Proyecto final

Visión por Computador

Alejandro Alonso Membrilla
Alejandro Palencia Blanco

29/01/2021

Índice

1. Introducción

La creación de conjuntos de datos correctamente etiquetados para el entrenamiento de redes neuronales de forma supervisada es una tarea muy costosa de realizar. Concretamente, en el problema de clasificación de imágenes, esto supone asignar a cada una de las imágenes la etiqueta de la clase a la que pertenece. Alternativamente, en este proyecto estudiamos el aprendizaje autosupervisado como técnica para facilitar el entrenamiento de redes neuronales convolucionales de clasificación de imágenes. Al aplicar este método se parte de conjuntos de datos no etiquetados, los cuales son mucho más fáciles de generar. Normalmente, se suele utilizar una tarea de pretexto que el modelo debe aprender a resolver. A continuación, el modelo preentrenado para resolver dicha tarea puede adaptarse a otro problema que requiera de una extracción de características similar.

Para resolver la tarea de pretexto, previamente es necesario etiquetar los datos a partir de la estructura o características de los mismos y luego entrenar el modelo con el conjunto de datos ya etiquetado de forma supervisada. La clave del aprendizaje autosupervisado es que, al contrario que en aprendizaje supervisado, el proceso de etiquetado necesario para resolver esta tarea es muchísimo menos costoso ya que se realiza de forma automática.

1.1. Resolución de puzzles

Existen varios ejemplos de tareas de pretexto que se pueden utilizar para preentrenar el modelo: colorización de imágenes, resolución de puzzles, ordenación de frames de vídeo, clasificación de imágenes en correctas o corruptas... En este proyecto, tomaremos la resolución de puzzles como nuestra tarea de pretexto.

El problema de la resolución de puzzles consiste en, dada una imagen dividida en un *grid* 3x3 de la que se extraen nueve piezas cuadradas y se desordenan, encontrar la permutación que reordena las piezas y recupera la imagen original. Esta aplicación de aprendizaje autosupervisado ya ha sido estudiada en este artículo, en el cual nos basaremos para el desarrollo de este proyecto. Para resolver esta tarea, sus autores han diseñado una red convolucional con nueve ramas siamesas a la que han puesto el nombre de Context-Free Network (CFN). Cada una de estas ramas es una copia exacta de la red AlexNet (salvo por la primera convolución, que usa un stride de 2 en lugar de 4). Todas ellas comparten todos sus pesos y son las encargadas de procesar cada una de las nueve piezas en las que se particiona la imagen, respectivamente. Después, los vectores de características obtenidos en cada rama se concatenan y se procesan a través de dos capas totalmente conectadas para finalmente llegar al clasificador softmax. Cada una de las posiciones del vector devuelto por el clasificador está asociado a una permutación concreta, luego el objetivo de esta red es devolver un vector con un 1 en la posición correspondiente a la permutación que reordena la imagen.

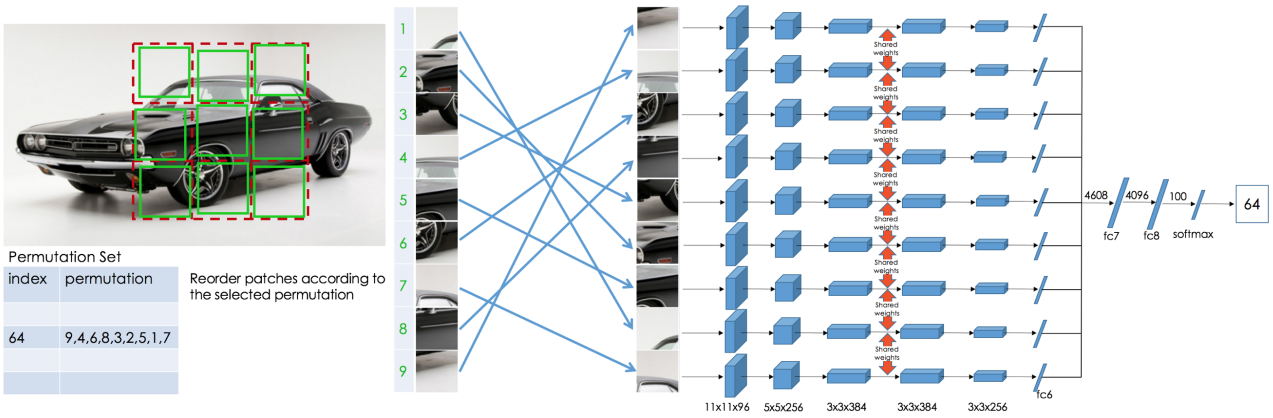

Figura 1: Arquitectura de Context-Free Network

Este modelo, mostrado en la Figura 1, fue entrenado tomando 1.3M de imágenes del conjunto ImageNet, recortándolas en un *grid* 3x3, permutando las nueve piezas resultantes y pasándoselas a la red junto con el orden correcto de las piezas, a modo de etiqueta. Partiremos de esta red convolucional ya entrenada por los autores y realizaremos sobre ella ligeras modificaciones.

2. Framework usado en el proyecto

Hemos decidido usar `Caffe` (`pyCaffe`, en particular) como framework para el desarrollo de nuestro proyecto, el mismo que usan los autores del artículo y el modelo que usamos como base. En su página oficial, podemos encontrar tanto los pesos de su red CFN en un archivo con extensión **.caffemodel** como la arquitectura de la misma en un archivo **.prototxt**. En su repositorio de Github encontramos los ficheros prototxt usados para el entrenamiento de la red.

`Caffe` permite definir ficheros con el diseño del modelo y con los parámetros del proceso de compilación (en `Caffe`, realizado por un *solver*). A partir de los ficheros con el diseño del modelo, es posible cargar los pesos preentrenados que se deseen, simplemente llamando a la capa del nuevo modelo con el mismo nombre de la capa cuyos pesos se quieran importar. `Caffe` también incluye distintos tipos de políticas de variación del ratio de aprendizaje con las que se han experimentado, en particular con el decaimiento **exponencial** y en **escalera** (más sobre esto en la sección siguiente).

Este *framework* no incluye opciones para especificar un criterio de parada, a parte de un número máximo de iteraciones. Para evitar un posible *overfitting* en el entrenamiento, hemos implementado el *early stopping* manualmente, que realizamos pasando cada *batch* a la red, de uno en uno, monitorizando la **pérdida de validación media de cada época** y deteniendo el entrenamiento cuando esta aumenta (o cuando no decrece significativamente) durante cierto número de épocas.

`Caffe`, al igual que `Keras` y `TensorFlow`, admite soporte para cómputo en GPU, que se ha aprovechado en este proyecto. El código implementado se ha ejecutado en Google Colab y en el hardware personal de los autores de este proyecto. Sus especificaciones se muestran en la Figura 2.

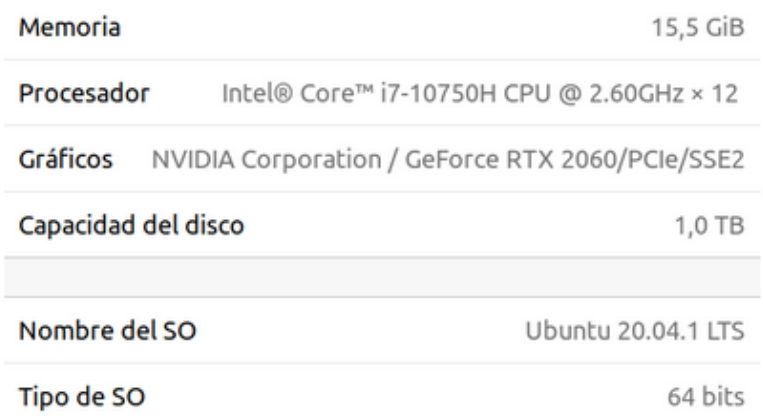

Figura 2: Hardware personal de los autores

3. Ajuste fino sobre CIFAR100

Nuestro primer experimento consistirá en tomar una de las ramas de la red CFN y adaptarla para resolver el problema de clasificación de CIFAR100 aplicándole un ajuste fino (*fine tuning*).

En lo que respecta al conjunto de datos utilizado, CIFAR100 cuenta con 100 clases distintas, cada una con 600 imágenes a color (500 para entrenamiento, 100 para test) de 32x32 píxeles de tamaño. Este dataset ha sido importado directamente desde los conjuntos de muestra que ofrece `Keras`, en forma de array de `numpy`.

A nivel de arquitectura, lo único que cambia es que se eliminan las capas *ReLU* y *Dropout* tras la capa densamente conectada, esta se cambia por otra con 100 nodos (uno por cada clase) y se añade un clasificador *softmax* al final de lo que antes era la rama de la red CFN. El resto de capas densamente conectadas, al final del modelo original, se eliminan para este entrenamiento supervisado. El diseño del modelo puede apreciarse en la Figura 3.

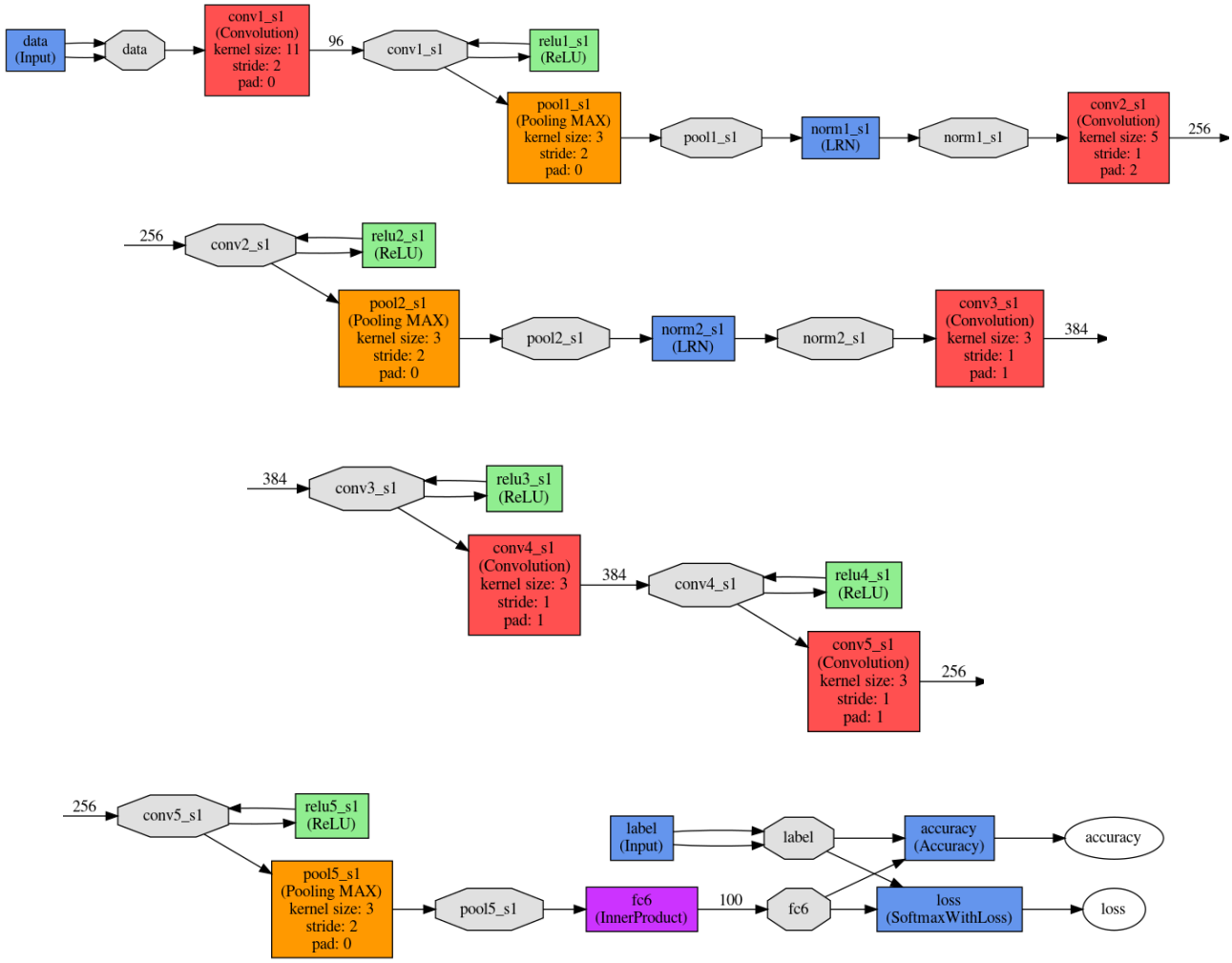

Figura 3: Arquitectura de nuestra red

3.1. Preprocesamiento de las imágenes

En primer lugar, aplicamos un preprocesamiento sobre las imágenes de los conjuntos de entrenamiento y de test redimensionándolas a un tamaño de 64x64 (el utilizado como entrada de las imágenes en cada rama del modelo original) y normalizándolas. Esta normalización se ha llevado a cabo hallando la media y la desviación típica de los píxeles de todas las imágenes de entrenamiento en cada uno de sus canales. A continuación, para cada imagen normalizamos cada uno de sus canales restando por la media total en ese canal y dividiendo por su desviación típica. Para el preprocesado del conjunto de test se han aplicado la media y la desviación típica del conjunto de entrenamiento.

3.2. Entrenamiento de la última capa

Antes de realizar un ajuste fino es necesario entrenar la nueva capa de salida fijando los pesos de las capas previas, tal como indica aquí (capítulo 5.3.2 Fine-tuning, pág. 154), para evitar una propagación del error desmedida generada en la última capa que perjudique a los pesos pre entrenados.

Para realizar el entrenamiento, hemos decidido fijar un número máximo de 200 épocas y separar un 15 % de las imágenes del conjunto de entrenamiento para usarlas como conjunto de validación. El tamaño de lote aplicado en el entrenamiento original fue de 256, pero se ha reducido a 128 para mejorar la regularización, siguiendo las indicaciones dadas aquí, aunque conservando un tamaño de lote lo suficientemente alto como para aprovechar el procesamiento paralelo de la GPU.

Para poder entrenar la capa de salida previa al *softmax*, (*fc6*), esta se inicializa usando el método de Xavier Glorot uniforme, siguiendo las indicaciones de este artículo, a diferencia de la inicialización gaussiana usada en el entrenamiento original. El resto de pesos de la red se mantienen congelados, como ya se ha explicado.

Además, con el objetivo de evitar un posible sobreajuste, aplicamos un *early stopping* con *patience* igual a 5. Hemos considerado que esta paciencia era adecuada teniendo en cuenta la política de aprendizaje utilizada (exponencial), puesto que es suficientemente alta para tolerar perturbaciones al principio del entrenamiento debidas a un *learning rate* demasiado alto, pero también lo suficientemente baja para detener el entrenamiento cuando nos alejemos de un mínimo razonablemente bueno. A su vez, se ha añadido al criterio de parada que el entrenamiento se detenga no solamente cuando la pérdida de validación no mejore, sino también cuando esta baje menos de un $\epsilon = 10^{-4}$ durante un número de épocas que supere a la paciencia anterior.

El método de optimización utilizado ha sido Adam, por sus buenos resultados generales (como se indica en este artículo). Como esta capa va a ser entrenada desde cero, hemos decidido fijar un *learning rate* de partida igual a 0.01, relativamente alto, que disminuye exponencialmente multiplicándose por una constante *gamma* igual a 0.9995 después de cada iteración (aprendizaje sobre 1 batch). Considerando que el conjunto de entrenamiento tiene 42500 imágenes y que el tamaño de lote es de 128, esto equivale a multiplicar el ratio de aprendizaje por 0.85, aproximadamente, después de cada época. También hemos fijado un *weight decay* igual a 0.001, superior al de 0.0005 usado en el entrenamiento original, para favorecer la regularización.

El entrenamiento ha durado 64 épocas. En la Figura 4 se muestra una gráfica con la evolución de la función de pérdida en entrenamiento y validación a lo largo del entrenamiento. Observamos un pico en las dos primeras iteraciones, probablemente a causa del error generado por la capa recién inicializada. Vemos que rápidamente la capa añadida empieza a aprender y las pérdidas de entrenamiento y validación comienzan a bajar.

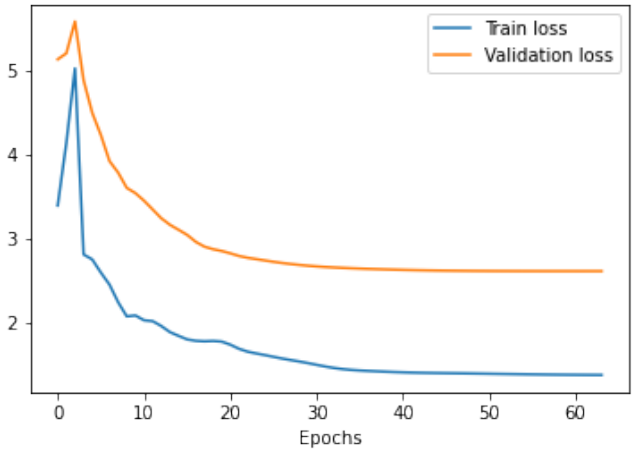

Figura 4: Evolución de la función de pérdida en entrenamiento y validación a lo largo del entrenamiento

Después de evaluar el modelo con la última capa ya entrenada, hemos obtenido una precisión del 37.66 %.

3.3. Ajuste fino

Después de haber entrenado la última capa, intentaremos aplicar un ajuste fino para mejorar los resultados de nuestra red. Para ello, aunque hemos vuelto a utilizar Adam como optimizador en el ajuste fino, hemos decidido aplicar ligeros cambios a la política de entrenamiento que se ha seguido anteriormente.

Por un lado, hemos bajado la paciencia establecida en *early stopping* a 1 ya que el número de épocas invertidas para entrenar el modelo debe ser muy bajo (un máximo establecido de 50). Para evitar que los pesos cambien de forma drástica, se ha fijado un *learning rate* base igual a 0.0005 que disminuye con una política en escalera, cuando se avanza un número de épocas prefijado, el *learning rate* es multiplicado por un *gamma* igual a 0.2. El valor de *weight decay* también lo hemos bajado a 0.0005. Como apoyo a la visualización de las políticas aplicadas al ratio de aprendizaje, tanto para este entrenamiento como para el del apartado anterior, la gráfica de la Figura 5 representa la evolución de ambos *learning rates*.

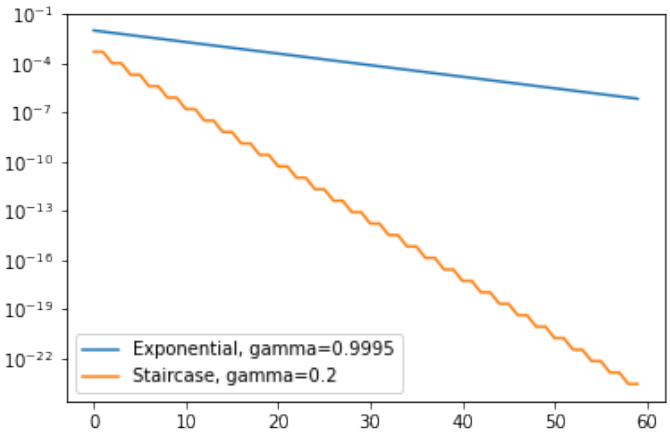

Figura 5: Evolución de políticas de *learning rates* aplicadas en el entrenamiento de la última capa y el ajuste fino. Nótese que la escala es logarítmica y que el ajuste fino se detiene en solamente 16 épocas aplicando el criterio de parada.

El entrenamiento ha durado 15 épocas. La gráfica de la Figura 6 muestra la evolución de las funciones de pérdida en entrenamiento y validación a lo largo del ajuste fino.

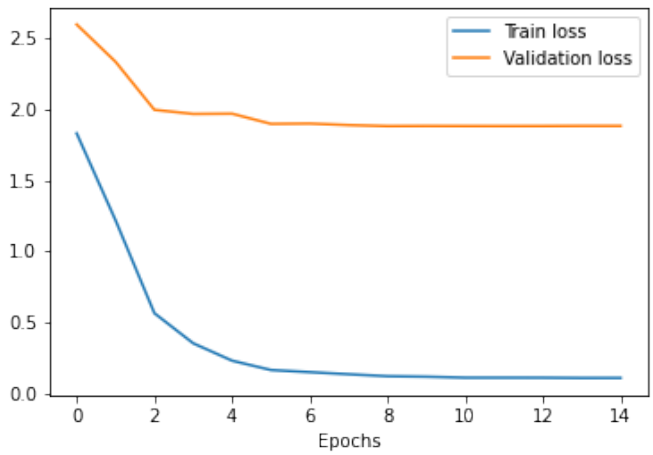

Figura 6: Evolución de las pérdidas asociadas a los conjuntos de entrenamiento y validación durante el ajuste fino

En este caso, la evaluación del modelo ha devuelto una precisión del 52.7 %, una notable mejoría con respecto a los resultados anteriores al *fine tuning*.

3.4. Filtros aprendidos

Como la red preentrenada que ajustamos ha sido entrenada de formas distintas a la habitual, nos interesa comprobar si la información que contiene es parecida a la de una red entrenada para clasificación. Hemos visualizado los filtros aprendidos en la primera capa convolucional que, como se indica en este artículo, se destina al reconocimiento de patrones geométricos y fotométricos básicos. Esto se muestra en la Figura 7, en la que podemos ver que algunos de los patrones que registran ambos conjuntos de filtros son parecidos, por ejemplo aquellos cuya forma consiste en varias bandas paralelas. Por otro lado, hay otros que solo aparecen en una de las dos redes, como aquellos que tienen forma circular, que solo aparecen en AlexNet. En general, parece que los filtros de la primera capa convolucional de la CFN son capaces de discernir patrones sencillos.

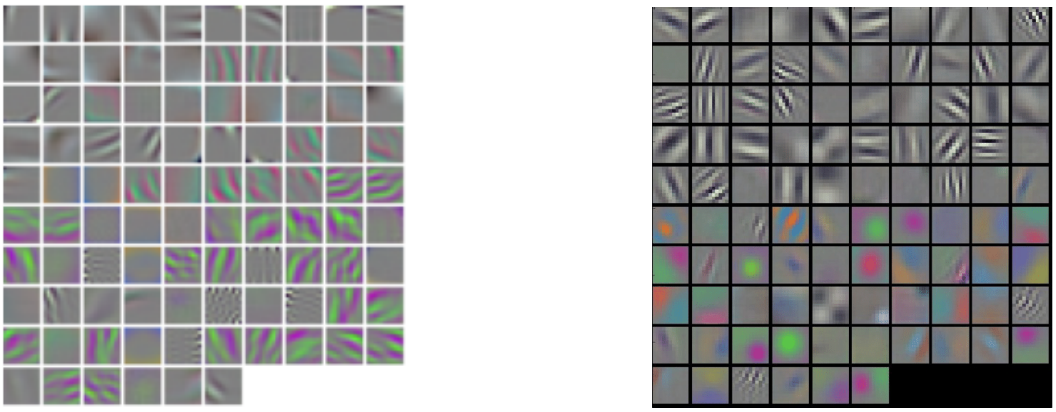

Figura 7: Comparativa de los filtros aprendidos en la primera capa convolucional (destinada a la detección de patrones simples) por la CFN (a la izquierda) y por una AlexNet estándar (a la derecha).

También hemos comparado los mismos filtros de la CFN con los filtros obtenidos por nuestra red después del ajuste fino. Como se puede apreciar en la Figura 8, los filtros apenas cambian después de llevar a cabo el ajuste fino, lo cual es algo esperable pues esta técnica no pretende modificar los pesos de la red de forma drástica.

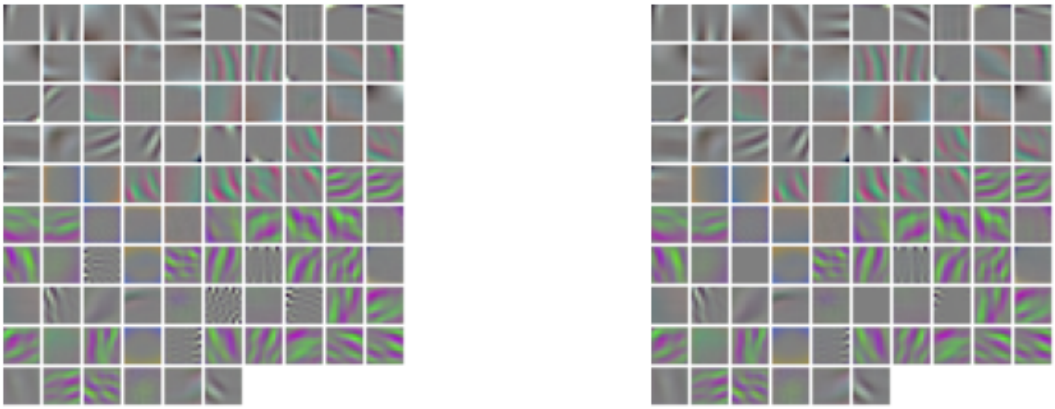

Figura 8: Comparativa de los filtros aprendidos en la primera capa convolucional por la CFN (a la izquierda) y por nuestra red después de realizar el ajuste fino (a la derecha).

3.5. Comparación con AlexNet

Es clave recordar que la red pre entrenada utilizada para este experimento se basa en AlexNet y ha sido entrenada de forma autosupervisada, es decir, nuestro objetivo es acercarnos a los resultados obtenidos por AlexNet sobre el conjunto CIFAR100 entrenada desde cero y de forma supervisada para clasificar dicho conjunto de datos.

Por un lado, la CFN comparte los mismos pesos en todas sus ramas, lo que reduce el número de parámetros de este modelo con respecto al de AlexNet, como se indica en su artículo correspondiente, de los 61M parámetros de AlexNet a solamente 27.5M parámetros. La estructura en ramas de la CFN es necesaria para la extracción

de características de todas las partes de una imagen para aprender a resolver los puzzles, pero el menor número de parámetros además permite un aprendizaje más rápido.

Por otro lado, de acuerdo con este artículo en el que se muestran, entre otras cosas, los resultados de AlexNetC100 (AlexNet entrenada con CIFAR100), esta red consigue en torno a un 54 % de precisión.

Si comparamos estos resultados con los obtenidos a partir de la red entrenada de forma autosupervisada, vemos que nuestro error en test es ligeramente mayor (un 1.3 %), pero teniendo en cuenta la desventaja del modelo del que partimos (falta de anotaciones y entrenamiento a partir de imágenes de un conjunto de datos distinto) consideramos que nuestros resultado son positivos en favor del aprendizaje autosupervisado con la resolución de puzzles como pretexto.

4. Ajuste fino sobre FRUITS360

El conjunto CIFAR100 es un dataset variado y relativamente complejo. Los resultados obtenidos en el experimento anterior son relativamente buenos si los comparamos con los de AlexNet, pero un error del 47.3 % no ofrece garantías en un problema real. Como complemento al experimento anterior, hemos probado a repetirlo con la misma red (una rama de la CFN) con otro conjunto de datos, Fruits360, que contiene imágenes de un gran número de frutas, cada una desde distintos ángulos.

4.1. Preprocesamiento de las imágenes

Este conjunto de datos contiene más de 90000 imágenes, que no ha sido posible preprocesar conjuntamente debido a las limitaciones de memoria RAM y de velocidad de flujo de datos en Colab. Por tanto, hemos tomado la decisión de reducir el conjunto de entrenamiento a una cuarta parte, lo cual no debería suponer un perjuicio a la validez del experimento debido al enorme parecido entre muchas de las imágenes, que corresponden a la misma fruta desde ángulos muy cercanos. En la Figura 9 tomamos una pequeña muestra de las 8 primeras imágenes correspondientes a la clase *Apple Golden* en el conjunto de entrenamiento.

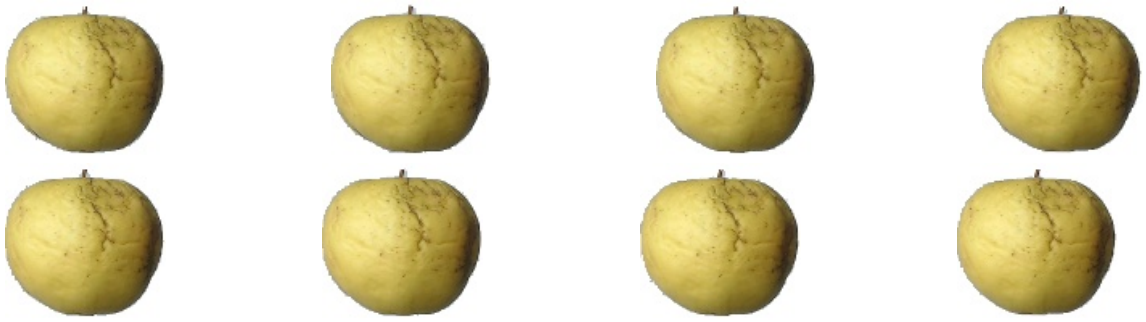

Figura 9: Muestra de las 8 primeras imágenes de la clase *Apple Golden* en el conjunto de entrenamiento

La criba se ha realizado recorriendo los directorios que se encuentran dentro del directorio de entrenamiento del dataset (uno por cada clase) y leyendo solamente 1 de cada 4 imágenes. También se ha reducido el número de frutas, de 131 clases de frutas distintas a 81 clases. Cada imagen ha sido redimensionada a 64x64 píxeles, buscando que la red aproveche sus pesos aprendidos para extracción de características de forma óptima. Las imágenes se barajan y, como en el experimento anterior, un 15 % de las imágenes del conjunto de entrenamiento se separa y se destina a validación.

4.2. Entrenamiento de la última capa

Las condiciones del entrenamiento para la capa de salida, con el resto de la red congelada, han sido las mismas que en el análogo para CIFAR100, salvo por la necesidad de cambiar la capa densa de salida a una con 81 nodos en correspondencia con el número de clases. El entrenamiento ha concluido en 98 épocas y la evolución de la función de pérdida puede verse en la Figura 10. La precisión en la fase de test es un 91 %.

4.3. Ajuste fino

Para el *fine tuning*, a partir de los pesos obtenidos del entrenamiento anterior para la última capa, sí se han modificado ligeramente los hiperparámetros. A raíz de los mejores resultados obtenidos en la fase anterior, se ha considerado que los resultados obtenidos eran relativamente buenos y que podría merecer la pena reducir el ratio de aprendizaje inicial, de 0.0005 a 0.0003, para evitar que la red pueda salir de un óptimo local. El valor del *stepsize* (número de lotes procesados por la red antes de la reducción del *learning rate*) se ha aumentado para tener en cuenta el mayor número de batches en este conjunto de datos (el tamaño de *batch* se ha dejado igual que en experimento con CIFAR100).

El entrenamiento ha durado 15 épocas y las funciones de pérdida de entrenamiento y validación han evolucionado según se muestra en la Figura 11. La precisión obtenida en la fase de prueba ha sido del 96.91 %

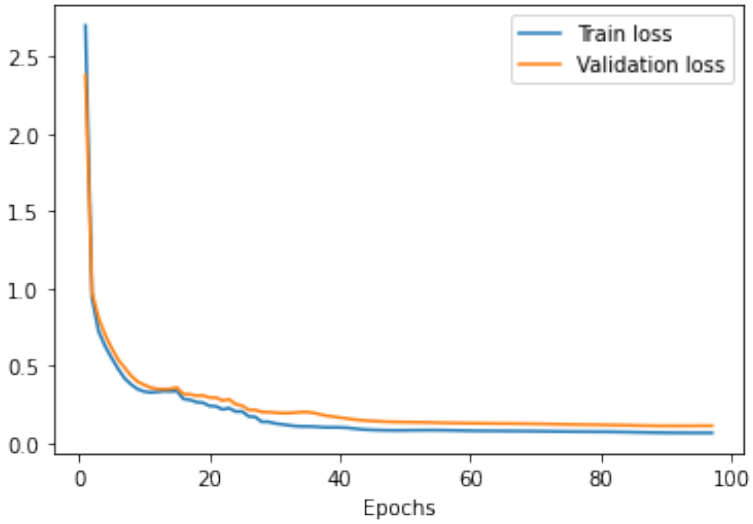

Figura 10: Evolución de las pérdidas asociadas a los conjuntos de entrenamiento y validación durante el entrenamiento de la última capa

que, a pesar de la reducción del número de clases y de la criba que se realiza posteriormente en el conjunto de entrenamiento, son resultados bastante buenos en términos generales.

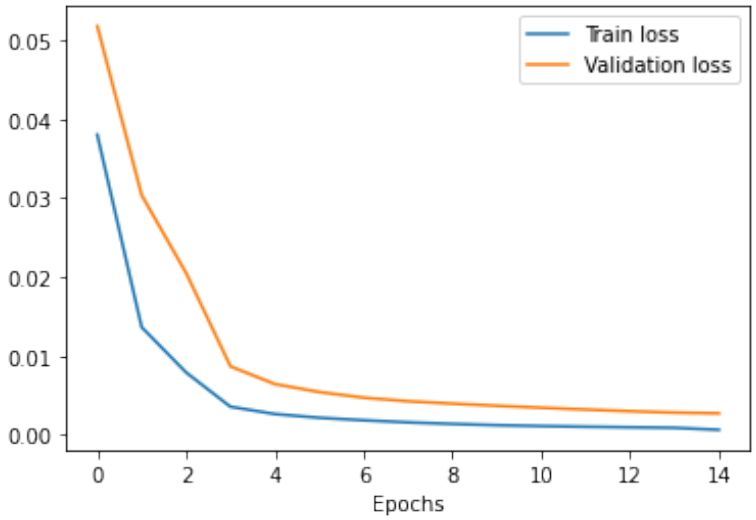

Figura 11: Evolución de las pérdidas asociadas a los conjuntos de entrenamiento y validación durante el ajuste fino

5. Conclusiones

En esta práctica hemos realizado una serie de experimentos a partir de un modelo basado en AlexNet, pre entrenado de forma auto supervisada usando como pretexto la resolución de puzzles a partir de imágenes de ImageNet. En particular, hemos adaptado el modelo a dos datasets diferentes, CIFAR100 y una versión reducida de Fruits360 con 81 clases, y hemos realizado un ajuste fino del modelo sobre ambos.

Consideramos que los resultados alcanzados han sido buenos en los dos casos. Para el conjunto CIFAR100, hemos obtenido una precisión del 52.7 % después del ajuste fino, considerablemente cercano al 54 % de precisión que obtiene AlexNet entrenado totalmente a partir de este conjunto de datos de forma supervisada. Para el conjunto Fruits360 reducido, hemos logrado un error ligeramente por encima del 3 %, lo que es un éxito en términos generales.

En conclusión, consideramos que los experimentos realizados suponen un argumento para el entrenamiento de CNNs mediante el pretexto de la resolución de puzzles, suponiendo esta una alternativa al aprendizaje supervisado que requiere de conjuntos de datos de gran tamaño etiquetados manualmente.

6. Bibliografía

1. Self-supervised learning and computer vision
2. Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles (project)
3. Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles (repositorio de Github)
4. Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles (paper)
5. Beginner's guide to Caffe
6. Fine-tuning a Pretrained Network for Style Recognition
7. Deep Learning with Python
8. Practical recommendations for gradient-based training of deep architectures
9. Understanding the difficulty of training deep feedforward neural networks
10. An overview of gradient descent optimization algorithms
11. How transferable are features in deep neural networks?
12. Deep Neural Network with Caffe
13. Network of Experts for Large-Scale Image Categorization
14. Fruits 360 dataset