Universidad Nacional Autónoma de México
Facultad de Ciencias

Medición de la pérdida de información en imágenes con Longitud de Huffman y Entropía

Diego Arias Cabrera

Índice general

Capítulo 1

Introducción

El creciente uso de las redes neuronales convolucionales (CNN) ayuda a que las tareas de clasificación de imágenes se hagan automáticamente y en mucho menor tiempo. Esto genera un aumento en productividad en muchos casos; sin embargo, para poder entrenar una CNN como CheXNet (CNN de 121 capas entrenada para la detección de neumonía a partir de imágenes de Rayos X [5]) se necesita una cantidad de poder computacional muy grande lo que puede dificultar la producción de estas en ambientes que no cumplen esta condición.

Principalmente, los lugares que no cumplen con el poder computacional antes mencionado son aquellos donde los recursos son muy limitados. Para poder dar una herramienta similar a lo que ofrece ChesXNet se utilizará una CNN que cuenta con solamente 12 capas. Si bien es una CNN muy pequeña a comparación de ChesXNet y tiene una precisión del $90.38\,\%$, tiene un problema y es el tiempo de cómputo ya que, aunque no sean tantas capas, la CNN recibe imágenes de 256×256 pixeles lo que ralentiza el entrenamiento de la misma.

Debido a este problema, lo que busca este proyecto es minimizar el tamaño de las imágenes aplicando teoría de códigos para calcular la información que se pierde al reducir el tamaño de las imágenes, y que los resultados no pierdan precisión. Posteriormente, se analizan estos datos y se entrena nuevamente la CNN con esta nueva resolución sin cambiar ninguna de las 12 capas.

Capítulo 2

Marco teórico

2.1. Red Neuronal Convolucional

Una red neuronal convolucional (CNN por sus siglas en inglés) es una arquitectura de red para aprendizaje profundo que se encuentra en la parte de aprendizaje supervisado, es decir, se tienen que tener los datos clasificados antes de proceder con el entrenamiento. El principal enfoque de estas redes es identificar patrones en imágenes para reconocer objetos, clases y categorías; no obstante, también se utilizan para clasificar datos de audio, señales y series temporales. [4]

Un pixel es la componente más pequeña posible ya que las imágenes digitales salen a partir de un conjunto de estos. Cualquier imagen que se ve en una pantalla digital, está construida con bases en los pixeles [1]. Estos pueden tener colores, pueden ser blanco y negro, o pueden estar en escala de grises [3].

Un kernel es una matriz de números que se utiliza para enfocar, desenfocar, grabar, detectar bordes y características de una imagen [3].

Una imagen digital en escala de grises es una matriz donde cada uno de los pixeles está representado con valores que van desde 0 (negro) hasta 255 (blanco). Se le llama normalización de datos al proceso de dividir el valor de los pixeles entre 255 lo que hace que se obtengan valores entre 0 y 1 para disminuir la carga computacional y que la red neuronal se entrene más rápido.

Las CNN cuentan con diferentes capas que cumplen diferentes funciones. Estas capas son las siguientes:

- Capa de convolución: Lo que la convolución hace es que, a través de un filtro (kernel), extrae las características o los patrones de una imagen.
- Capa de reducción: Esta capa se encarga de reducir el flujo de datos de la capa de convolución mediante la extracción de la información o las características más importantes de la imagen que ya pasó por la capa convolucional. Al reducir el flujo

de datos, también reduce la carga computacional que va a tener la CNN ayudando a que la red se entrene y segmente de una mejor manera cuáles son las características más importantes.

- Capa densa: Esta capa se encarga de asignar una puntuación a cada clase posible, utilizando las características que fueron extraídas por las capas de convolución y de reducción (max pooling). Lo hace combinando toda esa información para que la red pueda tomar una decisión final y clasificar correctamente la imagen. [3].

2.2. Teoría de códigos

En el contexto de la teoría de códigos, una fuente de información se modela como una sucesión de símbolos extraídos de un conjunto finito denominado alfabeto fuente. A cada uno de estos símbolos se le asigna un código, es decir, una sucesión finita de símbolos tomada de otro conjunto llamado alfabeto de códigos, usualmente binario (como $\{0, 1\}$). Esta asignación se realiza mediante una función llamada código fuente, cuya salida son las llamadas palabras código [2].

Dado que los símbolos de la fuente $s_1, s_2, \ldots, s_q$ pueden tener diferentes probabilidades de aparición, se define la *longitud promedio* de un código como una medida de eficiencia del mismo. Esta longitud corresponde al promedio ponderado de la longitud de las palabras código según la distribución de probabilidades de los símbolos, es decir:

$$L(C) = \sum_{i=1}^{q} p_i \cdot \ell_i$$

donde p_i es la probabilidad del símbolo s_i, y ℓ_i es la longitud de la palabra código correspondiente. Minimizar $L(C)$ implica diseñar un sistema de codificación más eficiente, ya que reduce el número promedio de bits necesarios para representar un símbolo [2].

Por otro lado, la entropía es una noción fundamental que cuantifica la cantidad media de información contenida en una fuente. Se define como:

$$H(P) = -\sum_{i=1}^{q} p_i \log_2 p_i$$

y representa el límite inferior teórico de la longitud promedio que cualquier codificación sin pérdida podría alcanzar. Esta función, además de capturar la incertidumbre asociada a una fuente, establece un marco de comparación contra cualquier esquema de codificación posible ya que una propiedad importante establecida en la teoría es que la longitud promedio de cualquier código decodificable de forma única está acotada inferiormente por la entropía de la fuente, es decir:

$$H(P) \leq L(C)$$

La entropía es máxima cuando todos los símbolos son equiprobables y mínima (cero) cuando la fuente siempre emite el mismo símbolo [2].

Esta desigualdad indica que, aunque se pueden diseñar códigos eficientes como el código de Huffman, no es posible construir un código que tenga una longitud promedio menor a la entropía de la fuente sin perder información. Por tanto, la entropía actúa como un criterio teórico para evaluar el rendimiento de los esquemas de compresión [2].

Capítulo 3

Metodología

El conjunto de datos fue descargado de Kaggle el cual es de imágenes de Rayos X del tórax de diferentes personas, algunas diagnosticadas con neumonía según dos expertos y otras sanas. Los datos están divididos en tres carpetas: train (imágenes de entrenamiento del modelo), val (imágenes de validación) y test (imágenes de prueba). Cada una de estas, a su vez, se divide en una carpeta "NORMAL" (imágenes de pacientes sanos) y otra "PNEUMONIA" (imágenes de pacientes con neumonía), conteniendo un total de 5856 imágenes en formato ".jpeg" o ".jpg".

Este conjunto de datos está desbalanceado debido a que la clase "PNEUMONIA" cuenta con una gran cantidad de imágenes de este tipo a comparación de la cantidad de imágenes de la clase "NORMAL". Esto se puede ver reflejado en la siguiente imagen:

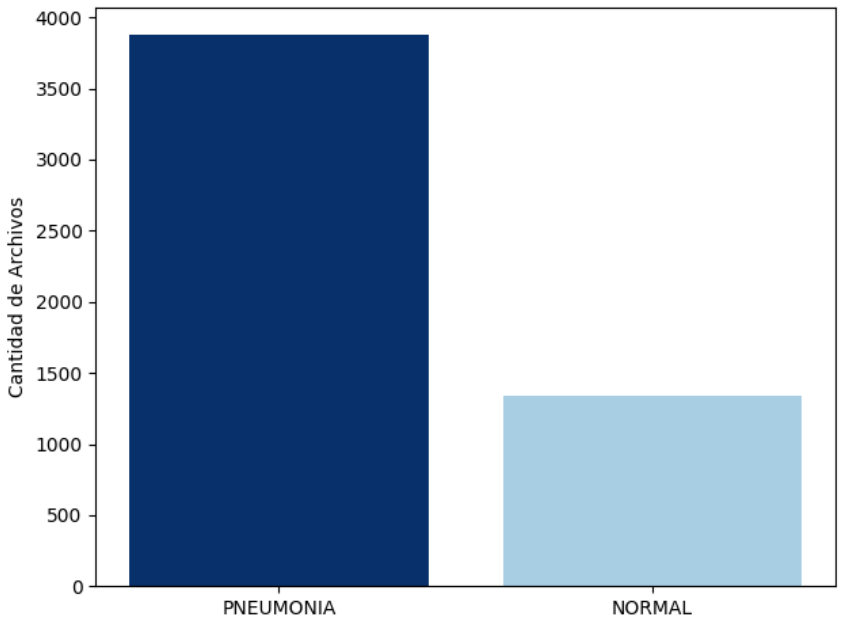

Figura 3.1: Comparación de la cantidad de imágenes en la carpeta "train".

Para mitigar este fallo, se modifican los pesos de las clases para penalizar fuertemente

los errores en la clase "NORMAL".

El modelo original tiene una entrada de (256, 256, 1) donde las primeras dos entradas de la tupla representan el tamaño de la imagen (256×256 pixeles) y la última entrada se refiere al canal del color; en este caso, es en escala de grises.

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 256, 256, 1)	0
conv2d (Conv2D)	(None, 254, 254, 32)	320
batch_normalization (BatchNormalization)	(None, 254, 254, 32)	128
max_pooling2d (MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_1 (Conv2D)	(None, 125, 125, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 125, 125, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 62, 62, 64)	0
conv2d_2 (Conv2D)	(None, 60, 60, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 60, 60, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 30, 30, 128)	0
conv2d_3 (Conv2D)	(None, 28, 28, 256)	295,168
batch_normalization_3 (BatchNormalization)	(None, 28, 28, 256)	1,024
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 256)	0
flatten (Flatten)	(None, 50176)	0
dense (Dense)	(None, 128)	6,422,656
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 64)	8,256
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 32)	2,080
dense_3 (Dense)	(None, 1)	33

Figura 3.2: Arquitectura del modelo.

Inicialmente, el modelo cuenta con sobre-ajuste, pero, para mantener homogeneidad en los resultados, no se modificarán las capas a excepción de la primera la cual es la que recibe los pixeles de las imágenes.

Además, para entrenar el modelo se usaron dos diferentes *callbacks*: el primero fue *EarlyStopping* el cual detiene el modelo cuando el parámetro *val_loss* incrementa durante

5 épocas, y si se detiene, se restaruan los mejores pesos; el segundo fue *ModelCheckpoint* el cual guarda los mejores resultados debido a que se tiene el parámetro *save_best_only* para poder usar el modelo posteriormente a su entrenamiento.

Al tener un conjunto de datos relativamente grande, no se puede asegurar que todas las imágenes sean de la misma resolución, por lo que se procede a ver esta característica de las mismas en la siguiente imagen:

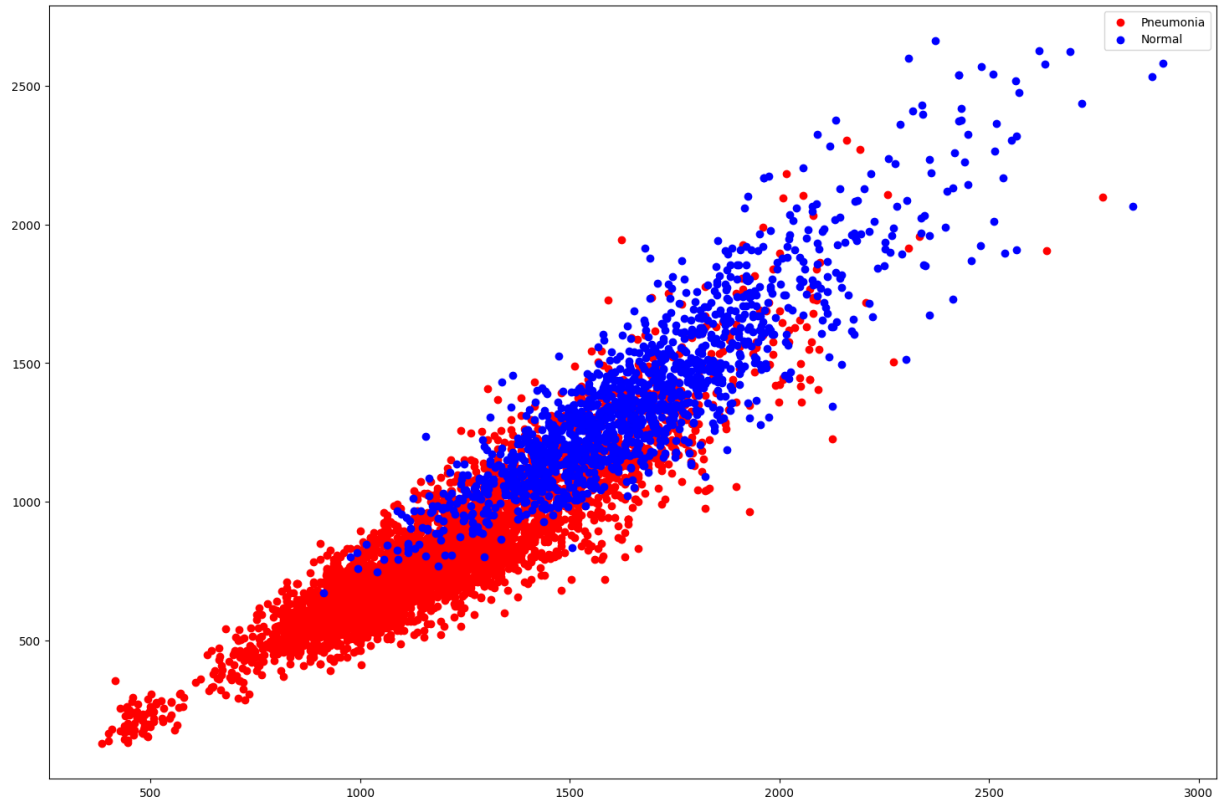

Figura 3.3: Pixeles de las imágenes.

A raíz de la figura 3.3, se puede notar que existe una gran variedad de resoluciones en el conjunto de datos, por lo que se puede asumir que, al momento de reducir el tamaño de todas las imágenes a 256×256 pixeles, se perderá información. Como esta resolución es la usada para entrenar la CNN, servirá de referencia para calcular la pérdida de información.

Para medir esta pérdida de información se usará la entropía y la longitud de Huffman, que se define como $L(C) = \sum_{i=1}^{q} p_i \cdot \ell_i$. Cada una de estas implementada en un código de Python el cual usa las siguientes librerías:

```
import os
import cv2
import heapq
```

```
import numpy as np
import matplotlib.pyplot as plt
from skimage.measure import shannon_entropy
```

Para calcular la entropía se utiliza la función *shannon_entropy* de la librería *skimage.measure* y la función utilizada es:

```
def entropia(img):
    if img.ndim == 3:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    return shannon_entropy(img)
```

Por otro lado, para calcular la longitud de Huffman se crea la función *longitud_huffman* que calcula la longitud promedio óptima de un código de Huffman para los niveles de gris de una imagen convirtiendo, si es necesario, la imagen a escala de grises, luego construyendo su histograma de 256 niveles y normalizándolo para obtener las probabilidades de cada nivel presente; después, inserta esas probabilidades en una estructura de montículo mínimo y, en cada iteración, extrae las dos menores, suma sus valores (que equivale a crear un nodo interno en el árbol de Huffman), acumula esa suma en un contador y vuelve a insertar el nodo combinado hasta que solo quede uno, de modo que la suma total de todas esas combinaciones corresponde precisamente a la longitud esperada (en bits) del código resultante. La función es la siguiente:

```
def longitud_huffman(img):
    if img.ndim == 3:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    hist = cv2.calcHist([img], [0], None, [256], [0, 256]).flatten()
    p = hist[hist > 0] / hist.sum()
    heap = list(p); heapq.heapify(heap)
    total = 0
    while len(heap) > 1:
        a = heapq.heappop(heap)
        b = heapq.heappop(heap)
        s = a + b
        total += s
        heapq.heappush(heap, s)
    return total  # L promedio
```

Para llevar a cabo todo el proceso se usa el siguiente código:

```
def reescala(img, size, interp):
    return cv2.resize(img, size, interpolation=interp)

# Configuración
```

```
ref_size = (256, 256)
resoluciones = {
    "224×224": (224, 224),
    "192×192": (192, 192),
    "160×160": (160, 160),
    "128×128": (128, 128),
}

carpetas = {
    "PNEUMONIA": "./chest_xray/train/PNEUMONIA",
    "NORMAL"   : "./chest_xray/train/NORMAL",
}

# Métricas a acumular
metricas = ["H256", "Hr", "ΔH", "L256", "Lr", "ΔL"]
resultados = {
    clase: {tag: {m: [] for m in metricas} for tag in resoluciones}
    for clase in carpetas
}

# Bucle de procesamiento
for clase, path in carpetas.items():
    for entry in os.scandir(path):
        if not entry.name.lower().endswith(('.jpg','.jpeg','.png')):
            continue
        img_orig = cv2.imread(entry.path)
        if img_orig is None:
            continue

        # Fuente de referencia 256×256 (INTER_AREA)
        X256 = reescala(img_orig, ref_size, cv2.INTER_NEAREST)
        H256 = entropia(X256)
        L256 = longitud_huffman(X256)

        for tag, size in resoluciones.items():
            Yr = reescala(img_orig, size, cv2.INTER_NEAREST)
            Hr = entropia(Yr)
            Lr = longitud_huffman(Yr)

            resultados[clase][tag]["H256"].append(H256)
            resultados[clase][tag]["Hr"].append(Hr)
            resultados[clase][tag]["ΔH"].append(H256 - Hr)
            resultados[clase][tag]["L256"].append(L256)
            resultados[clase][tag]["Lr"].append(Lr)
            resultados[clase][tag]["ΔL"].append(L256 - Lr)

# Cálculo de promedios
```

```
prom = {
    clase: {
        tag: {m: np.mean(resultados[clase][tag][m]) for m in metricas}
        for tag in resoluciones
    }
    for clase in carpetas
}

# Impresión
for clase in carpetas:
    print(f"\n--- {clase} ---")
    for tag, vals in prom[clase].items():
        print(f"{tag}: ΔH={vals['ΔH']:.4f} bits, ΔL={vals['ΔL']:.4f} bits")

# Gráficas
tags = list(resoluciones.keys())
x = np.arange(len(tags))
width = 0.35

plt.figure(figsize=(18, 6))
plt.subplot(1, 2, 1)
plt.bar(x + width/2, [prom["NORMAL"][t]["ΔH"] for t in tags], width,
↪  color="#87cefa", label="NORMAL")
plt.bar(x - width/2, [prom["PNEUMONIA"][t]["ΔH"] for t in tags], width,
↪  color="#1f77b4", label="PNEUMONIA")
plt.xticks(x, tags)
plt.title("Δ Entropía")
plt.xlabel("Resolución")
plt.ylabel("Δ Entropía (bits)")
plt.legend()

plt.subplot(1, 2, 2)
plt.bar(x + width/2, [prom["NORMAL"][t]["ΔL"] for t in tags], width,
↪  color="#87cefa", label="NORMAL")
plt.bar(x - width/2, [prom["PNEUMONIA"][t]["ΔL"] for t in tags], width,
↪  color="#1f77b4", label="PNEUMONIA")
plt.xticks(x, tags)
plt.title("Δ Longitud Huffman")
plt.xlabel("Resolución")
plt.ylabel("Δ Longitud Huffman (bits)")
plt.legend()
plt.show()
```

Capítulo 4

Resultados

Los resultados son los siguientes:

```
--- PNEUMONIA ---
224×224: ΔH=0.0006 bits, ΔL=0.0007 bits
192×192: ΔH=0.0015 bits, ΔL=0.0017 bits
160×160: ΔH=0.0031 bits, ΔL=0.0034 bits
128×128: ΔH=0.0066 bits, ΔL=0.0071 bits

--- NORMAL ---
224×224: ΔH=0.0009 bits, ΔL=0.0012 bits
192×192: ΔH=0.0025 bits, ΔL=0.0029 bits
160×160: ΔH=0.0049 bits, ΔL=0.0057 bits
128×128: ΔH=0.0090 bits, ΔL=0.0103 bits
```

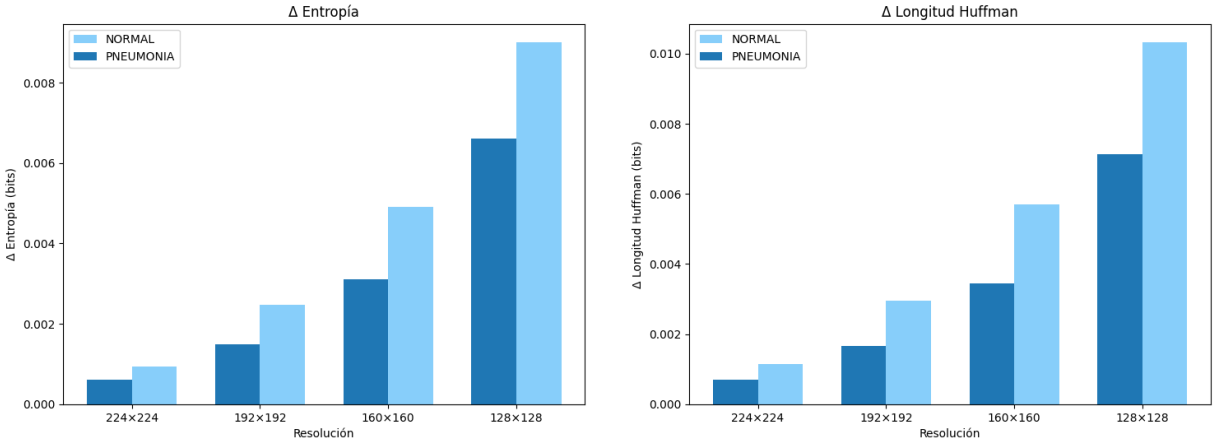

Figura 4.1: Diferencia de Entropía y Longitud de Huffman.

En el caso de "PNEUMONIA" la entropía cae 0.0006 bits en la resolución 224×224 y, para la resolución 128×128 llega a 0.0066 bits. Esta poca variabilidad en la diferencia de

entropía indica que la complejidad visual y la dispersión de contraste en las zonas grises no se ven afectadas en gran manera. Por otro lado, en la clase "NORMAL" la diferencia va desde 0.0009 bits a 0.0090 bits la cual es una diferencia un poco mayor que en el otro caso, pero la imagen se preserva en gran medida.

Para la longitud de Huffman, "PNEUMONIA" va desde 0.0007 bits a 0.0071 bits lo que indica que esta longitud parece estar intacta ya que se requieren algunas milésimas adicionales para poder representarla. Para "NORMAL" pasa de 0.0012 bits a 0.103 bits que ya hace que se haga un salto a décimas extra después de la compresión; no obstante, en la resolución de 160×160 se puede observar una $\Delta L = 0.0057$ bits lo que indica que, en este caso, todavía es aceptable esta resolución.

Dados estos resultados, se puede observar que una resolución que no pierde demasiada información y podría ayudar a que la CNN se entrene de una manera más rápida y sin perder eficiencia es 160×160. Los resultados se muestran a continuación:

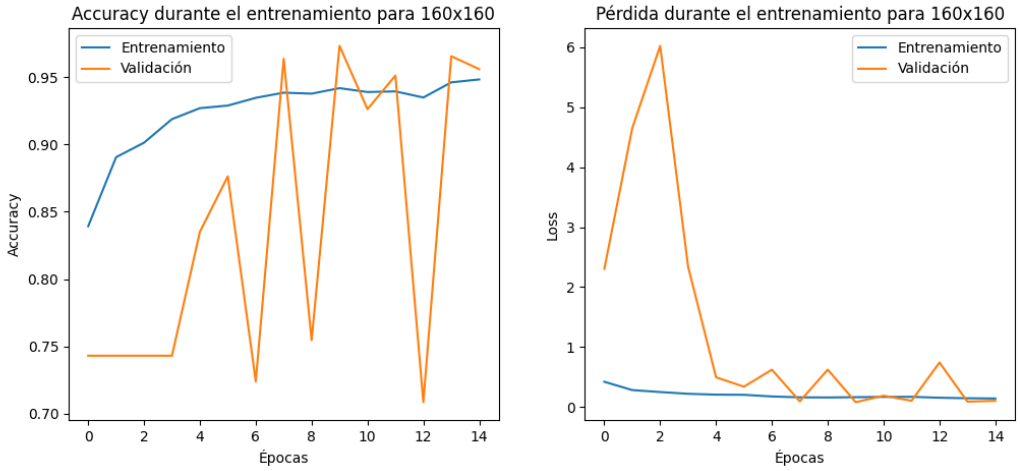

Figura 4.2: Entrenamiento de la CNN.

Como se mencionó anteriormente, la CNN cuenta con un sobre-ajuste durante sus 15 épocas de entrenamiento, y se puede notar en los picos agresivos que se ven en la gráfica de validación tanto para la precisión como para la pérdida. A pesar de esto, los resultados en el conjunto de pruebas fueron relativamente buenos:

```
33/33 ———————————— 3s 100ms/step - accuracy: 0.9698 - loss: 0.0987
Loss: 0.1019, Accuracy: 0.9655
Found 624 images belonging to 2 classes.
624/624 ———————————— 3s 4ms/step - accuracy: 0.8358 - loss: 0.4024
Precisión en el conjunto de prueba de resolución 160x160: 88.62%
```

Se obtuvo un 88.65 % de precisión lo cual no está muy lejos del 90.38 % del original. Además, como se mencionó anteriormente, se entrenó durante 15 épocas con un tiempo de

37 segundos por cada una, mientras que la CNN original se entrenó por 20 épocas durando 105 segundos cada una lo que indica una gran disminución de tiempo sin sacrificar un gran porcentaje de precisión.

Finalmente, la matriz de confusión se ve de la siguiente manera:

```
624/624 ———————————————————— 3s 4ms/step

Reporte de clasificación para 160x160:
              precision    recall  f1-score   support

      NORMAL       0.95      0.74      0.83       234
   PNEUMONIA       0.86      0.98      0.91       390

    accuracy                           0.89       624
   macro avg       0.91      0.86      0.87       624
weighted avg       0.89      0.89      0.88       624
```

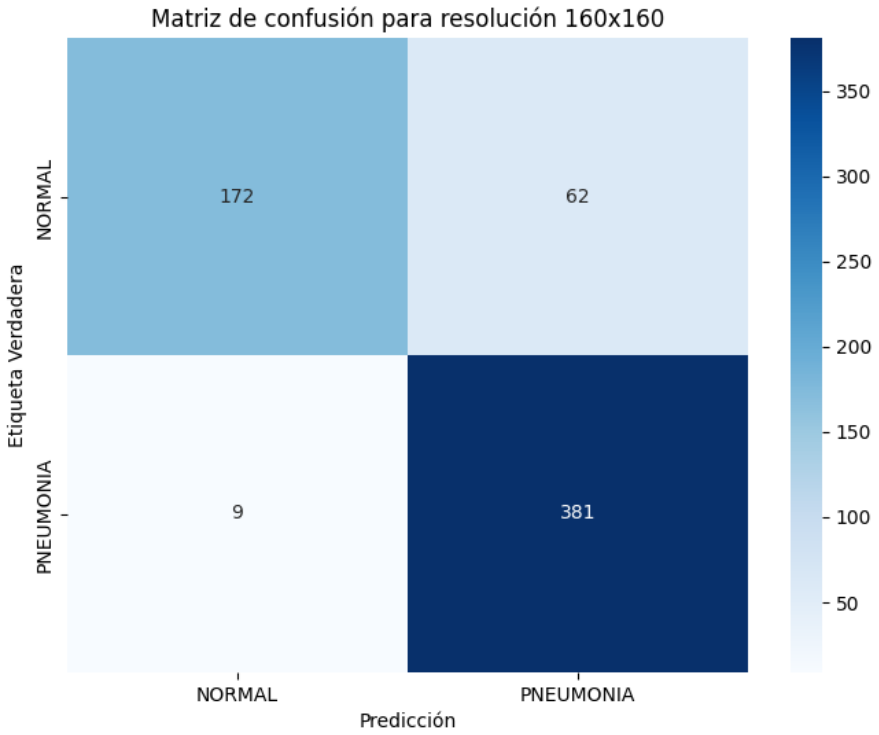

Figura 4.3: Matriz de confusión.

Como se puede observar, el sobre-ajuste se hace evidente al ver que predice "PNEUMONIA" en casos de "NORMAL" y se puede deber al desbalance en el conjunto de datos.

Capítulo 5

Conclusiones

Tras el análisis de la caída de entropía y de la longitud óptima de código de Huffman en resoluciones de 224×224, 192×192, 160×160 y 128×128 para ambas clases, se concluye que:

- La resolución de 160×160 es una buena resolución: conserva casi toda la complejidad visual ($\Delta H \approx 0.003$-0.005 bits) y la longitud de Huffman ($\Delta L \approx 0.003$-0.006 bits), mientras que permite entrenar la CNN en 37 segundos por época en lugar de 105 segundos, con apenas una caída de precisión de $90.38\,\% \rightarrow 88.62\,\%$ reduciendo hasta en un $65\,\%$ el tiempo de entrenamiento.
- El uso de métricas de teoría de códigos demuestra ser un indicador fiable y cuantitativo de la pérdida de información al reescalar, facilitando la selección de la resolución que ayude a minimizar tiempo.
- A pesar del sobre-ajuste residual y del sesgo hacia la clase "PNEUMONIA", evidenciado en la matriz de confusión, el desempeño del modelo en 160×160 se mantiene aceptable.
- Reducir la resolución a 160×160 abre la posibilidad de desplegar sistemas de diagnóstico asistido por CNN en entornos con recursos limitados sin comprometer significativamente la efectividad.

Como trabajo futuro, se puede explorar resoluciones intermedias e incorporar otras técnicas de balanceo para evitar el sesgo que tiene la CNN. Además, es un proyecto escalable que permite trasladarse a otras áreas de clasificación por imágenes.

Bibliografía

[1] BEEP Blog. ¿Qué es un píxel?, 2023. Último acceso: 15 de abril de 2025.

[2] Gareth A. Jones and J. Mary Jones. *Information and Coding Theory*. Springer Undergraduate Mathematics Series. Springer-Verlag London, 2000.

[3] Federico Lubinus Badillo, César Andrés Rueda Hernández, Boris Marconi Narváez, and Yhary Estefanía Arias Trillos. Redes neuronales convolucionales: un modelo de *Deep Learning* en imágenes diagnósticas. revisión de tema. *Revista Colombiana de Radiología*, 32(3):5591–5599, 2021.

[4] MathWorks. Redes Neuronales Convolucionales (CNN), 2024. Último acceso: 14 de abril de 2025.

[5] Pranav Rajpurkar, Jeremy Irvin, Kaylie Zhu, Brandon Yang, Hershel Mehta, Tony Duan, Daisy Yi Ding, Aarti Bagul, Curtis P. Langlotz, Katie S. Shpanskaya, Matthew P. Lungren, and Andrew Y. Ng. CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning. *CoRR*, abs/1711.05225, 2017.