

Las Redes Neuronales Artificiales (Artificial Neural Network, ANN / NN) son un enfoque conexionista de la Inteligencia Artificial (IA), vagamente inspirado en sus homólogos biológicos.

Nos encontramos con varias arquitecturas en función de sus dimensiones, diferentes capas conectadas a regiones de las anteriores y siguientes capas, diferentes tipos de neuronas, minimizando el preprocesamiento previo de datos.

En el apartado 4 del capítulo 5 de este proyecto se explica con más detalle.

OCR

Técnicamente, OCR significa Reconocimiento Óptico de Caracteres, lo que permite que el sistema reconozca los caracteres como parte de un alfabeto.

Todos los algoritmos de OCR tienen la finalidad de poder diferenciar un texto de una imagen cualquiera. Para hacerlo se basan en cuatro etapas, comentadas en el apartado 5 del capítulo 5.

ANEXO 3: CÓDIGO RED NEURONAL TENSORFLOW-KERAS

Cargar y procesar las imagenes

```
[1] import numpy as np
    import os
    import io
    import zipfile
    import PIL
    import PIL.Image
    #  %tensorflow_version 1.x
    import tensorflow as tf
    import tensorflow_datasets as tfds
    import pathlib
    from tensorflow.python.framework.convert_to_constants import convert_variables_to_constants_v2
```

Descomprimir imagenes

```
from google.colab import files
uploader=files.upload()
data = zipfile.ZipFile(io.BytesIO(uploader['letras.zip']),'r')
data.extractall()
```

Elegir archivos letras.zip

- **letras.zip**(application/x-zip-compressed) - 21672251 bytes, last modified: 22/5/2022 - 100% done

```
Saving letras.zip to letras.zip
```

Número de imagenes totales

```
[3] data_dir = pathlib.Path("letras")
    image_count = len(list(data_dir.glob('*/*.jpg')))
    print(image_count)
```

```
6456
```

Abrimos una imagen para comprobar si la carga ha sido correcta

```
[4] letraC = list(data_dir.glob('15x24letraC/*'))
    PIL.Image.open(str(letraC[0]))
```

Creamos el dataset

Parametros para el entrenamiento

```
[5]  batch_size = 62
     img_height = 24
     img_width = 15
```

Dividimos las imagenes en entrenamiento(80%) y validación (20%).

```
[6]  train_ds = tf.keras.utils.image_dataset_from_directory(
       data_dir,
       validation_split=0.2,
       subset="training",
       seed=123,
       image_size=(img_height, img_width),
       batch_size=batch_size)
```

```
Found 6658 files belonging to 31 classes.
Using 5327 files for training.
```

```
[7]  val_ds = tf.keras.utils.image_dataset_from_directory(
       data_dir,
       validation_split=0.2,
       subset="validation",
       seed=123,
       image_size=(img_height, img_width),
       batch_size=batch_size)
```

```
Found 6658 files belonging to 31 classes.
Using 1331 files for validation.
```

Imprimimos nombre de las clases para comprobar que esten todas las letras

```
[8]  class_names = train_ds.class_names
     print(class_names)
```

```
['14x28azul', '15x24letraB', '15x24letraC', '15x24letraD', '15x24letraF', '15x24letraG',
```

Visualizar los datos

Visualizar las primeras 9 figuras

```
[9]  import matplotlib.pyplot as plt

     plt.figure(figsize=(10, 10))
     for images, labels in train_ds.take(1):
       for i in range(9):
         ax = plt.subplot(3, 3, i + 1)
         plt.imshow(images[i].numpy().astype("uint8"))
         plt.title(class_names[labels[i]])
         plt.axis("off")
```

```
[10] for image_batch, labels_batch in train_ds:
       print(image_batch.shape)
       print(labels_batch.shape)
       break
```

```
(62, 24, 15, 3)
(62,)
```

La `image_batch` es un tensor de la forma `(62, 24, 15, 3)`. Es un lote de 62 imagenes de la forma `24x15x3` (la ultima dimensión corresponde a los canales RGB). El `label_batch` es un tensor de la forma `(62,)`, que corresponde a las etiquetas de las clases.

Estandarizar los datos

Los valores de los canales RGB son valores comprendidos entre `[0, 255]`. Esto no es ideal para una red neuronal, en general es bueno que los valores sean mas pequeños.

Por eso, aqui estandarizamos a valores entre `[0, 1]` usando `tf.keras.layers.Rescaling`:

```
[11] normalization_layer = tf.keras.layers.Rescaling(1./255)
```

Hay 2 maneras para utilizar esta capa, llamando a la función `Dataset.map` o incluirlo dentro de la definición del modelo. Esto simplifica el proceso. Tambien se podia haber utilizado la capa `tf.keras.layers.Resizing` si las imagenes no hubieran estado ya escaladas.

```
[12] normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
     image_batch, labels_batch = next(iter(normalized_ds))
     first_image = image_batch[0]
     # Notice the pixel values are now in `[0,1]`.
     print(np.min(first_image), np.max(first_image))
```

```
0.19215688 0.7176471
```

Configurar el dataset para el entrenamiento.

- `Dataset.cache` mantiene las imagenes en memoria despues de haberlas cargado del disco durante el primer epoch, para que no se produzca un cuello de botella, pero si las imagenes son demasiado pesadas o grandes podemos elegir la opcion de performant on-disk cache.

```
[13] AUTOTUNE = tf.data.AUTOTUNE

     train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
     val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

```
[36] num_classes = len(class_names)

     model = Sequential([
       layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
       layers.Conv2D(16, 3, padding='same', activation='relu'),
       layers.MaxPooling2D(),
       layers.Conv2D(32, 3, padding='same', activation='relu'),
       layers.MaxPooling2D(),
       layers.Conv2D(64, 3, padding='same', activation='relu'),
       layers.MaxPooling2D(),
       layers.Flatten(),
       layers.Dense(128, activation='relu'),
       layers.Dense(num_classes)
     ])
```

Se elige el optimizador `tf.keras.optimizers.Adam` y como función de perdida `tf.keras.losses.SparseCategoricalCrossentropy`. Para ver en cada fase los resultados de entrenamiento y validación se activa la metrica `accuracy`.

```
[15] model.compile(
       optimizer='adam',
       loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
       metrics=['accuracy'])
```

Resumen del modelo

Visión de las capas del model utilizando la función `Model.summary`

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_3 (Rescaling)     (None, 24, 15, 3)         0

 conv2d_3 (Conv2D)           (None, 24, 15, 16)        448

 max_pooling2d_3 (MaxPooling  (None, 12, 7, 16)        0
 2D)

 conv2d_4 (Conv2D)           (None, 12, 7, 32)         4640

 max_pooling2d_4 (MaxPooling  (None, 6, 3, 32)         0
 2D)

 conv2d_5 (Conv2D)           (None, 6, 3, 64)          18496

 max_pooling2d_5 (MaxPooling  (None, 3, 1, 64)         0
 2D)

 flatten_1 (Flatten)         (None, 192)               0

 dense_2 (Dense)             (None, 128)               24704

 dense_3 (Dense)             (None, 31)                3999

=================================================================
Total params: 52,287
Trainable params: 52,287
Non-trainable params: 0
_________________________________________________________________
```

Entrenando el modelo

```
epochs=10
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
Epoch 1/10
167/167 [==============================] - 2s 8ms/step - loss: 2.3528 - accuracy: 0.3396 - val_loss: 0.8346 - val_accuracy: 0.7708
Epoch 2/10
167/167 [==============================] - 1s 4ms/step - loss: 0.5933 - accuracy: 0.8213 - val_loss: 0.5194 - val_accuracy: 0.8550
Epoch 3/10
167/167 [==============================] - 1s 5ms/step - loss: 0.3801 - accuracy: 0.8859 - val_loss: 0.3161 - val_accuracy: 0.9181
Epoch 4/10
167/167 [==============================] - 1s 4ms/step - loss: 0.2794 - accuracy: 0.9174 - val_loss: 0.2711 - val_accuracy: 0.9279
Epoch 5/10
167/167 [==============================] - 1s 4ms/step - loss: 0.2259 - accuracy: 0.9371 - val_loss: 0.2582 - val_accuracy: 0.9249
Epoch 6/10
167/167 [==============================] - 1s 4ms/step - loss: 0.1753 - accuracy: 0.9495 - val_loss: 0.1818 - val_accuracy: 0.9482
Epoch 7/10
167/167 [==============================] - 1s 4ms/step - loss: 0.1445 - accuracy: 0.9564 - val_loss: 0.1893 - val_accuracy: 0.9489
Epoch 8/10
167/167 [==============================] - 1s 4ms/step - loss: 0.1216 - accuracy: 0.9668 - val_loss: 0.1817 - val_accuracy: 0.9512
Epoch 9/10
167/167 [==============================] - 1s 4ms/step - loss: 0.1056 - accuracy: 0.9688 - val_loss: 0.2135 - val_accuracy: 0.9339
Epoch 10/10
167/167 [==============================] - 1s 4ms/step - loss: 0.0859 - accuracy: 0.9762 - val_loss: 0.1916 - val_accuracy: 0.9452
```

Visualizar resultados

Crear plots de perdida y exactitud en los conjuntos de entrenamiento y validación.

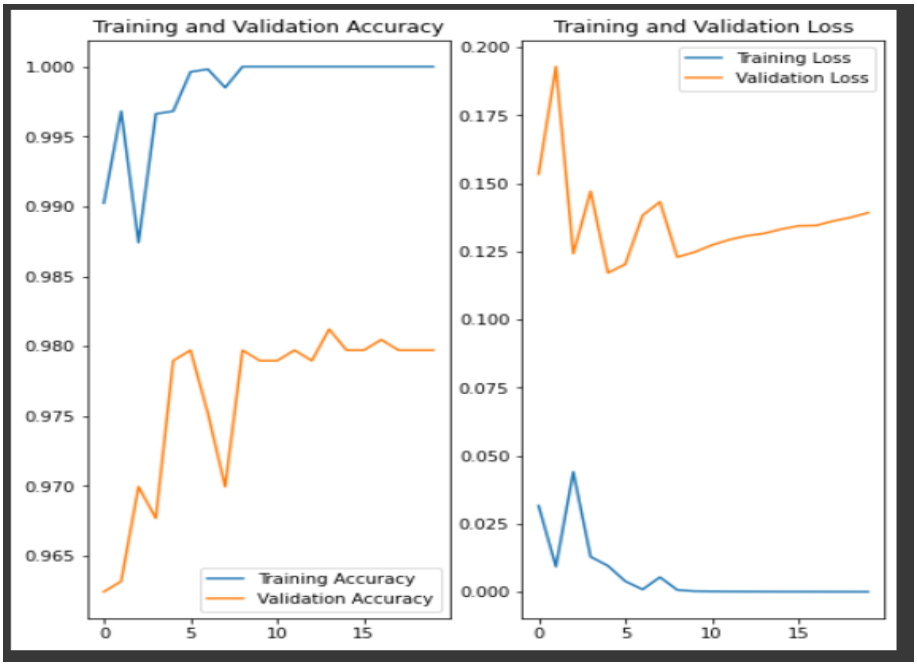
```
[40] acc = history.history['accuracy']
     val_acc = history.history['val_accuracy']

     loss = history.history['loss']
     val_loss = history.history['val_loss']

     epochs_range = range(epochs)

     plt.figure(figsize=(8, 8))
     plt.subplot(1, 2, 1)
     plt.plot(epochs_range, acc, label='Training Accuracy')
     plt.plot(epochs_range, val_acc, label='Validation Accuracy')
     plt.legend(loc='lower right')
     plt.title('Training and Validation Accuracy')

     plt.subplot(1, 2, 2)
     plt.plot(epochs_range, loss, label='Training Loss')
     plt.plot(epochs_range, val_loss, label='Validation Loss')
     plt.legend(loc='upper right')
     plt.title('Training and Validation Loss')
     plt.show()
```



Guardar el modelo

```
# Convertir el modelo
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Salvarlo en disco
with open('model.tflite', 'wb') as f:
  f.write(tflite_model)
```

Predicción

```
#Subimos una imagen escalada en formato
imagen_letra= 'letra_desconocida.jpg'

img = tf.keras.utils.load_img(
    imagen_letra, target_size=(img_height, img_width)
)
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "Esta imagen pertenece a la clase {} con un porcentaje de confianza de un {:.2f}."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

```
Esta imagen pertenece a la clase 15x24letraK con un porcentaje de confianza de un 100.00.
```

Introduccion de metadatos

```
model_buffer = writer_utils.load_file("model_without_metadata/model.tflite")

# Información general del modelo
general_md = metadata_info.GeneralMd(
    name="Clasificador de letras",
    version="v1",
    description=("Identifica la letra mas parecida dentro un conjunto de categorias "),
    author="Berta",
    licenses="Apache License. Version 2.0")

# Información tensor de entrada
input_md = metadata_info.InputImageTensorMd(
    name="Imagen de entrada",
    description=("Las imagenes esperadas son de 15x24 con 3 canales (RGB) "),
    norm_mean=[127.5],
    norm_std=[127.5],
    color_space_type=_metadata_fb.ColorSpaceType.RGB,
    tensor_type=writer_utils.get_input_tensor_types(model_buffer)[0])

# Información tensor de salida
output_md = metadata_info.ClassificationTensorMd(
    name="probabilidad",
    description="Probabilidad respectivamente de las 31 etiquetas.",
    label_files=[
        metadata_info.LabelFileMd(file_path="model_without_metadata/labels.txt",
                                  locale="es")
    ],
    tensor_type=writer_utils.get_output_tensor_types(model_buffer)[0])
```

Guardado con metadatos

```
[ ] ImageClassifierWriter = image_classifier.MetadataWriter
    # Create the metadata writer.
    writer = ImageClassifierWriter.create_from_metadata_info(
        model_buffer, general_md, input_md, output_md)

    # Verify the metadata generated by metadata writer.
    print(writer.get_metadata_json())

    # Populate the metadata into the model.
    writer_utils.save_file(writer.populate(), "model_without_metadata/model_data.tflite")
```