Sistema de detección automática de baches en el asfalto a partir de imágenes

Diego Castro Viadero

Septiembre 2019

Tutor: Ricardo Moya García

Resumen

El estado del asfalto en carreteras tanto de ámbito nacional como de ámbito urbano es de alta importancia en relación a la seguridad vial. En la actualidad, gracias a los avances tecnológicos, se están desarrollando sistemas de detección automática de baches en el asfalto, permitiendo una detección precoz de estas irregularidades en la carretera.

Este proyecto pretende contribuir en este ámbito desarrollando un sistema de detección automática de baches a partir de imágenes. Partiendo de un conjunto de imágenes etiquetadas con baches, se ha entrenado una red neuronal YOLO v3 y otra YOLO v3 Tiny. El entrenamiento se ha realizado con distintos tamaños de red y con distintos subconjuntos de imágenes, dando lugar a quince modelos. Tras un estudio comparativo de las precisiones de los modelos, aquellos con mejores resultados han sido exportados y transformados para ser ejecutables en un dispositivo móvil. Finalmente se ha desarrollado una aplicación móvil Android que carga los modelos anteriormente exportados y los ejecuta con imágenes obtenidas de la cámara del dispositivo.

Abstract

The state of asphalt on both national and urban roads is of high importance in relation to road safety. Currently, thanks to technological advances, automatic detection systems are being developed to detect potholes in the asphalt, allowing early detection of these irregularities on the road.

This project aims to contribute to this field by developing an automatic pothole detection system based on images. Starting from a set of images labeled with potholes, a neural network YOLO v3 and another YOLO v3 Tiny have been trained. The training has been carried out with different network sizes and with different subsets of images, resulting in fifteen models. After a comparative study of the precisions of the models, those with better results have been exported and transformed to be executable in a mobile device. Finally, an Android mobile application has been developed that loads the previously exported models and executes them with images obtained from the device's camera.

Agradecimientos

A mi tutor Ricardo Moya García, por su esfuerzo, dedicación, consejo y por guiarme y darme ánimos en todo momento.

A mi pareja Nerea, por toda la paciencia que ha tenido y por el apoyo que me ha brindado a lo largo de esta aventura.

!!! TODO

Contenido

1. Introducción

1.1. Motivación

En los últimos años las prioridades en relación a la seguridad vial han provocado un cambio de mentalidad en la sociedad. Muchos de los últimos avances tecnológicos están orientados al desarrollo de medios de transporte más seguros. Uno de estos avances que contribuye a la mejora de la seguridad es la inclusión de un sistema de detección de desperfectos en la calzada.

En la actualidad, en el ayuntamiento de Madrid, existe un sistema de sugerencias y reclamaciones [9] que permite al ciudadano, entre otras opciones, denunciar la existencia de irregularidades en el asfalto, tener un registro de las mismas y planificar su subsanación. Según el informe de sugerencias y reclamaciones del ayuntamiento de Madrid [8], en el primer semestre del año, se registraron 3.000 reclamaciones en materia de *vías y espacios públicos* de las cuales el 50 % correspondieron a la submateria *aceras y calzadas*.

Este sistema de funcionamiento actual, que delega en el ciudadano la tarea de reporte de este tipo de desperfectos, dificulta y retrasa la puesta en conocimiento de las irregularidades a las autoridades responsables aumentando la probabilidad de que suceda algún incidente.

Algunas marcas de vehículos han desarrollado innovaciones, que son capaces de detectar baches cuando pasan sobre ellos y adaptar la dureza de la suspensión para conseguir una conducción más segura y cómoda. También contemplan un envío sistemático de la detección de baches, en tiempo real, tanto a otros vehículos como a las autoridades pertinentes. Otras marcas plantean la inclusión de cámaras que permitan la detección del bache sin necesidad de pasar por encima de este.

Con estas innovaciones se recupera el control sobre la detección de baches, liberando al ciudadano de esta tarea y reduciendo la probabilidad de incidentes gracias a disponer de la información con más antelación.

1.2. Objetivos

Este trabajo trata de desarrollar un sistema de detección de baches en tiempo real en línea con los últimos avances tecnológicos. El objetivo es entrenar una red neuronal con un conjunto de imágenes donde los baches han sido etiquetados previamente y obtener como resultado un modelo exportable para ser ejecutado en un dispositivo móvil.

Gracias a este trabajo he podido ampliar los conocimientos adquiridos durante el máster de Big Data & Data Science, impartido por la U-TAD, centrando el proyecto en el procesamiento de imágenes y tratando al mimso tiempo que tenga una utilidad social.

1.3. Estructura del trabajo

La sección *Estado del arte* explica cómo se pueden solucionar este tipo de problemas, y se centra en la resolución de los mismos mediante el uso de redes neuronales, valorando los pros y contras de cada uno de ellas.

La sección *Definición de requisitos y análisis* realiza un análisis de requisitos del proyecto y plantea una arquitectura para su desarrollo.

La sección *Datos* describe el conjunto de datos utilizado, el análisis exploratorio que se ha realizado sobre el mismo y el preprocesamiento que se realiza sobre los datos antes de ser utilizados.

La sección *Técnicas de Deep Learning y métodos de evaluación* realiza una descripción a nivel teórico de las principales técnicas y conceptos que se aplican en el tipo de red neuronal seleccionada para el desarrollo del proyecto. También describe las métricas que se utilizan para evaluar los modelos generados.

La sección *Implementación y evaluación de las técnicas* describe cómo se ha implementado la red neuronal y el resultado de la evaluación de los modelos generados.

La sección *Resultados* muestra algunos ejemplos concretos obtenidos con los distintos modelos generados.

La sección *Conclusiones* contiene una valoración del proyecto y se comentan posibles mejoras a realizar.

2. Estado del arte

El problema que se pretende resolver podría ser afrontado bien como un problema de clasificación de imágenes o bien como un problema de detección de objetos.

El primero de los enfoques es más sencillo y está más estudiado. Dada una imagen, se determina una clase a la que pertenece la imagen. En los problemas de clasificación cada una de las imágenes se centran en un único objeto. Este tipo de problemas se resuelven comúnmente con redes neuronales convolucionales. Existen numerosas arquitecturas de redes neuronales convolucionales ya definidas y estudiadas para resolver este tipo de problemas, como por ejemplo: VGG-16, LeNet, ResNet, GoogLeNet/Inception, etc.

El segundo de los enfoques es más complicado, y presenta varios retos. El primero de ellos es que las imágenes no se centran en un único objeto, sino que puede haber múltiples objetos a detectar y además tratarse de objetos de distintos tipos. El segundo de los restos es el tamaño de los objetos a identificar, que puede ser variable. Y el tercero de los retos es que se están resolviendo dos problemas al mismo tiempo: localizar objetos en una imagen y clasificar los objetos localizados.

Para resolver los problemas de detección de objetos existen dos aproximaciones. La primera de las aproximaciones es una aproximación clásica, basada en técnicas de machine learning. Un ejemplo representativo de esta aproximación clásica es Viola-Jones, que se basa en clasificadores binarios y que se ha usado en las cámaras de fotos para la detección de caras.

El uso del deep learning para la detección de objetos ha supuesto una revolución y ha cambiado las reglas del juego. Esta aproximación para la resolución de este tipo de problemas es relativamente reciente y ha estado en constante evolución.

En estos últimos años han habido múltiples aproximaciones para afrontar el problema de detección de objetos. A continuación se va a hacer un repaso de las más relevantes [13] [15] [11] [14] [12].

R-CNN

Una de las primeras aproximaciones que hicieron uso de técnicas de deep learning para la resolución de problemas de detección de objetos es *R-CNN* (Region-based Convolutional Neural Networks) [2]. El funcionamiento de R-CNN consiste en 3 pasos:

1. Se escanea la imagen en busca de posibles objetos. Mediante un algoritmo de proposición de regiones, el más habitual es la *búsqueda selectiva* (selective search) [1], se obtienen una serie de regiones (aproximadamente unas 2000 regiones) candidatas de contener un objeto (RoI, Region of Interest)

2. Se ejecuta una red neuronal convolucional para extraer las características (features) de cada una de las regiones de interés
3. Las características obtenidas de la red neuronal convolucional alimentan:
   a) un SVM para clasificar el objeto
   b) un regresor lineal para ajustar la región candidata al objeto

Aunque con esta aproximación se obtienen buenos resultados, tiene muchos inconvenientes, el principal es la dificultad para entrenar. Por un lado hay que obtener las regiones de interés del conjunto de entrenamiento, después hay que ejecutar la red neuronal convolucional sobre cada una de las regiones candidatas para obtener las características. Finalmente hay que entrenar el clasificador SVM.

Fast R-CNN

La aproximación *R-CNN* no tardó en evolucionar hacia una aproximación más pura de deep learning. El mismo autor de *R-CNN* fue el autor de su evolución *Fast R-CNN* [5].

En este caso se ejecuta la red neuronal convolucional sobe la imagen completa para obtener las características. Una vez obtenidas las características se aplica la *búsqueda selectiva* para obtener las regiones de interés. A continuación se reduce el número de regiones de interés con una capa *RoI Pooling* y una red neuronal totalmente conectada. Por último para realizar la clasificación se utiliza un clasificador *softmax*, en lugar de SVM. Para ajustar las regiones se sigue utilizando una regresión lineal.

Esta aproximación presenta múltiples ventajas con respecto a su predecesora. Por un lado, la red neuronal convolucional se ejecuta una única vez, en lugar de una vez por cada región de interés (aproximadamente unas 2000 regiones de interés). Por otro lado, en vez de utilizar múltiples SVM para realizar la clasificación, se utiliza un único clasificador *softmax*. Con todo esto se mejora mucho el rendimiento y se facilita el entrenamiento. Sin embargo, sigue teniendo una pega, que es el uso de la *búsqueda selectiva* para obtener las regiones de interés.

!!! TODO

- Faster R-CNN
- SDD
- YOLO (YOLO, YOLOv2, YOLOv3)
- Mask R-CNN

3. Definición de requisitos y análisis

3.1. Definición de requisitos

A continuación se enumeran los requisitos para el desarrollo del trabajo:

- Implementar y entrenar una red neuronal que genere un modelo capaz de detectar baches en el asfalto a partir de imágenes
- El modelo deberá ser capaz de procesar una imagen en un tiempo lo más cercano posible a 1/30s, para ser capaz de procesar video en tiempo real
- El modelo deberá poderse ejecutar en un dispositivo móvil Android
- El modelo será alimentado directamente con la salida de la cámara del dispositivo móvil

3.2. Arquitectura

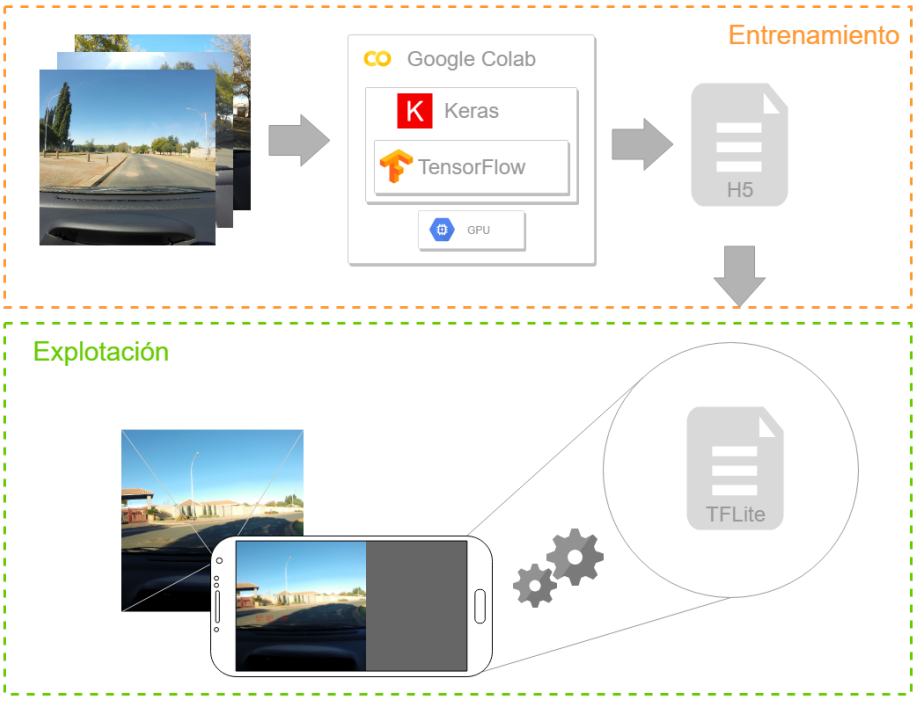


Figura 1: Arquitectura de la solución

Como se puede observar en la figura 1, la arquitectura está dividida en dos partes. Una parte se utiliza para entrenar la red neuronal y generar el modelo. La otra parte se utiliza para explotar el modelo generado.

La parte utilizada durante el entrenamiento requiere de un hardware potente y del uso de GPU para reducir los tiempos de entrenamiento. La explotación del modelo, al contrario de lo que sucede con el entrenamiento, no requiere de un hardware potente y se ejecuta directamente en un dispositivo móvil en posesión del usuario final. El modelo obtenido después de la fase de entrenamiento, se transforma a un formato que está pensado para ser ejecutado en dispositivos con recursos limitados.

3.3. Tecnologías

Todo el proyecto ha sido desarrollado utilizando Python, salvo la aplicación Android, que ha sido desarrollada en Java.

Para el procesamiento de imágenes se ha utilizado el paquete Python OpenCV. Este procesamiento incluye la lectura de imágenes en formato jpg, recorte, reescalado y volteo de las imágenes, visualización de las predicciones obtenidas, etc.

Para la implementación de la red neuronal se ha utilizado Keras, que es un envoltorio sobre Tensorflow que simplifica su uso. Dada una arquitectura de red neuronal, con Keras es muy sencillo definir las capas y sus interconexiones. Además Keras proporciona clases que facilitan la definición de un conjunto de imágenes como entrada de la red neuronal. Keras también proporciona una serie de eventos con la idea de poder suscribirse a los mismos y poder reaccionar en consecuencia, como por ejemplo, suscribirse al evento de final de época y salvar el modelo si ha mejorado con respecto a la época anterior.

Las redes neuronales utilizadas en el proyecto no se han implementado de cero, sino que se ha partido de dos implementaciones en Keras de *YOLO v3* [4] y *YOLO v3 Tiny* [6]. Se han unido ambas implementaciones en una única [3] que soporta ambos tipos de red YOLO. También se han realizado múltiples desarrollos para mejorar la funcionalidad, como por ejemplo: soporte en formato txt de las etiquetas de las imágenes, nuevos parámetros de configuración para mejorar el resultado del entrenamiento, etc.

Para la ejecución de los modelos obtenidos en un dispositivo móvil de ha utilizado TFLite. Este paquete permite transformar distintos tipos de modelo (keras, tensorflow) a formato TFLite y ejecutar estos modelos en dispositivos con recursos reducidos. En las últimas versiones de esta librería se soporta, aunque de forma experimental, el uso de la GPU del dispositivo.

La aplicación móvil ha sido desarrollada en java para la plataforma Android. TFLite dispone de una librería java que simplifica la carga del modelo y su ejecución.

4. Datos

4.1. Descripción de las fuentes de datos a utilizar

El juego de datos ha sido obtenido de kaggle [10] y se compone de un total de 1900 imágenes, tomadas desde el interior de un coche, con un tamaño igual a 3680x2760 píxeles (formato 4:3), y de un conjunto de ficheros de texto con el etiquetado de las mismas. Las imágenes se dividen en dos subconjuntos: uno de 1297 imágenes para el entrenamiento y otro de 603 imágenes para la evaluación del modelo. Por cada uno de los subconjuntos de imágenes existe un fichero de texto con el etiquetado de las mismas. Cada una de las líneas del los ficheros de texto contiene las etiquetas de una imagen. La estructura de cada línea es la siguiente:

```
<RUTA_IMG> <NUMERO_DE_ETIQUETAS>( <X0> <Y0> <ANCHO> <ALTO>)+
```

Para facilitar el posterior tratamiento, se ha realizado una transformación del formato de los ficheros de etiquetas al siguiente formato:

```
<RUTA_IMG>( <X0>,<Y0>,<ANCHO>,<ALTO>,<CLASE>)+
```

4.2. Estudio de los datos

En una fase inicial se ha realizado un análisis del tamaño de los baches con respecto al tamaño de la imagen. Esto es un aspecto importante a tener en cuenta de cara a determinar el algoritmo a utilizar para la detección de objetos. Los algoritmos de detección de objetos, en general se comportan peor cuanto más pequeños son los objetos a detectar.

Como se observa en la figura 2, la mayoría de los baches tienen una anchura inferior a 200 píxeles y una altura inferior a 50 píxeles. Este factor será tenido en cuenta en el preprocesamiento de las imágenes.

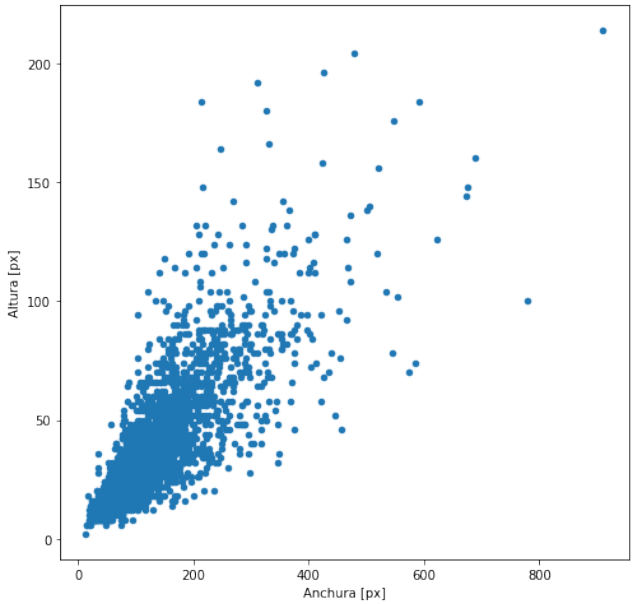


Figura 2: Tamaños de los baches en píxeles

También se ha realizado un estudio de la localización de los baches en las imágenes. Tal y como se ve en la figura 3, los baches están localizados principalmente en el centro de la imagen. La parte inferior se corresponde con el salpicadero del coche y la parte superior se corresponde con paisaje.

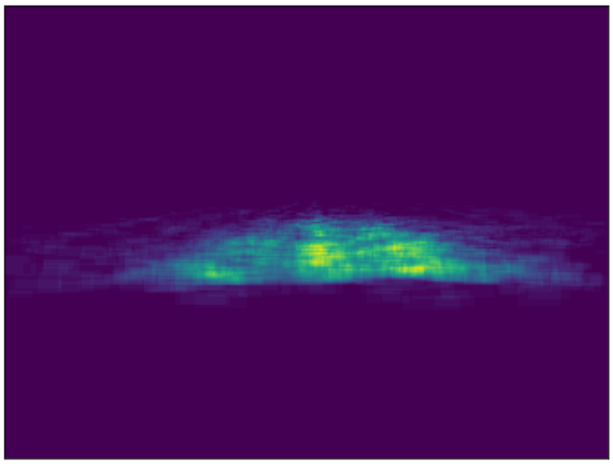


Figura 3: Localizaciones de los baches en las imágenes

4.3. Preprocesamiento de las imágenes

Tanto en la fase de entrenamiento, como para hacer una predicción, las imágenes van a ser redimensionadas al tamaño de la red neuronal, la cual tiene una relación de aspecto 1:1. Para redimensionar una imagen con una relación de aspecto 4:3, y al mismo tiempo, transformarla en una imagen con relación de aspecto 1:1, lo que se hace es redimensionar el lado más grande de la imagen manteniendo la relación de aspecto, es decir, aplicando el mismo factor de redimensionamiento al lado más pequeño. Una vez redimensionada, se rellena con gris la zona superior y la zona inferior de la imagen para cuadrarla. En la figura 4 se muestra un ejemplo gráfico.

Figura 4: A la izquierda la imagen original redimensionada a tamaño 920x690 px (manteniendo la relación de aspecto 4:3). A la derecha la imagen redimensionada con el relleno para que tenga una relación de aspecto 1:1 (920x920 px)

El redimensionamiento se hace en base al lado más grande de la imagen, que en el ejemplo anterior es la anchura. Para determinar el factor de redimensionamiento, se divide la anchura la imagen final entre la anchura de la imagen original, en este caso: $920/3680 = 0{,}25$. A continuación, se aplica este factor de redimensionamiento a ambos lados de la imagen, resultando en un tamaño de 920x690 píxeles. Por último se calcula el relleno que haría falta a cada lado de la imagen: $(920 - 690)/2 = 115$.

Siguiendo con este ejemplo, si en la imagen original hubiese un bache de tamaño 160x24 píxeles, y se aplicase este factor de redimensionamiento, el bache redimensionado tendría unas dimensiones de 40x6 píxeles, lo cual sería un tamaño bastante pequeño ya que únicamente tiene 6 píxeles de alto (de 920 que tiene la imagen).

Sin embargo, si previo al redimensionamiento de la imagen, se recortan los extremos izquierdo y derecho de la imagen, de tal forma que tenga una relación de aspecto 1:1, se consigue que el factor de redimensionamiento sea mayor y que

por tanto los baches redimensionados sean también más grandes. Esta técnica tiene un inconveniente, y es que la imagen original se está recortando, por lo que está habiendo una pérdida de información. Este inconveniente no es un impedimento, ya que en el apartado 4.3 se ha comprobado que la mayor parte de los baches están en el centro de las imágenes, y que recortando los extremos de las mismas la pérdida de información es mínima.

Para aplicar esta técnica, en primer lugar habría que calcular los recortes que hay que hacer a cada lado de la imagen original. Para ello se calcula la diferencia entre la anchura y la altura de la imagen y se divide por dos: $(3680 - 2760)/2 = 460$. Una vez recortada la imagen se calcula el factor de redimensionamiento: $920/2760 = 0{,}333$. Por último se aplicaría este factor de redimensionamiento a la altura y la anchura de la imagen.

Si aplicamos este nuevo factor de redimensionamiento al tamaño del bache del ejemplo anterior (160x24 píxeles), el tamaño del bache redimensionado sería 53x8 (un 75 % más grande).

5. Técnicas de Deep Learning y métodos de evaluación

5.1. Explicar las técnicas de DL que se van a utilizar en el proyecto

!!! TODO

- mencionar transfer learning
- redes convolucionales
- non maximal suppression

Para el entrenamiento se ha utilizado la técnica *hold-out*, que consiste en dividir el conjunto de datos en dos subconjuntos: uno que será utilizado para la fase de entrenamiento y el otro que será utilizado para evaluar el modelo entrenado. Dada la escasez de imágenes no se ha utilizado un conjunto de validación durante el entrenamiento.

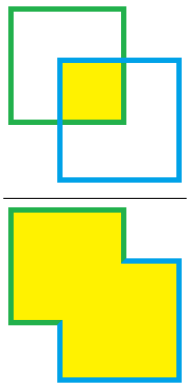
5.2. Explicar los métodos de evaluación que se van a utilizar en el proyecto

La métrica que se ha utilizado para evaluar el modelo obtenido es la AP (Average Precision), que es la métrica que se utiliza para evaluar modelos de detección de objetos.

Antes de explicar en qué consiste la métrica AP hay que explicar una serie de conceptos en los cuales está basada: IoU (intersección sobre la unión), precisión (precision) y sensibilidad (recall).

El concepto de IoU mide cuánto se solapan dos regiones: la predicha y la que debería ser detectada. Se calcula dividiendo la región obtenida mediante la intersección de la región predicha y la región a detectar entre la región obtenida mediante la unión de ambas regiones.

$$AP = \frac{intersección}{unión} =$$



La *precisión* (precision) mide la capacidad del modelo para detectar únicamente los objetos relevantes. Se calcula como el porcentaje de predicciones positivas acertadas frente a todas las predicciones positivas predichas:

$$precisión = \frac{TP}{TP + FP} = \frac{TP}{todas\ las\ predicciones\ positivas}$$

La *sensibilidad* (recall) mide la capacidad del modelo para detectar todos los objetos relevantes. Se calcula como el porcentaje de predicciones positivas acertadas frente a todas las existentes:

$$sensibilidad = \frac{TP}{TP + FN} = \frac{TP}{todas\ las\ regiones\ a\ detectar}$$

Tanto en el cálculo de la *precisión* como en el cálculo de la *sensibilidad*, para determinar si una predicción es positiva, se utiliza el IoU. Se define un umbral para el IoU (normalmente suele ser 0.5) y si se supera dicho umbral, la predicción es considerada una predicción positiva.

La métrica AP se calcula como el área debajo de la curva *precisión-sensibilidad* (precision-recall). En el eje de las abscisas se representa la *sensibilidad* (recall) y en el eje de las ordenadas se representa la *precisión* (precision).

A continuación se va a mostrar un ejemplo práctico de cómo se calcula la AP. Para este ejemplo se dispone de una serie de imágenes con un total de 4 baches a detectar. En la tabla 1 se puede ver el cálculo de la *precisión* y de la *sensibilidad* para las predicciones obtenidas. La columna *Positivo* indica si la predicción es positiva, es decir, si el valor de IoU supera el umbral definido, que en este caso es 0.5. Las columnas TP y FP muestran el acumulado de sus respectivos valores.

IoU	Positivo	TP	FP	Precisión	Sensibilidad
0.912933	1	1	0	1.000000	0.25
0.711111	1	2	0	1.000000	0.50
0.387983	0	2	1	0.666667	0.50
0.387983	0	2	2	0.500000	0.50
0.387983	0	2	3	0.400000	0.50
1.000000	1	3	3	0.500000	0.75
0.225986	0	3	4	0.428571	0.75
0.225986	0	3	5	0.375000	0.75
1.000000	1	4	5	0.444444	1.00

Tabla 1: Cálculo de la precisión y sensibilidad para las predicciones

Una vez se tienen calculados los valores de *precisión* y de *sensibilidad* se calcula la curva *precisión-sensibilidad* como se puede ver en la figura 5. Para realizar

el cálculo del área debajo de la curva se realiza un suavizado de la misma. Este suavizado consiste en establecer como valor de *precisión* para un determinado valor de *sensibilidad*, el valor de *precisión* más alto que se encuentre a su derecha. Por ejemplo, para la *sensibilidad* 0.6 se establece como valor de *precisión* el valor más alto a su derecha, que en este caso es 0.5. En color naranja se puede ver la curva suavizada.

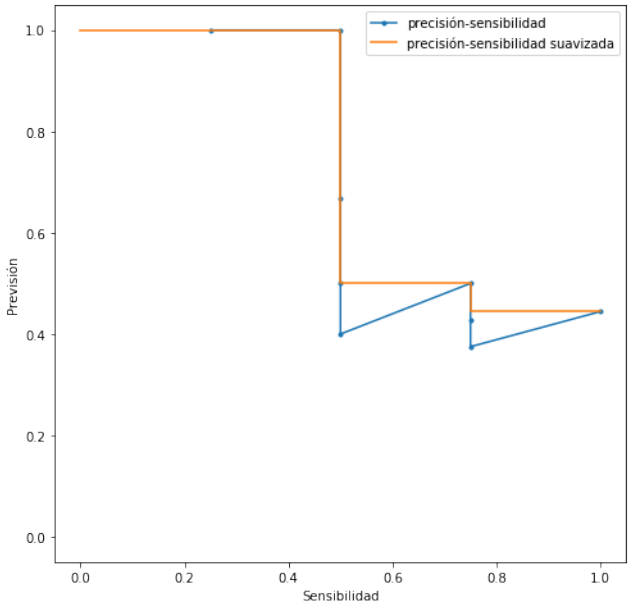


Figura 5: Curva precisión-sensibilidad

Por lo que finalmente, para este ejemplo, el cálculo del AP sería:

$$AP = (0{,}5-0)\cdot 1+(0{,}75-0{,}5)\cdot 0{,}5+(1-0{,}75)\cdot 0{,}44 = 0{,}5+0{,}125+0{,}11 = \mathbf{0.735}$$

6. Implementación y evaluación de las técnicas

6.1. Detalles de la implementación de las técnicas de DL aplicadas

!!! TODO

6.2. Evaluación de las técnicas

Se han entrenado dos versiones de YOLO: la versión 3 y la versión 3 tiny. Inicialmente se entrenó únicamente la versión 3, pero al ejecutarla en un dispositivo móvil se observó que el rendimiento lo hacía inutilizable, es este el motivo por el cual que se ha entrenado la versión tiny.

Para cada una de estas versiones se han entrenado varios modelos con distintos tamaños de red, por dos motivos principalmente: por una cuestión de rendimiento a la hora de ejecutar el modelo en un dispositivo móvil y por analizar cómo varía la precisión del modelo cambiando el tamaño de la red.

Además se han utilizado distintos conjuntos de entrenamiento para entrenar todas las variantes del modelo. El primero de los conjuntos de entrenamiento se corresponde con el conjunto íntegro original (denominado *completo*). Los resultados obtenidos con este conjunto de entrenamiento obtuvieron unos valores bajos para la métrica AP, y tras analizar los motivos, se observó que había una gran cantidad de baches demasiado pequeños que podían ser los causantes malos resultados. Por este motivo, se han utilizado dos conjuntos de entrenamiento adicionales aplicando filtros sobre los baches. En el primero de estos conjuntos de entrenamiento adicionales se han filtrado los baches con tamaño superior a 75x30 píxeles (denominado *filtro 75x30*) y en el segundo se han filtrado los baches con tamaño superior a 100x40 píxeles (denominado *filtro 100x40*). Para cada uno de estos conjuntos de entrenamiento adicionales se ha creado también su correspondiente conjunto de evaluación aplicando el mismo filtro.

Con todos los modelos resultantes obtenidos se ha realizado una doble evaluación. Por un lado se han evaluado con los conjuntos de evaluación correspondientes para cada uno de los conjuntos de entrenamiento (resultados en la tabla 2). Por otro lado se han evaluado con un conjunto de imágenes generado (resultados en la tabla 3). Este conjunto de evaluación (denominado *propio*) se compone de unas 30 imágenes de 4032x3024 píxeles, con unos 60 baches en total, obtenido desde la acera (a diferencia del original que fue obtenido desde el coche) y compuesto por fotos realizadas en España (a diferencia del original que fueron realizadas en Sudáfrica).

Versión YOLO	Tamaño	Juego datos	Épocas	Mejor AP
V3	256	completo	43	0.0747
V3	256	filtro 100x40	93	0.3077
V3	256	filtro 75x30	88	0.2513
V3	416	completo	18	0.1467
V3	416	filtro 100x40	93	0.4161
V3	416	filtro 75x30	93	0.3611
V3	640	completo	13	0.0186
V3	640	filtro 100x40	63	0.5475
V3	640	filtro 75x30	53	0.4106
V3 Tiny	256	completo	144	0.0046
V3 Tiny	256	filtro 100x40	136	0.0510
V3 Tiny	256	filtro 75x30	153	0.0392
V3 Tiny	416	completo	153	0.0145
V3 Tiny	416	filtro 100x40	153	0.1307
V3 Tiny	416	filtro 75x30	146	0.0869

Tabla 2: Resultados obtenidos con los conjuntos de evaluación originales

Versión YOLO	Tamaño	Juego datos	Épocas	Mejor AP
V3	256	propio completo	43	0.0289
V3	256	propio filtro 100x40	93	0.1018
V3	256	propio filtro 75x30	88	0.0179
V3	416	propio completo	18	0.0354
V3	416	propio filtro 100x40	93	0.0089
V3	416	propio filtro 75x30	93	0.0294
V3	640	propio completo	13	0.0017
V3	640	propio filtro 100x40	63	0.0342
V3	640	propio filtro 75x30	53	0.0961
V3 Tiny	256	propio completo	144	0.0086
V3 Tiny	256	propio filtro 100x40	136	0.0232
V3 Tiny	256	propio filtro 75x30	153	0.0371
V3 Tiny	416	propio completo	153	0.0000
V3 Tiny	416	propio filtro 100x40	153	0.0000
V3 Tiny	416	propio filtro 75x30	146	0.0006

Tabla 3: Resultados obtenidos con el conjunto de evaluación propio

7. Resultados

7.1. Resultados del proyecto

A continuación se van a mostrar algunos ejemplos de las predicciones obtenidas con los modelos que mejores resultados han obtenido, que son los que han sido entrenados con el juego de datos de entrenamiento *filtro 100x40*.

(a) Baches a detectar **(b)** YOLO v3 tamaño 256x256

(c) YOLO v3 tamaño 416x416 **(d)** YOLO v3 tamaño 640x640

Figura 6: Ejemplo de predicción con modelos YOLO v3 de distintos tamaños. Arriba a la izquierda, la imagen con los baches a detectar en azul y en amarillo los baches que fueron descartados por el filtro 100x40. En el resto de las imágenes se pueden ver las predicciones realizadas en rojo.

En la figura 6 se muestran las predicciones realizadas por los modelos *YOLO v3*. Se trata de una imagen en la que originalmente se han etiquetado 2 baches, uno de los cuales se ha descartado por ser demasiado pequeño. Se puede observar que existe un defecto en el etiquetado, ya que entre los dos baches etiquetados

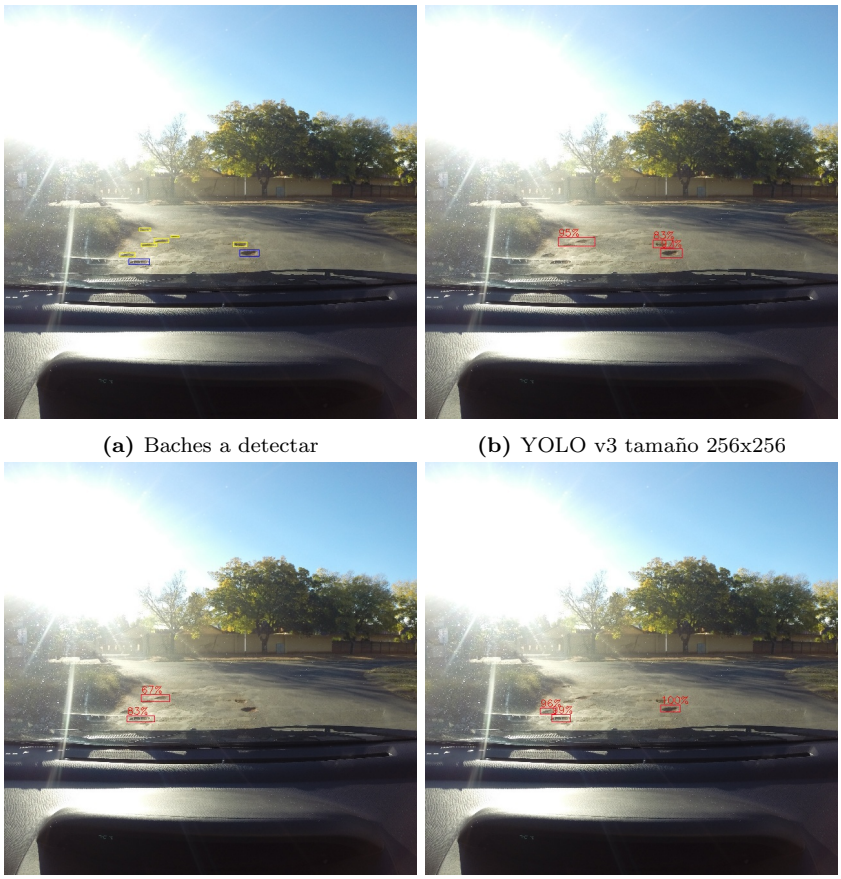
existe un tercer bache sin etiquetar. Aún habiendo filtrado los baches pequeños se puede comprobar que el modelo es capaz de detectarlos (en los modelos de tamaño 416x416 y 640x640). También se puede observar que el modelo de tamaño 640x640 es capaz de detectar el bache sin etiquetar.

En la figura 7 se muestran las predicciones realizadas por los modelos *YOLO v3 tiny* para la misma imagen. Únicamente el modelo de tamaño 416x416 es capaz de detectar el bache, aunque lo hace de manera poco precisa ya que la región detectada es demasiado grande y abarca también al bache sin etiquetar.

(a) YOLO v3 tiny tamaño 256x256 **(b)** YOLO v3 tiny tamaño 416x416

Figura 7: Misma predicción que en la figura 6, pero en esta ocasión con modelos YOLO v3 tiny.

En la figura 8 se muestran más predicciones realizadas por los modelos *YOLO v3*. En esta ocasión se trata de una imagen en la que hay múltiples baches, de los cuales únicamente se han mantenido 2 y el resto se han descartado por tener un tamaño demasiado pequeño. En esta ocasión los 3 modelos detectan baches de forma correcta. El único modelo que detecta los baches esperados es el de tamaño 640x640. Además de detectar los baches detectados, es capaz de detectar uno de los baches que fue descartado por tamaño. Los otros dos modelos de tamaño inferior únicamente detectan uno de los baches esperados, aunque son capaces de detectar también algunos de los baches descartados.



(a) Baches a detectar

(b) YOLO v3 tamaño 256x256

(c) YOLO v3 tamaño 416x416

(d) YOLO v3 tamaño 640x640

Figura 8: Ejemplo de predicción con modelos YOLO v3 de distintos tamaños. Arriba a la izquierda, la imagen con los baches a detectar en azul y en amarillo los baches que fueron descartados por el filtro 100x40. En el resto de las imágenes se pueden ver las predicciones realizadas en rojo.

En la figura 9 se muestran las predicciones realizadas por los modelos *YOLO v3 tiny* para el segundo ejemplo. En esta ocasión ambos modelos son capaces de detectar baches de forma correcta. Además el modelo de tamaño 416x416 es capaz de identificar los dos baches esperados.

(a) YOLO v3 tiny tamaño 256x256 **(b)** YOLO v3 tiny tamaño 416x416

Figura 9: Misma predicción que en la figura 8, pero en esta ocasión con modelos YOLO v3 tiny.

8. Conclusiones

8.1. Evaluación del proyecto

!!! TODO

8.2. Alternativas y posibles mejoras que podrían haberse aplicado al proyecto (trabajos futuros)

!!! TODO

8.3. Conclusiones personales

!!! TODO

Referencias

[1] J.R.R. Uijlings et al. *Selective Search for Object Recognition*. 2012. URL: http://www.huppelen.nl/publications/selectiveSearchDraft.pdf.

[2] Ross Girshick et al. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2014. URL: https://arxiv.org/pdf/1311.2524.pdf.

[3] dicastro. URL: https://github.com/dicastro/tfm.

[4] experiencor. URL: https://github.com/experiencor/keras-yolo3.

[5] Ross Girshick. *Fast R-CNN*. 2015. URL: https://arxiv.org/pdf/1504.08083.pdf.

[6] HoracceFeng. URL: https://github.com/HoracceFeng/keras-yolo3-tiny.

[7] Jonathan Hui. *mAP (mean Average Precision) for Object Detection*. URL: https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173. (accedido: 29/08/2019).

[8] Ayuntamiento de Madrid. *Informe de sugerencias y reclamaciones del ayuntamiento de Madrid de 2018*. URL: https://www.madrid.es/UnidadesDescentralizadas/UGDefensorContribuyente/05_Publicaciones/informes/INFORME_1%20semestre_2018_SyRversion_definitiva_18_diciembre.pdf.

[9] Ayuntamiento de Madrid. *Sistema de Sugerencias y Reclamaciones*. URL: https://www.madrid.es/portales/munimadrid/es/Inicio/El-Ayuntamiento/Contacto/Sugerencias-y-reclamaciones?vgnextfmt=default&vgnextchannel=5eadc1ab4fd86210VgnVCM2000000c205a0aRCRD. (accedido: 01/09/2019).

[10] Felipe Muller. *Nienaber Potholes 2 Complex*. URL: https://www.kaggle.com/felipemuller5/nienaber-potholes-2-complex. (accedido: 26/08/2019).

[11] Arthur Ouaknine. *Review of Deep Learning Algorithms for Object Detection*. URL: https://medium.com/zylapp/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852. (accedido: 03/09/2019).

[12] Dhruv Parthasarathy. *A Brief History of CNNs in Image Segmentation: From R-CNN to Mask R-CNN*. URL: https://blog.athelas.com/a-brief-history-of-cnns-in-image-segmentation-from-r-cnn-to-mask-r-cnn-34ea83205de4. (accedido: 03/09/2019).

[13] Javier Rey. *Object Detection with Deep Learning: The Definitive Guide*. URL: https://tryolabs.com/blog/2017/08/30/object-detection-an-overview-in-the-age-of-deep-learning. (accedido: 29/08/2019).

[14] Ankit Sachan. *Zero to Hero: Guide to Object Detection using Deep Learning: Faster R-CNN,YOLO,SSD*. URL: https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/amp/. (accedido: 03/09/2019).

[15] Joyce Xu. *Deep Learning for Object Detection: A Comprehensive Review*. URL: https://towardsdatascience.com/deep-learning-for-object-detection-a-comprehensive-review-73930816d8d9. (accedido: 03/09/2019).