

TRABAJO FINAL DE GRADO



GRADO EN TECNOLOGÍAS DE TELECOMUNICACIÓN

**OpenStreetCam: reconocimiento automático de
objetos en imágenes mediante *machine learning*.**

Jose Luis Villaluenga Morán

Plan Estudios: Ingeniería de Sistemas de Telecomunicación

Área: Servicios basados en localización y espacio inteligente.

Enero de 2019

Directores: Anna Muñoz
David Merino



Esta obra está sujeta a una licencia de
Reconocimiento-NoComercial-
SinObraDerivada 3.0 España de Creative
Common

FICHA DEL TRABAJO FINAL

Título del trabajo:	<i>OpenStreetCam: reconocimiento automático de objetos en imágenes mediante machine learning.</i>
Nombre del autor:	<i>Jose Luis Villaluenga Morán</i>
Nombre del consultor/a:	<i>Anna Muñoz Bolas</i>
Nombre del PRA:	<i>David Merino Arranz</i>
Fecha de entrega (mm/aaaa):	<i>01/2019</i>
Titulación:	<i>Ingeniería de Sistemas de Telecomunicación</i>
Área del Trabajo Final:	<i>Servicios basados en localización y espacio inteligente</i>
Idioma del trabajo:	<i>Español</i>
Palabras clave	<i>Machine Learning</i> <i>OpenStreetCam</i> <i>Visión por ordenador</i>

Resumen del Trabajo:

OpenStreetCam es un proyecto cuyo objetivo principal es mapear cualquier localización geográfica mediante las imágenes aportadas por sus usuarios. Una de las características más interesantes que aporta el servicio, la cual ha sido el objeto principal para el desarrollo de este trabajo, es la detección en imágenes de diferentes elementos como señales de tráfico, personas y vehículos.

Para la identificación de las diferentes entidades enumeradas, el sistema utiliza un algoritmo de código abierto que aplica el concepto *Machine Learning* (aprendizaje automático) para la detección automática de objetos. Además del algoritmo con la *I.A (inteligencia artificial)* el sistema utiliza una librería de visión por computador desarrollada por Facebook llamada RetinaNet. Entre sus características destaca su modelo de red, que le confiere unas prestaciones superiores a otros detectores existentes dotándolo de una precisión y velocidad que lo hace idóneo para su uso en aplicaciones en tiempo real.

Este trabajo tiene como objetivo el estudio del funcionamiento del sistema de detección de *OpenStreetCam*, así como la evaluación de su rendimiento y su grado de resolución.

Abstract:

OpenStreetCam is a project whose main objective is to map any geographical location using the images provided by its users. One of the most interesting features which provides the service, which has been the main object for the development of this work, is the detection in images of different elements such as traffic signs, people and vehicles.

To identify the different entities listed, the system uses an open-source algorithm that applies the Machine Learning concept for the automatic detection of objects. In addition to the I.A (artificial intelligence) algorithm, the system uses a computer vision library developed by Facebook called RetinaNet. Among its features stands out its network model, which gives it superior performance to other existing detectors giving it a precision and speed that makes it suitable for use in real time applications.

The objective of this work is to study the functioning of the OpenStretCam detection system, as well as the evaluation of its performance and degree of resolution.

Índice

1. Introducción	1
1.1 Contexto y justificación del trabajo	1
1.2 Objetivos principales.....	2
1.3 Planificación.....	3
1.3.1 Identificación de tareas.....	3
1.4 Descripción del contenido de la memoria.....	8
2. Machine Learning	8
2.1 Software basado en algoritmos <i>ML</i> para reconocimiento de imágenes	10
3. Estado del arte.....	13
3.1 Software basado en algoritmos <i>ML</i> para el reconocimiento de imágenes.	13
3.1.1 Características del software basado en algoritmos <i>ML</i> para el reconocimiento de imágenes.	16
3.2 Soluciones OpenSource.....	16
4. Evaluación del software de Telenav	18
4.1 Especificaciones de OpenStreetCam	19
4.2 Entornos de pruebas	21
4.2.1 Entorno para ejecución de script Unix.....	21
4.2.2 Entorno para ejecución de script Python.....	23
4.2.3 Entorno web.....	24
4.3 Pruebas y análisis de resultados.....	26
5. Fundamentos del algoritmo de detección en OSC	34
6. Funcionamiento del algoritmo <i>Machine Learning</i>	35
6.1 Implementación de las pruebas.	35
6.2 Evaluación de resultados.....	39
6.3 Análisis del algoritmo.....	40
7. Aplicaciones.....	45
8. Trabajo futuro	46
9. Conclusiones.....	48
10. Glosario	49

Anexo I

2. Diagrama de Gantt	I
-----------------------------------	----------

Anexo II

1. Requisitos técnicos	I
2. Análisis de riesgos y plan de contingencia	I

Índice de figuras

Figura 1: Clasificación Machine Learning	10
Figura 2: Esquema de funcionamiento de Mapillary.....	12
Figura 3: Catálogo de aplicaciones de OpenStreetCam	19
Figura 4: Aplicación web de OpenStreetCam.....	20
Figura 5: Fichero con el set de imágenes de entrenamiento de Telenav.....	21
Figura 6: Formulario de registro de OpenStreetMap	25
Figura 7: Opciones de identificación de OSC.....	25
Figura 8: Localización geográfica para ejecución de pruebas	26
Figura 9: Vías identificadas y tracks subidos por los usuarios	27
Figura 10: Pantalla principal de la aplicación	27
Figura 11: Panel de control de OSC	28
Figura 12: Filtros aplicados para la detección de señales	28
Figura 13: Ejecución del algoritmo de detección de señales	29
Figura 14: Identificación incorrecta de señal de giro	29
Figura 15: Identificación incorrecta de semáforo	30
Figura 16: Identificación correcta de varios semáforos	30
Figura 17: Identificación de semáforos	31
Figura 18: Identificación de señal de límite de velocidad	31
Figura 19: Detección de señal de giro prohibido.....	32
Figura 20: Detección incorrecta debido a un elemento del entorno	32
Figura 21: Detección de semáforos en imágenes propias	36
Figura 22: Señales no detectadas en imágenes propias	36
Figura 23: Señales de ceda el paso de EEUU (izquierda) y española (derecha)	37
Figura 24: Detección de señal de ceda el paso	37
Figura 25: Arriba, señales de límite de velocidad en EEUU, abajo, límites en España	38
Figura 26: Señales de límite de velocidad no detectadas por el algoritmo	38
Figura 27: Identificación de señal de stop	39
Figura 28: División de la imagen en regiones o cajas de anclaje	42
Figura 29: Red piramidal en detección de imágenes	44
Figura 30: Arquitectura de red en RetinaNet	45

Índice de tablas

Tabla 1: Resumen funcionalidades de las diferentes APIs de reconocimiento estudiadas....16

Tabla 2: Datos de evaluación del algoritmo33

1. Introducción

Este trabajo pretende evaluar las características de la plataforma OpenStreetCam y, en concreto, la funcionalidad que permite identificar señales de tráfico en las imágenes aportadas por sus usuarios. Para poder evaluar dicha funcionalidad, se estudia el concepto *Machine Learning* que permite implementarla. Posteriormente, antes de evaluar el grado de eficiencia del software de OpenStreetCam, en el estado del arte, se repasan los diferentes proyectos que cubren esa misma funcionalidad. Tras ello, se estudian los fundamentos utilizados por OpenStreetCam y los de la Inteligencia Artificial que implementa la funcionalidad.

1.1 Contexto y justificación del trabajo

En los últimos años han ido apareciendo diferentes herramientas para la elaboración de mapas digitales, esto unido a las mejoras tecnológicas desarrolladas en diferentes etapas, sobretodo con la popularización de los dispositivos GPS, la disponibilidad de imágenes aéreas de alta resolución y, a conceptos como el aprendizaje automático o *Machine Learning*, ha provocado una rápida evolución en esta disciplina. A lo largo de los últimos años se han superado las dos primeras etapas y actualmente nos encontramos en la tercera en la que las *I.A* (Inteligencias artificiales) unidas al procesamiento de grandes volúmenes de datos está permitiendo la aparición de multitud de servicios basados en la localización que serían impensables hace tan sólo unos años.

Entre esas múltiples aplicaciones una de las más destacables es la funcionalidad de OpenStreetCam capaz de identificar señales viales en las imágenes subidas por los usuarios para representar una localización geográfica concreta. Esta aplicación, en concreto, invita a un futuro uso en sistemas de seguridad vial que procese las imágenes en tiempo real, o incluso como parte de un sistema autónomo de conducción.

En este trabajo se estudiarán los fundamentos que implementan este software, cómo se aplican los algoritmos *Machine Learning* para el reconocimiento de elementos viales en imágenes y cuál es su grado de efectividad.

1.2 Objetivos principales.

Un TFG supone la culminación de una extensa etapa de formación y su elaboración cubre diferentes competencias adquiridas en dicha etapa, es por ello que los objetivos que deben cumplirse van más allá del tema concreto que se quiere tratar.

Algunos de los objetivos más importantes que se deben cumplir desde el punto de vista de una de las competencias que más trascendencia tiene en la elaboración como es la gestión de proyectos son:

- Entender y poder aplicar las técnicas principales de análisis y evaluación de proyectos TIC.
- Utilizar los procedimientos y herramientas adecuadas para realizar una planificación del proyecto correcta y eficiente.
- Conocer, planificar y controlar los riesgos del proyecto.
- Conocer, planificar y controlar la calidad en la gestión del proyecto.
- Realización, presentación y defensa, del trabajo realizado individualmente ante un tribunal universitario, consistente en un proyecto integral de Ingeniería de Telecomunicación de naturaleza profesional en el que se sinteticen las competencias adquiridas en las enseñanzas.

Los objetivos relativos a la temática de este TFG son:

- Estudiar y presentar detalladamente el concepto de *Machine Learning* dando a conocer los modelos y tipos algoritmos más comunmente utilizados para su implementación.
- Conocer algunas de las herramientas, como **Mapillary** y **OpenStreetCam**, que aplicando la filosofía *Machine Learning* implementan la capacidad de detección de objetos en imágenes para su posterior georeferenciación.
- Estudiar y probar el funcionamiento del algoritmo *OpenSource* utilizado por *Telenav* para la implementación de **OpenStreetCam**.
 - Recopilar imágenes de ejemplo proporcionadas por el desarrollador.
 - Comprobar el funcionamiento del sistema evaluando el grado de resolución del algoritmo.
- Recopilar y preparar un conjunto de imágenes conocidas para ser procesadas por el algoritmo.
- Evaluar el grado de resolución del algoritmo sobre el nuevo conjunto de datos.
- Analizar la capacidad de procesado de imágenes del sistema evaluando la capacidad de reconocimiento de los diferentes elementos de señalización.

- Establecer, mediante el análisis de resultados obtenidos por las pruebas realizadas, una serie de conclusiones sobre la capacidad del algoritmo *Machine Learning* y, en concreto, de su funcionamiento en **OpenStreetCam**.
- Proponer diferentes líneas de trabajo futuro que permitan mejorar el funcionamiento del algoritmo de reconocimiento.

1.3 Planificación.

En este apartado se presentará la planificación temporal de las diferentes tareas cuya ejecución es necesaria para la consecución de los objetivos del proyecto. Para ello, se presentarán las distintas fases en las que serán llevadas a cabo, así como las actividades ligadas a cada tarea y sus entregables asociados. Adicionalmente se puede consultar el [Anexo I](#) donde se presenta el diagrama de Gantt de la planificación.

1.3.1 Identificación de tareas

Diccionario WBS (Work Breakdown Structure)

1.	Definición del TFG y planificación
Descripción	Se realizan las tareas necesarias para determinar la duración de las distintas actividades del proyecto.
Actividades	Definir las diferentes tareas del proyecto, determinar el esfuerzo necesario para llevarlas a cabo, presentar una planificación especificando las fechas para afrontarlas y los entregables derivados de ellas.
Duración	11 días.
Entregables	PEC 1

1.1.	Búsqueda de información e investigación
Descripción	Se determinará la información básica para afrontar las diferentes tareas del proyecto.
Actividades	Búsqueda en diferentes fuentes de información académica como <i>papers</i> y artículos técnicos.
Duración	2 días.
Entregables	PEC 1

1.2.	Definición de apartados y tareas
Descripción	Se definen las diferentes actividades del proyecto y los apartados que formaran parte de la memoria final.
Actividades	Búsqueda en diferentes Fuentes de información académica como <i>papers</i> y artículos técnicos.
Duración	2 días.

Entregables	PEC 1
--------------------	-------

1.3.	Definición de objetivos
Descripción	Se establecerán los principales objetivos que deberá cumplir el proyecto tras su finalización.
Actividades	Definición de objetivos del proyecto.
Duración	1 día.
Entregables	PEC 1

1.4.	Análisis de requerimientos de software
Descripción	Se evaluarán los requisitos mínimos necesarios para poder ejecutar el algoritmo <i>ML</i> implementado por OpenStreetCam.
Actividades	Definición de requisitos mínimos de software.
Duración	1 día.
Entregables	N/A

1.5.	Asignación temporal de tareas
Descripción	Determinar el esfuerzo necesario para llevar a cabo las diferentes actividades del proyecto.
Actividades	Presentar una planificación especificando las fechas para afrontar las diferentes actividades del proyecto.
Duración	2 días.
Entregables	PEC 1

1.6.	Definición de entregables
Descripción	Se asignarán los entregables correspondientes a las diferentes tareas en caso de haberlos.
Actividades	Definición de documentación de tareas.
Duración	2 días.
Entregables	PEC 1

1.7.	Establecimiento del plan de contingencia.
Descripción	Se definirá un paquete de medidas para mitigar el impacto de los riesgos subyacentes a la ejecución del proyecto.
Actividades	Evaluación de los posibles riesgos que puedan poner en peligro la ejecución del proyecto. Asignación de diferentes medidas con el objetivo de mitigar los riesgos.
Duración	2 días.
Entregables	PEC 1

2.	Investigación y pruebas.
Descripción	Se estudiará el diferente software basado en algoritmos <i>ML</i> con capacidad para detección de elementos en imágenes y se evaluará OpenStreetCam.
Actividades	Estudio de las diferentes soluciones basadas en <i>ML</i> para identificación de objetos sobre imágenes. Recolección de imágenes y diseño del plan de pruebas.

	Evaluación de OpenStreetCam. Análisis de resultados.
Duración	26 días.
Entregables	PEC 2

2.1.	Estado del arte.
Descripción	Estudio del diferente software basado en algoritmos <i>ML</i> para detección de imágenes.
Actividades	Estudio del software <i>ML</i> para identificación de imágenes, descripción de las opciones OpenSource y comparativa con OpenStreetCam.
Duración	10 días.
Entregables	PEC 2

2.1.1.	Software <i>ML</i> de detección de imágenes.
Descripción	Estudio del diferente software basado en algoritmos <i>ML</i> utilizado en reconocimiento de elementos.
Actividades	Búsqueda y síntesis de información.
Duración	5 días.
Entregables	PEC 2

2.1.2.	Selección de software Opensource.
Descripción	Evaluación de las diferentes soluciones OpenSource disponibles.
Actividades	
Duración	3 días.
Entregables	PEC 2

2.1.3.	Comparativa con OpenStreetCam.
Descripción	Comparativa entre las diferentes soluciones del mercado respecto a OpenStreetCam.
Actividades	
Duración	2 días.
Entregables	PEC 2

2.2.	Evaluación de OpenStreetCam mediante ejemplos.
Descripción	Se evaluará el rendimiento del algoritmo <i>ML</i> de OpenStreetCam mediante las imágenes de ejemplo proporcionadas por Telenav.
Actividades	Búsqueda de imágenes de ejemplo.
Duración	4 días.
Entregables	PEC 2

2.3.	Diseño de pruebas.
Descripción	Se diseñará un plan de pruebas para evaluar el grado de acierto del algoritmo.
Actividades	Estructurar plan de pruebas.
Duración	3 días.

Entregables	PEC 2
--------------------	-------

2.4.	Análisis de resultados.
Descripción	Evaluación de los resultados obtenidos.
Actividades	
Duración	4 días.
Entregables	PEC 2

2.5.	Funcionamiento del algoritmo <i>ML</i> en OpenStreetCam.
Descripción	Se elaborarán las conclusiones sobre el funcionamiento de <i>ML</i> en OpenStreetCam a través de los resultados obtenidos.
Actividades	
Duración	7 días.
Entregables	PEC 2

3.	Implementación.
Descripción	Se recopilarán las imágenes propias necesarias para evaluar el funcionamiento de OpenStreetCam y su capacidad de resolución.
Actividades	Recopilación de imágenes válidas para OpenStreetCam. Diseño de plan de pruebas. Ejecución del algoritmo. Evaluación de resultados.
Duración	17 días.
Entregables	PEC 3

3.1.	Recolección de datos propios.
Descripción	Se recopilarán las imágenes propias necesarias para evaluar el funcionamiento de OpenStreetCam.
Actividades	Recopilación de imágenes válidas para OpenStreetCam.
Duración	6 días.
Entregables	PEC 3

3.2.	Ejecución del algoritmo.
Descripción	Se realizarán las pruebas necesarias para evaluar el funcionamiento del algoritmo sobre las imágenes suministradas.
Actividades	Diseño del plan de pruebas, ejecución del algoritmo y validación de resultados.
Duración	7 días.
Entregables	PEC 3
3.3.	Análisis de resultados.
Descripción	Se evaluará la capacidad de resolución del algoritmo en base a las pruebas realizadas.
Actividades	Evaluación de resultados.
Duración	7 días.

Entregables	PEC 3
--------------------	-------

4.	Edición de memoria.
Descripción	Se editará la memoria final del proyecto en la que se documentarán todas las tareas realizadas hasta entonces.
Actividades	Síntesis de resultados obtenidos en las diferentes tareas ejecutadas. Elaboración de conclusiones obtenidas acerca del concepto Machine Learning y su funcionamiento para el reconocimiento de elementos en imágenes. Propuestas de líneas de mejora. Elaboración de la presentación del proyecto. Preparación de la defensa del proyecto.
Duración	20 días.
Entregables	Memoria final

4.1.	Análisis y conclusiones.
Descripción	Análisis extraído a partir de las pruebas realizadas en la investigación y pruebas de las tareas anteriores.
Actividades	Evaluación global extraída de la realización del proyecto. Síntesis de conclusiones.
Duración	5 días.
Entregables	Memoria final

4.2.	Mejoras propuestas.
Descripción	Establecer líneas de trabajo para mejorar la capacidad de resolución del algoritmo.
Actividades	Proponer mejoras que pudiesen ser implementadas en un futuro.
Duración	5 días.
Entregables	Memoria final

4.3.	Presentación del proyecto.
Descripción	Realizar una presentación que sintetice los puntos e ideas más relevantes extraídos de la ejecución del proyecto.
Actividades	
Duración	6 días.
Entregables	Presentación

4.4.	Preparación de la defensa.
Descripción	Extraer las ideas más relevantes derivadas de la ejecución del proyecto.
Actividades	

Duración	4 días.
Entregables	N/A

1.4 Descripción del contenido de la memoria

El primer capítulo de la memoria es introductorio, en el se sitúa el contexto del trabajo y la motivación para su realización, además se detallan los objetivos que deberán cumplirse tras la ejecución del proyecto y cómo se ha de planificar este para lograr llegar a ellos. En el segundo capítulo se estudia el concepto *Machine Learning* y sus principales modelos. A continuación, en el capítulo 3 se repasan algunos de los principales softwares de detección de imágenes disponibles tanto *OpenSource* como comerciales. El capítulo 4, contiene el núcleo del trabajo, en el se estudia el funcionamiento de la plataforma *OpenStreetCam* y se evalúa el funcionamiento del software de detección de imágenes utilizando el juego de imágenes de entrenamiento de Telenav. En el siguiente capítulo se analiza el funcionamiento del algoritmo de detección. En el apartado 6 se evalúa el algoritmo *Machine learning* mediante el uso de imágenes propias. Por último, en el capítulo de aplicaciones se detallan los usos más importantes en los que se está poniendo en funcionamiento el software de detección.

2. *Machine Learning*

El aprendizaje automático o *Machine Learning* es una rama de la inteligencia artificial que tiene como finalidad el desarrollo de técnicas que permiten a un sistema la toma de decisiones de manera autónoma mediante aprendizaje. De forma más concreta, se trata de crear algoritmos con capacidad de generalizar comportamientos a partir de una información suministrada en forma de ejemplos.

Se trata, por tanto, de un proceso de inducción del conocimiento estrechamente relacionado con la estadística mediante el análisis de grandes volúmenes de datos. Sin embargo, el aprendizaje automático también se centra en el estudio de la complejidad computacional de los problemas. El aprendizaje

automático puede verse como un intento de automatizar algunas partes del método científico mediante métodos matemáticos.

Existe una gran variedad de aplicaciones entre las que se incluyen: motores de búsqueda, diagnósticos médicos, detección de fraude en el uso de tarjetas de crédito, análisis del mercado de valores, clasificación de secuencias de ADN, reconocimiento del lenguaje, de imágenes, juegos, robótica...¹

El aprendizaje automático tiene como resultado un modelo para resolver una tarea dada, entre estos modelos están:

- Los **modelos geométricos**, contruidos en el espacio de instancias y que pueden tener múltiples dimensiones. Si hay un borde de decisión lineal entre las clases, se dice que los datos son linealmente separables.
- Los **modelos probabilísticos**, que intentan determinar la distribución de probabilidades que describe la función que enlaza las características con valores determinados. Uno de los conceptos claves para desarrollar modelos probabilísticos es la estadística bayesiana.
- Los **modelos lógicos**, que transforman y expresan las probabilidades en reglas organizadas en forma de árboles de decisión.

Los modelos pueden clasificarse, además, como **modelos de agrupamiento** y **modelos de gradiente**. Los primeros tratan de dividir el espacio de instancias en grupos. Los segundos, como su nombre lo indican, representan un gradiente en el que se puede diferenciar entre cada instancia. Clasificadores geométricos como las máquinas de vectores de apoyo son modelos de gradientes.²

En el aprendizaje automático existen dos tipos de datos a procesar, dando lugar al aprendizaje supervisado y no supervisado. En un problema de aprendizaje supervisado, los datos proporcionan las entradas del problema, tomando las soluciones en los ejemplos anteriormente proporcionados. Con ello se pueden utilizar ciertos algoritmos para obtener patrones de los datos, y así realizar futuras predicciones. En la figura 1 se puede ver esta clasificación y

¹ https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico

² <https://medium.com/soldai/tipos-de-aprendizaje-autom%C3%A1tico-6413e3c615e2>

sus usos más comunes, incluyendo además, el aprendizaje por refuerzo, una variante inspirada en el conductismo, cuya ocupación es determinar qué acciones debe escoger un agente de software en un entorno dado con el fin de maximizar alguna noción de "recompensa".



Figura 1: Clasificación Machine learning
<https://medium.com/ai-learners>

Los algoritmos más usados en el aprendizaje supervisado son las regresiones, las redes neuronales y las máquinas de soporte vectorial (SVM). En un problema de aprendizaje no supervisado, los ejemplos proporcionados únicamente sirven para obtener las entradas del problema, desconociéndose las posibles salidas o soluciones. La técnica de mayor uso en el aprendizaje no supervisado se basa en realizar agrupamientos (clustering).³

2.1 Software basado en algoritmos *Machine Learning* para reconocimiento de imágenes georeferenciadas.

Existen diferentes aplicaciones basadas en algoritmos *Machine Learning* que permiten a los usuarios referenciar imágenes en una localización geográfica concreta. Entre las más importantes están OpenStreetCam y Mapillary, esta última basada en una rama específica del *Machine Learning*, el *deep learning*, que es un conjunto de algoritmos de clase de aprendizaje automático que intenta modelar abstracciones de alto nivel en datos usando arquitecturas compuestas de transformaciones no lineales múltiples. Esto significa que, a partir de una gran cantidad de información almacenada, se podrá definir un

³ <https://www.raona.com/los-10-algoritmos-esenciales-machine-learning/>

patrón que, dada una entrada, pueda definir su salida calculada en base al conjunto de estados más probable, en los que un estado no sólo dependerá de su estado inmediatamente anterior si no que será resultado de sus n estados precedentes.

Mapillary⁴ es un servicio *online*, desarrollado por una compañía sueca, que permite compartir fotos geoetiquetadas. El objetivo de sus creadores es la representación de cualquier localización existente a través de fotografías. Para llevar a cabo esta tarea se necesita la colaboración masiva de los usuarios (*crowd-sourced*), que a su vez pueden beneficiarse del trabajo realizado por otros, y una aproximación sistemática que cubra las áreas de interés. Servicios como los de Google, que disponen de vehículos especialmente equipados con soportes para cámaras, sólo son capaces de cubrir las localizaciones accesibles, Mapillary pretende, además, cubrir las inaccesibles. Según la filosofía del servicio, el conocimiento local es casi imbatible en este punto, pues la gente sabe lo que realmente importa capturar en una fotografía. Están interesados en la cobertura de cualquier lugar al aire libre y en poder contribuir a un sistema que logre representar el mundo con un alto nivel de detalle. La mayor parte de la "inteligencia" de procesamiento de imágenes se hace en el lado del servidor usando tecnologías de *Big data* y visión por ordenador, haciendo la recopilación de datos muy simple para el usuario. Como resultado de todo esto, el servicio mejorará con cada nueva fotografía a partir de todas las nuevas fotos que estén relacionadas con cualquier foto existente en las cercanías. En la figura 2 se puede ver un diagrama en el que se describe el funcionamiento del servicio y, como con cada imagen proporcionada por los usuarios, se consigue mejorar el nivel de resolución de la aplicación.

⁴ <https://es.wikipedia.org/wiki/Mapillary>

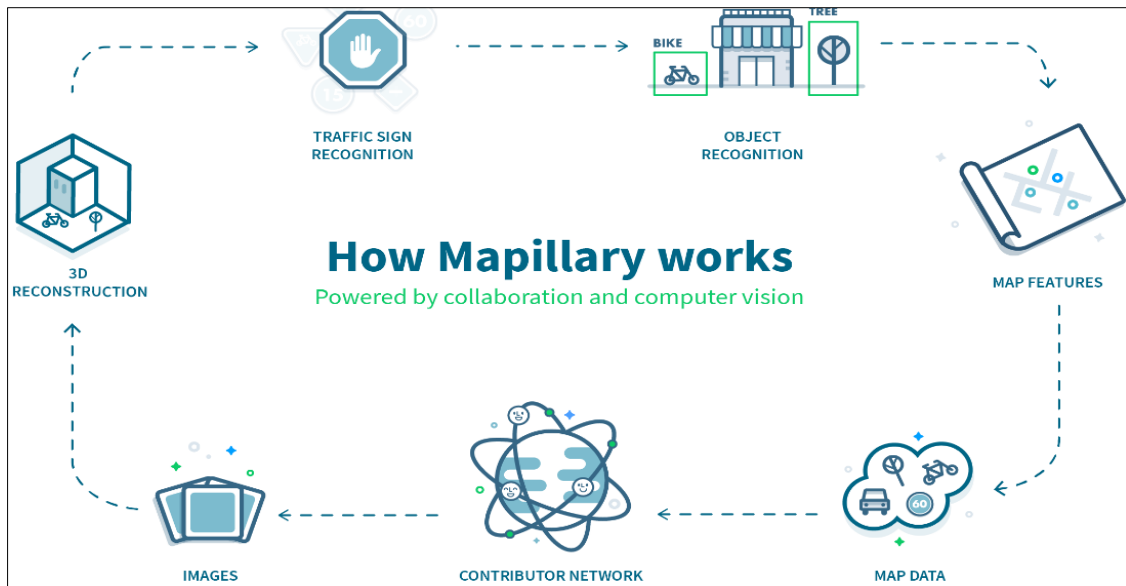


Figura 2: Esquema del funcionamiento de Mapillary

<https://www.mapillary.com/technology>

La idea es que los usuarios de los datos sean autosuficientes para incrementar la cobertura en áreas que sean de su interés. Los desarrolladores de Mapillary creen que existe un lugar en el mercado para un proveedor de imágenes neutrales e independientes. Los usuarios pueden contribuir instalando la aplicación de Mapillary en WindowsPhone, Android o iPhone, existiendo reportes exitosos incluso desde Kindle App Store, Jolla y para dispositivos que puedan ejecutar aplicaciones de Android.

OpenStreetCam⁵, software desarrollado por Telenav, es una plataforma que permite recopilar y proporcionar imágenes libres y de contenido abierto a nivel de calle. Dispone de un sitio web, aplicación de código abierto para Android e iOS, herramientas de subida y un editor de mapas propio.

OpenStreetCam esta formado por un sitio web, aplicaciones móviles de código abierto, un complemento para JOSM, herramientas para subida de imágenes y un servidor de interfaz final (*back end*). Las aplicaciones están optimizadas para ser utilizadas en cámaras localizadas en vehículos, aunque también pueden usarse caminando. Todo el código fuente de los servicios, tanto de la web como de las aplicaciones es accesible y de conocimiento público.

⁵ <https://wiki.openstreetmap.org/wiki/OpenStreetCam>

3. Estado del arte

Existen diversos algoritmos para la identificación de elementos en imágenes (la mayor parte de ellos basados en *Deep Learning*). Las imágenes son conjuntos de píxeles continuos y cada uno contiene información del color que tiene que “iluminar” (por ejemplo RGB). Algunos ejemplos comunes son los de OCR Reconocimiento de Caracteres ópticos, es decir, encontrar letras, agruparlas, encontrar espacios y poder descifrar textos. Otro ejemplo recurrente es la detección de personas en imágenes, presencia humana en cámaras de seguridad o el uso de redes neuronales para detectar rostros mediante reconocimiento facial⁶.

3.1 Software basado en algoritmos machine learning para el reconocimiento de imágenes.

Actualmente existen multitud de ejemplos de software basado en algoritmos *Machine Learning*, una muestra de la importancia que tiene este tipo de aplicaciones actualmente, puede verse en los sistemas desarrollados por algunas de las principales empresas tecnológicas actuales como *Google*, *Microsoft* o *Amazon*.

Google Cloud Vision API⁷ es una librería creada por *Google* que permite a los desarrolladores abstraerse de toda la complejidad matemática implicada en los algoritmos *Deep Learning* que componen la aplicación. La API es capaz de clasificar imágenes en categorías muy concretas, detectar caras y las emociones que reflejan, e incluso, leer textos en diferentes idiomas directamente desde una fotografía. Entre otras cosas permite, a partir de una imagen, hacer cosas como estas:

- **Identificar la entidad principal:** detecta qué tipo de objeto predomina en la imagen.

⁶ <http://www.aprendemachinelearning.com/aplicaciones-del-machine-learning/>

⁷ <https://cloud.google.com/vision/>

- **Identificación facial:** detecta dónde se encuentran las caras de la gente e identifica las diferentes partes de éstas (boca, ojos, nariz...).
- **Localización de puntos de interés:** puede reconocer infinidad de monumentos, paisajes naturales, etc... y además proporcionar la geo-localización del mismo.
- **Reconocimiento de escritura:** reconoce texto en diversos idiomas.
- **Reconocimiento de logos:** reconoce los logos de diferentes empresas y su ubicación en la imagen, de modo que permite, por ejemplo, clasificar productos de manera automática.

Azure Computer Vision⁸ es un servicio en la nube creado por *Microsoft* que proporciona a los desarrolladores acceso a algoritmos *Machine Learning* de procesamiento de imágenes en los formatos más populares como *JPEG* o *PNG*. La aplicación puede analizar las imágenes desde diferentes enfoques en función de los tipos de elementos que se deseen detectar, en concreto puede realizar las siguientes acciones:

- **Identificar y etiquetar las características de la imagen:** basándose en más de 2.000 objetos, seres vivos, paisajes y acciones reconocibles. El etiquetado no se limita al tema principal, como una persona en primer plano, sino que también incluye el entorno, muebles, herramientas, plantas, animales, accesorios, artilugios, etc. Cuando las etiquetas son ambiguas o no son de conocimiento común, la respuesta proporciona sugerencias para aclarar el significado de la etiqueta en el contexto de una configuración conocida.
- **Clasificar imágenes por categorías:** utilizando una categoría de taxonomía con jerarquías hereditarias padre-hijo.
- **Generar una descripción de la imagen:** los algoritmos generan varias descripciones basadas en los objetos identificados en la imagen, estas descripciones son evaluadas y se genera una puntuación de confianza.
- **Detectar caras:** proporciona información sobre cada cara detectada identificando el género y la edad de cada rostro detectado.

⁸ <https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/home>

- **Identificar contenido específico del dominio en una imagen:** como, por ejemplo, personalidades, puntos de referencia ...
- **Miniaturizar una imagen resaltando el elemento principal:** el software genera una miniatura de alta calidad y luego analiza los objetos dentro de la imagen para determinar la región de interés (ROI). Posteriormente, recorta la imagen para que se ajuste a los requisitos de la región de interés. La miniatura generada se puede presentar utilizando una relación de aspecto diferente a la imagen original.
- **Reconocimiento de escritura:** reconoce texto en diversos idiomas.

Aws rekognition⁹ es la solución desarrollada por Amazon para el reconocimiento de imágenes. La API identifica objetos, personas, texto, escenas y actividades, además de detectar contenido inapropiado. Además, ofrece análisis y reconocimiento facial de alta precisión en imágenes y videos. Puede detectar, analizar y comparar rostros para una amplia variedad de casos de uso de verificación de usuarios, contabilización de personas y seguridad pública. El software está basado en la misma tecnología *deep learning* de alta escalabilidad que desarrollaron los ingenieros de visión artificial de Amazon para analizar miles de millones de imágenes y videos diariamente. No es necesario contar con experiencia en aprendizaje automático para utilizarla, es una API simple y fácil de usar que puede analizar rápidamente cualquier archivo de video o imagen almacenado en Amazon S3. Amazon Rekognition siempre está aprendiendo de los datos nuevos, por lo que continuamente se añaden nuevas características de reconocimiento facial y etiquetas al servicio.

⁹ <https://aws.amazon.com/es/rekognition/>

3.1.1 Características del software basado en algoritmos machine learning para el reconocimiento de imágenes.

En la siguiente tabla puede verse un resumen de las funcionalidades de cada aplicación.

FUNCIONALIDAD	AWS REKOGNITION	GOOGLE CLOUD VISION API	AZURE COMPUTER VISION API/ FACE API
ETIQUETADO DE IMÁGENES (OBJETOS Y ESCENAS)	SI	SI	SI
DETECCIÓN DE CARAS	SI	SI	SI
RECONOCIMIENTO DE RASGOS FACIALES	SI	SI	SI
DETECCIÓN DE SENTIMIENTO	SI	SI	SI
OCR	NO	SI	SI
DETECCIÓN DE LOGOS Y MARCAS	NO	SI	NO
DETECCIÓN DE CONTENIDO INAPROPIADO	SI	SI	SI
COMPARACIÓN DE CARAS	SI	NO	SI
BÚSQUEDA DE CARAS	SI	NO	NO
RECONOCIMIENTO DE CELEBRIDADES	SI	NO	SI
ANÁLISIS DE VIDEO	NO	SI	SI

Tabla 1: Resumen funcionalidades de las diferentes APIs de reconocimiento estudiadas
<https://ingenio.edu.pe/aws-vs-microsoft-azure-vs-google-cloud-platform-cual-es-la-mejor-opcion/>

3.2 Soluciones OpenSource

En este apartado se describirán los principales proyectos existentes en la actualidad que ofrecen software de reconocimiento de imágenes de código abierto, accesible a cualquier usuario que así lo desee, basado en algoritmos Machine Learning.

Luminoth¹⁰

Es un kit de herramientas de visión artificial desarrollada en Python y que utiliza funcionalidades proporcionadas por las librerías Tensorflow y Sonnet.

Tensorflow¹¹ es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas, fue desarrollado por Google para satisfacer sus necesidades de sistemas capaces de construir y entrenar redes neuronales para la detección de patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos. Actualmente es utilizado tanto en la investigación como en los productos de Google, fue originalmente desarrollado por el equipo de Google Brain para uso interno en Google antes de ser publicado bajo la licencia de código abierto en Noviembre de 2015

¹⁰ <https://es.wikipedia.org/wiki/TensorFlow>

¹¹ <https://aws.amazon.com/es/rekognition/>

Luminoth es un ejemplo de las mejoras proporcionadas por las técnicas de *deep learning* y, en particular, por algunos grupos de arquitecturas de redes neuronales como son las *redes neuronales convolucionales* (CNN o ConvNets). Estas redes tienen una serie de propiedades que las hacen muy adecuadas para las tareas de procesamiento de imágenes. Las CNN tienen la capacidad de deslizar espacialmente diferentes filtros a través de una imagen, y usar pilas de estos filtros para reconocer patrones de niveles crecientes de abstracción, de esta manera pueden comprender conceptos complejos que de otra forma serían difíciles de expresar.

Las redes neuronales profundas permiten aplicar gran variedad de diseños de arquitecturas diferentes, ya que permite disponer de diferentes tipos de capas, probar diferentes configuraciones y experimentar con diferentes parámetros. De hecho, el estado del arte de varias de estas tecnologías está cambiando rápidamente casi a diario, debido a las numerosas variantes que ofrece, permitiendo crear multitud de prototipos. La parte negativa de algunos de estos métodos es la enorme cantidad de datos de entrenamiento necesarios para obtener resultados significativos, que a veces pueden ser muy costosos de obtener y, además, suponer un coste computacional elevado. Las últimas versiones del software proporcionan reconocimiento de imágenes en tiempo real soportando alrededor de 60 imágenes por segundo, lo que lo hace adecuado para la transmisión de secuencias de video o videos en general.

Open CV¹²

Es una biblioteca de código abierto de visión artificial originalmente desarrollada por Intel. Desde su aparición se ha utilizado para implementar gran cantidad de aplicaciones, desde sistemas de seguridad con detección de movimiento, hasta aplicaciones de control de procesos que requieren reconocimiento de objetos. Un ejemplo interesante de su uso puede encontrarse en el sistema de visión del vehículo no tripulado Stanley desarrollado por la Universidad de Stanford en el año 2005. Ha sido publicada

¹² <https://en.wikipedia.org/wiki/OpenCV>

bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación con las condiciones en ella expresadas.

Open CV es multiplataforma, existiendo versiones para GNU/Linux, Mac OS X y Windows. Contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos (reconocimiento facial), calibración de cámaras, visión estérea y visión robótica.

El proyecto pretende proporcionar un entorno de desarrollo fácil de utilizar y altamente eficiente. Ha sido desarrollado en C y C++ optimizados, aprovechando además las capacidades que proveen los procesadores multinúcleo. Además, puede utilizar el sistema de primitivas de rendimiento integradas de Intel, un conjunto de rutinas de bajo nivel específicas para procesadores Intel.

Las áreas de aplicación de OpenCV incluyen: kits de herramientas de características 2D y 3D, estimación de egomoción¹³, sistema de reconocimiento facial, reconocimiento de gestos, interacción humano-computadora (HCI), robótica móvil, comprensión del movimiento, segmentación y reconocimiento, stereopsis o percepción de profundidad de 2 cámaras, estructura desde el movimiento (SFM), rastreo de movimiento y realidad aumentada.

OpenStreetCam

Es un proyecto colaborativo que pretende cualquier localización geográfica existente mediante las imágenes georeferenciadas subidas proporcionadas por los usuarios. Adicionalmente, mediante la I.A (Inteligencia Artificial) desarrollada por Telenav permite detectar diferentes elementos contenidos en las fotografías, como, señales de tráfico, personas o vehículos, estos últimos, a través de la detección de sus placas de matrícula.

¹³ Estimación del movimiento de una cámara en relación a un entorno fijo, por ejemplo, la estimación sería estimar la posición de movimiento de un automóvil en relación con las líneas en la carretera.

https://en.wikipedia.org/wiki/Visual_odometry

4. Evaluación del software de Telenav

Como se ha mencionado en el apartado 1.2, todas las aplicaciones de OpenStreetCam son de código abierto, por lo que son totalmente accesibles desde la herramienta de control de versiones de Github disponible desde la siguiente url: <https://github.com/openstreetcam>

Como se puede ver en la figura 3, hay disponibles diferentes aplicaciones, scripts de Python que permiten subir fotos georeferenciadas al sistema, el editor de mapas JOSM, las aplicaciones Android e IOS, la web principal y una herramienta de subida de fotos con interfaz gráfica.

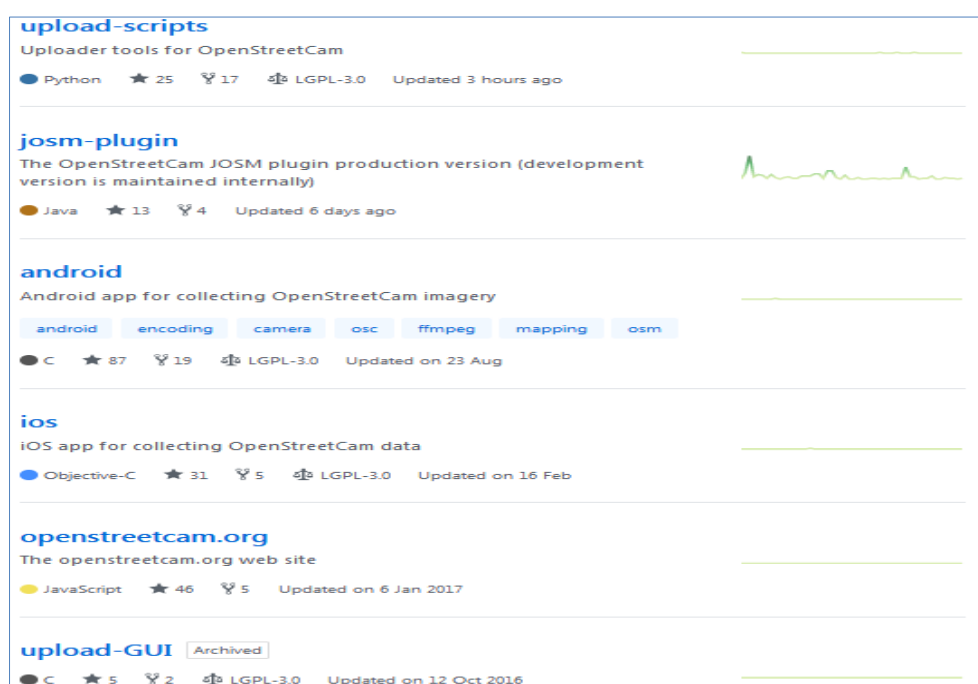


Figura 3: Catálogo de aplicaciones de OpenStreetCam
<https://github.com/openstreetcam>

En posteriores apartados se verá con mayor profundidad algunas de estas herramientas.

4.1 Especificaciones de OpenStreetCam

El software principal de OpenStreetCam es su aplicación web que es la que contiene el núcleo de la lógica de negocio, integrando entre otros elementos la inteligencia artificial de Telenav utilizada para la identificación de elementos viales, siendo el resto de aplicaciones, herramientas complementarias. Las aplicaciones para dispositivos móviles, Android e iOS, permiten subir fotos

georeferenciadas a la cuenta de usuario, para poder aplicarles posteriormente el algoritmo de reconocimiento de imágenes. Además de las apps para móviles es posible subir fotografías mediante un script de Python y una interfaz gráfica de usuario (GUI). La gran ventaja de las apps frente a las aplicaciones tradicionales es que permiten añadir las coordenadas en los metadatos de las imágenes de manera autónoma, por lo que junto a un montaje del dispositivo móvil que ejecute la aplicación en un vehículo, permitirá capturar todas las imágenes de un itinerario. Desde la propia aplicación el usuario puede subir las imágenes que serán asociadas a un punto concreto del mapa mediante las coordenadas contenidas en la imagen. Aunque en este trabajo no nos centraremos demasiado en ello, hay que mencionar el editor de mapas JOSM¹⁴, que permite a los usuarios componer los diferentes elementos que forman la cartografía de un territorio, permitiendo añadir vías, edificios, puntos de interés...

Puede accederse a la aplicación web a través de la siguiente url <http://openstreetcam.org>. En la figura 4 puede verse la aplicación, una vez en ella es recomendable crear una cuenta de *OpenStreetMap*¹⁵ que permitirá subir imágenes mediante alguna de las herramientas mencionadas anteriormente.

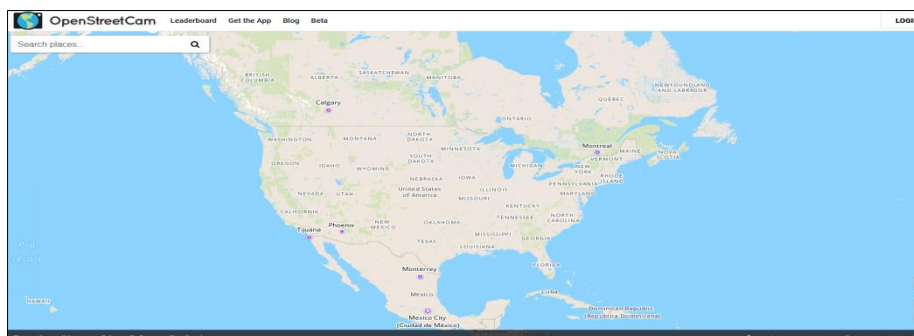


Figura 4: Aplicación web de OpenStreetCam

<http://openstreetcam.org>

Para acceder a la inteligencia artificial desarrollada por Telenav, que también sigue la filosofía *OpenSource*, es necesario entrar al siguiente repositorio Git <https://github.com/Telenav/Telenav.AI>.

¹⁴ <https://josm.openstreetmap.de>

¹⁵ <https://www.openstreetmap.org>

4.2 Entornos de pruebas

Hay tres formas distintas de poder probar el funcionamiento del algoritmo *Machine Learning* de reconocimiento de señales de tráfico y otros elementos viales en imágenes. Como se ha comentado en el apartado anterior se puede encontrar el software ya integrado y accesible a través de la aplicación web de OpenStreetCam, además, es posible instalar dos entornos diferentes, un script Python y un script de Unix, con el módulo que contiene el código con el algoritmo de detección de imágenes mediante el repositorio de Telenav en Github.

4.2.1 Entorno para ejecución de script Unix

Si se opta por esta opción hay que realizar una serie de acciones para poder establecer el entorno de pruebas, lo primero que debe hacerse es descargar el conjunto de imágenes de prueba disponible en el repositorio Git, https://s3.eu-central-1.amazonaws.com/telenav.ai/telenav_ai_dataset.zip.

En el fichero comprimido se encontrará una estructura de carpetas como la que se puede ver en la figura 5.






	protobuf	259,8 kB	Folder
	test_data	3,2 GB	Folder
	train_data	61,6 GB	Folder
	LICENSE	181 bytes	unknown
	README.md	508 bytes	Markdown ..

Figura 5: Fichero con el set de imágenes de entrenamiento de Telenav

En la carpeta `train_data` se pueden encontrar todas las imágenes de ejemplo que alimentan los procesos de aprendizaje y ejecución del algoritmo. La carpeta además de imágenes contiene un archivo de texto con las etiquetas de todas las imágenes de la carpeta, indicando los diferentes elementos viales de cada una. La carpeta `test_data` contiene imágenes no etiquetadas para poder probar el funcionamiento del algoritmo y la carpeta `protobuf` contiene referencias a librerías y diferentes elementos para establecer propiedades de configuración.

Para poder probar el software, además, es necesario disponer de un sistema Unix, Ubuntu 16.04, puede ser una opción recomendable. Una vez instalado es necesario instalar la herramienta de control de versiones Git introduciendo las instrucciones siguientes en el terminal.

```
sudo apt-get update  
  
sudo apt-get install git
```

El siguiente paso es clonar en nuestro entorno local el repositorio de Telenav.

```
git clone https://github.com/Telenav/Telenav.AI
```

El software se apoya en el uso de *Docker*, que son contenedores ligeros y portables para las aplicaciones software que puedan ejecutarse en cualquier máquina con Docker instalado, independientemente del sistema operativo que la máquina tenga por debajo, facilitando así también los despliegues. Por tanto lo siguiente que se deberá hacer es instalar Docker, para ello es necesario añadir la clave GPG del repositorio Docker.

```
sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --  
recv-keys 58118E89F3A912897C070ADBF76221572C52609D
```

Además, es necesario agregar las fuentes APT.

```
sudo apt-add-repository 'deb https://apt.dockerproject.org/repo  
ubuntu-xenial main'
```

Tras esto se deberá actualizar, de nuevo, la base de datos de paquetes.

```
sudo apt-get update
```

Se establecerá el repositorio Docker para la instalación.

```
apt-cache policy docker-engine
```

Por último, se instalará Docker.

```
sudo apt-get install -y docker-engine
```

Una vez instalado Docker se deberá ejecutar el siguiente script que desplegará el software de Telenav que se encuentra en los contenedores.

```
chmod +x ./docker_build_image_retinanet_mq.sh  
  
sh ./docker_build_image_retinanet_mq.sh
```

Para poder ejecutar el contenedor es necesario, además, instalar el plugin de nvidia, primero se deberá comprobar si hay una versión anterior instalada y si es así borrarla.

```
docker volume ls -q -f driver=nvidia-docker | xargs -r -I{} -n1 docker
ps -q -a -f volume={} | xargs -r docker rm -f
sudo apt-get purge -y nvidia-docker
```

Tras ello, será necesario añadir el repositorio.

```
curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | \
  sudo apt-key add -
curl -s -L https://nvidia.github.io/nvidia-
docker/ubuntu16.04/amd64/nvidia-docker.list | \
  sudo tee /etc/apt/sources.list.d/nvidia-docker.list
sudo apt-get update
```

Ahora se puede ejecutar la instalación del plugin.

```
sudo apt-get install -y nvidia-docker2
sudo pkill -SIGHUP dockerd
```

Finalmente, una vez instalados todos los elementos del entorno se deberá ejecutar la siguiente instrucción que permitirá ejecutar el contenedor en un nuevo terminal.

```
nvidia-docker run -v {ARTIFACTS_PATH}:/data/ --name retinanet -it
telenav/retinanet_mq /bin/bash
```

{ARTIFACTS_PATH} contendrá la ruta con del fichero, una vez extraído, con el conjunto de imágenes de entrenamiento.

4.2.2 Entorno para ejecución de script Python

Otra forma de ejecutar el algoritmo de Telenav es hacerlo directamente desde Python, al igual que para el entorno anterior, es recomendable hacerlo desde una distribución Unix, ya que es el sistema operativo utilizado para el desarrollo. Por ejemplo, se puede utilizar Ubuntu 16.04 y Python 2.7. Para prepara el entorno de pruebas es necesario realizar las acciones que se indican a continuación:

En primer lugar, al igual que en el caso anterior, es necesario descargar un archivo comprimido con las imágenes de prueba, sus metadatos, y los scripts de Python, desde la siguiente dirección.

```
wget : https://s3.eu-central-
1.amazonaws.com/telenav.ai/telenav_ai_dataset_sample.zip
```

Una vez descargado, se descomprime el fichero.

```
unzip telenav_ai_dataset_sample.zip
```

Hecho lo anterior, se deberá acceder al prompt de python y cargar todas las definiciones y librerías incluidas en la carpeta protobuf del archivo descargado anteriormente en el path de Python.

Python 2.7

```
sys.path.append("./telenav_ai_dataset_sample/protobuf")
import proto_api as proto_api
```

Ahora, es necesario establecer las carpetas que contendrán tanto las imágenes de ejemplo como las que permitirán probar el algoritmo.

```
images_folder = "./telenav_ai_dataset_sample/sample_data/"
rois_path = os.path.join(images_folder, "rois.bin")
images_paths = glob(images_folder + "*[jpg,jpeg]")
```

Se lanzan las funciones incluidas en las definiciones cargadas que permiten al algoritmo leer las imágenes.

```
imageset_proto = proto_api.read_metadata(rois_path)
data_df = read_df(imageset_proto)
data_df.head()
```

Es posible detectar todos los puntos de interés o filtrar por algunos de ellos.

```
non_empty_df = data_df[data_df["nr_rois"] != 0]
plot_data_df(non_empty_df, nr_images_to_plot = 10)
```

Por ejemplo, es posible filtrar para que el algoritmo detecte sólo las señales de prohibición de giro a la derecha.

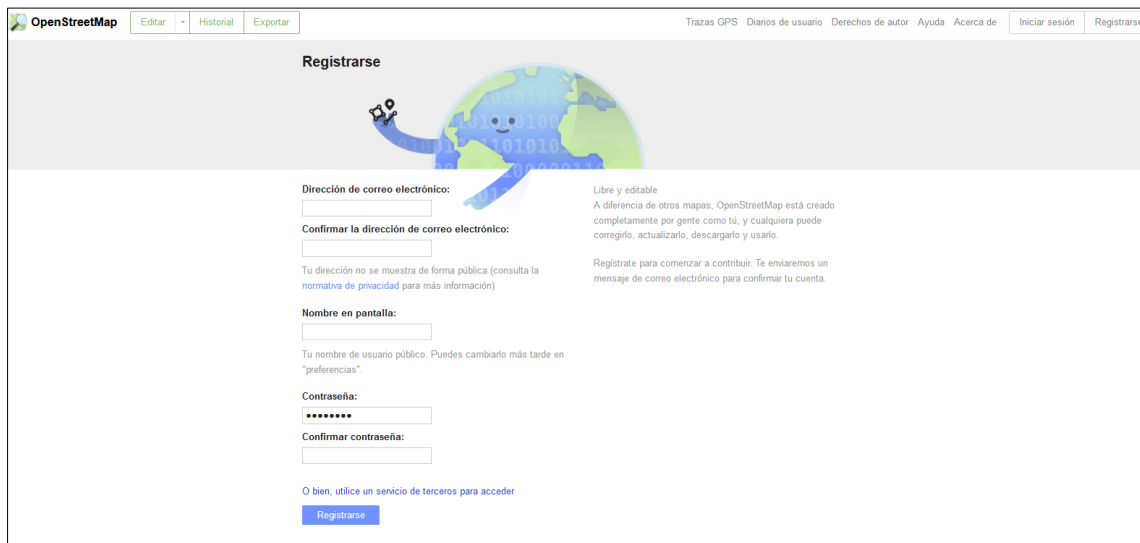
```
specific_type_rois_df = rois_df[rois_df["roi_type"] ==
"TURN_RESTRICTION_US_RIGHT"]
plot_roi_df(specific_type_rois_df, nr_rois_to_plot = 10)
```

4.2.3 Entorno web

Si se va a ejecutar el algoritmo integrado desde la aplicación web, aunque no es estrictamente necesario para trabajar con las imágenes de Telenav o con las subidas por otros usuarios, es recomendable crear una cuenta en OpenStreetMap que posteriormente nos permitirá subir imágenes para ser procesadas por el algoritmo. En la siguiente figura puede verse la página de

registro accesible desde la siguiente dirección:

<https://www.openstreetmap.org/user/new>



Registrarse

Dirección de correo electrónico:

Confirmar la dirección de correo electrónico:

Tu dirección no se muestra de forma pública (consulta la [normativa de privacidad](#) para más información)

Nombre en pantalla:

Tu nombre de usuario público. Puedes cambiarlo más tarde en "preferencias".

Contraseña:

Confirmar contraseña:

O bien, utilice un servicio de terceros para acceder

[Registrarse](#)

Libre y editable
A diferencia de otros mapas, OpenStreetMap está creado completamente por gente como tú, y cualquiera puede corregirlo, actualizarlo, descargarlo y usarlo.

Regístrate para comenzar a contribuir. Te enviaremos un mensaje de correo electrónico para confirmar tu cuenta.

Figura 6: Formulario de registro de OpenStreetMap

<http://www.openstreetmap.org/user/new>

Una vez creada la cuenta se puede volver a la página principal de OpenStreetCam y acceder utilizando la cuenta recientemente creada como puede verse en la figura 7.

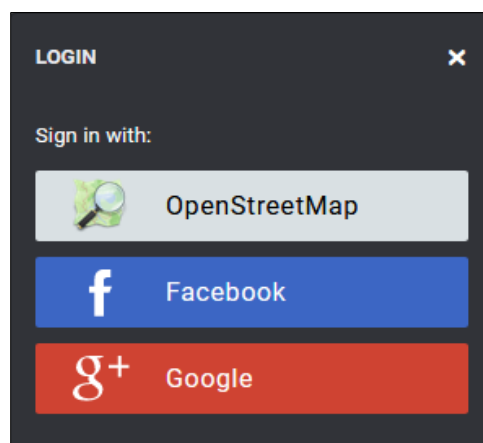


Figura 7: Opciones de identificación de OSC

<http://openstreetcam.org>

Para poder probar el funcionamiento del algoritmo sobre imágenes reales basta con explorar el mapa y dirigirse a alguna de las zonas con un mayor número de fotografías subidas por los usuarios, es recomendable hacerlo en EEUU, ya que hay un número grande de aportaciones, algunas ciudades como Phoenix, están completamente fotografiadas.

4.3 Pruebas y análisis de resultados

Para comprobar el grado de acierto del algoritmo de detección se ha elegido la aplicación web como entorno de trabajo ya que permite comparar entre miles de *tracks* diferentes subidos por diferentes usuarios y filtrar por diferentes criterios las imágenes contenidas en cada uno de ellos, permitiendo elegir desde los elementos detectados automáticamente, los que han sido etiquetados por los usuarios, los que el algoritmo ha fallado... Además, la aplicación web es la que mejor ilustra la utilidad y funcionamiento del algoritmo en un entorno real.

Para probar el funcionamiento del algoritmo se han elegido diferentes ciudades de EEUU, ya que permiten disponer del mayor número de imágenes y con ello dotar a las pruebas de la máxima fiabilidad posible. En concreto se han elegido diferentes *tracks* de las ciudades de Phoenix y Mesa, tal como aparece en la siguiente imagen.

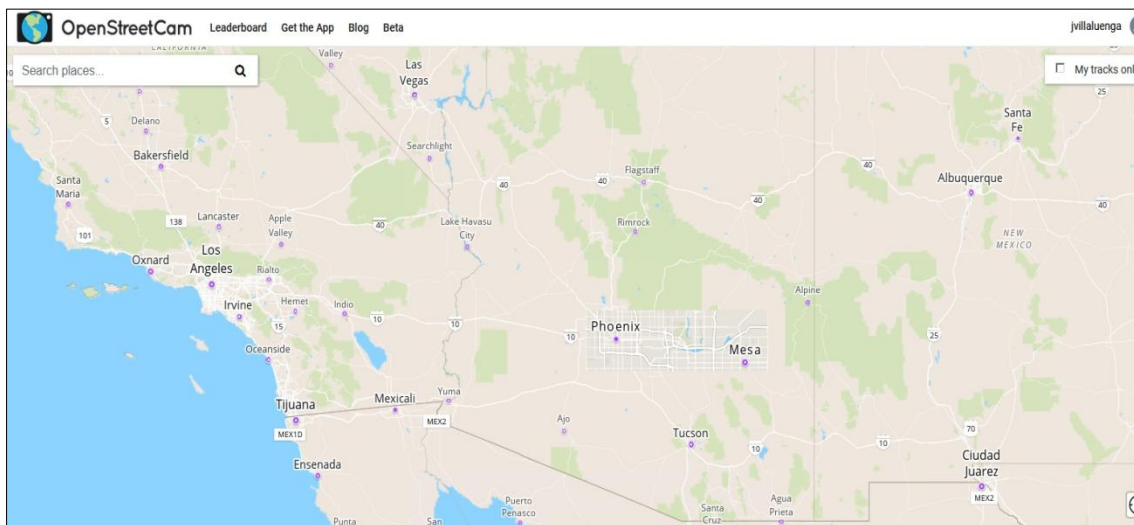


Figura 8: Localización geográfica para ejecución de pruebas

<http://openstreetcam.org>

Una vez situada la localización es necesario ampliar el mapa hasta que en la parte inferior de la pantalla aparezcan todos los *tracks* cargados por los diferentes usuarios que corresponden a la referencia geográfica en la que nos encontramos, simultáneamente comenzaran a dibujarse en el mapa todas las vías identificadas por la aplicación web. En la figura 9 se puede ver una ilustración de esto.

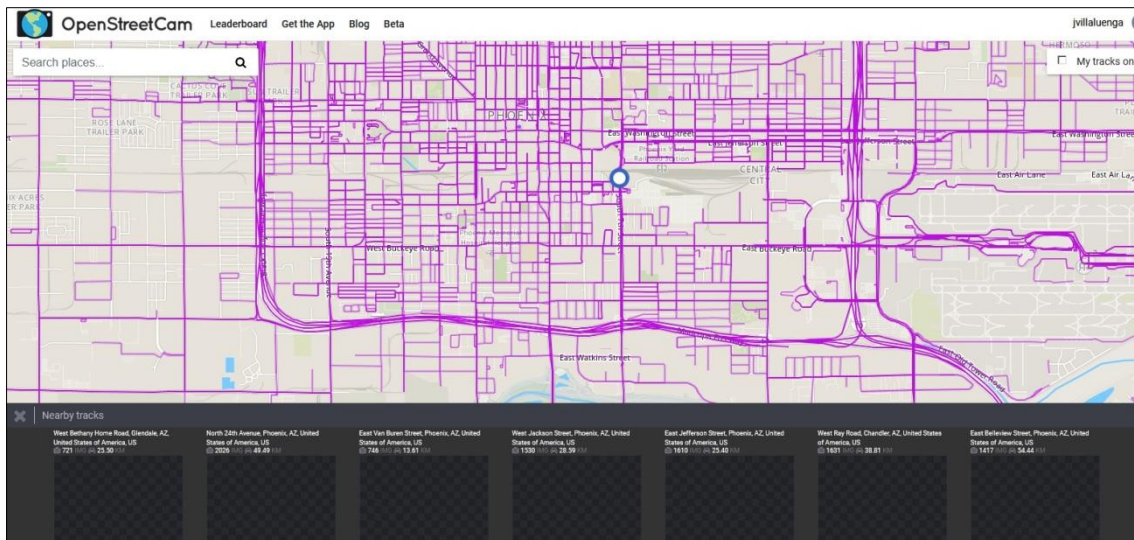


Figura 9: Vías identificadas y tracks subidos por los usuarios

<http://openstreetcam.org>

Si se carga cualquiera de los *tracks* aparecerá una pantalla como la que se puede ver en la siguiente imagen, en la parte izquierda aparecen los principales datos de la ruta cargada, entre ellos el número de Kilometros cubiertos en la ruta, las señales viales etiquetadas, entre las cuales se pueden identificar las que han sido detectadas por el algoritmo o las que lo han sido por un usuario. Justo debajo de la imagen se puede ver el número de fotografías que contiene la ruta y en la que se encuentra el usuario en ese punto de la ruta, tal como se muestra en la figura 10.

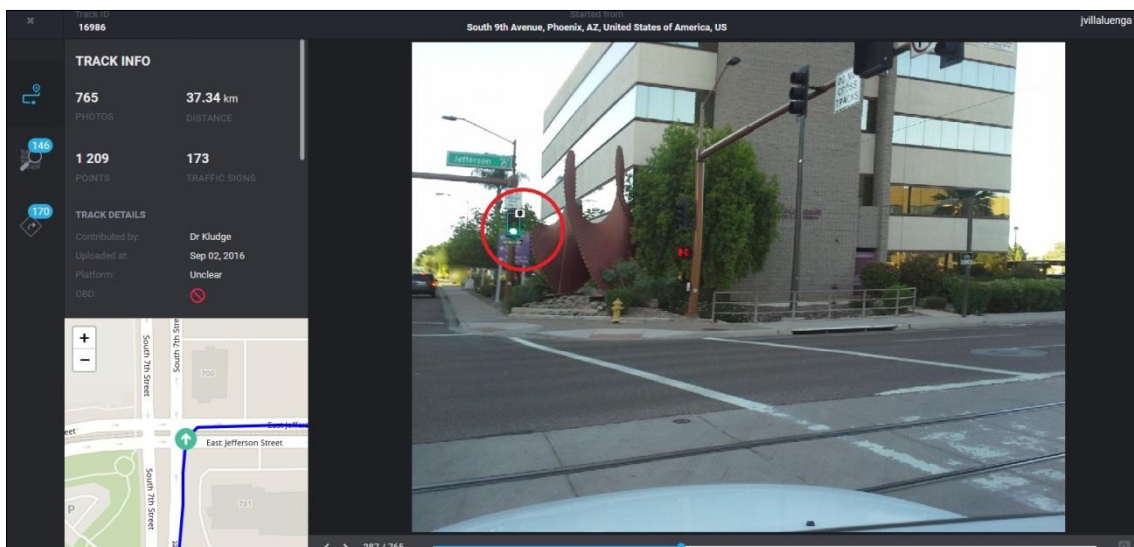


Figura 10: Pantalla principal de la aplicación

<http://openstreetcam.org>

En la parte izquierda del panel de control, como se puede ver en el recuadro rojo de la figura 11, la aplicación dispone de tres entradas diferentes, la primera de ellas muestra la información de la ruta seleccionada, la segunda entrada integra el acceso a JOSM e ID y permite editar los elementos del mapa y, por último, la tercera entrada contiene los diferentes filtros aplicables a la detección de señales.

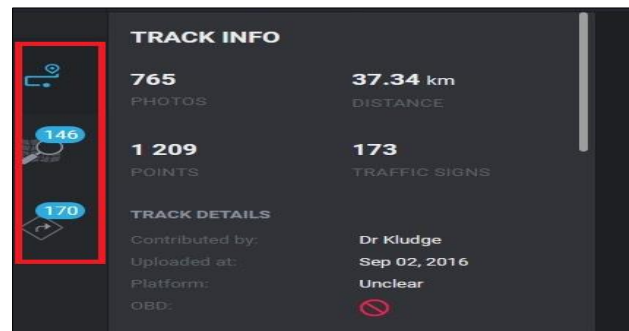


Figura 11: Panel de control de OSC

<http://openstreetcam.org>

A continuación, se pueden ver las diferentes opciones disponibles para filtrar imágenes en el apartado de detección de señales, permitiendo mostrar únicamente las imágenes tratadas de manera autónoma por el algoritmo, tal como se puede ver en la figura 12.

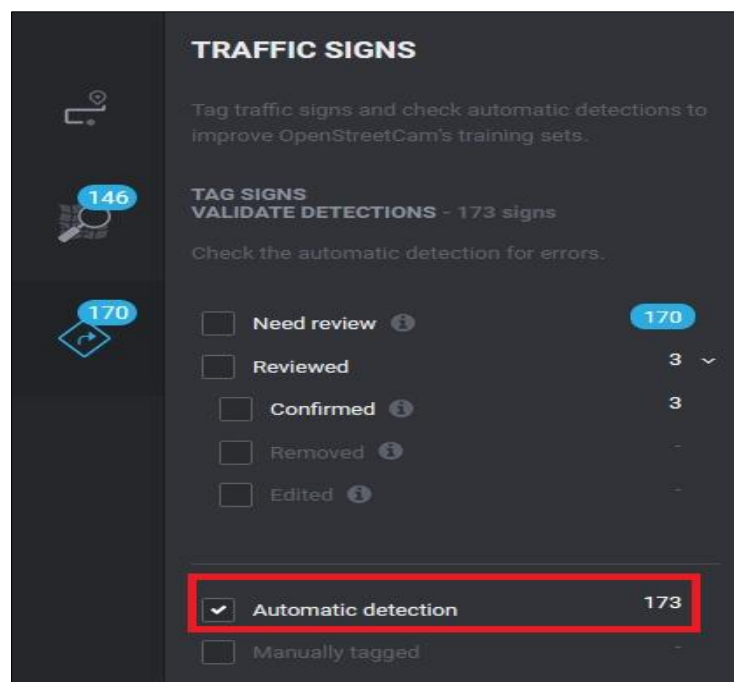


Figura 12: Filtros aplicados para la detección de señales

<http://openstreetcam.org>

Presentados los conceptos básicos de manejo de la aplicación, se va a evaluar el funcionamiento del algoritmo de detección. Accedemos a una las rutas de la ciudad de Mesa, en Arizona y filtramos las imágenes en las que se ha aplicado el algoritmo para detectar las señales de manera automática. Como se puede ver en la figura 13 aparecerán todas las señales detectadas justo debajo de la galería de imágenes de la ruta. En los iconos de las señales, además, aparecerá una equis roja, si algún usuario ha notificado un error en la detección de la señal, o un signo de acierto en color verde si la identificación ha sido evaluada como correcta.

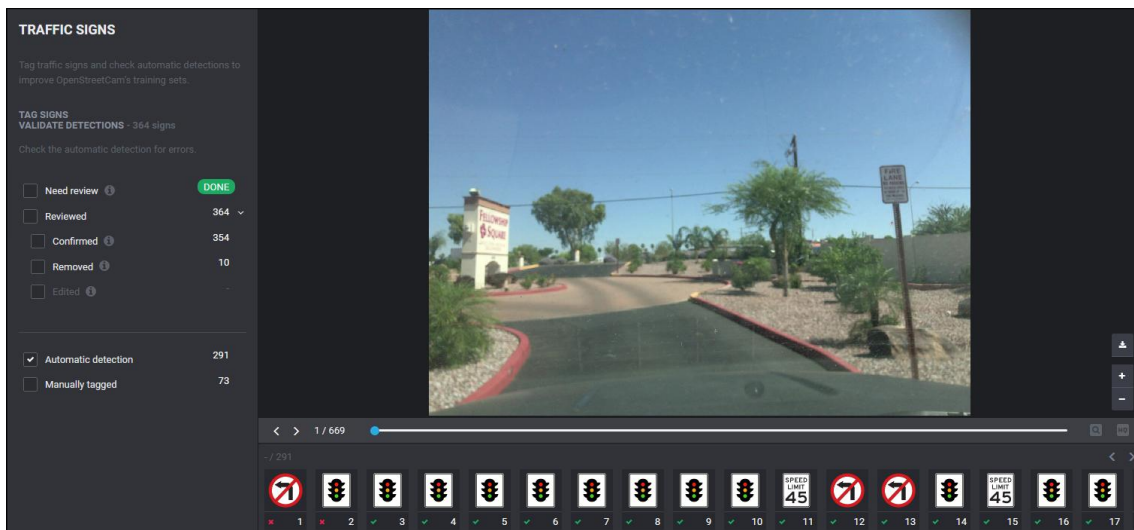


Figura 13: Ejecución del algoritmo de detección de señales

Si se sigue el orden de la ruta y nos situamos en la primera imagen en la que el algoritmo ha detectado una señal, se puede comprobar, como en este caso, se ha producido una identificación incorrecta detectando la señal como una prohibición de giro en lugar de una limitación de velocidad, como se puede ver en la siguiente figura.

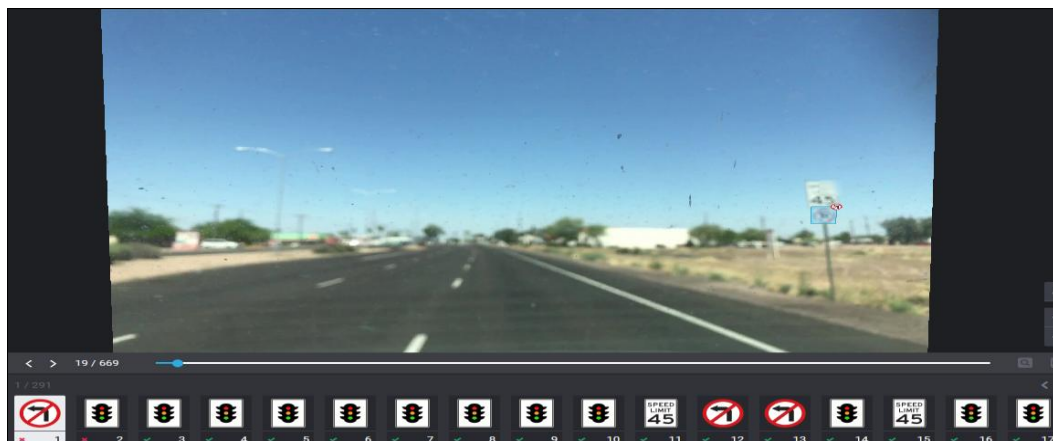


Figura 14: Identificación incorrecta de señal de giro

Avanzamos hasta la siguiente imagen en la que se ha detectado una señal de tráfico, en este caso se trata de un semáforo, también detectado de manera fallida. Como se puede apreciar en la figura 15, la imagen no es demasiado nítida y esto puede justificar que el algoritmo no realice la detección de manera correcta.

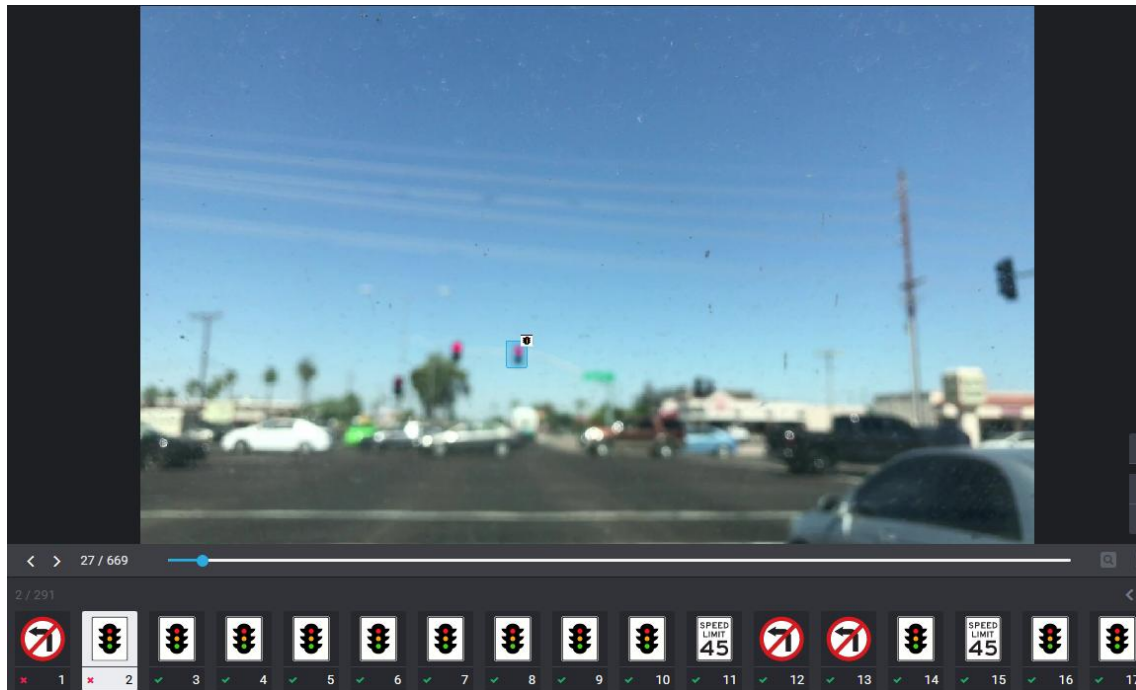


Figura 15: Identificación incorrecta de semáforo

Si se sigue avanzando hasta imágenes posteriores de la ruta, se pueden ver diferentes ejemplos que sirven para ilustrar el grado de resolución del algoritmo con diferentes tipos de señales.

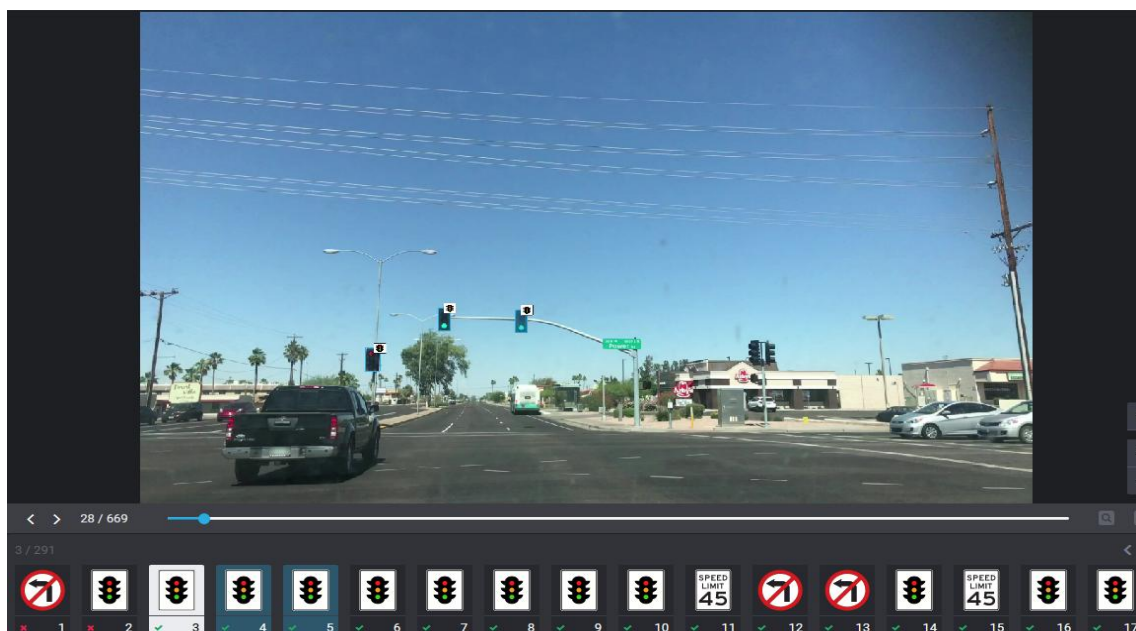


Figura 16: Identificación correcta de varios semáforos

En la imagen anterior se puede comprobar como el algoritmo es capaz de detectar los tres semáforos situados de frente, en el mismo sentido en el que se ha tomado la fotografía (el algoritmo no está diseñado para detectar los semáforos situados en la parte derecha porque no están situados de frente al objetivo de la imagen). A continuación, se puede observar un ejemplo de esto, en la siguiente imagen pueden distinguirse hasta nueve semáforos, pero el algoritmo identifica sólo los que están situados en el sentido de la marcha desde el que se ha tomado la fotografía.

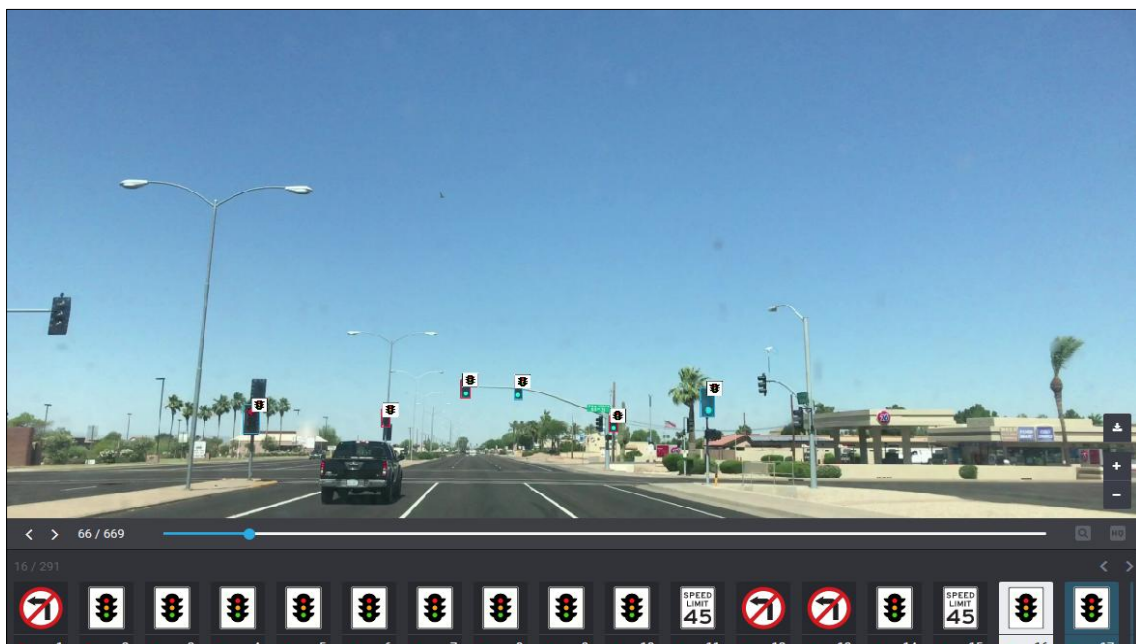


Figura 17: Identificación de semáforos

En la siguiente imagen puede verse un ejemplo de identificación con otras señales, en este caso se trata de una señal de limitación de velocidad.

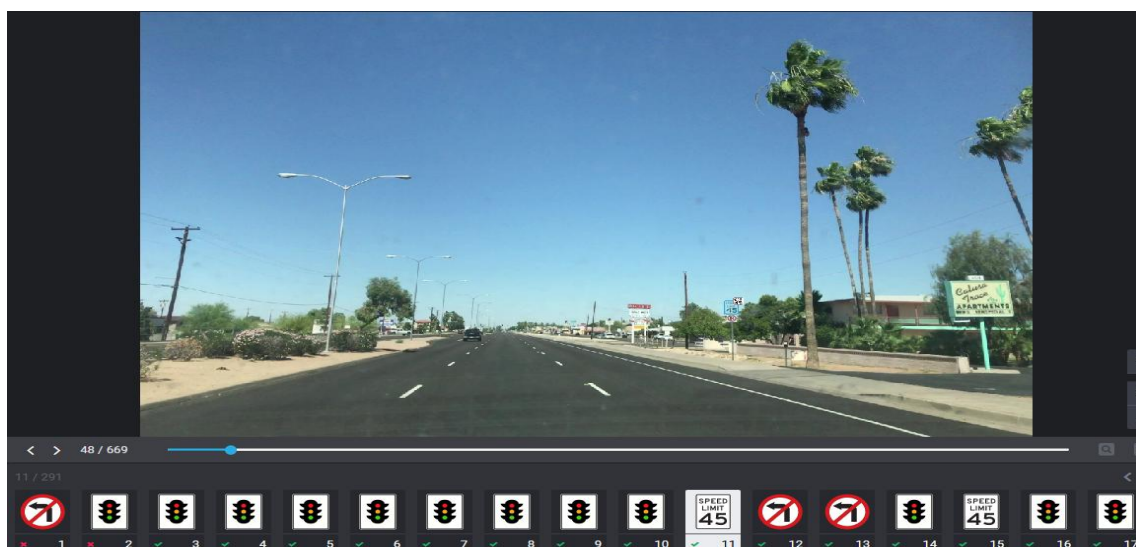


Figura 18: Identificación de señal de límite de velocidad

A continuación, en la figura 19,, se puede ver otro ejemplo de detección correcta con dos señales de prohibición de giro a la izquierda.

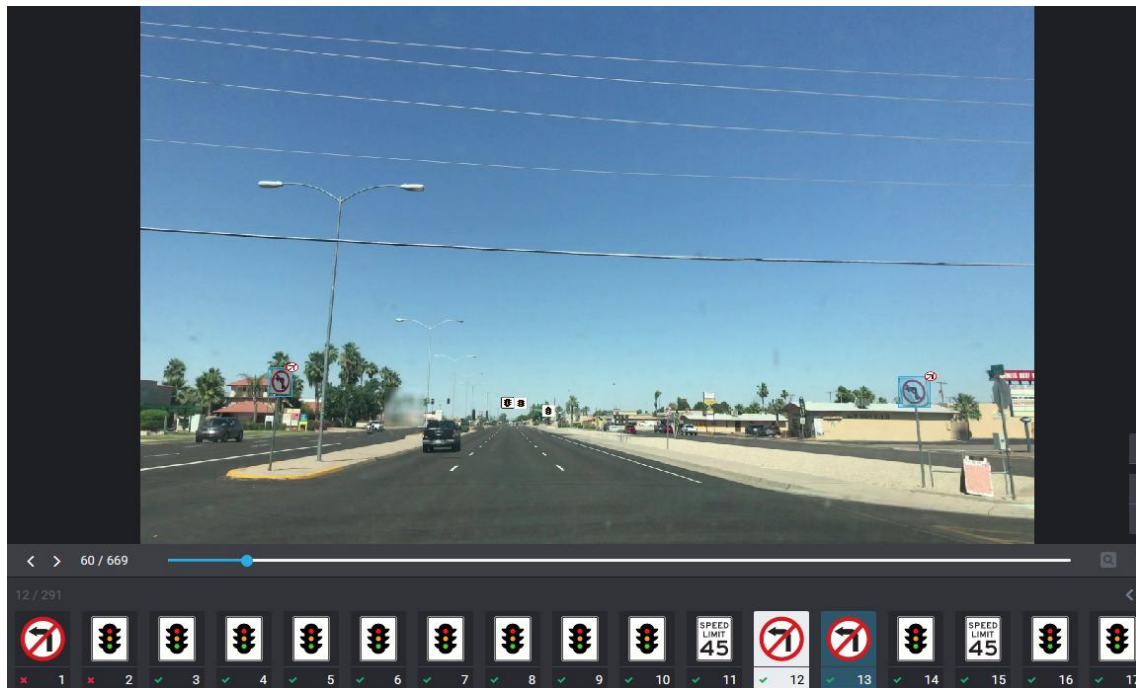


Figura 19: Detección de señal de giro prohibido

Aunque es algo que sucede en contadas ocasiones, a veces algunos elementos del entorno pueden provocar fallos en el algoritmo, en la imagen que se muestra a continuación, en la figura 20, se puede apreciar uno de estos casos. Como puede verse, en la parte derecha de la imagen el algoritmo ha identificado una de las palmeras como si fuera un semáforo, además, se observa que es un error puntual ya que aún existiendo varias palmeras en una posición similar el error sólo sucede con una de ellas.



Figura 20: Detección incorrecta debido a un elemento del entorno

Las pruebas anteriores sirven para tener una primera impresión de la forma de funcionamiento del algoritmo ilustrando con ejemplos concretos algunos de sus puntos fuertes en la identificación de imágenes y, también, ver en que circunstancias puede llegar a producirse un fallo. A continuación se recopilarán datos tomados sobre diferentes rutas que permitirán evaluar, de una manera más empírica, el grado de resolución del sistema.

Track ID	Situación	Nº de señales totales	Nº de señales detectadas	Nº de señales correctas	Nº de señales erroneas	Porcentaje de acierto (sobre detectadas)	Porcentaje de acierto (sobre el total)
393097	Mesa, EEUU	364	291	281	10	96.6%	77.2%
1300033	New York, EEUU	10	9	9	0	100%	90%
1272483	New York, EEUU	214	214	206	8	96.3%	96.3%

Tabla 2: Datos de evaluación del algoritmo

Como se aprecia en los datos recogidos en la tabla anterior la capacidad de acierto del algoritmo es especialmente alta en las señales que puede identificar y, únicamente, disminuye un poco su grado de eficiencia si se tiene en cuenta las señales que no es capaz de detectar. Aún así, teniendo en cuenta todas las señales existentes el grado de acierto está por encima del 77% en el peor de los casos.

Es interesante mencionar que el algoritmo no es sólo capaz de identificar señales de tráfico, si no que, además, puede detectar coches, a través de las placas de sus matrículas, y personas. De hecho, los datos recogidos en la última fila de la tabla anterior hacen referencia a una ruta en la que la detección automática se hizo sobre estos elementos y, como se observa, el nivel resolución es incluso mayor que respecto a las señales de tráfico.

5. Fundamentos del algoritmo de detección en OSC

El detector de objetos en imágenes de Telenav se basa en un mecanismo impulsado en dos etapas. La primera etapa genera un conjunto disperso de ubicaciones de objetos candidatos y la segunda etapa clasifica cada ubicación candidata como uno de los primeros planos o como fondo utilizando una red neuronal convolucional. A través de una secuencia de avances, este marco de dos etapas logra constantemente la máxima precisión en el punto de referencia. Los detectores de dos etapas se han ido popularizando debido a la aparición de las redes neuronales convolucionales (R-CNN), donde un clasificador se aplica a un conjunto disperso de ubicaciones de objetos candidatos.

*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 (V1) de un cerebro biológico. Este tipo de red es una variación de un perceptron multicapa, sin embargo, debido a que su aplicación es realizada en matrices bidimensionales, son muy efectivas para tareas de visión artificial, como en la clasificación y segmentación de imágenes, entre otras aplicaciones aplicaciones.*¹⁶

En contraste, los detectores de una etapa que se aplican sobre un muestreo regular y denso de posibles ubicaciones de objetos son, potencialmente, más rápidos y simples, pero menos precisos que los detectores de dos etapas. La principal causa de ello es el extremo desequilibrio de clase encontrado durante la fase de entrenamiento de detectores densos.

Telenav, utiliza para la identificación de imágenes el detector denso simple, RetinaNet, basado en un modelo *deep learning* y desarrollado por Facebook. Cuando se entrena con la pérdida focal, RetinaNet es capaz de igualar la velocidad de los detectores de una etapa, mientras que supera la precisión de todos los detectores de dos etapas existentes. Habilitado por la pérdida focal, el detector RetinaNet supera a los mejores sistemas Faster R-CNN. Para lograr este resultado, se identifica el desequilibrio de clase durante el entrenamiento como el principal obstáculo que impide al detector de una etapa lograr la máxima precisión. El desequilibrio de clase se aborda en detectores de tipo R-CNN situados en cascada de dos etapas y muestreos heurísticos. La primera etapa reduce rápidamente el número de ubicaciones de objetos candidatos a un número pequeño (por ejemplo, 1000-2000), filtrando la mayoría de las

¹⁶ https://es.wikipedia.org/wiki/Redes_neuronales_convolucionales

muestras. En la segunda etapa de clasificación heurística de muestreo, se establece una relación fija de primer plano a fondo, para mantener un equilibrio entre primer plano y fondo.

En el caso clásico de los detectores de una etapa, se debe procesar un mayor conjunto de ubicaciones de objetos candidatos que se muestrean regularmente a través de una imagen. En la práctica esto a menudo equivale a enumerar una gran cantidad de ubicaciones que cubren densamente las posiciones espaciales, escalas, y relaciones de aspecto. Son ineficientes ya que la formación el procedimiento todavía está dominado por antecedentes fácilmente clasificados. Esta ineficiencia es un problema clásico en el objeto de detección que generalmente se aborda a través de técnicas tales como *bootstrapping* o minería.

6. Funcionamiento del algoritmo *Machine Learning*

En este apartado se evalúa la capacidad de resolución del algoritmo de aprendizaje automático con imágenes propias y como se debe proceder para poder probar la *I.A* desarrollada por Telenav.

6.1 Implementación de las pruebas

Tal como se describe en el apartado 3.2, en el que se han visto los diferentes entornos que permiten probar el funcionamiento del algoritmo, una de las posibles formas de ejecutar el algoritmo es mediante el script Unix que compila y lanza el código con el algoritmo de detección. En realidad, aunque se vieron diferentes formas de ejecutar la *IA* de Telenav, para poder probar datos de elaboración propia, a día de hoy, sólo es posible hacerlo a través del script de Unix ya que la web de OpenStreetCam no tiene habilitada la detección de señales de tráfico para todas las localizaciones geográficas, debido a la falta de una cantidad de imágenes de entrenamiento (ya etiquetadas) propias de cada localización que hagan que la ejecución del algoritmo tenga la fiabilidad suficiente. Las imágenes de ejemplo utilizadas por Telenav y, que sirven como semilla del algoritmo de aprendizaje, corresponden todas a EEUU, lo que supone que existan pequeñas diferencias en algunas señales de tráfico respecto a las europeas. Aunque esto no es impedimento para que el algoritmo

pueda detectar señales que no se corresponden a las que tiene etiquetadas en los ejemplos, ya que precisamente se trata de un algoritmo de *aprendizaje*, capaz de identificar nuevas situaciones en función de los datos extraídos del periodo de aprendizaje, si que supone una situación en la que el nivel de acierto en la identificación puede descender de forma notable. A continuación se muestran algunas imágenes que verifican esto.



Figura 21: Detección de semáforos en imágenes propias

En la imagen anterior puede verse como el sistema detecta correctamente los dos semáforos situados en el primer plano de la fotografía pero no los que están situados al fondo de la imagen, si bien esto no puede considerarse como un error, ya que además de aparecer en el plano más lejano, ni siquiera aparecen con nitidez en la imagen.



Figura 22: Señales no detectadas en imágenes propias

A continuación se verá un ejemplo de detección de una señal de “ceda el paso”, en este caso la morfología de la señal americana y española es prácticamente idéntica como se puede ver en la siguiente figura.



Figura 23: Señales de ceda el paso de EEUU (izquierda) y española (derecha)

En la imagen siguiente se puede ver como el algoritmo identifica de manera correcta la señal de ceda el paso situada en la parte derecha de la calzada.



Figura 24: Detección de señal de ceda el paso

En el siguiente ejemplo se verá un caso en el que el algoritmo no es capaz de detectar una señal de límite de velocidad. Como se puede observar en la figura 25, en este caso, las señales de limitación de velocidad estadounidense y

española son totalmente diferentes, además ni siquiera expresan la misma magnitud, ya que en EEUU la limitación está expresada en millas por hora.

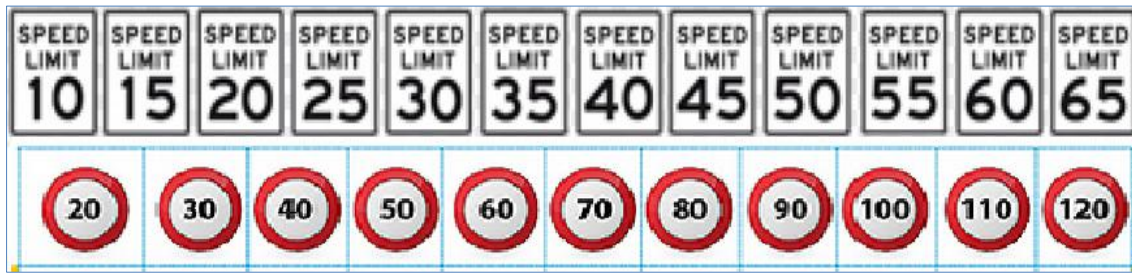


Figura 25: Arriba, señales de límite de velocidad en EEUU, abajo, límites en España

En la siguiente figura se puede observar como el algoritmo no identifica las dos señales de limitación de velocidad situadas en los laterales de la calzada.



Figura 26: Señales de límite de velocidad no detectadas por el algoritmo

Por último, veremos otro ejemplo de identificación de una señal de stop, al igual que en el caso de la señal de “ceda el paso”, las señales de stop son idénticas en EEUU y en España.



Figura 27: Identificación de señal de stop

6.2 Evaluación de resultados

Como se ha podido ver en el apartado anterior donde se ha probado el algoritmo sobre imágenes propias y, también, en el apartado 3.3 donde se ejecutó sobre imágenes proporcionadas por *OpenStreetCam*, se puede decir que, en líneas generales, el sistema de identificación tiene una fiabilidad muy elevada y, que para señales conocidas en las imágenes de entrenamiento, supera el 95% de tasa de acierto. En realidad, para hacerse una idea fiable acerca del grado de resolución del algoritmo es más razonable tener en cuenta los resultados observados en el apartado 3.3 que en el punto anterior. El motivo es, como ya se ha comentado, que las imágenes de ejemplo que sirven para alimentar y entrenar al algoritmo contienen señales, en la gran mayoría de casos, muy diferentes en forma y color a las contenidas por las imágenes propias. Se debe recordar que las imágenes contienen píxeles de diferente nivel RGB, el algoritmo analiza diferentes grupos de ellos, los agrupa y calcula la probabilidad de que correspondan a uno de los conjuntos (en este caso señales de tráfico) que tiene identificados en las imágenes de ejemplo. En el

caso de utilizar señales totalmente diferentes a las que alimentan el algoritmo el valor de la probabilidad de identificarlo como una señal reconocida disminuye, en la mayor parte de los casos, por debajo del umbral de detección.

Dentro del campo del *Machine learning*, se pueden presentar diferentes ejemplos que sirven para visualizar lo expuesto en el párrafo anterior. Una situación análoga sería, por ejemplo, un algoritmo *Machine Learning* de reconocimiento del lenguaje, en el que el algoritmo es capaz de identificar las diferentes palabras que forman un idioma a través de un texto de gran extensión¹⁷ proporcionado como ejemplo y, en el que las palabras pueden tener una identificación totalmente diferente en función de todas sus predecesoras (estados anteriores). En este caso, una situación similar a la producida en el algoritmo de reconocimiento de imágenes sería utilizar un texto de ejemplo en un idioma diferente al que queremos detectar. Lo importante, tanto en este ejemplo, como en el nuestro, es que simplemente modificando el texto o las imágenes previamente etiquetadas, en nuestro caso, suministrados como ejemplo del algoritmo este podría adaptarse y resolver el nuevo problema sin necesidad de modificar el código del programa.

6.3 Análisis del algoritmo

En este apartado se expondrá en mayor profundidad el modelo de detección de imágenes utilizado por Telenav, RetinaNet, al que ya hizo mención en el apartado 4. RetinaNet es una solución reciente a problemas producidos en los detectores clásicos, antes de ver sus características se hará una retrospectiva de las soluciones *deep learning* implementadas históricamente para tratar la detección de objetos en imágenes.

Los primeros detectores trataban de resolver la identificación de un único objeto mediante la aplicación de modelos *deep learning*, frecuentemente, redes neuronales. La aplicación de las redes neuronales a la identificación de objetos intenta resolver problemas de clasificación (predecir clases) y problemas de

¹⁷ Algunos de los textos de ejemplo utilizados en este tipo de algoritmo pueden ser libros como *El Quijote* o *La Biblia*

regresión (predecir valores continuos). La red neuronal que toma la imagen como entrada tiene que predecir 4 valores que representan las coordenadas del cuadro delimitador pronosticado y que se comparará con las cuatro coordenadas del cuadro delimitador especificado en la etiqueta. Además de eso, la red neuronal tiene que predecir una probabilidad de clase para cada una de las n categorías que se deseen clasificar, con lo que la red deberá predecir $4 + n$ valores. Para implementar el detector único, se utiliza cualquier arquitectura de red clasificadora de imágenes convolucional (entrenada previamente), como VGG-16 o ResNet eliminando cualquier capa de clasificación en la parte superior de la red. También se debe eliminar cualquier capa de agrupación (adaptativa) antes de las capas de clasificación, ya que destruyen la información espacial necesaria para retroceder las coordenadas de los bordes de los cuadros delimitadores. Predecir la clase del objeto (n probabilidades de clase) es un problema de clasificación, mientras que predecir las cuatro coordenadas para el cuadro delimitador es un problema de regresión.

El siguiente hito pasa por la aparición de los sistemas de detección de múltiples objetos (de un número desconocido) en una sola imagen. La visión por computadora clásica a menudo usaba un enfoque denominado *ventana deslizante*. En dicho enfoque un detector se desliza sobre una imagen y, una vez que detecta un objeto, se dibuja un cuadro delimitador alrededor del área en la que el detector está posicionado.

Posteriormente aparecen los sistemas de 2 etapas, también conocidos como métodos de *propuesta de región*. La primera etapa predice un conjunto de ubicaciones de objetos candidatos. El segundo estado o etapa, una red de convolución, clasifica los objetos en esas ubicaciones candidatas como una de las categorías buscadas o como fondo. Los métodos de propuesta de región producen resultados de última generación, pero a menudo son demasiado costosos computacionalmente para la detección de objetos en tiempo real, especialmente en sistemas integrados.

YOLO¹⁸ (*You only look once*) y SSD¹⁹ (*Single shot detector*) son arquitecturas de detección de imágenes en tiempo real de una sólo etapa que resuelven el problema aparejado al coste temporal y computacional, al predecir las coordenadas del cuadro delimitador y las probabilidades para diferentes categorías en un solo paso a través de la red. Están optimizados para obtener la mayor velocidad posible en la detección a costa de sacrificar la precisión. En estas arquitecturas la imagen de entrada obtenida va directamente a una gran red neuronal convolucional. Dentro de la red, la imagen de entrada se divide en muchas celdas, a las que se les asigna puntuaciones de clasificación y coordenadas de delimitación. Las coordenadas de la caja y las escalas se determinan en cada cuadrícula. Por otro lado, las clases de objetos generales y los cuadros delimitadores son calculados en base a los resultados obtenidos de cada celda. Estos dos enfoques reducen aún más el entrenamiento y el tiempo de resolución, en detrimento de la precisión en comparación con los métodos de propuestas regionales²⁰

En la siguiente imagen, se puede ver como definen varias cuadrículas de cajas predeterminadas de diferentes tamaños que permiten detectar objetos en diferentes escalas en una sola pasada. Muchas variantes a estas arquitecturas anteriores detectan objetos a diferentes escalas, por ejemplo, pasando versiones de diferente tamaño de la misma imagen al detector, lo que es computacionalmente más costoso.



Figura 28: División de la imagen en regiones o cajas de anclaje
<https://towardsdatascience.com/retinanet-how-focal-loss-fixes-single>

¹⁸ <https://pjreddie.com/darknet/yolo/>

¹⁹ <https://towardsdatascience.com/review-ssd-single-shot-detector-object-detection-851a94607d11>

²⁰ CNNs for Face Detection and Recognition, Stanford University

Para cada cuadro predeterminado en cada cuadrícula, la red genera n probabilidades de clase y 4 compensaciones a las respectivas coordenadas del cuadro predeterminado que dan las coordenadas pronosticadas del cuadro delimitador.

SSD y YOLO únicamente necesitan un pase hacia adelante a través de la red para predecir cuadros de delimitación de objetos y probabilidades de clase. En comparación con los sistemas de ventanas deslizantes y los métodos de propuesta de región, son mucho más rápidos y, por lo tanto, adecuados para la detección de objetos en tiempo real. El SSD (que utiliza mapas de características convolucionales de escala múltiple en la parte superior de la red en lugar de capas totalmente conectadas como lo hace YOLO) es, a su vez, más rápido y más preciso que YOLO.

RetinaNet intenta abordar la falta de precisión de los métodos como SSD y, para ello, propusieron abordar el problema reescalando la función de pérdida. Con esta mejora, los métodos de disparo único no solo son más rápidos que los métodos de dos etapas, sino que también son tan precisos que permiten nuevas aplicaciones en el campo de detección de objetos en tiempo real. RetinaNet integra la precisión de los métodos de dos etapas con la velocidad de los métodos de una etapa. La ventaja que tienen los métodos de dos etapas, que les dota de una mayor precisión, es que primero predicen *algunas* ubicaciones de objetos candidatos y luego usan una red neuronal convolucional para clasificar cada una de estas ubicaciones de objetos candidatos como una de las clases o como fondo. El énfasis aquí está en *algunas ubicaciones candidatas*. Métodos como SSD o YOLO sufren un *desequilibrio de clase* extrema.

*Al enfrentarse a la situación de crear un modelo de clasificación es habitual que las clases no se encuentran balanceadas. Esto es, el número de registros para una de las clases es inferior al resto. Cuando el **desequilibrio** es pequeño, uno a dos, esto no supone un problema, pero cuando es grande es un problema para la mayoría de los modelos de clasificación. Esta situación se conoce como el **Problema del Desequilibrio de Clases** (Class Imbalance Problem)²¹*

Los detectores evalúan aproximadamente entre diez y cien mil ubicaciones candidatas y, por supuesto, la mayoría de estas cajas no contienen un

²¹ <https://www.analyticslane.com/2018/07/04/el-problema-de-desequilibrio-de-clases-en-conjuntos-de-datos-de-entrenamiento/>

objeto. Incluso si el detector clasifica fácilmente esta gran cantidad de cajas como negativas (o fondo de imagen), supone un problema en la identificación.

RetinaNet es una red única y unificada compuesta por una red troncal y dos subredes específicas de tareas. La red troncal es responsable de calcular un mapa de características de convolución sobre una imagen de entrada completa. La primera subred realiza la clasificación en la salida de la red troncal, la segunda realiza la regresión del cuadro delimitador de convolución.

- **Red troncal:** Red de Piramide construida sobre ResNet50 o ResNet101, aunque puede usar cualquier otro clasificador para diseñar la red.

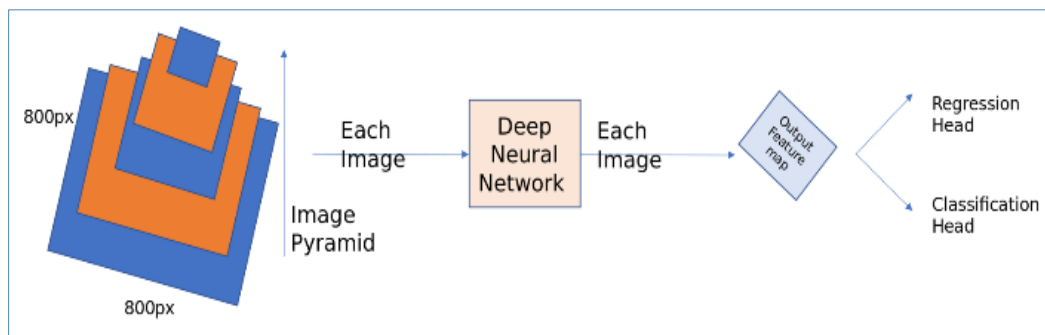


Figura 29: Red piramidal en detección de imágenes

<https://medium.com/@14prakash/the-intuition-behind-retinanet>

- **Subred de clasificación:** predice la probabilidad de presencia del objeto en cada posición espacial para cada uno de los anclajes y clases de objetos. Toma un mapa de características de entrada con canales desde un nivel piramidal, la subred aplica cuatro capas, cada una con filtros y seguida por activaciones ReLU²² (unidad lineal rectificada). Finalmente se adjuntan activaciones sigmoideas a las salidas.
- **Subred de regresión de cuadro:** similar a la red de clasificación utilizada, pero los parámetros no se comparten. Da salida a la ubicación del objeto con respecto al cuadro de ancla si existe un objeto.

²² Unidades utilizadas por la función de activación de redes neuronales conocida como rectificador.
[https://es.wikipedia.org/wiki/Rectificador_\(redes_neuronales\)](https://es.wikipedia.org/wiki/Rectificador_(redes_neuronales))

En la siguiente figura se puede ver un diagrama que resume el funcionamiento de RetinaNet y como funcionan su arquitectura de red. RetinaNet utiliza la Red de pirámide sobre la red neuronal convolucional ResNet como una red troncal para generar una pirámide de característica convolucional. La subred de clase clasifica cuadrículas de anclaje, y, por último, esta retrocede desde las regiones de anclaje a las de objetos de verdad.

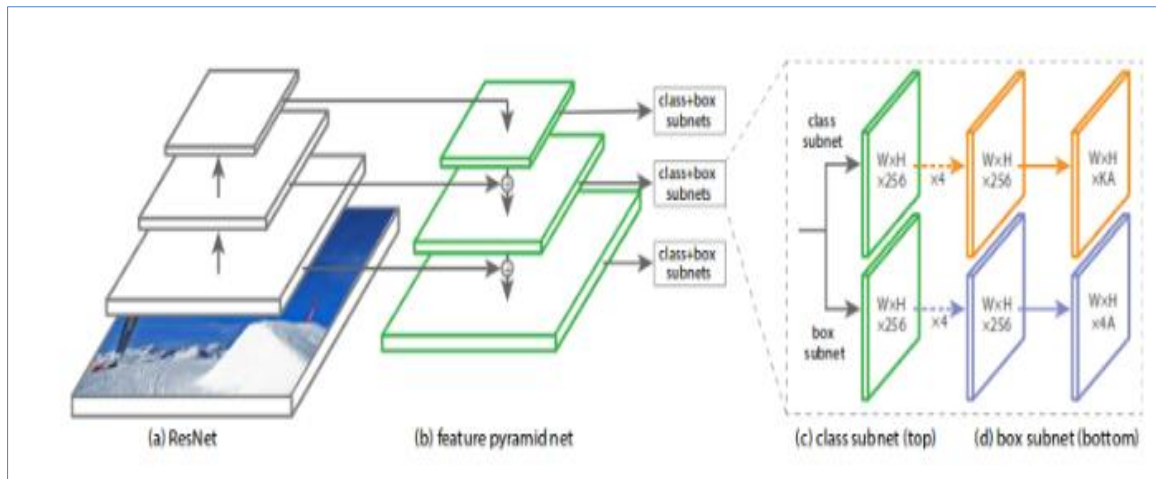


Figura 30: Arquitectura de red en RetinaNet

<https://medium.com/@14prakash/the-intuition-behind-retinanet>

7. Aplicaciones

Como se ha visto a lo largo de los apartados anteriores la visión por computador tiene infinidad de aplicaciones, pero si se concreta en la detección de señales de tráfico hay dos, estrechamente relacionadas, que destacan por encima del resto, se trata de los sistemas de conducción autónomos y los sistemas de seguridad activa y pasiva incluidos actualmente una gran cantidad de modelos. Los sistemas de reconocimiento de señales de tráfico se incluyen dentro de los sistemas denominadas ADAS (*Advanced driver assistance systems*). Esta tecnología tuvo su aparición comercial a finales de 2008, en sus comienzos sólo detectaban los límites de velocidad, actualmente son capaces de detectar la mayoría de señales, las líneas que delimitan la calzada e incluso el estado de fatiga del conductor.

Los vehículos con este sistema incorporan una cámara, normalmente situada en la parte central, en el interior de la luna delantera, cuyas imágenes son

tratadas digitalmente para que un sistema en tiempo real procese y evalúe los datos obtenidos. Esta tecnología puede aplicarse de manera pasiva o activa. En el primer caso se avisa al conductor mediante pictogramas, sonidos o cualquier otro tipo de señales cuando se produce una infracción o existe un especial riesgo de accidente. En el segundo caso el sistema envía una señal que actúa sobre el control del vehículo haciendo que este reduzca la velocidad o incluso se detenga.

Una implementación mejorada de esto son los sistemas de conducción autónomos. Existen diferentes implementaciones en función de los diferentes fabricantes, uno de los que más ha apostado por este modelo es Tesla. El piloto automático de Tesla, es una característica avanzada de la funcionalidad de asistencia al conductor que cuenta con un sistema capaz de detectar el abandono de carriles, control de crucero adaptativo, auto-estacionamiento y capacidad de cambiar automáticamente de carril con confirmación del conductor. Las últimas mejoras incluyen la transición de una carretera a otra y salir de la autovía cuando el destino del usuario está cerca.

La intención de la compañía es ofrecer auto conducción completa en un futuro, reconociendo, no obstante, que deben superarse los obstáculos legales, regulatorios y técnicos para lograr dicho

8. Conclusiones

A lo largo de este trabajo se han estudiado los diferentes componentes que integran OpenStreetCam, el cual se ha presentado como un proyecto altamente colaborativo, tanto a la hora de proporcionar imágenes que permitan referenciar cualquier localización, como, incluso, en la implementación de sus aplicaciones. Este trabajo se ha centrado, sobretodo, en la I.A, desarrollada por Telenav, que confiere a OpenStreetCam la capacidad de detección de señales viales en las imágenes previamente suministradas por los usuarios. También, se ha visto el gran potencial que tiene esto junto con la aplicación móvil que permite capturar imágenes de recorridos completos georeferenciados y, aunque ahora mismo no es algo disponible al usuario, sería capaz de identificar

los diferentes elementos viales en tiempo real, pudiendo convertirse en un instrumento de gran utilidad en la seguridad vial de los conductores.

Respecto a la *I.A* desarrollada por Telenav se ha presentado como un sistema formado por dos partes muy diferenciables, por un lado, el algoritmo *Machine Learning* capaz de definir un estado (en este caso un elemento vial) y, por otro lado, la librería de visión por computador RetinaNet, capaz de realizar la identificación de elementos en imágenes. Respecto a esto último, es importante recalcar su alta eficiencia y rendimiento, lo que lo hace idóneo para su utilización en sistemas de detección en tiempo real, pudiendo usarse, por ejemplo, en aplicaciones de seguridad vial incorporado en los sistemas *ADAS* en vehículos de tracción motora.

Uno de los objetivos principales planteados en la propuesta de este trabajo era la evaluación del software de detección, en este punto se ha visto el alto rendimiento que ofrece el algoritmo, con una capacidad de acierto del 90-95% sobre elementos detectados, también se ha comprobado que la capacidad de acierto bajaba (alrededor de 10-15 puntos porcentuales) si se incluían los elementos no detectados. En este punto se ha visto la importancia de las imágenes de ejemplo utilizadas por el algoritmo y como es indispensable disponer de un buen número de ellas con todos los elementos que posteriormente queramos detectar, ya que servirán para que el sistema realice la mejor elección (el estado más probable en términos *Machine Learning*) a la hora de enfrentarse a una imagen desconocida. Por tanto, el uso de un buen set de imágenes (con un gran número de imágenes y elementos) de entrenamiento es un elemento fundamental para conferir al algoritmo un rendimiento elevado. Cabe mencionar en este aspecto, que a mayor número de imágenes mayor es el coste computacional y, por tanto el tiempo empleado para la identificación, con lo cual el tamaño del set de ejemplo utilizado tiene que permitir un alto grado de acierto sin que esto produzca una merma en el tiempo de ejecución. Según la aplicación para la que se requiera el software se podrá sacrificar uno en beneficio del otro.

9. Trabajo futuro

Como ya se ha apuntado en el apartado anterior el equilibrio entre precisión y velocidad depende de la cantidad de imágenes de ejemplo procesadas en la etapa de entrenamiento del algoritmo. Por tanto, las líneas de acción que permiten mejorar el rendimiento del sistema han de pasar por tratar el cuello de botella generado al procesar un número elevado de imágenes en esa etapa con la finalidad de obtener la mayor precisión posible.

Actualmente las técnicas que permiten procesar grandes volúmenes de datos están presentes en multitud de servicios y aplicaciones, de entre ellas no todas son válidas para su uso en sistemas que requieran procesamiento de datos en tiempo real. Para nuestro caso de uso la técnica que mejor se adapta para mejorar la velocidad en la etapa de procesamiento de imágenes de ejemplo es el **procesamiento en stream**.

El streaming o procesamiento de flujo implementa un modelo de flujo de datos asociados a series de tiempo que fluyen continuamente a través de una red de entidades de transformación que componen un sistema. Sus únicas limitaciones son:

- Se debe disponer de suficiente memoria para almacenar entradas en cola.
- La tasa de productividad del sistema a largo plazo debería ser más rápida, o por lo menos igual, a la tasa de entrada de datos en ese mismo periodo. Si esto no fuese así, los requisitos de almacenamiento del sistema crecerían sin límite.

Entre sus principales características están:

- Procesamiento en tiempo real.
- Procesamos los datos en el momento mismo de generarse.
- Más rápido, procesamos los datos antes de llegar al disco.

Una API concreta candidata a ocuparse del procesamiento de imágenes es Apache Spark Streaming²³.

²³ <http://www.diegocalvo.es/spark-streaming/>

10. Glosario

Deep Learning: *El aprendizaje profundo, también conocido como redes neuronales profundas, es un aspecto de la inteligencia artificial (AI) que se ocupa de emular el enfoque de aprendizaje que los seres humanos utilizan para obtener ciertos tipos de conocimiento.*

JOSM: *Es el acrónimo de Java OpenStreetMap Editor, es un software libre de escritorio programado en Java para la edición de datos en el proyecto OpenStreetMap.*

Machine Learning: *Es una disciplina científica del ámbito de la Inteligencia Artificial que crea sistemas que aprenden automáticamente; aprender en este contexto quiere decir, identificar tipos de patrones complejos en millones de datos de forma concreta.*

Mapillary: *Es un servicio en línea para compartir fotos geo etiquetadas desarrollado por una compañía sueca. Su objetivo es representar el mundo entero (no solamente sus calles) con fotos mediante colaboración masiva.*

OpenStreetCam: *Es un proyecto colaborativo de código abierto que tiene como objetivo la representación mediante imágenes de cualquier localización geográfica.*

OpenSource: *Cualquier programa cuyo código fuente se pone a disposición para su uso o modificación, conforme los usuarios u otros desarrolladores lo consideren conveniente.*

Red Neuronal Convolucional: *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.*

RetinaNet: *Librería de detección de objetos desarrollada por Facebook que utiliza una arquitectura basada en redes neuronales para la identificación de entidades.*

Script: *Programa simple, que por lo regular se almacena en un archivo de texto plano. Par interactuar interactuar con el sistema operativo o con el usuario.*

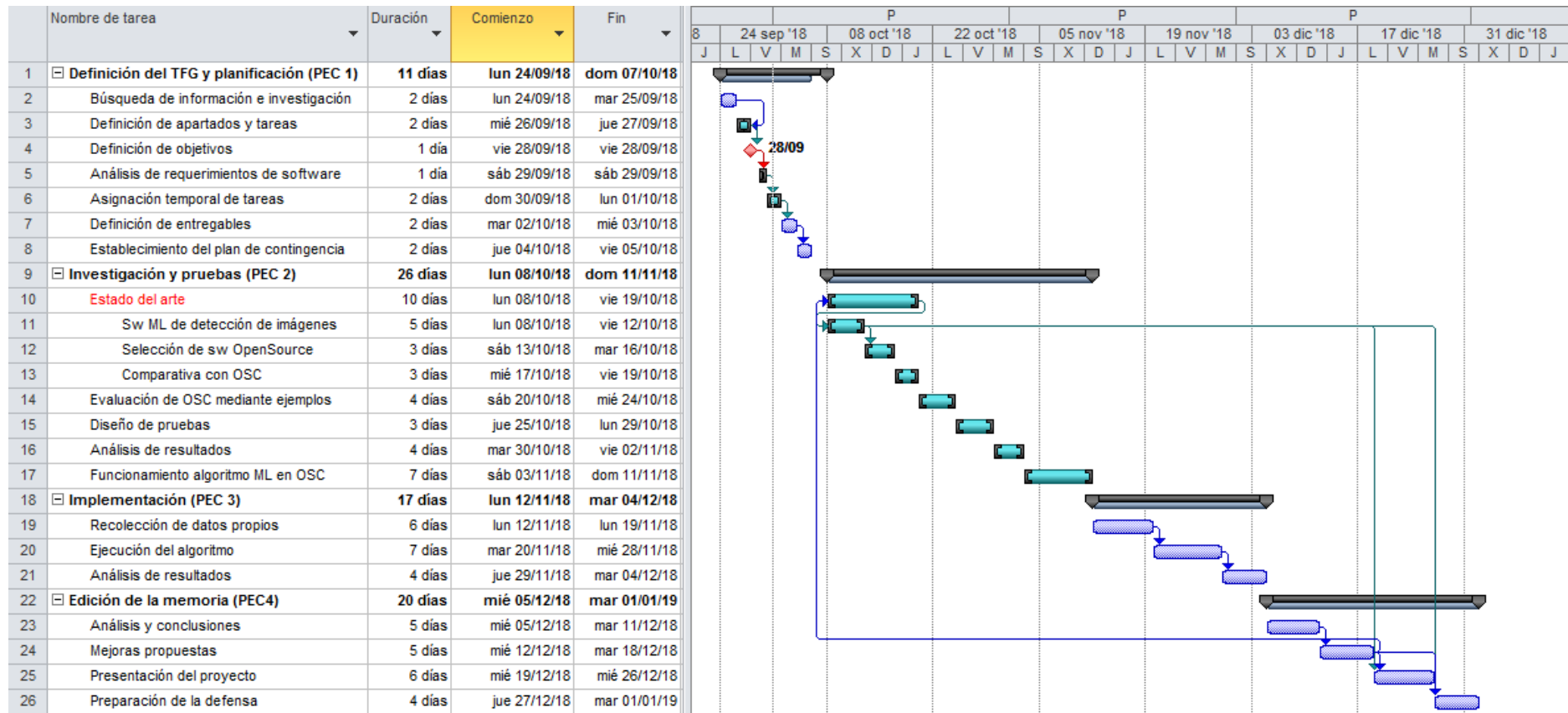
Telenav: *Es una empresa de servicios basados en ubicación inalámbrica que brinda funciones que incluyen navegación satelital con sistema de posicionamiento global, búsqueda local, soluciones de navegación automotriz, publicidad móvil, movilidad empresarial y automatización de flujos de trabajo.*

Bibliografía

- [1] https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico
- [2] <https://medium.com/soldai/tipos-de-aprendizaje-autom%C3%A1tico-6413e3c615e2>
- [3] <https://www.raona.com/los-10-algoritmos-esenciales-machine-learning/>
- [4] <https://es.wikipedia.org/wiki/Mapillary>
- [5] <https://wiki.openstreetmap.org/wiki/OpenStreetCam>
- [6] <http://www.aprendemachinelearning.com/aplicaciones-del-machine-learning/>
- [7] <https://cloud.google.com/vision/>
- [8] <https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/home>
- [9] <https://aws.amazon.com/es/rekognition/>
- [10] <https://luminoth.readthedocs.io/en/latest/>
- [11] <https://es.wikipedia.org/wiki/TensorFlow>
- [12] <https://en.wikipedia.org/wiki/OpenCV>
- [13] https://en.wikipedia.org/wiki/Visual_odometry
- [14] <https://josm.openstreetmap.de>
- [15] <https://www.openstreetmap.org>
- [16] https://es.wikipedia.org/wiki/Redes_neuronales_convolucionales
- [18] <https://pjreddie.com/darknet/yolo/>
- [19] <https://towardsdatascience.com/review-ssd-single-shot-detector-object-detection-851a94607d11>
- [20] CNNs for Face Detection and Recognition, Stanford University
- [21] <https://www.analyticslane.com/2018/07/04/el-problema-de-desequilibrio-de-clases-en-conjuntos-de-datos-de-entrenamiento>
- [22] [https://es.wikipedia.org/wiki/Rectificador_\(redes_neuronales\)](https://es.wikipedia.org/wiki/Rectificador_(redes_neuronales))
- [23] <http://www.diegocalvo.es/spark-streaming/>
- Long, Jonathan; Shelhamer, Evan; Darrell, Trevor "Fully Convolutional Networks for Semantic Segmentation". UC Berkley
- <https://www.analyticslane.com/2018/07/04/el-problema-de-desequilibrio-de-clases-en-conjuntos-de-datos-de-entrenamiento/>
- Ng, A. (2015). Machine learning. Universidad de Stanford

Anexo I

1. Diagrama de Gantt.



Anexo II

1. Requisitos técnicos

El software necesario para el desarrollo de las diferentes actividades necesarias para la realización de este proyecto es:

- Microsoft Office Word version 2010
- Microsoft Office Power Point version 2010
- Microsoft Office Project version 2010
- OpenStreetCam (<http://openstreetcam.org/map>)

2. Análisis de riesgos y plan de contingencia.

La evaluación del nivel de un riesgo se ha realizado en base al siguiente producto entre su probabilidad y el impacto asociado.

Impacto	Alta	Medio	Alta	Alta
	Media	Medio	Medio	Alta
	Baja	Baja	Baja	Medio
		Baja	Media	Alta
		Probabilidad		

A continuación se describen los posibles riesgos que pueden producirse y las acciones que deberán ejecutarse para mitigar su impacto.

Código	Descripción	Probabilidad **	Impacto **	Plan de mitigación
R001	Interacción con las obligaciones laborales	Media	Medio	Reestructuración de la planificación del TFG para cumplir con los plazos establecidos en las tareas principales.
R002	Dificultades asociadas al software OpenStreetCam	Baja	Bajo	Incrementar el número de horas de uso de OpenStreetCam

