Informe Final Práctica Deep Learning

Víctor Ayala Sánchez.

Introducción

El glaucoma es una enfermedad ocular degenerativa e irreversible, considerada como una de las principales causas de discapacidad visual en el mundo (Diaz-Pinto, Morales, Naranjo et al, 2019). Dicha enfermedad según la Organización mundial de la salud (citado en Diaz-Pinto, Morales, Naranjo et al, 2019), actualmente, afecta a 65 millones de personas en el mundo.

Al tratarse de una enfermedad asintomática, es importante su detección temprana con el objetivo de prevenir la pérdida de la visión. El presente trabajo pretende dar respuesta a la necesidad de detección mediante el uso de redes neuronales para la clasificación de imágenes oculares en normales o anormales.

Sección 1. Análisis Exploratorio de los datos

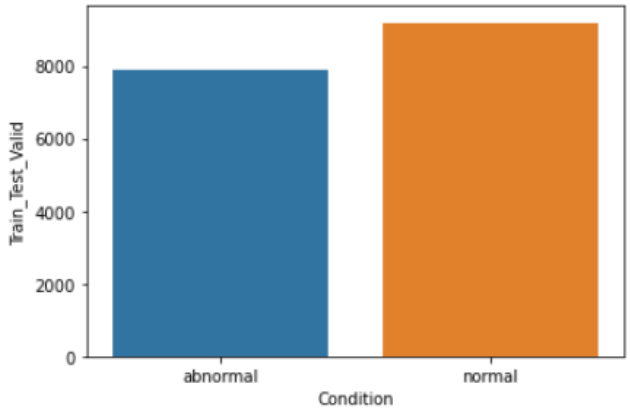
El conjunto de datos de imágenes oculares se encuentra compuesto por un total 17.070 registros. Estos se encuentran distribuidos en un sistema de directorios: se tienen 10 carpetas principales, cada una con una partición del conjunto; a su vez cada carpeta principal contiene 3 subdirectorios: uno con datos train, otro con datos de test y otro con datos de validación. Cada uno de los 3 subdirectorios comentados, a su vez con tiene dos subcarpetas: **normal** y **abnormal**, en el interior de estas se encuentran las imágenes correspondientes.

A continuación, se presenta la distribución de las imágenes en función de la condición del paciente y su grupo (test, train o valid):

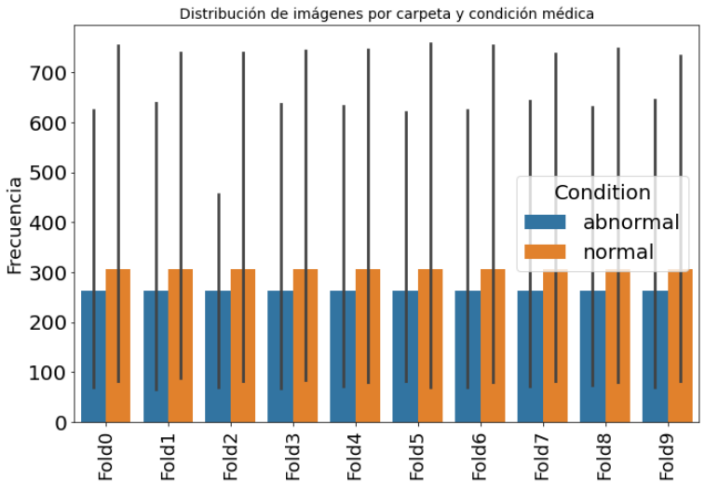
	Condition	Count	Total	
test	abnormal	821	1740	
	normal	919		
train	abnormal	6338	13790	
	normal	7452		
valid	abnormal	721	1540	
	normal	819		
				17070

		Count
Condition	abnormal	7880
	normal	9190
		17.070

En la tabla de la derecha podemos observar cómo se distribuye el número de imágenes en función de la condición del paciente. Acá podemos concluir que no tenemos un problema de clases desbalanceadas porque la proporción es cercana al 45% condición de enfermedad contra 55% normal.



Analizando la distribución de imágenes por carpeta principal, también observamos similitud en el número total en cada una:



Se concluyen que las carpetas son similares en relación con el número de imágenes que contiene.

Para la visualización de alguna imágenes se armó una función en Python que tomas todos los paths de las imágenes contenidas en todos los subdirectorios y arma una tabla tomando en cuenta de donde provienen: folder >> train/test/valid >> normal / abnormal; dando como resultado la siguiente tabla, de la cual se muestra solamente los primeros 5 registros:

	ImgsPaths	Folder	Train_Test_Valid	Condition
0	data/practica_DL_UOC_2022\Fold0/test/normal\agSTJamWau.jpg	Fold0	test	normal
1	data/practica_DL_UOC_2022\Fold0/test/normal\asxdlIWFGf.jpg	Fold0	test	normal
2	data/practica_DL_UOC_2022\Fold0/test/normal\BdKtGcRtgZ.jpg	Fold0	test	normal
3	data/practica_DL_UOC_2022\Fold0/test/normal\CaLpCWNeiQ.jpg	Fold0	test	normal
4	data/practica_DL_UOC_2022\Fold0/test/normal\CJerLIzywj.jpg	Fold0	test	normal

En adelante dicha tabla será referida como tabla base o dataframe base, debido a que contiene el path de cada imagen en la estructura de directorios y sus respectivas características.
Tomando una muestra de dicha tabla base, procedemos a visualizar las imágenes:

No fue necesario un preprocesamiento de imágenes o re escalado de las mismas dado a que la red neuronal base que se está utilizando fue entrenada con imágenes que se encontraban en la misma escala que las utilizadas en el presente estudio.

Sección 2. Entrenamiento de una red neuronal sobre una única partición

Para el problema de clasificación se ha propuesto una serie de modelos basados en redes neuronales a continuación se presentan las configuraciones y la búsqueda de los parámetros más adecuados para cada uno de los modelos. También es importante resaltar que los primeros modelos fueron alimentando con los pesos los modelos posteriores para obtener mejores resultados en el problema de clasificación.

Como modelo base, en la presente investigación se utilizó una red presentada en un paper por Tan y Le (2020). La familia de modelos EfficientNet fue propuesta como una solución equilibrada entre el coste computacional y la performance en los problemas de clasificación de imágenes.

Modelo 1.

Para el primer modelo, de acuerdo con las instrucciones de la práctica y para obtener modelos base sin consumir mucho tiempo y recursos computacionales, se filtró la tabla de imágenes base comentada con anterioridad y se filtró por aquellos registros correspondientes al folder0, luego se dividió en 6: X train, test y validación e y test, train y validación.

Posteriormente se definió una función que nos permite construir múltiples modelos y comprobar su performance para de esta forma obtener el mejor modelo 1 de esta sección. El código de la función es:

```
def HyperParameter_tunning_Model1(model, optimizer, learning_rate, metric, epochs,
                                  batch_size, X_train, y_train, X_test, y_test):
    start_time = time.time()
    baseModel = EfficientNetB0(include_top=False, weights="imagenet", classes=2, input_shape = (224,224,3))
    baseModel.trainable = False
    model_1 = baseModel.output
    model_1 = GlobalMaxPooling2D()(model_1)
    model_1 = layers.BatchNormalization()(model_1)
    model_1 = Dropout(0.20)(model_1)
    model_1 = Dense(2, activation='softmax')(model_1)
    model_1 = Model(inputs=baseModel.inputs, outputs = model_1)
    optimizer = optimizer(learning_rate=learning_rate)
    model_1.compile(optimizer=optimizer, loss="binary_crossentropy", metrics=[metric])
    epochs = epochs
    batch_size = batch_size
    hist_model = model_1.fit(X_train,y_train,epochs=epochs,
                             validation_data=(X_test,y_test),
                             batch_size=batch_size,verbose=0)
    pred_y = model_1.predict(X_test)
    target_names = ["Normal", "Abnormal"]
    cm = pd.DataFrame(confusion_matrix(y_test.argmax(axis=1),
                                       pred_y.argmax(axis=1)), index= target_names, columns=target_names)

    Accuracy = accuracy_score(y_test.argmax(axis=1),pred_y.argmax(axis=1))
    F1_Score = f1_score(y_test.argmax(axis=1), pred_y.argmax(axis=1), average='macro')
    end_time = time.time() - start_time
    results = {"Model Configuration": {'Optimizer': optimizer,
                                'Learning Rate': learning_rate,
```

```
                                'Metric': metric,
                                'Epochs': epochs,
                                'Batch Size': batch_size},
            'Model': model_1,'History': hist_model,
            'Matriz de Confusión': cm,'Accuracy': Accuracy,
            'f1_score': F1_Score,'Time Seconds': end_time
        }
        return results
```

La función se encarga de descargar el modelo base, configurar todas las capas de la red a no "trainable", entrenar el modelo sobre el batch size, optimizador y ratio de aprendizaje escogido sobre los conjuntos de entrenamiento y test pasados como argumentos.

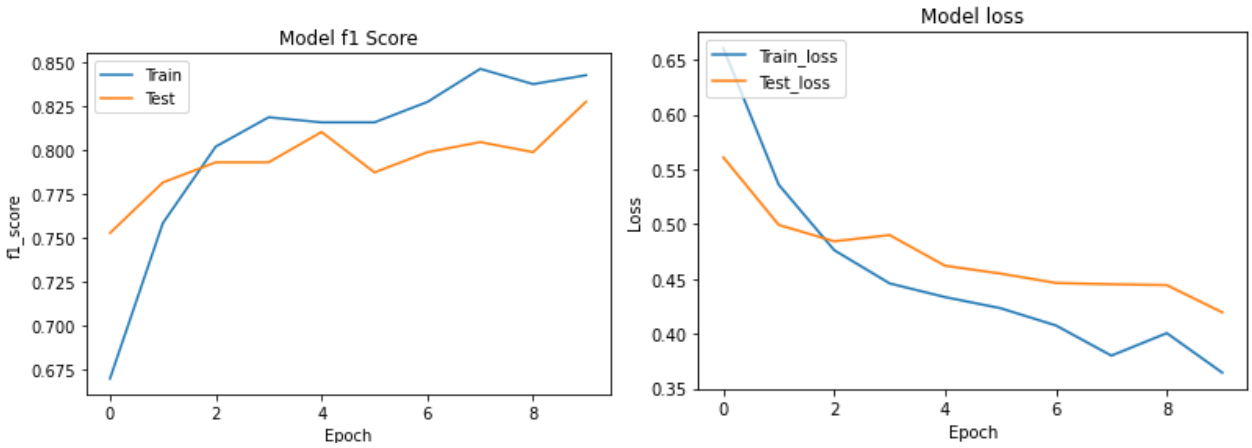
Haciendo uso de la anterior función y generado un conjunto de listas, con diferentes valores de cada parámetro:

```
lr = [0.0001,
      0.001]
opt = [
       tf.keras.optimizers.Adam,
       tf.keras.optimizers.SGD,
       tf.keras.optimizers.Adadelta,
       tf.keras.optimizers.Adagrad
       ]
batch_sizes = [16,32]
epochs = [10,50]
```

Se arman una serie de loops para comprobar todos los parámetros y verificar aquellos que son óptimos para este primer modelo base. Analizando la función presentada, podemos observar que retorna el nombre del optimizador, el accuracy, el f1-score, batch size, épocas y tiempo total en segundos que ha tomado entrenar el modelo. A continuación, se presenta el top 5 modelos 1 ordenados por el F1 Score y el accuracy:

Optimizer	Learning Rate	Epochs	Batch Size	Accuracy	F1_Score	Time
Adam	0.0010	10	32	0.856322	0.856089	39,8
Adam	0.0010	50	32	0.850575	0.850397	156,7
SGD	0.0010	50	16	0.839080	0.838740	172,4
Adam	0.0010	10	16	0.839080	0.838547	43,1
Adam	0.0010	50	16	0.821839	0.821550	174,2

Como es evidente en la tabla presentada, los modelos que involucran la mejor performance son aquellos que tienen como optimizador Adam y learning rate de 0.0010. Con esta información se corre un modelo 1 final, del cual se guardarán los pesos para alimentar a los modelos de la siguiente sección. El modelo final uno, entrenado con los siguientes parámetros: learning rate de 0.0010, batch size de 32, optimizador Adam, 10 épocas y todas las capas del modelo base seteada a False en trainable.



Observando la evolución a través de las épocas se evidencia que el F1 score puede mejorar y la que aún puede disminuir la pérdida. No se evidencia overfitting aún debido a que la performance es similar en el train y test.

	precision	recall	f1-score	support
Normal	0.77	0.90	0.83	82
Abnormal	0.90	0.76	0.82	92
accuracy			0.83	174
macro avg	0.83	0.83	0.83	174
weighted avg	0.84	0.83	0.83	174

	Normal	Abnormal
Normal	74	8
Abnormal	22	70

Si analizamos el reporte de clasificación y la matriz de confusión, observamos que el modelo base es más preciso clasificando las muestras anormales que las normales, el recall es mayor para las muestras normales y las métricas de accuracy y F1-Score son moderadamente buenas. Teniendo en cuenta el número de épocas, tomamos esta red como un punto de partida para los siguientes modelos.

Modelo 2.

Para el juego de modelos 2, modificamos la función presentada con anterioridad y añadimos una línea para cargar los pesos previamente guardados del modelo 1 con mejor performance y además configuramos a True aquellas últimas 20 capas de la red que no sean de batch normalization:

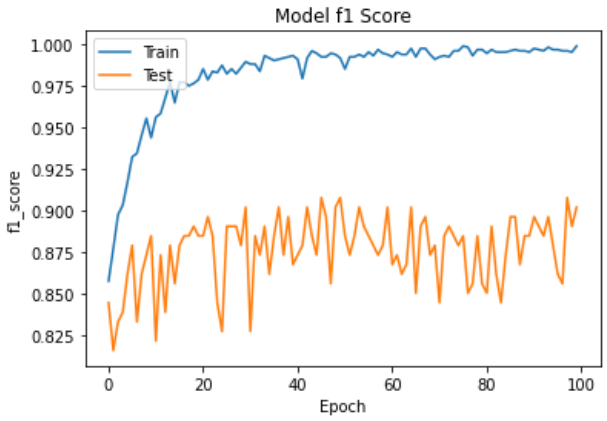
```
#Activamos las últimas 20 capaz excepto las Batch Normalization
    r = re.compile(".*bn$")
    for layer in baseModel.layers[-20:]:
      layerName = layer.name
      if r.match(layerName):
        layer.trainable = False
        #print(layer.name)
      else:
         layer.trainable =True
```

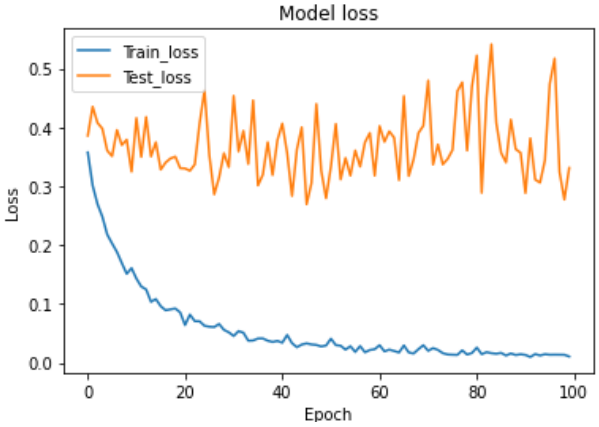
```
model_2.load_weights("Model_Results/Model1_Final_Wgts.hdf5")
```

En esta segunda búsqueda de hiperparámetros se utilizaron learning rates de 0.0001 y 0.001, los optimizadores Adam y SGD, batch sizes de 16,32 y 54 y número de épocas de 10, 50 y 100.
A continuación, se presenta una tabla del top 5 modelos ordenados por F1 Score y accuracy:

Optimizer	Learning Rate	Epochs	Batch Size	Accuracy	F1_Score	Time
Adam	0.0001	100	64	0.913793	0.913790	298,684
Adam	0.0001	50	64	0.908046	0.908034	156,198
Adam	0.0010	100	16	0.908046	0.908034	348,041
Adam	0.0010	50	64	0.908046	0.907997	154,524
Adam	0.0001	50	16	0.902299	0.902296	180,142

Se visualizan incrementos significativos en los tiempos de entrenamiento debido al número de épocas, al igual que se observan incrementos en el accuracy el F1 score de los modelos. El mejor modelo y el que se ha tomado como modelo final de este segundo paso utiliza Adam como optimizador, un learning rate del 0.0001, 100 épocas, batch size de 64 y tarda unos 300 segundos en entrenamiento. A continuación, se presentan los gráficos de F1 Score, Loss, reporte de clasificación y la matriz de confusión del modelo óptimo de esta sección:





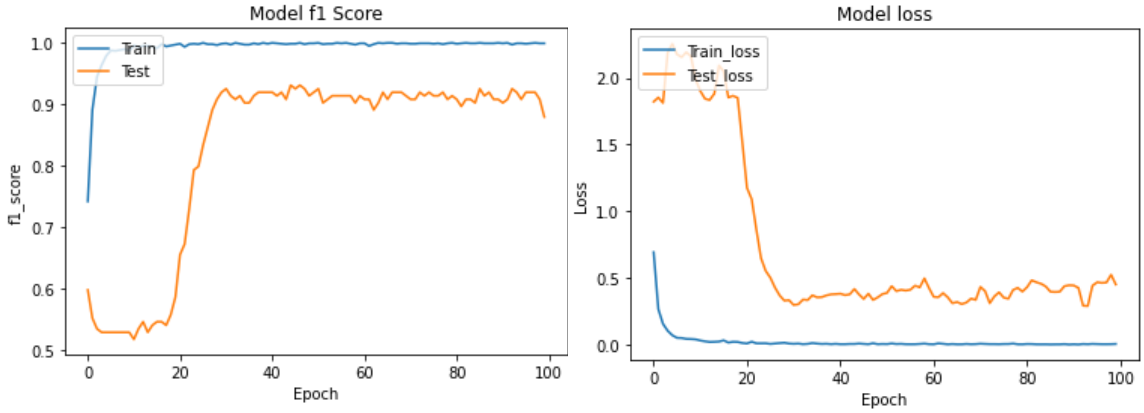
	precision	recall	f1-score	support
Normal	0.85	0.96	0.90	82
Abnormal	0.96	0.85	0.90	92
accuracy			0.90	174
macro avg	0.91	0.91	0.90	174
weighted avg	0.91	0.90	0.90	174

	Normal	Abnormal
Normal	79	3
Abnormal	14	78

Se ven mejoras significativas con respecto al modelo final 1, hay una mayor precisión sobre las muestras normales, así como también puntajes más altos en el F1 score y en el accuracy. También se observan menor cantidad de falsos negativos en la clase normal, lo cual es excelente para este modelo, porque sería el tipo de error (error tipo 1) que queremos evitar; es decir dar una falso de diagnóstico de normalidad cuando en realidad hay presencia de enfermedad.

Modelo 3.

Se realizó el mismo procedimiento para el juego de modelos 3, pero esta vez se modificó la función comentada con anterioridad para encender a trainable todas las capas de la red y cargar los pesos del modelo final 2. Se tomó el modelo con mayor performance a partir de la tabla de resultados con los siguientes parámetros: learning rate de 0.00010, batch size de 64, 100 épocas, los pesos del modelo final 2. Se presentan los resultados de dicho modelo.



En el modelo 3 final, tenemos evidencia de overfitting debido al comportamiento de las curvas del F1 socre y de pérdida. El accurracy y F1 Score, en general tienen valores aceptables. Por lo que este modelo puede ser elegido para la sección 3.

	precision	recall	f1-score	support
Normal	0.82	0.96	0.88	82
Abnormal	0.95	0.85	0.88	92
accuracy			0.88	174
macro avg	0.89	0.88	0.88	174
weighted avg	0.89	0.88	0.88	174

	Normal	Abnormal
Normal	78	4
Abnormal	17	75

Modelo 4.

Los modelos 4 y 5, de libre configuración fueron propuestos de la siguiente manera: en el caso del modelo 4: luego de las capas del modelo base configuradas a True para el atributo trainable se sumaron 3 conjuntos de capas con la siguiente arquitectura: una capa del tipo GlobalMaxPooling2D, una capa de Batch Normalization y una capa DropOut al 0.20. Se modificó la función utilizada para las anteriores búsquedas de parámetros teniendo como mejores modelos los presentados:

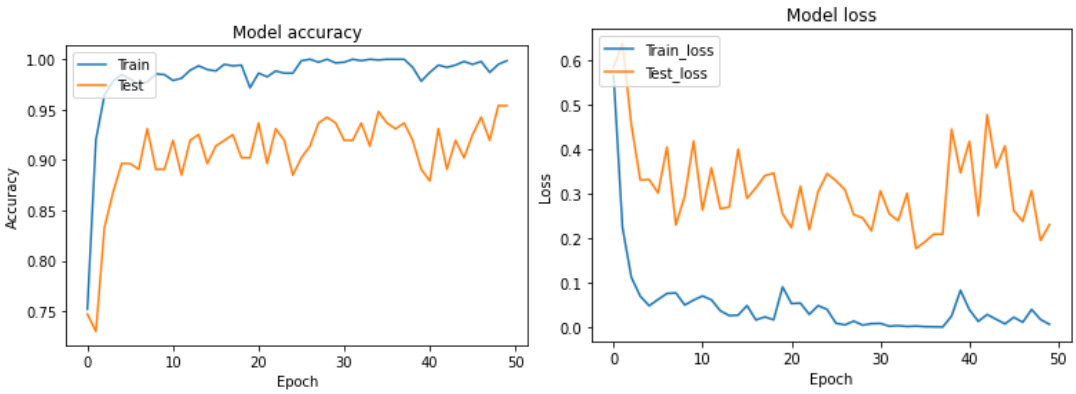
Optimizer	Learning Rate	Epochs	Batch Size	Accuracy	F1_Score	Time
Adam	0.0001	50	16	0,9253	0,9252	695,940
Adam	1,00E-06	50	16	0,9080	0,9080	705,545
Adam	1,00E-05	50	16	0,9080	0,9080	693,290
Adam	1,00E-06	50	64	0,9023	0,9023	641,599
Adam	1,00E-05	50	64	0,9023	0,9023	637,259
Adam	0.0001	50	64	0,8793	0,8793	632,737

Se observan ciertas ganancias en las métricas de F1 Score y Accuracy, pero el tiempo de entrenamiento de cada modelo es alto.

Modelo 5.

Por último, para el modelo 5, se utilizó la EfficientNetB7, con los pesos de imagenet, las capas del modelo base configuradas a trainable True, agregando además una capa de GlobalMaxPooling2D, una de batch Normalization, de Drop Out al 0.20 y una la de salida. Este único modelo se entrenó con un learning rate al 0.00010; 50 épocas y un batch size de 16.

Pese a tener un tiempo de entrenamiento muy elevado (cercano a la hora con 15 minutos), los resultados de esta red neuronal son excelentes:



	precision	recall	f1-score	support
Normal	0.94	0.96	0.95	82
Abnormal	0.97	0.95	0.96	92
accuracy			0.95	174
macro avg	0.95	0.95	0.95	174
weighted avg	0.95	0.95	0.95	174

	Normal	Abnormal
Normal	79	3
Abnormal	5	87

Este último modelo mantiene una precisión muy elevada en ambas clases, un accuracy y F1-score elevado y una tasa baja de falsos negativos en relación con la clase de ausencia de enfermedad. También hay evidencia de cierto nivel de overfitting por el comportamiento de las curvas en el gráfico de F1 Score.

Sección 3. Validación cruzada y discusión

Pese a que los modelos 3 y 4; tienen excelente puntuación en las métricas de F1-Score y Accuracy, debido a los tiempos de entrenamiento se decidió la configuración del modelo 3 con los pesos

correspondientes al modelo final 2. La configuración final fue: el modelo base EfficienteNetB0, con todas las capas configuradas a entrenables, sumando una capa de GlobalMaxPooliong2D, una de Batch Normalization, una DropOut al 0.20 y la capa de salida; un learning rate de 0.00010, 50 épocas y un batch size de 16.

Para el entrenamiento del modelo con cada set de datos, se modificó la función presentada en el modelo 1, para que aceptara el nombre de la carpeta, construya la arquitectura del modelo y retorne los datos de haber testeado la red con los diferentes directorios para test y validación. Se presentan los resultados por directorio:

Accuracy	F1 Score	Accuracy Validacion	F1 Score Validacion	Time Seconds	Folder
0,874	0,874	0,916	0,914	532,54	Fold0
0,943	0,942	0,903	0,902	529,12	Fold1
0,902	0,900	0,922	0,921	534,65	Fold2
0,856	0,855	0,903	0,902	531,80	Fold3
0,914	0,913	0,903	0,902	531,52	Fold4
0,879	0,879	0,961	0,961	525,48	Fold5
0,920	0,919	0,935	0,934	528,78	Fold6
0,948	0,946	0,942	0,941	532,10	Fold7
0,897	0,896	0,909	0,908	527,93	Fold8
0,925	0,922	0,864	0,863	533,00	Fold9

Como se observa en la tabla por directorio, todos los tiempos de entrenamientos estuvieron por el orden los 530 segundos, el accuracy con los conjuntos de test entre 0.87 y 0.94 al igual que el F1 Score, se presentan además las mismas métricas, pero con los conjuntos de validación, donde se observa un excelente performance del modelo, teniendo en cuenta que son datos que no se usaron en ningún momento para el entrenamiento.

El promedio del F1 score entre todos los directorios es del 0.90 en el caso de los conjuntos test y 0.91 para validación. Se concluye es altamente efectivo para discriminar en las fotografías a los pacientes con enfermedad y los pacientes sanos.

	count	mean	std	min	25%	50%	75%	max
Accuracy	10	0,906	0,030	0,856	0,884	0,908	0,924	0,948
f1_score	10	0,905	0,030	0,855	0,883	0,907	0,921	0,946
Accuracy_Validacion	10	0,916	0,027	0,864	0,903	0,912	0,932	0,961

f1_score_Validacion	10	0,915	0,027	0,863	0,902	0,911	0,931	0,961
Time Seconds	10	530,693	2,773	525,485	528,864	531,659	532,431	534,652

Sección 4. Análisis crítico

A.

Una estrategia diferente de entrega de los datos correspondientes a las imágenes hubiese sido tener solamente 2 directorios: X e Y; en el primero todas las imágenes del conjunto de datos y en el segundo el nombre del fichero de la imagen correspondiente a X con su etiqueta, normal o anormal. De esta forma es más fácil o práctico generar un único dataframe con todas las imágenes. Posteriormente se hubiese tomando una pequeña muestra de validación y luego se hubiese realizado un Split en train y tes para X e Y. Teniendo las estructuras mencionadas y haciendo uso de la función gridsearchCV() de scikit-learn se replicarían los folders con la ventaja que configurando el parámetro njobs = -1 el entrenamiento sería en diferentes nodos, aprovechando así del procesamiento en paralelo para ser más eficientes computacionalmente.

Es importante realizar particiones y diferentes folders o procesos de entrenamiento, porque de esta manera el modelo nunca es entrenado con todos los datos disponibles, previniendo de esta forma que el sobre ajuste de la red al conjunto de datos. Si esto sucede el modelo pierde capacidad de generalización y por lo tanto de predicción sobre nuevos datos que puedan estar disponible; es por esta misma razón que también se guarda una muestra pequeña de validación, para demás de predecir con el test, realizar pruebas con muestras que nunca fueron pasadas al modelo en el proceso de entrenamiento.

B. Como principales resultados tenemos:

- La transferencia de pesos de modelos previamente entrenados suma a la performance de los modelos.
- Es necesario probar más de un modelo de la familia de redes de Efficient.
- Al escoger una arquitectura de red final se debe mantener un equilibrio entre tiempo de entrenamiento y métricas de evaluación.
- Data augmentation puede ser una estrategia válida cuando tenemos a disposición un bajo número de muestras. En el caso de la presente investigación, al tener cerca de 17K muestras no se requirió.

- En general, una aproximación adecuada a la construcción del modelo es ir pasando por diferentes pasos en los que se toman los parámetros más adecuados y que se han probado que funcionan con el conjunto de datos y con la arquitectura planteada.
- El tiempo de procesamiento es alto, es necesario realizar este tipo de ejercicio en entornos cloud.
- El error que se debe minimizar en el clasificador es el tipo 1, es decir un falso positivo con respecto a la clase normal. Si este caso se da, significa que el algoritmo ha clasificado como sana a una persona que realmente no lo está. Una métrica adecuada para la medición es justamente el F1-Score, que representa un KPI que engloba la precisión y el recall del modelo. Además de lo comentado, también es importante tener una alta precisión al detectar a los pacientes anormales.

Para la observación a detalle de los resultados obtenidos por modelo y por fase se adjunta un directorio con 5 ficheros CSV, donde se pueden encontrar los resultados de todos los modelos entrenados en la búsqueda de parámetros adecuados.

Referencias

- Diaz-Pinto, A., Morales, S., Naranjo, V. *et al.* CNNs for automatic glaucoma assessment using fundus images: an extensive validation. *BioMed Eng OnLine* **18,** 29 (2019). https://doi.org/10.1186/s12938-019-0649-y
- Tan, M. & Le, Q. EfficientNet (2020). Rethinking Model Scaling for Convolutional Neural Networks. https://arxiv.org/pdf/1905.11946.pdf