Práctica 4 – Reconocimiento de escenas con Redes Convolucionales Neuronales

Moreno Díez, Juan – Pascual Francés, Jaime

4 PREGUNTASTAREAS OPCIONALES

4.1 Estudie la variación en rendimiento para diferentes valores del *batch_size* = [8 ,16, 32, 64] y 50 épocas. Razone porque obtiene los resultados observados. (1 punto)

Para la realización de este apartado utilizamos el notebook **p4_pregunta_41.ipynb.** En este fichero realizamos todas las partes necesarias para poder posteriormente entrenar y clasificar el modelo. Lo primero de todo es preparar el entorno de trabajo instalando las librerías necesarias, después de esto crear el directorio donde vamos a almacenar el dataset y almacenarlo y seleccionar el hardware a utilizar inicializando la CPU y la GPU.

Ahora ya viene la parte de estudiar la variación para los diferentes *batch_sizes*, probando como dice el enunciado con valores igual a 8, 16 32 y 64. Para ello hemos inicializado una lista de *batch_sizes* con estos valores.

Para almacenar el rendimiento y las pérdidas para los diferentes *batch_size* hemos creado 4 listas, 2 de ellas para el rendimiento de los datos de entrenamiento y validación, y otros 2 para la pérdida de los datos de entrenamiento y validación. Para realizar el estudio con los diferentes valores hemos hecho un bucle que recorra cada uno de los valores de la lista de *batch_size*.

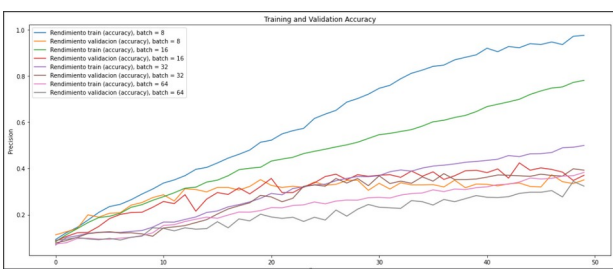

Figura 1. Rendimiento en validación y train para los diferentes *batch_size*

Ahora tenemos que mostrar los resultados, para ello usamos las gráficas de matlotlib, incluyendo en una misma gráfica el rendimiento, ya sea en validación o en entrenamiento. Como podemos ver en la figura 1. Varía mucho el rendimiento según el valor de *batch_size* que demos. Podemos observar que a mayor *bacth_size* encontramos un menor rendimiento en cuanto al rendimiento de los datos de entrenamiento, siendo el de validación bastante similares entre ellos.

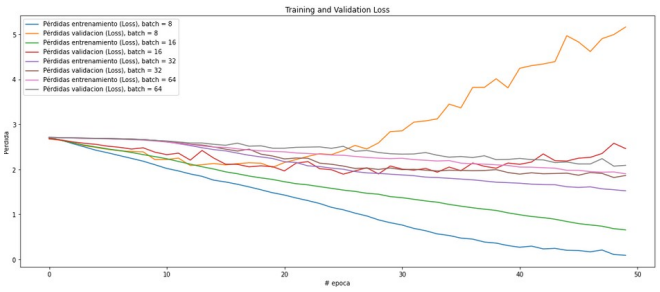

Figura 2. Perdidas en validación y train para los diferentes *batch_size.*

En cuanto a las pérdidas, podemos ver en la figura 2, que al igual que en el rendimiento, varía mucho según el *batch_size* que utilicemos. Podemos ver que las pérdidas de validación para el *batch_size* igual a 8 incrementa de forma bastante notoria, esto se debe a que el factor de aprendizaje es demasiado alto.

Finalmente, hemos decidido que el *batch_size* que da mejores resultados es 32. Nos hemos basado en que en el caso del rendimiento de los datos de entrenamiento, para un *batch_size* muy bajo, tiende a aparecer sobreajuste, mientras que para un *batch_size* alto, es muy bajo el rendimiento. Además el número de pérdidas en validación es muy alto para *batch_sizes* más pequeños.

4.2 Estudie la variación en rendimiento para diferentes tamaños de la imagen de entrada ([32x32, 64x64, 128x128, 224x224]) durante 50 épocas. Elija un valor de *batch_size* acorde a sus conclusiones en la pregunta anterior 4.1. Razone porque obtiene los resultados que observa. (1 punto)

Para la realización de este apartado utilizamos el notebook **p4_pregunta_42.ipynb.** En este fichero realizamos todas las partes necesarias para poder posteriormente entrenar y clasificar el modelo. Es decir, preparamos el entorno de trabajo, creamos y cargamos los datos necesarios y posteriormente empieza la lógica del ejercicio.

Para la realización, iteramos sobre la generación de datos, entrenamiento y clasificación para los distintos tamaños de imágenes propuestos. Mantenemos el batch_size a 32 ya que es el valor con el que mejores resultados se han obtenido en el apartado anterior. En esa iteración vamos metiendo en distintas listas las precisiones y las pérdidas de entrenamiento y validación. Por último se realiza una representación gráfica para los distintos tamaños de imagen y luego mostramos la que mejores resultados ha sacado.

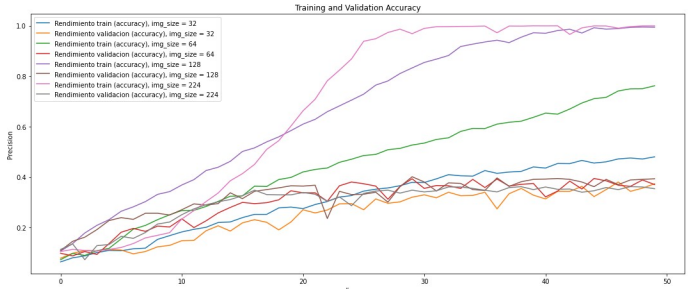

Figura 3. Precisión en validación y entrenamiento para los distintos tamaños de imagen.

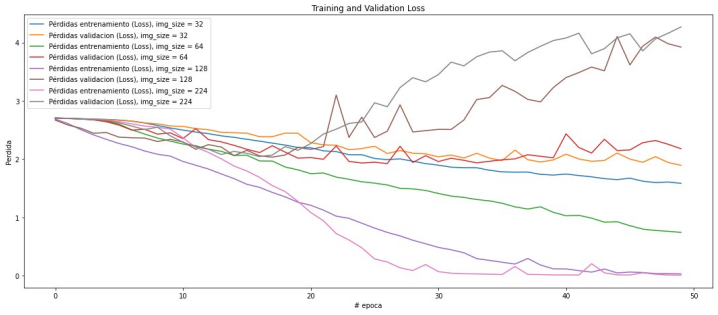

Figura 4 Pérdidas en validación y entrenamiento para los distintos tamaños de imagen.

Para realizar el análisis inicial nos fijaremos primero en la Figura 4, ya que algunas curvas de pérdidas las podemos descartar a simple vista. Lo que intentamos buscar es una curva en validación que vaya disminuyendo progresivamente. Las curvas con tamaño alto de imagen (224 y 128) que son la gris y la marrón respectivamente, las descartamos. Esto se debe a que dado un punto están subiendo y a nosotros nos interesa que las pérdidas vayan bajando a lo largo de la validación. Como podemos ver, la curva de pérdidas naranja (tamaño 32) es la que mejores resultados saca, es decir un menor valor de pérdidas en validación.

Al comparar las curvas de entrenamiento con las de validación en el caso de los tamaños altos de imagen, se puede ver que se ha producido un gran sobreajuste. Ya que entrenando se sacan muy buenos resultados, pero a la hora de validar el modelo no se ajusta adecuadamente a los datos. Sin embargo, para los tamaños 32 y 64, las curvas de entrenamiento y validación son mucho más parecidas y no se está produciendo casi sobreajuste. El modelo se ajusta a los datos y la validación lo comprueba de forma correcta en estos dos casos.

En la Figura 3 podemos realizar conclusiones muy parecidas a las de la Figura 4. Para tamaños de imagen altos (128 y 224) el entrenamiento saca precisiones muy buenas, pero a la hora de validar, las curvas están mucho más abajo. Se puede ver que se está produciendo un gran sobreajuste para esos tamaños de imagen, eso quiere decir que el modelo está entrenando, pero a la hora de validar, no se han ajustado los datos y el modelo correctamente. Con 32 y 64 ocurre igual que con las pérdidas, las curvas de entrenamiento y validación son muy parecidas. Los resultados obtenidos en precisión, son casi iguales para los tamaños 32 y 64, por lo tanto, ya que en pérdidas saca mejores resultados el modelo con tamaño 32, concluimos que el tamaño óptimo de imagen es el de **32x32**.

Al estar utilizando un tamaño más pequeño de imagen, generalizamos mejor la red y así hay menos datos que se pueden sobreajustar. De todas formas, podemos ver que tampoco se obtienen resultados muy precisos. Si tuviésemos un dataset más grande es muy probable que las precisiones aumenten y las pérdidas bajen de forma considerable.

4.3 Estudie la variación en rendimiento para diferentes funciones de activación[1] durante 25 épocas. Seleccione 3-4 opciones y, salvo la última capa con activación *softmax*, cambie TODAS las funciones de activación de la red en las distintas capas a las opciónes seleccionadas. Como inicialización de parámetros, aplique el valor por defecto en cada capa. Utilice un valor de *batch_size* e *img_size* acorde sus conclusiones en las preguntas 4.1 y 4.2. Razone porque obtiene los resultados que observa. (1 punto)

Para la realización de este apartado utilizamos el notebook **p4_pregunta_43.ipynb.** En este fichero realizamos todas las partes necesarias para poder posteriormente entrenar y clasificar el modelo, preparando el entorno de trabajo, creando y cargando los datos, es decir, el dataset, y seleccionando el hardware a utilizar.

En este apartado, nuestro objetivo es estudiar la variación del rendimiento según las distintas funciones de activación. Para realizar esto, al igual que en los anteriores apartados, hemos ido recorriendo diferentes funciones para comparar su rendimiento. Para ello hemos creado una lista llamada *list_activation* la cual contiene 4 funciones de activación que hemos considerado las más óptimas para estudiar. Tambié, hemos cambiado el número de épocas, siendo este igual a 25 para este ejercicio.

Estas 4 funciones las hemos escogido según el conocimiento que tenemos con respecto a ellas, ya que hemos cogido las que hemos estudiado en clase de teoría. La 4 funciones son linear, tanh, relu y sighmoid. Para los valores de *batch_size*, *img_height* y *img_width*, hemos escogido, tal y como dice el enunciado, los valores que en las pre-

1https://www.tensorflow.org/api_docs/python/tf/keras/activations

guntas 4.1 y 4.2 nos ha dado mejor resultado, siendo *batch_sice* igual a 32, *img_height* y *img_width* igual a 32x32.

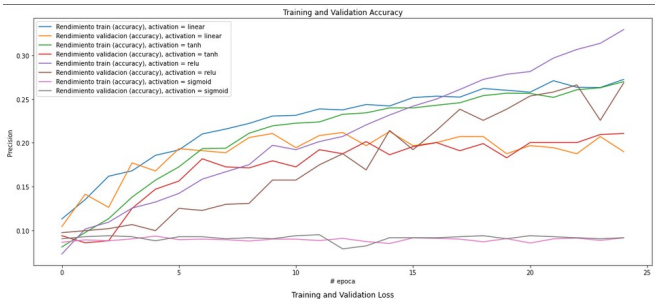

Figura 5. Rendimiento en validación y train para las diferentes funciones de activación.

Como podemos ver en la figura 5, el rendimiento varía bastante según la función de activación que escojamos. De primeras, sin ver las pérdidas, podemos descartar la función sigmoid, ya que vemos que el rendimiento es bastante bajo, esto se debe a que sigmoid suele usarse para problemas binarios, y estos datos que tenemos nos proporcionan imágenes de diferentes escenarios. Además podríamos descartar la función linear, ya que esta tiende a tener sobreajuste.

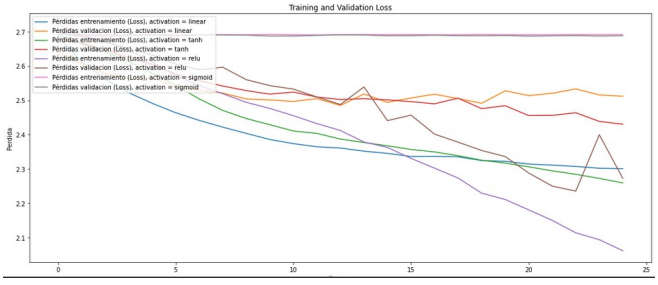

Figura 6. Pérdidas en validación y train para las diferentes funciones de activación.

Finalmente, podemos ver en la figura 6 las pérdidas en las diferentes funciones de activación. Como Hemos dicho antes, la función sigmoid la descartamos, porque además en este gráfico podemos ver que las pérdidas son bastante notorias.

Después de ver el rendimiento, hemos decidido que las mejores opciones sería relu y tanh. Ambas funcionan muy bien con problemas no binarios como en el que estamos trabajando, sin embargo, relu tiene una salida mejor en cuanto a pérdidas. Esto se sebe a que tiene un entrenamiento más rápido ya que la velocidad de convergencia es más rápida que la de tanh. El único inconveniente de relu sería cuando la tasa de aprendizaje es bastante alta, pero al tratarse de un dataset pequeño como el que estamos viendo, relu es bastante eficiente.

4.4 Estudie la variación en rendimiento para diferentes opciones de *Data Augmentation* (consulta *ImageDataGenerator*[2]) durante 25 épocas. Seleccione 3-4 opciones, argumente su elección y compare el rendimiento obtenido con la red original del tutorial (que no aplica *Data Augmentation*). Utilice un valor de *batch_size* e *img_size* acorde sus conclusiones en las preguntas 4.1 y 4.2. Razone porque obtiene los resultados que observa. (1 punto)

Para la realización de este apartado utilizamos el notebook **p4_pregunta_44.ipynb.** En este fichero realizamos todas las partes necesarias para poder posteriormente entrenar y clasificar el modelo. Es decir, preparamos el entorno de trabajo, creamos y cargamos los datos necesarios y posteriormente empieza la lógica del ejercicio.

Para la realización, iteramos sobre la generación de datos, entrenamiento y clasificación para distintas configuraciones de Data Augmentation. Mantenemos el batch_size a 32 y img_size a 32x32 ya que son los valores con los que mejores resultados se han obtenido en los apartados 4.1 y 4.2. Para esta tarea hicimos una ejecución previa para 3 distintas opciones de Data Augmentation añadiendo un parámetro en cada una y comprobamos los resultados. Posteriormente, vimos cual de las tres opciones generaba mejores precisiones y pérdidas y creamos una cuarta opción con los dos mejores parámetros. Tendríamos entonces 4 opciones de Data Augmentation sobre las que vamos iterando para entrenar y clasificar. Éstas serían: aplicar flip horizontal sobre las imágenes, aplicar un rango de brillo distinto, meterle zoom a la imagen, y la última combina el zoom y el flip horizontal.

El principal objetivo de usar técnicas de Data Augmentation es incrementar el tamaño del dataset añadiendo copias ligeramente distintas de las imágenes que tenemos originalmente. Esto puede ayudar bastante a reducir el sobreajuste. El flip horizontal lo hemos utilizado ya que aunque la imagen se invierta horizontalmente, seguimos teniendo una escena válida para poder entrenar y clasificar. Los objetos que pueda haber por ejemplo en un dormitorio no van a cambiar de forma. También hemos pensado que un cambio en la iluminación podría ser una buena idea para generar más datos; ya que se está haciendo un cambio ligero y tendremos datos un poco distintos para poder entrenar mejor. Por último, meterle un zoom no muy alto también es buena opción ya que como tenemos imágenes de escenas y paisajes, seguramente haya zonas exteriores de las imágenes que no son muy relevantes y así podremos tener más datos y algunos de ellos más precisos. Hay que tener cuidado eligiendo los parámetros de Data Augmentation a utilizar y sus valores. Para datasets con otros tipos de imágenes distintos a los que tenemos en el nuestro, seguramente convenga utilizar otras opciones. Si por ejemplo tenemos un dataset con rostros de animales, puede que no nos interese meter mucho

2https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

zoom, o directamente no meterlo, ya que crearíamos datos que no benefician.

Los resultados obtenidos para las distintas opciones de Data Augmentation han sido los siguientes:

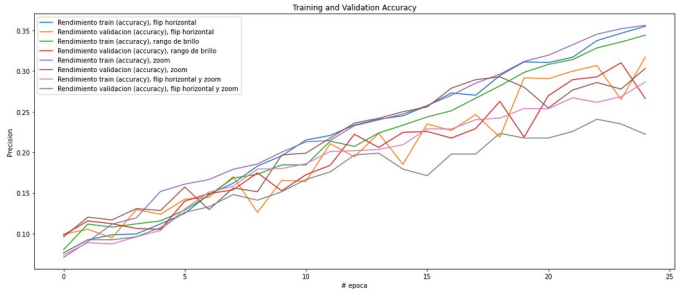

Figura 7. Precisión en validación y entrenamiento para los distintos parámetros de Data Augmentation.

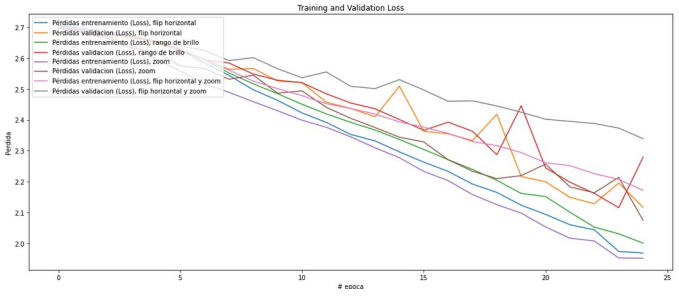

Figura 8. Pérdidas en validación y entrenamiento para los distintos parámetros de Data Augmentation.

Las gráficas obtenidas difieren bastante de los resultados del apartado 4.2 (red original, sin Data Augmentation). A simple vista, se observa en la Figura 8 que todas las curvas de pérdidas van disminuyendo y al comparar entrenamiento con validación, están próximas en todas las opciones. Esto quiere decir que no se pueden observar problemas notables de sobreajuste. En la figura 7 podemos ver que la precisión va aumentando a medida que van pasando las épocas y también se observa que entre la curva de validación y la de entrenamiento correspondiente a cada configuración, no hay una gran separación. No se está produciendo sobreajuste, el modelo está clasificando acorde con lo que ha ido entrenando. Finalmente se observa que la mejor opción de Data Augmentation es aplicando un flip horizontal sobre las imágenes.

4.5 Estudie la variación en rendimiento para diferentes "complejidades" durante 15 épocas[3]. Proponga dos opciones con menos/más parámetros que la red original, argumente su elección y compare el rendimiento obtenido con la red original. Utilice un valor de *batch_size*, *img_size* y función de activación acorde sus conclusiones en las preguntas 4.1, 4.2 y 4.3. Razone porque obtiene los resultados que observa. (1.5 puntos)

Para la realización de este apartado hemos utilizado el notebook **p4_pregunta_45.ipynb,** realizando todas las partes necesarias para entrenar y clasificar el modelo, preparando el entorno de trabajo, creando y cargando los datos, es decir, el dataset, y seleccionando el hardware a utilizar.

Para este ejercicio tenemos que estudiar el rendimiento, variando las complejidades de la red. Para hacer esto, hemos creado una lista de tuplas las cuales contienen cada una un filtro y un kernel, el cual va a ir variando en cada momento del bucle. Para este ejercicio hemos usado los filtro 16 y 32, y el kernel (7, 7) y (9, 9) .

Como dice el enunciado, para acelerar la convergencia y el tiempo de entrenamiento utilizamos el optimizador Adam con *learning rate* por defecto (0.001), y reducimos el número de épocas a 15. Además, hemos usado los valores que mejor resultado nos han generado en las preguntas 4.1, 4.2 y 4.3, siendo *batch_size* igual a 32, *img_size* igual a 32x32 y la función de activación relu.

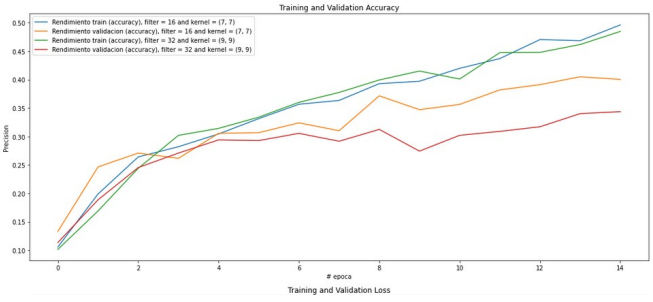

Figura 9. Rendimiento en validación y train para las diferentes filtros y kernels.

Como podemos apreciar en la figura 9, la variación del rendimiento no es muy significativa, sin embargo, el mejor resultado es con filtro 16 y kernel (7, 7), ya que como podemos ver, con filtro 32 y kernel (9, 9), se produce un sobreajuste al haber más diferencia en el rendimiento de entrenamiento y el rendimiento de validación.

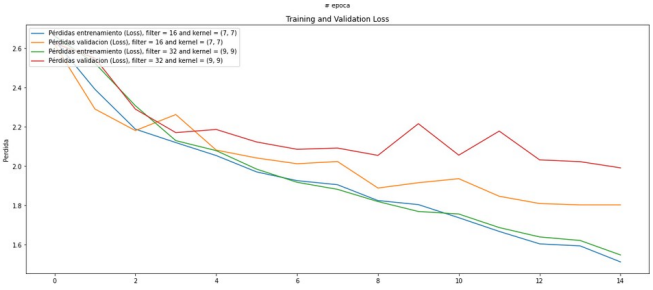

Figura 10. Pérdidas en validación y train para las diferentes filtros y kernels.

Además, podemos ver en la figura 10, como las pérdidas para el caso con menor filtro y dimensión del kernel son menores.

4.6 Aplique *Transfer Learning* al problema de clasificación de esta práctica durante 15 épocas[3]. Para ello, seleccione dos arquitecturas conocidas de clasificación[4] y añada dos capas adicionales: una capa fully-connected de 120 unidades y una capa de salida para las 15 clases. Argumente las opciones que elige y la estrategia que aplica para *Transfer Learning*. Después compare el rendimiento obtenido con la red original del tutorial. Utilice un valor de *batch_size*, *img_size* y función de activación acorde de sus conclusiones en las preguntas 4.1, 4.2 y 4.3. Razone porque obtiene los resultados que observa. (2 puntos)

Para la realización de este apartado utilizamos el notebook **p4_pregunta_46.ipynb.** En este fichero realizamos todas las partes necesarias para poder posteriormente entrenar y clasificar el modelo. Es decir, preparamos el entorno de trabajo, creamos y cargamos los datos necesarios y posteriormente empieza la lógica para aplicar Transfer Learning.

Lo primero que realizamos es definir los parámetros que mejores resultados nos han dado en apartados anteriores. Como hemos comentado previamente, el mejor batch_size es igual a 32, el mejor tamaño de imagen es igual a 32x32 y la mejor función de activación es 'ReLu'. También cambiamos el número de épocas a 15 y utilizamos el optimizador Adam con un learning rate de 0.001.

Para saber qué estrategia de Transfer Learning nos conviene aplicar, primero tenemos que indicar con qué pesos preentrenados queremos cargar el modelo. Ya que usaremos los pesos de imagenet, al tener buena similitud y ya que nuestro dataset es 'pequeño', **solamente entrenaremos el clasificador**. Al conocer qué estrategia vamos a aplicar, tenemos que definir dos arquitecturas que queramos estudiar. En nuestro caso, hemos utilizado las arquitecturas VGG16 y ResNet50. Son modelos que hemos estudiado en teoría y nos vienen bien para analizar más en profundidad su arquitectura y funcionamiento.

Antes de entrenar y clasificar, tenemos que definir de forma correcta los parámetros que se le pasarán a estos modelos. Ya que nuestro dataset utiliza imágenes de tamaño distinto al que piden estas arquitecturas. Para poder ajustar los datos al tamaño que queremos (32x32x3), quitamos las capas Fully Connected de la arquitectura, ya que son las que nos inhabilitan le utilización de otros tamaños de entrada. Esto se realiza cambiando el parámetro "include_top" al valor **False**. También tenemos que incluir los pesos preentrenados de imagenet de la siguiente forma: "weights" lo cambiamos al valor '**imagenet**'. Por último, modificamos el tamaño de los datos de entrada de esta manera: "input_shape" lo modificamos a **(32, 32, 3)**; ya que por defecto está a (224, 224, 3). De esta forma, ya tenemos los modelos casi preparados para poder realizar las demás operaciones.

El último paso a realizar antes de analizar los resultados es crear el modelo con las especificaciones 'extra' que se mencionan en el enunciado. Instanciamos un modelo secuencial y le añadimos la arquitectura que hemos definido antes (VGG16 o ResNet50). Posteriormente, mostramos un resumen del modelo para ver cuantos parámetros entrenables tiene.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 1, 1, 512)         14714688
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

Figura 11. Resumen de la arquitectura VGG16 antes de aplicar Tranfer Learning.

Como podemos observar en la Figura 11, el número de parámetros entrenables es bastante alto y eso no nos interesa demasiado en este caso. Lo que queremos es reducir ese número de parámetros para que entrene solo el clasificador y no la red entera. Para realizar eso, congelamos todas las capas de la arquitectura que hemos añadido a nuestro modelo secuencial. Posteriormente, añadimos la capa fully connected de 120 unidades y la capa de salida para las 15 clases de este problema. Mostramos otra vez el resumen del modelo para ver si cambian los parámetros.

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 1, 1, 512)         14714688
_________________________________________________________________
flatten_6 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_28 (Dense)             (None, 120)               61560
_________________________________________________________________
dense_29 (Dense)             (None, 15)                1815
=================================================================
Total params: 14,778,063
Trainable params: 63,375
Non-trainable params: 14,714,688
```

Figura 12. Resumen de la arquitectura VGG16 después de congelar y añadir las capas que se piden.

Efectivamente, al observar la Figura 12, se puede ver que el número de parámetros entrenables ha cambiado considerablemente a la cifra de 63.375. Hemos pasado de tener casi 15 millones a muchísimos menos. Esto hace que el modelo pase a ser mucho menos complejo y también ahorraremos mucho tiempo a la hora de entrenar.

Una vez hechos los pasos previos, entrenamos y clasificamos la arquitectura, se repite lo mismo también para ResNet50. Los resultados obtenidos son los siguientes:

3Para reducir el número de épocas necesarias para convergencia y acelerar el tiempo de entrenamiento, considere utilizar el optimizador ADAM con parámetros por defecto, salvo *learning rate* igual a 0.001 (https://www.tensorflow.org/api_docs/python/tf/keras/optimizers)

4https://www.tensorflow.org/api_docs/python/tf/keras/applications

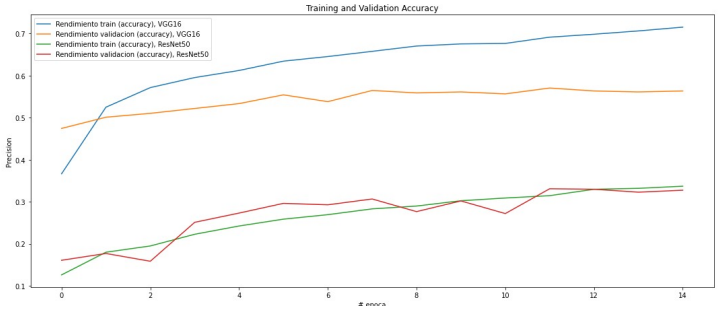

Figura 13. Precisión en entrenamiento y validación para las arquitecturas con VGG16 y ResNet50 tras aplicar Transfer Learning.

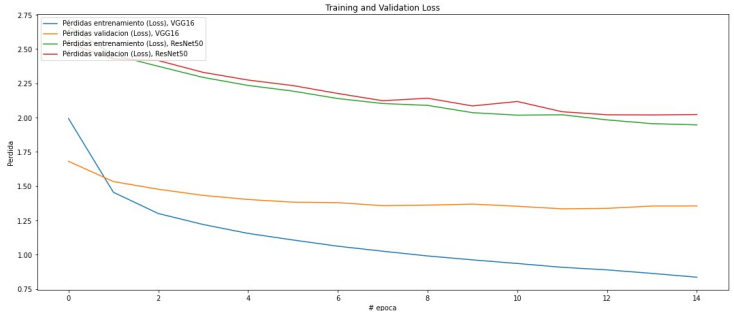

Figura 14. Pérdidas en entrenamiento y validación para las arquitecturas con VGG16 y ResNet50 ras aplicar Transfer Learning.

Tanto en la Figura 13 como en la Figura 14, se puede observar que el modelo que utiliza la arquitectura VGG16 saca resultados bastante mejores. Aún así, los dos casos se están sobreajustando ya que la precisión en validación se acaba manteniendo constante. Al ver las pérdidas se puede observar que van a mejor, ya que están disminuyendo, pero ocurre muy parecido, a partir de un punto se mantienen casi constantes. Esto se debe a que ambos modelos son bastante complejos y tampoco tenemos una gran cantidad de datos para probar.

En el output del notebook, podemos ver también el número de parámetros entrenables con ResNet50. En ese caso, tenemos un total de 247.695 después de congelar y añadir las dos capas. Comparado con VGG16, es el cuádruple de parámetros que se usan para entrenar, por lo tanto al ser más complejo, está sacando resultados mucho peores.

CARGA DE TRABAJO

Indique brevemente la carga (en horas) de cada tarea de esta práctica. Puede utilizar como ejemplo la Tabla 1.

Tarea	Horas dedicadas (Jaime Pascual)	Horas dedicadas (Juan Moreno)
P4.1	3.5h	3.5h
P4.2	2h	2.5h
P4.3	2.5h	2h
P4.4	2h	2.5h
P4.5	3h	2.5h
P4.6	3.5h	4h
Total	16.5h	17h

Tabla 1. Carga de trabajo.

REFERENCIAS