Universidad Internacional de La Rioja (UNIR)

ESIT

Máster Universitario en Inteligencia Artificial

Comparativa y entrenamiento de modelos de Smart Mobility: detección y alerta temprana de ciclistas.

Trabajo Fin de Máster

Presentado por: Fernández Alonso, Juan Carlos

Director/a: Tejeda Lorente, Álvaro

Ciudad: Madrid
Fecha: 22 de septiembre 2021

Resumen

Este proyecto se enfocará en el ámbito de la Movilidad Inteligente y, más concretamente, en la detección temprana de objetos en la vía. En primer lugar, se realizará una breve descripción del estado del arte de dicha línea de trabajo, en el que se revisarán los modelos y técnicas de Inteligencia Artificial más ampliamente utilizados en los últimos años. Posteriormente, se abordará un estudio comparativo en el que se elegirán cinco de los modelos de Aprendizaje Profundo de uso más extendido en esta área. La comparativa dará como ganador un modelo en concreto, bajo determinadas figuras de mérito tanto en términos de efectividad del modelo como de tiempo de inferencia. A continuación, se utilizará dicho modelo ganador como base para la construcción de uno nuevo utilizando técnicas de Transferencia de Aprendizaje para un caso de uso concreto: detección temprana de ciclistas en la vía. A modo de conclusión del proyecto y con el fin de verificar y validar el modelo desarrollado, se construye un Mínimo Producto Viable consistente en la detección temprana de ciclistas y envío de alertas. Finalmente se realiza una demostración de su funcionalidad y se proponen líneas de trabajo futuras para su evolución y productivización.

Palabras Clave: Movilidad Inteligente, Aprendizaje Profundo, Redes Neuronales Convolucionales, Detección de objetos, Ciclista.

Abstract

This project is focused on the field of Smart Mobility, and more specifically, on early object detection on the road. First, the state of the art to which this work belongs will be briefly described, and the most relevant models and techniques over the last years will be reviewed. Subsequently, a comparative study will be carried out, by evaluating five of the most widely used Deep Learning models in this area. As a result, a particular model will be selected as the best performing candidate, regarding certain figures of merit such as effectiveness and inference time. Right afterwards, we will rely on this model in order to build a new model, by means of Transfer Learning techniques, for a specific use-case: early cyclist detection on the road. As a project closure, and in order to verify and validate the developed model, a Minimum Viable Product consisting of the early cyclist detection and alerts delivery is built. Finally, a demonstration of its functionality is performed and future lines of work are proposed for its enhancement and productivization.

Keywords: Smart Mobility, Deep Learning, Convolutional Neural Networks, Object Detection, Cyclist.

Índice de contenidos

Índice de figuras

1. Introducción

“En los últimos 10 años, los accidentes de tráfico con implicación de bicicletas han pasado de 340 en 2010 a 921 en 2020, un aumento del 270%, según los datos disponibles en el portal del Ayuntamiento de Madrid” (Gallello, 2021).

Este dato y el auge que está tomando la bicicleta como medio de transporte o de ocio en los últimos años hace más que preocupante la seguridad e integridad de las personas que utilizan la bici a diario.

En este trabajo se aborda esta preocupación para proponer una posible solución, construida mediante técnicas y herramientas de Inteligencia Artificial, que ayude a mejorar y aumentar la seguridad de los ciclistas, así como disminuir el número de accidentes, de heridos y de fallecidos para poder continuar promoviendo el uso de este medio de transporte tan beneficioso, tanto para el usuario como para el medioambiente.

1.1 Motivación

En el presente Trabajo Fin de Máster se toma en cuenta esta problemática para encontrar, desde el punto de vista de la Inteligencia Artificial, una manera de ayudar a visibilizar a los ciclistas por parte del resto de vehículos con los que comparten carreteras y trayectos.

Actualmente, ya hay a disposición de los conductores distintas herramientas de movilidad inteligente y conectada, de forma que un conductor puede seleccionar un trayecto a realizar, y durante el mismo, en tiempo real, ver en pantalla si hay retenciones o atascos, accidentes, coches averiados, radares, etc (Acosta, 2019). Esto ayuda al conductor a identificar el punto concreto del mapa donde aparece esa alerta, permitiendo en cada caso aumentar la atención, disminuir la velocidad, o cualquier otra maniobra que se considere oportuna para maximizar la seguridad tanto del propio conductor como del resto de usuarios de la vía.

Sin embargo, no hay ninguna que alerte de la presencia de ciclistas en la vía, cuando en realidad este hecho también requiere de atención por parte de los conductores: aminorar velocidad, comprobar que no vienen vehículos en sentido contrario, adelantar con suficiente distancia, etc.

Es en este punto donde surge la idea de implementar un sistema de visión artificial capaz de detectar un ciclista en tiempo real, y enviar la notificación correspondiente de forma automática para que los conductores que vienen detrás puedan recibir esa alerta y haya un conocimiento global de qué carreteras están más frecuentadas y transitadas por ciclistas, y en qué momentos y días.

Con esto, se conseguiría mejorar la seguridad del propio ciclista, al mismo tiempo que la seguridad al volante en el uso de un sistema de alertas que, a diferencia de los que ya implementan algunas aplicaciones, serían automáticas y no realizadas de forma manual por el propio conductor.

Esta herramienta de alertas de ciclistas automáticas, finalmente, se podría llegar a generalizar, en un futuro, a alertas de otra índole (accidentes, retenciones, coches averiados o parados en la carretera, …) y conseguir así un sistema de alertas que cubra la detección de todo tipo de obstáculos que pueda haber en un trayecto, de forma automática.

1.2 Planteamiento del trabajo

Para ello, se abordará dicho problema de detección y alerta de ciclistas, tanto individuales como grupos pequeños, hasta grandes pelotones, con la posterior notificación en una aplicación de movilidad para que los vehículos que se acerquen a la zona con posterioridad sean conscientes de que circulan por una vía transitada por algún (o muchos) ciclistas y puedan extremar la precaución de la forma que consideren más oportuna.

Con estos avisos automáticos mediante la detección de los ciclistas con técnicas de visión artificial no solo se mejora la seguridad de los conductores, pues no tienen que notificar manualmente la presencia de ciclistas en la vía, si no la de los propios ciclistas, puesto que evidencia su presencia al resto de conductores.

Para ello se realizará un estudio de los modelos de visión artificial y reconocimiento de objetos más utilizados en el ámbito de movilidad inteligente. Posteriormente, se escogerá el que mejor desempeño tenga, en relación al objetivo de la detección de ciclistas, y se intentará mejorar para el caso de uso concreto que se trata en este trabajo con las técnicas estudiadas a lo largo del presente Máster en Inteligencia Artificial.

1.3 Estructura de la memoria

La presente memoria trata en detalle todo el estudio previo y el desarrollo posterior realizado en la dirección del producto final deseado. En este sentido, los sucesivos apartados introducen al lector en cada una de las etapas que se ha seguido, desde una contextualización y situación en el estado del arte de las áreas de la Inteligencia Artificial que se van a emplear, pasando por un estudio de los modelos más utilizados en *Smart Mobility* en la actualidad, hasta finalmente la elección de uno de ellos y su reentrenamiento, obteniendo finalmente el producto buscado:

- El apartado *2. Contexto y estado del arte* expone en detalle el problema propuesto, realiza un repaso de las áreas de la Inteligencia Artificial que se emplean para llevar a cabo el desarrollo de la solución a dicho problema, y sitúa y contextualiza al lector en las técnicas y modelos empleados en la actualidad de forma más extendida en lo referente a *Smart Mobility*, o Movilidad Inteligente o Conectada.

- El apartado *3. Objetivos y metodología de trabajo* plantea la hoja de ruta que se seguirá para el desarrollo de la solución buscada, detallando así las tareas a realizar, orden, y tiempos.

- El apartado *4. Identificación de requisitos* propone una manera de cumplir y disponer, tanto a nivel de hardware como de software, de lo necesario para llevar a cabo las tareas mencionadas en el apartado anterior.

- El apartado *5. Descripción de la herramienta de software desarrollada* explica el grueso del proceso analítico llevado a cabo en la puesta a punto del entorno, la comparación y evaluación de los modelos, la recopilación de los datos utilizados, y la decisión del modelo ganador y su reentrenamiento.

- El apartado *6. Evaluación* muestra el producto final obtenido tras todo el trabajo realizado, en forma de Mínimo Producto Viable.

- El apartado *7. Conclusiones y trabajo futuros* comenta, a modo de cierre, la experiencia y aprendizaje obtenido tras todo el trabajo realizado, así como siguientes mejoras y evoluciones del MVP propuesto.

2. Contexto y estado del arte

En este Trabajo Fin de Máster confluyen y se trabajan, principalmente, dos potentes ámbitos de estudio:

- La Percepción Computacional
- El Aprendizaje Profundo

En este capítulo se realiza una breve explicación, a modo repaso, de las tareas que abarca cada ámbito. Se presentan algunos de los problemas más representativos de la Visión Artificial, y se recorre la historia y se contextualiza como éstos dieron lugar al nacimiento del *Deep Learning*, o Aprendizaje Profundo. Por último, se habla de cómo éste ha ido obteniendo reconocimiento y popularidad y se introducen los conceptos de Redes Neuronales Convolucionales, Transferencia del Aprendizaje (o *Transfer Learning*) y Aumento de Datos (o *Data Augmentation*).

2.1. Introducción a la Percepción Computacional

Uno de los campos de estudio e investigación más populares de la IA es la Percepción Computacional, y más en concreto la Visión Artificial (o Visión por Computador, *Computer Vision*).

El ámbito de la Percepción Computacional (Partida, Manrique y Barrón, 1995) se encarga de estudiar los sistemas complejos de percepción del ser humano para poder entender cómo son y cómo funcionan y ser capaces de replicarlos, en la medida de lo posible, en una máquina. En concreto, uno de los sentidos perceptivos más ampliamente tratado es la visión, de ahí sus nombres (Visión Artificial o Visión por Computador).

El estudio de la Visión Artificial tiene como objetivo procesar imágenes (o vídeos, al fin y al cabo, son sucesiones de imágenes en el tiempo, *frames*) de una forma bioinspirada en el ser humano, para poder identificar y reconocer objetos, y llegar a entender lo que se ve en la imagen.

Hasta hace poco, la capacidad de los algoritmos de Visión Artificial era limitada. Sin embargo, los avances en Inteligencia Artificial en campos como el Aprendizaje Automático (*Machine Learning*) y el Aprendizaje Profundo (*Deep Learning*), el auge de las Redes Neuronales, el notable incremento de los datos que se generan y de los que disponemos actualmente (cada día se comparten más de mil ochocientos millones de imágenes por internet (Khedekar, 2014)), y sobre todo, la capacidad de cómputo de las máquinas a las que podemos acceder

hoy en día han propiciado la evolución de modelos y algoritmos más eficientes y la aparición de máquinas más potentes.

2.1.1 Problemas dentro de la Visión Artificial

Los problemas más habituales en los que los algoritmos de Visión Artificial (Brownlee, 2019) hacen acto de presencia suelen ser los siguientes:

- Clasificación: dado un conjunto de categorías, ¿en cuál se engloba un objeto?
- Identificación: ¿qué tipo de objeto es?
- Verificación: dada una categoría, ¿pertenece el objeto de la imagen a ella?
- Detección: dado un tipo de objeto, ¿dónde está en la imagen?
- Segmentación: ¿qué parte de la imagen se corresponde con el objeto?
- Reconocimiento: ¿qué tipos de objetos hay en la imagen y dónde están?

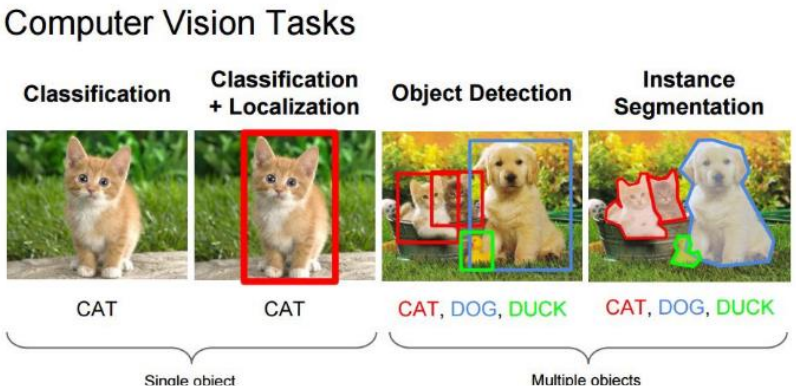

Ilustración 1: Problemas de visión artificial.

En este Trabajo Fin de Máster, los objetos en los que vamos a centrar la detección y el reconocimiento serán los relacionados con la movilidad inteligente, concretamente y como primer caso de uso los ciclistas.

2.2. Introducción al Aprendizaje Profundo

El Aprendizaje Profundo, o *Deep Learning* (Guo et al. 2016), es una parte del Aprendizaje Automático que ha ido tomando tanta relevancia en los últimos años que ya se trata y se estudia como una rama de la Inteligencia Artificial en sí misma.

El *Deep Learning* estudia los métodos para aprender características generalistas y de alto nivel de un conjunto de datos dado, utilizando para ello estructuras de características jerárquicas, de ahí el calificativo ‘profundo’.

Estos modelos computacionales se construyen en capas, de forma que cada una aprende y extrae una determinada información de los datos, simulando el comportamiento de un cerebro humano. De esta forma, la conjunción del conocimiento aportado por cada capa permite reconocer patrones complejos dentro de los datos.

Esta intención de replicar computacionalmente el comportamiento del pensamiento de un cerebro propició en 1943 la aparición del primero modelo de neurona (McCulloh y Pitts, 1943). Este fue el inicio de las redes neuronales artificiales, que hoy en día alcanzan tanta popularidad porque los estudios y avances en este ámbito han conseguido superar los de otras ramas, como el Procesamiento de Lenguaje Natural (Mikolov, Sutskever, Chen, Corrado y Dean 2013) o la Visión Artificial (Tokui, Oono, Hido, y Clayton, 2015) propiamente dichas.

A continuación, podemos observar los principales hitos (Krizhevsky, Sutskever y Hinton, 2012) contributivos en el Aprendizaje Profundo:

Milestone/contribution	Contributor, year
MCP model, regarded as the ancestor of the Artificial Neural Network	McCulloch & Pitts, 1943
Hebbian learning rule	Hebb, 1949
First perceptron	Rosenblatt, 1958
Backpropagation	Werbos, 1974
Neocognitron, regarded as the ancestor of the Convolutional Neural Network	Fukushima, 1980
Boltzmann Machine	Ackley, Hinton & Sejnowski, 1985
Restricted Boltzmann Machine (initially known as Harmonium)	Smolensky, 1986
Recurrent Neural Network	Jordan, 1986
Autoencoders	Rumelhart, Hinton & Williams, 1986 Ballard, 1987
LeNet, starting the era of Convolutional Neural Networks	LeCun, 1990
LSTM	Hochreiter & Schmidhuber, 1997
Deep Belief Network, ushering the "age of deep learning"	Hinton, 2006
Deep Boltzmann Machine	Salakhutdinov & Hinton, 2009
AlexNet, starting the age of CNN used for ImageNet classification	Krizhevsky, Sutskever, & Hinton, 2012

Ilustración 2: Hitos más importantes del Aprendizaje Automático que han desembocado en la era del Deep Learning.

Si bien es cierto que las redes neuronales existen desde mediados del siglo XX, no comenzaron a tener la popularidad actual hasta el año 2006, cuando Hilton introdujo la *Deep Belief Network* (Hinton, Osindero, y Teh 2006), que supuso el despertar de las arquitecturas profundas y algoritmos de Aprendizaje Profundo.

El interés actual en el estudio de las técnicas de Aprendizaje Profundo también tiene su origen en las propias características contextuales, en cuanto a datos y tecnología, de la época en que vivimos:

- Más datos con los que entrenar modelos
- Más capacidades de computación
- Más capacidad de procesamiento (GPUs)

- Aparición de frameworks como Tensorflow (Abadi et al., 2016), PyTorch (Paszke et al., 2017), Caffe (Jia et al., 2014), Chainer (Tokui, Oono, Hido, y Clayton, 2015) , etc. que facilitan el trabajo con estas técnicas

2.2.1. Redes Convolucionales

Son numerosos los métodos tradicionales de la Percepción Computacional (Partida, Manrique y Barrón, 1995), que se siguen estudiando y utilizando a día de hoy: detección y cancelación de anomalías, operadores de histograma, filtros aritméticos, detección de bordes, filtros morfológicos, crecimiento de regiones, extracción de características…

Sin embargo, la mayoría de ellos, aunque funcionan bien, necesitan de un experto que los configure de forma manual y ad-hoc para cada problema.

Es en este punto donde entra en juego el Aprendizaje Profundo, principalmente con las Redes Neuronales Convolucionales (Lindsay, 2020), que aprenden, con suficiente entrenamiento, las características más descriptivas de las imágenes.

Las redes neuronales convolucionales surgieron en el contexto de la visión por computador y son ubicuas en todo problema que implique el uso de imágenes. Una Red Neuronal Convolucional es un algoritmo de Aprendizaje Profundo (Voulodimos, Doulamis, Doulamis, y Protopapadakis, 2018) que, partiendo de un conjunto de datos de entrada (una imagen, en este caso), aprende y asigna pesos o importancia a las características de cada imagen consiguiendo “entender” qué hay en ella.

Para ello, las CNN (C*onvolutional Neural Network*) asumen ciertas características espaciales de los inputs que permiten simplificar las arquitecturas, reduciendo en gran medida el número de parámetros.

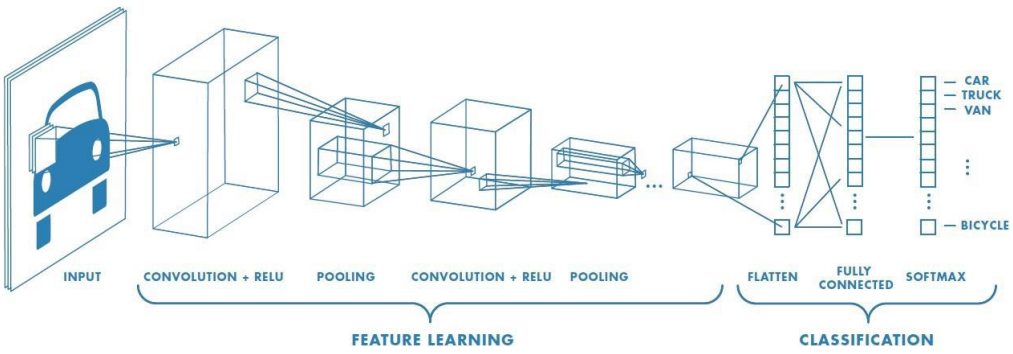

Ilustración 3: Arquitectura genérica de las Redes Neuronales Convolucionales.

De nuevo, con inspiración en el funcionamiento del cerebro humano, las redes convolucionales se diseñan intentando simular los patrones de conectividad de las neuronas del córtex visual (Lindsay, 2020). Esta bioinspiracion fue fundada en 1959, cuando Hubel y Wiesel experimentaron con un gato y midieron en su cerebro los estímulos cuando se le mostraban formas o figuras concretas (Hubel y Wiesel, 1962). Con esto se descubrió que ciertas áreas del córtex se activaban con esas imágenes, y que las neuronas poseen una organización en estructura jerárquica, con neuronas simples que producen respuestas ante cambios de luz, y otras neuronas más complejas que responden ante movimientos y formas.

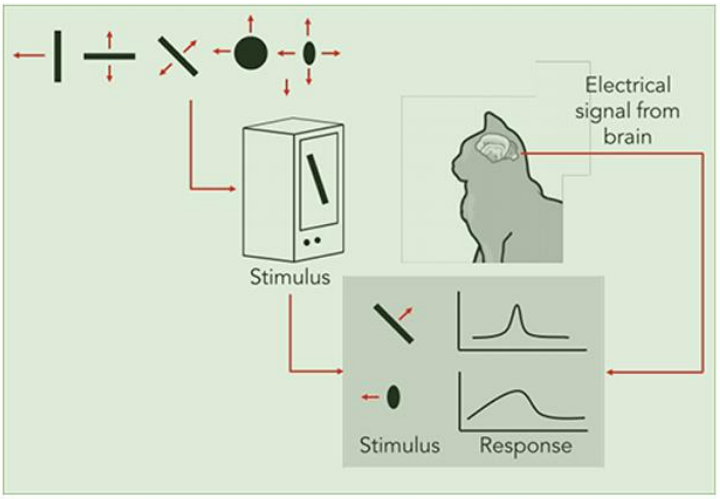

Ilustración 4: Reacción del cerebro de un gato ante estímulos.

En 1998, Yann LeCun, considerado uno de los padres del Aprendizaje Profundo moderno, introdujo el primer caso práctico de una red neuronal convolucional entrenada con *Gradient Descent* (LeCun, Bottou, Bengio, y Haffner, 1998) para reconocer dígitos para el servicio postal.

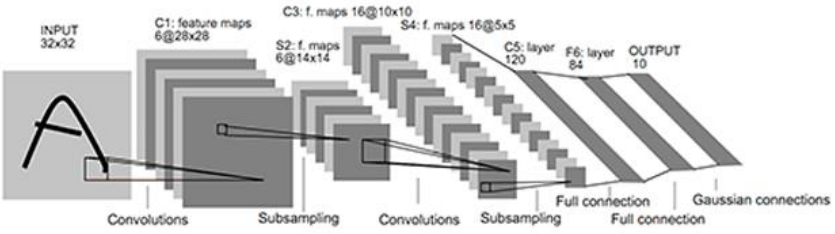

Ilustración 5: Arquitectura LeNet5 para el reconocimiento de dígitos.

Sin embargo, la verdadera explosión en el uso de las CNN vino cuando una red de este tipo ganó en 2012 la competición de clasificación de imágenes ImageNet por un gran margen.

Algunas de las CNN más conocidas actualmente son: LeNet (LeCun et al., 1989), AlexNet (Krizhevsky, Sutskever y Hinton, 2012), GoogLeNet (Szegedy et al., 2015) ResNet (He, Zhang, Ren, y Sun, 2016)

2.2.2. Transfer Learning

Habitualmente, nos encontramos en una situación comprometida a la hora de entrenar modelos de visión Artificial debido, principalmente, a que no disponemos de una cantidad suficiente de datos de entrenamiento. Como se ha explicado anteriormente, las CNN son modelos complejos con muchos parámetros y, por lo tanto, requieren de un conjunto de datos de entrenamiento de tamaño considerable para conseguir entrenar una red con éxito. Un modelo con una arquitectura tan grande y compleja entrenado con pocos datos terminará, muy probablemente, con sobreajuste.

Transfer Learning (Torrey y Shavlik, 2010), o Transferencia del Aprendizaje, es una técnica que surge para resolver estos problemas, aplicando una transferencia de lo aprendido con conjuntos de datos grandes a otros problemas de índole similar con menos datos.

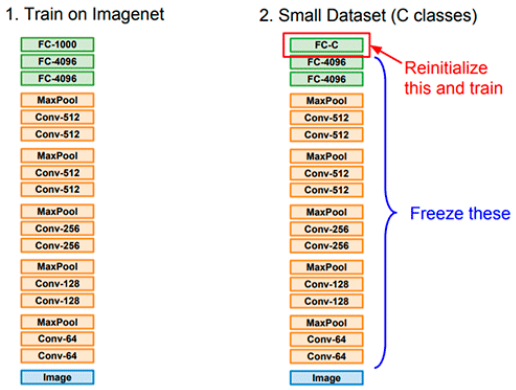

Ilustración 6: Descripción genérica de la técnica Transfer Learning.

El concepto subyacente es utilizar un modelo ya entrenado y utilizar el conjunto de datos de nuestro problema concreto para ajustar únicamente los parámetros de las últimas capas del modelo, manteniendo el resto. De esta manera, el aprendizaje que había adquirido el modelo original sobre el problema original se transfiere en cierta medida a nuestro problema concreto, manteniendo elementos que posiblemente son comunes a ambos (cabe recordar que en las primeras capas de las redes neuronales convolucionales se realizan las representaciones simples y generales como bordes, texturas, esquinas y formas).

No hay realmente una manera exacta de aplicar el *Transfer Learning*. Dependiendo de los datos de los que dispongamos, podemos reinicializar y entrenar un mayor número de capas del modelo original.

Por último, siempre hay que tener en cuenta que la efectividad de esta técnica puede verse comprometida si el problema al que nos enfrentamos difiere demasiado del problema original para el que se entrenó la red original.

2.2.3. Data Augmentation

Otra técnica muy popular en el caso de que no dispongamos de una cantidad de datos suficiente para garantizar un buen entrenamiento de la red consiste en la obtención de nuevos datos de entrada a partir de los que ya se tienen, y así poder ampliar el conjunto de datos. Esta técnica es la llamada *Data Augmentation*, o Aumento de Datos (Van Dyk y Meng, 2001).

El proceso de *Data Augmentation* consiste en la generación artificial de datos utilizando pequeñas perturbaciones en los datos originales que ya tenemos. Esto nos permite obtener nuevos datos, que a ojos de la red son distintos, pero que nosotros sabemos que mantienen la clase. Es decir, tenemos nuevos datos ya etiquetados.

Este método es ampliamente utilizado en el ámbito de la visión artificial y el tratamiento de imágenes, llegando a convertirse en un estándar de regularización ya que ayuda a combatir el sobreajuste en las redes neuronales convolucionales.

Aplicado al tratamiento de imágenes, algunas de las transformaciones simples más comúnmente aplicadas para generar nuevos datos son:

- transformaciones geométricas como volteo, rotación, traslación, recorte, escalado…
- transformaciones del espacio de color como fundición de color, brillo variable e inyección de ruido.

Las transformaciones geométricas tienen buen resultado cuando tenemos imágenes con sesgos en cuanto a posición, por ejemplo en un problema de reconocimiento de caras.

Las transformaciones del espectro del color son más utilizadas para problemas relativos a iluminación de escenas u objetos.

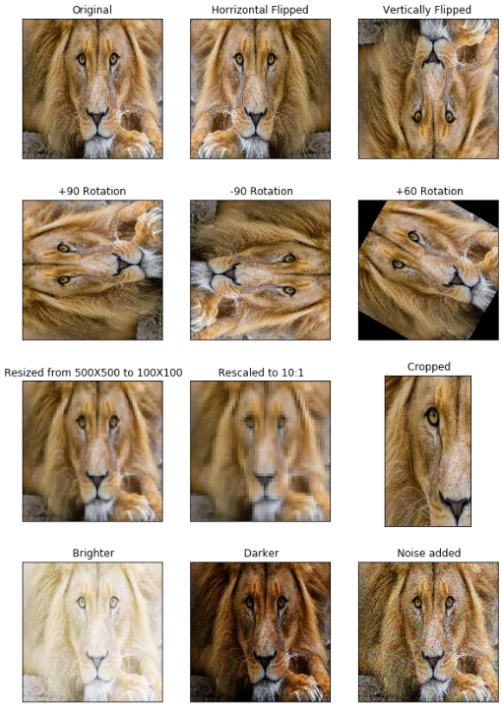

Ilustración 7: Tipos de transformaciones en Data Augmentation.

Si bien es verdad que al aplicar estas segundas transformaciones, en ciertos problemas, puede resultar contraproducente porque las features o características de las imágenes varían, es muy habitual encontrar *Data Augmentation* con transformaciones del primer tipo como ayuda de regularización en multitud de entrenamientos.

Para solventar este tipo de problemas, actualmente se está trabajando en la generación de nuevos datos a partir de Redes Adversarias Generativas, o GAN (*Generative Adversarial Networks*), utilizadas para sintetizar imágenes siguiendo los patrones de las imágenes originales (Tanaka y Aranha, 2019). No se profundizará más en este tipo de redes dado que se escapan del alcance del presente trabajo.

2.3. Modelos más utilizados en Smart Mobility

Entre las arquitecturas más utilizadas en la actualidad a la hora de construir un modelo de Aprendizaje Profundo encontramos dos líneas de investigación: arquitecturas de dos fases, y arquitecturas de una fase.

Las arquitecturas de dos fases trabajan, como su propio nombre indica, en dos etapas. En la primera etapa se generan regiones candidatas a contener un objeto a detectar, y en la segunda etapa se identifica y clasifica el objeto en sí dentro de esa región. Los modelos

basados en este tipo de arquitectura son conocidos también como detectores de objetos basados en regiones.

En contraposición, las arquitecturas de una fase realizan esta tarea en un nivel mayor de abstracción, dado que realizan la inferencia de la localización del objeto y la identificación del mismo en una sola etapa.

2.3.1. Arquitecturas de dos fases: métodos basados en regiones

El primer tipo de arquitectura que se expone en este trabajo se llama arquitectura de dos fases, también conocido como arquitectura basada en regiones. Esta tipología de construcción de redes vio la luz en el año 2012 en la competición propuesta por ImageNet, *Large Scale Visual Recognition Challenge* (ILSVRC) de la mano del que fue su ganador: AlexNet (Krizhevsky, Sutskever y Hinton, 2012). Este acontecimiento revolucionó las propuestas de soluciones para problemas de detección de objetos, ya que dejó claro que las redes neuronales convolucionales se iban a abrir paso en este ámbito.

Hasta el momento, se utilizaban metodologías similares a la fuerza bruta, o incluso la propia fuerza bruta en sí misma, para detectar en una imagen un objeto en concreto. Esto significa que se desarrollaban soluciones basadas en una ventana que se deslizaba a lo largo de la imagen buscando el objeto deseado.

Esto provoca que si se quiere detectar objetos distintos se tengan que usar ventanas distintas, cada una acorde al tamaño y forma del objeto en cuestión: cada ventana deslizante va realizando un recorte de la imagen en cada posición que se desliza, y este recorte se pasa como entrada a un modelo de clasificación (regresión, svm, …). Este modelo de clasificación es el responsable de decidir si en ese recorte está o no el objeto en búsqueda.

Ilustración 8: Ventana deslizante sobre la imagen.

Como podemos imaginar, este tipo de procedimiento es demasiado costoso e ineficiente, tanto en tiempos de ejecución como en recursos computacionales requeridos para ello. Es

por esto que surge de forma casi inmediata una propuesta para solucionar este inconveniente: hay que disminuir el número de ventanas que evalúa el modelo de clasificación.

Partiendo de esta idea, se propone un método cuya misión principal es realizar una propuesta fundamentada de ventanas deslizantes cuya evaluación resulte de interés, diferenciándolas con claridad de las ventanas de las que no obtendríamos un resultado positivo en la evaluación del modelo de clasificación. De esta manera, surgen los modelos de propuestas de regiones de interés (Kong, Yao, Chen y Sun, 2016), o ROI, de una imagen.

Sobre propuesta de regiones hay mucho trabajo realizado en el campo de la Percepción Computacional. El área de estudio de Crecimiento de Regiones o *Region Growth*, (Yuheng y Hao, 2017) estudia y propone diversos métodos para segmentar una imagen explorando los píxeles de ésta y su vecindad, de forma que va expandiendo dichos pixeles a grupos de pixeles más grandes que contienen información relacionada. Hay muchas líneas de desarrollo de este tipo de métodos: basado en semillas, de tipo *split and merge*, basados en *Gradient Vector Flow*, basados en *Watershed* (por inundación), basados en grafos… No profundizaremos en estos métodos dado que excede al tema principal del trabajo.

Ilustración 9: Propuesta de regiones.

2.3.1.1 R-CNN

En el año 2013, Ross Girshick lanzó una propuesta de arquitectura basada en la idea que acabamos de comentar de propuesta de regiones de interés, la red neuronal convolucional que hoy conocemos como R-CNN (Girshick, Donahue, Darrell, Malik y Mercan, 2014).

Esta red convolucional propone alrededor de 2000 regiones de interés por cada imagen que trata. Posteriormente, cada una de estas regiones de interés propuestas se toma como dato de entrada para la red convolucional de modo que cada una de estas regiones se evalúa de forma estrictamente individual. A continuación, todas estas evaluaciones individuales

confluyen finalmente a través de capas *fully connected*, haciendo posible, de esta manera, la estimación de una caja delimitadora (o *bounding box*) y la clasificación del objeto de forma simultánea. En la siguiente figura se puede observar cómo es el flujo de este tipo de arquitectura de red neuronal:

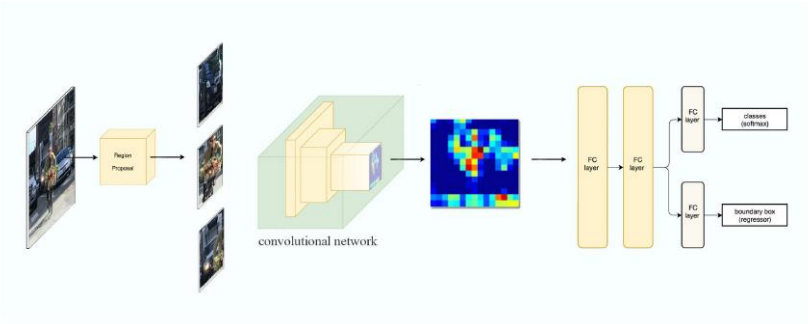

Ilustración 10: Arquitectura de R-CNN.

Una vez obtenido este modelo, las ventajas obtenidas con respecto a los primeros métodos que comentábamos basados en fuerza bruta son más que notorias, básicamente debido a que la identificación de regiones de interés aumenta la precisión en la detección de objetos, así como el tiempo de ejecución es sustancialmente menor ya que con la propuesta de regiones se calculan y evalúan muchas menos cajas delimitadoras hasta encontrar el objeto buscado en la imagen.

A pesar de estas mejoras, el modelo R-CNN sigue siendo bastante lento, tanto en la fase de entrenamiento como en la fase de evaluación o inferencia. Esto es, fundamentalmente, porque este método aun propone un número de regiones demasiado alto, produciendo así un solape (en cierta medida) de muchas de ellas. Esto provoca que este método, aunque sí es mejor, aun no sea óptimo.

2.3.2.1 Fast R-CNN

Fue el mismo Ross Girshick quien intentó evolucionar este método y propuso una solución para mejorar el algoritmo. Dos años después de publicar R-CNN, en 2015 publicó una nueva red neuronal convolucional llamada Fast R-CNN, cuya principal diferencia con respecto a la primera radica en la manera en que se calculan y se utilizan las ROIs (Girshick, 2015).

Como comentábamos en el apartado anterior, la R-CNN calcula alrededor de 2000 regiones de interés y evalúa independientemente cada una de ellas. Es decir, se realiza el proceso de extracción de características alrededor de 2000 veces sobre las mismas zonas. Ross Girshick reflexionó sobre esto y propuso que, en vez de evaluar cada una de esas 2000 imágenes desde cero, de forma independiente, sería interesante realizar una extracción de

características de toda la imagen, combinarlo con un método externo de propuesta de regiones, y posteriormente unificar todo ello mediante las mismas capas *fully connected* que llevarán a cabo la localización del objeto y su posterior clasificación. En la siguiente figura podemos ver cómo sería este método:

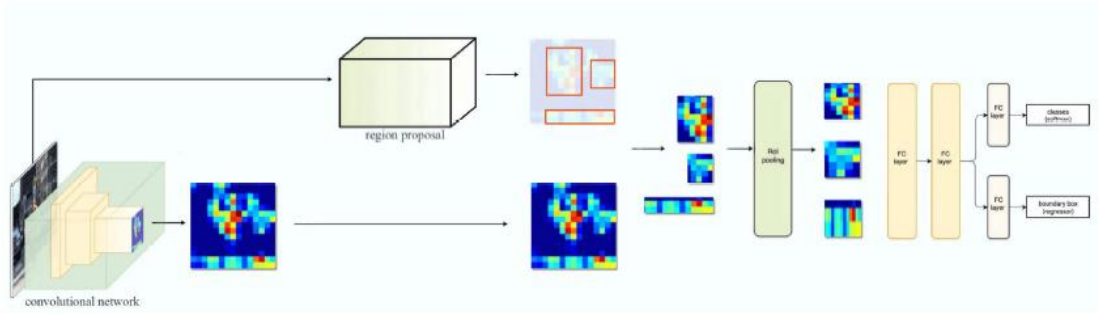

Ilustración 11: Arquitectura de Fast R-CNN

Este detector de objetos, Fast R-CNN, reduce a su vez el tiempo con respecto a su antecesor. No obstante, sigue siendo bastante lento debido a la incorporación del método de propuesta de regiones de interés de forma externa.

2.3.3.1 Faster R-CNN

Fue en ese mismo año, 2015, cuando Shaoquiq Ren y sus compañeros realizaron una nueva propuesta que alcanza una siguiente mejora en lo que a tiempos de ejecución se refiere: la red Faster R-CNN (Ren, He, Girshick y Sun, 2015).

El modelo Faster R-CNN se asemeja mucho a Fast R-CNN, con la salvedad de la sustitución del método de propuesta de ROIs por una red neuronal que realiza esa función, llamada *Region Proposal Network* (o RPN). Esta red RPN tiene como misión realizar la propuesta de regiones partiendo de las características extraídas de la imagen por la red neuronal convolucional, lo cual significa un avance más en la dirección de la optimización en la generación de regiones de interés.

A continuación, podemos observar en la figura la arquitectura de Faster R-CNN y su semejanza con la de Fast R-CNN con la única salvedad de lo explicado anteriormente: el método externo de proposición de regiones se sustituye por la red RPN, partiendo de la extracción inicial de características del modelo CNN:

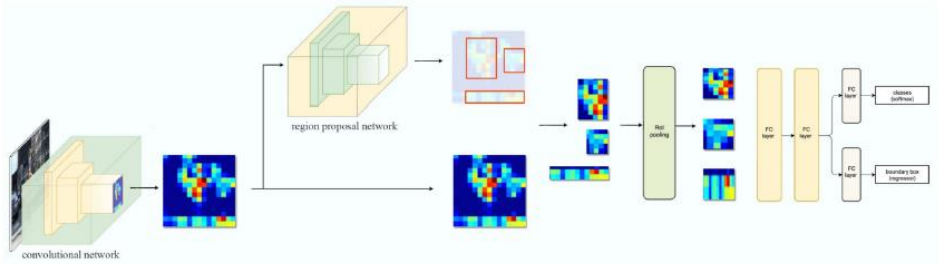

Ilustración 12: Arquitectura de Faster R-CNN

En la actualidad, hay muchas arquitecturas que superan en velocidad a Faster R-CNN. Basándose en el hecho de que el uso de capas *fully connected* resulta bastante caro en cuanto a tiempos y recursos de ejecución, se han empezado a investigar nuevas redes basadas en capas convolucionales, que resultan bastante más ágiles. De esta forma, surgió la red convolucional Resnet (He, Zhang, Ren y Sun, 2016) y se integró en el modelo Faster R-CNN, dando lugar así al modelo R-FCN (Dai, Li, He y Sun, 2016), un detector totalmente convolucional cuyos rendimiento y precisión superan con creces el de Faster R-CNN.

2.3.2. Arquitecturas de una fase

2.3.2.1 Single Shot Detector

En 2015 se presentó un método para detectar objetos en imágenes utilizando una única red neuronal profunda.

Este novedoso enfoque, al que llamaron *Single Shot Detector*, o SSD (Liu et al., 2016), procesa las imágenes discretizando el espacio de salida en cajas delimitadoras, o *bounding boxes*, en un conjunto de cajas delimitadoras predeterminados en diferentes proporciones y escalas que se distribuyen por toda la ubicación del mapa de características de la imagen.

Para generar la inferencia, la red genera una serie de puntuaciones para cada categoría de objeto a detectar en cada caja delimitadora, y posteriormente ajusta estos cuadros para que encajen mejor en el objeto con mayor puntuación.

Además, la red combina predicciones de múltiples mapas de características a diferentes resoluciones para poder ajustar las cajas delimitadoras a objetos de tamaños variados.

SSD es más simple que los métodos que utilizan la denominada *Region Proposal Network,* o red de propuestas de región, porque elimina todas las fases de propuestas de regiones innecesarias hasta que encuentra la región que se adapta al objeto, a la vez que elimina

también las etapas de remuestreo de píxeles o características posteriores, de forma que aglutina todos los cálculos a realizar en una sola red. Esto la convierte en una red más rápida tanto en la fase de entrenamiento como en la de inferencia.

2.3.2.1.1 SSD Resnet

Al eliminar la fase de propuesta de regiones y usar en su lugar cajas de anclaje se gana en velocidad de inferencia pero se pierde en la detección de objetos pequeños (definidos habitualmente como menores de 36x36 píxeles) ya que, al pasar por todas las capas convolucionales de la red, un objeto de ese tamaño pierde mucha forma y resolución.

Para solucionar esto, se introdujo el extractor de características *Residual Net*, o ResNet, un clasificador deconvolucional que se utiliza para aumentar la resolución de los mapas de características tras las capas convolucionales (Lu, Kang, Nishide y Ren, 2019).

Este tipo de redes residuales deconvolucionales proporciona conexiones de salto entre bloques convolucionales, lo que disminuye los efectos del gradiente de fuga y permite que las redes sean más profundas. De hecho, los ResNets normalmente pueden llegar a 101 capas, mientras que otro tipo de arquitecturas, como las redes VGG, solo pueden llegar hasta 19 (Sengupta, Ye, Wang, Liu y Roy, 2019).

2.3.2.1.2 SSD Mobile lite

La arquitectura SSD ha dado lugar a otro conocido detector de objetos, el SSD Mobile Lite (Sandler, Howard, Zhu, Zhmoginov y Chen, 2018). Este modelo de detección es una red convolucional entrenada con el dataset COCO, y diseñada particularmente para ser ejecutada en entornos con ciertas restricciones, como podría ser el de conducción autónoma u otros ámbitos en *Smart Mobility*.

Ese diseño especial consiste en una arquitectura que necesita hasta una décima parte de los parámetros que necesitarían otras redes, como por ejemplo Yolo v2, a la vez que mantiene la precisión al nivel de otros modelos con estructura SSD.

Esto lo hace un perfecto candidato para el estudio que se realiza en este trabajo, ya que por sus características podría ser una buena opción para realizar inferencia en tiempo real en un problema de detección de objetos en *Smart Mobility*.

2.3.2.2. You Only Look Once

El enfoque que había hasta entonces consistía en procesos de detección que reutilizaban clasificadores para efectuar esa detección. A la hora de detectar un objeto, estos procesos tomaban un clasificador de ese objeto y lo evaluaban en varias ubicaciones y varias escalas en una imagen de prueba, bajo el concepto de ‘ventana deslizante’ en el que el clasificador

se ejecuta en ubicaciones uniformemente espaciadas sobre toda la imagen (Felzenszwalb, Girshick, McAllester y Ramanan, 2009).

Como hemos visto en el apartado anterior, enfoques más modernos como, como la R-CNN, utilizan métodos de propuesta de ROIs para generar, en primera instancia, posibles cajas delimitadoras en una imagen y, a continuación, ejecutar un clasificador en estos cuadros propuestos. Después, se refinan los cuadros delimitadores, eliminando las detecciones duplicadas y volviendo a puntuar los cuadros basándose en otros objetos de la escena. Todo este proceso hace que el sistema de detección en general sea lento y difícil de optimizar, dado que cada uno de los componentes que lo forman debe entrenarse de forma diferente y particular.

Cuando en 2015 se presenta *You Only Look Once*, o comúnmente abreviado por sus siglas YOLO, supone un nuevo enfoque para los problemas de detección de objetos. Este trabajo sobre métodos de detección de objetos replantea utilizar clasificadores como detectores de objetos. Considera la detección de objetos como un problema de regresión entre las cajas delimitadoras separadas en el espacio de la imagen y las probabilidades de clase asociadas.

Se trata de una única red neuronal que predice las cajas delimitadoras y las probabilidades de clase directamente a partir de imágenes completas en una sola evaluación. Como todo el proceso de detección es una sola red, puede optimizarse de principio a fin directamente en el rendimiento de la detección.

Esta manera unificada de llevar a cabo un proceso de detección tiene ciertas ventajas sobre las metodologías tradicionales vistas hasta ahora:

- Por un lado, la velocidad de inferencia. Puesto que estamos encajando la detección de objetos dentro de un problema de regresión, el sistema resultante es bastante simple. Solo hay que ejecutar la red neuronal convolucional sobre una imagen para obtener las predicciones/detecciones. La red YOLO con estructura básica puede llegar a funcionar a 45 fps (*frames per second*), y la versión simplificada puede alcanzar ejecuciones a más de 150 fps. Esto supone una latencia menor de 25 milisegundos a la hora de procesar un video en tiempo real.

- Por otro lado, YOLO lleva a cabo un aprendizaje global sobre la imagen cuando realiza las predicciones. A diferencia de los modelos construidos sobre el concepto de las ventanas deslizantes y de propuesta de ROIs, YOLO visualiza la imagen al completo en cada ejecución, de forma que le resulta posible codificar toda la información posible sobre las clases a detectar, así como su apariencia.

- Por último, YOLO es capaz de aprender representaciones generalizables de los objetos. Cuando se entrena con imágenes naturales y se prueba con obras de arte, por ejemplo, YOLO supera varios de los detectores de objetos, como DPM y R-CNN con una amplia diferencia.

No obstante, YOLO sigue estando por detrás de los modelos de detección más avanzados en cuanto a precisión de detección, y aunque es capaz de detectar rápidamente los objetos en las imágenes, encuentra dificultades para encontrar algunos objetos, especialmente los pequeños, con precisión.

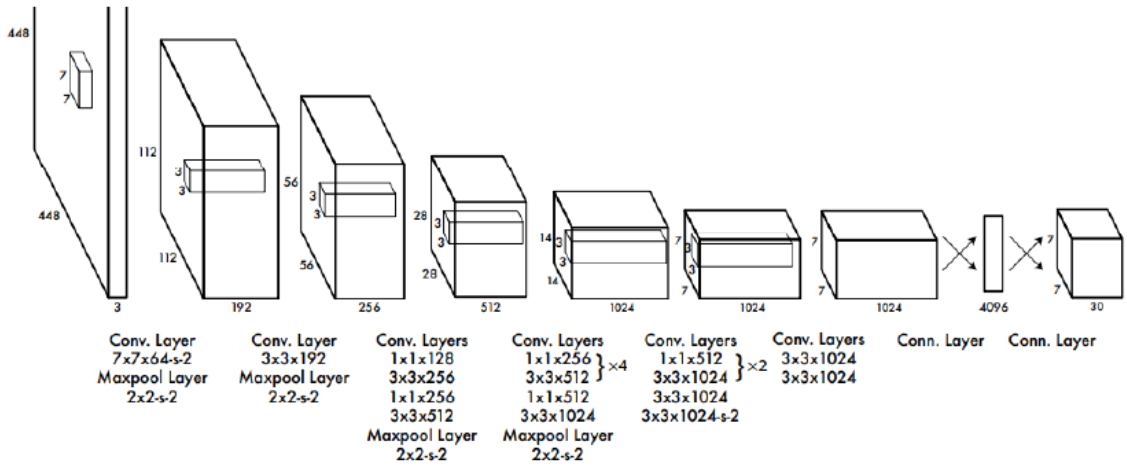

Ilustración 13: Estructura de la primera version de YOLO, con 24 capas convolucionales seguias de dos capas fully-connected.

Se han ido publicando varias versiones de YOLO, desde la versión inicial (Redmon, Divvala, Girshick y Farhadi, 2016), hasta la más actual YOLOv4 (Bochkovskiy, Wang y Liao, 2020), publicada en abril de 2020.

Estudiaremos las versiones YOLOv3 (Redmon y Farhadi, 2018) y YOLOv3 tiny (He, Huang, Wei, Li y Guo, 2019), de especial interés por su tamaño *tiny* (diminuto, en inglés) en cuanto a capas de profundidad, lo que se traduce en mayor rapidez, aspecto clave en *Smart Mobility*, para llegar a realizar un procesamiento en tiempo real.

2.3.2.2.1. YOLO v3

Comenzamos comentando la red YOLO v3 (Redmon y Farhadi, 2018) ya que incorpora varias mejoras con respecto a su primera *release*, algunas de ellas ya encontradas en YOLO v2. Las más destacadas son las siguientes:

- La normalización de lotes en las capas convolucionales.

- Un clasificador de alta resolución.
- En la última capa convolucional nos encontramos con las cajas de anclaje (o *anchor boxes*). En esta capa, en lugar de realizar la predicción directamente sobre las cajas, se realiza sobre unas cajas de anclaje previamente definidas. Esto resulta muy interesante, ya que con ello podemos adecuar la definición de estas cajas predefinidas a la forma y tamaño de los objetos que vamos a tratar en nuestras imágenes.
- Clústeres dimensionales: dado que cada objeto posee un tamaño y una forma concretos, para calcular las cajas de anclaje que cubren toda la variedad de formas de objetos en un conjunto de imágenes se usa el algoritmo *K-Means*.
- Una predicción directa de la ubicación, dentro de la imagen, del objeto en cuestión.

2.3.2.2.2. YOLO v3 tiny

YOLOv3 tiny (He, Huang, Wei, Li y Guo, 2019) es una red derivada de YOLOV3 pero que reduce notablemente su arquitectura hasta un total de 15 capas convolucionales.

Esto hace que esta red sea considerablemente más rápida en ejecución y en inferencia, aunque para ello sacrifica en cierto grado la precisión, principalmente en objetos pequeños.

Es por este motivo que se considera en este trabajo dado el objetivo final, que es ejecutar la red en un dispositivo portable de poca potencia y bajo consumo.

2.3.2.2.3 YOLO v4

La versión de YOLO v4 (Bochkovskiy, Wang y Liao, 2020), si bien presenta mejoras notables frente a su antecesor, YOLOv3, tanto en precisión como en velocidad, no se utilizará en este trabajo por los mismos motivos por los que sí utilizaremos YOLOv3 tiny: los dispositivos finales en los que va a ser utilizado.

Sí sería un buen modelo para dedicarlo a la conducción autónoma, con distintos requerimientos a nivel de hardware, ya que posee una gran velocidad en cuanto a FPS y una precisión muy alta con respecto al conjunto de detectores actuales.

Si bien es cierto que existe una versión llamada YOLO v5 publicada con escasamente un mes de diferencia a partir de la v4, ésta no proporciona una comparativa en profundidad con respecto a YOLO v4 ni deja claras cuales son las diferencias o mejoras con respecto a ésta.

3. Objetivos y metodología de trabajo

El gran objetivo global del proyecto es realizar un trabajo tanto de investigación como de análisis y desarrollo que permita sentar una base analítica para aplicaciones de futuro desarrollo en el ámbito de la Movilidad Inteligente.

3.1. Objetivo general

Este Trabajo fin de Máster, dentro del alcance que permiten los conocimientos obtenidos tras el estudio de sus asignaturas, tiene como objetivo general desarrollar un modelo de detección de ciclistas en tiempo real. Este modelo estará basado en una red neuronal convolucional ya existente, que se seleccionará para este caso de uso de entre las más utilizadas en el ámbito de la movilidad inteligente, mediante una comparación y evaluación de las mismas.

Posteriormente, a partir de ese modelo ganador, se utilizarán técnicas de *Transfer Learning* para ajustar su precisión a este caso de uso, así como otras técnicas de recopilación de datos y *Data Agumentation* para confeccionar un conjunto de datos de entrenamiento adecuado para tal finalidad.

Fuera del alcance de este Proyecto, se pretende continuar este trabajo partiendo de este modelo como base para, a futuro y sin pérdida de generalidad, detectar otros objetos u obstáculos relevantes en el ámbito de la conducción, e implementar un sistema de alertas compartidas de forma que con cada detección se emita una notificación con la información de objeto detectado, momento, y lugar, que sea recogida por una aplicación de movilidad y navegación.

Esto es de especial interés dado que los sistemas de alertas implementados en las aplicaciones de navegación actuales no son automáticos, sino que es el propio conductor quien tiene que seleccionar manualmente, de entre un catálogo de notificaciones, la que quiere señalizar mientras va conduciendo.

3.2. Objetivos específicos

A lo largo de este trabajo se planifican y desarrollan una serie de tareas destinadas a cubrir los siguientes objetivos específicos de una manera *SMART (Specific*, *Medible*, *Attainable*, *Relevant*, y *Time-related*):

- Buscar y comprender la información necesaria sobre los modelos de detección de objetos más relevantes en la actualidad.
- Diseñar un entorno de prueba y comparación de modelos.
- Importar y utilizar modelos ya entrenados por terceros.
- Evaluar modelos, compararlos y elegir el mejor para un caso de uso.
- Diseñar un conjunto de datos de entrenamiento y validación, aplicando para ello técnicas de *Data Augmentation* donde sea necesario.
- Reentrenar un modelo ya existente aplicando técnicas de *Transfer Learning.*
- Diseñar una herramienta básica de prueba para implementar un envío de alertas a partir del modelo de detección.
- Planificar las tareas necesarias para la consecución de estos objetivos, teniendo en cuenta orden, prioridad, y tiempo.
- Comunicar todas las tareas realizadas para la consecución de estos objetivos de forma clara, precisa, y entendible.

3.3. Metodología del trabajo

A continuación se muestra la relación de tareas necesarias para la consecución de cada uno de los objetivos específicos. Además, se detalla el orden, prioridad y tiempo de cada tarea.

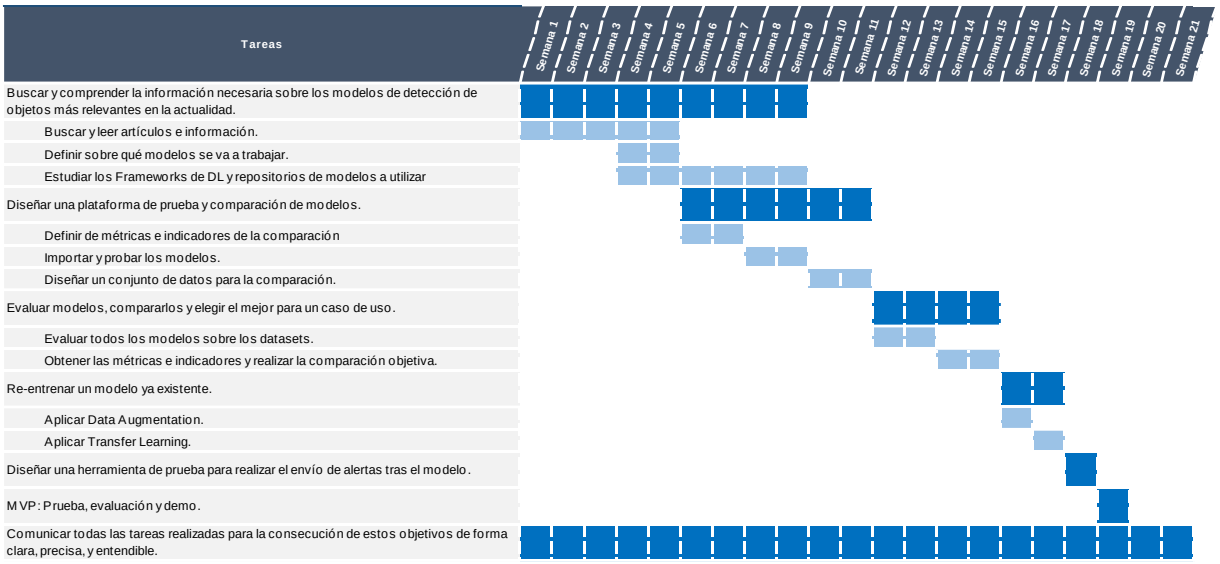

Ilustración 14: Diagrama de Gantt y planificación de tareas del TFM.

4. Identificación de requisitos

En esta sección se habla de los requisitos necesarios, a distintos niveles, para poder llevar a cabo la consecución de las tareas mencionadas en el apartado anterior, con el único objetivo del transcurso continuo del flujo de conocimiento y trabajo hacia el objetivo final deseado.

Para ello, mencionaremos la infraestructura que se utilizará tanto a nivel hardware, como los repositorios analíticos de terceros a nivel de software.

4.1. Hardware

4.1.1. CPU vs GPU

Una **CPU** (*Central Processing Unit*) es un procesador hardware fundamental en la construcción de cualquier dispositivo electrónico. En concreto, en un ordenador, es el *cerebro* que se encarga de procesar todas las instrucciones, así como los datos de las mismas. Es decir, la encargada de realizar las operaciones que necesitan los programas o aplicaciones para realizar unas determinadas tareas.

Una **GPU** (*Graphics Processing Unit*) es un coprocesador hardware dedicado al procesamiento de gráficos y operaciones de coma flotante, con el objetivo de aligerar la carga de trabajo del procesador principal (CPU). Tradicionalmente, las GPU se han utilizado para videojuegos o aplicaciones gráficas con efectos avanzados como, por ejemplo, 3D.

Visto así, una GPU es simplemente una tarjeta gráfica de toda la vida, pero a continuación detallamos las diferencias con respecto a una CPU que hacen que, en el mundo del *Deep Learning* y de entrenamiento de modelos, sean bastantes populares.

La labor de las GPU es, por tanto, complementar a la CPU en cierto tipo de operaciones para las cuales han sido diseñadas y optimizadas. Veamos en qué se diferencian estas dos unidades de procesamiento:

- Por tipo de tareas para el que han sido diseñadas:
    - Las CPU resuelven tareas de propósito general.
    - Las GPU están optimizadas para tareas muy particulares.

- Por número de cores:
    - Las GPU tienen muchos cores (varios cientos o miles), pero más lentos y limitados a unas pocas operaciones
    - Las CPU tienen pocos cores pero muy rápidos y capaces de realizar una gran variedad de tareas.
- Por tipo de ejecuciones:
    - Las GPU destacan en su capacidad de paralelizar de manera masiva operaciones sencillas.
    - Las CPU están más orientadas a tareas secuenciales generales.
- Otra gran diferencia es el uso de memoria
    - Las CPU utilizan la memoria RAM del sistema.
    - Las GPU tienen su propia memoria RAM integrada.
- Por uso de memoria:
    - Las GPU están optimizadas para obtener grandes cantidades de memoria de manera rápida (tienen un mayor ancho de banda), aunque con mayor latencia.
    - Las CPU son buenas en utilizar pequeñas cantidades de memoria de manera muy rápida.

La cantidad de memoria disponible en una GPU ha ido en aumento durante los últimos años, lo que permite entrenar modelos más grandes. El modelo a entrenar (es decir, los parámetros de la red) tiene que estar en la memoria de la GPU, lo que supone en muchas ocasiones una limitación. Esto lleva a sistemas complejos donde los modelos se dividen y entrenan entre varias GPU, como por ejemplo AlexNet.

Todo esto hace que las GPU sean más efectivas para grandes operaciones numéricas como son las redes neuronales (recordemos que podemos tener millones de parámetros en una red).

Una GPU puede leer de manera rápida matrices gigantescas de una manera eficiente y utilizar sus miles de núcleos para obtener de manera paralela todos los valores de salida. Las operaciones a realizar son muy sencillas, ya que se trata simplemente de multiplicaciones y

sumas de números reales, lo cual entra en el catálogo de operaciones relativamente simples que una GPU puede realizar.

A diferencia de la GPU, una CPU iría elemento a elemento de manera secuencial (o de 8 en 8, si dedicamos los 8 cores a 8 hilos distintos). El número de cores de una CPU se aleja bastante, como hemos visto, de los cientos o miles de cores disponibles en una GPU, por lo que es fácil ver de dónde viene la mejora de velocidad. Sin embargo, este ejemplo está muy simplificado ya que las CPU actuales suelen tener operaciones vectorizadas y librerías algebraicas de alto rendimiento capaces de realizar multiplicaciones de matrices y otras operaciones numéricas de manera muy eficiente y utilizando varios procesadores.

Este ejemplo de la efectividad de las GPU calculando multiplicaciones de matrices es clave para entender por qué son tan útiles para el *Deep Learning*. Las redes neuronales, al fin y al cabo, se pueden ver como un conjunto de multiplicaciones de matrices. Todas las operaciones correspondientes al entrenamiento capa a capa con los parámetros *w* de una red pueden efectuarse en forma de producto de matrices. En muchos casos, productos de matrices enormes, ya que hay redes donde las capas pueden tener cientos o miles de elementos.

Del mismo modo, operaciones típicas como las convoluciones pueden ser también paralelizadas de manera sencilla. Todo esto permite que una GPU acelere el proceso de entrenamiento de una red neuronal en gran medida.

4.1.2. NVIDIA Jetson Nano Developer Kit

No hace mucho, el trabajo sobre tareas de procesamiento de voz o traducción, de manejo de vídeos, o reconocimiento y detección de imágenes estaba reservado únicamente a expertos con medios suficientes como para acceder a las herramientas computacionales que estas tareas requieren. En este sentido, la compañía NVIDIA se ha propuesto poner remedio a esta limitación y desde hace poco ofrece al gran público diversas soluciones orientadas a los desarrolladores menos expertos, independientes o pequeñas empresas.

Una de estas soluciones es la Jetson Nano («Jetson Nano 2GB Developer Kit», 2020), una GPU que da una opción de bajo coste y tamaño reducido al usuario para que desarrolle herramientas y aplicaciones de Inteligencia Artificial. De esta forma, abre nuevos frentes para crear todo tipo de aplicaciones IoT (*Internet of Things*) integradas, entre las que se incluyen grabadores de vídeo de red básicos, pequeños robots y *gateways* inteligentes con capacidades de análisis de datos muy potente, del mismo modo que habilita las posibilidades de entrenamiento y generación de modelos de *Deep Learning* de manera fácil, económica y reduciendo tiempos de entrenamiento con respecto a un ordenador convencional.

Ilustración 15: Jetson Nano

En concreto, para este proyecto se va a utilizar la **Jetson Nano 2GB Developer Kit**, cuyas características se detallan a continuación:

GPU	128-core NVIDIA Maxwell™
CPU	Quad-core ARM® A57 @ 1.43 GHz
Memory	2 GB 64-bit LPDDR4 25.6 GB/s
Storage	64GB microSD
Video Encode	4Kp30 | 4x 1080p30 | 9x 720p30 (H.264/H.265)
Video Decode	4Kp60 | 2x 4Kp30 | 8x 1080p30 | 18x 720p30 (H.264/H.265)
Connectivity	Gigabit Ethernet, 802.11ac wireless
Camera	1x MIPI CSI-2 connector
Display	HDMI
USB	1x USB 3.0 Type A,2x USB 2.0 Type A, USB 2.0 Micro-B
Others	40-pin header (GPIO, I2C, I2S, SPI, UART) 12-pin header (Power and related signals, UART) 4-pin Fan header
Mechanical	100 mm x 80 mm x 29 mm

Ilustración 16: Especificaciones técnicas de la Jetson Nano.

4.2. Software

4.2.1. Frameworks de deep learning

De todos los *frameworks* de *Deep Learning*, probablemente el más utilizado sea TensorFlow (TF): es un *framework open-source* para computación numérica que utiliza grafos de computación donde los datos, en forma de tensores (*Tensor*), que fluyen (*Flow*) entre distintas operaciones. En principio, TF puede utilizarse para una gran cantidad de aplicaciones que precisen de cálculos numéricos, si bien sus aplicaciones comunes giran en torno al *Machine Learning* y, más en particular, al aprendizaje profundo

Fue desarrollado por Google, donde es utilizado tanto en investigación como para sistemas en producción. Su código fue liberado como *open-source* en 2015, y desde entonces se ha convertido en un ecosistema software muy completo, con una gran comunidad de usuarios.

Se utiliza principalmente con una API en Python, tal y como haremos en este curso, aunque tiene también API en otros lenguajes como C++ o Java. Sin embargo, la mayor parte de las operaciones y del núcleo de TF están programados en C++ para una mayor velocidad de ejecución.

Básicamente, TensorFlow funciona mediante la construcción de un grafo de computación y el uso de una sesión que ejecuta las operaciones en el grafo. Con la definición y construcción del grafo, el *framework* obtiene la información necesaria sobre qué operaciones han de ejecutarse, en qué orden y, además, se asegura de su correcta definición y compatibilidad. Una vez el grafo se ha construido, basta con ejecutar las operaciones con los datos necesarios.

TensorFlow y las otras librerías vistas en este tema son *frameworks* muy potentes y versátiles, capaces de implementar a bajo nivel toda suerte de redes neuronales innovadoras; por «bajo nivel» entendemos el nivel de operaciones matemáticas y de arquitectura de red. Sin embargo, en muchas ocasiones queremos utilizar una arquitectura estándar que nos gustaría definir rápidamente, evitando en la medida de lo posible tener que reescribir el código de los mismos elementos una y otra vez. Es por esto que casi todos los *frameworks* aquí mencionados disponen de API de alto nivel que permiten la definición de redes neuronales de una manera más sencilla y directa.

Keras es una de estas librerías de alto nivel, probablemente la primera y más utilizada de ellas. Keras especifica una interfaz modular y *user-friendly* en Python para el desarrollo de redes neuronales, facilitando la tarea a gran parte de los desarrolladores que no necesitan

definir arquitecturas de bajo nivel. Internamente, Keras funciona sobre TensorFlow, aunque también es posible usar como *backend* otros *frameworks* como Theano y CNTK.

Por último, cabe destacar un *framework* más llamado Pytorch. Evolucionado a partir de Torch, y desarrollado en Python (como su nombre deja entrever) para hacerlo más accesible a desarrolladores e investigadores, es otra librería de aprendizaje automático y profundo de código abierto, utilizado habitualmente para el desarrollo de aplicaciones con componentes de visión artificial y procesamiento de lenguaje natural, entre otros.

4.2.2. Repositorios de modelos pre-entrenados

TF HUB

TensorFlow Hub es un repositorio de modelos de aprendizaje automático entrenados, listos para optimizarlos e implementarlos donde se quiera. Se pueden reutilizar modelos entrenados, como BERT y Faster R-CNN, con solo unas pocas líneas de código.

La librería de modelos de TensorFlow Hub proporciona la clase hub.KerasLayer que permite al usuario inicializar con una URL (o ruta de sistema de archivos) modelo guardaro previamente para, posteriormente, realizar cálculos, inferencias y reentrenamientos.

De Tensorflow Hub he obtenido los modelos preentrenados:

- SSD Mobilnet Lite: *SSD Mobilenet V2 Object detection model with FPN-lite feature extractor, shared box predictor and focal loss, trained on COCO 2017 dataset with trainning images scaled to 640x640*.
- SSD Resnet101: *Retinanet (SSD with Resnet 101 v1) Object detection model, trained on COCO 2017 dataset with trainning images scaled to 1024x1024*.
- Faster R-CNN: *Faster R-CNN with Resnet-101 V1 Object detection model, trained on COCO 2017 dataset with trainning images scaled to 1024x1024*.

Como podemos observar, estos modelos están entrenados con el dataset COCO 2017 (Lin et al., 2014)

ULTRALYTICS/yolov3

Este repositorio representa la investigación de código abierto de Ultralytics sobre métodos futuros de detección de objetos e incorpora lecciones aprendidas y mejores prácticas desarrolladas durante miles de horas de capacitación y evolución en conjuntos de datos de clientes anonimizados.

De este repositorio se han utilizado los modelos Yolov3 y Yolov3-tiny, desarrollados sobre Keras y Pytorch, y entrenados también sobre el dataset COCO 2017.

DARKNET

Darknet es un marco de trabajo de redes neuronales de código abierto, escrito en C y CUDA. Esto permite poder realizar ejecuciones tanto en CPU como en GPU. Esta es la razón por la que utilizaremos este repositorio para aplicar *Transfer Learning* y entrenar nuestro modelo en GPU.

5. Descripción de la herramienta software desarrollada

En este apartado vamos a describir el proceso llevado a cabo para realizar la comparativa de los modelos de *Deep Learning* en estudio, empezando por la misma confección del dataset a evaluar para realizar dicha comparación, pasando por la evaluación e inferencia de los modelos sobre las imágenes y viendo distintas métricas resumen sobre cómo detectan tanto bicicletas como personas, hasta la conclusión final y selección de un modelo ganador.

5.1. Criterios de comparación

Nuestro problema consiste en aplicar *Transfer Learning* para detectar ciclistas partiendo de un modelo que sea capaz de reconocer en una imagen multitud de categorías, entre ellas bicicletas y personas.

Nos interesa, por lo tanto, ver cómo se comporta cada modelo a la hora de detectar las categorías 'bicicleta' y 'persona', pues cuanto mejor sea esa detección y lo que el modelo sabe sobre el reconocimiento de esas categorías, más conocimiento podremos transferir a nuestra solución final.

Por la propia naturaleza de nuestro objetivo, nos interesa también un modelo capaz de realizar una inferencia rápida, cumpliendo requisitos de *near real time*.

Por lo tanto, en resumen, nuestros criterios para decidir con qué modelo vamos a continuar serán los siguientes:

- Efectividad detectando categoría 'bicicleta'.
- Efectividad detectando categoría 'persona'.
- Tiempo esperado de ejecución de inferencia sobre una imagen.

5.2. Construcción del conjunto de datos de evaluación

Para realizar la comparación de los citados modelos de detección de objetos, se ha confeccionado un conjunto de imágenes constituido por 1.726 imágenes, en las cuales podemos observar las siguientes casuísticas:

- 675 imágenes de bicicletas. En cada una de estas imágenes podemos ver una bicicleta sobre fondo natural o artificial, en perspectivas variadas.

- 502 imágenes de personas. En cada imagen sale al menos una persona de forma clara, en perspectivas variadas

- 549 imágenes de otras cosas (cualquier cosa excluida de los tres conjuntos anteriores, por ejemplo, un pez, un árbol, un coche, una catedral…)

El objetivo con estas imágenes es realizar sobre ellas la inferencia de los modelos a comparar, teniendo dos categorías de interés inicial (bicicleta y persona), y una categoría para ver si los modelos dan falsas alarmas (otros).

También se han recopilado imágenes de una última categoría, la que se utilizará para la aplicación del *Transfer Learning* en el reentrenamiento del modelo que resulte ganador tras la comparación. Este conjunto de datos consta de:

- 413 imágenes de ciclistas. En cada imagen sale una persona montando en bici, en perspectivas variadas, donde el fondo de la imagen es el habitual para un ciclista: carreteras o senderos.

Recordemos que los modelos a comparar han sido entrenados con el dataset de imágenes COCO 2017, por lo que la evaluación debe hacerse con imágenes distintas para no comprometer los resultados y poder obtener datos representativos. Para poder garantizar esa distinción y variedad, se han utilizado varias fuentes de imágenes:

- https://opensource.google/projects/open-images-dataset: La fuente de datos *open source* ofrecida por Google. De aquí se obtuvieron las imágenes de personas y de otras cosas.

- http://maviintelligence.com/bicycle-image-dataset/ Un dataset con más de 4000 imágenes de bicicletas. De aquí se obtuvieron las imágenes de bicicletas.

- Las imágenes de ciclistas son de obtención propia, tanto tomadas por mí como tomadas o protagonizadas por compañeros y amigos de mi equipo de triatlón.

Todas las imágenes han sido revisadas manualmente, para asegurar que el etiquetado es correcto y que los resultados son representativos.

5.3. Comparación y selección de modelos

Como decíamos con anterioridad, los cinco modelos están entrenados para detectar bicicletas y personas, pero no solo eso sino que pueden detectar hasta 78 clases más de objetos en una imagen. Estas clases detectables son las siguientes:

Class	Label	Class	Label
1	person	41	wineglass
2	bicycle	42	cup
3	car	43	fork
4	motorcycle	44	knife
5	airplane	45	spoon
6	bus	46	bowl
7	train	47	banana
8	truck	48	apple
9	boat	49	sandwich
10	trafficlight	50	orange
11	firehydrant	51	broccoli
12	stopsign	52	carrot
13	parkingmeter	53	hotdog
14	bench	54	pizza
15	bird	55	donut
16	cat	56	cake
17	dog	57	chair
18	horse	58	couch
19	sheep	59	pottedplant
20	cow	60	bed
21	elephant	61	diningtable
22	bear	62	toilet
23	zebra	63	tv
24	giraffe	64	laptop
25	backpack	65	mouse
26	umbrella	66	remote
27	handbag	67	keyboard
28	tie	68	cellphone
29	suitcase	69	microwave
30	frisbee	70	oven
31	skis	71	toaster
32	snowboard	72	sink
33	sportsball	73	refrigerator
34	kite	74	book
35	baseballbat	75	clock
36	baseballglove	76	vase
37	skateboard	77	scissors
38	surfboard	78	teddybear
39	tennisracket	79	hairdrier
40	bottle	80	toothbrush

Ilustración 17: Clases detectadas por los modelos en estudio.

Dado que nuestro caso de uso va a ser detectar ciclistas y buscamos aplicar la transferencia de conocimiento que los modelos tienen en la detección de estas 80 clases, nos interesa evaluar dichos modelos en base a cómo detectan esas dos de esas categorías en concreto: bicicletas y personas.

Posteriormente, vamos a tener en cuenta también el tiempo de ejecución de la inferencia en una imagen, ya que el objetivo a futuro es implantar este modelo de detección de ciclistas en un sistema complejo de detección de objetos capaz de enviar notificaciones en tiempo real.

Para evaluar todas estas condiciones, primero vamos a realizar la inferencia con cada uno de los modelos sobre nuestro dataset, y con los resultados obtenidos realizaremos un estudio de la efectividad de cada modelo sobre las detecciones de bicicletas y personas, así como un análisis de la distribución de tiempos de inferencia de cada uno de ellos.

Al final, obtendremos para cada modelo los siguientes indicadores (Fernández, 2013):

- ***Precision***: la precisión es el ratio o porcentaje de clasificaciones correctas de nuestro clasificador, es decir, de todo lo que nuestro clasificador clasifica como positivo, correcta o incorrectamente (TP + FP), cual es la proporción de clasificaciones correctas.

$$PRECISION = \frac{TP}{TP + FP}$$

- ***Sensitivity / True positive rate / Recall***: el *recall* o sensibilidad de nuestro modelo es el ratio de positivos detectado en el dataset por nuestro clasificador. En otras palabras, de todos los positivos reales de nuestro dataset, detectados o no (TP + FN), cual es la proporción de positivos detectados.

$$SENSITIVITY = RECALL = \frac{TP}{TP + FN}$$

- ***Specificity*** (1 - FPR): La especificidad de nuestro modelo es el ratio de negativos detectados por nuestro clasificador, es decir, de todos los negativos reales cuantos ha detectado nuestro modelo como tal

$$SPECIFICITY = 1 - FPR = 1 - \frac{FP}{FP + TN} = \frac{TN}{FP + TN}$$

- ***Accuracy***: Por último, uno de los indicadores más utilizados como métrica resumen de la efectividad de un modelo, que se puede entender como el ratio de cuantas

clasificaciones ha hecho bien (tanto positivas como negativas) de entre todos los elementos del dataset (tanto positivos como negativos)

$$ACCURACY = \frac{TP + TN}{P + N}$$

Con ellos, obtenemos las siguientes métricas y gráficas:

- **Curva ROC y AUC** (área bajo la curva): Relaciona el *recall* con el ratio de falsos positivos. Es decir, relaciona la sensibilidad de nuestro modelo con los fallos optimistas (clasificar los negativos como positivos).
- **Curva Precision-Recall y AUC**: Relaciona la *precision* y el *recall*. Esta gráfica nos permite ver a partir de qué *recall* tenemos una degradación de la precisión, y viceversa. Lo ideal sería una curva que se acerque lo máximo posible a la esquina superior derecha (alta precisión y alto *recall*).

5.3.1. Detección de personas

Para realizar la comparación en lo respectivo a la detección de personas, se ha transformado el problema multiclase en uno de tipo 'one vs others', de forma que se comparan las clases Persona (positivo) vs No persona (negativo).

- **Yolo v3**
  - **Precision:** 0.80
  - **Recall:** 0.99
  - **Accuracy:** 0.93

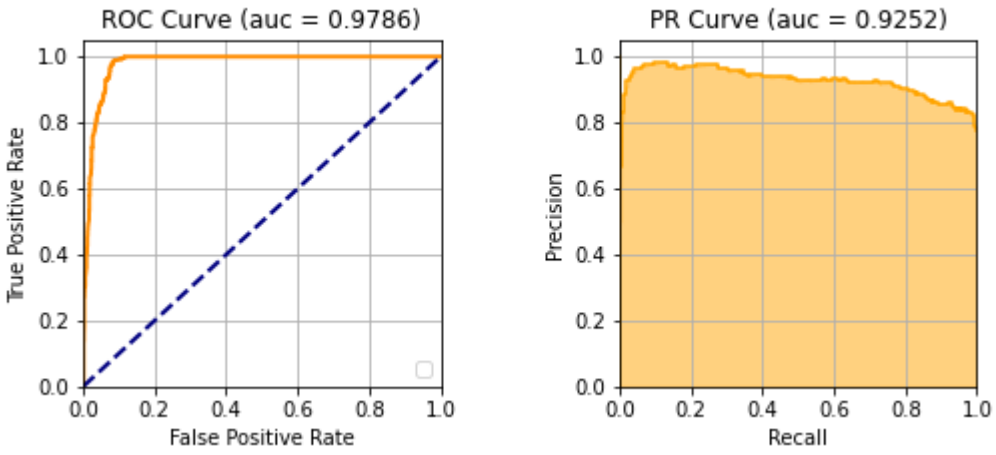

Ilustración 18: Métricas de Yolo v3 en la detección de personas.

- **Yolo v3 Tiny**
  - **Precision:** 0.83
  - **Recall:** 0.60
  - **Accuracy:** 0.85

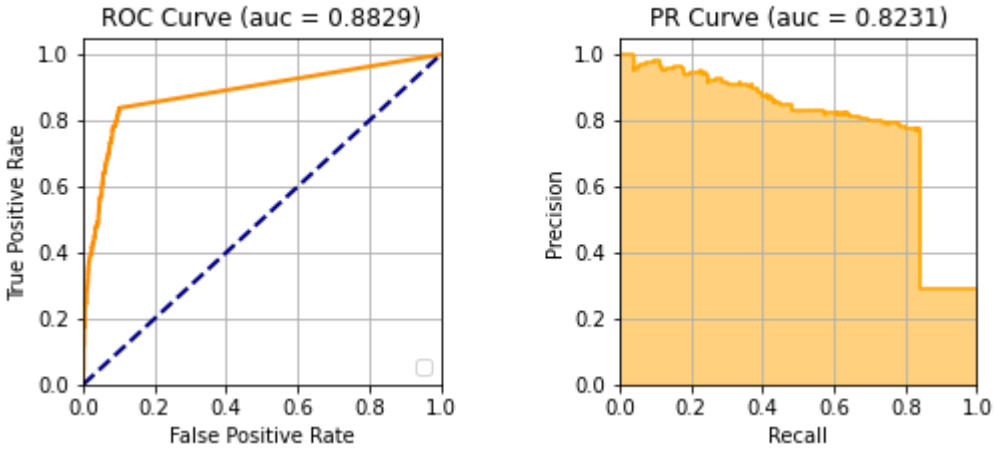

Ilustración 19: Métricas de Yolo v3 Tiny en la detección de personas.

- **SSD ResNet101**
  - **Precision:** 0.80
  - **Recall:** 0.99
  - **Accuracy:** 0.93

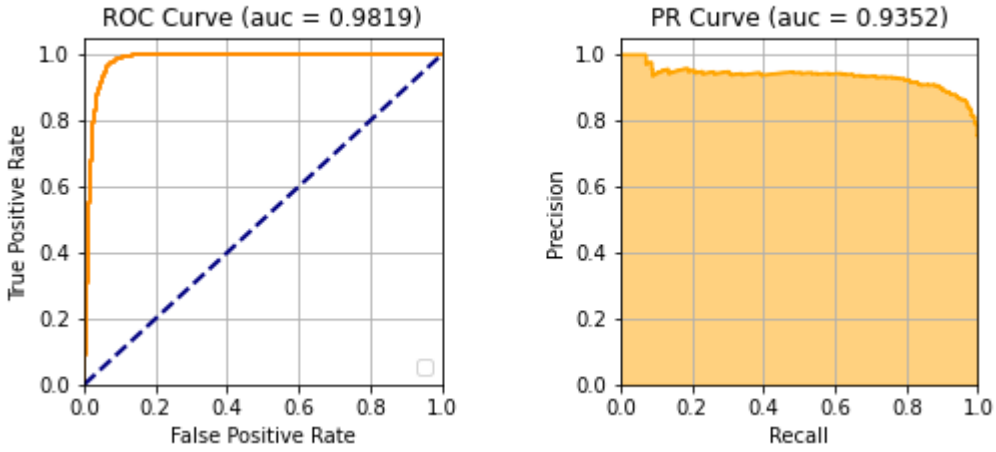

Ilustración 20: Métricas de SSD ResNet101 en la detección de personas.

- **SSD Mobilnet Lite**
  - **Precision:** 0.84
  - **Recall:** 0.99
  - **Accuracy:** 0.94

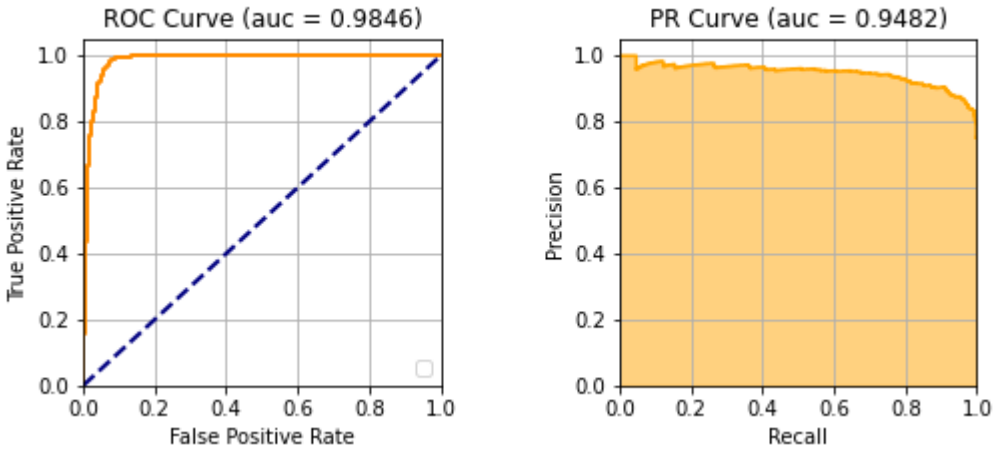

Ilustración 21: Métricas de SSD Mobilnet Lite en la detección de personas.

- **Faster R-CNN**
  - **Precision:** 0.75
  - **Recall:** 1
  - **Accuracy:** 0.90

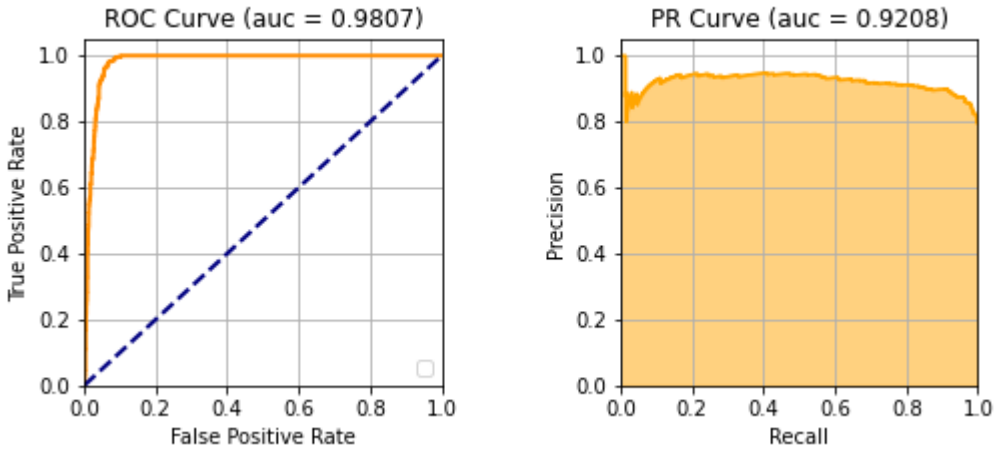

Ilustración 22: Métricas de Faster R-CNN en la detección de personas.

Por otro lado, podemos analizar también las distribuciones de los *scorings* de cada modelo a la hora de determinar si en una imagen aparece una persona o no. Podemos entender esto como 'cuán seguro está un modelo de que algo es una persona'.

Para ello, vamos a ver las probabilidades de la clase Persona para las imágenes cuya etiqueta real es Persona:

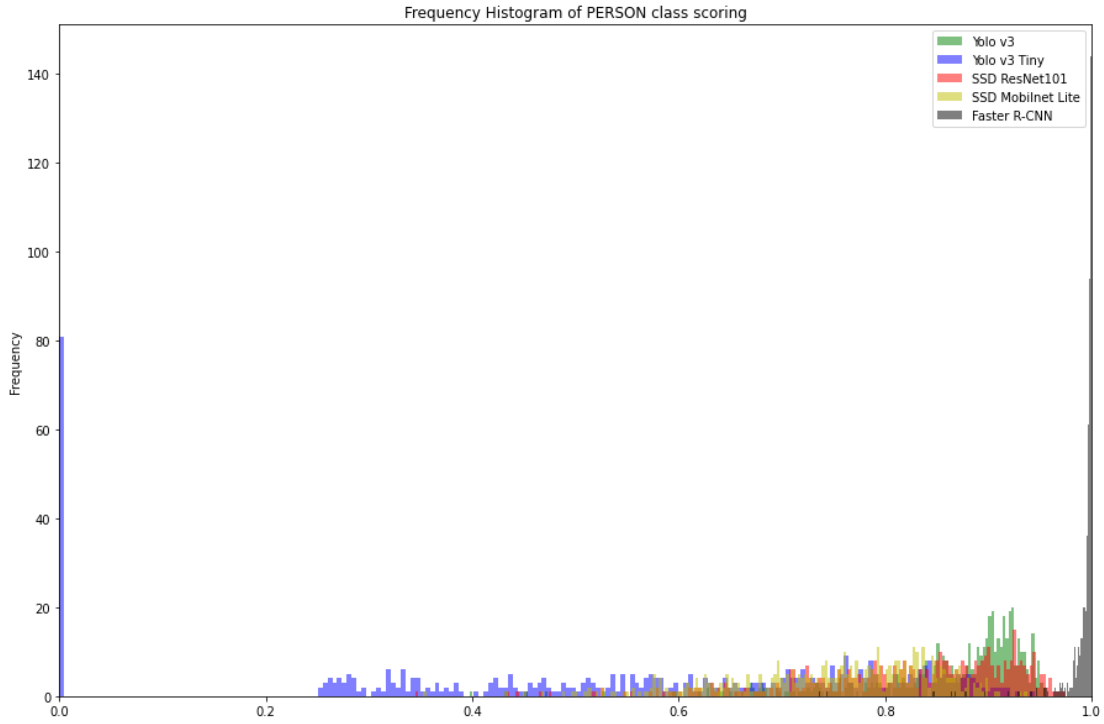

Ilustración 23: Densidades de probabilidades de los modelos en la detección de personas.

En este caso, vemos que el modelo que detecta personas 'con más seguridad' es, sin duda, Faster R-CNN, ya que para la clase Persona, otorga probabilidades de persona muy cercanas a 1 y muy agrupadas, hay poca varianza,

El siguiente modelo en cuanto a 'seguridad en la decision' es Yolo v3, vemos como otorga probabilidades también bastante altas (no tanto como Faster R-CNN) y bastante agrupadas.

Vemos también como el resto de modelos van perdiendo seguridad: SSD Resnet otorga probabilidades similares a Yolo v3 pero más dispersas, es decir, duda un poco más. SSD Mobilnet Lite va con probabilidades algo más bajas y también algo más dispersas, hasta Yolov3 Tiny, que otorga probabilidades muy bajas y muy dispersas.

Además, para Yolov3 Tiny vemos que hay una cantidad importante de 0, a pesar de que la etiqueta real es Persona.

5.3.2. Detección de bicicletas

Para realizar la comparación en lo respectivo a la detección de bicicletas, se ha transformado el problema multiclase en uno de tipo 'one vs others', de forma que se comparan las clases Bicicleta (positivo) vs No bicicleta (negativo).

- **Yolo v3**
  - **Precision:** 0.99
  - **Recall:** 0.99
  - **Accuracy:** 0.99

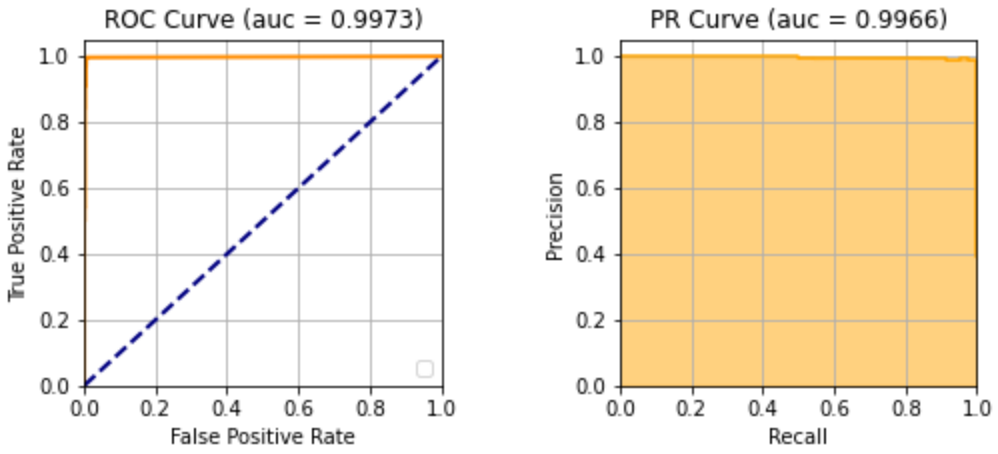

Ilustración 24: Métricas de Yolo en la detección de bicicletas.

- **Yolo v3 Tiny**
  - **Precision:** 0.99
  - **Recall:** 0.16
  - **Accuracy:** 0.67

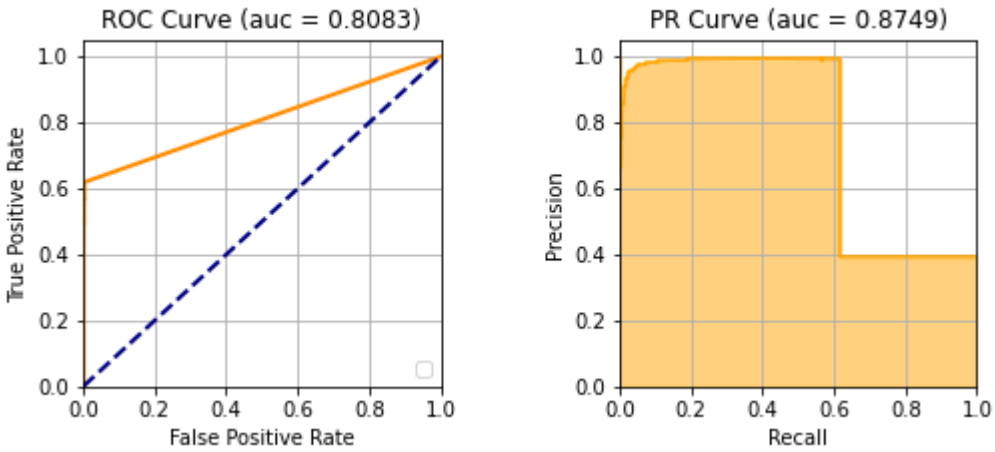

Ilustración 25: Métricas de Yolo v3 Tiny en la detección de bicicletas.

- **SSD ResNet101**
  - **Precision:** 0.99
  - **Recall:** 0.93
  - **Accuracy:** 0.97

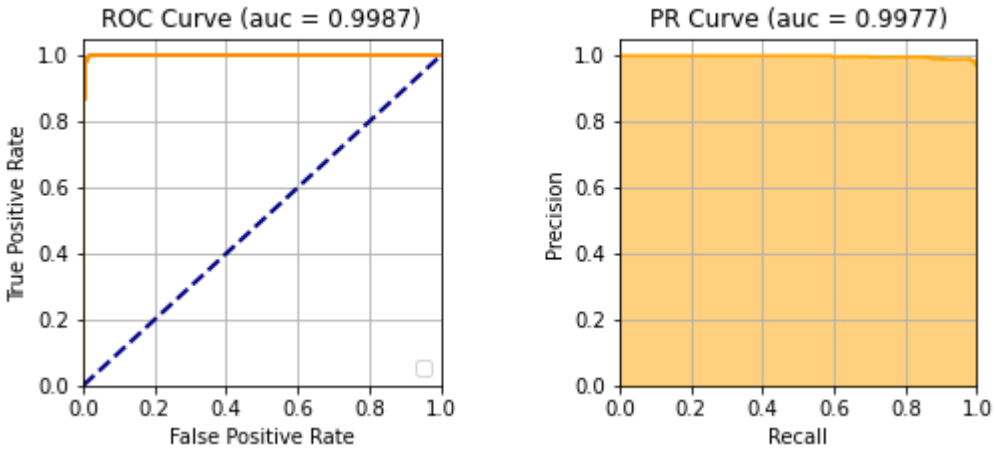

Ilustración 26: Métricas de SSD ResNet101 en la detección de bicicletas.

- **SSD Mobilnet Lite**
  - **Precision:** 0.99
  - **Recall:** 0.95
  - **Accuracy:** 0.98

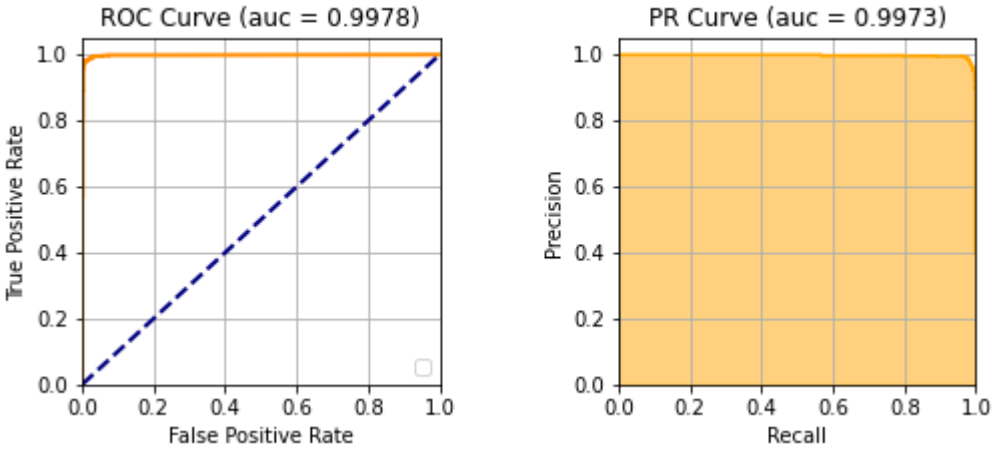

Ilustración 27: Métricas de SSD Mobilnet Lite en la detección de bicicletas.

- **Faster R-CNN**
  - **Precision:** 0.99
  - **Recall:** 0.99
  - **Accuracy:** 0.99

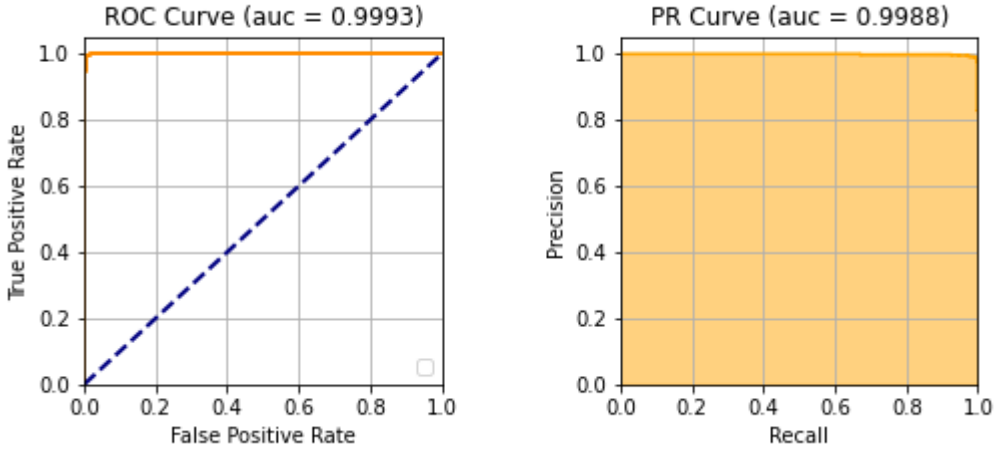

Ilustración 28: Métricas de Faster R-CNN en la detección de bicicletas.

Vamos a analizar en este caso también las distribuciones de los *scorings* de cada modelo a la hora de determinar si en una imagen aparece una bicicleta o no. Podemos entender esto, al igual que en el caso anterior, como 'cuán seguro está un modelo de que algo es una bicicleta.

Para ello, vamos a ver las probabilidades de la clase Bicicleta que otorgan los modelos a las las imágenes cuya etiqueta real es Bicicleta:

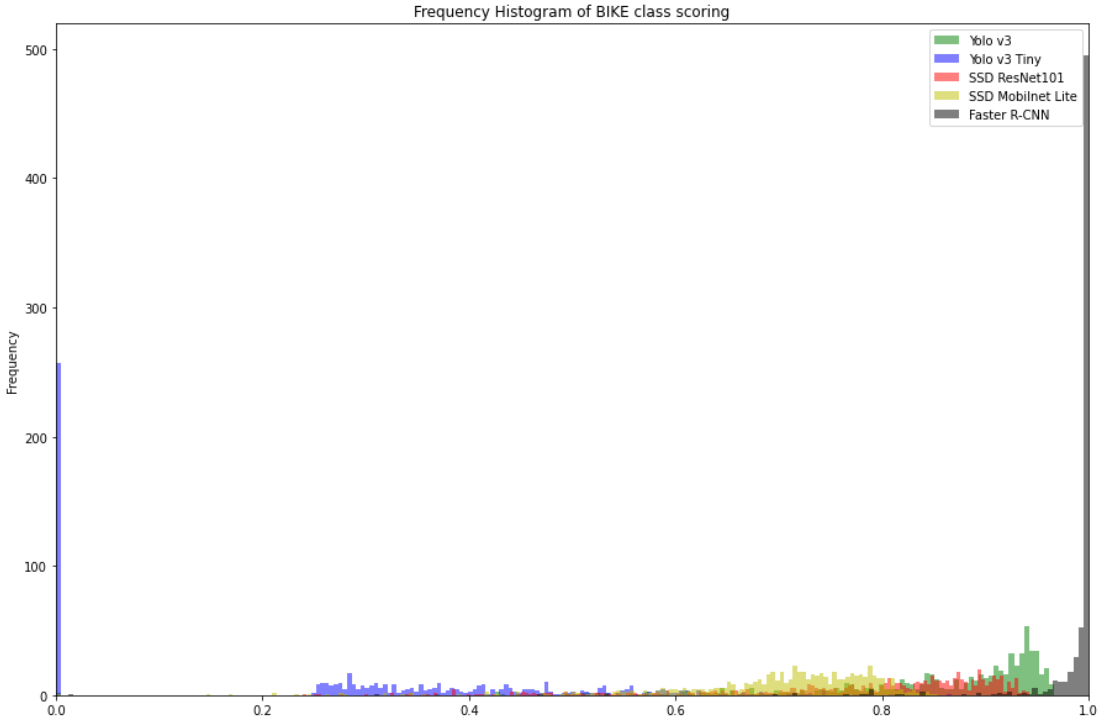

Ilustración 29: Densidades de probabilidades de los modelos en la detección de bicicletas.

Lo que vemos en esta gráfica es muy similar a lo que veíamos en la gráfica análoga para la clase Persona.

Vemos que el modelo que otorga probabilidades más altas a la clase bicicleta es, igualmente, Faster R-CNN, y con menos varianza que el resto de modelos.

El siguiente modelo en cuanto a 'seguridad en la decision' es Yolo v3, ya que vemos como otorga probabilidades también bastante altas (no tanto como Faster R-CNN) y casi tan agrupadas como Faster R-CNN.

El resto de modelos van perdiendo seguridad: SSD Resnet otorga probabilidades similares aunque ligeramente inferiores a Yolo v3, y también más dispersas, es decir, duda un poco más. SSD Mobilnet Lite va con probabilidades algo más bajas y también algo más dispersas, hasta Yolov3 Tiny, que otorga probabilidades muy bajas y muy dispersas.

Vemos de igual modo que Yolov3 Tiny otorga probabilidad 0 a un número de casos importante.

5.3.3. Tiempos de ejecución

Recordemos que, para el caso de uso real de nuestro problema, un factor importante es el tiempo de ejecución de la inferencia del modelo sobre la imagen. Si estamos intentando detectar un ciclista a medida que vamos conduciendo para enviar una notificación a un entorno compartido, simplemente por sentido común concluimos que dicha inferencia ha de ser rápida, en tiempo real, o casi en tiempo real.

Para ello, vamos a observar en la siguiente gráfica la distribución de tiempos de ejecución de inferencia para cada modelo:

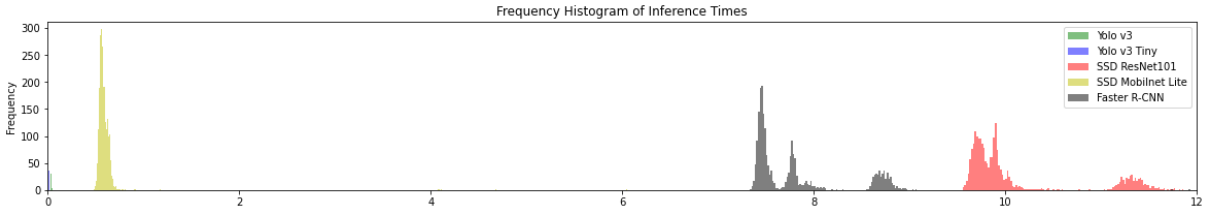

Ilustración 30: Densidades de tiempos de ejecución de los modelos al realizar la inferencia.

Observamos que hay dos modelos que tardan demasiado para el objetivo que tenemos: Faster R-CNN y SSD ResNet101, que tardan del orden de los 8 segundos y 10 segundos para cada inferencia, respectivamente.

Modelo	Media	Mediana
Yolo v3	0,040	0,038
Yolo v3 Tiny	0,014	0,014
SSD ResNet101	10,124	9,861
SSD Mobilnet Lite	0,591	0,577
Faster R-CNN	7,810	7,537

Ilustración 31: Estadísticos muestrales de los tiempos de inferencia de los modelos.

Esto los hace incompatibles con nuestro problema y la solución que buscamos.

Por otro lado, tenemos tres modelos que emplean menos de un segundo por cada inferencia. Con esto, podríamos gestionar el número de *frames* por cada segundo que evaluar, de forma que:

- Con SSD Mobilnet Lite, dado que tarda casi 0.6 segundos de media, podríamos realizar una inferencia por segundo, o tres inferencias cada dos segundos. Esto podría acercarnos a un proceso de tipo casi tiempo real.

- Con Yolov3 obtenemos casi 0,04 segundos por inferencia, con lo que podríamos realizar unas 25 inferencias por segundo, algo que ya podríamos considerar evaluaciones en tiempo real.

- Con Yolov3 Tiny, en cuanto a tiempos de ejecución, obtenemos el mejor resultado ya que con una media de 0,014 podriamos llegar a realizar unas 70 inferencias por segundo.

5.3.4. Conclusiones

A la vista de los resultados obtenidos en la evaluación de los modelos anteriormente descritos en este dataset, podemos concluir varios datos importantes.

En cuanto a la detección de personas, con los datos obtenidos podemos concluir:

PERSONAS	PRECISION	RECALL	ACCURACY	ROC AUC	PR AUC
Yolo v3	0,80	0,99	0,93	0,97	0,92
Yolo v3 Tiny	0,83	0,60	0,85	0,88	0,82
SSD ResNet101	0,80	0,99	0,93	0,98	0,93
SSD Mobilnet Lite	0,84	0,99	0,94	0,98	0,94
Faster R-CNN	0,75	1,00	0,90	0,98	0,92

Ilustración 32: Resumen de las métricas obtenidas en la evaluación de los modelos para la detección de personas.

- Podemos ver que el modelo SSD Mobilnet Lite tiene, en general, los valores mayores tanto en las métricas como en las curvas, y le sigue muy de cerca el modelo Yolo v3, con unas métricas muy similares (valoradas en conjunto).

- Sobre el resto de modelos, podemos destacar que Faster R-CNN ha sido excelente a la hora de reconocer los positivos reales (es decir, personas) ya que las ha categorizado todas como tal (*recall* = 1) mientras que por otro lado, 1 de cada 4 positivos que ha señalado ha sido falso (no hay persona), es decir, un negativo (precisión = 0.75). Esto traducido a un caso de uso real de notificación al detectar personas, significaría que por cada 4 notificaciones que se realicen, 1 es errónea.

- Por último, cabe destacar la mala métrica de Yolo v3 Tiny en cuanto a recall, con valor 0.6. Esto vendría a significar que de todos los positivos, es decir, de todas las personas, detecto poco más de la mitad.

En cuanto a la detección de bicicletas, con los datos obtenidos podemos concluir:

BICICLETAS	PRECISION	RECALL	ACCURACY	ROC AUC	PR AUC
Yolo v3	0,99	0,99	0,99	0,99	0,99
Yolo v3 Tiny	0,99	0,16	0,67	0,80	0,87
SSD ResNet101	0,99	0,93	0,97	0,99	0,99
SSD Mobilnet Lite	0,99	0,95	0,98	0,99	0,99
Faster R-CNN	0,99	0,99	0,99	0,99	0,99

Ilustración 33: Resumen de las métricas obtenidas en la evaluación de los modelos para la detección de bicicletas.

- Cabe destacar el buen desempeño en general de todos los modelos, excepto Yolo v3 Tiny, en cuanto a la detección de bicicletas, algo bastante llamativo en comparación a la detección de personas.

- En este caso, los mejores resultados tanto en métricas como en las gráficas los obtienen, a la par, Faster R-CNN y Yolo v3, ambos con casi un acierto del 100% en todas las métricas.

- Las métricas aquí de Yolo v3 Tiny se desploman, cayendo hasta el 0.16 el recall, lo que significaría que solo detecta una bicicleta real de cada 6 imágenes de bicicletas.

En cuanto a los tiempos de inferencia:

Modelo	Personas	Bicicletas
Yolo v3	0,04	0,04
Yolo v3 Tiny	0,01	0,01
SSD ResNet101	9,84	9,92
SSD Mobilnet Lite	0,6	0,58
Faster R-CNN	7,47	7,78

Ilustración 34: Resumen de los tiempos de inferencia de los modelos.

- Sin duda alguna, para nuestro caso de uso podemos descartar por este criterio los modelos Faster R-CNN y SSD ResNet101, que tardan del orden de los 8 segundos y 10 segundos para cada inferencia, respectivamente.

- Los modelos cuyos tiempos de inferencia son mejores son los basados en arquitectura YOLO, cuyos tiempos de ejecución son del orden de centésima de segundo, seguidos de cerca (en comparación con el resto) por SSD Mobilnet Lite, cuyos tiempos de ejecución son del orden de décima de segundo.

Como conclusión final, y partiendo de toda la información que nos otorgan estos resultados, descartamos directamente Faster R-CNN y SSD ResNet101 por los altos tiempos de inferencia, y Yolo v3 Tiny por sus malos resultados obtenidos en la evaluación de las imágenes de ambas clases.

Por último, la decisión entre SSD MobilNet Lite y Yolo v3 la tomaremos a favor de este último, ya que, aunque ambos tienen excelentes resultados en la detección de ambas clases, el tiempo de inferencia de Yolo v3 es 14 veces menor que el de SSD MobilNet Lite. Esto supone una diferencia de orden de magnitud suficientemente elevado como para asumir que la diferencia de rendimiento es transferible a un dispositivo de test en el que se evaluará el futuro modelo de detección de ciclistas, al mismo tiempo que garantiza que las pruebas en un dispositivo con hardware no diseñado específicamente para este problema puedan llevarse a cabo dentro de la definición de *real-time* (véase capítulo 6).

5.4. Data Augmentation

Como comentábamos en capítulos previos, las técnicas de aumento de datos son muy populares y habituales ya que suponen, principalmente, dos beneficios importantes:

- Por un lado, el aumento de datos en sí mismo. El hecho de poder obtener un dataset de entrenamiento x2, x4, x10… con respecto al dataset originario de que disponemos, supone que podamos realizar un entrenamiento más completo y consistente en nuestra red neuronal.

- Por otro lado, provoca un efecto regularizador en la efectividad de la red neuronal, ya que el hecho de crear pequeñas perturbaciones o modificaciones de las imágenes originales provoca en la red que no se fije o aprenda aspectos muy concretos de nuestro dataset que puedan llevar a *overfitting*.

Para realizar este proceso de Data Augmentation vamos a utilizar la clase *Image Data Generator* existente en Keras, uno de los *framework* de *Deep Learning* más conocidos y utilizados.

Con esta función podemos realizar diversos tipos de modificaciones, alteraciones o transformaciones a las imágenes. Vamos a verlo con un ejemplo con la siguiente imagen.

Ilustración 35: Imagen ejemplo para Data Augmentation.

Esta utilidad de Keras nos permite elegir qué tipo de transformación realizar, y dentro de cada tipo también podemos escoger la magnitud de dicha transformación.

Transformaciones geométricas que podemos realizar:

- Rotación: podemos elegir el ángulo máximo de rotación de la imagen.
- Anchura: modifica ligeramente el aspecto de la imagen en anchura.
- Altura: modifica ligeramente el aspecto de la imagen en altura.
- Zoom: amplia o disminuye la imagen.
- Simetría horizontal: voltea la imagen en modo ‘espejo’ en dirección horizontal.

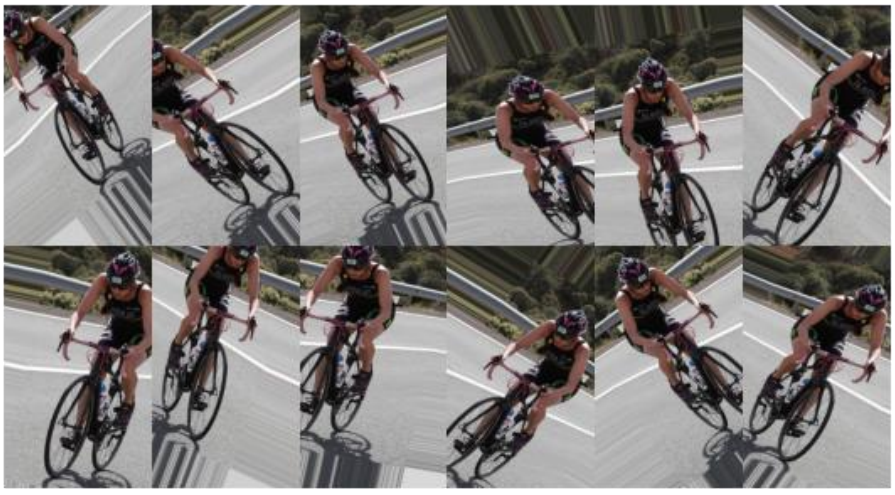

Ilustración 36: Transformaciones Data Augmentation - fase 1.

Transformaciones de color que podemos realizar:

- Rango de colores: estirar o encoger el rango de cada capa de color RGB (roja, verde, o azul) un factor aleatorio propuesto.
- Contraste: estirar o encoger el rango de contraste en un factor aleatorio propuesto.

Ilustración 37: Transformaciones Data Augmentation - fase 2.

Finalmente, optamos por realizar todas estas transformaciones juntas para tener mayor variedad, con lo que obtendríamos el siguiente rango de imágenes por cada una de las imágenes de nuestro dataset original:

Ilustración 38: Transformaciones Data Augmentation - final.

Cabe destacar que, si bien la palabra "aumentar" significa hacer algo "mayor" (en este caso, datos), la clase Keras ImageDataGenerator en realidad funciona de la siguiente manera:

1. Acepta un lote de imágenes utilizadas para el entrenamiento.
2. Tomando este lote y aplicando una serie de transformaciones aleatorias a cada imagen en el lote (incluyendo rotación aleatoria, cambio de tamaño, corte, etc.).
3. Reemplazo del lote original con el nuevo lote transformado aleatoriamente.
4. Entrenamiento de la CNN en este lote transformado al azar (es decir, los datos originales en sí no son utilizados para el entrenamiento).

Así es, la clase ImageDataGenerator de Keras no es una operación "aditiva". No se trata de tomar los datos originales, transformarlos aleatoriamente y luego devolver tanto los datos originales como los transformados. En su lugar, ImageDataGenerator acepta los datos originales, los transforma aleatoriamente y devuelve solo los datos nuevos transformados.

5.5. Transfer Learning

Como se ha indicado en apartados anteriores, la técnica de *Transfer Learning* consiste, conceptualmente, en aplicar el conocimiento obtenido a partir de la solución de un problema en la solución de otro problema de índole similar.

En el caso que nos ocupa, aplicar esta técnica es más que legítimo, puesto que partimos de soluciones que detectan tanto bicicletas como personas en imágenes, y nuestro nuevo problema trata de detectar ciclistas, que no dejan de ser ‘bicicletas y personas juntas’. Es decir, en este caso está más que justificado que podamos utilizar el conocimiento ya aprendido por los modelos preentrenados para obtener un buen modelo que resuelva nuestra problemática.

Para realizar el entrenamiento de nuestra red neuronal sobre Yolo v3, partimos del modelo Yolo v3 existente en el repositorio Darknet. Este repositorio es de código abierto y está desarrollado sobre C y CUDA, lo que nos permitirá realizar un entrenamiento sobre GPU, agilizando la tarea, para lo que utilizaremos el dispositivo Jetson Nano de NVIDIA comentado en párrafos anteriores.

5.5.1. Anotación del dataset

Para entrenar nuestro modelo, lo primero que necesitamos tener es un dataset debidamente anotado, y en el formato que Yolo v3 requiere. Para ello, utilizamos una herramienta llamada Label Img, que nos permite visualizar las imágenes de nuestro dataset, una por una, indicando

en cada una de ellas la porción de imagen que corresponde al objeto que queremos detectar (en este caso, el ciclista) y generando, de forma automática, el fichero de anotación correspondiente.

1- Seleccionamos en la imagen dónde se encuentran los objetos a detectar y seleccionamos en la parte derecha de la aplicación la clase a la que pertenecen, en este caso ciclistas (única clase en estudio):

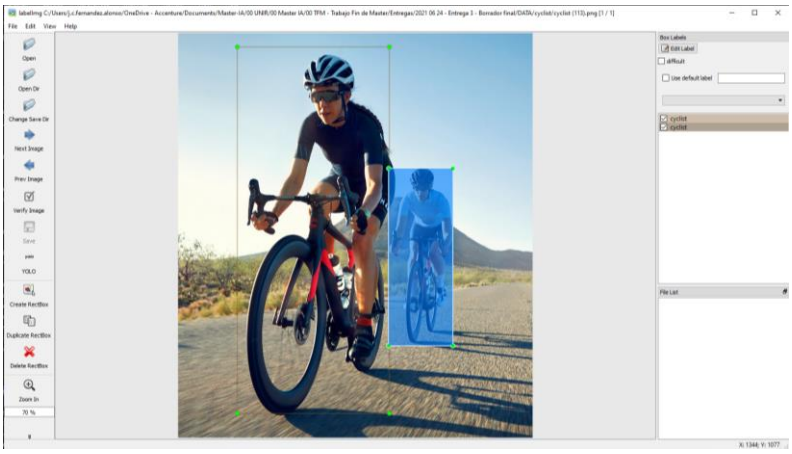

Ilustración 39: Herramienta para el etiquetado del conjunto de datos.

2- Al guardar el etiquetado, la aplicación genera un fichero txt asociado a la imagen, con el mismo nombre que la imagen, que contiene una línea por cada objeto indicado en la imagen, en la que se puede leer la clase (en formato numérico) y las coordenadas del recuadro en formato *centro x, centro y, anchura, altura* (normalizadas en 0-1 con respecto al tamaño de la imagen):

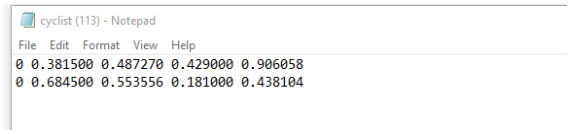

Ilustración 40: Etiquetas generadas de una imagen.

3- Además, tenemos que generar un fichero classes.txt que contendrá el nombre de cada clase, en este caso ciclista:

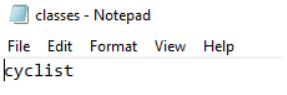

Ilustración 41: Fichero de clases del modelo.

Este tipo de etiquetado es necesario porque una de las métricas en las que se basa el entrenamiento de este modelo es el *Intersection Over Union* (IoU).

La métrica IoU es una métrica de evaluación que se utiliza habitualmente en el ámbito de detección de objetos, independientemente del modelo utilizado, con la única condición de que la salida de éste sea una caja delimitadora con el objeto detectado.

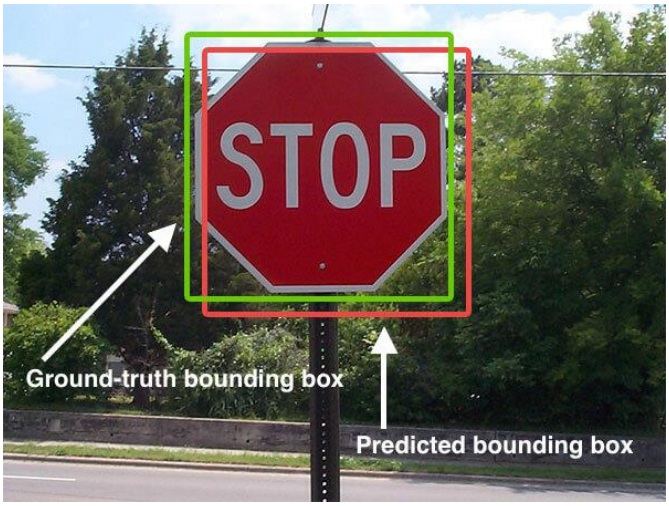

Ilustración 42: Caso práctico de IoU.

Lo que hace esta métrica es comparar la caja delimitadora real (*ground-truth*) con la caja delimitadora de la detección (*predicted*), realizando simplemente una proporción entre la intersección de ambas cajas con respecto a la unión de las mismas:

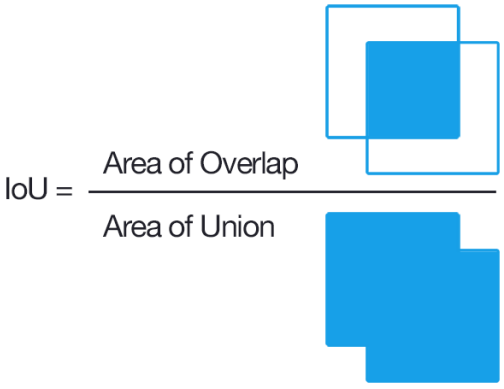

Ilustración 43: Fórmula de IoU.

De esta forma, cuanto mejor coincidan ambas cajas delimitadoras, más cercana a 1 será la métrica IoU:

Ilustración 44: Tipos de IoU.

5.5.2. Re-entrenamiento de la red y transferencia de conocimiento

Para realizar la transferencia de conocimiento de Yolo v3 original y crear un nuevo modelo capaz de detectar ciclistas, revisamos la documentación del repositorio que vamos a utilizar y las indicaciones y recomendaciones del autor (Redmon y Farhadi, 2018), esto nos ayudará a configurar la parametrización del entrenamiento.

En esta ocasión, para realizar el proceso de Transfer Learning se fijan las primeras 75 capas, es decir, conservamos el conocimiento que la red ya tiene en esas capas, y dejamos libres para el reentrenamiento las 32 restantes, divididas en tres bloques predictores.

Estas 32 capas restantes que sí se van a entrenar, las inicializamos también con los pesos de la red Yolo v3 original. Esta recomendación del autor es adecuada en nuestro caso ya que el problema de detección de ciclistas está estrechamente relacionado con la detección de personas y bicicletas que ya realiza la red, con lo que inicializando estas capas con unos pesos que ya tienen sentido e información funcional, teóricamente, llegaremos a un mejor modelo y en menos tiempo que si realizamos una inicialización aleatoria de los mismos.

Tras la realización de numerosas pruebas con distintos valores, los criterios de entrenamiento y de parada del modelo obtenido finalmente son los siguientes:

- Epochs = 2000
- Learning rate = 0.001
- Momentum = 0.9
- Decay = 0.0005
- IoU threshold = 0.75

Y finalmente, tras las 2000 epochs de entrenamiento bajo esos criterios, se construye un modelo con una pérdida media (*avg loss*) de 0.185087.

Con esto, se genera los ficheros de configuración .cfg y de pesos .weights del modelo, necesarios para cargarlo y utilizarlo en cualquier proceso que lo requiera. En nuestro caso, bastará con disponer de una instalación Python compatible y la inclusión de la librería de visión artificial Opencv.

6. Evaluación

El objetivo de este estudio es llegar a crear un producto que pueda ser utilizado en la vida real para la detección temprana de ciclistas en la vía de modo que esto suponga tanto una mejora en la seguridad de los propios ciclistas como una ayuda en atención y visibilización para el resto de usuarios de dicha vía.

En esta sección se muestra el trabajo realizado en esta dirección a partir del modelo obtenido en apartados anteriores para desarrollar un Mínimo Producto Viable (MVP, *Mínimum Valuable Product*) que detecte ciclistas en tiempo real a través de una cámara de tipo *dashcam* y dispare, a su vez, una notificación de presencia de ciclistas en un sistema de alertas compartidas. De esta forma, quedará demostrada su utilidad, así como su potencial futuro mediante determinadas evoluciones y mejoras, que se comentarán en la sección siguiente.

6.1. Aplicación real al caso de uso

Para este MVP, se van a desarrollar varios procesos independientes que, posteriormente, se integrarán para dar lugar a la herramienta final:

1. Un proceso en Python que carga el modelo y ejecuta la inferencia sobre una imagen, detectando ciclistas donde proceda.
2. Un proceso que realiza la tarea en 1. para cada imagen (*frame*) de un vídeo.
3. Un proceso que generaliza 2. para un vídeo tomado en tiempo real desde una webcam.
4. Un disparador de alertas basado en el envío de mensajes http a una herramienta de mensajería instantánea.

6.1.1. Inferencia con el modelo obtenido

Para realizar la inferencia con el modelo obtenido se ha desarrollado un script en Python, que cumple las dependencias requeridas de la librería de visión artificial Opencv, y que carga los ficheros .cfg y .weights del modelo.

Primero se ha desarrollado una versión que realiza la inferencia del modelo de detección de ciclistas sobre una imagen de forma offline, obteniendo resultados satisfactorios sobre imágenes de ciclistas del dataset reservado para test (es decir, imágenes con las que no se ha entrenado el modelo). A continuación, podemos ver algunos ejemplos:

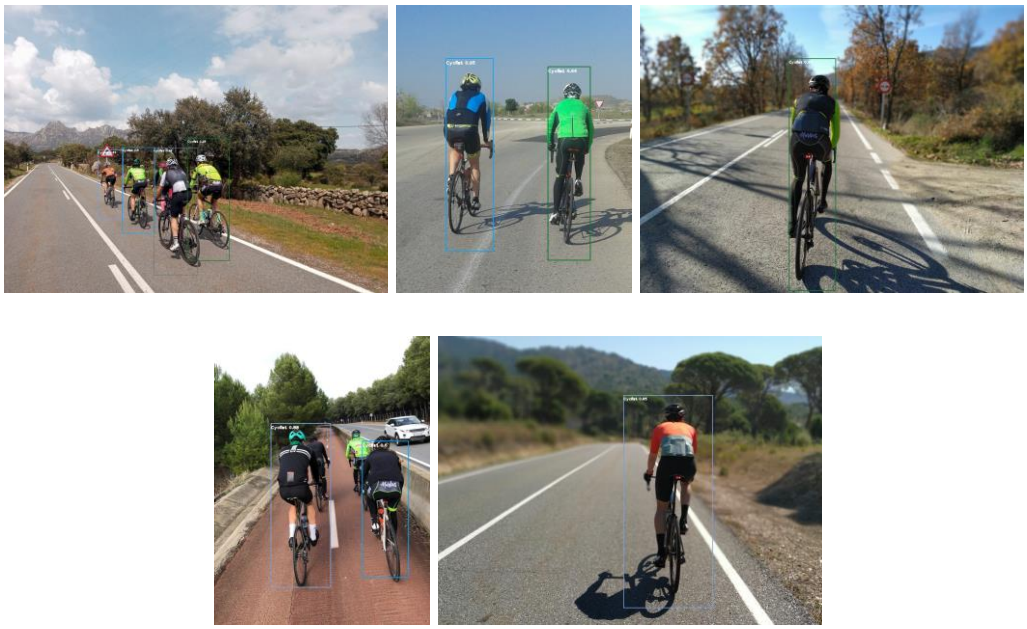

Ilustración 455: Ejemplos de inferencia del modelo obtenido mediante Transfer Learning.

Posteriormente, y siguiendo en la dirección marcada por el MVP, se ha desarrollado un programa que realiza la misma inferencia, pero iterada sobre un conjunto de imágenes finito, de forma offline también, obtenidas de un vídeo.

Por último, se generaliza este proceso añadiendo una función que lee vídeo en tiempo real desde una *webcam*, transformando automáticamente el vídeo en imágenes (para esta prueba y debido a limitaciones de hardware se convierte el vídeo a 3 *frames* por segundo) y se ejecuta la inferencia del modelo por cada uno de esos *frames*.

6.1.2. Dashcam

Una *dashcam* (o *dash cam*) es una cámara convencional que está preparada y diseñada específicamente para ser usada dentro de un coche.

Por esto, es bastante común que vengan acompañadas de un soporte o una ventosa, para instalarlas en el cristal o en el salpicadero del vehículo. Habitualmente se utilizan colocándolas en la parte frontal y enfocando hacia el frente del vehículo, aunque hay quien utiliza dos y la segunda vigila la parte trasera del mismo.

También, actualmente, se fabrican vehículos que ya traen estas cámaras incorporadas y no requieren de su instalación de forma externa.

Para esta prueba de campo se utilizará una cámara EMEET 1080p AUTO FOCUS, una webcam convencional de uso doméstico con conectividad usb al ordenador. Se dispondrá en el salpicadero del vehículo enfocando hacia el frente, conectada al ordenador donde se ejecutará el modelo y se podrá visualizar en la pantalla el proceso en tiempo real de detección de los ciclistas en las imágenes del vídeo, y cómo acto seguido se recibe en la herramienta de mensajería la notificación de alerta.

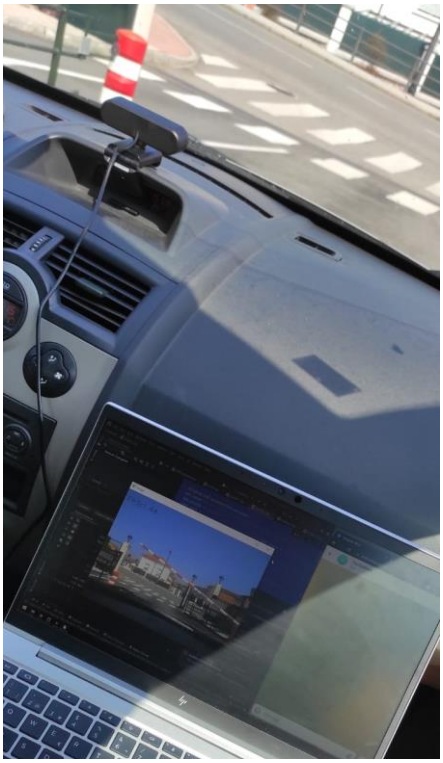

Ilustración 466: Prueba de campo.

Hay que tener en cuenta que éste es un hardware para uso genérico. Cabría esperar una mejora bastante notable de los resultados obtenidos si utilizásemos un dispositivo específicamente diseñado para tal propósito.

6.1.3. Sistema de alertas y notificaciones

Para ejemplificar el envío de alertas a una aplicación de uso compartido cuando se detecta un ciclista se ha utilizado como herramienta de mensajería la llamada Telegram, junto con un paquete de Python llamado *requests*, que permite enviar mensajes por http.

Para ello, se ha creado un bot en Telegram para el uso específico y único de esta prueba, llamado *CyclistAlert*, y se ha conectado con el script de detección de ciclistas mediante las claves http que proporciona el propio Telegram.

Los mensajes de alerta se envían desde el propio módulo de detección de ciclistas implementado, justo después de cada inferencia en la que se ha detectado algún ciclista en la imagen de video a través de la cámara.

A este mensaje de alerta se le ha añadido la ubicación geoespacial de la detección mediante un paquete de Python que se llama *geocoder,* que obtiene la ubicación declarada en la ip del dispositivo. También se le ha añadido la fecha y hora actual del dispositivo, mediante el paquete *time*.

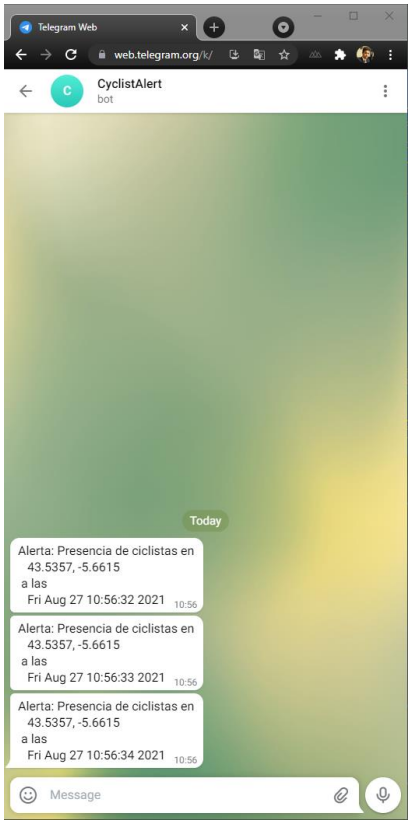

Ilustración 477: Notificaciones recibidas vía Telegram a partir de la detección de ciclistas a través del modelo obtenido mediante Transfer Learning.

Con estos datos de tiempo y ubicación se podría hacer un trazado básico de las carreteras y vías más transitadas por ciclistas en cada momento del día, generar un mapa de calor para alertar al resto de usuarios, y diversas utilidades más.

Este diseño sirve para ejemplificar como sería el envío de una alerta a un sistema compartido para los usuarios, en el que cuando un vehículo detecta un ciclista o grupo de ciclistas pone esa detección a disposición del resto de usuarios mejorando así la seguridad de todos, ciclistas y no ciclistas, en una misma vía.

6.2. Mínimo producto viable (MVP)

Por último, cabe destacar que esta prueba se ha llevado a trabajo de campo real, en el que yo mismo me he metido en el coche con el ordenador y la webcam y he ido probando el MVP, comprobando así su funcionalidad.

Como demostración de una primera versión funcional del producto desarrollado, se presenta entonces,todo lo descrito en el apartado anterior en su conjunto.

En esta primera versión se puede circular con la cámara instalada en el salpicadero del coche, conectada por USB al ordenador, y ejecutar el programa de detección de ciclistas para ir recibiendo, al mismo tiempo, las alertas en formato notificación, de forma automática, a través del bot de Telegram creado a tal efecto llamado *CyclistAlert*.

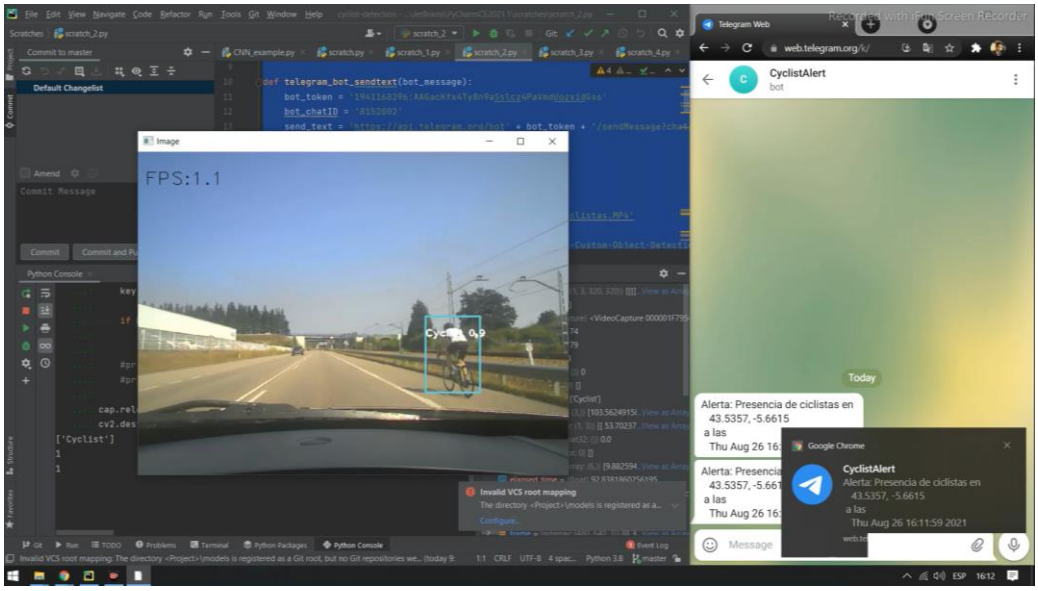

Ilustración 488: Captura de pantalla de la demo.

Con esto queda demostrada la funcionalidad y utilidad del sistema de alerta temprana de ciclistas. Esto podría incorporarse como uno más de los sistemas de seguridad de los vehículos modernos, o transformarse en una aplicación de uso en smartphone de modo que se pueda utilizar con el propio smartphone independientemente del vehículo y de sus condiciones.

Esto podría suponer un antes y un después en la relación y convivencia de los ciclistas con el resto de vehículos y usuarios de las carreteras, pudiendo así entre todos empezar a disminuir las cifras que se comentan al principio del trabajo.

7. Conclusiones y trabajo futuro

Tras la realización de este trabajo se han extraído una serie de conclusiones, así como también se han planteado diversas líneas de trabajo para seguir avanzando y evolucionando la herramienta en el corto y medio plazo.

Todas ellas se enumeran a continuación.

7.1. Conclusiones y aprendizaje

En este trabajo se ha investigado y estudiado acerca de los modelos más relevantes y populares en lo referente a Smart Mobility.

Se ha trabajado desde los primeros modelos de redes neuronales cercanos a la pura fuerza bruta, pasando por los modelos de dos fases, los cuales incluyen métodos de propuesta de regiones de interés, y llegando por último a los modelos de una fase, en los que tanto la clasificación del objeto como la ubicación del mismo dentro de la imagen se realizan de forma simultánea.

Concretamente, se han evaluado y comparado las redes neuronales convolucionales Faster R-CNN, SSD ResNet, SSD Mobilnet Lite, Yolo v3 y Yolo v3 Tiny, realizando un estudio comparativo entre ellos para un caso de uso en concreto: la detección de bicicletas y personas.

En este punto, varios de los resultados obtenidos que resultan más llamativos han sido el mal desempeño general de Yolo v3 Tiny en la detección tanto de bicicletas como personas, y el buen funcionamiento generalizado del resto en la categoría bicicleta. Tomando en cuenta todos los resultados al detalle, incluidos tiempos de ejecución donde los modelos basados en arquitectura Yolo sacan notable ventaja al resto, el ganador final de la comparativa ha sido Yolo v3.

Posteriormente, a este modelo ganador se le han aplicado satisfactoriamente técnicas de *Transfer Learning* para el reentrenamiento del mismo, obteniendo finalmente un modelo de detección de ciclistas que, bajo ciertos criterios de parada del entrenamiento, tiene una métrica final de pérdida media de 0.18.

Finalmente, se ha productivizado este modelo dando lugar a una herramienta que detecta ciclistas en tiempo real y envía automáticamente una alerta en forma de notificación en una aplicación de mensajería. Para ello se ha creado un bot en Telegram, que recibe los mensajes

vía http desde el proceso de inferencia en Python del modelo sobre vídeo en tiempo real, obtenido directamente desde una webcam instalada en el salpicadero del vehículo, donde se ha realizado una prueba de campo para validar su funcionamiento.

De todo ello se extrae, como conclusión final, que se han cumplido todos y cada uno de los objetivos específicos planteados en el apartado 3. Objetivos y metodología de trabajo, así como el objetivo global de este trabajo en cuanto a aportar una propuesta novedosa para solucionar una problemática real del campo en estudio en el presente Máster.

7.2. Líneas de trabajo futuro

Tras obtener todo lo mencionado en el epígrafe anterior, surgen de forma inmediata varias posibles mejoras, en direcciones divergentes:

- Por un lado, sería interesante trabajar en la dirección del hardware. Implementar esta herramienta en un hardware específico diseñado para el uso de la misma implicaría mejoras notables en la ejecución del producto.

- En segundo lugar, y en la línea de ampliar el uso de la herramienta en el ámbito de la ayuda a la conducción, también resultaría de interés extender y ampliar el número de clases de objetos que detecta, pasando de solamente ciclistas a detectar también coches averiados o parados en el arcén, accidentes, retenciones, etc…

- Al hilo de lo anterior, sería de gran utilidad integrar esta herramienta con una aplicación de navegación o mapas, tipo Google Maps, de forma que no sea necesario indicar en ésta una alerta de forma manual si no que, mediante la cámara del smartphone, se detecte automáticamente el objeto u obstáculo en la vía y la alerta se dispare de forma automática.

- Derivado del punto anterior, también se podría desarrollar un proceso de NLP (*Natural Language Processing*) que pida *feedback* mediante voz al usuario por cada alerta disparada automáticamente, de forma que esto sirva para realizar un continuo reentrenamiento de los modelos de detección.

- Por último, se podría integrar esta herramienta desarrollada en servicios o *frameworks* de conducción condicionada (CD), conducción asistida (ADAS) o conducción autónoma (AD).

8. Bibliografía

«Jetson Nano 2GB Developer Kit». NVIDIA Developer, 5 de octubre de 2020, https://developer.nvidia.com/embedded/jetson-nano-2gb-developer-kit.

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... & Zheng, X. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467.

Acosta, S. «Google Maps permite a los usuarios informar de accidentes y radares en tiempo real». TreceBits - Redes Sociales y Tecnología, 18 de octubre de 2019, https://www.trecebits.com/2019/10/18/google-maps-permite-a-los-usuarios-informar-de-accidentes-y-radares-en-tiempo-real/.

Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). Yolov4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934.

Brownlee, Jason. «9 Applications of Deep Learning for Computer Vision». Machine Learning Mastery, 12 de marzo de 2019, https://machinelearningmastery.com/applications-of-deep-learning-for-computer-vision/.

Dai, J., Li, Y., He, K., & Sun, J. (2016). R-fcn: Object detection via region-based fully convolutional networks. In Advances in neural information processing.

Felzenszwalb, P. F., Girshick, R. B., McAllester, D., & Ramanan, D. (2009). Object detection with discriminatively trained part-based models. IEEE transactions on pattern analysis and machine intelligence, 32(9), 1627-1645.

Fernández Alonso, J.C. (2013). Métodos de ayuda a la toma de decisiones: Curvas Roc. Universidad de Oviedo.

Gallello, I. «Madrid se cae de la bici: aumentan un 270% los accidentes en 10 años». EL PAÍS, 26 de marzo de 2021, https://elpais.com/espana/madrid/2021-03-26/madrid-se-cae-de-la-bici-aumentan-un-270-los-accidentes-en-10-anos.html.

Girshick, R. (2015). Fast r-cnn. In Proceedings of the IEEE international conference on computer vision (pp. 1440-1448).

Girshick, R., Donahue, J., Darrell, T., Malik, J., & Mercan, E. (2014). R-CNN for Object Detection. In IEEE Conference.

Guo, Y., Liu, Y., Oerlemans, A., Lao, S., Wu, S., & Lew, M. S. (2016). Deep learning for visual understanding: A review. Neurocomputing, 187, 27-48.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

He, W., Huang, Z., Wei, Z., Li, C., & Guo, B. (2019). TF-YOLO: An improved incremental network for real-time object detection. Applied Sciences, 9(16), 3225.

Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. Neural computation, 18(7), 1527-1554.

Hubel, D. H., & Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. The Journal of physiology, 160(1), 106-154.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., ... & Darrell, T. (2014, November). Caffe: Convolutional architecture for fast feature embedding. In Proceedings of the 22nd ACM international conference on Multimedia (pp. 675-678).

Khedekar, N. (2014). We now upload and share over 1.8 billion photos each day: Meeker Internet report. FirstPost Tech, 2.

Kong, T., Yao, A., Chen, Y., & Sun, F. (2016). Hypernet: Towards accurate region proposal generation and joint object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 845-853).

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25, 1097-1105.

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., & Jackel, L. (1989). Handwritten digit recognition with a back-propagation network. Advances in neural information processing systems, 2.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... & Zitnick, C. L. (2014, September). Microsoft coco: Common objects in context. In European conference on computer vision (pp. 740-755). Springer, Cham.

Lindsay, G. W. (2020). Convolutional neural networks as a model of the visual system: Past, present, and future. Journal of cognitive neuroscience, 1-15.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016, October). Ssd: Single shot multibox detector. In European conference on computer vision (pp. 21-37). Springer, Cham.

Lu, X., Kang, X., Nishide, S., & Ren, F. (2019, December). Object detection based on SSD-ResNet. In 2019 IEEE 6th International Conference on Cloud Computing and Intelligence Systems (CCIS) (pp. 89-92). IEEE.

McCulloh, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in neural nets. Bull Math. Biophys, 5, 133-137.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems (pp. 3111-3119).

Partida Tapia, M. A., Manrique Ramírez, P., & Barrón Fernández, R. (1995). Percepción Computacional. Polibits, 14, 20-23.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., ... & Lerer, A. (2017). Automatic differentiation in pytorch.

Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. Advances in neural information processing systems, 28, 91-99.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4510-4520).

Sengupta, A., Ye, Y., Wang, R., Liu, C., & Roy, K. (2019). Going deeper in spiking neural networks: VGG and residual architectures. Frontiers in neuroscience, 13, 95.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

Tanaka, F. H. K. D. S., & Aranha, C. (2019). Data augmentation using GANs. arXiv preprint arXiv:1904.09135.

Tokui, S., Oono, K., Hido, S., & Clayton, J. (2015, December). Chainer: a next-generation open source framework for deep learning. In Proceedings of workshop on machine learning systems (LearningSys) in the twenty-ninth annual conference on neural information processing systems (NIPS) (Vol. 5, pp. 1-6).

Torrey, L., & Shavlik, J. (2010). Transfer learning. In Handbook of research on machine learning applications and trends: algorithms, methods, and techniques (pp. 242-264). IGI global.

Van Dyk, D. A., & Meng, X. L. (2001). The art of data augmentation. Journal of Computational and Graphical Statistics, 10(1), 1-50.

Voulodimos, A., Doulamis, N., Doulamis, A., & Protopapadakis, E. (2018). Deep learning for computer vision: A brief review. Computational intelligence and neuroscience, 2018.

Yuheng, S., & Hao, Y. (2017). Image segmentation algorithms overview. arXiv preprint arXiv:1707.02051.

Comparativa y entrenamiento de modelos de Smart Mobility: detección y alerta temprana de ciclistas.

Juan Carlos Fernández Alonso

Universidad Internacional de la Rioja,
Logroño (España)

22 de septiembre de 2021

Resumen

Este proyecto se enfocará en el ámbito de la Movilidad Inteligente y, más concretamente, en la detección temprana de objetos en la vía. En primer lugar, se realizará una breve descripción del estado del arte en el que se revisan los modelos y técnicas de Inteligencia Artificial más utilizados. Posteriormente, se abordará un estudio comparativo en el que se elegirán cinco de los modelos de Aprendizaje Profundo de uso más extendido, que dará como ganador un modelo en concreto, bajo determinados criterios previamente expuestos. A continuación, se utilizará dicho modelo como base para la construcción de uno nuevo utilizando técnicas de Transferencia de Aprendizaje para un caso de uso concreto: detección temprana de ciclistas en la vía. A modo de conclusión del proyecto y con el fin de verificar y validar el modelo desarrollado, se construye un Mínimo Producto Viable consistente en la detección temprana de ciclistas y envío de alertas. Finalmente se realiza una demostración de su funcionalidad y se proponen líneas de trabajo futuras para su evolución y productivización.

Palabras Clave

Movilidad Inteligente, Aprendizaje Profundo, Redes Neuronales Convolucionales, Detección de objetos, Ciclista.

I. Introducción

En el presente artículo se expone, de forma resumida, el estudio e investigación realizados en lo respectivo al área de Movilidad Inteligente y los modelos de Aprendizaje Profundo más utilizados en ella.

En los sucesivos apartados se introduce brevemente al lector en cada una de las etapas que se ha seguido para realizar el estudio, desde una contextualización y situación en el estado del arte de las áreas de la Inteligencia Artificial que se ven involucradas y que se van a emplear, pasando por un estudio de los modelos más utilizados en Smart Mobility en la actualidad, hasta finalmente la elección de uno de ellos, bajo determinados criterios, y su reentrenamiento, obteniendo finalmente un producto funcional final.

En el apartado II. Estado del Arte se expone de forma esquemática el problema propuesto, se realiza un repaso de las áreas de la Inteligencia Artificial que se emplean para llevar a cabo el desarrollo de la solución a dicho problema, y se presentan al lector las técnicas y modelos empleados en la actualidad de forma más extendida en lo referente a *Smart Mobility*, o Movilidad Inteligente o Conectada.

En el apartado III. Objetivos y metodología se plantea la hoja de ruta que se seguirá para el desarrollo de la solución buscada, detallando así las tareas a realizar, orden, y tiempos. Del mismo modo, se comentan brevemente los requerimientos a cumplir, tanto a nivel de hardware como de software, de todo lo necesario para llevar a cabo dichas tareas.

En el apartado IV. Contribución se presenta cómo se ha construido el producto final, así como el proceso analítico que se ha seguido para llegar hasta él.

A continuación, en el apartado V. Evaluación y resultados se muestra el producto finalmente desarrollado y los componentes que lo integran, así como las pruebas de campo que se han realizado para validar su funcionalidad, consiguiendo una demostración como Mínimo Producto Viable.

Por último, se cierra el presente artículo con los apartados VI. Discusión y VII. Conclusiones y Trabajo Futuro, en los que se presenta un debate sobre lo aprendido tras la realización del trabajo y, finalmente, se comentan posibles líneas de trabajo a futuro para evolucionar y optimizar el producto desarrollado.

II. Estado del Arte

En este Trabajo Fin de Máster se trabajan, principalmente, dos potentes ámbitos de estudio:

- La Percepción Computacional
- El Aprendizaje Profundo

En este apartado se realiza una breve explicación de ambos, se presentan algunos de los problemas más representativos de la Visión Artificial, y se recorre brevemente la historia y se

contextualiza como éstos dieron lugar al nacimiento del *Deep Learning*, o Aprendizaje Profundo. Por último, se habla de cómo éste ha ido obteniendo reconocimiento y popularidad y se introducen los conceptos de Redes Neuronales Convolucionales, Transfer Learning y Data Augmentation.

Uno de los campos de estudio e investigación más populares de la Inteligencia Artificial (IA) es la Percepción Computacional, y más en concreto la Visión Artificial (o Visión por Computador, Computer Vision).

El ámbito de la Percepción Computacional [1] se encarga de estudiar los sistemas complejos de percepción del ser humano para poder entender cómo son y cómo funcionan y ser capaces de replicarlos, en la medida de lo posible, en una máquina. Uno de los sentidos perceptivos más tratado es la visión, de ahí sus nombres (Visión Artificial o Visión por Computador).

El estudio de la Visión Artificial tiene como objetivo procesar imágenes de una forma bioinspirada en el ser humano, para poder identificar y reconocer objetos, y llegar a entender lo que se ve en la imagen.

Hasta hace poco, la capacidad de los algoritmos de Visión Artificial era limitada. Sin embargo, los avances en Inteligencia Artificial en campos como el Aprendizaje Automático (*Machine Learning*) y el Aprendizaje Profundo (*Deep Learning*), el auge de las Redes Neuronales, el notable incremento de los datos que se generan y de los que disponemos actualmente (cada día se comparten más de mil ochocientos millones de imágenes por internet [2]), y sobre todo, la capacidad de cómputo de las máquinas a las que podemos acceder hoy en día han propiciado la evolución de modelos y algoritmos más eficientes y la aparición de máquinas más potentes.

Los problemas más habituales en los que los algoritmos de Visión Artificial [3] hacen acto de presencia suelen ser los siguientes:

- Clasificación: dado un conjunto de categorías, ¿en cuál se engloba un objeto?
- Identificación: ¿qué tipo de objeto es?
- Verificación: dada una categoría, ¿pertenece el objeto de la imagen a ella?
- Detección: dado un tipo de objeto, ¿dónde está en la imagen?
- Segmentación: ¿qué parte de la imagen se corresponde con el objeto?
- Reconocimiento: ¿qué tipos de objetos hay en la imagen y dónde están?

El Aprendizaje Profundo, o *Deep Learning* [4], es una parte del Aprendizaje Automático que estudia los métodos para aprender características generalistas y de alto nivel de un conjunto de datos dado, utilizando para ello estructuras de características jerárquicas, de ahí el calificativo 'profundo'.

Esta intención de replicar computacionalmente el comportamiento del pensamiento de un cerebro propició en 1943 la aparición del primero modelo de neurona [5]. Este fue el inicio de las redes neuronales artificiales, que hoy en día alcanzan tanta popularidad porque los estudios y avances en este ámbito han conseguido superar los de otras ramas, como el Procesamiento de Lenguaje Natural [6] o la Visión Artificial [7] propiamente dichas.

Si bien es cierto que las redes neuronales existen desde mediados del siglo XX, no comenzaron a tener la popularidad actual hasta el año 2006 cuando Hilton introdujo la *Deep Belief Network* [8], que supuso el despertar de las arquitecturas profundas y algoritmos de Aprendizaje Profundo. Además, el interés actual en el estudio de estas técnicas también tiene su origen en las propias características contextuales, en cuanto a datos y tecnología, de la época en que vivimos:

- Más datos con los que entrenar modelos
- Más capacidades de computación
- Más capacidad de procesamiento (GPUs)
- Aparición de frameworks como Tensorflow [9], PyTorch [10], Caffe [11], Chainer [7] , etc. que facilitan el trabajo con estas técnicas

Son numerosos los métodos tradicionales de la Percepción Computacional [1], que se siguen estudiando y utilizando a día de hoy. Sin embargo, la mayoría de ellos, aunque funcionan bien, necesitan de un experto que los configure de forma manual y ad-hoc para cada problema.

Es en este punto donde entra en juego el Aprendizaje Profundo, principalmente con las Redes Neuronales Convolucionales [12], que aprenden, con suficiente entrenamiento, las características más descriptivas de las imágenes [13].

Entre las arquitecturas más utilizadas en la actualidad a la hora de construir un modelo de aprendizaje profundo encontramos dos líneas de investigación: arquitecturas de dos fases, y arquitecturas de una fase.

Hasta el momento, se utilizaban metodologías similares a la fuerza bruta para detectar en una imagen un objeto en concreto. Esto significa que se desarrollaban soluciones basadas en una ventana que se deslizaba a lo largo de la imagen buscando el objeto deseado. Esto producía modelos muy poco eficientes, ya que si se quiere detectar objetos distintos se tienen que usar ventanas distintas, cada una acorde al tamaño y forma del objeto en cuestión, además del coste tanto en términos temporales como computacionales de evaluar todas las ventanas posibles a lo largo de una imagen.

La arquitectura de dos fases, también conocida como arquitectura basada en regiones, vio la luz en el año 2012 en la competición propuesta por ImageNet, *Large Scale Visual Recognition Challenge* (ILSVRC) de la mano del que fue su ganador: AlexNet [14].

Esta arquitectura surge como primera propuesta intuitiva para mejorar la fuerza bruta: disminuir el número de ventanas que evalúa el modelo. Partiendo de esta idea, se propone un método cuya misión principal es realizar una propuesta fundamentada de ventanas deslizantes cuya evaluación resulte de interés, diferenciándolas con claridad de las ventanas de las que no obtendríamos un resultado positivo en la evaluación del modelo de clasificación. De esta manera, surgen los modelos de propuestas de regiones de interés [15], o ROI, de una imagen.

En este trabajo se evaluará un modelo con esta arquitectura: Faster R-CNN [16].

Por otro lado, están las arquitecturas de una fase, que son algoritmos más simples que los que utilizan métodos de propuesta de regiones o redes de propuestas de regiones, porque elimina todas las fases de propuestas de regiones innecesarias hasta que encuentra la región que se adapta al objeto, a la vez que elimina también las etapas de remuestreo de píxeles o características posteriores, de forma que aglutina todos los cálculos a realizar en una sola red. Esto la convierte en una red más rápida tanto en la fase de entrenamiento como en la de inferencia.

En este trabajo se evaluarán dos tipos de redes de una fase: por un lado las SSD (Single Shot Detector), de las cuales se trabajará

con SSD Resnet y SSD Mobile lite, y las YOLO, de las cuales se evaluarán Yolo v3 y Yolo v3 Tiny.

III. Objetivos y Metodología

El objetivo final de este trabajo es desarrollar un modelo de detección de ciclistas en tiempo real.

Este modelo estará basado en una red neuronal convolucional ya existente, que se seleccionará para este caso de uso de entre las más utilizadas en el ámbito de la movilidad inteligente, mediante una comparación y evaluación de las mismas.

Posteriormente, a partir de ese modelo ganador, se utilizarán técnicas de Transfer *Learning* para ajustar su precisión a este caso de uso, así como otras técnicas de recopilación de datos y *Data Agumentation* para confeccionar un conjunto de datos de entrenamiento adecuado para tal finalidad.

Más en detalle, se plantean las siguientes tareas para llevar a cabo el presente estudio:

- Buscar y comprender la información necesaria sobre los modelos de detección de objetos más relevantes en la actualidad.
- Diseñar una plataforma de prueba y comparación de modelos.
- Importar y utilizar modelos ya entrenados por terceros.
- Evaluar modelos, compararlos y elegir el mejor para un caso de uso.
- Diseñar un conjunto de datos de entrenamiento y validación, aplicando para ello técnicas de Data Augmentation donde sea necesario.
- Re-entrenar un modelo ya existente aplicando técnicas de Transfer Learning
- Diseñar una herramienta básica de prueba para implementar un envío de alertas a partir del modelo de detección.
- Planificar las tareas necesarias para la consecución de estos objetivos, teniendo en cuenta orden, prioridad, y tiempo.
- Comunicar todas las tareas realizadas para la consecución de estos objetivos de forma clara, precisa, y entendible.

Para ello, se utilizará una unidad hardware de Jetson Nano, una GPU que permite al usuario el desarrollo de nuevos sistemas de IA (Inteligencia Artificial) de pequeño tamaño, económicos y con un bajo consumo de energía. De esta forma, se abre un abanico de posibilidades de entrenamiento y generación de modelos de *Deep learning* de manera fácil, económica y reduciendo tiempos de entrenamiento con respecto a un ordenador convencional, cosa que ayudará bastante en el transcurso fluido del proyecto.

En cuanto a requerimientos software, se utilizará una instalación convencional de Python adecuada al hardware, y diversos repositorios de modelos de *Deep Learning* para la comparación, así como finalmente para el *Transfer Learning*.

De Tensorflow Hub se han utilizado los modelos preentrenados SSD Mobilnet Lite, SSD Resnet101, y Faster R-CNN.

De Ultralytics/yolov3 se han utilizado los modelos Yolo v3 y Yolo v3 Tiny.

Finalmente, para demostrar la funcionalidad del Mínimo Producto Viable, se integrará con un bot de Telegram que recibirá las notificaciones en tiempo real del detector de ciclistas.

IV. Contribución

El problema en cuestión consiste en aplicar *Transfer Learning* para detectar ciclistas partiendo de un modelo que sea capaz de reconocer en una imagen multitud de categorías, entre ellas bicicletas y personas.

Es de interés, por lo tanto, ver cómo se comporta cada modelo a la hora de detectar las categorías 'bicicleta' y 'persona', pues cuanto mejor sea esa detección y lo que el modelo sabe sobre el reconocimiento de esas categorías, más conocimiento se podrá transferir a la solución final.

Por la propia naturaleza del objetivo, nos interesa también un modelo capaz de realizar una inferencia rápida, cumpliendo requisitos de *near real time*.

Por lo tanto, en resumen, nuestros criterios para decidir con qué modelo vamos a continuar serán los siguientes:

- Efectividad detectando categoría 'bicicleta'.
- Efectividad detectando categoría 'persona'.
- Tiempo esperado de ejecución de inferencia sobre una imagen.

Para realizar la comparación de los citados modelos de detección de objetos, se ha confeccionado un conjunto de imágenes constituido por 1.726 imágenes, en las cuales podemos observar las siguientes casuísticas:

- 675 imágenes de bicicletas. En cada una de estas imágenes podemos ver una bicicleta sobre fondo natural o artificial, en perspectivas variadas.
- 502 imágenes de personas. En cada imagen sale al menos una persona de forma clara, en perspectivas variadas
- 549 imágenes de otras cosas (cualquier cosa excluida de los tres conjuntos anteriores, por ejemplo, un pez, un árbol, un coche, una catedral…)

El objetivo con estas imágenes es realizar sobre ellas la inferencia de los modelos a comparar, teniendo dos categorías de interés inicial (bicicleta y persona), y una categoría para ver si los modelos dan falsas alarmas (otros).

También se han recopilado imágenes de una última categoría, la que se utilizará para la aplicación del Transfer Learning en el re-entrenamiento del modelo que resulte ganador tras la comparación. Este conjunto de datos consta de:

- 413 imágenes de ciclistas. En cada imagen sale una persona montando en bici, en perspectivas variadas, donde el fondo de la imagen es el habitual para un ciclista: carreteras o senderos.

Tras evaluar los cinco modelos sobre estos conjuntos de imágenes, obtenemos los resultados que se comentan a continuación.

En cuanto a la detección de personas, con los datos obtenidos podemos concluir:

PERSONAS	PRECISION	RECALL	ACCURACY	ROC AUC	PR AUC
Yolo v3	0,80	0,99	0,93	0,97	0,92
Yolo v3 Tiny	0,83	0,60	0,85	0,88	0,82
SSD ResNet101	0,80	0,99	0,93	0,98	0,93
SSD Mobilnet Lite	0,84	0,99	0,94	0,98	0,94
Faster R-CNN	0,75	1,00	0,90	0,98	0,92

Tabla 1: Resumen de las métricas obtenidas en la evaluación de los modelos para la detección de personas.

Podemos ver que el modelo SSD Mobilnet Lite tiene, en general, los valores mayores tanto en las métricas como en las curvas, y le sigue muy de cerca el modelo Yolo v3, con unas métricas muy similares (valoradas en conjunto).

Sobre el resto de modelos, podemos destacar que Faster R-CNN ha sido excelente a la hora de reconocer los positivos reales (es decir, personas) ya que las ha categorizado todas como tal (recall = 1) mientras que por otro lado, 1 de cada 4 positivos que ha señalado ha sido falso (no hay persona), es decir, un negativo (precisión = 0.75). Esto traducido a un caso de uso real de notificación al detectar personas, significaría que por cada 4 notificaciones que se realicen, 1 es errónea.

Por último, cabe destacar la mala métrica de Yolo v3 Tiny en cuanto a recall, con valor 0.6. Esto vendría a significar que de todos los positivos, es decir, de todas las personas, detecto poco más de la mitad.

En cuanto a la detección de bicicletas, con los datos obtenidos podemos concluir:

BICICLETAS	PRECISION	RECALL	ACCURACY	ROC AUC	PR AUC
Yolo v3	0,99	0,99	0,99	0,99	0,99
Yolo v3 Tiny	0,99	0,16	0,67	0,80	0,87
SSD ResNet101	0,99	0,93	0,97	0,99	0,99
SSD Mobilnet Lite	0,99	0,95	0,98	0,99	0,99
Faster R-CNN	0,99	0,99	0,99	0,99	0,99

Tabla 2: Resumen de las métricas obtenidas en la evaluación de los modelos para la detección de bicicletas.

Cabe destacar el buen desempeño en general de todos los modelos, excepto Yolo v3 Tiny, en cuanto a la detección de bicicletas, algo bastante llamativo en comparación a la detección de personas.

En este caso, los mejores resultados tanto en métricas como en las gráficas los obtienen, a la par, Faster R-CNN y Yolo v3, ambos con casi un acierto del 100% en todas las métricas.

Las métricas aquí de Yolo v3 Tiny se desploman, cayendo hasta el 0.16 el recall, lo que significaría que solo detecta una bicicleta real de cada 6 imágenes de bicicletas.

En cuanto a los tiempos de inferencia:

Modelo	Personas	Bicicletas
Yolo v3	0,04	0,04
Yolo v3 Tiny	0,01	0,01
SSD ResNet101	9,84	9,92
SSD Mobilnet Lite	0,6	0,58
Faster R-CNN	7,47	7,78

Tabla 3: Resumen de los tiempos de inferencia de los modelos.

Sin duda alguna, para nuestro caso de uso podemos descartar por este criterio los modelos Faster R-CNN y SSD ResNet101, que tardan del orden de los 8 segundos y 10 segundos para cada inferencia, respectivamente.

Los modelos cuyos tiempos de inferencia son mejores son los basados en arquitectura YOLO, cuyos tiempos de ejecución son del orden de centésima de segundo, seguidos de cerca (en comparación con el resto) por SSD Mobilnet Lite, cuyos tiempos de ejecución son del orden de décima de segundo.

Como conclusión final, y partiendo de toda la información que nos otorgan estos resultados, descartamos directamente Faster R-CNN y SSD ResNet101 por los altos tiempos de inferencia, y Yolo v3 Tiny por sus malos resultados obtenidos en la evaluación de las imágenes de ambas clases.

Por último, la decisión entre SSD MobilNet Lite y Yolo v3 la tomaremos a favor de este último, ya que, aunque ambos tienen excelentes resultados en la detección de ambas clases, el tiempo de inferencia de Yolo v3 es 14 veces menor que el de SSD MobilNet Lite. Esto supone una diferencia de orden de magnitud suficientemente elevado como para asumir que la diferencia de rendimiento es transferible a un dispositivo de test en el que se evaluará el futuro modelo de detección de ciclistas, al mismo tiempo que garantiza que las pruebas en un dispositivo con hardware no diseñado específicamente para este problema puedan llevarse a cabo dentro de la definición de real.

V. Evaluación y Resultados

En esta sección se presentan los resultados finales del trabajo desarrollado a partir de la comparación de modelos descrita previamente, y el modelo obtenido tras el reentrenamiento de la red neuronal ganadora de dicha comparación.

Del mismo modo, se presenta finalmente la construcción del MVP y su validación en una prueba de campo real, detectando ciclistas en una carretera.

1. Transfer Learning

Para realizar la transferencia de conocimiento de Yolo v3 original y crear un nuevo modelo capaz de detectar ciclistas, revisamos la documentación del repositorio que vamos a utilizar y las indicaciones y recomendaciones del autor (Redmon y Farhadi, 2018), esto nos ayudará a configurar la parametrización del entrenamiento.

En esta ocasión, para realizar el proceso de Transfer Learning se fijan las primeras 75 capas, es decir, conservamos el conocimiento que la red ya tiene en esas capas, y dejamos libres para el reentrenamiento las 32 restantes, divididas en tres bloques predictores.

Estas 32 capas restantes que sí se van a entrenar, las inicializamos también con los pesos de la red Yolo v3 original. Esta recomendación del autor es adecuada en nuestro caso ya que el problema de detección de ciclistas está estrechamente relacionado con la detección de personas y bicicletas que ya realiza la red, con lo que inicializando estas capas con unos pesos que ya tienen sentido e información funcional, teóricamente, llegaremos a un mejor modelo y en menos tiempo que si realizamos una inicialización aleatoria de los mismos.

Tras la realización de numerosas pruebas con distintos valores, los criterios de entrenamiento y de parada del modelo obtenido finalmente son los siguientes:

- Epochs = 2000
- Learning rate = 0.001
- Momentum = 0.9
- Decay = 0.0005
- IoU threshold = 0.75

Y finalmente, tras las 2000 epochs de entrenamiento bajo esos criterios, se construye un modelo con una pérdida media (avg loss) de 0.185087.

Con esto, se genera los ficheros de configuración .cfg y de pesos .weights del modelo, necesarios para cargarlo y utilizarlo

en cualquier proceso que lo requiera. En nuestro caso, bastará con disponer de una instalación Python compatible y la inclusión de la librería de visión artificial Opencv.

2. Mínimo Producto Viable

Cabe recordar que el objetivo de este estudio es llegar a crear un producto que pueda ser utilizado en la vida real para la detección temprana de ciclistas en la vía de modo que esto suponga una mejora en la seguridad tanto de los propios ciclistas como una ayuda en atención y visibilizacion para el resto de usuarios de dicha vía.

Para este MVP, se han desarrollado varios procesos independientes que, finalmente, se integraron para dar lugar a la herramienta final:

1. Un proceso en Python que carga el modelo y ejecuta la inferencia sobre una imagen, detectando ciclistas donde proceda.
2. Un proceso que realiza la tarea en 1. para cada imagen (frame) de un vídeo.
3. Un proceso que generaliza 2. para un vídeo tomado en tiempo real desde una webcam.
4. Un disparador de alertas basado en el envío de mensajes http a una herramienta de mensajería instantánea.

Primero se ha desarrollado una versión que realiza la inferencia del modelo de detección de ciclistas sobre una imagen de forma offline, obteniendo resultados satisfactorios sobre imágenes de ciclistas del dataset reservado para test (es decir, imágenes con las que no se ha entrenado el modelo):

Ilustración 1: Ejemplos de inferencia del modelo obtenido mediante Transfer Learning.

A continuación, y siguiendo en la dirección marcada por el MVP, se ha desarrollado un programa que realiza la misma inferencia pero iterada sobre un conjunto de imágenes finito, de forma offline también, obtenidas sobre un vídeo.

Por último, se generaliza este proceso añadiendo una función que lee vídeo en tiempo real desde una webcam, transformando automáticamente el vídeo en imágenes (para esta prueba y debido a limitaciones de hardware se convierte el vídeo a 3 *frames* por segundo) y se ejecuta la inferencia del modelo por cada uno de esos *frames*.

Para ejemplificar el envío de alertas a una aplicación de uso compartido cuando se detecta un ciclista se ha utilizado como herramienta de mensajería instantánea llamada Telegram, junto con un paquete de Python llamado *requests*, que permite enviar mensajes por http.

Para ello, se ha creado un bot en Telegram para el uso específico y único de esta prueba, llamado CyclistAlert, y se ha conectado con el script de detección de ciclistas mediante las claves http que proporciona el propio Telegram.

Los mensajes de alerta se envían desde el propio módulo de detección de ciclistas implementado, justo después de cada inferencia en la que se ha detectado algún ciclista en la imagen de video a través de la cámara.

A este mensaje de alerta se le ha añadido la ubicación geoespacial de la detección mediante un paquete de Python que se llama *geocoder*, que obtiene la ubicación declarada en la ip del dispositivo. También se le ha añadido la fecha y hora actual del dispositivo, mediante el paquete time.

Con estos datos de tiempo y ubicación se podría hacer un trazado básico de las carreteras y vías más transitadas por ciclistas en cada momento del día, generar un mapa de calor para alertar al resto de usuarios, y diversas utilidades más.

Este diseño sirve para ejemplificar como sería el envío de una alerta a un sistema compartido para los usuarios, en el que cuando un vehículo detecta un ciclista o grupo de ciclistas pone esa detección a disposición del resto de usuarios mejorando así la seguridad de todos, ciclistas y no ciclistas, en una misma vía.

Tras la realización de este MVP, cabe destacar que esta prueba se ha llevado a trabajo de campo real, en el que yo mismo me he metido en el coche con el ordenador y la webcam y he ido probando el producto, comprobando así su funcionalidad.

Como demostración de una primera versión funcional del producto desarrollado, se presenta entonces todo lo descrito anteriormente en su conjunto.

En esta primera versión se puede circular con la cámara instalada en el salpicadero del coche, conectada por USB al ordenador, y ejecutar el programa de detección de ciclistas para ir recibiendo, al mismo tiempo, las alertas en formato notificación, de forma automática, a través del bot de Telegram creado a tal efecto llamado CyclistAlert.

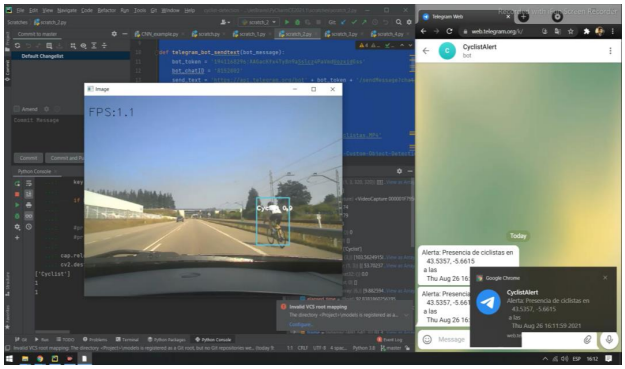

Ilustración 2: Ejemplos de inferencia del modelo obtenido mediante Transfer Learning.

VI. Discusión

Hay que reseñar que las limitaciones de hardware y el hecho de no utilizar dispositivos específicamente diseñados para tal finalidad puede no haber producido unos resultados óptimos, aunque si se ha conseguido probar la funcionalidad al completo de la herramienta desarrollada.

Es por esto que cabe destacar que las mejoras serían notablemente sustanciales al ejecutar la herramienta con hardware y dispositivos escogidos para este fin.

También es preciso señalar que, tras el análisis comparativo, se eligió como ganador a Yolo v3 frente a SSD Mobilnet Lite, los dos modelos más potentes en este ámbito de estudio. A pesar de que ambos modelos tenían métricas excelentes en la detección

tanto de bicicletas como de personas, Yolo v3 era unas 14 veces más rápido. Esto podría dar lugar a una comparativa de ambos modelos mucho más exhaustiva. Yolo v3 tenía un tiempo de inferencia medio de 0.04, frente al 0.6 de SSD Mobilnet Lite. Con hardware específico y orientado a tal objetivo cabe esperar que esa diferencia mantenga el ratio en favor de Yolo, pero se reduzcan los tiempos de ejecución en un orden de magnitud suficiente como para que SSD Mobilnet Lite pueda evaluar varios *frames* por segundo y así, obtener otros modelos y resultados.

VII. Conclusiones y Trabajo Futuro

En este trabajo se ha investigado y estudiado acerca de los modelos más relevantes y populares en lo referente a Smart Mobility.

Concretamente, se han evaluado y comparado las redes neuronales convolucionales Faster R-CNN, SSD ResNet, SSD Mobilnet Lite, Yolo v3 y Yolo v3 Tiny, realizando una estudio comparativo entre ellos para un caso de uso en concreto: la detección de bicicletas y personas.

Posteriormente, se han aplicado técnicas de *Transfer Learning* para el reentrenamiento del modelo ganador de la comparativa, Yolo v3, para obtener un modelo de detección de ciclistas.

Finalmente, se ha productivizado dicho modelo dando lugar a una herramienta que detecta ciclistas en tiempo real y envía una alerta en forma de notificación vía Telegram.

Tras ello, surgen de forma inmediata varias posibles mejoras, en direcciones divergentes:

Por un lado, sería interesante trabajar en la dirección del hardware. Implementar esta herramienta en un hardware específico diseñado para el uso de la misma implicaría mejoras notables en la ejecución del producto.

En segundo lugar, y en la línea de ampliar el uso de la herramienta en el ámbito de la ayuda a la conducción, también resultaría de interés extender y ampliar el número de clases de objetos que detecta, pasando de solamente ciclistas a detectar también coches averiados o parados en el arcén, accidentes, retenciones, etc…

Al hilo de lo anterior, sería de gran utilidad integrar esta herramienta con una aplicación de navegación o mapas, tipo Google Maps, de forma que no sea necesario indicar una alerta de forma manual si no que, mediante la cámara del smartphone, se detecte automáticamente el objeto u obstáculo en la vía y la alerta se dispare de forma automática.

Derivado del punto anterior, también se podría desarrollar un proceso de NLP que pida *feedback* mediante voz al usuario por cada alerta disparada automáticamente, de forma que esto sirva para realizar un continuo re-entrenamiento de los modelos de detección.

Por último, se podría integrar esta herramienta desarrollada en servicios o *frameworks* de conducción condicionada (CD), conducción asistida (ADAS) o conducción autónoma (AD).

Referencias

[1] Partida Tapia, M. A., Manrique Ramírez, P., & Barrón Fernández, R. (1995). Percepción Computacional. Polibits, 14, 20-23.

[2] Khedekar, N. (2014). We now upload and share over 1.8 billion photos each day: Meeker Internet report. FirstPost Tech, 2.

[3] Brownlee, Jason. «9 Applications of Deep Learning for Computer Vision». Machine Learning Mastery, 12 de marzo de 2019, https://machinelearningmastery.com/applications-of-deep-learning-for-computer-vision/.

[4] Guo, Y., Liu, Y., Oerlemans, A., Lao, S., Wu, S., & Lew, M. S. (2016). Deep learning for visual understanding: A review. Neurocomputing, 187, 27-48.

[5] McCulloh, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in neural nets. Bull Math. Biophys, 5, 133-137.

[6] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems (pp. 3111-3119).

[7] Tokui, S., Oono, K., Hido, S., & Clayton, J. (2015, December). Chainer: a next-generation open source framework for deep learning. In Proceedings of workshop on machine learning systems (LearningSys) in the twenty-ninth annual conference on neural information processing systems (NIPS) (Vol. 5, pp. 1-6).

[8] Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. Neural computation, 18(7), 1527-1554.

[9] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... & Zheng, X. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467.

[10] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., ... & Lerer, A. (2017). Automatic differentiation in pytorch.

[11] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., ... & Darrell, T. (2014, November). Caffe: Convolutional architecture for fast feature embedding. In Proceedings of the 22nd ACM international conference on Multimedia (pp. 675-678).

[12] Lindsay, G. W. (2020). Convolutional neural networks as a model of the visual system: Past, present, and future. Journal of cognitive neuroscience, 1-15.

[13] Voulodimos, A., Doulamis, N., Doulamis, A., & Protopapadakis, E. (2018). Deep learning for computer vision: A brief review. Computational intelligence and neuroscience, 2018.

[14] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25, 1097-1105.

[15] Kong, T., Yao, A., Chen, Y., & Sun, F. (2016). Hypernet: Towards accurate region proposal generation and joint object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 845-853).

[16] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. Advances in neural information processing systems, 28, 91-99.