UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR

INGENIEROS DE TELECOMUNICACIÓN

TRABAJO DE FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN:

Reconocimiento eficiente de caras mediante *Deep Learning* a partir de imágenes en el espectro visible

Autor:

D. Mario Pérez Rodríguez

Tutores:

Dr. D. Juan Pablo Casaseca de la Higuera
Dr. D. Javier Manuel Aguiar Pérez

Valladolid, Julio 2021

TÍTULO:	**Reconocimiento eficiente de caras mediante *Deep Learning* a partir de imágenes en el espectro visible**
AUTOR:	**D. Mario Pérez Rodríguez**
TUTORES:	**Dr. D. Juan Pablo Casaseca de la Higuera** **Dr. D. Javier Manuel Aguiar Pérez**
DEPARTAMENTO:	**Departamento de Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

Tribunal

PRESIDENTE:	**Dr. D. Javier Manuel Aguiar Pérez**
VOCAL:	**Dr. D. Pablo Casaseca de la Higuera**
SECRETARIO:	**Dr. D. F. J. Simmross Wattenberg**
FECHA:	**Julio 2021**
CALIFICACIÓN:	

Agradecimientos

Quiero agradecer en gran medida a mis tutores de TFG, a Juan Pablo Casaseca de la Higuera y a Javier Manuel Aguiar Pérez por su ayuda a lo largo del desarrollo de este trabajo de fin de grado, por su constancia y apoyo y por ponerme los pies en la tierra, ya que sin ellos, no podría haber desarrollado semejante trabajo.

A un gran amigo, David Arévalo, por permitirme utilizar su plantilla de LaTeXpara realizar este trabajo y también a mi mejor amigo, Diego Ruíz, por su ayuda en la explicación de las funciones principales de este editor de texto así como la resolución de dudas que surgieron por el camino.

Gracias a Federico Simmross Wattenberg por su ayuda en la conexión con los servidores de tanis y tebas de la UVa, desde los que se llevó a cabo el 90 % del proyecto, y su posterior apoyo cuando surgían problemas con las redes.

Muchas gracias también a Javier del Pozo por su apoyo en la realización del TFG, no le conocía previamente, ni me debía nada e hizo todo lo posible por ayudarme sin pedir nada a cambio o sin que pareciera que se cansara de responder.

Quiero agradecer a todos mis amigos por estar ahí, a los de toda la vida y a los nuevos, a los que pensé que desaparecerían pero lucharon por quedarse y a todos los que alguna vez pusieron un hombro sobre el que pudiera llorar, habéis sido mi salvación en muchas ocasiones y quizás ni lo sabíais. Gracias a todos.

A mi novia, Elsa, que ha sido mi apoyo todo este tiempo, me ha dado los ánimos y el cariño que varias veces he necesitado y que siempre ha estado ahí aguantando y dándome todo su amor. Muchas gracias cariño.

Por último, y con mayor importancia, me gustaría agradecer de corazón a mi familia por todo su apoyo emocional y personal, gracias a ellos soy quien soy hoy día y me han permitido alcanzar todas mis metas. A mi hermano Pablo por ser el sol que ha iluminado mi vida en tiempos de oscuridad y penumbra, a mi madre Cayetana por darme la alegría de ver cada día una maravilla diferente y enseñarme lo que es la bondad y la alegría y a mi padre Manuel por ser mi pilar y enseñarme y dirigirme a ser como soy hoy. Muchas gracias desde lo más profundo de mi ser.

Resumen

La detección facial está presente en múltiples ámbitos de la vida cotidiana hoy en día, ya sea para desbloquear nuestro teléfono móvil o como medida de seguridad en una empresa.

Ahondando en esta idea, el objetivo de este Proyecto Fin de Carrera es estudiar el rendimiento de los detectores genéricos y específicos para un reconocimiento facial eficiente. Para ello, primero se entrenaron los detectores genéricos de forma específica para utilizarlos como detectores faciales y se utilizó la misma base de datos para los específicos. Después, los modelos se simplificaron utilizando herramientas estándar para su posterior análisis y evaluación. Se evaluó el rendimiento tanto en términos de velocidad (medida en fotogramas por segundo) como de precisión (utilizando la precisión media).

A la vista de los resultados, tanto los modelos genéricos como los específicos pueden utilizarse para la detección facial, si bien los diseñados específicamente dan lugar a un mejor equilibrio entre precisión y eficacia.

Palabras Clave

Aprendizaje Profundo, detección facial, análisis comparativo, mean Average Precision, fotogramas por segundo, TensorFlow.

Abstract

Nowadays, facial detection is present in multiple aspects of daily life, whether it is for unlocking our phones or as a company's security measurement.

Delving into this idea, the aim of this Final Year Project is to study the performance of both generic and specific detectors for efficient facial recognition. To this end, the generic ones were first specifically trained to use them as facial detectors and the same database was used for the specific ones. Afterwards, the models were simplified using standard tools for subsequent analysis and evaluation. Performance in terms of both speed (measured in frames per second) and accuracy (using average precision) was assessed.

In view of the results, both generic and specific models can be used for facial detection, although specifically designed result in a better trade off between accuracy and efficiency.

Keywords

Deep Learning, face detection, comparative analysis, mean Average Precision, frames per second, TensorFlow.

Índice general

1. Introducción **1**

1.1. Contexto y Motivación 1

1.2. Objetivos 3

1.3. Fases y Procedimiento 4

1.4. Organización de la Memoria 5

1.5. Medios materiales empleados 6

1.5.1. Hardware 6

1.5.2. Software y lenguajes de programación 6

2. Revisión del Estado del Arte **8**

2.1. Historia del Machine Learning y su aplicación en la detección de objetos 8

2.2. Aprendizaje Profundo y su aplicación en detección de objetos 14

2.2.1. Convolutional Neural Networks 16

2.2.2. Backbone 19

2.2.3. Feature Pyramid Networks 21

2.2.4. ReLU 22

2.3. Historia y Estado del Arte de la Detección de Objetos 22

2.3.1. Detectores tradicionales 23

2.3.2. Detectores de dos etapas basados en CNN 23

2.3.3. Detectores de una etapa basados en CNN 25

2.4. Historia y Estado del Arte de la Detección Facial 26

2.5. Ventajas e inconvenientes de la detección facial 29

3. Metodología de la comparativa **31**

3.1. Bases de Datos 31

3.1.1. WIDER FACE 31

3.1.2. PASCAL VOC 32

3.1.3. Microsoft COCO 33

3.2. Metodología de entrenamiento 33

3.2.1. Transfer Learning 33

3.2.2. Elección de los conjuntos de entrenamiento, validación y test. 34

3.3. Métricas utilizadas a lo largo de la comparativa 36

3.4. Explicación en detalle de los modelos de detección utilizados 40

3.4.1. Modelos genéricos de detección 40

3.4.2. Modelos específicos de detección 46

3.5. Herramientas de desarrollo 48

3.5.1. TensorFlow 48

3.5.2. TensorFlow Lite 49

3.5.3. DarkNet 50

3.6. Técnicas de simplificación 50

3.7. Metodología general 51

4. Comparativa de modelos de detección **53**

4.1. Justificación de la decisión de los modelos genéricos a comparar 53

4.2. Entrenamiento y comparativa de los modelos de detección genéricos 56

4.3. Comparativa de los modelos de detección específicos 61

4.4. Comparativa de los modelos de detección simplificados 64

5. Conclusiones **70**

5.1. Líneas futuras 72

6. Bibliografía **74**

Índice de figuras

1. Procedimiento de elaboración del TFG 5

2. Agrupación de la Inteligencia Artificial, el Machine Learning y el Deep Learning 8
3. Enfoque del Machine Learning con un programa de detección de spam 11
4. Ejemplo de aprendizaje supervisado 12
5. Ejemplo de aprendizaje sin supervisar 12
6. Ejemplo de aprendizaje semi supervisado 13
7. Ejemplo funcionamiento del aprendizaje reforzado 13
8. Procesado de una imagen por una red profunda, a partir de la cual se obtiene el resultado 15
9. Pasos básicos seguidos en el entrenamiento de un modelo de Deep Learning . . 16
10. Entrada y salida correspondientes tras aplicar un kernel con stride de 1 17
11. Entrada y salida correspondientes tras aplicar un kernel con stride de 2 17
12. Aplicación de un padding de ceros al tamaño de entrada original 18
13. Operación llevada a cabo en el max pooling 19
14. Operación llevada a cabo en el average pooling 19
15. Esquema básico del aprendizaje residual 20
16. Procesamiento de una imagen en los modelos MobileNet 21
17. Estructura de un extractor FPN 21
18. Diagrama temporal ordenado de los detectores de objetos unidos mediante flechas, indicando el orden de aparición de los modelos 26
19. Ejemplos de las partes de una cara del dataset de WIDER FACE 26
20. Historia de la detección de caras 27

21. Ejemplos del dataset de WIDER FACE 32
22. Ejemplos del dataset de PASCAL VOC 33
23. Ejemplos del dataset de Microsoft COCO 33
24. Ejemplo de Transfer Learning con congelación de pocas capas 34
25. Cálculo de la IoU 37
26. Tabla con los valores de precisión y recall obtenidos de una serie de imagenes procesadas 37
27. Curva precisión vs recall 38
28. Curva precisión vs recall alisada 38
29. Curva precisión vs recall con muestreo $p(r_i)$ 39
30. Esquema de detección del modelo EfficientDet 41
31. Esquema de detección del modelo SSD 42

32. Esquema de la arquitectura de Faster R-CNN 44
33. Esquema de la arquitectura de YOLOv1 45
34. Entrada y salida de la arquitectura principal de Faced 47
35. Entrada y salida de la arquitectura principal de Faced 47
36. Arquitectura del primer componente de un RFB 48

37. Diagrama de dispersión con los diferentes modelos del zoo de TF 54
38. Diagrama de dispersión con los datos teóricos de los modelos del Model Zoo de tamaño 640x640. 55
39. Diagrama de dispersión con los modelos del zoo de TF antes y después del entrenamiento 58
40. Detección de los diferentes modelos genéricos entrenados parte 1 59
40. Detección de los diferentes modelos genéricos entrenados parte 2 60
41. Detección de los diferentes modelos específicos entrenados 62
42. Diagrama de dispersión con los diferentes modelos entrenados. 63
43. Comparativa de los diferentes modelos antes de la simplificación y después de esta para las diferentes optimizaciones 68

Índice de tablas

1. Resultados de la comparativa entre modelos genéricos entrenados para detección de caras 57
2. Resultados de la comparativa para modelos sin simplificar 64
3. Tabla comparativa de la precisión de los modelos antes y después de su simplificación 66
4. Tabla comparativa de la velocidad y peso de los modelos antes y después de su simplificación 67

Capítulo 1

Introducción

Este capítulo tiene como objeto servir como una descripción del marco de trabajo y desarrollo de este Trabajo de Fin de Grado. A lo largo del mismo se expondrán los objetivos, métodos y resultados obtenidos a lo largo de la realización del proyecto.

1.1. Contexto y Motivación

La principal motivación para la elección y elaboración de este trabajo de Fin de Grado se sustenta en el interés como futuro profesional en conocer y profundizar en el funcionamiento de los detectores de objetos y la teoría que conforma el Deep Learning o aprendizaje profundo. Más concretamente, el funcionamiento y aplicación de detectores faciales para su posterior implementación en proyectos de electrónica permitiendo así una fusión de software y hardware. Este TFG supone por tanto la primera fase de la elaboración de un proyecto de aplicación al ámbito cotidiano.

La detección de caras en imágenes es objeto de un extenso estudio a día de hoy, ya no solo por el afán de descubrir cosas nuevas y avanzar en el aprendizaje, sino gracias a sus múltiples aplicaciones en el ámbito cotidiano y empresarial. Esta puede servir para mejorar el reconocimiento facial, cuyo objetivo es identificar automáticamente a una persona por sus rasgos faciales. Puede ser utilizada con múltiples objetivos también, ya sea en empresas de seguridad con objeto de localizar criminales y encontrar gente desaparecida, o permitir el acceso solo a aquellas personas autorizadas en establecimientos gubernamentales. Igualmente pueden servir como factor extra de autenticación a la hora de iniciar sesión en cualquier aplicación o en accesos a servicios en línea. Y en el futuro podría servir como método de pago en tiendas online y físicas, como control de acceso en instalaciones, evitar robos y suplantaciones de identidad, etc.

Hasta hace unos años, la detección facial se había llevado a cabo con el algoritmo de Viola & Jones [1], el cual, creado en el año 2001, permitía la detección de imágenes trabajando exclusivamente con imágenes en escala de grises para después aplicar la localización de la cara detectada en la imagen coloreada. Previo a este, el detector Eigenfaces fue un hito gracias a sus increíbles resultados, inigualados hasta la fecha. 2014 fue un año que marcó un antes y un después en la detección en general, ya fuera detección de objetos o facial, gracias a la aplicación del Deep Learning en múltiples algoritmos.

En 2014 se llevó a cabo el primer y mayor salto hacia adelante en la utilización del Deep Learning para la detección facial, consiguiendo resultados que estaban casi a la altura del ojo humano, conseguido en el sistema DeepFace de Yi Sun, et al. [2]. Hasta la fecha y con la aplicación de las redes convolucionales profundas, se podrían clasificar los métodos de detección de caras en cuatro tipos [3]:

- Métodos basados en la apariencia: Los modelos son aprendidos a partir de un set de imágenes de entrenamiento el cual debería capturar la variabilidad representativa del aspecto de las caras. Este tipo de métodos son usados en su mayoría para la detección facial.
- Métodos basados en el conocimiento: Estos métodos eran muy típicos antes de la aparición del Deep Learning y las GPUs debido a que, por la falta de capacidad computacional, se trataba de métodos basados en la escritura manual de reglas que englobaban el conocimiento humano de las partes que constituyen una cara típica. Son diseñados actualmente para localización de las partes de una cara.
- Métodos de coincidencias con plantillas: Se almacenan varios patrones de caras para comparar los datos nuevos con los almacenados. Se computan las relaciones entre patrones guardados e imágenes de entrada para detectar la cara. Estos métodos se utilizan mayoritariamente en localización y detección de caras.
- Aproximaciones de características invariantes: Estos algoritmos apuntan a encontrar características estructurales que existen incluso cuando la pose, el punto de vista y las condiciones de luz varían. Estos métodos son utilizados para localización facial.

En mayor o menor medida, estos cuatro métodos nombrados son los seguidos por la mayoría de detectores faciales que hacen uso de las herramientas de Deep Learning.

Además de que el Deep Learning tiene cuantiosos aspectos positivos, es innegable que posee ciertos aspectos negativos. Lógicamente, la disminución de la velocidad de ejecución a la hora de la detección está entre ellos, esto es debido a que, si para que una imagen pueda ser procesada se le deben aplicar multitud de procesos y operaciones matemáticas, el tiempo necesario para que un modelo pueda llevar a cabo estas operaciones, será superior a uno que no necesite hacerlo. Como se explica en [4], a pesar de reaccionar adecuadamente en imágenes similares a las utilizadas durante el entrenamiento, variaciones del contraste, brillo, adición de

borrosidad o ruidos gaussianos y salt&pepper y la compresión JPG, suelen hacer que un modelo que funcionaba muy bien para un cierto tipo de imágenes, de repente no sea tan viable. Este es un factor a tener en cuenta ya que el entrenamiento de modelos puede ser llevado a cabo con un set de imágenes "cotidiano", pero si el dispositivo utilizado capta un tipo de imágenes menos nítidas, el desempeño puede verse severamente empeorado en comparación.

Teniendo en cuenta todo lo anterior, la idea del proyecto que se desarrolla, se enfoca entre otras cuestiones a la realización de una comparativa sobre los diferentes modelos de detección de objetos genéricos tras su entrenamiento para la detección de caras. Esta comparativa tiene como objetivo llegar a un resultado final, a partir del cual poder obtener varios modelos de entre los elegidos, los cuales serán, supuestamente, los modelo más rápidos y precisos o una fusión equitativa de ambos parámetros. El objeto es concluir cuales son los modelos más eficientes a la hora de llevar a cabo la detección facial y comprobar la viabilidad de los detectores en un trabajo concreto.

1.2. Objetivos

El presente TFG parte del hecho de que los detectores de objetos basado en DL habituales necesitan una cantidad masiva de operaciones computacionales y de características del modelo para poder clasificar cada caja delimitadora a su clase apropiada. Esto se traduce en una gran cantidad de parámetros a manejar, muchísimos filtros y una gran cantidad de capas. Es decir, son redes grandes que pueden ser entrenadas con la finalidad de obtener mayor precisión en la detección.

Desde esta perspectiva, el objetivo general de este proyecto es estudiar la viabilidad de detectores de objetos genéricos basados en Deep Learning para el reconocimiento facial eficiente mediante su simplificación y entrenamiento y comparar con detectores desarrollados de manera específica para esta tarea.

Este objetivo general se desglosa en 3 objetivos específicos:

- Entrenar modelos genéricos de detección de objetos para su utilización como detectores de caras.
- Comparar los detectores de objetos entrenados para la detección de caras y modelos específicos de detección facial.
- Comprobar la mejoría de estos modelos tras su simplificación mediante la herramienta TensorFlow Lite.
- Determinar cuál es la selección óptima sobre la base del estudio.

1.3. Fases y Procedimiento

El presente proyecto se divide en dos fases: Fase preparatoria e implementación del proyecto.

1. Fase preparatoria: Previa a la investigación como tal, se realiza un estudio bibliográfico sobre el tema al objeto de recopilar los conceptos necesarios para la posterior realización del mismo. Se profundiza en los siguientes conceptos claves: Inteligencia Artificial, Machine Learning y Deep Learning. Posteriormente se lleva a cabo la adaptación y creación del entorno de trabajo necesario para la puesta en marcha del proyecto.
2. Fase de implementación del proyecto: Abarca, a grandes rasgos, la comparativa entre detectores de caras optimizados y sin optimizar, siendo estos últimos, los primeros a comprobar.

   La fase de la comparativa de detectores de caras sin optimizar, se subdivide a su vez en dos apartados:

   - Comparativa entre modelos de detección de objetos genéricos entrenados para la detección de caras. Estos modelos son los ofrecidos por la API (interfaz de programación de aplicaciones) de TensorFlow [5] y el detector de objetos del estado del arte, YOLO [6].
   - Comparativa de modelos específicos de detección facial. De todas las posibilidades en el mercado, se han escogido dos modelos que destacan por su pequeño tamaño y velocidad de inferencia en imágenes y vídeos.

Por último se llevará a cabo la simplificación de los modelos de detección utilizados, ya sean genéricos o específicos, mediante las herramientas ofrecidas por TensorFlow Lite para su posterior análisis, evaluación y comparación.

Para la consecución de los objetivos del proyecto se pone en marcha el siguiente procedimiento a partir de la revisión de la literatura y el estudio del arte tanto de la detección de objetos como la detección facial.

- Comparación de los modelos de la API de TensorFlow [7] con los valores existentes en el Model-Zoo [5].
- Entrenamiento de estos modelos genéricos para su mejor desempeño en la tarea de la mejor detección del contorno de una cara empleando para ello el dataset de imágenes disponible en WIDER FACE [8].
- Adición de los detectores faciales específicos a la comparativa mediante su entrenamiento y evaluación.

- Por último, se procede a la simplificación de estos modelos con la herramienta de TensorFlow [7], TensorFlow Lite [9].

La figura 1 recoge de forma esquemática el proceso llevado a cabo en el presente documento.

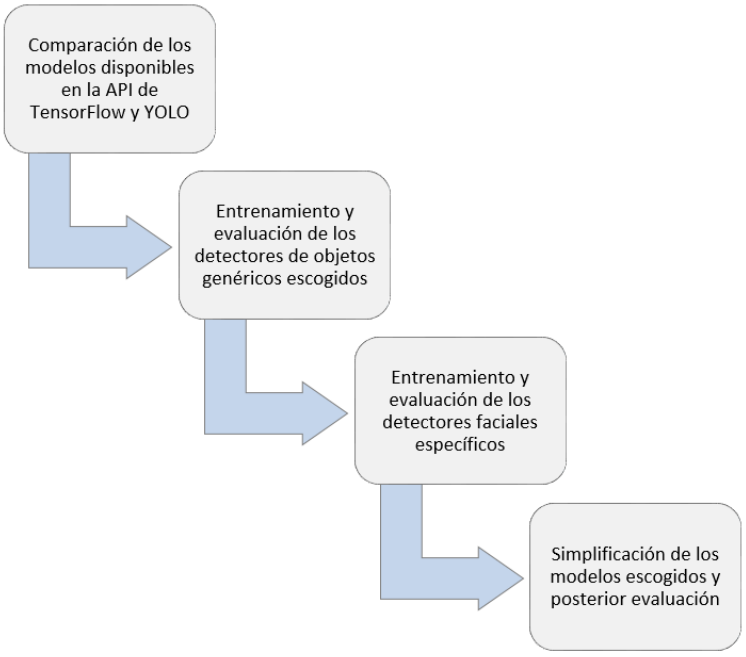


Figura 1: *Procedimiento de elaboración del TFG.*

1.4. Organización de la Memoria

La presente memoria se estructura en 5 capítulos:

El primer capítulo de la memoria tiene un carácter introductorio en el que se recoge el marco de trabajo. Se plantea la motivación para la búsqueda y selección de la temática a desarrollar, así como los objetivos y la metodología seguida a lo largo de todo el TFG.

En el segundo capítulo se aborda una revisión del Estado del Arte tanto en relación con la detección de objetos, a nivel genérico, como en la detección facial de manera más específica. En este capítulo se expone el estado actual de estas tecnologías, así como los avances más relevantes que se han ido viendo en ellas.

El tercer capítulo se centra en los requisitos previos a la comparativa de modelos de cara a conseguir los mejores resultados sin que lleguen a producirse problemas de memoria durante la fase de entrenamiento. Para esta comparativa, se utiliza un mismo modelo de detección con diferente cantidad de imágenes para el entrenamiento.

El cuarto capítulo contiene una descripción acerca del procedimiento realizado para la toma de decisiones de los modelos genéricos a comparar. En él se incluye una explicación en detalle de los modelos de detección utilizados, el entrenamiento llevado a cabo en cada uno de ellos y su simplificación. Al mismo tiempo se lleva a cabo una evaluación de los resultados obtenidos y una reflexión sobre estos objeto del presente TFG.

Por último, en el quinto capítulo se recogen las conclusiones tras el análisis de los resultados, así como una prospectiva de futuro.

1.5. Medios materiales empleados

Para la realización de este proyecto se han empleado los siguientes medios materiales:

1.5.1. Hardware

Servidor con distribución Ubuntu basada en Linux [10]:

- Procesadores: Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30GHz (x2)
- RAM: 512GB
- Tarjeta Gráfica 1: Quadro RTX 5000, 16GB
- Tarjeta Gráfica 2: GeForce GTX 1070, 8GB

1.5.2. Software y lenguajes de programación

- **Anaconda:** Administrador de paquetes gratuito, libre y de código abierto así como un administrador de entorno y distribución de Python. Está orientado a simplificar el despliegue y administración de los paquetes de software. [11].
- **Python:** Lenguaje de programación interpretado de alto nivel orientado a objetos con semántica dinámica. Es muy utilizado para el desarrollo web en la parte backend, el

desarrollo de software, aplicaciones matemáticas y especialmente en el campo de la inteligencia artificial gracias a las múltiples bibliotecas enfocadas a este cometido [12, 13].

- **TensorFlow:** "Plataforma de código abierto de extremo a extremo para el aprendizaje automático. Cuenta con un ecosistema integral y flexible de herramientas, bibliotecas y recursos de la comunidad que permite que los investigadores innoven con el aprendizaje automático y los desarrolladores creen e implementen aplicaciones fácilmente" [7].

- **TensorFlow Lite:** Conjunto de herramientas cuya finalidad es ayudar a los desarrolladores a ejecutar modelos de TensorFlow en dispositivos incorporados, móviles o de IoT (Internet of Things). Permite la inferencia de aprendizaje automático en dispositivos con una latencia baja y un tamaño de objeto binario pequeño [9].

- **Keras:** Biblioteca gratuita de Python de código abierto potente y fácil de usar para desarrollar y evaluar modelos de aprendizaje profundo. Engloba las bibliotecas TensorFlow y Theano [14].

- **Pycharm Community Version:** Entorno de desarrollo integrado (IDE) utilizado en programación [15], específicamente para el lenguaje Python. Según [16], en 2020 y según [17], en 2021, Pycharm fue el IDE más utilizado para Python.

- **MATLAB R2019a:** "Plataforma de programación diseñada específicamente para ingenieros y científicos para analizar y diseñar sistemas y productos que transformen nuestro mundo. El corazón de MATLAB es el lenguaje MATLAB, un lenguaje basado en matrices que permite la expresión más natural de las matemáticas computacionales" [18].

- **LaTeX:** Sistema de composición tipográfica de alta calidad; incluye características diseñadas para la producción de documentación técnica y científica. LaTeX es el estándar de facto para la comunicación y publicación de documentos científicos. [19]

- **Putty:** "Desarrollado originalmente por Simon Tatham para la plataforma Windows, PuTTY es un cliente SSH y telnet de código abierto". Es muy utilizado para hacer túneles SSH entre clientes y servidores y permite su utilización conjuntamente con herramientas gráficas [20].

- **TigerVNC:** Aplicación cliente/servidor que permite a los usuarios iniciar e interactuar con aplicaciones gráficas en máquinas remotas. TigerVNC, como implementación de alto rendimiento y plataforma neutral de VNC (Virtual Network Computing) proporciona los niveles de rendimiento necesarios para ejecutar aplicaciones 3D y de vídeo [21].

En este proyecto se ha empleado MATLAB para la generación de gráficas. El resto ha sido desarrollado en el IDE de Pycharm y codificado en Python. Se han utilizado además, las herramientas Putty y TigerVNC para llevar a cabo la conexión virtual con las máquinas disponibles del servidor de la escuela desde el que se ha llevado a cabo la mayoría de este trabajo.

Capítulo 2

Revisión del Estado del Arte

Este segundo capítulo de la memoria del Trabajo de fin de Grado está enfocado en el estudio y entendimiento de la IA, el Machine Learning y el Deep Learning, además de una breve revisión del Estado del Arte de la detección de objetos, es decir, en qué estado se encuentra actualmente esta tecnología y cuáles son los avances más recientes. Se incluye igualmente la historia de los detectores faciales y como han ido evolucionando desde sus comienzos hasta su estado actual. Por último se plantea una reflexión en torno a las potenciales ventajas e inconvenientes.

2.1. Historia del Machine Learning y su aplicación en la detección de objetos

La Inteligencia Artificial, como campo de estudio, engloba el campo del Machine Learning, el cual, a su vez engloba al Deep Learning. En la figura 2 se puede observar una representación de esta afirmación. Pero antes de entrar en materia conviene saber a qué nos referimos cuando hablamos de Machine Learning y para qué sirve.

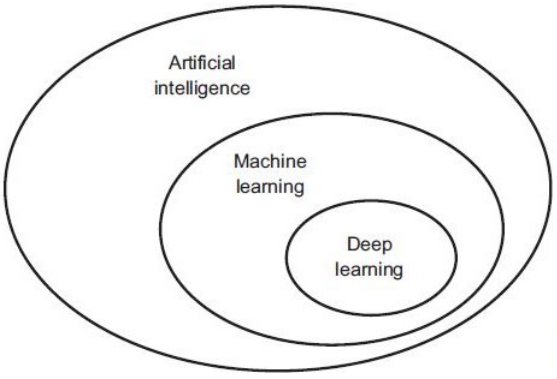


Figura 2: *Agrupación de la Inteligencia Artificial, el Machine Learning y el Deep Learning [22].*

Darle un cerebro artificial a un robot, hacerlo aprender a caminar o que aprenda a reconocer voces, crear un filtro anti-spam, mejorar los antivirus, vehículos autónomos, etc. Esas son solo algunas de las posibles aplicaciones que tiene el Machine Learning. A continuación se describen algunos hitos en la historia de esta rama de la inteligencia artificial.

En 1943, el matemático Walter Pitts y el neurofisiólogo Warren McCulloch propusieron una teoría en la que se pretendía analizar el cerebro como un organismo computacional y proponían, para ello, la creación de computadoras que funcionaran de manera igual o mejor que la red neuronal humana [23].

Años después, en 1950, Alan Turing creó el mundialmente conocido test de Turing, que se consideraba logrado si una máquina era capaz de engañar a un humano haciéndole creer que se encontraba delante de un humano en vez de un ordenador.

Arthur Samuel escribió el primer programa de aprendizaje informático a finales de 1952. Este consistía en un software con la capacidad de jugar a las damas que mejoraba su respuesta dependiendo del nivel de juego, volviéndose mejor tras cada juego.

Y en 1956, Martin Minsky, John McCarthy y otro grupo de profesionales acuñaron el nombre de Inteligencia Artificial en medio de una conferencia en Darthmouth.

Un par de años más tarde, en 1958, un psicólogo innovador conocido como Frank Rosenblatt diseñó la primera red neuronal artificial, a la que llamó “Perceptron” [24].

Finalmente, se acuñó el término de Machine Learning (ML) en 1967 gracias a la creación del algoritmo Nearest Neighbor [25], puesto que este algoritmo le brindaba a una máquina la capacidad de aprender patrones por primera vez.

Después de esto llegó una época de muy altas expectativas y pocos avances, hasta que en la segunda mitad de los años setenta apareció la primera época conocida como “El primer invierno de la Inteligencia Artificial”. Época que no terminó hasta 1979 cuando unos estudiantes de la universidad de Stanford inventaron el Stanford Cart, un robot capaz de desplazarse por una habitación sorteando los obstáculos que hubiera en ella.

Seguido del nacimiento de los sistemas basados en reglas, que generó gran interés en el ML, en 1981, se introdujo el concepto EBL (Explanation Based Learning) donde, a partir del análisis de datos de entrenamiento, un computador era capaz de crear reglas generales que permitían descartar datos menos importantes.

Y en 1985, el programa NetTalk del profesor Terry Sejnowski, que constituyó el último gran avance de la década, precedió al segundo AI Winter (Segundo invierno de la IA) que duró desde 1987 hasta 1993.

Dicha etapa, permitió al Machine Learning desvincularse como herramienta de la Inteligencia Artificial, lo que la impulsó al estatus de materia propia a finales de los años 90 [23]. Esta década obtuvo como hito la creación del ordenador Deep Blue, el cual, por primera vez en la historia, fue capaz de vencer al campeón mundial de ajedrez, Gary Kaspárov.

Una de las razones del segundo AI Winter, fue el abandono de la idea de entrenar una red neural profunda (Deep Learning), pues se consideraba una tarea imposible en 1990.

El ML quedó en varado hasta que en 2006, Geoffrey Hinton junto con un grupo de investigadores, publicaron un artículo mostrando como entrenar este tipo de redes, demostrando que volvía a ser posible continuar con los avances tecnológicos de este campo. Desde ese punto en adelante, la comunidad científica recuperó el interés y unos cuantos años después, el Machine Learning ya había conquistado la industria [26].

En la década de 2010 hubo cuantiosos avances en este campo, entre los que se pueden destacar:

- La creación del ordenador Watson de IBM. Cuya capacidad de aprendizaje, permitió su victoria ante varios competidores humanos en el concurso Jeopardy, el cual consistía en responder a preguntas en su lengua natural.
- El desarrollo del proyecto GoogleBrain de Jeff Dean y Andrew Ng, que consistía en una red neuronal la cual, por medio de Google, era capaz de detectar patrones en vídeos e imágenes.
- Desarrollo de DeepFace por Facebook, cuyo algoritmo basado en redes neuronales profundas, era capaz de reconocer personas con una precisión cercana a la del ojo humano.
- En 2014 Google compra DeepMind, una startup inglesa de Deep Learning que recientemente había demostrado las capacidades de las redes neuronales profundas con un algoritmo capaz de jugar a juegos de la consola Atari simplemente viendo los píxeles de la pantalla, tal y como lo haría una persona. Dicho algoritmo fue capaz de vencer a algunos jugadores expertos.
- El año 2015 vino acompañado de 3 avances importantes: El lanzamiento de Amazon de su propia plataforma de ML, el despliegue del "Distributed Machine Learning Toolkit" de Microsoft, cuya finalidad era compartir programas y problemas de machine learning entre equipos manera eficiente, y la fundación de una organización sin ánimo de lucro llamada Open AI. Esta compañía, creada por un grupo en el que se encontraban Elon Musk y Sam Altman, buscaba promover investigaciones que permitieran avances en el campo de la IA en pro de un impacto positivo en la humanidad.

Aparte de su enorme potencial, debemos preguntarnos, ¿Cómo funciona y por qué es tan importante el ML?

Según [26], "el Machine Learning es la ciencia o arte de programar ordenadores para que puedan aprender a partir de datos". Arthur Samuel lo definió como el campo de estudio que le da a las computadoras la posibilidad de aprender sin estar explícitamente programadas. Y según la IEEE SMC (Systems, Man and Cybernetics Society), "es el estudio de algoritmos computacionales que mejoran automáticamente con la experiencia. Sus aplicaciones abarcan desde programas de minería de datos que descubren reglas generales en grupos de datos grandes hasta sistemas de filtrado de información que aprenden sobre los gustos de los usuarios automáticamente" [27].

El enfoque tradicional para llevar a cabo un programa de, por ejemplo, filtrado de correos, sería analizar los errores primeramente, después, estudiar el problema, escribir una serie de reglas al respecto que el programa deberá seguir y por último, lanzar el programa. El problema es que al final, el programa constaría de un sinfín de reglas complejas, las cuales serían muy difíciles de controlar y mantener. Por el contrario, el enfoque del Machine Learning, como se puede ver en la figura 3, se basa en analizar los errores, estudiar el problema y, con nuevos datos, entrenar un algoritmo para que detecte automáticamente que palabras o frases son típicas en el spam y lanzar el programa. Este sistema es bastante más simple y fácil de mantener que el mencionado previamente.

A grandes rasgos, algunas aplicaciones del ML servirían para:

- Conseguir información y conocimiento de grandes cantidades de datos y problemas complejos.
- Problemas complejos para los cuales, el enfoque tradicional no aporta una solución viable.
- Problemas cuyas soluciones necesitan de mucho trabajo manual o una gran lista de reglas.
- Entornos que sufren de fluctuaciones habituales.

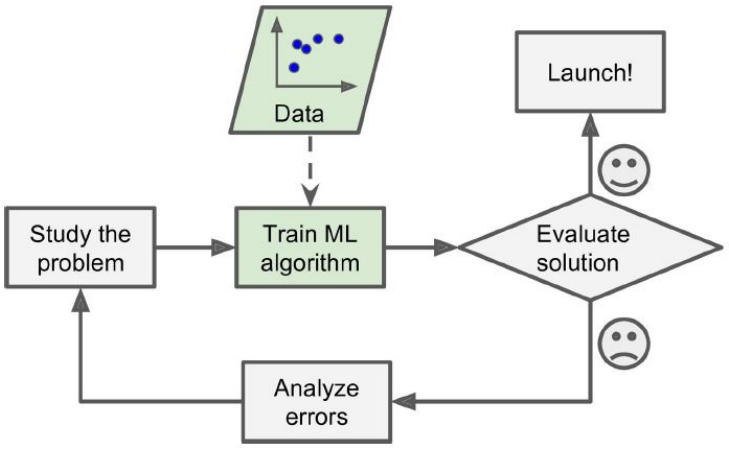


Figura 3: *Enfoque del Machine Learning con un programa de detección de spam [26].*

Dependiendo de la cantidad de supervisión necesaria, un sistema ML puede ser clasificado en sistemas supervisados, sin supervisión, semi supervisados y de aprendizaje reforzado.

- **Aprendizaje supervisado**: En el aprendizaje supervisado, los datos de entrenamiento entregados al algoritmo, incluyen al mismo tiempo las soluciones de este, y se conocen como labels. Cuando las labels vienen de un set finito, se llama clasificación, mientras que si estas proceden de un set de valores continuos, se conoce como regresión. Un ejemplo de regresión puede ser el análisis del tiempo, que toma, además de la información actual, patrones repetidos a lo largo de la historia para predecir cómo va a hacer [26].

  Para este Trabajo de Fin de Grado, se ha utilizado este tipo de aprendizaje, más concretamente, la clasificación y regresión. En la figura 4 se puede observar un ejemplo de cómo, a partir del aprendizaje supervisado, se puede crear un filtro de correos electrónicos a partir del cual, la máquina en cuestión será capaz de discernir entre correos electrónicos deseados o no deseados.

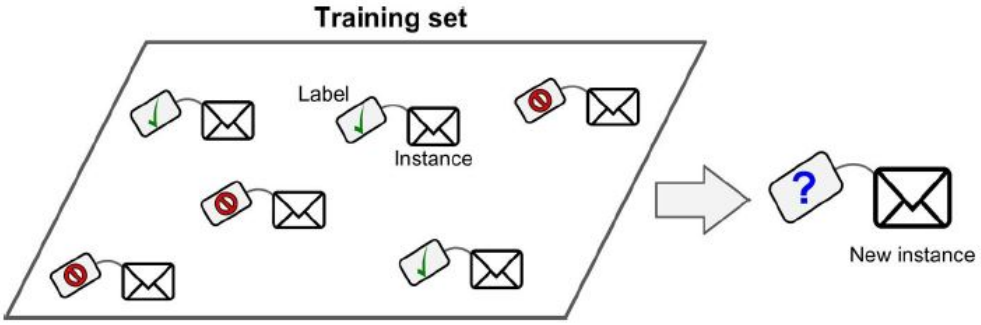


Figura 4: *Ejemplo de aprendizaje supervisado [26].*

- **Aprendizaje sin supervisar**: Este método se refiere a la utilización de algoritmos para la identificación de patrones en conjuntos de datos sin ningún tipo de información etiquetada previamente. El algoritmo analiza estructuras subyacentes de los conjuntos de datos y extrae información útil de ellos buscando relaciones entre las diferentes muestras o parámetros de entrada. En la figura 5 se muestra un ejemplo de un conjunto de datos sin etiquetar y la agrupación de estos.

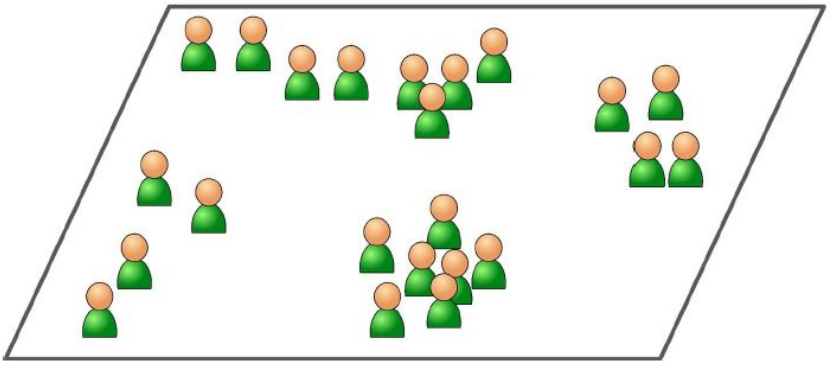


Figura 5: *Ejemplo de aprendizaje sin supervisar [26].*

- **Aprendizaje semi supervisado**: En el aprendizaje semi supervisado, el dataset contiene ejemplos etiquetados y sin etiquetar. Habitualmente, la cantidad de ejemplos sin etiquetar es mucho mayor que los que sí lo están. En la figura 6 se muestran varios puntos grises que representarían la información sin etiquetar y algunos triángulos y cuadrados que harían referencia a las labels, y cómo, a partir de ellos, se intenta deducir a que clase pertenecerá un resultado. La idea de este algoritmo es exactamente igual que la de el algoritmo supervisado, pero gracias a que se le están aportando muchos ejemplos sin etiquetar, se espera que el algoritmo consiga unos mejores resultados. [28]

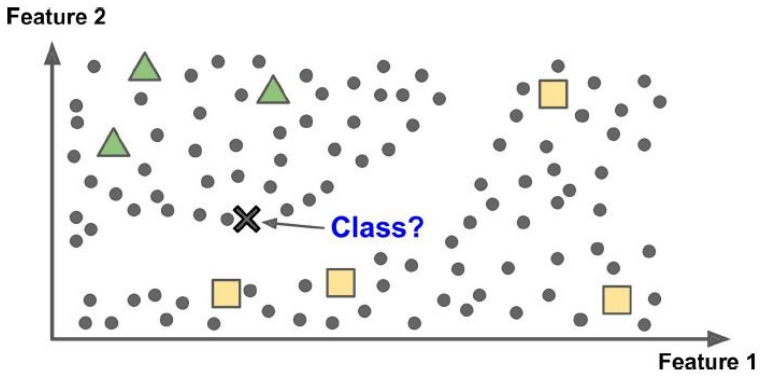


Figura 6: *Ejemplo de aprendizaje semi supervisado [26].*

- **Aprendizaje reforzado**: Este sistema de aprendizaje es muy diferente a los anteriores, puesto que es capaz de analizar el entorno en el que se encuentra y llevar a cabo acciones, de las que obtiene resultados positivos o negativos. Por si mismo, el algoritmo debe aprender cuál es la mejor estrategia (o política) a seguir para obtener el mayor número de resultados positivos. Una política define que acción debe escoger el sistema en una situación dada [26]. La figura 7 muestra un ejemplo básico del aprendizaje reforzado.

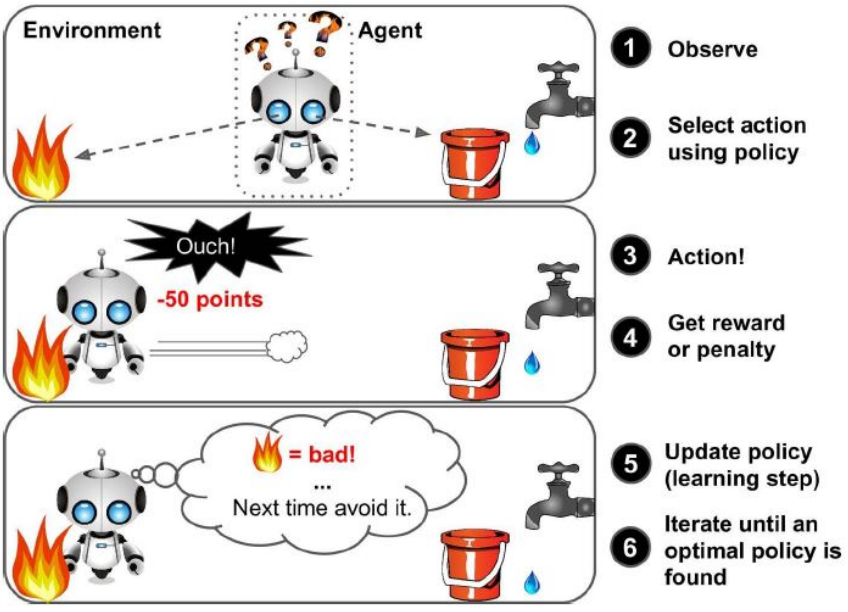


Figura 7: *Ejemplo funcionamiento del aprendizaje reforzado [26].*

Un término relevante en el aprendizaje del Machine Learning en general es el overfitting. La definición del diccionario de Oxford de overfitting es: "La producción de un análisis que se corresponde demasiado o exactamente con un conjunto particular de datos y, por lo tanto, puede no ajustarse a datos adicionales o predecir observaciones futuras de manera confiable" (y es mayoritariamente utilizada para estadística). Esta característica es indeseable en un detector, debido a que el modelo podría comenzar a detectar caras (u objetos) "de memoria" en lugar de por sus predicciones.

Como ya se ha comentado, el Machine Learning lleva a cabo la búsqueda de atributos en objetos. Este proceso ha sido llevado a cabo manualmente a lo largo de la historia, hasta que el Deep Learning dió paso a la automatización.

2.2. Aprendizaje Profundo y su aplicación en detección de objetos

En los últimos años, la Inteligencia Artificial (IA) y el Machine Learning han sido el centro de atención para muchas empresas y ha creado muy altas expectativas para la gente de a pie. Esto es debido a los avances conseguidos tras la aparición del Deep Learning. Desde esta perspectiva, al igual que se hizo con el ML, se intentará dar respuesta a si el Deep Learning es realmente tan importante.

Durante muchos años, los expertos creían que la inteligencia artificial a nivel humano podía ser conseguida teniendo a varios programadores escribiendo un conjunto de reglas lo suficientemente largas para manipular el conocimiento. Esto se llamó "IA simbólica". No obstante, este tipo de trabajo estaba destinado a quedar obsoleto y no fue hasta 1974 que Paul Werbos dió a conocer el término Deep Learning en su tesis doctoral, definido como un nuevo método de entrenamiento de aprendizaje a través de redes neuronales retropropagadas [29]. Geoffrey E. Hinton consiguió aplicar este algoritmo en una máquina para simular en ella el aprendizaje humano convirtiéndose en uno de los primeros investigadores que demostró que se podía entrenar redes neuronales multicapa mediante estos algoritmos [30].

Según viene definido en [22], "el Deep Learning (DL) es un subcampo específico del Machine Learning: una nueva manera de aprender representaciones a partir de datos, que pone un énfasis en el aprendizaje en capas sucesivas de representaciones cada vez más significativas". La palabra deep hace referencia a las sucesivas capas de representación, las cuales generalmente no son superficiales, de ahí su nombre. A medida que hay un mayor número de capas, se dice que el modelo es más profundo. La palabra learning hace referencia al aprendizaje.

A diferencia del Machine Learning, el DL es capaz de obtener características de diferentes objetos de manera automatizada. Esta es la razón de que sea tan empleado junto al ML. El

aspecto negativo de esta automatización, es la carga computacional añadida necesaria, ya que estos procesamientos requieren una cantidad de cálculos mayor. A causa de esto, resultó inviable la continuación de su estudio del Deep Learning.

Es esta necesidad de potencia la que llevó a la utilización de las GPU o unidades gráficas de procesamiento, las cuales fueron ganando protagonismo durante las últimas décadas del siglo XX. Una GPU es un coprocesador dedicado al procesamiento de gráficos u operaciones de coma flotante cuya finalidad es aligerar la carga de trabajo del procesador central en diferentes aplicaciones. Gracias a ellas, se pudo continuar el estudio y desarrollo del DL.

En Deep Learning, las representaciones en capas son casi siempre aprendidas a través de unos modelos llamados redes neuronales, estructuradas en capas literalmente apiladas unas encimas de otras, las cuales, a pesar de su nombre, no son modelos de nuestro cerebro. Un ejemplo del funcionamiento de una red de unas pocas capas se puede visualizar en la figura 8. En la imagen se observa como, para saber de que dígito se trata, se llevan a cabo una serie de transformaciones de una imagen en representaciones muy diferentes de la imagen original que son más informativas a medida que va atravesando las capas. Se puede pensar en una red profunda como una operación de destilado de información en múltiples etapas, donde la información va atravesando filtros sucesivos y termina aportándonos la información deseada [22].

Cada capa almacena un conjunto de números o pesos, que indican el tipo de transformación, que hará dicha capa a los datos que le llegan a su entrada. Se dice que la transformación esta parametrizada por los pesos. En cuanto al entrenamiento, se necesita observar algo para poder controlarlo, y las salidas se controlan con las llamadas funciones de pérdidas, que indican la respuesta más o menos adecuada del modelo. Con esos datos se adecuan los pesos con el optimizador y con ello se mejoran los resultados para que estos se acerquen más a los esperados. La figura 9 representa los pasos básicos a seguir en el entrenamiento de un modelo de Deep Learning.

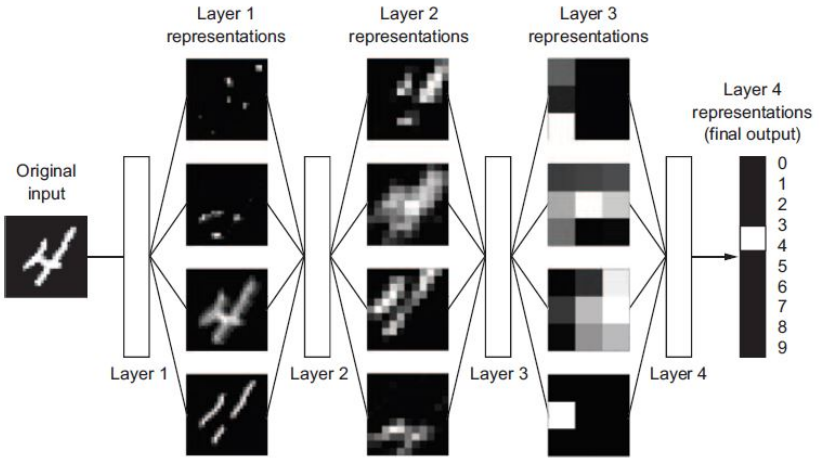


Figura 8: *Procesado de una imagen por una red profunda, a partir de la cual se obtiene el resultado [22].*

Todos los avances mencionados, han facilitado la realización de tareas tan importantes como: Clasificación de imágenes, reconocimiento del habla, transcripción de la escritura a mano, conducción autónoma, mejoría de las búsquedas en la web, etc.

En este sentido, puede llegar a afirmarse la importancia del Deep Learning tal y como se formulaba previamente.

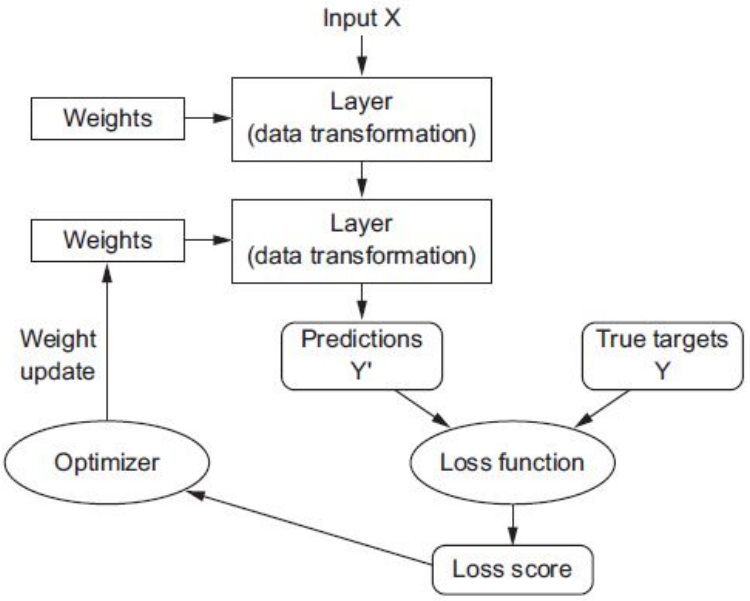


Figura 9: *Pasos básicos seguidos en el entrenamiento de un modelo de Deep Learning [22].*

En la figura 9 se observa como, para cada una de las capas se introducen una serie de pesos. Inicialmente, estos pesos son escogidos por el usuario, pero a medida que se comparan las predicciones con los valores reales a comprobar se obtiene el score de pérdidas. Este devuelve un valor que entrega al optimizador, el cual sirve para corregir los pesos entregados a las capas y gracias a esta modificación, se consiguen adecuar las predicciones para asimilarlas a los valores reales.

2.2.1. Convolutional Neural Networks

Las CNN o redes neuronales convolucionales [31] son un tipo de redes convolucionales profundas, que se utilizan comunmente para el análisis visual de imágenes. El nombre de las CNN indica que emplea un tipo de operación matemática llamada convolución. Una convolución es una tipo especial operación lineal. Estas redes son redes neuronales normales cuya característica es el empleo de convoluciones en algunas de sus capas, en lugar de las multiplicaciones matriciales habituales [32].

Estas redes son especialmente útiles en el procesado de imágenes debido a su capacidad de capturar las dependencias espaciales de los datos. Habitualmente se pueden encontrar 3 tipos de capas en la arquitectura CNN.

- **Capas convolucionales:** Como su nombre indican, son las encargadas de llevar a cabo una convolución, que consisten en tomar "grupos de pixeles cercanos" de la imagen de entrada e ir operando haciendo el producto escalar de estos píxeles contra una pequeña matriz que se llama kernel. El kernel habitualmente es de tamaño 3x3 y recorre toda la imagen de entrada de izquierda a derecha por cada fila hasta terminar en la parte inferior derecha. El procedimiento es llevado tantas veces como sea necesario hasta cubrir toda la superficie de la imagen de entrada. La metodología con la que se lleva a cabo esta convolución viene determinada por dos parámetros: el stride (desplazamiento) y el zero padding (término generalmente utilizado para la inclusión de ceros alrededor de algún valor con la finalidad de que las dimensiones sean las adecuadas).

  El stride controla cómo el filtro se mueve alrededor de la imagen de entrada. Como se puede observar en las figuras 10 y 11, a mayor stride, menor será la dimesión de salida. La figura 10 tiene un desplazamiento de 1, mientras que la figura 11 tiene un desplazamiento de 2.

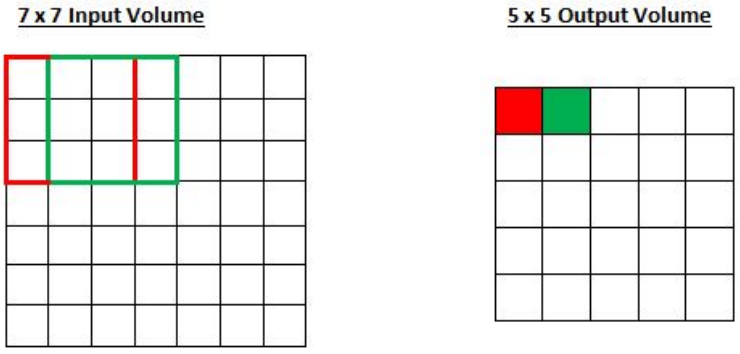


Figura 10: *Entrada y salida correspondientes tras aplicar un kernel con stride de 1 [33].*

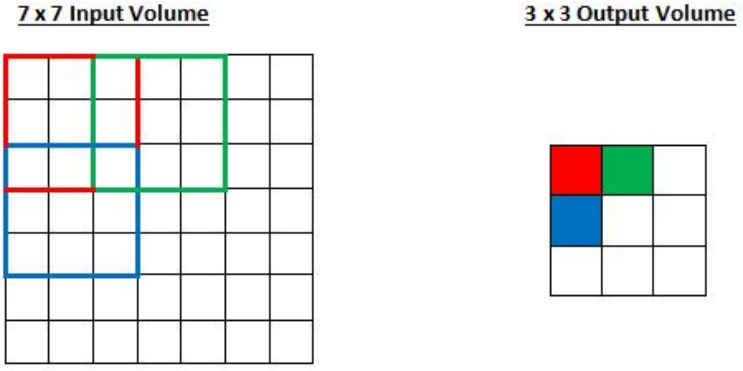


Figura 11: *Entrada y salida correspondientes tras aplicar un kernel con stride de 2 [33].*

Cabe destacar que habitualmente el stride es elegido de manera que el volumen de salida es un valor entero y no una fracción.

Para la explicación del padding hay que pensar que, cuando se aplica un kernel, el tamaño final sufre un decrecimiento. A medida que esto se siga aplicando a las diferentes capas convolucionales, el tamaño del volumen disminuirá más rápido de lo buscado, debido a que en las primeras capas de una red, se busca preservar la mayor cantidad de información sobre el volumen de entrada original para poder extraer esas características de bajo nivel. Por lo que, para mantener el tamaño original, se suele emplear el Zero padding, que rellena el volumen de entrada con ceros alrededor del borde. Por lo que, tras el relleno correspondiente, y la aplicación del kernel, el tamaño original se mantendría. Esto se ve ilustrado en la figura 12.

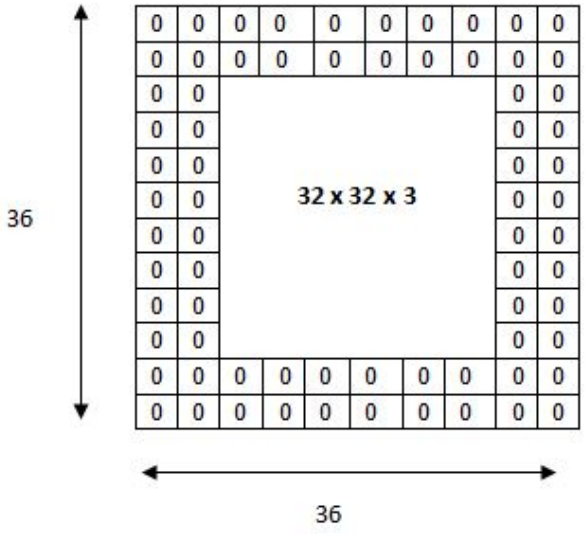


Figura 12: *Aplicación de un padding de ceros al tamaño de entrada original [33].*

- **Capa de pooling:** Esta capa también conocida como capa de agrupación es utilizada para reducir el tamaño de los mapa de características. La finalidad buscada es la obtención de eficiencia computacional, gracias a su menor tamaño de procesamiento; la mejor obtención de características dominantes, y una mayor robustez frente al ruido. Hay muchos métodos diferentes de pooling, pero de todos ellos se van a destacar dos: el Max pooling y el Average pooling [34].

    - **Max Pooling:** Es utilizado mayormente para reducir la variabilidad de las muestras. De las regiones de pooling escogidas, toma el mayor de los valores para formar el nuevo mapa de características como se puede observar en la figura 13

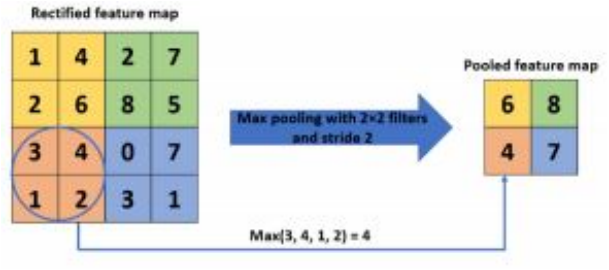


Figura 13: *Operación llevada a cabo en el max pooling [34].*

- **Average Pooling:** Este método lleva a cabo un submuestreo al dividir la entrada en regiones de pooling rectangulares (o cuadradas) y tomar el valor medio de cada región. La figura 14 muestra un ejemplo la operación llevada a cabo.

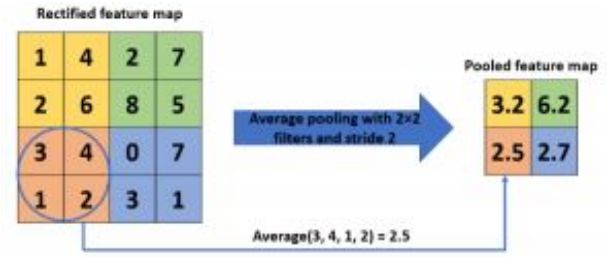


Figura 14: *Operación llevada a cabo en el average pooling [34].*

- **Capa completamente conectada:** Las fully connected layers son las encargadas de combinar la información local recogida en las capas anteriores para la identificación de patrones más generales. Estas conforman las últimas capas de la red habitualmente. La entrada de una capa completamente conectada es la salida de las capas de pooling o las capas convolucionales, la cual es aplanada e introducida a esta capa. El término "aplanar" hace referencia a la conversión de una matriz tridimensional en un vector unidimensional con el que poder trabajar.

2.2.2. Backbone

Según viene definido en el diccionario de Oxford, "un backbone es la parte más importante de un sistema, organización, etc. Esta da soporte y potencia". Los backbones juegan un rol muy importante en la detección de objetos, puesto que su desempeño recae en gran medida en las características extraídas por estos. En este trabajo, uno de los backbones más utilizados es el conocido como ResNet [35].

No obstante, muchos detectores de objetos directamente utilizan redes diseñadas para la clasificación de imágenes como backbones. Un ejemplo que se puede encontrar en este documento es el modelo SSD MobileNet, el cual utiliza la CNN MobileNet como un backbone para mejorar su obtención de características.

ResNet

Las redes residuales, o ResNets [35, 36], aprenden funciones residuales las cuales hacen referencia a la capa de entrada, en lugar de aprender funciones sin referenciar. En lugar de esperar que unas pocas capas agrupadas coincidan directamente con el mapeado subyacente esperado, las ResNets hacen que las capas coincidan explícitamente con el mapeado residual. Y, mayoritariamente, agrupan bloques residuales, los cuales se pueden ver en la figura 15, para formar la red. Por ejemplo, algunos de los modelos utilizados en esa comparativa utilizan diferente cantidad de capas gracias a estos bloques, este es el caso de los SSD Resnet50/101/152 o los Faster R-CNN Resnet50/101/152.

La función $F(x, Wi)$ representa el mapeado residual a aprender. Hay evidencias empíricas de que este tipo de redes son más simples a la hora de realizar optimizaciones y que pueden ganar mayor precisión a medida que se aumenta la profundidad. Todo esto, a costa de memoria y velocidad de procesado.

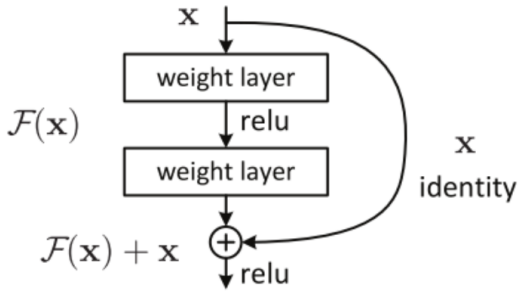


Figura 15: *Esquema básico del aprendizaje residual [36].*

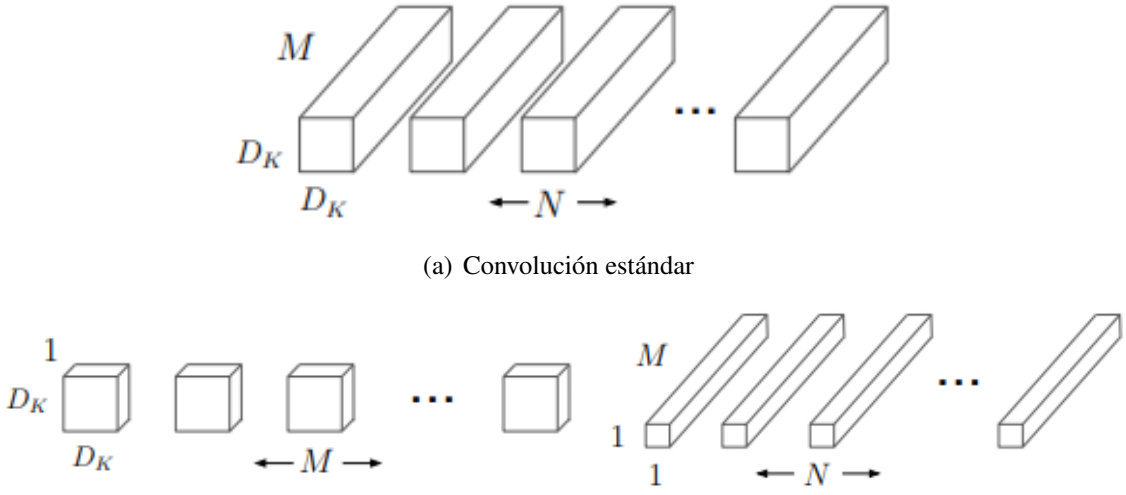
MobileNet

Las Mobile Networks [37] son especialmente útiles para dispositivos móviles o aplicaciones de visión embebidas. Se basan en la optimización de la latencia, pero también en la utilización de redes pequeñas. El modelo de MobileNet se basa en convoluciones en profundidad, cuya función primordial es factorizar convoluciones estándar en 2 tipos: convoluciones en profundidad, y una única convolución puntual (1×1). Este tipo de proceso aplica un único filtro para cada canal de entrada.

La factorización llevada a cabo por este modelo, reduce drásticamente la computación y el tamaño del modelo. En la figura 16 podemos ver como una convolución estándar (a) es factorizada en una convolución en profundidad (b) y una convolución puntual (c).

Todas las capas están seguidas de una batchnorm (normalización de los bloques de imágenes tomados) [38] y una ReLU (Rectified Linear Unit) no lineal, con la excepción de la capa final, la

cual no tiene no linealidad y es alimentada a una capa que aplica una función softmax para su clasificación.



(a) Convolución estándar

(b) Convolución en profundidad

(c) Convolución puntual

Figura 16: *Procesamiento de una imagen en los modelos MobileNet [37].*

2.2.3. Feature Pyramid Networks

Una FPN, o Feature Pyramid Network, [39] es un extractor de características que toma de entrada una imagen con un tamaño arbitrario y a la salida saca mapas de características a múltiples niveles de manera completamente convolucional como se puede ver en la figura 17. Este proceso es independiente de las arquitecturas convolucionales troncales (backbone). Tiene una amplia utilidad en ramas como la detección de objetos ya que actúa como solución genérica para la construcción de pirámides de características dentro de redes convolucionales profundas.

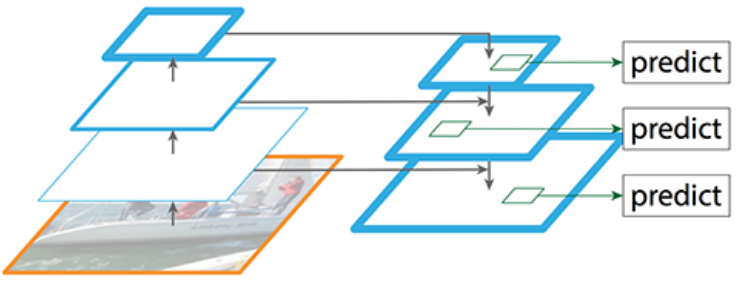


Figura 17: *Estructura de un extractor FPN [39].*

2.2.4. ReLU

Una función de activación lineal rectificada es una función lineal por partes que dará como resultado la entrada en caso de esta sea positiva, o, cero en caso contrario. Las ReLU se han convertido en la función de activación predeterminada para muchos tipos de redes neuronales, debido a que los modelos que la utilizan son más fáciles de entrenar y, a menudo, logran un mejor rendimiento [40].

La función de activación es una "puerta" matemática entre la entrada que alimenta la neurona actual y su salida que va a la siguiente capa. Básicamente deciden si la neurona debe activarse o no [41].

2.3. Historia y Estado del Arte de la Detección de Objetos

La detección de objetos tiene como objetivo localizar cada instancia de un objeto y asignarle una clase en una imagen. En general, un detector de objetos puede dividirse en dos partes, un backbone y una "cabeza", siendo esta última la motivadora de múltiples avances en los últimos años. El término "cabeza" se refiere a la parte del detector encargado de la tarea específica a llevar a cabo, es decir, detección, segmentación, clasificación, etc. La manera de hacer esto es mediante la aplicación de la cabeza al mapa de características generado por la salida del backbone.

La historia de la detección de objetos se puede dividir justo en el año 2014 en dos periodos históricos [42]:

- Periodo de la detección de objetos tradicional (antes de 2014): Detectores HOG (Histogram of Oriented Gradients) y DPM (Deformable Part Model).
- Periodo de la detección basada en aprendizaje profundo (después de 2014), dividida en dos subetapas:
    - Detectores de dos etapas basados en CNN
    - Detectores de una etapa basados en CNN

A continuación se nombran y describen los principales detectores desarrollados a lo largo de las etapas históricas mencionadas, y en la figura 18, se puede ver un diagrama de estos agrupados en los diferentes grupos de detectores.

2.3.1. Detectores tradicionales

Detector HOG

El histograma de gradientes orientados, o según sus siglas en inglés, HOG, fue propuesto en 2005 por N.Dalal y B. Triggs [43]. Este detector supuso una gran mejora en cuanto a la transformación de características de escala invariante y a los shape contexts en ese tiempo. Este podía ser utilizado como detector de objetos genérico, pero inicialmente fue creado con la idea de detectar viandantes. Además, ha sido la base de muchos detectores de objetos o aplicaciones de computer vision.

Para obtener las características, primero se introduce una imagen de la que se preprocesa la información, después, se calculan los gradientes en las direcciones X e Y mediante los valores de cada píxel y sus vecinos, y por último, con estos gradientes, se determina la magnitud y dirección para cada valor de píxel. Con estos datos, se genera el histograma utilizado para llevar a cabo la detección.

DPM

El modelo basado en partes deformables era el pináculo de los métodos de detección de objetos tradicionales [44], tal fue así que consiguió ser ganador 3 años consecutivos del concurso de detección de VOC que consiste, básicamente en un concurso anual dividido en 3 categorías: detección, clasificación y segmentación. El ganador de cada categoría es el grupo de personas que muestre el método con los mejores resultados en un dataset de imágenes concreto.

Propuesto por Pedro Felzenszwalb en 2008 [44] como una extensión del ya comentado detector HOG fue posteriormente mejorado por una serie de correcciones hechas por R. Girshick [45, 46]. Este modelo típicamente consiste en un filtro raíz y un número de filtros adicionales encargados de separar las partes del objeto, cuyas configuraciones pueden ser aprendidas automáticamente como variables por un método de aprendizaje supervisado. La detección se basa ampliamente en el "divide y vencerás", por ejemplo, la detección de un coche se podría considerar como la suma de detecciones de las ventanas, el chásis y las ruedas. Este modelo, aunque algo obsoleto y superado de lejos por los demás, sigue influenciando enormemente a algunos detectores de objetos a día de hoy. [42]

2.3.2. Detectores de dos etapas basados en CNN

En 2012 nacieron las redes neuronales convolucionales [31] y al igual que éstas, este tipo de detectores pueden aprender características robustas y de alto nivel a partir de una imagen. El equipo de R. Girshick propuso en 2014 lo que hoy es conocido como regiones con características de las redes neuronales convolucionales o, por sus siglas en inglés, R-CNN [47].

R-CNN

El funcionamiento de este detector parte de la extracción de un conjunto de posibles detecciones por búsqueda selectiva, estas son reescaladas a un tamaño fijo y pasadas como parámetros de entrada a un modelo CNN entrenado. Por último, se utilizan clasificadores SVM (máquinas de vectores de soporte) lineales para predecir la presencia de un posible objeto en cada región y para reconocer la categoría de cada uno [48]. El problema de este detector es el gran número de posibles detecciones solapadas, lo que supone una gran reducción en la velocidad de detección.

SPPNet

Propuesto por Kaiming He y su equipo en 2014 [49], introdujo las capas de Spatial Pyramid Pooling, que permitía a las CNN generar representaciones de tamaño fijo independientemente del tamaño de la imagen o la region de interés sin la necesidad de reescalarlos. Gracias a esto se evitó el problema de la repetición computacional de características convolucionales, aunque aún tenía ciertos inconvenientes, los cuales se intenaron solucionar con el modelo Fast R-CNN.

Fast R-CNN

Nuevamente Ross Girshick apareció con un nuevo modelo revolucionario en 2015 [50], el cual mejoraba en gran medida a los anteriores R-CNN y al SPPNet y conseguía integrar las ventajas de ambos dos. Este modelo permite entrenar un detector y un regresor de cajas delimitadoras bajo la misma configuración de red. Pero seguía siendo bastante lento debido a que estaba limitado a la detección de propuestas de objeto, lo cual fue solucionado algo más tarde ese mismo año.

Faster R-CNN

En 2015 un equipo junto con R. Girshick propusieron el que para la época sería un modelo del estado del arte llamado Faster R-CNN [51]. Fue el primer detector end-to-end y también el primer detector que consiguió una detección de aprendizaje profundo casi en tiempo real. Su mayor aporte fue la introducción de las RPN o Region Proposal Network: que permitían obtener propuestas de regiones con un coste más bajo que antes. Este modelo no quedó aquí ya que posteriormente ha sufrido una serie de mejoras incluyendo el R-FCN [52] o el Light head R-CNN .

Feature Pyramid Networks

En 2017, Tsung-Yi Lin y un grupo de investigadores entre los que se encontraba Ross Girshick, propusieron las redes piramidales de características [39] sobre una base de Faster R-

CNN. Previo a las FPN, muchos de los detectores basados en aprendizaje profundo se ejecutaban en la capa más superficial de la red, cualidad que dejó de ser necesaria gracias a la creación de este extractor de características. Una vez se aplicó a algunos modelos como Faster R-CNN, se consiguieron resultados sorprendentes para esa época, tanto como para considerar Faster R-CNN, un detector del estado del arte nuevamente. Las FPN son a día de hoy un bloque de construcción básico para la mayoría de los detectores.

2.3.3. Detectores de una etapa basados en CNN

YOLO

You Only Look Once o mejor conocido como YOLO, fue el primer modelo de detección de una etapa creado hasta la fecha [53]. Fue desarrollado en 2015 por Redmon, Divval, Girshic y Farhad. Este modelo sigue la filosofía de aplicar solamente una red neuronal a la imagen completa. El modelo divide la imagen en regiones y predice donde caerá la caja delimitadora y su probabilidad al mismo tiempo. El mayor problema de la primera versión de este detector, a pesar de su gran velocidad, es la falta de precisión en comparación con los detectores de dos etapas sobre todo en objetos de menor tamaño, problema que fue mejor estudiado en las siguientes versiones YOLOv2 y v3.

YOLO ha recibido varias actualizaciones hasta el punto de crear 4 nuevos modelos: YOLOv2 (el cual, como curiosidad, se llamó inicialmente YOLO9000), v3, v4 y v5 [54, 55, 6, 56]. Todos ellos han sido modelos del estado del arte a medida que iban saliendo debido a sus correcciones y mejoras. Actualmente, en 2021, el modelo puntero en detección de objetos es YOLOv5, desarrollado este mismo año.

SSD

Fue propuesto en 2015 por Wei Liu et al. el año 2015 [57]. Sus siglas significan Single Shot MultiBox Detector y fue el segundo detector de una etapa (detrás de YOLO). Su mayor contribución fue la introducción de dos técnicas de detección: multi referencia y multi resolución, las cuales mejoraban la precisión de la detección de este tipo de detectores sobre todo en pequeños objetos, que era precisamente el detalle en el que el YOLO más flaqueaba.

RetinaNet

Tsung-Yi Lin et al. descubrieron las razones por las cuales los modelos de una etapa estaban siempre por detrás de los de dos etapas en cuanto a precisión, y con eso, creó una nueva función de perdidas llamada Focal Loss, la cual fue introducida en el modelo RetinaNet [58] con la finalidad de que el detector se fijara más en ejemplos difíciles y sin clasificar durante el entrenamiento.

Con esto, se pudo obtener resultados muy similares a los buscados en los detectores de dos etapas.

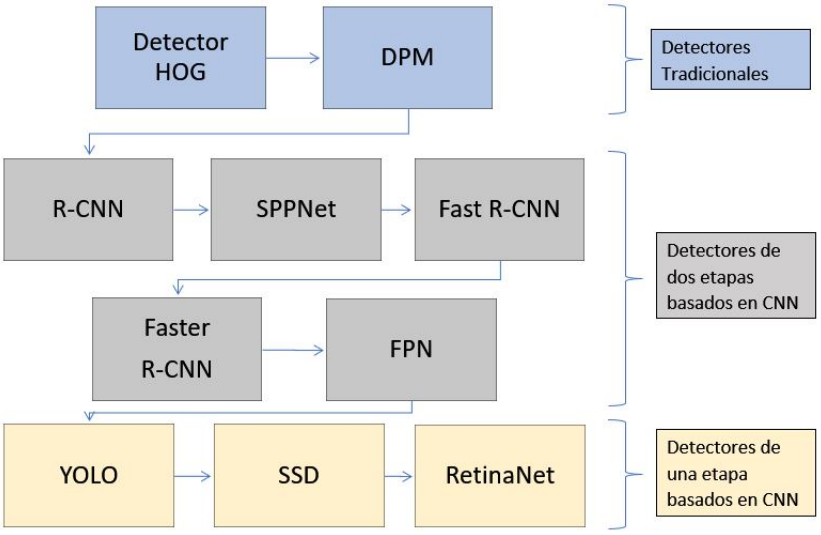
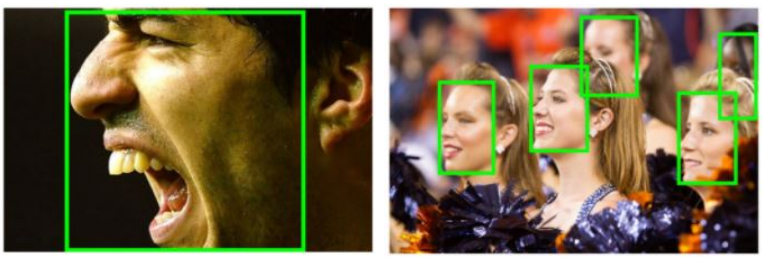


Figura 18: *Diagrama temporal ordenado de los detectores de objetos unidos mediante flechas, indicando el orden de aparición de los modelos.*

2.4. Historia y Estado del Arte de la Detección Facial

Antes de comenzar con la historia de la detección facial, resulta adecuado recordar qué partes forman concretamente una cara, y que debe ser buscado cuando se esté detectando una. Según el capítulo 2 del Manual de Disecciones de Gonzalo López et al. [59], "la cara o rostro representa una región que abarca desde las cejas hasta la barbilla, y la componen los músculos que permiten la transmisión de las emociones de las personas. Incluye las cejas, los ojos, la nariz, las mejillas, los labios, la boca y el menton". Estas son las características buscadas, y como podemos ver en la figura 19, obtenida de los ejemplos aportados en el dataset de WIDER FACE [8], están todas las partes nombradas. A mayores, WIDER FACE añade la frente a todas las imágenes en las que esto resulta posible.



(a) (b)

Figura 19: *Ejemplos de las partes de una cara del dataset de WIDER FACE [8].*

El libro “Past, Present, and Future of Face Recognition: A Review” [60] hace un muy buen resumen de la historia de la detección facial computada, desde sus comienzos en 1964 hasta el 2020 (año de publicación del libro). La figura 20 resume a un simple vistazo, todos los avances que se van a nombrar a continuación.

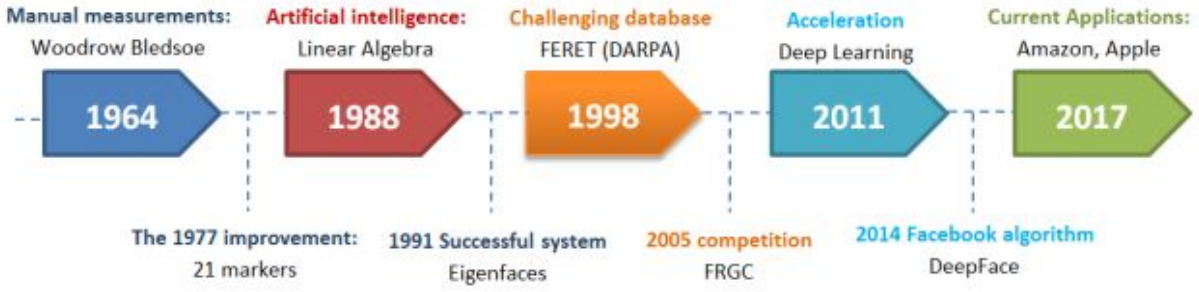


Figura 20: *Historia de la detección de caras [60].*

- En 1964 un grupo americano de investigadores estudiaba programación computacional de reconocimiento facial. Idearon un método (teórico) semiautomático en el cual los operadores debían introducir en el ordenador, 20 medidas de sus caras como pueden ser los tamaños de la boca o de los ojos. Estos tempranos pasos de Woody Bledsoe, Helen Chan Wolf y Charles Bisson, fueron gravemente obstaculizados por la tecnología de la época, pero siguió siendo un gran primer paso importante en la demostración de que el reconocimiento facial es una métrica viable.

- En 1977, 13 años más tarde, reforzaron el sistema previo añadiendo otras 21 nuevas medidas, como podrían ser la anchura de los labios o el color del pelo. Los datos aún debían ser introducidos manualmente.

- 1988 fue el año en el que se pudieron ver avances en este campo gracias a la introducción de la IA. Desafortunadamente, el sistema presentaba un gran número de problemas sin resolver. Por ello pasaron a utilizar álgebra lineal con la intención de darle un nuevo punto de vista al problema y poder interpretar las imágenes de manera diferente, así como encontrar una manera de simplificarlas y manipularlas independientemente de las medidas introducidas a mano por los integrantes.

- Tan solo 3 años después, Alex Pentland y Matthew Turk del MIT presentaron el primer ejemplo exitoso de tecnología de detección facial, Eigenfaces [61], que utilizaba el método estadístico de PCA o Principal Component Analysis.

- En 1998, con la intención de animar tanto a la industria como a las universidades en el avance de la detección de caras, DARPA desarrolló el programa FERET (Face Recognition Technology) [62], el cual proveía de un dataset compuesto por 2413 imágenes de 856 personas.

- El detector de Viola & Jones fue creado en el año 2001 y fue, en un Pentium III a 700MHz, el primer detector que consiguió detectar en tiempo real caras humanas . El algoritmo, que

recibe el nombre de sus creadores, era varias veces más rápido que cualquier algoritmo de la época superándolos de 10 a 100 veces en velocidad [1]. Este detector sigue un tipo de detección muy directa, esta es lo que llamaban sliding windows y consiste en recorrer todas las posibles localizaciones y escalas de una imagen para ver si alguna de ellas contiene una cara humana. Posteriormente fue mejorado con técnicas como la imagen integral, selección de características y detección en cascada.

- Para incentivar nuevamente el desarrollo de tecnologías de reconocimiento facial, en 2005, se lanzó una competición llamada Face Recognition Grand Challenge (FRGC). Los resultados indicaban que los nuevos algoritmos eran 10 veces más precisos que los algoritmos de reconocimiento facial de 2002 y unas 100 veces más precisos que los de 1995.

- Fue en el año 2011 cuando se introdujo el Deep Learning, un hito para todos los tipos de detectores, lo que se tradujo en varias mejoras para todos los modelos que lo utilizaran. Además, permitió la adición de un mayor número de imágenes para el entrenamiento de estos.

- En 2014, Facebook crea un algoritmo interno llamado Deepface [2], el cual, según la red social, se aproxima a un 97 % del desempeño del ojo humano en este campo. Este software crea modelos 3D de los rostros en las fotografías y después los analiza por medio de tecnología de Deep Learning. “Para alinear caras rotadas fuera de plano, utilizamos un modelo 3D de formas genérico y registramos una cámara 3D afín, que es utilizada para deformar la plantilla alineada en 2D al plano de la imagen 3D”.

- Otro gran avance se vió en 2017 cuando Apple lanzó su nuevo modelo, iPhoneX, el primer iPhone que se podía desbloquear con FaceID (término de Apple de marketing para el reconocimiento de caras). “La cámara TrueDepth captura datos precisos de las caras proyectando y analizando unos 30000 puntos invisibles para crear un mapa en profundidad de tu cara y también captura una imagen infrarroja de ella. Parte del motor neuronal transforma la profundidad del mapa y la imagen infrarroja en una representación matemática y compara dicha representación con la información de caras registrada” [63].

Gracias a todos estos avances, se pudieron crear algunos de los detectores de caras utilizados para este documento. Se destacan FACED [64] y el Ultra Light Face Detector [65].

- FACED: Creado en 2018, ha supuesto un enfoque tecnológico basado en la simplicidad del modelo. A diferencia de otros detectores que utilizan muchas operaciones para separar los objetos en diferentes clases, este detector fue creado para la detección exclusiva de caras, y por ello, consigue llevar a cabo los procesamientos en tiempo real.

- UltraLight Face Detection: Similar al anterior modelo, este detector fue creado para obtener velocidades grandes de procesamiento en imágenes, con la ligera diferencia de que permite la utilización de 2 versiones diferentes: una más rápida, y otra más precisa. Su característica más distintiva es el tamaño del modelo, que ocupa aproximadamente 1 MB.

Hay que destacar la imposibilidad de la utilización de modelos como el Deepface de Amazon y el FaceID de Apple debido a su confidencialidad.

2.5. Ventajas e inconvenientes de la detección facial

A sabiendas de todos los avances conseguidos en apenas 60 años, es normal pensar a dónde nos llevará este tipo de detección. Las posibilidades son muy extensas:

- Medicina: puede ser capaz de descubrir en detalle el estado físico de un paciente así como enfermedades que pueda estar sufriendo o tendencias degenerativas en enfermedades. Este es el caso de Cliniface [66], el cual, desarrollado en Australia, pretende ayudar al diagnóstico clínico, así como el tratamiento y monitorización de un paciente medicado a través de esta tecnología.
- Fines legales: videovigilancia avanzada de presos con libertad condicional o seguimiento de sospechosos. También puede facilitar en gran medida la búsqueda de desaparecidos o ayudar en investigaciones policiales.
- Seguridad de la información: como el acceso a aplicaciones únicamente si se utiliza tu cara, seguridad en bases de datos o cajeros automáticos.
- Aplicaciones en domótica: Hay sistemas que permiten el acceso a una vivienda a través del reconocimiento facial si su propietario ha olvidado las llaves.

La detección facial ha venido para quedarse y puede ofrecer grandes mejoras en cuanto a comodidad en la vida cotidiana.

No obstante, hay varios posibles inconvenientes en este tipo de detección y esa es la razón de que este tema sea tan controvertido y haga que, la opinión en relación a este tema en gran parte de la sociedad esté dividida en dos grupos: aquellos que consideran que son avances necesarios y útiles, y aquellos que consideran que esto podría ser peligroso a medio y largo plazo.

Algunos de los principales inconvenientes son:

- La pérdida del derecho a la privacidad a escala global es un miedo creciente en muchas personas. Por ejemplo, el FBI posee una base de datos llamada NGI-IPS con fotografías de personas acusadas o juzgadas en procedimientos civiles o penales. El problema es que una investigación de la American Government Accountability Office sacó a la luz que su base de datos incluía imágenes de personas que nunca habían sido objeto de investigación, además de poseer fotografías de permisos de conducción, pasaportes y visados.

- Hace unos años, una persona con suficiente paciencia y una simple fotografía tuya, podía encontrar tus datos sin muchas complicaciones. Este ejemplo se ve claro con la aplicación FindFace, que, creada por una startup rusa, permitía a sus usuarios identificar extraños gracias a imágenes de sus caras. Afortunadamente, en 2018, la empresa dejo de proporcionar este servicio para transformarse en una línea de servicios enfocada en empresas privadas.

Como se ha indicado, la detección de caras posee grandes cualidades para mejorar la calidad de vida media y a su vez posee argumentos en contra de su desarrollo.

No obstante, el estado del arte de la detección posee algunos problemas que motivan la realización de este proyecto. ¿Por qué se utilizan menos los modelos genéricos para la detección de objetos específicos (coches, animales, personas, etc.)? ¿Cual es la razón de que en algunos casos, los modelos específicos se utilicen en mayor medida que los detectores de objetos genéricos? ¿Tales son sus diferencias? ¿Podría servir para la detección un modelo genérico bien entrenado o es necesario crear uno desde cero para que sea más eficiente?.

Este documento pretende servir de respuesta, haciendo uso para ello de resultados experimentales llevados a cabo bajo las mismas condiciones de trabajo y un análisis en profundidad de los modelos a utilizar.

Capítulo 3

Metodología de la comparativa

Este capítulo va enfocado a la explicación de la metodología seguida para la evaluación y comparación de los detectores. Se explica brevemente en que consisten las bases de datos, el transfer learning, las métricas a utilizar y más en detalle se cuenta cómo funcionan los modelos de detección empleados. También se explican las diferentes técnicas de simplificación, así como las herramientas de desarrollo utilizadas.

3.1. Bases de Datos

Una base de datos buena es el elemento principal para la creación de un detector de objetos. Es necesario que esta sea precisa y que las imágenes sean similares a aquellas que el modelo encontrará una vez salga del proceso de entrenamiento y evaluación.

3.1.1. WIDER FACE

En el presente TFG se lleva a cabo la comparativa de modelos de detección de caras, por lo que habrá que buscar una buena base de datos con caras suficientes. Debido a que esta es una comparativa entre modelos, el conjunto de imágenes proporcionado por WIDER FACE es suficiente [8].

El dataset de WIDER FACE es un conjunto de datos de referencia para la detección de rostros, cuyas imágenes son seleccionadas de un conjunto de datos disponibles públicamente. Un ejemplo se puede observar en la figura 21. Este dataset dispone de 32.203 imágenes y 393.703 rostros etiquetados con un alto grado de variabilidad en la escala, la pose y la oclusión. El conjunto de datos de WIDER FACE está organizado en 61 clases de eventos, para los cuales se selecciona aleatoriamente un 40 %/10 %/50 % como conjuntos de entrenamiento, validación

y prueba respectivamente. Este dataset adopta la misma métrica de evaluación empleada en el conjunto de datos PASCAL VOC.

Figura 21: *Ejemplos del dataset de WIDER FACE [67].*

3.1.2. PASCAL VOC

El reto de PASCAL VOC (Visual Object Classes) [68] es un dataset muy popular para la construcción y evaluación de algoritmos para la clasificación de imágenes, detección de objetos y segmentación.

Los principales objetivos de este proyecto son:

- Proveer de conjuntos de datos de imágenes estandarizadas para el reconocimiento de diferentes clases de objetos.
- Proveer de un conjunto de herramientas común para acceder a datasets y las anotaciones.
- Permitir la evaluación y comparación de diferentes métodos de detección.
- Realizar retos evaluando el rendimiento en el reconocimiento de clases de objetos (desde 2005-2012).

Este reto fomenta dos tipos de participación:

- La participación de métodos entrenados, utilizando únicamente los datos de entrenamiento y validación.
- La participación de métodos construidos o entrenados, utilizando cualquier tipo de datos excepto los entregados como datos de test. En este grupo se incluyen los detectores comerciales.

En ambos casos, los datos de test deben ser estrictamente utilizados para la evaluación de los métodos, no pueden ser usados para el entrenamiento o afinamiento de sistemas [68]. Un ejemplo del dataset disponible se puede observar en la figura 22.

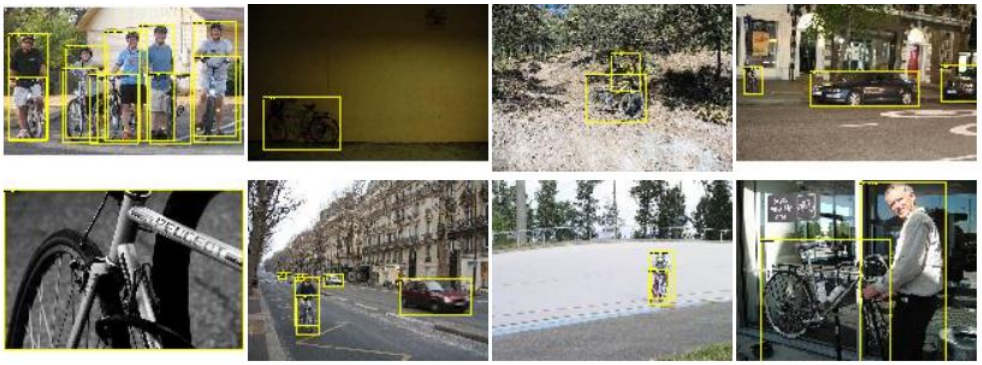


Figura 22: *Ejemplos del dataset de PASCAL VOC [69].*

3.1.3. Microsoft COCO

Common Objects in Context es un conjunto de datos de detección, segmentación y etiquetado de objetos a gran escala. Posee un conjunto de 328000 imágenes con más de 2500000 etiquetas, 91 categorías de objetos comunes y segmentación semántica de imágenes.

Cada anotación de objetos contiene un conjunto de campos, incluyendo la identificación de la categoría y la máscara de segmentación del objeto. El formato de segmentación depende de si la instancia representa un único objeto o una colección de estos. La imagen 23 muestra varios ejemplos de este conjunto de datos.

Figura 23: *Ejemplos del dataset de Microsoft COCO [70].*

3.2. Metodología de entrenamiento

3.2.1. Transfer Learning

Previo a mostrar los métodos seguidos para llegar a los dataset de entrenamiento utilizados, es necesario repasar el término de Transfer Learning puesto que su importancia en el entrenamiento de redes neuronales es muy alta.

Un error común que había hace unos años en la comunidad del DL era que se creía que se necesitaban enormes cantidades de datos sin los cuales, no se puede esperar crear un modelo de aprendizaje efectivo de Deep Learning. Si bien es cierto que la cantidad de datos es muy importante para crear una red, la idea del Transfer Learning ha ayudado a disminuir la necesidad de la gran cantidad de datos.

El Transfer Learning es el proceso de tomar un modelo preentrenado (por otra persona con sus propios parámetros y pesos) y afinarlo con otro dataset escogido por el usuario. La idea es que el modelo preentrenado funcionará como un extractor de características, y de él, se reemplaza la última capa con el clasificador a utilizar (dependiendo del problema para el que se utilice el modelo). Después, se "congelan" los pesos de todas las demás capas y se entrena la red de manera normal. El término "congelar" significa mantener los pesos durante la optimización [33, 71].

Gracias a este método, en lugar de entrenar toda la red mediante una inicialización aleatoria de los pesos, podemos utilizar los pesos del modelo preentrenado (y congelarlos) y centrarnos en las capas más importantes (las que están más arriba) para el entrenamiento. Si el conjunto de datos a utilizar difiere en gran medida frente al dataset base, entonces habría que llevar a cabo un entrenamiento más en profundidad, entrenar más capas y congelar sólo un par de las capas bajas. Un ejemplo de esto se puede observar en la figura 24.

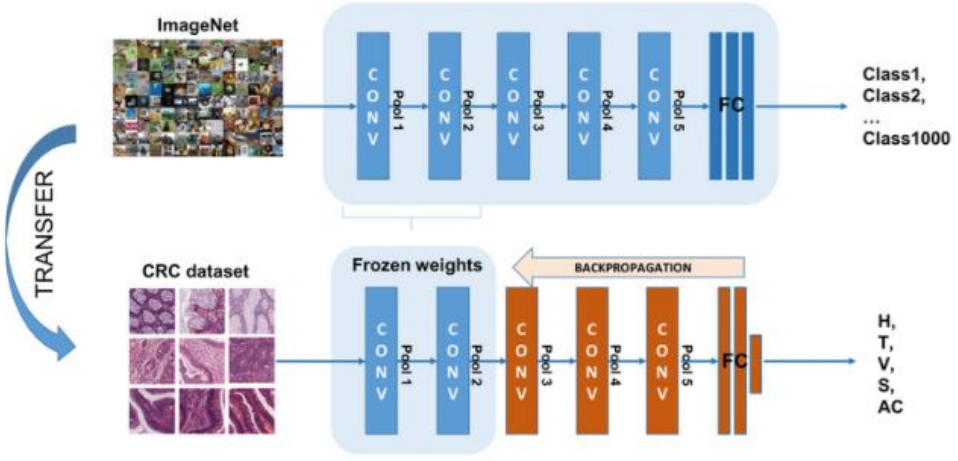


Figura 24: *Ejemplo de Transfer Learning con congelación de pocas capas [72].*

3.2.2. Elección de los conjuntos de entrenamiento, validación y test.

Es muy importante definir un buen conjunto de entrenamiento puesto que, gracias a él, la red convolucional de nuestro modelo irá aprendiendo a diferenciar entre imágenes. Esta es la razón por la que se recomienda entrenar los modelos con imágenes similares a las que va a ser sometido una vez llevado a una situación real. Por ello no puede haber una cantidad de imágenes

de entrenamiento demasiado pequeña (puesto que no aprenderá suficiente), ni demasiado grande (puesto que sufrirá de overfitting, y no aprenderá a diferenciar imágenes sino que las clasificará de memoria). Para ello se utilizó el dataset proporcionado por WIDER FACE [8]. Cabe señalar que un entrenamiento correcto suele necesitar 3 conjuntos de imágenes en total:

- Set de entrenamiento: a partir del cual se irá enseñando a la red convolucional los objetos que debe buscar.
- Set de validación: con el que el modelo irá viendo cada cierto tiempo como de bien se está comportando, comparándolo con imágenes no utilizadas en el entrenamiento.
- Set de test: sobre el que se comprueba los resultados finales del modelo tras el aprendizaje.

Por como funciona la API de TensorFlow, el conjunto de validación ya viene incluido en el conjunto de test, ya que, citando con palabras textuales de la página de la API de TensorFlow, "Mientras el proceso de entrenamiento está activo, ocasionalmente, se generará un checkpoint dentro del directorio training_demo/training, el cual corresponde a capturas del modelo en algunos de los pasos por los que ha ido avanzando el modelo en su entrenamiento. Cuando un set de nuevos checkpoint es generado, el proceso de evaluación emplea dichos archivos para evaluar cuán bien está desempeñando el modelo en la detección de los objetos en el conjunto de test. Los resultados de esta evaluación están resumidos en forma de una serie de métricas, las cuales pueden ser examinadas a lo largo del tiempo de ejecución del programa".

Un parámetro de entrenamiento relevante en esta comparativa es el llamado batch_size, que es indicativo del número de imágenes entregado al modelo en cada paso del entrenamiento, por lo que, a menor batch_size, más rápido se podrá entrenar el modelo, pero si el número de pasos dados no es acorde, tendrá peor precisión. En la comparativa, el batch_size utilizado para entrenar los modelos es de 8 imágenes por cada paso de entrenamiento.

Para la decisión del conjunto de imágenes utilizados en la comparativa, se decidió utilizar 1000 imágenes de test para la comprobación inicial de resultados, y se llevó a cabo una comparativa previa para encontrar la cantidad óptima de imágenes en el entrenamiento.

Tras los entrenamientos y comprobaciones pertinentes, el tamaño del conjunto de entrenamiento fue decidido en 3000 imágenes con un batch_size de 8, y el conjunto de test será de 1000 imágenes. Al no utilizar en su totalidad el dataset disponible por problemas computacionales de memoria, este conjunto posee de una proporción 60 %/20 %/20 %, diferente del 40/10/50 de WIDER FACE. La razón está fundamentada en una serie de experimentos llevados a cabo con diferentes conjuntos de entrenamiento, test y validación, cuyo resultado final demostraba mejores resultados con esta proporción.

3.3. Métricas utilizadas a lo largo de la comparativa

- Para medir la velocidad se utilizan los fps, o frames per second, es decir, fotogramas procesados en 1 segundo. Este parámetro es calculado de una manera muy simple con la expresión matemática

$$Frecuencia = \frac{Numero\,de\,fotogramas}{Tiempo\,total} \tag{1}$$

  mediante la cual, si obtienes el tiempo de inferencia de una imagen (es decir, el tiempo necesario para detectar todas las caras y mostrar la salida), se puede obtener la frecuencia de imágenes procesadas por segundo. La velocidad es medida en tiempo total de ejecución, desde que se comienza a procesar la imagen hasta que se terminan los cálculos necesarios.

- Antes de ir con el segundo parámetro, por simplicidad, se explicará el término de recall, el cual se calcula como:

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

  Donde TP representa los verdaderos positivos y FN los falsos negativos.

  Un TP (verdadero positivo) es utilizado para mostrar que el detector ha categorizado correctamente la imagen, es decir, ha mostrado la imagen de un gato como un gato.

  Un FP (falso positivo) se utiliza cuando se categoriza una imagen que no debería haber sido. Por ejemplo, etiquetar la imagen de un gato como un perro.

  Un TN (verdadero negativo) consiste en no cometer un error en el etiquetado. Es decir, no etiquetar una imagen de un gato como un caballo. Esta métrica no es utilizada en la detección de objetos.

  Un FN (falso negativo) no categoriza una imagen que debería haber sido etiquetada. Por ejemplo, no etiquetar una imagen de un gato como un gato.

- El siguiente parámetro es el mAP, o mean Average Precision, pero antes de eso, se hará una breve explicación de como se computa la precisión y un concepto conocido como IoU (Intersection over Union).

  - La precisión es calculada como

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

    Siendo FP los falsos positivos.

  - La IoU calcula la superposición de 2 cajas delimitadoras: la predicha, frente a la real (que suele ser conocida también como ground truth). Un ejemplo es mostrado en la figura 25. En algunos datasets se predefine un umbral de IoU, generalmente de 0.5, a partir del cual se considera si el objeto detectado es TP (IoU$\geq$0.5) o FP (IoU$<$0.5) [73]. Por otro lado, si el ground truth está presente en la imagen y el modelo ha fallado a la hora de detectarlo, se clasifica como FN.

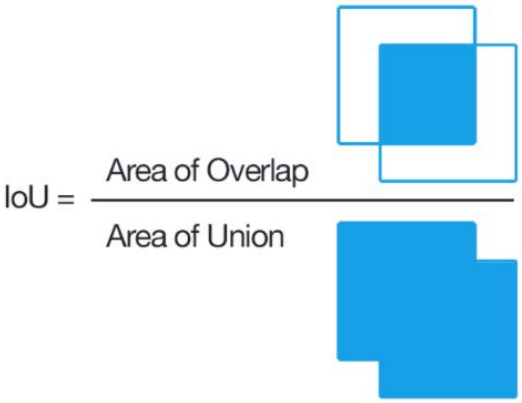


Figura 25: *Cálculo de la IoU [74].*

- AP o Average Precision: Aquí hace falta hacer distinción entre la evaluación hecha por COCO y la hecha por PASCAL, que son las dos principales evaluaciones que empleadas en esta comparativa.

  Para COCO, mAP y AP son lo mismo, no hacen distinción: "El AP es promediado entre todas las categorías. Tradicionalmente, es llamado mAP. No hacemos distinción entre AP y mAP (a diferencia de AR y mAR) y se asume que la diferencia es clara dependiendo del contexto" [75]. El mAP se calcula haciendo la media entre todas las categorías de objetos, y 10 umbrales IoU, los cuales abarcan desde 0.5 a 0.95 con saltos de 0.05 cada uno. Habitualmente viene representado como mAP(0.5:0.05:0.95).

  En el caso de la evaluación de PASCAL VOC, se calcula la precisión y recall de las imágenes. Una vez obtenidos estos parámetros, estos suelen ser organizados en una tabla similar a la mostrada en la figura 26.

Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0	0.4
3	False	0.67	0.4
4	False	0.5	0.4
5	False	0.4	0.4
6	True	0.5	0.6
7	True	0.57	0.8
8	False	0.5	0.8
9	False	0.44	0.8
10	True	0.5	1.0

Figura 26: *Tabla con los valores de precisión y recall obtenidos de una serie de imagenes procesadas.*

Una vez generada la tabla anterior, si se dibujara una gráfica con los valores de la precisión en el eje Y, y los del recall en el eje X, la gráfica resultante, será similar a la que se puede ver en la figura 27, debido a que los valores de la precisión provocan un patrón en zig-zag.

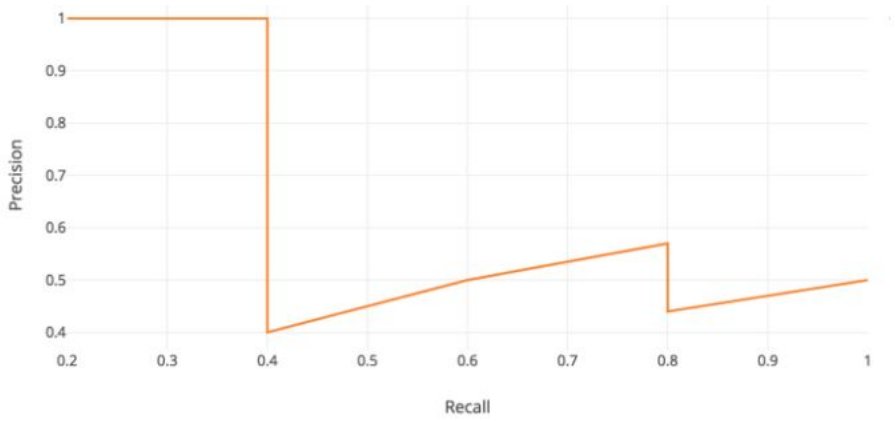


Figura 27: *Curva precisión vs recall [76].*

El valor buscado, es decir, el Average Precision, es el área bajo la curva de la figura 27, cuyo valor viene dado por la siguiente expresión matemática $AP = \int_0^1 p(r)dr$. Generalmente, la curva es ligeramente alisada antes de calcular el AP, reemplazando cada valor de precisión con el máximo valor de precisión a su derecha en ese nivel de recall como se puede observar en la figura 28. De esa manera, la curva decrecerá de manera monótona en lugar de con patrón zig-zagueante y el AP calculado será menos susceptible a pequeñas variaciones. Matemáticamente, se sustituye el valor de precisión por recall ($\widetilde{r}$) con el mayor valor de precisión para cualquier recall $\geq \widetilde{r}$.

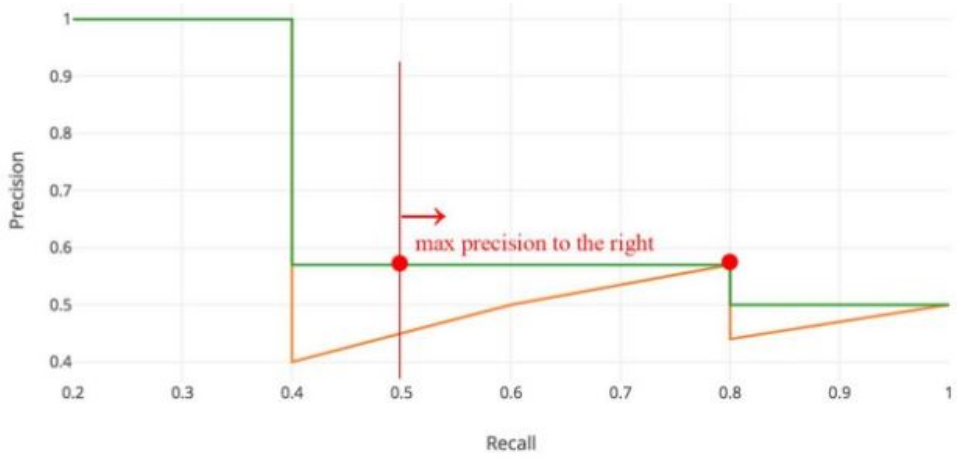


Figura 28: *Curva precisión vs recall alisada [76].*

En PASCAL VOC2008, se calcula una media de 11 puntos interpolados en el AP. Primero se divide la curva anterior en 11 valores (0, 0.1, 0.2, ..., 1) y se computa la

media de los máximos valores de precisión para estos 11 valores de recall mediante la siguiente expresión:

$$AP = \frac{1}{11} \times \sum_{\{r \in 0,0,...,1,0\}} AP_r = \frac{1}{11} \times \sum_{\{r \in 0,0,...,1,0\}} p_{interp}(r) \tag{4}$$

El problema de este método para calcular el AP es que es menos preciso con valores relativamente bajos de precisión. Por esa razón, las últimas competiciones de PASCAL VOC (2010-2012) calculan exactamente el área bajo la curva una vez eliminados los zig-zags. En lugar de muestrear 11 puntos, se muestrea $p(r_i)$ cada vez que cae la gráfica, lo cual se puede ver en la figura 29.

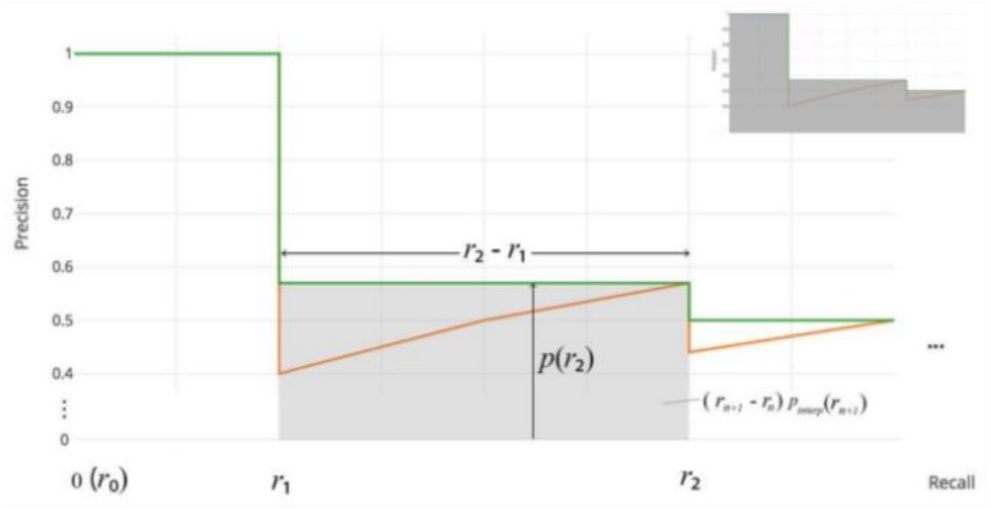


Figura 29: *Curva precisión vs recall con muestreo* $p(r_i)$ *[76].*

Además, se computa el Average Precision como el sumatorio de los bloques rectangulares con la expresión:

$$AP = \sum_{n} (r_{n+1} - r_n) p_{interp}(r_{n+1}) \tag{5}$$

Donde

$$p_{interp}(r_{n+1}) = \max_{\widetilde{r} \geq r} p(\widetilde{r}) \tag{6}$$

Para terminar con todo lo dicho anteriormente, hay que dejar claro que COCO, como se ha dicho anteriormente, no hace diferenciación entre AP y mAP, y PASCAL VOC, una vez calculado el AP de una clase, lleva a cabo un promediado de los distintos AP para todas las clases, obteniendo así un valor final de mAP.

Para este trabajo, se ha utilizado una única clase (caras) en lugar de las recomendadas por COCO y PASCAL, por lo que el AP calculado de la clase utilizada, coincide a su vez con el mean Average Precision (en la métrica de PASCAL).

- Total Loss es un parámetro calculado como la suma de las pérdidas de clasificación y las pérdidas de localización. Generalmente estas tres son mostrados en gráficas, las cuales suelen decrecer a medida que aumenta el número de pasos en el entrenamiento del modelo de detección y el valor que toman es el del último valor de la gráfica tras completar el entrenamiento (en esta comparativa, el valor se toma en el paso 25000).

  Estas funciones de pérdidas indican cómo de bueno es el modelo en una tarea determinada. Y, dependiendo de la tarea concreta, casi todos los modelos tienen como objetivo, minimizar las pérdidas.

  - Las pérdidas de clasificación están indicadas para cualquier tarea que requiera clasificación. A partir de k categorías se comprueba como de bien trabaja el modelo en cuestión en la clasificación de x número de objetos en k categorías.
  - Las pérdidas de localización hacen referencia a cómo de bien es capaz de localizar el modelo una serie de objetos detectados en una imagen, por ello suele ir de la mano con el término IoU, ya que se necesita saber la posición de las cajas delimitadoras ground truth y de las cajas delimitadoras detectadas.

  Idealmente, el resultado final de la gráfica de pérdidas totales se busca que esté en torno al 1 %.

3.4. Explicación en detalle de los modelos de detección utilizados

3.4.1. Modelos genéricos de detección

EfficientDet

Este detector [77] surgió en el año 2019 y proponía 2 retos con su salida: la fusión multiescalada y eficiente de características y el escalado del modelo. Ambas se pueden observar en la figura 30.

Primer reto: Fusión multiescalada y eficiente de características. Al fusionar diferentes características de entrada, la mayoría de detectores previos simplemente las suman sin distinción, sin embargo, teniendo en cuenta que estas características de entrada están a diferentes resoluciones, observamos que generalmente contribuyen a las características salida de manera desigual. Para

atajar este problema, se propuso una Feature Pyramid Network bidireccional (BiFPN).

Segundo reto: Escalado del modelo. Mientras trabajos previos utilizan redes backbone mayores, cuyo propósito es mejorar el desempeño del modelo [78], o imágenes de entrada de mayores dimensiones, inspirado en recientes trabajos como [79], se propuso un método de escalado combinado para detectores de objetos. Este método escala conjuntamente la resolución de todos los backbone, FPNs, y redes de predicción de cajas y clases. Por esto, combinando las EfficientNets [79] con el propuesto BiFPN y el escalado combinado, desarrollaron una nueva familia de detectores de objetos: EfficientDet, que consigue mejor precisión con un menor número de parámetros y FLOPs (Floating Point Operations) que previos detectores. Los EfficientDet son un tipo de detectores de una sola etapa, los cuales destacan por su gran eficiencia, debido a sus arquitecturas directas, frente a los modelos de dos etapas, los cuales mantienen una firme ventaja en cuanto a precisión se refiere [80]. Este, además, mantiene el ratio inicial de la imagen a la hora de hacer el redimensionado.

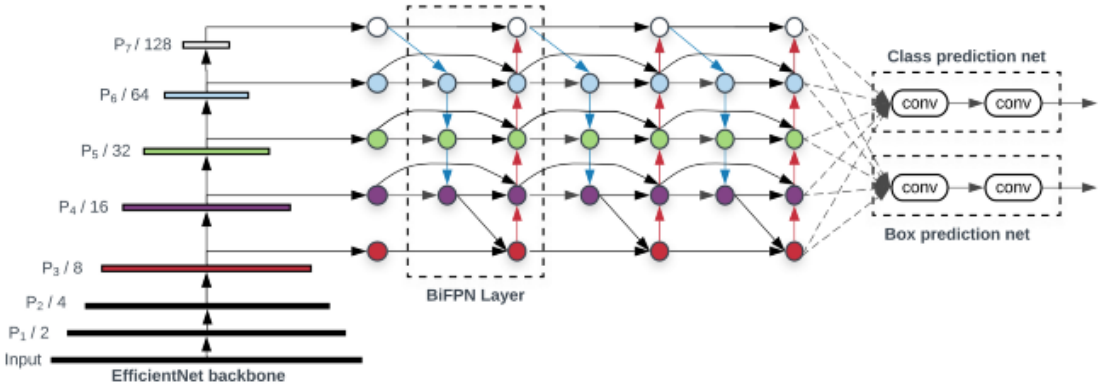


Figura 30: *Esquema de detección del modelo EfficientDet [81].*

SSD

La aproximación del modelo Single Shot Multibox Detector [57] está basada en una red convolucional hacia adelante que produce una colección de cajas delimitadoras y puntuaciones dependiendo del objeto detectado en las cajas. Un ejemplo de este tipo de modelo se puede observar en la figura 31. Las primeras capas de la red están basadas en una arquitectura VGG-16 [82] a la cual se suele referir como backbone y que es empleada para una clasificación de alta calidad de las imágenes. Posteriormente, se añade la estructura auxiliar de la red que permite detectar objetos con las siguientes características.

Mapas de características multi-escala para la detección:
Se utilizan capas convolucionales al final del backbone (VGG-16) las cuales decrecen progresivamente de tamaño y permiten predicciones a múltiples escalas. El modelo convolucional de predicción de detecciones es diferente para cada una de las capas.

Predictores convolucionales para detección:
Cada capa añadida puede producir un set fijo de predicciones usando un set de filtros convolucionales. Estos se indican en la parte más externa de la arquitectura de red en la figura 31. Para cada capa de tamaño m×n con p canales, el elemento básico para la predicción de parámetros de una posible detección es un pequeño kernel de tamaño 3×3×p que produce, o una puntuación (score) por categoría, o una forma compensada de tamaño relativo a las coordenadas por defecto de la caja. En cada lugar de los que se aplica el kernel, se obtiene un valor de salida, los cuales están medidos de manera relativa a la posición de una caja, que a su vez es también relativa a la localización de cada mapa.

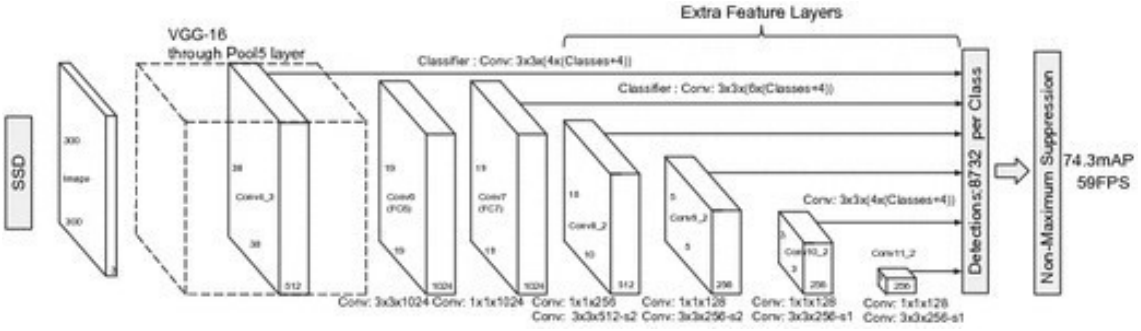


Figura 31: *Esquema de detección del modelo SSD [57].*

Para el entrenamiento se sigue una técnica diferente, ya que la clave reside en que, a diferencia de otros detectores típicos, para SSD (al igual que en YOLO, Faster R-CNN y MultiBox) la información acertada debe ser asignada a salidas específicas del detector. Una vez hecho esto, la función de pérdidas y propagación hacia atrás (recall), es aplicada end-to-end.

Faster R-CNN

Este modelo ha ido sufriendo varias mejorías a lo largo del tiempo, por ello, se explicarán brevemente cada uno de los pasos que se han dado.

- R-CNN:
  La idea de los creadores de las Region Based Convolutional Neural Networks [48] fue combinar dos factores clave. Aplicar Redes Neuronales Convolucionales (CNN) de gran capacidad sobre las propuestas de regiones para localizar y segmentar objetos. Y que, cuando los datos etiquetados de entrenamiento fueran escasos, se pudiera conseguir una mejora significativa del desempeño gracias al preentrenamiento supervisado para llevar a cabo tareas auxiliares y gracias a ajustes específicos de dominios. Además, debido a que combinan regiones propuestas con CNNs, llamaron a su método, R-CNN, es decir, regiones con características de las CNN.

Este sistema de detección de objetos consiste en tres módulos. El primero genera propuestas de regiones independientes de las categorías, las cuales determinan los candidatos a posibles detecciones. El segundo módulo es una gran CNN que extrae un vector de características de longitud fija para cada una de las regiones. Y el tercer módulo es un conjunto de máquinas de vectores de soporte (algoritmos de aprendizaje supervisado) o SVMs lineales de clases específicas.

Las R-CNN usan búsquedas selectivas para identificar un número de candidatos para regiones de cajas delimitadoras (regiones de interés), y posteriormente extraen características de cada una de las regiones independientemente por clasificación.

- Fast R-CNN:
  Este modelo [50] de detección de objetos mejora a su predecesor (R-CNN) en bastantes aspectos:

  - Mejor calidad de detección (mAP).
  - Entrenamiento en una sola etapa.
  - El entrenamiento puede actualizar todas las capas de la red.
  - No se necesita almacenamiento en disco para el cacheado de las características.

  A la hora de detectar un objeto, la red procesa inicialmente toda la imagen con varias capas convolucionales y capas de "max pooling" para producir un mapa de características convolucional. Entonces, para cada propuesta de objeto, una capa de pooling de la región de interés, extrae un vector de características de longitud fija del mapa de características. Cada uno de estos vectores es introducido en una secuencia de capas completamente conectadas para finalmente ser separado en dos capas de salida. La primera de ellas produce el score en la salida, que es una estimación de probabilidad de que el objeto pertenezca a una de las K clases de objetos posibles. Y la otra capa da de salida 4 valores que corresponden con los extremos superiores e inferiores de la caja delimitadora (bounding-box) para una de las K clases.

- Faster R-CNN:
  Al igual que su predecesor, este modelo de detección de objetos [51] es una mejora de los anteriores modelos, solo que, en este caso, es una mejora respecto a Fast R-CNN debido a la utilización de redes de propuesta de regiones (RPN) en combinación con el modelo CNN. Las RPN comparten características convolucionales de la imagen completa con la red de detección, permitiendo propuestas de regiones prácticamente libres de coste. Es una red convolucional completa que simultáneamente predice los límites de los objetos y sus scores en cada posición. La RPN es entrenada de extremo a extremo para generar propuestas de regiones de gran calidad, las cuales son utilizadas por Fast R-CNN para la detección. RPN y Fast R-CNN son fusionadas en una sola red al compartir sus características convolucionales, en la cual, la RPN le dice a la red unificada donde buscar.

Como conjunto, Faster R-CNN consiste en dos módulos. El primer módulo es una red profunda completamente convolucional encargada de proponer regiones y el segundo módulo es el detector de Fast R-CNN que utiliza las regiones propuestas. Esto se puede ver en la siguiente figura.

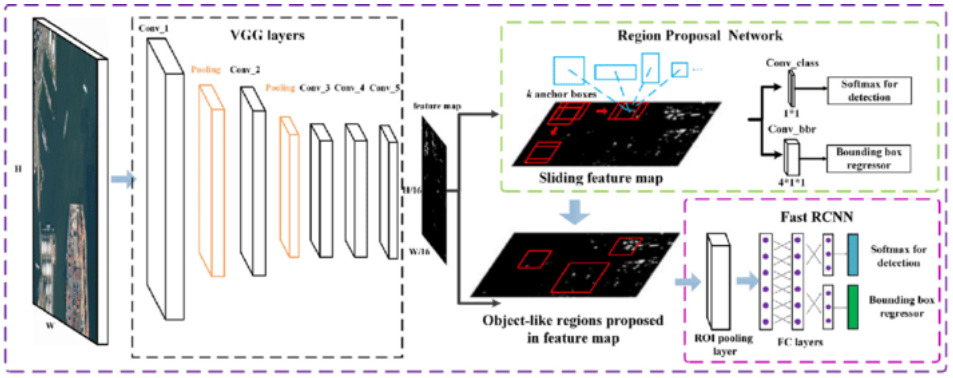


Figura 32: *Esquema de la arquitectura de Faster R-CNN [83].*

YOLO

Al igual que Faster R-CNN [51], YOLO es un modelo que ha sufrido muchas modificaciones con el tiempo. A continuación se explicarán algunos avances recibidos.

- YOLO:
  La teoría detrás de You Only Look Once [53] es relativamente sencilla: Una sola red convolucional que predice simultáneamente múltiples cajas delimitadoras y scores para esas cajas. El mayor fuerte de este modelo, y la razón por la que fue tan importante, es que entrena sobre imágenes completas en lugar de dividirlas en porciones (de ahí su nombre), lo cual optimiza directamente el desempeño de la detección.

  Las primeras capas convolucionales de la red son las encargadas de obtener las características de la imagen mientras que las capas posteriores predicen las coordenadas de la salida, así como sus probabilidades. Esto se puede ver en la figura 33, así como algunas partes más concretas del procesado que se lleva a cabo en esta arquitectura.

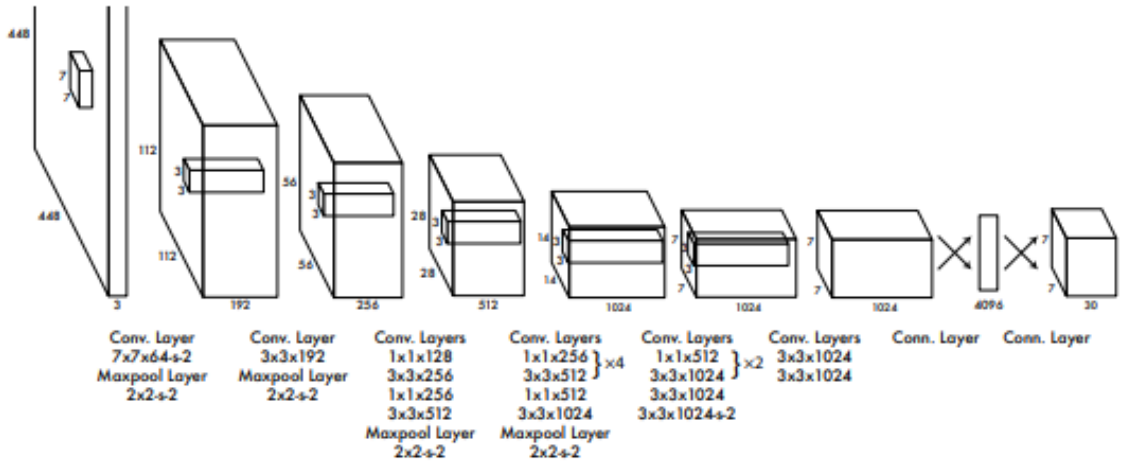


Figura 33: *Esquema de la arquitectura de YOLOv1 [53].*

- YOLOv2:
  También conocido como YOLO9000 [54], es un modelo de detección de objetos en tiempo real de una sola etapa. Algunos de los mayores avances con respecto a YOLOv1 es la adición de Darknet-19 como el "backbone", la normalización de los bloques (batch normalization), el uso de un clasificador de alta resolución y el uso de cajas de apoyo para la predicción de las cajas delimitadoras.

- YOLOv3:
  La versión 3 de este modelo [55] está construida a partir de YOLOv2 con una serie de mejoras como el nuevo backbone Darknet-53 que utiliza conexiones residuales así como algunas mejorías en las etapas de predicción de cajas delimitadoras y el uso de 3 diferentes escalas de las que extraer las características (similar a una FPN).

- YOLOv4:
  Este modelo [6] se centra más en la obtención de altas velocidades de operación en los detectores de objetos y la optimización de computación en paralelo que en reducir el indicador teórico de computación, el cual se mide en miles de millones de operaciones en punto flotante (BFLOPs). Para ello utiliza como backbone el CSPDarknet53, y se basa mayoritariamente en YOLOv3. También utiliza las llamados Bag of Freebies, que son un conjunto de técnicas que cambian la estrategia y el coste de entrenamiento, y Bag of Specials, que contienen diferentes plugins y módulos post-procesado que pueden mejorar drásticamente la precisión del detector.

Las siguientes versiones de YOLO no han sido utilizadas por la falta de soporte en TensorFlow 2 al comienzo de la comparativa, aunque merecen ser nombradas por su desempeño.

- YOLOv5:
  YOLO v5 [56] es diferente de todas las versiones anteriores, ya que se trata de una

implementación de PyTorch en lugar de una partición del Darknet original. Al igual que YOLO v4, YOLO v5 tiene un backbone CSPDarknet53. Las principales mejoras incluyen el aumento de los datos de mosaico y el aprendizaje automático de los anclajes de las cajas delimitadoras.

- YOLOX:
  YOLOX [84] es un detector de objetos de una sola etapa que realiza varias modificaciones a YOLOv3 con una el mismo backbone: DarkNet53. Se han modificado las estrategias previas para mejorar el rendimiento de YOLOX mediante la adición de MixUp y Mosaic [84] y se han eliminado los mecanismos de anclaje. Para el etiquetado se ha utilizado SimOTA, que a su vez sirve para mejorar la precisión.

- PP-YOLO:
  PP-YOLO [85] es un modelo basado en YOLOv3. Su principal objetivo es la implementación de un detector de objetos con un balanceo de efectividad y eficiencia que pudiera ser aplicado en escenarios reales más que servir como una nueva propuesta de modelo de detección. Entre sus cambios se incluye el reemplazo del backbone Darknet53 con un backbone ResNet y el incremento del batch_size de 64 a 192 (como suma de batch_sizes mínimos de 24 en 8 GPUs).

3.4.2. Modelos específicos de detección

FACED

La idea principal de este detector de caras es construir la red más pequeña posible para poder ejecutarse en tiempo real en una CPU sin perder precisión [64, 86].

Otros detectores de objetos habituales necesitan una cantidad masiva de obtención de características para poder clasificar cada caja delimitadora a su clase apropiada, lo que se traduce en una gran cantidad de parámetros a manejar, muchísimos filtros y una gran cantidad de capas. Es decir, son redes grandes.

Este modelo de detección evita la necesidad de clasificar cada caja delimitadora ya que solo hay una clase disponible, lo que además significa que se necesitarán un menor número de características a aprender, que se traduce en una red más pequeña. Las redes pequeñas necesitan, por tanto, menor computación y serán más rápidas. Faced es una unión de dos redes neuronales.

La primera es una implementación customizada de una red neuronal completamente convolucional basada en YOLO. Esta toma imágenes RGB de tamaño 288x288 y de salida aporta una malla de 9x9 (más pequeña comparada a la malla de YOLO de 13x13) donde cada celda puede predecir las cajas delimitadoras y la probabilidad de una cara. La figura 34 muestra una representación de la primera parte de la red.

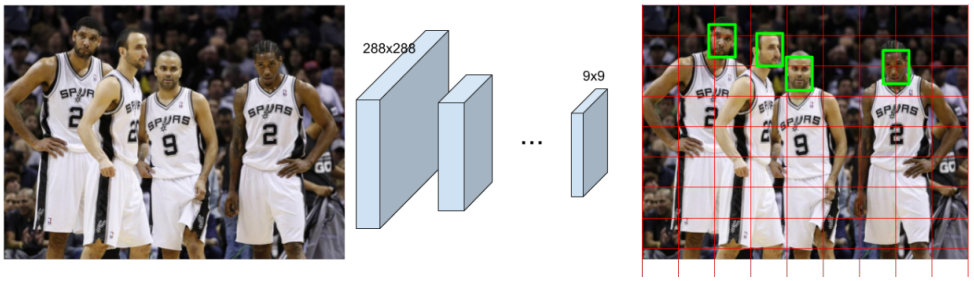


Figura 34: *Entrada y salida de la arquitectura principal de Faced [86].*

La segunda es una CNN estándar customizada que se utiliza para tomar rectángulos que contengan la cara y predigan la caja delimitadora de esta. Este es un paso de corrección y es necesario porque la salida de la red principal no es tan acertada como se esperaba. En la figura siguiente se ve este paso extra necesario para obtener correctamente las características de la cara.

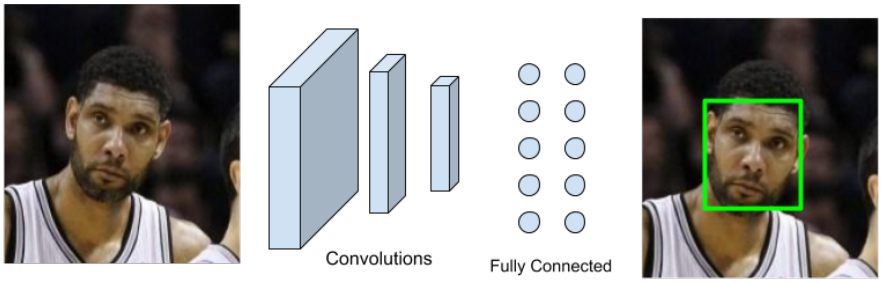


Figura 35: *Entrada y salida de la segunda red de Faced [86].*

UltraLight Face Detection

Este modelo [65] de detección facial fue diseñado para aplicaciones de reconocimiento facial de propósito general para dispositivos con una computación no tan potente y es aplicable tanto a teléfonos Android, como iOS, como a PC. Para el proceso de entrenamiento de este detector se utilizó un dataset generado por WIDER FACE [8] y un tamaño imágenes de entrada de 640x640, igual que con los anteriores modelos.

El modelo viene con una versión Slim con una simplificación que lo hace ligeramente más rápido, y una versión RFB [87] con un módulo RFB modificado para mayor precisión. El tamaño por defecto del archivo del modelo es de 1.1 MB, lo que significa que su tiempo de inferencia será muy reducido también.

- **RFB**

Un Receptive Field Block o RFB [87] es un módulo de refuerzo de las características profundas aprendidas de los modelos CNN ligeros para que así puedan contribuir a detectores rápidos y precisos. Específicamente, RFB es un bloque convolucional con múltiples ramas. Su estructura interna se divide en dos componentes: La capa convolucional con múltiples ramificaciones y diferentes kernels y unas capas convolucionales finales expandidas.

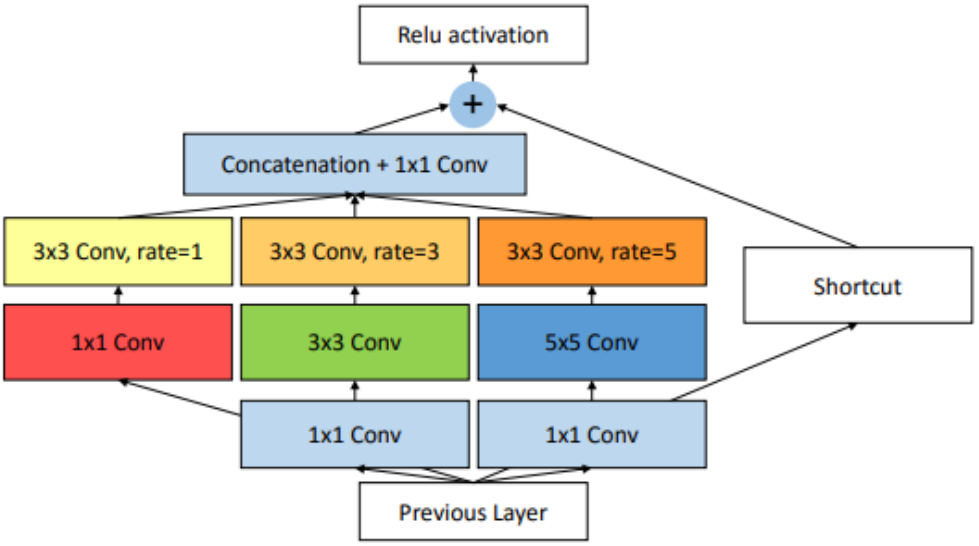


Figura 36: *Arquitectura del primer componente de un RFB [87].*

La capa convolucional con multiples ramificaciones es una manera simple y natural de aplicar diferentes kernels para conseguir RFs (Receptive Fields) de múltiples tamaños, los cuales se supone que deben ser mejor que los RFs que tienen un tamaño fijo.

Las capas convolucionales expandidas pretenden generar mapas de características de una mayor resolución, capturando información en un área más grande con mayor contexto manteniendo el mismo número de parámetros. Esto fue posteriormente adaptado a modelos con gran reputación como puede ser SSD [57].

3.5. Herramientas de desarrollo

3.5.1. TensorFlow

TensorFlow [7] es una plataforma de extremo a extremo creada por el equipo de Google Brain, que facilita tanto la creación como la implementación de modelos de aprendizaje automático.

Ofrece un ecosistema completo para ayudar al usuario a resolver problemas complejos del mundo real.

TensorFlow ofrece varios niveles de abstracción al usuario y mediante la API de Keras, se pueden crear y entrenar modelos de manera sencilla en servidores, dispositivos perimetrales o en la web de independientemente del lenguaje utilizado o la plataforma de ejecución. A su vez, posee varias herramientas:

- TensorFlow Extended (TFX): Para conseguir la canalización del aprendizaje automático de producción completa.
- TensorFlow Lite: Para ejecutar la inferencia en dispositivos móviles y perimetrales.
- TensorFlow.js: Permite el entrenamiento e implementación de modelos en entornos de JavaScript.

3.5.2. TensorFlow Lite

TensorFlow Lite es un conjunto de herramientas de que pretenden permitir a los usuarios, ejecutar sus modelos de TensorFlow en dispositivos móviles, incorporados o programables. Este consta de dos componentes principales: El intérprete, que ejecuta modelos optimizados en varios dispositivos hardware diferentes, destacando entre ellos teléfonos móviles, dispositivos Linux incorporados y microcontroladores. Y el conversor, cuyo objetivo es "convertir modelos de TensorFlow en un formato eficiente para que los use el intérprete y además, puede implementar optimizaciones para mejorar el tamaño y el rendimiento de los objetos binarios" [9].

TensorFlow Lite se diseñó con la intención de mejorar los métodos de aprendizaje automático de los dispositivos en "el perímetro" de la red, ya que a menudo tienen memoria o potencia computacional limitada, y así evitar su dependencia con los servidor. A mayores, permite mejorar los modelos en varios aspectos:

- Latencia: debido a que, si no se necesita un llevar a cabo la conexión con un servidor, no habrá envío y recibimiento de datos.
- Privacidad: al no enviar dichos datos, la información no sale del dispositivo y no hay posibilidad de interceptarla.
- Conectividad: no se requiere una conexión a Internet.
- Consumo de energía: si bien es cierto que las conexiones de red necesitan mucha energía, al no estar manteniendo una, no habrá problemas de consumos extra.

A los modelos se les puede aplicar varias optimizaciones para que sean ejecutados con estas restricciones. Hay varias razones por las que se deberían optimizar:

- Para reducir el tamaño: Los modelos más pequeños tienen algunos beneficios como un tamaño de almacenamiento menor, un menor tiempo necesario para su descarga o un menor uso de la memoria RAM al ejecutarse. [88]

- La reducción de latencia en este caso se refiere al tiempo necesario en ejecutar una única inferencia. Algunas formas de optimización permiten reducir la cantidad de cálculo necesario para ejecutar la inferencia mediante un modelo. Esta también puede ser una de las causas de un mayor consumo energético.

- Para conseguir una mejor compatibilidad con el acelerador. Esto puede ser necesario para aceleradores de hardware como Edge TPU[1] [89].

- Las compensaciones pueden provocar cambios en la precisión del modelo. Por ejemplo, un modelo entrenado para la reducción del tamaño o la latencia, perderá una pequeña cantidad de precisión o, en casos poco habituales, podría suponer una pequeña mejora de esta.

3.5.3. DarkNet

Darknet es un entorno de trabajo para redes neuronales de código abierto para C y CUDA. Es rápido, fácil de instalar y permite la computación en CPU y GPU. Algunas de sus herramientas son:

- YOLO: Algoritmo de detección de objetos en tiempo real.

- Clasificación en ImageNet: Permite clasificar imágenes con modelos populares como ResNext.

- Tiny Darknet: Es una versión simplificada de Darknet, utilizada para la clasificación de imágenes.

3.6. Técnicas de simplificación

Actualmente, TensorFlow Lite admite la optimización de modelos a través de la cuantificación, la poda y la agrupación en clústeres.

Estos son parte del kit de herramientas de optimización de modelos de TensorFlow, que proporciona recursos para técnicas de optimización de modelos que son compatibles con TensorFlow Lite. Las técnicas principales de simplificación son:

[1]La aceleración por hardware se refiere al proceso por el cual una aplicación descarga ciertas tareas de computación en componentes de hardware especializados dentro del sistema. Esto que permite una mayor eficiencia que si se ejecutara en una CPU de propósito general.

- **Cuantificación:** Funciona reduciendo la precisión de los números utilizados para representar los parámetros de un modelo, que por defecto son números de coma flotante de 32 bits. Esto supone como resultado un tamaño de modelo menor y una velocidad de cálculo mayor.
- **Poda:** Funciona eliminando parámetros dentro de un modelo que solo tienen un impacto menor en sus predicciones. Los modelos podados tienen el mismo tamaño en disco y poseen la misma latencia en tiempo de ejecución, pero su compresión es más eficaz. Esto hace que la poda sea una técnica bastante útil para reducir el tamaño de descarga del modelo.
- **Agrupación:** Funciona agrupando los pesos de cada capa en un modelo en un número predefinido de grupos y luego compartiendo los valores de centrales para los pesos que pertenecen a cada grupo individual. Esto reduce el número de valores de peso únicos en un modelo, reduciendo así su complejidad. Como resultado, los modelos agrupados se pueden comprimir de manera más eficaz, proporcionando beneficios de implementación similares a la poda.

Naturalmente, ante semejantes cambios, es habitual ver un compromiso de velocidad y tamaño a cambio de algo de precisión. Este compromiso será diferente para cada modelo ya que cada uno posee una estructura de capas diferente y es muy difícil pensar cómo afectará al modelo a priori. De hecho, en algunos casos no muy comunes, se puede observar una mejoría muy pequeña en cuanto a precisión se refiere.

En este TFG se ha utilizado la cuantificación como optimizador, dejando los otros tipos para líneas futuras de investigación.

3.7. Metodología general

La metodología para llevar a cabo la comparativa ha sido la siguiente:

Una vez elegidos los modelos y tras ser entrenados con 3000 imágenes en 2 GPUs de Nvidia nombradas en el apartado 1.5, han sido evaluados con un conjunto de 1000 imágenes del dataset de WIDER FACE [8]. Posteriormente, se repitió el procedimiento con los modelos de detección simplificados, utilizando las mismas métricas y aplicando diferentes optimizaciones.

De la evaluación se ha obtenido la puntuación o "score" de la detección (la confianza con la que el detector considera que el objeto recuadrado es una cara) y los extremos laterales y superior e inferior del recuadro. Con estos parámetros, se ha evaluado el Average Precision del modelo tanto en PASCAL-VOC [90, 68, 91] como en el COCO dataset [75] (con la medida 0.5:0.05:0.95). Estos valores de precisión se han podido obtener a partir de una herramienta de

obtención de métricas de detección de objetos disponible en [92], cuya primera versión [93] solo aporta el mAP obtenido por PASCAL VOC.

En cuanto al otro parámetro, la media de los fps o frames (fotogramas) por segundo, ha sido obtenida al utilizar el detector de caras en un mismo vídeo para todos los modelos. La medida se ha calculado en cada uno de los frames dependiendo del tiempo de inferencia sobre la imagen, lo cual incluye la obtención de la imagen, su procesamiento, la obtención de las bounding boxes o cajas delimitadoras y la aplicación de estas en el fotograma de salida.

De esta manera se han obtenido las dos métricas principales utilizadas a lo largo de este Trabajo de Fin de Grado: Velocidad y precisión.

Capítulo 4

Comparativa de modelos de detección

Tras la elección de los datasets a utilizar en la comparativa, se pretende en este capítulo analizar qué modelos de detección de objetos serán utilizados para su entrenamiento como detectores faciales atendiendo a sus características. Este capítulo ilustra la toma de decisiones para tal fin y algunos de los problemas que ello ha supuesto. Se comienza con la toma de decisión de los modelos a utilizar y por último, se muestran los resultados obtenidos de cada uno de ellos tras su entrenamiento y simplificación.

4.1. Justificación de la decisión de los modelos genéricos a comparar

Los modelos disponibles para este trabajo y los que han sido empleados para hacer esta comparativa pertenecen en su mayoría al Zoo de modelos de detección de TensorFlow 2 [5]. Estos modelos han sido entrenados previamente con el dataset de COCO 2017 [75] y en este caso se reentrenarán para su uso como detectores de caras.

En [5] aparecen varios modelos repetidos con la diferencia de que algunos de ellos poseen dimensiones de entrada diferentes, a partir de las cuales, los valores de fps y mAP mostrados varían. Estas dimensiones afectan sólo a las imágenes de entrada, por lo que, una vez introducida una imagen en el detector, éste la redimensionará para poder trabajar correctamente con ella. Dicha redimensión será llevada a cabo mediante una interpolación y/o un diezmado bidimensional. Parece lógico pensar que cuanto mayores sean las dimensiones de la imagen una vez llevado a cabo el procesamiento, menos rápido será el modelo. Esto será debido a que tendrá que trabajar con un mayor número de valores, y por consiguiente, llevar a cabo un número aún más grande de operaciones en punto flotante o FLOPS. También se puede pensar que con imágenes más grandes la precisión será mayor debido a que será más sencillo encontrar los objetos a detectar. Esto último es lógico, ya que las propias métricas de precisión proporcionan valores de AP para tamaños de objeto grandes, medianos y pequeños, siendo estos últimos los que tienen peor

precisión en la mayoría de ocasiones.

Para ayudar en la elección de los métodos a emplear, se generó un diagrama de dispersión a partir de todos los datos de precisión y velocidad encontrados en [5]. Este diagrama se muestra en la figura 37.

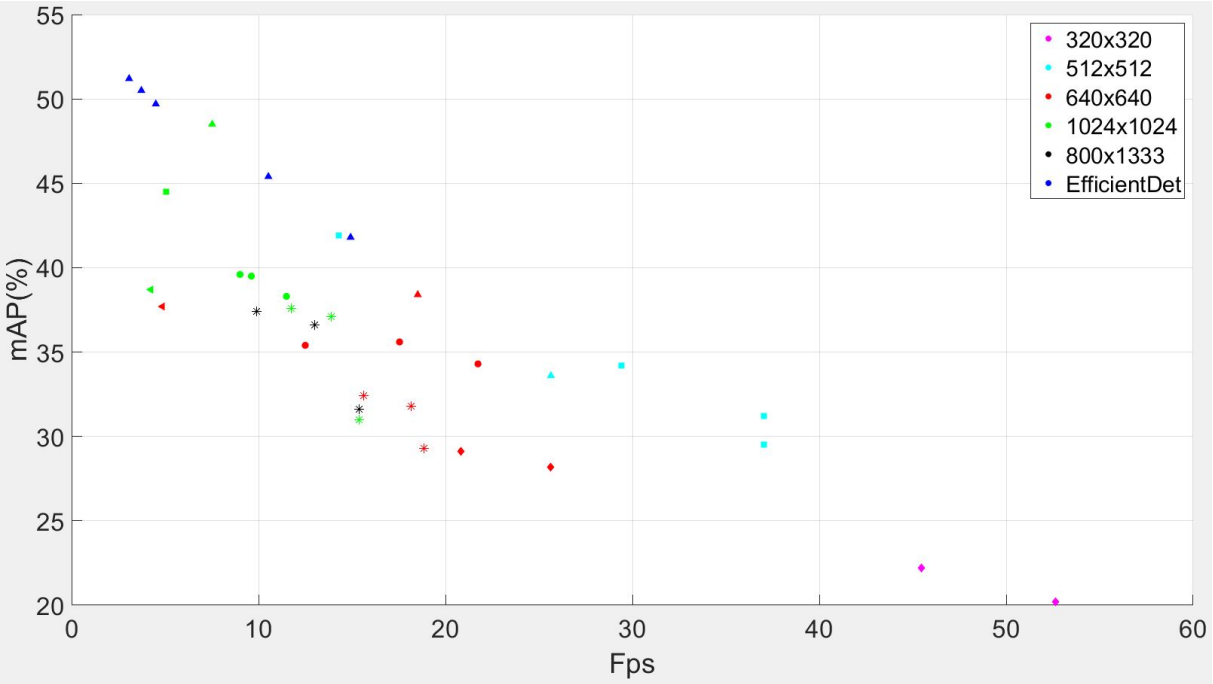


Figura 37: *Diagrama de dispersión con los diferentes modelos del zoo de TF. En la leyenda los diferentes colores hacen referencia a las dimensiones de los modelos añadiendo un color exclusivo para EfficientDet [77] debido a que no entraban agrupados en ningún otro conjunto de dimensiones. Las diferentes formas hacen referencia a los diferentes modelos.*

En la figura anterior se debe indicar de cara a su interpretación que los valores de velocidad serán mejores hacia la derecha del gráfico y la precisión será mejor hacia arriba.

La mayor cantidad de modelos está agrupada entre los 5 y los 30 fotogramas por segundo, de los cuales, los modelos con dimensión 640x640 parecen tener buena combinación de velocidad y precisión. Los modelos con dimensión de imagen 512x512 son algo más rápidos, pero tienen peor precisión, mientras que los de 1024x1024, aunque algo más precisos en general, son bastante más lentos. Con la finalidad de comparar la mayor cantidad de modelos y sus diferentes backbones, se decidió escoger los modelos con dimensiones 640x640.

Con los modelos ya escogidos, se elaboró otro diagrama de dispersión que muestra los diferentes modelos más en detalle. Este diagrama se muestra en la figura 38.

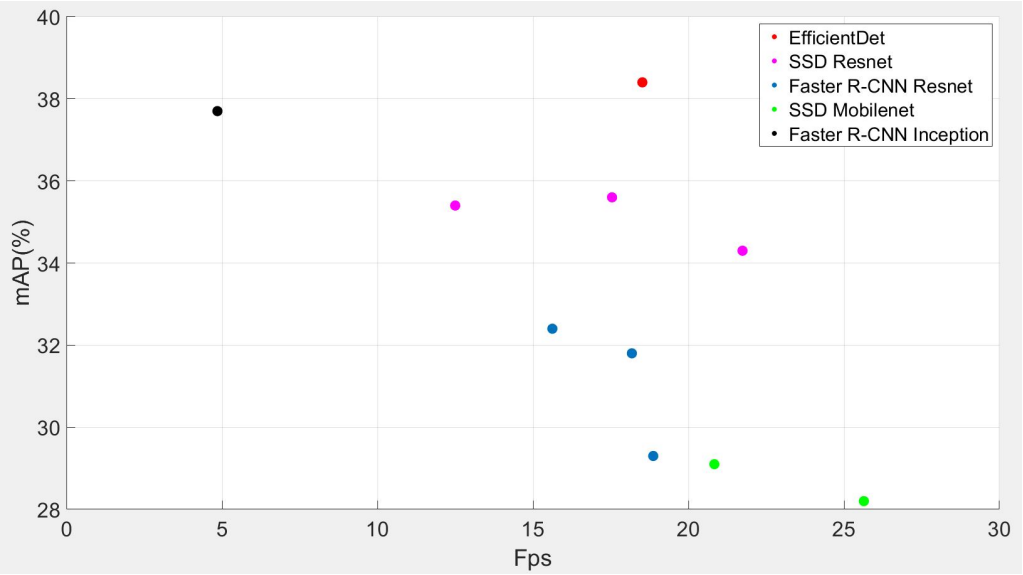


Figura 38: *Diagrama de dispersión con los datos teóricos de los modelos de [5] de tamaño 640x640.*

Los modelos genéricos obtenidos de [5] son los siguientes:

- EfficientDet D1 640x640
- SSD MobileNet V1 FPN 640x640
- SSD MobileNet V2 FPNLite 640x640
- SSD ResNet50 V1 FPN 640x640 (RetinaNet50)
- SSD ResNet101 V1 FPN 640x640 (RetinaNet101)
- SSD ResNet152 V1 FPN 640x640 (RetinaNet152)
- Faster R-CNN ResNet50 V1 640x640
- Faster R-CNN ResNet101 V1 640x640
- Faster R-CNN ResNet152 V1 640x640
- Faster R-CNN Inception ResNet V2 640x640

Además de todos estos modelos, se incluirá YOLOv4 en la comparativa de los modelos de detección genéricos por ser uno de los modelos más eficientes y uno de los más utilizados en la actualidad.

4.2. Entrenamiento y comparativa de los modelos de detección genéricos

Para todos y cada uno de los modelos genéricos entrenados, se utilizó la API de detección de objetos de TensorFlow 2, a partir de la cual se obtienen las herramientas necesarias para el entrenamiento de los modelos a excepción del modelo YOLOv4, cuyo entrenamiento fue a través de Darknet. Pero antes de dar los resultados finales se debe indicar cuál es el procedimiento de cálculo de las métricas de velocidad y precisión.

La propia API de detección de objetos de TensorFlow, permite disponer de herramientas para la evaluación de la precisión de los modelos entrenados de esa manera, pero, debido a que ciertos modelos no disponían de la capacidad de llevar a cabo esa evaluación concreta, fue necesario buscar otro método.

Este nuevo método consiste en una interfaz gráfica creada por Rafael Padilla en 2021 [92]. Dicha herramienta permite obtener una serie muy amplia de métricas de evaluación entre las que se incluyen la métrica principal utilizada para los desafíos de COCO: AP@(0.5:0.05:0.95) y la métrica para los desafíos de PASCAL VOC: mAP con un umbral de 0.5. Para que funcione correctamente esta herramienta, se debe disponer de las imágenes a evaluar y sus anotaciones ground truth, es decir, las anotaciones que servirán como referencia para medir los resultados. Después de llevar a cabo la detección y obtener correctamente sus anotaciones, el programa puede calcular estas métricas como se indica en el apartado 3.3.

En cuanto a la medida de la velocidad de ejecución, el método seguido para evaluar los modelos de detección, ha sido igual para todos. Primeramente se modifica el script correspondiente de cada uno de los modelos para que estos sean capaces de procesar un vídeo, y una vez hecho eso, se ejecuta el script. Este inicialmente carga el modelo y, para cada uno de los frames, marca el instante de tiempo inicial, obtiene la imagen, la procesa e intenta detectar todas las caras de la imagen, después mide el tiempo nuevamente. Una vez hecho esto, se calcula el tiempo de procesado por imagen para cada uno de los fotogramas del vídeo. Por último, se espera a que termine el vídeo y se calcula la media de las frecuencias como:

$$FPS = \frac{Numero\ de\ frames}{Tiempo\ total} \tag{7}$$

para posteriormente mostrarlo por pantalla.

En la tabla 1 se ven los resultados de los modelos entrenados empleando las 2 GPUs nombradas en el apartado 1.5: Nvidia Quadro RTX 5000 y Nvidia GeForce GTX 1070 de 16 y 8 GB respectívamente. Cabe destacar que, para el cálculo de las diferentes métricas, se han utilizado paralelamente las 2 GPUs disponibles.

	Postentrenamiento			Preentrenamiento
Modelo	Fps	COCO AP (%)	PASCAL AP (%)	COCO AP (%)
SSD MobileNet V1 FPN	9.09	19.39	29.63	29.1
SSD MobileNet V2 FPNLite	9.48	18.87	29.39	28.2
SSD ResNet50 V1 FPN	7.81	16.79	25.28	34.3
SSD ResNet101 V1 FPN	6.64	16.19	24.55	35..6
EfficientDet D1 640x640	6.77	1.77	3.40	38.4
Faster R-CNN ResNet50 V1	4.60	18.72	37.46	29.3
Faster R-CNN ResNet101 V1	4.24	23.12	40.96	31.8
Faster R-CNN Inception ResNet V2	1.35	22.26	45.31	37.7
YOLOv4	8.37	32.26	57.96	–

Tabla 1: *Resultados de la comparativa entre modelos genéricos entrenados para detección de caras.*

Tanto en la tabla 1 como en el diagrama de dispersión que se muestra en la figura 39 se puede observar como la mayoría de modelos de la API de TensorFlow que pertenecen a una misma familia, comparten una precisión muy similar en cualquiera de las dos métricas, ya sea PASCAL [68], o COCO [75].

Las diferencias de valores entre los resultados de COCO y PASCAL tienen sentido debido a que, para que PASCAL acepte una detección como válida, solo tiene que superar el umbral de 0.5. Mientras que para COCO, se tienen que superar 10 umbrales: desde 0.5 hasta 0.95 con intervalos crecientes de 0.05 y después llevar a cabo la media de estos valores resultantes, obteniendo unos valores de precisión por debajo de los del otro método.

La figura 39 presenta los resultados de la tabla 1 en un diagrama de dispersión.

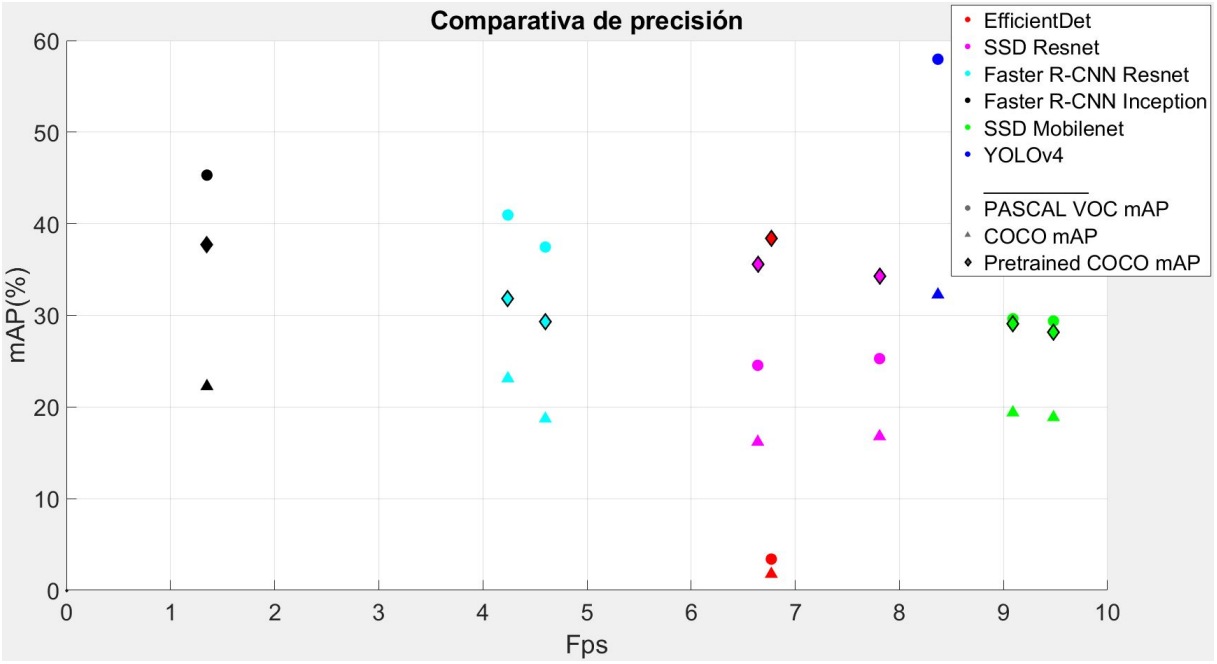


Figura 39: *Diagrama de dispersión con los modelos del zoo de TF antes de su entrenamiento (indicados con rombos) y después de él (indicados con círculos y triángulos dependiendo de la métrica). También aparecen incluidos los resultados de YOLOv4.*

En la figura 39 se toma de referencia la velocidad de ejecución de los modelos después del entrenamiento de detección de caras para que las diferencias de precisión se vean claramente. Las diferencias de velocidad no se comparan debido a que el método utilizado para la medición de este parámetro llevado a cabo en [5] no queda claro, por lo que a cabo una comparativa no sería justo.

La figura 39, muestra como la precisión se ha visto algo reducida tras el entrenamiento. En este aspecto destaca el modelo EfficientDet, el cual ha sufrido la mayor reducción de precisión. Los demás modelos, poseen una precisión similar en el preentrenamiento y en la métrica PASCAL del postentrenamiento, mientras que está ligeramente empeorada en la métrica COCO del postentrenamiento.

Las figuras 40(a) a 40(i) muestran la detección de los diferentes modelos en una imagen arbitraria después de su entrenamiento.

(a) *Detección del modelo SSD MobileNet V1.*

(b) *Detección del modelo SSD MobileNet V2 FPNLite.*

(c) *Detección del modelo SSD ResNet50.*

(d) *Detección del modelo SSD ResNet101.*

(e) *Detección del modelo EfficientDet D1.*

(f) *Detección del modelo Faster R-CNN ResNet50.*

Figura 40: Detección de los diferentes modelos genéricos entrenados parte 1.

(g) *Detección del modelo Faster R-CNN ResNet101.*

(h) *Detección del modelo Faster R-CNN Inception.*

(i) *Detección del modelo YOLO v4.*

Figura 40: Detección de los diferentes modelos genéricos entrenados parte 2.

Se debe indicar que las caras mostradas en verde en las figuras anteriores son las consideradas detecciones ground-truth, mientras que los recuadros rojos son los que, ha detectado el modelo, es decir, son su respuesta para esa imagen de entrada.

El modelo SSD Mobilenet, a pesar de estar indicado para teléfonos móviles y dispositivos computacionalmente menos potentes, posee un gran desempeño con respecto a la precisión y la velocidad, siendo esta última la que más destaca. No existe una gran diferencia entre la versión 1 y la 2 en cuanto a precisión, no obstante, sí que hay un aumento de velocidad por parte de la segunda versión.

Con SSD Resnet ocurre lo mismo que con el modelo SSD Mobilenet. Entre las 2 versiones no hay una diferencia muy extensa con respecto a la precisión, pero cuando hablamos de velocidad de inferencia, sí que se ve una mejoría en el modelo con 50 capas. Esto puede ser debido a la estructura del modelo, ya que, a menor cantidad de capas atravesadas para obtener una detección, más rápidos serán los resultados, mientras que para una cantidad mayor, tendrá que haber un procesamiento mayor, y, por consiguiente, trabajará de manera más lenta.

El modelo EfficientDet consigue un 3.4 % de precisión con la métrica de PASCAL [68] y un 1.77 % con la métrica de COCO ante las imágenes detectadas, por lo que su eficiencia queda en entredicho en este aspecto. La velocidad, no obstante, es bastante superior a los modelos de Faster R-CNN, y algo mejor incluso que el modelo SSD ResNet 101 FPN.

Faster R-CNN tiene indiscutiblemente la mejor precisión de todos los modelos comparados hasta ahora, pero su gran punto débil es evidente ante los demás, su velocidad de trabajo. El modelo Faster R-CNN ResNet50 posee una velocidad de trabajo superior a sus homónimos, por la misma razón que se ha explicado en el modelo SSD ResNet50, pero a cambio posee una precisión menor que los modelos subsiguientes. La versión de 101 capas es la más equilibrada de las 3 versiones de Faster R-CNN. Con una precisión superior a todos los demás modelos, es superado en precisión únicamente por Faster R-CNN Inception y YOLOv4.

Por último llegamos al modelo más nuevo de los aquí presentes, la versión 4 de YOLO, que consigue una precisión superior a todas las demás con bastante holgura sea cual sea la métrica escogida. Además, la velocidad de ejecución es muy superior a casi todos los modelos, solo siendo superado por los modelos SSD MobileNet.

4.3. Comparativa de los modelos de detección específicos

No olvidemos que uno de los objetivos de esta comparativa es, además de ser capaz de encontrar un modelo eficiente y fiable de detección de caras, poder utilizar estos mecanismos de detección de caras en dispositivos y hardwares menos potentes, por lo que cuanto menos pesado sea un modelo, más posibilidades habrá de que eso sea posible. Este es el caso del siguiente modelo a comparar, cuyo nombre completo es “Ultra-Light-Fast-Generic-Face-Detector-1MB”, pero al que se ha nombrado reiteradas veces en este documento como Ultra Light Face Detector [65].

Estos detectores pesan aproximadamente 4.5 MB para TensorFlow, lo que significa que son candidatos muy viables para una rápida velocidad de ejecución. Fueron entrenados con el dataset de 3000 imágenes de WIDER FACE al igual que los demás modelos, y, como previamente habían sido entrenados con WIDER FACE, puede que sus resultados se vieran ligeramente afectados debido a una situación de overfitting, la cual haría que presentaran peores resultados al intentar detectar una cara como ya se ha explicado previamente.

Este detector de caras ultra ligero [65] presenta dos posibles versiones, una Slim, que lleva a cabo una simplificación del backbone de red, lo que lo hace ligeramente más rápido, y la versión RFB, que ofrece una mayor precisión gracias al módulo de refuerzo RFB.

Por otro lado, el modelo FACED [64] pretende conseguir una simplificación directa respecto a los demás detectores de objetos. Esto es gracias a la idea de que, por lo general, los detectores necesitan una cierta cantidad de potencia computacional únicamente dirigida a clasificar la detección en su clase correspondiente. Si se eliminara esa variable y se llevara a cabo una simplificación para solo tener una posible clase, se estarían reduciendo las operaciones necesarias.

Las figuras 41(a), 41(b) y 41(c) muestran la detección de estos 3 modelos tras su entrenamiento.

(a) *Detección del modelo UL Face Detector RFB.* (b) *Detección del modelo UL Face Detector Slim.*

(c) *Detección del modelo FACED.*

Figura 41: Detección de los diferentes modelos específicos entrenados.

Por último, la figura 42 muestra la comparativa de todos los modelos de detección entre los que se incluyen los genéricos y específicos. La tabla 2 muestra de un vistazo los valores de precisión y velocidad de todos los modelos tras su entrenamiento.

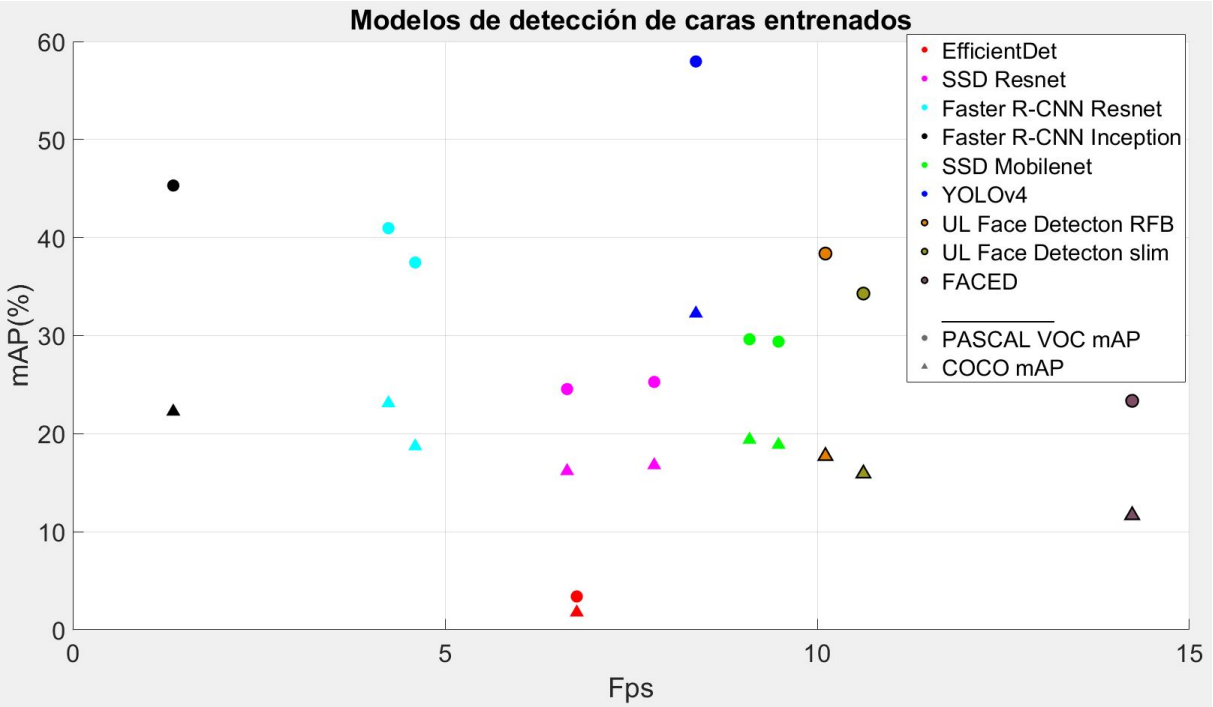


Figura 42: *Diagrama de dispersión con los diferentes modelos entrenados.*

Modelo	Fps	COCO AP (%)	PASCAL AP (%)
SSD MobileNet V1 FPN 640x640	9.09	19.39	29.63
SSD MobileNet V2 FPNLite 640x640	9.48	18.87	29.39
SSD ResNet50 V1 FPN 640x640 (RetinaNet50)	7.81	16.79	25.28
SSD ResNet101 V1 FPN 640x640 (RetinaNet101)	6.64	16.19	24.55
EfficientDet D1 640x640	6.77	1.77	3.40
Faster R-CNN ResNet50 V1 640x640	4.60	18.72	37.46
Faster R-CNN ResNet101 V1 640x640	4.24	23.12	40.96

Modelo	Fps	COCO AP (%)	PASCAL AP (%)
Faster R-CNN Inception ResNet V2 640x640	1.35	22.26	45.31
YOLOv4 640x640	8.37	32.26	57.96
Ultra Light Face Detection RFB	10.11	17.72	38.37
Ultra Light Face Detection slim	10.62	15.93	34.29
FACED	14.23	11.68	23.35

Tabla 2: *Resultados de la comparativa para modelos sin simplificar.*

En la tabla 2 observamos que el modelo UL Face Detector RFB es más preciso que su contraparte, mientras que la variante slim posee mayor velocidad de ejecución. Ambos dos poseen una precisión aceptable, aunque siguen siendo superados por el detector YOLO y la familia de Faster R-CNN.

Por otro lado, el modelo FACED, a pesar de no tener una gran precisión, es hasta el momento, el detector más rápido de todos superando incluso a la versión Slim de UltraLight Face Detection. Además es mucho más rápido que cualquiera de los modelos genéricos obtenidos en [5].

Una vez completado el entrenamiento y evaluación de los modelos genéricos y específicos para detección facial, se procederá a una simplificación de estos para la comparación entre ellos mismos y los modelos sin simplificar. Estos detectores ocuparán mucho menos espacio y sus aplicaciones podrán estar más enfocadas a dispositivos con poca capacidad computacional (Android, iOS, etc.). Dicha simplificación será llevada a cabo con las herramientas disponibles de TensorFlow Lite.

4.4. Comparativa de los modelos de detección simplificados

Se llevó a cabo una simplificación de los modelos de los puntos 4.2 y 4.3, pero, debido a un problema de falta de soporte en TensorFlow Lite para algunos de ellos, no se pudieron simplificar.

Debido a esto, la conversión de [5] se llevó a cabo únicamente en las variantes de los modelos SSD, con backbones ResNet y MobileNet, a los que, por simplicidad, se les denominará de ahora en adelante SSD ResNet101/50 Lite o SSD MobileNet v1/2 Lite. También se simplificaron los modelos YOLOv4 y las 2 variantes de UltraLight Face Detection. El formato resultante de la conversión es float32. Este formato es el empleado por defecto en las conversiones al formato de TensorFlow Lite y por consiguiente, es la simplificación menos profunda.

Cabe destacar que, a diferencia de los modelos sin simplificar, los simplificados necesitan ser invocados para su aplicación en una imagen o vídeo. Dicha invocación será más o menos costosa dependiendo del tamaño del objeto invocado, por lo que, modelos de 100 MB, serán mucho más lentos que aquellos que pesen 10 MB. Esta información hay que tenerla en cuenta desde este punto en adelante, debido a que la velocidad de estos modelos viene determinada por este factor, por el formato del modelo y por el tipo de optimización.

Posteriormente se procedió a la optimización de estos modelos. Dicha optimización fue llevada a cabo mediante una cuantificación posterior al entrenamiento, que funciona reduciendo la precisión de los números utilizados para representar los parámetros de un modelo. Se llevaron a cabo dos optimizaciones:

- Float16: La conversión de pesos a valores de punto flotante de 16 bits da como resultado una reducción a la mitad en el tamaño del modelo. Algunos hardware, como las GPU, pueden computar de forma nativa en esta aritmética de precisión reducida, logrando una aceleración con respecto a la ejecución tradicional de punto flotante. Un modelo convertido a pesos float16 aún se puede ejecutar en la CPU sin modificaciones adicionales: los pesos float16 se muestrean a float32 antes de la primera inferencia, lo que permite una reducción significativa en el tamaño del modelo a cambio de un impacto mínimo en la latencia y la precisión.

- Int8: Es una estrategia de optimización que convierte números de coma flotante de 32 bits (como pesos y salidas de activación) a los números de coma fija de 8 bits más cercanos. Esto da como resultado un modelo más pequeño y una mayor velocidad de inferencia, lo que es valioso para dispositivos de baja potencia como los microcontroladores. La única diferencia es que este modelo está optimizado para ejecutarse en CPU en lugar de GPU, por lo que algunos resultados podrían ser peores que el original.

  Se debe recordar que este tipo de optimización, no tiene como objetivo reducir la velocidad de inferencia en GPU, sino permitir que los dispositivos funcionen en dispositivos con poca capacidad de procesamiento, puesto que a un procesador le será más sencillo trabajar antes con valores enteros que con coma flotante. Esto es debido a cómo son utilizados los valores enteros o de punto flotante dentro del ordenador. Un entero es utilizado como un valor binario verdadero potencia de 2, generalmente reservando un valor para el signo.

Mientras que un número en coma flotante necesita el bit de signo, un exponente y una mantisa para ser formado [94].

Tras la conversión al formato TF Lite, los modelos sufrieron cambios en su peso, precisión y velocidad. Estos valores se pueden observar en las tablas 3 y 4.

	No simplificados	float32	float16	int8		No simplificados	float32	float16	int8
Modelo	COCO AP (%)					PASCAL AP (%)			
SSD MobileNet v1	19.39	15.56	15.56	15.33		29.63	26.50	23.51	23.50
SSD MobileNet v2	18.87	15.25	15.26	15.24		29.39	23.17	23.16	23.16
SSD ResNet50	16.79	12.12	12.12	12.12		25.28	18.39	18.37	18.19
SSD ResNet101	16.19	11.44	11.43	11.38		24.55	17.40	17.39	17.38
YOLOv4	32.26	32.25	32.25	31.96		57.96	57.96	57.93	57.58
UL Face Det RFB	17.72	16.58	16.59	16.75		38.37	33.72	33.74	33.69
UL Face Det Slim	15.93	14.75	14.76	14.73		34.29	30.01	30.09	30.28

Tabla 3: *Tabla comparativa de la precisión de los modelos antes y después de su simplificación.*

La tabla 3 muestra un descenso generalizado de la precisión de los modelos entrenados tras la simplificación. Los modelos más afectados son los pertenecientes a [5], mientras que YOLO no ha resultado afectado prácticamente nada en ninguna de las métricas para ninguna de las simplificaciones. Los modelos de UltraLight Face Detection no han sufrido apenas disminución de la precisión de una simplificación a la siguiente, no obstante, sí que se observa una diferencia notable de los modelos sin simplificar a los simplificados.

Por lo observado en la tabla 3, queda demostrado que, al simplificar un modelo, apenas se va a ver afectada su precisión.

La tabla 4 muestra las diferentes velocidades de ejecución de los modelos tras su simplificación y en comparación muestra también sus pesos.

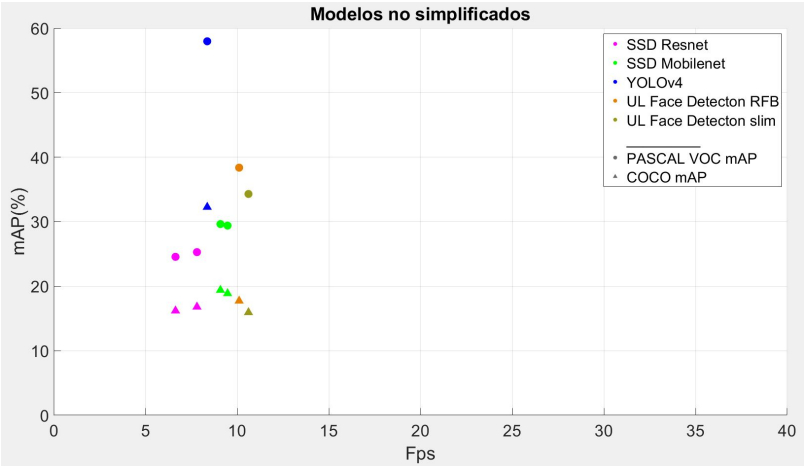
	No simplificados	float32	float16	int8		No simplificados	float32	float16	int8
Modelo	Fps					Peso en MB			
SSD MobileNet v1	9.09	1.00	0.94	0.0063		7.5	114.2	57.2	29.7
SSD MobileNet v2	9.48	3.15	2.93	0.10		10.5	11.6	5.9	4.0
SSD ResNet50	7.81	0.61	0.62	0.0043		11.0	193.3	96.7	49.8
SSD ResNet101	6.64	0.45	0.49	0.0032		18.1	265.7	133.0	68.5
YOLOv4	8.37	0.25	0.25	0.15		11.0	244.1	122.2	61.4
UL Face Det RFB	10.11	32.96	31.04	4.60		4.8	1.1	0.6	0.41
UL Face Det Slim	10.62	35.80	34.23	6.15		4.4	1.0	0.55	0.38

Tabla 4: Tabla comparativa de la velocidad y peso de los modelos antes y después de su simplificación.

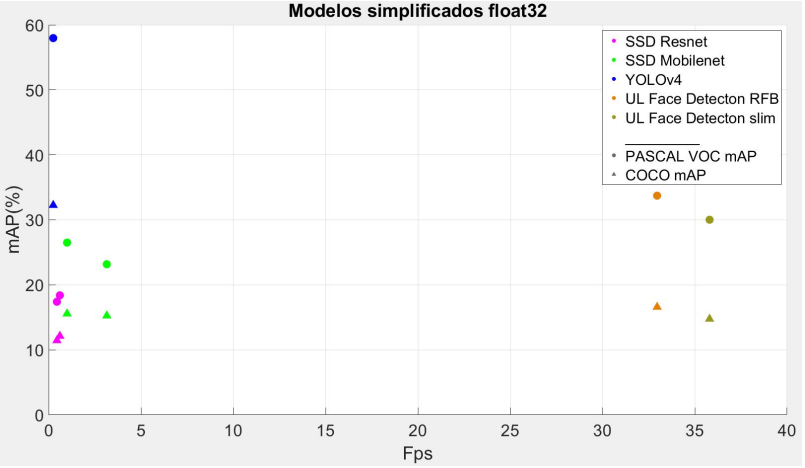
Se puede observar en la tabla 4 como los modelos sin simplificar no tienen una dependencia clara entre el peso y la velocidad, pero si se miran con detenimiento los valores de peso y velocidad de los modelos simplificados en float32, se puede intuir una relación. A mayor peso del modelo, menor velocidad, y viceversa. Los modelos que más destacan son las variantes de UltraLight Face Detection por su increíble velocidad. Por otro lado, una vez se lleva a cabo la conversión a float16, se esperaría un aumento de velocidad acompañando a la disminución del tamaño de los modelos, pero esto no parece suceder, sino que en algunos casos, incluso disminuye.

Por último, cabe recordar que la optimización en int8 no está pensada para ser ejecutada en GPU, por lo que esas velocidades son las relativas a la ejecución de los detectores de caras en la CPU del equipo (Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30GHz), por lo que sus resultados tan bajos al medir la velocidad tienen sentido.

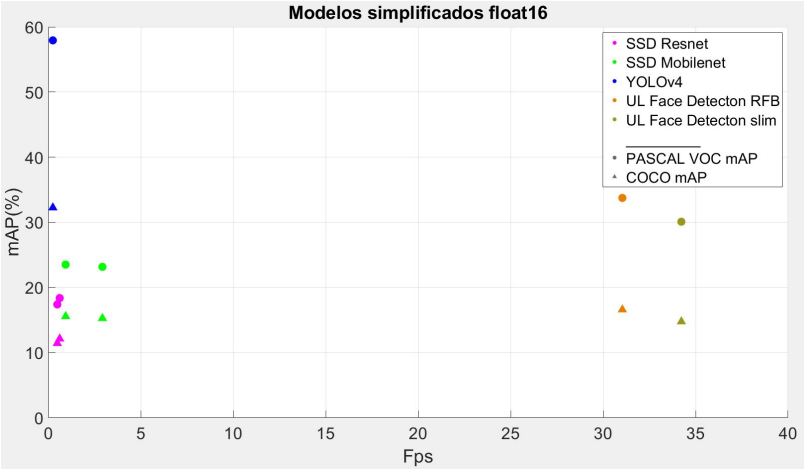
La figura 43 muestra una comparativa gráfica de los modelos de detección simplificados y sin simplificar, tras las optimizaciones pertinentes.



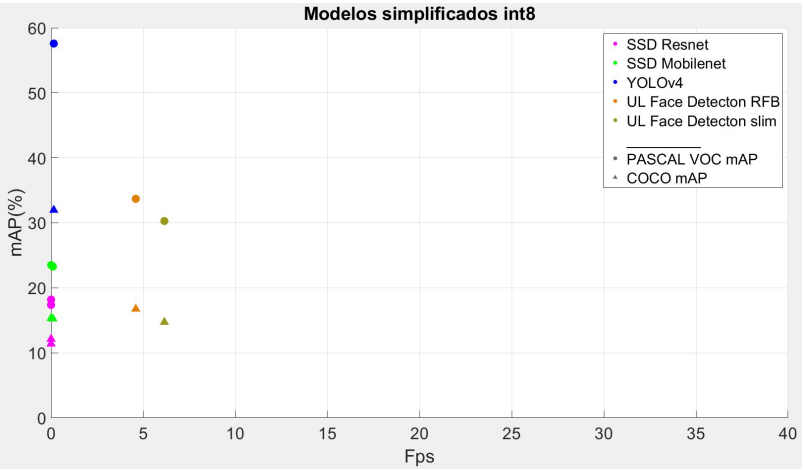
(a) *Diagrama de dispersión de los modelos sin simplificar.*



(b) *Diagrama de dispersión de los modelos simplificados en el formato float32.*



(c) *Diagrama de dispersión de los modelos en el formato float16.*



(d) *Diagrama de dispersión de los modelos en el formato int8.*

Figura 43: *Comparativa de los diferentes modelos antes de la simplificación y después de esta para las diferentes optimizaciones.*

En la figura 43 se puede observar, al igual que se comentó previamente, como la precisión permanece casi intacta para todos los detectores mientras que la velocidad es la gran afectada. Se observa mejor como, los modelos de UltraLight Face Detection, poseen una velocidad muy superior a los demás una vez han sido simplificados y optimizados. Por ello, para cualquier dispositivo computacionalmente poco potente, la recomendación sería emplear cualquiera de las versiones del detector de caras ultraligero.

Capítulo 5

Conclusiones

En relación al objetivo general comentado, en este trabajo de fin de grado se ha presentado una comparativa de modelos de detección genéricos y específicos entrenados para el reconocimiento facial eficiente en el espectro visible. Así como una comparativa de los mismos modelos tras su simplificación y posterior optimización.

Para ello, inicialmente se ha realizado un estudio de los algoritmos basados en Deep Learning, empleados habitualmente en tareas de detección de objetos. Tras su análisis, se ha optado por la comparación de estos modelos para la obtención de un grupo final sobre el que realizar los experimentos objeto del presente TFG.

Posteriormente, se ha llevado a cabo el entrenamiento de estos algoritmos de la manera más justa posible con las herramientas disponibles en la Object Detection API de TensorFlow para finalmente llevar a cabo la simplificación de los modelos disponibles.

Por último, se han evaluado estos algoritmos de reconocimiento facial mediante una serie de pruebas con la finalidad de anotar los resultados fruto de la evaluación, y proceder al análisis final de estos.

El entrenamiento llevado a cabo en los modelos de detección ha aportado un conjunto de resultados a partir de los cuales se puede percibir cuales de los modelos pueden funcionar como mejores detectores.

De los modelos entrenados destaca la versión 4 del modelo YOLO por su gran precisión en la inferencia, y el modelo FACED por su gran velocidad en comparación con los demás. Y, aunque la diferencia de velocidad es algo grande entre estos dos, YOLO sobresale ante los demás por ser el modelo más completo de todos, consiguiendo resultados destacables en los dos parámetros

medidos: Velocidad y exactitud en la detección.

De la simplificación de los modelos al formato float32 con las herramientas de TensorFlow Lite, destacan claramente los modelos de UltraLight Face Detection por su gran velocidad de inferencia, la cual, gracias a su baja latencia, pueden procesar imágenes en tiempo real. A su vez, la precisión de YOLO destaca por encima de todos los demás, esta vez siendo altamente empeorado por su tiempo de procesamiento por imagen.

Por ello, la comparativa de los modelos optimizados en float32 muestra que el modelo más eficiente para la detección facial en dispositivos computacionalmente poco potentes, es posiblemente el modelo UltraLight Face Detection versión RFB, ya que, a tales velocidades se busca más una precisión mayor que la posibilidad de procesar unos pocos más frames por segundo.

La simplificación al formato float16 devuelve unos resultados bastante similares a float32 siendo la reducción de los tamaños de modelos, el parámetro a destacar. Al igual que el anterior caso, el modelo UltraLight Face Detection sobresale de entre todos los demás gracias a su gran velocidad de inferencia que, tras la optimización, apenas ha perdido precisión.

Los resultados tras la simplificación de los modelos a int8 reflejan la velocidad a la que un microcontrolador o un dispositivo poco potente procesarían las imágenes de las cuales detectar una cara. Nuevamente UltraLight Face Detection sería el modelo óptimo para utilizar bajo estas características, ya que, aunque su velocidad de inferencia se haya visto reducida, sigue poseyendo unos resultados bastante mejores que los demás.

A pesar de todo ello, se pueden destacar las siguientes conclusiones sobre los procesamientos llevados a cabo en los modelos de detección:

- **Entrenamiento:** A pesar de ser el paso más largo de todos, es el más importante, puesto que será el que permita que un modelo obtenga unos resultados buenos o mediocres. Es importante entrenar con un batch_size grande, para así obtener mejores resultados siempre sin perder de vista las gráficas de pérdidas totales.
- **Conjuntos de imágenes:** Obtener un buen set de entrenamiento y evaluación es clave para conseguir buenos resultados. Generalmente, a mayor similitud entre el dataset de entrenamiento escogido y las imágenes que se encontrará el modelo en situaciones reales, mejores resultados se obtendrán. Por ello se concluye que la correcta elección de los sets de entrenamiento y validación son muy relevantes para un buen desempeño.
- **Detectores genéricos frente a específicos:** Los resultados muestran claramente que los modelos creados para una tarea concreta, tienen generalmente un mejor desempeño que los demás en cuanto a velocidad se refiere. No obstante, el paso más importante para obtener los mejores resultados posibles es el entrenamiento del modelo. En esto se incluye

el asegurar que se evitan las situaciones de overfitting y controlar que no se detenga el entrenamiento prematuramente. Por otro lado, un modelo genérico bien entrenado, como se ha podido observar, puede poseer una precisión mayor que un modelo creado para ese fin.

- **Optimización:** La optimización utilizada es muy útil para reducir el tamaño del modelo sin perder apenas precisión de detección. El formato float32 es el estándar utilizado en la simplificación del modelo a la versión Lite, no obstante, el formato float16 presenta unos resultados casi iguales con tamaños en el modelo la mitad de grandes. Por otro lado, el formato int8 está pensado concretamente para dispositivos en los que los cálculos con punto flotante sean muy costosos. En caso de tener un dispositivo algo potente, float16 sería el formato ideal para la simplificación del modelo escogido. En caso contrario, int8 deberá ser el formato escogido para que el dispositivo pueda llevar a cabo la detección.
- **Modelos finales:** Los detectores que mejores resultados han presentado son YOLOv4 por su gran precisión, FACED por su velocidad y UltraLight Face Detection en los modelos optimizados por su gran velocidad de procesamiento y precisión aceptable.

Finalmente, observando el desempeño de los algoritmos con una visión global, se puede concluir que, aunque existe cierto margen de mejora, todos los modelos son funcionales y cumplen su labor correctamente en la mayoría de ocasiones. Si bien es cierto que hay algunos modelos con un mejor desempeño que otros, cualquier modelo de los presentes en este documento serviría aceptablemente para un trabajo de exigencia media.

5.1. Líneas futuras

A continuación se esbozan algunas líneas futuras cuyo desarrollo podría llevarse a cabo en un estudio futuro debido a que no se han podido explorar con suficiente detenimiento durante el desarrollo de este documento:

- Optimización de los modelos con los tipos restantes nombrados en el apartado 3.6.
- Combinación de los diferentes tipos de optimización en un mismo modelo con objeto de determinar cual es la combinación más eficiente.
- Estudio en profundidad de la combinación óptima entre pasos en el entrenamiento de los modelos y el batch_size introducido para diferentes hardwares.
- Comparativa en detalle de los modelos de detección de una misma familia con diferentes tamaños de entrada.
- Aplicación de los modelos utilizados en el espectro infrarrojo.
- Aplicación de los mejores modelos en microcontroladores para conseguir una detección eficiente.

- Pruebas de los modelos en escenarios reales.

Bibliografía

[1] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I, 2001.

[2] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. *DeepFace: Closing the Gap to Human-Level Performance in Face Verification*. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, 2014.

[3] Dr Qaim Rizvi. A review on face detection methods. *Journal of Management Development and Information Technology*, 11, 02 2011.

[4] Klemen Grm, Vitomir Štruc, Anaïs Artiges, Matthieu Caron, and Hazim Ekenel. Strengths and weaknesses of deep learning models for face recognition against image degradations. *IET Biometrics*, 7, 09 2017.

[5] Vighnesh Birodkar. Tensorflow Object Detection Model Zoo. `https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md`, 2021.

[6] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv, 2020.

[7] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. Software available from tensorflow.org.

[8] Shuo Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. WIDER FACE: A Face Detection Benchmark. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[9] *Tensorflow lite*. https://www.tensorflow.org/lite/. Último acceso: agosto 2021.

[10] *Linux*. https://www.linux.org/. Último acceso: agosto 2021.

[11] *Anaconda documentation*. https://docs.anaconda.com/. Último acceso: agosto 2021.

[12] *Python*. https://www.python.org/. Último acceso: agosto 2021.

[13] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[14] *Keras*. https://keras.io/. Último acceso: agosto 2021.

[15] *PyCharm*. https://www.jetbrains.com/es-es/pycharm/. Último acceso: agosto 2021.

[16] *Top 10 Python IDE and Code Editors in 2020*. https://www.geeksforgeeks.org/top-10-python-ide-and-code-editors-in-2020/. Último acceso: agosto 2021.

[17] *12 Best Python IDEs And Code Editors In 2021*. https://www.softwaretestinghelp.com/python-ide-code-editors/. Último acceso: agosto 2021.

[18] *MATLAB*. https://es.mathworks.com/. Último acceso: agosto 2021.

[19] *LaTeX*. https://www.latex-project.org/. Último acceso: agosto 2021.

[20] *Putty*. https://www.putty.org/. Último acceso: agosto 2021.

[21] *TigerVNC*. https://tigervnc.org/. Último acceso: agosto 2021.

[22] François Chollet. *Deep Learning with Python*. Manning, November 2017.

[23] D. Hinestroza Ramírez. *El Machine Learning a través de los tiempos, y los aportes a la humanidad*. Universidad Libre Seccional Pereira, 2018.

[24] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[25] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[26] Aurlien Gron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 1st edition, 2017.

[27] *Definition of Machine Learning*. https://www.ieeesmc.org/technical-activities/cybernetics/machine-learning. Último acceso: agosto 2021.

[28] A. Burkov. *The Hundred-Page Machine Learning Book*. Andriy Burkov, 2019.

[29] Paul John Werbos. *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*. Wiley-Interscience, USA, 1994.

[30] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning Representations by Back-propagating Errors. *Nature*, 323(6088):533–536, 1986.

[31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM*, 60(6):84–90, May 2017.

[32] *Getting Started With CNN*. https://medium.com/@kinisanketh/getting-started-with-cnn-18c03efc7d06, 2020. Último acceso: septiembre 2021.

[33] *A Beginner's Guide To Understanding Convolutional Neural Networks Part 2*. https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/, 2016. Último acceso: septiembre 2021.

[34] Hossein Gholamalinezhad and Hossein Khosravi. Pooling methods in deep neural networks, a review. arXiv, 2020.

[35] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. arXiv, 2015.

[36] *An Overview of ResNet and its Variants*. https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035. Último acceso: agosto 2021.

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

[38] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv, 2015.

[39] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature Pyramid Networks for Object Detection. arXiv, 2017.

[40] *A Gentle Introduction to the Rectified Linear Unit (ReLU)*. https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-network 2020. Último acceso: septiembre 2021.

[41] Sagar Sharma and Simone Sharma. Activation functions in neural networks. *Towards Data Science*, 6(12):310–316, 2017.

[42] Zhengxia Zou, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object Detection in 20 Years: A Survey. arXiv, 2019.

[43] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, 2005.

[44] Pedro Felzenszwalb, David McAllester, and Deva Ramanan. A discriminatively trained, multiscale, deformable part model. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

[45] Pedro F. Felzenszwalb, Ross B. Girshick, and David McAllester. Cascade object detection with deformable part models. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2241–2248, 2010.

[46] 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.

[47] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[48] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. arXiv, 2014.

[49] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *Lecture Notes in Computer Science*, page 346–361, 2014.

[50] Ross Girshick. Fast R-CNN. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015.

[51] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. arXiv, 2016.

[52] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. arXiv, 2016.

[53] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. arXiv, 2016.

[54] Joseph Redmon and Ali Farhadi. YOLO9000: Better, Faster, Stronger. arXiv, 2016.

[55] Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement. arXiv, 2018.

[56] Delong Qi, Weijun Tan, Qi Yao, and Jingfeng Liu. YOLO5Face: Why Reinventing a Face Detector. arXiv, 2021.

[57] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single Shot MultiBox Detector. *Lecture Notes in Computer Science*, page 21–37, 2016.

[58] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal Loss for Dense Object Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2):318–327, 2020.

[59] Gonzalo López Aguirre, María Guadalupe Treviño Alanís, Ismael Herrera Vázquez, Enrique Canchola Martínez, Sebastián Manuel Arteaga Martínez, Rufo Agenor Aguilar Tejada, Lorena Lucina Ocampo Tallavas, Óscar Díaz Flores, and Miguel Ángel Herrera Enríquez. *Manual de disecciones*. MC GRAW HILL INTERAMERICANA, 1º ed. edition, 2015.

[60] Adjabi Insaf, A. Ouahabi, Amir Benzaoui, and Abdelmalik taleb ahmed. Past, Present, and Future of Face Recognition: A Review. *Electronics*, 9:1188, 07 2020.

[61] Matthew Turk and Alex Pentland. Eigenfaces for Recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 01 1991.

[62] P.Jonathon Phillips, Harry Wechsler, Jeffery Huang, and Patrick J. Rauss. The feret database and evaluation procedure for face-recognition algorithms. *Image and Vision Computing*, 16(5):295–306, 1998.

[63] *About Face ID advanced technology*. `https://support.apple.com/en-us/HT208108`. Último acceso: agosto 2021.

[64] Ivan Itzcovich. faced. `https://github.com/iitzco/faced`, 2018.

[65] Linzai. Ultra-Light-Fast-Generic-Face-Detector-1MB. `https://github.com/Linzaer/Ultra-Light-Fast-Generic-Face-Detector-1MB`, 2020.

[66] *Reconocimiento facial en 3D para detectar enfermedades raras*. `https://www.elespanol.com/omicrono/software/20190412/usan-reconocimiento-facial-detectar-enfermedades-raras-ninos/390462238_0.html`. Último acceso: agosto 2021.

[67] *WIDER FACE: A Face Detection Benchmark*. `http://shuoyang1213.me/WIDERFACE/`, 2016. Último acceso: septiembre 2021.

[68] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[69] *PASCAL VOC2007 Example Images*. `http://host.robots.ox.ac.uk/pascal/VOC/voc2007/examples/index.html`, 2007. Último acceso: septiembre 2021.

[70] *Common Objects in Context*. `https://cocodataset.org/#home`, 2020. Último acceso: septiembre 2021.

[71] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? arXiv, 2014.

[72] Francesco Ponzio, Gianvito Urgese, Elisa Ficarra, and Santa Di Cataldo. Dealing with lack of training data for convolutional neural networks: The case of digital pathology. *Electronics*, 8:256, 02 2019.

[73] *Evaluating performance of an object detection model.* https://towardsdatascience.com/evaluating-performance-of-an-object-detection-m 2020. Último acceso: septiembre 2021.

[74] *Intersection over Union (IoU) for object detection.* https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/, 2016. Último acceso: septiembre 2021.

[75] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft COCO: Common Objects in Context. arXiv, 2015.

[76] *mAP (mean Average Precision) for Object Detection.* https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173, 2019. Último acceso: septiembre 2021.

[77] Mingxing Tan, Ruoming Pang, and Quoc V. Le. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[78] Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Xinyu Xiao, and Jian Sun. Detnas: Backbone search for object detection. arXiv, 2019.

[79] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. arXiv, 2020.

[80] Xin Lu, Quanquan Li, Buyu Li, and Junjie Yan. Mimicdet: Bridging the gap between one-stage and two-stage object detection. arXiv, 2020.

[81] *EfficientDet: Scalable and Efficient Object Detection.* https://medium.com/@nainaakash012/efficientdet-scalable-and-efficient-object-detection-ea05ccd28427, 2019. Último acceso: septiembre 2021.

[82] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv, 2015.

[83] Zhipeng Deng, Hao Sun, Shilin Zhou, Juanping Zhao, Lin Lei, and Huanxin Zou. Multi-scale object detection in remote sensing imagery with convolutional neural networks. *ISPRS Journal of Photogrammetry and Remote Sensing*, 145, 05 2018.

[84] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. Yolox: Exceeding yolo series in 2021, 2021.

[85] Xiang Long, Kaipeng Deng, Guanzhong Wang, Yang Zhang, Qingqing Dang, Yuan Gao, Hui Shen, Jianguo Ren, Shumin Han, Errui Ding, and Shilei Wen. Pp-yolo: An effective and efficient implementation of object detector, 2020.

[86] Ivan Itzcovich. faced: CPU Real Time face detection using Deep Learning. *towards data science*, sep 2018.

[87] Songtao Liu, Di Huang, and Yunhong Wang. Receptive field block net for accurate and fast object detection. arXiv, 2018.

[88] *Tensorflow lite Modelo Optimization*. `https://www.tensorflow.org/lite/performance/model_optimization`. Último acceso: agosto 2021.

[89] Amir Yazdanbakhsh, Kiran Seshadri, Berkin Akin, James Laudon, and Ravi Narayanaswami. An evaluation of edge tpu accelerators for convolutional neural networks, 2021.

[90] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The Pascal Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision*, 111(1):98–136, January 2015.

[91] Mark Everingham, Luc Gool, Christopher K. Williams, John Winn, and Andrew Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *Int. J. Comput. Vision*, 88(2):303–338, June 2010.

[92] Rafael Padilla, Wesley L. Passos, Thadeu L. B. Dias, Sergio L. Netto, and Eduardo A. B. da Silva. A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit. *Electronics*, 10(3), 2021.

[93] R. Padilla, S. L. Netto, and E. A. B. da Silva. A Survey on Performance Metrics for Object-Detection Algorithms. In *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 237–242, 2020.

[94] David González Ortega. *Apuntes asignatura Programación*, 2017-18.