Facultad de Ciencias

FUNDAMENTOS DE VISIÓN ARTIFICIAL CON APLICACIÓN A LA LECTURA DE MATRÍCULAS DE COCHES

ARTIFICIAL VISION FUNDAMENTALS AND ITS APPLICATIONS TO CAR LICENSE PLATE READING

Trabajo de fin de grado
para acceder al

GRADO EN MATEMÁTICAS

Autor: Rodrigo Cepeda Marín

Director: Domingo Gómez Pérez

11 de MAYO de 2018

Agradecimientos

A mi familia y amigos.

FUNDAMENTOS DE VISIÓN ARTIFICIAL CON APLICACIÓN A LA LECTURA DE MATRÍCULAS DE COCHES

Resumen

palabras clave: reconocimiento de matrículas, clasificador en cascada, árbol de decisión, análisis de imagen

La visión artificial es una rama de la ciencia computacional que investiga como extraer información de imágenes reales, de forma que esta información pueda ser tratada sin supervisión humana. Entre las tecnologías más importantes que se estudian en esta rama están el reconocimiento de objetos y el reconocimiento óptico de caracteres. Se definen brevemente a continuación:

- El reconocimiento de objetos es la tarea de encontrar e identificar objetos en una imagen. Esta tarea requiere de algoritmos de minería de datos como son los clasificadores en cascada y Adaboost entre otras técnicas.
- El reconocimiento óptico de caracteres es el problema de reconocer una serie de símbolos pertenecientes a un determinado alfabeto conocido.

En este trabajo se han estudiado los fundamentos teóricos detrás de dos paquetes de software que implementan soluciones a estos problemas, OpenCV y Tesseract, y han sido aplicados al reconocimiento de matrículas de vehículos. La tareas fundamentales realizadas han sido:

- Análisis de la sensibilidad de un clasificador de la librería OpenCV ante la variación de parámetros presentes en el clasificador. Posteriormente se ha buscado y seleccionado el valor de los parámetros que aplican el clasificador de manera óptima.
- Aplicación de la librería Pytesseract a un conjunto de imágenes y realización de un análisis de sensibilidad de dicha librería ante una serie de parámetros que la condicionan.

ARTIFICIAL VISION FUNDAMENTALS AND ITS APPLICATIONS TO CAR LICENSE PLATE READING

Abstract

keywords: license plate recognition, cascade classifier, decision tree, image analysis

Artificial vision is a branch of computational science that investigates how to extract information from real images, so that this information can be treated without human supervision. Among the most important technologies that are studied in this branch are object recognition and optical character recognition. They are briefly defined below:

- The recognition of objects is the task of finding and identifying objects in an image. This task requires data mining algorithms such as cascading classifiers and Adaboost among other techniques.
- Optical character recognition is the problem of recognizing a series of symbols belonging to a certain known alphabet.

In this paper we have studied the theoretical foundations behind two software packages that implement solutions to these problems, OpenCV and Tesseract, and have been applied to the recognition of vehicle license plates. The fundamental tasks carried out have been:

- Sensitivity analysis of a classifier from the OpenCV library, varying on the parameters contained in the classifier. Subsequently, the value of the optimal parameters that apply to the classifier have been seeked and selected. Subsequently, the value of the parameters that optimally apply the classifier have been seeked and selected.
- Application of the Pytesseract library to a set of images, then a sensitivity analysis was performed on that library depending on a series of parameters that condition it.

Índice

1. Introducción y herramientas utilizadas **1**
1.1. Introducción . . . 1
1.2. Herramientas utilizadas . . . 2

2. Marco de referencia **5**
2.1. Convergencia de los conceptos presentados . . . 9

3. Trabajo y metodología **13**
3.1. Obtención de resultados en OpenCV . . . 14
3.2. Análisis de los resultados obtenidos de OpenCV y aplicación de Pytesseract . . . 16

4. Resultados y Análisis **17**
4.1. Resultados Obtenidos . . . 17
4.2. Análisis de los conjuntos de resultados retornados por el clasificador . . . 18
4.3. Selección de imágenes a procesar . . . 20
4.4. Análisis de sensibilidad para medir la efectividad de la librería Pytesseract . . . 22

5. Resumen y conclusiones **27**
5.1. Resumen . . . 27
5.2. Conclusiones . . . 27

A. Código Utilizado **29**

Referencias **33**

1 Introducción y herramientas utilizadas

1.1. Introducción

> Una imagen puede definirse como una función bidimensional, $f(x, y)$ donde x e y son coordenadas espaciales (planas), y la amplitud de f en cualquier par de coordenadas (x, y) se llama intensidad o nivel de grises de la imagen en ese punto. Cuando x, y y los valores de intensidad de f son finitos y discretos, llamamos a la imagen: imagen digital.
>
> (Gonzalez [2016])

Desde comienzos del siglo XX, el tratamiento de imágenes ha sido una práctica frecuente. La transmisión a través del cable de Bartlante, una de las primeras aplicaciones conocidas, codificaba y transmitía imágenes para su posterior impresión a través de un cable que cruzaba el océano atlántico. Aunque esto no se puede considerar procesamiento de imágenes digitales ya que no involucraba un equipo informático, se puede considerar como uno de los primeros pilares de desarrollo de este campo se comenzaban a situar.
Fue poco después, en los años 60, cuando se creó la primera computadora con capacidad suficiente para realizar procesos en imágenes digitales. Este hecho, unido con el desarrollo de áreas como la observación terrestre en astronomía, o técnicas de Rayos-X en la medicina, que utilizaban dichos procesos, provocó un crecimiento exponencial en este campo.

De los múltiples procesos que se pueden realizar en una imagen digital, este trabajo se centrará en dos en concreto. El primero, será el de la detección de objetos, un proceso que consiste en la localización de uno o varios elementos iguales determinados en la totalidad de una imagen.

> La detección de objetos de interés del mundo real, como caras y personas, plantea problemas desafiantes: estos objetos son difícil de modelar, hay una gran variedad en el color y textura, y no existen restricciones para el número de fondos sobre los que el objeto se puede apoyar
>
> (Papageorgiou et al. [1998])

El segundo proceso sobre una imagen digital en el que se centrará este trabajo será el de «Optical Character Recognition» (OCR en adelante).

> El OCR trata el problema del reconocimiento óptico de caracteres procesados. El reconocimiento óptico se realiza off-line después de que la escritura o impresión haya sido completada.
>
> (Eikvil [1993])

Es importante diferenciar entre procesos de OCR en documentos escaneados que han sido previamente impresos con una tipografía determinada, y procesos de OCR en imágenes, donde la falta de calidad o de contraste de la imagen puede dificultar su procesado.

El propósito del presente Trabajo de Fin de Grado es el de aplicar la detección de objetos así como los procesos de OCR a matrículas de vehículos. En este caso las imágenes provendrán de diferentes fuentes de Internet, obtenidas especialmente de dispositivos de captura de vídeo y de cámaras digitales.

La diversa procedencia de las imágenes tiene un problema principal, y es la falta de uniformidad de formato (ángulo, distancia a la matrícula, resolución...) que tendrán la imágenes.
El número de aplicaciones que tiene la detección y lectura de matrículas en vehículos es muy elevado, por poner algunos ejemplos:

- Control de entradas y salidas de los vehículos en un aparcamiento, así como su tiempo de estancia.
- Control de vehículos que incurren en infracciones de tráfico. Ya sea con imágenes tomadas por un radar como por vídeos grabados por una cámara de tráfico.
- Control fronterizo de vehículos asociados a personas con antecedentes penales o bajo búsqueda policial.
- Posibilidad de, mediante el posicionamiento de varias cámaras, conseguir conocer todos los movimientos que realiza el flujo de tráfico en una glorieta con el objetivo de modelizarla.

1.2. Herramientas utilizadas

Para lograr el objetivo fijado, las herramientas utilizadas han sido un ordenador portátil HP ENVY x360 Convertible con procesador Intel Core i7-6500U y 8,00 GB de RAM que ha dado soporte para escribir el código en lenguaje Python 3.6.3 y la memoria en LaTeX 2_ε.

Además, se deben añadir las dos librerías presentes en Python sobre las que se ha desarrollado el presente Trabajo de Fin de Grado: OpenCV y Pytesseract

- OpenCV: Open Source Computer Vision Library es una librería abierta de software de visión y aprendizaje automático desarrollada en 1999 por Intel Research capaz de aplicar el algoritmo de la imagen integral.

  > OpenCV es empleada por empresas bien establecidas tales como Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda o Toyota.
  >
  > (OpenCV [2018])

  La librería OpenCV, a su vez, tiene implementado un clasificador en cascada llamado «haarcascade-russian-plate-number.xml», que permite el reconocimiento de matrículas de origen ruso en imágenes de vehículos. Este clasificador se apoya en una serie de parámetros, de los cuales los más importantes se consideran:

  - ScaleFactor: Parámetro estrechamente ligado con el tamaño de la imagen en el que se encuentra una matrícula. Durante su entrenamiento, el clasificador tiene fijado el tamaño de la imagen en la que se encuentra una matrícula. Aumentando el parámetro scaleFactor, aumenta a su vez el tamaño que puede tener una matrícula en la imagen y así la posibilidad de encontrarla.
  - MinNeighbors: Parámetro que especifica el número mínimo de veces que tiene que ser detectado un objeto en bloques continuos dentro de una imagen para que su detección sea admitida. Un valor mayor pequeño en este parámetro tendrá como consecuencia una gran cantidad de falsas detecciones, mientras que un valor mayor conlleva la posible no detección de los objetos buscados.

- Pytesseract: La librería implementada para Python de Tesseract.

> Tesseract es un motor abierto de OCR que fue desarrollado entre 1984 y 1994. Tesseract comenzó como un proyecto de investigación de doctorado en los laboratorios de HP en Bristol, y ganó impulso como un posible complemento de software y/o hardware para la línea de escáneres planos de HP. La motivación del proyecto fue proporcionada por el hecho de que los motores comerciales de OCR en aquella época estaban en sus inicios y fallaban en cualquier documento que no tuviera una impresión de gran calidad.
>
> (Smith [2007])

Tesseract es considerado como el mejor OCR de código abierto en la actualidad.

Cabe destacar que Pytesseract funciona con la librería «Python Imaging Library», también conocida como PIL, una librería implementada en Python que permite trabajar con imágenes aplicando diferentes operaciones sobre estas, ya sea recorte, rotación, conversión a escala de grises, simetrías y un largo etcétera. La librería PIL permite tratar con múltiples formatos de imagen, entre los que se encuentran por ejemplo PPM, PNG, JPEG, GIF, TIFF, and BMP lo cual le aporta una gran versatilidad.

2 Marco de referencia

El presente apartado tiene el objetivo de presentar los conceptos teóricos más importantes, así como su relación entre ellos, que constituyen la base de las herramientas utilizadas para la realización de este Trabajo de Fin de Grado.

El primer concepto que se debe introducir para explicar el marco de referencia utilizado es el de «clasificador».

> Un clasificador es un sistema en el que se introduce un vector de características discretas y/o continuas, y se extrae como único valor discreto, la clase.
>
> (Domingos [2012]: página 1)

Se denomina «entrenar un clasificador» a cualquier algoritmo que, dado un conjunto de vectores, junto a las clases que corresponden, devuelve un clasificador. La tarea de entrenar un clasificador se conoce como «aprendizaje supervisado». A los datos de entrada de este algoritmo, se denominan «datos de entrenamiento». Dado cualquier conjunto de vectores, siempre es posible definir un clasificador que devuelva siempre la clase que más aparece en los datos de entrenamiento. También es posible siempre devolver un clasificador aleatoriamente, sin tener en cuenta los datos de entrenamiento.

El concepto clasificador débil se otorga a aquellos clasificadores que tienen un rendimiento relativamente pobre, es decir, ligeramente por encima del rendimiento que tendría un clasificador aleatorio en media sobre un mismo conjunto.

Una vez introducido el concepto de clasificador, y más en concreto el de clasificador débil, se puede presentar al lector el algoritmo «AdaBoost». Adaboost tal y como se muestra en Wu et al. [2008]:

Algoritmo 1: Algoritmo de AdaBoost

Dado: $(x_1, y_i), ..., (x_m, y_m)$ donde $x_i \in X_i, y_i \in \{-1, +1\}$.
Algoritmo de aprendizaje base L
Número T de rondas de aprendizaje
Sea: $D_1(i) = 1/m$
for *t in 1:T* **do**
  Entrenar el clasificador débil usando la distribución $h_t = L(D, D_t)$
  Se selecciona h_t con bajo error de peso

$$\epsilon_t = Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$$

  Se elige $\alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t})$
  for *i in 1:m* **do**

$$D_{t+1}(i) = \frac{D_t(i) exp(-\alpha_t y_i h_t(x_i)))}{Z_t}$$

    dónde Z_t es un factor de normalización (elegido tal que D_{t+1} sea una distribución)
  end
end
Output: $H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$

AdaBoost acrónimo de «Adaptative Boosting», es un algoritmo desarrollado por Robert Schapire y Yoav Freund que tiene como base la combinación de varios clasificadores débiles, aportando a cada cual un peso determinado dependiendo de la fiabilidad de sus predicciones. El algoritmo Adaboost genera una serie de clasificadores a los que otorga un mayor peso dependiendo de la dificultad de su clasificación. La combinación de clasificadores débiles produce un clasificador robusto.

El segundo concepto importante que debe ser introducido al lector es el de los «árboles de decisión», el cual se irá definiendo de manera escalonada a partir de elementos más pequeños que lo conforman. Comenzamos con la definición de «árbol».

> Los grafos conexos en los cuales solo existe un camino entre cada par de vértices se conocen como árboles
>
> (Biggs et al. [1976])

A partir de la definición de árbol se puede definir «árbol con raíz»

> Los arboles con raíz son aquellos árboles con un vértice x tal que para cada vértice $x_i \neq x$ existe un único camino orientado de x a x_i
>
> (Knuth [1997])

Finalmente, un árbol de decisión es un árbol con raíz en el que cada nodo intermedio, es decir, cada nodo con aristas salientes conecta de manera directa con dos o más vértices. Aquellos vértices sin aristas salientes son denominados «hojas».

> Los árboles de decisión son estructuras de datos jerárquicos para el aprendizaje supervisado por el cual el espacio de entrada se divide en regiones locales con el fin de predecir la variable dependiente
>
> (Alpaydin [2014])

En la figura presentada a continuación se muestra un árbol de decisión sencillo:

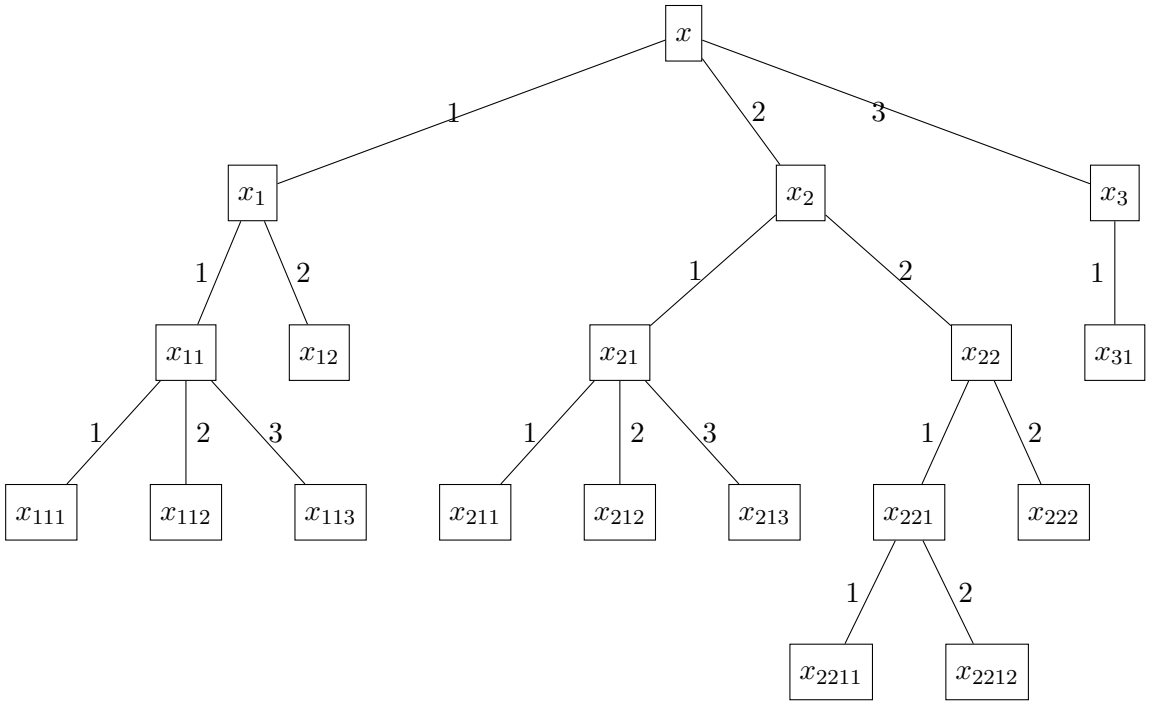


Ilustración 1: Árbol de decisión

El siguiente concepto que conforma el marco de referencia sobre el que se asienta el presente Trabajo de Fin de Grado es el de la «imagen integral». La imagen integral es una forma de representación de imágenes que se consigue aplicando el «algoritmo de la imagen integral». Este algoritmo permite ordenar la información de un área de datos de tal manera que dicha información puede ser obtenida de todas las subáreas presentes en ella con pocas y sencillas operaciones. Una imagen puede ser representada por un area de datos en el cual el dato numérico situado en la posición (i, j) del area se corresponde con el píxel (i, j) de la imagen.

Sea I el área sobre el que se quiere aplicar el algoritmo, sea (x, y) un punto cualquiera tal que $(x, y) \in I$, el valor de I en el punto (x, y) es

$$I(x, y) = \sum_{x_0 \leq x; y_0 \leq y} i(x_0, y_0)$$

donde $i(x, y)$ es el valor de (x, y). A continuación se presenta un ejemplo de la aplicación del algoritmo de la imagen integral a partir de un área de datos determinada.

1	3	7	5	9	8
4	5	6	5	1	2
7	8	1	7	5	2
3	9	1	2	8	1

Ilustración 2: Área I

1	4	11	16	25	33
5	13	26	36	46	56
12	28	42	59	74	86
15	40	55	74	97	110

Ilustración 3: Aplicación del algoritmo de imagen integral al area I

Una vez el algoritmo de la imagen integral ha sido aplicado, cualquier $I' \subset I$ puede ser computado de manera muy rápida a través del valor de los datos en sus cuatro esquinas. Sea el subárea I' de esquinas $(x_0, y_0), (x_0 + h, y_0), (x_0, y_0 + j), (x_0 + h, y_0 + j)$

$$\sum_{x_0 < x \leq x_0 + h; y_0 < y \leq y_0 + j} i'(x, y) = I'(x_0 + h, y_0) + I'(x_0, y_0 + j) - I'(x_0 + h, y_0 + j) - I'(x_0, y_0)$$

1	4	11	16	25	33
5	13	26	36	46	56
12	28	42	59	74	86
15	40	55	74	97	110

Ilustración 4: Cálculo de un área $I' \in I$

Para poner un ejemplo claro para el lector, utilizando el area I al cual se le ha aplicado el algoritmo de la imagen integral anteriormente, si se quisiera conocer el valor del área comprendida por los números en color rojo, la única operación que debería ser realizada es $I'(x, y) = 97 + 4 - 40 - 25 = 36$

El último concepto que será introducido al lector para explicar el marco de referencia será el de la «binarización», un algoritmo que permite representar imágenes estrictamente en blanco y negro. El algoritmo de binarización está definido para imágenes en escala de grises, por lo que habrá que definir inicialmente la manera de convertir imágenes en escala RGB (imágenes a color) a escala HSV (imágenes en escala de grises), ya que en el presente trabajo comenzaremos tratando con imágenes en escala RGB.
Existen diferentes procesos para binarizar una imagen, pero nos centraremos específicamente en uno, el «método del valor umbral». Este método consiste en variar el color de un pixel a negro o blanco dependiendo si la saturación «s» de dicho pixel es mayor o menor respectivamente que un parámetro fijado conocido como valor umbral. A continuación, se presenta el algoritmo del método descrito.

Algoritmo 2: Transformar una imagen de código RGB a HSV y su posterior binarización

Parte 1: Transformar imagen de código RBG a HSV
Sean:
$R, G, B \in [0, 255]$;
$Color_{RGB} = RGB(R, G, B)$;
$R' = \frac{R}{255}$;
$G' = \frac{G}{255}$;
$B' = \frac{B}{255}$;
$C_{Max} = max\{R', G', B'\}$;
$C_{Min} = min\{R', G', B'\}$;
$\Delta = C_{Max} - C_{Min}$;
if $C_{Max} = R'$ **then**
  $H = 60 * (\frac{G'-B'}{\Delta} mod 6)$;
else
  if $C_{Max} = G'$ **then**
    $H = 60 * (\frac{B'-R'}{\Delta} + 2)$;
  else
    if $C_{Max} = B'$ **then**
      $H = 60 * (\frac{R'-G'}{\Delta} + 4)$;
    else
      H=0;
    end
  end
end
if $C_{Max} = 0$ **then**
  $S = 0$;
else
  $S = \frac{\Delta}{C_{Max}}$;
end
$V = C_{Max}$;
#De los tres parámetros calculados, H de Hue, S de Saturation y V de Value, solo nos interesa el S. El parámetro saturación representa la escala de grises, con un valor entre 0 y 1#

Parte 2: Binarización
Sea $s \in [0, 1]$
if $S < s$ **then**
  $Color_{HSV} = HSV(255, 255, 255)$ **Color blanco**
else
  $Color_{HSV} = HSV(0, 0, 0)$. **Color negro**
end

Ilustración 5: Facultad de Ciencias de la Universidad de Cantabria antes y después del proceso de binarización

El método utilizado para la binarización de las imágenes en este trabajo ha sido el método «Adaptive Threshold Mean», un método en el que para cada pixel se adapta el valor umbral utilizado en el algoritmo de la binarización dependiendo de los píxeles que le rodean.

0,1	0,3	0,7
0,4	0,5	0,6
0,7	0,5	0,2

Cuadro 1: Ejemplo del valores de saturación para una serie de píxeles en una imagen

En el cuadro 1 se ofrece un pequeño ejemplo de lo explicado anteriormente. Si los valores presentados fueran las saturaciones de un bloque píxeles y se estuviese aplicando el algoritmo para thresholding adaptativo al píxel central, se aplicaría primero el algoritmo de la imagen integral:

0,1	0,4	1,1
0,5	1,3	2,6
1,2	2,5	4

Cuadro 2: Aplicación del algoritmo de imágen integral al ejemplo anterior

Y por tanto

$$S_{Bloque} = \frac{Suma\ de\ saturaciones}{Numero\ de\ pixeles} = \frac{4}{9} = 0,44$$

Y como $S_{PixelCentral} = 0,5$

$$S_{PixelCentral} > S_{Bloque}$$

El color de dicho píxel tras la binarización sería el negro. En el caso contrario, que $S_{PixelCentral} < S_{Bloque}$ el color que se aplicaría al pixel central tras la binarización sería el blanco. El cálculo de la media de las saturaciones para este algoritmo se calcula utilizando el algoritmo «summed area table» explicado anteriormente.

2.1. Convergencia de los conceptos presentados

Aunque los conceptos explicados en el presente apartado parecen inconexos, el artículo Viola y Jones [2001] propuso uno de los algoritmos más utilizados para la detección de objetos en el que se unifican los tres conceptos más interesantes que han sido presentados anteriormente: la imagen integral,

los clasificadores y los árboles de decisión. El artículo describe un método supervisado de aprendizaje es decir, que utiliza un conjunto de fotografías para entrenar al clasificador donde los objetos que se quieren detectar ya están marcados. Este método descrito en el artículo permite detectar imágenes de manera muy rápida con una muy alta tasa de acierto. Para lograr dicho objetivo, el artículo hace uso de los conceptos mencionados anteriormente de la siguiente manera:

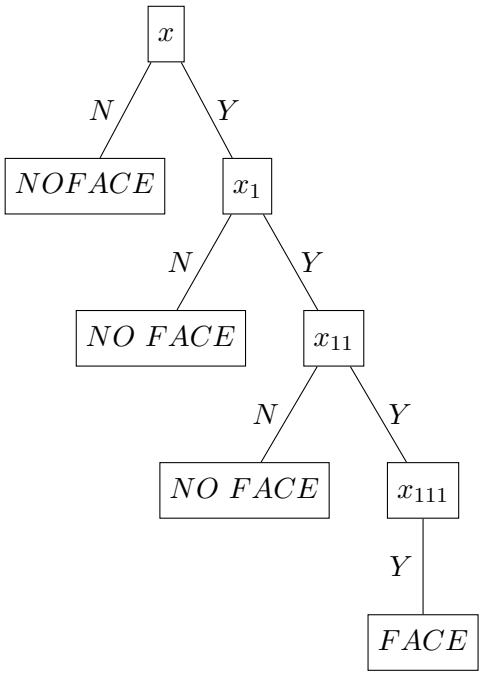
- Primero almacena la información de las imágenes procesándolas mediante el algoritmo de la imagen integral. Para ello, se apoya en el código de colores HSV, aportando a cada pixel un valor dependiendo de su color. Esto permite transformar una imagen en una matriz sobre la cual se puede obtener información con escasas operaciones por pixel.

- A través del algoritmo AdaBoost, se construye un clasificador robusto utilizando como clasificadores débiles lo denominado como «features» o «rasgos». Los rasgos pueden ser definidos como características o coincidencias que comparten todos los objetos que son buscados en las imágenes. Es decir, en caso de que el objeto a clasificar fuera una cara, los rasgos podrías ser:

  - La región de los ojos es más oscura que la región de las mejillas
  - Los ojos se encuentran encima de la nariz, que a su vez se encuentra encima de la boca
  - La nariz se localiza entre las orejas

  La unión de estos clasificadores débiles genera un clasificador robusto que puede detectar la presencia de una cara en una imagen aleatoria. También se puede entender la necesidad de aportar un peso diferente a cada uno de los clasificadores, ya que no tiene la misma importancia en la detección de un rostro la diferencia de color entre las mejillas y la nariz que el orden correcto de ojos-nariz-boca.

- Por último, se genera un tipo de árbol de decisión conocido como detector en cascada. El factor diferencial del detector en cascada es que se trata de árbol de decisión en el que los vértices que llegan de ejes «negativos» conllevan el final del árbol. Se aporta un ejemplo gráfico para la comprensión del lector.



Para entender el ejemplo, tomando una imagen inicial, se puede esquematizar de la siguiente manera

1. ¿La imagen cumple el primer clasificador débil o rasgo?
   a) Si, puede ser una cara. Será evaluada por el siguiente clasificador.
   b) No, DESCARTADA
2. ¿La imagen cumple el segundo clasificador débil o rasgo?
   a) Si, puede ser una cara. Será evaluada por el siguiente clasificador.
   b) No, DESCARTADA
3. ¿La imagen cumple el tercer y último clasificador débil o rasgo?
   a) Si, ES UNA CARA
   b) No, DESCARTADA

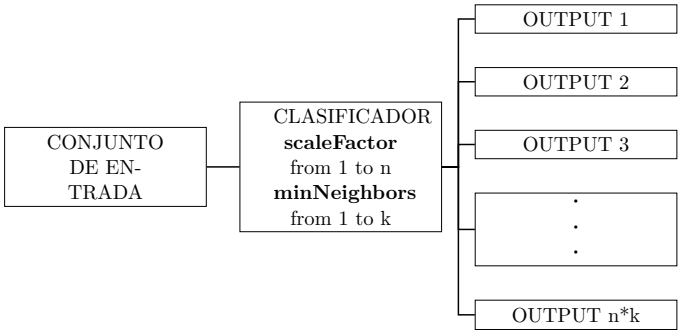
   El artículo describe la presencia de clasificadores más simples y por tanto menos pesados a la hora de ser ejecutados en las etapas iniciales del proceso, que rechazan a la extensa mayoría de imágenes, con el fin de que estas no tengan que ser evaluadas por clasificadores que requieren mayor tiempo de cómputo, es decir, son más pesados.

Aunque en el artículo Viola y Jones [2001] haga referencia a la detección de rostros en imágenes, el presente trabajo se basará en la detección de matrículas, como ha sido mencionado anteriormente. Aunque exista esta diferencia en el objeto que se desea detectar, el almacenamiento, procesado y binarizado de las imágenes seguirán los mismos principios. Lo que difiere es que el clasificador «haarcascade-russian-plate-number.xml» está específicamente diseñado para la detección de matrículas de origen ruso.

3 Trabajo y metodología

Una vez realizada la explicación del marco sobre el que se ha basado el Trabajo de Fin de Grado, así como la presentación de las herramientas utilizadas para su elaboración, el apartado actual tiene como objetivo la presentación tanto de las tareas que serán desarrolladas como de la metodología para posibilitar su realización.

Los parámetros scaleFactor y minNeighbors, presentes en el clasificador «haarcascade-russian-plate.xml» pueden ser variados por el usuario, generando cada variación un resultado diferente para un mismo conjunto de imágenes de entrada.



Es por eso, que la primera parte del presente trabajo tendrá la finalidad de realizar un estudio de sensibilidad del clasificador en base a dichos parámetros, para lo cual, se utilizará como entrada un conjunto de quince imágenes de vehículos con matrícula rusa obtenidos de Internet, las cuales, como se ha comentado en la introducción, provendrán cada una de una fuente diferente, con la diferencia de formato que este hecho conlleva. Para la selección de estas imágenes se ha intentado tomar una muestra representativa de las imágenes que se pueden generar a través de las diversas aplicaciones mencionadas en la introducción. Por ejemplo, la ilustración 6, mostrada más adelante, puede ser utilizada para aplicaciones pensadas para el control del estacionamiento de los vehículos.

El clasificador «haarcascade-russian-plate.xml» se apoyará en las imágenes imágenes utilizadas como entrada indicando sobre ellas las regiones en las que se ha detectado la presencia de una matrícula. En este caso, dichas regiones serán delimitadas con rectángulos de color azul marino.
Para aportar una idea al lector de la manera en la que procede el programa desarrollado, se recoge a continuación, una imagen de entrada y la misma imagen tras ser procesada por el clasificador.

Ilustración 6: Ejemplo de la detección de una matrícula en un vehículo

La segunda parte del trabajo depende directamente de la primera, ya que consiste en la aplicación de la librería Pytesseract al conjunto de imágenes resultantes sobre los que se haya aplicado mejor el clasificador presente en OpenCV.

En la siguiente subsección se detalla la metodología utilizada para la realización de la primera parte del trabajo, es decir la obtención de resultados en OpenCV.

3.1. Obtención de resultados en OpenCV

Para extraer los resultados retornados por el clasificador, se ha estimado mediante prueba y error, que los valores de los parámetros scaleFactor y minNeighbors que generan un mejor resultado son [1.05, 1.10, 1.15, 1.20] y [2, 3, 4] respectivamente. Los resultados obtenidos utilizando parámetros con valor diferente a los nombrados anteriormente generan unos resultados que no merece la pena analizar, ya que son muy pobres.

Como se ha comentado en el punto de herramientas utilizadas, el parámetro scaleFactor determina el tamaño que pueden tener los objetos detectados dentro de la imagen, mientras que el parámetro minNeighbors determina el número de veces que tiene que ser detectado un objeto en bloques continuos para que se acepte dicha detección.
Realizando todas las combinaciones posibles para los dos parámetros, se concluye con un total de doce situaciones a estudiar, cada una de la cuales generará un conjunto de imágenes resultante diferente. De esta manera se obtendrán finalmente doce conjuntos de resultados conteniendo quince imágenes cada uno, es decir, 180 imágenes que deberán ser analizadas a mano.

Para realizar la iteración paramétrica de forma dinámica, se ha introducido en el código del programa un bucle for, este se muestra a continuación:

```
for i in range(105,125,5):
    var1=i/100
    %%El primer parámetro varia desde 1,05 hasta 1,20 con incremento de 0.05%%
    for j in range(2,5,1):
            %%El segundo parámetro varía entre 2, 3 y 4%%
        var2=j
        newpath= os.path.join(Dir,str(var1) +"_"+ str(var2))
        if not os.path.exists(newpath):
            os.makedirs(newpath)
            %%Se crea una carpeta para cada combinacion de parametros.%%
        for file in filelist:
            Input = os.path.join(filepath,file)
            Img= cv2.imread(Input)
            gray = cv2.cvtColor(Img, cv2.COLOR_BGR2GRAY)
            face_cascade = cv2.CascadeClassifier(Class)
            faces = face_cascade.detectMultiScale(Img, var1, var2)
            for (x,y,w,h) in faces:
                cv2.rectangle(Img,(x,y),(x+w,y+h),(255,0,0),2)
                %%Se realiza la eleccion del color y el grosor del rectángulo.%%
            print("Saving : " + Input)
            cv2.imwrite(os.path.join(newpath,file),Img)%%Se guardan las imagenes%%
```

Las quince imágenes presentes en las doce carpetas generadas serán estudiadas manualmente, contando en ellas los **positivos reales, falsos positivos y falsos negativos**. Esta información será utilizada para generar la matriz de confusión de cada imagen, que sumándola a la matriz de confusión de todas las imágenes pertenecientes a su combinación de parámetros servirá para obtener los datos finales de rendimiento de los parámetros scaleFactor y minNeighbors y realizar su análisis de sensibilidad.

La matriz de confusión es una matriz 2x2 que se compone de:

$$M_{scaleFactor-minNeighbors} = \begin{pmatrix} Positivo\ real & Falso\ negativo \\ Falso\ positivo & Negativo\ real \end{pmatrix}$$

A continuación, se muestra el estudio de la imagen IMG06.jpg con la combinación de parámetros 1.05-3, así como su matriz de confusión.

Ilustración 7: Ejemplo de detección de matrículas y no matrículas en un vehículo

En la Ilustración 7 se observan:

- Un positivo real; la matrícula existente es detectada por el clasificador;
- Tres falsos positivos; tres «no matrículas» detectados como matrículas;
- Ningún falso negativo; no aparecen matrículas no detectadas por el clasificador.

Por tanto, la matriz de confusión de la imagen superior es:

$$M_{1,05-3} = \begin{pmatrix} Positivo\ real\ = 1 & Falso\ negativo\ = 0 \\ Falso\ positivo\ = 3 & Negativo\ real\ = \infty \end{pmatrix}$$

Donde el subíndice 1.05-3 se corresponde a los valores de scaleFactor y minNeighbors.

Cabe destacar que el número de «no matrículas» no detectado como matrículas se puede considerar como infinito, tanto en esta, como en todas las imágenes del estudio, por lo tanto, dicho dato no tendrá relevancia sobre el análisis de sensibilidad.

3.2. Análisis de los resultados obtenidos de OpenCV y aplicación de Pytesseract

El presente subapartado tiene como objetivo describir la metodología que será utilizada en la segunda parte del trabajo, que se basa en la aplicación de Pytesseract al mejor conjunto de imágenes procesado con la combinación óptima de parámetros para el clasificador «haarcascade-russian-plate.xml» presente en OpenCV.

Lo primero que se hará será un análisis de los resultados obtenidos en OpenCV:

- **Análisis de los conjuntos de resultados retornados por el clasificador:** Una vez obtenidas las matrices de confusión de los doce casos estudiados, uno por cada una de las combinaciones posibles realizadas con los valores estimados para los parámetros scaleFactor y minNeighbors, se procederá al análisis de resultados obtenidos, para lo cual se tendrá en cuenta tanto el ratio aciertos entre el total de matrículas como el número de falsos positivos.

Una vez analizados los mejores resultados, se les aplicará la librería Pytesseract realizando también un análisis de sensibilidad según los parámetros usados para su binarización:

- **Análisis de sensibilidad para medir la efectividad de la librería Pytesseract**: Al conjunto de imágenes resultantes sobre el que se haya implementado el clasificador de manera óptima se le aplicará la librería Pytesseract.
  Para el correcto funcionamiento de esta librería las imágenes deberán ser binarizadas siguiendo el método «Adaptive Threshold Mean», el cual puede ser calibrado variando los parámetros Blockxblock y Resto. Por ese motivo se realizará un análisis de la sensibilidad de la librería Pytesseract ante la variación de los parámetros anteriormente mencionados, y se encontrarán los valores para Blockxblock y Resto que generan el conjunto de imágenes sobre el cual se aplica Pytesseract de manera óptima.

4 Resultados y Análisis

4.1. Resultados Obtenidos

Los resultados obtenidos para los valores de los parámetros scaleFactor=[1.05, 1.10, 1.15, 1.20] y minNeighbors=[2, 3, 4] se recogen a continuación, así como una selección aleatoria de imágenes tras ser tratadas por el clasificador. Se puede observar que el número de positivos reales oscila entre 9 y 14, los falsos positivos oscilan entre 49 y 7, y los falsos negativos entre 1 y 4.

$$M_{Total} = \left(\begin{array}{l} M_{1,05-2} = \begin{pmatrix} 13 & 2 \\ 49 & \infty \end{pmatrix} M_{1,05-3} = \begin{pmatrix} 13 & 2 \\ 36 & \infty \end{pmatrix} M_{1,05-4} = \begin{pmatrix} 12 & 3 \\ 25 & \infty \end{pmatrix} \\ \\ M_{1,10-2} = \begin{pmatrix} 14 & 1 \\ 26 & \infty \end{pmatrix} M_{1,10-3} = \begin{pmatrix} 12 & 3 \\ 20 & \infty \end{pmatrix} M_{1,10-4} = \begin{pmatrix} 12 & 3 \\ 14 & \infty \end{pmatrix} \\ \\ M_{1,15-2} = \begin{pmatrix} 12 & 3 \\ 16 & \infty \end{pmatrix} M_{1,15-3} = \begin{pmatrix} 12 & 3 \\ 9 & \infty \end{pmatrix} M_{1,15-4} = \begin{pmatrix} 12 & 3 \\ 7 & \infty \end{pmatrix} \\ \\ M_{1,20-2} = \begin{pmatrix} 12 & 3 \\ 11 & \infty \end{pmatrix} M_{1,20-3} = \begin{pmatrix} 12 & 3 \\ 9 & \infty \end{pmatrix} M_{1,20-4} = \begin{pmatrix} 9 & 4 \\ 9 & \infty \end{pmatrix} \end{array} \right)$$

Ilustración 8: Muestra aleatoria de vehículos tratados

4.2. Análisis de los conjuntos de resultados retornados por el clasificador

La presente sección tiene el objetivo de analizar los resultados obtenidos y estimar con qué combinación de parámetros ha sido más eficiente el clasificador. Lo primero es presentar los datos.

scaleFactor	min Neighbors	**Caso**	Positivo Real	Falso Positivo	Falso Negativo	«Rate»
1.05	2	1.05-2	13	49	2	**86,67 %**
1.05	3	1.05-3	13	36	2	**86,67 %**
1.05	4	1.05-4	12	25	3	**80,00 %**
1.10	2	1.10-2	14	26	1	**93,33 %**
1.10	3	1.10-3	12	20	3	**80,00 %**
1.10	4	1.10-4	12	14	3	**80,00 %**
1.15	2	1.15-2	12	16	3	**80,00 %**
1.15	3	1.15-3	12	9	3	**80,00 %**
1.15	4	1.15-4	12	7	3	**80,00 %**
1.20	2	1.20-2	12	11	3	**80,00 %**
1.20	3	1.20-3	12	9	3	**80,00 %**
1.20	4	1.20-4	9	9	4	**69,23 %**

Cuadro 3: Resultados obtenidos tras aplicar el clasificador en cada uno de los casos

El «rate» que aparece en el Cuadro 3 se ha calculado siguiendo la fórmula

$$Rate = \frac{PositivoReal}{DeteccionesTotales} * 100$$

Siendo:

$$DeteccionesTotales = PositivosReales + FalsoNegativo$$

Para lograr un análisis en profundidad de los resultados obtenidos vamos a tener en cuenta todos los valores que componen la matriz de confusión para cada caso, es decir, positivos reales, falsos positivos y falsos negativos. Los positivos reales y los falsos negativos se combinan en el «rate», de modo que estudiando el «rate» para cada caso estaremos teniendo en cuenta ambos factores, mientras que el número de falsos positivos se estudiará por su cuenta.

«Rate»

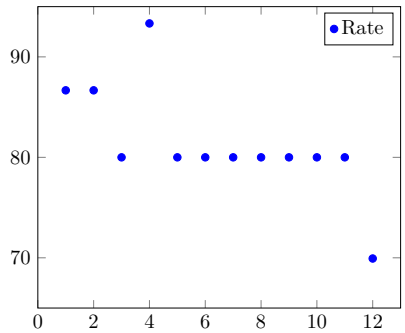


Ilustración 9: «Rate» para todos los casos

En la ilustración 9 se ha representado el «rate» para los doce resultados obtenidos, y se ha observado que, exceptuando los casos 1.10- 2 y 1.20-4, que ocupan extremos opuestos de la distribución, el «rate» para los casos toma valores ciertamente parecidos, oscilando diez de las doce casuísticas entre 80 y 87 puntos porcentuales.
Para apoyar este análisis se calcula la desviación típica del conjunto de «rates» siguiendo la fórmula

$$\sigma = \sqrt{(\frac{\sum_{k=1}^{n}(x_k - \mu)^2}{n}} = \sqrt{(\frac{\sum_{k=1}^{12}(Rate_k - 81,23\,\%)^2}{12}} = 5,49\,\%$$

La desviación típica indica lo que tienden los datos a desviarse de la media, que en el caso de los «rates» es 81,23 %, por lo tanto, nuestros datos tienden a estar dispersos en el intervalo [75,84 %, 86,81 %]. Como se ha apuntado anteriormente, solo dos de los doce casos se encuentran fuera del intervalo; el caso 1.10-2, con un «rate» de 93,33 %, y el caso 1.20-4, con un «rate» de 69,23 %. Estos resultados se muestran en la ilustración 10.

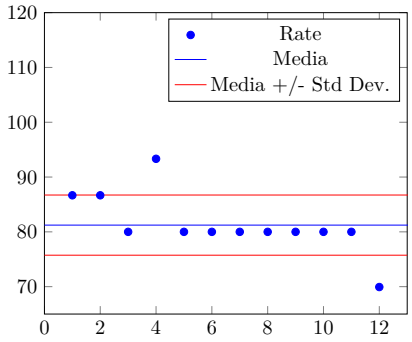


Ilustración 10: Media del «rate» y su desviación típica

Falsos Positivos
El número de falsos positivos determina la cantidad de elementos han sido detectados como matrículas no siéndolo. Es un factor importante a tener en cuenta, ya que, aunque un mayor número de falsos positivos puede traer consigo un mayor número de detecciones correctas, ya que el clasificador aplica un filtro más liviano al conjunto de imágenes, también trae consigo un aumento en el tiempo de procesamiento de las imágenes.
En el extremo opuesto, un número muy bajo de falsos positivos implica que el clasificador ha impuesto filtros muy estrictos por los cuales existirán matriculas que hayan sido no detectadas y por consiguiente descienda su «rate». A continuación, se muestra la gráfica que recoge los datos de falsos positivos:

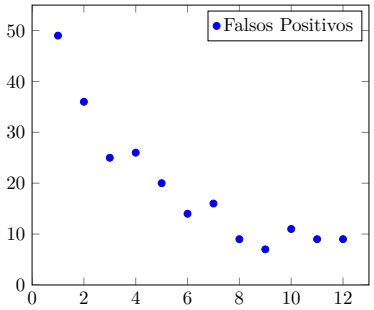


Ilustración 11: Número de Falsos Positivos para todos los casos

En la Ilustración 11 se puede observar que es número de falsos positivos tiene su máximo con el primero de los casos, el 1.20-4 y decrece exponencialmente hasta el menor valor, siete, correspondiente a 1.15-4.
Al igual que para el caso del «rate», se ha calculado la desviación típica del conjunto de datos de los falsos positivos.

$$\sigma = \sqrt{(\frac{\sum_{k=1}^{n}(x_k - \mu)^2}{n}} = \sqrt{(\frac{\sum_{k=1}^{12}(FP_k - 12,25)^2}{12}} = 12,30$$

La media de los falsos positivos es 19,25, por lo que el intervalo en el que tienden a estar dispersos los datos es [6,94 , 31,55]. En este caso, a diferencia de en el caso de los «rates», el intervalo es muy extenso, ya que los datos están más dispersos. A continuación se muestra gráficamente

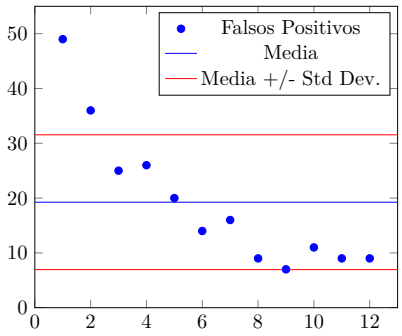


Ilustración 12: Media del número de Falsos Positivos y su desviación típica

4.3. Selección de imágenes a procesar

Es evidente que la situación óptima pasa por una combinación de parámetros por la cual el clasificador no detecte ningún falso positivo y a la vez tenga un «rate» cercano al 100 %, pero como esa situación no es posible, es imprescindible otorgar un peso determinado a cada factor medido a la hora de realizar la elección sobre que parámetros han logrado la mejor configuración del clasificador.
Para seleccionar el conjunto de imágenes sobre el cual se ha aplicado el clasificador de forma óptima se ha desarrollado un análisis «multicriterio» que puntuará los resultados de la matriz de confusión de cada conjunto en función del «rate» y del número de falsos positivos. La manera en la que puntuará cada conjunto de resultados se recoge a continuación:

- El conjunto de resultados con un «rate» más alto $Rate_{MAX}$ obtendrá cien puntos. Las puntuaciones de los «rates» restantes $Rate_i$ será calculada a partir de la puntuación más alta siguiendo la fórmula:

$$RATE\ SCORE_i = \frac{Rate_i * 100}{Rate_{MAX}}$$

  Las puntuaciones obtenidas son las siguientes

scaleFactor	min Neighbors	**Caso**	«Rate»	Puntuación «Rate»
1.05	2	1.05-2	**86,67 %**	92,9
1.05	3	1.05-3	**86,67 %**	92,9
1.05	4	1.05-4	**80,00 %**	85,7
1.10	2	1.10-2	**93,33 %**	100
1.10	3	1.10-3	**80,00 %**	85,7
1.10	4	1.10-4	**80,00 %**	85,7
1.15	2	1.15-2	**80,00 %**	85,7
1.15	3	1.15-3	**80,00 %**	85,7
1.15	4	1.15-4	**80,00 %**	85,7
1.20	2	1.20-2	**80,00 %**	85,7
1.20	3	1.20-3	**80,00 %**	85,7
1.20	4	1.20-4	**69,23 %**	74,2

Cuadro 4: Puntuación «Rate»

- El conjunto de resultados con menor número de falsos positivos FP_{MAX} obtendrá la peor puntuación, a partir de esta se escalará la puntuación del resto de conjuntos FP_i siguiendo la fórmula

$$FP\ SCORE_i = \frac{FP_i * 100}{FP_{MAX}}$$

Las puntuaciones obtenidas para el número de falsos positivos de cada caso son:

scaleFactor	min Neighbors	**Caso**	FP	Puntuación FP
1.05	2	1.05-2	49	0,00
1.05	3	1.05-3	36	26,5
1.05	4	1.05-4	25	49,0
1.10	2	1.10-2	26	46,9
1.10	3	1.10-3	20	59,2
1.10	4	1.10-4	14	71,4
1.15	2	1.15-2	16	67,3
1.15	3	1.15-3	9	81,6
1.15	4	1.15-4	7	85,7
1.20	2	1.20-2	11	77,6
1.20	3	1.20-3	9	81,6
1.20	4	1.20-4	9	81,6

Cuadro 5: Puntuación Falsos Positivos

Como se ha mencionado anteriormente, el número de falsos positivos detectados por el clasificador y la puntuación otorgada en este ámbito son inversamente proporcionales. Se ha elaborado una tabla que agrupa ambos valores para cada caso

- Finalmente, se ha otorgado a cada uno de los criterios un peso dependiendo de la importancia que deberían tener a la hora de elegir el conjunto de resultados sobre el que se ha aplicado el clasificador de manera óptima.
  La selección de los pesos se ha hecho teniendo en cuenta la mayor importancia que tiene la detección de positivos reales, ligado al criterio «rate», sobre la detección de falsos positivos, ya que, como se verá más adelante, los falsos positivos serán eliminados a mano a la hora de realizar el análisis de Pytesseract. **El peso considerado para el criterio «rate» es de 0.75, y para el criterio del número de falsos positivos es de 0.25.**

Las puntuaciones finales obtenidas por cada caso se presentan a continuación:

Caso	Puntuación «Rate»	Puntuación FP	Puntuación Total
1.05-2	92,9	0,0	69,6
1.05-3	92,9	26,5	76,3
1.05-4	85,7	49,0	76,5
1.10-2	100,0	46,9	86,7
1.10-3	85,7	59,2	79,1
1.10-4	85,7	71,4	82,1
1.15-2	85,7	67,3	81,1
1.15-3	85,7	81,6	84,7
1.15-4	85,7	85,7	85,7
1.20-2	85,7	77,6	83,7
1.20-3	85,7	81,6	84,7
1.20-4	74,2	81,6	76,0

Cuadro 6: Puntuación final

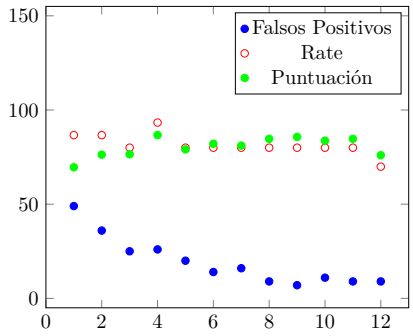


Ilustración 13: Representación gráfica de la puntuación final

Como conclusión al análisis de «haarcascade-russian-plate.xml» se ha establecido que la combinación de parámetros que permite aplicar el clasificador de mejor manera al conjunto inicial de imágenes ha sido scaleFactor = 1.10 y minNeighbors = 2.

4.4. Análisis de sensibilidad para medir la efectividad de la librería Pytesseract

Una vez que ha sido concretado que se realizará el análisis de sensibilidad de la librería Pytesseract sobre las imágenes generadas por el clasificador con la elección de parámetros scalefactor = 1.10 y minNeighbors = 2, se deberán comenzar a tratar las imágenes de salida para posibilitar su trabajo con la librería. Para recordar al lector, la matriz de confusión del caso que estamos tratando es

$$M_{1,10-2} = \begin{pmatrix} Postivo\ real\ = 14 & Falso\ negativo\ = 1 \\ Falso\ postivo\ = 26 & Negativo\ real\ = \infty \end{pmatrix}$$

Lo primero que se realizará será guardar cada una de las regiones detectadas por el clasificador en el apartado anterior como una imagen nueva. Teniendo en cuenta que el caso 1.10-2 tiene 14 positivos reales y 26 falsos negativos, se extraerán un total de 40 imágenes nuevas de las cuales 14 corresponderán a matrículas y las restantes 26 deberán borrarse a mano. Para lograr extraer imágenes siguiendo la forma de los rectángulos dictaminada por el clasificador se aplica el siguiente código.

```
for fichero in filelist:
    Input = os.path.join(filepath,fichero)
    Img= cv2.imread(Input)
    face_cascade = cv2.CascadeClassifier(Class)
    faces = face_cascade.detectMultiScale(Img, var1, var2)
    for (x,y,w,h) in faces:
        cv2.imwrite(os.path.join(newpath, fichero+"_"+ str(x) +"_"+ str(y)+
        "_"str(w)+"_"+str(h) + '.jpg'),Img[y:y+h,x:x+w])
        %%Aqui se guarda una imagen con las coordenadas (x,y,h,w) de cada
        réctangulo%%
```

Una vez realizada esta operación, y tras borrar las «no matrículas», se han obtenido 14 imágenes de matrículas con esta forma.

Ilustración 14: Selección de matrículas recortadas

Para posibilitar el reconocimiento de caracteres en imágenes, es preferible utilizar una imagen binarizada, es por eso que el análisis de Tesseract se realizará ligado a los parámetros escogidos a la hora de binarizar la imagen. Como se vio en el capítulo 2, existe un método llamado «Adaptive Threshold Mean» que utiliza el valor medio de la saturación de un bloque de píxeles para determinar si el píxel situado en el centro del bloque torna blanco o en su defecto negro en el proceso de binarización. Los parámetros que varían la efectividad de la binarización de la imagen son dos:

- BlockxBlock: Parámetro que determina el tamaño del bloque del que se calculará el valor medio. Siempre son valores impares. El valor del ejemplo de la Ilustración 1 es 3
- Resto: Valor que se le resta a la media de la saturación del bloque

A continuación se muestra el código en el que se varían dichos parámetros de manera dinámica en Python:

```
for i in range(2,9,1):
    for j in range(2,5,1):
        res=j;tam=(2*i)+1
        for fichero in filelist:
            Input = os.path.join(filepath,fichero)
            Img= cv2.imread(Input)
            gray = cv2.cvtColor(Img, cv2.COLOR_BGR2GRAY)
            # Hacemos threshold adaptativo
            th2 = cv2.adaptiveThreshold(gray,255,cv2.ADAPTIVE_THRESH_MEAN_C,\
                cv2.THRESH_BINARY,tam,res)
            # Aplicamos un filtro para eliminar ruido.
            th2=cv2.medianBlur(th2,3)
            cv2.imwrite(os.path.join(newpath,str(tam) + fichero),th2)
```

Como se ha podido ver en el código mostrado anteriormente, el parámetro BlockxBlock toma los valores [5, 7, 9, 11, 13, 15, 17] y el parámetro Resto toma los valores [2, 3, 4]. Para cada combinación de estos parámetros se ha generado un conjunto de imágenes resultante al cual se le ha aplicado posteriormente Tesseract, es decir, se ha aplicado el OCR a un total de 21 escenarios diferentes conteniendo cada uno 14 imágenes, lo cual hace un total de 294 imágenes procesadas y analizadas.

Para determinar la efectividad de la aplicación de la librería Tesseract, nos basaremos en el número total de caracteres acertados por cada conjunto de resultados. Como se puede observar en las imágenes a lo largo del presente trabajo, las matrículas rusas se componen de tres letras y cinco o seis números (dependiendo de la matrícula) siguiendo el orden [letra, número, número, número, letra, letra, número, número, número]. Para medir la efectividad de la librería en cada matrícula se ha seguido un código binario;

$$Punt_i = \begin{cases} 1, & \text{if } Caracter\ Retornado_i = Caracter\ Correcto_i \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

$$Punt_{TOTAL} = \sum_{i=1}^{n} Punt_i$$

A continuación se presenta una matrícula aleatoria perteneciente al conjunto de imágenes que se ha utilizado para trabajar en Pytesseract con la finalidad de explicar al lector de manera más detallada la metodología seguida para la puntuación.

Ilustración 15: Ejemplo de matrícula rusa

En el caso de la matrícula recogida en la Ilustración 15 tuviera $Output = x515xx73$ la puntuación obtenida sería $Punt_{Total} = 1+1+0+1+1+1+1+0 = 6$

La puntuación obtenida por cada matrícula se sumará, pudiendo puntuar cada escenario un máximo de 115 en el mejor de los casos, y 0 puntos en el caso en el que Pytesseract no haya detectado ningún caracter de ninguna de las 14 matrículas. Las puntuaciones obtenidas en cada escenario se recogen en el cuadro 7, mostrado a continuación.

Escenario	Número de bloques	Resto	Puntuación	Porcentaje de acierto
1	5	2	0	0 %
2	5	3	2	1.74 %
3	5	4	6	5.22 %
4	7	2	9	7.83 %
5	7	3	8	6.96 %
6	7	4	7	6.09 %
7	9	2	7	6.09 %
8	9	3	8	6.96 %
9	9	4	12	10.43 %
10	11	2	6	5.22 %
11	11	3	8	6.96 %
12	11	4	5	4.35 %
13	13	2	4	3.48 %
14	13	3	3	2.61 %
15	13	4	3	2.61 %
16	15	2	5	4.35 %
17	15	3	2	1.74 %
18	15	4	6	5.22 %
19	17	2	2	1.74 %
20	17	3	4	3.48 %
21	17	4	5	4.35 %

Cuadro 7: Puntuación obtenida para cada escenario en Pytesseract.

Las puntuaciones presentadas en el cuadro 7 son representadas gráficamente en la tabla que se muestra a continuación.

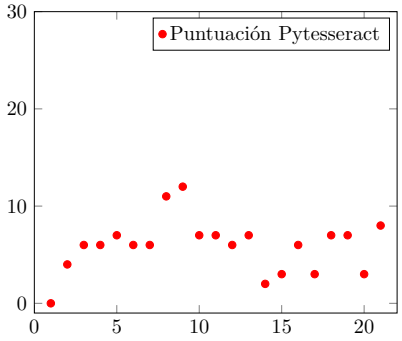


Ilustración 16: Puntuación obtenida para cada escenario en Pytesseract

Como se puede observar tanto en la tabla como en las gráficas mostradas superiormente, el número de aciertos no es muy elevado, llegando como máximo a 12 de 115 puntos posibles para el caso de Bloques=9 y Resto=4, lo que corresponde a un 10.43 % de caracteres acertados sobre el total.

Este bajo acierto por parte de la librería Pytesseract se puede explicar por una serie de factores que se enumeran a continuación:

- La binarización en las imágenes genera ruido. Parte de ese ruido puede ser eliminado utilizando un filtro, pero no se asegura que la totalidad del ruido desaparezca.

- Los caracteres se pueden deformar por la presencia de ruido no eliminado. Esto provoca que la detección de los caracteres sea más complicado.
- La librería Pytesseract está entrenada para la detección de diferentes fuentes tipográficas, y parece que no detecta bien la fuente utilizada en las matrículas rusas.

A continuación se recogen una selección de imágenes tras binarización que permiten ilustrar los problemas que se mencionan anteriormente.

Ilustración 17: Selección de imágenes de matrículas rusas tras binarización

Teniendo en cuenta los problemas mencionados anteriormente que han generado la baja funcionalidad de Pytesseract en las matriculas de origen ruso, cabe decir que la matrícula sobre la que más acierto se ha generado ha sido la que se muestra a continuación:

Ilustración 18: Matrícula con mayor nivel de acierto en cada caso

En el caso Blockxblock=9 y Resto=4 la imagen tras binarización tiene el siguiente aspecto:

Ilustración 19: Matrícula anterior tras binarización

El resultado que ha retornado Pytesseract para 19 ha sido:

$$EA098ABE\S°1$$

5 Resumen y conclusiones

5.1. Resumen

En el el presente trabajo se ha tratado de conducir al lector desde las nociones más básicas que constituyen los algoritmos desarrollados para el reconocimiento de objetos, hasta su posterior aplicación al reconocimiento de matriculas en imágenes y los resultados sobre un conjunto de datos específico.

Para ello ha sido presentado el marco de referencia sobre el cual se cimienta el estudio realizado, introduciendo conceptos más básicos como clasificador, los árboles de decisión, las «summed area tables» o la binarización a la que se tiene que someter una imagen para facilitar la lectura de los caracteres presentes en ella.

Tras introducir al lector en dichos conceptos, se ha realizado un análisis de la sensibilidad del clasificador «haarcascade-russian-plate.xml» que encuentra la región rectangular de una imágen en la que se detecta una matrícula, encontrando los parámetros que consigan detectar dichas matrículas de manera óptima, que han sido scaleFactor=1.10 y minNeighbors=2. Para concluir, se ha aplicando la librería Pytesseract al conjunto de imágenes resultantes procesados de manera ótima por «haarcascade-russian-plate.xml» y se ha realizado un análisis de la sensibilidad de dicha librería ante la variación de los parámetros Blockxblock y Resto.

5.2. Conclusiones

Sobre la base de todo el trabajo desarrollado a lo largo de este estudio, podemos decir que los resultados obtenidos muestran una efectividad del 93.33 % en el mejor de los casos para la librería OpenCV, mientras que solo del 10.43 % para la librería Pytesseract.

Si bien estos resultados muestran una gran diferencia entre el rendimiento de ambas librerías, la explicación puede ser sencilla. Mientras que la librería OpenCV posee un clasificador específicamente diseñado para la tarea que le ha sido encomendada en el presente trabajo, el motor de reconocimiento de caracteres no ha sido entrenado específicamente para la detección de estos en matrículas de origen ruso, por lo que una posible solución para su bajo rendimiento podría ser el entrenamiento de dicho motor con la fuente de los caracteres presentes en las matrículas procesadas.

Esta gran diferencia entre la efectividad de las librerías demuestra la importancia que tienen los procesos supervisados de aprendizaje a los que ha sido sometido el clasificador «haarcascades-russian-plate-number.xml» a la hora de la detección de objetos.

Por ultimo, cabe destacar que el análisis de ambas librerías se ha tenido que hacer a mano, revisando si las regiones estimadas como matrículas eran correctas en el caso de OpenCV, y comparando los resultados generados por Pytesseract con cada matrícula para todos los casos. El hecho de tener que comprobar de manera manual todos los resultados limita el tamaño del conjunto de imágenes de entrada que puede ser utilizado.

A Código Utilizado

En el presente apéndice se recogerá el código utilizado. Se pueden diferenciar tres códigos que han sido implementados en la realización del presente Trabajo de Fin de Grado, y son:

- Código para implementar el clasificador en cascada
- Código para recortar las imágenes
- Código para realizar la binarización de las imágenes y aplicar Pytesseract

Código para implementar el clasificador en cascada

```
import os.path
import cv2
import sys,os
from PIL import Image, ImageOps, ImageFont, ImageDraw
Dir = os.path.join('C:\\','Users','Rodrigo','Desktop','TFG_V02','Fotos')
filelist= [fichero for fichero in os.listdir(Dir) if fichero.endswith('.jpg')]
filepath = os.path.join(os.path.dirname(sys.argv[0]),Dir)
Dir3 = os.path.join('/','Users','Rodrigo','Anaconda3','pkgs',

'opencv3-3.1.0-py35_0','Library','etc','haarcascades')
Filename = 'haarcascade_russian_plate_number.xml'
Class = os.path.join(Dir3,Filename)
for i in range(105,125,5):
    var1=i/100
    print (var1)
    for j in range(2,5,1):
        var2=j
        print (var2)
        newpath= os.path.join(Dir,str(var1) +"_"+ str(var2))
        if not os.path.exists(newpath):
            os.makedirs(newpath)
        for file in filelist:
            Input = os.path.join(filepath,file)
            Img= cv2.imread(Input)
            gray = cv2.cvtColor(Img, cv2.COLOR_BGR2GRAY)
            face_cascade = cv2.CascadeClassifier(Class)
            faces = face_cascade.detectMultiScale(Img, var1, var2)
            for (x,y,w,h) in faces:
                cv2.rectangle(Img,(x,y),(x+w,y+h),(255,0,0),2)
            print("Saving : " + Input)
            cv2.imwrite(os.path.join(newpath,file),Img)
```

Código para recortar las imágenes

```
import os.path
import numpy as np
import cv2
import sys,os
from PIL import Image, ImageOps, ImageFont, ImageDraw
Dir = os.path.join('C:\\','Users','Rodrigo','Desktop','TFG_V02','Fotos')
filelist= [fichero for fichero in os.listdir(Dir) if fichero.endswith('.jpg')]
filepath = os.path.join(os.path.dirname(sys.argv[0]),Dir)
Dir3 = os.path.join('/','Users','Rodrigo','Anaconda3','pkgs',
'opencv3-3.1.0-py35_0','Library','etc','haarcascades')
Filename = 'haarcascade_russian_plate_number.xml'
Class = os.path.join(Dir3,Filename)
var1=1.1; var=2
newpath= os.path.join(Dir,str(var1) +"_"+ str(var2)+"Tesseract")
if not os.path.exists(newpath):
    os.makedirs(newpath)
for fichero in filelist:
    Input = os.path.join(filepath,fichero)
    Img= cv2.imread(Input)
    face_cascade = cv2.CascadeClassifier(Class)
    faces = face_cascade.detectMultiScale(Img, var1, var2)
    for (x,y,w,h) in faces:
        cv2.imwrite(os.path.join(newpath, fichero +"_" + str(x)
        +"_"+ str(y)+ "_"+ str(w)+"_"+ str(h) +
        '.jpg'),Img[y:y+h,x:x+w])
```

Código para realizar la binarización de las imágenes y aplicar Pytesseract

```
import os.path
import numpy as np
import cv2
import PIL
import sys,os
from PIL import Image, ImageOps, ImageFont, ImageDraw
import pytesseract
Dir = os.path.join('C:\\','Users','Rodrigo','Desktop','TFG_V02','Fotos',
'1.1_2Tesseract')
tessdata_dir_config =""
filelist= [fichero for fichero in os.listdir(Dir) if fichero.endswith('.jpg')]
filepath = os.path.join(os.path.dirname(sys.argv[0]),Dir)
```

```
for i in range(2,9,1):
    for j in range(2,5,1):
        res=j; tam=(2*i)+1
        newpath= os.path.join(Dir,str(tam)+"_"+str(res))
        if not os.path.exists(newpath):
            os.makedirs(newpath)
        for fichero in filelist:
            Input = os.path.join(filepath,fichero)
            Img= cv2.imread(Input)
            gray = cv2.cvtColor(Img, cv2.COLOR_BGR2GRAY)
            # Hacemos threshold adaptativo
            th2 = cv2.adaptiveThreshold(gray,255,cv2.ADAPTIVE_THRESH_MEAN_C,\
                cv2.THRESH_BINARY,tam,res)
            # Aplicamos un filtro para eliminar ruido.
            th2=cv2.medianBlur(th2,3)
            cv2.imwrite(os.path.join(newpath,str(tam) + fichero),th2)
#RESIZE FOR TESS
for i in range(2,9,1):
    for j in range(2,5,1):
        res=j; tam=(2*i)+1;
        DirResize=os.path.join(Dir,str(tam)+"_"+str(res))
        filelistResize= [fichero for fichero in os.listdir(DirResize) if
        fichero.endswith('.jpg')]
        filepathResize=os.path.join(os.path.dirname(sys.argv[0]),DirResize)
        for fichero in filelistResize:
            InputII=os.path.join(filepathResize,fichero)
            ImgII=Image.open(InputII)
            basewidth=640
            wpercent=(basewidth/float(ImgII.size[0]))
            hsize= int((float(ImgII.size[1])*float(wpercent)))
            ImgII= ImgII.resize((basewidth,hsize),PIL.Image.ANTIALIAS)
            ImgII.save(os.path.join(DirResize,fichero),dpi=(300,300))
#TESS
for i in range(2,9,1):
    for j in range(2,5,1):
        res=j; tam=(2*i)+1
        print(tam)
        print(res)
        DirResize=os.path.join(Dir,str(tam)+"_"+str(res))
        filelistResize= [fichero for fichero in os.listdir(DirResize) if
        fichero.endswith('.jpg')]
        filepathResize=os.path.join(os.path.dirname(sys.argv[0]),DirResize)
        for fichero in filelistResize:
            #print(fichero)
            InputIII=os.path.join(filepathResize,fichero)
            ImgIII=Image.open(InputIII)
            print(fichero,": ",pytesseract.image_to_string(ImgIII,config='-psm 8 '
            +tessdata_dir_config))
```

Referencias

E. Alpaydin. *Introduction to machine learning*. MIT press, 2014.

N. Biggs, E. K. Lloyd y R. J. Wilson. *Graph Theory, 1736-1936*. Oxford University Press, 1976.

P. Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55 (10):78–87, 2012.

L. Eikvil. Optical character recognition. 1993.

R. C. Gonzalez. Digital image processing, 2016.

D. E. Knuth. *The art of computer programming*, volume 3. Pearson Education, 1997.

OpenCV. Description, 2018.
https://opencv.org/

C. P. Papageorgiou, M. Oren y T. Poggio. A general framework for object detection. In *Computer vision, 1998. sixth international conference on*, pages 555–562. IEEE, 1998.

X. S Yu, Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, y Philip. Top 10 algorithms in data mining. *Knowledge and information systems*, 14(1): 1–37, 2008.

R. E. Schapire. Explaining adaboost. In *Empirical inference*, pages 37–52. Springer, 2013.

R. Smith. An overview of the tesseract ocr engine. In *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, volume 2, pages 629–633. IEEE, 2007.

P. Viola y M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE, 2001.