



Escuela Técnica Superior de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería de Computadores

Detección de casos Covid-19 mediante Deep Learning

Autor/es:

Joaquín Almarcho Conejero

Tutor/es:

Juan Antonio Nepomuceno Chamorro

Departamento:

Lenguajes y Sistemas Informáticos

Primera convocatoria

Curso 2021/2022

AGRADECIMIENTOS

Transmitir mis más sinceros agradecimientos a todas aquellas personas que me han ayudado en esta etapa.

En primer lugar, dar las gracias a mi tutor Juan Antonio Nepomuceno, por su ayuda en la planificación, organización e información para este trabajo fin de grado.

En segundo lugar, familiares y amigos que han estado apoyándome a lo largo de mi carrera profesional y personal.

Desarrollar y exponer este proyecto marca el fin de mi etapa de grado, dando lugar al comienzo de otra etapa para seguir creciendo.

A todos ellos, gracias.

ÍNDICE GENERAL

| | |
|--|----|
| Agradecimientos | 2 |
| Índice general | 3 |
| Índice de tablas | 4 |
| Índice de figuras | 5 |
| Índice de código | 6 |
| Resumen..... | 7 |
| Capítulo 1: Planificación | 8 |
| Objetivos | 9 |
| Planificación temporal | 10 |
| Costes | 14 |
| Capítulo 2: Introducción..... | 16 |
| Aprendizaje automático | 16 |
| Aprendizaje profundo o Deep learning..... | 18 |
| Librerías: TensorFlow y Keras..... | 31 |
| Diagnostico mediante radiografías pulmonares..... | 32 |
| Capítulo 3: Estudio práctico del estado del arte | 35 |
| Estudio previo | 35 |
| Estudio práctico: 1º caso MNIST | 36 |
| Estudio práctico: 2º caso Cats vs Dogs..... | 40 |
| Capítulo 4: Diseño experimental..... | 44 |
| Decisiones de tipo técnico | 45 |
| Procesamiento de los datos | 47 |
| Generación del modelo..... | 49 |
| Capítulo 5: Resultados y discusión | 52 |
| Capítulo 6: Conclusiones y líneas futuras..... | 55 |
| Capítulo 7: Bibliografía | 56 |
| Anexo | 58 |

ÍNDICE DE TABLAS

| | |
|---|----|
| Tabla 1: Planificación temporal del proyecto..... | 10 |
| Tabla 2: Planificación temporal Fase 1..... | 10 |
| Tabla 3: Planificación temporal Fase 2..... | 11 |
| Tabla 4: Planificación temporal Fase 3..... | 11 |
| Tabla 5: Planificación temporal Fase 4..... | 12 |
| Tabla 6: Planificación temporal Fase 5..... | 12 |
| Tabla 7: Costes analista programador..... | 14 |
| Tabla 8: Costes investigador | 14 |
| Tabla 9: Costes aplicaciones informáticas..... | 15 |
| Tabla 10: Tipos de funciones de activación. [8] | 26 |

ÍNDICE DE FIGURAS

| | |
|---|----|
| Figura 1: Diagrama de Gantt del Trabajo de Fin de Grado. | 13 |
| Figura 2: Tipos de aprendizaje automático..... | 17 |
| Figura 3: Inteligencia Artificial, Aprendizaje Automático y Aprendizaje Profundo..... | 18 |
| Figura 4: Esquema de un perceptrón simple. | 19 |
| Figura 5: Flujo de ejecución de una red neuronal..... | 19 |
| Figura 6: Estructura básica de una CNN. | 20 |
| Figura 7: Funcionamiento de una capa convolucional..... | 21 |
| Figura 8: Ejemplo de detección de una capa convolucional [4]..... | 22 |
| Figura 9: Ejemplo de capa densa..... | 23 |
| Figura 10: Funcionamiento de una capa de Max Pooling. | 24 |
| Figura 11: Funcionamiento de una capa de Dropout. | 24 |
| Figura 12: Funcionamiento de una capa de Flatten..... | 25 |
| Figura 13: Ejemplo de división de los datos. | 27 |
| Figura 14: Ejemplo de epoch, batch size e iteraciones. | 28 |
| Figura 15: Tipos de ajuste de un modelo. | 29 |
| Figura 16: Funcionamiento de la técnica data augmentation. | 29 |
| Figura 17: Funcionamiento de la técnica early stopping. | 30 |
| Figura 18: Esquema del modelo generado para el conjunto MNIST. | 37 |
| Figura 19: Resultado del entrenamiento del modelo. | 38 |
| Figura 20: Resultado de la clasificación realizada por el modelo. | 39 |
| Figura 21: Esquema del modelo generado para el conjunto Cats vs Dogs. | 41 |
| Figura 22: Resultado del entrenamiento del modelo. | 42 |
| Figura 23: Resultado de la clasificación realizada por el modelo | 43 |
| Figura 24: Flujo de trabajo del capítulo 4..... | 44 |
| Figura 25: Ejemplo de imágenes clasificadas por categoría..... | 46 |
| Figura 26: División de los datos en los conjuntos de entrenamiento, validación y test. | 47 |
| Figura 27: Esquema del modelo generado para el conjunto COVID-19 Radiography Database..... | 50 |
| Figura 28: Resultado del entrenamiento del modelo. | 52 |
| Figura 29: Resultado del proceso de evaluación del modelo. | 53 |
| Figura 30: Resultado de la clasificación realizada por el modelo | 53 |

ÍNDICE DE CÓDIGO

| | |
|---|----|
| Código 1: División de los datos en X_train, Y_train y X_test..... | 36 |
| Código 2: Entrenamiento del modelo generado..... | 38 |
| Código 3: Ejemplo de uso de la técnica data augmentation..... | 40 |
| Código 4: Entrenamiento del modelo generado..... | 42 |
| Código 5: Proceso de normalización realizado con la función ImageDataGenerator..... | 48 |
| Código 6: Fuente: Elaboración propia..... | 48 |
| Código 7: Creación de directorios temporales..... | 58 |
| Código 8: Creación del modelo | 58 |
| Código 9: Entrenamiento del modelo | 59 |
| Código 10: Predicción y visualización de las imágenes | 59 |

RESUMEN

La pandemia por el virus SARS-CoV-2 ha desencadenado una crisis económica y sanitaria sin precedentes. Aunque el diagnóstico es microbiológico, las técnicas de imagen tienen un papel importante para apoyar el diagnóstico, graduar la gravedad de la enfermedad, guiar el tratamiento, detectar posibles complicaciones y valorar la respuesta terapéutica. La afectación es principalmente pulmonar.

El objetivo principal del presente proyecto se basa en la introducción al aprendizaje en el ámbito *del Deep Learning*, es decir, mediante diferentes cursos e implementación de algoritmos, se pretende llevar a cabo su práctica y desarrollo.

Con el fin de poder alcanzar los objetivos planteados del proyecto, se han realizado unos estudios que pretenden facilitar la ejecución de este.

Previamente, se ha llevado a cabo la realización y el aprendizaje de varios cursos que facilita la plataforma *Kaggle*, acerca de Machine Learning y Deep Learning, posteriormente, se pondrán en práctica los conocimientos adquiridos de estos cursos con los conjuntos de datos de MNIST y Cats vs Dogs.

Los estudios finales consistirán en la construcción de un clasificador de imágenes a partir de una base de datos publicada en el entorno Kaggle, se trata de un dataset de imágenes de rayos-x pulmonares.

Se discutirán las conclusiones logradas, el código implementado dará lugar a la posible creación de una herramienta analítica y de clasificación de imágenes pulmonares.

Por último, este proyecto puede dar lugar a futuras líneas de investigación con el fin de detectar otras patologías a partir de los modelos COVID ya entrenados.

Palabras claves

COVID-19, Deep Learning, imágenes de rayos-x, redes neuronales convolucionales, radiografías pulmonares

CAPÍTULO 1: PLANIFICACIÓN

La enfermedad causada por el COVID-19 apareció por primera vez en la ciudad de Wuhan (China) extendiéndose días más tarde por todo el mundo. Esta patología dejó gran número de fallecidos y pacientes con episodios de neumonías bastante graves.

La evolución grave de la enfermedad se caracteriza por la aparición de una neumonía característica, lo cual hace más fácil su detección, por ello es importante identificar la enfermedad en el menos tiempo posible.

Para su detección temprana o diagnóstico preciso, números sanitarios utilizan las radiografías de tórax, permitiendo evaluar el estado pulmonar del paciente.

El aprendizaje profundo, ha logrado demostrar, a lo largo de la historia, unos resultados muy favorables en su aplicación, lo que genera su uso en el ámbito sanitario, permitiéndole a los facultativos diversas herramientas que aceleren el proceso de diagnóstico, y por tanto una detección temprana. Gracias a la capacidad de aprendizaje de características, el Deep Learning es capaz de extraer las características relacionadas con los resultados clínicos de las radiografías de tórax de manera automática.

En este capítulo, se abordarán los objetivos del proyecto enfocados en su aplicación en el ámbito sanitario, así como la planificación detallada que se ha desarrollado y el coste para poder implementar este trabajo fin de grado.

OBJETIVOS

Objetivos docentes

Objetivo 1: Poner en práctica técnicas de planificación y gestión de proyectos informáticos.

Objetivo 2: Adquirir competencias docentes sobre la técnica de Aprendizaje Automático.

- Realización de cursos en *Kaggle*; Introducción al Aprendizaje Automático, Aprendizaje Automático intermedio.

Objetivo 3: Adquirir conocimientos técnicos sobre la metodología del Aprendizaje Profundo

- Realización de cursos en *Kaggle*; Introducción al Aprendizaje Profundo .

Objetivos técnicos

- **Objetivo 1:** Realizar un estudio de las librería Keras y de las redes convolucionales
- **Objetivo 2:** Implementar un clasificador de imágenes haciendo uso del *dataset* MNIST en *Kaggle*.
- **Objetivo 3:** Implementar un clasificador de imágenes de “Perros y Gatos” en *Kaggle*.
- **Objetivo 4:** Participar en una competición real de Kaggle e implementar un clasificador de imágenes pulmonares para la detección del COVID-19, la neumonía viral y la opacidad torácica en radiografías de tórax.

PLANIFICACIÓN TEMPORAL

En el presente apartado, se expone la planificación temporal del proyecto. En la tabla 1, se muestran las fases planteadas para la realización del proyecto junto con la fecha de inicio y fin de cada una. Además, se muestra una descripción de las tareas a realizar en cada fase y se adjunta un diagrama de Gantt que refleja el flujo de trabajo.

| Fase | Inicio | Fin |
|--|------------|------------|
| Fase 1: Planificación | 01/03/2022 | 04/03/2022 |
| Fase 2: Estudio teórico | 07/03/2022 | 18/03/2022 |
| Fase 3: Estudio práctico | 21/03/2022 | 08/04/2022 |
| Fase 4: Diseño experimental | 21/04/2022 | 29/04/2022 |
| Fase 5: Redacción de la memoria y finalización del proyecto | 09/05/2022 | 10/06/2022 |

*Tabla 1: Planificación temporal del proyecto.
Fuente: Elaboración propia*

A continuación, se describen las tareas a desarrollar en las distintas fases.

| Fase 1: Planificación | |
|--|---|
| T1: Descripción y planificación temporal del proyecto. | Descripción del proyecto y organización. Diagrama de Gantt |
| T2: Estimación del coste del proyecto. | |
| Resultado: Plan de proyecto | |

*Tabla 2: Planificación temporal Fase 1.
Fuente: Elaboración propia*

| Fase 2: Estudio teórico | |
|--|--|
| T1: Introducir el contexto del problema. | <p>Introducción al problema y revisión bibliográfica de artículos de aprendizaje automático y aprendizaje profundo aplicado al sector sanitario.</p> <p>Revisión de artículos del estado del arte de las en redes neuronales convolucionales y lectura del libro <i>Deep Learning with Python</i> – François Chollet</p> |
| T2: Estudio del estado del arte del Aprendizaje Automático | |
| T3: Estudio del estado del arte del Aprendizaje Profundo | |
| T4: Estudio del estado del arte del COVID y la imagen médica (radiografías de tórax) | |
| Resultado: Situación en el marco de trabajo | |

Tabla 3: Planificación temporal Fase 2.

Fuente: Elaboración propia

| Fase 3: Estudio práctico | |
|--|---|
| T1: Realización de los cursos de Kaggle. | Realización de un estudio previo basado en la realización de cursos en Kaggle. Realización de estudios prácticos con los bancos de imágenes <i>MNIST</i> y <i>Cats vs Dogs</i> |
| T2: Realización e implementación del primer caso práctico: MNIST | |
| T3: Realización e implementación del primer caso práctico: Cats vs Dogs | |
| Resultado: Notebooks elaborados en Jupyter notebook con el código fuente de los ejemplos. | |

Tabla 4: Planificación temporal Fase 3.

Fuente: Elaboración propia

| Fase 4: Diseño experimental | |
|--|---|
| T1: Introducción a la competición y toma de decisiones técnicas | Introducción a la competición y elección del entorno y las librerías a utilizar. Preprocesamiento de los datos y creación de los conjuntos de entrenamiento, validación y test. Generación, entrenamiento, validación y test del modelo |
| T2: Preprocesamiento de los datos | |
| T3: Generación del modelo | |
| Resultado: Notebooks elaborados en Jupyter notebook con el código fuente del modelo generado y sus explicaciones correspondientes. Resultados estadísticos del proceso realizado. | |

Tabla 5: Planificación temporal Fase 4.

Fuente: Elaboración propia

| Fase 5: Redacción de la memoria y finalización del proyecto | |
|---|--|
| T1: Estructura de la memoria | Organización, estructuración, y redacción del documento final del Trabajo de Fin de Grado. |
| T2: Preparación del material necesario | |
| T3: Redacción de la memoria | |
| T4: Corrección de la memoria | |
| Resultado: Memoria final del Trabajo de Fin de Grado | |

Tabla 6: Planificación temporal Fase 5.

Fuente: Elaboración propia

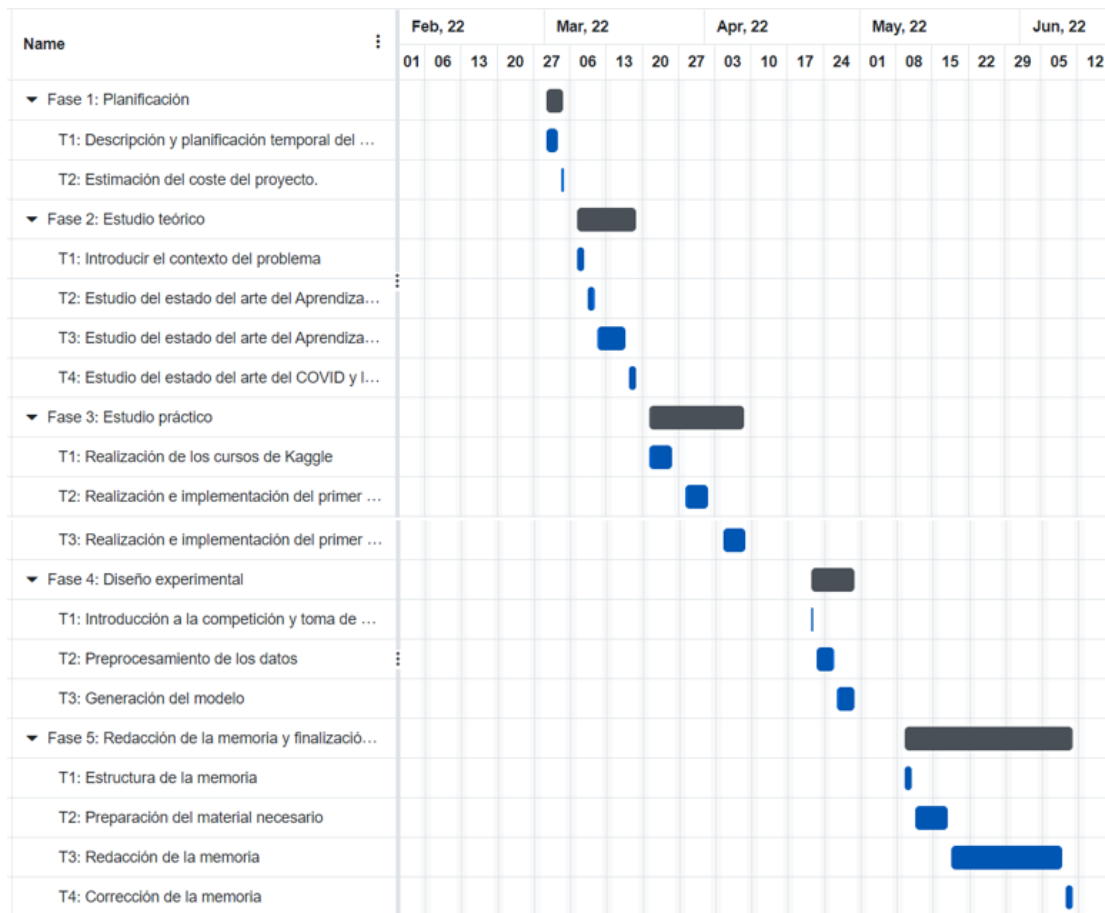


Figura 1: Diagrama de Gantt del Trabajo de Fin de Grado.

Fuente: Elaboración propia

COSTES

En el presente apartado, se presentan los costes del proyecto a realizar. Para su consecución, será necesario la actuación de dos roles: Analista programador e investigador.

Además, se requiere de aplicaciones informáticas para su elaboración. A continuación, se muestran los costes estimados del proyecto.

| Analista programador | |
|--|---|
| Sueldo (neto/mes): $29.600 / 12 = 2.467 \text{ €}$ | Función: El rol de analista programador se encargará de la planificación general, además de llevar a cabo el estudio práctico y diseño experimental del proyecto. Para finalizar su tarea, proveerá el material necesario para la realización de la memoria final. |
| Impuestos <ul style="list-style-type: none">• Cotización por contingencias comunes: 23,60%• Cotización por formación: 0,60%• Cotización por desempleo (contratos indefinidos): 5,50%• Cotización al Fondo de Garantía Salarial (FOGASA): 0,20% Total: 29.9% | |
| Costes sueldo (bruto/mes): $2.467 \text{ €} * 1,299 = 3.204,63 \text{ €}$ | |
| Contrato laboral: 150 horas | |
| Coste del contrato: $(3.204,63 \text{ €} / (4*40 \text{ h})) * 150 \text{ h} = 3.004,3 \text{ €}$ | |

Tabla 7: Costes analista programador

| Investigador | |
|--|--|
| Sueldo (neto/mes): $22.421 / 12 = 1.868,4 \text{ €}$ | Función: El rol de investigador se encargará de llevar a cabo el estudio teórico, así como la redacción de la memoria. |
| Impuestos <ul style="list-style-type: none">• Cotización por contingencias comunes: 23,60%• Cotización por formación: 0,60%• Cotización por desempleo (contratos indefinidos): 5,50%• Cotización al Fondo de Garantía Salarial (FOGASA): 0,20% Total: 29.9% | |
| Costes sueldo (bruto/mes): $1.868,4 \text{ €} * 1,299 = 2.427 \text{ €}$ | |
| Contrato laboral: 150 horas | |
| Coste del contrato: $(2.427 \text{ €} / (4*40 \text{ h})) * 150 \text{ h} = 2.275,3 \text{ €}$ | |

Tabla 8: Costes investigador

| Aplicaciones informáticas | |
|--|--|
| Costes Software: 0 € | El software requerido para la consecución del proyecto es gratuito o dispone de licencia <i>open source</i> . |
| Costes Hardware: 899 € / 4*12 meses = 18,73 €. | Se requiere de computador para el desarrollo del proyecto. Características del PC: Intel(R) Core (TM) i5-8250U, 12GB RAM, 512 GB SSD, NVIDIA GeForce MX130. Modelo 2018 |
| Coste de las aplicaciones informáticas: 18,73 € | |

Tabla 9: Costes aplicaciones informáticas

El coste total del proyecto se cifra en 3.004,3 € + 2.275,3 € + 18,73 € = **5.298,33 €**

CAPÍTULO 2: INTRODUCCIÓN

En este capítulo se exponen los fundamentos informáticos y clínicos que se utilizarán en el desarrollo del proyecto.

A continuación, se analizarán diferentes conceptos teóricos que serán necesarios para el cumplimiento del objetivo del presente proyecto.

APRENDIZAJE AUTOMÁTICO

La combinación de diversos algoritmos matemáticos e informáticos para llevar a cabo soluciones a problemas complejos aplicando el razonamiento inteligente humano, dan lugar al concepto de “Inteligencia Artificial” (IA). [1]

Un sistema basado en la IA es capaz de analizar gran cantidad de datos, identificar patrones y tendencias y formular predicciones de forma automática, a gran velocidad y de alta precisión.

Las aplicaciones más frecuentes de la inteligencia artificial convergen en el campo de la robótica, procesamiento y análisis de imágenes o incluso el tratamiento automático avanzado de textos

La inteligencia artificial engloba diversas áreas de aplicación tales como el procesamiento de lenguaje natural, la robótica o el aprendizaje automático entre otras.

Aunque la IA probó ser muy útil para la resolución de sistemas complejos, más tarde se entendió que no es suficiente para la resolución de grandes escalas, es decir, procesos donde no hay reglas definidas o procesamientos aprendidos de otras experiencias, de esta manera nace un nuevo concepto capaz de ir más allá de la definición que hasta entonces se tenía de la Inteligencia Artificial: *Machine Learning o aprendizaje automático*.

El aprendizaje automático es una disciplina perteneciente a la Inteligencia artificial que se centra en el desarrollo de algoritmos que permiten a las computadoras generalizar comportamientos y reconocer patrones a partir de los datos suministrados.

El resultado de esta técnica es la obtención de una máquina capaz de aprender automáticamente, entendiendo por aprendizaje aquello que la máquina puede aprender a partir de la experiencia obtenida de los datos y no a partir de los patrones programados a priori, para posteriormente aplicar el conocimiento obtenido para realizar predicciones o clasificaciones.

Existen distintas categorías dependiendo de los datos de entrada y la salida principales dentro del aprendizaje automático: [2]

- Aprendizaje supervisado
- Aprendizaje no supervisado
- Aprendizaje semisupervisado
- Aprendizaje de refuerzo

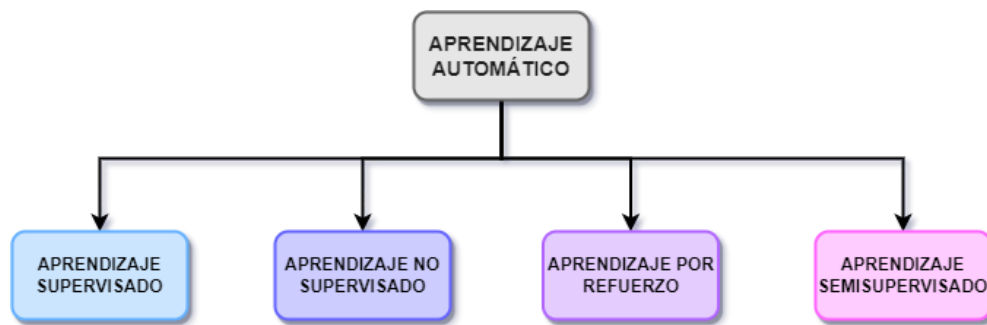


Figura 2: Tipos de aprendizaje automático

Fuente: Elaboración propia

Aprendizaje supervisado

El aprendizaje supervisado se caracteriza por el uso de datos etiquetados. Estos datos poseen una correspondencia entre la entrada y la salida esperada. A lo largo del periodo de entrenamiento, el algoritmo realiza ajustes en su modelo.

Existen distintos problemas que pueden ser resueltos con el aprendizaje supervisado entre los que destacan la clasificación y la regresión. La clasificación intenta predecir la clase a la que pertenece un objeto dentro de una lista de clases prefijadas mientras que la regresión busca predecir un valor numérico. [3]

Aprendizaje no supervisado

El aprendizaje no supervisado se lleva a cabo sobre un conjunto de datos no etiquetados y en los que no se tiene información de la salida esperada. Por lo tanto, es necesario que los algoritmos sean capaces de reconocer patrones para poder etiquetar o agrupar las entidades por afinidad. [3]

Aprendizaje semisupervisado

El aprendizaje semisupervisado ofrece un punto intermedio entre el aprendizaje supervisado y el no supervisado. Durante el periodo de entrenamiento, se hace uso de un conjunto de datos etiquetados y datos no etiquetados para la clasificación. [3]

Aprendizaje de refuerzo

El aprendizaje por refuerzo se centra en recompensar los comportamientos deseados y penalizar los no deseados. Este entrenamiento es logrado a través del método prueba y error. [3]

APRENDIZAJE PROFUNDO O DEEP LEARNING

El aprendizaje profundo es un subconjunto del aprendizaje automático inspirado en el funcionamiento del cerebro humano. Se caracteriza por la utilización de redes neuronales artificiales, un conjunto de elementos interconectados o nodos de procesamiento simple que simulan la actuación de las redes neuronales del cerebro. Estas redes neuronales son capaces de aprender a identificar patrones y clasificar datos de manera similar a como lo haría el cerebro.

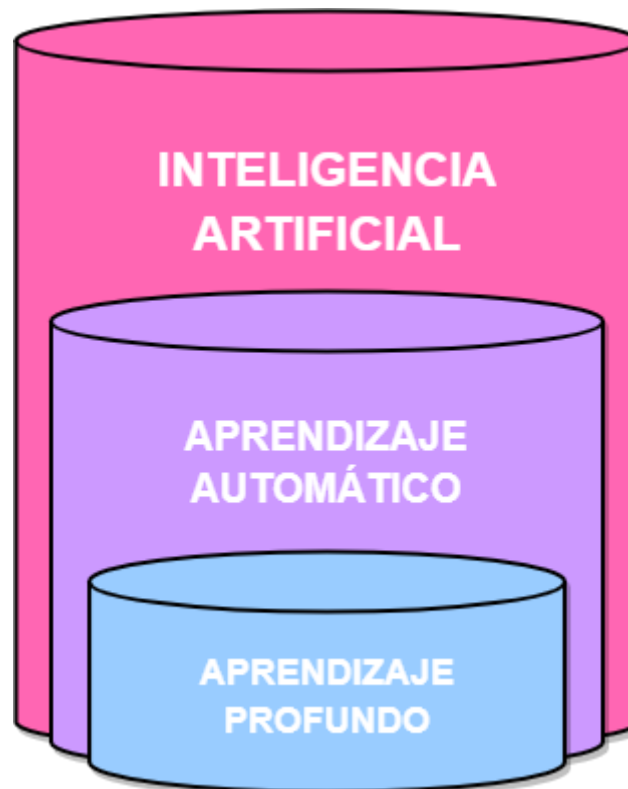


Figura 3: Inteligencia Artificial, Aprendizaje Automático y Aprendizaje Profundo
Fuente: Elaboración propia

El cerebro está formado por millones de neuronas que permanecen interconectadas formando una red. Las neuronas están compuestas por las dendritas, un cuerpo celular o soma y el axón.

La función de cada una de las neuronas es de recibir la información a través de sus dendritas, procesarla en el núcleo y enviarla al resto a través de su axón, donde ocurre el intercambio de información o sinapsis con las dendritas de otras neuronas. Este intercambio se realiza a través de señales químicas y eléctricas.

Las redes neuronales artificiales tratan de replicar el modelo biológico del cerebro. Están formadas fundamentalmente por elementos de procesamiento simple o neuronas que permanecen interconectadas. [4]

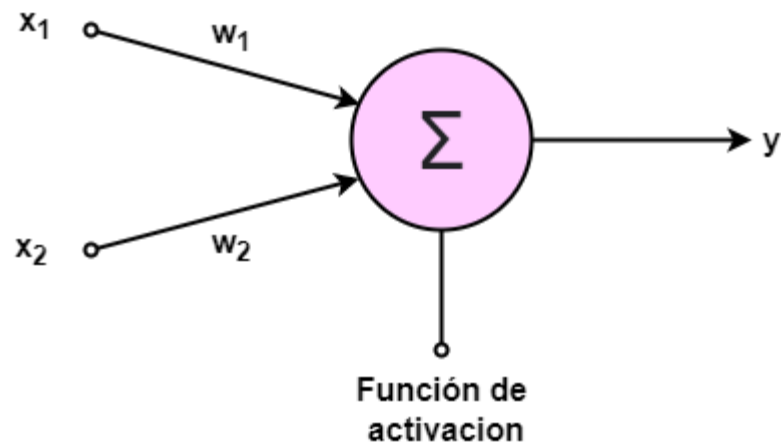


Figura 4: Esquema de un perceptrón simple.
Fuente: Elaboración propia

Las neuronas disponen de entradas o conexiones por las que reciben la información. Cada entrada dispone de un peso asociado que es utilizado para ponderar cada una de las señales que transcurran a través de ellas. Posteriormente, estas ponderaciones son sumadas y emitidas en forma de señal a través de una función de activación f . [4]

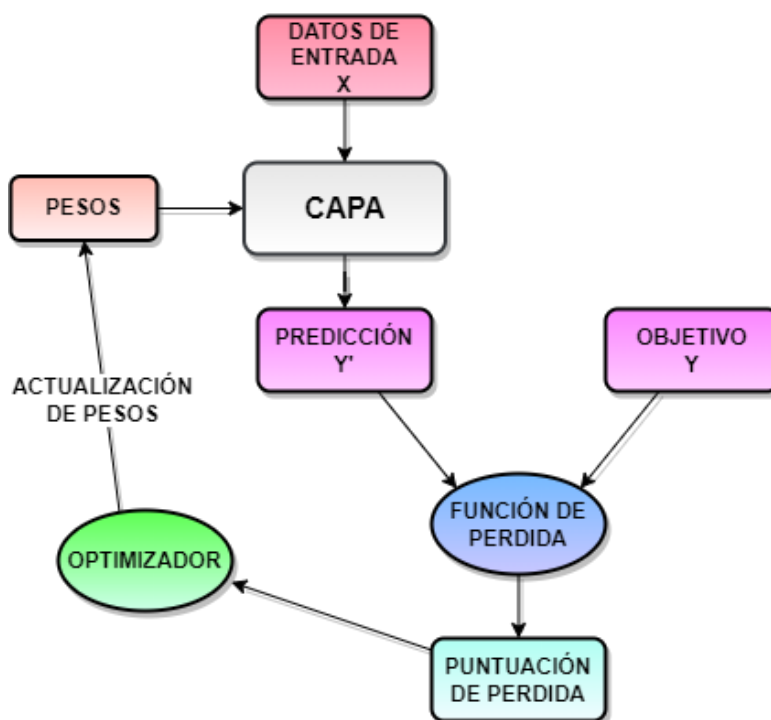


Figura 5: Flujo de ejecución de una red neuronal.
Fuente: Elaboración propia

Las redes neuronales artificiales se caracterizan principalmente por su topología en forma de red, la función de activación y el algoritmo de entrenamiento elegido.

Existen distintas arquitecturas de redes neuronales, entre las que destacan el perceptrón simple, el perceptrón multicapas o las redes convolucionales.

Una red neuronal convolucional o CNN es un tipo de red neuronal que imita el funcionamiento del córtex visual del cerebro para identificar las distintas características de los datos.

La arquitectura clásica de una CNN dedicada al reconocimiento y clasificación de imágenes se divide en dos partes: La etapa convolucional y la etapa de clasificación. [4]

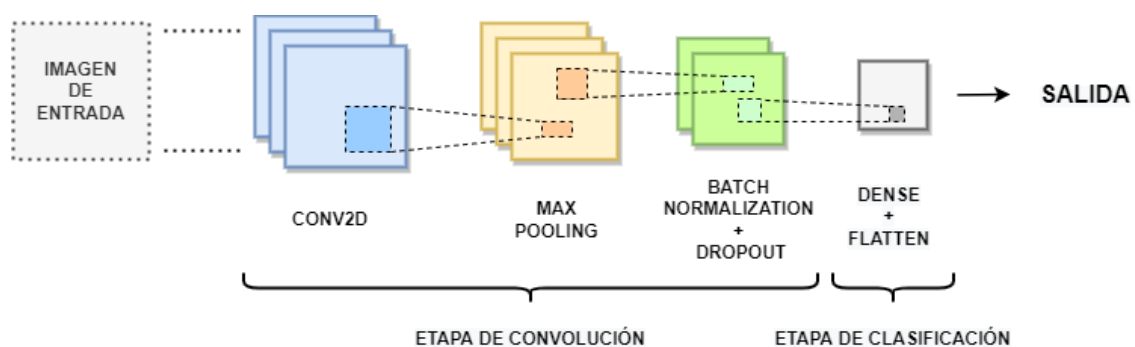


Figura 6: Estructura básica de una CNN.

Fuente: Elaboración propia

La etapa convolucional está formada por una jerarquía de capas que se caracteriza por la capacidad de detección y aprendizaje. Las primeras capas, se encargan de distinguir características básicas de la imagen, como son las líneas y curvas que la componen y transmitir el conocimiento a capas posteriores permitiendo a las capas más "profundas" detectar y diferenciar objetos más complejos tales como personas o animales.

Esta característica permite a la red especializarse en el procesamiento de datos cuadrículados, como son las imágenes digitales.

Una imagen digital es una representación bidimensional de una imagen a través de una matriz numérica. La resolución indica la densidad de píxeles que posee la imagen y permite medir la calidad y la nitidez de estas. Dependiendo de si la resolución de una imagen es estática o dinámica, esta puede ser un gráfico matricial o un gráfico vectorial. [5]

La jerarquía de capas de la red neuronal convolucional [5] está formada principalmente 3 partes: una capa de entrada, encargada de recibir los datos; una o varias capas ocultas, encargadas de procesar los datos; y una capa de salida, encargada de extraer los resultados obtenidos.

A continuación, se muestra el funcionamiento y las aplicaciones de las capas más comunes en la construcción de un modelo CNN.

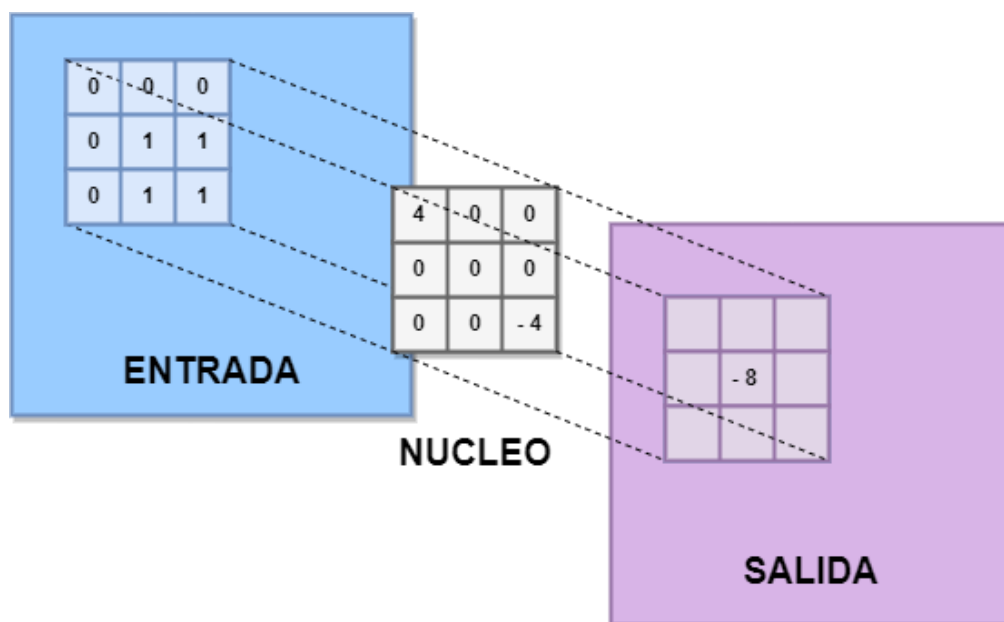
CONV2D

Las capas convolucionales son el núcleo de una red neuronal convolucional. Se encargan de extraer la información de los datos de entrada (imágenes en la mayoría de los casos) actuando de forma muy similar al córtex visual.

La convolución es una operación de agrupación matemática que involucra dos matrices: La imagen de entrada a la que se le aplicará la convolución (en forma de matriz), y una matriz llamada kernel.

La función principal del kernel es actuar como un filtro que permite detectar los bordes y líneas, así como desenfocar o enfocar las imágenes.

Al aplicar este filtro, se origina una nueva matriz de las mismas dimensiones que la imagen original.



*Figura 7: Funcionamiento de una capa convolucional.
Fuente: Elaboración propia*

El cálculo de la matriz de convolución se realiza desplazando el kernel por cada uno de los elementos de la matriz principal y haciendo la suma de los productos de cada uno de los elementos de ambas matrices. Como resultado de este procedimiento se obtiene un mapa de activaciones llamado mapa de características, que indica las ubicaciones y la fuerza de una característica detectada en la imagen.

Las primeras capas convolucionales de una CNN detectan líneas y características básicas de la imagen. A medida que se profundiza en la red, las capas detectan características y elementos específicos de la imagen que les permiten diferenciar objetos más complejos, tales como personas o animales. Véase figura 8 [6]

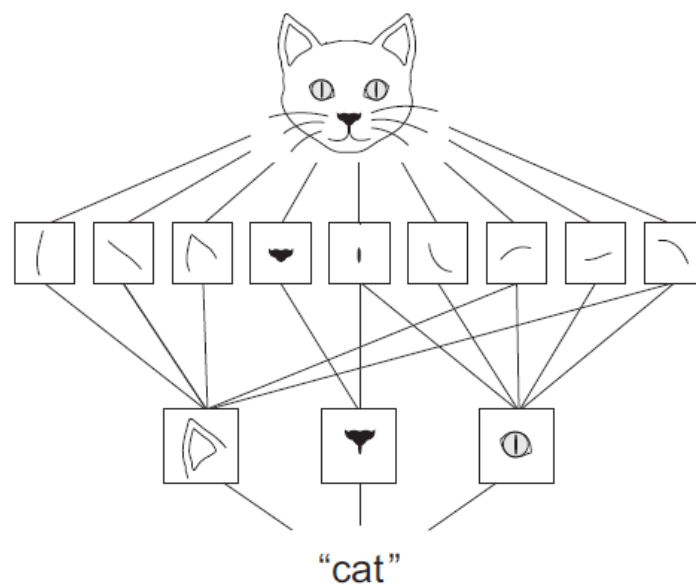


Figura 8: Ejemplo de detección de una capa convolucional [4]

DENSE

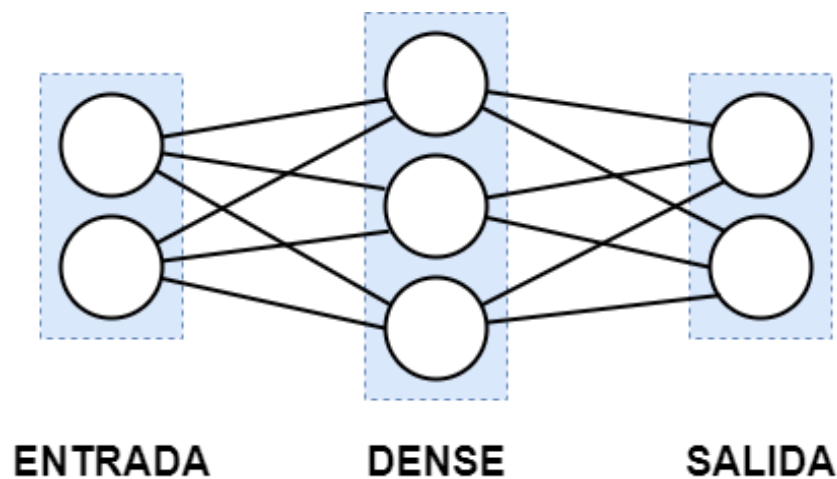
La capa densa es la capa más utilizada en la construcción de redes neuronales. Dispone una conexión profunda, lo que significa que cada neurona en la capa densa se encuentra conectada con todas las neuronas de la capa anterior, por lo tanto, reciben información de cada una de ellas.

Para procesar los datos recibidos, se aplica la siguiente operación en cada uno de los nodos

$$y = \text{activacion}(W * x + b)$$

Donde *activación* es la función de activación elegida, **W** es una matriz de pesos creada por la capa, **x** son los datos de entrada y **b** hace referencia a un conjunto adicional de pesos. Esta operación genera un vector **y** de orden *n* (número de nodos)

En las redes convolucionales, la capa densa es utilizada como capa de salida. Otros usos de esta capa son el cambio de dimensionalidad de los datos y la escalada, la rotación y traducción del vector.



*Figura 9: Ejemplo de capa densa.
Fuente: Elaboración propia*

MAX POOLING

Max Pooling es una operación de agrupación que calcula el valor máximo para cada dato de entrada y crea una nueva muestra reducida que contiene las características más importantes de dicha muestra. Generalmente, la capa de *Max Pooling* se usa después de una capa convolucional. Esta operación logra reducir el gasto computacional y prevenir el *overfitting*.

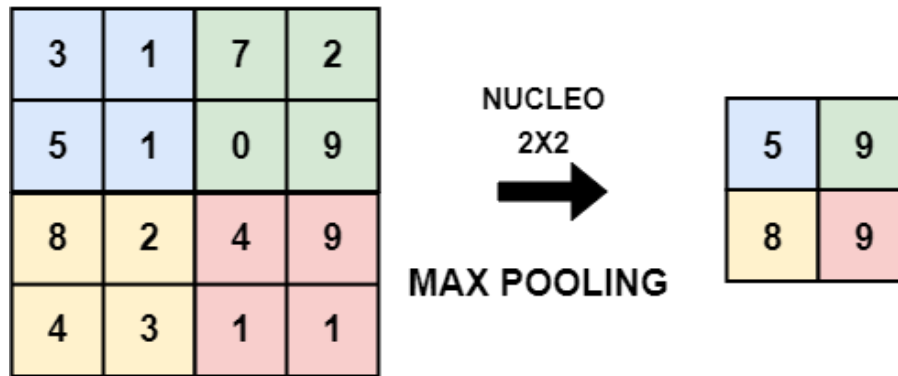


Figura 10: Funcionamiento de una capa de Max Pooling.
Fuente: Elaboración propia

DROPOUT

El Dropout es una de las técnicas de regularización más efectivas y utilizadas en la creación de redes neuronales.

Esta técnica desactiva un número de neuronas de una red neuronal de forma aleatoria. En cada iteración se desactivan distintas neuronas provocando una reducción de la dependencia entre las distintas neuronas vecinas y por consecuencia, una reducción del *overfitting*.

El número de neuronas desactivadas es controlado mediante un parámetro de entrada comprendido en el rango de 0 a 1. Por defecto, el parámetro se sitúa en 0.5, indicando que la mitad de las neuronas quedaran desactivadas.

La capa de dropout se ubica de forma general detrás de una capa de activación.

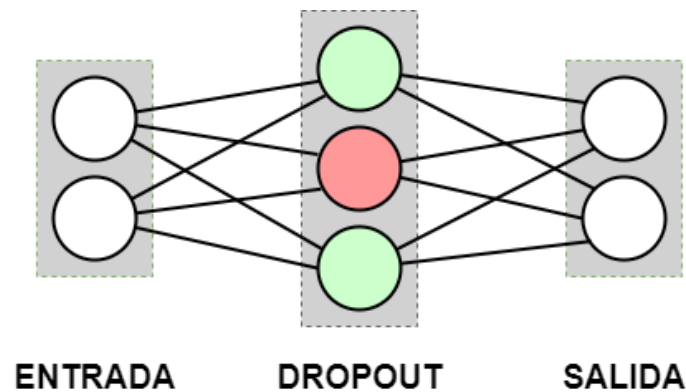


Figura 11: Funcionamiento de una capa de Dropout.
Fuente: Elaboración propia

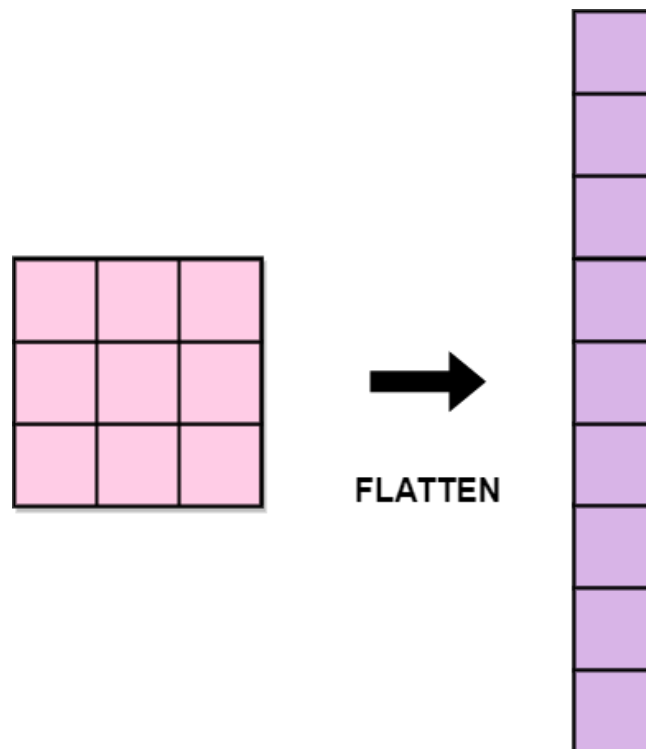
BATCH NORMALIZATION

Batch normalization es una técnica de estandarización que normaliza la contribución a una capa para cada lote de datos. Su función principal es reducir la distancia entre las distintas muestras, ayudando al modelo en cuestión a aprender y a generalizar bien los nuevos datos de entrada. Mediante un parámetro de entrada se especifica la característica del eje que debe de ser normalizado. Por defecto, este parámetro es -1

La capa de se ubica de forma general detrás de una capa convolucional o de una capa densa.

FLATTEN

La capa de *flatten* o aplanamiento realiza una operación de transformación de los datos de una matriz en una lista de datos de una dimensión. De forma genérica, este tipo de capas se ubica al final del modelo, previo a la capa de salida.



*Figura 12: Funcionamiento de una capa de Flatten.
Fuente: Elaboración propia*

Además de las capas, existen otros aspectos relevantes en la creación de una red neuronal convolucional, como son la función de activación o los conjuntos de datos utilizados, entre otros.

FUNCION DE ACTIVACIÓN

La función de activación decide si una neurona debe activarse o no, a partir del cálculo de la suma ponderada. El propósito es introducir no linealidad en la salida de una neurona y generalmente el rango de valores de salida está comprendido entre (0,1) o (-1,1). Se buscan funciones cuyas derivadas sean simples para minimizar los costes computacionales. [7]

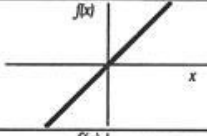
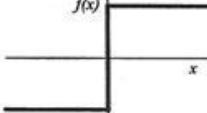
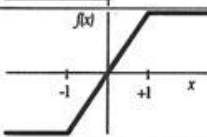
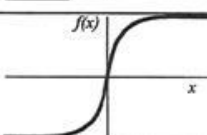
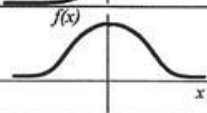
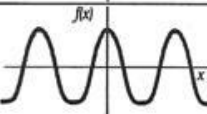
| | Función | Rango | Gráfica |
|------------------------|---|-----------------------------|--|
| Identidad | $y = x$ | $[-\infty, +\infty]$ |  |
| Escalón | $y = \text{sign}(x)$ $y = H(x)$ | $\{-1, +1\}$ $\{0, +1\}$ |  |
| Lineal a tramos | $y = \begin{cases} -1, & \text{si } x < -l \\ x, & \text{si } -l \leq x \leq +l \\ +1, & \text{si } x > +l \end{cases}$ | $[-1, +1]$ |  |
| Sigmoidea | $y = \frac{1}{1+e^{-x}}$ $y = \text{tgh}(x)$ | $[0, +1]$ $[-1, +1]$ |  |
| Gaussiana | $y = Ae^{-Bx^2}$ | $[0, +1]$ |  |
| Sinusoidal | $y = A \sin(\omega x + \varphi)$ | $[-1, +1]$ |  |

Tabla 10: Tipos de funciones de activación. [8]

TRAIN, TEST Y VALIDACION

Los datos de entrada utilizados para la creación de la red neuronal son divididos generalmente en 3 conjuntos:

- Conjunto de entrenamiento, utilizado para entrenar el modelo y hacer que aprendan las características/patrones ocultos de los datos.
- Conjunto de validación, utilizado para validar el rendimiento del modelo durante el entrenamiento y ajustar los hiperparámetros y las configuraciones del modelo en consecuencia.
- Conjunto de pruebas, utilizado para probar el modelo, una vez terminado el entrenamiento.

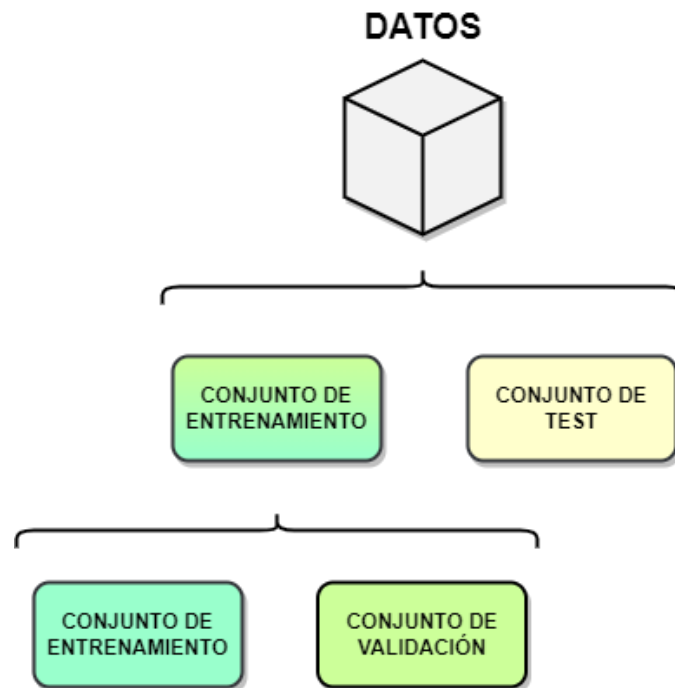


Figura 13: Ejemplo de división de los datos.
Fuente: Elaboración propia

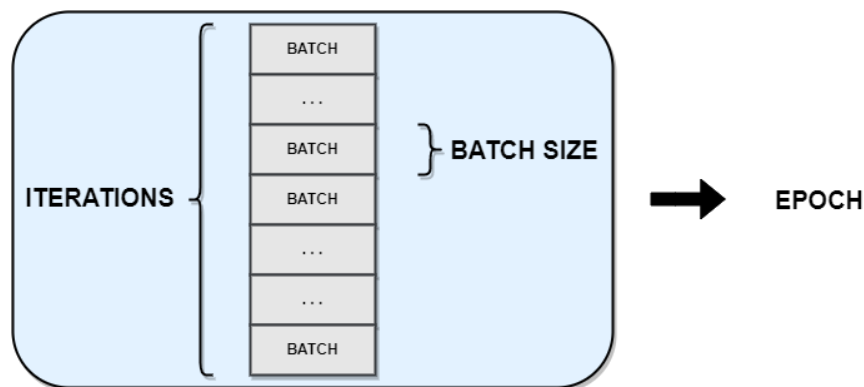
EPOCH, BATCH SIZE, ITERACIONES

Los hiperparámetros de una red neuronal controlan la profundidad y el alcance del entrenamiento de una red. De este proceso depende el rendimiento del modelo entrenando, por lo que es crucial realizar una adecuada configuración en cada caso. Entre los hiperparámetros existentes destacan los epochs, el batch size y las iteraciones.

Los epochs o épocas indican el número de veces que el conjunto de datos entero es utilizado para entrenar el modelo.

El batch size o tamaño de lote indica la cantidad de muestras que se toman en cada iteración

Las iteraciones indican el número de lotes necesarios para completar una época. [7]



*Figura 14: Ejemplo de epoch, batch size e iteraciones.
Fuente: Elaboración propia*

FUNCION DE PERDIDA

La función de pérdida en una red neuronal cuantifica el error cometido entre el resultado esperado y el resultado obtenido por la ejecución del modelo. Esta evaluación es utilizada posteriormente para derivar los gradientes y con ello actualizar los pesos.

OPTIMIZERS

El optimizador realiza ajustes en los parámetros de la red neuronal, como los pesos o la tasa de aprendizaje, para reducir la pérdida durante el entrenamiento del modelo y con ello mejorar la precisión de este.

OVERFITTING Y UNDERFITTING

El rendimiento de los modelos depende en mayor medida del ajuste de los parámetros realizado. El sobreajuste o *overfitting*, causa que el modelo entrenado solo se ajuste a aprender los casos particulares facilitados en el entrenamiento y sea incapaz de reconocer nuevos datos. Por otro lado, el desajuste o *underfitting*, aparece cuando el modelo dispone de pocos datos de entrenamiento, incapacitando al modelo para generalizar el conocimiento adquirido y poder clasificar nuevos datos. [9]

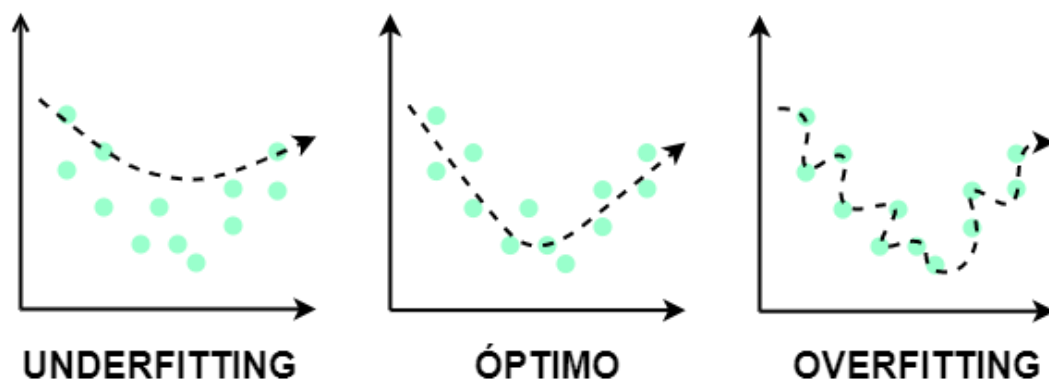


Figura 15: Tipos de ajuste de un modelo.
Fuente: Elaboración propia

DATA AUGMENTATION

La técnica de data augmentation permite agregar nuevos datos a un conjunto a partir de pequeñas modificaciones realizadas a datos ya existentes, provocando de esta forma una reducción considerable del overfitting en el proceso de entrenamiento del modelo. [10]

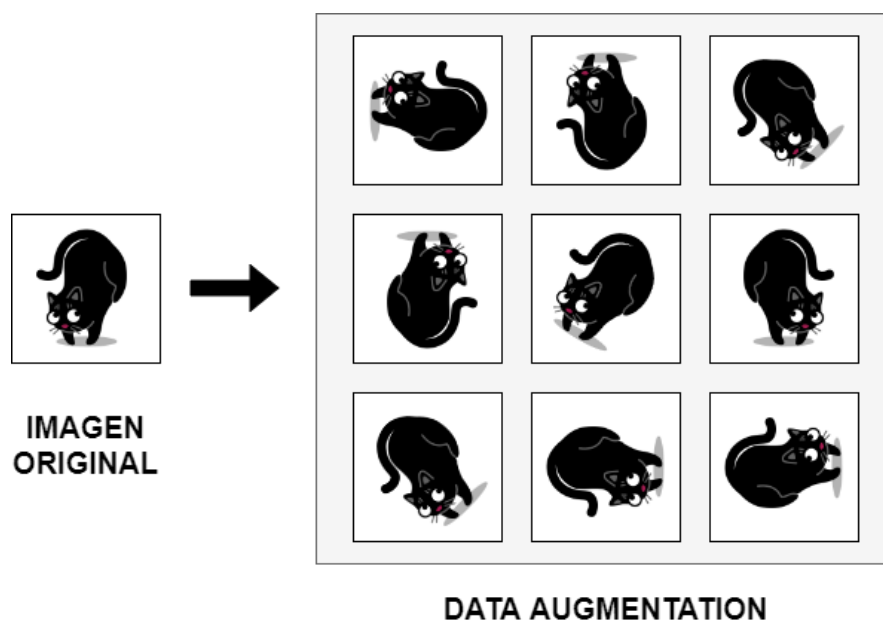


Figura 16: Funcionamiento de la técnica data augmentation.
Fuente: Elaboración propia

EARLY STOPPING

La técnica *early stopping* permite detener el proceso de entrenamiento una vez que el rendimiento del modelo deje de mejorar al aplicar el conjunto de datos de validación. De esta forma, se puede elegir un elevado número de épocas para el entrenamiento sin provocar *overfitting*. [10]

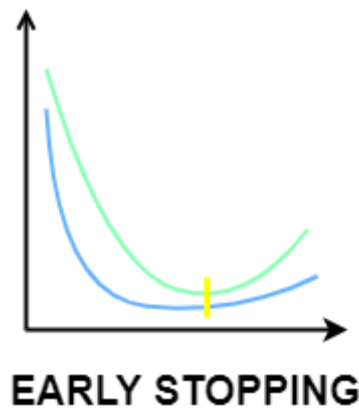


Figura 17: Funcionamiento de la técnica *early stopping*.
Fuente: Elaboración propia

LIBRERÍAS: TENSORFLOW Y KERAS.

TENSORFLOW

Tensorflow es una plataforma de código abierto orientada al aprendizaje automático. Fue desarrollada originalmente por el equipo de Google Brain y se implementa principalmente en lenguaje de programación Python.

La plataforma cuenta con un ecosistema integral y flexible de herramientas y bibliotecas que permiten a las grandes empresas y a los investigadores desarrollar e implementar sus proyectos. Entre las grandes empresas que usan esta tecnología, se encuentran Airbnb, Coca-Cola, General Electric, entre otras.

Entre las principales aplicaciones de Tensorflow, se encuentra el desarrollo de redes neuronales, sistemas de recomendación o redes generativas adversas.

Dispone de una biblioteca de JavaScript, para entrenar e implementar modelos en navegadores web y en Node.js, una biblioteca orientada a los dispositivos móviles y IoT y una biblioteca enfocada a los entornos de gran producción. [11]

KERAS

Keras es una biblioteca de redes neuronales artificiales de código abierto implementada en el lenguaje de programación Python. Principalmente ha sido desarrollada por François Chollet. El objetivo de la biblioteca es acelerar la creación de redes neuronales, para ello, Keras funciona como una API que permite acceder a frameworks de aprendizaje automático entre los que se encuentran TensorFlow, Theano y Microsoft Cognitive Toolkit.[12]

DIAGNOSTICO MEDIANTE RADIOGRAFÍAS PULMONARES.

La COVID-19, enfermedad por coronavirus 2019, se trata de una enfermedad infecciosa generada por una cepa de coronavirus llamada SARS-CoV-2. [13]

Los primeros casos aparecieron en Wuhan, China, en diciembre de 2019 y desde allí ha ido transmitiéndose rápidamente por todo el mundo. Finalmente fue reconocida como pandemia por la Organización Mundial de la Salud (OMS) el 11 de marzo de 2020 [14]

Esta pandemia ha dejado como consecuencia una grave crisis económica y sanitaria, imponiendo el estricto confinamiento mundial para evitar el colapso de los centros sanitarios.

El virus se transmite principalmente a través del contacto con gotitas de secreciones del tracto respiratorio superior de las personas infectadas. La infección ocurre por lo general, entre 7-14 días posteriores a la expansión. [15]

Diagnóstico

Normalmente la prueba que se lleva a cabo para la detección de esta cepa de coronavirus es la reacción en cadena de la polimerasa con transcriptasa inversa (RT-PCR) obtenidas habitualmente de muestras nasofaríngeas. [16]

Las pruebas de imagen poseen un rol fundamental para la detención de esta patología. Además, esta herramienta sirve para determinar la gravedad de la patología y guiar el tratamiento para el paciente infectado. Es por ello, números profesionales de la salud, afirman que este método no es tan eficaz para un diagnóstico, sino para evaluar la complejidad de la enfermedad. [16]

Radiografía de tórax

Las radiografías (rayos x) utilizan la radiación para producir una imagen en 2D. Ocasionalmente, se llevan a cabo en hospitales donde el radiólogo utiliza un equipo fijo, no obstante, también se pueden llevar a cabo con equipos portátiles.

Los rayos X son una forma de radiación tales como la luz o las ondas de sonido. Los rayos x atraviesan la mayoría de los objetos y cuerpos. La persona encargada de llevar a cabo esta prueba (el tecnólogo), debe apuntar de manera eficaz el haz de rayos x hacia el área de interés.

La radiación imprime una imagen en película fotográfica o en detector especial.

Los rayos X son absorbidos por diferentes partes del cuerpo en variables grados. Los huesos absorben gran parte de la radiación mientras que los tejidos blandos (los músculos, la grasa, y los órganos) permiten que una mayor cantidad de los rayos X pasen a través de ellos. Como consecuencia, los huesos aparecen blancos en los rayos X mientras que los tejidos blandos se muestran en matices de gris y el aire aparece en negro.

En una radiografía de tórax, las costillas y la columna absorberán gran parte de la radiación y se visualizarán en blanco o gris claro en la imagen. El tejido pulmonar absorbe poca radiación, y aparecerá en negro en la imagen.

La mayoría de las imágenes son imágenes que se archivan en forma de archivos digitales. Su médico puede acceder fácilmente a estas imágenes grabadas para diagnosticar y controlar su condición.

La investigación identifica qué pacientes pueden necesitar someterse a una prueba de radiodiagnóstico dependiendo de las sospechas del facultativo de una posible neumonía. Así como la necesidad de una hospitalización, ingreso en la unidad de cuidados intensivos (UCI) etc.

La radiografía de tórax puede ser normal en los casos leves o en las fases precoces de la enfermedad, pero es poco probable que los pacientes con clínica moderada o grave tengan una radiografía de tórax normal. La mayoría son patológicas en aquellos que precisan hospitalización (el 69% al ingreso y el 80% en algún momento del ingreso). Los hallazgos son más extensos a los 10-12 días del inicio de los síntomas. [17]

Sistema automático de apoyo al diagnóstico mediante radiografías

El proceso comienza con la recopilación de las radiografías que serán utilizadas en la construcción del sistema, cada uno etiquetado con su categoría.

Durante el entrenamiento el sistema recibe el conjunto de datos y genera una salida en forma de vector que contiene la puntuación para cada salida o categoría. El sistema evalúa con la ayuda de una función el error cometido entre la salida deseada y la obtenida en el entrenamiento y adecua los parámetros internos o pesos para reducir el error.

Posteriormente, se realiza el cálculo de un vector gradiente que indica para cada peso, la mejora o empeoramiento que sufriría al aumentar el peso una pequeña cantidad.

El vector gradiente resultante es utilizado en la dirección contraria para ajustar el vector peso y mejorar el resultado de este.

Este procedimiento se realiza de forma iterativa y finalmente se obtiene un sistema final entrenado capaz de realizar un diagnóstico mediante radiografías. [17]

CAPÍTULO 3: ESTUDIO PRÁCTICO DEL ESTADO DEL ARTE

En este capítulo se describe el procedimiento realizado para la adquisición de los conocimientos necesarios en el proyecto y su posterior puesta en práctica haciendo uso del banco de datos MNIST y Cats vs Dogs.

El proceso se ha llevado a cabo en la plataforma Kaggle donde existen multitud de cursos destinados al aprendizaje, además de disponer de los conjuntos de datos necesarios.

A continuación, se detallan brevemente los cursos realizados

ESTUDIO PREVIO

Curso de introducción al aprendizaje automático

Orientado a la iniciación de la materia, se explica el funcionamiento básico de los modelos de aprendizaje automático haciendo uso de los “árboles de decisión” y se introduce la librería “pandas”, que es de gran utilidad para el manejo de los datos. Además, se definen los conceptos de *Overfitting* y *Underfitting*.

Curso de aprendizaje automático intermedio

Se explican las técnicas utilizadas cuando se poseen datos incompletos y los tipos de variables existentes (categóricas, binarias, ...). Además, se exponen técnicas de preprocesamiento como las tuberías y técnicas de evaluación de modelos como la validación cruzada.

Curso de introducción al aprendizaje profundo

Se realiza una introducción al aprendizaje profundo y a las redes neuronales, definiendo los modelos básicos y sus componentes (neuronas, capas, funciones de activación, ...), las funciones estadísticas utilizadas en la creación de las redes y las librerías Keras y Tensorflow, que proporcionan hábiles y funciones para el modelado y entrenamiento de las redes neuronales.

ESTUDIO PRÁCTICO: 1º CASO MNIST

La competición MNIST [18] considerada el “Hola mundo” para todo aquel que se inicia en la materia, está formada por un conjunto de imágenes de dígitos manuscritas (dígitos del 0-9) y sus correspondientes etiquetas. El objetivo de la competición es la correcta clasificación e identificación de los dígitos.

Descripción de las imágenes

El conjunto de datos se compone por 60.000 muestras con una resolución de 28x28 píxeles con escala de grises.

Se encuentran distribuidas en 2 ficheros: *train.csv* y *test.csv*

- *train.csv*: 42.000 imágenes
- *test.csv*: 28.000 imágenes

El conjunto de entrenamiento dispone de 785 columnas. La primera columna indica el valor de la etiqueta y mientras que el resto de las columnas hacen referencia a los valores de los píxeles de la imagen asociada.

El conjunto de test posee 784 columnas que hacen referencia a los valores de los píxeles de la imagen asociada, sin embargo, este conjunto no posee una columna con el valor de la etiqueta.

Preprocesado de los datos

Para la creación del modelo, se almacenan los datos de entrenamiento y de test en las variables *X_train*, *Y_train* y *X_test*. En el caso de los datos de entrenamiento, se ha utilizado la función *drop* perteneciente a la librería **Pandas** para separar la columna referente a la etiqueta de la imagen.

Para finalizar el tratamiento de los datos, se ha reescalado cada una de las imágenes (escala de 0 a 255).

```
Y_train = train[1b]
X_train = train.drop(labels = 1b,axis = 1)

X_train = X_train / sc
X_test = test / sc

Y_train = to_categorical(Y_train, num_classes = numClass)

X_train = X_train.values.reshape(-1,imageWidht,imageHeight,imageChannels)
X_test = X_test.values.reshape(-1,imageWidht,imageHeight,imageChannels)
```

*Código 1: División de los datos en X_train, Y_train y X_test.
Fuente: Elaboración propia*

Creación del modelo

Se ha creado un modelo CNN formado dividido en dos partes: Una etapa convolucional y una etapa de clasificación.

La etapa convolucional está formada por capas convolucionales (Conv2D), capas de batch_normalization y capas de dropout mientras que la etapa de clasificación, situada al final del modelo, se compone por una capa de flatten, una capa de dropout y una capa densa que actúa como capa de salida.

En la figura 18 se puede observar el modelo utilizado.

Model: "sequential"

| Layer (type) | Output Shape | Param # |
|---|--------------------|---------|
| conv2d (Conv2D) | (None, 26, 26, 32) | 320 |
| batch_normalization (Batch Normalization) | (None, 26, 26, 32) | 128 |
| conv2d_1 (Conv2D) | (None, 24, 24, 32) | 9248 |
| batch_normalization_1 (Batch Normalization) | (None, 24, 24, 32) | 128 |
| conv2d_2 (Conv2D) | (None, 12, 12, 32) | 25632 |
| dropout (Dropout) | (None, 12, 12, 32) | 0 |
| conv2d_3 (Conv2D) | (None, 10, 10, 64) | 18496 |
| batch_normalization_2 (Batch Normalization) | (None, 10, 10, 64) | 256 |
| conv2d_4 (Conv2D) | (None, 8, 8, 64) | 36928 |
| batch_normalization_3 (Batch Normalization) | (None, 8, 8, 64) | 256 |
| conv2d_5 (Conv2D) | (None, 4, 4, 64) | 102464 |
| dropout_1 (Dropout) | (None, 4, 4, 64) | 0 |
| conv2d_6 (Conv2D) | (None, 1, 1, 128) | 131200 |
| batch_normalization_4 (Batch Normalization) | (None, 1, 1, 128) | 512 |
| flatten (Flatten) | (None, 128) | 0 |
| dropout_2 (Dropout) | (None, 128) | 0 |
| dense (Dense) | (None, 10) | 1290 |
| Total params: 326,858 | | |
| Trainable params: 326,218 | | |
| Non-trainable params: 640 | | |

Figura 18: Esquema del modelo generado para el conjunto MNIST.

Fuente: Elaboración propia

Entrenamiento y verificación del modelo

Una vez obtenido el modelo, se realiza ha realizado el entrenamiento y validación del modelo con la función *fit* perteneciente a la librería **Keras**. Se ha utilizado un *batch_size* de 64 y 50 *epochs*

```
X_train2, X_val2, Y_train2, Y_val2 = train_test_split(X_train, Y_train, test_size = 0.1)

history = myModel.fit(
    X_train2, Y_train2,
    validation_data=(X_val2, Y_val2),
    batch_size=64,
    epochs=epochs,
    steps_per_epoch = X_train2.shape[0]//64,
    callbacks=[early_stopping],
    verbose=0, # suppress output since we'll plot the curves
)
```

Código 2: Entrenamiento del modelo generado.

Fuente: Elaboración propia

Para obtener el conjunto de validación, se ha utilizado la función *train_test_split* perteneciente a la librería **Sklearn** para obtener el 10% de las imágenes reservadas para el entrenamiento.

```
Minimum Validation Loss: 0.0305
0.996396005153656
0.9923809766769409
```

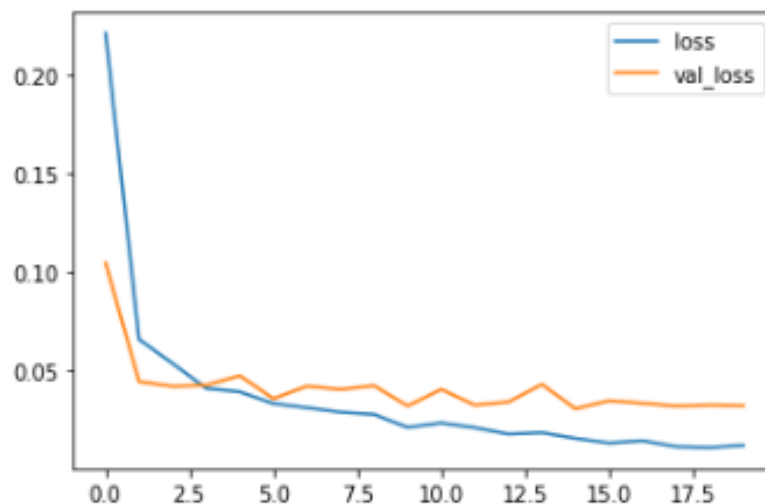


Figura 19: Resultado del entrenamiento del modelo.

Fuente: Elaboración propia

El resultado obtenido del entrenamiento se muestra en la figura 19. Como se puede observar, la gráfica muestra la pérdida mínima conseguida por el modelo en el proceso de entrenamiento y de validación. Ambas curvas logran un resultado similar llegando a obtener como mejor resultado una pérdida aproximada de 0.03.

En cuanto a precisión, el modelo consigue clasificar aproximadamente el 99% de las muestras en la etapa de entrenamiento y de validación

Para comprobar la precisión del modelo, se ha realizado la verificación del entrenamiento con el conjunto de test haciendo uso de la función *predict* perteneciente a la librería **Keras**.

En la figura 20 se muestran 16 imágenes aleatorias resultantes de este proceso.

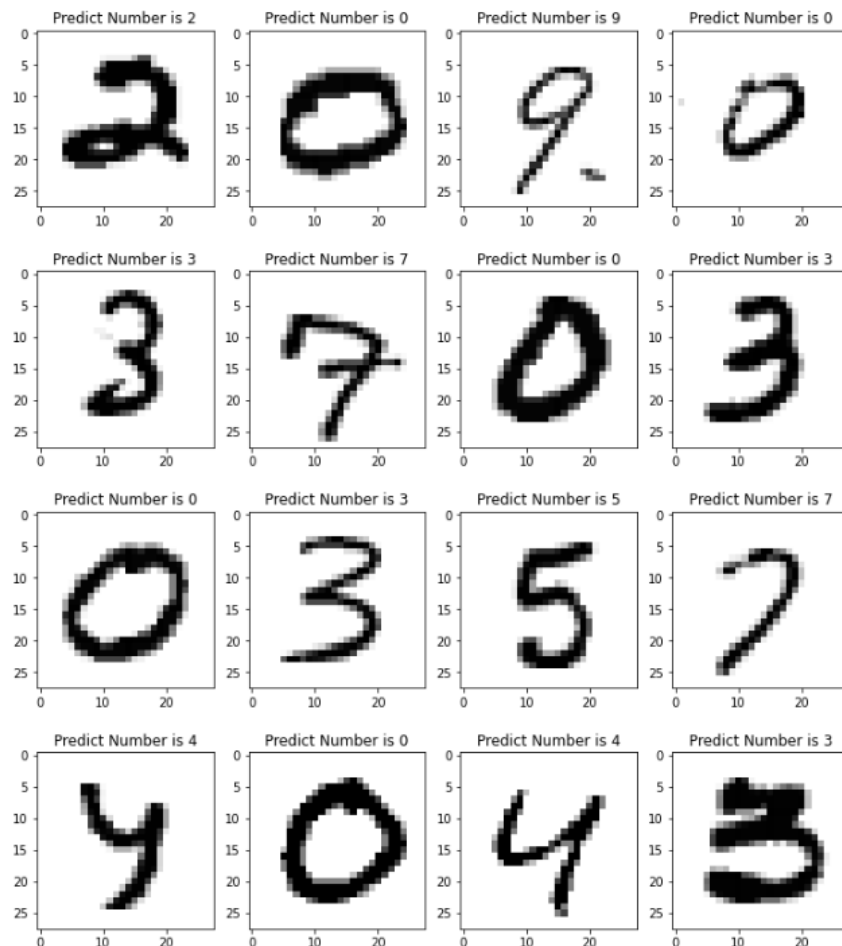


Figura 20:Resultado de la clasificación realizada por el modelo.

Fuente: Elaboración propia

Como puede observarse en la figura 20, el modelo ha logrado clasificar correctamente las 16 imágenes, obteniendo un 100% de tasa de acierto

ESTUDIO PRÁCTICO: 2º CASO CATS VS DOGS

La competición *Cats vs Dogs* [19] tiene como objetivo la creación de un modelo capaz de identificar y diferenciar imágenes de perros y gatos.

Descripción de las imágenes

El conjunto de datos se compone por 37.500 muestras con una resolución de 150x150 píxeles.

Se encuentran distribuidas en 2 ficheros: *train.zip* y *test1.zip*

- *train.zip*: 25.000 imágenes
- *test1.zip*: 12.500 imágenes

Preprocesado de los datos

Para la creación del modelo, se extraen los datos y se crean *DataFrames* con la librería **Panda**, para el conjunto de entrenamiento y para el conjunto de test.

Los Dataframes son unas estructuras de datos que almacenan las imágenes junto a sus categorías, a las que previamente se les ha asignado un 0 para las imágenes que contienen un perro y 1 para las imágenes que contienen un gato.

Al conjunto de entrenamiento se le aplica la función *train_test_split* perteneciente a la librería **sklearn** para obtener el 20% de los datos como conjunto de validación.

Para finalizar el preprocesamiento de los datos (entrenamiento, validación y test), se realiza un proceso de normalización de los píxeles de cada una de las imágenes (escalado de los píxeles a un rango de 0 a 255) y se emplea la técnica del *Data Augmentation* para obtener un conjunto de 40.000 imágenes para el entrenamiento y de 10.000 imágenes para la validación.

Este procedimiento se ha realizado con la función *ImageDataGenerator*, definida en la librería **Keras**.

```
train_datagen = ImageDataGenerator(
    rotation_range=15,
    rescale=1./255,
    shear_range=0.1,
    zoom_range=0.2,
    horizontal_flip=True,
    width_shift_range=0.1,
    height_shift_range=0.1
)
```

Código 3: Ejemplo de uso de la técnica data augmentation.
Fuente: Elaboración propia

Creación del modelo

Se ha creado un modelo CNN formado dividido en dos partes: Una etapa convolucional y una etapa de clasificación.

La etapa convolucional está formada por capas convolucionales (Conv2D), capas de batch_normalization y capas de dropout mientras que la etapa de clasificación, situada al final del modelo, se compone por una capa de flatten, una capa de dropout y una capa densa que actúa como capa de salida.

En la figura 21 se puede observar el modelo utilizado.

Model: "sequential"

| Layer (type) | Output Shape | Param # |
|---|---------------------|---------|
| conv2d (Conv2D) | (None, 74, 74, 32) | 1568 |
| batch_normalization (Batch Normalization) | (None, 74, 74, 32) | 128 |
| dropout (Dropout) | (None, 74, 74, 32) | 0 |
| conv2d_1 (Conv2D) | (None, 71, 71, 64) | 32832 |
| batch_normalization_1 (Batch Normalization) | (None, 71, 71, 64) | 256 |
| conv2d_2 (Conv2D) | (None, 34, 34, 64) | 65600 |
| batch_normalization_2 (Batch Normalization) | (None, 34, 34, 64) | 256 |
| dropout_1 (Dropout) | (None, 34, 34, 64) | 0 |
| conv2d_3 (Conv2D) | (None, 31, 31, 128) | 131200 |
| batch_normalization_3 (Batch Normalization) | (None, 31, 31, 128) | 512 |
| conv2d_4 (Conv2D) | (None, 31, 31, 128) | 262272 |
| batch_normalization_4 (Batch Normalization) | (None, 31, 31, 128) | 512 |
| conv2d_5 (Conv2D) | (None, 14, 14, 128) | 262272 |
| batch_normalization_5 (Batch Normalization) | (None, 14, 14, 128) | 512 |
| dropout_2 (Dropout) | (None, 14, 14, 128) | 0 |
| conv2d_6 (Conv2D) | (None, 7, 7, 256) | 524544 |
| batch_normalization_6 (Batch Normalization) | (None, 7, 7, 256) | 1024 |
| dropout_3 (Dropout) | (None, 7, 7, 256) | 0 |
| flatten (Flatten) | (None, 12544) | 0 |
| dense (Dense) | (None, 512) | 6423040 |
| dropout_4 (Dropout) | (None, 512) | 0 |
| dense_1 (Dense) | (None, 2) | 1026 |
| Total params: 7,707,554 | | |
| Trainable params: 7,705,954 | | |
| Non-trainable params: 1,600 | | |

Figura 21: Esquema del modelo generado para el conjunto Cats vs Dogs.

Fuente: Elaboración propia

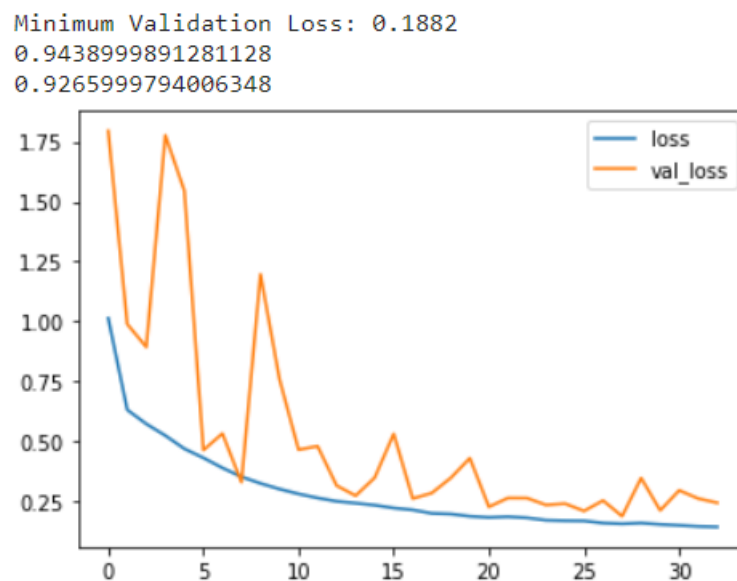
Entrenamiento y verificación del modelo

Una vez obtenido el modelo, se realiza ha realizado el entrenamiento y validación del modelo con la función *fit* perteneciente a la librería **Keras**. Se ha utilizado un *batch_size* de 100 y 50 *epochs*

```
history = myModel.fit(
    train_generator,
    validation_data=val_generator,
    batch_size=batch_size,
    validation_steps=X_val2.shape[0]//batch_size,
    steps_per_epoch=X_train2.shape[0]//batch_size,
    epochs=epoch,
    callbacks=[early_stopping],
    verbose=0, # suppress output since we'll plot the curves
)
```

*Código 4: Entrenamiento del modelo generado.
Fuente: Elaboración propia*

El resultado obtenido del entrenamiento se muestra en la figura 22. Como se puede observar, la gráfica muestra la perdida mínima conseguida por el modelo en el proceso de entrenamiento y de validación.



*Figura 22: Resultado del entrenamiento del modelo.
Fuente: Elaboración propia*

La curva azul, referente a la etapa del entrenamiento realiza un suave descenso durante su ejecución, sin embargo, la curva naranja referente a la etapa de validación comienza su ejecución sufriendo cambios bruscos y no es hasta la ejecución 20, cuando consigue suavizar su curva. El modelo obtener como mejor resultado una perdida aproximada de 0.18

En cuanto a precisión, el modelo consigue clasificar aproximadamente el 94% de las muestras en la etapa de entrenamiento y el 92% de las muestras en la etapa de validación

Para comprobar la precisión del modelo, se ha realizado la verificación del entrenamiento con el conjunto de test haciendo uso de la función *predict* perteneciente a la librería **Keras**.

En la figura 23 se muestran 18 imágenes aleatorias resultantes de este proceso.

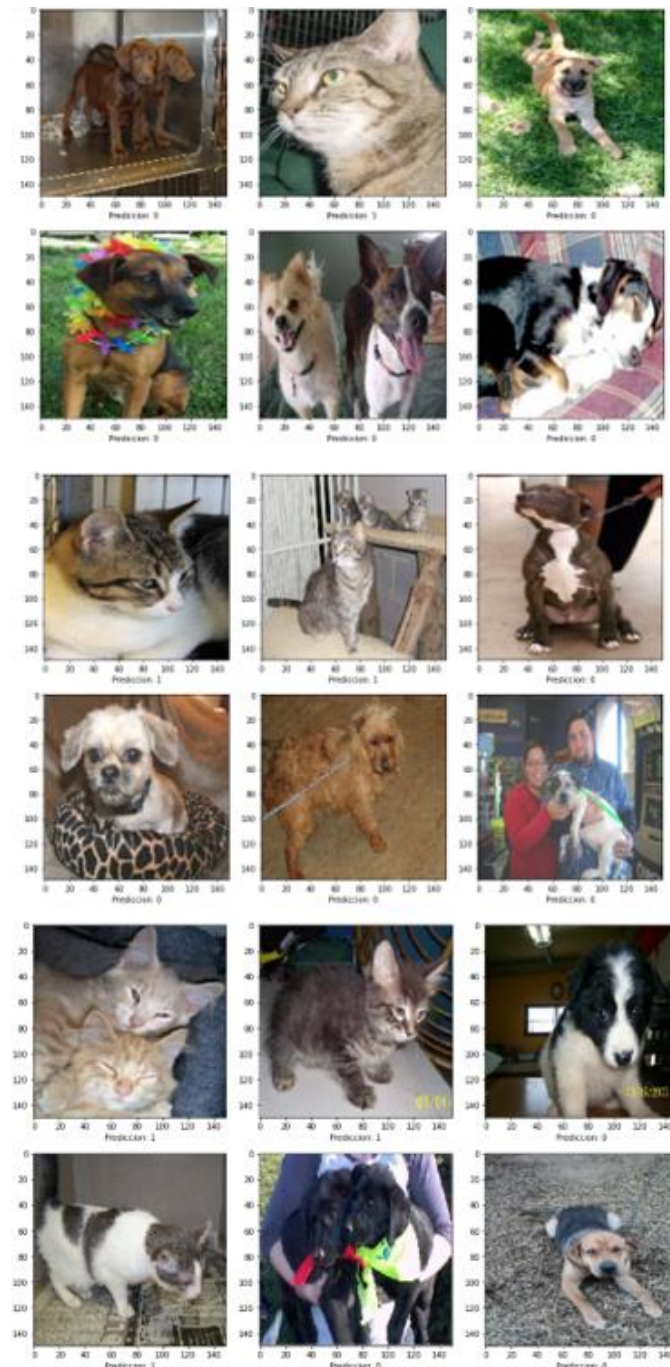


Figura 23: Resultado de la clasificación realizada por el modelo
Fuente: Elaboración propia

Como puede observarse en la figura 23, el modelo ha logrado clasificar correctamente las 18 imágenes, obteniendo un 100% de tasa de acierto

CAPÍTULO 4: DISEÑO EXPERIMENTAL

En este capítulo se van a describir los pasos que se han seguido para alcanzar el objetivo final de este proyecto fin de grado, es decir, la implementación de un clasificador de imágenes basadas en radiografías rayos X con *Deep Learning*.

En la figura 24 se observa el flujo de trabajo con los pasos principales que se detallan a continuación.

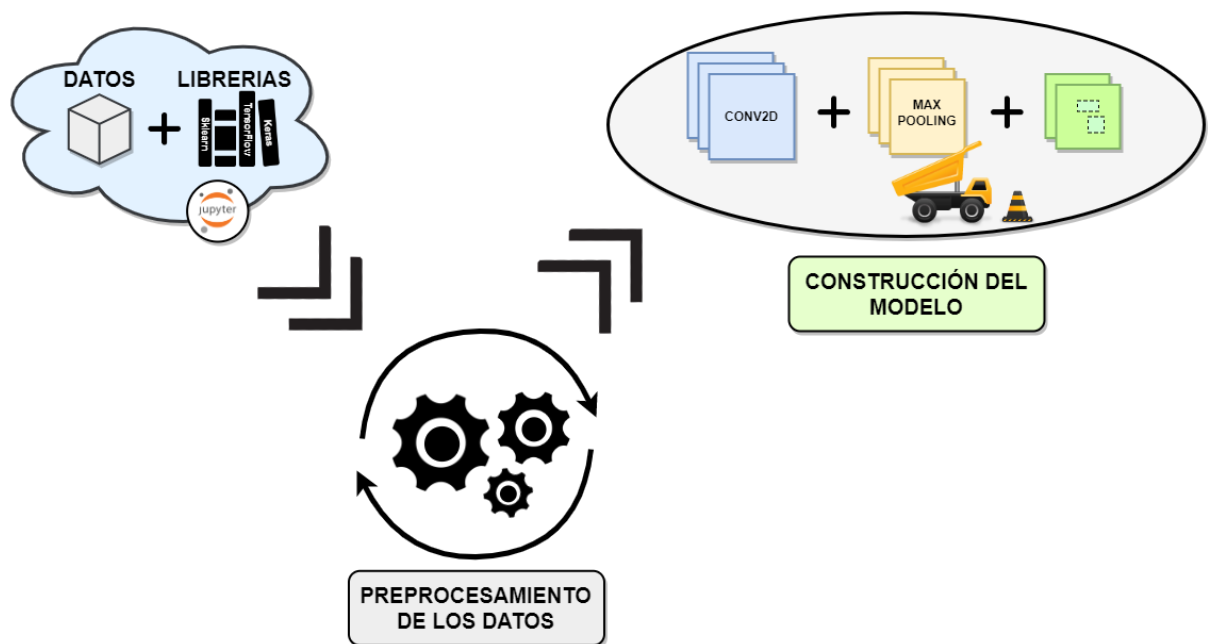


Figura 24: Flujo de trabajo del capítulo 4.
Fuente: Elaboración propia

DECISIONES DE TIPO TÉCNICO

Banco de imágenes:

La base de datos utilizada para la creación del clasificador de imágenes se denomina *COVID-19 Radiography Database* y en su creación han participado un grupo de investigadores de la universidad de Qatar (Doha, Qatar), y de la universidad de Dhaka (Bangladesh).

Se compone por 21165 muestras distribuidas en 4 categorías: Covid-19, neumonía viral, opacidad torácica y normales. [20]

Entorno de trabajo:

El reto se ha desarrollado en el entorno de Jupyter notebook que proporciona la plataforma de Kaggle, tal como se realizó con anterioridad.

La plataforma permite ejecutar los notebooks en la nube y seleccionar la unidad de procesamiento deseada (CPU, GPU, TPU) para el entrenamiento del modelo, aunque existen limitaciones temporales para el uso de la GPU y la TPU.

Librerías utilizadas

Para la creación del modelo, ha sido necesario el empleo de las siguientes librerías:

- Tensorflow
- Keras
- Sklearn

Así como las siguientes librerías adicionales

- Numpy
- Pandas
- Os
- Re
- Matplotlib
- Cv2
- Glob

Descripción de las imágenes

El conjunto de datos se compone por 21165 muestras distribuidas en 4 categorías

- Imágenes de rayos X de tórax de pacientes positivos en COVID-19: 3616
- Imágenes de rayos X de tórax de opacidad torácica: 6012
- Imágenes de rayos X de tórax normal: 10192
- Imágenes de rayos X de tórax de neumonía viral: 1345

Además, la base de datos alberga mascarar pulmonares de cada una de las distribuciones. Estas imágenes son utilizadas para mejorar la precisión de los modelos convolucionales, sin embargo, su uso requiere de una mayor capacidad de cómputo y de una etapa de preprocesamiento, por lo que no serán utilizadas en el presente proyecto.

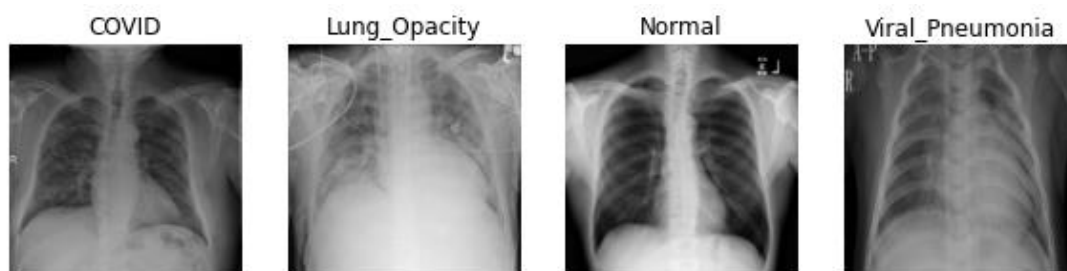
A nivel estructural, la base de datos se encuentra dividida en 4 directorios (COVID, Lung_Opacity, Normal y Viral Pneumonia). Dentro de cada directorio, se alberga un directorio *Images*, que contiene las radiografías y un directorio *Masks* que contiene las máscaras.

Las imágenes poseen una escala de color gris, su resolución es de 299x299 píxeles y se encuentran en formato PNG.

El nombre de cada una de estas imágenes sigue el siguiente formato:

[Categoría] - [número identificativo].png

donde *categoría* hace referencia a los distintos directorios.



*Figura 25: Ejemplo de imágenes clasificadas por categoría.
Fuente: Elaboración propia*

PROCESAMIENTO DE LOS DATOS

Es necesario realizar un tratamiento de los datos, sin embargo, Kaggle no permite editar los datos de entrada. Para solucionar este inconveniente existen 2 opciones:

- Crear un directorio temporal, copiar los archivos y realizarles el tratamiento adecuado
- Modificar los archivos de entrada externamente, realizarles el tratamiento y reintroducirlos en el notebook

Se ha optado por la primera opción y hacer uso de la librería **Os**. Dicha librería permite crear los directorios necesarios para cada categoría y almacenar las imágenes en ellos para su posterior tratamiento.

Para el tratamiento de datos, se ha usado la librería **Glob** para obtener la ruta de cada una de las imágenes y posteriormente con la librería **Os** y **Shutil** se realiza una copia en el directorio temporal adecuado. Una vez almacenadas las radiografías, se crea una estructura de datos denominada *DataFrame* con la librería **Panda** donde se almacenan las rutas y la categoría de las imágenes.

A esta estructura obtenida, se le aplica la función *train_test_split* perteneciente a la librería **sklearn** para dividir el conjunto total de los datos en los conjuntos de entrenamiento, de validación y de test. En la figura 26, se puede apreciar el número de datos que posee cada conjunto.

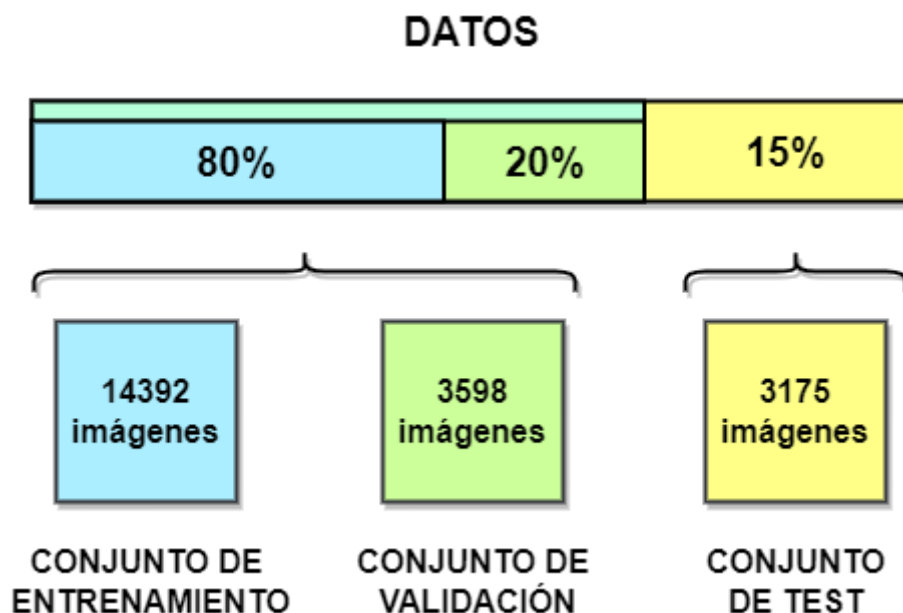


Figura 26: División de los datos en los conjuntos de entrenamiento, validación y test.
Fuente: Elaboración propia

Por último, se realiza un proceso de normalización de los píxeles a cada conjunto de datos obtenido escalando cada imagen a un rango de 0 a 255, para finalmente obtener una distribución establecida dentro del intervalo de 0 a 1.

Este proceso de normalización y división se ha realizado con la función *ImageDataGenerator*, definida en la librería **Keras**. A continuación, se muestra el procedimiento realizado con los datos de entrenamiento. Véase Código 5 y 6

```
train_generator = ImageDataGenerator(  
    rescale=1./255,  
)
```

Código 5: Proceso de normalización realizado con la función *ImageDataGenerator*.
Fuente: Elaboración propia

```
image_train = train_generator.flow_from_dataframe(  
    train_filenames,  
    target_size=(imgSize),  
    x_col='filename',  
    y_col='category',  
    class_mode="categorical",  
    shuffle=True,  
    batch_size=batch_size,  
)
```

Código 6: Fuente: Elaboración propia

El procesamiento de las imágenes se lleva a cabo mediante esta función, además permite llevar a cabo modificaciones y la obtención de información relevante, por ejemplo, nombre de categoría.

Por ello, se toman los nombres de las clases con el objetivo de finalizar el tratamiento de las imágenes o preprocesamiento.

GENERACIÓN DEL MODELO.

Construcción del modelo

Se detallan las características del modelo de red neuronal convolucional (CNN) implementado.

Para la creación del modelo, se ha realizado un estudio exhaustivo de los notebooks del resto de participantes. Entre los notebooks revisados, cabe destacar el de los participantes *Mushfirat* [21], *Saravananms* [22], *sana306* [23] y *harshwalia* [24].

Estos notebooks han actuado como una guía para la toma de decisión sobre las siguientes configuraciones:

- Ajuste de hiperparámetros
- Tipos de capas a utilizar y su combinación
- Preprocesado de los datos y visualización de las imágenes

Se ha implementado una arquitectura que consta de 6 capas. Las primeras 5 capas están formadas por una capa convolucional (*Conv2D*), seguida por una capa de normalización (*BatchNormalization*), una capa de *MaxPooling2D* y una capa de *Dropout*.

En las capas convolucionales, se ha especificado el número de filtros, el tamaño de la ventana de convolución (*kernel_size*), el tipo de *padding* y la función de activación utilizada.

En especial, para el presente proyecto, se han utilizado capas de 128, 128, 64, 64 y 32 filtros y se ha establecido un tamaño de la ventana de 3 x 3, un *padding* de tipo *same* y la función de activación *ReLU*. Además, en la primera capa se indica el tamaño de los datos utilizados en el modelo, en este caso, 299x299 píxeles.

Para las capas de tipo *BatchNormalization* y de tipo *Dropout*, se ha utilizado valores de 0.2, 0.3 o 0.5 y para capas de *MaxPooling2D*, se ha especificado un tamaño de *pool_size* de 2 x 2.

La última capa está formada por una capa de aplanamiento (*Flatten*), que tiene como objetivo reducir las dimensiones de la información de entrada, y una capa densa formada por 4 filtros, uno por cada tipo de categoría y una función de activación de tipo *Softmax*.

Model: "sequential"

| Layer (type) | Output Shape | Param # |
|---|-----------------------|---------|
| conv2d (Conv2D) | (None, 299, 299, 128) | 3584 |
| batch_normalization (Batch Normalization) | (None, 299, 299, 128) | 512 |
| max_pooling2d (MaxPooling2D) | (None, 149, 149, 128) | 0 |
| dropout (Dropout) | (None, 149, 149, 128) | 0 |
| conv2d_1 (Conv2D) | (None, 149, 149, 128) | 147584 |
| batch_normalization_1 (Batch Normalization) | (None, 149, 149, 128) | 512 |
| dropout_1 (Dropout) | (None, 149, 149, 128) | 0 |
| conv2d_2 (Conv2D) | (None, 149, 149, 64) | 73792 |
| batch_normalization_2 (Batch Normalization) | (None, 149, 149, 64) | 256 |
| dropout_2 (Dropout) | (None, 149, 149, 64) | 0 |
| conv2d_3 (Conv2D) | (None, 149, 149, 64) | 36928 |
| batch_normalization_3 (Batch Normalization) | (None, 149, 149, 64) | 256 |
| max_pooling2d_1 (MaxPooling2D) | (None, 74, 74, 64) | 0 |
| dropout_3 (Dropout) | (None, 74, 74, 64) | 0 |
| conv2d_4 (Conv2D) | (None, 74, 74, 32) | 18464 |
| batch_normalization_4 (Batch Normalization) | (None, 74, 74, 32) | 128 |
| max_pooling2d_2 (MaxPooling2D) | (None, 37, 37, 32) | 0 |
| dropout_4 (Dropout) | (None, 37, 37, 32) | 0 |
| flatten (Flatten) | (None, 43808) | 0 |
| dense (Dense) | (None, 4) | 175236 |
| Total params: 457,252 | | |
| Trainable params: 456,420 | | |
| Non-trainable params: 832 | | |

Figura 27: Esquema del modelo generado para el conjunto COVID-19 Radiography Database.
Fuente: Elaboración propia

Una vez establecido el modelo, se procede a realizar su compilación. Este procedimiento permite definir las métricas de rendimiento y añadir las funciones de pérdida u optimizadores deseados. Para el modelo definido, se ha elegido un optimizador de tipo *Adam*, una función de pérdida de tipo *categorical_crossentropy* y unas métricas orientadas a la precisión.

Entrenamiento y evaluación del modelo

Para evaluar el rendimiento del modelo, es necesario realizar un entrenamiento del modelo con el conjunto de entrenamiento y sus respectivas etiquetas.

Se ha definido un *batch_size* de 64 muestras y se ha fijado la duración del entrenamiento en un máximo de 75 épocas. Además, se ha activado un mecanismo de *EarlyStopping* orientado a la pérdida del modelo.

Dicho mecanismo de control finaliza las iteraciones si el índice de pérdida del modelo no se reduce en cinco épocas.

Finalmente se obtendrá el valor que mejor se adecue al modelo, junto a sus respectivos pesos.

CAPÍTULO 5: RESULTADOS Y DISCUSIÓN

En este capítulo se exponen los resultados obtenidos del entrenamiento, validación y test del modelo creado en el capítulo anterior (Véase Capítulo 4)

El entrenamiento del modelo se ha realizado en 2 horas. Para valorar el rendimiento del proceso, se muestra una gráfica (Véase Figura 28) que ilustra los valores mínimos de pérdida conseguidos por el modelo en el entrenamiento y en la validación. Además, se muestra el valor máximo de precisión logrado.

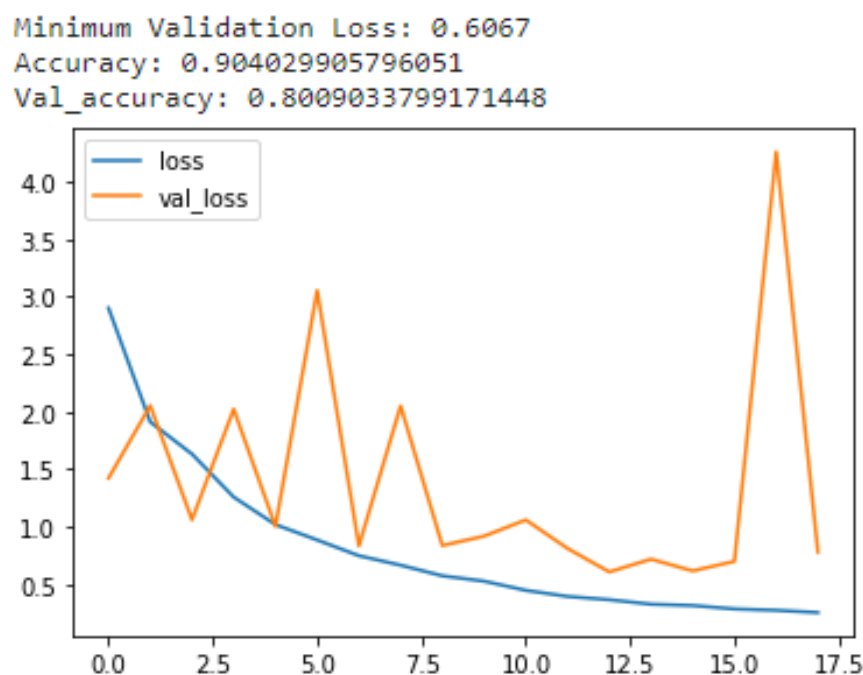


Figura 28: Resultado del entrenamiento del modelo.
Fuente: Elaboración propia

Como se puede observar en la figura 28, la curva azul, referente a la etapa del entrenamiento comienza con una pérdida inicial de 3.0 y durante su ejecución realiza un descenso hasta lograr una pérdida aproximada de 0.5

Por otro lado, la curva naranja referente a la etapa de validación comienza con una pérdida inicial de 1.5 y durante su ejecución sufre constantes cambios, incrementos y decrementos de la pérdida lograda. Es entorno a la época 12 cuando logra su mínimo, siendo este de 0.6067.

Durante el entrenamiento, el modelo consigue una precisión aproximada de 0.904 mientras que en la etapa de validación logra una precisión aproximada de 0.8.

Realizado el entrenamiento, se utiliza el método *evaluate* disponible en la librería **Keras** para evaluar el rendimiento del modelo entrenado y comparar los resultados con los obtenidos previamente.

```
180/180 [=====] - 52s 284ms/step - loss: 0.3396 - accuracy: 0.8704
Train Accuracy: 87.04%

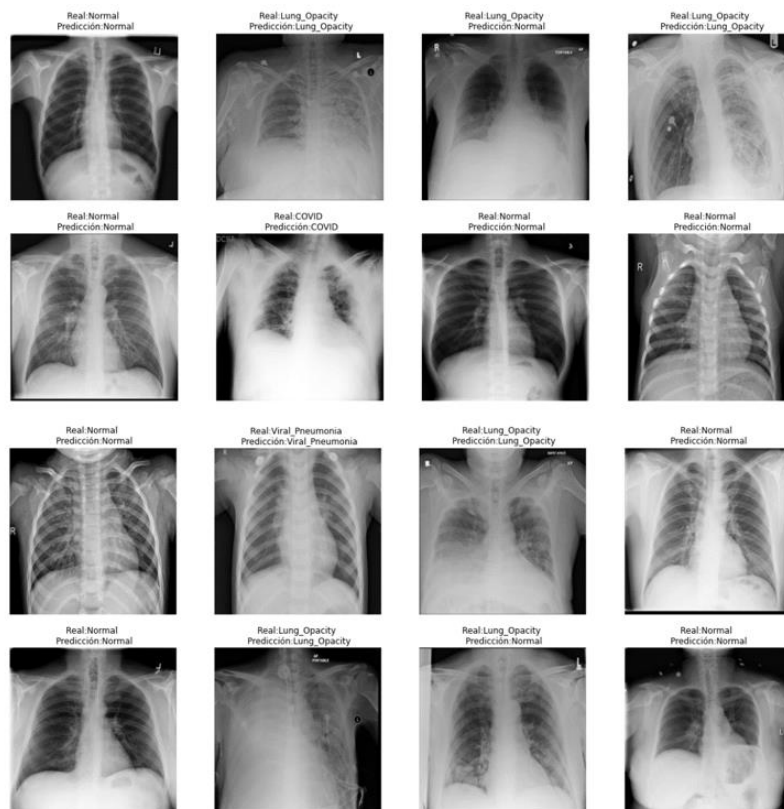
45/45 [=====] - 13s 289ms/step - loss: 0.6067 - accuracy: 0.8006
Test Accuracy : 80.06%
```

*Figura 29: Resultado del proceso de evaluación del modelo.
Fuente: Elaboración propia*

Como puede observarse en la figura 29, el modelo logra una pérdida de 0.3396 y una precisión del 87.04% con el conjunto de entrenamiento, resultados muy parecidos a los obtenidos en el proceso de entrenamiento del modelo. Por otro lado, el modelo logra una pérdida de 0.6067 y una precisión del 80.06% con el conjunto de validación, resultados idénticos a los obtenidos previamente.

Para finalizar, se ha realizado una última comprobación con la función *predict* perteneciente a la librería **Keras**. Esta función predecirá la clasificación de las imágenes que se introduzcan como entrada en el modelo.

Se ha realizado la predicción sobre imágenes no clasificadas pertenecientes al conjunto de test. En la figura 30 se muestran 16 imágenes aleatorias resultantes de este proceso.



*Figura 30: Resultado de la clasificación realizada por el modelo
Fuente: Elaboración propia*

En la figura 30 se muestran las imágenes junto a su categoría real y la categoría resultante de la predicción realizada por el modelo.

Como puede observarse, el modelo ha clasificado correctamente 14 de las 16 imágenes mostradas (tasa de acierto de 87.5%)

CAPÍTULO 6: CONCLUSIONES Y LÍNEAS FUTURAS

Este trabajo fin de carrera, tiene como objetivo principal, ampliar los conocimientos acerca de las técnicas de Deep Learning, así como la realización de formaciones encontradas en la plataforma Kaggle, para su posterior aplicación en el reto basado en la detección del COVID-19 mediante imágenes radiológicas.

La fase 3 dentro de la planificación detallada del trabajo, conforma las formaciones y cursos antes mencionados, esto tiene como objetivo principal, la familiarización con Deep Learning.

Esto permite la adquisición y el análisis sobre qué es una red neuronal convolucional básica, sus hiperparámetros, la configuración de sus capas para sus distintos procedimientos, los errores más habituales y cómo solucionarlo.

Una vez adquirido estas nociones básicas sobre redes neuronales y su funcionamiento, se procedió a realizar la clasificación de imágenes de rayos x pulmonares, dando lugar a la creación de tres redes neuronales convolucionales.

En la fase 4, durante el entrenamiento del modelo generado, se logró una precisión del 87% con los datos del conjunto de entrenamiento, posteriormente se evaluó el modelo con los datos de test obteniendo una precisión del 80%.

Para comprobar la eficiencia del modelo creado, se ha realizado la predicción sobre imágenes no clasificadas pertenecientes al conjunto de test, mostrando 16 imágenes aleatorias resultantes de este procedimiento.

Como conclusión, el modelo logró clasificar correctamente 14 de las 16 imágenes mostradas.

Futuras líneas de investigación

A partir del caso implementado basado en Deep Learning, se muestra una evidencia de que esta técnica resulta una herramienta bastante atractiva para el campo sanitario entre otros, generado un importante cambio de la productividad, costes y tiempo.

Hay que tener en cuenta que el Deep Learning y otras técnicas de IA, permitirán generar mayor conocimiento en un tiempo reducido, esto quiere decir que, cuanto más conocimiento se obtenga, aumentarán las opciones de generar nuevo conocimiento.

Actualmente, diversas investigaciones buscan obtener un resultado de estas aplicaciones que se aproximen aún más al razonamiento humano, para ello, pretenden utilizar modelos previamente entrenados de cualquier aplicación sanitaria, y mediante Transfer Learning clasificar nuevos datos de otros campos sanitarios. Este procedimiento se realiza con modelos no genéricos y, por tanto, no limitándose a los clásicos como podrían ser el VGG19 o Xception.

Probablemente se avanzará hacia escenarios de automatización de algunos procesos de investigación, permitiendo obtener nuevos conocimientos y reduciendo el tiempo y los costes del proyecto, logrando conseguir nuevas herramientas útil para los profesionales sanitarios.

CAPÍTULO 7: BIBLIOGRAFÍA

- [1] R. Confalonieri, L. Coba, B. Wagner, and T. R. Besold, "A historical perspective of explainable Artificial Intelligence," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 11, no. 1, Jan. 2021, doi: 10.1002/widm.1391.
- [2] B. Mahesh, "Machine Learning Algorithms-A Review," *International Journal of Science and Research*, 2018, doi: 10.21275/ART20203995.
- [3] B. Martín, D. Brío, and C. Serrano Cinca, "Fundamentos de las redes neuronales artificiales: hardware y software."
- [4] F. Chollet, "Deep Learning with Python," 2018.
- [5] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553. Nature Publishing Group, pp. 436–444, May 27, 2015. doi: 10.1038/nature14539.
- [6] J. Carlos, A. Pérez, D. Peluffo, and R. Therón, "Visualización y métodos kernel: Integrando inteligencia natural y artificial Optimization of the University Transportation by Contraction Hierarchies Method and Clustering Algorithms View project ExploreAT! View project," 2016. [Online]. Available: <https://www.researchgate.net/publication/313244494>
- [7] P. Larrañaga, "Redes Neuronales."
- [8] <http://grupo.us.es/gtocom/pid/pid10/RedesNeuronales.htm> (accessed Jun. 14, 2022).
- [9] I. Pérez, B. Manuel, and E. Gegúndez Arias, "DEEP LEARNING," 2021.
- [10] P. Chlap, H. Min, N. Vandenberg, J. Dowling, L. Holloway, and A. Haworth, "A review of medical image data augmentation techniques for deep learning applications," *Journal of Medical Imaging and Radiation Oncology*, vol. 65, no. 5. John Wiley and Sons Inc, pp. 545–563, Aug. 01, 2021. doi: 10.1111/1754-9485.13261.
- [11] "TensorFlow." <https://www.tensorflow.org/?hl=es-419> (accessed Jun. 14, 2022).
- [12] "Keras: the Python deep learning API." <https://keras.io/> (accessed Jun. 14, 2022).
- [13] F. J. Tarazona-Santabalbina, J. M. de la Cámara de las Heras, M. T. Vidán, and J. A. García Navarro, "Enfermedad por coronavirus 2019 (COVID-19) y edadismo: revisión narrativa de la literatura," *Revista Española de Geriatria y Gerontología*, vol. 56, no. 1, pp. 47–53, Jan. 2021, doi: 10.1016/J.REGG.2020.08.002.
- [14] G. Aguilar, G. Tamayo, M. Varela, and E. Maseda, "COVID-19: Es el momento de estar más unidos que nunca," *Revista Espanola De Anestesiologia Y Reanimacion*, vol. 67, no. 5, p. 225, May 2020, doi: 10.1016/J.RENDAR.2020.04.003.
- [15] A. Dey, R. Das, H. S. Misra, and S. Uppal, "Coronavirus disease 2019: scientific overview of the global pandemic," *New Microbes and New Infections*, vol. 38, p. 100800, Nov. 2020, doi: 10.1016/J.NMNI.2020.100800.

- [16] M. Vila Muntadas, I. Agustí Sunyer, and A. Agustí Garcia-Navarro, "COVID-19 diagnostic tests: Importance of the clinical context," *Medicina Clínica (English Edition)*, vol. 157, no. 4, pp. 185–190, Aug. 2021, doi: 10.1016/J.MEDCLE.2021.03.008.
- [17] E. Saez de Gordo, A. Portella, J. M. Escudero-Fernández, and J. Andreu Soriano, "Usefulness of chest X-rays for detecting COVID 19 pneumonia during the SARS-CoV-2 pandemic," *Radiologia (Panama)*, 2022, doi: 10.1016/J.RX.2021.11.001.
- [18] "Digit Recognizer | Kaggle." <https://www.kaggle.com/competitions/digit-recognizer> (accessed Jun. 14, 2022).
- [19] "Dogs vs. Cats | Kaggle." <https://www.kaggle.com/competitions/dogs-vs-cats> (accessed Jun. 14, 2022).
- [20] T. Rahman *et al.*, "Exploring the effect of image enhancement techniques on COVID-19 detection using chest X-ray images," *Computers in Biology and Medicine*, vol. 132, May 2021, doi: 10.1016/J.COMPBIOMED.2021.104319.
- [21] "🫁 Lungs Segmentation using U-Nets | Kaggle." <https://www.kaggle.com/code/mushfirat/lungs-segmentation-using-u-nets#7-Evaluate-Model> (accessed Jun. 14, 2022).
- [22] "Covid19 X-Ray image classification | Kaggle." <https://www.kaggle.com/code/saravananms/covid19-x-ray-image-classification/notebook> (accessed Jun. 14, 2022).
- [23] "Detection of Covid Positive Cases using DL | Kaggle." <https://www.kaggle.com/code/sana306/detection-of-covid-positive-cases-using-dl/notebook#Data-Source> (accessed Jun. 14, 2022).
- [24] "Pneumonia VS Normal Lungs Transfer learning | Kaggle." <https://www.kaggle.com/code/harshwalia/pneumonia-vs-normal-lungs-transfer-learning#Splitting-Train,-Validation-&-Test-Data%C2%B6> (accessed Jun. 14, 2022).

ANEXO

En esta sección, se muestran 4 imágenes donde se observa la creación de los directorios temporales mencionados en el capítulo 4, así como la creación del modelo y su posterior entrenamiento y validación.

```
if(os.path.isdir('../output/images/') == False):
    os.makedirs('../output/images/')
if(os.path.isdir('../output/images/COVID') == False):
    os.makedirs('../output/images/COVID')
    print("Creado directorio COVID")
if(os.path.isdir('../output/images/Lung_Opacity') == False):
    os.makedirs('../output/images/Lung_Opacity')
    print("Creado directorio Lung_Opacity")
if(os.path.isdir('../output/images/Normal') == False):
    os.makedirs('../output/images/Normal')
    print("Creado directorio Normal")
if(os.path.isdir('../output/images/Viral_Pneumonia') == False):
    os.makedirs('../output/images/Viral_Pneumonia')
    print("Creado directorio Pneumonia")

path = glob.glob('/kaggle/input/covid19-radiography-database/COVID-19_Radiography_Dataset/*/*.png')

for file in path:
    if((os.path.split(file)[1]).split('-')[0] == 'COVID'):
        dst = "../output/images/COVID/"
        if(os.path.isfile(dst + os.path.split(file)[1]) == False):
            shutil.copyfile(file, dst + os.path.split(file)[1])
    elif((os.path.split(file)[1]).split('-')[0] == 'Lung_Opacity'):
        dst = "../output/images/Lung_Opacity/"
        if(os.path.isfile(dst + os.path.split(file)[1]) == False):
            shutil.copyfile(file, dst + os.path.split(file)[1])
    elif((os.path.split(file)[1]).split('-')[0] == 'Normal'):
        dst = "../output/images/Normal/"
        if(os.path.isfile(dst + os.path.split(file)[1]) == False):
            shutil.copyfile(file, dst + os.path.split(file)[1])
    elif((os.path.split(file)[1]).split('-')[0] == 'Viral_Pneumonia'):
        dst = "../output/images/Viral_Pneumonia/"
        if(os.path.isfile(dst + 'Viral_Pneumonia-' + (os.path.split(file)[1]).split('-')[1]) == False):
            shutil.copyfile(file, dst + os.path.split(file)[1])
            os.rename(dst + os.path.split(file)[1], dst + 'Viral_Pneumonia-' + (os.path.split(file)[1]).split('-')[1])

print("Todas las imagenes han sido almacenadas")
```

Código 7: Creación de directorios temporales

Fuente: Elaboración propia

```
input_shape = (299, 299, 3)

myModel = keras.Sequential(
    [
        layers.Conv2D(128, kernel_size = (3, 3), activation="relu", padding='same', input_shape = input_shape),
        layers.BatchNormalization(),
        layers.MaxPooling2D(2,2),
        layers.Dropout(0.3),
        layers.Conv2D(128, kernel_size = (3, 3), activation="relu", padding='same', input_shape = input_shape),
        layers.BatchNormalization(),
        layers.Dropout(0.5),
        layers.Conv2D(64, kernel_size = (3, 3), activation="relu", padding='same'),
        layers.BatchNormalization(),
        layers.Dropout(0.3),
        layers.Conv2D(64, kernel_size = (3, 3), activation="relu", padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D(2,2),
        layers.Dropout(0.5),
        layers.Conv2D(32, kernel_size = (3, 3), activation="relu", padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D(2,2),
        layers.Dropout(0.2),
        layers.Flatten(),
        layers.Dense(4, activation="softmax"),
    ]
)

myModel.compile( optimizer='adam', loss="categorical_crossentropy", metrics=["accuracy"])

myModel.summary()
```

Código 8: Creación del modelo

Fuente: Elaboración propia

```

early_stopping = EarlyStopping(monitor='val_loss',
                                mode='min',
                                patience = 5 ,
                                restore_best_weights=True)

model_chkpt = ModelCheckpoint('model_1.h5', monitor='val_loss', mode='min', verbose=1, save_best_only=True)

epoch = 75

history = myModel.fit(
    image_train,
    validation_data=image_val,
    batch_size=batch_size,
    epochs=epoch,
    callbacks=[model_chkpt ,early_stopping],
    verbose=0,
)
history_df = pd.DataFrame(history.history)
history_df.loc[0:, ['loss', 'val_loss']].plot()
print("Minimum Validation Loss: {:.4f}".format(history_df['val_loss'].min()))

print(max(history.history['accuracy']))
print(max(history.history['val_accuracy']))

```

*Código 9: Entrenamiento del modelo
Fuente: Elaboración propia*

```

class_namesT = image_test.class_indices
print(class_namesT)
classesT = list(class_namesT.keys())

imagesT, labelsT = next(image_test)
labelsT = np.argmax(labelsT, axis=1)
class_dictT = image_test.class_indices
class_dict_invT = dict((v, k) for k, v in class_dictT.items())
y_names = [class_dict_invT[key] for key in labelsT]

plt.figure(figsize=(20,20))
dictCatTr = {'0': 'COVID', '1': 'Lung_Opacity', '2': 'Normal', '3': 'Viral_Pneumonia'}
ls = next(image_test)
_, labelsT = ls
labelsT = np.argmax(labelsT, axis=1)
prediction = myModel.predict(ls[0])

for i in range(16):

    pred = np.argmax(prediction[i])

    plt.subplot(4, 4, i+1)
    plt.imshow(ls[0][i])
    plt.title(f'Real:{dictCatTr[str(labelsT[i])]} \n Predicción:{dictCatTr[str(pred)]}')
    plt.axis("off")

```

*Código 10: Predicción y visualización de las imágenes
Fuente: Elaboración propia*