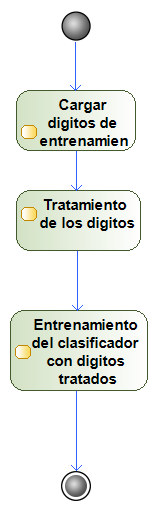
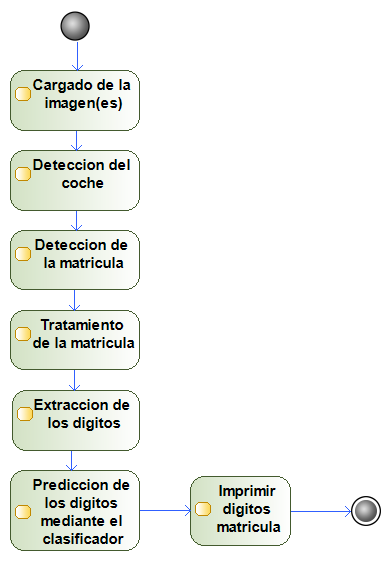
Practica 2: Segmentacion y reconocimiento de caracteres

Agustín Daniel Schüler Allub y Patricia de Gregorio Ruiz

# Introducción

Como introducción, hemos realizado dos diagramas UML correspondientes al entrenamiento del sistema, para después testearlo.



Como es obvio, el sistema necesita ser entrenado con una serie de dígitos de entrenamiento antes de ser testeado. Después, se le proporciona una imagen, donde puede haber el numero de coches (y por tanto el numero de matrículas) que sean. El sistema puede detectar dichos coches y matriculas y devolverte el numero de la matrícula. Cabe destacar que depende muchísimo de la calidad de la imagen, a más resolución es mucho más fácil que el sistema reconozca las matrículas. En realidad, a pesar de que hemos realizado todos los puntos que propone la práctica, estará todo integrado en un solo sistema de detectar una serie de objetos secuencialmente (coche -> matricula -> dígitos de la matricula) así que resulta complicado detallar qué se ha realizado en cada punto exactamente. A continuación, procedemos a explicar más detalladamente el código desarrollado para la realización de la práctica.

# Explicación detallada y justificación del código

Para que se más cómodo la explicación del código desarrollado hemos decidido separar la explicación por clases y por la clase principal.

* Leer\_matricula.py (Clase principal)

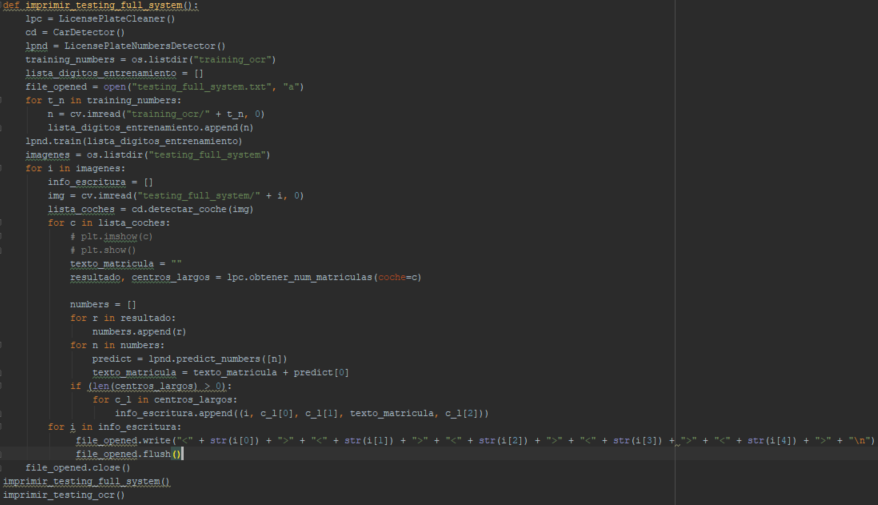
La clase principal consta de dos métodos que se encargan de entrenar al sistema, para después cargar las imágenes, detectar los coches y obtener los dígitos de la matrícula, para posteriormente preguntarle al sistema de que digito se trata.



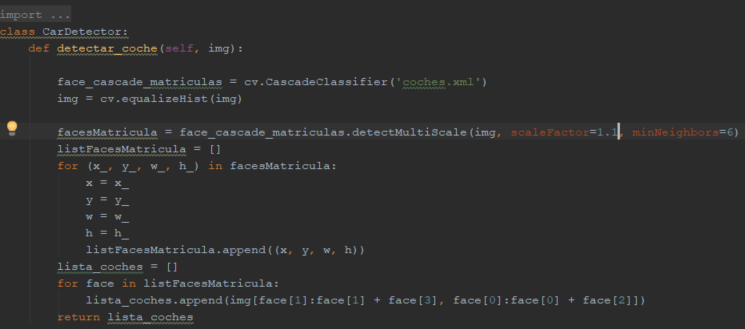
En este caso tenemos la primera función, que se encarga de cargar las imágenes del directorio “testing\_ocr”. En el caso de “testing\_ocr”, hemos decidido no detectar el coche antes de detectar la matricula. La razón es sencilla: Sabemos que los coches están centrados y se muestran frontalmente, hemos comprobado que utilizar la detección del coche hace que la detección empeore, así que directamente le decimos que detecte la matricula. En las primeras líneas, podemos ver nos creamos diferentes objetos que nos ayudaran a la realización del objetivo:

* Objeto “lpc”: Objeto de la clase LicensePlateCleaner. Este objeto se encarga de recibir una matrícula, realizar un procesado de la matrícula, obtener los dígitos y tratarlos para devolverlos listos para ser proporcionados al clasificador.
* Objeto “cd”: Objeto que no usamos en esta función, pero que es necesario para detectar el coche. Dada una imagen donde se supone que hay un coche, devuelve la imagen del coche que se supone que debería detectar.
* Objeto “lpdn”: Objeto de la case LicensePlateNumbersDetector. Esta clase contiene el clasificador y puede ser entrenada para después predecir dígitos (1-9 y A-Z).

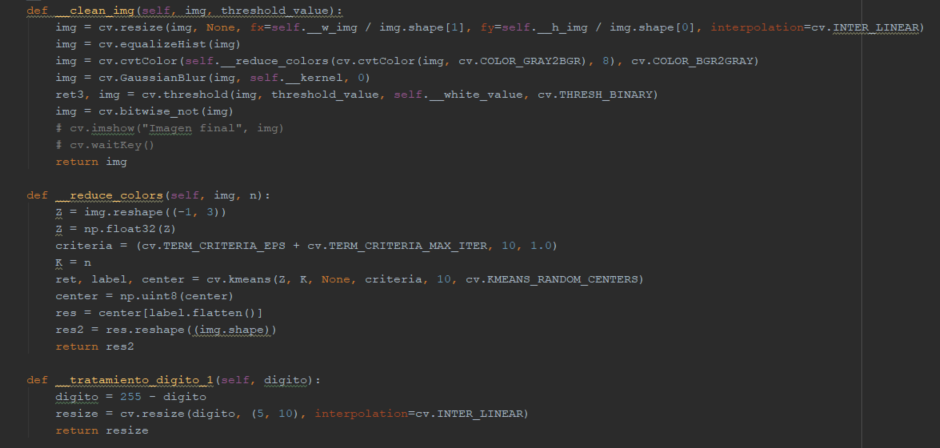
Lo que queda es bastante sencillo. Cargamos los dígitos que servirán de entrenamiento para le clasificador, contenidos en el directorio “training\_ocr”. También abrimos el fichero donde vamos a escribir el resultado con el formato pedido. Por cada digito que obtenemos del directorio, los dejamos en una lista para poder pasárselo a la función “train()” del objeto “lpnd”. Una vez entrenado el sistema, obtenemos las imágenes de testeo. Por cada imagen obtenemos los números de la matriculas gracias al objeto “lpc”. La función “obtener\_num\_matriculas”, aparte de los dígitos también devuelve un array de tuplas donde se encuentra el centro de la matrícula y la mitad de su ancho. Después por cada numero en resultado, lo guardamos en numbers, que será proporcionado al clasificador encargado de predecir los dígitos. Se lo concatenamos a “texto\_matricula” para mostrarlo en el fichero. Ahora bien, para poder mostrar todo de una sola vez, nos creamos una lista “info\_escritura” que será una lista de tuplas donde cada una contendrá información de la imagen asociada. Mas tarde después de añadir a la lista lo necesario, lo recorremos escribiendo en el fichero.



Ahora bien, esta función encargada de cargar las imágenes de full\_system es prácticamente igual que la anteriormente explicada. Tiene dos diferencias, ahora utilizamos el detector de coches porque no sabemos con certeza donde esta el coche ni cuantos coches hay, así que ahora tenemos un bucle mas encargado de recorrer todos los posibles coches, y otro recorriendo el array “centros\_largos”, ya que puede haber más de un coche.

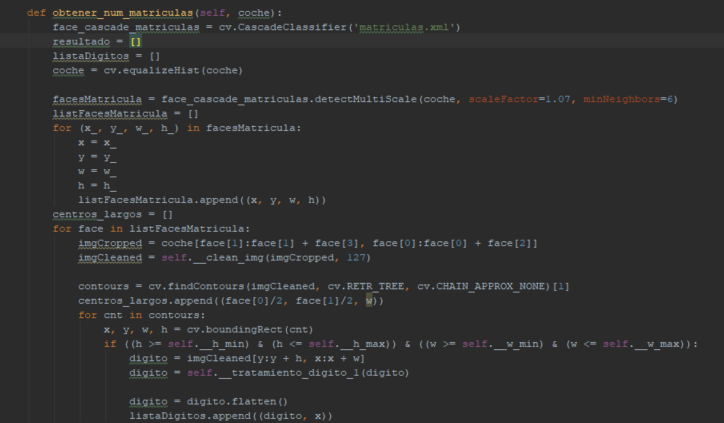
* Clase CarDetector.py

Esta clase solo tiene un método, encargado de decirte todos los coches que hay una determinada imagen. Después de obtener las “caras” donde se supone que se encuentran, se empaquetan en una lista de tuplas. Después, recortamos esas caras y las añadimos en una lista de coches, para después devolverla.

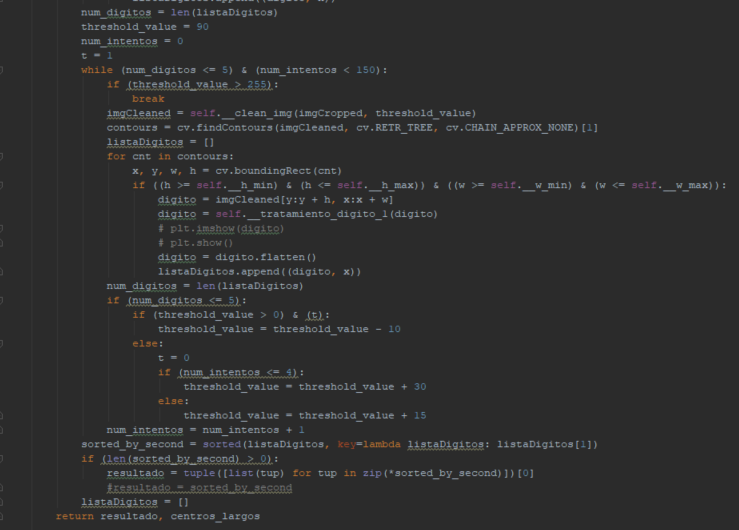
* Clase LicensePlateCleaner.py

Para empezar, procedemos a explicar las tres funciones que supondrán el esqueleto del algoritmo a seguir para la detección de los dígitos de la matrícula.

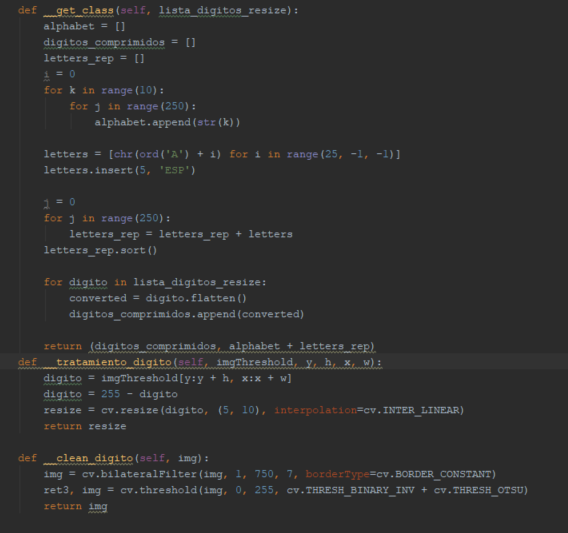
* La primera función “clean\_img”, sigue un proceso de tratado de la imagen en orden. En primer lugar, hacemos un resize de la matrícula, para tenerlas todas del mismo tamaño. A partir de aquí, el objetivo sobre la matricula es aumentar el contraste entre los números y el fondo de la matrícula, así que en primer lugar ecualizamos el histograma de la imagen y reducimos la imagen a 8 colores mediante las k-medias, así tenemos una imagen con un contraste alto entre números y lo demás. Se realiza el umbralizado para separar números de lo demás y después se devuelve en negativo para dejar lo que no interesa de color negro.
* En segundo lugar, la función “reduce\_colors”. Función sencilla que dado el numero de colores, te devuelve una imagen con solo 8 colores. Como hemos dicho antes usa el clasificador no supervisado de las k-medias para aumentar el contraste.
* Por último, tenemos la función “tratamiento\_digito\_1”, función que utilizamos después de extraer el numero de la matrícula, para dejarla en el formato que necesita el clasificador para decir qué número es. Básicamente lo que hace es dejarla como estaba antes de realizar el negativo en “clean\_img”, para después hacer un resize de (5,10), que es como estarán los dígitos de entrenamiento que se le han proporcionado el sistema.



La última función de la clase a explicar es la mas importante del algoritmo. Esa función se encarga de devolver los números listos para ser pasados al clasificador. En primer lugar, dado una imagen donde se supone que hay coches, obtiene sus matrículas, las recorre empaquetándolas en tuplas, para después recorrerlas, recortar la imagen para obtener la matricula en “imgCropped”, al final se limpia la imagen y se tiene la imagen “imgCleaned”. Ahora, para detectar los números se obtienen los contornos de la imagen y por cada contorno, se halla su bounding box, es decir, una caja que envuelve el digito.

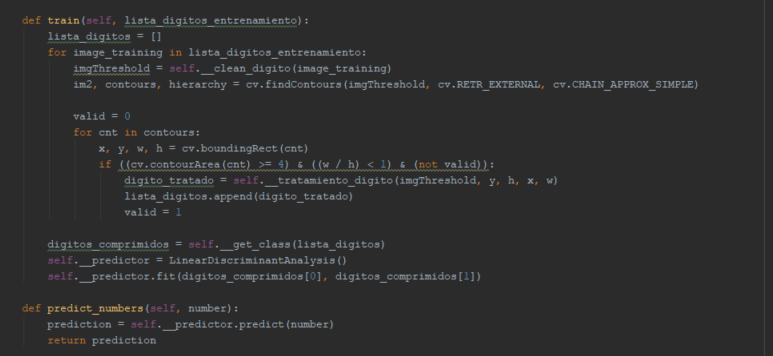
Una vez hecho esto, se comprueba que dicha caja es un numero y no otra cosa (como ruido), así que como hemos escalado la imagen siempre al mismo tamaño, sabemos que los dígitos tendrán un alto y un ancho más o menos fijo, se comprueba dicho rango, y se recorta el digito de la imagen, se trata el digito y se deja en un array de características (mediante la función “flatten”) ya que así lo necesita el clasificador, al final se añade a una lista de dígitos, seguido de su coordenada X, la razón se explica en la segunda parte de la función.

En esta segunda parte de la función, hacemos un segundo tratamiento de la matrícula. Es posible, que el valor que le hemos pasado a la función “clean\_img” para que umbralice no nos de ningún digito, así que hacemos que el algoritmo vuelva a pasar distintos umbralizados dependiendo de lo que ha detectado. Para ello sacamos el numero de dígitos y fijamos un valor de umbralizado. Mientras no se mejoren los dígitos y además se haya intentado la mejora de la detección de dígitos menos de 150 veces, lo que hacemos es volver a limpiar la imagen, pero con otro valor de umbralizado y repetimos el proceso de antes. Obtenemos los contornos y vemos si ahora detecta más dígitos que antes. Puede ocurrir que aun así el valor de umbralizado nuevo no haya servido, así que si el numero de dígitos sigue sin mejorar, restamos un valor al umbralizado. También puede ocurrir que no haya que restar para obtener más números si no que sea necesario sumar, así que, si el valor de umbralizado llega a 0, se levanta una bandera indicando que a partir de ahora se suma. Como el valor de umbralizado esta a 0 en ese punto, los 4 primeros intentos se suman de 30 en 30 ya que se sabe que no mejora mucho yendo de 10 en 10. A partir del 4 intento se suma de 15 en 15, para tener más precisión ya que no se sabe que pasara pasado el valor 127. Cuando este proceso finaliza, se devuelve el resultado además de los datos necesarios para el formato que se pide al escribir en fichero.

* Clase LicensePlateNumbersDetector.py

En esta clase al igual que en la anterior tenemos tres funciones que supondrán el esqueleto del algoritmo usado para entrenar al sistema.

* En primer lugar, la función “get\_class”, se encarga de la preparar las clases para el clasificador, así que saca el alfabeto y los números, lo que viene a ser el conjunto de dígitos. Posteriormente se ordena y se usa para el método “flatten()” para dejar el digito en el formato necesario para el clasificar, al final se devuelven los dígitos comprimidos y el conjunto de dígitos.
* En segundo lugar, tenemos la función “tratamiento\_digito”, que se encarga de dejar el digito tal y como lo necesita el clasificador. Se recorta de la matrícula, se deja como estaba antes de ser limpiado, y después se deja el digito en 5x10, tal y como habíamos dejado los dígitos que queremos predecir o reconocer. Posteriormente se devuelve.
* Al final, tenemos la función “clean\_digito”, función parecida a “clean\_img”, dado un digito, se le realiza un filtro bilateral, el umbralizado y posteriormente se devuelve.



Ahora bien, podríamos decir que estas son las funciones de la clase. Empecemos por la más fácil:

* La función “predict\_numbers”. Esta función recibe un numero y simplemente te devuelve su predicción de qué digito podría ser.
* Ahora bien, la función “train”. Esta función se encarga de entrenar al sistema con los dígitos pasados por parámetro. Recorremos la lista, por cada digito, se limpia, se obtienen los contornos, se recorren y se comprueba si es un digito valido mediante el área del contorno. Si cumple a condición, se trata el digito para dejarlo como lo necesita el clasificador, y se agrega a la lista. Al acabar el bucle y obtener todos los dígitos de entrenamiento, se obtienen las clases de esos dígitos, ya que el clasificador lo necesita. Se crea un objeto de tipo LinearDiscriminantAnalysis y se entrena mediante el método fit, pasándole los dígitos obtenidos después de llamar a la función “get\_class”.