



PREDICCIÓN DE NEUMONIA CON RADIOGRAFIAS TORÁCICAS

Autores

Gonzalez Marcos, Álvaro



KSCHOOL

Master de Ciencia de Datos de Kschool

Abstract

Las enfermedades respiratorias suponen una de las principales causas de muerte año tras año alrededor de todo el mundo. Enfermedades tan comunes y estudiadas como la neumonía supone un 30% de las defunciones provocadas por este tipo de afecciones y el reciente COVID-19, puso en jaque a los sistemas sanitarios en todos los países, poniendo de manifiesto sus limitaciones. Esto demuestra que los métodos actuales para el diagnóstico de enfermedades se están quedando obsoletos en gran medida por la gran cantidad de tiempo que necesitan para obtener un resultado. Por eso, es necesario el desarrollo de tecnologías de diagnóstico eficaces que permitan una valoración de la enfermedad de manera rápida y precisa. En esa línea, el gran avance de la inteligencia artificial, el big data y sus técnicas, en especial el Deep learning han supuesto un importante punto de inflexión, aportando esperanza y nuevas herramientas muy prometedoras para el desarrollo de nuevas metodologías de diagnóstico. En este estudio hacemos uso de todas estas tecnologías con el fin de construir un modelo predictivo eficaz y preciso para la predicción de neumonía a partir de radiografías torácicas, históricamente empleadas por médicos y radiólogos expertos para el diagnóstico de esta patología. Entre ellas destacan las redes neuronales convolucionales y una metodología conocida como transfer learning o aprendizaje cruzado. Paralelamente, abordamos otro de los grandes problemas existentes en la inteligencia artificial, que es la reducción de dimensionalidad ante el gran tamaño de las bases de datos actuales mediante el uso de una tecnología conocida como Autoencoder y una función que calcula la capacidad predictiva de las variables. Mediante este estudio, demostramos que el desarrollo de modelos predictivos gracias a los grandes avances en la inteligencia artificial es posible, pero aún es necesaria una alta potencia computacional, así como un conjunto de datos grande y fiable para obtener resultados buenos y relevantes.

Contenido

Abstract.....	
1. Introducción.....	1
Contexto del Problema.....	1
Motivación y Relevancia del Estudio	1
2. Objetivos.....	3
Objetivo General.....	3
Objetivos específicos	3
3. Datos.....	3
Fuentes de Datos.....	3
Descripción de los Datos.....	3
Preprocesamiento de los Datos.....	5
División de los Datos.....	5
4. Tecnología.....	6
5. Modelización.....	8
5.1 Primera Modelización.....	8
5.2 Segunda Modelización.....	13
5.2.1 Autoencoder	13
5.2.2 Compresor con CNN	21
5.3 Tercera Modelización	27
6. Resultados.....	29
Beneficios Económicos y Sociales.....	29
7. Despliegue Tecnológico	30
Infraestructura de Despliegue	30
Proceso de Despliegue.....	30
8. Puesta en Valor	31
Impacto en el Diagnóstico Médico	31
Futuras Líneas de Investigación.....	32
9. Conclusiones, Observaciones y Contribuciones	33
Conclusiones.....	33
Observaciones	33
Contribuciones.....	33
10. Bibliografía.....	34

1. Introducción

Contexto del Problema

Las enfermedades respiratorias, incluyendo la neumonía que año tras año supone el 30% de las muertes causadas por este tipo de afecciones en todo el mundo (1); o el fatal COVID-19 que desde su aparición ha provocado la muerte de más de medio millón de personas y contagiado a más de 10 (2), constituyen una de las principales causas de muerte y una de las grandes preocupaciones de los expertos. Uno de los mejores mecanismos para asegurar la supervivencia de los pacientes es un rápido diagnóstico de la enfermedad que permita una rápida actuación y comienzo de un tratamiento adecuado y eficiente. Históricamente, técnicas como la tomografía computarizada de alta resolución, la ecografía endobronquial o pruebas genéticas han sido algunos de los principales métodos de diagnóstico (3). No obstante, no parecen ser suficientes para reducir el número de muertes alrededor del mundo, en gran parte debido a la gran cantidad de tiempo que necesitan para realizar un diagnóstico acertado (4).

Recientemente, el COVID-19 ha desafiado los sistemas de salud en todo el mundo, poniendo a prueba su capacidad de respuesta y sus recursos. Por lo tanto, existe la necesidad imperiosa de encontrar y desarrollar métodos de diagnóstico rápidos, seguros y eficaces que nos permitan detectar a la población enferma para evitar su propagación y tratar a los pacientes de manera adecuada. La herramienta más empleada para su diagnóstico ha sido la prueba de la PCR, con una gran precisión, pero no pudo frenar su propagación hasta el último rincón del planeta (4). Otra de las herramientas empleadas ha sido la radiografía torácica, la cual puede revelar signos característicos de infecciones pulmonares.

Motivación y Relevancia del Estudio

Con el avance de la inteligencia artificial (IA) y el big data, surge la oportunidad de utilizar estos recursos para mejorar y acelerar el diagnóstico de estas enfermedades respiratorias, entre otras (5). Los primeros intentos por utilizar las principales tecnologías de la IA con el fin de mejorar la interpretación de imágenes radiológicas y sentar las bases de la detección y seguimiento del cáncer de pulmón datan de hace casi 60 años (5). No obstante, estas primeras aproximaciones no consiguieron grandes resultados dadas las limitaciones tecnológicas de la época tanto por la falta de grandes repositorios de radiografías como de potencia computacional (6, 7, 8).

En las siguientes décadas el gran avance de la IA ligado al surgimiento de nuevas y potentes técnicas y más concretamente, el Deep learning que permite la creación de complejos algoritmos que permiten la clasificación de imágenes o el entendimiento del lenguaje entre otros, supuso un hito determinante (6, 7, 9). Esto ha permitido su aplicación en campos científicos como la radiología llegando a desarrollar algoritmos, incluso mejores que el ojo humano para la clasificación y diagnóstico de enfermedades a partir de imágenes (8, 9).

Los principales esfuerzos para la implementación de esta clase de algoritmos en el campo de la radiología han ido encaminados al estudio de importantes enfermedades que tenían en su origen esta metodología como uno de sus principales métodos de diagnóstico. Estamos hablando tanto de patologías cardiovasculares (10), como del cáncer de mama (8), la cual es una de las enfermedades con mayor inversión en todo el

mundo; enfermedades pulmonares como el cáncer de pulmón, la tuberculosis (10), o el reciente COVID-19 (4); y enfermedades torácicas en general como el cáncer de tiroides (11), o la fibrosis (10). Incluso, enfermedades neuronales como el Alzheimer también se están beneficiando de esta cooperación entre radiología y DL para mejorar los mecanismos de diagnóstico, consiguiendo resultados prometedores (12).

Todos los algoritmos desarrollados para mejorar el diagnóstico de estas patologías tienen un factor común dentro del DL y no es otro que las redes neuronales convolucionales (CNN). Las CNN han permitido desde su surgimiento a principios de este siglo (8), el desarrollo de una gran cantidad de programas para el análisis y clasificación de imágenes, muchos de ellos de una gran precisión, lo que las ha convertido en una prometedora herramienta que garantice el éxito del DL en el ámbito médico (13). Sin embargo, hasta hace pocos años, la cantidad de datasets con imágenes de ámbito médico y el tamaño de los mismos era bastante limitado (9), comparado con la gran cantidad de datasets disponibles en otros muchos sectores. Por lo tanto, los principales algoritmos desarrollados hasta nuestros días y que han sido utilizados en diferentes estudios para evaluar su utilización para el diagnóstico de enfermedades, fueron contruidos y entrenados en base a datasets con imágenes cuyas características son muy distintas (9).

Esta metodología es ampliamente utilizada en el DL y es conocida con el nombre de transfer learning o aprendizaje cruzado, el cual también emplearemos en este estudio. Este se refiere a la utilización de modelos entrenados con anterioridad utilizando un dataset distinto, pero que guarda ciertos patrones de similitud con nuestro dataset de interés (9, 14). Todo esto ha llevado a que, a pesar de que haya estudios que han conseguido grandes resultados aún no se haya desarrollado un algoritmo entrenado y construido a partir de datos e imágenes puramente médicos. Es por eso que, mediante este trabajo, queremos construir a partir de un dataset de imágenes radiológicas, un modelo de predicción propio que nos permita predecir y diagnosticar la enfermedad que padece el paciente de manera rápida, eficaz e inmediata a partir de una radiografía torácica. La enfermedad que hemos elegido ha sido la pneumonia al tratarse de una de las principales causas de muerte en enfermedades respiratorias en todo el mundo.

Adicionalmente, en este trabajo, también trataremos uno de los principales problemas existentes dentro de la IA. La existencia de datasets enormes con una gran cantidad de información han provocado que ante la falta de la potencia computacional necesaria aparezcan problemas para procesar estas grandes cantidades de información y emplearlas para realizar modelos y predicciones adecuadas (15). Como solución ante este suceso surgió la reducción de dimensionalidad que permite la utilización de grandes datasets reduciendo ostensiblemente su tamaño, pero conservando sus principales características (15).

En este proyecto hemos reducido las dimensiones de nuestro dataset utilizando dos metodologías distintas. La primera de todas ha sido un reajuste directo de las dimensiones de las imágenes de radiografías a unas dimensiones previamente fijadas, y que serán comunes a todo el dataset, previo a su procesamiento y paso por el modelo. La segunda consiste en la utilización de un algoritmo de DL desarrollado con este fin y conocido con el nombre de Autoencoders (15, 16). Esta clase de algoritmo está dividido en dos arquitecturas distintas. La primera o Encoder se encarga de recoger las principales características de nuestros datos y las transforma a un espacio dimensional

muy inferior al original (15, 16). La segunda o Decoder se encarga de restaurar la representación hecha por el Encoder a sus dimensiones originales haciendo que esta sea lo más parecida a los datos originales (15, 16).

Una de las múltiples aplicaciones que tiene este algoritmo es como compresor de imágenes (15, 16), de la cual nos hemos beneficiado en este estudio. Esta se centra en la parte del Encoder y su salida la cual supone una representación fiel de la imagen original, pero con unas dimensiones muy inferiores. Para ello, y con el fin de probar su eficacia e influencia en la precisión del modelo predictivo hemos construido dos compresores. El primero emplea una red neuronal, mientras que el segundo utiliza una CNN.

2. Objetivos

Objetivo General

Crear un modelo de inteligencia artificial basado en redes neuronales convolucionales (CNN) para la detección de neumonía y diversas enfermedades respiratorias con radiografías torácicas (25).

Objetivos específicos

- Desarrollar un modelo para la detección automática de neumonía en radiografías torácicas.
- Evaluar el rendimiento del modelo
- Estudiar y aplicar diferentes técnicas de dimensionalidad de nuestro dataset y ver los resultados que tiene en nuestro modelo
- Identificar limitaciones

3. Datos

Fuentes de Datos

El conjunto de datos utilizado proviene del Chest X-ray Images (Pneumonia) Dataset disponible en Kaggle:

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

Descripción de los Datos

El conjunto de datos está organizado en 3 carpetas (train, test, val) y contiene subcarpetas para cada categoría de imagen (Neumonía/Normal). Hay 5,863 imágenes de rayos X (JPEG) y 2 categorías (Neumonía/Normal).

Las imágenes de rayos X torácicas (Fig. 1), provienen del Centro Médico de Mujeres y Niños de Guangzhou y fueron realizadas como parte de la atención clínica rutinaria de los pacientes.

Las imágenes fueron revisadas y etiquetadas por radiólogos expertos eliminando todas las exploraciones de baja calidad o ilegibles. Los diagnósticos de las imágenes fueron evaluados por dos médicos expertos antes de ser aprobados para el entrenamiento del

sistema de IA donde además el conjunto de evaluación también fue revisado por un tercer experto.

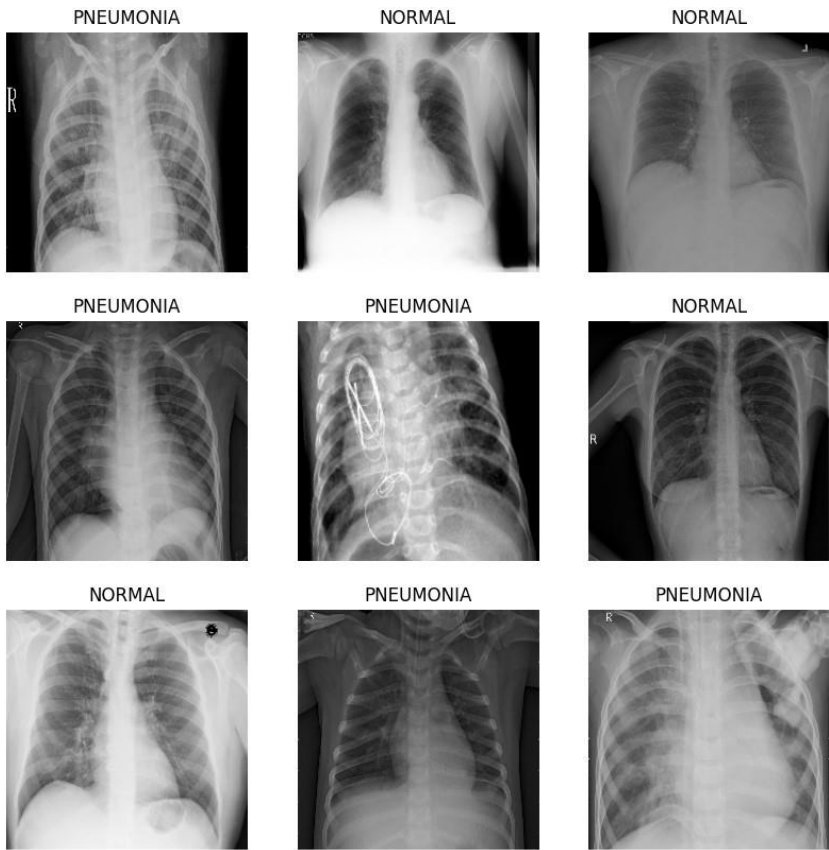


Figura 1: ejemplos de imágenes torácicas presentes en el conjunto de datos

Las imágenes presentan tamaños variados como se presenta en la siguiente tabla y en la Figura 2.

Clase	anchura mínima [pixels]	altura mínima [pixels]
Neumonía	127	384
Normal	912	496

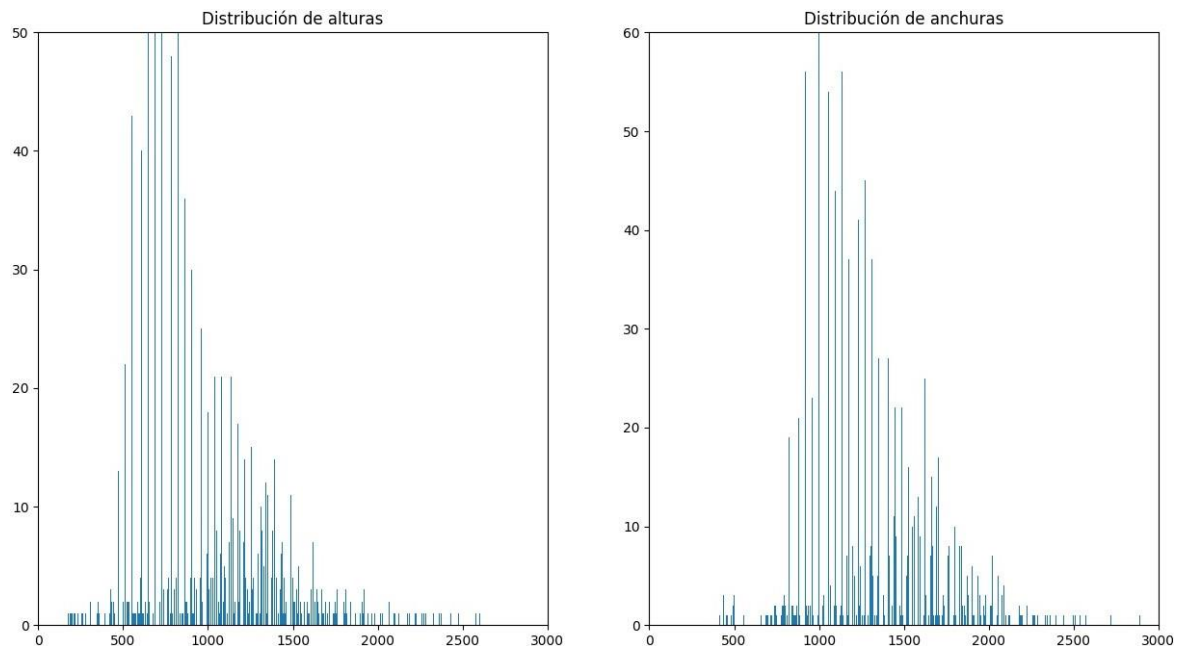


Figura 2: Distribución de las distintas alturas y anchuras de las imágenes del conjunto de datos

Preprocesamiento de los Datos

El preprocesamiento de las imágenes tiene las siguientes etapas:

- **Normalización:** Ajuste de los valores a un rango entre (0,1) para su estandarización
- **Redimensionamiento:** Ajuste del tamaño de las imágenes a 150x150 para que coincida con la entrada esperada por las CNN
- **Aumento de Datos:** Uso de técnicas como rotaciones, traslaciones, ajustes de brillo para una mayor diversidad del conjunto de datos

División de los Datos

Los datos están divididos en conjuntos y presentan una distribución mostrada en la figura 3:

- **Entrenamiento:** permite entrenar al modelo con el conjunto de datos
- **Validación:** permite ajustar al modelo con el conjunto de datos
- **Prueba:** permite evaluar el rendimiento con el conjunto de datos

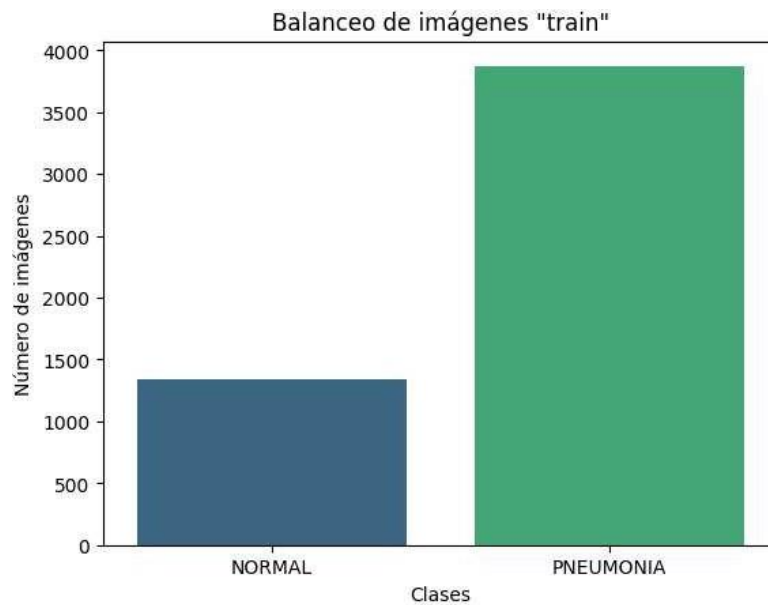


Figura 3: Proporción Neumonía-Sanos del conjunto de datos

Como se ha mencionado, los datos están divididos en conjunto de entrenamiento, validación y prueba para asegurar una evaluación robusta del modelo. Esta división permite entrenar el modelo con un conjunto de datos, ajustarlo con un conjunto de validación y evaluar su rendimiento final en un conjunto de pruebas separado. Las imágenes están compuestas por 3 canales (RGB) y debido a que los datos estaban preparados para la descarga y utilización no ha habido ningún problema para su recopilación

4. Tecnología

Los modelos de predicción han sido desarrollados utilizando el lenguaje Python y se han integrado las siguientes librerías (Fig. 4)(23):

- **TensorFlow y Keras:** Estas librerías han sido fundamentales para la construcción de las redes neuronales. Hemos utilizado tensorflow y su módulo keras para definir, entrenar y evaluar los modelos predictivos basados en redes neuronales convolucionales (CNN). tensorflow.keras proporciona un entorno robusto y flexible para diseñar arquitecturas complejas de redes neuronales.
- **Pandas:** La librería pandas ha sido crucial para el almacenado y disposición de las imágenes. Ha sido utilizado en el desarrollo del compresor para procesar las distintas variables de salida y aislar las de interés.
- **NumPy:** La librería numpy ha sido esencial para el manejo y procesamiento de datos numéricos. Se ha utilizado para manipular arrays y realizar operaciones matemáticas necesarias durante el preprocesamiento de imágenes y en el cálculo de métricas.

- **Matplotlib y Seaborn:** Para la visualización de los resultados del entrenamiento y la evaluación del modelo, hemos empleado matplotlib y su módulo pyplot. Esta herramienta ha facilitado la creación de gráficos para observar la evolución de la pérdida y precisión durante el entrenamiento. Para la elaboración de gráficos para la visualización de datos se ha utilizado la librería seaborn.
- **ImageDataGenerator:** Esta clase de tensorflow.keras.preprocessing.image ha sido utilizada para el aumento de datos y la generación de lotes de imágenes durante el entrenamiento. ImageDataGenerator permite aplicar técnicas de aumento como rotaciones, traslaciones y cambios de brillo, mejorando así la generalización del modelo.
- **RandomForestClassifier:** El módulo RandomForestClassifier de la librería sklearn se ha utilizado para la construcción de un modelo predictivo.
- **GridSearch y RepeatedKFold:** ambos módulos de la librería sklearn ha sido utilizados para encontrar los mejores parámetros del RandomForestClassifier para nuestro conjunto de datos.
- **EarlyStopping y ModelCheckpoint:** Para gestionar el proceso de entrenamiento y evitar el sobreajuste, hemos utilizado los callbacks EarlyStopping y ModelCheckpoint de tensorflow.keras.callbacks. EarlyStopping permite detener el entrenamiento cuando la mejora en la validación se estabiliza, mientras que ModelCheckpoint guarda el modelo con mejor desempeño en el conjunto de validación.
- **class_weight:** La función class_weight de sklearn.utils ha sido utilizada para ajustar el peso de las clases en el modelo, equilibrando así las predicciones en presencia de clases desbalanceadas.
- **VGG16:** Este modelo de la librería Keras de Tensorflow ha sido utilizado para la creación de un modelo predictivo por aprendizaje cruzado.
- **OneHotEncoder:** Este módulo de sklearn fue utilizado para el procesado de la variable dependiente previo a su división en el conjunto de entrenamiento y prueba.
- **Sklearn.metrics:** De este paquete se han utilizado módulos como confusion_matrix, classification_report o roc_auc_score para medir las métricas de evaluación de los modelos predictivos.

```

import pandas as pd
import numpy as np
import os
import cv2
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, utils
from keras import saving
from sklearn import metrics
import matplotlib.pyplot as plt
from keras.applications import VGG16
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import OneHotEncoder
import pathlib
from tensorflow.keras.utils import plot_model
from tensorflow.keras.layers import BatchNormalization
from collections import Counter
import seaborn as sns
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.utils import class_weight, resample
from sklearn.metrics import precision_score, recall_score, roc_auc_score, roc_curve

```

Figura 4: Librerías que se han empleado en este estudio

5. Modelización

5.1 Primera Modelización

Selección del Modelo

Con la labor de cumplir los objetivos a lo largo de este proyecto hemos adoptado distintas perspectivas. Empezando por una sencilla y buscando un modelo que haga predicciones rápidas y directas, optando por una arquitectura de redes neuronales convolucionales

Arquitectura de la Red Neuronal Convolucional

La arquitectura del modelo CNN incluye(25):

- **Capas Convolucionales:** Para extraer características locales de las imágenes.
- **Capas de Max-Pooling:** Para reducir la dimensionalidad y mejorar las extracciones de las características.

Verifica si estás usando GPU

```

print("Num GPUs encontradas: ", len(tf.config.list_physical_devices('GPU')))
if tf.test.gpu_device_name():
    print(f"GPU: {tf.test.gpu_device_name()}")
else:
    print("GPU no encontrada.")

```



```

        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),
        Flatten(),
        Dense(512, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])

```

Compilación del modelo

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Callbacks

```

early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
model_checkpoint = ModelCheckpoint('best_model.keras', monitor='val_loss',
save_best_only=True)

```

Entrenamiento del modelo

```

history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    epochs=10,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // validation_generator.batch_size,
    class_weight=class_weights,
    callbacks=[early_stopping, model_checkpoint]
)

```

Función para graficar el historial del entrenamiento

```

def graficar_historial(history):
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['loss'], label='Pérdida de Entrenamiento')
    plt.plot(history.history['val_loss'], label='Pérdida de Validación')
    plt.xlabel('Épocas')
    plt.ylabel('Pérdida')
    plt.legend()
    plt.title('Curvas de Pérdida')

    plt.subplot(1, 2, 2)
    plt.plot(history.history['accuracy'], label='Precisión de Entrenamiento')
    plt.plot(history.history['val_accuracy'], label='Precisión de Validación')
    plt.xlabel('Épocas')
    plt.ylabel('Precisión')
    plt.legend()
    plt.title('Curvas de Precisión')

    plt.show()

```

Graficar el historial del entrenamiento

```
graficar_historial(history)
```

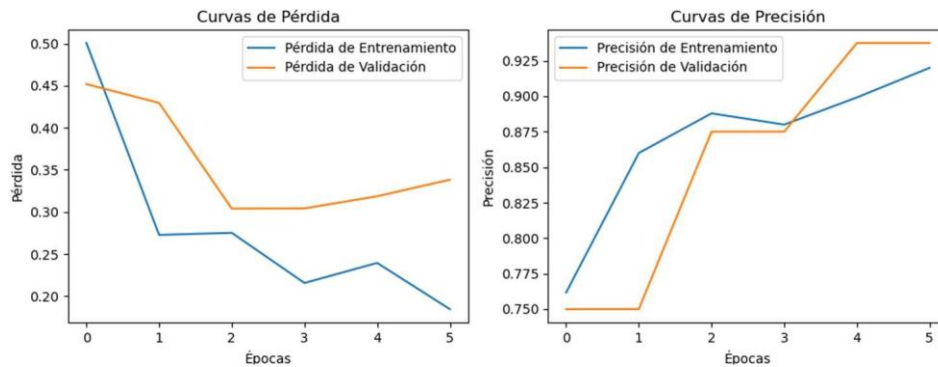


Figura 6: Curvas de aprendizaje y pérdida del primer modelo predictivo construido con una CNN

Evaluación del Modelo

El rendimiento del modelo se probó usando varias métricas:

- **Precisión:** Proporción de verdaderos positivos sobre el total de predicciones positivas.
- **Sensibilidad:** Capacidad del modelo para identificar correctamente los casos de neumonía.
- **Especificidad:** Capacidad del modelo para identificar correctamente los casos no neumonía.
- **AUC-ROC:** Área bajo la curva ROC, que proporciona una medida agregada del rendimiento en todos los posibles umbrales de clasificación.

Evaluar el modelo en el conjunto de prueba

```
test_loss, test_accuracy = model.evaluate(test_generator)
print(f'Precisión en el conjunto de prueba: {test_accuracy * 100:.2f}%')
```

Predicciones en el conjunto de prueba

```
test_steps = test_generator.samples // test_generator.batch_size
y_pred_prob = model.predict(test_generator, steps=test_steps)
y_pred = (y_pred_prob > 0.5).astype(int)
```

Obtener etiquetas verdaderas

```
y_true = test_generator.classes[:len(y_pred)]
```

Calcular métricas

```
precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
auc_roc = roc_auc_score(y_true, y_pred_prob)
```

```
print(f'Precisión: {precision:.2f}%')
```

```
print(f'Sensibilidad: {recall:.2f}%')
```

```
print(f'AUC-ROC: {auc_roc:.2f}%')
```

```
test accuracy: 80.00%
13/13 ————— 10s 771ms/step
Precisión: 61.50%
Sensibilidad: 67.18%
Especificidad: 99.57%
AUC-ROC: 0.48
```

Figura 7: Métricas de evaluación del primer modelo

Análisis de Resultados

Vamos a analizar las curvas de entrenamiento y empezamos por las de Pérdidas de Entrenamiento y de Validación.

Pérdidas de Entrenamiento y de Validación

Época 0-1:

- Entrenamiento: La pérdida disminuye rápidamente
- Validación: La pérdida disminuye poco.

El modelo está aprendiendo rápidamente en esta etapa inicial

Época 1-3:

- Entrenamiento: La pérdida disminuye hasta 0.21.
- Validación: La pérdida se ajusta a 0.30.

El descenso de la pérdida de entrenamiento sugiere que el modelo está ajustándose cada vez más a los datos, la de validación se estanca un poco lo que debe estar alcanzando un punto de equilibrio

Época 4 y sucesivas:

- Entrenamiento: La pérdida baja hasta 0.2.
- Validación: La pérdida aumenta y se mantiene.

En esta etapa, el modelo ha alcanzado un punto donde la pérdida de entrenamiento es baja y estable, pero la de validación se ha estabilizado, signo de que ha alcanzado su capacidad de generalización

Precisión de Entrenamiento y Validación

Época 0 - 1

- **Entrenamiento:** La precisión sube rápidamente
- **Validación :** La precisión se mantiene.

En las primeras épocas del entrenamiento se está ajustando el modelo a los datos de entrenamiento lo que mejora precisión,

Época 1 - 2

- **Entrenamiento:** La precisión sigue subiendo
- **Validación :** La precisión sube

El modelo sigue mejorando su precisión pero aún no ha sido capaz de visualizar patrones generales, el aumento de la precisión indica que el modelo está encontrando un equilibrio en el ajuste de los datos

Época 3 - 5

- **Entrenamiento:** La precisión sigue alta alcanzando 0.925.
- **Validación:** Sigue estando bastante alta.

En este punto el modelo está en su mejor momento de rendimiento dando alta precisión tanto en el conjunto de entrenamiento como en el de validación, esto sugiere que el modelo ha logrado un buen equilibrio entre los datos de entrenamiento y su capacidad de generalizar

```
104/104 ————— 165s 2s/step - accuracy: 0.8699 - loss: 0.3068 - val_accuracy: 0.9375 - val_loss: 0.2488
Epoch 4/10
1/104 ————— 1:22 802ms/step - accuracy: 0.8200 - loss: 0.31892024-07-24 20:18:47.131696: W tensorflow/core/framev
endezvous is aborting with status: OUT_OF_RANGE: End of sequence
[[{{node IteratorGetNext}}]]
104/104 ————— 8s 65ms/step - accuracy: 0.8200 - loss: 0.3189 - val_accuracy: 0.8750 - val_loss: 0.2284
Epoch 5/10
76/104 ————— 42s 2s/step - accuracy: 0.8804 - loss: 0.2724
```

Figura 8: Datos de precisión y pérdida época a época en el primer modelo

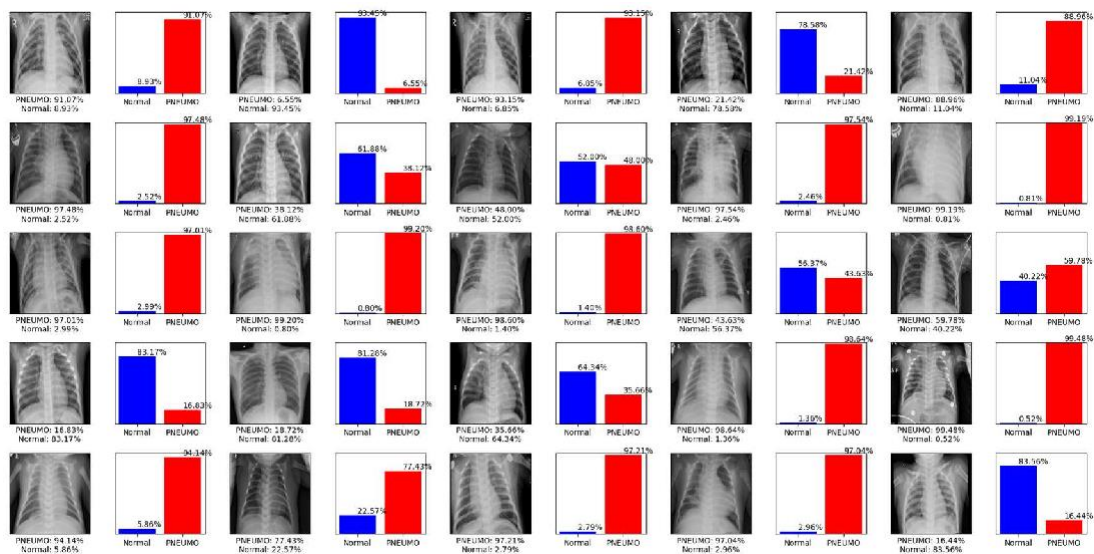


Figura 9: Ejemplo de predicciones del primer modelo

5.2 Segunda Modelización

5.2.1 Autoencoder

Sin embargo, la primera aproximación presenta una serie de limitaciones que nos hacen pensar que no sea la manera más acertada de abordar el problema que hemos expuesto con anterioridad. Las imágenes que constituyen nuestro conjunto de datos tienen cada una un tamaño distinto (Fig. 2), lo que nos obliga a llevar a cabo un reajuste del mismo para que a la hora de construir la arquitectura de nuestro modelo tenga un número de variables homogéneo. Como se puede comprobar esto se ha subsanado reajustando el tamaño de cada una de las imágenes de manera directa a unas dimensiones fijadas con

anterioridad. No obstante, esto puede provocar que perdamos una gran cantidad de información en el proceso. Esta información, puede ser importante a la hora de diagnosticar la enfermedad que padece el paciente o distinguir a los pacientes sanos de los enfermos, ya que, reducimos las dimensiones de las imágenes directamente sin ningún tipo de rigor.

Para evitar esta limitación y poder analizar la mayor cantidad de imágenes posibles conservando la mayor cantidad de información posible, hemos desarrollado unos compresores que reducen considerablemente el tamaño de las radiografías de entrada y que supone una representación fiel de la imagen original. El primero de ellos está basado en un Autoencoder:

```
class AutoEncoder2(tf.keras.Model):
    # Combina el codificador y el descodificador en un modelo para su entrenamiento.
    def __init__(
        self,
        original_dim,
        latent_dim,
        input_shape,
        name="autoencoder",
        **kwargs
    ):
    # Argumentos que recibe la clase al llamarla
        super(AutoEncoder2, self).__init__(name=name, **kwargs)
        self.latent_dim = latent_dim
        self.original_dim = original_dim
        self.input_shape = input_shape
        self.encoder = Encoder(latent_dim=latent_dim, input_shape=input_shape)
        self.decoder = Decoder(original_dim=original_dim, latent_dim=latent_dim)
    # Métodos de la clase
        def call(self, inputs):
            x = self.encoder(inputs)
            x = self.decoder(x)
            return x
    # Función para guardar la configuración de la clase
        def get_config(self):
            config = super(AutoEncoder2, self).get_config()
            config.update({
                "original_dim": self.original_dim,
                "latent_dim": self.latent_dim,
                "input_shape": self.input_shape,
            })
            return config

        def custom_fn(x):
            return x**2

        @classmethod
        def from_config(cls, config):
            return cls(**config)
```

Este está compuesto por un primer módulo o Encoder que se encarga de codificar la imagen reduciendo sus dimensiones mediante una red neuronal:

```
class Encoder(layers.Layer):
    # Definición del codificador y su red neuronal
    def __init__(self, latent_dim, input_shape, name="encoder", **kwargs):
        super(Encoder, self).__init__(name=name, **kwargs)
```

```

self.input_shape = input_shape
self.latent_dim = latent_dim
self.encoder = tf.keras.Sequential([
    tf.keras.Input(shape=input_shape, name='input'),
    layers.Flatten(),
    layers.Dense(8*latent_dim, activation='relu'),
    layers.Dense(4*self.latent_dim, activation='relu'),
    layers.Dense(2*self.latent_dim, activation='relu'),
    layers.Dense(self.latent_dim, activation='relu')])
# Métodos de la clase
    def call(self, inputs):
        x = self.encoder(inputs)
    return x
# Función para guardar la configuración de la clase
def get_config(self):
    config = super(Encoder, self).get_config()
    config.update({
        "latent_dim": self.latent_dim,
        "input_shape": self.input_shape,
    })
    return config

@classmethod
def from_config(cls, config):
    return cls(**config)

```

Luego será reconstruida por un segundo módulo o Decoder que desharrá los cálculos realizados en las primeras capas de la red neuronal hasta recuperar las dimensiones originales de nuestra imagen para luego llevar a cabo su reconstrucción:

```

class Decoder(layers.Layer):
# Definición del decodificador y su red neuronal
    def __init__(self, original_dim, latent_dim, name="decoder", **kwargs):
        super(Decoder, self).__init__(name=name, **kwargs)
        self.original_dim = original_dim
        self.latent_dim = latent_dim
        self.decoder = tf.keras.Sequential([
            layers.Dense(2*latent_dim, activation='relu'),
            layers.Dense(4*latent_dim, activation='relu'),
            layers.Dense(8*latent_dim, activation='relu'),
            layers.Dense(self.original_dim, activation='sigmoid'),
            layers.Reshape((1000,1000))]
# Métodos de la clase
    def call(self, inputs):
        x = self.decoder(inputs)
    return x
# Función para guardar la configuración de la clase
    def get_config(self):
        config = super(Decoder, self).get_config()
        config.update({
            "original_dim":self.original_dim,
            "latent_dim":self.latent_dim,
        })
        return config

@classmethod
def from_config(cls, config):
    return cls(**config)

```

No obstante, para que el compresor funcione correctamente, todas las imágenes empleadas para su entrenamiento, así como de las que se hará una predicción deben presentar todas las mismas dimensiones de entrada. Tras analizar el tamaño original de las radiografías de nuestro conjunto de datos (Fig. 2) hemos fijado esas dimensiones de entrada en 1000x1000 píxeles, ya que, es un tamaño muy similar al original para la mayoría de imágenes, y de esa manera perdemos la menor cantidad de información posible. Para el reajuste de las imágenes a las dimensiones de entrada utilizaremos el mismo método que en nuestra primera aproximación, de manera directa.

Dado el gran tamaño que tienen estas imágenes (1000x1000=1000000 de variables), hemos seleccionado unas 100 fotografías del dataset original que usaremos como muestra para entrenar el compresor y de esa forma evitar tener problemas de memoria durante su ejecución. Para medir el nivel de error del compresor a la hora de realizar los cálculos hemos utilizado la función mean squared error.

Llamamos a la clase del compresor, definimos los argumentos, compilamos el modelo y lo entrenamos

```
autoencoder = AutoEncoder2(original_dim=original_dim, latent_dim=latent_dim,
input_shape=input_shape)
autoencoder.compile(optimizer='adam', loss='mse')
autoencoder.fit(muestra,muestra,
                epochs=1,
                batch_size=10,
                validation_split=0.2)
```

Una vez hemos construido nuestro compresor, hemos cargado y comprimido nuestro dataset utilizando solo el primer módulo del mismo y guardando su salida.

```
def compress_images(path,image_list,label_list,label):
    for image in os.listdir(path):
        image = os.path.sep.join([path, image])

        if os.path.exists(image):
            # Leemos imagen
            image = cv2.imread(image)
            # Ajustamos tamaño inicial
            image = cv2.resize(image, (1000, 1000))
            image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
            image = image/255.0
            images, labels = rotate_images(image,label)
            # Comprimimos imagen
            images = autoencoder.encoder(images).numpy()
            images = images.tolist()
            image_list.extend(images)
            label_list.extend(labels)

        else:
            continue
```

Esta salida es una representación de dimensiones muy inferiores a la imagen original, presenta solo 64 variables. Sin embargo, previo a su paso y entrenamiento del modelo de predicción hemos procesado estas variables, para reducir aún más las dimensiones de estas imágenes y aislar solo a aquellas variables que tengan carácter predictivo sobre la variable dependiente.

Para procesar estas imágenes hemos utilizado una función que se encarga de calcular dos valores distintos para cada una de estas variables. Estos valores se conocen como Weight of evidence (WoE) y Information value (IV) (17). Ambos son ampliamente utilizados para problemas de regresión logística y de clasificación binaria, cómo es nuestro caso. Estos determinan la capacidad predictiva de las variables, de manera que nos permiten aislar aquellas que son de nuestro interés para construir un modelo adecuado y rechazar aquellas que nos impedirían alcanzar nuestro objetivo con normalidad.

```
def calculate_woe_ivs(df,variable,target,bins=10):

    df_woe=pd.DataFrame()

    if df[variable].nunique() > bins:
        cat_variable = pd.qcut(df[variable],bins,duplicates='drop')
        df_cat = pd.DataFrame({'cats':cat_variable,'Y':df[target]})
    else:
        df_cat = pd.DataFrame({'cats':df[variable],'Y':df[target]})

    df_woe=pd.concat([df_woe,df_cat['cats'].astype('category'),df_cat['Y']],axis=1)
    df_woe=df_woe.groupby(by=['cats'],observed=True)[['Y']].agg(['count','sum'])
    df_woe.columns = ['Total', 'Bad']
    df_woe['Good'] = df_woe['Total'] - df_woe['Bad']
    df_woe['Good_porcentaje']=df_woe['Good']/df_woe['Good'].sum()
    df_woe['Bad_porcentaje']=df_woe['Bad']/df_woe['Bad'].sum()
    df_woe['WoE']=np.log(df_woe['Good_porcentaje']/df_woe['Bad_porcentaje'])
    df_woe['IV']=df_woe['WoE']*(df_woe['Bad_porcentaje']-df_woe['Good_porcentaje'])
    woe = df_woe['WoE'].sum()
    iv = df_woe['IV'].sum()

    if (woe < 0) & (iv < 0):
        woe = -woe
        iv = -iv
    elif woe < 0:
        woe = -woe
    elif iv < 0:
        iv = -iv

    df_ivs = {'Variable': variable, 'WoE': woe, 'IV': iv}
    return df_ivs
```

Esta función nos ha permitido reducir las dimensiones de las imágenes a partir de la salida del Encoder, de 64 variables a 9, eliminando aquellas con un valor de IV de 0, las cuales son constantes y no tienen ningún valor predictivo. También hemos eliminado aquellas con un valor de IV inferior a 0,1 al tratarse de variables con una relación muy débil con la variable dependiente y, por tanto, con una capacidad predictiva muy pobre (17) (Fig. 10).

Variable	WoE	IV
39	0.754955	0.269272
23	0.729010	0.259298
13	0.614313	0.221795
19	0.597984	0.215750
50	0.519296	0.186779
22	0.491757	0.179362
27	0.393937	0.148743
3	0.360649	0.135004
37	0.323070	0.117548

Figura 10: Valores de WoE y IV para las 9 variables aisladas para construir el modelo

Una vez hemos procesado nuestro dataset, lo hemos dividido en un conjunto de train y en un conjunto de test con unas proporciones de 80, 20:

```
x_train, x_test, y_train, y_test = train_test_split(pca_df, targets, test_size=0.2, random_state=5)
```

Una vez hemos definido estos dos conjuntos hemos construido dos tipos distintos de clasificadores a fin de encontrar el mejor modelo predictivo a partir de nuestros datos. El primero de ellos consistía en una red neuronal simple, de baja complejidad:

```
inputs = keras.Input(shape=(9,), name='input_layer')
```

```
l_1 = layers.Dense(9, activation='relu')(inputs)
```

```
l_2 = layers.Dense(18, activation='relu')(l_1)
```

```
l_3 = layers.Dense(18, activation='relu')(l_2)
```

```
l_4 = layers.Dense(9, activation='relu')(l_3)
```

```
outputs = layers.Dense(1, activation='sigmoid', name='output')(l_4)
```

```
model2 = keras.Model(inputs=inputs, outputs=outputs, name='second_model')
```

Mientras que, el segundo consistía en un modelo tipo árbol el cual es muy apropiado en esta clase de problemas de clasificación binaria con múltiples variables al ser capaz de conseguir grandes resultados. Los mejores parámetros para nuestro conjunto de datos fueron seleccionados mediante validación cruzada de entre un grupo de parámetros:

```
# Conjunto de parámetros
```

```
param_grid = {'n_estimators': [50,75,100,150],
              'max_features': [3, 6, 9],
              'max_depth' : [None, 3,9,10,20]
             }
```

```
# Búsqueda por grid search con validación cruzada
```

```
grid = GridSearchCV(
    estimator = RandomForestClassifier(random_state = 123),
    param_grid = param_grid,
    scoring = 'accuracy',
    n_jobs = -1,
    cv = RepeatedKfold(n_splits=10, n_repeats=3, random_state=123),
    refit = True,
```

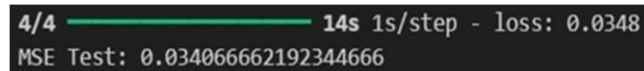
```

        verbose = 0,
        return_train_score = True
    )
    grid.fit(X = x_train, y = y_train)
    # Definición del clasificador
    RF = RandomForestClassifier(max_features=9, n_estimators=150, random_state=123)
    RF.fit(x_train, y_train)
    acierto_entrenamiento = RF.score(x_train, y_train)
    acierto_test = RF.score(x_test, y_test)

```

Evaluación del modelo

Tras el entrenamiento del compresor con los datos de muestra obtuvimos un valor de error de 0.05 (Fig. 11).



```

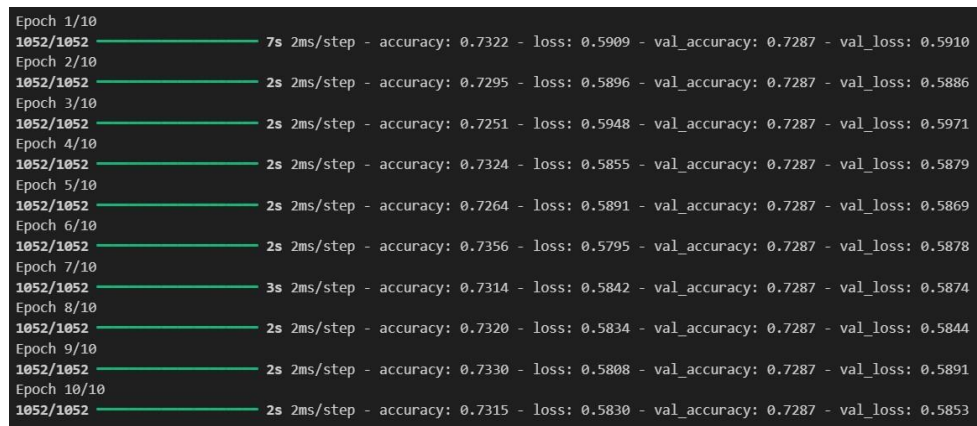
4/4 ————— 14s 1s/step - loss: 0.0348
MSE Test: 0.034066662192344666

```

Figura 11: Error del compresor calculado con la función mean squared error

Una vez leímos todas las imágenes de nuestro conjunto de datos con el compresor y definimos los conjuntos de entrenamiento y prueba corrimos el primer modelo de predicción basado en una red neuronal obteniendo los siguientes resultados (Fig. 12).

a)



```

Epoch 1/10
1052/1052 ————— 7s 2ms/step - accuracy: 0.7322 - loss: 0.5909 - val_accuracy: 0.7287 - val_loss: 0.5910
Epoch 2/10
1052/1052 ————— 2s 2ms/step - accuracy: 0.7295 - loss: 0.5896 - val_accuracy: 0.7287 - val_loss: 0.5886
Epoch 3/10
1052/1052 ————— 2s 2ms/step - accuracy: 0.7251 - loss: 0.5948 - val_accuracy: 0.7287 - val_loss: 0.5971
Epoch 4/10
1052/1052 ————— 2s 2ms/step - accuracy: 0.7324 - loss: 0.5855 - val_accuracy: 0.7287 - val_loss: 0.5879
Epoch 5/10
1052/1052 ————— 2s 2ms/step - accuracy: 0.7264 - loss: 0.5891 - val_accuracy: 0.7287 - val_loss: 0.5869
Epoch 6/10
1052/1052 ————— 2s 2ms/step - accuracy: 0.7356 - loss: 0.5795 - val_accuracy: 0.7287 - val_loss: 0.5878
Epoch 7/10
1052/1052 ————— 3s 2ms/step - accuracy: 0.7314 - loss: 0.5842 - val_accuracy: 0.7287 - val_loss: 0.5874
Epoch 8/10
1052/1052 ————— 2s 2ms/step - accuracy: 0.7320 - loss: 0.5834 - val_accuracy: 0.7287 - val_loss: 0.5844
Epoch 9/10
1052/1052 ————— 2s 2ms/step - accuracy: 0.7330 - loss: 0.5808 - val_accuracy: 0.7287 - val_loss: 0.5891
Epoch 10/10
1052/1052 ————— 2s 2ms/step - accuracy: 0.7315 - loss: 0.5830 - val_accuracy: 0.7287 - val_loss: 0.5853

```

b)

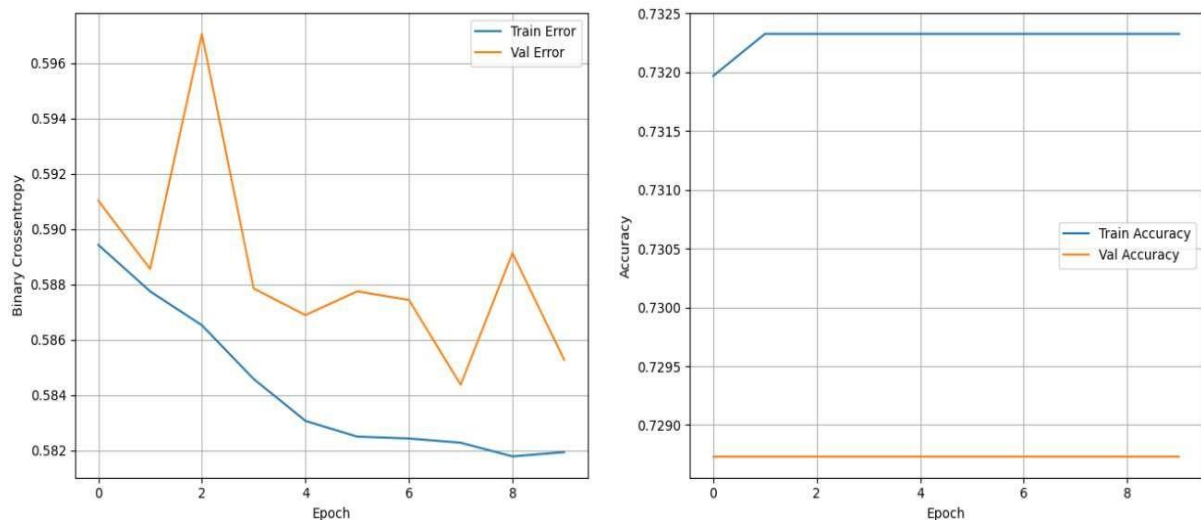


Figura 12: a): resultados época por época del entrenamiento de la red neuronal. b): curvas de aprendizaje y pérdida de la red neuronal época a época

El modelo era incapaz de aprender a partir de las variables que representaban cada imagen haciendo predicciones aleatorias en cada una de las épocas de entrenamiento como refleja el valor de probabilidad de salida: 0,7287; el cual se corresponde con la proporción de radiografías sanas y con neumonía del conjunto de datos original (Fig. 3).

Por el contrario, con el clasificador de random forest obtuvimos resultados muy distintos. Con los parámetros: max_features=9, n_estimators=150 y random_state=123; el clasificador conseguía una precisión del 99,99% en el conjunto de entrenamiento. Sin embargo, en el conjunto de prueba conseguimos una precisión bastante pobre, del 77,76% (Fig. 9, a), por tanto, tenemos un problema de sobreajuste. Para el área bajo la curva en este modelo obtuvimos un valor de 0.66 (Fig. 13, b).

a)

```

Acierto en el conjunto de entrenamiento: 99.99%
Acierto en el conjunto de test: 77.76%

```

b)

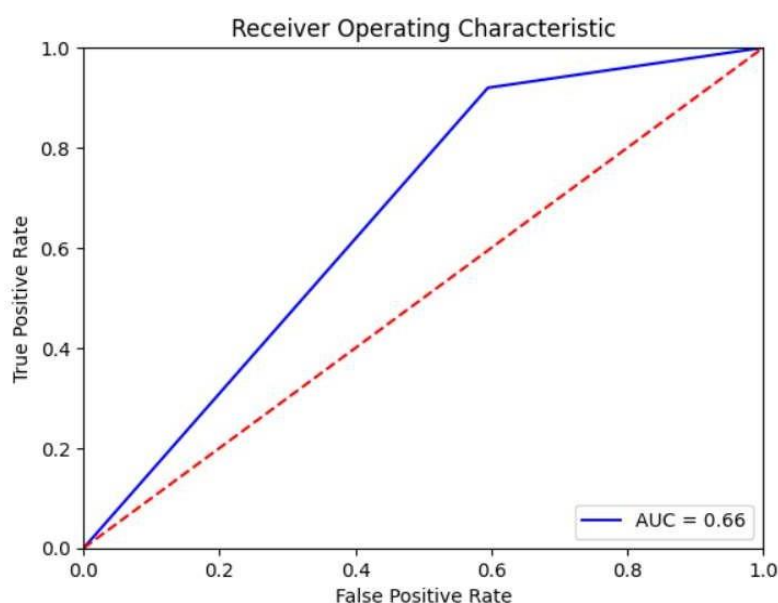


Figura 13: a): precisión del modelo random forest en los conjuntos de entrenamiento y prueba. b): área bajo la curva del modelo random forest

No obstante, la matriz de confusión muestra que para el conjunto de pruebas nuestro clasificador de random forest obtiene buenos resultados a la hora de clasificar las radiografías con neumonía (etiqueta=1) (Fig. 14) con una precisión de 0.80 y una recuperación de 0.92.

[[523 768]					
[271 3110]]					
	precision	recall	f1-score	support	
0.0	0.66	0.41	0.50	1291	
1.0	0.80	0.92	0.86	3381	
accuracy			0.78	4672	
macro avg	0.73	0.66	0.68	4672	
weighted avg	0.76	0.78	0.76	4672	

Figura 14: matriz de confusión para el modelo de random forest

5.2.2 Compresor con CNN

Esta modelización es una red neuronal convolucional (CNN) basada en la arquitectura U-Net (Fig. 15). Esta arquitectura se diseñó principalmente para tareas de segmentación de imágenes especialmente en el campo de la visión artificial. Fue desarrollada por Olaf Ronneberger, Philipp Fischer y Thomas Brox en 2015, inicialmente para segmentación de imágenes médicas (18).

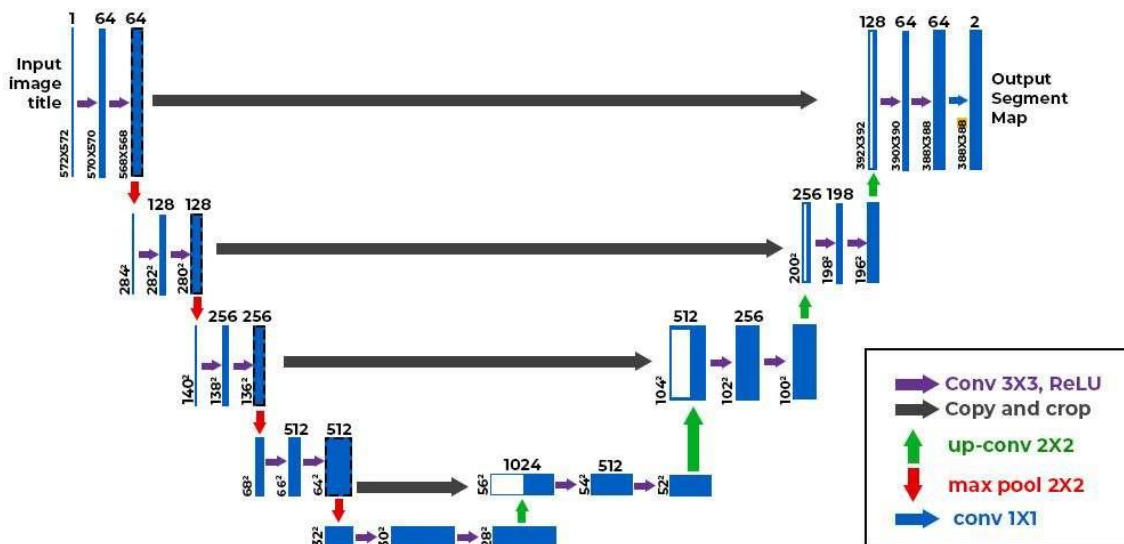


Figura 15: Arquitectura U-Net

1. **Arquitectura en forma de U:** La red tiene una estructura simétrica que se asemeja a una “U”. Esta arquitectura consta de dos partes principales:
 - **Encoder:** Aquí, la imagen de entrada se reduce progresivamente mediante capas de convolución y pooling, extrayendo características importantes.
 - **Decoder:** En esta parte, la resolución de la imagen se incrementa mediante capas de convolución transpuesta, combinando la información de características extraídas con las características locales de la imagen original.
2. **Skip connections:** Estas conexiones unen las capas correspondientes del encoder y el decoder, permitiendo que la red recupere detalles perdidos durante la contracción y mejorando la precisión de la segmentación.
3. **Segmentación semántica:** U-Net es especialmente eficaz para la segmentación semántica, donde cada píxel de una imagen se clasifica en una categoría específica. Esto es crucial en aplicaciones como la segmentación de tejidos en imágenes médicas o la identificación de objetos en imágenes de vehículos autónomos.

Para esta modelización se ha empleado “media” arquitectura U-Net (Fig. 16) junto con una capa flatten y tres capas densas, la última de las cuales arroja la predicción de las categorías del dataset (Fig. 17). Al utilizar la arquitectura U-Net se obtiene una entrada a las capas densas de las imágenes comprimidas con diferentes filtros, lo cual permite identificar a la red de una manera más consistente los patrones de la enfermedad.

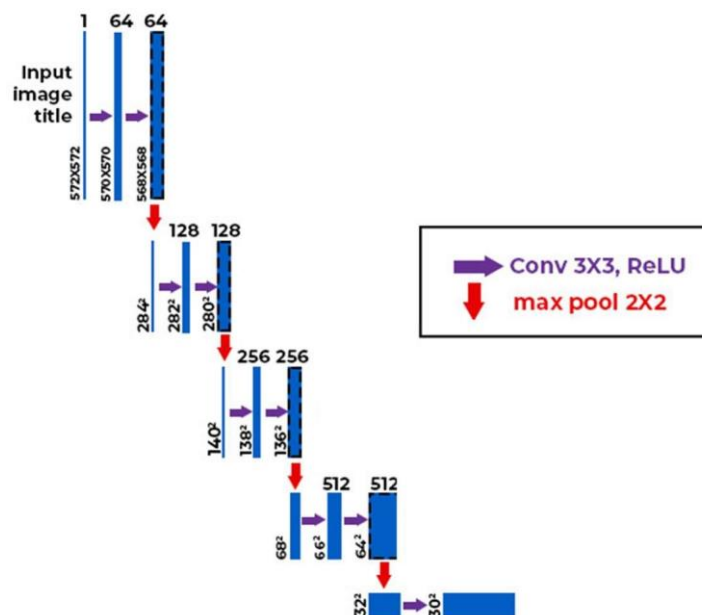


Figura 16: Fragmento de arquitectura U-Net empleado en el modelo

Layer (type)	Output Shape	Param #
=====		
U-Net (Sequential)	(None, None, None, 192)	223992
flatten_3 (Flatten)	(None, 37632)	0
dense_9 (Dense)	(None, 128)	4817024
dropout_6 (Dropout)	(None, 128)	0
dense_10 (Dense)	(None, 64)	8256
dropout_7 (Dropout)	(None, 64)	0
dense_11 (Dense)	(None, 1)	65
=====		
Total params: 5049337 (19.26 MB)		
Trainable params: 5047849 (19.26 MB)		
Non-trainable params: 1488 (5.81 KB)		

Figura 17: Resumen del tercer modelo con el compresor y la red neuronal para hacer predicciones

En cuanto al entrenamiento del modelo, los datos se han dividido en conjuntos de entrenamiento y test utilizando el método `train_test_split` de 0.2. Se ha utilizado la métrica de accuracy para evaluar el rendimiento del modelo y el entrenamiento se llevó a cabo durante 15 epoch con un tamaño de batch de 12. Además, se ha implementado

la técnica de early stopping con un criterio de patience de 5 epoch para evitar el sobreajuste.

#Características del aumento de datos y normalización

```
datagen = ImageDataGenerator(  
    rotation_range=30, # Rotación aleatoria  
    width_shift_range=0.1, # Desplazamiento horizontal  
    height_shift_range=0.1, # Desplazamiento vertical  
    zoom_range=0.2, # Zoom aleatorio  
    horizontal_flip=True, # Volteo horizontal  
    fill_mode='nearest', # Modo de relleno)
```

```
datanorm = ImageDataGenerator(  
    rescale=1./255, # Solo normalización)
```

#Formulación para el aumento de datos

```
def augmentation(data_im, data_lb, name):  
    all_arrays = []  
    all_labels = []  
    BATCH_SIZE_im = 32  
    im2=datagen.flow(data_im, y=data_lb, batch_size=BATCH_SIZE_im, seed=12)  
  
    for i in range(len(data_im)//BATCH_SIZE_im):  
        new_array, new_label = next(im2)  
        all_arrays.append(new_array)  
        all_labels.append(new_label)  
        print(f'{i+1} of {len(data_im)//BATCH_SIZE_im} of {name}')  
  
    all_arrays = np.concatenate(all_arrays, axis=0)  
    all_labels = np.concatenate(all_labels, axis=0)  
  
    bal_im_data = np.concatenate([data_im, all_arrays], axis=0)  
    bal_lb_data = np.concatenate([data_lb, all_labels], axis=0)  
  
    return bal_im_data, bal_lb_data
```

#Formulación para la normalización de datos

```
def normalitation(data_im, data_lb, name):  
    all_arrays = []  
    all_labels = []  
    BATCH_SIZE_im = 64  
    im2=datanorm.flow(data_im, y=data_lb, batch_size=BATCH_SIZE_im, seed=12)  
  
    for i in range(len(data_im)//BATCH_SIZE_im):  
        new_array, new_label = next(im2)  
        all_arrays.append(new_array)  
        all_labels.append(new_label)  
        print(f'{i+1} of {len(data_im)//BATCH_SIZE_im} of {name}')  
  
    bal_data_im = np.concatenate(all_arrays, axis=0)  
    bal_data_lb = np.concatenate(all_labels, axis=0)  
    return bal_data_im, bal_data_lb
```

#Definición de U-Net

```
import tensorflow as tf  
from tensorflow.keras.layers import Conv2D, BatchNormalization, ReLU, MaxPooling2D, Dropout
```

```

def Conv_3_k(in_channels, out_channels):
    model = tf.keras.Sequential()
    model.add(Conv2D(out_channels, kernel_size=3, strides=1, padding='same', input_shape=(None,
None, in_channels)))
    return model

def Double_Conv(in_channels, out_channels):
    model = tf.keras.Sequential()
    model.add(Conv_3_k(in_channels, out_channels))
    model.add(BatchNormalization())
    model.add(ReLU())
    model.add(BatchNormalization())
    model.add(ReLU())
    return model

def Down_Conv(in_channels, out_channels):
    model = tf.keras.Sequential()
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Double_Conv(in_channels, out_channels))
    return model

def middle_UNET(in_channels, out_channels):
    model = tf.keras.Sequential()
    model.add(Double_Conv(in_channels, out_channels))
    model.add(Down_Conv(out_channels, out_channels*2))
    model.add(Down_Conv(out_channels*2, out_channels*4))
    model.add(Down_Conv(out_channels*4, out_channels*8))
    model.add(Down_Conv(out_channels*8, out_channels*16))
    return model

```

#Definición del modelo

```

model = tf.keras.Sequential()

model.add(Input(shape=IMAGE_SIZE + (3, )))

#compresor
model.add(middle_UNET(3,12))
print(model.output_shape)

#Capas completamente conectadas
model.add(Flatten())
print(model.output_shape)
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
print(model.output_shape)
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu'))
model.add(Dense(10, activation='relu'))

#Capa de salida
model.add(Dense(1, activation='softmax'))
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss='binary_crossentropy',
    metrics=['accuracy']
)

model.summary()

```

#Definición de CallBacks

```
es_callback = keras.callbacks.EarlyStopping(
    monitor='val_loss', # can be 'val_accuracy'
    patience=5, # if during 5 epochs there is no improvement in `val_loss`, the execution will stop
    verbose=1)
mc_callback = keras.callbacks.ModelCheckpoint(
    'best_model.keras',
    monitor='val_loss',
    save_best_only=True)
```

#Entrenamiento del modelo

```
epochs = 15
BATCH_SIZE = 32
history = model.fit(
    x=x_train,
    y=y_train,
    validation_split=0.1,
    batch_size=BATCH_SIZE,
    epochs=epochs,
    callbacks=[es_callback, mc_callback]
)
show_loss_accuracy_evolution(history)
```

#Graficar el histórico de precisión

```
def show_loss_accuracy_evolution(history):

    hist = pd.DataFrame(history.history)
    hist['epoch'] = history.epoch

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))

    ax1.set_xlabel('Epoch')
    ax1.set_ylabel('Sparse Categorical Crossentropy')
    ax1.plot(hist['epoch'], hist['loss'], label='Train Error')
    ax1.plot(hist['epoch'], hist['val_loss'], label = 'Val Error')
    ax1.grid()
    ax1.legend()

    ax2.set_xlabel('Epoch')
    ax2.set_ylabel('Accuracy')
    ax2.plot(hist['epoch'], hist['accuracy'], label='Train Accuracy')
    ax2.plot(hist['epoch'], hist['val_accuracy'], label = 'Val Accuracy')
    ax2.grid()
    ax2.legend()
    plt.show()
```

#Mostrar los errores del modelo

```
def show_errors(val_ds, model, class_names_list, n_images=10):
    n_plots = 0
    for images, labels in val_ds:
        pred_probs = model.predict(images)
        preds = pred_probs.argmax(axis=-1)
        bad_pred_inds = np.where(preds != labels)[0]
        for ind in list(bad_pred_inds):
            n_plots += 1
            real_class = class_names_list[labels[ind].numpy()]
```

```

pred_class = class_names_list[preds[ind]]
prob = pred_probs[ind].max()
prob_real = pred_probs[ind][[
    i for i, c in enumerate(class_names_list) if c == real_class
]][0]
plt.imshow(images[ind].numpy().astype("uint8"))
plt.title(
    'Predicted: {0}, prob: {1:.2f} \n real: {2}, prob: {3:.2f}'
    .format(pred_class, prob, real_class, prob_real))
plt.show()

if n_plots == n_images:
    return
return

```

5.3 Tercera Modelización

La tercera y última aproximación que hemos adoptado en este proyecto es el empleo de un modelo entrenado anteriormente con un conjunto de datos distinto al nuestro en un método que se conoce como transfer learning o aprendizaje cruzado. El modelo que hemos elegido es el modelo VGG16 (19) el cual fue creado y entrenado para la clasificación de imágenes, una tarea parecida a la que llevamos a cabo en este proyecto.

Primero de todo, definimos el modelo VGG16, especificando que sus parámetros no son entrenables. A continuación, acoplamos a este modelo nuestra propia arquitectura que se encargará de recibir su salida y hacer una predicción en base a ella. Este nuevo elemento consiste en una red neuronal que se encarga de leer e interpretar los cálculos realizados por la CNN previa y hacer una predicción en base a ellos. Al ser un método de aprendizaje cruzado, la única parte que será entrenada utilizando nuestro conjunto de datos será la que hemos añadido en último lugar.

```

def pretrain_model(model, shape):
    pretrained_model = model(include_top=False, input_tensor=shape,
                             weights='imagenet')
    pretrained_model.trainable=False
    return pretrained_model

# Llamamos a la función y definimos el modelo
pretrained_model=pretrain_model(VGG16,tf.keras.Input(shape=(150,150,3)))
# Construimos el modelo predictivo incluyendo el modelo entrenado con anterioridad
inputs = tf.keras.Input(shape=(150,150,3), name='input')

x = pretrained_model(inputs)

flat = layers.Flatten(name='flatten')(x)
dense = layers.Dense(64, activation='relu', name='dense')(flat)
drop = layers.Dropout(0.5, seed=30, name='dropout')(dense)
dense = layers.Dense(32, activation='relu', name='dense1')(drop)
outputs = layers.Dense(1, activation='sigmoid', name='output')(dense)

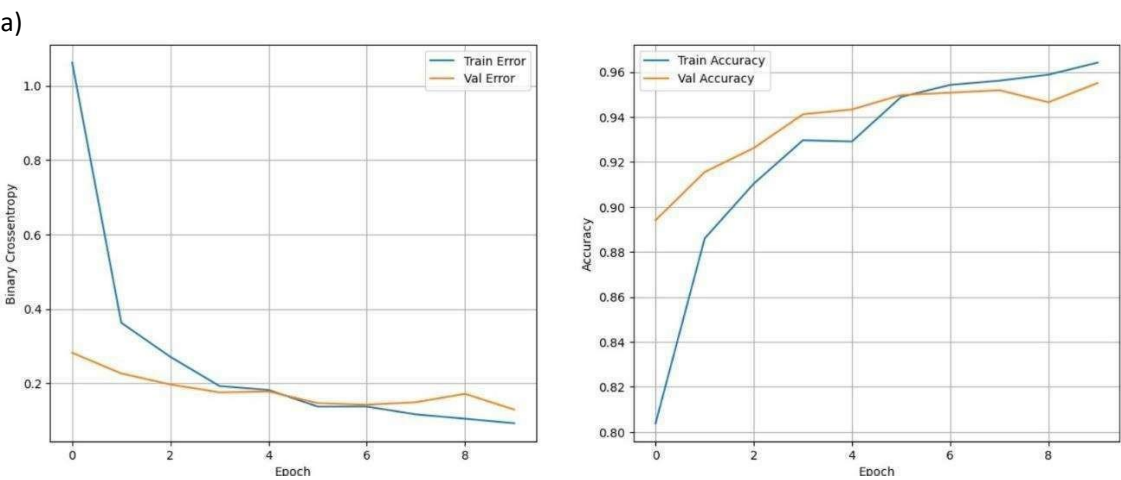
model3 = keras.Model(inputs=inputs, outputs=outputs, name='Third_model')

```

Evaluación del modelo

En el modelo desarrollado mediante aprendizaje cruzado obtuvimos unos resultados muy prometedores. Como demuestran las curvas de aprendizaje y pérdida del modelo,

este aprende a medida que avanzan el número de épocas, aumentando el valor de precisión y disminuyendo el de pérdida tanto en el conjunto de entrenamiento como en el de validación (Fig. 18, a).



b)

```
[[268 33]
 [ 29 838]]
```

	precision	recall	f1-score	support
0.0	0.90	0.89	0.90	301
1.0	0.96	0.97	0.96	867
accuracy			0.95	1168
macro avg	0.93	0.93	0.93	1168
weighted avg	0.95	0.95	0.95	1168

c)

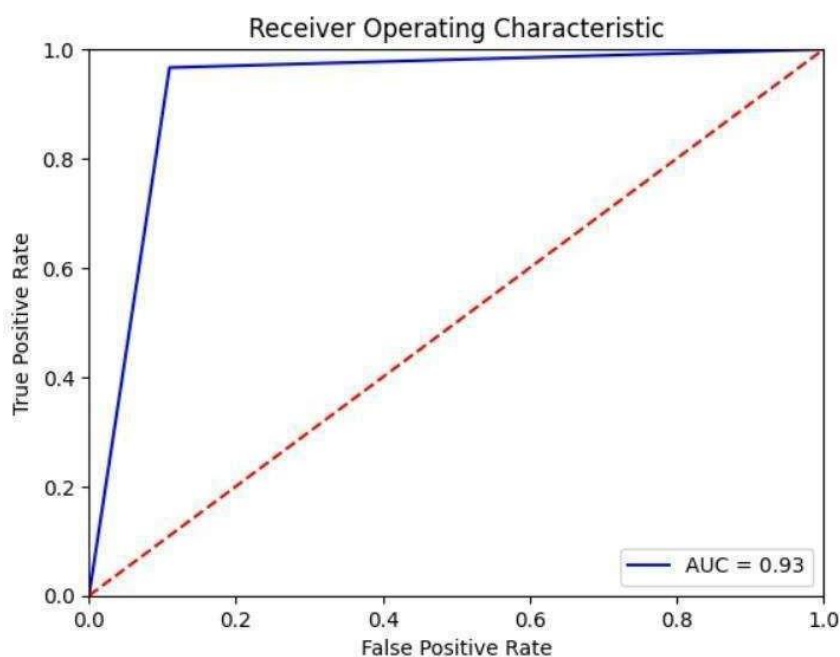


Figura 18: a): curvas de aprendizaje y pérdida del modelo de aprendizaje cruzado. b): matriz de confusión del modelo de aprendizaje cruzado. c) área bajo la curva del modelo de aprendizaje cruzado.

Además, este modelo presenta unos resultados muy buenos en el resto métricas empleadas para evaluar su comportamiento. En general presenta una precisión de 0.95 (Fig. 18, b), siendo particularmente bueno prediciendo aquellas radiografías con diagnóstico de neumonía con una precisión de 0.96 y una recuperación de 0.97 (Fig 18, b). No así con las radiografías etiquetadas como sanas en el conjunto de pruebas con una precisión de 0.90 y una recuperación de 0.89 (Fig. 18, b). En cuanto al área bajo la curva este modelo ha conseguido un valor de 0.93 convirtiéndolo en el mejor modelo predictivo en este estudio (Fig. 18, c).

6. Resultados

Beneficios Económicos y Sociales

1. Reducción de Costos:

- La automatización del diagnóstico puede reducir significativamente los costos asociados con pruebas y diagnósticos tradicionales, como las PCR, disminuyendo el uso de recursos pudiendo emplearlos en la investigación y mejora de esta clase de tecnologías.
- La implementación en la nube reduce la necesidad de infraestructura local costosa y permite escalar el servicio según la demanda sin la necesidad de un despliegue tecnológico masivo y costoso.

2. Mejora en la Salud Pública:

- La detección temprana y precisa de neumonía puede contribuir a un mejor manejo y control de la patología, reduciendo su propagación, evitando colapsos en hospitales y centros de salud, permitiendo la actuación sobre otras enfermedades de importancia.

7. Despliegue Tecnológico

Infraestructura de Despliegue

En cuanto al despliegue del modelo de predicción de neumonía utilizando radiografías torácicas, es fundamental elegir una infraestructura adecuada que asegure su disponibilidad, escalabilidad, así como la seguridad del sistema. Con este fin, proponemos el uso de la nube debido a las grandes ventajas que esta proporciona, facilitando el acceso global a los distintos modelos desarrollados en este estudio y su integración con otros sistemas de salud. A continuación, se detallan los componentes clave del despliegue:

1. Plataforma en la Nube:

- **Google Cloud Platform (GCP):** Hemos elegido esta opción por su gran cantidad de servicios en IA y almacenamiento. Entre otras cosas permite una implementación rápida y eficiente del modelo, aprovechando Google Kubernetes y Cloud Storage para el almacenamiento y manejo de las imágenes de un modo intuitivo y eficiente.

2. Contenedores y Orquestación:

- **Docker:** Crearemos una imagen Docker que contenga el modelo entrenado, así como todas las librerías necesarias (TensorFlow, Keras, etc.) facilitando su portabilidad. Del mismo modo, también permite la reproducción del modelo en local permitiendo aportaciones por parte de gente externa al estudio que consideren puedan mejorar los resultados obtenidos.
- **Kubernetes:** Utilizaremos Kubernetes para gestionar el despliegue de contenedores en clústeres. Kubernetes también permitirá la gestión eficiente de los recursos y la carga de trabajo.

3. Interfaz de Usuario y API:

- **Flask:** Desarrollaremos una API REST utilizando Flask, que permitirá la integración del modelo a través de solicitudes HTTP. De esta manera, el usuario podrá obtener una predicción de una radiografía de su interés rápida y en el momento subiendo su imagen a la API la cual se encargará de utilizar el modelo previamente cargado y devolver la predicción al usuario.
- **Interfaz Web:** Para facilitar el uso por parte de profesionales de la salud, como radiólogos o médicos, desarrollaremos una interfaz web sencilla donde los usuarios puedan cargar radiografías de manera intuitiva y muy ilustrativa para recibir las predicciones en el momento. La interfaz web será desarrollada utilizando tecnologías como HTML, CSS y JavaScript, integrando la API desarrollada con Flask.

Proceso de Despliegue

El despliegue del modelo incluirá las siguientes etapas:

1. **Preparación del Entorno:** Esta incluiría la configuración del entorno de desarrollo y pruebas para luego proceder con el entrenamiento final del modelo y la verificación de su rendimiento.
2. **Despliegue en la Nube:** El primer paso sería la creación de un clúster de Kubernetes, cuya función hemos expuesto anteriormente. Seguiríamos con el despliegue de la imagen Docker para terminar con la configuración de distintos parámetros como los balanceadores de carga y el escalado automático con el fin de gestionar el tráfico y la demanda.
3. **Pruebas y Validación:** A fin de cerciorarnos de que todo funciona correctamente, realizaríamos unas pruebas que validen el rendimiento y precisión del modelo en el entorno donde será explotado y que hemos creado anteriormente.
4. **Mantenimiento y Actualización:** Estableceríamos un plan de mantenimiento para asegurar que el sistema reciba actualizaciones constantes con el fin de evitar cualquier clase de problema que evite el correcto funcionamiento del entorno.

8. Puesta en Valor

Impacto en el Diagnóstico Médico

Los modelos desarrollados tienen potencial para mejorar el diagnóstico de neumonía de las siguientes maneras:

1. **Diagnóstico Rápido y Preciso:**
 - La utilización de distintas técnicas de IA para analizar radiografías torácicas permite obtener resultados de diagnóstico en cuestión de segundos cubriendo el mayor número de supuestos posibles, proporcionando una predicción lo más generalista posible.
 - El modelo proporciona a profesionales y expertos una herramienta confiable y eficiente para identificar casos de neumonía.
2. **Apoyo a Profesionales de la Salud:**
 - El modelo puede servir como una segunda opinión automatizada teniendo en cuenta que siempre es recomendable su verificación por parte de un experto, reduciendo la carga de trabajo de radiólogos y médicos permitiendo un enfoque más eficiente y rápido que permita ganar tiempo para las siguientes etapas de actuación sobre la enfermedad.
 - Ayuda a estandarizar el diagnóstico, minimizando la variabilidad entre diferentes profesionales y mejorando la consistencia en los resultados, lo cual hemos logrado construyendo un modelo robusto y generalista.
3. **Acceso en Zonas Remotas:**
 - Su implementación en la nube permite que el sistema sea accesible desde cualquier lugar con conexión a internet, haciéndolo accesible en aquellas

zonas donde el equipamiento médico es escaso para realizar diagnósticos precisos.

Futuras Líneas de Investigación

El éxito de esta investigación ha sentado las bases para futuras investigaciones enfocadas en la mejora de esta clase de algoritmos:

1. Ampliación del Conjunto de Datos:

- Recopilación de más datos de diversas fuentes y países para mejorar la generalización y robustez del modelo, así como encontrar más patrones en las imágenes que ayuden a realizar predicciones con más precisión.
- Ampliación con imágenes de diferentes tipos de patologías pulmonares con el fin entrenar un modelo multiclase que pueda diferenciar entre varias enfermedades ampliando el espectro de acción del programa y cubriendo una mayor población.

2. Optimización del Modelo:

- Implementación de técnicas de reducción de dimensionalidad más potentes, como pueden ser Autoencoders de una mayor complejidad que cuenten con un mayor número de capas densas que permitan una mejor representación de las imágenes originales.

3. Integración con Sistemas de Salud:

- Desarrollo de soluciones integradas que permitan la conexión del modelo con los sistemas de información hospitalarios (HIS) y los registros electrónicos de salud (EHR) para incluir la predicción en el registro médico del paciente en el momento.
- Implementación de flujos de trabajo automatizados que integren el análisis de imágenes como apoyo junto con otros datos clínicos extraídos a partir de otras pruebas para proporcionar diagnósticos más completos y precisos.

4. Evaluación Clínica y Validación:

- Realización de estudios clínicos a gran escala con pacientes reales en múltiples zonas en el mundo para validar el rendimiento y evaluar su eficacia y alcance en un entorno médico real.
- Colaboración con instituciones médicas y académicas para llevar a cabo investigaciones conjuntas que mejoren continuamente la tecnología basada en los comentarios y resultados obtenidos, así como coordinar su validación en ambientes reales con constante intercambio de resultados.

9. Conclusiones, Observaciones y Contribuciones

Conclusiones

Los resultados obtenidos en este estudio nos han permitido obtener las siguientes conclusiones:

- Las CNN tienen un gran potencial para mejorar el diagnóstico de neumonía usando radiografías torácicas. Los resultados obtenidos sugieren que el modelo puede ser una herramienta potente en la práctica hospitalaria. Sin embargo, es posible que sea necesario construir un modelo más complejo, con un mayor número de capas intermedias que sean capaces de recopilar la información en cada imagen de una forma más eficiente para que el modelo encuentre patrones para hacer una predicción más precisa.
- Los compresores como los Autoencoders y las técnicas de reducción de dimensionalidad son unas herramientas muy interesantes para poder incluir una mayor cantidad de imágenes en el modelo y poder construir un modelo generalista y robusto. No obstante, las limitaciones tecnológicas que nos hemos encontrado han impedido que desarrollemos algoritmos más complejos que hagan representaciones fieles de las imágenes originales que puedan aprovechar los modelos para hacer mejores predicciones.
- Ante la falta de la potencia computacional necesaria, y la creación de un modelo con CNN desde cero con buenos resultados, la construcción de un modelo predictivo por aprendizaje cruzado sigue siendo el mejor método para conseguir un algoritmo que haga predicciones eficaces y generalistas con una alta precisión cómo ha sido demostrado en estudios anteriores, incluso con otro tipo de enfermedades (20, 21).

Observaciones

Durante el desarrollo de este proyecto, observamos que la calidad del modelo está sujeta a la calidad y magnitud del dataset. La incorporación de mejores técnicas de preprocesamiento y aumento de datos nos daría una mayor precisión y robustez.

Contribuciones

Introducción: Alejandro y Álvaro

Objetivos: Alejandro, Álvaro e Ismael

Datos: Álvaro e Ismael

Tecnología: Alejandro y Álvaro

Modelización: Alejandro, Álvaro e Ismael

Resultados: Alejandro y Álvaro

Despliegue Tecnológico: Alejandro, Álvaro e Ismael

Puesta en valor: Alejandro, Álvaro e Ismael

Conclusiones: Alejandro y Álvaro

10. Bibliografía

1. Hespanhol V, Bárbara C. Pneumonia mortality, comorbidities matter? *Pulmonology*. 2020 May-Jun;26(3):123-129. doi: 10.1016/j.pulmoe.2019.10.003. Epub 2019 Nov 29. PMID: 31787563.
2. Purohit D, Ahirwar AK, Sakarde A, Asia P, Gopal N. COVID-19 and lung pathologies. *Horm Mol Biol Clin Investig*. 2021 Aug 2;42(4):435-443. doi: 10.1515/hmbci-2020-0096. PMID: 34333882.
3. National Academies of Sciences, Engineering, and Medicine; Health and Medicine Division; Board on Health Care Services; Committee on Identifying New or Improved Diagnostic or Evaluative Techniques. *Advances in the Diagnosis and Evaluation of Disabling Physical Health Conditions*. Washington (DC): National Academies Press (US); 2023 May 1. 6, Techniques for Respiratory Disorders. Available from:
4. Emara HM, Shoaib MR, Elwekeil M, El-Shafai W, Taha TE, El-Fishawy AS, El-Rabaie EM, Alshebeili SA, Dessouky MI, Abd El-Samie FE. Deep convolutional neural networks for COVID-19 automatic diagnosis. *Microsc Res Tech*. 2021 Nov;84(11):2504-2516. doi: 10.1002/jemt.23713. Epub 2021 Jun 14. PMID: 34121273; PMCID: PMC8420362.
5. An Q, Rahman S, Zhou J, Kang JJ. A Comprehensive Review on Machine Learning in Healthcare Industry: Classification, Restrictions, Opportunities and Challenges. *Sensors (Basel)*. 2023 Apr 22;23(9):4178. doi: 10.3390/s23094178. PMID: 37177382; PMCID: PMC10180678.
6. Wagner MW, Namdar K, Biswas A, Monah S, Khalvati F, Ertl-Wagner BB. Radiomics, machine learning, and artificial intelligence-what the neuroradiologist needs to know. *Neuroradiology*. 2021 Dec;63(12):1957-1967. doi: 10.1007/s00234-021-02813-9. Epub 2021 Sep 18. PMID: 34537858; PMCID: PMC8449698.
7. Jan, M., Spangaro, A., Lenartowicz, M., & Mattiazzi Usaj, M. (2024). From pixels to insights: Machine learning and deep learning for bioimage analysis. *BioEssays*, 46, e2300114. <https://doi.org/10.1002/bies.202300114>
8. Lee JG, Jun S, Cho YW, Lee H, Kim GB, Seo JB, Kim N. Deep Learning in Medical Imaging: General Overview. *Korean J Radiol*. 2017 Jul-Aug;18(4):570-584. doi: 10.3348/kjr.2017.18.4.570. Epub 2017 May 19. PMID: 28670152; PMCID: PMC5447633.
9. Mazurowski MA, Buda M, Saha A, Bashir MR. Deep learning in radiology: An overview of the concepts and a survey of the state of the art with focus on MRI. *J Magn Reson Imaging*. 2019 Apr;49(4):939-954. doi: 10.1002/jmri.26534. Epub 2018 Dec 21. PMID: 30575178; PMCID: PMC6483404.

10. Mann M, Badoni RP, Soni H, Al-Shehri M, Kaushik AC, Wei DQ. Utilization of Deep Convolutional Neural Networks for Accurate Chest X-Ray Diagnosis and Disease Detection. *Interdiscip Sci.* 2023 Sep;15(3):374-392. doi: 10.1007/s12539-023-00562-2. Epub 2023 Mar 26. PMID: 36966476; PMCID: PMC10040177.
11. Chan WK, Sun JH, Liou MJ, Li YR, Chou WY, Liu FH, Chen ST, Peng SJ. Using Deep Convolutional Neural Networks for Enhanced Ultrasonographic Image Diagnosis of Differentiated Thyroid Cancer. *Biomedicines.* 2021 Nov 26;9(12):1771. doi: 10.3390/biomedicines9121771. PMID: 34944587; PMCID: PMC8698578.
12. Jun Gao, Qian Jiang, Bo Zhou, Daozheng Chen. Convolutional neural networks for computer-aided detection or diagnosis in medical image analysis: An overview[J]. *Mathematical Biosciences and Engineering*, 2019, 16(6): 6536-6561. doi: 10.3934/mbe.2019326
13. Coleman S, Kerr D, Zhang Y. Image Sensing and Processing with Convolutional Neural Networks. *Sensors (Basel).* 2022 May 10;22(10):3612. doi: 10.3390/s22103612. PMID: 35632021; PMCID: PMC9146735.
14. Hosna, A., Merry, E., Gyalmo, J. *et al.* Transfer learning: a friendly introduction. *J Big Data* **9**, 102 (2022). <https://doi.org/10.1186/s40537-022-00652-w>
15. Berahmand, K., Daneshfar, F., Salehi, E.S. *et al.* Autoencoders and their applications in machine learning: a survey. *Artif Intell Rev* **57**, 28 (2024). <https://doi.org/10.1007/s10462-023-10662-6>
16. Pengzhi Li, Yan Pei, Jianqiang Li, A comprehensive survey on design and application of autoencoder in deep learning, *Applied Soft Computing*, Volume 138, 2023, 110176, ISSN 1568-4946, <https://doi.org/10.1016/j.asoc.2023.110176>.
17. E. Zdravevski, P. Lameski and A. Kulakov, "Weight of evidence as a tool for attribute transformation in the preprocessing stage of supervised learning algorithms," *The 2011 International Joint Conference on Neural Networks*, San Jose, CA, USA, 2011, pp. 181-188, doi: 10.1109/IJCNN.2011.6033219.
18. La Máquina Oráculo. (n.d.). U-Net y Segmentación Semántica. Retrieved from <https://lamaquinaoraculo.com/deep-learning/u-net-y-segmentacion-semantica/>
19. Simonyan K, Zisserman A. *Very deep convolutional networks for large-scale image recognition*. arXiv; 2014. arXiv:1409.1556. <https://doi.org/10.48550/arXiv.1409.1556>
20. Kermany DS, Goldbaum M, Cai W, Valentim CCS, Liang H, Baxter SL, McKeown A, Yang G, Wu X, Yan F, Dong J, Prasadha MK, Pei J, Ting MYL, Zhu J, Li C, Hewett S, Dong J, Ziyar I, Shi A, Zhang R, Zheng L, Hou R, Shi W, Fu X, Duan Y, Huu VAN, Wen C, Zhang ED, Zhang CL, Li O, Wang X, Singer MA, Sun X, Xu J, Tafreshi A, Lewis MA, Xia H, Zhang K. Identifying Medical Diagnoses and Treatable Diseases by Image-Based Deep Learning. *Cell.* 2018 Feb 22;172(5):1122-1131.e9. doi: 10.1016/j.cell.2018.02.010. PMID: 29474911.
21. Prakash D, Madusanka N, Bhattacharjee S, Kim CH, Park HG, Choi HK. Diagnosing Alzheimer's Disease Based on Multiclass MRI Scans using Transfer Learning

- Techniques. Curr Med Imaging. 2021;17(12):1460-1472. doi: 10.2174/1573405617666210127161812. PMID: 33504310.
22. Adi, A. (2023, October 5). ****Types of Neural Network Architectures****. ***The Clever Programmer***. Retrieved from [\[https://thecleverprogrammer.com/2023/10/05/types-of-neural-network-architectures/#google_vignette\]](https://thecleverprogrammer.com/2023/10/05/types-of-neural-network-architectures/#google_vignette)(https://thecleverprogrammer.com/2023/10/05/types-of-neural-network-architectures/#google_vignette)
 23. Adi, A. (2024, March 26). ****Neural Networks Guide****. ***The Clever Programmer***. Retrieved from [\[https://thecleverprogrammer.com/2024/03/26/neural-networks-guide/\]](https://thecleverprogrammer.com/2024/03/26/neural-networks-guide/)(<https://thecleverprogrammer.com/2024/03/26/neural-networks-guide/>)
 24. StatQuest with Josh Starmer. (n.d.). ****Neural Networks****. Retrieved from [\[https://youtube.com/playlist?list=PL-Ogd76BhmcBaUXZGPJkmQpLgrBgGZ7v0&si=GUsN1Mt3QFV_kNSf\]](https://youtube.com/playlist?list=PL-Ogd76BhmcBaUXZGPJkmQpLgrBgGZ7v0&si=GUsN1Mt3QFV_kNSf)(https://youtube.com/playlist?list=PL-Ogd76BhmcBaUXZGPJkmQpLgrBgGZ7v0&si=GUsN1Mt3QFV_kNSf)
 25. DeepLearning AI. (n.d.). ****Convolutional Neural Networks****. Retrieved from [\[https://youtube.com/playlist?list=PLZ8REt5zt2Pn0vfJjTAPaDVSACDvnuGiG&si=c0gHSqhEwPvixVTm\]](https://youtube.com/playlist?list=PLZ8REt5zt2Pn0vfJjTAPaDVSACDvnuGiG&si=c0gHSqhEwPvixVTm)(<https://youtube.com/playlist?list=PLZ8REt5zt2Pn0vfJjTAPaDVSACDvnuGiG&si=c0gHSqhEwPvixVTm>)