Clasificador Neuronal para la detección del Glaucoma

Detección del Glaucoma a través de la clasificación de imágenes utilizando redes neuronales Convolucionales.

Jorge Álvarez Gracia
Universitat Oberta de Catalunya. Master Oficial de Ciencia de Datos. Asignatura de Deep Learning.

Introducción

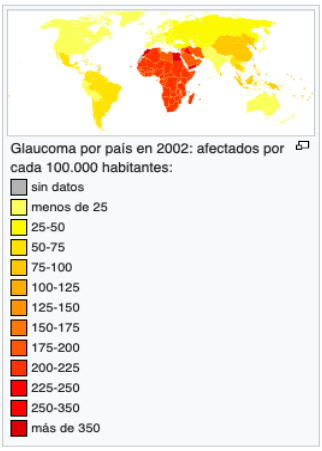
Motivación

Este trabajo forma parte de la práctica de la asignatura Deep Learning del Máster Oficial de Ciencia de Datos de la Universidad Oberta de Cataluña, del que he formado parte durante el año 2022 como estudiante de máster y en concreto de esta asignatura optativa.

Agradecer a los profesores Iván González Torre y Jordi Casas Roma, que han planteado esta práctica siendo necesaria y fundamental para consolidar todos los conceptos teóricos aprendidos durante el curso y que han apoyado a mí y a mis compañeros durante este periodo académico.

El glaucoma

El glaucoma es un conjunto de enfermedades que pueden dañar el nervio óptico y la retina del ojo y pueden generar pérdida de visión o ceguera. Generalmente es causado por el aumento patológico de la presión intraocular, por falta de drenaje del humor acuoso y tiene como última consecuencia una neuropatía óptica que se caracteriza por la pérdida progresiva de las fibras nerviosas del nervio óptico y cambios en su aspecto. **Definición de wikipedia.*



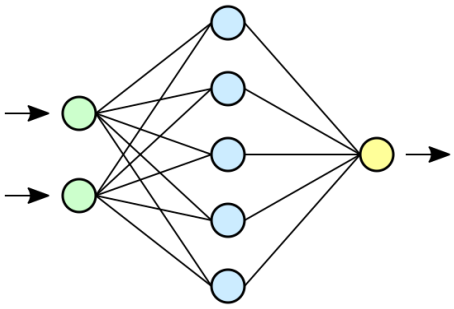
La mayoría de las personas afectadas no presentan síntomas en las primeras fases de la enfermedad, habitualmente cuando la enfermedad está en un punto avanzado aparecen defectos en el campo visual y pérdida progresiva de visión. Es inusual que exista dolor ocular en el glaucoma crónico, pero es frecuente en el glaucoma agudo (glaucoma de ángulo cerrado), el cual sí puede ocasionar intensos síntomas desde su inicio. Por ello es fundamental una detección temprana de esta enfermedad.

Estudios demuestran que aunque esta enfermedad es incurable, alrededor de un 90% de la ceguera provocada por el Glaucoma se podría evitar gracias a su detección temprana.

National Glaucoma Resarch estima que en 2020 alrededor de 80 millones de personas han padecido glaucoma en todo el mundo y se espera que este número aumente a más de 111 millones para 2040.

Según la OMS (Organización Mundial de la Salud), el glaucoma es la segunda causa más importante de ceguera en el mundo. Además, como podemos observar en el mapa, el número de las personas afectadas por Glaucoma por cada 100.000 habitantes es mucho mayor en aquellas zonas más desfavorecidas y con menores recursos económicos.

Redes neuronales convolucionales.



Dentro del Machine Learning o Aprendizaje Automático encontramos varios tipos de problemas a resolver, que se diferencian entre sí, en aquellos si a priori conocemos o no la solución. Supervisado (conocemos la solución al problema) y no supervisado (no conocemos la soluciòn)

En el Machine learning supervisado, que es el que nos ocupa, ya que conocemos de antemano si una persona padecerá o no glaucoma, nos encontramos ante el problema de clasificación, distinguiremos entre aquellas personas que sufrirán o no Glaucoma y para esta tarea utilizaremos una técnica de Machine Learning llamada deep learning y en concreto utilizaremos modelos de redes neuronales convolucionales.

Las redes neuronales son una técnica de deep learning, aprendizaje profundo, que simula el funcionamiento del cerebro humano. Está formado por un conjunto de nodos conocidos como neuronas artificiales que están conectadas y que reciben y envían cierta información y conocimiento generado entre sí. Estas señales reciben un input inicial, datos, imágenes, etc y obtienen un output resultante que contiene la solución al problema que pretendemos resolver.

Las redes neuronales convolucionales están optimizadas para la visión artificial, en la clasificación de imágenes o segmentación de imágenes, entre otras tareas, Están basadas en las neuronas en la corteza visual primaria de un cerebro biológico. Más información sobre CNN.

Por lo tanto, vamos a utilizar las técnicas descritas anteriormente para poder generar un modelo al que como inputs le hemos introducido distintas imágenes oculares, categorizadas entre aquellas donde hay glaucoma o no, como output resultante del modelo obtendremos la respuesta si para esa imagen tendríamos problemas de glaucoma o no, el porcentaje de sufrir glaucoma dada la imagen.

A continuación explicaremos más extensamente las características sobre el conjunto de datos utilizado y las distintas técnicas utilizadas.

Tecnologias Utilizadas

Api Tensorflow y Keras

En todo el proyecto hemos utilizado básicamente diferentes clases y métodos de esta API, tensor flow es una herramienta de código abierto para el aprendizaje automático desarrollada por google y Keras es una biblioteca que proporciona bloques modulares sobre los que se pueden desarrollar modelos complejos de aprendizaje profundo.

Otras librerías de python

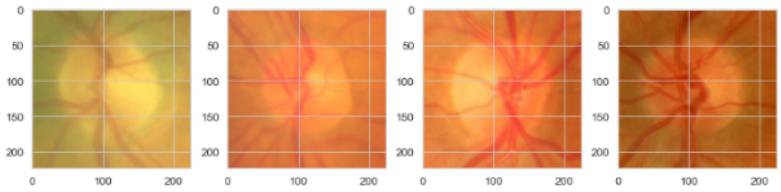
Además he usado otras librerías de python como son Scikit-learn , biblioteca para aprendizaje automático, pandas, seaborn, entre otras.

Conjunto de datos

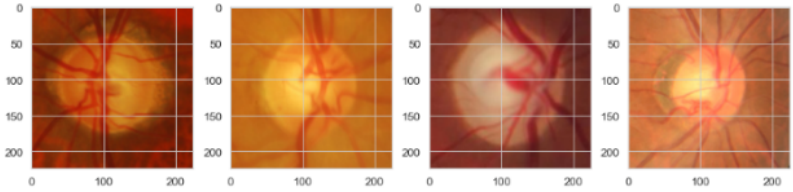
Descripción general

Como hemos descrito, los datos utilizados para poder realizar la tarea de clasificación son imágenes oculares diferenciadas entre aquellas que parecen glaucoma y las que no.

Glaucoma normal:



Glaucoma Abnormal



Como observamos a simple vista y para personas no expertas en la materia no encontramos diferencias singulares entre los distintos grupos de imágenes.

En total, el conjunto total de datos es de 17070 imágenes de las cuales 9190 son de glaucoma normal y 7880 de glaucoma anormal. *Ver tabla 1*. Es importante que el número de imágenes esté más o menos balanceado para aumentar la eficiencia de los modelos.

Las imagenes estan a color y tienen una dimensión de 224*224.

Por razones técnicas y de sencillez, que explicaremos a continuación, a la hora de realizar estos modelos se han dividido dichas imágenes en distintas carpetas y a su vez estas en tres directorios diferenciados.

En este tipo de problemas de machine learning lo recomendable es dividir el conjunto de datos en dos subgrupos, aquellos que utilizaremos para el **entrenamiento** de los modelos y aquellos que usaremos para su testeo o verificación de que el modelo funciona de manera correcta para un conjunto de datos nuevo y no sirve específicamente para los datos de entrenamiento, es decir, el modelo no solo funciona para los datos de entrenamiento, este conjunto es el de **test**, y como hemos dicho, estos datos deben ser independientes al resto. *Ver Figura 1.*

Conjunto de entrenamiento:
- Abnormal Glaucoma
- Normal Glaucoma

Conjunto de Validación:
- Abnormal Glaucoma
- Normal Glaucoma

Conjunto de Test:
- Abnormal Glaucoma
- Normal Glaucoma

A su vez los datos de entrenamiento se pueden dividir en conjunto de **entrenamiento** y **validación,** la separación de estos dos conjuntos nos permite obtener modelos más eficientes ya que serán menos propensos a la generalización y nos servirán para utilizar técnicas de validación cruzada pudiendo comparar y eligiendo los mejores modelos. En relación a este aspecto destacar que tendremos diez directorios diferentes llamados fold con los distintos conjuntos de datos en su interior y como en el aspecto anterior serán utilizados en la parte final de este estudio y cuya utilidad explicaremos más adelante. En un primer momento, trabajaremos únicamente con el fold 0.

Figura 1: Datos de entrenamiento, validación y test

Por lo tanto el esquema de nuestros conjuntos de datos con sus respectivos directorios sería el siguiente:

```
└── practica_DL_UOC_2022
    ├── Fold0
    │   ├── test
    │   │   ├── abnormal
    │   │   └── normal
    │   ├── train
    │   │   ├── abnormal
    │   │   └── normal
    │   └── valid
    │       ├── abnormal
    │       └── normal
    ├── Fold1
    │   ├── test
    │   │   ├── abnormal
    │   │   └── normal
    │   ├── train
    │   │   ├── abnormal
    │   │   └── normal
    │   └── valid
    │       ├── abnormal
    │       └── normal
    ├── (...)
```

	Fold_name	Train_normal	Train_abnormal	Validation_normal	Validation_abnormal	Test_normal	Test_abnormal	Total_normal	Total_abnormal
0	Fold0	754	625	83	71	82	92	919	788
1	Fold1	740	639	88	66	91	83	919	788
2	Fold2	739	640	83	71	97	77	919	788
3	Fold3	743	636	85	69	91	83	919	788
4	Fold4	746	633	81	73	92	82	919	788
5	Fold5	758	621	71	83	90	84	919	788
6	Fold6	754	625	84	70	81	93	919	788
7	Fold7	737	642	82	72	100	74	919	788
8	Fold8	748	631	80	74	91	83	919	788
9	Fold9	733	646	82	72	104	70	919	788

Tabla 1. Descripción del conjunto de datos, imágenes por conjunto de datos, folds y categoría.

Pre-procesamiento

Al trabajar con imágenes como inputs de entrada podríamos haber realizado ciertos cambios concretos, como reducir la dimensión de las imágenes para realizar los entrenamientos de forma más rápida o utilizar otro tipo de redes neuronales más potentes, opción que llevamos a cabo pero que no condujo a modelos más consistentes. Por ello, hemos evitado utilizar esta técnica ya que disminuiría la predicción del modelo. Además en nuestro caso no deberemos normalizar los datos, ya que los modelos que utilizaremos lo harán por nosotros.

Data argumentation

Esta técnica se utiliza para evitar el problema del que ya hemos hablado anteriormente, la no generalización de los modelos o en inglés conocido como overfitting, siendo una de las principales causas por las que nuestro modelo podría ser no válido. Esta técnica consiste en realizar distintos cambios y modificaciones a distintos lotes de imágenes, como por ejemplo girarlas levemente, añadir zoom, etc. De esta manera incrementamos la diversidad de las imágenes y realizamos modelos más generalistas que no solo funcionan con los datos de entrenamiento. En nuestro caso, hemos fijado los distintos parámetros, correspondientes al método ImageDataGenerator de la api *tensorflow.keras*.

Este proceso solo será necesario realizar para el conjunto de datos de entrenamiento

Parámetros ImageDatagerator
rotation_range=20
zoom_range=0.05
width_shift_range=0.05,
height_shift_range=0.05,
horizontal_flip=True
fill_mode='nearest'

Carga de los datos

Gracias a que teníamos las distintas imágenes, outputs del modelo, ordenadas adecuadamente por carpetas, hemos usado la función data generator y en concreto, de la librería tensor flow keras para manejar y cargar nuestros datos de una forma rápida y sencilla, así como para etiquetar automáticamente estas imágenes por tipo de glaucoma.

Parámetros flow_from_directory
flow_from_directory
target_size=((img_size))
color_mode='rgb'
shuffle=True, seed=SEED)

Para el caso del conjunto de datos de test no necesitamos 'barajar las imágenes', *shuffle.* Esta técnica se usa para elegir aleatoriamente las distintas imágenes de entrenamiento y validación y que el modelo no tenga el riesgo de aprender el orden específico de las imágenes.

Modelos

Introducción

En este proyecto realizaremos cinco modelos distintos con sus características particulares que comentaremos a continuación. Elegiremos el modelo más eficiente y realizaremos la técnica de validación *cross validation* para certificar que el modelo funciona adecuadamente.

Esquema Final:

```
├──models
│   ├──model_1
│   │   ├──assets
│   │   └──variables
│   ├──model_2
│   │   ├──assets
│   │   └──variables
│   ├──model_3
│   │   ├──assets
│   │   └──variables
│   ├──model_4
│   │   ├──assets
│   │   └──variables
│   ├──model_5
│   │   ├──assets
│   │   └──variables
│   └──model_final
│       ├──Results_Fold_0
│       │   ├──assets
│       │   └──variables
│       ├──Results_Fold_1
│       ├──Results_Fold_2
│       ├──Results_Fold_3
│       ├──Results_Fold_4
│       ├──Results_Fold_5
│       ├──Results_Fold_6
│       ├──Results_Fold_7
│       ├──Results_Fold_8
│       └──Results_Fold_9
```

Antes de eso, crearemos los directorios necesarios para poder guardar nuestros modelos o cargarlos a posteriori.

Transfer Learning:

En todos estos modelos utilizaremos técnicas de transferencia de aprendizaje (*transfer learning*), utilizadas habitualmente en este tipo de problemas y cuyo funcionamiento se basa en reutilizar un modelo que ha sido entrenado y utilizado para resolver problemas similares.

La api de keras nos permite cargar distintos modelos de redes neuronales pre-entrenadas. Todos estos modelos que utilizamos han sido previamente entrenados con los pesos de ImageNet. *ImageNet es una base de datos de imágenes que contiene 1000 clases de objetos y 1.281.167 imágenes de entrenamiento, 50.000 imágenes de validación y 100.000 imágenes de prueba.*

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130.2	10.0
MobileNet	16	70.4%	89.5%	4.3M	55	22.6	3.4
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8
DenseNet121	33	75.0%	92.3%	8.1M	242	77.1	5.4
DenseNet169	57	76.2%	93.2%	14.3M	338	96.4	6.3
DenseNet201	80	77.3%	93.6%	20.2M	402	127.2	6.7
NASNetMobile	23	74.4%	91.9%	5.3M	389	27.0	6.7
NASNetLarge	343	82.5%	96.0%	88.9M	533	344.5	20.0
EfficientNetB0	29	77.1%	93.3%	5.3M	132	46.0	4.9
EfficientNetB1	31	79.1%	94.4%	7.9M	186	60.2	5.6
EfficientNetB2	36	80.1%	94.9%	9.2M	186	80.8	6.5
EfficientNetB3	48	81.6%	95.7%	12.3M	210	140.0	8.8
EfficientNetB4	75	82.9%	96.4%	19.5M	258	308.3	15.1
EfficientNetB5	118	83.6%	96.7%	30.6M	312	579.2	25.3
EfficientNetB6	166	84.0%	96.8%	43.3M	360	958.1	40.4
EfficientNetB7	256	84.3%	97.0%	66.7M	438	1578.9	61.6
EfficientNetV2B0	29	78.7%	94.3%	7.2M	-	-	-
EfficientNetV2B1	34	79.8%	95.0%	8.2M	-	-	-
EfficientNetV2B2	42	80.5%	95.1%	10.2M	-	-	-
EfficientNetV2B3	59	82.0%	95.8%	14.5M	-	-	-
EfficientNetV2S	88	83.9%	96.7%	21.6M	-	-	-
EfficientNetV2M	220	85.3%	97.4%	54.4M	-	-	-
EfficientNetV2L	479	85.7%	97.5%	119.0M	-	-	-

Tabla 2: Distintos modelos disponibles , parámetros y su precisión para ImageNet, tabla obtenida de la api de keras.

Para los tres primeros modelos utilizaremos la red neural EficientNetB0 realizando los cambios necesarios para adaptarla a nuestros datos de entrada y salida. En el cuarto modelo utilizaremos la aplicación de keras VGG16 y para el último modelo usaremos el EffcienteNetB2, con distintas tareas de optimización.

Funciones, hiper-parámetros y técnicas utilizadas.

Las funciones o parámetros que debemos definir a la hora de realizar un modelo de estas características son varios y debemos diferenciar entre ellos que fijamos a la hora de compilación o creación del modelo y los que utilizamos a la hora de su entrenamiento conforme a la Api tensor flow keras.

Para el primer grupo deberemos definir los siguientes:

Función de pérdida: a grandes rasgos, podemos definirla como la medida qué valora lo bien que funciona su modelo en términos de poder obtener el resultado esperado. Convierte el problema de aprendizaje en un problema de optimización, se define una función de pérdida y luego optimizamos el algoritmo para minimizar dicha función. Para todos los modelos hemos usado la función de pérdida, ***'categorical_crossentropy'***, ya que esta se usa para modelos de clasificación de múltiples clases donde hay dos o más etiquetas de salida.

Optimizador: Los optimizadores son algoritmos utilizados para cambiar los atributos de la red neuronal, como los pesos y la tasa de aprendizaje, para reducir las pérdidas. Los optimizadores se utilizan para resolver problemas de optimización minimizando la función de pérdida. Utilizaremos para todos nuestros modelos el optimizador **Adam**, combinación de los optimizadores *Adagrad y momentum*, ya que es el que mejor funciona para este tipo de problemas.

Ratio de aprendizaje: Es un parámetro de ajuste del algoritmo de optimización que determina el tamaño del paso en cada iteración mientras se mueve hacia un mínimo de una función de pérdida. A continuación obtendremos cual es mejor para nuestros modelos.

Función de activación: Esta función se encarga de devolver el output de la red neuronal tal y como necesitamos para resolver nuestro problema, en nuestro caso, es la clasificación binaria, así que necesitamos una función que nos devuelva la probabilidad o los valores 0 y 1 según el tipo de diagnóstico que nos encontremos. Utilizaremos pues la función Softmax, utilizada para la clasificación no binaria pero que funciona de manera óptima en nuestro caso.

Métricas: A la hora de compilar un modelo deberemos establecer las métricas de evaluación del modelo, estas se utilizan para supervisar y medir el rendimiento de un modelo (durante el entrenamiento y el test). No es necesario que estas sean las mismas, en nuestro caso utilizaremos la métrica **F1 Score**, para entrenamiento y test, aunque para este segundo también deberemos tener en cuenta, los falsos positivos ya que estamos ante un problema médico y el objetivo principal es no diagnosticar favorablemente a una persona que realmente sufre glaucoma. La métrica F1 Score es una métrica que combina,la precisión, que porcentaje de pacientes tendrán glaucoma y la exhaustividad, en nuestro caso qué porcentaje de personas con glaucoma somos capaces de identificar.

F1 se calcula haciendo la media armónica entre la precisión y la exhaustividad:

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Para el segundo grupo, referente al entrenamiento del modelo:

Epochs: El número de épocas es un hiper-parámetro que define el número de veces que el algoritmo de aprendizaje funcionará en todo el conjunto de datos de entrenamiento.Una época se compone de uno o más lotes. En nuestros modelos estableceremos un Epochs inicial de 100.

Batch Size : Batch size, o tamaño del lote es un hiper-parámetro que define el número de muestras para trabajar antes de actualizar los parámetros del modelo interno. Un conjunto de datos de entrenamiento se puede dividir en uno o más lotes.

En nuestro caso utilizaremos la técnica de **Mini-Batch Gradient Descent,** cuando el tamaño del lote es menor que el tamaño del conjunto de datos de entrenamiento y mayor que el de una muestra de este, En el caso del descenso de gradiente de mini lotes, los tamaños de lotes populares incluyen 32, 64 y 128 muestras. Para nuestros modelos elegiremos entre 32 y 64 diferenciados por la pérdida obtenida.

Uso de Callbacks**:** Por otro lado, cabe destacar que la Api, ya mencionada, nos permite especificar todavía más algunas características de nuestros modelos. Esta característica es llamada *Callback*s. Para los cuatro primeros modelos la utilizaremos un callback llamado early stopping que nos permitirá finalizar el entrenamiento cuando la pérdida de nuestro modelo no mejoré sustancialmente, es decir, no llevar a cabo más epochs. Para el modelo 5, fijamos 100 epochs y usaremos un ratio de aprendizaje decreciente. Además usaremos otros callbacks para guardar nuestros modelos y sus informes de actividad.

Encontramos todos los modelos, parámetros, capas, entrenamiento y resultados en el Anexo.

Modelo 1.

Implementaremos un modelo basado en el EfficientNet B0, pre-entrenado con los pesos de ImageNet al que modificaremos la salida para adaptarla a nuestro modelo añadiendo las distintas capas:

- GlobalAveragePooling2D: toma un tensor de tamaño (ancho de entrada) x (alto de entrada) x (canales de entrada) y calcula el valor medio de todos los valores en toda la matriz (ancho de entrada) x (alto de entrada) para cada uno de los (canales de entrada).
- BatchNormalization: La normalización por lotes es un método que se utiliza para hacer que las redes neuronales artificiales sean más rápidas y estables mediante la normalización de las entradas de las capas.
- Dropout : Es una técnica de regularización para reducir el sobreajuste en redes neuronales artificiales. Trata de omitir aleatoriamente neuronas durante el proceso de entrenamiento de una red neuronal.
- Capa fully connected: Capa de una red neuronal completamente conectada.

Mejores hiperparametros del modelo

Entrenaremos el primer modelo con las distintas combinaciones de Bacth size y learning rate definidas previamente, en nuestro caso:

LEARNING_RATES = [1e-4, 3e-4, 5e-4, 0.001, 0.01]
BATCH_SIZES = [32,64]

	Batch	Learning rate	Time	F1_Score	Loss
2	32	0.0005	155.117387	0.802484	0.453109
1	32	0.0003	156.062228	0.789742	0.481586
0	32	0.0001	198.554367	0.735928	0.505434
3	32	0.0010	155.300179	0.778685	0.572243
8	64	0.0010	80.556701	0.832398	0.682221
7	64	0.0005	81.372994	0.820839	0.728233
6	64	0.0003	80.850434	0.790333	0.803889
5	64	0.0001	79.782974	0.801982	0.837273
4	32	0.0100	159.559914	0.802484	0.913599
9	64	0.0100	79.307136	0.803324	1.006532

Elegiremos la combinación de parámetros que mejor resultados nos da fijándonos en la pérdida, en este caso correspondiente al ratio de aprendizaje de 0.0005 y batch size de 32. Este resultado es consistente con los estudios realizados por los expertos en la materia, en relación a la optimización de parámetros de este tipo de redes neuronales convolucionales, así que por este motivo fijamos estos parámetros para el resto de los modelos. Destacar que para evitar tiempos de ejecución extremadamente largos en todos los modelos excepto en el último estableceremos el callback de *earlystop* por lo que el número de épocas de entrenamiento será variable según los resultados obtenidos.

El f1 score obtenido es de 79.08 % y el número de personas con glaucoma anormal que hemos definido como normal es de 32.

Modelo 2.

A partir de los pesos del modelo 1 descongelamos las últimas veinte capas pero dejando las capas de BatchNorm congeladas.

El f1 score obtenido es de 80.8 % y el número de personas con glaucoma anormal que hemos definido como normal es de 31

Modelo 3.

Entrenaremos el modelo anterior con todas las capas descongeladas.
El f1 score obtenido es de 84.4 % y el número de personas con glaucoma anormal que hemos definido como normal es de 24.

Modelo 4.

Creamos un nuevo modelo usando la aplicación de Keras VGG16. Utilizaremos los mismos parámetros que en los modelos anteriores para poder comparar las dos redes neuronales. Además como anteriormente utilizaremos el callback Early Stopping.

El f1 score obtenido es de 81.74 % y el número de personas con glaucoma anormal que hemos definido como normal es de 7

Modelo 5.

Para este modelo utilizaremos una red neuronal EffcientNet, pero en este caso la EfficientNetB2. Por otro lado, utilizaremos una técnica diferente, fijamos un EPOCHS de 100 y en este caso utilizaremos un callback de reduce lr y no el de EarlyStopping.

Destacar que podríamos haber elegido distintos modelos dentro de un gran abanico de posibilidades, incluso un EfficienteNet más elevado pero después de realizar distintas pruebas nos hemos decidido por este modelo por varias causas, entre ellas: por tiempos de ejecución y debido a que sl trabajar con imágenes de 224*224 y no reducir su tamaño los recursos computacionales son elevados.

El f1 score obtenido es de 91.9 % y el número de personas con glaucoma anormal que hemos definido como normal es de 10.

Modelo Final

Elegiremos el modelo 3 para este apartado, ya que obtenemos un valor de f1 score razonable, así como de loss, además de ser el modelo más estable. Para este modelo realizaremos la técnica de **cross validation**, validación cruzada. Esta técnica consiste en evaluar los resultados y garantizar que son independientes de la partición entre datos de entrenamiento y prueba.

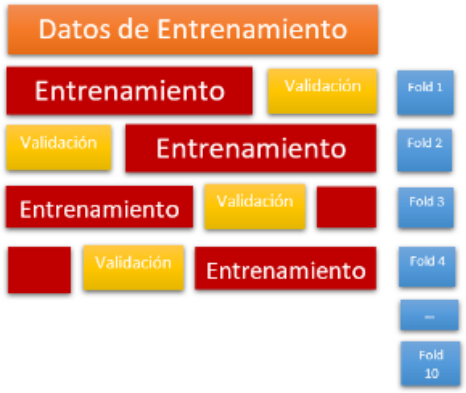


Figura 2: Método de Cross Validation de nuestro modelo

Entrenaremos el modelo de cero y lo evaluaremos 10 veces con diferentes imágenes para el conjunto de entrenamiento y validación, Como vemos en la figura 2. Ya disponemos de las imágenes guardadas de esta manera en cada directorio llamado fold, por ello, a continuación hemos cargado el modelo elegido y lo hemos entrenado para cada fold, de manera independiente, seguidamente hemos calculado la media de las métricas y comprobado que el modelo es consistente y se comporta bien con los datos de test, no existe overfitting, y por lo tanto es válido para cada directorio.

Resultados

Resumen de los resultados por modelos:

Modelo	**Definición y creación del modelo:**	**Entrenamiento**			**Test y resultados**	
		Parámetros utilizados:	Callbacks utilizados:	Tiempos de ejecución:	F1 Score	Falsos negativos:
Modelo 1	EfficientNetB0, modificando las últimas capas	Lr = 0.0005 Batch = 32	Early Stopping		79%	32
Modelo 2	EfficientNetB0 descongelamos las últimas veinte capas pero dejando las capas de BatchNorm congeladas.	Lr = 0.0005 Batch = 32	Early Stopping	221.6	80%	31
Modelo 3	EfficientNetB0 Descongelamos todas las capas	Lr = 0.0005 Batch = 32	Early Stopping	236.8	84.4%	24
Modelo 4	VGG16. descongelamos las últimas veinte capas pero dejando las capas de BatchNorm congeladas.	Lr = 0.0005 Batch = 32	Early Stopping	188.6	81.7%	7
Modelo 5	EfficientNetB2 descongelamos las últimas veinte capas pero dejando las capas de BatchNorm congeladas.	Lr = 0.0005 Batch = 32	Reducing Lr	1730.4	91.9%	10

Para más información por modelo consultar el Anexo.

El f1 score medio del modelo final es de aproximadamente 90%, realizando pruebas con otros modelos en el estudio hemos llegado a obtener valores algo superiores al 95% en concreto para EfficienNetB7 o B8 con tamaños de 32*32, el problema es que con estos modelos realizando el cross validation para determinados folds obtenemos f1 scores muy bajos así que hemos optado por un modelo con menores valores en las métricas pero con mayor estabilidad dando importancia a una menor desviación típica tanto como de f1 como del error.

Lo mismo ocurría si usábamos los modelos 4 y 5 descongelando todas las capas, habríamos tenido un mayor riesgo de overfitting y menos estabilidad si los hubiéramos usado para realizar la validación cruzada final.

	Fold_name	Loss	F1_Score
0	0	0.428648	0.867777
1	1	0.086543	0.976963
2	2	0.289543	0.905602
3	3	0.262663	0.918854
4	4	0.338792	0.906263
5	5	0.578037	0.814327
6	6	0.267800	0.884997
7	7	0.313607	0.909233
8	8	0.453673	0.815999
9	9	0.241049	0.927187

```
[INFO]: Final model stadistics:
        Fold_name       Loss   F1_Score
count   10.00000  10.000000  10.000000
mean     4.50000   0.326035   0.892720
std      3.02765   0.134974   0.049781
min      0.00000   0.086543   0.814327
25%      2.25000   0.263947   0.872082
50%      4.50000   0.301575   0.905932
75%      6.75000   0.406184   0.916449
max      9.00000   0.578037   0.976963
```

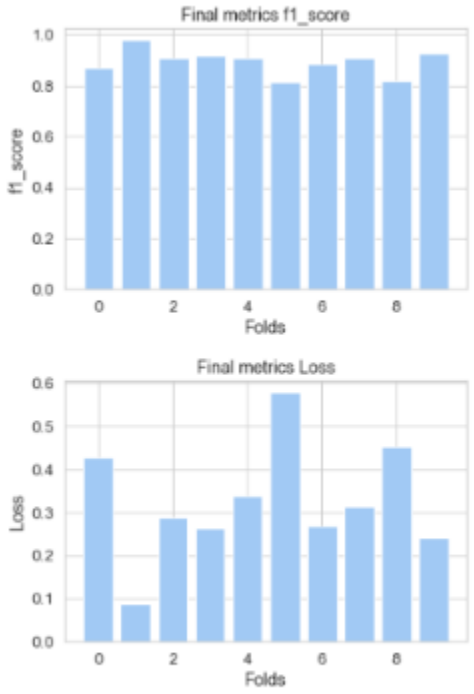


Gráfico 1: F1 Score y Loss por folds

Conclusiones y trabajos futuros

Conclusión

A pesar de que se trata de un artículo académico y que los recursos computacionales son limitados hemos conseguido obtener unos resultados consistentes, un f1 score media próxima al 90% . Con una GPU más potente, se podría entrenar este modelo con imagenes de 224*224 píxeles, con mayores capas descongeladas y redes neuronales más potentes y con ellos incrementar las métricas, pero nos encontramos en un importante punto de partida.

Los resultados nos muestran que hay ciertos tipos de imágenes de personas que tienen glaucoma que son más difíciles de interpretar por la red neuronal, en cambio a aquellas personas que no tiene glaucoma las detecta más fácilmente.

Trabajos Futuros

El objetivo final debe ser obtener el mínimo número de falsos negativos posibles, es decir que nuestro sistema no de por sanos a aquellas personas con glaucoma, para ello podríamos seguir entrenado el modelo con mayores cantidades de imágenes o complementar el modelo con otras técnicas de machine learning, añadiendo ciertos atributos de los pacientes o estudiando aquellos falsos negativos del modelo y conocer aquellas características que los hacen especiales.

Tendría especial utilidad la posibilidad de implementar estos modelos es sistemas que permitan realizar fotografías oculares como las que hemos trabajado en este proyecto y de esta manera poder realizar cribados masivos de detección del glaucoma, más rápidos y con menores costes.

Agradecimientos

Este proyecto no habría sido posible sin la ayuda de los profesores y compañeros del máster de Ciencia de Datos de la Universitat Oberta de Catalunya y especialmente para todos aquellos de la asignatura Deep Learning.

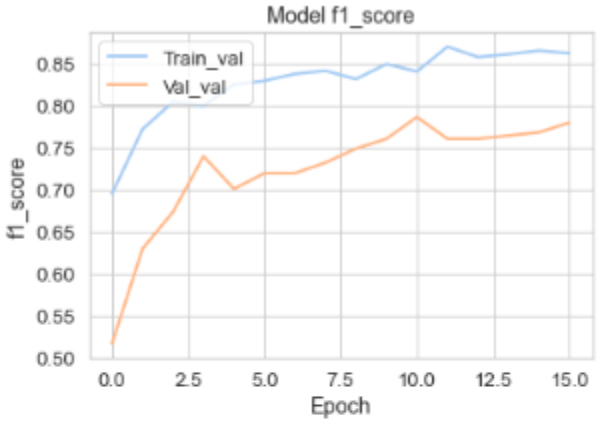
Referencias bibliográficas

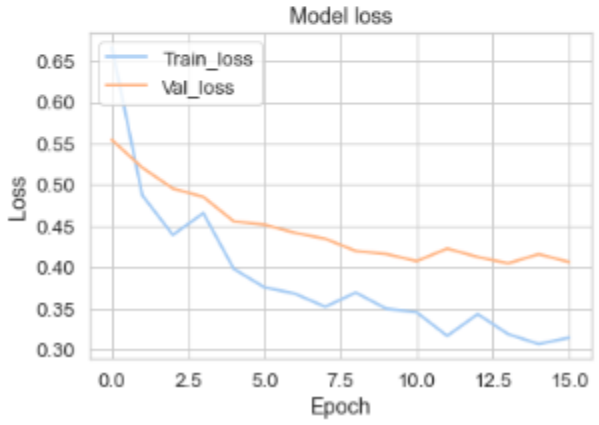
Deep Learning Principios y Fundamentos 2022, Anna Bosch Rué, Jordi Casas Roma, Toni Lozano Bagén
Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition, Aurélies Géron
Python Deep Learning, introducción práctica con keras y Tensor,Flow Jordi Torres
https://www.brightfocus.org/glaucoma/article/glaucoma-facts-figures

Anexo 1. Diferentes modelos utilizados y sus resultados:

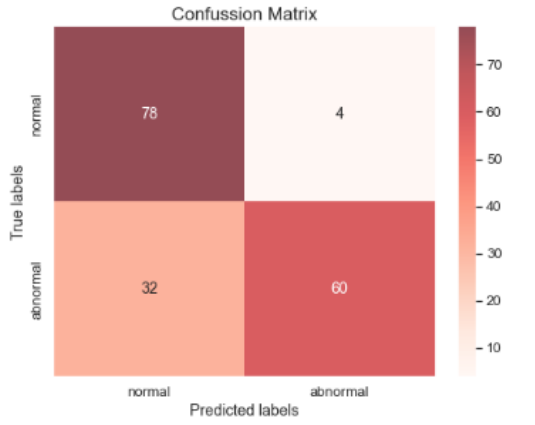
I) **Modelo inicial**

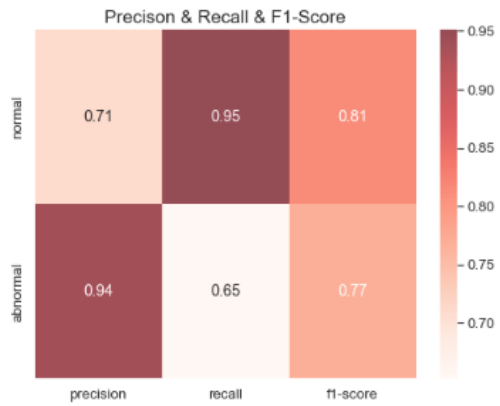
```
Model: "EfficientNetB0_G"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_layer (InputLayer)    [(None, 224, 224, 3)]     0

 efficientnetb0 (Functional)  (None, None, None, 1280)  4049571

 avg_pool (GlobalAveragePool  (None, 1280)             0
 ing2D)

 batch_norm (BatchNormalizat  (None, 1280)             5120
 ion)

 top_dropout (Dropout)       (None, 1280)              0

 activationLayer (Dense)     (None, 2)                 2562

=================================================================
Total params: 4,057,253
Trainable params: 5,122
Non-trainable params: 4,052,131
_________________________________________________________________
```






```
[INFO]: The model has a evaluate f1 score of: 0.7908653020858765
```

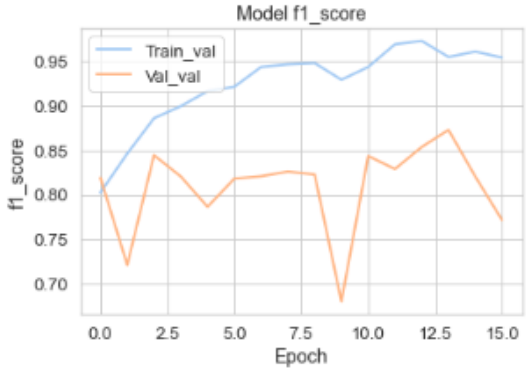


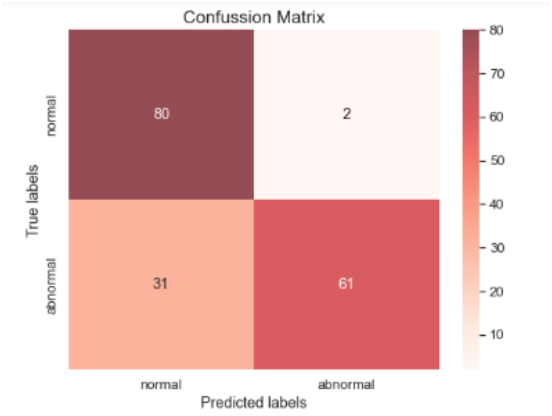


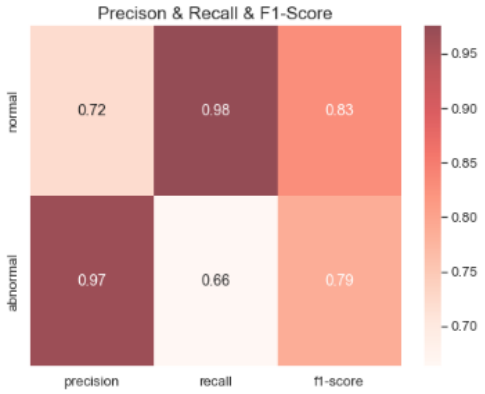
II) **Modelo 2.** Descongelamos las últimas 20 capa

```
Model: "EfficientNetB0_G"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_layer (InputLayer)    [(None, 224, 224, 3)]     0

 efficientnetb0 (Functional)  (None, None, None, 1280)  4049571

 avg_pool (GlobalAveragePool  (None, 1280)             0
 ing2D)

 batch_norm (BatchNormalizat  (None, 1280)             5120
 ion)

 top_dropout (Dropout)       (None, 1280)              0

 activationLayer (Dense)     (None, 2)                 2562

=================================================================
Total params: 4,057,253
Trainable params: 1,347,890
Non-trainable params: 2,709,363
_________________________________________________________________
```

```
[INFO]: Model evaluating....loss: 0.575090765953064, F1 score: 0.808056116104126.
```



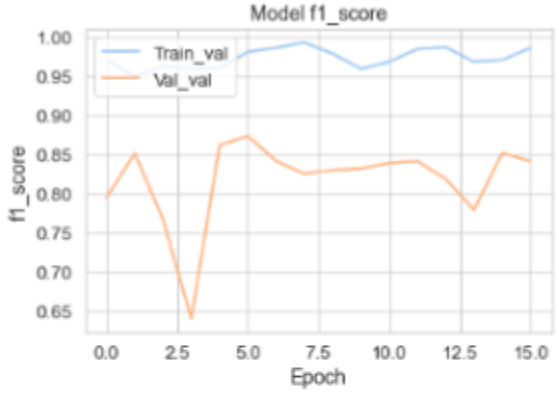


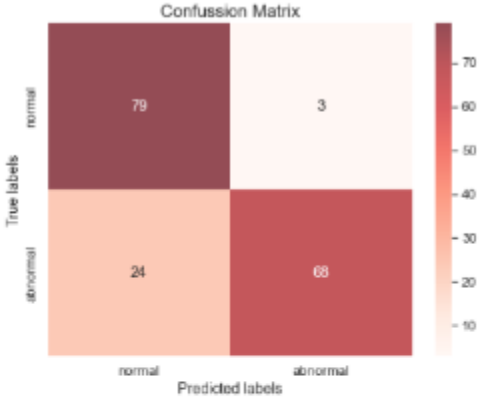


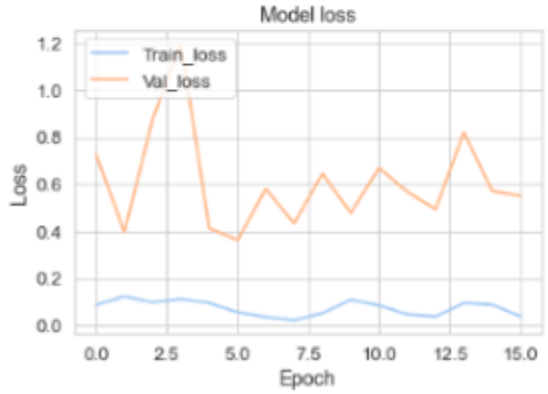
III) **Modelo 3.** A partir del modelo anterior descongelamos todas las capas del modelo y se entrena en su totalidad.

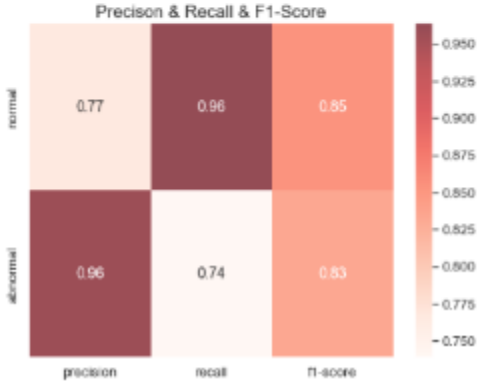
```
Model: "EfficientNetB0_G"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_layer (InputLayer)    [(None, 224, 224, 3)]     0

 efficientnetb0 (Functional)  (None, None, None, 1280)  4049571

 avg_pool (GlobalAveragePool  (None, 1280)             0
 ing2D)

 batch_norm (BatchNormalizat  (None, 1280)             5120
 ion)

 top_dropout (Dropout)       (None, 1280)              0

 activationLayer (Dense)     (None, 2)                 2562

=================================================================
Total params: 4,057,253
Trainable params: 4,012,670
Non-trainable params: 44,583
_________________________________________________________________
```

```
[INFO]: Model evaluating....loss: 0.5913786292076111, F1 score: 0.8442049026489258
```





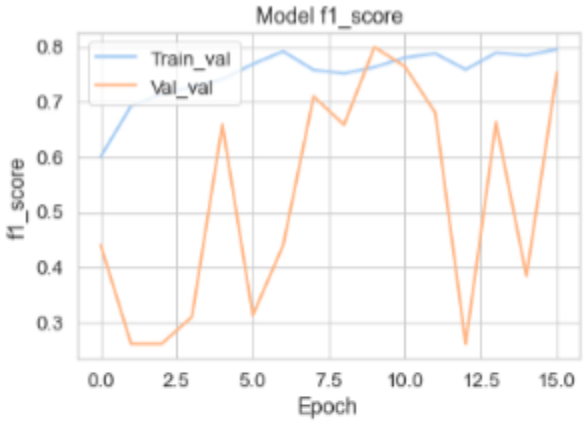


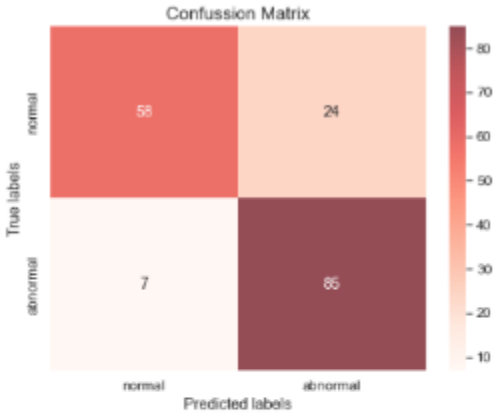


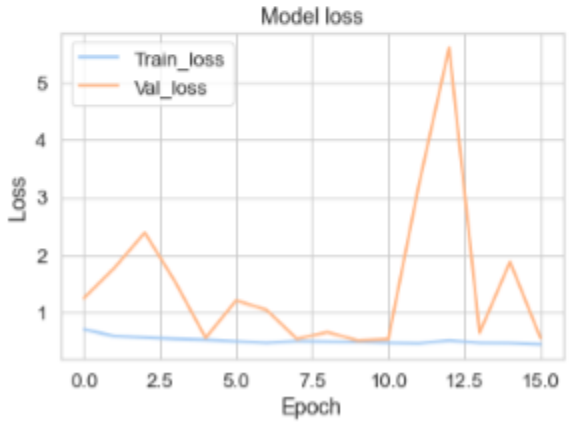
IV) **Modelo 4.** Modelo VGG16 con todas las capas descongeladas.

```
Model: "vgg16_Gloucoma"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_layer (InputLayer)    [(None, 224, 224, 3)]     0

 vgg16 (Functional)          (None, None, None, 512)   14714688

 avg_pool (GlobalAveragePool  (None, 512)              0
 ing2D)

 batch_norm (BatchNormalizat  (None, 512)              2048
 ion)

 top_dropout (Dropout)       (None, 512)               0

 activationLayer (Dense)     (None, 2)                 1026

=================================================================
Total params: 14,717,762
Trainable params: 14,716,738
Non-trainable params: 1,024
_________________________________________________________________
```

```
[INFO]: Model evaluating....loss: 0.45560017228126526, F1 score: 0.8174433708190918..
```



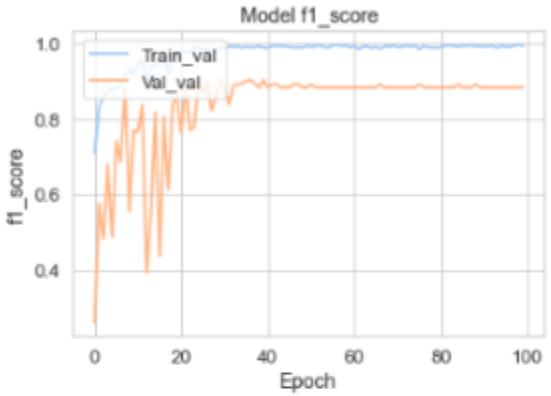


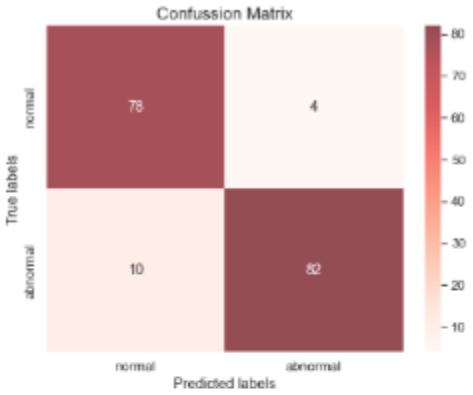


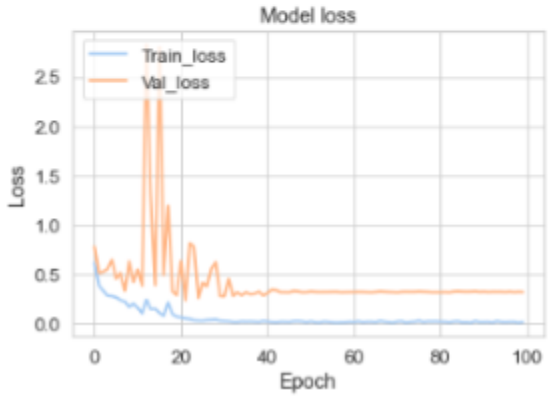
V) **Modelo 5.** Modelo EffcientNetB2 con todas las capas descongeladas con reducción del ratio de aprendizaje,

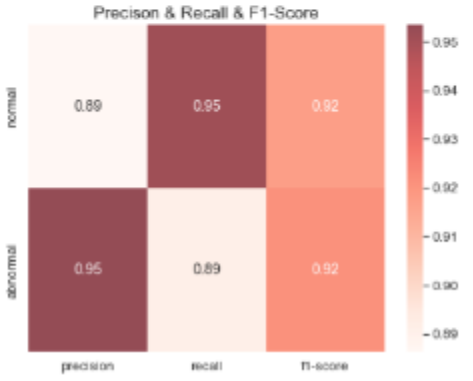
```
Model: "EfficientNetB2_Gloucoma"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_layer (InputLayer)    [(None, 224, 224, 3)]     0

 efficientnetb2 (Functional)  (None, None, None, 1408)  7768569

 avg_pool (GlobalAveragePool  (None, 1408)             0
 ing2D)

 batch_norm (BatchNormalizat  (None, 1408)             5632
 ion)

 top_dropout (Dropout)       (None, 1408)              0

 activationLayer (Dense)     (None, 2)                 2818

=================================================================
Total params: 7,777,019
Trainable params: 4,891,948
Non-trainable params: 2,885,071
_________________________________________________________________
```

```
[INFO]: Model evaluating....loss: 0.32399776577949524, F1 score: 0.9194977283477783..
```



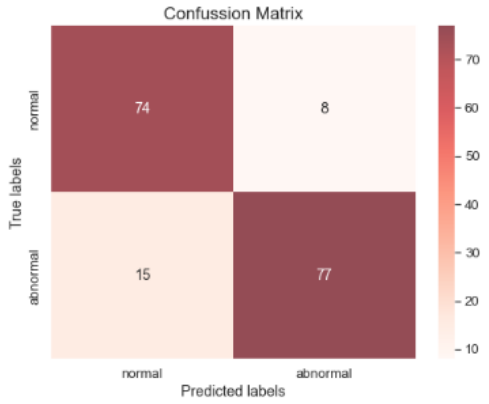


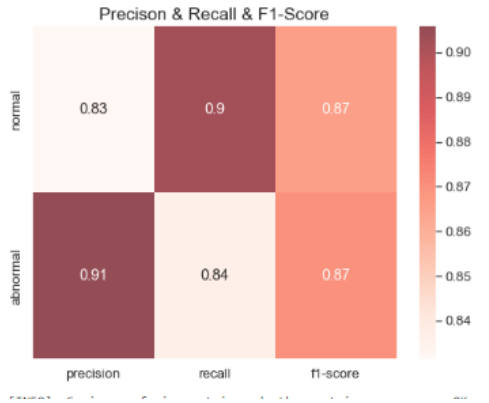




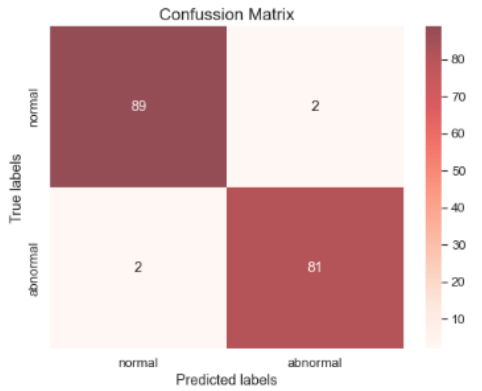
VI) Modelo Final

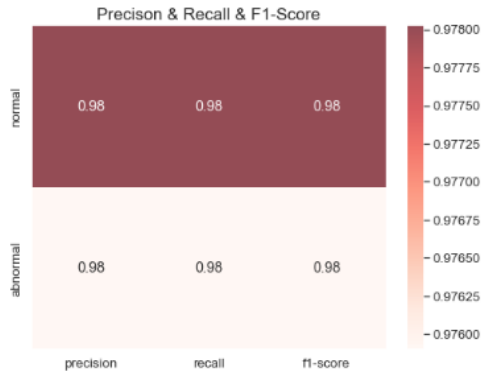
Fold 0



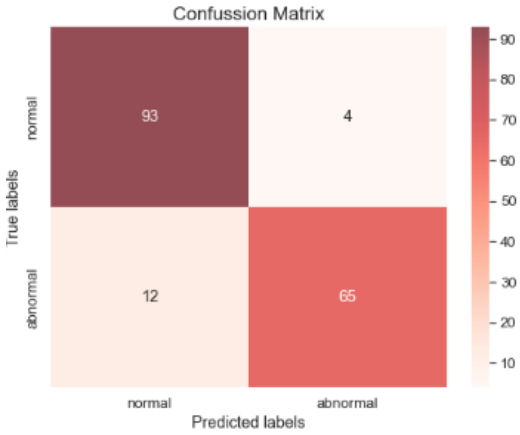


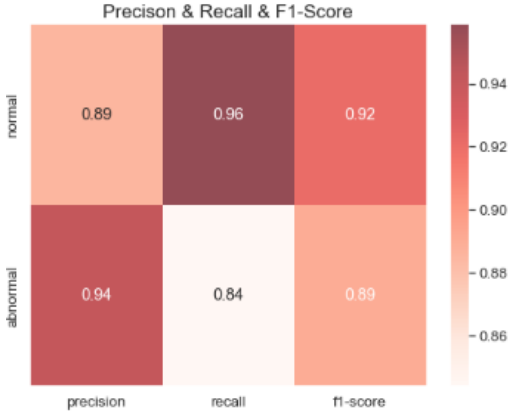
Fold 1



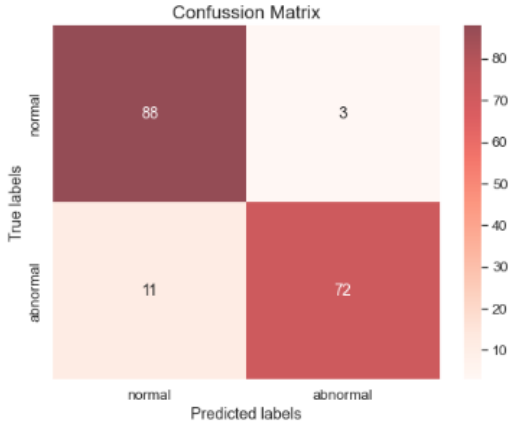


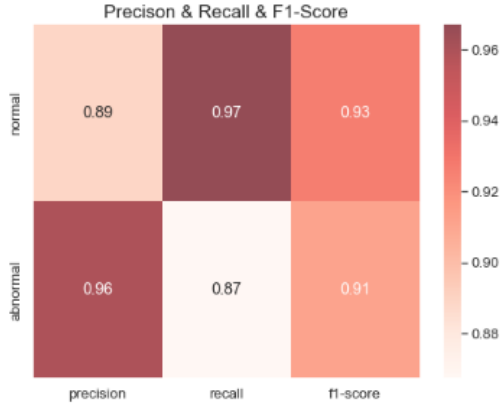
Fold 2



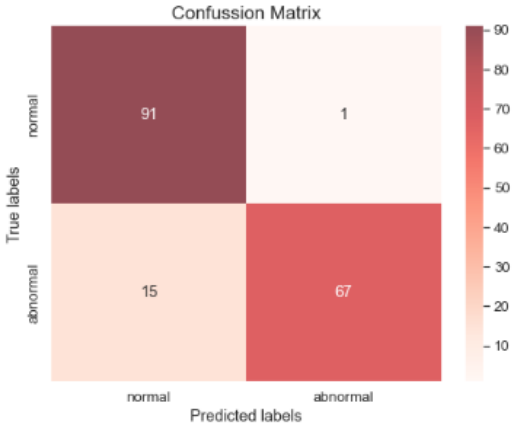


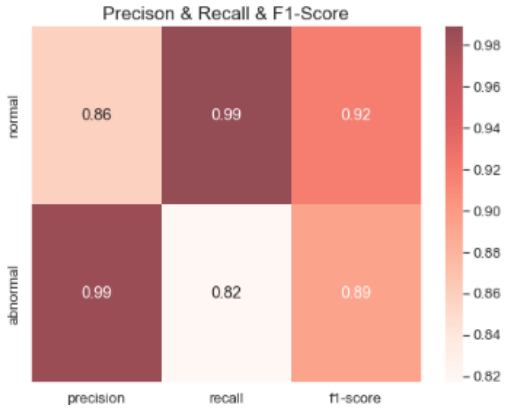
Fold 3



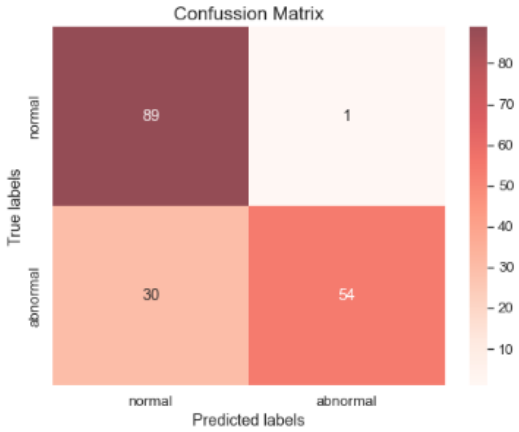


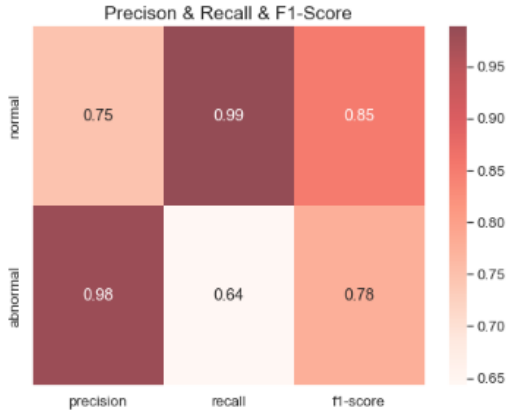
Fold 4



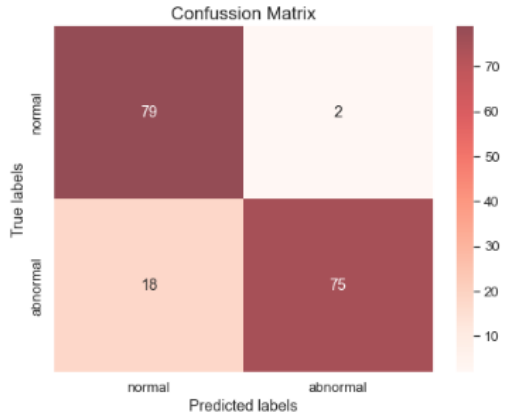


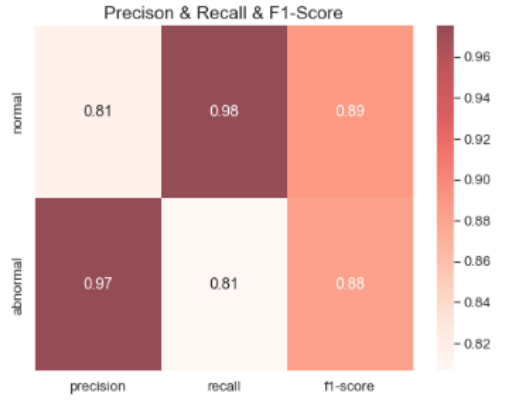
Fold 5



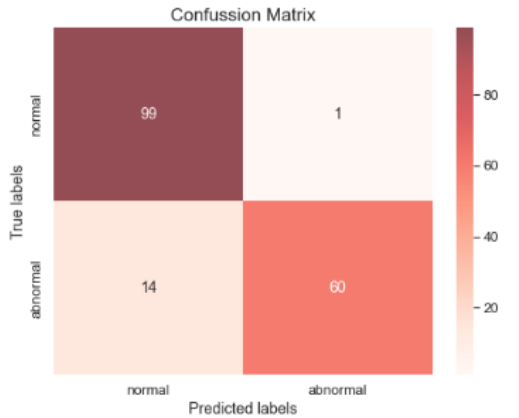


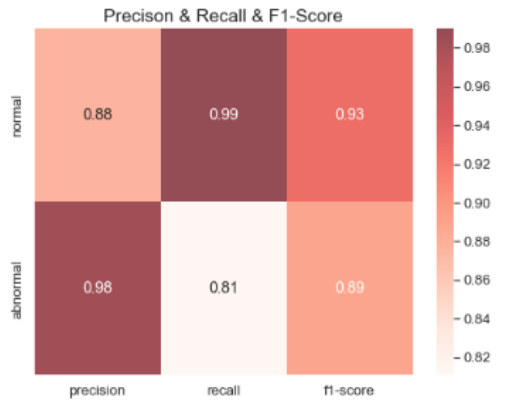
Fold 6



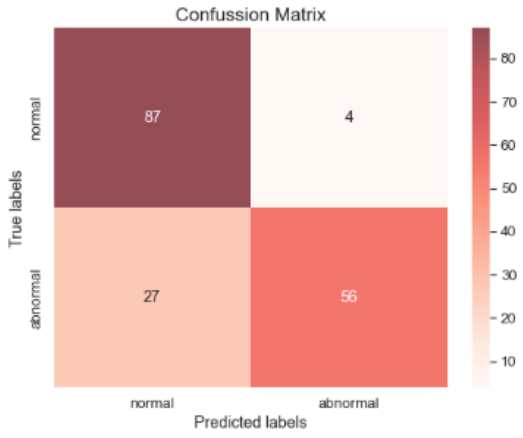


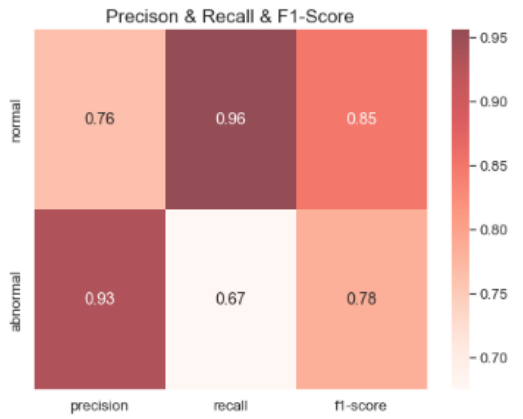
Fold 7



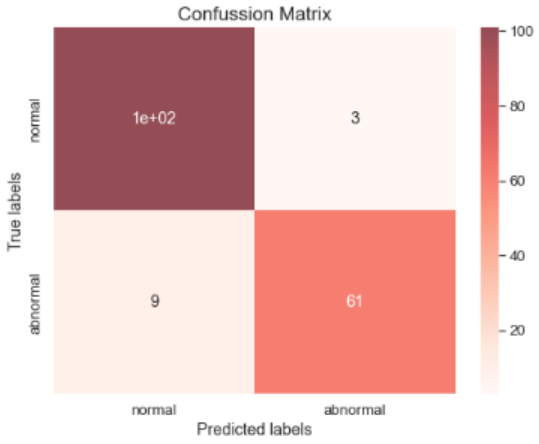


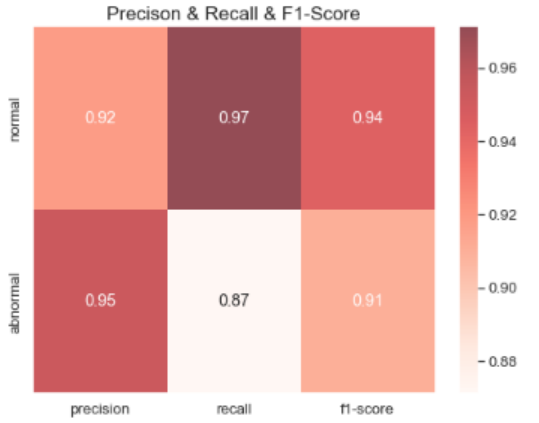
Fold 8


Confussion Matrix
True labels
normal
abnormal
87
4
27
56
normal
abnormal
Predicted labels
80
70
60
50
40
30
20
10


Precison & Recall & F1-Score
normal
abnormal
0.76
0.96
0.85
0.93
0.67
0.78
precision
recall
f1-score
0.95
0.90
0.85
0.80
0.75
0.70

Fold 9


Confussion Matrix
True labels
normal
abnormal
1e+02
3
9
61
normal
abnormal
Predicted labels
100
80
60
40
20


Precison & Recall & F1-Score
normal
abnormal
0.92
0.97
0.94
0.95
0.87
0.91
precision
recall
f1-score
0.96
0.94
0.92
0.90
0.88