

Trabajo Fin de Grado Experimental

# DETECCIÓN DE NEUMONÍA UTILIZANDO MODELOS DE DEEP LEARNING EN RADIOGRAFÍAS

HENRY ALEX FUENTES BENITO

**Grado en Biotecnología**

Tutor/a: Carlo Manzo

Vic, junio de 2023

# Agradecimientos

A vosotros mis padres, Tomasa Benito y Valentín Fuentes que sois todo para mi y siempre agradecido, ya que, sin vuestra ayuda no sería posible llegar a esta meta. Y a mi hermana Erlinda Fuentes que ha sido un gran apoyo durante todo el transcurso de esta carrera hasta este proyecto.

En especial a ti papá, que desgraciadamente no puedes estar aquí en estos momentos importantes de mi vida a causa de una neumonía provocada por el virus SARS-CoV-2, siendo una inspiración para realizar este trabajo y entender mejor esta enfermedad infecciosa. Gracias por transmitirme tu fortaleza y fuerza de voluntad de seguir adelante. Te echaré siempre de menos.

A mis profesores por todo lo aprendido durante este camino, la motivación de ser biotecnólogo y la pasión por las nuevas tecnologías. Gracias especialmente a mi tutor de este proyecto Carlo Manzo por darme la posibilidad de adentrarme en un concepto hasta ahora desconocido para mí. Descubriendo la impresionante capacidad de la inteligencia artificial y con su ayuda he logrado entender y llevar a cabo la práctica aun siendo complicado debido a mis pocos conocimientos de programación.

# Resumen

La neumonía es una enfermedad infecciosa provocando inflamación a uno o ambos pulmones, siendo la causa general más común de muerte en los países con servicios médicos insuficientes. También es una de las principales causas de muerte entre niños y ancianos. Por lo que resulta necesario su diagnóstico precoz. El diagnóstico suele realizarse mediante las radiografías de rayos x de los pulmones, en la que las redes neuronales convolucionales (CNN) gracias a su capacidad de reconocimiento de imágenes, puede resultar una herramienta complementaria muy útil para su detección. Proponer modelos de aprendizaje profundo eficientes y el estudio de las CNN para detectar y clasificar la neumonía es el objetivo principal de este trabajo. En este documento se desarrollan cinco modelos para la detección de neumonía a partir de imágenes de rayos x; dos modelos propuestos entrenados desde cero (uno sin aumento de datos y otro con aumento de datos) y tres modelos previamente entrenados usando la transferencia de aprendizaje, VGG16, MobileNetV2 y ResNetV2. Los modelos se implementan y evalúan utilizando Python. Los dataset usados en el modelo fueron de dos tipos: uno para clasificación binaria (normal y neumonía) y otro para clasificación multiclase (normal, neumonía bacteriana y neumonía viral). Como métricas de evaluación se uso el valor de la precisión y perdida obtenidos de cada entrenamiento, la matriz de confusión y el método Grad-CAM. Los resultados demuestran que el aumento de datos y la transferencia de aprendizaje mejoran la precisión, siendo el modelo que usa la CNN preentrenada de MobileNetV2 el que obtiene mejores resultados con un 88,46% en la precisión con imágenes de su propio dataset, siendo similar en este caso a los otros modelos, pero con un 92,54% de precisión en la evaluación con imágenes de otro dataset habiendo una diferencia significativa al resto de modelos en la clasificación binaria. Los modelos en la clasificación multiclase obtuvieron buenos resultados en la evaluación con su propio dataset, obteniendo cerca del 80% de precisión, mientras que, en la evaluación con otro dataset, el resultado más elevado fue de MobileNetV2 con 56,01%, por lo que se considera que, en este tipo de clasificación, los modelos no realizan una clasificación adecuada. Se determinó que los modelos en la evaluación de la clasificación binaria son adecuados para clasificar pacientes normales de pacientes con neumonía, por lo que el modelo podría ayudar a los profesionales de la salud en el diagnostico de neumonía. Sin embargo, los resultados obtenidos por diferentes investigadores, muestra que se puede obtener resultados más significativos por lo que a futuro se podría mejorar los modelos usando diferentes técnicas de aprendizaje profundo, una mayor cantidad de imágenes de radiografías y probar diferentes métricas dentro del modelo.

# Summary

Pneumonia is an infectious disease causing inflammation of one or both lungs, and is the most common overall cause of death in countries with insufficient medical services. It is also one of the leading causes of death among children and the elderly. Therefore, its early diagnosis is necessary. The diagnosis is usually made by x-rays of the lungs, in which convolutional neural networks (CNN) thanks to its image recognition capacity, can be a very useful complementary tool for its detection. Proposing efficient deep learning models and the study of CNNs to detect and classify pneumonia is the main objective of this work. In this document, five models are developed for the detection of pneumonia from x-ray images; two proposed models trained from scratch (one without data augmentation and one with data augmentation) and three models previously trained using transfer learning, VGG16, MobileNetV2, and ResNetV2. The models are implemented and evaluated using Python. The dataset used in the model were of two types: one for binary classification (normal and pneumonia) and another for multiclass classification (normal, bacterial pneumonia, and viral pneumonia). As evaluation metrics, the value of precision and loss accuracy obtained from each training, the confusion matrix and the Grad-CAM method were used. The results show that the increase in data and the transfer of learning improve precision, being the model that uses the MobileNetV2 pretrained CNN the one that obtains better results with 88.46% in precision with images of its own dataset, being similar in in this case to the other models, but with 92.54% accuracy in the evaluation with images from another dataset, with a significant difference to the rest of the models in the binary classification. The models in the multiclass classification obtained good results in the evaluation with their own dataset, obtaining close to 80% accuracy, while, in the evaluation with another dataset, the highest result was MobileNetV2 with 56.01%, therefore It is considered that, in this type of classification, the models do not carry out an adequate classification. It was determined that the models in the evaluation of the binary classification are adequate to classify normal patients from patients with pneumonia, so the model could help health professionals in the diagnosis of pneumonia. However, the results obtained by different researchers show that more significant results can be obtained, so that in the future the models could be improved using different deep learning techniques, a greater number of x-ray images and testing different metrics within the model.

# Índice de Contenidos

<b>1. Introducción.....</b>	<b>1</b>
1.1. Redes neuronales .....	1
1.2. ¿Cómo funcionan las redes neuronales?.....	2
1.3. Aplicaciones de las redes neuronales .....	4
1.3.1. Campo biomédico .....	4
<b>2. Objetivos .....</b>	<b>6</b>
<b>3. Metodología.....</b>	<b>7</b>
3.1. Redes neuronales convolucionales .....	7
3.2. Función de activación.....	8
3.3. Maxpooling.....	9
3.4. Flattening .....	10
3.5. Dropout.....	11
3.6. Overfitting y underfitting.....	12
3.7. Transferencia de aprendizaje .....	12
3.8. Grad-CAM.....	13
3.9. Entorno de trabajo .....	14
3.10. Dataset .....	15
3.10.1. Dataset clasificación binaria .....	15
3.10.2. Dataset clasificación multiclase .....	15
<b>4. Resultados .....</b>	<b>17</b>
4.1. clasificación de los datos .....	17
4.1.1. Modelo binario .....	17
4.1.2. Modelo multiclase.....	17
4.2. Creación del modelo inicial .....	18
4.3. Validación del modelo .....	21
4.4. Mejora del modelo con aumento de datos .....	22
4.5. Uso de redes preentrenadas .....	25
4.6. Métricas de evaluación del desempeño .....	27
4.6.1. Matriz de confusión .....	29
4.6.1.1. Matriz de confusión de la clasificación binaria.....	29
4.6.1.2. Matriz de confusión de la clasificación multiclase.....	30

4.6.2. Visualización a través de la CNN.....	32
4.6.2.1. Grad-CAM clasificación binaria .....	33
4.6.2.2. Grad-CAM clasificación multiclase.....	34
4.7. Evaluación intercambio de datasets .....	36
<b>5. Discusión .....</b>	<b>37</b>
5.1. Comparación con otros métodos .....	38
<b>6. Conclusión.....</b>	<b>41</b>
<b>7. Bibliografía .....</b>	<b>42</b>
<b>Anexo A .....</b>	<b>i</b>
<b>Anexo B .....</b>	<b>ii</b>
<b>Anexo C .....</b>	<b>iii</b>
<b>Anexo D .....</b>	<b>iv</b>
<b>Anexo E .....</b>	<b>v</b>
<b>Anexo F.....</b>	<b>vii</b>
<b>Anexo G.....</b>	<b>ix</b>
<b>Anexo H.....</b>	<b>x</b>

# Lista de Tablas

Tabla 1: Parametros de los modelos.....	21
Tabla 2: Comparación de la evaluación de precisión y perdida de los modelos de clasificación binaria y multiclase. ....	28
Tabla 3: Comparación del tiempo de cálculo entre los modelos .....	32
Tabla 4: Comparación de la evaluación de precisión y perdida intercambiando datasets entre los modelos de clasificación binaria y multiclase .....	36
Tabla 5: Comparación con obras relacionadas en la clasificación binaria. En negrita el modelo ResNet152V2 con más porcentaje de precisión. ....	39
Tabla 6: Comparación con obras relacionadas en la clasificación multiclase. En negrita el modelo DenseNet-201 con más porcentaje de precisión. ....	40

# Lista de Figuras

Figura 1: Representación de la neurona de una red neuronal con sus diferentes partes. Entrada (X), pesos (w), salida (Y), sesgo o bias (b) y función de activación (f).....	3
Figura 2: Esquema de una red neuronal .....	3
Figura 3: Ejemplo de extracción de características de la imagen de un gato.....	7
Figura 4: Parametros de los modelos visualización de las características que toma cada capa de una CNN preentrenada (VGG16) usando la imagen de un gato. ....	8
Figura 5: Ejemplo de convolución en 2D utilizando un filtro 3x3.....	8
Figura 6: Representación 2D de las funciones de activación sigmoide y ReLU. ....	9
Figura 7: Ejemplo pictórico de Max-Pooling2D.....	10
Figura 8: Representación de aplanamiento de un mapa de características agrupado.....	11
Figura 9: Dropout aplicado a una matriz de activación durante el entrenamiento. Los cuadros con valores de 0.0 representan neuronas desactivadas.....	11
Figura 10: Representación de las tres formas distintas del modelo adaptándose a los datos ..	12
Figura 11: Grad-CAM de un pastor alemán predicho por el modelo Xception.....	14
Figura 12: Importación de las bibliotecas usadas en el proyecto .....	15
Figura 13: Repartición de las imágenes en las diferentes carpetas para la clasificación binaria .....	17
Figura 14: Código para la repartición de las imágenes en las diferentes carpetas .....	18



Figura 15: Repartición de las imágenes en las diferentes carpetas para la clasificación multiclase .....	18
Figura 16: Codigos de los parametros de entrada de las imágenes en el modelo .....	19
Figura 17: Creación del modelo con las diferentes capas .....	20
Figura 18: Compilación y entrenamiento del modelo .....	20
Figura 19: Función acc y loss para el estudio de la precisión y perdida del modelo .....	21
Figura 20: Precisión del entrenamiento (train_accuracy) y validación (test_accuracy) de la clasificación binaria (A) y multiclase (C). Perdida del entrenamiento (train_loss) y validación (test_loss) de la clasificación binaria (B) y multiclase (D) .....	22
Figura 21: Código del aumento de datos que se uso en el modelo en las imágenes de entrenamiento .....	23
Figura 22: Imágenes de radiografías generadas por el aumento de datos .....	23
Figura 23: Ultimas capas del modelo en la que se añade la capa Dropout .....	24
Figura 24: Modelos con aumento de datos. Precisión del entrenamiento (train_accuracy) y validación (test_accuracy) de la clasificación binaria (A) y multiclase (C). Perdida del entrenamiento (train_loss) y validación (test_loss) de la clasificación binaria (B) y multiclase (D) .....	24
Figura 25: Adición de las ultimas capas a la CNN preentrenada VGG16 de manera funcional .....	25
Figura 26: Esquema de transferencia de aprendizaje para clasificación multiclase.....	26
Figura 27: Precisión del entrenamiento (train_accuracy) y validación (test_accuracy) de los modelos binario y multiclase con CNN preentrenados. ....	26
Figura 28: Perdida del entrenamiento (train_accuracy) y validación (test_accuracy) de los modelos binario y multiclase con CNN preentrenados .....	27

Figura 29: Código para la evaluación de la precisión y pérdida del modelo .....	28
Figura 30: Matriz de confusión .....	29
Figura 31: Matrices de confusión para todos los modelos CNN de la clasificación binaria, (a) modelo inicial con aumento de datos, (b) VGG16, (c) ResNet50V2, (d) MobileNetV2. ....	30
Figura 32: Matrices de confusión para todos los modelos CNN de la clasificación multiclase, (a) modelo inicial con aumento de datos, (b) VGG16, (c) ResNet50V2, (d) MobileNetV2. ..	31
Figura 33: Grad-CAM de una imagen de radiografía de paciente con neumonía predecida por el modelo VGG16 .....	33
Figura 34: Grad-CAM de radiografía de paciente normal con todos los modelos propuestos en clasificación binaria.....	33
Figura 35: Grad-CAM de radiografía de paciente con neumonía con todos los modelos propuestos en clasificación binaria.....	34
Figura 36: Grad-CAM de radiografía de paciente normal con todos los modelos propuestos en clasificación multiclase .....	35
Figura 37: Grad-CAM de radiografía de paciente con neumonía bacteriana con todos los modelos propuestos en clasificación multiclase.....	35
Figura 38: Grad-CAM de radiografía de paciente con neumonía viral con todos los modelos propuestos en clasificación multiclase. ....	35
Figura 39: Mapa de calor del modelo inicial de la figura 37. ....	35

## **1. Introducción.**

La neumonía es una enfermedad de infección respiratoria aguda pulmonar que afecta a uno o ambos pulmones, la cual es un problema de salud importante siendo una de las principales causas de muerte en todo el mundo. Con una alta tasa de morbilidad y mortalidad a largo y corto plazo [1].

Puede ser causada por múltiples microorganismos (bacterias, virus y hongos) y adquirirse en la comunidad o entornos hospitalarios [2].

Aunque los síntomas iniciales pueden ser similares tanto de la neumonía bacteriana como viral, su tratamiento para cada uno es diferente. Su diagnóstico precoz es de vital importancia, por lo que se requiere un diagnóstico rápido por parte de un radiólogo experto mediante las radiografías de tórax, ya que son la manera más común y económica para la detección de la neumonía [3]. Sin embargo, a pesar de que muchas de las técnicas de diagnóstico por imagen son cada vez más precisas, es difícilmente accesible para países de bajo recurso o zonas rurales, en la que su diagnóstico se hace más lenta y difícil por lo que aumenta la tasa de mortalidad [4].

A parte, su diagnóstico puede confundirse con otras enfermedades con características similares por lo que puede resultar no claro su diagnóstico y una malinterpretación de los resultados por parte del analista [5].

Por lo tanto, estas técnicas, no se pueden usar tan fácilmente, por lo que es donde entra la Inteligencia Artificial (IA) teniendo la capacidad de revolucionar el diagnóstico y manejo de las enfermedades realizando una clasificación difícil y de inmensas cantidades para los expertos humanos. El sistema de diagnóstico por computadora, en la que el deep learning como el uso de las redes neuronales convolucionales o convolutional neural network (CNN), han sido propuestos como herramientas para ayudar al análisis por rayos x, por lo que podrían ayudar a los especialistas en su diagnóstico rápido y preciso.

En este Trabajo se pretende evaluar y analizar el uso del deep learning como las CNN como método de diagnóstico para evaluar imágenes de radiografía de tórax, de manera que el modelo creado pueda diferenciar entre pacientes normales de los que presenten neumonía, así también como intentar diferenciar dentro del grupo de neumonía, los que son provocados por virus o bacterias.

### **1.1 Redes neuronales.**

El campo de la inteligencia artificial es un término con el que posiblemente todos nos habremos cruzado en algún momento, y es que las redes neuronales se han convertido en la familia del machine learning más popular de esta última oleada que estamos viviendo. Como modelos computacionales existen desde mediados del siglo pasado [6], pero con la mejora de la técnica y la tecnología, que estos últimos años su uso se está volviendo cada vez más indispensable para nuestro día a día.

Reconocimiento de caracteres, de voz, predicción bursátil, generación de texto, traducción de idiomas, prevención de fraude, conducción autónoma, análisis genético, pronóstico de enfermedades, diagnóstico y otras muchas más aplicaciones y cada vez encontrando un nuevo uso.

El machine learning a diferencia de la IA, permite a los sistemas aprender por sí mismo, en la que aprenderá las reglas para llegar a obtener una determinada respuesta, y esta mejorara cada vez más sus reglas para dar mejores resultados, mientras que la IA, ya obtiene las reglas y da una respuesta en base a las reglas que debe respetar [7].

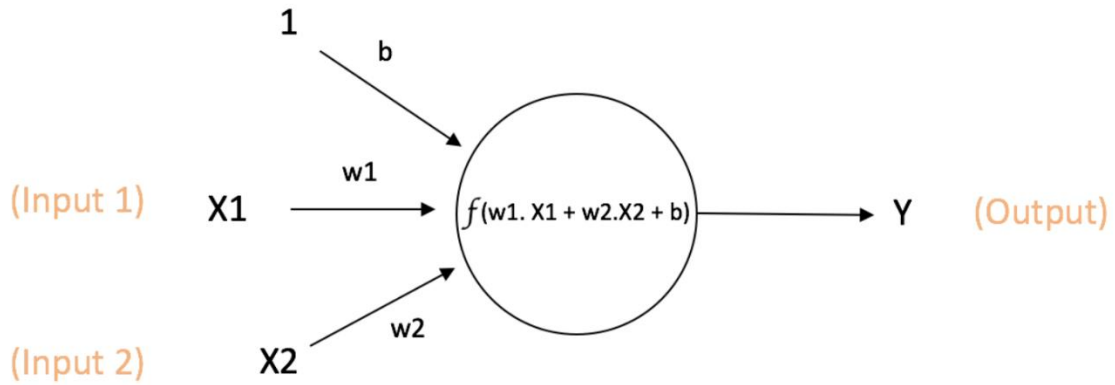
Dentro del machine learning, está el proceso llamado deep learning (aprendizaje profundo) que utiliza nodos o las neuronas interconectadas. Estas conexiones forman la red neuronal que es un modelo simplificado que emularía el funcionamiento del cerebro, si bien este término es una referencia a la neurobiología, aunque algunos modelos se desarrollan en parte inspirándose en el funcionamiento de nuestra comprensión del cerebro, los modelos de aprendizaje profundo no son modelos de cerebro [8].

## **1.2 ¿Cómo funcionan las redes neuronales?**

Se trata de una familia de algoritmos muy potentes con los que podemos modelar comportamientos inteligentes.

La complejidad de estos sistemas emerge de la interacción de muchas partes más simples trabajando conjuntamente. En este caso, a cada una de estas partes se le denomina neurona que es la unidad básica de procesamiento dentro de una red neuronal. Estas neuronas tienen conexiones de entrada a través de los que recibe estímulos externos. Con estos valores la neurona realizara un cálculo interno y generara un valor de salida [6].

Internamente la neurona utiliza todos los valores de entrada para realizar una suma ponderada en la que viene dada por el peso que se le asigna a cada una de las conexiones de entrada, en la que cada conexión tendrá asociado un valor que determinara con que intensidad cada variable de entrada afecta a la neurona como se observa en la figura 1. A parte la neurona tendrá un término independiente que dará control para mover la función que se denomina sesgo o bias, proporcionando a cada nodo un valor constante entrenable (además de las entradas normales que el nodo recibe) [9]. Se representa como otra conexión a la neurona, de manera que la neurona actuaría como el modelo de regresión lineal, sin embargo, la diferencia se dará con un último componente que se mostrara luego (función de activación). Siendo  $X$  las conexiones de entrada,  $w$  los pesos que se la asigna a cada una de las conexiones,  $b$  el sesgo e  $Y$  la salida [9].

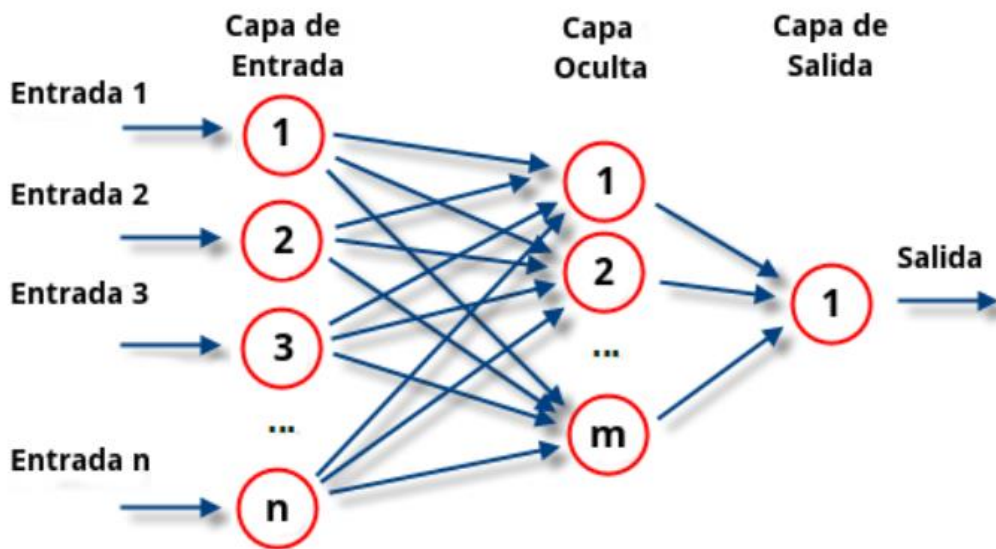


$$\text{Output of neuron} = Y = f(w1.X1 + w2.X2 + b)$$

**Figura 1:** Representación de la neurona de una red neuronal con sus diferentes partes. Entrada (X), pesos (w), salida (Y), sesgo o bias (b) y función de activación (f).

Todo lo explicado hasta ahora sobre la neurona, podría darse a entender que es básicamente como una regresión lineal, pero la función f dentro de la neurona en la figura 1, no es lineal y se conoce como función de activación.

Las neuronas se pueden encadenar entre ellas, de manera que se ordenan en capas, por lo que las neuronas que se encuentre en la misma capa recibirán la misma información de entrada de la capa anterior, y los cálculos que realicen la pasaran a la siguiente capa. Lo que daría a la estructura de lo que se conoce como red neuronal (figura 2). Entre más capas añadimos más complejo puede ser el conocimiento que elaboremos. Esta profundidad en la profundidad de capas es lo que da nombre al deep learning (aprendizaje profundo) [10].



**Figura 2:** Esquema de una red neuronal

### **1.3. Aplicaciones de las redes neuronales**

Las redes neuronales han revolucionado la IA al permitir la creación de sistemas capaces de aprender y mejorar con la experiencia, permitiendo múltiples aplicaciones que tiene en campos tan dispares como la medicina, el comercio electrónico, la robótica, la pintura o la música.

Resultan especialmente útiles en el campo de reconocimiento de imágenes donde la red neuronal puede procesar millones de imágenes en cuestión de segundos para identificar patrones y características importantes [11].

La identificación de objetos en imágenes tiene múltiples aplicaciones: desde algo tan prosaico como identificar gatos o perros en fotografías, hasta la detección de tumores en pruebas diagnósticas o clasificar las piezas de una línea de producción según su calidad [12].

#### **1.3.1. Campo biomédico**

Las aplicaciones de las redes neuronales en el campo de la medicina son múltiples y variadas:

Como a nivel dermatológico para la clasificación de cáncer de piel, en la que el uso de redes neuronales muestra potencial para clasificar las lesiones cutáneas, utilizando solo píxeles y etiquetas de enfermedades como entradas. logrando un rendimiento a la par con todos los expertos, demostrando una inteligencia artificial capaz de clasificar el cáncer de piel con un nivel de competencia comparable al de los dermatólogos [13].

Los métodos convencionales de diseño de fármacos han sido reemplazados por diseños de fármacos asistidos por ordenador en los últimos tiempos. Las redes neuronales se están utilizando ampliamente para mejorar las técnicas de diseño y el tiempo requerido de los medicamentos. Como la identificación conveniente de las proteínas diana, lo que mejora el éxito del fármaco [14].

También hay estudios que han utilizado redes neuronales para detectar diversas enfermedades de la retina analizando solamente un estudio de fondo de ojo en los pacientes. Tal es el caso de la clasificación de retinopatía diabética (RD) en sus diferentes etapas alcanzando una precisión del 96,22%, teniendo un gran alcance de predicción [15].

Cabe mencionar los estudios de su aplicación en la detección del SARS-COV-2, virus responsable de la última pandemia vivida hace poco. Siendo necesario crear soluciones para detectar a tiempo y de forma efectiva esta enfermedad. De manera que se han realizado estudios que analizan la aplicabilidad del deep learning para distinguir entre radiografías de pacientes sanos, pacientes con neumonía y aquellos que presenten una neumonía producto de la COVID-19 [16]. Varios estudios muestran que es posible detectar la neumonía provocada por COVID, obteniendo resultados significativos por encima del 90% de precisión. Sin embargo, hay que tener en cuenta que son resultados preliminares construidos sobre bases de datos concretas, por lo que es necesario probar el

modelo con radiografías de los hospitales donde se desee implementar este tipo de herramienta y así comprobar aún más su efectividad [17].

Estos serían algunos ejemplos del desempeño de las redes neuronales dentro del campo médico, demostrando que pueden ser de gran utilidad para la comunidad médica complementando la detección y el diagnóstico de diversas enfermedades.

Las capacidades del deep learning, lo convierten en una herramienta con gran potencial en la investigación y en la atención médica al igual que en otros sectores.

## 2. Objetivo

El objetivo principal de este Trabajo final de grado es estudiar el funcionamiento de las redes neuronales convolucionales y su utilidad en el reconocimiento de imágenes, más concretamente en radiografías de tórax para detectar pacientes con neumonía. Para ello se centrará en la consecución de los siguientes puntos:

Estudio de las redes neuronales convolucionales, y descripción de los mecanismos que estas llevan asociado para llevar a cabo la tarea de aprender como: el proceso de convolución, la función de activación, el Maxpooling, el flatten y el dropout entre otros.

Mitigar el problema del overfitting y underfitting de los datos con diferentes métodos.

Creación de modelos que entrenaran dos tipos de datasets diferentes. Uno aplicado a la clasificación binaria que distinguirá entre radiografías normales y con neumonía. Y otro aplicado a una clasificación multiclase, en la que las radiografías de neumonía pasaran a distinguirse entre neumonía viral o neumonía bacteriana. De manera que distinguiremos tres clases: normales (class0), neumonía bacteriana (class1) y neumonía viral (class2).

Uso de tres tipos de redes preentrenadas (ResNet50V2, MobileNetV2 y VGG16) para los dos tipos de datasets de clasificación binaria y multiclase. Haciendo una comparación de que método ofrece mejores resultados

Se aspira a que el porcentaje de acierto obtenido sea elevado y que la red evalúe con precisión.



### 3. Metodología

#### 3.1. Redes neuronales convolucionales.

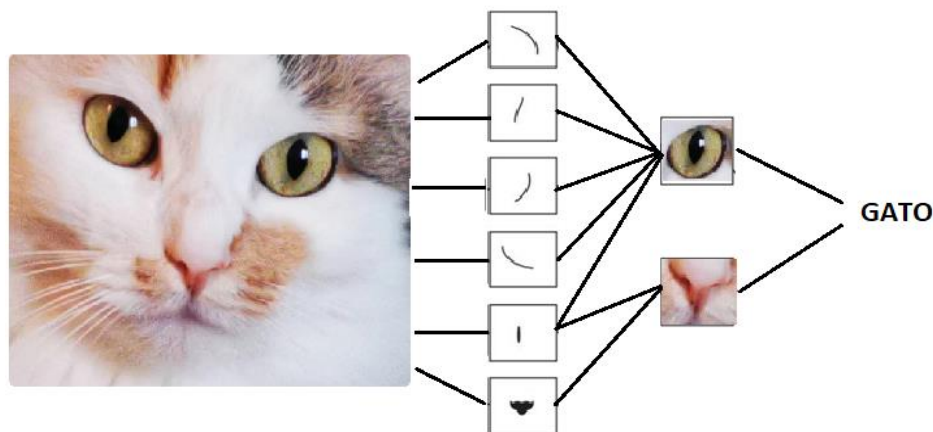
Se realizó una breve descripción de las redes neuronales más simples, sin embargo, este trabajo usara redes que van más allá de estas características presentadas. Así surge la necesidad de trabajar con las redes neuronales convolucionales (CNN) o convnets.

Gran parte de la revolución que se ha vivido en los últimos años dentro del deep learning se lo debemos en sus inicios a los avances del campo de la visión por ordenador, en concreto al desarrollo de un tipo de red neuronal muy importante que está especializada para trabajar con imágenes, las famosas CNN [18].

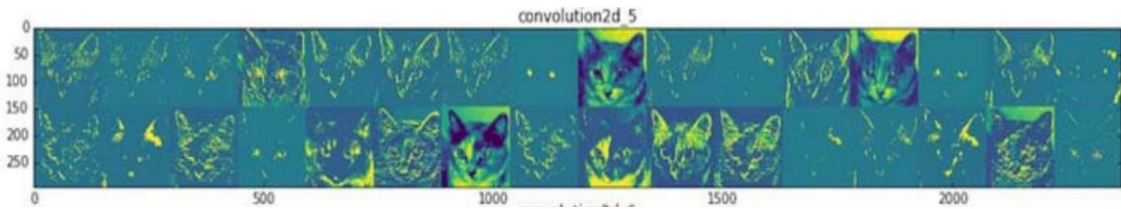
Su importancia viene de poder descifrar los patrones más complejos en enormes datasets de imágenes, dotando de ojos a maquinas que tanto pueden observar rostros de personas, radiografías de pacientes, como los peatones que se cruzan ante un coche autónomo, consiguiendo que las maquina puedan percibir el mundo que los rodea [19].

Se caracteriza por aplicar un tipo de capa donde se realiza una operación matemática que se conoce como convolución. Para ello, la CNN contiene varias capas ocultas especializadas y con una jerarquía: esto significa que las primeras capas detectan propiedades o formas básicas y se van especializando hasta llegar a capas más profundas capaces de reconocer formas complejas como un rostro o una silueta. A diferencia de las capas densas que aprenden patrones globales en su espacio de características de entrada, las capas convolucionales aprenden patrones locales [8].

Pueden aprender jerarquías espaciales de patrones. Una primera capa de convolución aprenderá pequeños patrones locales como bordes, una segunda capa de convolución aprenderá patrones más grandes hechos de las características de las primeras capas, y así sucesivamente como se muestra en la figura 3 [8].

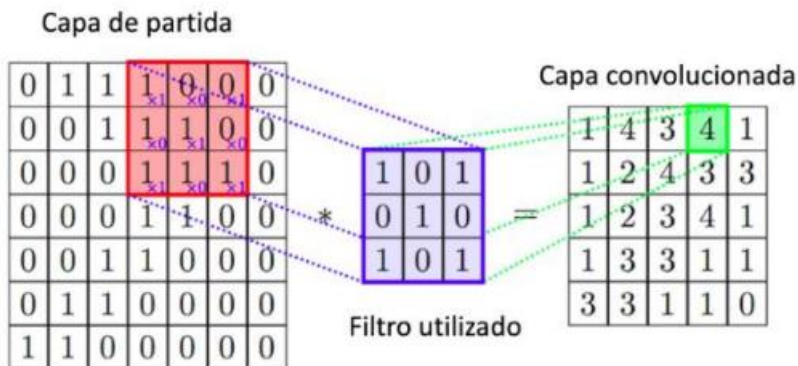


**Figura 3:** ejemplo de extracción de características de la imagen de un gato



**Figura 4:** visualización de las características que toma cada capa de una CNN preentrenada (VGG16) usando la imagen de un gato.

La convolución consistirá en tomar grupos de píxeles cercanos de la imagen de entrada e ir operando matemáticamente contra una pequeña matriz a la que se denomina kernel conocida como filtro. En el caso de este proyecto, las imágenes serán escalas de grises y las entradas serán de 250 x 250 de alto y de ancho. Pero en este caso, a pesar de que las imágenes sean grises, las presentamos en la entrada como si fuera a color, por lo que se añadirían 3 canales y quedaría 250 x 250 x 3. Ese kernel recorrerá todas las neuronas de entrada, de izquierda a derecha y de arriba hacia abajo, generando una nueva matriz de salida, que será la nueva capa de neuronas ocultas, y que también se conoce como la matriz de activación como se visualiza en la figura 5 [20].



**Figura 5:** Ejemplo de convolución en 2D utilizando un filtro 3x3.

A medida que el kernel se va desplazando obtenemos una nueva imagen filtrada por este. Una vez la imagen realiza una convolución con un kernel, aplica la función de activación.

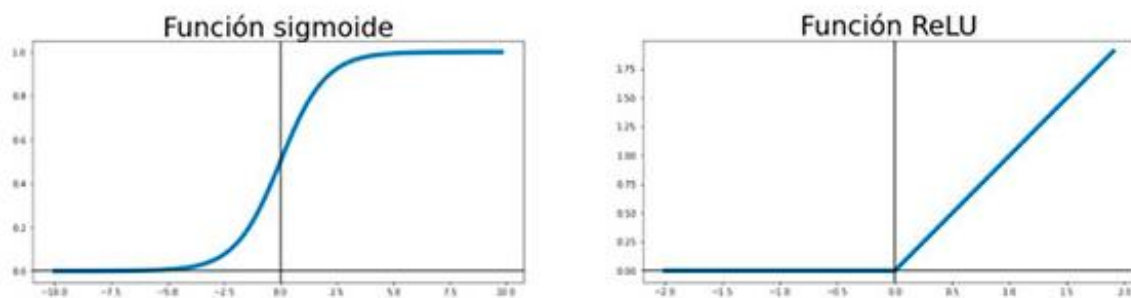
### 3.2 Función de activación

Su función es introducir la no linealidad en la salida de la neurona. Estas neuronas necesitan aprender estas representaciones no lineales ya que la mayoría de los datos del mundo real no son lineales.

Básicamente si en nuestra neurona lo que hacíamos era calcular como valor de salida una suma ponderada de nuestras entradas, lo que queremos hacer ahora es pasar dicho valor de salida por la función de activación, lo que distorsionará nuestro valor de salida, añadiéndole deformaciones no lineales. Para que así podamos encadenar de forma efectiva la computación de varias neuronas [21].

¿Cómo son estas deformaciones? Pues depende de la función de activación. En la que cada una está representada por un número y realiza una determinada operación matemática sobre ella. Existen varias funciones de activación pero en este trabajo nos centraremos en 3 tipos: ReLU, sigmoide y softmax. Que son las que se usaran en el modelo de nuestra red neuronal.

- **ReLU:** La unidad lineal rectificadora (ReLU) es una de las más Populares. Se aplica al número real de una entrada y la ajusta al umbral de cero (sustituye cualquier valor negativo por cero). Proporciona transformación no lineal, pudiéndose usar suficientes rectificadores lineales para aproximar funciones no lineales arbitrarias. Es una función usada en las capas ocultas de nuestra red neuronal, NO en las de salida [21].
- **Sigmoide:** Se aplica al número real de una entrada y lo ajusta a un valor entre 0 y 1, convirtiendo variables independientes de rango casi infinito en probabilidades simples entre 0 y 1, y la mayor parte de su salida estará muy cerca de estos valores. Son utilizadas para sistemas de etiquetado binario, donde el resultado es una probabilidad o resultado binario. Por lo que, si queremos obtener clasificaciones binarias por salida, utilizaremos esta función [21].
- **Softmax:** Es una generalización de la función de activación sigmoide, siendo apropiada para resultados categóricos ya que maneja sistemas de etiquetado multiclase. Por ejemplo, supongamos que nuestra respuesta categórica tiene diez clases; Con esta función de activación calculamos una probabilidad para cada categoría (la suma de las diez categorías es una) y clasificamos a un individuo en particular. en la clase con mayor probabilidad [21].



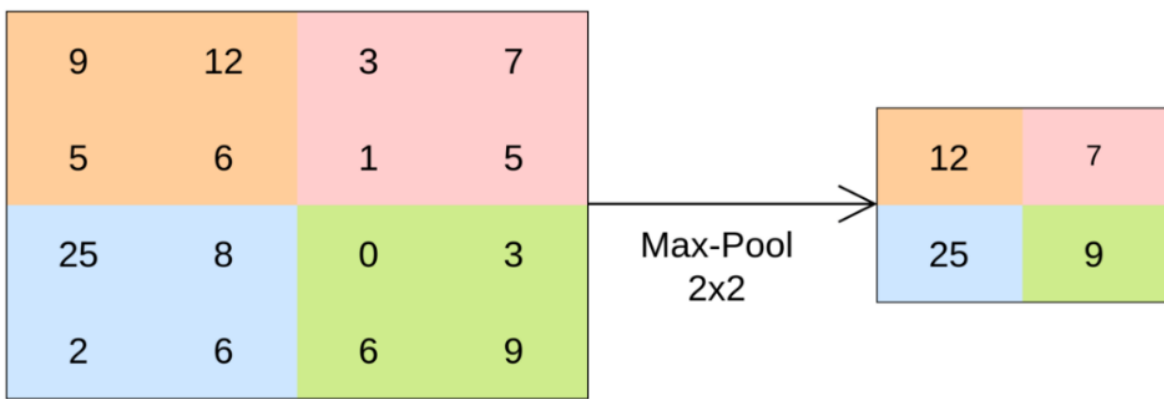
**Figura 6:** Representación 2D de las funciones de activación sigmoide y ReLU.

### 3.3 Maxpooling

Terminado el proceso anterior, se tiene que reducir el número de neuronas antes de pasar a la siguiente convolución. Como se dijo anteriormente, partíamos de una entrada de imágenes 250 x 250 x 3, por lo que, si se realiza una nueva convolución a partir de la siguiente capa, el número de neuronas crecería exponencialmente implicando un mayor procesamiento.

Aquí es donde entra el método del subsampling para reducir el tamaño de la próxima capa, reduciendo el tamaño de las imágenes filtradas en la que conservara las características más importantes que detectó cada filtro [20].

Hay diversos tipos, pero el que usaremos es el Maxpooling2D, una operación de agrupación máxima que calcula el valor máximo en cada parche y el mapa de características. En nuestro caso usaremos una Maxpooling2D de 2 x 2 que recorrerá cada una de las imágenes de características obtenidas anteriormente de 250 x 250, pero en lugar de un píxel, tomará dos de ancho y dos de alto (2 x 2) preservando el valor más alto de esos cuatro pixeles (figura 7). De esta manera, teóricamente conservara los valores más importantes reduciendo considerablemente el número de neuronas que pasara a la siguiente convolución [20].

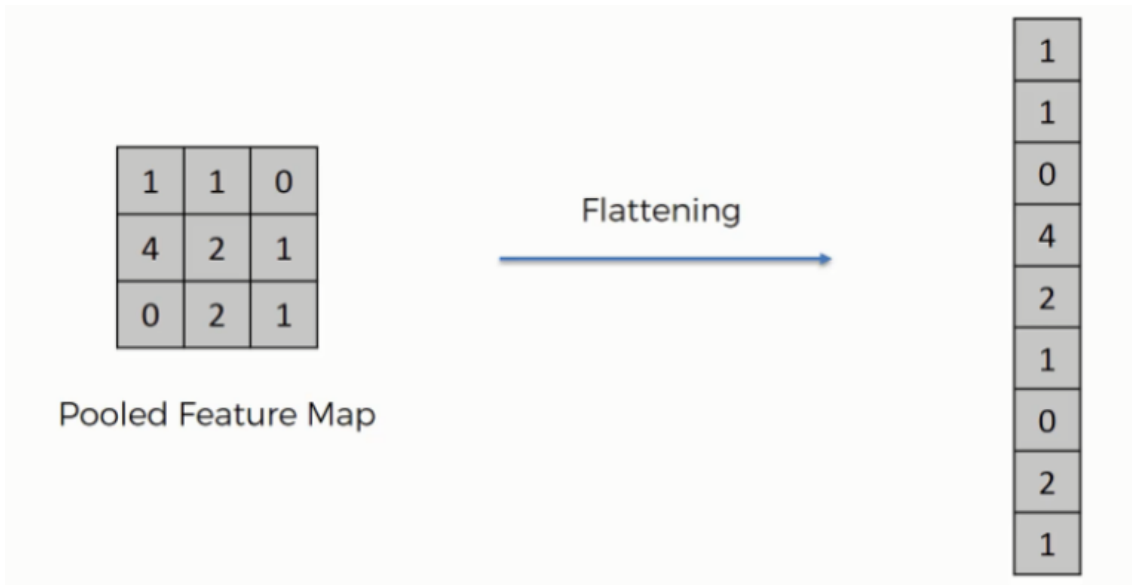


**Figura 7:** Ejemplo pictórico de Max-Pooling2D de 2 x 2 preservando los valores más altos

De esta manera la primera capa de convolución detectara características globales como líneas o curvas. Mientras que en la siguiente capa y a medida que se vayan usando más capas convolucionales, estas irán reconociendo formas más precisas y complejas.

### 3.4 Flattening

Este paso es bastante simple, ya que solo consiste en aplanar. Acabado los pasos anteriores, tendríamos un mapa de entidades agrupado. Por lo que será necesario aplanarlo en una columna como en la figura 8 [22].



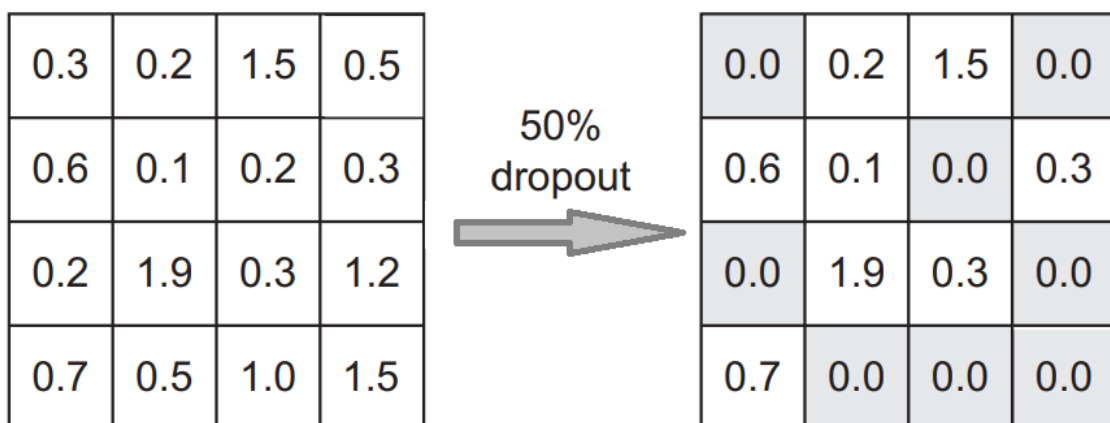
**Figura 8:** Representación de aplanamiento de un mapa de características agrupado.

Esto es necesario debido a que más adelante insertaremos estos datos en una red neuronal artificial para procesarlos aún más.

### 3.5. Dropout.

Es una de las técnicas de regularización más efectivas y utilizadas para redes neuronales. Consiste en desactivar aleatoriamente un número determinado de neuronas en una red neuronal. La tasa de dropout es la fracción de las funciones que se ponen a cero; generalmente se establece entre 0.2 y 0.5, lo que indica el porcentaje de neuronas que queremos que se desactiven aleatoriamente durante el entrenamiento del modelo. (figura 9) [8].

Este hecho ayuda a reducir el overfitting con lo que las neuronas necesitan trabajar mejor de forma solitaria para no depender tanto de las relaciones con las neuronas vecinas



**Figura 9:** Dropout aplicado a una matriz de activación durante el entrenamiento. Los cuadros con valores de 0.0 representan neuronas desactivadas.

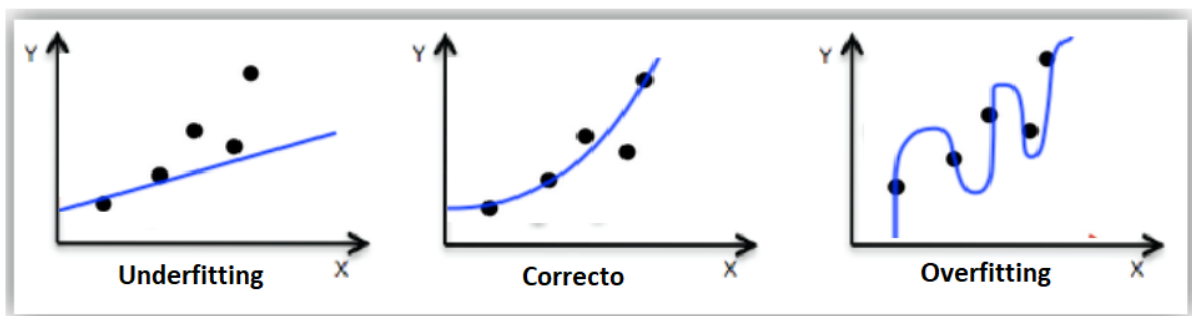
### 3.6. Overfitting y underfitting.

El principal problema durante el entrenamiento es el overfitting y el underfitting de los datos (figura 10). Podemos traducir estos términos como sobreajuste y subajuste respectivamente. Hacen referencia al fallo en el modelo al generalizar o encajar el conocimiento que pretendemos que adquiera.

Después de un cierto número de iteraciones en los datos de entrenamiento, la generalización deja de mejorar y las métricas de validación se estancan y luego comienzan a degradarse, por lo que el modelo comienza a sobre ajustarse, ya que está comenzando a aprender patrones que son específicos de los datos de entrenamiento pero que son engañosos o irrelevantes cuando se trata de datos nuevos [23].

Si nuestros datos de entrenamiento son muy pocos nuestra máquina no será capaz de generalizar el conocimiento y estará incurriendo en underfitting [8].

Ambos problemas son malos para el aprendizaje ya que no permite que nuestra maquina generalice el conocimiento y no nos darán buenas predicciones. Durante el diseño de nuestro modelo se explicará que métodos influirán para disminuir estos problemas.



**Figura 10:** Representación de las tres formas distintas del modelo adaptándose a los datos.

### 3.7. Transferencia de aprendizaje.

Al aprender a usar redes neuronales aplicándolos a proyectos nuestros, nos acabamos topando con dos problemas. Conseguir suficientes datos para entrenar nuestras redes y entrenar redes neuronales con muchas capas y millones de parámetros, rápidamente se vuelve muy tardado o costoso. Por lo que la técnica que nos puede ayudar a obtener mejores resultados es la transferencia de aprendizaje.

Esta técnica consiste en utilizar una CNN preentrenada. Los pesos del modelo previamente entrenado en algún conjunto de datos, como ImageNet, generalmente en una tarea de clasificación de imágenes a gran escala, se utilizan para resolver un problema relacionado con un conjunto de datos relativamente más pequeño. Si este conjunto de datos original es lo suficientemente grande y general, entonces la jerarquía espacial de características aprendidas por la red preentrenada puede actuar efectivamente como un modelo genérico del mundo visual, por lo tanto, sus características pueden resultar útiles

para muchos problemas diferentes de visión por computadora, incluso aunque estos nuevos problemas pueden involucrar clases completamente diferentes a las de la tarea original [8].

Además, se ha encontrado que una red previamente entrenada funciona mejor que una red entrenada desde cero. Por lo que la CNN con transferencia de aprendizaje juega un papel interesante en la clasificación, y es más rápido y eficiente que desarrollar soluciones de clasificación en modelos desde cero.

Por lo tanto, en este proyecto se ha optado por usar tres modelos de CNN preentrenados basados en el concepto de transferencia de aprendizaje los cuales son: VGG16, ResNet50V2 y MobileNetV2.

- **VGG16:** La arquitectura VGG fue desarrollada por un equipo de la Universidad de Oxford. Dado que la red VGG16 ha recibido una amplia capacitación, puede proporcionar una excelente precisión incluso si el conjunto de datos de imagen es pequeño. VGG16 ha dado el nombre debido a las 16 capas en su arquitectura [24].
- **MobileNetV2:** es una red neuronal convolucional con 53 capas de profundidad. Arquitectónicamente, MobileNetV2 introduce cuellos de botella lineales entre capas y conexiones de acceso directo entre los cuellos de botella. Codifican entradas y salidas intermedias del modelo, y las capas internas se utilizan para encapsular el modelo desde conceptos de bajo nivel (por ejemplo, píxeles, etc.) hasta descriptores de alto nivel (por ejemplo, categorías de imágenes, etc.) [25].
- **ResNet50V2:** es una red neuronal convolucional con 50 capas de profundidad. ResNet50V2 es una versión modificada de ResNet50 que funciona mejor que ResNet50 y ResNet101 en el conjunto de datos de ImageNet. En ResNet50V2, se realizó una modificación en la formulación de propagación de las conexiones entre bloques [26].

Estos modelos pueden cargar una versión preentrenada de la red entrenada en más de un millón de imágenes desde la base de datos de ImageNet. La red preentrenada puede clasificar imágenes en 1000 categorías de objetos (por ejemplo, teclado, ratón, lápiz y animales). Como resultado, la red ha aprendido representaciones ricas en características para una gran gama de imágenes [27].

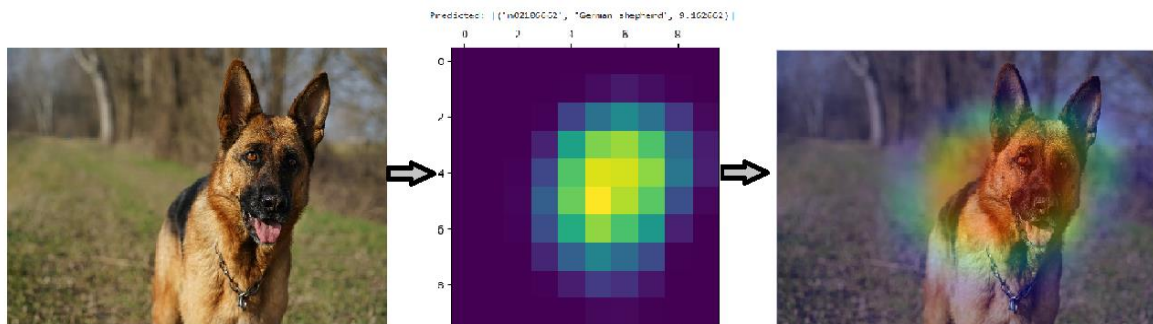
### **3.8. Grad - CAM.**

Las representaciones aprendidas por las CNN son altamente susceptibles de visualización, en gran parte porque son representaciones de conceptos visuales. Desde 2013, se ha desarrollado una amplia serie de técnicas para visualizar e interpretar estas representaciones [8]. De los cuales en este trabajo se mostrará uno de los más accesibles y útiles que es el método Grad - CAM.

Este método es útil para comprender qué partes de una imagen se identificaron como pertenecientes a una clase determinada, lo que le permite localizar objetos en imágenes.

Esta categoría general de técnicas se denomina visualización de mapas de activación de clases (CAM), y consiste en producir mapas de calor de activación de clases sobre imágenes de entrada [28].

Como por ejemplo lo que visualizamos en la figura 11, en la que se evaluó la imagen de pastor alemán mediante el modelo Xception, indicando las partes más importantes de la imagen que tuvo en cuenta para determinar la clase de perro, lo que llevó a la CNN a su decisión de clasificación final. [29].



**Figura 11:** Grad-CAM de un pastor alemán predicho por el modelo Xception, se genera un mapa de calor y se superpone en la imagen original [30].

### 3.9. Entorno de Trabajo

Para realizar este proyecto se utilizó Google Collaboratory ("Colab"), que ha permitido ejecutar el código a través del navegador. Permite a cualquier usuario escribir y ejecutar código arbitrario de Python en el navegador. Es especialmente adecuado para tareas de aprendizaje automático, análisis de datos y educación.

Destacamos el hecho de que nos otorga acceso a la potencia de procesamiento (CPU/GPU/TPU) de las máquinas de Google Research

Python fue elegido como lenguaje de programación para el desarrollo de este proyecto. Ya que la API (application programming interface) que se usará es Keras. Es una API de redes neuronales escrita en lenguaje Python. Se trata de una biblioteca de código abierto que se ejecuta sobre frameworks como Theano y TensorFlow. Además de ser un lenguaje sencillo a la hora de programar.

Las bibliotecas utilizadas en este proyecto se han presentado en la figura 12



```
import tensorflow as tf
import os, shutil
import numpy as np
import tensorflow_datasets as tfds
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import cv2
import matplotlib.cm as cm
```

**Figura 12:** Importación de las bibliotecas usadas en el proyecto

### 3.10. Dataset.

Los datos utilizados en este proyecto se obtuvieron de Kaggle. Una plataforma web que reúne la comunidad Data Science más grande del mundo con un gran número de base de datos e imágenes de libre acceso [31].

A parte también esta plataforma permite a los especialistas de datos y a otros desarrolladores participar en concursos y data challenges de Machine Learning, escribir y compartir códigos y guardar conjuntos de datos.

#### 3.10.1. Dataset clasificación binaria.

Los datos para la clasificación binaria consistieron en radiografías de tórax pediátricas con 5,856 radiografías marcadas como neumonía o normales. Con un peso de 1GB en formato JPEG [32].

Se almacenaron en mi cuenta personal de google drive, la cual se vincula con Colab para trabajar directamente los datos desde ahí.

En la carpeta del conjunto de datos se encuentran las carpetas Train (entrenamiento) y validation (validacion) que contienen la imagen subagrupada según el diagnóstico: normal y neumonía. La ruta de trabajo general era `"/content/drive/MyDrive/radiografia/chest_x_ray"`

#### 3.10.2. Dataset clasificación multiclase.

Los datos para la clasificación multiclase consisten en radiografías de tórax con 4672 imágenes. Con un peso de 1GB en formato JPEG, en la que la neumonía pasa a ser de dos tipos, causadas por una infección bacteriana o por una infección viral. son en escala de grises con varios tamaños [24]. Hay 3 clases de imágenes:

- Clase 0: sin enfermedad
- Clase 1: neumonía bacteriana
- Clase 2: neumonía viral

En la carpeta del conjunto de datos se encuentran la carpeta Train\_images (4672 imágenes) y test\_images (1168 imágenes) que contienen las imágenes de las tres clases, sin estar separadas en subgrupos. La ruta de trabajo general era `"/content/drive/MyDrive/chest_x_ray"`

El archivo contiene las etiquetas de clase de las imágenes en formato labels\_train.csv, referente a la carpeta train\_images de las cuales:

- 1227 imágenes pertenecen a la clase 0
- 2238 imágenes pertenecen a la clase 1
- 1207 imágenes pertenecen a la clase 2

## 4. Resultados

### 4.1. Clasificación de los datos

#### 4.1.1. Modelo binario

De las 5856 imágenes, los datos importados ya se habían agrupado inicialmente de la siguiente manera:

- Train: imágenes clasificadas como entrenamiento. Estas son las imágenes utilizadas inicialmente para entrenar con 5232 imágenes
- Validation: en esta carpeta están las imágenes para evaluar el modelo, siendo 624 imágenes

Se creo la carpeta test\_1 donde se pasó 20 imágenes de la carpeta train (diez imágenes normales y diez de neumonía), de manera que train quedo con 5212 imágenes. Esto se hizo como ejemplo para probar el modelo y visualizar cada imagen con su respectiva clasificación de predicción y visualizarlo mediante el Grad-CAM como se verá más adelante en los resultados.

Las imágenes quedaron repartidas como se muestra en la siguiente figura 13.

```
[7] print('total train_normal images:', len(os.listdir("/content/drive/MyDrive/radiografia/chest_x_ray/train/NORMAL")))
print('total train_pneumonia images:', len(os.listdir("/content/drive/MyDrive/radiografia/chest_x_ray/train/PNEUMONIA")))
print('total validation_normal images:', len(os.listdir("/content/drive/MyDrive/radiografia/chest_x_ray/validation/NORMAL")))
print('total validation_pneumonia images:', len(os.listdir("/content/drive/MyDrive/radiografia/chest_x_ray/validation/PNEUMONIA")))
print('total test_1_normal_predict images:', len(os.listdir("/content/drive/MyDrive/radiografia/chest_x_ray/test_1/NORMAL")))
print('total test_1_pneumonia images:', len(os.listdir("/content/drive/MyDrive/radiografia/chest_x_ray/test_1/PNEUMONIA")))

total train_normal images: 1339
total train_pneumonia images: 3873
total validation_normal images: 234
total validation_pneumonia images: 390
total test_1_normal_predict images: 10
total test_1_pneumonia images: 10
```

**Figura 13:** Repartición de las imágenes en las diferentes carpetas para la clasificación binaria.

#### 4.1.2. Modelo multiclase

Debido a que en la carpeta train\_images estaban todas las imágenes (las tres clases), se tuvo que separar estas imágenes según su clase indicada en el archivo label\_train.csv. Para eso se crearon dos carpetas: train\_images y validation\_images. Dentro de cada una de estas, se crearon tres subcarpetas: Class0, Class1 y Class2. Se prosiguió a coger aleatoriamente el 80% de las imágenes de la carpeta inicial de train\_images para pasarlas dentro de las subcarpetas class0, class1 y class2 contenidas en la nueva carpeta train\_images. El 20% restante se pasó a la carpeta validation\_images de la misma manera según su clase, como se muestra el código usado en la figura 14.

```
[ ] split_r = 0.8
for i, (label, filename) in enumerate(zip(labels,filenames)):
    print(f"Saving image {i+1}/{train_dataset_size}", end="\r")
    label = label
    im = plt.imread(os.path.join(train_images, filename), 'jpg')
    split = "train_images" if np.random.rand()<=split_r else "validation_images"
    plt.imsave(f"/content/drive/MyDrive/chest_x_ray/{split}/class{label}/{filename}", im)
```

**Figura 14:** Código para la repartición de las imágenes en las diferentes carpetas

Al igual que en el modelo binario se creó la carpeta test\_1 donde se pasó 30 imágenes de la carpeta train (diez imágenes class0, diez imágenes class1 y diez imágenes class2), quedando la carpeta train\_images con 3730 imágenes.

De manera que las imágenes quedaron repartidas como se muestra en la figura 14.

```
[ ] print('total train_class0 images:', len(os.listdir("/content/drive/MyDrive/chest_x_ray/train_images/class0")))
print('total train_class1 images:', len(os.listdir("/content/drive/MyDrive/chest_x_ray/train_images/class1")))
print('total train_class2 images:', len(os.listdir("/content/drive/MyDrive/chest_x_ray/train_images/class2")))
print('total validation_class0 images:', len(os.listdir("/content/drive/MyDrive/chest_x_ray/validation_images/class0")))
print('total validation_class1 images:', len(os.listdir("/content/drive/MyDrive/chest_x_ray/validation_images/class1")))
print('total validation_class2 images:', len(os.listdir("/content/drive/MyDrive/chest_x_ray/validation_images/class2")))
print('total test_class0 images:', len(os.listdir("/content/drive/MyDrive/chest_x_ray/test_1/class0")))
print('total test_class1 images:', len(os.listdir("/content/drive/MyDrive/chest_x_ray/test_1/class1")))
print('total test_class2 images:', len(os.listdir("/content/drive/MyDrive/chest_x_ray/test_1/class2")))

total train_class0 images: 958
total train_class1 images: 1810
total train_class2 images: 962
total validation_class0 images: 259
total validation_class1 images: 418
total validation_class2 images: 235
total test_class0 images: 10
total test_class1 images: 10
total test_class2 images: 10
```

**Figura 15:** Repartición de las imágenes en las diferentes carpetas para la clasificación multiclase.

## 4.2. Creación del modelo inicial

La red neuronal por sí misma ha de reconocer una gran cantidad de imágenes para que pueda captar las características únicas de cada objeto y a su vez poder generalizarlo. Cada imagen se trata de una matriz de píxeles cuyo valor va de 0 a 255 pero se normaliza para la red neuronal de 0 a 1 (rescale=1. /255). Debido a las diferencias en las dimensiones de las imágenes en el conjunto de datos, todas las imágenes del conjunto de datos se han cambiado de tamaño a 224 x 224 píxeles (target\_size) para la entrada a las diferentes arquitecturas creadas, así como también para los modelos preentrenados (VGG16, MobileNetV2 y ResNet50V2) que se verán en el apartado 4.5. Esto sería lo que se llama preprocesamiento de la imagen. El batch size que indica la cantidad de imágenes que tendrá cada lote que ira entrenando el modelo es de 32 y finalmente el class\_mode en el caso de clasificación binaria es binario y en la clasificación multiclase es categorical. Todos estos parámetros se adjudicaron a los datos tanto de entrenamiento como de validación (figura 16).

```
[ ] train_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224,224),
    batch_size=32,
    class_mode="binary")

validation_datagen = ImageDataGenerator(rescale=1./255)
validation_generator = validation_datagen.flow_from_directory(
    validation_images,
    target_size=(224,224),
    batch_size=32,
    class_mode = "binary")

Found 5212 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
```

**Figura 16:** Codigos de los parametros de entrada de las imágenes en el modelo

Para entrenar el modelo inicial, se utilizó la función secuencial, a la que se agregaron seis capas convolucionales a través de la clase Conv2D (32, 64, 128 y 256), y cuando se normalizaron los datos, se usó la función "ReLU" como activador.

Una forma común de mitigar el sobreajuste es imponer restricciones a la complejidad de una red al obligar a sus pesos a tomar solo valores pequeños, lo que hace que la distribución de los valores de peso más regular. Esto se llama regularización de peso, que en este caso se usó la regularización L2. L2(0.001) significa que cada coeficiente en la matriz de peso de la capa agregará  $0.001 * \text{valor\_coeficiente\_peso}$  a la pérdida total de la red. Tenga en cuenta que debido a que esta penalización solo se agrega en el momento del entrenamiento, la pérdida de esta red será mucho mayor en el momento del entrenamiento que en el momento de la prueba

La función MaxPooling2D es responsable de reducir la muestra en ancho y alto (2D) al final de cada capa añadida. En última instancia, se agrega la capa flatten (figura 17).

Finalmente, se utilizó la función Dense de la unión de las capas, con 512 unidades, como capa de salida y la función de activación sigmoide, usada para casos binarios dando como salida dos valores como se explicó anteriormente respecto a esta función de activación, y la función de activación softmax se usó en el caso de multiclase, ya que esta cuenta con tres valores de salida (class0, class1 y class2)

```

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(224, 224, 3)))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), kernel_regularizer=regularizers.l2(0.001), activation='relu'))
model.add(layers.Conv2D(64, (3, 3), kernel_regularizer=regularizers.l2(0.001), activation='relu'))
model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))

model.add(layers.Conv2D(128, (3, 3), kernel_regularizer=regularizers.l2(0.001), activation='relu'))
model.add(layers.Conv2D(128, (3, 3), kernel_regularizer=regularizers.l2(0.001), activation='relu'))
model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))

model.add(layers.Conv2D(256, (3, 3), kernel_regularizer=regularizers.l2(0.001), activation='relu'))

model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))

model.add(layers.Flatten())

model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

```

**Figura 17:** Creación del modelo con las diferentes capas.

Como función de pérdida, para el caso binario se usó `binary_crossentropy`, usada habitualmente para modelos de clasificación binaria y el modelo se entrenó durante 30 épocas (`epochs = 30`) que determinan el número de veces que el algoritmo trabajará con todos los datos de entrenamiento. Mientras que con el modelo multiclase se usó `categorical_crossentropy` como función de pérdida y se entrenó durante 40 épocas, debido a que al tener que aprender otra clasificación más, se optó por añadir más épocas (figura 18).

```

[ ] model.compile(loss='binary_crossentropy',
                  optimizer=optimizers.legacy.RMSprop(learning_rate=0.0005),
                  metrics=['accuracy'])
history = model.fit(
    train_generator,
    epochs=30,
    validation_data= validation_generator
)

```

**Figura 18:** Compilación y entrenamiento del modelo.

Los parámetros de precisión del modelo se estudiaron y se mostraron para su posterior evaluación. Las funciones `acc` y `loss` se utilizaron para el estudio de precisión, como se muestra en la figura 19.

```
# Accuracy

acc = training_model.history['accuracy']
val_acc = training_model.history['val_accuracy']

loss = training_model.history['loss']
val_loss = training_model.history['val_loss']
```

Figura 19: Estudio de la precisión del modelo.

Los parámetros utilizados para entrenar los modelos se muestran en la tabla 1. En la que la función de activación para el modelo binario fue sigmoid y para el modelo multiclase se usó softmax. Las épocas fueron 30 y el class mode binary para el modelo binario, y épocas de 40 y class mode categorical para el modelo multiclase.

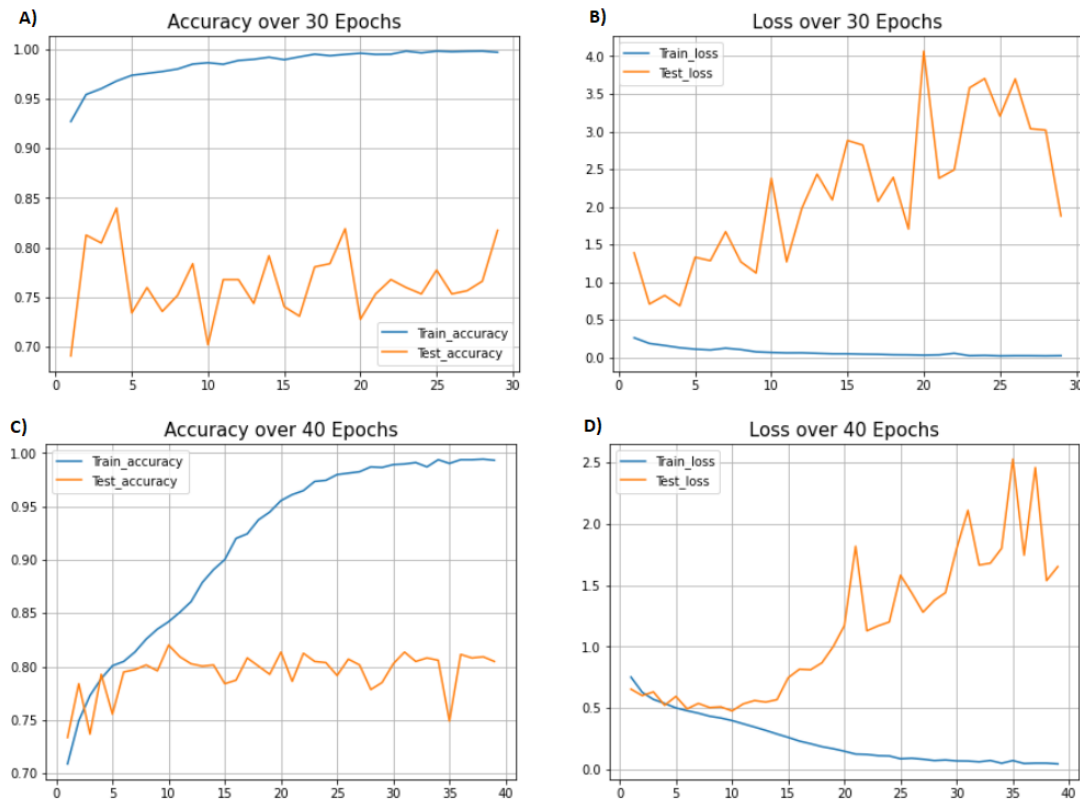
Parámetros	Modelo inicial	VGG16	ResNet50V2	MobileNetV2
Batch size	32	32	23	32
Layers	13	16	50	53
Optimizer	RMSprop	RMSprop	RMSprop	RMSprop
Parameters (M)	16.433	21.138	23.564	18.314
Activation Funtion	Sigmoid/Softmax	Sigmoid/Softmax	Sigmoid/Softmax	Sigmoid/Softmax
Learning rate	0.0005	0.0005	0.0005	0.0005
Custom input size	224 x 224	224 x 224	224 x 224	224 x 224
Epochs	30/40	30/40	30/40	30/40
Class mode	Binary/categorical	Binary/categorical	Binary/categorical	Binary/categorical

Tabla 1: Parámetros de los modelos.

### 4.3. Validación del modelo

La figura 20 muestra la relación entre precisión y perdida (entrenamiento y validación) según la evolución de la época de ambos modelos. Si observamos la relación entre la precisión del entrenamiento y la precisión de validación, en el modelo binario (figura 20 A) no muestran la misma tendencia, mientras que en el modelo multiclase muestran la misma tendencia hasta que se alcanza la época 10 (figura 20 C), momento en el que las curvas divergen; Para la precisión del entrenamiento, en ambos modelos tiende a aumentar, pero no ocurre lo mismo con la precisión de validación, que llega a disminuir

y aumentar dentro del mismo rango durante las épocas de entrenamiento (0,7 a 0,82 modelo binario y 0,75 a 0,81 en modelo multiclase). En esta misma figura, la relación de la pérdida entre entrenamiento y validación comienza con una tendencia descendente similar hasta la época 10 en el modelo multiclase (figura 20 D), mientras que el modelo binario no muestra la misma tendencia, llegando a aumentar (figura 20 B). En ambos modelos la pérdida de validación aumenta nuevamente hasta superar los niveles iniciales. Por otro lado, la pérdida de entrenamiento continúa con la tendencia a la baja.



**Figura 20:** Precisión del entrenamiento (train\_accuracy) y validación (test\_accuracy) de la clasificación binaria (A) y multiclase (C). Pérdida del entrenamiento (train\_loss) y validación (test\_loss) de la clasificación binaria (B) y multiclase (D).

#### 4.4. Mejora del modelo con aumento de datos.

El modelo de deep learning que utilizamos en el apartado anterior fue un modelo sin aumento de datos (en los siguientes apartados se procederá a nombrarlo modelo inicial -) y esta investigación requiere una gran cantidad de datos para ser entrenado de manera efectiva. Sin embargo, el tamaño de los datos de entrada no es lo suficientemente grande. El sobreajuste puede ocurrir en conjuntos de datos pequeños. El sobreajuste es una situación en la que el modelo no puede identificar patrones desconocidos. En esta situación, se aplicó una técnica de aumento de datos para aumentar sintéticamente el tamaño de los datos sobre los datos originales, y también ayudar a reducir el sobreajuste durante el entrenamiento de CNN por lo que se creó un modelo con aumento de datos (en los siguientes apartados se lo nombrará modelo inicial +). En este trabajo, se utilizan las siguientes técnicas para aumentar la imagen: (1) rango de rotación, (2) cambio de ancho

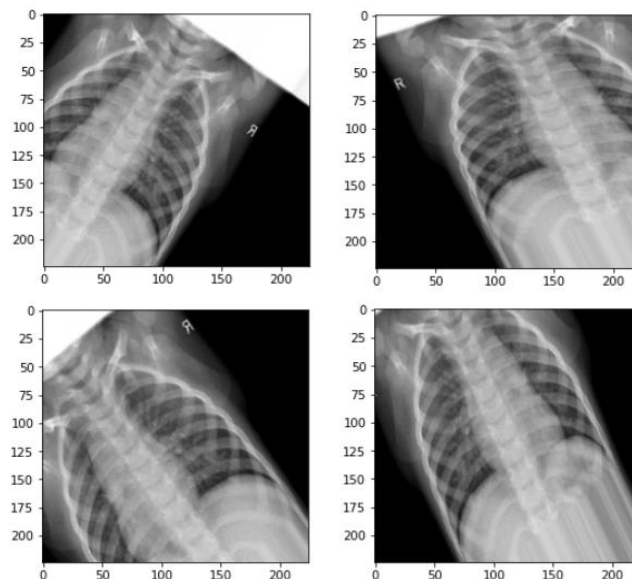


y de altura, (3) aumento de zoom y (4) cambio de sentido de la imagen como se muestra en la figura 21. El aumento de datos fue el mismo para ambos modelos.

```
[ ] train_datagen = ImageDataGenerator(  
    rescale=1./255,  
    rotation_range=40,  
    width_shift_range=0.2,  
    height_shift_range=0.2,  
    shear_range=0.2,  
    zoom_range=0.3,  
    horizontal_flip=True)  
  
validation_datagen = ImageDataGenerator(rescale=1./255)  
  
train_generator = train_datagen.flow_from_directory(  
    train_dir,  
    target_size=(224, 224),  
    batch_size=32,  
    class_mode="binary")  
  
validation_generator = validation_datagen.flow_from_directory(  
    validation_images,  
    target_size=(224, 224),  
    batch_size=32,  
    class_mode='binary')
```

**Figura 21:** Código del aumento de datos que se usó en el modelo en las imágenes de entrenamiento.

El aumento de datos solo se usa en las imágenes de la carpeta de entrenamiento (figura 22), ya que estas son las que el modelo usara para saber que parámetros tomar en cuenta para detectar la neumonía. Las imágenes de validación se dejan igual, ya que queremos que evalúe tal cual las imágenes que le mostraremos.



**Figura 22:** Imágenes de radiografías generadas por el aumento de datos.

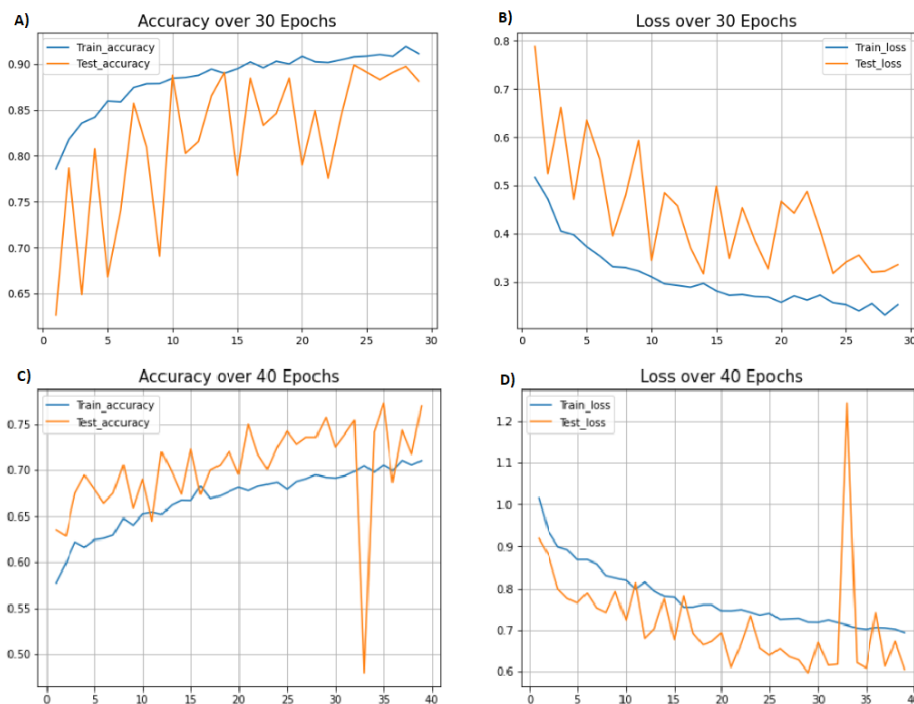
Por último, después de la capa flatten, se agregó la capa de dropout de 0,5, lo que indica que el modelo ira desactivando aleatoriamente el 50% de las neuronas durante el entrenamiento, como ya se explicó anteriormente (figura 23)

```
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

**Figura 23:** Ultimas capas del modelo en la que se añade la capa Dropout.

Con el aumento de datos, se observa una mejora en ambos modelos. Se observa en la figura 24 la relación entre la precisión del entrenamiento y validación tiende a aumentar a medida que aumenta el número de épocas. De hecho, no siguen un aumento logarítmico idéntico ya que la precisión de validación presenta una tendencia más "irregular" con más aumentos y disminuciones dependiendo de la época en el modelo binario (figura 24 A), en comparación del modelo multiclase (figura 24 C). Además, en el modelo multiclase se observa que la tendencia de la validación va por encima del entrenamiento, en la que se observa una enorme caída en la época 33 para luego seguir aumentando.

En esta misma figura 24, la relación de la pérdida entre entrenamiento y validación comienza con una tendencia a la baja similar mantenida hasta la última época. En la que el modelo binario (figura 24 B) presenta una tendencia más irregular que el modelo multiclase (figura 24 D). Se observa un gran aumento en la época 36 en el gráfico de pérdida de validación para luego seguir bajando (figura 24 D).



**Figura 24:** Modelos con aumento de datos. Precisión del entrenamiento (train\_accuracy) y validación (test\_accuracy) de la clasificación binaria (A) y multiclase (C). Pérdida del entrenamiento (train\_loss) y validación (test\_loss) de la clasificación binaria (B) y multiclase (D).

#### 4.5. Uso de redes preentrenadas.

A parte del aumento de datos, otro método de mejorar el entrenamiento es el uso de redes preentrenadas como se explicó en el apartado 3.7. En este caso el propósito es usar tres modelos preentrenados diferentes para proponer un modelo de aprendizaje profundo para la clasificación de neumonía. Se compararán los resultados que se obtenga de cada uno comprobando cual puede ser una mejor opción.

Tanto para el modelo binario y multiclase, se usó los siguientes modelos preentrenados: VGG16, ResNet50V2 y MobileNetV2

A diferencia de los primeros modelos creados, que se usó model secuencial, en este caso para los modelos se hace de manera funcional, como se observa en la figura 25, con el modelo VGG16. Esto es debido a que de esta manera más adelante se podrá observar en el mapa de calor, lo que los modelos toman en cuenta dentro de la imagen para hacer la clasificación. De esta manera podemos tener acceso a la última capa de los modelos preentrenados y visualizar la salida de la última capa para determinar qué tipo de imagen es.

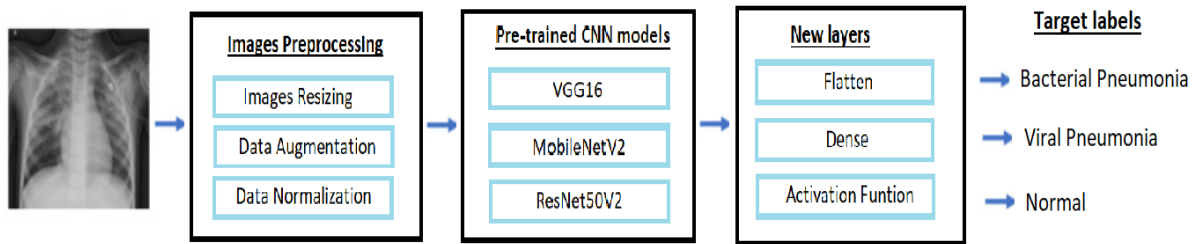
```
[ ] from tensorflow.keras.applications import VGG16
conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(224, 224, 3))
x = conv_base.output #(in your case pre_trained model is efficientnet-b3)
flatten = layers.Flatten()(x)
dense1 = layers.Dense(256, activation='relu')(flatten)
pred = layers.Dense(1, activation='sigmoid')(dense1)
model = Model(inputs= conv_base.input, outputs=pred)
model.summary()
```

**Figura 25:** Adición de las ultimas capas a la CNN preentrenada VGG16 de manera funcional.

Se conservo las capas base convolucionales y se personalizo la capa de clasificación final agregando nuevos conjuntos de capas, como dos capas densas, una capa flatten, la funcion de activacion ReLU, como se observa en la figura anterior (figura 25) así como en el anexo A se puede visualizar como ejemplo las capas que tiene el modelo VGG16 y las capas nuevas añadidas. El nuevo conjunto de capas consiste en una capa plana en la parte superior, que transforma los datos de la capa anterior en un vector de datos unidimensional. La parte de clasificación consta de dos capas densas. La primera capa densa consta de 256 neuronas con función de activación RELU y la salida final es producida por una capa densa con dos neuronas que utiliza la función de activación Sigmoid para la clasificación binaria (1 indicando que tiene dos salidas), mientras que para la clasificación multiclase, la salida final es producida por una capa densa con tres neuronas que utiliza la función de activación Softmax (3 indicando que tiene tres salidas). En esta etapa, solo se entrenan las capas adicionales que añadimos, mientras que los otros pesos de los modelos permanecen congelados.

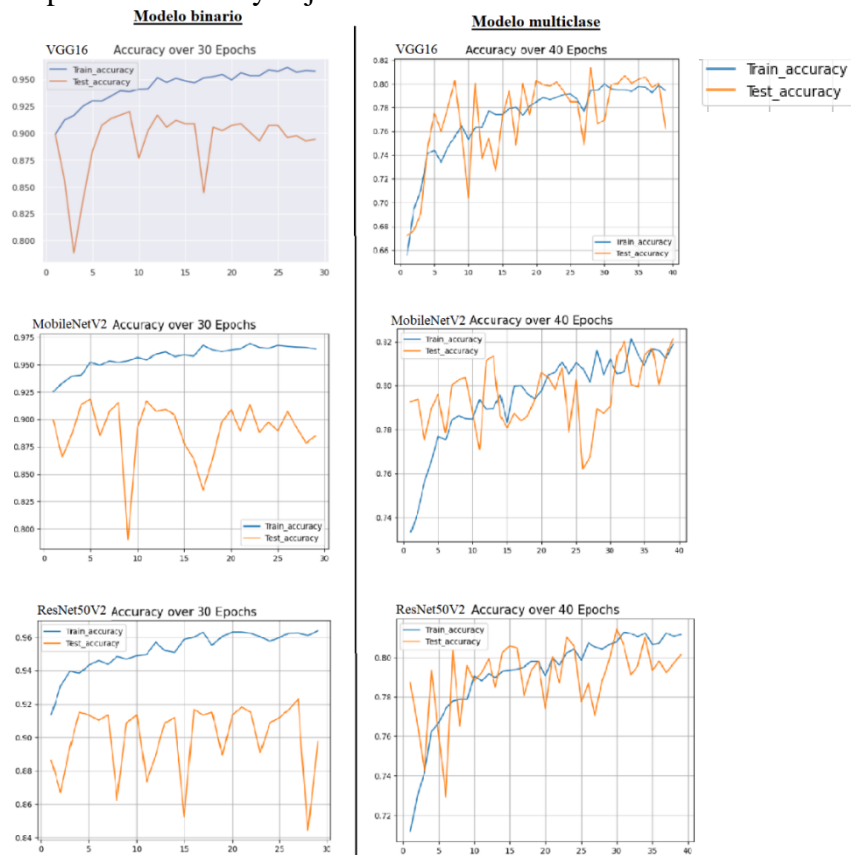
El procesamiento de entrada de las imágenes es igual que como se hizo en la parte de aumento de datos.

El procedimiento de la transferencia de aprendizaje se muestra en la figura 26.



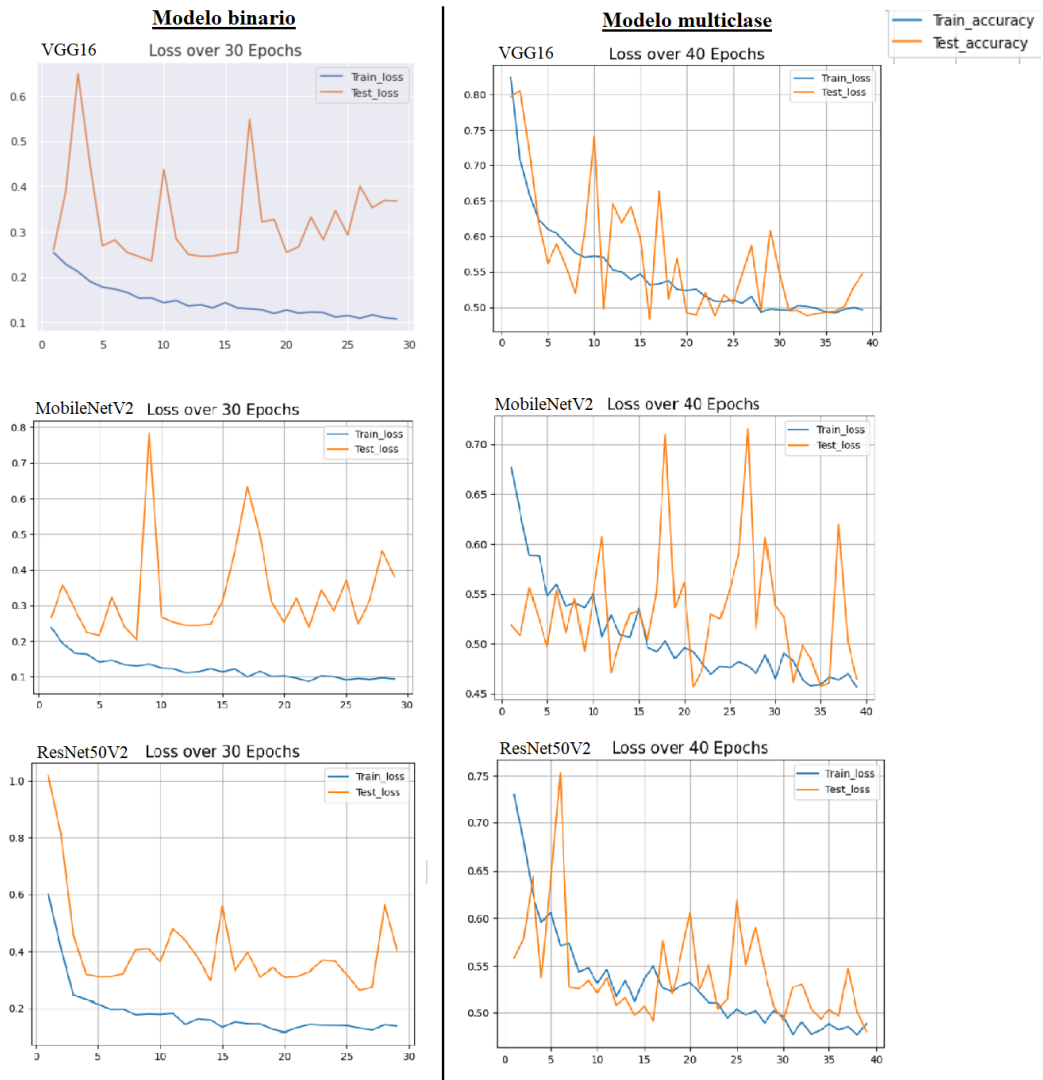
**Figura 26:** Esquema de transferencia de aprendizaje para clasificación multiclase

En la figura 27 el modelo binario no presenta una relación entre la precisión del entrenamiento y la precisión de la validación en los modelos, no sigue una tendencia, llegando a su máximo en la época 10 en el modelo VGG16, para después disminuir poco su valor de precisión. En el anexo B se muestra el valor de precisión y pérdida que tiene VGG16 en cada época de entrenamiento. A lo largo de las épocas hacen subidas y bajadas dentro de un mismo rango. Mientras que en el modelo multiclase hay una misma tendencia ascendente en los tres modelos entre el entrenamiento y validación, aunque de forma irregular con picos de subida y bajada.



**Figura 27:** Precisión del entrenamiento (train\_accuracy) y validación (test\_accuracy) de los modelos binario y multiclase con CNN preentrenados.

Con respecto a la relación de pérdida del entrenamiento y validación mostrado en la figura 28, en los modelos binarios no se observa una relación. Durante las primeras épocas descienden, pero acaban presentando picos de subida y bajada manteniéndose por un mismo rango. El modelo multiclase si presenta una relación descendente entre el entrenamiento y validación, aunque de forma irregular con grandes picos de subida y bajada, llegando a situarse ambos cerca del mismo valor en la última época.



**Figura 28:** Pérdida del entrenamiento (train\_accuracy) y validación (test\_accuracy) de los modelos binario y multiclase con CNN preentrenados.

#### 4.6. Métricas de evaluación del desempeño.

En esta sección, mencionamos diferentes métricas de evaluación utilizadas para evaluar la efectividad del modelo de CNN propuesto. Los modelos se probaron en las imágenes de la carpeta de validación con 624 imágenes para la clasificación binaria y 912 imágenes para la clasificación multiclase. El rendimiento de cada modelo se evalúa en función de diferentes métricas, como la precisión y pérdida, mediante una matriz de confusión y el mapa de calor.

Para obtener los valores de precisión y pérdida de cada modelo se utilizó el siguiente código de la figura 29 en la que se evalúa las imágenes de validación aplicando la normalización (rescale=1./255). Para el caso binario el class mode es igual a binary y para el modelo multiclase es igual a categorical. En el anexo C se muestra los resultados obtenidos en la evaluación de cada modelo de la precisión y pérdida.

```
[ ] test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_directory(
    validation_images,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary')
test_loss, test_acc = model.evaluate(test_generator)
print("test loss", test_loss)
print('test acc:', test_acc)
```

**Figura 29:** Código para la evaluación de la precisión y pérdida del modelo.

A continuación, en la tabla 2 se muestra los datos de la precisión y pérdida de cada modelo.

		N.º imágenes de evaluación	Test_acc (Precisión)	Test_loss (Pérdida)
<b>CLASIFICACIÓN BINARIA</b>	Modelo inicial -	624	81,73%	1,87
	Modelo inicial +	624	88,14%	0,33
	VGG16	624	89,42%	0,36
	ResNet50V2	624	89,74%	0,40
	MobileNetV2	624	88,46%	0,38
<b>CLASIFICACIÓN MULTICLASE</b>	Modelo inicial -	912	80,48%	1,65
	Modelo inicial +	912	76,97%	0,76
	VGG16	912	76,20%	0,54
	ResNet50V2	912	80,15%	0,48
	MobileNetV2	912	82,12%	0,46

**Tabla 2:** Comparación de la evaluación de precisión y pérdida de los modelos de clasificación binaria y multiclase. Modelo inicial - corresponde a sin aumento de datos y modelo inicial + con aumento de datos.

#### 4.6.1. Matriz de confusión.

Se eligió la matriz de confusión para obtener los valores de las métricas de rendimiento. La matriz de confusión se muestra en la figura 30. Estas métricas se calcularon utilizando varios parámetros de la matriz de confusión, como TP, FP, TN y FN, que representan verdadero positivo, falso positivo, verdadero negativo y falso negativo.

- TP indica el número de pacientes positivos correctamente predichos,
- FN indica pacientes infectados identificados incorrectamente como casos normales
- FP indica pacientes normales pronosticados con neumonía.
- TN indica pacientes sanos correctamente predichos.

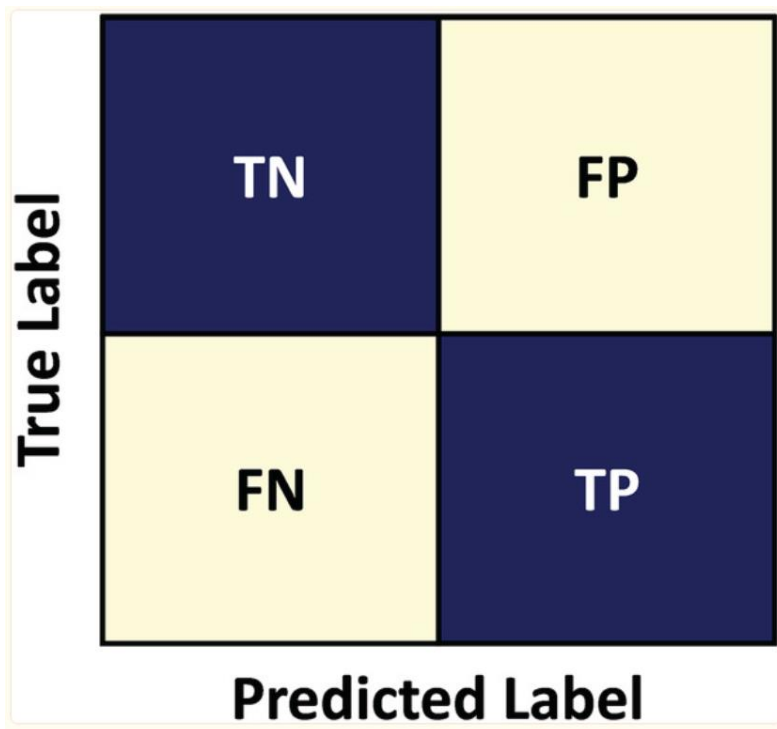


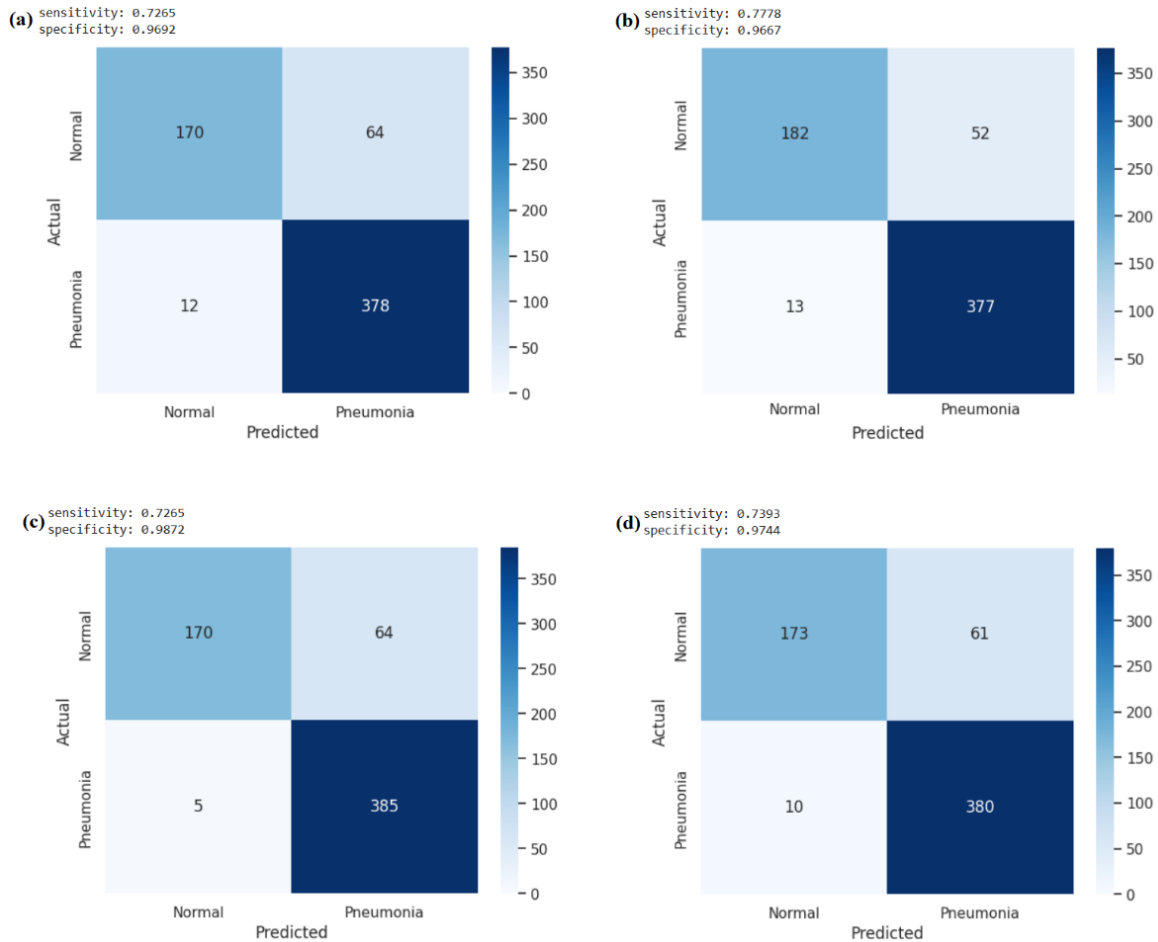
Figura 30: Matriz de confusión.

##### 4.6.1.1. Matriz de confusión de la clasificación binaria.

La matriz de confusión es la principal herramienta para evaluar errores en problemas de clasificación. Muestra el número de imágenes reconocidas correcta e incorrectamente por el modelo [34]. Construimos la matriz de confusión de las cuatro arquitecturas propuestas en el estudio para evaluar el rendimiento (modelo inicial con aumento de datos, VGG16, ResNet50V2 y MobileNetV2), como se muestra en la Figura 31. El número de variables de salida también decide el tamaño de la matriz. Entonces, en este caso, la matriz de confusión para la clasificación binaria será 2 x 2 y para la clasificación multiclase 3 x 3.

Las imágenes evaluadas en la clasificación binaria constan de 234 imágenes de pacientes normal y 390 imágenes de pacientes con neumonía. Se observa en la figura 31 que los

modelos tienen un número de aciertos similar entre ellos en la predicción, siendo las imágenes de neumonía donde obtienen más fallos al clasificarlos.

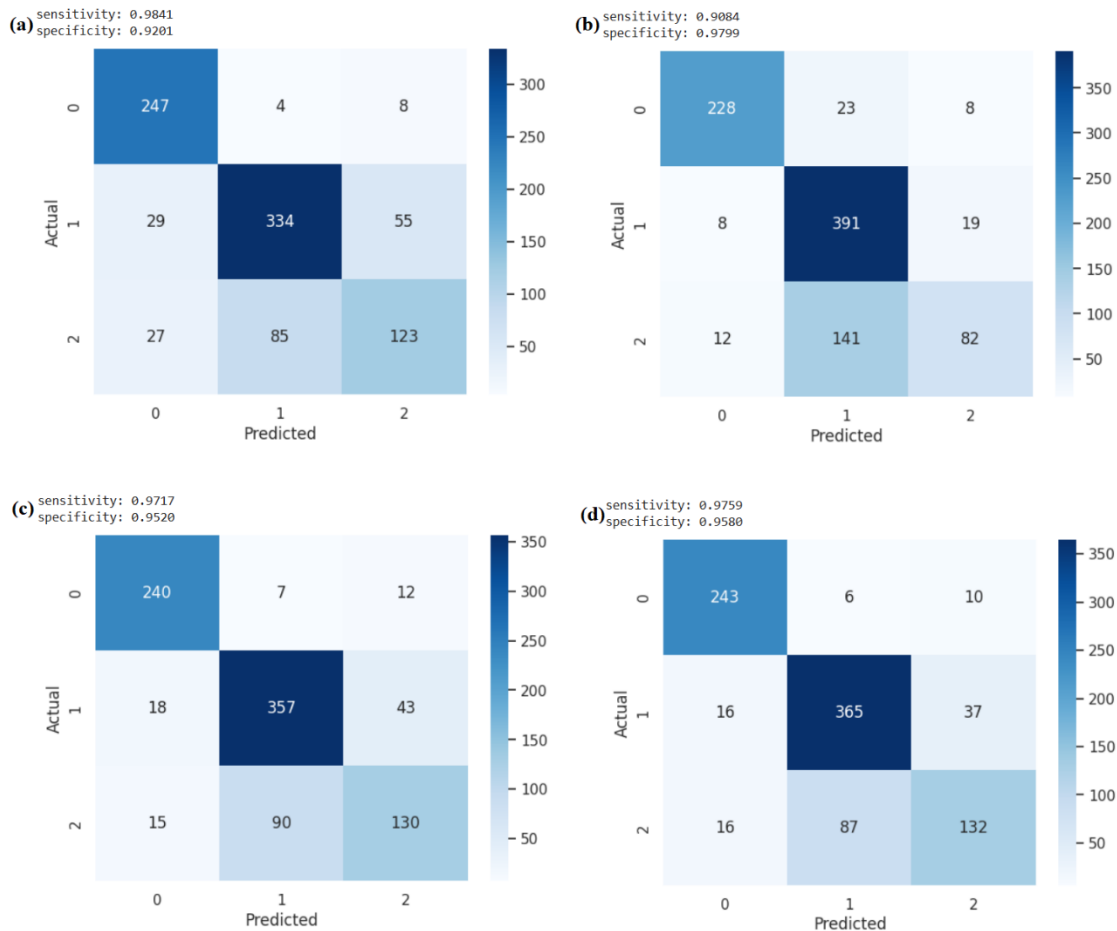


**Figura 31:** Matrices de confusión para todos los modelos CNN de la clasificación binaria, (a) modelo inicial con aumento de datos, (b) VGG16, (c) ResNet50V2, (d) MobileNetV2.

#### 4.6.1.2. Matriz de confusión de la clasificación multiclase.

Las imágenes evaluadas en la matriz de confusión de la clasificación multiclase constan de: 259 imágenes class0, 418 imágenes class1 y 235 imágenes class3. Se puede observar en la figura 32 que el número de aciertos de la class0 (paciente normal) en todos los modelos es mayor que el resto de las clases, en la que hay una mayor dificultad de distinguir los pacientes con neumonía viral (class2) que acaba prediciéndolas como pacientes con neumonía bacteriana (class1). Los códigos con los que se hizo la matriz de confusión se encuentran en el anexo D.





**Figura 32:** Matrices de confusión para todos los modelos CNN de la clasificación multiclase, (a) modelo inicial con aumento de datos, (b) VGG16, (c) ResNet50V2, (d) MobileNetV2.

También se realizaron comparaciones en términos de costo computacional. Tabla 3 muestra el costo computacional de los modelos en términos de tiempo total de entrenamiento y prueba y tiempo por época (en segundos). Cabe mencionar que el proceso se hizo con GPU. El tiempo es mayor en todas las fases del modelo multiclase, debido a que agregamos más épocas (como se mencionó anteriormente), más imágenes de entrenamiento y validación y tomar en cuenta las tres clases. En ambos tipos de modelo, el que menos tiempo requiere es el modelo inicial sin aumento de datos (modelo -) debido a que ocupa menos parámetros a diferencia del resto, y al ser un modelo más sencillo seguido del modelo con aumento de datos (modelo+).

Clasificación binaria	Tiempo de entrenamiento (s)	Tiempo de prueba (s)	Tiempo por época (s)
Modelo inicial –	2430	7	74 - 88
Modelo inicial +	3770	12	127 - 133
VGG16	3870	19	123 - 135
ResNet50V2	4380	9	135 - 157
MobileNetV2	3810	7	126 - 132
Clasificación multiclase			
Modelo inicial –	4080	266	97 - 106
Modelo inicial +	5760	91	140 - 148
VGG16	6160	20	137 - 169
ResNet50V2	5880	22	134 - 160
MobileNetV2	6120	190	139 - 175

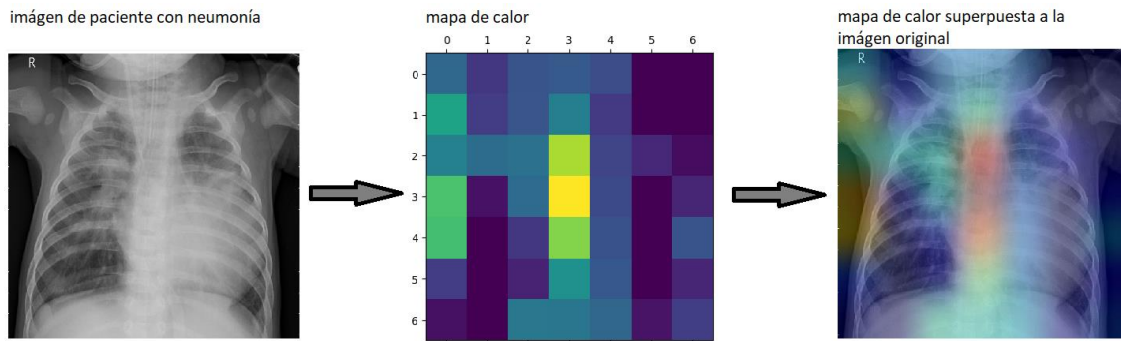
**Tabla 3:** Comparación del tiempo de cálculo entre los modelos.

#### 4.6.2 Visualización a través de la CNN.

Para visualizar las partes de la imagen que han llevado a la CNN a su decisión final, se usó el método Grad-CAM, una manera de asegurarse lo que está mirando realmente la CNN.

El mapa de activación de clase ponderada por el gradiente (Grad-CAM) produce un mapa de calor que destaca las regiones importantes de una imagen utilizando los gradientes del objetivo de la capa convolucional final. Por este motivo es que en las redes preentrenadas se cambió de modo secuencial a funcional. Ya que, en un primer intento al usar la manera secuencial, no fue posible visualizar la última capa de los modelos preentrenados. El anexo E contiene los códigos que se ejecutaron para realizar el Grad-CAM de las imágenes.

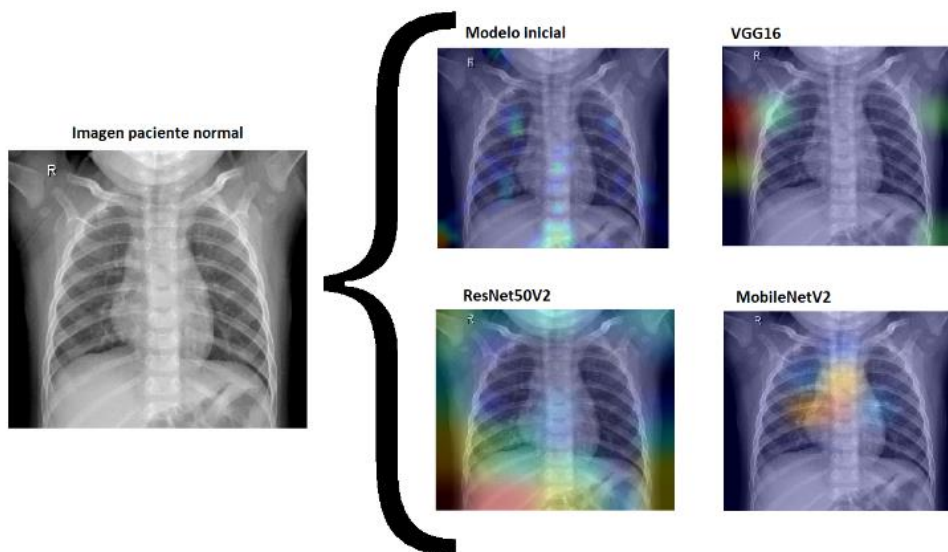
Evaluamos las imágenes con los modelos y obtenemos la predicción del modelo sobre la imagen. Se genera un mapa de calor sobre que partes de la imagen ha tenido en cuenta para su decisión. Finalmente se genera una imagen que superponga la imagen original en el mapa de calor que obtuvimos, como se indica en la figura 33. En la que el modelo VGG16 evaluó la imagen como neumonía y muestra las características que tuvo en cuenta para su decisión, siendo las partes de color más rojizo las partes más destacables de la imagen.



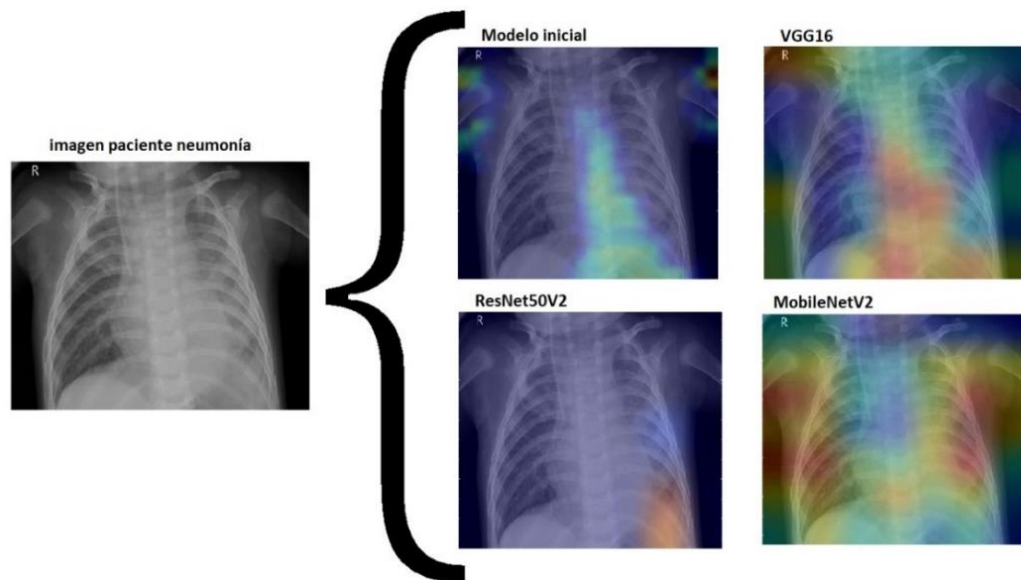
**Figura 33:** Grad-CAM de una imagen de radiografía de paciente con neumonía predecida por el modelo VGG16.

#### 4.6.2.1. Grad-CAM de la clasificación binaria

Las veinte imágenes (diez paciente normal y diez paciente con neumonia) separadas en la carpeta test\_1 se uso para evaluar y observar cada imagen como era clasificada, en la que de estas veinte imágenes, se cogio dos de las imágenes en la que todos los modelos acertaron la clasificación para comparar con el metodo Grad-CAM que genera cada modelo, siendo una de paciente normal (figura 34) y la otra de paciente con neumonia (figura 35). Cada modelo acerto correctamente la categoria a la que pertenecia cada imagen con mas de 90% de probabilidad en su decisión final. En el anexo F se muestra como ejemplo las veinte imágenes que se evaluó con el modelo VGG16, cada imagen con su respectiva predicción. Se puede observar que para cada imagen, los modelos generan diferentes mapas de calor. Siendo en la imagen de paciente con neumonia donde se tienen en cuenta mas características de la imagen. Se podria decir que todos los modelos coinciden en cierta medida en tomar en cuenta como zona prioritaria para la clasificación, la parte inferior del pulmon izquierdo en el paciente con neumonia, mientras que en la imagen del paciente normal, no se observa una zona coincidente entre los modelos.



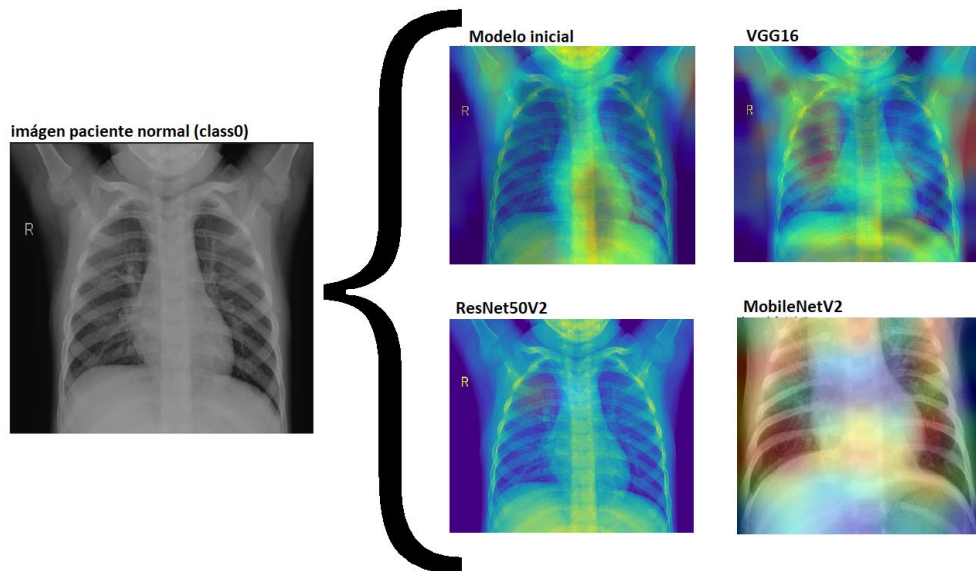
**Figura 34:** Grad-CAM de radiografía de paciente normal con todos los modelos propuestos en clasificación binaria.



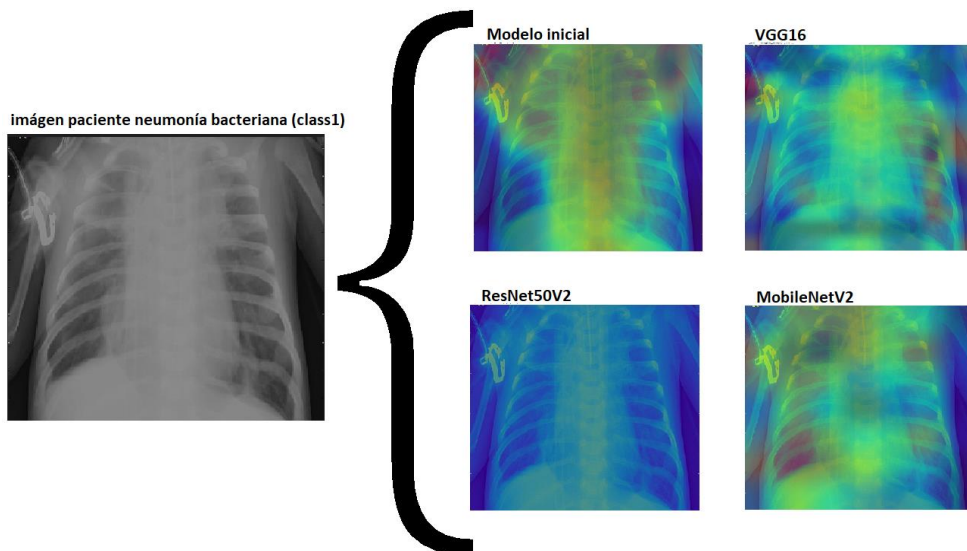
**Figura 35:** Grad-CAM de radiografía de paciente con neumonía con todos los modelos propuestos en clasificación binaria.

#### 4.6.2.2. Grad - CAM de la clasificación multiclase.

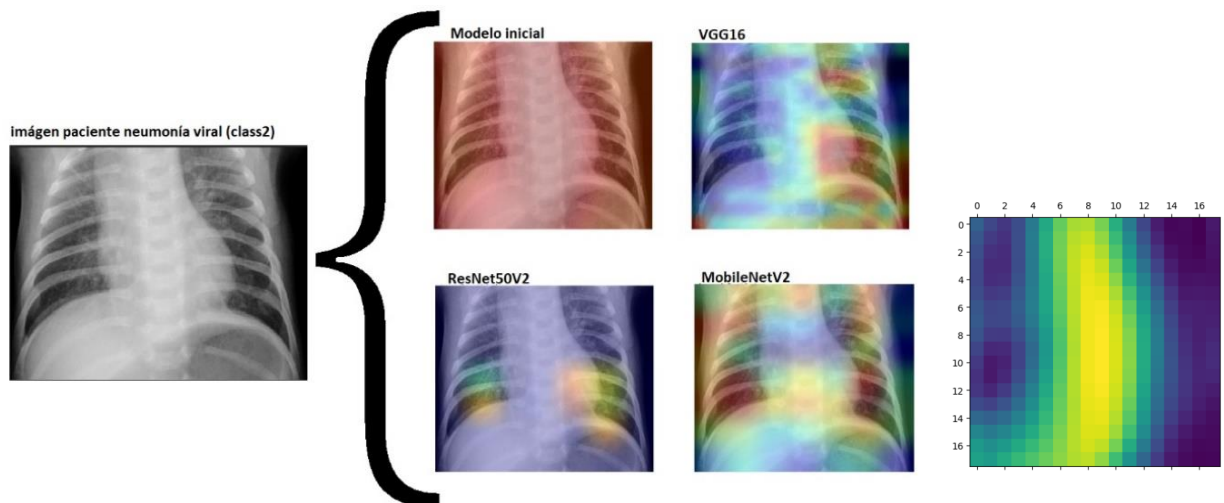
Para el modelo multiclase, también se evaluó las diez imágenes de cada clase de la carpeta test\_1, cada una con la etiqueta que el modelo lo clasificó. En el anexo G se encuentran las imágenes clasificadas por el modelo VGG16 en la clasificación multiclase, así como el código que se usó. En este caso se cogieron tres imágenes, una de cada clase, en las que cada modelo acertó las imágenes con su clasificación correspondiente. La figura 36 de paciente normal (class0), no se observa una relación, cada una toma diferentes características de la imagen como relevantes para su clasificación. En la figura 37 con el paciente con neumonía bacteriana (class1) el modelo ResNet50V2 no llega a generar una zona relevante para generar el mapa de calor a pesar de predecir correctamente la imagen, mientras que los otros modelos sí generan zonas de calor en diferentes zonas de la imagen. Por último, la figura 38 con el paciente con neumonía viral (class2), el modelo inicial genera una imagen que difiere del resto, en la que parece tomar toda la imagen como zona de calor. Pero visualizando solamente el mapa de calor en la figura 39, se observa que toma en cuenta la zona central del tórax pasando por el corazón yendo hacia la parte inferior del pulmón derecho, mientras que el resto de los modelos, si bien tienen diferentes zonas de calor, llegan a coincidir en la zona de la parte inferior del pulmón izquierdo.



**Figura 36:** Grad-CAM de radiografía de paciente normal con todos los modelos propuestos en clasificación multiclase



**Figura 37:** Grad-CAM de radiografía de paciente con neumonía bacteriana con todos los modelos propuestos en clasificación multiclase.



**Figura 38:** Grad-CAM de radiografía de paciente con neumonía viral con todos los modelos propuestos en clasificación multiclase.

**Figura 39:** Mapa de calor del modelo inicial de la figura 37

#### 4.7. Evaluación intercambio de datasets.

Por último, se evaluó los dos tipos de clasificación (binario y multiclase) intercambiando el dataset de validación entre ellos. Es decir, se usó las imágenes de validación de la clasificación binaria en la clasificación multiclase y viceversa.

Para la clasificación binaria, se cogió las imágenes class0 y se puso como etiqueta de normal, mientras que las imágenes de class1 y class2 se juntaron en una sola carpeta con etiqueta de neumonía. Se procedió evaluar mediante la precisión y perdida como se puede observar los resultados en la tabla 4. La CNN de MobileNetV2 es sin duda el que mejor resultados obtuvo con más de un 90% de precisión. Los demás modelos llevan una precisión similar entre poco más del 70%.

En cambio, para la clasificación multiclase, se separó manualmente las imágenes de neumonía (en la carpeta de validación de la clasificación multiclase) en la que cada imagen contaba con etiquetas de si la neumonía correspondía a bacteriana o viral. De manera que las imágenes de pacientes normales pasaron a ser class0, y las imágenes de neumonía se dividió en dos carpetas con las etiquetas de class 1 y class 2. También se evaluó mediante la precisión y perdida como en la tabla 4, en la que al igual que en el anterior caso, MobileNetV2 es el que obtiene unos resultados mayores en comparación a las demás CNN, sin embargo, de forma general, los resultados son muy bajos como para considerar que hace una buena predicción. En el anexo H se muestra los resultados obtenidos en la evaluación de cada modelo de la precisión y perdida.

		N.º imágenes de evaluación	Test_acc (Precisión)	Test_loss (Perdida)
<b>MODELO MULTICLASE</b>	Modelo inicial -	624	44,30%	5,78
	Modelo inicial +	624	23,75%	2,92
	VGG16	624	50,98%	4,08
	ResNet50V2	624	32,74%	16,01
	MobileNetV2	624	56,01%	2,56
<b>MODELO BINARIO</b>	Modelo inicial -	912	71,49%	5,76
	Modelo inicial +	912	72,14%	0,63
	VGG16	912	77,30%	3,28
	ResNet50V2	912	77,19%	1,79
	MobileNetV2	912	92,54%	0,58

**Tabla 4:** Comparación de la evaluación de precisión y perdida intercambiando datasets entre los modelos de clasificación binaria y multiclase.

## 5. Discusión

El uso de las redes neuronales convolucionales se ha convertido especialmente útil en el reconocimiento de imágenes, en la que ha atraído gran atención en el diagnóstico de diversas enfermedades como es el caso de la neumonía, y varios investigadores han utilizado diferentes enfoques para desarrollar sistemas de diagnóstico confiables. En este proyecto se propuso dos modelos sin entrenar, uno sin aumento de datos y otro con aumento de datos (para aumentar las muestras de entrenamiento con el fin de generalizar los modelos y evitar el sobreajuste) adaptados a la clasificación binaria y multiclase. El modelo con aumento de datos presenta un mejor resultado en la precisión de la clasificación binaria, mientras que en la clasificación multiclase la precisión es inferior. Esto se debe que, al aumentar los datos, el modelo requiera más épocas de entrenamiento para llegar a su valor máximo. Como se observa en la figura 24, con cada época va mejorando el valor de la precisión de validación, aunque de forma irregular con picos de bajadas y subidas, presentando un aumento en la última época. Esto debido a que el modelo tiene en cuenta tres clases, mientras que, en la clasificación binaria, no da indicios que vaya a mejorar su valor de precisión por más épocas que se aumente en el entrenamiento. Aun así, no se espera que llegue a obtener una mejora significativa si se aumenta más épocas de entrenamiento.

Una de las limitaciones es que la evaluación de los métodos fue en un conjunto de datos limitados al igual que la mayoría de los investigadores de diferentes artículos que desarrollaron diferentes métodos. El concepto de aprendizaje por transferencia ha sido ampliamente utilizado porque ofrece varias características beneficiosas, como alta precisión, velocidad y facilidad de aplicación. En este estudio, proponemos tres modelos basados en el aprendizaje por transferencia profunda para detectar la neumonía tanto para la clasificación binaria como multiclase. Se evaluó la efectividad de los tres modelos de CCN propuestos que incluyen VGG16, MobileNetV2 y ResNet50V2. En este caso se usó también el aumento de datos con los modelos preentrenados. Las capas que vienen de los modelos preentrenados se congeló y solo se entrenó las capas finales que añadimos para no modificar los pesos que ya tiene en cuenta el modelo preentrenado y no tengamos que entrenarlo desde cero. Los resultados experimentales y la comparación general de los modelos se muestran en la tabla 2. Los resultados demuestran la superioridad del modelo MobileNetV2 en la clasificación multiclase. Se puede ver que el modelo logró el rendimiento más alto con una precisión del 82,12%. Mientras que, en la clasificación binaria, el modelo ResNet50V2 obtiene mejores resultados, sin embargo, no hay una diferencia significativa en comparación a los otros modelos preentrenados.

Por otra parte, los modelos de cada clasificación se evaluaron intercambiando datasets, como se muestran en la tabla 4. Esto para comprobar la eficacia al evaluarlo con otro dataset distinto. Los resultados entre ambas clasificaciones presentan grandes diferencias en cuanto a la precisión. En la clasificación multiclase el modelo MobileNetV2, es quien mejor resultados obtiene con un 56,01%, sin embargo, este resultado es muy bajo como para considerar que hace una buena clasificación. Los demás modelos obtienen resultados

aún más bajos, que podría ser los modelos no lleguen a predecir en si las imágenes. Por lo que se podría catalogar como dependiente de dataset, en la que solo obtiene una precisión elevada con su propio dataset entrenado. A parte también si observamos la matriz de confusión en la figura 31, muestra que la precisión es más imprecisa al momento de clasificar entre neumonía bacteriana o viral, de manera que muchas imágenes de neumonía vírica las acaba catalogando como bacteriana. Esto en si ya es una dificultad de clasificar este tipo de neumonías, ya que muchos tipos de neumonía bacteriana y viral presentan síntomas similares lo que lleva a especialistas a proceder a pruebas complementarias a parte de los rayos x. Por esta razón, algunos estudios proponen la tomografía computarizada como el método más eficaz para detectar y diferenciar la neumonía viral y bacteriana. En cambio, la clasificación binaria generado ofrece al usuario una buena clasificación de las imágenes, distinguiendo entre radiografía de tórax normal y neumonía siendo sin duda el modelo de MobileNetV2 el que mejor resultados obtiene con diferencia con un 92,54% de precisión al evaluarlo con otro dataset como se muestra en la tabla 4.

En las figuras generadas por el Grad - CAM en general no presentan similitud respecto a las partes de la imagen que han llevado a la CNN a su decisión final. Salvo que algunos modelos hayan coincidido en una zona en específico de todas las partes de la imagen que han tenido en cuenta. Las imágenes mostradas en las figuras 33, 34, 35, 36 y 37 usadas para la comparación de la visualización de las CNN han sido etiquetadas correctamente por los modelos, acertando a que tipo de imagen correspondía. En la clasificación binaria, los modelos llegan a generar una zona de calor por la parte inferior del pulmón derecho en la predicción de neumonía. Mientras que en la clasificación multiclase hay más diferencias en las zonas de calor de cada modelo. En la que ResNet50V2 no presenta ninguna zona de calor al predecir la class1 y el modelo inicial con aumento de datos parece tomar toda la imagen como relevante para su predicción de class2, a pesar de que ambos modelos predijeron correctamente las imágenes. Se podría tomar en cuenta que al predecir la imagen de class2, los modelos VGG16, ResNet50V2 y MobileNetV2 llegan a tomar en cuenta la parte inferior del pulmón izquierdo como zona relevante para hacer su predicción.

Hubiera resultado interesante comparar estos mapas de calor con las zonas que tienen en cuenta los profesionales de radiología al determinar si un paciente tiene neumonía o no, y si es posible que puedan determinar si ha sido provocado por virus o bacteria observando la radiografía. De esta manera determinar qué modelo se acerca más a la visualización de un experto radiólogo. Debido a la falta de tiempo y espacio de este proyecto, ya que supondría hacer un estudio más profundo sobre el tema de la neumonía, se deja como un tema a complementar a este trabajo, además que el objetivo del proyecto es el uso de las CNN como método de diagnóstico.

### **5.1. Comparación con otros métodos.**

Se realizó un análisis comparativo para probar la efectividad de los modelos de CNN propuestos para la clasificación de neumonía. Comparamos los resultados de nuestro



sistema propuesto con los métodos de aprendizaje de transferencia y modelos propios propuestos por diferentes investigadores para el diagnóstico automático de neumonía utilizando radiografías de tórax (tabla 5 y tabla 6). Como se muestra en las tablas 5 y 6, hay algunos estudios de alta calidad que han desarrollado modelos de CNN para detectar neumonía. Sin embargo, el principal problema que debe tenerse en cuenta es que la mayoría de ellos utilizan un número limitado de datos. Teniendo todo esto en cuenta, se han desarrollado modelos CNN para la detección de neumonía basados en varias arquitecturas preentrenadas como modelos propios de los autores. Por último, mencionar la dificultad para encontrar diversos estudios en clasificación multiclase entre normal, neumonía bacteriana y viral, ya que la mayoría de los estudios revisados, se hicieron respecto a la detección de neumonía por COVID-19, ya que muchos estudios desarrollaron diferentes modelos en base a usarlos como métodos para combatir la pandemia provocada por el SARS-CoV-2, ya que su detección rápida y precisa era indispensable para detener el avance de esta enfermedad. Por lo que, para una comparación más adecuada, solo se tomó en cuenta estudios que desarrollaran arquitecturas para clasificaciones como las de este proyecto.

<b>Clasificación binaria</b>			
<b>Autor</b>	<b>Modelo</b>	<b>Precisión</b>	<b>Perdida</b>
<b>N. M. Elshennawy [3]</b>	<b>ResNet152V2</b>	<b>99.44%</b>	0.0523
<b>M. Salehi [33]</b>	Xception	93.7	42.8
<b>E. Ayan [34]</b>	Xception	95.03	
<b>D. Hardick [35]</b>	Modelo propio	91,18	
<b>Y. Han [36]</b>	ResNet-18AttRadi	88,6	
<b>Modelos propuestos</b>	Modelo inicial -	81,73%	1,87
	Modelo inicial +	88,14%	0,33
	VGG16	89,42%	0,36
	MobileNetV2	88,46%	0,38
	ResNet50V2	89,74%	0,40

**Tabla 5:** Comparación con obras relacionadas en la clasificación binaria. En negrita el modelo ResNet152V2 con más porcentaje de precisión.

<b>Clasificación multiclase</b>			
<b>Autor</b>	<b>Modelo</b>	<b>Precisión</b>	<b>Perdida</b>
<b>K. Alshamrani, [4]</b>	Modelo propio	78,37%	0,698
<b>P. Naronglerdrit, [17]</b>	<b>DenseNet-201</b>	<b>96.76</b>	
<b>E. Ayan, [34]</b>	Vgg-16	92.94	
<b>Modelos propuestos</b>	Modelo inicial -	80,48%	1,65
	Modelo inicial +	76,97%	0,76
	VGG16	76,20%	0,54
	MobileNetV2	82,12%	0,46
	ResNet50V2	80,15%	0,48

**Tabla 6:** Comparación con obras relacionadas en la clasificación multiclase. En negrita el modelo DenseNet-201 con más porcentaje de precisión.

Otra limitación del proyecto llevado a cabo es que se han utilizado pocas épocas en comparación con los estudios consultados. En nuestro caso, solo se han hecho 30 épocas para la clasificación binaria y 40 para la clasificación multiclase. Esta diferencia de época se debe a la falta de equipos adecuados para hacer un modelo que compile de manera rápida y eficiente, ahorrando tiempo al investigador.

Hubiera sido interesante hacer más modelos y compararlos entre sí, ya que el factor tiempo también ha sido una limitación para este trabajo.

## 6. Conclusión

En este estudio se visualiza la enorme utilidad de las redes neuronales convolucionales en el campo de diagnóstico mediante el reconocimiento de imágenes, centrándonos en el diagnóstico de la neumonía.

En general, se han alcanzado todos los objetivos específicos marcados, en la que se propuso un marco de deep learning para clasificar las radiografías de rayos X de tórax en neumonía y casos normales, y dentro de los casos con neumonía, distinguir neumonía bacteriana de vírica usando cinco modelos diferentes de CNN. Dos de ellos, modelos propuestos en la que uno era sin aumento de datos y otro con aumento de datos, usado para mitigar el problema del sobreajuste obteniendo una mejora en la precisión. Otro método para reducir el sobreajuste es la transferencia de aprendizaje usando redes preentrenadas en los que se basaron los siguientes tres modelos (VGG16, ResNet50V2 y MobileNetV2).

El rendimiento de aprendizaje de los modelos se evaluó en función de la precisión. En la que la clasificación de clase binaria obtuvo buenos resultados en la evaluación con dos tipos de datasets, siendo MobileNetV2 el modelo que presenta mejor resultados. Si bien en la evaluación de su propio dataset, no hay mucha diferencia respecto a los demás modelos, en la evaluación con otro dataset presenta una mayor diferencia de mejora de precisión en comparación a los demás modelos con un 92,54%. Podríamos concluir que el modelo puede considerarse adecuado en la detección de pacientes normales de los que tienen neumonía. Respecto a la clasificación multiclase, la precisión en la primera evaluación con su propio dataset presenta buenos resultados cerca de un 80% de precisión en los modelos, al usar imágenes de otro dataset, la precisión cae considerablemente como para determinar que los modelos hacen una buena predicción. También se evaluó los modelos propuestos comparándolos con investigaciones recientes y similares, en la que diversos estudios muestran resultados significativamente elevados, muchos de ellos por encima del 90%.


Visualizando las partes relevantes de la imagen que toma en cuenta la CNN mediante el Grad - CAM, aunque los modelos generen mapas de calor diferentes, se determina que en general toman en cuenta el interior de la parte torácica para determinar si hay neumonía.

Con la finalización del proyecto, y con la gran cantidad de bibliografía existente actualmente en relación con el tema elegido, los resultados de la evaluación indicaron el potencial de detectar neumonía cobrando mayor importancia para ayudar a nuestros profesionales en el diagnóstico. Si bien los resultados en la clasificación multiclase en este trabajo no resultaron significativos, se pretende poder mejorar los modelos en un futuro, utilizando otras técnicas de aprendizaje profundo, así como una mayor cantidad de imágenes de rayos x y usar diferentes parámetros en los modelos.

## 7. Bibliografía

- [1] A. Torres *et al.*, “Pneumonia,” *Nat. Rev. Dis. Prim.* 2021 71, vol. 7, no. 1, pp. 1–28, Apr. 2021, doi: 10.1038/s41572-021-00259-0.
- [2] I. I. Daood, R. H. Ibrahim, A. A. Hussein, N. Norsamsi, and N. Mazlan, “Knowledge of pneumonia among nursing staff in Mosul Hospitals, Iraq,” *Int. J. Med. Toxicol. Leg. Med.*, vol. 23, no. 3–4, pp. 107–116, 2020, doi: 10.5958/0974-4614.2020.00054.6.
- [3] N. M. Elshennawy and D. M. Ibrahim, “Deep-Pneumonia Framework Using Deep Learning Models Based on Chest X-Ray Images,” *Diagnostics*, vol. 10, no. 9, Sep. 2020, doi: 10.3390/DIAGNOSTICS10090649.
- [4] K. Alshamrani, H. A. Alshamrani, A. A. Asiri, F. F. Alqahtani, W. T. Mohammad, and A. H. Alshehri, “The Use of Chest Radiographs and Machine Learning Model for the Rapid Detection of Pneumonitis in Pediatric,” *Biomed Res. Int.*, vol. 2022, 2022, doi: 10.1155/2022/5260231.
- [5] R. S. Fraser, N. Colman, N. L. Müller, and P. D. Paré, “Enfermedades infecciosas de los pulmones,” *Fundam. las enfermedades del tórax*, p. 222, 2006, doi: 10.1016/B978-84-458-1603-5.50006-X.
- [6] “Las redes neuronales | Qué son y cómo se comportan | Mobile World Capital Barcelona,” *MOBILE WORLD CAPITAL, BARCELONA*, 2022. <https://mobileworldcapital.com/redes-neuronales-funcionamiento/> (accessed May 22, 2023).
- [7] “‘Machine Learning’: definición, tipos y aplicaciones prácticas - Iberdrola.” <https://www.iberdrola.com/innovacion/machine-learning-aprendizaje-automatico> (accessed May 22, 2023).
- [8] F. Chollet, “DEEP LEARNING with Python”, M A N N I N G, 2018.
- [9] “Qué son las redes neuronales y sus funciones | ATRIA Innovation,” 2019. <https://www.atriainnovation.com/que-son-las-redes-neuronales-y-sus-funciones/> (accessed May 22, 2023).
- [10] K. Ujjwal, “TECHNOLOGIES RUNNING - TECNOLOGÍAS EN ACCIÓN: Introducción rápida a las redes neuronales,” 2016. <https://technologiesrunning.blogspot.com/2016/11/introduccion-rapida-las-redes-neuronales.html> (accessed May 22, 2023).
- [11] P. Huet, “Qué son las redes neuronales y sus aplicaciones | OpenWebinars,” 2023. <https://openwebinars.net/blog/que-son-las-redes-neuronales-y-sus-aplicaciones/> (accessed May 22, 2023).
- [12] D. Charte, “Reconocimiento y clasificación de imágenes con Deep Learning | campusMVP.es.” <https://www.campusmvp.es/recursos/post/reconocimiento-y-clasificacion-de-imagenes-con-deep-learning.aspx> (accessed May 22, 2023).
- [13] A. Esteva *et al.*, “Dermatologist-level classification of skin cancer with deep neural networks,” *Nature*, vol. 542, no. 7639, p. 115, Feb. 2017, doi: 10.1038/NATURE21056.
- [14] G. E. Dahl, N. Jaitly, and R. Salakhutdinov, “Multi-task Neural Networks for QSAR Predictions,” Jun. 2014, Accessed: May 22, 2023. [Online]. Available: <http://arxiv.org/abs/1406.1231>.
- [15] R. Yasashvini, V. Raja Sarobin M, R. Panjanathan, S. Graceline Jasmine, and L. Jani

- Anbarasi, "Diabetic Retinopathy Classification Using CNN and Hybrid Deep Convolutional Neural Networks," *Symmetry* 2022, Vol. 14, Page 1932, vol. 14, no. 9, p. 1932, Sep. 2022, doi: 10.3390/SYM14091932.
- [16] "Detección de COVID-19 en radiografías con Deep Learning - IIC." <https://www.iic.uam.es/lasalud/deteccion-covid19-en-radiografias-con-deep-learning/> (accessed May 22, 2023).
- [17] P. Naronglerdrit, I. Mporas, and A. Sheikh-Akbari, "COVID-19 detection from chest X-rays using transfer learning with deep convolutional neural networks," *Data Sci. COVID-19*, p. 255, Jan. 2021, doi: 10.1016/B978-0-12-824536-1.00031-9.
- [18] P. P. Torralba, "Qué son las Redes Neuronales Convolucionales," *Think. Innov.*, Sep. 2022, Accessed: May 22, 2023. [Online]. Available: <https://www.iebschool.com/blog/redes-neuronales-convolucionales-big-data/>.
- [19] K. Redacción, "¿Qué son las Redes Neuronales Convolucionales? | KeepCoding Bootcamps," 2023. <https://keepcoding.io/blog/redes-neuronales-convolucionales/> (accessed May 22, 2023).
- [20] Á. Artola Moreno, "TFG-2402-ARTOLA," 2019.
- [21] O. A. M. López, A. M. López, and D. J. Crossa, "Fundamentals of Artificial Neural Networks and Deep Learning," *Multivar. Stat. Mach. Learn. Methods Genomic Predict.*, pp. 379–425, Jan. 2022, doi: 10.1007/978-3-030-89010-0\_10.
- [22] T. SuperDataScience, "Convolutional Neural Networks (CNN): Step 3 - Flattening - Blogs - SuperDataScience | Machine Learning | AI | Data Science Career | Analytics | Success," 2018. <https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening> (accessed May 22, 2023).
- [23] "Qué es overfitting y underfitting y cómo solucionarlo | Aprende Machine Learning," 2017. <https://www.aprendemachinelearning.com/que-es-overfitting-y-underfitting-y-como-solucionarlo/> (accessed May 22, 2023).
- [24] S. Asif, Y. Wenhui, K. Amjad, H. Jin, Y. Tao, and S. Jinhai, "Detection of COVID-19 from chest X-ray images: Boosting the performance with convolutional neural network and transfer learning," *Expert Syst.*, vol. 40, no. 1, Jan. 2023, doi: 10.1111/EXSY.13099.
- [25] "Red neuronal convolucional MobileNet-v2 - MATLAB mobilenetv2 - MathWorks España." <https://es.mathworks.com/help/deeplearning/ref/mobilenetv2.html> (accessed May 22, 2023).
- [26] M. Rahimzadeh and A. Attar, "A modified deep convolutional neural network for detecting COVID-19 and pneumonia from chest X-ray images based on the concatenation of Xception and ResNet50V2," *Informatics Med. Unlocked*, vol. 19, p. 100360, Jan. 2020, doi: 10.1016/J.IMU.2020.100360.
- [27] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, 2015.
- [28] H. Panwar, P. K. Gupta, M. K. Siddiqui, R. Morales-Menendez, P. Bhardwaj, and V. Singh, "A deep learning and grad-CAM based color visualization approach for fast detection of COVID-19 cases using chest X-ray and CT-Scan images," *Chaos. Solitons. Fractals*, vol. 140, p. 110190, Nov. 2020, doi: 10.1016/J.CHAOS.2020.110190.
- [29] F. Chollet, "Grad-CAM class activation visualization," 2020. [https://keras.io/examples/vision/grad\\_cam/](https://keras.io/examples/vision/grad_cam/) (accessed May 22, 2023).

- [30] “Pastor Alemán Perro Del Lobo - Foto gratis en Pixabay - Pixabay.” <https://pixabay.com/es/photos/pastor-alem%C3%A1n-perro-del-lobo-mirada-5362375/> (accessed Jun. 04, 2023).
- [31] “Pediatric Pneumonia Chest X-ray | Kaggle.” <https://www.kaggle.com/datasets/andrewmvd/pediatric-pneumonia-chest-xray> (accessed May 22, 2023).
- [32] “Chest Xrays: bacterial / viral pneumonia / normal | Kaggle.” <https://www.kaggle.com/datasets/kostasdiamantaras/chest-xrays-bacterial-viral-pneumonia-normal> (accessed May 22, 2023).
- [33] M. Salehi, R. Mohammadi, H. Ghaffari, N. Sadighi, and R. Reiazi, “Automated detection of pneumonia cases using deep transfer learning with paediatric chest X-ray images,” *Br. J. Radiol.*, vol. 94, no. 1121, May 2021, doi: 10.1259/BJR.20201263.
- [34] E. Ayan, B. Karabulut, and H. M. Ünver, “Diagnosis of Pediatric Pneumonia with Ensemble of Deep Convolutional Neural Networks in Chest X-Ray Images,” *Arab. J. Sci. Eng.*, vol. 47, no. 2, p. 2123, Feb. 2022, doi: 10.1007/S13369-021-06127-Z.
- [35] D. Hardick, “Medical X-ray  Image Classification using Convolutional Neural Network | by Hardik Deshmukh | Towards Data Science,” 2020. <https://towardsdatascience.com/medical-x-ray--image-classification-using-convolutional-neural-network-9a6d33b1c2a> (accessed May 22, 2023).
- [36] Y. Han, C. Chen, A. Tewfik, Y. Ding, and Y. Peng, “PNEUMONIA DETECTION ON CHEST X-RAY USING RADIOMIC FEATURES AND CONTRASTIVE LEARNING,” *Proceedings. IEEE Int. Symp. Biomed. Imaging*, vol. 2021, p. 247, Apr. 2021, doi: 10.1109/ISBI48211.2021.9433853.

## Anexo A

Capas que contiene el modelo preentrenado VGG16 junto con las capas de salida añadidas al final. El total de pesos entrenables son treinta, por lo que se congela los pesos propios del modelo VGG16, de manera que queda cuatro pesos entrenables que son los que añadimos al final.

```
[ ] model.summary()
```

```
Model: "model"
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 256)	6422784
dense_1 (Dense)	(None, 1)	257

```
=====  
Total params: 21,137,729
```

```
Trainable params: 21,137,729
```

```
Non-trainable params: 0
```

```
[ ] print('This is the number of trainable weights '  
        'before freezing the conv base:', len(model.trainable_weights))
```

```
This is the number of trainable weights before freezing the conv base: 30
```

```
[ ] conv_base.trainable = False  
print('This is the number of trainable weights '  
      'after freezing the conv base:', len(model.trainable_weights))
```

```
This is the number of trainable weights after freezing the conv base: 4
```

## Anexo B

Entrenamiento de la clasificación binaria con VGG16 durante 30 épocas. Loss y accuracy corresponden a las imágenes de entrenamiento y val\_loss y val\_accuracy corresponden a las imágenes de validación. Se muestra los valores de precisión y perdida de cada época, así como el tiempo de cálculo.

```
history = model.fit(
    train_generator,
    epochs=30,
    validation_data=validation_generator)

Epoch 1/30
163/163 [=====] - 135s 802ms/step - loss: 0.5238 - accuracy: 0.8358 - val_loss: 0.4502 - val_accuracy: 0.8285
Epoch 2/30
163/163 [=====] - 126s 772ms/step - loss: 0.2540 - accuracy: 0.8987 - val_loss: 0.2596 - val_accuracy: 0.8990
Epoch 3/30
163/163 [=====] - 126s 776ms/step - loss: 0.2282 - accuracy: 0.9121 - val_loss: 0.3881 - val_accuracy: 0.8558
Epoch 4/30
163/163 [=====] - 128s 788ms/step - loss: 0.2117 - accuracy: 0.9163 - val_loss: 0.6481 - val_accuracy: 0.7885
Epoch 5/30
163/163 [=====] - 126s 771ms/step - loss: 0.1898 - accuracy: 0.9252 - val_loss: 0.4445 - val_accuracy: 0.8381
Epoch 6/30
163/163 [=====] - 126s 770ms/step - loss: 0.1778 - accuracy: 0.9300 - val_loss: 0.2687 - val_accuracy: 0.8830
Epoch 7/30
163/163 [=====] - 125s 765ms/step - loss: 0.1732 - accuracy: 0.9298 - val_loss: 0.2817 - val_accuracy: 0.9071
Epoch 8/30
163/163 [=====] - 126s 773ms/step - loss: 0.1660 - accuracy: 0.9344 - val_loss: 0.2544 - val_accuracy: 0.9135
Epoch 9/30
163/163 [=====] - 125s 768ms/step - loss: 0.1531 - accuracy: 0.9394 - val_loss: 0.2449 - val_accuracy: 0.9167
Epoch 10/30
163/163 [=====] - 125s 770ms/step - loss: 0.1538 - accuracy: 0.9386 - val_loss: 0.2353 - val_accuracy: 0.9199
Epoch 11/30
163/163 [=====] - 129s 792ms/step - loss: 0.1432 - accuracy: 0.9407 - val_loss: 0.4371 - val_accuracy: 0.8766
Epoch 12/30
163/163 [=====] - 127s 779ms/step - loss: 0.1482 - accuracy: 0.9411 - val_loss: 0.2847 - val_accuracy: 0.9022
Epoch 13/30
163/163 [=====] - 130s 799ms/step - loss: 0.1361 - accuracy: 0.9515 - val_loss: 0.2496 - val_accuracy: 0.9167
Epoch 14/30
163/163 [=====] - 128s 786ms/step - loss: 0.1387 - accuracy: 0.9472 - val_loss: 0.2457 - val_accuracy: 0.9054
Epoch 15/30
163/163 [=====] - 128s 785ms/step - loss: 0.1319 - accuracy: 0.9509 - val_loss: 0.2461 - val_accuracy: 0.9119
Epoch 16/30
163/163 [=====] - 128s 786ms/step - loss: 0.1434 - accuracy: 0.9486 - val_loss: 0.2511 - val_accuracy: 0.9087
Epoch 17/30
163/163 [=====] - 124s 761ms/step - loss: 0.1319 - accuracy: 0.9469 - val_loss: 0.2546 - val_accuracy: 0.9087
Epoch 18/30
163/163 [=====] - 125s 765ms/step - loss: 0.1297 - accuracy: 0.9513 - val_loss: 0.5476 - val_accuracy: 0.8446
Epoch 19/30
163/163 [=====] - 125s 766ms/step - loss: 0.1276 - accuracy: 0.9524 - val_loss: 0.3215 - val_accuracy: 0.9054
Epoch 20/30
163/163 [=====] - 124s 763ms/step - loss: 0.1196 - accuracy: 0.9545 - val_loss: 0.3270 - val_accuracy: 0.9022
Epoch 21/30
163/163 [=====] - 125s 765ms/step - loss: 0.1273 - accuracy: 0.9493 - val_loss: 0.2546 - val_accuracy: 0.9071
Epoch 22/30
163/163 [=====] - 129s 792ms/step - loss: 0.1204 - accuracy: 0.9561 - val_loss: 0.2666 - val_accuracy: 0.9087
Epoch 23/30
163/163 [=====] - 125s 764ms/step - loss: 0.1225 - accuracy: 0.9534 - val_loss: 0.3319 - val_accuracy: 0.9006
Epoch 24/30
163/163 [=====] - 124s 762ms/step - loss: 0.1217 - accuracy: 0.9534 - val_loss: 0.2821 - val_accuracy: 0.8926
Epoch 25/30
163/163 [=====] - 128s 787ms/step - loss: 0.1116 - accuracy: 0.9587 - val_loss: 0.3463 - val_accuracy: 0.9071
Epoch 26/30
163/163 [=====] - 125s 767ms/step - loss: 0.1151 - accuracy: 0.9574 - val_loss: 0.2926 - val_accuracy: 0.9071
Epoch 27/30
163/163 [=====] - 123s 753ms/step - loss: 0.1089 - accuracy: 0.9611 - val_loss: 0.4002 - val_accuracy: 0.8958
Epoch 28/30
163/163 [=====] - 123s 754ms/step - loss: 0.1166 - accuracy: 0.9566 - val_loss: 0.3529 - val_accuracy: 0.8974
Epoch 29/30
163/163 [=====] - 126s 774ms/step - loss: 0.1103 - accuracy: 0.9582 - val_loss: 0.3690 - val_accuracy: 0.8926

[ ] acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
```



## Anexo C

Resultados de la evaluación de la precisión y pérdida de cada modelo de los dos tipos de clasificación con sus respectivos datasets de imágenes de validación, así como el tiempo de cálculo.

### Clasificación binaria

#### Modelo inicial -

```
Found 624 images belonging to 2 classes.  
20/20 [=====] - 6s 315ms/step - loss: 1.8787 - accuracy: 0.8173  
test loss 1.8787246942520142  
test acc: 0.817307710647583
```

#### Modelo inicial +

```
Found 624 images belonging to 2 classes.  
20/20 [=====] - 12s 585ms/step - loss: 0.3355 - accuracy: 0.8814  
test loss 0.3355235457420349  
test acc: 0.8814102411270142
```

#### VGG16

```
Found 624 images belonging to 2 classes.  
20/20 [=====] - 19s 710ms/step - loss: 0.3676 - accuracy: 0.8942  
test loss 0.3676004707813263  
test acc: 0.8942307829856873
```

#### MobileNetV2

```
Found 624 images belonging to 2 classes.  
20/20 [=====] - 7s 357ms/step - loss: 0.3807 - accuracy: 0.8846  
test loss 0.3807266056537628  
test acc: 0.8846153616905212
```

#### ResNet50V2

```
Found 624 images belonging to 2 classes.  
20/20 [=====] - 9s 438ms/step - loss: 0.4019 - accuracy: 0.8974  
test loss 0.401906818151474  
test acc: 0.8974359035491943
```

### Clasificación multiclase

#### Modelo inicial -

```
Found 912 images belonging to 3 classes.  
29/29 [=====] - 266s 9s/step - loss: 1.6530 - accuracy: 0.8048  
test loss 1.653046727180481  
test acc: 0.8048245906829834
```

#### Modelo inicial +

```
Found 912 images belonging to 3 classes.  
29/29 [=====] - 91s 3s/step - loss: 0.6046 - accuracy: 0.7697  
test acc: 0.7697368264198303
```

#### VGG16

```
Found 912 images belonging to 3 classes.  
29/29 [=====] - 20s 667ms/step - loss: 0.5469 - accuracy: 0.7621  
test loss 0.5469269156455994  
test acc: 0.7620614171028137
```

#### MobileNetV2

```
Found 912 images belonging to 3 classes.  
29/29 [=====] - 190s 6s/step - loss: 0.4646 - accuracy: 0.8213  
test loss 0.4645577073097229  
test acc: 0.8212719559669495
```

#### ResNet50V2

```
Found 912 images belonging to 3 classes.  
29/29 [=====] - 22s 744ms/step - loss: 0.4803 - accuracy: 0.8015  
test loss 0.48032695055007935  
test acc: 0.8015350699424744
```

## Anexo D

Códigos con los que se realizó la matriz de confusión para la clasificación multiclase.

```
[ ] class0 = '/content/drive/MyDrive/chest_x_ray/validation_images/class0'  
class1 = '/content/drive/MyDrive/chest_x_ray/validation_images/class1'  
class2 = '/content/drive/MyDrive/chest_x_ray/validation_images/class2'
```

```
[ ] import cv2  
from imutils import paths  
imagePaths = list(paths.list_images(class0))  
data = []  
labels = []  
for imagePath in imagePaths:  
    label = 0  
    # 224x224 pixels while ignoring aspect ratio  
    image = cv2.imread(imagePath)  
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  
    image = cv2.resize(image, (224, 224))  
    # update the data and labels lists, respectively  
    data.append(image)  
    labels.append(label)  
data = np.array(data) / 255  
labels = np.array(labels)  
  
imagePaths1 = list(paths.list_images(class1))  
data1 = []  
labels1 = []  
for imagePath in imagePaths1:  
    label1 = 1  
    # 224x224 pixels while ignoring aspect ratio  
    image = cv2.imread(imagePath)  
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  
    image = cv2.resize(image, (224, 224))  
    data1.append(image)  
    labels1.append(label1)  
  
data1 = np.array(data1) / 255  
labels1 = np.array(labels1)  
  
imagePaths2 = list(paths.list_images(class2))  
data2 = []  
labels2 = []  
for imagePath in imagePaths2:  
    label2 = 2  
    image = cv2.imread(imagePath)  
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  
    image = cv2.resize(image, (224, 224))  
    data2.append(image)  
    labels2.append(label2)  
data2 = np.array(data2) / 255  
labels2 = np.array(labels2)  
  
dataset = np.concatenate((data, data1, data2), axis=0)  
label = np.concatenate((labels, labels1, labels2), axis=0)  
  
label2 = to_categorical(label)  
  
label1 = to_categorical(label)
```

```
▶ from sklearn.metrics import classification_report  
BS = 8  
print("[INFO] evaluating network...")  
predIdxs = model.predict(dataset, batch_size=BS)  
predIdxs = np.argmax(predIdxs, axis=1)  
print(classification_report(label1.argmax(axis=1), predIdxs, digits=4))
```

```
▶ from sklearn.metrics import confusion_matrix  
import seaborn as sns  
cm = confusion_matrix(label1.argmax(axis=1), predIdxs)  
sns.set(font_scale=1)#for label size  
sns.heatmap(cm, cmap="Blues", annot=True, annot_kws={"size": 12})# font size  
plt.ylabel('Actual')  
plt.xlabel('Predicted')  
plt.figure(figsize = (16,12))  
total = sum(sum(cm))  
acc = (cm[0, 0] + cm[1, 1]) / total  
sensitivity = cm[0, 0] / (cm[0, 0] + cm[0, 1])  
specificity = cm[1, 1] / (cm[1, 0] + cm[1, 1])  
# show the confusion matrix, accuracy, sensitivity, and specificity  
print("acc: {:.4f}".format(acc))  
print("sensitivity: {:.4f}".format(sensitivity))  
print("specificity: {:.4f}".format(specificity))
```

## Anexo E

Código con los que se visualizó el Grad-CAM de las imágenes.

Definimos la función Grad-CAM que generará el mapa de calor, así como la variable que tendrá la imagen y la que tiene el nombre de la última capa del modelo, siendo la capa de salida.

```
[ ] def make_gradcam_heatmap(img_array, model, last_conv_layer_name, pred_index=None):
    # First, we create a model that maps the input image to the activations
    # of the last conv layer as well as the output predictions
    grad_model = tf.keras.models.Model(
        [model.inputs], [model.get_layer(last_conv_layer_name).output, model.output]
    )

    # Then, we compute the gradient of the top predicted class for our input image
    # with respect to the activations of the last conv layer
    with tf.GradientTape() as tape:
        last_conv_layer_output, preds = grad_model(img_array)
        if pred_index is None:
            pred_index = tf.argmax(preds[0])
        class_channel = preds[:, pred_index]

    # This is the gradient of the output neuron (top predicted or chosen)
    # with regard to the output feature map of the last conv layer
    grads = tape.gradient(class_channel, last_conv_layer_output)

    # This is a vector where each entry is the mean intensity of the gradient
    # over a specific feature map channel
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

    # We multiply each channel in the feature map array
    # by "how important this channel is" with regard to the top predicted class
    # then sum all the channels to obtain the heatmap class activation
    last_conv_layer_output = last_conv_layer_output[0]
    heatmap = last_conv_layer_output @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap)

    # For visualization purpose, we will also normalize the heatmap between 0 & 1
    heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
    return heatmap.numpy()
```

Preprocesamos la imagen de manera que los valores de entrada sean iguales a los que se hizo con las imágenes de entrada en el entrenamiento del modelo. A continuación, con el siguiente código visualizamos el mapa de calor.

```
[ ] #img_array = preprocess_input(get_img_array(img_path, size=img_size))
    image = cv2.imread(img_path)
    image = cv2.resize(image, (224, 224))
    image = image.astype('float32') / 255
    image = np.expand_dims(image, axis=0)
    # Remove last layer's softmax
    model.layers[-1].activation = None

    # Generate class activation heatmap
    heatmap = make_gradcam_heatmap(image, model, last_conv_layer_name)

    # Display heatmap
    plt.matshow(heatmap)
    plt.show()
```

Creamos una función que superponga el mapa de calor en la imagen de entrada y lo visualizamos.

```
[ ] def save_and_display_gradcam(img_path, heatmap, cam_path="cam.jpg", alpha=0.4):
    # Load the original image
    img = keras.preprocessing.image.load_img(img_path)
    img = keras.preprocessing.image.img_to_array(img)

    # Rescale heatmap to a range 0-255
    heatmap = np.uint8(255 * heatmap)

    # Use jet colormap to colorize heatmap
    jet = cm.get_cmap("jet")

    # Use RGB values of the colormap
    jet_colors = jet(np.arange(256))[:, :3]
    jet_heatmap = jet_colors[heatmap]

    # Create an image with RGB colorized heatmap
    jet_heatmap = keras.preprocessing.image.array_to_img(jet_heatmap)
    jet_heatmap = jet_heatmap.resize((img.shape[1], img.shape[0]))
    jet_heatmap = keras.preprocessing.image.img_to_array(jet_heatmap)

    # Superimpose the heatmap on original image
    superimposed_img = jet_heatmap * alpha + img
    superimposed_img = keras.preprocessing.image.array_to_img(superimposed_img)

    # Save the superimposed image
    superimposed_img.save(cam_path)

    # Display Grad CAM
    display(Image(cam_path))

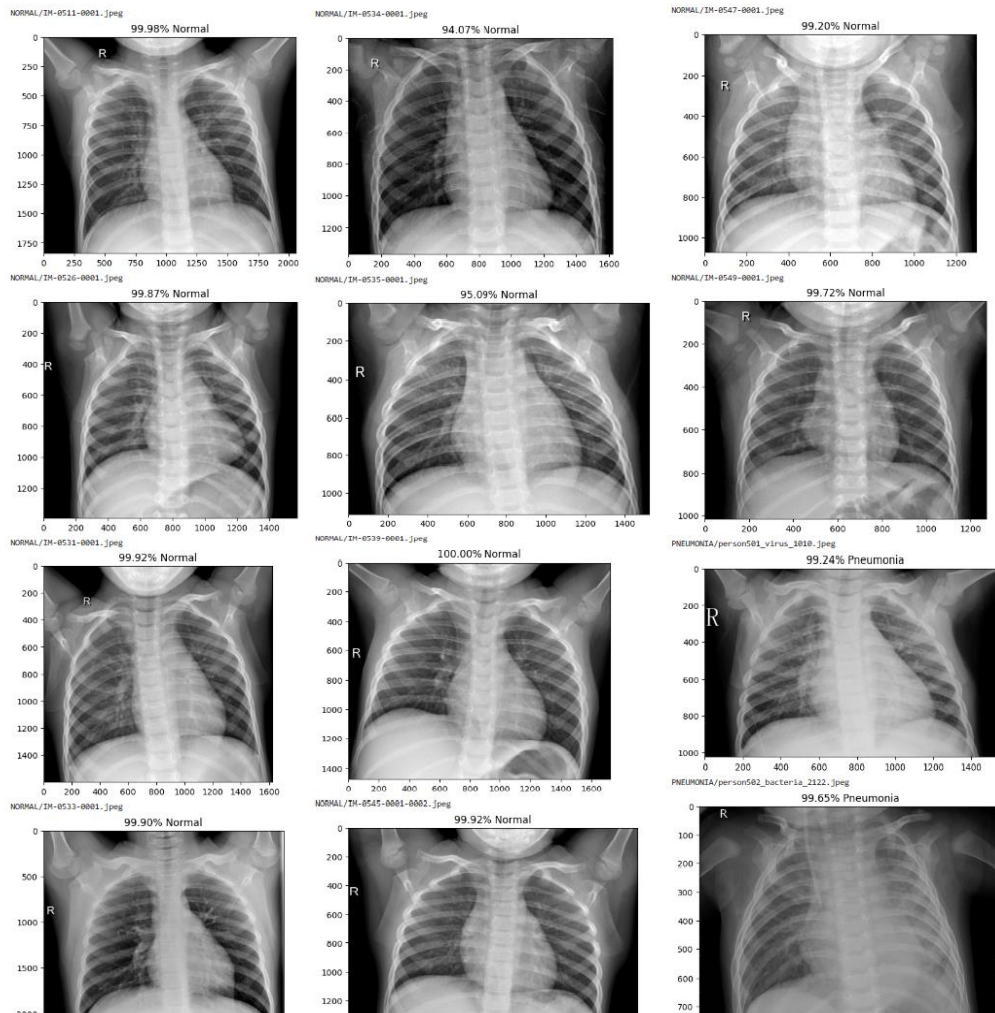
save_and_display_gradcam(img_path, heatmap)
```

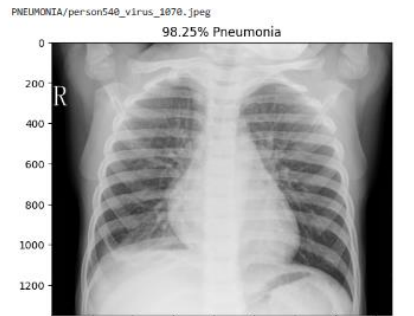
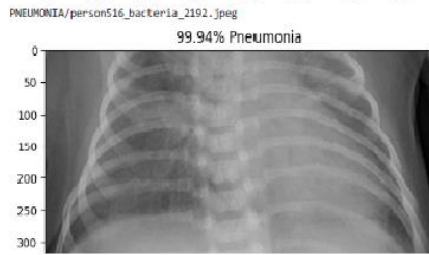
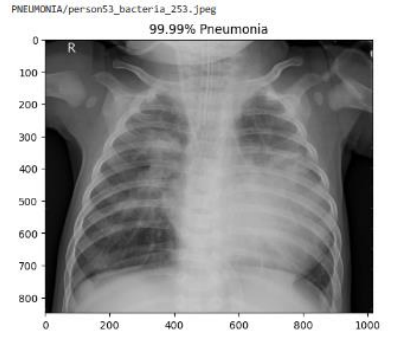
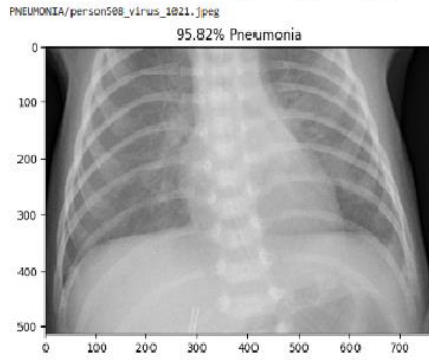
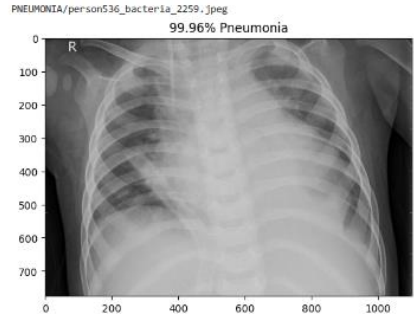
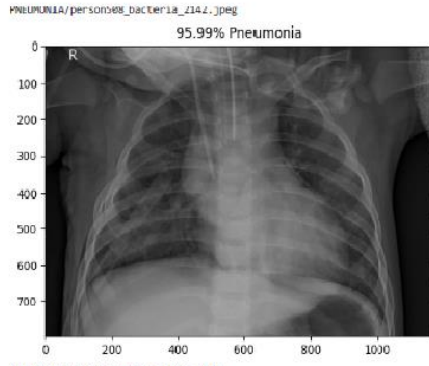
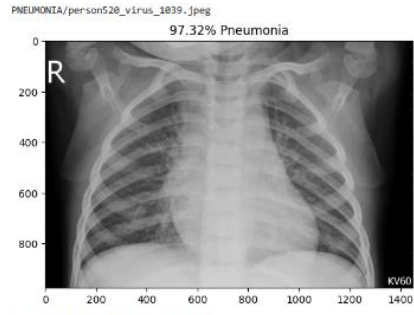
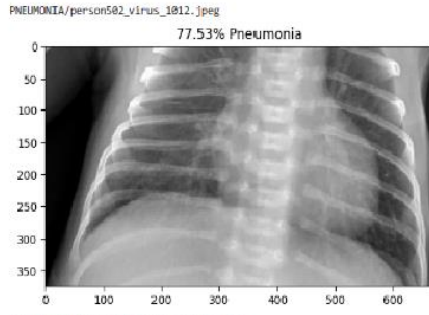
## Anexo F

Código con el que se visualiza la predicción de las veinte imágenes en la carpeta test\_1 (diez de paciente normal y diez de paciente con neumonía) con el modelo VGG16 en la clasificación binaria. Cada imagen con su predicción.

```
[ ] test_generator.reset()
pred = model.predict(test_generator,1000,verbose=1)
print("Predictions finished")
import cv2
import matplotlib.image as mpimg
for index, probability in enumerate(pred):
    image_path = test_dir_1 + "/" + test_generator.filesnames[index]
    image = mpimg.imread(image_path)
    #BGR TO RGB conversion using CV2
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    pixels = np.array(image)
    plt.imshow(pixels)

    print(test_generator.filesnames[index])
    if probability > 0.5:
        plt.title("%.2f" % (probability[0]*100) + "% Pneumonia")
    else:
        plt.title("%.2f" % ((1-probability[0])*100) + "% Normal")
    plt.show()
```





## Anexo G

Código con las que se visualiza las imágenes de la carpeta test\_1 de las tres clases (diez imágenes de cada clase) en la clasificación multiclase con el modelo VGG16. Se muestra 20 de las 24 imágenes que el modelo predijo correctamente.



## Anexo H

Resultados de la evaluación de la precisión y pérdida de cada modelo de los dos tipos de clasificación intercambiando datasets de imágenes de validación, así como el tiempo de cálculo.

### Clasificación binaria

#### Modelo inicial -

Found 912 images belonging to 2 classes.  
29/29 [=====] - 127s 4s/step - loss: 5.7687 - accuracy: 0.7149  
test loss 5.768734931945801  
test acc: 0.7149122953414917

#### Modelo inicial +

Found 912 images belonging to 2 classes.  
29/29 [=====] - 171s 6s/step - loss: 0.6390 - accuracy: 0.7215  
test loss 0.638984203338623  
test acc: 0.7214912176132202

#### VGG16

Found 912 images belonging to 2 classes.  
29/29 [=====] - 230s 8s/step - loss: 2.5217 - accuracy: 0.7160  
test loss 2.521702766418457  
test acc: 0.7160087823867798

#### MobileNetV2

Found 912 images belonging to 2 classes.  
29/29 [=====] - 544s 19s/step - loss: 0.5825 - accuracy: 0.9254  
test loss 0.582502543926239  
test acc: 0.9254385828971863

#### ResNet50V2

Found 912 images belonging to 2 classes.  
29/29 [=====] - 186s 6s/step - loss: 1.7925 - accuracy: 0.7719  
test loss 1.7924884557724  
test acc: 0.7719298005104065

### Clasificación multiclase

#### Modelo inicial -

Found 623 images belonging to 3 classes.  
20/20 [=====] - 97s 5s/step - loss: 5.7850 - accuracy: 0.4430  
test loss 5.7849578857421875  
test acc: 0.4430176615715027

#### Modelo inicial +

Found 623 images belonging to 3 classes.  
20/20 [=====] - 174s 9s/step - loss: 2.9287 - accuracy: 0.2376  
test loss 2.9287302494049072  
test acc: 0.2375601977109909

#### VGG16

Found 612 images belonging to 3 classes.  
20/20 [=====] - 12s 591ms/step - loss: 4.0851 - accuracy: 0.5098  
test loss 4.085146427154541  
test acc: 0.5098039507865906

#### MobileNetV2

Found 623 images belonging to 3 classes.  
20/20 [=====] - 225s 12s/step - loss: 2.5690 - accuracy: 0.5602  
test loss 2.5689566135406494  
test acc: 0.5601926445960999

#### ResNet50V2

Found 623 images belonging to 3 classes.  
20/20 [=====] - 123s 6s/step - loss: 16.0134 - accuracy: 0.3274  
test loss 16.01339340209961  
test acc: 0.3274478316307068