Práctica 4 – Reconocimiento de escenas con Redes Convolucionales Neuronales

Fernández, Luis – Juez, Rodrigo

4 PREGUNTAS TAREAS 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 los histogramas comparando el Accuracy y el Loss final se ha usado la función *evaluate_generator()* sobre los splits de train y test, podríamos haber usado el último valor del historial al entrenar el modelo (es decir, el de la última época), pero consideramos que era más seguro inferir con el modelo finalizado y calcular el rendimiento.

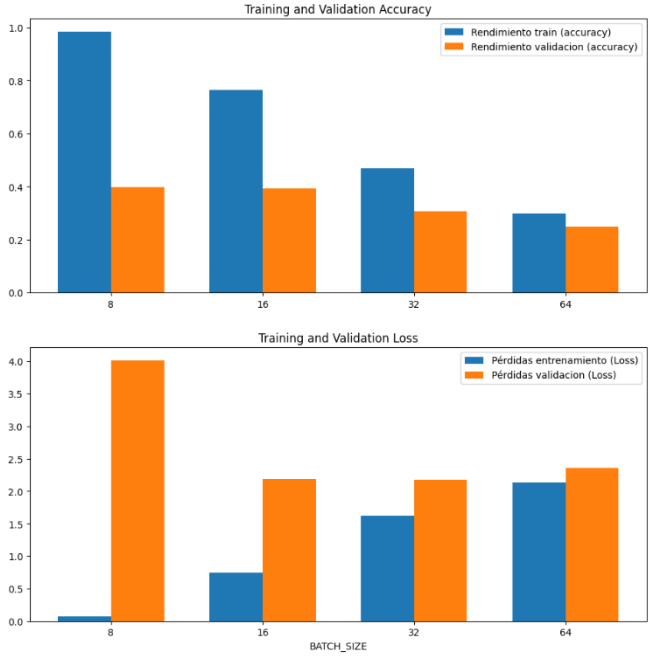

Figura 1. Comparación accuracy Training y Validation Accuracy.

En este caso se puede observar que tanto el mini-batch de 8 como de 16 tienen un rendimiento muy similar, sin embargo, la de 8 obtuvo un mejor accuracy en el Split de validación (mirando el accuracy en números, en la gráfica es difícil de discernir), a pesar de haber ejecutado varias veces en un mayor número de veces ganó 8, por eso se eligió esta opción para continuar adelante, sin embargo, también podría haber sido una opción válida 16 ya que tiene un menor loss. Como usamos la función de C*rossentropy loss* esto quiere decir que cuando se equivoca, el mini batch de 16 se aleja menos de la probabilidad correcta. Esto es bueno para el entrenamiento, sin embargo, para evaluar qué modelo es mejor consideramos que el accuracy final es el más representativo.

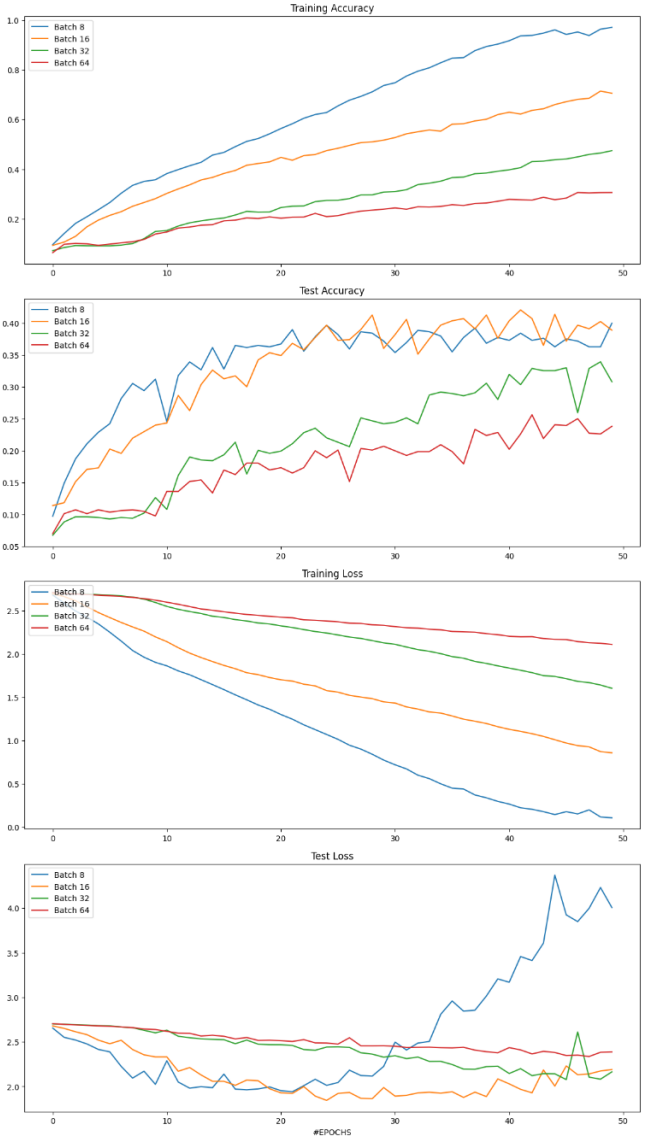

Figura 2. Evolución diferentes tipos de batch durante épocas

En la Figura 2 hemos comparado los 4 tipos de *batch* a lo largo de las épocas, se puede ver como el batch 8 converge mucho más rápido que el resto en el *split* de *training* (se observa tanto en el accuracy como en el *loss*, que no tiene por qué tener relación 1:1 ya que la función de loss no se basa únicamente en si acierta o no como ya hemos explicado anteriormente). Esta misma tendencia se puede ver en la validación donde batch 8 converge ligeramente antes que sus alternativas.

Este comportamiento nos choca mucho ya que en la página 11 de la teoría de entrenamiento de *CNNs* (diapositiva 22) vemos como un tamaño mayor de mini batches debería converger más rápido y de manera más estable, sin embargo, en este caso el loss del batch-64 disminuye mucho más lento y ni si quiera le da tiempo a converger (razón por la que creemos que tiene peor resultado).

Puede ser que como realiza menos ajustes por cada iteración (cuanto más grande el batch menos steps) no le dé tiempo en solo 50 iteraciones a hacerlo.

Finalmente, en la evolución de loss en validación, el comportamiento de batch8 es bastante curioso, se puede deber a que los batches de 8 ajusten demasiado cuando les dejamos hasta 50 épocas y la red coge confianza con las clasificaciones del *training* y la razón de que aumente el loss es que coja confianza en etiquetar incorrectamente.

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 el resto de los apartados se ha elegido batch 8 por las razones ya explicadas (converge más rápido y tiene mayor accuracy).

Volvemos a ver mucho overfitting en estas pruebas también. En esta ocasión todos los tamaños de imagen mayores a 64 dan resultados muy similares, sin embargo, el tamaño 224 obtiene un 3% mejor accuracy que cualquier otro (de nuevo, mirando los porcentajes directamente).

Es una historia muy diferente cuando miramos los resultados finales en el *loss*, todas ellas tienen valores muy similares. La función de loss es minimizada hasta prácticamente ser 0, esto significa que a mayor tamaño de imagen consigue minimizar la función de loss en un menor número de épocas. Esto podría ser overfitting ya que la función de pérdidas en la validación aumenta en algunos casos.

En la Figura 4 podemos observar una comparación igual a la de la pregunta 1. En esta ocasión vemos que el tamaño original (32 píxeles) era demasiado pequeño ya que no

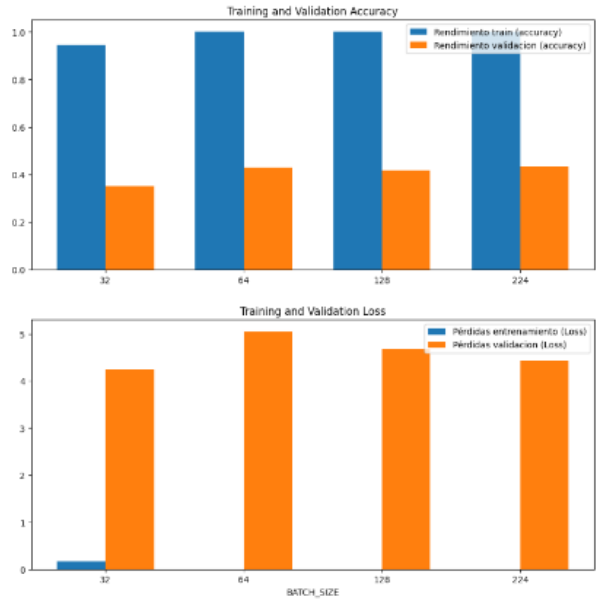

Figura 3. Comparación accuracy Training y Validation Accuracy.

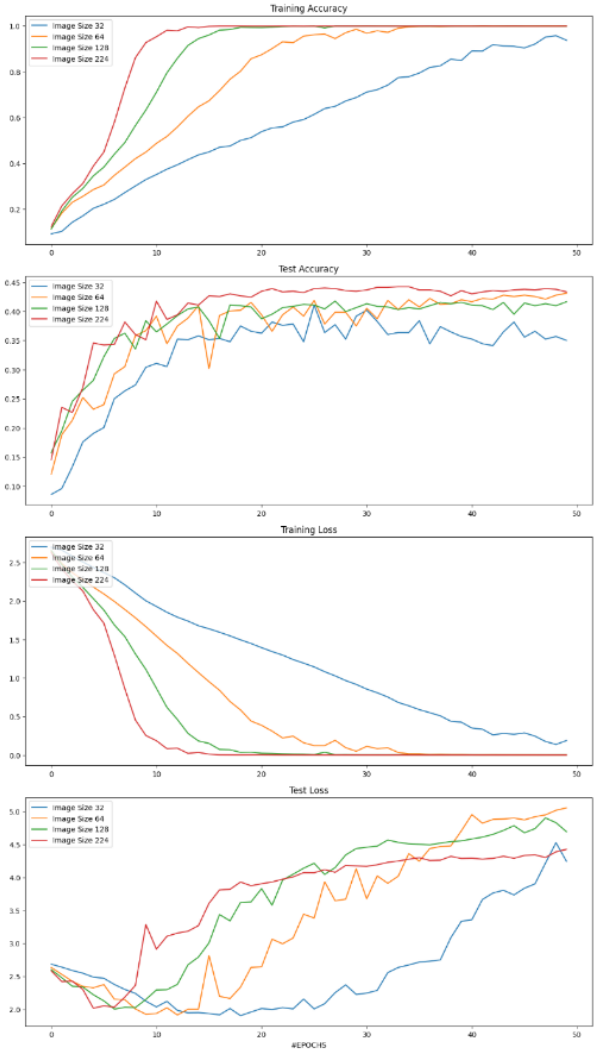

Figura 4. Comparación diferentes tamaños de imagen durante épocas de entrenamiento

consigue converger antes de las 50 iteraciones, mientras que los otros tamaños convergen mucho más rápido, especialmente el tamaño de 224.

Observando la validación no podemos confirmar lo anteriormente explicado ya que si que llegan todos a una zona de puntuación estable. Aún así la imagen 32 es la que más se queda atrás

Respecto al training *loss* vemos una historia muy similar que, con el accuracy, 128 y 224 convergen en muy pocas iteraciones, 64 en un poco más, pero le da tiempo y 32 no consigue converger en 50 iteraciones.

La evolución de pérdidas de la validación no se ve muy modificada, sin embargo, sí que vemos que la de imagen 32 es la que menor perdidas da, aunque cuando llegan las 50 iteraciones todas están muy emparejadas. Es interesante ver que, aunque la imagen de 224 no es la que más minimiza las perdidas en validación es la que converge más rápido ya que apenas varia a partir de 25 iteraciones.

Para futuros ejercicios decidimos usar imágenes de 224 de tamaño, esto es porque converge mucho más rápido que las demás (y en algunos ejercicios vamos a usar solo 25 y 15 épocas), y tiene un rendimiento superior al resto, solo 64 consigue acercarse, pero con un mayor número de épocas.

El tamaño elegido incurre en un mayor coste computacional, sin embargo, tampoco había una diferencia de tiempo tan notable como para que fuese un impedimento

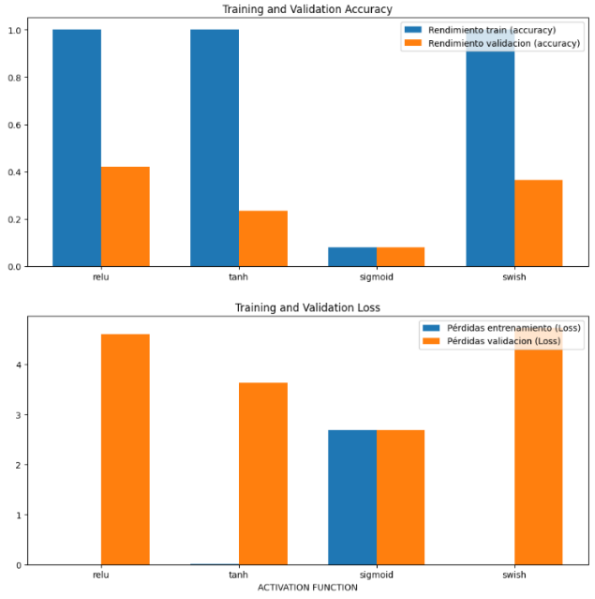

Figura 5. Comparación accuracy Training y Validation Accuracy.

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 opciones 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)

Se han cambiado las funciones de activación de las capas 1, 3, 5 y 6, las capas 2 y 4 son de pooling, y la capa de output se ha dejado con "*softmax*".

Los índices de capas usados no cuentan layers.*Flatten* como una capa ya que normalmente en las visualizaciones conectan las fully connected a las convolucionales sin mostrar el paso de aplanamiento.

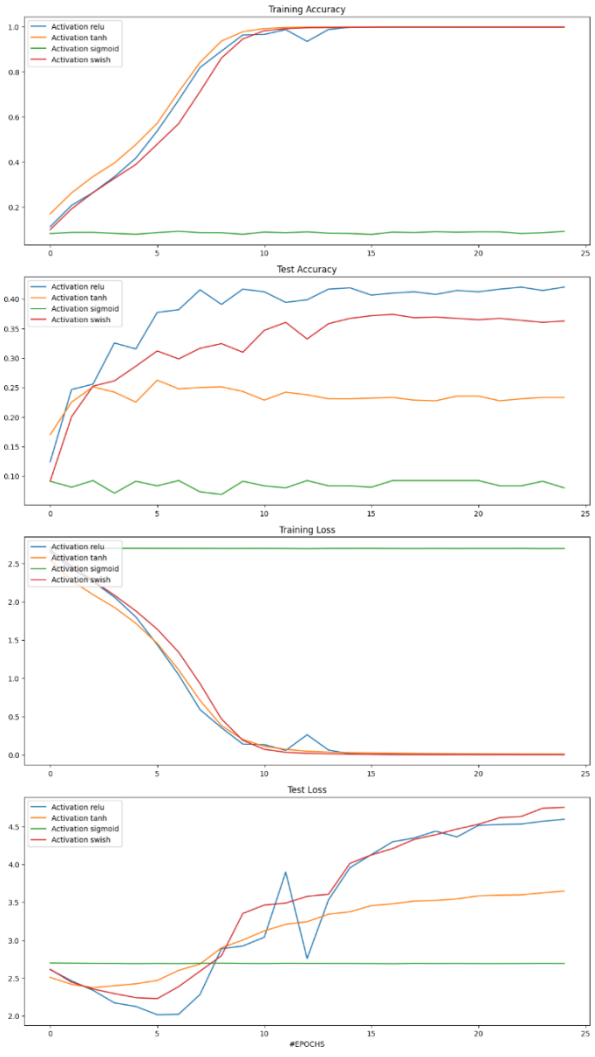

Figura 6. Evolución de distintas funciones de activación a lo largo de las épocas.

La función que mejor rendimiento da es *relu*, aunque *swish* da un rendimiento muy similar y *tanh* no es un completo desastre, sin embargo, la función *sigmoid* da un resultado peor a si eligiéramos clases aleatorias. A continuación, detallamos cómo opera cada función de activación y por qué funciona así en cada red.

"relu":

La función de activación "**relu**" tiene un rango de output entre 0 e ∞.

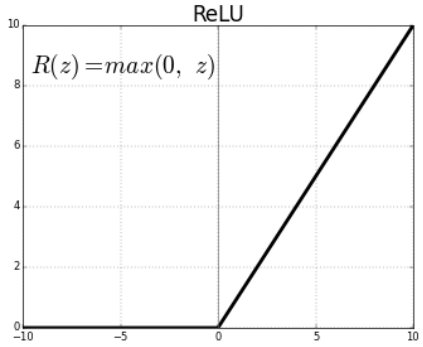

Figura 7. Representación función de activación relu

Gracias a esto, el gradiente es más grande cuando x es mayor a 0. También permite que el entrenamiento sea más rápido (esto se puede observar viendo la validación en la figura 6, *relu* es la que mejor accuracy obtiene en el menor número de épocas) y que se necesiten menos parámetros pues muchos filtrados valen 0. Además, es comúnmente conocida por ser útil para tratar imágenes y esto se puede ver en los resultados finales (en caso de funciones lógicas esto no es cierto).

"tanh":

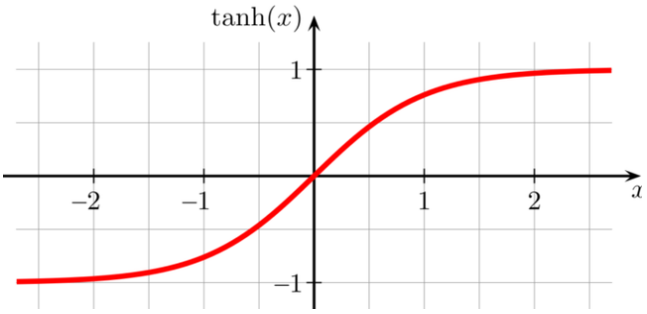

Figura 8. Representación función de activación tanh

Con la "**tanh**" obtenemos valores entre -1 y 1, por lo que se encuentra en un rango limitado y encima está centrada en 0. Sin embargo, presenta el problema de la derivación del gradiente para cuando se alcanza valores de saturación, lo que produce errores de propagación del algoritmo hacia atrás y provoca que el error se transmita al resto de capas.

Creemos que el rendimiento más bajo que las otras opciones es debido a esto. En la Figura 5. vemos que el accuracy y el loss en el conjunto de entrenamiento es prácticamente 1 sin embargo a la hora de comprobarlo en la validación no consigue los mismos resultados, esto es un signo de overfitting, que se puede confirmar viendo el conjunto de validación evolucionar en las épocas, este no mejora si no que se estanca y empeora, seguramente debido a lo explicado anteriormente.

"sigmoid":

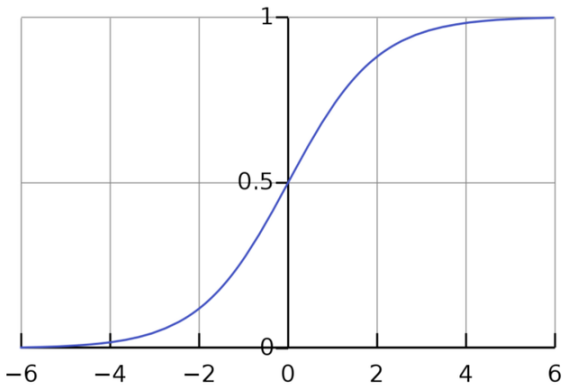

Figura 9. Representación función de activación sigmoid

Con la sigmoid obtenemos valores entre 0 y 1, por lo que se encuentra en un rango limitado y su resultado se interpreta como una probabilidad salida interpretada como una probabilidad. Pero presenta serios problemas dado que no está centrada en 0 sino en 0,5, también presenta problemas de derivación debido a las regiones de saturación de datos lo que dificulta el aprendizaje de esta y no es buena para datos de entrada que son imágenes como es este caso. Aun así, esto no explica porque *sigmoid* rinde de manera tan pobre.

Creemos que podría ser debido al problema de desvanecimiento de gradiente, es decir lo que hemos explicado de que se satura en los valores límites cuando aplicamos la regla de la cadena, sin embargo, *tanh* tiene ese mismo problema y no tiene un comportamiento tan errático.

Tras mucho buscar por internet no hemos conseguido averiguar la razón de este comportamiento, solo podemos asumir que el rango en el que se mueve *sigmoid* (0,1) es demasiado pequeño y está en constante saturación y es incapaz de entrenar correctamente, en *tanh* no es tan problemático porque tiene más margen en (-1, 1).

La función de activación del "**swish**" lo que hace es realizar por cada valor del vector pasado es calcular su correspondiente valor con la función sigmoidal y multiplicarlo por dicho valor, es decir la siguiente ecuación:

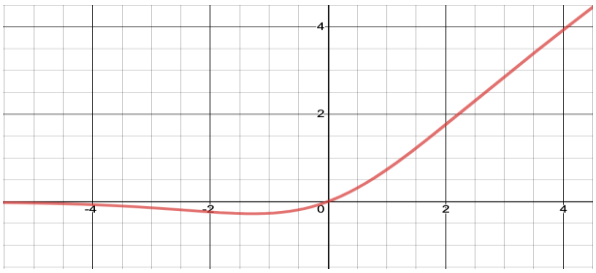

Figura 10. Representación función de activación swish.

Con la "**swish**" se solucionan los problemas de saturación ya que se obtienen valores entre -0 e ∞ (- 0 porque cuando x tiende a -∞ tiende a converger a 0 desde los números negativos, es decir y(-∞) = -∞ · 0 -> número negativo muy pequeño y cercano a 0). Gracias a esto, puede utilizarse para datos de entrada que sean imágenes y su centrado se aproxima a 0. Además, produce un comportamiento similar a la activación de la "**relu**", pero no mejor como se ha podido ver en las gráficas.

Tras esto, la capa de salida realiza una función de activación de "**softmax**", esta activación lo que realiza es pasar todos los valores del vector pasado a valores pertenecientes al intervalo [0, 1] y las sumas de todos esos valores da 1.

En resumen, *relu* es la mejor función de activación para redes convolucionales que procesan imágenes y eso se ve claramente en las figuras 5 y 6, todas ellas overfittean y consiguen un accuracy de 1 en training, pero esto es una característica que llevamos arrastrando de otros parámetros y no tiene que ver con la función de activación.

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)

Hemos probado 3 técnicas de Data Augmentation diferentes, la 4ª incluye todas a la vez y la 1ª es el dataset sin alterar, tal y como lo hemos estado usando hasta ahora.

- **"default"**: sin alterar.
- **"zoom_range"**: zoom_range = [0.5, 1]
- **"rotation"**: rotation_range = 90, "shift": fill_mode="constant", width_shift_range=0.3 y height_shift_range=0.3.
- **"all"**: todas las anteriores a la vez.

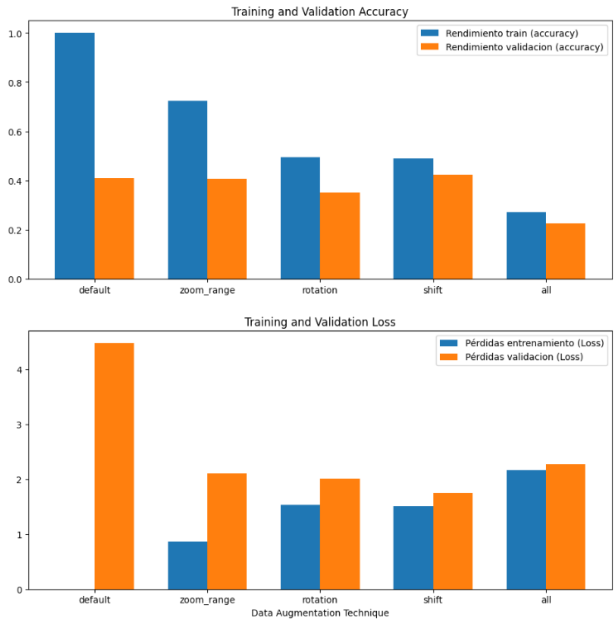

Figura 11. Comparación accuracy Training y Validation Accuracy.

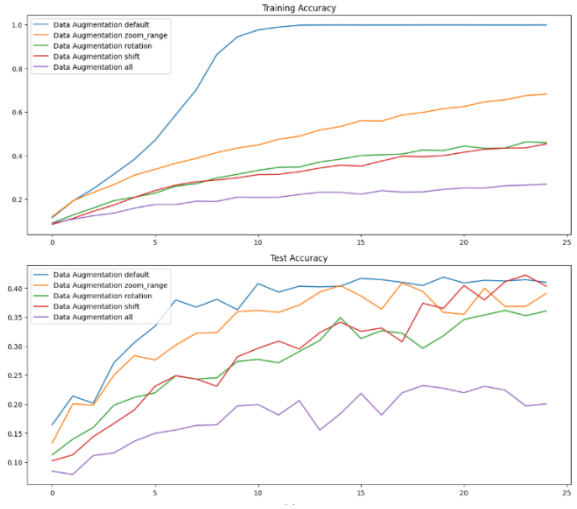

Figura 12. Evolución de la precisión a lo largo de las épocas con distintas técnicas de Data Augmentation

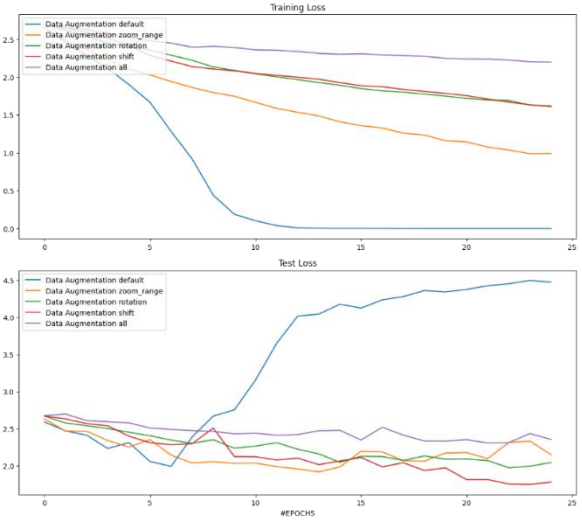

Figura 13. Evolución de la función de pérdidas a lo largo de las épocas con distintas técnicas de Data Augmentation

Si observamos solo los resultados finales (Figura 11) podemos ver cómo realizar *Data Augmentation* disminuye el overfitting, ya que en la mayoría de los casos el accuracy en el validation set no se ve muy perjudicado mientras que el del train set se ve seriamente perjudicado.

Esto no nos cuenta la historia completa ya que la Figura 12 nos muestra que los modelos con Data Augmentation han no han llegado a un punto de estabilidad como si lo ha hecho el *default*. La función de loss reporta un comportamiento muy similar al del accuracy.
Tensorflow realiza *Data Augmentation* transformando las imágenes originales por cada época, entonces en cada iteración la cantidad de imágenes es igual a la original, pero están transformadas de una manera aleatoria.

Esto último es la razón por la que el overfitting se reduce en gran cantidad y el accuracy del conjunto de entrenamiento se ve reducido. Como en cada iteración las imágenes son modificadas ligeramente, la red no es capaz de ajustar los pesos tan exactamente, y por eso en la Figura 12 todos los que tienen Data Augmentation tardan más en converger.

Hemos notado que esto aumenta el tiempo de ejecución entre época y época, ya que tiene que realizar todas las transformaciones.

¿Por qué empeora al rotar la imagen?

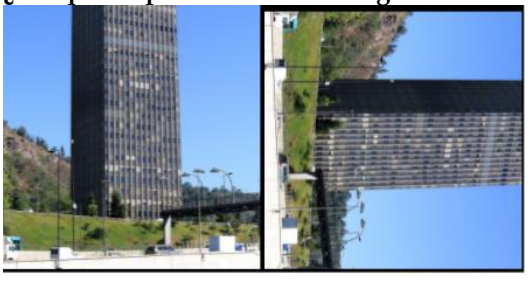

Figura 14. Ejemplo de fotografía rotada

La mayoría de las imágenes de nuestro dataset son de paisajes, habitaciones, edificios, rotar una imagen, no ayuda, ya que las fotos siempre se toman en horizontal, puede que alguna esté un poco torcida, sin embargo, girarlas hasta 90º es inútil ya que va a aprender a identificar fotos que no están en la validación y empeora en las que sí que tiene que aprender realmente.

En la Figura 14 tenemos un ejemplo de un edificio rotado, ninguna foto que recibimos viene así así que es inútil.

Solo consideraría este tipo de Augmentation si fuésemos a hacer un deploy para aplicaciones de uso general en la que hagas una foto y el giroscopio del móvil/cámara no detecte bien la orientación y la guarde incorrectamente, lo cual me ha pasado alguna vez en mi dispositivo personal, pero no es el caso con el dataset con el que trabajamos.

¿Cómo afecta el zoom?
Esta transformación es de las que mejor rinden, no consigue superar a la red por defecto, sin embargo, observando la Figura 12 creemos que dadas más iteraciones podría conseguir un mejor resultado. Esto es porque es la que mejor resultado da en el conjunto de entrenamiento con una minimización de la función de pérdidas constante, a la vez que el accuracy en el conjunto de validación está muy cerca del "*default*".

Nosotros hemos puesto un zoom in del 50% como máximo esto es para no perder tanta información, sin embargo, podría ser que en algunas ocasiones pueda empeorar el resultado si el objeto ya está muy centrado, ya que si haces zoom a un edificio y cortas el cielo es difícil distinguir donde empieza y acaba, o algunos features.

¿Cómo afecta el shift?
Tenemos una casuística muy similar al zoom, aunque pierde datos moviendo la imagen sí que es verdad que es muy positivo para la red que disminuya el overfitting, ya que creemos que hacer shift en una imagen puede simular una foto descentrada de un edificio. Por eso hemos elegido el modo "constant" ya que es el menos disruptivo con la imagen y si dejásemos los píxeles vacíos a negro podría aprender cosas que no queremos.

Aun así, parece que es mejor idea hacer zoom ya que con esta técnica tarda más en alcanzar un rendimiento similar a la técnica de zoom y el modelo por defecto.

¿Cómo afecta aplicar todas las técnicas anteriores?
Como era de esperar el rendimiento es muy malo, tiene sentido que mezclar la rotación con el zoom y el shift no mejore la red ya que se va a entrenar con imágenes demasiado distintas a las originales.

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)

Se ha cambiado el modelo básico del tutorial para incluir el optimizador nuevo (Adam). Y a partir de ese modelo se han generado dos iguales que hemos ido modificando para hacer pruebas.

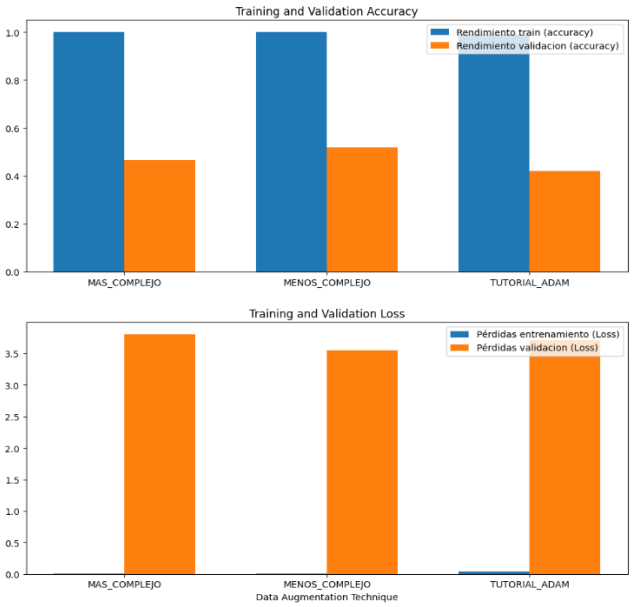

Figura 15. Comparación accuracy Training y Validation Accuracy.

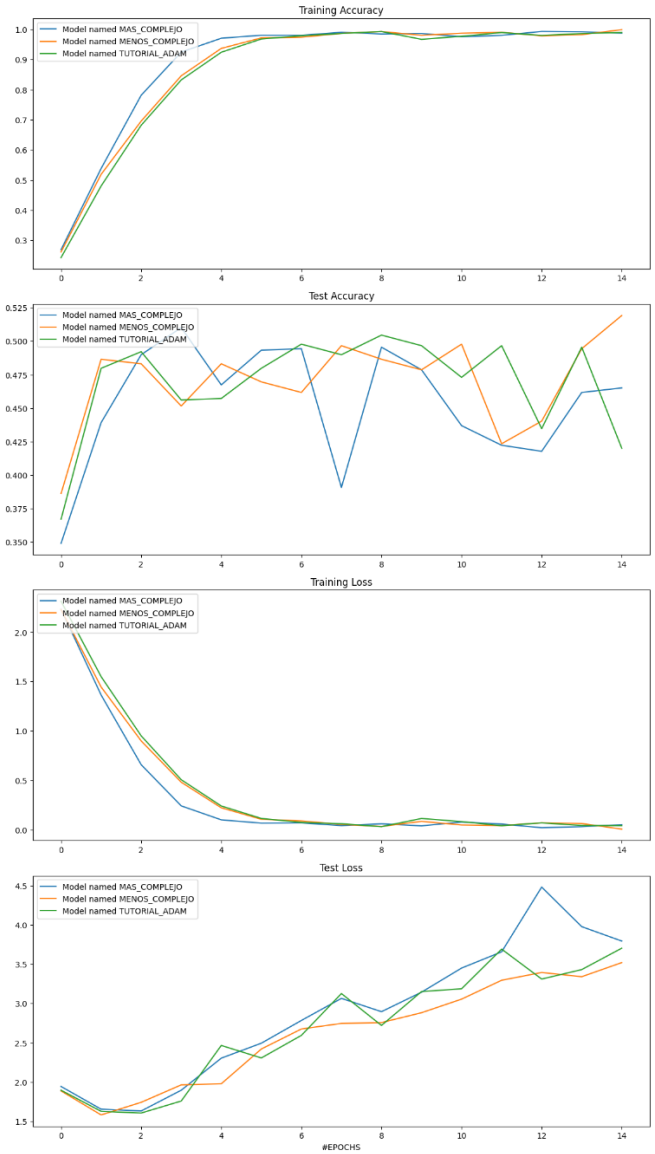

Figura 16. Evolución de las distintas variaciones del modelo base a lo largo de las épocas.

¿Qué aporta usar el Optimizador Adam?

Este optimizador utiliza un descenso de gradiente distinto al gradiente estocástico original.

El gradiente estocástico utiliza una tasa de aprendizaje única para actualizar todos los pesos, y dicha tasa no varía a lo largo del entrenamiento. Sin embargo, Adam utiliza un algoritmo de gradiente estocástico extendido adaptativo (AdaGrad) y de propagación cuadrática (RMSProp)

El primero permite reservar una tasa de aprendizaje para cada parámetro para mejorar el rendimiento en gradientes dispersos, y el segundo permite reservarlo adaptativamente para cada parámetro haciéndolo idóneo en rendimiento para problemas no estacionarios y en línea.

En la Figura 15 y 16 se ve el rendimiento de los modelos que hemos creado, los cuales solo incluyen las mejoras que aumentaron el rendimiento:

- "MAS_COMPLEJO": Tiene un mayor número de parámetros.
- "MENOS_COMPLEJO": Tiene un menor número de parámetros.
- "TUTORIAL_ADAM": Es la red que hemos estado usando hasta ahora (la del tutorial) pero con el optimizador Adam.

	Complejidad Original	Complejidad Disminuida	Complejidad Aumentada
C1	6	6	12
C3	16	12	32
F5	120	84	240
F6	84	60	168

Figura 17. Cantidad de filtros

"MAS_COMPLEJO":
Tras diversas pruebas vimos que para aumentar el rendimiento con una mayor complejidad no debíamos tocar el tamaño de los filtros, cuando probábamos con filtros de 5x5 o 11x11 el rendimiento era peor.

Estos filtros aumentan el campo de visión de la red, creemos que como no son imágenes tan grandes, las características y detalles que tienen que extraer las capas convolucionales no son muy grandes y aumentar el rango de visionado del filtro solo le confunde y añade ruido.

Sin embargo, añadir más filtros sí que fue de utilidad, estos consiguieron aumentar ligeramente el accuracy sobre el split de validación. Esto seguramente sea gracias a que ahora es capaz de detectar patrones más diversos.

También ayudó aumentar el número de neuronas en las capas Fully Connected, creemos que esto es lo que ayuda a converger antes el modelo y el mayor número de filtros a que obtenga una ligera mejoría en el accuracy de validación.

Respecto a la función de perdidas no hay mucho que comentar, simplemente, al igual que con la precisión del entrenamiento converge ligeramente antes (2 iteraciones antes).

"MENOS_COMPLEJO":
Como normalmente sufrimos de overfitting empezamos reduciendo la cantidad de neuronas que había en el clasificador, esto no afectó sustancialmente al rendimiento, sin embargo, cuando redujimos la cantidad de filtros de la C3, conseguimos, por primera vez, pasar del 50% de accuracy sobre la partición de validación. Seguramente la combinación de ambos hace que se generalice un poco mejor y por eso validación aumenta.

Respecto a la evolución durante las épocas vemos como es muy parecido a el modelo del tutorial en todos los parámetros, a diferencia del de mayor complejidad que converge antes por tener más parámetros y por ende mayor overfitting.

Conclusión:
En resumen, variar el número de parámetros ha resultado ser menos notable de lo que esperábamos, aunque el modelo con menos parámetros nos ha ayudado a generalizar un poco más el modelo y superar la barrera del 50%.

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 sus conclusiones en las preguntas 4.1, 4.2 y 4.3. Razone porque obtiene los resultados que observa. (2 puntos)

Estamos muy satisfechos con el resultado de este ejercicio ya que por primera vez en toda la práctica hemos conseguido llegar a una precisión en la validación superior al 80%, con tan solo 15 épocas. Esto ha sido gracias a la técnica de *Transfer Learning* aplicada con el modelo VGG16.

Esto es sorprendente ya que este modelo es más antiguo que ResNet50, el cual es el estándar en la industria y aun así su rendimiento no ha sido superior al del modelo por defecto (aun así, si consigue reducir el overfitting).

Para entender un poco mejor lo que está ocurriendo debemos fijarnos en cómo evoluciona la precisión y la función de pérdidas a lo largo de las iteraciones.

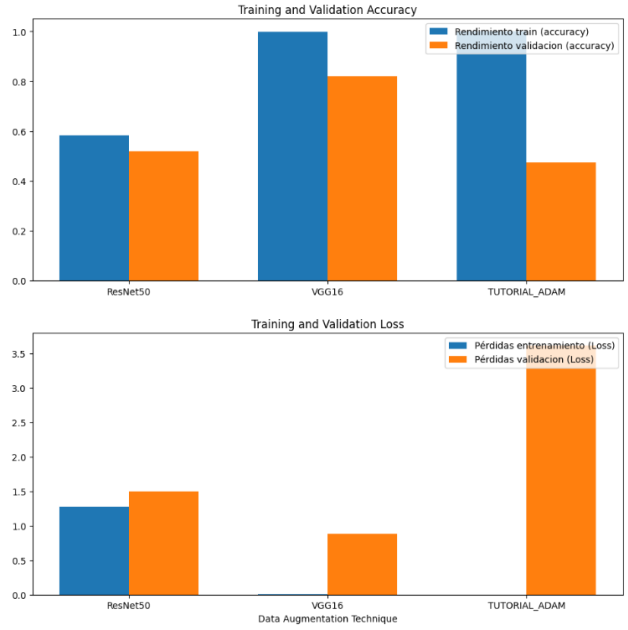

Figura 18. Comparación accuracy Training y Validation Accuracy.

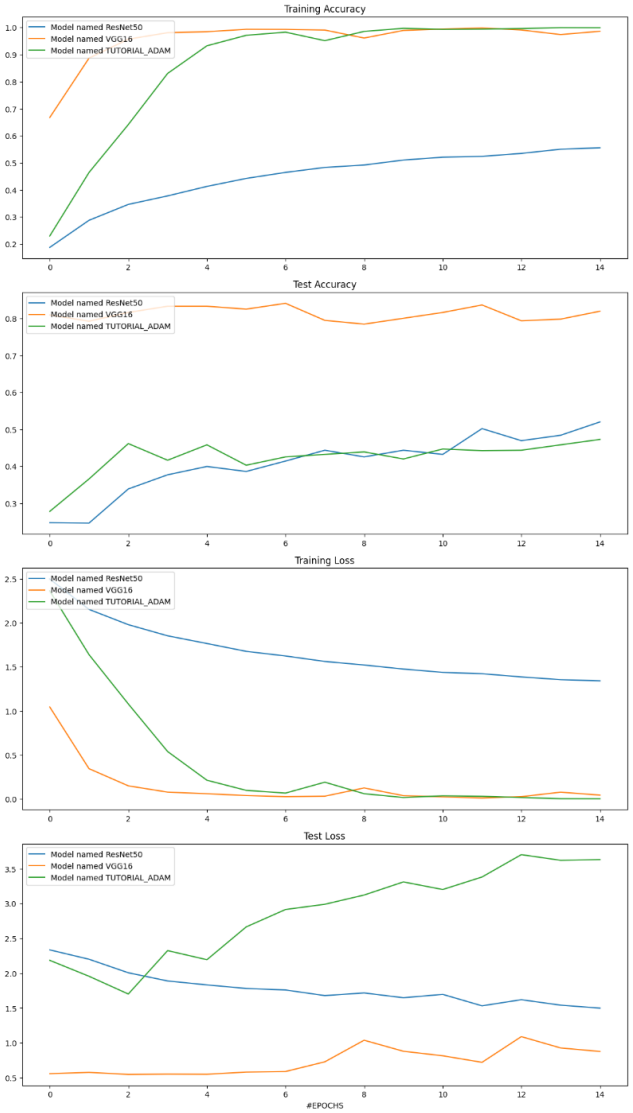

Figura 19. Evolución de los distintos modelos preentrenados a lo largo de las épocas.

ResNet no llega a converger en las 15 iteraciones, sin embargo, tanto el modelo del tutorial como VGG sí que lo hacen. Nótese que VGG16 empieza con unos valores de precisión mucho más altos, de hecho, en el conjunto de validación desde el primer momento ya tiene un resultado excelente, por lo que la inicialización es muy buena.

Tuvimos que comprobar varias veces que estábamos usando transfer learning y sustituíamos el clasificador (las capas Fully Connected) ya que nos parecía muy extraño que empezase con un 80% de precisión desde la primera época.

Además, VGG16 siempre tiene una función de perdidas muy baja, es decir que cuando se equivoca la probabilidad de la clase que debería ser la predicha no se aleja tanto.
Ahora hablaremos de porqué elegimos estos modelos:

VGG16:
Es un modelo de segunda generación, podríamos haber elegido Inception, la red de Google, sin embargo, elegimos esta porque tenía más parámetros, y como no íbamos a reentrenarla no importaba. Somos conscientes de que un mayor número de parámetros no se relaciona directamente con un mayor rendimiento, pero queríamos aprovechar la característica de transfer learning.

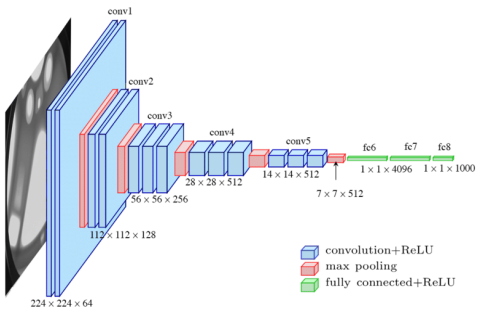

Figura 20. Arquitectura de VGG16

Además, como se puede ver en la Figura 20, la arquitectura de esta red es muy sencilla de entender y apenas tiene complejidad teórica. La mayor desventaja que tiene es que en entrenarla se tarda un tiempo desmesuradamente alto en comparación a otras, pero como ya la usamos preentrenada y nuestras capas Fully Connected tienen un número de neuronas relativamente bajo no nos incumbe.

Para adaptarla a nuestro caso de uso y realizar transfer learning importamos la red y comprobamos que, en la red original antes de eliminar el clasificador usa Flatten para conectar con dos capas *Fully Connected,* una estructura muy similar a la que usaremos nosotros (Flatten y una capa Dense + output) (como veremos otros modelos usan GlobalAveragePooling2D).

Algunos portales en internet nos recomendaron usar EfficientNet, una arquitectura de 4º generación que es mucho más rápida de entrenar, sin embargo, el tiempo de entrenamiento no nos afectaba y no conseguimos obtener un buen resultado con ninguna de las 7 variantes. Viendo la arquitectura original vemos que el clasificador tiene una capa de Dropout y características un poco diferentes a lo que nosotros hacemos (ellos tienen dropout y directamente el output), por lo que creemos que al modificar el clasificador al que queremos nosotros no funciona bien.

ResNet50:
Por la misma razón que EfficientNet no tenía el rendimiento de VGG16 también podríamos explicar el de ResNet. Para conectar las capas Fully Connected requeridas por el enunciado, en vez de usar Flatten usamos GlobalAveragePooling2D, esto es debido a que en la arquitectura original, el clasificador tampoco tiene una capa Flatten(), de hecho hicimos una prueba y el rendimiento de usar una capa Flatten vs una GlobalAveragePooling2D para conectar con el resto de Fully Connected era menor (15% vs 55%).

Aun así, es el resultado final es mejor que aplicar el modelo del tutorial, ya que como vemos en las figuras 18 y 19 no solo el *accuracy* final es mejor, si no que la función de *loss* es menor. También nos da a entender que hay menos overfitting ya que tanto el accuracy como el loss están equilibrados entre el split de entrenamiento y validación.

Además, una ventaja respecto a VGG16 es que tarda menos en la inferencia, cuando realizamos los test finales post entrenamiento fue un 30/40% más rápida. Aun que si buscamos coste computacional bajo la mejor sin duda es la del tutorial ya que tiene muchas menos capas por lo que tanto a la hora de entrenar como de inferir es varios ordenes de magnitud más rápida.

En caso de volver a usar esta arquitectura con este dataset habrá que o aumentar el número de iteraciones para que converja o aumentar el learning rate para que avance más en menos iteraciones.

Conclusiones:
Creemos que debido a que VGG16 se parece a la red que usamos nosotros, pero con más complejidad y profundidad, a que, en la arquitectura original, el clasificador tenía la estructura clásica de capa Flatten() y luego Fully Connected y eso se parece al nuevo clasificador implementado, por eso genera, de primeras, resultados muy buenos.

Mientras que ResNet obtiene peor precisión en el conjunto de entrenamiento, pero posiblemente con más épocas le diese tiempo a converger y aumentar el rendimiento.

Además, es más rápida y mucho menos pesada de entrenar (claro que, de nuevo, estamos aplicando transfer lear-

ning y no nos incumbe el tiempo de entrenamiento de las capas CNN).

Consideramos que, en un futuro, si tenemos que hacer un proyecto que involucre pocos datos, como los del dataset que estamos usando, es mejor usar transfer learning, ya que nosotros ya teníamos la red del tutorial creada, pero estar cambiando parámetros y probando distintas opciones consume demasiado tiempo comparado con usar una red existente pre-entrenada y que obtiene mejores resultados con muy poco trabajo de adaptación.

CARGA DE TRABAJO

Tarea	**Horas dedicadas** **Rodrigo Juez Hernández**	**Horas dedicadas** **Luis Fernández Freire**
P4.1	1.5	1
P4.2	1	0.5
P4.3	1	1
P4.4	2	0.7
P4.5	2	1.5
P4.6	2	1.5
Total	9.5	6.2

Tabla 21. Carga de trabajo.

REFERENCIAS

[1] Juan Carlos San Miguel Avedillo, "Tema 4 redes CNN" *Tratamiento de Señales Multimedia I: señales visuales (TSMI)*

[2] programador clic, "Algoritmo de optimizador común (Adam SGD)" *https://programmerclick.com/article/1375926681/*

[3] Adventures in Machine Learning, "Transfer learning in TensorFlow 2 tutorial" *https://adventuresinmachinelearning.com/transfer-learning-tensorflow-2/*