Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza
1542

Trabajo Fin de Grado

Sistemas de aprendizaje automático eficientes para reconocimiento visual en sistemas embebidos

Efficient machine learning systems for visual recognition in embedded systems

Autora

Pilar Salvo Ibáñez

Directora

Ana Cristina Murillo Arnal

Ingeniería Electrónica y Automática
Departamento de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

Agosto 2019

Resumen

En la actualidad encontramos sistemas embebidos en todas partes, en el sistema de frenos ABS, en un proceso de montaje o producción, en los hospitales en equipos de medicina, incluso en radares y en cámaras de vídeo. Un sistema embebido es un sistema electrónico computacional cuyo hardware y software están diseñados para realizar una o varias funciones específicas, en vez de ocuparse de varias tareas generales a la vez, centradas en llevar a cabo de forma eficiente la aplicación final para la que fue creado.

El objetivo de este proyecto es realizar un estudio de algoritmos de *deep learning* para un reconocimiento visual eficiente para sistemas de vigilancia y monitorización. La idea es que puedan funcionar en tarjetas gráficas específicas para dispositivos embebidos, buscando conseguir que desde un hardware embebido se sea capaz de detectar objetos de interés, como pueden ser principalmente personas y distintos tipos de vehículos.

En el arranque del proyecto se ha seleccionado el algoritmo de YOLO, el cual es destacado por su rapidez en la detección de objetos tanto en imágenes como en vídeos a tiempo real. Se ha estudiado el funcionamiento de este, y se ha realizado la adaptación del modelo para los datos y las clases elegidas. En cuanto a los datos a utilizar para entrenar el modelo, se ha seleccionado el *dataset* público VIRAT. Este contiene 29 horas de vídeos de vigilancia, en las que se muestran escenas reales de la calle con distintos eventos de personas y vehículos. Al ser vídeos de cámaras de vigilancia reales, la visualización de las imágenes no es frontal como suele ocurrir en la mayoría de los *dataset*, sino que tiene un enfoque elevado que se busca para que el modelo funcione de manera correcta en cualquier tipo de sistema de seguridad, desde cámaras en los edificios a cámaras llevadas por drones. Las clases en las que se llevara a cabo la detección de objetos en estos vídeos son cinco: personas, coches, bicicletas, vehículos y objetos.

Se han realizado varias pruebas entrenando distintos modelos de YOLO v3, utilizando distintos conjuntos de entrenamiento. Se ha evaluado cada modelo respecto a su precisión en la detección de las distintas clases y se observa que en las clases de personas y coches el modelo propuesto obtiene una detección mejor, mientras que los elementos más pequeños como objetos y bicicletas los reconoce peor. Además, se ha analizado el tiempo de ejecución y el consumo al evaluar una imagen con el modelo creado.

Por último, el modelo que mejor rendimiento obtiene se ha instalado en el dispositivo embebido elegido, la Nvidia Jetson AGX Xavier. Esta plataforma, con su diseño embebido y su consumo eficiente, resulta muy competente para implementar algoritmos de *deep learning* en sistemas embebidos como el prototipo de vigilancia que se ha construido en este proyecto. Con este prototipo se han realizado pruebas en vivo adicionales donde se ha observado el buen funcionamiento en escenarios reales alrededor del Instituto de Ingeniería e Investigación de Aragón.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Motivación

En el ser humano uno de los órganos más complejos es el cerebro, es gracias a sus millones de neuronas que las personas son capaces de pensar, razonar y de resolver problemas de distinta complejidad.

En los últimos años se ha avanzado mucho en el campo del aprendizaje automático donde, inspirados por las capacidades humanas, se quiere dar a un sistema la capacidad para interpretar datos, aprender de ellos y lograr resolver tareas y problemas a partir de este aprendizaje. Este tipo de técnicas de aprendizaje automático se utilizan en numerosos ejemplos de nuestro día a día, desde navegar por internet, los asistentes de los teléfonos móviles o en las redes sociales analizando fotografías.

Con la inspiración del cerebro humano, se propusieron las redes neuronales [1], una técnica de aprendizaje automático que imita de forma simplificada las funciones de las neuronas en el cerebro. Trabajando con las neuronas ordenadas en capas, la información se va procesando mediante operaciones en las distintas capas, y en la ultima de ellas se obtiene como salida el resultado final. Conforme más capas intermedias tenga la red, y mas neuronas, el resultado obtenido suele ser mejor, pero también la red sera más compleja y por lo tanto será más difícil entrenar y obtener un modelo adecuado, debido a la cantidad creciente de parámetros y de datos de entrenamiento requeridos.

Este tipo de redes neuronales con muchas capas intermedias se conocen como *deep neural networks*, o redes neuronales profundas [2]. Estas técnicas han tenido grandes avances en los últimos años, obteniendo resultados impresionantes en numerosos campos como la visión por computador, el procesamiento de lenguaje natural o de música entre otros.

Sin embargo, en general los sistemas de *deep learning* aún conllevan cargas computacionales muy altas. Esto hace que en muchos casos, por ejemplo en algunos sistemas de tiempo real, los resultados tarden demasiado tiempo en obtenerse y no resulten adecuados. Los sistemas embebidos son un ejemplo claro donde sistemas de *deep learning* más eficientes facilitarían aplicaciones más realistas. Esto es debido a que un sistema embebido está diseñado para realizar una o unas pocas funciones específicas pero con unas restricciones de

Figura 1.1: Cámara de vigilancia implementada en un dron.

tiempo limitadas que vienen dadas de manera estricta por la aplicación final.

Aunque el nombre de sistemas embebidos no se escucha mucho en la sociedad y parece algo inusual, la verdad es se encuentran en todas partes, pueden ser sistemas de frenos ABS, sistemas de control de accesos, una impresora, un router o un sistema de cámaras de vigilancia, como la de la Figura 1.1 llevada por un dron. Debido a sus numerosas aplicaciones, resulta de gran interés mejorar las capacidades de los mismos.

En este proyecto, se busca estudiar y diseñar un sistema de reconocimiento visual adecuado para sistemas embebidos. En particular pensando en su aplicación para sistemas de vigilancia (fijos o móviles), donde se puedan conseguir mejoras en el procesado automático de datos de vigilancia y monitorización.

1.2. Objetivos y planificación

El principal objetivo de este trabajo, es el estudio, evaluación y adaptación de algoritmos de *deep learning* para reconocimiento visual eficiente en un ámbito de vigilancia y monitorización de una escena. Para llevarlo a cabo se ha decidido usar una implementación en Keras del algoritmo YOLOv3 [3], por ser un modelo eficiente y rápido entre aquellos que nos encontramos en la literatura de reconocimiento visual automático, para realizar los análisis de objetos de interés y adaptarlo al objetivo.

Las tareas más detalladas que se han abordado para la realización del trabajo son:

- Tarea 1: Estudio del modelo de *deep learning* YOLO. Primero se trabajara con Tensorflow y Teras, para comprender su funcionamiento básico. A continuación se estudiaran las bases de YOLOv3, como reconocer los objetos, cuantas capas tiene el modelo, que datos utiliza... Para luego poder entender y manejar la implementación en Keras de YOLOv3 [1], que es el algoritmo con el que se va a realizar finalmente el proyecto.
- Tarea 2: Selección de datos realistas para la monitorización. Se llevara a cabo un análisis de varios *dataset*, para encontrar uno centrado en cámaras de vigilancia y a que tenga imágenes o vídeos tomados con un ángulo ligeramente picado, para que el modelo se pueda utilizar de manera eficiente tanto si es implementado en robots terrestres como en drones.

[1] `https://github.com/qqwweee/keras-yolo3`

- Tarea 3: Evaluación y adaptación de dicho modelo para el reconocimiento de elementos centrados en sistemas de vigilancia. Con el dataset seleccionado, se pone en prueba el algoritmo para evaluarlo y proceder las modificaciones y experimentos necesarios para intentar obtener resultados más eficientes.
- Tarea 4: Instalación y puesta en marcha del entorno de reconocimiento en el sistema embebido Nvidia-Jetson. Probando la plataforma e instalando los programas necesarios para poder trabajar en ella con el modelo finalmente obtenido.
- Tarea 5: Evaluación del sistema en ese entorno y realización de las modificaciones necesarias del modelo para elementos en casos reales en dicho dispositivo. Comparando el modelo en las dos o más diferentes plataformas en las que se ha implementado y sacando las conclusiones pertinentes.
- Tarea 6: Desarrollo de la documentación, incluyendo la memoria del proyecto y un manual de como utilizar el prototipo. Escribiendo paso a paso las tareas realizadas en el trabajo, las evaluaciones llevadas a cabo y las conclusiones a las que se han llegado.

La organización temporal de estas tareas se puede ver de manera aproximada en la Figura 1.2. En la izquierda de la imagen se pueden ver las tareas a realizar y el día de inicio y de finalización de cada una de ellas. En la derecha se observa el diagrama de Gantt resultante.

1.3. Entorno del trabajo

Este trabajo se ha llevado a cabo en el laboratorio del grupo *Robotics, Perception and Real Time* (RoPeRT), en el Instituto de Ingeniería e Investigación de Aragón (I3A), con la guía de la directora del proyecto y los compañeros del grupo del laboratorio.

También se ha contado con la ayuda de la cátedra Mobility City, creada por la Universidad de Zaragoza con la Fundación Ibercaja. La cual fomenta la investigación y el estudio de Tecnologías de la Información y de las Comunicaciones sobretodo en el campo de las comunicaciones móviles, y en caso de este proyecto en relación con la movilidad y modelado del comportamiento social de agentes implicados en el transporte inteligente y en la movilidad sostenible.

El proyecto se ha desarrollado utilizando el entorno de programación típico de Keras-Tensorflow con python, tanto en un ordenador con Linux versión 16.04, como en una placa Jetson AGX Xavier detallada más adelante en el capítulo 3.

1.4. Trabajos relacionados

Los principales temas relacionados con este trabajo son los resultados recientes en sistemas de reconocimiento visual automático y los trabajos relativos a esfuerzos para conseguir sistemas embebidos cada vez más capaces pero a su vez más eficientes. A continuación se comentan brevemente trabajos relacionados en estos aspectos.

Reconocimiento visual. Los avances de este campo en la actualidad son muy notables, sobretodo gracias al desarrollo de nuevas técnicas y algoritmos basados en *deep learning*, llegando incluso a obtener sistemas más precisos detectando y clasificando imágenes que las propias personas [4]

Figura 1.3: Ejemplos de reconocimiento visual con el algoritmo de *deep learning* YOLO v3.

Los autores del dataset ImageNet [5] llevan años lanzando un desafío donde cantidad de instituciones investigan e intentan mejorar las técnicas para reconocimiento visual a gran escala [6]. Especialmente relacionados con este trabajo son los métodos que se centran en obtener una respuesta más rápido, de manera más eficiente, como pueden ser los algoritmos de YOLO [3], que se especializa en la detección de objetos a tiempo real con el uso de una única red neuronal convolucional, o de DSSD513 [7], un detector de disparo único creado a partir de un clasificador y un marco de detección rápida con capas de deconvolución, de manera que consigue un aumento de precisión, sobretodo en objetos pequeños. Como se detalla en la siguiente sección, este proyecto se basa en la arquitectura de YOLO.

También son de especial relevancia para este trabajo las investigaciones centradas en imágenes y vídeos de vigilancia, que intentan ayudar a que el área de la seguridad este más automatizada. Como este estudio del movimiento de objetos y sus comportamientos [8] que intenta desarrollar una vigilancia visual inteligente que pueda reemplazar la tradicional videovigilancia, haciendo más eficaz el monitorizar imágenes y vídeos, detectando objetos y personas y reconociendo sus comportamientos. Otro proyecto con un propósito más especifico es el reconocimiento de comportamiento anormal para sistemas inteligentes de videovigilancia [9], que busca una manera de garantizar una mayor seguridad, detectando patrones y modelos en el comportamiento de las personas, a través de vídeos de vigilancia, para hallar conductas que se salgan fuera de lo habitual. La Figura 1.3 muestra dos ejemplos típicos de reconocimiento visual identificando los elementos de las imágenes con el algoritmo YOLO.

Sistemas embebidos eficientes. En este proyecto se quiere trabajar con sistemas embebidos, en particular estudiando que problemas existen o que adaptaciones se puede realizar al ejecutar algoritmos de *deep learning* dentro de estos. Como se ha mencionado anteriormente en la motivación del trabajo, es esencial conseguir sistemas embebidos eficientes en todos los aspectos, para poder ser utilizados en nuevas aplicaciones o mejorar las existentes, cumpliendo las

restricciones de las mismas. La Figura 1.4 muestra algunas de las aplicaciones realizadas por sistemas embebidos.

Si miramos investigaciones que se llevan acabo con estos sistemas, se encuentran desde, idear nuevos lenguajes de programación que permitan diseñar nuevas aplicaciones y algoritmos como nesC [10], a retos donde buscan mejorar el diseño de la memoria [11] o de la seguridad y la duración de las baterías de los sistemas [12] de manera que sean más fiables, o incluso la exploración de nuevas maneras de alimentar la energía del sistema integrado que no sea el habitual uso de la batería [13].

Figura 1.4: Posibles aplicaciones de un sistema embebido

Lo más cercano a este proyecto son los trabajos que se centran en algoritmos de reconocimiento visual eficientes. Trabajos más tradicionales como [14] o [8], que se centran en el reconocimiento de imágenes dirigido a la vigilancia y la seguridad. En el caso de [8], detectando vehículos y personas basándose en el movimiento de estos a través de *Background subtraction*, que compara las imágenes que va procesando con otra imagen de fondo para poder encontrar los elementos que antes no estaban ahí, y de *Optical flow*, el cual utiliza vectores de flujo para encontrar patrones en el movimiento de objetos causados por el movimiento relativo entre la cámara y la escena. O como [14], simplemente usando distintos algoritmos que mediante mapas de bits, modelos de color y filtros que segmentan el cuerpo humano para proceder a su detección. También trabajos más recientes basados en sistemas de *deep learning* optimizados y eficientes para poder ejecutarse con pocos recursos computacionales. Tanto para desarrollar una segmentación semántica para vehículos inteligentes, como se hace en ERF-Net: ConvNet factorizada residual eficiente para la segmentación semántica en tiempo real [15], que con una arquitectura basada en una capa con conexiones residuales, que facilita el aprendizaje y reduce el problema de degradación en arquitecturas con gran cantidad de capas, y convoluciones factorizadas, se ejecuta en tiempo real y mantiene la precisión. Como para conseguir redes neuronales convolucionales eficientes para aplicaciones de visión móvil [16], gracias a las redes neuronales profundas e hiperparámetros que permiten elegir el modelo del tamaño correcto para compensar recursos y precisión.

1.5. Organización de la memoria

En este capítulo introductorio se han descrito las motivaciones y los objetivos de este trabajo, mostrando los pasos a seguir para su correcto desarrollo. En el capítulo 2 se enseña el estudio del entono de desarrollo y el análisis de alternativas realizado para la elección del algoritmo de *deep learning* y del *dataset* que se han considerado mejores para cumplir los objetivos del proyecto. En el capítulo 3 el análisis de alternativas para el sistema embebido y se explica el pequeño prototipo llevado a cabo con elegido. En el capítulo 4 se presentan los diferentes modelos con los que se ha buscado mejorar la eficacia y precisión del modelo y la evaluación y experimentación que se ha llevado a cabo para analizar los resultados obtenidos. En el capítulo 5 se exponen las conclusiones extraídas a lo largo del trabajo.

También se incluyen dos anexos. El anexo A contiene imágenes del *dataset* utilizado en el proyecto. Tiene una primera sección donde hay imágenes de los 11 escenarios con los que se trabaja, una segunda sección con un conjunto de diferentes imágenes de las clases observadas en la detección y en la última parte se muestran imágenes de los vídeos propios grabados en Zaragoza para realizar mas pruebas en los modelos. El anexo B, contiene los resultados obtenidos con los conjuntos de datos de validación de los vídeos analizados individualmente en el capitulo 4.

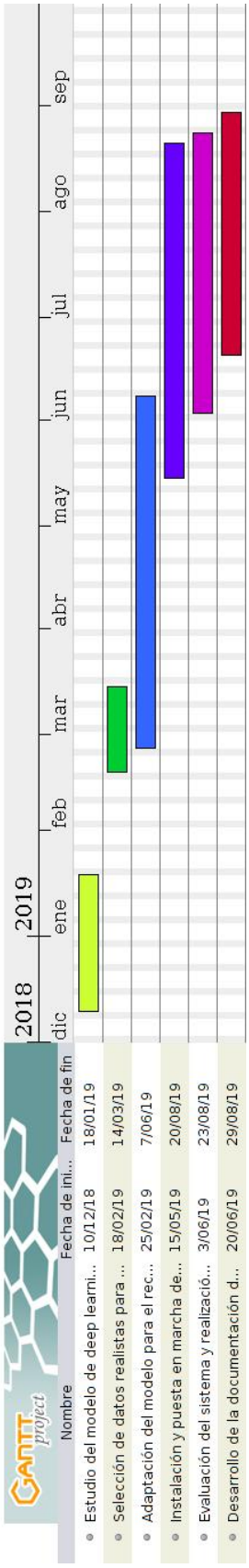


Figura 1.2: Diagrama de Gantt con la duración de las tareas realizadas en el proyecto

Capítulo 2

Reconocimiento Visual con *Deep Learning*

En este capítulo se describen los principales conceptos teóricos y teórico-prácticos estudiados para llevar a cabo el trabajo. En la primera sección, se explica el entorno de desarrollo con soporte para *deep learning* utilizado, en la segunda, el algoritmo de *deep learning* que se ha utilizado para realizar el reconocimiento visual, por otro último, los datos (*dataset*) elegidos para entrenar el algoritmo.

2.1. Tensorflow y Keras

Para desarrollar este proyecto, el primer paso fue estudiar y aprender los conocimientos básicos de dos de las bibliotecas de código abierto que más se utilizan en *deep learning*.

Tensorflow [17] es una interfaz para expresar e implementar algoritmos de aprendizaje automático, desarrollada por un equipo de investigación de inteligencia artificial de Google. Es capaz de funcionar en variedad de sistemas heterogéneos, en dispositivos móviles, ordenadores locales, en CPU, en GPU..., y a gran escala, ya que trabaja con cientos de servidores habilitados con GPU y ejecuta sus modelos en varias plataformas. Trabaja creando un gráfico de flujos de datos, que representa el calculo y el estado del algoritmo, incluyendo el preprocesamiento de entrada, las operaciones matemáticas y los parámetros del mismo. Este gráfico de flujo, facilita la ejecución de cálculos en paralelo y dividirlos en múltiples dispositivos. A diferencia de los sistemas de flujo tradicionales, Tensorflow permite a las variables tener un estado cambiante, compartido en diferentes ejecuciones del gráfico. Esto permite realizar actualizaciones de parámetros grandes y propagar estas actualizaciones a entrenamientos paralelos rápidamente, lo cual es crucial para llevar a cabo entrenamientos de modelos grandes. Esta arquitectura dota al sistema de mayor flexibilidad, pues es capaz de realizar subgrafos en los dispositivos que comparten el parámetro compartido, permitiendo así experimentar con nuevos algoritmos de optimización y estrategias de paralelización.

Tensorflow admite una variedad de aplicaciones, con un enfoque en entrenamiento e inferencia en redes neuronales profundas, que es la utilidad que le vamos a dar en este proyecto. Se usara la API (*application programming interface*) disponible en C++[1] y Python[2], el lenguaje más utilizado para el aprendizaje automático, que además cuenta con una gran variedad de librerías que se pueden exportar al programa cuando se buscan funcionamientos o herramientas específicos. Además con la herramienta de visualización Tensorboard, se podrán observar los gráficos obtenidos con Tensorflow en la web, para poder validar y comprobar los datos obtenidos en el programa.

La otra biblioteca de código abierto que también se va a utilizar en este proyecto, es Keras[3], una API de redes neuronales de alto nivel capaz de ejecutarse sobre Tensorflow y escrita en Python. Fue diseñada para realizar de forma rápida prototipos y experimentación de *deep learning*. Posee una interfaz sencilla de manejar y puede trabajar con redes convoluciones, recurrentes o con una mezcla de ambas, lo que da lugar a una de sus mejores ventajas, el hacer muy fácil desarrollar modelos de redes neuronales, siendo sencillo tanto crearlos desde cero como tener un modelo y extenderlo. Por lo que, usando Keras sobre TensorFlow, conseguimos que este sea más simple de usar sin sacrificar la flexibilidad y el rendimiento.

Aunque también hay otras bibliotecas como CNTK [18], o PyTorch[4], se ha escogido Tensorflow por presentar una mayor facilidad a la hora de llevar a cabo su estudio e implementación, y por poder manejar con suficiente eficiencia cargas de trabajo grandes y complejas. Y Keras por que consigue que el anterior sea más simple de usar sin sacrificar la flexibilidad y el rendimiento.

2.2. Algoritmos de detección

El principal objetivo del trabajo es llevar a cabo un sistema de reconocimiento visual eficiente, para eso lo primero que se hizo es elegir con que algoritmo de detección se quería trabajar.

2.2.1. Alternativas estudiadas.

Se analizaron fundamentalmente tres posibilidades, buscando aquellos que además de ser precisos, fueran conocidos por ser rápidos.

Faster R-CNN. Una de las opciones fue Faster R-CNN [19]. Sus predecesores R-CNN [20] y Fast R-CNN [21] usaban búsqueda selectiva, un algoritmo de propuesta de región basado en la agrupación de regiones similares en color, textura, tamaño y forma, para la detección de objetos. Pero Faster R-CNN utiliza una RPN (*Region Proposal Network*), una red totalmente convolucional, que habilita propuestas de región sin costo, donde pueden ubicarse los objetos, en mucho menor tiempo que la búsqueda selectiva, prediciendo simultáneamente

[1] `http://www.cplusplus.com/`
[2] `https://docs.python.org/3.5/reference/`
[3] `https://keras.io/`
[4] `https://pytorch.org/docs/stable/index.html`

límites de objetos y puntuajes de objetividad en cada posición. Esta red comparte capas convolucionales con otra red de detección que es la que termina de clasificar el objeto y de concluir el cuadro delimitador. Con esta arquitectura, este método de detección llega a tener una velocidad de 5 fps en GPU y unos mAP de 73.2 % en PASCAL VOC 2007. Si se utiliza el dataset de COCO Se obtiene 42.1 % de mAP con IoU 0.5.

YOLO. Otra posibilidad era el algoritmo de YOLO (You Only Look Once) [3], un modelo de detección de objetos que utiliza *deep learning* y redes neuronales convolucionales para localizar elementos de imágenes y vídeos. Su elemento más distintivo es que para llevar a cabo la detección, solo mira una vez la imagen, lo que le permite ser muy rápido aunque sacrifica algo de precisión. Para detectar los objetos, primero divide la celda en una cuadricula y en esas celdas predice posibles cuadros delimitadores y la probabilidad de que contengan un objeto. Seguidamente elimina las cajas que tienen baja probabilidad y las que podrían ser de objetos detectados por duplicado para dejar el mejor cuadro posible. Todo esto es llevado a cabo por una sola red. Estos atributos le permiten a YOLO ser un método muy útil en la detección de objetos a tiempo real. En su ultima versión, YOLO v3, el algoritmo consigue procesar imágenes a 30 *frames*por segundo (FPS) y tiene un mAP del 57.9 % en COCO test, y en la métrica de detección 0.5 IOU mAP obtiene un resultado de 57.9 AP50 en 51 ms.

RetinaNet. La última opción que se observó fue RetinaNet [22], una red unificada compuesta por una red principal y dos subredes de tareas especificas. La red principal, es una red piramidal que calcula un mapa de características convolucionales de la imagen de entrada. De las dos subredes, una realiza la clasificación a la salida de la red principal, y la otra realiza la regresión del cuadro delimitador. Un atributo que caracteriza a RetinaNet es que su función de perdida es la Pérdida focal en vez de la comúnmente usada pérdida de entropía cruzada. Esto hace que los cuadros con altas probabilidades de pertenecer al fondo, de no tener ningún objeto, contribuyen muy poco a la perdida para que el entrenamiento se centre en los casos más interesantes. Si se miran los resultados de la métrica de detección 0.5 IOU mAP RestinaNet obtuvo 57.5 AP50 en 198 ms, donde se puede apreciar que la ultima versión de YOLO nombrada antes es más rápida.

Por lo tanto, después de ver estos resultados, se eligió trabajar con el algoritmo de YOLO v3, ya que sobrepasa a Faster R-CNN en términos de velocidad y precisión, y obtiene resultados similares a RetinaNet pero 3.8 veces más rápido.

Por lo que hubo que llevar a cabo un estudio más profundo de este sistema, para su comprensión y realización del proyecto. En la figura 2.1 se ven unos ejemplos de detección de objetos con este algoritmo.

2.2.2. Algoritmo de detección YOLO.

Como se ha dicho anteriormente, YOLO, se conoce por *mirar* (procesar) una única vez la imagen para predecir donde hay objetos y de que clase son. Por lo que con una sola red convolucional predice a la vez los cuadros que delimitan los objetos y la probabilidad de que pertenezca a una clase, optimizando así el

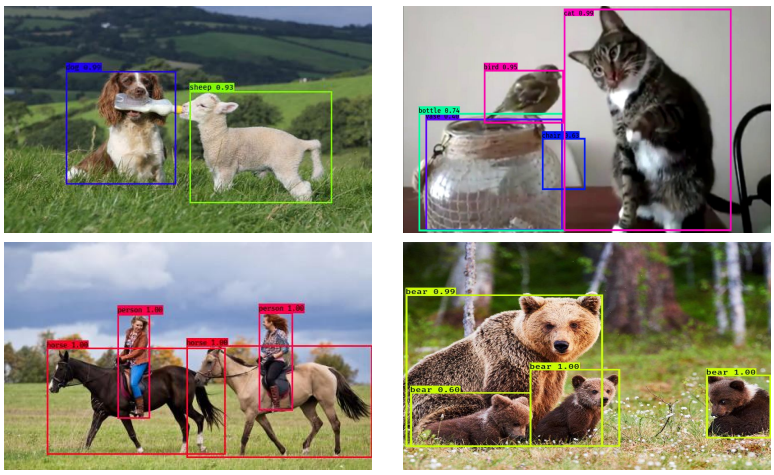


Figura 2.1: Ejemplos de detección de objetos con el algoritmo de YOLO v3

rendimiento de detección. Gracias a este único vistazo de la imagen, se obtiene un algoritmo rápido que además consigue información contextual sobre las clases al ver toda la imagen completa, consiguiendo menos confusiones entre los objetos y el fondo.

YOLO v3 consta de una red con 53 capas convolucionales, bloques residuales y conexiones directas sin capas completamente conectadas (*fully conected*). La imagen que entre a esta red puede ser de cualquier tamaño mientras sea múltiplo de 32, ya que la detección en los mapas de características se realiza en tres tamaños diferentes, que tienen un paso de 32, 16 y 8 respectivamente, por el que se vera dividido el tamaño de la imagen de entrada. La red neuronal trabaja con cajas de anclajes (*anchor boxes*), cajas ya predefinidas que se colocan en la imagen y a través de las predicciones del entrenamiento, se refinan para hallar los cuadros delimitadores. El uso de estas cajas de anclaje, permite a la red detectar múltiples objetos, objetos de diferentes escalas y objetos superpuestos. La predicción conseguida por la red será tal que:

$$\begin{aligned} bx &= \sigma(t_x) + c_x \\ by &= \sigma(t_y) + c_y \\ bw &= p_w e^{t_w} \\ bh &= p_h e^{t_h} \end{aligned} \tag{2.1}$$

Siendo t_x, t_y, t_w, t_h las coordenadas para el cuadro delimitador, c_x y c_y las distancias desde la esquina superior izquierda de la imagen y p_w y p_h el ancho y alto del cuadro anterior.

Con todos los cuadros delimitadores predichos, se pasa a hacer un filtrado de estos, de manera que aquellos que estén por debajo de un limite de probabilidad son eliminados. Y los que restan son procesados con *Non-maximum suppression*, una función que escoge el cuadro con mejor probabilidad y borra aquellos que sean muy cercanos para eliminar los cuadros superpuestos. Las predicciones realizadas por YOLO se llevan a cabo a tres escalas diferentes, basándose en las

redes piramidales, siendo de la forma: $NxNx[3 * (4 + 1 + C)]$ Siendo N x N la dimensión de la cuadrícula en la que quedara dividida la imagen de entrada, y C el numero de clases. Para ello, adjunta un conjunto de capas convolucionales a la red principal, que genera las predicciones a la primera escala. Desde ahí, saca los mapas de características de las dos capas anteriores, que son muestreados y concatenados a los mapas de características de la red principal y alimentados por otro conjunto de capas convolucionales que generan la segunda escala. Este mismo proceso se repite una vez más para conseguir la tercera y ultima escala de predicción.

Para el entrenamiento se *Batch Normalization*, que agrega lotes de normalización en cada una de las capas convolucionales de la red, normalizando así sus salidas para regularizar el modelo y obtener mejoras en la convergencia. También se utilizan varias funciones de perdida. En la localización de los cuadros delimitadores, usa la suma de la pérdida de error al cuadrado, para sacar el error de la localización o tamaño de las cajas. A la hora de evaluar las perdidas de confianza, utiliza una función logística para predecir la probabilidad del objeto. Para las predicciones de clases utiliza perdida de entropía cruzada, usando una clasificación *multilabel* sin utilizar ninguna capa *softmax*, ya que esto impondría una única clase por cuadro delimitador y no es la idea del modelo.

Comúnmente YOLO se apoya en Darknet, un código abierto de redes neuronales escrito en C y CUDA, pero para este proyecto se ha decidido por usar una implementación de YOLO v3 en Keras[5] para facilitar la creación de modelos y su uso e implementación.

2.3. Análisis de los *dataset*

Para llevar a cabo el entrenamiento del algoritmo de *deep learning* se han buscado y revisado distintos *dataset*. La idea que se ha querido llevar a cabo con este proyecto, es que el modelo de reconocimiento este entrenado para centrarse en vídeos de vigilancia. Por esta razón se busco un conjunto de datos que no tuviera muchas clases, pero que las pocas que tuviera fueran útiles para este propósito, como por ejemplo personas y vehículos.

Se consideró el uso de datasets públicos conocidos y muy utilizados en el campo del reconocimiento de objetos, como COCO [23], ImageNet y PASCAL VOC [24], por haber sido usados anteriormente con el algoritmo de YOLO y por lo tanto tener varios datos con los que poder comparar el modelo del proyecto. Por un lado, COCO posee 91 clases de objetos, todos ellos fácilmente reconocibles por una persona y etiquetados a partir de segmentación. Aunque esta base de datos es de las más utilizadas y cuenta con herramientas como su API de evaluación del modelo, se decidió no trabajar con ella por su gran numero de categorías y porque contenía gran cantidad de imágenes centradas en objetos que no interesaban al proyecto. En el caso de ImageNet, igual que pasaba con COCO, se tenían demasiadas categorías, ya que aunque en un principio solo tiene 27, luego cuenta con 21.000 categorías, por esta razón esta idea se rechazó pronto. Por otro lado PASCAL VOC cuenta con 20 clases, esto la hacía mucho más propicia para ser utilizada, pero ante la idea de centrarse en un

[5] `https://github.com/qqwweee/keras-yolo3`

Imágenes recogidas del dataset de VIRAT [27]

Figura 2.2: Ejemplos de imágenes del dataset VIRAT.

sistema de vigilancia, se decidió utilizar otro *dataset* cuyas imágenes fueran más específicas de escenas de calles y vías peatonales donde se suelen aplicar tareas de monitorización y vigilancia.

Se consideraron conjunto de datos centrados en cámaras de vigilancia, para detección de peatones [25] o vehículos [26]. Finalmente, el *dataset* elegido fue VIRAT [27], que es un conjunto de datos que contiene 29 horas de vídeos de vigilancia, correspondientes a 11 escenas distintas, en las que se muestran escenas reales de la calle, algunas con eventos más interesantes como puede ser un intercambio de objetos entre dos personas y otras con un día a día de tráfico de vehículos y gente. El *dataset* tiene dos partes, un conjunto de datos de vídeos terrestres y otro de vídeos aéreos. Para el proyecto se ha elegido trabajar con los datos de los vídeos terrestres, obtenidos de cámaras de vigilancia reales, que siguen teniendo un enfoque más aéreo que otros conjuntos de datos, pues estas cámaras suelen estar situadas en lugares altos.

VIRAT tiene tres tipos de anotaciones en sus datos. La primera es la anotación de eventos, que suelen relacionar personas y vehículos con acciones llevadas a cabo en los vídeos. La segunda es de mapeo, donde relacionan el evento y el objeto asociado al mismo. El tercer tipo es la anotación de objetos donde simplemente se detectan a personas, coches, vehículos, objetos y bicicletas, siendo los objetos mochilas Para el reconocimiento visual que se busca en el proyecto usaremos las anotaciones de objetos, para seguir la pista a las cinco clases descritas. En la figura 2.2 se pueden ver dos *frames*de los vídeos del dataset.

Capítulo 3

Prototipo Desarrollado

En este capítulo se va a detallar el prototipo desarrollado para el proyecto. Se quiere implementar el algoritmo de reconocimiento visual en un sistema embebido y probar el modelo con una cámara que grabe vídeo a tiempo real. Por lo tanto necesitaremos tres componentes principales, una cámara, una batería, y un kit del sistema embebido elegido.

3.1. Kit de desarrollo Jetson Xavier

Una de las cuestiones que hubo que analizar es con que módulo para sistemas embebidos se quería trabajar. Desde el principio se consideró trabajar con algún módulo de Nvidia Jetson[1], por su bajo consumo de energía y por sus resultados en el rendimiento computacional con el uso de la GPU (*Graphics Processing Unit*), que se encarga de la parte de computación más intensiva y deja el resto del código ejecutándose en la CPU, consiguiendo una computación de alto rendimiento y acelerando el funcionamiento de las aplicaciones de aprendizaje profundo.

Se consideraron distintas plataformas para llevar a cabo el proyecto: la Jetson Nano, las Jetson TX2 y la Jetson AGX Xavier. Comparando las tres, la Jetson AGX Xavier tiene el mayor número de núcleos Nvidia CUDA para usar la GPU, la Nano tiene 128 núcleos, las TX2 tienen 256 y la AGX Xavier posee hasta 512. Pasa lo mismo con su CPU, la cual es de 64 bits ARM de 8 núcleos, mientras que la Nano y la TX2 poseen 4 y dos núcleos respectivamente. Al ver que también tiene mejor descodificador de vídeo y mayor capacidad de memoria que las otras plataformas Jetson, se decidió trabajar finalmente con la Nvidia-Jetson AGX Xavier a pesar de que tenga un precio mayor y sea la plataforma con mayor tamaño de las tres alternativas consideradas. En la Figura 3.1 podemos ver las tres opciones de Nvidia Jetson que se consideraron usar en el proyecto.

Este módulo para sistemas embebidos ha sido diseñado principalmente para trabajar en robots terrestres, drones y máquinas autónomas que necesitan un máximo de cómputo para ejecutar las cargas de trabajo de la inteligencia artificial moderna. Su tamaño compacto es de 105mm x 105mm y tiene menos de

[1] `https://www.nvidia.com/es-es/`

(a) (b) (c)

Fuente: `https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/`

Figura 3.1: Modulos Nvidia de sistemas embebidos. (a) Nvidia Jetson Nano. (b) Nvidia Jetson TX2. (c) Nvidia Jetson AGX Xavier.

30W de consumo, lo que lo hace perfecto para cualquier aplicación embebida. Si nos focalizamos en tareas de visión por computador, la Jetson Xavier es capaz de ejecutar más de 30 trillones de operaciones. Resulta muy competente para implementar algoritmos de *deep learning* a dispositivos como sensores y cámaras, pues posee un consumo eficiente, lo cual puede aprovecharse para trabajar en sistemas de seguridad y vigilancia como es la idea de este trabajo. En la tabla 3.1 se pueden observar varias especificaciones técnicas de la NVIDIA Jetson AGX Xavier.

	Jetson AGX Xavier
GPU	512-Core Volta GPU with 64 Tensor Cores 11 TFLOPS (FP16) 22 TOPS (INT8)
Acelerador DL	(2x) NVDLA Engines 5 TFLOPS (FP16) 10 TOPS (INT8)
CPU	8-Core ARM v8.2 64-Bit CPU, 8MB L2 + 4MB L3
Memoria	16GB 256-bit LPDDR4x 2133MHz - 137GB/s
Almacenamiento	32GB eMMC 5.1
Codificación de vídeo	8x 4K @ 30 (HEVC)
Descodificación de vídeo	12x 4K @ 30 (HEVC)

Tabla 3.1: Tabla de especificaciones técnicas de la Jetson AGX Xavier

Para el proyecto, se adquirió el kit de desarrollo de la Jetson AGX Xavier. En el la plataforma con el sistema embebido incluye interfaces de hardware estándar y bibliotecas y API de inteligencia y visión artificial, con lo que se obtiene flexibilidad y rapidez a la hora de crear prototipos.

3.2. Otros componentes

Con el sistema embebido ya elegido, lo único que falta para terminar el prototipo es una cámara y una batería.

La cámara que se ha elegido es una webcam logitech C170, mostrada en la Figura 3.2 (a), por ser fácil de incorporar en la Jetson vía USB y por su bajo consumo. Cuenta con una captura de vídeo nítido con resolución de 1024 x 768 y una velocidad máxima de fotogramas de 30 fps. Es pequeña y ligera, por lo que

(a) (b)

Figura 3.2: Cámara y batería utilizadas en el prototipo desarrollado. (a) Webcam logitech C170 . (b) Batería XT-20000Q3-PA

se puede cambiar fácilmente de ángulo o posición para poder grabar distintas escenas.

La batería utilizada es la XT-20000Q3-PA de XT POWER[2], en la Figura 3.2 la imagen (b). Esta batería recargable esta pensada para el funcionamiento de dispositivos pequeños y medianos como el sistema embebido. Es de iones de litio por lo que es duradera y de alto rendimiento. Tiene un voltaje de 12 a 24 voltios y una carga de 0.01 a 50 vatios. Su capacidad de batería es de 20100 mAh. Y posee tres salidas, una optimizada para un computador portátil, que es la que se usara para alimentar la Jetson, y dos para dispositivos compatibles con USB. Su tamaño, como el de la cámara, también es relativamente pequeño y ligero, 2 cm de grosor y 0,45 kg de peso. Y cuenta con protección contra sobrecarga, descargas y cortocircuitos. Cuando la batería esta conectada al sistema embebido, y se pone en marcha el algoritmo de detección de objetos a tiempo real a través de la webcam, este consume aproximadamente 15.5 vatios. Si se mantiene el modelo funcionando de forma continua, la batería dura algo menos de 4 horas.

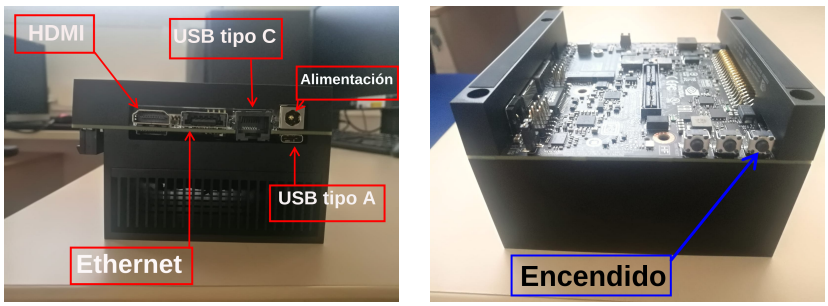
Además de estos componentes principales, se necesitaran periféricos básicos, una pantalla, un ratón y un teclado.

3.3. Utilización del prototipo

Con todos los componentes elegidos, solo queda montar el prototipo y ponerlo en marcha. Para ello se tendrá que conectar la cámara, la batería, el ratón, el teclado y la pantalla, a la Jetson AGX Xavier.

En la Figura 3.3 (a) se puede observar una imagen indicando como conectar los componentes necesarios al módulo del sistema embebido. En el puerto a la izquierda de la imagen se conectara la pantalla a través de un cable HDMI. Al lado de este, se tiene un puerto USB tipo C donde se conectara la webcam de logitech. El siguiente puerto es para un cable Ethernet para la conexión a internet, puesto que le modulo de Jetson no posee Wifi, aunque para poner en marcha el modelo creado este cable no es necesario. A la derecha del todo quedan dos puertos, el de arriba es el de la alimentación, por lo que es ahí donde se añadirá la batería. Abajo hay un puerto USB tipo A, pero como único que

[2] `https://www.xtpower.de/`



(a) (b)

Figura 3.3: Indicaciones para conectar y encender el módulo Jetson AGX Xavier. (a) Indicaciones para el conexionado. (b) Indicaciones de encendido

queda por conectar al módulo es un ratón y un teclado, se usara un adaptador de tipo C a tipo A, que viene incluido en el kit de la Jetson Xavier. Para encender el módulo Jetson solo es necesario pulsar el botón indicado en la Figura 3.3 (b). Con esto hecho el prototipo estará completamente conectado y listo para funcionar.

Hay que tener en cuenta que para trabajar con este modelo en el sistema embebido, este debe tener varios programas instalados:

- Python 3: Por ser el lenguaje empleado en la programación del modelo.
- YOLO v3, Tensorflow y Keras: Para poder trabajar con el algoritmo de reconocimiento visual y poder ejecutar la implementación de YOLO v3.
- Implementación de YOLO v3 en Keras: Es el algoritmo final que realizara el trabajo de detección en el vídeo grabado por la cámara en vivo.

Con los programas funcionando correctamente en el sistema embebido, solo se debe ejecutar el archivo de Python que pone a grabar la webcam y saca por pantalla el vídeo recibido desde esta y la detección llevada a cabo por el algoritmo a tiempo real. Para eso se abre un terminal en la pantalla conectada al sistema embebido y localizándose en la carpeta YOLO3KERAS con el comando *cd Pilar/YOLO3KERAS* se ejecuta el archivo con la orden *python3 Test.py*

En la Figura 3.4 se puede observar el prototipo completamente montado en la mesa, con el modelo de detección de objetos a tiempo real funcionando, detectando una persona en el laboratorio del grupo RoPeRT, en el Instituto de Ingeniería e Investigación de Aragón.

También se ha desarrollado otro pequeño programa que además de hacer la detección de objetos a tiempo real, graba el vídeo que se crea. Con esto se consigue eliminar la necesidad de estar conectado a la pantalla todo el tiempo. Así, si el usuario quiere grabar otro entorno solo necesita la pantalla y el teclado para activar el programa, luego podrá desenchufar estos elementos y trasladar los tres componentes principales, Jetson AGX Xavier, webcam y batería, que son más pequeños y ligeros, por el lugar deseado. Cuando se desee ver la grabación con la detección realizada, simplemente habrá que volver a conectar la pantalla y el teclado y buscar el vídeo grabado en el mismo directorio donde está el

(a) (b)

Figura 3.4: Prototipo desarrollado, montado en el laboratorio del grupo RoPeRT. (a) Prototipo desarrollado detectando una persona en vídeo a través de la webcam. (b) Captura del vídeo detectado en el prototipo.

Figura 3.5: Imágenes con el prototipo portátil grabando diferentes entornos.

Figura 3.6: Capturas de los vídeos grabados por el prototipo en los alrededores del I3A.

programa a utilizar. En la Figura 3.5 se puede ver a una persona llevando el prototipo, en la 3.6 una captura de los vídeos grabados en el entorno elegido.

Capítulo 4

Evaluación y experimentación

A lo largo de este proyecto se han realizado diferentes experimentos para evaluar tanto las distintas alternativas como el prototipo final. En este capítulo se procederá a analizar estos experimentos y los resultados obtenidos con ellos, sobretodo centrándose en la precisión y el rendimiento conseguidos.

4.1. Configuración y evaluación de los experimentos

El objetivo principal del proyecto es conseguir un sistema de reconocimiento visual con el algoritmo YOLO v3 y el *dataset* VIRAT que resulte eficiente en un sistema embebido, en este caso en la NVIDIA Jetson AGX Xavier. Para intentar alcanzarlo se han llevado a cabo diferentes experimentos, variando principalmente parámetros del modelo YOLO o el conjunto de datos de entrenamiento utilizados del *dataset* VIRAT.

En esta sección se resumen los datos y métricas utilizadas en los experimentos de este capítulo.

4.1.1. Datos utilizados

4.1.1.1. Dataset VIRAT.

VIRAT tiene 315 vídeos, pertenecientes a 11 escenarios diferentes, y en cada uno de ellos se captura distintas horas y días, donde ocurren diferentes sucesos, más o menos relevantes. En el anexoA se pueden ver imágenes pertenecientes a los 11 escenarios.

Para entrenar los diferentes modelos, se extrajeron los *frames* de los vídeos, puesto que resultaba más fácil trabajar con imágenes directamente. Se obtiene una cantidad muy grande de imágenes, unas 900.000, que al ser de vídeos de vigilancia, sobretodo las pertenecientes a escenarios de aparcamientos, resultaban muy similares. Para facilitar su procesado y evitar que se produjera un

sobreajuste en el entrenamiento, se hizo un *sampleado* homogéneo para reducir este número en un 95 %, quedando un total de 45.000 *frames* finales.

Las anotaciones de VIRAT vienen dadas en un formato txt con 8 columnas. La primera representa el id de la anotación y la segunda la duración de la esta. La tercera el frame al que pertenece. La cuarta, quinta, sexta y séptima son las coordenadas de la caja delimitadora, siendo x, y, anchura y altura respectivamente. La ultima columna contiene el id de la clase objeto que hay en la caja. Para llevar a cabo el entrenamiento en el algoritmo elegido, la implementación de YOLOv3 en Keras, las anotaciones han de tener una forma especifica, por lo que estos txt fueron cambiados al formato xml, siguiendo el ejemplo de VOC Pascal desde donde se podría crear fácilmente el formato final de las anotaciones. Este último txt con las anotaciones finales debe contener en cada línea, la dirección del *frame* utilizado seguido de todas sus cajas delimitadoras, más el id de la clase encontrada en cada caja. Con las anotaciones correctamente escritas, se puede proseguir a realizar el entrenamiento.

Variaciónes de YOLO entrenadas. Se realizaron distintos experimentos para buscar el mejor resultado para el proyecto. Para eso se crearon diferentes subconjuntos de datos, que varían en el número de clases o de elementos etiquetados con las que se entrena el modelo de YOLO. La razón principal de entrenar los siguientes modelos es el número de ejemplos etiquetados de cada clase en el *dataset*. El total de anotaciones en cada categoría es: 115.053 para personas, 327.422 para los coches, 181 de vehículos, 5.167 objetos y 21.734 bicicletas. Se puede observar que las categorías de vehículos y objetos son mucho menos frecuentes en los vídeos del *dataset*. Los tres modelos entrenados son los siguientes:

- VIRAT: Se ha entrenado el modelo con los datos de VIRAT completos utilizando el conjunto mencionado de los *frames* extraídos y *sampleados*. Se utilizan las 5 clases, *persona, coche, vehículo, objeto* y *bicicletas* y todas las anotaciones que se tienen de cada una de ellas. Como el número de estas anotaciones es grande se considera que no hace falta realizar muchas épocas para este entrenamiento, llevándose a cabo con solo 18 épocas, teniendo en las primeras 9, capas congeladas para acelerar el tiempo de cómputo.

- VIRAT-20000: Se ignoran dos de las clases que tienen menos anotaciones, *vehículos* y *objetos*, y entrenamos este modelo para tres categorías: *personas*, *coches* y *bicicletas*. Para balancear el número de anotaciones de estas clases, se submuestrean tanto las personas como los coches y se quedan con 20.000 anotaciones de cada una, que es el número aproximado que hay de bicicletas. Este entrenamiento se lleva a cabo con 60 épocas, y las primeras 30 se hacen con capas congeladas, un número mayor que en el caso anterior ya que en número de anotaciones que tenemos en este modelo es menor.

- VIRAT-2200: El tercer modelo se entrena ignorando solo la clase *vehículos*. Las otras cuatro, *persona*, *coche*, *objeto* y *bicicleta*, si se incluyen en el entrenamiento. Igual que se hace en el modelo reducido con 20.000 anotaciones, en esta prueba también se lleva a cabo un balance de clases. Como

esta vez la clase más reducida es la clase *objeto*, con aproximadamente 2.200 anotaciones, se han submuestreado las otras tres clases a entrenar para que también tengan este número de datos. Al tener aún menos anotaciones para llevar a cabo el entrenamiento que con los otros dos modelos, el número de épocas se ha aumentado a 90, con capas congeladas en las primeras 50.

subconjuntos de entrenamiento y evaluación. Para el entrenamiento y evaluación de los modelos, se separaron los 315 vídeos de VIRAT en tres grupos, entrenamiento, validación y test. Esta separación es necesaria en cualquier sistema de aprendizaje supervisado, ya que el modelo necesita de un grupo de datos anotados para poder aprender de ellos, este conjunto de datos es la agrupación de entrenamiento. Se necesita además un grupo de datos de validación, que corrobora el funcionamiento del modelo y sirve para seleccionar el mejor entrenamiento realizado. Por último, el conjunto de test ofrece una evaluación final adicional del modelo elegido, y reporta la eficacia de este según los resultados obtenidos en este conjunto. Se realizaron dos separaciones distintas para estos conjuntos:

- VIRAT-Escenarios: La separación de los vídeos en entrenamiento, validación y en test se hizo por escenarios. Se dejaron 7 escenarios para el entrenamiento, con 214 vídeos que contienen 27.902 *frames*. En validación 2 escenarios, con 72 vídeos, 5.859 *frames*. Y en test otros dos escenarios, siendo 29 vídeos con 9.604 *frames*.
- VIRAT-Aleatorio: En este caso la separación no tuvo en cuenta a que escenario pertenecía cada vídeo. Simplemente se realizó un reparto aleatorio de los vídeos del *dataset*, teniendo en cuenta que aunque sean de mismos escenarios, se graban distintos momentos con distintos sucesos, por lo que los vídeos son lo suficientemente distintos para llevar a cabo el aprendizaje, la validación y el test correspondiente. Cuando separamos los datos, nos queda en el entrenamiento 110 vídeos, con 17.122 *frames*. En validación 102, con 14.648 *frames*. Y en test otros 102 vídeos con 12.105 *frames*.

4.1.1.2. Dataset Zaragoza.

Además de probar la validación del modelo con vídeos del propio *dataset* VIRAT, se quiso verificar el funcionamiento del algoritmo en otro tipo de escenas. Para ello se grabaron cuatro vídeos de algunas calles de la ciudad de Zaragoza, donde salieran tanto personas como vehículo. Los vídeos están grabados desde ventanas de edificios, buscando que no se pierda el enfoque típico de la cámara de vigilancia.

- Santander: vídeo de la calle Santander conseguido desde un segundo piso.
- Goya: vídeo de la avenida Francisco de Goya, grabado desde un cuarto piso.
- Academia: vídeo de la avenida Academia General Militar, registrado desde un segundo piso.

- España: vídeo de la plaza España grabado desde el primer piso de puerta Cinegia, con *zoom*.

Todos los vídeos tienen una duración aproximada de 30 segundos. Los tres primeros son calles donde pasan vehículos, peatones y/o motocicletas. España es una plaza donde parte de la escena esta tapada por árboles y sombrillas, obteniendo un escenario distinto.

4.1.2. Métricas de evaluación

Para calcular la precisión de cada experimento, se decidió usar la métrica mAP (*mean Average Precision*) con una IoU (*Intersection over union*) de 0.5, ya que en la actualidad, muchas investigaciones trabajan con esta medida.

La IoU es una métrica que evalúa la superposición entre 2 cajas delimitadoras. Se usa para medir cuánto se superpone la caja predicha con la anotación real. Es común que se defina un umbral de IoU, normalmente de 0.5, para clasificar si una predicción es positiva o no.

$$IoU = \frac{área\,de\,intersección}{área\,de\,unión} \tag{4.1}$$

Por otro lado tenemos la precisión y la *recall*, que miden el rendimiento de un modelo, basándose en las equivocaciones y aciertos llevados a cabo por el detector.

$$precision = \frac{tp}{tp + fp} \qquad recall = \frac{tp}{tp + fn} \tag{4.2}$$

Con tp el número de verdaderos positivos acertados por el sistema y fp y fn los falsos positivos y negativos fallados. Al hacer una gráfica con estas dos medidas, con la precisión en función del *recall*, el área bajo la curva que mide el valor promedio de la precisión en cada intervalo desde que el *recall* es 0, hasta que es 1.

$$AP = \int_0^1 p(r)dr \tag{4.3}$$

Este promedio bajo la curva, es la AP (*Average precision*) y la mAP es simplemente la media de esta precisión promedio.

$$mAP = \frac{\sum_{q=1}^{Q} AP}{Q} \tag{4.4}$$

Con la idea de calcular este valor, se usó la API de COCO[1]. Esta interfaz de programación viene dada en Matlab[2], Python y Lua [28], para tratar, visualizar y analizar las anotaciones del *dataset*. Para poder utilizarla se han de pasar las anotaciones al formato requerido por COCO, lo que implica convertir los txt usados para el entrenamiento al formato json. También es necesario pasar al mismo formato, las predicciones llevadas a cabo por los modelos entrenados,

[1] `https://github.com/cocodataset/cocoapi`
[2] `https://es.mathworks.com/products/matlab.html`

en los vídeos donde se quiere sacar la precisión. En este caso serán aquellos reservados para validación. Con las anotaciones y las predicciones en el formato requerido, y de la forma indicada en la página web de COCO[3], se usa un pequeño programa que evalúa las cajas delimitadoras de los dos archivos json, y saca los mAP de cada clase.

4.2. Comparación de diferentes modelos de YOLO en *benchmarks* públicos

Análisis de precisión de los distintos modelos. En primer lugar se realizó un experimento para comparar el modelo COCO con el modelo VIRAT. El primero es el modelo de la implementación de YOLO v3 en Keras, disponible online, el cual esta entrenado con el *dataset* de COCO y por eso se le ha dado en este trabajo ese nombre. El segundo, es el modelo de YOLO que se ha entrenado en este trabajo con el *dataset* VIRAT en la misma implementación. Con esta prueba se busca ver el rendimiento en el *dataset* completo de los modelos más generales. Se utilizan los datos de validación del subconjunto VIRAT-Escenarios para llevar a cabo la evaluación de los modelos, este grupo de datos consta de 72 vídeos y la evaluación mostrara la media de todos ellos, para cada categoría. Sin embargo, no se ha tenido en cuenta la clase vehículo a la hora de llevar a cabo los análisis. Esto es debido a que las 181 anotaciones pertenecientes a la clase vehículo, pertenecen todas al mismo vídeo, por lo que al entrenar con el, para que el modelo fuera capaz de aprender la categoría, no se dispone de ninguna anotación de esta clase en otro conjunto de datos, ni de validación, ni de test.

La Tabla 4.1 muestra los resultados obtenidos, donde se puede ver que las clases de coches y de personas son las que mejor salen en los dos modelos. En la clase de coches se puede apreciar que se obtiene un mejor resultado con el modelo existente de COCO, no obstante en personas el modelo VIRAT se comporta mejor. Las clases objetos y bicicletas son distintas en el *dataset* de COCO que en el de VIRAT, los objetos porque en COCO no existe dicha categoría, y las bicicletas porque en mientras que en VIRAT se refiere tanto a bicis como a motos, en COCO estas dos clases están separadas. Por lo que, al llevar a cabo el análisis se ha medido la precisión de esa clase con motocicletas por ser de las que más datos hay.

Modelo	Personas	Coches	Objetos	Bicicletas
COCO	17.4	45.2	-1	0
VIRAT	19.7	26.0	0	0

Tabla 4.1: Tabla de precisión (IoU 0.5 en el conjunto de validación VIRAT-Escenarios) de dos modelos de YOLO.

Para poder analizar mejor estos resultados, se ha procedido a realizar un análisis de los vídeos de validación de manera individual. Se han elegido dos vídeos como ejemplo de los 72 presentes, para poder discutir los resultados. Cada uno de ellos pertenece a uno de los dos escenarios que crean los datos de

[3] `http://cocodataset.org/#format-data`

Figura 4.1: Escenarios del conjunto de evaluación VIRAT-Escenarios

validación y que se pueden ver en las imágenes de la Figura 4.1. El primer vídeo escogido pertenece a la escena de la plaza con sombrillas a la izquierda de la Figura y es VIRAT_S_010003_08_000739_000778, será el Vídeo 1S. El segundo vídeo, el aparcamiento de la derecha, es VIRAT_S_050300_09_001789_001858 y será el Vídeo 2S. Los resultados obtenidos se pueden ver en la Tabla 4.2.

Vídeos	Personas	Coches	Objetos	Bicicletas
VÍDEO 1S	10.4	-1	-1	-1
VÍDEO 2S	27.9	47.4	-1	-1

Tabla 4.2: Tabla de precisión en distintos vídeos de validación (IoU 0.5 en el conjunto de validación VIRAT-Escenarios) con el modelo de YOLO VIRAT

Observando los resultados es interesante el escenario del Vídeo 1S, puesto que es uno de los que más difieren dentro del *dataset*. En general todo lo que hay son personas con algunos objetos, pero alguna vez por la zona superior de la imagen pasa un coche o alguna bicicleta. La detección de los objetos casi siempre es baja, pues son pequeños y difieren de forma como se ve en el anexo A, y los coches y las bicicletas que circulan en esa zona alejada que difiere tanto de las carreteras no son reconocidos. La precisión de la clase personas va aumentando y disminuyendo dependiendo de la posición de estas. Cuando se encuentran en las escaleras o la parte de arriba de la imagen la precisión obtenida es mayor, cuando se encuentran entre las sombrillas que obstaculizan la visión el resultado es el que se ve en la Tabla 4.2. En el segundo escenario, el del Vídeo 2S, se encuentra un aparcamiento. Estas escenas son más comunes dentro de los datos de entrenamiento y se puede apreciar que el modelo por lo tanto se equivoca menos, las bicicletas y los objetos en cambio siguen siendo poco frecuentes, pero obtiene una buena precisión en la clase coche y también aumenta la correspondiente a personas. No obstante la cantidad de vídeos de este escenario dentro del conjunto de validación es menor que la del primer escenario, y al obtener la media del conjunto, la precisión obtenida al final disminuye.

Podemos concluir con este experimento que el modelo de YOLO VIRAT baja su precisión cuando se encuentra con un escenario muy diferente de los escenarios de entrenamiento, como puede ser una plaza sin carreteras. Sin embargo, en escenas más comunes, como aparcamientos y calles, obtiene tan buen resultado como el modelo ya existente de la implementación.

Tras estos resultados, se lleva a cabo un segundo experimento donde se compararan de nuevo los modelos de COCO y VIRAT y además las variaciones

VIRAT-20.000 y VIRAT-2.200. Este experimento se realiza para comprobar la eficacia de los modelos cuando se les incorporan todos los escenarios posibles. Esta vez el grupo de datos de validación pertenece al subconjunto de VIRAT-Aleatorio, el cual contiene un total de 102 vídeos. Igual que en la prueba anterior los resultados serán las medias obtenidas con los 102 vídeos para cada categoría menos la de vehículo. Además en el modelo VIRAT-20.000 tampoco se incluye la clase objeto.

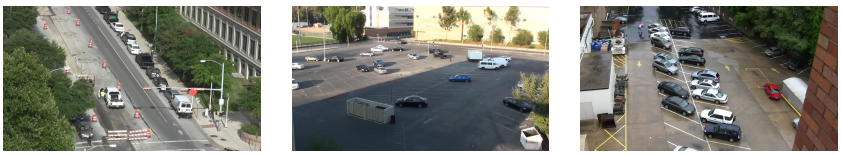
Como se puede apreciar en la Tabla 4.3, con esta prueba la precisión en la detección del modelo VIRAT sube respecto al caso anterior, sobrepasando en las clases de personas y coches al modelo de COCO, que con este grupo de validación ha empeorado su comportamiento en la clase de coches y otro poco en la de personas. Los resultados en las clases de objetos y bicicletas siguen siendo bajos para ambos modelos. Si se miran los otros dos modelos, VIRAT-20.000 y VIRAT-2.200 los resultados reflejan que se ha conseguido incrementar un poco la precisión de las clases que poseen menos datos de entrenamiento, objetos y bicicletas. Pero el balance global de todas las clases es peor, ya que con el submuestreo de datos en las clases de personas y coches se baja la precisión de estas. Se determina entonces que los modelos generales son más adecuados para implementarse en el prototipo.

Modelo	Personas	Coches	Objetos	Bicicletas
COCO	15.2	17.1	-1	0
VIRAT	32.6	46.9	1.2	7.4
VIRAT 20.000	28.4	41.1	-1	7.6
VIRAT 2.200	10.9	23.8	4.6	8.8

Tabla 4.3: Tabla de precisión (IoU 0.5 en el conjunto de validación VIRAT-Aleatorio) comparando los diferentes modelos de YOLO

Igual que en la prueba anterior, para analizar con más detalle los resultados, se procede a obtener los resultados de cada vídeo de validación de manera individual. De los 102 vídeos, se escogen tres vídeos con resultados muy distintos para llevar a cabo el análisis y buscar las posibles causas de una mejor o peor precisión en los modelos.El Vídeo 1 es VIRAT_S_050000_03_000585_000639, el Vídeo 2 VIRAT_S_010107_01_000068_000196 y el Vídeo 3 VIRAT_S_050202_03_000608_000684, se puede ver una imagen de cada uno de ellos en la Figura 4.2 de izquierda a derecha respectivamente. La precisión obtenida en estos tres vídeos realizada con el modelo VIRAT, que obtuvo la mejor precisión en este experimento, se puede ver en la Tabla 4.4.

En el primer vídeo se obtiene un porcentaje alto sobretodo en coches, y un buen número comparado con los registros pasados en bicicletas. Esto es debido a que varias anotaciones de bicicletas son de motos aparcadas en aceras, donde es más fácil distinguirlas de los coches que en la carretera. Además la visión del escenario, permite ver los coches que pasan de frente o de espalda, sin posiciones ladeadas ni complicadas, lo que facilita su detección. El segundo vídeo es un punto intermedio, los porcentajes han disminuido respecto al caso anterior pero todavía no son muy bajos. La mitad de este escenario se encuentra al sol, y



(a) (b) (c)

Figura 4.2: Imágenes de los tres vídeos de validación del subconjunto VIRAT-Aleatorio analizados individualmente con el modelo VIRAT. (a) VIRAT_S_050000_03_000585_000639. (b) VIRAT_S_010107_01_000068_000196. (c) VIRAT_S_050202_03_000608_000684

la otra mitad a la sombra, este cambio de luminosidad dificulta la detección de elementos más pequeños como las personas e incluso la de algunos coches. También el enfoque del aparcamiento crea que algunos coches oculten partes de otros, dejando alguno sin reconocer. En el tercer vídeo se puede verificar que los porcentajes de las categorías de personas y coches, que son los únicos que aparecen en este vídeo, son muy pequeños. En el caso de las personas se puede deducir que la principal causa es el tiempo, pues al estar lloviendo los peatones llevan paraguas, y puesto que es la única escena donde esto pasa, el modelo no es capaz de detectarlo correctamente. A la vez la perspectiva de este vídeo es más cercana a los coches, comparada con otras cintas de vigilancia que tienen un enfoque desde más altitud, esto hace que vea los coches con un mayor tamaño que en el resto de escenarios, además como pasaba en el vídeo anterior algunos coches tapan partes de otros haciendo más difícil la detección.

Vídeo	Personas	Coches	Objetos	Bicicletas
VÍDEO 1	36.5	90.5	5.1	28.5
VÍDEO 2	20.8	65.1	-1	1.9
VÍDEO 3	10.6	12.2	-1	-1

Tabla 4.4: Tabla de precisión en distintos vídeos de validación (IoU 0.5 en el conjunto de validación VIRAT-Aleatorio) con el modelo de YOLO VIRAT

Se concluye el experimento con estos datos, de manera que se ha obtenido una mejora en la precisión cuando el modelo entrena con más escenarios. El modelo VIRAT consigue un mejor reconocimiento que las variaciones VIRAT-20.000 y VIRAT-2.200 que no han conseguido mejorar de forma notable el resultado de las categorías de bicicletas y objetos. Por lo que el modelo a elegir entre estas variaciones de YOLO es VIRAT, que consigue la precisión más alta en personas y coches con un 32.6 % y un 46.9 % respectivamente.

Para finalizar el análisis se ha constatado la precisión del modelo en el conjunto de datos de test para los dos subconjuntos existentes. En VIRAT-Aleatorio se obtiene un 37.5 % de mAP en personas, un 52.5 % en coches y un 2.7 % en objetos sin bicicletas. En VIRAT-Separación se consigue un 43.8 % en la clase personas, un 38.5 % en coches y un 7.1 % en objetos igualmente sin bicicletas. Como se puede observar estos datos son mejores que los obtenidos con los conjuntos de validación. Esto es debido a que casualmente los escenarios de validación proporcionaban una mayor dificultad a la hora de detectar objetos, como

puede verse sobretodo en los ejemplos de las imágenes vistas anteriormente ala izquierda en la Figura 4.1 y a la derecha en la Figura 4.2.

Tiempo de ejecución y Consumo. Además de la precisión, se ha querido analizar el tiempo y el consumo que requiere ejecutar el modelo. Para ello se han usado el módulo *time* de python, *powerstat* [4] y la interfaz del sistema NVIDIA, nvidia-smi, en los algoritmos de detección de imágenes y vídeos. Estas medidas se han llevado a cabo en tres dispositivos distintos:

- Portátil: Ordenador portátil Acer, con tarjeta gráfica AMD Radeon R4.
- Titan: Ordenador de sobremesa LG, con tarjeta gráfica NVIDIA Titan Xp.
- Xavier: El sistema embebido utilizado en el trabajo, la NVIDIA Jetson AGX Xavier.

En la Tabla 4.5 podemos ver los tiempos de predicción de una imagen y el consumo alcanzado con el modelo VIRAT. Se ha utilizado un solo modelo ya que aunque cambie el número de datos de entrenamiento y las épocas, la red utilizada en los entrenamientos es la misma para todos los experimentos, con el mismo número de capas y el mismo número de neuronas en cada una, por lo que los tiempos y los consumos de todos ellos han de ser el mismo y solo se verán cambios dependiendo de donde se ejecute el modelo.

	Portatil	Titan	Xavier
Tiempo (s)	5.6842	0,0760	0,4049
Consumo (w)	14.59	160.78	15,58

Tabla 4.5: Medidas de tiempo de ejecución y consumo de una predicción (inferencia) en tres dispositivos diferentes

Como se puede apreciar, los tiempos de predicción de una imagen son mayores en el portátil Acer, que contiene una gráfica menos potente y no posee GPU. No obstante en el ordenador de sobremesa el tiempo es de 76 ms, con lo que se puede apreciar la rapidez del modelo aunque no llegue a la velocidad que tiene YOLO v3 por si solo. En el sistema embebido se consigue un punto intermedio entre los otros dos dispositivos, pues aunque también posee GPU no tiene tanta potencia, por lo que el resultado de 405 ms resulta un valor eficiente para la Jetson.

Por otro lado tenemos el consumo del modelo, en el portátil Acer para hacer esta medida se ha instalado el programa *powerstat*, con el, se ha comprobado que el consumo llevado por el ordenador sin realizar ninguna tarea era de 7.61 w, mientras que al ejecutar simplemente el modelo esta cantidad incrementaba a 14,59 w. Aunque este consumo no es elevado, le detección llevada a cabo en esta plataforma ya se ha visto que es lenta. Cuando sin embargo comprobamos los valores del ordenador de sobremesa con la Titan Xp el valor inicial de consumo era de 25 w, y al ejecutar el programa alcanzaba los 160.78 w. El consumo es

[4] `https://github.com/ColinIanKing/powerstat`

mucho mayor pero la velocidad a la que lleva a cabo la detección de las clases también lo es puesto que utiliza más potencia para ello. Por último se encuentra la Jetson AGX Xavier la cual comenzaba con un consumo inicial de 9.1 w, 6.9 si los ventiladores no se encuentran encendidos, y con el modelo en marcha llega a consumir 15.58 w. El nivel de consumo se parece al portátil Acer, sin embargo la rapidez con la que se lleva a cabo el reconocimiento de imágenes es 10 veces mayor, lo que demuestra el bajo consumo más eficiencia que se consigue con un sistema embebido.

4.3. Experimentos con vídeos propios

Con los modelos analizados, se decidió comprobar el funcionamiento del modelo VIRAT con el conjunto de datos VIRAT-Aleatorio, pues ha entrenado con todos los escenarios que se tenían, en otros panoramas. Para ello se va a usar el *dataset* Zaragoza, con vídeos de las calles de la ciudad grabados desde pisos altos para intentar dar ese enfoque de vídeo de vigilancia.

La Figura4.3 muestra varios ejemplos de los resultados obtenidos en el vídeo Santander. En este vídeo el modelo funciona en general correctamente, se puede apreciar alguna equivocación pues puede dejarse sin detectar algunos elementos de la imagen. Además la clase moto parece que solo la detecta cuando está en la acera, en cambio cuando está en la carretera le da la categoría coche o incluso persona más coche. Pero en general lleva a cabo una buena detección en los elementos de la imagen. Cuando estas pruebas se hacen en el ordenador Acer, la detección se lleva a cabo a un fps. Mientras que al ejecutarlo en el ordenador

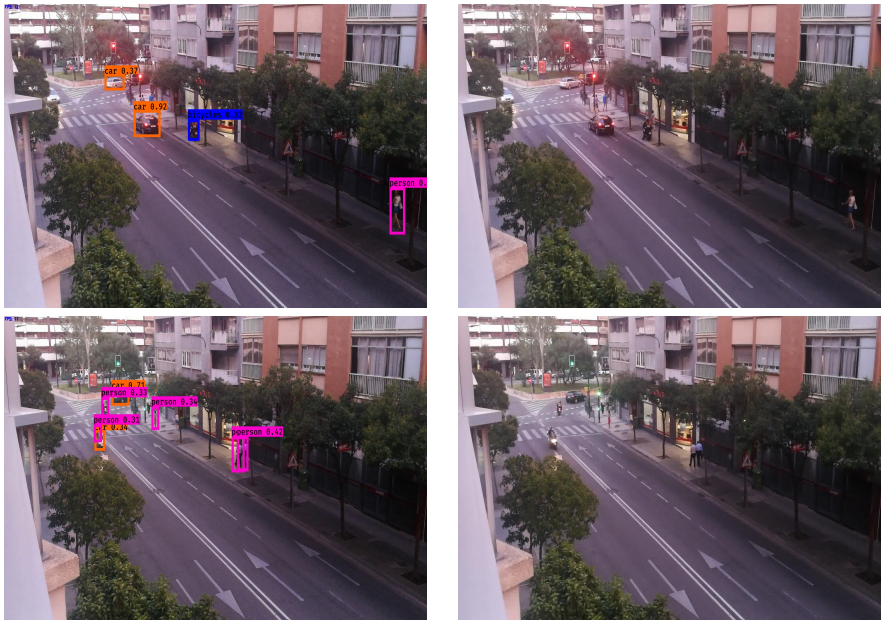


Figura 4.3: Capturas de la detección llevada a cabo en el vídeo Santander con el modelo VIRAT

Titan se alcanzan los 14 fps.

En el vídeo Goya, Figura4.4, vemos que hay momentos donde hay muchos

Figura 4.4: Capturas de la detección llevada a cabo en el vídeo Goya con el modelo VIRAT

Figura 4.5: Capturas de la detección llevada a cabo en el vídeo Academia con el modelo VIRAT

elementos en el escenario. Aquí se detectan bastante bien los coches y las personas. El problema está en que confunde partes de semáforos o maniquíes de los escaparates con personas, detectando más personas de las que debería haber, dando falsos positivos para esta clase. La razón es que este vídeo se ha grabado a mayor altura, donde las formas humanas suelen ser simplemente cuerpos alargados que son más fáciles de confundir. Igual que con el vídeo anterior en el ordenador Acer el modelo sigue funcionando a un fps, en Titan alcanza los 12, una pequeña disminución comparada al escenario anterior debido a la mayor concentración de objetos detectados.

En el vídeo de Academia sale una carretera donde se aprecian los coches a mayor velocidad, puesto que se ha encuadrado una zona sin semáforo para que tener una diferencia respecto a los dos escenarios anteriores. Estos coches si son detectados por el modelo, pero también se verifica que al pasar un autobús o un camión no se lleva a cabo el reconocimiento como se aprecia en la Figura4.5. Al tener menos elementos que detectar, pues pasan menos coches, el modelo se llega a procesarse a 23 fps en Titan, en Acer continua con un fps.

El último vídeo que se ha probado es el de España. Como se dice en la sección anterior este escenario tiene arboles y sombrillas que dificultan la visión de la cámara, además fue grabado con zoom lo que ha disminuido la calidad de imagen. Con esto comprobamos que el modelo no reconoce tantos elementos como en los otros vídeos, también es cierto que la dificultad en este escenario es mayor. En general se aprecia que detecta mejor a las personas que están en la parte inferior de la imagen, es decir más cerca, y menor frecuentemente a aquellas más lejanas. Los coches se van detectando a intervalos, debido a las sombrillas que obstaculizan la visión de la carretera como se puede ver en las

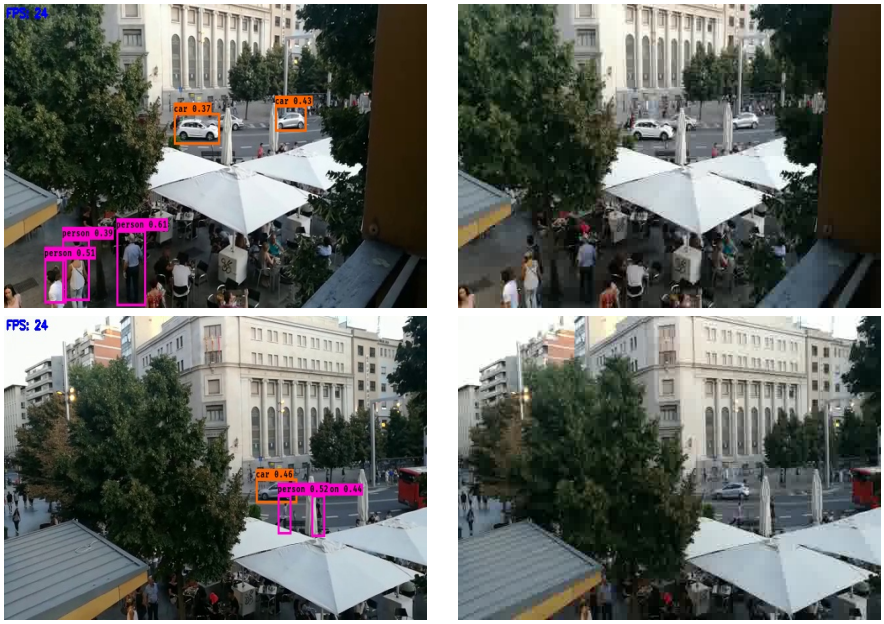


Figura 4.6: Capturas de la detección llevada a cabo en el vídeo España con el modelo VIRAT

imágenes de la Figura 4.6. Igual que con el vídeo anterior en el ordenador Acer se mantiene a un fps y se alcanzan los 23 fps en Titan, puesto que es este escenario se reconocen menos objetos.

En general se puede resumir que el modelo ha dado buen resultado en las calles con coches y peatones. Aunque sufre más con algunos escaparates, donde reconoce algunos elementos que no están, y las motocicletas, que generalmente las confunde con coches e incluso con personas.

4.4. Evaluación del prototipo final

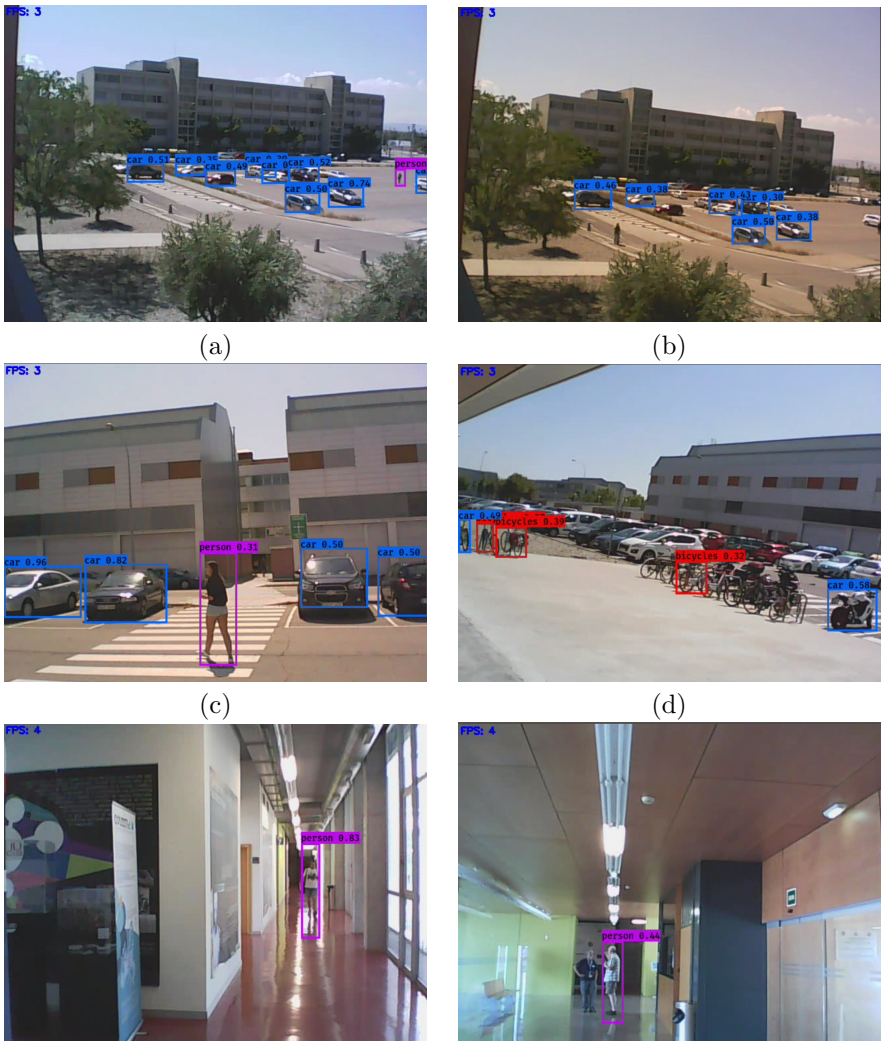
Se ha querido probar la eficiencia del modelo VIRAT ya utilizado en los vídeos propios, en el prototipo desarrollado en el capítulo 3. Se ha utilizado el programa, nombrado en dicho capítulo, capaz de grabar un vídeo con la detección de objetos a través de la webcam, para llevar el prototipo por el edificio del Instituto de Ingeniería e Investigación de Aragón y analizar el funcionamiento del modelo en la NVIDIA Jetson AGX Xavier.

En las imágenes (a) y (b) de la Figura 4.7 se pueden ver dos capturas de la grabación desde el primer piso del I3A, consiguiendo cierta altura para obtener ese ángulo ligeramente picado que tienen casi todos los vídeos del *dataset* VIRAT. Podemos comprobar que la precisión obtenida es bastante buena, pues detecta casi todos los coches e incluso alguna persona que pasa. Sin embargo también hay un momento donde a la cámara le llega una luz más cálida, como en la imagen de la derecha, y deja de detectar algún elemento. Por lo que se aprecia que la luminosidad de la escena también puede mejorar o empeorar la detección.

En la las imágenes (c) y (d) de la Figura 4.7 se ha grabado desde el la puerta del edificio, cambiando así el enfoque del escenario a uno frontal. Cuando se observan los resultados se comprueba que se sigue obteniendo una buena detección, aunque deje sin reconocer mas elementos que con el ángulo de visión anterior. También ha sido capaz de detectar alguna bicicleta a pesar de tener menos acierto con esta categoría.

Mientras se recorría el edificio se seguía grabando con el prototipo y se ha podido observar que al cruzarse con personas reconocía a la mayoría a pesar de ser un enfoque muy distinto al que se tenia en los escenarios del *dataset* de entrenamiento. Se puede ver en las imágenes (e) y (f) de la Figura 4.7.

La detección en estos vídeos en directo se lleva a cabo a 3 fps en la NVIDIA Jetson AGX Xavier. Esta velocidad de procesado no llega a procesar en tiempo real todos los *frames* de un vídeo, detectara objetos que permanezcan al menos unos 0.3 segundos en el campo de visión. Dadas las distancias típicas de las cámaras de monitorización y vigilancia y los objetos a detectar, esto parece una restricción común que permitiría detectar la mayoría de los objetos de interés. Sin embargo queda abierta la opción futura de optimizar más, para poder procesar todos los *frames* de un vídeo.



(a) (b) (c) (d) (e) (f)

Figura 4.7: Capturas de los vídeos capturados en el edificio del I3A con los resultados del modelo VIRAT superpuestos. (a), (b) Vídeo grabado por el prototipo desde el primer piso del I3A. (c), (d)Vídeo grabado por el prototipo desde la puerta del I3A. (e), (f) Vídeo grabado por el prototipo en los pasillos del I3A.

Capítulo 5

Conclusiones

En este capítulo se recogen las conclusiones sacadas del desarrollo de este trabajo. También se incluyen algunas mejoras y el trabajo futuro que podría derivar de este proyecto.

5.1. Conclusiones

Los sistemas embebidos, tan frecuentes en la vida cotidiana de cualquier persona, son pequeños dispositivos con una o pocas funciones dirigidas a una única aplicación, lo que los hace muy eficientes. En este proyecto se ha implementado en uno de estos sistemas un algoritmo de *deep learning* para llevar a cabo un reconocimiento visual centrado en sistemas de vigilancia, haciendo hincapié en modelos de *deep learning* eficientes. Para ello se ha seleccionado estudiar el algoritmo YOLOv3, el cual se ha verificado que resulta muy adecuado para aplicaciones con requisitos de eficiencia temporal. El *dataset* seleccionado, VIRAT, ha resultado adecuado para entrenar un modelo para la detección de objetos de utilidad en servicios de vigilancia y seguridad, sobretodo para el reconocimiento de personas y coches, que al salir frecuentemente en este tipo de vídeos, tenían más datos para el entrenamiento.

Se llevaron a cabo distintas pruebas y análisis de los modelos, realizando cuatro variaciones del modelo de detección de objetos, diferenciadas sobre todo por el conjunto de datos de entrenamiento. De esta manera se pudo observar que las clases de personas y coches del *dataset* se reconocían mucho mejor que las otras tres clases restantes, vehículos, objetos y bicicletas. Una de las principales razones de estos resultados es la cantidad de datos de entrenamiento de cada clase, además de la dificultad extra que tiene por ejemplo la categoría de objetos, que como se comprobó eran muy pequeños y muy variados. Las pruebas también sirvieron para ver que, si bien es mejor a la hora de evaluar un modelo separar los datos por sus distintos escenarios, cuando esto se llevaba a cabo el modelo no funcionaba tan bien fuera de calles y aparcamientos. Sin embargo al incluir todos los escenarios en el entrenamiento, se consiguió que el modelo fuese más flexible y se adaptara a más ambientes, obteniendo una detección más robusta.

Tras estos experimentos se ha concluido que el modelo más eficaz para construir el prototipo de vigilancia es la variación entrenada con todo el *dataset*

VIRAT, que ha sido entrenada con todos los escenarios posibles. Este modelo obtuvo un 32.6 % de mAP en la clase de personas y un 46.9 % en coches, siendo estos los mejores resultados. Comparando este resultado con el modelo existente de la implementación de YOLOv3 en Keras entrenado para propósito general, este modelo mejora las clases de persona y coche, aunque solo ligeramente, y añade la variante de objeto como cualquier elemento que sea o haya sido llevado por una persona. La evaluación del modelo obtenido en vídeos propios y sobretodo en el prototipo realizado con el sistema embebido, han dado resultados más satisfactorios, detectando un gran número de elementos de interés. Con esto vemos que el modelo entrenado generaliza satisfactoriamente a otros escenarios reales totalmente independientes de los *dataset* utilizados para el entrenamiento. Se ha conseguido finalmente que el modelo detecte objetos en el sistema embebido a 3 fps, que con las distancias de los vídeos de vigilancia logra detectar la mayoría de objetos de interés, concluyendo que el modelo resulta eficiente en el sistema.

5.2. Conclusiones personales

A nivel personal estoy muy contenta con lo aprendido en este trabajo. Durante el desarrollo del proyecto he profundizado mi aprendizaje en asignaturas de las que antes solo conocía las bases, sobretodo en visión por computador un campo que llamaba mucho mi atención y ahora que conozco más de él, me interesa todavía más. También he mejorado en el área de programación, puesto que he tenido que pensar pequeños programas que me ayudasen en tareas básicas como extraer los *frames* de los vídeos, grabar un vídeo en vivo con detección de objetos para visualizarlo después y sobretodo realizar cambios en los formatos de las anotaciones. Con estos programas siempre salían errores, algunos con una solución más sencilla que otros, pero he aprendido a encontrarlos, buscar la solución adecuada e implementarla las veces que hiciera falta. Por último, decir que al realizar el proyecto en el laboratorio del grupo RoPeRT, he podido apreciar el trabajo de un grupo de investigación, informarme sobre sus métodos de trabajo y obtener consejos y ayuda con la que he podido adquirir mayor conocimiento de forma muy satisfactoria y entretenida.

5.3. Trabajo futuro

Con la idea de implementar en un futuro el modelo en cámaras de vigilancia, una mejora interesante seria añadir más datos de entrenamiento de las clases bicicletas, objetos y sobretodo vehículos. Puesto que al final las clases que detecta son principalmente personas y coches y con las otras tres clases tiene más dificultades. Si el modelo se usara finalmente en el servicio de vigilancia y seguridad, seria una buena idea que además de llevar a cabo la detección de objetos, pudiese realizar un seguimiento de estos. De manera que se podría observar las trayectorias de personas y vehículos y extraer con ellos datos y patrones de movilidad y comportamiento. Entonces se podría estudiar llevar a cabo además del algoritmo de reconocimiento de objetos, uno de seguimiento e intentar que afecte lo menos posible a la eficiencia del modelo y a los *frames* por segundo que puede procesar el sistema embebido.

Apéndice A

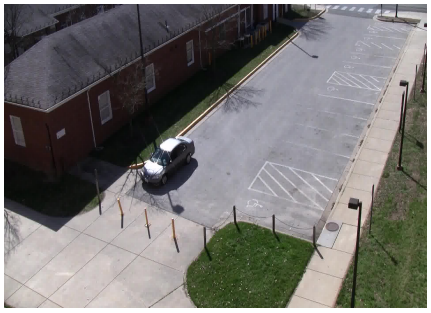
Capturas de los *dataset*

En este anexo se recogen imágenes representativas de los dos *dataset* que se han utilizado en el trabajo, los vídeos de VIRAT y los propios de ZARAGOZA.

A.1. *Dataset* VIRAT

VIRAT presenta 315 vídeos de datos pertenecientes a 11 escenarios diferentes. Aquí se muestran esas 11 escenas y un conjunto de imágenes con ejemplos correspondientes a cada clase del *dataset*.

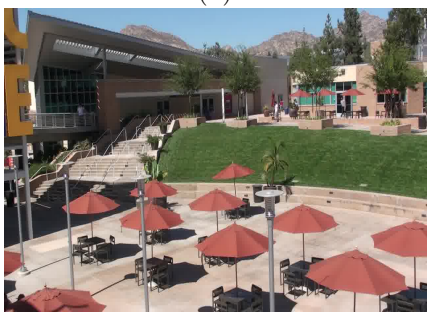
A.1.1. Escenarios



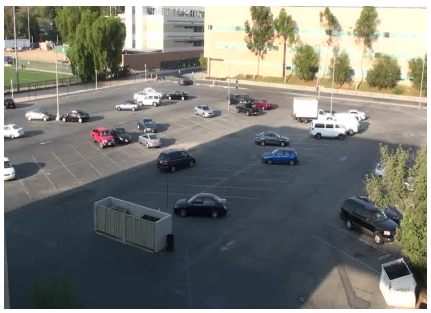
(a)

(b)

(c)



(d)

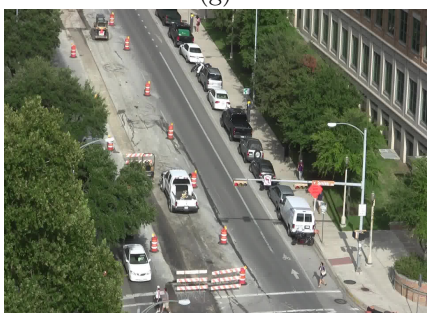


(e)

(f)

(g)

(h)



(i)

(j)

(k)

Figura A.1: Imágenes de los escenarios del *dataset* VIRAT. (a) Escenario 0000. (b) Escenario 0001. (c) Escenario 0002. (d) Escenario 0100. (e) Escenario 0101. (f) Escenario 0102. (g) Escenario 0400. (h) Escenario 0401. (i) Escenario 0500. (j) Escenario 0502. (k) Escenario 0503.

A.1.2. Clases

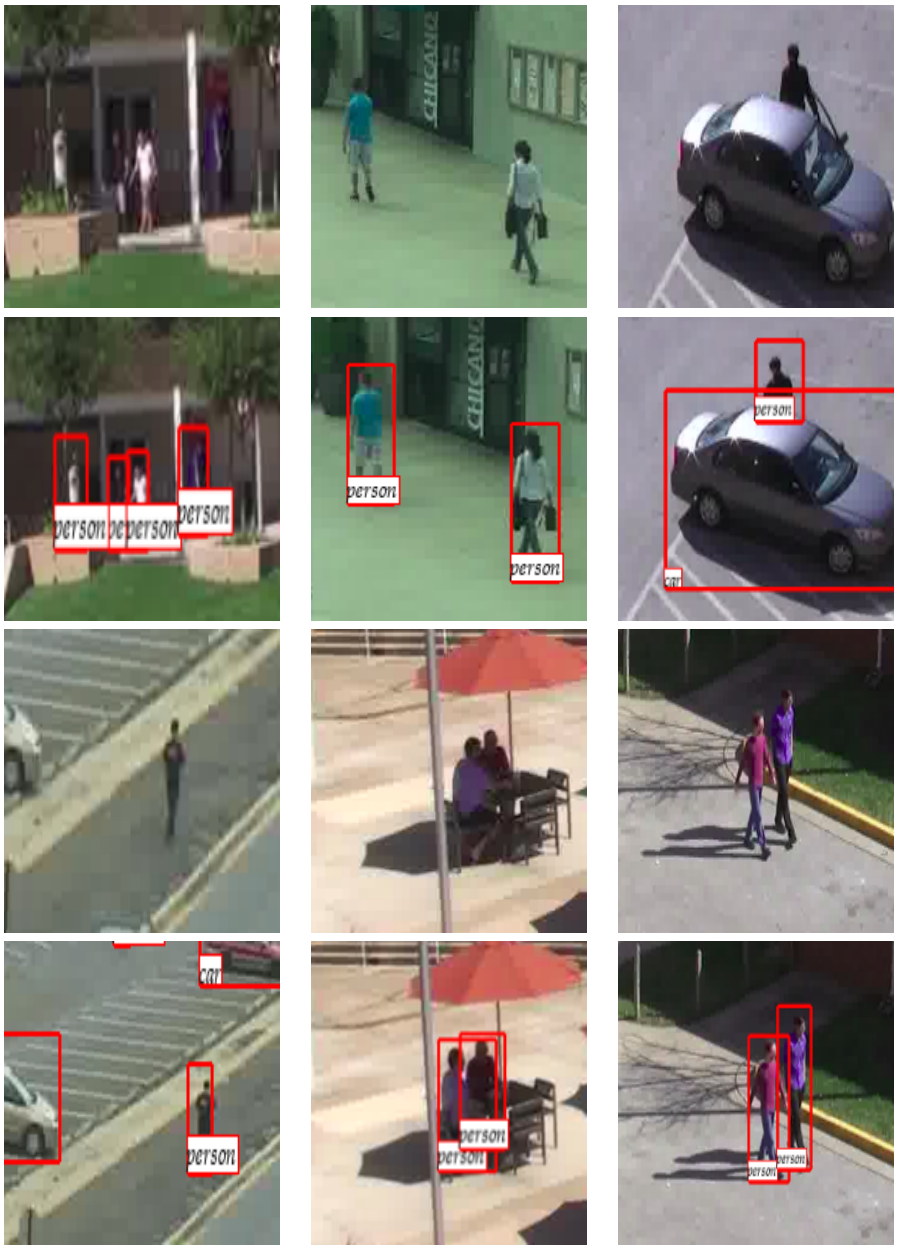


Figura A.2: Ejemplos de la categoría personas del *dataset* VIRAT.

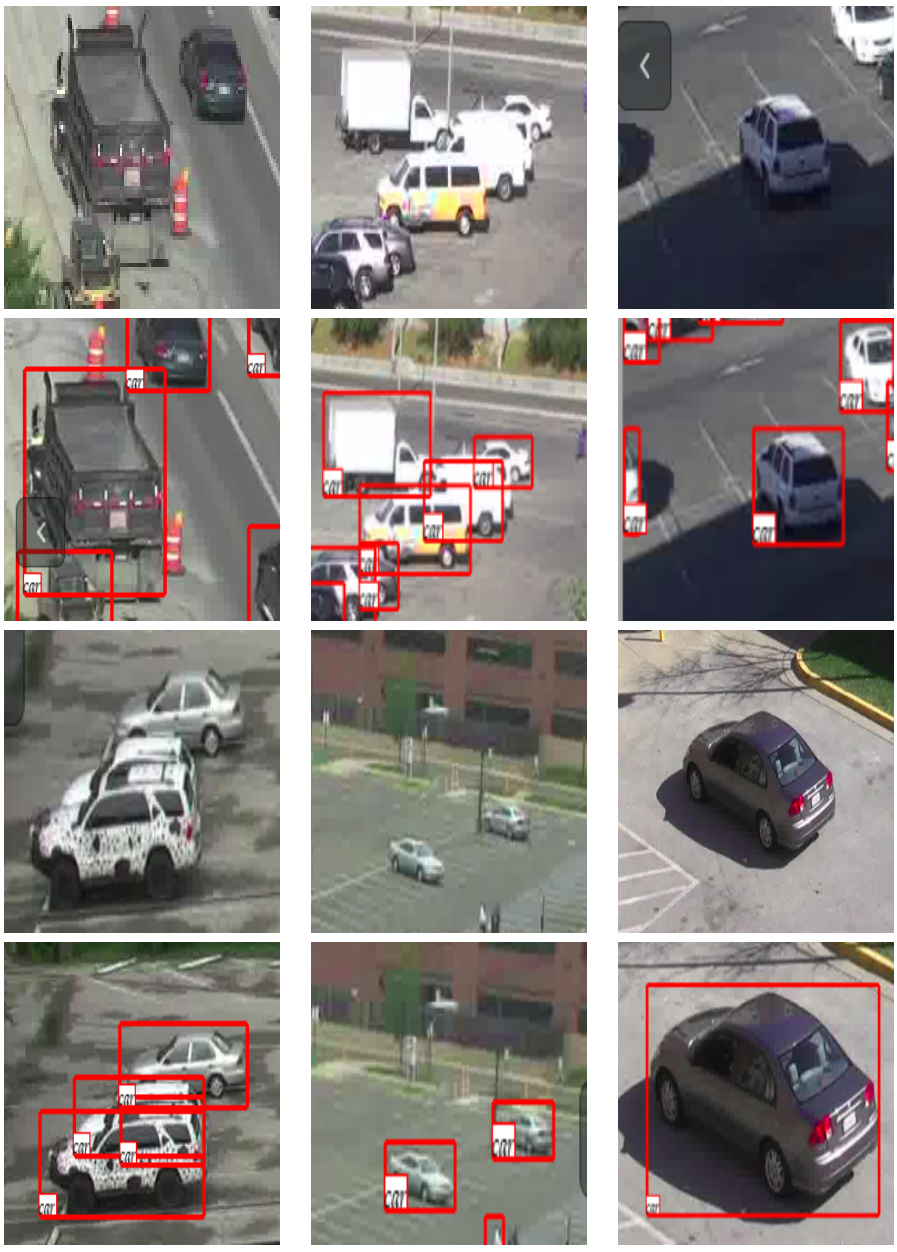


Figura A.3: Ejemplos de la categoría coches del *dataset* VIRAT.

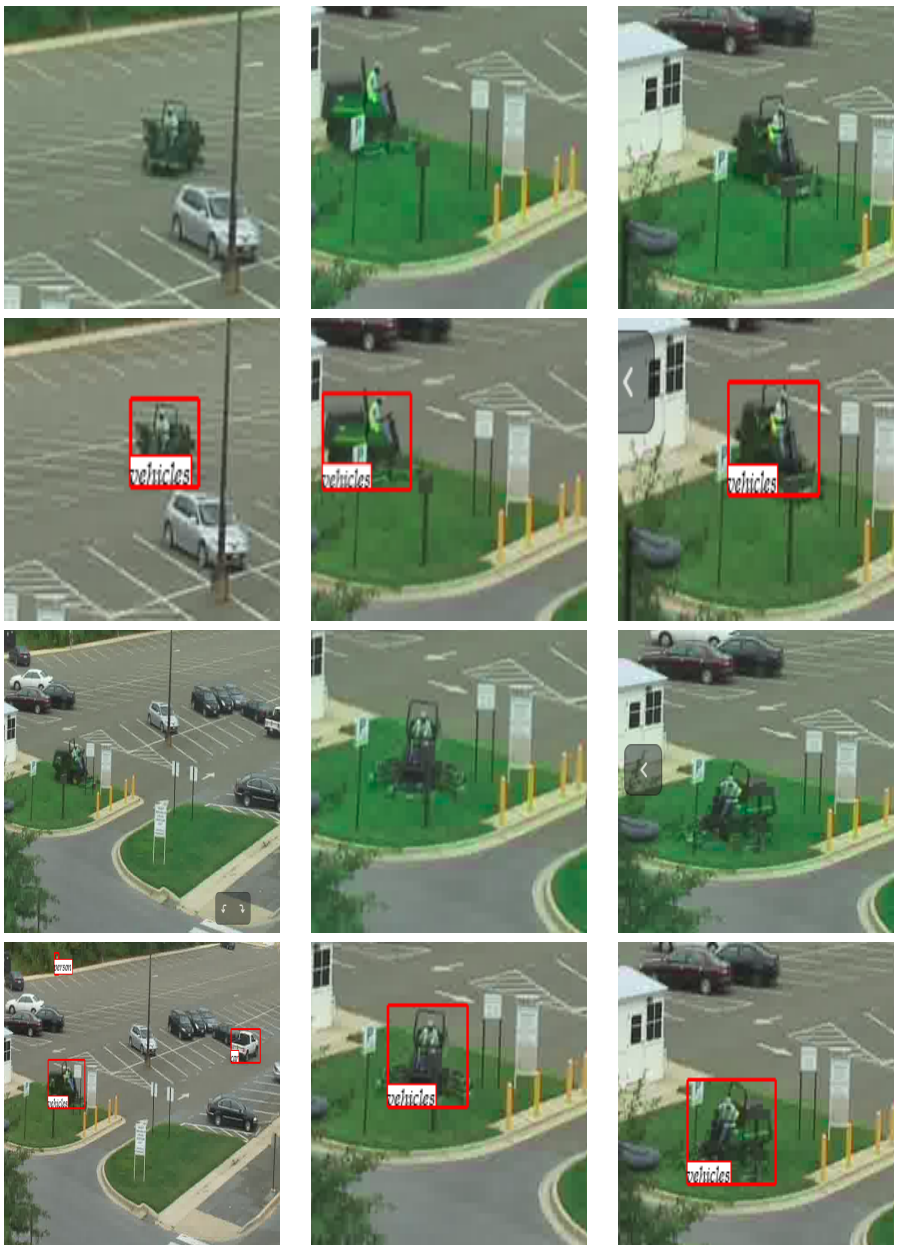


Figura A.4: Ejemplos de la categoría vehículo del *dataset* VIRAT.

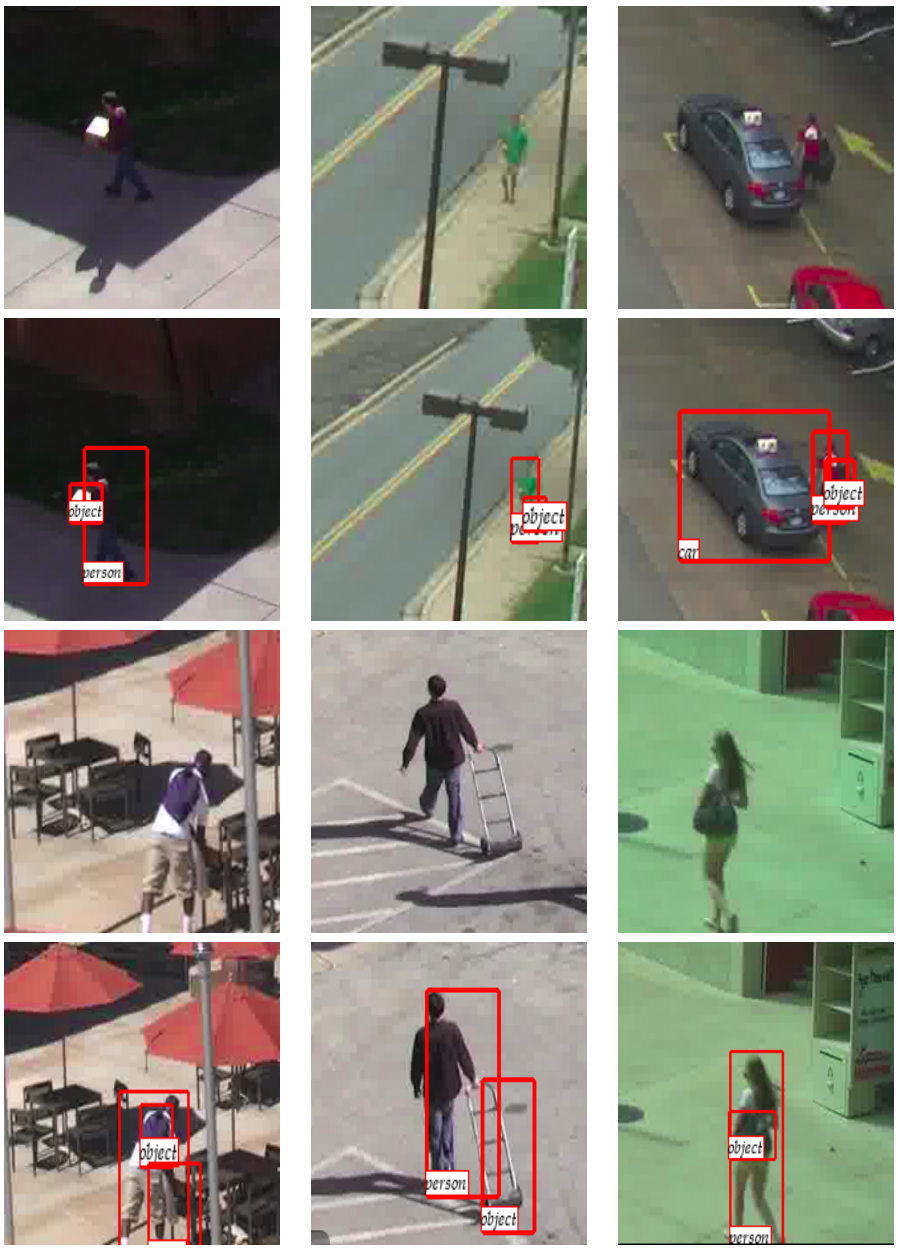


Figura A.5: Ejemplos de la categoría objetos del *dataset* VIRAT.

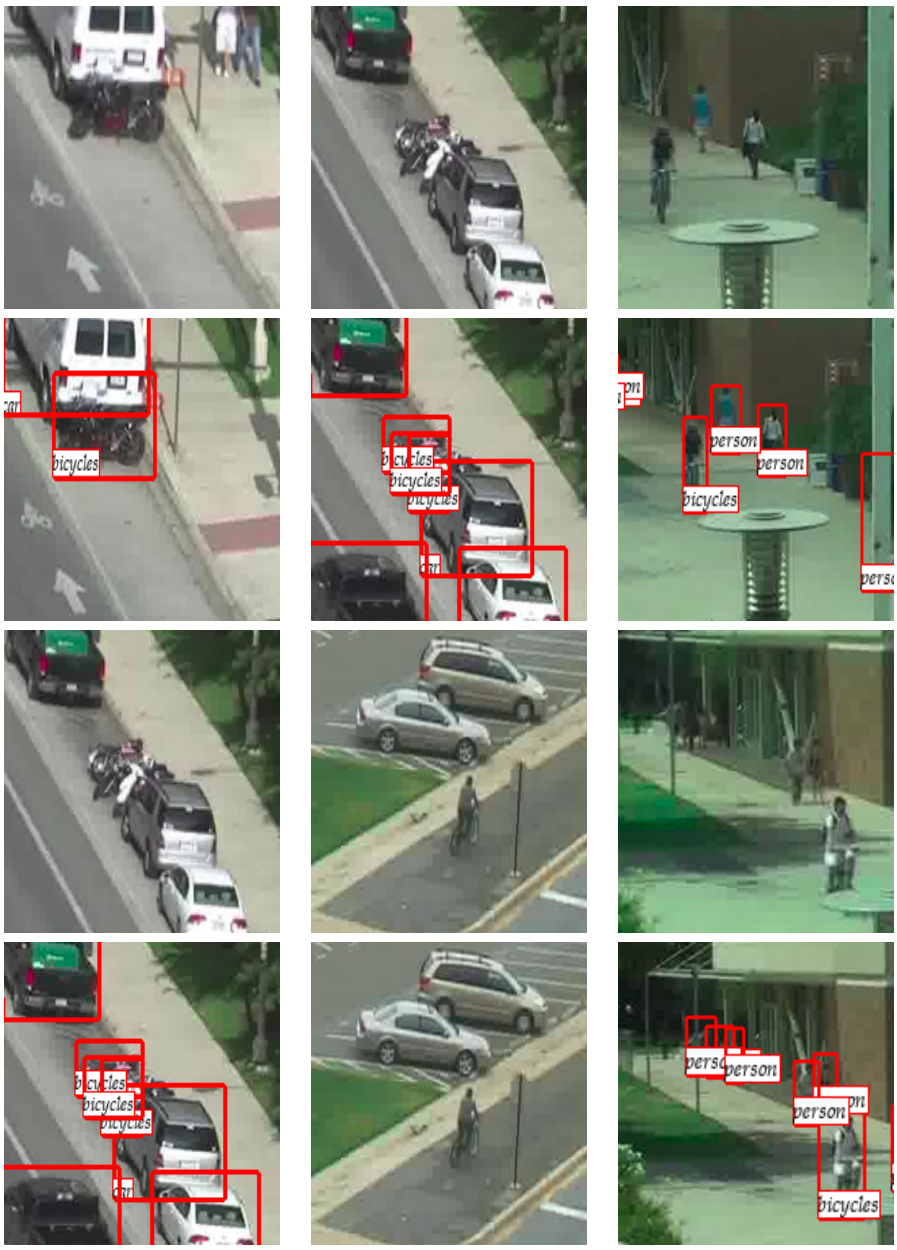


Figura A.6: Ejemplos de la categoría bicicletas del *dataset* VIRAT.

A.2. *Dataset* Zaragoza

Imágenes del *dataset* propio creado con cuatro vídeos de las calles de la cuidad de Zaragoza.

Figura A.7: Imágenes de la calle Santander, Zaragoza.

Figura A.8: Imágenes de la calle Francisco de Goya, Zaragoza.

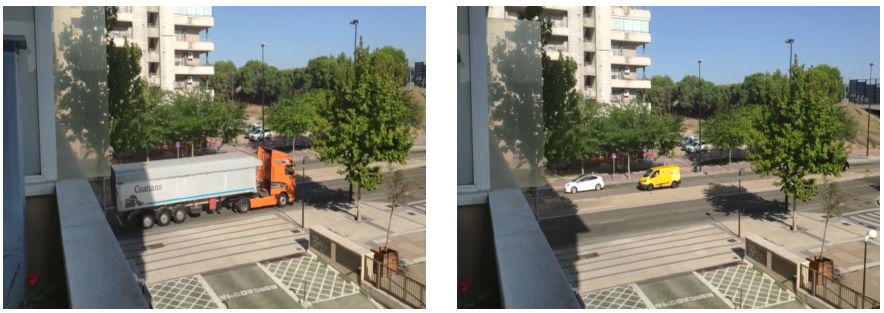


Figura A.9: Imágenes de la avenida Academia Militar General, Zaragoza.

Figura A.10: Imágenes de la plaza la España, Zaragoza.

Apéndice B

Resultados de los experimentos

En este anexo se recogen resultados adicionales más detallados obtenidos para los vídeos de validación de los conjuntos de datos VIRAT-Escenarios y VIRAT-Aleatorio. Se muestran los resultados en detalle de los cinco vídeos analizados individualmente en el capítulo 4, como una muestra representativa de los 182 vídeos analizados.

En el capítulo se muestran los resultados resumidos, aquí se muestra el nombre de cada vídeo, los mAP y *recall* sacados como media de todas las clases y después para cada una de las cinco clases existentes.

También en el capítulo 4 se encuentran explicados los parámetros AP (*Average Precision*), que indica el promedio del valor de la precisión obtenida a cada intervalo de *recall*, y la IoU (*Intersection over union*) que evalúa la superposición entre 2 dos cajas, en este caso la de las anotaciones del *dataset* y las de predicción calculadas por el modelo. Siguiendo con los parámetros que encontramos en los resultados, queda decir que el área es el tamaño de objeto para el que realiza la métrica y maxDets es el numero de detecciones que lleva a cabo para realizar el promedio.

Al final del análisis de cada vídeo, se encuentran dos vectores. En el primero de ellos se muestran el número del conjunto de datos de validación de cada clase. En el segundo el número de datos predichos por el modelo, igualmente para cada clase. En los dos vectores, los números pertenecen a la clase persona, coche, vehículo, objeto y bicicleta es ese mismo orden.

B.1. Resultados de de validación en el subconjunto VIRAT-Escenarios

Evaluación del vídeo VIRAT_S_010003_08_000739_000778 Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=0.026

Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=0.104

Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=0.010

Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=0.051
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=0.020
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= 0.059
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=0.072
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=0.072
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area= large maxDets=100]=-1

Evaluación de la categoría persona en el vídeo

Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=0.026
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=0.104
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=0.010
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=0.051
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=0.020
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= 0.059
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=0.072
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=0.072
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1

Evaluación de la categoría coche en el vídeo

Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= =-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1

Evaluación de la categoría vehículo en el vídeo

Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=-1

Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= =-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Evaluación de la categoría objeto en el vídeo
Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large — maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= =-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Evaluación de la categoría bicicleta en el vídeo
Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= =-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1
[29426, 64447, 0, 634, 209] [116460, 98225, 4058, 46541, 44008]

Evaluación del vídeo VIRAT_S_050300_09_001789_001858 Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=0.139
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=0.376

Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=0.051
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=0.193
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=0.070
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= 0.156
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=0.205
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=0.205
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1

Evaluación de la categoría persona en el vídeo

Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=0.091
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=0.279
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=0.014
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=0.120
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=0.122
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= 0.138
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=0.139
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=0.139
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1

Evaluación de la categoría coche en el vídeo

Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=0.186
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=0.474
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=0.088
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=0.266
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=0.019
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= 0.174
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=0.272
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=0.272
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1

Evaluación de la categoría vehículo en el vídeo

Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=-1

Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= =-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Evaluación de la categoría objeto en el vídeo
Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= =-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Evaluación de la categoría bicicleta en el vídeo
Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= =-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1
[29426, 64447, 0, 634, 209] [116460, 98225, 4058, 46541, 44008]

B.2. Resultados de de validación en el subconjunto VIRAT-Aleatorio

Evaluación del vídeo VIRAT_S_050000_03_000585_000639 Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=0.185

Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=0.402
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=0.167
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=0.362
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=0.045
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= 0.310
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=0.363
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=0.363
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1

Evaluación de la categoría persona en el vídeo

Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=0.088
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=0.365
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=0.002
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=0.195
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=0.042
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= 0.182
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=0.199
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=0.199
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1

Evaluación de la categoría coche en el vídeo

Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=0.566
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=0.905
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=0.651
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=0.638
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=0.061
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= 0.511
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=0.638
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=0.638

Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Evaluación de la categoría vehículo en el vídeo
Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= =-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Evaluación de la categoría objeto en el vídeo
Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=0.018
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=0.051
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=0.004
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=0.291
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=0.000
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= 0.241
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=0.293
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=0.293
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Evaluación de la categoría bicicleta en el vídeo
Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=0.067
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=0.285
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=0.013
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=0.325
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=0.077
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= 0.308
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=0.324
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=0.324
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1

Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1
[30970, 72694, 0, 1254, 1187] [132355, 86941, 0, 44836, 17766]

Evaluación del vídeo VIRAT_S_010107_01_000068_000196 Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=0.110
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=0.293
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=0.054
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=0.145
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=0.033
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= 0.133
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=0.209
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=0.209
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Evaluación de la categoría persona en el vídeo
Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=0.047
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=0.208
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=0.004
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=0.051
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=0.065
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= 0.161
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=0.199
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=0.199
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Evaluación de la categoría coche en el vídeo
Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=0.280
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=0.651
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=0.158
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=0.382
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=0.021
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= 0.193
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=0.383

Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=0.383
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1

Evaluación de la categoría vehículo en el vídeo

Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= =-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1

Evaluación de la categoría objeto en el vídeo

Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= =-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1

Evaluación de la categoría bicicleta en el vídeo

Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=0.002
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=0.019
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=0.000
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=0.002
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=0.013
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= 0.045
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=0.045
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=0.045

Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1
[30970, 72694, 0, 1254, 1187] [132355, 86941, 0, 44836, 17766]

Evaluación del vídeo VIRAT_S_050202_03_000608_000684 Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=0.044
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=0.114
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=0.042
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=0.081
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=0.051
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= 0.076
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=0.076
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=0.076
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Evaluación de la categoría persona en el vídeo
Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=0.024
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=0.106
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=0.000
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=0.059
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=0.020
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= 0.057
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=0.057
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=0.057
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Evaluación de la categoría coche en el vídeo
Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=0.065
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=0.122
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=0.083
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=0.102
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=0.082
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= 0.095

Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=0.095
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=0.095
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Evaluación de la categoría vehículo en el vídeo
Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= =-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Evaluación de la categoría objeto en el vídeo
Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= =-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Evaluación de la categoría bicicleta en el vídeo
Average Precision (AP) @[IoU=0.50:0.95 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.50 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.75 area=all maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=small maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Precision (AP) @[IoU=0.50:0.95 area=large maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=1]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=10]= =-1
Average Recall (AR) @[IoU=0.50:0.95 area=all maxDets=100]=-1

Average Recall (AR) @[IoU=0.50:0.95 area=small maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=medium maxDets=100]=-1
Average Recall (AR) @[IoU=0.50:0.95 area=large maxDets=100]=-1
[30970, 72694, 0, 1254, 1187] [132355, 86941, 0, 44836, 17766]

Bibliografía

[1] Damián Jorge Matich. Redes neuronales: Conceptos básicos y aplicaciones. *Universidad Tecnológica Nacional, México*, 2001.

[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[3] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[4] R Johnson. Microsoft, google beat humans at image recognition. *EE Times*, 2015.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[6] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[7] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C Berg. Dssd: Deconvolutional single shot detector. *arXiv preprint arXiv:1701.06659*, 2017.

[8] Weiming Hu, Tieniu Tan, Liang Wang, and Steve Maybank. A survey on visual surveillance of object motion and behaviors. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 34(3):334–352, 2004.

[9] Amira Ben Mabrouk and Ezzeddine Zagrouba. Abnormal behavior recognition for intelligent video surveillance systems: A review. *Expert Systems with Applications*, 91:480–491, 2018.

[10] David Gay, Philip Levis, Robert Von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesc language: A holistic approach to networked embedded systems. *Acm Sigplan Notices*, 49(4):41–51, 2014.

[11] Rajeshwari Banakar, Stefan Steinke, Bo-Sik Lee, Mahesh Balakrishnan, and Peter Marwedel. Scratchpad memory: A design alternative for cache on-chip

memory in embedded systems. In *Proceedings of the Tenth International Symposium on Hardware/Software Codesign. CODES 2002 (IEEE Cat. No. 02TH8627)*, pages 73–78. IEEE, 2002.

[12] Srivaths Ravi, Anand Raghunathan, Paul Kocher, and Sunil Hattangady. Security in embedded systems: Design challenges. *ACM Transactions on Embedded Computing Systems (TECS)*, 3(3):461–491, 2004.

[13] Sravanthi Chalasani and James M Conrad. A survey of energy harvesting sources for embedded systems. In *IEEE SoutheastCon 2008*, pages 442–447. IEEE, 2008.

[14] Wayne Wolf, Burak Ozer, and Tiehan Lv. Smart cameras as embedded systems. *computer*, (9):48–53, 2002.

[15] Eduardo Romera, José M Alvarez, Luis M Bergasa, and Roberto Arroyo. Erfnet: Efficient residual factorized convnet for real-time semantic segmentation. *IEEE Transactions on Intelligent Transportation Systems*, 19(1):263–272, 2017.

[16] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[17] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[18] Frank Seide and Amit Agarwal. Cntk: Microsoft's open-source deep-learning toolkit. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2135–2135. ACM, 2016.

[19] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[20] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[21] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[22] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[23] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[24] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[25] Jingwen Li, Lei Huang, and Changping Liu. Robust people counting in video surveillance: Dataset and system. In *2011 8th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 54–59. IEEE, 2011.

[26] Bo Li, Tianfu Wu, and Song-Chun Zhu. Integrating context and occlusion for car detection by hierarchical and-or model. In *European Conference on Computer Vision*, pages 652–667. Springer, 2014.

[27] Sangmin Oh, Anthony Hoogs, Amitha Perera, Naresh Cuntoor, Chia-Chih Chen, Jong Taek Lee, Saurajit Mukherjee, JK Aggarwal, Hyungtae Lee, Larry Davis, et al. A large-scale benchmark dataset for event recognition in surveillance video. In *CVPR 2011*, pages 3153–3160. IEEE, 2011.

[28] Roberto Ierusalimschy, Luiz Henrique De Figueiredo, and Waldemar Celes Filho. Lua—an extensible extension language. *Software: Practice and Experience*, 26(6):635–652, 1996.