Pontificia Universidad Javeriana Bogotá D.C

Técnicas de aprendizaje de máquina

Proyecto 1

Camilo Ortiz

Santiago Forero

Julián Páez

1 marzo de 2025

. Introducción

Las enfermedades pulmonares, como la neumonía y el COVID-19, representan un desafío significativo para los sistemas de salud debido a su potencial gravedad y alta tasa de contagio. La detección temprana es crucial para garantizar un tratamiento oportuno y reducir complicaciones, y una de las herramientas más utilizadas para este propósito es la radiografía de tórax. Sin embargo, la interpretación manual de estas imágenes puede ser un proceso subjetivo y propenso a errores, especialmente en situaciones de alta demanda o en regiones con escasez de especialistas.

En este contexto, la inteligencia artificial ha demostrado ser una herramienta prometedora para mejorar la precisión y la rapidez en el diagnóstico médico. Específicamente, el aprendizaje profundo y las redes neuronales convolucionales (CNN) han mostrado resultados notables en la clasificación de imágenes médicas. Este proyecto tiene como objetivo desarrollar un modelo basado en CNN para la clasificación de radiografías de tórax con el fin de detectar la presencia de neumonía o COVID-19.

Para lograrlo, se trabajará con bases de datos públicas disponibles en plataformas como Kaggle, específicamente el *Chest X-ray Images (Pneumonia)* y el *COVID-19 Radiography Database*. Se llevará a cabo un análisis exploratorio de datos, se aplicarán técnicas de preprocesamiento para mejorar la calidad de las imágenes y se diseñará una arquitectura de red neuronal optimizada para la clasificación de enfermedades pulmonares.

A lo largo del informe, se detallará cada una de las etapas del desarrollo del modelo, incluyendo la justificación de las decisiones tomadas en el preprocesamiento de datos, la arquitectura de la red, el proceso de entrenamiento y evaluación, así como un análisis de los resultados obtenidos. Finalmente, se discutirán los desafíos enfrentados y se propondrán posibles mejoras para futuros desarrollos en este campo.

1. Análisis exploratorio de datos

```
import os
import shutil
import cv2
import numpy as np
import matplotlib.pyplot as plt
import keras
from random import randint
from keras.layers import Dense
from keras.models import Sequential
from keras.layers import Conv2D,MaxPooling2D,Flatten,Dropout
```

Para comenzar, se hizo la importación de estas librerías las cuales ayudan principalmente al manejo de archivos y directorios, cv2 para el cargado y procesamiento de imágenes, también librerías para el manejo de arreglos numéricos, carga de imágenes, Deep learning, Números aleatorios y librerías como keras.layers para las capas de la red neuronal.

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("tawsifurrahman/covid19-radiography-database")

print("Path to dataset files:", path)
```

Después, KaggleHub se encarga de descargar el conjunto de datos COVID-19 RADIOGRAPHY DATABASE y descargar esto como base de datos para el manejo normal de los datos, la variable path almacena la ruta del dataset descargado.

```
direccion = "/root/.cache/kagglehub/datasets/tawsifurrahman/covid19-radiography-database/versions/5"
for folder in os.listdir(direccion):
    subfolder_path = os.path.join(direccion, folder)
    if os.path.isdir(subfolder_path):
        for sub_folder in os.listdir(subfolder_path):
            sub_subfolder_path = os.path.join(subfolder_path, sub_folder)
            print(sub_subfolder_path)
```

Posteriormente se hace un listado del contenido que el dataset tiene, es decir, se recorren todas las carpetas y subcarpetas para imprimir sus rutas.

```
for folder in os.listdir(direccion):
  subfolder_path = os.path.join(direccion,folder)
  if os.path.isdir(subfolder_path):
    for subfolder in os.listdir(subfolder_path):
      sub_subfolder_path = os.path.join(subfolder_path,subfolder)
      if os.path.isdir(sub_subfolder_path) and ("Normal" not in subfolder and "COVID" not in subfolder):
          # removemos los directorios que hacen referencia a la metadata
          shutil.rmtree(sub_subfolder_path)
      elif ".xlsx" in subfolder:
        # remover los xlsx
        os.remove(sub_subfolder_path)
      elif ".txt" in subfolder:
        os.remove(sub_subfolder_path)
```

En este extracto se hace limpieza del contenido del dataset. Se eliminan las carpetas y los archivos que no sean imágenes de “Normal” y “Covid” eliminando así carpetas completas y archivos .xlsx y .txt innecesarios.

```
for folder in os.listdir(direccion):
    subfolder_path = os.path.join(direccion, folder)
    if os.path.isdir(subfolder_path):
        print(f"\nContenido de '{folder}':", os.listdir(subfolder_path))
```

Se hace la verificación de cuáles dataset quedaron activos.

```
clases = ['Normal', 'COVID']
lista = []
lista_labels = []
for clase in clases:
  direccion_clase = os.path.join("/root/.cache/kagglehub/datasets/tawsifurrahman/covid19-radiography-database/versions/5/COVID-19_Radiography_Dataset
  for subdir in os.listdir(direccion_clase):
    subdir_direccion = os.path.join(direccion_clase,subdir)
    imagen_numpy = cv2.imread(subdir_direccion,cv2.IMREAD_GRAYSCALE)
    imagen_resize = cv2.resize(imagen_numpy,(200,200))
    lista_labels.append(1) if clase == "COVID" else lista_labels.append(0)
    lista.append(imagen_resize)
```

Posteriormente se crea una lista con las dos categorías de imágenes que son las de “Covid” y “Normal” y las almacena para etiquetarlas como 0 para normal y 1 para covid. Después se construye la ruta donde está cada imagen, almacena la ruta de imagen, carga la imagen y la convierte a escala de grises para simplificar el pre procesamiento y reducir la dimensionalidad a un tamaño de 200x200 pixeles y les asigna las etiquetas a las imágenes para agregarla a una lista ya redimensionada.

```
x = np.array(lista)
y = np.array(lista_labels)
```

Ya aquí, x contiene todas las imágenes procesadas y/o y contiene todas las etiquetas correspondientes (0 o 1).

```
unique_shapes = set(img.shape for img in x)
print("Dimensiones únicas de las imágenes:", unique_shapes) # se seteo las dimensiones de la imagen anteriormente en 200 aca solo verificamos que este bien
```

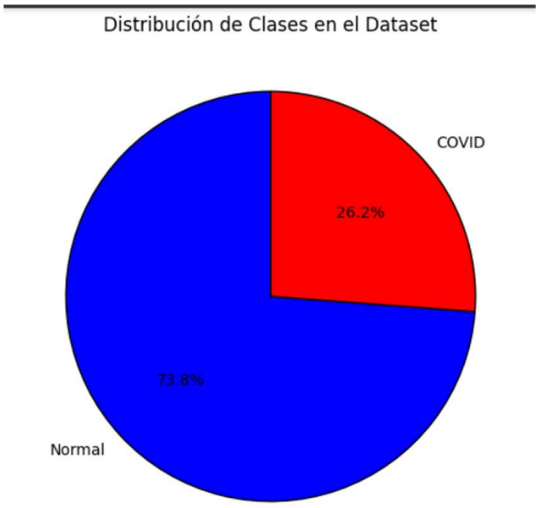
Aquí se verifica que todas las imágenes tengan la misma dimensionalidad ya que es vital para entrenar una red neuronal sin errores, con suerte se pudo determinar que todas las imágenes tenían la misma dimensionalidad (200 x 200).

```
[ ]  import numpy as np

     # Buscar imágenes con valores anómalos (pixeles fuera del rango 0-255)
     imagenes_corruptas = [i for i in range(len(x)) if x[i].min() < 0 or x[i].max() > 255]

     # Ver si encontramos imágenes corruptas
     if imagenes_corruptas:
         print(f"Se encontraron {len(imagenes_corruptas)} imágenes corruptas.")
     else:
         print("No hay imágenes corruptas en el dataset.")
```

En esta parte, se quiso detectar si las imágenes eran corruptas, es decir, si los pixeles tenían un rasgo fuera del esperado ([0, 255]) lo cual podría indicar que la imagen está dañada. Con suerte se pudo determinar que ninguna imagen estaba corrupta.



Nuestro siguiente paso fue hacer un gráfico pastel para determinar la distribución de las clases que pertenecen a “Normal” vs “Covid” esto para ver si se reajuste el dataset para que no haya errores

como overfitting. Viendo el gráfico podemos determinar que hay una gran diferencia entre los datos de Normal y Covid por lo que el dataset está desbalanceado.

2. Pre procesamiento de imágenes

```
indices_normal = np.where(y == 0)[0]  # Índices de imágenes "Normal"
indices_covid = np.where(y == 1)[0]   # Índices de imágenes "COVID"

# Seleccionar 6,000 imágenes aleatorias de la clase Normal
indices_normal_subsampled = np.random.choice(indices_normal, 6000, replace=False)

# Unir con todos los datos de COVID
indices_finales = np.concatenate([indices_normal_subsampled, indices_covid])

# Barajar
np.random.shuffle(indices_finales)

# Crear nuevo dataset balanceado parcialmente
x_balanced = x[indices_finales]
y_balanced = y[indices_finales]

print(f"Nuevo dataset tras submuestreo - Normal: {np.sum(y_balanced == 0)}, COVID: {np.sum(y_balanced == 1)}")
```

```
Nuevo dataset tras submuestreo - Normal: 6000, COVID: 3616
```

Para este pre procesamiento, primero quisimos arreglar ese desbalance que había entre datos para lograr un modelo más eficiente. Por esto mismo, se combinan las clases y se mezclan aleatoriamente para reducir la cantidad de imágenes normales con objetivo a equilibrar el dataset y aunque haya más imágenes en “Normal” que en “Covid” no hay un desbalance tan grande como el que había antes.

```
import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Aplicar Data Augmentation solo a imágenes COVID
datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.05,
    zoom_range=0.1,
    brightness_range=[0.8, 1.2]
)

# Extraer imágenes de COVID
covid_images = x_balanced[y_balanced == 1]
num_extra = 6000 - len(covid_images)  # Cuántas imágenes extra necesitamos

x_augmented = []
y_augmented = []

for i in range(num_extra):
    img = covid_images[i % len(covid_images)]  # Seleccionar imagen de COVID (repite si es necesario)
    img = img.reshape((1,) + img.shape + (1,))  # Agregar dimensión extra
    aug_iter = datagen.flow(img, batch_size=1)  # Aplicar Data Augmentation
    x_augmented.append(next(aug_iter).reshape(img.shape[1:3]))  # ✅ USAMOS next()
    y_augmented.append(1)  # Etiqueta COVID
```

```
# Convertir listas en arrays de NumPy
x_augmented = np.array(x_augmented)
y_augmented = np.array(y_augmented)

# Unir datos originales con los aumentados
x_balanced = np.concatenate([x_balanced, x_augmented], axis=0)
y_balanced = np.concatenate([y_balanced, y_augmented], axis=0)

# Mezclar nuevamente
indices = np.random.permutation(len(x_balanced))
x_balanced = x_balanced[indices]
y_balanced = y_balanced[indices]

print(f"Dataset final balanceado - Normal: {np.sum(y_balanced == 0)}, COVID: {np.sum(y_balanced == 1)}")
```

```
Dataset final balanceado - Normal: 6000, COVID: 6000
```

Esta parte es fundamental para el proceso anterior. Acá se usa “Data Augmentation” a imágenes Covid y se definen transformaciones aleatorias como rotaciones, desplazamientos y cambios de brillo, esto para generar imágenes nuevas y así poder equilibrar el dataset. En conclusión, se agregan imágenes transformadas con x_augmented con etiquetas y_augmented = 1 (COVID) para lograr el equilibrio entre dataset como se puede observar que cada uno tiene 6000 ahora.

```
#Necesitamos aleatorizar el orden del conjunto de datos para despues dividir en test y train
indices = np.random.permutation(len(x_balanced))

# Randomizamos el array
x_shuffled = x_balanced[indices]
y_shuffled = y_balanced[indices]
```

Después se barajan los datos de forma aleatoria para impedir que las imágenes de una clase queden agrupadas y evitar algún sesgo en el entrenamiento.

```
x_shuffled.shape
```

```
(12000, 200, 200)
```

```
y_shuffled.size
```

```
12000
```

Por último, se evalúa la longitud total del dataset.

```
# División 80% - 20%
#Justificar en el informe porque se hizo esta division
split_idx = int(0.8 * len(x_shuffled))

#Entrenamiento
x_train, y_train = x_shuffled[:split_idx], y_shuffled[:split_idx]

#Prueba
x_test, y_test = x_shuffled[split_idx:], y_shuffled[split_idx:]

#Tamaños de los conjuntos
print("Tamaño de x_train:", x_train.shape)
print("Tamaño de y_train:", y_train.shape)
print("Tamaño de x_test:", x_test.shape)
print("Tamaño de y_test:", y_test.shape)
```

```
Tamaño de x_train: (9600, 200, 200)
Tamaño de y_train: (9600,)
Tamaño de x_test: (2400, 200, 200)
Tamaño de y_test: (2400,)
```

Posteriormente se divide el dataset para entrenarlo, en este caso se decidió dividir el dataset en una proporción de 80 a 20, es decir, 80% para datos de entrenamiento y 20% de datos de testing. Esta división se hizo a partir del análisis de imágenes para el modelo.

```
num_classes = 2
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

Después de la división, se hizo un One-Hot Encoding en y_train y/o y_test para codificar las etiquetas de esta manera: Normal (0) -> [1,0] y Covid (1) -> [0,1] esto requerido por las redes neuronales para interpretar las clases correctamente.

```
#normalizamos las imagenes del dataset
x_train = x_train / 255.0
x_test = x_test / 255.0
```

Esto para convertir los valores en pixeles [0,255] a [0,1] lo que ayuda a mejorar la convergencia en el modelo.

```
[ ]  x_train = x_train.reshape(-1, 200, 200, 1)
     x_test = x_test.reshape(-1,200,200, 1)
```

Por último, se agrega una dimensión extra (1) para indicar que son imágenes de escala de grises, necesario para que el modelo funcione correctamente.

3. Arquitectura de red neuronal

Antes de comenzar con este punto, queremos decir que la arquitectura de la red neuronal se basó con esta red neuronal de referencia encontrada en el siguiente link: https://www.kaggle.com/code/kanncaa1/convolutional-neural-network-cnn-tutorial la cual era para números, pero se utilizó esa cantidad de capas para no hacer un modelo tan computacionalmente costoso, además de esta red tener regularización lo cual era un pro para usarlo.

También cabe recalcar que como la clase minoritaria era covid, usamos como métricas accuracy y precisión para ver como el modelo iba frente a una predicción de esta clase la cuál es la de mayor interés.

Teniendo en cuenta esta información, el código de este punto es de la siguiente manera:

```
model = Sequential()
```

Primeramente, se agrega un modelo secuencial para que las capas se agreguen unas tras otras.

```
model = Sequential()
#Capa convolusional 1
model.add(Conv2D(filters = 8, kernel_size = (5,5), activation ='relu', input_shape = (200,200,1)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
# capa convolusional 2
model.add(Conv2D(filters = 16, kernel_size = (5,5), activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# capa convolusional
model.add(Conv2D(filters = 32, kernel_size = (5,5), activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
```

Después se agregaron las capas necesarias a la red neuronal para la extracción suficiente de las características importantes de las imágenes utilizando maxplooling y dropout teniendo una salida de esta manera (21,21,32). También es importante recalcar que se utilizó un kernel de 5 que fue el número más pequeño con funcionalidad que pudimos encontrar.

```
# fully connected
model.add(Flatten())
model.add(Dense(200, activation = "relu"))
model.add(Dropout(0.4))
model.add(Dense(2, activation = "sigmoid"))
```

Posteriormente se hace una conectividad entre capas y convierte una estructura de 3D en una de 1D con toda la capa conectada con 200 neuronas y se apaga el 40% de las neuronas para prevenir el overfitting para terminar con la última capa de 2 neuronas usando una sigmoide, ya que es una clasificación binaria

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 196, 196, 8)	208
max_pooling2d (MaxPooling2D)	(None, 98, 98, 8)	0
dropout (Dropout)	(None, 98, 98, 8)	0
conv2d_1 (Conv2D)	(None, 94, 94, 16)	3,216
max_pooling2d_1 (MaxPooling2D)	(None, 47, 47, 16)	0
dropout_1 (Dropout)	(None, 47, 47, 16)	0
conv2d_2 (Conv2D)	(None, 43, 43, 32)	12,832
max_pooling2d_2 (MaxPooling2D)	(None, 21, 21, 32)	0
dropout_2 (Dropout)	(None, 21, 21, 32)	0
flatten (Flatten)	(None, 14112)	0
dense (Dense)	(None, 200)	2,822,600
dropout_3 (Dropout)	(None, 200)	0
dense_1 (Dense)	(None, 2)	402

Total params: 2,839,258 (10.83 MB)
Trainable params: 2,839,258 (10.83 MB)
Non-trainable params: 0 (0.00 B)

Luego se realiza un resumen del modelo dando como resultado un estimado de 2,839,258 datos entrenables.

4. Entrenamiento

```
optimizer = keras.optimizers.SGD(learning_rate=0.01)
```

```
model.compile(optimizer="sgd", loss='categorical_crossentropy', metrics=['accuracy',metrics.Precision(name='precision')])
```

Se usa una tasa de aprendizaje del 0.01 y se van a monitorear dos métricas: accuracy y precisión.

```
history = model.fit(x_train, y_train, batch_size=8, epochs=12, verbose=2, validation_split=0.1)
```

```
Epoch 1/12
1080/1080 - 16s - 15ms/step - accuracy: 0.7009 - loss: 0.5725 - precision: 0.6630 - val_accuracy: 0.7594 - val_loss: 0.5964 - val_precision: 0.7583
Epoch 2/12
1080/1080 - 7s - 6ms/step - accuracy: 0.7753 - loss: 0.4737 - precision: 0.7401 - val_accuracy: 0.7865 - val_loss: 0.4658 - val_precision: 0.7423
Epoch 3/12
1080/1080 - 6s - 6ms/step - accuracy: 0.8108 - loss: 0.4127 - precision: 0.7701 - val_accuracy: 0.8000 - val_loss: 0.4746 - val_precision: 0.8177
Epoch 4/12
1080/1080 - 11s - 10ms/step - accuracy: 0.8468 - loss: 0.3577 - precision: 0.8136 - val_accuracy: 0.8719 - val_loss: 0.3133 - val_precision: 0.8493
Epoch 5/12
1080/1080 - 7s - 6ms/step - accuracy: 0.8654 - loss: 0.3177 - precision: 0.8502 - val_accuracy: 0.8687 - val_loss: 0.3635 - val_precision: 0.8626
Epoch 6/12
1080/1080 - 7s - 6ms/step - accuracy: 0.8791 - loss: 0.2883 - precision: 0.8761 - val_accuracy: 0.8646 - val_loss: 0.3093 - val_precision: 0.8612
Epoch 7/12
1080/1080 - 6s - 6ms/step - accuracy: 0.8909 - loss: 0.2643 - precision: 0.8596 - val_accuracy: 0.8885 - val_loss: 0.2757 - val_precision: 0.8753
Epoch 8/12
1080/1080 - 11s - 10ms/step - accuracy: 0.8970 - loss: 0.2478 - precision: 0.8776 - val_accuracy: 0.9073 - val_loss: 0.2588 - val_precision: 0.8849
Epoch 9/12
1080/1080 - 7s - 6ms/step - accuracy: 0.9064 - loss: 0.2259 - precision: 0.8813 - val_accuracy: 0.9219 - val_loss: 0.2211 - val_precision: 0.9111
Epoch 10/12
1080/1080 - 10s - 9ms/step - accuracy: 0.9130 - loss: 0.2088 - precision: 0.8993 - val_accuracy: 0.9083 - val_loss: 0.2342 - val_precision: 0.8909
Epoch 11/12
1080/1080 - 10s - 9ms/step - accuracy: 0.9201 - loss: 0.1973 - precision: 0.8949 - val_accuracy: 0.9177 - val_loss: 0.2179 - val_precision: 0.8953
Epoch 12/12
1080/1080 - 10s - 9ms/step - accuracy: 0.9278 - loss: 0.1769 - precision: 0.9146 - val_accuracy: 0.9333 - val_loss: 0.1843 - val_precision: 0.9061
```

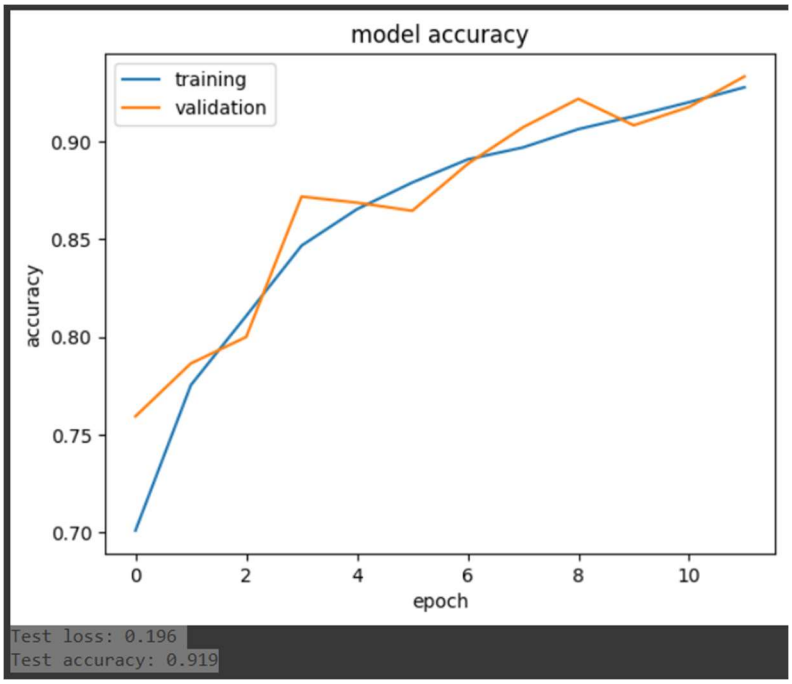
El paso siguiente es pasar directamente al entrenamiento del modelo dando como conclusión que la precisión mejora con cada época, alcanzando un 92.7% en entrenamiento y 93.3% en validación lo que indica que el modelo se generaliza de manera correcta.

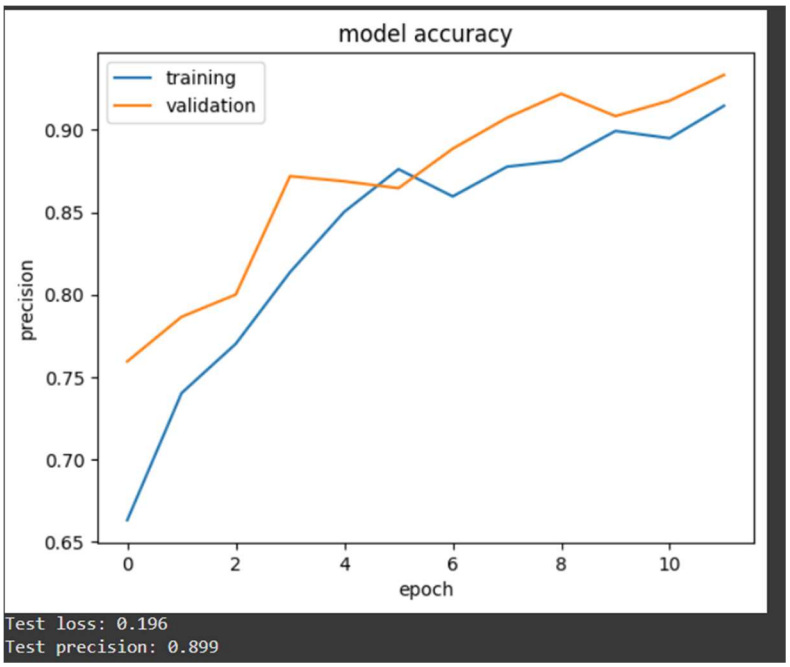
```
loss, accuracy,precision  = model.evaluate(x_test, y_test, verbose=1)
```

```
75/75 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - accuracy: 0.9155 - loss: 0.2068 - precision: 0.8901
```

Aquí probamos el modelo con el conjunto de test dando como resultados en precisión del test 91.5%, una precisión como métrica del test de 89.0% y una pérdida del 89.01% dando como conclusión un buen desarrollo del modelo con datos nuevos.

Se visualizan los resultados:



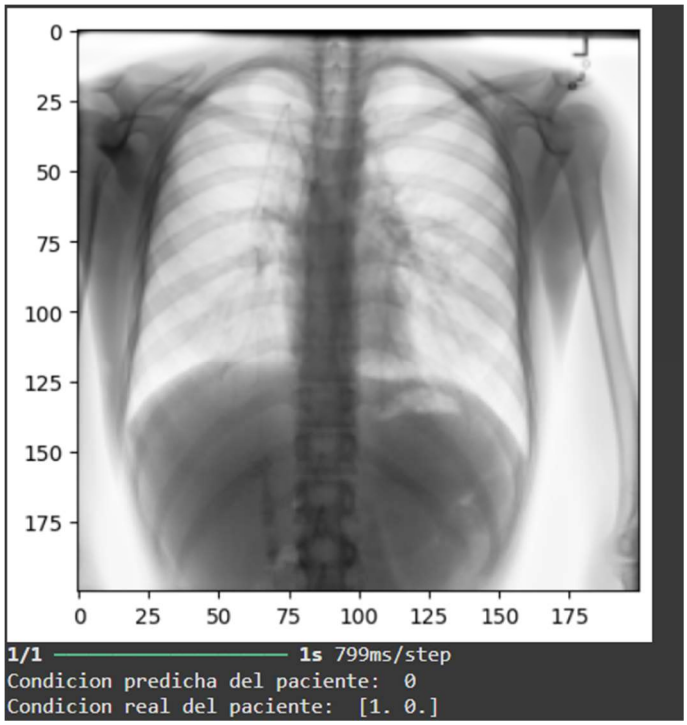


- Para el primer gráfico podemos interpretar que la curva azul (entrenamiento) y la curva naranja (testing) son similares, lo que indica que el modelo no está sobre ajustado.
- En el segundo gráfico se afirma lo que se dijo anteriormente, que la precisión aumenta con las épocas y que el test se mantiene estable en torno al 90% lo cual es una buena señal.

5. Pruebas y predicciones del modelo

```
test_img = randint(0,x_test_copia.shape[0])
X_test_img = x_test_copia[test_img]
plt.imshow(X_test_img, cmap='Greys')
plt.show()
X_test_img = X_test_img.reshape(1, 200, 200, 1)
prediction = model.predict(X_test_img)
print('Condicion predicha del paciente: ', prediction.argmax())
print('Condicion real del paciente: ', y_test[test_img])
```

Por último, se genera una imagen aleatoria para ser evaluada y se le aplican todos los cambios posibles para que sea compatible con el modelo dando el siguiente resultado:



Aquí se puede determinar que la predicción fue hecha de manera correcta, ya que la condición predicha fue la misma que lanzó el modelo.

. Conclusiones

A partir de este análisis, se puede concluir que el modelo desarrollado fue construido de manera exitosa, reflejándose en la arquitectura de la red neuronal utilizada. Aunque se emplearon pocas capas en la red, se logró un equilibrio entre eficiencia y rendimiento. Si se buscara una mayor precisión, podría considerarse la incorporación de más capas y la optimización del kernel. Sin embargo, con la configuración actual, el modelo ha demostrado ser eficiente en la predicción de enfermedades a partir de imágenes de rayos X.