

Diagnósticos médicos a través de Deep Learning

Indice

1. Introducción.....	5
1.1 Contexto y Justificación del Trabajo.....	5
1.2 Objetivos del Trabajo.....	7
1.2.1 Objetivos generales.....	7
1.2.2 Objetivos específicos.....	7
1.3 Enfoque y método seguido.....	7
1.3.1 Enfoque.....	7
1.3.2 Enumeración de las etapas del trabajo.....	8
1.4 - Estudios consultados.....	8
2.- Marco teórico.....	9
2.1.- Deep Learning.....	9
2.2 Convolutional Neural Network.....	10
3.- Desarrollo del trabajo.....	11
3.1.- Descripción del dataset.....	12
3.2.- Carga de datos.....	12
3.3.- Breve descripción de los datos.....	12
3.4.- Reducción del dataset y preprocesamiento.....	13
3.5 Convolution Neural Network.....	14
3.6.- Entrenamiento y validación.....	16
3.7.- Posibles mejoras del modelo.....	16
4.- Resultados.....	21
4.1.- Modelo general.....	21
4.2.- Data Augmentation.....	22
4.3.- VGG19 con Data Augmentation.....	24
4.4.- Fine Tuning.....	25
4.5.- Elección del modelo.....	26
5.- Aplicación.....	26
5.1.- Diseño.....	26
5.2.- Uso.....	26
6.- Conclusiones.....	28
7.- Lineas futuras.....	29
8.- Glosario.....	29
9.- Bibliografía.....	30

Índice de figuras

Figura 1: Fallecidos por Covid-19 en España por Comunidades Autonomas.....	7
Figura 2: Comparación de CNN con otros algoritmos de Deep Learning.....	11
Figura 3: Funcionamiento CNN.....	12
Figura 4: Max Pooling.....	12
Figura 5: Agrupación por diagnostico de radiografias.....	14
Figura 6: Estructura de red.....	16
Figura 7: Modelo secuencial cnn.....	17
Figura 8: Ejemplo de imágenes con data augmentation.....	18
Figura 9: Entrenamiento del modelo.....	19
Figura 10: Funcionamiento interno de la arquitectura VGG19.....	19
Figura 11: Arquitecturas VGG16 y VGG19.....	20
Figura 12: Modelo secuencial con Fine Tuning.....	22
Figura 13: Precisión del entrenamiento y validación del modelo general.....	23
Figura 14: Perdida del entrenamiento y validación del modelo general.....	23
Figura 15: Precisión del entrenamiento y validación del modelo con Data Augmentation.....	24
Figura 16: Perdida del entrenamiento y validación del modelo con Data Augmentation.....	24
Figura 17: Precisión del entrenamiento y validación del modelo con VGG19 y Data Augmentation	25
Figura 18: Perdida del entrenamiento y validación del modelo con VGG19 y Data Augmentation.	25
Figura 19: Precisión del entrenamiento y validación del modelo con VGG19 ,Data Augmentation y Fine Tuning.....	26
Figura 20: Perdida del entrenamiento y validación del modelo con VGG19 ,Data Augmentation y Fine Tuning.....	26
Figura 21: Aplicacion con Streamlit.....	28
Figura 22: Carga de imagen en la aplicación.....	28
Figura 23: Resultados obtenidos con la aplicación.....	29

Ficha del trabajo:

Título:	<i>Diagnósticos médicos a través de Deep Learning</i>
Autor:	<i>Eva M.^a Ruiz Macías</i>
Titulación	<i>Master Python avanzado</i>
Idioma	<i>Castellano</i>
Palabras clave	<i>Redes neuronales, deep learning, convolutional neural network, radiografías</i>

Resumen:

Las radiografías son pruebas rápidas e indoloras que generan imágenes de las estructuras internas del cuerpo, en especial de los huesos.

Los haces de rayos X pasan a través del cuerpo y se absorben en diferentes cantidades según la densidad del material a través del cual pasan. Los materiales densos, como huesos y metales, aparecen de color blanco en las radiografías. El aire en los pulmones aparece de color negro. La grasa y los músculos aparecen como sombras de color gris.

Como es sabido tras la gran pandemia sufrida estos años atrás, han sido de gran ayuda para el diagnóstico de la neumonía causada por el tan temido coronavirus.

Mediante deep learning seremos capaces de sacar conclusiones de las características de las radiografías en función de los diferentes resultados clínicos.

Utilizando el lenguaje Python se ejecutará un modelo de red neuronal convolucional, tratándose de mejorar mediante el ajuste fino y la transferencia de aprendizaje.

Los datos de imágenes radiográficas se han descargado de la plataforma Kaggle, en cuyos archivos vienen diferenciadas las diferentes categorías de neumonía.

Se crearán diferentes modelos, se procederá a su comparación y se mostrarán los resultados del mejor modelo creado.

Por último, se desarrollará una aplicación con el framework Streamlit, para mostrar los resultados.

Abstract:

Radiographs are fast and unpain test, wich generates body internal streucture images, specially bones.

XRay waves pass through the body and get absorbed at different levels depending on the material density they pass. Dense materiales like Bones or metal appear as white in the bone scan. The Air in the lungs appears as black, and fat and muscle as grey shadows.

As we know after the great pandemic suffered along last years, they been very helpful to diagnose pneumony cases, through Deep learnig we could get conclusions about radiography characteristics depending on clinical results.

Using Python language we develop a neuronal convolucional net model, improved by fine adjust and learning Transfer.

Radiography imagen data had been download from Kaggle platform, in wich files appears different categories of pneumonia.

We create different models, after that they been compared and get the results of the best model.

Finally, an application will be developed with the Streamlit framework, to show the results.

1. Introducción

1.1 Contexto y Justificación del Trabajo

En estos años anteriores, hemos sufrido una pandemia de grandes proporciones.

Que algo así pudiera pasar era algo impensable para la mayoría de nosotros.

La proporción de enfermos a nivel mundial alcanzó cifras insospechadas, y la propagación de la enfermedad fue rapidísima, debido a que el covid se propaga cuando una persona infectada exhala gotitas y partículas respiratorias muy pequeñas que contienen el virus. Estas gotitas y partículas respiratorias pueden ser inhaladas por otras personas o depositarse sobre sus ojos, nariz o boca. En algunas circunstancias, pueden contaminar las superficies que tocan[1].

Si nos fijamos en el número de víctimas en España a lo largo del tiempo, las cifras son poco menos que impresionantes:

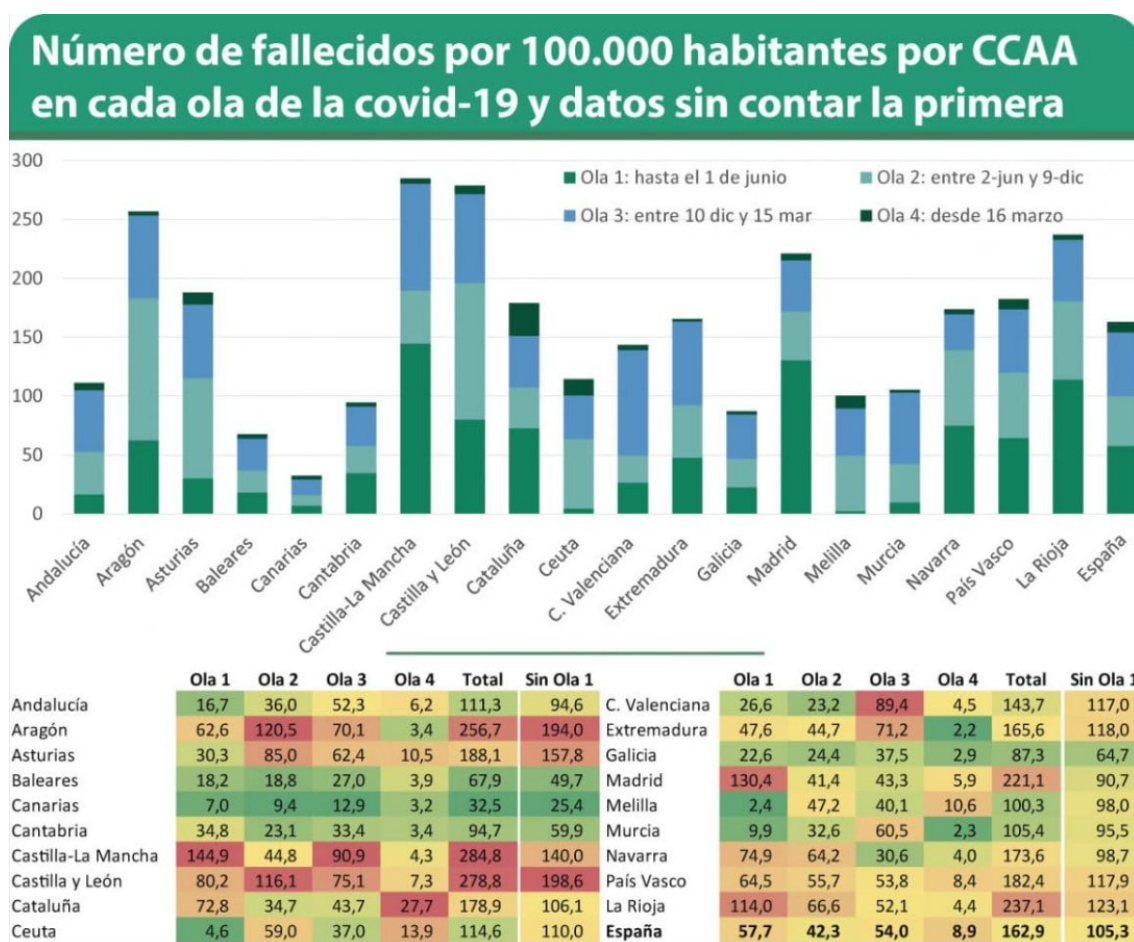


Figura 1: Fallecidos por Covid-19 en España por Comunidades Autonomas

Las imágenes radiográficas han sido de gran utilidad a la hora de diagnosticar la enfermedad de manera rápida, además de los test de antígenos, ya que las PCR son la prueba más fiable, pero los resultados son más lentos.

Una radiografía es una prueba rápida e indolora que genera imágenes de las estructuras internas del cuerpo, en especial de los huesos.

Los haces de rayos X pasan a través del cuerpo y se absorben en diferentes cantidades según la densidad del material a través del cual pasan. Los materiales densos, como huesos y metales, aparecen de color blanco en las radiografías. El aire en los pulmones aparece de color negro. La grasa y los músculos aparecen como sombras de color gris.[2]

Los servicios sanitarios de todo el mundo han sufrido un gran saturación, y se han visto desbordados debido a la cantidad de personas enfermas que necesitaban ser diagnosticadas y atendidas.

El aprendizaje automático puede ser de ayuda dada la gran cantidad de radiografías que se realizaron y se siguen realizando. Más concretamente con deep learning se puede contribuir al procesamiento y clasificación de imágenes médicas, lo cual ayudaría a la des congestión de los servicios sanitarios, haciéndoles ganar un tiempo de vital importancia.

1.2 Objetivos del Trabajo

1.2.1 Objetivos generales

Creación de un algoritmo eficaz a la hora de procesar y evaluar radiografías de tórax, siendo capaz de distinguir entre pacientes sanos y pacientes con algún tipo de neumonía, a través de deep learning.

1.2.2 Objetivos específicos

Distinción entre pacientes sanos, con opacidad pulmonar, neumonía vírica o neumonía causada por covid, mediante convolucional neural network y creación de una aplicación para que su uso sea posible para cualquier persona.

El desarrollo del trabajo se encuentra en el repositorio de Github con el siguiente [enlace](#).

1.3 Enfoque y método seguido

1.3.1 Enfoque

La base de datos elegida pertenece al repositorio kaggle, y consta de imágenes de radiografías de pacientes clasificadas en 4 grupos: sanos, neumonía vírica, neumonía causada por covid, y con opacidad pulmonar.

Debido a la gran cantidad de características inherentes al tratamiento de imágenes se ha optado por la utilización de deep learning, más concretamente, de redes neuronales convolucionales.

Las CNN son un tipo de red neuronal de alimentación hacia adelante inspirada en la arquitectura de la corteza visual de los cerebros de las personas y los animales. Consisten en múltiples capas de filtros convolucionales de una o más dimensiones. Después de cada capa, por lo general se añade una función para realizar un mapeo causal no-lineal.

Como cualquier red empleada para clasificación, al principio estas redes tienen una fase de extracción de características, compuesta de neuronas convolucionales, luego hay una reducción por muestreo y al final tendremos neuronas de perceptrón mas sencillas para realizar la clasificación final sobre las características extraídas.

La fase de extracción de características se asemeja al proceso estimulante en las células de la corteza visual. Esta fase se compone de capas alternas de neuronas convolucionales y neuronas de reducción de muestreo. Según progresan los datos a lo largo de esta fase, se disminuye su dimensionalidad, siendo las neuronas en capas lejanas mucho menos sensibles a perturbaciones en los datos de entrada, pero al mismo tiempo siendo estas activadas por características cada vez más complejas.

1.3.2 Enumeración de las etapas del trabajo

- 1.- Análisis exploratorio de datos
- 2.- Procesamiento de imágenes: transformación matricial, normalización de valores y redimensionado de imágenes.
- 3.- Creación de la red neuronal
- 4.- Entrenamiento del modelo
- 5.- Validación del modelo
- 6.- Evaluación y ejecución de posibles mejores.
- 7.- Validación de la mejoras
- 8.- Creación de una aplicación que permita el uso general.

1.4 - Estudios consultados

Se puede observar como en la mayoría de los trabajos se prefieren las imágenes radiográfica en detrimento de otro tipo imágenes medicas, aunque algunos grupos de investigación las han combinado[3]

En muchos de ellos también se opta por la diferenciación entre sanos y enfermos, en lugar de clasificar en diferentes tipos (multiclase).[3] [4] [5] [6]

Se han usado redes neuronales convolucionales creadas de la nada [7] , y se han usado uno o varios modelos preentrenados [8]

Queda referenciado cada uno de los casos por si resulta de interés su consulta.

2.- Marco teórico

2.1.- Deep Learning

El deep learning es una de las partes con mayor proyección de Machine learning y de la IA.

Deep learning, es decir, aprendizaje profundo, pero.... ¿de verdad aprendizaje?

Esta palabra hace que se produzca cierta confusión, porque claramente un ordenador, no aprende solo. Esta palabra, aprendizaje, lo que significa en realidad en este contexto es una búsqueda para poder conseguir mejores **representaciones** de los datos con los que estemos trabajando.

Esto se verá más claro con un ejemplo:

Imaginemos que tenemos una serie de puntos, unos azules y otros verdes, representado en un eje de coordenadas cartesianas y queremos separar los grupos con una línea recta. Si tenemos los puntos diseminados se podría decir que es imposible trazar una recta y que los grupos queden separados.

Pero esto puede ser posible si **representamos** los datos de manera diferente. Si cogemos las coordenadas cartesianas y las transformamos en coordenadas polares, se nos quedaría un grupo de puntos azules a un lado y el otro grupo de puntos verdes al otro, por lo que fácilmente podríamos trazar una recta para separarlos. En este ejemplo, hemos elegido una mejor forma de representar los datos para conseguir una mejor representación.

Si creamos un programa que pueda buscar diferentes representaciones y calcular el tanto por ciento de categorías que se clasifican de manera correcta, estaríamos haciendo lo mismo que antes, pero con el ordenador.

El deep learning es en realidad representation learning, es decir, el aprendizaje de representaciones usando distintos tipos de redes neuronales para conseguir o “aprender” la mejor representación de los datos que estemos tratando.[9]

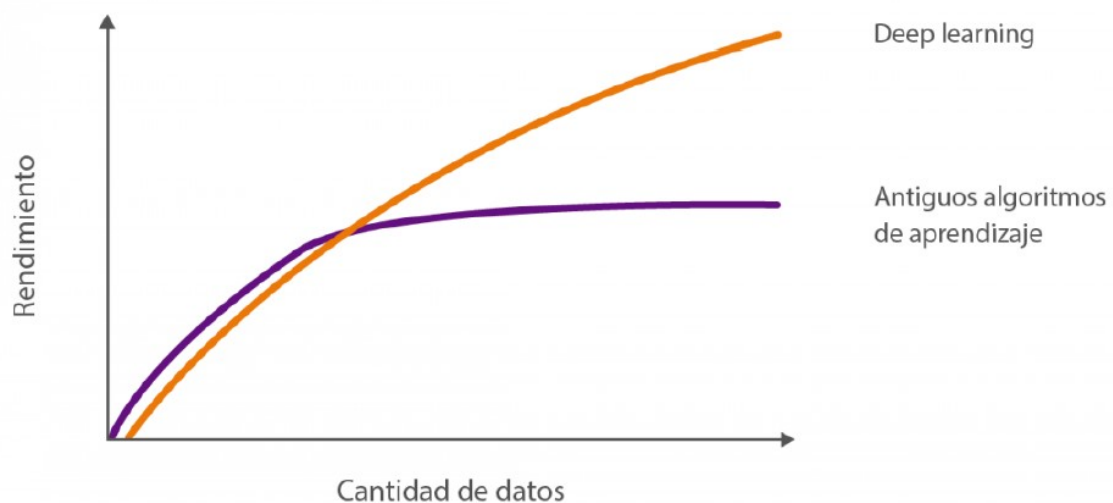


Figura 2: Comparación de CNN con otros algoritmos de Deep Learning

2.2 Convolutional Neural Network

Uno de los algoritmos más importantes dentro de deep learning son las redes neuronales convolucionadas.

El CNN es un algoritmo que toma como entrada una imagen, le asigna pesos y sesgos a varios aspectos de esta, y el procesamiento previo requiere menos esfuerzo comparado con otros algoritmos.

Si ponemos como ejemplo el reconocimiento de una imagen de gatos, primero tendríamos que ver que es lo que identifica a un gato, tiene 4 patas, orejas, nariz, boca.....pero eso no es suficiente, ya que puede tener todos esos elementos y no ser un gato. Tendríamos que identificar además bordes, líneas, tamaños relativos...

El objetivo del algoritmo sería reducir las imágenes de forma que sea más fácil procesarlas, pero sin perder características que pueden ser importantes para la predicción. Una imagen, es una matriz de píxeles, con un ancho y un largo determinados, y también se diferencian las imágenes en blanco y negro y las de

color (1 canal, 3 canales). El número de píxeles de la imagen determinará el número de neuronas.

En CNN tenemos grupos de neuronas especializadas en distintas cosas: una capa convolucional, que “aprenderá” patrones locales y que detectará características como bordes, aristas, color... No tiene por qué existir una sola capa, pueden existir varias. Estas capas operan sobre tensores 3D, o feature maps. En una convolución recibimos unos datos de entrada, y a estos se les aplicará un filtro (kernel) que nos dará una serie de características y se podrán discriminar las más importantes. Se recorrerán todas las neuronas de entrada y se obtendrá otra matriz, que será una capa oculta.

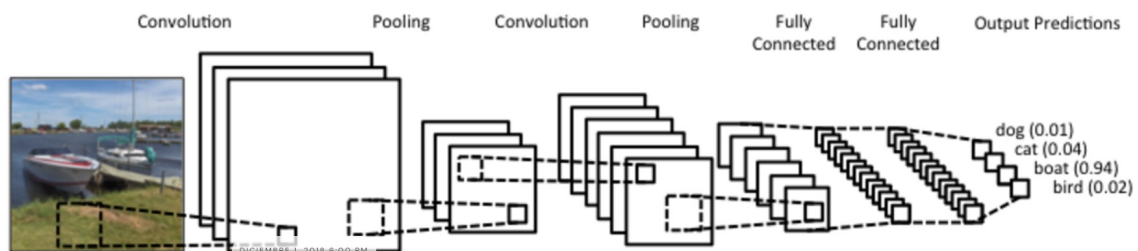


Figura 3: Funcionamiento CNN

Además de estas capas, hay capas de pooling, que se aplican después de las convolucionales. Hacen una simplificación de los resultados obtenidos en las últimas capas y condensan la información, para disminuir la potencia computacional necesaria para procesar los datos. Algunas funciones para condensar la información son max-pooling o average-pooling.

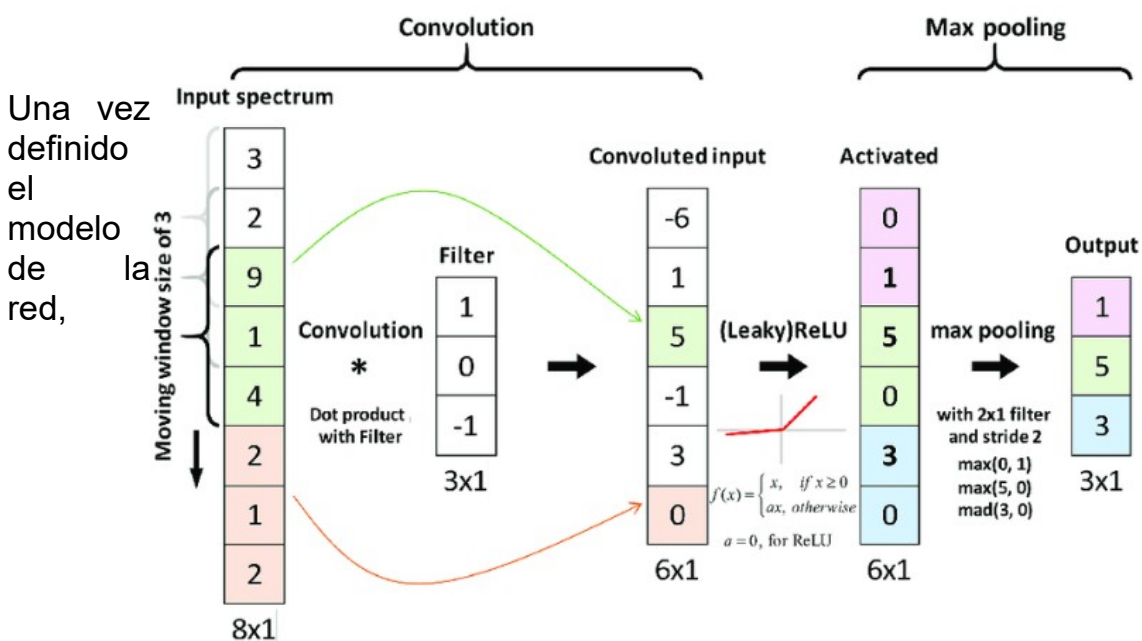


Figura 4: Max Pooling

Tendríamos que aplanar el resultado y pasar al entrenamiento del modelo.

Pero al llegar aquí, nos podemos preguntar, cuales han sido los argumentos para saber identificar, por ejemplo, los bordes o que valores tendrán estos filtros. Esto se hace mediante un mecanismo llamado retropropagacion o backpropagation. [10][11][12]

3.- Desarrollo del trabajo

3.1.- Descripción del dataset

Un equipo de investigadores de la Universidad de Qatar, Doha y la Universidad de Dhaka, Bangladesh, en colaboración con médicos de Pakistán y Malasia, crearon una base de datos de imágenes de radiografías de tórax para casos positivos de covid-19 junto con imágenes de neumonía normal y viral. Dicha base de datos se ha ido actualizando por etapas, añadiendo nuevas imágenes.[13].

En el dataset se pueden ver imágenes de radiografías clasificadas en 4 grupos: normal, neumonía vírica, neumonía causada por el covid-19, y opacidad pulmonar. Se puede acceder a los datos pinchando [aquí](#).

3.2.- Carga de datos

Los datos descargados del repositorio de kaggle están estructurados como sigue: 1 carpeta llamada "COVID- 19_Radiography_Dataset", que a su vez tiene 4 subcarpetas llamadas: COVID, Lung_Opacity, Normal y Viral Pneumonia.

Aunque son muchas las librerías que se utilizarán, las más importantes son "keras" y "Tensorflow".

Para la carga de datos se usará la librería "os". Las radiografías se guardarán en una lista llamada Imagenes en formato PIL, y el tipo de radiografía en otra lista llamada diagnostico, en formato cadena.

3.3.- Breve descripción de los datos

Se creará un dataframe con la librería pandas, y le pediremos que nos de la longitud de una de las columnas para ver cuantos datos tenemos (todas las columnas tienen la misma longitud), lo que nos devolverá como resultado que nuestro dataframe tienen 21165 imágenes.

Si le pedimos información sobre nuestro dataframe, nos devolverá el número de datos y el número de variables que tenemos, que son tres.

Si agrupamos nuestros datos (con groupby) por el diagnostico, los resultados serán: 3616 con covid, 6012 con opacidad pulmonar, 10192 normales y 1345 con neumonía vírica. Se muestran los resultados en el siguiente gráfico:

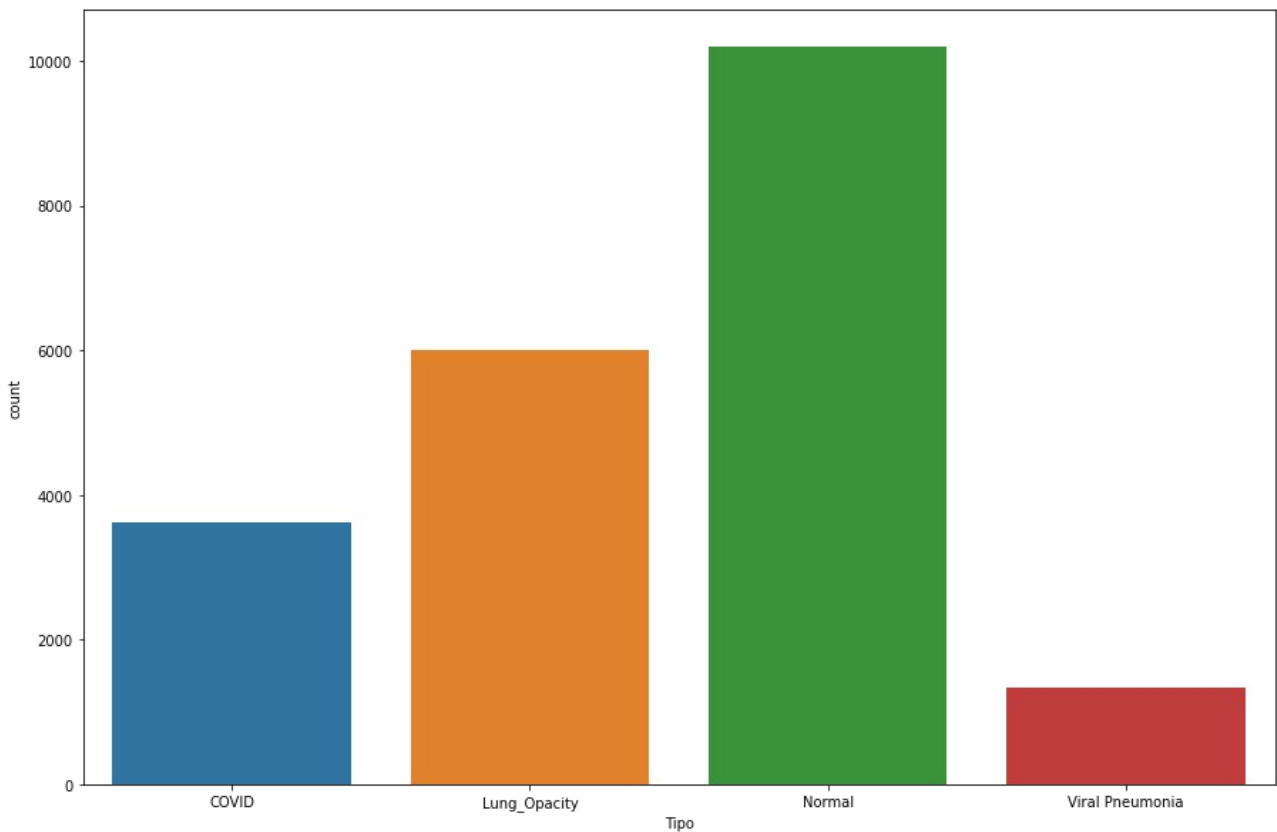


Figura 5: Agrupación por diagnostico de radiografias

3.4.- Reducción del dataset y preprocesamiento

Dada la imposibilidad de procesar el dataset completo en un equipo doméstico por su gran dimensión, se procede a seleccionar de manera aleatoria 1300 imágenes de cada tipo, para su procesamiento. Se creará un nuevo directorio con sus subcarpetas correspondientes para su almacenamiento.

Se creará una función de preprocesado de datos que recibirá imágenes en formato PIL. La función toma una imagen como entrada y realiza las siguientes operaciones:

- Si la imagen no está en modo RGB, se convierte a modo RGB.
- La imagen se transforma en una matriz de píxeles(convertirá la imágenes en vector numpy).
- Si el tamaño de la imagen no es (150, 150), se redimensiona a (150, 150) para un menor coste computacional.
- Los píxeles se normalizan dividiéndolos por 255 ya que todo debe estar en la misma escala.

Codificación de variables y conversión a vectores

Se le asignarán los valores numéricos (cero, uno, dos y tres) a la lista de cadenas con los tipos de imágenes. Se le asignará a cada clase su valor numérico mediante el método `get`.

Tenemos por tanto una lista cuyos elementos son cada una de las imágenes y su etiqueta. Con `np.array()` se convertirá en vectores, para poder trabajar con los modelos de CNN.

Creación de los conjuntos de entrenamiento, prueba y validación

El método que se utiliza para evaluar el rendimiento de un método de Machine Learning se basa en dividir los datos en diferentes conjuntos: uno para el proceso de entrenamiento y otro destinado a la validación de modelo.

La división se realiza por instancias, nunca por características. Es decir, del conjunto de datos original se selecciona un subconjunto de observaciones con las que alimentará al algoritmo de entrenamiento. Con las instancias restantes, y sin aportar la variable a predecir, se realizan predicciones. Finalmente, se comparan las predicciones realizadas frente a las reales o esperadas, pudiendo cuantificar el resultado.

Esta técnica de evaluación como tal no se suele utilizar en la práctica para valorar el rendimiento de un método. No obstante, es la base sobre la que se fundamentan todas las técnicas de evaluación existentes. El principal inconveniente de este método es que puede tener una alta dispersión en la precisión.

Diferentes segmentaciones del conjunto de datos para entrenamiento y validación pueden llegar a obtener diferencias significativas en la respuesta del algoritmo. El módulo de `scikit-learn` **model_selection** cuenta con la función **train_test_split** que parte los conjuntos de datos en subconjuntos de entrenamiento y validación de modo aleatorio.[14] [15] [16]

Los conjuntos de entrenamiento, prueba y validación constarán de 3328, 1040 y 832 imágenes respectivamente.

Utilizaremos one-hot-encoding mediante la función `to_categorical()` de la librería `keras` para codificar las etiquetas de las imágenes (se crearan 2 variables por cada etiqueta y se le asignará 0 o 1 en función de su pertenencia)

3.5 Convolution Neural Network

En la clase `sequential` se van a ir añadiendo de manera ordenada las distintas capas.

Las redes neuronales convolucionales son un tipo de red neuronal de alimentación hacia adelante inspirada en la arquitectura de la corteza visual en los cerebros de

los humanos. Es una buena opción para tareas de procesamiento de imágenes. Usaremos una red neuronal convolucional 2D con 4 capas de convolución, con kernel de tamaño 3x3, lo que significa que tendremos 9 perceptrones interconectados. Se alimentarán los primeros 32 elementos del vector de entrada. Con MaxPooling2D se comprimirá el espacio de datos sin afectar mucho su información. Esto ayuda a acelerar el procesamiento y entrenamiento. Se repetirá este proceso.

La instrucción Flatten convierte los elementos de la matriz de imágenes de entrada en un array plano.

Con Dropout se eliminarán un 50 % de las conexiones.

Por último se añade una capa densa (todas las neuronas se interconectan con todas las neuronas de la siguiente capa) con el número de salidas, que serán el total de clases que haya.

Con summary se verá la estructura de la red.

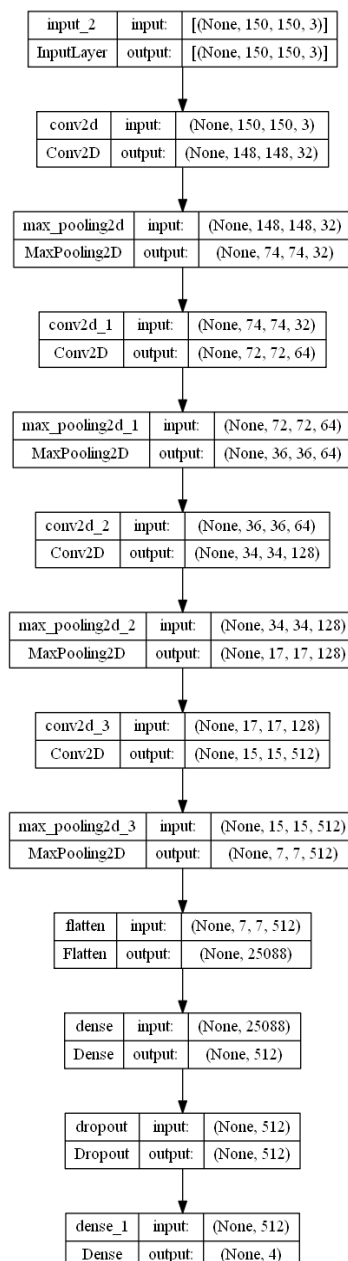


Figura 6: Estructura de red

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_2 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 17, 17, 128)	0
conv2d_3 (Conv2D)	(None, 15, 15, 512)	590336
max_pooling2d_3 (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 512)	12845568
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 4)	2052
Total params: 13,531,204		
Trainable params: 13,531,204		
Non-trainable params: 0		

Figura 7: Modelo secuencial cnn

3.6.- Entrenamiento y validación

Se utilizará la función `fit()` para el entrenamiento.

El `batch_size` (tamaño) se ha fijado en 128, es decir se usarán 128 imágenes para la actualización del gradiente. Se han establecido 20 épocas (epoch).

Para ver la precisión del modelo se usará la función `evaluate()`.

3.7.- Posibles mejoras del modelo

Data Augmentation

Una imagen de entrada, será procesada tantas veces como epoch se ejecuten, por lo que la red puede llegar a “memorizar” la imagen si se entrena demasiado.

Lo que se hace con el aumento de datos es aplicar transformaciones de manera aleatoria cada vez que se introduzca de nuevo la imagen en la red.

Algunas transformaciones posibles son:

- Ajustar brillo, contraste o cualquiera de los parámetros de la imagen.
- Introducir ruido en la imagen.
- Redimensionar
- Recortar
- Rotar
- Cualquier combinación de las anteriores (o con más posibles transformaciones).

Se contará por tanto de más información para el entrenamiento, sin necesidad de obtener nuevos datos, y con poca inversión de tiempo.

Se tendrán en cuenta entonces distintas imágenes vistas desde diferentes perspectivas y con distintos parámetros de la imagen (como luminosidad o brillo), lo cual hará que se tenga menos probabilidad de sobreajuste, y el modelo generalice mejor.

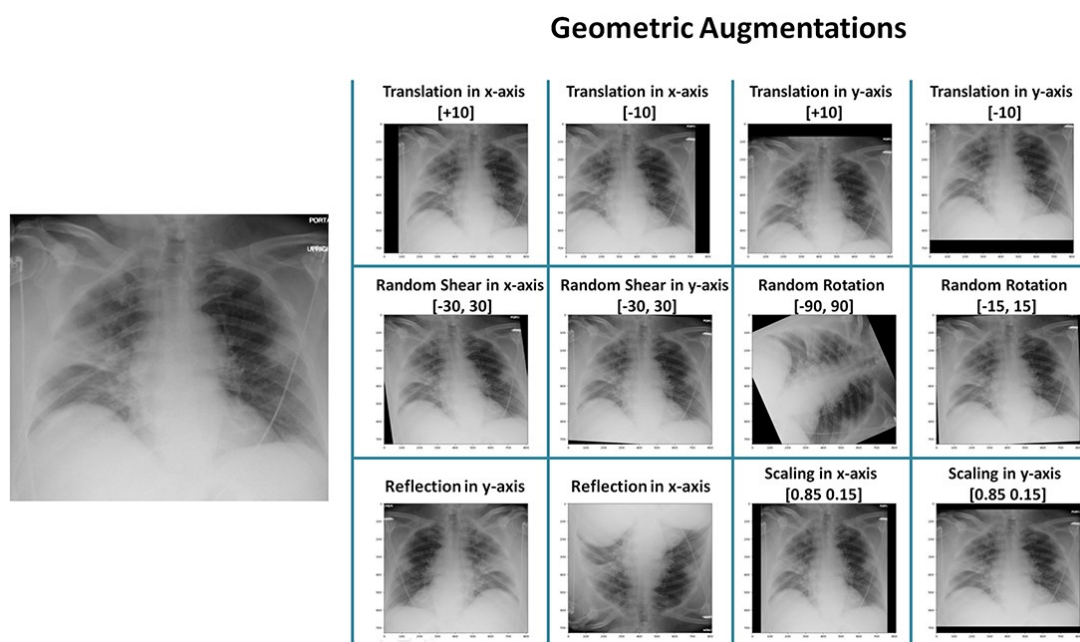


Figura 8: Ejemplo de imágenes con data augmentation

En el caso que nos ocupa, se ha creado un generador con `ImageDataGenerator()` para cada una de los conjuntos de datos train, test y valid, aunque solo se hará un aumento de datos en el conjunto train (para generar los datos se cogerán imágenes de train y de valid).

Se realizan las siguientes transformaciones:

- Alejamiento o acercamiento de un 20% (`zoom_range`)

- Volteo horizontal de forma aleatoria (horizontal_flip)
- Transformaciones aleatorias de corte (shear_range)

Se creará después la red neuronal y se procederá al entrenamiento del modelo:

```
0.8208
Epoch 45/50
80/80 [=====] - 54s 677ms/step - loss: 0.5001 - accuracy: 0.8012 - val_loss: 0.5288 - val_accuracy:
0.7771
Epoch 46/50
80/80 [=====] - 54s 672ms/step - loss: 0.5026 - accuracy: 0.8069 - val_loss: 0.4744 - val_accuracy:
0.8500
Epoch 47/50
80/80 [=====] - 53s 665ms/step - loss: 0.5381 - accuracy: 0.8031 - val_loss: 0.5247 - val_accuracy:
0.8021
Epoch 48/50
80/80 [=====] - 54s 666ms/step - loss: 0.5586 - accuracy: 0.7819 - val_loss: 0.4566 - val_accuracy:
0.8313
Epoch 49/50
80/80 [=====] - 54s 669ms/step - loss: 0.5315 - accuracy: 0.7956 - val_loss: 0.4842 - val_accuracy:
0.8208
Epoch 50/50
80/80 [=====] - 54s 669ms/step - loss: 0.5583 - accuracy: 0.7962 - val_loss: 0.4716 - val_accuracy:
0.8354
```

Figura 9: Entrenamiento del modelo

Se puede apreciar será costoso computacionalmente y tardará alrededor de 2 horas en ejecutarse.

Arquitectura VGG19

La arquitectura VGG16 y VGG19 en Deep Learning fue una de las primeras en aparecer. Fue introducida por Simonyan y Zisserman en 2014 con su artículo titulado *Very Deep Convolutional Networks for Large Scale Image Recognition*.

(vgg19 network). Internamente este tipo de arquitectura en el aprendizaje profundo se vería así:

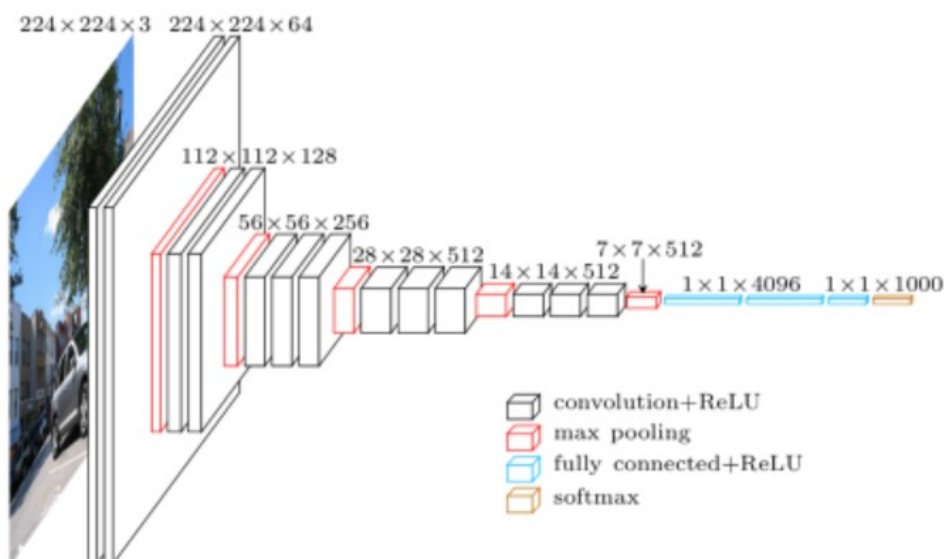


Figura 10: Funcionamiento interno de la arquitectura VGG19

La arquitectura VGG16 y VGG19 en Deep Learning se trata de una arquitectura bastante simple, usando solo bloques compuestos por un número incremental de capas convolucionales con filtros de tamaño 3×3. Además, para reducir el tamaño de los mapas de activación que se van obteniendo, se intercalan bloques *maxpooling* entre los convolucionales, reduciendo a la mitad el tamaño de estos mapas de activación. Finalmente, se utiliza un bloque de clasificación compuesto por dos capas densas de 4096 neuronas cada una, y una última capa, que es la de salida, de 1000 neuronas.

El 16 y 19 se refiere al número de capas con pesos que tiene cada red (las convolucionales y las densas, las de *pooling* no se cuentan). Se corresponden con las columnas D y E de la tabla que verás a continuación; esta expone este tipo de arquitectura vgg19 vs vgg16 en Deep Learning:

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figura 11: Arquitecturas VGG16 y VGG19

El motivo por el que se muestran otras arquitecturas en la tabla es porque, por aquel entonces, a Simonyan y Zisserman les costó bastante entrenar su arquitectura de forma que convergiera. Como no lo conseguían, lo que se les

ocurrió fue entrenar primero redes con arquitecturas más sencillas, y una vez estas convergían y estaban entrenadas, aprovechaban sus pesos para inicializar la siguiente red, que era un poco más compleja. Así hasta llegar a la VGG19.

La inicialización de los pesos tiene gran importancia, como ya hemos visto. A este proceso se le conoce como *pretraining*.

Esta red, sin embargo, tiene un par de desventajas:

En primer lugar, tarda muchísimo tiempo en entrenar.

Por otra parte, tiene un número muy elevado de parámetros por consolidar.[17]

Se importará VGG19 con la función `VGG19()` de la librería Keras con el argumento `"include_top=False"` para que no se incluya el clasificador de la red preentrenada ya que debemos crear nuestro propio clasificador que se ajuste a nuestros datos.

Se realizará el procesamiento de las imágenes con `vgg19.preprocess_input()`, que escalará los píxeles y se utilizará el método `predict()` para obtener las características de los grupos de entrenamiento, test y validación. Estas quedarán almacenadas en matrices.

Por último se establecerá el clasificador densamente conectado y se entrenará el modelo con las características extraídas.

Se realizará también una transferencia de aprendizaje junto con un aumento de datos para ver si se consigue alguna mejora.

En este caso se congelará la base convolucional del modelo y se establecerá un clasificador densamente conectado propio. Hecho esto, se entrenará el modelo.

Ajuste fino

Otra técnica ampliamente utilizada en la reutilización de modelos, complementaria a la *Feature Extraction*, es *Fine-Tuning*, que consiste en hacer un ajuste más fino y entrenar también algunas de las capas finales de la base convolucional del modelo usado para la extracción de características (que hasta ahora se mantenía congelado), y entrenar conjuntamente tanto la parte agregada del clasificador como estas capas. Esto se llama *Fine-Tuning* porque se trata de un "ajuste fino" de las representaciones más abstractas del modelo que se está reutilizando como base. [18]

Se descongelará la parte convolucional del modelo VGG19 y se congelarán todas las capas convolucionales desde la 4ª hasta la última. Por último se compilará el modelo y se volverá a entrenar utilizando también aumento de datos.

Figura 12: Modelo secuencial con Fine Tuning

Model: "vgg19"

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 299, 299, 3)]	0
block1_conv1 (Conv2D)	(None, 299, 299, 64)	1792
block1_conv2 (Conv2D)	(None, 299, 299, 64)	36928
block1_pool (MaxPooling2D)	(None, 149, 149, 64)	0
block2_conv1 (Conv2D)	(None, 149, 149, 128)	73856
block2_conv2 (Conv2D)	(None, 149, 149, 128)	147584
block2_pool (MaxPooling2D)	(None, 74, 74, 128)	0
block3_conv1 (Conv2D)	(None, 74, 74, 256)	295168
block3_conv2 (Conv2D)	(None, 74, 74, 256)	590880
block3_conv3 (Conv2D)	(None, 74, 74, 256)	590880
block3_conv4 (Conv2D)	(None, 74, 74, 256)	590880
block3_pool (MaxPooling2D)	(None, 37, 37, 256)	0
block4_conv1 (Conv2D)	(None, 37, 37, 512)	1180160
block4_conv2 (Conv2D)	(None, 37, 37, 512)	2359808
block4_conv3 (Conv2D)	(None, 37, 37, 512)	2359808
block4_conv4 (Conv2D)	(None, 37, 37, 512)	2359808
block4_pool (MaxPooling2D)	(None, 18, 18, 512)	0
block5_conv1 (Conv2D)	(None, 18, 18, 512)	2359808
block5_conv2 (Conv2D)	(None, 18, 18, 512)	2359808
block5_conv3 (Conv2D)	(None, 18, 18, 512)	2359808
block5_conv4 (Conv2D)	(None, 18, 18, 512)	2359808
block5_pool (MaxPooling2D)	(None, 9, 9, 512)	0
Total params: 20,024,384		
Trainable params: 20,024,384		
Non-trainable params: 0		

4.- Resultados

Se compararán ahora los resultados obtenidos con cada uno de los modelos, teniendo en cuenta las métricas de precisión y perdida.

4.1.- Modelo general

Modelo con 4 capas de convolución y entrenado con 1300 imágenes:

Precisión	Perdida
0,853	0,499

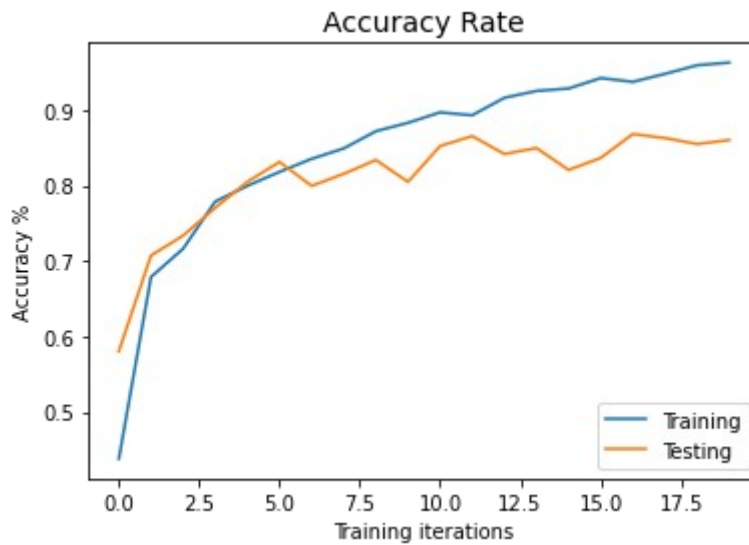


Figura 13: Precisión del entrenamiento y validación del modelo general

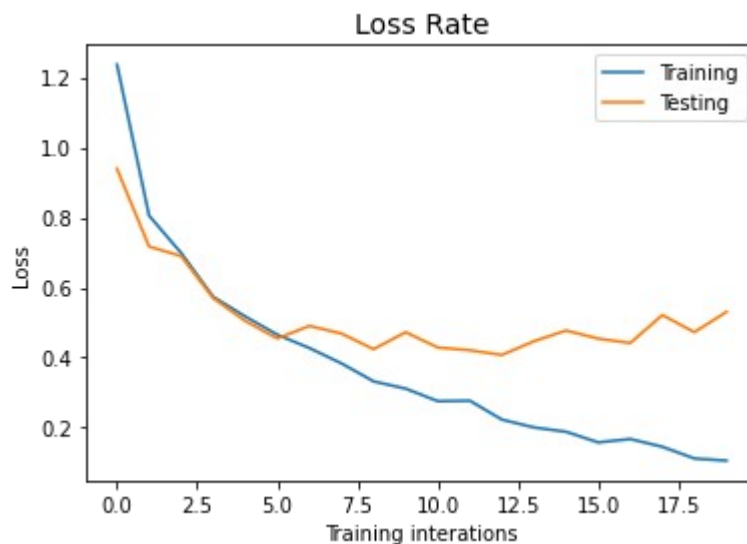


Figura 14: Perdida del entrenamiento y validación del modelo general

Podemos ver como la precisión en el entrenamiento sube con rapidez hasta alcanzar casi el 1, mientras en el test tarda poco en llegar a su tope y se queda estancada. Se observa el patrón de overfitting, es decir, el modelo aprende unas características determinadas y no puede generalizar los resultados, dandonos solo respuesta a datos que sean muy parecidos a los del entrenamiento.

4.2.- Data Augmentation

Se han aumentado las imágenes de entrenamiento mediante un generador de imágenes.

Precisión	Perdida
0.8317	0.4675

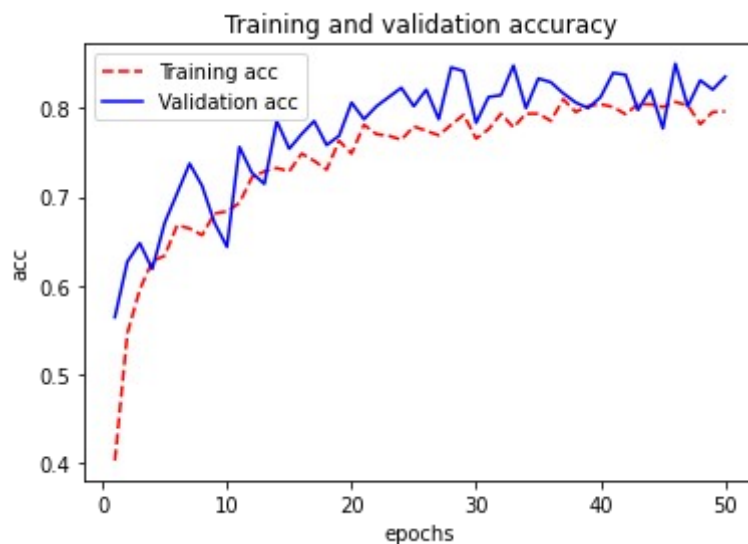


Figura 15: Precisión del entrenamiento y validación del modelo con Data Augmentation

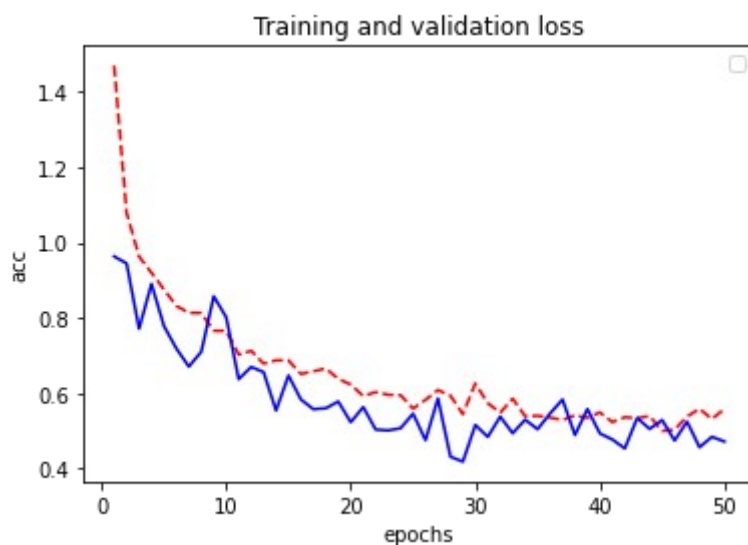


Figura 16: Perdida del entrenamiento y validación del modelo con Data Augmentation

A pesar de que las métricas puedan parecer peores que en el caso anterior, aquí no se aprecia overfitting, por lo que este modelo ofrece mejores resultados.

4.3.- VGG19 con Data Augmentation

Se ha usado la red pre-entrenada VGG19 más un aumento de datos.

Precisión	Perdida
0.8458	0.4530

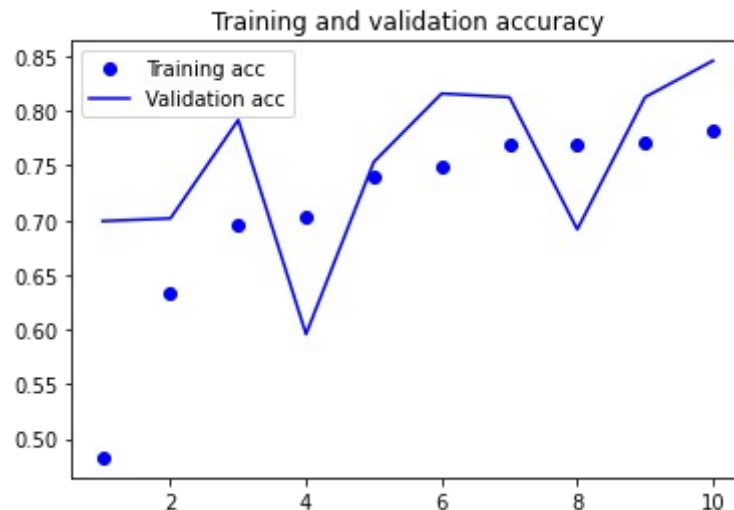


Figura 17: Precisión del entrenamiento y validación del modelo con VGG19 y Data Augmentation

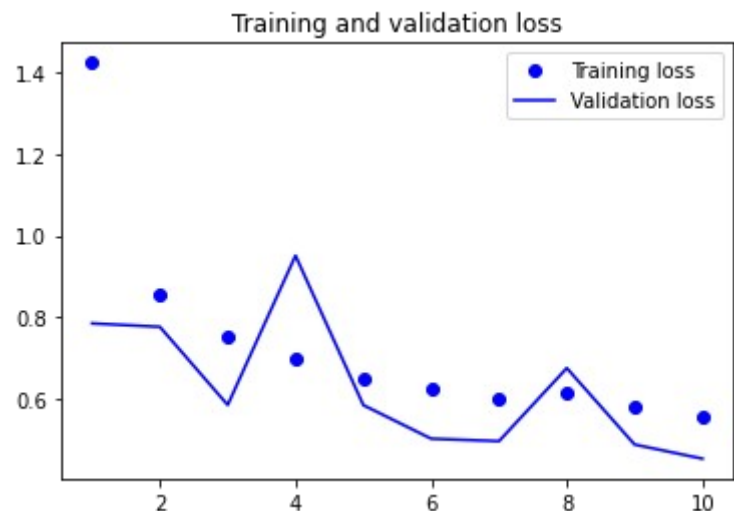


Figura 18: Perdida del entrenamiento y validación del modelo con VGG19 y Data Augmentation

Mejoran las métricas, y se observa en la gráfica como sus valores con respecto a train y test son medianamente parejos.

Tampoco se observa overfitting, por lo que mejoran los resultados con respecto a los modelos anteriores.

4.4.- Fine Tuning

Por último tenemos un modelo VGG19 con aumento de datos y ajuste fino.

Precisión	Perdida
0.8492	0.4198

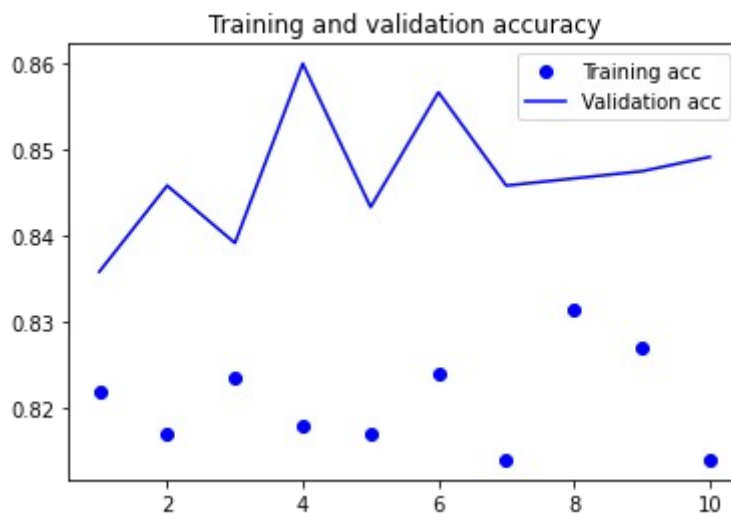


Figura 19: Precisión del entrenamiento y validación del modelo con VGG19 ,Data Augmentation y Fine Tuning

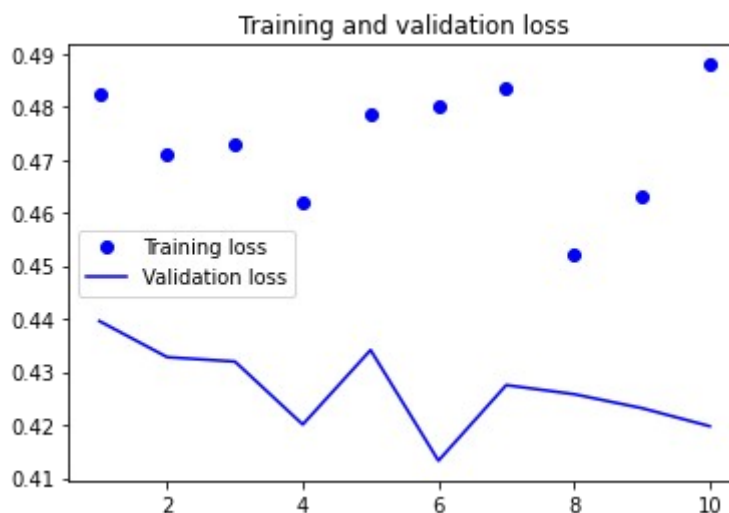


Figura 20: Perdida del entrenamiento y validación del modelo con VGG19 ,Data Augmentation y Fine Tuning

Obtenemos unos resultados muy parecidos al caso anterior, por lo que podríamos quedarnos con cualquiera de los dos modelos.

4.5.- Elección del modelo

Si aumentásemos el número de etapas en los modelos VGG19 y Fine Tuning, probablemente obtendríamos mejores resultados, pero estos tardaron aproximadamente 1 hora y 50 minutos en compilar, por lo que, para aumentar el número de etapas sería conveniente tener un ordenador más potente.

Podríamos optar por la elección de cualquiera de los dos, ya que aunque la precisión es mayor en el primero, también lo es la pérdida, por lo que quedaría a nuestra elección optar por uno o por otro.

5.- Aplicación

5.1.- Diseño

Se ha creado una aplicación con el modelo que nos da los mejores resultados con el framework streamlit.

Primero, importaremos las librerías necesarias para la ejecución, a parte de la librería streamlit.

Se pondrá el path donde está guardado el modelo preentrenado a utilizar, que es un archivo h5.

Se procederá a cargar la imagen utilizando la instrucción `st.file_uploader`, la cual tendrá que tener formato jpg, jpeg o png. En caso de cargar un archivo no válido, nos lanzará un aviso. La imagen se almacenará en un buffer.

Se redimensionará la imagen, se preprocesará y se tendrá que multiplicar por 255, ya que al usar PIL la imagen se carga con valores entre 0 y 1 y la red neuronal se entrenó con imágenes de 0 a 255.


Por último se llamará a la predicción del modelo, que nos devolverá con que precisión la radiografía que hayamos cargado pertenece a una clase.

5.2.- Uso

La imagen que se mostrará al ejecutar la aplicación será la siguiente:

Clasificación de radiografías de torax

Cargue una imagen:

 Drag and drop file here
Limit 200MB per file • PNG, JPG, JPEG

Browse files

Oops! Esto no es una imagen, carga de nuevo.


Figura 21: Aplicacion con Streamlit

Se pide que se cargue una imagen, la cual se debe tener disponible.

Se seleccionarán imágenes de prueba guardadas en una carpeta para este objetivo. Hay una imagen de cada tipo, de las cuales sabemos el resultado y las usaremos para comprobar si la aplicación funciona.

Clasificación de radiografías de torax

Cargue una imagen:

 Drag and drop file here
Limit 200MB per file • PNG, JPG, JPEG

Browse files

Oops! Esto no es una imagen, carga de nuevo.

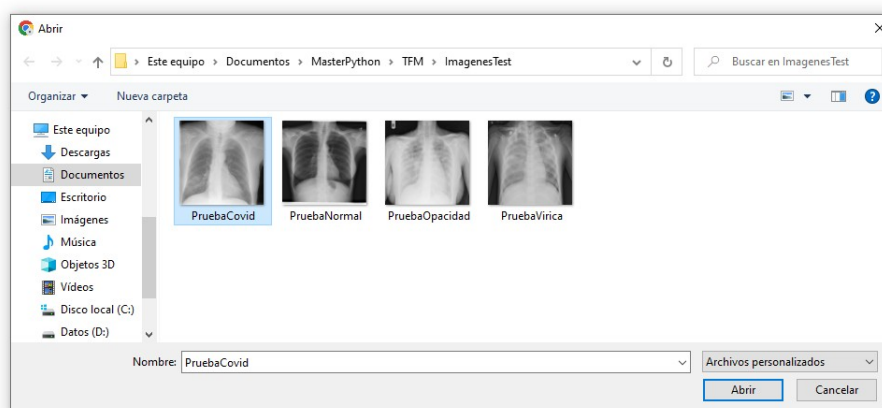


Figura 22: Carga de imagen en la aplicación

Una vez seleccionada la imagen, se realiza la predicción como se muestra a continuación:



Figura 23: Resultados obtenidos con la aplicación

6.- Conclusiones

A lo largo del trabajo se han desarrollado varios modelos de redes neuronales convolucionales, desde el modelo más sencillo a modelos algo más elaborados.

Los resultados en general son buenos, superando el 80% a nivel de predicción en todos los casos, aunque algunos de ellos como el modelo original no sea realmente bueno, ya que tiene overfitting y podría no generalizar bien.

Los modelos de vgg19 con aumento de datos y el de fine tuning son los que mejores resultados ofrecen, aunque computacionalmente son costosos.

A la hora de mirar los resultados de los modelos no debemos guiarnos solamente por la precisión, ya que en el caso que nos ocupa, que nos diera un falso negativo en una radiografía, sería bastante grave. Para no extenderse demasiado se ha mirado solo la precisión y la pérdida, aunque un estudio más pormenorizado nos serviría para ver cual es el modelo que mejor se ajusta a lo que queremos.

7.- Lineas futuras

El uso de modelos de deep learning en el campo de la salud podría suponer una gran ayuda para el personal sanitario.

Los modelos mostrados podrían mejorarse, ejecutandolos en máquinas más potentes, lo cual nos permitiría aumentar el número de imágenes y modificar los parametros de entrenamiento.

Se podría hacer un estudio más pormenorizado teniendo en cuenta otras métricas, por ejemplo la sensibilidad o la especificidad.

Por otro lado, este tipo de modelos también se podrían usar en otro tipo de imágenes médica, por lo que se tendría un campo enorme en el que investigar: desde tomografías a ecografías, mamografías, resonancias y muchas más.

Por último, la aplicación se podría desarrollar más la aplicación para ser más completa y ofrecer más resultados.

8.- Glosario

CNN: red neuronal convolucional

Covid-19: Corona virus disease 2019

Transfer learning: Transferencia de aprendizaje

Fine tuning: Ajuste fino

Overfitting: sobre ajuste

9.- Bibliografía

- [1] "Como se propaga el covid" ,
<https://espanol.cdc.gov/coronavirus/2019-ncov/prevent-getting-sick/how-covid-spreads.html>
- [2] "Mayo Clinic. Radiografia" , <https://www.mayoclinic.org/es-es/tests-procedures/x-ray/about/pac-20395303>
- [3] "CoroDet: A deep learning based classification for COVID-19 detection using chest X-ray images".(Vol.142, num, 110495). Elsevier.
<https://www.sciencedirect.com/science/article/pii/S0960077920308870>
- [4] "Covid-19: automatic detection from X-ray images utilizing transfer learning with convolutional neural networks" ,
<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7118364/>
- [5] "COVID-Net: a tailored deep convolutional neural network design for detection of COVID-19 cases from chest X-ray images".
<https://www.nature.com/articles/s41598-020-76550-z>
- [6] "Sistema Auxiliar para el Diagnóstico de COVID-19 Mediante Análisis de Imágenes de CR Torácica Basado en Deep Learning". Memorias Del Congreso Nacional De Ingeniería Biomédica.
<http://memorias.somib.org.mx/index.php/memorias/article/view/810>
- [7] j"CovXNet: A multi- dilation convolutional neural network for automatic COVID-19 and other pneumonia detection from chest X-ray images with transferable multi- receptive feature optimization". Comput Biol Med
<https://pubmed.ncbi.nlm.nih.gov/32658740/>
- [8] "A deeplearning based multimodal system for Covid-19 diagnosis using breathing sounds and chest X-ray images".
<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8149173/>
- [9] "Deep Learning", Ian Goodfellow .
[http://alvarestech.com/temp/deep/Deep%20Learning%20by%20Ian%20Goodfellow,%20Yoshua%20Bengio,%20Aaron%20Courville%20\(z-lib.org\).pdf](http://alvarestech.com/temp/deep/Deep%20Learning%20by%20Ian%20Goodfellow,%20Yoshua%20Bengio,%20Aaron%20Courville%20(z-lib.org).pdf)
- [10] <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>
- [11] <https://cs231n.github.io/convolutional-networks/>
- [12] <https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/>
- [13] Base de datos: <https://www.kaggle.com/datasets/tawsifurrahman/covid19-radiography-database>
- [14] Machine Learning and Training Data: What You Need to Know
<https://labeledyourdata.com/articles/machine-learning-and-training-data>
- [15] Ways to Evaluate your Machine Learning Model: Cross-Validation Techniques
<https://www.analyticsvidhya.com/blog/2021/05/4-ways-to-evaluate-your-machine-learning-model-cross-validation-techniques-with-python-code/>

- [16] Selección de métricas de evaluación correcta: Métricas de clasificación.
<https://sitiobigdata.com/2019/01/19/machine-learning-metrica-clasificacion-parte-3/>
- [17] Arquitectura VGG16 y VGG19 en deep learning.
<https://keepcoding.io/blog/arquitectura-vgg16-vgg19-deep-learning/>
- [18] Data Augmentation y transfer learning.
<https://torres.ai/data-augmentation-y-transfer-learning-en-keras-tensorflow/>
- [19] The Effectiveness of Image Augmentation in Deep Learning Networks for Detecting COVID-19: A Geometric Transformation Perspective
<https://www.frontiersin.org/articles/10.3389/fmed.2021.629134/full>
- [20] Técnicas de Regularización Básicas para Redes Neuronales
<https://medium.com/metadatos/t%C3%A9cnicas-de-regularizaci%C3%B3n-b%C3%A1sicas-para-redes-neuronales-b48f396924d4>