Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Detección de eventos en partidos de fútbol utilizando técnicas de Aprendizaje Máquina

Autor: Juan Luis Verdugo Blanco
Tutor: Francisco José Simois Tirado

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Detección de eventos en partidos de fútbol utilizando técnicas de Aprendizaje Máquina

Autor:
Juan Luis Verdugo Blanco

Tutor:
Francisco José Simois Tirado
Profesor Contratado Doctor

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2024

Trabajo Fin de Grado: Detección de eventos en partidos de fútbol utilizando técnicas de Aprendizaje Máquina

Autor: Juan Luis Verdugo Blanco

Tutor: Francisco José Simois Tirado

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Sevilla, 2024

A mi familia

Agradecimientos

Aprovecho este espacio para agradecer a todas las personas que han estado conmigo en estos años de carrera. Mi familia, amigos, pareja, compañeros de clase, compañeros de piso, y en general todo aquel que ha querido compartir parte de su tiempo conmigo, haciendo que este periodo de tiempo se hiciese mucho más corto de lo que pensaba. Personas que han sabido darme los elogios que han creído justos, ánimos cuando creían que lo necesitaba y "apretarme las tuercas" cuando creían que era capaz de dar más de lo que estaba dando. A todas estas personas, muchas gracias por vuestro cariño, comprensión y exigencia.

Juan Luis Verdugo Blanco

Écija, Sevilla, 2024

Resumen

Este trabajo comienza describiendo de forma teórica los fundamentos de la Inteligencia Artificial, analizando su importancia en la actualidad y el poder que ha ido adquiriendo con el paso de los años. Además, se desarrollan distintos campos de interés, en especial aquellos relacionados con el Aprendizaje Máquina, haciendo hincapié en los algoritmos que se utilizan más adelante en la descripción de la solución planteada para la detección de eventos en partidos de fútbol.

En concreto, el objetivo de la solución que se va a desarrollar en el trabajo es la detección y clasificación de tres eventos distintos en el transcurso de un partido de fútbol. Estos tres eventos son: saques de banda, pases y disputas entre jugadores de distintos equipos. Además de esto, se va a desarrollar otra solución mediante la cual se realiza un seguimiento del balón, haciendo un seguimiento de su recorrido.

Para ello, se presentan diversas tecnologías como EfficientNet o YOLOv5, explicando las herramientas que ofrecen y las ventajas que presentan para el desarrollo de redes neuronales o el tratamiento de datos para su uso en dichas redes.

Con todo esto, en último lugar, se presentan los resultados obtenidos para estos modelos, para los que se hace un análisis centrado en tres métricas principales: recall, precisión y AP. Además, se comparan distintas posibles opciones de hiperparámetros para el modelo, justificando la opción elegida.

Abstract

This work begins with a theoretical description of the fundamentals of Artificial Intelligence, analyzing its current importance and the power it has been acquiring over the years. In addition, different fields of interest are developed, especially those related to Machine Learning, emphasizing the algorithms that are used later in the description of the solution proposed for the detection of events in soccer matches.

Specifically, the objective of the solution to be developed in the work is the detection and classification of three different events during a soccer match. These three events are: throw-ins, passes and disputes between players of different teams. In addition to this, another solution is going to be developed by means of which the ball is tracked, following its path.

For this purpose, various technologies such as EfficientNet or YOLOv5 are presented, explaining the tools they offer and the advantages they present for the development of neural networks or data processing for use in such networks.

Finally, the results obtained for these models are presented, with an analysis focused on three main metrics: recall, precision and AP. In addition, different possible options of hyperparameters for the model are compared, justifying the chosen option.

Índice

Agradecimientos **ix**

Resumen **xi**

Abstract **xiii**

Índice **xiv**

Índice de Figuras **xvi**

1 INTRODUCCIÓN **1**
1.1 Contexto y motivación *1*
1.2 Objetivos y alcance *1*
1.3 Estructura del trabajo *2*

2 INTELIGENCIA ARTIFICIAL **3**
2.1 ¿Qué es la IA? *3*
2.2 Orígenes y evolución de la Inteligencia Artificial *5*
2.3 Clasificación de la Inteligencia Artificial *5*
2.3.1 Clasificación según la capacidad de la IA 5
2.3.2 IA vs Machine Learning vs Deep Learning 7

3 MACHINE LEARNING **11**
3.1 ¿Qué es el Machine Learning? *11*
3.2 Tipos de sistemas Machine Learning *11*
3.2.1 Aprendizaje supervisado 11
3.2.2 Aprendizaje no supervisado 12
3.2.3 Aprendizaje semisupervisado 13
3.2.4 Aprendizaje por refuerzo 13
3.2.5 Aprendizaje Batch 14
3.2.6 Aprendizaje online 14
3.3 Datos *15*
3.3.1 Tipos de datos 15
3.3.2 Datos de imágenes, audio y vídeo 17
3.3.3 Preparación y obtención de los datos 18
3.3.4 Problemas asociados a los datos 18
3.3.5 Datos no representativos 20
3.4 Modelo y algoritmos *20*
3.4.1 Generalización y overfitting 22
3.5 Clasificación de problemas y algoritmos utilizados *23*
3.5.1 Clustering 23
3.5.2 Regresión 26
3.5.3 Clasificación 28
3.6 Métricas de evaluación de modelos *33*
3.6.1 Precisión 33
3.6.2 Recall o sensibilidad 33
3.6.3 Mean Average Precision (mAP) 34

4 REDES NEURONALES **35**

4.1 Orígenes y evolución de las redes neuronales *35*
4.2 Conceptos de redes neuronales *36*
4.3 Tipos de redes neuronales *38*
4.3.1 Perceptrón multicapa (MLP) 38
4.3.2 Redes neuronales recurrentes 39
4.3.3 Redes neuronales generativas antagónicas (GAN) 40
4.3.4 Redes neuronales convolucionales 40
4.4 Transfer Learning *44*
4.5 Modelos pre-entrenados *45*
4.5.1 Tf_efficientnet_b0 45
4.5.2 Resnet50 46
4.5.3 ViT Base Patch16 224 46
4.5.4 DPN68 47

5 DETECCIÓN DE EVENTOS EN PARTIDOS DE FÚTBOL **48**
5.1 Conjunto de datos *48*
5.2 Desarrollo del modelo *49*
5.2.1 Entorno de ejecución 49
5.2.2 Preparación de los datos 50
5.3 Función adicional: detección del balón *60*
5.3.1 Etiquetado de las imágenes 60
5.3.2 Modelo para la detección del balón 61
5.3.3 Preparación y entrenamiento 62

6 RESULTADOS **64**
6.1 Resultados para el modelo de detección de pases *64*
6.1.1 Comparación de resultados 64
6.2 Resultados del modelo para la detección del balón *73*

7 CONCLUSIÓN **83**
7.1 Líneas futuras *83*

ANEXO **84**

Referencias **94**

ÍNDICE DE FIGURAS

Figura 2-1. Estimación de la potencia de procesamiento para emular un cerebro humano [14] 6
Figura 2-2. Clasificación IA [17] 7
Figura 3-1. Ejemplo de aprendizaje supervisado [18] 12
Figura 3-2. Aprendizaje supervisado vs Aprendizaje supervisado [19] 13
Figura 3-3. Esquema de funcionamiento del aprendizaje por refuerzo [20] 14
Figura 3-4. Aprendizaje por lotes VS Aprendizaje online [21] 15
Figura 3-5. Tipos principales de datos en Machine Learning [22] 15
Figura 3-6. Datos numéricos. Continuos y discretos. [22] 16
Figura 3-7. Ejemplo de uso de series temporales [22] 16
Figura 3-8. Ejemplo de tensor tridimensional [23] 17
Figura 3-9. Respuesta de distintos algoritmos ante el mismo problema [26] 19
Figura 3-10. Fase de entrenamiento y test de un modelo de Machine Learning [28] 21
Figura 3-11. Ubicación de los hiperparámetros en el modelo [29] 21
Figura 3-12. Subajuste y sobreajuste [30] 22
Figura 3-13. Bias-Variance tradeoff [31] 23
Figura 3-14. Ejemplo de K-Means sencillo. [33] 24
Figura 3-15. Ejemplo de clustering usando DBSCAN [35] 26
Figura 3-16. Comparación del descenso por gradiente estocástico con el estándar [37] 28
Figura 3-17. Diferencia entre regresión y clasificación [38] 28
Figura 3-18. Ejemplo de clasificación binaria con gatos y perros [40] 29
Figura 3-19. Representación de la función sigmoide [41] 30
Figura 3-20. Representación de la curva de la función softmax [44] 31
Figura 3-21. Ejemplo de matriz de confusión 2 x 2 [45] 32
Figura 3-22. ROC y AUC [46] 33
Figura 3-23. Curva Precisión-Recall 34
Figura 4-1. Neurona de McCulloch-Pitts [51] 35
Figura 4-2. Red neuronal de dos capas que realiza el cálculo de la XOR [52] 36
Figura 4-3. Ejemplo de red neuronal artificial con cuatro capas [54] 37
Figura 4-4. Perceptrón multicapa totalmente conectado [55] 38
Figura 4-5. Red neuronal recurrente [57] 39
Figura 4-6. Neurona LSTM [58] 39
Figura 4-7. Red neuronal generativa antagónica [60] 40
Figura 4-8. Operación del kernel sobre la entrada [62] 41
Figura 4-9. Diferencia en la salida con stride 1 y stride 2 [63] 41

Figura 4-10. Padding para obtener el tamaño de la salida igual que el de la entrada [64] 42
Figura 4-11. Ejemplo de Max pooling con una ventana de tamaño 2x2 [61] 42
Figura 4-12. Función de activación ReLU [65] 43
Figura 4-13. Estructura de una red neuronal convolucional [61] 44
Figura 4-14. Entrenamiento desde cero VS Transfer Learning [67] 44
Figura 4-15. Arquitectura de bloques y capas de EfficientNet [77] 45
Figura 4-16. Arquitectura de resnet50 [84] 46
Figura 4-17. Ejemplo de un ViT (Vision Transformer) [87] 47
Figura 4-18. Ejemplo de DPN como combinación de ResNet y DenseNet 47
Figura 5-1. Ejemplo de evento en el fichero 'train.csv' 49
Figura 5-2. Plataforma Kaggle 49
Figura 5-3. Inicio del pase, evento 'play' 50
Figura 5-4. Fin del pase, evento 'play' 50
Figura 5-5. Inicio y final del evento 'play' 51
Figura 5-6. Inicio del evento 'throwin' 51
Figura 5-7. Fin del evento 'throwin' 51
Figura 5-8. Inicio y final del evento 'throwin' 52
Figura 5-9. Inicio del evento 'challenge' 52
Figura 5-10. Fin del evento 'challenge' 52
Figura 5-11. Datos correspondientes a un ejemplo de evento 'throwin' 53
Figura 5-12. Jugador sacando de banda 54
Figura 5-13. Fotograma correspondiente al evento 'pre_throwin' 55
Figura 5-14. Fotograma correspondiente al evento 'start_throwin' 55
Figura 5-15. Fotograma correspondiente al evento 'end_throwin' 56
Figura 5-16. Fotograma correspondiente al evento 'post_throwin' 56
Figura 5-17. Aplicación de las tolerancias al fichero 'train.csv' 57
Figura 5-18. Librería OpenCV 57
Figura 5-19. Herramienta de etiquetado VOTT 60
Figura 5-20. Ejemplo de etiquetado del balón mediante una caja delimitadora en VOTT 60
Figura 5-21. Exportación de los datos etiquetados a fichero CSV 61
Figura 5-22. Proceso de detección de objetos en YOLOv5 [80] 62
Figura 5-23. Arquitectura YOLOv5 [81] 62
Figura 5-24. Salida con el número de cajas delimitadoras y el nombre de la etiqueta 63
Figura 5-25. Comando para el entrenamiento del modelo 63
Figura 6-1. Evolución del Average Precision (AP) a lo largo de las épocas en función del modelo usado 64
Figura 6-2. Evolución del recall en función de la tolerancia 66
Figura 6-3. Evolución de la precisión en función de la tolerancia 67
Figura 6-4. Evolución del Average Precision (AP) en función de la tolerancia 67
Figura 6-5. Evolución del Average Precision (AP) a lo largo de las épocas para distintas tasas de aprendizaje

Figura 6-6. Evolución del Average Precision (AP) a lo largo de las épocas en función del optimizador usado 70

Figura 6-7. Evolución del AP para el evento 'challenge' a lo largo de las 40 épocas de entrenamiento con los distintos valores de tolerancia 70

Figura 6-8. Evolución del AP para el evento play a lo largo de las 40 épocas de entrenamiento con los distintos valores de tolerancia 71

Figura 6-9. Evolución del AP para el evento throwin a lo largo de las 40 épocas de entrenamiento con los distintos valores de tolerancia 71

Figura 6-10. Evolución del recall a lo largo de las 30 épocas de entrenamiento para los eventos 72

Figura 6-11. Evolución de la precisión a lo largo de las 30 épocas de entrenamiento para los eventos 72

Figura 6-12. Salida del proceso de entrenamiento 73

Figura 6-13. Evolución de la precisión por época 74

Figura 6-14. Evolución del recall por época 74

Figura 6-15. Evolución del mAP@0.5 por época 75

Figura 6-16. Uso del script detect.py 76

Figura 6-17. Salida del script detect.py 76

Figura 6-18. Fallo en la detección del balón en el aire 77

Figura 6-19. Fallo en la detección del balón entre jugadores 77

Figura 6-20. Detección del balón poco precisa 78

Figura 6-21. Detección del balón en el aire 78

Figura 6-22. Detección más precisa del balón 79

Figura 6-23. Detección del balón en vídeo (I) 79

Figura 6-24. Detección del balón en vídeo (II) 80

Figura 6-25. Detección del balón en vídeo (III) 80

Figura 6-26. Comparación de la evolución de la precisión en los modelos de 200 y 500 etiquetas 81

Figura 6-27. Comparación de la evolución del recall en los modelos de 200 y 500 etiquetas 81

Figura 6-28. Comparación de la evolución del mAP en los modelos de 200 y 500 etiquetas 82

Figura A-1. Parámetros del modelo 84

Figura A-2. Extracción de los modelos de timm 84

Figura A-3. Librerías necesarias para el modelo 85

Figura A-4. Tareas de preparación del fichero CSV y del conjunto de datos 86

Figura A-5. Función para extraer fotogramas de los vídeos 87

Figura A-6. Clase para crear la red neuronal 87

Figura A-7. Función para el entrenamiento del modelo 87

Figura A-8. Código para el bucle de entrenamiento y validación 88

Figura A-9. Red neuronal convolucional 1D 89

Figura A-10. Entrenamiento del modelo 89

Figura A-11. Clonación del repositorio YOLOv5 90

Figura A-12. Librerías necesarias 90

Figura A-13. Preparación de los datos para el modelo 90
Figura A-14. Lectura del fichero CSV 91
Figura A-15. División del conjunto de datos 91
Figura A-16. Creación del fichero YAML 91
Figura A-17. Función para ajusta el formato de las cajas delimitadoras 92
Figura A-18. Entrenamiento del modelo 93
Figura A-19. Detección del balón 93

1 INTRODUCCIÓN

La única manera de que no pierda su trabajo con la llegada de la IA es hacer algo que la IA no puede hacer, pero un humano sí, ser original.

- Abhijit Naskar -

1.1 Contexto y motivación

En la última década, el crecimiento de la Inteligencia Artificial, de aquí en adelante mencionada como IA, ha sido exponencial. Ha pasado de ser un tema únicamente usado dentro del ámbito tecnológico, visto por muchos como algo demasiado futurista, a estar en boca de todos. Esta evolución provoca que existan cada vez más herramientas que basan su funcionamiento en la IA y que cada vez más empresas se vean obligadas a incluirlas en su trabajo diario si no quieren ver reducida su productividad frente a sus competidores. En este escenario, el ámbito deportivo no es una excepción.

Si se pone el enfoque concretamente en el fútbol, deporte rey en España y en otros muchos países, todo se magnifica de forma sideral. En el fútbol, así como en otros deportes, cada vez se les da más importancia a los datos, siendo ya parte fundamental de la retransmisión de los partidos, sirviendo de apoyo para que los comentaristas puedan hacer llegar el mensaje de forma más certera al espectador. Esta enorme cantidad de datos también ha ganado mucha importancia para los clubes, que tratan de analizar el impacto que tiene un jugador sobre el terreno de juego en base a sus estadísticas o incluso justifican un fichaje de un jugador sobre otro en función de estas. Es decir, la capacidad de extraer conocimientos de este enorme conjunto de datos se presenta como una oportunidad única para el mundo del deporte en términos de análisis y comprensión del juego.

Esto lleva a la motivación del trabajo: realizar un acercamiento a las soluciones innovadoras que se presentan en este ámbito y a las distintas tecnologías que facilitan la aplicación de esta enorme cantidad de datos mediante técnicas de aprendizaje máquina.

1.2 Objetivos y alcance

El objetivo principal de este trabajo es, en primer lugar, esclarecer qué es la IA y cuál es su impacto en la sociedad actual, haciendo especial enfoque en el mundo del fútbol. A partir de esto, analizar algunas de las herramientas de aprendizaje máquina existentes para desarrollar un sistema que sea capaz de detectar eventos, concretamente pases, diferenciando entre disputas del balón entre jugadores, saques de bandas y pases estándar, en partidos de fútbol de forma automática y en tiempo real y que permita recoger estadísticas que serán de gran valor para los analistas.

Otros objetivos específicos del trabajo serán:

- Explorar las oportunidades que brinda la IA y el aprendizaje máquina a nivel general, identificando algoritmos y técnicas relevantes.
- Analizar distintos modelos además del implementado, comparando los resultados entre sí y destacando las diferencias entre ellos.
- Evaluar el rendimiento de la solución, sobre todo en términos de precisión.

1.3 Estructura del trabajo

El trabajo consta de tres partes diferenciadas. En la primera, se da un contexto general sobre la IA y se trata de dar una definición para la misma, pasando de forma teórica por los algoritmos más utilizados y haciendo hincapié en el aprendizaje máquina y en las redes neuronales convolucionales, que son las utilizadas para el caso particular de detectar eventos en un partido de fútbol.

En la segunda parte, se procede a aplicar de forma práctica estos conocimientos teóricos que se han desarrollado en la primera parte, desarrollando un modelo de red neuronal que se cumple con los objetivos citados en el apartado anterior.

Por último, se desarrolla un apartado de resultados, en el que se analiza la solución al problema y se reflexiona sobre su aplicación directa al ámbito del análisis futbolístico, exponiendo sus principales ventajas y desventajas, así como algunas posibles mejoras.

2 INTELIGENCIA ARTIFICIAL

En este capítulo se trata de dar una definición a la Inteligencia Artificial, pasando por los distintos tipos en los que se divide y exponiendo algunas aplicaciones que tiene a día de hoy. Además, se pone en contexto la evolución que ha tenido en los últimos años para entender su importancia en la actualidad, partiendo de cómo se originó y quiénes fueron sus principales valedores.

2.1 ¿Qué es la IA?

A pesar de que la IA ha pasado a ser protagonista en nuestras vidas, aún no existe una definición exacta que sea capaz de expresar todo lo que significa. Según Wikipedia, la Inteligencia Artificial es una disciplina y un conjunto de capacidades cognoscitivas e intelectuales expresadas por sistemas informáticos cuyo propósito es imitar la inteligencia humana para realizar tareas [1]. Según la definición de Nils Nilsson, uno de los pioneros de la IA, es la actividad de construir programas informáticos que realicen tareas que requieren inteligencia cuando se realizan por personas [2]. Como estas, existen muchas otras definiciones, firmadas por autores expertos en distintos ámbitos de la tecnología, pero gran parte de ellos coinciden en el mismo punto: el principal objetivo de la IA es automatizar tareas que realizan los humanos haciendo uso de su inteligencia.

En este punto, son muchas las preguntas que se pueden plantear en torno a ella: ¿qué es capaz de hacer una IA?, ¿qué limitaciones tiene?, ¿conforma un peligro para la sociedad?

En primer lugar, es importante conocer que la IA aparece en una gran variedad de campos que afectan al día a día de las personas: medicina, industria, sector financiero, investigación científica, deportes y un largo etcétera. Esto hace de ella una herramienta tremendamente versátil.

En concreto, sus principales **capacidades** son las siguientes [1] [3]:

- **Aprendizaje Máquina** (Machine Learning): Una IA es capaz de aprender patrones y generalizarlos para adaptarse a nuevas circunstancias y realizar mejoras en su rendimiento.
- **Procesamiento del Lenguaje Natural** (NLP): Tiene la habilidad de comprender, interpretar y generar lenguaje humano, permitiendo la comunicación.
- **Razonamiento lógico**: Es capaz de tomar decisiones basadas en la información que tiene almacenada, haciendo uso de reglas lógicas y siendo capaz de sacar sus propias conclusiones.
- **Visión por computador**: Tiene capacidad de comprender información procedente de imágenes o vídeos, reconociendo objetos en ellos, por ejemplo.
- **Reconocimiento de voz**: Es capaz de entender el habla humana, permitiendo así la interacción a través de la voz.
- **Robótica**: Tiene habilidad de manipular y mover objetos, interactuando con el entorno.

Mediante el uso de estas capacidades, una IA es capaz de **tomar decisiones**, **adaptarse a un cambio** en las circunstancias del problema y de **generar una solución creativa** a dicho problema planteado. Esto hace de ella una herramienta muy poderosa.

A pesar de las impresionantes capacidades que presenta, la IA tiene importantes limitaciones que la separan de la capacidad humana, sobre todo en términos de creatividad, pensamiento abstracto o entendimiento emocional. A destacar, los siguientes puntos suponen obstáculos importantes en las capacidades de una IA [4]:

- **Datos sesgados**: La principal forma de aprendizaje de una IA es a través de la información con las que sus programadores las alimentan en forma de datos de entrada. Esto provoca que, el resultado obtenido y la validez de este depende de forma irremediable de los datos que se le han proporcionado. Por ejemplo, se conoce que la empresa Amazon comenzó a utilizar una IA en el año 2014 para realizar contrataciones. Esta IA se entrenó con informes generados en los anteriores diez años. Gran parte de estos informes fueron escritos por hombres, lo que llevó a que el algoritmo excluyera a las mujeres del proceso de selección, pues llegó a la conclusión de que ser hombre era una característica prioritaria a la hora de buscar nuevos empleados [4].

- **Falta de ética y emociones**: Como bien es sabido, muchas de las decisiones que se toman en el mundo en el que vivimos están enormemente influenciadas por la ética y las emociones de las personas que las toman y de las personas que se ven afectadas por estas decisiones. Es por eso por lo que resulta difícil que una IA puede cumplir ciertas tareas que los seres humanos ejecutamos día a día, prácticamente sin darnos cuenta. Sin ir más lejos, las conexiones entre humanos, imprescindibles para poder llevar a cabo con éxito un trabajo en equipo, son imposibles de trasladar a un algoritmo [4].

  Existen varios ejemplos de casos carentes de ética y moralidad en relación con la IA. Por ejemplo, en 2015, una aplicación de Google colocaba etiquetas a las imágenes que sus usuarios tenían almacenadas en la galería de sus teléfonos móviles usando su propio software de IA. El gran problema relacionado con la aplicación fue que se estaba clasificando a usuarios de piel negra como gorilas, lo que hizo que la empresa fuera calificada de racista en redes sociales y tuviera que pedir disculpas por el evidente error que había cometido [5]. Otras muchas empresas han tenido problemas de este tipo, como Facebook, que ha recibido varias demandas por orientar sus anuncios de forma discriminatoria, sesgando por géneros, razas o edad.

  A la vista de estos casos, no se puede decir que una IA sea capaz de "sentir" o, en resumidas cuentas, de comportarse como un ser humano, aunque el futuro en este aspecto es incierto. En caso de conseguirlo, es muy probable que hubiera que regular su comportamiento estrictamente, aunque eso desembocaría en un debate filosófico muy extenso que en este punto no viene al caso.

  Por lo tanto, se llega a la conclusión de que una IA puede ejecutar muchas tareas que para los humanos serían prácticamente imposibles o increíblemente arduas de llevar a cabo, pero que, a la hora de actuar en situaciones que los humanos vemos como algo cotidiano o natural, los algoritmos encuentran dificultades para actuar de la misma manera que la haría cualquiera de nosotros.

- **Costes y seguridad**: Como último punto a resaltar, es importante hablar de los costes de utilizar la IA para una empresa que quiera implementarlo como base de su estructura de resolución de problemas. Estos costes incluyen el almacenamiento de una enorme cantidad de datos, la exigencia computacional que demanda el análisis de estos datos, el consumo de energía, mantenimiento, reparaciones… no es algo que se pueda pasar por alto [4]. Aun así, según la empresa McKinsey, en el año 2023, la mitad de las empresas encuestadas ya utilizan la IA de una u otra manera, denotando un importante incremento desde el año 2017, en el que solo el veinte por ciento la utilizaba [6].Además, en otra encuesta llevada a cabo por la misma empresa en el año 2019, el sesenta y tres por ciento de las empresas que utilizaban IA habían incrementado sus ganancias, y otro gran porcentaje había reducido sus costes en el campo que la aplicaban [4]. Esto indica que los costes de la adopción de la IA son muy relativos, dependiendo en gran medida de la rentabilidad que saque de su uso o la reducción de costes que pueda suponer.

  Otro coste que no puede pasar inadvertido y que puede conllevar grandes problemas es la seguridad. La seguridad abarca infinidad de situaciones que pueden afectar de forma muy grave, pudiendo poner en juego la vida de las personas. Por ejemplo, se conoce el caso de la muerte de una mujer atropellada por un vehículo autónomo en el año 2018, debido a que el coche no reconocía a los peatones que cruzan por lugares que no son los indicados [7]. Este tipo de casos ponen de manifiesto la importancia de invertir en seguridad, más aún si cabe con el crecimiento del Internet de las Cosas y las posibles vulnerabilidades que los atacantes tratarán de explotar.

2.2 Orígenes y evolución de la Inteligencia Artificial

La idea de IA nace en la década de 1950, de la mano de Alan Turing, Isaac Asimov y Claude Shannon. Como es natural, existen muchos nombres y avances importantes previos a la aparición de ellos, desde filósofos que desarrollaron la lógica y el pensamiento hasta científicos e informáticos que lograron que las prestaciones técnicas evolucionaran a pasos agigantados [8].

Empezando con Alan Turing, el británico publicó un artículo titulado "Computing Machinery and Intelligence", introduciendo una forma de medir la inteligencia de las máquinas. Esto deja claro que conocía de la importancia de este tema y de que su evolución en los siguientes años sería un punto de debate muy importante [9].

Por su parte, Isaac Asimov publicó "Three Laws of Robotics", donde describió desde un punto de vista ético el comportamiento de los robots y de los seres artificiales. Dichas leyes aún se tienen en cuenta en la actualidad [10].

Por último, Claude Shannon publicó un análisis sobre cómo programar un ordenador para que juegue al ajedrez, siendo pionero en el uso de algoritmos para el juego [11].

Tras esto, en 1956 tuvo lugar el "Dartmouth Summer Research Project", considerado el evento fundador de la IA como campo de estudio. John McCarthy fue el responsable de introducir el término Inteligencia Artificial.

En los siguientes años, tiene lugar un periodo que es conocido como el "Invierno de la IA" durante el cual los estudios en este campo se dejan de lado debido a los pobres resultados obtenidos.

Retrocediendo atrás, hasta 1943, Warren McCulloch y Walter Pitts abrieron el paradigma de las neuronas, entonces llamadas TLU (Threshold Logic Unit), que sientan la base de las redes neuronales que tanto se han desarrollado en los últimos años y a las que se le dedicará buena parte de este trabajo. Esto viene al caso pues, en 1969, Marvin Minsky y Seymour Papert publican su libro "Perceptrons", poniendo en duda la idea del modelo de Perceptron de las neuronas artificiales, derivando así en un nuevo periodo de poco interés investigativo en redes neuronales [8].

Ya a partir de 1975, se producen avances significativos que aún son claves hoy. En ese mismo año, Paul Werbos presenta el algoritmo de "Backpropagation", esencial para entrenar redes neuronales. Una década más tarde, en 1986, Rumelhart, Hinton y Williams continúan con la evolución de este algoritmo y realizan importantes avances para ajustar los pesos de las redes neuronales. Si se une esto con las mejoras en algoritmos de entrenamiento, la gran evolución computacional y la enorme cantidad de datos disponibles, da como resultado el paradigma que vivimos en la actualidad [8].

El paradigma actual es muy diverso, pero se ha centrado principalmente en el desarrollo de las Redes Neuronales Artificiales y, sobre todo, el Deep Learning. También cabe destacar el auge reciente de los chatbots y otros tipos de IA generativa, que está teniendo un impacto muy importante en la industria y en la sociedad, cambiando la forma en la que accedemos a la información y en la que trabajamos [8].

2.3 Clasificación de la Inteligencia Artificial

2.3.1 Clasificación según la capacidad de la IA

La IA se puede clasificar en dos tipos principales según su capacidad:

- **Inteligencia Artificial débil (IAD)**: Se refiere a sistemas diseñados para realizar tareas específicas, sin capacidad de generalización más allá de la tarea que ejecutan. Estos sistemas dependen en gran medida de los datos de entrenamiento y su rendimiento estará muy vinculado a la calidad de los dichos datos. Todos los sistemas de IA actuales pertenecen a esta categoría, pues funcionan correctamente sólo en determinados contextos. Algunos ejemplos pueden ser: reconocimiento facial, motor de búsquedas en Internet, Siri o Alexa [12].

- **Inteligencia Artificial fuerte o general (IAF o IAG)**: La IA fuerte es, en la actualidad, un nivel teórico por el que la IA presentaría una comprensión y habilidades equivalentes o superiores a las de los seres humanos. Es decir, tendría la capacidad de generalizar su conocimiento y aplicarlo en distintos campos o tareas, sin necesidad de estar explícitamente programada para uno de ellos. De alguna forma, se podría decir que la IA fuerte tendría conciencia propia y autonomía, lo que le permitiría adaptarse a las distintas problemáticas que tuviera que enfrentar sin la intervención humana [13].

En la siguiente imagen, se ve representada una estimación de cuánta potencia de procesamiento se necesita para emular un cerebro humano, realizada por Ray Kurzweil, Anders Sandberg y Nick Bostrom. Las predicciones de estos investigadores también se muestran en la gráfica. En el eje horizontal se representa el año en cuestión para el que se hace la estimación y en el eje vertical los FLOPS (Floating point operations per second), unidad que se utiliza para medir los cálculos matemáticos que puede hacer por segundo una CPU o una GPU, es decir, que mide el rendimiento de una computadora. La línea gris marca el crecimiento exponencial de la capacidad computacional, que se duplica cada 1.2 años, tomando como referencia la computadora más rápida del TOP500.

Kurzweil predice que la carga mental será posible mediante simulación neuronal, y que es posible que se consiga entre 2025 y 2030. Por su parte, Sandberg es menos seguro, y define muchos pasos intermedios antes la consecución de la conciencia, pues expresa dudas sobre de dónde surge la conciencia humana. Algunos de estos pasos hacen referencia a conjuntos de moléculas que dan vida al ser humano, como el metaboloma o el proteoma [13].

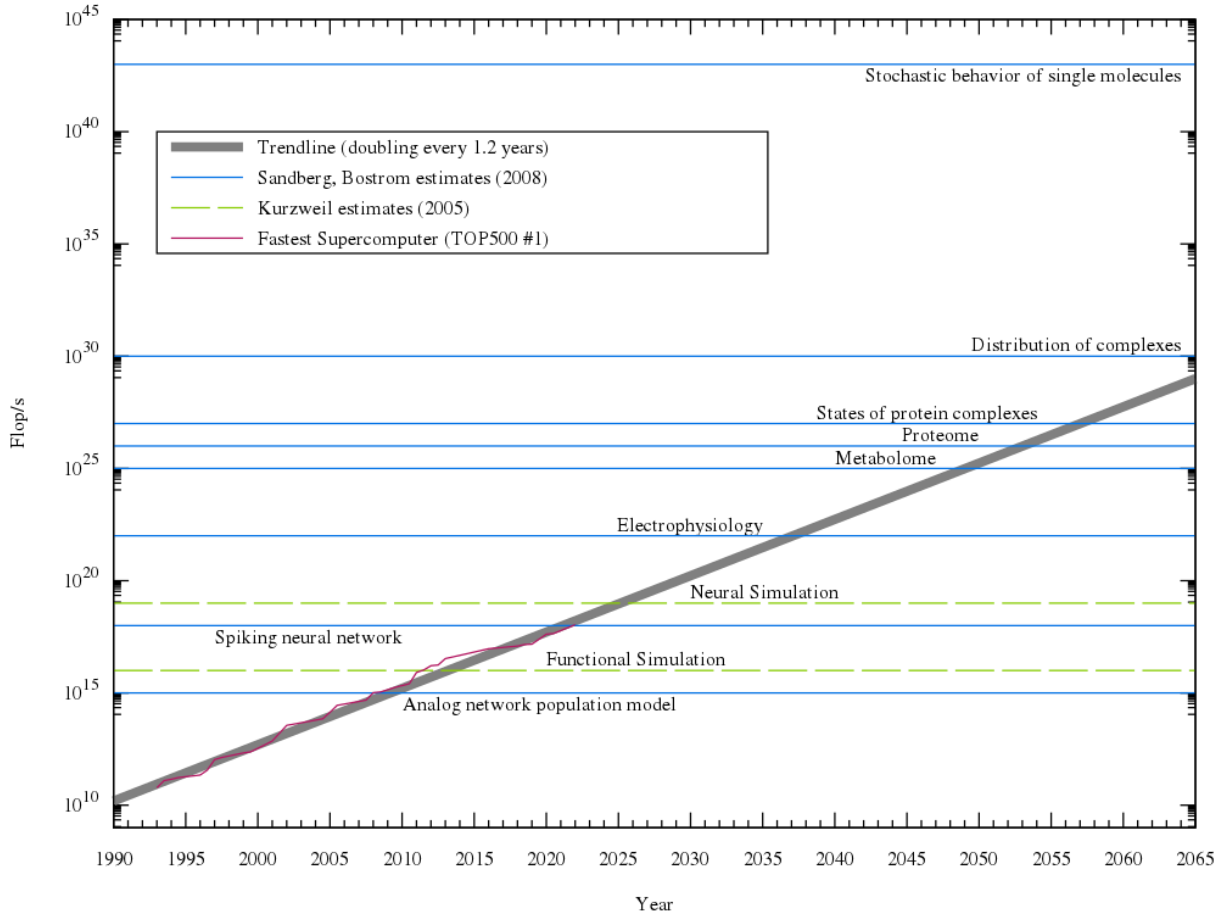


Figura 2-1. Estimación de la potencia de procesamiento para emular un cerebro humano [14]

Esto no es más que una predicción de estos investigadores, con distintos puntos de vista, pero ninguno de ellos descarta que la IAG se convierta en una realidad no demasiado lejana.

Y es que la creación de las IAG centra la atención de las grandes empresas de Inteligencia Artificial de la

actualidad, como OpenAI, DeepMind o Anthropic, a pesar de que siguen existiendo dudas de cuándo se conseguirán desarrollar y del riesgo que pueden suponer para la humanidad.

2.3.2 IA vs Machine Learning vs Deep Learning

A menudo es fácil encontrarse estos términos siendo usados indistintamente, asignándoles un significado que no es el correcto y que puede dar lugar a confusión al lector. Es por eso por lo que en este apartado se van a destacar de forma breve las diferencias entre ellos y en qué consiste cada uno:

En primer lugar, la Inteligencia Artificial. Poco más queda que decir sobre ella, pues se ha dedicado todo el capítulo a su estudio y evolución. Es el campo más general, que abarca un amplio espectro de enfoques, técnicas y herramientas, entre las que se encuentra el Machine Learning y el Deep Learning.

Por otro lado, el Machine Learning es un subconjunto dentro de la IA que se centra en desarrollar algoritmos y modelos que permiten a las máquinas aprender patrones y mejorar sin necesitar ser programadas explícitamente. Esta disciplina supondrá gran parte del trabajo y se dedicará el próximo capítulo de forma íntegra a su estudio [15].

Por último, el Deep Learning es una subcategoría dentro del Machine Learning que se basa en la utilización de redes neuronales artificiales con un gran número de capas, implicando un gran número de datos y una capacidad de procesamiento mayor para resolver problemas complejos como el procesamiento del lenguaje natural (NLP) o el reconocimiento de patrones en imágenes [16].

Por lo tanto, sí, tanto Machine Learning como Deep Learning son subcategorías dentro de la Inteligencia Artificial, pero existen diferencias entre ellos y en los ámbitos en los que se aplica cada uno.

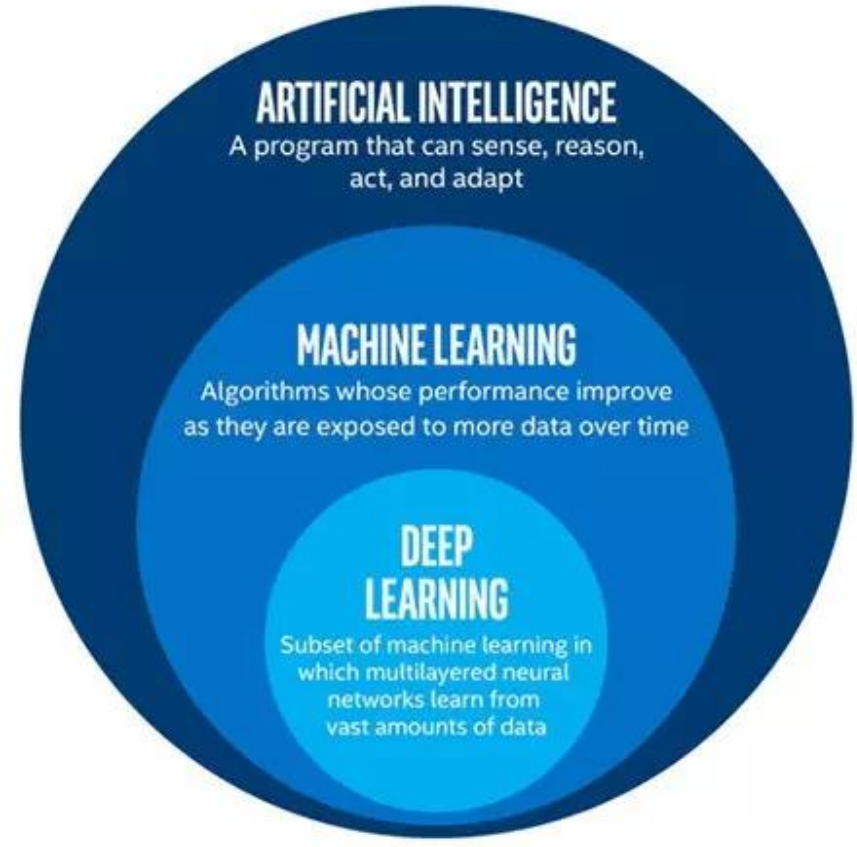


Figura 2-2. Clasificación IA [17]

3 MACHINE LEARNING

Tal y como se ha adelantado en el capítulo anterior, el Machine Learning es uno de los campos más interesantes y profundos dentro de la Inteligencia Artificial. En ese capítulo, se busca recorrer este campo desde sus aspectos más básicos hasta algunos más profundos, con el fin del entender el problema final que se busca solucionar: la detección de eventos en partidos de fútbol.

3.1 ¿Qué es el Machine Learning?

En primer lugar, es importante conocer cuáles son los puntos principales del Machine Learning y qué le hace especial por encima de otros campos de la IA.

El Machine Learning, o Aprendizaje Automático, se centra en el uso de algoritmos y modelos que permiten que las máquinas o computadoras aprendan a partir de su propia experiencia. Es decir, aprenden sin estar explícitamente preparadas para una tarea concreta. En esta tarea, son de vital importancia los datos de entrada, pues son la fuente de información de la computadora que posibilita el aprendizaje. Por ello, se le dedicará un apartado exclusivamente al tratamiento de los datos y la importancia de estos [15].

Por lo general, el proceso de aprendizaje en Machine Learning consta de tres etapas, que se desglosarán con todo lujo de detalles, dada su importancia. Estas etapas son: entrenamiento, validación y prueba.

Todo esto permite que el Machine Learning se aplique en una amplia variedad de campos, pues con la evolución de los algoritmos de aprendizaje en los últimos años, permite automatizar tareas complejas y tediosas para el ser humano.

3.2 Tipos de sistemas Machine Learning

Tras haber hecho un pequeño acercamiento al concepto general de qué es el Machine Learning, es conveniente diferenciar entre los distintos tipos de sistemas en los que se clasifica normalmente en base a distintos criterios.

Comúnmente, los sistemas Machine Learning se clasifican en función de si son sistemas supervisados o no por el humano, de su capacidad de mejorar y aprender una vez está en funcionamiento y de si son capaces de generar modelos predictivos o se dedican únicamente a comparar los nuevos datos de entrada con los que ya son conocidos. Dados estos tres criterios principales, se procede a tratar cada una de estas clasificaciones de forma individual, entrando en un mayor detalle [15].

3.2.1 Aprendizaje supervisado

El aprendizaje supervisado consiste en técnica mediante la cual se entrena un modelo utilizando un conjunto de datos que incluye la salida o la solución deseada. A este conjunto de datos de entrada que indican el valor esperado de la salida se le conoce como etiqueta (label).

Por ejemplo, en la imagen que aparece a continuación, se puede ver un ejemplo que representa el procedimiento que se lleva a cabo con este tipo de aprendizaje. La entrada del modelo que se va a entrenar consiste en un conjunto de figuras geométricas, entre las que aparecen hexágonos, cuadrados y triángulos. En la propia entrada, se añaden etiquetas que le indican al modelo qué tipo de figura geométrica es cada una de ellas. De esta forma, el modelo analiza estas etiquetas y aprende de la entrada. Tras esto, se ha de probar que el modelo ha aprendido,

lo cual se lleva a cabo, en este caso, mostrándole al modelo un cuadrado y un triángulo, esta vez sin etiquetar, para comprobar si es capaz de predecir qué tipo de figura geométrica es [15].

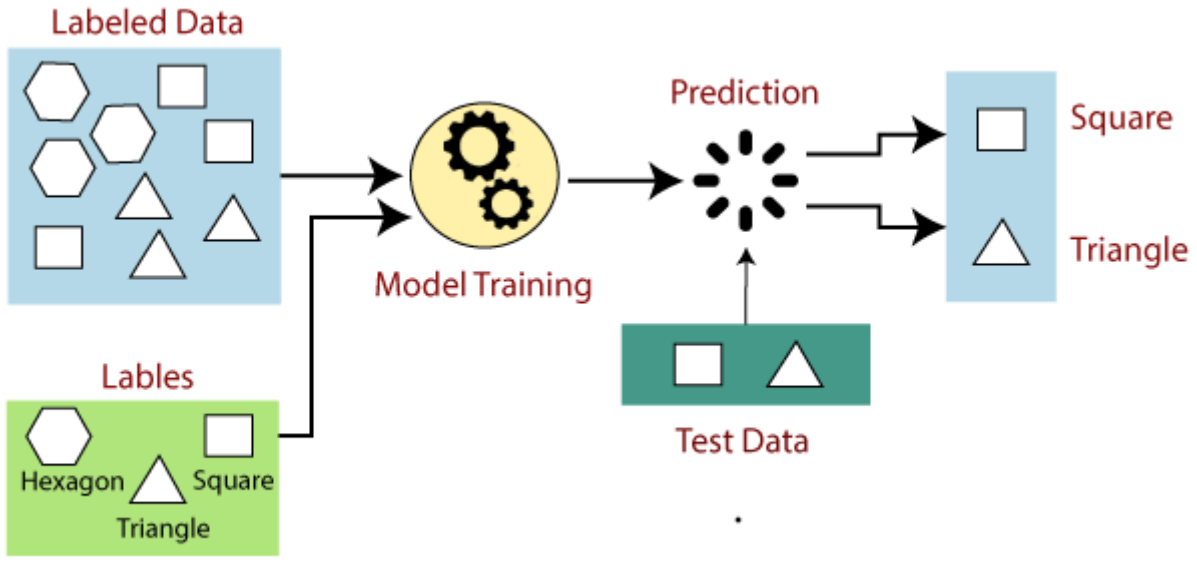


Figura 3-1. Ejemplo de aprendizaje supervisado [18]

Este ejemplo es un caso de uso del aprendizaje supervisado en un problema de clasificación. Los problemas de clasificación son uno de los casos de uso más típicos para el aprendizaje supervisado, pero no el único. También, se utiliza en problemas de regresión, en los que se pide al modelo predecir un valor numérico en base a los datos de entrada que se le proporciona. Estos tipos de problema se abordarán en apartados posteriores, así como algunos de los algoritmos más utilizados para su resolución.

3.2.2 Aprendizaje no supervisado

En contraposición, el aprendizaje no supervisado es una técnica en la que el modelo se entrena utilizando conjuntos de datos que no tienen etiquetas que especifiquen el valor de la salida.

En estos casos, el modelo debe encontrar patrones, estructuras o relaciones en los datos de entrada que le sirvan para conseguir predecir el valor de la salida.

Existen multitud de algoritmos enfocados al aprendizaje no supervisado, como K-Means o DBSCAN, que son algoritmos de clustering. Además de los problemas de clustering, el aprendizaje no supervisado puede utilizarse en problemas de reducción de dimensionalidad o de detección de anomalías, entre otros [15].

En la imagen se puede ver una comparación entre dos posibles resultados dados por un caso de lenguaje supervisado y por otro de lenguaje no supervisado. En el caso del lenguaje supervisado, el problema se ataja como un problema de clasificación, en el que se dan como datos de entrada las etiquetas con el valor de salida esperado en función del valor de entrada, tal y como se ha explicado en el anterior subapartado. En el caso del lenguaje no supervisado, la tarea se encara como un problema de clustering, en el que no se introducen etiquetas, sino que el algoritmo que se utiliza busca patrones o similitudes entre los datos de entrada para agruparlos en clusters.

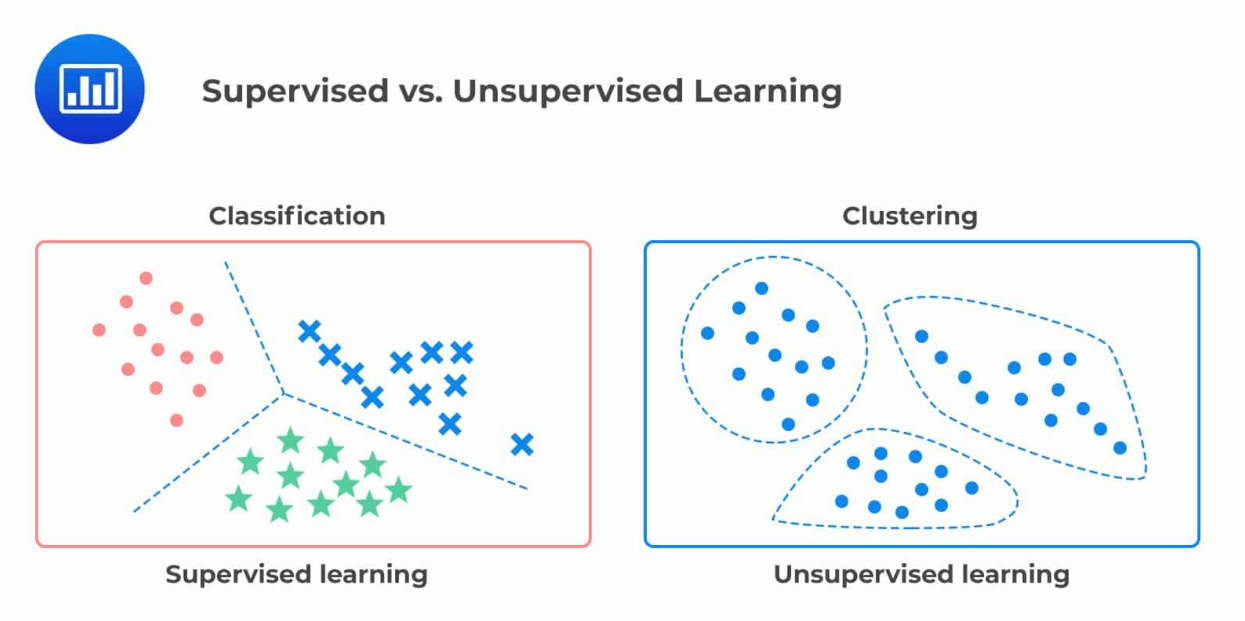


Figura 3-2. Aprendizaje supervisado vs Aprendizaje supervisado [19]

3.2.3 Aprendizaje semisupervisado

Algunos algoritmos pueden combinar los métodos del aprendizaje supervisado y del no supervisado. En lugar de que los datos de entrada sean únicamente etiquetados o no etiquetados, se puede hacer una combinación de ambos. Normalmente, la proporción de datos sin etiquetar es mayor que la de los etiquetados.

Un ejemplo de esto es Google Photos, que reconoce automáticamente a personas que aparecen en distintas fotos con un algoritmo de clustering y que si se le añade una etiqueta indicando quien es esa persona, es capaz de nombrar a esa persona en todas las fotos en las que aparece. [15]

3.2.4 Aprendizaje por refuerzo

El aprendizaje por refuerzo es un paradigma distinto a los anteriores. En este tipo de aprendizaje se puede identificar la figura del agente, que representa al sistema de aprendizaje, y que aprende a raíz de su observación del entorno y realizando acciones, las cuales serán recompensadas o castigadas en función de si se acerca al objetivo final o no. Es decir, el sistema aprende por sí mismo, de sus propias acciones, buscando siempre la mayor recompensa posible.

Este tipo de aprendizaje es utilizado, por ejemplo, para algoritmos que se utilizan en conducción autónoma o en robots para que aprendan a andar [20].

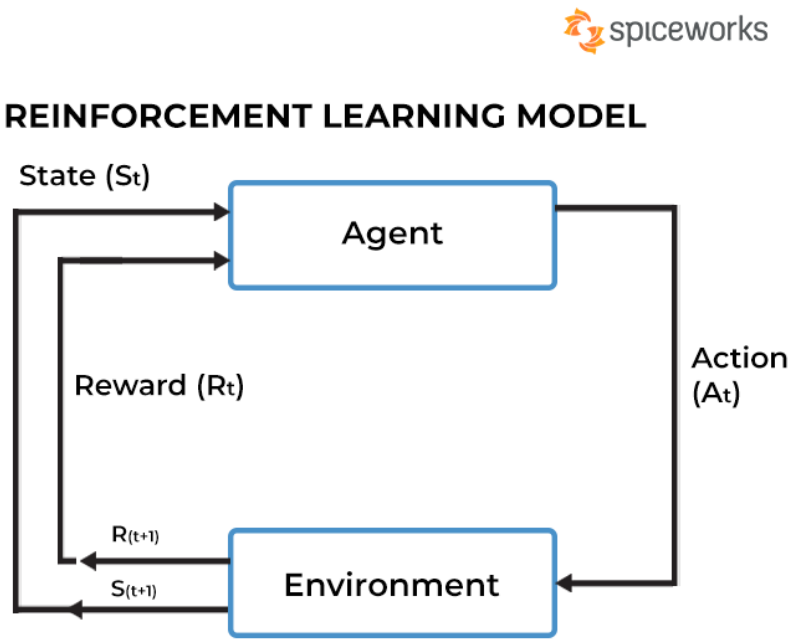


Figura 3-3. Esquema de funcionamiento del aprendizaje por refuerzo [20]

3.2.5 Aprendizaje Batch

El aprendizaje batch o aprendizaje por lotes entra dentro de una clasificación distinta que los anteriores. En este caso, se clasifica en función de si puede o no aprender en tiempo real.

El aprendizaje batch consiste en entrenar el modelo con todos los datos disponibles, agrupándolos en lotes o batches. Por lo tanto, no es capaz de aprender de forma incremental. Esto hace que el modelo se lance sin que pueda aprender más, únicamente aplicando lo que ya sabe.

El entrenamiento con esta enorme cantidad de datos puede llevar bastante tiempo de computación y requerir de una cantidad significativa de memoria, por lo que suele hacerse de forma offline [21].

3.2.6 Aprendizaje online

Por su parte, el aprendizaje online consiste en entrenar el modelo de forma continua y progresiva, añadiendo nuevos datos a medida que están disponibles.

Este tipo de aprendizaje es útil para sistemas en los que los datos se reciben de forma continua, haciendo el modelo más adaptable a los cambios y dándole la capacidad de cambiar con facilidad.

También tiene se usa en los casos en los que el conjunto de datos es muy grande y no se dispone de memoria suficiente para poder lidiar con él [21].

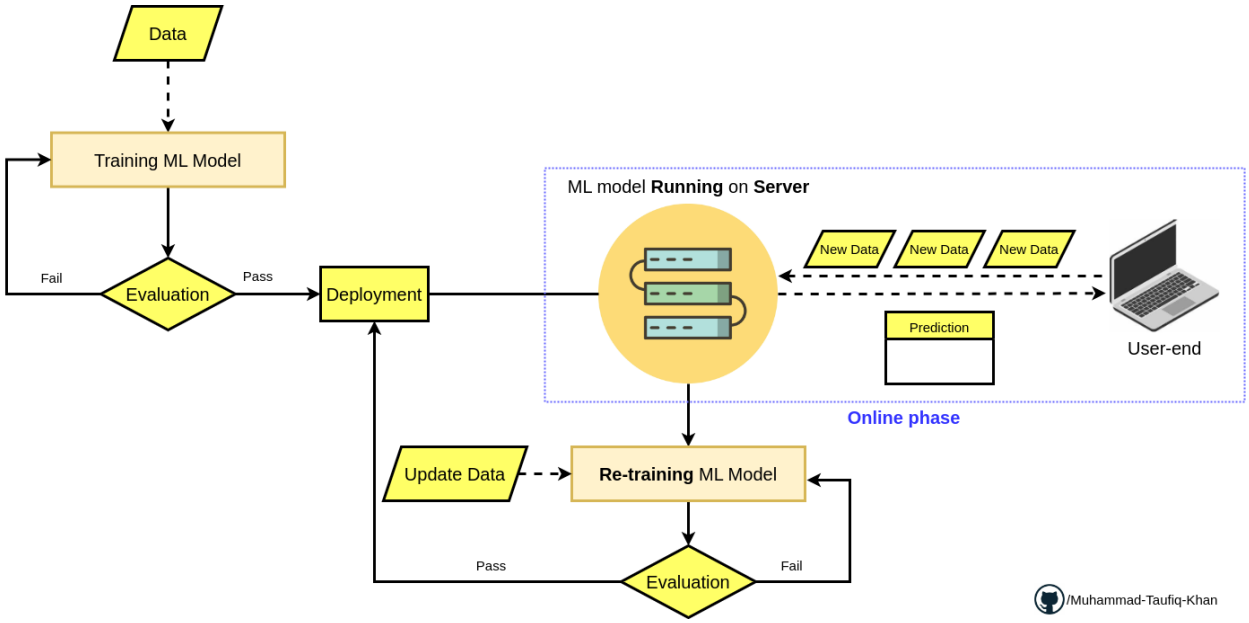


Figura 3-4. Aprendizaje por lotes VS Aprendizaje online [21]

3.3 Datos

Uno de los puntos fundamentales, si no el que más, para que el modelo funcione acorde a lo que se busca son los datos de entrada. Son la materia prima con la que se alimenta el modelo, utilizándose tanto para su entrenamiento como para su posterior evaluación.

Dada su importancia, se dedica este apartado a analizar algunas de las características que hacen que los datos de entrada sean correctos para el modelo a utilizar y otras que ayudan a detectar problemas y futuros inconvenientes, así como su clasificación dependiendo de su tipo.

3.3.1 Tipos de datos

A pesar de que los tipos de datos pueden ser creados según la necesidad del programador y, por lo tanto, ser ilimitados, existen cuatro tipos de datos predominantes y bajo los cuales pueden agruparse la gran mayoría de ellos:

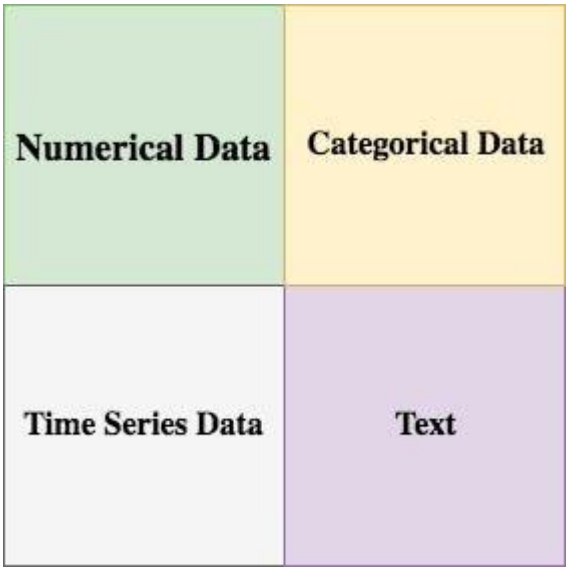


Figura 3-5. Tipos principales de datos en Machine Learning [22]

- Datos numéricos: Son aquellos datos que son números exactos. Estos datos tienen sentido de medida, como, por ejemplo, el número de eventos que tienen lugar en un vídeo de un partido de fútbol. A su vez, los datos numéricos pueden ser clasificados en datos continuos o discretos [22].

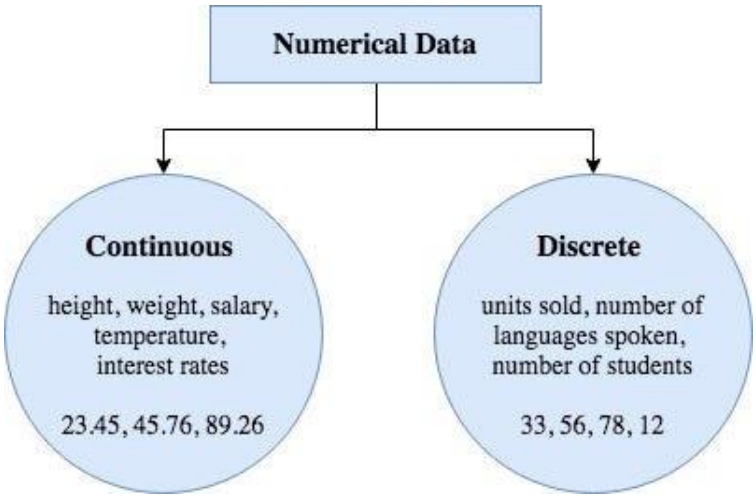


Figura 3-6. Datos numéricos. Continuos y discretos. [22]

- Datos categóricos: Este tipo de dato representa características, como, por ejemplo, la posición de un jugador de fútbol, su equipo o su lugar de nacimiento. Este tipo de datos también puede tomar valores numéricos, pero este número no tiene un sentido matemático, sino identificativo, como índice de una lista de valores, por lo que no tendría ningún sentido hacer una media aritmética, sumar o restar dichos valores numéricos [22].
- Series temporales: Son secuencias de valores numéricos que, a diferencia de los datos numéricos, tienen un valor temporal asociado a ellas. Es un tipo de dato que se utiliza de forma recurrente en finanzas. Por ejemplo, se puede medir el número de ventas mensuales de viviendas durante el paso de los años, como en la imagen que se puede ver a continuación.

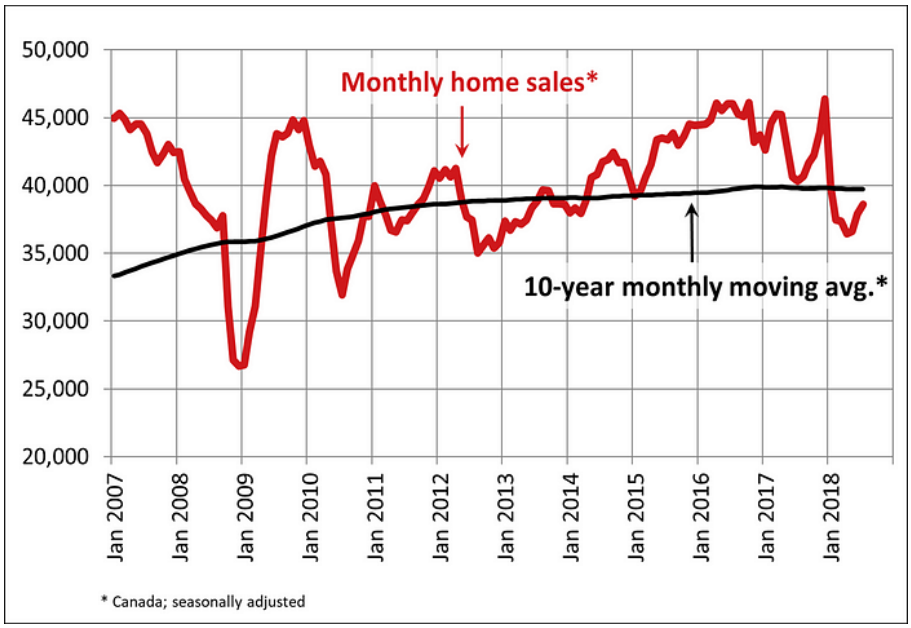


Figura 3-7. Ejemplo de uso de series temporales [22]

- Texto: Los datos de tipo texto son, básicamente, palabras. Todas las letras tienen un número asociado, ya sea haciendo uso del código ASCII o cualquier otra codificación. Estos datos son de vital importancia en procesamiento de texto y en tareas que están a la orden del día en el mundo de la Inteligencia Artificial como el procesamiento del lenguaje natural (Natural Language Processing). [22]

3.3.2 Datos de imágenes, audio y vídeo

Además de los cuatro tipos de datos desarrollados en el apartado anterior, es importante destacar de qué forma se representan los datos cuando se tratan de imágenes, audio y vídeo, es decir, datos multimedia. En este trabajo será de especial interés el tratamiento de los datos de vídeo y, por consiguiente, también el de imágenes.

En primer lugar, el audio puede clasificarse como una serie temporal, formada por una secuencia de valores numéricos que tienen un instante temporal asociado. Cada muestra de audio captura la amplitud de la señal de audio en un instante temporal, uniéndose todas las muestras para conformar la forma de onda que representa la señal de audio a lo largo del tiempo. Es de especial utilidad en reconocimiento de voz o detección de sonidos. [23]

Por otro lado, las imágenes son representadas por matrices de píxeles, que pueden ser bidimensionales o tensores, dependiendo de si son imágenes en color o no.

En el caso de las imágenes que no son en color, es decir, en escala de grises, cada píxel se representa mediante un valor numérico que indica la intensidad del gris en ese punto. Este valor suele estar comprendido entre 0 y 255, siendo 0 el color negro y 255 el color blanco. [23]

Por otro lado, para las imágenes a color, se utilizan tres canales de color, uno para el color rojo (R), otro para el color verde (G) y otro para el color azul (B). Cada píxel se representa como una combinación de estos tres valores. En este caso, no se tiene una matriz bidimensional, sino un array tridimensional, en el que cada dimensión representa un canal de color (RGB).

Estas representaciones permiten el desarrollo de tareas como el reconocimiento de objetos, clasificación de imágenes o segmentación de imágenes.

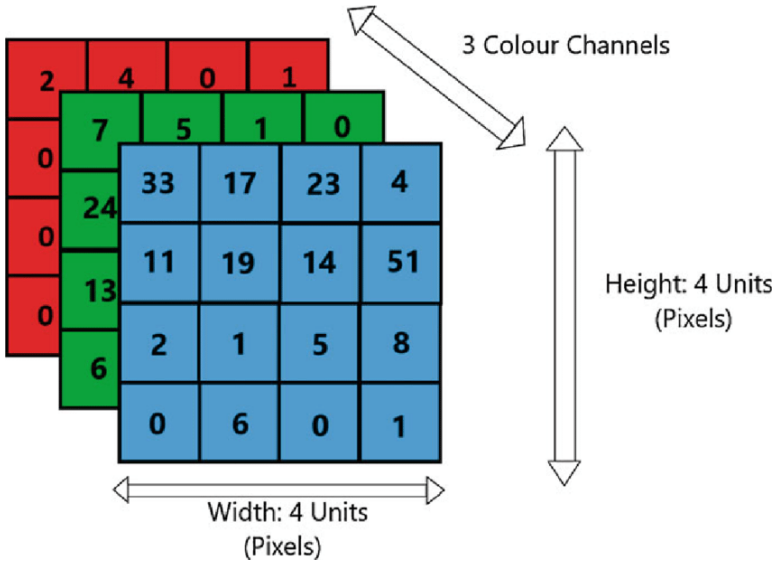


Figura 3-8. Ejemplo de tensor tridimensional [23]

Por último, los vídeos se representan como una secuencia temporal de imágenes, donde cada frame del vídeo es una imagen estática que muestra un instante específico de tiempo. Cada una de las imágenes se representará como se ha explicado justo antes, con una matriz bidimensional o un tensor.

3.3.3 Preparación y obtención de los datos

El proceso de obtención y preparación de los datos es un proceso arduo, a la vez que primordial. Y es que no siempre los datos proceden de una gran base de datos en la que el formato está adecuado a lo que se introducirá en el modelo, sino que es posible que sea necesario etiquetar manualmente los datos de entrada, proceso que implica un gran trabajo.

El proceso de etiquetado (labelling) consiste en asignar una etiqueta (label) a un conjunto de datos. Las etiquetas se utilizan para clasificar o categorizar los datos con el fin de adecuarlos para entrenar a los modelos. En ocasiones, será necesario contar con la ayuda de profesionales de otros sectores que ayuden a clasificar los datos y a etiquetarlos correctamente. Esto hace que el tiempo de este proceso pueda elevarse, al involucrar la labor de varias personas, y que implique más del 60 o 70 porciento del tiempo total del proyecto [24].

Pero no solo el etiquetado forma parte de este proceso de preparación de datos, también conocido como data curation, sino que implica una serie de tareas entre las que se encuentran, por ejemplo [15]:

- Obtención de los datos: Implica la identificación de los datos y la recopilación de estos, provenientes de diversas fuentes como bases de datos, conjuntos de datos públicos o medidas tomadas mediante sensores.
- Limpieza de los datos: Consiste en identificar y corregir errores, posibles inconsistencias y valores incorrectos de los datos. Es común encontrar datos duplicados, algunos valores que faltan o medidas que no se han tomado correctamente.
- Añadir metadatos: Consiste en añadir información adicional a los datos para mejorar su comprensión y futuras investigaciones. Pueden añadirse descripciones, etiquetas o citar fuentes.
- Validar los datos: Hacer una comprobación del conjunto de datos con otro profesional que pueda identificar posibles inconsistencias no detectadas en fases previas.

Además de estas mencionadas, existen otras tareas que pueden depender del proyecto que se esté abordando.

Esto hace ver que, como se ha mencionado anteriormente, esta tarea puede ser tediosa y no se debe tratar como algo menor, sino lo contrario. Sin un buen conjunto de datos, no se tendrá materia prima para alimentar el algoritmo que utilice el modelo, por lo que el resultado no será satisfactorio.

3.3.4 Problemas asociados a los datos

En relación con la importancia de los datos, pueden existir distintos problemas en ellos que hagan que el modelo no funcione como se espera. Por ello, es de vital importancia tratar de forma muy seria la fase de obtención y preparación de los datos, como se ha mencionado en el apartado anterior. Este apartado se centra en definir algunos de los problemas más comunes que hacen que un conjunto de datos sea inadecuado para su uso o que pueda provocar algún problema futuro en el modelo.

3.3.4.1 Ausencia de datos

Es uno de los problemas más comunes que pueden surgir en un conjunto de datos. Hace referencia a que los datos son de poca calidad o incluso inexistentes para algunos de las instancias o características que se están evaluando.

Esta situación puede deberse a múltiples factores como, por ejemplo, la toma de medidas inexactas, errores de muestreo, respuestas omitidas o pérdidas de datos. Es de vital importancia abordar este problema y ajustar las entradas en las que se tenga este problema, adaptándolas al formato que exige el modelo [15].

Para ello, existen distintas soluciones, siendo la siguientes las más comunes:

- Eliminar la instancia que contenga el error o la falta de datos.
- Sustituir las instancias erróneas con la media del resto de entradas o el valor más repetido, la moda.
- Eliminar los atributos en los que se encuentra el problema.

3.3.4.2 Ruido y errores en los datos

Otro de los problemas más comunes es encontrar valores atípicos en un conjunto de datos, que se desvían de forma significativa del resto de los datos. La aparición de estos valores puede ser resultado de errores en la toma de medidas, de un mal etiquetado o de algún fenómeno inusual que no se ha tenido en cuenta a la hora de realizar el análisis, entre otras cosas. Estos valores atípicos son comúnmente conocidos como "outliers" y tienen un gran impacto en el análisis de los datos ya que pueden distorsionar la salida del modelo, sesgando los resultados o provocando algo inesperado [25].

Por lo tanto, es importante prestar atención a estos outliers y detectarlos lo antes posible para poder eliminarlos del conjunto de datos, pues la presencia de una única muestra errónea en el conjunto de datos puede llevar a que el modelo no aprenda de forma correcta. Normalmente, cuando se detecta un outlier se procede a eliminar esa muestra del conjunto de datos, aunque si son varias las muestras que sufren este problema, se debe analizar si el problema va más allá de un simple error de medida. Otra posible solución es trabajar en la robustez del modelo, haciendo que este sea menos sensible a los outliers, usando, por ejemplo, estimadores o algoritmos especializados en tratar con este problema.

También puede darse el caso de que exista ruido en los datos. Este ruido puede aparecer debido a múltiples factores, como interferencias en el punto que se realiza la medida o algún defecto en el objeto de medición. Si los niveles de ruido son muy elevados, los datos no serán de utilidad, algo que será fatal para el modelo. Como resulta obvio, un dato con un nivel de ruido muy alto puede ser considerado como un outlier [25].

3.3.4.3 Cantidad insuficiente de datos

Para que un algoritmo de Machine Learning aprenda algo tan simple como diferenciar un gato de un perro o un plátano de una manzana se necesitan miles de ejemplos, es decir, una gran cantidad de datos. Por lo tanto, en problemas mucho más complejos, como reconocimiento del habla o trabajar con imágenes, los algoritmos necesitan millones de datos.

En un paper publicado en julio de 2001 por Michele Banko y Eric Brill, trabajadores de Microsoft, demostraron que distintos algoritmos, algunos de ellos muy simples, eran capaces de llegar a una solución muy similar cuando eran entrenados con suficientes datos de entrada. La conclusión de los autores fue que, vista la importancia de tener un conjunto de datos, se debería replantear el hecho de invertir la mayoría del tiempo y del presupuesto en desarrollar algoritmos más complejos y empezar a invertir en tener mejores datos de entrada. Aun hoy, en muchas ocasiones no es fácil ni barato contar con un buen conjunto de datos para entrenar el modelo. [26]

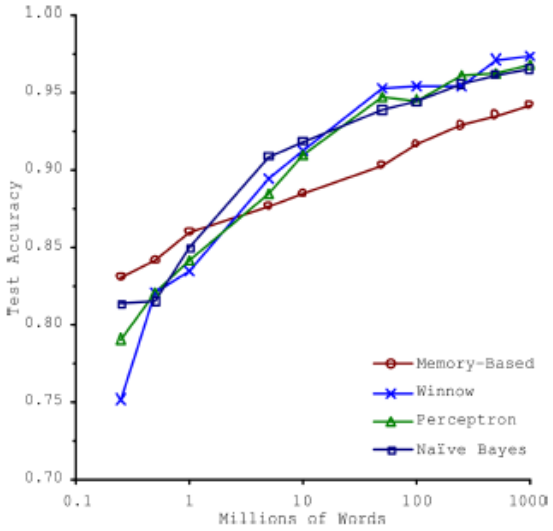


Figura 3-9. Respuesta de distintos algoritmos ante el mismo problema [26]

3.3.5 Datos no representativos

Este problema ya se mencionó en el primer capítulo de este trabajo, y es que para que el modelo pueda estar correctamente generalizado, es esencial que los datos con los que se entrena representen adecuadamente los casos que se quieren tratar. Este es el caso de la Inteligencia Artificial que se entrenó únicamente con datos escritos por hombres y que, por consiguiente, no contrataba mujeres [15].

Este tipo de problema en el que los datos están sesgados es muy común en sondeos políticos. Se han dado numerosos casos en los que las encuestas apuntaban a que un candidato sería elegido para finalmente ocurrir justo lo contrario. Uno de los casos más famosos tuvo lugar en las elecciones presidenciales de Estados Unidos en el año 1936. El periódico Literary Digest, uno de los más famosos en aquella época y que años anteriores había predicho con gran exactitud los resultados de las elecciones, dio como ganador a Landon con un 57% de los votos. Llevaron a cabo la mayor encuesta en ese año, tomando 2.4 millones de muestras. [27]

Sin embargo, los resultados no fueron los esperados, obteniendo Landon solo el 38% de los votos, por un 62% de su rival, Roosevelt. Tras esto, se investigó cuál fue el error a la hora de realizar esa encuesta, llegando a las siguientes conclusiones:

- Las direcciones que el periódico utilizó para enviar las encuestas se obtuvieron de los suscriptores de este y de páginas telefónicas. Se pudo comprobar que estas listas estaban mayormente asociadas a personas de clase media y clase alta, que estadísticamente tendían a votar al partido republicano, liderado por Landon.
- De los 10 millones de encuestas enviadas, solo 2.4 millones fueron respondidas. Las personas que respondieron a la encuesta, muy probablemente, tenían un interés en política, mientras que los que no respondieron, no. Esto dejaba fuera de la encuesta una importante parte de la población estadounidense, los ciudadanos no interesados en política.

Este ejemplo ilustra de forma excelente como el conjunto de datos marca el resultado de forma directa. Si el conjunto de datos que se utiliza es poco representativo, es muy poco probable que el modelo consiga hacer buenas predicciones.

Conseguir un conjunto de datos correcto puede ser complicado, puesto que si se toman pocas muestras es posible que haya un problema de ruido, como se ha visto en el apartado anterior, pero incluso teniendo grandes conjuntos de datos, si el método de obtención de estos no es el correcto, exista un problema de sesgo, como se acaba de explicar en el ejemplo.

3.4 Modelo y algoritmos

Junto a los datos, el modelo y la función de coste conforman las partes más importantes del Machine Learning. El algoritmo es alimentado con el conjunto de datos de entrada y ajusta el modelo para reducir la función de coste al máximo posible.

La función de coste o función de pérdida es una medida que se utiliza para evaluar el funcionamiento del modelo en la tarea que se le ha asignado. Esta función calcula la diferencia entre el valor esperado y el valor predicho por el modelo, representándolo como un número real. Es una medida de como de "mala" es la estimación del modelo en relación con el que es el resultado esperado. Por lo tanto, el objetivo del modelo ha de ser buscar el conjunto de parámetros que minimicen la función de coste. Más adelante en este apartado, se tratarán algunas de las funciones de coste más comunes y sus principales características.

Volviendo al proceso mediante el cual se entrena al modelo, se pueden destacar tres etapas principales [15]:

- En primer lugar, el entrenamiento. Esta fase consiste en utilizar el conjunto de datos de entrada para alimentar el modelo, de forma que se van ajustando los parámetros para minimizar la función de coste. Se suele utilizar de un 70% a 90% del total del conjunto de datos para esta fase.

- De forma paralela al entrenamiento se realiza la validación. Es un proceso continuo que ocurre durante el entrenamiento del modelo para evaluar su rendimiento en datos que no forman parte del conjunto de entrenamiento. Para este proceso también se utiliza un conjunto de datos independiente, que suele ser de menor tamaño que el de pruebas. Se utiliza de un 10% a un 20% del conjunto de datos para esta fase, aunque podría omitirse si el modelo utilizado es lo suficientemente robusto.
- A continuación, se pasa a la fase de pruebas o testing. En esta etapa se utiliza una parte del conjunto de datos independiente de los datos utilizados para el entrenamiento, con el fin de evaluar el rendimiento del modelo tras haber sido entrenado. Es muy importante que este conjunto de datos no se utilice durante el entrenamiento, para que este proporcione una evaluación no condicionada del modelo, a partir de nuevos datos de entrada. Se utiliza de un 10% a un 20% del conjunto de datos.

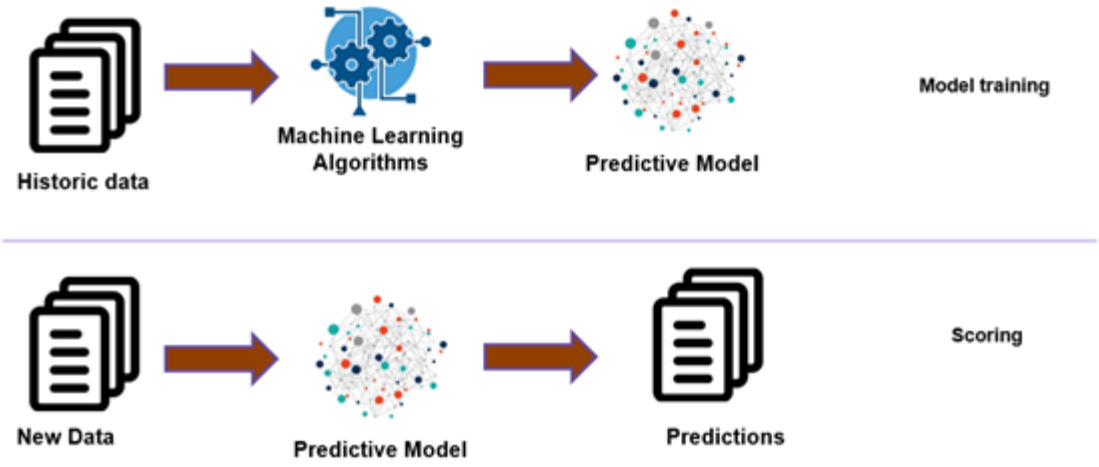


Figura 3-10. Fase de entrenamiento y test de un modelo de Machine Learning [28]

Pasando por estas etapas, se puede ajustar el modelo, modificando algunos parámetros, el algoritmo que se está utilizando o haciendo modificaciones en el conjunto de datos para hacer que el modelo se comporte acorde al objetivo.

Esto lleva a definir qué es exactamente un modelo. Un modelo es una herramienta matemática que utiliza una serie de parámetros e hiperparámetros para traducir una entrada es una salida, la cual predice a partir del ajuste de estos parámetros.

Cabe resaltar la diferencia entre los parámetros y los hiperparámetros del modelo. Los parámetros son las variables internas del modelo que el algoritmo ajusta durante el proceso de entrenamiento para hacer que el modelo se adapte de la mejor forma a estos, tratando de minimizar la función de coste. Por otro lado, los hiperparámetros se utilizan para controlar el entrenamiento y son variables que definen el modelo. Estos hiperparámetros han de ajustarse antes del entrenamiento, a diferencia de los parámetros que se van ajustando durante el entrenamiento. Algunos de los hiperparámetros más comunes son la tasa de aprendizaje o learning rate, el número de épocas o el tamaño del lote, que cobrarán más importancia en la parte final del trabajo [29].

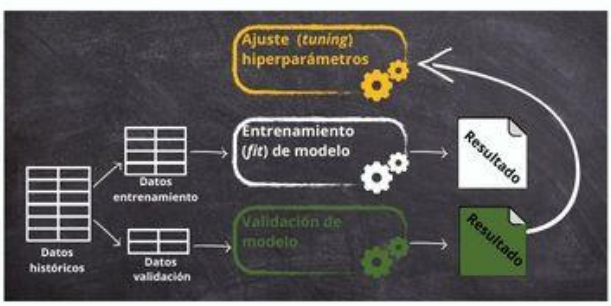


Figura 3-11. Ubicación de los hiperparámetros en el modelo [29]

3.4.1 Generalización y overfitting

Una vez el modelo ha sido entrenado, es el momento de comprobar si es capaz de predecir las salidas a partir de las entradas acorde a lo que se espera de él. Para poder decir que un modelo cumple con la **generalización**, el modelo debe ser capaz de predecir de forma correcta los valores de salida ante nuevos valores de entrada, es decir, ante datos nunca vistos por el modelo. De esta forma, se comprueba que el modelo ha sido capaz de aprender y capturar los patrones subyacentes en los datos de entrenamiento y aplicarlos a los nuevos datos, y no se ha limitado a "memorizar" las respuestas a los datos de entrada, cosa que provocaría que, al llegar un nuevo dato, el modelo no supiera responder a él.

La existencia de generalización o no en el modelo puede darse debido a varios factores [15]:

- Complejidad del modelo y número de parámetros: Es lógico pensar que, si un modelo es demasiado simple, es posible que no sea capaz de capturar los patrones de los datos, si es que estos son muy complejos, pero también puede darse el caso de que el modelo sea demasiado complejo, lo que provoca que el modelo se sobreajuste a los datos de entrenamiento y no sea capaz de generalizar.
- Calidad del conjunto de datos de entrenamiento: Como ya se ha explicado largo y tendido en apartados anteriores, un conjunto de datos de entrenamiento grande, representativo y de calidad proporciona al modelo una información más limpia para aprender las relaciones en los datos.
- Regularización: Existen técnicas para añadir un factor de penalización en la función de coste con el objetivo de que algunos de los parámetros del modelo no sean demasiado altos y prevenir así los problemas de generalización.

Existe un concepto muy ligado a la generalización, que ya ha sido mencionado unas líneas atrás, que es el sobreajuste u overfitting en inglés. El sobreajuste se da cuando el modelo está demasiado ajustado a los datos de entrenamiento. Esto provocará que el modelo responda muy bien cuando los datos de entrada sean conocidos, pero es posible que falle ante datos que le planteen una nueva situación, debido a su pobre generalización.

Al igual que existe el sobreajuste, también existe el efecto contrario, el subajuste o underfitting, que ocurre cuando el modelo incapaz de capturar adecuadamente los patrones en los datos de entrenamiento, normalmente por ser demasiado simple.

La figura 3-12 representa de forma muy gráfica estos dos conceptos. En los primeros ejes, se tiene un modelo que trata de aproximar esos valores de entrada con una línea recta y que, al ser demasiado simple, no es capaz de ajustarse lo suficiente a estos valores. Por el contrario, en los ejes que se encuentran más a la derecha, se produce un caso de sobreajuste en el que el modelo se ajusta demasiado a los datos de entrada. Lo óptimo sería que ocurriera lo que ocurre en los ejes centrales, en los que el algoritmo detecta ciertos patrones en los datos de entrada pero no se ajusta en exceso, siendo capaz de generalizar y responder ante nuevos valores [30].

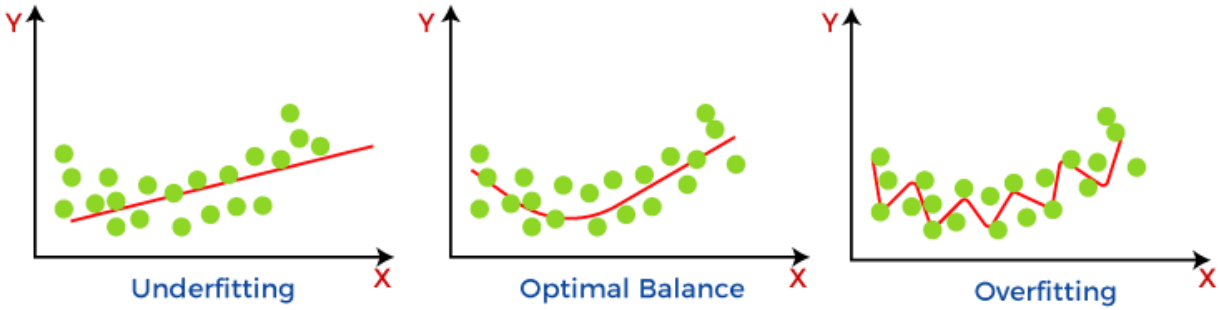


Figura 3-12. Subajuste y sobreajuste [30]

Dos últimos conceptos relacionados con esto son el sesgo y la varianza. Cuando el modelo no es capaz de identificar los patrones de los datos de entrenamiento, como en el caso de la línea recta en la figura anterior, se dice que tiene un problema de sesgo. Por el contrario, cuando el modelo está sobreajustado a los datos de entrada, se dice que tiene un problema de varianza.

En la siguiente figura se puede ver una gráfica que relaciona ambos conceptos con la complejidad del modelo y el error. Es una imagen conocida como "trade-off" sesgo-varianza y es ampliamente conocida en el mundo del Machine Learning. En la imagen se puede comprobar que, cuando el sesgo (en color rojo) es elevado, se ubica en la zona de subajuste y cuando la varianza es elevada (en color verde) en la zona de sobreajuste. Además, es importante la relación con el error y la complejidad del modelo: a medida que el modelo se va haciendo más complejo, disminuye la probabilidad de error, pero dando, como se ha explicado anteriormente, un problema de sobreajuste. Es por eso que, en ocasiones, es difícil concretar los parámetros del modelo para conseguir que funcione de una forma correcta [31].

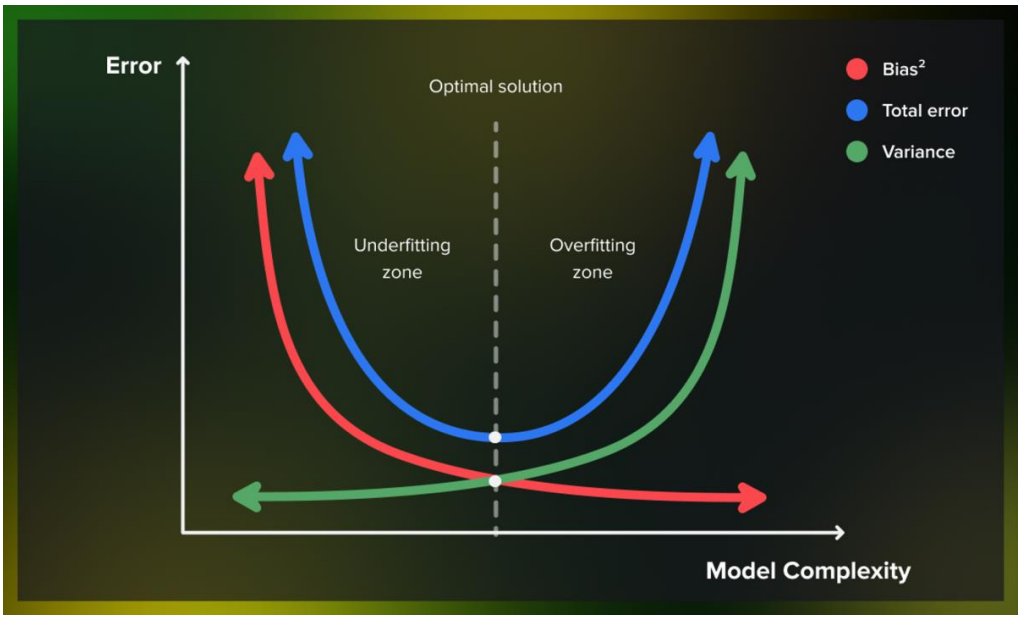


Figura 3-13. Bias-Variance tradeoff [31]

3.5 Clasificación de problemas y algoritmos utilizados

El último apartado de este capítulo se dedicará a realizar un recorrido sobre los tipos de problema más recurrentes en Machine Learning, siendo estos los problemas de clustering, de regresión y de clasificación, explicando sus puntos principales y los algoritmos más utilizados para su solución.

3.5.1 Clustering

El clustering o análisis de grupos es una técnica de Machine Learning que se utiliza para agrupar conjuntos de datos en grupos o clusters, con el fin de agrupar elementos que tienen aspectos en común dentro de un mismo grupo. El objetivo principal de esta agrupación es identificar estructuras en los datos. El clustering se trata de un problema de aprendizaje no supervisado, lo que significa que no se dispone de etiquetas ni de información sobre el origen de las muestras. El clustering es una técnica utilizada en gran variedad de campos como por ejemplo: el análisis de mercado, el procesamiento de imágenes, el análisis de estadísticas en redes sociales o la minería de datos, debido a su capacidad de identificación de patrones.

Sin más dilación, se procede a desarrollar dos de los algoritmos más utilizados para clustering:

3.5.1.1 K-Means

El algoritmo K-Means es uno de los algoritmos más simples y rápidos que se utilizan para clustering [32]. Se divide en los siguientes pasos:

- En primer lugar, se seleccionan 'k' centroides de forma aleatoria, siendo 'k' el número de grupos o clústeres predefinido. Estos centroides se eligen aleatoriamente del conjunto de datos o puede seguirse algún método para elegirlos. Los centroides serán los puntos representativos de cada clúster.
- A continuación, se toma cada punto dentro del conjunto de datos y se asigna al clúster cuyo centroide es el más cercano, midiendo la distancia euclídea.
- Una vez todos los puntos han sido asignados al clúster más cercano, se han de recalcular los centroides haciendo el promedio de todos los puntos asignados a un mismo clúster.
- Los pasos 2 y 3 se repiten de forma iterativa hasta que se cumpla con un criterio de convergencia, que normalmente suele ser que los centroides no cambien de posición entre una iteración y la siguiente o que se alcance un número máximo definido de iteraciones.

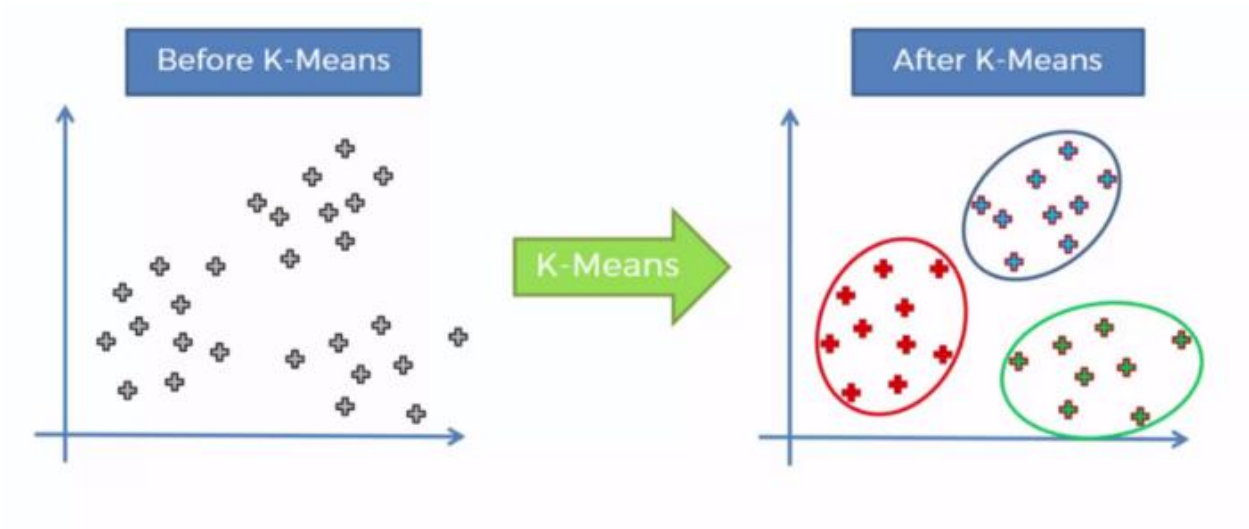


Figura 3-14. Ejemplo de K-Means sencillo. [33]

Para medir cuán buena es la solución, es decir, medir el éxito del modelo en su tarea de agrupar los datos del conjunto de datos utilizando el algoritmo K-Means, se utiliza la **inercia**. La inercia mide la dispersión de los puntos de datos dentro de cada clúster o lo que es lo mismo, cómo de cerca están los puntos de sus respectivos centroides.

Para calcular la inercia, se realiza la suma de las distancias entre cada punto y el centroide de su clúster al cuadrado, para todos los puntos y todos los clústeres [32]:

$$Inercia = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

Donde:

- k es el número de clústeres.
- Ci es el conjunto de puntos de datos asignados al clúster i.
- μ_i es el centroide del clúster i.
- $\|x - \mu_i\|^2$ es la distancia al cuadrado entre un punto de datos x y el centroide μ_i de su clúster asignado.

Cuanto más baja sea la inercia, los clústeres estarán mejor definidos, pues indicará que los puntos de datos son cercanos a sus respectivos centroides. Por lo tanto, la inercia funciona como función de coste en el algoritmo K-Means, pues se tratará de minimizarla.

Puesto que las inicializaciones de los centroides son comúnmente aleatorias, se producirán diferentes inicializaciones en cada uso del algoritmo. Esto lleva a que el resultado final del algoritmo, es decir, el valor de la inercia, puede variar según la inicialización. Por ello, se recomienda probar distintas inicializaciones y elegir la que dé como resultado un valor más pequeño de inercia.

Por último, también hay que destacar algunos puntos negativos de este algoritmo, que dificultan su uso en determinadas situaciones [32]:

- El resultado final puede depender en gran medida de la inicialización de los centroides. Una mala inicialización puede llevar a un mínimo local en vez de a un mínimo global. Por lo tanto, si se quiere intentar obtener una solución distinta, será necesario ejecutar el algoritmo varias veces, lo cual también es un aspecto negativo.
- Se necesita especificar el número de clústeres, que puede ser desconocido o difícil de conocer a priori en algunas situaciones.
- Es un algoritmo sensible a los outliers o datos atípicos, que pueden llegar a tener un gran impacto en la posición de los centroides, afectando negativamente en los clústeres que se forman a partir de ellos.
- Es sensible a los conjuntos de datos con formas no esféricas y a conjuntos de datos con variedad de densidad o tamaño.
- No es adecuado para datos con varias dimensiones, pues la distancia euclídea puede convertirse en un dato menos significativo.

3.5.1.2 DBSCAN

DBSCAN (Density-based spatial clustering of applications with noise) es otro de los algoritmos de clustering más utilizados y que suple alguno de los problemas que tiene K-Means. A diferencia de K-Means, DBSCAN no necesita que se especifique el número de clústeres a priori, lo que hace que sea especialmente útil en casos en los que el número de clústeres es difícil de estimar. [34]

Los pasos que sigue este algoritmo son los siguientes:

- Se definen dos parámetros fundamentales: épsilon y min_samples. El primero, épsilon, especifica la distancia máxima entre dos puntos para que sean considerados vecinos el uno del otro. Mediante este parámetro se podrá regular la densidad del clúster, puesto que cuanto mayor sea el valor de épsilon, puntos más lejanos podrán ser vecinos, lo que creará clústeres más grandes y un menor número de clústeres.
  Por otro lado, el segundo parámetro, min_points, especifica el número mínimo de puntos que deben de estar dentro de un radio de valor épsilon para que un punto sea considerado como "punto central".
- El algoritmo selecciona cada uno de los puntos y se examina el "vecindario" de radio épsilon de estos puntos. Si el número de puntos dentro del vecindario es igual o mayor que min_points, este punto se considera punto central y se forma un clúster con los puntos que se encuentran dentro de un radio épsilon.
- Se añaden los puntos que no han sido considerados como puntos centrales al clúster más cercano siempre que se encuentren dentro del radio épsilon de algún punto central. En caso de que no sea así, ese punto se considerará ruido (outlier). Este es otro de los problemas que se encontraban en el algoritmo K-Means, su sensibilidad o poca robustez ante datos ruidosos o outliers, y que DBSCAN soluciona de esta manera.

El resultado puede ser algo similar a la siguiente figura, donde se ven dos clústeres diferenciados y algunos puntos que son considerados ruido al no ser vecinos de ningún punto central [35].

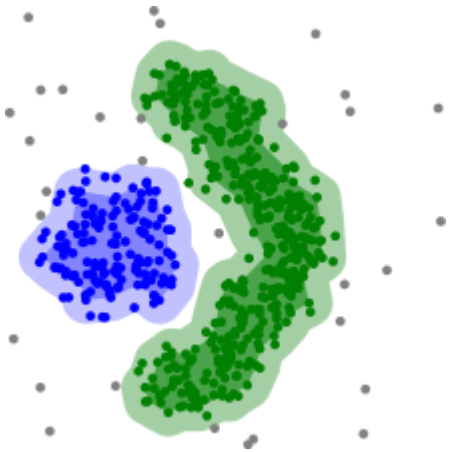


Figura 3-15. Ejemplo de clustering usando DBSCAN [35]

Al igual que con K-Means, también se van a comentar algunos problemas asociados a DBSCAN [34]:

- Es un algoritmo muy sensible a la elección de los parámetros épsilon y min_points. Esto hace que, en conjuntos de datos con densidades variables, sea difícil optimizar estos parámetros y disminuya la eficacia del algoritmo.
- Requiere de una capacidad computacional más alta que K-Means, pues debe realizar consultas de vecindarios para cada punto.
- No es un algoritmo completamente determinista, pues los puntos que se encuentran en la frontera entre un clúster y otro pueden formar parte de uno de los dos clústeres indistintamente, dependiendo del orden en el que se analicen los datos.
- Encuentra dificultades para detectar clústeres de densidades variables, pues se tendrían que ajustar los parámetros épsilon y min_points para cada clúster, cosa que no es posible ni tiene sentido con este algoritmo.

Con esto se da por finalizado el análisis de los problemas de clustering y de los dos principales algoritmos que se utilizan para dar solución a ellos en Machine Learning, K-Means y DBSCAN.

3.5.2 Regresión

En segundo lugar, se abordarán los problemas de regresión en Machine Learning. Los problemas de regresión consisten en la predicción de un valor numérico continuo a partir de unos datos de entrada. En este caso, a diferencia de los problemas de clustering en los que se utilizaba aprendizaje no supervisado, el modelo se entrena con etiquetas, siendo conocidos el valor de entrada y el valor a predecir. El objetivo de la regresión es encontrar la relación entre las variables de entrada y la variable de salida, de forma que se puedan hacer predicciones precisas sobre la variable de salida para nuevos datos [15].

Existen multitud de modelos de regresión, como la regresión lineal, la polinómica o la regresión de vecinos más cercanos (KNN – K Nearest Neighbours), pero con el fin de hacer un primer acercamiento didáctico se tratará la regresión lineal al ser el caso más básico. A continuación, se explican algoritmos de optimización que no son propios de Machine Learning, pero que ayudan a la comprensión de este tipo de problema.

3.5.2.1 Regresión lineal

En el contexto de la regresión lineal es importante tratar en primer lugar las funciones de pérdidas más utilizadas:

- Una de las más comúnmente utilizadas es la función del error cuadrático medio o MSE (Mean Square Error). Esta función se define como la media de los cuadrados de las diferencias entre los valores observados y los valores que ha predicho el modelo. Su expresión es la siguiente [36]:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Donde:

- n es el número de observaciones en el conjunto de datos.
- y_i es el valor observado de la variable de respuesta para la i-ésima observación.
- $\hat{y}_i$ es el valor predicho por el modelo para la i-ésima observación.

- Otra de las funciones más usadas, y muy similar al MSE, es el RMSE (Root Mean Square Error), que no es más que la raíz cuadrada del error cuadrático medio. Por lo tanto, su expresión es la siguiente [36]:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- Además de estas dos funciones, también es una alternativa el error medio absoluto o MAE (Mean Absolute Error), que calcula la media de las diferencias en valor absoluto entre los valores observados y los valores predichos por el modelo. Esta función es menos sensible a los valores atípicos en los datos que el MSE [36].

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\|y_i - \hat{y}_i\|^2$$

Como se ha explicado con anterioridad, el objetivo del modelo será minimizar la función de pérdida elegida, por lo que, extrapolándolo a estas funciones de pérdida, el modelo ajustará los parámetros para que la diferencia entre los valores observados y los valores predichos sea lo mínima posible.

3.5.2.2 Descenso por gradiente

Para minimizar la función de coste se utiliza el algoritmo de descenso por gradiente. El descenso por gradiente consiste en ajustar de forma iterativa los parámetros del modelo en la dirección en la que se reduce el valor de la función pérdidas. Esta dirección en la que se minimiza la función objetivo se logra mediante la derivada de la función, que indica la dirección en la que la función crece, por lo que se acabará usando el sentido contrario de la derivada [15].

De esta forma, el descenso por gradiente utiliza estos cálculos para actualizar los parámetros del modelo progresiva e iterativamente, desplazándose hasta el mínimo de la función objetivo.

Existen variantes del descenso por gradiente, que mejorar la velocidad de cálculo y la convergencia dependiendo del contexto en el que se use este método. Algunos de ellos son los siguientes:

- Descenso por gradiente por lotes (Batch Gradient Descent): En este método se calcula el gradiente de la función de coste utilizando todo el conjunto de datos de entrenamiento en cada paso. Al usar toda la información disponible puede conseguir una convergencia más rápida, aunque es computacionalmente más costoso [15].

- Descenso por gradiente por minibatch: En este caso, en lugar de utilizar todo el conjunto de datos de entrenamiento, se utilizan pequeños subconjuntos de los datos de entrenamiento seleccionados al azar. Gracias a esto, utiliza una cantidad moderada de memoria y es más eficiente computacionalmente. Si el tamaño del subconjunto es 1, es decir, se utiliza una única muestra del conjunto de entrenamiento en cada iteración, estamos ante el método de descenso por gradiente estocástico. [15].

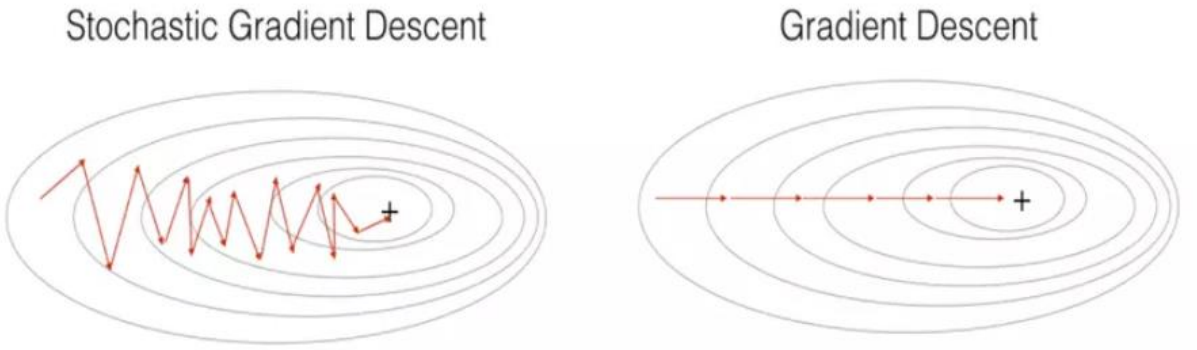


Figura 3-16. Comparación del descenso por gradiente estocástico con el estándar [37]

3.5.3 Clasificación

En último lugar, se tratará la clasificación en Machine Learning, uno de los problemas más comunes y extendidos en el ámbito de la inteligencia artificial, pues puede aplicarse a multitud de casos de uso, como reconocimiento de imágenes, diagnóstico médico, detección y clasificación de spam y otros muchos más.

La clasificación, al igual que la regresión, es un problema en el que se utiliza el aprendizaje supervisado, haciendo uso de etiquetas que se asignan a los datos de entrada en función de unas características o atributos que lo diferencian del resto. A diferencia de en los problemas de regresión, en los que la respuesta era un valor continuo, en los problemas de clasificación se obtiene una respuesta discreta.

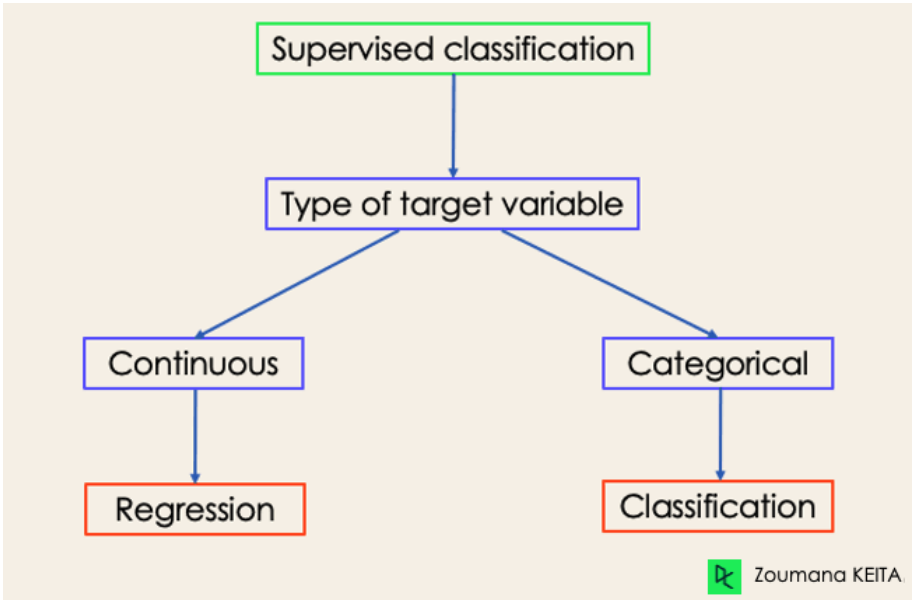


Figura 3-17. Diferencia entre regresión y clasificación [38]

En este apartado se abordará la clasificación en Machine Learning haciendo distinción entre la clasificación binaria, es decir, entre dos clases, y la clasificación multiclase, en la que el modelo debe elegir entre más de dos clases. [39]

3.5.3.1 Clasificación binaria

En clasificación binaria, el objetivo es clasificar los datos de entrada en dos posibles categorías. Para ello, los datos de entrenamiento se etiquetan de forma binaria, acorde con lo que se quiera predecir: verdadero o falso, 0 o 1, spam o no spam, etc. Un ejemplo de esto es la figura a continuación, que ilustra un modelo de Machine Learning que clasifica imágenes de gatos como positivas y de perros como negativas para la distinción entre ambos animales. [40]

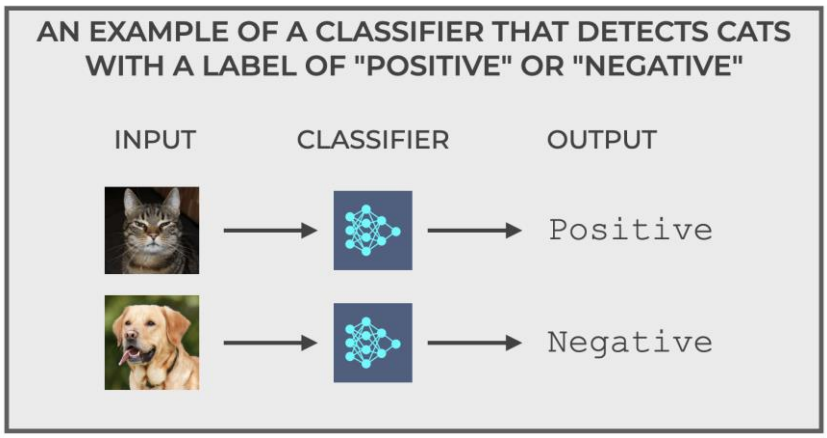


Figura 3-18. Ejemplo de clasificación binaria con gatos y perros [40]

Para este tipo de tareas es común utilizar algoritmos como el de regresión logística, árboles de decisión, de máquinas de vectores de soporte o KNN. A continuación, se tratará el algoritmo de regresión logística.

3.5.3.2 Regresión logística

La regresión logística es un algoritmo de machine learning utilizado comúnmente para problemas de clasificación binaria. Este algoritmo hace uso de la función logística o sigmoide, que convierte cualquier valor real en una probabilidad entre 0 y 1.

La expresión de la función sigmoide tiene la siguiente forma [40]:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Donde z es la combinación lineal de las características de entrada y los coeficientes del modelo:

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

Siendo x_n las características de entrada, es decir, las variables independientes que se utilizan como entrada del modelo, y $\beta_1, \beta_2, \ldots, \beta_n$ los pesos atribuidos a cada una de estas características. Por otro lado, β_0 es el término de sesgo.

La función sigmoide convierte los valores de z, que pueden estar comprendidos entre menos infinito y más infinito, a un rango entre 0 y 1, lo que permite interpretar la salida del modelo como una probabilidad.

Junto con la regresión logística se utiliza una función de pérdida para evaluar la precisión de las predicciones del modelo en comparación con los valores de las etiquetas. La función de pérdida que se utiliza es la de entropía cruzada, que se define a través de la siguiente expresión [40]:

$$L(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

Donde y_i es la etiqueta real (0 o 1) para la i-ésima instancia, $\hat{y}_i$ es la probabilidad predicha por el modelo para la i-ésima instancia, y N es el número total de instancias en el conjunto de datos de entrenamiento.

En la siguiente figura se representa la función sigmoide y se puede comprobar que los valores de la función están

comprendidos entre 0 y 1, por lo que ante cualquier valor de entrada, representado como x en la gráfica, se le asocia un valor entre 0 y 1, que puede interpretarse como una probabilidad.

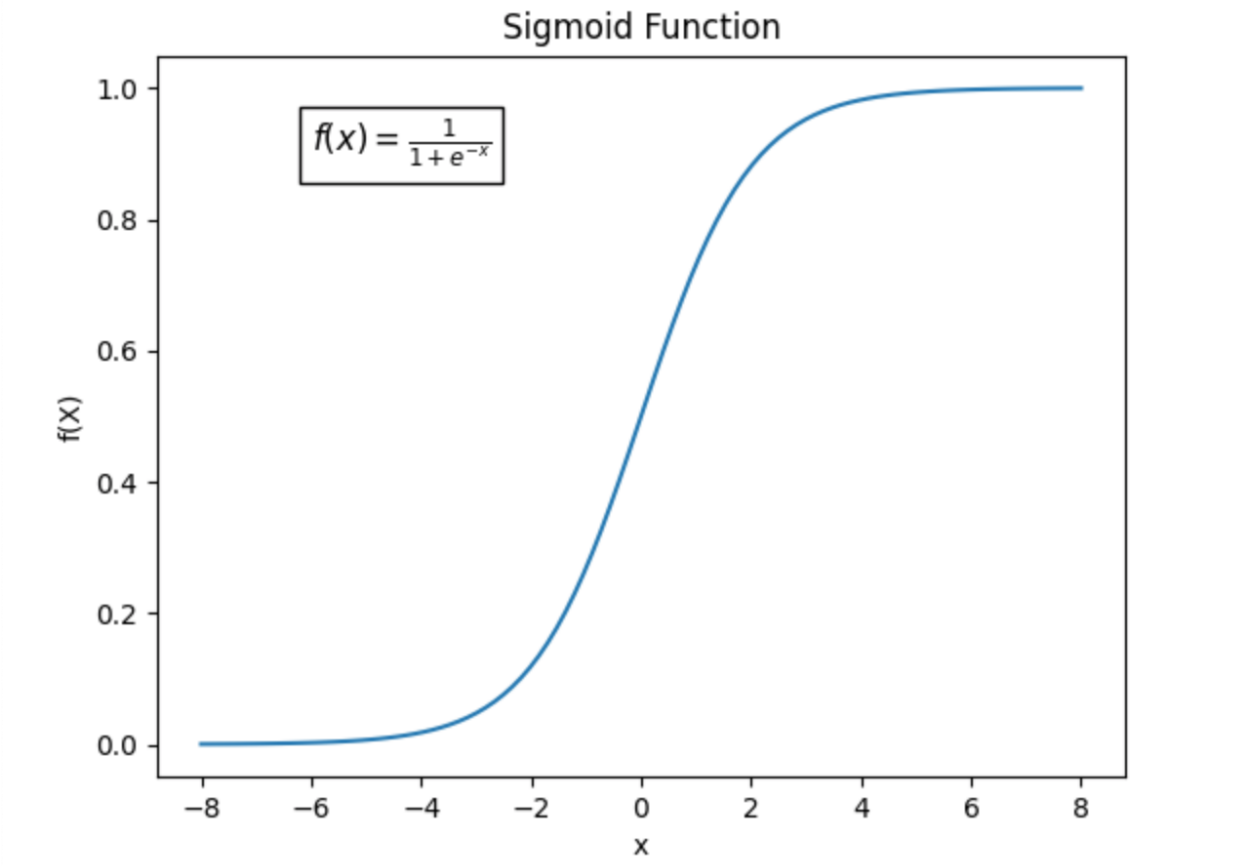


Figura 3-19. Representación de la función sigmoide [41]

3.5.3.3 Clasificación multiclase

Por otro lado, está la clasificación multiclase, para la que existen distintos enfoques que para la clasificación binaria. Los principales enfoques son los siguientes [42]:

- One-vs-Rest o One-vs-All: En este enfoque se utilizan múltiples clasificadores binarios, uno para cada posible clase. Cada uno de estos clasificadores binarios predice si una instancia pertenece o no a una clase. Tras esto, se selecciona la clase con la mayor puntuación entre todos los clasificadores.
- One-vs-One: En este enfoque se entrenan clasificadores binarios para cada par de clases posibles. Por lo tanto, si hay N clases posibles, se entrenan N(N-1)/2 clasificadores. Para predecir la salida, se elige la clase con mayor número de "victorias" sobre el resto de clases.
- El último de los enfoques consiste en entrenar un solo clasificador que pueda manejar múltiples clases directamente. Para ello, se utiliza la función Softmax, que se verá a continuación.

3.5.3.4 Softmax

Softmax es una función comúnmente utilizada en la capa de salida de las redes neuronales para problemas de clasificación multiclase. La salida de esta función será un vector de N dimensiones, siendo N el número de clases, cuyos elementos sumarán 1, representando de esta forma la probabilidad de que la entrada corresponda con cada una de las posibles clases.

La función Softmax toma un vector de números reales y lo convierte en un vector de probabilidades usando la siguiente expresión [43]:

$$\hat{y}_j = \frac{e^{z_j}}{\sum_{n=1}^{N} e^{z_n}}$$

Donde:

- z es el vector de activación antes de aplicar la función Softmax.
- $\hat{y}_j$ es la probabilidad predicha para la clase j.
- N es el número total de clases.

En la siguiente figura se representa la función Softmax. La función Softmax tiene la misma forma que la función sigmoide, pero se caracteriza por operar en un vector de valores de entrada, mientras que la sigmoide actúa ante un único valor de entrada. Este hace que la función Softmax sea de utilidad en problemas de clasificación multiclase.

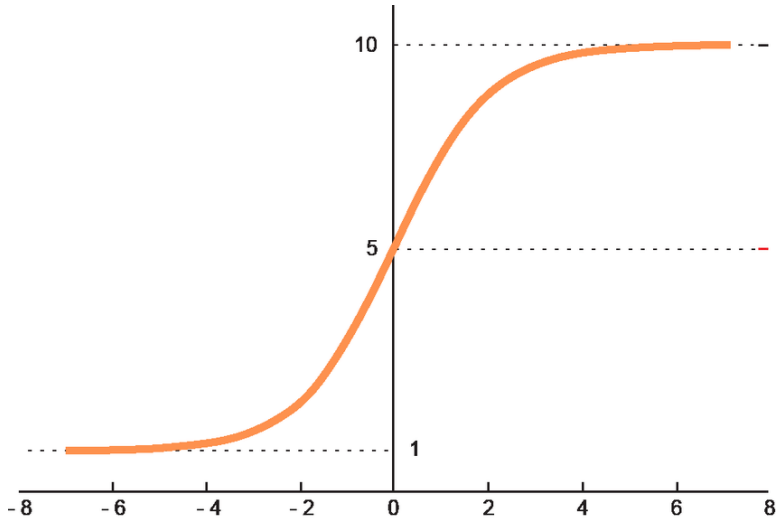


Figura 3-20. Representación de la curva de la función softmax [44]

Como función de pérdida, una de las expresiones utilizadas es:

$$L(y, \hat{y}) = -\sum_{i=1}^{K}\sum_{j=1}^{N} y_{i,j} \log(\hat{y}_{i,j})$$

Donde:

- $y_{i,j}$ es la etiqueta real (0 o 1) para la i-ésima instancia y la j-ésima clase.
- $\hat{y}_{i,j}$ es la probabilidad predicha por el modelo para la i-ésima instancia y la j-ésima clase.
- K es el número total de instancias en el conjunto de datos de entrenamiento.
- N es el número total de clases.

3.5.3.5 Matriz de confusión, ROC y AUC

Una matriz de confusión es una herramienta que permite la evaluación del rendimiento de un modelo de clasificación. Proporciona un resumen de las predicciones que ha realizado el modelo, comparándolas con los valores reales indicados en las etiquetas.

Una matriz de confusión es una matriz de dimensiones N x N, siendo N el número de posibles clases. En las filas de la matriz se representan las instancias de las predicciones del modelo, mientras que en las columnas se representan las instancias de la clase real, si bien es cierto que pueden representarse de la forma contraria, según el criterio que se siga. Las celdas de la matriz contienen el número de instancias clasificadas correctamente y los errores de predicción del modelo.

Los elementos de la matriz de confusión se pueden clasificar en [45]:

- Verdaderos Positivos o True Positives (TP): Datos que son positivos y que han sido clasificados como positivos por el modelo.
- Falsos Positivos o False Positives (FP): Datos que son negativos y que han sido clasificados de forma incorrecta por el modelo como positivos.
- Verdaderos Negativos o True Negatives (TN): Datos que son negativos y que han sido clasificados correctamente por el modelo como negativos.
- Falsos Negativos o False Negatives (FN): Datos que son positivos y que han sido clasificados incorrectamente por el modelo como negativos.

Como puede verse en la figura a continuación, la diagonal principal de la matriz contiene los verdaderos positivos y los verdaderos negativos, por consiguiente, contiene los aciertos del modelo en la predicción. Las otras celdas de la matriz representan errores de clasificación del modelo.

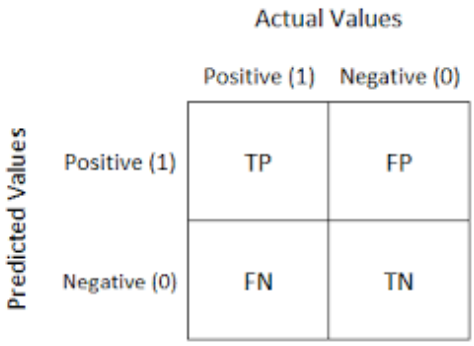


Figura 3-21. Ejemplo de matriz de confusión 2 x 2 [45]

Otra de las herramientas utilizadas recibe el nombre de ROC, Receiver Operating Characteristic. ROC es una representación gráfica que muestra la relación entre verdaderos positivos y falsos positivos. En el eje horizontal se representa la tasa de falsos positivos y en el eje vertical la relación de verdaderos positivos. Cada punto de la curva corresponde a un umbral de clasificación diferente. Una curva ROC para un caso ideal se acercaría al vértice superior izquierdo del gráfico, indicando una tasa alta de verdaderos positivos y baja de falsos positivos.

Otro concepto es el AUC (Area Under the Curve), que es el área bajo la curva ROC, y que da una medida cuantitativa del rendimiento del modelo. Un AUC de 1 indica un modelo perfecto, que distingue con éxito en todos los casos. Un AUC de 0.5 indica un rendimiento del modelo aleatorio, caso en el que el modelo es tan bueno como lanzar una moneda al aire. Un AUC menor que 0.5 indica un rendimiento peor que en el caso aleatorio, lo que significa que el modelo está fallando al clasificar la mayoría de los datos. [15]

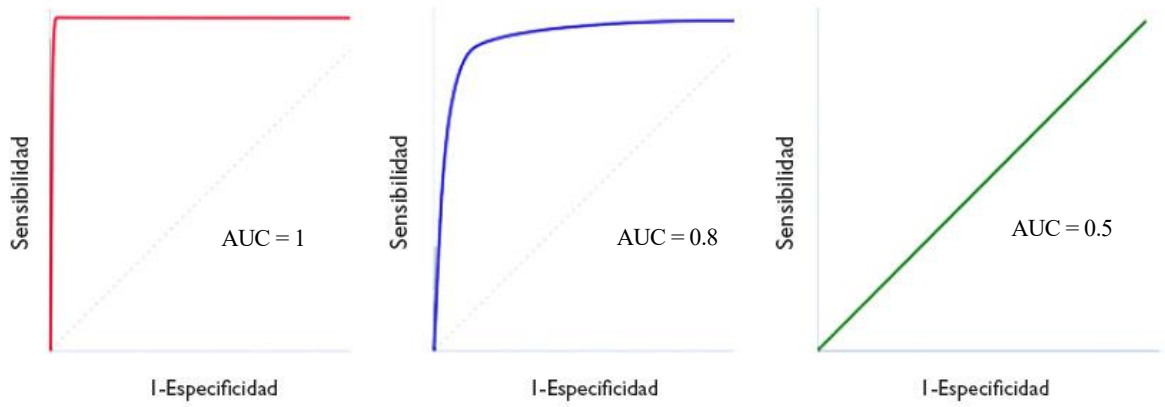


Figura 3-22. ROC y AUC [46]

3.6 Métricas de evaluación de modelos

Por último, en este apartado, se van a explicar algunas de las métricas de evaluación de modelos más utilizadas en Machine Learning, y que posteriormente se utilizarán a la hora de evaluar el resultado obtenido en el modelo que se va a desarrollar en este trabajo.

3.6.1 Precisión

La precisión es una métrica de evaluación utilizada en problemas de clasificación. Esta métrica se define como la proporción de predicciones positivas correctas respecto al total de predicciones realizadas por el modelo. Es decir:

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

Siendo TP el número de verdaderos positivos, es decir, instancias 'positivas' correctamente clasificadas, y FP el número de falsos positivos, es decir, instancias que realmente eran negativas y han sido clasificadas de forma incorrecta como positivas.

La precisión se centra en la calidad de las predicciones positivas que realiza un modelo, por lo que es de gran importancia en casos en los que un falso positivo puede ser muy costoso o dañino. [47]

3.6.2 Recall o sensibilidad

El recall es una métrica que mide la efectividad que tiene el modelo en la identificación de todos los casos relevantes del conjunto de datos. Mide la proporción de casos positivos identificados correctamente sobre el total de casos que son verdaderamente positivos. También es conocido como sensibilidad y su fórmula es la siguiente:

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

Siendo TP (True Positives) el número de casos positivos correctamente identificados y FN (False Negative) el número de ejemplos positivos que el modelo identificó incorrectamente como negativos.

La sensibilidad es de vital importancia cuando el daño que puede causar un falso negativo es alto. Por ejemplo, en el diagnóstico médico, al identificar un paciente que está enfermo como uno sano puede hacer que este no reciba un tratamiento a tiempo, aumentando la gravedad del problema. [47]

3.6.3 Mean Average Precision (mAP)

El Mean Average Precision (mAP) es una métrica muy utilizada para medir el rendimiento de un modelo en tareas relacionadas con la detección de objetos.

El AP es una métrica que mide la precisión de las predicciones de un modelo en una clase específica. Se calcula como el área bajo la curva (AUC) precisión-recall. El mAP no es más que el promedio de los AP de todas las clases existentes para el conjunto de datos en cuestión. Esta métrica es de especial utilidad cuando se requiere como resultado un único valor numérico, es decir, proporciona una medida unificada del rendimiento del modelo. Un alto valor de mAP indica que el modelo es capaz de distinguir con precisión entre distintas clases. [48]

Un ejemplo de curva precisión-recall es el siguiente, extraído del modelo para detección de pases que se comienza a desarrollar en el capítulo 5 del trabajo:

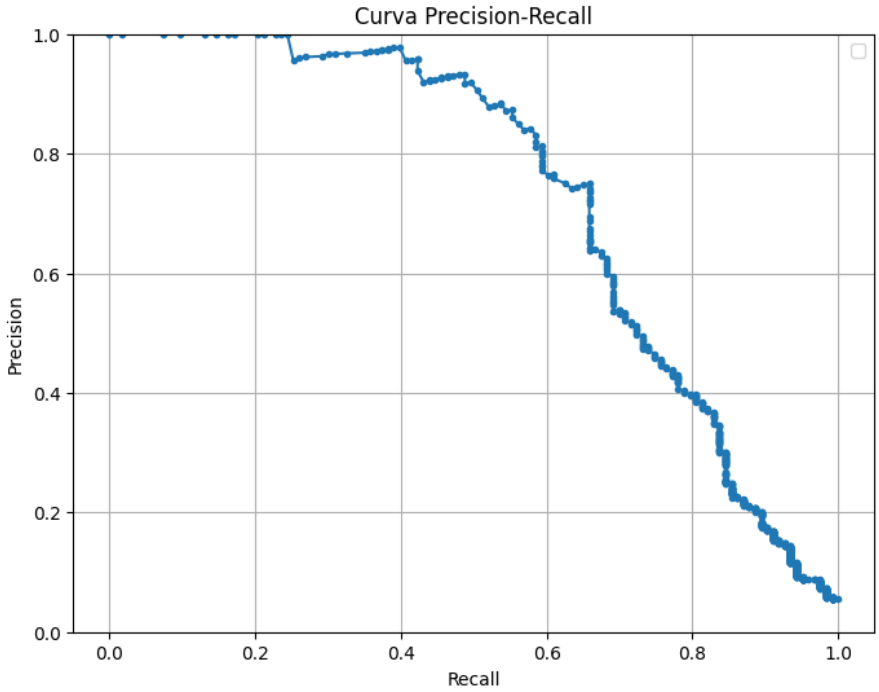


Figura 3-23. Curva Precisión-Recall

4 REDES NEURONALES

Tras haber ahondado en las ideas principales del Machine Learning, toca el turno de las redes neuronales. Las redes neuronales son un subcampo específico dentro del Machine Learning, pero debido a sus altas capacidades y a sus diversos usos, se ha decidido dedicar un capítulo para ellas en exclusiva.

Las redes neuronales son modelos computacionales compuestos por capas de neuronas artificiales conectadas entre sí, que aprenden patrones complejos para realizar tareas de manera autónoma, tratando de emular el funcionamiento del cerebro humano.

Con el auge de la inteligencia artificial, las redes neuronales se postulan como una de las herramientas más poderosas para abordar problemas de dominios muy diversos, como reconocimiento de imágenes, procesamiento del lenguaje natural, análisis de datos, robótica y muchos más.

En este capítulo, se hará un recorrido a través de las redes neuronales, comenzando por sus orígenes, su evolución en el último siglo y su importancia en la actualidad, los tipos de redes que existen, haciendo hincapié en las redes neuronales convolucionales, con el objetivo de encarar el siguiente capítulo del trabajo, donde se usarán para llevar a cabo el objetivo final.

4.1 Orígenes y evolución de las redes neuronales

La idea de red neuronales comenzó como un modelo que trataba de emular el comportamiento de las neuronas del cerebro. En el año 1943, el neurofisiólogo Warren McCulloch y el matemático Walter Pitts consideraron un modelo computacional que no era capaz de aprender, pero que sería uno de los primeros pasos hacia las redes neuronales modernas, pues marcaba el camino para que de ahí en adelante los campos de investigación se dividieran en dos: los procesos biológicos y la aplicación de las redes neuronales a la inteligencia artificial [49].

Unos años más tarde, en 1949, Donald Hebb consiguió llevar estas ideas a otro nivel, creando una hipótesis sobre el aprendizaje que sería conocida como Hebbian learning. Este modelo de aprendizaje es equivalente al aprendizaje no supervisado que ya se ha desarrollado en el capítulo anterior.

Otro avance muy importante fue la creación del perceptrón en 1958, por Frank Rosenblatt. Era un sistema con una relación entrada-salida y que ejecutaba un algoritmo de clasificación binaria. Su funcionamiento se basaba en recibir un conjunto de entradas, hacer una suma de las entradas con distintos pesos y devolver como salida un ‘0’ si el resultado estaba por debajo del umbral o un ‘1’ en caso contrario. Lo interesante de este proceso era que los pesos serían aprendidos a medida que se probaran más entradas, minimizando las diferencias entre las salidas que había calculado el perceptrón y las deseadas [50].

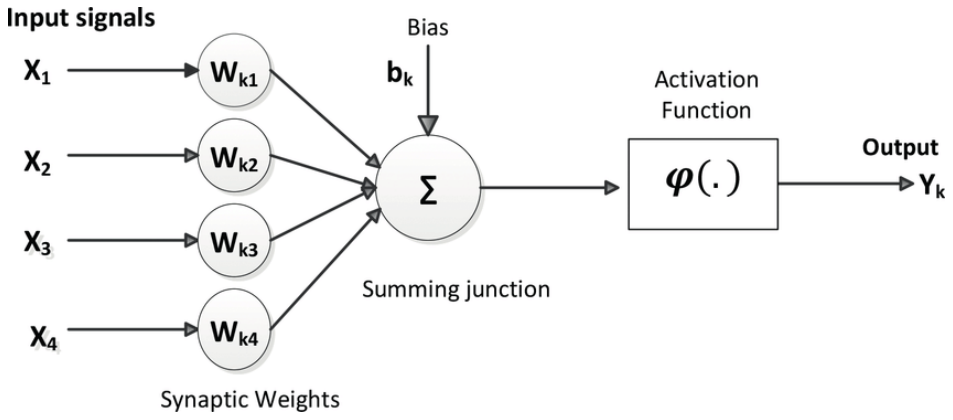


Figura 4-1. Neurona de McCulloch-Pitts [51]

En el año 1969, Minsky y Papert demostraron que el perceptrón no era capaz de procesar problemas no lineales, como la lógica detrás de una puerta XOR y que los computadores de la época no tenían suficiente potencia para utilizar redes neuronales útiles. Tras la publicación del libro de Minsky, no se llevó a cabo ninguna gran investigación sobre redes neuronales en al menos 10 años, lo que fue conocido como el invierno de la IA. [49]

La solución a este problema fue el perceptrón multicapa o multilayer perceptron (MLP). El perceptrón multicapa consiste en una red neuronal artificial que consta de múltiples capas de neuronas. Cada neurona de cada capa está conectada a todas las neuronas de la capa siguiente. Estas conexiones transmiten señales desde las neuronas de una capa hasta las neuronas de la capa siguiente, propagando la información a través de la red. Una de las claves del perceptrón multicapa es su capacidad para aprender y modelar funciones no lineales, como por ejemplo, el cálculo de la puerta XOR. [50]

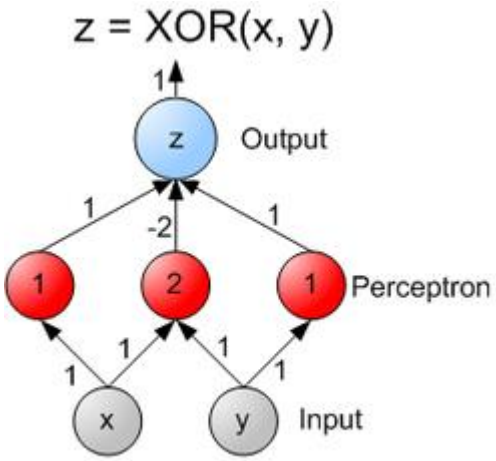


Figura 4-2. Red neuronal de dos capas que realiza el cálculo de la XOR [52]

El aprendizaje en un perceptrón multicapa se realiza mediante el algoritmo de propagación hacia atrás (backpropagation), que ajusta los pesos de las conexiones entre las neuronas para minimizar la función de coste.

En los años 90, las redes neuronales volvieron después de ese "invierno", cautivando la atención del mundo y cumpliendo con las expectativas que se tenían décadas atrás. Se desarrollaron nuevos algoritmos y evolucionaron otros muchos, hasta llegar a la actualidad, tiempo en el que, gracias a las mejoras computacionales, las mejoras en los algoritmos de entrenamiento y la accesibilidad a una mayor cantidad de datos han permitido que el uso de las redes neuronales se convierta en una realidad del día a día del ser humano. [49]

4.2 Conceptos de redes neuronales

A continuación, se tratan los conceptos más relevantes para el uso y entendimiento de las redes neuronales [53]:

- Neurona: Es la unidad básica de la red neuronal y como se ha comentado en el apartado anterior, está inspirada en las neuronas biológicas. Una neurona recibe un conjunto de entradas (inputs), los procesa y devuelve una salida (output).
- Pesos: Son parámetros ajustables que determinan la influencia de las conexiones entre las neuronas. Durante el entrenamiento de la red neuronal, los pesos se ajustan de forma iterativa con el objetivo de minimizar la función de coste, usando, por ejemplo, el algoritmo de descenso por gradiente.
- Sesgo: Es otro parámetro que se utiliza en el cálculo de la activación de una neurona. Ayuda a ajustar la salida junto con los pesos para mejorar las relaciones de los datos.

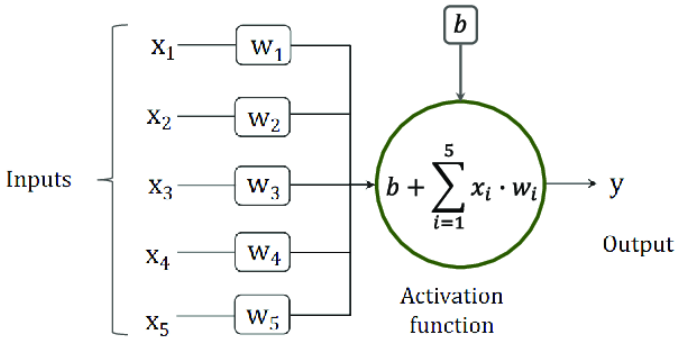
  La salida de una neurona, expresada de forma matemática es la siguiente:

$$y_i = f\left(\sum_{j=1}^{n} w_{ij}\, x_j + b_i\right)$$

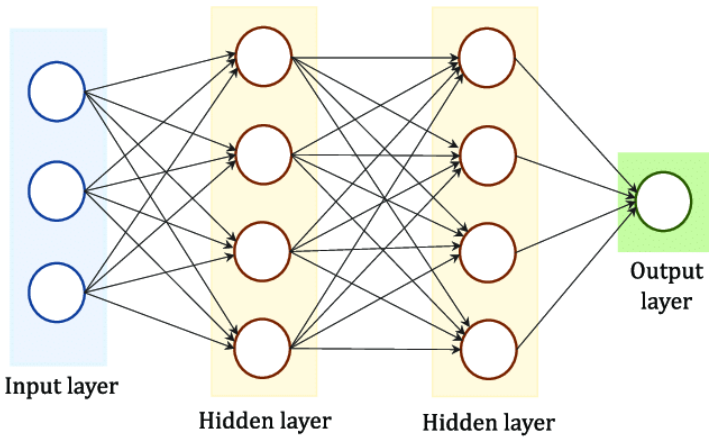
Donde:

- y_i es la salida de la neurona i.
- f es la función de activación utilizada.
- x_j es la entrada de la neurona.
- w_{ij} es el peso asociado a la j-ésima entrada de la neurona i.
- b_i es el sesgo de la neurona i.

- Capas: Las redes neuronales están compuestas por capas de neuronas. Comúnmente, las capas se clasifican en:
  - Capa de entrada: Es la primera capa que se encuentra en la red y que recibe los datos de entrada.
  - Capas ocultas: Son las capas intermedias situadas entre la capa de entrada y la de salida. Se encargan de la mayor parte del procesamiento.
  - Capa de salida: Es la última capa de la red, encargada de producir la salida final del modelo.



(a) A perceptron with $N = 5$ inputs x_i and one output y.



(b) Feed-forward fully connected neural network.

Figura 4-3. Ejemplo de red neuronal artificial con cuatro capas [54]

- Propagación hacia adelante y hacia atrás [53]:
  - Hacia adelante: Es el proceso de pasar datos de entrada a través de la red hasta obtener una salida. Este proceso requiere de la realización de cálculos usando las funciones de activación

en cada neurona

- Hacia atrás: Es el proceso de ajustar los pesos de la red neuronal usando el error de la predicción obtenida por la red. Normalmente se calcula el gradiente del error con respecto a cada peso usando la regla de la cadena para actualizar los pesos en la dirección que minimiza el error.

- Herramientas: Las bibliotecas más populares para crear y entrenar redes neuronales son Keras, TensorFlow y Pytorch.

4.3 Tipos de redes neuronales

Una vez desarrollados los conceptos principales para poder entender el funcionamiento de las redes neuronales, se procede a describir los principales tipos de redes neuronales, haciendo hincapié en las redes neuronales convolucionales, que se utilizarán en el capítulo siguiente para la detección de eventos en partidos de fútbol.

4.3.1 Perceptrón multicapa (MLP)

El perceptrón multicapa es el tipo de red neuronal del que se ha estado hablando hasta el momento. Consiste en múltiples capas de neuronas: una de entrada, una o varias capas ocultas y una de salida. Al estar formado por múltiples capas, es capaz de resolver problemas que no son linealmente separables, lo cual es la principal limitación del perceptrón. Además, es una red feedforward, es decir, las conexiones entre las neuronas no forman ciclos, cosa que sí ocurre en otros tipos de redes neuronales como las recurrentes [55].

El perceptrón multicapa puede estar totalmente conectado o localmente conectado. De estar totalmente conectado (fully connected), cada salida de una neurona es entrada de todas las neuronas de la capa siguiente. Si está localmente conectado, cada salida de una neurona es entrada de una región de neuronas de la capa siguiente.

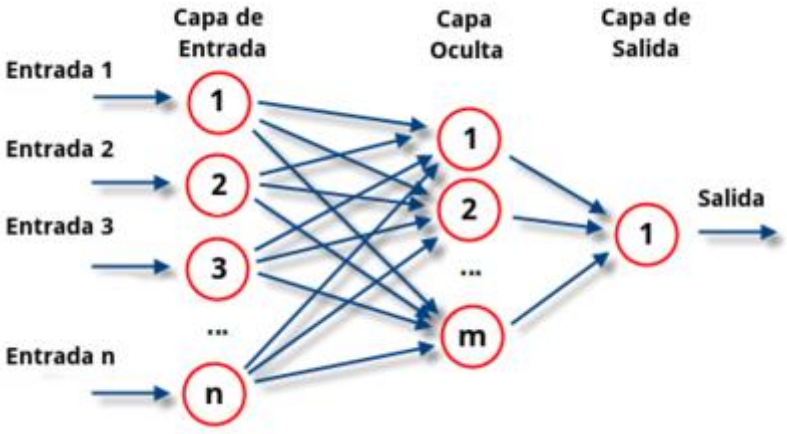


Figura 4-4. Perceptrón multicapa totalmente conectado [55]

4.3.1.1 Funcionamiento

En primer lugar, los datos de entrada se propagan a través de la red, atravesando todas las capas y siendo modificados por la multiplicación de los pesos y la suma del sesgo, además de aplicarse la función de activación que se haya elegido. Las funciones de activación más utilizadas en este tipo de redes son ReLU, sigmoide o la tangente hiperbólica, que se utilizan para introducir no linealidades en la red neuronal y habilitar así que aprenda funciones más complejas.

Tras esto, se procede a utilizar la propagación hacia atrás como se ha explicado anteriormente, con el objetivo de ajustar los pesos y los sesgos mediante el gradiente de la función de coste [15].

4.3.2 Redes neuronales recurrentes

Las redes neuronales recurrentes (RNN) son una clase de redes neuronales en las que las conexiones entre las neuronas forman ciclos. Esto permite el procesamiento de datos secuenciales, pues son capaces de mantener un "estado" o memoria de entradas anteriores. A diferencia de las redes feedforward, que procesan los datos en una única dirección, este tipo de redes neuronales tienen conexiones que permiten el flujo de la información hacia delante y hacia atrás. Las redes neuronales recurrentes se utilizan con asiduidad en el campo del procesamiento del lenguaje natural o el reconocimiento del habla, entre otros campos.

En una RNN, cada capa oculta se calcula a partir de la entrada actual y la salida de la capa anterior. Además, tienen parámetros compartidos a lo largo de la secuencia. [56]

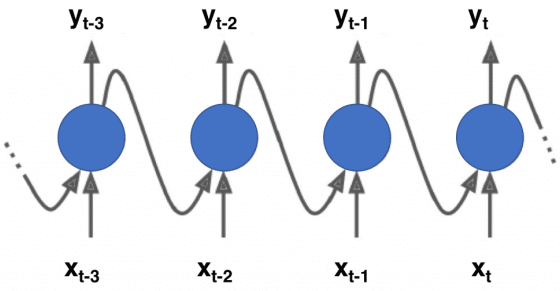


Figura 4-5. Red neuronal recurrente [57]

En la figura 4-5 se pude ver como cada neurona recibe la entrada x de la capa anterior, así como la salida del instante anterior de la misma capa, para generar su salida y.

Puesto que la salida de una neurona en un instante de tiempo determinado es una función de los instantes de tiempo anteriores, se podría decir que una neurona tiene "memoria". La parte de una red neuronal que preserva el estado a través del tiempo se suele conocer como memory cell.

El hecho de tener esa memoria interna hace que las redes neuronales recurrentes sean de gran utilidad en problemas de aprendizaje automático que involucran datos secuenciales, pues consiguen recordar información relevante de la entrada, manteniendo un contexto de la información y consiguiendo predecir lo que puede venir después [57].

4.3.2.1 Long-Short Term Memory

Una extensión de las redes neuronales recurrentes y que merece la pena mencionar son las Long-Short Term Memory (LSTM). Estas permiten a las RNN recordar entradas durante un largo período de tiempo gracias a una celda en la que almacena la información. La célula decide si almacenar o eliminar la información que contiene en su interior en función de la importancia que asigna a la información que se está recibiendo, que vendrá dada por los pesos.

En una neurona LSTM hay tres puertas a las celdas de información: la puerta de entrada (input gate), la puerta para olvidar la información (forget gate) y la puerta de salida (output gate). Estas puertas se encargan de gestionar si se permite o no una nueva entrada, si se elimina la información porque ya no se considera relevante o se deja que afecte a la salida. [57]

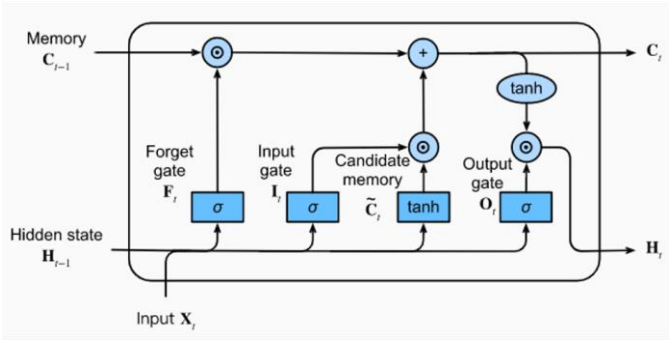


Figura 4-6. Neurona LSTM [58]

4.3.3 Redes neuronales generativas antagónicas (GAN)

Las redes neuronales generativas antagónicas son un tipo de red neuronal diseñada con el objetivo de generar nuevos datos similares a los datos de entrenamiento. Este objetivo se diferencia claramente de otros que se han estudiado a lo largo del trabajo como la clasificación, la regresión u otros modelos discriminativos, pues en este caso lo que se intenta es aprender de qué forma están distribuidos los datos de entrada para poder generar nuevas muestras a partir de ellos.

En las redes generativas antagónicas se tienen dos modelos que funcionan al mismo tiempo, uno generativo y otro discriminativo. Generalmente, la red generativa aprende a asignar elementos de un espacio latente a una distribución de datos determinada, mientras que la red discriminativa diferencia entre elementos del conjunto de datos original y los candidatos producidos por el generador. El objetivo principal de estas redes es aumentar el índice de error de la red discriminativa, de esta forma se estaría consiguiendo engañar a la red discriminativa, lo que significa que los nuevos elementos generados por la red generativa parecen provenir del conjunto de datos original. [59]

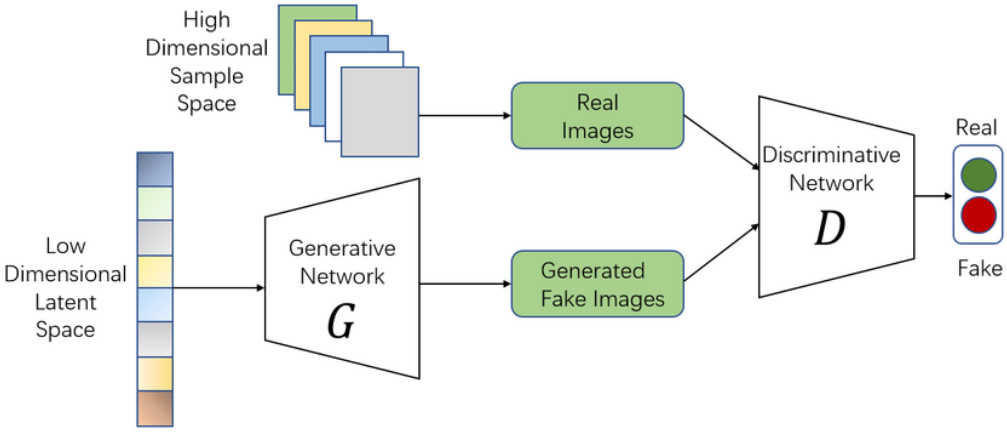


Figura 4-7. Red neuronal generativa antagónica [60]

Las redes neuronales generativas se han utilizado para producir imágenes de diseño industrial, para mejorar la resolución de imágenes de baja calidad, convirtiéndolas en imágenes de alta resolución, para crear entornos propios de videojuegos o de realidad virtual, para ampliar conjuntos de datos generando datos artificiales entre otras muchas utilidades.

4.3.4 Redes neuronales convolucionales

Por último, se tratan las redes neuronales convolucionales, desarrollando sus conceptos principales y haciendo de enlace con el siguiente capítulo, en el que se desarrollará la parte principal del trabajo.

Una red neuronal convolucional es una variación de un perceptrón multicapa en la que su aplicación se realiza en matrices bidimensionales, haciendo que sean muy efectivas para tareas de visión artificial.

4.3.4.1 Conceptos de interés

En este apartado se describen algunos conceptos de interés que ayudan a comprender el funcionamiento de las redes neuronales artificiales.

4.3.4.1.1 Kernel

Un kernel o filtro es una matriz de pesos utilizada para realizar la operación de convolución sobre la entrada. Los kernels se van desplazando sobre la entrada y se utilizan para extraer características locales de los datos, como patrones o texturas. [61]

La expresión de la convolución es la siguiente:

$$Salida(i,j) = \sum_{k,l} Entrada(i+k, j+l) \cdot Kernel\ (k,l)$$

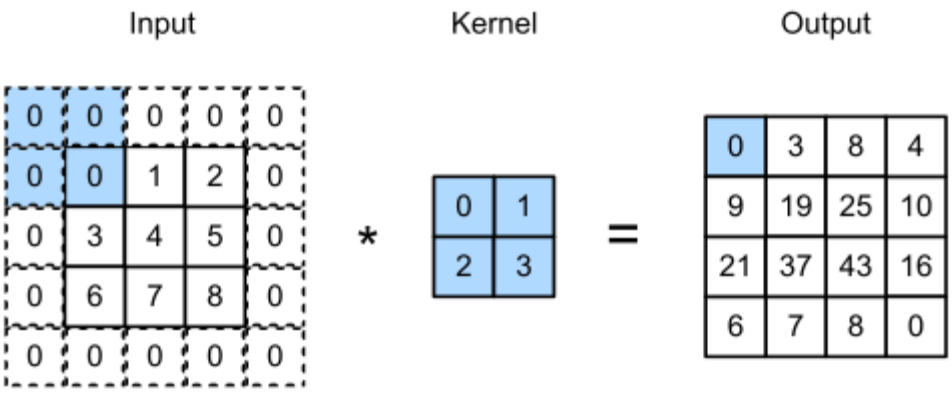


Figura 4-8. Operación del kernel sobre la entrada [62]

4.3.4.1.2 Stride

El stride es el paso con el que se desplaza el kernel sobre la entrada, es decir, el número de píxeles que el kernel se mueve cada vez que se aplica la convolución. Cuanto mayor sea el valor del stride, la dimensión de la salida será menor. [61]

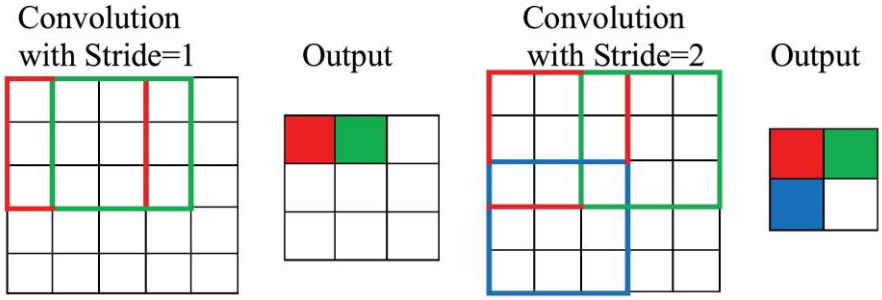


Figura 4-9. Diferencia en la salida con stride 1 y stride 2 [63]

4.3.4.1.3 Padding

El padding consiste en añadir ceros alrededor de los bordes de la entrada antes de aplicar la convolución. Se utiliza para controlar el tamaño de la salida. Normalmente, se suele añadir suficiente padding para que la salida tenga el mismo tamaño que la entrada o no añadir nada de padding.

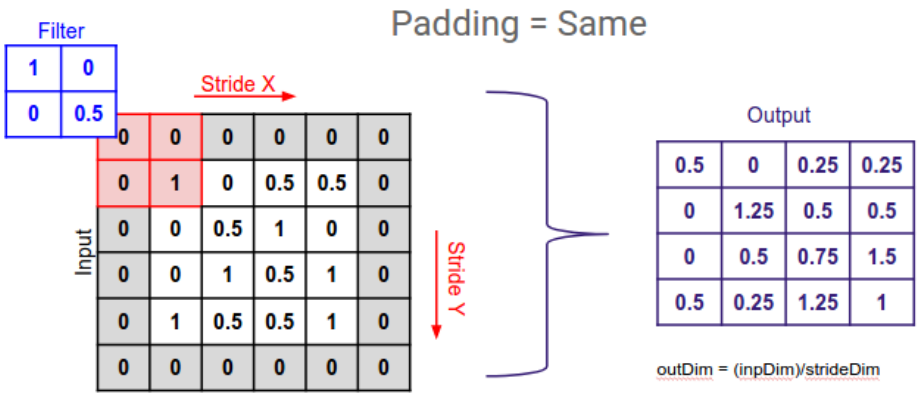


Figura 4-10. Padding para obtener el tamaño de la salida igual que el de la entrada [64]

4.3.4.1.4 Pooling

El pooling es una operación que se utiliza para reducir la dimensionalidad de los mapas de características generados por las capas convolucionales, tratando de retener la información más importante. Al reducir la dimensionalidad de los mapas de características se reduce la carga computacional.

Los tipos de pooling son:

- Max pooling: Se toma una ventana, normalmente de tamaño 2x2, y se toma el valor máximo de cada ventana.
- Average pooling: Se calcula el valor promedio de cada ventana que se toma.

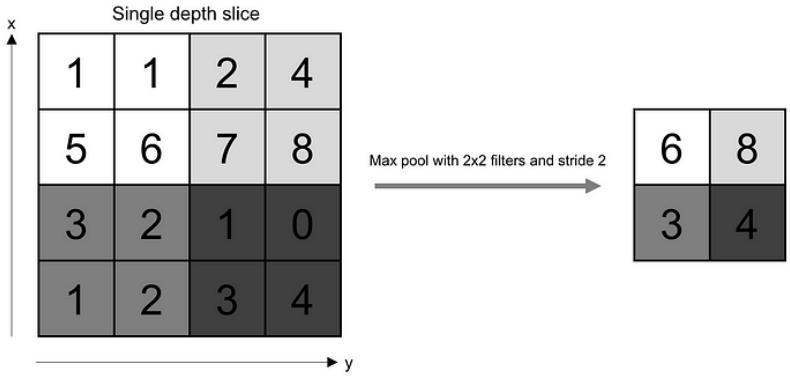


Figura 4-11. Ejemplo de Max pooling con una ventana de tamaño 2x2 [61]

4.3.4.2 Estructura de capas

Las redes neuronales convolucionales tienen una estructura de capas específica, diseñadas para procesar datos espaciales con múltiples dimensiones como imágenes o vídeos. Una posible estructura de capas es la que se explica a continuación: [61]

- Datos de entrada: Los datos de entrada a una CNN suelen ser imágenes representadas como arrays de dimensiones altura, ancho y canales de color.
- Capa convolucional: Es una de las partes principales de las CNN y ocupa la parte principal de la carga computacional. Esta capa se encarga de aplicar el kernel a la entrada, produciendo una representación bidimensional de la imagen, conocida como mapa de activación.
- Capa de pooling: Las capas de pooling se colocan tras las capas convolucionales y las de activación en algunos puntos de la red con el objetivo de reducir el tamaño de las salidas de estas capas, disminuyendo la carga computacional y la cantidad de pesos que se necesitan. A pesar de que existen varias funciones

de pooling, la más común es el max pooling, explicado anteriormente.

- Capa totalmente conectada (fully connected): Como ya se ha comentado, las neuronas de estas capas están completamente conectadas a las neuronas de las capas anteriores y las neuronas de la capa siguiente. Esta capa ayuda a mapear la representación entre la entrada y la salida.
- Capas no lineales: Al ser la convolución una operación lineal, se necesitan capas que aporten no linealidad a los mapas de activación que se generan a la salida de las capas convoluciones para conseguir adaptarse a la no linealidad de las imágenes. Algunas de las más comunes son:
    - Sigmoide: La función sigmoide ya ha sido explicada en el capítulo referente a Machine Learning y también tiene un uso importante en las redes neuronales. Como ya se mencionó, la función sigmoide ajusta el valor del número a un rango entre 0 y 1.

      Con la función sigmoide puede aparecer un problema al "aplanar" los valores de entrada, puesto que en regiones donde la entrada es muy alta o muy baja, el gradiente será cercano a cero. Los gradientes podrían volverse tan pequeños que no actualizaran los pesos de la red.
    - Tangente hiperbólica (tanh): La función tangente hiperbólica ajusta el valor de entrada en el rango de -1 a 1. En esta función puede ocurrir el mismo problema de desvanecimiento de gradiente que se ha explicado para la función sigmoide.
    - ReLU: La función ReLU (Rectified Linear Unit) se ha convertido en una de las más populares para este tipo de problemas. La función ReLU se define como:

      $$ReLU(x)=max(0,x)$$

      Es decir, devuelve el valor de entrada si es positivo y 0 de lo contrario.

      En comparación con la función sigmoide y la tanh, ReLU ofrece una mayor seguridad en cuanto a desvanecimiento de gradiente y también acelera la convergencia hasta en seis veces con respecto a las otras dos opciones planteadas, pues reduce la complejidad de las operaciones a una simple comparación.

      Uno de los problemas que pueden ocurrir al utilizar esta función es que existan neuronas que no vuelvan a ser actualizadas y no contribuyan al modelo debido a que el valor de gradiente que ha obtenido sea muy elevado. Este problema puede evitarse en cierta medida eligiendo una tasa de aprendizaje apropiado para el modelo en cuestión.

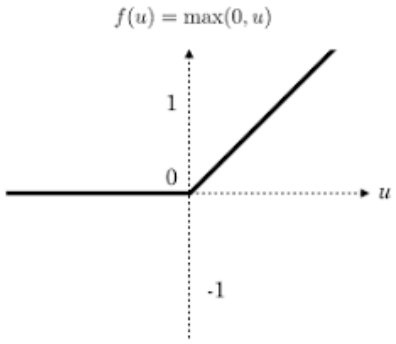


Figura 4-12. Función de activación ReLU [65]

Dicho esto, la arquitectura común de las redes neuronales convolucionales queda reflejada en la figura 4-13. En ella puede distinguirse una primera capa convolucional, a la que se le aplica la función de activación ReLU. A continuación, pasa por una capa pooling que reduce la dimensión de la salida y de nuevo se repite la misma operación. Esta secuencia puede repetirse en múltiples ocasiones, dependiendo de las dimensiones de la entrada y del problema que se esté tratando.

Una vez finaliza esta secuencia de capas convolucionales y pooling, se utilizan una capa fully connected para transformar la salida de esas capas en un conjunto de valores a los que se les pueda realizar una operación de clasificación. Tras esto, se utiliza la función Softmax al final de la red para clasificar finalmente la entrada entre sus posibles clases.

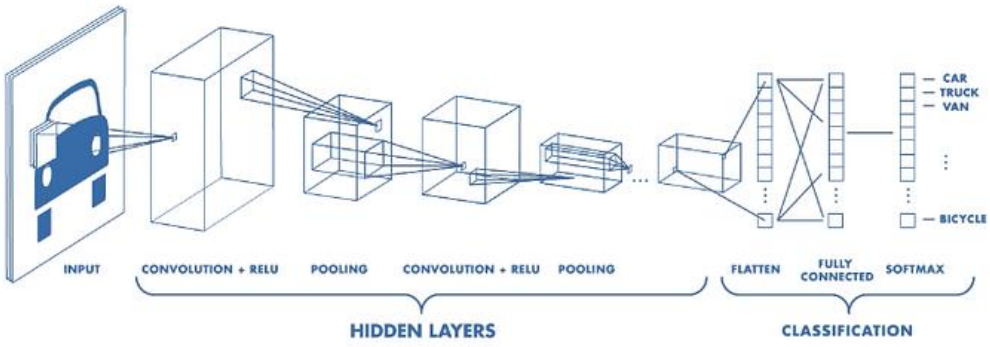


Figura 4-13. Estructura de una red neuronal convolucional [61]

4.4 Transfer Learning

Es de utilidad dedicar este apartado a explicar la técnica conocida como Transfer Learning. Esta es una técnica utilizada en Machine Learning y en Deep Learning en la que un modelo es entrenado en una tarea y su salida se utiliza como punto de partida de otro modelo, que realizará una tarea relacionada, pero distinta. Este enfoque es de utilidad cuando se necesitan grandes tiempos de cálculo y muchos recursos, pero que, utilizando modelos previamente entrenados como punto de partida, permite desarrollar con mayor facilidad modelos eficaces para resolver problemas complejos de Computer Vision o NLP.

En este trabajo esta técnica se utilizará haciendo uso de un modelo pre-entrenado para extraer las características necesarias del conjunto de datos inicial, usando la salida de este modelo pre-entrenado como entrada de otro modelo más sencillo, que llevará al resultado final.

De esta forma, se pueden reutilizar los conocimientos que ya ha adquirido una red que ha sido entrenada con millones de parámetros y obtener mejores resultados en un tiempo más corto. [66]

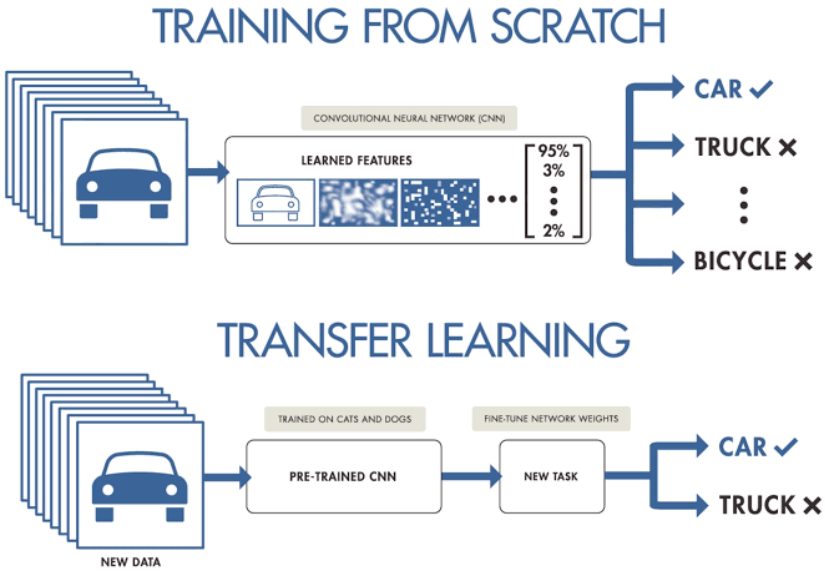


Figura 4-14. Entrenamiento desde cero VS Transfer Learning [67]

4.5 Modelos pre-entrenados

En este apartado se desarrollan distintos modelos pre-entrenados que serán de utilidad para este trabajo. Estos modelos pre-entrenados suponen la primera parte de la técnica de Transfer Learning que se ha explicado en el apartado anterior.

Este apartado se limita a tratar su arquitectura, así como algunas de sus características principales. Los resultados asociados a cada modelo se pueden comprobar en el capítulo 6 de esta memoria.

Estos modelos están disponibles en la librería 'timm', que es la que se ha utilizado en este trabajo.

4.5.1 Tf_efficientnet_b0

EfficientNet es una familia de modelos de redes neuronales convolucionales que tienen como objetivo el ser eficientes en precisión y tamaño [74]. Concretamente, el modelo tf_efficientnet_b0 el primero que se introdujo en la familia de EfficientNet, en el año 2019, y está entrenado con más de un millón de imágenes de la base de datos ImageNet [75].

Los modelos EfficientNet utilizan compound scaling, para mejorar la precisión y la eficiencia del modelo. Esta técnica permite regular de forma óptima la profundidad, el ancho y el tamaño de entrada de la red. Es decir, las dimensiones del modelo se escalan en función de las necesidades de este, con el objetivo de aportar elasticidad a la estructura del modelo, y, por lo tanto, más eficiencia. [76]

En cuanto a sus características generales, el modelo elegido ha sido entrenado con aproximadamente 5.3 millones de parámetros y realiza aproximadamente 0.39 GPLOPs. Estos modelos son comúnmente utilizados en Transfer Learning, cosa que se va a aplicar en este trabajo.

En cuanto a su arquitectura, sus capas principales son las capas MBConv (Mobile Inverted Bottleneck). Estas capas son una combinación de convoluciones depth-wise y bloques residuales. La convolución depth-wise se caracteriza por aplicarse de forma independiente a cada canal de entrada individual, en lugar de operar simultáneamente sobre todos los canales de entrada como ocurre en las convoluciones estándar. Por su parte, los bloques residuales permiten la propagación del gradiente sin perder información y contienen conexiones de salto o residuales, que permiten el intercambio de información entre neuronas que no están en capas directamente conectadas, ayudando también a evitar el desvanecimiento de gradiente. [77]

El tamaño de la convolución que realiza cada una de las capas varía, como se puede ver en la figura a continuación, utilizando algunas un kernel de 3x3 y otras de 5x5.

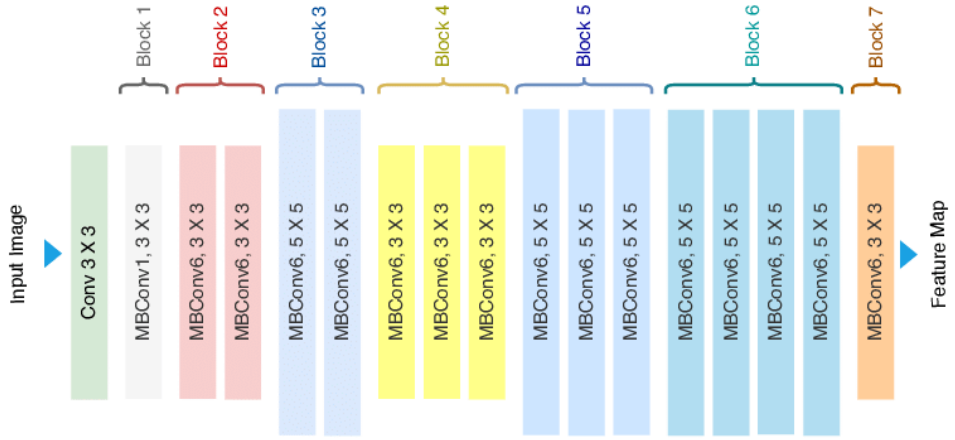


Figura 4-15. Arquitectura de bloques y capas de EfficientNet [77]

Además de las capas MBConv, EfficientNet incluye el bloque SE (Squeeze and excitation), que ayuda a que el modelo se centre en las características más importantes de los datos de entrada, suprimiendo las menos importantes.

4.5.2 Resnet50

Otro de los modelos es '**resnet50'**. Las diferencias con el modelo 'tf_efficient_b0' son notables, especialmente en términos de arquitectura y de eficiencia computacional. 'Resnet50' utiliza bloques residuales que permiten la propagación del gradiente sin perder información y que contienen conexiones de salto o residuales, que permiten el intercambio de información entre neuronas que no están en capas directamente conectadas, ayudando también a evitar el desvanecimiento de gradiente. [83]

En concreto, 'resnet50' tiene una arquitectura de 50 capas. En la primera capa se realizan convoluciones 7x7 y en el resto de capas, llamadas bloques residuales, se realizan convoluciones 3x3. Como es común en las redes neuronales convolucionales, aparecen capas de Pooling y una capa completamente conectada al final. [84] [85]

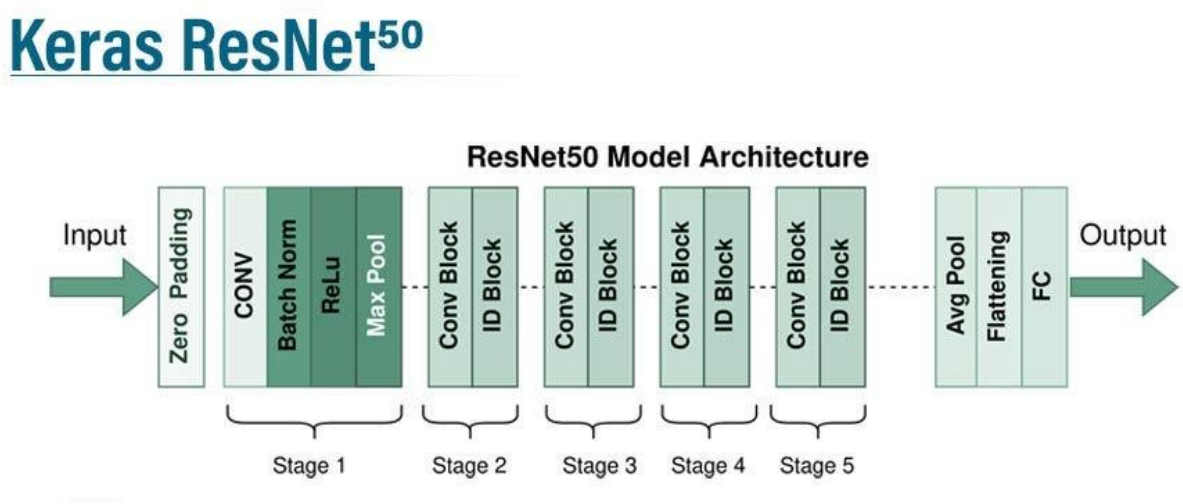


Figura 4-16. Arquitectura de resnet50 [84]

Este modelo es especialmente bueno en el entrenamiento de redes con muchas capas, pero no es tan eficiente comparado con otros modelos en el uso de los parámetros y FLOPs.

4.5.3 ViT Base Patch16 224

Otro modelo que se ha probado es el 'ViT Base Patch16 224'. Este modelo pertenece a la familia ViT (Vision Transformer). Estos modelos se utilizan principalmente para tareas de procesamiento de lenguaje natural, aunque también en redes neuronales convolucionales para el procesamiento de imágenes, como es el caso. [86]

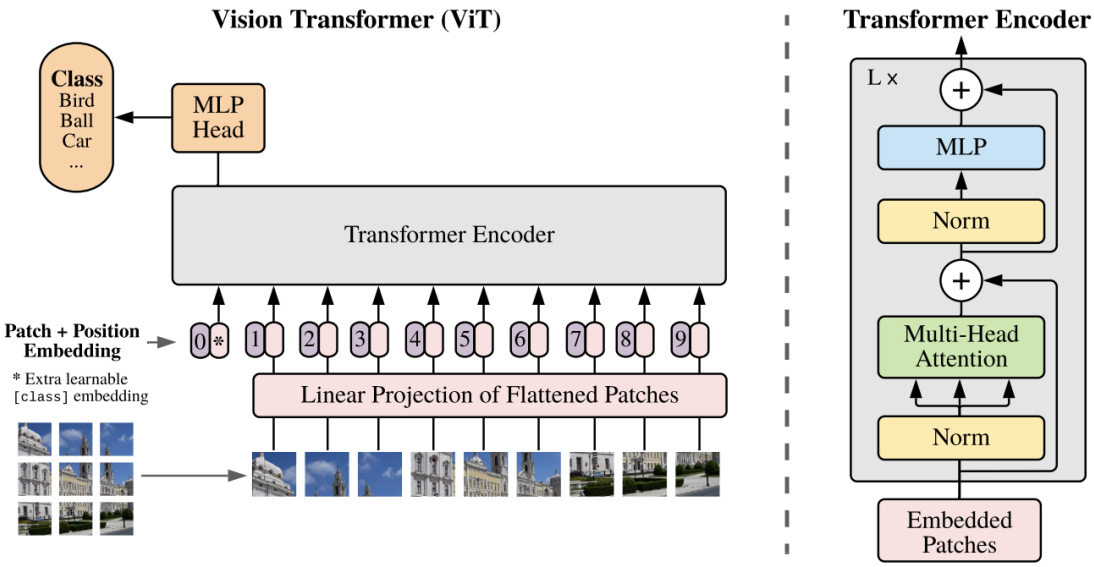


Figura 4-17. Ejemplo de un ViT (Vision Transformer) [87]

4.5.4 DPN68

El modelo 'dpn68' utiliza una arquitectura Dual Path. Esta arquitectura combina características de ResNet, que ha sido vista anteriormente, y de DenseNet, tratando de aprovechar las ventajas que ofrecen ambas. [88]

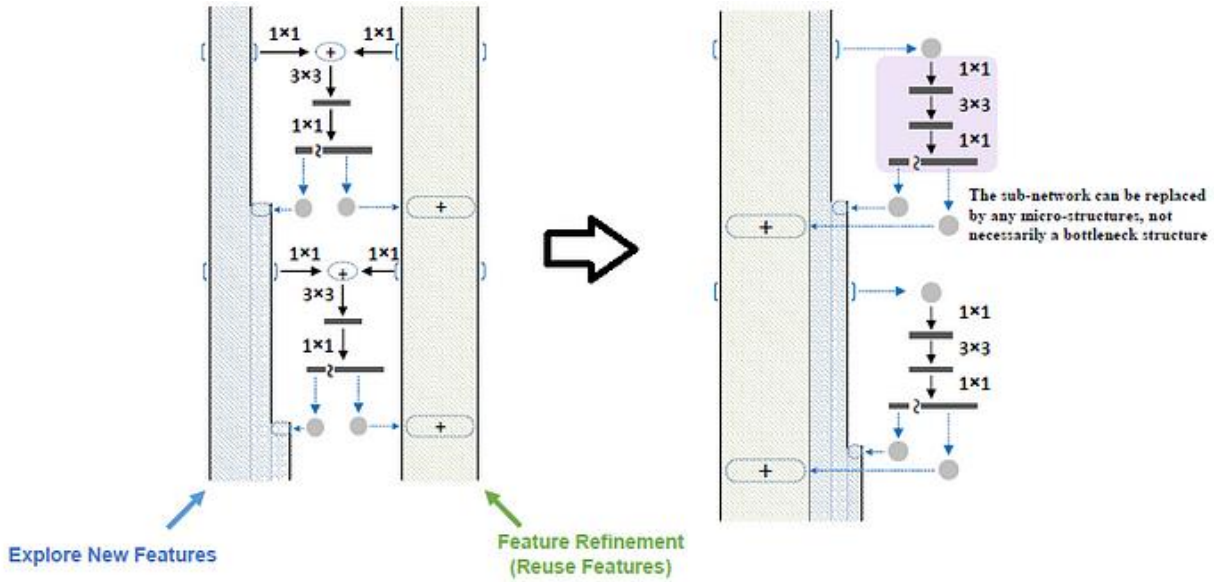


Figura 4-18. Ejemplo de DPN como combinación de ResNet y DenseNet

5 DETECCIÓN DE EVENTOS EN PARTIDOS DE FÚTBOL

En los anteriores capítulos de este trabajo se han desarrollado en profundidad los fundamentos y técnicas de la Inteligencia Artificial y el aprendizaje automático, con un enfoque especial en las redes neuronales. Se ha partido desde los principios básicos, pasando por los aspectos éticos y sus aplicaciones hasta llegar a este punto, donde el objetivo es utilizar lo que se ha explorado con el fin de aplicarlo a la detección de eventos en los partidos de fútbol.

El análisis automático de partidos de fútbol, así como ocurre de la misma forma en otros deportes, se ha convertido en un factor muy a tener en cuenta en los últimos años. Son muchas las personas implicadas en el análisis de estos datos, desde entrenadores o narradores hasta los aficionados.

En este capítulo se explican las tecnologías que se han utilizado para cumplir con este cometido, el conjunto de datos que se utiliza, el modelo de red que se ha diseñado y algunas alternativas que son igual de válidas.

5.1 Conjunto de datos

Este trabajo se ha basado en una competición publicada por Kaggle [68], llamada Bundesliga Data Shootout. El objetivo de esta competición es diseñar un modelo de visión por computador que sea capaz de detectar y clasificar pases en vídeos de partidos de fútbol, diferenciando entre pases estándar entre jugadores, saques de banda y centros.

La organización que plantea esta competición es la Bundesliga, liga de fútbol profesional de Alemania, que comenta que, en la actualidad, la mayoría de estos datos los toman personas de forma manual, involucrando a muchos trabajadores en el proceso. Es por eso por lo que estos datos se toman únicamente en las ligas de fútbol profesional. Por lo tanto, el objetivo es que con esta solución tecnológica los datos se tomen de forma más rápida y con una mayor profundidad, pudiendo aplicarse a un mayor rango de competiciones.

Dicho esto, el conjunto de datos con el se trabaja es con el que el organizador de la competición proporciona en Kaggle. El conjunto de datos consta de 246 ficheros, siendo la mayor parte de estos archivos de vídeo en formato MP4. Además de estos vídeos, se incluye un fichero de tipo csv que se describe a continuación. En total, le conjunto de datos tiene un tamaño de 37.55 GBytes.

El conjunto de datos contiene grabaciones de nueve partidos de fútbol divididos en partes y se divide de la siguiente forma:

- Directorio ‘train’: Esta carpeta contiene las grabaciones que se usarán como datos de entrenamiento, incluyendo los vídeos pertenecientes a ocho de los nueve partidos mencionados anteriormente. En cuatro de estos ocho partidos se incluyen ambas partes del partido, mientras que en los otros cuatro partidos se incluye únicamente una de las partes.
- Directorio ‘test’: Esta carpeta contiene las grabaciones que se usarán como datos de prueba. Este conjunto de pruebas incluye un partido completo y una parte de otros cuatro partidos.
- Directorio ‘clips’: Esta carpeta incluye vídeos cortos de diez partidos adicionales. En estos vídeos no se incluyen anotaciones de los eventos que tienen lugar, por lo que se utilizan para que el modelo sea capaz de generalizar para aplicarse en entornos y situaciones que no aparecen en el conjunto de datos de entrenamiento.
- Fichero ‘train.csv’: Este fichero incluye las anotaciones de los eventos que transcurren en los vídeos que contiene el directorio ‘train’. El formato de los datos incluye:
    - video_id: Identifica el vídeo en el que transcurre el evento.
    - event: El tipo de evento que tiene lugar: challenge, play o throwin. El evento ‘challenge’ se

refiere a aquel que ocurre cuando el balón se encuentra en una disputa entre jugadores de distintos equipos. Por su parte, el evento 'play' hace referencia a los pases que tienen lugar durante el transcurso normal del juego. Por último, el evento 'throwin' indica aquellos pases que son efectuados a través de un saque de banda.

- event_attributes: Atributos adicionales del evento que aportan información.
- time: El tiempo, en segundos, en el que transcurre el evento en el vídeo.

Un ejemplo de cómo se presentan los datos en el fichero train.csv es el siguiente:

```
video_id,time,event,event_attributes
1606b0e6_0,200.2658219228331,start,
1606b0e6_0,201.15,challenge,['ball_action_forced']
1606b0e6_0,202.7658219228331,end,
```

Figura 5-1. Ejemplo de evento en el fichero 'train.csv'

El identificador del vídeo es el "1606b0e6_0", el evento comienza en el segundo 200.2658219… y finaliza en el 202.7658219… y es un evento de tipo "challenge". Además, como atributo adicional se añade "ball_action_forced".

5.2 Desarrollo del modelo

A continuación, se procede con el desarrollo del modelo de red neuronal convolucional que va a ser entrenado con los vídeos del directorio train y posteriormente evaluado con los vídeos del directorio test.

5.2.1 Entorno de ejecución

Como entorno de ejecución se ha decidido usar Kaggle. La idea original era utilizar el entorno que proporciona Google Colab, pero debido al alto peso de los vídeos y a la necesidad de computación al procesarlos, no era suficiente con la memoria RAM de 12 GB que ofrece esta plataforma.

La plataforma Kaggle ofrece 30 GB de memoria RAM, además de la posibilidad de utilizar GPUs y TPUs, que son esenciales para entrenar modelos de Deep Learning. Por supuesto, además de estas características mencionadas, el entorno de Kaggle tiene preinstaladas muchas bibliotecas muy utilizadas en Python para Machine Learning, como TensorFlow, Keras, pandas o Pytorch entre otras.

Figura 5-2. Plataforma Kaggle

5.2.2 Preparación de los datos

Antes de entrar a detallar el modelo que se va a entrenar, es de vital importancia preparar los datos de forma correcta y óptima para ello.

En este caso, preparar los datos para el entrenamiento no es algo sencillo, pues se dispone de múltiples vídeos de larga duración. Será de interés dividir estos vídeos en fotogramas y rescatar esas imágenes en la que tengan lugar los eventos que se quieren clasificar (challenge, play, throwin).

Para aclarar gráficamente en qué consisten los eventos que se está buscando detectar, se exponen varias imágenes que señalan dichos eventos.

En primer lugar, el evento '**play'**, como se ha indicado en el apartado anterior, es un pase que tiene lugar durante el curso natural del juego, entre jugadores del mismo equipo. En la siguiente imagen puede apreciarse como el jugador del equipo que viste de blanco tiene el balón en su poder y va a realizar un pase hacia un compañero.

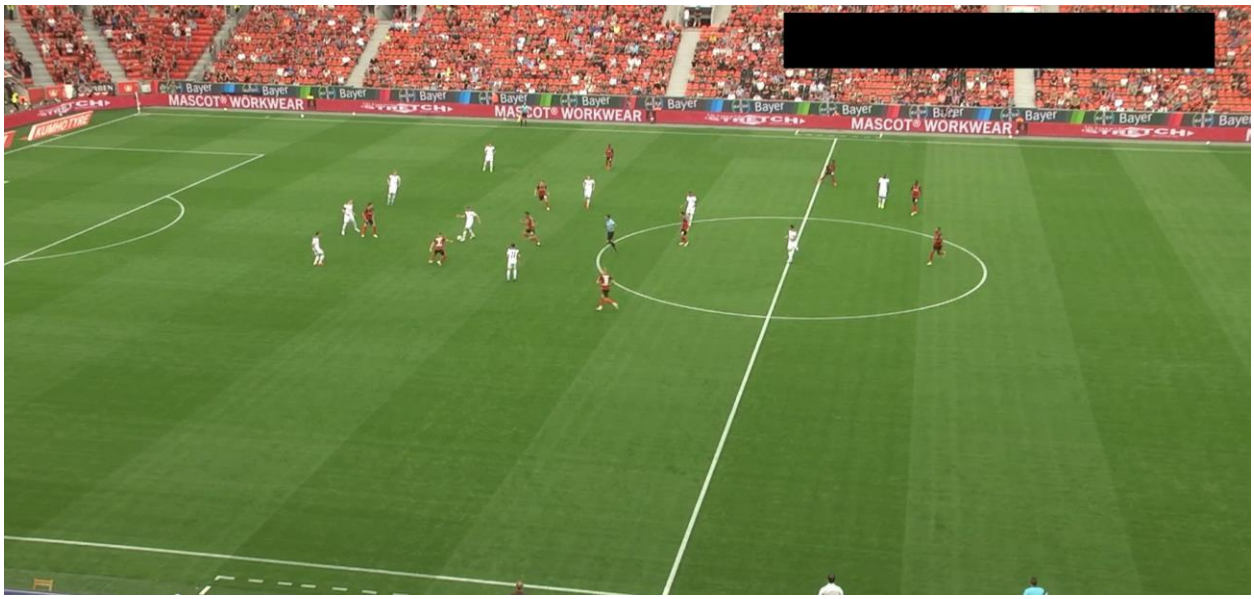


Figura 5-3. Inicio del pase, evento 'play'

La pelota viaja hacia su compañero, que la recibe:

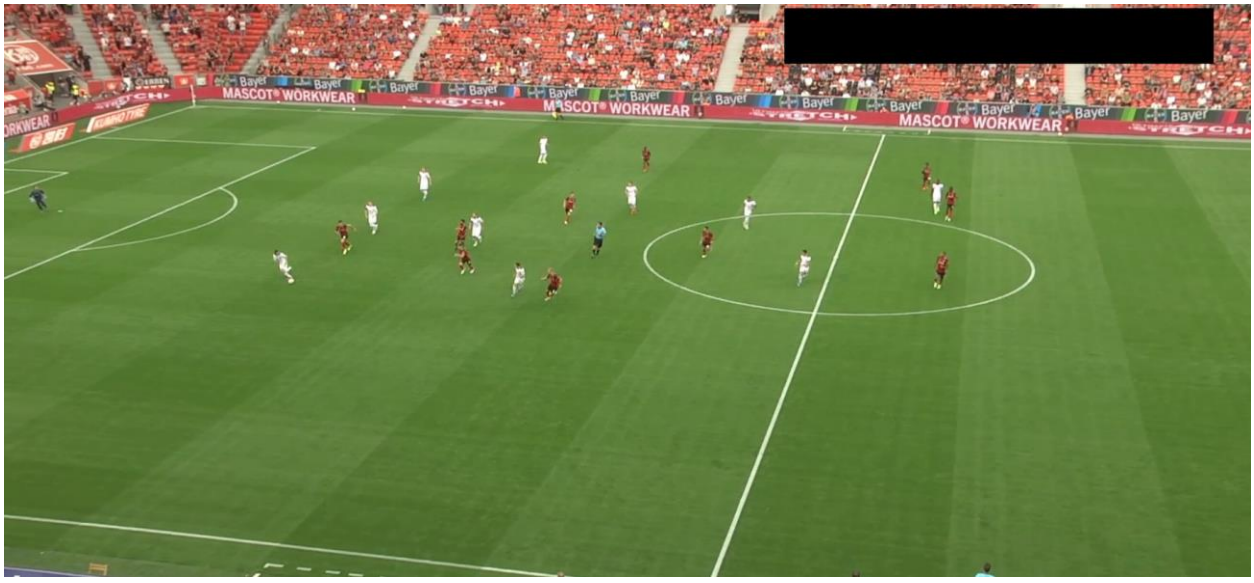


Figura 5-4. Fin del pase, evento 'play'

Esto se indica en el fichero 'train.csv' de la siguiente manera, señalando el inicio y el final del evento 'play':

```
ecf251d4_0,270.27002018776284,start,
ecf251d4_0,271.187,play,"['pass', 'openplay']"
ecf251d4_0,272.77002018776284,end,
```

Figura 5-5. Inicio y final del evento 'play'

El siguiente evento es '**throwin'**, que comprende todos los pases procedentes de saques de banda. En la siguiente imagen puede verse a un jugador preparado para sacar de banda:

Figura 5-6. Inicio del evento 'throwin'

La pelota viaja hasta su compañero, que la recibe:

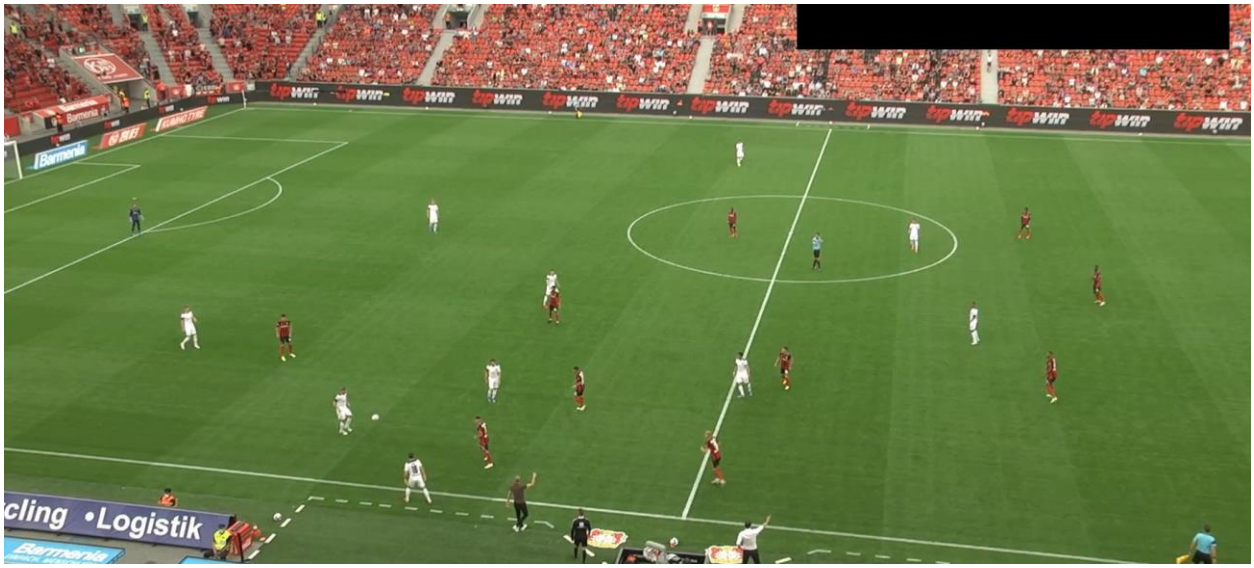


Figura 5-7. Fin del evento 'throwin'

En el fichero ‘train.csv’ esto se indica de la siguiente manera:

ecf251d4_0,2215.282975246609,start,
ecf251d4_0,2216.147,throwin,['pass']
ecf251d4_0,2219.067,play,"['pass', 'openplay']"
ecf251d4_0,2219.926557438906,end,

Figura 5-8. Inicio y final del evento ‘throwin’

Por último, se va a ejemplificar el evento ‘challenge’, que consiste en una disputa entre jugadores de distintos equipos, que puede acabar con un cambio de posesión. Un ejemplo de esto puede verse en las siguientes figuras.

En la primera imagen, dos jugadores disputan el balón en el centro del campo:

Figura 5-9. Inicio del evento ‘challenge’

En este caso, el jugador del equipo que viste de rojo ha conseguido mantener la posesión del balón después de la disputa:

Figura 5-10. Fin del evento ‘challenge’

Además, habrá que procesar los datos del fichero train.csv, y definir los hiperparámetros que se van a utilizar en el entrenamiento. Para le explicación de esta parte me he basado en la solución desarrollada por un usuario en Kaggle, que puede consultarse en esta referencia [69]. Todo este proceso se detalla a continuación:

5.2.2.1 Procesamiento del fichero 'train.csv'

Para procesar los datos del fichero 'train.csv' se utiliza la librería de Python 'pandas' [70]. Pandas es una librería que se utiliza para el análisis y manipulación de datos y que permite leer y escribir en distintos tipos de formatos como CSV, JSON o SQL.

Antes de explicar la modificación que se ha llevado a cabo en los datos del fichero 'train.csv' conviene entrar en otro detalle, que no es otro que la definición de tolerancias para cada uno de los eventos.

En el fichero CSV se especifica el instante en el que comienza y en que termina cada uno de los eventos que tienen lugar en los vídeos, si bien es cierto que, en un caso real, un pase de un jugador a otro es algo que no ocurre de manera instantánea en todas las ocasiones, por lo que es conveniente definir valores de tolerancia para ajustar el rango de tiempo en el que puede ocurrir el evento. En ciertos pases, esta tolerancia no es necesaria, pero en otras puede hacer que el modelo adquiera una mejor comprensión del evento al haberle proporcionado un mejor contexto temporal.

Por este motivo, se definen cinco valores de tolerancia, medida en segundos, para cada uno de los eventos, que aportan una mayor flexibilidad al modelo y que se compararán en el apartado de resultados para determinar qué efecto tienen.

De esta manera, haciendo uso de la tolerancia, se han creado dos eventos adicionales que se añaden a cada una de las entradas del fichero CSV, 'pre_' y 'post_', los cuales se han calculado haciendo uso de la tolerancia definida. Estos eventos ajustan el margen alrededor del pase, lo cual permite al modelo concretar el momento exacto en el que tiene que "prestar atención" a lo que está ocurriendo en el vídeo.

Para que se entiendan mejor los eventos que se han añadido es mejor analizar un caso real. Tomemos el ejemplo de un saque de banda de los muchos que están señalados en el fichero 'train.csv', nombrado como 'throwin'. En concreto, para este ejemplo se utilizará el último saque de banda correspondiente al vídeo con id 'ecf251d4_0', el cual, en el fichero 'train.csv' original, comienza en el instante 2215.28 segundos (36 minutos y 55 segundos aproximadamente), tiene lugar en el instante 2216.147 segundos y finaliza en el instante 2219.92 segundos. A continuación, se puede ver como se presentan estos datos en el fichero 'train.csv' original.

```
ecf251d4_0,2215.282975246609,start,
ecf251d4_0,2216.147,throwin,['pass']
ecf251d4_0,2219.067,play,"['pass', 'openplay']"
ecf251d4_0,2219.926557438906,end,
```

Figura 5-11. Datos correspondientes a un ejemplo de evento 'throwin'

Para comprobar que en efecto se trata de un saque de banda, se puede acudir al vídeo en cuestión y a los 36 minutos y 55 segundos, esta es la imagen que aparece en pantalla, en la que se puede ver a un jugador preparado para sacar de banda:

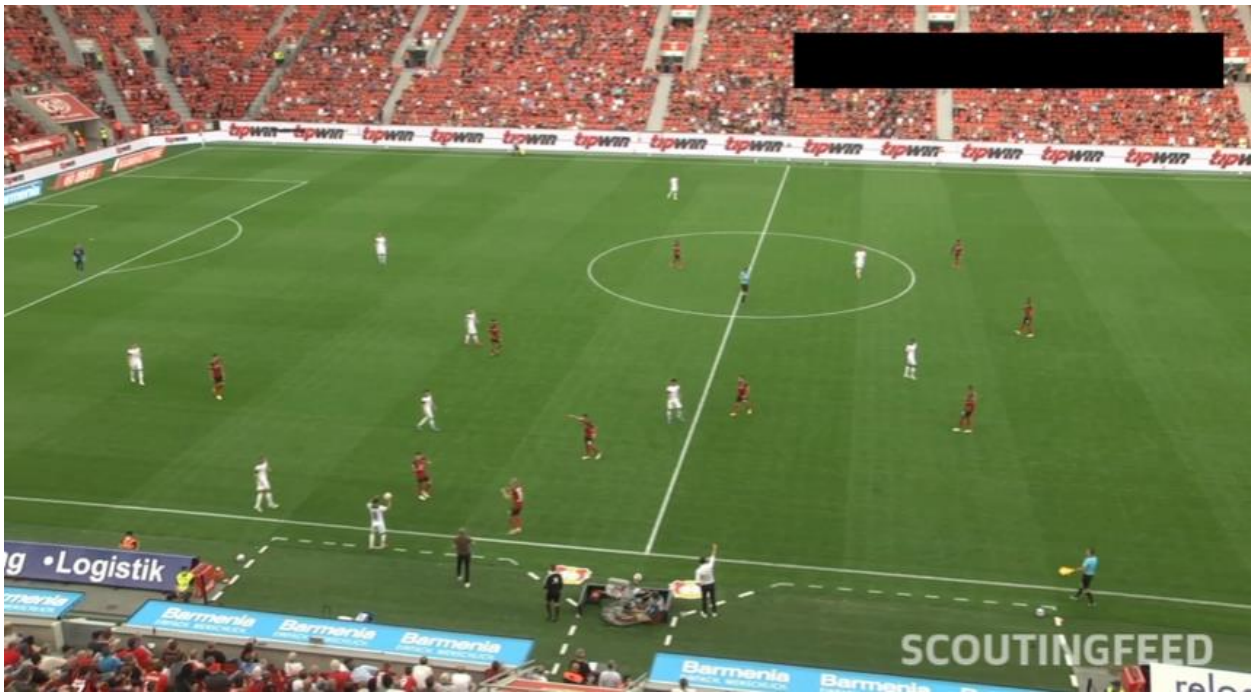


Figura 5-12. Jugador sacando de banda

Dicho esto, la idea de crear estos nuevos eventos es, como se ha indicado previamente, ajustar el momento en el que ocurre uno de los eventos, en este caso, el saque de banda.

Para realizar este ajuste se realiza un cálculo a partir del valor de tolerancia elegido. Para los eventos start_ y end_, este cálculo consiste en dividir entre dos el valor de tolerancia escogido y desplazar el tiempo original de inicio del evento. Es decir, si originalmente el evento tenía lugar en el instante 2216.147 y el valor de tolerancia elegido es 0.20 segundos, los eventos start_ y end_ estarán desplazados 0.10 segundos antes y después de ese momento en el que se marca que tiene lugar el evento en el fichero original.

Por otro lado, para los eventos pre_ y post_, se hace el mismo cálculo que para los eventos start_ y end_, solo que el desplazamiento sobre el instante original en el que tenía lugar el evento es el doble. Es decir, si el evento comenzaba en el instante 2216.147 y de nuevo el valor de tolerancia elegido es de 0.20 segundos, los eventos pre_ y post_ estarán desplazados 0.20 segundos antes y después de ese momento en el que transcurría el evento según el fichero 'train.csv' original.

Le elección de los valores de las tolerancias se desarrolla en el siguiente capítulo, comparando los resultados para distintas tolerancias.

De esta forma, para el nuevo fichero 'train.csv', este mismo saque de banda queda definido por los siguientes eventos, haciendo uso de una tolerancia de 0.20 segundos:

- pre_throwin: Que ocurre en el instante 2215.947 segundos, casi un segundo posterior al evento 'start' que se tenía originalmente.
- start_throwin: Que tiene lugar en el instante 2216.047 segundos y marca el momento en el que comienza el movimiento del saque de banda.
- end_throwin: Marcado en el instante 2216.247 segundos y que indica el final del movimiento del saque de banda.
- post_throwin: Que tiene lugar en el instante 2216.347 segundos, muy cercano al final del saque de banda, marcando correctamente el instante "exacto" en el que tiene lugar el pase.

Estos ajustes ejemplifican el porqué de añadir estos nuevos eventos. En el fichero 'train.csv' original, el inicio y el final de muchos eventos estaban marcados con una distancia temporal muy amplia con respecto al instante en el que tenía lugar el evento. Por lo tanto, el objetivo de este ajuste es que el marco temporal en el que ocurre el evento sea más concreto, haciendo que los eventos pre_ y post_ sean más cercanos al momento en el que tiene

lugar. Por otro lado, los eventos start_ y pre_ hacen que el instante en el que ocurre el evento no esté marcado únicamente por un fotograma, si no que exista un pequeño margen anterior y posterior que ayude al modelo.

Esto concluye en que el único instante temporal que se ha considerado de interés del conjunto de datos inicial es el instante en el que ocurre el evento. Por lo tanto, se han ignorado los tiempos correspondientes a las etiquetas 'start' y 'end' del fichero 'train.csv' original, debido a que, como se ha podido ver, en muchos de los casos, estaban muy distanciadas del momento en el que realmente ocurría el evento.

Estos instantes son directamente mapeados con sus correspondientes fotogramas (frames), en los que se puede comprobar esto que se acaba de explicar.

En la siguiente figura, correspondiente al evento 'pre_throwin' se puede ver como el jugador tiene el balón en las manos y está preparándose para sacar de banda:

Figura 5-13. Fotograma correspondiente al evento 'pre_throwin'

La siguiente imagen es muy similar a la anterior, pues hay una diferencia de 0.1 segundos entre una y otra.

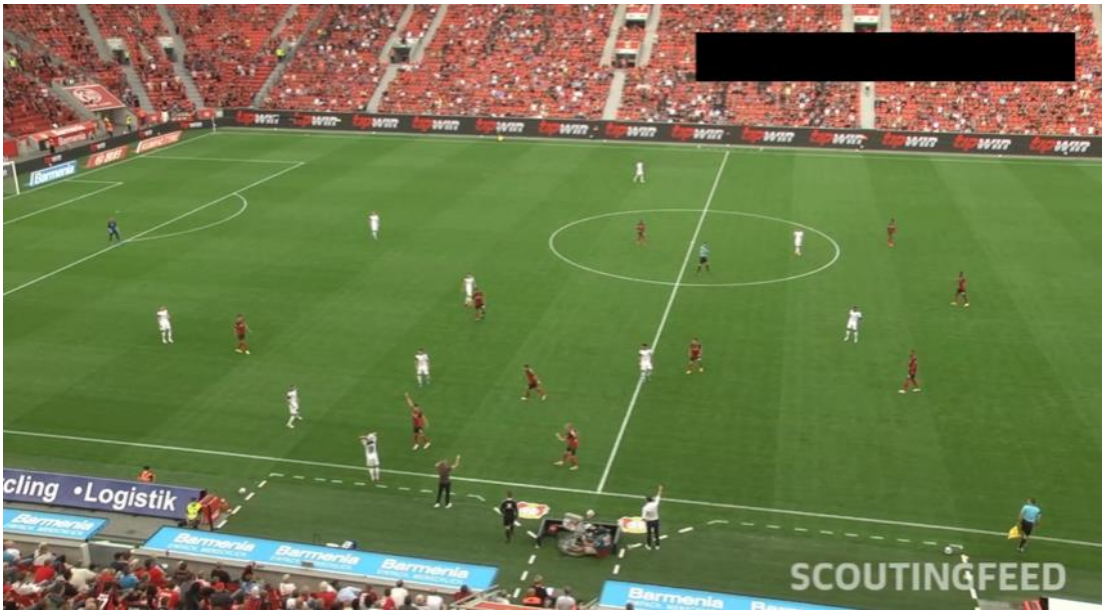


Figura 5-14. Fotograma correspondiente al evento 'start_throwin'

En la siguiente imagen, el jugador ya ha sacado de banda, por lo que corresponde al evento ‘end_throwin’.

Figura 5-15. Fotograma correspondiente al evento ‘end_throwin’

Por último, se puede ver el fotograma correspondiente al evento ‘post_throwin’. En la imagen, la pelota ha avanzado con respecto al anterior evento. Ambas imágenes son muy similares, pues de nuevo se diferencian en 0.1 segundos.

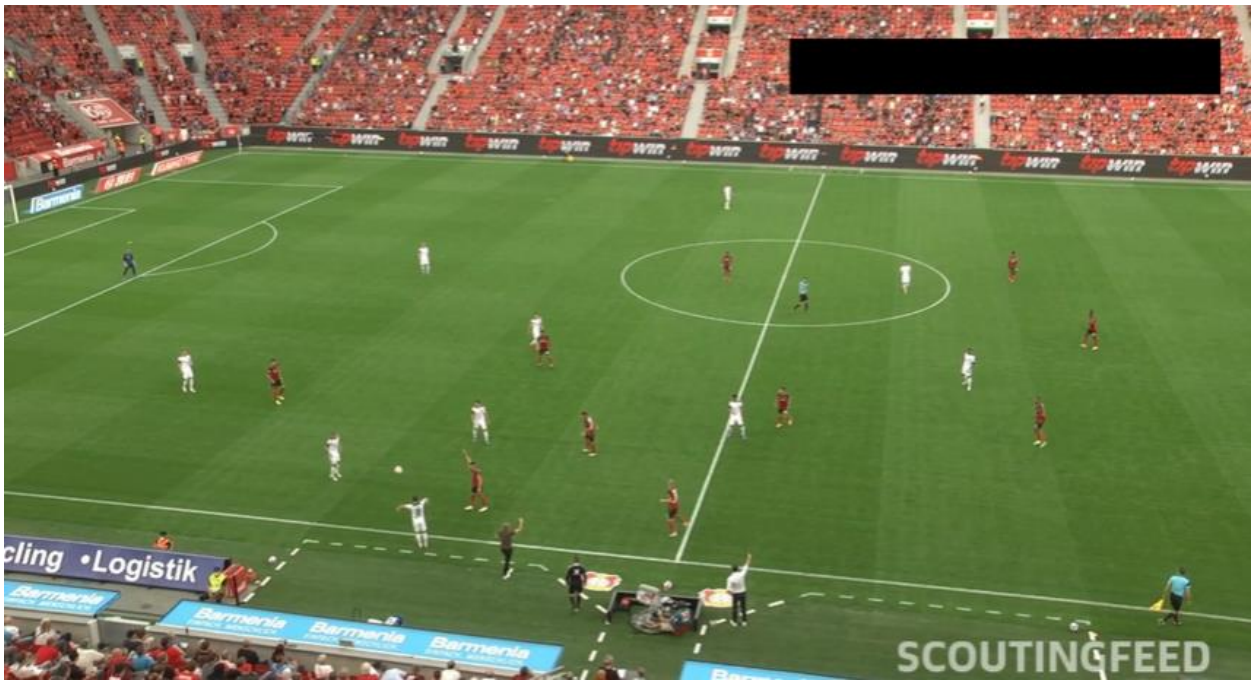


Figura 5-16. Fotograma correspondiente al evento ‘post_throwin’

Tras eliminar las entradas sobrantes (los eventos ‘start’ y ‘end’ que estaban en el fichero ‘train.csv’ original), el fichero ‘train.csv’ queda como en la figura a continuación, en la que se puede ver que cada pase tiene un evento ‘pre_’, ‘start_’, ‘end_’ y ‘post_’.

	video_id	time	event	event_attributes	frame
0	1606b0e6_0	200.265822	start	NaN	5006.645548
8764	1606b0e6_0	200.750000	pre_challenge	['ball_action_forced']	5018.750000
0	1606b0e6_0	200.950000	start_challenge	['ball_action_forced']	5023.750000
1	1606b0e6_0	201.350000	end_challenge	['ball_action_forced']	5033.750000
8765	1606b0e6_0	201.550000	post_challenge	['ball_action_forced']	5038.750000
...	...	...	...	...	...
17526	ecf251d4_0	3069.347000	pre_throwin	['pass']	76733.675000
8762	ecf251d4_0	3069.447000	start_throwin	['pass']	76736.175000
8763	ecf251d4_0	3069.647000	end_throwin	['pass']	76741.175000
17527	ecf251d4_0	3069.747000	post_throwin	['pass']	76743.675000
11217	ecf251d4_0	3070.780519	end	NaN	76769.512977

Figura 5-17. Aplicación de las tolerancias al fichero 'train.csv'

5.2.2.2 Procesamiento de video

Para procesar los vídeos se hace uso de la librería OpenCV (Open Source Computer Vision). Esta es una librería de código abierto que se utiliza para técnicas de visión artificial (computer vision) y que ofrece diversas funcionalidades para el procesamiento de imágenes y vídeos como, por ejemplo, la lectura y escritura de imágenes en distintos formatos, aplicación de filtros o el escalado y rotación de estas [71].

Figura 5-18. Librería OpenCV

En primer lugar, se va a obtener la tasa de fotogramas por segundo (fps) de los vídeos, lo cual es un valor de interés pues los eventos en el fichero CSV están señalados en segundos.

Haciendo uso de las funciones que ofrece OpenCV se obtiene que los vídeos tienen 25 fps. Con este valor se convierten los tiempos de los eventos, que estaban en segundos, a número de fotogramas, simplemente multiplicando el tiempo por la tasa de fotogramas por segundo. Con este resultado, se añade también al fichero 'train.csv' una columna con el fotograma en el que tiene lugar cada uno de los eventos.

A continuación, se hace una operación bastante común a la hora de trabajar con vídeos en redes neuronales, que es extraer los fotogramas del vídeo uno a uno y ajustando cada uno de ellos a un tamaño óptimo. En este caso se ha decidido ajustar los fotogramas, de un tamaño original 1920x1080, a un tamaño de 540x768, el cual es un tamaño lo suficientemente grande para capturar detalles relevantes y patrones en las imágenes, pero lo suficientemente pequeño como para poder manejarse sin ocupar demasiada memoria ni necesitar una carga computacional demasiado elevada. Además, para el modelo pre-entrenado que se utilizará más adelante, el tamaño de entrada esperado de las imágenes es de 540x768, por lo que se ha decidido ajustarse a dicho modelo. Para el redimensionamiento se utiliza una interpolación de área, que es la que recomienda OpenCV para reducir el tamaño de las imágenes, que consiste en un muestreo de la imagen usando una relación del área de cada píxel [72].

Con este paso se habrá conseguido transformar los vídeos en conjuntos de imágenes, de forma que se pueden utilizar como datos de entrada para entrenar y probar al modelo.

Aún queda un paso más antes de llegar al entrenamiento del modelo, que consiste en identificar los eventos que tienen lugar en cada uno de los vídeos y asociarlos a los fotogramas en los que tienen lugar.

Este análisis se hace vídeo a vídeo, haciendo uso de las columnas 'frame' y 'event' del fichero CSV y organizando los eventos, dividiéndolos por su inicio, su transcurso y su final.

Una vez finalizada esta tarea, se pasa a preparar el conjunto de datos de entrada y las etiquetas que se van a pasar al modelo.

5.2.2.3 Transformación de los datos para el modelo

Llegados a este punto, he llegado el momento de crear el modelo. El modelo que se va a utilizar es un modelo pre-entrenado de la librería 'timm'.

La librería 'timm' es una librería de Python que proporciona una colección de modelos pre-entrenados (Pytorch Image Models), junto con otras funciones para el procesamiento de imágenes y el entrenamiento de dichos modelos [73].

Se utilizarán distintos modelos pre-entrenados, los cuales se explican en el capítulo 4. Concretamente se utilizarán los modelos: tf_efficientnet_b0, resnet50, ViT Base Patch16 224 y dpn68, que se compararán en el siguiente capítulo para elegir el óptimo.

Una vez definido el modelo hay que tener en cuenta que, a la hora de asignar las etiquetas, existen 10 salidas posibles. Esto es debido a que no solo se tienen los eventos play, throwin y challenge originales, sino que se han añadido los eventos pre_ y post_ a cada uno de ellos. Por lo tanto, se usa una etiqueta distinta para cada uno de ellos, quedando, por lo tanto, las siguientes etiquetas:

- Background: Etiqueta asignada para indicar que no está ocurriendo ninguno de los eventos, simplemente está el juego en curso.
- Pre_challenge: Etiqueta destinada a señalar el inicio del evento 'challenge'.
- Challenge: Etiqueta asignada a una disputa de balón entre jugadores de distinto equipo.
- Post_challenge: Etiqueta destinada a marcar el final del evento 'challenge'.
- Pre_throwin: Etiqueta destinada a señalar el inicio del evento 'throwin'.
- Throwin: Etiqueta asignada a un pase que tiene lugar en un saque de banda.
- Post_challenge: Etiqueta destinada a marcar el final del evento 'throwin'.
- Pre_play: Etiqueta destinada a señalar el inicio del evento 'play'.
- Play: Etiqueta asignada a un pase que tiene lugar durante el transcurso natural del juego.
- Post_play: Etiqueta destinada a marcar el final del evento 'play'.

Las etiquetas start_ y end_ de cada uno de los eventos se utilizan para definir los intervalos de tiempo de los eventos y así poder asociarlos a los fotogramas en los que estos tienen lugar, mientras que estas etiquetas que se acaban de describir son las que se utilizarán para la clasificación directa de los eventos. Es por eso que en esta lista no aparecen las etiquetas start_ y end_, que ya han cumplido previamente su cometido en la obtención de los fotogramas. Esto provoca un nuevo cambio en el fichero 'train.csv', ahora en su estado definitivo, con las etiquetas mencionadas.

A la hora de cargar las imágenes que se han obtenido en los pasos anteriores se ha decidido cargar tres imágenes por cada etiqueta, una correspondiente al fotograma actual, otra al fotograma anterior y otra al posterior. Estas imágenes se cargan directamente en escala de grises y se normalizan dividiéndolas entre 255. De estas muestras se obtiene la etiqueta correspondiente, mapeándola con el evento que tienen asociado. Esto se hace para reducir ambigüedades y conseguir capturar el contexto temporal. Siendo más concreto, hay ciertos eventos que, al

mapear el instante temporal en el que ocurren con el fotograma correspondiente, el número del fotograma no es exacto, por lo que, para no perder información, también se utilizan el fotograma anterior y el siguiente. Además, es una práctica común en casos en los que se utilizan las redes neuronales en vídeos, pues puede que en alguno de los fotogramas la imagen sea borrosa y el modelo tenga dificultades para detectar lo que está ocurriendo.

5.2.2.4 Entrenamiento del modelo

Para la fase de entrenamiento se ha decidido hacer dos fases diferenciadas, aplicando Transfer Learning, método que se explica en el capítulo 4 del trabajo. En primer lugar, se hará un entrenamiento de 30 épocas, haciendo un recorrido completo por todo el conjunto de datos. En el capítulo siguiente, se muestran resultados de entrenamientos realizados para distinto número de épocas y se comparan los resultados. En segundo lugar, se usarán las salidas de este primer entrenamiento para hacer un nuevo entrenamiento, ahora con una red neuronal convolucional 1D. Para estos pasos es importante utilizar la GPU que ofrece la plataforma que se esté utilizando, en este caso Kaggle.

En este primer entrenamiento se utiliza el modelo pre-entrenado que se ha mencionado en el apartado anterior, por lo que solo se necesita especificar algunos parámetros del modelo, sin haber necesidad de crear una red completa desde cero. Por lo tanto, en la definición del bucle de entrenamiento solo hay que tener en cuenta algunos pasos fundamentales como reiniciar los gradientes (poner a cero sus valores) y elegir la función de pérdidas que se va a utilizar que, en este caso, será la de entropía cruzada (cross entropy), que como se explicó en un capítulo anterior, es comúnmente utilizada para problemas de clasificación. También se elige el optimizador, en este caso Adam, que es una variante del descenso por gradiente. Tras esto, simplemente hay que realizar las primeras predicciones y seguidamente la propagación hacia atrás (backward) para calcular los valores de los gradientes y actualizar los pesos del modelo.

En el segundo entrenamiento, se crea una red neuronal convolucional en la que se distinguen tres tipos de capas:

- Capas convolucionales: Como se explicó en el capítulo anterior, se encargan de la extracción de características de la imagen.
- Capas de normalización: Su objetivo es normalizar la salida de las capas convolucionales, con el fin de prepararlas para ser la entrada de la siguiente capa.
- Capas de Dropout: Se utilizan para desactivar neuronas durante el entrenamiento, lo que sirve para prevenir el sobreajuste y mejorar la generalización del modelo. Su funcionamiento se basa en que desactivando algunas neuronas de forma aleatoria se consigue disminuir la dependencia entre ellas y que el funcionamiento de la red no depende de algunas neuronas concretas.

Se utilizará la función de activación ReLU después de cada capa de convolución y normalización.

Cuatro de los vídeos del conjunto de datos de entrada se reservan para la validación del modelo.

5.2.2.5 Cálculo de la puntuación

Como es obvio, para saber cuan bueno es el modelo que se ha entrenado es necesario medir su precisión. En este caso, la precisión se va a medir acorde con los criterios que se estipulan en las reglas de la competición de Kaggle.

Para ello, se han de filtrar las detecciones de eventos que no estén dentro de los intervalos de puntuación. Esto se consigue hacer comprobando el tiempo en el que se produce cada detección y comparándolo con el tiempo del intervalo de puntuación. Si la detección ocurre antes del intervalo de puntuación, esta se descarta y se pasa a la siguiente detección. Si la detección está dentro del intervalo, se conserva y se pasa a la siguiente detección. Si la detección ocurre después del intervalo de puntuación, se ha de pasar a comprobar el siguiente intervalo de puntuación. Tras esto, se emparejan las detecciones, ya ordenadas, con los eventos.

Por último, se calculan las puntuaciones de recall y precisión.

5.3 Función adicional: detección del balón

Dejando de lado el modelo del apartado anterior para la detección de pases, se ha desarrollado otro modelo que es capaz de detectar el balón en el transcurso del partido. Este modelo ha de entrenarse con imágenes etiquetadas en las que se pueda distinguir el balón del resto de elementos que aparecen.

Para ello, se han tomado varios vídeos ubicados en la carpeta 'train', aunque en este caso no ha sido necesarios usar todos ni tomar la duración entera de estos, y se han seguido los pasos que se desarrollan a continuación.

5.3.1 Etiquetado de las imágenes

Puesto que para esta función no se dispone de un fichero CSV con la información de las etiquetas, el etiquetado de las imágenes se ha hecho de forma manual.

Para ello, se han tomado fragmentos de los vídeos ofrecidos en el conjunto de datos de Kaggle y se ha utilizado la herramienta VOTT (Visual Object Tagging Tool) para generar etiquetas que indican la posición del balón en distintos fotogramas de estas imágenes.

VOTT es un software desarrollado por Microsoft que permite etiquetar varias imágenes de forma manual (distintos fotogramas de vídeos en este caso) para tareas de detección de objetos [78].

Figura 5-19. Herramienta de etiquetado VOTT

Para la detección de la pelota se ha utilizado un único tipo de etiqueta, pero en el caso de haber decidido detectar tanto la pelota como los jugadores, VOTT permite usar distintos tipos de etiqueta que distingan los elementos que se crean convenientes.

Para marcar la ubicación de la pelota en las distintas imágenes, VOTT permite crear diferentes polígonos que marcan el lugar de la imagen que es interesante para el modelo. En este caso, se han utilizado únicamente cajas cuadradas y rectangulares para marcar la pelota. Estas cajas reciben el nombre de cajas circundantes o delimitadoras. El uso de cajas delimitadoras es una de las técnicas más utilizadas para la anotación y etiquetado de imágenes y vídeos.

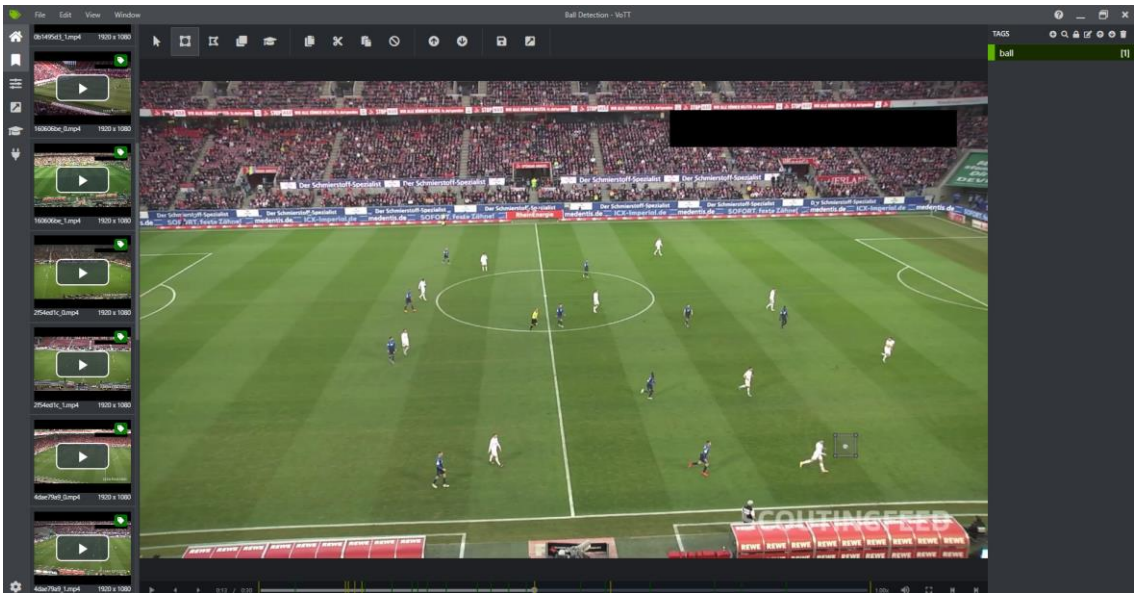


Figura 5-20. Ejemplo de etiquetado del balón mediante una caja delimitadora en VOTT

Tras etiquetar todos estos fotogramas, VOTT permite exportar toda la información que se ha generado mediante estas cajas delimitadoras.

Existen distintos métodos de exportación para estos datos. En este caso, se va a generar un archivo CSV (Comma-Separated Values), que es un tipo de archivo que permite mostrar los datos en forma de tabla, con los valores separados por coma.

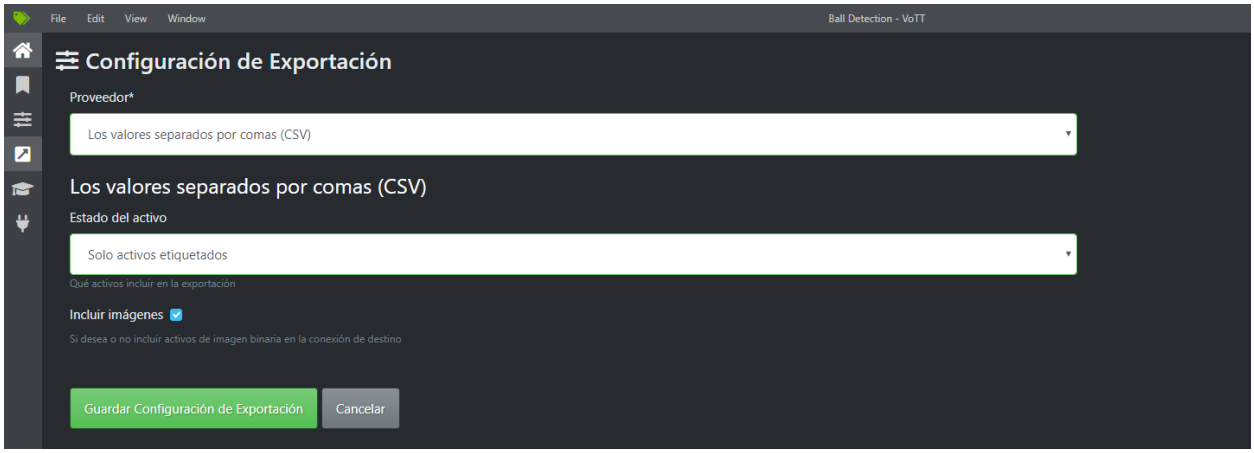


Figura 5-21. Exportación de los datos etiquetados a fichero CSV

En este archivo se encuentran las coordenadas de todas las cajas delimitadoras que se han marcado previamente, así como la etiqueta asignada a dichas coordenadas.

Como resultado se han etiquetado de forma manual 500 fotogramas aproximadamente para la detección de la pelota, procedentes de distintos vídeos del conjunto de datos de Kaggle .

5.3.2 Modelo para la detección del balón

Para llevar a cabo la detección del balón se ha utilizado el algoritmo YOLOv5. El algoritmo YOLO (You Only Look Once) es un sistema de código abierto para detección de objetos en tiempo real. Este algoritmo hace uso de una red neuronal convolucional para detectar objetos en imágenes. [79]

La red neuronal divide la imagen en regiones, usando esas cajas delimitadoras que se han mencionado anteriormente, asignando una probabilidad a cada una en función de la región a la que pertenecen.

El algoritmo aprende con representaciones generalizadas del objeto en cuestión, como en este caso se ha hecho con el balón, y esto le permite tener un bajo error para entradas nuevas, diferentes obviamente del conjunto de datos de entrenamiento.

5.3.2.1 Arquitectura de la red convolucional

La red consiste en tres partes principales:

- Backbone: Es una red neuronal convolucional que agrega y forma características de la imagen en diferentes granularidades.
- Neck: Consiste en una serie de capas que mezclan y combinan las características de la imagen como paso previo a la predicción.
- Head: Consume características del Neck y da pasos en la predicción.

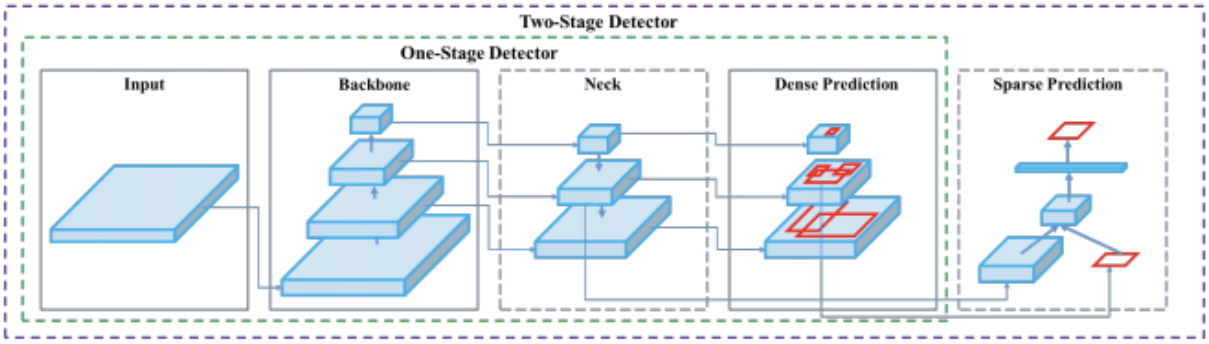


Figura 5-22. Proceso de detección de objetos en YOLOv5 [80]

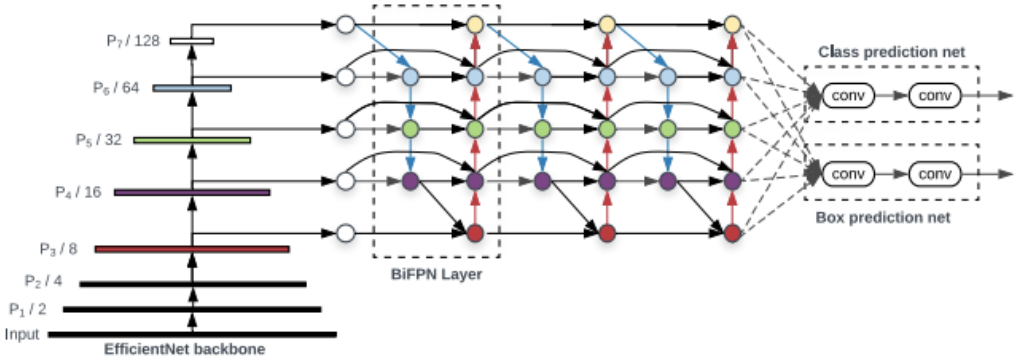


Figura 5-23. Arquitectura YOLOv5 [81]

Las capas convolucionales se encargan de la extracción de características de la imagen y las de conexión completa se encargan de calcular la probabilidad de salida y las coordenadas del objeto.

YOLO usa una arquitectura de detección de objetos muy rápida y precisa, lo que la hace idónea para vídeos en tiempo real. Esto es debido a que se reduce la detección a un problema de regresión lineal. Por otro lado, otro punto a favor es que, a diferencia de otros métodos que realiza cálculos solo sobre regiones de la imagen, YOLO realiza cálculos sobre la imagen completa, reduciendo de esta manera los errores a la hora de reconocer las clases de objetos.

Existen múltiples versiones del algoritmo, cada una con resultados diferentes en distintas aplicaciones. Cabe decir que YOLO no es una marca registrada por lo que algunas versiones pertenecen a distintos desarrolladores [82].

A nivel interno, la imagen se divide en una rejilla de tamaño M x M y si las coordenadas de un objeto caen en una celda de la cuadrícula, esa celda tendrá que detectar ese objeto.

Por otro lado, cada celda predice un número de cajas delimitadoras en base a sus puntuaciones de confianza (score). Este score refleja cómo de fiable es el modelo y cómo de precisa está siendo la predicción. Si no hay un objeto en una celda, el score en esa celda deberá ser cero.

5.3.3 Preparación y entrenamiento

Con un procedimiento similar al que se ha seguido para el apartado anterior para la detección de pases, en primer lugar, se ha de leer el fichero CSV que se ha generado usando el software VOTT. Se puede comprobar como el número de cajas delimitadoras que se mapean en el fichero es de 468 y el único tipo de etiqueta que se está utilizando es ‘ball’, que es la utilizada para el balón.

```
Number of ground truth bounding boxes:  468
Unique labels:  {'ball': 0}
```

Figura 5-24. Salida con el número de cajas delimitadoras y el nombre de la etiqueta

A continuación, se divide el conjunto de datos de entrada en un conjunto para el entrenamiento y otro para la validación, siendo el tamaño de este último un 20% del total. El modelo se probará con vídeos del conjunto de prueba que ofrece Kaggle.

Antes de pasar a entrenar el modelo, se deben ajustar las cajas delimitadoras al formato que acepta YOLO. Una caja delimitadora consta de distintas coordenadas que se usan para su identificación y la del objeto que se ha de detectar: x, y, w, h y la confianza. Las coordenadas x e y representan el centro del cuadro en relación con los límites de la celda de la cuadrícula. La coordenada w representa la anchura y h la altura. La confianza representa la posibilidad de que el objeto que se está tratando de detectar esté dentro de la caja.

Como ocurre en este caso, hay veces que es necesario convertir entre distintos formatos para el uso de estas coordenadas y se ha realizado de la siguiente forma:

$$X_c = \frac{X1 + X2}{2}$$

$$Yc = \frac{Y1 + Y2}{2}$$

$$Width = (X2 - X1)$$

$$Height = (Y2 - Y1)$$

Siendo (X1, Y1) las coordenadas de la esquina superior izquierda de la celda y (X2, Y2) las coordenadas de la esquina inferior derecha de la misma. (Xc, Yc) representan las coordenadas del centro de la celda.

Tras esto, se procede al entrenamiento del modelo. Para ello, se utiliza el siguiente comando:

```
!python train.py --img 1280 \
                 --batch {BATCH_SIZE} \
                 --epochs {EPOCHS} \
                 --data data.yaml \
                 --weights yolov5l6.pt \
                 --project dfl-ball
```

Figura 5-25. Comando para el entrenamiento del modelo

Con este comando se entrenará al modelo YOLOv5l6, con las imágenes redimensionadas a 1280 píxeles de ancho y 720 de alto, para mantener la relación de aspecto. Se elige esta dimensión de imagen pues YOLOv5 permite dimensionar las imágenes a 640 o 1280 píxeles de ancho, así que se ha elegido la segunda opción pues mejora la precisión en la detección de objetos pequeños, como es el caso del balón. El tamaño de lote es 2 y el número de épocas para el entrenamiento es 30.

En este caso, también se utiliza un archivo de pesos preentrenados, 'yolov5l6.pt', que sirve de punto de partida para el entrenamiento. En este archivo de pesos viene definido una tasa de aprendizaje inicial de 0.01.

Tras esto, se procede a utilizar vídeos de la carpeta 'test' para probar la eficacia del entrenamiento del modelo, cosa que se puede ver en el siguiente capítulo, en el que se muestran los resultados.

6 RESULTADOS

En este capítulo se muestran los resultados de los modelos desarrollados en el capítulo anterior. Las métricas principales para medir la calidad del modelo serán la precisión, el recall y el Average Precision, por lo que se mostrarán distintas iteraciones en el entrenamiento del modelo y la evolución de este resultado a lo largo de ellas.

6.1 Resultados para el modelo de detección de pases

En este primer apartado, se analizan los resultados obtenidos para el modelo de detección de pases, así como otras posibles alternativas que podrían haberse utilizado y el motivo del uso final de las elegidas.

Es importante aclarar que estos resultados que se muestran en este capítulo son fruto de detener el entrenamiento en el número de épocas que se indica y a continuación, pasar los datos de test a la red entrenada hasta ese momento. Esto puede realizarse gracias a las funcionalidades que aporta la plataforma Kaggle, que dispone de todo el conjunto de datos y que permite hacer uso de él (en este caso del conjunto de test) de forma sencilla, facilitando la prueba de los resultados de la red.

6.1.1 Comparación de resultados

6.1.1.1 Comparación de modelos disponibles en la librería 'timm'

Para el cálculo del resultado se han comparado distintos modelos pre-entrenados de la librería 'timm'. La arquitectura y algunas de las características de estos modelos se exponen en el capítulo 4. En este apartado se muestran los resultados obtenidos para cada uno de ellos y se explica el motivo de la elección final.

En la figura a continuación se muestra la comparación de los resultados de AP conseguidos para los distintos modelos pre-entrenados utilizados:

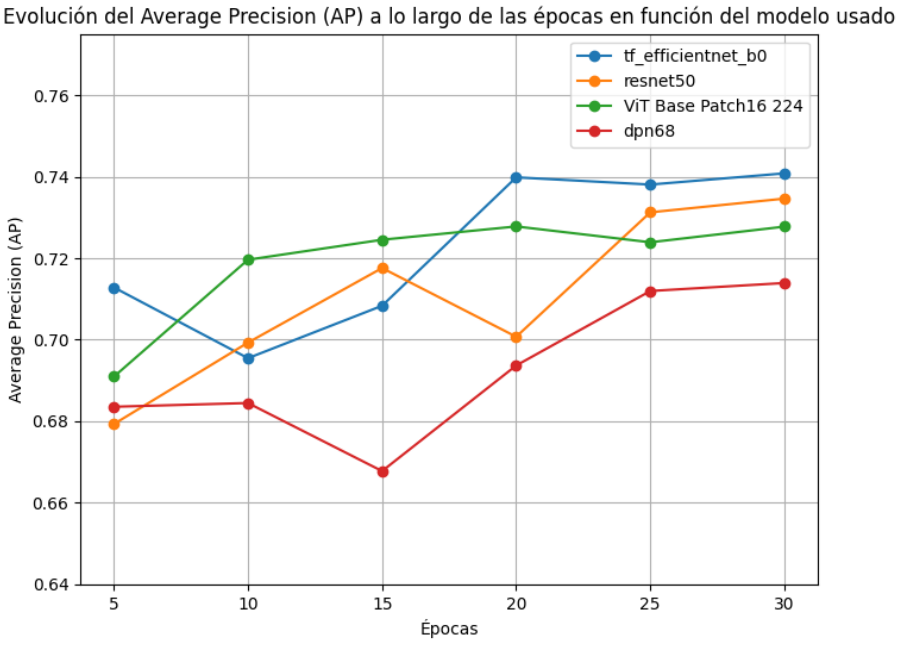


Figura 6-1. Evolución del Average Precision (AP) a lo largo de las épocas en función del modelo usado

En la siguiente tabla se expone a modo de resumen una comparación entre los 4 modelos que se han probado:

Modelo	Mejor AP conseguido	Número de parámetros	FLOPs (GFLOPs)
tf_efficient_b0	0.74	5.3 millones	0.39
resnet50	0.7346	25.6 millones	4
ViT Base Patch16 224	0.7277	87 millones	67
dpn68	0.7138	13 millones	3

Tabla 1. Resumen de los resultados obtenidos con distintos modelos pre-entrenados

En este caso, se obtiene un promedio para el **AP de 0.7408** lo cual es un valor que indica equilibrio general entre ambas medidas. Es importante recordar que los valores de AP se moverán entre 0 y 1, indicando un valor próximo a 0 que el modelo es muy pobre, con baja precisión y recall, y un valor cercano a 1 que el modelo es excelente y que obtiene valores altos tanto de precisión como de recall. En este caso, un valor entre 0.5 y 1 indica que el modelo tiene cierto equilibrio entre la precisión y el recall pero que aún tiene espacio para mejorar. Generalmente, por encima de 0.8, es considerado que el modelo tiene un muy bien rendimiento, por lo que este modelo se queda cerca de ese valor, siendo un modelo aceptable, pero con margen de mejora.

Con el modelo 'resnet50', el mejor resultado obtenido de **AP es de 0.7346**, es decir, muy similar a lo obtenido con el modelo de EfficientNet. El motivo por el que se elige este último mencionado es que, es un modelo más fácil de escalar, por lo que, a la hora de elegir un modelo para un proyecto en el que a priori no se conoce a la perfección la cantidad de parámetros que se necesitarán, esta es una decisión que puede decantar la balanza. Además, es un modelo que, por lo general, utiliza un menor número de parámetros que el modelo ResNet, del orden de 5 veces menos, lo que conlleva un mayor número de operaciones y, por lo tanto, un mayor coste computacional.

Con el modelo 'ViT Base Patch16 224' se consigue un valor de **AP de 0.7277.** De nuevo, este valor es muy similar al obtenido con el modelo de EfficientNet y el de ResNet, pero existe el mismo problema que con el de ResNet: el número de parámetros que utiliza el modelo y el número de FLOPs es mucho mayor, usando en este modelo aun más parámetros que en el anterior.

Con el modelo 'dpn68' se consigue un valor de **AP de 0.7138**, y, como ocurría con los anteriores modelos, requiere más recursos que el que se plantea de EfficientNet. Aun así, es un modelo que se acerca al nivel de eficiencia de EfficientNet, aunque cabe decir que su arquitectura es más compleja al ser una combinación de los dos modelos mencionados.

En definitiva, los resultados de AP son muy similares para todos los modelos probados, incluso puede dependiendo de la simulación alguno de ellos pudiera obtener un mejor resultado que el EfficientNet, pero, en resumidas cuentas, se eligió este modelo mencionado debido a su menor carga computacional, característica muy a tener en cuenta pues los recursos de los que se disponían eran limitados.

Por lo tanto, se concluye que, si se quiere un modelo que sea eficiente, sin un gran número de parámetros y que sea capaz de obtener un resultado correcto sin necesidad de lo mencionado, el que ofrece **EfficientNet** es muy válido. Por el contrario, si se dispone de una mayor cantidad de recursos y/o se quiere utilizar un modelo con mayor número de capas, se puede utilizar alguno de los otros mencionados, aunque, a priori, el resultado puede ser similar.

6.1.1.2 Uso de los valores de tolerancia de error

En segundo lugar, se procede a comparar los resultados en ausencia de la tolerancia de error. Como ya se ha mencionado en diversas ocasiones, estos valores de tolerancia aportan un mayor contexto al modelo y ajustan el arco temporal en el que el modelo aprende de cada uno de los eventos de interés.

El hecho de añadir estos valores de tolerancia es una práctica común en este tipo de modelos, que son entrenados con etiquetas que marcan el instante en el que ocurre un evento concreto, así que se ha decidido comprobar si en este caso es capaz de afectar correctamente el resultado final y de qué forma lo hace.

Se puede comprobar en los resultados que se muestran a continuación que, sin usar tolerancia, los resultados obtenidos son mucho peores que los que sí la usan.

En esta parte se obtienen resultados para los distintos valores de tolerancia utilizados, los cuales se detallan a continuación:

- Tolerancia para evento ‘challenge’: [0 0.20 0.40 0.60 0.70] medido en segundos.
- Tolerancia para evento ‘play’: [0 0.10 0.25 0.30 0.35] medido en segundos.
- Tolerancia para evento ‘throwin: [0 0.10 0.25 0.30 0.35] medido en segundos.

Estos valores de tolerancia se han elegido en vista de las pruebas que se realizan a continuación. Desde un principio, se comprobó que para el evento ‘challenge’ el valor de tolerancia necesitaba ser más alto que para los eventos ‘play’ y ‘throwin’. Esto tiene una fácil explicación, el evento ‘challenge’ se trata de una disputa del balón, cosa que en la mayoría de ocasiones no es algo puntual, sino que puede alargarse en el tiempo. Por el contrario, los eventos ‘play’ y ‘throwin’ sí que son instantáneos, pues se tratan de un pase durante el transcurso de juego y un saque de banda, que simplemente consisten en el desplazamiento del balón de un jugador a otro.

Dicho esto, la decisión de qué valores de tolerancia eran los más adecuados para el modelo se ha realizado en función de los resultados de las siguientes figuras, que representan la evolución del recall, precisión y AP en función de los valores de tolerancia, para cada uno de los eventos que se han mencionado, utilizando un umbral de 0.5:

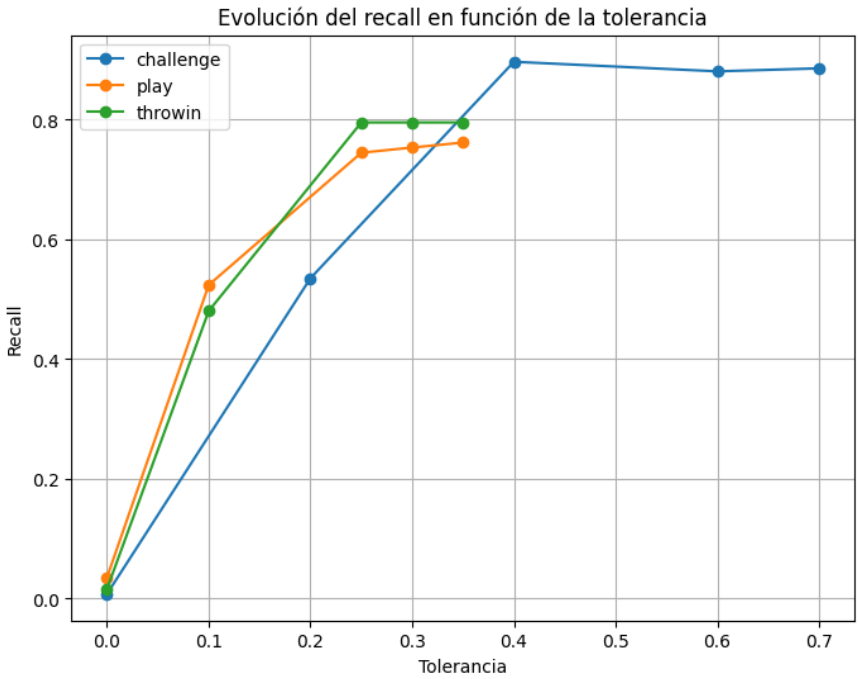


Figura 6-2. Evolución del recall en función de la tolerancia

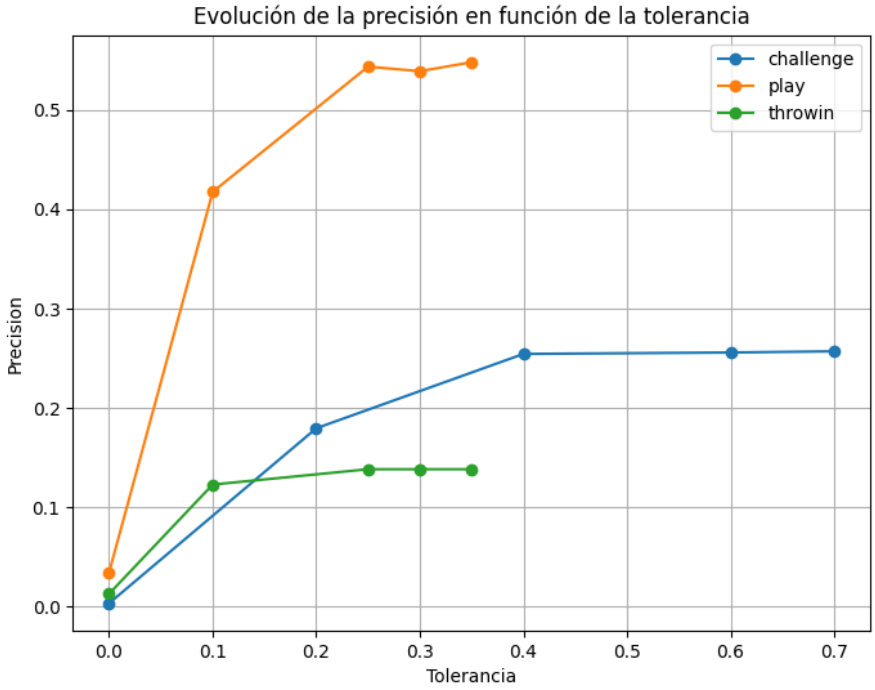


Figura 6-3. Evolución de la precisión en función de la tolerancia

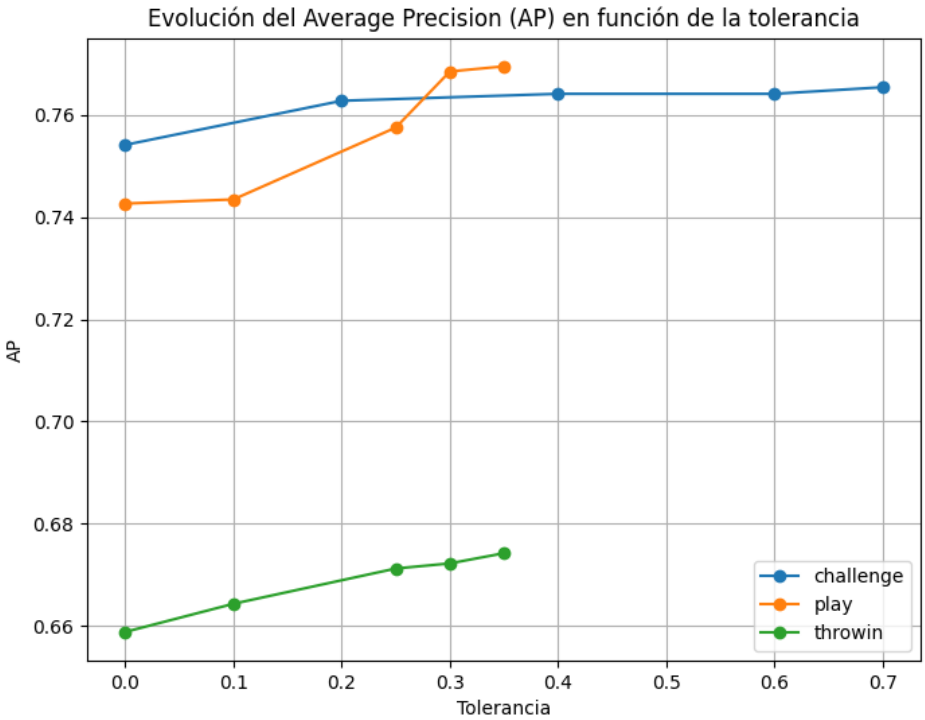


Figura 6-4. Evolución del Average Precision (AP) en función de la tolerancia

Los resultados dejan a la vista que el uso de tolerancia está más que justificado, pues los resultados para tolerancia 0 son exageradamente negativos con respecto a los que utilizan un valor más alto de esta. Esto significa que el modelo que se ha diseñado es completamente incapaz de detectar los eventos cuando se entrena con una tolerancia de error demasiado baja, aunque cabe decir que, desde un principio, este ha sido diseñado con el objetivo de utilizar un cierto valor de tolerancia que permitiera detectar más pases.

También cabe resaltar que a partir de un valor de tolerancia de 0.50 en el caso del evento 'challenge' y de 0.25 en el caso de los eventos 'play' y 'throwin' los resultados obtenidos no mejoran, por lo que se ha identificado

que los valores óptimos de tolerancia son los que se han indicado.

Por último, comentar la diferencia entre los valores de recall y de precisión, siendo mejores los resultados obtenidos para el recall. Esto se puede explicar de la mano con lo comentado anteriormente, pues el modelo se ha diseñado para detectar el mayor número de pases posible, aunque eso lleva al problema de que, en ocasiones, detecta como pases acciones del juego que realmente no lo son, es decir, falsos positivos. Siendo consciente de esto, se ha interpretado que es importante que la mayoría de los pases sean detectados, aunque algunos de ellos sean después clasificados como falsos positivos.

Dicho esto, se ha considerado el valor medio de la Precisión Promedio (AP) como la métrica para evaluar el resultado final del modelo. El AP proporciona un balance entre la precisión y el recall, y se ha observado que se consigue un mejor desempeño al ajustar la tolerancia en la detección de eventos. Así, aunque se priorice el recall para asegurar la detección de la mayoría de los pases, el uso del AP permite mantener un equilibrio adecuado, asegurando que el modelo sea efectivo y preciso en general.

6.1.1.3 Variaciones debidas a los distintos valores de tasa de aprendizaje

Por otro lado, se ha comprobado como afecta el valor de la tasa de aprendizaje al resultado, probando tres valores distintos y prestando atención a los distintos resultados obtenidos.

En primer lugar, se decidió utilizar una tasa de aprendizaje de 0.01, el cual es un valor comúnmente utilizado en Machine Learning y considerado, por lo general, como un buen punto de partida. De hecho, en muchas librerías, el valor de 0.01 para la tasa de aprendizaje está elegido de forma predeterminada.

Tras este valor, se decidió probar 0.1, valor diez veces mayor, y 0.001, diez veces menor, buscando mejorar el resultado obtenido con 0.01. Además de estos valores, se ha querido probar otros valores intermedios, en este caso, 0.05 y 0.0005.

Por lo tanto, los valores que van a utilizarse son 0.1, 0.05, 0.01, 0.005, 0.001. Los resultados se muestran en la siguiente figura:

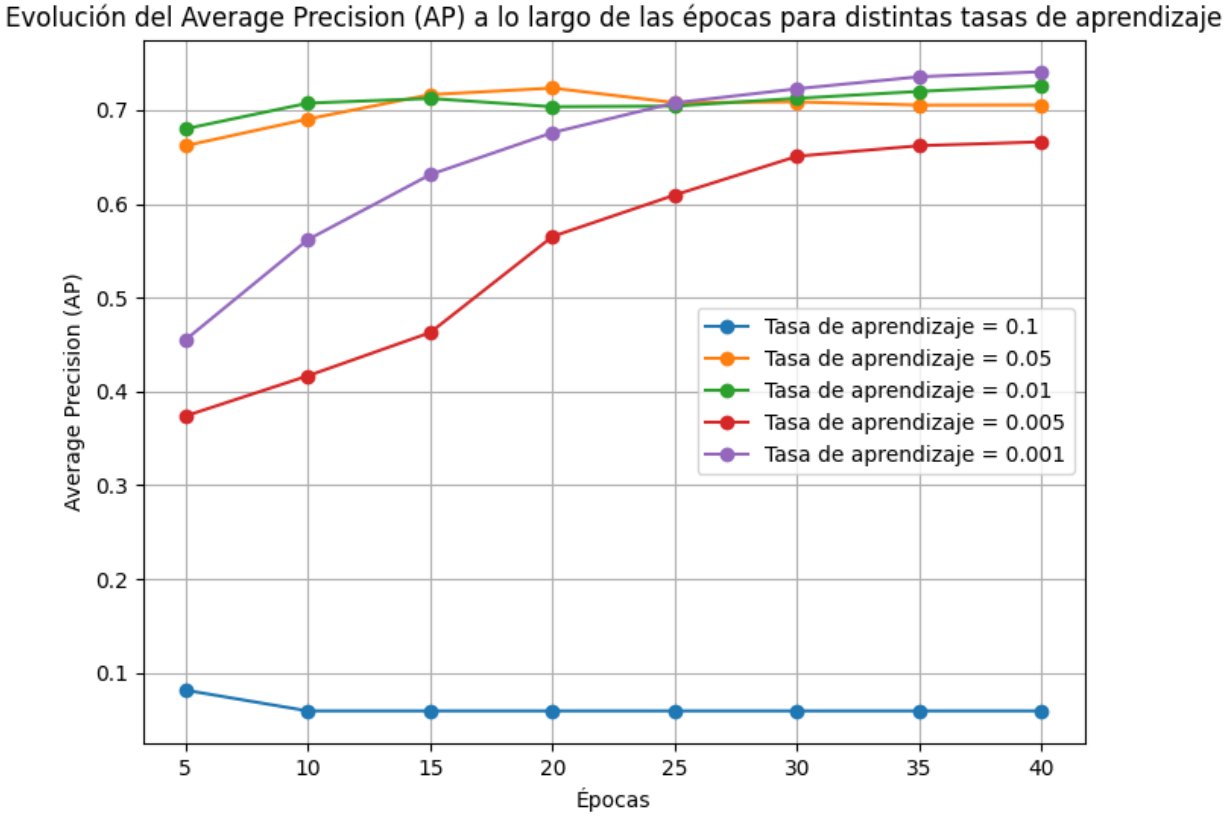


Figura 6-5. Evolución del Average Precision (AP) a lo largo de las épocas para distintas tasas de aprendizaje

Para una tasa de aprendizaje de 0.1, el modelo no converge, es decir, no consigue acercarse a la solución. Esto lleva a pensar que el salto que da el modelo entre iteración e iteración en busca de la solución es demasiado grande. El problema para este valor es que no se detectan la mayoría de los eventos, obteniéndose valores de recall que apenas superan 0.5. Este es un problema que se ha mencionado al hablar de la selección de la tolerancia y para el que se concluía que, según las reglas de la competición que plantea Kaggle, se prioriza el hecho de detectar la mayoría de estos eventos. Por lo tanto, este valor de tasa de aprendizaje queda descartado. Esto lleva a probar otro valor intermedio entre 0.1 y 0.01, que en este caso será 0.05.

Los resultados para una tasa de aprendizaje de 0.05 y 0.01 mejoran de forma drástica. Los valores de AP obtenidos para estas tasas de aprendizaje no distan mucho el uno del otro, siendo ligeramente mejor el resultado obtenido para 0.01.

Además, se prueba una tasa de aprendizaje de 0.005 y de 0.001. Para 0.005, se necesitan más épocas de entrenamiento que en las anteriores, debido a que el paso a la hora de buscar la solución es mucho más pequeño. Para 40 épocas de entrenamiento, el resultado que se obtiene es cercano a los conseguidos con 0.05 y 0.01. Algo similar ocurre para la tasa de 0.001, que necesita más épocas para converger en una mejor solución pero que obtiene, aunque por poco margen, los mejores resultados.

A la vista de esto, y a pesar de necesitar más épocas de entrenamiento, se ha decidido elegir una tasa de aprendizaje de 0.001.

6.1.1.4 Elección del optimizador

En cuanto a optimizadores, las opciones también son amplias y variadas, por lo que antes de elegir uno de ellos para el modelo, se han estudiado varias de estas opciones. Se van a comparar las siguientes: Adam, SGD (Stochastic Gradient Descent), RMSprop y Adagrad.

Los mejores resultados se obtienen para los optimizadores Adam, RMSProp y Adagrad, para los que se ha obtenido un AP máximo de 0.7579, 0.7147 y 0.6944, respectivamente. Por el contrario, SGD consigue peores resultados, con un AP máximo de 0.33566.

Esto se puede explicar a través de una característica que tienen en común el optimizador Adam y el RMSProp, que es el ajuste de la tasa de aprendizaje de forma individual para cada parámetro. Es decir, los parámetros del modelo que se actualizan con más frecuencia tienen tasas de aprendizaje más bajas, mientras que los que se mantienen más estables tienen tasas de aprendizaje más altas.

Por su parte, SGD no utiliza adaptación en su tasa de aprendizaje, lo que puede provocar que la convergencia sea menos eficiente. Se ha comprobado que dándole más epocas de entrenamiento, el resultado no mejora.

Por otro lado, Adagrad utiliza un método que consiste en que la tasa de aprendizaje disminuye a medida que se van actualizando los parámetros, lo que hace que el uso de este optimizer requiera de más épocas de entrenamiento que los anteriores para conseguir un mejor resultado. Se ha comprobado que para 5 épocas de entrenamiento el valor de AP obtenido es 0.559, pero que a medida que se entrenaba con más épocas este valor sube considerablemente, siendo de 0.6397 para 15 épocas de entrenamiento, de 0.6913 para 30 épocas de entrenamiento y de 0.6944 para 40 épocas de entrenamiento.

La comparación de los resultados usando los distintos optimizadores se puede apreciar en la siguiente figura:

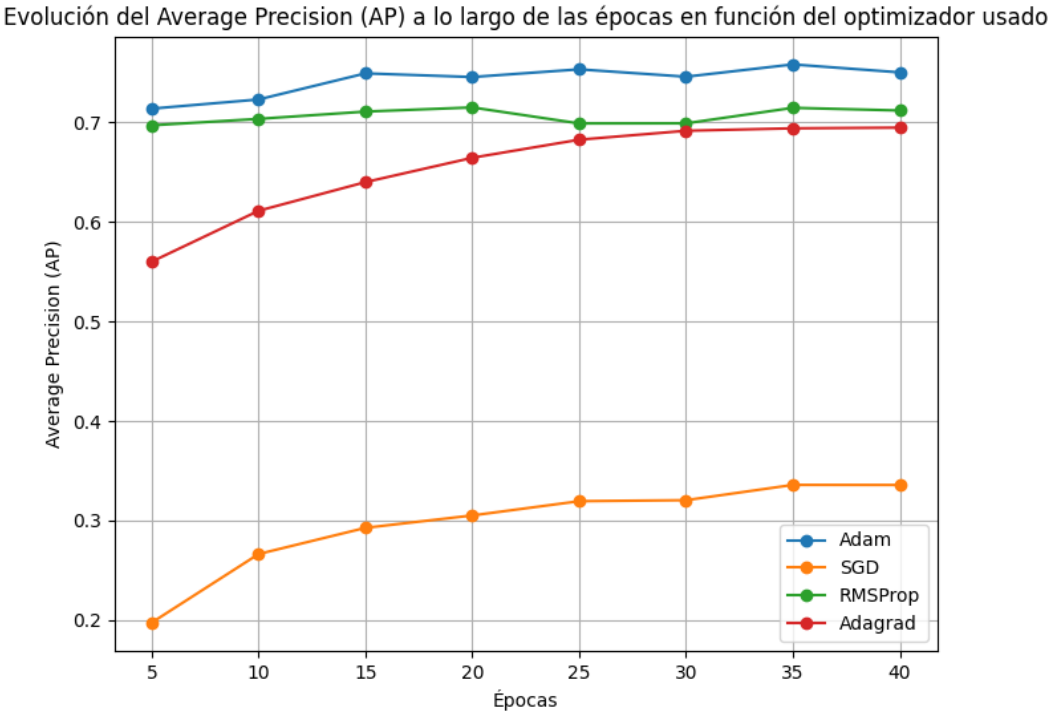


Figura 6-6. Evolución del Average Precision (AP) a lo largo de las épocas en función del optimizador usado

A la vista de estos resultados, se eligió el optimizador Adam para el resultado final del modelo, pues es el que obtuvo mejor resultado.

6.1.1.5 Resultado final

Por último, en este apartado simplemente se va a representar la evolución del modelo, con todas las elecciones que se han hecho en los apartados anteriores, para que quede claro cual es el resultado final obtenido:

En la siguiente figura se puede ver la evolución del AP para el evento 'challenge' a lo largo de las épocas de entrenamiento, para cada valor de tolerancia:

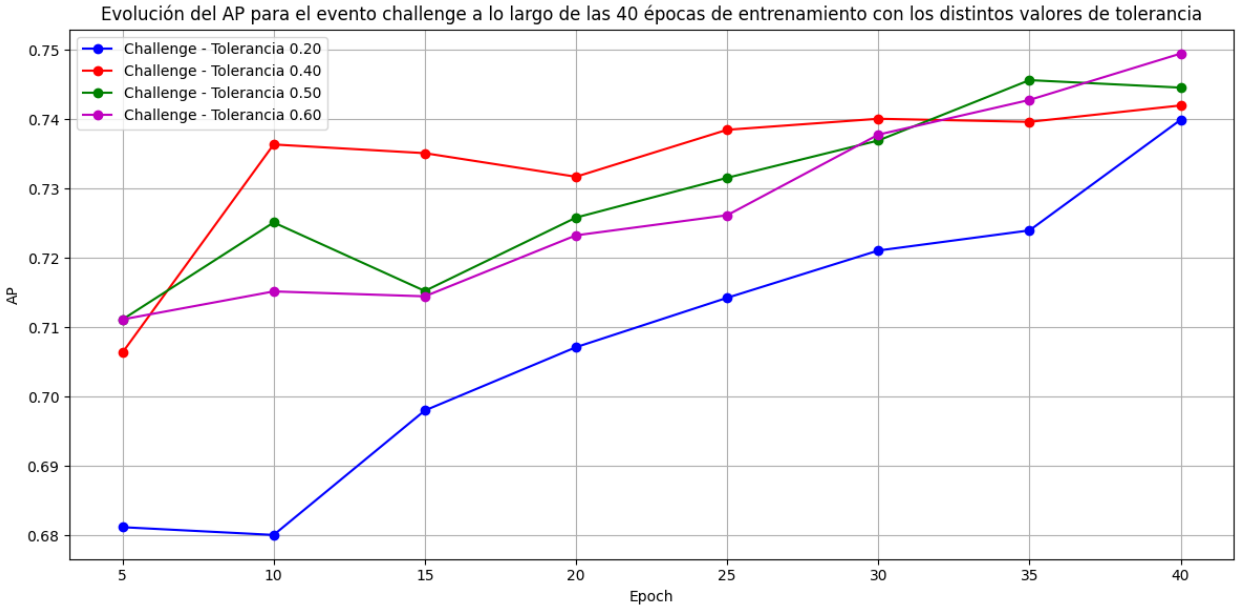


Figura 6-7. Evolución del AP para el evento 'challenge' a lo largo de las 40 épocas de entrenamiento con los distintos valores de tolerancia

Se puede ver en la figura como los valores de AP para el evento ‘challenge’ mejoran con el paso de las épocas y alcanzan un mejor resultado para las tolerancias de 0.50 y 0.60, aunque cabe decir que los valores son muy similares y pueden depender de la simulación realizada.

A continuación, se expone este mismo resultado para el evento ‘play’:

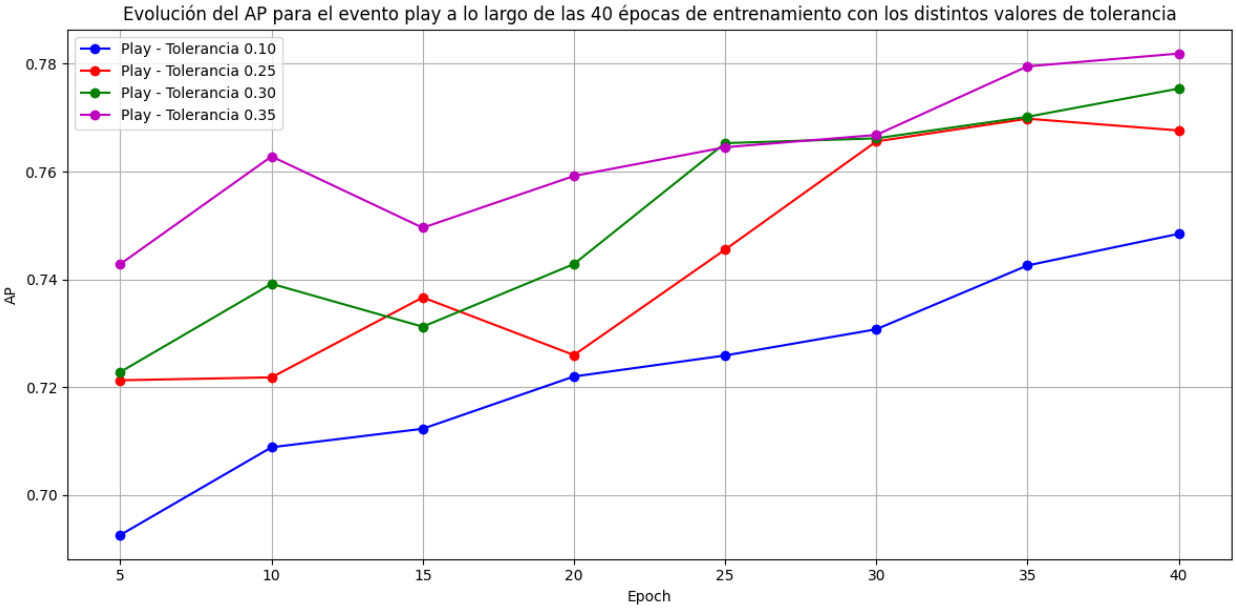


Figura 6-8. Evolución del AP para el evento play a lo largo de las 40 épocas de entrenamiento con los distintos valores de tolerancia

Lo obtenido en este caso es similar al anterior, con la evolución ascendente de los valores a medida que avanzan las épocas de entrenamiento. En este caso, los valores obtenidos para tolerancias 0.35, 0.30 y 0.25 son bastante mejores que los obtenidos con tolerancia 0.20.

Por último, el resultado del AP para el evento ‘throwin’:

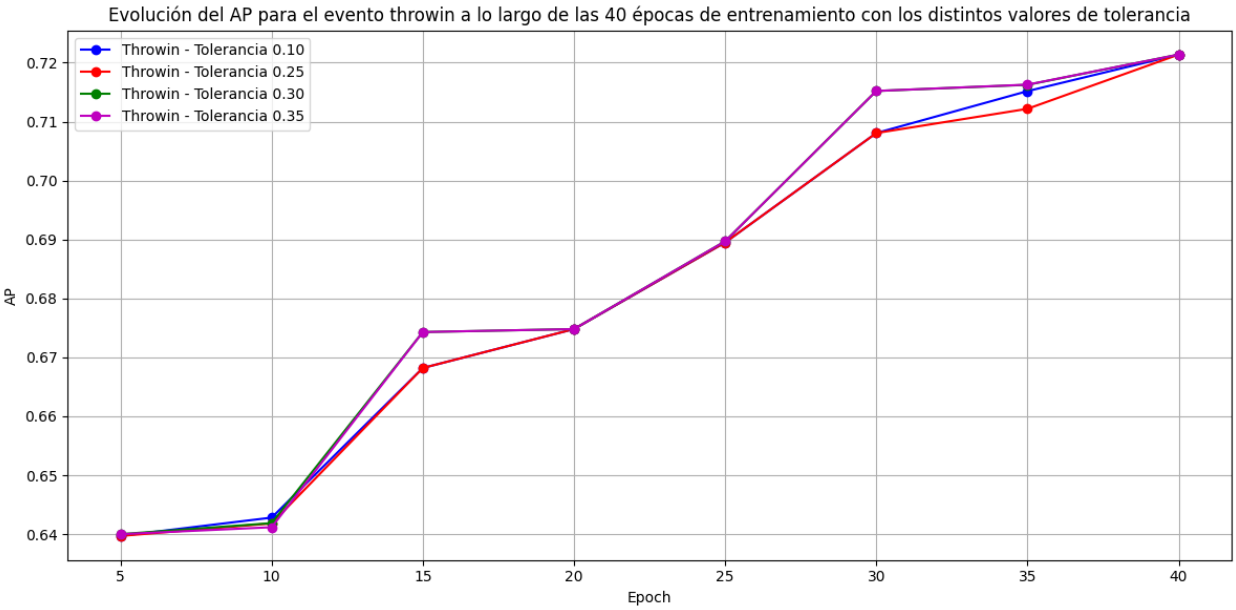


Figura 6-9. Evolución del AP para el evento throwin a lo largo de las 40 épocas de entrenamiento con los distintos valores de tolerancia

En este caso, el evento ‘throwin’ tiene unos resultados un tanto especiales, pues los valores de AP obtenidos son prácticamente idénticos para todos los valores de tolerancia. Esto puede ser debido a la propia naturaleza del

saque de banda, que se trata de una acción instantánea y que no requiere de tolerancia para medirse correctamente.

En las siguientes gráficas pueden verse los valores de recall y precisión a lo largo de las 30 épocas de entrenamiento para los distintos eventos. Para el evento challenge se ha escogido una tolerancia de 0.50 y para los eventos play y throwin de 0.30. Estos valores se han elegido por ser valores intermedios y por ser valores a partir de los cuales las variaciones en los resultados son mínimas:

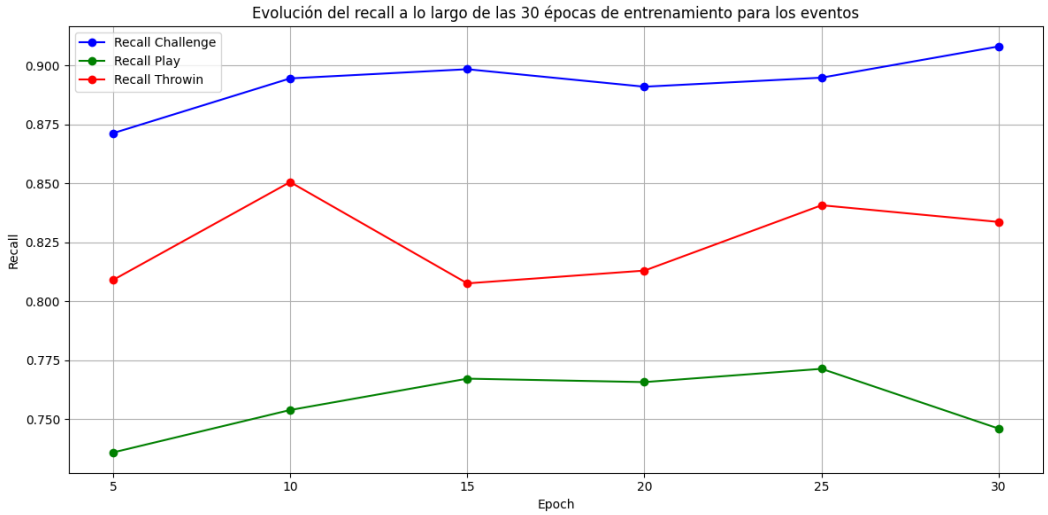


Figura 6-10. Evolución del recall a lo largo de las 30 épocas de entrenamiento para los eventos

En ellos se puede ver como el recall que se obtiene es de 0.87 en el peor de los casos para el evento challenge, de 0.74 en el evento play y de 0.81 en el evento throwin.

A continuación, lo mismo, pero para los valores de precisión:

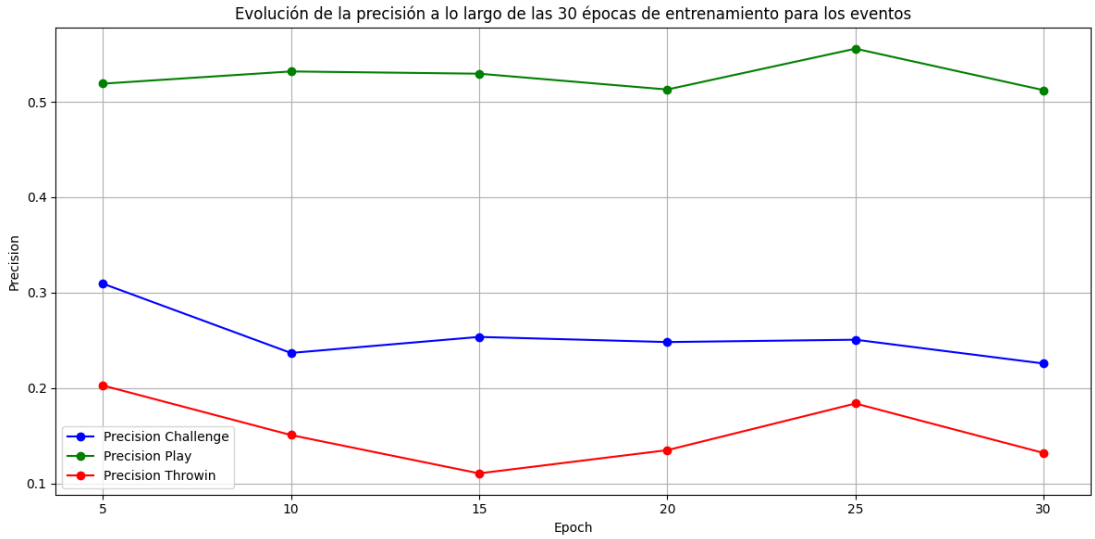


Figura 6-11. Evolución de la precisión a lo largo de las 30 épocas de entrenamiento para los eventos

Por otro lado, los resultados de la precisión no parecen tan buenos a simple vista, para los eventos challenge y throwin, que no superan los 0.3 y 0.2, respectivamente. Sí son mejores para el evento play, que supera el 0.58. Esto sugiere que el modelo es capaz de identificar muchos casos de verdaderos positivos pero que también genera muchos falsos positivos.

6.2 Resultados del modelo para la detección del balón

A continuación, se van a evaluar los resultados del entrenamiento del modelo para, posteriormente, utilizar dos vídeos de la carpeta 'test' y comprobar su funcionamiento.

En la siguiente figura puede verse la salida del proceso de entrenamiento, que muestra la siguiente información:

- Columna from: El valor -1 indica que la entrada de esa capa proviene de la salida de la capa anterior.
- Columna params: Indica el número de parámetros de esa capa.
- Columna module: Indica el tipo de capa utilizado. 'Conv' se refiere a una capa convolucional y 'C3' es una capa específica de YOLOv5.
- Columna arguments: Indica características concretas de esa capa. Por ejemplo, en la primera capa [3,64,6,2,2] indica que la capa tiene 3 canales de entrada, 63 de salida, un kernel de tamaño 6, un stride de 2 y un padding de 2.

En la parte baja de la figura se señala que el modelo tiene 346 capas en total, 76118664 parámetros, 8 gradientes y requiere 109.9 GFLOPs (giga operaciones de coma flotante por segundo).

Por último, como resultados del entrenamiento se consigue una precisión del 82.1% y un recall del 74.5%.

```
Overriding model.yaml nc=80 with nc=1

                 from  n    params  module                                  arguments
  0                -1  1      7040  models.common.Conv                      [3, 64, 6, 2, 2]
  1                -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
  2                -1  3    156928  models.common.C3                        [128, 128, 3]
  3                -1  1    295424  models.common.Conv                      [128, 256, 3, 2]
  4                -1  6   1118208  models.common.C3                        [256, 256, 6]
  5                -1  1   1180672  models.common.Conv                      [256, 512, 3, 2]
  6                -1  9   6433792  models.common.C3                        [512, 512, 9]
  7                -1  1   3540480  models.common.Conv                      [512, 768, 3, 2]
  8                -1  3   5611008  models.common.C3                        [768, 768, 3]
  9                -1  1   7079936  models.common.Conv                      [768, 1024, 3, 2]
...
Model summary: 346 layers, 76118664 parameters, 0 gradients, 109.9 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 24/24
                   all         94         94      0.821      0.745      0.716      0.235
```

Figura 6-12. Salida del proceso de entrenamiento

A continuación, se insertan distintos gráficos que muestran la evolución del entrenamiento y de la validación con el paso de las épocas. El algoritmo YOLO tiene dos componentes principales que minimiza en su función de pérdidas que son box_loss, obj_loss y cls_loss.

En primer lugar, el box_loss es una pérdida asociada a la predicción de las cajas delimitadoras sobre los objetos, en este caso, el balón. Concretamente, este componente mide la diferencia entre las cajas predichas por el modelo y las cajas reales de las imágenes de entrenamiento.

Por otro lado, obj_loss se refiere a la pérdida asociada a la probabilidad de que un objeto, en este caso el balón, esté dentro de una de las cajas que el modelo ha predicho. Es decir, mide la diferencia entre la probabilidad predicha de que una caja contenga el balón y la realidad.

Por último, cls_loss no será de mucha utilidad en este caso, pues mide la precisión de la clasificación de los objetos según su clase. En este ejemplo solo se utiliza una clase, el balón, así que el valor de cls_loss no será relevante.

La primera de las métricas que se ha utilizado es la precisión. La evolución de la precisión a lo largo de las

épocas del entrenamiento y de validación es la siguiente:

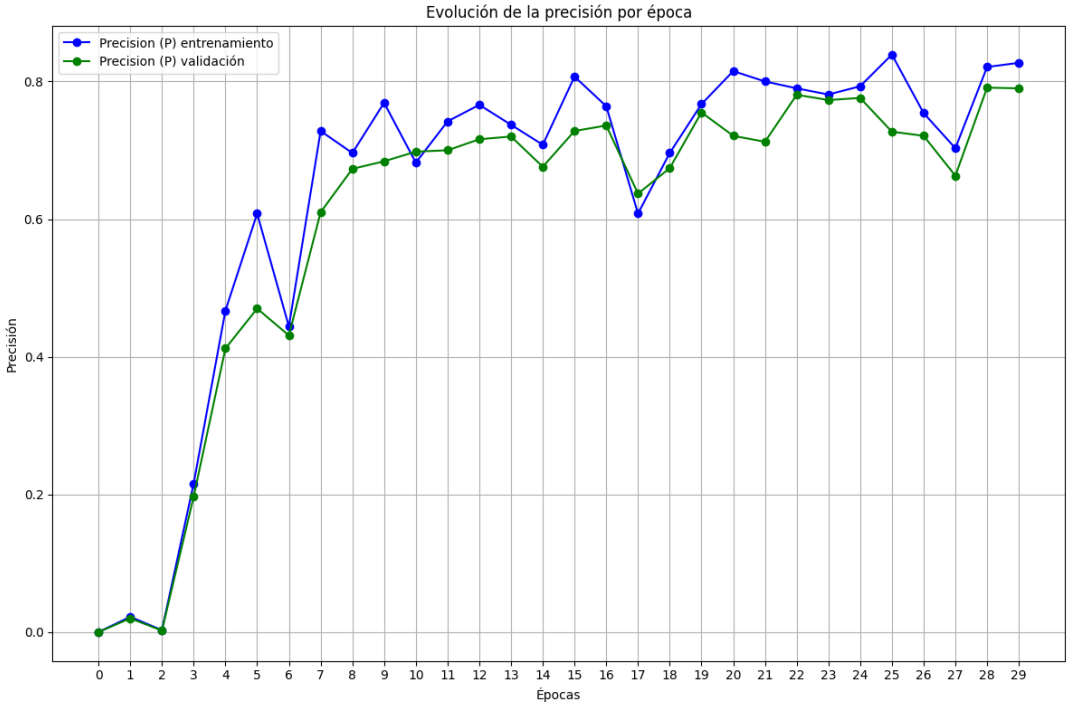


Figura 6-13. Evolución de la precisión por época

En esta gráfica se puede apreciar como la precisión experimenta una gran mejora a partir de la tercera época de entrenamiento y finalmente se estabiliza alrededor de 0.8 a partir de la vigésima época.

Los resultados de la validación son ligeramente peores, cosa que es lógica, pues el modelo está familiarizado con los datos del conjunto de entrenamiento, por lo que, al introducir nuevos datos, en este caso del conjunto de validación se obtienen resultados algo peores. Aun así, los resultados obtenidos son bastante satisfactorios y no distan mucho de los del entrenamiento.

Por otro lado, se ha evaluado el recall. Su evolución es la siguiente:

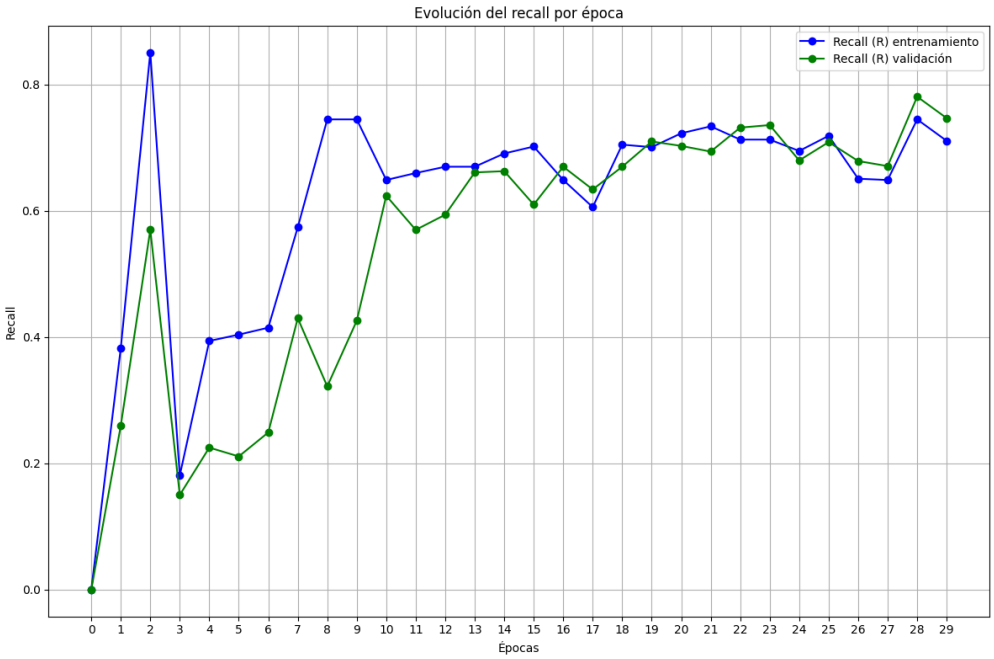


Figura 6-14. Evolución del recall por época

En el caso del recall, este encuentra un valor estable, alrededor de 0.7, a partir de la novena época de entrenamiento.

Por último, se utiliza el mAP@0.5, es decir el Mean Average Precision con valor de umbral para el IoU de 0.5. El IoU (Intersection Over Unit) es un parámetro que mide la coincidencia entre las áreas de la caja delimitadora predicha y la caja real. Esto quiere decir que usa un umbral de forma que una detección se considera como correcta (verdadero positivo) si la coincidencia entre el área de la caja predicha y el de la caja real es igual o superior 0.5. Este es un parámetro propio de la detección de objetos, que proporciona una evaluación global del rendimiento del modelo, pues combina tanto la precisión como el recall.

Su evolución es la siguiente:

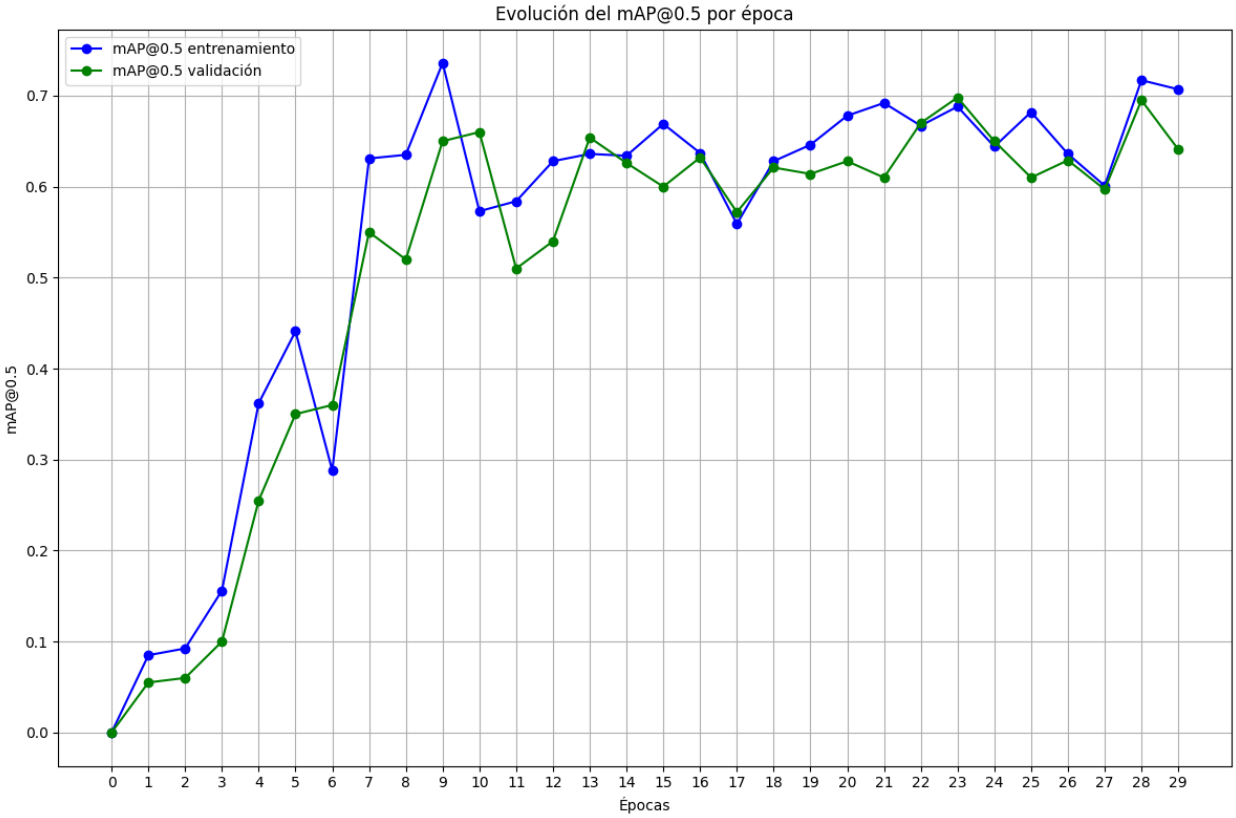


Figura 6-15. Evolución del mAP@0.5 por época

Su valor se mantiene estable entre 0.6 y 0.7 a partir de la duodécima época.

Para evaluar los resultados del modelo para la detección del balón se han utilizado dos vídeos del directorio 'test' de Kaggle, distinto de los que se han utilizado para el entrenamiento del modelo.

Para probar la detección se ha utilizado el script 'detect.py' que ofrece YOLOv5. Este script recibe como argumentos:

- Weights: Se le indica el archivo de pesos que es una de las salidas del modelo previamente entrenado.
- Source: Se le indica el vídeo del que hará la detección.
- Img: Se le indica la resolución de las imágenes que se van a procesar. En este caso 1280 es el ancho en píxeles.
- Project: Nombre del proyecto.

```
!python detect.py --weights {best_weights} \
                  --source '/content/dataset/test/0b1495d3_1.mp4' \
                  --img 1280 \
                  --project {project_name}
```

Figura 6-16. Uso del script detect.py

El script utiliza los pesos del modelo entrenado y trata de encontrar el balón en el vídeo seleccionado, del que se obtiene la siguiente salida:

```
detect: weights=['/content/yolov5/dfl-ball/exp/weights/best.pt'], source=/content/dataset/test/0b1495d3_1.mp4,
YOLOv5 🚀 v7.0-207-gdf48c20 Python-3.10.12 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)

Fusing layers...
Model summary: 346 layers, 76118664 parameters, 0 gradients, 109.9 GFLOPs
video 1/1 (1/750) /content/dataset/test/0b1495d3_1.mp4: 768x1280 1 ball, 81.8ms
video 1/1 (2/750) /content/dataset/test/0b1495d3_1.mp4: 768x1280 1 ball, 58.2ms
video 1/1 (3/750) /content/dataset/test/0b1495d3_1.mp4: 768x1280 1 ball, 58.5ms
video 1/1 (4/750) /content/dataset/test/0b1495d3_1.mp4: 768x1280 1 ball, 57.5ms
video 1/1 (5/750) /content/dataset/test/0b1495d3_1.mp4: 768x1280 1 ball, 59.4ms
video 1/1 (6/750) /content/dataset/test/0b1495d3_1.mp4: 768x1280 1 ball, 58.1ms
video 1/1 (7/750) /content/dataset/test/0b1495d3_1.mp4: 768x1280 1 ball, 56.7ms
video 1/1 (8/750) /content/dataset/test/0b1495d3_1.mp4: 768x1280 1 ball, 58.3ms
video 1/1 (9/750) /content/dataset/test/0b1495d3_1.mp4: 768x1280 1 ball, 58.0ms
video 1/1 (10/750) /content/dataset/test/0b1495d3_1.mp4: 768x1280 1 ball, 58.2ms
video 1/1 (11/750) /content/dataset/test/0b1495d3_1.mp4: 768x1280 1 ball, 58.9ms
video 1/1 (12/750) /content/dataset/test/0b1495d3_1.mp4: 768x1280 1 ball, 58.4ms
video 1/1 (13/750) /content/dataset/test/0b1495d3_1.mp4: 768x1280 1 ball, 57.7ms
video 1/1 (14/750) /content/dataset/test/0b1495d3_1.mp4: 768x1280 1 ball, 58.3ms
video 1/1 (15/750) /content/dataset/test/0b1495d3_1.mp4: 768x1280 1 ball, 57.6ms
video 1/1 (16/750) /content/dataset/test/0b1495d3_1.mp4: 768x1280 1 ball, 58.2ms
video 1/1 (17/750) /content/dataset/test/0b1495d3_1.mp4: 768x1280 1 ball, 58.4ms
video 1/1 (18/750) /content/dataset/test/0b1495d3_1.mp4: 768x1280 1 ball, 58.2ms
video 1/1 (19/750) /content/dataset/test/0b1495d3_1.mp4: 768x1280 1 ball, 57.7ms
video 1/1 (20/750) /content/dataset/test/0b1495d3_1.mp4: 768x1280 1 ball, 58.4ms
...
video 1/1 (749/750) /content/dataset/test/0b1495d3_1.mp4: 768x1280 (no detections), 62.0ms
video 1/1 (750/750) /content/dataset/test/0b1495d3_1.mp4: 768x1280 (no detections), 61.2ms
Speed: 1.2ms pre-process, 59.5ms inference, 1.3ms NMS per image at shape (1, 3, 1280, 1280)
Results saved to 0b1495d3_1/exp
```

Figura 6-17. Salida del script detect.py

En esta salida, que está truncada puesto que tiene 750 líneas, se indican las detecciones que el modelo ha realizado sobre el vídeo que se le ha pasado.

Tal y como se ha explicado en el capítulo anterior, se han generado unas 500 etiquetas aproximadamente para el correcto funcionamiento del modelo para la detección del balón. Previamente a esto, se realizó una prueba con un menor número de etiquetas, alrededor de 200, con un peor resultado. Esto puede verse en el vídeo que se encuentra en el siguiente enlace:

https://drive.google.com/file/d/1S617nJLmt0CzmaybFgvu98p4Le_acAOs/view?usp=sharing

En el vídeo puede apreciarse que el modelo no es capaz de detectar el balón cuando está en el aire y aun cuando está en el suelo, no siempre lo detecta. Además, la precisión con la que lo hace no es óptima, llegando a valores alrededor de 0.5 como máximo.

Figura 6-18. Fallo en la detección del balón en el aire

En esta primera imagen, el balón se encuentra en el aire y apenas es diferenciable en pantalla debido a la toma que aporta la cámara del vídeo y a los colores que se ven en el fondo. Se ha marcado manualmente el balón con un recuadro azul para que pueda comprobarse su ubicación de forma más sencilla. Este es un de los problemas que tiene el modelo y que es difícil de solventar, en especial si la calidad del vídeo no es especialmente buena o los colores del fondo de la imagen son fácilmente confundibles con el balón.

Figura 6-19. Fallo en la detección del balón entre jugadores

En este otro caso, el balón se encuentra "escondido" entre dos jugadores, que dificultan su visión desde el ángulo que proporciona la cámara en este vídeo. De nuevo, se ha marcado con un cuadro azul para facilitar su ubicación. Al igual que en el caso anterior, en este el modelo también encuentra dificultades para seguir el movimiento del balón y, por tanto, detectarlo

Figura 6-20. Detección del balón poco precisa

Por último, recalcar el hecho de que, en ocasiones, a pesar de detectar el balón, el nivel de precisión con el que el modelo indica que se trata del balón es muy bajo.

Por ello, se ha decidido generar más etiquetas, llegando a 500 etiquetas distintas, para optimizar el funcionamiento del modelo.

En este nuevo intento, ahora con más etiquetas, el modelo consigue detectar el balón en más ocasiones que en el caso anterior, y con una mayor precisión. Si bien es cierto que, en ocasiones, cuando el balón va por el aire o se encuentra en zonas en las que hay sombra o están más alejadas del plano de cámara, sigue sin ser capaz de detectarlo con facilidad.

Algunas imágenes de ejemplo son las siguientes:

Figura 6-21. Detección del balón en el aire

En esta primera imagen, el model es capaz de detectar el balón en el aire, cosa que antes no ocurría, aunque como se podrá ver en el vídeo completo, lo pierde con facilidad, y en ocasiones confunde otros objetos de color blanco con el balón.

Figura 6-22. Detección más precisa del balón

En esta otra imagen, se puede ver como la precisión con la que se detecta el balón es mayor que en el caso anterior, en el que las detecciones oscilaban alrededor de 0.5. En este caso, las detecciones de una mayor precisión, como se puede ver en esta imagen, que alcanza un valor de 0.85

El enlace al vídeo completo es el siguiente: https://drive.google.com/file/d/1RFOvAujDSBvL9HJIOb-PG0oBjh_0gb3s/view?usp=sharing

Tras esto, se ha decidido usar otro vídeo del conjunto de test. En este vídeo, el balón pasa mucho más tiempo en el suelo. Además, el ángulo de la cámara y las luces debido al sol hacen que la pelota sea detectable más fácilmente.

En este nuevo vídeo, se consigue detectar el balón con más precisión que el anterior, obteniendo valores de precisión cercanos a 0.9 y haciendo un mejor seguimiento del mismo a lo largo de todo el vídeo.

Pero la mejor forma de comprobar si está detectando el balón correctamente es comprobarlo en el vídeo que se genera a la salida. Aquí algunas imágenes del vídeo:

Figura 6-23. Detección del balón en vídeo (I)

Figura 6-24. Detección del balón en vídeo (II)

Figura 6-25. Detección del balón en vídeo (III)

En el vídeo, el balón se marca con una caja roja, el nombre de la etiqueta (ball) y un número a su lado que indica el nivel de precisión con el que el modelo considera que realmente está detectando el objeto que dice.

El vídeo completo puede verse en el siguiente enlace:

https://drive.google.com/file/d/14YHSyHAOGWTWSjHOHhM0mjEIyuWQQms9/view?usp=drive_link

Por lo general, el modelo es capaz de detectar el balón correctamente y de seguir su movimiento a lo largo del campo, si bien es cierto que, cuando el balón vuela por el aire o se esconde detrás de algún jugador o algún otro elemento, el modelo encuentra complicaciones para ubicarlo.

Para completar el estudio, se comparan los valores de AP, precisión y recall para ambos modelos, el de 200 etiquetas y el de 500.

En primer lugar, se comparan los valores de precisión de ambos modelos:

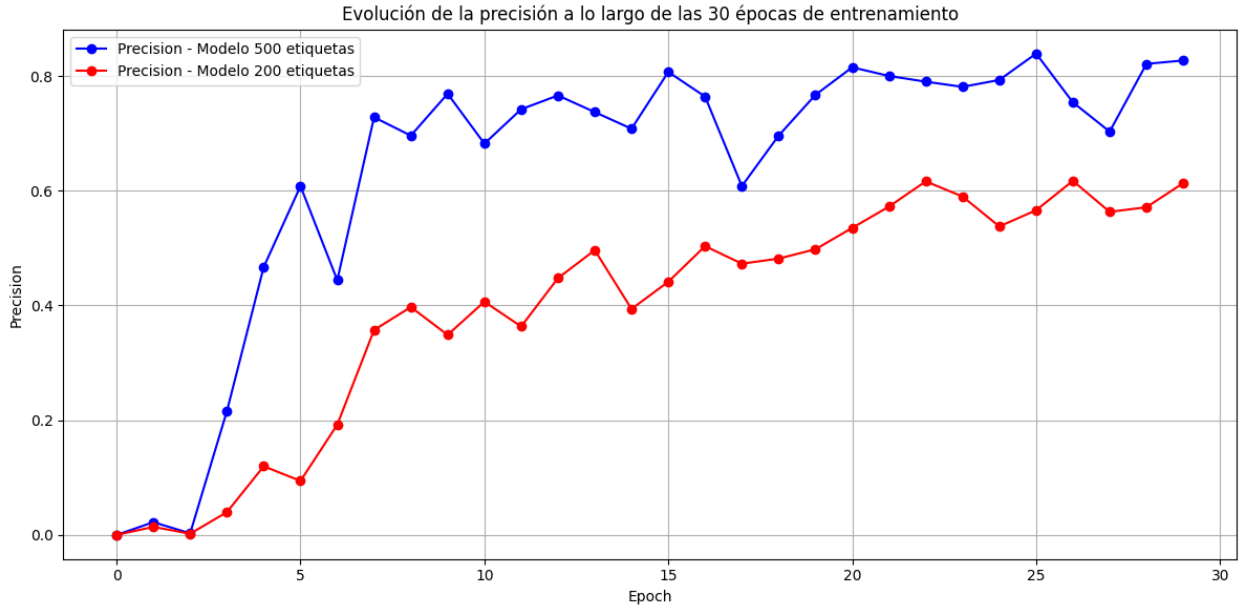


Figura 6-26. Comparación de la evolución de la precisión en los modelos de 200 y 500 etiquetas

Se consigue una precisión superior a 0.8 en el caso del modelo entrenado con 500 etiquetas, mientras que para el modelo entrenado con 200 etiquetas la precisión obtenida está alrededor de 0.6

A continuación, los valores de recall:

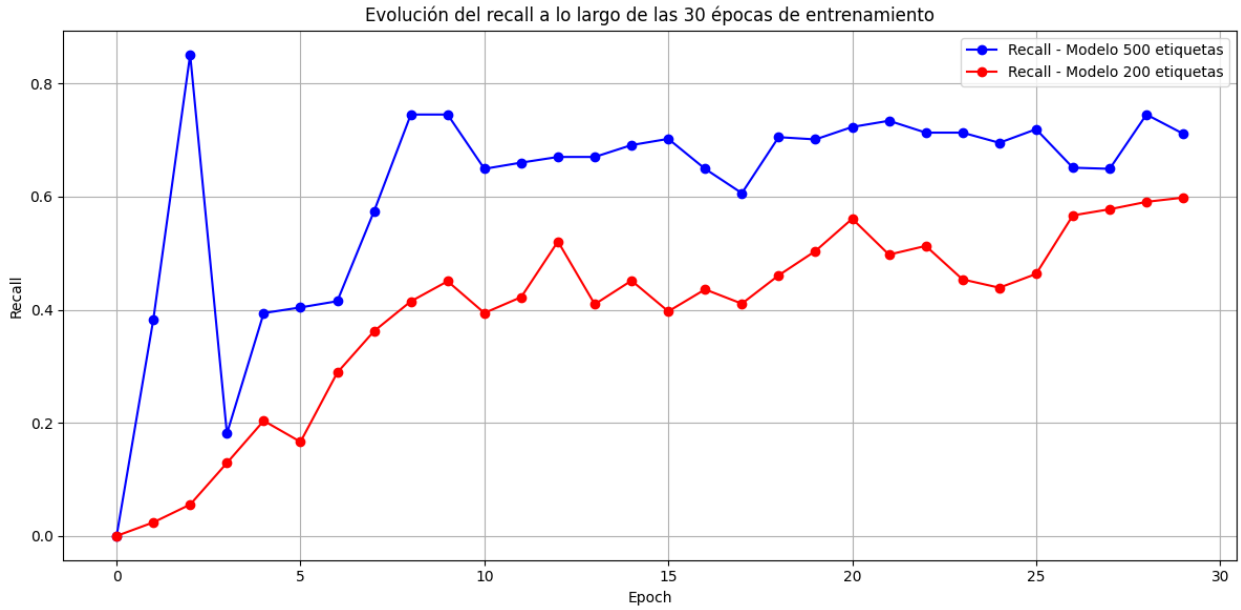


Figura 6-27. Comparación de la evolución del recall en los modelos de 200 y 500 etiquetas

Los valores de recall más altos para el modelo entrenado con 500 etiquetas están en torno a 0.73, mientras que para el modelo de 200 etiquetas el valor más alto es de 0.598.

Por último, los valores de AP:

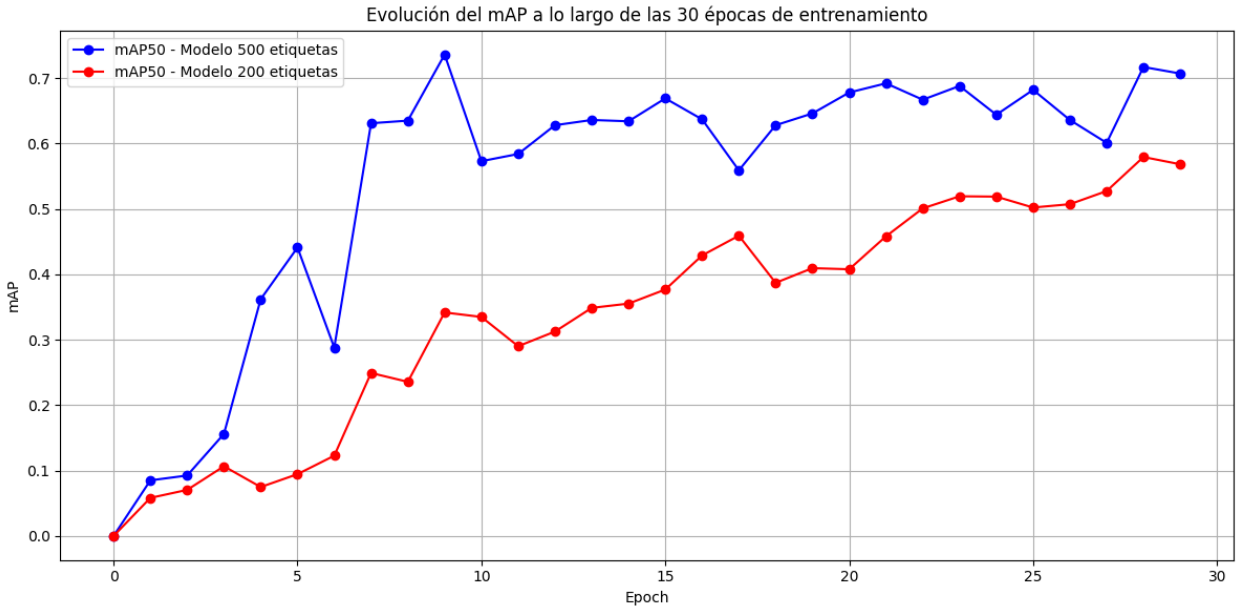


Figura 6-28. Comparación de la evolución del mAP en los modelos de 200 y 500 etiquetas

Los valores obtenidos para el mAP para el modelo entrenado 500 etiquetas son superiores a 0.7, mientras que el valor más alto del modelo entrenado con 200 etiquetas es 0.58.

7 CONCLUSIÓN

Este último capítulo sirve como una breve conclusión del trabajo, tratando de forma general los temas que se han tratado y posibles mejoras y líneas futuras que incluir y desarrollar.

Bajo mi experiencia, este trabajo sirve como primer acercamiento a las redes neuronales y al Machine Learning a una persona que nunca ha trabajado en este campo. Digo esto porque, yo mismo, antes de comenzar el trabajo, no conocía prácticamente nada de Machine Learning y he tenido que aprender poco a poco todo lo que he expuesto el proyecto. Por ese motivo, he tratado de documentar todas las cosas que he aprendido en este camino, o al menos aquellas que he considerado más importante.

El trabajo contextualiza la Inteligencia Artificial en la actualidad, haciendo un recorrido previo por la evolución que ha tenido en el pasado y que ha llevado a su situación actual, extendiéndose en el aprendizaje máquina y las redes neuronales, para acabar con la aplicación de muchas de las herramientas que se han desarrollado a lo largo del trabajo. Si bien es cierto que todo lo que se explica no se usa de forma directa en la aplicación final, el entendimiento de todas las partes hace que se logre comprender de mejor manera el porqué de usar una solución frente a otra y de forma general, entender mejor el contexto y la visión que se ha querido dar.

7.1 Líneas futuras

Como líneas futuras y posibles mejoras en los modelos que se han expuesto como resultados, se cree que resultaría de interés desarrollar los siguientes puntos:

- Afinar el valor de algunos hiperparámetros del modelo de detección de pase, especialmente la tasa de aprendizaje, con el objetivo de conseguir un mejor resultado para el AP, que significaría una mejora del modelo.
- Mejorar el modelo de detección del balón para que también sea capaz de detectar a los jugadores, haciéndolo de una forma similar a como se ha desarrollado con el balón.
- Tratar de mejorar el modelo de detección del balón para conseguir que siga al balón en situaciones desfavorables como cuando está en el aire o detrás de un jugador y es difícilmente visible en la imagen. Esto podría conseguirse añadiendo un mayor margen a las etiquetas, añadiendo una tolerancia como se ha hecho en el modelo de detección de pases.
- Tratar de unificar ambos modelos, haciendo que la detección del balón y/o de los jugadores sea una información útil para el modelo de detección de pases, mejorando la precisión de este último.
- Hacer pruebas con otros modelos pre-entrenados y nuevas versiones de YOLO que quizá puedan adaptarse de mejor manera a los problemas ya comentados.

ANEXO

En el Anexo se incluye el código utilizado en los capítulos anteriores para obtener los resultados que se han mencionado.

Modelo para la detección de pases

Como se ha indicado anteriormente, el código del modelo para la detección de pases está basado en el del usuario de Kaggle que se indica en esta referencia [69].

No está incluido el código completo.

Definición de parámetros útiles para el modelo

```
IMSIZE = [540, 768]
IMG_SIZE = (540, 768)
modelname = "tf_efficientnet_b0"
use_amp = True
batch_size = 40
n_epochs = 30
num_workers = 2
COSINE = True
init_lr = 2e-4
kernel_type = "{}-{}".format(modelname, IMSIZE[0])

IMG_SOURCE = "img"
BACK_INTERVAL = 20
BACK_INTERVAL_VAL = 1
ERR_TOL = 1
mixup = True
DEBUG = True
```

Figura A-1. Parámetros del modelo

Extracción de los modelos de 'timm'

```
!mkdir -p ../work
!cd ../work && tar xfz ../input/dflfiles/timm.tgz
import sys
sys.path.append('../work/timm/pytorch-image-models')
```

Figura A-2. Extracción de los modelos de timm

Importación de las librerías necesarias

```
import time
import os
import numpy as np
import pandas as pd
import cv2

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.optim import lr_scheduler
from torch.utils.data import DataLoader, Dataset
from torch.utils.data.sampler import SubsetRandomSampler, RandomSample
r, SequentialSampler
from torch.optim.lr_scheduler import CosineAnnealingWarmRestarts, Cosin
eAnnealingLR, ReduceLROnPlateau

import albumentations as A
import albumentations
import matplotlib.pyplot as plt
from tqdm.notebook import tqdm

import torchvision
import glob
import timm
from sklearn.metrics import confusion_matrix
import copy
```

Figura A-3. Librerías necesarias para el modelo

Definición de tolerancias, vídeos para entrenamiento y validación, extracción del número de fotogramas de los vídeos y cambios en el fichero CSV

```
err_tol = {
    'challenge': [ 0.30, 0.40, 0.50, 0.60, 0.70 ],
    'play': [ 0.15, 0.20, 0.25, 0.30, 0.35 ],
    'throwin': [ 0.15, 0.20, 0.25, 0.30, 0.35 ]
}

video_id_split = {
    'val':[
        '3c993bd2_0',
        '3c993bd2_1',
        '35bd9041_0',
        '35bd9041_1',
    ],
    'train':[
        '1606b0e6_0',
        '1606b0e6_1',
        '407c5a9e_1',
        '4ffd5986_0',
        '9a97dae4_1',
        'cfbe2e94_0',
        'cfbe2e94_1',
        'ecf251d4_0',
    ]
}
event_names = ['challenge', 'throwin', 'play']

df = pd.read_csv("../input/dfl-bundesliga-data-shootout/train.csv")
additional_events = []
for arr in df.sort_values(['video_id','time','event','event_attributes']).values:
    if arr[2] in err_tol:
        tol = err_tol[arr[2]][ERR_TOL]/2
        additional_events.append([arr[0], arr[1]-tol, 'start_'+arr[2], arr[3]])
        additional_events.append([arr[0], arr[1]+tol, 'end_'+arr[2], arr[3]])

for arr in df.sort_values(['video_id','time','event','event_attributes']).values:
    if arr[2] in err_tol:
        tol = err_tol[arr[2]][ERR_TOL]/2
        additional_events.append([arr[0], arr[1]-tol*2, 'pre_'+arr[2], arr[3]])
        additional_events.append([arr[0], arr[1]+tol*2, 'post_'+arr[2], arr[3]])

df = pd.concat([df, pd.DataFrame(additional_events, columns=df.columns)])
df = df[~df['event'].isin(event_names)]
df = df.sort_values(['video_id', 'time'])

cap = cv2.VideoCapture("../input/dfl-bundesliga-data-shootout/train/ecf251d4_0.mp4")
fps = cap.get(cv2.CAP_PROP_FPS)
print("fps:", fps)
df["frame"] = df["time"]*fps

df
```

Figura A-4. Tareas de preparación del fichero CSV y del conjunto de datos

Función para extraer fotogramas de los vídeos y guardarlos como imágenes

```
def extract_images(video_path, out_dir):
    video_name = os.path.basename(video_path).split('.')[0]
    cam = cv2.VideoCapture(video_path)
    print(video_path)
    frame_count = 1
    while True:
        successed, img = cam.read()
        if not successed:
            break
        outfile = f'{out_dir}/{video_name}-{frame_count:06}.jpg'
        img = cv2.resize(img, dsize=IMG_SIZE, interpolation=cv2.INTER_AREA)
        cv2.imwrite(outfile, img)
        frame_count += 1

OUT_DIR = "./work/img/"
IN_VIDEOS = glob.glob("../input/dfl-bundesliga-data-shootout/train/*") # video files
```

Figura A-5. Función para extraer fotogramas de los vídeos

Clase para crear la red neuronal utilizando el modelo de la librería 'timm'

```
class net(nn.Module):
    def __init__(self, modelname, out_dim=10, freeze_bn=True):
        super(net, self).__init__()
        self.model = timm.create_model(modelname, pretrained=True)
        self.model.reset_classifier(out_dim)

    def forward(self, x):
        x = self.model(x)
        return x
```

Figura A-6. Clase para crear la red neuronal

Función para el entrenamiento del modelo

```
scaler = torch.cuda.amp.GradScaler(enabled=use_amp)
def train_epoch(loader, optimizer):
    model.train()
    train_loss = []
    bar = tqdm(loader)
    for (data, target, target2, lam, _) in bar:
        data, target, target2, lam = data.to(device), target.to(device).long(), target2.to(device).long(), lam.to(device).flo
at()
        loss_func = criterion
        optimizer.zero_grad()
        with torch.cuda.amp.autocast(enabled=use_amp):
            logits = model(data).squeeze(1)
            if mixup:
                loss = mixup_criterion(criterion, logits, target, target2, lam).mean()
            else:
                loss = loss_func(logits, target)

        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()
        optimizer.zero_grad()

        loss_np = loss.detach().cpu().numpy()
        train_loss.append(loss_np)
        smooth_loss = sum(train_loss[-100:]) / min(len(train_loss), 100)
        bar.set_description('loss: %.5f, smth: %.5f' % (loss_np, smooth_loss))
    return np.mean(train_loss)
```

Figura A-7. Función para el entrenamiento del modelo

Bucle de entrenamiento y validación

```
device = "cuda"
criterion = nn.CrossEntropyLoss()

def mixup_criterion(criterion, pred, y_a, y_b, lam):
    return lam * criterion(pred, y_a) + (1 - lam) * criterion(pred, y_b)

if not DEBUG:
    # setup dataset
    dataset_train = dflDataset(df_train, transform=transforms_train)
    dataset_valid = dflDataset(df_val, transform=transforms_val, test=True)

    # Setup dataloader
    train_loader = torch.utils.data.DataLoader(dataset_train, batch_size=batch_size, sampler=RandomSampler(dataset_train), nu
m_workers=num_workers)
    valid_loader = torch.utils.data.DataLoader(dataset_valid, batch_size=batch_size, sampler=SequentialSampler(dataset_vali
d), num_workers=num_workers)

    # Initialize model
    model = net(modelname)
    model = model.to(device)

    print(len(dataset_train), len(dataset_valid))

    # We use Cosine annealing LR scheduling
    optimizer = optim.Adam(model.parameters(), lr=init_lr)
    scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, n_epochs)

    best_file = f'./models/{kernel_type}_best_fold{fold}.pth'

    best = 0
    for epoch in range(1, n_epochs+1):
        start = time.time()
        torch.cuda.empty_cache()
        print(time.ctime(), 'Epoch:', epoch)

        scheduler.step(epoch-1)

        train_loss = train_epoch(train_loader, optimizer)
        val_loss, acc, score, PREDS, FILES, TARGETS  = val_epoch(valid_loader)

        if score > best:
            torch.save(model.state_dict(), os.path.join(f'models/{kernel_type}_epoch{epoch}_fold{fold}.pth'))
            best = score
```

Figura A-8. Código para el bucle de entrenamiento y validación

Implementación de la red neuronal convolucional 1D

```
STEPS = 25

class cnn(nn.Module):
    def __init__(self):
        super(cnn, self).__init__()
        hdn = 32
        self.fc = nn.Conv1d(10, hdn, 13, bias=False, stride=2)
        self.bn = nn.BatchNorm1d(hdn)
        self.do = nn.Dropout(0.2)
        self.fc2 = nn.Conv1d(hdn, hdn*2, 7, bias=False, stride=2)
        self.bn2 = nn.BatchNorm1d(hdn*2)
        self.do2 = nn.Dropout(0.25)
        self.fc3 = nn.Conv1d(hdn*2, hdn*2, 5, bias=False, stride=1)
        self.bn3 = nn.BatchNorm1d(hdn*2)
        self.do3 = nn.Dropout(0.25)
        self.fc4 = nn.Conv1d(hdn*2, hdn*2, 3, bias=False, stride=1)
        self.bn4 = nn.BatchNorm1d(hdn*2)
        self.do4 = nn.Dropout(0.25)
        self.fc5 = nn.Linear(hdn*2,10)

    def extract(self, x):
        return self.basemodel(x)

    def forward(self, x):
        x = self.do(F.relu(self.bn(self.fc(x))))
        x = self.do2(F.relu(self.bn2(self.fc2(x))))
        x = self.do3(F.relu(self.bn3(self.fc3(x))))
        x = self.do4(F.relu(self.bn4(self.fc4(x))))
        #x = self.do4(F.relu(self.bn4(self.fc4(x))))
        #x = self.fc4(x)
        return self.fc5(F.adaptive_avg_pool1d(x, 1).squeeze(-1))
```

Figura A-9. Red neuronal convolucional 1D

Entrenamiento del modelo de red neuronal convolucional

```
event_names_with_background = ['background','challenge','play','throwin']

scores = []
best = 0
for i in range(30):
    print("epoch ", i)
    train_epoch(train_loader, optimizer)
    if i%5==4:
        _,_,score = val_epoch(valid_loader)
        scores.append(score)
        if score > best:
            best = score
            torch.save(model.state_dict(), os.path.join('./cnn1d_{}_32_psi.pth'.format(STEPS)))
```

Figura A-10. Entrenamiento del modelo

Modelo para la detección del balón

Clonación del repositorio de YOLOv5

```
!git clone https://github.com/ultralytics/yolov5  # clone repo
%cd yolov5
%pip install -qr requirements.txt
```

Figura A-11. Clonación del repositorio YOLOv5

Importación de las librerías necesarias

```
import torch
import cv2
import os
import gc
import numpy as np
import pandas as pd
from tqdm import tqdm
from shutil import copyfile
from sklearn.model_selection import train_test_split
from IPython.core.display import Video, display
import matplotlib.pyplot as plt
import yaml
import subprocess
```

Figura A-12. Librerías necesarias

Elección del tamaño de lote y número de épocas

```
TRAIN_PATH = '/content/dataset/input/'
BATCH_SIZE = 2
EPOCHS = 30

print(f'Number of extra images: {len(os.listdir(TRAIN_PATH))}')
```

Figura A-13. Preparación de los datos para el modelo

Recolección de etiquetas y cajas delimitadoras procedentes del fichero CSV

```
extra_df = pd.read_csv('/content/dataset/input/Ball-Detection-export.csv')
print('Number of ground truth bounding boxes: ', len(extra_df))

label_to_id = {label: i for i, label in enumerate(extra_df.label.unique())}
print('Unique labels: ', label_to_id)

image_bbox_label = {}
for image, df in extra_df.groupby('image'):
    image_bbox_label[image] = df.reset_index(drop=True)

# Visualize
extra_df.head()
```

Figura A-14. Lectura del fichero CSV

División del conjunto de datos en conjunto de entrenamiento y validación

```
train_names, valid_names = train_test_split(list(image_bbox_label), test_size=0.2, random_state=42)
print(f'Size of dataset: {len(image_bbox_label)},\
      training images: {len(train_names)},\
      validation images: {len(valid_names)}')
```

Figura A-15. División del conjunto de datos

Creación del fichero de configuración YAML para entrenar el modelo

```
data_yaml = dict(
    train = '/content/dataset/images/train',
    val = '/content/dataset/images/valid',
    nc = 1,
    names = list(extra_df.label.unique())
)

# Note that the file is created in the yolov5/data/ directory.
with open('/content/yolov5/data/data.yaml', 'w') as outfile:
    yaml.dump(data_yaml, outfile, default_flow_style=True)

%cat /content/yolov5/data/data.yaml
```

Figura A-16. Creación del fichero YAML

Función para ajustar el formato de las cajas delimitadoras al formato de YOLOv5

```
def get_yolo_format_bbox(img_w, img_h, box):
    """
    Convert the bounding boxes in YOLO format.

    Input:
    img_w - Original/Scaled image width
    img_h - Original/Scaled image height
    box - Bounding box coordinates in the format, "left, width, top, height"

    Output:
    Return YOLO formatted bounding box coordinates, "x_center y_center width height".
    """
    w = box.xmax - box.xmin # width
    h = box.ymax - box.ymin # height
    xc = box.xmin + int(np.round(w/2)) # xmin + width/2
    yc = box.ymin + int(np.round(h/2)) # ymin + height/2

    return [xc/img_w, yc/img_h, w/img_w, h/img_h] # x_center y_center width height

# Iterate over each image and write the labels and bbox coordinates to a .txt file.
for img_name, df in tqdm(image_bbox_label.items()):
    # open image file to get the height and width
    img = cv2.imread(TRAIN_PATH+'/'+img_name)
    height, width, _ = img.shape
    # iterate over bounding box df
    bboxes = []
    for i in range(len(df)):
        # get a row
        box = df.loc[i]
        # get bbox in YOLO format
        box = get_yolo_format_bbox(width, height, box)
        bboxes.append(box)

    if img_name in train_names:
        img_name = img_name[:-4]
        file_name = f'/content/dataset/labels/train/{img_name}.txt'
    elif img_name in valid_names:
        img_name = img_name[:-4]
        file_name = f'/content/dataset/labels/valid/{img_name}.txt'

    with open(file_name, 'w') as f:
        for i, bbox in enumerate(bboxes):
            label = label_to_id[df.loc[i].label]
            bbox = [label]+bbox
            bbox = [str(i) for i in bbox]
            bbox = ' '.join(bbox)
            f.write(bbox)
            f.write('\n')
```

Figura A-17. Función para ajusta el formato de las cajas delimitadoras

Entrenamiento del modelo

```
!python train.py --img 1280 \
                 --batch {BATCH_SIZE} \
                 --epochs {EPOCHS} \
                 --data data.yaml \
                 --weights yolov5l6.pt \
                 --project dfl-ball
```

Figura A-18. Entrenamiento del modelo

Detección del balón

```
!python detect.py --weights {best_weights} \
                  --source '/content/dataset/test/0b1495d3_1.mp4' \
                  --img 1280 \
                  --project {project_name}
```

Figura A-19. Detección del balón

REFERENCIAS

[1] Wikipedia, «Inteligencia artificial,» 13 Junio 2024. [En línea]. Available: https://es.wikipedia.org/wiki/Inteligencia_artificial .

[2] Linkedin, «Desde sus inicios: La evolución de la inteligencia artificial,» 16 Mayo 2023. [En línea]. Available: https://es.linkedin.com/pulse/desde-sus-inicios-la-evolución-de-inteligencia-artificial#:~:text=-%20Nils%20J.,cuando%20se%20realizan%20por%20personas.

[3] P. Chauhan, «What Are The Objectives of Artificial Intelligence?,» 24 Abril 2024. [En línea]. Available: https://arramton.com/blogs/what-are-the-objectives-of-artificial-intelligence.

[4] D. Fardian, «glair.ai,» 1 Julio 2022. [En línea]. Available: https://glair.ai/post/5-biggest-limitations-of-artificial-intelligence.

[5] BBC, «bbc.com,» 1 Julio 2015. [En línea]. Available: https://www.bbc.com/news/technology-33347866.

[6] Europa Press, «europapress.es,» 15 Febrero 2023. [En línea]. Available: https://www.europapress.es/portaltic/empresas/noticia-mitad-empresas-aplican-inteligencia-artificial-negocio-mckinsey-20230215121015.html.

[7] G. Granda, «larazon.es,» 6 Noviembre 2019. [En línea]. Available: https://www.larazon.es/actualidad/20191106/bbg2nc24hzg6lbcgast4r3zvqu.html.

[8] T. Roy, «alltechmagazine.com,» 19 Julio 2023. [En línea]. Available: https://alltechmagazine.com/the-evolution-of-ai/.

[9] Wikipedia, «Alan Turing,» [En línea]. Available: https://es.wikipedia.org/wiki/Alan_Turing.

[10] Wikipedia, «Isaac Asimov,» [En línea]. Available: https://es.wikipedia.org/wiki/Isaac_Asimov.

[11] Wikipedia, «Claude Shannon,» [En línea]. Available: https://es.wikipedia.org/wiki/Claude_Shannon.

[12] DeepAI, «Narrow AI,» [En línea]. Available: https://deepai.org/machine-learning-glossary-and-terms/narrow-ai.

[13] Wikipedia, «wikipedia.org,» 13 Junio 2024. [En línea]. Available: https://es.wikipedia.org/wiki/Inteligencia_artificial_general.

[14] Wikipedia, «Estimations of Human Brain Emulation Required Performance,» [En línea]. Available: https://es.wikipedia.org/wiki/Archivo:Estimations_of_Human_Brain_Emulation_Required_Performance.svg.

[1 A. Géron, Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow, O'Reilly, 2019.

5]

[16] Wikipedia, «Aprendizaje profundo,» [En línea]. Available: https://es.wikipedia.org/wiki/Aprendizaje_profundo.

[17] A. Holst, «El liderazgo colaborativo y el papel de la IA,» 15 Octubre 2021 . [En línea]. Available: https://blog.webex.com/es/videoconferencias/el-liderazgo-colaborativo-y-el-papel-de-la-ia/.

[18] Medium, «Supervised Machine Learning,» 15 Agosto 2023. [En línea]. Available: https://medium.com/@qazisaim121/supervised-machine-learning-4e5b355018f6.

[19] Fineproxy, «Aprendizaje no supervisado,» [En línea]. Available: https://fineproxy.org/es/wiki/unsupervised-learning/.

[20] V. Kanade, «What Is Reinforcement Learning? Working, Algorithms, and Uses,» 29 Septiembre 2022. [En línea]. Available: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-reinforcement-learning/.

[21] T. H. K. Tusar, «Batch (Offline) Learning and Incremental (Online) Learning in Machine Learning with Implementation,» 14 Diciembre 2022. [En línea]. Available: https://www.linkedin.com/pulse/batch-offline-learning-incremental-online-machine-tusar-stmieee.

[22] A. Zhang, «towardsdatascience.com,» 16 Agosto 2018. [En línea]. Available: https://towardsdatascience.com/data-types-from-a-machine-learning-perspective-with-examples-111ac679e8bc.

[23] K. Vasanthi, «Tensors in Machine Learning: An In-Depth Exploration,» [En línea]. Available: https://www.kaggle.com/discussions/getting-started/500238.

[24] B. Samanci, «Data Labeling for Machine Learning,» 13 Febrero 2024. [En línea]. Available: https://medium.com/@betulsamancii/what-is-data-labeling-how-to-do-it-05ce22c10b76.

[25] K. Muralidhar, «Outlier detection methods in Machine Learning,» 15 Febrero 2021. [En línea]. Available: https://towardsdatascience.com/outlier-detection-methods-in-machine-learning-1c8b7cca6cb8.

[26] M. Banko, «Scaling to very very large corpora for natural language disambiguation,» 6 Julio 2001. [En línea]. Available: https://dl.acm.org/doi/10.3115/1073012.1073017.

[27] J. Alex, «How sampling bias affected the US Presidential Elections!!,» 30 Abril 2020. [En línea]. Available: https://medium.com/@jithinalex619/how-sampling-bias-affected-the-us-presidential-elections-4ef5ed59d842.

[28] G. Shamir, «Neural Network Construction Practices in Elementary School,» Junio 2021. [En línea]. Available: https://www.researchgate.net/publication/352678116_Neural_Network_Construction_Practices_in_Elementary_School.

[29] B. Díaz, «El superpoder de los hiperparámetros.,» 15 Octubre 2021. [En línea]. Available: https://impulsatek.com/8-el-superpoder-de-los-hiperparametros/.

[30] V. N. Kumbhar, «Bias Variance tradeoff,» [En línea]. Available: https://iq.opengenus.org/bias-variance-tradeoff/.

[3 1] E. Mousavi, «ML Series: Day 10 — Bias Variance Trade-off,» 16 Marzo 2024. [En línea]. Available: https://medium.com/@ebimsv/machine-learning-series-day-10-bias-variance-trade-off-78174d54a378.

[3 2] T. Hastie, R. Tibshirani y J. Friedman, The Elements of Statistical Learning, Springer, 2008.

[3 3] P. Patil, «K Means Clustering : Identifying F.R.I.E.N.D.S in the World of Strangers,» 20 Mayo 2018. [En línea]. Available: https://towardsdatascience.com/k-means-clustering-identifying-f-r-i-e-n-d-s-in-the-world-of-strangers-695537505d.

[3 4] M. Ester, H.-P. Kriegel, J. Sander y X. Xu, «A Density-Based Algorithm for Discovering Clusters,» 2006.

[3 5] Wikipedia, «DBSCAN,» 9 Octubre 2023. [En línea]. Available: https://es.wikipedia.org/wiki/DBSCAN.

[3 6] A. Chugh, «MAE, MSE, RMSE, Coefficient of Determination, Adjusted R Squared — Which Metric is Better?,» 8 Diciembre 2020. [En línea]. Available: https://medium.com/analytics-vidhya/mae-mse-rmse-coefficient-of-determination-adjusted-r-squared-which-metric-is-better-cd0326a5697e.

[3 7] J. Singh, «Stochasticity in Stochastic Gradient Descent (SGD),» 22 Mayo 2022. [En línea]. Available: https://medium.com/@jasraj.singh/what-is-stochastic-in-stochastic-gradient-descent-sgd-20d462aec672.

[3 8] K. Zoumana, «Clasificación en machine learning: Introducción,» Marzo 2024. [En línea]. Available: https://www.datacamp.com/es/blog/classification-machine-learning.

[3 9] Z. Keita, «Classification in Machine Learning: An Introduction,» Septiembre 2022. [En línea]. Available: https://www.datacamp.com/blog/classification-machine-learning.

[4 0] J. Ebner, «Binary Classification, Explained,» 10 Diciembre 2023. [En línea]. Available: https://www.sharpsightlabs.com/blog/binary-classification-explained/.

[4 1] codecademy, «Sigmoid Functions,» 7 Julio 2023. [En línea]. Available: https://www.codecademy.com/resources/docs/ai/neural-networks/sigmoid-activation-function.

[4 2] J. Brownlee, «One-vs-Rest and One-vs-One for Multi-Class Classification,» 27 Abril 2021. [En línea]. Available: https://machinelearningmastery.com/one-vs-rest-and-one-vs-one-for-multi-class-classification/.

[4 3] Wikipedia, «Función Softmax,» Febrero 2024. [En línea]. Available: https://es.wikipedia.org/wiki/Función_SoftMax.

[4 4] Junaid Qadir, «Graphic representation of the softmax activation function,» [En línea]. Available: https://www.researchgate.net/figure/Graphic-representation-of-the-softmax-activation-function_fig5_348703101.

[4 5] S. Narkhede, «Understaning Confusion Matrix,» 9 Mayo 2018. [En línea]. Available: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62.

[4 6] A. Kumar, 18 Noviembre 2023. [En línea]. Available: https://vitalflux.com/ml-metrics-sensitivity-vs-specificity-difference/.

[4 DataBitAI, «Métricas de Evaluación en Machine Learning,» 17 Abril 2023. [En línea]. Available:

7] https://databitai.com/machine-learning/metricas-de-evaluacion-en-machine-learning/.

[4 N. A. Ahmed, «Mean Average Precision (mAP): A Complete Guide,» [En línea]. Available: https://kili-
8] technology.com/data-labeling/machine-learning/mean-average-precision-map-a-complete-guide#mean-average-precision-(map)--definition.

[4 Jaspreet, «A Concise History of Neural Networks,» 14 Agosto 2016. [En línea]. Available:
9] https://towardsdatascience.com/a-concise-history-of-neural-networks-2070655d3fec.

[5 Wikipedia, «History of artificial neural networks,» 12 Junio 2024. [En línea]. Available:
0] https://en.wikipedia.org/wiki/History_of_artificial_neural_networks.

[5 ResearchGate, «McCulloch and Pitts Neuron Model,» [En línea]. Available:
1] https://www.researchgate.net/figure/McCulloch-and-Pitts-Neuron-Model-13_fig1_356858632.

[5 Wikipedia, «Archivo:XOR perceptron net.png,» [En línea]. Available:
2] https://es.m.wikipedia.org/wiki/Archivo:XOR_perceptron_net.png.

[5 D. J. Matich, «Redes Neuronales: Conceptos Básicos y,» [En línea]. Available:
3] https://www.frro.utn.edu.ar/repositorio/catedras/quimica/5_anio/orientadora1/monograias/matich-redesneuronales.pdf.

[5 Research Gate, «Perceptron image,» [En línea]. Available: https://www.researchgate.net/figure/General-
4] scheme-of-an-artificial-neural-network-ANN_fig12_352338692.

[5 Wikipedia, «Perceptrón multicapa,» 7 Mayo 2024. [En línea]. Available:
5] https://es.wikipedia.org/wiki/Perceptrón_multicapa.

[5 Wikipedia, «Redes neuronales recurrentes,» 18 Abril 2024. [En línea]. Available:
6] https://es.wikipedia.org/wiki/Redes_neuronales_recurrentes#:~:text=Las%20redes%20neuronales%20recurrentes%20(RNNs,música%20y%20las%20series%20temporales.

[5 J. Torres, «Redes Neuronales Recurrentes,» 22 Septiembre 2019. [En línea]. Available:
7] https://torres.ai/redes-neuronales-recurrentes/.

[5 O. Calzone, «An Intuitive Explanation of LSTM,» 21 Febrero 2022. [En línea]. Available:
8] https://medium.com/@ottaviocalzone/an-intuitive-explanation-of-lstm-a035eb6ab42c.

[5 Wikipedia, «Red generativa adversativa,» 15 Julio 2023. [En línea]. Available:
9] https://es.wikipedia.org/wiki/Red_generativa_adversativa.

[6 Linkedin, «What Are Generative Adversarial Networks (GANs)? Understanding and Implications,» 12
0] Enero 2024. [En línea]. Available: https://www.linkedin.com/pulse/what-generative-adversarial-networks-gans-sushant-babbar-qpc9c.

[6 M. Mishra, «Convolutional Neural Networks, Explained,» 26 Agosto 2020. [En línea]. Available:
1] https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939.

[6 d2l, «Padding and Stride,» [En línea]. Available: https://d2l.ai/chapter_convolutional-neural-
2] networks/padding-and-strides.html.

[6 D. Kalita, «Basics of CNN in Deep Learning,» 19 Abril 2024. [En línea]. Available:

3] https://www.analyticsvidhya.com/blog/2022/03/basics-of-cnn-in-deep-learning/.

[6 J. Cuartas, «Convolutional Neural Networks (CNN) o redes convolucionadas,» 30 Junio 2021. [En línea].
4] Available: https://josecuartas.medium.com/el-concepto-de-la-convolución-en-gráficos-para-comprender-las-convolutional-neural-networks-cnn-519d2eee009c.

[6 Research Gate, «ReLU function,» [En línea]. Available: https://www.researchgate.net/figure/ReLU-
5] activation-function_fig3_319235847.

[6 DataScientest, 6 Enero 2024. [En línea]. Available: https://datascientest.com/es/que-es-el-transfer-learning.
6]

[6 ViewNext, 19 Agosto 2020. [En línea]. Available: https://www.viewnext.com/transfer-learning-y-redes-
7] convolucionales/.

[6 M. Tan y Q. V. Le, «EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,» 28 Mayo
8] 2019. [En línea]. Available: https://arxiv.org/abs/1905.11946.

[6 ImageNet, «ImageNet,» [En línea]. Available: https://www.image-net.org.
9]

[7 P. Potimba, «What is EfficientNet? The Ultimate Guide.,» 9 Agosto 2023. [En línea]. Available:
0] https://blog.roboflow.com/what-is-efficientnet/.

[7 SKY ENGINE AI, «What is EfficientNet?,» [En línea]. Available: https://skyengine.ai/se/skyengine-
1] blog/121-what-is-efficientnet.

[7 Wikipedia, «Red Neuronal Residual,» [En línea]. Available:
2] https://es.wikipedia.org/wiki/Red_neuronal_residual.

[7 A. D. Serej, «ResNet-50,» 23 Diciembre 2022. [En línea]. Available:
3] https://medium.com/@arashserej/resnet-50-83b3ff33be7d.

[7 MathWorks, «resnet50,» [En línea]. Available: ResNet-50.
4]

[7 Wikipedia, «Transformador de visión,» [En línea]. Available:
5] https://es.wikipedia.org/wiki/Transformador_de_visión.

[7 rwightman, «Vision Transformer,» 14 Febrero 2021. [En línea]. Available:
6] https://paperswithcode.com/lib/timm/vision-transformer.

[7 rwightman, «dpn68,» 14 Febrero 2021. [En línea]. Available:
7] https://paperswithcode.com/model/dpn?variant=dpn68.

[7 Kaggle, «DFL - Bundesliga Data Shootout,» 29 Julio 2022. [En línea]. Available:
8] https://www.researchgate.net/figure/ReLU-activation-function_fig3_319235847.

[7 arutema47, «Train DFL Effnet+1DCNN [CV0.77],» [En línea]. Available:
9] https://www.kaggle.com/code/kyoshioka47/train-dfl-effnet-1dcnn-cv0-77/notebook#Prepare-model,-dataset.

[80] Pandas Pydata, «Pandas,» [En línea]. Available: https://pandas.pydata.org.

[81] OpenCV, «OpenCV,» [En línea]. Available: https://opencv.org.

[82] J. Durán, «Guía Rápida sobre Preprocesamiento de Imágenes Faciales para Redes Neuronales usando OpenCV en Python,» 2 Julio 2019. [En línea]. Available: https://medium.com/metadatos/guía-rápida-sobre-preprocesamiento-de-imágenes-faciales-para-redes-neuronales-usando-opencv-en-c68599ca08dd.

[83] R. Wightman, «GitHub,» [En línea]. Available: https://github.com/huggingface/pytorch-image-models.

[84] Microsoft VOTT, «GitHub Microsoft VOTT,» [En línea]. Available: https://github.com/microsoft/VoTT.

[85] G. Jocher, «GitHub yolov5,» [En línea]. Available: https://github.com/ultralytics/yolov5.

[86] A. Bochkovskiy, C.-Y. Wang y H.-Y. M. Liao, «YOLOv4: Optimal Speed and Accuracy of Object Detection,» 23 Abril 2020. [En línea]. Available: https://arxiv.org/pdf/2004.10934.

[87] M. Tan, R. Pang y Q. V. Le, «EfficientDet: Scalable and Efficient Object Detection,» 27 Julio 2020. [En línea]. Available: https://arxiv.org/pdf/1911.09070.

[88] Wikipedia, «Algoritmo You Only Look Once (YOLO),» 1 Mayo 2024. [En línea]. Available: https://es.wikipedia.org/wiki/Algoritmo_You_Only_Look_Once_(YOLO).

[89] A. Perera, «What is Padding in CNN's,» 2 Septiembre 2018. [En línea]. Available: https://ayeshmanthaperera.medium.com/what-is-padding-in-cnns-71b21fb0dd7.

[90] ResearchGate, «Función de transferencia empleada para entrenar la red neuronal,» [En línea]. Available: https://www.researchgate.net/figure/Funcion-de-transferencia-empleada-para-entrenar-la-red-neuronal-9_fig5_286460225.