Redes de Neuronas

Práctica 2
Problema de Clasificación
Curso 2020-2021

Ramiro Leal Meseguer
100346124
100346124@alumnos.uc3m.es

Claudia Fernández Estebaranz
100366696
100366696@alumnos.uc3m.es

Índice

1. PARTE II **2**
1.1. Introducción . . . 2
1.2. Perceptrón multicapa . . . 2
1.2.1. Experimentos . . . 2
1.2.2. Resultados . . . 3
1.2.3. Análisis de los resultados . . . 3
1.3. Red convolucional . . . 4
1.3.1. Experimentos . . . 5
1.3.2. Resultados . . . 5
1.3.3. Análisis de resultados . . . 5
1.4. Comparación de modelos y conclusiones . . . 7

1. PARTE II

1.1. Introducción

Para la segunda parte de esta práctica vamos a trabajar con imágenes es decir, la entrada de la red van a ser directamente los pixeles. En este caso, la base de datos va a estar formada por 60000 imágenes (compuestas por 3 colores) de 10 clases diferentes, por tanto, contaremos con 6000 imágenes por clase. Si definimos las clases en las que podemos clasificar, contamos con:

0	airplane	5	dog
1	automobile	6	frog
2	bird	7	horse
3	cat	8	ship
4	deer	9	truck

Figura 1: 10 clases de imágenes

De las 60000 imágenes con las que contamos, destinaremos **50000 para entrenar a la red** y **10000 para testearla**, dividiremos los datos de manera equitativa, es decir, por cada clase destinaremos 5000 a entrenamiento y 1000 a test.

Para resolver esta clasificación aplicaremos tanto el perceptrón multicapa como redes convolucionales, para así, posteriormente comparar ambas técnicas. Como sabemos, una arquitectura CNN nos dará mejores resultados, dado que es una técnica más adecuada cuando estamos trabajando en el dominio de la imágenes.

Las CNN son un tipo de Red Neuronal de aprendizaje supervisado que procesa sus capas como si fuera un cortex visual del ojo humano. De esta manera, identifica las distintas características en las entradas que le permite identificar objetos. Las CNN contienen varias capas ocultas especializadas y con una jerarquía, esto quiere decir que las primeras capas pueden detectar lineas, curvas... y estas formas se van especializando según vamos avanzando por las capas más profundas. De esta forma acabamos reconociendo formas más complejas como puede ser una cara, la silueta de un animal o en este caso las 10 clases que le proponemos a las red.

Debido a la complejidad que requiere este programa, trabajaremos a través de *Colaboratory de Google* (https://colab.research.google.com), donde haremos uso de *Keras* que permite el diseño y uso sencillo de CNN y PM.

1.2. Perceptrón multicapa

En esta sección explicaremos los cinco experimentos elegidos, los resultados y las conclusiones para la resolución de este problema con perceptrón multicapa.

1.2.1. Experimentos

De manera general, todos los experimentos en la capa de entrada y salida tendrán la misma estructura:

- Capa de entrada: formada por tantas neuronas como pixeles tiene la imágen. En este caso, tendremos 32 x 32 x 3, que será igual a 3072 neuronas de entrada.
- Cada de salida: Formada por 10 neuronas, correspondientes a las 10 clases en las que se puede clasificar. Estas neuronas utilizarán la función *softmax* (función exponencial normalizada).

Por tanto, en esta siguiente sección solo nombraremos de cada experimento las modificaciones que realizaremos en el número de capas ocultas, en el número de neuronas y en las epochs.

- El **primer experimento (E1)** será el modelo inicial ofrecido por el enunciado. Este tendrá únicamente **una capa oculta** con 50 neuronas y con función de activación *ReLU*.

- El **segundo experimento (E2)** contara con **cinco capas ocultas** repartidas en el siguiente orden (2500, 2000, 1500, 1000, 500), con función de activación *ReLU*.
- El **tercer experimento (E3)** que realizamos intenta imitar la condensación de información que se produce en las redes convolucionales a través de la función *pooling*. En este caso, tendremos **cinco capas ocultas** repartidas en el siguiente orden (1024, 512, 256, 128, 64), con función de activación *ReLU*.
- El **cuarto experimento (E4)** tendremos **cuatro capas ocultas** cada una formada por 2048 neuronas, con función de activación *ReLU*.
- El **quinto experimento (E5)** estará formado por 7 capas ocultas, repartidas en el siguiente orden (64, 128, 256, 512, 256, 128, 64).

1.2.2. Resultados

En la tabla 1 podemos ver los mejores resultados obtenidos para los distintos experimentos. Hemos querido probar la función *DropOut* para eliminar aquellas neuronas con una determinada probabilidad p, para prevenir el sobreaprendizaje, posteriormente analizaremos su impacto. Por otra parte, se puede ver que los distintos experimentos se han probado con distintos *epochs*, donde los resultados varían en función de estos y cada experimento es un caso en particular. Sin embargo, hemos decidido no tocar la tasa de aprendizaje que viene determinada en el modelo inicial, 1e-3, pues preferimos centrar nuestro estudio en modificar el número de capas ocultas y las neuronas que contienen estos.

La totalidad de los experimentos se puede encontrar en un *excel* dentro de la carpeta */Redes convolucionales/PM* que se denomina ***PM-Pruebas***, en el se detallan todos los experimentos realizados.

Resultados mejores cinco experimentos						
Experimento	**TrainLoss**	**TrainAcc**	**TestLoss**	**TestAcc**	**DropOut**	**Epochs**
E1	1,2086	0.5743	1,4169	**0.5069**	NO	12
E1	1,463	0.4812	1,4409	0.5016	SI	12
E1	1,2522	0.5582	1,4292	0.5047	NO	10
E1	1,4943	0.4723	1,4402	0.4918	SI	10
E2	0.8683	0.6944	1,4418	0.5460	NO	10
E2	0.9389	0,6758	1,4283	0.5394	SI	10
E2	0.5616	0.8073	1,7062	**0.5577**	NO	15
E2	0.6548	0.7782	1,6354	0.5494	SI	15
E3	0.8488	0.7006	1,4588	**0.5486**	NO	10
E3	0.9867	0.6565	1,4332	0.5319	SI	10
E3	0.7182	0.7492	1,5585	0.5441	NO	12
E3	0.8922	0.6918	1,4758	0.5465	SI	12
E4	0.5692	0.8036	1,7123	**0.5487**	NO	14
E4	0.5692	0.8036	1,7123	0.5487	SI	14
E4	1,0421	0.6314	1,3672	0.5482	NO	7
E4	0.5692	0.8036	1,7123	0.5487	SI	7
E5	1,1104	0.6059	1,4308	**0.5126**	NO	12
E5	1,3421	0.5321	1,4308	0.4961	SI	12

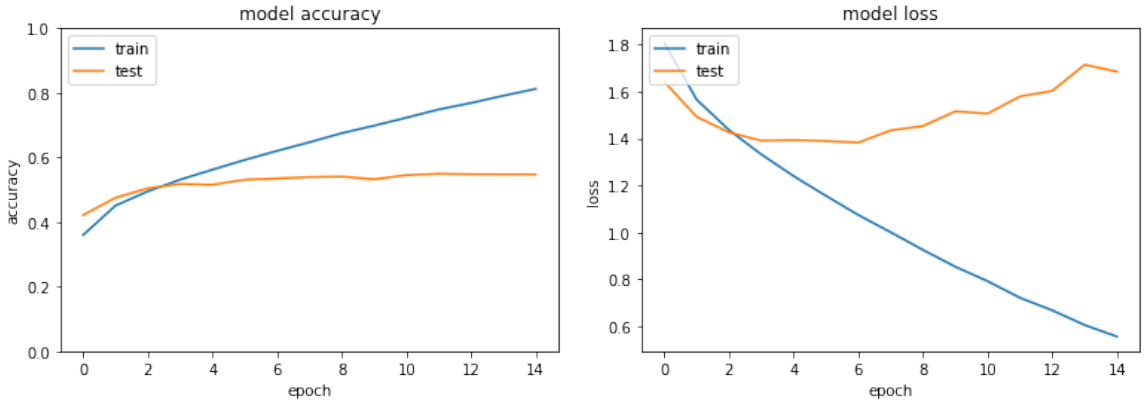
Cuadro 1: Resultados de los experimentos

1.2.3. Análisis de los resultados

Para evaluar los resultados, principalmente, nos hemos basado en el resultado ofrecido por la métrica de Test Accuracy. Como podemos ver, en este caso, las arquitecturas donde incluíamos la

función de *DropOut*, no conseguían mejores resultados. Comentar que los mejores resultados, en su mayoría se obtienen alrededor de las 10 epochs o superior. Podemos ver marcados en negrita en la *tabla 1* cuales son los mejores resultados de cada experimento. Hay que comentar que en general los resultados son bastante bajos, pero dado que todos estos giraban en torno a los mismos valores, llegamos a la conclusión de que no los podíamos mejorar más. El **mejor modelo que encontramos**, es a través del **experimento 2** que tiene casi un 56 % de aciertos en el test.

Finalmente, eligiendo el experimento 2 como mejor modelo, procedemos a ampliar los resultados que se obtienen con este:



(a) Evolución accuracy **(b)** Evolución loss

Figura 2: Evolución de accuracy y loss en entrenamiento y test

```
[[722  21  36  27  32   3  13   6  93  47]
 [ 65 636  11  33  16   9  14  10  73 133]
 [111  12 416  96 171  54  54  45  18  23]
 [ 49  19  74 410 107 147  94  49  16  35]
 [ 58   6 135  69 540  24  67  66  21  14]
 [ 29   3  78 276 107 362  53  71   6  15]
 [ 15  11  74 107 149  40 566  11  12  15]
 [ 53   7  50  64 117  65  21 560  10  53]
 [149  55   9  35  30  11   4   5 661  41]
 [ 73 124  18  50  15   9  13  27  67 604]]
```

Figura 3: Matriz de confusión

1.3. Red convolucional

En esta segunda parte, realizaremos el mismo proceso que en la primera pero esta vez usaremos redes convolucionales. Es por esto que en este caso, el estudio se basará en modificar parámetros como el número de filtros y el tamaño del kernel en las capas de convolución y el número de épocas. Comentar que en este caso, al igual que en la parte anterior, hemos decidido no modificar la razón de aprendizaje que viene por defecto en el enunciado pues así en las comparaciones ambos modelos estarán, más o menos, en igualdad de condiciones. Para este entrenamiento en concreto se solicita que después de cada capa convolucional se introduzca una *capa de pooling con kernel 2x2*, para hacer *subsampling* de la matriz obtenida en la capa anterior. Además, también se hace el estudio con el optimizador *Adam* y se utiliza como función *loss sparse_categorical_crossentropy*. En los siguientes apartados procederemos a mostrar los resultados de los experimentos realizados en estas condiciones y además, añadiremos el estudio realizado, de los mejores experimentos obtenidos, modificando el *optimizador* y añadiendo *dropout*.

1.3.1. Experimentos

Para la realización de las pruebas hemos definido los siguientes 6 experimentos. Comentar que todos ellos tendrán la misma capa de entrada y salida, que será igual que en el perceptrón multicapa. La **capa de entrada** tendrá la estructura (32, 32, 3) que vendrá definida por el número de píxeles que tiene la imagen de entrada, en este caso estamos tratando con imágenes de 32x32 y con 3 colores. Por otro lado, la **capa de salida** vendrá definida por 10 neuronas que serán las 10 clases en las que podemos clasificar las imágenes. Esta función de salida tendrá asociada la función de activación *softmax*(función exponencial normalizada).

- Primer experimento (E1): Este experimento es el que nos proporcionaba el enunciado. Tendrá una capa de convolución con 16 filtros, un kernel de 3x3 y una función de activación ReLU. Después de esta capa, nos encontramos con la capa de MaxPooling de 2x2. Después de esta sección pasaríamos a el bloque neuronal regular, donde tendremos una primera capa encargada de aplanar los resultados obtenidos de la convolución y después una única capa formada por 32 neuronas con función de activación ReLU.
- Segundo experimento (E2): En este caso tendremos dos capas de convolución. La primera formada por 32 filtros, con un kernel de 5x5 y función ReLU. Y la segunda, con 64 filtros, un kernel de 3x3 y la función ReLU. Comentar que después de cada capa de convolución tendremos una capa de MaxPooling. Tras esto, tendremos la misma estructura que en el experimento 1 (aplanamiento y capa de 32 neuronas)
- Tercer experimento (E3): En este experimento hemos querido comprobar cómo afecta añadir una capa más al bloque neuronal regular. Por tanto, este experimento será igual que el E2 pero en el bloque neuronal regular añadiremos una nueva capa, tras la de 32 neuronas, de 64 neuronas.
- Cuatro experimento (E4): Dado que observamos que el experimento 3 no obtenía mejores resultados, volvimos a realizar modificaciones a partir del experimento 2. Es por esto, que tenemos la misma estructura que este pero probamos con un kernel constante, para probar su efecto. Es por esto que ambas capas de convolución tendrán un kernel de 3x3.
- Quinto experimento (E5): En este experimento partimos de la arquitectura E4 y añadimos una capa de convolución más, tras las ya existentes. La capa de convolución tendrá 128 filtros, un kernel de 3x3 y una función ReLU.
- Sexto experimento (E6): Hemos querido hacer un sexto experimento viendo que ocurriría si cogíamos el E5 y quitábamos el kernel constante. Por esto, tendremos entonces 3 capas convolucionales.
  - 1: Filtros: 32, Kernel: 5x5, ReLU
  - 2: Filtros: 64, kernel: 3x3, ReLU
  - 3: Filtros: 128, kernel: 2x2, ReLU

Comentar, que si en algún caso no se ha mencionado, hemos incluido siempre, tras un capa *Con2D* una capa de *MaxPooling*.

1.3.2. Resultados

Todos los experimentos se puede encontrar en un *excel* dentro de la carpeta */Redes convolucionales/CNN* que se denomina ***CNN-Pruebas***, en el se detallan todos los experimentos realizados. En la **tabla 2**, procederemos a mostrar los resultados obtenidos:

1.3.3. Análisis de resultados

Para evaluar los resultados, principalmente, nos hemos basado en el resultado ofrecido por la métrica de Test Accuracy. Si miramos la **tabla 2**, podemos ver que el primer experimento, el ofrecido por el enunciado es el más débil y el que peores resultados nos ofrece, por tanto, lo descartamos. Como

Resultados experimentos					
Experimento	**TrainLoss**	**TrainAcc**	**TestLoss**	**TestAcc**	**Epochs**
E1	0.8386	0.7100	1,0916	**0.6296**	**10**
E1	0.7917	0.7221	1,1275	0.6269	15
E1	0.6722	0.7630	1,2125	0.6223	20
E2	0.4287	0.8476	1,023	0.7037	12
E2	0.4389	0.8466	0.9737	0.7045	10
E2	0.5680	0.8003	0.8892	**0.7164**	**7**
E2	0.8316	0.7054	0.9321	0.6807	5
E3	0.4599	0.8383	0.9570	0.7085	10
E3	0.5682	0.8011	0.8951	**0.7088**	**7**
E3	0.6637	0.7683	0.8796	0.7075	5
E4	0.4243	0.8499	0.9699	0.7150	10
E4	0.5651	0.8024	0.8347	**0.7286**	**7**
E4	0.3802	0.8643	1,0621	0.6959	12
E5	0.2617	0.9075	0.9337	0.7503	10
E5	0.4235	0.8519	0.7946	**0.7506**	**7**
E5	0.2262	0.9196	1,0481	0.7487	12
E6	0.2889	0.8955	0.9636	0.7404	10
E6	0.3507	0.8768	0.9132	0.7382	8
E6	0.2277	0.9188	1,1161	**0.7410**	**12**

Cuadro 2: Resultados de los experimentos

hemos comentado previamente en la explicación de los experimentos, la única diferencia entre E2 y E3 es que E3 cuenta con una capa más en el bloque neuronal regular, esta prueba la quisimos hacer para ver como afectaba. Como podemos ver en los resultados, estos son muy parecidos en ambos, pero si tenemos que elegir nos quedaríamos con la arquitectura E2. Es por esto, que tratamos de mejorar esta, y así lo hicimos. Llegamos a E5, una arquitectura formada por 3 capas convolucionales, todas ellas con el kernel constante. Los filtros de esta arquitectura van aumentando según avanzamos en las capas. Con esta arquitectura obtuvimos los mejores resultados, un 75 % de aciertos en el test. Tratamos de mejorar este modelo, modificando el kernel en las distintas capas, haciendo que este se fuera reduciendo, pero esto no consiguió mejorar el modelo, como se puede observar en los resultados de E6. Por tanto, **diremos que nuestra arquitectura final elegida con los parámetros ajustados del enunciado será E5 ejecutándose 10 épocas.**

Quisimos realizar un estudio (al que denominaremos experimentos extra) donde queremos observar cómo afectaría al modelo final aplicar otro tipo de optimizador, en este caso, el *RMRprop* o añadiendo *DropOut*. Obtuvimos los resultados mostrados en la **tabla 3**:

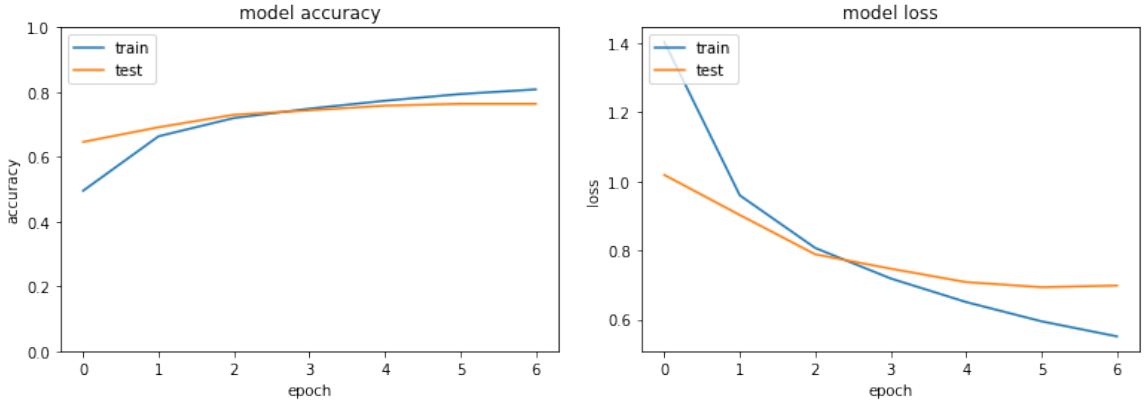
Resultados experimentos extra							
Experimento	**TrainLoss**	**TrainAcc**	**TestLoss**	**TestAcc**	**DropOut**	**Optimizador**	**Epochs**
E5	0.5492	0.8084	0.6921	**0.7644**	**0.25**	**Adam**	**7**
E5	0.4249	0.8533	0.8562	0.7504	0	RMSprop	7
E5	0.5617	0.8070	0.7158	0.7619	0.25	RMSprop	7

Cuadro 3: Resultados de los experimentos extra

Como podemos observar añadiendo *dropout* obtenemos mejoras en nuestro modelo, ya sea individualmente o también con el optimizador RMSprop. Sin embargo, el modelo donde solo se modifica el optimizador nos devuelve una tasa de aciertos menor que el modelo original de la tabla 2. Es por esto, que nos quedaremos como **mejor modelo aquel que añade un dropout de 0.25, con el**

optimizador Adam.

Procedemos a ampliar los resultados del modelo final. Es cierto, que si solo hubiésemos modificado los parámetros que se comentaban en el enunciado nos habríamos quedado con el experimento 5 ejecutado con en 7 épocas. Pero dado que hemos decidido ampliar nuestra búsqueda y además, hemos encontrado un mejor modelo, **procedemos a ampliar los resultados que obtenemos con el E5, epochs = 7, dropout = 0.25, optimizador = Adam, dado que lo hemos considerado como nuestro mejor modelo.**



(a) Evolución accuracy **(b)** Evolución loss

Figura 4: Evolución de accuracy y loss en entrenamiento y test

```
[[780  16  45  27  12   9   9  11  56  35]
 [  9 873   5   9   2   4   4  10  20  64]
 [ 53   4 620  89  67  48  63  35  12   9]
 [ 18   6  53 640  57 114  61  33  11   7]
 [ 16   3  57  46 747  31  34  56   7   3]
 [ 10   5  40 163  45 644  27  56   4   6]
 [  8   1  29  59  33  17 834   8   4   7]
 [ 10   0  32  40  70  28   8 804   3   5]
 [ 49  21  16  17   4   2   6  10 847  28]
 [ 25  77   5  18   8   4   2  16  22 823]]
```

Figura 5: Matriz de confusión

1.4. Comparación de modelos y conclusiones

En esta sección procederemos a realizar una breve comparación de ambos modelos a través de los resultados. Como dijimos al principio de esta memoria, se esperaba que los resultados de CNN fueran mejores que los de PM, y así ha sido. Si comparamos la tasa de aciertos de los modelos finales de ambos, vemos que PM obtiene un 56 % de aciertos en el test y CNN obtiene un 76 %. Podemos ver que hay una clara diferencia entre ambos valores, y que seleccionamos CNN. Podemos comparar la evolución de los aciertos y pérdidas de ambos modelos en las gráficas que hemos mostrado previamente 4 y 2. Para el caso de CNN, podemos ver que el accuracy se mantiene entre 0.6 y 0.8, dentro del rango de unos resultados buenos, mientras que en PM los valores oscilan entre los 0.4 y 0.6. Si observamos las gráficas de pérdidas, en el caso de PM el test nunca disminuye de 1.0 y sin embargo, en CNN no pasa de ese valor.

Entre estos modelos podemos encontrar ciertas diferencias, que marcan estos cambios en los resultados, como por ejemplo, los datos de entrada de cada modelo. En el perceptrón introducimos características de la imagen y en cambio en CNN se introducen los píxeles (los canales). La estructura de los modelos tienen una parte en común que será el bloque neuronal regular, sin embargo, las

convolucionales añadirán estas capas características de estas redes, además de las capas *subsampling*. Podríamos comentar más diferencias, pero en definitiva, las CNN consiguen, a través de todas estas características, representar un mapa para detectar puntos importantes en la imagen, consiguiendo que haya una cierta jerarquía, simulando el comportamiento biológica del ojo humano.

Referencias

[1] Na8. ¿cómo funcionan las convolutional neural networks? https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador 2018.

[2] UC3M. Problema de clasificación con redes convolucionales. https://aulaglobal.uc3m.es/pluginfile.php/4160847/mod_resource/content/1/Practica2-parte2_2020-21.pdf, 2020.