Detección de objetos basada en Deep Learning y aplicada a vehículos autónomos

Tesis de Máster

KISA/ICSI

Ignacio Arriola Oregui

Directores

J. Javier Yebes Torres

Ignacio Arganda Carreras

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
MDE
Master eta Doktorego Eskola
Escuela de Máster y Doctorado
Master and Doctoral School
informatika fakultatea
facultad de informática

Konputazio Ingenieritza eta sistema adimendunak
Ingeniería computacional y sistemas inteligentes

Agradecimientos

En primer lugar me gustaría agradecer a mis directores, a Ignacio por estar siempre disponible cuando he necesitado ayuda, desde el primer día acompañándome a la entrevista hasta el último, pendiente siempre de la evolución del trabajo. A Javier por haberme guiado a lo largo de mis prácticas y por preocuparse en todo momento por que mi estancia fuera la mejor posible, gracias por haberlo conseguido.

También quiero agradecer a David, a Nerea y a Mikel por estar siempre dispuestos a ayudarme cuando tenía alguna duda relacionada con Tensorflow. A Jorge por haber compartido todas esas horas peleando codo con codo con los problemas que nos surgían. Y por último pero no menos importante a Asier, por ser un magnífico compañero y amigo y por compartir su infinita sabiduría sobre redes neuronales siempre que lo he necesitado.

Resumen

En este trabajo se han utilizado las redes neuronales profundas para el cometido de la detección de objetos en imágenes. En concreto, el trabajo se ha desarrollado en el contexto de los vehículos autónomos.

Se han entrenado y comparado tres redes Faster-RCNN para la detección de peatones partiendo desde diferentes inicializaciones de sus parámetros para estudiar la influencia de la transferencia del aprendizaje.

Se ha analizado un caso práctico de detección de baches en carretera. Se ha recopilado una base de datos, partiendo de repositorios y de etiquetado manual. Se ha explorado la base de datos para inicializar el entrenamiento de una manera más efectiva. Se ha evaluado y comparado el rendimiento de tres modelos Faster-RCNN para la detección de baches con diferentes extractores de características.

El trabajo se ha desarrollado utilizando la librería Tensorflow y los modelos se han probado en el dispositivo Nvidia Drive PX2, el cual está diseñado para la investigación en conducción autónoma.

Índice general

1. Introducción, motivación y objetivos **3**

2. Estado del arte **6**

2.1. Vehículos autónomos 6

2.2. Detección automática de objetos mediante redes neuronales artificiales 7

3. Redes Neuronales Artificiales para detección de objetos en imágenes **10**

3.1. La neurona 10

3.2. Estructuras de las redes neuronales 11

3.2.1. Redes *feedforward* 12

3.2.2. Redes con conexiones residuales 13

3.3. Algoritmos de entrenamiento 13

3.3.1. Función de coste 13

3.3.2. Descenso de gradiente 14

3.3.3. Descenso de gradiente estocástico 15

3.3.4. Propagación hacia atrás 16

3.3.5. Desvanecimiento del gradiente 18

3.4. Teorema de aproximación universal y análisis de profundidad 19

3.5. Medidas para prevenir el sobreajuste 20

3.5.1. Regularización *dropout* 20

3.6. Redes neuronales convolucionales 21

3.6.1. Convolución 22

3.6.2. Pooling 24

3.7. Redes para detección de objetos: Faster R-CNN 24

3.7.1. Redes de propuesta de región de interés 25

3.7.2. RoI *Pooling* 27

3.7.3. Clasificador Fast-RCNN 28

4. Transferencia del aprendizaje: Comparativa de inicialización de parámetros 29

4.1. Comparativa de Faster-RCNN Resnet 101 partiendo desde COCO y desde KITTI . 30

4.1.1. Métricas de evaluación 31

4.1.2. Resultados 32

4.1.3. Discusión de los resultados 34

4.2. Detección de peatones con Faster-RCNN Resnet 101 partiendo con inicialización aleatoria. 34

4.2.1. Discusión de los resultados 36

5. Evaluación de modelos DNN en Nvidia DrivePX2 38

5.1. Nvidia Drive PX2 38

5.2. Detección de objetos basada en modelos DNN 39

6. Caso práctico: Detector de baches en carretera 41

6.1. Análisis de la base de datos 41

6.1.1. Datos de entrenamiento 42

6.1.2. Datos de evaluación 43

6.2. Evaluación de modelos de redes profundas para la detección de baches 43

6.2.1. Resultados 44

6.2.2. Discusión de los resultados 47

7. Conclusiones 48

Capítulo 1

Introducción, motivación y objetivos

En los últimos años se han producido grandes avances tecnológicos en los sistemas de transporte autónomos, pero aún quedan muchos aspectos por investigar y desarrollar hasta que sea posible ver coches completamente autónomos y sin conductor en las carreteras. La investigación académica en este campo aumenta cada año. En el sector industrial se han creado grandes alianzas entre empresas de Internet, desarrollo software, hardware y las grandes marcas del sector de la automoción. En los dos ámbitos existe una fuerte inversión en I+D para diseñar, desarrollar y probar los vehículos autónomos del futuro.

Hasta hace unos años, los sistemas de asistencia a la conducción se basaban principalmente en la percepción del entorno mediante cámaras y sensores de proximidad. Sus señales se procesaban de forma independiente para resolver tareas muy concretas. Sin embargo, el estado del arte actual y los retos futuros del coche sin conductor requieren sistemas complejos con múltiples sensores y potentes módulos hardware y software. En respuesta a dichas necesidades, están surgiendo nuevos sistemas para adquisición, sincronización y procesamiento de múltiples flujos de datos y específicamente diseñados para instalarse en vehículos autónomos. Por un lado, dichos sistemas deben contar con módulos robustos, fiables y de rápida respuesta para las tareas más críticas. Por otro lado, deben ofrecer múltiples interfaces para los sensores e interconexión con otros sistemas y capacidades de procesamiento para grandes volúmenes de datos (imágenes 2D y nubes de puntos 3D) así como APIs de programación ágiles para la I+D de funcionalidades de alto nivel. Por ejemplo, el entendimiento de escenas y la extracción de información semántica útil y fácil de usar.

La percepción del entorno es un elemento clave en el desarrollo de vehículos autónomos. Dado que el automóvil comparte la vía con otros participantes en el tráfico, este debe detectar estos participantes, así como las señales y los posibles obstáculos para evitar accidentes [1]. Para este cometido, existen sistemas de detección automática de objetos en imágenes. En los últimos años, los métodos han evolucionado desde técnicas básicas de procesamiento de imágenes hasta complejos algoritmos de aprendizaje automático (o *Machine Learning).* Además, desde que una red neuronal profunda llamada AlexNet [2] obtuviese la victoria en el concurso de clasificación LSVRC2012, las redes neuronales artificiales y las técnicas de aprendizaje profundo o *Deep Learning* se han revelado como una alternativa muy prometedora en el desarrollo de los sistemas de clasificación y detección de imágenes.

Para poder procesar los datos en tiempo real desde dentro del vehículo es necesario contar con un hardware capaz de satisfacer las necesidades de la conducción autónoma, comprender el entorno procesando la información que obtiene desde los sensores y tomar decisiones adecuadas. La plataforma Nvidia Drive PX2[1] ofrece un dispositivo muy avanzado que combina aprendizaje profundo, fusión de sensores y visión envolvente para cambiar la experiencia de la conducción. Se trata de un dispositivo para realizar I+D en sistemas de asistencia a la conducción con el objetivo de comprender en tiempo real lo que sucede alrededor del vehículo, localizarse con precisión en un mapa de alta definición y planificar una ruta segura. Dichas tareas presentan una serie de retos aún sin

[1] `https://www.nvidia.com/en-us/self-driving-cars/drive-platform/`

resolver por completo. No obstante, el estado del arte actual está muy avanzado y existen distintas líneas de investigación focalizadas en aspectos concretos.

Este trabajo fin de máster se centra en estudiar, implementar y evaluar redes neuronales artificiales para la detección de objetos en imágenes de carretera. En primer lugar, se trabajará la teoría básica de las redes neuronales. En segundo lugar, se estudiarán las redes profundas que se utilizan para el procesado de imágenes y detección de objetos. Después, se realizarán una serie de experimentos para probar el rendimiento de los detectores de redes neuronales.

En el apartado experimental, primero se estudiará la importancia del fenómeno de la transferencia del aprendizaje. En las redes para detección de objetos, es una práctica habitual empezar el entrenamiento de las redes desde parámetros previamente entrenados en lugar de empezar desde cero. En las redes profundas, o *Deep Neural Networks* (DNNs), es necesaria una base de datos muy extensa, y por tanto, mucho tiempo de entrenamiento para poder obtener buenos resultados de detección. Por esa razón, es habitual partir de una red previamente entrenada y afinar los valores de los parámetros para el caso específico que se desea detectar.

En segundo lugar, se utilizará el Nvidia Drive PX2 para hacer pruebas relacionadas con la fase de inferencia, y se probará el tiempo de inferencia en diferentes modelos de referencia.

En el último experimento, se analizará un caso práctico de detección de baches en carretera. El estudio de este caso particular viene motivado por varias causas. En primer lugar, a la hora de desarrollar vehículos autónomos, es necesario percibir el entorno de la manera más precisa posible, y esto también implica detectar las imperfecciones que existen en la carretera, entre las cuales se encuentran los baches. Por otro lado, los modelos de detección que se estudian en este trabajo han sido entrenados en el departamento de sistemas de transporte inteligente e ingeniería de Vicomtech[2] para el proyecto AUTOPILOT H2020 [3], en el cual los baches detectados se enviarán a una plataforma de internet de las cosas (IoT) para que los automóviles conectados a esa plataforma puedan recibir información de dónde se encuentran las imperfecciones en la carretera.

Para ello se utilizará la librería Tensorflow[3], desarrollada por Google, la cual es una herramienta que permite un fácil desarrollo de redes neuronales profundas. Tensorflow está implementado para trabajar con unidades centrales de procesamiento (CPU) o con unidades de procesamiento gráfico (GPU). Estas últimas, por su capacidad de realizar múltiples cálculos en paralelo, habilitan que el aprendizaje de las redes neuronales se acelere muy considerablemente en comparación con las CPUs.

El trabajo está dividido en siete capítulos, de los cuales este capítulo introductorio es el primero. El resto de capítulos están ordenados de la siguiente forma, y tratan los siguientes objetivos:

- **Estado del arte**. Se estudia el contexto de la I+D de detectores de objetos automáticos aplicados a los sistemas de asistencia a la conducción y vehículos autónomos.
- **Redes Neuronales Artificiales para detección de objetos en imágenes**. El objetivo de este capítulo es entender y profundizar en los conocimientos básicos de las redes neuronales artificiales y el *Deep Learning*: desde las redes más simples hasta las redes para detección de objetos en imágenes.
- **Transferencia del aprendizaje: comparativa de inicialización de parámetros**. Se estudiará la diferencia que supone partir de los parámetros entrenados con dos bases de datos diferentes en la misma arquitectura de red, para ello se entrenarán y compararán dos modelos con el mismo procedimiento y misma arquitectura, partiendo de dos inicializaciones distintas para el cometido de la detección de peatones. Después se entrenará la misma red partiendo desde cero para ver las diferencias reales entre aplicar la transferencia del aprendizaje o no aplicarla. Se aprenderá a evaluar modelos de *Deep Learning* con Tensorflow.
- **Evaluación de modelos DNN en Nvidia DrivePX2**. Los objetivos de este capítulo son analizar las características del hardware Nvidia Drive PX2, realizar pruebas para observar la capacidad/limitaciones del dispositivo en comparación con otros dispositivos de referencia y aprender a trabajar con computadoras de arquitectura ARM.

[2] `http://www.vicomtech.org/d1/sistemas-de-transporte-inteligentes-e-ingenieria`
[3] `https://www.tensorflow.org/`

- **Caso práctico: Detector de baches en carretera**. Se analizará un caso de uso real en un detector de baches en carretera. Se construirá una base de datos y se evaluarán los modelos entrenados a partir de esta.
- **Conclusiones**. Tras exponer el análisis y tareas realizadas, se presentarán las conclusiones del trabajo, en las que se detalla el alcance de los resultados obtenidos. Al mismo tiempo se analizarán las posibilidades de mejora y continuidad del trabajo.

Capítulo 2

Estado del arte

El estado del arte de este trabajo se ha dividido en dos secciones. La primera cuenta los avances de los vehículos autónomos desde su origen hasta el desarrollo más reciente, para acabar poniendo el foco en la percepción del entorno. La segunda sección se centra en las redes neuronales artificiales para percibir y detectar los objetos del entorno.

2.1. Vehículos autónomos

Los últimos años han sido testigos de un progreso sorprendente en los campos relacionados con la IA (Inteligencia Artificial), como la visión artificial, el aprendizaje automático y los vehículos autónomos. Desde las primeras demostraciones exitosas en la década de 1980 [4, 5, 6], se ha avanzado mucho en el campo de los vehículos autónomos. Sin embargo, a pesar de estos avances, es seguro creer que la navegación totalmente autónoma en entornos complejos todavía está a décadas de distancia. La razón de esto es doble: en primer lugar, los sistemas autónomos requieren de inteligencia artificial que opere ante situaciones impredecibles. En segundo lugar, las decisiones requieren una percepción precisa, sin embargo, la mayoría de los sistemas de visión por computadora existentes producen errores inaceptables para la navegación autónoma [1].

Muchas instituciones gubernamentales de todo el mundo comenzaron varios proyectos para explorar sistemas inteligentes de transporte (ITS). El proyecto PROMETHEUS comenzó en 1986 en Europa e involucró a más de 13 fabricantes de vehículos, varias unidades de investigación de gobiernos y universidades de 19 países europeos. Uno de los primeros proyectos en los Estados Unidos fue Navlab [6] de la Carnegie Mellon University, que logró un hito importante en 1995 al completar la primera conducción autónoma de Pittsburgh, PA y Sand Diego, CA. Después de que muchas universidades, centros de investigación y compañías automotrices lanzaran iniciativas, el gobierno de los EE. UU. estableció el Consorcio Nacional de Sistemas de Carreteras Automatizadas (NAHSC) en 1995. De forma similar lo hizo Japón en 1996 (Asociación Avanzada de Investigación de Sistemas de Autopistas Cruise). Poco después, en [7] vaticinaron que la potencia informática necesaria para desarrollar vehículos autónomos estaría cada vez más cerca, pero las dificultades como los reflejos, las carreteras mojadas, la luz directa del sol, los túneles y las sombras todavía dificultarían la interpretación de los datos. Por lo tanto, sugirieron mejoras en las capacidades de los sensores.

Waymo (división de vehículos autónomos de Google) y Tesla son las compañías que actualmente se encuentran en la fase más avanzada del desarrollo de coches autónomos. Google comenzó su proyecto de automóvil autónomo en 2009 y completó más de 1,498,000 millas de forma autónoma hasta marzo de 2016 en Mountain View, California, Austin, TX y Kirkland, WA. Diferentes sensores (por ejemplo, cámaras, radares, LiDAR, GPS) permiten detectar peatones, ciclistas, vehículos, obras en carreteras, etc. en todas las direcciones. De acuerdo con sus informes de accidentes, los automóviles autónomos de Google estuvieron involucrados solo en 14 accidentes, y 13 de los 14 incidentes fueron causados por terceros. En 2016, el proyecto continuó con el nombre de Waymo[1],

[1] `https://www.waymo.com`

que es una compañía independiente de tecnología de auto-conducción.

Por otro lado, Tesla Autopilot[2] es un sistema avanzado de asistencia a la conducción desarrollado por Tesla que se lanzó por primera vez en 2015 con la versión 7 de su software. El nivel de automatización del sistema permite la automatización total, pero requiere la plena atención del conductor para tomar el control si es necesario. A partir de octubre de 2016, todos los vehículos producidos por Tesla estaban equipados con ocho cámaras, doce sensores ultrasónicos y un radar orientado hacia adelante para permitir la capacidad de conducción automática.

En [8] un automóvil condujo de forma autónoma el 98 % del tiempo desde Holmdel hasta Atlantic Highlands en el condado de Monmouth NJ, así como otras 10 millas en Garden State Parkway sin intervención. Para cumplir este objetivo, se utilizó una red neuronal convolucional que predice el control del vehículo directamente desde las imágenes que obtiene del exterior, utilizando la computadora Nvidia Drive PX2.

Para desarrollar sistemas de conducción autónoma competente es necesario percibir los posibles peligros que aparecen en la carretera. En [9] se ha realizado un trabajo de inspección de posibles daños en el asfalto utilizando imágenes capturadas desde teléfonos móviles. En [10] se construye un sistema de detección de baches en carretera mediante técnicas de aprendizaje automático. Los resultados obtenidos en una base de datos de 53 imágenes que contienen 97 baches etiquetados se muestran en la Tabla 2.1.

Verdaderos positivos	**Falsos positivos**	**Falsos negativos**	**Precisión (%)**	**Exhaustividad (%)**
72	16	24	81.8	74.4

Tabla 2.1: Resultados del algoritmo de detección de baches presentado en [10].

2.2. Detección automática de objetos mediante redes neuronales artificiales

Los enfoques de detección de objetos basados en características y clasificadores de aprendizaje automático han sido muy fructíferos hasta tiempos recientes [11]. Sin embargo, cuando se aplican a diferentes tareas o se adaptan para desafíos adicionales, estos requieren de un ajuste de parámetros y una reducción de dimensionalidad para lograr un rendimiento razonable [12].

Por otro lado, las tendencias recientes en Deep Learning han logrado un rendimiento de detección impresionante en varios desafíos [2], incluida la detección de objetos [13]. Además, hoy en día, varios modelos básicos de DNN se comparten públicamente y se pueden utilizar como paso de inicialización para un mayor entrenamiento y puesta a punto en diferentes tareas de clasificación de objetos.

El dataset KITTI [14] es una de las bases de datos más populares en el contexto de la detección de objetos para vehículos autónomos. Para la detección de peatones en concreto también es muy popular el la base de datos de Caltech [15].

Las redes neuronales convolucionales permitieron una mejora significativa en el rendimiento de los detectores de objetos. Al principio, estas redes utilizaban un enfoque de ventana deslizante [16]. Sin embargo, la localización precisa de los objetos es un desafío debido a los grandes campos receptivos. Más adelante, en [17] se proponen R-CNNs para solucionar el problema de la localización con un paradigma de reconocimiento usando regiones. Generan propuestas de región mediante búsqueda selectiva [18], extraen un vector de características de longitud fija para cada propuesta usando una red convolucional y clasifican cada región con un SVM lineal. Las redes basadas en regiones son computacionalmente costosas, pero se han propuesto varias mejoras para reducir la carga computacional [19, 20]. En [19] utilizan la agrupación de pirámides espaciales que permite calcular un mapa de características convolucionales para toda la imagen con solo una ejecución de la red convolucional en contraste con la R-CNN que debe aplicarse en muchas regiones de imágenes. En [20] mejoran aún más con un algoritmo de entrenamiento de una sola etapa que aprende de forma

[2] `https://www.tesla.com/autopilot`

conjunta a clasificar propuestas de objetos y refinar sus ubicaciones espaciales. A pesar de que estas redes basadas en regiones han demostrado ser muy exitosas en el benchmark PASCAL VOC, no pudieron lograr un rendimiento similar en KITTI. La razón principal es que el conjunto de datos KITTI contiene objetos en diferentes escalas y objetos pequeños que a menudo están muy ocluidos o truncados. Estos objetos son difíciles de detectar usando las redes basadas en regiones. Por lo tanto, se han propuesto varios métodos para obtener mejores propuestas de objetos [21, 22, 23].

En [21] se expone una Red de Propuesta de Región (RPN) la cual comparte características convolucionales de imagen completa con la red de detección y, por lo tanto, no aumenta los costos computacionales. Las RPN se entrenan de extremo a extremo para generar propuestas de región de alta calidad que después se clasifican utilizando el detector Fast R-CNN[20]. En [24] utilizan información 3D estimada a partir de un par de cámaras estéreo para extraer mejores propuestas de cuadro delimitador. Colocan cuadros de candidatos 3D sobre el *ground-truth* y los puntúan usando características de nubes de puntos 3D. Por último, una red convolucional que utiliza la información contextual y tiene una función de coste multitarea calcula las coordenadas y orientación del objeto mediante regresión. Inspirado por este enfoque, en [22] se expone un método que aprende a generar propuestas de objetos 3D de clase específica para imágenes monoculares, haciendo uso de modelos contextuales y semánticos. Ambos métodos logran resultados comparables al método de mejor rendimiento en todas las tareas de detección de objetos mientras que superan a todos los demás métodos en ejemplos fáciles de la clase coche de KITTI (Tabla 2.2).

En [23], se presenta un enfoque alternativo. En el caso de objetos pequeños, es más probable que ocurra una fuerte activación de neuronas convolucionales en las primeras capas. Se utiliza un *pooling* dependiente de la escala que permite representar un cuadro delimitador candidato utilizando las características convolucionales de la escala correspondiente. Además, proponen clasificadores de rechazo en cascada en capas, que tratan las características convolucionales en las primeras capas como clasificadores débiles, para eliminar de manera eficiente las propuestas de objetos negativos. El enfoque de *pooling* dependiente de escala propuesto es uno de los mejores métodos en todas las tareas (Tabla 2.2).

En [25] se propone una red convolucional multiescala que consiste en una subred de propuesta de regiones y una subred de detección. La red de propuestas realiza la detección en múltiples capas de salida y estos detectores de escala complementarios se combinan para producir un potente detector de objetos a múltiples escalas. Este detector supera a todos los otros métodos en detectar peatones y ciclistas en KITTI, mientras que es el segundo mejor detectando coches (Tabla 2.2). En [26] se propone una red de propuesta de región que utiliza información de subcategoría obtenida de 3DVP [27], para guiar el proceso de generar propuestas, y una red de detección para detección y clasificación de subcategoría. La información de la subcategoría les permite superar a todos los demás métodos para la tarea de detección en la clase coche de KITTI (Tabla 2.2).

A lo largo del trabajo se empleará la red Faster-RCNN. Además de ser una red que muestra un gran rendimiento en detección de objetos, estando entre las 7 mejores en el benchmark de KITTI, cuenta con implementaciones eficientes en Tensorflow y sobre todo con varias redes con parámetros preentrenados y de código abierto en el repositorio de [28], lo cual hace que sea un candidato apropiado para la iniciación en redes para la detección de objetos.

Método	Moderado	Fácil	Difícil	Tiempo de ejecución
SubCNN [26]	**89.04** %	90.81 %	**79.27** %	2 s / GPU
MS-CNN [25]	89.02 %	90.03 %	76.11 %	0.4 s / GPU
SDP+RPN [23]	88.85 %	90.14 %	78.38 %	0.4 s / GPU
Mono3D [22]	88.66 %	92.33 %	78.96 %	4.2 s / GPU
3DOP [24]	88.64 %	**93.04** %	79.10 %	3s / GPU
SDP+CRC [23]	83.53 %	90.33 %	71.13 %	0.6 s / GPU
Faster R-CNN [21]	81.84 %	86.71 %	71.12 %	2 s / GPU

(a) Valores de la precisión media en la clase coche de KITTI.

Método	Moderado	Fácil	Difícil	Tiempo de ejecución
MS-CNN [25]	**73.70** %	**83.92** %	**68.31** %	0.4 s / GPU
SubCNN [26]	71.33 %	83.28 %	66.36 %	2 s / GPU
SDP+RPN [23]	70.16 %	80.09 %	64.82 %	0.4 s / GPU
3DOP [24]	67.47 %	81.78 %	64.70 %	3 s / GPU
Mono3D [22]	66.68 %	80.35 %	63.44 %	4.2 s / GPU
Faster R-CNN [21]	65.90 %	78.86 %	61.18 %	2 s / GPU
SDP+CRC [23]	64.19 %	77.74 %	59.27 %	0.6 s / GPU

(b) Valores de la precisión media en la clase peatón de KITTI.

Método	Moderado	Fácil	Difícil	Tiempo de ejecución
MS-CNN [25]	**75.46** %	**84.06** %	**66.07** %	0.4 s / GPU
SDP+RPN [23]	73.74 %	81.37 %	65.31 %	0.4 s / GPU
SubCNN [26]	71.06 %	79.48 %	62.68 %	2 s / GPU
3DOP [24]	68.94 %	78.39 %	61.37 %	3 s / GPU
Mono3D [22]	66.36 %	76.04 %	58.87 %	4.2 s / GPU
Faster R-CNN [21]	63.35 %	72.26 %	55.90 %	2 s / GPU
SDP+CRC [23]	61.31 %	74.08 %	53.97 %	0.6 s / GPU

(c) Valores de la precisión media en la clase ciclista de KITTI.

Tabla 2.2: Precisión media en detección de objetos de varios métodos del estado del arte en tres categorías de la base de datos KITTI [1].

Capítulo 3

Redes Neuronales Artificiales para detección de objetos en imágenes

Como se ha contado en el capítulo introductorio de esta memoria, las redes neuronales artificiales se han convertido en una de las alternativas más prometedoras en la tarea de desarrollar sistemas de detección de objetos en imágenes. Para poder entender su funcionamiento, en este capítulo se presenta una evolución desde los conceptos teóricos más básicos hasta las redes neuronales para detección de objetos. Además, se explicarán los algoritmos y diferentes métodos que se aplican en el apartado experimental de la memoria.

Las redes neuronales artificiales son un paradigma de programación inspirado en el funcionamiento de las neuronas y conexiones del cerebro que sirve para procesar información. Con una cierta estructura, estas son capaces de computar cualquier función real ($f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$)[29].

Los modelos matemáticos de las redes neuronales biológicas fueron introducidos por primera vez en 1943 por Warren S. McCulloch y Walter Pitts [30]. Posteriormente, la primera y más básica de las neuronas artificiales fue el perceptrón [31] que conceptualmente se trata de un algoritmo de clasificación binario en el que una función lineal produce una salida binaria (0 o 1) en función del producto de unos pesos por los valores de entrada. Estos fueron los puntos de partida que dieron paso a un posterior desarrollo de algoritmos relacionados con las redes neuronales, aunque los nuevos retos de la tecnología y la mejora en la capacidad de computación en la última década han provocado un avance más acelerado en esta materia.

En muchos de los problemas complejos de reconocimiento de patrones con redes neuronales, la cantidad de parámetros que se han de ajustar es muy grande, por lo que el tiempo de computación era demasiado elevado en la mayoría de casos en el pasado. La mejora de las computadoras y el uso de las tarjetas gráficas para hacer cálculos en paralelo han permitido superar esa barrera en tiempo de computación, haciendo que las redes neuronales sirvan de gran utilidad para muchas tareas.

3.1. La neurona

Las neuronas artificiales son los elementos que se encargan de llevar a cabo el cálculo de funciones matemáticas dentro de una red. En el caso más básico de un perceptrón, una función lineal combina unos pesos dados con los datos de entrada y a continuación aplica la función escalón unitario (o de Heaviside) para obtener una salida binaria (0—1).

La notación matemática en este trabajo será la siguiente: los vectores se escribirán en negrita y en minúscula ($\boldsymbol{x}$), sus valores entre corchetes ($[x_1, \ldots, x_N]$). Los escalares se escribirán en minúscula (c) y las matrices se escribirán en mayúscula y en negrita ($\boldsymbol{M}$).

La evolución y generalización del perceptrón es la neurona artificial que permite obtener como salida valores no solamente binarios. Las neuronas constan de los siguientes componentes [32]:

- Variables de entrada, debe constar al menos de una. Se representan comúnmente en forma de vector de la siguiente manera, $\boldsymbol{x} = [x_1, x_2, x_3, \ldots, x_N]$.
- Un peso por variable de entrada que determinará la importancia de cada conexión. El vector de pesos se representará utilizando la notación $\boldsymbol{w} = [w_1, w_2, w_3, \ldots, w_N]$.
- Un parámetro llamado sesgo, b.

  El sesgo indicará la facilidad que tiene la neurona de activarse. Este proceso está basado en las neuronas biológicas, las cuales envían un impulso eléctrico solo si a su entrada (constan de un único axón) hay un potencial eléctrico mayor a un potencial umbral. El sesgo sería la inversa a ese potencial umbral. En el ámbito de reconocer patrones, las salidas de las neuronas indican ciertas características de la muestra con la que se trabaja, en este caso los pesos determinan cómo de relacionada está la característica con cada entrada, y el sesgo cómo de fácil de detectar es esa característica.
- Una función de propagación que actúa sobre los pesos y el sesgo antes de acceder a la función de activación, $y = f(x_1, \ldots, x_N, w_1, \ldots, w_N, b)$. En el caso del perceptrón, la función de propagación es lineal y tiene la siguiente forma matemática:

$$y = \sum_{i=1}^{N} w_i x_i + b \tag{3.1}$$

  Aunque esta es la función de propagación más comúnmente utilizada, de manera general las funciones implementadas en cada neurona pueden ser no lineales.
- Una función de activación que determinará si se activa la salida de la neurona en función del valor resultante de la función de propagación, $z = f_{act}(y)$. Entre las funciones de activación más comúnmente utilizadas se encuentran la tangente hiperbólica, la cual ofrece una salida acotada entre -1 y 1, la función sigmoide, una versión escalada de la tangente hiperbólica que ofrece una salida acotada entre 0 y 1 o la unidad lineal rectificadora (ReLU), la cual lleva a cabo la siguiente operación: $f(y) = max(0, y)$.

En la Figura 3.1 se muestra un esquema de la estructura interna de una neurona.

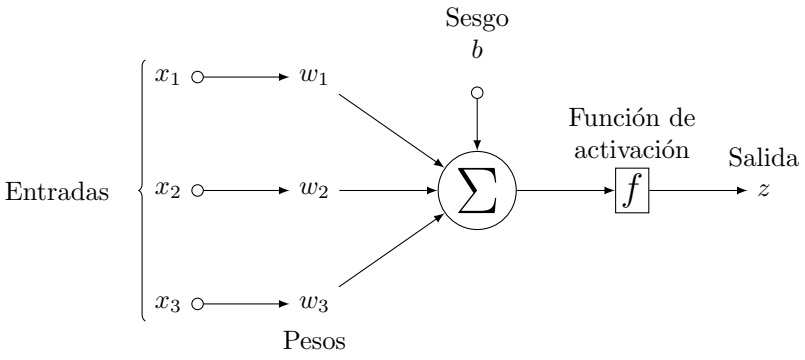


Figura 3.1: Representación gráfica de una neurona artificial.

Por lo tanto, la salida de una neurona cuya entrada es un vector $\boldsymbol{x} = [x_1, \ldots, x_N]$ tendrá esta forma:

$$z = f_{act}\left(\sum_{i=1}^{N} w_i x_i + b\right) \tag{3.2}$$

3.2. Estructuras de las redes neuronales

Para construir una red neuronal capaz de resolver problemas complejos es necesario interconectar un cierto número de neuronas. Las neuronas pueden agruparse con diferentes estructuras, de esta

manera, se transforma el vector $\boldsymbol{x}$ de entrada aplicando las operaciones pertinentes en cada neurona, y consiguiendo así un vector de salida.

Entre las estructuras más comunes para implementar redes destacan las redes recurrentes y sobre todo las más utilizadas, las redes *feedforward*. En la detección de objetos, para formar un tipo específico de redes conocidas como convolucionales (Sección 3.6) es habitual utilizar estructuras de redes con conexiones residuales.

3.2.1. Redes *feedforward*

En este tipo de estructura las neuronas están agrupadas en varias capas: una capa de entrada, n capas ocultas (cuyas salidas son invisibles desde el exterior y cuyas neuronas también se denominan ocultas) y una capa de salida.

En una red *feedforward* cada neurona de una capa solo tiene conexión directa con las neuronas de la siguiente capa, en dirección hacia la capa de salida [32]. La capa de entrada en realidad no es una capa compuesta de neuronas (esto ocurre en todas las estructuras, no solo en la *feedforward*), sino una herramienta visual que se utiliza para representar las entradas a la primera capa oculta, la cual es la primera que consta de neuronas que se encargan de realizar cálculos para el cometido que se desea desempeñar. En la Figura 3.2 se puede observar la disposición de capas y neuronas en este tipo de red.

El número de neuronas de entrada será igual al número de variables del conjunto de datos, y el número de neuronas de salida igual al número de símbolos necesarios para representar la salida deseada. En el caso de que se trate de un clasificador, el número de salidas será igual al número de clases existentes.

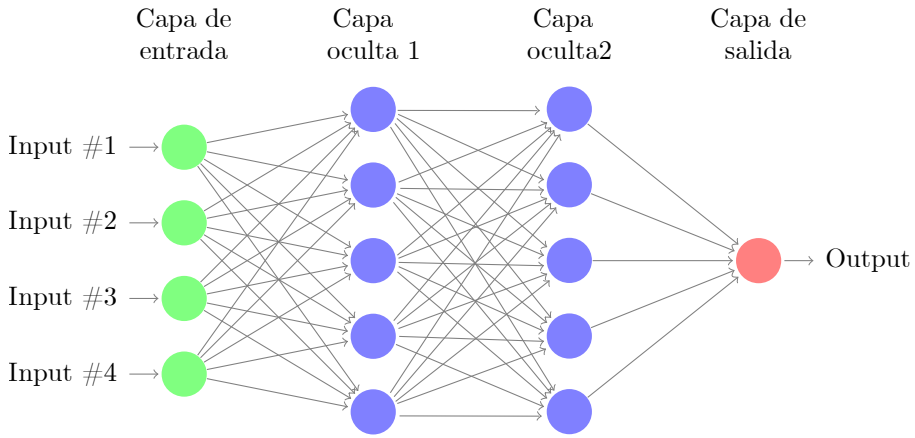


Figura 3.2: Representación de una red neuronal de estructura feedforward con dos capas ocultas.

La salida de una red *feedforward* es sencilla de calcular conociendo los pesos, sesgos y funciones de activación de la red. Teniendo en cuenta que la entrada de una capa l es la salida de la capa $l-1$ (Figura 3.2), la salida de la capa l dependerá únicamente de la salida de la capa $l-1$. Empezando desde la primera capa, se calcula su salida en función de las entradas y los parámetros (pesos, sesgos y funciones de activación) de las neuronas de esta capa. Después, con esa salida, se calcula la salida de la segunda capa de la misma manera, cogiendo como entrada la salida de la capa 1, y así sucesivamente hasta obtener el vector de salida.

La salida de una neurona i de la capa l se escribe utilizando la siguiente expresión:

$$z_i^l = f_{act}\left(\sum_{j=1}^{J} z_j^{l-1} w_{ij}^l + b_i^l\right) \tag{3.3}$$

Donde z_j^{l-1} es la salida de una neurona j de la capa $l-1$ en la que hay un total de J neuronas,

w_{ij}^l es el peso que conecta la neurona i de la capa l con la neurona j de la capa $l-1$ y el sesgo b_i^l es el sesgo de la la neurona i de la capa l. El vector de salida $\boldsymbol{z}^l$ no es más que el vector que conforman todos los posibles z_i^l de la capa l.

Las redes que utilizan un número elevado de capas ocultas se denominan redes profundas. En función del problema que se desea abordar puede ser más o menos adecuado hacer uso de redes más profundas. El fenómeno de la profundidad se comentará más en detalle en el Apartado 3.4.

3.2.2. Redes con conexiones residuales

Una red residual es una red en la cual existen conexiones directas entre capas no consecutivas, siempre que la conexión siga la relación capa anterior → capa posterior (Figura 3.3), es decir, una conexión en la cual la salida de una red se "salta" capas [33]. En el caso de que la conexión fuera en sentido contrario se trataría de una red recurrente.

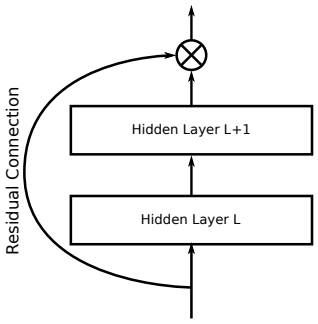


Figura 3.3: Ejemplo de una conexión residual [33].

Este tipo de conexiones ofrecen una serie de ventajas a medida que la profundidad aumenta. En el ajuste de los parámetros de las redes muy profundas aparece con frecuencia un fenómeno conocido como desvanecimiento del gradiente (Sección 3.3.5). En el apartado citado se analizará este fenómeno y se expondrán las ventajas que presentan las redes con conexiones residuales para hacer frente a dicho problema.

3.3. Algoritmos de entrenamiento

Los diferentes parámetros que existen en una red neuronal han de ser ajustados para aproximar con mayor precisión la función que se desea modelar. Para dicho cometido se utilizan los algoritmos de entrenamiento. Este apartado se centrará exclusivamente en tres técnicas, la optimización por descenso de gradiente, la optimización por descenso de gradiente estocástico y la propagación hacia atrás. Estas técnicas son las más comúnmente utilizadas para entrenar redes neuronales, y son las que se han utilizado en la parte experimental del trabajo.

El entrenamiento puede entenderse como un problema de optimización, por ello es necesario definir la función que se desee optimizar. Variando los valores de los parámetros de la red (pesos y sesgos) se tratará de encontrar un óptimo de esta función siguiendo una determinada estrategia.

3.3.1. Función de coste

La función de coste, también conocida como función objetivo o función de pérdidas, es una función que mide cómo de bien se ajustan los parámetros de una red al conjunto de los ejemplos de entrenamiento [29]. Normalmente las funciones de coste se calculan haciendo una media de los

costes de todos los ejemplos de entrenamiento como se muestra en la siguiente ecuación:

$$C(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{N} \sum_{i=1}^{N} C_i(\boldsymbol{W}, \boldsymbol{b}) \tag{3.4}$$

Donde $\boldsymbol{W}$ y $\boldsymbol{b}$ hacen referencia a los conjuntos totales de pesos y sesgos de la red respectivamente. C_i representa la función de coste para la instancia i del conjunto de entrenamiento.

Es de vital importancia seleccionar una función de coste adecuada para el problema que se quiere abordar, ya que el ajuste de los parámetros de la red se efectuará con miras a minimizar esta función.

Una de las funciones de coste más comunes y sencillas es la función de coste cuadrático, la cual sigue la siguiente expresión:

$$C(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{N} \sum_{i=1}^{N} \frac{||\boldsymbol{t}_i - \boldsymbol{o}_i||^2}{2} \tag{3.5}$$

Siendo $\boldsymbol{t}_i$ el valor de salida deseado asociado a una entrada i, comúnmente conocido como *ground-truth*, y $\boldsymbol{o}_i$ la salida de la red para una entrada i, la cual por supuesto depende de $\boldsymbol{W}$ y $\boldsymbol{b}$. Es una función que siempre devuelve valores positivos, la cual obtiene sus valores más bajos cuanto menor sea la diferencia entre la predicción de la red con el valor deseado.

Puesto que las funciones de coste en redes neuronales dependen de muchísimas variables no es viable calcular los mínimos analíticamente, por ello se utilizan algoritmos de optimización como por ejemplo el descenso de gradiente.

3.3.2. Descenso de gradiente

El descenso de gradiente permite encontrar mínimos locales de una función. El procedimiento para implementarlo se explica mediante los siguientes pasos [29]:

1. Partiendo de una inicialización, por ejemplo aleatoria, de la función que se desea minimizar, la cual consta de un número N de variables, $C(\boldsymbol{v}) = C(v_1, v_2, \ldots, v_N)$, el gradiente en el punto $\boldsymbol{v}_0$ se calcula de la siguiente manera:

$$\nabla C(\boldsymbol{v}_0) = \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}, \ldots, \frac{\partial C}{\partial v_N} \right) \Bigg|_{\boldsymbol{v}=\boldsymbol{v}_0} \tag{3.6}$$

2. Para avanzar en busca del óptimo, el siguiente punto al que moverse en el espacio de variables se calcula mediante la siguiente ecuación:

$$\boldsymbol{v}_{t+1} = \boldsymbol{v}_t - \eta \nabla C(\boldsymbol{v}_t) \tag{3.7}$$

   El símbolo η representa una constante positiva denominada velocidad de aprendizaje o *learning rate*[1]. Con un valor de η suficientemente pequeño, puede demostrarse que una vez actualizada la variable siguiendo la ecuación 3.7, el nuevo valor será menor al anterior:

$$\Delta C \approx \nabla C \cdot \Delta v = \nabla C \cdot (-\eta \nabla C) = -\eta (\nabla C)^2 \leq 0 \tag{3.8}$$

   Este proceso se puede repetir iterativamente hasta alcanzar un criterio de parada, como por ejemplo un número máximo de iteraciones. Cada iteración del algoritmo hasta finalizar se

[1] No siempre es un valor constante, de hecho en variantes basadas en este algoritmo como por ejemplo el ADAM [34] se utilizan valores adaptativos del *learning-rate*.

conoce como época de aprendizaje. Las expresiones correspondientes a la actualización de los pesos y sesgos de la red son las siguientes:

$$w_{k_{t+1}} = w_{k_t} - \eta \frac{\partial C}{\partial w_{k_t}} \tag{3.9}$$

$$b_{k_{t+1}} = b_{k_t} - \eta \frac{\partial C}{\partial b_{k_t}} \tag{3.10}$$

Aunque con un número de épocas y una velocidad de aprendizaje adecuados este algoritmo es capaz de converger en un óptimo, presenta los siguiente problemas principales:

1. Solamente permite encontrar mínimos locales (Figura 3.4), por lo que dependiendo de la inicialización de las variables el algoritmo podría converger a óptimos que se alejan mucho del óptimo global.
2. Calcular el gradiente de la función de coste puede resultar muy costoso. En la Ecuación 3.5 se muestra como la función de coste se calcula haciendo la media de los costes de N ejemplos de entrenamiento. Gracias a la linealidad del gradiente, es posible calcular el gradiente de la función de coste haciendo una media de los gradientes de los costes de cada ejemplo de entrenamiento. En el caso de que el número de instancias sea muy alto, y en el caso particular de la detección de objetos en imágenes suele ser muy extenso, el tiempo que toma la red para llevar a cabo una actualización de los parámetros suele ser demasiado largo.

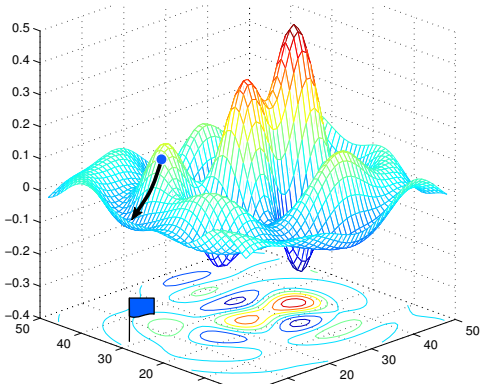


Figura 3.4: Ejemplo de los posibles mínimos alcanzados por el descenso de gradiente en una función de dos variables [35].

Debido a estos problemas, con frecuencia se suele optar por una versión modificada del descenso de gradiente, el descenso de gradiente estocástico.

3.3.3. Descenso de gradiente estocástico

Esta variante del descenso de gradiente puede explicarse mediante los siguientes pasos [29, 36]:

1. Se divide el conjunto de datos de entrenamiento en M subconjuntos iguales elegidos aleatoriamente, estos subconjuntos se denominan *mini-batches*.
2. Por cada *mini-batch* se aproxima el valor de la función de coste utilizando la siguiente expresión:

$$C(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{N} \sum_{i=1}^{N} C_i(\boldsymbol{W}, \boldsymbol{b}) \approx \frac{1}{M} \sum_{j=1}^{M} C_j(\boldsymbol{W}, \boldsymbol{b}) \tag{3.11}$$

   Donde C_j es el coste asociado a cada ejemplo dentro de un *mini-batch*.

3. Se calcula el gradiente para la aproximación que se muestra en la Ecuación 3.11, y se actualizan los parámetros de esta manera, para cada *mini-batch*:

$$w_{k_{t+1}} = w_{k_t} - \eta \frac{1}{M} \sum_{j=1}^{M} \frac{\partial C_j}{\partial w_{k_t}} \approx w_{k_t} - \eta \frac{\partial C}{\partial w_{k_t}} \tag{3.12}$$

$$b_{k_{t+1}} = b_{k_t} - \eta \frac{1}{M} \sum_{j=1}^{M} \frac{\partial C_j}{\partial b_{k_t}} \approx b_{k_t} - \eta \frac{\partial C}{\partial b_{k_t}} \tag{3.13}$$

De esta manera, por el hecho de actualizar los valores de los parámetros con mayor frecuencia el aprendizaje se acelera considerablemente. Por otra parte, calcular una aproximación del gradiente y ajustar los parámetros teniendo en cuenta subconjuntos aleatoriamente escogidos hace que el algoritmo no tienda a converger hacia el mismo mínimo local constantemente, de forma que es más probable acabar en un mínimo local de menor valor o en el mínimo global de la función (Figura 3.5).

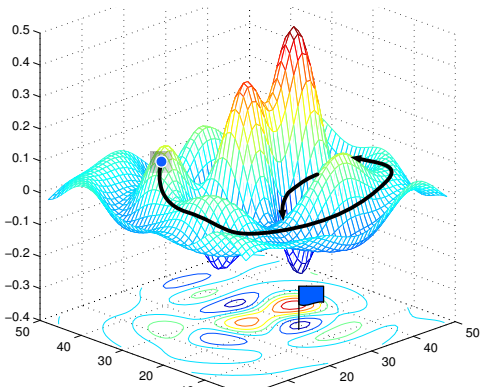


Figura 3.5: Ejemplo del posibles mínimo alcanzados por el descenso de gradiente estocástico en una función de una dos variables [35].

En la parte experimental de este trabajo se ha utilizado una variante de este algoritmo, llamada descenso de gradiente estocástico con reinicios [37]. El concepto más importante de esta variante es que la velocidad de aprendizaje va disminuyendo poco a poco a lo largo de las iteraciones, y vuelve al valor inicial periódicamente, de esta manera se evita mejor la tendencia de caer en mínimos locales.

Además la velocidad de aprendizaje que se utiliza de base, es decir, el valor al que el algoritmo retorna después de un número de iteraciones, toma diferentes valores. Toma valores cada vez más pequeños dependiendo en la época en la que esté, y estos valores se definen manualmente.

3.3.4. Propagación hacia atrás

El algoritmo de propagación hacia atrás, conocido comúnmente como *back-propagation*, se introdujo originalmente en la década de 1970, pero no cogió importancia hasta que en 1986 David Rumelhart, Geoffrey Hinton y Ronald Williams demostraran en [38] que funcionaba mucho más rápido que los algoritmos utilizados anteriormente para el aprendizaje en redes neuronales de varias capas, lo que supuso la posibilidad de resolver problemas que antes eran irresolubles en materia de redes neuronales.

El objetivo de la propagación hacia atrás es calcular las derivadas parciales $\frac{\partial C}{\partial w}$ y $\frac{\partial C}{\partial b}$ que aparecen en las Ecuaciones 3.12 y 3.13. Para ello, se comenzará por definir la función de error δ de cada neurona de la red, el cual representa cuánto varía la función de coste correspondiente a un ejemplo

de entrenamiento al cambiar la entrada a su función de activación [29]:

$$\delta_j^l \equiv \frac{\partial C_i}{\partial y_j^l} \tag{3.14}$$

Donde δ_j^l es el error correspondiente a la neurona j de la capa l, C_i es la función de coste asociada a un ejemplo de entrenamiento i (que depende de los pesos y los sesgos de la red) e y_j^l es la salida de la función de propagación de la neurona j de la capa l. Por lo tanto y_j^l es también la entrada a la función de activación de la neurona j de la capa l. Por lo tanto, siendo $z_j^l = f_{act}(y_j^l)$ es posible reescribir la expresión 3.14 de la siguiente manera:

$$\delta_j^l \equiv \frac{\partial C_i}{\partial y_j^l} = \frac{\partial C_i}{\partial z_j^l}\frac{\partial z_j^l}{\partial y_j^l} = \frac{\partial C_i}{\partial z_j^l} f'_{act}(y_j^l) \tag{3.15}$$

Para continuar con la explicación del algoritmo, es conveniente utilizar una notación matricial que englobe el error todas las neuronas de una capa:

$$\boldsymbol{\delta}^l = \nabla_z C_i \odot f'_{act}(\boldsymbol{y}^l) \tag{3.16}$$

De manera que $\nabla_z C_i$ representa el conjunto de derivadas parciales de la función de coste respecto a la salida de todas las neuronas de una capa, $f'_{act}(\boldsymbol{y}^l)$ es un vector que contiene el valor de $f'_{act}(y_j^l)$ para cada neurona j de la capa y $\odot$ representa el producto de Hadamard, el cual calcula el producto elemento a elemento.

El algoritmo de propagación hacia atrás se basa en propagar hacia las neuronas de las capas iniciales el error de las capas más cercanas a la salida de la red. Por lo tanto el primer paso es calcular el error en la capa de salida. Suponiendo que $l \in [1, L]$ y tomando como ejemplo el coste cuadrático (Ecuación 3.5), para obtener el error de las neuronas de la capa de salida, primero debe calcularse el valor de $\nabla_z C_i$, agrupando las derivadas parciales de cada neurona de la capa, las cuales se calculan así:

$$C_i = \frac{||\boldsymbol{t}_i - \boldsymbol{z}_i||^2}{2} \Rightarrow \frac{\partial C_i}{\partial z_j^L} = z_j^L - t_{ij} \tag{3.17}$$

donde t_{ij} es la salida deseada de la red. Por otro lado, para calcular el valor de $\boldsymbol{\delta}^L$, como la función de activación es conocida, también lo es su derivada y es posible de esa manera calcular cada componente del vector $f'_{act}(\boldsymbol{y}^l)$.

La utilidad de la propagación hacia atrás reside en que una vez se ha calculado el error en la capa de salida, puede calcularse el error en el resto de las neuronas del resto de capas utilizando la información de los errores de la siguiente capa. La siguiente ecuación muestra cómo calcular el vector de errores de la capa l en función de los errores de la capa $l+1$.

$$\boldsymbol{\delta}^l = \nabla_z C_i \odot f'_{act}(\boldsymbol{y}^l) = (\boldsymbol{W}^{l+1}\boldsymbol{\delta}^{l+1}) \odot f'_{act}(\boldsymbol{y}^l) \tag{3.18}$$

siendo $\boldsymbol{W}^{l+1}$ el vector de pesos de la capa $l+1$.

Para demostrar la igualdad $\nabla_z C_i = \boldsymbol{W}^{l+1}\boldsymbol{\delta}^{l+1}$, es necesario tratar varios factores. Lo primero es representar el error de una neurona de la capa l en función de los errores de las neuronas de la capa $l+1$. Teniendo en cuenta que existe dependencia entre todos los $y_k^{l+1}, k \in [1, K]$ e y_j^l, aplicando la regla de la cadena puede reescribirse la Ecuación 3.15 de la siguiente manera:

$$\delta_j^l \equiv \frac{\partial C_i}{\partial y_j^l} = \sum_k \frac{\partial C_i}{\partial y_k^{l+1}}\frac{\partial y_k^{l+1}}{\partial y_j^l} = \sum_k \frac{\partial y_k^{l+1}}{\partial y_j^l}\delta_k^{l+1} \tag{3.19}$$

Además, para calcular el componente $\frac{\partial y_k^{l+1}}{\partial y_j^l}$, es necesario representar y_k^{l+1} en función de y_j^l. Rememorando la Ecuación 3.1, se puede escribir la siguiente expresión:

$$y_k^{l+1} = \sum_j w_{kj}^{l+1} z_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} f_{act}(y_j^l) + b_k^{l+1} \tag{3.20}$$

siendo w_{kj}^{l+1} el peso que conecta la neurona j de la capa l y la neurona k de la capa $l+1$. Por lo tanto:

$$\frac{\partial y_k^{l+1}}{\partial y_j^l} = w_{kj}^{l+1} f'_{act}(y_j^l) \tag{3.21}$$

Y por último, la expresión de δ_j^l puede escribirse de esta forma:

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} f'_{act}(y_j^l) \tag{3.22}$$

Lo cual representa lo mismo que la ecuación 3.18, pero escrito en forma de componentes.

Por último, ya que pueden obtenerse los valores de los errores para cada neurona, es posible calcular los valores de las derivadas parciales necesarias para las actualizaciones de los pesos y los sesgos que aparecían en las ecuaciones 3.13 y 3.12.

En el caso de los sesgos, teniendo en cuenta que $y_j^l = \sum_k w_{jk}^l z_k^{l-1} + b_j^l$ se obtiene:

$$\frac{\partial C_i}{\partial b_j^l} = \frac{\partial C_i}{\partial y_j^l} \frac{\partial y_j^l}{\partial b_j^l} = \delta_j^l \tag{3.23}$$

Y es similar en el caso de los pesos:

$$\frac{\partial C_i}{\partial w_{jk}^l} = \frac{\partial C_i}{\partial y_j^l} \frac{\partial y_j^l}{\partial w_{jk}^l} = z_k^{l-1} \delta_j^l \tag{3.24}$$

Puesto que para la propagación del error hacia atrás es imprescindible calcular la derivada de la función de activación, para poder aplicar la optimización por descenso de gradiente o descenso de gradiente estocástico con propagación hacia atrás, es necesario que las funciones de activación sean derivables. En los casos en los que la función de activación no tiene derivada en un número de puntos finito, como por ejemplo la función ReLU, cuya derivada no existe en el punto $x = 0$, se asignan valores arbitrarios para esos puntos concretos, para que en el caso remoto en el que la función de activación reciba como entrada uno de estos puntos, el algoritmo no falle.

El descenso de gradiente estocástico con la propagación hacia atrás, aunque con ligeras variaciones en algunos casos, sigue siendo el procedimiento más utilizado para ajustar los parámetros de las redes neuronales. Estos métodos, aunque han demostrado un gran rendimiento, no están libres de problemas. Uno de los problemas más comunes cuando se entrenan redes profundas es el problema del desvanecimiento del gradiente.

3.3.5. Desvanecimiento del gradiente

Para el entrenamiento de redes neuronales, habitualmente el valor de los pesos y sesgos se inicializa entre 0 y 1 para evitar que estos acaben tomando valores muy altos (explosión del gradiente [29]). Cuando se trata de funciones de activación que saturan como la función sigmoide, el problema se explica mediante los siguientes pasos:

1. Como se puede observar en la Ecuación 3.22, el error en una neurona de la capa l es proporcional a la derivada de la función de activación de las neuronas de esta capa, y además también es proporcional al error de la capa $l+1$, el cual a su vez es proporcional a la derivada de la función de activación de las neuronas de la capa $l+1$ y así sucesivamente hasta la capa de salida.

2. La derivada de la función sigmoide está acotada entre 0 y 0,25. A consecuencia de esto, en redes profundas, el error de las neuronas de las primeras se ha multiplicado en repetidas ocasiones por un factor entre 0 y 0,25 por lo que el valor resultante es un número muy pequeño.

3. Debido a este problema, las primeras capas de una red neuronal son las más lentas y difíciles de entrenar ya que el valor del gradiente que se usa para actualizarlas en cada iteración del entrenamiento es muy pequeño.

4. Además, esto causa otro problema: si las primeras capas no están bien entrenadas, el problema se arrastra a las capas posteriores, lo cual hace que el entrenamiento de este tipo de redes se complique considerablemente.

Este problema hace que en redes muy profundas como las redes convolucionales para procesado de imágenes (Apartado 3.8) se utilice con frecuencia la función de activación ReLU, cuya derivada se define con la siguiente expresión[2]:

$$ReLu'(x) = \theta(x) = \begin{cases} 0 & para \quad x < 0 \\ \\ 1 & para \quad x \geq 0 \end{cases} \tag{3.25}$$

Por lo tanto, al tener valores de la derivada que llegan hasta 1, esta función de activación no contribuye a que el valor del gradiente vaya menguando sistemáticamente.

Otra solución al problema del desvanecimiento del gradiente consiste en utilizar una arquitectura de conexiones residuales (Apartado 3.2.2). Este tipo de conexión conlleva que cuando el error se propaga hacia atrás en una red, y existe un salto de varias capas (Figura 3.3), se "inyecta" el error de capas más alejadas de la entrada a capas más cercanas, y así se consigue que el gradiente se mantenga suficientemente grande como para poder seguir propagándolo hasta la entrada.

3.4. Teorema de aproximación universal y análisis de profundidad

Como se ha comentado anteriormente, las redes neuronales, con una estructura dada son capaces de representar cualquier función. El teorema de aproximación universal [39] demuestra que una red *feedforward* con una salida lineal y al menos una capa oculta con una determinada función de activación no lineal puede representar cualquier función continua y acotada en el espacio $\mathbb{R}^n$, siempre que exista un número suficiente de neuronas en la capa oculta.

Aunque en el teorema original se demostró para unidades con funciones de activación que saturan para argumentos muy negativos y muy positivos, posteriormente también se ha demostrado para un numero más amplio de funciones de activación, entre las que se encuentra por ejemplo la función ReLU [40].

El teorema de aproximación universal demuestra que con una configuración concreta la función deseada puede representarse, aún así no garantiza que el algoritmo de entrenamiento sea capaz de aprender dicha función. El algoritmo podría fallar por varias razones: Primero, el algoritmo de optimización utilizado puede no tener la capacidad de encontrar los valores adecuados para los parámetros que se han de ajustar. Segundo, el algoritmo podría elegir los parámetros equivocados como consecuencia de sobreajustarse a la muestra de entrenamiento.

[2]La derivada de la función ReLU no existe para el punto $x = 0$, en este ejemplo se le asignará el valor 1.

Por otro lado, en el caso particular de una red con una sola capa oculta, aunque con un número suficiente de neuronas en esta capa la red pueda aprender cualquier función, ese número podría aumentar considerablemente en muchos de los casos. En [41] se estiman los límites del tamaño de una red para aproximar un amplio número de funciones. En el peor de los casos el número de neuronas crece exponencialmente con el tamaño de la entrada.

En resumen, una red de *feedforward* con una sola capa es suficiente para representar cualquier función, pero la capa puede ser demasiado grande y no aprender a generalizar correctamente. En muchas circunstancias, el uso de modelos más profundos puede reducir el número de unidades requeridas para representar la función deseada y puede reducir la cantidad de error de generalización [42].

3.5. Medidas para prevenir el sobreajuste

Uno de los problemas más habituales en las redes neuronales profundas es el sobreajuste, fenómeno al que a menudo suele hacerse referencia por su nombre en inglés, *overfitting*. Esto ocurre cuando el modelo se ajusta excesivamente a los datos de entrenamiento, aprendiendo características demasiado específicas de estos, incluyendo el ruido que puede haber en ellos, y, como consecuencia, falla al generalizar el comportamiento que se desea aprender para nuevos casos.

Existen muchas medidas para tratar de prevenir este fenómeno. Lo más habitual es utilizar un subconjunto de los datos de entrenamiento para validar el modelo. Este subconjunto es invisible al ajuste de los parámetros de la red, por lo cual la red modificará los valores de los pesos y sesgos con el objetivo de minimizar el error que obtenga sobre el conjunto de entrenamiento, pero se podrá observar el error sobre los datos de validación. Un claro indicativo de que el modelo se está ajustando demasiado a los datos de entrenamiento aparece en el momento en el que el error sobre el conjunto de entrenamiento sigue disminuyendo a lo largo de las épocas o iteraciones del entrenamiento, pero el error sobre los datos de validación comienza a incrementarse considerablemente.

En el ámbito del aprendizaje automático es habitual utilizar la validación cruzada, la cual consiste en partir los datos en k partes iguales y generar k modelos, utilizando siempre una de las k particiones para validar y el resto para entrenar, cambiando la distribución en cada iteración. Aunque es una técnica interesante ya que proporciona una información más fiable del rendimiento del modelo, en el entrenamiento de redes profundas en la mayoría de casos no resulta práctico. En función del problema que se esté afrontando, el entrenamiento de un modelo puede llevar semanas, y entrenar más de un modelo de manera simultánea requeriría utilizar muchas unidades de procesamiento gráfico. Además, en la mayoría de casos en problemas relacionados con redes profundas, la cantidad de datos es suficientemente grande como para que una partición fija sirva para estimar el rendimiento del modelo con bastante fiabilidad.

Otra técnica interesante, la cual puede combinarse con la anterior para prevenir aún más el problema es la regularización *dropout*.

3.5.1. Regularización *dropout*

La regularización *dropout* es un método para la prevención del sobreajuste en redes profundas que consiste en suprimir aleatoriamente neuronas (junto con sus conexiones) de la red neuronal durante el entrenamiento (Figura 3.6). Esto significa que su contribución a la activación de las neuronas se anula temporalmente durante la propagación hacia adelante y las actualizaciones de los parámetros no se aplican a la neurona en la propagación hacia atrás.

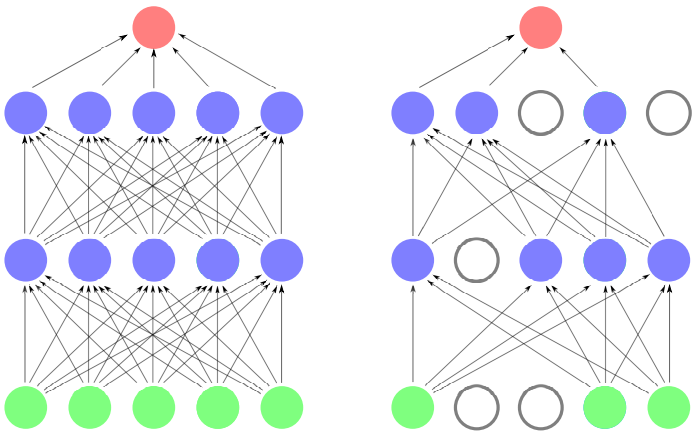


Figura 3.6: Red original (izquierda) y red a la que se le ha aplicado el *dropout* (derecha) [43].

El aprendizaje estándar de propagación hacia atrás genera relaciones de dependencia entre neuronas que funcionan para los datos de entrenamiento, pero perjudican la generalización a casos previamente no observados. El *dropout* aleatorio rompe estas adaptaciones conjuntas al hacer que la presencia de una unidad oculta particular no sea determinante. Se ha probado que esta técnica mejora el rendimiento de las redes neuronales en una amplia variedad de aplicaciones, que incluyen la clasificación de objetos, el reconocimiento de dígitos, el reconocimiento de voz, la clasificación de documentos y el análisis de datos de biología computacional [43].

3.6. Redes neuronales convolucionales

Después de definir los conceptos teóricos básicos de las redes neuronales, este apartado se centra en un tipo de redes específicas que destacan en aplicaciones como las series temporales y sobre todo en procesar imágenes. Estas redes, llamadas redes convolucionales [44] son un tipo especializado de red neuronal para procesar datos ordenados en forma de cuadrícula. Las redes convolucionales toman este nombre dado que realizan una operación matemática lineal llamada convolución [42].

La mayoría de redes convolucionales que se utilizan para procesar imágenes realizan dos operaciones principales, la primera es la anteriormente comentada convolución y la segunda es una operación llamada *pooling*. Aunque en los siguientes apartados se explicarán estas operaciones con mayor detalle, en términos generales, la convolución aplica diferentes filtros a la imagen, y obtiene como salida una imagen transformada y de dimensión reducida para cada filtro, estas imágenes transformadas se conocen comúnmente como mapas de características. Después, se le aplica la función de activación, entre las cuales destaca la ReLU. Por último, se vuelven a reducir todos los mapas de características extrayendo los valores más significativos con la operación *pooling*. Este proceso puede repetirse en bucle repetidas veces en una red convolucional.

Una vez extraídas las características deseadas después de aplicar un número de convoluciones y *pooling* adecuado, la salida de la última capa de *pooling* se conecta con una red neuronal convencional para realizar una clasificación de la imagen. En la Figura 3.7 se muestra un ejemplo visual de una red convolucional básica en la que se aplican tres filtros en cada convolución.

Desde la victoria de AlexNet (5 capas convolucionales) [2] en el concurso de clasificación ILSVRC 2012, el desarrollo de las redes que se utilizan hoy en día en el procesado de imágenes, como extractores de características de la imagen, ha llevado a redes mucho más profundas como la Resnet [33] (hasta 152 capas convolucionales) o la Inception v2 [45] (105 capas convolucionales).

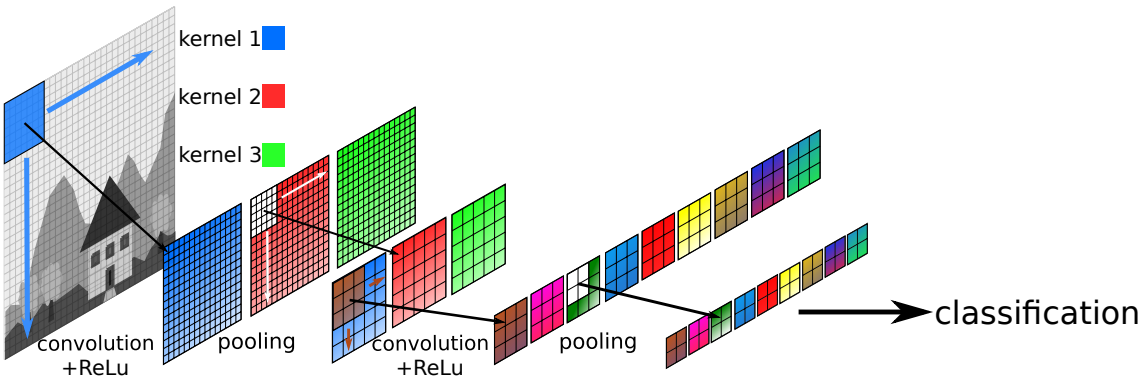


Figura 3.7: Esquema de una red convolucional con dos capas de convolución y *pooling*.

3.6.1. Convolución

La convolución es una operación de filtrado lineal básica en la visión computacional, la cual utilizando diferentes filtros o kernels se aplica para cometidos como el desenfoque, el aumento de nitidez, acentuar el relieve o la detección de bordes en una imagen [46].

Cuando se habla de convolución en términos de aprendizaje automático, visión por computador o redes neuronales, hay tres conceptos principales a tener en cuenta: la entrada, habitualmente una imagen la cual se representará como I, el kernel, el cual se representa con la letra K y la salida, el mapa de características. La convolución de un kernel K de tamaño $m \times n$ colocado en el píxel (i, j) de una imagen I se representa mediante la siguiente expresión [42]:

$$(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n) \tag{3.26}$$

Las capas convolucionales de las redes normalmente aplican varias convoluciones en paralelo en diferentes puntos (i, j), esto puede entenderse mejor visualizando un kernel como una ventana deslizante que recorre toda la imagen y calcula el mapa de características resultante en cada salto. En la Figura 3.8 se muestra cómo se desliza un kernel (la ventana azul) por toda la matriz que constituye la imagen, se multiplican los valores del kernel con los valores de la imagen en cada posición, y se suman todos los valores de la ventana deslizante, consiguiendo como resultado un mapa de características (matriz de la derecha en la Figura 3.8), la cual tiene una dimensión menor que la imagen original. Esta reducción en la dimensión dependerá del tamaño de salto a la hora de deslizar el kernel por la imagen, y del tamaño del kernel en sí.

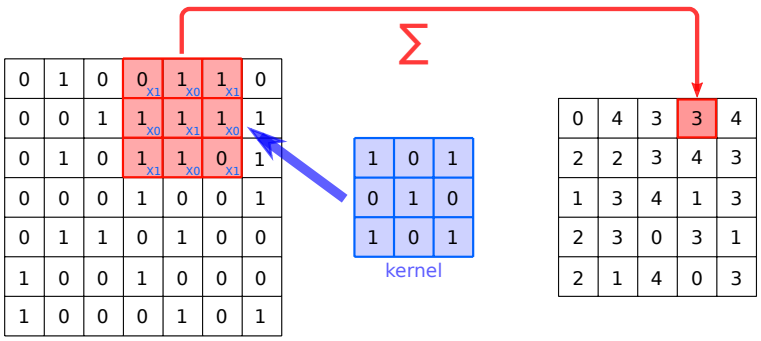


Figura 3.8: Convolución de un kernel sobre una imagen binaria, tamaño de salto 1 píxel, y kernel de 3×3.

Las redes convolucionales presentan varias ventajas comparadas con las redes convencionales en algunas aplicaciones. Para el caso de análisis de imágenes, sería muy costoso procesar independientemente cada píxel o conllevaría una reducción drástica de resolución de las imágenes de entrada,

con la consiguiente pérdida en el nivel de detalle que eso supone. Dado que se utilizan kernels más pequeños que las imágenes, se consigue aligerar el número de parámetros que hay que ajustar. Esto se debe a que los pesos de las capas que conectan la imagen de entrada con el mapa de características de salida se repiten deliberadamente. Los únicos pesos posibles son los valores que contenga el kernel, y el número de valores en el kernel es muy inferior al número de conexiones entre la imagen y el mapa de características.

Aunque existen y se utilizan formas más eficientes para computar una convolución, en este trabajo se representarán siguiendo las mismas estructuras de redes explicadas en las secciones anteriores. Como ejemplo para entender mejor los pesos repetidos, en el caso de una imagen en escala de grises de tamaño 4×4, siendo los valores de los píxeles $I_1, I_2, \ldots, I_{16}$, un kernel de 2×2 con valores $k_1, \ldots, k_4$ y un mapa de características de salida de tamaño 3×3 con valores $O_1, \ldots, O_9$, la capa de la red tendrá la siguiente estructura que se muestra en la Figura 3.9.

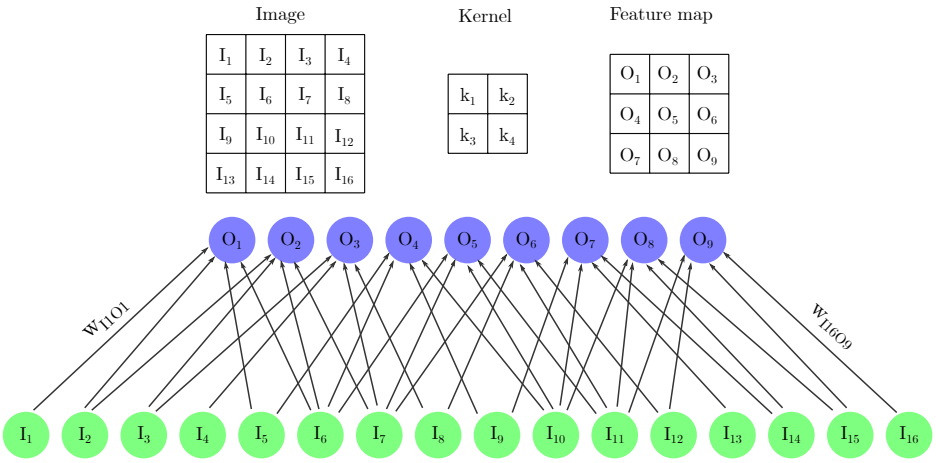


Figura 3.9: Esquema de una capa convolucional con una imagen de entrada de 4×4 y una salida de 3×3.

Siendo el peso $w_{I_i O_j}$ el que conecta la entrada i y la salida j. En este caso, los valores del kernel k_1, k_2, k_3, k_4 serán los únicos pesos que aparezcan en esta capa, y estos serán los valores de los pesos:

$$k_1 = w_{I_1 O_1} = w_{I_2 O_2} = w_{I_3 O_3} = w_{I_5 O_4} = w_{I_6 O_5} = w_{I_7 O_6} = w_{I_9 O_7} = w_{I_{10} O_8} = w_{I_{11} O_9} \tag{3.27}$$

$$k_2 = w_{I_2 O_1} = w_{I_3 O_2} = w_{I_4 O_3} = w_{I_6 O_4} = w_{I_7 O_5} = w_{I_8 O_6} = w_{I_{10} O_7} = w_{I_{11} O_8} = w_{I_{12} O_9} \tag{3.28}$$

$$k_3 = w_{I_5 O_1} = w_{I_6 O_2} = w_{I_7 O_3} = w_{I_9 O_4} = w_{I_{10} O_5} = w_{I_{11} O_6} = w_{I_{13} O_7} = w_{I_{15} O_8} = w_{I_{15} O_9} \tag{3.29}$$

$$k_4 = w_{I_6 O_1} = w_{I_7 O_2} = w_{I_8 O_3} = w_{I_{10} O_4} = w_{I_{11} O_5} = w_{I_{12} O_6} = w_{I_{14} O_7} = w_{I_{15} O_8} = w_{I_{16} O_9} \tag{3.30}$$

Por lo tanto, dado que los pesos se comparten, únicamente es necesario entrenar 4 parámetros en lugar de 36.

Por el hecho de utilizar parámetros compartidos, las redes convolucionales tienen una propiedad llamada equivarianza a la traslación. Por ejemplo siendo g cualquier función que traslada la entrada, es decir, la desplaza, entonces la función de convolución es equivariante a g. Por ejemplo, siendo I una imagen en sus coordenadas, e $I' = g(I)$, $g(I(x, y)) = I(x - 1, y)$ (desplazamiento lateral hacia la izquierda), entonces si se aplicase esta transformación a la imagen y después se aplicase la convolución, se obtendría el mismo resultado que aplicando la convolución y después la transformación.

Al procesar imágenes, puede resultar útil procesar las diferentes regiones de la imagen de la misma manera, para por ejemplo detectar bordes en la primera capa de una red convolucional. Los mismos bordes aparecen más o menos en todas partes en la imagen, por lo que es práctico compartir los parámetros en toda la imagen. En algunos casos, es posible que no sea conveniente compartir los parámetros en toda la imagen. Por ejemplo, si se procesan imágenes recortadas para que se centren en la cara de un individuo, es probable que se deseen extraer diferentes características en diferentes ubicaciones: la parte de la red que procesa la parte superior de la cara debe buscar cejas, mientras que la parte de la red que procesa la parte inferior de la cara necesita buscar un mentón.

3.6.2. Pooling

Una función de *pooling* reemplaza la salida de la red en una ubicación determinada con una estadística que resume de alguna manera los valores de salida de una región. Un ejemplo común de operación de *pooling* es el *max pooling*, el cual, dada una región de píxeles, asigna en la matriz de salida el valor máximo de esa región [47]. Existen muchas operaciones de *pooling* aparte del *max pooling*, por ejemplo calcular la media de la región, o calcular una media ponderada en función de la distancia con el píxel central. En la Figura 3.10 se muestra una representación visual de una operación de *max pooling*.

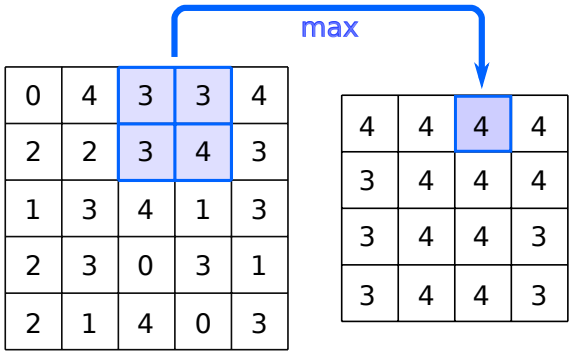


Figura 3.10: *Max pooling* para una ventana deslizante de 2×2.

Al igual que en la convolución, después de aplicar el *pooling* sobre el mapa de características con una ventana deslizante, el tamaño de la matriz resultante es menor que la matriz de entrada, disminuyendo en función del tamaño de salto y del tamaño de ventana deslizante.

El *pooling* ayuda a hacer que el sistema sea casi invariante para pequeñas translaciones de la entrada. La invariancia a la translación significa que si se trasladase la entrada en una pequeña cantidad, los valores de la mayoría de las salidas del *pooling* no cambirían [42].

3.7. Redes para detección de objetos: Faster R-CNN

En el Apartado 3.6 se han explicado los elementos básicos necesarios para la clasificación de imágenes, pero el acelerado avance en campos como los vehículos autónomos o la vigilancia inteligente hace que sean necesarios sistemas capaces no solo reconocer y clasificar cada objeto en una imagen, sino localizar cada uno dibujando un cuadro alrededor del objeto que se desea detectar. Esto hace que la detección de objetos sea una tarea significativamente más compleja que la clasificación de imágenes.

Una forma eficiente de llevar a cabo una detección de objetos utilizando redes neuronales es utilizar un tipo de arquitectura basada en propuestas de región de interés [17]. En este trabajo se ha utilizado una determinada arquitectura, la cual se denomina Faster R-CNN [21] y se compone de los siguientes módulos:

1. Un extractor de características. Una red convolucional profunda compuesta por los bloques

básicos introducidos en el Apartado 3.6.

2. Una red de propuesta de región de interés. Es una red convolucional profunda que coge la salida del extractor de características y devuelve regiones que contienen candidatos a ser un objeto que se desea detectar.

3. El Clasificador Fast R-CNN [20]. Utiliza las regiones propuestas para asignar las clases pertinentes a cada objeto que aparece en la imagen.

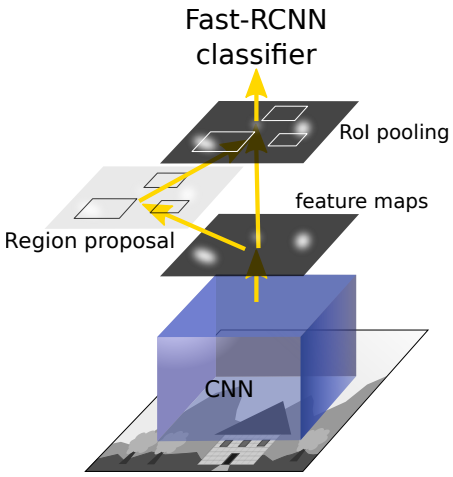


Figura 3.11: Arquitectura del módulo Faster-RCNN [21].

En los siguientes apartados se introducirán los diferentes módulos que conforman la red Faster-RCNN y que se muestran en la Figura 3.11.

3.7.1. Redes de propuesta de región de interés

Una Red de propuesta de región o *region proposal network* (RPN) toma una imagen como entrada y devuelve un conjunto de propuestas de objetos rectangulares, cada una con una puntuación que mide si esa región contiene un objeto o es parte del fondo.

La red Faster R-CNN utiliza una RPN específica que se introduce en [21]. Para generar propuestas desliza una pequeña red sobre la salida del módulo que extrae el mapa de características. Esta pequeña red toma como entrada una ventana espacial $n \times n$ del mapa de características convolucionales. Para localizar un objeto en el mapa de características, se realiza un barrido de diferentes cuadros delimitadores, los cuales se conocen como anclajes.

Anclajes

Se colocan sobre la imagen simétricamente distribuidos los puntos de anclaje. Sobre cada punto de anclaje se colocarán diferentes anclajes, y cada uno de estos será candidato de contener un objeto de interés. Sobre cada punto de anclaje se coloca un número k de anclajes, a diferentes escalas y diferentes relaciones de aspecto como se muestra en la Figura 3.12, donde los puntos de anclaje están uniformemente distribuidos en la escena, y sobre cada punto se coloca un número $k = 9$ de anclajes, 3 escalas y 3 relaciones de aspecto.

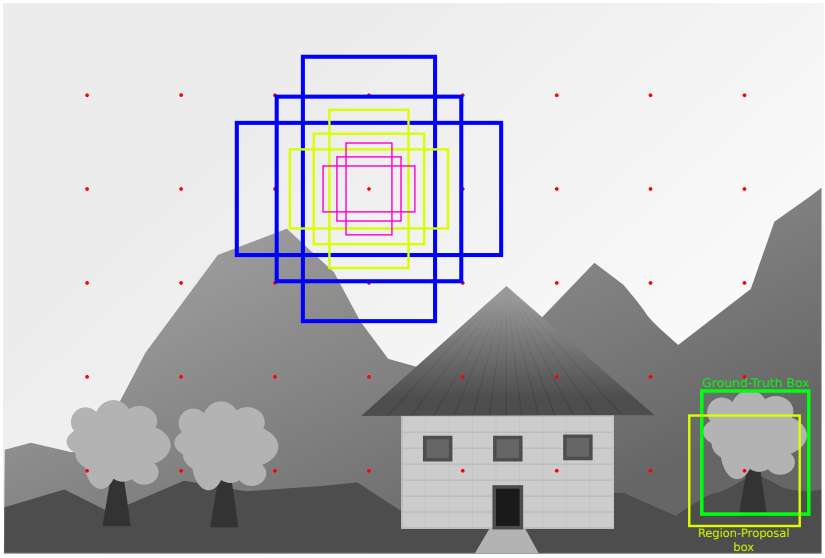


Figura 3.12: Ejemplo de propuesta de regiones en una imagen mediante anclajes.

Entrenamiento y función de coste

Para poder entrenar la RPN del Faster-RCNN primero se asigna una clase binaria a cada anclaje, la cual indica si lo que contiene es un objeto o es el fondo de la imagen [21]. Para activar los anclajes que contienen un objeto se utiliza un umbral en el valor de la intersección entre la unión del anclaje y el *ground-truth*. La intersección entre la unión, en inglés, *intersection over union* (IoU) (ecuación 3.31) , conocido originalmente como índice de Jaccard [48], es una métrica que indica cuánto se parecen dos conjuntos, en detección de objetos, cuánto se acerca el área de una detección, o el anclaje en este caso, a la etiqueta en los datos (Figura 3.13). A los anclajes que tengan un IoU>0.7 se les asigna una etiqueta positiva, en el caso especial en el que ningún anclaje supere el valor de 0.7 (esto puede ocurrir por ejemplo porque las relaciones de aspecto utilizadas no son adecuadas) se etiquetarán como objetos los anclajes con mayor valor IoU. Los anclajes con un valor de IoU<0.3 se etiquetan como fondo y el resto, los anclajes que no son ni fondo ni objeto, no se utilizan para el entrenamiento. Por ejemplo en la Figura 3.12, en la esquina inferior derecha aparece un árbol etiquetado como objeto cuyo *ground-truth* se solapa con un IoU>0.7 con el anclaje de la escala amarilla y relación de aspecto 1:1.

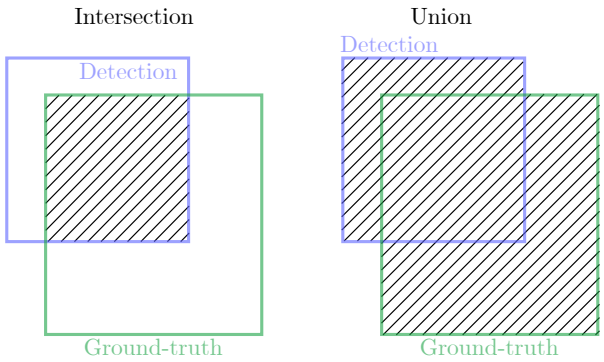


Figura 3.13: Representación de los áreas de intersección e unión en el par detección-*ground-truth*.

$$\text{IoU}(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{3.31}$$

Una vez se ha especificado cuales serán los datos con los que trabajará la RPN, para definir la función de coste es imprescindible tener en cuenta dos aspectos: el error de clasificación que supone el clasificar como fondo un objeto y viceversa, y el error de regresión que representa cuánto se acerca

un anclaje a la etiqueta real. La función de coste para una imagen se define mediante la siguiente expresión [21]:

$$C(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i C_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* C_{reg}(\boldsymbol{t}_i, \boldsymbol{t}_i^*) \tag{3.32}$$

donde i hace referencia al anclaje i, p_i es la probabilidad predicha de que un anclaje contenga o no un objeto, p_i^* es 1 si el anclaje contiene un objeto y 0 en caso de que no, $\boldsymbol{t}_i$ es un vector de cuatro componentes que representan las coordenadas de el cuadro delimitador predicho, y $\boldsymbol{t}_i^*$ es el cuadro delimitador del *ground-truth*. El coste de clasificación C_{cls} es el *log loss* [29]:

$$C_{cls} = -p_i^* \log(p_i) \tag{3.33}$$

El coste de localización C_{reg} es la función $smooth_{L_1}(\boldsymbol{t}_i - \boldsymbol{t}_i^*)$ [20]:

$$\text{smooth}_{L_1}(x) = \begin{cases} \dfrac{x^2}{2} & si \quad |x| < 1 \\ \\ |x| - \dfrac{1}{2} & si \quad |x| \geq 1 \end{cases} \tag{3.34}$$

El hecho de que el coste de localización C_{reg} esté multiplicándose con el valor p_i^* implica que únicamente contribuirán al ajuste de la localización los anclajes que contienen un objeto, ya que no tiene sentido llevar a cabo un ajuste en el cuadro delimitador en un área en la que no debería existir detección alguna. A diferencia de otras RPNs que entrenan una única red de regresión con pesos compartidos [19, 20], la que utiliza el Faster-RCNN entrena k diferentes subredes que no comparten pesos, de manera que se ajustan las relaciones de aspecto y tamaños de cada uno de los k anclajes que se deslizan por la imagen.

Los dos términos están normalizados por N_{cls} y N_{reg} y ponderados por un parámetro de equilibrio λ. En la implementación por defecto en [21], el término C_{cls} en la ecuación 3.32 se normaliza por el tamaño del *mini-batch* (es decir, $N_{cls} = 256$) y el término C_{reg} se normaliza por el número de ubicaciones de anclaje (es decir, $N_{reg} \approx 2400$). El valor por defecto de λ en [21] es de $\lambda = 10$.

Es importante tener en cuenta que la RPN no se encarga de asignar clases a los objetos, sino de devolver candidatos a ser un objeto, es decir, indicar que en una región concreta puede haber un objeto, sin especificar qué es ese objeto. De la asignación de clases a los objetos se encargará el módulo RCNN en una fase posterior.

También es importante destacar que una vez entrenada la RPN, los tamaños y formas de los anclajes son fijos. Durante el entrenamiento se ajusta cuales serán esos tamaños y formas, pero una vez terminado el entrenamiento siempre se utilizarán las mismas relaciones de aspecto y tamaños de anclaje para proponer regiones que contengan un objeto. Al igual que en la clasificación, la tarea de ajustar el cuadro delimitador de manera personalizada para cada imagen la llevará a cabo el módulo RCNN.

Puesto que muchos de los anclajes generalmente se superponen, como cabe esperar, muchas de las propuestas también se superponen sobre el mismo objeto. Para resolver el problema de las propuestas duplicadas, se utiliza el método llamado *Non-Maximum Suppression* (NMS). El NMS toma la lista de propuestas ordenadas por puntuación y descarta las propuestas que tienen un valor de IoU mayor que un umbral predefinido con la propuesta que tiene la puntuación más alta.

Por último, después de eliminar las propuestas duplicadas, se seleccionan las N propuestas que mayor puntuación tienen, descartando el resto. En [21] el valor por defecto es de $N = 2000$.

3.7.2. RoI *Pooling*

Después de haber localizado las regiones de interés con la RPN, estas serán introducidas al clasificador y regresor que se encargarán de clasificar las regiones de interés en sus clases correspondientes,

y de afinar la posición del cuadro delimitador que contiene los objetos en la imagen. El problema es que las regiones propuestas por la RPN serán de tamaños diferentes, y crear una estructura eficiente que trabaje con tamaños diferentes es complicado, para ello se lleva a cabo el *pooling* de la región de interés, o *region of interest pooling* (RoI *pooling*).

El RoI *pooling* consiste en fijar un número concreto q, partir las regiones de interés en q partes iguales, o lo más parecidas posibles, y realizar un *max pooling* en esas q partes, de esta manera, a la salida del *pooling*, todas las regiones de interés tendrán el mismo tamaño (Figura 3.14).

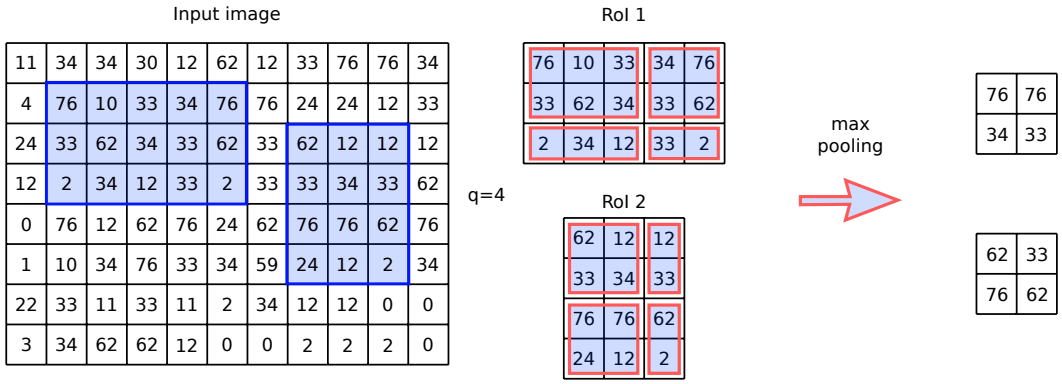


Figura 3.14: Ejemplo de RoI *pooling* para un tamaño de $q = 4$.

3.7.3. Clasificador Fast-RCNN

La última fase del Faster-RCNN es el clasificador de imágenes empleado en [20]. El cometido del clasificador es el de un clasificador de imágenes común, recibe el mapa de características de las regiones propuestas como entrada y les asigna una clase. Se añade una clase extra, la cual hace referencia al fondo de la imagen, y se utiliza para descartar regiones mal propuestas. Además en esta última fase del detector se modifica el cuadro delimitador que contendrá el objeto para ajustarlo más a los limites del objeto.

Este módulo está formado por dos subredes:

1. Una capa totalmente conectada con $N + 1$ unidades donde N es el número total de clases y ese extra es para la clase de fondo.
2. Una capa completamente conectada con $4N$ unidades. El objetivo es realizar una regresión del cuadro delimitador, por lo tanto es necesario ajustar $\Delta_{center_x}, \Delta_{center_y}, \Delta_{width}, \Delta_{height}$ para cada una de las N clases posibles.

El procedimiento para entrenar el clasificador Fast-RCNN es similar al que se aplica en la RPN. En este caso, a las propuestas con un valor de IoU>0.5 con alguna clase se les asigna dicha clase, a las propuestas con un valor de IoU entre 0.1 y 0.5 se les asigna la clase fondo, y las propuestas con un valor menor a 0.1 se ignoran. Esto se debe a que al ser la fase más avanzada de la red se cuenta con tener propuestas suficientemente buenas, y el objetivo real y la tarea más complicada que debe llevar a cabo esta fase es clasificar los objetos.

Los valores $\Delta_{center_x}, \Delta_{center_y}, \Delta_{width}$ y Δ_{height} de la red de regresión se obtienen comparando los valores de salida con los valores del anclaje.

Las funciones de coste en estas capas son las mismas que en la RPN, la función $smooth_{L_1}$ para los valores de la regresión y el *log loss* para la clasificación.

Capítulo 4

Transferencia del aprendizaje: Comparativa de inicialización de parámetros

En muchas aplicaciones, las redes neuronales suelen entrenarse desde cero, y se basan únicamente en los datos de entrenamiento para ajustar sus parámetros. Sin embargo, a medida que se entrenan cada vez más redes para diversas tareas, es razonable buscar métodos que eviten tener que empezar desde cero y, en su lugar, pueda basarse en los resultados de las redes previamente entrenadas [49]. En el caso de la detección de objetos, puesto que las redes tienden a ser muy profundas, y por lo tanto el número de parámetros que se han de ajustar es muy elevado, para un entrenamiento eficiente se requiere una base de datos de entrenamiento muy extensa, al igual que un tiempo de entrenamiento muy alto.

Por ello, es habitual partir de una red previamente entrenada para un propósito similar en lugar de inicializar los pesos y sesgos aleatoriamente. Las redes de detección de objetos que han sido entrenadas con conjuntos de datos grandes y para detectar una amplia variedad de objetos cuentan con la capacidad de distinguir e identificar múltiples formas, lo cual hace que partir de los parámetros de una red preentrenada sea mucho más conveniente que entrenarla desde cero en muchos casos, sobre todo cuando no se cuenta con un gran número de ejemplos de entrenamiento.

El primer objetivo de este capítulo es comparar dos inicializaciones distintas. El repositorio *Tensorflow object detection model zoo* [28] ofrece la posibilidad de descargar varios modelos preentrenados con diferentes bases de datos. La red neuronal que se comparará en este caso es una Faster-RCNN [21] con la red convolucional Resnet 101 [33] como extractor de características. Los dos modelos difieren en las bases de datos con las que han sido entrenadas, una ha sido entrenada con los datos de COCO y otra con los datos de KITTI.

Se reentrenarán estas redes para desempeñar el cometido de la detección de personas, o de peatones si se habla en términos relacionados con el tráfico. Para ello se ha utilizado un subconjunto de 1000 imágenes extraídas del conjunto de datos de [50], en las cuales aparecen un total de 8466 peatones. Para la evaluación de los modelos se han utilizado 446 imágenes que contienen 3474 peatones anotados, de un conjunto diferente, también de [50].

En la parte final de este capítulo, se entrenará una red para detección de objetos partiendo desde cero, la misma arquitectura que la que se compara en la sección anterior. En este caso se hará un entrenamiento con mayor número de épocas, ya que la inicialización es aleatoria y el módulo convolucional tendrá que aprender las características desde cero. Se comparará el rendimiento de esta red con las dos analizadas anteriormente para poder observar con mayor detalle la importancia de la transferencia del aprendizaje en redes muy profundas.

4.1. Comparativa de Faster-RCNN Resnet 101 partiendo desde COCO y desde KITTI

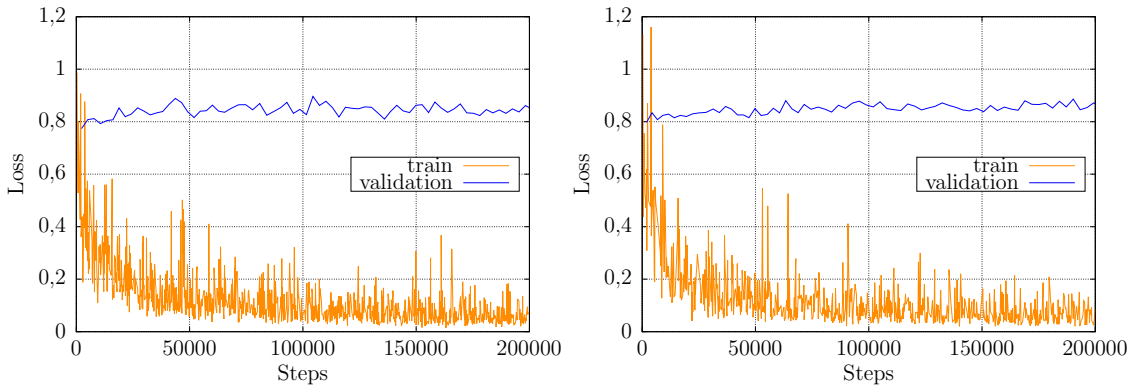
Como se ha explicado en la anterior sección, las bases de datos utilizadas para entrenar las dos inicializaciones de parámetros que se van a comparar son COCO y KITTI:

- La base de datos COCO[1] es gran conjunto que contiene más de 200 000 imágenes distribuidas en 80 clases de objetos que representan escenas del mundo real [51]. Es una base de datos lo suficientemente grande para que una red bien entrenada con este conjunto de datos sea capaz de aprender características visuales de calidad para el reconocimiento y detección de objetos en imágenes.
- KITTI[2] es una base de datos recopilada mediante distintos sensores, como cámaras, pares estéreo o LiDARs, colocados en un automóvil [14]. Por lo tanto, contiene datos 2D y 3D. El conjunto de datos para detección de objetos y estimación de orientación de objetos consta de 7481 imágenes de entrenamiento y 7518 imágenes de evaluación, que comprenden un total de 80,256 objetos etiquetados.

Para hacer la comparación entre los dos modelos, estos se han reentrenado bajo especificaciones idénticas:

- **Tamaño de imagen:** 640 × 480 píxeles
- **Tamaño de mini-batch:** 2 imágenes
- **Número de pasos:** 200.000
- **Velocidad de aprendizaje:** 0,0002
- **Relaciones de aspecto inicial de los anclajes:** [0.25, 0.33, 0.5, 1.0, 2.0]
- **Escalas iniciales de los anclajes:** [0.25, 0.5, 1.0, 2.0]
- **Número máximo de regiones propuestas:** 100
- **Regularización** *Dropout*

En la siguiente figura se muestra la evolución de la función de coste a lo largo del entrenamiento:



(a) Modelo preentrenado con COCO

(b) Modelo preentrenado con KITTI

Figura 4.1: Valores de la función de coste en cada paso del entrenamiento. La línea naranja muestra los valores aplicados a los ejemplos de entrenamiento, y la línea azul muestra los valores obtenidos desde los datos de evaluación.

[1] `http://cocodataset.org`
[2] `http://www.cvlibs.net/datasets/kitti/`

Como puede observarse, aunque el coste sobre los datos de entrenamiento baja (esto es lógico, ya que los parámetros se ajustan para que esto ocurra), el coste aplicado a los ejemplos de evaluación no varía mucho, incluso comienza a subir ligeramente. También puede observarse que los valores son muy similares para ambas inicializaciones de pesos.

Para evaluar los resultados de los detectores en mayor detalle, en el siguiente apartado se explicarán las métricas utilizadas.

4.1.1. Métricas de evaluación

Para medir el rendimiento de los modelos existen dos cuestiones importantes, la primera hace referencia a la velocidad de inferencia, es decir, cuánto tarda el modelo en recibir una imagen en la entrada, procesarla y devolver las detecciones. La segunda cuestión está más enfocada en la capacidad que tiene el modelo de hacer buenas detecciones.

Este apartado se centrará en la segunda cuestión, para eso se han utilizado dos medidas, la de la precisión:

$$Precision = \frac{TP}{TP + FP} \tag{4.1}$$

donde TP es el número total de verdaderos positivos y FP es el número de falsos positivos. La segunda medida se conoce como exhaustividad o *recall* en inglés:

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

donde FN es el número de falsos negativos.

El modelo de detección devuelve no solo detecciones sino probabilidades de que dicha detección sea acertada. Fijando diferentes umbrales puede construirse una curva de precisión y exhaustividad, calculando los valores de cada uno para un umbral dado (Figura 4.2).

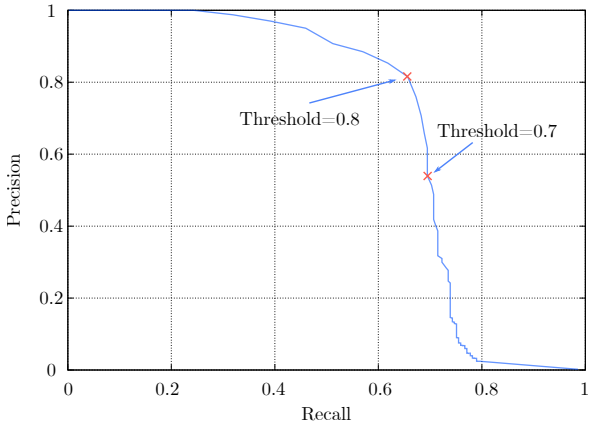


Figura 4.2: Ejemplo de una curva de precisión y exhaustividad

El área bajo la curva de precisión y exhaustividad se conoce como la precisión media o *Average Precision* . En el caso particular de la detección de peatones solamente existe una clase a predecir, pero generalmente los detectores de objetos suelen tener múltiples clases. La precisión media se calcula para cada clase y el promedio de la precisión media o *mean average precision* (mAP), proporciona una visión más general de cómo de bueno es el detector para todas las clases. Para un

número total de N clases:

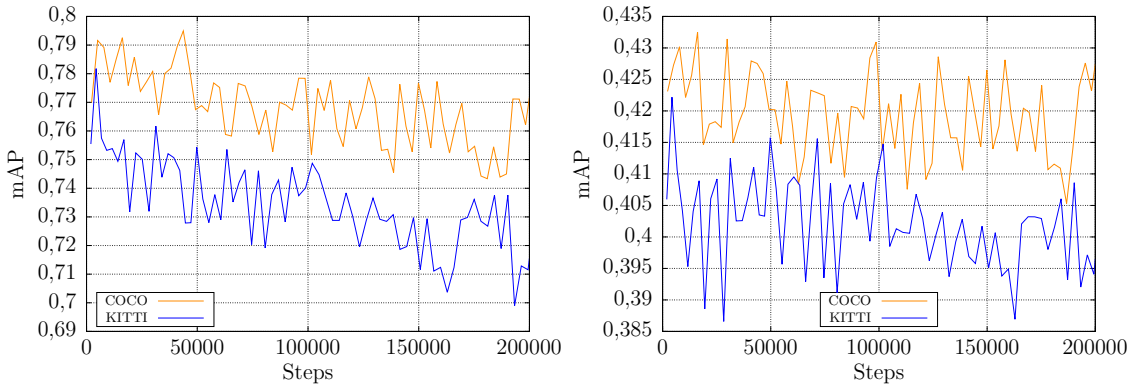
$$mAP = \frac{1}{N} \sum_{class=1}^{N} AP_{class} \tag{4.3}$$

El criterio que se utiliza para determinar si una detección es buena, es decir, si es un verdadero positivo, tiene que ver con el IoU (Ecuación 3.31). Si una detección con una clase determinada tiene un IoU mayor a un umbral con un *ground-truth* de esa clase, entonces se considerará como buena, en caso contrario se considerará como falso positivo.

La medida del promedio de la precisión media o mAP para un único valor umbral de IoU=0.5 se conoce como el mAP de PASCAL. Existe otra medida, el mAP de COCO, y también se utiliza comúnmente para evaluar detectores de objetos. Esta última métrica se calcula haciendo la media de diez valores de mAP, obtenidos para los valores de IoU=[0.5, 0.55, ..., 0.95].

4.1.2. Resultados

En este apartado se evaluarán los modelos, para ello se utilizará exclusivamente el conjunto de datos de evaluación. Para cada paso del entrenamiento se obtiene un valor del mAP tanto de PASCAL como de COCO. En los siguientes gráficos se muestra la evolución de estas medidas a lo largo del entrenamiento:



(a) mAP de PASCAL

(b) mAP de COCO

Figura 4.3: Evolución de los valores del mAP de PASCAL y COCO sobre los datos de evaluación para los dos modelos durante el entrenamiento.

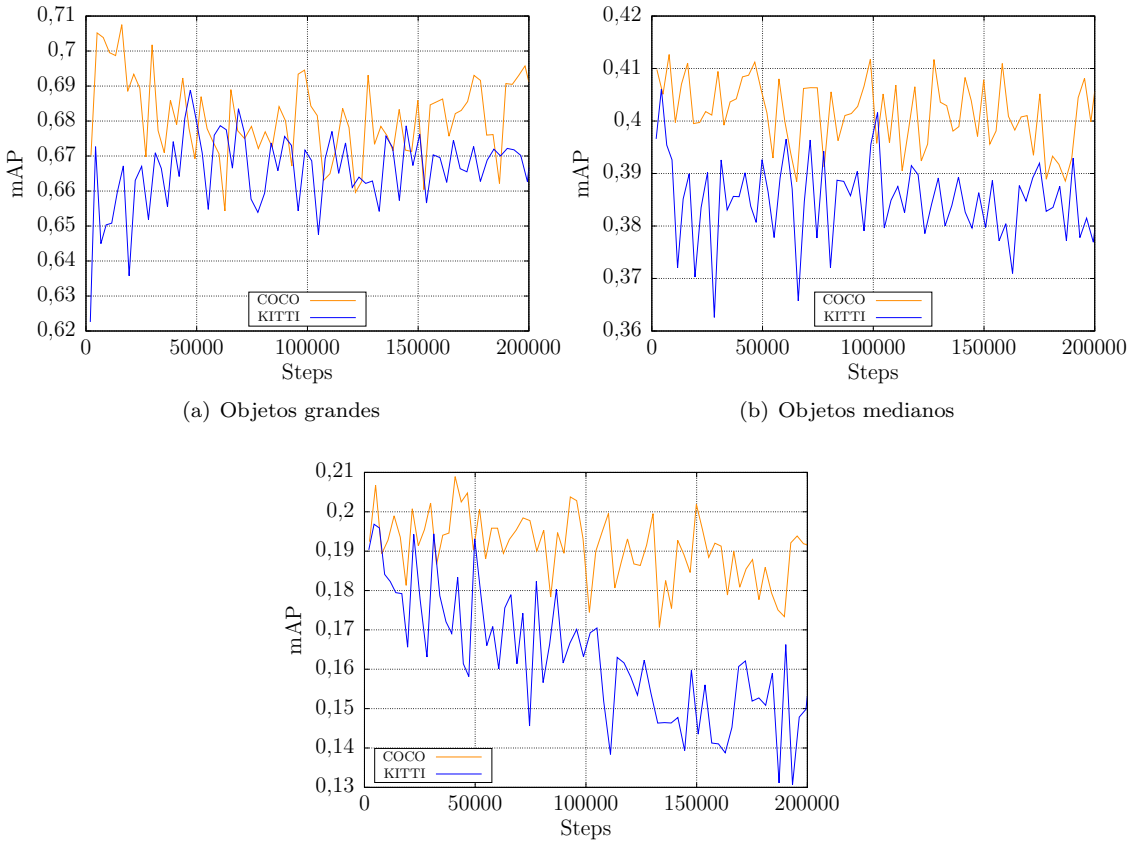
Puede observarse en ambas figuras, como para los dos modelos, la tendencia es descendente. Los modelos, conforme avanza el aprendizaje, tienen una capacidad de generalizar ligeramente peor. En la siguiente tabla se muestran los resultados más significativos obtenidos durante el aprendizaje:

	PASCAL mAP			**COCO mAP**		
Inicialización	**max**	**step**	**final**	**max**	**step**	**final**
COCO	0.7949	43712	0.7718	0.4325	16171	0.4275
KITTI	0.7818	4330	0.7309	0.4222	4330	0.3940

Tabla 4.1: Valores de los mAP de PASCAL y COCO de los dos modelos. Se muestra el valor máximo alcanzado, y el paso en el que se ha alcanzado, así como el valor obtenido al final del entrenamiento.

El siguiente análisis se centra en estudiar el mAP (de COCO) para peatones de diferentes tamaños. Se han fijado tres rangos: grande (área mayor a 96×96 píxeles), mediano (área entre 96×96 y 32×32 píxeles) y pequeño (área menor a 32×32 píxeles). Puede ocurrir, y es común en detección

de objetos, que algunos tamaños, sobre todo los pequeños, sean más complicados de detectar y clasificar.



(a) Objetos grandes

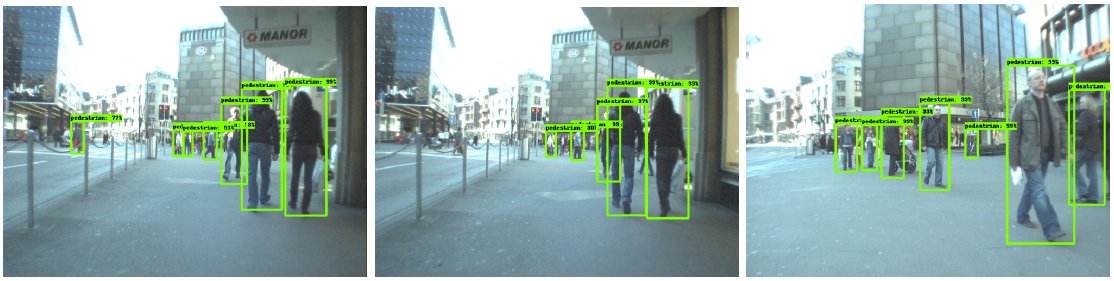
(b) Objetos medianos

(c) Objetos pequeños

Figura 4.4: Evolución de los valores del mAP de COCO sobre los datos de evaluación para los dos modelos durante el entrenamiento aplicados a diferentes rangos de tamaño de peatón.

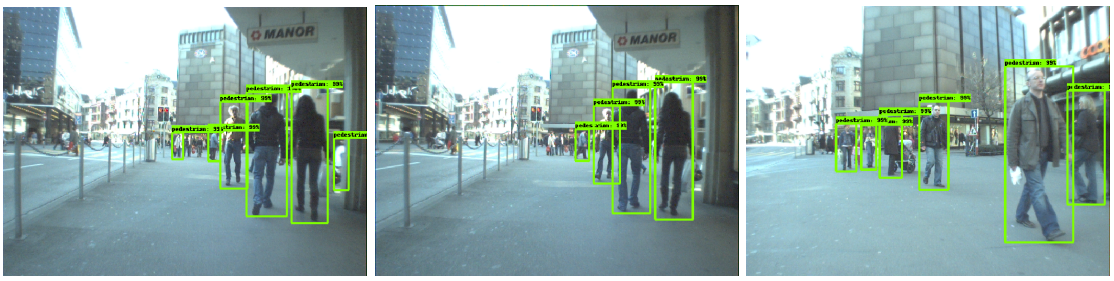
Como se observa en las curvas de la Figura 4.4, mientras que el valor del mAP en los tamaños grande y mediano oscila, pero no muestra una tendencia descendente, en los peatones pequeños sí que aparece una pendiente negativa, bastante notable para el modelo inicializado con los pesos de KITTI.

En la siguientes imágenes se muestran unos cuantos ejemplos significativos de detecciones de peatones en escenas muy transitadas:



(a) (b) (c)

Figura 4.5: Detecciones del modelo preentrenado con COCO.



(a) (b) (c)

Figura 4.6: Detecciones del modelo preentrenado con KITTI.

4.1.3. Discusión de los resultados

En general, comparando las dos redes entrenadas de manera idéntica y con inicializaciones diferentes frente a frente, el mayor rendimiento lo ofrece la red preentrenada con COCO. Este fenómeno no resulta muy sorprendente ya que es un conjunto de datos considerablemente mayor a KITTI. En las curvas superpuestas de las Figuras 4.3 y 4.4, así como en la Tabla 4.1 la red preentrenada con COCO muestra los mejores valores en todos los casos. Aún así, la diferencia con la inicialización de KITTI no es muy grande.

Por otro lado, en la Figura 4.1 se muestra el valor de la función de coste durante el entrenamiento en los dos modelos, tanto para el conjunto de entrenamiento como para el conjunto de evaluación. El valor del coste de entrenamiento y el coste de evaluación es muy similar al principio del entrenamiento, esto puede resultar lógico, ya que ambas bases de datos (las que se han utilizado para entrenar y evaluar) tienen características similares, han sido capturadas con la misma cámara y ambas son subconjuntos del mismo conjunto de datos. A lo largo del entrenamiento el valor del coste sobre los datos de entrenamiento disminuye, como es de esperar, ya que los parámetros se ajustan precisamente para que eso ocurra. Sin embargo, la curva del coste sobre los datos de evaluación apenas cambia en los dos modelos, lo cual puede ser indicativo de que, aunque el modelo aprende a clasificar mejor el conjunto de datos de entrenamiento, no es capaz de generalizar ese aprendizaje a datos externos, llegando incluso a empeorar ligeramente (Figura 4.3), dando señales de un pequeño sobreajuste a los datos de entrenamiento.

También resulta significativo que en la comparativa del mAP para diferentes tamaños (Figura 4.4), aparece una diferencia considerable entre el descenso del mAP para peatones pequeños si se compara la curva de COCO y la de KITTI. Teniendo en cuenta que en la Figura 4.3, al final del entrenamiento la diferencia de valores de mAP entre los dos modelos es mayor que la que existe al inicio, el descenso en los peatones pequeños puede ser una causa de que esto ocurra. Además, en las Figuras 4.5 y 4.6, se muestran ejemplos de cómo el modelo preentrenado con COCO es más efectivo clasificando peatones pequeños que el de KITTI.

En la base de datos de entrenamiento [50], los peatones con un tamaño de altura de píxel menor a 48 píxeles no aparecen etiquetados. Esta característica de la base de datos puede acarrear un menor rendimiento a la hora de clasificar objetos pequeños.

Por último, poniendo en contexto los resultados obtenidos con el estado del arte (Tabla 2.2 b), aunque las bases de datos utilizadas para evaluar los modelos no son las mismas, los valores de mAP con IoU=0.5 obtenidos para los dos modelos entrenados muestran un rendimiento muy similar al que aparece en el Faster-RCNN en la modalidad “Fácil”.

4.2. Detección de peatones con Faster-RCNN Resnet 101 partiendo con inicialización aleatoria.

En este apartado se expondrá la evolución del entrenamiento de la red Faster-RCNN Resnet 101 partiendo de una inicialización aleatoria. Para el entrenamiento se ha utilizado exactamente la

misma configuración expuesta en el apartado anterior, con una única modificación: en lugar de 200 000 pasos, se han realizado 500 000, dado que al partir desde cero, el tiempo de entrenamiento necesario para alcanzar valores medianamente aceptables de detección es considerablemente mayor. La base de datos que se ha empleado para entrenamiento y validación también coincide con la expuesta en la sección anterior. En la Figura 4.7 se muestran los valores de la función de coste a lo largo del entrenamiento.

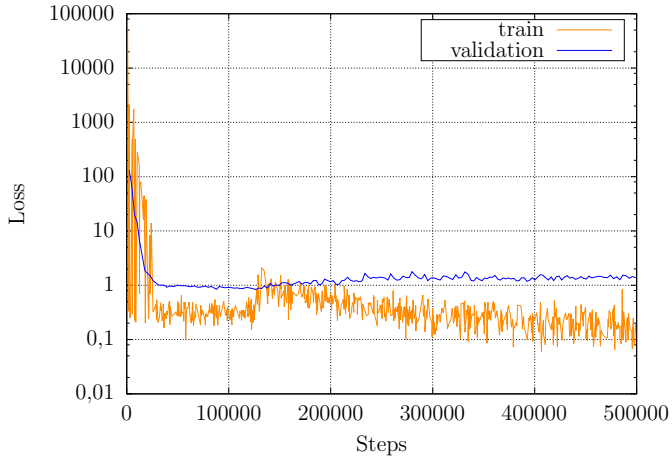
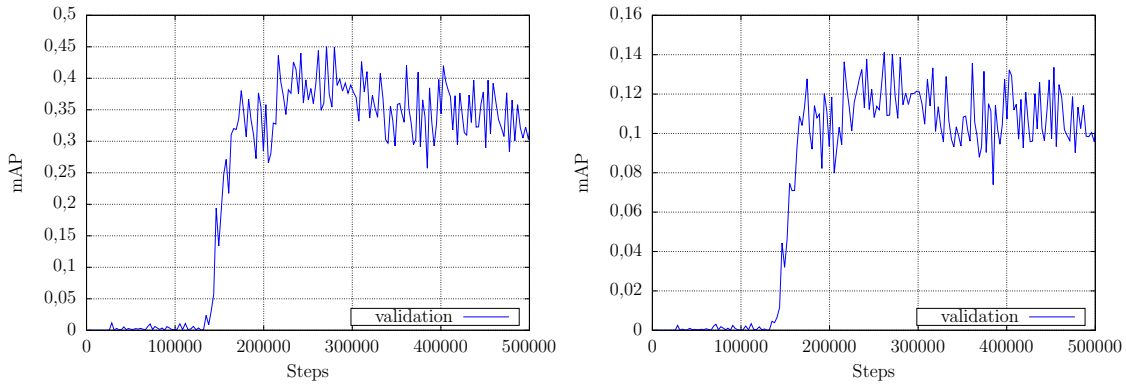


Figura 4.7: Evolución de los valores de la función de coste a lo largo del entrenamiento. Dado que los valores iniciales son notablemente más elevados que los finales, se ha representado el coste en escala logarítmica para una visualización más cómoda.

A partir de las 150 000 épocas aproximadamente, los valores de la función de coste a penas varían. Sobre el conjunto de entrenamiento el coste baja ligeramente, mientras que ocurre lo contrario sobre los datos de evaluación, mostrando síntomas de sobreajuste.

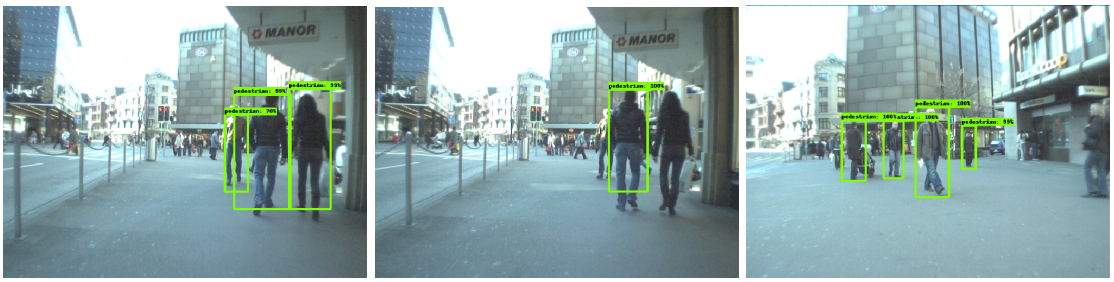
En las siguiente figuras, siguiendo en la línea del análisis del apartado anterior, primero se mostrará la evolución de los valores del mAP sobre los datos de evaluación a lo largo de las iteraciones del entrenamiento (Figura 4.8, y después se mostrarán ejemplos de detecciones significativas (Figura 4.9):



(a) mAP de PASCAL

(b) mAP de COCO

Figura 4.8: Valores de las medidas mAP de PASCAL y COCO sobre los datos de evaluación a lo largo del entrenamiento.



(a) (b) (c)

Figura 4.9: Detecciones del modelo entrenado desde cero.

Con la intención de comparar este modelo con los previamente analizados, en la siguiente figura se muestran las curvas de precisión y exhaustividad de los tres modelos, una vez terminado el entrenamiento de estos:

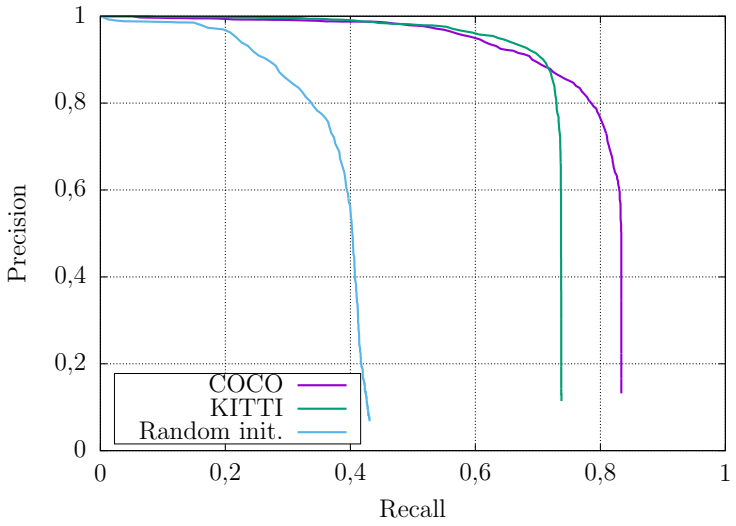


Figura 4.10: Curva de precisión-exhaustividad de los dos modelos Faster-RCNN Resnet para un IoU=0.5. Cada una de las curvas representa una inicialización de los parámetros de la red, y la leyenda hace referencia a la base de datos con la que se ha realizado el entrenamiento de los pesos iniciales.

Por ultimo, en la siguiente tabla se muestran los mismos valores que en la Tabla 4.1, añadiendo los obtenidos con la inicialización desde cero:

	PASCAL mAP			**COCO mAP**		
Inicialización	**max**	**step**	**final**	**max**	**step**	**final**
COCO	0.7949	43712	0.7718	0.4325	16171	0.4275
KITTI	0.7818	4330	0.7309	0.4222	4330	0.3940
Aleatoria	0.4502	270911	0.33508	0.1412	261719	0.1098

Tabla 4.2: Valores de los mAP de PASCAL y COCO de los tres modelos. Se muestra el valor máximo alcanzado, y el paso en el que se ha alcanzado, así como el valor obtenido al final del entrenamiento.

4.2.1. Discusión de los resultados

En general la red entrenada desde cero tiene un rendimiento muy inferior a las que parten de inicializaciones preentrenadas. Además, a lo largo del entrenamiento, cerca de la iteración 200 000,

en el modelo no consigue mejorar el rendimiento de generalización, el valor de mAP se estanca. Ese fenómeno se debe sobre todo a la cantidad de datos con la que se está entrenando la red, 1000 imágenes no son suficientes para entrenar una red de estas características de manera efectiva. Para poder alcanzar un rendimiento comparable a las redes entrenadas con KITTI y con COCO es necesario contar con un conjunto de datos de entrenamiento que se asemeje a estas bases de datos, tanto en número de ejemplos como en variedad de escenarios.

En cuanto a las curvas que se muestran en la Figura 4.10, el detector entrenado desde cero obtiene una precisión aceptable, pero una exhaustividad baja. Esto quiere decir que aunque no detecta muchos falsos positivos, sí que deja de detectar muchos peatones cuando estos aparecen en la imagen. Ocurre algo similar con el detector preentrenado con KITTI, que obtiene una precisión mayor a la exhaustividad, aunque con valores muy superiores al entrenado desde cero. En el caso del detector preentrenado en COCO, es un detector bastante equilibrado, proporciona valores altos tanto en precisión como en exhaustividad.

Capítulo 5

Evaluación de modelos DNN en Nvidia DrivePX2

Como ya se ha descrito, el área de interés de este trabajo es la detección automática de objetos en escenas de carretera mediante el análisis de imágenes para su aplicación a los sistemas de asistencia a la conducción. En este capítulo se evaluarán los tiempos de ejecución de varios detectores de objetos basados en modelos DNN al ser utilizados en el dispositivo Nvidia Drive^{TM}PX2. En primer lugar se presentan las características técnicas de dicha plataforma para la conducción autónoma y a continuación se describe la evaluación realizada junto con los resultados obtenidos.

5.1. Nvidia Drive PX2

El dispositivo Nvidia Drive PX2 Autochauffeur[1] está diseñado específicamente para la investigación y la integración de soluciones de inteligencia artificial en vehículos sin conductor (Figura 5.1). Esta plataforma tiene dos Nvidia Tegra X2 SoC (conocidos como TegraA y TegraB), donde cada SoC contiene 2 núcleos Denver, 4 núcleos ARM A57 y una GPU Pascal. Esta distribución simétrica está pensada en principio para fines de redundancia, es decir, llevar a cabo procesos independientes en cada Tegra y no para paralelización de un mismo proceso.

La dGPU tiene 1152 núcleos CUDA y 4 GB de memoria, mientras que la iGPU tiene 256 núcleos y utiliza la memoria del sistema de 7 GB. Además, el dispositivo incluye un conjunto de interfaces (GMSL, USB, Ethernet, CAN, FlexRay, etc.) para diferentes sensores como cámaras, LiDAR, GPS, señales de bus CAN, etc. El sistema operativo se basa en la distribución Ubuntu Linux.

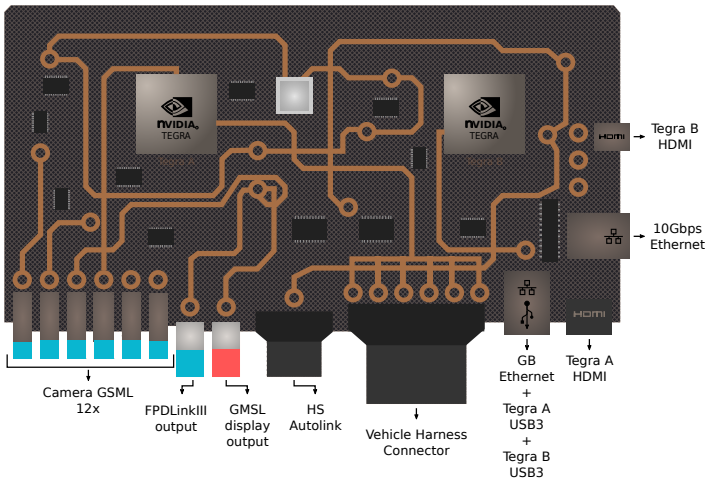


Figura 5.1: Esquema de la apariencia del Drive PX2 y descripción de sus conectores.

[1] https://www.nvidia.com/en-us/self-driving-cars/drive-platform/

5.2. Detección de objetos basada en modelos DNN

El dispositivo Nvidia Drive PX2 cuenta con un software llamado DriveWorks[2], el cual contiene librerías de detección, las cuales incluyen procesamiento de sensores, correlación de sensores, fusión de sensores, segmentación y detección y clasificación de objetos por aprendizaje profundo. También incluye una librería de localización y otra de planificación de ruta. El problema con estas librerías es que el acceso a parámetros y detalles de sus modelos es opaco y/o cerrado. Por este motivo y basado en la experiencia previa dentro del departamento de Vicomtech, se ha optado por usar Tensorflow, que es de código abierto y existen modelos ya entrenados por otros grupos de investigación y disponibles públicamente. Esta biblioteca permite trabajar utilizando la capacidad computacional de las GPUs, lo cual agiliza de manera muy considerable tanto el entrenamiento como la inferencia de las redes neuronales, por el hecho de poder realizar un gran número de operaciones en paralelo. En particular, este capítulo se centra en la fase de inferencia.

Con la intención de medir los tiempos de inferencia de los detectores de objetos basados en DNNs durante su ejecución el Drive PX2, se han seleccionado tres modelos preentrenados de *Tensorflow object detection model zoo* [28]: tres modelos que utilizan la arquitectura Faster-RCNN con los extractores de características **Resnet101**, **Inception v2** y **Inception Resnet v2**. El motivo principal de su elección es su demostrada capacidad para obtener tasas de detección elevadas, siendo redes de distinta complejidad. Se ha inferido sobre estos modelos utilizando el Drive PX2 y además, se ha comparado la velocidad de inferencia con dos tarjetas gráficas de alto nivel, la de referencia en [28], una Nvidia GeForce GTX TITAN X y también con una Nvidia Tesla P100M.

Por último, para observar la gran diferencia de velocidad existente entre GPU y CPU, se han repetido las mismas pruebas en un PC con 4 GB de RAM y procesador Intel® Core™ i5 CPU 650 @ 3.20GHz × 4. En la Tabla 5.1 se pueden observar los valores obtenidos.

El tamaño de las imágenes que se han probado es de 600×600 píxeles, que es el tamaño de entrada por defecto de las redes.

	Tiempo de inferencia (ms) por modelo Faster-RCNN		
Dispositivo	**Resnet101**	**Inception v2**	**Inception Resnet v2**
Drive PX2	331.9	215.7	1697.2
Tesla P100M 16GB	90.1	52.3	344.7
GeForce GTX TITAN X	106	58	620
Intel Core i5 3.2Ghz	9210.9	2284.9	55596.4

Tabla 5.1: Tiempos de inferencia de tres modelos Faster-RCNN en diferentes dispositivos.

En la Tabla 5.1 puede observarse que el Drive PX2 es más lento que las GPUs de alto rendimiento, alrededor de un orden de magnitud. Sin embargo, comparado con un PC convencional, que procesa las imágenes con CPU, es considerablemente más rápido, en el caso más extremo, el del Faster-RCNN inception resnet v2, el PC procesa una sola imagen en 55.6 segundos mientras que el Drive PX2 tarda aproximadamente 1.7 segundos.

Aunque la tarjeta Nvidia Drive PX2 muestra ganancias importantes comparado con una CPU, los tiempos que presenta no son suficientes para la aplicación de procesamiento en tiempo real de vehículos autónomos (comúnmente se suele trabajar con el límite de 33.3333ms/imagen = 30fps). Por otro lado, las GPUs de alto rendimiento no pueden instalarse en coches a día de hoy por cuestiones prácticas y de consumo: ordenadores sin ventilador (las GPUs grandes no podrían ventilarse bien), demasiados vatios y restricciones de tamaño. La tarjeta Nvidia Drive PX2 es un dispositivo diseñado para I+D y pruebas, por lo que aunque es muy útil para el desarrollo, es indicativo de que, como se ha comentado en el capítulo del estado del arte, aún queda mucho trabajo por hacer.

En definitiva, los algoritmos actuales están más avanzados que las capacidades e integración de la tecnología en coches. Por este motivo hay muchas alianzas de empresas tecnológicas, con empresas de electrónica y chips y fabricantes de coches. Buscan romper esas barreras y ver quién llega antes

[2] `https://developer.nvidia.com/driveworks`

a tener coches autónomos disponibles.

En cuanto a las diferencias en el tiempo de inferencia de los tres modelos, el modelo que utiliza el módulo convolucional Inception v2 es el más rápido de los tres, seguido por el Resnet 101, y el Inception Resnet v2 es el más lento. Eso se debe en parte al número de parámetros que contiene cada red, las más ligeras son las más rápidas. Además, aunque el número de capas convolucionales que tiene la Inception v2 es similar al de la Resnet 101, en la Inception v2, por su arquitectura, se pueden hacer muchas más operaciones en paralelo que en la Resnet.

Capítulo 6

Caso práctico: Detector de baches en carretera

Este capítulo tiene como objetivo estudiar un caso de uso real de detección de baches en carretera. Lo que se describe a lo largo del capítulo es fruto del trabajo del departamento de sistemas de transporte inteligentes e ingeniería de Vicomtech como parte del proyecto AUTOPILOT[3].

Un bache es un pequeño desnivel en el suelo o en el pavimento, producido por la pérdida o hundimiento de la capa superficial. La formación de baches en la carretera es generada por un conjunto de factores, entre los cuales, las causas más relevantes son el volumen de tráfico, la carga por eje de vehículos pesados (autobuses y camiones) y las condiciones ambientales (contraste de temperatura día / noche, nieve, lluvia, etc.). Además, el tipo de material de la superficie de la carretera, el terreno subyacente y las técnicas de construcción influyen en la calidad y la resistencia del pavimento [52].

Los baches en carretera son visibles para el ojo humano y pueden detectarse automáticamente mediante algoritmos apropiados de procesamiento de imágenes. En este capítulo del trabajo se estudiarán modelos de redes neuronales profundas para la detección de baches en carretera. Como ya se ha descrito en los capítulos anteriores, dichas redes han logrado un rendimiento de detección impresionante en varios desafíos [13], y en particular aplicados a la detección de objetos como se estudia a lo largo de este trabajo. Para entrenar estos modelos, en primer lugar se ha recopilado un gran conjunto de datos de imágenes en color con anotaciones de baches. Estas contienen escenas de caminos de diferentes lugares y condiciones ambientales, tomadas con diferentes cámaras, vehículos y puntos de vista, como se describe ampliamente en la Sección 6.1.

6.1. Análisis de la base de datos

Para este trabajo se han recopilado imágenes con varios tipos de baches bajo diferentes condiciones climáticas y de iluminación. Los baches más comunes muestran un borde pronunciado que describe una forma elíptica. Sin embargo, hay algunos que describen una forma más cuadrada y otros que no tienen un borde pronunciado. La apariencia de los baches puede ser más oscura o más clara que la carretera en función de su profundidad. Los más profundos se ven más oscuros debido a la sombra del borde, mientras que en los planos es posible ver el suelo o la grava dentro del bache. Además, algunos baches aparecen llenos de agua y algunos de ellos muestran reflejos de la escena en la superficie (Figura 6.1).

Figura 6.1: Ejemplo de varios tipos de bache en la base de datos.

La base de datos consta de dos conjuntos, datos de entrenamiento y datos de evaluación.

6.1.1. Datos de entrenamiento

La base de datos de entrenamiento la constituyen 5874 imágenes, de las cuales 5774 se han utilizado para el entrenamiento y 100 (elegidas aleatoriamente) se han utilizado para la validación. El número total de baches en el subconjunto de entrenamiento es de 9716 mientras que en el subconjunto de validación aparecen 171. Todas las imágenes han sido capturadas desde un vehículo, utilizando diferentes cámaras, colocadas en distintas posiciones y con distintas perspectivas. Las fuentes desde las cuales se ha obtenido esta base de datos son las siguientes:

- 4030 imágenes del tamaño de 3680×2760 píxeles y sus etiquetas se han obtenido de [10]. Esta imágenes han sido capturadas con una cámara GoPro colocada en el interior del vehículo, apuntando al frente desde el salpicadero, capturando lo que ocurre más allá del parabrisas frontal.
- 1644 imágenes de 3680×2760 píxeles se han obtenido de grabaciones de vídeo de Valeo[1] para el proyecto AUTOPILOT[3]. Estas imágenes se han grabado en los aledaños de París, con una cámara de ojo de pez dedicada para vehículos colocada en el parachoques frontal de un Volkswagen Tiguan 2. Los baches que aparecían en estas imágenes han sido manualmente anotados.
- Las 100 imágenes restantes han sido obtenidas con la herramienta *street-view* del programa Google Earth Pro. Estas imágenes tienen una resolución de 1236×804 píxeles y han sido tomadas desde el techo del vehículo de Google. La mayoría de estas imágenes se han obtenido de las calles de la ciudad de Tirana en Albania, y algunas pocas de las calles de San Sebastián y Bilbao. Los baches han sido anotados manualmente.

Las imágenes han sido anotadas utilizando la herramienta LabelImg [53] y todas estas han sido redimensionadas a un tamaño de 1024×800 para no reducir más el pequeño tamaño de los baches con respecto al resto de la escena.

Además, con la intención de estudiar la forma y tamaño de los baches del conjunto de datos, se ha realizado un diagrama de caja mostrando las relaciones de aspecto y el área de píxel de las anotaciones de los baches, como se muestra en la Figura 6.2.

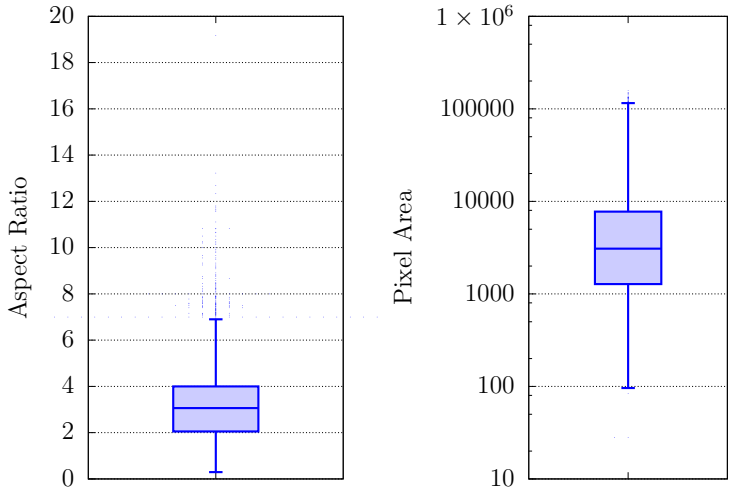


Figura 6.2: Diagrama de cajas de la relación de aspecto (izquierda) y de el área de píxeles en escala logarítmica (derecha) de los baches que aparecen en el conjunto de entrenamiento (1024×800 píxeles). Los valores para el gráfico de la izquierda son: $Q_1 = 2{,}053, Q_2 = 3{,}062$ y $Q_3 = 4{,}0$. Los valores para el gráfico de la derecha son: $Q_1 = 1276, Q_2 = 3081$ y $Q_3 = 7744$.

[1] Valeo es una organización que trabaja conjuntamente con Vicomtech en el proyecto AUTOPILOT.

6.1.2. Datos de evaluación

El conjunto de datos de evaluación o test también se ha obtenido desde diferentes fuentes:

- 265 imágenes que contienen un total de 128 baches han sido recopilados desde grabaciones de Valeo para el proyecto AUTOPILOT. Estas grabaciones han sido realizadas en diferentes fechas y bajo distintas condiciones climatológicas en comparación con las que se han utilizado para entrenamiento. De hecho, estas imágenes han sido seleccionadas intencionadamente dado que contienen escenas aparentemente complicadas. Las imágenes contienen túneles, condiciones meteorológicas adversas, atascos de tráfico y carreteras con resaltos, manchas en el asfalto y alcantarillas. Estas últimas son candidatas potenciales a ser clasificadas como bache, es decir, a ser falsos positivos.
- La segunda fuente es un vídeo de YouTube[2], grabado desde una cámara en el interior del automóvil conduciendo a través de un tramo de carretera en el área de Willamette National Forest, Oregon, EE. UU. De ahí se han recopilado 482 imágenes y se han etiquetado 475 baches.
- Se han extraído algunas imágenes del conjunto de datos que se utiliza en [9]. En el trabajo de inspección de daños en la carretera que se presenta en [9], se utiliza un conjunto de datos de 9892 de imágenes capturadas con smartphones. Una de las clases etiquetadas en el conjunto de datos toma el nombre de "resaltos, grietas y baches". Se han revisado todas las imágenes que contienen objetos de dicha clase y se han seleccionado 54 imágenes. Dado que las etiquetas que aparecían no eran excesivamente precisas y rigurosas, algunas de ellas se han modificado. En las 54 imágenes se han etiquetado 62 baches.

6.2. Evaluación de modelos de redes profundas para la detección de baches

En este apartado se evaluan tres redes de detección de objetos Faster-RCNN (Sección 3.7). Estas tres redes, antes de ser entrenadas, fueron descargadas desde [28], y aunque las tres comparten la arquitectura Faster-RCNN, el módulo convolucional que se encarga de extraer las características es distinto en cada red. Los extractores de características son los siguientes:

- **Inception v2:** Es una red convolucional que realiza convoluciones con filtros de diferentes tamaños. Esto se hace debido a que el tamaño del objeto que se desea clasificar en una imagen varía mucho en función de la imagen. El Inception v2 [45] comparte la premisa de su predecesor el Inception v1 [54], pero consigue una implementación mucho más eficiente.
- **Resnet 101:** Es una red convolucional profunda (101 capas convolucionales) que utiliza conexiones residuales (Apartado 3.2.2) para hacer frente a problemas como el desvanecimiento de gradiente y la degradación [33].
- **Inception Resnet v2**: Esta red utiliza una estructura híbrida que combina el Inception y el Resnet. En [55] se muestra un mayor rendimiento de esta red frente a las dos anteriores.

Es conveniente aclarar que las redes evaluadas en este apartado fueron entrenadas partiendo de los pesos originales de las redes entrenadas con COCO, y se afinaron los parámetros para ajustarse al problema de la detección de baches. Para ello se modificó la estructura de la red (el número de clases bajó de 80 a una), y también la entrada de la red con el tamaño de imagen 1024×800. Una vez modificada la estructura, el entrenamiento parte desde cero en los casos en los que la estructura se ha modificado (capa de salida) y parte desde los pesos y sesgos preentrenados en el resto de capas.

Dado que esta sección del trabajo forma parte del estudio que se ha presentado en un artículo de investigación que se encuentra actualmente en fase de revisión, para preservar la confidencialidad, el resto de detalles del entrenamiento de las redes se ha omitido.

[2] `https://www.youtube.com/watch?v=BQo87tGRM74`

En [28] se muestran unos valores de referencia del mAP de COCO para cada modelo preentrenado. Estos valores se obtienen sobre un subconjunto de la base de datos de COCO:

Modelo	**COCO mAP(ˆ1)**
Faster-RCNN Inception v2	28
Faster-RCNN Resnet 101	32
Faster-RCNN Inception Resnet v2	37

Tabla 6.1: Valores del mAP de COCO para los modelos preentrenados de [28].

6.2.1. Resultados

Se han probado los tres modelos sobre el conjunto de datos de evaluación. Para calcular los valores de precisión y exhaustividad necesarios para ilustrar las curvas, el umbral de IoU más común para los sistemas de detección es de IoU=0.5. Dado que el tamaño de un bache en la imagen es muy pequeño en comparación con la escena, se ha considerado que la tarea principal es localizar el bache, y no tanto dibujar un cuadro delimitador muy preciso. Por lo tanto las detecciones con un solapamiento de IoU>0.4 se han considerado como suficientemente acertadas.

Se han dibujado seis curvas de precisión y exhaustividad. Se han dibujado tres, una por cada modelo, tomando primero el umbral de IoU estándar, es decir, 0.5 y después otras tres con el umbral fijado en 0.4 (Figura 6.3). Como puede observarse en el gráfico, el modelo Faster RCNN con el extractor Resnet 101 es el que mejor rendimiento muestra sobre los datos de evaluación, siendo incluso el área del umbral alto mayor a los áreas de los otros dos modelos con el umbral bajo.

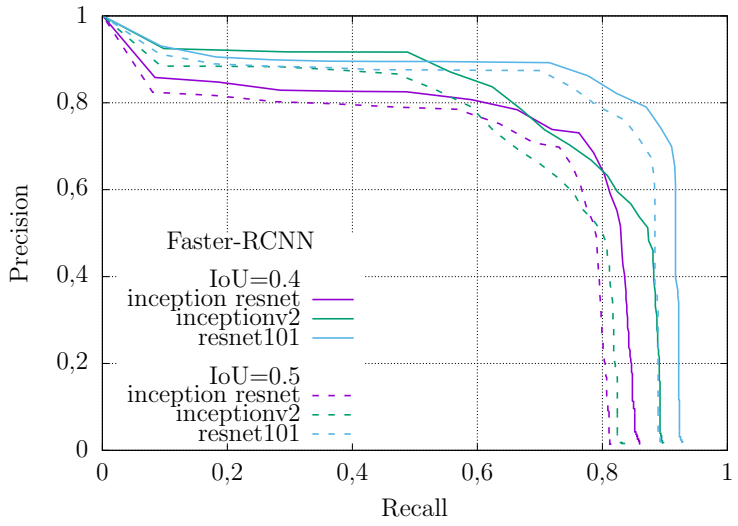


Figura 6.3: Curva de precison-exhaustividad de los tres modelos Faster-RCNN con diferentes extractores de características. Las líneas continuas representan el umbral de IoU=0.4 y las discontinuas IoU=0.5.

Los valores de la precisión media, que en este caso, puesto que únicamente existe una clase, son iguales al promedio de la precisión media, se muestran en la siguiente tabla:

		mAP(^1)	
IoU	**Modelo**	**PASCAL**	**COCO**
0.5	Inception v2	66.05	27.51
	Resnet 101	**77.49**	**31.12**
	Inception Resnet v2 atrous	68.71	26.67
0.4	Inception v2	75.45	—
	Resnet 101	**82.02**	—
	Inception Resnet v2 atrous	69.72	—

Tabla 6.2: Valores mAP de los diferentes modelos Faster-RCNN.

Efectivamente el mAP tanto de PASCAL como de COCO alcanza sus mayores valores en los modelos que utilizan el Resnet101 como extractor de características. En la siguiente tabla se muestra el tiempo medio que emplea el modelo para procesar una imagen.

	Tiempo de inferencia (ms)	
Modelos	**Tesla P100M 16GB**	**DrivePX2 TegraA**
Inception v2	53.2	177.1
Resnet 101	94.2	432.7
Inception Resnet v2 atrous	172.1	732.9

Tabla 6.3: Tiempo medio de inferencia de una imagen en distintos dispositivos de los tres modelos Faster-RCNN.

El modelo más rápido es el Inception v2, algo que concuerda con los tiempos de referencia que se ofrecen en [28]. Es una red más simple que las otras dos, por lo tanto es de esperar que el tiempo de inferencia sea menor. Por su parte, el Resnet 101, el que mejor rendimiento de detección ofrece, está más cerca del más rápido que del más lento en ambos dispositivos.

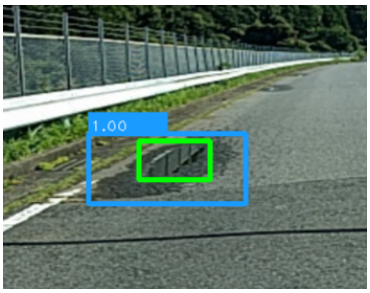
En la Figura 6.4 se muestran tres diferentes escenas, de las tres diferentes fuentes del conjunto de datos. Estas escenas han sido procesadas por el modelo de detección, proporcionando este unas detecciones correctas.

Figura 6.4: Imágenes de toda la escena en detecciones del modelo Faster-RCNN Resnet 101. El cuadro delimitador verde representa el *ground-truth* y el cuadro delimitador azul la detección. A la izquierda una imagen de [9], en el centro una imagen de Valeo, y a la derecha una imagen del vídeo de Youtube.

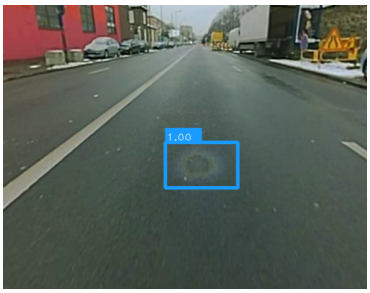
Figura 6.5: Imágenes del cuadro delimitador en buenas detecciones del Faster-RCNN Resnet 101.

En la Figura 6.5 se puede observar una serie recortes en algunas detecciones del modelo en las cuales el cuadro delimitador de la detección se acerca considerablemente al de la etiqueta. Por último, en la siguiente figura se muestran varias imágenes en las cuales el modelo ha fallado por diversos motivos, ya sean falsos positivos o falsos negativos.

(a) Charco a una distancia muy lejana.

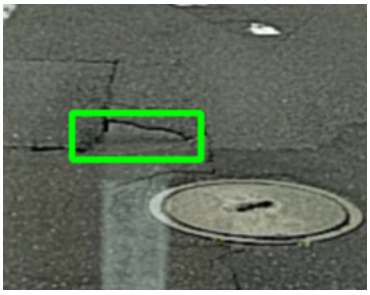


(b) Pequeña alcantarilla ligeramente hundida en el asfalto.

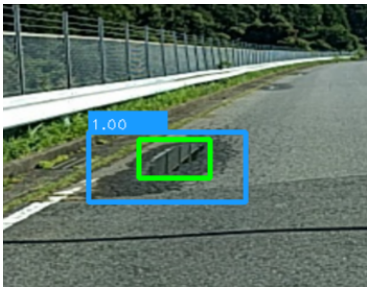


(c) Mancha de aceite con forma parecida a un bache.

(d) Alcantarilla con borde muy pronunciado.



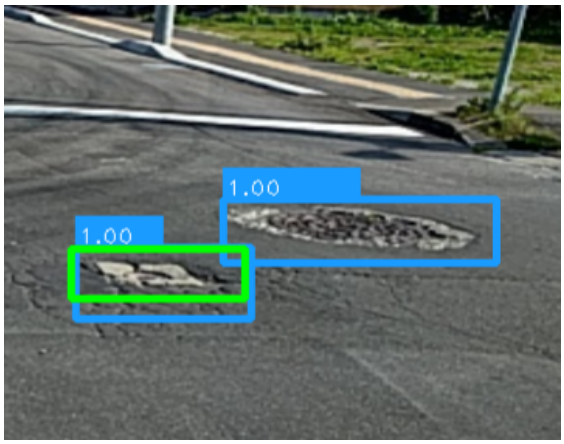
(e) Bache no detectado.



(f) Bache detectado pero con un cuadro delimitador muy grande por la humedad del asfalto.

Figura 6.6: Detecciones erróneas del modelo Faster-RCNN Resnet 101.

Una de las mayores causas de falsos positivos es la apariencia de alcantarillas. Al revisar los resultados se ha observado que las alcantarillas que muestran un borde muy pronunciado o un hundimiento en el asfalto son más propensas a ser clasificadas como bache por los tres modelos entrenados. Sin embargo, el modelo evita las alcantarillas más planas. En la Figura 6.6, las subfiguras (b), (d) y la (e) muestran ejemplos de este fenómeno. En la Figura 6.7 puede observarse de manera más clara.



(a) Falso positivo en alcantarilla con borde muy pronunciado y hundimiento.

(b) Verdadero negativo en alcantarilla sin hundimiento.

Figura 6.7: Ejemplos de detecciones en alcantarillas.

6.2.2. Discusión de los resultados

Como puede observarse en la Tabla 6.2 y el la gráfica de la Figura 6.3, el modelo que mejores resultados de detección ofrece es el Faster-RCNN Resnet 101. Aunque no es el modelo más lento, los 432.7 ms que tarda en procesar una imagen (aproximadamente dos imágenes por segundo) en el Drive PX2 pueden resultar excesivos para intentar una detección en tiempo real. Sin embargo, estas redes han sido entrenadas para enviar información a plataformas IoT, con la intención de enviar la información a centros de control y que después esta se utilice para advertir a otros conductores de posibles desperfectos en la vía. Para este cometido es un tiempo aceptable, así que podría decirse que es el candidato más valido, dado que su rendimiento es alto y su tiempo de procesamiento es suficientemente bueno para el cometido que se desea desempeñar.

En el caso del Faster-RCNN Inception Resnet v2, puede resultar sorprendente que la red más compleja, y la que mayores valores de rendimiento presenta en la referencia [28] sea la que peores valores obtiene en prácticamente todos los casos. Esto puede deberse a que, puesto que es la red con más parámetros libres, es la red que mayor riesgo de sobreajuste tiene, y es probable que se haya sobreajustado al conjunto de datos de entrenamiento. Para poder entrenar esta red de manera más eficiente es probable que se necesite una cantidad de datos considerablemente mayor. Además, se observa un incremento en el valor del mAP cuando el IoU=0.5. Esto quiere decir que obtiene más falsos negativos y más falsos positivos que no se solapan correctamente al *ground-truth*.

Por su parte, la red Faster-RCNN Inception v2 obtiene valores discretos de mAP, dado que es una red más simple que la que utiliza el Resnet 101. Aún así, el tiempo de inferencia en el Drive PX2 permite procesar vídeo a aproximadamente seis imágenes por segundo. Esto hace que sea una red valida para enviar los datos a la IoT, y estaría cerca de servir como detector en tiempo real, ya que el sistema es capaz de detectar baches a una larga distancia.

Por otra parte, comparando el tiempo de procesamiento de los modelos de referencia (Tabla 5.1) y los modelos de detección de baches (Tabla 6.3) en el Drive PX2, el que utiliza el módulo convolucional Resnet 101 es el único que es más lento habiendo sido entrenado específicamente para detectar baches. Aunque la entrada de la red sea mayor para el caso del detector de baches, el número máximo de propuestas de región es tres veces menor. La diferencia de tiempo en los modelos Inception v2 e Inception Resnet v2 es considerable, esto es indicativo de que aunque una entrada mayor ralentiza el modelo, bajar el número de propuestas puede acelerar el tiempo de procesamiento del el modelo considerablemente.

Por último, para comparar los resultados obtenidos en las redes de detección de baches que se evalúan en este apartado, con los que se presentan en [10](Tabla 2.1), los valores de precisión y exhaustividad en los puntos de mayor rendimiento de la curva que se muestra en la Figura 6.3 para IoU=0.4, se muestran en la Tabla 6.4.

Modelo	**Precisión (%)**	**Exhaustividad (%)**
Faster-RCNN Inception v2	74.79	70.26
Faster-RCNN Resnet 101	**82.27**	**82.16**
Faster-RCNN Inception Renset	76.5	73.07

Tabla 6.4: Valores óptimos de precisión y exhaustividad de los modelos de detección de baches para el IoU=0.4.

Los valores que obtiene el modelo Faster-RCNN Resnet 101 superan a los que aparecen en la Tabla 2.1. Además, la base de datos de evaluación utilizada en [10] es considerablemente inferior en tamaño a la que se ha probado en los experimentos de este capítulo. No solo inferior en tamaño, si no que es de una única fuente, el modelo que se ha probado en éste capítulo ha sido evaluado en una base de datos obtenida desde tres fuentes diferentes. Por lo tanto, la diferencia de rendimiento que hay entre los dos sistemas de detección puede ser aún más grande de lo que muestran los resultados de las Tablas 2.1 y 6.4.

Capítulo 7

Conclusiones

En primer lugar, en este trabajo se han estudiado las redes neuronales desde los aspectos más básicos hasta los detectores de objetos en imágenes. Se ha explicado tanto su estructura, como los algoritmos utilizados para el entrenamiento de sus parámetros.

En segundo lugar, se ha estudiado el fenómeno de la transferencia del aprendizaje, entrenando tres redes Faster-RCNN Resnet partiendo de tres inicializaciones distintas para la detección de peatones. Entre los tres modelos entrenados el que peores resultados obtiene es el que parte de una inicialización aleatoria. Los modelos preentrenados con COCO y KITTI obtienen resultados muy similares, siendo el modelo de COCO ligeramente superior en rendimiento. Además, los resultados de rendimiento de ambos modelos son similares a los que se obtienen en la clasificación de peatones fáciles de detectar del estado del arte.

Los modelos de detección de peatones que partían de COCO y KITTI no consiguen mejorar las prestaciones iniciales a lo largo del entrenamiento. Esto puede ser debido a que la base de datos de entrenamiento cuenta con pocos ejemplos y que los modelos se están sobreajustando ligeramente. Además, la velocidad de aprendizaje aplicada puede ser demasiado grande. Como la clase peatón existe tanto en KITTI como en COCO, los módulos convolucionales Resnet 101 de las redes, en su estados iniciales cuentan con buenos filtros aprendidos para la detección de personas. Aplicar una velocidad de aprendizaje de 0.0002 puede conllevar que esos pesos se muevan demasiado y que se distorsionen ligeramente esos filtros, teniendo como resultado un rendimiento inferior.

En tercer lugar, se ha generado una base de imágenes con alta varianza intraclase de varias fuentes con el objetivo de aprender modelos de detección de baches robustos y generales que pueden ayudar a la localización de este tipo de peligros en las carreteras. Para ello se han recopilado bases de datos libres que contienen baches etiquetados, y se han etiquetado manualmente 2491 imágenes.

Se han evaluado tres redes diferentes para la detección de baches en carretera. Se ha llegado a la conclusión de que la localización precisa de los baches es un desafío debido a los errores en el *ground-truth* durante la anotación debido a la naturaleza de los baches de la carretera. Por lo tanto, se ha relajado el índice de Jaccard a IoU $= 0.4$ para la evaluación. En consecuencia, el Faster-RCNN Resnet101 alcanza valores del mAP del 82 %, mientras que el Faster-RCNN Inception v2 toma un valor de mAP del 75 % a un costo de procesamiento más bajo. Este último, cuando ha sido probado en la plataforma Nvidia DrivePX2 Autochauffeur, puede funcionar a velocidades de entre 5-6 fotogramas por segundo. Se ha comparado el mejor modelo con el estado del arte y se ha demostrado un rendimiento de detección superior en un conjunto de datos más amplio.

En cuanto a los detectores de peatones, como trabajo futuro, podría probarse un afinado de los parámetros utilizando una velocidad de aprendizaje más baja, o incluso congelar los pesos del módulo convolucional y ajustar únicamente el resto. De esta manera podría observarse si el menor rendimiento de generalización viene asociado a un cambio demasiado grande en los parámetros preentrenados, o es consecuencia de utilizar pocos datos para el entrenamiento.

Otra medida que podría aplicarse es aumentar el tamaño del *mini-batch*, para así obtener una mejor aproximación al gradiente real, y reducir el posible ruido que se obtiene actualizando los

parámetros con un *mini-batch* pequeño.

En cuanto al detector de baches en carretera, aumentar la base de datos podría ayudar considerablemente a mejorar los ratios de detección y la capacidad de generalización de los modelos. Cuánta mayor variedad de escenarios aparezca en el conjunto de datos de entrenamiento, más probable será que el sistema aprenda a detectar bien la posición y a diferenciar lo que es un bache de lo que no es un bache.

En la detección de baches, al ser una aplicación que puede ser de mucha utilidad si se consigue la detección en tiempo real, puede resultar conveniente optimizar la estructura de la red para que se adapte a la arquitectura ARM del Drive PX2 y así el tiempo de inferencia sea menor del que es ahora.

Por último, se han probado todos los modelos en la plataforma Drive PX2 de Nvidia. Aunque es un dispositivo de altas prestaciones, y amplio número de posibilidades, la potencia de cómputo está considerablemente por debajo de una computadora de alto rendimiento como las que se encuentran en los racks de cómputo. Las computadoras de alto rendimiento no se pueden instalar en los vehículos por problemas de refrigeramiento, consumo y espacio, por lo tanto el Drive PX 2 es una de las alternativas más potentes hoy en día. Aún así, sus limitaciones no permiten que los modelos de deep learning probados en este trabajo sean capaces de proporcionar detecciones en tiempo real, ya que no es capaz de procesar las imágenes suficientemente rápido.

Bibliografía

[1] Joel Janai, Fatma Güney, Aseem Behl, and Andreas Geiger. Computer vision for autonomous vehicles: Problems, datasets and state-of-the-art. *arXiv preprint arXiv:1704.05519*, 2017.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[3] AUTOPILOT H2020 project. `http://autopilot-project.eu/`. *Autopilot project*, Última visita 03/09/2018.

[4] Ernst D. Dickmanns and Birger D. Mysliwetz. Recursive 3-d road and relative ego-state recognition. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (2):199–213, 1992.

[5] Ernst Dieter Dickmanns and Volker Graefe. Dynamic monocular machine vision. *Machine vision and applications*, 1(4):223–240, 1988.

[6] Charles Thorpe, Martial H Hebert, Takeo Kanade, and Steven A Shafer. Vision and navigation for the carnegie-mellon navlab. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(3):362–373, 1988.

[7] Massimo Bertozzi, Alberto Broggi, and Alessandra Fascioli. Vision-based intelligent vehicles: State of the art and perspectives. *Robotics and Autonomous systems*, 32(1):1–16, 2000.

[8] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[9] Hiroya Maeda, Yoshihide Sekimoto, Toshikazu Seto, Takehiro Kashiyama, and Hiroshi Omata. Road damage detection using deep neural networks with images captured through a smartphone. *arXiv preprint arXiv:1801.09454*, 2018.

[10] S Nienaber, Marthinus J Booysen, and RS Kroon. Detecting potholes using simple image processing techniques and real-world footage. *IEEE Symposium Series on Computational Intelligence*, 2015.

[11] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1627–1645, 2010.

[12] J Javier Yebes, Luis M Bergasa, and Miguel García-Garrido. Visual object recognition with 3d-aware features in kitti urban scenes. *Sensors*, 15(4):9228–9250, 2015.

[13] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *IEEE CVPR*, volume 4, 2017.

[14] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3354–3361. IEEE, 2012.

[15] Piotr Dollar, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: An evaluation of the state of the art. *IEEE transactions on pattern analysis and machine intelligence*, 34(4):743–761, 2012.

[16] Pierre Sermanet, Koray Kavukcuoglu, Soumith Chintala, and Yann LeCun. Pedestrian detection with unsupervised multi-stage feature learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3626–3633, 2013.

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

[18] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *European conference on computer vision*, pages 346–361. Springer, 2014.

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

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

[22] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3d object proposals using stereo imagery for accurate object class detection. *IEEE transactions on pattern analysis and machine intelligence*, 40(5):1259–1272, 2018.

[23] Fan Yang, Wongun Choi, and Yuanqing Lin. Exploit all the layers: Fast and accurate cnn object detector with scale dependent pooling and cascaded rejection classifiers. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2129–2137, 2016.

[24] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Andrew G Berneshawi, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3d object proposals for accurate object class detection. In *Advances in Neural Information Processing Systems*, pages 424–432, 2015.

[25] Zhaowei Cai, Quanfu Fan, Rogerio S Feris, and Nuno Vasconcelos. A unified multi-scale deep convolutional neural network for fast object detection. In *European Conference on Computer Vision*, pages 354–370. Springer, 2016.

[26] Yu Xiang, Wongun Choi, Yuanqing Lin, and Silvio Savarese. Subcategory-aware convolutional neural networks for object proposals and detection. In *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*, pages 924–933. IEEE, 2017.

[27] Yu Xiang, Wongun Choi, Yuanqing Lin, and Silvio Savarese. Data-driven 3d voxel patterns for object category recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1903–1911, 2015.

[28] Tensorflow object detection model zoo. `https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md`. *Tensorflow, Google inc.*, Última visita 22/08/2018.

[29] Michael A Nielsen. Neural networks and deep learning. *URL: http://neuralnetworksanddeeplearning. com/.(visited: 19. 7. 2018)*, 2015.

[30] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[31] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[32] David Kriesel. *A brief Introduction on Neural Networks*. Citeseer, 2007.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[34] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[35] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. Snapshot ensembles: Train 1, get m for free. *arXiv preprint arXiv:1704.00109*, 2017.

[36] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[37] Ilya Loshchilov and Frank Hutter. Sgdr: stochastic gradient descent with restarts. *Learning*, 10:3, 2016.

[38] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[39] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[40] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

[41] Andrew R Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945, 1993.

[42] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[43] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[44] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[45] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[46] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[47] Y-T Zhou, Rama Chellappa, Aseem Vaid, and B Keith Jenkins. Image restoration using a neural network. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(7):1141–1151, 1988.

[48] Paul Jaccard. The distribution of the flora in the alpine zone. 1. *New phytologist*, 11(2):37–50, 1912.

[49] Lorien Y Pratt. Discriminability-based transfer between neural networks. In *Advances in neural information processing systems*, pages 204–211, 1993.

[50] A. Ess, B. Leibe, K. Schindler, , and L. van Gool. A mobile vision system for robust multi-person tracking. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'08)*. IEEE Press, June 2008.

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

[52] S Nienaber, RS Kroon, and Marthinus J Booysen. A comparison of low-cost monocular vision techniques for pothole distance estimation. In *Computational Intelligence, 2015 IEEE Symposium Series on*, pages 419–426. IEEE, 2015.

[53] Tzutalin. Labelimg is a graphical image annotation tool that labels object bounding boxes in images. *https://github.com/tzutalin/labelImg*, Última visita 22/08/2018.

[54] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[55] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017.