Trabajo Fase 2. Deep Learning

Ignacio David López Miguel

18 de junio de 2020

Contents

1 Resumen

En la fase 1 del trabajo, se probaron diferentes aproximaciones al modelo y se propusieron tres para desarrollar en la fase 2. Ahora, se implementan estas tres aproximaciones intentando optimizar los hiper-parámetros, comparando redes diferentes con la misma aproximación y utilizando los resultados que se van obteniendo para intentar mejorar el RMSE en la muestra de test y en la de validación sin causar overfitting.

La primera aproximación es usar redes sencillas que consten de dos o tres bloques de capas convolucionales. La segunda aproximación utiliza la red VGG16 pre-entrenada con Imagenet añadiendo capas totalmente conectadas haciendo tanto transfer learning, como fine tuning. La tercera aproximación hace uso de data augmentation para intentar obtener un mejor RMSE en los modelos que tenían overfitting en las anteriores aproximaciones. En la fase 1 se dijo que esta última aproximación se haría con el modelo sencillo, pero se ha visto que también tiene sentido usarlo con el modelo pre-entrenado y hacer fine-tuning. Es de esta manera como se obtienen los mejores resultados.

2 Introducción

El objetivo del modelo de deep learning que se pretende construir en este trabajo es el de estimar el porcentaje de píxeles que contienen comida en imágenes de bandejas de comedores de la universidad de Milano-Bicoca [1]. El conjunto de imágenes que se utiliza es el UNIMIB2016 Food Database, que contiene un total de 1027 imágenes RGB de 3264×2448 píxeles y se puede encontrar aquí. En el trabajo original de los autores de este dataset, la altura de las imágenes se reduce a 320 píxeles. En el actual trabajo, también se reducirán los tamaños de las imágenes probando varios valores.

La métrica que se usará para definir la bondad del modelo será la raíz del error cuadrático medio:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{x_i} - x_i)^2}$$

El baseline para esta métrica es de $4,8$, por lo que los modelos desarrollados deberían tener un RMSE inferior.

Un ejemplo de las imágenes que contiene este dataset se puede encontrar en la figura 1, donde se ven los diferentes alimentos sobre una bandeja del comedor.

Figure 1: Ejemplo de imagen del UNIMIB2016 Food Database

3 Métodos

El código usado se encuentra tanto en Google Colab (link) como en Github [1] (link).

Para llevar a cabo el entrenamiento y comparativa entre modelos, se hace la siguiente partición aleatoria, con

- 70% (718 observaciones) entrenamiento, donde se entrena el modelo,
- 15% (154 observaciones) validación, donde se testea en cada epoch la bondad del modelo y sirve para evitar el overfitting,
- 15% (155 observaciones) test, donde se testea la bondad del modelo una vez el entrenamiento ha finalizado.

[1] Los resultados que se presentan en este trabajo se han obtenido de forma local usando una GPU sencilla de AMD (Radeon RX Vega 56). Hay ciertas diferencias en el código entre Google Colab y Github, ya que las GPU de Google Colab son NVIDIA y funcionan con CUDA (plataforma cerrada de cómputo en paralelo de NVIDIA), mientras que las GPU de AMD se usan con ROCm (plataforma de código abierto para el cómputo con GPU de AMD). Google Colab limita el tiempo de uso de sus GPU y asigna peores recursos a usuarios que dejan código ejecutándose sin interactuar con el notebook [3]. Por ese motivo, aunque se probó en innumerables ocasiones y se consiguieron muchos resultados, se decidió abandonar Google Colab y usar la GPU local por mayor conveniencia y por obtener resultados más rápido que con Google Colab.

Si el MSE en la muestra de validación deja de mejorar (0.1 MSE) durante una serie de epochs consecutivas, el entrenamiento se termina (early stopping) para evitar el overfitting. Para las dos primeras aproximaciones se usará un valor de 5 epochs sin mejora, mientras que para la última aproximación se usará el valor de 7 por ser el entrenamiento más lento al tener data augmentation. El modelo final es el que obtiene un mejor MSE, es decir, 5 o 7 epochs antes de la última iteración.

3.1 Modelo sencillo

En esta primera aproximación, se entrena un modelo que conste de varios bloques formados por dos capas convolucionales y una capa de subsampling con Max-Pooling. La arquitectura del modelo básico se puede ver en la figura 2, con dos bloques junto con dos capas completamente conectadas a la salida. En este caso, cada capa convolucional tiene 32 filtros de tamaño 3 y una capa Max-Pooling de tamaño de 4×4. La primera capa convolucional tiene 512 neuronas y la última 1 al ser un problema de regresión. La función de activación de todas las capas convolucionales y de las totalmente conectadas es relu.

Además de esta arquitectura, se probaron otras dos arquitecturas similares para ver si se podía encontrar alguna mejora:

1. añadiendo un bloque convolucional como en el esquema anterior o
2. añadiendo un bloque convolucional pero cambiando el número de los filtros, teniendo las dos primeras capas convolucionales 8 filtros, las del segundo bloque 16, y las del último 32, con la idea de ir capturando los detalles más pequeños según se va avanzando en la red.

Se intenta encontrar los mejores hiper-parámetros de estas arquitecturas probando diferentes valores para el tamaño del batch (4, 8 y 16), la altura de las imágenes (408 y 272), el optimizador (Adam y SGD) y el paso de entrenamiento (0.1, 0.01 y 0.001).

Los mejores 5 resultados para el modelo con dos bloques convolucionales iguales según el valor del RMSE en la muestra de test se pueden ver en la tabla 1. De entre estos modelos, se ha escogido el señalado en negrita por ser el que presenta resultados más similares tanto en entrenamiento como en validación.

Los primeros resultados indican cierto overfitting, como en el que tiene mejor RMSE en test, cuya curva de entrenamiento se muestra en la figura 3 y se ve que hay cierta distancia entre el MSE en entrenamiento y en validación incluso 5 epochs (early stopping) antes del final del entrenamiento.

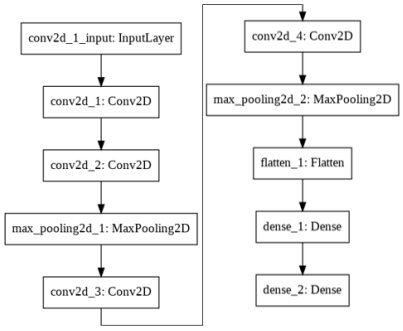

Figure 2: Arquitectura del modelo sencillo con dos bloques convolucionales

Sin embargo, en el modelo señalado en negrita, estas dos curvas están más cerca 5 epochs antes del final del entrenamiento, como se puede ver en la figura 4.

Los mejores 5 resultados para el modelo con tres bloques convolucionales iguales según el valor del RMSE en test se muestran en la tabla 2. De igual manera que en el modelo más sencillo, no se elige el modelo solo con respecto al RMSE en la muestra de test, sino también teniendo en cuenta la diferencia entre entrenamiento y validación. De esta manera, el mejor resultado sería el señalado en negrita.

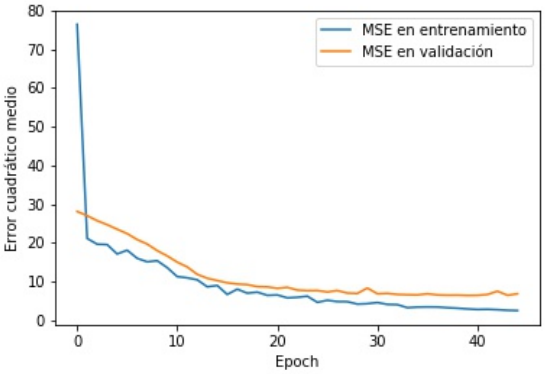

Figure 3: Curva de entrenamiento para el modelo sencillo con un poco de overfitting

Batch size	Ancho (px)	Altura (px)	Learning rate	Optimizador	Tiempo (s)	Train MSE	Val MSE	Test MSE	Test RMSE
16	544	408	0,1	Adam	375	2,87	6,35	4,99	2,23
8	544	408	0,001	Adam	224	2,44	6,56	5,13	2,26
16	544	408	0,01	Adam	252	3,80	6,33	5,51	2,35
4	**362**	**272**	**0,001**	**Adam**	**107**	**5,09**	**6,56**	**5,52**	**2,35**
4	362	272	0,01	Adam	153	4,10	6,65	5,62	2,37

Table 1: Resultados del modelo sencillo con dos bloques convolucionales iguales

Batch size	Ancho (px)	Altura (px)	Learning rate	Optimizador	Tiempo (s)	Train MSE	Val MSE	Test MSE	Test RMSE
16	544	408	0,01	Adam	693	4,78	6,50	4,63	2,15
4	544	408	0,1	Adam	586	3,97	6,43	4,98	2,23
4	544	408	0,001	Adam	519	4,89	6,45	5,04	2,24
4	**544**	**408**	**0,01**	**Adam**	**481**	**5,15**	**6,39**	**5,61**	**2,37**
16	544	408	0,001	Adam	718	5,51	5,91	5,62	2,37

Table 2: Resultados del modelo sencillo con tres bloques convolucionales iguales

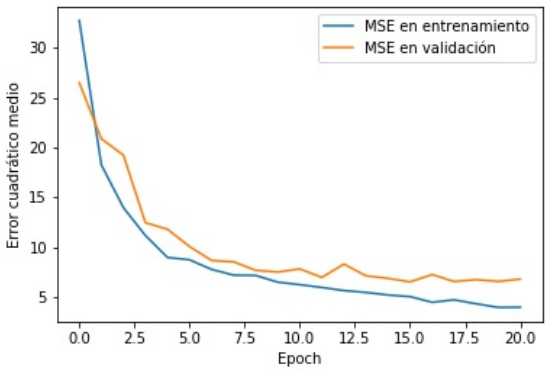

Figure 4: Curva de entrenamiento para el modelo sencillo sin overfitting

Los mejores 5 resultados para el modelo con tres bloques convolucionales con número de filtros incremental según el valor del RMSE en la muestra de test se enseñan en la tabla 3. De la misma manera que en los resultados anteriores, se escoge el modelo con mejor RMSE y diferencia entre entrenamiento y validación.

Por lo tanto, la diferencia es muy pequeña entre las tres diferentes arquitecturas y no habría casi diferencia en usar una u otra además de los tiempo de entrenamiento, en donde la primera arquitectura es más rápida al tener menos parámetros. Es importante destacar la importancia de buscar los mejores hiperparámetros, ya que hay combinaciones de hiperparámetros que resultan en un RMSE mayor que $3,5$ para cualquiera de las arquitecturas [2].

3.2 Modelo pre-entrenado

En esta aproximación, se usa la red VGG16 [2] pre-entrenada con Imagenet [4], cuya arquitectura se muestra en la figura 5 incluyendo tres capas totalmente conectadas al final de la misma para adaptar el problema a nuestro caso, la primera con 8192 neuronas, la segunda con 512 neuronas y la última con 1, todas ellas con activaciones relu.

Primero se dejan todas las capas del VGG16 fijas y se entrenan solamente las dos últimas añadidas (transfer learning). De esta forma, se supone que la red pre-entrenada ya ha aprendido características que pueden ser útiles en nuestro caso. Como en la primera aproximación, se buscan los mejores hiperparámetros haciendo un grid search del tamaño del batch (4, 8 y 16), la altura de las imágenes (408 y 272), el optimizador (Adam y SGD) y el paso de entrenamiento (0.1, 0.01 y 0.001). Los 5 mejores resultados se muestran en la tabla 4 [3].

El mejor resultado es similar al obtenido con la anterior aproximación y no se consigue reducir el MSE en entrenamiento más. Esto indica que se podría aumentar la complejidad del modelo para intentar obtener un menor MSE en entrenamiento. Por eso, se prueba a entrenar también los parámetros pre-entrenados de la red VGG16 (fine tuning). De esta

[2] Todos los resultados de los modelos sencillos se encuentran en el Excel summ_results_base.xlsm
[3] Todos los resultados para esta segunda aproximación se encuentran en el Excel summ_results_pre_trained.xlsm

Batch size	Ancho (px)	Altura (px)	Learning rate	Optimizador	Tiempo (s)	Train MSE	Val MSE	Test MSE	Test RMSE
4	544	408	0,1	Adam	451	4,93	7,06	5,72	2,39
8	544	408	0,001	Adam	629	4,48	7,82	5,86	2,42
8	544	408	0,1	Adam	549	5,62	8,04	5,96	2,44
4	**544**	**408**	**0,001**	**SGD**	**358**	**5,97**	**7,35**	**6,28**	**2,51**
8	544	408	0,01	Adam	436	5,04	7,05	6,33	2,52

Table 3: Resultados del modelo sencillo con tres bloques convolucionales con número de filtros incremental

Batch size	Ancho (px)	Altura (px)	Learning rate	Optimizador	Tiempo (s)	Train MSE	Val MSE	Test MSE	Test RMSE
16	362	272	0,01	Adam	289	4,05	7,49	4,91	2,22
16	362	272	0,1	Adam	313	3,87	8,01	5,34	2,31
4	**362**	**272**	**0,001**	**Adam**	**188**	**4,27**	**7,33**	**5,39**	**2,32**
8	362	272	0,001	Adam	239	3,84	6,81	5,43	2,33
8	362	272	0,01	Adam	216	4,37	7,57	5,57	2,36

Table 4: Resultados de hacer transfer learning con VGG16

manera, y haciendo un grid search más pequeño al tener mayores cálculos al entrenar una red grande, se obtienen los 5 mejores resultados mostrados en la tabla 5.

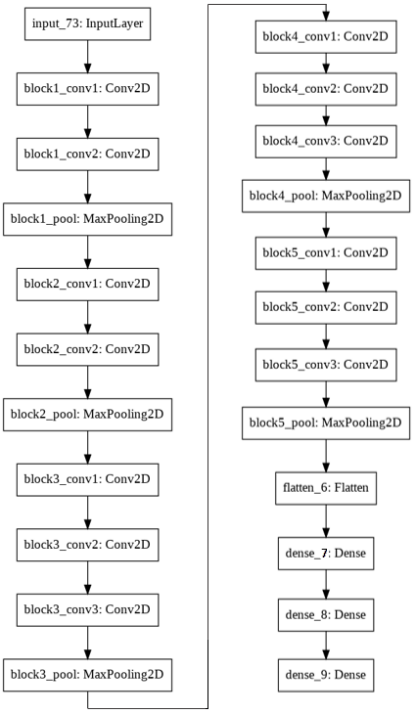

Figure 5: Arquitectura VGG16

Como era de esperar, el error en la muestra de entrenamiento disminuye casi hasta a hacerse nulo mientras que el error en validación se queda a una gran distancia (más de 4 puntos en algunos casos). Esto indica un claro overfitting, que también se puede ver en las curvas de entrenamiento, como en la figura 6, donde se muestran para el modelo con el mejor RMSE en test. Este overfitting se podría corregir con alguna de las técnicas de regularización como dropout o data augmentation. Esta última se probará en la siguiente sección.

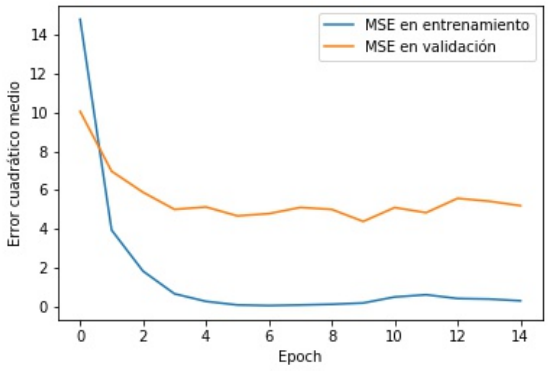

Figure 6: Curva de entrenamiento fine tuning con claro overfitting

Batch size	Ancho (px)	Altura (px)	Learning rate	Optimizador	Tiempo (s)	Train MSE	Val MSE	Test MSE	Test RMSE
4	362	272	0,01	Adam	500	0,22	4,40	3,60	1,90
4	362	272	0,001	SGD	486	0,61	5,20	3,65	1,91
4	362	272	0,001	Adam	268	1,71	4,32	3,68	1,92
4	544	408	0,001	Adam	978	0,43	4,62	3,77	1,94
4	362	272	0,01	SGD	421	1,06	5,13	3,84	1,96

Table 5: Resultados de hacer fine tuning sobre el modelo VGG16

3.3 Modelo sencillo con data augmentation

El número de imágenes es pequeño (718), por lo que hacer data augmentation puede incrementar la bondad del modelo. Esta técnica es de regularización, por lo que tiene sentido aplicarse en modelos que tengan sobreajuste en las secciones anteriores. Por eso, para todos los enfoques, se escogen los modelos que tengan un menor MSE en la muestra de entrenamiento, ya que casi todos presentan overfitting.

Las transformaciones que se han incluido son las siguientes:

1. Rotación de hasta 30º (rotation),
2. simetría horizontal (horizontal flip),
3. simetría vertical (vertical flip),
4. cambio del brillo (brightness),
5. pequeña deformación en un eje (shear) y
6. desplazamiento en el canal de la imagen (channel shift)

Por lo tanto, cada vez que el modelo vea una imagen, verá una transformación de la imagen original. De esta forma, no podrá sobreajustar el modelo a un set de imágenes, ya que las transformaciones son aleatorias. Como ejemplo, en la figura 7, se muestran 8 transformaciones que podrían ocurrir sobre una imagen del dataset original (esquina superior izquierda). Se ve cómo todas las imágenes son distintas. Esto ocasionará que el proceso de entrenamiento sea más lento y con mayor volatilidad. Por esto, el número de epochs que hay que esperar hasta terminar el entrenamiento se incrementa a 7, en lugar de 5 como en los casos anteriores.

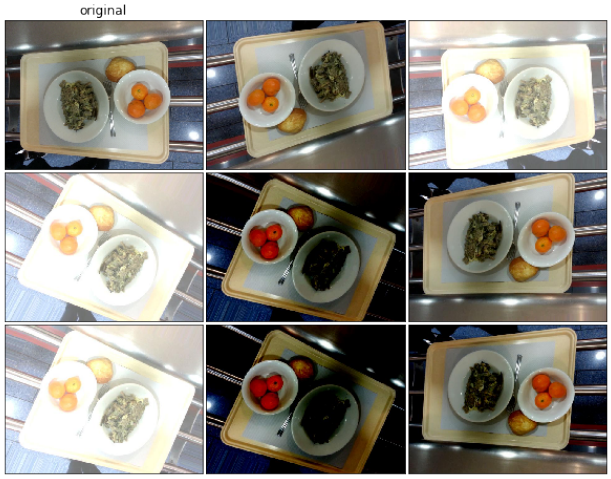

Figure 7: Ejemplos data augmentation

Para las tres arquitecturas del modelo sencillo, se toman los 6 mejores resultados y se aplica data augmentation. Los resultados se muestran en la tabla 6 [4]. Se puede apreciar que los resultados son peores. Este aumento en el RMSE en test se debe a que se han incorporado muchas transformaciones a las imágenes y el modelo sencillo no es lo suficientemente complejo como para aprender todas las características necesarias para predecir correctamente el porcentaje de píxeles con comida.

Como se explicó en la aproximación anterior sobre el fine tuning, dicho modelo sobreajustaba los datos y era necesaria la regularización. Por eso, se ha aplicado también data augmentation al fine tuning del VGG16, obteniendo los resultados mostrados en la tabla 7.

[4] Todos los resultados de la aproximación con data augmentation se encuentran en el Excel summ_results_data_augm.xlsm

Arquitectura	Batch size	Ancho (px)	Altura (px)	Learning rate	Optimizador	Tiempo (s)	Train MSE	Val MSE	Test MSE	Test RMSE
DA_Base	4	544	408	0,01	Adam	1088	12,94	12,51	11,88	3,45
DA_Base	8	544	408	0,01	Adam	1831	12,69	11,69	12,23	3,50
DA_Base	16	544	408	0,1	Adam	1797	13,94	12,44	12,66	3,56
DA_Base	4	544	408	0,001	Adam	699	14,80	13,98	12,74	3,57
DA_Base	8	544	408	0,001	Adam	1991	13,13	10,43	12,96	3,60
DA_Base	8	362	272	0,01	Adam	841	13,93	13,86	13,44	3,67
DA_Extra	4	544	408	0,1	Adam	1397	14,47	12,90	10,68	3,27
DA_Extra	8	362	272	0,001	Adam	844	13,52	13,35	12,42	3,52
DA_Extra	4	544	408	0,001	Adam	943	14,33	14,66	13,15	3,63
DA_Extra	16	544	408	0,01	Adam	827	16,91	16,24	14,14	3,76
DA_Extra	16	544	408	0,1	Adam	1076	16,43	14,32	15,78	3,97
DA_Extra	4	544	408	0,01	Adam	779	16,32	15,37	16,23	4,03
DA_Extra_filt	8	544	408	0,001	Adam	1643	13,71	13,27	11,51	3,39
DA_Extra_filt	4	544	408	0,01	Adam	2187	11,54	10,32	11,57	3,40
DA_Extra_filt	4	544	408	0,1	Adam	977	13,90	13,51	13,37	3,66
DA_Extra_filt	8	544	408	0,01	Adam	1209	13,95	13,36	13,50	3,67
DA_Extra_filt	8	544	408	0,1	Adam	1083	14,91	14,86	13,58	3,69
DA_Extra_filt	4	544	408	0,001	SGD	944	14,03	14,12	16,52	4,07

Table 6: Resultados de usar data augmentation con las arquitecturas sencillas

Batch size	Ancho (px)	Altura (px)	Learning rate	Optimizador	Tiempo (s)	Train MSE	Val MSE	Test MSE	Test RMSE
4	**544**	**408**	**0,01**	**Adam**	**2386**	**3,07**	**2,83**	**3,13**	**1,77**
4	362	272	0,001	Adam	1064	2,53	3,18	3,18	1,78
4	362	272	0,01	Adam	1030	2,46	3,13	3,44	1,86
4	362	272	0,01	SGD	1236	3,97	4,06	3,83	1,96
4	362	272	0,001	SGD	1036	4,35	4,60	4,08	2,02

Table 7: Resultados de usar data augmentation y fine tuning sobre VGG16

Estos resultados son ciertamente los mejores, ya que tienen el menor RMSE en la muestra de test y no presentan overfitting. El mejor modelo tiene un RMSE en test de $1,77$ y su curva de entrenamiento se muestra en la figura 8, donde se aprecia el comportamiento más volátil debido al data augmentation de las dos curvas, pero siempre estando la una cerca de la otra.

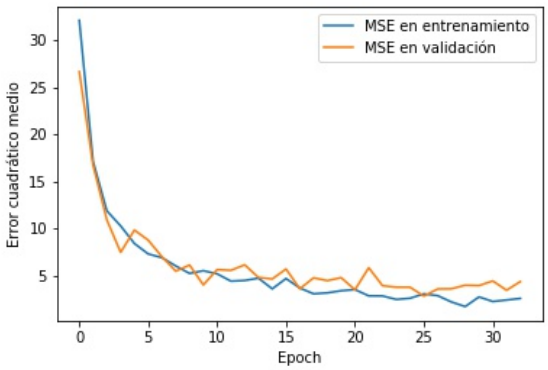

Figure 8: Curva de entrenamiento fine tuning con data augmentation

4 Conclusiones

Todas las aproximaciones mejoran el baseline de RMSE $= 4,8$ en la muestra de test en más de 2 puntos, siendo el modelo VGG16 con fine tuning y data augmentation el que obtiene mejores resultados (RMSE $= 1,77$). Además, fine tuning requiere un mayor tiempo de computación al tener que entrenar todos los parámetros de una red compleja.

Se ha visto que la optimización de los hiperparámetros es muy importante, pudiendo alcanzar mejores resultados escogiéndolos acertadamente.

Data augmentation obtiene mejores resultados si la red es compleja que si es sencilla debido al aumento en la complejidad de las imágenes, y provoca unas curvas de entrenamiento más erráticas.

References

[1] Food recognition: a new dataset, experiments and results (Gianluigi Ciocca, Paolo Napoletano, Raimondo Schettini) In IEEE Journal of Biomedical and Health Informatics, volume 21, number 3, pp. 588-598, IEEE, 2017.

[2] Very Deep Convolutional Networks for Large-Scale Image Recognition. Karen Simonyan, Andrew Zisserman

[3] Google Colaboratory FAQ (link)

[4] Imagenet (link)