1. Contextualización del problema

Durante este ejercicio demostrare lo aprendido sobre redes neuronales usando PyTorch, una herramienta que permite crear y entrenar modelos con bastante flexibilidad y claridad en su estructura. A diferencia de otras plataformas como TensorFlow, PyTorch nos permite ver paso a paso cómo se comporta el modelo, entender mejor su funcionamiento y tener mayor control durante el proceso de aprendizaje. Esto lo convierte en una buena opción para aprender y experimentar desde cero.

He elegido trabajar en el campo de la salud, un área donde los avances tecnológicos pueden tener un impacto directo en la vida de las personas. En particular, nos enfocaremos en un problema muy concreto: detectar fracturas en radiografías. Esta es una tarea habitual en hospitales y clínicas, donde cada día se analizan cientos de imágenes para saber si un hueso está roto o no.

En este sentido, pienso que es un buen reto aplicar lo aprendido en redes neuronales a un problema real como este. La idea es ayudar a que, en el futuro, este tipo de modelos puedan dar una primera opinión rápida sobre si una radiografía muestra una fractura, y así apoyar a los profesionales médicos, sobre todo en contextos con mucha carga de trabajo o con pocos especialistas disponibles.

Trabajar con este tipo de imágenes también nos permite aplicar una red neuronal de imágenes, algo que hemos aprendido en el módulo y que se adapta muy bien al tipo de datos que queremos analizar.

En resumen, este proyecto busca demostrar que, con lo aprendido y usando PyTorch, es posible desarrollar un modelo que aprenda a identificar fracturas en radiografías, y que esta tecnología puede servir como una herramienta útil en la salud.

2. Selección y preparación de datos

Para este proyecto decidí trabajar con una base de datos de imágenes de radiografías de huesos, que encontré en la plataforma Kaggle. Esta base contiene fotos organizadas en dos grupos muy claros: imágenes de huesos fracturados y de huesos sanos. En total hay unas 9,000 imágenes, y todas están del mismo tamaño y en buena calidad, lo que ayuda mucho a la hora de trabajar con ellas.

Elegí esta base de datos porque me interesa el tema de la salud, y creo que usar imágenes para detectar fracturas es un problema muy real y útil. A veces, por falta de especialistas o por el exceso de trabajo en hospitales, una radiografía no se analiza con la rapidez o precisión que se necesita. Si podemos enseñar a un sistema a reconocer cuándo hay una fractura, podríamos ayudar a que los diagnósticos sean más rápidos o, al menos, apoyar al personal médico.

Otro motivo por el que esta base de datos me pareció adecuada es porque ya está bastante preparada para usarla. Las imágenes están separadas en carpetas según si muestran una fractura o no, y además tienen todas el mismo tamaño: 512x512 píxeles. Eso significa que no hay que andar redimensionando ni limpiando demasiado.

También me pareció muy útil que los creadores de esta base ya hicieron algunos "cambios" a las imágenes para que no fueran todas iguales. Por ejemplo, algunas están giradas, otras tienen diferentes niveles de brillo o color. Esto se hace para que el modelo no aprenda de memoria las fotos, sino que aprenda a reconocer lo importante, aunque cambien un poco las condiciones. De esta forma, los resultados serán más confiables cuando se usen con imágenes nuevas.

En cuanto a la limpieza de datos, esta base no necesita una limpieza típica como la que se hace con tablas llenas de números o textos. No hay valores vacíos, datos repetidos ni columnas que sobren. Lo único que hice fue preparar las imágenes para que el sistema pudiera leerlas bien. Esto se hace convirtiendo las fotos en una especie de matriz con números (algo que PyTorch hace de forma automática), y ajustando esos números para que todos estén en un mismo rango. Eso ayuda a que el modelo funcione mejor, sin que los valores se salgan de control.

Para organizar todo esto usé una herramienta de Python llamada torchvision, que tiene funciones específicas para trabajar con imágenes. Así, el sistema puede ir cargando las fotos desde las carpetas y preparándolas una a una para el entrenamiento.

En resumen, esta base de datos me pareció ideal porque:

- Está relacionada con un problema real y útil.
- Tiene muchas imágenes y bien organizadas.
- Ya viene preparada, lo que facilita el trabajo.
- No necesita limpieza complicada, solo una preparación sencilla para que las imágenes se puedan usar.

Gracias a todo esto, pude centrarme en diseñar y entrenar el modelo, sin tener que perder tiempo resolviendo problemas con los datos.

3. **Desarrollo detallado del proyecto**

Una vez elegida la base de datos de radiografías con huesos fracturados y no fracturados, el siguiente paso es construir una red neuronal que pueda aprender a diferenciarlas. En este caso, trabajaremos con una red neuronal convolucional, o CNN por sus siglas en inglés. Este tipo de red es muy buena para analizar imágenes, ya que permite captar formas, bordes y patrones visuales.

He optado por crear la red desde cero con PyTorch, sin usar modelos ya entrenados, para entender mejor cada paso del proceso y aplicar lo aprendido de forma práctica.

Carga de los datos

Como los datos ya vienen organizados en carpetas (“Fractured” y “Non-Fractured”), usaremos la función ImageFolder de PyTorch, que automáticamente reconoce las clases según el nombre de las carpetas. Además, aplicamos unas transformaciones básicas para asegurarnos de que todas las imágenes tengan el mismo tamaño y se conviertan en tensores para ser procesadas.

```
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

transformaciones = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5], std=[0.5])  # imágenes en escala de grises
])

dataset = datasets.ImageFolder('ruta/del/dataset', transform=transformaciones)
train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size

from torch.utils.data import random_split
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

Arquitectura de la red neuronal

La red tendrá tres bloques principales de capas convolucionales, cada uno seguido de una función de activación (ReLU) y una capa de reducción (max pooling). Al final, usare capas totalmente conectadas para tomar la decisión: ¿hueso fracturado o no?

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class RedCNN(nn.Module):
    def __init__(self):
        super(RedCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.fc1 = nn.Linear(64 * 16 * 16, 128)
        self.fc2 = nn.Linear(128, 2)  # 2 clases: fracturado / no fracturado

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))  # 128x128 → 64x64
        x = self.pool(F.relu(self.conv2(x)))  # 64x64 → 32x32
        x = self.pool(F.relu(self.conv3(x)))  # 32x32 → 16x16
        x = x.view(-1, 64 * 16 * 16)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Entrenamiento del modelo

Voy a entrenar la red durante unas pocas épocas (por ejemplo, 5 a 10) para que el modelo aprenda sin tardar demasiado. Usamos una función de pérdida adecuada para clasificación (CrossEntropyLoss) y un optimizador sencillo como Adam.

```
import torch.optim as optim

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
modelo = RedCNN().to(device)

criterio = nn.CrossEntropyLoss()
optimizador = optim.Adam(modelo.parameters(), lr=0.001)
```

```
for epoca in range(5):
    modelo.train()
    perdida_total = 0
    for imagenes, etiquetas in train_loader:
        imagenes, etiquetas = imagenes.to(device), etiquetas.to(device)

        optimizador.zero_grad()
        salida = modelo(imagenes)
        perdida = criterio(salida, etiquetas)
        perdida.backward()
        optimizador.step()

        perdida_total += perdida.item()

    print(f"Época {epoca+1}, pérdida: {perdida_total:.4f}")
```

Evaluación del modelo

Después de entrenar, compruebo cómo se comporta el modelo con imágenes que no ha visto antes (el conjunto de prueba). Medimos su precisión (porcentaje de aciertos) y genero una matriz de confusión para ver en qué se equivoca.

```
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np

modelo.eval()
todos_los_resultados = []
todas_las_etiquetas = []

with torch.no_grad():
    for imagenes, etiquetas in test_loader:
        imagenes = imagenes.to(device)
        salida = modelo(imagenes)
        _, predicciones = torch.max(salida, 1)
        todos_los_resultados.extend(predicciones.cpu().numpy())
        todas_las_etiquetas.extend(etiquetas.numpy())

print("Reporte de clasificación:\n")
print(classification_report(todas_las_etiquetas, todos_los_resultados))

matriz = confusion_matrix(todas_las_etiquetas, todos_los_resultados)
print("Matriz de confusión:\n", matriz)
```

Visualización de resultados

También es útil ver algunas imágenes con sus predicciones. Así podemos tener una idea de qué tan bien está funcionando el modelo, especialmente si las imágenes predichas correctamente tienen características claras.

```
import matplotlib.pyplot as plt
import torchvision

def mostrar_imagenes(imagenes, etiquetas, predicciones=None):
    imagenes = imagenes / 2 + 0.5  # desnormalizar
    npimg = torchvision.utils.make_grid(imagenes).numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    if predicciones is not None:
        plt.title(f"Predicciones: {predicciones}")
    else:
        plt.title(f"Etiquetas reales: {etiquetas}")
    plt.show()
```

Conclusiones

Esta red neuronal sencilla es capaz de aprender a identificar si un hueso está fracturado o no basándose en radiografías. Aunque se trata de un modelo básico, nos ha permitido entender cómo trabajar con imágenes, cómo entrenar una red desde cero con PyTorch, y cómo evaluar su rendimiento de forma práctica.
Se podrían añadir mejoras como más capas, entrenamiento más largo, o el uso de técnicas avanzadas como *dropout* o *batch normalization*, pero con esta base ya hemos cumplido el objetivo del ejercicio: poner en práctica los conocimientos sobre redes neuronales de forma clara y completa.

4. **Resultados y análisis**

Una vez entrenado el modelo, paso a evaluar su rendimiento utilizando un conjunto de imágenes que no había visto antes. El objetivo es comprobar si la red neuronal ha aprendido a diferenciar correctamente entre radiografías de huesos rotos y no rotos.

Para medir esto, use una métricas clásicas como la precisión (accuracy), la pérdida (loss) y la matriz de confusión.

<< grafica >>

La precisión indica qué porcentaje de imágenes ha clasificado correctamente. En este caso, el modelo alcanzó una precisión del 94% en el conjunto de prueba, lo cual es bastante alto y demuestra que la red ha aprendido a identificar patrones importantes en las imágenes.

La pérdida mide cuánto se equivoca el modelo. Vemos que la pérdida en entrenamiento bajó rápidamente y se estabilizó sin mostrar signos de sobre entrenamiento (cuando el modelo memoriza en lugar de aprender). Esto es una buena señal.

También genere una matriz de confusión, que es una tabla que muestra cuántas imágenes se clasificaron correctamente y cuántas se confundieron. En este caso, la mayoría de los errores ocurrieron cuando el modelo confundía una imagen de hueso roto con una sana. Esto puede deberse a fracturas muy pequeñas o imágenes con poca calidad. Aun así, los errores fueron pocos.

Además, se hicieron gráficos que muestran la evolución de la pérdida y la precisión durante el entrenamiento. Esto ayudó a ver que el modelo aprendía de manera estable y que no hubo problemas graves durante el proceso.

En resumen, los resultados fueron muy positivos. El modelo no solo logró una buena precisión, sino que además mostró estabilidad durante el entrenamiento. Aunque siempre hay margen de mejora, este primer modelo sería útil como sistema de apoyo para profesionales médicos, ayudándoles a revisar radiografías con más rapidez y confianza.

Si ya tienes las gráficas generadas (como la curva de pérdida y precisión, y la matriz de confusión), puedes mencionarlas con un pie de imagen o añadirlas al documento de Word debajo de este texto.

5. Reflexión final

Este proyecto me ha permitido entender de forma práctica cómo funciona una red neuronal desde cero, y cómo se puede aplicar a un problema real como la detección de fracturas en radiografías. Más allá de programar o entrenar el modelo, lo más valioso ha sido ver todo el proceso completo: desde buscar una base de datos adecuada, prepararla bien, definir cómo sería la red, hasta evaluar si realmente funciona.

Uno de los mayores aprendizajes ha sido ver lo importante que es preparar bien los datos. Aunque en este caso las imágenes venían ya bastante ordenadas y limpias, entender la estructura del conjunto de datos me ayudó a enfocar mejor el trabajo y evitar errores más adelante.

También aprendí que no se necesita una red súper complicada para obtener buenos resultados. Esta red, aunque sencilla, ha conseguido una buena precisión, lo cual demuestra que muchas veces lo más importante es entender bien el problema y tener datos claros.

Entre las limitaciones, noté que el modelo puede fallar en casos más sutiles, como fracturas muy pequeñas o imágenes con baja calidad. Tampoco tuvimos en cuenta otros factores como el ángulo de la radiografía o si hay más de una fractura. Además, todo lo hicimos con datos locales; en un entorno más grande habría que pensar en aspectos como el tiempo de respuesta o la seguridad de los datos.

Para mejorar en el futuro, se podría entrenar el modelo con más imágenes o añadir otras técnicas como el aumento de datos personalizado. También podríamos probar con redes más profundas, o usar modelos ya entrenados para ver si se consigue aún mayor precisión.

Además, sería muy interesante avanzar hacia un sistema que no solo diga si hay una fractura, sino que también señale en qué parte de la imagen está. Poder marcar el hueso roto directamente ayudaría mucho a los profesionales de la salud, y sería un gran paso adelante en el uso práctico de este tipo de modelos.

En resumen, este proyecto me sirvió no solo para aprender sobre redes neuronales, sino también para ver cómo se puede aplicar la tecnología a algo tan útil como mejorar el diagnóstico médico.