Deep Learning
Práctica – Javier Gómez

Informe

Sección 1: Análisis exploratorio de los datos

Los datos de esta práctica consisten en imágenes de **224 píxeles** x **224 píxeles** en formato de color **RGB**. Cada una de estas imágenes representa el fondo de ojo de un paciente, que puede padecer glaucoma o no. La figura 1 muestra diez imágenes aleatorias del conjunto de datos.

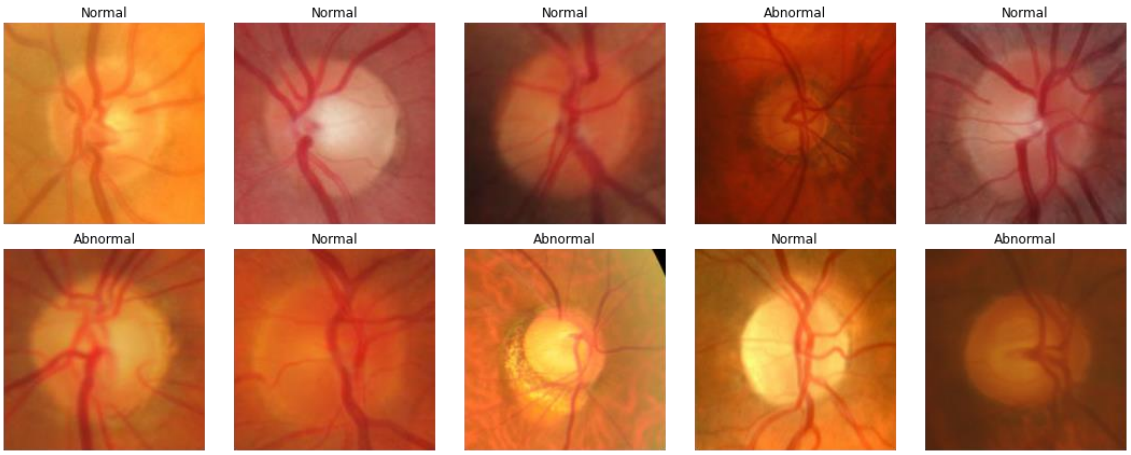

Figura 1. *Imágenes aleatorias del conjunto de datos etiquetadas.*

Las imágenes, tal y como han sido provistas, vienen estructuradas de la siguiente manera:

- ***Fold0***. Primera partición de los datos. Se estructura en:
  - ***Train***. Imágenes para realizar el entrenamiento de los modelos. Se dividen en dos grupos:
    - ***Normal***. Imágenes de pacientes sanos.
    - ***Abnormal***. Imágenes de pacientes con glaucoma.
  - ***Valid***. Imágenes para realizar la validación durante el entrenamiento de los modelos. Se dividen en los mismos dos grupos:
    - ***Normal***. Imágenes de pacientes sanos.
    - ***Abnormal***. Imágenes de pacientes con glaucoma.
  - ***Test***. Imágenes para testear el modelo ya entrenado y medir su performance. También están divididas en dos:
    - ***Normal***. Imágenes de pacientes sanos.
    - ***Abnormal***. Imágenes de pacientes con glaucoma.
- ***Fold1***. Segunda partición de los datos.
- ***…***
- ***Fold9.*** Última partición de los datos

Para realizar la práctica, los datos se han cargado en arrays según la utilidad que se les van a dar:

- **`X_train_0`**: imágenes de las carpetas ***Fold0*** -> ***Train*** -> ***Normal*** & ***Abnormal***.
- **`y_train_0`**: array de etiquetas. Contiene tantos **0**s y **1**s como imágenes en `X_train_0` según si provienen de la carpeta *Normal* o *Abnormal*, respectivamente. Se respeta estrictamente el mismo orden en ambos arrays; es decir, si la posición 4 de `X_train_0` contiene una imagen de la carpeta *Normal*, la misma posición 4 del array `y_train_0` contiene un 0.
- **`X_valid_0`**: imágenes de las carpetas ***Fold0*** -> ***Valid*** -> ***Normal*** & ***Abnormal***.
- **`y_valid_0`**: etiquetas con las mismas características que `y_train_0`, pero para el array `X_valid_0`
- **`X_test_0`**: imágenes de las carpetas ***Fold0*** -> ***Test*** -> ***Normal*** & ***Abnormal***.
- **`y_valid_0`**: etiquetas con las mismas características que `y_train_0` e `y_valid_0`, pero para el array `X_test_0`.

Por lo tanto, la forma y estructura de los datos para *Fold0* (obtenidas mediante la función `print_data_info` creada por el alumno) es la siguiente:

```
ENTRENAMIENTO          (1379, 224, 224, 3)
Nº de imágenes:        1379
Tamaño:                224 x 224 píxeles
Colores:               3
Nº de casos:           1379 (Normal + Abnormal)
- Normal:              754 (54.7%)
- Abnormal:            625 (45.3%)

TEST                   (174, 224, 224, 3)
Nº de imágenes:        174
Tamaño:                224 x 224 píxeles
Colores:               3
Nº de casos:           174 (Normal + Abnormal)
- Normal:              82 (47.1%)
- Abnormal:            92 (52.9%)

VALIDACIÓN             (154, 224, 224, 3)
Nº de imágenes:        154
Tamaño:                224 x 224 píxeles
Colores:               3
Nº de casos:           154 (Normal + Abnormal)
- Normal:              83 (53.9%)
- Abnormal:            71 (46.1%)
```

Como se puede observar, se cuenta con entorno a un **80%** de imágenes para *train*, **10%** para *test* y otro **10%** para validación. Además, los datos están bastante bien balanceados (**55%** - **45%**), por lo que no es necesario realizar ninguna técnica de balanceamiento o *data augmentation*.

Sección 2. Entrenamiento sobre una partición

Se van a construir 5 modelos diferentes y se van a entrenar con los datos de la partición ***Fold0***. La *figura 2* muestra la estructura de los cinco modelos.

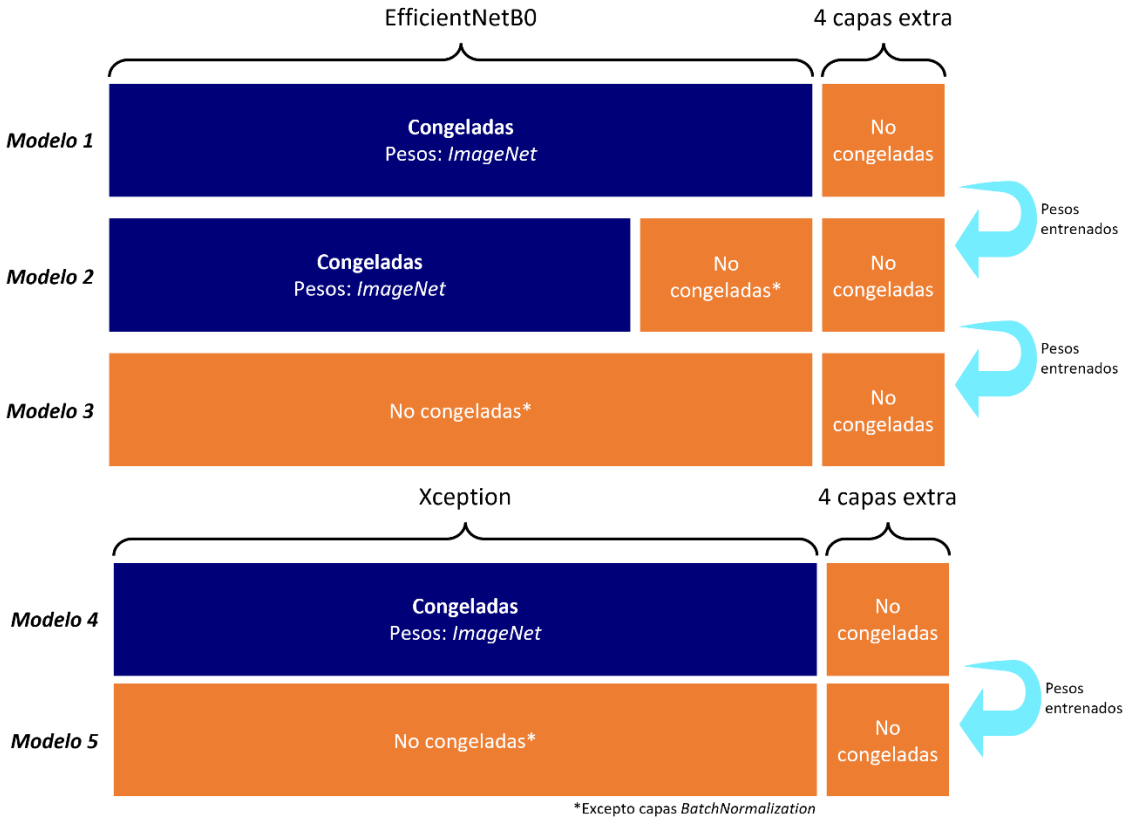

***Figura 2**. Esquema de los modelos 1, 2, 3, 4 y 5*

Tras varias ejecuciones, los hiperparámetros globales que mejores resultados han dado, y que se aplican a todos los modelos de la práctica, son los siguientes:

- `Epochs`: **60**. A partir de este límite, los modelos comienzan a sobrentrenarse.
- `BatchSize`: **32**. Tras probar con varios tamaños, este es el que mejor rendimiento da minimizando el tiempo de ejecución.
- `Loss`: **BinaryCrossEntropy**, ya que tan solo hay que predecir una clase binaria.
- `Metrics`: **F1-Score**.
- `Optimizer`: **Adam**.
- `Callbacks`:
  - `EarlyStopping`: monitorizando **F1-Score** de validación con **`patience`** = **10** para detener el entrenamiento cuando el **F1-Score** de los datos de validación no se incremente durante 10 *epochs* consecutivos.
  - `ModelCheckPoint`: guardar los mejores pesos obtenidos durante el entrenamiento del modelo.
- `Threshold`: **0.5**. Los modelos devuelven las predicciones con valores entre 0 y 1. Este *threshold* se utiliza para decidir cuándo consideramos una predicción

como resultado negativo y cuándo positivo. Toda predicción por debajo del *threshold* será interpretada como negativa; y en caso contrario, como positiva.

Modelo 1

- **Learning rate**: **1e-4.**
- **Mejor epoch**: **16.**
- **F1-Score** sobre datos de test: **0.767**.
- Gráficas de **Loss** y **F1-Score**: figura 3.
- **Matriz de confusión**: figura 4.

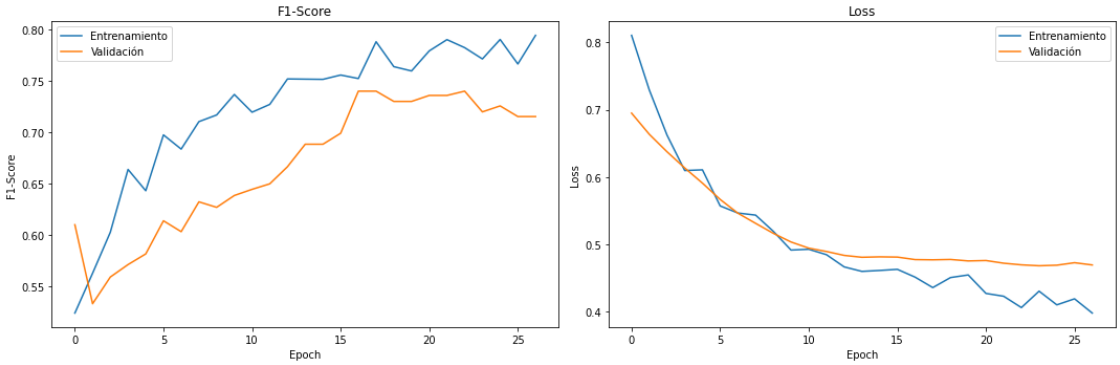

Figura 3. *F1-Score y Loss function durante el entrenamiento del modelo 1*

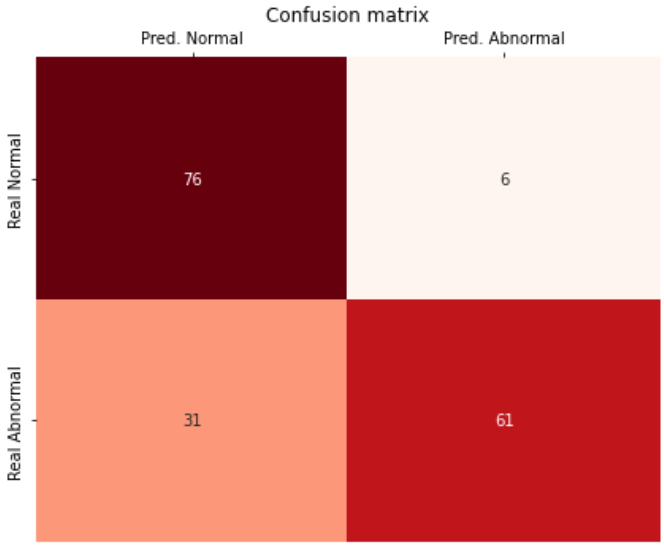

Figura 4*. Matriz de confusión con las predicciones del modelo 1 sobre los datos de test*

Modelo 2

- **Learning rate**: **1e-5**.
- **Mejor epoch**: **55.**
- **F1-Score** sobre datos de test: **0.882.**
- Gráficas de **Loss** y **F1-Score**: figura 5.
- **Matriz de confusión**: figura 6.

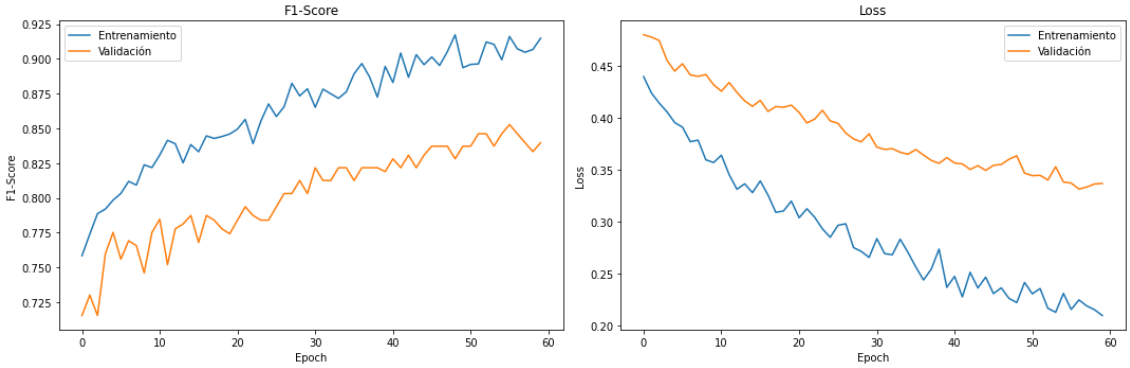

***Figura 5**. F1-Score y Loss function durante el entrenamiento del modelo 2*

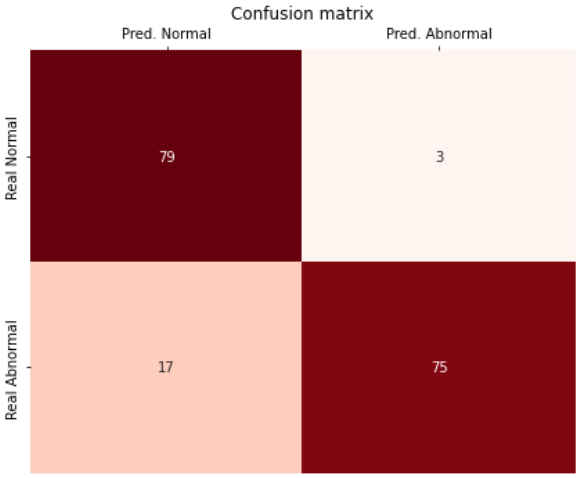

***Figura 6**. Matriz de confusión con las predicciones del modelo 2 sobre los datos de test*

Modelo 3

- **Learning rate**: **1e-5**.
- **Mejor epoch**: **32.**
- **F1-Score** sobre datos de test: **0.903.**
- Gráficas de **Loss** y **F1-Score**: figura 7.
- **Matriz de confusión**: figura 8.

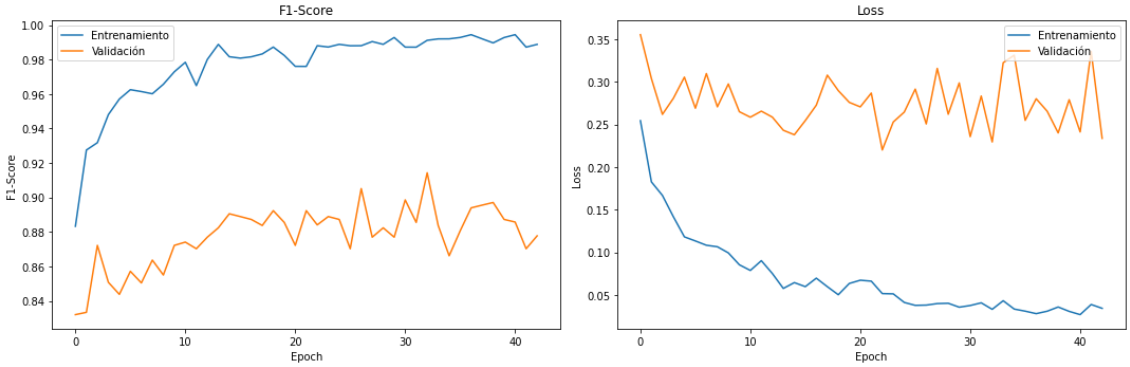

***Figura 7**. F1-Score y Loss function durante el entrenamiento del modelo 3*

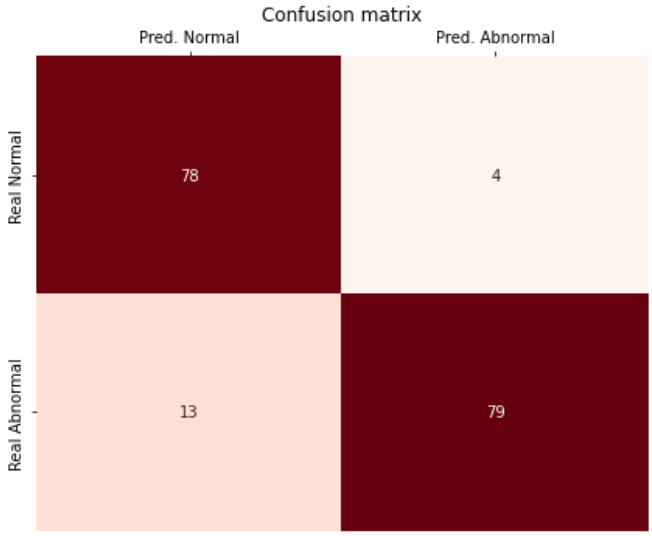

***Figura 8**. Matriz de confusión con las predicciones del modelo 3 sobre los datos de test*

Modelo 4

- **Learning rate**: **1e-4**.
- **Mejor epoch**: **3.**
- **F1-Score** sobre datos de test: **0.772.**
- Gráficas de **Loss** y **F1-Score**: figura 9.
- **Matriz de confusión**: figura 10.

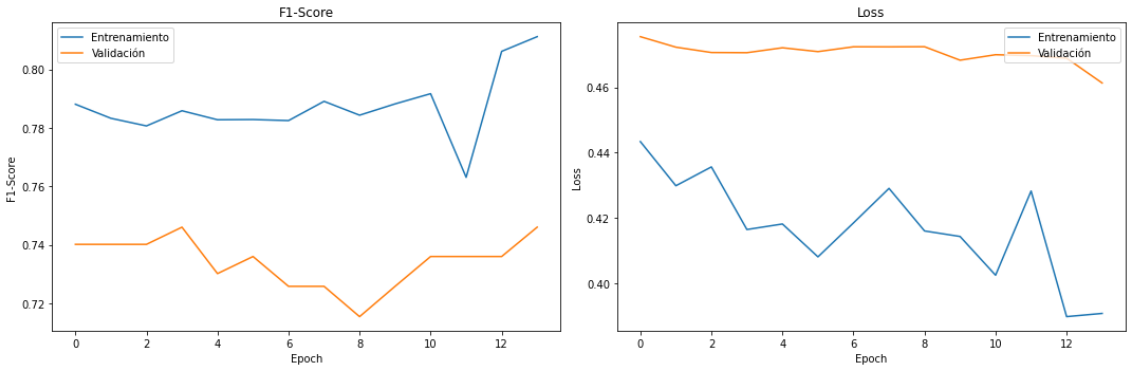

***Figura 9**. F1-Score y Loss function durante el entrenamiento del modelo 4*

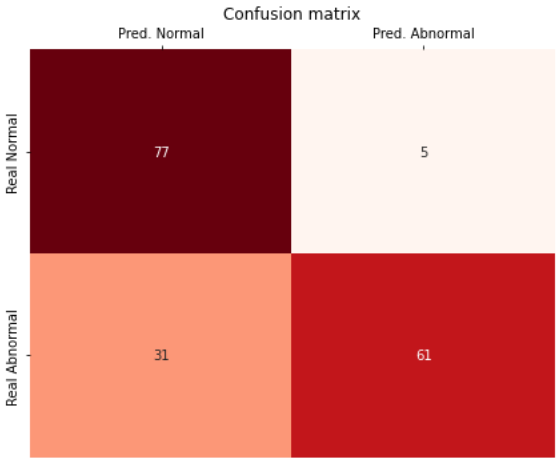

***Figura 10**. Matriz de confusión con las predicciones del modelo 4 sobre los datos de test*

Modelo 5

- **Learning rate**: **1e-5**.
- **Mejor epoch**: **22.**
- **F1-Score** sobre datos de test: **0.877.**
- Gráficas de **Loss** y **F1-Score**: figura 11.
- **Matriz de confusión**: figura 12.

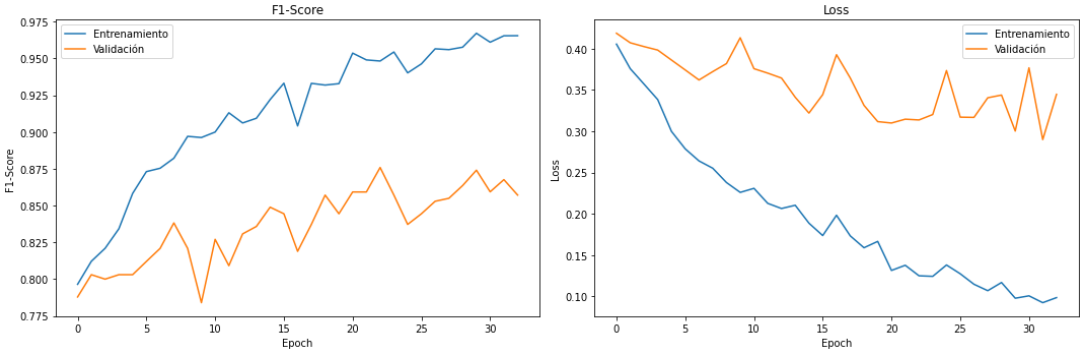

***Figura 11**. F1-Score y Loss function durante el entrenamiento del modelo 5*

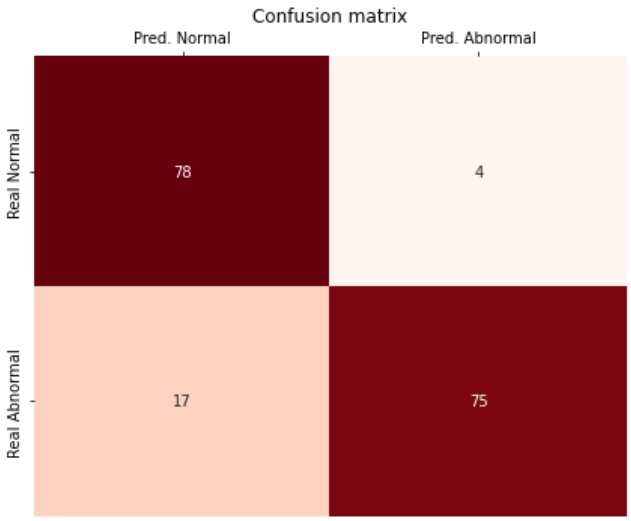

***Ilustración 12**. Matriz de confusión con las predicciones del modelo 5 sobre los datos de test*

Sección 3. Validación cruzada y discusión

Mejor modelo

El modelo con mejor F1-Score es el modelo 3.

```
F1-SCORES
Modelo 1 -> 0.767
Modelo 2 -> 0.882
Modelo 3 -> 0.903
Modelo 4 -> 0.772
Modelo 5 -> 0.877

El mejor modelo es el Modelo 3
```

Para cada partición (*Fold*), entrenamos el modelo 3 con los datos de *train* y validación, y medimos su performance con los datos de *test*.

Cross validation

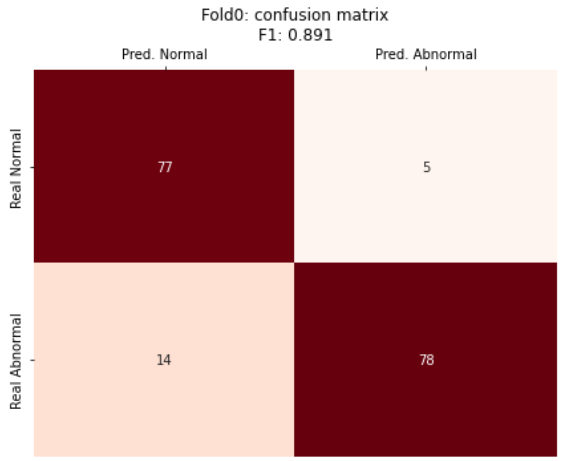

***Figura 13**. Matriz de confusión con Fold0*

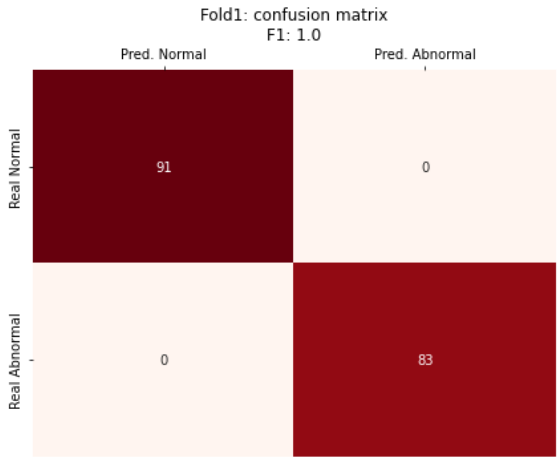

***Figura 14**. Matriz de confusión con Fold1*

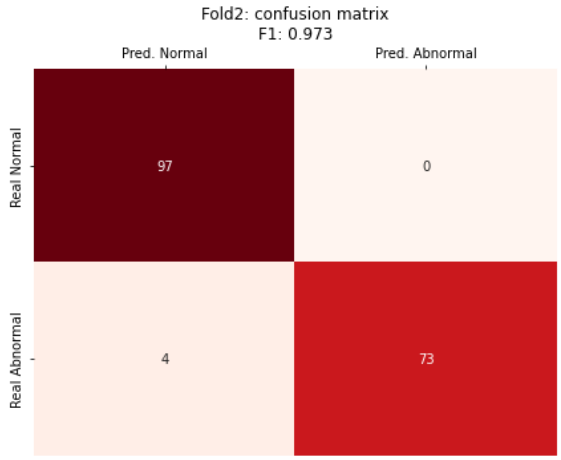

***Figura 15**. Matriz de confusión con Fold2*

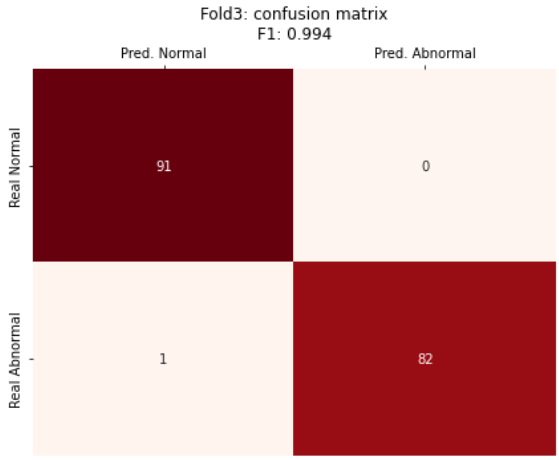

***Figura 16**. Matriz de confusión con Fold3*

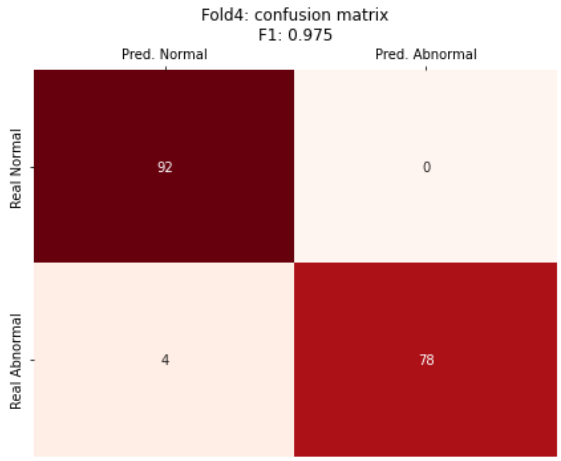

***Figura 17**. Matriz de confusión con Fold4*

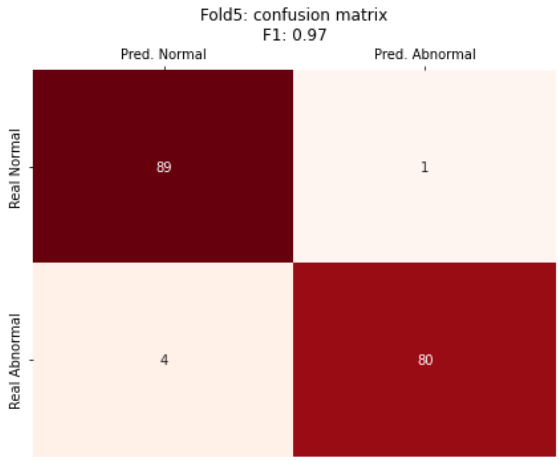

***Figura 18**. Matriz de confusión con Fold5*

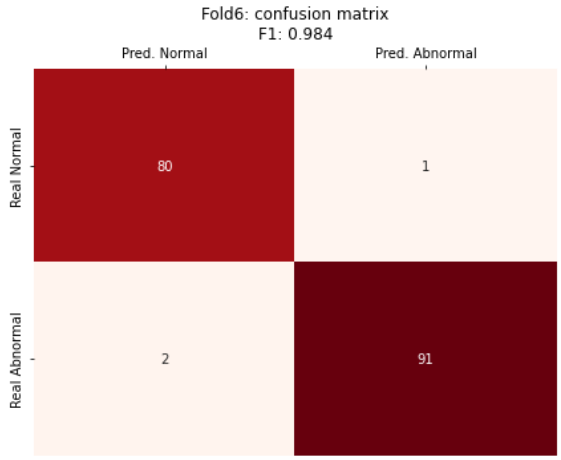

***Figura 19**. Matriz de confusión con Fold6*

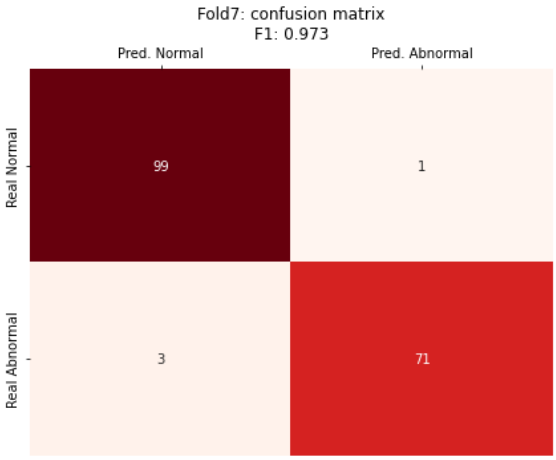

***Figura 20**. Matriz de confusión con Fold7*

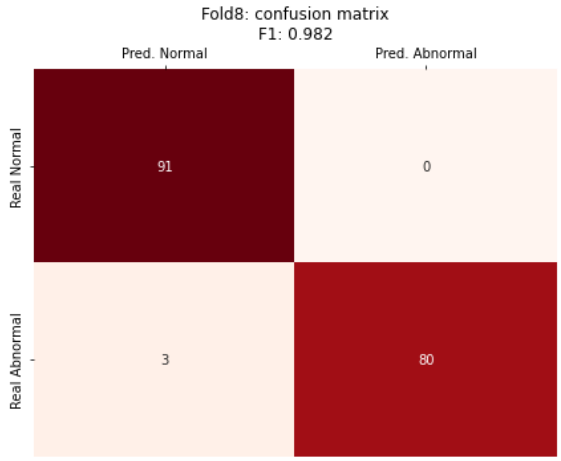

***Figura 21**. Matriz de confusión con Fold8*

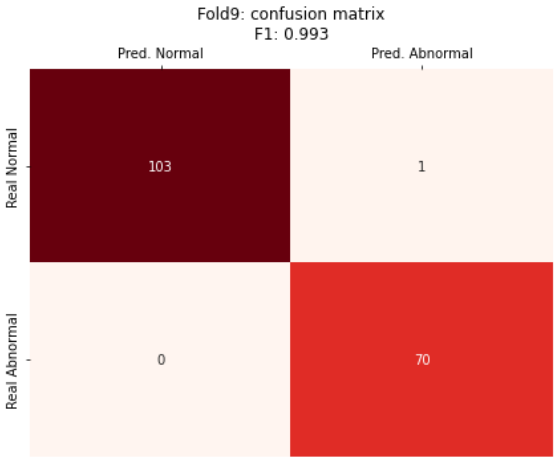

***Figura 22**. Matriz de confusión con Fold9*

F1-SCORES

```
Fold0 -> 0.8914
Fold1 -> 1.0
Fold2 -> 0.9733
Fold3 -> 0.9939
Fold4 -> 0.975
Fold5 -> 0.9697
Fold6 -> 0.9838
Fold7 -> 0.9726
Fold8 -> 0.9816
Fold9 -> 0.9929

Valor medio -> 0.9734
Desviación estándar -> 0.029
```

Minimizar falsos negativos

Una forma sencilla de minimizar los falsos negativos es ajustando el *threshold* (que anteriormente hemos establecido en 0.5) a partir del cuál interpretamos las predicciones del modelo como positivos. Cuanto más alto sea este *threshold*, con más facilidad clasificaremos a un paciente nuevo como negativo; y cuanto más pequeño sea, más

difícil será clasificarlo como tal. Esta forma de interpretar las probabilidades que nos da el modelo nos permite ajustar las predicciones en función de nuestros intereses, por lo que en este caso nos convendría bajarlo para minimizar los falsos negativos.

Sin embargo, al hacer esto estamos aumentando las probabilidades de clasificar como positivos a pacientes que no lo son, es decir, de aumentar los falsos positivos. En función de cuánto queramos minimizar los falsos negativos en detrimento de aumentar la probabilidad de los falsos positivos, estableceremos un *threshold* más o menos bajo.

```
Antes de minimizar falsos negativos:
Threshold -> 0.5
Falsos Negativos -> 4
F1-Score -> 0.9697

Después de minimizar falsos negativos:
Threshold -> 0.35
Falsos Negativos -> 3
F1-Score -> 0.9759
```

En este ejemplo (modelo 3 entrenado y testeado con *Fold5*), vemos cómo al decrementar el *threshold* hemos reducido los falsos negativos. Además, vemos que el F1-Score también ha mejorado. Esto se debe a que el falso negativo no se ha transformado en un falso positivo, si no en un verdadero positivo, lo que ha mejorado la performance del modelo; esto es debido a que es mucho más probable que una predicción positiva sea en realidad un verdadero positivo que un falso positivo.

Sección 4. Análisis crítico

Estrategia de diseño

Hubiera mantenido la misma estructura, pero el hecho de tener que crear un array de etiquetas manualmente (añadir un 0 cuando se carga una imagen de *Normal* y un 1 de *Abnormal*) puede ser una posible fuente de errores. Para solucionarlo, hubiera incluido un *csv* por cada carpeta de *train*, *valid* y *test* que incluyera, por orden, los nombres de las imágenes y su etiqueta. De esta forma tan solo bastaría con leer el *csv* para saber si una imagen está bien etiquetada o no.

Puntos importantes para la generalización

Es importante que cada partición contenga aproximadamente el mismo número de imágenes de cada clase, es decir, que estén balanceadas. Además, que los tamaños de las particiones sean similares entre sí para evitar la falta de datos en alguna de ellas.

Otro aspecto importante es mantener una diversidad de características de los datos entre todas las particiones; por ejemplo, no acumular todas las imágenes oscuras o borrosas en una sola partición para mejorar la capacidad de generalización. Cuantas más características se identifiquen (tonalidad, luminosidad, e incluso la marca o tipo de máquina utilizada para obtener los datos, etc.), más diversas serán las particiones.

Análisis y conclusión

Los resultados muestran que los modelos pre-entrenados proporcionan una **sólida base** con la que no es necesario "reinventar la rueda" una y otra vez. Con tan solo adaptar el *input* y retocar las capas de salida hemos obtenido una *performance* bastante decente (F1 modelo 1: **0.767** -*EfficientNet*-; F1 modelo 4: **0.772** -*Xception*-). Con este gran punto de partida, tras realizar un *fine-tuning* ligero en cuanto a computación, hemos obtenido modelos muy **potentes** y **consistentes** que se adaptan perfectamente a una nueva tarea, obteniendo resultados excepcionales tras realizar *cross validation*, con un F1-Score medio de **0.97** y una desviación estándar de **0.029**. Obtener estos resultados desde cero sería altamente costoso.

Por lo tanto, no solo se obtienen mejores resultados al realizar estas técnicas, sino que también se requiere de menos recursos para lograrlos.