Informe Técnico: Clasificación de Imágenes Médicas con Redes Neuronales Convolucionales

1. Objetivo del Proyecto

El objetivo principal de este proyecto es desarrollar un modelo de red neuronal convolucional (CNN) capaz de clasificar imágenes médicas, específicamente radiografías de tórax, para identificar casos de neumonía. Las CNN permiten extraer y aprender patrones visuales relevantes en imágenes, lo cual es fundamental para apoyar diagnósticos médicos automatizados.

dataset: https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia

2. Recolección y Organización del Conjunto de Datos

Se utilizan tres subconjuntos: entrenamiento, validación y prueba. Las imágenes se encuentran organizadas en carpetas correspondientes a cada clase (NORMAL y PNEUMONIA), con las siguientes cantidades:

Conjunto de entrenamiento:
NORMAL: 1072 imágenes
PNEUMONIA: 3100 imágenes

Conjunto de validación:
NORMAL: 277 imágenes
PNEUMONIA: 783 imágenes

Conjunto de prueba:
NORMAL: 234 imágenes
PNEUMONIA: 390 imágenes

3. Verificación y Limpieza de Datos

Antes de entrenar el modelo, se verifica la integridad y tamaño de las imágenes,aunque los tamaños sean anómalos se ajustarán para el entrenamiento por el anterior bloque img_width, img_height = 224, 224 y target size.

Imágenes corruptas: 0 en todas las carpetas

Imágenes con tamaños atípicos:

Entrenamiento: 4172

Validación: 1060

Prueba: 624

4. Preprocesamiento de Imágenes

Las imágenes se procesan mediante generadores que realizan las siguientes tareas:

Normalización: Los valores de los píxeles se escalan al rango [0, 1] dividiendo por 255 (rescale=1./255).

Aumento de datos (solo en entrenamiento): Se aplican transformaciones aleatorias para mejorar la generalización:

Rotación (hasta 20 grados)

Desplazamiento horizontal y vertical (hasta 20%)

Zoom

Cizallamiento

Volteo horizontal

Estos pasos se aplican mediante ImageDataGenerator.

5. Generadores de Datos

Se crean generadores para los tres subconjuntos:
train_generator = train_datagen.flow_from_directory(...)
val_generator = val_datagen.flow_from_directory(...)
test_generator = test_datagen.flow_from_directory(...)

Estos generadores permiten cargar las imágenes en bloques (batch) y aplicar pre procesamientos en tiempo real.

6. Definición de la Arquitectura del Modelo CNN
La red neuronal tiene la siguiente estructura secuencial:
Cuántas capas tiene el modelo CNN?

1. Conv2D (32 filtros) → extracción de características
2. MaxPooling2D → reducción de dimensionalidad
3. BatchNormalization → estabiliza el entrenamiento

4. Conv2D (64 filtros) → más abstracción de características
5. MaxPooling2D
6. BatchNormalization

7. Conv2D (128 filtros) → aún más abstracción
8. MaxPooling2D
9. BatchNormalization

10. Flatten → aplana los tensores 3D a 1D
11. Dropout → reduce sobreajuste
12. Dense (128 neuronas, ReLU) → capa oculta tradicional
13. Dense (1 neurona, Sigmoid) → salida para clasificación binaria

🔢 Capas totales: 13 (todas las que definiste en Sequential)
Última capa (output): Dense(1, activation='sigmoid') → adecuada para clasificación binaria

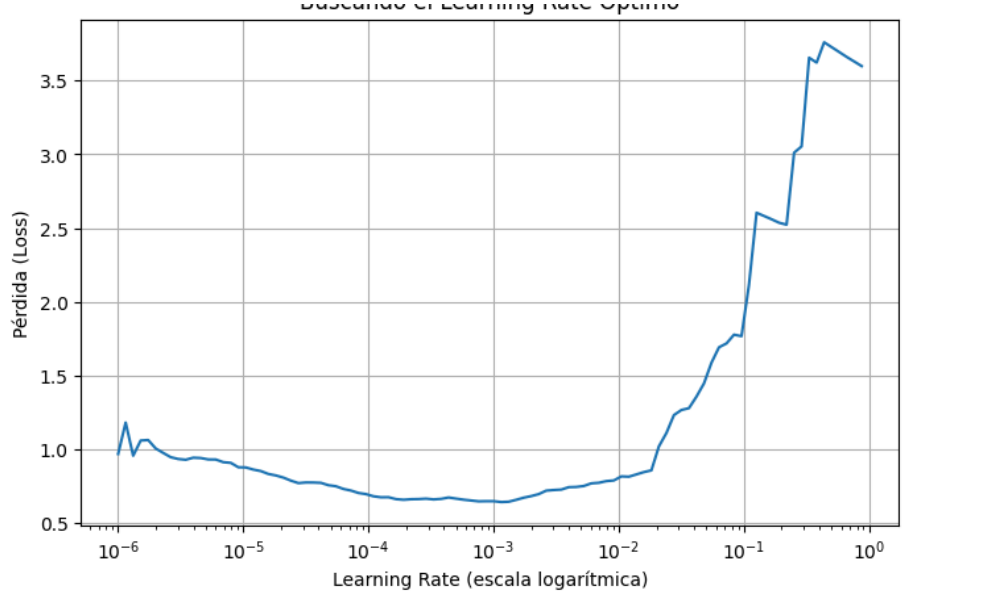
7. Compilación Inicial del Modelo
Antes del entrenamiento, se compila el modelo con el optimizador Adam, la función de pérdida binary_crossentropy y la métrica de precisión.

8. Búsqueda del Learning Rate Óptimo
Se implementa una clase LearningRateFinder personalizada para probar múltiples tasas de aprendizaje (lr) en lotes sucesivos. Este procedimiento ayuda a identificar el valor óptimo de lr al observar cuándo la pérdida (loss) disminuye rápidamente sin volverse inestable.

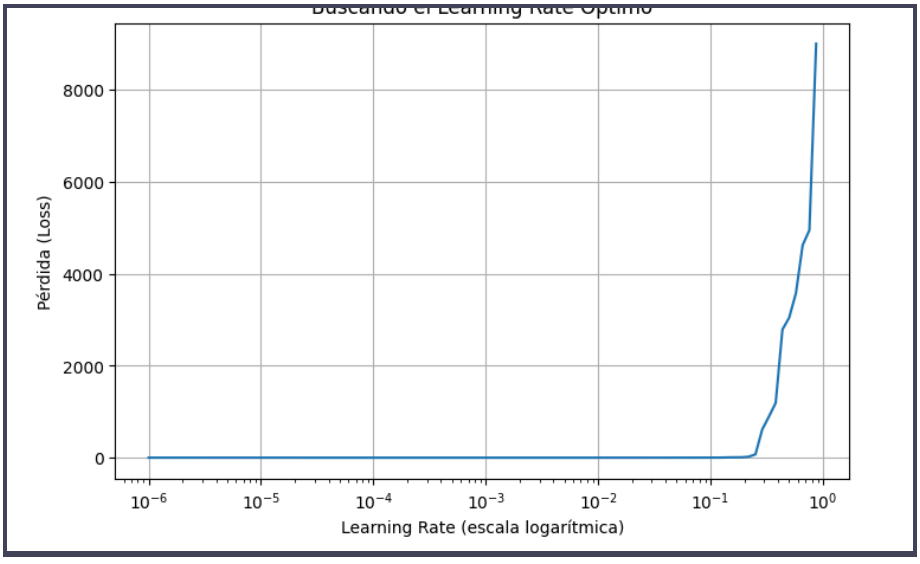
Luego del análisis gráfico, se selecciona un valor de learning_rate óptimo.

9. Recompilación con el Learning Rate Óptimo
Se recompila el modelo con el valor óptimo encontrado se probo 3 veces con distintos puntos óptimos:

1. Antes de la normalización el lr, se probó con el punto óptimo ≈ 10^-3 (o sea, 0.001) pero Ya en los **primeros 11 batches de la época 2**, el loss **bajó mucho** de 1.24 (época 1) a **0.58**, lo cual es una gran mejora temprana. Pero también ves que el accuracy se mantiene cerca de 0.7386, casi igual que antes sin cambiar asique se optó por normalizar. se usó 20 epochs

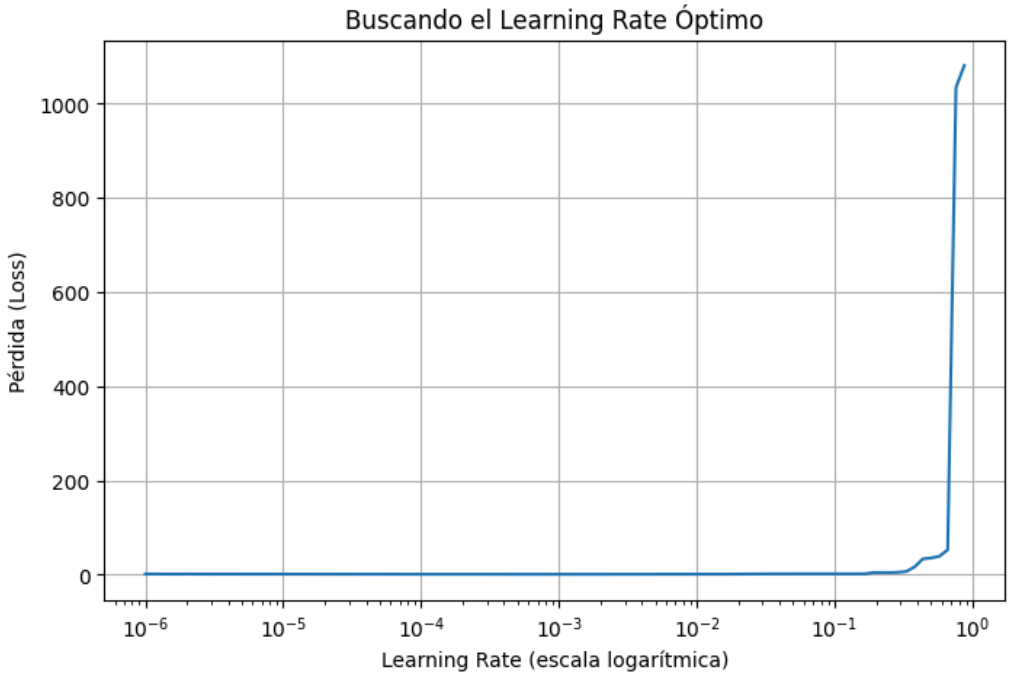


2. luego de la normalización el lr. se probó conel punto óptimo de 0.03
El resultado fue de: **accuracy: 0.8794**: El modelo acierta el 87.94% de las veces en el conjunto evaluado. En principio parece bueno pero el **loss: 488.9123**: Este valor es **demasiado grande**. seuso 20 epochs



3. A partir de acá nos dimos cuenta de un error en la categoría, que debía ser binaria por lo que se editó varias partes del código para volver a ejecutarlas.

El punto óptimo ocurre 0.003 Es un valor más conservador, todavía en la zona segura y con menos riesgo de inestabilidad y prevenir overfiting o saltos bruscos.
se uso 15 epochs



14. Conclusiones

🔍 Métricas clave:

precision De todas las veces que predijo una clase, ¿cuántas veces acertó?

recall De todas las veces que realmente ocurrió una clase, ¿cuántas veces la detectó el modelo?

f1-score Promedio entre precisión y recall.

support Cuántas muestras reales hay de cada clase.

🟢 Para la clase NORMAL:

precision = 0.26: solo el 26% de las veces que predijo "NORMAL" fue correcto.

recall = 1.00: el modelo detectó todas las imágenes normales correctamente.

f1-score = 0.41: un resultado bajo, porque la precisión fue mala aunque el recall fue perfecto.

support = 277: hay 277 imágenes normales reales.

🔴 Para la clase PNEUMONIA:
precision = 0.00: ninguna predicción de neumonía fue correcta.
recall = 0.00: el modelo nunca detectó ninguna de las 783 imágenes de neumonía.
f1-score = 0.00: resultado desastroso.
support = 783: había 783 casos reales de neumonía.

📉 Métricas globales:
accuracy = 0.26: el modelo acertó solo el 26% de las veces.
macro avg: promedio simple entre ambas clases (sin pesar por cantidad de casos).
precision = 0.13, recall = 0.50, f1 = 0.21
weighted avg: promedio ponderado por cantidad de casos.
precision = 0.07, recall = 0.26, f1 = 0.11

🧠 ¿Qué significa todo esto?
EL modelo está prediciendo siempre "NORMAL", por eso el recall de "NORMAL" es 1.00 y el de "PNEUMONIA" es 0.00.
Esto puede deberse a un desequilibrio en los datos o a que el modelo no aprendió a diferenciar las clases.

Pasos a seguir a partir de los resultados negativos:

1. Desbalances de pesos:

Durante la evaluación del modelo se observó que **siempre predecía la clase "NORMAL"**, con un **recall de 1.00 para "NORMAL"** y **0.00 para "PNEUMONIA"**. Esto indicaba que el modelo no lograba distinguir ambas clases, posiblemente por **sobreajuste o desbalance**.

Al revisar los datos de entrenamiento, se vio que había **más imágenes de PNEUMONIA (3100)** que de **NORMAL (1070)**. Sin embargo, el modelo se sesgaba hacia "NORMAL", lo cual fue una **contradicción aparente**.

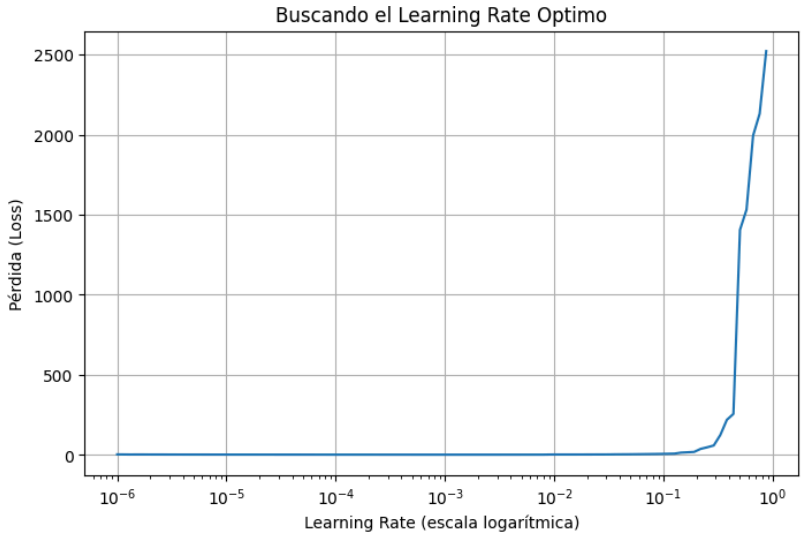
Para corregirlo, se usó el parámetro `class_weight` al entrenar el modelo, **dando más peso a la clase con menos ejemplos ("NORMAL")**:

2. Aumento de datos (data augmentation):
Esta técnica ya se encontraba implementada en el flujo de entrenamiento. Consiste en generar versiones ligeramente modificadas de las imágenes originales (rotaciones, desplazamientos, zoom, etc.) para aumentar la diversidad del conjunto de entrenamiento y prevenir que el modelo memorice patrones fijos. Dado que esta estrategia ya estaba activa, se descartó como causa principal del problema.

Regularización:
El modelo también incluía técnicas de regularización como Dropout y BatchNormalization, las cuales ayudan a reducir el sobreajuste. Estas capas estaban correctamente ubicadas en la arquitectura de la red. Como refuerzo adicional, se propuso sumar una penalización L2 (regularización de tipo Ridge) en la capa densa final. Esta técnica añade una restricción sobre los pesos del modelo, evitando que se ajusten de manera excesiva a los datos de entrenamiento. Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.001))

Estas medidas se fueron aplicando de manera progresiva para mejorar la capacidad del modelo de generalizar y reconocer correctamente ambas clases, especialmente la clase "PNEUMONIA", que es de particular interés en este problema médico.

Luego de ejecutar de nuevo el codigo pasamos de nuevo al LR:



👉 E**l LR óptimo recomendado sería entre 0.001 y 0.01**. Un valor común y seguro sería: 0.005

Evaluación del modelo luego de las modificaciones

fue de exactamente 50 % acertando en la mitad del conjunto de los casos.
De nuevo no identificar correctamente ninguna de las imágenes que realmente tenían neumonia.