

CAPÍTULO 3

Deep learning

En este capítulo se profundiza en los conceptos teóricos detrás de las redes neuronales profundas, los cuales son necesarios para entender su funcionamiento. En primer lugar se describen los elementos fundamentales que componen cualquier tipo de red neuronal artificial como son las neuronas y su entrenamiento. Por último, se hace especial énfasis en analizar las características del modelo de aprendizaje que se propone en este estudio, las redes neuronales convolucionales.

3.1– Conceptos básicos de redes neuronales

Una red neuronal artificial (ANN) es un modelo computacional inspirado en la forma de procesar la información de la red neuronal biológica de un ser humano [64].

Existen tres elementos fundamentales que definen a una red neuronal:

- **Modelo de neuronas:** la unidad de procesamiento de información.
- **Arquitectura:** conjunto de neuronas y enlaces ponderados que las conectan.
- **Algoritmo de aprendizaje:** un método de entrenamiento de la red ajustando los pesos en los enlaces entre neuronas.

3.1.1. Unidad de procesamiento: neurona

La unidad básica de cálculo de una red neuronal es la neurona, a la que también se conoce como nodo o unidad. Estas neuronas reciben entradas de otros nodos y calculan una salida. Cada entrada tiene asociado un peso (w), que se asigna según la importancia relativa de ésta con otras entradas. Como se puede apreciar en la Figura 3.1, el nodo aplica una función a la suma ponderada de las entradas para obtener un valor de salida. Además de los valores de entrada, se introduce un valor adicional artificial conocido como bias. Este sesgo se añade para que el valor de salida no dependa únicamente de la entrada. De esta forma podemos incrementar la flexibilidad del modelo a la hora de ajustarse a los datos sobre los que entrena, sobre todo cuando todas las entradas tienen valor cero.

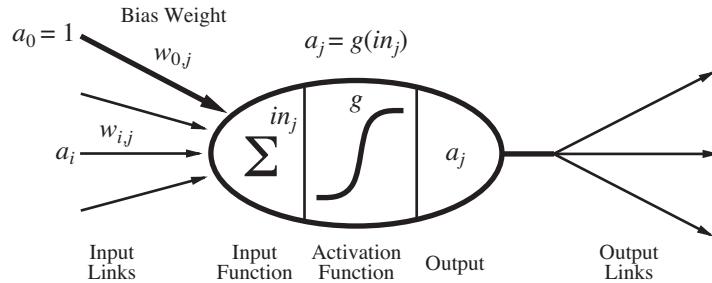


Figura 3.1: Modelo matemático simple en una neurona

La función g es una función no lineal que se conoce como función de activación. El motivo de aplicar este operador es introducir no-linealidad en la salida de la neurona. Este hecho es importante dado que la mayoría de los datos de la vida real serán no lineales. Existen distintas funciones de activación aplicables a las neuronas, algunas de ellas se muestran en la Figura 3.2.

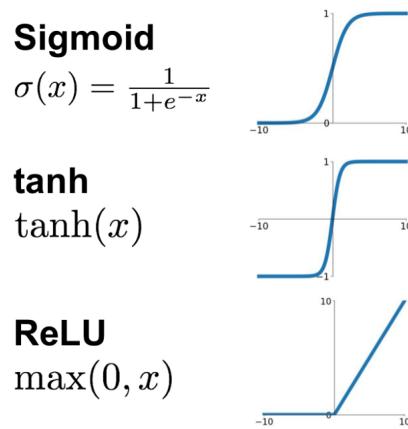


Figura 3.2: Distintas funciones de activación en neuronas

3.1.2. Arquitectura de una red neuronal

Existen distintos tipos de arquitectura para las ANN, pero en este apartado estudiaremos las conocidas como *feed-forward networks*, las cuales son la base de las redes neuronales convolucionales. En estas redes las conexiones solo se producen una dirección, hacia delante, formando un grafo dirigido acíclico (a diferencia de las redes neuronales recurrentes en las que se forman ciclos en las conexiones). Como podemos observar en la Figura 3.3, las neuronas están distribuidas en capas. Los nodos de capas adyacentes están conectados, y estas conexiones tienen pesos asociados. Podemos distinguir entre tres tipos de nodos: [65]

- **Input nodes:** Los nodos de entrada alimentan a la red con información del exterior. En estos nodos no se realiza ningún cálculo, su función es meramente la de transmitir información a la siguiente capa.
- **Hidden nodes:** Los nodos ocultos no tienen conexión con el exterior. Su función es la de transmitir información desde los nodos de entrada a los de salida. Una red solo tendrá una capa de entrada y una de salida, pero puede tener cero o múltiples capas ocultas.

- **Output nodes:** Los nodos de salida son los responsables de transferir información al exterior.

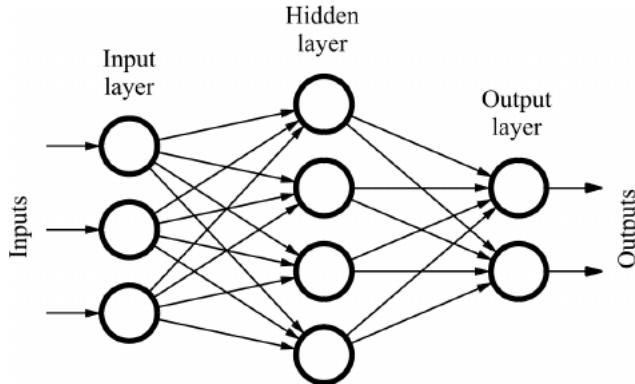


Figura 3.3: Ejemplo de red neuronal MLP con una capa oculta

En la arquitectura se puede apreciar como cada nodo recibe entradas de nodos en capas previas y transmite las salidas a la siguiente capa. Según el número de capas ocultas existen dos tipos de redes feed-forward:

- **Single Layer Perceptron:** ninguna capa oculta.
- **Multi Layer Perceptron (MLP):** cuentan con una o más capas ocultas.

3.1.2.1. Limitaciones del perceptrón simple

Según la naturaleza de los datos, éstos pueden ser linealmente separables o no. Serán linealmente separables si existe un hiperplano que sea capa de dividir el espacio de forma que los elementos que pertenezcan a cada clase queden separados por éste.

El perceptrón simple solo garantiza convergencia cuando los datos son linealmente separables. Es decir, si la naturaleza de los datos es no lineal, no será posible encontrar una configuración de un perceptrón que devuelva la salida esperada para todos los elementos que deseamos clasificar.

En la Figura 3.4 se muestran distintas funciones en las que se etiquetan los datos en torno a dos clases: negro o blanco. Podemos observar como para las funciones *AND* y *OR*, el perceptrón simple es capaz de encontrar una recta que separa los puntos de clases distintas. Sin embargo, con *XOR* tenemos un caso no linealmente separable, puesto que no podemos encontrar ninguna recta que separe los elementos de las dos clases. Este ejemplo en dos dimensiones usando rectas es extrapolable a mayores dimensiones usando hiperplanos.

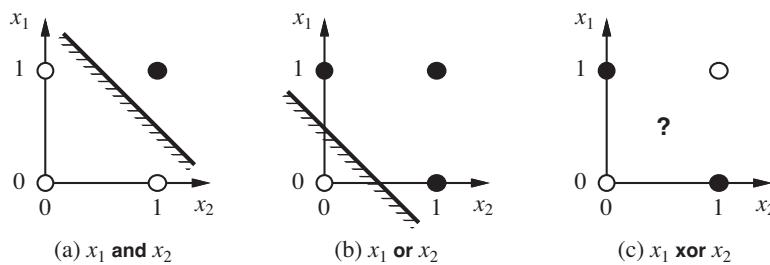


Figura 3.4: Problema de las clases no linealmente separables en el perceptrón simple

Debido a esta limitación, procedemos a estudiar las características de las MLP, ya que son más útiles en las aplicaciones de hoy en día, y para el problema de este trabajo en concreto. La mayoría de los datos en la vida real son no lineales, y las MLP solventan este problema de clasificación para datos no linealmente separables usando capas ocultas. Estas capas adicionales pueden ser interpretadas geométricamente como un intento de añadir hiperplanos adicionales, lo que mejora la capacidad de la red para separar los datos.

3.1.2.2. Perceptrón multicapa (MLP)

El funcionamiento de las redes multicapa es aparentemente simple: reciben un vector de entrada y lo transforman tras realizar operaciones en una serie de capas ocultas. Cada una de estas capas ocultas está compuesta de un conjunto de neuronas, y cada neurona está conectada a todas las neuronas de la capa anterior, lo que se conoce como *fully connected*. Sin embargo, las neuronas de una misma capa funcionan independientemente y no comparten ninguna conexión.

En la Figura 3.5 se muestra un ejemplo del cálculo de salida de un nodo oculto, cuyo valor se transmite a la capa final.

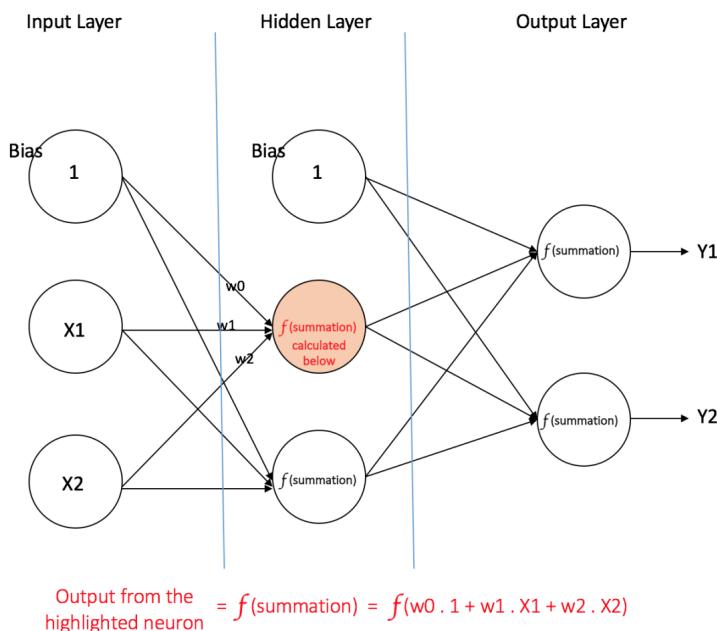


Figura 3.5: Operaciones en los nodos ocultos de una red MLP

La última capa totalmente conectada es la capa de salida (*output layer*) y en tareas de clasificación representa la “puntuación” de cada clase. Para calcular esta puntuación o probabilidad de pertenecer a una determinada clase, se usa generalmente la función de activación *softmax* en las neuronas de la capa de salida. Esta función toma un vector de valores reales arbitrarios y los transforma en un vector de valores entre 0 y 1 que suman uno. Por tanto, tras la operación, la suma de las probabilidades es 1 y podemos comparar las puntuaciones asociadas a cada clase. Finalmente, el nodo con mayor probabilidad es el que proporciona la etiqueta para la clasificación.

Matemáticamente, la función *softmax* se expresa tal y como muestra la Ecuación 3.1, donde p es la distribución de probabilidad, i simboliza una determinada clase, y z es el vector de C dimensiones que contiene los valores reales de la capa de salida de la red, siendo C el número de clases:

$$p_i(z) = \frac{e^{z_i}}{\sum_{j=1}^C e^{z_j}} \quad (3.1)$$

En resumen, la arquitectura presentada en este apartado nos muestra cómo, dado un conjunto de características $X = (x_1, x_2, \dots, x_n)$, y un valor de salida esperado y , una red MLP puede aprender las relaciones entre las características y el resultado esperado, bien para tareas de clasificación o de regresión.

En el siguiente apartado estudiaremos el mecanismo que utiliza la red para aprender estas relaciones a partir de un conjunto de ejemplos de entrenamiento, lo que se conoce como *training set*.

3.1.3. Entrenamiento de una red neuronal

Entrenar una red neuronal implica buscar los pesos adecuados en las conexiones para obtener un comportamiento deseado, dado un conjunto de entrenamiento. El proceso más común con el que se entrena a una red neuronal multicapa se conoce como algoritmo de retro propagación (*Backpropagation algorithm*).

Se trata de un esquema de aprendizaje supervisado, lo que significa que aprende con datos de entrenamiento etiquetados. Para cada entrada contamos con un valor deseado/esperado de salida (en inglés se utiliza el término *label*). Por tanto, existe un “supervisor” que va guiando el aprendizaje para mejorar los resultados. En pocas palabras, podríamos decir que este algoritmo se dedica a aprender de los errores. El supervisor rectifica a la red cada vez que ésta se equivoca en su predicción.

Es importante tener en mente que el objetivo del aprendizaje es modificar los pesos en las conexiones entre neuronas para que la predicción sea correcta. Inicialmente, los pesos son asignados aleatoriamente. Para cada entrada del conjunto de entrenamiento, la ANN se activa y se observa el valor de salida. Este valor se compara con el valor deseado que ya conocíamos, y el error se propaga hacia atrás a la capa anterior. Teniendo en cuenta este error, los pesos son ajustados en consecuencia. Este proceso se repite iterativamente hasta que el error sea menor a un determinado umbral. Una vez concluya el algoritmo, tendríamos una red entrenada a la que podríamos introducir nuevas entradas y obtener mejores resultados en las predicciones.

A continuación se muestra un esquema general del algoritmo de retro propagación que resume las ideas explicadas en el párrafo anterior.

Algoritmo 1: Esquema general del algoritmo de retro propagación

```

1 Inicializar pesos de la red aleatoriamente
2 mientras no se cumpla la condición de parada hacer
    para cada ejemplo  $(x, y)$  en el conjunto de entrenamiento hacer
        1) Calcula la salida  $a_i$  de cada neurona  $i$ , propagando los valores hacia
           delante
        2) Calcular los errores  $\Delta_i$  de cada unidad  $i$ 
        3) Actualizar los pesos  $w_{ij}$  propagando los valores hacia atrás
3 devolver red

```

Tras explicar los conceptos intuitivos de cómo una red puede aprender sobre el conjuntos de entrenamiento, procedemos a explicar paso a paso el funcionamiento del algoritmo de retro propagación desde un punto de vista más matemático.

3.1.3.1. Descenso por gradiente

La retro propagación es básicamente una herramienta para poder aplicar el algoritmo de optimización conocido como descenso por gradiente a una red neuronal multicapa. La propagación de errores hacia atrás nos permite calcular el gradiente de la función de error que definamos, y así poder actualizar los pesos para encontrar el mínimo de ésta función de coste.

La no-linealidad de los datos implica que los parámetros no actúan de forma independiente al influenciar la forma de la función de error. Al hacer las redes neuronales más flexibles introduciendo capas ocultas, no encontramos un método analítico para encontrar el mínimo en un paso. Por tanto, nos vemos obligados a un usar algún método numérico de forma iterativa para alcanzar la solución. Para este caso, el descenso por gradiente es el más popular y efectivo.

En éste método de optimización el objetivo es encontrar la dirección de máximo descenso de la función de coste, y realizar un movimiento pequeño en esa dirección. Es decir, se pretende determinar los cambios a realizar en los parámetros para que la función descienda lo máximo posible, acercándose al mínimo. Este proceso se repite hasta que estemos satisfechos con el mínimo encontrado, que puede ser bien local o global.

Para poder calcular la dirección de máximo descenso, es necesario obtener los gradientes de la función de coste con respecto a todos los parámetros. Un gradiente es una generalización multidimensional de la derivada; es un vector que contiene las derivadas parciales respecto a cada variable. Por tanto, con estas derivadas, obtendríamos la pendiente de la función en cada dimensión, la cual apunta siempre al mínimo más cercano.

En la Figura 3.6 podemos observar un ejemplo de aplicar el descenso por gradiente a una función de coste definida por dos parámetros.

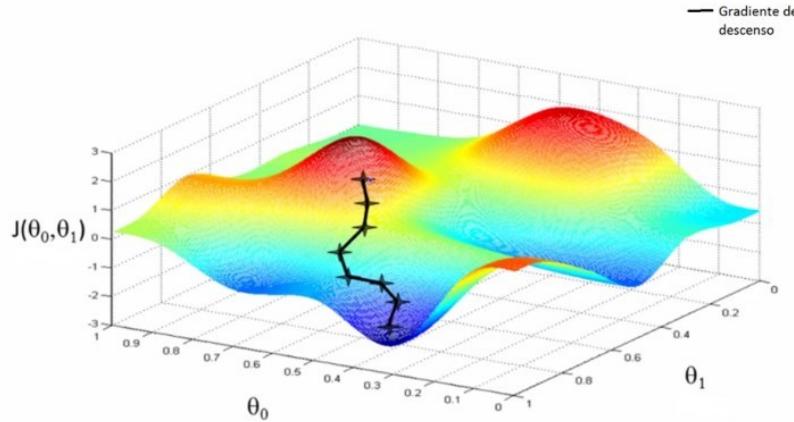


Figura 3.6: Descenso por gradiente

Una vez encontrada la dirección en la que debemos realizar el movimiento, también debemos saber cómo de grande debe ser el movimiento. Este elemento se denomina la tasa de aprendizaje. Su valor se convierte en una decisión importante para el entrenamiento y se trata como un hiper-parámetro que se establece manualmente.

Una tasa de aprendizaje alta indica que se dan pasos más grandes en las actualizaciones de pesos, por lo que es posible que el modelo tarde menos tiempo en converger a la solución óptima. Sin embargo, si la tasa de aprendizaje es demasiado alta, podría resultar en “saltos” demasiado grandes y de poca precisión que impedirían alcanzar el valor óptimo, como se puede observar en la Figura 3.7.

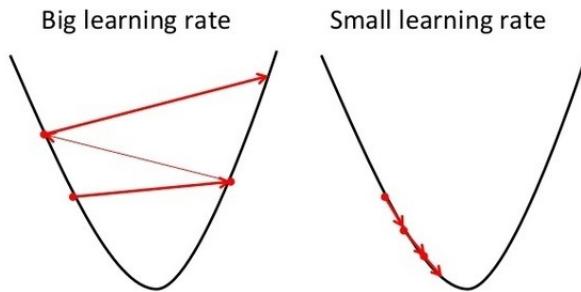


Figura 3.7: Descenso por gradiente aplicando diferentes tasas de aprendizaje

Existen diferentes tipos de descenso por gradientes dependiendo de cómo usamos los datos de entrenamiento para calcular las derivadas: [66]

- **Batch Gradient Descent:** Se usa todo el conjunto de entrenamiento para calcular el gradiente de la función de coste. Se trata de la opción más costosa y a menudo inviable, porque para una sola actualización se utilizan todos los datos, lo cual provoca que el tiempo para converger sea mayor.
- **Stochastic Gradient Descent (SGD):** Se usa un único elemento del conjunto de entrenamiento para calcular el gradiente. Aunque se pierda precisión en la convergencia, se gana rapidez en el cálculo. En este método es importante mezclar el conjunto tras cada iteración para obtener mejores resultados.

- **Mini-Batch Gradient Descent:** Se usa un conjunto de n elementos para calcular el gradiente. Es la mejor opción para conjuntos de entrenamiento grandes ya que combina las ventajas de cada uno de los dos métodos anteriores: el cálculo del gradiente es más estable y preciso, a la vez que reducimos el tiempo de cálculo.

La existencia de mínimos locales en la función de error dificulta considerablemente el entrenamiento, pues una vez alcanzado un mínimo el entrenamiento se detiene aunque no se haya alcanzado la tasa de convergencia fijada, como se puede apreciar en la Figura 3.8.

Cuando caemos en un mínimo local sin satisfacer el porcentaje de error permitido se puede considerar: cambiar la topología de la red (número de capas y número de neuronas), comenzar el entrenamiento con unos pesos iniciales diferentes, modificar los parámetros de aprendizaje, modificar el conjunto de entrenamiento o presentar los patrones en otro orden.

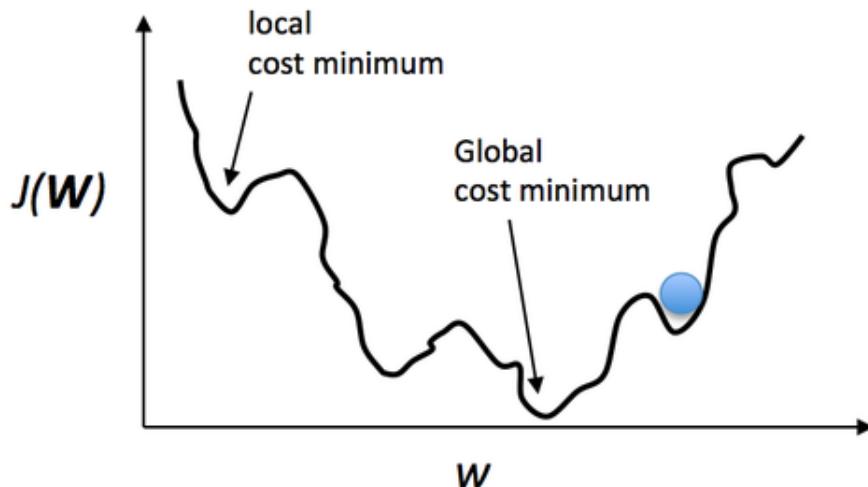


Figura 3.8: Mínimos locales y globales en una función de coste

Todos estos conceptos explicados anteriormente son la base matemática que subyace bajo el algoritmo de retro propagación, el cual procedemos a analizar con mayor detalle.

3.1.3.2. Algoritmo de retro propagación

El análisis del algoritmo de retro propagación se puede dividir en cuatro fases principales [67]:

- **Forward propagation:** En este primer paso, ilustrado en la Figura 3.9, introducimos la entrada en la red y calculamos los valores de salida, es decir, la probabilidad de pertenecer a cada clase. En los primeros ejemplos de entrenamiento, dado que la inicialización de los pesos es completamente aleatoria, la red no será capaz de alcanzar una conclusión razonable sobre cuál debe ser la clasificación. Por tanto, debemos definir una forma de medir el error que hemos cometido.

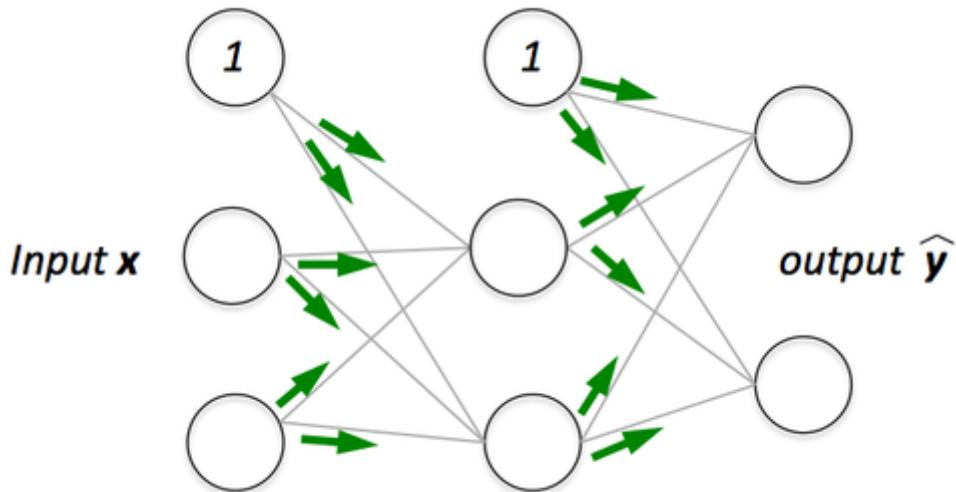


Figura 3.9: Propagación hacia delante

- **Loss function** (función de coste o pérdida): Recordamos que para cada ejemplo de entrenamiento contamos con una etiqueta, una predicción deseada. Por tanto, definiremos nuestro error en base a cuánto nos hemos alejado de la predicción correcta. La función de coste se puede definir de diversas maneras. Una de las más comunes es el error cuadrático medio (MSE: Mean Squared Error), que se expresa: $E_{total} = \sum \frac{1}{2}(target - output)^2$, siendo *target* el valor esperado y *output* el valor obtenido.

Una vez definida la función de coste, nos encontramos con un problema de optimización, cuyo objetivo es minimizar el error cometido. Debemos encontrar cuáles son los pesos que más contribuyen al error, y modificarlos para reducir el coste al máximo posible. Una de las formas de resolver este problema es utilizando el descenso por gradiente, y para ello debemos calcular el gradiente de todos los parámetros.

- **Back propagation:** Propagamos el error (el “coste” calculado al comparar la salida obtenida con la salida objetivo esperada) hacia atrás calculando el gradiente, $\frac{dL}{dW}$, tal y como se muestra en la Figura 3.10.

Comenzamos calculando el error en la capa de salida y obtenemos el gradiente en los nodos de capa oculta, que es el resultado de multiplicar la salida de los nodos ocultos por el error en la capa de salida.

A continuación, calculamos el error en los nodos de la capa oculta propagando hacia atrás los errores calculados anteriormente en la capa de salida. La idea es que cada nodo oculto sea “responsable” de una fracción de error por cada nodo al que se conecta. Una vez obtenidos estos valores, podemos calcular el gradiente en la capa de entrada.

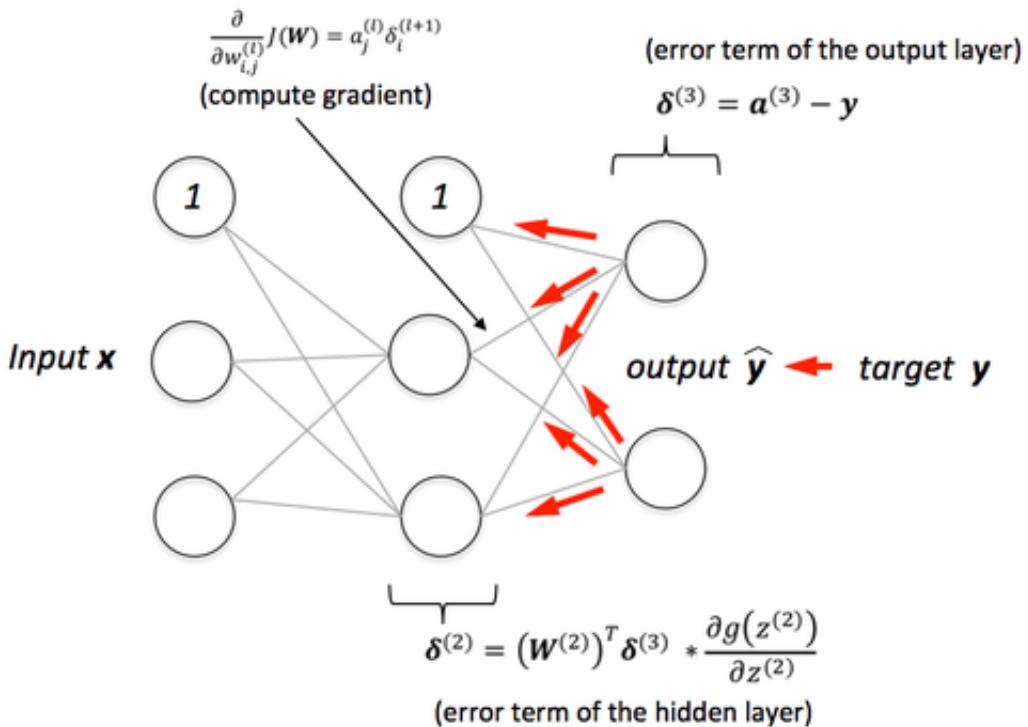


Figura 3.10: Propagación hacia atrás

- **Actualización de pesos:** Calculados todos los gradientes, el último paso es actualizar los pesos en las conexiones de forma que cambien en la dirección del gradiente controlando el movimiento con el factor de aprendizaje $(w \leftarrow w + \eta \frac{dL}{dW})$.

Los cuatro operaciones explicadas anteriormente constituyen una iteración de entrenamiento. Comúnmente, con el fin de obtener mejores resultados, el conjunto de datos de entrenamiento no se introduce una única vez en el modelo. El programa repetirá este proceso un número fijo de iteraciones (número de *epochs*) para cada subconjunto de datos de entrenamiento, lo que se conoce como *batch*. El número de epochs y tamaño de los *batches* para entrenar la red será objeto de discusión posteriormente.

A continuación, se resumen todas las operaciones en la definición formal del algoritmo.

Algoritmo 2: Algoritmo de retro propagación

entrada: conjunto de entrenamiento D con ejemplos en forma (x, y) , la estructura de la red, un factor de aprendizaje η , y una función de activación g

```

para cada peso  $w_{i,j}$  en la red hacer
|    $w_{i,j} \leftarrow$  número pequeño aleatorio

repetir
|   para cada ejemplo  $(x, y) \in D$  hacer
|   |   /* Propagar las entradas hacia delante calculando el valor
|   |   |   de salida */ 
|   |   para cada nodo  $i$  en la capa de entrada hacer
|   |   |    $a_i \leftarrow x_i$ 

|   |   para  $l = 2$  a  $L$  hacer
|   |   |   para cada nodo  $j$  en capa  $l$  hacer
|   |   |   |    $in_j \leftarrow \sum_i w_{i,j} a_i$ 
|   |   |   |    $a_j \leftarrow g(in_j)$ 

|   |   /* Propagar errores (deltas) hacia atrás desde la capa de
|   |   |   salida hasta la de entrada */ 
|   |   para cada nodo  $j$  en la capa salida hacer
|   |   |    $\Delta[j] \leftarrow g'(in_i) \times (y_j - a_j)$ 
|   |   para  $l = L - 1$  a  $1$  hacer
|   |   |   para cada nodo  $i$  en capa  $l$  hacer
|   |   |   |    $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \Delta[j]$ 

|   |   /* Actualizar los pesos usando los deltas */
|   |   para cada peso  $w_{i,j}$  en la red hacer
|   |   |    $w_{i,j} \leftarrow w_{i,j} + \eta \times a_i \times \Delta[j]$ 

hasta que se cumpla el criterio de parada
devolver red

```

3.2– Redes neuronales convolucionales

Las redes neuronales convolucionales forman parte de un conjunto de algoritmos de aprendizaje automático, conocidos como técnicas de aprendizaje profundo o *deep learning*, que intentan modelar abstracciones de alto nivel en datos usando arquitecturas compuestas de transformaciones no lineales múltiples [68]. La multiplicidad en las capas permite una mayor capacidad para la separación de los datos no lineales en diferentes clases, lo cual es necesario para nuestro problema. Existen diferentes tipos de redes neuronales profundas: convolucionales, recurrentes, auto-codificadores, etc. En esta sección se explican las características de las redes neuronales convolucionales (conocidas como *CNNs*), que ha sido el modelo elegido para abordar el problema de clasificación de usos y cobertura del suelo. Estas redes han demostrado ser muy eficaces en áreas como el reconocimiento de imágenes y la clasificación. Son capaces de realizar con éxito la identificación de rostros, objetos, señales de tráfico, etc. Además resultan muy útiles para potenciar la visión en robots y coches autónomos. En la Figura 3.11 podemos observar un ejemplo de segmentación con CNNs, en el que la red es capaz de asociar etiquetas a los distintos objetos que se encuentran en la imagen [69].

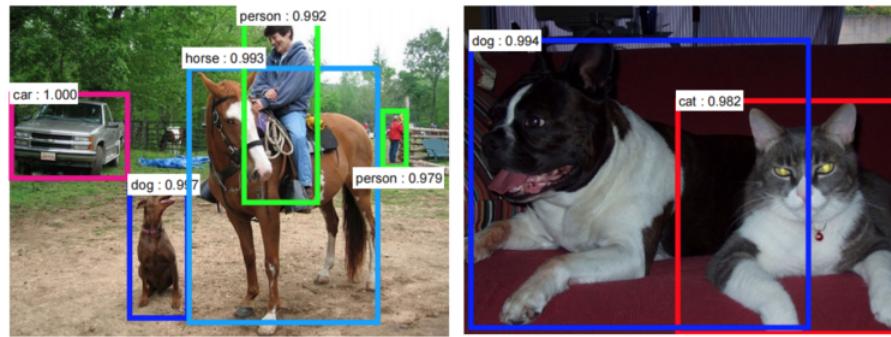


Figura 3.11: Segmentación de imágenes con redes neuronales convolucionales

Las redes neuronales convolucionales son muy similares a las redes neuronales ordinarias como el perceptrón multicapa. Se componen de neuronas que tienen pesos y sesgos que se aprenden tras un proceso de entrenamiento. Cada neurona recibe algunas entradas, realiza un producto escalar y luego aplica una función de activación. Al igual que en el perceptrón multicapa, también se tiene una función de pérdida o coste (*loss function*) sobre la última capa, la cual estará totalmente conectada (*dense or fully connected layer*).

El motivo de usar las redes convolucionales reside en que las redes neuronales ordinarias no escalan bien para imágenes de gran tamaño. Por ejemplo, para una imagen RGB de un tamaño solamente de $32 \times 32 \times 3$ (32 píxeles de ancho, 32 píxeles de altura, 3 canales de color), una única neurona de una capa totalmente conectada tendría $32 \times 32 \times 3 = 3072$ pesos. Aunque este tamaño parezca todavía manejable, se empieza a percibir que esta estructura no escala bien si la imagen aumenta. Por ejemplo, una imagen de tamaño $200 \times 200 \times 3$, una neurona tendría $200 \times 200 \times 3 = 120,000$ pesos. Por lo tanto, observamos que el uso de capas totalmente conectadas supone un alto coste en recursos que además nos puede conducir al problema del sobreajuste al tener tanto parámetros, es decir, el modelo se ajusta tanto a los datos de entrenamiento que es incapaz de generalizar y predecir correctamente.

datos nuevos. Además, en nuestro problema en particular, trabajaremos con imágenes aéreas de 27 (radar) o 200 (hiper-espectrales) bandas, por lo que sería totalmente inviable modelar nuestra solución de esta forma. Por tanto, es conveniente estudiar las características de las redes neuronales convolucionales, las cuales son más adecuadas para intentar abordar este problema de clasificación.

3.2.1. Características generales

La característica fundamental que diferencia a las redes neuronales convolucionales es que se asume que las entradas son datos con cierta estructura espacial como las imágenes, lo que nos permite codificar ciertas propiedades en la arquitectura; permitiendo ganar en eficiencia y reducir la cantidad de parámetros en la red. Estas redes trabajan modelando de forma consecutiva pequeñas piezas de información, y luego combinando esta información en las capas más profundas de la red [70].

Una imagen puede ser representada como una matriz con distintos valores para cada píxel. El concepto de canal o *channel* se refiere a las componentes de una imagen. Una imagen de una cámara digital estándar tendría tres canales: rojo, verde y azul. Podemos imaginar esto como un conjunto de tres matrices apiladas, una para cada color, y cada una con valores para los píxeles entre 0 y 255. El número de canales de una imagen lo asociaremos al concepto de profundidad, la tercera dimensión de la imagen de entrada o *input*. Por tanto, es natural que las neuronas de una capa de una red convolucional, a diferencia de las ordinarias, están organizadas en tres dimensiones: altura, anchura y profundidad. De esta forma, la entrada pasa de ser un vector a un volumen en tres dimensiones. Además, las neuronas de una capa solo estarán conectadas a una pequeña región de la capa anterior, en vez de estar totalmente conectadas. En el último paso, en la capa de salida, se reduce la imagen completa en un vector de puntuación de clases. En la Figura 3.12 podemos observar la arquitectura general de este tipo de redes. Como se puede ver, la capa de entrada de tres dimensiones (en rojo) produce una nueva capa 3D de salida (en azul).

Una manera de entender el funcionamiento de estas redes es que la primera capa intentará establecer patrones de detección de bordes. Luego, las capas posteriores tratarán de combinarlos en formas más simples y, finalmente, en patrones de las diferentes posiciones de los objetos, iluminación, escalas, etc. Las capas finales intentarán hacer coincidir una imagen de entrada con todos los patrones y llegar a una predicción final como una suma ponderada de todos ellos. De esta forma las redes neuronales convolucionales son capaces de modelar variaciones complejas dando predicciones bastante precisas.

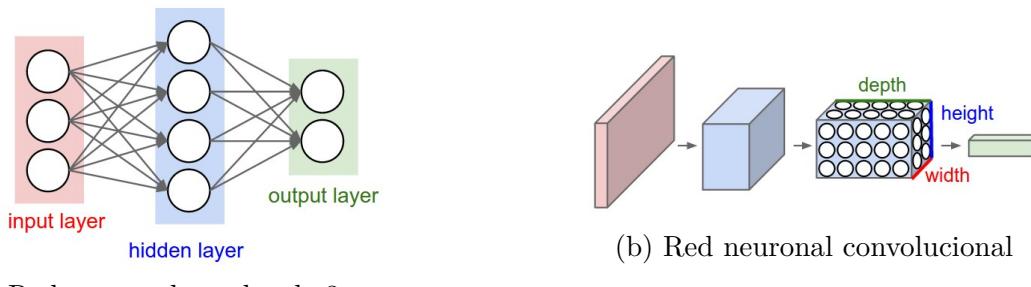


Figura 3.12: Diferencias en la arquitectura de las redes neuronales convolucionales frente a las redes estándar [70]

3.2.2. Capas en una red convolucional

A lo largo de estos últimos años se han desarrollado infinidad de redes neuronales distintas para adaptarlas a diferentes problemas y propósitos. LeNet fue una de las primeras redes convoluciones que permitió comenzar a desarrollar el campo del *deep learning*. En la Figura 3.13 se muestra la arquitectura de esta red. Este trabajo pionero de Yann LeCun fue nombrado LeNet5 en 1994, y se comenzó a utilizar principalmente para reconocimiento de caracteres. Aunque se han propuesto nuevas arquitecturas recientemente: AlexNet, VGG, GoogLeNet, ResNet,... [70], todas usan los conceptos básicos de LeNet y se pueden entender estudiando la estructura básica de la pionera.

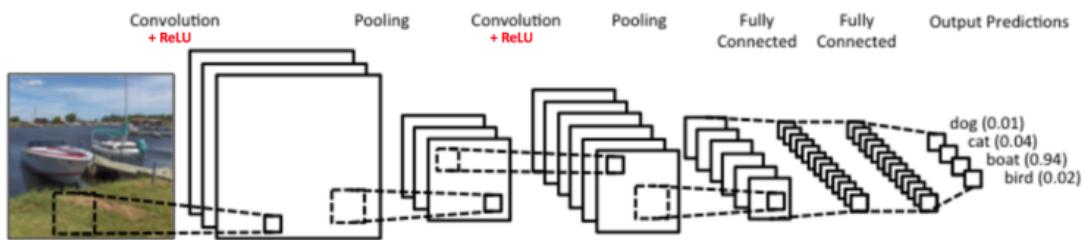


Figura 3.13: Red neuronal convolucional LeNet [69]

En general, en una red neuronal convolucional se usan principalmente cuatro tipos de capas, tal y como observamos en la Figura 3.13:

- Convolución
- Rectificación no lineal (ReLU)
- Reducción o *pooling*
- Densa o totalmente conectada

Además, existen otro tipo de capas que tienen funciones muy específicas como las capas de dropout para reducir el sobreajuste, o las capas de batch normalization para acelerar el entrenamiento.

3.2.2.1. Capa de convolución

En primer lugar analizaremos la capa más importante, donde se realiza el mayor número de operaciones y la que da nombre a este tipo de red. La capa de convolución consiste aplicar a la imagen de entrada un conjunto de filtros o *kernels*, que son objeto de aprendizaje, para obtener un mapa de características de la imagen original. La convolución conserva las relaciones espaciales entre los píxeles, ya que solo utiliza pequeñas matrices que se van deslizando para aprender características de la imagen.

Cabe destacar que cada filtro es pequeño en cuanto al tamaño espacial, pero se extiende por toda la profundidad del volumen de entrada. Por ejemplo, los filtros más típicos tienen un volumen de $5 \times 5 \times 3$ (5 píxeles de anchura y altura y profundidad 3, los canales de colores RGB) o $3 \times 3 \times 3$. Durante su aplicación, el filtro se desliza (en inglés se utiliza el término *convolve*) a lo largo del ancho y alto del volumen de entrada y calcula productos escalares entre la imagen de entrada y los valores del

filtro. Tras completar la convolución obtendremos un mapa de características, lo que se conoce como feature map, en 2 dimensiones, que nos indica la respuesta del filtro a cada posición espacial.

Intuitivamente, lo que se intenta conseguir es que la red aprenda los valores necesarios en los filtros para activar características visuales importantes, como por ejemplo los bordes de un objeto, como se muestra en la Figura 3.14. Para cada filtro que establezcamos en la capa convolucional obtendremos un mapa bidimensional, los cuales apilaremos a lo largo de la dimensión de profundidad para obtener el volumen 3D de salida. Es importante tener en mente que lo que aprende la CNN durante el proceso de entrenamiento son los valores de estos filtros.

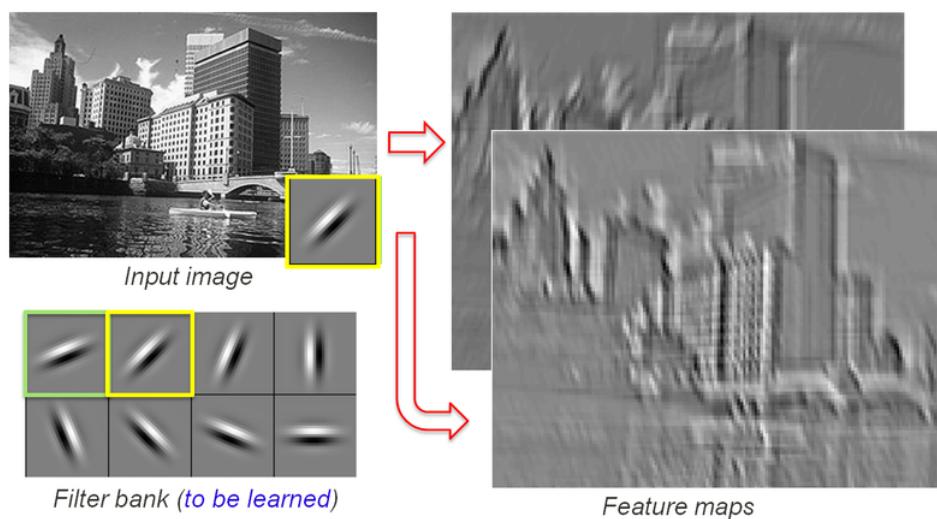
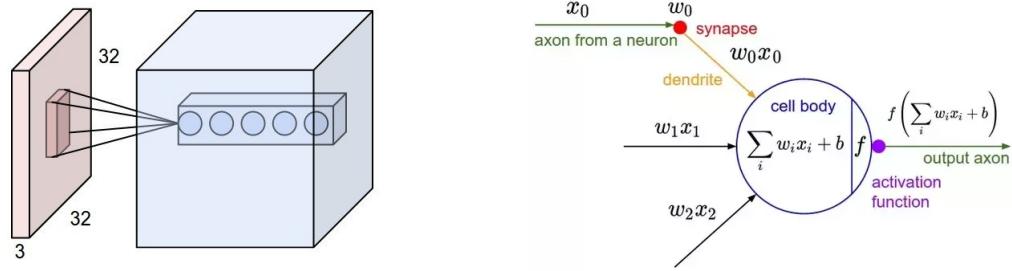


Figura 3.14: Mapas de características resultantes al aplicar la convolución [69]

Dividiremos el análisis de esta capa en tres ideas fundamentales: la conectividad local de las neuronas, la disposición espacial de las neuronas, y el uso de parámetros compartidos en el aprendizaje.

- **Conectividad local:** Como vimos anteriormente, al tener imágenes de entrada multidimensionales, no es práctico conectar las neuronas de una capa a todas las neuronas de la capa anterior. En cambio, ahora se conecta cada neurona solo con una región del volumen de entrada. La extensión espacial de esta conexión la indica el tamaño del filtro que se va a aplicar. La extensión en la dimensión de profundidad es siempre el total de la profundidad de la imagen de entrada. Es importante destacar la asimetría en la que tratamos la dimensión espacial (altura y anchura) y la de profundidad. Es decir, las conexiones son locales en el espacio, pero totales en la profundidad del volumen de entrada.

En la Figura 3.15 se muestra un ejemplo en el que se considera una imagen de entrada de tamaño $32 \times 32 \times 3$ y un tamaño de filtro de 5×5 . En este caso, cada neurona de la capa convolucional estará conectada con una región de $5 \times 5 \times 3$ del volumen de entrada, lo que resultaría en 75 pesos distintos.

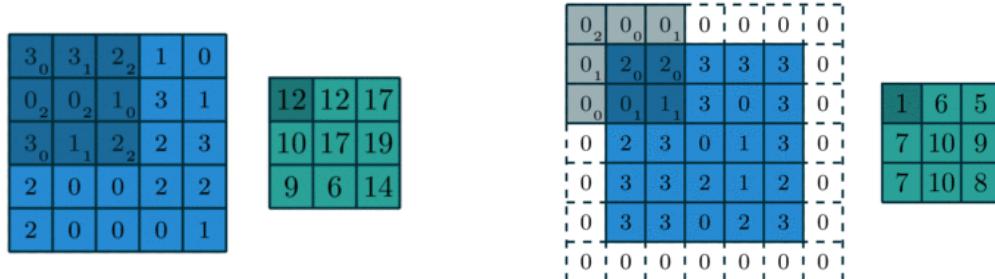


(a) Cada neurona de la capa convolucional se conecta solamente a una región espacial del volumen de entrada, pero en toda su profundidad (todos los canales)

(b) Las neuronas siguen actuando de la misma forma: producto escalar de los pesos con los valores de entrada seguido de una función de activación

Figura 3.15: Conectividad local en una red neuronal convolucional [70]

- **Disposición espacial:** Hasta ahora hemos discutido sobre la conectividad de las neuronas, pero no sobre el número y la disposición de éstas en una capa, lo cual influye en los mapas de características resultantes. Existen tres factores o parámetros que determinan el tamaño del volumen de salida:
 - **Profundidad:** Corresponde al número de filtros que se desea usar, es decir, los distintos mapas de características que deseamos obtener, cada uno intentando buscar patrones diferentes.
 - **Padding:** El uso de *zero-padding* permite añadir ceros en el borde de la imagen de entrada para no tener que descartar los píxeles de los bordes de la imagen al realizar la convolución. Normalmente se usa para preservar el tamaño espacial de entrada, manteniendo así la altura y anchura en la salida. Por ejemplo, en la Figura 3.19b podemos observar como al no usar *padding* el tamaño de salida es menor que el original [71]. Es importante usar el *padding*, además de la finalidad de mantener la dimensionalidad espacial, también para evitar que la información en los bordes de la imagen se pierda rápidamente.



(a) Padding=0, Stride=1

(b) Padding=1, Stride=2

Figura 3.16: Ejemplo de convolución con un filtro 3×3 y distintos parámetros de stride y padding [71]

- **Stride:** Indica el número de píxeles que avanza la ventana deslizante de convolución en cada cálculo. Cuando el *stride* es 1, el filtro se mueve de uno en uno manteniendo el tamaño constante. Sin embargo cuando el *stride* es 2 el filtro salta dos casillas, lo cual produce una reducción espacial del volumen de salida.

En la práctica, *strides* pequeños funcionan mejor, por lo que se suele usar un valor de 1 [71]. Además, esto permite que la reducción de la dimensionalidad espacial solo se lleve a cabo en la capa de pooling, como explicaremos posteriormente.

- **Parámetros compartidos:** Este esquema de compartición de pesos entre los distintos tipos de filtros ayuda a mejorar la eficiencia del sistema. Podemos reducir drásticamente el número de parámetros asumiendo una idea bastante razonable: Una característica que sea útil en una determinada posición (x, y) también lo será en otra (x_2, y_2) . Esto quiere decir que si tenemos, por ejemplo, una imagen hiper-espectral de 200 bandas, obligaremos a que las neuronas en cada nivel de profundidad utilicen los mismos pesos y bias. De esta forma, para cada banda existirá un filtro con unos parámetros que se aplicarán de la misma forma en todas las zonas de la imagen. Este hecho es lo que verdaderamente da lugar al nombre de la capa: Una convolución de los pesos de un filtro o *kernel* con una imagen de entrada.

Tras explicar los elementos y características que definen a una capa de convolución, a continuación se detalla más concretamente, y de forma matemática, el proceso que se lleva a cabo en esta capa. Aunque existen distintos tipos de convolución en función de las dimensiones del *kernel* convolucional, en este estudio se usará la convolución en dos dimensiones dado que es la más conveniente para trabajar con imágenes [72]. En esta operación, los datos de entradas compuestos por m mapas son convolucionados con n filtros o *kernels* bidimensionales de tamaño $K_i \times K_j$, donde i y j representan el tamaño en cada dimensión. Durante el proceso de propagación hacia delante de la red neuronal, cada *kernel* realiza convoluciones a lo largo de todo el ancho y alto del correspondiente mapa de entrada. Posteriormente, la suma de las n respuestas convolucionales produce un mapa de características 2D que pasa a través de una función de activación no lineal (Sección 3.2.2.2). El proceso de convolución al completo se puede formular de la siguiente forma [61]:

$$a_{ln}^{xy} = g \left(\sum_m \sum_{i=0}^{I_l-1} \sum_{j=0}^{J_l-1} w_{lm}^{ij} a_{(l-1)m}^{(x+i)(y+j)} + b_{ln} \right) \quad (3.2)$$

donde l indica la capa que se está considerando actualmente; n es el número de mapas de características de ésta capa; a_{im}^{xy} es el valor de la neurona en la posición (x, y) del mapa de características m de la capa i ; g es la función de activación; m indexa sobre el conjunto de mapas de características de la capa $(l-1)$ que se conecta a la capa actual; I_l y J_l son la altura y anchura del filtro de convolución; w_{lm}^{ij} se corresponde con el peso que conecta la posición (i, j) con el mapa m ; y b_{ln} es el valor de sesgo o *bias*.

Por último, a modo ilustrativo y con el fin de tener una visión global de todos los elementos anteriormente descritos que componen y definen una capa de convolución se presenta el ejemplo que se muestra en la Figura 3.17. En esta demostración se usan los valores digitales de una imagen de tamaño 5×5 y 3 canales. Todos los volúmenes 3D son mostrados con cada componente de profundidad en filas para poder visualizar las operaciones más fácilmente: volumen de entrada (en azul), los filtros de convolución (en rojo), y el volumen de salida (en verde). Los parámetros que se definen para esta capa de convolución son los siguientes:

- Número de filtros = 2
- Tamaño espacial de los filtros = 3
- *Stride* = 2
- *Padding* = 1

Al tener filtros de tamaño 3×3 y *stride* 2, el tamaño del volumen de salida se reduce a 3 de anchura y altura. La figura muestra como los filtros van recorriendo la entrada para producir un valor de activación en la salida. Si nos fijamos en las partes resaltadas de la imagen, podemos observar como cada elemento del output se calcula de la siguiente forma: multiplicamos cada elemento de la imagen de entrada por su correspondiente valor el filtro, sumamos todos los valores obtenidos y añadimos el bias.

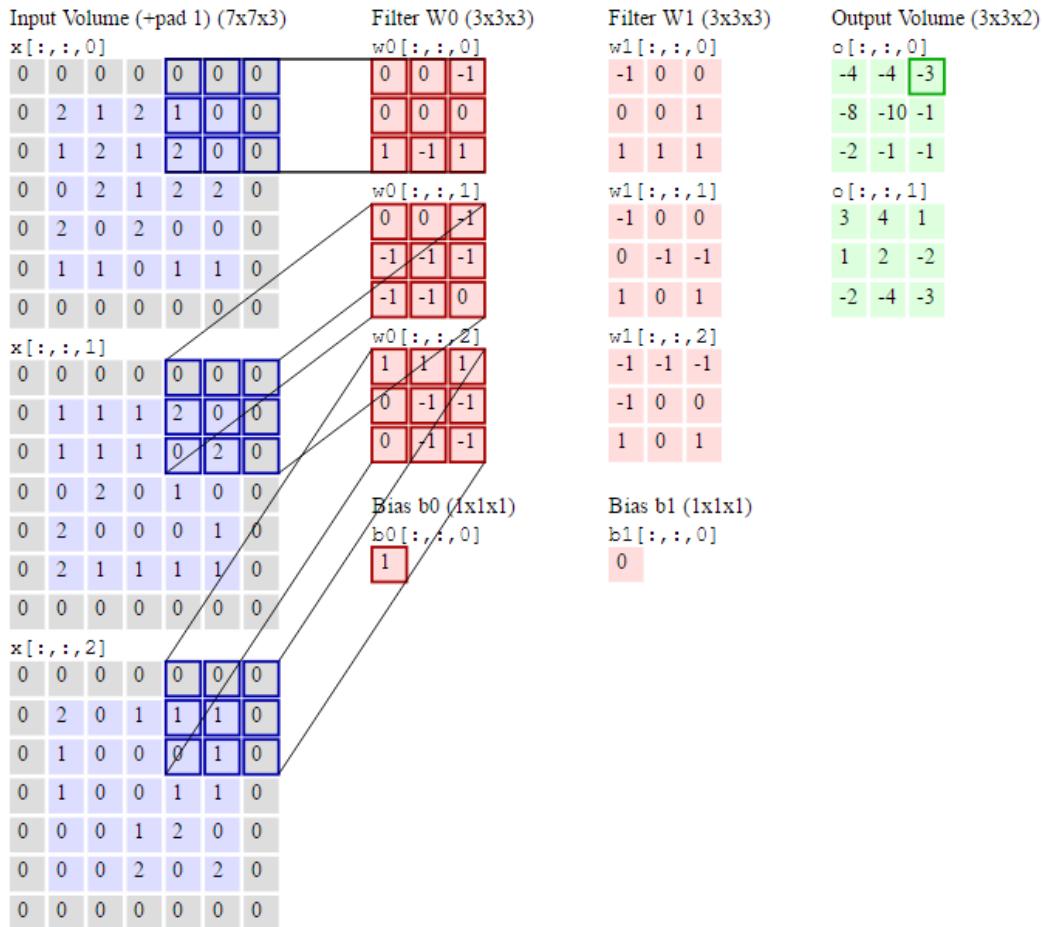


Figura 3.17: Operación de convolución [70]

3.2.2.2. Rectificación no lineal

Una operación adicional conocida como ReLU (*Rectified Linear Unit*) es comúnmente utilizada tras la operación de convolución. Se trata de una operación no lineal que se aplica a todos los píxeles (es decir, a todas las neuronas) y reemplaza todos los valores negativos del mapa de características por valor cero. En forma matemática: $f(x) = \max(0, x)$, siendo x la entrada de la neurona. Las funciones de activación no-lineales son preferidas frente a las lineales ya que permiten aprender estructuras más complejas en los datos. Además, la mayoría de los datos de la vida real son no lineales. Es importante recordar que la operación de convolución es totalmente lineal (multiplicación de matrices y suma de sesgos). Por ello es necesario introducir este tipo de activaciones que tradicionalmente han sido la función sigmoide y la tangente. Sin embargo, estas dos funciones han perdido popularidad debido a que sufrían el problema de desvanecimiento del gradiente en redes neuronales profundas como las convolucionales. Este problema ocurre porque el valor del gradiente en las primeras capas es muy pequeño (la derivada de la función sigmoide se mueve entre 0 y 0,25) y esto se va propagando al resto de capas con número cada vez más bajos, impidiendo que la red entrene de forma correcta. Este problema es aliviado por el uso de la función ReLU, cuyos valores de derivada se mueven entre 0 y 1. Además la función ReLU cuenta con una computación más simple al evitar los cálculos exponenciales de la función sigmoide.

La Figura 3.18 muestra de forma visual el efecto de la operación ReLU al aplicarla a un mapa de características. Como se puede observar, lo que se consigue es detectar mejor los bordes de las figuras presentes en la imagen y descartar el ruido.

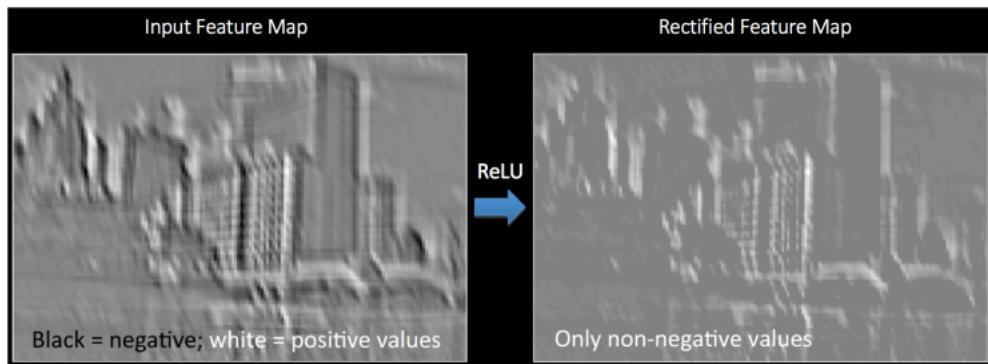
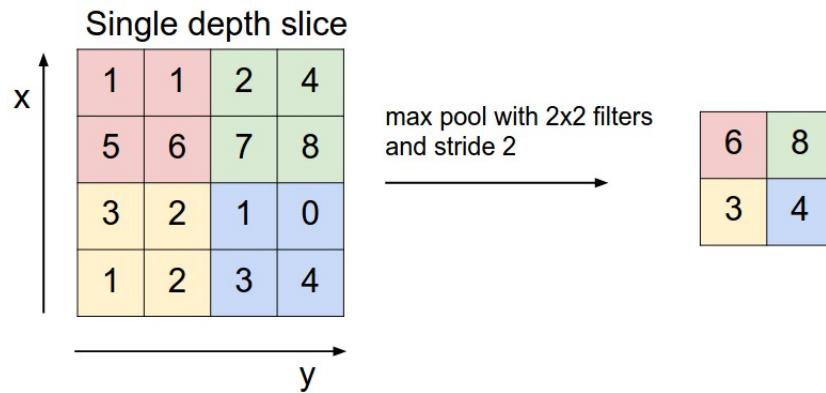


Figura 3.18: Operación ReLU [69]

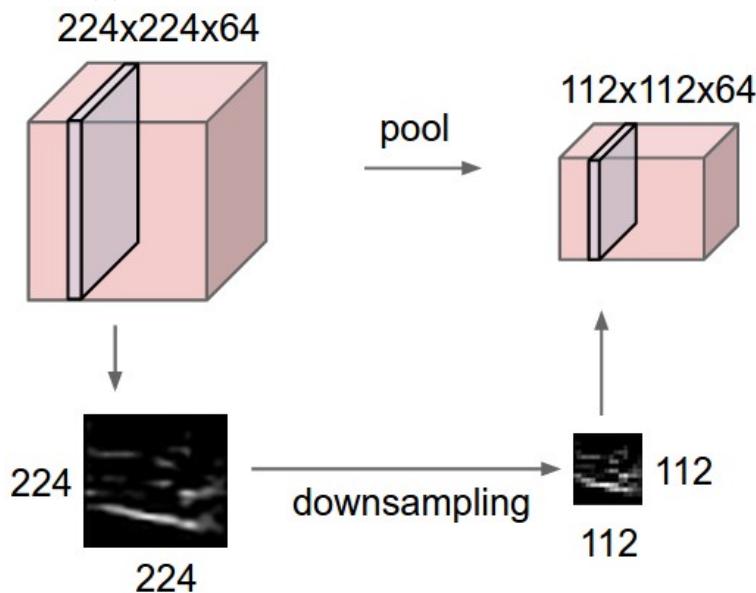
3.2.2.3. Capa de reducción o *pooling*

Es muy común introducir periódicamente una capa de reducción o *pooling* entre capas de convolución. Su función es la de reducir progresivamente el tamaño espacial de los mapas de características para así disminuir el número de parámetros y de cálculos en la red. Además, también se puede considerar una forma de controlar el sobreajuste. La capa de *pooling* opera de forma independiente en cada componente de profundidad y la redimensiona espacialmente usando normalmente la operación *max-pooling*, aunque se pueden utilizar otras como la media. En estas capas, los mapas de entradas son reducidos (*downsampled*) bajo un factor F_x y F_y sobre su anchura y altura, quedando la dimensión de profundidad sin alterarse. Para cada región rectangular no superpuesta de tamaño (F_x, F_y) . la salida se calcula seleccionando

el mayor valor de activación de la región . El motivo de que el max-pooling tenga mayor popularidad se debe a que las características extraídas tienden a codificar la presencia espacial de algún patrón sobre los mapas, y resulta más informativo fijarse en las presencias máximas de distintas características que en las promedias [73]. Lo más común en esta capa es usar filtros de tamaño 2×2 aplicados con un *stride* 2. Esto supone que la entrada se reduce a la mitad tanto en anchura como en altura, descartando así el 75 % de los valores de activación de salida tras la convolución. En este caso, cada operación *max* elegiría el mayor de entre 4 números (una pequeña ventana de 2×2). Téngase en cuenta que en este paso la dimensión de profundidad se mantiene sin cambios. En la Figura 3.19 se muestran los resultados al aplicar la capa de *pooling* a una imagen de entrada. Como se puede observar la imagen de entrada pasa de tener 224 píxeles de anchura y altura a tener 112, pero sigue manteniendo las características y formas más importantes de la imagen, y el mismo número de canales.



(a) Resultado de la aplicación de max-pooling



(b) La capa de pooling reduce la dimensionalidad espacial

Figura 3.19: Capa de reducción o pooling [70].

3.2.2.4. Capa densa o totalmente conectada

Al aplicar todas las capas convolucionales y de *pooling*, las redes utilizan generalmente capas completamente conectadas, *fully connected layers*, en la que cada píxel se considera como una neurona separada al igual que en una red neuronal regular. Esto implica que cada neurona está conectada con todas las neuronas de la capa anterior. Por tanto, sus activaciones se pueden calcular como una simple multiplicación de matrices seguido de una compensación con un *bias*. Su objetivo es formar un vector de características mezclando información de todos los mapas que se han ido generando en las capas ocultas de la red. Además, otra función de esta capa es la de utilizar las características obtenidas para clasificar la imagen de entrada entre varias clases. Esta última capa clasificadora tendrá tantas neuronas como el número de clases que se deben predecir. Como se comentó anteriormente en la Sección 3.1, es necesario aplicar la función *softmax* para obtener las probabilidades asociadas a cada clase y asignar la más probable como predicción.

A parte de usarse para la clasificación, añadir una capa totalmente conectada también es una forma de aprender combinaciones no lineales de las características obtenidas para obtener mejores resultados en la clasificación. Es por eso que es común añadir una o varias capas totalmente conectadas antes de la capa final de clasificación, como podemos observar en la Figura 3.13.

3.2.2.5. Capa dropout

El término dropout hace referencia simplemente al hecho de ignorar aleatoriamente ciertas unidades (neuronas) durante la fase de entrenamiento, con lo que no contribuyen en el algoritmo de retro-propagación. De forma técnica, este método mantiene una neurona activa de acuerdo a una configuración de probabilidad o pone su salida a cero, quedando una red reducida frente a la original. Las aristas salientes y entrantes de una neurona apagada también son eliminadas. Este efecto se muestra en la Figura 3.20, donde se puede apreciar como el número de conexiones de la red original se reduce enormemente al desactivar varias neuronas en cada capa.

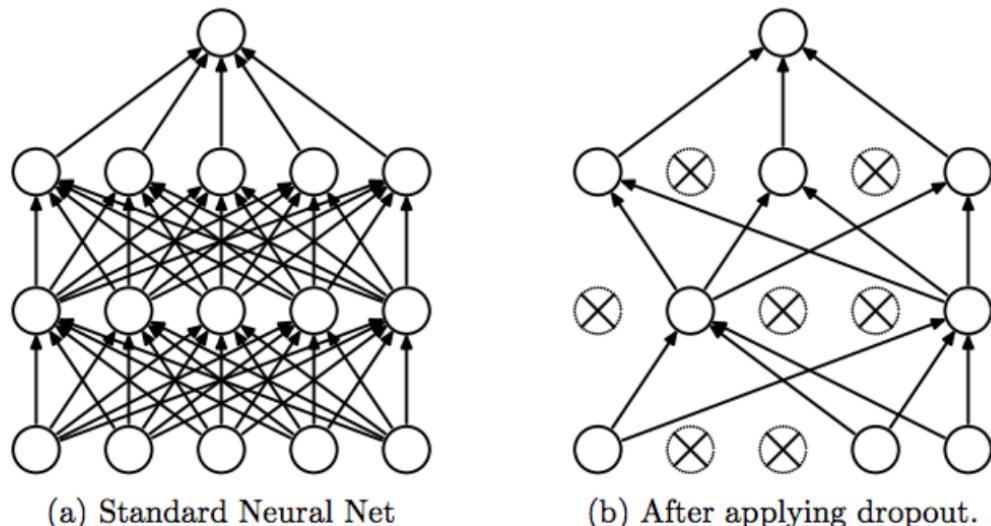


Figura 3.20: Efecto del dropout en una red neuronal [74]

El objetivo principal del dropout es el de actuar como mecanismo de regularización para reducir el sobreajuste. Las neuronas que se apagan no son consideradas en el entrenamiento, lo que obliga al resto de neuronas a no crear dependencias con sus neuronas vecinas. Si no se usa el dropout, las neuronas pueden llegar a aprender relaciones entre ellas, formando un patrón muy específico sobre los datos de entrenamiento, lo cual impide generalizar cuando en la red entran datos nuevos. Usando dropout, las redes consiguen aprender de forma más solitaria, consiguiendo que la red lleve a cabo un entrenamiento mucho más efectivo [75].

Para usar el dropout hay que fijar un parámetro que indique la probabilidad de desactivar las neuronales, siendo 0,5 un valor muy común. Este valor puede ser distinto para cada capa en función de las necesidades, por ejemplo, en las capas ocultas se suele aplicar una tasa de apagado más alta que en las de entrada, para así evitar perder mucha información al inicio. Hay que tener en cuenta que el dropout solo se usa durante la fase de entrenamiento. Para la fase de predicción se usa la red al completo escalando el valor de las neuronas con la probabilidad de dropout para compensar. Otro aspecto a considerar es que usar dropout suele requerir el doble de iteraciones para llegar a la convergencia de la red. Sin embargo, dado que muchas conexiones se desactivan, el tiempo de entrenamiento para cada época es mucho menor.

3.2.2.6. Batch normalization

Las capas de batch normalization tienen como función normalizar la entrada de las capas para cada *mini-batch* de entrenamiento. La normalización es muy importante a la hora de trabajar con imágenes. Dado que los valores de los píxeles de una imagen van de 0 a 255, es necesario concentrarlos en valores más pequeños (de 0 a 1) para ayudar a la red a acelerar el entrenamiento, ya que es más fácil encontrar separación entre atributos a una menor escala. Sin embargo, si la normalización se realiza únicamente en la capa de entrada de la red, su efecto va disminuyendo en las capas ocultas. Por tanto, usar batch normalization ayuda a que los valores sean normalizados también en capas ocultas de la red, manteniendo las distribuciones de los valores en rangos convenientes para el entrenamiento.

Su aplicación tiene muchas ventajas, siendo la principal la aceleración que se consigue en el entrenamiento, ya que impide que las capas tengan que adaptarse a cambios sustanciales en la distribución de los valores de salida de las capas anteriores [76]. Además, también permite que las capas puedan aprender por sí mismas de forma un poco más independiente del resto de capas, mejorando así la eficiencia del entrenamiento. Asimismo, permite usar tasas de aprendizaje con valores más altos, ya que el batch normalization asegura que los valores de activación no se vayan a un rango muy alto o muy bajo. Esto facilita el entrenamiento y ayuda a solventar el problema del desvanecimiento del gradiente. Por último, también cabe destacar que tiene un cierto efecto de regularizador al añadir ruido aleatorio a los valores de las neuronas ocultas. Por ello, su combinación con capas de dropout puede conseguir un mecanismo potente para evitar el sobreajuste en redes muy profundas y con muchos parámetros.

Las operaciones que se llevan a cabo en una capa de batch normalization aparecen desglosadas en la Figura 3.21. Dada un batch de imágenes de entrada con valores x_1, x_2, \dots, x_m se calculan dos estadísticos: la media μ y la varianza σ^2 . Se calculan los valores \hat{x} restando la media y dividiendo por la varianza. Posteriormente, las capas tienen dos parámetros que calculan el valor de salida y que se aprenden durante el entrenamiento: γ realiza un escalado a \hat{x} y β se añade como término independiente (lo que se conoce como *scale and shift*).

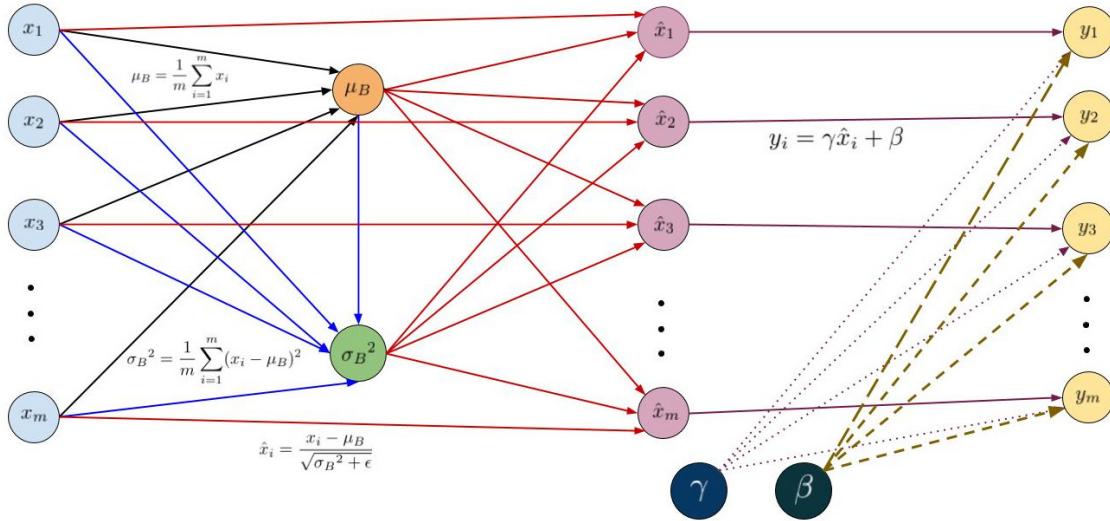


Figura 3.21: Capa de batch normalization [77]