Práctica 3: Redes neuronales convolucionales

Exequiel Alberto Castro Rivero

78308513E

Subgrupo 2

Contenido

[Introducción 3](#_Toc91088364)

[Ejercicio 1 3](#_Toc91088365)

# Introducción

El trabajo de esta práctica está relacionado con un tema de gran actualidad: las redes neuronales. En el caso que nos ocupa, vamos a realizar la implementación de **redes neuronales convolucionales**. Este tipo de redes han demostrado ser capaces de obtener resultados sorprendentes en determinadas áreas de la Visión por Computador.

El objetivo que se sigue con esta práctica es el de que nos familiaricemos con el diseño de redes neuronales convolucionales que aporta la librería Keras. De este modo, la experimentación va a ser un aspecto clave.

# Ejercicio 1

En este ejercicio se nos pide la construcción de un modelo base, *BaseNet*, para la resolución de un subproblema de la base de datos de *CIFAR100*. En concreto, esta base de datos representa un problema de clasificación con 100 clases. Sin embargo, para el caso que nos ocupa, nos quedaremos con los datos pertenecientes a sólo **25** de las 100 clases que forman la base de datos. Esta base de datos es cargada en memoria con la función de ayuda dada por los profesores *cargarImagenes*.

La descripción del modelo de *BaseNet* se muestra en la posterior tabla:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Layer No.** | **Layer Type** | **Kernel size (for conv layers)** | **Input|Output dimension** | **Input|Output channels (for conv layers)** |
| 1 | Conv2D | 5 | 32|28 | 3|6 |
| 2 | Relu | - | 28|28 | - |
| 3 | MaxPooling2D | 2 | 28|14 | - |
| 4 | Conv2D | 5 | 14|10 | 6|16 |
| 5 | Relu | - | 10|10 | - |
| 6 | MaxPooling2D | 2 | 10|5 | - |
| 7 | Linear | - | 400|50 | - |
| 8 | Relu | - | 50|50 | - |
| 9 | Linear | - | 50|25 | - |

Tabla : Descripción de BaseNet

La función que define dicha red convolucional es *get\_basenet*. Dicha función recibe los siguientes parámetros:

* *input\_shape*: El tamaño de las imágenes de entrada a la red (*[32, 32, 3]* en el caso de las imágenes de nuestro problema).
* *num\_classes*: El número de clases que se desea clasificar por la red neuronal, *25* en el caso de nuestro problema.

La definición de la red se ha hecho de forma secuencial, ya que esta forma es la que me parece más intuitiva y directa. Esta forma de definir redes tiene ciertas limitaciones, pero debido a que en esta práctica no se van a diseñar modelos muy complejos, me parece la más apropiada. A continuación se explica con mayor detalle cómo se ha realizado la definición del modelo con Keras:

* En primer lugar se especifica que el modelo se va a realizar de forma secuencial con *model = Sequential()*.
* Se añade una capa convolucional con activación *Relu* para implementar las capas número 1 y 2 de la tabla. Esto se realiza de la siguiente manera: *model.add(Conv2D(6, kernel\_size=(5, 5), activation='relu', input\_shape=input\_shape))*. Esta capa de convolución tiene 6 filtros debido a que así está especificado en los canales de salida de la tabla. Del mismo modo se obtienen los tamaños del kernel (*kernel\_size*). El parámetro *input\_shape* es necesario debido a que esta es la capa inicial del modelo y se necesitan saber las dimensiones de la entrada. El parámetro *activation* se usa para que la activación tras esta capa sea implementada con la función *Relu*. Todos los demás parámetros son los establecidos por defecto por Keras, bien porque los parámetros por defectos son los correctos (es el caso del parámetro *padding*, el cual tiene que estar establecido a *valid* para que se produzca una reducción en anchura y en altura de los bloques convolucionales) o bien porque no tienen un gran impacto en el rendimiento (se han realizado pruebas con distintos parámetros y se han obtenido resultados similares).
* Se añade una capa *MaxPooling* para reducir las dimensiones del bloque convolucional (el número de canales no se reduce). Esto se realiza de la siguiente manera: *model.add(MaxPooling2D(pool\_size=(2, 2)))*. Para que la reducción de dimensiones se haga a la mitad (tal y como se especifica en la tabla del modelo) el *pool\_size* debe tener valor 2. Los demás parámetros son los establecidos por defecto por Keras, ya que estos son los parámetros correctos (por ejemplo, *padding* establecido a *valid* para que se produzca la reducción a la mitad).
* Se añade otra capa convolucional del mismo modo que anteriormente: *model.add(Conv2D(16, kernel\_size=(5, 5), activation='relu'))*.
* Se añade otra capa *MaxPooling* del mismo modo que anteriormente: *model.add(MaxPooling2D(pool\_size=(2, 2)))*.
* Se añade una capa *Flatten* de modo que todos los píxeles del último bloque convolucional se disponen en un único array 1D. Una forma intuitiva de ver esta capa es que ya se ha decidido que la extracción de características de las imágenes ha llegado a su fin y, por lo tanto, todos los píxeles del último bloque convolucional son características que van a ser la entrada del clasificador de la red.
* Se añade una capa Densa para implementar las capas número 7 y 8 de la tabla. Esto se realiza de la siguiente manera: *model.add(Dense(50, activation='relu'))*. El número de neuronas se establece a 50 ya que así está especificado en la tabla. Lo mismo pasa con la función de activación. Los demás parámetros son los establecidos por defecto por Keras porque no tienen un gran impacto en el rendimiento (se han realizado pruebas con distintos parámetros y se han obtenido resultados similares).
* Se añade la capa totalmente conectada final, la cual se corresponde con la clasificación propiamente dicha. Esto se realiza de la siguiente forma: *model.add(Dense(num\_classes, activation='softmax'))*. El número de neuronas de esta capa totalmente conectada se debe de corresponder con el número de clases del problema que se está tratando. La función de activación usada es *softmax* debido a que el problema que se está resolviendo es un problema de clasificación multiclase: cada neurona “se encarga” de clasificar una clase y la salida de cada neurona es la probabilidad de que la imagen de entrada pertenezca a dicha clase.

Una aproximación intuitiva de lo que hace esta red es la siguiente: mediante el bloque formado por *Conv2D, Relu y MaxPooling* lo que se está realizando son transformaciones de la información de los bloques convolucionales anteriores (*Conv2D*) para luego realizar una selección de las características más relevantes (aquellas con mayor respuesta) con *Relu* y *MaxPooling*. Esta selección es importante, ya que si sólo se realizaran transformaciones de la información inicial el aprendizaje sería imposible. En la selección de aquellas características más relevantes es donde está la potencia de este tipo de aprendizaje (esto es similar a la forma de clasificar imágenes de los humanos: en una fotografía no todo es relevante a la hora de clasificarla). Una vez ya se ha completado la extracción de características (transformación + selección) dichas características son pasadas como entrada a un modelo de aprendizaje automático supervisado (una red neuronal en el caso de este modelo) con el fin de obtener un modelo de clasificación.

Como apunte adicional, si se desea ver con mayor profundidad el modelo definido o comprobar la correctitud del mismo se puede visualizar el resumen ofrecido por Keras (con la función *summary*) que se encuentra disponible en la ejecución del código.

Una vez terminada la etapa de la definición del modelo