Práctica 2: Redes Neuronales Convolucionales

Alejandro Alonso Membrilla

Contents

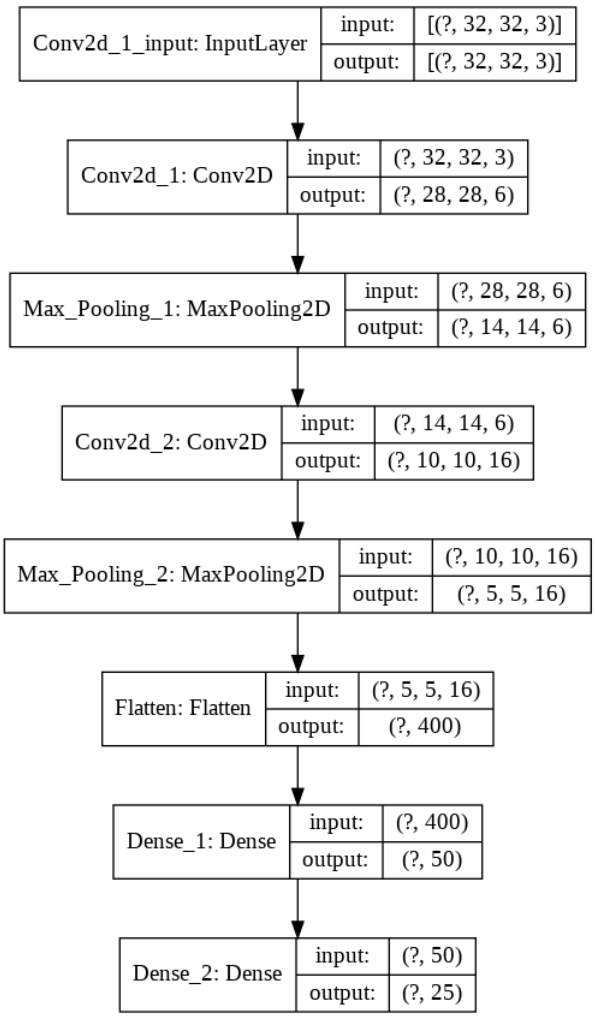
Introducción

Esta práctica consistirá en una serie de ejercicios relacionados con las redes neuronales convolucionales y su aplicación al problema de clasificación de imágenes. Trabajaremos con una parte del dataset CIFAR100 y con la colección de imágenes de pájaros de Caltech CUB-200. Haremos uso de la API de Keras para implementar una serie de redes convolucionales sencillas y para importar la arquitectura preentrenada ResNet50 y adaptarla a nuestros conjunto de datos.

El código implementado para la práctica ha sido probado mediante el servicio de Google Colab, debido a los largos tiempos de entrenamiento de las redes y a la ganancia en velocidad que aporta el uso de GPU proporcionado por este servicio en la nube. Dicho código se comparte junto a la presente memoria en formato .py como script de Python.

Apartado 1: BaseNet

En este apartado implementaremos una red sencilla predefinida por los profesores de prácticas, de nombre BaseNet. Su diseño es el siguiente:



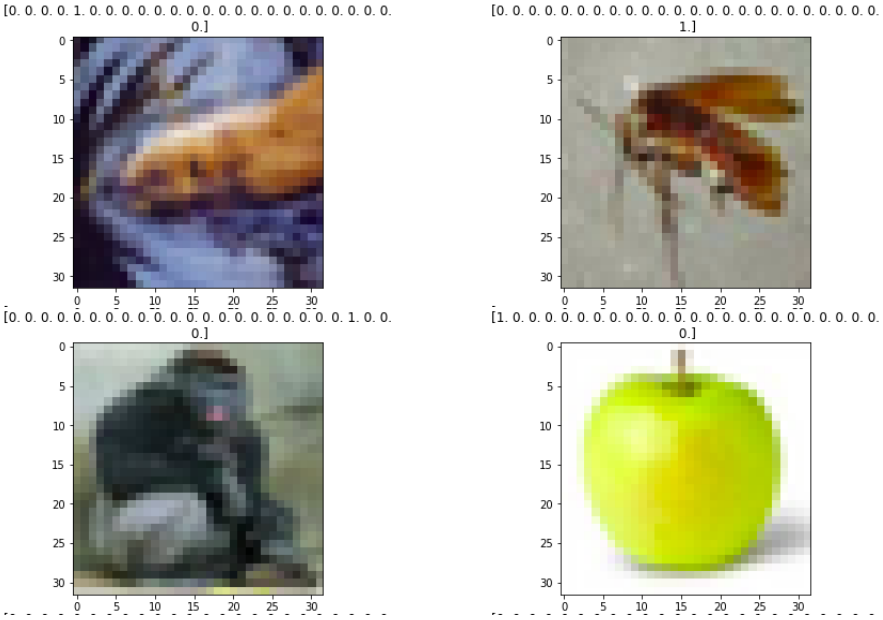
Donde los signos de interrogación indican que dicha capa admite un número arbitrario de tensores de las dimensiones que los acompañan. Los pesos iniciales de las capas convolucionales se han tomado con una distribución Glorot Uniforme debido a que este es un método de inicialización muy utilizado en

experimentos de Deep Learning y, siguiendo las conclusiones del siguiente artículo publicado recientemente por la Delft University of Technology la inicialización de los pesos no juega un papel importante en datasets sencillos (como debería ser nuestro conjunto tomado de CIFAR100), e incluso en conjuntos más complejos la distribución inicial más adecuada depende altamente de los datos. Ante la falta de las herramientas y los recursos necesarios para un estudio más exhaustivo, usaremos la Glorot Uniforme.

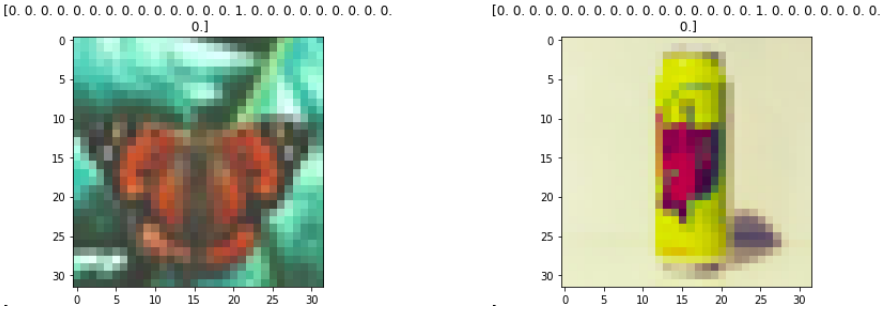
Carga y visualización del conjunto CIFAR100

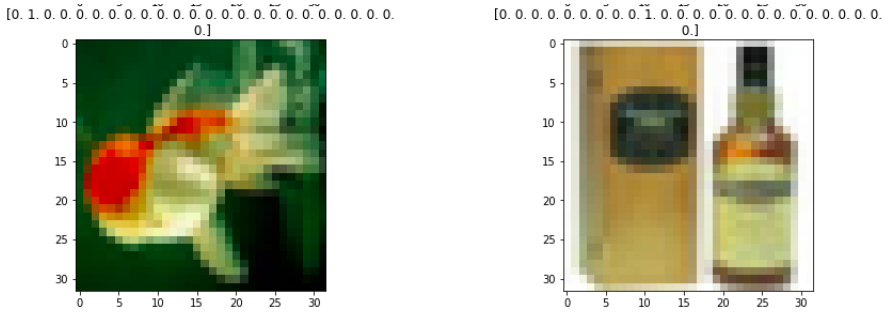
Keras facilita la descarga directa de algunos conjuntos de datos de uso común, entre los que se encuentra CIFAR100. El alumno ha usado las funciones dadas para la práctica que descargan, reducen y dan un formato adecuado al dataset para el desarrollo de la práctica. Posteriormente se han permutado los elementos del conjunto de datos para asegurar que sus clases se encuentren repartidas uniformemente y se han visualizado algunos de sus elementos junto con su respectiva etiqueta. A continuación se muestran algunos ejemplos del conjunto de entrenamiento y del de prueba, respectivamente:

Algunas imágenes del conjunto de entrenamiento



Algunas imágenes del conjunto de prueba





Comparación de distintos optimizadores mediante validación cruzada

Para garantizar que los resultados obtenidos en este apartado y en el siguiente sean lo mejor posibles, vamos a comparar una serie de criterios para optimización por descenso de gradiente estocástico (aquellos preimplementados por Keras y de uso común en Deep Learning) mediante validación cruzada. Esto consistirá en dividir, para cada optimizador, el conjunto de entrenamiento en 5 subconjuntos (*folds*). Iterativamente, se elegirá uno de esos subconjuntos para validación y se entrenará el modelo BaseNet con el resto de los datos de entrenamiento aplicando el criterio de optimización elegido. De esta forma obtenemos una serie de "tasas de precisión" independientes entre sí (puesto que se han tomado con distintos conjuntos de validación) que pueden promediarse para evaluar cada optimizador sobre el modelo y conjunto de entrenamiento utilizados.

Para evitar tiempos de ejecución demasiado largos, se ha utilizado el aumento de la función de pérdida en el conjunto de validación como criterio de parada, con una paciencia de 4.

A continuación mostramos la precisión obtenida por cada optimizador en cada iteración y en promedio:

	RMSprop	Adam	Adadelta	Adagrad	Adamax	Nadam
I1	0.2968	0.1676	0.0552	0.1336	0.2340	0.2896
I2	0.2708	0.1876	0.0508	0.0688	0.2428	0.2796
I3	0.2508	0.2676	0.0500	0.0628	0.1864	0.2272
I4	0.2496	0.2248	0.0560	0.0708	0.2308	0.1796
I5	0.2672	0.2640	0.0488	0.0620	0.1872	0.2428
Media	0.2670	0.2223	0.0522	0.0796	0.2162	0.2438

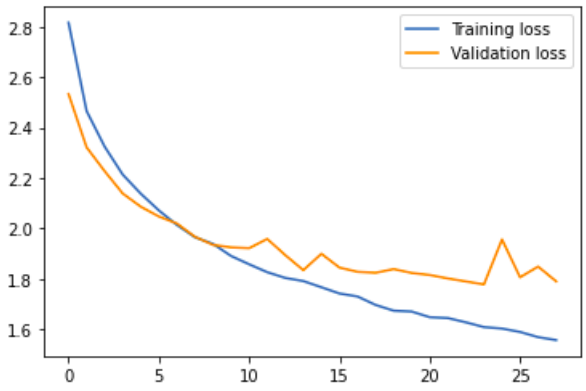
Vemos que los resultados de RMSprop superan a los del resto de optimizadores, lo que legitima su uso para entrenar BaseNet y sus futuras modificaciones (apartado 2) con el conjunto de entrenamiento tomado de CIFAR100.

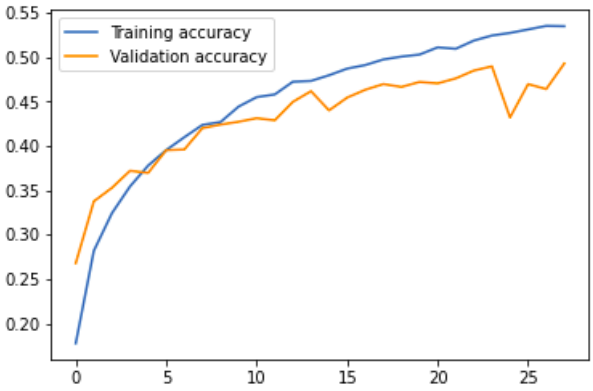
Entrenamiento y prueba de BaseNet

Ya en la sección anterior comprobamos que los resultados obtenidos por BaseNet, incluso con el optimizador que ha conseguido las precisiones más altas, no son buenos. Si bien hay que tener en cuenta que el error podría reducirse al entrenar la red con el conjunto de entrenamiento al completo, no podemos esperar una mejora muy grande. Por tanto, probaremos directamente a normalizar y aumentar los datos mediante zoom y reflexión horizontal de las imágenes, y a reentrenar BaseNet. Tal como se indicaba, se ha utilizado un conjunto de validación del 10% (usando el método `train_test_split` de Scikit Learn). Cabe destacar que la implementación de esta normalización y aumento se ha realizado usando la clase `ImageDataGenerator` de Keras, que **por defecto aplica el mismo aumento de datos a entrenamiento y validación**. Para esta práctica no se ha deseado aumentar los datos

de validación, por lo que ha sido necesario usar en este subconjunto otro `ImageDataGenerator` que solamente lo normalice (a partir, claro está, de la media y desviación estándar del subconjunto de entrenamiento). Por el mismo motivo, para normalizar los datos de test en la fase de prueba se ha usando un `ImageDataGenerator` semejante.

La evolución de la precisión y función de pérdida en entrenamiento y validación de BaseNet y su precisión final en el test para un tamaño de batch de 64, 30 épocas y usando datos aumentados se muestran a continuación:





Test accuracy: 0.4948

En general observamos una tasa de error muy elevada y una precisión de validación que crece a distinto ritmo que la de entrenamiento, lo que puede indicar un ligero sobreajuste que se intentará paliar en el siguiente apartado.

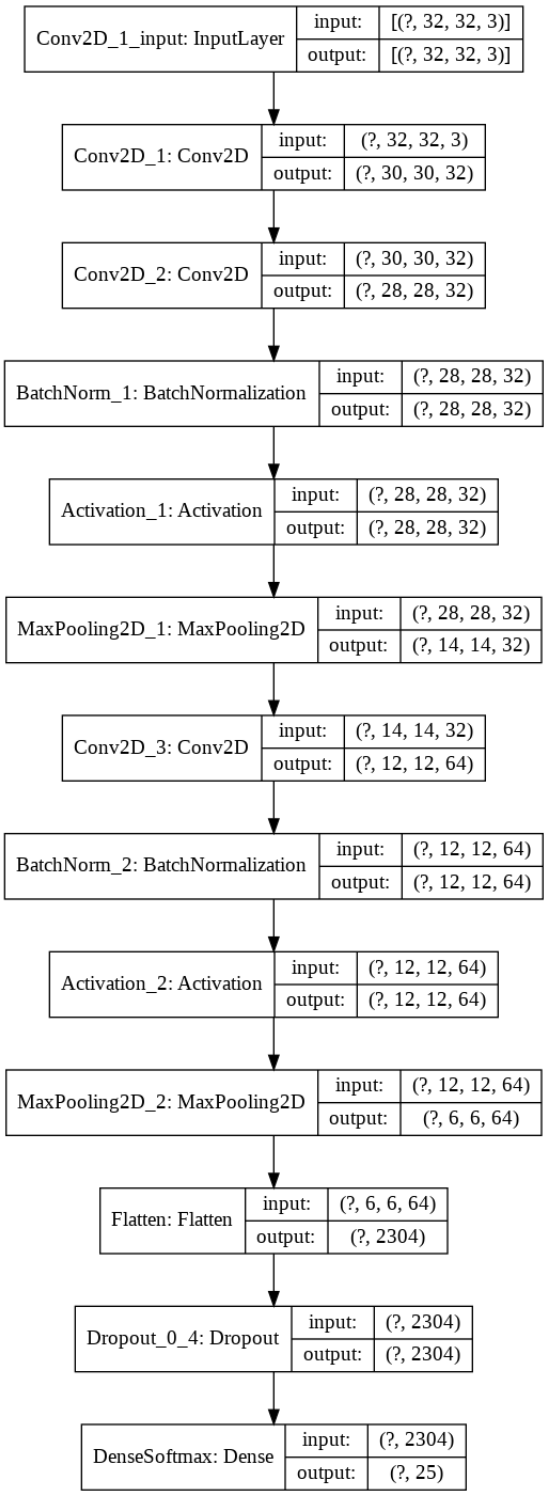
Apartado 2: Mejorando BaseNet

Viendo el diseño original de BaseNet en el apartado anterior, podría sospecharse que tanto el número de filtros como la profundidad de la red no son suficientes. Hemos hablado también de un problema de sobreajuste, aunque no muy severo, pero que permanece tras aumentar los datos. En esta sección presentaremos una serie de modelos, en orden creciente de complejidad, creados a partir del diseño de BaseNet y que buscan mejorar los resultados obtenidos. Cada uno de ellos se ha entrenado a partir del mismo conjunto de entrenamiento que el modelo inicial y para los mismos tamaño de batch y número de épocas.

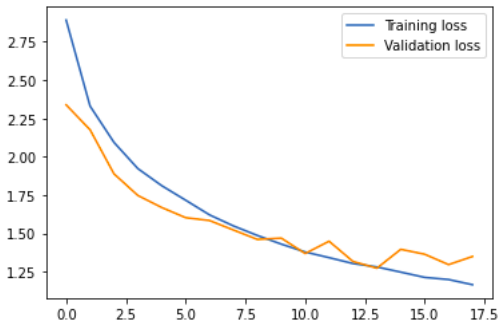
BaseNet2.1

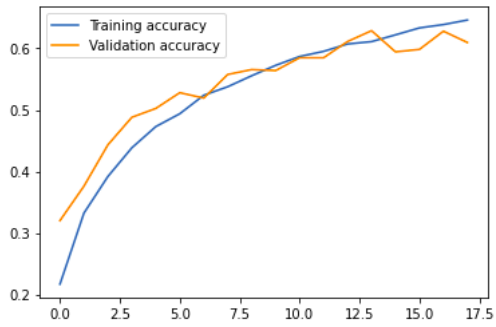
Nuestra primera propuesta de mejora de BaseNet aumenta el número de filtros e introduce Batch Normalization y una capa de Dropout (de ratio 0.4) como medidas de regularización. En el siguiente artículo sobre Batch Normalization se justifica que este método elimina el cambio covariable (*covariate shift*) lo que puede acelerar notablemente el entrenamiento y reducir el sobreajuste. En este otro estudio de la Universidad de Toronto (del año 2014) se presenta Dropout como un método de regularización que, en particular, "logra resultados estado-del-arte en SVHN, ImageNet, CIFAR-100 y MNIST", lo que justifica su uso en nuestro caso. También, para facilitar el aprendizaje de la red, hemos eliminado la penúltima capa FC, puesto que las capas de los perceptrones multicapa incorporan gran parte de la complejidad y número de parámetros al modelo. Por último se han sustituido las capas convolucionales de tamaño 5×5 por dos sucesivas de 3×3 que, tal como se ha visto en teoría, permiten un cálculo más eficiente.

El diseño de la nueva arquitectura es el siguiente:



Para evitar el sobreentrenamiento, incorporamos un criterio de parada tal como hicimos en el apartado 1 para la validación cruzada. La evolución del entrenamiento y la validación utilizando el mismo aumento de datos que con BaseNet, junto con la precisión alcanzada en la fase de testeo se muestran a continuación:





Test accuracy: 0.6252

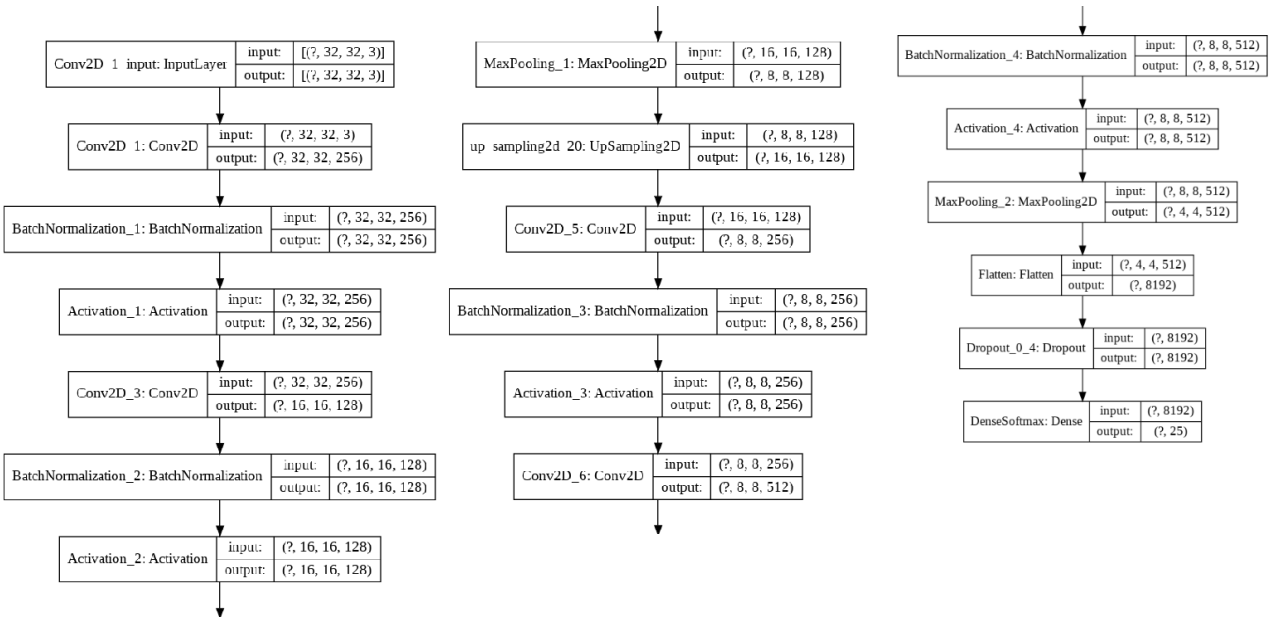
Observamos una precisión más elevada (en este caso, en torno a un 13% por encima de BaseNet) y unas curvas de evolución de pérdida y de precisión en validación mucho más cercanas a las de entrenamiento.

BaseNet2.2

En el modelo anterior no ha llegado a implementarse un aumento de la profundidad. En BaseNet2.2 aumentamos la profundidad con una nueva capa convolucional, seguida de su respectiva capa de normalización y activación. También aprovechamos para probar otro aumento del número de filtros, que en nuestro nuevo modelo oscilan entre los 128 y 512 canales por capa de convolución.

Con el objetivo de aumentar la profundidad ha sido necesario añadir una capa de escalado (UpSampling) seguida de un *stride* 2×2 en la siguiente convolución 2D (el stride permite ignorar aquella información insertada artificialmente en el escalado), además de añadir *padding* en cada convolución para conservar un tamaño de tensor adecuado al llegar a la última capa.

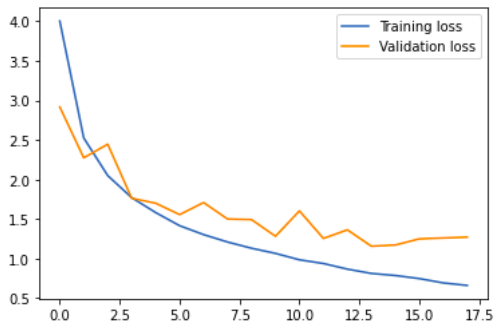
El diseño de esta arquitectura se muestra a continuación:

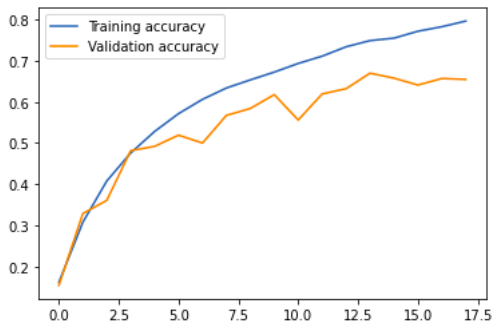


El criterio de parada y el aumento de datos establecidos para el entrenamiento de BaseNet2.2 es el mismo que para BaseNet2.1. El incremento en el número de parámetros nos hace pensar que pueda

ser necesario reducir el tamaño de batch para regularizar la convergencia del entrenamiento.

Los resultados son los siguientes:





Test accuracy: 0.6908

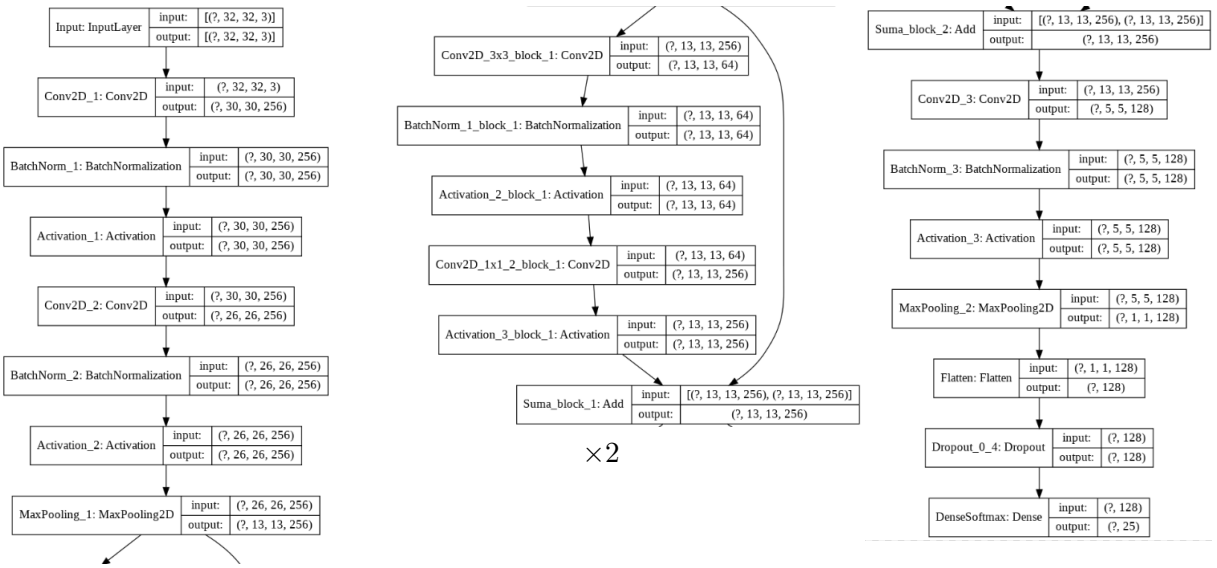
Si observamos las diferencias de precisión y función de pérdida entre entrenamiento y validación, vemos que vuelve a haber un sobreajuste considerable. Esto es, probablemente, debido al gran aumento de parámetros en el modelo con respecto a BaseNet2.1. También observamos un nuevo aumento de la precisión del modelo con respecto a los anteriores.

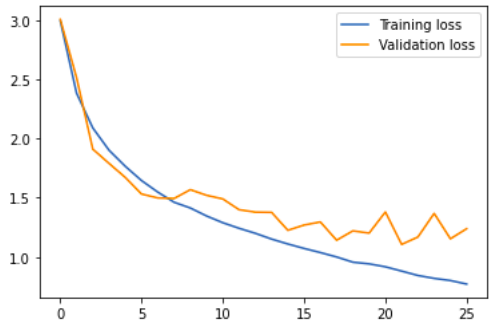
BaseNet2.3

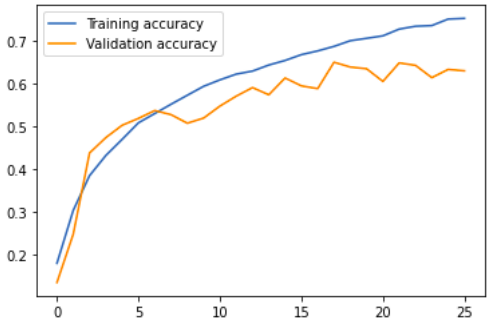
Finalmente, se ha probado la implementación de una tercera mejora de BaseNet inspirada en ResNet. Su diseño consiste en añadir una serie de capas de cálculo de residuos basadas en convoluciones 1×1 y 3×3 a la arquitectura de BaseNet2.2. El motivo se debe a que el cálculo de residuos se ha convertido en una pieza fundamental en cualquier CNN posterior a ResNet, lo cual demuestra el potencial de esta metodología. La influencia de este modelo en la literatura sobre el Deep Learning es justificación suficiente para intentar adoptar algunos de sus conceptos.

También hemos sustituido el MaxPooling 2×2 que seguía a la última capa convolucional por uno 4×4, que reduce la dimensión de cada canal a 1 único nodo que contiene la información más representativa de dicho canal.

El diseño final es el siguiente:







Test accuracy: 0.6784

Los resultados siguen mejorando los de BaseNet2.1, pero quedan por debajo de los de BaseNet2.2 que ha servido como base. Para mejorar esta arquitectura podría experimentarse con el número de canales por convolución. También podría probarse a eliminar las capas de convolución directa (Conv2D 1, 2 y 3) y sustituirlas por un cálculo de residuos, pero dado que los modelos propuestos para esta apartado deben tomar como base el diseño de BaseNet este experimento queda fuera de los objetivos del ejercicio.

Apartado 3: Redes Preentrenadas

En este apartado estudiaremos como importar una red preentrenada de las facilitadas por la API de Keras, en particular el modelo ResNet50, y adaptarlo de diversas formas para resolver un problema de clasificación de 200 tipos distintos de pájaros a partir de la base de datos CUB-200, de Caltech.

Carga y visualización del conjunto CUB-200

En primer lugar hubo que descargar el conjunto de datos de la web de Caltech. Las imágenes se encuentran almacenadas en formato jpg y fueron leídas la primera vez empleando las funciones aportadas por los profesores de prácticas.

La primera complicación fueron los tiempos de carga de las imágenes. Dado que, como se indica en la Introducción, los experimentos para esta práctica se han realizado en Google Colab, cargar las imágenes cada vez que se desee reiniciar sesión y entrenar las redes diseñadas lleva un tiempo enorme (en torno a 45 minutos cada vez). Para paliar este problema, se ha hecho uso de los métodos `save` y `load` de `numpy`. Una vez leídas las imágenes por primera vez, fueron almacenadas en formato binario (.npy) para poder ser recargadas en cualquier momento. Los tiempos de carga y barajado de las imágenes desde binario ronda en torno a los 30 segundos. En el código que se presenta junto a esta memoria, la escritura y lectura desde binario se encuentran comentadas para no escribir nada en disco, siguiendo las indicaciones del guión de prácticas.

Para trabajar con un conjunto de imágenes es conveniente visualizar algunas de ellas primero. A continuación mostramos algunos ejemplos del conjunto de entrenamiento, con su respectiva etiqueta:

Extendiendo ResNet50

A continuación es necesario importar el modelo preentrenado que adaptaremos a nuestro dataset. Consideraremos varias formas de extender la red sin modificar las capas convolucionales de ResNet50, que ya cuenta con unos pesos obtenidos al entrenar este modelo con la base de datos de ImageNet.

Las dos primeras consisten en eliminar únicamente la última capa densa de 1000 nodos, conservando un GlobalAveragePooling que resume cada uno de los 2048 canales resultantes del proceso convolucional en un vector de características, tras la que se coloca a)una capa densa con 200 nodos (número de clases) que hace de salida o b)un perceptrón multicapa que termina en una capa densa análoga. El proceso de extracción de características puede implementarse de dos formas:

1. Como preprocesamiento de las imágenes, cuyo resultado se toma como entrada de un perceptrón multicapa o cualquier otro clasificador.
2. Como parte de un modelo más grande, que conecta la salida de ResNet50 con las capas densamente conectadas que se deseen. El modelo se entrena dejando las capas de ResNet50 inalterables, esto es, se modifican únicamente los pesos de las capas añadidas.

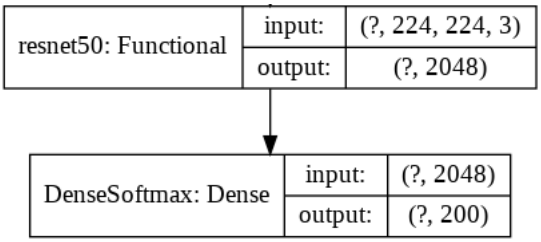
En este caso hemos optado por la segunda opción. El motivo es que posteriormente realizaremos un ajuste fino de toda la red que obligará a tener un modelo de la forma descrita en 2), por lo que en este apartado entrenaremos únicamente las últimas capas y observaremos como cambian los resultados al tomar esa misma red y aplicarle un ajuste a todas sus capas.

Como tercera forma de extensión de ResNet también eliminaremos el pooling que tiene tras su última convolución y añadiremos nuevas capas convolucionales.

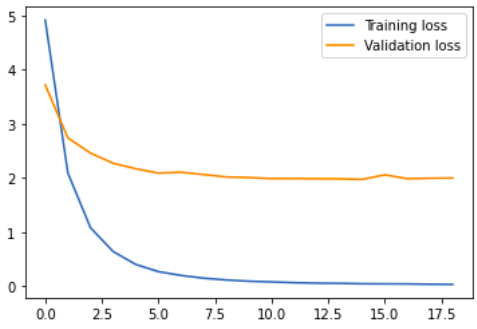
El criterio de optimización para las redes entrenadas en este apartado ha sido Adam, siguiendo las indicaciones de este curso de la Universidad de Stanford sobre CNN, en el que se recomienda el uso de Adam como optimizador por defecto en ausencia de un estudio específico del comportamiento de otros optimizadores para el problema particular.

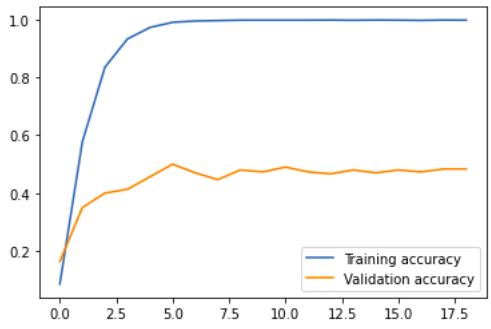
Entrenando solo la salida

Este es el primer modelo de los explicados anteriormente. Su diseño (simplificado) puede verse en el siguiente diagrama:



A la hora de entrenar el modelo, como se ha dicho anteriormente, dejamos fijas las capas convolucionales de ResNet. Para el entrenamiento ha sido necesario aplicar una función de preprocesado a los valores de entrada de ResNet, facilitada por Keras. La evolución de las funciones de pérdida y precisiones en entrenamiento y validación a lo largo del entrenamiento es la siguiente:





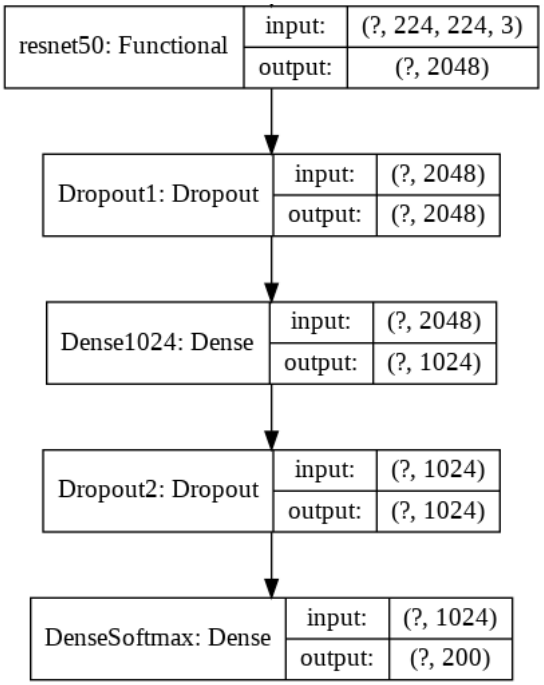
Test accuracy: 0.4421

Vemos unos resultados relativamente malos y un destacado sobreajuste. Lo primero puede deberse al bajo número de parámetros que hemos optimizado para clasificar nuestro dataset actual, y a que las características extraídas por ResNet50 no representen adecuadamente las nuevas imágenes. Lo segundo tal vez sea debido a una insuficiencia de datos.

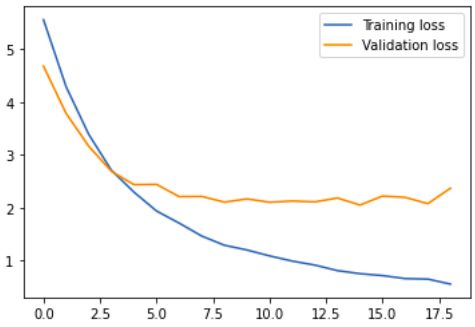
Añadiendo capas densamente conectadas

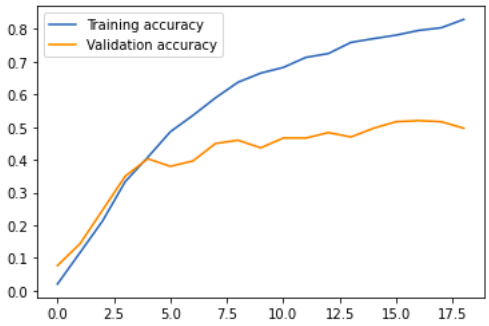
Vistos los resultados del modelo anterior, buscamos una extensión de ResNet50 que mejore la clasificación y, en el proceso, reduzca el sobreajuste. Hemos probado a añadir una capa FC con 1024 nodos (de tamaño similar al de la capa densamente conectada que hacía de salida en el ResNet50 original) seguida por otra de 200 activada con softmax. El objetivo ha sido replicar la estructura de salida

que tiene la ResNet original y comunicarla con una capa de salida adaptada a nuestro problema, pero añadiendo capas de Dropout para favorecer la regularización. Su diseño queda de la siguiente forma:



Las condiciones del entrenamiento son las mismas que las de la versión anterior. Sus resultados son los siguientes:





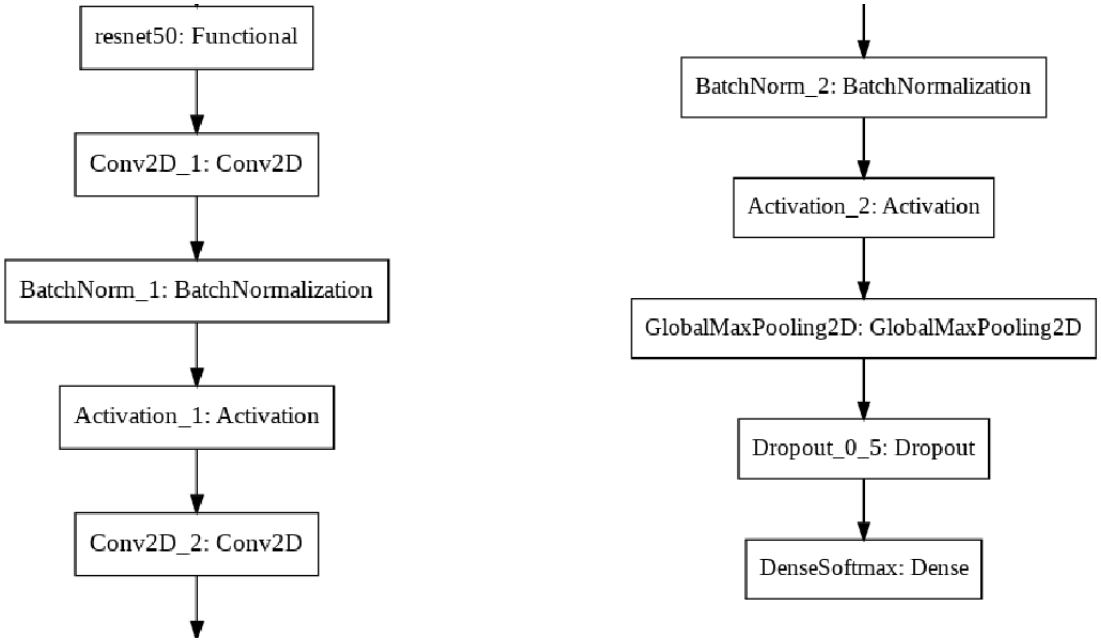
Test accuracy: 0.4309

A falta de un test estadístico más exhaustivo, la precisión de este modelo parece rondar en torno a la del anterior. Sin embargo, sí que observamos una palpable reducción del sobreajuste, que tal vez podría eliminarse aplicando un aumento suficientemente completo de los datos. Al no disponer de recursos de cómputo más elevados, dejaremos aquí nuestra experimentación con este modelo.

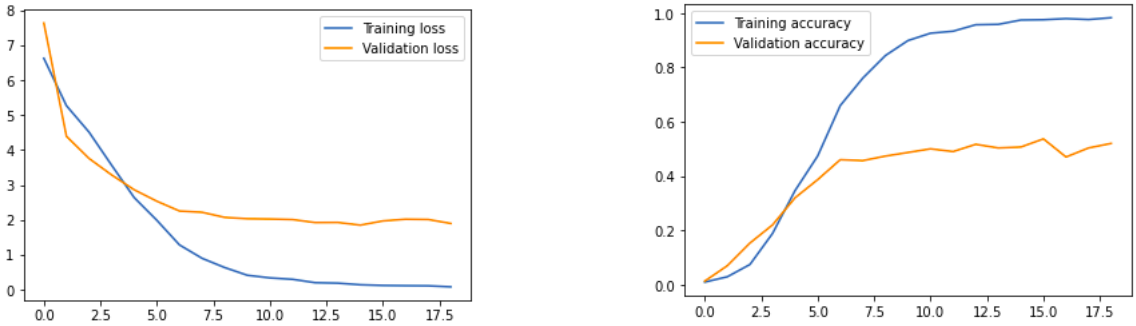
Añadiendo capas convolucionales

Añadiendo capas convolucionales no solamente aumentamos el número de parámetros que añadimos para clasificar nuestro conjunto de datos, sino que nos permite reducir el número de canales extraídos sin hacer uso de capas FC adicionales a la salida. Debido a los resultados obtenidos en el apartado 2, en el que eliminar la penúltima capa densamente conectada ayudó a obtener mejores resultados, creemos que aplicar el mismo procedimiento podría ayudar de la misma forma si extendemos ResNet50 con una

serie de capas en una estructura similar a la de nuestra mejora de BaseNet. El diseño final queda de la siguiente forma:



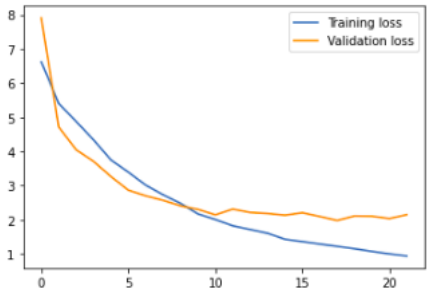
Los resultados del entrenamiento y prueba de este modelo se muestran a continuación:

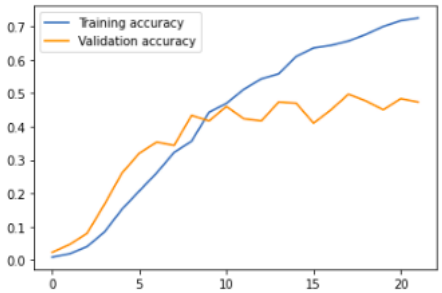


Test accuracy: 0.4467

Vemos una pequeña mejoría, aunque nada muy significativo, sobre la versión anterior. El sobreajuste, de nuevo, es menor que en el caso de solo modificar la salida, pero aún es muy marcado, lo que puede indicar dos cosas: o bien las imágenes de entrenamiento son insuficientes, o bien entrenar la salida no es adecuado para obtener buenos resultados en nuestro caso. Para tratar de solucionar este problema aplicaremos un aumento de datos y reentrenaremos el modelo. Las aves que aparecen en las imágenes de nuestro dataset, como hemos visto, se muestran en muchos tamaños y posiciones diferentes. Para mejorar el aprendizaje aplicamos transformaciones de zoom (en un rango de [0.7, 1.3]), una posible rotación entre -100 y $+100$ grados, y reflexiones horizontales. Incrementaremos ligeramente el número de épocas para poder aprender adecuadamente los datos aumentados (el sobreajuste no debería empeorar los resultados puesto que, como hemos visto, la precisión de validación tiende a estancarse pero no a decrecer).

Los resultados de este experimento se muestran a continuación:





Test accuracy: 0.4355

De nuevo, el sobreajuste baja pero la precisión obtenida en test por nuestro modelo entrenado con datos aumentados no mejora. Todo apunta a que entrenar solamente las últimas capas no es suficiente para obtener un modelo adecuado para nuestro dataset.

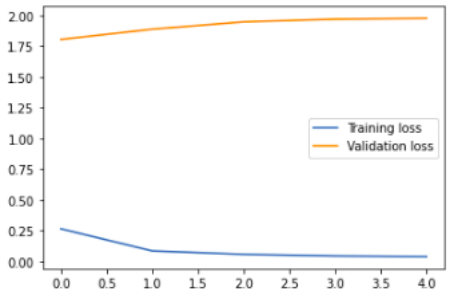
Ajuste fino de ResNet50

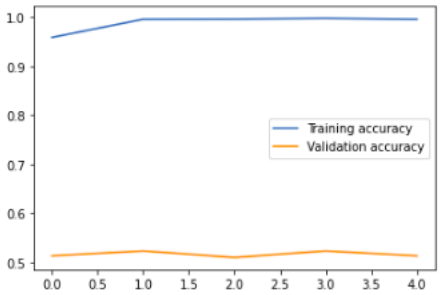
Los resultados obtenidos en el apartado anterior no son buenos. Además de las precisiones bajas obtenidas en test, el sobreajuste no ha logrado eliminarse ni siquiera aplicando aumento de datos. En estos casos puede ser adecuado realizar un ajuste fino. Para evitar un sobreajuste indeseado de la red es conveniente contar con una cantidad suficiente de datos, tal como se explica en el curso de la Universidad de Stanford ya citado anteriormente, para lo que repetiremos el aumento de datos del apartado anterior.

El primer paso será repetir el procedimiento seguido al importar la red ResNet preentrenada y modifcar, únicamente, su salida. Congelamos la red base y repetimos el entrenamiento de la última capa. Este paso es importante porque, tal como se indica en el capítulo sobre *Transfer Learning (5.3.2)* del libro Deep Learning with Python de François Chollet, no hacerlo provocaría que la señal de error propagada durante el entrenamiento creciera demasiado, destruyendo las representaciones previamente aprendidas por la red en su entrenamiento original.

Por otro lado, también será necesario reducir el ratio de aprendizaje de la red. Un aprendizaje demasiado acelerado modificaría los pesos iniciales de una forma muy brusca, echando a perder la ventaja de utilizar una red preentrenada. Para el ajuste fino siguiente hemos decidido aplicar un ratio de aprendizaje de 0.00001, siguiendo el siguiente ejemplo en la web de Keras. El número de épocas de aprendizaje también se ha limitado a 10 con el mismo propósito. El tamaño de mini-batch usado por los creadores de ResNet en su artículo original es de 256, pero en este otro estudio comparativo sobre la influencia del tamaño de batch, se concluye que tamaños pequeños, de 32 elementos o inferiores, son los que logran resultados mejores y más estables. Por tanto, para el ajuste fino hemos decidido dejar en 32 el tamaño de lote.

El primer paso para ajustar la red completa es descongelar el modelo base, tras lo que podemos entrenar la red, obteniendo los siguientes resultados:





Test accuracy: 0.4643

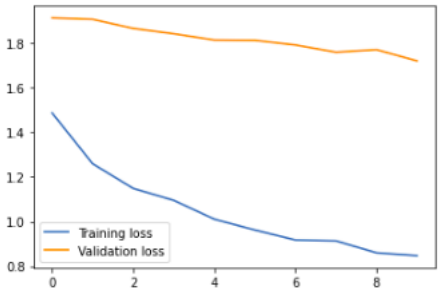
Debido a los largos tiempos de ejecución no ha sido posible repetir el experimento un número suficiente de veces como para afirmar que los resultados sean estadísticamente significativos. Sin embargo, vemos una mejora en torno al 2% de precisión en fase de test.

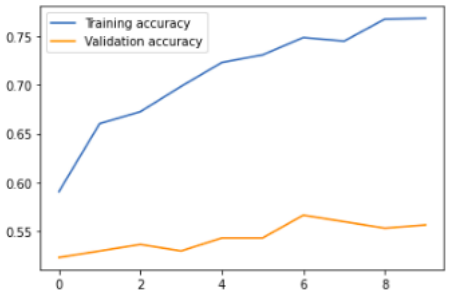
En general, aplicar un ajuste fino de la red preentrenada ResNet50 a nuestro dataset ha logrado mejorar sus resultados en la clasificación de imágenes del conjunto CUB-200. Sin embargo, a nivel experimental, se ha deseado comprobar si descongelando el modelo base usado en la última extensión de ResNet (aquella con capas convolucionales adicionales) puede mejorar sus resultados. El motivo es que, en esta, la evolución del entrenamiento muestra un aprendizaje más regular y menos sobreajustado, lo que parece conveniente a la hora de reajustarse conjuntamente con el resto de la red.

Aplicamos un procedimiento análogo al del experimento anterior:

1. Importamos ResNet, sin pooling ni última capa, y congelamos sus pesos.
2. Lo extendemos con las capas convolucionales y la salida correspondientes.
3. Entrenamos el modelo con los pesos de la red base congelados.
4. Descongelamos estos pesos y repetimos el entrenamiento, para un ratio de aprendizaje y número de épocas bajos.
5. Evaluamos el modelo con el conjunto de test.

Los resultados obtenidos en este caso son los siguientes:





Test accuracy: 0.4824

En este caso los resultados parecen haber mejorado más que añadiendo solo la salida, y la precisión obtenida en test supera a la simple extracción de características en torno a un 4%.

Bonus: MedMNIST

Para el bonus de esta práctica se ha propuesto entrenar una red neuronal que clasifique los elementos del dataset PathMNIST, un conjunto de imágenes de tejidos con 9 tipos diferentes de cáncer colorrectal. Este forma parte de una colección de datasets de imágenes médicas para educación en Visión por Computador llamada MedMNIST.

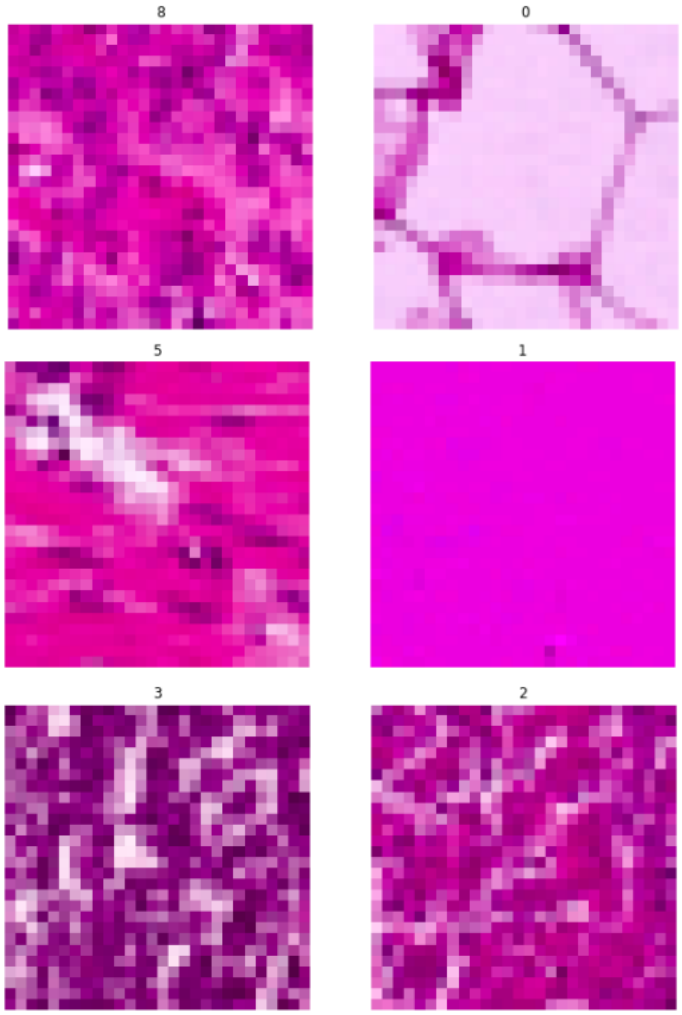
Nuestra propuesta consiste en el uso de la red preentrenada **NASNet Mobile**, facilitada por Keras con los pesos calculados a partir de ImageNet, como extractora de características de estas imágenes que

posteriormente se clasifican mediante regresión logística en una de las 9 clases que componen nuestro dataset. Esta red posee un número de parámetros relativamente bajo en comparación con otras redes estado-del-arte por lo que es rápida de entrenar, lo que la hace adecuada para esta práctica teniendo en cuenta los recursos utilizados.

Carga y visualización de PathMNIST

El conjunto de datos se encuentra dividido en 89.996 imágenes de entrenamiento, 10.004 para validación y 7.180 de prueba, almacenadas en formato binario `.npy`. Cada una tiene un tamaño de 28×28 píxeles y una etiqueta en forma de número entero entre el 0 y el 8. Las imágenes se leen desde el directorio *imagenes/pathmnist*.

Algunos ejemplos del conjunto de imágenes de entrenamiento de PathMNIST pueden verse a continuación:



Tras su carga, usando el método `load` de NumPy, en distintos arrays, cada conjunto es barajado y redimensionado a un tamaño de 224×224 para ser compatibles con la red NASNetMobile preentrenada. El conjunto completo de imágenes redimensionadas requiere de demasiada memoria RAM para ser empleada en el entrenamiento, por lo que hemos necesitado quedarnos con un subconjunto del 11% de las imágenes de entrenamiento (los conjuntos de validación y test se han usado en su totalidad).

Extracción de características con NASNet Mobile

El procedimiento a seguir ha sido importar la red preentrenada eliminando la capa de salida, pero conservando la capa de pooling mediante cálculo del promedio en cada canal para conservar la estructura original de la red. Al resultado, se le añade una capa densamente conectada con 9 nodos (uno por clase), activada por una capa *softmax*.

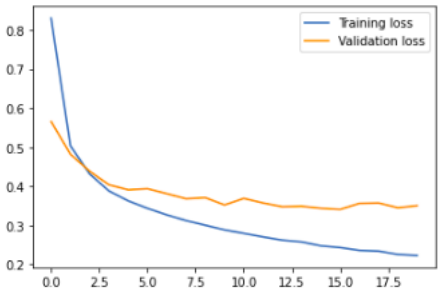
El entrenamiento se ha realizado aplicando la crosentropía categórica dispersa como función de pérdida, debido al tipo de problema a resolver (clasificación) y a la forma en la que vienen dadas las etiquetas (como número decimal, en vez de binario). El método de optimización utilizado ha sido Adam, por los motivos expuestos en el apartado 3.

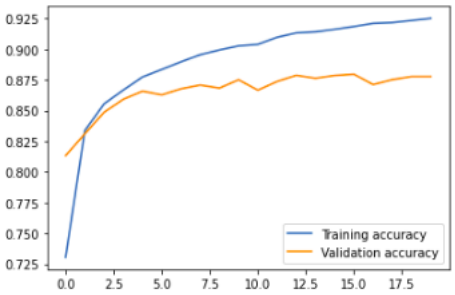
Tanto para el entrenamiento como para la fase de testeo ha sido necesario preprocesar las imágenes utilizadas siguiendo el procedimiento de NASNet Mobile. Al igual que en el caso de ResNet50, Keras facilita el uso de una función de preprocesado que puede aplicarse sobre los datos mediante un `ImageDataGenerator`.

Entrenando solo la salida

En este caso hemos optado por congelar todas las capas de NASNet, para que solo los pesos de la capa de salida se vean influidos por el entrenamiento. Se ha establecido un tamaño de lote de 32 por las mismas causas que en el ajuste fino del apartado 3 (conseguir una convergencia más estable de la red neuronal) y un número máximo de épocas de 100, aunque de nuevo, para evitar el sobreentrenamiento, se ha configurado que el algoritmo se detenga prematuramente si la pérdida de validación no disminuye durante 4 épocas consecutivas (de hecho, el entrenamiento para después de 19 épocas).

Los resultados obtenidos son los siguientes:





Test accuracy: 0.8649

Los resultados muestran unas precisiones de validación y test relativamente altas, pero también existe un sobreajuste palpable.

A continuación, reentrenaremos el mismo modelo aplicando una serie de medidas para mejorar la precisión todo lo posible y reducir el sobreajuste.

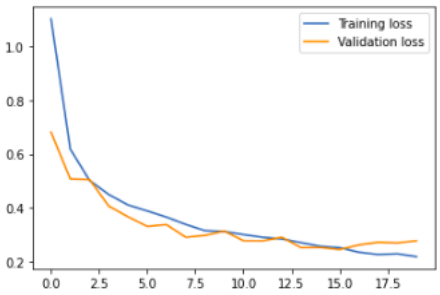
Entrenando las capas superiores de NASNet Mobile

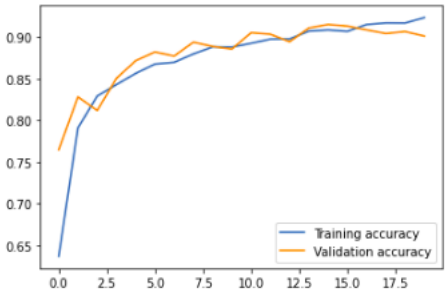
En este caso se ha optado por entrenar también las últimas 50 capas, aquellas más cercanas a la salida, para permitir que la red olvide los detalles más específicos aprendidos de ImageNet y aprenda aquellos particulares a nuestro conjunto de imágenes.

Dado que hemos necesitado reducir nuestro conjunto de entrenamiento, que queremos aumentar el número de parámetros a entrenar considerablemente y que nuestro modelo ya había mostrado sobreajuste en el entrenamiento anterior, también consideramos un aumento de datos para dar mayor variedad a nuestro conjunto de entrenamiento. Se han tenido en cuenta aumentos de tamaño (*zoom range* de 0.3), reflexiones horizontales y rotaciones aleatorias de hasta 50 grados. Además, se ha reducido el ratio de aprendizaje a 8×10^{-5}, para favorecer que las últimas capas de NASNet Mobile aprovechen parte del conocimiento aprendido en sus pesos preentrenados, pero sin llegar a los ratios tan bajos de 10^{-5}, establecidos para el *fine-tuning* del apartado 3, para darle la oportunidad a la red de ser influenciada en mayor medida por los detalles de nuestro dataset, y para entrenar apropiadamente la capa de salida.

Por otro lado, teniendo en cuenta el probable aumento del tiempo de entrenamiento de la red tras el descongelamiento, se ha reducido el número máximo de épocas a 20. De nuevo, monitorizamos la función de pérdida de validación y detenemos el entrenamiento cuando esta empiece a crecer, con una paciencia de 5 épocas.

Los resultados obtenidos son los siguientes:





Test accuracy: 0.8779

En primer lugar apreciamos una considerable reducción de sobreajuste, teniendo en cuenta que las curvas de pérdida y precisión de validación ahora fluctúan ligeramente en torno a las de entrenamiento. Si bien el error de test no ha decrecido mucho (en torno a un 1%), viendo la escala de estas gráficas nos damos cuenta de que el error de validación sí mejora sus resultados, rondando la precisión de validación el 90% en las últimas etapas, que en nuestro anterior experimento apenas llegaba al 87.5%.

En conclusión, hemos descongelado las últimas capas de una red preentrenada, las hemos entrenado en conjunto con la nueva capa de salida aplicando una serie de medidas de aumento de datos y ajuste de los parámetros del entrenamiento, y hemos observado una mejora significativa de los resultados.

Referencias

- A Comparison of Weight Initializers in Deep Learning-based Side-channel Analysis.
- Dropout: A Simple Way to Prevent Neural Networks from Overfitting.
- Curso de Stanford sobre redes neuronales convolucionales (sección sobre *optimización de NN*).
- Curso de Stanford sobre redes neuronales convolucionales (sección sobre *Transfer Learning*).
- Documentación de la API de Keras.
- Ejemplo de la web de Keras sobre transferencia de conocimiento.

- Deep Residual Learning for Image Recognition.
- Revisiting Small Batch Training for Deep Neural Networks.
- MedMNIST.