UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática

Trabajo de Fin de Grado

QBarrier: Control de acceso a parking utilizando aprendizaje profundo

Alberto Carreras Franco

Septiembre, 2018

UNIVERSIDAD DE EXTREMADURA
Escuela Politécnica
Grado en Ingeniería Informática

Trabajo de Fin de Grado

QBarrier: Control de acceso a parking utilizando aprendizaje profundo

Autor: Alberto Carreras Franco
Fdo:

Director: Juan María Hernández Núñez
Fdo:

Cotutor: José Enrique Moguel Márquez
Fdo:

Tribunal Calificador

Presidente: José María Conejero Manzano
Fdo:

Secretario: Elena Jurado Málaga
Fdo:

Vocal: Adolfo Lozano Tello
Fdo:

Índice general

1. Introducción **11**

2. Estado del arte **13**

2.1. Aprendizaje automático 13

2.1.1. Aprendizaje supervisado 14

2.1.2. Aprendizaje no supervisado 14

2.1.3. Aprendizaje basado en refuerzo 14

2.1.4. Deep Learning 14

2.1.5. Redes neuronales artificiales 15

2.2. Almacenaje de datos 19

2.2.1. SQL - MySQL 19

2.2.2. NoSQL - MongoDB 19

2.2.3. Elección del gestor de base de datos 19

2.3. Detección de objetos 20

2.3.1. IBM Watson Visual Recognition 20
2.3.2. Clarifai 21
2.3.3. Yolo 21
2.3.4. Elección de la tecnología de detección de objetos 23
2.4. OCR 23
2.4.1. Tesseract 24
2.4.2. OpenALPR 24
2.4.3. Elección de tecnología de OCR 25
2.5. Lenguajes de programación 25
2.5.1. Python 25
2.5.2. Selección del lenguaje de programación 25
2.6. Herramientas de gestión de parking 26
2.6.1. Conclusiones sobre las herramientas de gestión de aparcamiento 26
2.7. Conclusiones generales 27

3. Análisis de Requisitos **29**

3.1. Objetivos 29
3.2. Análisis y diseño 29
3.2.1. Requisitos funcionales 29
3.2.2. Requisitos no funcionales 30

3.2.3. Diagramas de casos de uso 30

3.2.4. Diagrama de componentes 31

3.2.5. Diagramas de clases 32

3.2.6. Hardware 36

4. Propuesta de solución **39**

4.1. Sistema general 40

4.2. ObjectDetector 41

4.3. PlateRecognizer 43

4.4. CarCounter 44

4.5. QBarrier 47

4.5.1. Sensor Virtual HAL 50

4.5.2. Estructura de datos del log 51

5. Metodología **53**

6. Validación del proceso **55**

6.1. Caso positivo 55

6.2. Caso en el que no se detecta vehículo 56

6.3. Caso en el que no se detecta matrícula 57

6.4. Conclusiones finales sobre los resultados obtenidos 57

7. Conclusiones **59**

8. Trabajos futuros **61**

8.1. Investigación sobre nuevas API de reconocimiento de objetos y OCR 61

8.2. Detección de vehículos especiales 61

8.3. Cálculo de horas de cada vehículo dentro de las instalaciones . 62

8.4. Detección de matrículas en lista negra 62

9. Agradecimientos **63**

A. Manual del programador **65**

A.1. Dependencias 65

A.2. Instalación de Cuda 66

A.3. Estructura del código 68

A.4. Despliegue del sistema 70

A.4.1. ObjectDetector 70

A.4.2. PlateRecognizer 70

A.4.3. CarCounter 70

A.4.4. QBarrier 71

Índice de figuras

1.1. Diagrama básico del proyecto 12

2.1. Comparativa entre deep learning y machine learning 15

2.2. Ejemplo de RNA 16

2.3. Ejemplo de RNA Convolucional 17

2.4. Neurona de pooling 18

2.5. Diagrama tecnológico 28

3.1. Diagrama de casos de uso 31

3.2. Diagrama de componentes 31

3.3. Diagrama de clases de Object Detector 32

3.4. Diagrama de clases de Plate Recognizer 33

3.5. Diagrama de clases de Car Counter 34

3.6. Diagrama de clases de MongoClient 35

3.7. Diagrama de clases de QBarrier 36

3.8. Imagen tomada desde la cámara AXIS Q1614 37

3.9. Imagen tomada desde la cámara AXIS Q1602 38

4.1. Imagen Arquitectura general del proyecto 39

4.2. Esquema de funcionamiento de ObjectDetector 41

4.3. Esquema de funcionamiento de PlateRecognizer 43

4.4. Esquema de funcionamiento de CarCounter 44

4.5. Visual de QSS 45

4.6. Sensor CarCounter en QSS 46

4.7. Funcionamiento de QBarrier en la cámara de salida 47

4.8. Funcionamiento de QBarrier en la cámara de entrada 48

4.9. Sensor de matrículas incrustado en QSS 50

4.10. Sensor de matrículas incrustado en QSS 50

5.1. Incidencias creadas durante la implementación del proyecto . . 54

6.1. Caso positivo de imagen 55

6.2. Falso positivo como resultado del movimiento de la cámara . . 56

6.3. Resultados obtenidos en la cámara de entrada 57

6.4. Resultados obtenidos en la cámara de salida 58

A.1. Estructura general del código del proyecto 68

A.2. Estructura del código del módulo Plate Recognizer 68

A.3. Estructura del módulo Car Counter 68

A.4. Estructura del código de QBarrier 69

A.5. Estructura del código de la librería Util 69

A.6. Estructura del código del módulo Object Detector 69

Resumen

Uno de los problemas más recurrentes en cualquier parking es la gestión y el control del aparcamiento. Actualmente, en la Escuela Politécnica ocurre el mismo problema en la zona de aparcamientos de la zona superior. Previamente, en dicho parking, se implantó un sistema de tarjetas para abrir la barrera, pero supone un gasto para los usuarios que quieren entrar, ya que tenían que depositar una cantidad de dinero por dicha tarjeta además de no asegurar la entrada indiscriminada de vehículos. Posteriormente se instaló un botón a la entrada y salida de la barrera para abrir la barrera y para ahorrar el trámite anterior, pero también deriva en el problema de la entrada indiscriminada.

Ante esta situación, proponemos QBarrier, una aplicación para la gestión inteligente del aparcamiento utilizando técnicas de aprendizaje profundo. Utilizando las imágenes que proveen las cámaras de la universidad, se aplican técnicas de detección de objetos para reconocer los vehículos que se aproximan a la barrera del parking y se detecta qué vehículo es y se comprueba si tiene acceso a través de su matrícula.

Abstract

Parking management and control is one of the most recurrent problems in any car park. Currently, the same problem occurs in the *Escuela Politécnica* in the parking area of the upper zone. Previously, in said parking, a card system was implemented to open the barrier, but it is an expense for the users who want to enter, since they had to deposit a quantity of money for said card. It also does not ensure the indiscriminate entry of vehicles. Later a button was installed at the entrance and exit of the barrier to open the barrier and to save the previous expenditure, but it also leads to the problem of indiscriminate entry.

Faced with this situation, we propose QBarrier, an application for intelligent parking management using deep learning techniques. Using the images provided by the cameras of the university, object detection techniques are applied to recognize the vehicles approaching the parking barrier and the vehicle is detected and checked if it has access through its plate code.

Capítulo 1

Introducción

Con la evolución de la inteligencia artificial en los últimos años han surgido nuevas técincas de aprendizaje como son el ***machine learning*** y dentro de ella el ***deep learning***. Estas formas de aprendizaje sumadas a era de los datos masivos ofrece un abanico de posibilidades muy amplio no solo con datos numéricos, sino con diversos contenidos multimedia como imágenes, vídeo, audio o texto natural.

Dentro de este contexto nace QBarrier, la evolución de un proyecto interno dentro del grupo Quercus, que proveía de un acceso al parking de la universidad y ofrecía una interfaz de usuario a través de Telegram. El problema de este proyecto era la falta de mantenibilidad y la escalabilidad. Por ello QBarrier ofrece una solución escalable y adaptable a cualquier entorno, con fácil despliegue e integración debido al diseño de una arquitectura con todos sus componentes desacoplados.

Una vez descrito el contexto y los objetivos del sistema se plasma una primera aproximación de arquitectura que se irá detallando en los siguientes apartados:

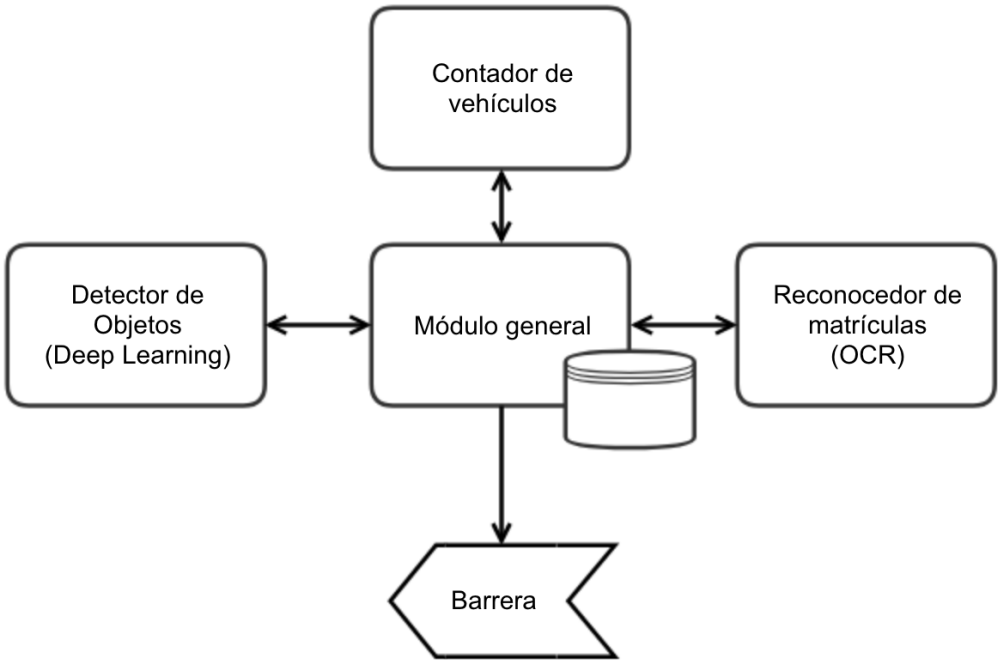

Figura 1.1: Diagrama básico del proyecto

El presente documento se estructura de la siguiente manera: en el capítulo 2 se presenta el Estado del Arte dónde evalúan y valoran las distintas alternativas tecnológicas referentes a nuestro caso de uso expuesto anteriormente que existen en el mercado, en el capítulo 3 se detalla la fase de análisis y diseño del proyecto que se propone en el capítulo 4. En el capítulo 5 se detalla la metodología seguida en el proyecto seguida de la validación de los procesos en el capítulo 6. Posteriormente se incluyes las conclusiones (capítulo 7) y los trabajos futuros y agradecimientos en los capítulos 8 y 9 respectivamente.

Además se incluyen cuatro anexos, uno con las dependencias del proyecto, otro con la instalación de Cuda (una de las dependencias con gran dificultad de instalación). El siguiente contiene la estructura del código del proyecto para referenciar las menciones de archivos durante la explicación y el último con las instrucciones de despliegue del sistema.

Capítulo 2

Estado del arte

En este apartado, se describe el estado de las tecnologías existentes relacionadas con nuestro problema evidenciando las ventajas e inconvenientes de cada una y el porqué de la elección de la tecnología escogida.

2.1. Aprendizaje automático

Como se ha hablado anteriormente, el Aprendizaje Automático (en inglés Machine Learning) es la rama de la Inteligencia Artificial que tiene como objetivo crear algoritmos capaces de generalizar comportamientos y reconocer patrones a partir de una información suministrada en forma de ejemplos. Es, por lo tanto, un proceso de inducción del conocimiento, es decir, un método que permite obtener por generalización un enunciado a partir de enunciados que describen casos particulares. Según Tom Mitchell en 1997, "una máquina aprende de una particular tarea T considerando las experiencias de tipo E respecto de una medida de rendimiento P, si la máquina efectivamente mejora su rendimiento P en la tarea T a partir de la experiencia E". [1] Dentro de esta rama del conocimiento, se encuentra una clasificación de cada algoritmo según el tipo de aprendizaje que utilicen:

2.1.1. Aprendizaje supervisado

Aprendizaje supervisado: Cuenta con un conjunto de ejemplos de los cuales se conoce la respuesta, denominados datos etiquetados. El objetivo del aprendizaje supervisado es formular algún tipo de regla o correspondencia que permita dar (o aproximar) la respuesta para todos los objetos que se presenten.

2.1.2. Aprendizaje no supervisado

Tiene lugar cuando no se dispone de datos etiquetados para el entrenamiento. Solo se conocen los datos de entrada, pero estos datos no están etiquetados con respecto a una variable que los clasifique, por lo que solo se puede describir la estructura de los datos para intentar encontrar algún tipo de organización que simplifique el análisis.

2.1.3. Aprendizaje basado en refuerzo

Es el tipo menos común. Los algoritmos de aprendizaje por refuerzo reciben información del entorno y pueden modificar el mismo. Además es no supervisado, ya que sustituye la información supervisada por información del tipo acción/reacción.

2.1.4. Deep Learning

Deep Learning es la etiqueta para un conjunto de algoritmos que habitualmente se emplean para el entrenamiento no supervisado de redes neuronales multicapa. No existe una única definición de aprendizaje profundo o Deep Learning. En general, se trata de una clase de algoritmos para aprendizaje automático. El aprendizaje profundo o Deep Learning es una rama del aprendizaje automático que enseña a las computadoras lo que naturalmente los seres humanos hacen: aprender de la experiencia. Los algoritmos de aprendizaje de máquinas o Machine Learning utilizan métodos computacionales para aprender información directamente de los datos sin depender de una ecuación predeterminada como modelo. [2] Este aprendizaje es especialmente adecuado para el reconocimiento de imágenes, lo cual es importante

para resolver problemas como reconocimiento facial, detección de movimiento y muchas tecnologías avanzadas de asistencia al conductor, tales como conducción autónoma, detección de carriles, detección de peatones y estacionamiento autónomo. La popularidad del Deep Learning se basa en que el método conocido como descenso por gradiente estocástico (SGD) puede entrenar eficazmente redes neuronales de entre 10 y 20 capas para resolver problemas que ocurren en la práctica. Lo cual es relevante porque el reajuste de pesos en una red neuronal a partir de los errores es un problema de optimización no convexa NP-Completo.

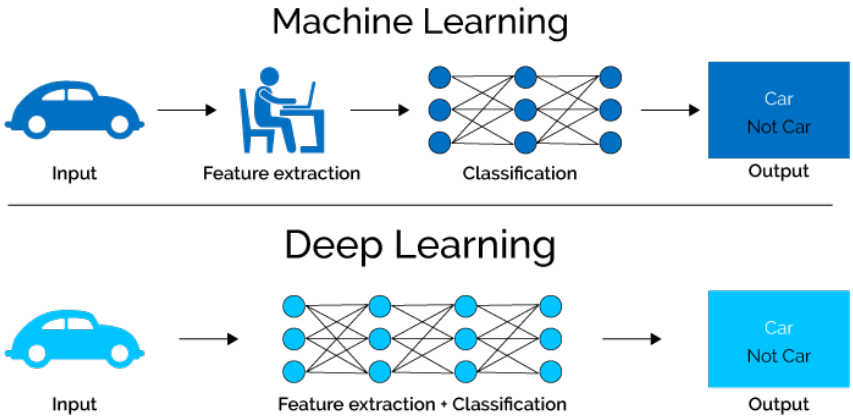

Figura 2.1: Comparativa entre deep learning y machine learning
Fuente: `https://www.quora.com/What-is-the-difference-between-deep-learning-and-usual-machine-learning`

Como se puede apreciar en la imagen 2.1.4, la diferencia entre Machine Learning y Deep Learning es la fase conocida como ***feature engineering***. Esta fase mapea los datos de entrenamiento (x) a una salida (y) acorde a f(x)=y, la cual debe replicarse a unos datos nuevos (x') para obtener otra salida acorde a la función.

2.1.5. Redes neuronales artificiales

Las redes neuronales artificiales (RNAs), tratan de emular el comportamiento del cerebro humano, caracterizado por el aprendizaje a través de la experiencia y la extracción de conocimiento genérico a partir de un conjunto de datos [3]. En la estructura de estas redes se puede destacar que unos elementos simples (neuronas) están unidos por una cantidad masiva de enlaces

de forma jerárquica y procesan información por medio de su estado dinámico como respuesta a entradas externas.

Desde el punto de vista formal, una RNA puede definirse haciendo uso del concepto de grafo, objeto integrado por un conjunto de nodos (vértices), y de conexiones (links) entre los mismos.

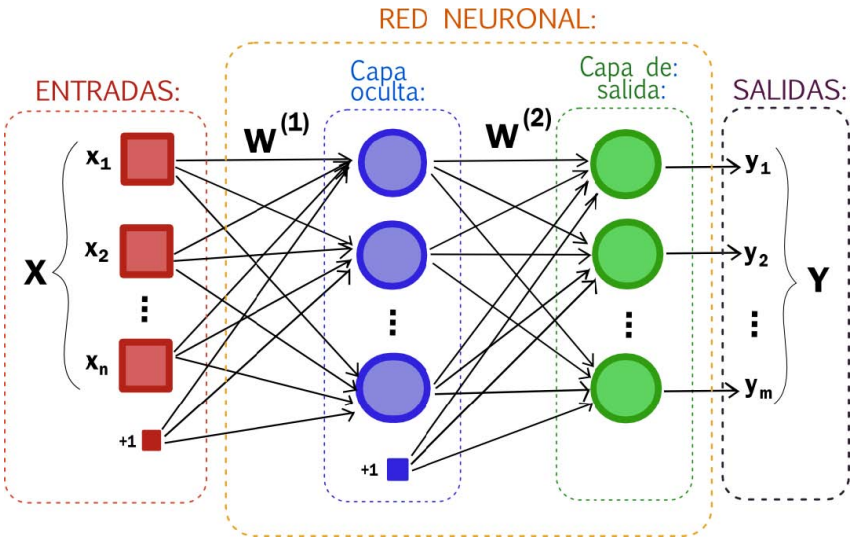

Figura 2.2: Ejemplo de RNA
Fuente: `https://artfromcode.wordpress.com/2017/04/18/red-neuronal-en-python-con-numpy-parte-1/`

En esta imagen se puede ver más claramente como es la unión entre neuronas. Todas las neuronas de una capa mantienen una conexión con todas las de la capa siguiente y con las de la anterior.

Redes neuronales convolucionales

Una red neuronal convolucional es un tipo de red neuronal artificial donde las neuronas corresponden a campos receptivos de una manera muy similar a las neuronas en la corteza visual primaria de un cerebro biológico. Este tipo de red es una variación de un perceptron multicapa [4], sin embargo, debido a que su aplicación es realizada en matrices bidimensionales, son muy efectivas para tareas de visión artificial, como la clasificación y segmentación de imágenes, entre otras aplicaciones. [5]

Como redes de clasificación, al principio se encuentra la fase de extracción de características, compuesta de neuronas convolucionales y de reducción de muestreo. Al final de la red se encuentran neuronas de perceptron sencillas para realizar la clasificación final sobre las características extraídas. La fase de extracción de características se asemeja al proceso estimulante en las células de la corteza visual. Esta fase se compone de capas alternas de neuronas convolucionales y neuronas de reducción de muestreo. Según progresan los datos a lo largo de esta fase, se disminuye su dimensionalidad, siendo las neuronas en capas lejanas mucho menos sensibles a perturbaciones en los datos de entrada, pero al mismo tiempo siendo estas activadas por características cada vez más complejas.

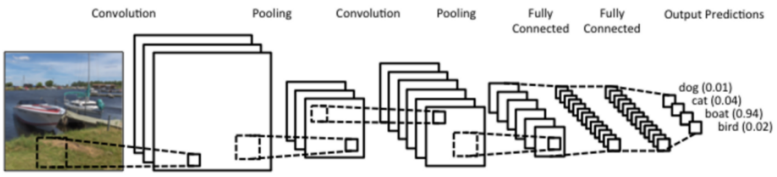

Figura 2.3: Ejemplo de RNA Convolucional
Fuente: `https://technologiesrunning.blogspot.com/2017/03/aprendizaje-profundo-para-principiantes.html`

Como se puede observar en esta imagen 2.3 se distinguen 3 tipos de neuronas en la imagen:

- **Neuronas convolucionales**: El operador de convolución tiene el efecto de filtrar la imagen de entrada con un núcleo previamente entrenado. Esto transforma los datos de tal manera que ciertas características (determinadas por la forma del núcleo) se vuelven más dominantes en la imagen de salida al tener estas un valor numérico más alto asignados a los pixeles que las representan. Estos núcleos tienen habilidades de procesamiento de imágenes específicas, como por ejemplo la detección de bordes que se puede realizar con núcleos que resaltan la gradiente en una dirección en particular. Sin embargo, los núcleos que son entrenados por una red neuronal convolucional generalmente son más complejos para poder estos extraer otras características más abstractas y no triviales.
- **Neuronas de reducción de muestreo (Polling)**: Las redes neuronales cuentan con cierta tolerancia a pequeñas perturbaciones en los datos

de entrada. Por ejemplo, si dos imágenes casi idénticas (diferenciadas únicamente por un traslado de algunos pixeles lateralmente) se analizan con una red neuronal, el resultado debería de ser esencialmente el mismo. Esto se obtiene, en parte, dado a la reducción de muestreo que ocurre dentro de una red neuronal convolucional. Al reducir la resolución, las mismas características corresponderán a un mayor campo de activación en la imagen de entrada.

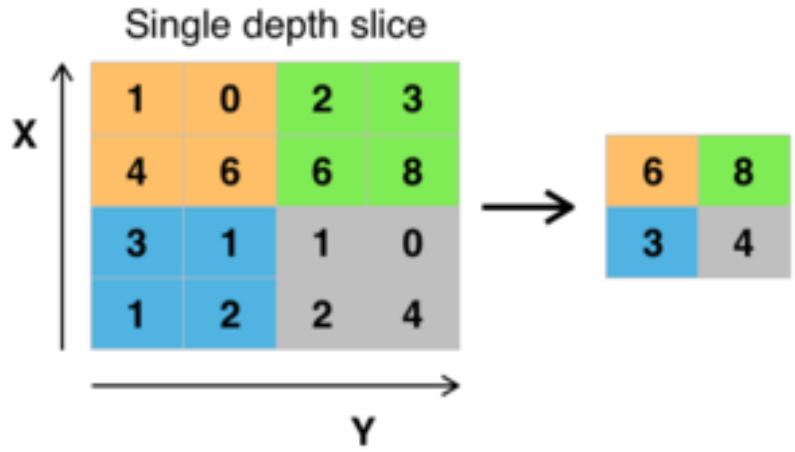

Figura 2.4: Neurona de pooling
Fuente: `http://turing.iimas.unam.mx/~gibranfp/cursos/ap2018-I/slides/03_redes_convolucionales.pdf`

En la imagen 2.4 se puede apreciar como se realiza la función de pooling. A partir de una entrada de longitud MxN se obtiene una feature consecuencia de toda esta entrada. Por ejemplo un conjunto de puntos colocados horizontalmente y muy próximos es una línea horizontal. A partir de esta abstracción se pueden ir detectando patrones cada vez más genéricos hasta llegar a detectar figuras, por ejemplo.

- **Neuronas de clasificación**:Después de una o más fases de extracción de características, los datos finalmente llegan a la fase de clasificación. Para entonces, los datos han sido depurados hasta una serie de características únicas para la imagen de entrada, y es ahora la labor de esta última fase el poder clasificar estas características hacia una etiqueta u otra, según los objetivos de entrenamiento.

2.2. Almacenaje de datos

Respecto a esta cuestión se barajaron dos tecnologías distintas y que a priori solucionaban el problema que se previó para guardar el histórico de las operaciones que ocurrían en el sistema. Ambas tecnologías

2.2.1. SQL - MySQL

MySQL es un sistema de gestión de bases de datos relacional desarrollado bajo licencia dual: Licencia pública general/Licencia comercial por Oracle Corporation y está considerada como la base datos de código abierto más popular del mundo, y una de las más populares en general junto a Oracle y Microsoft SQL Server, sobre todo para entornos de desarrollo web.[6] Uno de los aspectos interesantes para nuestro sistema es la replicación (por la ocurrencia de posibles caídas), lo cual añadía una confianza e integridad en los datos.

2.2.2. NoSQL - MongoDB

MongoDB es una base de datos NoSQL que provee de alta disponibilidad y escalabilidad horizontal, útil para nuestro sistema, que aunque no realice un tráfico masivo de datos, realiza operaciones(la mayoría de las veces insercciones) de un gran volumnen. Esta BD almacena los datos en documentos, estilo JSON, lo cual quiere decir que los campos pueden variar en cada documento y la estructura de datos puede variar a lo largo del tiempo.

2.2.3. Elección del gestor de base de datos

A la hora de elegir la tecnología de base de datos, se tuvo en cuenta los siguientes factores:

- El gran volumen de datos que se generaba, hacía que la elección de una base de datos NoSQL frente a una SQL tuviera mucho más sentido ya que las operaciones de inserción son mucho más rápidas.

- A priori nuestros datos iban a tener una estructura fija, ya que cada registro del histórico iba a tener unos campos fijos dependiendo del evento que se producía, aunque se detallará con más detenimiento en el capítulo 3. Como se quería guardar la imagen que producía dicho evento para su posterior validación, esta debía representarse en un campo BLOB o TEXT (si la imagen se guardaba en Base64), y se obtenían imágenes con distintas resoluciones debido a la heterogeneidad de las cámaras, se desperdiciaba mucha memoria para dicho almacenado.

2.3. Detección de objetos

Dentro del campo de la detección de objetos, existe una gran amplitud de herramientas que poder utilizar, debido al auge del uso del ***Machine learning*** para este tipo de problemas. De entre todas las encontradas, se realizó una selección debido a que el gran volumen de imágenes que se requerían para resolver el problema (unas 10000 fotos mensuales). Después de esta selección, cabía la posibilidad de utilizar dos tecnologías cloud, que dentro de los márgenes, se mantenían dentro de unos límites de precio razonables y con una herramienta local, que se puede desplegar en nuestros terminales, y que por ende, no suponía ningún coste en su consulta.

2.3.1. IBM Watson Visual Recognition

Watson [7] es una herramienta propietaria de IBM que ofrece su servicio de detección de imágenes. A través de su api, disponible en Python, Java y Node, se pueden crear modelos entrenados a medida, o utilizar los clasificadores provistos por el servicio.

Respecto a nuestro caso de uso, se dispone de imágenes con todo tipo de resoluciones y rangos lumínicos y esta herramienta ofrecía una gran precisión en condiciones idílicas, pero en cuanto las condiciones de la imagen empeoraban, el resultado no se ajustaba a las necesidades que la aplicación práctica requería, por tanto, se descartó como opción.

2.3.2. Clarifai

En el concurso de reconocimiento de imágenes promovido por ImageNet en la edición de 2013[8], uno de los mayores proveedores de datasets de imágenes del mundo, aparece Clarifai en la parte superior de varios rankings, tales como acierto en Top1 y Top5. Por tanto, se comprobó si el servicio que ofrecían se ajustaba a nuestro caso de uso. El uso de esta herramienta se realizaba a través de una api, la cual daba soporte a varios lenguajes de programación. Además esta herramienta suponía un coste aproximado de 10€mensuales por su uso, calculado con el volumen de consultas citado anteriormente.

Tras realizar las distintas pruebas, se puede comprobar que los resultados eran muy buenos en condiciones lumínicas idóneas, pero en cuanto la cantidad de luz disminuía, el número de errores aumentaba significativamente.

2.3.3. Yolo

Esta herramienta se diferenciaba de las demás por ser de código abierto, por lo tanto se poseía la capacidad de modificar y adaptar el código para nuestro uso. Además en su página web, incluyen las instrucciones para usarlo y como realizar entrenamientos ya que como especifican en el las notas de investigación e implementación de sus creadores [9], esta herramienta se basa en una red neuronal convolucional para realizar las predicciones y se incluye la especifiación de ésta. La especificación de la red neuronal utilizada es la siguiente:

```
layer     filters    size              input                output
    0 conv     32  3 x 3 / 1  608 x 608 x  3   ->  608 x 608 x 32
    1 max          2 x 2 / 2  608 x 608 x 32   ->  304 x 304 x 32
    2 conv     64  3 x 3 / 1  304 x 304 x 32   ->  304 x 304 x 64
    3 max          2 x 2 / 2  304 x 304 x 64   ->  152 x 152 x 64
    4 conv    128  3 x 3 / 1  152 x 152 x 64   ->  152 x 152 x 128
    5 conv     64  1 x 1 / 1  152 x 152 x 128  ->  152 x 152 x 64
    6 conv    128  3 x 3 / 1  152 x 152 x 64   ->  152 x 152 x 128
    7 max          2 x 2 / 2  152 x 152 x 128  ->   76 x 76 x 128
    8 conv    256  3 x 3 / 1   76 x 76 x 128   ->   76 x 76 x 256
    9 conv    128  1 x 1 / 1   76 x 76 x 256   ->   76 x 76 x 128
   10 conv    256  3 x 3 / 1   76 x 76 x 128   ->   76 x 76 x 256
   11 max          2 x 2 / 2   76 x 76 x 256   ->   38 x 38 x 256
   12 conv    512  3 x 3 / 1   38 x 38 x 256   ->   38 x 38 x 512
```

```
13 conv   256  1 x 1 / 1   38 x  38 x 512  ->   38 x  38 x 256
14 conv   512  3 x 3 / 1   38 x  38 x 256  ->   38 x  38 x 512
15 conv   256  1 x 1 / 1   38 x  38 x 512  ->   38 x  38 x 256
16 conv   512  3 x 3 / 1   38 x  38 x 256  ->   38 x  38 x 512
17 max         2 x 2 / 2   38 x  38 x 512  ->   19 x  19 x 512
18 conv  1024  3 x 3 / 1   19 x  19 x 512  ->   19 x  19 x1024
19 conv   512  1 x 1 / 1   19 x  19 x1024  ->   19 x  19 x 512
20 conv  1024  3 x 3 / 1   19 x  19 x 512  ->   19 x  19 x1024
21 conv   512  1 x 1 / 1   19 x  19 x1024  ->   19 x  19 x 512
22 conv  1024  3 x 3 / 1   19 x  19 x 512  ->   19 x  19 x1024
23 conv  1024  3 x 3 / 1   19 x  19 x1024  ->   19 x  19 x1024
24 conv  1024  3 x 3 / 1   19 x  19 x1024  ->   19 x  19 x1024
25 route 16
26 conv    64  1 x 1 / 1   38 x  38 x 512  ->   38 x  38 x  64
27 reorg             / 2   38 x  38 x  64  ->   19 x  19 x 256
28 route 27 24
29 conv  1024  3 x 3 / 1   19 x  19 x1280  ->   19 x  19 x1024
30 conv   425  1 x 1 / 1   19 x  19 x1024  ->   19 x  19 x 425
31 clasification
```

Como se puede ver, la estructura de la red neuronal utilizada tiene 30 capas además de la capa de salida. Además de las capas convolucionales añade unas capas max (polling) que sirven para reducir la resolución y así activar las neuronas de las capas más internas.

En la web [10] se incluyen archivos de configuración que han sido los utilizados durante la realización del proyecto, ya que son de propósito general e incluyen coches, motos y camiones, que son los necesarios para el dominio de nuestra aplicación. Esta herramienta está escrita en C, pero existen wrappers en una gran variedad de lenguajes tales como C++, Java o en este caso Python.

Otra de las características más importantes de esta herramienta, es que soporta la aceleración por GPU, se puede compilar para usar CUDA y así conseguir un mejor tiempo de ejecución.

Las pruebas realizadas con esta herramienta arrojaban unos resultados estables en cualquier tipo de situación, tales como baja luminosidad o reflejos, si la resolución de la imagen se mantenía dentro de unos límites (1080p).

2.3.4. Elección de la tecnología de detección de objetos

Las siguientes tablas contienen los coeficientes de acierto (de 0 a 1) en cada una de las franjas horarias obtenidos por todas los detectores partiendo de un conjunto de 200 imágenes similares a la imagen 3.8:

Cuadro 2.1: Cámara de entrada barrera

	Día	Tarde	Noche
Watson	0,935	0,62	0,06
Clarifai	1	0,965	0,55
Yolo	0,985	0,955	0,78
Total	200	200	200

Cuadro 2.2: Cámara de salida barrera

	Día	Tarde	Noche
Watson	0,42	0,27	0
Clarifai	1	0,91	0,425
Yolo	0,95	0,87	0,52
Total	200	200	200

Con los resultados obtenidos, se puede comprobar que durante las horas de luz solar, se conseguía una tasa de acierto similar con las herramientas Clarifai y Yolo, mientras que con Watson, el ratio era notablemente peor. En cuanto a la franja horaria sin luz, no se asegura el funcionamiento correcto con ninguna de las herramientas, ya que los focos que incorporan los vehículos hacen que la calidad de la foto disminuya en gran medida. Por otra parte, también es un factor a tener en cuenta, es que Clarifai requiere de un coste continuo, ya que la empresa vende el servicio, mientras que el Yolo se puede instalar en local y no supondrá ningún coste, debido a que no hay que comprar el software ni mantener un servicio.

2.4. OCR

Dentro de este apartado, el espectro en el que realizar investigación se reduce notablemente, ya que hay herramientas y proyectos establecidos que cumplen con solvencia su función.

Para el caso de uso se requiere detectar las matrículas que identifican los distintos vehículos. Se han tenido en cuenta dos reconocedores de texto en imágenes, uno de propósito general, Tesseract y otro específico de reconocimento de matrículas basado en el anterior, conocido como OpenALPR.

2.4.1. Tesseract

Tesseract [11] es una herramienta para realizar OCR de propósito general, es decir, para reconocer cualquier tipo de texto en imágenes. Reconoce caracteres de más de 100 lenguajes y además puede entrenarse para reconocer otros tipos. Incluye api para C y C++, aunque la comunidad ha implementado wrappers en distintos lenguajes como Java y Python.

Las pruebas que se han realizado sobre esta herramienta han realizado unos resultados muy positivos, acertando la totalidad de las matrículas, pero en algunas ocasiones, se detectan vehículos con distintas marcas publicitarias que entorpecen el funcionamiento de la herramienta.

2.4.2. OpenALPR

Es una herramienta [12] similar a Tesseract, ya que usa su mismo motor de OCR, pero se especializa en la detección de matrículas. Por lo tanto, se trata con una herramienta especifica del dominio que nos ocupa y además incluye soporte directamente en Python.

Otra funcionalidad importante de este software es que se puede configurar para detectar matrículas europeas o de EEUU para afinar más su precisión.

De las pruebas realizadas sobre esta herramienta se han obtenido resultados similares a los anteriores, pero está vez eliminando el ruido que obtenía con la herramienta anterior.

2.4.3. Elección de tecnología de OCR

Como se especificó anteriormente, OpenALPR está basada en Tesseract y por lo tanto los resultados que arrojan son similares, pero OpenALPR elimina el posible ruido que pueda ocurrir de otros caracteres que puedan llegar a aparecer en los vehículos.

Cuadro 2.3: Análisis de reconocedores OCR - Cámara de entrada

	Día	Tarde	Noche
Tesseract	1	1	0,55
OpenALPR	1	1	0,55

Otro aspecto a destacar es que, con imágenes como 3.9 ninguno de los detectores ha conseguido detectar ni una sola matrícula debido a la bajísima calidad de imagen.

2.5. Lenguajes de programación

2.5.1. Python

Es un lenguaje de programación interpretado que tiene como premisa el código legible. Soporta múltiples paradigmas, tales como el orientado a objetos, el imperativo o el funcional.

Según IEEE Spectrum [13], se ha convertido en el lenguaje más utilizado en 2017, ya que se utiliza en ámbitos que están en auge constante, como el análisis de datos, o la programación web y además goza de una gran comunidad que lo respalda.

2.5.2. Selección del lenguaje de programación

Solo se ha valorado la utilización del lenguaje Python por que era el único que englobaba todas las tecnologías vistas anteriormente, ya que Yolo,

por ejemplo, solo se encontraba implementado en C y Python. Además aportaba una ventaja adicional que se denota en la capacidad de poder hacer scripts, tales como una API escuchando en un puerto sin necesidad de crear clases como en Java, o aportando mucha más simpleza en el código frente al propio lenguaje C/C++.

Además engloba todas las tecnologías web y de comunicación que se utilizarán para la comunicación tales como HTTP, para realizar APIs que comunicarán los distintos módulos entre sí y por ende TCP para comunicar los módulos necesarios con las cámaras3.2.6 IP que ofrecen las imágenes, ya que son la única interfaz que ofrecen.

2.6. Herramientas de gestión de parking

Dentro de la sección de herramientas completas de gestión de parking se evidencian:

- Equinsa Parking comercializa [14] un sistema de gestión de aparcamiento que funciona con tarjetas de banda magnética, código de barras, matrícula o por móvil. Esta solución ofrece un diseño modulable y escalable.
- Al igual que la anterior, I+D3 [15] comercializa una solución versátil en cuanto a método de control, escalable, modulable y gestionable remotamente.

2.6.1. Conclusiones sobre las herramientas de gestión de aparcamiento

Cuadro 2.4: Análisis de tecnologías sobre gestión del aparcamiento

	Equinsa	I+D3	QBarrier
Detección de vehículos	Sí	Sí	Sí
Detección de matrículas	Sí	Sí	Sí
Dependencia de servicios externos	Sí	Sí	No
Código abierto	No	No	Sí

Todas las herramientas ofrecen una buena solución a la gestión del aparcamiento, que cumple con los objetivos funcionales del problema y a la vez cumpliendo con los requisitos funcionales tales como los mencionados en cada uno de sus apartados. Pero todas mantienen varios problemas, tales como la imposiblidad de extender la funcionalidad de la herramienta, ya que todas las herramientas son de software propietario. Además con lo que ello supone hay que comprar una licencia y mantener el servicio que provee cada una de las empresas.

2.7. Conclusiones generales

Como resumen sobre la selección de tecnologías se recalca:

- La elección de crear un sistema propio frente a las tecnologías de gestión de parking comerciales por no ser de código abierto además de conllevar una gran inversión inicial comparado con la solución aportada.
- Se ha seleccionado Yolo por delante de las otras herramientas, Clarifai y Watson, porque no conlleva el pago de un servicio externo y por arrojar en general mejores resultados.
- La elección de OpenALPR frente a Tesseract porque ofrece un nivel de abstracción mayor al detectar solamente matrículas y por ende elimina otro tipos de caracteres que puedan aparecer en la imagen.
- Se ha seleccionado Flask como motor de APIs, ya que es un framework sencillo y rápido que ofrece la funcionalidad suficiente para el caso de uso que se trata.

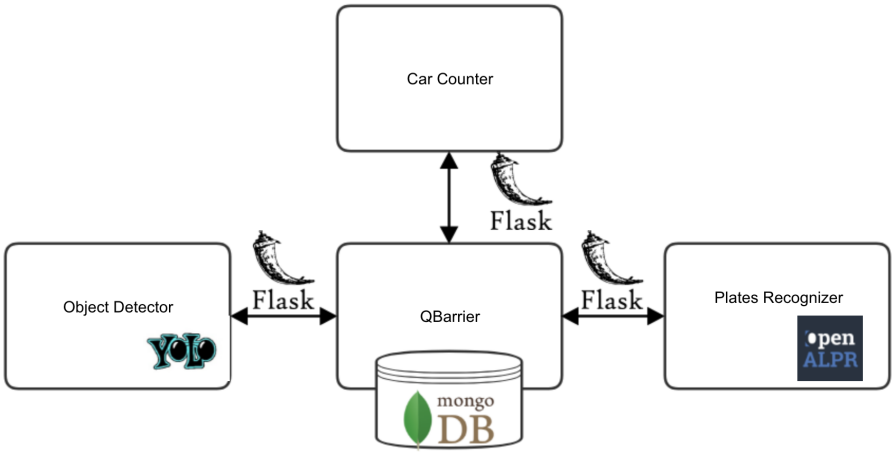

Figura 2.5: Diagrama tecnológico

Capítulo 3

Análisis de Requisitos

3.1. Objetivos

Una vez analizadas las tecnologías actuales y el problema a abordar se determinaron los objetivos del proyecto a realizar como los siguientes:

- El sistema debe ser fácil de desplegar y adaptable a cualquier lugar de aparcamiento.
- Debe ser robusto ante posibles posibles problemas que ocurran refereridos a la concurrencia de entrada y salida del párking.
- El sistema debe tener un tiempo de respuesta razonable, sin superar un par de segundos para no hacer demorar al usuario y no provocar atascos por el funcionamiento lento del sistema.

3.2. Análisis y diseño

3.2.1. Requisitos funcionales

- El sistema debe reconocer los vehículos y posteriormente las matrículas de vehículos para comprobar si ésta está contemplada para abrir la

barrera en caso de que éste quiera entrar.

- En caso contrario, el sistema en caso de reconocer un vehículo abrirá la barrera y posteriormente, para no hacer demorar al usuario, se reconoce su matrícula para saber que vehículo ha salido de la zona de estacionamiento.
- Todas las entradas y salidas son almacenadas con su matrícula, si es que se ha reconocido, para su posterior estudio.
- El sistema debe integrarse con el sistema de seguridad, QSS, para la visualización del estado del sistema, incluyendo la última matrícula reconocida además de alguna alerta que indique si es un coche reconocido o no.

3.2.2. Requisitos no funcionales

- Tiempo de respuesta inferior a dos segundos.
- Robustez: El sistema debe estar dispuesto ante caídas ya que el servicio debe mantenerse en ejecución para que los vehículos puedan acceder o salir del parking.
- Mantenibilidad: El sistema se ejecutará modularmente para que su mantenimiento y mejora suponga el menor coste en horas y esfuerzo posible.

3.2.3. Diagramas de casos de uso

Se incluyen los diagramas de casos de uso que los usuarios pueden realizar:

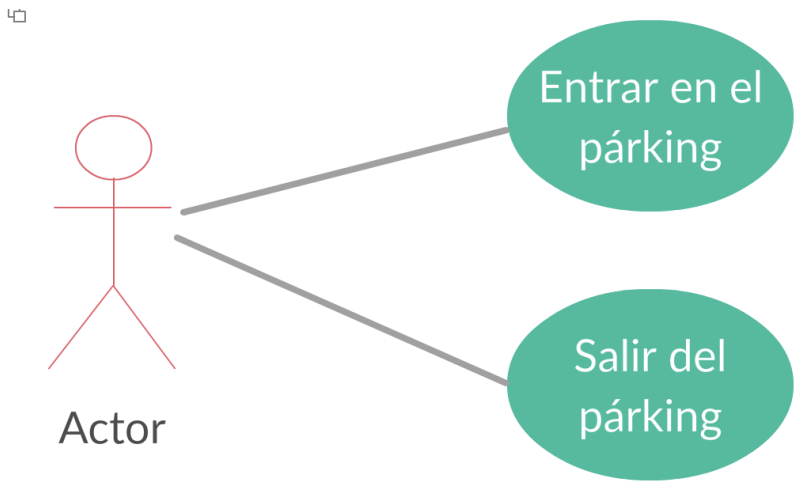

Figura 3.1: Diagrama de casos de uso

3.2.4. Diagrama de componentes

Se incluye el diagrama de componentes del sistema:

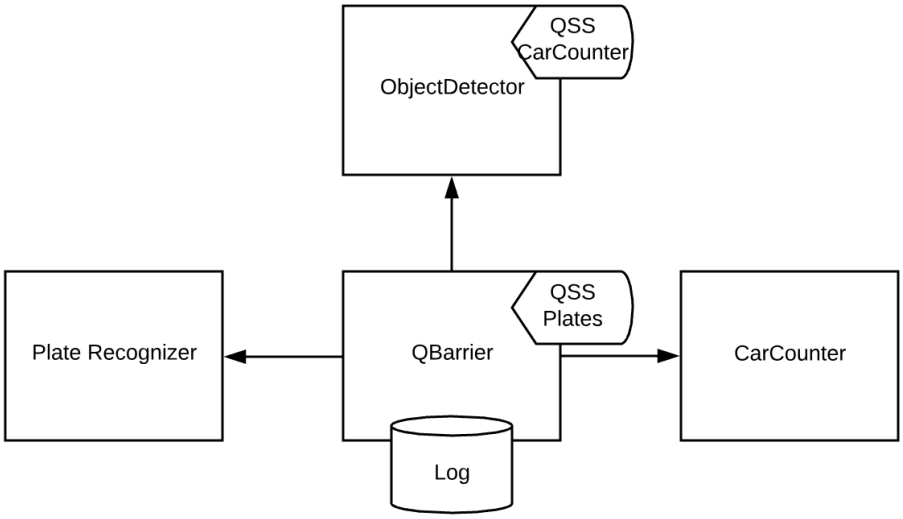

Figura 3.2: Diagrama de componentes

3.2.5. Diagramas de clases

Se incluyen los diagramas de clases de los distintos módulos del proyecto: [1]

Object Detector

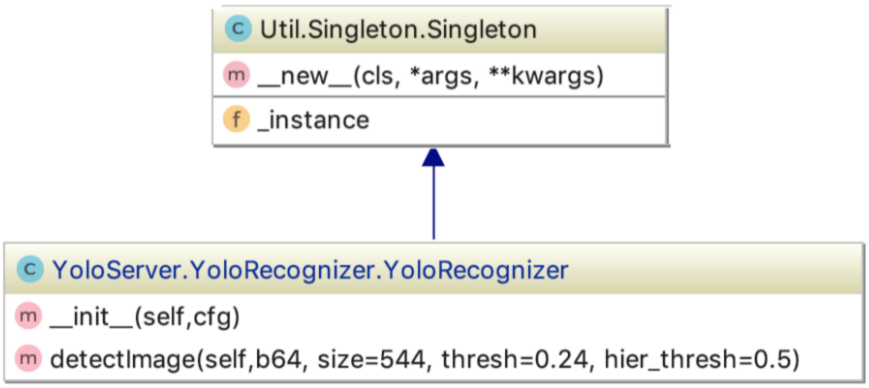

Figura 3.3: Diagrama de clases de Object Detector

[1] Además de las clases que se ven en el diagrama se utilizan scripts para inicializar los procesos, los cuales no se incluyen en dichos diagramas.

Plate Recognizer

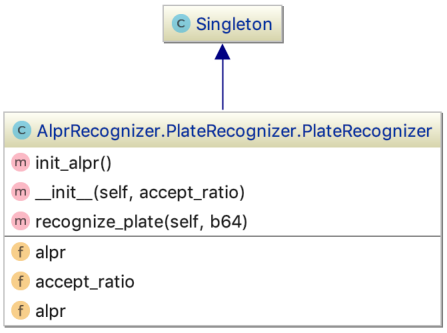

Figura 3.4: Diagrama de clases de Plate Recognizer

Car Counter

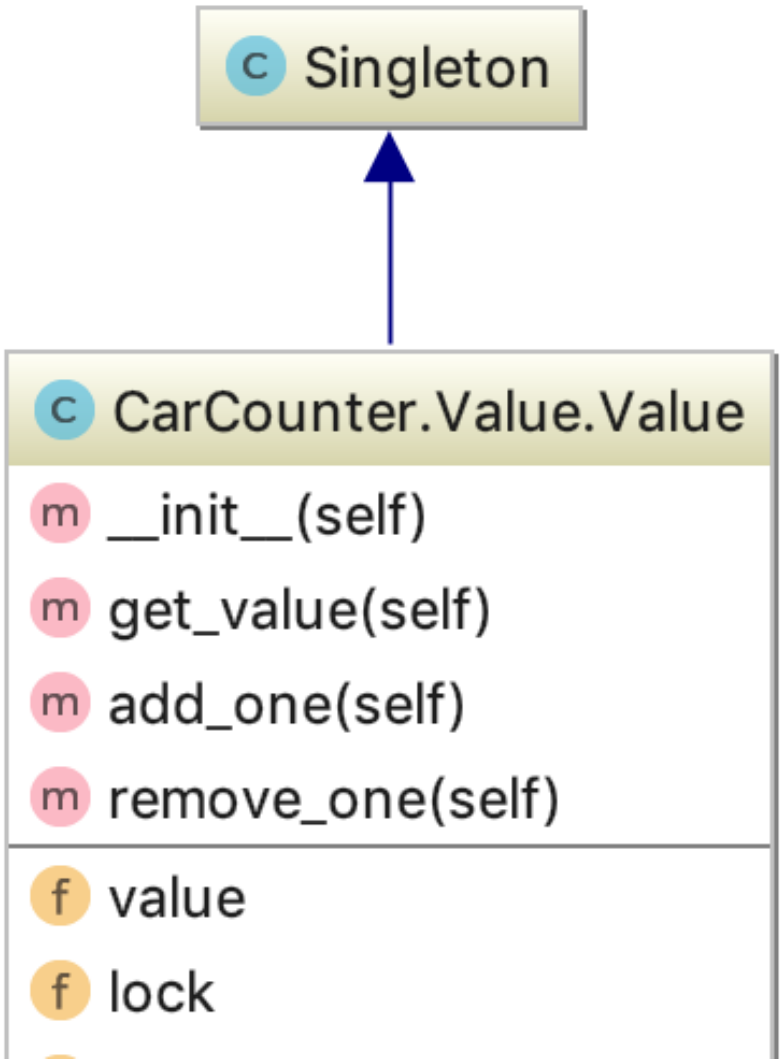

Figura 3.5: Diagrama de clases de Car Counter

QBarrier

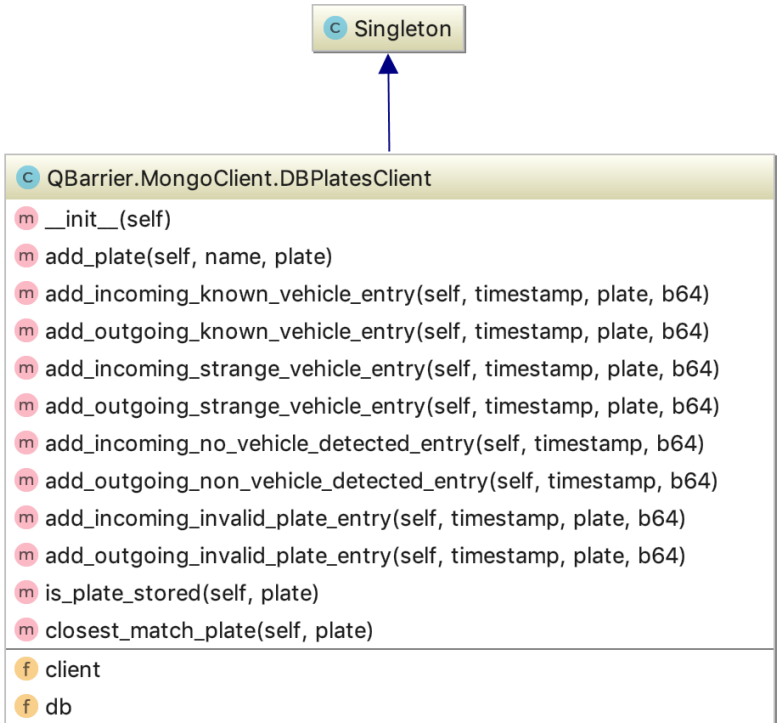

Figura 3.6: Diagrama de clases de MongoClient

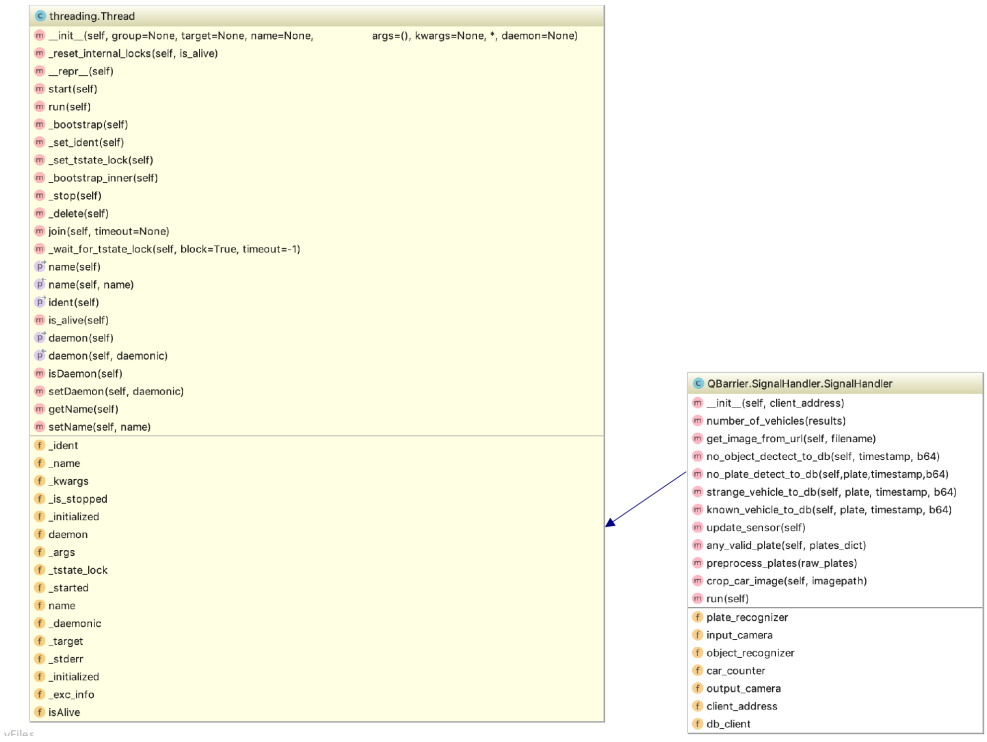

Figura 3.7: Diagrama de clases de QBarrier

3.2.6. Hardware

En esta sección se detallarán los dispositivos que se han utilizado y se dividirán en dos secciones, el terminal en el que se ejecutan todos los módulos y los periféricos que se conectan a estos módulos:

Terminal

Todos los módulos citados anteriormente consumen relativamente pocos recursos a excepción del ObjectRecognizer, el cual debe mover la red neuronal y sin soporte de GPU tarda 5s aprox. en realizar una predicción, un tiempo de espera de usuario inviable. Por tanto, se adquirió una tarjeta gráfica de marca Nvidia para, utilizar el soporte de CUDA y acelerar el proceso de predicción. De entre todo el abanico de posibilidades se escogió una GTX 1060[16] de

6GB GDDR5 reduciendo el tiempo de respuesta de dicha predicción un 95 % del tiempo inicial.

Periféricos

Para controlar la entrada y salida del parking, se dispone de una cámara para cada uno de los sentidos, una AXIS Q1614[17] que dispone de una resolución 720p y una AXIS Q1602[18] con una resolución 480p, esta diferencia de resolución añadida a la orientación de las cámaras, la primera con el sol lateral y la segunda enfrentada al sol durante el atardecer, hará que la calidad de las imágenes sea muy dispar como se puede ver:

Figura 3.8: Imagen tomada desde la cámara AXIS Q1614

Figura 3.9: Imagen tomada desde la cámara AXIS Q1602

Capítulo 4

Propuesta de solución

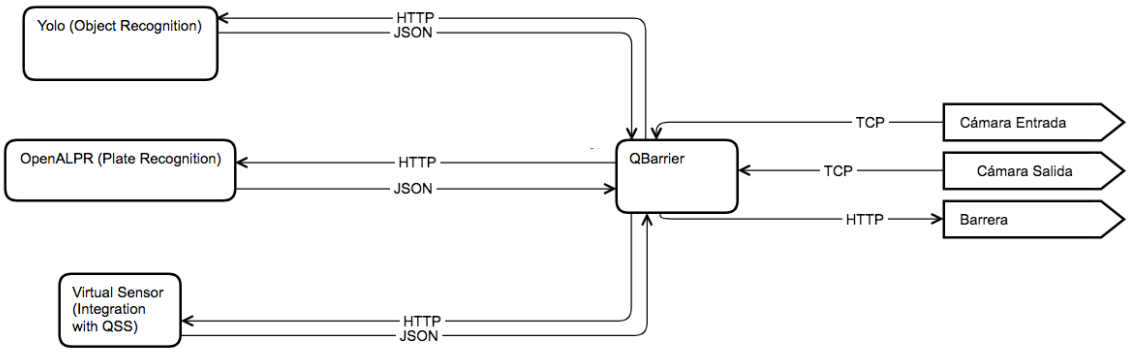

Figura 4.1: Imagen Arquitectura general del proyecto

A partir de la imagen, comenzará la explicación de la propuesta, comenzando desde una vista completa del sistema hasta ahondar en cada uno de los módulos detalladamente.

4.1. Sistema general

Como se puede apreciar en la imagen 4.1, el sistema esta compuesto por 4 componentes que posteriormente se detallará más ampliamente. QBarrier, el módulo principal, que se interconecta con los 3 restantes, el que se encarga de la detección de objetos, Yolo, el de las matrículas, OpenALPR y el sensor virtual que indica el número de coches en el parking, Virtual Sensor.

Todas estas conexiones se realizan mediante HTTP y sus métodos básicos y las respuestas se realizan mediante JSON, una notación de datos muy común y utilizada en el ámbito web. Todos los módulos incluyen un archivo *api.cfg* [1] en el que se especifican los sockets por los que cada módulo además de algunas configuraciones adicionales para cada uno.

[1] Los archivos .cfg del proyecto están detallados en el tanto su ubicación como su contenido

4.2. ObjectDetector

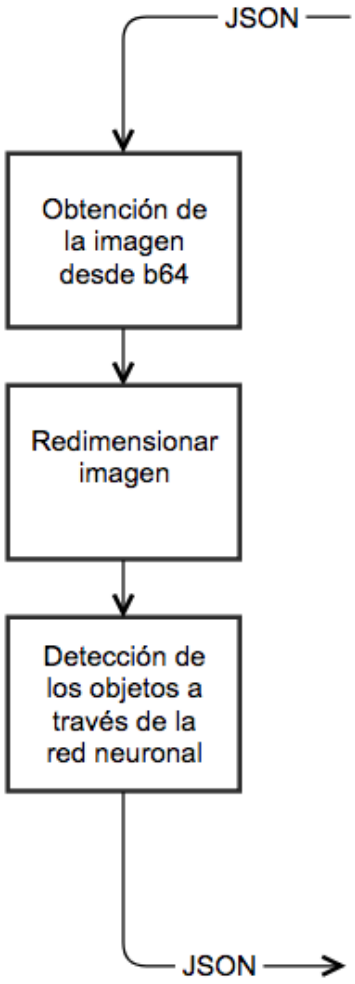

Figura 4.2: Esquema de funcionamiento de ObjectDetector

A partir de esta imagen se pueden diferenciar las distintas tareas que se realizan en este módulo en línea temporal:

1. Del cuerpo de la petición se extrae la imagen en ***base64*** la cual se guarda para ser tratada.
2. La imagen se redimensiona a 640x480 para adaptarla a la entrada de la red neuronal.
3. Cuando la imagen se adapta a la entrada de la red neuronal, que previamente ha sido iniciado con el archivo de pesos que se incluye en la web, se predice los objetos que existen en esa imagen.
4. Las predicciones se convierten a JSON y se envían como respuesta a la petición realizada.

Este módulo se nutre del archivo yolo.cfg, para indicar los archivos de configuración y de pesos.

4.3. PlateRecognizer

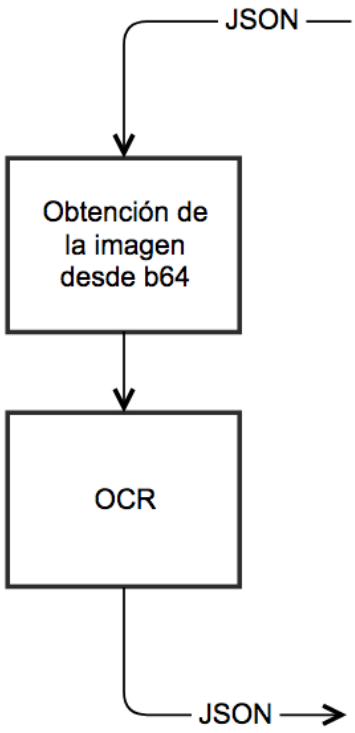

Figura 4.3: Esquema de funcionamiento de PlateRecognizer

El proceso de este módulo es similar al del módulo anterior, se recoge la imagen a partir del código en ***base64*** de la petición HTTP. Cuando se obtiene la imagen, utilizando la interfaz que ofrece OpenALPR para realizar OCR. Cuando se han obtenido las predicciones se convierten a JSON y se envían como respuesta a la petición correspondiente.

Para inicializar este módulo necesita del runtime y de la configuración que están especificadas en el archivo *alpr.cfg*.

4.4. CarCounter

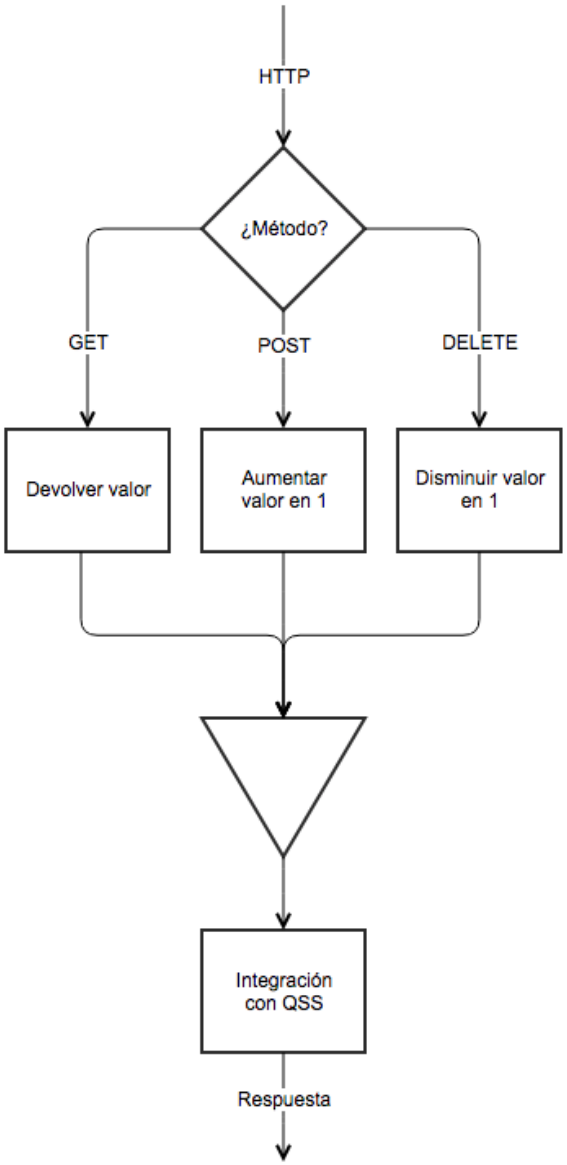

Figura 4.4: Esquema de funcionamiento de CarCounter

Dentro de este apartado, se verá mas en detalle la implementación del sensor virtual que simboliza la cantidad de coches que están estacionados en

el parking.

1. Como se ha especificado en la introducción del capítulo, todos los módulos están comunicados con el protocolo HTTP. Dependiendo del método que se llama se realiza una acción u otra:

   a) **GET**: Se devuelve el valor actual del sensor.

   b) **POST**: Se aumenta en 1 el valor del sensor y se devuelve el valor actualizado.

   c) **DELETE**: Se disminuye en 1 el valor del sensor y se devuelve el valor actualizado.

2. Además, se puede apreciar en el diagrama el sensor está integrado en la interfaz de QSS [19]. Este sensor se incluye como un valor del panel del control de dicha herramienta, pudiendo visualizar el valor del sensor en tiempo real.

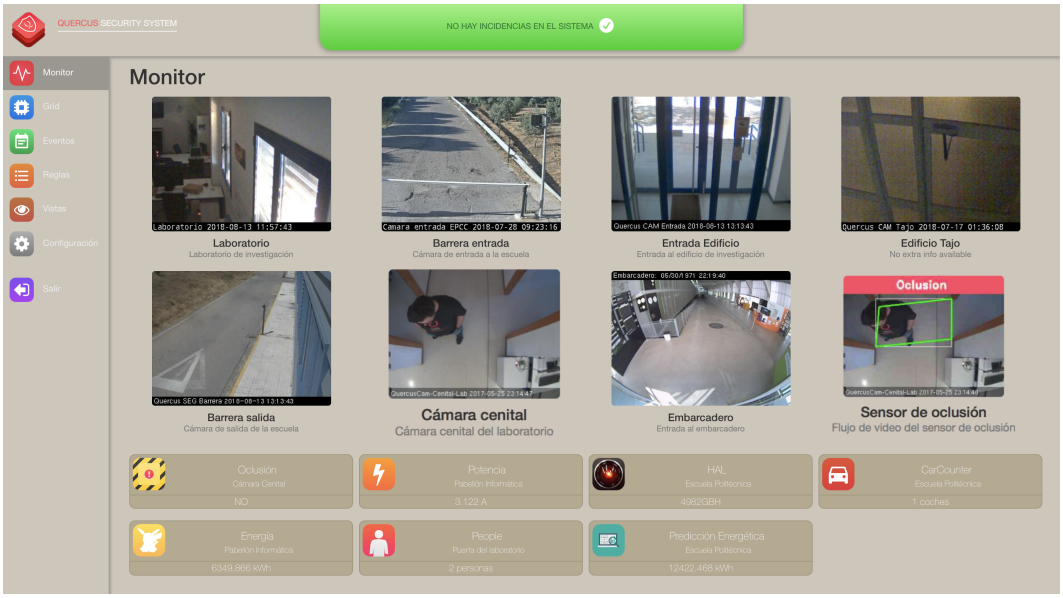

Figura 4.5: Visual de QSS

Figura 4.6: Sensor CarCounter en QSS

El valor del sensor se almacena en un archivo binario que se actualiza simultáneamente en cada actualización del valor volátil por si el sistema sufre alguna caída.

4.5. QBarrier

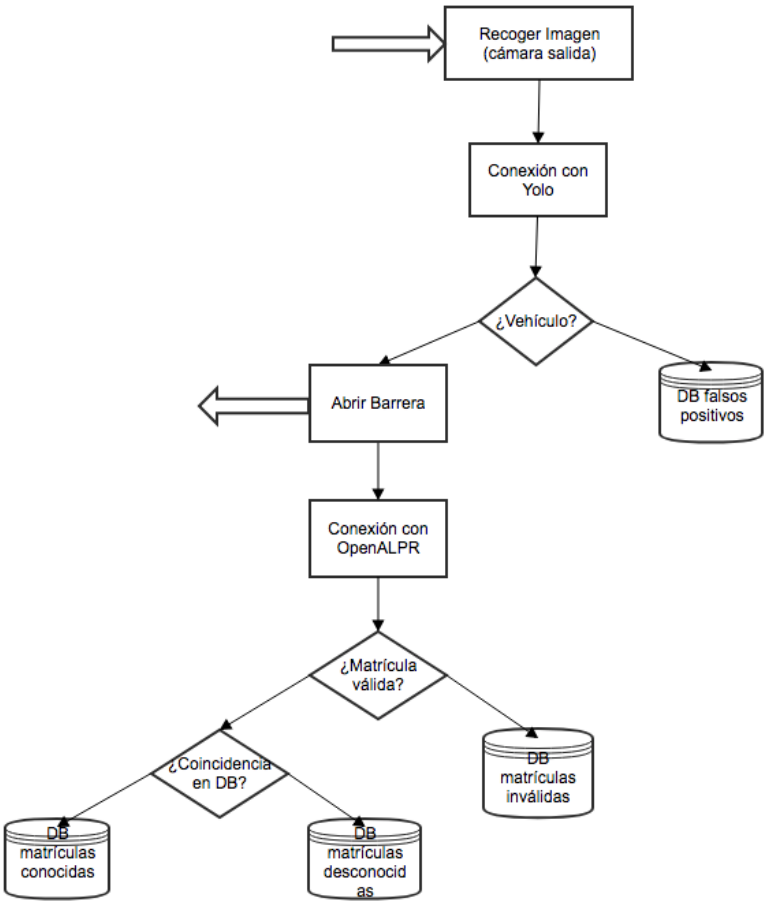

Figura 4.7: Funcionamiento de QBarrier en la cámara de salida

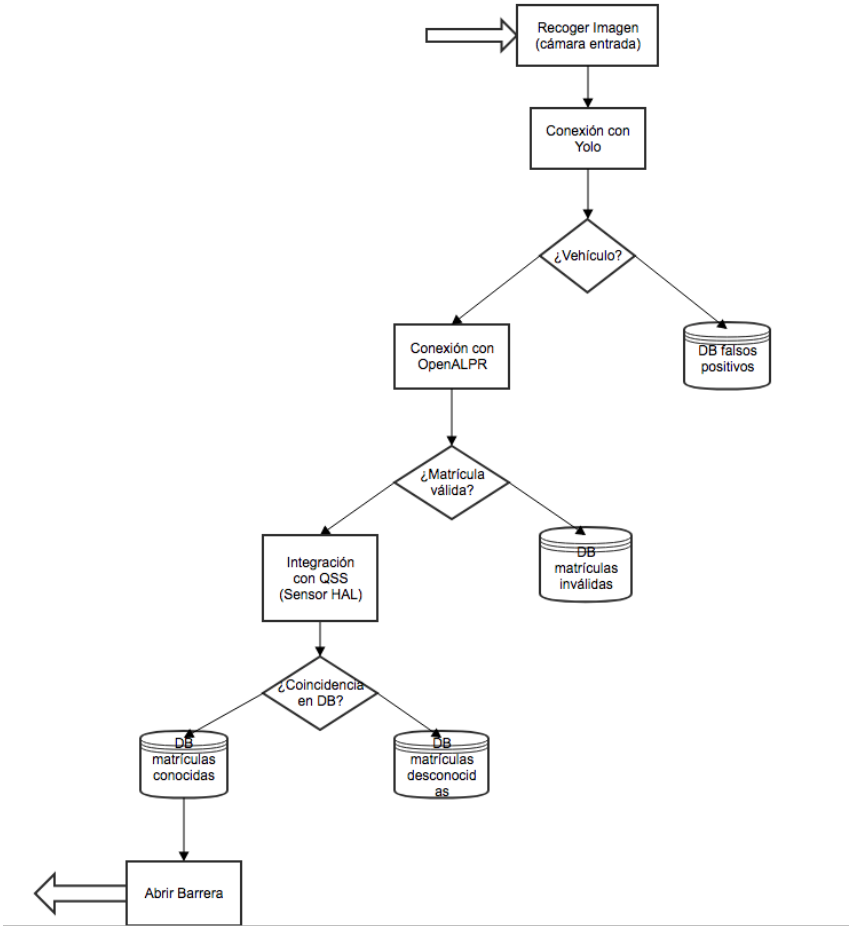

Figura 4.8: Funcionamiento de QBarrier en la cámara de entrada

En este apartado se centra en el módulo principal del proyecto, el que se interconecta con los demás y añade la lógica de negocio.

Para iniciar el proceso, se levanta el servidor TCP por el que las cámaras que están configuradas para enviar peticiones cuando se detecta movimiento en su rango de visión. Cuando llega una petición se crea un hilo que realiza las veces de manejador del evento que acabe de ocurrir.

Además de inicializar las direcciones de los diferentes servicios a los que debe acceder y crear el cliente con el que se comunicará con la base de datos (éste se detallará en el apartado 4.5.2), el proceso que sigue es el siguiente:

1. Se descarga la imagen desde la cámara desde la que se ha realizado la petición.
2. Dicha imagen se convierte a base64 para realizar las peticiones a las distintas APIs
3. Se realiza una petición HTTP de tipo POST al servicio de detección de objetos llevando en el cuerpo de la petición la imagen en base64.
   a) Si en el resultado de la consulta contiene la existencia de un vehículo en la imagen continúa el proceso.
   b) En caso contrario, se anota en la base de datos un falso positivo (Vehículo no encontrado).
4. Se recorta la imagen en un ROI para mejorar la tarea de OCR.
5. Se realiza una petición HTTP de tipo POST al servicio de reconocimiento de texto en imágenes con el código base64 en el cuerpo de dicha petición.
6. A partir del resultado de la petición, comienza el procesado de las matrículas, para solucionar posibles errores en la detección de texto. Dicho proceso consta de dos fases:
   a) Primeramente se contrasta la estructura de la cadena de texto contra una expresión regular:

   $$([0-9]4)([A-Z]3)$$

   Las matrículas que no sigan dicha expresión regular, son desechadas. Si no existe ninguna matrícula correcta se anota una entrada de matrícula inválida en la base de datos (Caso erróneo)
   b) Posteriormente, se contrasta con la base de datos si alguna de las matrículas candidatas existe en ella. Se realiza una búsqueda por coincidencia, que debe ser mayor a un 0,85, es decir, que todos los caracteres excepto uno coinciden en valor y orden con la matrícula ideal, para subsanar algún posible error en la detección de texto.
7. A partir del pre procesado, se abre la siguiente alternativa:
   a) Si coincide alguna matrícula continúa el proceso.
   b) Si no coincide se guarda en la base de datos como una entrada de matrícula desconocida (Caso de éxito).
8. Se abre la barrera al conductor con vehículo de matrícula conocida y se anota en la base de datos una entrada de vehículo conocido (Caso de éxito).

Este proceso es aplicable al evento que se lanza cuando se detecta un coche en la cámara de entrada (para dejar pasar a las personas conocidas), y referente a la cámara de salida, cambia un hecho en el proceso, cuando detecta un coche en la cámara (Paso 3), se abre la barrera para no hacer esperar al usuario.

4.5.1. Sensor Virtual HAL

Como se puede apreciar en la imagen que representa el proceso anterior, este módulo también se integra en QSS, junto a CarCounter. Este sensor está denominado como HAL en el panel web.

Figura 4.9: Sensor de matrículas incrustado en QSS

A partir del paso 7 del apartado anterior, si la matrícula es reconocida por la base de datos, se actualiza el valor del sensor a la última matrícula reconocida y en caso contrario, el sensor dará la alarma (poniéndose en rojo) además de actualizar el valor para que un operario solucione la incidencia.

Figura 4.10: Sensor de matrículas incrustado en QSS

4.5.2. Estructura de datos del log

Para guardar un histórico de los falsos negativos (cuando no se detecta un vehículo en una foto correcta, o cuando no se detecta de uno de éstos), falsos positivos (la cámara detecta movimiento y no se halla ningún vehículo). Además esta información está dividida para las dos cámaras, para detectar un mayor número de ocurrencias en una de otra. Se distinguen así las siguientes colecciones:

- Vehículos conocidos en la cámara de entrada: Imagen(String), timestamp(String), matrícula(String)
- Vehículos conocidos en la cámara de salida: Imagen(String), timestamp(String), matrícula(String)
- Vehículos no conocidos en la cámara de entrada: Imagen(String), timestamp(String), matrícula(String)
- Vehículos no conocidos en la cámara de salida: Imagen(String), timestamp(String), matrícula(String)
- Vehículos sin matrícula en la cámara de entrada: Imagen(String), timestamp(String)
- Vehículos sin matrícula en la entrada de salida: Imagen(String), timestamp(String)
- No vehículos detectados en la cámara de entrada: Imagen(String), timestamp(String)
- No vehículos detectados en la cámara de salida: Imagen(String), timestamp(String)

Cabe destacar el hecho de haber guardado las imágenes referentes a cada uno de estos eventos en el log, además de para validar el funcionamiento de la herramienta, para realizar un análisis de dichos y obtener una posible información relevante.

Capítulo 5

Metodología

Durante las fases de análisis e implementación se siguió una metodología ágil basada en SCRUM y KANBAN, desarrollado en sprints de 1 semana, utilizando la pizarra que ofrece la herramienta Jira para gestionar las distintas tareas que se planificaban y evaluando en estos dichos sprints si se habían cumplido o no.

El desarrollo ha sido iterativo e incremental, realizando una primera implementación de un producto mínimo viable y aumentando la funcionalidad de éste hasta llegar al estado final.

Como repositorio de código se ha utilizado BitBucket ya que permite la creación de repositorios privados de forma gratuita.

En esta imagen se puede ver el flujo de indicendias creadas en la pizarra durante el transcurso del tiempo, ya sea durante la fase de investigación tanto como la de implementación:

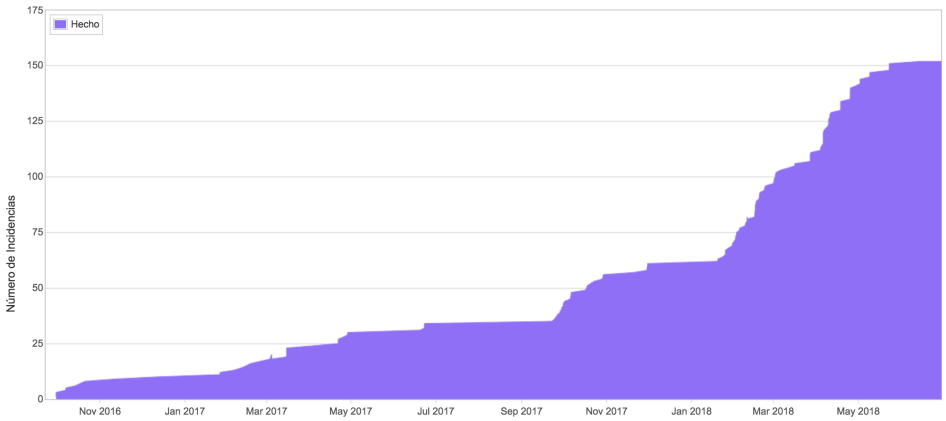

Figura 5.1: Incidencias creadas durante la implementación del proyecto

Como se puede apreciar en la figura A.3 la fecha de comienzo, especialmente en las fase, en la investigación sobre las tecnologías, y dejando la fase de implementación del proyecto sobre Marzo/Abril en la que se evidencian una mayor cantidad de indicendias en Jira.

Capítulo 6

Validación del proceso

A continuación se proponen una serie de casos representativos de los diferentes que han ocurrido en la ejecución del proyecto:

6.1. Caso positivo

Figura 6.1: Caso positivo de imagen

```
6|QBarrier | WARNING:root:2018-06-08 11:30:50: Getting image from
    158.49.245.74
2|Yolo    | Cam frame predicted in 0.038999 seconds.
4|PlatesSe | 158.49.245.86 - - [03/Jul/2018 09:20:01] "POST /
    HTTP/1.1" 200 -
6|QBarrier | WARNING:root:['4306BXL', '4306BX', 'U4306BXL']
6|QBarrier | WARNING:root:Plate valid
6|QBarrier | WARNING:root:Sending 5#1#0##4306BXL to QSS
6|QBarrier | WARNING:root:db plate
6|QBarrier | WARNING:root:Abriendo barrera
```

6.2. Caso en el que no se detecta vehículo

Figura 6.2: Falso positivo como resultado del movimiento de la cámara

```
6|QBarrier | WARNING:root:2018-07-09 21:06:12: Getting image from
    158.49.245.73
2|Yolo    | Cam frame predicted in 0.054776 seconds.
6|QBarrier | WARNING:root:No object detected
```

6.3. Caso en el que no se detecta matrícula

De este tipo no se han encontrado ejemplos significativos para mostrar en esta sección ya que si se detectan vehículos en la zona de interés en un 100 % de los casos se realiza el reconocimiento de caracteres de forma correcta.

6.4. Conclusiones finales sobre los resultados obtenidos

Como resultado final de la validación del sistema se ha contabilizado la cantidad de aciertos y fallos proporcionados por el sistema (casos en los que habiendo un vehículo esperando a entrar en el parking no se ha detectado que es un vehículo o que no se ha detectado su matrícula correctamente) partiendo de una batería de 200 pruebas en todos los rangos lumínicos, eliminando de este recuento las señales de movimiento falsas que ofrece las cámaras en modo nocturno las cuales aumentan significativamente el número de imágenes en las que se detecta movimiento y no se evidencia ningún vehículo.

Los resultados obtenidos son los siguientes:

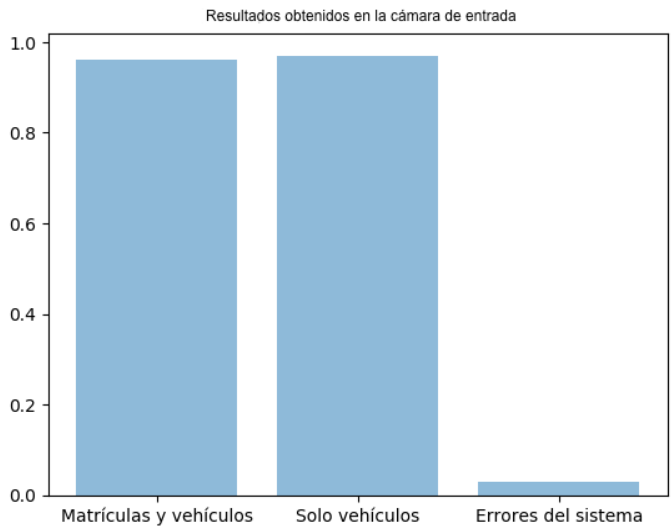

Figura 6.3: Resultados obtenidos en la cámara de entrada

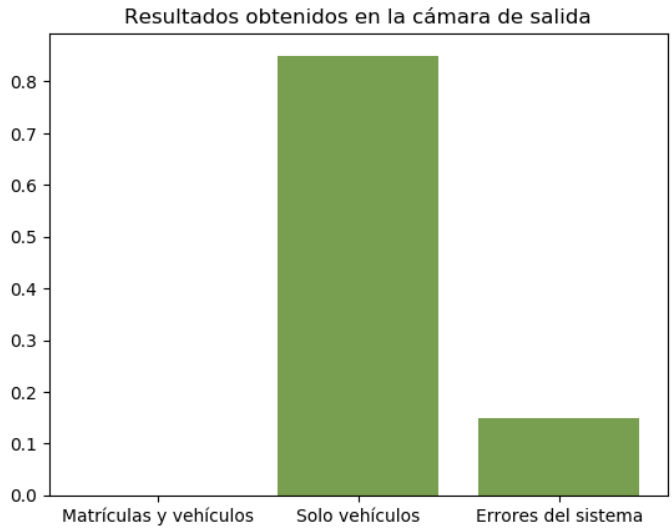

Figura 6.4: Resultados obtenidos en la cámara de salida

Los gráficos arrojan muy buenos resultados sobre la cámara de entrada ya que se evidencian un 95 % de aciertos sobre el sistema global, recayendo los errores repartidos en un 3 % a a la detección de objetos y el 2 % restante al OCR. Por otra parte, se puede comprobar que la calidad de imagen de la cámara de salida afecta a los resultados de nuestro sistema siendo la detección de objetos mínimamente viable con un 85 % pero el OCR no se realiza de forma idónea en ninguno de los casos que se han contabilizado, lo que denota que se debería cambiar dicha cámara para conseguir un funcionamiento correcto.

Capítulo 7

Conclusiones

Como conclusiones del proyecto, se puede recalcar los resultados favorables del proyecto, consiguiendo un 95 % de porcentaje de acierto de media, y consiguiendo satisfacer todos los requisitos tanto funcionales como no funcionales satisfactoriamente ya que durante la realización del proyecto hubo que realizar varios ajustes en los umbrales de movimiento de la cámara y en los de detección para conseguir unos resultados óptimos en la detección de objetos en fases en las que la luz escaseaba.

Llevar a cabo este proyecto ha sido un reto para mí, principalmente por no basarse en contenido explícitamente impartido en el grado. Sin embargo, enfrentarse a un problema real existente en la universidad y solucionarlo siguiendo una metodología me ha causado una gran satisfacción.

Dentro del apartado de las tecnologías, quiero recalcar el lenguaje de programación Python, ya que es uno de los lenguajes más utilizados en la actualidad en una gran cantidad de ámbitos, y el soporte que he recibido por parte de la comunidad además del buen ambiente ha sido excepcional. Como prueba de ello, he creado y mantenido una comunidad de Python en Extremadura a la que asistían unas 60 personas recurrentemente y en la que hemos creído tanto profesional como moralmente. Dentro del apartado tecnológico creo importante el estudio de las distintas APIs de detección de objetos ofrecen un gran abanico de posibilidades a la hora de realizar cualquier desarrollo que utilice imágenes.

Estoy bastante contento con el desarrollo final ya que puede extrapolar-

se de forma muy sencilla a cualquier parking y con una inversión en hardware mínima se pueden obtener unos resultados como los observados durante la realización del proyecto.

Capítulo 8

Trabajos futuros

8.1. Investigación sobre nuevas API de reconocimiento de objetos y OCR

Con el tiempo surgen nuevas APIs de terceros y nuevas herramientas de reconocimiento de imágenes, como las de Google [20] o el de Amazon [21], así que se puede realizar un trabajo de investigación sobre estas herramientas y comprobar si están dentro un rango de precios asumible y consiguen unos resultados mejores a los obtenidos con las que se han investigado en este trabajo.

8.2. Detección de vehículos especiales

Con el objetivo de permitir el acceso a vehículos con necesidades especiales, tales como coches de policía, ambulancias, etc, se podría entrenar el modelo de detección para que en caso de encontrarse con alguno de estos vehículos, abriera la barrera directamente sin esperar a validar su matrícula ya que la acudida de estos vehículos suele ser por motivos urgentes.

8.3. Cálculo de horas de cada vehículo dentro de las instalaciones

Los datos almacenados en el log se pueden utilizar para extraer información sobre ellos, por ejemplo calcular momentos durante el día de entrada y salida masiva de vehículos y de las personas que más asisten a la universidad o las que más tiempo pasan.

8.4. Detección de matrículas en lista negra

Se puede extender la funcionalidad del proyecto para generar algún tipo de notificación especial para la llegada o salida de aquellos vehículos cuya matrícula se determine en una lista negra, esto mejoraría la seguridad de la barrera frente a agentes externos.

Capítulo 9

Agradecimientos

Quiero agradecer a todas y a cada una de las personas que me han acompañado en este viaje que han sido estos cuatro años pero en especial:

- Gracias a todos los profesores por hacerme crecer como profesional y como persona.
- Gracias a mi familia por todo el apoyo y por creer en mí cuándo ni yo mismo lo hacía.
- Gracias Laura por aguantarme en los momentos en los que este camino ha sacado lo peor de mí.
- Gracias a mis amigos y mi segunda familia Víctor, Paula, Mario, Justo y Javi por todas esas tardes (y desgraciadamente algunas noches) de buenos momentos en el laboratorio.
- Gracias Peri por todos esos consejos positivos y por la todopoderosa bilis.
- Gracias a Fernando por los memes.
- Gracias al zulo porque pasando esa puerta, siempre te brindaban un momento de desconexión.
- Gracias a Enrique Mogul por ofrecer tu hombro en todos los aspectos de mi vida. Siempre ha sabido como ayudarme y quiero que sepa, por escrito, que siempre tendrá el mismo apoyo por mi parte.

- Y por último, gracias a Juan por creer en mí y ofrecerme la mayor oportunidad de mi vida, en la que he aprendido todo lo que sé hasta ahora, y por ser la definición perfecta de líder.
- Ah, y gracias Chema.

Apéndice A

Manual del programador

A.1. Dependencias

En primer lugar, para desplegar nuestro propio sistema es necesario instalar una serie de dependencias, las cuales son listadas a continuación:

- Sistema Operativo Linux
- CUDA
- Python 2.7. Además se requieren las siguientes librerías:
  - OpenALPR
- Python 3.5+ Además se requieren las siguientes librerías:
  - PIL
  - requests
  - Flask
  - pymongo
  - shutil
  - logging
- OpenCV 2.7
- Yolo

A.2. Instalación de Cuda

1. Para instalar CUDA se necesita instalar previamente build-essential:

```
$ sudo apt-get install build-essential
```

2. Creamos un directorio para extraer todos los archivos necesarios(En este caso se utiliza la versión 7.5.18, pero es compatible con cualquiera): Enlace de descarga: https://developer.nvidia.com/cuda-downloads

```
$ mkdir ~/Descargas/nvidia_installers;
$ cd ~/Descargas
$ ./cuda_7.5.18_linux.run
    -extract=~/Descargas/nvidia_installers;
```

3. Eliminamos todas las versiones de cuda anteriores debido a problemas de compatibilidad:

```
$ sudo apt-get --purge remove nvidia-*
```

4. Eliminamos el archivo:

```
$ sudo rm /etc/X11/xorg.conf
```

5. Editamos o creamos el archivo /etc/modprobe.d/blacklist-nouveau.conf añadiendo estas dos líneas:

```
blacklist nouveau

options nouveau modeset=0
```

6. Ejecutamos:

```
$ sudo update-initramfs -u
```

7. Reiniciamos el PC

8. Iniciamos sesión sin utilizar la interfaz gráfica pulsando Ctrl+alt+F1 en la ventana de login, y añadimos las credenciales de usuario

9. Entramos en la carpeta donde han sido extraído los archivos y ejecutamos:

```
$ cd ~/Descargas/nvidia_installers;
$ sudo service lightdm stop
```

10. Instalamos el driver conveniente según la versión de Ubuntu:

    - Para Ubuntu 14.04:

      ```
      $ sudo ./NVIDIA-Linux-x86_64-352.39.run --no-opengl-files
      ```

    - Para Ubuntu 16.04:

      ```
      $ sudo ./NVIDIA-Linux-x86_64-367.27.run --no-opengl-files
      ```

11. Instalamos el toolkit de CUDA:

    ```
    $ sudo ./cuda-linux64-rel-6.0.37-18176142.run
    $ sudo ./cuda-samples-linux-6.0.37-18176142.run
    ```

12. Volviendo a la carpeta personal, editamos el archivo .bashrc añadiendo estas dos líneas:

    ```
    export PATH=/usr/local/cuda-7.5/bin:$PATH
    export
       LD_LIBRARY_PATH=/usr/local/cuda-7.5/lib64:$LD_LIBRARY_PATH
    ```

13. Arrancamos la interfaz gráfica ejecutando:

    ```
    $ sudo service lightdm start
    ```

14. Comprobamos la versión de cuda con:

    ```
    $ nvcc -V
    ```

A.3. Estructura del código

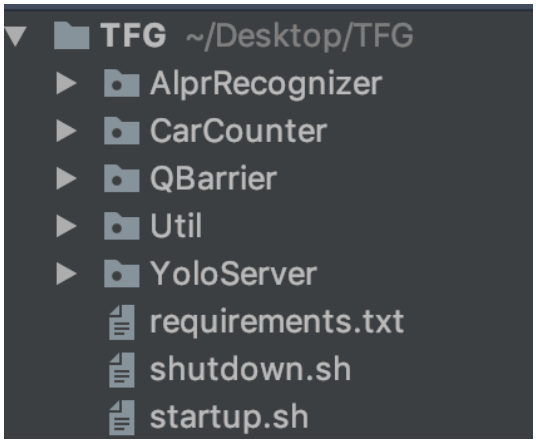

Figura A.1: Estructura general del código del proyecto

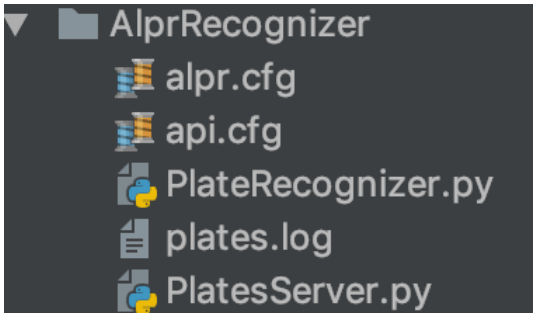

Figura A.2: Estructura del código del módulo Plate Recognizer

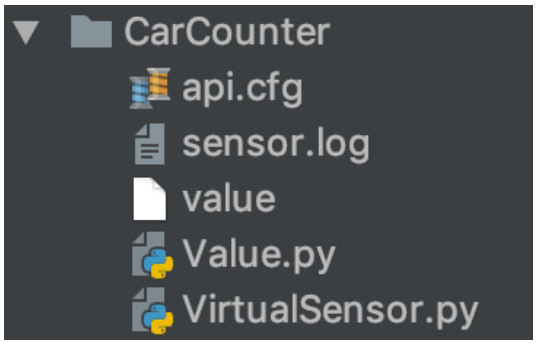

Figura A.3: Estructura del módulo Car Counter

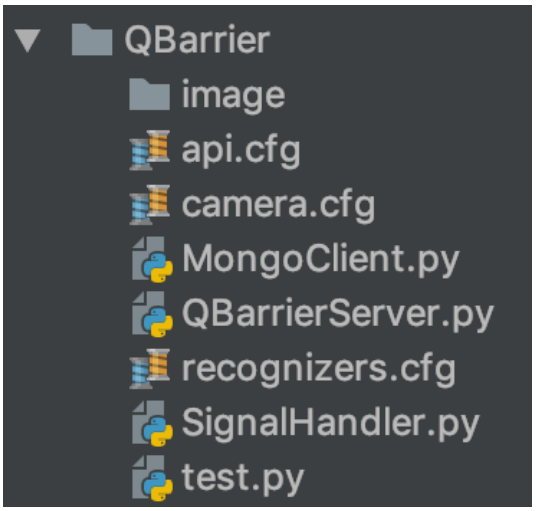

Figura A.4: Estructura del código de QBarrier

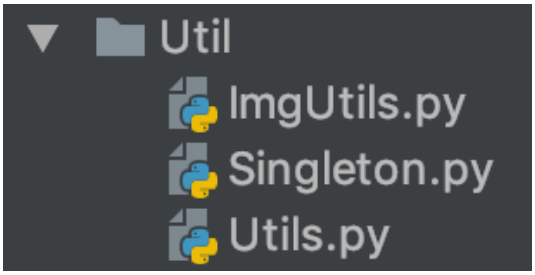

Figura A.5: Estructura del código de la librería Util

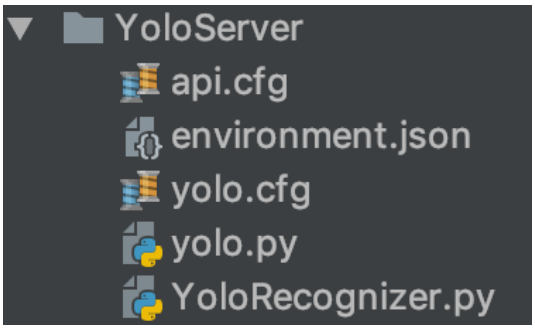

Figura A.6: Estructura del código del módulo Object Detector

A.4. Despliegue del sistema

Para desplegar, el sistema se deben desplegar los 4 módulos por separado:

A.4.1. ObjectDetector

Para desplegar el detector de objetos se deben rellenar los ficheros api.cfg indicando la dirección donde se desplegará el servicio y el yolo.cfg indicando la ruta de los archivos de configuración y de pesos necesarios para ejecutar Yolo. Por último se debe ejecutar el siguiente comando en la raíz de la carpeta del módulo:

```
python3 yolo.py
```

A.4.2. PlateRecognizer

Con el objetivo de desplegar este servicio se deben insertar en los archivos api.cfg la dirección donde escuchará peticiones y en el alpr.cfg la ruta del runtime del motor OCR y de los archivos de configuración. Una vez se han rellenado los archivos, se ejecuta el siguiente comando:

```
python PlatesServer.py
```

A.4.3. CarCounter

Este módulo es más sencillo que los dos anteriores ya que solo se debe rellenar el archivo api.cfg indicando la dirección IP y puerto por el que escuchará peticiones, además de ejecutar el comando:

```
python3 VirtualSensor.py
```

A.4.4. QBarrier

Además de rellenar el archivo api.cfg como en los servicios anteriores, hay que especificar en el archivo recognizers.cfg las direcciones de los demás módulos para realizar la interconexión con ellos. Finalmente, para ejecutar el servicio se debe realizar el siguiente comando:

```
python3 QBarrierServer.py
```

Bibliografía

[1] T. M. Mitchell, Machine Learning. McGraw-Hill Scien- ce/Engineering/-Math, 1997.

[2] Qué es Deep Learning y Cómo funciona. https://geographica.gs/es/blog/deep-learning/.

[3] Teuvo Kohonen. An introduction to neural computing. *Neural Networks*, 1(1):3–16, 1988.

[4] Multilayer Perceptron Definition. https://www.techopedia.com/definition/20879/multilayer-perceptron-mlp.

[5] What is a convolutional neural network? https://www.quora.com/What-is-a-convolutional-neural-network.

[6] MySQL definición. https://es.wikipedia.org/wiki/MySQL#Caracter%C3%ADsticas_adicionales.

[7] IBM Watson Visual Recognition. https://www.ibm.com/watson/services/visual-recognition/.

[8] ImageNet Challenge Contest 2013. http://image-net.org/challenges/LSVRC/2013/.

[9] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. (2016 YOLO) You Only Look Once: Unified, Real-Time Object Detection. *Cvpr 2016*, pages 779–788.

[10] Web pjreddie. https://pjreddie.com/darknet/yolo/.

[11] Wikipedia. Tesseract (software), 2018.

[12] GitHub OpenALPR. https://github.com/openalpr/openalpr.

[13] Stephen Diakopoulo, Nick; Cass. Interactive: The Top Programming Languages 2017, 2017.

[14] EquinsaParking. https://equinsaparking.com/soluciones-de-gestion/sistema-de-control-y-gestion-de-aparcamientos/.

[15] I+D3. https://imasdetres.com/sistema-control-accesos-gestion-parking/.

[16] Tarjeta gráfica GTX 1060. http://www.nvidia.es/graphics-cards/geforce/pascal/gtx-1060/.

[17] Cámara IP AXIS-1614. https://www.axis.com/products/axis-q1614.

[18] Cámara IP AXIS-1602. https://www.axis.com/products/axis-q1602.

[19] Quercus Security System. http://qss.unex.es.

[20] Goggle Vision API. https://cloud.google.com/vision/.

[21] Amazon Rekognition. https://aws.amazon.com/es/rekognition/?sc_channel=PS&sc_campaign=acquisition_ES&sc_publisher=google&sc_medium=spanish_rekognition_nb&sc_content=recognition_e&sc_detail=reconocimiento%20de%20imagen&sc_category=rekognition&sc_segment=207052563308&sc_matchtype=e&sc_country=ES&s_kwcid=AL!4422!3!207052563308!e!!g!!reconocimiento%20de%20imagen&ef_id=WyvweQAAAHyTCwNC:20180630103316:s.