

Universidad de Alcalá Escuela Politécnica Superior

Máster Universitario en Ingeniería Industrial

Trabajo Fin de Máster

Diseño, implementación y evaluación de una estrategia de
detección de objetos abandonados en aplicaciones de
videovigilancia

ESCUELA POLITECNICA
SUPERIOR

Autor: Jesús Mudarra Luján

Tutor: Javier Macías Guarasa

2021

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Máster Universitario en Ingeniería Industrial

Trabajo Fin de Máster

**Diseño, implementación y evaluación de una estrategia de
detección de objetos abandonados en aplicaciones de
videovigilancia**

Autor: Jesús Mudarra Luján

Tutor: Javier Macías Guarasa

Tribunal:

Presidente: Carlos Luna Vázquez

Vocal 1º: Sergio Lafuente Arroyo

Vocal 2º: Javier Macías Guarasa

28 de julio de 2021

A mis padres y mi hermano Carlos por su incondicional apoyo

“Si tus sueños no te asustan, no son lo suficientemente grandes.”
Richard Branson

Agradecimientos

Este proyecto trae consigo el punto y final de mi etapa como estudiante universitario. Han supuesto muchísimas horas de investigación y trabajo y, en mayor o menor medida, quiero agradecer el apoyo recibido a todas las personas que han participado en esta aventura.

Mención especial a mi tutor Javier Macías y a Marta Marrón por todo el apoyo y sabiduría recibida durante este tiempo. Les agradezco enormemente haber podido contar con ellos para desarrollar un tema que fue totalmente desconocido para mí al principio. Han sacado tiempo para orientarme siempre que lo he necesitado ante todos los baches que me he ido encontrado a lo largo de este proyecto.

A mi amigo Justo ya que, desde el primer día que comencé este trabajo, mostró un gran interés y apoyo absoluto. Me ha enseñado a ver desde otra perspectiva los problemas que me he encontrado durante el camino y afrontarlos de la mejor forma posible. Agradecerle también que haya sido partícipe en la infinitas evaluaciones de los algoritmos.

A mis amigos del Máster, con especial mención a Juanjo, Luis, Sergio y Nacho. Personas maravillosas, de las que no puedes dejar escapar, con las que he compartido innumerables experiencias en estos dos últimos años.

Por último, a los pilares más fundamentales de mi vida, mis padres y mi hermano Carlos. Sin el apoyo diario que me han dado, esto no habría sido posible. Gracias de corazón por aguantarme estos últimos meses, levantarme el ánimo cuando lo he necesitado y por mostrar interés por lo que estaba desarrollando.

Resumen

Este trabajo aborda el estudio e implementación de algoritmos de aprendizaje profundo (*Deep Learning*) con la finalidad de detectar objetos abandonados en aplicaciones de videovigilancia.

Se ha realizado un estudio teórico de los algoritmos de detección y seguimiento disponibles en el Estado del Arte. Para la detección de objetos en tiempo real se ha empleado YOLOv4 [1]. Como algoritmo de seguimiento se ha optado por Deep SORT [2]. Por último, se ha desarrollado un algoritmo que determine si un objeto ha sido abandonado o no. Todos ellos han sido implementados sobre el dataset de referencia MS COCO [3] y evaluados sobre los datasets más relevantes en la detección de objetos abandonados como son GBA2018 [4], PETS2007 [5], AVSSAB2007 [6] o ABODA [7].

Palabras clave: Deep Learning, YOLOv4, Deep SORT, videovigilancia, visión por computadora.

Abstract

This Master's Thesis proposes the study and implementation of Deep Learning algorithms in order to detect abandoned objects in video surveillance applications.

A theoretical study of the detection and monitoring algorithms available in the State of the Art has been carried out. YOLOv4 [1] has been used to detect objects in real time. Deep SORT [2] has been chosen as tracking algorithm. Finally, an algorithm has been developed to determine when an object has been abandoned or not. All of them have been implemented on the MS COCO [3] benchmark dataset and evaluated on the most relevant datasets in the detection of abandoned objects such as GBA2018 [4], PETS2007 [5], AVSSAB2007 [6] or ABODA [7].

Keywords: Deep Learning, YOLOv4, Deep SORT, Video Surveillance, Computer Vision.

Resumen extendido

La detección de objetos abandonados se trata de una de las aplicaciones más importantes dentro de los sistemas de detección por videovigilancia en los últimos años [8]. La demanda de detección de objetos abandonados está al alza y se precisa disponer de aplicaciones capaces de detectar y evaluar conductas en tiempo real y con márgenes de error reducidos. Este trabajo pretende cubrir una de las etapas de desarrollo en el rastreo, la asociación entre persona y objeto, con la finalidad de poder identificar al propietario y determinar si ha abandonado el objeto o no.

En este trabajo se realizó un estudio exhaustivo del Estado del Arte actual en estrategias para abordar la problemática de la detección de objetos abandonados mediante el uso de aplicaciones de videovigilancia.

Se ha implementado y evaluado YOLOv4 [1], un algoritmo de detección de objetos que hace uso de una única red neuronal convolucional para detectar objetos a partir de imágenes. Esta red neuronal, que está previamente entrenada con el dataset de MS COCO [3], ha vuelto a ser entrenada con el dataset Open Images Dataset v4 [9] con el objetivo de que solo detecte ciertos objetos de interés: personas, mochilas, bolsas de mano y maletas. Tras el entrenamiento se han calculado las métricas de calidad más utilizadas en la evaluación de algoritmos de detección de objetos para determinar si se superan las del modelo pre-entrenado del dataset de MS COCO.

El dataset de referencia con el que se obtuvieron mejores métricas en YOLOv4 fue MS COCO. Posteriormente se han realizado evaluaciones sobre los datasets de detección de objetos abandonados PETS2007 [5], AVSSAB2007 [6], GBA2018 [4] y ABODA [7].

En base a YOLOv4 se ha realizado un estudio en el Estado del Arte de los algoritmos de seguimiento más actuales con la finalidad de que asignar una identidad a cada detección. Se ha implementado el algoritmo Deep SORT [2] junto a YOLOv4. Deep SORT es un algoritmo predecesor de SORT [10], que realiza un seguimiento basado en la detección, realizando los procesos de predicción y actualización con filtros de Kalman. Empleando este algoritmo de seguimiento se ha podido rastrear el movimiento de las personas y los objetos asignándoles una identidad única. Del mismo modo que en el algoritmo de detección, se ha evaluado su funcionamiento sobre los datasets más relevantes.

Posteriormente se ha diseñado, implementado y evaluado un algoritmo que determine si un objeto ha sido abandonado o no en base a los algoritmos de detección y seguimiento antes nombrados. Para ello, se ha calculado en los 5 primeros segundos del vídeo la distancia existente entre las personas con todos los objetos de interés detectables. Con la distancia mínima que exista entre una persona y un objetos se puede establecer una asociación.

Obtenida la vinculación persona-objeto se puede evaluar el comportamiento calculando la distancia en píxeles a la que se encuentran en los siguientes fotogramas del vídeo, y así determinar si se produce un abandono del objeto. Otra posibilidad es que el objeto se encuentre durante el transcurso de todo el vídeo estático [11] en el mismo punto y sin una persona asociada. En este caso se puede deducir que ese objeto está abandonado sin posibilidad de detectar al propietario.

Con el desarrollo del algoritmo capaz de detectar objetos abandonados se ha evaluado los resultados en distintos escenarios, del mismo modo que con los algoritmos de detección y seguimiento, teniendo como métrica de calidad la tasa de fallos en la determinación de si un objeto ha sido abandonado.

Índice general

Resumen	v
Abstract	vii
Resumen extendido	ix
Índice general	xI
Índice de figuras	xv
Índice de tablas	xix
Índice de códigos	xxI
Lista de Acrónimos	xxIII
1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	3
1.3. Estructura de la memoria	4
2. Estudio teórico	5
2.1. Introducción	5
2.2. Estado del Arte	5
2.2.1. Segmentación de objetos en primer plano	5
2.2.2. Detección de objetos estacionarios	8
2.2.3. Detección de personas y objetos	9
2.2.4. Reconocimiento del comportamiento	11
2.3. Redes neuronales convolucionales (CNN)	14
2.4. Algoritmos de detección de objetos	16
2.4.1. Faster R-CNN	16
2.4.2. SSD: Single Shot MultiBox Detector	19
2.4.3. EfficientDet	21
2.4.4. YOLOv4	23
2.4.5. Comparativa de los diferentes detectores	25
2.5. Algoritmos de seguimiento de objetos	26
2.5.1. Seguimiento de un objeto	27
2.5.2. Seguimiento de múltiples objetos	27

2.5.3. Métodos tradicionales	28
2.5.4. Deep SORT	28
3. Desarrollo	33
3.1. Introducción	33
3.2. Detección de personas y objetos con YOLOv4	34
3.3. Datasets utilizados para el evaluación de YOLOv4	37
3.3.1. MS COCO Dataset	37
3.3.2. Open Images Dataset v4	38
3.4. Evaluación de las métricas de calidad de MS COCO dataset	39
3.5. Entrenamiento YOLOv4 con Open Image Dataset v4	41
3.5.1. Recopilación y etiquetado del dataset personalizado	41
3.5.2. Configuración de ficheros para el entrenamiento	42
3.5.3. Entrenamiento del dataset personalizado	44
3.6. Seguimiento de personas y objetos con YOLOv4 y Deep SORT	45
3.7. Algoritmo de detección de objetos abandonados	48
3.8. Conclusiones	52
4. Resultados	53
4.1. Introducción	53
4.2. Entorno experimental	53
4.2.1. Métricas de calidad	53
4.2.1.1. Intersección sobre la unión (IoU)	54
4.2.1.2. TP, TN, FP y FN	54
4.2.1.3. Precisión	55
4.2.1.4. Recall	55
4.2.1.5. F-Score	55
4.2.1.6. Precisión media	55
4.2.2. Datasets utilizados	55
4.2.2.1. PETS2007 Dataset	56
4.2.2.2. AVSSAB2007 Dataset	58
4.2.2.3. GBA2018 Dataset	59
4.2.2.4. ABODA Dataset	60
4.3. Resultados experimentales	60
4.3.1. Métricas de calidad en Open Image Dataset v4	61
4.3.2. Métricas de calidad en MS COCO Dataset	65
4.3.3. Resultados en detección de objetos y personas con YOLOv4	66
4.3.4. Resultados en seguimiento de objetos y personas con YOLOv4 y Deep SORT	67
4.3.5. Resultados en algoritmo de detección de objetos abandonados	69
4.4. Conclusiones	73
5. Conclusiones y líneas futuras	75
5.1. Conclusiones	75
5.2. Líneas futuras	76

Bibliografía	79
Apéndice A. Pliego de condiciones	85
A.1. Introducción	85
A.2. Características del equipo A	85
A.2.1. Especificaciones hardware del equipo A	85
A.2.2. Especificaciones software del equipo A	85
A.3. Características del equipo B	85
A.3.1. Especificaciones hardware del equipo B	86
A.3.2. Especificaciones software del equipo B	86
Apéndice B. Presupuesto	87
B.1. Introducción	87
B.2. Equipo de trabajo	87
B.3. Timing	87
B.4. Costes	88
B.4.1. Costes mano de obra	88
B.4.2. Recursos hardware	88
B.4.3. Recursos software	89
B.5. Presupuesto total	89
Apéndice C. Manual de usuario	91
C.1. Introducción	91
C.2. Guía de instalación	91
C.2.1. Instalación de Git	91
C.2.2. Instalación de Anaconda	91
C.2.3. Descarga de los repositorios del proyecto	92
C.2.4. Crear entorno virtual con Anaconda	93
C.2.5. Descargar los datasets	93
C.3. Guía de ejecución	94
C.3.1. Ejecutar algoritmo de detección de objetos YOLOv4	94
C.3.2. Ejecutar algoritmo de seguimiento de objetos YOLOv4 + Deep SORT	94
C.3.3. Ejecutar algoritmo de detección de objetos abandonados	95

Índice de figuras

1.1.	Persona cruzando del radio de 2 metros (marcado en amarillo) al radio de 3 metros (marcado en rojo) alrededor de su equipaje (marcado con una cruz verde) [12]	2
1.2.	Marco de referencia en detección de objetos abandonados [11]	2
1.3.	Diagrama de bloques del módulo de generación de candidatos [11]	3
2.1.	Proceso de la sustracción del fondo [13]	6
2.2.	Ejemplo de sustracción del fondo en aplicaciones de tráfico [13]	8
2.3.	Clasificación de sustracción del fondo basados en métodos de detección de objetos estacionarios [14]	8
2.4.	Clasificación de detección de personas enfocado a la detección de objetos [15]	9
2.5.	Clasificación de detección de personas según modelo de persona [15]	10
2.6.	Sistema de videovigilancia inteligente [16]	11
2.7.	Comportamientos anormales realizados por una sola persona [16]	13
2.8.	Comportamientos anómalos en multitudes	13
2.9.	Las CNN son un subconjunto de redes neuronales de Deep Learning [17]	14
2.10.	Ejemplo de convolución de datos de entrada bidimensionales [17]	15
2.11.	Ejemplo de convolución de datos de entrada bidimensionales [17]	16
2.12.	Arquitectura de Faster R-CNN [18]	17
2.13.	Variación de los anchor boxes Faster R-CNN [18]	18
2.14.	Arquitectura de una red neuronal convolucional con un detector SSD [19]	19
2.15.	Ejemplo de una cuadrícula 4x4 [19]	19
2.16.	Ejemplo con 2 anchor boxes [19]	20
2.17.	El cuadro delimitador del edificio 1 es más alto, mientras que el cuadro delimitador del edificio 2 es más ancho [19]	20
2.18.	Visualización de mapas de características de CNN y campo receptivo [19]	21
2.19.	Arquitectura de EfficientDet [20]	22
2.20.	Comparativa entre biFPN y las demás redes de características previas [20]	22
2.21.	Comparativa velocidad y precisión de YOLOv4 frente a otras arquitecturas [1]	23
2.22.	Arquitectura de detectores de objetos de una y dos etapas [1]	24
2.23.	Ejemplo seguimiento de una única persona [21]	27
2.24.	Ejemplo seguimiento de personas y objetos de interés en [4]	27
2.25.	Matching Cascade [2]	31
3.1.	Detección de objetos con YOLOv4 en Darknet en [5]	35
3.2.	Detección de objetos con YOLOv4 en Tensorflow en [4]	36

3.3. Detecciones de YOLOv4 con Darknet y Tensorflow. (a) Detección de YOLOv4 con Darknet. (b) Detección de YOLOv4 con Tensorflow	36
3.4. Categorías de objetos del dataset MS COCO [22]	37
3.5. MS COCO detección de objetos [22]	37
3.6. MS COCO segmentación semántica [22]	38
3.7. MS COCO detección de puntos clave [22]	38
3.8. Ejemplo anotaciones en Open Images Dataset v4 [9]	38
3.9. Categorías de objetos del dataset Open Images Dataset v4 [9]	39
3.10. Estructura de las etiquetas de Open Images Dataset v4 [23]	41
3.11. Métricas durante el entrenamiento de la red neuronal con el dataset de OIDv4	44
3.12. Evolución del mAP y pérdidas a lo largo de las interacciones durante el entrenamiento de la red neuronal con el dataset de OIDv4	45
3.13. Ejemplo de MOT con Deep SORT en [24]	47
3.14. Ejemplo de MOT con Deep SORT filtrando clases de interés en [4]	48
3.15. Esquema hipótesis detección objeto abandonado	49
3.16. Asociación persona-objeto [6]	50
3.17. Aviso de alerta posible objeto abandonado [6]	50
3.18. Detección de objeto abandonado [6]	51
 4.1. Área de superposición IoU entre los cuadros delimitadores [25]	54
4.2. Matriz de confusión [26]	54
4.3. Imágenes extraídas del dataset PETS2007 [5]. (a) Fotograma de la secuencia S08-camera4 donde un hombre deja su equipaje en el suelo. (b) Otro fotograma de la secuencia S08-camera4 donde el hombre abandona el lugar sin su equipaje.	57
4.4. Imágenes extraídas del dataset PETS2007 [5]. (a) Fotograma de la secuencia S07-thirdView donde una mujer se encuentra junto a su equipaje. (b) Otro fotograma de la secuencia S07-thirdView donde la mujer abandona el lugar sin su bolsa de mano.	57
4.5. Regiones de interés del dataset AVSSAB2007 [6]	58
4.6. Imágenes extraídas del dataset AVSSAB2007 [6]. (a) Fotograma de la secuencia AVSSAB-Easy donde el hombre abandona su maleta en la zona cercana. (b) Fotograma de la secuencia AVSSAB-Medium donde una mujer se encuentra junto a su bolsa de mano en la zona media del andén del metro.	58
4.7. ROI del hall de la Escuela Politécnica Superior (UAH) [27]	59
4.8. Imágenes extraídas de secuencias del primer escenario del dataset GBA2018 [4]. (a) Fotograma de la secuencia GBA-far-video2 donde dos bolsas de mano han sido abandonadas en medio del hall. (b) Fotograma de la secuencia GBA-far-video3 donde varias bolsas y maletas están alejadas de sus propietarios.	59
4.9. Imágenes extraídas de secuencias del segundo escenario del dataset GBA2018 [4]. (a) Fotograma de la secuencia GBA-near-big-video2 donde una bolsa de mano es abandonada en el pasillo de la cafetería. (b) Fotograma de la secuencia GBA-near-big-video4 donde dos personas abandonan una pequeña bolsa de mano.	59
4.10. Imágenes extraídas del dataset ABODA [7]. (a) Fotograma donde dos chicos conversan en el hall. (b) Otro fotograma donde los dos chicos abandonan una mochila.	60
4.11. Imágenes extraídas del dataset ABODA [7]. (a) Fotograma de la secuencia video5 de una grabación nocturna. (b) Fotograma de la secuencia video11 donde hay varias personas haciendo cola en un aeropuerto.	60
4.12. Resumen métricas primer entrenamiento de la red neuronal con el dataset de OIDv4	62
4.13. Resumen métricas segundo entrenamiento de la red neuronal con el dataset de OIDv4	64

4.14. Ejemplo 1 detección YOLOv4 Tensorflow [4]	66
4.15. Ejemplo 2 detección YOLOv4 Tensorflow [6]	66
4.16. Ejemplo 3 detección YOLOv4 Tensorflow [7]	67
4.17. Ejemplo 1 seguimiento YOLOv4 y Deep Sort [6]	67
4.18. Ejemplo 2 seguimiento YOLOv4 y Deep Sort [4]	68
4.19. Ejemplo 3 seguimiento YOLOv4 y Deep Sort [5]	68
4.20. Individuo abandona su maleta en el la zona cercana del andén [6]	69
4.21. Individuo abandona su mochila en el hall de un edificio [7]	70
4.22. Individuo abandona su bolsa de mano en el pasillo de cafetería de la EPS (1) [4]	70
4.23. Individuo abandona su bolsa de mano en el pasillo de cafetería de la EPS (2) [4]	71
4.24. Una bolsa de mano se encuentra abandonada al final de un andén de trenes [6]	71
4.25. Individuo abandona una bolsa de deporte en un aeropuerto [5]	72
C.1. Descarga del instalador de Anaconda [28]	92

Índice de tablas

2.1. Velocidad y precisión YOLOv4 y SSD con Maxwell GPU: GTX Titan X (Maxwell) o Tesla M40 GPU [1] [29]	25
2.2. Velocidad y precisión YOLOv4 y Faster R-CNN con Pascal GPU: Titan X (Pascal), Titan Xp, GTX 1080 Ti, o Tesla P100 GPU [1] [18]	26
2.3. Velocidad y precisión YOLOv4 y EfficientDet con Volta GPU: Titan Volta or Tesla V100 GPU [1] [20]	26
2.4. Batch final con normalización ℓ_2 proyectan las características en la hiperesfera unitaria [2]	31
4.1. Datasets utilizados en la evaluación de los algoritmos	56
4.2. Métricas de calidad en el primer entrenamiento con OIDv4 [1]	61
4.3. Métricas de calidad en el primer entrenamiento con OIDv4 [2]	61
4.4. Métricas de calidad en el primer entrenamiento con OIDv4 [3]	61
4.5. Métricas de calidad en el primer entrenamiento con OIDv4 [4]	62
4.6. Métricas de calidad en el segundo entrenamiento con OIDv4 [1]	63
4.7. Métricas de calidad en el segundo entrenamiento con OIDv4 [2]	63
4.8. Métricas de calidad de MS COCO en las clases de interés	65
4.9. Comparativa métricas de calidad entre los test en OIDv4 y MS COCO [1]	65
4.10. Comparativa métricas de calidad entre los dos test en OIDv4 y MS COCO [2]	65
B.1. Costes de mano de obra	88
B.2. Recursos hardware	89
B.3. Recursos software	89
B.4. Presupuesto total	89

Índice de códigos

3.1. Evaluación del detector de objetos YOLOv4 en Darknet (1)	34
3.2. Evaluación del detector de objetos YOLOv4 en Darknet (2)	34
3.3. Evaluación del detector de objetos YOLOv4 en Darknet (3)	35
3.4. Evaluación del detector de objetos YOLOv4 en Tensorflow (1)	35
3.5. Evaluación del detector de objetos YOLOv4 en Tensorflow (2)	36
3.6. Evaluación del detector de objetos YOLOv4 en Tensorflow (3)	36
3.7. Evaluación de las métricas de calidad de MS COCO en YOLOv4 Darknet (1)	39
3.8. Evaluación de las métricas de calidad de MS COCO en YOLOv4 Darknet (2)	40
3.9. Fichero coco.data	40
3.10. Evaluación de las métricas de calidad de MS COCO en YOLOv4 Darknet (3)	40
3.11. Descarga dataset Open Images Dataset v4 (1)	41
3.12. Descarga dataset Open Images Dataset v4 (2)	41
3.13. Descarga dataset Open Images Dataset v4 (3)	42
3.14. Fichero obj.data	43
3.15. Fichero obj.names	43
3.16. Generación del fichero train.txt	43
3.17. Generación del fichero test.txt	43
3.18. Entrenamiento del dataset personalizado	44
3.19. Evaluación métricas de calidad del dataset utilizado para el entrenamiento de la red neuronal de detección de objetos	44
3.20. Evaluación del seguimiento de objetos Deep SORT y YOLOv4 en Tensorflow (1)	46
3.21. Evaluación del seguimiento de objetos Deep SORT y YOLOv4 en Tensorflow (2)	46
3.22. Evaluación del seguimiento de objetos Deep SORT y YOLOv4 en Tensorflow (3)	46
3.23. Prueba test del seguimiento de objetos Deep SORT y YOLOv4	47
3.24. Clases permitidas en el seguimiento de objetos con Deep SORT	47
3.25. Colores y cuadros delimitadores de las clases en el seguimiento	48
3.26. Diccionario centroides personas y objetos de la función asociar	49
3.27. Cálculo distancia entre persona y objetos	49
3.28. Evaluación de la detección de objetos abandonados con DeepSORT y YOLOv4 en Tensorflow (1)	51
3.29. Evaluación de la detección de objetos abandonados con DeepSORT y YOLOv4 en Tensorflow (2)	51
3.30. Evaluación de la detección de objetos abandonados con DeepSORT y YOLOv4 en Tensorflow (3)	51
C.1. Instalación de Git	91
C.2. Verificación de la integridad de la instalación de Anaconda	92

C.3. Ejecutar el instalador de Anaconda para Linux	92
C.4. Hacer efectivo los cambios en el fichero .bashrc	92
C.5. Descarga repositorio	92
C.6. Comprobar capacidad computación de la GPU	93
C.7. Creación entorno virtual en Anaconda	93
C.8. Activar entorno virtual de Anaconda	93
C.9. Descarga de pesos y conversion modelo YOLOv4	94
C.10. Ejecutar script detección de objetos con YOLOv4 en Tensorflow	94
C.11. cambiar a la rama develop	94
C.12. Ejecutar script seguimiento de personas y objetos con DeepSORT	95
C.13. cambiar a la rama abandoned	95
C.14. Ejecutar script detección de objetos abandonados con YOLOv4 y Deep SORT	95

Lista de Acrónimos

ABODA	Abandoned Objects Dataset.
AP	Average precision.
ASFF	Adaptively Spatial Feature Fusion.
AUC	Area Under the ROC Curve.
AVSSAB2007	Advanced Video and Signal based Surveillance Abandoned Baggage.
BOW	Bag of Words.
CNN	Redes Neuronales Convolucionales.
CUDA	Compute Unified Device Architecture.
cuDNN	CUDA Deep Neural Network.
Deep SORT	Simple Online and Realtime Tracking with a Deep Association Metric.
DNN	Deep Neural Network.
DSSD	Deconvolutional Single Shot Detector.
EPS	Escuela Politécnica Superior.
FairMOT	On the Fairness of Detection and Re-Identification in Multiple Object Tracking.
FC	Fully Connected.
FLOPS	Floating Point Operations Per Second.
FN	False Negative.
FP	False Positive.
FPN	Feature Pyramid Network.
FPS	Frames Per Second.
GBA2018	GEINTRA Behaviour Analysis 2018.
GEINTRA	Grupo de Ingeniería Electrónica Aplicada a Espacios Inteligentes y Transporte.
GPU	Graphics Processing Unit.
HOG	Histograma de Gradientes Orientados.
IDE	Integrated Development Environment.
IoU	Intersection over Union.
IPM	Inverse Perspective Mapping.
mAP	Mean average precision.
MHT	Multiple Hypothesis Tracking.
MLP	Multilayer perceptron.
MOT	Multi Object Tracking.
MS COCO	Microsoft Common Objects in Context (test-dev 2017).

NAS	Network Architecture Search.
OIDv4	Open Images Dataset v4.
PAN	Path Aggregation Network.
PETS2007	Performance Evaluation of Tracking and Surveillance 2007.
R-CNN	Region Based Convolutional Neural Networks.
RGB	Red, Green, Blue.
ROI	Region of interest.
RPN	Region Proposal Network.
SFAM	Scale-wise Feature Aggregation Module.
SORT	Simple Online and Realtime Tracking.
SOT	Single Object Tracking.
SPP	Spatial Pyramid Pooling.
SSD	Single Shot MultiBox Detector.
SVDD	Support Vector Data Description.
SVM	Support Vector Machines.
TFM	Trabajo Fin de Máster.
TN	True Negative.
TP	True Positive.
UAH	Universidad de Alcalá de Henares.
YOLO	You Only Look Once.
YOLOv4	You Only Look Once v4.

Capítulo 1

Introducción

La mayoría de las personas gastan más tiempo y energías en hablar de los problemas que en afrontarlos.

Henry Ford

1.1. Motivación

El desarrollo de sistemas de videovigilancia automatizados ha despertado un gran interés en los últimos años en la monitorización de lugares públicos y privados. Conforme que estos sistemas crecen, la forma de observar todas las cámaras en un momento concreto se convierte en todo un desafío, especialmente en lugares públicos y concurridos como pueden ser aeropuertos, estaciones de trenes o edificios. Una característica muy deseable de estos sistemas es la detección automática de eventos de interés, característica que permite centrar la atención en ubicaciones de vigilancia potencialmente peligrosas.

En los últimos años la detección de objetos abandonados se ha investigado para detectar eventos de gran interés como objetos abandonados [30] y vehículos estacionados ilegalmente [31]. Los sistemas de detección de objetos abandonados analizan los objetos que se encuentran en movimiento en un determinado escenario con el objetivo de identificar los estáticos, los cuales se convierten en aspirantes a ser objetos abandonados. Posteriormente, una serie de pasos de filtrado validan a los candidatos para determinar si son vehículos, personas u objetos abandonados.

Los desarrollos de técnicas de detección de objetos abandonados se encuentran constantemente enfrentados contra diferentes desafíos durante su implementación. Es necesario que funcionen correctamente en escenarios complejos con condiciones cambiantes y una alta densidad de objetos en movimiento. Muchos factores visuales afectan el rendimiento de la detección de objetos abandonados, como el ruido de la imagen, cambios en la iluminación, ya sean graduales o inesperados, fluctuación de la cámara y camuflaje entre un primer plano del objeto y el fondo son algunos de los desafíos de *background subtraction*. Fondos dinámicos, que contienen objetos en movimiento, también son un tema importante a tener en cuenta. Además, los desafíos con el procesamiento de datos en tiempo real surgen por la gran cantidad de datos que deben de ser manejados por los (relativamente) complejos sistemas de detección de objetos abandonados compuestos por varias etapas. Otro desafío crítico es la operación sin supervisión durante largos períodos de tiempo donde el efecto de los factores visuales disminuyen el rendimiento y los errores suelen aparecer en las primeras etapas de los sistemas de detección de objetos abandonados, que se propagan a las etapas posteriores.

Los sistemas de detección de objetos abandonados actuales se centran principalmente en dos etapas principales: detección estacionaria y clasificación de objetos. La tarea de detección de objetos estáticos tiene como objetivo detectar en el primer plano los objetos de la escena que permanecen inmóviles después de haberse movido anteriormente. Una vez ubicados los objetos estacionarios, la tarea de clasificación identifica si el objeto estático se trata de un objeto abandonado o no. A pesar de la gran variedad de propuestas, hay una falta de comparaciones cruzadas (tanto teóricas como experimentales), lo que dificulta la evaluación. Además, estos enfoques proporcionan soluciones parciales para los sistemas de detección de objetos abandonados. El impacto de estas soluciones parciales rara vez se estudia para sistemas *end-to-end* más grandes, cuya entrada es la secuencia de vídeo y la salida es el evento del objeto abandonado. Las

validaciones experimentales generalmente se limitan a vídeos de corta duración o de baja complejidad. Por lo tanto, los parámetros del sistema pueden estar ajustados en exceso a los desafíos específicos que aparecen en los datasets pequeños, lo cual dificulta la extrapolación de conclusiones a datos no vistos.

Los objetos abandonados se pueden determinar mediante dos reglas: el objeto aspirante se encuentra estático o desatendido. El primer enfoque corresponde a una regla espacial, en la que un objeto se considera desatendido si el propietario del objeto se encuentra apartado del objeto. La cercanía al objeto se define considerando una elipse o círculo cuyo radio es proporcional al tamaño del objeto. En [32] se establece una distancia máxima de 3 metros para evaluar si el objeto es abandonado o no al pasar 30 segundos, tal como se muestra en la figura 1.1.

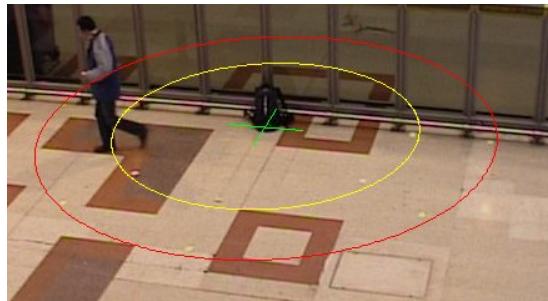


Figura 1.1: Persona cruzando del radio de 2 metros (marcado en amarillo) al radio de 3 metros (marcado en rojo) alrededor de su equipaje (marcado con una cruz verde) [12]

El segundo enfoque define una regla temporal en la que un objeto se considera estacionario si se encuentra inmóvil durante un cierto período de tiempo, dependiendo de la aplicación, siendo típicamente 30 o 60 segundos. Ambas reglas se deben de cumplir para considerar un evento de objeto abandonado. Los sistemas de detección de objetos abandonados propuestos en la literatura se pueden unificar utilizando el diagrama de la figura 1.2.

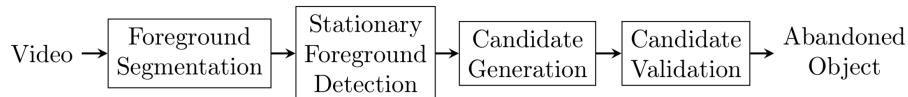


Figura 1.2: Marco de referencia en detección de objetos abandonados [11]

Este diagrama está formado por varias etapas para la segmentación del primer plano (es decir, detectar las [Region of interest \(ROI\)](#)), detección de primer plano estacionaria (es decir, determinar cuáles no se mueven durante un cierto período de tiempo), la generación de posibles candidatos (es decir, la identificación de los objetos aspirantes a ser abandonados), y validación del candidato (es decir, decidir si el objeto ha sido abandonado o no). Las primeras dos etapas pueden ser aplicables mediante la regla temporal antes mencionada y la tercera y cuarta etapa con la regla espacial. El rendimiento de cada etapa está directamente relacionada con la de la etapa anterior por lo que, las investigaciones de detección de objetos abandonados están enfocadas hacia la primera y segunda etapa.

En los últimos años se ha mostrado un gran progreso en la detección de personas debido a la aparición de métodos de aprendizaje profundo [33] [34] [35]. Los detectores de personas y objetos basados en las [Redes Neuronales Convolucionales \(CNN\)](#) pueden aprender características de píxeles sin procesar, superando modelos basados en características hechas a mano. Los enfoques de dos etapas primero calculan los métodos de propuesta de región sobre la entrada para calcular los potenciales cuadros delimitadores que se clasifican en segundo lugar. Los enfoques de [Region Based Convolutional Neural Networks \(R-CNN\)](#) [36] [18] son actualmente uno de los métodos de dos etapas de detección superiores. Por otro lado, los enfoques de una sola etapa replantean las dos etapas (propuesta de región y clasificación) en un problema de regresión de una sola etapa, lo que requiere menos tiempo. Cuatro modelos de etapa única de última generación son SqueezeDet [37], You Only Look Once (YOLO) [38], Single Shot MultiBox Detector (SSD) [29] y Deconvolutional Single Shot Detector (DSSD) [39].

Algunos sistemas de detección de objetos abandonados del Estado del Arte no incluyen la etapa de detección de personas, ya que no consideran detecciones falsas causadas por personas inmóviles. Alternativamente, otros trabajos incorporan una etapa de detección de personas para la clasificación de candidatos.

El clasificador de cuerpo completo de características similares a Haar, descrito en [40], es un clasificador basado en un modelo de persona; por tanto, es muy eficaz. Se utiliza en el sistema de detección de objetos abandonados propuesto en [41]. El modelo deformable basado en partes, propuesto en [42], es un modelo de persona basado en partes, que fue utilizado en [43] para la detección de personas. Dependiendo del propósito, los candidatos pueden restringirse a una categoría de objeto, como coches; por lo tanto, se requiere un detector/clasificador específico. El [Histograma de Gradientes Orientados \(HOG\)](#) aplica una búsqueda exhaustiva basada en descriptores de apariencia a lo largo de toda la imagen. Se utilizó para la detección de automóviles en [31], aunque inicialmente se propuso en [44] para la detección humana. Todas las tecnologías mencionadas anteriormente estaban basadas en la apariencia, pero esto también se puede combinar con seguimiento para la detección de personas, como se hizo en [45]. La figura 1.3 muestra un diagrama de bloques de esta etapa, donde se ilustra el funcionamiento.

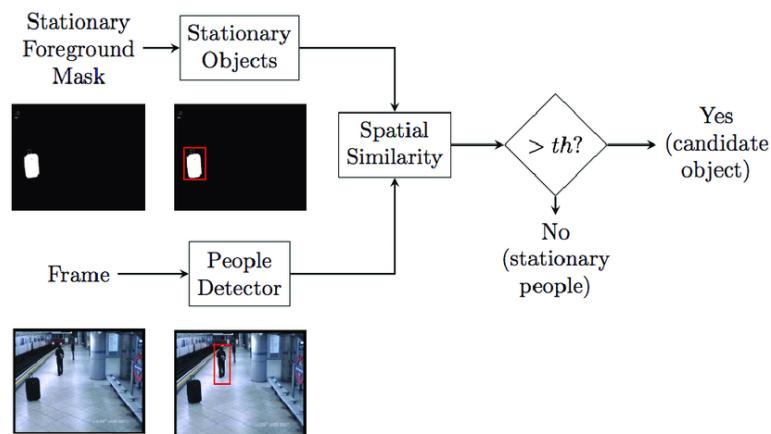


Figura 1.3: Diagrama de bloques del módulo de generación de candidatos [11]

1.2. Objetivos

El objetivo que se persigue es el desarrollo de una estrategia de detección de objetos abandonados mediante el uso de [CNN](#) en aplicaciones de videovigilancia. En concreto se va a estudiar cuando se ha abandonado los siguientes tipos de objetos: mochilas, bolsas de mano y maletas. Los espacios donde se va a evaluar la eficacia del sistema de detección desarrollado será tanto en interiores como exteriores: aeropuertos, estaciones de metro, edificios o cualquier tipo de infraestructura que disponga de una o varias cámaras de videovigilancia.

Los pasos para abordar este problema son los siguientes:

- **Revisión del Estado de Arte.** Búsqueda y estudio de estrategias en la identificación de objetos abandonados en aplicaciones de videovigilancia dentro del Estado del Arte actual para tener un punto de partida. Por otro lado, se deberá de buscar los datasets más relevantes en la evaluación de detección de objetos abandonados.
- **Evaluación de algoritmos de detección de objetos más relevantes.** Se estudiará y comparará los algoritmos de detección de objetos actuales y se argumentará el motivo de la elección de uno concreto. Una vez seleccionado el algoritmo de detección se deberá de evaluar si trabajar sobre un dataset disponible o si por el contrario resulta interesante el entrenamiento de una red neuronal personalizada en la que se detecten solamente los objetos de interés. La elección del dataset de referencia para la evaluación del algoritmo de detección se decidirá teniendo en cuenta las principales métricas de clasificación de *Machine Learning*. Teniendo un dataset de referencia seleccionado, se ejecutará el algoritmo sobre los datasets más utilizados en evaluación de algoritmos de detección de objetos abandonados.
- **Evaluación de algoritmos de seguimiento o *tracking* de objetos más relevantes.** En base al modelo del algoritmo de detección de objetos seleccionado se estudiará y evaluará los algoritmos de seguimiento actuales. El objetivo de este punto es que en la detección de objetos y personas,

cada elemento tenga una identidad propia a lo largo del tiempo, o lo que es lo mismo, a lo largo de los fotogramas de un vídeo, de tal manera que, cuando se implemente el algoritmo de detección de objetos abandonados sea más sencillo la asociación de persona-objeto. De igual manera que en el algoritmo de detección, también se ejecutará el algoritmo en los datasets más relevantes en detección de objetos abandonados para evaluar el rastreo sobre personas y objetos de interés a lo largo de una secuencia de vídeo.

- **Implementación y evaluación de un algoritmo de detección de objetos abandonados.** Se desarrollará un algoritmo capaz de determinar si un objeto ha sido abandonado o no. Existen dos posibles escenarios. El primero es que el objeto se encuentre estático durante toda la ejecución del vídeo y no se pueda asociar a ninguna persona como propietario. El segundo es que a una persona a la que se le ha asociado un objeto se encuentre en una zona de alerta a lo largo de un número determinado de fotogramas, se le aleje una máxima distancia establecida o que la persona a la que se le ha asociado un objeto desaparezca del plano de visión de la cámara. Para estos dos últimos casos se deberá de establecer una asociación persona-objeto y estudiar su comportamiento en una secuencia de vídeo determinada.

1.3. Estructura de la memoria

En este apartado se resume brevemente como se encuentra organizados los contenidos que componen el presente [Trabajo Fin de Máster \(TFM\)](#).

- **Capítulo 1: Introducción.** Se expondrá la motivación que ha impulsado la realización de este [TFM](#). Se citarán brevemente estudios previos que han servido de esqueleto del proyecto. Por otro lado, se argumentarán los objetivos que se pretenden alcanzar.
- **Capítulo 2: Estudio teórico.** Se realizará un estudio exhaustivo del Estado del Arte en lo referente a los métodos de detección de objetos abandonados que se han empleado en los últimos años y se describirán los algoritmos de detección y seguimiento que se utilizarán en el desarrollo del proyecto.
- **Capítulo 3: Desarrollo.** Se desarrollará la implementación de los algoritmos que se van a utilizar para la detección y rastreo de personas y objetos así como el algoritmo de detección de objetos abandonados.
- **Capítulo 4: Resultados.** Se expondrán los resultados obtenidos en base a métricas de calidad y los datasets utilizados para la evaluación de los algoritmos.
- **Capítulo 5: Conclusiones y líneas futuras.** Se detallarán las conclusiones que se han llegado al finalizar este proyecto. Se explicarán las ventajas y limitaciones que presenta la idea propuesta para su desarrollo. Por otro lado, se razonarán posibles vías de desarrollo derivadas de este proyecto, así como futuros proyectos donde se puedan emplear las bases de los algoritmos de detección y seguimiento y únicamente se tenga que programar un algoritmo que realice una función concreta.
- **Bibliografía.** Se incluirán cada uno de los artículos, repositorios, datasets y todo el material consultado para la elaboración de este [TFM](#).
- **Apéndice A.** Se hará referencia al pliego de condiciones donde se tendrán en cuenta las especificaciones *hardware* y *software* que se han empleado en el desarrollo de este proyecto.
- **Apéndice B.** Se mostrará el presupuesto donde se incluye los costes materiales *hardware* y *software* y el coste de la mano de obra en función a la duración estimada del proyecto.
- **Apéndice C.** Este apéndice se dividirá en dos partes. Primero, en la guía de instalación, se detallarán cada uno de los pasos de la instalación de las dependencias necesarias para el correcto funcionamiento de los algoritmos. Posteriormente, se podrá consultar la guía de ejecución, donde se indicará como poner en funcionamiento cada uno de los algoritmos desarrollados en este proyecto.

Capítulo 2

Estudio teórico

Si buscas resultados distintos no hagas siempre lo mismo.

Albert Einstein

2.1. Introducción

En este capítulo se ha realizado un estudio del marco teórico que engloba este trabajo donde se revisará los últimos estudios relacionados con los sistemas de detección de objetos abandonados.

En primer lugar, se ha realizado un repaso de las diferentes técnicas que se han utilizado hasta día de hoy en la detección de objetos abandonados en imágenes. La segmentación de objetos en movimiento situados en primer plano, la detección de objetos estacionarios, el reconocimiento de comportamientos o la detección de personas y objetos mediante el uso de [CNN](#) son los métodos de detección más relevantes en los últimos años.

En segundo lugar, se hará una breve introducción a las [CNN](#), un tipo de red neuronal artificial de *Deep Learning*, que utiliza imágenes en la entrada de la red para encontrar patrones en las imágenes con el objetivo de reconocer formas dentro de los objetos. También resulta interesante su uso en la clasificación de datos de audio o señales. En los últimos años se están utilizando para el reconocimiento facial o vehículos autónomos.

Por último, y más enfocado a los contenidos de este trabajo, se extenderá la sección [2.2.3](#) realizando un breve recorrido por los principales algoritmos de detección y seguimiento de personas y objetos basados en [CNN](#).

2.2. Estado del Arte

En esta sección se va a enumerar las distintas técnicas que han sido utilizadas en la identificación de objetos abandonados citando los trabajos más relevantes de otros investigadores.

2.2.1. Segmentación de objetos en primer plano

El análisis de vídeo se trata de uno de los campos de investigación más amplios en la actualidad. Muchas de las aplicaciones han necesitado tener un primer paso en la detección de movimiento de objetos en un escenario como en [\[46\]](#), donde se ha realizó una sustracción del fondo para la videovigilancia de tráfico urbano o en espacios de aprendizaje multimedia [\[47\]](#). Una etapa básica en estos sistemas se trata de la separación de los objetos que se encuentran en un primer plano con el fondo.

Típicamente, la forma de modelar el fondo es obtener una imagen que se encuentre en el fondo sin ningún objeto en movimiento [\[13\]](#). En ocasiones el modelo de representación debe de ser robusto, ya que nos podemos encontrar con fondos de escenarios que sufran cambios debidos a alteraciones en la iluminación u objetos que han sido introducidos y/o retirados.

Por tanto, los dos principales problemas con los que nos encontramos en la sustracción del fondo son la detección de cambios y la detección de movimientos. Cuando hablamos de detección de cambios hablamos de los cambios producidos entre dos imágenes. Cuando se realiza una sustracción del fondo podemos encontrarnos con dos casos, que una imagen corresponda al fondo y la otra imagen corresponda a la imagen actual, o bien que los cambios se han producido por el movimiento de las personas u objetos.

En la figura 2.1 se muestra las etapas de la sustracción del fondo donde inicialmente se emplean N fotogramas para obtener la imagen del fondo sin que haya ningún objeto en movimiento. La siguiente etapa corresponde a la detección de objetos en movimiento en el primer plano donde se clasifica los píxeles que se encuentran en el primer plano comparando el fondo con la fotograma actual. Hay una etapa de mantenimiento para actualizar la imagen del fondo en todo momento. Estas dos últimas etapas nombradas se realizan en bucle a lo largo del tiempo.

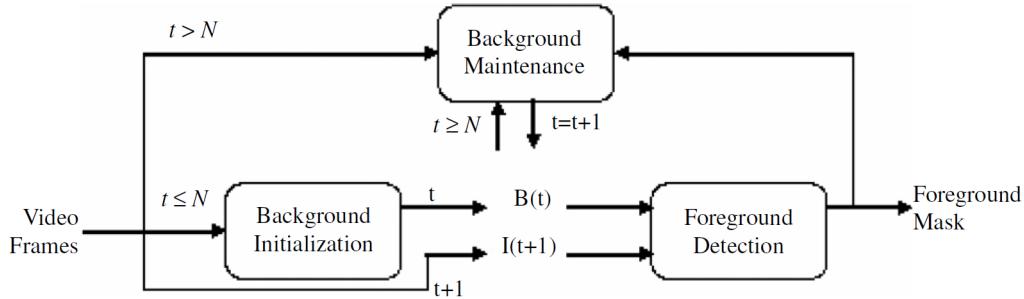


Figura 2.1: Proceso de la sustracción del fondo [13]

A continuación, se exponen con más detalle cada una de las etapas que compone la sustracción del fondo:

Inicialización del fondo

Consiste en el modelado del fondo donde se describe el tipo de modelo que se está utilizando para representar. Principalmente se determina la capacidad del modelo para tratar con fondos estáticos (unimodal) o fondos dinámicos (multimodal).

En esta etapa se inicializa el modelo donde generalmente se utiliza el primer fotograma sobre un conjunto de fotogramas de entrenamiento, los cuales contienen o no objetos en primer plano. El principal desafío es obtener un primer modelo de fondo cuando más de la mitad de los fotogramas de entrenamiento contiene objetos en primer plano.

Mantenimiento de fondo

El mantenimiento del fondo se encarga de adaptar el modelo a los cambios que puedan ser ocasionados a lo largo del tiempo. En esta etapa de aprendizaje se debe de realizar en línea por lo que el algoritmo debe de ser incremental. Los puntos claves en este proceso son los siguientes:

- **Esquemas de mantenimiento.** En trabajos previos como en [48] se presentan tres esquemas de mantenimientos: ciegos, selectivos y adaptativos difusos. El mantenimiento ciego del fondo actualiza todos los píxeles con las mismas reglas que se emplean en un filtro IIR:

$$B_{t+1}(x, y) = (1 - \alpha)B_t(x, y) + \alpha I_t(x, y) \quad (2.1)$$

donde:

- α es el ratio de aprendizaje y tiene un valor comprendido entre $[0,1]$.
- B_t y I_t son el fondo y la imagen actual respectivamente en el tiempo t .

La principal desventaja de este esquema es que el valor de los píxeles clasificados como primer plano son utilizados en el cálculo del nuevo fondo y por tanto, afecta a la imagen de fondo. Para lidiar con este problema algunos investigadores utilizan un esquema de mantenimiento selectivo que se

basa en actualizar la nueva imagen de fondo con diferentes ratios de aprendizaje en función de la clasificación previa del píxel en primer plano o fondo:

$$B_{t+1}(x, y) = (1 - \alpha)B_t(x, y) + \alpha I_t(x, y)$$

si (x, y) es el fondo

$$B_{t+1}(x, y) = (1 - \beta)B_t(x, y) + \beta I_t(x, y)$$

si (x, y) es el primer plano

La idea es adaptar el píxel clasificado como fondo de manera rápida y un píxel clasificado como primer plano muy despacio. Por esta razón $\beta \ll \alpha$ y generalmente $\beta = 0$. Por tanto, la ecuación 2.1 se convierte en:

$$B_{t+1}(x, y) = B_t(x, y) \quad (2.2)$$

El problema es que una clasificación errónea puede resultar un error permanente en el modelo del fondo. Este problema se puede solucionar mediante un esquema adaptativo difuso que toma en cuenta la incertidumbre de la clasificación. Esto puede lograrse graduando la regla de actualización utilizando el resultado de la detección del primer plano.

- **Ratio de aprendizaje.** Determina la velocidad de adaptación en los cambios de escena. Este ratio puede ser fijo, ajustado dinámicamente o difuso.
- **Mecanismos de mantenimiento.** El ratio de aprendizaje determina la velocidad de adaptación a los cambios de iluminación pero también el tiempo que necesita un cambio en el fondo hasta que se incorpora en el modelo, así como el tiempo donde un objeto que se encuentra en el primer plano estático puede sobrevivir antes de ser incluido en el modelo. El ratio de aprendizaje por tanto se encarga de diferentes desafíos diferenciados. Para desacoplar el mecanismo de adaptación y el de incorporación, [49] se utilizó un conjunto de contadores que representan el número de veces que un píxel se clasifica como píxel de primer plano. Cuando este número es mayor que cierto umbral, el píxel se considera que se encuentra en el fondo. Esto da un límite de tiempo de cuanto tiempo un píxel se encuentra como píxel del primer plano estático.
- **Frecuencia de actualización.** El objetivo es actualizar el fondo solo cuando sea necesario. El mantenimiento se puede realizar en cada fotograma, sin embargo si no se producen cambios no es necesaria la actualización de los píxeles en cada fotograma.

Detección del primer plano

Esta etapa se trata de una tarea de clasificación que se encarga de etiquetar píxeles como fondo o como píxeles de primer plano.

Aplicaciones donde se utiliza la sustracción del fondo

La segmentación de objetos en movimiento sobre el primer plano se utiliza en multitud de aplicaciones donde se emplea visión por computadora como pueden ser:

- **Videovigilancia inteligente.** Se trata de una de las aplicaciones donde más se utiliza esta metodología. El objetivo es detectar objetos en movimiento u objetos abandonados para garantizar la seguridad aérea, para calcular estadísticas de tráfico como se puede ver en la figura 2.2 o para vigilancia marítima. Los objetos de interés suelen ser variados como vehículos, aviones, barcos, personas o equipajes.

- **Codificación de vídeo basada en el contenido.** Para generar un vídeo, debe de estar segmentado en objetos de vídeo y seguidos a medida que transcurren los fotogramas del vídeo. El fondo y los objetos presentes en el vídeo son codificados por separado. En definitiva, la codificación en vídeo necesita métodos efectivos para la detección de objetos en entornos estáticos y dinámicos.
- **Captura de movimiento óptico.** El objetivo es obtener una captura completa y precisa de las personas mediante el uso de cámaras. La silueta se extrae generalmente en cada vista de la sustracción del fondo.

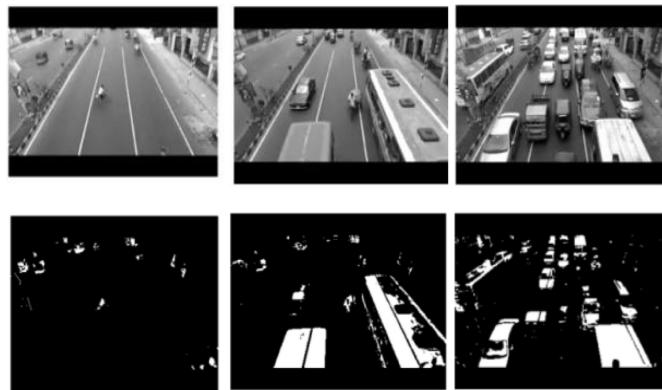


Figura 2.2: Ejemplo de sustracción del fondo en aplicaciones de tráfico [13]

2.2.2. Detección de objetos estacionarios

La detección de objetos estacionarios está recibiendo una atención especial ya que se trata de una fase de análisis crítico en aplicaciones como la detección de objetos abandonados o vehículos estacionados en áreas públicas. El reconocimiento de objetos estacionarios en escenarios de grandes aglomeraciones de personas supone una tarea desafiante.

Se producen problemas ocasionados por las occlusiones o variación de colores y formas conforme las personas se mueven. Otros problemas que surgen son la iluminación, la velocidad de los objetos y la densidad de los objetos se deben de tener en cuenta. En la detección de objetos en primer plano, los métodos basados en la sustracción de fondo se han vuelto muy populares debido a que se suelen emplear cámaras fijas y los cambios de iluminación son muy graduales [50]. En trabajos previos como en [14] se propone enfoque en el análisis de imágenes estáticas.

En la figura 2.3 se muestra clasificación de sustracción del fondo en base al método utilizado.

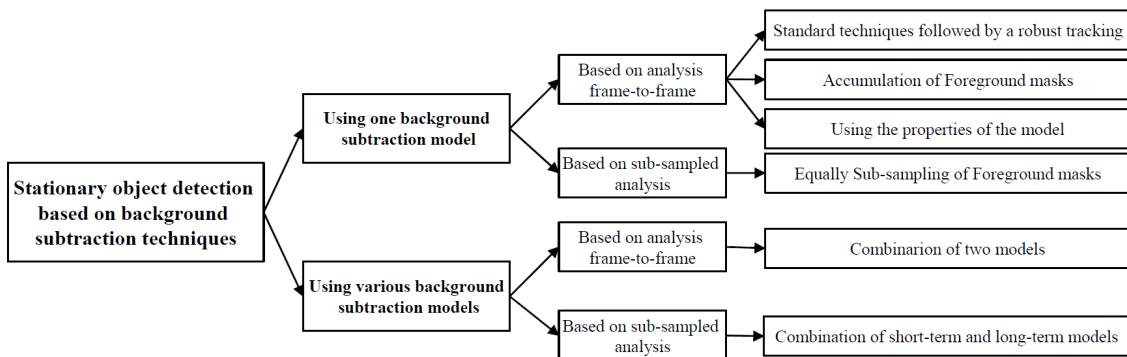


Figura 2.3: Clasificación de sustracción del fondo basados en métodos de detección de objetos estacionarios [14]

Dependiendo del uso de los mapas en primer plano calculados en el análisis de sustracción de fondo, los enfoques basados en un modelo se pueden clasificar en:

- **Basado en análisis fotograma a fotograma.** Se emplean técnicas de sustracción del fondo seguidas de otro tipo análisis. En función de este tipo de análisis se puede clasificar en: basados en el uso de técnicas estándar de fondo seguido de otra etapa de análisis, basados en la acumulación de máscaras en primer plano calculadas fotograma a fotograma, o basado en las propiedades del modelo de sustracción de fondo utilizado.
- **Basado en análisis de submuestreo.** Estas propuestas tratan de detectar objetos estacionarios analizando las secuencias de vídeos a diferentes velocidades de fotogramas.

Los enfoques que combinan uno o más modelos de sustracción de fondo han sido menos estudiados. No obstante, una clasificación basada en el procesamiento de la velocidad de los fotogramas se puede realizar de la siguiente manera:

- **Basado en análisis fotograma a fotograma.** En esta categoría tenemos métodos que combinan diferentes propiedades usando dos o varias técnicas de sustracción de fondo.
- **Basado en un análisis de submuestreo.** Estos enfoques detectan objetos estacionarios analizando las secuencias de vídeo con varios métodos de sustracción de fondo a diferentes velocidades de fotogramas.

2.2.3. Detección de personas y objetos

Existe una gran cantidad de estudios de detección de personas en la literatura, algunos de ellos cubren parcialmente solo el Estado del Arte o están claramente enfocados en alguna aplicación de videovigilancia en particular. En [51] se presenta un estudio de detección de personas y también la integración de los detectores en sistemas completos a bordo. Descompone los enfoques de detección de personas en tres tareas de procesamiento: generación de hipótesis de objeto inicial o selección de la **ROI**, verificación (clasificación) e integración temporal (seguimiento). [52] presenta una descripción general de los algoritmos de detección de personas centrados solo en enfoques de búsqueda exhaustivos, mientras que en [53] presentan una descripción general centrada únicamente en enfoques de ventana deslizante.

Se pueden clasificar los algoritmos de detección de personas en: las técnicas utilizadas, el tipo de modelos utilizados, el uso de información 2D o 3D, la modalidad del sensor, la multiplicidad del sensor, la ubicación del sensor o la movilidad del sensor. Los algoritmos de detección de personas se clasifican según el enfoque utilizado para generar o extraer los objetos iniciales y en base al modelo de persona.

Hipótesis de objeto inicial

Hay dos enfoques principales de detección de objetos convencionales (ver figura 2.4): los que se basan en algún tipo de segmentación de la escena en primer plano (objetos) y el fondo [54] y los que se basan en un enfoque de búsqueda exhaustiva [55]. También hay algunos enfoques que intentan combinar ambas técnicas juntas [56]. El resultado de esta etapa es la ubicación y dimensión (cuadro delimitador) de los diferentes objetos de la escena candidatos a ser una persona.

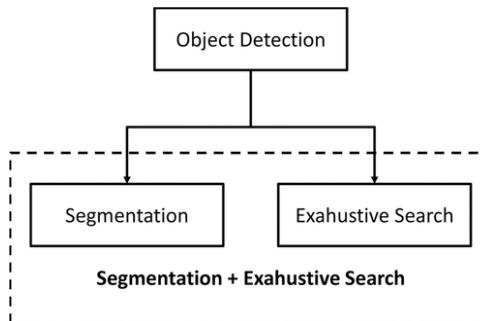


Figura 2.4: Clasificación de detección de personas enfocado a la detección de objetos [15]

Segmentación

El uso de la segmentación genera directamente los objetos candidatos a ser persona y fácilmente se rechaza las áreas irrelevantes de la imagen, es decir, sin objetos de interés. Por este motivo, la tarea de clasificación posterior se simplifica claramente y, por tanto, el modelo de persona suele ser más sencillo y de menor coste computacional. Sin embargo, como existe una fuerte dependencia de la segmentación, todos los problemas de segmentación se heredan (segmentaciones por debajo y por encima). Estos problemas pueden afectar el rendimiento de la detección global, principalmente limitando la tasa máxima de detección (objetos no detectados) pero también aumentando el número de detecciones falsas (detecciones de objetos parciales u objetos superpuestos). Además, estos problemas se magnifican en escenarios complejos donde es bastante difícil obtener una segmentación confiable.

Búsqueda exhaustiva

La otra técnica para obtener la hipótesis de ubicación inicial del objeto es la búsqueda exhaustiva. Por lo general, consiste en escanear la imagen completa buscando similitudes con el modelo de persona elegido en múltiples escalas y ubicaciones. A través de este mecanismo se obtiene un mapa de confianza de detección denso (escala y ubicación). Para llegar a detecciones individuales, estos enfoques deben buscar máximos locales en el volumen de densidad y luego aplicar alguna forma de supresión no máxima.

Hay muchas propuestas de detección de personas en el Estado del Arte que utilizan esta técnica, de hecho, esta técnica es actualmente la más utilizada. Dentro de esta técnica, se pueden utilizar dos enfoques diferentes como se expone en [57]. Existen algunas propuestas que obtienen este volumen de densidad implícitamente muestreando en una cuadrícula 3D discreta (ubicación y escala) evaluando diferentes ventanas de detección con un clasificador. Este es el caso del uso de detectores basados en ventanas deslizantes como en [58].

Modelos de personas

El proceso de verificación o clasificación aplica un modelo de persona previamente definido o entrenado a los objetos candidatos a ser persona a partir de una imagen o secuencia y toma una decisión final en función de su similitud. Por lo tanto, la definición de un modelo de persona adecuado es una tarea crítica para el proceso de verificación o clasificación. Hay dos fuentes principales de información discriminativa para caracterizar el modelo de personas: apariencia y movimiento (ver figura 2.5). El modelo debería poder discriminar entre personas y cualquier otro objeto en la escena.

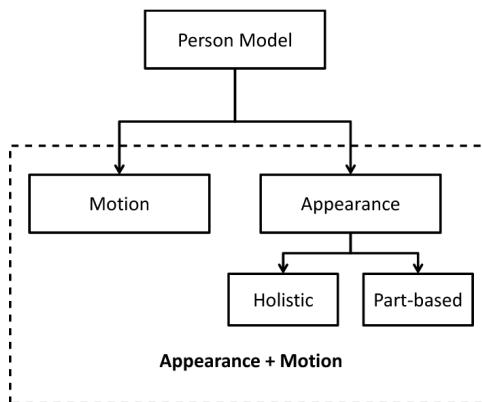


Figura 2.5: Clasificación de detección de personas según modelo de persona [15]

Basado en el movimiento

La apariencia humana varía debido a factores ambientales como las condiciones de luz, vestimenta o contrastes. Además de la enorme variabilidad intrínseca de las personas como diferentes alturas, anchos o poses. Por estas razones, existen algunos enfoques que intentan evitarlos utilizando sólo información

de movimiento. Dentro de esta clasificación, [54] proponen un sistema de clasificación de objetos basado en análisis de movimiento periódico. El algoritmo segmenta el movimiento, rastrea los objetos en primer plano, alinea cada objeto a lo largo del tiempo y finalmente calcula la auto-similitud entre los objetos y cómo evoluciona en el tiempo. [59] propone un sistema de detección de personas basado en la detección de patrones de movimiento de personas. Para cada objeto presente en dos imágenes consecutivas, se realiza la normalización de tamaño y se calcula su patrón de flujo que consiste en flujos ópticos horizontales y verticales.

Basado en apariencia

Hay muchos enfoques que utilizan información de apariencia para definir el modelo de persona. Esto se debe a que la apariencia discrimina más que el movimiento. Los modelos de apariencia se clasifican según modelos humanos simplificados o modelos complejos. Existen modelos de persona simples que definen a la persona como una región o forma, es decir, modelos holísticos [59] y modelos más complejos que definen a la persona como una combinación de múltiples regiones o formas, es decir, modelos basados en piezas [42].

2.2.4. Reconocimiento del comportamiento

La detección de un comportamiento anormal en la videovigilancia es esencial para garantizar la seguridad tanto en lugares interiores como exteriores, como estaciones de trenes o aeropuertos. La detección de conductas anormales es un problema particular del reconocimiento de la acción humana. Con el creciente número de cámaras de vigilancia, la tarea de supervisar múltiples monitores por parte del personal de seguridad se vuelve muy difícil debido a la falta de atención y a la fatiga humana. Además, los eventos anormales son relativamente raros y no ocurren con frecuencia. Esto hace que la tarea de supervisión sea más compleja. Por tanto, existe una demanda creciente de un sistema de videovigilancia inteligente que detecte de manera automática los comportamientos anormales y avise mediante una alarma.

En esta sección se va a revisar los métodos existentes [16] que se utilizan en aplicaciones de videovigilancia destacando los avances actuales en el campo de la detección de comportamientos anormales. El objetivo de un sistema de videovigilancia inteligente es detectar de manera eficiente un evento interesante a partir de una gran cantidad de vídeos para prevenir situaciones peligrosas. Esta tarea requiere dos niveles de procesamiento de vídeo tal y como se muestra en la figura 2.6.

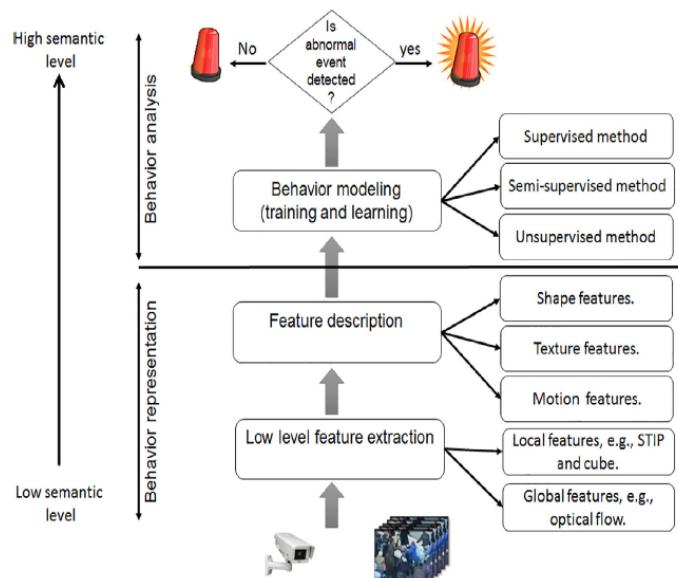


Figura 2.6: Sistema de videovigilancia inteligente [16]

El primero consta de dos pasos. Primero, se extraen características de bajo nivel, con el objetivo de detectar la ROI en la escena. Luego, se generan primitivas basadas en características de bajo nivel para

describir la [ROI](#). El segundo nivel proporciona información semántica sobre la acción humana y determina si el comportamiento es normal o no.

La detección de comportamientos anormales en la videovigilancia es una tarea desafiante en la visión por computadora y últimamente ha experimentado importantes avances. Las etapas de procesamiento de bajo nivel permiten detectar y describir el objeto en movimiento en la escena. Sin embargo, esos pasos no permiten comprender el tipo de acción que realiza el objeto en movimiento ni determinar si su comportamiento es normal o no. Dado que existen múltiples trabajos propuestos que se relacionan con el reconocimiento de conductas anormales en la videovigilancia, en esta sección se va a hacer una revisión de:

- Modelado de marcos y métodos de clasificación.
- Densidad de escenas e interacción de objetos en movimiento.

Modelado de marcos y métodos de clasificación

El reconocimiento de un comportamiento anormal depende del marco de referencia propuesto y del método utilizado para clasificar los comportamientos. Dado el tipo de muestras requeridas para el proceso de aprendizaje (normal o anormal), los métodos de clasificación se pueden categorizar en métodos supervisados, semi-supervisados y no supervisados.

Los métodos supervisados tienen como objetivo modelar comportamientos normales y anormales a través de datos etiquetados. Por lo general, están diseñados para detectar comportamientos anormales específicos predefinidos en la fase de entrenamiento, como la detección de enfrentamiento entre personas, la detección de merodeos y detección de caídas. En la literatura se proponen varios métodos supervisados con el objetivo de detectar un evento interesante en un video. Uno de los más populares es el enfoque [Bag of Words \(BOW\)](#) [60]. Consiste en representar cada video o fotograma mediante un histograma de palabras. Primero, se construye un diccionario de palabras. Luego, el histograma se calcula contando la frecuencia de cada palabra dentro del diccionario en el video. El enfoque [BOW](#) se usa generalmente con el clasificador de [Support Vector Machines \(SVM\)](#), que es una herramienta eficiente para la detección de comportamiento agresivo y el reconocimiento de anomalías de multitudes.

Los métodos semi-supervisados solo necesitan datos de video normales para el entrenamiento y se pueden dividir en categorías basadas en reglas y basadas en modelos. La primera categoría tiene como objetivo desarrollar una regla utilizando patrones normales. Cualquier muestra que no se ajuste a esta regla se considera un valor atípico (anomalía). En [61] se propuso un método basado en reglas que utiliza codificación escasa para detectar comportamientos anormales. Aunque se logró un buen resultado en un tiempo de ejecución corto ([150 Frames Per Second \(FPS\)](#)), su resultado se ve muy afectado por el valor de umbral. Otros trabajos se basan en la construcción de algunas reglas para clasificar el comportamiento en normal y anormal. En [62] se propone un sistema de detección de caídas basado en reglas extraídas utilizando características de forma.

Los métodos no supervisados tienen como objetivo aprender comportamientos normales y anormales utilizando propiedades estadísticas extraídas de datos no etiquetados. En [63] propusieron un método de comportamiento anormal utilizando un marco de aprendizaje no supervisado basado en Dominant Set. En [64] presentaron un marco de kernel no supervisado para la detección de anomalías basado en el espacio de características y la [Support Vector Data Description \(SVDD\)](#).

Densidad de escenas e interacción de objetos en movimiento

La densidad de la escena corresponde al número de personas presentes en ella. La elección de las técnicas a utilizar para caracterizar el comportamiento está directamente influenciada por la densidad de la escena. Por lo tanto, el objeto en movimiento en la escena puede ser un pequeño número de personas o un grupo de personas. Se distinguen dos tipos de escenas. El primer tipo, llamado escena con poca gente, se caracteriza por la presencia de una o unas pocas personas al mismo tiempo dentro del campo de visión de la cámara. El segundo tipo se llama escena llena de gente, ya que contiene muchas personas.

Escena con poca gente

En este tipo de escenas, es interesante detectar un comportamiento anormal realizado por una o varias personas presentes dentro del campo de la cámara. Cuando solo hay una persona en la escena, generalmente se consideran tres comportamientos anormales principales que son detección de caída, merodeo y estar en un lugar equivocado (ver figura 2.7).



Figura 2.7: Comportamientos anormales realizados por una sola persona [16]

La detección de caídas humanas es una tarea interesante y varios trabajos propusieron sistemas que se utilizan para garantizar la seguridad y protección, especialmente para las personas mayores y para las personas que viven solas. En [65], se propone un método de rastreo de partes del cuerpo humano para detectar caídas de personas mayores. Utilizaron solo una cámara de profundidad que hace que la aproximación funcione incluso en la oscuridad. En [66], se propone un algoritmo que es capaz de reconocer el comportamiento anormal de las personas mayores solitarias a partir de la información obtenida de un espacio inteligente. En [67] propusieron un nuevo sistema para monitorear a las personas solas (personas mayores, pacientes que viven solos, etc.) mediante la explotación de la información de imagen y audio en vídeo para detectar eventos anormales.

Otro tema interesante en la escena con poca gente merodeando. Se entiende merodear como el acto de estar durante un largo período en un espacio público en particular sin ningún objetivo, como que una persona tenga una maleta en un aeropuerto y se quede mucho tiempo sin ningún propósito. Este acto es anormal y varios trabajos propusieron diferentes técnicas para detectar la ocurrencia de este evento. En [68] se propuso un sistema de detección de merodeo de dos etapas basado en micropatrones secuenciales. Esos micropatrones son acciones repetidas realizadas por un individuo que caracteriza el comportamiento de merodeo y se obtienen con el algoritmo de patrones secuenciales generalizados. En [69] proponen un método para detectar merodeo en videovigilancia utilizando el historial de direcciones de la trayectoria del objeto en movimiento y el método de **Inverse Perspective Mapping (IPM)**.

Escena llena de gente

En este tipo de escena, no es posible rastrear y analizar el comportamiento de cada persona de forma individual. Esto se debe a la oclusión y al pequeño número de píxeles que representan a cada persona en el fotograma. Por ello, es mejor modelar la interacción entre personas para detectar un comportamiento anormal de la multitud. Varios trabajos previos propusieron métodos de detección de comportamientos anormales en escenarios llenos de personas basados en la interacción entre personas. La figura 2.8 muestra algunos comportamientos anormales de la multitud.



Figura 2.8: Comportamientos anómalos en multitudes

La interacción grupal incluye comportamientos realizados por varias personas, como el causado por el pánico grupal y la violencia en el estadio de fútbol. Muchos trabajos se enfocaron en detectar eventos inusuales que ocurren en escenas concurridas. En [70] se propuso un método para detectar el comportamiento anormal de la multitud utilizando un modelo de fuerza social que estima la fuerza de interacción entre individuos. Primero, calcularon el modelo de fuerza social y luego usaron el enfoque **BOW** para clasificar los eventos como normales y anormales. En [71] propusieron un método para la detección de anomalías utilizando múltiples modelos de comportamiento social que se determinan en función del flujo óptico y la advección de partículas. En [72] utilizaron agentes estáticos y dinámicos para caracterizar la interacción grupal. Los agentes estáticos tienen como objetivo observar los comportamientos individuales calculando la variación del flujo óptico. Los agentes dinámicos calculan la interacción grupal utilizando el modelo de fuerza social. En [73] se basan en la detección y el análisis de movimiento para describir la anomalía. En [74] proponen un método de detección de grupos sociales en una escena abarrotada basado en dos características que son la dirección de la mirada y la atención visual. Esas dos características se utilizan para especificar la intención de la persona en el video.

En [75] detectaron la violencia de masas en tiempo real basándose en un nuevo detector **Violent Flows**. En [76] se introdujeron un marco no supervisado basado en el modelo de red social para capturar la interacción de la multitud y la dinámica de la escena. El comportamiento de la multitud se detectó en [77] utilizando la estimación de la posición del flujo adyacente.

2.3. Redes neuronales convolucionales (CNN)

Las redes neuronales convolucionales (**CNN**) son un subconjunto de algoritmos de *Deep Learning* de aprendizaje tanto supervisado, como por ejemplo en la clasificación de imágenes, y no supervisado como puede ser la incrustación de palabras. Las **CNN** son un tipo de red neuronal artificial que se utiliza en el análisis de datos con estructura similar a una cuadrícula. Un ejemplo buen ejemplo son los datos de imágenes que pueden ser representados en dos dimensiones con valores **Red, Green, Blue (RGB)**. En problemas como pueden ser la clasificación imágenes, hay tres principales desafíos donde se usan **Multilayer perceptron (MLP)** los cuales las **CNN** pueden resolver:

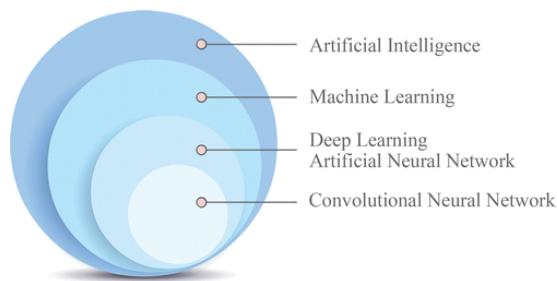


Figura 2.9: Las CNN son un subconjunto de redes neuronales de Deep Learning [17]

- **Crecimiento de parámetros.** El uso de un perceptrón por cada píxel hace que la cantidad de parámetros aumente rápidamente.
- **Traducciones.** Un **MLP** estándar trataría una imagen y su versión ligeramente desplazada como dos imágenes completamente diferentes. Por ejemplo, reconocer un automóvil en una imagen no debería de depender de en qué lugar de la imagen se encuentra.
- **Espacialidad.** Los **MLP** no tienen en cuenta las relaciones espaciales en las imágenes. El hecho de que dos píxeles se encuentren cerca es información significativa.

Las **CNN** resuelven el problema de la comprensión de las imágenes, utilizando redes de complejidad más manejable. La red neuronal especial tiene en cuenta que la cercanía entre píxeles tiene significado y que los elementos de interés pueden aparecer en cualquier parte de una imagen. Esto se logra mediante el uso de una operación de convolución lineal. El uso de esta operación en una o más capas es lo que define a una **CNN**. En ocasiones las suposiciones subyacentes a las opciones de diseño de las **CNN** deben disminuirse o alterarse debido a la naturaleza de los datos de entrada. Aunque la complejidad

computacional es más manejable, las redes tienden a ser más profundas. Esto crea algoritmos inteligentes para calcular las convoluciones.

La idea principal detrás de la convolución es la identificación de características en los datos de entrada mediante la aplicación de un kernel (también conocido como filtro) en los datos de entrada. Tanto los datos de entrada como el kernel tienen una estructura similar a una cuadrícula y se pueden representar como tensores, que son matrices multidimensionales. El kernel puede ser de cualquier tamaño y, por lo general, es más pequeño que los datos de entrada. Los núcleos se utilizan para identificar características en los datos de entrada, como los bordes de una imagen. Los datos de entrada se convolucionan con el kernel, lo que significa que el kernel se “desliza” a través de los datos de entrada, calculando el producto escalar o el producto matricial (según las dimensiones) entre la parte superpuesta de los datos de entrada y el kernel. En la figura 2.10 se puede ver un ejemplo ilustrativo de la operación de convolución.

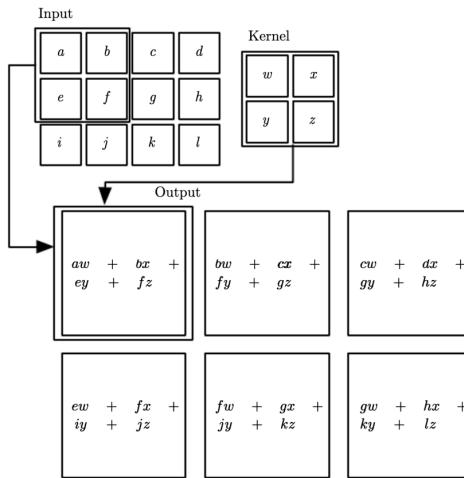


Figura 2.10: Ejemplo de convolución de datos de entrada bidimensionales [17]

La operación de convolución se define como:

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da \quad (2.3)$$

donde x es la función que se asigna a un valor específico en los datos de entrada y w representa el núcleo. Esta formulación se puede considerar como un promedio de suavizado de x en todo su dominio, dando mayor peso a los valores más cercanos a t . Si los valores de entrada son discretos, la operación de convolución se puede reescribir mediante el siguiente sumatorio:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \quad (2.4)$$

La entrada suele ser multidimensional. En ese caso, se pueden reemplazar las funciones con funciones multivariadas, es decir, operando en tensores. Suponiendo un ejemplo de aplicación de convolución a una imagen bidimensional I como entrada. Luego, se puede usar un kernel K bidimensional, y la operación se puede escribir de la siguiente manera:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n) \quad (2.5)$$

Es decir, dado un píxel en la entrada, ubicado en la fila i y la columna j , la convolución se calcula colocando el centro del núcleo sobre el píxel de entrada y sumando el producto de los parámetros del núcleo superpuestos y los píxeles de entrada para producir el valor de salida para i y j .

Ejemplo de aplicación: Detección de tumores en mamografías

Una aplicación interesante de las redes neuronales convolucionales se encuentra dentro del campo de la radiología. Cada año, millones de mujeres se someten a un tratamiento para la detección de cánceres en una etapa temprana mediante mamografías. El artículo [78] muestra que las CNN ya pueden detectar cánceres en una etapa temprana con alta precisión. De hecho, su algoritmo ya está superando en muchos aspectos a los medicina estadounidense. El uso de CNN en el diagnóstico puede reducir en gran medida el coste para los hospitales, haciendo que las pruebas estén disponibles para un público más amplio y, al mismo tiempo, aumentando la precisión del diagnóstico.

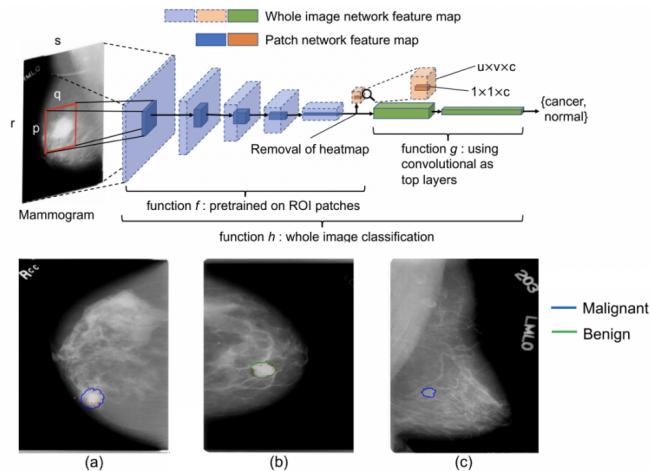


Figura 2.11: Ejemplo de convolución de datos de entrada bidimensionales [17]

La CNN resultante se ejecutó en varias bases de datos con un [Area Under the ROC Curve \(AUC\)](#) en las imágenes de menor calidad de 0,91. Entrenando una red con imágenes de alta resolución de la base de datos INbreast [79], su mejor modelo logró un [AUC](#) por imagen de 0,95. Combinando todos sus modelos, este número se elevó hasta 0,98 con una sensibilidad del 86,7 % y una especificidad del 96,1 %. Esto muestra que las CNN pueden realizar tareas que ahorran costes pero, lo que es más importante, salvan vidas. En base al resultado del artículo [78], esto significaría que la CNN está superando al personal médico en la detección y clasificación de tumores. Los médicos tienen solo un 0,3 % más de probabilidades de detectar correctamente un tumor maligno, pero tienen un 7,2 % menos de probabilidades de encontrar correctamente que un paciente esté sano.

2.4. Algoritmos de detección de objetos

En esta sección se va exponer los distintos algoritmos de detección de objetos que están siendo utilizados en los últimos años. Por un lado, están los detectores basados en regiones como Faster R-CNN, donde a partir de una imagen de entrada, se proponen múltiples regiones dentro de la imagen a través de un algoritmo de búsqueda selectiva, de donde se obtienen múltiples regiones en base a características de la imagen que proporcionan potenciales zonas que pueden contener objetos. Estas regiones serán con las que se alimentará a la CNN para clasificarlas y obtener a la clase que pertenecen. Esto último se consigue a través de SVM. Por otro lado, están los detectores en tiempo real como YOLO, SSD o EfficientDet donde la red neuronal solo necesita “mirar” una vez para predecir los objetos que hay en una imagen.

2.4.1. Faster R-CNN

Faster R-CNN [18] es una extensión de Fast R-CNN [80]. Como su nombre indica, Faster R-CNN es más rápido que Fast R-CNN gracias a la red de propuesta regional ([Region Proposal Network \(RPN\)](#)).

La arquitectura de Faster R-CNN se muestra en la figura 2.12. Consta de 2 módulos:

- **RPN:** para generar propuestas regionales.

- Fast R-CNN: para detectar objetos en las regiones propuestas.

El módulo RPN es responsable de generar propuestas regionales. Aplica el concepto de atención en redes neuronales, por lo que guía al módulo de detección Fast R-CNN hacia dónde buscar objetos en la imagen.

Las capas convolucionales se comparten entre los módulos RPN y Fast R-CNN.

Faster R-CNN funciona de la siguiente manera:

- El RPN genera propuestas regionales.
- Para todas las propuestas de región en la imagen, se extrae un vector de características de longitud fija de cada región utilizando la capa de agrupación de ROI.
- Los vectores de características extraídos luego se clasifican usando Fast R-CNN.
- Se devuelven las puntuaciones de clase de los objetos detectados además de sus cuadros delimitadores.

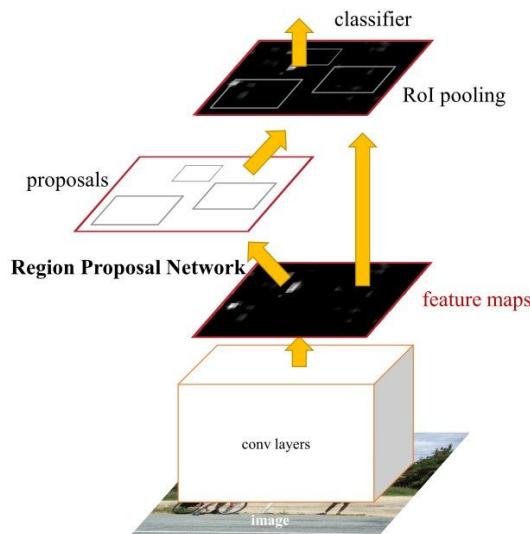


Figura 2.12: Arquitectura de Faster R-CNN [18]

Region Proposal Network (RPN)

Los modelos R-CNN y Fast R-CNN dependen del algoritmo de búsqueda selectiva para generar propuestas de región. Cada propuesta se envía a una CNN previamente capacitada para su clasificación. En [18] se propuso una red denominada red de propuestas regionales (RPN) que puede producir las propuestas regionales. Esto tiene algunas ventajas:

1. Las propuestas de región ahora se generan utilizando una red que podría entrenarse y personalizarse de acuerdo con la tarea de detección.
2. Debido a que las propuestas se generan utilizando una red, ésta se puede entrenar de un extremo a otro para personalizarla en la tarea de detección. Por lo tanto, produce mejores propuestas de región en comparación con métodos genéricos como Selective Search y EdgeBoxes.
3. El RPN procesa la imagen utilizando las mismas capas convolucionales utilizadas en la red de detección Fast R-CNN. Por lo tanto, el RPN no necesita más tiempo para producir las propuestas en comparación con los algoritmos como la búsqueda selectiva.
4. Debido a que comparten las mismas capas convolucionales, el RPN y el Fast R-CNN se pueden fusionar/unificar en una sola red. Por consiguiente, el entrenamiento se realiza solo una vez.

El **RPN** funciona en el mapa de características de salida devuelto desde la última capa convolucional compartida con Fast **R-CNN**. Esto se muestra en la figura 2.13. Sobre la base de una ventana rectangular de tamaño $n * n$, una ventana deslizante atraviesa el mapa de características. Para cada ventana, se generan varias propuestas de regiones candidatas. Estas propuestas no son las propuestas finales, ya que se filtrarán en función de su “puntuación de objetividad”.

Anchor

Como se puede ver en la figura 2.13, el mapa de características de la última capa de convolución compartida se pasa a través de una ventana deslizante rectangular de tamaño $n * n$, donde $n = 3$ para la red VGG-16. Para cada ventana, se generan propuestas de región K . Cada propuesta se parametriza según un cuadro de referencia que se denomina *anchor box*. Los 2 parámetros de los anchor boxes son:

1. Escala.
2. Relación de aspecto.

Generalmente, hay 3 escalas y 3 relaciones de aspecto y, por lo tanto, hay un total de $K = 9$ casillas de anclaje. Pero K puede ser diferente de 9. En otras palabras, las K regiones se producen a partir de cada propuesta de región, donde cada una de las K regiones varía en la escala o en la relación de aspecto. Algunas de las variaciones del anchor se muestran en la figura 2.13.

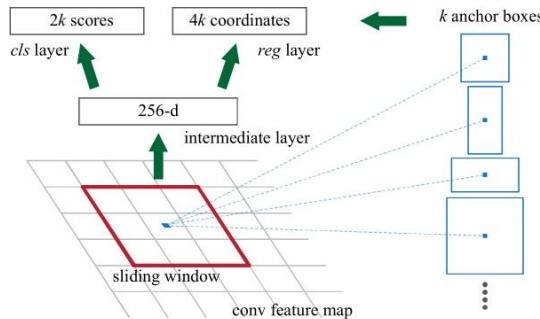


Figura 2.13: Variación de los anchor boxes Faster R-CNN [18]

Utilizando anclajes de referencia (anchor boxes), se utiliza una sola imagen a una sola escala y, al mismo tiempo, se pueden ofrecer detectores de objetos invariantes en escala, ya que los anclajes existen a diferentes escalas. Esto evita el uso de múltiples imágenes o filtros. Los anclajes de múltiples escalas son clave para compartir características en el **RPN** y la red de detección Fast **R-CNN**.

Para cada propuesta de región $n * n$, se extrae un vector de características (de longitud 256 para la red ZF y 512 para la red VGG-16). Este vector luego se alimenta a 2 capas hermanas completamente conectadas:

1. La primera capa **Fully Connected (FC)** se llama **cls** y representa un clasificador binario que genera la puntuación de objetividad para cada propuesta de región (es decir, si la región contiene un objeto o es parte del fondo).
2. La segunda capa **FC** se llama **reg**, que devuelve un vector 4-D que define el cuadro delimitador de la región.

La primera capa **FC** (es decir, clasificador binario) tiene 2 salidas. El primero es para clasificar la región como fondo y el segundo es para clasificar la región como un objeto. La siguiente sección analiza cómo se asigna la puntuación de objetividad a cada anchor box y cómo se utiliza para producir la etiqueta de clasificación.

2.4.2. SSD: Single Shot MultiBox Detector

SSD [29] es un modelo de detección de objetos en imágenes empleando únicamente una Deep Neural Network (DNN). SSD discretiza el espacio de salida de los cuadros delimitadores en un conjunto de cuadros predeterminados en diferentes proporciones y escalas por ubicación del mapa de características. En el momento de la predicción, la red genera puntuaciones para la presencia de cada categoría de objeto en cada cuadro predeterminado y produce ajustes en el cuadro para que coincida mejor con la forma del objeto. Además, la red combina predicciones de múltiples mapas de características con diferentes resoluciones para manejar de forma natural objetos de varios tamaños.

SSD tiene dos componentes: un modelo backbone y un SSD head. El modelo backbone suele ser una red de clasificación de imágenes previamente entrenadas como extractor de características. Típicamente suele ser una red como ResNet entrenada en ImageNet [81] de la que se ha eliminado la capa de clasificación final FC. Por tanto, nos quedamos con una DNN que es capaz de extraer el significado semántico de la imagen de entrada al tiempo que conserva la estructura espacial de la imagen, aunque con una resolución más baja. Para ResNet34, el backbone da como resultado un mapa de características de 256 7x7 para una imagen de entrada. El SSD head es solo una o varias capas convolucionales agregadas a este backbone y las salidas se interpretan como los cuadros delimitadores y las clases de objetos en la ubicación espacial de las activaciones de las capas finales.

En la figura 2.14, las primeras capas (cuadros blancos) son el backbone, las últimas capas (cuadros azules) representan el SSD head.

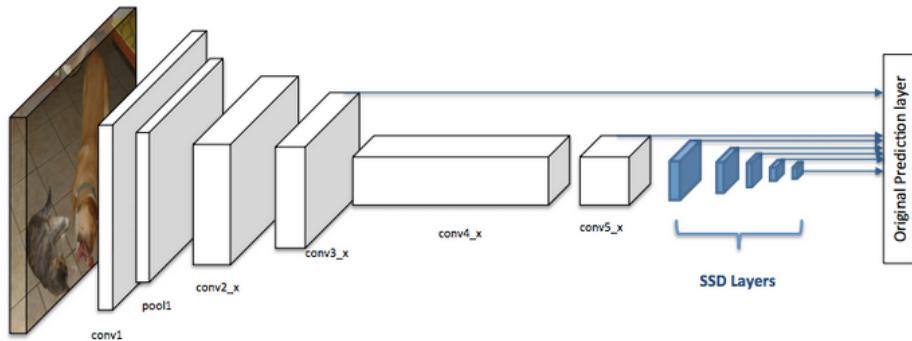


Figura 2.14: Arquitectura de una red neuronal convolucional con un detector SSD [19]

Grid cell

En lugar de usar una ventana deslizante, SSD divide la imagen usando una cuadrícula y cada celda de la cuadrícula es responsable de detectar objetos en esa región de la imagen. La detección de objetos simplemente significa predecir la clase y ubicación de un objeto dentro de esa región. Si no hay ningún objeto presente, se considera como la clase de fondo y se ignora la ubicación. Por ejemplo, como se puede observar en la figura usar una cuadrícula de 4x4 en el siguiente ejemplo. Cada celda de la cuadrícula puede mostrar la posición y la forma del objeto que contiene.

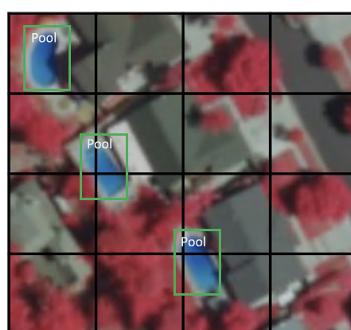


Figura 2.15: Ejemplo de una cuadrícula 4x4 [19]

Anchor box

Cada grid cell en [SSD](#) se puede asignar con múltiples anchor boxes. Estos anchor boxes están predefinidos y cada uno es responsable de un tamaño y forma dentro de una grid cell. Por ejemplo, la piscina de la imagen siguiente corresponde a la anchor box más alta, mientras que el edificio corresponde a la anchor box más ancha.

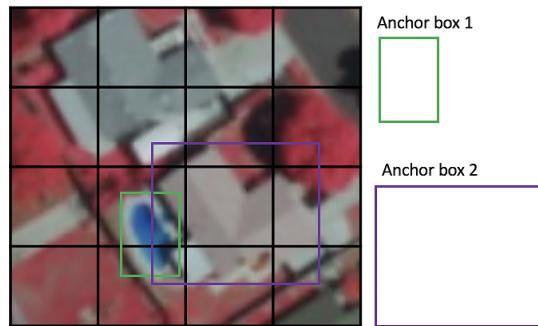


Figura 2.16: Ejemplo con 2 anchor boxes [19]

[SSD](#) usa una fase de coincidencia durante el entrenamiento, para hacer coincidir el anchor box apropiado con los cuadros delimitadores de cada objeto de ground truth dentro de una imagen. Básicamente, el anchor box con el mayor grado de superposición con un objeto es responsable de predecir la clase de ese objeto y su ubicación. Esta propiedad se utiliza para entrenar la red y para predecir los objetos detectados y sus ubicaciones una vez que la red ha sido entrenada. En la práctica, cada anchor box se especifica mediante una relación de aspecto y un nivel de zoom.

Relación de aspecto

No todos los objetos tienen forma cuadrada. Algunos son más largos y otros más anchos, en diversos grados. La arquitectura [SSD](#) permite relaciones de aspecto predefinidas de los anchor boxes para tener en cuenta esto. El parámetro de proporciones se puede utilizar para especificar las diferentes proporciones de los cuadros de anclaje asociados con cada celda de la cuadrícula en cada nivel de zoom/escala.



Figura 2.17: El cuadro delimitador del edificio 1 es más alto, mientras que el cuadro delimitador del edificio 2 es más ancho [19]

Nivel de zoom

No es necesario que los anchor boxes tengan el mismo tamaño que la grid cell. Podríamos estar interesados en encontrar objetos más pequeños o más grandes dentro de una celda de la cuadrícula. El parámetro de zoom se utiliza para especificar cuánto deben ampliarse o reducirse los cuadros de anclaje con respecto a cada celda de la cuadrícula. Al igual que lo que hemos visto en el ejemplo de el anchor box, el tamaño del edificio es generalmente más grande que la piscina.

Campo receptivo

El campo receptivo se define como la región en el espacio de entrada que está mirando una función de CNN en particular, es decir, que se ve afectada. Debido a la operación de convolución, las características en diferentes capas representan diferentes tamaños de región en la imagen de entrada. A medida que se profundiza, el tamaño representado por una característica aumenta. En este ejemplo que se puede observar en la figura 2.18, se comienza con la capa inferior (5×5) y luego se aplica una convolución que da como resultado la capa intermedia (3×3) donde una característica (píxel verde) representa una región de 3×3 de la capa de entrada (capa inferior). Después se aplica la convolución a la capa intermedia y se obtiene la capa superior (2×2) donde cada característica corresponde a una región de 7×7 en la imagen de entrada. Este tipo de matriz 2D verde y naranja también se denomina mapa de características, que se refieren a un conjunto de características creadas al aplicar el mismo extracto de características en diferentes ubicaciones del mapa de entrada en una ventana deslizante. Las características del mismo mapa de características tienen el mismo campo receptivo y buscan el mismo patrón pero en diferentes ubicaciones. Esto crea la invariancia espacial de ConvNet.

El campo receptivo es la premisa central de la arquitectura SSD, ya que permite detectar objetos a diferentes escalas y generar un cuadro delimitador más ajustado. El backbone ResNet34 genera mapas de características de $256 \times 7 \times 7$ para una imagen de entrada. Si especificamos una cuadrícula de 4×4 , el enfoque más simple es sencillamente aplicar una convolución a este mapa de características y convertirlo a 4×4 . Este enfoque puede funcionar hasta cierto punto y es exactamente la idea de YOLO. El paso extra dado por SSD es que aplica más capas convolucionales al mapa de características del backbone y hace que cada una de estas capas de convolución genere resultados de detección de objetos. Como las capas anteriores que tienen un campo receptivo más pequeño pueden representar objetos de menor tamaño, las predicciones de las capas anteriores ayudan a tratar con objetos de menor tamaño.

Debido a esto, SSD permite definir una jerarquía de celdas de cuadrícula en diferentes capas. Por ejemplo, se podría usar una cuadrícula de 4×4 para encontrar objetos más pequeños, una cuadrícula de 2×2 para encontrar objetos de tamaño medio y una cuadrícula de 1×1 para encontrar objetos que cubran toda la imagen.

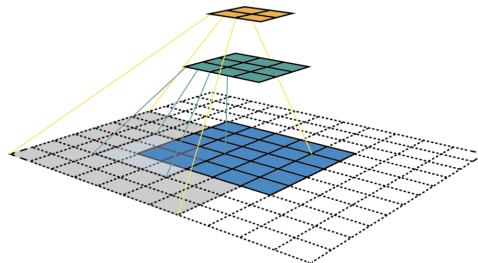


Figura 2.18: Visualización de mapas de características de CNN y campo receptivo [19]

2.4.3. EfficientDet

EfficientDet [20] se trata de un detector de objetos escalable y eficiente. Sobre la base del trabajo que se realizó anterior sobre el escalado de redes neuronales se logra una alta precisión y es hasta 9 veces más pequeño y usa significativamente menos cálculos en comparación con los detectores del Estado del Arte. En la figura 2.19 se muestra la arquitectura de red general de los modelos.

EfficientDet surge en noviembre de 2019 con la necesidad de aplicar soluciones en la mejora de la eficiencia computacional mediante la realización de un estudio sistemático de modelos de detección de última generación. Como ya se vio en la sección 2.4.2, los detectores de objetos tienen tres componentes principales: un backbone que extrae características de una imagen dada, una red de características que toma múltiples niveles de características de la red troncal como entrada y genera una lista de características fusionadas que representan características de la imagen y una red final que usa las características fusionadas para predecir la clase y ubicación de cada objeto. Al examinar las opciones de diseño para estos componentes, EfficientDet identifica varias optimizaciones clave para mejorar el rendimiento y la eficiencia.

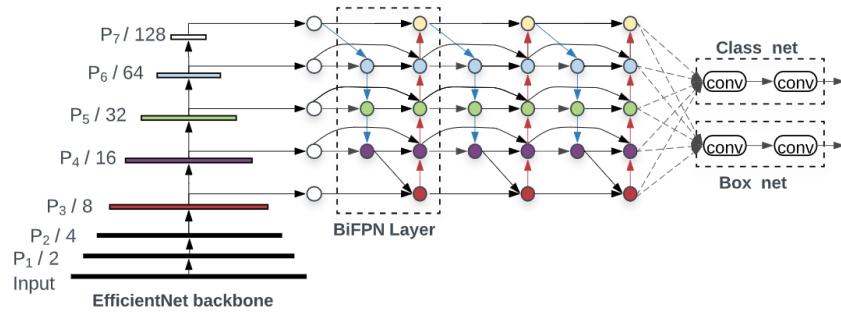


Figura 2.19: Arquitectura de EfficientDet [20]

Los detectores de objetos anteriores se basan principalmente en ResNet, ResNeXt o AmoebaNet como backbones, que son menos potentes o tienen menor eficiencia que la de EfficientNet. Al implementar primero un backbone EfficientNet, es posible lograr una eficiencia mucho mayor. Por ejemplo, a partir de una RetinaNet que emplea el backbone ResNet-50, se puede observar que simplemente reemplazar ResNet-50 con EfficientNet-B3 puede mejorar la precisión en un 3 % y reducir los cálculos en un 20 %.

Otra optimización es mejorar la eficiencia de las redes de características. Si bien la mayoría de los detectores anteriores simplemente emplean una **Feature Pyramid Network (FPN)**, se puede observar que el **FPN** de arriba hacia abajo está inherentemente limitado por el flujo de información unidireccional. Los **FPN** alternativos, como PANet, agregan un flujo ascendente adicional a costa de más cálculos. Los esfuerzos para aprovechar la **Network Architecture Search (NAS)** descubrieron la arquitectura **NAS-FPN** más compleja. Sin embargo, si bien esta estructura de red es efectiva, también es irregular y altamente optimizada para una tarea específica, lo que dificulta la adaptación a otras tareas.

Para abordar estos problemas, EfficientDet presenta una nueva red de funciones bidireccionales, **BiFPN**, que incorpora la idea de fusión de funciones multinivel de **FPN/PANet /NAS-FPN** que permite que la información fluya tanto en la dirección de arriba hacia abajo como de abajo hacia arriba, mientras utiliza conexiones regulares y eficientes.

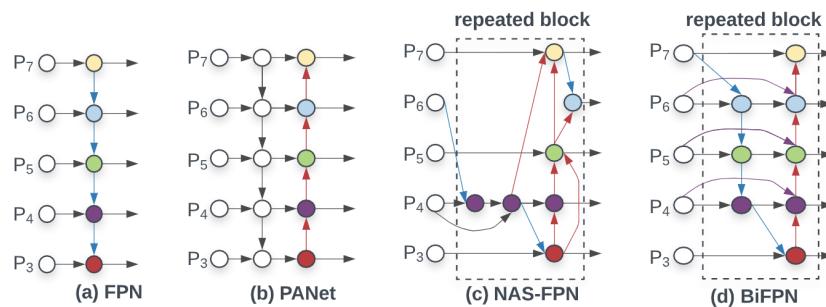


Figura 2.20: Comparativa entre biFPN y las demás redes de características previas [20]

Para mejorar aún más la eficiencia, EfficientDet plantea una nueva técnica de fusión rápida normalizada. Las propuestas tradicionales generalmente tratan todas las características de entrada al **FPN** por igual, incluso aquellas con diferentes resoluciones. Sin embargo, se observa que las características de entrada en diferentes resoluciones a menudo tienen contribuciones desiguales a las características de salida. Por lo tanto, agregando un peso adicional para cada característica de entrada se permite que la red aprenda la importancia de cada una. EfficientDet también propone reemplazar todas las convoluciones regulares con convoluciones separables en profundidad menos costosas. Con estas optimizaciones, BiFPN mejora aún más la precisión en un 4 %, al tiempo que reduce el coste de cálculo en un 50 %.

Una tercera optimización implica lograr mejores compensaciones de precisión y eficiencia bajo diferentes limitaciones de recursos. Escalar conjuntamente la profundidad, el ancho y la resolución de una red puede mejorar significativamente la eficiencia del reconocimiento de imágenes. EfficientDet ofrece un nuevo método de escalado compuesto para detectores de objetos, que escala conjuntamente la resolución/profundidad/ancho. Cada componente de la red, es decir, el backbone, la característica y la red de predicción de cuadro/clase, tendrá un único factor de escala compuesto que controla todas las dimensio-

nes de escala utilizando reglas basadas en heurísticas. Este enfoque permite determinar fácilmente cómo escalar el modelo calculando el factor de escala para las restricciones de recursos de destino dadas.

Combinando el nuevo backbone y BiFPN, surge una línea de base EfficientDet-D0 de tamaño pequeño y aplicando una escala compuesta para obtener de EfficientDet-D1 a D7. Cada modelo consecutivo tiene un coste computacional más alto, que cubre una amplia gama de restricciones de recursos desde 3 mil millones de **Floating Point Operations Per Second (FLOPS)** hasta 300 mil millones de **FLOPS**, proporcionando una mayor precisión.

2.4.4. YOLOv4

YOLO [1] es uno de los algoritmos de detección de objetos más eficientes que existen. La primera versión fue publicada por Joseph Redmon [38] en 2016 y la implementación más reciente [1] está liderada por Alexey Bochkovsky. Predice tanto la posición (representada como un cuadro delimitador) como la clasificación de objetos en imágenes.

YOLO tiene como objetivo encontrar las siguientes variables en una imagen:

- (bx, by) - el centro de un cuadro delimitador.
- (bw, bh) - el ancho y alto de un cuadro delimitador.
- c - la clase del objeto.
- P_c - la probabilidad de que haya un objeto de clase c en el cuadro.

YOLO divide la imagen en una imagen de 19x19, donde cada celda predice 5 cuadros delimitadores de la forma $y = (P_c, bx, by, bw, bh, c)$. Esto da $19 \times 19 \times 5 = 1,805$ cuadros delimitadores diferentes por imagen. La eliminación de las cajas con un P_c bajo se denomina *non-max suppression*.

La principal diferencia entre You Only Look Once v4 (YOLOv4) y las implementaciones anteriores es el enfoque en la velocidad. El objetivo del nuevo algoritmo **YOLO** es que cualquier persona que disponga de una **Graphics Processing Unit (GPU)** de alta gama pueda aplicar el algoritmo para lograr una alta precisión para el reconocimiento de imágenes ejecutándose en tiempo real. En la figura 2.21, se muestran los resultados de la comparación entre **YOLOv4** y otras arquitecturas, donde es evidente que **YOLOv4** supera a la mayoría de las otras redes neuronales en términos de precisión promedio y lo hace a más del doble de la velocidad de fotogramas. Esto diferencia mucho a **YOLOv4** de los otros algoritmos, ya que se puede utilizar en la clasificación de objetos en tiempo real con una precisión casi humana. Por ejemplo, la **CNN** puede diferenciar entre automóviles, bicicletas y camiones que circulan por una carretera. Con su alta velocidad y precisión sorprendentemente buena, **YOLO** es ampliamente adoptado y, como tal, es un éxito.

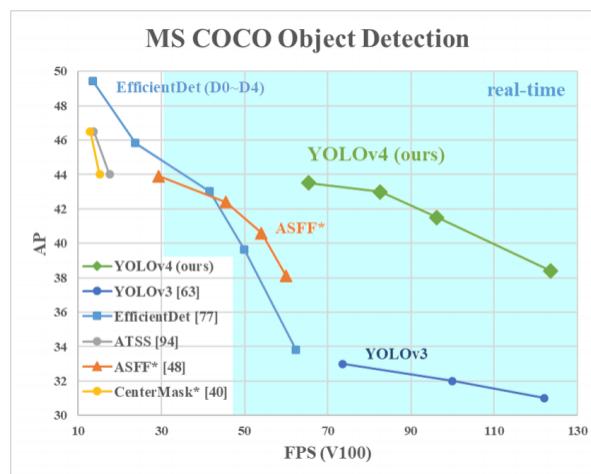


Figura 2.21: Comparativa velocidad y precisión de YOLOv4 frente a otras arquitecturas [1]

Estructura de un detector de objetos

Todos los detectores de objetos toman una imagen como entrada y comprimen las características a través de una red neuronal convolucional. En la clasificación de imágenes, estos backbones son el final de la red y se pueden hacer predicciones a partir de ellas. En la detección de objetos, es necesario dibujar varios cuadros delimitadores alrededor de las imágenes junto con la clasificación, por lo que las capas de características convolucionales del backbone deben mezclarse unas de otras. La combinación de capas de características del backbone ocurre en el neck.

También es útil dividir los detectores de objetos en dos categorías: detectores de una etapa y detectores de dos etapas. La detección ocurre en el head. Los detectores de dos etapas desacoplan la tarea de localización y clasificación de objetos para cada cuadro delimitador. Los detectores de una etapa hacen las predicciones para la localización y clasificación de objetos al mismo tiempo. **YOLO** es un detector de una etapa, por lo tanto, solo mira una vez (You Only Look Once).

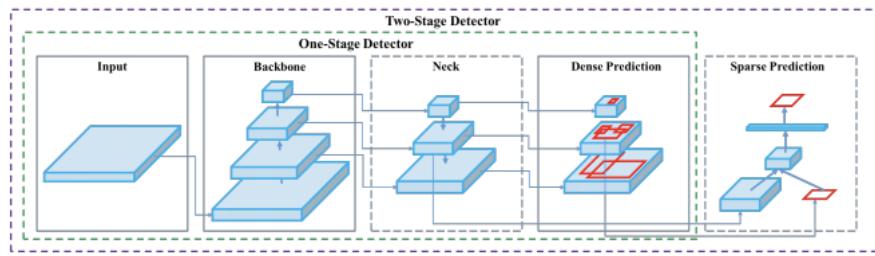


Figura 2.22: Arquitectura de detectores de objetos de una y dos etapas [1]

Backbone de YOLOv4

El backbone de la red de un detector de objetos típicamente suele estar pre-entrenada en la clasificación de ImageNet [81]. El entrenamiento previo significa que los pesos de la red ya se han adaptado para identificar características relevantes en una imagen, aunque se modificarán en la nueva tarea de detección de objetos.

Los autores consideraron los siguientes backbones para el detector de objetos **YOLOv4**:

- CSPResNext50.
- CSPDarknet53.
- EfficientNet-B3.

CSPResNext50 y CSPDarknet53 se basan en DenseNet. DenseNet fue diseñado para conectar capas en redes neuronales convolucionales con los siguientes objetivos: aliviar el problema del gradiente de desaparición (es difícil retropropulsar señales de pérdida a través de una red muy profunda), reforzar la propagación de características, alentar a la red a reutilizar características y reducir el número de parámetros de red.

En CSPResNext50 y CSPDarknet53, DenseNet se ha editado para separar el mapa de características de la capa base copiándolo y enviando una copia a través del bloque denso y enviando otra directamente a la siguiente etapa. La idea con CSPResNext50 y CSPDarknet53 es eliminar los cuellos de botella computacionales en DenseNet y mejorar el aprendizaje al pasar una versión sin editar del mapa de características.

EfficientNet fue diseñado por Google Brain para estudiar principalmente el problema de escala de las redes neuronales convolucionales. Hay muchas decisiones que puede tomar al escalar su ConvNet, incluido el tamaño de entrada, la escala de ancho, la escala de profundidad y la escala de todo lo anterior. El artículo [20] postula que hay un punto óptimo para todos estos y, a través de la búsqueda, lo encuentran.

EfficientNet supera a las otras redes de tamaño comparable en clasificación de imágenes. Los autores de **YOLOv4** postulan, sin embargo, que las otras redes pueden funcionar mejor en la configuración de detección de objetos y decidieron experimentar con todas ellas.

En base a los resultados experimentales, la red **YOLOv4** final implementa CSPDarknet53 para el backbone.

Neck de YOLOv4

El siguiente paso en la detección de objetos es mezclar y combinar las características formadas en el backbone de ConvNet para prepararse para el paso de detección. **YOLOv4** considera algunas opciones para el neck que incluyen: **FPN**, **Path Aggregation Network (PAN)**, **NAS-FPN**, **BiFPN**, **Adaptively Spatial Feature Fusion (ASFF)** y **Scale-wise Feature Aggregation Module (SFAM)**.

Los componentes del neck normalmente fluyen hacia arriba y hacia abajo entre las capas y conectan solo las pocas capas al final de la red convolucional.

Como se pudo observar en la figura 2.20, EfficientDet utiliza la búsqueda de arquitectura neuronal para encontrar la mejor forma de bloques en la parte del neck de la red, llegando a **NAS-FPN**. Los autores de EfficientDet lo modificaron ligeramente para hacer que la arquitectura sea más intuitiva (y probablemente funcione mejor en sus conjuntos de desarrollo).

YOLOv4 elige **PANet** para la agregación de funciones de la red. No hay escrito mucho sobre el fundamento de esta decisión, y dado que **NAS-FPN** y **BiFPN** se escribieron al mismo tiempo, se prevé que sea un área de investigación futura.

Además, **YOLOv4** agrega un bloque **Spatial Pyramid Pooling (SPP)** después de CSPDarknet53 para aumentar el campo receptivo y separar las características más importantes del backbone.

Head de YOLOv4

YOLOv4 implementa el mismo head **YOLO** que **YOLOv3** [82] para la detección con el anchor basado en pasos de detección y tres niveles de granularidad de detección.

2.4.5. Comparativa de los diferentes detectores

En esta sección se va a evaluar cuantitativamente las principales métricas de los detectores de objetos que se han expuesto anteriormente. Estas métricas, junto a otras muy importantes en la evaluación de modelos entrenados en las distintas arquitecturas, se explicarán con más detalle en la sección 4.2.1.

Las métricas que se recogen en las siguientes tablas han sido obtenidas en base al dataset **Microsoft Common Objects in Context (test-dev 2017) (MS COCO)**, un conjunto de datos de referencia que se emplea para la evaluar el rendimiento de los modelos de visión por computadora de última generación. Este dataset se describirá con mayor detalle en la sección 3.3.1.

En la tabla 2.1 se muestran los resultados obtenidos en los detectores **YOLOv4** y **SSD** utilizando GPU's NVIDIA Maxwell.

Como se puede observar, **YOLOv4** obtuvo un AP50 o **Mean average precision (mAP)** de 64,9 % a 31 **FPS** frente a un 48,5 % a 22 **FPS** que logró **SSD**. Cabe destacar que se están comparando ambos detectores a partir de un tamaño de imágenes de la capa de entrada a la red de 512x512. En términos de velocidad y precisión, **YOLOv4** se posiciona por encima.

Tabla 2.1: Velocidad y precisión YOLOv4 y SSD con Maxwell GPU: GTX Titan X (Maxwell) o Tesla M40 GPU [1] [29]

Method	Backbone	Size	FPS	AP	AP50
YOLOv4	CSPDarknet-53	416	38 (M)	41,2 %	62,8 %
YOLOv4	CSPDarknet-53	512	31 (M)	43,0 %	64,9 %
YOLOv4	CSPDarknet-53	608	23 (M)	43,5 %	65,7 %
SSD	VGG-16	300	43 (M)	25,1 %	43,1 %
SSD	VGG-16	512	22 (M)	28,8 %	48,5 %

En la tabla 2.2 se muestran los resultados obtenidos en los detectores **YOLOv4** y Faster **R-CNN** utilizando GPU's NVIDIA Pascal.

Como se puede observar, **YOLOv4** obtuvo un AP50 o **mAP** de 62,8 % a 54 **FPS** frente a un 59,2 % a 9,4 **FPS** que logró Faster **R-CNN**. El tamaño de las imágenes de capa de entrada a la red de Faster **R-CNN** no se especificó en el benchmark, por lo que se ha comparado con el tamaño más pequeño con el que se

realizó la evaluación en **YOLOv4**, es decir, 416x416. En términos de velocidad y precisión, **YOLOv4** se posiciona por encima.

Tabla 2.2: Velocidad y precisión YOLOv4 y Faster R-CNN con Pascal GPU: Titan X (Pascal), Titan Xp, GTX 1080 Ti, o Tesla P100 GPU [1] [18]

Method	Backbone	Size	FPS	AP	AP50
YOLOv4	CSPDarknet-53	416	54 (P)	41,2 %	62,8 %
YOLOv4	CSPDarknet-53	512	43 (P)	43,0 %	64,9 %
YOLOv4	CSPDarknet-53	608	33 (P)	43,5 %	65,7 %
Faster R-CNN	ResNet-50	—	9,4 (P)	39,8 %	59,2 %

En la tabla 2.3 se muestran los resultados obtenidos en los detectores **YOLOv4** y EfficientDet utilizando GPU's NVIDIA Volta.

Como se puede observar, **YOLOv4** obtuvo un AP50 o mAP de 64,9 % a 83 FPS frente a un 52,2 % a 62,5 FPS que logró EfficientDet. Cabe destacar que se están comparando ambos detectores a partir de un tamaño de imágenes de la capa de entrada a la red de 512x512. En términos de velocidad y precisión, **YOLOv4** se posiciona por encima.

Tabla 2.3: Velocidad y precisión YOLOv4 y EfficientDet con Volta GPU: Titan Volta or Tesla V100 GPU [1] [20]

Method	Backbone	Size	FPS	AP	AP50
YOLOv4	CSPDarknet-53	416	96 (V)	41,2 %	62,8 %
YOLOv4	CSPDarknet-53	512	83 (V)	43,0 %	64,9 %
YOLOv4	CSPDarknet-53	608	62 (V)	43,5 %	65,7 %
EfficientDet-D0	Efficient-B0	512	62,5 (V)	33,8 %	52,2 %
EfficientDet-D1	Efficient-B1	640	50,0 (V)	39,6 %	58,6 %
EfficientDet-D2	Efficient-B2	768	41,7 (V)	43,0 %	62,3 %
EfficientDet-D3	Efficient-B3	896	23,8 (V)	45,8 %	65,0 %

En vista a las métricas se puede concluir que, de todos los detectores de objetos que se han expuesto, **YOLOv4** es el mejor en cuanto a velocidad y precisión. Es preciso señalar que siempre se obtiene la misma precisión y la única métrica que se ve afectada por la utilización de una GPU u otra es la velocidad en términos de FPS. Por tanto, se tomará **YOLOv4** como algoritmo de detección de personas y objetos para el desarrollo de este proyecto.

2.5. Algoritmos de seguimiento de objetos

El seguimiento de objetos tiene como objetivo localizar uno o varios objetos de interés en cada fotograma de un vídeo. Normalmente, se ubica el objetivo dibujando el rectángulo más pequeño posible (cuadro delimitador) donde se encuentra el objeto. Las aplicaciones de seguimiento de objetos en vídeo son amplias, como por ejemplo en la conducción autónoma, videovigilancia, interacción persona-computadora o en el análisis deportivo.

Existe una estrecha relación entre el seguimiento y la detección. La detección consiste en ubicar uno o varios objetos en una imagen determinada, mientras que el objetivo del seguimiento es ubicar estos objetos a lo largo de todos los fotogramas de un vídeo. Para rastrear un objeto, primero se debe proporcionar la imagen de dicho objeto al algoritmo de rastreo, y esto se realiza mediante un algoritmo de detección (rastreadores basados en detección) o manualmente (rastreadores sin detección).

Una forma inocente de realizar el seguimiento es aplicar un algoritmo de detección a cada fotograma de un vídeo, pero hay varias razones por las que el seguimiento es necesario o útil:

- El seguimiento permite mantener las identidades de los objetos.
- La detección requiere un alto coste computacional.

- Los rastreadores sin detección permiten rastrear objetos para los que no se ha entrenado ningún detector.
- El seguimiento puede ayudar a abordar problemas comunes, como los cambios de iluminación, desenfoque de movimiento, cambio de escala, occlusiones (cuando el objetivo está parcial o completamente oculto por otro objeto durante un período de tiempo en el vídeo) o una mala calidad de la imagen.

2.5.1. Seguimiento de un objeto

En **Single Object Tracking (SOT)**, se le da al rastreador el cuadro delimitador del objetivo en el primer fotograma. El objetivo del rastreador es localizar el mismo objetivo en todos los demás fotogramas. **SOT** pertenece a la categoría de seguimiento sin detección puesto que el primer cuadro delimitador que se da al rastreador se dibuja manualmente. Esto significa que los rastreadores de un solo objeto deberían poder rastrear cualquier objeto que se les proporcione, incluso un objeto en el que no se entrenó ningún modelo de clasificación.



Figura 2.23: Ejemplo seguimiento de una única persona [21]

2.5.2. Seguimiento de múltiples objetos

En **Multi Object Tracking (MOT)**, como su nombre lo indica, se rastrean varios objetos al mismo tiempo. Se espera a que el algoritmo de seguimiento primero determine la cantidad de objetos en cada fotograma y, posteriormente, realice un seguimiento del ID de cada objeto de un fotograma al siguiente.

MOT es un problema desafiante ya que los cambios de ID son difíciles de evitar, especialmente en vídeos de grandes aglomeraciones de personas, donde se desconoce la naturaleza y la cantidad de objetos en cada fotograma. Los algoritmos **MOT** se basan en gran medida en algoritmos de detección, los cuales no son perfectos.

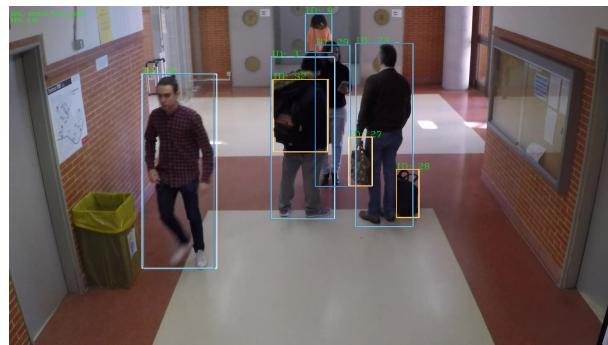


Figura 2.24: Ejemplo seguimiento de personas y objetos de interés en [4]

2.5.3. Métodos tradicionales

Mean Shift

Mean Shift es un algoritmo iterativo no paramétrico y versátil que se puede usar para muchos propósitos, como modos de búsqueda o clusterings. Se ha utilizado ampliamente en el campo de seguimiento de objetivos debido a algunas ventajas como menos tiempos de iteración y mejor rendimiento en tiempo real. Sin embargo, debido a que solo se ha utilizado la representación de histograma de un solo color de la característica de destino en el algoritmo de Mean Shift, no se puede rastrear muy bien en algunos casos, especialmente en condiciones muy complicadas. Existen principalmente dos problemas que pueden hacer que el algoritmo de cambio medio tradicional sea inestable. El primer problema es cuando el color de fondo y el color de destino son similares, el rendimiento de seguimiento es significativamente insuficiente, el segundo es el problema de la oclusión parcial.

Optical Flow

Optical Flow es el patrón de movimiento aparente de los objetos de la imagen entre dos fotogramas consecutivos causado por el movimiento del objeto o la cámara. El seguimiento con Optical Flow se basa en tres supuestos importantes:

- Consistencia de brillo: se supone que el brillo alrededor de una región pequeña permanece casi constante, aunque la ubicación de la región puede cambiar.
- Coherencia espacial: los puntos vecinos en la escena normalmente pertenecen a la misma superficie y, por lo tanto, suelen tener movimientos similares.
- Persistencia temporal: el movimiento de un parche tiene un cambio gradual.
- Movimiento limitado: los puntos no se mueven muy lejos o de forma desordenada.

Una vez que se satisfacen estos criterios, se usa método de Lucas-Kanade para obtener una ecuación para la velocidad de los puntos que se van a rastrear y, junto a técnicas de predicción, se puede rastrear un objeto dado a lo largo del vídeo.

2.5.4. Deep SORT

[Simple Online and Realtime Tracking with a Deep Association Metric \(Deep SORT\)](#) [2] es un algoritmo reciente para el seguimiento que amplía [Simple Online and Realtime Tracking \(SORT\)](#) [10] y ha mostrado resultados notables en el problema de [MOT](#).

[SORT](#) es un framework simple que emplea filtros de Kalman en el espacio de la imagen y la asociación de datos fotograma a fotograma utilizando el algoritmo húngaro con una métrica de asociación que mide el solapamiento del cuadro delimitador. Esta sencilla estrategia consigue un rendimiento favorable a altas velocidades de fotogramas. En el dataset del reto [MOT](#) [83], [SORT](#) junto con Faster [R-CNN](#) se sitúa por encima de [Multiple Hypothesis Tracking \(MHT\)](#) en las detecciones estándar. Esto remarca la influencia del rendimiento del detector de objetos en los resultados generales del seguimiento.

Aunque consigue un buen rendimiento general en términos de precisión y exactitud del seguimiento, [SORT](#) devuelve un número relativamente alto de cambios de ID. Esto se debe a que la métrica de asociación empleada sólo es precisa cuando la incertidumbre en la estimación del estado es baja. Por lo tanto, [SORT](#) tiene una deficiencia en el seguimiento a través de oclusiones, tal y como suelen aparecer en escenas de cámaras de vista frontal. Este problema se soluciona sustituyendo la métrica de asociación por una métrica más informada que combina la información sobre el movimiento y la apariencia. Aplicando una [CNN](#) se aumenta la robustez frente a fallos y oclusiones, manteniendo el sistema fácil de implementar, eficiente y aplicable a los escenarios en tiempo real.

[Deep SORT](#) adopta una metodología convencional de seguimiento de una sola hipótesis con filtro de Kalman recursivo y asociación de datos fotograma a fotograma. En las siguientes secciones se describirá con más detalle los componentes principales de este sistema.

Tratamiento de los rastreos y estimación de estado

El marco de tratamiento de rastreos y el filtro de Kalman son mayormente idénticos a la formulación original de **SORT** [10]. Se supone un escenario de seguimiento muy general en el que la cámara no está calibrada y no se dispone de información del *egomotion*. Aunque estas circunstancias suponen un reto para el marco de filtrado, es la configuración más común considerada en los recientes benchmarks de **MOT** [84]. El escenario de seguimiento se define en el espacio de estado de ocho dimensiones $(u, v, \gamma, h, \dot{u}, \dot{v}, \dot{\gamma}, \dot{h})$ que contiene la posición del centro al cuadro delimitador (u, v) , la relación de aspecto γ , la altura h y sus respectivas velocidades en coordenadas de imagen. Se utiliza un filtro de Kalman estándar con movimiento de velocidad constante y un modelo de observación lineal, en el que se toma las coordenadas de delimitación (u, v, γ, h) como observaciones directas del estado del objeto.

Para cada rastreo k se cuenta con un número de fotogramas desde la última asociación con éxito de la medición a_k . Este contador se incrementa durante la predicción del filtro de Kalman y se pone a cero cuando la pista se ha asociado a una medición. Se considera que los rastreos que superan una edad máxima predefinida A_{\max} han abandonado la escena y se eliminan del conjunto de rastreos. Se inician nuevas hipótesis de rastreos para cada detección que no puede asociarse a un rastreo existente. Estos nuevos rastreos se clasifican como tentativas durante sus tres primeros fotogramas. Durante este tiempo, se espera una asociación exitosa de la medición en cada paso de tiempo. Los rastreos que no se asocian con éxito a una medición dentro de los tres primeros fotogramas se eliminan.

Problema de asignación

Una forma convencional de resolver la asociación entre los estados de Kalman predichos y las mediciones llegadas es construir un problema de asignación que pueda resolverse mediante el algoritmo húngaro. En la formulación del problema se integra la información sobre el movimiento y la apariencia mediante la combinación de dos métricas adecuadas.

Para incorporar la información de movimiento se utiliza la distancia (al cuadrado) de Mahalanobis entre los estados de Kalman predichos y las nuevas mediciones llegadas:

$$d^{(1)}(i, j) = (\mathbf{d}_j - \mathbf{y}_i)^T \mathbf{S}_i^{-1} (\mathbf{d}_j - \mathbf{y}_i), \quad (2.6)$$

donde se denota la proyección de la i -th distribución del rastreo en el espacio de medición por $(\mathbf{y}_i, \mathbf{S}_i)$ y la j -th detección del cuadro delimitador por \mathbf{d}_j . La distancia Mahalanobis tiene en cuenta la incertidumbre de la estimación del estado midiendo cuantas desviaciones estándar se aleja la detección de la ubicación media del rastreo. Además, utilizando esta métrica es posible excluir las asociaciones poco probables mediante el umbral de la distancia de Mahalanobis en un intervalo de confianza del 95 % calculado a partir de la inversa de distribución χ^2 . Se denota esta decisión con un indicador:

$$b_{i,j}^{(1)} = \mathbb{1}[d^{(1)}(i, j) \leq t^{(1)}] \quad (2.7)$$

que evalua a 1 si la asociación entre el rastreo i -th y la detección j -th es admisible. Para el espacio de medición cuatridimensional el umbral de Mahalanobis correspondiente es $t^{(1)} = 9,4877$.

Mientras que la distancia de Mahalanobis es una métrica de asociación adecuada cuando la incertidumbre del movimiento es baja, la formulación del problema del espacio de la imagen de distribución de estado predicha obtenida del marco de filtrado de Kalman sólo proporciona una estimación aproximada de la ubicación del objeto. En particular, el movimiento de la cámara de movimiento puede introducir desplazamientos rápidos en el plano de la imagen, lo que hace que la distancia de Mahalanobis sea una métrica poco informada para el seguimiento a través de occlusiones. Por tanto, se integra una segunda métrica en el problema de asignación. Para cada cuadro de detección \mathbf{d}_j se calcula un descriptor de apariencia \mathbf{r}_j con $\|\mathbf{r}_j\| = 1$. Además, se mantiene una galería $\mathcal{R}_k = \{\mathbf{r}_k^{(i)}\}_{k=1}^{L_k}$ de los últimos $L_k = 100$ descriptores de apariencia asociados a cada rastreo k . Por tanto, la segunda métrica mide la menor distancia del coseno entre el rastreo i -th y la detección j -th en el espacio de apariencia:

$$d_{i,j}^{(2)} = \min\{1 - \mathbf{r}_j^T \mathbf{r}_k^{(i)} \mid \mathbf{r}_k^{(i)} \in \mathcal{R}_i\}. \quad (2.8)$$

Se introduce una variable binaria para indicar si una asociación es admisible según esta métrica:

$$b_{i,j}^{(2)} = \mathbb{1}[d^{(2)}(i,j) \leq t^{(2)}] \quad (2.9)$$

y se encuentra un umbral adecuado para este indicador en un conjunto de datos de entrenamiento separado. En la práctica, se aplica una CNN previamente capacitada para calcular los descriptores de apariencia del cuadro delimitador. La arquitectura de la red se explica en la sección 2.5.4.

En combinación, ambas métricas se complementan sirviendo diferentes aspectos del problema de asignación. Por un lado, la distancia de Mahalanobis proporciona información sobre las posibles ubicaciones de los objetos basadas en el movimiento que son especialmente útiles para las predicciones a corto plazo. Por otro lado, la distancia del coseno tiene en cuenta la información de la apariencia que son particularmente útiles para recuperar identidades después de oclusiones a largo plazo, cuando el movimiento es menos discriminatorio. Para construir el problema de asociación se combina ambas técnicas mediante una suma ponderada:

$$c_{i,j} = \lambda d^{(1)}(i,j) + (1 + \lambda)d^{(2)}(i,j) \quad (2.10)$$

donde se considera asociación admisible si se encuentra dentro de la puerta regional de ambas métricas:

$$b_{i,j} = \prod_{m=1}^2 b_{i,j}^{(m)}. \quad (2.11)$$

La influencia de cada métrica en el coste de asociación combinado se puede controlar mediante el hiperparámetro λ . Se establece un $\lambda = 0$ cuando hay un movimiento sustancial de la cámara. Con esta configuración solo se usa información de apariencia en el término de coste de asociación. Sin embargo, la puerta de Mahalanobis todavía se usa para ignorar asignaciones no factibles basadas en posibles ubicaciones de objetos inferidas por el filtro de Kalman.

Matching Cascade

En lugar de resolver las asociaciones de medidas de rastreos en un problema de asignación global, se introduce una cascada que resuelve una serie de subproblemas. Se considera que cuando un objeto está oculto durante un periodo de tiempo más largo, las predicciones posteriores del filtro Kalman aumentan la incertidumbre asociada a la localización del objeto. En consecuencia, la masa de probabilidad se extiende en el espacio de estados y la probabilidad de la observación se vuelve más baja. Intuitivamente, la métrica de asociación debería tener en cuenta esta dispersión de la masa de probabilidad aumentando la distancia de medición del rastreo. De forma contraria a la intuición, cuando dos rastreos compiten por la misma detección, la distancia de Mahalanobis favorece una mayor incertidumbre, porque reduce la distancia en desviaciones estándar de cualquier detección hacia la media del rastreo proyectado. Este es un comportamiento no deseado, ya que puede conducir a un aumento de las fragmentaciones de los rastreos y a la inestabilidad de las mismas. Por lo tanto, se introduce una cascada de coincidencia que da prioridad a los objetos vistos con más frecuencia para codificar la noción de dispersión de probabilidades en la probabilidad de asociación.

En la figura 2.25 se recalca el algoritmo de emparejamiento. Como entrada se proporciona el conjunto de índices de rastreos \mathcal{T} y detecciones \mathcal{D} así como la edad máxima A_{\max} . En las líneas 1 y 2 se calcula la matriz de costes de asociación y la matriz de asociaciones admisibles. A continuación, se itera sobre la edad del rastreo n para resolver un problema de asignación lineal para rastreos de edad creciente. En la línea 6 se selecciona el subconjunto de rastreos \mathcal{T}_n que no han sido asociadas a una detección en los últimos n fotogramas. En la línea 7 se resuelve la asignación lineal entre \mathcal{T}_n y las detecciones no coincidentes \mathcal{U} .

En las líneas 8 y 9 se actualiza el conjunto de coincidencias y detecciones no coincidentes, que se devuelve tras la finalización en la línea 11.

En la etapa final de emparejamiento, se ejecuta la Intersection over Union (IoU) en el conjunto de rastreos no confirmados y no emparejados de edad $n = 1$. Esto ayuda a tener en cuenta los cambios repentinos de apariencia, por ejemplo, debido a la oclusión parcial con la geometría estática de la escena, y a aumentar la robustez contra la inicialización errónea.

Listing 1 Matching Cascade

Input: Track indices $\mathcal{T} = \{1, \dots, N\}$, Detection indices $\mathcal{D} = \{1, \dots, M\}$, Maximum age A_{\max}

- 1: Compute cost matrix $\mathbf{C} = [c_{i,j}]$ using Eq. 5
- 2: Compute gate matrix $\mathbf{B} = [b_{i,j}]$ using Eq. 6
- 3: Initialize set of matches $\mathcal{M} \leftarrow \emptyset$
- 4: Initialize set of unmatched detections $\mathcal{U} \leftarrow \mathcal{D}$
- 5: **for** $n \in \{1, \dots, A_{\max}\}$ **do**
- 6: Select tracks by age $\mathcal{T}_n \leftarrow \{i \in \mathcal{T} \mid a_i = n\}$
- 7: $[x_{i,j}] \leftarrow \text{min_cost_matching}(\mathbf{C}, \mathcal{T}_n, \mathcal{U})$
- 8: $\mathcal{M} \leftarrow \mathcal{M} \cup \{(i, j) \mid b_{i,j} \cdot x_{i,j} > 0\}$
- 9: $\mathcal{U} \leftarrow \mathcal{U} \setminus \{j \mid \sum_i b_{i,j} \cdot x_{i,j} > 0\}$
- 10: **end for**
- 11: **return** \mathcal{M}, \mathcal{U}

Figura 2.25: Matching Cascade [2]**Descriptor de apariencia profunda**

Mediante el uso de consultas simples al vecino más cercano sin aprendizaje métrico adicional, se requiere una incrustación de características discriminante que debe ser entrenada fuera de línea, antes de la aplicación real de seguimiento en línea. En la versión original de Deep SORT, se empleó una CNN entrenada en el dataset de reidentificación de personas MARS [85] que contiene más de 1.100.000 imágenes de 1.261 peatones, lo que la hace muy adecuada para el aprendizaje métrico profundo en un contexto de seguimiento de personas.

Tabla 2.4: Batch final con normalización ℓ_2 proyectan las características en la hiperesfera unitaria [2]

Name	Patch Size/Stride	Output Size
Conv 1	3 x 3/1	32 x 128 x 64
Conv 2	3 x 3/1	32 x 128 x 64
Max Pool 3	3 x 3/2	32 x 64 x 32
Residual 4	3 x 3/1	32 x 64 x 32
Residual 5	3 x 3/1	32 x 64 x 32
Residual 6	3 x 3/2	64 x 32 x 16
Residual 7	3 x 3/1	64 x 32 x 16
Residual 8	3 x 3/2	128 x 16 x 8
Residual 9	3 x 3/1	128 x 16 x 8
Dense 10		128
Batch and ℓ_2 normalization		128

La arquitectura de la CNN de la red se muestra en la tabla 2.4. Se trata de una red residual amplia con dos capas convolucionales seguidas de seis bloques residuales. El mapa global de características de dimensionalidad 128 se computa en la capa densa 10. Un batch final y una normalización ℓ_2 proyectan los rasgos en la hiperesfera unitaria para que sean compatibles con la métrica de apariencia del coseno. En total la red tiene 2.800.864 parámetros, y el pase de avance de 32 cuadros delimitadores tarda aproximadamente 30 ms en una GPU NVIDIA GeForce GTX 1050. Esta red es muy adecuada para el seguimiento en línea, siempre que se disponga de una GPU moderna.

Capítulo 3

Desarrollo

Cada uno de nosotros debe trabajar para su propia mejora, y al mismo tiempo compartir una responsabilidad general para toda la humanidad.

Marie Curie

3.1. Introducción

En el presente [TFM](#) se ha desarrollado una estrategia de detección de objetos abandonados para ser aplicado en sistemas de videovigilancia. Como ya se comentó al final de la sección [2.4.5](#), se utilizará como detector de objetos y personas [YOLOv4](#) y como algoritmo de seguimiento [Deep SORT](#). A continuación, se describen las diferentes secciones que componen este capítulo.

Primero se evaluará [YOLOv4](#) sobre Darknet, framework de código abierto escrito en C y [Compute Unified Device Architecture \(CUDA\)](#), y se analizará la precisión y velocidad que se obtienen en la detección de objetos y personas. Posteriormente se convertirá el modelo de [YOLOv4](#) de Darknet a Tensorflow, framework de código abierto escrito en Python y C++ orientado al desarrollo de algoritmos inteligentes de Machine Learning. Utilizar [YOLOv4](#) con Tensorflow facilitará la programación del algoritmo de detección de objetos abandonados con Python, ya que Darknet no es un framework de uso extendido y podría ser más difícil encontrar soluciones ante los posibles errores. A continuación, se re-entrenará el modelo de la red [YOLOv4](#) sobre el dataset [Open Images Dataset v4 \(OIDv4\)](#) para observar si se obtienen mayores valores en las métricas de calidad en comparación a las obtenidas con [MS COCO](#).

Una vez obtenido el modelo de [YOLOv4](#) definitivo, se probará el algoritmo de seguimiento [Deep SORT](#). Solo nos interesa la detección y seguimiento de personas y objetos de interés, por lo que se filtrará la detección para que solo se identifiquen las clases deseadas.

Finalmente, se expondrá una estrategia para la detección de objetos abandonados basada en la detección de objetos y personas mediante CNN's y se implementará sobre el detector de objetos [YOLOv4](#) junto al algoritmo de detección [Deep SORT](#). En el planteamiento del algoritmo de detección de objetos abandonados se tendrá que considerar dos posibles escenarios. El primero es que se identifique un objeto sin propietario que se encuentre estacionario desde el comienzo de la secuencia de vídeo. En este caso, se emitirá una señal de alarma cuando se superen los 15 segundos del objeto inmóvil. En el segundo escenario se deberá de crear una asociación entre persona y objeto. Una vez establecida la asociación se podrá evaluar cuando una persona abandona un objeto de su propiedad a una distancia en píxeles 5 veces mayor a la distancia a la que se encontraba en el momento que se realizó la asociación o el objeto se encuentra en zona de alarma durante más de 15 segundos.

Cabe recalcar que en este capítulo se va a exponer cada uno de los procedimientos que se han llevado a cabo para poner en funcionamiento los algoritmos de detección, seguimiento y detección de objetos abandonados. Todos los resultados obtenidos durante el desarrollo de esta parte del proyecto se pueden consultar en el capítulo [4](#).

3.2. Detección de personas y objetos con YOLOv4

Para la utilización de **YOLOv4** se ha instalado previamente la librería opencv-python 4.1.1.26 junto con **CUDA** 10.1.243 y cuDNN 7.6.5 para poder ejecutar de manera más optima los algoritmos utilizando una **GPU** NVIDIA.

Se ha clonado el repositorio original de **YOLOv4** de GitHub [86] en el framework Darknet. Para obtener el máximo rendimiento por parte del equipo que se está utilizando se ha modificado el fichero Makefile para habilitar la utilización de la **GPU** junto con **CUDA**, así como OpenCV para poder visualizar por pantalla los resultados de las detecciones.

En la evaluación del algoritmo de detección de **YOLOv4** se va a emplear una **GPU** NVIDIA Tesla T4 del servicio cloud de Google Colab [87], un servicio cloud de Google basado en los Notebooks de Jupyter donde ya vienen instaladas la mayoría de librerías necesarias para la programación de algoritmos de Machine Learning. Esta **GPU** tiene una capacidad de computación de 7,5 por lo que será necesario descomentar una línea del fichero Makefile para indicar que se está empleando una **GPU** con esas características. Tras modificar el fichero se ha ejecutado el comando make para compilar el repositorio, de tal manera que se generan los ficheros necesarios para el funcionamiento de la red neuronal de **YOLOv4** sobre Darknet.

```

1 # Clonar el repositorio de GitHub de YOLOv4 darknet
2 git clone https://github.com/AlexeyAB/darknet
3
4 # Entrar dentro de la carpeta del repositorio
5 cd darknet
6
7 # Modificar las siguientes líneas del fichero makefile para habilitar la GPU y OPENCV
8 sed -i 's/OPENCV=0/OPENCV=1/' Makefile
9 sed -i 's/GPU=0/GPU=1/' Makefile
10 sed -i 's/CUDNN=0/CUDNN=1/' Makefile
11 sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
12 sed -i 's/# ARCH= -gencode arch=compute_75/ARCH= -gencode arch=compute_75/' Makefile
13
14 # Compilar mediante el comando make
15 make

```

Código 3.1: Evaluación del detector de objetos YOLOv4 en Darknet (1)

Otro fichero que es necesario de modificar antes de ejecutar el algoritmo de detección es el fichero yolov4.cfg que es encuentra dentro de la carpeta ./cfg. Este fichero contiene parámetros de configuración de la CNN de **YOLOv4** como el learning rate, max batches, momentum o hue, necesarios para el entrenamiento de la red, las dimensiones de las imágenes que se introducen a la capa de entrada de la red o los filtros y clases que se deben de tener en cuenta tanto para el entrenamiento como para la evaluación.

Se ha modificado los parámetros de batch y subdivisions con un valor de 1, ya que se va a usar la red para testear y no para entrenarla. Estos parámetros determinan la cantidad de imágenes que son procesadas de manera paralela.

```

1 # Cambiar batch y subdivisions de yolov4.cfg para test
2 cd cfg
3 sed -i 's/batch=64/batch=1/' yolov4.cfg
4 sed -i 's/subdivisions=8/subdivisions=1/' yolov4.cfg
5 cd ..
6
7 # Descargar los pesos pre-entrenados de YOLOv4
8 wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.
     weights
9
10 # Ejecutar detector de objetos en video con YOLOv4 en Darknet
11 ./darknet detector demo cfg/coco.data cfg/yolov4.cfg yolov4.weights ./data/videos-datasets/{
     input_file_name.avi} -i 0 -out_filename ./results/{output_file_name.avi}

```

Código 3.2: Evaluación del detector de objetos YOLOv4 en Darknet (2)

Se han descargado los pesos de **YOLOv4** previamente entrenados sobre el dataset **MS COCO**. Por último, se ha ejecutado el algoritmo de detección de objetos. En el código 3.3 se especifica que la detección es en formato vídeo, se va a utilizar **YOLOv4** como detector y la ubicación tanto del vídeo de entrada que se quiere procesar como el vídeo de salida resultante de las detecciones.

```

1 # Descargar los pesos pre-entrenados de YOLOv4
2 wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.
   weights
3
4 # Ejecutar detector de objetos en video con YOLOv4 en Darknet
5 ./darknet detector demo cfg/coco.data cfg/yolov4.cfg yolov4.weights ./data/videos-datasets/{
   input_file_name.mp4} -i 0 -out_filename ./results/{output_file_name.avi}

```

Código 3.3: Evaluación del detector de objetos YOLOv4 en Darknet (3)

En la figura 3.1 se muestra el funcionamiento de **YOLOv4** en Darknet donde se puede observar un escenario lleno de personas y equipajes como bolsas de mano, mochilas y maletas.

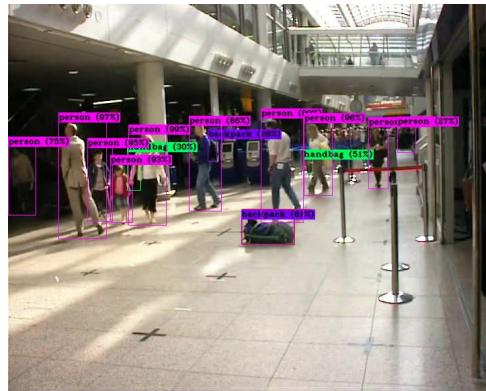


Figura 3.1: Detección de objetos con YOLOv4 en Darknet en [5]

Las últimas evaluaciones del algoritmo de detección de **YOLOv4** en Darknet se han realizado con Google Colab. En el siguiente [link](#) se puede ejecutar el Notebook de Google Colab donde se realizan los mismos pasos descritos anteriormente.

Durante el desarrollo del proyecto se ha decidido convertir el modelo de Darknet a Tensorflow, ya que se trata de un framework que emplea Python y facilitará la tarea de diseño del algoritmo de detección de objetos al no basarse únicamente en Darknet, framework que no es de uso extendido y puede ser una tarea complicada su utilización cuando aparezcan posibles errores.

Para la utilización de **YOLOv4** basado en Tensorflow se necesita convertir el modelo. La implementación que se expone a continuación está totalmente inspirada en [88]. A continuación, se describen los pasos que se han seguido para poner en funcionamiento de **YOLOv4** sobre Tensorflow 2.3.0rc0.

Primero se ha clonado el repositorio [89], el cual es un *fork* de [88], y se han instalado las librerías y dependencias necesarias que se contemplan en el fichero `requirements-gpu.txt`. Para facilitar su instalación se ha creado con Anaconda un entorno virtual. La instalación de Anaconda y el entorno virtual se explica con mayor detalle en la sección C.2.2 y en la sección C.2.4.

```

1 # Descarga del repositorio de GitHub
2 git clone https://github.com/jmudy/tensorflow-yolov4-tflite
3
4 # Entrar dentro de la carpeta del repositorio
5 cd tensorflow-yolov4-tflite
6
7 # Descarga de los pesos de YOLOv4
8 wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.
   weights -P ./data/
9
10 # Crear entorno de Anaconda basado en Python 3.7.0 y acceder a el
11 conda env create -f conda-gpu.yml
12 conda activate yolov4-gpu

```

Código 3.4: Evaluación del detector de objetos YOLOv4 en Tensorflow (1)

En el código 3.5 se muestra el comando para convertir el modelo pre-entrenado de **YOLOv4** en Darknet a Tensorflow. Se ha especificado un tamaño de 608x608, por dos motivos. El primer motivo es que es el mismo tamaño de redimensionamiento de las imágenes a la entrada de la capa de la red que en Darknet, con lo cual, se puede comparar ambos modelos en las mismas condiciones. El segundo, es que se han obtenido mejores resultados que empleando el clásico tamaño de 416x416.

```

1 # Convertir pesos de YOLOv4 Darknet a Tensorflow
2 python save_model.py --weights ./data/yolov4.weights --output ./checkpoints/yolov4-608 --
   input_size 608 --model yolov4

```

Código 3.5: Evaluación del detector de objetos YOLOv4 en Tensorflow (2)

En el código 3.6 se puede apreciar el comando que se ha utilizado para la ejecución del algoritmo de detección de objetos en **YOLOv4** en Tensorflow. Como se puede observar en las primeras líneas del script **detect_video.py** [88], se disponen de una serie de flags donde se pueden indicar una gran variedad de parámetros. En este caso, se ha indicado que se quiere utilizar un modelo de **YOLOv4** convertido con tamaño de redimensionamiento 608x608 (si no se indica, el código por defecto busca un modelo de tamaño 416x416). También se ha especificado que se quiere emplear como detector **YOLOv4**, ya que este repositorio también permite ejecutar la detección de objetos sobre **YOLOv3**. Del mismo modo que con Darknet, en el siguiente [link](#) se puede consultar el Notebook de Google Colab donde se realizan los mismos pasos descritos anteriormente.

```

1 # Ejecutar algoritmo de detección de objetos YOLOv4 en Tensorflow
2 python detect_video.py --weights ./checkpoints/yolov4-608 --size 608 --model yolov4 --video
   {input_file_name.mp4} --output {output_file_name.avi}

```

Código 3.6: Evaluación del detector de objetos YOLOv4 en Tensorflow (3)

En la figura 3.2 se muestra un ejemplo del funcionamiento de **YOLOv4** en Tensorflow donde se puede observar el hall de la **Escuela Politécnica Superior (EPS)** llena de personas y equipajes como bolsas de mano y maletas.

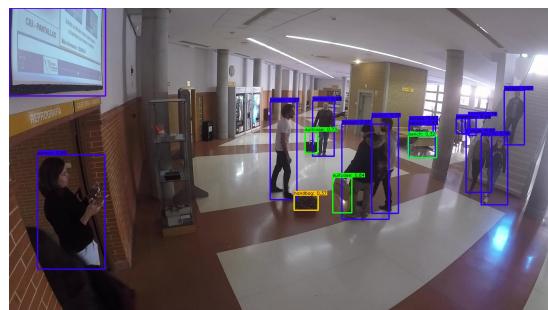


Figura 3.2: Detección de objetos con YOLOv4 en Tensorflow en [4]

Como se ilustra en la figura 3.3 se puede apreciar que la precisión en las detecciones de **YOLOv4** en Darknet y Tensorflow son prácticamente idénticas. Por tanto, se continuará el desarrollo del proyecto empleando Tensorflow como framework para la evaluación y diseño de los distintos algoritmos utilizados. En la sección 4.3.3 se analizará con detalle las detecciones de objetos de **YOLOv4** en Tensorflow.

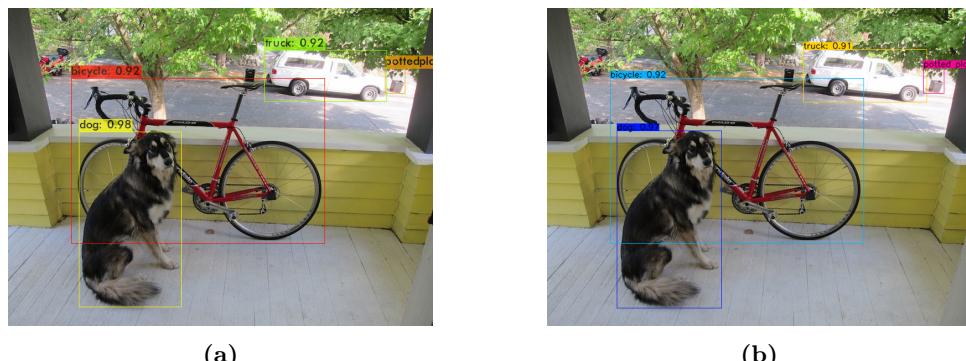


Figura 3.3: Detecciones de YOLOv4 con Darknet y Tensorflow. (a) Detección de YOLOv4 con Darknet. (b) Detección de YOLOv4 con Tensorflow.

3.3. Datasets utilizados para el evaluación de YOLOv4

La detección de objetos con [YOLOv4](#) está basada en [MS COCO](#), un dataset que contiene 80 categorías de objetos muy diversos y que se adaptan excelentemente a la problemática que se presenta en este proyecto. Contiene imágenes de nuestras clases de interés: personas, mochilas y maletas.

Para poder contrastar los resultados obtenidos en la evaluación de las métricas de calidad de [MS COCO](#) sobre [YOLOv4](#), se va a emplear el dataset [OIDv4](#), donde previamente se entrenará un modelo de detección en base a las imágenes de [OIDv4](#). La ventaja de utilizar [OIDv4](#) es que dispone de 600 categorías de objetos para evaluar los algoritmos de detección de objetos. De esta manera, se puede construir un gran dataset más específico con más clases de interés que las que ofrece [MS COCO](#).

En las siguientes secciones se explicará en mayor profundidad y detalle como están formados los datasets [MS COCO](#) y [OIDv4](#).

3.3.1. MS COCO Dataset

El dataset [MS COCO](#) [3] es un conjunto de datos de referencia utilizado para evaluar el rendimiento de los modelos entrenados por visión por computadora. Está diseñado para representar una amplia gama de objetos que encontramos regularmente en la vida cotidiana (ver figura 3.4).

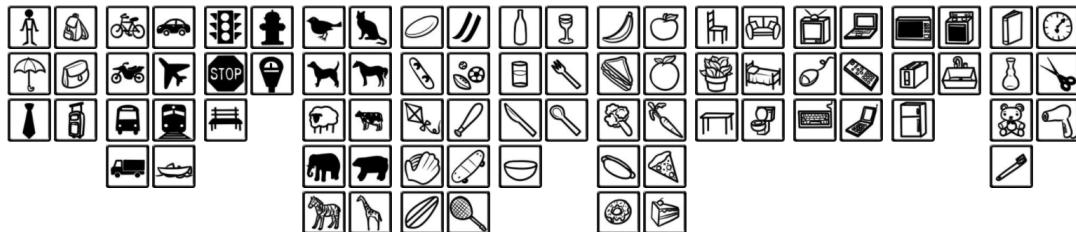


Figura 3.4: Categorías de objetos del dataset MS COCO [22]

[MS COCO](#) está etiquetado en un formato especial llamado COCO JSON, y proporciona datos para entrenar modelos supervisados de visión por computadora que son capaces de identificar los objetos comunes del conjunto de datos. Estos modelos están lejos de ser perfectos, por lo que el dataset [MS COCO](#) proporciona un punto de referencia para evaluar la mejora periódica de estos modelos a través de la investigación en visión por computadora.

Otra motivación para el dataset [MS COCO](#) es proporcionar un conjunto de datos base para entrenar modelos de visión por computadora. Una vez entrenado el modelo, se puede perfeccionar para aprender otras tareas, como las que se van a describir a continuación.

Tareas de MS COCO

[MS COCO](#) tiene múltiples tareas de visión por computadora. A continuación, se enumeran en orden decreciente en base a su uso:

- **Detección de objetos:** los objetos se anotan con un cuadro delimitador y una etiqueta de clase. El dataset [MS COCO](#) tiene 121.408 imágenes para detección de objetos, 883.331 anotaciones de objetos, 80 clases y una resolución media de imagen de 640x480.



Figura 3.5: MS COCO detección de objetos [22]

- **Segmentación semántica:** los límites de los objetos se etiquetan con una máscara y las clases de objetos se etiquetan con una etiqueta de clase. La segmentación semántica requiere modelos para trazar los límites entre los objetos.



Figura 3.6: MS COCO segmentación semántica [22]

- **Detección de puntos clave:** las personas son etiquetadas con puntos claves de interés (como pueden ser codos, rodillas o cabezas). El dataset [MS COCO](#) dispone de 250.000 personas con puntos clave etiquetados.



Figura 3.7: MS COCO detección de puntos clave [22]

3.3.2. Open Images Dataset v4

[OIDv4](#) [9] es un conjunto de datos de 9,2 millones de imágenes con anotaciones en formato .txt unificadas para la clasificación de imágenes, detección de objetos y detección de relaciones visuales (ver figura 3.8). Las imágenes tienen una licencia Creative Commons Attribution que permite compartir y adaptar el material descargado de Flickr sin una lista predefinida de nombres de clases o etiquetas.

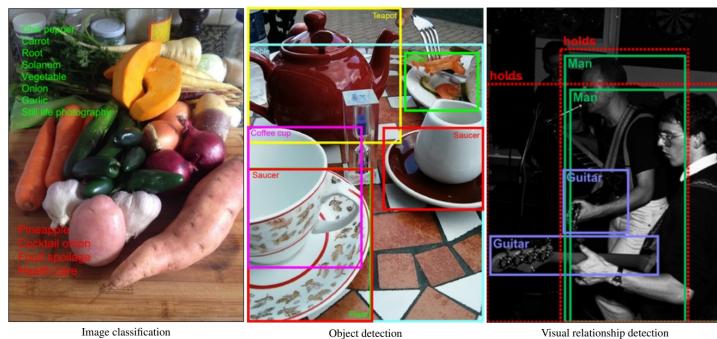


Figura 3.8: Ejemplo anotaciones en Open Images Dataset v4 [9]

[OIDv4](#) ofrece una gran escala en varias dimensiones: 30,1 millones de etiquetas a nivel de imagen para 19,8 mil conceptos, 15,4 millones de cuadros delimitadores para las 600 clases de objetos que se muestran

en la figura 3.9, y 375 mil anotaciones de relaciones visuales que involucra a 57 clases. Para la detección de objetos se proporciona más de 15 veces cuadros delimitadores que otros grandes datasets como [MS COCO](#) o ImageNet. Las imágenes suelen mostrar escenas complejas con varios objetos (de promedio tiene 8 objetos anotados por imagen).



Figura 3.9: Categorías de objetos del dataset Open Images Dataset v4 [9]

3.4. Evaluación de las métricas de calidad de MS COCO dataset

En esta sección se va a realizar la evaluación de las métricas de calidad del dataset [MS COCO](#) para tener una referencia a la hora de valorar, si re-entrenando la red, se pueden obtener mejores métricas para la aplicación del proyecto.

Primero, se ha descargado el repositorio oficial de [YOLOv4](#) de GitHub [86]. Para aprovechar maximizar la eficiencia de la evaluación se ha modificado el fichero Makefile para habilitar la [GPU](#) junto con [CUDA](#) y OpenCV. En esta evaluación se ha utilizado una NVIDIA Tesla T4, por lo que también ha sido necesario especificar que se va a utilizar una [GPU](#) con una capacidad de cálculo de 7,5.

```

1 # Clonar el repositorio de GitHub de YOLOv4 darknet
2 git clone https://github.com/AlexeyAB/darknet
3
4 # Entrar dentro de la carpeta del repositorio
5 cd darknet
6
7 # Modificar las siguientes líneas del fichero makefile para habilitar la GPU y OPENCV
8 sed -i 's/OPENCV=0/OPENCV=1/' Makefile
9 sed -i 's/GPU=0/GPU=1/' Makefile
10 sed -i 's/CUDNN=0/CUDNN=1/' Makefile
11 sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
12 sed -i '# ARCH= -gencode arch=compute_75/ARCH= -gencode arch=compute_75/' Makefile
13
14 # Compilar mediante el comando make
15 make

```

Código 3.7: Evaluación de las métricas de calidad de MS COCO en YOLOv4 Darknet (1)

Dentro de la carpeta del repositorio se ha creado una carpeta llamada `images` donde se almacenarán las imágenes del entrenamiento y validación del modelo de YOLOv4. Tal y como se muestra en el código 3.8 se han descargado los ficheros comprimidos de las imágenes desde la web oficial de MS COCO [22].

```

1 # Crear y entrar en la carpeta ./images donde descargaremos las imagenes
2 mkdir images
3 cd images
4
5 # Descargar, descomprimir y borrar archivo comprimido .zip de las imagenes de entrenamiento
6 wget -c http://images.cocodataset.org/zips/train2017.zip
7 unzip -q train2017.zip
8 rm train2017.zip
9
10 # Descargar, descomprimir y borrar archivo comprimido .zip de las imagenes de validacion
11 wget -c http://images.cocodataset.org/zips/val2017.zip
12 unzip -q val2017.zip
13 rm val2017.zip
14
15 # Vuelta a la carpeta raiz
16 cd ..

```

Código 3.8: Evaluación de las métricas de calidad de MS COCO en YOLOv4 Darknet (2)

En el siguiente link se encuentran los ficheros `train2017.txt` y `val2017.txt`, que contienen las rutas relativas de las imágenes de entrenamiento y validación. Se han copiado dentro de la carpeta `./data`. Por otro lado, se han copiado los ficheros de las etiquetas de las imágenes de entrenamiento que se encuentran dentro de la carpeta comprimida `./labels/train2017` y pegado dentro de la carpeta `./images/train2017` del repositorio junto a las imágenes de entrenamiento. Del mismo modo, se han copiado los ficheros de las etiquetas de las imágenes de validación que se encuentran dentro de la carpeta comprimida `./labels/val2017` y pegado dentro de la carpeta `./images/val2017` del repositorio junto a las imágenes de validación.

Es necesario modificar el fichero `./cfg/coco.data` para especificar por un lado, el número de clases que tiene el modelo que se quiere evaluar (no se ha tocado ya porque viene por defecto las 80 clases que componen el dataset MS COCO), la ubicación de los ficheros `train2017.txt` y `val2017.txt` y el fichero `coco.names` que contiene el nombre de cada una de las 80 clases.

```

1 classes = 80
2 train   = data/train2017.txt
3 valid   = data/val2017.txt
4 names   = data/coco.names
5 #backup = /home/pjreddie.backup/

```

Código 3.9: Fichero coco.data

Por último, se han descargado los pesos pre-entrenados de YOLOv4 y ejecutado el siguiente comando para evaluar las métricas de calidad en base al dataset MS COCO.

```

1 # Descargar los pesos pre-entrenados de YOLOv4
2 wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.weights
3
4 # Ejecutar evaluacion de las metricas de calidad de MS COCO sobre YOLOv4
5 ./darknet detector map ./cfg/coco.data ./cfg/yolov4.cfg ./yolov4.weights

```

Código 3.10: Evaluación de las métricas de calidad de MS COCO en YOLOv4 Darknet (3)

Al finalizar el proceso de evaluación de las métricas se observan las métricas de calidad más relevantes. Los resultados obtenidos servirán de referencia a la hora de comparar con los resultados que se obtengan en el entrenamiento de YOLOv4 con el dataset OIDv4. En la sección 4.2.1 se explicarán con mayor detalle cada una de las métricas que se han obtenido en la evaluación y en la sección 4.3.2, se expondrán los resultados obtenidos tras la evaluación de las métricas de calidad.

Se ha creado un Notebook de Google Colab para la evaluación de las métricas de calidad de MS COCO donde se ha seguido el mismo procedimiento de esta sección. Se puede acceder entrando en este link.

3.5. Entrenamiento YOLOv4 con Open Image Dataset v4

En las siguientes secciones se describen los pasos que se han seguido para entrenar un modelo de **YOLOv4** con Darknet a partir de un dataset personalizado basado en imágenes de **OIDv4**.

3.5.1. Recopilación y etiquetado del dataset personalizado

Para descargar las imágenes de entrenamiento y validación que se utilizarán en el entrenamiento del modelo **YOLOv4** se ha utilizado la herramienta **OIDv4_Toolkit** [23]. Se trata de un repositorio disponible en GitHub que permite la descarga de imágenes del dataset de **OIDv4** para el entrenamiento de algoritmos de detección. Se ha decidido utilizar esta herramienta porque tiene disponible un script que convierte las etiquetas del formato por defecto de **OIDv4** a Darknet.

Primero se ha descargado el repositorio de GitHub [23] e instalado las librerías y dependencias necesarias. Se ha instalado un entorno virtual con Anaconda para que no interfiera con las versiones de las librerías con los otros repositorios utilizados en el proyecto.

```

1 # Clonar el repositorio de Github
2 git clone https://github.com/theAIGuysCode/OIDv4_ToolKit.git
3 cd OIDv4_ToolKit
4
5 # Instalacion de las librerias y dependencias (recomendable crear un entorno virtual con
   Anaconda)
6 pip install -r requirements.txt

```

Código 3.11: Descarga dataset Open Images Dataset v4 (1)

Ejecutando los comandos del código 3.12 se han descargado las imágenes de entrenamiento y validación de las clases: ‘Person’, ‘Handbag’, ‘Backpack’ y ‘Suitcase’. Se ha limitado a 1.500 las imágenes de cada clase para el entrenamiento y a 300 las imágenes de validación (20 % del tamaño del entrenamiento).

```

1 # Descarga de las imagenes de entrenamiento con un limite de 1500
2 python main.py downloader --classes Person Handbag Backpack Suitcase --type_csv train --
   limit 1500 --multiclasses 1
3
4 # Descarga de las imagenes de validacion con un limite de 300
5 python main.py downloader --classes Person Handbag Backpack Suitcase --type_csv validation --
   limit 300 --multiclasses 1

```

Código 3.12: Descarga dataset Open Images Dataset v4 (2)

La estructura de las etiquetas del dataset de Open Images Dataset v4 sigue la siguiente distribución de atributos: {nombre_de_la_clase, left_x, top_y, right_x, bottom_y} (ver figura 3.10).

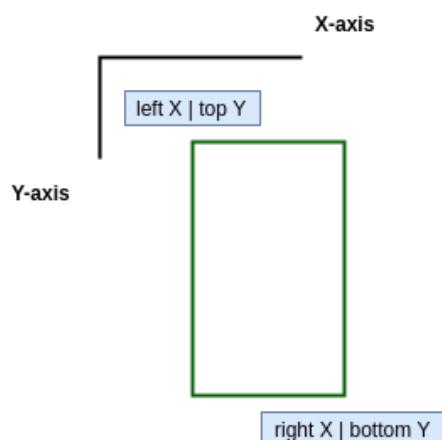


Figura 3.10: Estructura de las etiquetas de Open Images Dataset v4 [23]

Las etiquetas que se obtienen mediante esta herramienta no tienen el formato deseado. Las etiquetas en Darknet tienen la siguiente estructura `{label_ID, x_center_norm, y_center_norm, width_norm, height_norm}`

donde:

- **label_ID**: es un número de índice del fichero `coco.names`, empezando por el 0 y seguido de números enteros crecientes.
- $x_center_norm = x_center_abs / \text{image_width}$.
- $y_center_norm = y_center_abs / \text{image_height}$.
- $x_center_norm = \text{width_of_the_label_abs} / \text{image_width}$.
- $x_center_norm = \text{height_of_the_label_abs} / \text{image_height}$.

Es preciso señalar que todos los atributos de posición de las etiquetas de Darknet no son absolutos, sino normalizados.

Afortunadamente, el repositorio [23] dispone del script `convert_annotations.py` para convertir el etiquetado tanto de las imágenes de entrenamiento como las de validación del dataset **OIDv4** a Darknet. Antes de su ejecución, se han sustituido en el fichero `classes.txt`, situado en la raíz del repositorio, las clases que habían por defecto por el nombre de las clases que se han descargado las imágenes, en el mismo orden que cuando se nombraron en el código 3.12 para que la conversión se realice correctamente. Después se puede ejecutar la siguiente línea de código:

```
1 # Convertir etiquetas al formato de Darknet
2 python convert_annotations.py
```

Código 3.13: Descarga dataset Open Images Dataset v4 (3)

Al finalizar la conversión de las etiquetas, se han borrado las carpetas `Label` que contienen el etiquetado en el formato original. Las carpetas `obj` y `test` con las imágenes de entrenamiento y validación y sus etiquetas se han copiado a la carpeta `./data` del repositorio de **YOLOv4** con Darknet, ya que se realizará el entrenamiento del modelo **YOLOv4** en base al dataset personalizado en ese repositorio [86].

3.5.2. Configuración de ficheros para el entrenamiento

Antes de comenzar el entrenamiento de la red con el dataset **OIDv4** es necesario configurar el fichero `.cfg` donde se encuentran las variables de la CNN de **YOLOv4**.

Se ha generado el fichero `yolov4-obj.cfg` basado en `yolov4.cfg`, y se han modificado los siguientes parámetros en base a las recomendaciones indicadas en [86] para el entrenamiento:

- **batch = 64** (Cantidad de imágenes y etiquetas que se calculan de una pasada en el cálculo del gradiente y se actualiza los pesos a través del *backpropagation*).
- **subdivisions = 16** (Cantidad de `batches` que se subdividen en cada bloque. Las imágenes de cada bloque se ejecutan en paralelo en la **GPU**).
- **width = 416** (Ancho del tamaño de la red. Cada imagen se redimensiona al tamaño de la red durante el entrenamiento y la detección).
- **height = 416** (Altura del tamaño de la red. Cada imagen se redimensiona al tamaño de la red durante el entrenamiento y la detección).
- **max_batches = (# de clases) * 2.000 = 4 * 2.000 = 8.000** (Número máximo de `batches`).
- **steps = 6.400** (80 % de `max_batches`), **7.200** (90 % de `max_batches`) (Ajuste del `learning rate` después de un determinado número de `batches`).
- **filters = (# de clases + 5) * 3 = (4 + 5) * 3 = 27** (Número de núcleos convolucionales que contiene cada capa).

Se ha creado el fichero `obj.data` basado en el `coco.data` que se vio en la sección 3.4. En este caso, se ha especificado que el número de clases que se va a entrenar es de 4, la ubicación de los ficheros `train.txt`, `test.txt`, `coco.names` y la carpeta donde se generarán los ficheros `.weights` con los pesos de la red entrenada cada 1.000 iteraciones.

```

1 classes = 4
2 train = data/train.txt
3 valid = data/test.txt
4 names = data/obj.names
5 backup = /mydrive/yolov4/backup

```

Código 3.14: Fichero `obj.data`

Otro fichero generado ha sido el `obj.names`, basado en el `coco.names` que trae por defecto YOLOv4. En este fichero se han sustituido las 80 clases de MS COCO que vienen predeterminadas por las 4 clases que se desean entrenar. El orden de escritura ha sido el mismo que el fichero `classes.txt`. Como ya se explicó en la sección 3.5.1, el primer parámetro de las etiquetas de las imágenes de entrenamiento y validación es el ID de cada clase, por lo que debe de corresponder a la misma que se escribe en este fichero.

```

1 Person
2 Handbag
3 Backpack
4 Suitcase

```

Código 3.15: Fichero `obj.names`

Los últimos archivos de configuración necesarios antes de comenzar el entrenamiento son los ficheros `train.txt` y `test.txt`, los cuales contienen las rutas relativas a todas las imágenes de entrenamiento e imágenes de validación.

Ejecutando el siguiente script en Python se genera el fichero `train.txt`.

```

1 import os
2
3 image_files = []
4 os.chdir(os.path.join("data", "obj"))
5 for filename in os.listdir(os.getcwd()):
6     if filename.endswith(".jpg"):
7         image_files.append("data/obj/" + filename)
8 os.chdir("..")
9 with open("train.txt", "w") as outfile:
10     for image in image_files:
11         outfile.write(image)
12         outfile.write("\n")
13     outfile.close()
14 os.chdir("..")

```

Código 3.16: Generación del fichero `train.txt`

Para generar el fichero `test.txt`:

```

1 import os
2
3 image_files = []
4 os.chdir(os.path.join("data", "test"))
5 for filename in os.listdir(os.getcwd()):
6     if filename.endswith(".jpg"):
7         image_files.append("data/test/" + filename)
8 os.chdir("..")
9 with open("test.txt", "w") as outfile:
10     for image in image_files:
11         outfile.write(image)
12         outfile.write("\n")
13     outfile.close()
14 os.chdir("..")

```

Código 3.17: Generación del fichero `test.txt`

3.5.3. Entrenamiento del dataset personalizado

Se han descargado los pesos de las capas convolucionales del modelo pre-entrenado `yolov4.conv.137`. El uso de estos pesos hace que el detector de objetos personalizado ayude a que converja, a que sea mucho más preciso y que no tenga que entrenar tanto tiempo.

Como último paso, se ha ejecutado el comando para realizar el entrenamiento del detector de objetos personalizado con **YOLOv4**. Dado que se ha utilizado un conjunto de imágenes para la validación, se ha utilizado el flag `-map` para que se genere un diagrama con el **mAP** con la finalidad de poder visualizar la precisión del modelo entrenado.

```

1 # Descarga de los pesos de las capas convolucionales de YOLOv4
2 wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.
   conv.137
3
4 # Comenzar a entrenar la red neuronal del dataset personalizado de OIDv4
5 ./darknet detector train data/obj.data cfg/yolov4-obj.cfg yolov4.conv.137 -dont_show -map

```

Código 3.18: Entrenamiento del dataset personalizado

Durante el entrenamiento se puede visualizar por terminal información muy relevante como la que se muestra en la figura 3.11.

```

Tensor Cores are used.
(next mAP calculation at 4756 iterations)
4756: 1.817470, 2.770160 avg loss, 0.001000 rate, 4.198377 seconds, 304384 images, 6.902508 hours left
Resizing to initial size: 416 x 416 try to allocate additional workspace_size = 111.05 MB
CUDA allocate done!
calculation mAP (mean average precision)...
Detection layer: 139 - type = 28
Detection layer: 150 - type = 28
Detection layer: 161 - type = 28
396
detections_count = 5779, unique_truth_count = 703
class_id = 0, name = Person, ap = 34.69% (TP = 334, FP = 634)
class_id = 1, name = Handbag, ap = 85.85% (TP = 45, FP = 6)
class_id = 2, name = Backpack, ap = 69.47% (TP = 23, FP = 13)
class_id = 3, name = Suitcase, ap = 44.75% (TP = 18, FP = 22)

for conf_thresh = 0.25, precision = 0.38, recall = 0.60, F1-score = 0.47
for conf_thresh = 0.25, TP = 420, FP = 675, FN = 283, average IoU = 28.07 %

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 0.586899, or 58.69 %
Total Detection Time: 13 Seconds

Set -points flag:
`-points 10` for MS COCO
`-points 11` for PascalVOC 2007 (uncomment 'difficult' in voc.data)
`-points 0` (AUC) for ImageNet, PascalVOC 2010-2012, your custom dataset

mean_average_precision (mAP@0.5) = 0.586899
New best mAP!
Saving weights to /mydrive/yolov4/backup/yolov4-obj_best.weights
Loaded: 0.000113 seconds
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.700612), count: 32, class_loss = 6.096747, iou_loss = 55.813667, total_loss = 61.910416
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.775456), count: 49, class_loss = 5.580284, iou_loss = 12.592869, total_loss = 18.173153
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.790256), count: 26, class_loss = 1.332650, iou_loss = 2.060724, total_loss = 3.393374
total_bbox = 213715, rewritten_bbox = 0.624196 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.000000), count: 1, class_loss = 0.336086, iou_loss = 0.000000, total_loss = 0.336086
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.762050), count: 25, class_loss = 7.279247, iou_loss = 4.255256, total_loss = 11.534503
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.769026), count: 25, class_loss = 5.818727, iou_loss = 1.423375, total_loss = 6.442101
total_bbox = 213765, rewritten_bbox = 0.624097 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.663652), count: 22, class_loss = 3.985775, iou_loss = 31.648819, total_loss = 35.634594
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.767802), count: 31, class_loss = 5.791464, iou_loss = 14.015648, total_loss = 19.007112
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.810753), count: 18, class_loss = 2.353435, iou_loss = 1.116390, total_loss = 3.469825
total_bbox = 213836, rewritten_bbox = 0.623843 %

```

Figura 3.11: Métricas durante el entrenamiento de la red neuronal con el dataset de OIDv4

Cada cierto número de iteraciones se calcula el **mAP** y otras métricas de interés. Siguiendo las recomendaciones de [86], las iteraciones mínimas por cada clase para realizar un entrenamiento es de 2.000. En el primer entrenamiento del dataset personalizado se han realizado 8.000 iteraciones para 4 clases. En el caso de que se hubiera fijado unas máximas iteraciones superiores a las mínimas recomendadas, un indicio para saber que se debe finalizar el entrenamiento es cuando la media de las pérdidas (*average loss*) no disminuye en las siguientes iteraciones. También, se debe de tener en cuenta que, en datasets con una dificultad alta, como es este caso, la media de pérdidas final suele ser de *avg loss* = 3.0.

Al finalizar el entrenamiento se han evaluado las métricas de calidad. Gracias al framework Darknet [90] es fácil poder evaluarlas aplicando el siguiente comando en el terminal:

```

1 # Evaluacion de metricas de interes
2 ./darknet detector train data/obj.data cfg/yolov4-obj.cfg /mydrive/yolov4/backup/yolov4-
   obj_last.weights -dont_show

```

Código 3.19: Evaluación métricas de calidad del dataset utilizado para el entrenamiento de la red neuronal de detección de objetos

Durante el entrenamiento se ha generado un diagrama que se encuentra en la carpeta raíz del repositorio con nombre `chart.png`, donde se puede observar la evolución de las pérdidas por el error en función de las iteraciones. En la figura 3.12 se visualiza en color azul la evolución de las pérdidas y en rojo la evolución del mAP.

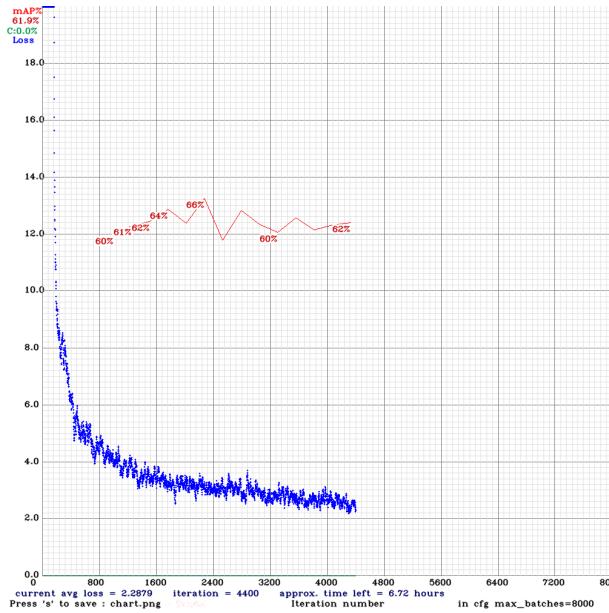


Figura 3.12: Evolución del mAP y pérdidas a lo largo de las interacciones durante el entrenamiento de la red neuronal con el dataset de OIDv4

Cabe resaltar que, cada 1.000 iteraciones se genera un fichero `.weights` con los pesos del modelo que estamos entrenando en **YOLOv4** y se almacenan en la carpeta `./backup`. Es importante tener en cuenta que se ha establecido las máximas iteraciones del entrenamiento a 8.000, sin embargo, es posible que los mejores resultados en las métricas hayan sido en la iteración 5.000. Esto puede ser producido por el *overfitting*. Afortunadamente, durante el entrenamiento se genera un fichero llamado `yolov4-obj_best.weights` donde se almacenan los pesos de la red entrenada con mejores resultados. Por tanto, se ha podido ahorrar tiempo en analizar las métricas cada 1.000 iteraciones para determinar en qué iteración el error ha sido mínimo.

En la sección 4.3.1 se observarán y analizarán los resultados tanto de este entrenamiento como de un segundo que ha sido necesario, ya que no se obtuvieron métricas de calidad superiores a las del dataset **MS COCO**.

3.6. Seguimiento de personas y objetos con YOLOv4 y Deep SORT

En esta sección se expone el esqueleto sobre el se ha desarrollado el algoritmo de detección de objetos abandonados. Para su aplicación, se ha utilizado el algoritmo de seguimiento **Deep SORT** trabajando conjuntamente con el detector de objetos **YOLOv4**. La implementación que se va a exponer a continuación está totalmente inspirada en [91].

Se ha clonado el repositorio de GitHub de seguimiento de objetos y personas con **Deep SORT** y **YOLOv4** en Tensorflow [92], donde previamente se ha realizado un *fork* de [91]. Se ha empleado el entorno virtual Anaconda `yolov4-gpu` utilizado en el código 3.4, ya que se van a utilizar las mismas versiones de librerías que en la implementación del detector de objetos con **YOLOv4** en Tensorflow [89]. Se han descargado los pesos de **YOLOv4** previamente entrenados sobre el dataset **MS COCO** para ser convertidos a modelos Tensorflow. Una vez descargado el repositorio [92], se han creado varias ramas Git para el desarrollo del proyecto. En esta parte del proyecto se va a trabajar sobre la rama `develop` para la implementación del algoritmo de seguimiento de objetos y personas con **Deep SORT** y **YOLOv4**.

Del mismo modo que en la sección 3.2, se ha utilizado en las últimas evaluaciones del algoritmo la GPU NVIDIA Tesla T4 del servicio cloud de Google Colab, ya que se trata de la GPU con mejores prestaciones que se puede utilizar de forma gratuita en el servicio de Google. Proporciona la mayor velocidad de FPS en las evaluaciones de los algoritmos.

```

1 # Descarga del repositorio de GitHub
2 git clone https://github.com/jmudy/yolov4-deepsort
3
4 # Entrar dentro de la carpeta del repositorio
5 cd yolov4-deepsort
6
7 # Cambiar a la rama de git develop y comprobar que nos encontramos en ella
8 git checkout develop
9 git branch -a
10
11 # Descarga de los pesos de YOLOv4
12 wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.
    weights -P ./data/

```

Código 3.20: Evaluación del seguimiento de objetos Deep SORT y YOLOv4 en Tensorflow (1)

En el código 3.21 se muestra el comando para convertir el modelo pre-entrenado de YOLOv4 en Darknet a Tensorflow. Se ha especificado un tamaño de 608x608, por dos motivos. El primero, es el mismo tamaño de redimensionamiento de las imágenes de entrada de la capa de la red Darknet, con lo cual, se puede comparar ambos modelos en las mismas condiciones en la etapa de detección. El segundo, se han obtenido mejores resultados que empleando el clásico tamaño de 416x416.

```

1 # Convertir pesos de YOLOv4 Darknet a Tensorflow
2 python save_model.py --weights ./data/yolov4.weights --output ./checkpoints/yolov4-608 --
   input_size 608 --model yolov4

```

Código 3.21: Evaluación del seguimiento de objetos Deep SORT y YOLOv4 en Tensorflow (2)

En el código 3.22 se puede observar el comando que se ha utilizado para la ejecución del algoritmo de seguimiento de Deep SORT con YOLOv4 en Tensorflow. Como se puede apreciar en las primeras líneas del script object_tracker.py [92], se disponen de una serie de flags donde se pueden indicar una gran variedad de parámetros, del mismo modo que en [88]. En este caso, se ha indicado que se quiere utilizar un modelo de YOLOv4 convertido a Tensorflow con tamaño de redimensionamiento 608x608 (si no se indica, el código por defecto busca un modelo de tamaño 416x416). Se ha especificado que se quiere emplear en la etapa de detección YOLOv4, ya que este repositorio también permite ejecutar la detección de objetos sobre YOLOv3. También se han incluido los flags --count para obtener por terminal el número de objetos que se están rastreando y --info para obtener por terminal las variables track.track_id que indica la identidad del objeto rastreado en cada fotograma, class_name que indica el nombre de la clase y {bbox[0], bbox[1], bbox[2], bbox[3]} para conocer los valores de {xmin, ymin, xmax, ymax} de los cuadros delimitadores.

```

1 # Ejecutar algoritmo de seguimiento de objetos con Deep SORT y YOLOv4 en Tensorflow
2 python object_tracker.py --video {input_file_name.mp4} --output {output_file_name.avi} --
   model yolov4 --size 608 --weights ./checkpoints/yolov4-608 --count --info

```

Código 3.22: Evaluación del seguimiento de objetos Deep SORT y YOLOv4 en Tensorflow (3)

En la figura 3.13 se muestra un fotograma del seguimiento de objetos y personas en una secuencia extraída de [24], un dataset que no se va a contemplar en este proyecto, pero que se utiliza con mucha frecuencia en la evaluación de algoritmos de detección y seguimiento de personas y objetos. Como se puede apreciar, cada persona y objeto detectado tiene una identidad única asignada durante el rastreo. Como se pudo ver en la figura 2.24, en los MOT, hay una primera etapa correspondiente a la detección (representado con el cuadro delimitador en color blanco) y una segunda etapa correspondiente al seguimiento (representado con los cuadros delimitadores de color azul y naranja). En esta propuesta únicamente se va a visualizar el cuadro delimitador que corresponde al seguimiento, asignando un color a cada elemento detectado. Cabe destacar que, en los primeros fotogramas de la secuencia se identificó como una motocicleta la bicicleta que acompaña a la persona con ID person-21. Durante los fotogramas siguientes se mantiene ese seguimiento erróneo. Se trata de un error poco frecuente que sucede hasta que se pierde el rastreo del objeto y se vuelva a asignar una nueva identidad después de la detección.



Figura 3.13: Ejemplo de MOT con Deep SORT en [24]

En el siguiente código 3.23, se muestra la información que se arroja por terminal al ejecutar el script `object_tracker.py` con los flags `--count` y `--info`. Por cada fotograma del vídeo procesado se muestra la cantidad de objetos que se están rastreando, el ID de cada persona u objeto detectado y el valor de las variables que componen los cuadros delimitadores. También, se indica la velocidad a la que se ha procesado el vídeo sobre una GPU NVIDIA Tesla T4. El acceso a las variables de ID y parámetros de los cuadros delimitadores en cada fotograma facilitarán el desarrollo del algoritmo de detección de objetos abandonados en la sección 3.7.

```

1 Frame number: 53
2 Objects being tracked: 17
3 Tracker ID: 1, Class: person, BBox Coords (xmin,ymin,xmax,ymax): (1483,0,1563,144)
4 Tracker ID: 2, Class: person, BBox Coords (xmin,ymin,xmax,ymax): (920,187,973,360)
5 Tracker ID: 3, Class: person, BBox Coords (xmin,ymin,xmax,ymax): (1643,780,1760,1030)
6 Tracker ID: 4, Class: person, BBox Coords (xmin,ymin,xmax,ymax): (843,171,910,343)
7 Tracker ID: 6, Class: person, BBox Coords (xmin,ymin,xmax,ymax): (968,52,1032,201)
8 Tracker ID: 7, Class: person, BBox Coords (xmin,ymin,xmax,ymax): (1650,78,1713,233)
9 Tracker ID: 9, Class: person, BBox Coords (xmin,ymin,xmax,ymax): (704,496,822,734)
10 Tracker ID: 10, Class: person, BBox Coords (xmin,ymin,xmax,ymax): (43,388,127,588)
11 Tracker ID: 13, Class: person, BBox Coords (xmin,ymin,xmax,ymax): (1102,0,1152,106)
12 Tracker ID: 15, Class: bicycle, BBox Coords (xmin,ymin,xmax,ymax): (1532,58,1573,144)
13 Tracker ID: 16, Class: person, BBox Coords (xmin,ymin,xmax,ymax): (1339,0,1368,26)
14 Tracker ID: 17, Class: person, BBox Coords (xmin,ymin,xmax,ymax): (1043,0,1093,98)
15 Tracker ID: 20, Class: bicycle, BBox Coords (xmin,ymin,xmax,ymax): (724,603,857,773)
16 Tracker ID: 21, Class: person, BBox Coords (xmin,ymin,xmax,ymax): (776,814,948,1075)
17 Tracker ID: 22, Class: person, BBox Coords (xmin,ymin,xmax,ymax): (430,187,502,362)
18 Tracker ID: 24, Class: person, BBox Coords (xmin,ymin,xmax,ymax): (1714,711,1781,834)
19 Tracker ID: 32, Class: motorbike, BBox Coords (xmin,ymin,xmax,ymax): (760,928,859,1080)
20 FPS: 10.06

```

Código 3.23: Prueba test del seguimiento de objetos Deep SORT y YOLOv4

Como se puede apreciar en la figura 3.13 y el código 3.23 se están rastreando clases que no son de interés para el objetivo de este TFM, en concreto, se identifica las 80 clases que componen el dataset MS COCO y que se pueden ver dentro del fichero `data/classes/coco.names`. Así pues, se va a filtrar las detecciones para que solo se muestre por pantalla el seguimiento de las clases: `person`, `backpack`, `handbag` y `suitcase`. Esta tarea resulta sencilla modificando unas pocas líneas de código.

Para filtrar las clases de interés solo ha sido necesario comentar la línea 136 y descomentar la línea 137 del script `object_tracker.py` [92] añadiendo las clases `person`, `backpack`, `handbag` y `suitcase`. Dentro de la lista `allowed_classes` simplemente se han añadido las clases que se desea rastrear. En el siguiente código 3.24 se muestran los cambios realizados en las líneas de código nombradas.

```

1     # by default allow all classes in .names file
2     #allowed_classes = list(class_names.values())
3     allowed_classes = ['person', 'handbag', 'backpack', 'suitcase']

```

Código 3.24: Clases permitidas en el seguimiento de objetos con Deep SORT

Como se ha podido observar en la figura 3.13 cada objeto o persona rastreada se representa con un cuadro delimitador de distinto color. Para evitar la dificultad de visualizar con claridad se ha cambiado a verde la clase `person` y a color azul las clases `backpack`, `handbag` y `suitcase`. Para ello, se han modificado las líneas 190-201 tal y como se puede ver en el código 3.25.

```

1      # draw bbox on screen only person, handbag, backpack and suitcase
2      color1 = (50, 203, 115)
3      color2 = (50, 138, 203)
4
5      if class_name == 'person':
6          cv2.rectangle(frame, (int(bbox[0]), int(bbox[1])), (int(bbox[2]), int(bbox[3])), color1, 1)
7          cv2.rectangle(frame, (int(bbox[0]), int(bbox[1]-20)), (int(bbox[0])+len(class_name)+len(str(track.track_id))*12, int(bbox[1])), color1, -1)
8          cv2.putText(frame, class_name + "-" + str(track.track_id),(int(bbox[0]), int(bbox[1]-5)), cv2.FONT_HERSHEY_SIMPLEX, 0.50, (255,255,255), 1, cv2.LINE_AA)
9
10     elif class_name == 'handbag' or 'backpack' or 'suitcase':
11         cv2.rectangle(frame, (int(bbox[0]), int(bbox[1])), (int(bbox[2]), int(bbox[3])), color2, 1)
12         cv2.rectangle(frame, (int(bbox[0]), int(bbox[1]-20)), (int(bbox[0])+len(class_name)+len(str(track.track_id))*12, int(bbox[1])), color2, -1)
13         cv2.putText(frame, class_name + "-" + str(track.track_id),(int(bbox[0]), int(bbox[1]-5)), cv2.FONT_HERSHEY_SIMPLEX, 0.50, (255,255,255), 1, cv2.LINE_AA)

```

Código 3.25: Colores y cuadros delimitadores de las clases en el seguimiento

En la siguiente figura 3.14 se refleja el resultado de filtrar las clases y utilizar únicamente dos colores para los cuadros delimitadores para cada una de las clases de objetos que se desea rastrear. De esta manera resulta más sencillo trabajar con los objetos de interés de cara al desarrollo del algoritmo de detección de objetos abandonados en la siguiente sección 3.7.



Figura 3.14: Ejemplo de MOT con Deep SORT filtrando clases de interés en [4]

Las últimas evaluaciones del algoritmo de seguimiento se han realizado con Google Colab. En el siguiente [link](#) se puede ejecutar el Notebook donde se realizan los mismos pasos descritos anteriormente. Todos los resultados obtenidos durante la evaluación del seguimiento de objetos y personas con **Deep SORT** y **YOLOv4** en Tensorflow se visualizarán y analizarán en la sección 4.3.4.

3.7. Algoritmo de detección de objetos abandonados

En esta sección se explica en detalle la propuesta del algoritmo para la detección de objetos abandonados a partir del algoritmo de detección **YOLOv4** junto con el algoritmo de seguimiento **Deep SORT**.

Durante los primeros 5 segundos de ejecución del script el algoritmo calcula la distancia entre las personas y los objetos de interés mediante la función **asociar**. Si el vídeo de entrada tiene un frame rate de 30, el script estará calculando la distancia entre los centroides de los cuadros delimitadores de todos los objetos durante las primeras 150 iteraciones del bucle while del programa principal.

La función **asociar** devuelve tres valores. Los dos primeros son los números identificadores de los objetos y personas que se han asociado mediante el algoritmo de seguimiento, y el último valor es el de la distancia. Dado que en un fotograma cualquiera se pueden encontrar numerosas personas y objetos de interés, el script actualiza la asociación persona-objeto comparando la distancia medida con la calculada en el fotograma anterior y quedándose con la más pequeña, de tal manera que se garantiza que dicho objeto es propiedad de la persona a la que se le ha asignado.

El siguiente código 3.26 muestra la estructura del diccionario **centroid_dict**. Está formado por diferentes elementos de los cuales el primero es el número identificador del objeto o persona detectado y los dos siguientes son las coordenadas *x, y* de los centroides de los cuadros delimitadores.

```
1 centroid_dict = idx, (int(x), int(y), xmin, ymin, xmax, ymax)
```

Código 3.26: Diccionario centroides personas y objetos de la función asociar

La función `is_close` calcula la distancia euclídea en píxeles de los centroides de los objetos detectados. Posteriormente, mediante un bucle `for` y la función `combinations` de la librería `itertools` se calcula la distancia entre todos los objetos y personas detectables en un determinado fotograma.

```
1 def is_close(p1, p2):
2     dst = math.sqrt(p1**2 + p2**2)
3     return dst
4
5 for (id1, p1), (id2, p2) in combinations(centroid_dict.items(), 2):
6     dx, dy = p1[0] - p2[0], p1[1] - p2[1]
7     distance = is_close(dx, dy)
```

Código 3.27: Cálculo distancia entre persona y objetos

En el caso de que el identificador del primer elemento corresponda a la clase persona y el segundo a la clase de un objeto de interés se guardarán los datos en una matriz con nombre `distances`. Esta matriz está formada por 3 columnas, donde se alojan el número de identidad de la persona y objeto que sí que cumplen con la condición de asociación y la distancia, y tantas filas como personas se detecten durante el paso de los fotogramas. Como se acaba de comentar anteriormente, al guardar los datos en esta matriz se tendrá en cuenta que la distancia entre la persona y el objeto sea la mínima calculada durante los primeros 5 segundos de vídeo.

Tal y como se puede contemplar en la figura 3.15, en el caso de que un objeto de interés no se encontrase durante los primeros 5 segundos de la secuencia de vídeo a una distancia máxima de 2 veces el ancho del cuadro delimitador de una persona, se considera que el objeto no tiene propietario, y por tanto no se encontrará dentro de la matriz `distances`. En ese caso aparecerá un mensaje de objeto abandonado cuando transcurran 15 segundos estando el objeto estático dentro del plano.

En el caso de que sí que exista asociación, es decir, que un objeto se encuentre a una distancia máxima en píxeles del doble del ancho del cuadro delimitador de una persona, y además, sea la distancia mínima de entre todos los objetos que se ha calculado su distancia hasta el sujeto, el algoritmo continuará leyendo el tercer valor de la columna del matriz `distances` para comprobar si la persona que es portadora del objeto que se le ha sido asignado abandona o no el objeto en los siguientes fotogramas.

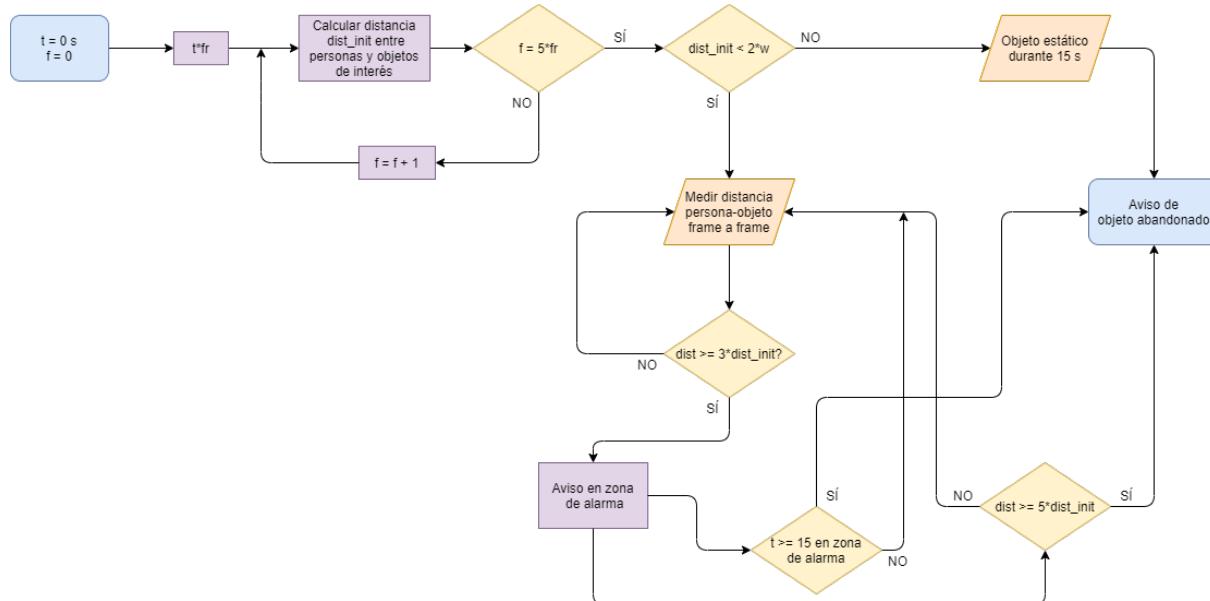


Figura 3.15: Esquema hipótesis detección objeto abandonado

En la siguiente figura 3.16 se muestra como la persona con número de identidad 5 se le ha asignado como objeto de su propiedad la maleta con número de identidad 2. Al crearse la asociación persona-objeto se dibuja automáticamente una línea que une ambos centroides indicando que la asociación se ha realizado correctamente.

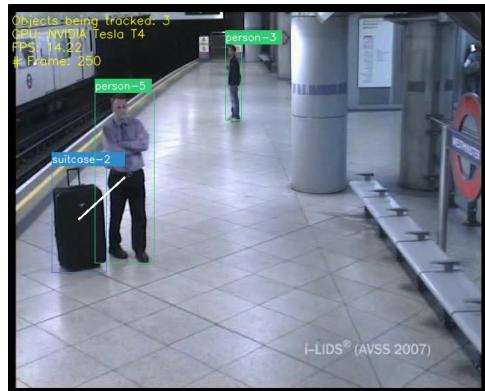


Figura 3.16: Asociación persona-objeto [6]

Si la distancia entre la persona y objeto es mayor o igual a 3 veces la distancia mínima que se calculó en el momento de la asociación se indicará que el objeto se encuentra en riesgo de abandono, indicando mediante un color amarillo los cuadros delimitadores de la persona y objeto así como una línea que une ambos centroides. Como se puede observar en el diagrama de funcionamiento del algoritmo en la figura 3.15, en el caso de que este suceso no ocurra el algoritmo continuará evaluando la distancia entre personas y objetos asociados hasta que ocurra algún incumplimiento.



Figura 3.17: Aviso de alerta posible objeto abandonado [6]

En el caso de que el objeto se encuentre en zona de alarma, pueden ocurrir dos sucesos. El primero es que el objeto se encuentre de manera estática durante los siguientes 15 segundos dentro de esa zona de alarma (persona y objeto se encuentran a más de 3 veces la distancia de asociación inicial). En ese caso se considerará que el objeto ha sido abandonado. El segundo suceso es que la persona siga alejándose del objeto del que es propietario a una distancia igual o mayor a 5 veces la distancia inicial de asociación o desaparezca del plano. En esa situación el objeto se considerará abandonado.

Como se puede observar en la figura 3.18 siguiente, la persona portadora de la maleta se aleja a una distancia en píxeles mayor a 5 veces la distancia de asociación. En ese caso se indicará que el objeto ha sido abandonado con un color rojo en los cuadros delimitadores de la persona y el objeto abandonado, de manera que se podrá identificar la identidad de la persona que abandonó el objeto. De la misma forma que en el caso de cuando el objeto se encontraba en zona de alarma, se dibujará una línea que une el centroide entre la persona y el objeto para visualizar con mayor facilidad la persona y objeto asociado que está provocando un abandono.

Del mismo modo que en la sección 3.2, se ha utilizado en las últimas evaluaciones del algoritmo la GPU NVIDIA Tesla T4 del servicio cloud de Google Colab, ya que se trata de la GPU con mejores prestaciones que se puede utilizar de forma gratuita en el servicio de Google. Proporciona la mayor velocidad de FPS en las evaluaciones de los algoritmos.



Figura 3.18: Detección de objeto abandonado [6]

Para ejecutar el algoritmo de detección de objetos abandonados se ha accedido al mismo repositorio `yolov4-deepsort` que se utilizó para evaluar el funcionamiento del seguimiento de objetos con **YOLOv4** y **Deep SORT** en la sección 3.6. En esta ocasión se ha cambiado de rama de Git para poder acceder a los ficheros necesarios. Dado que se trata de otra rama no se habrán guardado los cambios realizados en la rama `develop`, por tanto, ha sido necesario descargar de nuevo los pesos de **YOLOv4** Darknet.

```

1 # Entrar dentro de la carpeta del repositorio
2 cd yolov4-deepsort
3
4 # Cambiar a la rama de git abandoned y comprobar que nos encontramos en ella
5 git checkout abandoned
6 git branch -a
7
8 # Descarga de los pesos de YOLOv4
9 wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.
    weights -P ./data/

```

Código 3.28: Evaluación de la detección de objetos abandonados con DeepSORT y YOLOv4 en Tensorflow (1)

Del mismo modo que en la sección 3.6, se ha convertido el modelo Darknet a Tensorflow con un tamaño de redimensionamiento de 608x608 ejecutando el comando que se muestra en el código 3.29, dado que han obtenido mejores resultados que con un tamaño de 416x416.

```

1 # Convertir pesos de YOLOv4 Darknet a Tensorflow
2 python save_model.py --weights ./data/yolov4.weights --output ./checkpoints/yolov4-608 --
    input_size 608 --model yolov4

```

Código 3.29: Evaluación de la detección de objetos abandonados con DeepSORT y YOLOv4 en Tensorflow (2)

En el código 3.30 se puede observar el comando que se ha utilizado para la ejecución del algoritmo de detección de objetos abandonados con **Deep SORT** y **YOLOv4** en Tensorflow. De igual manera al código 3.22 se ha utilizado los flags `--info` y `--count` para obtener los valores {xmin, ymin, xmax, ymax} de los cuadros delimitadores así como el número de objetos y personas que se está detectando en pantalla.

```

1 # Ejecutar algoritmo de detección objetos abandonados con Deep SORT y YOLOv4 en Tensorflow
2 python abandoned_object.py --video {input_file_name.mp4} --output {output_file_name.avi} --
    model yolov4 --size 608 --weights ./checkpoints/yolov4-608 --count --info

```

Código 3.30: Evaluación de la detección de objetos abandonados con DeepSORT y YOLOv4 en Tensorflow (3)

Las últimas evaluaciones del algoritmo de seguimiento se han realizado con Google Colab. En el siguiente [link](#) se puede ejecutar el Notebook donde se realizan los mismos pasos descritos anteriormente. Todos los resultados obtenidos durante la evaluación del seguimiento de objetos y personas con **Deep SORT** y **YOLOv4** en Tensorflow se visualizarán y analizarán en la sección 4.3.5.

3.8. Conclusiones

Este capítulo ha tenido como objetivo el desarrollo de un algoritmo de detección de objetos abandonados utilizando **YOLOv4** como algoritmo de detección y **Deep SORT** como algoritmo de seguimiento de objetos y personas. **YOLOv4** viene previamente entrenado sobre el dataset **MS COCO**. Dado que se quiere utilizar un sistema donde no se necesita detectar las 80 clases de **MS COCO** y solo se precisa la detección de personas y los objetos de interés: mochilas, maletas y bolsas de mano, se ha entrenado dos modelos de **YOLOv4** a partir de un dataset personalizado **OIDv4** con la finalidad de obtener mayores valores en las métricas de calidad sobre dichas clases. Variando los parámetros de la **CNN** así como el número de clases y de imágenes de entrenamiento y evaluación no se han conseguido mejorar las métricas que se obtienen con **MS COCO**. Por tanto, se ha decidido continuar el desarrollo del proyecto con el modelo pre-entrenado de **YOLOv4**.

En primer lugar se ha probado **YOLOv4** sobre el framework Darknet original. Para facilitar la programación de los scripts en la detección, seguimiento y detección de objetos abandonados con Python, se ha decidido convertir el modelo **YOLOv4** a Tensorflow, dado que Darknet no es de uso extendido y puede resultar más difícil solucionar los posibles errores que puedan aparecer.

Las implementaciones tanto de **YOLOv4** como de **Deep SORT** han estado totalmente inspiradas en las aplicaciones desarrolladas en los repositorios de GitHub [88] y [91]. Se han realizado las modificaciones oportunas para satisfacer las necesidades de cara a la etapa de desarrollo del algoritmo de detección de objetos abandonados, como es la filtración en la etapa de detección para que solo tenga en cuenta las clases de interés que se quieren detectar e ignore el resto de las clases de **MS COCO**.

Con la reciente llegada de **YOLOv4** se ha podido aumentar la precisión en las detecciones que se realizaron con versiones previas de **YOLO** en [93]. Además, la implementación de **YOLOv4** junto con **Deep SORT**, que incluye tanto filtros de Kalman como el algoritmo húngaro, ha provocado un mejor rastreo de los objetos y personas respecto a las propuestas que se realizaron en [93] utilizando únicamente filtros de Kalman.

Una vez establecido como base del proyecto la utilización de **YOLOv4** como algoritmo de detección y **Deep SORT** como algoritmo de seguimiento de objetos y personas se ha diseñado un algoritmo de detección de objetos abandonados para aplicaciones de videovigilancia en tiempo real basado en [93].

Para ello, se han tenido en cuenta dos posibles escenarios. El primero trata cuando un objeto está alejado de cualquier persona y además se encuentra de manera estática durante más de 15 segundos. En ese caso se establece que el objeto ha sido previamente abandonado y además no se puede identificar un propietario del mismo. En el segundo escenario se asocian los objetos a las personas más cercanas que se encuentran durante los primeros segundos de la secuencia de vídeo. Una vez establecida la asociación persona-objeto se estudia cuando la distancia que une ambos se aleja más de cierta distancia. Primeramente, se establece una distancia de zona de alarma donde el objeto es candidato a poder ser abandonado. Esta distancia es medida en píxeles entre los centroides de la persona y el objeto asociado. En el caso de que la persona se aleje más de la distancia máxima permitida o desaparezca del plano de la cámara, se lanzará un aviso por pantalla de que el objeto ha sido abandonado. Y en el caso de que el objeto se encuentre en zona de alarma durante más de 15 segundos también se considerará abandonado por su propietario.

Cuando el algoritmo de seguimiento con **Deep SORT** trabaja correctamente sin perder la identidad de las personas y objetos que se estaban detectando, el algoritmo de detección de objetos abandonados funciona de forma eficiente y sin fallos. Sin embargo, tal y como se verá en la sección 4.3.5, pueden aparecer fenómenos que produzcan la incapacidad de detectar un objeto abandonado.

Capítulo 4

Resultados

Cuando algo es lo suficientemente importante, lo haces incluso si las probabilidades no están a tu favor.

Elon Musk

4.1. Introducción

En este capítulo se recogen los resultados obtenidos en el diseño del sistema la detección de objetos abandonados. Para evaluar el funcionamiento de los distintos algoritmos que han sido empleados o diseñados es necesario realizar evaluaciones cuantitativas y cualitativas para validar su funcionamiento. En la detección de objetos, los investigadores evalúan sus algoritmos sobre los mismos datasets, de tal manera que se pueda contrastar los resultados con las propuestas de trabajos previos. Para evaluar un modelo de una red neuronal artificial se utilizan datasets de referencia para medir métricas de calidad para cuantificar el funcionamiento de los algoritmos.

Este capítulo está dividido en dos secciones. En la primera, se van a detallar las métricas de calidad que se han empleado para validar los datasets de referencia empleados en el entrenamiento de la red neuronal **YOLOv4** y se expondrán los datasets empleados para evaluar los algoritmos que se han propuesto en el capítulo 3. En la segunda, se expondrán y interpretarán los resultados cuantitativos obtenidos en los entrenamientos en base a las métricas de calidad, así como los resultados cualitativos en la ejecución de los algoritmos sobre los datasets descritos anteriormente.

4.2. Entorno experimental

En el capítulo anterior se manifestó que **YOLOv4** está pre-entrenado sobre **MS COCO**. Sin embargo, durante las primeras pruebas de funcionamiento del detector de objetos se pudo observar que en la detección sobre los objetos de interés arrojaba en algunas ocasiones un *confidence score* bajo dependiendo de la posición y ángulo que se encontraran. **YOLOv4** viene configurado por defecto con un *threshold* sobre el *confidence score* del 25 %, parámetro que se deberá de tener muy en cuenta más adelante. Por este motivo, se ha decidido re-entrenar la red sobre el dataset **OIDv4** con el objetivo de mejorar las métricas. Por ello, en el siguiente subapartado se va a hacer una introducción sobre las métricas más utilizadas en la evaluación de algoritmos de detección de objetos. Más adelante, en la sección 4.3.1 se visualizarán los resultados obtenidos tras el entrenamiento de la red.

4.2.1. Métricas de calidad

En esta sección se va a exponer las métricas de calidad [25] que han sido utilizadas para evaluar los datasets utilizados en el entrenamiento de **YOLOv4**.

4.2.1.1. Intersección sobre la unión (IoU)

IoU es una medida basada en el índice Jaccard que evalúa la superposición entre dos cuadros delimitadores. Requiere un cuadro delimitador de ground truth B_{gt} y un cuadro delimitador de predicción B_p . Aplicando el **IoU** podemos saber si una detección es válida (verdadero positivo) o no (falso positivo).

El **IoU** viene dado por el área de superposición entre el cuadro delimitador de predicción y el cuadro delimitador de ground truth dividido por el área de unión entre ellos:

$$\text{IoU} = \frac{\text{area}(B_p \cap B_{gt})}{\text{area}(B_p \cup B_{gt})} \quad (4.1)$$

La siguiente figura ilustra el **IoU** entre un cuadro delimitador de ground truth (en verde) y un cuadro delimitador detectado (en rojo).

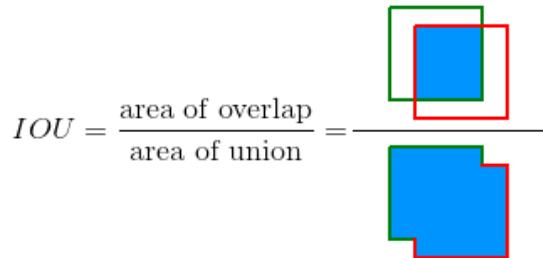


Figura 4.1: Área de superposición IoU entre los cuadros delimitadores [25]

4.2.1.2. TP, TN, FP y FN

Otros parámetros básicos en las métricas de calidad que conforman la matriz de confusión [26] son:

- **True Positive (TP)**: Número de predicciones donde el clasificador predice correctamente la clase positiva como positiva. $\text{IoU} \geq threshold$
- **True Negative (TN)**: Número de predicciones donde el clasificador predice correctamente la clase negativa como negativa. No se utiliza en el cálculo de métricas.
- **False Positive (FP)**: Número de predicciones donde el clasificador predice incorrectamente la clase negativa como positiva. $\text{IoU} < threshold$
- **False Negative (FN)**: Número de predicciones donde el clasificador predice incorrectamente la clase positiva como negativa.

Típicamente el *threshold* toma valores del 50 %, 75 % o 95 %.

		True Class	
		Positive	Negative
Predicted Class	Positive	TP	FP
	Negative	FN	TN

Figura 4.2: Matriz de confusión [26]

4.2.1.3. Precisión

La precisión es la capacidad de un modelo para identificar solo los objetos relevantes. Es el porcentaje de predicciones positivas correctas y viene dado por la siguiente expresión 4.2:

$$P = \frac{TP}{TP + FP} = \frac{TP}{\text{all detections}} \quad (4.2)$$

4.2.1.4. Recall

El Recall es la capacidad de un modelo para encontrar todos los casos relevantes (todos los cuadros delimitadores de ground truth). Es el porcentaje de verdadero positivo detectado entre todos los ground truths relevantes y viene dado por la siguiente expresión 4.3:

$$R = \frac{TP}{TP + FN} = \frac{TP}{\text{all ground truths}} \quad (4.3)$$

4.2.1.5. F-Score

El F-Score se trata de una medida estadística de precisión muy utilizada en las pruebas test de algoritmos. Es la media armónica que combina los valores de la precisión y el Recall. Viene dado por la expresión 4.4:

$$F = \frac{2 \cdot P \cdot R}{P + R} \quad (4.4)$$

4.2.1.6. Precisión media

La precisión media es el valor medio de 11 puntos en la curva P-R para cada posible umbral (cada probabilidad de detección) para la misma clase (Precisión-Recall). En la ecuación 4.5 se muestra el cálculo de la precisión media:

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, \dots, 1\}} \rho_{\text{interp}(r)} \quad (4.5)$$

con

$$\rho_{\text{interp}} = \max_{\tilde{r}: \tilde{r} \geq r} \rho(\tilde{r})$$

donde $\rho(\tilde{r})$ es la precisión medida en el Recall \tilde{r}

Por otro lado, el mAP es la media de los Average precision (AP) de todas las categorías de objetos. El mAP se representa mediante la siguiente ecuación:

$$mAP = \frac{1}{N} \sum_{i=1}^N AP_i \quad (4.6)$$

4.2.2. Datasets utilizados

A continuación, se van a describir los datasets más relevantes en la detección de objetos abandonados en los sistemas de videovigilancia así como los datasets de referencia sobre las que se validará previamente, en base a las métricas de calidad, el modelo entrenado o pre-entrenado utilizado.

En la tabla 4.1 se resumen los contenidos más relevantes de los datasets como son: número de secuencias, longitud media en minutos, escenario o tipos de challenges.

Los challenges que se consideran de interés para la detección de objetos abandonados se numeran a continuación:

- I = cambios de iluminación/sombras.
- R = objetos alejados o pequeños.
- P = personas estáticas en un punto durante un período de tiempo.
- O = occlusiones.
- LR = resolución vídeo baja.
- RO = objetos abandonados o eliminados.

Tabla 4.1: Datasets utilizados en la evaluación de los algoritmos

Nombre del dataset	# Secuencias	Longitud media (min)	Escenario	Challenge
PETS2007	28	1,96	Aeropuerto	I, R, P, O
AVSSAB2007	3	3,46	Estación de metro	I, R, P, O
GBA2018	8	0,74	Interior	I, R, P, O, RO
ABODA	11	1,90	Interior/exterior	I, R, P, O

4.2.2.1. PETS2007 Dataset

El dataset [Performance Evaluation of Tracking and Surveillance 2007 \(PETS2007\)](#) [5] está formado por secuencias que contienen tres tipos de escenarios con una complejidad ascendente: personas merodeando, robo de equipaje y equipaje desatendido/abandonado.

Definición de merodear

Merodear se define como el acto donde una persona entra en escena y permanece dentro de la escena durante más de t segundos. Para los propósitos de [PETS2007](#), 60 segundos.

Definición de objeto desatendido

Se utilizan tres reglas para determinar si el equipaje está atendido por parte de una persona o no:

- Un equipaje es propiedad y es atendido por una persona o personas que ingresan al lugar con el equipaje hasta el punto en que el equipaje no está en contacto físico con la persona (regla contextual).
- En este punto, el equipaje es atendido por el propietario únicamente cuando se encuentran a una distancia de un metro del equipaje (regla espacial). Todas las distancias se miden entre los centroides del objeto en el plano del suelo (es decir, $z = 0$). Si una persona se encuentra a 2 metros de su equipaje, el sistema no debe activar ninguna alarma.
- Un equipaje está desatendido cuando el propietario está a más de 3 metros del equipaje. Si una persona cruza la línea de los 3 metros, el sistema debe usar la regla espacio-temporal el equipaje. Si el equipaje se encuentra en el rango de [2,3] metros se determina como una zona de advertencia.

Definición de objeto abandonado

El abandono de una pieza de equipaje se define espacial y temporalmente. El abandono se define como:

Un bulto de equipaje que ha sido desatendido por el propietario por un período de 25 segundos consecutivos en el cual el propietario no ha vuelto a atender el equipaje, ni el equipaje ha sido atendido por una segunda persona. Si una pieza de equipaje se deja desatendida durante 25 segundos, se activa una alarma.

Definición de robo de pieza de equipaje

El robo de una pieza de equipaje se define utilizando únicamente una restricción espacial. El robo se define como un artículo de equipaje movido a más de 3 metros del propietario. Se puede emitir una advertencia 2 metros del propietario.

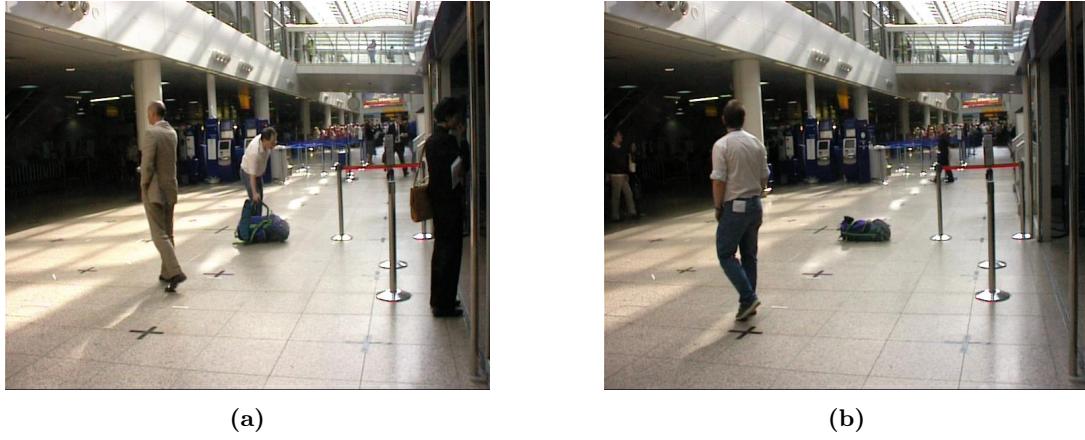


Figura 4.3: Imágenes extraídas del dataset PETS2007 [5]. (a) Fotograma de la secuencia S08-camera4 donde un hombre deja su equipaje en el suelo. (b) Otro fotograma de la secuencia S08-camera4 donde el hombre abandona el lugar sin su equipaje.

Las cámaras utilizadas para la grabación de las distintas secuencias son las siguientes:

- Cámara 1: Canon MV-1 1xCCD w/progressive scan.
- Cámara 2: Sony DCR-PC1000E 3xCMOS.
- Cámara 3: Canon MV-1 1xCCD w/progressive scan.
- Cámara 4: Sony DCR-PC1000E 3xCMOS.

La resolución de todas las secuencias es PAL standard (768 x 576 píxeles y 25 **FPS**) y comprimidas como secuencias de imágenes JPEG (aprox. 90 % de calidad).



Figura 4.4: Imágenes extraídas del dataset PETS2007 [5]. (a) Fotograma de la secuencia S07-thirdView donde una mujer se encuentra junto a su equipaje. (b) Otro fotograma de la secuencia S07-thirdView donde la mujer abandona el lugar sin su bolsa de mano.

En este proyecto solo se va a tener en cuenta cuando un equipaje ha sido abandonado sin propietario en un tiempo determinado o cuando ha sido desatendido por su propietario alejándose cierta distancia en relación a la distancia que se establece en el momento de la asociación persona-objeto.

4.2.2.2. AVSSAB2007 Dataset

Advanced Video and Signal based Surveillance Abandoned Baggage (AVSSAB2007) [6] es un subconjunto de datos del dataset AVSS 2007, el cual fue creado para el *i-LIDS bag and vehicle detection challenge* que se celebró en la 14th IEEE International Conference on Advanced Video and Signal based Surveillance en septiembre de 2007. En dicha conferencia se tenía como objetivo atraer artículos del Estado del Arte para presentar metodologías para la detección de eventos. Tiene la intención de informar sobre la precisión, solidez y complejidad de las secuencias de vídeo del dataset.

Se ha utilizado el subconjunto de secuencias dedicadas a la detección de objetos abandonados. Este subdataset está formado por tres secuencias de vídeo grabadas a una resolución de 720 x 576 píxeles a 25 **FPS**. El modelo de la cámara no se especifica.

La **ROI** está dividido en tres zonas: cercana, media y lejana. En cada una de las secuencias de vídeo el objeto que es abandonado se encuentra en cada una de las zonas que se muestran en la figura 4.5.

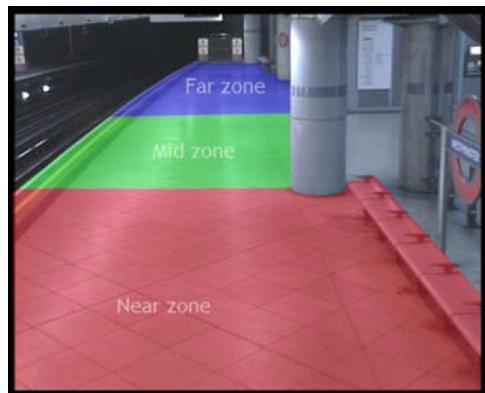


Figura 4.5: Regiones de interés del dataset AVSSAB2007 [6]

Todas las secuencias presentan la misma estructura en la sucesión de eventos:

- Una persona ha colocado un objeto que estaba en su posesión en una de las áreas de detección.
- Esa persona abandona el área de detección sin el objeto.
- Esa persona aún no ha regresado al objeto después de 60 segundos tras haber abandonado el área de detección.
- El objeto permanece en el área de detección hasta finalizar la secuencia de vídeo.

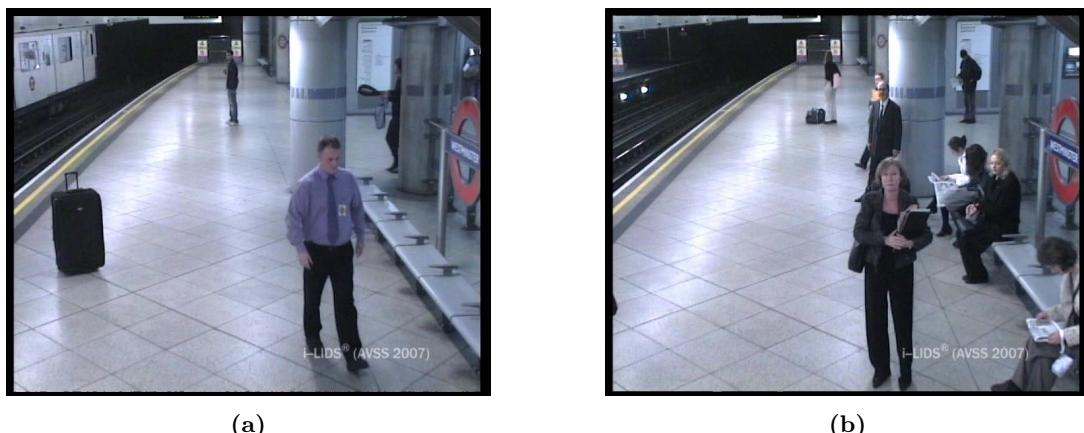


Figura 4.6: Imágenes extraídas del dataset AVSSAB2007 [6]. (a) Fotograma de la secuencia AVSSAB-Easy donde el hombre abandona su maleta en la zona cercana. (b) Fotograma de la secuencia AVSSAB-Medium donde una mujer se encuentra junto a su bolsa de mano en la zona media del andén del metro.

4.2.2.3. GBA2018 Dataset

El dataset [GEINTRA Behaviour Analysis 2018 \(GBA2018\)](#) [4] fue grabado y etiquetado por el grupo de investigación [Grupo de Ingeniería Electrónica Aplicada a Espacios Inteligentes y Transporte \(GEINTRA\)](#) en la [EPS](#) de la Universidad de Alcalá de Henares (UAH) durante la realización del [TFM](#) de David Valdivieso López [93]. Está orientado a la evaluación de algoritmos de detección de objetos abandonados y eventos anómalos como estampidas. El dataset está formado 8 secuencias en 2 escenarios distintos grabadas con una GoPro HERO4 a una resolución de 1920 x 1080 píxeles a 60 [FPS](#).



Figura 4.7: ROI del hall de la Escuela Politécnica Superior (UAH) [27]

En el primer escenario se muestra como [ROI](#) el hall de la [EPS](#) desde un plano alejado donde ocurren eventos como abandono de maletas, mochilas y bolsas de mano. Se trata de un escenario complejo ya que hay cambios tanto de luz natural como artificial y personas y objetos situados a distancias largas, lo que puede dificultar la tarea de detección.



Figura 4.8: Imágenes extraídas de secuencias del primer escenario del dataset GBA2018 [4]. (a) Fotograma de la secuencia GBA-far-video2 donde dos bolsas de mano han sido abandonadas en medio del hall. (b) Fotograma de la secuencia GBA-far-video3 donde varias bolsas y maletas están alejadas de sus propietarios.

La [ROI](#) del segundo escenario se encuentra en el pasillo que conecta el hall con la cafetería. El plano de grabación es más cercano respecto al anterior por lo que se consideran secuencias más fáciles de evaluar ya que no se encuentran elementos de interés alejados ni tampoco hay cambios bruscos en la iluminación.



Figura 4.9: Imágenes extraídas de secuencias del segundo escenario del dataset GBA2018 [4]. (a) Fotograma de la secuencia GBA-near-big-video2 donde una bolsa de mano es abandonada en el pasillo de la cafetería. (b) Fotograma de la secuencia GBA-near-big-video4 donde dos personas abandonan una pequeña bolsa de mano.

4.2.2.4. ABODA Dataset

[Abandoned Objects Dataset \(ABODA\)](#) [7] es un dataset propuesto por primera vez en 2015 para la detección de objetos abandonados. [ABODA](#) está formado por 11 secuencias etiquetadas con varios escenarios de aplicaciones reales, que son un desafío para la detección de objetos abandonados. Las situaciones incluyen escenas de grandes aglomeraciones de personas, cambios en las condiciones de iluminación, detección nocturna, así como ambientes interiores y exteriores.

Algunas secuencias de vídeo están grabadas a resoluciones: de 720 x 480 píxeles a 30 [FPS](#), y otras secuencias a 640 x 480 píxeles y 30 [FPS](#). El modelo de la cámara no se especifica.

La figura 4.10 muestra los fotogramas de [video1](#) grabados en un entorno interior con luz artificial. En esta secuencia dos personas están interactuando entre sí y una de ellas tiene una mochila (Fig. 4.10a). Más tarde, una de las dos personas deja caer su mochila y ambos abandonan la escena ((Fig. 4.10b))

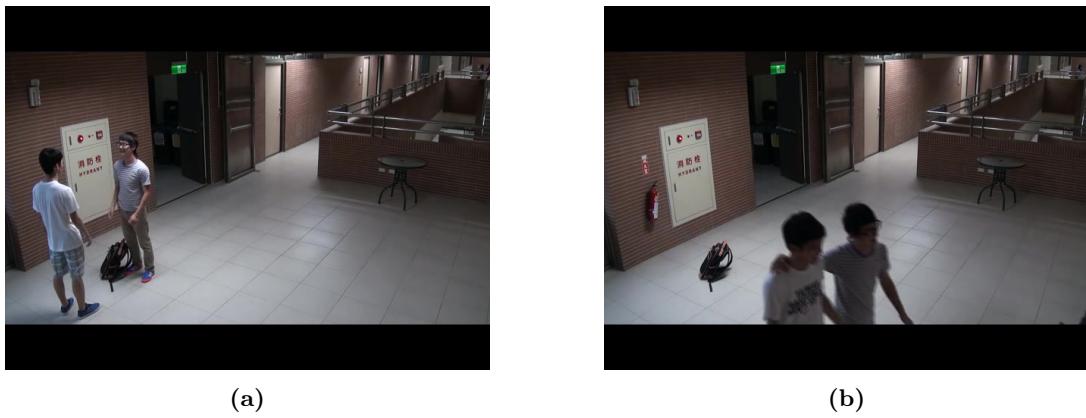


Figura 4.10: Imágenes extraídas del dataset ABODA [7]. (a) Fotograma donde dos chicos conversan en el hall. (b) Otro fotograma donde los dos chicos abandonan una mochila.

Este dataset también incluye escenas de aglomeraciones de personas que se encuentran en aeropuertos o estaciones de tren, iluminación variable de día y grabaciones nocturnas.



Figura 4.11: Imágenes extraídas del dataset ABODA [7]. (a) Fotograma de la secuencia video5 de una grabación nocturna. (b) Fotograma de la secuencia video11 donde hay varias personas haciendo cola en un aeropuerto.

4.3. Resultados experimentales

Elegido [MS COCO](#) como dataset de referencia para entrenar y evaluar [YOLOv4](#), en las siguientes secciones se van a plasmar los resultados obtenidos en los distintos algoritmos que han sido desarrollados en el capítulo 3.

Esta sección está dividida en 5 partes. Las 2 primeras partes corresponden a los resultados obtenidos en los entrenamientos de [YOLOv4](#) con el dataset [OIDv4](#) que se presenció en la sección 3.5. Se realizará

una evaluación cuantitativa para determinar si se toma el modelo entrenado como válido o no. Como se podrá ver a continuación, se realizaron dos entrenamientos. En el segundo entrenamiento se aumentaron el número de clases, imágenes de entrenamiento e imágenes de validación y se modificaron parámetros de la CNN respecto al primer entrenamiento. En las 3 últimas partes, se observarán los resultados obtenidos en los 3 algoritmos desarrollados en el capítulo 3, evaluando de manera cualitativa el funcionamiento de los mismos.

4.3.1. Métricas de calidad en Open Image Dataset v4

En las tablas 4.2, 4.3, 4.4 y 4.5 se reflejan las métricas más relevantes cada 1.000 iteraciones del primer entrenamiento de la CNN de YOLOv4.

Tabla 4.2: Métricas de calidad en el primer entrenamiento con OIDv4 [1]

Iterations	AP person (%)	AP handbag (%)	AP backpack (%)	AP suitcase (%)
1.000	31,26	85,45	67,99	42,15
2.000	43,96	92,39	63,88	64,79
3.000	36,16	90,10	67,88	53,33
4.000	35,56	91,99	64,61	57,74
5.000	34,21	87,35	68,73	48,77
6.000	36,74	89,48	65,83	49,09
7.000	34,76	88,94	68,20	51,24
8.000	38,30	87,69	72,00	58,89

Tabla 4.3: Métricas de calidad en el primer entrenamiento con OIDv4 [2]

Iterations	TP person	TP handbag	TP backpack	TP suitcase
1.000	249	52	19	16
2.000	338	54	19	19
3.000	324	54	21	18
4.000	329	54	20	20
5.000	313	50	23	16
6.000	323	52	20	13
7.000	288	53	19	18
8.000	308	49	21	19

Tabla 4.4: Métricas de calidad en el primer entrenamiento con OIDv4 [3]

Iterations	FP person	FP handbag	FP backpack	FP suitcase
1.000	416	20	11	42
2.000	489	15	16	7
3.000	479	22	18	36
4.000	521	10	20	19
5.000	474	14	19	22
6.000	446	7	14	13
7.000	391	19	13	16
8.000	412	11	16	17

Como se observa en tabla 4.2, el AP obtenido en las clases es más alto respecto a los de MS COCO (ver tabla 4.8), excepto el valor obtenido en la clase Person que es significativamente más bajo.

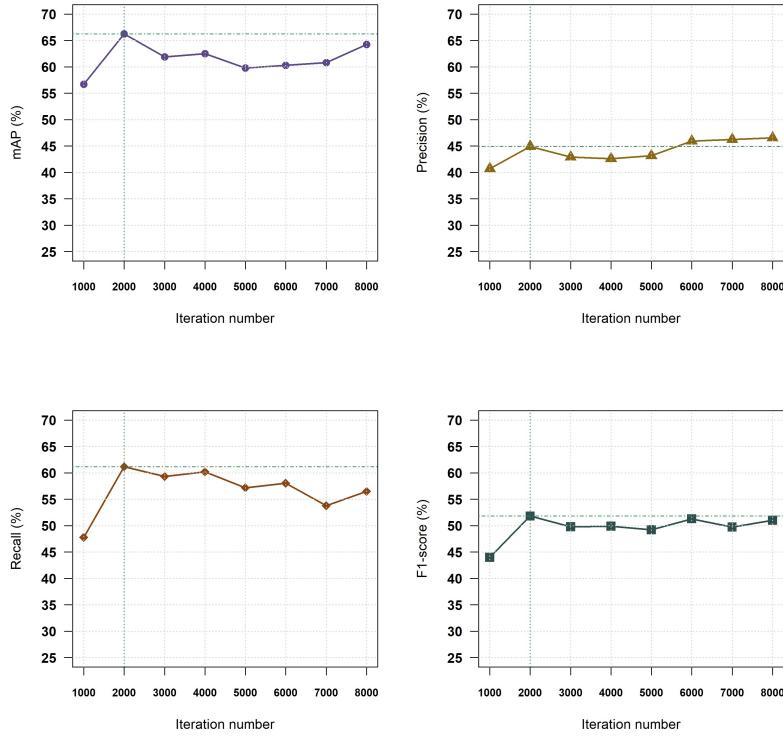
En las tablas 4.3 y 4.4 destaca la aparición de muchos más FP que TP en la clase Person, lo que ha conducido a tener más FP que TP globales en todas las iteraciones (ver tabla 4.5).

En la siguiente tabla 4.5 se aprecia que los mejores resultados obtenidos tuvieron lugar en la iteración 2.000, con un 44,93 % de precisión, un 61,17 % de Recall y un 51,81 % de F-score. A pesar de obtener un 66,25 % de mAP, el average IoU fue del 34,13 % debido a la alta aparición de FP durante el entrenamiento. Se trata de un valor muy bajo.

Tabla 4.5: Métricas de calidad en el primer entrenamiento con OIDv4 [4]

Iterations	TP	FP	FN	Precision (%)	Recall (%)	F-score (%)	Average IoU (%)	mAP @ 0.5 (%)
1.000	336	489	367	40,73	47,80	43,98	29,36	56,71
2.000	430	527	273	44,93	61,17	51,81	34,13	66,25
3.000	417	555	286	42,90	59,32	49,79	32,29	61,87
4.000	423	570	280	42,60	60,17	49,88	33,30	62,47
5.000	402	529	301	43,18	57,18	49,20	32,53	59,77
6.000	408	480	295	45,95	58,04	51,29	36,76	60,28
7.000	378	439	325	46,27	53,77	49,74	36,90	60,78
8.000	397	456	306	46,54	56,47	51,03	37,69	64,22

En la figura 4.12 se ilustra la evolución de las métricas en función de las iteraciones transcurridas. Estableciendo como métrica de prioridad el mAP, los mejores resultados se obtuvieron en la iteración 2.000. A partir de este punto oscilaron levemente con tendencia decreciente.

**Figura 4.12:** Resumen métricas primer entrenamiento de la red neuronal con el dataset de OIDv4

Como se ha podido observar, los resultados obtenidos no han sido favorables. A pesar de tener un mAP superior al de MS COCO, el average IoU es significativamente más bajo que el **56,04 %** de MS COCO (ver tabla 4.10).

Por ello, se ha realizado un nuevo entrenamiento modificando algunos parámetros de la CNN y aumentando el número de clases y de imágenes de entrenamiento y validación. El dataset **OIDv4** dispone de clases más específicas de equipajes o bolsas de mano que **MS COCO**.

Se ha realizado el mismo procedimiento que se hizo en la sección 3.5, añadiendo las clases de interés: ‘Plastic bag’, ‘Luggage and bags’ y ‘Briefcase’. Estas 3 nuevas clases posteriormente se han unificado junto a las anteriores como una única clase llamada ‘bags’. Se han aumentado el valor de las variables de redimensionamiento de imágenes a la entrada de la red `width` y `height` a 608 para garantizar una mayor precisión a costa de un mayor tiempo de entrenamiento. El número de imágenes de entrenamiento se ha subido a 3.000 y las de validación a 600 en la clase Person.

El número máximo de iteraciones (`max_batches`) se ha fijado a 20.000 iteraciones. A pesar de que lo recomendable es fijarlo a (# de clases) * 2.000, se ha aumentado para analizar en un mayor número de iteraciones la evolución de los valores de las métricas. El resto de parámetros de configuración de la red **YOLOv4** se han adaptado al nuevo número de clases a entrenar, del mismo modo que se realizó en la sección 3.5.2.

En las tablas 4.6 y 4.7 se reflejan las métricas más relevantes cada 1.000 iteraciones del segundo entrenamiento de la CNN de **YOLOv4**.

Tabla 4.6: Métricas de calidad en el segundo entrenamiento con OIDv4 [1]

Iterations	AP person (%)	AP bags (%)	TP person	TP bags	FP person	FP bags
1.000	30,81	21,61	5.216	231	9.220	579
2.000	38,38	53,59	6.542	362	11.789	354
3.000	33,51	68,56	7.232	419	18.887	411
4.000	41,31	77,12	7.105	427	11.397	222
5.000	38,86	75,78	6.586	444	11.735	398
6.000	36,29	66,49	6.556	426	12.506	537
7.000	39,94	67,78	6.246	418	9.744	523
8.000	31,69	69,07	6.082	417	13.353	422
9.000	43,34	78,37	6.773	451	9.846	373
10.000	43,40	78,53	6.174	426	7.149	217
11.000	38,81	76,60	7.166	446	12.162	387
12.000	41,72	78,10	6.926	444	10.387	289
13.000	39,48	74,67	6.575	406	9.850	238
14.000	41,85	73,69	6.844	432	10.092	385
15.000	40,19	75,37	6.915	423	11.426	252
16.000	41,26	75,17	6.661	423	9.330	219
17.000	42,13	78,77	6.991	433	9.924	242
18.000	40,97	75,45	6.871	432	10.243	300
19.000	39,01	73,23	6.822	428	10.791	333
20.000	41,37	78,38	7.011	432	10.103	232

Tabla 4.7: Métricas de calidad en el segundo entrenamiento con OIDv4 [2]

Iterations	TP	FP	FN	Precision (%)	Recall (%)	F-score (%)	Average IoU (%)	mAP @ 0.5 (%)
1.000	5.447	9.799	6.379	35,73	46,06	40,24	25,72	26,21
2.000	6.904	12.143	4.922	36,25	58,38	44,73	27,30	45,98
3.000	7.651	19.298	4.175	28,39	64,70	39,46	21,66	51,04
4.000	7.532	11.619	4.294	39,33	63,69	48,63	30,90	59,22
5.000	7.030	12.133	4.796	36,69	59,45	45,37	28,28	57,32
6.000	6.982	13.043	4.844	34,87	59,04	43,84	27,23	51,39
7.000	6.664	10.267	5.162	39,36	56,35	46,35	30,96	53,86
8.000	6.499	13.775	5.327	32,06	54,96	40,49	24,96	50,38
9.000	7.224	10.219	4.602	41,41	61,09	49,36	32,95	60,85
10.000	6.600	7.366	5.226	47,26	55,81	51,18	37,83	60,97
11.000	7.612	12.549	4.214	37,76	64,37	47,59	30,20	57,70
12.000	7.370	10.676	4.456	40,84	62,32	49,34	32,34	59,91
13.000	6.981	10.088	4.845	40,90	59,03	48,32	32,83	57,07
14.000	7.276	10.477	4.550	40,98	61,53	49,20	32,61	57,77
15.000	7.338	11.678	4.488	38,59	62,05	47,58	30,53	57,78
16.000	7.084	9.549	4.742	42,59	59,90	49,78	33,87	58,22
17.000	7.424	10.166	4.402	42,21	62,78	50,48	34,53	60,45
18.000	7.303	10.543	4.523	40,92	61,75	49,22	33,11	58,21
19.000	7.250	11.124	4.576	39,46	61,31	48,01	31,90	56,12
20.000	7.443	10.335	4.383	41,87	62,94	50,28	34,35	59,93

De nuevo, no se han obtenido los resultados esperados. En la tabla 4.6 se observa un valor de **AP** de la clase **Person** inferior al conseguido en **MS COCO** (ver tabla 4.8) a pesar de aumentar al doble el número de imágenes en el entrenamiento. El conjunto de todas las clases de interés **bags** representan un valor alto de **AP** y una vez más, la cantidad de **FP** es mayor a los **TP** debido a la clase **Person**. Esto deriva a obtener un average **IoU** más bajo del esperado a pesar de obtener valores aceptables en precisión, Recall, F-score y **mAP**.

Los resultados obtenidos son muy similares a los del primer entrenamiento pese a modificar parámetros de configuración del fichero **yolov4-obj.cfg**, aumentar el número de clases de los objetos de interés y aumentar el número de imágenes de entrenamiento y validación de la clase **Person**, clase que obtuvo malos resultados en el primer entrenamiento.

En la tabla 4.7 se aprecia que los mejores resultados obtenidos tuvieron lugar en la iteración 10.000, con un **47,26 %** de precisión, un **55,81 %** de Recall y un **51,18 %** de F-score. A pesar de obtener un **60,97 %** de **mAP**, el average **IoU** fue del **37,83 %** debido a la alta aparición de **FP** durante el entrenamiento. Se trata de un valor muy bajo.

Del mismo modo que en el primer entrenamiento, en la figura 4.13 se ilustra la evolución de las métricas en función de las iteraciones transcurridas. Estableciendo como métrica de prioridad el **mAP**, los mejores resultados se obtuvieron en la iteración 10.000. A partir de este punto oscilaron levemente con tendencia decreciente.

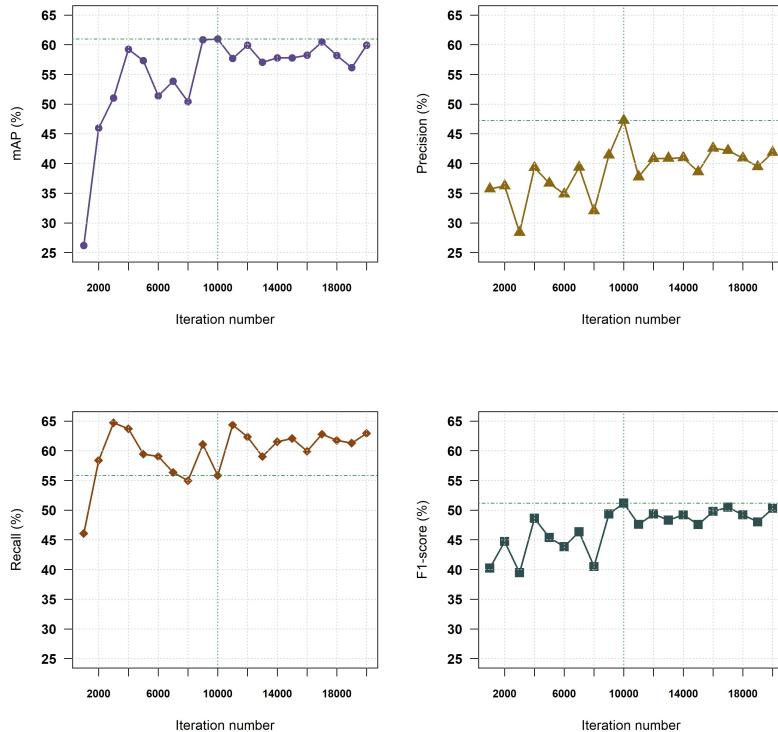


Figura 4.13: Resumen métricas segundo entrenamiento de la red neuronal con el dataset de OIDv4

Cabe detallar que, en los cálculos de las métricas de calidad de los dos entrenamientos que se han realizado se ha considerado un *threshold* del **IoU** del 0,5. Evidentemente, al aumentar el *threshold* a otros valores típicos como a 0,75 ó 0,95 disminuye drásticamente los valores de las métricas. Por norma general, cuando se ejecuta el detector de objetos de **YOLOv4** el *threshold* del **IoU** que viene definido por defecto es del 0,5. Es un valor suficientemente alto para considerar que la detección que se produce en cada fotograma de un vídeo es un **TP**.

En vista a los resultados, no se han superado las métricas de calidad del dataset **MS COCO** sobre el que se entrenó **YOLOv4**. Por tanto, se ha determinado continuar el proyecto con el modelo pre-entrenado de **YOLOv4** con **MS COCO** como dataset de referencia.

4.3.2. Métricas de calidad en MS COCO Dataset

En la siguiente tabla 4.8 se muestran los resultados obtenidos en la evaluación que se realizó en la sección 3.4. Las métricas más importantes sobre las clases de interés son las siguientes:

Tabla 4.8: Métricas de calidad de MS COCO en las clases de interés

Class	AP (%)	TP	FP
Person	79,53	7.923	3.168
Backpack	44,10	172	156
Handbag	29,83	158	215
Suitcase	71,08	205	102

Como se contempla en la tabla 4.8, excepto en la clase **Handbag**, se obtuvieron más **TP** que **FP** en las demás clases. El valor de **AP** en la clase **Person** es muy superior al obtenido durante los entrenamientos en el dataset personalizado. Por el contrario, en las clases **Backpack**, **Handbag** y **Suitcase** se obtuvieron valores más bajos que en el primer entrenamiento con el dataset **OIDv4**.

En las tablas 4.9 y 4.10 se puede comparar de mejor forma las métricas más importantes obtenidas en los dos entrenamientos realizados sobre el dataset **YOLOv4** respecto a las métricas obtenidas en la evaluación de **YOLOv4** sobre **MS COCO**. Destaca la superioridad **TP** sobre los **FP** de **MS COCO**, en concreto más del doble. En los dos entrenamientos se obtuvieron muchos más **FP** que **TP** causados por el mal entrenamiento de la clase **Person**.

Llama la atención que se superó el valor de **mAP** de **MS COCO** en el primer entrenamiento de **OIDv4**, a causa de los altos valores de **AP** obtenidos en las clases de objetos que no eran personas. Sin embargo, en ninguno de los entrenamientos se igualó el valor del average **IoU** de **MS COCO**. Respecto a la precisión, Recall y F-score, si bien no se obtuvieron malos resultados en los dos últimos, no se superaron en ningún caso a los de **MS COCO**.

Tabla 4.9: Comparativa métricas de calidad entre los test en OIDv4 y MS COCO [1]

Dataset	TP	FP	FN
MS COCO	22.730	10.889	13.027
OIDv4 test 1	430	527	273
OIDv4 test 2	6.600	7.366	5.226

Tabla 4.10: Comparativa métricas de calidad entre los dos test en OIDv4 y MS COCO [2]

Dataset	Precision (%)	Recall (%)	F-score (%)	average IoU (%)	mAP @ 0.5 (%)
MS COCO	67,61	63,57	65,53	56,04	64,16
OIDv4 test 1	44,93	61,17	51,81	34,13	66,25
OIDv4 test 2	47,26	55,81	51,18	37,83	60,97

Los resultados en los entrenamientos no fueron satisfactorios y se ha continuado el proyecto con el modelo de **YOLOv4** sobre el dataset **MS COCO**. Se ha llegado a la conclusión de que los datasets que han sido utilizados en los dos entrenamientos no contenían las imágenes idóneas para un correcta detección. Se ha observado que cuando las mochilas o bolsas de mano que se encontraban suspendidas en el suelo no se detectaban correctamente, dado que la mayoría de las imágenes de entrenamiento y validación son de mochilas que se encuentran perfectamente apoyadas en el suelo o las porta alguna persona. Por otro lado, las maletas de mano se han detectado correctamente ya que son equipajes típicamente rígidos que no muestran ninguna deformación, por lo que, aunque se encuentren en diferentes ángulos respecto a la visión de la cámara, las detecciones son eficientes.

Resultaría interesante evaluar las métricas en **YOLOv4** sobre otros datasets que contengan un gran número de clases de los objetos de interés, o realizar un dataset con un conjunto de imágenes propias para esta aplicación, con el fin de determinar si se superan las métricas de **MS COCO**, teniendo en cuenta que el principal objetivo de este proyecto es la detección de objetos abandonados.

En el siguiente [link](#) se pueden acceder a los resultados obtenidos en la evaluación de las métricas de calidad tanto del dataset **MS COCO** como en la evaluación de los entrenamientos sobre el dataset **OIDv4**.

4.3.3. Resultados en detección de objetos y personas con YOLOv4

Antes de realizar pruebas y evaluar los resultados del algoritmo de detección de objetos abandonados con **YOLOv4** y **Deep SORT** se ha realizado en primera instancia una evaluación del algoritmo de detección de **YOLOv4** con Tensorflow.

En la siguiente figura 4.14 se muestran los resultados obtenidos en las detecciones sobre el dataset **GBA2018** [4], donde se puede observar como la detección sobre las personas es muy buena, obteniéndose valores de *confidence score* por encima del 90 %. Un problema muy común que ha ocurrido en la evaluación de todos los datasets con las clases **Handbag** y **Backpack** es que, como se puede observar en la primera imagen de la figura 4.14, en ocasiones el detector se confunde y realiza una doble detección con valores muy bajos de *confidence score* sobre el objeto. Esto resulta un gran inconveniente a la hora de realizar la asociación persona-objeto en el algoritmo de detección de objetos abandonados, ya que en esas ocasiones se pierde con mucha facilidad la detección y en consecuencia hay muchos cambios de identidad. Otro problema, pero menos común, que se ha encontrado es cuando una mochila de dimensiones reducidas se encuentra apoyada en el suelo. Como se aprecia en la segunda imagen de la figura 4.14, el detector de objetos confunde la clase mochila con la clase perro. Típicamente las mochilas se detectan mejor cuando una persona la lleva encima.

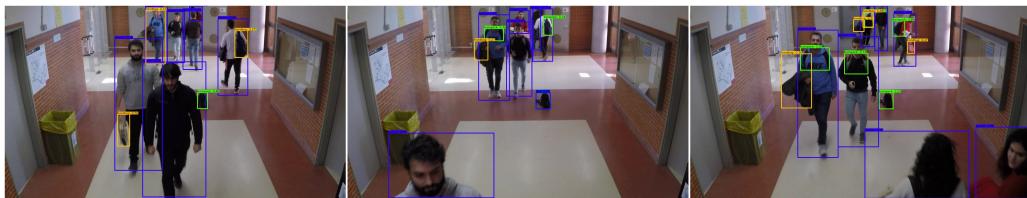


Figura 4.14: Ejemplo 1 detección YOLOv4 Tensorflow [4]

En la siguiente figura 4.15 se muestran los resultados obtenidos en las detecciones sobre el dataset **AVSSAB2007** [6], donde se puede observar como la detección en maletas son excelentes. En todas las evaluaciones que se han realizado del detector **YOLOv4** sobre todos los datasets empleados se han obtenido elevados *confidence score* para la clase **Suitcase**.

Se ha establecido un umbral de confianza mínima para todas las clases de 0,25. Tal y como se puede ver en la última imagen de la figura 4.15 no se estaba detectando la mochila porque no llegaba a ese umbral. No se ha disminuido el valor de este umbral de confianza dado que se obtenían muchos **FP** sobre todas las clases.



Figura 4.15: Ejemplo 2 detección YOLOv4 Tensorflow [6]

En la siguiente figura 4.16 se ilustran los resultados obtenidos en las detecciones sobre el dataset **ABODA** [7]. Como se observa en la primera imagen, una de las dos personas que se encuentran en la parte inferior no se detecta y la otra se detecta con el *confidence score* más bajo establecido. Por otro lado, la otra persona que se encuentra en un ángulo más favorable se detecta con un 40 % de *confidence score*, valores muy alejados de los que se obtienen sobre la clase **Person** en las evaluaciones de otros datasets.

Esta secuencia del dataset **ABODA** es un claro ejemplo de la gran influencia que tiene los cambios drásticos de iluminación, la distancia, el ángulo y la baja resolución de la cámara. En escenarios como el de esta secuencia se debe de entrenar una **CNN** con un conjunto de imágenes específico, teniendo en cuenta la posición donde se encontrará la cámara, así como su resolución, y por supuesto las condiciones lumínicas del entorno.



Figura 4.16: Ejemplo 3 detección YOLOv4 Tensorflow [7]

Todos los fotogramas han sido extraídos de las secuencias procesadas, las cuales se pueden acceder en el siguiente [link](#).

4.3.4. Resultados en seguimiento de objetos y personas con YOLOv4 y Deep SORT

Las pruebas que se han realizado previas a la evaluación del algoritmo de detección de objetos abandonados diseñado han sido el funcionamiento del algoritmo de seguimiento con **Deep SORT** en los diferentes datasets. En la siguiente figura 4.17 se aprecia como cada detección que ha realizado **YOLOv4** se le ha asignado un número identificativo. Esto quiere decir que a cada elemento detectado, tanto personas como objetos de interés se le asigna una identidad única que va a servir para estudiar su posición a lo largo de los siguientes fotogramas. Los cuadros delimitadores verdes representan a las personas y los cuadros delimitadores de color azul representan los objetos de interés que se han filtrado en la etapa de detección.

En la figura se muestra un fragmento de una de las secuencias de vídeo del dataset **AVSSAB2007** [6]. Siempre que no se producieran oclusiones prolongadas, el algoritmo trabajaba correctamente pudiendo rastrear objetos que en determinados momentos se salieran del plano. Como se puede observar en la primera imagen de la figura 4.17, la persona con número de identidad 93 pasa por detrás de la columna del andén, perdiéndose tanto la detección como el seguimiento. Sin embargo, fotogramas más tarde, cuando vuelve a aparecer al otro lado de la columna, se vuelve a detectar y se le asigna la identidad que tenía anteriormente. En las dos primeras imágenes de la figura se puede apreciar también la identidad que se le ha asignado a la bolsa de mano que aparece al final del andén, con el número identificativo 71.

En la última imagen de la figura se perciben cambios de identidad tanto de la persona que anteriormente tenía asignado como número de identidad el 93, así como la bolsa de mano que ha pasado de tener el número 71 al 88. Si nos fijamos bien en las dos últimas imágenes de la figura se puede observar que lo que se ha producido con la persona es un intercambio de identidad con la persona que se encuentra a la derecha de la columna en la segunda imagen. Suele ser un problema muy común en el rastreo de personas y objetos cuando se producen oclusiones o desaparición de la detección durante un tiempo determinado.



Figura 4.17: Ejemplo 1 seguimiento YOLOv4 y Deep Sort [6]

En la siguiente figura 4.18 se observa claramente uno de los mayores problemas con los que nos hemos encontrado en el desarrollo del algoritmo de detección de objetos abandonados y es el cambio repentino de identidad al perderse la detección durante una corta duración de tiempo. En la primera imagen se puede observar a una persona con número de identidad 3 que ha dejado en el pasillo de la cafetería de la **EPS** una bolsa de mano con número de identidad 6. Pocos fotogramas después, cuando el individuo vuelve para recoger la bolsa de la cual es propietario, aparece con un nuevo número de identidad, ya que cuando se encontraba al fondo del pasillo se perdió la detección y se le asignó un nuevo número identificativo.

Además, al recoger la bolsa del mano que se encontraba en el suelo, debido a los cambios de ángulos y deformaciones que se producen en la bolsa, se pierde la detección y tanto el algoritmo de detección como el algoritmo de rastreo determinan que esa bolsa de mano se trata de un nuevo objeto.



Figura 4.18: Ejemplo 2 seguimiento YOLOv4 y Deep Sort [4]

Con ello, se produce un cambio en el número identificativo del 6 al 70. Esto se trata de un grave problema a la hora de realizar la asociación persona-objeto en el algoritmo de detección de objetos abandonados. En muchos objetos, sobre todo en bolsas de mano, suelen tener asas y morfologías muy distintas al poder deformarlas al cogerlas o dejarlas en el suelo, que provocan que resulte muy difícil determinar que se sigue tratando del mismo objeto a lo largo de un número determinado de fotogramas. Este problema no aparece en maletas o equipajes de mano grandes y robustos, dado que no pierden su forma y resulta más fácil su identificación independientemente del ángulo en el que se encuentren.

Otro problema ajeno a la etapa de detección totalmente dependiente del algoritmo de seguimiento y que ocurre con relativa frecuencia en secuencias de vídeo de grandes aglomeraciones de personas y obstáculos que provocan la pérdida de las detecciones durante breves instantes de tiempo, es el cambio de identidad entre personas y objetos. En la última imagen de la figura 4.18 se observa como la mujer que se encuentra en el centro del pasillo se le ha asignado el número identificativo 90, la identidad de la bolsa de mano que portaba una persona en la segunda imagen de la figura.

La siguiente figura corresponde al análisis del algoritmo de seguimiento sobre un fragmento de una de las secuencias de vídeo del dataset PETSc007 [5].



Figura 4.19: Ejemplo 3 seguimiento YOLOv4 y Deep Sort [5]

En esta figura 4.19 se pueden observar nuevamente los dos grandes problemas que nos podemos encontrar en el seguimiento de objetos y personas, oclusiones o cambios de identidad y clases. En las dos primeras imágenes se pueden observar a dos mujeres con números de identidad 37 y 72 respectivamente, que se encuentran en un primer plano del área de detección de la cámara. Como se puede observar en ambas imágenes, estas dos personas están por delante de otras personas que se encuentran en planos posteriores, provocando oclusiones e impidiendo que se puedan detectar y rastrear. Por otro lado, se puede observar en la última imagen de la figura como, tras moverse la mujer que tenía como número identificativo el 72, aparece detectado y rastreado un niño al que se le está detectando como una bolsa de mano con número identificativo 38. Esto ha sucedido porque en fotogramas previos se detectó durante un instante de tiempo muy corto una bolsa de mano que se le asignó ese número de identidad y al desaparecer y aparecer el niño se le ha re-asignado esa identidad errónea.

Como se puede observar en todos los ejemplos descritos, a pesar de que Deep SORT haya solucionado los graves problemas que tenía SORT en las oclusiones y en los cambios de identidad, este problema persiste en muchas ocasiones.

Todos los fotogramas han sido extraídos de las secuencias procesadas, las cuales se pueden acceder en el siguiente [link](#).

4.3.5. Resultados en algoritmo de detección de objetos abandonados

Finalmente, tras comprobar el correcto funcionamiento de **YOLOv4** como detector de objetos en el framework Tensorflow y evaluando su funcionamiento con el algoritmo de rastreo con **Deep SORT**, se ha evaluado el desarrollo del algoritmo de detección de objetos abandonados en los distintos datasets descritos en la sección 4.2.2.

En la siguiente figura 4.20 se aprecia un fragmento de una secuencia del dataset **AVSSAB2007** [6]. En él se puede observar que tras los primeros segundos de vídeo se ha creado una asociación entre la persona con número de identidad 5 con la maleta que porta con número de identidad 2. Al fondo del andén se puede observar otra persona que se le ha asociado con número de identidad 3. Dado que la distancia respecto a la maleta con identidad 2 no es menor a 2 veces el ancho del cuadro delimitador de la persona, no se le ha asignado como propietario la maleta. No obstante, en el caso de que si que se cumpliera la distancia mínima a la que se le pudiera asignar la maleta, cobra mayor peso la mínima distancia que se encuentra respecto a una persona, que en este caso, como ya se ha comentado, es la persona con número de identidad 5.

En los siguientes fotogramas de la figura 4.20 se puede observar como la persona se está alejando de la maleta. En la segunda imagen de la segunda fila se puede observar que la persona se encuentra a una distancia 3 veces mayor a la distancia que se estableció como distancia de asociamiento, con lo cual, la maleta se encuentra en zona de alarma y se indica mediante cuadros delimitadores tanto en la persona como en la maleta, que ésta se encuentra en riesgo de ser abandonada.

En la segunda imagen de la tercera fila ya se puede ver que la persona ha sobrepasado la zona delimitada como zona de alarma, y ahora se encuentra a una distancia 5 veces mayor a la distancia de asociamiento. En este caso se ha establecido que es una distancia considerablemente grande para determinar que el objeto ha sido abandonado. En el caso de que no se hubiera llegado a esa distancia de abandono, pero la persona hubiera desaparecido del plano de la cámara, se habría determinado de igual manera que la maleta ha sido abandonada, es decir, en el momento en el que se pierda la asociación porque la identidad de la persona ya no es rastreada, se entiende que la persona ya no se encuentra en escena y ha quedado abandonado el objeto del cual es propietario.



Figura 4.20: Individuo abandona su maleta en el la zona cercana del andén [6]

En la siguiente figura 4.21 se ilustra otro ejemplo similar al anterior. En este caso, se ha evaluado el algoritmo sobre la primera secuencia de vídeo del dataset **ABODA** [7].

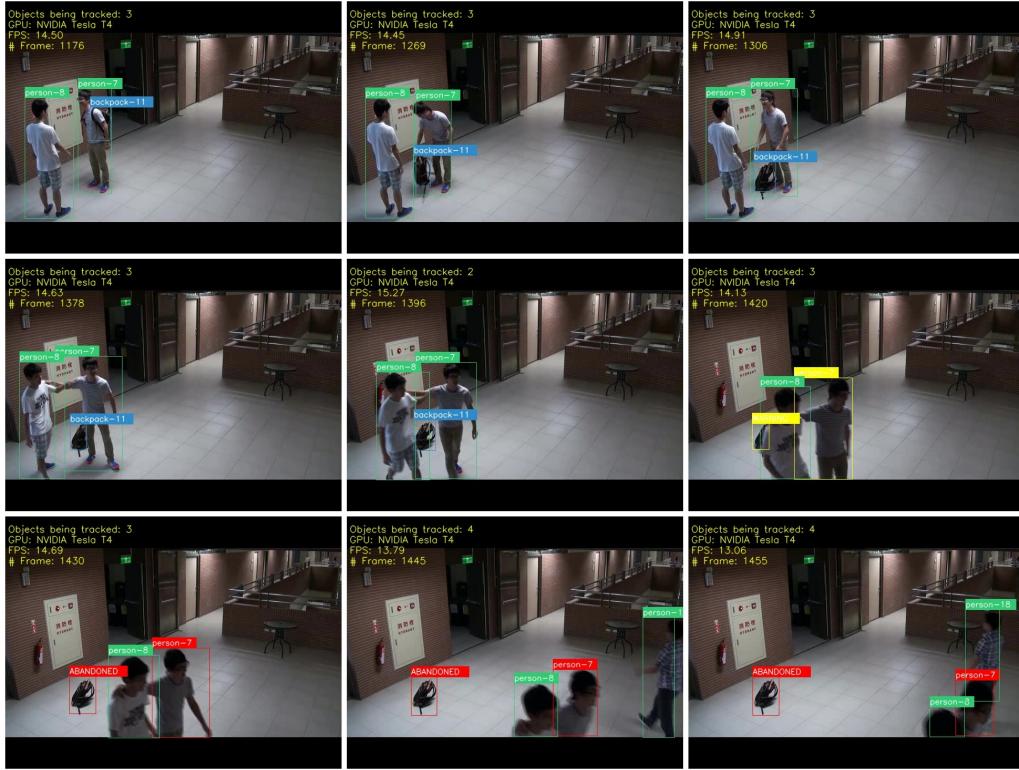


Figura 4.21: Individuo abandona su mochila en el hall de un edificio [7]

En la escena se observan dos chicos jóvenes que se encuentran en el hall de un edificio y uno de ellos con número de identidad 7 porta una mochila con número de identidad 11. A pesar de que en el plano de visión también se encuentre otra persona, prima la mínima distancia entre mochila y persona, que en este caso es con la persona con identidad 7. Una vez creada la asociación se puede presenciar en los fotogramas siguientes como las dos personas abandonan el lugar dejando la mochila desatendida.



Figura 4.22: Individuo abandona su bolsa de mano en el pasillo de cafetería de la EPS (1) [4]

En las figuras 4.22 y 4.23 se puede observar un claro ejemplo de uno de los problemas más comunes a la hora de detectar y rastrear objetos y personas, que se tratan de las occlusiones. Este fenómeno se

produce cuando una persona u objeto se sitúa delante del otro, dificultando en la mayoría de ocasiones las detecciones y provocando cambios de identidad en el rastreo.

En la figura 4.22 se puede observar como una persona con número de identidad 17 porta una bolsa de mano con número de identidad 23 y la posa sobre el suelo del pasillo de la entrada de la cafetería de la EPS [4]. También lleva otra bolsa de mano con número de identidad 26, pero no se detectó correctamente durante los primeros de segundos de vídeo y no se produjo la asociación. En cambio, con la bolsa de mano con identidad 23 sí. En los siguientes fotogramas se aprecia como la persona se aleja hacia el fondo del pasillo abandonando la bolsa de mano que dejó en el suelo anteriormente. De igual manera que en ejemplos anteriores, el algoritmo avisa cuando la bolsa de mano se encuentra en zona de alarma mediante cuadros delimitadores amarillos y cuando se considera que ha sido abandonada en rojo.



Figura 4.23: Individuo abandona su bolsa de mano en el pasillo de cafetería de la EPS (2) [4]

En la figura 4.23 anterior se puede apreciar que ocurre el fenómeno de oclusión. Cuando el individuo llega al final del pasillo y se da la vuelta queda por detrás de la persona con número de identidad 33 y ésta a su vez se encuentra detrás de la mujer con número de identidad 36. Durante estos breves instantes que se está produciendo oclusión debido a la posición de los sujetos se pierden las detecciones y en consecuencia el rastreo, produciéndose un cambio de identidad entre la persona con identidad 17 y la persona con identidad 33. Como se puede ver, las occlusiones provocan grandes problemas ya que pueden producir cambios de identidad sobre personas a las que ya se le había asignado un objeto de su propiedad, y en este caso, lo acababa de abandonar. De tal manera que se está identificando como identidad provocadora del abandono a la persona incorrecta.



Figura 4.24: Una bolsa de mano se encuentra abandonada al final de un andén de trenes [6]

Uno de los posibles escenarios que se barajó durante el desarrollo del algoritmo del detector de objetos abandonados en la sección 3.7 es la posibilidad de que al inicio de una secuencia de vídeo se encontrara

un objeto que no se le pudiera asignar como propietario a ninguna persona por que no se encontraba a la distancia mínima de asociación persona-objeto. En este fragmento de vídeo de una secuencia del dataset [AVSSAB2007](#) [6] se puede observar como se detecta una bolsa de mano al final de andén que se encuentra desde un principio abandonado. En este caso, al tratarse de un vídeo de 25 FPS, si desde un primer momento el objeto está abandonado, al cabo de 15 segundos, es decir, 375 fotogramas después, el objeto se identifica como abandonado.

Como se puede observar en la primera imagen de la segunda fila de la figura 4.24, en el fotograma 402 ya se ha determinado esa bolsa de mano como abandonada. Cabe recalcar que, en objetos que se encuentran a una distancia alejada, en la etapa de detección suele confundir cuando es una maleta, una mochila o una bolsa de mano. En este caso está detectando la bolsa de mano como si fuera una maleta.

En la figura 4.25, perteneciente a la evaluación del algoritmo de detección de objetos abandonados sobre un fragmento del dataset [PETS2007](#) [5], se aprecia a una persona que inicialmente se le ha asignado como número de identidad el 28, porta una bolsa grande de deporte donde en un principio el algoritmo de detección lo confunde con una persona.

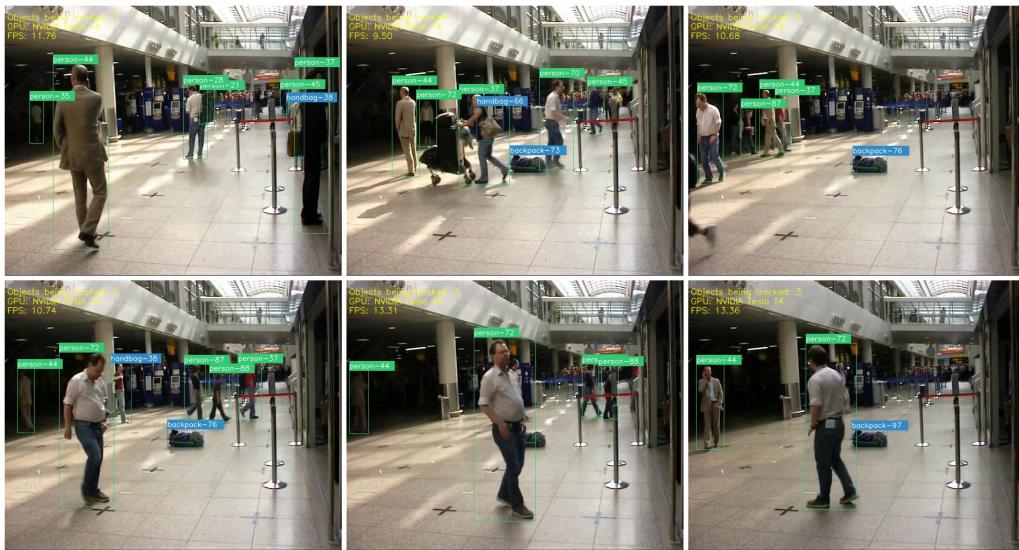


Figura 4.25: Individuo abandona una bolsa de deporte en un aeropuerto [5]

En los siguientes fotogramas cambia la identidad de la persona al número 70 y ya se le ha asignado una identidad de objeto a la bolsa de deporte con la identidad 73 (ver segunda imagen de la primera fila de la figura 4.25). Pocos fotogramas después vuelven a cambiar las identidades tanto de la persona como de la bolsa de deporte a 72 y 76 respectivamente. En este punto ya se ha terminado el periodo de asignación de objetos a personas con lo cual no es posible crear una asociación, además en ese momento la persona se encuentra alejada de la bolsa desatendida. Cuando la persona vuelve a acercarse a la bolsa de la cual es propietario vuelve a ocurrir el fenómeno de occlusión que se vio en el ejemplo sobre el dataset [GBA2018](#) [4]. Al moverse, una vez más, vuelve a cambiar la identidad de la bolsa de deporte a 97. Todos estos sucesos de cambios de identidad que se producen en esta secuencia de vídeo son producidos por las occlusiones de las personas que se van moviendo por la zona del aeropuerto, imposibilitando la creación de asociación entre personas y objetos.

Como se ha podido observar en los ejemplos que se acaban de exponer, el algoritmo que se ha propuesto para el propósito de este proyecto funciona correctamente siempre y cuando se ejecuten las etapas de detección y rastreo de objetos y personas sin errores. El principal problema que se ha encontrado en esta parte del trabajo ha sido las occlusiones producidas, lo que conlleva a pérdidas de detecciones y en consecuencia a cambios de identidad o tener que volver a reidentificar, impidiendo la asociación persona-objeto o asignando como responsable de un abandono a la persona equivocada.

Todos los fotogramas han sido extraídos de las secuencias procesadas, las cuales se pueden acceder en el siguiente [link](#).

4.4. Conclusiones

Este capítulo se ha dividido en dos partes claramente diferenciadas. En la primera parte se han calculado las métricas de calidad sobre los dos datasets utilizados en este proyecto, [MS COCO](#) y [OIDv4](#). En la segunda parte se han evaluado, primeramente [YOLOv4](#), seguido del algoritmo de rastreo [Deep SORT](#) y por último, el algoritmo de detección de objetos abandonados que se desarrolló en la sección 3.7 sobre los datasets más relevantes en la evaluación algoritmos de detección de objetos abandonados como pueden ser [ABODA](#) [7], [GBA2018](#) [4], [AVSSAB2007](#) [6] o [PETS2007](#) [5].

Pese a que las métricas obtenidas en el dataset [MS COCO](#) sobre las clases de interés: [Person](#), [Suitcase](#), [Handbag](#) y [Backpack](#) son buenas, se pretendió intentar mejorarlas entrenando una [CNN](#) basada en imágenes de las mismas clases del dataset [OIDv4](#), ya que dispone de imágenes diferentes a las de [MS COCO](#). Tras finalizar el primer entrenamiento, se pudo ver en las tablas 4.5 y 4.10 que a pesar de que se obtuvieron buenos resultados de [mAP](#), los valores de precisión, Recall y F-score fueron inferiores a las de [MS COCO](#).

Posteriormente, se volvió a intentar mejorar las métricas aumentando el número de clases de entrenamiento, ya que el dataset [OIDv4](#) dispone de más clases de interés como pueden ser [Plastic bag](#), [Luggage and bags](#) y [Briefcase](#). También, se aumentaron el número de imágenes de entrenamiento y validación así como el número de máximas iteraciones. Los resultados que se obtuvieron en las tablas 4.7 y 4.10 fueron similares a las del primer entrenamiento, por lo que se decidió continuar el proyecto utilizando el dataset [MS COCO](#) sobre el que viene pre-entrenado [YOLOv4](#).

En la segunda parte del presente capítulo se ha evaluado en primer lugar la etapa de detección con el funcionamiento de [YOLOv4](#) sobre los distintos datasets antes nombrados. A pesar de que se mejoran las predicciones en términos de velocidad-precisión respecto a los resultados obtenidos en [93], siguen apareciendo el problema de las occlusiones. Cuando una persona se sitúa por delante del otra respecto al plano frontal de la cámara, fácilmente identifica únicamente a la persona que se encuentra más cerca, lo que conlleva que en la etapa de rastreo no se le pueda asignar una identidad a dicha persona. Lo mismo ocurre con los objetos de interés. Se ha observado que con una única red pre-entrenada no se detecta de la misma forma sobre los datasets evaluados. Como ya se vio en el último ejemplo de la sección 4.3.3, fenómenos como pueden ser los cambios de iluminación, calidad en la resolución del vídeo, escenas de día/noche, objetos de interés a largas distancias y en ángulos desfavorables u occlusiones, provocan que las predicciones se vean afectadas ocasionalmente en múltiples ocasiones [FP](#).

Posterior a la evaluación de la etapa de detección con [YOLOv4](#) se ha analizado los resultados ejecutando el algoritmo de detección y seguimiento con [Deep SORT](#). Con el fin de que no se producieran [FP](#) se aumentó el umbral de confianza de 0,25, valor que se ha utilizado por defecto en la evaluación de las predicciones en la detección con [YOLOv4](#), a un valor de 0,45. Con ello se ha asegurado un rastreo confiable en los objetos detectados en la etapa previa. Sin embargo, del mismo modo que cuando se evaluaron las detecciones, las clases [backpack](#) y [handbag](#) se confundían con frecuencia y con el transcurso de los fotogramas cambiaba la clase detectada y en consecuencia se perdía la identidad que se le había asignado a dicho objeto. Otro problema que se ha encontrado con frecuencia y que está totalmente vinculado a la etapa de detección es la aparición de occlusiones en secuencias de vídeo de grandes aglomeraciones de personas. Al pasar una persona por delante de otra desde el punto de vista frontal de la cámara podían ocurrir dos sucesos. El primero es que solo se detectara a una persona (la persona que se encontraba en el primer plano), con lo cual se perdía la detección y con ello el rastreo de la identidad de la persona que se encontraba en el segundo plano. El otro suceso se trata cuando dos personas se cruzaban y se producía un intercambio de identidad. En ambos casos perjudicaba el rastreo de objetos y personas, así como la asociación persona-objeto en el algoritmo de detección de objetos abandonados.

Finalmente, el algoritmo de detección de objetos abandonados que se planteó en la figura 3.15 funciona correctamente, pero es totalmente dependiente de que la etapa de detección y la etapa de seguimiento de personas y objetos se realicen sin fallos. Aparecieron dos principales problemas durante la evaluación del algoritmo. Uno de ellos fueron las occlusiones producidas por las personas durante las secuencias de vídeo de grandes aglomeraciones, lo que provocaba cambios de identidad entre los sujetos y en consecuencia, incapacidad de establecer una asociación persona-objeto al inicio del vídeo. El segundo problema fue los cambios de clase, sobre todo en bolsa de mano pequeñas. En ciertas ocasiones, según el ángulo donde se situase el objeto, en la etapa de detección se realizaban predicciones de que el objeto era una mochila, y fotogramas después pasaba a ser una bolsa de mano. Esto producía que no se pudiera crear la asociación persona-objeto o que, una vez creada, desapareciese la identidad del objeto de la secuencia de vídeo.

Capítulo 5

Conclusiones y líneas futuras

El trabajo ocupará una gran parte de tu vida, la mejor forma de lidiar con ello, es encontrar algo que realmente ames.

Steve Jobs

En este apartado se resumen las conclusiones obtenidas y se proponen futuras líneas de investigación derivadas del presente [TFM](#).

5.1. Conclusiones

Este proyecto se ha enfocado en el desarrollo de una estrategia para la detección de objetos abandonados en sistemas de videovigilancia. Para ello, se realizó un estudio del Estado del Arte sobre las distintas metodologías en la detección de objetos abandonados. Las técnicas que se llevaban utilizando en los últimos años son las de segmentación de objetos en primer plano, detección de objetos estacionarios, reconocimiento de comportamientos y detección de personas y objetos.

Se ha decidido utilizar [CNN](#) como estrategia en la detección de personas y objetos, ya que se pueden desarrollar sistemas de detección en tiempo real. Tras evaluar las métricas de los distintos algoritmos de detección de objetos se ha escogido [YOLOv4](#) ya que, respecto a sus principales competidores, [SSD](#), EfficientDet o Faster [R-CNN](#), presenta una mayor relación de [mAP](#) y [FPS](#) en los benchmarks del dataset [MS COCO](#).

Se ha entrenado dos redes neuronales para que solo detectase las clases de interés: personas, mochilas, bolsas de mano y maletas. Se ha empleado el dataset de [OIDv4](#) para el entrenamiento de la red, tomando 1.500 imágenes de cada clase para el entrenamiento y 300 de cada clase para la validación (un 20 % de las imágenes respecto a las del entrenamiento). Es aconsejable realizar un entrenamiento con 2.000 iteraciones por cada clase entrenada, por tanto, teniendo 4 clases, se fijó las iteraciones máximas a 8.000. En el primer entrenamiento salieron valores de [mAP](#) aceptables, sin embargo, como se observa en la tabla 4.5, aparecieron muchos más [FP](#) que [TP](#), lo que derivó a obtener valores de [IoU](#) muy bajos, inferiores al 50 %. En el segundo entrenamiento de la red neuronal se propuso aumentar el número de clases. A diferencia de [MS COCO](#), que solo dispone de 80 clases muy generalizadas, el dataset [OIDv4](#) ofrece clases más específicas, como modelos concretos de maletas o de bolsas de mano, que resultan interesantes para el entrenamiento de nuestra red. Se emplearon 7 clases y se subió las máximas iteraciones a 20.000, por no limitar el entrenamiento a lo mínimo recomendable. Los valores de las métricas de calidad obtenidas fueron muy similares a los del primer entrenamiento. En vista a los resultados de los dos entrenamientos, se decidió continuar el proyecto con el dataset de referencia [MS COCO](#), ya que las métricas de calidad de los dos entrenamientos realizados fueron muy inferiores.

Se han estudiado los métodos de [MOT](#) mas recientes, [SORT](#) y [Deep SORT](#). Se ha comprobado que utilizando el algoritmo de seguimiento [Deep SORT](#), el cual trabaja excelentemente con [YOLOv4](#), se obtuvieron buenos resultados en el rastreo de personas y objetos de interés. Se produjeron problemas en el seguimiento en secuencias de vídeo donde hay grandes aglomeraciones de personas que se superponen

entre sí produciendo oclusiones o desaparecen del plano durante varios segundos, lo cual origina que se pierda el seguimiento de la identidad, obligando a crear una nueva identidad y asociación entre persona y objeto.

Se ha diseñado un algoritmo capaz de detectar objetos abandonados. Se han expuesto dos posibles escenarios. El primer escenario es que el objeto permanezca inmóvil en el mismo punto durante toda la secuencia de vídeo y no tenga ninguna persona asociada como propietaria. En tal caso cuando han transcurrido 15 segundos, se indica que ese objeto se encuentra perdido o ha sido abandonado. El segundo escenario se ha tenido que lidiar con el problema de la asociación persona-objeto. Para ello, se ha calculado la distancia de los centroides de los cuadros delimitadores entre las personas y todos los objetos de interés detectables, estableciendo una asociación persona-objeto cuando se detectase la mínima en píxeles. Una vez creada, la asociación se ha podido comprobar que el algoritmo diseñado detecta cuando un objeto se aleja de su propietario una distancia equivalente a 3 ó 5 veces la distancia medida cuando se genera la asociación (dependiendo del nivel de alerta), durante un tiempo determinado.

Finalmente se ha validado el funcionamiento de los algoritmos con los datasets más empleados en la evaluación de sistemas de detección de objetos abandonados, como son [PETS2007](#), [ABODA](#), [AVS-SAB2007](#) o [GBA2018](#). Se han escogido secuencias de vídeo diferenciadas donde ocurren cambios de iluminación, objetos cercanos o alejados o distintas resoluciones de vídeo.

No se obtuvieron buenos resultados cuando el objeto que había sido abandonado se encontraba totalmente apoyado en el suelo. Ocurrían dos sucesos, el primero es que la detección cambiase cada pocos segundos y se perdiera tanto la identidad del objeto como la asociación persona-objeto. El segundo suceso es que la predicción bajara del 25 % y se dejara tanto de detectar como de rastrear. Esto se puede deber a que [MS COCO](#) no contiene las suficientes imágenes de mochilas, bolsas de mano y maletas desde ángulos complejos o sin un propietario portando dicho objeto, por tanto se pierde precisión en la detección.

En las secuencias de vídeo donde se evaluaba el abandono de maletas o bolsas grandes que no perdían su forma al apoyarlas en el suelo, se obtuvieron buenos resultados con tasas bajas de fallos en la detección de objeto abandonado.

5.2. Líneas futuras

A continuación, se van plantear posibles líneas futuras para mejorar la estrategia planteada en este proyecto, así como trabajos que pueden ser desarrollados utilizando el presente como base:

- **Utilizar Scaled-YOLOv4.** En [1] se demostró que la red neuronal de detección [YOLOv4](#) escala tanto hacia arriba como hacia abajo y es aplicable a redes tanto grandes como pequeñas manteniendo una alta precisión y velocidad. En diciembre de 2020 se propuso en [94] un enfoque de escalamiento de la red que modifica la profundidad, ancho, resolución y estructura de la red neuronal obteniéndose una [mAP](#) del 73,4 % sobre el dataset de [MS COCO](#) a una velocidad de 16 [FPS](#) con una NVIDIA Tesla V100. Se trata de la mayor precisión sobre el dataset de [MS COCO](#) que se conoce en todos los trabajos publicados. Esto permite mejorar las detecciones y en consecuencia un mayor seguimiento sobre las personas y los objetos con lo que podría solucionar los problemas de pérdidas de asociación persona-objeto producidos en las evaluaciones de datasets más complejos.
- **Diseño de un algoritmo de detección de objetos abandonados basado en [On the Fairness of Detection and Re-Identification in Multiple Object Tracking \(FairMOT\)](#).** Dos puntos clave en la reidentificación de objetos es la detección y seguimiento. Ningún trabajo publicado ha propuesto una solución que se realice estas dos tareas dentro de una misma red. Al depender el seguimiento de las detecciones hace que se pierda precisión y cause pérdidas de identidad. En [95] se propone una red neuronal capaz de detectar y reidentificar objetos de manera simultánea y obteniendo una alta puntuación en el benchmark dataset MOT16 [84].
- **Diferenciar cuando un objeto ha sido abandonado o robado.** Durante todo el proyecto se ha trabajado el desarrollo de un algoritmo capaz de detectar cuando un objeto se encontraba o ha sido abandonado. No se ha contemplado la posibilidad de que el objeto haya podido ser robado. En [96] se propone una estrategia para diferenciar cuando un objeto ha sido abandonado y cuando ha sido robado. Sería interesante tener ese trabajo como referencia para poder tener en cuenta esa variable en la detección de objetos en los sistemas de videovigilancia.

- **Entrenar red neuronal con otros datasets.** En el presente proyecto se han entrenado dos modelos de YOLOv4 basados en el dataset OIDv4 obteniéndose malos resultados. Finalmente se propuso continuar con el modelo pre-entrenado de YOLOv4 en MS COCO. Sería interesante probar otros datasets como el de ImageNet [81] o crear un dataset propio y comprobar si se mejoran las métricas de calidad.
- **Implementación en tarjetas NVIDIA Jetson.** Durante la realización de este TFM se han evaluado los algoritmos sobre equipos con especificaciones altas o mediante el uso de Google Colab, donde se emplean las GPU's NVIDIA Tesla K80, P4, T4 y P100. De cara a comercializar un servicio que ofrezca la detección de objetos abandonados y se descarte un servicio en la nube sería interesante comprobar si, utilizando tarjetas de baja potencia como pueden ser las NVIDIA Jetson Xavier NX o Jetson AGX Xavier, que vienen con CUDA instalado de fábrica, se puede utilizar durante largos tiempos de ejecución en sistemas de videovigilancia.
- **Distanciamiento social.** A partir de este trabajo se puede plantear una estrategia de detección y seguimiento en el distanciamiento social. En los últimos meses está creciendo las demandas de sistemas de seguridad que incluyan el control de distancia social en aforos debido al COVID-19. En el presente proyecto se plantea una estrategia donde se emplea un algoritmo de detección junto a un algoritmo de seguimiento que puede ser utilizado como base para desarrollar una red neuronal capaz de detectar si se está cumpliendo el distanciamiento social social o no. En [97], [98], [99] y [100] se están empleando CNN con YOLO y Deep SORT para abordar este problema.

Bibliografía

- [1] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “Yolov4: Optimal speed and accuracy of object detection,” 2020.
- [2] N. Wojke, A. Bewley, and D. Paulus, “Simple online and realtime tracking with a deep association metric,” in *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2017, pp. 3645–3649.
- [3] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, “Microsoft coco: Common objects in context,” 2015.
- [4] “Acceso al dataset gba2018,” <https://bit.ly/3tIJdUx> [Último acceso 17/diciembre/2020].
- [5] “Acceso al dataset pets2007,” <http://www.cvg.reading.ac.uk/PETS2007/data.html> [Último acceso 24/octubre/2020].
- [6] “Acceso al dataset avssab2007,” http://www.eecs.qmul.ac.uk/~andrea/avss2007_d.html [Último acceso 13/junio/2020].
- [7] “Acceso al repositorio de github del dataset aboda,” <https://github.com/kevinlin311tw/ABODA> [Último acceso 09/agosto/2020].
- [8] K. N. Plataniotis and C. S. Regazzoni, “Visual-centric surveillance networks and services [guest editorial],” *IEEE Signal Process. Mag.*, vol. 22, no. 2, pp. 12–15, 2005. [Online]. Available: <https://doi.org/10.1109/MSP.2005.1406464>
- [9] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov, and et al., “The open images dataset v4,” *International Journal of Computer Vision*, vol. 128, no. 7, p. 1956–1981, Mar 2020. [Online]. Available: <http://dx.doi.org/10.1007/s11263-020-01316-z>
- [10] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, “Simple online and realtime tracking,” *2016 IEEE International Conference on Image Processing (ICIP)*, Sep 2016. [Online]. Available: <http://dx.doi.org/10.1109/ICIP.2016.7533003>
- [11] E. Luna, J. Sanmiguel, D. Ortego, and J. Martínez, “Abandoned object detection in video-surveillance: Survey and comparison,” *Sensors*, vol. 18, p. 4290, 12 2018.
- [12] L. Patino, T. Cane, A. Vallee, and J. Ferryman, “Pets 2016: Dataset and challenge,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2016, pp. 1240–1247.
- [13] T. Bouwmans, “Traditional and recent approaches in background modeling for foreground detection: An overview,” *Computer Science Review*, vol. 11-12, pp. 31–66, 2014. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1574013714000033>
- [14] Álvaro Bayona, J. C. SanMiguel, and J. M. Martínez, “Comparative evaluation of stationary foreground object detection algorithms based on background subtraction techniques,” in *2009 Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance*, 2009, pp. 25–30.
- [15] Álvaro García-Martín and J. M. Martínez, “People detection in surveillance: classification and evaluation,” *IET Computer Vision*, vol. 9, no. 5, pp. 779–788, 2015. [Online]. Available: <https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-cvi.2014.0148>

- [16] A. Ben Mabrouk and E. Zagrouba, “Abnormal behavior recognition for intelligent video surveillance systems: A review,” *Expert Systems with Applications*, vol. 91, pp. 480–491, 2018. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0957417417306334>
- [17] L. L. Ankile, M. F. Heggland, and K. Krange, “Deep convolutional neural networks: A survey of the foundations, selected improvements, and some current applications,” 2020.
- [18] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” 2016.
- [19] “How single-shot detector works,” <https://developers.arcgis.com/python/guide/how-ssd-works/> [Último acceso 11/enero/2021].
- [20] M. Tan, R. Pang, and Q. V. Le, “Efficientdet: Scalable and efficient object detection,” 2020.
- [21] “Object tracking with dlib,” <https://n9.cl/f1ne4> [Último acceso 07/enero/2021].
- [22] “Coco - common objects in context,” <https://cocodataset.org/#home> [Último acceso 02/marzo/2021].
- [23] A. Vittorio, “Toolkit to download and visualize single or multiple classes from the huge open images dataset v4,” https://github.com/EscVM/OIDv4_ToolKit, 2018.
- [24] B. Benfold and I. Reid, “Stable multi-target tracking in real-time surveillance video,” *CVPR 2011*, pp. 3457–3464, 2011.
- [25] R. Padilla, S. L. Netto, and E. A. B. da Silva, “A survey on performance metrics for object-detection algorithms,” in *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, 2020, pp. 237–242.
- [26] J. Mohajon, “Confusion matrix for your multi-class machine learning model, [Último acceso 15/noviembre/2020],” <https://cutt.ly/AIBPiXj>, 2020.
- [27] M. Baptista-Ríos, C. Martínez-García, C. Losada-Gutiérrez, and M. Marrón-Romera, “Human activity monitoring for falling detection. a realistic framework,” in *2016 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, 2016, pp. 1–7.
- [28] “Descarga del instalador de anaconda para linux,” <https://www.anaconda.com/products/individual> [Último acceso 04/julio/2020].
- [29] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “Ssd: Single shot multibox detector,” *Lecture Notes in Computer Science*, p. 21–37, 2016. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-46448-0_2
- [30] A. Filonenko, K.-H. Jo *et al.*, “Unattended object identification for intelligent surveillance systems using sequence of dual background difference,” *IEEE Transactions on Industrial Informatics*, vol. 12, no. 6, pp. 2247–2255, 2016.
- [31] Wahyono and K.-H. Jo, “Cumulative dual foreground differences for illegally parked vehicles detection,” *IEEE Transactions on Industrial Informatics*, vol. 13, pp. 2464–2473, 2017.
- [32] F. Lv, X. Song, B. Wu, V. Kumar, and S. R. Nevatia, “Left luggage detection using bayesian inference,” in *In PETS*, 2006.
- [33] R. Benenson, M. Omran, J. Hosang, and B. Schiele, “Ten years of pedestrian detection, what have we learned?” 2014.
- [34] S. Zhang, R. Benenson, M. Omran, J. Hosang, and B. Schiele, “How far are we from solving pedestrian detection?” 2016.
- [35] J. Hosang, M. Omran, R. Benenson, and B. Schiele, “Taking a deeper look at pedestrians,” 2015.
- [36] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” 2014.

- [37] B. Wu, A. Wan, F. Iandola, P. H. Jin, and K. Keutzer, “Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving,” 2019.
- [38] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” 2016.
- [39] C.-Y. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg, “Dssd : Deconvolutional single shot detector,” 2017.
- [40] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, 2001, pp. I–I.
- [41] C.-Y. Lin, K. Muchtar, and C.-H. Yeh, “Robust techniques for abandoned and removed object detection based on markov random field,” *Journal of Visual Communication and Image Representation*, vol. 39, pp. 181–195, 2016. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1047320316300888>
- [42] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, “Object detection with discriminatively trained part-based models,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.
- [43] K. Lin, S. Chen, C. Chen, D. Lin, and Y. Hung, “Abandoned object detection via temporal consistency modeling and back-tracing verification for visual surveillance,” *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 7, pp. 1359–1370, 2015.
- [44] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*, vol. 1, 2005, pp. 886–893 vol. 1.
- [45] Y. Tian, R. S. Feris, H. Liu, A. Hampapur, and M. Sun, “Robust detection of abandoned and removed objects in complex surveillance videos,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 41, no. 5, pp. 565–576, 2011.
- [46] S.-C. S. Cheung and C. Kamath, “Robust background subtraction with foreground validation for urban traffic video,” *EURASIP Journal on Advances in Signal Processing*, vol. 2005, no. 14, pp. 1–11, 2005.
- [47] F. El Baf, T. Bouwmans, and B. Vachon, “Comparison of background subtraction methods for a multimedia application,” in *2007 14th International Workshop on Systems, Signals and Image Processing and 6th EURASIP Conference focused on Speech and Image Processing, Multimedia Communications and Services*, 2007, pp. 385–388.
- [48] ———, “A fuzzy approach for background subtraction,” in *2008 15th IEEE International Conference on Image Processing*, 2008, pp. 2648–2651.
- [49] Hanzi Wang and D. Suter, “A re-evaluation of mixture of gaussian background modeling [video signal processing applications],” in *Proceedings. (ICASSP ’05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.*, vol. 2, 2005, pp. ii/1017–ii/1020 Vol. 2.
- [50] M. D. Beynon, D. J. Van Hook, M. Seibert, A. Peacock, and D. Dudgeon, “Detecting abandoned packages in a multi-camera video surveillance system,” in *Proceedings of the IEEE Conference on Advanced Video and Signal Based Surveillance, 2003.*, 2003, pp. 221–228.
- [51] M. Enzweiler and D. M. Gavrila, “Monocular pedestrian detection: Survey and experiments,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 12, pp. 2179–2195, 2009.
- [52] D. Simonnet, S. Velastin, E. Turkbeyler, and J. Orwell, “Backgroundless detection of pedestrians in cluttered conditions based on monocular images: A review,” *Computer Vision, IET*, vol. 6, pp. 540–550, 11 2012.
- [53] P. Dollar, C. Wojek, B. Schiele, and P. Perona, “Pedestrian detection: An evaluation of the state of the art,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 4, pp. 743–761, 2012.

- [54] R. Cutler and L. S. Davis, "Robust real-time periodic motion detection, analysis, and applications," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 781–796, 2000.
- [55] B. Leibe, K. Schindler, and L. Van Gool, "Coupled detection and trajectory estimation for multi-object tracking," in *2007 IEEE 11th International Conference on Computer Vision*, 2007, pp. 1–8.
- [56] I. Parra Alonso, D. Fernandez Llorca, M. A. Sotelo, L. M. Bergasa, P. Revenga de Toro, J. Nuevo, M. Ocana, and M. A. Garcia Garrido, "Combination of feature extraction methods for svm pedestrian detection," *IEEE Transactions on Intelligent Transportation Systems*, vol. 8, no. 2, pp. 292–307, 2007.
- [57] M. D. Breitenstein, F. Reichlin, B. Leibe, E. Koller-Meier, and L. Van Gool, "Online multiperson tracking-by-detection from a single, uncalibrated camera," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 9, pp. 1820–1833, 2011.
- [58] W. Zhang, G. Zelinsky, and D. Samaras, "Real-time accurate object detection using multiple resolutions," in *2007 IEEE 11th International Conference on Computer Vision*, 2007, pp. 1–8.
- [59] H. Sidenbladh, "Detecting human motion with support vector machines," in *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, vol. 2, 2004, pp. 188–191 Vol.2.
- [60] G. Acampora, P. Foggia, A. Saggese, and M. Vento, "A hierarchical neuro-fuzzy architecture for human behavior analysis," *Information Sciences*, vol. 310, pp. 130–148, 2015. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0020025515001863>
- [61] C. Lu, J. Shi, and J. Jia, "Abnormal event detection at 150 fps in matlab," in *2013 IEEE International Conference on Computer Vision*, 2013, pp. 2720–2727.
- [62] V. D. Nguyen, M. T. Le, A. D. Do, H. H. Duong, T. D. Thai, and D. H. Tran, "An efficient camera-based surveillance for fall detection of elderly people," in *2014 9th IEEE Conference on Industrial Electronics and Applications*, 2014, pp. 994–997.
- [63] M. Alvar, A. Torsello, A. Sanchez-Miralles, and J. Armingol, "Abnormal behavior detection using dominant sets," *Machine Vision and Applications*, vol. 25, 07 2014.
- [64] W. Ren, G. Li, B. Sun, and K. Huang, "Unsupervised kernel learning for abnormal events detection," *The Visual Computer*, vol. 31, pp. 245–255, 2013.
- [65] Z. Bian, J. Hou, L. Chau, and N. Magnenat-Thalmann, "Fall detection based on body part tracking using a depth camera," *IEEE Journal of Biomedical and Health Informatics*, vol. 19, no. 2, pp. 430–439, 2015.
- [66] B. Huang, G. Tian, H. Wu, and F. Zhou, "A method of abnormal habits recognition in intelligent space," *Engineering Applications of Artificial Intelligence*, vol. 29, pp. 125–133, 2014. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0952197613002443>
- [67] D. Tran, Thi-Lan Le, and Thi-Thanh-Hai Tran, "Abnormal event detection using multimedia information for monitoring system," in *2014 IEEE Fifth International Conference on Communications and Electronics (ICCE)*, 2014, pp. 490–495.
- [68] H. F. Gómez A., R. M. Tomás, S. A. Tapia, A. F. Caballero, S. Ratté, A. G. Eras, and P. L. González, "Identification of loitering human behaviour in video surveillance environments," in *Artificial Computation in Biology and Medicine*, J. M. Ferrández Vicente, J. R. Álvarez-Sánchez, F. de la Paz López, F. J. Toledo-Moreo, and H. Adeli, Eds. Cham: Springer International Publishing, 2015, pp. 516–525.
- [69] J. Ko and J. Yoo, "Rectified trajectory analysis based abnormal loitering detection for video surveillance," in *2013 1st International Conference on Artificial Intelligence, Modelling and Simulation*, 2013, pp. 289–293.
- [70] R. Mehran, A. Oyama, and M. Shah, "Abnormal crowd behavior detection using social force model," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 935–942.

- [71] S. Cho and H. Kang, "Integrated multiple behavior models for abnormal crowd behavior detection," in *2012 IEEE Southwest Symposium on Image Analysis and Interpretation*, 2012, pp. 113–116.
- [72] S.-H. Cho and H.-B. Kang, "Abnormal behavior detection using hybrid agents in crowded scenes," *Pattern Recognition Letters*, vol. 44, pp. 64–70, 2014, pattern Recognition and Crowd Analysis. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167865513004613>
- [73] G. Santhiya, K. Sankaragomathi, S. Selvarani, and A. N. Kumar, "Abnormal crowd tracking and motion analysis," in *2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies*, 2014, pp. 1300–1304.
- [74] M. Leach, R. Baxter, N. Robertson, and E. Sparks, "Detecting social groups in crowded surveillance videos using visual attention," in *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2014, pp. 467–473.
- [75] T. Hassner, Y. Itcher, and O. Kliper-Gross, "Violent flows: Real-time detection of violent crowd behavior," in *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2012, pp. 1–6.
- [76] R. Chaker, Z. A. Aghbari, and I. N. Junejo, "Social network model for crowd anomaly detection and localization," *Pattern Recognition*, vol. 61, pp. 266–281, 2017. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0031320316301327>
- [77] G. Wang, H. Fu, and Y. Liu, "Real time abnormal crowd behavior detection based on adjacent flow location estimation," in *2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS)*, 2016, pp. 476–479.
- [78] L. Shen, L. Margolies, J. Rothstein, E. Fluder, R. McBride, and W. Sieh, "Deep learning to improve breast cancer detection on screening mammography," *Scientific Reports*, vol. 9, pp. 1–12, 08 2019.
- [79] I. Moreira, I. Amaral, I. Domingues, A. Cardoso, M. Cardoso, and J. Cardoso, "Inbreast: Toward a full-field digital mammographic database," *Academic radiology*, vol. 19, pp. 236–48, 11 2011.
- [80] R. Girshick, "Fast r-cnn," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448.
- [81] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," 2015.
- [82] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 2018.
- [83] L. Leal-Taixé, A. Milan, I. Reid, S. Roth, and K. Schindler, "Motchallenge 2015: Towards a benchmark for multi-target tracking," 2015.
- [84] A. Milan, L. Leal-Taixe, I. Reid, S. Roth, and K. Schindler, "Mot16: A benchmark for multi-object tracking," 2016.
- [85] L. Zheng, Z. Bie, Y. Sun, J. Wang, C. Su, S. Wang, and Q. Tian, "Mars: A video benchmark for large-scale person re-identification," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 868–884.
- [86] "Acceso al repositorio de github del yolov4 con darknet," <https://github.com/AlexeyAB/darknet> [Último acceso 22/febrero/2021].
- [87] "Acceso a google colaboratory," <https://colab.research.google.com/> [Último acceso 28/marzo/2021].
- [88] "Acceso al repositorio de github del yolov4 con tensorflow original," <https://cutt.ly/0mPNzV7> [Último acceso 06/septiembre/2020].
- [89] "Acceso al repositorio de github del yolov4 con tensorflow," <https://cutt.ly/pxU1zxr> [Último acceso 15/noviembre/2021].
- [90] J. Redmon, "Darknet: Open source neural networks in c," <http://pjreddie.com/darknet/>, 2013–2016.

- [91] “Acceso al repositorio de github del yolov4 deep sort original,” <https://github.com/theAIGuysCode/yolov4-deepsoft> [Último acceso 11/enero/2021].
- [92] “Acceso al repositorio de github del yolov4 deep sort,” <https://github.com/jmudy/yolov4-deepsoft> [Último acceso 20/marzo/2021].
- [93] D. Valdivieso López, “Design, implementation and evaluation of automated surveillance systems,” <http://hdl.handle.net/10017/37872>, 2018.
- [94] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, “Scaled-yolov4: Scaling cross stage partial network,” 2021.
- [95] Y. Zhang, C. Wang, X. Wang, W. Zeng, and W. Liu, “Fairmot: On the fairness of detection and re-identification in multiple object tracking,” *arXiv preprint arXiv:2004.01888*, 2020.
- [96] H. Park, S. Park, and Y. Joo, “Detection of abandoned and stolen objects based on dual background model and mask r-cnn,” *IEEE Access*, vol. 8, pp. 80 010–80 019, 2020.
- [97] N. S. Punn, S. K. Sonbhadra, and S. Agarwal, “Monitoring covid-19 social distancing with person detection and tracking via fine-tuned yolo v3 and deepsort techniques,” 2020.
- [98] M. Rezaei and M. Azarmi, “Deepsocial: Social distancing monitoring and infection risk assessment in covid-19 pandemic,” *Applied Sciences*, vol. 10, no. 21, p. 7514, Oct 2020. [Online]. Available: <http://dx.doi.org/10.3390/app10217514>
- [99] S. Gupta, R. Kapil, G. Kanahasabai, S. S. Joshi, and A. S. Joshi, “Sd-measure: A social distancing detector,” *2020 12th International Conference on Computational Intelligence and Communication Networks (CICN)*, Sep 2020. [Online]. Available: <http://dx.doi.org/10.1109/CICN49253.2020.9242628>
- [100] T. Fan, Z. Chen, X. Zhao, J. Liang, C. Shen, D. Manocha, J. Pan, and W. Zhang, “Autonomous social distancing in urban environments using a quadruped robot,” 2020.
- [101] “Nvidia cuda gpus compute capability,” <https://developer.nvidia.com/cuda-gpus> [Último acceso 21/noviembre/2020].

Apéndice A

Pliego de condiciones

A.1. Introducción

En este anexo se especifican el *software* y *hardware* empleados en el desarrollo del proyecto. Por un lado está el equipo A, que se ha empleado únicamente para la redacción del presente documento con L^AT_EX. Por otro lado está el equipo B, cuya finalidad ha sido poder programar y evaluar los algoritmos con mejor capacidad de computación dadas sus especificaciones.

A.2. Características del equipo A

A continuación, se detallan las especificaciones *software* y *hardware* que tiene el primer equipo. Cabe destacar que, las especificaciones necesarias pueden ser menores a las indicadas, ya que únicamente se ha utilizado para redactar la presente memoria sobre un sistema operativo Windows 10.

A.2.1. Especificaciones hardware del equipo A

- Procesador: Intel® Core™ i5-3570K @ 3.4Ghz Box
- Tarjeta gráfica: NVIDIA® Gigabyte GeForce GTX™ 770 OC 2GB GDDR5
- Memoria intalada (RAM): G.Skill Ares DDR3 1600 PC3-12800 8GB 2x4GB CL9
- Almacenamiento 1: Samsung 860 EVO Basic SSD 500GB SATA3
- Almacenamiento 2: WD Blue 1TB SATA3

A.2.2. Especificaciones software del equipo A

- Utilización de un sistema operativo de 64 bits, Windows 10 Pro (compilación de SO 19042.804)
- Lenguaje de programación R 3.6.3, recomendable utilizar RStudio 1.3.1093
- Procesador de textos L^AT_EX con TexMaker 5.0.4 y MiKTeX-TeX 4.0 (MiKTeX 20.12)
- Software de control de versiones Git version 2.29.2.windows.1

A.3. Características del equipo B

Para la correcta evaluación de los algoritmos programados se recomienda que las especificaciones del equipo sean igual o superiores a las detalladas. Tal y como se indica en la sección C.2.4 del Apéndice C, es obligatorio disponer de una tarjeta NVIDIA con una capacidad de cálculo mayor a 3,5 para poder utilizar CUDA en Tensorflow en su versión 2.3.0rc0.

A.3.1. Especificaciones hardware del equipo B

- Procesador: Intel® Core™ i7-6700HQ @ 2.60 GHz
- Tarjeta gráfica: NVIDIA® GeForce GTX™ 960M 2GB GDDR5
- Memoria instalada (RAM): 20GB
- Almacenamiento 1: Disco duro SSD 250GB
- Almacenamiento 2: Disco duro HDD 1TB

A.3.2. Especificaciones software del equipo B

- Utilización de un sistema operativo de 64 bits, Ubuntu 18.04.4 LTS
- Entorno de trabajo Anaconda 4.9.1
- NVIDIA® CUDA™ Toolkit 10.1.243
- CUDA Deep Neural Network (cuDNN) v7.6.5 (November 5th, 2019), for CUDA™ 10.1
- Lenguaje de programación Python™ 3.7.0, recomendable utilizar Visual Studio Code como editor de código fuente

Cabe resaltar que el motivo de la utilización de dos equipos ha sido únicamente por comodidad. Para el desarrollo del proyecto se puede emplear un único equipo que tenga las mismas prestaciones o superiores al equipo B. Se ha empleado un equipo con Windows 10 ya que ha facilitado el manejo de datos con Microsoft Excel.

Apéndice B

Presupuesto

B.1. Introducción

En este apéndice se muestra el presupuesto del proyecto. Para ello, en los siguientes apartados se detallarán el perfil de personal necesario para desarrollarlo, la duración estimada de cada una de las etapas que lo compone, los recursos *hardware* y *software* utilizados, el coste de la mano de obra y finalmente el coste total.

B.2. Equipo de trabajo

Para la realización del proyecto se necesita tener a un Ingeniero de Telecomunicaciones, Informático o Industrial especializado en percepción, altamente familiarizado con el uso de redes neuronales, en concreto, redes neuronales convolucionales, con gran dominio de la librería OpenCV y gran manejo en el lenguaje de programación PythonTM.

B.3. Timing

Para calcular el presupuesto del proyecto es imprescindible conocer la duración del proyecto para tener en cuenta los costes debidos a la mano de obra.

Las fases necesarias para la realización del proyecto son las siguientes.

- Formación inicial y revisión del Estado del Arte (1 mes)
 - Recopilar información sobre el Estado del Arte: algoritmos, sistemas y datasets. (0,25 meses)
 - Comparación entre diferentes estrategias de detección de objetos. (0,25 meses)
 - Buscar implementaciones disponibles en el alcance del proyecto (0,25 meses)
 - Estudiar las redes neuronales convolucionales ([CNN](#)) para identificar objetos en imágenes (0,25 meses)
- Diseño, implementación y evaluación de algoritmos y librerías en la detección de personas y objetos (0,5 meses)
 - Diseño, implementación/adaptación de la estrategia seleccionada para esta tarea (0,2 meses)
 - Refinar el Estado del Arte (0,1 meses)
 - Evaluación rigurosa de los algoritmos desarrollados en datasets relevantes (0,2 meses)
- Diseño, implementación y evaluación de algoritmos y librerías en la seguimiento/rastreo de personas y objetos (1,5 meses)
 - Diseño, implementación/adaptación de la estrategia seleccionada para esta tarea (0,7 meses)

- Refinar el Estado del Arte (0,2 meses)
- Evaluación rigurosa de los algoritmos desarrollados en datasets relevantes (0,6 meses)
- Diseño, implementación y evaluación de algoritmos y librerías en la tarea de detección de objetos abandonados (1,5 meses):
 - Diseño, implementación/adaptación de la estrategia seleccionada para esta tarea (0,7 meses)
 - Refinar el Estado del Arte (0,2 meses)
 - Evaluación rigurosa de los algoritmos desarrollados en datasets relevantes (0,6 meses)
- Evaluación de la integración final de los algoritmos desarrollados en los datasets más relevantes (0,5 meses)
- Corrección de errores y comienzo de la redacción de la memoria del proyecto (1 mes)
- Finalización de la redacción de la memoria del proyecto (0,5 meses)

Cabe indicar que ciertas tareas como la de refinar y recopilar información el Estado del Arte, la comparación de diferentes estrategias de detección y rastreo o la evaluación de los algoritmos en los distintos datasets se repiten a lo largo de los 6,5 meses de duración del desarrollo del proyecto.

B.4. Costes

En los siguientes subapartados se va a calcular los costes asociados a la mano de obra necesaria en el desarrollo de cada una de las tareas descritas en la sección B.3. También se tendrá en cuenta los costes asociados a los materiales *hardware* y *software* que componen los dos equipos utilizados para el desarrollo del proyecto. Por último se calculará el presupuesto total del proyecto teniendo en cuenta todo lo nombrado anteriormente.

B.4.1. Costes mano de obra

La tabla B.1 presenta el desglose de las tareas detalladas en la sección B.3 para calcular el coste de mano de obra que va a llevar cada una de las etapas.

Tabla B.1: Costes de mano de obra

Concepto	Horas	Coste Unitario ($\text{€}/\text{h}$)	Coste total (€)
Formación inicial y revisión del Estado del Arte	80	20	1.600
D + I + E algoritmos detección objetos	40	20	800
D + I + E algoritmos seguimiento objetos	120	20	2.400
Desarrollo algoritmo detección objetos abandonados	120	20	2.400
Evaluación algoritmos desarrollados en datasets	40	20	800
Corrección errores y comenzar a redactar la memoria	80	20	1.600
Finalizar redacción de la memoria	40	20	800
TOTAL	520		10.400 €

B.4.2. Recursos hardware

Los costes debidos a los recursos *hardware* utilizados en el equipo A y el equipo B se visualizan en la tabla B.2.

Tabla B.2: Recursos hardware

Concepto	Unidades	Coste Unitario (€)	Coste total (€)
Ordenador sobremesa equipo A	1	1.100	1.100
Periféricos	1	250	250
Monitor BenQ ZOWIE XL2411P	1	199	199
Ordenador portátil equipo B	1	1.150	1.150
Monitor BenQ GL2480	1	129	129
TOTAL			2.828 €

B.4.3. Recursos software

Los costes debidos a los recursos *software* utilizados en el equipo A y el equipo B se visualizan en la tabla B.3.

Tabla B.3: Recursos software

Concepto	Unidades	Coste Unitario (€)	Coste total (€)
Windows 10 Pro	1	259	259
Microsoft Office 365	1	49	49
TexMaker + MiKTeX-TeX	1	0	0
Git	1	0	0
Ubuntu 18.04.4 LTS	1	0	0
Anaconda Distribution	1	0	0
Python 3.7.0	1	0	0
R 3.6.3	1	0	0
TOTAL			308 €

B.5. Presupuesto total

La tabla B.4 recoge los costes totales del proyecto:

Tabla B.4: Presupuesto total

Concepto	Coste total (€)
Coste mano de obra	10.400
Recursos hardware	2.828
Recursos software	308
TOTAL	13.536 €

El presupuesto total del proyecto asciende a trece mil quinientos treinta y seis euros, IVA no incluido.

Apéndice C

Manual de usuario

C.1. Introducción

Este apéndice se va a dividir en dos secciones diferenciadas donde por un lado, se va a explicar como instalar y configurar todas las herramientas necesarias para la puesta a punto del proyecto. Y por otro lado, se va a proporcionar un manual de usuario donde se detalla como ejecutar cada uno de los algoritmos que se han utilizado en el capítulo 3. Cabe destacar que, tal y como se indica en el sección A.1, el equipo donde se instalará todo el software descrito en este apartado para programar los distintos algoritmos, tiene un sistema operativo Ubuntu 18.04.4 LTS. Por tanto, todos los comandos de instalaciones y puesta en funcionamiento estarán orientados a un equipo que disponga un sistema operativo Linux.

C.2. Guía de instalación

C.2.1. Instalación de Git

Git es una herramienta de control de versiones distribuido de código que trabaja de una manera muy rápida y potente. Tiene un sistema para trabajar mediante ramas que pueden seguir una línea de progreso diferente a la principal, de tal manera que se pueden realizar pruebas del código o que distintas personas trabajen en ramas paralelas y posteriormente, en una versión final de la implementación, se puedan incluir en la rama principal. Para proyectos como el presente, es necesario tener disponible un historial completo de versiones para su correcto desarrollo.

Para poder instalar Git abra el terminal y ejecute los siguientes comandos:

```
1 # Actualizar paquetes de los repositorios
2 sudo apt-get update
3
4 # Instalacion de Git con todas sus dependencias
5 sudo apt-get install git-all
```

Código C.1: Instalación de Git

C.2.2. Instalación de Anaconda

Anaconda se trata de la *suite* más compleja para la Ciencia de Datos en R y Python, lenguajes de programación que actualmente son líderes en Machine Learning, Inteligencia Artificial y Big Data. Esta *suite* gratuita y multiplataforma dispone de las [Integrated Development Environment \(IDE\)](#)'s y librerías adecuadas para su manejo. Con Anaconda se evita tener que instalar manualmente un [IDE](#) y el lenguaje Python, con sus correspondientes librerías, que en ocasiones puede ser una operación tediosa y compleja.

1. Accede al siguiente enlace [28] disponible en la bibliografía y descargue la versión más reciente del instalador de Anaconda para Linux:



Figura C.1: Descarga del instalador de Anaconda [28]

- Abra el terminal y acceda a la carpeta `Downloads` donde ha descargado el instalador. Es recomendable verificar la integridad del instalador mediante la suma de comprobación SHA-256:

```

1 # Acceder a la carpeta Downloads del usuario
2 cd ${HOME}/Downloads
3
4 # Verificacion de la integridad del instalador
5 sha256sum Anaconda3-2020.02-Linux-x86_64.sh

```

Código C.2: Verificación de la integridad de la instalación de Anaconda

- Ejecute la secuencia de comandos de Anaconda. Presione `yes` para aceptar los términos de la licencia. Pulse a continuación `ENTER` cuando seleccione la ubicación de la instalación y finalmente presione `yes` para confirmar el PATH de Anaconda con tu fichero `.bashrc`:

```

1 # Ejecutar el instalador de Anaconda para Linux
2 bash Anaconda3-2020.02-Linux-x86_64.sh

```

Código C.3: Ejecutar el instalador de Anaconda para Linux

- Cuando se complete la instalación cierre y abra el terminal de nuevo, o bien ejecute el siguiente comando:

```

1 # Hacer efectivos los cambios realizados en .bashrc
2 source ${HOME}/.bashrc

```

Código C.4: Hacer efectivo los cambios en el fichero `.bashrc`

C.2.3. Descarga de los repositorios del proyecto

En esta sección se indica como descargar los repositorios de GitHub donde se han programado los algoritmos de detección y seguimiento de objetos abandonados. Para ello, se debe de ejecutar los comandos que se muestran a continuación:

```

1 # Acceder a la carpeta Documents del usuario
2 cd ${HOME}/Documents
3
4 # Clonar los dos repositorios de GitHub
5 git clone https://github.com/jmudy/tensorflow-yolov4-tflite
6 git clone https://github.com/jmudy/yolov4-deepsort.git
7
8 # Acceder al repositorio de detección de objetos con YOLOv4
9 cd tensorflow-yolov4-tflite
10
11 # O bien al de seguimiento y detección de objetos abandonados con YOLOv4 y Deep SORT
12 cd yolov4-deepsort

```

Código C.5: Descarga repositorio

C.2.4. Crear entorno virtual con Anaconda

Con la finalidad de evitar la larga y compleja instalación de **CUDA** y **cuDNN** y tener únicamente instaladas las librerías necesarias para que funcione el código, se va a instalar un entorno virtual en Anaconda.

Desde cualquiera de las carpetas de los repositorios de Github que se han clonado, hay dos ficheros con extensión .yml, donde viene indicadas la versión de Python que se va a necesitar, la versión de **CUDA** y **cuDNN** así como las librerías y dependencias necesarias. Para utilizar **CUDA** es necesario disponer de una **GPU** de NVIDIA. Es muy importante verificar la capacidad de cálculo de la tarjeta gráfica NVIDIA que se emplee. En la siguiente dirección [101] se puede comprobar la capacidad de cálculo de los distintos modelos o bien puede consultar la que esté utilizando introduciendo los siguientes comandos en el terminal en el caso de que ya se tuviera instalada la librería de Tensorflow:

```

1 # Iniciar Python en terminal
2 python
3
4 # Importar libreria tensorflow
5 import tensorflow as tf
6
7 # Test sobre el dispositivo GPU que se este empleando
8 tf.test.gpu_device_name()

```

Código C.6: Comprobar capacidad computación de la GPU

La capacidad de cálculo mínima para poder utilizar **CUDA** junto a la librería tensorflow-gpu es de 3.5. En función de la **GPU** que se disponga, ejecute uno de los siguientes comandos para crear un entorno virtual de Anaconda con Python 3.7.0:

```

1 # Si tu GPU tiene una capacidad de calculo < 3.5
2 conda env create -f conda-cpu.yml
3
4 # Si tu GPU tiene una capacidad de calculo >= a 3.5
5 conda env create -f conda-gpu.yml

```

Código C.7: Creación entorno virtual en Anaconda

Una vez instalado el entorno virtual se puede activar ejecutando el siguiente comando:

```

1 # Si instalaste el entorno con conda-cpu.yml
2 conda activate yolov4-cpu
3
4 # Si instalaste el entorno con conda-gpu.yml
5 conda activate yolov4-gpu

```

Código C.8: Activar entorno virtual de Anaconda

C.2.5. Descargar los datasets

Los enlaces para descargar los datasets que se han utilizado para evaluar los distintos algoritmos a lo largo del proyecto se encuentran a continuación:

- **PETS2007** dataset: <http://www.cvg.reading.ac.uk/PETS2007/data.html> [5]
- **AVSSAB2007** dataset: http://www.eecs.qmul.ac.uk/~andrea/avss2007_d.html [6]
- **GBA2018** dataset: <https://bit.ly/3tIJdUx> [4]
- **ABODA** dataset: <https://github.com/kevinlin311tw/ABODA> [7]
- **MS COCO** dataset: <https://cocodataset.org/#download> [3]
- **OIDv4** dataset: <https://storage.googleapis.com/openimages/web/index.html> [9]

C.3. Guía de ejecución

En esta parte se especifica los comandos que se deben ejecutar para poner en funcionamiento el algoritmo de detección de objetos, el algoritmo de seguimiento de personas y objetos y por último, el algoritmo de detección de objetos abandonados. Cabe destacar que el repositorio con nombre `yolov4-deepsort` se utiliza tanto para ejecutar el script de seguimiento con **Deep SORT** como para ejecutar el algoritmo de detección de objetos abandonados.

C.3.1. Ejecutar algoritmo de detección de objetos YOLOv4

Para ejecutar el algoritmo de detección de objetos con **YOLOv4** en el framework Tensorflow entre en la carpeta del repositorio `tensorflow-yolov4-tflite`, descargue y convierta el modelo Darknet a Tensorflow. Este paso solo será necesario realizarlo una vez. Para ello, abra el terminal y ejecute los siguientes comandos:

```

1 # Entrar dentro de la carpeta del repositorio
2 cd ${HOME}/Documents/tensorflow-yolov4-tflite
3
4 # Descargar los pesos de YOLOv4
5 wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.
     weights -P ./data/
6
7 # Convertir modelo Darknet de YOLOv4 a modelo Tensorflow
8 python save_model.py --weights ./data/yolov4.weights --output ./checkpoints/yolov4-608 --
     input_size 608 --model yolov4

```

Código C.9: Descarga de pesos y conversión modelo YOLOv4

Por último, introduzca el vídeo que se quiera analizar dentro de la carpeta del repositorio. Se recomienda introducir los vídeos a analizar en la carpeta `./data/video/` y los vídeos generados en la carpeta `./detections`.

```

1 # Cambiar la ruta donde se encuentre el video que se quiera analizar y la ruta donde se
2 # quiere guardar el video que se genere
3 python detect_video.py --weights ./checkpoints/yolov4-608 --size 608 --model yolov4 --video
     {path_to_input_video} --output {path_to_output_video}

```

Código C.10: Ejecutar script detección de objetos con YOLOv4 en Tensorflow

Si no se dispone del equipo necesario para instalar todas las dependencias necesarias se recomienda acceder al siguiente [link](#) donde se facilita un Notebook de Google Colab para ejecutar exactamente los mismos comandos desde el servicio cloud de Google.

Recuerde que, tal y como se explicó en la sección [3.2](#), también se pueden modificar parámetros como el tamaño del resize, el umbral del **IoU** o el umbral de confianza añadiendo los flags que sean pertinentes.

C.3.2. Ejecutar algoritmo de seguimiento de objetos YOLOv4 + Deep SORT

Para ejecutar el algoritmo de seguimiento de objetos con **Deep SORT** y **YOLOv4** entre en la carpeta del repositorio `yolov4-deepsort` desde el terminal y cambie a la rama de git `develop` dedicada al rastreo de objetos y personas ejecutando los siguientes comandos:

```

1 # Entrar dentro de la carpeta del repositorio
2 cd ${HOME}/Documents/yolov4-deepsort
3
4 # Cambiar a la rama develop
5 git checkout develop
6
7 # Para confirmar que nos encontramos en la rama develop
8 git branch -a

```

Código C.11: cambiar a la rama develop

Posteriormente, descargue y convierta el modelo Darknet a Tensorflow de igual manera que se hizo el script C.9 en la sección C.3.1. Este paso solo será necesario realizarlo una vez. Una vez se tenga el modelo de YOLOv4 convertido, introduzca el vídeo que se quiera analizar dentro de la carpeta del repositorio. Se recomienda introducir los vídeos a analizar en la carpeta ./data/video/ y los vídeos generados en la carpeta ./outputs.

Por último, ejecutar el script de seguimiento de personas y objetos introduciendo el siguiente comando en el terminal:

```
1 # Cambiar la ruta donde se encuentre el video que se quiera analizar y la ruta donde se
2 # quiere guardar el video que se genere
3 python object_tracker.py --weights ./checkpoints/yolov4-608 --size 608 --model yolov4 --
video {path_to_input_video} --output {path_to_output_video}
```

Código C.12: Ejecutar script seguimiento de personas y objetos con DeepSORT

Si no se dispone del equipo necesario para instalar todas las dependencias necesarias se recomienda acceder al siguiente [link](#) donde se facilita un Notebook de Google Colab para ejecutar exactamente los mismos comandos desde el servicio cloud de Google.

Recuerde que, tal y como se explicó en la sección 3.2, también se pueden modificar parámetros como el tamaño del resize, el umbral del IoU o el umbral de confianza añadiendo los flags que sean pertinentes.

C.3.3. Ejecutar algoritmo de detección de objetos abandonados

Para ejecutar el algoritmo de detección de objetos abandonados con Deep SORT y YOLOv4 entre en la carpeta del repositorio yolov4-deepsort desde el terminal y cambie a la rama de git abandoned dedicada al rastreo de objetos y personas ejecutando los siguientes comandos:

```
1 # Entrar dentro de la carpeta del repositorio
2 cd ${HOME}/Documents/yolov4-deepsort
3
4 # Cambiar a la rama abandoned
5 git checkout abandoned
6
7 # Para confirmar que nos encontramos en la rama abandoned
8 git branch -a
```

Código C.13: cambiar a la rama abandoned

Posteriormente, descargue y convierta el modelo Darknet a Tensorflow de igual manera que se hizo el script C.9 de la sección C.3.1. Este paso solo será necesario realizarlo una vez. Una vez se tenga el modelo de YOLOv4 convertido, introduzca el vídeo que se quiera analizar dentro de la carpeta del repositorio. Se recomienda introducir los vídeos a analizar en la carpeta ./data/video/ y los vídeos generados en la carpeta ./outputs.

Por último, ejecutar el script de detección de objetos abandonados introduciendo el siguiente comando en el terminal:

```
1 # Cambiar la ruta donde se encuentre el video que se quiera analizar y la ruta donde se
2 # quiere guardar el video que se genere
3 python abandoned_object.py --weights ./checkpoints/yolov4-608 --size 608 --model yolov4 --
video {path_to_input_video} --output {path_to_output_video}
```

Código C.14: Ejecutar script detección de objetos abandonados con YOLOv4 y Deep SORT

Si no se dispone del equipo necesario para instalar todas las dependencias necesarias se recomienda acceder al siguiente [link](#) donde se facilita un Notebook de Google Colab para ejecutar exactamente los mismos comandos desde el servicio cloud de Google.

Recuerde que, tal y como se explicó en la sección 3.2, también se pueden modificar parámetros como el tamaño del resize, el umbral del IoU o el umbral de confianza añadiendo los flags que sean pertinentes.

Universidad de Alcalá
Escuela Politécnica Superior



ESCUELA POLITECNICA
SUPERIOR

