



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
INFORMÁTICA

MÁSTER UNIVERSITARIO EN VISIÓN ARTIFICIAL

TRABAJO FIN DE MÁSTER

Título

Autor: Jessica Fernández Martínez

Tutor: José María Cañas Plaza

Curso académico 2018/2019

Agradecimientos

Me gustaría dedicar este proyecto a toda la gente que me ha apoyado durante su realización. Sobre todo a mis padres y hermana que siempre están ahí y me ayudan en todo lo que necesito, pues sin ellos no hubiese sido posible conseguirlo.

Agradecer a mi familia todo el interés que han mostrado y como han conseguido que el trabajo fuera más sencillo. También quiero mencionar a mis amigos, los cuales siempre me han dado ánimos e incluso ideas para poder realizar el proyecto.

Por último, y no menos importante, quiero agradecer a mi tutor José María todo el apoyo y sabiduría que me ha dado durante el proceso. Sin su dedicación y paciencia nada hubiera sido posible.

Resumen

La gestión del tráfico es un tema muy complejo, pero con mucha importancia pues tiene grandes repercusiones. Esta gestión siempre se ha realizado de forma *clásica*, mediante una cámara que se centraba únicamente en capturar imágenes y un operario que era el encargado de extraer información de estas. En las últimas décadas se está tratando de automatizar todas aquellas cosas que pueden realizarse sin necesidad de que el ser humano actúe tan activamente.

Las cámaras son el sensor principal en la gestión del tráfico, ya que son las que capturan las imágenes de donde posteriormente se extraerá la información. Por tanto se puede decir que las imágenes son las que saben que está ocurriendo realmente en todo momento. Partiendo de esta premisa cada vez se está tendiendo más a diseñar algoritmos capaces de extraer información automáticamente de dichas imágenes, pues como se ha dicho son las principales fuentes de información. Por supuesto, esto puede extenderse a la monitorización de vehículos.

El objetivo de este trabajo es crear un sistema capaz de extraer información de las imágenes con el objetivo de llegar a monitorizar una carretera mediante una única cámara. Este sistema se llama *Smart-Traffic-Sensor* y parte de un sistema previo denominado *Traffic-Sensor* [1], en el cual se llevaba a cabo la monitorización del tráfico haciendo uso de técnicas clásicas.

Smart-Traffic-Sensor se centra en técnicas modernas como *Deep Learning* para la detección y clasificación de vehículos. La monitorización se fundamenta en criterios de proximidad espacial y Kanade–Lucas–Tomasi (KLT). Para poder realizar todo el proyecto ha sido necesario crear una base de datos con suficiente información y recopilar una serie de videos donde poder evaluarlo.

El sistema se ha evaluado con diversos videos en condiciones diferentes con el fin de saber como se comportaría ante posibles cambios meteorológicos o en condiciones de baja calidad de imagen.

Índice general

Índice de figuras	VI
Índice de tablas	VIII
1. Introducción	2
1.1. Cámaras en tecnologías automóviles y su entorno	4
1.2. Las cámaras como sensor de tráfico	11
1.2.1. Visión computacional en tráfico urbano	13
1.2.2. Monitorización de autopistas basada en visión	16
1.3. Sistemas comerciales basados en visión	17
1.4. Redes Neuronales	19
1.5. Objetivos	20
1.6. Estructura de la Memoria	21
2. Estado del arte	22
2.1. Detección de vehículos	23
2.2. Clasificación de vehículos	30
2.3. Seguimiento de vehículos	35
2.4. Bases de Datos para la detección de vehículos	37
2.4.1. GRAM Road-Traffic Monitoring	37
2.4.2. BIT-Vehicle Dataset	37
2.4.3. CarND-Vehicle-Detection	39
3. Herramientas	40
3.1. DeepLearningSuite	40
3.2. TensorFlow	43
3.3. Keras	44
3.4. Darknet	45
3.5. Biblioteca OpenCV	46
3.6. Lenguaje C++	47

3.7. Lenguaje Python	48
3.8. Gtkmm	49
4. Diseño Global del Sistema	50
4.1. Diseño	50
4.2. Bases de Datos de Vídeos	51
4.3. Funcionamiento General	52
4.3.1. Detección y Clasificación de Vehículos	53
4.3.1.1. TensorFlow	53
4.3.1.2. Keras	56
4.3.1.3. YOLO (Darknet)	57
4.3.1.4. Diseño Final	58
4.3.2. Seguimiento de Vehículos	62
4.3.3. Resumen General del Sistema	66
4.4. Estimación de la Velocidad	67
5. Experimentos	69
5.1. Evaluación de diversas redes neuronales	70
5.2. Evaluación ampliando Dataset	72
5.3. Evaluación de Darknet con diferentes conjuntos de imágenes	73
5.3.1. Red Neuronal con Imágenes de Buena Calidad	76
5.3.2. Red Neuronal con Imágenes de Buena Calidad y Malas Condiciones Meteorológicas	79
5.3.3. Red Neuronal con todo el conjunto de datos	80
5.4. Evaluación de Smart-Traffic-Sensor	80
5.5. Evaluación de los Tiempos de Procesamiento de las Redes	88
6. Conclusiones	89
6.1. Conclusiones	89
6.2. Trabajos Futuros	91
Bibliografía	101

Índice de figuras

1.1.	Centro de monitorización de tráfico	2
1.2.	División del sector SIT a nivel de aplicación	4
1.3.	Sistema de reconocimiento automático de matrículas	5
1.4.	Radar de tramo basado en reconocimiento de matrículas	5
1.5.	Sistema de ayuda a bordo del vehículo	6
1.6.	Aparcamiento automático	6
1.7.	Sistema de visión de ojo de pájaro para asistencia en el aparcamiento	7
1.8.	Cámara trasera de BMW para asistencia en el aparcamiento	7
1.9.	Sensores en vehículos autónomos	8
1.10.	Vehículo autónomo participante en el <i>DARPA Grand Challenge</i>	9
1.11.	Vehículo autónomo de Google	10
1.12.	Vehículo autónomo de Tesla	11
1.13.	Ejemplo de detección y seguimiento de vehículos de [2]	12
1.14.	Control de semáforos en rojo	14
1.15.	Monitorización del tráfico urbano	14
1.16.	Sistema de detección de incidencias en túnel	16
1.17.	Sistema de detección de incidencias en autopistas	16
1.18.	Sistema <i>TraffiCam</i> de FLIR	18
1.19.	Sistema <i>Mesta Fusion</i> de Morpho	18
1.20.	Neurona Biológica y Artificial	19
2.1.	Fases de R-CNN	29
2.2.	Fases de Faster R-CNN	30
2.3.	Detecciones del sistema de Zezhi Chen and Tim Ellis [3]	33
2.4.	Reconstrucción 3D empleando puntos de fuga	33
2.5.	Clasificación de vehículos basada en modelos 3D construidos mediante el uso de puntos de fuga	34
2.6.	Plantillas 3DHOG para la clasificación de vehículos	34
2.7.	Reconstrucción del histograma 3DHOG	35
2.8.	Imágenes de ejemplo del dataset GRAM Road-Traffic Monitoring	38

2.9.	Imágenes de ejemplo del dataset BIT-Vehicle	38
2.10.	Imágenes de ejemplo del dataset 2 de CarND-Vehicle-Detection	39
3.1.	Ejemplo de IOU	41
3.2.	Fórmula IOU	42
3.3.	Proceso de Yolo	46
3.4.	Funciones de OpenCV	47
4.1.	Red SSD Mobilenet V2	53
4.2.	Red SSD	56
4.3.	Red VGG-16	57
4.4.	Arquitectura YOLO	58
4.5.	Zonas de entrada y seguimiento	59
4.6.	Zona Evaluación	60
4.7.	Zonas identificadas en la zona de Evaluación	61
4.8.	Evolución del área alrededor del vehículo	63
4.9.	Elipse asociada a los vehículos	63
4.10.	Seguimiento con proximidad espacial	64
4.11.	KLT Piramidal	66
5.1.	Ejemplos de Imágenes de Buena Calidad	74
5.2.	Ejemplos de Imágenes de Malas Condiciones Climatológicas (Niebla y lluvia)	74
5.3.	Ejemplos de Imágenes de Mala Calidad	74
5.4.	Detecciones Smart-Traffic-Sensor Vídeo Buena Calidad	82
5.5.	Detecciones Smart-Traffic-Sensor Vídeo Malas Condiciones Meteorológicas	84
5.6.	Detecciones Smart-Traffic-Sensor Vídeo Mala Calidad	85

Índice de tablas

4.1. Muestras de la Base de Datos	52
4.2. Imágenes de la Base de Datos	52
5.1. Muestras de la Base de Datos del 1º entrenamiento	70
5.2. Muestras de los datos de Test de la primera evaluación	71
5.3. Resultados Redes Pre-Entrenadas	71
5.4. Resultados Redes Entrenadas	71
5.5. Características de la Base de Datos de Imágenes de Buena Calidad	72
5.6. Resultados Redes Entrenadas	73
5.7. Base de Datos Final	75
5.8. Características de la Base de Datos con condiciones climatológicas desfavorables	75
5.9. Características de la Base de Datos de Mala Calidad	76
5.10. Imágenes de Test de Buena Calidad	77
5.11. Imágenes de Test de Malas Condiciones Meteorológicas	77
5.12. Imágenes de Test de Mala Calidad	78
5.13. Resultados Modelo entrenado con Imágenes de Buena Calidad	78
5.14. Conjunto de Test Combinado	78
5.15. Resultados Modelo entrenado con Imágenes de Buena Calidad y Condiciones Meteorológicas Malas	79
5.16. Resultados Modelo entrenado con Toda la Base de Datos	80
5.17. Imágenes de Test del Vídeo de Buena Calidad	82
5.18. Resultados Vídeo de Buena Calidad	83
5.19. Imágenes de Test del Vídeo de Malas Condiciones Meteorológicas	84
5.20. Resultados Vídeo de Malas Condiciones Meteorológicas	85
5.21. Imágenes de Test del Vídeo de Mala Calidad	86
5.22. Resultados Vídeo de Mala Calidad	86
5.23. Resultados para las Diferentes Categorías en Vídeo de Buena Calidad	87
5.24. Resultados para las Diferentes Categorías en Vídeo de Mala Calidad	88
5.25. Tiempos de procesamiento	88

6.1. Resultados de Y. Abdullah, G. Mehmet, A. Iman and B. Erkan [4]	90
6.2. Resultados de Y. Abdullah, G. Mehmet, A. Iman and B. Erkan [4]	91

Acrónimos

CNN Convolutional Neural Network.

CNTK The Microsoft Cognitive Toolkit.

COCO Common Objects in Context.

CUDA Compute Unified Device Architecture.

EM Expectation Maximization.

EOH (Edge Orientation Histogram.

FPS Frames Per Second.

GMM Gaussian mixture models.

GRAM-RTM GRAM Road-Traffic Monitoring.

HDF5 Hierarchichal Data Format version 5.

HOG Histogram of Oriented Gradients.

IA Inteligencia Artificial.

ICE Internet Communications Engine.

IOU Interseccion Over Union.

IPHOG Intensity-based on a Pyramid of Histogram of Gradient Orientations.

KAA Kernel Auto Associator.

KLT Kanade–Lucas–Tomasi.

mAP mean Average Precision.

mAR mean Average Recall.

CAPÍTULO 0. INTRODUCCIÓN

MBF Measurement Based Features.

MEI Motion Energy Images.

MOG Mixture of Gaussian.

NMS Non-Maximum Suppression.

OpenCV Open Source Computer Vision Library.

R-CNN Region-based Convolutional Neural Network.

RNA Redes Neuronales Artificiales.

SAD Sum of Absolute Differences.

SIFT Scale-Invariant Feature Transform.

SSD Single Shot Detector.

SVM Support Vector Machine.

TFM Trabajo de Fin de Máster.

VA Visión Artificial.

XML Extensible Markup Language.

YOLO You only look once.

Capítulo 1

Introducción

Debido al crecimiento en la población se ha visto incrementada la cantidad de vehículos que se encuentran en circulación. Este incremento en la densidad de vehículos ha afectado en el tráfico y en la contaminación. Por ello se han ido buscando formas de mejorar la seguridad vial y el rendimiento en las carreteras. En este punto es donde intervienen las nuevas tecnologías, las cuales son capaces de ofrecernos mucha información para poder mejorar el tráfico. Esta aplicación de sistemas avanzados de información y comunicación a las infraestructuras de transporte y a los vehículos para mejorar la seguridad vial, la eficiencia y el rendimiento de las carreteras se denomina Sistemas inteligentes de transporte (SIT). Este área está en continuo desarrollo y con ello se pretende dotar a los vehículos de inteligencia para aportarnos información con la cual mejorar nuestra seguridad y monitorizar las redes de transporte para detectar posibles incidencias y mejorar el tráfico.



Figura 1.1: Centro de monitorización de tráfico

Los sistemas tradicionales de monitorización ofrecen una información muy limitada, pues normalmente es un humano el que analiza toda la información que le aportan las

CAPÍTULO 1. INTRODUCCIÓN

cámaras y con ello toma las decisiones. Gracias a los últimos avances en visión artificial muchas de las tareas que realizaba el humano se han podido automatizar. Con la visión artificial somos capaces de procesar todas las imágenes que recogen las cámaras y extraer información de ellas, como por ejemplo la cantidad de tráfico que hay, las condiciones meteorológicas que tenemos, la existencia de incidencias, etc.

Los principales sectores de los SIT son:

1. Gestión avanzada del transporte
2. Gestión avanzada del transporte público
3. Sistemas de cobro automático

El primer sector hace referencia a la gestión de carreteras a nivel global. Sus funciones principales son agilizar el tráfico, mejorar la movilidad, optimizar el uso de las infraestructuras y mejorar la seguridad de las redes de transporte. Para conseguir todo esto se emplea como sensor principal las cámaras. Con dichas cámaras se obtienen imágenes en todo momento, las cuales son procesadas y analizadas por seres humanos o automáticamente mediante algoritmos de visión artificial.

El segundo sector tiene el mismo fin que el primer sector pero limitado a sistemas de transporte público. Entre sus funciones se encuentra gestionar de forma eficaz y optimizada las redes de transporte público, e informar a los ciudadanos acerca de las distintas alternativas de transporte disponibles así como sus horarios o sus posibles incidencias en tiempo real. Este sector también se encarga de todo lo referente a los sistemas de cobro y el manejo de billetes en el transporte. Su objetivo es facilitar la integración y unificación de los sistemas de cobro para facilitar la creación de billetes multimodales y tarjetas de transporte subvencionadas por el estado.

El último sector (sector de cobro automático) se encarga de todos los sistemas electrónicos de gestión de cuotas. La gestión de cobros automáticos en autopista con sistemas de telepeaje se basa en el empleo de sistemas inalámbricos de comunicación entre el sistema de gestión de cobro y el vehículo. En estos sistemas el vehículo debe disponer de un dispositivo, el cual permite identificarlos de manera segura y gestionar su cuota sin necesidad de que el vehículo se detenga en el telepeaje. Con esto se consigue reducir el atasco en los peajes.

A parte de la división a nivel sectorial de los SIT explicada, existe una división a nivel de aplicaciones. Esto puede verse en la Figura 1.2.

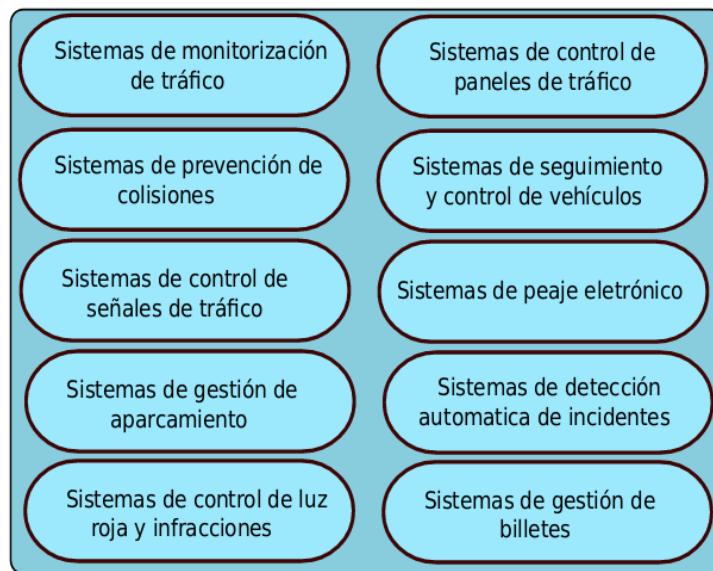


Figura 1.2: División del sector SIT a nivel de aplicación

1.1. Cámaras en tecnologías automóviles y su entorno

Si pensamos en una ciudad inteligente lo primero que nos viene a la cabeza son cámaras. Actualmente las cámaras se emplean en muchos ámbitos debido a la gran calidad que ofrecen, lo económicas que son y su reducido tamaño. Esto hace que sean uno de los mejores sensores de los que disponemos. Con las imágenes que nos ofrecen las cámaras necesitamos implementar algoritmos que sean capaces de procesarlas, extraer información y analizarla. Una de las aplicaciones pioneras donde la visión artificial ha demostrado ser una solución práctica frente a otras alternativas es la detección de matrículas. Este caso en concreto se usa en muchas puertas de acceso a aparcamientos, pues ha demostrado que tiene una gran fiabilidad. Además el hecho de encontrarse en puertas de acceso a aparcamientos hace que se trata de escenarios muy controlados, pues conocemos la iluminación del lugar, la posición que toman los vehículos, etc. Esto es una gran ventaja, pues nos aporta información adicional facilitándonos la identificación de las matrículas.



Figura 1.3: Sistema de reconocimiento automático de matrículas

Otra aplicación de las cámaras son los radares de tramo. En este caso se pretende calcular la velocidad media de cada vehículo haciendo uso del reconocimiento de matrículas. Las cámaras suelen situarse en los extremos de un túnel, ya que la trayectoria del vehículo está bajo control.



Figura 1.4: Radar de tramo basado en reconocimiento de matrículas

Otro campo en el que cada vez tienen mayor relevancia las cámaras es en seguridad vial. Hoy en día hay varios modelos de coches capaces de detectar señales de tráfico de manera automática. De esta forma el vehículo puede avisar al conductor del exceso de velocidad, de la dirección de la próxima curva, etc. Este tipo de sistemas se encuentran

CAPÍTULO 1. INTRODUCCIÓN

incorporados en vehículos de marcas comerciales como BMW 7-series, Ford Focus, Toyota, Opel Insignia, etc.



Figura 1.5: Sistema de ayuda a bordo del vehículo

Disponer de cámaras en los vehículos ha hecho posible aumentar la seguridad, pues nos aportan mucha información. Actualmente existen modelos de vehículos que nos avisan de la presencia de peatones e incluso poseen mecanismos de frenado automático evitando posibles atropellos.

Una aplicación de las cámaras muy presente en los vehículos es el asistente para el aparcamiento. En esta aplicación en concreto, las cámaras pueden aportar imágenes acerca de los alrededores de nuestro vehículo ,ofreciéndonos información acerca de como debemos aparcar, e incluso forman parte de sistemas multisensoriales para aparcar automáticamente.

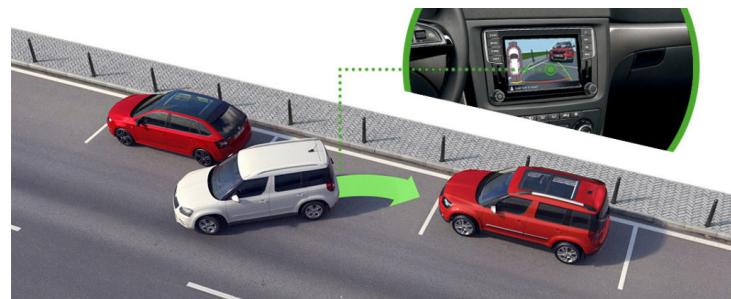


Figura 1.6: Aparcamiento automático

Un ejemplo de asistencia en el aparcamiento es el sistema de ojo de pájaro de Nissan. Dicho sistema ofrece una vista aérea simulada de la escena del aparcamiento. En ella se

CAPÍTULO 1. INTRODUCCIÓN

incluye el vehículo y sus alrededores, ofreciéndole al conductor gran información acerca de la situación. Con lo que podrá realizar la maniobra de forma más sencilla y segura. Para conseguir esto, el vehículo dispone de 4 cámaras, las cuales emplea para generar la simulación. A parte de esta simulación, el conductor puede usar la cámara trasera para obtener más detalles de la escena.

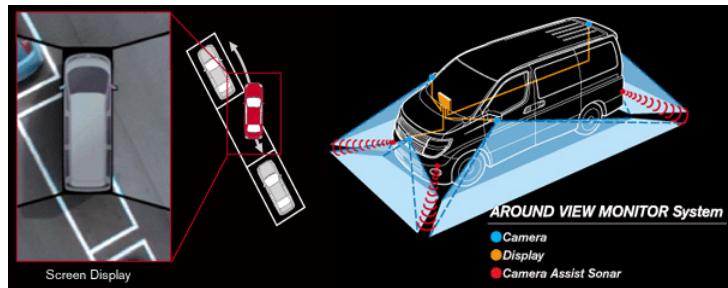


Figura 1.7: Sistema de visión de ojo de pájaro para asistencia en el aparcamiento

BMW tiene un sistema un poco más evolucionado, pues a parte del sistema de Nissan, posee una cámara trasera con sensores que miden las distancias con los obstáculos. En la imagen que ofrece proyecta una serie de líneas que ayudan en la orientación del vehículo durante la maniobra de aparcamiento. Estas imágenes son enriquecidas con marcas 3D que se basan en un código de color para marcar la distancia con los obstáculos. Además de todo esto, el sistema proporciona al conductor un mapa del vehículo y los obstáculos que le rodean en todo momento.



Figura 1.8: Cámara trasera de BMW para asistencia en el aparcamiento

Hoy en día uno de los campos en continua evolución es el de los coches autónomos, pues cada día hay más empresas interesadas en ello. Pero este tema no es tan actual, ya que en 1950 ya se trataba de crear coches autónomos. Y desde entonces se han conseguido continuos avances en su mayor medida destinados a aportar a los vehículos sistemas para navegar de forma autónoma.

CAPÍTULO 1. INTRODUCCIÓN

Los coches autónomos disponen de un radar en la parte frontal para detectar vehículos próximos. Este sensor se emplea para mantener la distancia de seguridad con el vehículo que se encuentra delante y para posibles frenadas de emergencia si fueran necesarias. En las ruedas traseras tienen un sensor ultrasonido capaz de detectar obstáculos muy cercanos al vehículo, permitiendo con ello realizar maniobras muy precisas. Este sensor es muy práctico en el aparcamiento autónomo, pues ofrece información muy precisa en todo momento acerca de los obstáculos que se encuentren en la parte trasera. En la parte superior del vehículo se dispone de una antena GPS, una cámara Lidar y una cámara frontal. La antena GPS permite que el vehículo pueda localizarse con gran precisión en exteriores. La cámara Lidar (Light detection and ranging) realiza un barrido continuo de 360° para construir un mapa 3D de los alrededores del vehículo. La cámara frontal analiza todo lo que el vehículo tenga en frente, como por ejemplo señales de tráfico, peatones, otros vehículos, etc. A parte de todo lo comentado por supuesto contienen un ordenador a bordo que analiza la información captada por todos los sensores y en función a ella toma las decisiones que correspondan. Todos estos sensores pueden verse en la Figura 1.9.

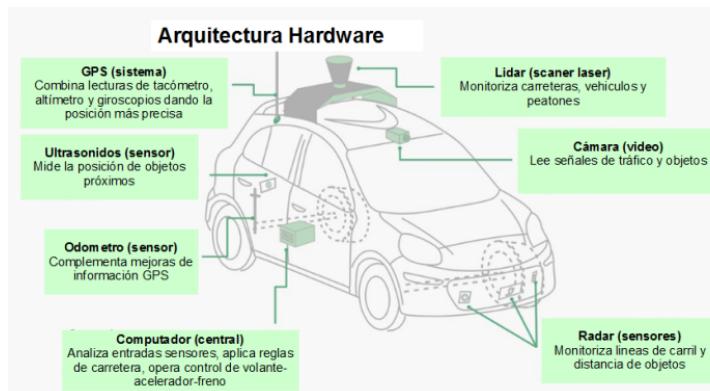


Figura 1.9: Sensores en vehículos autónomos

Si hablamos de conducción autónoma no nos podemos olvidar de mencionar al DARPA Grand Challenge. Se trata de una competición donde vehículos autónomos tratan de completar un circuito en el desierto en un tiempo determinado. En ella puede inscribirse cualquier entidad ya sea privada o pública. De hecho se pueden encontrar participando desde universidades hasta empresas privadas. Esta competición fue creada por la agencia de investigación DARPA con el objetivo de incentivar la creación y el desarrollo de vehículos autónomos capaces de llevar a cabo misiones preplanificadas. Con ello se pretende poder emplear esta tecnología en misiones de exploración así como para

CAPÍTULO 1. INTRODUCCIÓN

aplicaciones militares.

La agencia DARPA también organiza la competición DARPA Urban Challenge. Esta competición tiene la misma dinámica que la mencionada anteriormente pero se desarrolla en un entorno urbano, el cual consta de 96km los cuales deben completarse en menos de 6 horas. Durante el trayecto los vehículos tienen que interactuar con otros vehículos autónomos o vehículos ocupados por conductores profesionales. En este caso la competición es más compleja que la comentada anteriormente, pues durante el circuito si tenemos señales de tráfico, las cuales deben respetarse. Con esto se pretende desarrollar vehículos que se puedan emplear en la vida cotidiana.



Figura 1.10: Vehículo autónomo participante en el *DARPA Grand Challenge*

A pesar de todas las empresas implicadas en este desarrollo de vehículos autónomos la más famosa en este campo es Google. La división de coches autónomos de Google, conocida como Waymo, lleva varios años probando sus vehículos en situaciones reales. La única condición que tienen es que haya un conductor capaz de intervenir en caso de ser necesario ante una emergencia o posibles percances. Estos vehículos ofrecen continuamente información acerca de todo lo que les rodea, ya sea acerca de los peatones que hay, de los vehículos que les rodean, de las señales de tráfico, etc. Pues todo esto es por supuesto necesario para poder realizar una conducción totalmente autónoma. El coche de Google además posee un sensor Lidar capaz de alcanzar una distancia máxima de 200 metros.



Figura 1.11: Vehículo autónomo de Google

Otra de las empresas más importantes en el desarrollo de vehículos autónomos es Tesla. Actualmente Tesla está comercializando el modelo S ,el modelo 3, el modelo X y el modelo Y, los cuales disponen de un piloto automático para activar la conducción autónoma, pero necesita una supervisión activa del conductor por si necesitará tomar el control en cualquier emergencia. A pesar de esto los nuevos vehículos Tesla tienen el hardware que será necesario en el futuro para una conducción completamente automática en la mayoría de situaciones. Pero la ley aún prohíbe dicha conducción autónoma sin supervisión del conductor. Además de esto poseen un asistente de aparcamiento, la posibilidad de aparcar automáticamente, un radar de largo alcance capaz de detectar peatones y obstáculos con una gran antelación, un sonar ultrasónico envolvente que monitoriza el coche desde todos los ángulos, cámaras que detectan señales de tráfico, peatones y otros vehículos, y un sistema de navegación GPS.



Figura 1.12: Vehículo autónomo de Tesla

1.2. Las cámaras como sensor de tráfico

En comparación con otras disciplinas, la aplicación de la visión computacional en el análisis del tráfico es un campo de estudio muy joven. Las primeras aplicaciones de este tipo fueron presentadas por Onoe M., Ohba K. [5] y Hilbert [6]. Pero hasta la década de los 80 no hubo una actividad constante de publicaciones. En este periodo ya podemos encontrar referencias a aplicaciones basadas en visión artificial y orientadas al análisis del tráfico de vehículos. Por ejemplo Hoose, N [7] en 1989 presentó una técnica para calcular de manera automática la longitud de la cola formada por vehículos parados o circulando a una velocidad muy reducida. Ese año J.M Blosseville, C. Krafft, F. Lenior, V. Motyka, S. Beucher [8] presentaron el sistema TITAN, el cual es capaz de analizar escenas de entre 100 hasta 300 metros de longitud, día y noche. Con ello extrae parámetros tales como el flujo de vehículos, la velocidad y la longitud de la cola en atascos, todo esto con una velocidad de procesamiento de 4 imágenes por segundo. Las características más importantes de este trabajo son las luces de noche y los techos, capós y sombras frontales de día.

En 1989 Versavel, J. ;Lemaire, F. ; Van der Stede, D. [9] también presentó el sistema CCATS, el cual obtiene parámetros del tráfico como la ocupación de los carriles, el número de vehículos y la velocidad media de estos.

Además puede detectar anomalías en el flujo de vehículos como atascos o largas colas y alertar acerca de ello. Para mejorar el tráfico este sistema fue instalado en Bélgica.

Hasta finales de los 80 los trabajos se centraron en un macro análisis del tráfico para

CAPÍTULO 1. INTRODUCCIÓN

extraer datos relativamente simples. A partir de los 90 esto va a cambiar pues aumentará la investigación en este sector y se empezarán a realizar análisis más detallados de las escenas basándose en las características visuales de los vehículos. Es entonces cuando se empieza a usar plantillas 3D para la clasificación y seguimiento de vehículos. Baker, K.D Sullivan, G.D. [10] se dedicaron a analizar la viabilidad del seguimiento del tráfico basándose en modelos 3D. Para este seguimiento de tráfico han empleado reconocimiento y estimación de posición en secuencias de imágenes. En esa época el reconocimiento y estimación de posición estaba pensado para una sola imagen , por lo que fueron pioneros al aplicarlo en secuencias de imágenes. En su trabajo disponían de información a priori acerca de la escena 3D, la cual obtenían mediante una cámara calibrada. A pesar de no funcionar en tiempo real este trabajo fue el que abrió las puertas hacia nuevos estudios relacionados con el seguimiento de vehículos.

Koller, D., Weber, J., Malik, J. [2] presentaron un trabajo en el cual describían un sistema basado en el seguimiento de vehículos haciendo especial importancia en las occlusiones, uno de los grandes problemas en los sistemas de análisis de tráfico. En la Figura 1.13 podemos ver un ejemplo de su trabajo.

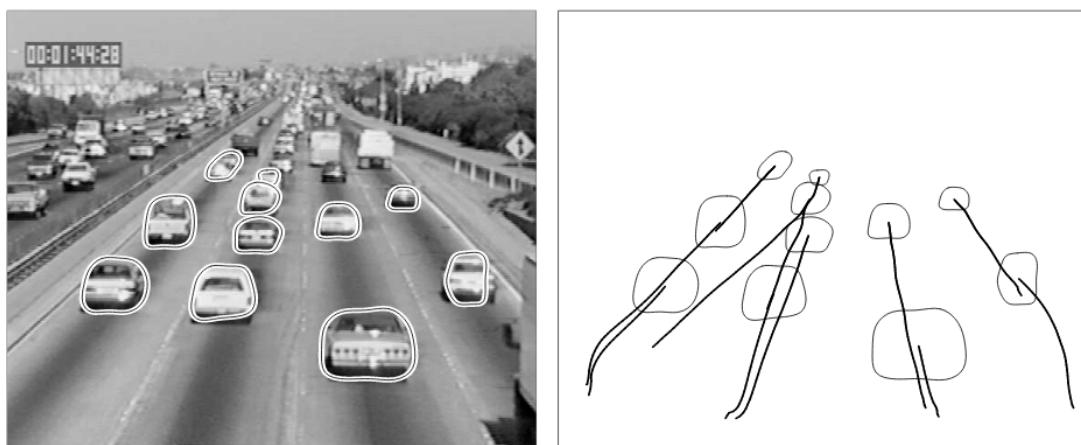


Figura 1.13: Ejemplo de detección y seguimiento de vehículos de [2]

G.D. Sullivan, K.D. Baker, A.D. Worral, C.I. Attwood, P.M. Remagnino [11] publicaron un sistema para la clasificación de vehículos empleando una combinación de plantillas en 1-D y 2-D. Las plantillas 1-D se generaban sobre modelos de vehículos y las 2-D se empleaban para la verificación y aceptación de las hipótesis generadas en la primera fase del algoritmo de clasificación. Coifman et al. [12] presentaron un algoritmo capaz de seguir un conjunto de características visuales de los vehículos (en lugar del vehículo completo)

con lo cual pretende ser más robusto ante oclusiones.

Las técnicas que se han ido publicando en las últimas décadas se presentan en dos escenarios (tráfico urbano y tráfico en autopistas). A continuación se explicará las situaciones que se encuentran en ambos escenarios.

1.2.1. Visión computacional en tráfico urbano

En el análisis de tráfico urbano se pretende aplicar las técnicas de visión computacional en ciudades o entornos urbanos en general. En este caso la visión computacional se emplea para asegurar la aplicación de las normas de tráfico, para obtener información acerca del flujo de tráfico o para detectar posibles incidencias. El entorno urbano es un entorno muy complejo, pues existen demasiadas variables involucradas. Podemos tener peatones, ciclistas, edificios, señales de tráfico, árboles, otros vehículos, etc. Esto hace que sea más probable tener oclusiones en la imagen.

Una aplicación muy habitual en entornos urbanos es la detección de conductores que no respetan los semáforos en rojo. Para ello se hace uso de cámaras que se encuentran conectadas a las luces del semáforo, y cada vez que éste se pone en rojo las cámaras se activan. Todo vehículo que sobrepasa la línea blanca una vez que el semáforo está en rojo quedará fotografiado. Pero esto no siempre se tiene totalmente en cuenta, pues en muchas legislaciones es necesaria una prueba irrefutable de que el vehículo se ha saltado el semáforo. Esto se debe a que en muchos casos cuando un semáforo está en ambar los coches aceleran, corriendo el riesgo de pasar justo cuando el semáforo se ponga en rojo. Con el fin de evitar estos problemas, los sistemas han ido avanzando. Actualmente se graban toda la escena para poder asegurarse de si existe una infracción o no. La cámara se sitúa antes del semáforo para poder ver claramente a los vehículos infractores. Además se les ha incorporado un mecanismo para medir la velocidad de los vehículos que pasan y así poder ver si tienen alta probabilidad de saltarse los semáforos. Si superan cierta velocidad se considera que tienen una alta probabilidad de cometer una infracción y es entonces cuando la cámara se activa para grabarlo.



Figura 1.14: Control de semáforos en rojo

En el tráfico interurbano una aplicación muy común es la detección de infracciones. Los centros de monitorización del tráfico disponen de muchas cámaras en las carreteras, pero el personal dedicado a revisar las imágenes que obtienen es bastante reducido, por lo que surge la necesidad de automatizar este proceso. A esta automatización de la detección de incidencias se le llama AID y gracias a ella los trabajadores dedicados a la monitorización del tráfico solo tienen que revisar las imágenes en las cuales las cámaras han detectado una incidencia. El hecho de automatizar este proceso hace que el tiempo de intervención se vea reducido, pues son capaces de detectar las incidencias en menor tiempo, minimizando el tiempo de respuesta de los servicios sanitarios. Con esta información pueden actualizarse los paneles informativos, para avisar al resto de vehículos de la existencia de un accidente.



Figura 1.15: Monitorización del tráfico urbano

CAPÍTULO 1. INTRODUCCIÓN

Vermeulen [13] para detectar incidentes combina la información visual con información obtenida por una cámara térmica. Con esto se consigue un sistema muy robusto ante las diferentes condiciones meteorológicas. Este sistema fue instalado en el puente de *Rion* en Grecia. Chang et al [14] presentó un sistema basado en la detección de matrículas para la identificación de incidencias. En este caso emplea la información del tiempo de viaje del vehículo para inferir incidencias en su trayectoria.

Otro tema en creciente desarrollo es el análisis del comportamiento de los vehículos en rotundas o bifurcaciones. A la hora de realizar modificaciones en las infraestructuras viales se realizan estudios acerca del flujo de vehículos que circulan por las redes de transporte, para ello es muy útil conocer el comportamiento de los vehículos. Nateghinia and Moradi [15] realizó un sistema capaz de seguir los vehículos en intersecciones y resolver el problema de las occlusiones. Para realizar las detecciones modelan cada píxel como una distribución gaussiana y lo combinan con el modelado dinámico de textura. Shirazi and Morris [16] presentó un sistema un poco más evolucionado, pues a parte de seguir y contar los vehículos en cada intersección, era capaz de estimar la capacidad y el tiempo de espera en la intersección y reconstruir la trayectoria de los vehículos.

Finalmente, no hay que olvidarse de los túneles, los cuales juegan un papel importante dentro del tráfico urbano. De hecho en la M-30 en Madrid tenemos una longitud de 43 km de túneles, con el fin de reducir el flujo de tráfico sobre la superficie. Debido al gran uso de los túneles y la dificultad que existe para acceder a ellos en caso de accidentes, es necesario monitorizarlos. El uso de cámaras en los túneles hace que se pueda identificar la existencia de incidencias dentro de ellos, para actuar en el menor tiempo posible. En este caso una actuación rápida es esencial para evitar segundos accidentes o catástrofes en caso de incendios.



Figura 1.16: Sistema de detección de incidencias en túnel

1.2.2. Monitorización de autopistas basada en visión

En este caso el análisis del tráfico se realiza en autopistas. Con ello se pretende obtener información acerca de la velocidad de los vehículos, el flujo de tráfico, la existencia de incidencias, etc. Este entorno es más sencillo que el comentado en el Apartado 1.2.1, ya que no hay que controlar tantas variables (peatones, edificios , etc), únicamente tendremos vehículos de diferentes tipos.



Figura 1.17: Sistema de detección de incidencias en autopistas

La mayoría de los sistemas mencionados en el caso de entornos urbanos se emplean en las autopistas, ya que las incidencias que ocurren son similares. La principal diferencia entre ambos escenarios es la velocidad de los vehículos. En el tráfico urbano el límite de velocidad es bajo, pero en las autopistas es mucho más elevado y depende de cada país. En España el límite está en 120km/h pero en Alemania la velocidad no está limitada. Esto se debe tener en cuenta a la hora de realizar el sistema de monitorización.

1.3. Sistemas comerciales basados en visión

Antiguamente la visión computacional solo se desarrollaba en el ámbito de la investigación pero actualmente se ha visto extendida hasta un ámbito comercial. Actualmente en el mercado hay numerosos sistemas basados en visión. Gracias a su reducido coste y su gran fiabilidad los sistemas basados en visión se han convertido en una alternativa perfecta a los sistemas clásicos de detección.

Desde los años 80 el empleo de la tecnología en los sistemas inteligentes de transporte ha ido creciendo. Esto se debe a que es un sector muy importante en los países en vías de desarrollo y en los países desarrollados. Algunas de las empresas que desarrollan este tipo de sistemas son: Flir (EEUU), Siemens (Alemania), Indra (España), Kapsch (Austria), Q-Free (Noruega), Thales (Francia), Sigtec(Austria).

Siemens es uno de los mayores proveedores de sistemas de control de tráfico a nivel mundial. Su familia de sistemas *SitTraffic* ofrece un amplio abanico de funcionalidades.

FLIR es una empresa muy activa en este sector, pues ofrece sistemas con diversas funciones (detección automática de atascos, presencia de vehículos en las intersecciones, detección de peatones , etc). Su sistema combina cámaras visuales y cámaras térmicas que le permiten regular las luces de tráfico. La serie de productos *TrafiCam* realiza la detección y seguimiento de vehículos en intersecciones de forma automática; y permite posicionar y verificar con exactitud las zonas de detección de presencia de vehículos. Además este sistema mide la longitud de las colas que se forman en los atascos.



Figura 1.18: Sistema *TraffiCam* de FLIR

La empresa Morpho ha desarrollado un sistema basado en visión que pretende sustituir a los radares tradicionales. Se trata de su sistema *Mesta Fusion*, el cual es capaz de detectar coches que circulen hasta a 300 km/h, puede controlar ocho carriles al mismo tiempo, supervisar 32 vehículos a la vez y discriminar entre ellos según sea un turismo, una moto, una furgoneta o un camión o autobús. Además es capaz de identificar casi cualquier acción indebida. Todo eso lo hace un sistema perfecto en temas de monitorización de tráfico. Este sistema ya está operativo en Francia y Dubai.

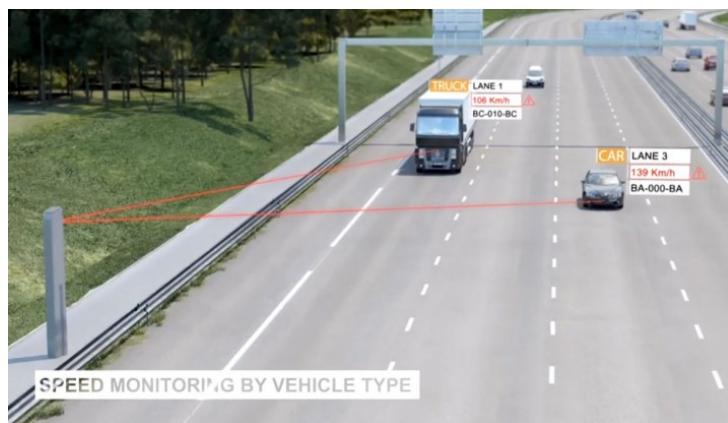


Figura 1.19: Sistema *Mesta Fusion* de Morpho

1.4. Redes Neuronales

Las Redes Neuronales Artificiales (RNA) son un modelo matemático que pretende asemejarse al comportamiento biológico de las neuronas. Su objetivo principal es poder aprender al igual que lo hacen los seres humanos, para ello siguen una estructura jerárquica similar.

Las neuronas biológicas poseen un cuerpo celular que contiene un núcleo, una serie de ramas denominadas dendritas y un axón. Las dendritas transmiten la información de las células vecinas mediante sinapsis al cuerpo celular, y éste genera impulsos nerviosos para transmitir la información a otras neuronas a través del axón.

En las redes neuronales la sinapsis y las dendritas son las entradas, y el proceso interno es el cuerpo neuronal. Cada entrada tiene asociado un peso. Estos pesos se multiplican por las propias entradas. Normalmente se suma el resultados de estos productos y se genera un resultado que se transmite por la salida o axón. En la Figura 1.20 se puede ver una comparativa de las neuronas biológicas y artificiales.

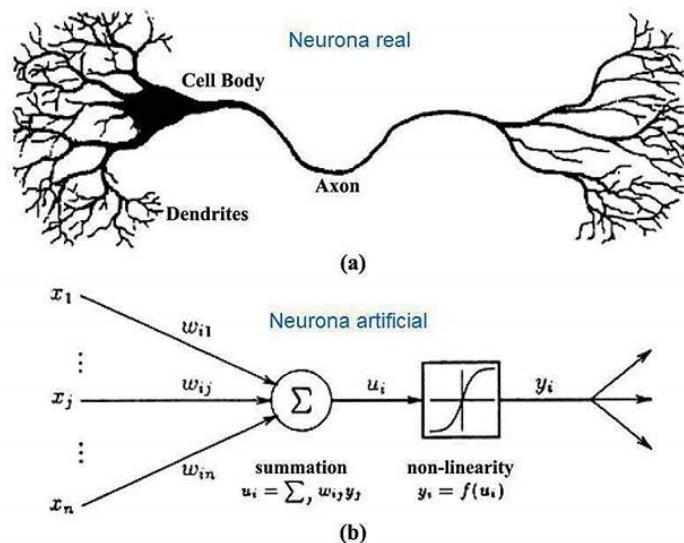


Figura 1.20: Neurona Biológica y Artificial

Actualmente las RNA han tomado mucha relevancia en el mundo de la Visión Artificial (VA) debido a sus grandes resultados. Pues se ha visto que son capaces de aprender gran cantidad de características.

1.5. Objetivos

Este Trabajo de Fin de Máster (TFM) se basa en la monitorización visual de vehículos mediante el uso de una cámara. El objetivo principal de este proyecto es obtener un sistema capaz de detectar, clasificar y realizar un seguimiento de vehículos en tiempo real. Para ello se ha partido de un sistema inicial realizado por Redouane Kachach [17] (*Traffic-Monitor* [1]) en su tesis doctoral. Dicho sistema se basa en técnicas clásicas. El objetivo es mejorar sus resultados con el uso de técnicas más actuales como el *Deep Learning*.

La detección se basa en identificar la posición de los vehículos en la imagen. Al hacer uso de *Deep Learning* la detección y clasificación de vehículos van de la mano. En este trabajo se evaluarán diferentes librerías relacionadas con el *Deep Learning*, tales como *TensorFlow*, *Keras* y *Darknet*, con el objetivo de emplear la que mejores resultados ofrezca.

La fase de clasificación debe categorizar los vehículos en función de 7 clases: Motocicletas, Coches, Furgonetas, Autobuses, Camiones Pequeños, Camiones y Camiones Cisterna. Esta es una mejora respecto al sistema del que se parte [17], pues en ese caso la clasificación se realizaba entre 5 tipos de vehículos. No existía la diferenciación entre los tres tipos de camiones, sino que se trataban como una única categoría.

En cuanto al seguimiento, el objetivo es tener la capacidad de realizar el tracking de cada vehículo durante todo el recorrido.

Para conseguir todos los objetivos mencionados es necesario recopilar una amplia base de datos que nos ofrezca datos de vehículos en diferentes condiciones meteorológicas, y con menor o mayor calidad de imagen, para tratar de conseguir un sistema lo más robusto posible.

Además el sistema debe cumplir los siguientes requisitos:

1. El sistema tiene que emplear como sensor una cámara.
2. El sistema debe funcionar en tiempo real.
3. Las bases de datos que se han empleado durante el desarrollo del proyecto son de la parte trasera del vehículo, por lo que todos los videos que se evalúen tendrán esta característica. Para funcionar con otra orientación de la cámara tendríamos que emplear una base de datos mayor con diferentes orientaciones de los vehículos. Esto no se ha podido realizar debido a la falta de tiempo.

4. Los algoritmos están preparados para funcionar de día.

Hay que decir que en la tesis de Redouane Kachach [17] no se tuvieron en cuenta secuencias de video con diferentes tipos de condiciones meteorológicas, pero en este caso se pretende que el sistema sea capaz de funcionar con diferentes condiciones. No obstante el sistema no se ha desarrollado para que sea capaz de funcionar por la noche.

1.6. Estructura de la Memoria

En el Capítulo 1 se comenta como la visión ha sido empleada en el campo de la monitorización del tráfico, y se realiza un breve repaso de la historia y evolución de esta disciplina.

El Capítulo 2 hace un repaso acerca de los trabajos científicos relacionados con la detección, clasificación y seguimiento de vehículos, los cuales sirven de referencia para nuestro estudio.

En el Capítulo 3 se da una explicación general de las herramientas empleadas a la hora de desarrollar el proyecto.

En el Capítulo 4 se explica en detalle como se ha desarrollado todo el sistema de monitorización. Haciendo hincapié en cada parte de la monitorización: clasificación, detección y seguimiento de vehículos.

El Capítulo 5 engloba todas las pruebas realizadas a lo largo del Trabajo de Fin de Máster (TFM). Así como la evaluación que se ha realizado para saber como de bien funcionaba el sistema realmente.

Finalmente el Capítulo 6 recaba todas las conclusiones a las que se ha llegado durante el desarrollo del trabajo, así como posibles mejoras futuras.

Capítulo 2

Estado del arte

En este capítulo se revisará la bibliografía más relevante relacionada con la monitorización del tráfico, obteniendo así información acerca de las diferentes técnicas propuestas.

Nos centraremos en las técnicas propuestas para la detección, clasificación y seguimiento de vehículos en las cuales se haga uso únicamente de una cámara como sensor.

En los últimos años el precio de las cámaras se ha visto reducido y la potencia de cómputo de nuestros dispositivos ha aumentado, favoreciendo al desarrollo de sistemas de análisis de tráfico. Debido a esto, se trata de un área en continuo desarrollo desde los años 90, en el cual se pretende dar una solución en tiempo real.

En la monitorización del tráfico hay que tener en cuenta tres puntos principales:

1. Detección de vehículos
2. Clasificación de vehículos
3. Seguimiento de vehículos

La detección consiste en localizar la posición de cada vehículo en la imagen. La clasificación trata de identificar a qué clase pertenece cada detección. Y el seguimiento consiste en asociar los vehículos en las sucesivas secuencias de video.

A continuación veremos las soluciones que plantean los diferentes autores para cada punto. Hay que decir que en muchas ocasiones la detección y la clasificación se hacen conjuntamente. No obstante vamos a comentar por separado cada fase.

2.1. Detección de vehículos

La detección de vehículos trata de identificar donde se localizan los vehículos en los fotogramas. En la literatura hay numerosas técnicas que han sido planteadas para esta función, las cuales van a ser comentadas a continuación.

Una técnica muy empleada en este problema es la sustracción del fondo. Se trata de restar la imagen del fondo de las sucesivas imágenes, con el fin de quedarnos con los objetos que se encuentran en movimiento, que en este caso en concreto serán los vehículos. S.I. Arroyo, F. Safar and D. Oliva [18] se basan en la diferencia entre el fotograma actual y la imagen referencia (imagen de fondo). Esta imagen referencia debe ajustarse a las condiciones de luminosidad en el tiempo.

Otra técnica muy parecida a la sustracción de fondo es la diferencia absoluta (Sum of Absolute Differences (SAD)) entre dos secuencias. A.F. Granados and J.I. Marin .H [19] presentan una técnica basada en la diferencia absoluta (SAD) entre dos fotogramas consecutivos. Para ello aplican filtrado homomórfico con el fin de reducir el efecto de los cambios de iluminación. Tras esto realizan la diferencia absoluta (SAD), umbralizan y segmentan los objetos en movimiento. J. Portillo, G. Sánchez, J. Olivares and H. Pérez [20] también propusieron un método en el que se hacía uso de la SAD y complementariamente se realizaba un análisis de bordes en la región considerada en movimiento.

En la vida real tenemos situaciones más complejas, pues los cambios de iluminación y la aparición de objetos en la escena (por ejemplo ramas de los árboles, personas, etc) es muy probable. Por tanto, necesitamos un método que sea capaz de funcionar correctamente a pesar de estos problemas.

Mixture of Gaussian (MOG) es una técnica que aplicaron C.Stauffer and W.E. Grimson [21] por primera vez al problema de la sustracción de fondo. Esta técnica se basa en el empleo de varias gaussianas para modelar cada píxel, permitiendo modelar varios estados del fondo a un mismo píxel. Es decir, esta técnica permite modelar un píxel cuyo valor cambia constantemente debido a movimientos periódicos. La probabilidad actual de un píxel se calcula en función de los valores que ha tomado a lo largo de un tiempo X_1, \dots, X_t mediante la fórmula:

$$P(X_x) = \sum_{i=1}^k \omega_{i,t} * \eta(X_t, \mu_{i,t}, \Sigma_{i,t}) \quad (2.1)$$

Donde k es el número de gaussianas, η es una gaussiana de media $\mu_{i,t}$ y cuya matriz

de covarianza es $\Sigma_{i,t}$. El valor $\omega_{i,t}$, es una estimación de su peso y puede variar con el tiempo (hace referencia a la proporción que supone esta gaussiana en la decisión final). La función de densidad de probabilidad η tiene la siguiente forma:

$$\eta(X_t, \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(X_t - \mu_t)^T \Sigma^{-1} (X_t - \mu_t)} \quad (2.2)$$

El número de gaussianas k viene determinado por el número de *modos* posibles para el fondo, los cuales dependen del escenario. Este número afecta de forma directa al tiempo de procesamiento. En una implementación básica de MOG se asume que los canales (R,G,B) son independientes, obteniendo de esta forma una matriz de covarianza reducida:

$$\Sigma_{k,t} = \sigma_k^2 I \quad (2.3)$$

En esta versión de MOG la pertenencia de cada nuevo píxel al fondo es evaluada contra las distintas gaussianas que lo forman. Si el valor de un píxel está en un rango de 2.5 la desviación estándar de la gaussiana, se considera que pertenece a dicha gaussiana. El nuevo píxel será clasificado como fondo si pertenece a alguna de estas gaussianas y los parámetros $\mu_{k,t}, \Sigma_{k,t}$ y $\omega_{k,t}$ serán actualizados con el nuevo valor mediante las siguientes fórmulas:

$$\omega_{k,t} = (1 - \alpha)\omega_{k,t-1} + \alpha(M_{k,t}) \quad (2.4)$$

$$\mu_t = (1 - \rho)\mu_{t-1} + \rho(X_t) \quad (2.5)$$

$$\sigma^2 = (1 - \rho)\sigma_{t-1}^2 + \rho(X_t - \mu_t)^T (X_t - \mu_t) \quad (2.6)$$

Donde ρ se corresponde con:

$$\rho = \alpha \eta(X_t | \mu_k, \sigma_k) \quad (2.7)$$

$M_{k,t}$ vale 1 para la gaussiana que describe el píxel y 0 para el resto. El valor de μ y σ no cambia para el resto de distribuciones. α es el factor de aprendizaje.

Tras esto, los valores de $\omega_{k,t}$ se normalizan entre las k distribuciones que modelan al píxel. Si el píxel no pertenece a ninguna de las distribuciones, la distribución de menor peso será remplazada por otra con una varianza inicial grande, una media igual al valor del nuevo píxel y un peso inicial bajo.

Lo normal es que no todas las gaussianas sean empleadas para formar parte del modelo de un píxel, sino que se escoge un número T de gaussianas para formarlo. Debido a los

grandes cambios que puede sufrir el fondo no todas las gaussianas que formen parte del modelo representarán algún modo del fondo.

Para realizar todo este proceso se ordenarán las distribuciones en función del valor de ω/σ . Los componentes con un alto peso en la mezcla de gaussianas y poca desviación serán los primeros en ser utilizados a la hora de evaluar la pertenencia de un píxel al fondo.

La mezcla de gaussianas es muy utilizada en detecciones de fondo, esto se puede ver reflejado en la cantidad de trabajos que hacen uso de esta técnica. De hecho el trabajo del que se ha partido en este caso [17] hace uso de una versión mejorada de MOG propuesta por Zoran Zivkovic [22]. El avance de este método es que para cada píxel se puede adaptar el número de gaussianas que se van a emplear.

P. Barcellos, C. Bouvié, F.L. Escouto and J. Scharcanski [23] propusieron un método para la detección de vehículos que se basaba en una combinación de Gaussian mixture models (GMM) y Motion Energy Images (MEI) [24]. MEI es la imagen de energía de movimiento e indica en cuales píxeles se detectó movimiento a lo largo de una secuencia de fotogramas. Esta técnica se incluye para introducir información temporal del primer plano, ya que gracias a ello tendremos una máscara binaria con los píxeles del primer plano que han sufrido movimiento.

Xia et al. [25] combinaron GMM y Expectation Maximization (EM). EM comienza prediciendo los parámetros de las distribuciones y los usa para calcular las probabilidades de que cada objeto pertenezca a una clase. Esas probabilidades se usan para re-estimar los parámetros de las probabilidades hasta que consigamos que converjan.

Buch et al. [26] presentaron un sistema que combina puntos de interés 3D y Histogram of Oriented Gradients (HOG) para detectar el fondo presentando así 3DHOG.

Bjorn Johansson, Johan Wiklund, Per-Erik Forssén and Gösta Granlund [27] emplean el método estadístico de J.Wood [28] para identificar si un píxel pertenece al fondo o al primer plano. El método es una modificación del conocido método de sustracción de fondo de C.Stauffer and W.E. Grimson [21], con una regla de actualización algo diferente y un límite de regularización más bajo para las desviaciones estándar de Gauss. Principalmente lo que se hace es usar un modelo de mezcla de gaussianas en cada píxel para estimar la distribución de color a lo largo del tiempo, detectándose como píxeles de primer plano aquellos cuyo color sea improbable que pertenezca a la distribución. Además los píxeles de primer plano pueden ser clasificados como sombras. Si el color de un píxel se encuentra

en una región cilíndrica entre el negro (el origen) y cualquiera de los colores centrales de las gaussianas de fondo se considerará sombra.

Otra técnica empleada a la hora de detectar el fondo es hacer uso de los denominados *codebooks*. Originalmente esta clase de técnicas fueron empleadas en la clasificación de texto. A la hora de clasificar texto es muy importante disponer de un diccionario de palabras que se ajuste al contenido que tendrá dicho texto. Estas palabras forman lo que se llama *codebooks* o vocabulario. Este mismo planteamiento se puede emplear a la hora de clasificar imágenes, pero en lugar de palabras tendremos vectores de elementos que describen las características de dichas imágenes. En este caso se está planteando esta técnica para la detección del fondo pero por supuesto se puede usar a la hora de clasificar los propios vehículos. Al ser una técnica no paramétrica estas palabras no pueden asociarse con distribuciones gaussianas. Hay que decir que este planteamiento no es muy usado en la detección de fondo.

K. Kim, T.H. Chalidabhongse, D. Harwood and L. Davis [29] dicen que 6 palabras son suficientes para clasificar un píxel. Como palabra los autores han empleado un vector RGB $vi = (R_i, G_i, B_i)$ y una tupla $aux_i = [I_{\min}, I_{\max}, f_i, \lambda_i, p_i, q_i]$ donde:

- I_{\min}, I_{\max} : es el mínimo y máximo valor del brillo, para cada píxel asignado a una palabra.
- f_i : es la frecuencia con la cual aparece una palabra.
- λ_i : se trata del máximo intervalo en el cual no ha aparecido la palabra.
- p_i, q_i : es la primera y la última vez respectivamente que ha aparecido la palabra.

Gracias a λ_i podemos saber que valores pertenecen o no al fondo. Si se tiene un valor grande de λ_i quiere decir que el objeto en cuestión aparece cada mucho tiempo. Esto puede indicar que se trate de un objeto que ha aparecido de forma temporal y por tanto no pertenece al fondo. Para estimar el fondo se comparará cada píxel de la nueva imagen con la palabra cuya media está más cercana. Si el píxel pertenece a la palabra, querrá decir que pertenece al fondo y por tanto se actualizará la palabra en cuestión con esta nueva información. Si por el contrario, no perteneciera a la palabra se trataría de un objeto en movimiento.

En el planteamiento de M. Mazaheri and S. Mozaffari [30] la detección esta basada en *codebooks*.

Otro de los métodos empleados en la detección de fondo son los basados en la estimación no paramétrica de la función de densidad a partir de un número de muestras determinadas. Un caso muy popular de este método son los histogramas, los cuales representan la frecuencia de cada valor de forma gráfica a través de barras. Con ello se puede ver la distribución de los datos. Esto tiene un problema, y es que en función del rango de la cuantización el resultado puede variar. Para evitarlo, se hace uso de una función matemática llamada *núcleo*, que obtiene una representación más suave de los datos. Dado un conjunto de valores de entrada (X_1, X_2, \dots, X_n), los cuales vienen de una distribución desconocida f , su *núcleo* se calcula de la siguiente forma:

$$\hat{f}_h(x) = \frac{1}{n} \sum_{i=1}^n K_h(x - x_i) = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{x - x_i}{h}\right) \quad (2.8)$$

K es una función no negativa que integra a uno. La h es el *ancho de ventana* y siempre toma valores positivos. A. Elgammal, D. Harwood and L. Davis [31] han aplicado este método para la estimación del fondo. En concreto plantean como *núcleo* una gaussiana $N(0, \Sigma)$, donde Σ hace referencia al *ancho de ventana*. Por tanto la ecuación 2.8 quedará de la siguiente manera:

$$P_r(x_t) = \frac{1}{N} \sum_{i=1}^N \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(x_t - x_i)^T \Sigma^{-1} (x_t - x_i)} \quad (2.9)$$

Si asumimos que el ancho de ventana σ_j es distinto para cada canal porque los canales de color son independientes tenemos que Σ es:

$$\Sigma = \begin{bmatrix} \sigma_1^2 & 0 & 0 \\ 0 & \sigma_2^2 & 0 \\ 0 & 0 & \sigma_3^2 \end{bmatrix} \quad (2.10)$$

Quedando la función de densidad como:

$$P_r(x_t) = \frac{1}{N} \sum_{i=1}^N \prod_{j=1}^d \frac{1}{\sqrt{2\pi\sigma_j^2}} e^{-\frac{(x_{tj} - x_{ij})^2}{2\sigma_j^2}} \quad (2.11)$$

En función del valor de $P_r(x_t)$ podemos saber si el píxel pertenece a un objeto en movimiento o no. Si $P_r(x_t)$ se encuentra por debajo de un umbral fijado consideraremos que se trata de un píxel perteneciente a un objeto en movimiento.

Todas las técnicas que se han ido comentando a lo largo de esta sección basan la detección de vehículos en la detección del fondo. En función de este fondo pueden saber cuales son los objetos que se encuentran en movimiento. Actualmente ha tomado mucha popularidad el *Deep Learning*, gracias a lo cual podemos ser capaces de detectar objetos con grandes resultados, sin necesidad de tener que detectar el fondo.

Rigoberto Vizcay [32] se basa en redes Convolutional Neural Network (CNN) para la detección de vehículos y peatones. Una red neuronal convolucional es una red que presenta una o varias capas convolucionales.

Y. Abdullah, G. Mehmet, A. Iman and B. Erkan [4] plantean el uso de Region-based Convolutional Neural Network (R-CNN) y Faster R-CNN para la detección de vehículos.

R-CNN para realizar las detecciones realiza tres fases:

1. Se emplear el algoritmo Selective Search, el cual solo realiza las pruebas de detección a las regiones candidatas a tener una posible detección. Con ello se extraen aproximadamente 2000 regiones de la imagen (Region Proposal).
2. Se implementa una red neuronal Convolutional Neural Network (CNN) en la parte superior de cada región.
3. Se extrapola la salida de cada CNN y se ingresa en una Support Vector Machine (SVM) para clasificar la región. Además se realiza una regresión lineal para restringir el cuadro de la detección.

Estas fases pueden verse en la Figura 2.1.

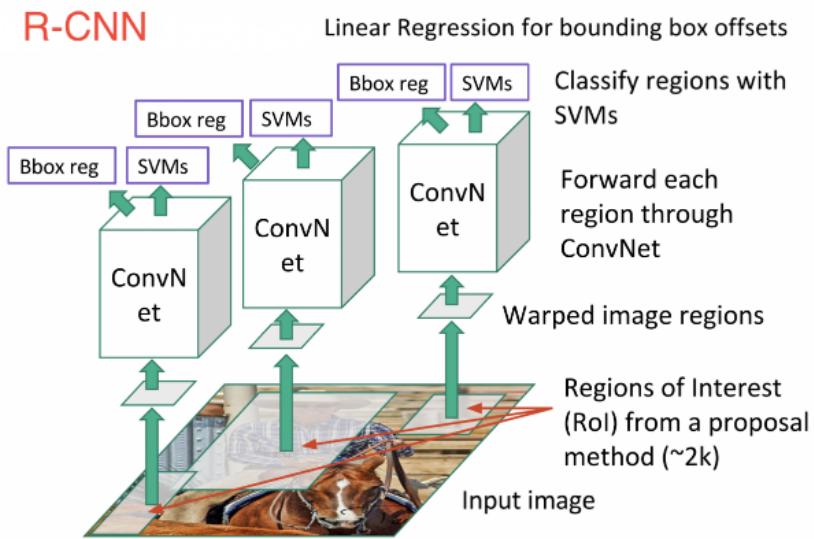


Figura 2.1: Fases de R-CNN

Faster R-CNN es una versión más rápida que R-CNN, en la que se modifican algunos aspectos para conseguirlo. En este método se incluye la técnica region proposal, la cual determina que regiones de la imagen tienen mayor probabilidad de contener objetos y por tanto cuales son las regiones que se introducirán en el clasificador. Esto optimiza mucho el trabajo pues evita introducir al clasificador regiones que no sean de interés.

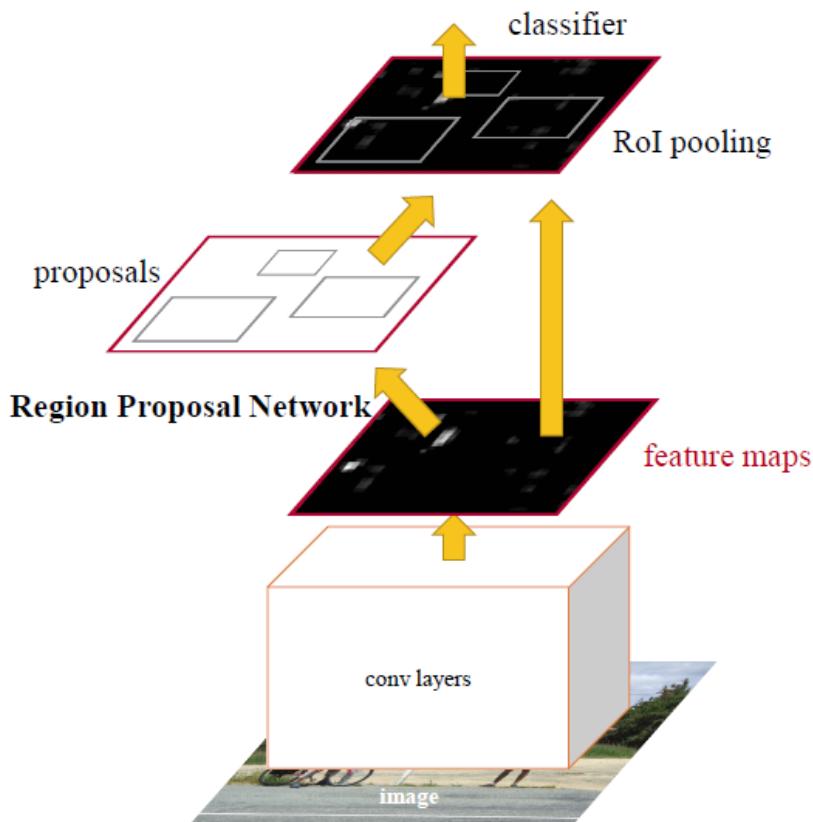


Figura 2.2: Fases de Faster R-CNN

Ignacio Arriola [33] hace también uso de Faster R-CNN.

2.2. Clasificación de vehículos

La clasificación de vehículos trata de determinar a qué clase pertenece cada objeto detectado. Analizando los trabajos que se centran en este problema se ha podido ver que son tres los métodos más empleados:

- Técnicas basadas en características.
- Técnicas basadas en modelos 3D.
- Técnicas basadas en *Deep Learning*

Las técnicas basadas en características se centran en extraer propiedades de la imagen que permitan poder clasificar cada objeto. En el caso de la clasificación de vehículos, tratarán de definir un conjunto de características, con las cuales pueda quedar bien

descrito el vehículo en cuestión. Para ello se necesitará mucha información, cuantas más características tengamos del objeto será más fácil de describir. En esta familia de técnicas se requiere una fase de entrenamiento previa donde el sistema sea capaz de aprender las diferentes características. Una vez tengamos el sistema entrenado le pasaremos los objetos detectados, para que estime a que clase pertenecen. Dicha clase será siempre la que más características tenga en común con el objeto que pretendemos clasificar.

Las características pueden ser visuales o geométricas (longitud, área, relación entre el alto y el ancho, etc). En el caso de las características geométricas Shih-Hao Yu et al. [34] presentaron un sistema que era capaz de distinguir entre tres clases: camiones, autobuses y coches. Para ello se basaba en dos características (tamaño y linealidad). La linealidad se empleaba para distinguir entre autobuses y camiones.

Jin-Cyuan Lai, Shih-Shinh Huang and Chien-Cheng Tseng [35] definieron las mismas clases que en [34](coches, camiones y autobuses). Ellos se basaban en la compacidad y la proporción entre el ancho y el alto de los cuadros detectados. Además filtran los blobs en función de estas restricciones con el fin de eliminar ruido y blobs muy pequeños.

H. Asaidi , A. Aarab and M. Bellouki [36] definieron tres categorías: furgonetas, coches y camiones. Para definir cada vehículo determinaron 7 características geométricas. En función a estas 7 características calculaban la distancia euclídea entre el vehículo detectado y las categorías definidas. El vehículo era clasificado a la categoría con la que se obtenía menor distancia.

Las características visuales son más complejas que las geométricas, pues tratan de definir la forma y el aspecto de cada objeto. Para ello emplean descriptores.

Dos descriptores que tienen mucha relevancia son el descriptor Histogram of Oriented Gradients (HOG) y el descriptor Haar. C.P. Papageorgiou, M. Oren and T. Poggio [37] fueron quien introducieron los descriptores Haar para la detección de caras humanas. Navneet Dalal and Bill Triggs [38] propusieron el descriptor HOG para detectar peatones.

A partir del descriptor HOG hay alguna variante como por ejemplo el descriptor 3DHOG. Buch et al. [26] ya emplearon este descriptor para detectar el fondo , el cual combina puntos de interés 3D y HOG.

Estos descriptores son muy aplicados en el mundo de la clasificación de objetos gracias a los buenos resultados que se ha visto que obtienen. HOG suele mezclarse con algun otro clasificador, pues se ha visto que da muy buenos resultados.

Bailing Zhang, Yifan Zhou and Hao PanTammam Tillo [39] presentaron un método

basado en Kernel Auto Associator (KAA) para la clasificación de vehículos. A la hora de detectar los vehículos empleaban HOG combinado con clasificadores Support Vector Machine (SVM) en cascada. Para la clasificación también hicieron uso de (Edge Orientation Histogram (EOH) como característica discriminante.

Otra técnica que ha sido aplicada a la hora de clasificar vehículos es la técnica eigenfaces, la cual fue introducida por L. Sirovich and M. Kirby [40] para detectar y reconocer caras. Wei Wang, Yulong Shang, Jinzhi Guo and Zhiwei Qian [41] fueron unos de los autores que aplicaron la técnica de eigenfaces (se trata de construir con características un subespacio de vectores) para la clasificación de vehículos. En este trabajo en concreto tan solo clasificaban las detecciones como coches o no coches. Inicialmente es necesario entrenar al sistema con una base de datos. Tras este aprendizaje ya se podrán clasificar los vehículos. Para ello se captura la parte frontal del vehículo y se le aplican operaciones de post-procesado para emplearlas en la entrada del clasificador. Esta proyección de la parte frontal del vehículo es comparada con las proyecciones recopiladas para el entrenamiento. Si la comparación se encuentra por debajo de un umbral fijado se considerará que la clasificación se ha realizado correctamente.

Hasta ahora hemos hablado de sistemas que empleaban o técnicas basadas en características geométricas o visuales. Las características visuales ofrecen grandes ventajas frente a las geométricas, las cuales son muy susceptibles ante cambios de posición de la cámara o vehículos con geometrías similares. Por el contrario las características visuales son muy complejas y requieren de una fase de entrenamiento. Por ello lo ideal es hacer uso de ambas características, para así conseguir un sistema muy robusto. Dentro de este contexto tenemos a Zezhi Chen and Tim Ellis [3], los cuales presentan un sistema que emplea dos conjuntos de características. Uno de ellos Measurement Based Features (MBF) formado por hasta 13 características, entre las cuales tenemos el perímetro, el área, etc. Y otro conjunto llamado Intensity-based on a Pyramid of Histogram of Gradient Orientations (IPHOG), los cuales codifican la forma y la distribución de los objetos dentro de la imagen. Las características son introducidas en un clasificador SVM, el cual realiza un entrenamiento para aprender dichas características. En la Figura 2.3 podemos ver un ejemplo de detecciones de este sistema.

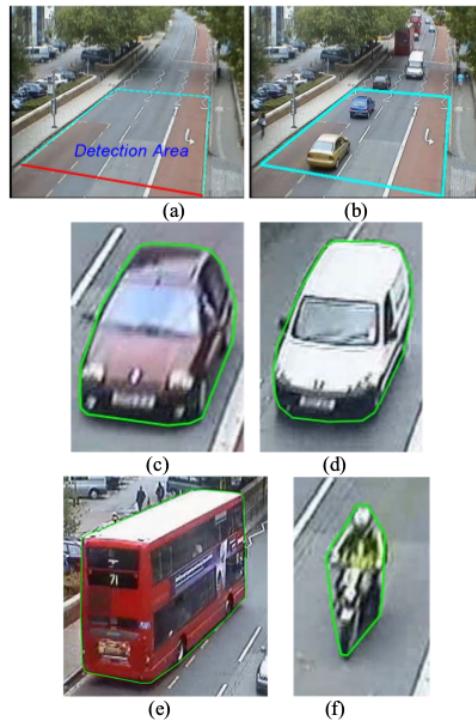


Figura 2.3: Detecciones del sistema de Zezhi Chen and Tim Ellis [3]

En las técnicas basadas en modelos 3D es necesario conocer los parámetros de la cámara que se emplee. En este caso tendremos una plantilla por cada clase y compararemos dicha plantilla con el objeto a clasificar, para ver cual es la clase que más se le ajusta. Wook-Sun Shin, Doo-Heon Song and Chang-Hun Lee [42] presenta un sistema basado en plantillas 3D que no necesita una cámara calibrada. Este sistema se basa en los puntos de fuga de los carriles para reconstruir la forma 3D de cada vehículo. Un ejemplo de esta reconstrucción se puede ver en la Figura 2.4

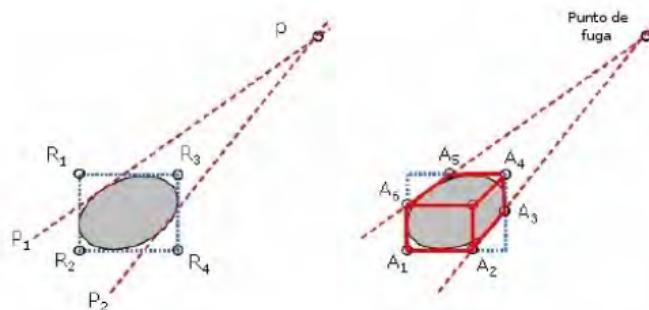


Figura 2.4: Reconstrucción 3D empleando puntos de fuga

Este sistema se basa en el algoritmo C4.5 de Quinlan [43] a la hora de clasificar. Este algoritmo emplea la construcción de árboles de decisión para clasificar y aprender las siluetas de los vehículos. Un ejemplo de esta técnica puede verse en la Figura 2.5.

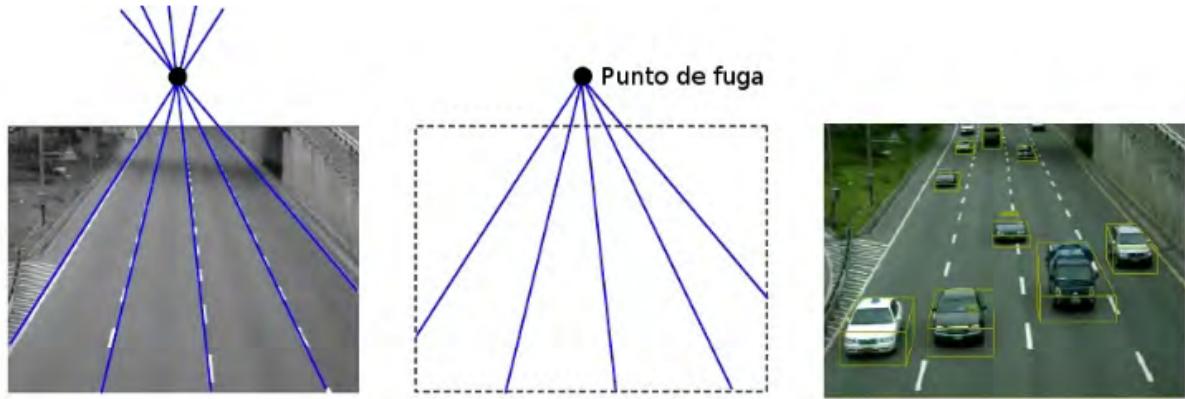


Figura 2.5: Clasificación de vehículos basada en modelos 3D construidos mediante el uso de puntos de fuga

Buch et al. [26] presentaron un sistema que combinaba plantillas 3D con HOG para clasificar los diferentes vehículos y los peatones. Este sistema se llama 3DHOG, y en él aplican descriptores HOG a plantillas 3D que definen los vehículos. Para cada categoría se define una plantilla 3D que permita definirla. En la Figura 2.6 se puede ver un ejemplo de dichas plantillas.

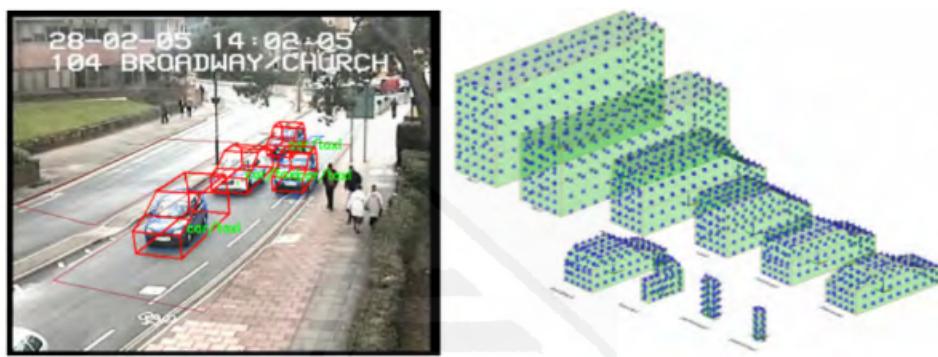


Figura 2.6: Plantillas 3DHOG para la clasificación de vehículos

En este sistema es necesario un previo entrenamiento. A la hora de realizar las clasificaciones se proyectan las plantillas 3D sobre los vehículos detectados y se les aplica transformaciones afines para ajustarlas al vehículo en cuestión. Tras esto se calculan

sus histograma 3DHOG y se comparan con los que se han aprendido en la fase de entrenamiento. Esta reconstrucción de los histogramas 3DHOG puede verse en la Figura 2.7.



Figura 2.7: Reconstrucción del histograma 3DHOG

Tal y como se ha comentado en la Sección 2.1 actualmente se ha extendido mucho el uso de *Deep Learning* tanto para la detección como para la clasificación de objetos. A.F. Granados y J.I. Marin.H [19] extraen los descriptores de Fourier de los vehículos detectados y los clasifican mediante una red neuronal. Dicha red neuronal consta de cuatro capas: una capa de entrada con una neurona por característica, dos capas ocultas con siete neuronas cada una y una capa de salida con una neurona clase.

2.3. Seguimiento de vehículos

El seguimiento es la localización de un objeto a medida que va moviéndose por la imagen. Para el ser humano es una tarea muy sencilla, pero para la visión artificial se trata de un tema complejo, pues pueden cambiar muchas características en el objeto a medida que avanza en la imagen. Tales como la forma, la iluminación, el tamaño, cambios en la perspectiva, occlusiones, movimiento de la cámara, etc.

Las técnicas más empleadas en el seguimiento de vehículos son:

- Seguimiento basado en regiones
- Seguimiento basado en características
- Seguimiento basado en modelos

Las técnicas basadas en las regiones se centran en el seguimiento de regiones conexas del objeto. Normalmente la propiedad que suele emplearse es el color. En los diferentes

artículos publicados se puede ver el uso de diferentes espacios de color como RGB, el espacio CEI lab o CEI LUV, el espacio HSV, etc. Lo más común es el uso de histogramas de color que nos permiten representar las regiones. Estas técnicas fueron introducidas por D. Comaniciu, V. Ramesh and P. Meer [44]. El seguimiento se basa en la comparación de los histogramas de las nuevas imágenes con el histograma de las regiones de interés calculadas en imágenes previas. Para ver el parecido entre los histogramas se emplea una medida similar a la distancia de Bhattacharyya y mean-shift para optimizar la selección del candidato.

Stefan Duffner and Christophe Garcia [45] presentó un algoritmo llamado *PixelTrack*, el cual combina la transformada de Hough con un modelo genérico de detección de fondo. Para ello necesita inicializar una ventana sobre el objeto. Gracias a esta técnica son capaces de seguir objetos en tiempo real con fondo cambiante, occlusiones y condiciones desfavorables. Lili Huang and M. Barth [41] plantean un algoritmo para llevar a cabo el seguimiento de vehículos y la resolución de occlusiones. En este algoritmo emplean un modelo de color basado en mean-shift para identificar a qué vehículo pertenece cada parche de 3x3 píxeles cuando existe una occlusión.

El seguimiento basado en características, tal y como su nombre indica se centra en seguir los objetos en función a las características que se crean oportunas. Cada autor hace uso de unas características. Entre estas características tenemos puntos característicos como esquinas, el perímetro del objeto, sus dimensiones, etc. Si lo comparamos con el seguimiento basado en regiones podemos ver que es más robusto, pues en el caso de las regiones trataban de seguir el objeto en función del color o la textura, lo cual es muy susceptible ante el cambio de iluminación. Entre las técnicas más empleadas en este ámbito podemos encontrar Scale-Invariant Feature Transform (SIFT)(D.G. Lowe [46]) y Kanade–Lucas–Tomasi (KLT) (J. Shi and C. Tomasi [47]). Otra técnica muy empleada en la literatura es HOG(Dalal and Bill Triggs [38]), la cual en muchas ocasiones ha sido combinada con SVM, tal y como se ha comentado en la sección anterior, Zezhi Chen and Tim Ellis [3] ya hicieron uso de ambos métodos.

El seguimiento basado en modelos trata de beneficiarse del conocimiento que tenemos de los objetos para poder realizar plantillas 2D y 3D para detectar objetos en la imagen y poder realizar su seguimiento. M.J. Leotta and J.L. Mundy [48] emplean esta técnica para detectar vehículos haciendo uso de una plantilla deformable que se ajusta para identificar diferentes formas de vehículos. En la literatura también se ha combinado esta técnica con

filtros de Kalman.

2.4. Bases de Datos para la detección de vehículos

La detección de vehículos pretende encontrar un vehículo en una imagen o en un vídeo y determinar qué tipo de vehículo es. Dado que queremos encontrar el vehículo en cuestión bajo diferentes circunstancias, es decir, en diferentes entornos y diferentes iluminaciones, necesitaremos entrenar el modelo con un conjunto de imágenes representativo. Por este motivo, a lo largo de los últimos años han surgido diferentes datasets con el fin de solucionar este problema.

2.4.1. GRAM Road-Traffic Monitoring

GRAM Road-Traffic Monitoring (GRAM-RTM) [49] [50] es un conjunto de datos para el seguimiento de vehículos en tiempo real. Consiste en 3 secuencias de video (Figura 2.8), grabadas bajo diferentes condiciones y con diferentes plataformas.

El primer video, llamado M-30 (*7520 frames*), se grabó en un día soleado con una cámara Nikon Coolpix L20, con una resolución de 800 x 480 a 30 FPS. La segunda secuencia, llamada M-30-HD (*9390 frames*), se grabó en una ubicación similar pero durante un día nublado, y con una cámara de alta resolución: una Nikon DX3100 a 1200 x 720 a 30 FPS. La tercera secuencia de video, llamada Urban1 (*23435 frames*), se grabó en una intersección concurrida con una cámara de tráfico de video vigilancia con una resolución de 600 x 360 a 25 FPS.

Todos los vehículos en el conjunto de datos GRAM-RTM fueron anotados manualmente. Este conjunto posee las siguientes categorías de clases: coches, camiones, furgonetas y camiones grandes. El número total de objetos diferentes en cada secuencia es: 256 para M-30, 235 para M-30-HD y 237 para Urban1. Todas las anotaciones en el conjunto GRAM-RTM se crearon en un formato XML compatible con PASCAL VOC.

2.4.2. BIT-Vehicle Dataset

El conjunto de datos BIT-Vehicle [51] contiene 9850 imágenes de vehículos. Contiene imágenes con tamaños de 1600 x 1200 y 1920 x 1080. Estas imágenes (Figura 2.9) fueron capturadas desde dos cámaras en diferentes momentos y lugares. Las imágenes contienen



Figura 2.8: Imágenes de ejemplo del dataset GRAM Road-Traffic Monitoring

cambios en las condiciones de iluminación, la escala, el color de la superficie de los vehículos y el punto de vista. Las partes superior o inferior de algunos vehículos no están incluidas en las imágenes debido a la demora en la captura y al tamaño del vehículo.

En cada imagen puede haber uno o dos vehículos y la ubicación de cada vehículo está previamente anotada. Además, el conjunto de datos se puede utilizar para evaluar el rendimiento de la detección de vehículos.

Los vehículos del conjunto de datos se dividen en seis categorías: bus, microbus, minifurgoneta, Sedan, SUV y camión. El número de vehículos por tipo de vehículo es de 558 para bus, 883 para microbus, 476 para minifurgoneta, 5922 para Sedan, 1392 para SUV, y 82 para camiones.

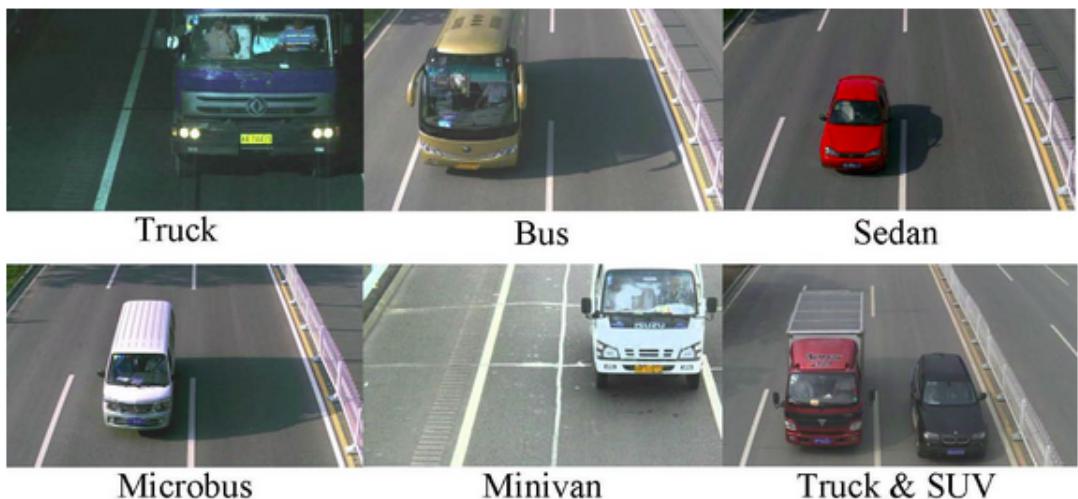


Figura 2.9: Imágenes de ejemplo del dataset BIT-Vehicle

2.4.3. CarND-Vehicle-Detection

CarND-Vehicle-Detection [52] es un proyecto dedicado a la detección de vehículos en vídeo, por ello crearon dos conjuntos de datos que se pueden encontrar en [53].

El *dataset* 1 incluye datos de conducción en Mountain View California y las ciudades vecinas durante el día. Contiene más de 65000 etiquetas en 9423 imágenes almacenadas por una cámara de investigación Point Grey que se ejecuta a una resolución máxima de 1920x1200 a 2Hz. El conjunto de datos fue anotado mediante CrowdAI utilizando una combinación entre aprendizaje automático y el trabajo de personas. Este conjunto de datos ocupa un total de 1.5 GB, y las clases que contiene son: coche, camión y peatón.

El *dataset* 2 es similar al conjunto de datos 1, pero contiene campos adicionales para la oclusión y una etiqueta adicional para los semáforos. El conjunto de datos fue anotado en su totalidad por humanos usando Autti y es un poco más grande que el anterior con 15000 imágenes (ocupa 3.3 GB). Este conjunto incluye las clases: coche, camión, peatón y farolas.



Figura 2.10: Imágenes de ejemplo del dataset 2 de CarND-Vehicle-Detection

Capítulo 3

Herramientas

En este capítulo se van a explicar las diferentes interfaces softwares que se van a emplear, así como los lenguajes de programación de los que haremos uso.

Como lenguajes de programación hay que decir que se ha hecho uso de *Python* y *C++*. Para el desarrollo del modelo de *Deep Learning* que se empleará para la detección, se han realizado pruebas con *TensorFlow*, *Keras* y *Darknet*. *OpenCV* se ha usado para todo lo relacionado con el tratamiento de imágenes.

La interfaz desarrollada, se ha hecho con *gtkmm*, herramienta que permite crear interfaces de usuario con múltiples funcionalidades.

Finalmente la evaluación de toda nuestra interfaz en general se ha realizado con *DeepLearningSuite*.

3.1. DeepLearningSuite

En este proyecto se ha hecho uso de la herramienta *DeepLearningSuite* [54] para evaluar los resultados de las redes neuronales entrenadas y de nuestra aplicación en general.

DeepLearningSuite es una herramienta que permite evaluar diferentes tipos de redes neuronales sobre un conjunto de datos. La idea es ofrecer una infraestructura genérica para evaluar los algoritmos de detección de objetos contra un conjunto de datos y calcular las estadísticas más comunes:

- Interseccion Over Union (IOU)
- Precisión
- Recall

La Intersección Over Union (IOU) en la detección de objetos, mide la precisión de un detector en un conjunto de datos en particular y sigue la siguiente fórmula:

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}} \quad (3.1)$$

Donde *Area of Overlap* es el área que pertenece a la intersección entre la predicción y la realidad, mientras que *Area of Union* es el área suma de la predicción y la realidad.

En la Figura 3.1 se puede ver un ejemplo visual del *ground-truth* y la predicción obtenida. En este caso el *Area of Overlap* es el área que incluye únicamente la intersección entre lo que hemos predecido y el *ground-truth*. Y el *Area of Union* es el área que se forma entre el *ground-truth* y la predicción.

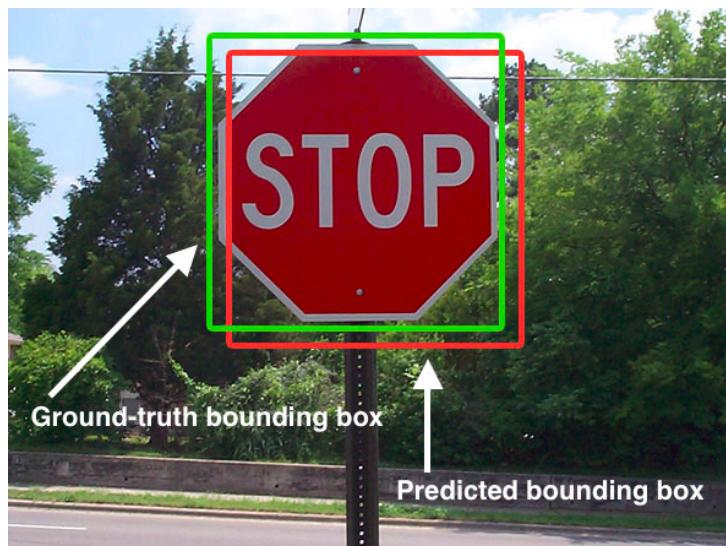


Figura 3.1: Ejemplo de IOU

En la Figura 3.2 se ve de forma muy ilustrativa la fórmula comentada anteriormente 3.1.

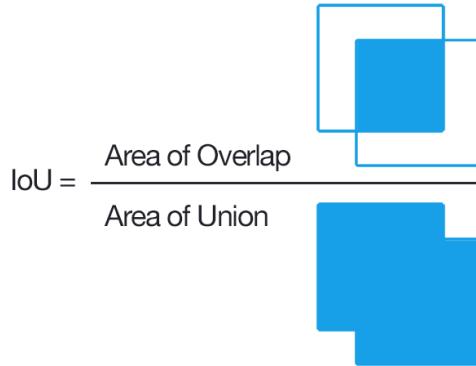


Figura 3.2: Fórmula IOU

Antes de explicar la precisión y el recall hay que aclarar algunos términos empleados a la hora de evaluar los resultados de las detecciones. A continuación se enumeran algunos términos necesarios de conocer para entender el recall y la precisión:

- True Positive (TP): son los verdaderos positivos, es decir aquellas detecciones que se han hecho correctamente.
- False Positive (FP): falsos positivos. Son aquellas detecciones que se han hecho pero son erróneas.
- False Negative (FN): falsos negativos. Es la proporción de casos positivos que la prueba detecta como negativos, es decir objetos que no se han detectado y deberían haberse detectado.
- True Negative (TN): verdadero negativo. Se refiere a los *boundingbox* que no deben detectarse en la imagen y no se han detectado. Este valor no se emplea en las métricas.

La precisión se trata del total de detecciones correctas entre la cantidad de detecciones obtenidas. La precisión que obtiene *DetectionSuite* es la promediada (mean Average Precision (mAP)), para aquellas predicciones que tienen un IOU mayor que un umbral (0.5).

$$Precision = \frac{TP}{TP + FP} \quad (3.2)$$

El recall es la cantidad de detecciones correctas entre la cantidad de detecciones reales, es decir las detecciones del *ground-truth*. Al igual que la precisión, se obtiene un

promediado (mean Average Recall (mAR)) de las detecciones que tienen un IOU superior a 0.5.

$$Precision = \frac{TP}{TP + FN} \quad (3.3)$$

Volviendo a DetectionSuite, es una herramienta que puede emplearse tanto en Linux como en MacOS. Permite evaluar modelos entrenados en *Tensorflow*, *Keras*, *Caffe* y *Darknet*. Los formatos de dataset que admite son: YOLO, COCO, *ImageNet Pascal VOC*, etc. A continuación se enumeran las entradas soportadas:

- WebCamera/ USB Camera
- Vídeos
- Streams from ROS
- Streams from ICE
- JdeRobot Recorder Logs

En este TFM en concreto se va a hacer uso de la herramienta *AutoEvaluator* de *DeepLearningSuite*, la cual es capaz de evaluar múltiples redes en un solo conjunto de datos o múltiples conjuntos de datos en una sola ejecución. Todo lo que necesita es un archivo de configuración que contenga detalles sobre los conjuntos de datos y las redes. Los resultados se escriben en archivos CSV en el directorio de salida especificado.

3.2. TensorFlow

Para el desarrollo de la red neuronal que queremos aplicar en nuestro *Smart-Traffic-Sensor* se han probado diferentes plataformas de desarrollo de *Deep Learning*, entre ellas *TensorFlow*.

*TensorFlow*¹ es una plataforma de código abierto *end-to-end* para el *Machine Learning* que fue liberada bajo licencia de *Apache 2* a finales de 2015 y que está disponible en github [55]. Fue desarrollada por el equipo de investigación en *Machine Learning* “*Google Brain*” en *C++ y Python*, y es usado en multitud de productos y servicios de Google como *Gmail* o *Google Translation*. Google ofrece en su plataforma *Cloud* ejecutar *Tensorflow*

¹<https://www.tensorflow.org/>

en *Tensor Processing Unit* (TPU), un nuevo tipo de procesadores en *Cloud* optimizados para ejecutar Inteligencia Artificial (IA). *TensorFlow* está orientado a problemas de *Deep Learning* y nos permite entrenar y construir redes neuronales. El *Deep Learning* es un área del *Machine Learning* que trata de asemejarse al funcionamiento del sistema nervioso humano.

TensorFlow puede correr tanto en CPUs como en GPUs (haciendo uso de Compute Unified Device Architecture (CUDA)). Está disponible en *Linux* de 64 bits, *MacOS*, y plataformas móviles que incluyen *Android* e *iOS*. Actualmente es la librería más popular en *Deep Learning*.

Puede ejecutar de forma rápida y eficiente gráficos de flujo. Un gráfico de flujo está formado por operaciones matemáticas representadas sobre nudos, y cuya entrada y salida es un vector multidimensional o tensor de datos, por este motivo recibe el nombre de *TensorFlow*.

Las ventajas de este software se extienden a muchas disciplinas a parte de la tecnología. Se emplea en medicina en imágenes médicas para la detección de tumores por ejemplo, también se usa en la detección y combinación de estilos artísticos en la pintura, etc.

En este TFM se emplea la versión 1.12.0 de *Tensorflow*.

3.3. Keras

Al igual que se ha empleado *TensorFlow* como plataforma de desarrollo de la red neuronal, se ha hecho uso de *Keras*.

*Keras*² es un *framework* de alto nivel para el aprendizaje, escrito en *Python* y capaz de correr sobre los *frameworks* *TensorFlow*, CNTK, o *Theano*. Fue desarrollado con el objetivo de facilitar el proceso de experimentación en redes neuronales de forma rápida y eficiente. Puede correr tanto en CPU como en GPU. Permite crear de forma sencilla y rápida los modelos de las redes neuronales (a través de su facilidad de uso, su modularidad y su extensibilidad). Admite redes convolucionales y redes recurrentes, así como combinaciones de las dos.

Inicialmente fue desarrollada en el proyecto de investigación ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System). Su creador es François Chollet, ingeniero de *Google*. En 2017, el equipo de *TensorFlow* de *Google* decidió dar soporte a

²<https://keras.io/>

Keras en la biblioteca de core de TensorFlow.

Al igual que *TensorFlow*, *Keras* se encuentra en github [56], pues es de código abierto. Con *Keras* puedes realizar redes neuronales de forma sencilla y con muy pocas líneas de código, a diferencia de *TensorFlow* que es algo más complejo.

Las principales ventajas de *Keras* son que es fácil de usar, modular, y relativamente fácil de extender, haciendo muy simple su uso.

En este proyecto se hace uso de la versión 2.2.4 de *Keras*.

3.4. Darknet

*Darknet*³ es un *framework* para redes neuronales de código abierto escrito en C y CUDA. Es rápido, fácil de instalar y admite el empleo de CPU y GPU. El código se puede encontrar en github [57].

Darknet se creó con el fin de emplearse en el diseño, ejecución y entrenamiento de redes neuronales profundas para la clasificación y detección de objetos.

Las principales ventajas de este sistema son su simplicidad en términos de uso y su tamaño reducido.

En nuestro proyecto haremos uso de You only look once (YOLO), que es un sistema de detección de objetos que funciona sobre *Darknet*. YOLO utiliza *Deep Learning* y CNN para detectar objetos, y se distingue de sus competidores porque requiere de ver la imagen una sola vez, permitiéndole ser mucho más rápido que el resto de algoritmos. Gracias a su gran rapidez es capaz de detectar objetos en tiempo real en vídeos (hasta 30 Frames Per Second (FPS)).

A la hora de realizar la detección, YOLO en primer lugar divide la imagen en una cuadrícula de SxS. En cada una de sus celdas se estiman sus N posibles *bounding boxes* con sus respectivas probabilidades de acierto. Teniendo así SxSxN *bounding boxes*. Aquellas predicciones que tengan una probabilidad por debajo de un umbral quedarán eliminadas. A las que superen dicho umbral se les aplicará *non-max suppression*, lo cual sirve para eliminar objetos detectados por duplicado. Este proceso puede verse en la Figura 3.3.

³<https://pjreddie.com/darknet/>

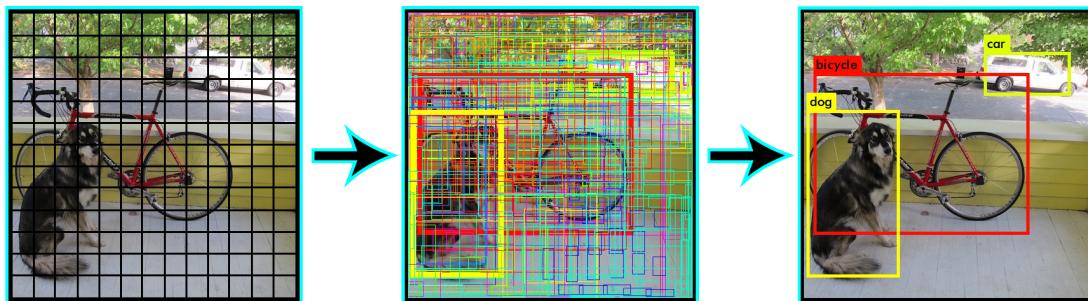


Figura 3.3: Proceso de Yolo

En este TFM se ha usado *Darknet* para el desarrollo de una red neuronal. La versión que se ha empleado es *YOLOv3*, es decir, la última versión existente.

3.5. Biblioteca OpenCV

*OpenCV*⁴ es una librería de código abierto desarrollada por *Intel* y publicada bajo licencia de BSD. Esta librería implementa gran variedad de herramientas para la interpretación de la imagen. Sus siglas provienen de los términos anglosajones “*Open Source Computer Vision Library*”, y tal y como se puede deducir es una librería destinada a aplicaciones de visión por computador en tiempo real. Puede ser empleada en MacOS, Windows y Linux, y existen versiones para *C#*, *Python* y *Java*, a pesar de que originalmente era una librería en *C/C++*. Además hay interfaces para *Ruby*, *Python*, *Matlab* y otros lenguajes.

OpenCV es una librería que implementa algoritmos para técnicas de calibración, detección de rasgos, rastreo, análisis de la forma, análisis del movimiento, reconstrucción 3D, segmentación de objetos y reconocimiento, etc. Los algoritmos se basan en estructuras de datos flexibles acopladas con estructuras IPL (*Intel Image Processing Library*), aprovechándose de la arquitectura de Intel en la optimización de más de las mitad de las funciones.

Incorpora funciones básicas para modelar el fondo, sustraer dicho fondo y generar imágenes de movimiento MHI (*Motion History Images*). Además incluye funciones para determinar dónde hubo movimiento y en qué dirección.

⁴<https://opencv.org/>

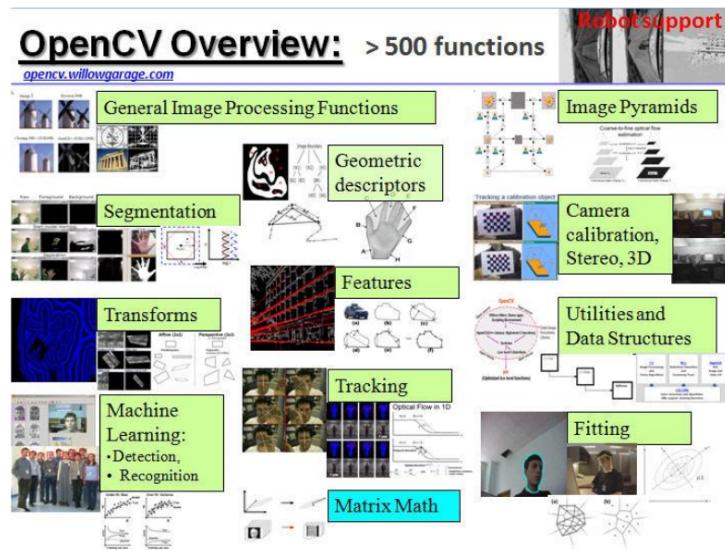


Figura 3.4: Funciones de OpenCV

Fue diseñado para tener una alta eficiencia computacional, está escrito en C y puede aprovechar las ventajas de los procesadores multinúcleo. Contiene más de 2500 funciones que abarcan muchas áreas de la visión artificial. También tiene una librería de aprendizaje automático (MLL, *Machine Learning Library*) destinada al reconocimiento y agrupación de patrones estadísticos.

Desde su aparición OpenCV ha sido usado en numerosas aplicaciones. Entre las cuales se encuentran la unión de imágenes de satélites y mapas web, la reducción de ruido en imágenes médicas, los sistemas de detección de movimiento, la calibración de cámaras, el manejo de vehículos no tripulados, el reconocimiento de gestos, etc. OpenCV es empleado también en reconocimiento de música y sonido, mediante la aplicación de técnicas de reconocimiento de visión en imágenes de espectrogramas del sonido.

Hay una gran cantidad de empresas y centros de investigación que emplean estas técnicas como IBM, Microsoft, Intel, SONY, Siemens, Google, Stanford, MIT, CMU, Cambridge e INRIA.

En este proyecto se hace uso de la versión *OpenCV 3.2*.

3.6. Lenguaje C++

El lenguaje *C++* nació en el laboratorio ATT, pero no surgió oficialmente hasta 1983, ya que fue entonces cuando comenzó a usarse. Fue creado por Bjarne Stroustrup, el cual

comenzó con *C++* en 1980.

Este lenguaje en un principio se llamaba *C* con clases, pues se diseñó conservando la mayoría de los conceptos de *C* y se añadieron características como por ejemplo clases. *C++* se desarrolló a partir de *C* y por tanto lo incluye. Esta parte de *C* incluida se llama *C-* y puede compilarse como *C++*.

Debido al gran éxito que obtuvo entre los programadores, la ATT comenzó a estandarizarlo internamente en 1987. Y en 1989 se formó un comité ANSI para estandarizarlo a nivel americano e internacional.

C++ es un lenguaje híbrido que ha tomado todas las características de la programación orientada a objetos y mejora las capacidades de *C*. Es muy versátil y potente, por ello ocupa el primer puesto como herramienta de desarrollo de aplicaciones.

3.7. Lenguaje Python

*Python*⁵ es un lenguaje de programación de alto nivel, muy fácil de aprender y de código abierto. *Python Software Foundation* es quien administra este lenguaje actualmente.

La primera versión surgió en 1901, pero no fue publicada hasta tres años después. Su creador fue Guido van Rossum, un investigador holandés, el cual trabaja en el centro de investigación CWI (Centrum Wiskunde & Informatica).

Este lenguaje incluye numerosas funcionalidades como el manejo de excepciones, la orientación a objetos, diccionarios, listas, etc. Dispone de módulos que permiten el uso de sockets, entrada y salida de ficheros, e incluso interfaces gráficas como Qt. Todo ello se creó con el objetivo de que fuera fácil de entender y emplear, pero sin perder funcionalidades que pueden ofrecer lenguajes más complejos como *C*. Al ser un lenguaje interpretado, no es necesario compilar el código.

Por compatibilidad con la versión original de *Smart-Traffic-Sensor* se ha empleado la versión 2.7.12 de *Python*.

⁵<https://www.python.org/>

3.8. Gtkmm

*Gtkmm*⁶ es una encapsulación en *C++* de *Gtk+*. Dicha encapsulación ofrece todos los beneficios de la orientación a objetos, e incorpora otras cualidades como una mejora en la comprobación de tipos, código más reducido y legible, y un menor uso de los punteros.

Se trata de un *toolkit* para desarrollar interfaces gráficas de usuario. Es un software libre distribuido bajo la Licencia Pública General Reducida de GNU (LGPL).

Gtkmm se organiza por medio de ventanas, las cuales contienen *widgets*, como por ejemplo botones, etiquetas, cuadros de texto, etc. Para cada *widget* debemos tener un objeto C++ con el cual controlar su funcionamiento, es decir si por ejemplo nuestro *widget* es un botón, necesitaremos funciones para saber si se ha pulsado o no, y acerca de que realizar si se hubiera pulsado, para todo ello se emplea un objeto de C++.

Aunque se puede especificar el diseño y apariencia de las ventanas y *widgets* con C++, resulta más conveniente usar *Glade* para el diseño de la interfaz de usuario y cargarlos en tiempo de ejecución con *Gtk::Builder*. Con *Glade* podemos desarrollar las interfaces de usuario, guardarlas en formato .glade y luego cargarlas desde *gtkmm* empleando *Gtk::Builder*.

En este caso se hace uso de la versión *gtkmm* 3.0.

⁶<http://www.gtkmm.org>

Capítulo 4

Diseño Global del Sistema

En este capítulo se hace una descripción del diseño global del sistema para llevar a cabo la monitorización de vehículos.

4.1. Diseño

Smart-Traffic-Sensor es un sistema que se realizó con el objetivo de monitorizar tráfico en tiempo real. Es decir para detectar, clasificar y hacer el seguimiento(*tracking*) de los diferentes vehículos que circulen por la carretera. Los vehículos que detectemos los clasificaremos en función a 7 clases: *car*, *motorcycle*, *van*, *bus*, *truck*, *small-truck* y *tank-truck*.

Actualmente se ha extendido mucho el uso de *Deep Learning* para la detección y clasificación de objetos, pues con ello se está llegando a obtener muy buenos resultados. Este trabajo se ha basado principalmente en el uso de *Deep Learning* para la clasificación y detección de vehículos. Además se ha apoyado en Kanade–Lucas–Tomasi (KLT), en los casos que pudiera haber pérdidas en la detección debido a occlusiones, o cuando los vehículos se encontraban muy alejados. Es decir, se ha apoyado en KLT cuando las condiciones a la hora de detectar eran algo complejas y por tanto el *Deep Lerning* no era capaz de realizar correctamente la detección.

A la hora de realizar el *tracking* se hace uso de la proximidad espacial, pero esto se explicará con más detalle en las siguientes secciones.

En resumen se puede decir que tenemos dos grandes bloques:

- Detección y Clasificación de vehículos

- Tracking de vehículos

4.2. Bases de Datos de Vídeos

La detección y clasificación de vehículos pretende encontrar un vehículo en una imagen o en un vídeo y determinar qué tipo de vehículo es. Dado que queremos encontrar el vehículo en cuestión bajo diferentes circunstancias, es decir, en diferentes entornos y con diferentes iluminaciones, necesitaremos entrenar el modelo con un conjunto de imágenes representativo. Por este motivo, a lo largo de los últimos años han surgido diferentes *datasets* con el fin de solucionar este problema. Para el caso de la detección de vehículos hay que decir que hay pocos *datasets*, por ello ha sido necesario crear una base de datos propia.

Esta base de datos propia consta de:

- La base de datos empleada por Redouane Kachach [58].
- La base de datos GRAM Road-Traffic Monitoring (GRAM-RTM) creada por R. Guerrero-Gomez-Olmedo, R. J. Lopez-Sastre, S. Maldonado-Bascon and A. Fernandez-Caballero [59]
- Imágenes recopiladas de cámaras en abierto de forma online.

Se ha tratado que la base de datos construida abarcara la mayor diversidad de vehículos posibles y en diferente tipos de escenarios. Hay que tener en cuenta que toda la base de datos tiene vehículos vistos por la parte trasera. En total consta de 9774 imágenes y está formada por 7 clases:

- Car
- Motorcycle
- Van
- Bus
- Truck
- Small-truck

- Tank-truck

En estas 9774 imágenes tenemos un total de 48914 muestras repartidas tal y como se muestra en la Tabla 4.1.

Clases	Muestras
Car	38976
Motorcycle	1886
Van	5631
Bus	401
Truck	963
Small-Truck	938
Tank-Truck	119

Tabla 4.1: Muestras de la Base de Datos

Para intentar conseguir un sistema robusto ante diferentes condiciones, se ha creado una base de datos que tiene imágenes en condiciones meteorológicas buenas, imágenes en condiciones meteorológicas malas (con niebla y lluvia) e imágenes de mala calidad. En la Tabla 4.2 se puede ver la cantidad de imágenes que hay para cada tipo.

	Nº de Imágenes
Buena calidad	8406
Malas Condiciones Meteorológicas	663
Mala Calidad	705

Tabla 4.2: Imágenes de la Base de Datos

4.3. Funcionamiento General

En esta sección se explica en qué técnicas se basa la aplicación *Smart-Traffic-Sensor* en cuanto a la detección, clasificación y el seguimiento de vehículos.

4.3.1. Detección y Clasificación de Vehículos

En este punto se va a explicar como se ha abordado el problema de la detección y clasificación de vehículos.

Tal y como ya se ha comentado anteriormente se ha hecho uso de *Deep Learning*. En concreto se ha diseñado un sistema capaz de soportar redes neuronales entrenadas con diferentes *frameworks* (*TensorFlow*, *Darknet* y *Keras*) con el objetivo de detectar y clasificar los diferentes vehículos que aparezcan en la imagen.

Se han probado tres *frameworks* diferentes para evaluar cual era el que mejores resultados obtenía. Para ello se hizo un primer entrenamiento con imágenes únicamente de buena calidad con el fin de determinar cual era el *frameworks* que mejor funcionaba.

En los siguientes puntos se va a explicar el diseño que se ha llevado a cabo con cada *frameworks*, asi como el diseño final empleado en *Smart-Traffic-Sensor*.

4.3.1.1. TensorFlow

Para llevar a cabo el diseño se ha hecho uso del *github models* de *TensorFlow* [60], con el cual podemos entrenar una red pre-entrenada con nuestra propia base de datos. En este caso se ha empleado una red Single Shot Detector (SSD) Mobilenet V2 entrenada con COCO, pues proporciona una buena relación entre la velocidad y la precisión. Para emplear dicha red se ha usado un archivo de configuración llamado *ssd_mobilenet_v2_coco.config* [61].

La red SSD MobileNet V2 es una red SSD que en lugar de tener una red VGG-16 de base, tiene una red MobileNet. En la Figura 4.1 se puede ver dicho diseño.

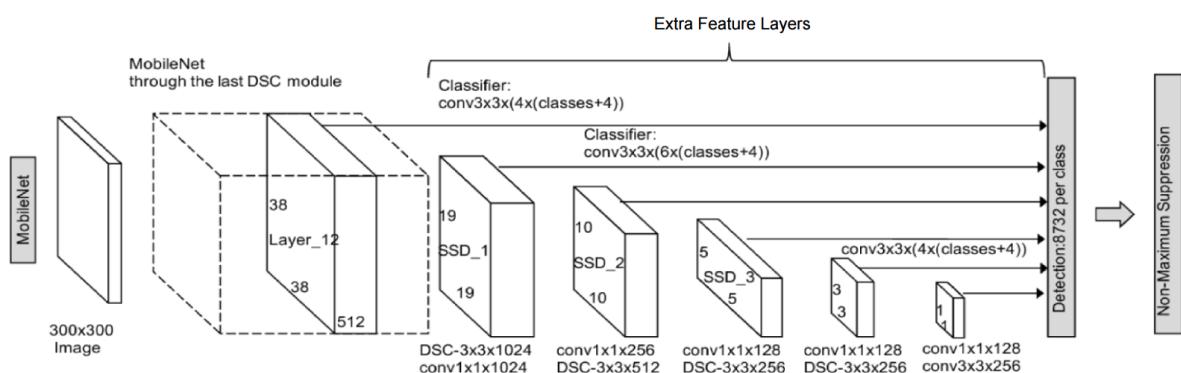


Figura 4.1: Red SSD Mobilenet V2

La primera parte de la red es la Mobilenet V2, en la cual se obtienen los mapas de características para poder realizar la clasificación y detección en las capas posteriores.

SSD [62] es un método de detección de objetos en imágenes empleando una única red neuronal profunda. SSD proporciona una gran ganancia de velocidad frente a Faster R-CNN [63], que funciona a una tasa de 7 frames por segundo.

El enfoque de SSD se basa en una red convolucional *feed-forward* que produce un conjunto de *bounding boxes* de tamaño fijo y puntúa la presencia de instancias de clase de objeto en esos *bounding boxes*. Tras esto se realiza *non-maximum suppression* para producir las detecciones finales.

Dada una imagen de entrada y un conjunto de etiquetas de *ground truth*, SSD realiza el siguiente proceso:

1. La imagen pasa a través de una serie de capas convolucionales, produciendo varios conjuntos de mapas de características a diferentes escalas.
2. Para cada ubicación en cada uno de estos mapas de características, emplea un filtro convolucional de 3x3 para evaluar un pequeño conjunto de *bounding boxes* por defecto.
3. Para cada *bounding box* predice simultáneamente el desplazamiento del *bounding box* y las probabilidades de clase.
4. Durante el entrenamiento, hace que coincida el *bounding box* del *ground truth* con los *bounding boxes* predichos según Intersección Over Union (IOU). El mejor *bounding box* predicho se etiquetará como "positivo", junto con todos los demás *bounding boxes* que tengan un ratio de *Intersection over Union* con el *ground truth* mayor de 0.5.

SSD parece sencillo, pero el entrenamiento tiene un desafío único. Clasificamos y estimamos *bounding boxes* desde cada posición en la imagen, usando múltiples formas diferentes, en diferentes escalas. Como resultado, generamos un número mucho mayor de *bounding boxes* que en otros modelos, y casi todos ellos son ejemplos negativos. Esto introduce un desequilibrio significativo entre los ejemplos de entrenamiento positivos y negativos.

Para solucionar este desequilibrio, SSD hace dos cosas. En primer lugar, utiliza la Non-Maximum Suppression (NMS) para agrupar *bounding boxes* muy superpuestos en un

solo *bounding box*. En otras palabras, si cuatro *bounding boxes* de formas, tamaños, etc. similares contienen el mismo perro, el NMS conservará el que tenga la mayor confianza y descartará el resto. En segundo lugar, el modelo usa una técnica llamada minería negativa para equilibrar las clases durante el entrenamiento. En la minería negativa dura, solo se utiliza un subconjunto de los ejemplos negativos con la mayor pérdida de entrenamiento (es decir, falsos positivos) en cada iteración de entrenamiento. SSD mantiene una relación de 3: 1 de negativos a positivos.

La red Mobilenet se basa en la clasificación. Mobilenet emplea unas capas llamadas *depthwise separable convolutions* en lugar de capas convolucionales. Las capas *depthwise separable convolutions* se componen de dos operaciones: *depthwise convolution* y *pointwise convolution*. En una capa convolucional se usan *kernels* que recorren toda la imagen con el objetivo de extraer información. Estos *kernels* necesitan tener la misma profundidad que la imagen para ser aplicados, es decir, si tenemos una imagen RGB necesitaremos 3 *kernels*, uno por cada canal de color. Finalmente estos *kernel* se combinan para obtener una única imagen.

En las capas *depthwise separable convolutions* se lleva a cabo el siguiente proceso:

1. Se realiza la operación de *depthwise convolution*, en la cual se aplican *kernels* de igual profundidad que la imagen. Cada *kernel* se emplea en cada canal de color por separado.
2. Se lleva a cabo la operación *pointwise convolution* en la cual se aplica un *kernel* de 1x1xprofundidad de la imagen, con lo que se obtendrá así un único canal.

Si por ejemplo tenemos una imagen de 10x10x3 y se aplican capas convolucionales con *kernels* de 3x3x3 el resultado será una imagen de 8x8x1. Si queremos aplicar 5 *kernels* en total tendremos 5 *kernels* de tamaño 3x3x3 que se moverán por la imagen 8x8 posiciones.

$$N^{\circ} \text{operaciones} = 5 \times 3 \times 3 \times 3 \times 8 \times 8 = 8640 \quad (4.1)$$

En el caso de *depthwise separable convolutions* primero se realizará la operación *depthwise convolution* con la cual obtendremos una imagen de 8x8x3 . Es decir, se emplearán 3 *kernel*st de tamaño 3x3x1 para cada canal de color y estos *kernels* se moverán 8x8 posiciones. Tras esto se aplicará la operación *pointwise convolution*, en la cual se usará un *kernel* de tamaño 1x1x3 para obtener finalmente una imagen de 8x8x1. Si tuvieramos 5 *kernels* de tamaño 1x1x3 se moverán 8x8 posiciones.

$$N^{\circ} \text{operaciones} = (3x3x3x1x8x8) + (5x1x1x3x8x8) = 2688 \quad (4.2)$$

En conclusión, las capas *depthwise separable convolutions* realizan el mismo trabajo que las convolucionales pero lo dividen en dos, consiguiendo así reducir el número de operaciones.

Con el diseño que se ha explicado se ha entrenado un total de 3173 imágenes, de las cuales 2700 eran de *train* y 473 de validación. Las imágenes de *train* son las que se usan para generar el modelo. Los datos de validación seleccionan el modelo que mejores resultados obtiene.

4.3.1.2. Keras

Con *Keras* se ha implementado una red SSD. Para ello se ha recurrido al diseño realizados por Pierluigi Ferrari [64], el cual define una red SSD que tiene como red base una VGG-16. En la Figura 4.2 se puede ver el diseño.

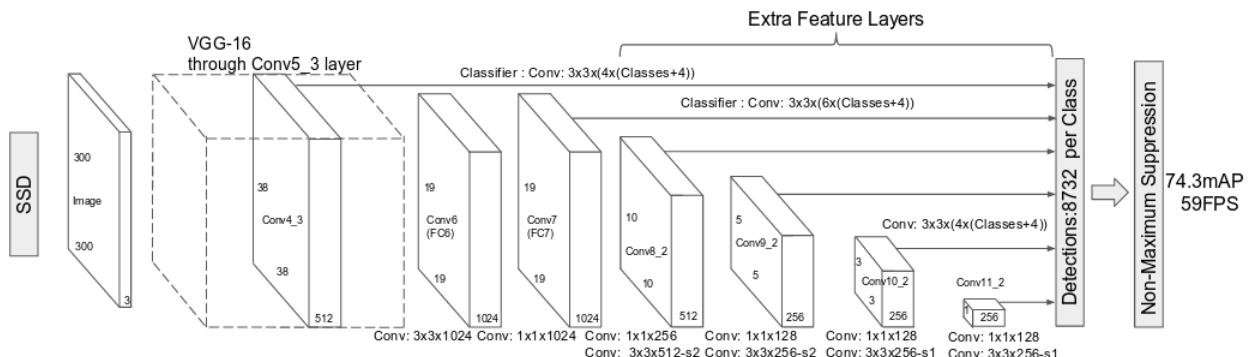


Figura 4.2: Red SSD

VGG-16 está formada por 16 capas, de las cuales 13 son capas convolucionales, 2 capas totalmente conectadas y una capa de *softmax* que se emplea para clasificar. En la Figura 4.3 se puede ver cual es la arquitectura de la red VGG.

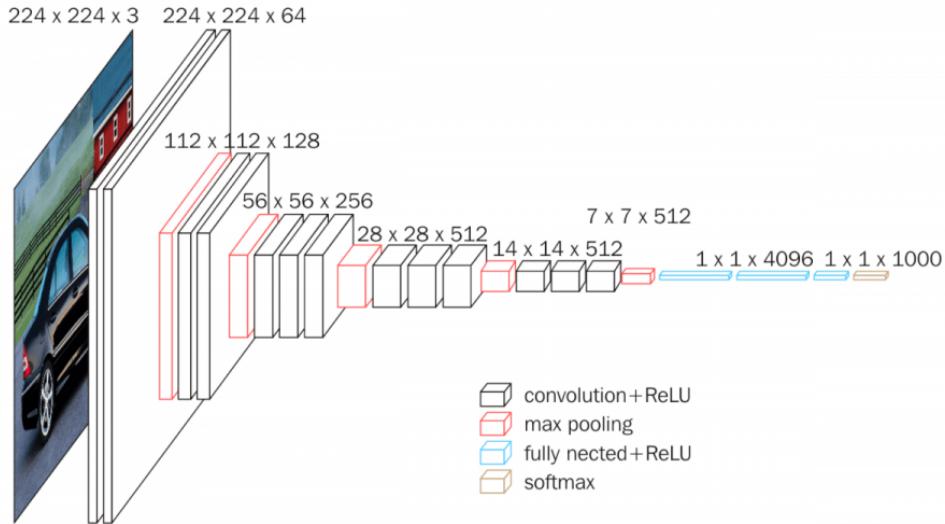


Figura 4.3: Red VGG-16

4.3.1.3. YOLO (Darknet)

En este sistema se ha incluido You only look once (YOLO) debido a su gran éxito actualmente. YOLO [65] es otro enfoque para la detección de objetos. El trabajo previo a YOLO emplea clasificadores para realizar la detección. En esta aproximación, se enmarca la detección de objetos como un problema de regresión en *bounding boxes* espacialmente separados y probabilidades de clase asociadas. En la evaluación, una red neuronal única predice *bounding boxes* y probabilidades de clase directamente desde imágenes completas.

El sistema de YOLO divide la imagen de entrada en una cuadrícula $S \times S$. Si el centro de un objeto cae en una celda de cuadrícula, esa celda de cuadrícula es responsable de detectar ese objeto. Cada celda de cuadrícula predice B *bounding boxes* y las puntuaciones de confianza para esos *bounding boxes*. Estas puntuaciones de confianza reflejan cómo está de seguro el modelo de que el *bounding box* contiene un objeto y también cómo de preciso es el *bounding box* que predice. Si no hay objeto en esa celda, las puntuaciones de confianza deben ser cero. De lo contrario, se espera que la puntuación de confianza sea igual al Intersección Over Union (IOU) entre el *bounding box* predicho y el *ground truth*.

YOLO impone fuertes restricciones espaciales en las predicciones de los *bounding boxes*, ya que cada celda de la cuadrícula solo predice dos *bounding boxes* y solo puede tener una clase. Esta restricción espacial limita el número de objetos cercanos que nuestro modelo puede predecir.

El modelo implementado es una red neuronal convolucional, donde las capas convolucionales iniciales extraen características de la imagen, mientras que las capas *fully connected* predicen las probabilidades y coordenadas de salida. La red tiene 24 capas convolucionales seguidas por 2 capas *fully connected*.

La arquitectura unificada de YOLO 4.4 es extremadamente rápida, procesando imágenes en tiempo real a 45 frames por segundo. Una versión más pequeña de la red, Fast YOLO, procesa 155 frames por segundo, logrando duplicar el mean Average Precision (mAP) de otros detectores en tiempo real.

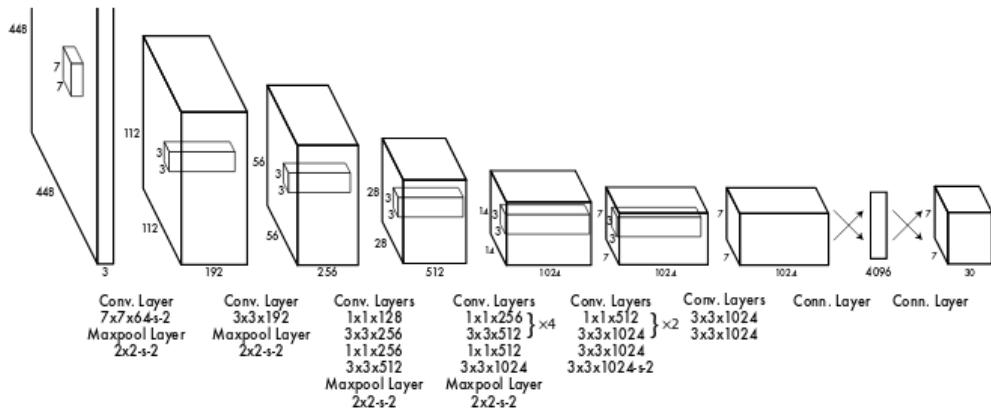


Figura 4.4: Arquitectura YOLO

En comparación con los sistemas de detección más modernos, YOLO comete más errores de localización (especialmente con objetos pequeños), pero es menos probable que prediga falsos positivos en el fondo. Finalmente, YOLO aprende representaciones muy generales de objetos. Supera a otros métodos de detección, incluidos DPM y R-CNN, como en imágenes naturales y trabajos artísticos.

4.3.1.4. Diseño Final

En el diseño final de *Smart-Traffic-Sensor* se usa *Darknet*, pues es el *framework* con el que hemos obtenido mejores resultados. Todas las pruebas realizadas para llegar hasta este diseño quedan explicadas en el Capítulo 5.

El sistema del que partíamos (*Traffic-Monitor* [1]) definía en la imagen una zona de entrada y otra de seguimiento. Esto puede verse en la Figura 4.5.

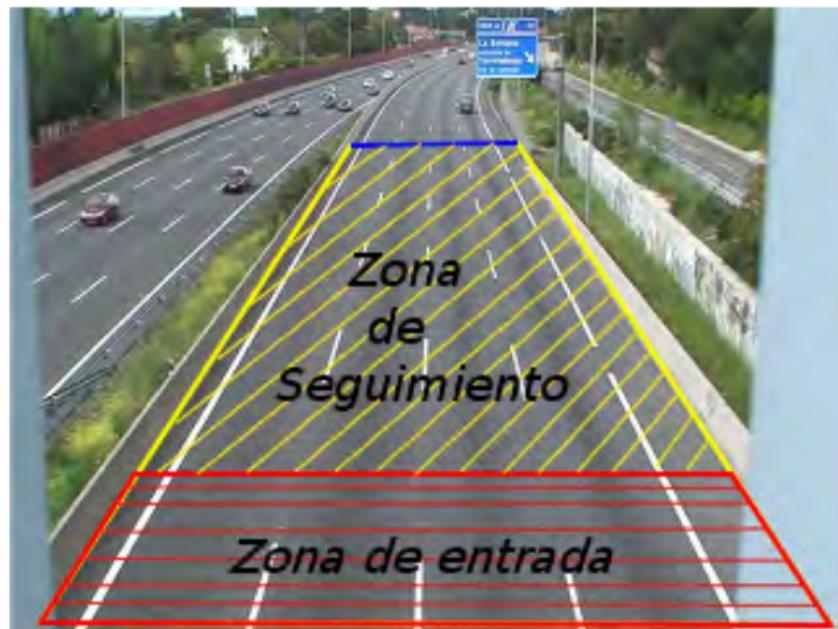


Figura 4.5: Zonas de entrada y seguimiento

En la zona de entrada se realizan las detecciones y en la de seguimiento es donde se lleva a cabo la clasificación y el tracking de los vehículos.

Este concepto de separar zona de entrada y seguimiento va a perder relevancia en nuestro sistema, pues ya no va a ser necesario hacer esta distinción. Tendremos una única zona en la que se lleve a cabo la detección, la clasificación y el tracking. Vamos a continuar teniendo una zona marcada en la imagen para identificar en qué parte de la carretera queremos centrar nuestras detecciones (por si existiesen carriles de diferente sentido). A esta zona de detección, clasificación y seguimiento vamos a llamarle zona de evaluación. En la Figura 4.6 se puede ver la zona de evaluación.



Figura 4.6: Zona Evaluación

La detección y clasificación de los vehículos se realiza gracias al uso de *Deep Learning*. En este caso gracias al uso de *Darknet* (YOLO). En vídeos en carretera es muy probable que tengamos occlusiones y por supuesto vehículos que se vayan alejando, los cuales son bastante complejos de detectar. Para solventar esto nos hemos apoyado en Kanade–Lucas–Tomasi (KLT). Por tanto tenemos un sistema que complementa *Deep Learning* con KLT.

Para tener un sistema lo más robusto posible se han tenido en cuenta varios detalles:

- Dentro de la zona de evaluación tenemos dos zonas. En la Figura 4.7 quedan identificadas esas dos zonas.

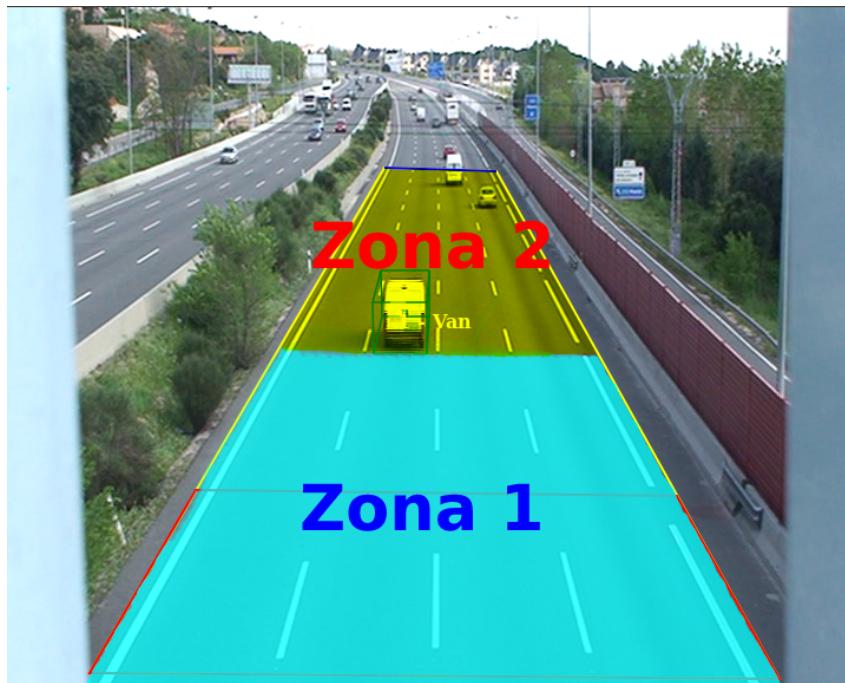


Figura 4.7: Zonas identificadas en la zona de Evaluación

La zona 1 se corresponde con la mitad de la zona de evaluación por donde entran los vehículos. En esta zona es más sencillo detectar y clasificar los vehículos pues tienen mayor tamaño. La zona 2 hace referencia a la mitad por la que salen los vehículos, la cual es más compleja, pues los vehículos tendrán menor tamaño.

- Un vehículo siempre va a entrar a la zona de evaluación por la parte de entrada. Nunca puede aparecer de repente. Por ello no puede aparecer ningún vehículo nuevo en el medio de la carretera, es decir nunca se podrá estimar que se ha detectado un vehículo nuevo en la zona 2.
- Si en la zona 1 un vehículo no es detectado durante 5 secuencias seguidas se dará por hecho que ha sido un falso positivo y por tanto quedará descartado.
- Todo vehículo que se encuentre en la zona 2 se considerará que es un vehículo correcto. Si mediante *Deep Learning* no somos capaces de detectar dicho vehículo, emplearemos KLT para localizarlo.

Como KLT va asociado al seguimiento de objetos se explicará en la Sección 4.3.2.

4.3.2. Seguimiento de Vehículos

Una vez tenemos los vehículos detectados y clasificados debemos realizar su seguimiento a lo largo de la carretera. Es decir, tenemos que asociar cada *blob* detectado a los *blob* anteriores de los que ya se llevaba un seguimiento. El algoritmo empleado para realizar este *tracking* está basado en la proximidad espacial.

La diferencia de píxeles en la imagen entre la posición de un vehículo en $t-1$ y en t es muy pequeña. Por tanto se puede decir que el *blob* de un vehículo en t cae en una zona muy cercana al *blob* de ese mismo vehículo en $t-1$. Esto se tendrá en cuenta a la hora de realizar el seguimiento, ya que cuando busquemos un vehículo en t deberíamos encontrarlo en un radio circular pequeño alrededor de la posición de ese mismo vehículo en $t-1$.

El algoritmo que se plantea en cuanto a la proximidad espacial es el empleado por Redouane Kachach [17]. En él se estima el área donde debería localizarse un vehículo en función de su posición en $t-1$. A medida que los vehículos vayan avanzando este área se irá actualizando.

Al principio el área se toma como un círculo pues no tenemos suficientes datos acerca de su orientación. Pero a medida que el vehículo avanza y tenemos suficiente información para conocer su orientación tomaremos el área como una elipse con centro en el centro del vehículo en $t-1$.

Consideraremos que tenemos suficiente información para estimar su orientación cuando tengamos 6 posiciones de un vehículo. Se empleará regresión lineal para calcular la orientación del vehículo basándonos en la posición que va tomando el vehículo a medida que va avanzando.

La regresión lineal consiste en minimizar $\sum_i \rho(r_i)$, donde r_i es la distancia con el i -ésimo punto y $\rho(r_i)$ es una función de la distancia. $\rho(r_i)$ se puede calcular como:

$$\rho(r_i) = 2\left(\sqrt{1 + \frac{r^2}{2}} - 1\right) \quad (4.3)$$

Una vez tenemos información acerca de la orientación definiremos el área de búsqueda como una elipse que tiene como centro el mismo que el vehículo en $t-1$ y dirección la calculada con la Fórmula 4.3.

Los emparejamientos entre los vehículos detectados en el instante t y los *blob* almacenados del instante $t-1$ se limitarán a los vehículos que caen dentro del área del círculo o la elipse que se obtiene en función de la posición del vehículo en el instante $t-1$.

En la Figura 4.8 se puede ver la evolución que sufre el área alrededor del vehículo a medida que avanza.

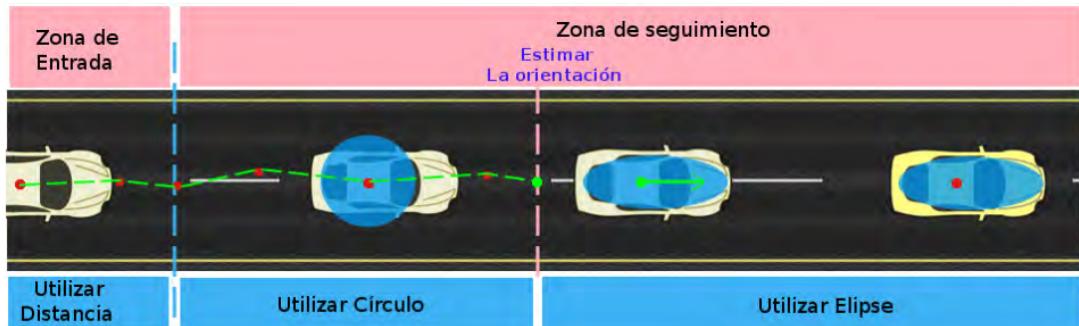


Figura 4.8: Evolución del área alrededor del vehículo

Estas elipses se definen como $C_{xc,yc,\omega}$, donde ω es la orientación y (x_c, y_c) es el centro del vehículo. Además a la hora de describir una elipse debemos conocer sus dos ejes. Estos parámetros se pueden ver en la Figura 4.9.

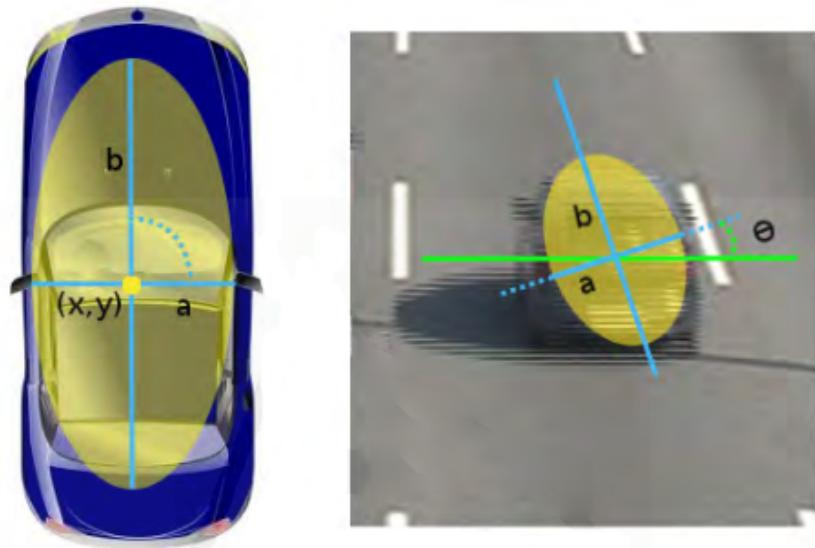


Figura 4.9: Elipse asociada a los vehículos

Un blob 2D cuyo centro es $B(x, y)$ se encontrará dentro de la elipse $C_{xc,yc,\omega}$ si se cumple:

$$C_\omega = \arctan\left(\frac{a_x}{a_y}\right) \quad (4.4)$$

$$\left(\frac{\cos(C_\omega)(B_x - C_{x_c}) + \sin(C_\omega)(B_y - C_{y_c})}{a} \right)^2 + \left(\frac{\cos(C_\omega)(B_y - C_{y_c}) - \sin(C_\omega)(B_x - C_{x_c})}{b} \right)^2 \leq 1 \quad (4.5)$$

a_x y a_y son los componentes del vector de orientación. En la Figura 4.10 se puede ver como se realiza el *tracking* entre dos *blob* consecutivos.

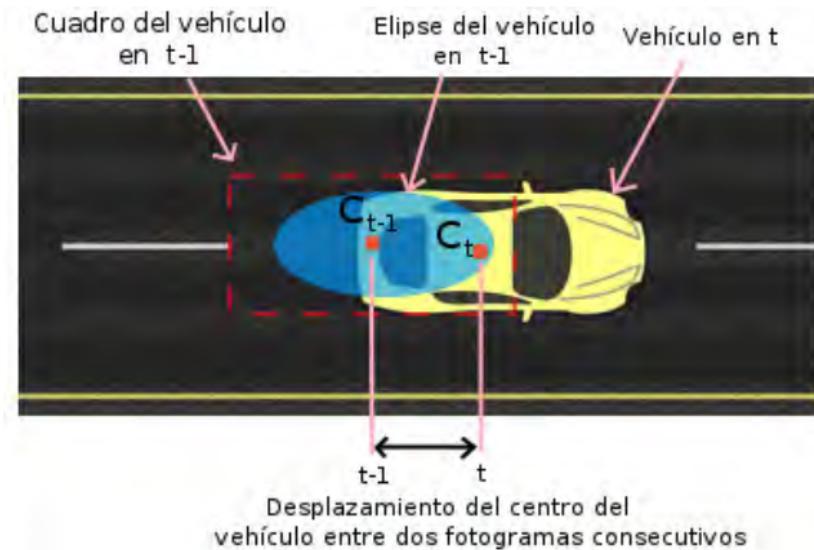


Figura 4.10: Seguimiento con proximidad espacial

En resumen una detección deberá encontrarse dentro de un cierto área alrededor del *blob* detectado en $t-1$ para poder ser identificado como el mismo vehículo. Podría darse el caso de que dos vehículos cayeran en dicho área. Por ello es necesario tener en cuenta la distancia euclídea entre el centro del blob del instante $t-1$ y el centro de los *blob* en t . Aquel *blob* del instante t que se encuentre a menor distancia del *blob* del instante $t-1$ y por supuesto esté dentro del área alrededor del *blob* $t-1$ será considerado como el mismo vehículo que el de $t-1$. Es decir si esto se cumple el *blob* de $t-1$ y t corresponden al mismo vehículo pero en instantes consecutivos.

Como ya se ha mencionado en apartados anteriores si no se detecta un vehículo ya sea porque haya alguna oclusión o se encuentre muy lejos se empleará KLT que es muy robusto y se ha visto que funciona incluso ante oclusiones.

Para poder realizar KLT necesitamos conocer el centro de masas de los vehículos y sus características visuales. En función de los puntos característicos del vehículo en $t-1$, KLT calculará el emparejamiento para cada punto característico y como resultado generará un nuevo conjunto de puntos correspondientes al vehículo en cuestión. Para llegar a conseguir

un emparejamiento correcto el sistema se basa en votos. A continuación se va a explicar más en detalle KLT.

Jean-Yves Bouguet [66] hicieron una implementación de KLT en la cual aplicaban KLT de forma recursiva sobre una pirámide de imágenes. Este misma implementación es la que se ha empleado en este trabajo.

Lukas Kanade es un método diferencial y local en el que se analiza la vecindad de cada píxel. En él se asume que el flujo óptico es constante en una vecindad, y se resuelve la ecuación del flujo óptico para todos los píxeles en esta vecindad por el método de los mínimos cuadrados. Para el cálculo de los vectores de velocidad se emplea la siguiente fórmula:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i I_{xi}^2 & \sum_i I_{xi} I_{yi} \\ \sum_i I_{xi} I_{yi} & \sum_i I_{yi}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_{xi} I_{ti} \\ -\sum_i I_{yi} I_{ti} \end{bmatrix} \quad (4.6)$$

El vector (u, v) es el vector de desplazamiento del flujo óptico. I_x es la media del gradiente en x entre dos imágenes consecutivas, es decir, si $I(t)$ es la imagen del instante actual e $I(t + 1)$ es la imagen en el instante siguiente, la I_x de estos fotogramas es:

$$I_x = \frac{I_x(t) + I_x(t + 1)}{2} \quad (4.7)$$

$I_x(t)$ es el gradiente en el eje x de la imagen $I(t)$ e $I_x(t + 1)$ es el gradiente en x de la imagen $I(t + 1)$. I_y es la media de los gradientes en y de la imagen $I(t)$ e $I(t + 1)$:

$$I_y = \frac{I_y(t) + I_y(t + 1)}{2} \quad (4.8)$$

I_t es la diferencia entre $I(t)$ suavizada e $I(t + 1)$ suavizada:

$$I_t = I'(t + 1) - I'(t) \quad (4.9)$$

Tal y como se ha dicho KLT se aplica en forma de *kernels* de tamaño $\omega \times \omega$ a lo largo de la imagen. El tamaño de los *kernels* debe ser definido en función de la cantidad de movimiento que tenga la imagen. Un valor de *kernel* pequeño será idóneo para evaluar desplazamientos pequeños de un punto. El uso de un tamaño grande de *kernel* aumenta el riesgo de obtener un error, pero hay en casos en los que el desplazamiento de un punto es muy grande y esto es necesario.

En la Figura 4.11 se puede ver como funciona KLT de forma piramidal.

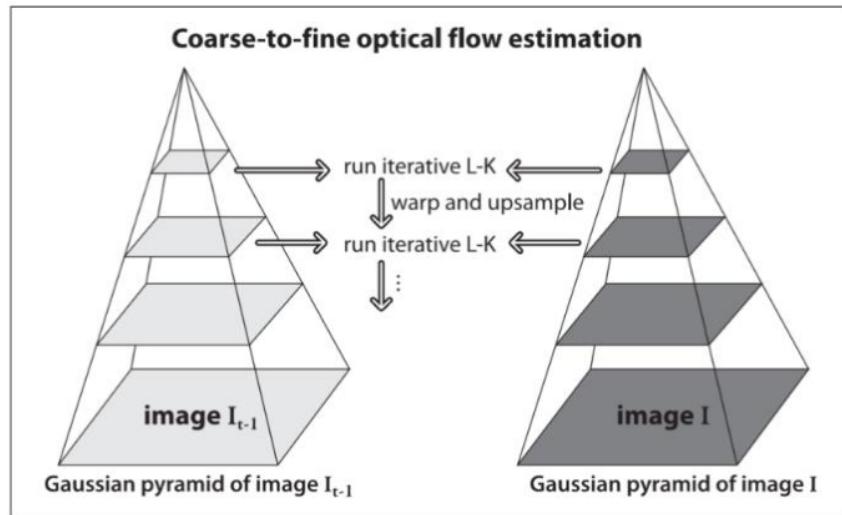


Figura 4.11: KLT Piramidal

Gracias al empleo de KLT piramidal se pueden estimar grandes desplazamientos con un tamaño de ventana muy pequeño.

En resumen el seguimiento se realiza mediante la proximidad espacial pero se hará uso de KLT en casos problemáticos, haciendo así más robusto nuestro sistema. KLT se calculará en todas las secuencias para ir actualizando los puntos característicos.

4.3.3. Resumen General del Sistema

Smart-Traffic-Sensor ha sido diseñado para llevar a cabo la monitorización de vehículos. Esta monitorización consta de tres elementos principales:

- Detección de vehículos
- Clasificación de vehículos
- Seguimiento de vehículos

Las detecciones y la clasificación van de la mano, pues se realizan con *Deep Learning*. En concreto con una red de YOLO. El seguimiento se centra en la proximidad espacial, y si esta falla se utiliza KLT. A todos los *blob* detectados se les realizará un seguimiento a lo largo del tiempo.

En cuanto a las detecciones y la clasificación hay que tener en cuenta que no pueden aparecer vehículos nuevos en medio de la imagen, por tanto si esto sucede se considerará un falso positivo y se descartará. Otro aspecto que hay que tener en cuenta es que se podría dar una detección errónea, por ello si durante 5 secuencias seguidas un vehículo no ha sido detectado en la zona 1 se considerará como que era un falso positivo y se dejará de hacer su seguimiento.

El seguimiento se centra en asociar las detecciones actuales con los *blobs* almacenados del instante anterior. En el seguimiento se tendrán ciertos aspectos en cuenta:

- Si un *blob* llega al final de la zona de evaluación se eliminará del seguimiento.
- Se recorrerán los *blob* del instante $t-1$ almacenados con el fin de emparejarlos con los *blob* detectados en el instante t . Este emparejamiento se establecerá entre los *blob* t y $t-1$ que tengan menor distancia euclídea entre sus centros.
- Si el *blob* t asociado al *blob* $t-1$ no cae dentro del área circular o elíptica que se genera alrededor del centro del *blob* $t-1$ no quedará emparejado a éste.
- Si mediante proximidad espacial no somos capaces de emparejar un *blob* $t-1$ emplearemos KLT.

4.4. Estimación de la Velocidad

Smart-Traffic-Sensor tiene opciones para configurar la cámara, es decir para realizar su calibración. Al tener información de la cámara empleada tendremos información 3D de la imagen. Es decir podemos realizar una proyección al mundo 3D de nuestros puntos en la imagen.

Para calcular la velocidad solo necesitamos conocer la distancia que ha recorrido el vehículo y el tiempo que ha tardado en hacerlo. Disponemos de una zona de evaluación por la cual discurren los vehículos, por tanto podemos saber cuanta distancia recorre por dicha zona de evaluación, y el tiempo que tarda en hacerlo. Para calcular la distancia tenemos que calcular la posición 3D de los vehículos y con ella hacer su diferencia. Esto se puede llevar a cabo haciendo uso de su homografía, la cual podemos estimar gracias a que conocemos los parámetros de la cámara.

Hay que puntualizar que las pruebas que se han hecho con *Smart-Traffic-Sensor* se han realizado sobre un servidor, por tanto el hecho de usar sus gráficos hace que se ralentice

el sistema. Esto se verá en la velocidad, la cual se ha percibido que va 5 veces más lenta que si lo probaramos en un ordenador con GPU propia.

Capítulo 5

Experimentos

A la hora de crear cualquier sistema es necesario hacer múltiples experimentos que nos permitan llegar a ciertas conclusiones que nos sean útiles a la hora de mejorar el diseño. Dichos experimentos deben ser evaluados de alguna forma para comprobar la calidad de los resultados que obtenemos. En este trabajo en concreto se ha recurrido a la herramienta *DeepLearningSuite* [54] para llevar a cabo dicha evaluación.

Con *DeepLearningSuite* se obtienen medidas de calidad tales como la precisión y el recall medios (mAP y mAR) de todas las clases en función de la Intersección Over Union (IOU). En nuestro caso nos hemos quedado con las medidas que tienen un IOU como mínimo de 0.5, pues se ha considerado que es suficiente para medir la calidad.

Este Capítulo vamos a dividirlo en cinco secciones en las cuales hablaremos de lo siguiente:

- Evaluación de diversas redes neuronales
- Evaluación ampliando Dataset
- Evaluación de Darknet con diferentes conjuntos de imágenes
- Evaluación de Smart-Traffic-Sensor
- Evaluación de los tiempos de procesamiento

A lo largo de este Capítulo se van a ver diferentes experimentos realizados y las medidas obtenidas con ellos en cuanto al mean Average Precision (mAP), mean Average Recall (mAR) con un mínimo de 0.5 IOU, y el tiempo de procesamiento que dedica al realizar las detecciones.

Hay que decir que todos los experimentos realizados con *DeepLearningSuite* se han hecho desde un ordenador sin GPU, por ello se verá que los tiempos medios de procesamiento son un poco elevados.

5.1. Evaluación de diversas redes neuronales

Tal y como se ha comentado en el Capítulo 4 se han realizado pruebas con diferentes *frameworks*. En concreto con *TensorFlow*, *Keras*, y *Darknet*. Con ello se pretendía ver cual era el framework que mejores resultados obtenía, además de dotar a *Smart-Traffic-Sensor* de la capacidad de admitir redes entrenadas con diversos *frameworks*.

Para llevar a cabo el entrenamiento en todos los *frameworks* mencionados se han empleado un total de 3173 imágenes, de las cuales 2700 eran de *train* y 473 de validación. Todas ellas en condiciones meteorológicas favorables y con buena calidad. La distribución de las muestras que posee la base de datos se indica en la Tabla 5.1.

Clases	Muestras
Car	15798
Motorcycle	143
Van	1437
Bus	274
Truck	765
Small-Truck	400
Tank-Truck	103
Total	18920

Tabla 5.1: Muestras de la Base de Datos del 1º entrenamiento

En el entrenamiento se ha partido de modelos pre-entrenados. Para ver como el entrenamiento con nuestros datos consigue mejorar la capacidad de las redes a la hora de detectar objetos, se han evaluado las redes pre-entrenadas y las entrenadas. La evaluación se ha realizado sobre un conjunto de 303 imágenes, con muestras de diferentes clases de vehículos. En la Tabla 5.2 se puede ver que clases de muestras contenían las imágenes con las que se ha realizado el test.

CAPÍTULO 5. EXPERIMENTOS

Clases	Muestras
Car	922
Motorcycle	19
Van	125
Truck	82
Small-Truck	112
Total	1260

Tabla 5.2: Muestras de los datos de Test de la primera evaluación

En la Tabla 5.3 se pueden ver los resultados obtenidos tras evaluar las redes pre-entrenadas con algunos datos de test.

Redes Neuronales	mAP	mAR	Mean Inference Time (ms)
Keras(VGG_ILSVRC_16_layers_fc_reduced.h5)	0	0	0
TensorFlow (frozen_inference_graph.pb)	0.0035	0.0373	142
Darknet (darknet53.conv.74)	0	0	14162

Tabla 5.3: Resultados Redes Pre-Entrenadas

Los resultados que se obtienen con nuestras redes entrenadas se pueden ver en la Tabla 5.4.

Redes Neuronales	mAP	mAR	Mean Inference Time (ms)
Keras	0.6709	0.7082	3194
TensorFlow	0.3283	0.4231	76
Darknet	0.8641	0.9385	16894

Tabla 5.4: Resultados Redes Entrenadas

Con toda esta información se puede ver claramente que es necesario re-entrenar las redes con nuestros datos para tener una cierta calidad, ya que cuanto más rico sea el *dataset* con el que se entrena más información podrá adquirir acerca de los objetos a detectar.

Con esta evaluación también se puede ver que los resultados que se obtienen por la red entrenada con *Darknet* son mejores que los de *TensorFlow* y *Keras*. Hay que recordar que con *Darknet* se implementó una red YOLO, con *TensorFlow* una SSD Mobilenet y con *Keras* una SSD VGR-16.

Viendo los tiempos de detección se puede ver que cuanto más tiempo tarda la red en realizar la detección mejores resultados obtiene. Esto puede deberse a que la red contiene más capas neuronales o que posee mayor complejidad y por tanto dedica mayor tiempo al procesamiento.

5.2. Evaluación ampliando Dataset

Tras la evaluación incial, la primera cuestión que se nos plantea es el enriquecer nuestra base de datos con mas imágenes para volver a entrenar la red y ver si sus resultados mejoran.

La base de datos empleada en este caso para realizar el entrenamiento consta de 6717 imágenes, todas ellas de buena calidad y en buenas condiciones meteorológicas. De esas 6717 imágenes, 5323 se toman como *train* y 1394 como validación. En la Tabla 5.5 se puede ver la distribución de las muestras de la base de datos.

Clases	Muestras
Car	28655
Motorcycle	1517
Van	4675
Bus	274
Truck	874
Small-Truck	663
Tank-Truck	103
Total	36762

Tabla 5.5: Características de la Base de Datos de Imágenes de Buena Calidad

Con esta nueva base de datos se ha vuelto a realizar el entrenamiento para evaluar como afecta el enriquecimiento de los datos. Además se ha evaluado una red entrenada

por Arvind Jayaraman [67] para la detección de vehículos con *TensorFlow*. Todo estos datos quedan recogidos en la Tabla 5.6

Redes Neuronales	mAP	mAR	Mean Inference Time (ms)
Keras	0.7478	0.7831	3427
TensorFlow	0.5484	0.61361	83
Darknet	0.9180	0.9499	15357
Arvind Jayaraman	0.0384	0.0613	289

Tabla 5.6: Resultados Redes Entrenadas

Con estos resultados se puede afirmar que el enriquecer la base de datos con mayor información nos da mayor calidad. Además se ha evaluado la red entrenada por Arvind Jayaraman, con lo que se ha visto que se obtienen muy malos resultados.

5.3. Evaluación de Darknet con diferentes conjuntos de imágenes

En este punto ya tenemos identificada cual es la red neuronal que mejores resultados obtiene (*Darknet*). Es con esta red con la que vamos a continuar haciendo experimentos, es decir, nos hemos centrado en esta red con el fin de mejorar nuestros resultados y llegar a un sistema lo más robusto posible.

Para poder mejorar nuestra red neuronal se ha incrementado la base de datos con imágenes de mala calidad y en condiciones meteorológicas desfavorables (lluvia y niebla). En las Figuras 5.1, 5.2 y 5.3 se pueden ver ejemplos de dichas imágenes.

CAPÍTULO 5. EXPERIMENTOS



Figura 5.1: Ejemplos de Imágenes de Buena Calidad



Figura 5.2: Ejemplos de Imágenes de Malas Condiciones Climatológicas (Niebla y lluvia)



Figura 5.3: Ejemplos de Imágenes de Mala Calidad

CAPÍTULO 5. EXPERIMENTOS

En total se ha creado una base de datos final para el entrenamiento de 9246 imágenes, de las cuales 7401 se usaban como *train* y 1845 como validación. En la Tabla 5.7 se puede observar la cantidad de imágenes que se han empleado en el entrenamiento en función de su tipo (buena calidad, mala calidad y condiciones meteorológicas desfavorables).

Tipo	Imágenes de Train	Imágenes de Test	Total
Buena Calidad	5323	1394	6717
Condiciones Meteorológicas malas	1568	324	1892
Mala Calidad	510	127	637

Tabla 5.7: Base de Datos Final

La cantidad de muestras de cada clase que tienen las imágenes de buena calidad puede verse en la Tabla 5.5. Los tipos de muestras que hay en las imágenes de malas condiciones climatológicas pueden verse en la Tabla 5.8 y los de las imágenes de mala calidad en la Tabla 5.9.

Clases	Muestras
Car	6921
Motorcycle	335
Van	709
Bus	100
Small-Truck	183
Total	8248

Tabla 5.8: Características de la Base de Datos con condiciones climatológicas desfavorables

Clases	Muestras
Car	1571
Motorcycle	21
Van	56
Bus	27
Truck	82
Small-Truck	60
Tank-truck	16
Total	1833

Tabla 5.9: Características de la Base de Datos de Mala Calidad

Para evaluar como afecta el hecho de incorporar imágenes con diferentes condiciones se ha realizado un estudio que engloba 3 etapas:

1. En primer lugar se entrenó la red neuronal con imágenes de buena calidad y se evaluó dicha red con conjuntos de test de imágenes de buena calidad, malas condiciones meteorológicas y mala calidad.
2. Entrenamos la red neuronal con imágenes de buena calidad y malas condiciones climatológicas. La red resultante la evaluámos con imágenes de buena calidad, malas condiciones meteorológicas y mala calidad.
3. Finalmente realizamos un entrenamiento con toda la base de datos y lo evaluámos con imágenes de todos los tipos.

5.3.1. Red Neuronal con Imágenes de Buena Calidad

En un primer lugar tal y como se contaba en la Sección 5.2 se entrenó nuestra red neuronal con un conjunto de un total de 6717 imágenes de buena calidad. Las imágenes de este conjunto incluían información acerca de las diferentes clases que contempla nuestro modelo. Se puede ver en la Tabla 5.5 la cantidad de muestras de cada clase que contiene este conjunto.

El objetivo de entrenar la red neuronal con solo imágenes de buena calidad es ver como se comporta frente a imágenes de diferentes condiciones. Para ello se ha evaluado

CAPÍTULO 5. EXPERIMENTOS

este modelo con los 3 tipos de imágenes que tenemos (buena calidad, mala calidad y condiciones desfavorables) por separado y finalmente con un conjunto que incluía imágenes de todos los tipos.

En la Tabla 5.10 se muestra información acerca de las imágenes de test de buena calidad.

Nº de Imágenes	389
Nº de Muestras Totales	1657
Nº Car	1463
Nº Motorcycle	9
Nº Van	155
Nº Truck	8
Nº Small-Truck	22

Tabla 5.10: Imágenes de Test de Buena Calidad

La Tabla 5.11 nos da información acerca de las imágenes de test de malas condiciones meteorológicas empleadas en la evaluación.

Nº de Imágenes	71
Nº de Muestras Totales	287
Nº Car	263
Nº Motorcycle	1
Nº Van	23

Tabla 5.11: Imágenes de Test de Malas Condiciones Meteorológicas

El conjunto de imágenes de test con mala calidad queda caracterizado en la Tabla 5.12.

CAPÍTULO 5. EXPERIMENTOS

Nº de Imágenes	68
Nº de Muestras Totales	199
Nº Car	176
Nº Motorcycle	3
Nº Van	11
Nº Small-Truck	9

Tabla 5.12: Imágenes de Test de Mala Calidad

Tras evaluar el modelo entrenado con todos los conjuntos de datos de test se han obtenido los resultados indicados en la Tabla 5.13. Hay que decir que se ha evaluado los conjuntos de datos por separado y finalmente se ha hecho una evaluación a un grupo de imágenes de test, a las cuales hemos llamado *Combinado* (incluyen 68 imágenes de cada tipo). Este grupo de imágenes llamado *Combinado* se ve caracterizado en la Tabla 5.14.

Conjuntos de Test	Nº de Imágenes	mAP	mAR
Buena Calidad	389	0.9200	0.9494
Malas Condiciones Meteorológicas	71	0.8986	0.9379
Mala Calidad	68	0.4727	0.5470
Combinado	204	0.8311	0.8599

Tabla 5.13: Resultados Modelo entrenado con Imágenes de Buena Calidad

Nº de Imágenes	204
Nº de Muestras Totales	771
Nº Car	666
Nº Motorcycle	7
Nº Van	77
Nº Truck	5
Nº Small-Truck	16

Tabla 5.14: Conjunto de Test Combinado

Observando los resultados se puede comprobar como la red entrenada se comporta perfectamente con las imágenes de buena calidad tal y como era de esperar. Con las imágenes con condiciones meteorológicas desfavorables los resultados también son bastante buenos a pesar de no haber sido entrenada con dichos datos. Pero en el caso de las imágenes de mala calidad se obtienen resultados muy pobres.

5.3.2. Red Neuronal con Imágenes de Buena Calidad y Malas Condiciones Meteorológicas

Tras realizar la evaluación del modelo entrenado con imágenes de buena calidad se procedió a entrenar de nuevo el modelo incluyendo imágenes con condiciones climatológicas malas. Es decir, se ha entrenado el modelo con 6717 imágenes de buena calidad y 1892 imágenes con malas condiciones meteorológicas. Los resultados que se obtienen tras la evaluación pueden comprobarse en la Tabla 5.15.

Conjuntos de Test	Nº de Imágenes	mAP	mAR
Buena Calidad	389	0.7759	0.8488
Malas Condiciones Meteorológicas	71	0.9697	0.9753
Mala Calidad	68	0.6835	0.6957
Combinado	204	0.8188	0.8442

Tabla 5.15: Resultados Modelo entrenado con Imágenes de Buena Calidad y Condiciones Meteorológicas Malas

El hecho de incluir imágenes con condiciones meteorológicas malas hace que el modelo sea capaz de funcionar en mejores condiciones con dicho tipo de imágenes. Además al tener estas imágenes de peor calidad debido a las malas condiciones meteorológicas beneficia al modelo a la hora de detectar vehículos en imágenes de mala calidad. Aunque también afecta a las imágenes de buena calidad, pues la calidad de la detección queda un poco reducida. Si nos fijamos en los resultados con el conjunto total son un pelín inferiores al modelo entrenado únicamente con imágenes de buena calidad, pero son bastante similares.

5.3.3. Red Neuronal con todo el conjunto de datos

Finalmente se ha entrenado el modelo con toda la base de datos. Esta base de datos tiene las características que se indican en las Tablas 5.7, 5.5, 5.8 y 5.9. Los resultados que se obtienen tras la evaluación pueden verse en la Tabla 5.16.

Conjuntos de Test	Nº de Imágenes	mAP	mAR
Buena Calidad	389	0.7287	0.7802
Malas Condiciones Meteorológicas	71	0.9730	0.9779
Mala Calidad	68	0.8844	0.9010
Combinado	204	0.8606	0.8899

Tabla 5.16: Resultados Modelo entrenado con Toda la Base de Datos

Al entrenar la red con toda la base de datos los resultados en cuanto a las imágenes de buena calidad quedan algo reducidos de nuevo. Pero en el caso de las imágenes de mala calidad y las de condiciones climatológicas desfavorables los resultados mejoran respecto a los experimentos anteriores. Consiguiendo así que la evaluación sobre un conjunto de datos de todos los tipos quede mejorada respecto a las pruebas anteriores.

Con esto se puede ver que cuanto más diversidad tenga la base de datos mayor capacidad tendrá para detectar vehículos en diferentes escenarios. Es decir, enriquecer la base de datos con mayores casos hace que el sistema sea más robusto frente a cambios.

5.4. Evaluación de Smart-Traffic-Sensor

Smart-Traffic-Sensor se basa principalmente en *Deep Learning*, pero lo combina con KLT cuando las detecciones realizadas por *Deep Learning* no son suficientes. Esto dota al sistema de mayor robustez.

El objetivo de este punto es evaluar la calidad del sistema *Smart-Traffic-Sensor* y compararla con el sistema inicial del que se partía (*Traffic-Monitor* [17]). En este sistema la carretera se dividía en una zona de entrada y otra de seguimiento. En la zona de entrada se confirmaba la detección de cada vehículo para posteriormente llevarle un seguimiento. El sistema de detección que emplea se basa en la detección del fondo para obtener las detecciones de los vehículos. Una vez pasan estos vehículos a la zona de seguimiento

es cuando comienza a clasificarlos y emparejarlos. El seguimiento combina KLT con proximidad espacial al igual que se hace en *Smart-Traffic-Sensor*.

Para la evaluación hemos empleado tres videos con condiciones diferentes (uno con buena calida, otro de lluvia y por último uno de mala calidad). *DeepLearningSuite* realiza evaluaciones de modelos entrenados sobre un conjunto de datos. En este caso no queremos evaluar el modelo sino el sistema global. Por ello se ha modificado *DeepLearningSuite* para poder evaluar muestras que se almacenen en un *.txt*.

En resumen la evaluación del sistema *Smart-Traffic-Sensor* se ha realizado siguiendo los siguientes pasos:

1. Se ejecuta *Smart-Traffic-Sensor* con un vídeo y se guardan sus detecciones finales en archivos *.txt*, así como sus respectivas imágenes.
2. Nos quedamos con una parte de las detecciones e imágenes obtenidas. Por ejemplo cada 6 imágenes nos quedamos con una. Esto se hace pues sino tendríamos que evaluar una cantidad excesiva de imágenes.
3. Etiquetamos las imágenes almacenadas con *labelImg* [68]. Pues necesitamos comparar las detecciones realizadas por *Smart-Traffic-Sensor* con alguna referencia.
4. Comparamos las detecciones obtenidas por *Smart-Traffic-Sensor* con las etiquetas. Para ello nos hemos apoyado en *DeepLearningSuite*.

A continuación se muestran los resultados obtenidos para cada video con *Smart-Traffic-Sensor*, *Traffic-Monitor* [17] y empleando únicamente el modelo entrenado.

En primer lugar se evaluó un vídeo de buena calidad, del cual podemos ver en la Figura 5.4 una de las detecciones realizadas por *Smart-Traffic-Sensor*.

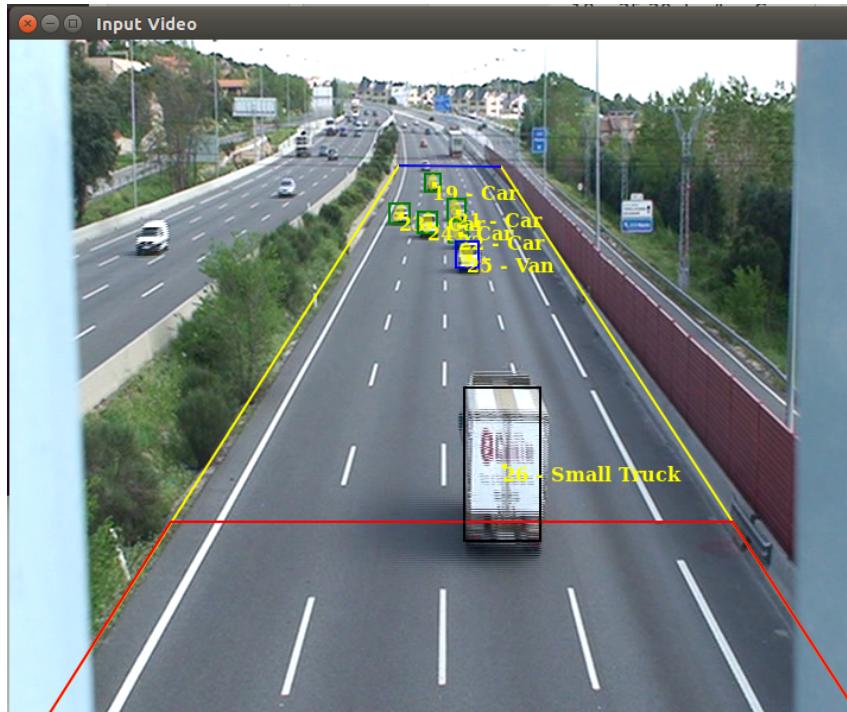


Figura 5.4: Detecciones Smart-Traffic-Sensor Vídeo Buena Calidad

De este vídeo se extrajo un total de 299 imágenes que se componen de las muestras que se indican en la Tabla 5.17.

Nº de Imágenes	299
Nº de Muestras Totales	1297
Nº Car	966
Nº Motorcycle	17
Nº Van	145
Nº Truck	71
Nº Small-Truck	98

Tabla 5.17: Imágenes de Test del Vídeo de Buena Calidad

Los resultados obtenidos con el vídeo de buena calidad se pueden ver en la Tabla 5.18. Un vídeo del funcionamiento de *Smart-Traffic-Sensor* se puede ver en ¹.

¹<https://www.youtube.com/watch?v=s0ozbxsoYmY&feature=youtu.be>

Tipo de Sistema	mAP	mAR
Smart-Traffic-Sensor	0.8926	0.9009
Traffic-Monitor	0.4374	0.5940
Redes Neuronales	0.8316	0.8966

Tabla 5.18: Resultados Vídeo de Buena Calidad

Observando los resultados en esta primera evaluación se puede verificar que el sistema que obtiene mejores resultados es el *Smart-Traffic-Sensor*. El hecho de combinar KLT con las detecciones realizadas mediante *Deep Learning* lo dota de mayor robustez. Esto se hace evidente si nos fijamos en los resultados que se obtienen mediante redes neuronales, con los cuales podemos ver que el hecho de complementarlo con KLT hace que el sistema mejore. No obstante los resultados que se obtienen mediante las redes neuronales son de muy buena calidad, demostrando su gran capacidad a la hora de detectar vehículos. El uso de KLT simplemente lo complementa en casos de occlusiones o vehículos que se vean muy pequeños debido a su lejanía. Por el contrario los resultados que da *Traffic-Monitor* son un poco pobres si los comparamos con los conseguidos gracias a *Smart-Traffic-Sensor*. En las sucesivas pruebas que se han hecho con *Traffic-Monitor* se ha apreciado que no funciona bien con vehículos lejanos (en muchas ocasiones los coches los clasifica como motocicletas) y que en muchas ocasiones tiene dificultad para diferenciar entre coche y furgoneta. Cuando se trata de furgonetas pequeñas las confunde con coches. Esto se debe a que la clasificación se hace mediante modelos 3D, razón por la cual una furgoneta pequeña puede aproximarse más al modelo 3D de un coche que al de una furgoneta grande.

El segundo vídeo que se ha evaluado es un vídeo con condiciones meteorológicas desfavorables. En concreto se trata de un vídeo en condiciones de lluvia. En la Figura 5.5 se puede ver un ejemplo de *Smart-Traffic-Sensor*.

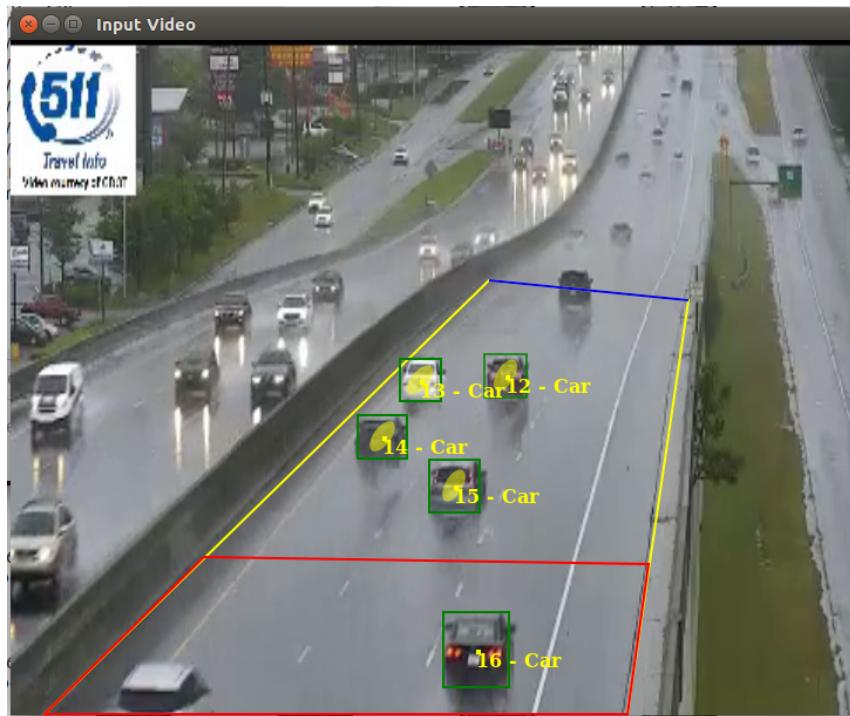


Figura 5.5: Detecciones Smart-Traffic-Sensor Vídeo Malas Condiciones Meteorológicas

Con este vídeo se han obtenido 138 imágenes que se componen tan solo de coches tal y como se indica en la Tabla 5.19. Los vídeos que se obtuvieron en condiciones de lluvia no contenían ningún otro tipo de vehículo.

Nº de Imágenes	138
Nº de Muestras Totales	544
Nº Car	544

Tabla 5.19: Imágenes de Test del Vídeo de Malas Condiciones Meteorológicas

Los resultados obtenidos al evaluar el vídeo de malas condiciones climatológicas se pueden observar en la Tabla 5.20. Podemos ver como se comporta *Smart-Traffic-Sensor* con este vídeo en ².

²https://www.youtube.com/watch?v=YxpfMtxIr_Q&feature=youtu.be

Tipo de Sistema	mAP	mAR
Smart-Traffic-Sensor	0.9899	0.9926
Traffic-Monitor	0.2407	0.3162
Redes Neuronales	0.9659	0.9889

Tabla 5.20: Resultados Vídeo de Malas Condiciones Meteorológicas

Recapitulando todos los resultados de nuevo se observa que *Smart-Traffic-Sensor* es el sistema que mejores resultados obtiene. A pesar de encontrarnos en condiciones de lluvia es capaz de funcionar y con muy buenos resultados. Con esta prueba se puede ver que *Traffic-Monitor* no es robusto ante cambios, pues no es capaz de funcionar correctamente con lluvia. Esto queda indicado en la tesis que describe *Traffic-Monitor* [17], en la cual se aclara que la aplicación funciona únicamente con buenas condiciones.

El tercer vídeo que se empleó para evaluar el sistema se trata de un vídeo con mala calidad. En la Figura 5.6 se muestra un ejemplo de dicho vídeo.

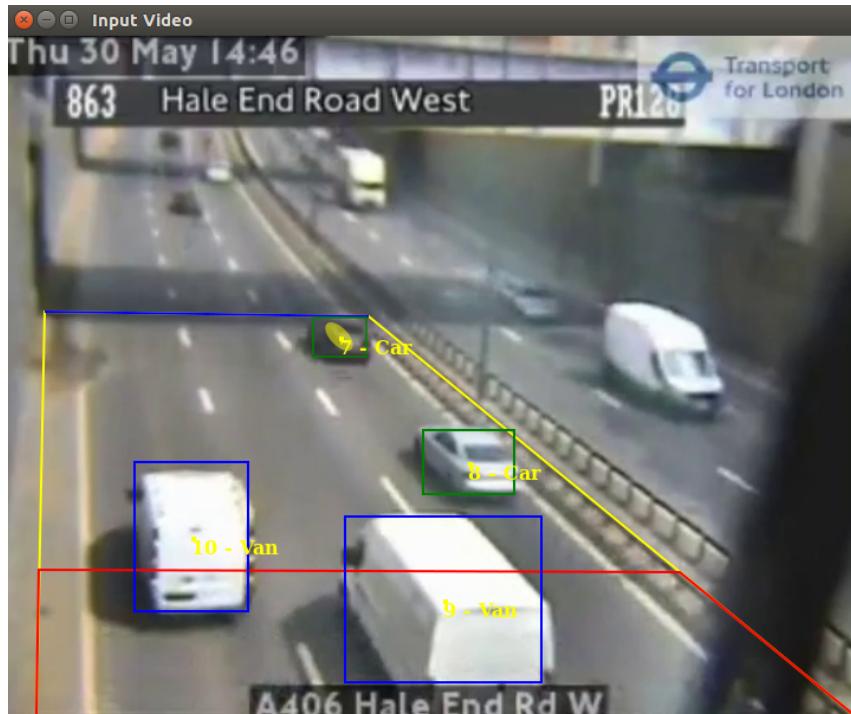


Figura 5.6: Detecciones Smart-Traffic-Sensor Vídeo Mala Calidad

De este vídeo se han obtenido 75 imágenes para poder realizar su evaluación. Estas

CAPÍTULO 5. EXPERIMENTOS

imágenes se componen de coches, motocicletas y furgonetas tal y como se indica en la Tabla 5.21.

Nº de Imágenes	75
Nº de Muestras Totales	109
Nº Car	72
Nº Motorcycle	6
Nº Van	31

Tabla 5.21: Imágenes de Test del Vídeo de Mala Calidad

Los resultados obtenidos al evaluar el vídeo de mala calidad se pueden observar en la Tabla 5.22. Se puede ver el comportamiento de *Smart-Traffic-Sensor* con este vídeo en ³.

Tipo de Sistema	mAP	mAR
Smart-Traffic-Sensor	0.9439	0.9444
Traffic-Monitor	0.4479	0.6303
Redes Neuronales	0.9390	0.9300

Tabla 5.22: Resultados Vídeo de Mala Calidad

Con toda la información recapitulada se puede decir que *Smart-Traffic-Sensor* es robusto ante imágenes de mala calidad y en condiciones meteorológicas malas. Además es capaz de continuar realizando el seguimiento de los vehículos cuando estos se encuentran muy lejos. Evidentemente funciona mejor con vehículos próximos, pues es más sencillo detectarlos, pero aún así es capaz de detectarlos con gran calidad. Si nos fijamos en los resultados que se obtienen en los tres vídeos se puede comprobar que son mejores para vídeos de mala calidad y condiciones meteorológicas desfavorables que en el caso de buena calidad. Esto tiene su explicación, ya que las exigencias que le marcamos a los datos con buena calidad son mayores. Es decir, en los vídeos de mala calidad y condiciones meteorológicas malas no esperamos que el sistema sea capaz de detectar vehículos lejanos, pues ni siquiera es sencillo para un ser humano poder clasificar dichos vehículos. Por ello la zona de evaluación que se marca no engloba vehículos que se encuentren muy lejos.

³<https://www.youtube.com/watch?v=WZLAyreBNyU&feature=youtu.be>

CAPÍTULO 5. EXPERIMENTOS

A parte de esto, da la casualidad que en estos vídeos la cámara se encuentra a menor distancia de los vehículos que en el caso del vídeo de buena calidad. Esto se hace evidente porque los vehículos que entran en la zona de evaluación tienen mayor tamaño que los que se pueden ver en el vídeo de buena calidad.

Otro detalle es que en el vídeo de buena calidad aparecen más clases de vehículos que en los otros casos, en los cuales la mayoría son coches. La base de datos con la que se ha entrenado la red neuronal posee mayor cantidad de coches que del resto de vehículos, es decir se encuentra desbalanceada. Esto nos lleva a que el modelo llegue a aprender mejor la categoría coche que el resto de categorías. No obstante, se ha apreciado que en todas las categorías se obtienen grandes resultados, exceptuando el caso de las motocicletas. Cuando avanzan los vehículos por la carretera se va reduciendo su tamaño, pues se van alejando. Las motocicletas son vehículos de menor tamaño que el resto de categorías, por ello a nada que avancen empezarán a tomar un tamaño muy reducido, haciendo muy difícil su clasificación. Por esta razón los resultados en cuanto a las motocicletas suelen ser peores a no ser que se encuentren próximos a la cámara y por tanto tengan mayores dimensiones.

Para ver como se comporta el sistema frente a las diversas categorías se han obtenido resultados para cada categoría en el vídeo de buena calidad y en el de mala calidad con *Smart-Traffic-Sensor*. En el de condiciones meteorológicas desfavorable no se ha realizado, pues todos los vehículos que aparecían se correspondían con la categoría coche.

En la Tabla 5.23 se pueden ver los resultados que se obtienen con el vídeo de buena calidad y en la Tabla 5.24 se pueden apreciar los resultados que nos da el vídeo de mala calidad.

Tipo de Vehículo	mAP	mAR
Car	0.9457	0.9679
Motorcycle	0.7029	0.7059
Van	0.8809	0.8897
Truck	0.9703	0.9718
Small-Truck	0.9604	0.9694

Tabla 5.23: Resultados para las Diferentes Categorías en Vídeo de Buena Calidad

Tipo de Vehículo	mAP	mAR
Car	1	1
Motorcycle	0.8317	0.8333
Van	1	1

Tabla 5.24: Resultados para las Diferentes Categorías en Vídeo de Mala Calidad

En ambos casos se hace evidente que la categoría motocicleta es la que peores resultados da, debido a su tamaño y por tanto a su complejidad para ser detectada y clasificada.

5.5. Evaluación de los Tiempos de Procesamiento de las Redes

Tal y como se ha indicado todos los experimentos realizados con *DeepLearningSuite* se han hecho desde un ordenador sin GPU. Para saber si estas redes neuronales tienen un tiempo de procesamiento que nos permita trabajar en tiempo real con un ordenador por supuesto con GPU hemos empleado un servidor. Del cual hemos extraido los resultados con el fin de saber que nuestras redes son capaces de trabajar rápidamente.

En la Tabla 5.25 se pueden ver los resultados obtenidos.

Tipo de Red	Tiempo (ms)
Keras	45
TensorFlow	19
Darknet	48

Tabla 5.25: Tiempos de procesamiento

Los resultados que se obtienen respecto a los tiempos de procesamiento son bastante razonables y por supuesto permiten trabajar en tiempo real.

Capítulo 6

Conclusiones

A lo largo de toda la memoria se ha explicado todo el proyecto realizado. Se han abordado todos los pasos que se han llevado a cabo; desde el comienzo en el que se parte de la literatura de otros autores hasta los experimentos sobre nuestro propio sistema. Es decir, se ha dado una visión general de todo lo que se ha hecho para llegar a tener nuestro *Smart-Traffic-Sensor*.

En este capítulo se va a realizar un resumen general del trabajo realizado y las conclusiones a las que se ha llegado. Finalmente se plantearán sus posibles líneas futuras.

6.1. Conclusiones

Este trabajo parte de una versión previa denominada *Traffic-Monitor* [1]. En ella se plantea una solución clásica para la monitorización de vehículos. El objetivo principal de *Smart-Traffic-Sensor* es mejorar los resultados respecto a los que se obtienen con *Traffic-Monitor*. Para ello se plantearon dos objetivos:

- Realizar la monitorización haciendo uso de técnicas más actuales como *Deep Learning*.
- Ofrecer la posibilidad de emplear diferentes *framework* de *Deep Learning* para la detección y clasificación. Es decir, poder emplear redes neuronales entrenadas con diferentes *frameworks* en *Smart-Traffic-Sensor*. En concreto, tener la posibilidad de usar redes entrenadas con *Keras*, *TensorFlow* y *Darknet*.
- Dotar al sistema de mayor robustez ante diversidad de imágenes tales como imágenes con mala calidad o con condiciones meteorológicas adversas.

CAPÍTULO 6. CONCLUSIONES

Ambos objetivos se han tenido en cuenta a lo largo de todo el trabajo, diseñando un sistema capaz de monitorizar vehículos basándonos principalmente en las detecciones que se obtienen mediante redes neuronales y dotándole de la capacidad de detectar vehículos en diferentes tipos de imágenes.

Tras evaluar nuestro trabajo se han podido extraer algunas conclusiones:

- Cuanta más información empleamos en el entrenamiento, mayor versatilidad tendremos.
- El tamaño de los vehículos influye a la hora de realizar las detecciones. Cuanto más pequeños sean los vehículos mayor dificultad habrá para detectarlos. Esta cuestión afecta directamente a las motocicletas, las cuales de por si poseen menor tamaño. Este hecho hace que en cuanto se alejen un poco su tamaño quede muy reducido, haciendo muy difícil su detección.
- Las clases de las que poseemos mayor cantidad de datos en el entrenamiento normalmente obtienen mejores resultados, pues se tiene mayor información acerca de ellas.

A continuación vamos a hacer un repaso de los resultados respecto a la detección de vehículos que hemos encontrado en la diferente literatura publicada. Para ver con ello si los resultados que obtenemos son comparables con los que se están publicando.

Y. Abdullah, G. Mehmet, A. Iman and B. Erkan [4] propusieron dos soluciones. Una con Faster R-CNN y otra con R-CNN. En su artículo indican el mAP obtenido para dos conjuntos de datos tanto con Faster R-CNN como con R-CNN. Esto se puede ver en la Tabla 6.1.

Método	mAP
Faster R-CNN (Dataset 1)	0.728
Faster R-CNN (Dataset 2)	0.757
R-CNN (Dataset 1)	0.647
R-CNN (Dataset 2)	0.657

Tabla 6.1: Resultados de Y. Abdullah, G. Mehmet, A. Iman and B. Erkan [4]

L. Chen, F. Ye, Y. Ruan, H. Fan and Q. Chen [69] usan *k-means* para obtener características de las imágenes y emplearlas durante el entrenamiento. Además concatenan

características de diferentes tamaños de imagen. Para realizar la detección emplean CNN. En la Tabla 6.2 se puede ver el mAP obtenido con su diseño, así como los resultados que ha obtenido con otro tipo de redes neuronales.

Método	mAP
Fast R-CNN	0.672
Faster R-CNN	0.692
YOLO	0.589
SSD300	0.688
SSD512	0.712
Diseño Implementado	0.757

Tabla 6.2: Resultados de Y. Abdullah, G. Mehmet, A. Iman and B. Erkan [4]

Ricardo Guerrero-Gómez-Olmedo, Roberto López-Sastre, Saturnino Maldonado-Bascón and Antonio Fernández-Caballero [49] plantean el uso de filtros de Kalman para realizar el seguimiento y descriptores HOG para la detección y clasificación. En su artículo dicen que la máxima precisión que obtuvieron era de 0.4872.

Albert Soto [70] propone el uso de YOLO para la detección de vehículos y llega a obtener una precisión de 0.5893 y un recall de 0.4092.

Viendo todos estos resultados se puede observar que *Smart-Traffic-Sensor* consigue superarlos. Con ello podemos afirmar que hemos llegado a obtener un sistema muy robusto y que aparentemente su calidad es suficientemente buena. Además hemos cumplido con creces el objetivo de mejorar los resultados que ofrecía *Traffic-Monitor*.

6.2. Trabajos Futuros

En esta sección se van a comentar posibles líneas futuras de este proyecto.

La base de datos que se ha empleado tan solo tiene vehículos que se ven por su parte trasera. Por tanto todo el trabajo realizado solo se ha centrado en secuencias de vídeos en las que los vehículos se ven por su parte trasera. Esto podría extenderse a diversas posiciones de los vehículos, desde vehículos que se vean por la parte frontal hasta vehículos que se vean de forma lateral. Con ello se conseguiría enriquecer la base de datos y por

CAPÍTULO 6. CONCLUSIONES

tanto obtener un modelo mucho más versatil que funcionará con cualquier disposición de los vehículos.

En este caso se han incluido imágenes con condiciones meteorológicas malas, pero solo se han incluido imágenes de lluvia y niebla, pues no se consiguió obtener una base de datos con otras condiciones. Se deberían incluir imágenes con mayores condiciones de niebla, con nieve , con borrascas, etc. Es decir, condiciones mucho más adversas para ver si es capaz de seguir comportándose bien.

Otra línea posible es dar el paso hacia la detección de vehículos en condiciones nocturnas, aunque en esta cuestión sería necesario el uso de cámaras infrarrojas.

En cuanto a las redes neuronales empleadas, hubiera sido muy interesante poder realizar una red propia, con la que poder hacer las detecciones y clasificaciones de los vehículos.

Finalmente se puede decir que por supuesto el sistema se podría mejorar en cuanto a los resultados obtenidos y ampliarlo incluyendo información acerca de lo que ocurre en la carretera, es decir, se podría detectar accidentes o comportamientos sospechosos.

Bibliografía

- [1] Redouane Kachach. Traffic-Monitor. <https://github.com/JdeRobot/traffic-monitor>. [Accedido 22 de Junio de 2019].
- [2] D. Koller, J. Weber and J. Malik. Robust multiple car tracking with occlusion reasoning. *University of California at Berkeley 1:188-196*, 1994.
- [3] Zezhi Chen and Tim Ellis. Multi-shape descriptor vehicle classification for urban traffic. *International Conference on Digital Image Computing: Techniques and Applications*, pages 465–461, 2011.
- [4] Y. Abdullah, G. Mehmet, A. Iman and B. Erkan. A Vehicle Detection Approach using Deep Learning Methodologies. *Computer Engineering Department Ankara University*, 2018.
- [5] M. Onoe and K. Ohba . Digital analysis of traffic flow. In *Proc Int Joint ConfPattern Recognition*, pages 803–804, 1976.
- [6] E.E. Hilbert, C. Carl, W. Goss, G.R. Hansen and M.J. Olsasky. Wide area detection system. *Report - Federal Highway Administration. FHWA-RD-77-86*, 1978.
- [7] N. Hoose. Queue detection using computer image processing.Road TrafficMonitoring. *Second International Conference on*, pages 94–98, 1989.
- [8] J.M Blosseville, C. Krafft, F. Lenior, V. Motyka and S. Beucher. New traffic measurement by image processing. *IFAC Control, Computers, Communicationsin Transportation*, pages 35–42, 1989.
- [9] J. Versavel, F. Lemaire and D. Van der Stede. Camera and computer-aided traffic sensor.Road Traffic Monitoring. *Second International Conference on*, pages 66–70, 1989.
- [10] Baker, K.D Sullivan and G.D. Performance assessment of model-based nacking. In *Applications of Computer Vision, Proceedings*, pages 28–35. IEEE Workshop on , 1992.

CAPÍTULO 6. CONCLUSIONES

- [11] G.D. Sullivan, K.D. Baker, A.D. Worral, C.I. Attwood and P.M. Remagnino. Model-based vehicle detection and classification using orthographic approximations. *IVC 15:649-654* , 1997.
- [12] B. Coifman, D. Beymer, P. McLauchlan and J. Malik. Areal-time computer vision system for vehicle tracking and traffic surveillance. *Institute of Transportation Studies, University of California, Berkeley* , 1998.
- [13] E. Vermeulen. Automatic incident detection (AID) with thermal cameras. *FLIR Intelligent Transportation System* , 2014.
- [14] Y. Chang, Z. Su and L. Qian-Yu. A new traffic incident detection method under low-volume condition based on automatic vehicle identification. In *Fuzzy Systems and Knowledge Discovery (FSKD), 9th International Conference on*, pages 2853–2859, 2012.
- [15] E. Nateghinia and H. Moradi. Video-based multiple vehicle tracking at intersections. In *Robotics and Mechatronics (ICRoM), Second RSI/ISM International Conference on*, 2014.
- [16] M.S. Shirazi and B. T. Morris. Vision-based turning movement monitoring: count, speed amp; waiting time estimation. *IEEE Intelligent Transpor-tation Systems Magazine 8(1):23–34*, 2016.
- [17] Redouane Kachach. *Monitorización visual automática de tráfico rodado*. PhD thesis, Universidad de Alicante. Instituto Universitario de Investigación Informática, 2016. [Accedido 20 de Mayo de 2019].
- [18] S.I. Arroyo, F. Safar and D. Oliva. Probabilidad de infracción de velocidad de vehículos utilizando visión artificial en cámaras de campo amplio. *IEEE Biennial Congress of Argentina (ARGENCON)*, 2016.
- [19] A.F. Granados and J. I. Marin .H . Detección de flujo vehicular basado en visión artificial. *Universidad Tecnológica de Pereira*, 2007.
- [20] J. Portillo, G. Sánchez, J. Olivares and H. Pérez. Detección de Movimiento de Vehículos en Secuencias de Video Basados en la Diferencia Absoluta entre Fotogramas y la Combinación de Bordes. *Instituto Politécnico Nacional, Sección de Estudios*

de Posgrado e Investigación, Escuela Superior de Ingeniería Mecánica y Eléctrica Unidad Culhuacán, 2013.

- [21] C. Stauffer and W.E. Grimson . Adaptive background mixture models for real-time tracking. *IEEE ICCV 2: 256–261*, 1999.
- [22] Zoran Zivkovic. Improved adaptive gaussian mixture model for background subtraction. *Pattern Recognition. Proceedings of the 17th International Conference on*, 2:28–31, 2004.
- [23] P. Barcellos, C.Bouvié, F.L.Escouto and J. Scharcanski . A novel video based system for detecting and counting vehicles at user-defined virtual loops. *Expert Systems with Applications*, 42(4):1845–1856, 2014.
- [24] J. W. Davis and A.F. Bobick . The representation and recognition of human movement using temporal templates. In *IEEE computer society conference on computer vision and pattern recognition*, pages 928–934, 1997.
- [25] Y. Xia, X. Shi, G.Song, Q. Geng and Y. Liu . Towards improving quality of video-based vehicle counting method for traffic flow estimation. *SignalProcess*, 120(C):672–681, 2014.
- [26] N. Buch, J. Orwell and S. Velastin . 3D Extended Histogram of Oriented Gradients (3DHOG) for Classification of Road Users in Urban Scenes. In *BMVC.British Machine Vision Association.*, 2009.
- [27] Bjorn Johansson, Johan Wiklund, Per-Erik Forssén and Gösta Granlund. Combining shadow detection and simulation for estimation of vehicle size andposition. *Pattern Recognition Letters*, 2009.
- [28] John Wood. Statistical Background Models with Shadow Detection for Video Based Tracking. *Linköping University, Department of Electrical Engineering, Computer Vision. Linköping University, The Institute of Technology*, 2007.
- [29] K. Kim, T.H. Chalidabhongse, D. Harwood and L. Davis. Real-time foreground–background segmentation using codebook model. *Elsevier, Real time imaging*, page 172–185, 2005.

- [30] M. Mazaheri and S. Mozaffari. Real time adaptive background estimation and road segmentation for vehicle classification. *IEEE*, page 1–6, 2011.
- [31] A. Elgammal, D. Harwood and L. Davis. Non-parametric model for background subtraction. *Proc. Eur. Conf. Comput. Vis. Part II, LNCS 1843:751–767*, 2000.
- [32] Rigoberto Vizcay. *Deep Learning para la Detección de Peatones y Vehículos sobre FPGA*. PhD thesis, Centro Universitario UAEM Valle de México, 2018. [Accedido 22 de Junio de 2019].
- [33] Ignacio Arriola. *Detección de objetos basada en Deep Learning y aplicada a vehículos autónomos*. PhD thesis, Universidad del País Vasco. Ingeniería computacional y sistemas inteligentes, 2018.
- [34] Shih-Hao Yu, Yung-Sheng Chen, Wen-Fong Hu and Jun-Wei Hsieh. An automatic traffic surveillance system for vehicle tracking and classification. *Lecture Notes in Computer Science Volume 2749:379–386*, 2003.
- [35] Jin-Cyuan Lai, Shih-Shinh Huang and Chien-Cheng Tseng . Image-based vehicle tracking and classification on the highway. *Green Circuits and Systems (ICGCS), 2010 International Conference on*, pages 666–670, 2010.
- [36] H. Asaidi , A. Aarab and M. Bellouki. Shadow elimination and vehicles classification approaches in traffic video surveillance context. *Elsevier*, 2014.
- [37] C.P. Papageorgiou, M. Oren and T. Poggio. A General Framework for Object Detection. In *Proceedings of the Sixth International Conference on Computer Vision, ICCV '98*, pages 555–562, 1998.
- [38] Navneet Dalal and Bill Triggs. Histograms of Oriented Gradients for Human Detection. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference, 1:886–893*, 2005.
- [39] Bailing Zhang, Yifan Zhou and Hao PanTammam Tillo. Hybrid model of clustering and kernel autoassociator for reliable vehicle type classification. *Machine Vision and Applications*, pages 437–450, 2013.
- [40] L. Sirovich and M. Kirby. Low-dimensional procedure for the characterization of human faces. *Journal of the Optical Society of America, 4:519–524*, 1987.

CAPÍTULO 6. CONCLUSIONES

- [41] Wei Wang, Yulong Shang, Jinzhi Guo and Zhiwei Qian. Real-time vehicle classification based on eigenface. *Consumer Electronics, Communications and Networks(CECNet)*:4292–4295, 2011.
- [42] Wook-Sun Shin, Doo-Heon Song and Chang-Hun Lee. Vehicle classification by road lane detection and model fitting using a surveillance camera. *Journal of Information Processing Systems*, 2009.
- [43] J.R. Quinlan. C4.5: Programs for machine learning. *Morgan Kaufmann Publishers, Inc.*, 1993.
- [44] D. Comaniciu, V. Ramesh and P. Meer. Kernel-based object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):564–575, 2003.
- [45] Stefan Duffner and Christophe Garcia. Pixeltrack: A fast adaptive algorithm for tracking non-rigid objects. *Elsevier, Real time imaging*, pages 2480–2487, 2013.
- [46] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91-110, 2004.
- [47] J. Shi and C. Tomasi. Good features to track. *Proc. IEEE CVPR*, pages 593–600, 1994.
- [48] M.J. Leotta and J.L. Mundy. Vehicle surveillance with a generic, adaptative, 3d vehicle model. *IEEE Transactions on pattern analysis and machine intelligence*, 33:1457-1469, 2011.
- [49] Saturnino Maldonado-Bascón Ricardo Guerrero-Gómez-Olmedo, Roberto López-Sastre and Antonio Fernández-Caballero. Vehicle Tracking by Simultaneous Detection and Viewpoint Estimation. <http://agamenon.tsc.uah.es/Personales/rlopez/docs/iwinac13-guerrero.pdf>, 2013. [Accedido 22 de Junio de 2019].
- [50] GRAM Road-Traffic Monitoring. Dataset. <http://agamenon.tsc.uah.es/Personales/rlopez/data/rtm/>, 2013. [Accedido 22 de Junio de 2019].
- [51] BIT-Vehicle. Dataset. <http://iitlab.bit.edu.cn/mcislab/vehicledb/>, 2015. [Accedido 22 de Junio de 2019].

- [52] CarND-Vehicle-Detection. Dataset. <https://github.com/udacity/CarND-Vehicle-Detection>, 2017. [Accedido 22 de Junio de 2019].
- [53] CarND-Vehicle-Detection. Dataset. <https://github.com/udacity/self-driving-car/tree/master/annotations>, 2017. [Accedido 22 de Junio de 2019].
- [54] Vinay Sharma. DeepLearningSuite. <https://github.com/JdeRobot/DetectionSuite>, 2019. [Accedido 31 de Mayo de 2019].
- [55] Tensorflow. <https://github.com/tensorflow/tensorflow>. [Accedido 22 de Junio de 2019].
- [56] Keras: Deep Learning for humans. <https://github.com/keras-team/keras>. [Accedido 22 de Junio de 2019].
- [57] Darknet. <https://github.com/pjreddie/darknet>. [Accedido 22 de Junio de 2019].
- [58] Redouane Kachach. Traffic-Monitor-Lab. <https://github.com/JdeRobot/smart-traffic-sensor-lab>. [Accedido 22 de Junio de 2019].
- [59] R. Guerrero-Gomez-Olmedo, R. J. Lopez-Sastre, S. Maldonado-Bascon, and A. Fernandez-Caballero. Vehicle tracking by simultaneous detection and viewpoint estimation. In *IWINAC 2013, Part II, LNCS 7931*, pages 306–316, 2013.
- [60] TensorFlow Models. <https://github.com/tensorflow/models>. [Accedido 22 de Junio de 2019].
- [61] SSD Mobilenet V2 COCO Config. https://github.com/tensorflow/models/blob/master/research/object_detection/samples/configs/ssd_mobilenet_v2_coco.config. [Accedido 22 de Junio de 2019].
- [62] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu and Alexander C. Berg. SSD: Single Shot MultiBox Detector. *Computer Vision – ECCV*, pages 21–37, 2016.
- [63] Shaoqing Ren, Kaiming He, Ross Girshick, Sun. Jian . Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1137 – 1149, 2016.

CAPÍTULO 6. CONCLUSIONES

- [64] Pierluigi Ferrari. SSD: Single-Shot MultiBox Detector implementation in Keras. https://github.com/pierluigiferrari/ssd_keras. [Accedido 22 de Junio de 2019].
- [65] Joseph Redmon, Santosh Divvala, Ross Girshick and Ali Farhadi . You Only Look Once:Unified, Real-Time Object Detection. *University of Washington*, 2016.
- [66] Jean- Yves Bouguet. Pyramidal Implementation of the Lukas Kanade Feature tracker. *Intel Corporation, Micriprocessor Research Labs*, 2000.
- [67] Arvind Jayaraman. Vehicle Detection Project. <https://github.com/ajayaraman/CarND-VehicleDetection>. [Accedido 22 de Junio de 2019].
- [68] Darren. LabelImg. <https://github.com/tzutalin/labelImg>. [Accedido 22 de Junio de 2019].
- [69] L. Chen, F. Ye, Y. Ruan, H. Fan and Q. Chen. An algorithm for highway vehicle detection based on convolutional neural network. *Chen et al. EURASIP Journal on Image and Video Processing*, 2018.
- [70] Albert Soto. YOLO Object Detector for Onboard Driving Images. *Universitat Autónoma de Barcelona*, 2017.
- [71] Tesla. https://www.tesla.com/es_ES/autopilot, 2019. [Accedido 19 de Junio de 2019].
- [72] Tesla pone coto al piloto automático en España. https://www.economiadigital.es/tecnologia-y-tendencias/tesla-pone-coto-al-piloto-automatico-en-espana_625493_102.html/, 2019. [Accedido 19 de Junio de 2019].
- [73] Raúl Álvarez. El coche autónomo de Google (Waymo) se vuelve completamente autónomo y por primera vez sale a la calle sin conductor. <https://www.xataka.com/automovil/el-coche-autonomo-de-google-waymo-se-vuelve-completamente-autonomo-y-por-primeravez>, 2017. [Accedido 19 de Junio de 2019].
- [74] Mesta Fusion: así funciona y multa el súper radar del futuro. <https://www.autopista.es/radares-dgt/articulo/>

CAPÍTULO 6. CONCLUSIONES

- mesta-fusion-asi-multa-funciona-video, 2018. [Accedido 20 de Junio de 2019].
- [75] El radar que lo caza todo. <https://www.elmundo.es/motor/2016/04/26/571e535f468aeb301c8b45b0.html>, 2016. [Accedido 20 de Junio de 2019].
- [76] Gonzalo Gasca. Precisión y recuperación (Precision and recall). https://medium.com/@gogasca_95055/precisi%C3%B3n-y-recuperaci%C3%B3n-precision-recall-dc3c92178d5b, 2018. [Accedido 22 de Junio de 2019].
- [77] Rafael Padilla. Metrics for object detection. <https://github.com/rafaelpadilla/Object-Detection-Metrics>, 2019. [Accedido 22 de Junio de 2019].
- [78] Julián Blasco. *Técnicas de estadística computacional para visión por computador*. PhD thesis, Universidad Nacional de Educación a Distancia, 2018. [Accedido 22 de Junio de 2019].
- [79] Sundar Pichai. Expertos en Tensorflow. <https://www.paradigmadigital.com/lineas-servicio/tensorflow/>. [Accedido 22 de Junio de 2019].
- [80] Deep Learning básico con Keras (Parte 1). <https://enmilocalfunciona.io/deep-learning-basico-con-keras-parte-1/>, 2018. [Accedido 22 de Junio de 2019].
- [81] Jessica Fernández Martínez. *Aplicación para la mejora motora y cognitiva para personas con discapacidad mediante juegos simples interactivos*. PhD thesis, Universidad de Alcalá. Escuela Politécnica Superior, 2015. [Accedido 22 de Junio de 2019].
- [82] Python Software Foundation. Tutorial de Python. <http://docs.python.org.ar/tutorial/3/real-index.html>, 2017. [Accedido 22 de Junio de 2019].
- [83] GNOME. GNOME Developer Center. <https://developer.gnome.org/>. [Accedido 22 de Junio de 2019].
- [84] S. Avendaño and G. Castillo. *Detección y Reconocimiento de las Señas del Juego del Truco en Tiempo Real*. PhD thesis, Facultad de Ciencias Exactas y Naturales Universidad de Buenos Aires, 2012. [Accedido 22 de Junio de 2019].

CAPÍTULO 6. CONCLUSIONES

- [85] Algoritmo EM. <https://ccc.inaoep.mx/~jagonzalez/ML/principal/node81.html>, 2011. [Accedido 22 de Junio de 2019].
- [86] Lili Huang and M. Barth. Real-time multi-vehicle tracking based on feature detection and color probability model. *Intelligent Vehicles Symposium (IV)*, pages 981–986, 2010.
- [87] Vicente Rodriguez. Machine learning in healthcare parte 2. <https://vincentblog.xyz/posts/machine-learning-in-healthcare-parte-2>. [Accedido 22 de Junio de 2019].