Desafío 1

Types of colorectal histology images

Álvaro Sirvent Navarro

74531046Y  sn26@alu.ua.es

**Parte básica**

En este primer apartado sobre el primer desafío, vamos a ver cada una de las cosas que hemos realizado para implementar lo que sería la parte básica de la práctica.

Dado que tendremos que completar cada uno de los puntos que aparecen en el enunciado, nosotros, vamos a ir mirando cada uno por separado.

Clasificación Binaria

En primer lugar, para realizar la clasificación binaria de si hay un tumor o no, realmente, no tendremos que tocar nada, y es que, con el ejemplo que nos encontrábamos en el enunciado ya lo tendríamos hecho.

De esta manera, para poder poner la clasificación binaria de las imágenes, lo que simplemente tendremos que hacer, es poner la variable global de **nb\_classes** al valor 2 (Si lo pusiésemos a 8, podríamos clasificar los 8 tipos diferentes de imágenes).

Aplicación de validación cruzada 10-CV

En segundo lugar, y entrando en cosas que realmente sí que nos ha llevado bastante más trabajo, nos encontraríamos con lo que sería la validación cruzada.

La validación cruzada, nos servirá una vez la tengamos implementada, para poder crear conjuntos partidos de datasets, donde los datos que dejamos para el entrenamiento y los que dejamos para las pruebas, son **completamente independientes** entre sí. Y, de esta manera, cuando nosotros hacemos pruebas para comprobar los resultados que obtenemos mediante un modelo de entrenamiento, podremos estar seguros, que **los conjuntos están bien separados**, y, que, en nuestros conjuntos **no se encuentran ningún tipo de dato repetido entre ellos**.

Es decir, nunca encontraremos una imagen que se encuentre en el conjunto de entrenamiento y en el de prueba al mismo tiempo.

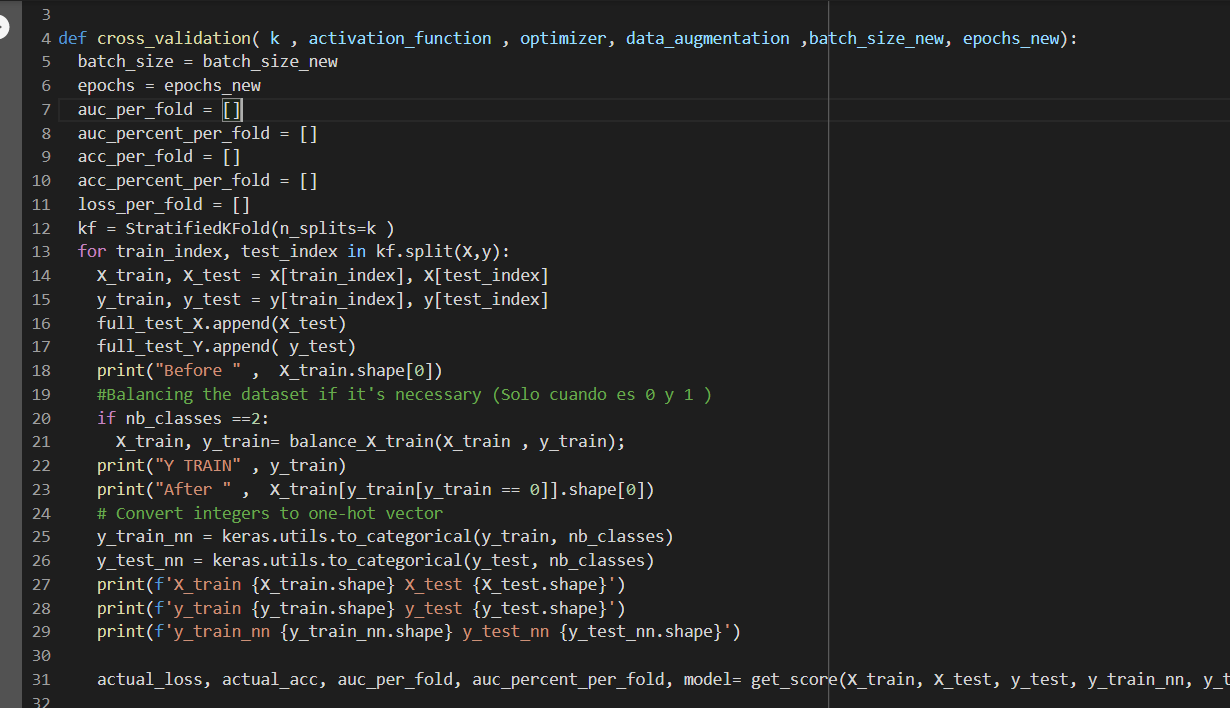
Así mismo, nosotros indicaremos el porcentaje que queremos dejar para el conjunto de test. Y, como bien nos indica en el enunciado, dejaremos un 10% para el conjunto de test, que nos llevará a tener 10 iteraciones en el kfold, esto será 10 conjuntos de entrenamiento y de prueba, formando pares que se separan de manera efectiva (Que, por cada iteración del bucle, tendremos una separación de los conjuntos donde ninguna imagen se repite en los dos conjuntos).

Cabe recalcar, que para la implementación de la validación cruzada diseñada en mi práctica, me he basado en los conceptos que se explican en el siguiente enlace.

<https://www.youtube.com/watch?v=gJo0uNL-5Qw&list=LL&index=3>

Y, como veremos a continuación en las diferentes capturas que adjuntamos sobre el código, mi validación cruzada, está implementada dentro de una función a la que he llamado **cross\_validation**.

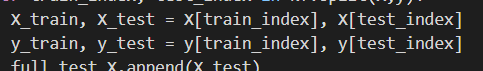
Dicho esto, podemos pasar a ver la implementación que hemos realizado para la validación cruzada.

Como podemos apreciar en esta primera captura adjuntada, lo que nos encontraremos, es el esquema propio de la validación cruzada.

Donde, lo que nos importa es ver cómo nos creamos primeramente un **StratifiedKFold**, con el número de Splits que queremos. En nuestro caso, se decidió por hacer que ese número viniese por parámetros, pero, básicamente, para cumplir con el enunciado y con lo que realmente tendría sentido, lo que hacemos, es poner ese valor de entrada a 10, que sería ese 10% que dejamos para los tests.

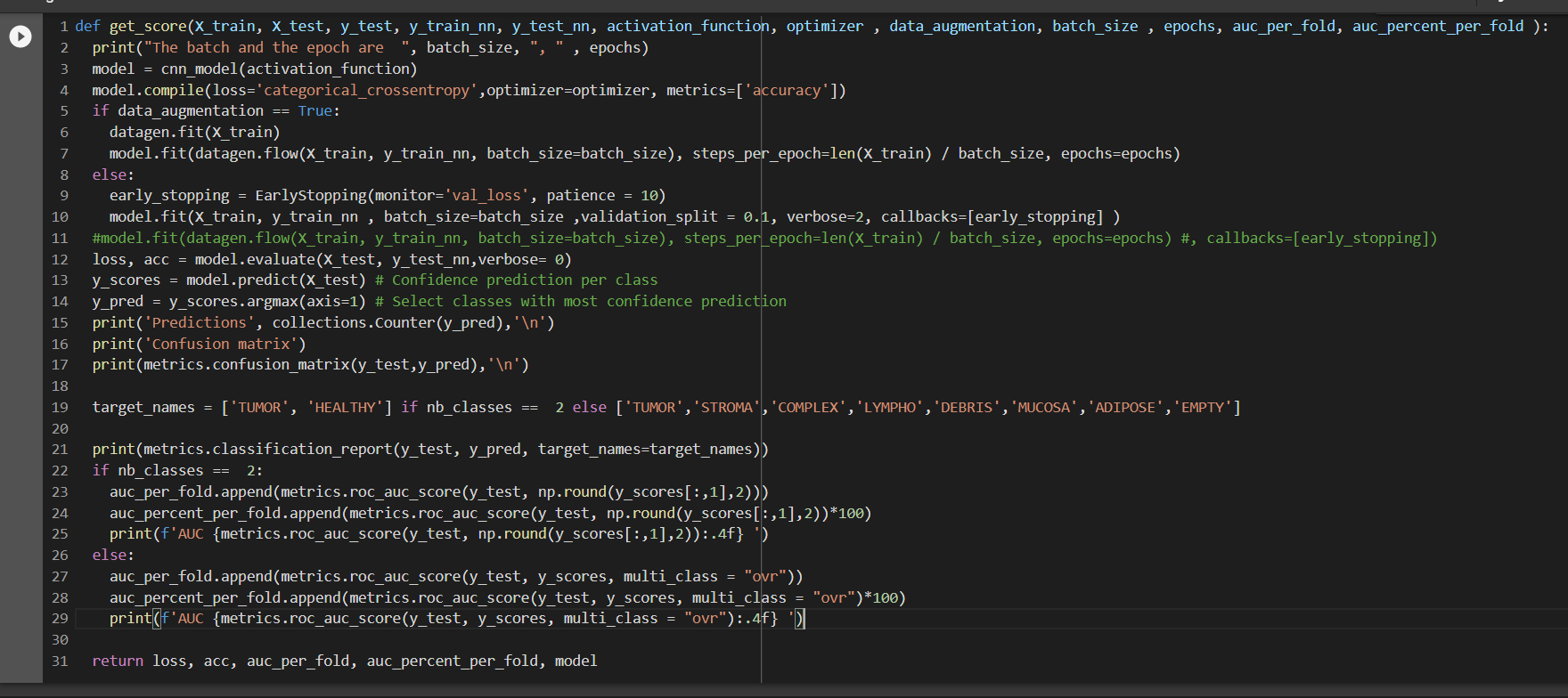
Seguidamente, tendríamos que sacar los rangos dentro del dataset inicial, para el conjunto de datos de entrenamiento y el de test para ese fold. Que es lo que nos podemos encontrar en el bucle for que aparece en la imagen.



De igual modo, para sacar los arrays con sus respectivos conjuntos, haremos lo siguiente.

Posteriormente, para poder realizar el entrenamiento del modelo y evaluarlo sobre el conjunto de test proporcionado por el fold en el que nos encontramos, tendremos que llamar a otra función que nos hemos creado en una celda del código, a la que hemos llamado **get\_score**.

Así mismo, dentro de esta función, nos encontraremos con dos tipos de entrenamiento del modelo distintos, dependiendo de si tenemos que realizar el aumentado de datos o no, del cual hablaremos más adelante.



Ahora bien, como podemos apreciar en el interior de esta función, cada vez que se llama a esta misma función, crearemos un nuevo modelo, que entrenaremos, lo evaluaremos con el conjunto de test y sacaremos sus resultados (Auc, acc y loss).

Así mismo, dependiendo de si hay que hacer un aumentado de datos o no, entrenaremos el modelo de una forma u otra.

Por otro lado, en el caso de que queramos hacer un aumentado de datos y estemos trabajando en binario, tendremos que hacer un par de acciones adicionales, que son las que vamos a pasar a comentar a continuación.

ImageDataGenerator

Para realizar el aumentado de datos en la parte básica, que nos permitirá tener más imágenes para el entrenamiento, cuando trabajamos en binario, nos encontraremos con que el conjunto de datos con el que estamos trabajando se encuentra **desequilibrado**, y, por esto mismo, no resulta tan sencillo el aumentado de datos.

Esto, se debe a que si nosotros hiciéramos el aumentado de datos sobre el conjunto como nos viene, directamente seguiríamos teniendo muchas más imágenes con un valor de 1 (no hay tumor), que de imágenes con un valor de 0 (sí hay tumor).

Haciendo que nuestro modelo identificase casi todas las imágenes que le pusiéramos para test, como imágenes donde no hay tumor, porque es de lo que más ha aprendido.

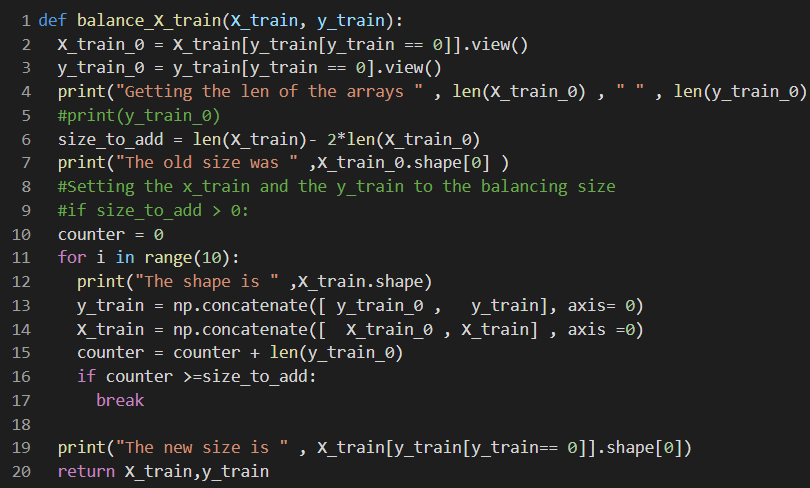
Precisamente por esto, es por lo que cuando nosotros realizamos el aumentado de datos en la parte básica, primero de realizar este paso, tendríamos que balancear las clases.

También, aunque no lo hemos mencionado, cabe recalcar, que para el balanceado, sólo lo podemos hacer con el conjunto de datos de entrenamiento, porque si nosotros lo hiciéramos en el de test, estaríamos metiendo **ruido** en el conjunto, y eso haría que la evaluación que hacemos del modelo, no fuese correcta.

De esta manera, nosotros nos hemos creado una función con el nombre de **balance\_X\_train.**

Que es a la que llamamos en la función **cross\_validation** justo después de sacar los dos conjuntos.



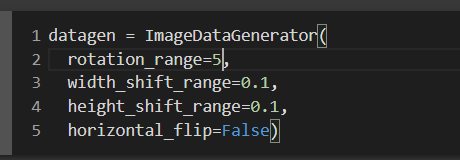


Si nos fijamos en lo que se ha incluido en su interior, veremos como lo que hacemos, es sacar dos arrays de los que son 0 que tenemos tanto en el conjunto de datos de la imagen, como en el de las etiquetas.

Y con esto hecho, **calcularemos la diferencia de 0 y de 1** que hay en el conjunto de entrenamiento. Para así, tener un bucle que va **concatenando los 0** que hemos sacado del conjunto de entrenamiento, al mismo conjunto de entrenamiento, y cortar cuando ya hemos llegado al tamaño que hace la diferencia entre uno y otro.

Y una vez balanceado, sí que podríamos pasar a realizar el aumentado de datos.

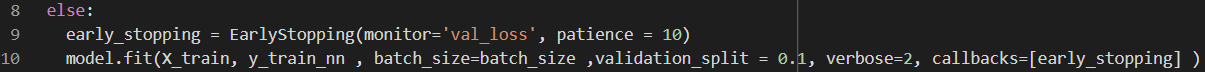
Para el aumentado de datos, nos hemos creado una instancia de la clase **ImageDataGenerator**, en una celda aparte, donde le hemos puesto los valores que nos han ofrecido mejores resultados en las pruebas.



Con esto hecho, ya nos iríamos nuevamente a donde teníamos la función en la que se entrena al modelo, para poder realizar el aumentado.

Aquí, lo que haríamos, es primeramente, entrenar a la instancia que nos acabamos de crear del ImageDataGenerator, con el conjunto de datos de entrada.

Como siguiente paso, entrenaremos al modelo con la función Flow del ImageDataGenerator en su interior.



Una vez entrenado con la función Flow, los siguientes pasos, pasarían por evaluar el modelo y sacar los resultados, pasos que son compartidos con el sin aumentado de datos.

Wilcoxon

Wilcoxon, será el siguiente paso que tendremos que realizar en el apartado básico y que nos permitirá comprobar pasándole dos resultados dados por dos modelos distintos, cuál es mejor de los dos.

Por lo que, para definir la función de Wilcoxon que nos permita hacer esto, nosotros, nos hemos definido la siguiente función.

def wilcoxon(x,y):

  wilcox\_V, p\_value = wilcoxon(x, y, alternative='greater',

zero\_method='wilcox', correction=False)

  print('Resultado completo del test de Wilcoxon')

  print(f'Wilcox V: {wilcox\_V}, p-value: {p\_value:.2f}')

  return wilcox\_V, p\_value

Si nos fijamos en lo que hacemos, veremos que realmente, es muy simple de realizar este apartado, dado que solamente tendremos que llamar a la función con los dos arrays de resultados, para ver cuál de los dos es mejor.

De esta manera, sabremos que el primero es mejor que el segundo, en el momento en el que tengamos un **p value menor al 0.05**, rechazando la hipótesis nula.

Por otro lado, con el **Wilcox\_V,** hallaremos cuál es la **diferencia aproximada entre los dos conjuntos**.

A continuación, veremos cómo hemos usado esta función para la realización de varias pruebas y la obtención de conjeturas.

Estudio de la métrica AUC

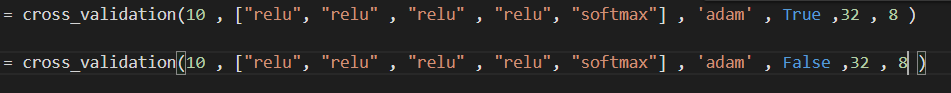
Para este primer apartado básico, estudiaremos la métrica AUC de cada uno de los modelos que vayamos haciendo, y como esto es una primera toma de contacto, lo que vamos a hacer, es probar con el aumentado de datos, y sin el aumentado de datos, para ver cuál de los dos nos da mejores resultados, y, posteriormente, usaremos Wilcoxon para ver que efectivamente uno es mejor que el otro.

Así mismo, destacamos, que, para realizar este paso, y los posteriores que nos vamos a ir encontrando, hemos tenido que modificar la función de **cross\_validation**, para que nos devuelva los diferentes arrays con los resultados. Que serán los que vamos a usar para comparar en este nivel y en posteriores.



Dicho esto, nos hemos creado una celda para probar los dos modelos que hemos comentado anteriormente.

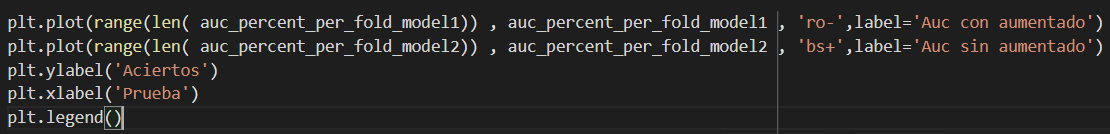
En el primero, lo hemos hecho con aumentado de datos, y el segundo, sin el aumentado.



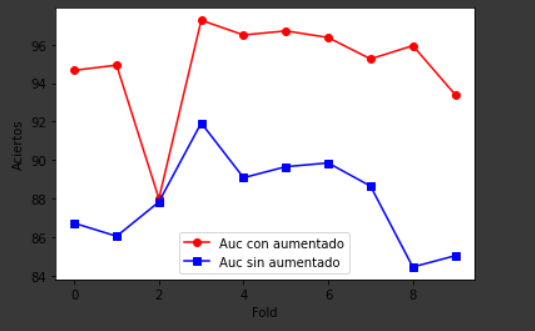
Que es lo que podemos ver al final.

De igual modo, los dos modelos, los hemos probado con un batch de 32, y con 8 épocas.

Además, mediante el uso de la siguiente celda, sacaremos una gráfica comparativa entre lo dos.

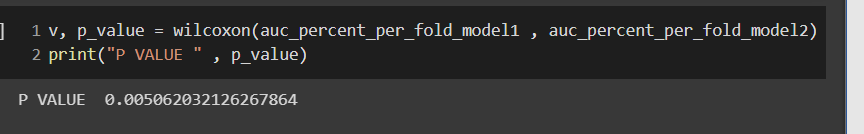


Y luego, usaremos la función que nos hemos definido de Wilcoxon, para sacar el resultado.

Como veremos, esto nos devuelve el siguiente resultado.

Aquí, se puede apreciar que efectivamente, cuando aplicamos un aumentado de datos, la métrica Auc aumenta considerablemente en casi todos los Fold.

De igual modo, haciendo uso de la función de Wilcoxon que nos hemos definido anteriormente, podríamos ver como el valor del **P value**, está por debajo del 0.05, por lo que, podríamos rechazar la hipótesis nula, demostrando que sí existe una diferencia entre los dos modelos, y que el primer modelo con el aumentado es mejor que el segundo.



De igual modo, en apartados siguientes, estudiaremos más en profundidad otras gráficas que hemos realizado, así como otras comparaciones entre modelos.

**Parte Media**

En esta segunda parte del desafío, vamos a ver lo que hemos ido implementando, para cumplir con cada uno de los puntos que aparecen en el enunciado.

Al igual que para el apartado anterior, hablaremos de cada uno de los puntos que aparecen en el enunciado, sobre esta segunda parte media del desafío.

Clasificación con todas las categorías

De nuevo, para cumplir con esta parte del enunciado, lo único que hemos tenido que hacer, es irnos nuevamente a la variable que nos encontramos al inicio, llamada **nb\_classes**, y ponerla a 8.

Esto, nos permitirá, que mediante la comprobación de su valor en las diferentes partes del código, podamos preparar el modelo para contemplar los 8 tipos diferentes de imágenes que nos encontramos en el conjunto de entrada.

Preprocesado de las imágenes

Para llevar a cabo este segundo apartado del nivel medio, hemos investigado varias formas de poder realizar el preprocesado de las imágenes, y al final, lo que he encontrado que cumplía con lo que se buscaba para llevar a cabo este punto de la práctica, es haciendo uso de la api **skimage**, más concretamente, del módulo **exposure**, que nos proporcionará ciertas funciones para el procesado de imágenes.

En nuestro caso, se han usado solamente tres funciones de ajuste, que son las que hemos enumerado a continuación.

* Ajustar la expresión logarítmica de la imagen (**exposure.adjust\_log**)
* Ajustar el contraste de la imagen (**exposure.adjust\_sigmoid**)
* Ajustar el gamma de la imagen (**exposure.adjust\_gamma**)

Para cada uno de ellos, se han ido probando varios valores para ver cuáles dejaban las imágenes de una mejor manera.

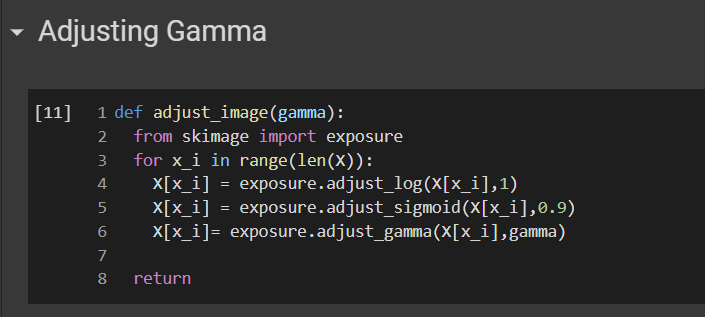
No obstante, en el que más nos hemos fijado a la hora de hacer el preprocesado, es en el valor del gamma, del cual, más adelante, en el apartado de la evaluación de las métricas, veremos los resultados que hemos ido obteniendo con los valores que hemos ido probando.

Esto, se debe, a que usualmente, los valores que se suelen utilizar para la función log por ejemplo, son de 1 o 0, y en mi caso, hice ciertas pruebas con el 0, y los resultados no eran los esperados.

De igual modo, con la operación del contraste de la imagen, pasaba algo similar, por lo que, el mejor valor, lo hemos encontrado dejando el 0.9 como estándar.

Como hemos dicho, el que hemos ido tocando, ha sido el gamma, y para sacar los valores que hemos utilizado para el gamma, se ha hecho uso de la siguiente guía, donde podemos ver diferentes valores dependiendo de la imagen utilizada.

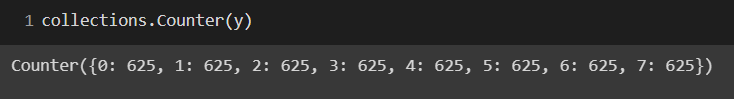
<https://www.cambridgeincolour.com/tutorials/gamma-correction.htm>



Aumentado de datos con ImageDataGenerator

Al igual que como hemos visto en el apartado anterior, en este apartado, nuevamente usaremos las funciones que habíamos usado en el básico para crear más imágenes de entrada.

Aunque, en este caso, y como veníamos adelantando en el apartado básico, no tendremos que hacer uso de la función que realiza un balanceado previo. Esto, sucede gracias a que nos encontramos con un conjunto donde ahora sí, todas las clases están balanceadas, como se puede apreciar en la siguiente imagen.



Por lo que, para realizar este apartado, nuevamente entrenaríamos al modelo con la función Flow del ImageDataGenerator, tal y como aparece a continuación.

if data\_augmentation == True:

    datagen.fit(X\_train)

    model.fit(datagen.flow(X\_train, y\_train\_nn, batch\_size=batch\_size), steps\_per\_epoch=len(X\_train) / batch\_size, epochs=epochs)

Uso de la notación funcional de Keras

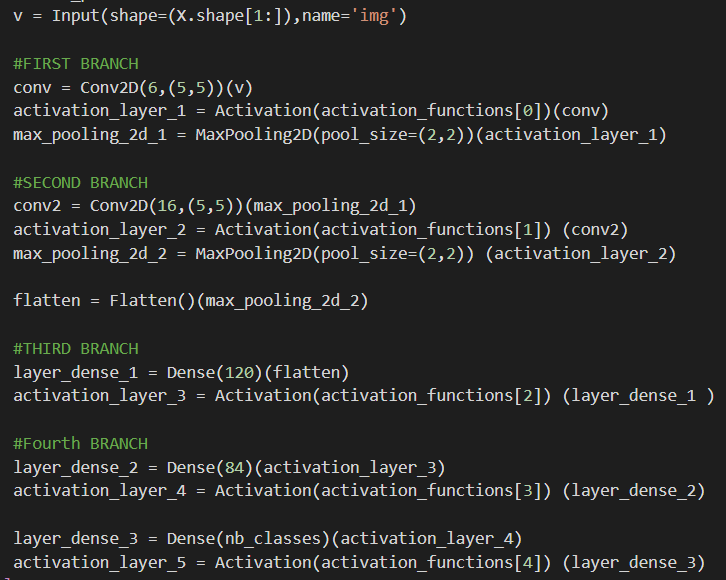
En este apartado del enunciado, lo que tendremos que hacer, es cambiar el modelo inicial, a uno nuevo donde hagamos uso de las notaciones funcionales de Keras.

Para este apartado, nosotros hemos hecho uso de la guía de la Api de Keras, donde se nos explica cómo hacer la notación funcional con Keras, y cuáles son las ventajas que nos brinda al hacerlo de este modo.

Lo podemos encontrar en el siguiente enlace.

<https://www.tensorflow.org/guide/keras/functional?hl=es-419>

De esta manera, basándonos en el modelo inicial que nos incluía el ejemplo, hemos podido sacar el siguiente modelo.



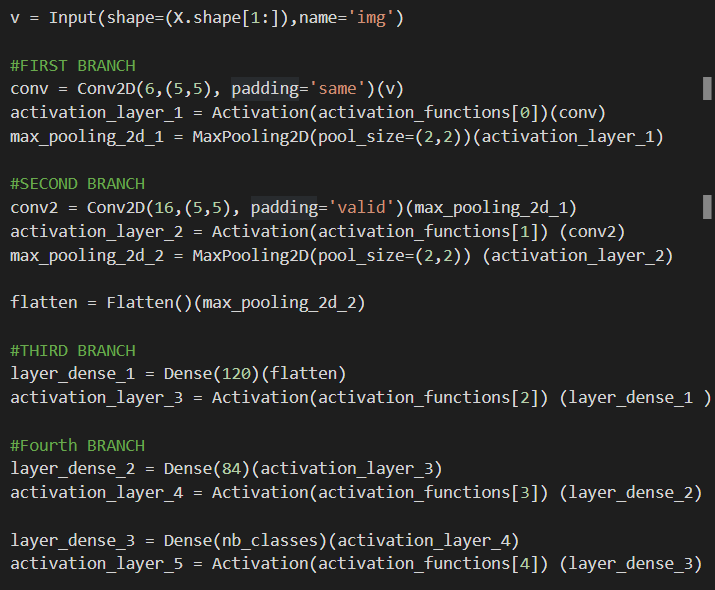


Que como vemos, realmente es similar al que teníamos anteriormente, solo que con la Api funcional de Keras, lo que nos permite realizar, es modelos más dinámicos que con Sequential, donde podemos tener modelos con topología no lineal, con capas compartidas, y con múltiples entradas o salidas.

Pero vamos, que el nuestro, mantiene el mismo funcionamiento que el que teníamos en un principio, aunque, sí que es verdad, que hemos desarrollado también otro modelo similar a este, pero incluyendo el parámetro padding en las capas de Conv2D.

Básicamente, mediante el uso del parámetro padding, lo que haríamos, es decirle si queremos que el output de esa capa tenga el mismo tamaño que el input, o un tamaño menor. Esto, se lo decimos con el valor ‘**same’**, o con el valor ‘**valid**’.

De manera que, el segundo modelo, se nos quedaría de la siguiente manera.



Estos dos modelos que hemos incluido, los compararemos más adelante, como hemos hecho en apartados anteriores, aunque el cambio es minúsculo.

Uso de SHAP para ver las partes de la imagen en las que se centra el algoritmo

Como hemos visto en el enunciado de la práctica, también se nos pide hacer uso de la Api Shap para ver una vez entrenado el modelo, en qué partes se centra, y sacar conclusiones más claras del por qué toma las decisiones que toma.

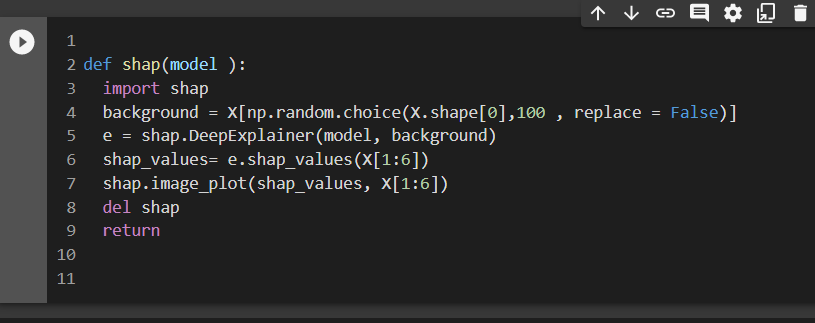
Para hacer este apartado, he hecho uso de la documentación que nos podemos encontrar sobre la propia Api en Github.

Aquí, se nos explican diferentes modelos para hacer una implementación correcta de lo que estamos buscando, y así poder obtener los resultados esperados.

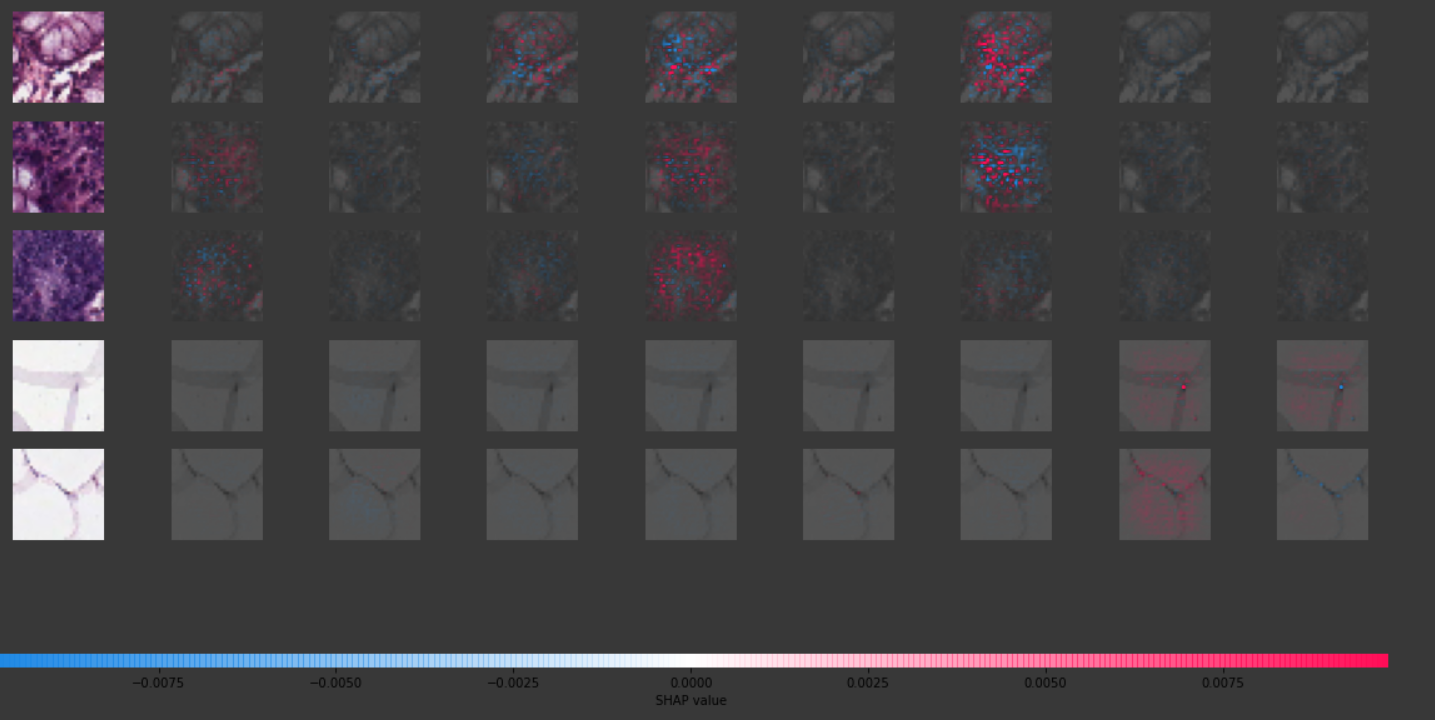
<https://github.com/slundberg/shap>

De esta manera, nosotros hemos seguido el ejemplo que hace uso de la función DeepExplainer.

Definiendo nuevamente una función que usaremos fuera pasándole el modelo deseado.



De esta manera, cuando la ejecutamos, nos encontramos con resultados como los siguientes.



Aquí, lo que se nos está mostrando, son 4 imágenes que se han seleccionado del conjunto de datos del que partimos, y de esas, lo que se hace, es ver para cada uno de los tipos que nos encontramos, la **sección de puntos rojos y azules**, que hacen que se seleccione como un tipo u como otro.

Por esto mismo, cuando vemos una selección de puntos azules, podríamos decir que son los **puntos en los que se fija para decir que no pertenece al tipo**, mientras que cuando hablamos del conjunto de los puntos rojos, hablamos de **los puntos concretos de la imagen en los que se fija para decir que sí que pertenece al conjunto**.

En nuestro caso, vemos como en las dos imágenes intermedias que aparecen en el ejemplo anterior, el modelo se centra en casi toda la imagen, esto, realmente tendría sentido, dado que como vemos, las imágenes de las que parte, a simple vista, no sucede como en otras en las que sí que hay una parte de la imagen que hace que se decante por esa misma, sino que son más un popurrí de colores.

Así mismo, podemos observar como el hecho de que haya aprendido que para las imágenes del cuarto tipo, se tiene que centrar en la parte central y en los alrededores de la imagen, hace que veamos como el modelo se fija en el centro de la imagen, hecho que podemos apreciar en los siguientes ejemplos.

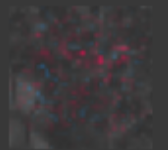
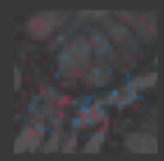
Lo cual, entiendo que el modelo también habrá aprendido que para que sea de ese tipo, tiene que mantener los colores de la imagen con la que ha aprendido.

Y como vemos, en las que sigue mirando por ese tipo, es imposible que diga que son de ese tipo, porque no encuentra los patrones de los colores en los píxeles de la imagen que está mirando, y por esto mismo, son descartadas.

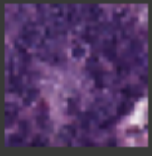
De igual modo, si nos fijamos en el resto de los tipos, nos encontraremos con algo similar, donde el modelo, para cada uno de los tipos, ha aprendido a realizar una cosa u otra.

Es decir, si nos vamos por ejemplo al tipo 1, veremos como el modelo, sí que se centra nuevamente en la parte central de la imagen. Pero, lo hace de una forma mucho más comedida, mientras que en el tipo 4 que hemos mostrado anteriormente, lo haría sobre casi toda la imagen. Pero aquí, no es el caso, y lo hace en una parte mucho más central,

De igual modo, el modelo, para clasificar a ese tipo, lo hace de una forma similar a la del tipo 4, dado que se ve claramente que tenemos muchos puntos rojos, aunque también alguno azul, por lo que parece, que el modelo para ese tipo ha aprendido con imágenes donde también hay un popurrí de colores, por lo que en las que nos encontramos con este popurrí, es fácil darse cuenta como algunos de los puntos de la imagen coinciden. Dando así, lo que podría ser una mala clasificación.

Una aclaración que quiero puntualizar es que cuando hablo de un popurrí en la imagen, me refiero a que son imágenes en las que es mucho más difícil distinguir lo que está pasando que en otras donde los colores son mucho más estáticos.

De igual modo, podríamos estudiar el resto de los tipos.

Si nos fuéramos al tipo 2 por ejemplo, veríamos como parece por las imágenes que tenemos, que clasifica como tipo 2, cuando tiene imágenes con una línea central, similar a lo que podemos ver en las dos últimas imágenes.

Aunque, también parece, que el modelo ha aprendido a fijarse solamente en el punto que acabamos de comentar, y de manera muy muy específica, esto, lo decimos, por el hecho, de que parece que, en la penúltima imagen, no llegan a haber tantos puntos rojos como en la última, y parece ser, que podría ser por el hecho de que la franja en este tipo de imágenes, se encuentra siempre bajo los mismos puntos, y no llega a coincidir del todo.

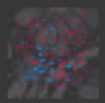
No obstante, cuando encuentra un patrón en el que hay como esa especia de franja en medio de la imagen, y tiene una tonalidad de color similar a las de las dos últimas imágenes, el modelo sí que hace clasificaciones sobre esa imagen, por eso, nos podemos encontrar con puntos rojos, como sucede en las siguientes capturas.

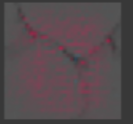
Del mismo modo, para el tipo 5, parece que el modelo también ha aprendido a clasificar las imágenes de este tipo de manera similar a las del tipo que acabamos de ver.

Por último, tenemos otros tipos que son curiosos de analizar, como pasaría en el tipo 3, donde a pesar de tener las tres primeras imágenes con la misma gama de colores, el modelo dice que coincide más lo que ha aprendido para este tipo, con el primero de las tres imágenes que nos encontramos.

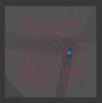
Seguramente, esto sea, porque el primero mantiene colores que tienen un tono más claro, y porque también sigue como pasaba en tipos anteriores, ciertos puntos de la imagen donde el modelo es más oscuro, y que serán los mismos puntos en los que el modelo dice que las otras imágenes de las que hemos hablado podrían ser clasificadas como del tipo 3.

Hay que recalcar, que también tenemos tipos, donde el modelo ha aprendido a clasificar que son del modelo por ser completamente de un color, o al menos, esa es la sensación que transmite. Como es en el caso del tipo 7, donde el modelo, ha aprendido que cuando mantiene colores blancos de fondo, y una especia de ‘y’ en la parte central, es de ese tipo. Y esto, lo podemos apreciar en las imágenes siguientes.

Por otro lado, si nos fuéramos al tipo 8, veríamos que sucede algo similar, solo que ahora, el modelo parece que ha aprendido a restringir más el dónde se encuentra la franja central del tipo.

Evaluación para la obtención de mejores métricas

Visto todo lo que hemos incluido en apartados anteriores, ahora nos quedaría mirar formas de sacar mejores resultados rebajando el número de épocas o preprocesando las imágenes, o cambiando las funciones de activación del modelo y demás.

Como vemos, las posibilidades de realizar cambios sobre el modelo son finitas, pero muy grandes.

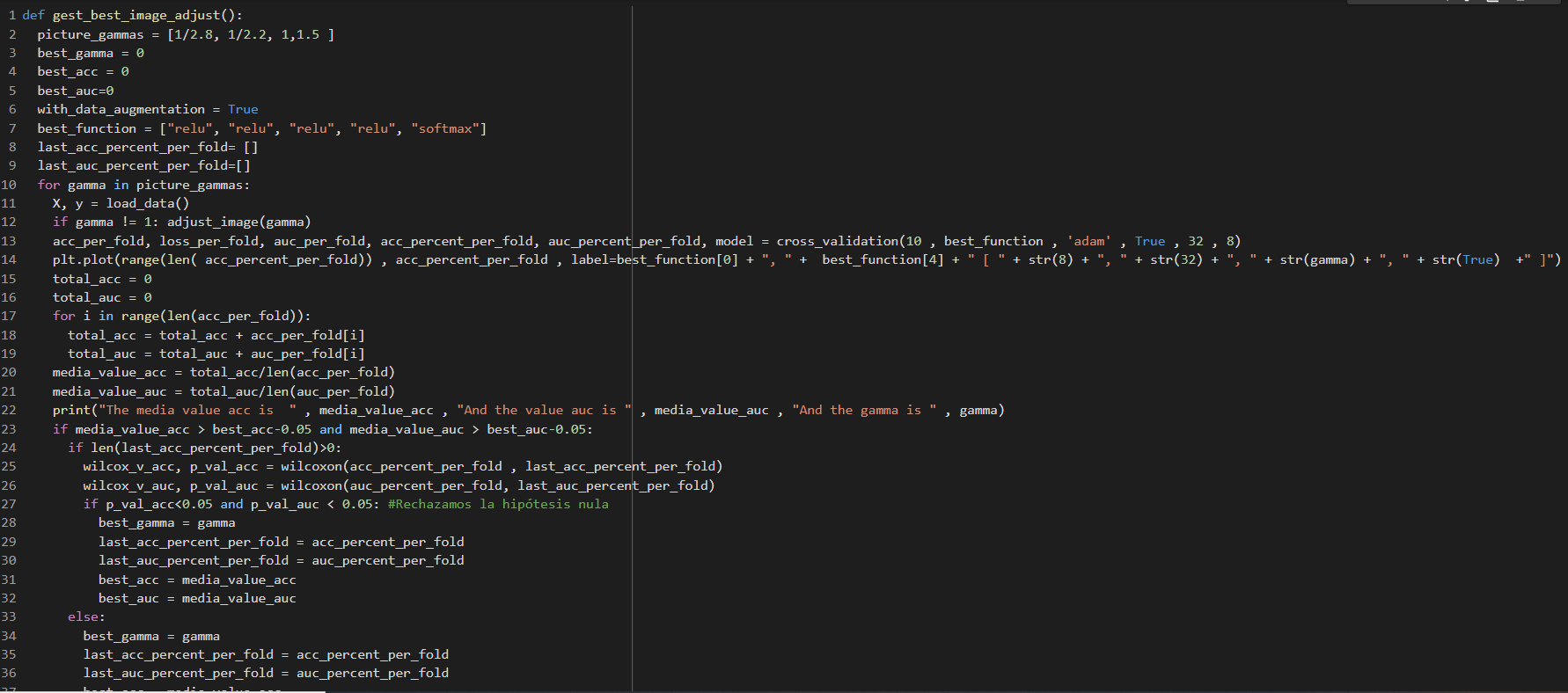
Por ello, personalmente, lo que he hecho para evaluar los diferentes avances que iba haciendo y sacar conjeturas, es hacerme una serie de funciones, para cambiar ciertos parámetros de manera automática, y comparar modelos para ver cuál es mejor.

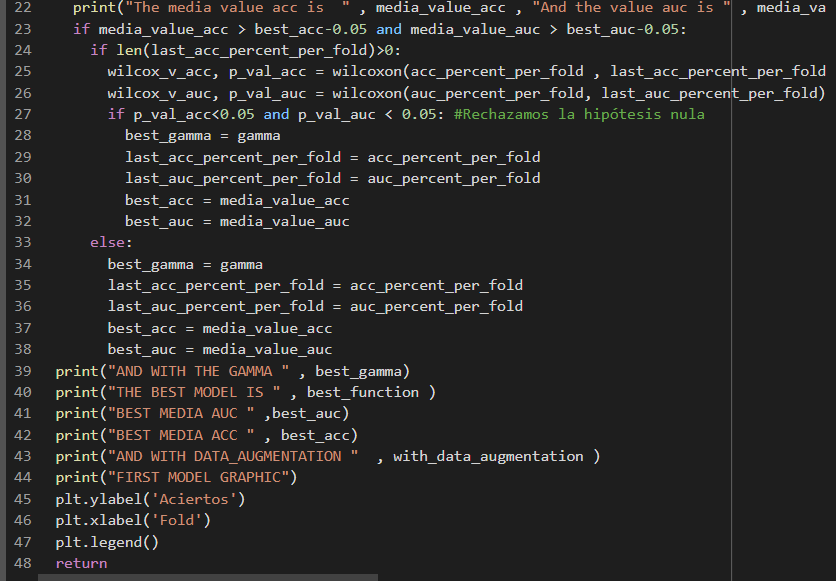
De esta manera, si vemos el funcionamiento de una de estas funciones por dentro, veremos que todas las que he hecho, mantienen un funcionamiento similar.

Y que es aquí dentro, donde he hecho uso de la función que nos habíamos definido de Wilcoxon, para saber si realmente un modelo que acabamos de conseguir mediante el cross\_validations, es mejor que otro o no.

Por esto mismo, tenemos funciones, por ejemplo, para ver cuál es el mejor optimizador de los tres que podemos usar, o funciones, para ver cuál es la mejor función de activación, o funciones, para ver con cuántas épocas podemos dar un accuracy mediantemente aceptable. Así mismo, también tenemos funciones para evaluar el escalado de la imagen, y funciones para evaluar el gamma que le aplicamos a la imagen.

De esta manera, las funciones que hemos definido mantienen un esquema similar al siguiente.





Aquí, recorremos el array de datos de entrada que hemos preparado, que dependerá de la función que estemos probando, y con eso, una vez dentro del bucle, iremos llamando a la función cross\_validation, para obtener los datos del modelo, y de esta manera, poder sacar gráficas y sacar también cuál es el que mejores resultados nos ha devuelto.

De igual modo, hay que destacar, que cada vez que miramos si el modelo es mejor que uno anterior, comparamos tanto su AUC media, como su acc medial. Qué quiere decir esto, que cogemos el array que nos devuelve la función cross\_validation, y sacamos la media de los resultados obtenidos, y si es mayor que la media que ya teníamos, pues entonces entramos.

Cuando entramos, también hay que tener en cuenta, que se hace uso de la función que nos hemos definido de Wilcoxon, para comprobar que realmente es mejor que el que ya teníamos, y por ello, al resultado que nos devuelve Wilcoxon, lo comparamos con 0.05, teniendo que ser menor el p value que 0.05, para rechazar la hipótesis nula, y considerarlo mejor que el que teníamos.

Una vez fuera del bucle, devolveríamos los resultados obtenidos, así como la gráfica de lo que hemos obtenido.

Y como hemos dicho, el resto de funciones que hemos incluido, funcionan de la misma manera, pero cambiando pequeñas cosas que las hacen adecuadas para su funcionamiento.

Por ello, vamos a ver lo que hemos obtenido en cada uno de sus apartados a continuación.