

Redes Convolucionales

Índice

1. La convolución de imágenes	2
1.1. La operación de convolución discreta	2
1.2. La convolución de una imagen	2
1.3. Parámetros de la convolución: <i>Size</i> , <i>Stride</i> y <i>Padding</i>	4
1.4. Midiendo la similitud con una plantilla	4
2. La neurona convolucional	5
3. Arquitecturas convolucionales	6
3.1. Construcción de la etapa convolucional	6
3.2. Ampliación para realizar tareas	8
4. Arquitecturas <i>muy</i> profundas	10
4.1. Inception	10
4.2. ResNet	10
4.3. Aprendizaje con redes preentrenadas.	11
5. Ejercicios teóricos	13
6. Ejercicios Prácticos	13

Referencias

☞ Capítulo 13 de *Hands-On Machine Learning with Scikit-Learn and TensorFlow*

(Todas las imágenes son originales o ©, salvo que se indique lo contrario.)

Tiempo estimado = 4 sesiones de 2 horas (2 semanas)

1. La convolución de imágenes

La convolución es una operación esencial tanto en procesamiento de la señal como de la imagen. En este tema se pretende repasar, de manera intuitiva, que resultado produce la convolución de imágenes y qué utilidad tiene en redes neuronales y visión artificial.

1.1. La operación de convolución discreta

$$f[n] \star g[n] = \sum_{k=-\infty}^{+\infty} f[k] \cdot g[n-k]$$

Recordemos, en primer lugar, qué sucede cuando se convoluciona un tren de impulsos $f(t)$ con una señal $g(t)$. Para simplificarlo supondremos que $g(t)$ tiende a cero por ambos lados, y de tal modo que la información esté concentrada en un intervalo de tiempo menor que el que transcurre entre dos impulsos (tal y como se muestra en el panel izquierdo de la Figura 1). El resultado es que la señal $g(t)$ se clonará en cada t donde haya un impulso.

Si aumentamos la *anchura* de $g(t)$ el resultado es el mismo, pero el aspecto es un poco diferente debido a que habrá algunos instantes donde se solapen dos clones, tal y como se muestra en el panel derecho de la Figura 1.

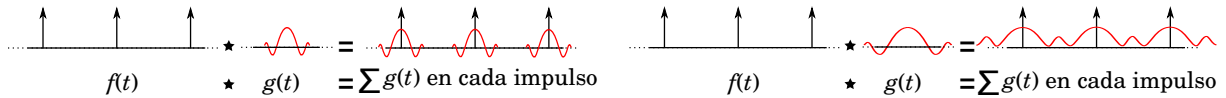


Figura 1: Convolución de un tren de impulsos $f(t)$ con una función $g(t)$ más *estrecha* que el intervalo entre impulsos (izq.) y con otra más *ancha* (der.)

1.2. La convolución de una imagen

Para dar el paso a la convolución de imágenes supongamos ahora que convolucionamos una imagen de tamaño *unidad* con otra que contiene un array de 4×4 unidades pero en cada una de ellas hay un impulso 2D. Al igual que en el caso 1D, el resultado es una imagen compuesta por 4×4 clones de la imagen dada. En el panel izquierdo de la Figura 2 se muestra un ejemplo.

Con estas ideas, consideremos ahora una imagen digital, que no es otra cosa que un array bidimensional (si es en escala de grises) de píxeles. Podemos asimilar cada píxel a un impulso de una cierta intensidad.

¿Qué ocurriría si, como se sugiere en el panel de la derecha de la Figura 2, convolucionamos la imagen con una campana de Gauss bidimensional (formalmente una “*distribución normal bivariada*”)? La campana se clonaría en cada píxel escalada según el valor de intensidad de dicho píxel. Además, como la campana es más ancha que la distancia entre píxeles los clones se solaparán sumándose tal y como ocurría en el panel derecho de la Figura 1, pero ahora en dos dimensiones. Como se puede ver, el resultado es que la imagen resultante está *suavizada*.

Antes de continuar hay que recordar que las funciones convolucionadas deben tener el mismo soporte. En otras palabras, no se puede convolucionar una imagen con otra de diferente tamaño. Lo que sí se puede hacer es convolucionar una parte (*parche*, *ventana*, *vecindad*, *región*) de tamaño $n \times n$, extraída de la imagen original, con otra también $n \times n$, a la que se denomina filtro (*matriz*, *núcleo*, *kernel*) de convolución.

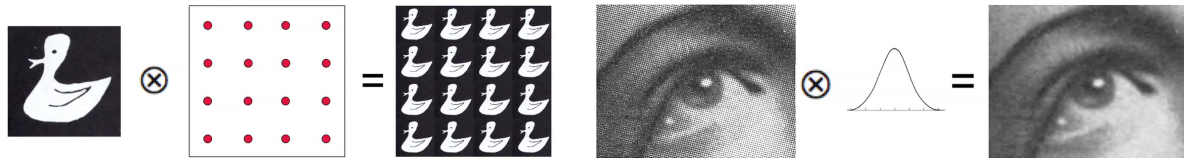


Figura 2: Intuición de la convolución 2D (izq.) y convolución de una imagen 2D una distribución normal bivariada (der.)

Por abuso del lenguaje se habla de convolución de una imagen con un filtro al proceso de:

1. Elegir una ventana del mismo tamaño que el filtro
2. Convolucionarlos, obteniendo un píxel de la imagen resultante
3. Desplazar la ventana en horizontal, vertical o ambos
4. Repetir desde 2 hasta que se recorra toda la imagen.

En la Figura 3 se muestra la etapa 2. En esa figura, además, podemos apreciar que no se cumple exactamente la definición de convolución, porque se multiplican las celdas de la región con su correspondiente celda del núcleo. Si suponemos que el núcleo está volteado tanto horizontal como verticalmente ya estaría resuelto. Pero además, normalmente los núcleos de convolución son simétricos en ambos ejes, con lo que ni siquiera haría falta esos volteos.

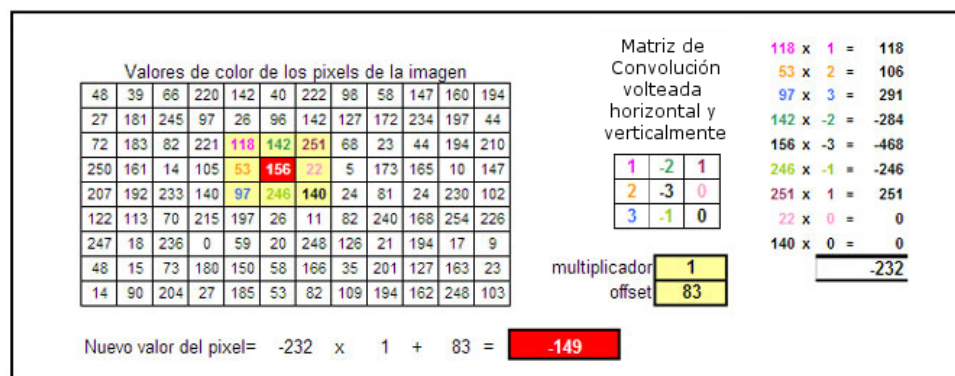


Figura 3: Convolución de una región 3×3 en una imagen con un filtro 3×3


Cualquiera puede probar el efecto de un filtro de convolución con GIMP , cuya herramienta se muestra en la Figura 4.



Figura 4: Herramienta de filtros de convolución de GIMP

1.3. Parámetros de la convolución: *Size*, *Stride* y *Padding*

Size (tamaño) son las dimensiones del filtro de convolución.

Stride (paso) es el número de píxeles que se hace avanzar al filtro de convolución sobre la imagen.

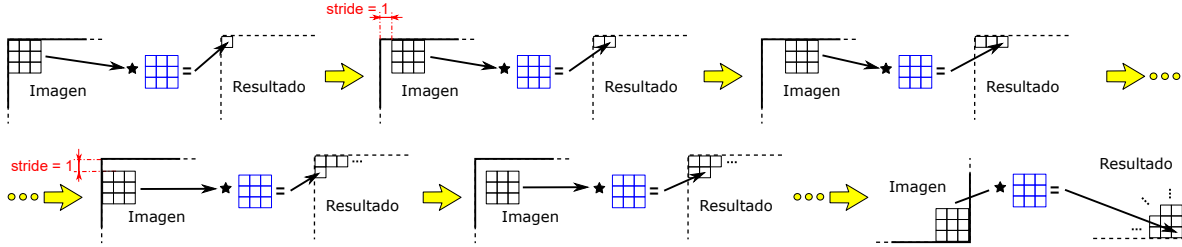


Figura 5: Representación de Stride horizontal y vertical de 1

Padding (relleno) es la manera de hacer que todos los píxeles de la imagen tengan la posibilidad de ser convolucionados, junto con su vecindad, con un núcleo. Al hacer avanzar el núcleo de convolución sobre la imagen desde la esquina superior izquierda hacia la derecha con stride 1, cuando el borde derecho del núcleo coincida con el borde derecho de la imagen no podremos desplazarnos más. La manera de evitarlo es crear un borde de ceros alrededor de la imagen, para poder desplazar el núcleo sobre ellos también y así obtener el mismo número de píxeles.

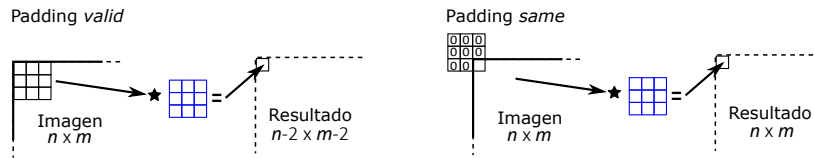


Figura 6: Representación de padding *valid* y *same*

Combinando estos tres parámetros podemos controlar el tamaño de la imagen resultante. Sea O la anchura (o altura) de la imagen de salida, I la anchura (o altura) de la imagen de entrada, K el tamaño del núcleo (cuadrado), P el valor de padding, y S el valor del stride horizontal (o vertical), entonces

$$O = 1 + \frac{2P + I - K}{S}$$

1.4. Midiendo la similitud con una plantilla

La convolución está muy relacionada con la correlación cruzada. La expresión de ambas es muy similar; básicamente se diferencian en *un signo menos*, que implica la inversión de una de las dos señales convolucionadas, y un factor de escala. Esto significa que, si invertimos (volteo en horizontal) una de las dos imágenes, el mismo algoritmo sirve para ambos.

Con la correlación cruzada se mide la similitud entre dos señales, o para nuestros intereses, dos imágenes. El resultado de la operación será mayor cuanto más parecidas sean.

En la Figura 7 se muestra un ejemplo en el que se pretende localizar una sección de la foto. Para ello se convoluciona dicha sección con una ventana del mismo tamaño que se va desplazando por la imagen. El resultado se puede ver en el panel de la derecha, donde hay una posición que destaca claramente, justo en la posición de la ventana que es más similar con la sección dada.

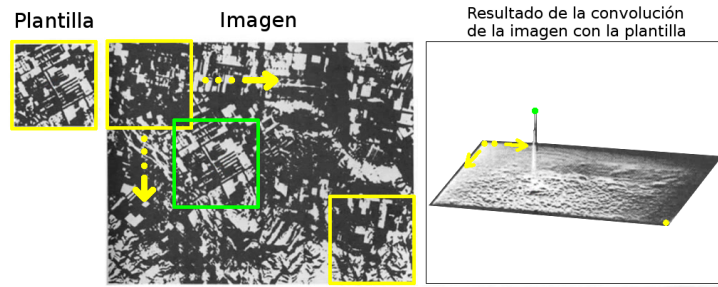


Figura 7: Usando la correlación cruzada con una ventana deslizante para encontrar la plantilla en una foto.

2. La neurona convolucional

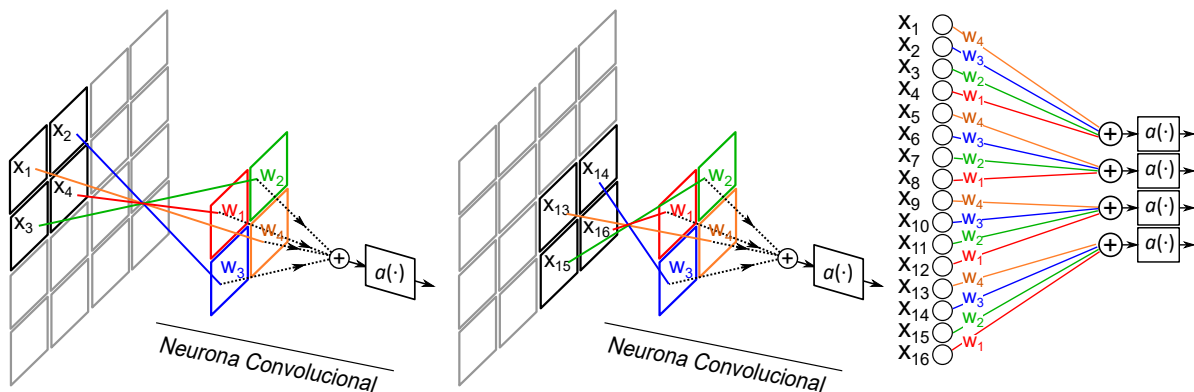


Figura 8: La neurona convolucional es equivalente a varias neuronas clásicas.

Una neurona artificial en una FCNN se conectaba a todas las neuronas de su capa anterior, ponderaba sus entradas, después las sumaba y por último modulaba la salida con la función de activación. La suma ponderada es simplemente una operación lineal de las entradas.

La convolución también es una operación lineal, donde los pesos son los valores de la matriz o filtro de convolución. Así, al entrenar una neurona convolucional estamos aprendiendo un filtro.

Si el filtro de convolución tuviera el mismo tamaño que la imagen que recibe entonces habría tantos pesos como píxeles y la neurona sería densamente conexas. Pero habitualmente el filtro es menor que la imagen. Por ejemplo, en la Figura 8 se muestra un filtro 2×2 con una entrada 4×4 . Como ya se ha explicado, en este caso se debe mover el filtro sobre la entrada haciendo la convolución del filtro con un área 2×2 de la imagen. En la imagen se ha mostrado la convolución sin voltear el filtro ni suponer ninguna simetría. Otro detalle importante es que a la suma se podría añadir un sesgo b , tal y como se explicó en el tema anterior. No se ha pintado por simplicidad, ya que lo único que aporta es un desplazamiento independiente de los valores de entrada.

En la Figura 8 se ha resaltado dicha área con las letras x_1, x_2, x_3 y x_4 . En la figura de la izquierda el área resaltada se corresponde con la esquina superior izquierda, mientras que en la derecha el área es la esquina inferior derecha. Sin embargo el número de pesos es sólo 4, y no 16 como ocurriría en una FCNN. Esta es una diferencia importante, pero no limitante. Se puede resolver,

por ejemplo, actualizando los pesos como si fueran diferentes y luego promediando aquellos que deben ser iguales. En definitiva, si somos capaces de aprender los pesos de las neuronas lo que estamos aprendiendo es el filtro convolucional.

En otras palabras, las neuronas convolucionales dan como resultado elementos que tradicionalmente se utilizan para extraer características visuales de las imágenes (los filtros), y además estos son resultado de un proceso entrenable, es decir no deben ser elegidos por el humano sino que dependerán del conjunto de entrenamiento que se utilice.

De esta manera, gracias a las neuronas convolucionales vamos a ser capaces de alimentar las redes neuronales profundas con las imágenes originales directamente, y no con un conjunto de características extraídas manualmente. Es más, serán las neuronas convolucionales las que se encarguen de aprender las características que se deben extraer. Esta característica es, en buena medida, el motivo de su excelente rendimiento en visión artificial.

3. Arquitecturas convolucionales

A finales de los años 90, Yann LeCun propuso la red LeNet [\[7\]](#). Tras un periodo de asentamiento y asimilación de su propuesta, en 2012 Alex Krizhevsky, en el equipo de Geoffrey Hinton, presentó la red AlexNet al *ImageNet Large Scale Visual Recognition Challenge*, ganando por mucho al resto de competidores y bajando enormemente el error respecto a retos pasados. Este hito se considera habitualmente el punto de arranque de la revolución del DL.

AlexNet contiene todos los elementos esenciales de una red neuronal convolucional (*Convolutional Neural Networks*, CNN): una etapa convolucional compuesta por capas convolucionales y capas de agrupación (*pooling*), seguida de una etapa especializada en la tarea a realizar denominada ‘cabezal’.

3.1. Construcción de la etapa convolucional

Una red CNN típicamente empieza por una etapa convolucional que recibe la imagen y tiene la función de extraer características visuales de ella.

Como hemos explicado, las neuronas involucradas son filtros convolucionales que se aprenden en el proceso de entrenamiento. Un filtro *reaccionará* a una sola característica visual, por tanto si queremos aprender varias, debemos usar varios filtros. Además, las características visuales que es capaz de aprender un filtro dependen del tamaño del filtro. Uno pequeño, 3×3 por ejemplo, será capaz de resaltar bordes horizontales, verticales, esquinas y poco más. Sin embargo, al aumentar el tamaño el número de posibilidades *explota*. La solución propuesta y utilizada habitualmente consiste en crear una jerarquía de características visuales, organizada en capas, de la siguiente manera:

1. Se crea un banco de filtros para extraer características visuales de bajo nivel.
2. La salida resultante de convolucionar la imagen con cada uno de los bancos de filtros debe ser modulada por una función de activación.
3. La salida resultante se reduce en altura y anchura, resultando una imagen en la que las características se han aproximado localmente entre sí.
4. Se crea un nuevo banco de filtros que extraiga las características visuales de la salida anterior. Como en la capa 2 sólo han sobrevivido los píxeles de mayor valor, las características visuales que se aprenden en esta etapa son, en cierta medida, la unión espacial características del nivel anterior, más bajo.

5. La salida anterior es modulada por una función de activación.
 6. La salida anterior se reduce en altura y anchura, aproximando las características de nuevo.
- El proceso se repite hasta que sea posible, o hasta que se considere oportuno. Por ejemplo con una imagen de resolución 4096×4096 , tras 10 repeticiones, se llega a una imagen de resolución 4×4 , en la que ya casi no podemos distinguir que hay representado.

Capa de convolución 2D

Por analogía con las redes FCNN, donde a un conjunto de neuronas ocupando una posición común en la red se le denominaba “capa”, al banco de filtros le denominaremos como “capa de convolución”, ya que es una agrupación de neuronas convolucionales en una misma posición.

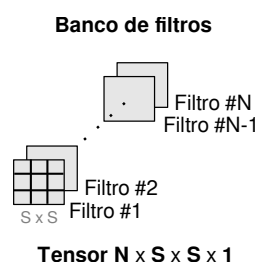


Figura 9: La estructura de datos para un banco de N filtros unidimensionales de tamaño $S \times S$ es un tensor 4D de dimensiones $N \times S \times S \times 1$. La última dimensión es el número de canales; si son en escala de grises es 1, si son color será 3.

Capa de activación

No hay que olvidar que, para que un filtro se pueda llamar con propiedad neurona convolucional, debe especificarse también el tipo de función de activación. Una de las aportaciones más determinantes en AlexNet fue la utilización de la Unidad Lineal Rectificada (*Rectified Linear Unit*, ReLU) como función de activación, cuya expresión es $a(x) = \max\{0, x\}$. Es decir, transmite la entrada sin ninguna modificación siempre que sea positiva, y 0 en otro caso.

El uso de ReLU tiene dos ventajas. En primer lugar hace que no todas las neuronas se activen siempre, sino sólo aquellas que producen una respuesta positiva. Además tiene una mejor propagación del gradiente hacia atrás ya que su derivada respecto de x es 0 ó 1. En comparación con otras funciones de activación, cuya derivada está en el intervalo $(0, 1)$, al concatenar sucesivas capas estamos multiplicando varios números menores que 1, con un resultado cada vez más pequeño. A este problema se le conoce como el desvanecimiento del gradiente (*vanishing gradient*) y es una de las causas por las que las redes neuronales de muchas capas anteriores al 2012 no funcionaban mejor que otras con menos.

Capa de agrupación

La agrupación espacial de características se realiza en la capa de *pooling*. La operación de pooling se define sobre una vecindad V de tamaño $S \times S$, de manera que: $\text{pooling}(V) = \max\{V\}$.

El pooling tiene los mismos parámetros, con el mismo significado, que la operación de convolución: el tamaño (*size*, el paso (*stride* y el relleno()). Es decir que la vecindad se va moviendo sobre la imagen, lo mismo que hacía el filtro en la convolución, sólo que ahora la operación es diferente.

Sin embargo, al contrario de la convolución, al terminar la operación de pooling, el resultado siempre tiene una resolución menor que el original.

3.2. Ampliación para realizar tareas

La etapa convolucional que hemos explicado realiza una extracción de características visuales de la imagen de entrada. Más específicamente, una etapa convolucional produce tantas imágenes como filtros haya en su último banco de filtros. La resolución de estas imágenes dependerá de la resolución de entrada y de las sucesivas operaciones que se hayan realizado sobre ella en las diferentes capas.

De esta manera, podemos sustituir la imagen original por sus características visuales y utilizar una FCNN para realizar tareas de clasificación o regresión.

La ventaja de esta arquitectura es que tanto la etapa convolucional como el cabezal son neuronas que se pueden entrenar con *back propagation*. Es decir que, si por ejemplo estamos aprendiendo a clasificar imágenes, el error de clasificación que se comete con una imagen sirve tanto para actualizar los pesos del cabezal como para aprender mejor las características visuales de las imágenes.

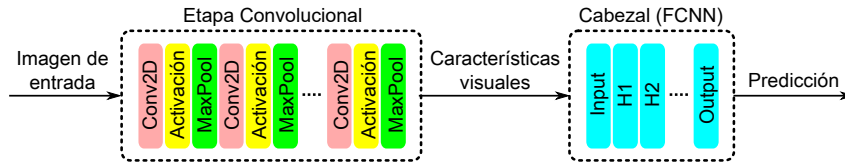


Figura 10: Para darle una utilidad a la CNN es necesario conectar la etapa convolucional a un *cabezal* clasificador o regresor. Como ambos son capas de neuronas se entrenan a la vez.

Con lo visto hasta ahora tenemos la intuición de qué es una CNN. A continuación vamos a ver en detalle algunas consideraciones prácticas para su construcción. Para ello nos planteamos una CNN que tenga una etapa convolucional de dos fases y un cabezal clasificador con 1 capa oculta para Q clases diferentes y cuya entrada son imágenes en escala de grises.

En la Figura 11 se muestra una representación de la 1ª fase de la etapa convolucional, que utiliza un banco de K_1 filtros (su tamaño es irrelevante para la explicación), seguido de MaxPool. En todos los casos se supone un stride de 1 y padding *valid*. Como se puede ver, el resultado son K_1 imágenes de la mitad de ancho y la mitad de alto respecto de la original. Podemos imaginar que este resultado es equivalente a una única “imagen” de resolución $N/2 \times M/2$ y K_1 canales.

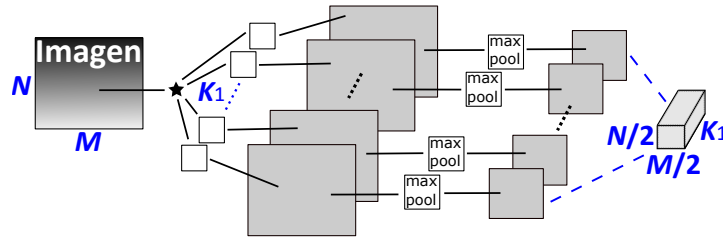


Figura 11: Procesado de la imagen de entrada en la 1ª fase convolucional.

Esta “imagen” es la entrada a la segunda fase convolucional, donde tendremos un banco de K_2 filtros. Pero estos ya no son planos como en la fase anterior, donde sí lo eran porque la imagen que recibían sólo tenía 1 canal. Ahora estos filtros reciben imágenes de K_1 canales, y por lo tanto tienen profundidad K_1 . Sin embargo, cuando se convoluciona cada uno de estos filtros con la salida de la fase anterior sí que resulta en una imagen plana. Tras pasar por el MaxPool llegamos a un resultado equivalente a una única “imagen” de resolución $N/4 \times M/4$ y K_2 canales. De nuevo, en todos los casos se supone un stride es de 1 y padding *valid*. Esta fase se muestra en la Figura 12

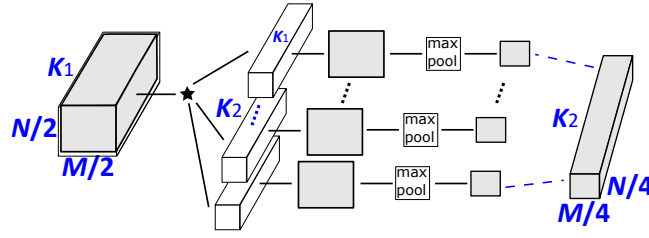


Figura 12: Procesado de la salida de la 1ª fase convolucional en la 2ª fase convolucional.

La entrada del cabezal clasificador la “imagen” de K_2 canales, tensor de dimensiones $N/4 \times M/4 \times K_2$, resultante de la 2ª fase convolucional, pero antes pasar a la capa de entrada de la red FCNN debe ser serializada (*flatten*). Esto significa los $N/4 \times M/4 \times K_2$ píxeles del tensor se recolocan en un array unidimensional con el mismo número de píxeles. El orden en el que se reordena el tensor es indiferente siempre que se mantenga el mismo. Habitualmente esto lo hará una función de la biblioteca de Deep Learning.

Cada píxel alimenta a una neurona de entrada. Después todas las neuronas de entrada se conectan con todas las neuronas de la capa oculta. Finalmente todas las neuronas ocultas se conectan a cada una de las neuronas de la capa oculta, y estas a su vez con las de salida. Esta fase se muestra en la Figura 13, donde la operación *flatten* está dibujada en rojo.

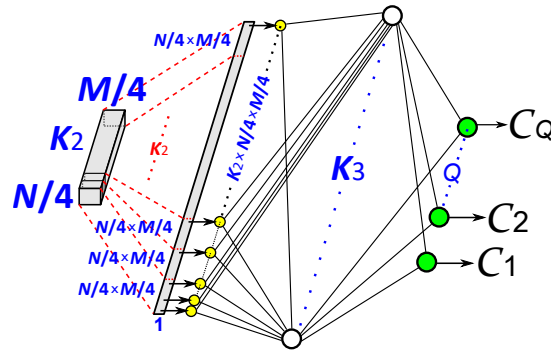


Figura 13: Procesado de la salida de la 2ª etapa convolucional en la etapa neuronal.

Dropout

El dropout es una técnica para evitar el sobre ajuste que consiste en no actualizar algunos pesos de aquella capa a la que se aplica durante el proceso de retropropagación. Normalmente tiene sentido en redes FCNN, es decir en el cabezal regresor y clasificador. La selección de cuales pesos sí se actualizan y cuales no es aleatoria. La única decisión que podemos tomar al respecto es qué porcentaje de ellos se quedará sin actualizar. Visualmente podemos imaginar que algunas neuronas se desconectan durante una fase de retropropagación (ver Figura 14).

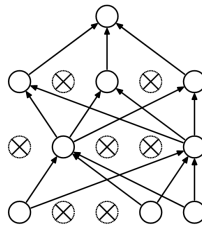


Figura 14: Dropout. (Fuente: N. Srivastava et al. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, JMLR, 2015 [↗](#))

4. Arquitecturas *muy* profundas

De lo explicado hasta ahora se podría deducir que tanto las redes FCNN como las CNN se construyen colocando una capa tras otra. Sin embargo se han propuesto arquitecturas profundas más complejas que, además, logran resultados mucho mejores. A continuación explicaremos las versiones iniciales de dos de ellas: Inception y Resnet. Además nos detendremos brevemente en cómo reentrenar este tipo de redes para no tener que hacerlo desde 0.

4.1. Inception

En 2015 Szegedy *et al.*, investigadores de Google, presentaron la primera versión de GoogleNet, denominada InceptionV1. La novedad consiste en el módulo Inception, que realiza en paralelo la convolución con filtros de diferentes tamaños, además de una reducción de la misma con Pooling, para luego concatenar los resultados, tal y como se muestra en la Figura 15(Izq.) Una versión más compleja del mismo realiza primero convoluciones con filtros 1×1 antes de las convoluciones 3×3 y 5×5 para comprimir el tensor de entrada a otro de profundidad 1. Concatenando módulos Inception, uno tras otro, se construye la etapa convolucional que, seguida de una FCNN, da lugar a la propuesta de red para reconocimiento de imágenes. En concreto, ésta tenía 22 módulos y obtuvo un 5% de error en el *ImageNet Large-Scale Visual Recognition Challenge 2014*.

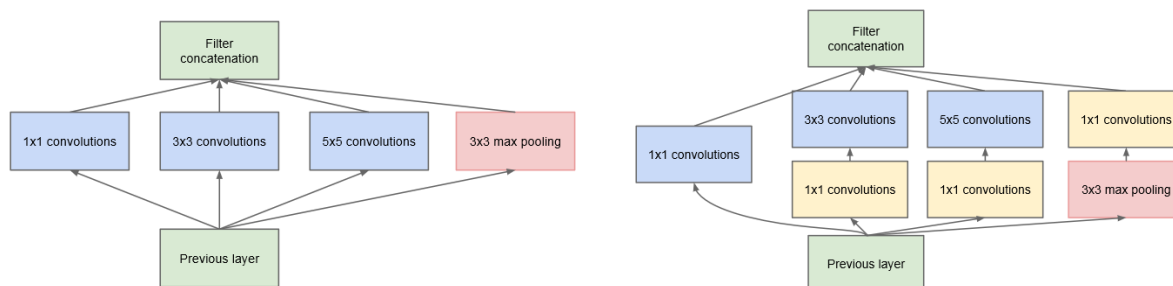


Figura 15: (Izq.) Módulo Inception básico. (Der.) Inception con reducción de dimensionalidad. (Fuente: ‘Going Deeper with Convolutions’, Szegedy et al. [↗](#))

4.2. ResNet

Tras la presentación de AlexNet se observó que aumentando la profundidad de la red se llegaba a un punto en el que no se conseguían resultados mejores, o incluso peores. La figura 16 muestra un resultado experimental de este último caso. Sin embargo parece lógico que, cuanto mayor riqueza se tenga en la representación de las características, mejores resultados tendremos, especialmente en aplicaciones de visión artificial. ¿Por qué ocurre esto? ¿Cómo se puede remediar?

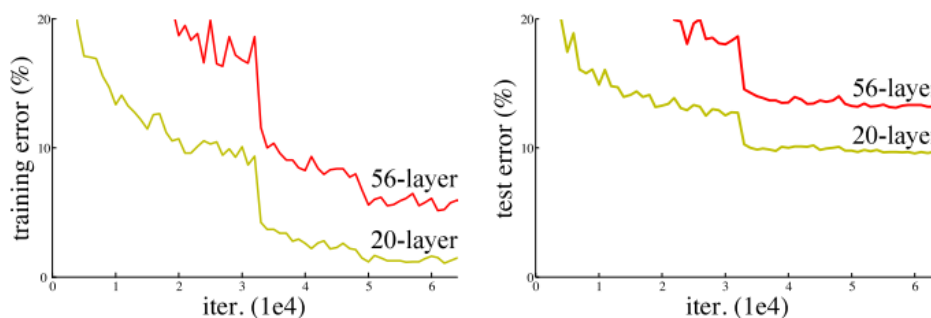


Figura 16: Comprobación experimental de que aumentando la profundidad de una CNN de 20 (línea amarilla) a 56 capas (línea roja) se obtienen peores resultados. (Fuente: ‘Deep Residual Learning for Image Recognition’, He et al. [↗](#))

Como ya hemos comentado, apilar capas y capas de neuronas provoca el desvanecimiento del gradiente. La función de activación ReLU es un modo de atacar el problema; pero si el número de capas es muy grande, resulta insuficiente.

En 2015, He *et al.* presentaron el bloque residual, mostrado en la Figura 17. La idea central es que apilar capas no debería ser un problema porque, idealmente, si tuviéramos una buena representación en la capa j -ésima, todas las capas sucesivas deberían aprender el equivalente a la “identidad”, de tal manera que esa representación se transmitiera hacia delante. Sin embargo, en la práctica, es muy difícil esto ocurra, da igual qué algoritmo de descenso de gradiente utilicemos.

Dicho de otra manera. Supongamos que presentamos una imagen tanto a la entrada como a la salida de una red con una única capa convolucional. Por sencillez, forzaremos el sesgo de las neuronas a cero. Entonces, si $a(x)$ es la función de activación, \mathbf{x} es la imagen de entrada y \mathbf{W} es una matriz que representa todos los pesos de la red, lo que estaremos intentando lograr es que $a(\mathbf{W}\mathbf{x}) = \mathbf{x}$, o sea que la capa convolucional actúe como una matriz identidad. Pero la matriz identidad es muy difícil de aprender (hay que lograr que todos los elementos se hagan cero excepto la diagonal donde todos son unos). Sin embargo, si añadimos a la salida de la capa de nuevo la entrada tendríamos $a(\mathbf{W}\mathbf{x}) + \mathbf{x} = \mathbf{x}$. Este caso es más fácil de aprender ya que sólo se tienen que anular todos los pesos, lo cual es muy fácil porque gracias a haber sumado \mathbf{x} todo el error que se cometa viene de lo que añade la red, que además es modificable.

En la propuesta original de Resnet He *et al.* argumentan que experimentalmente funciona mejor el bloque mostrado en la Figura 17 que sumar directamente \mathbf{x} a la salida de la capa convolucional. Esto puede ser estudiado e incluso puede que optimizado según el problema. Pero sí que es importante darse cuenta de que la suma debe ir **después** de la activación. Si lo colocásemos antes tendríamos $a((\mathbf{W} + \mathbb{I})\mathbf{x}) = \mathbf{x}$, pero $\mathbf{W} + \mathbb{I} = \mathbf{W}'$ no deja de ser una transformación lineal, igual que \mathbf{W} ; tendríamos $a(\mathbf{W}'\mathbf{x}) = \mathbf{x}$ y por tanto estaríamos igual.

De este modo los autores logran entrenar una red de 1000 capas con mejores resultados que otras muchísimo menos profundas.

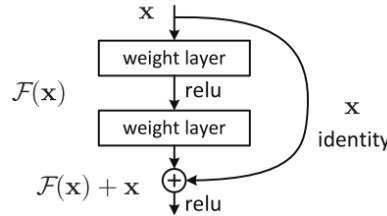


Figura 17: Bloque *Resnet*

El módulo Resnet se ha utilizado también en InceptionV4, la última versión de Inception hasta la fecha. Además, recientemente se han propuesto modificaciones de ResNet como *ResNeXt* [↗](#), *Densely Connected CNN* [↗](#) o *Deep Network with Stochastic Depth* [↗](#).

4.3. Aprendizaje con redes preentrenadas.

Estas arquitecturas de tantas capas normalmente necesitan de un conjunto de entrenamiento enorme y de muchísimo tiempo de ejecución, incluso corriendo en ordenadores con mucha capacidad de computación en paralelo, principalmente debido al uso intensivo de GPUs. Si el conjunto de entrenamiento son imágenes, la dificultad es aún mayor, simplemente por el tamaño de estas y por la gran variedad que nos podemos encontrar. Esta capacidad de cómputo no está al alcance de todos, pero afortunadamente varios equipos investigadores han distribuido libremente redes preentrenadas con un número enorme de imágenes, de muchísimas clases diferentes. Por ejemplo, uno de estos conjuntos de datos es ImageNet, con 1 millón de imágenes *anotadas* y 20.000 categorías, aunque en los retos normalmente se utilizan sólo 1000.

Estas redes se pueden aprender para nuestro problema cambiando la capa de salida por otra con tantas neuronas como clases tengamos (suponiendo que se trata de hacer clasificación). Después se entrena la red con el nuevo conjunto de datos, pero actualizando sólo las últimas capas de la misma. La intuición es que las características visuales de bajo nivel serán comunes para cualquier clase de imágenes, por lo que sólo necesitamos aprender las características de alto nivel y como se conectan entre ellas para predecir la clase. La Figura 18 representa esta idea.

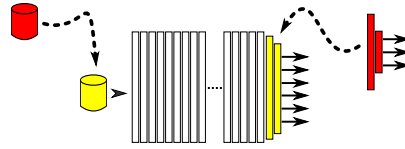


Figura 18: La red ha sido preentrenada con el conjunto de datos amarillo, para un cierto número de clases. Cambiando el conjunto de datos y las últimas capas podemos preservar las características visuales extraídas y aprender sólo las nuevas clases.

5. Ejercicios teóricos

- P.1:** ¿Qué dimensiones tiene la convolución 2D de un tensor de dimensiones $(a \times b \times c)$ con otro de dimensiones $(1 \times 1 \times c)$, padding *valid*? ¿Y con padding *same*?
- P.2:** ¿Qué dimensiones tiene la convolución 2D de un tensor de dimensiones $(a \times b \times c)$ con otro de dimensiones $(a \times b \times 1)$, padding *valid*? ¿Y con padding *same*?
- P.3:** A G.Hinton “no le gusta” la capa de Pooling. Una alternativa es utilizar sólo convoluciones, modificando el padding y el stride. ¿Cómo se construiría una CNN para clasificar el MNIST sin utilizar MaxPool?
- P.4:** Diseñar cabezal clasificador construido únicamente con capas de convolución, de maxpool (opcionalmente) y de activación.
- P.5:** Diseñar también un cabezal regresor con las mismas características.

6. Ejercicios Prácticos

- Construir clasificadores para el MNIST o el CIFAR-10 probando diferentes arquitecturas convolucionales, incluyendo cabezales construidos con FCNN y con CNN, tal y como plantea la pregunta teórica P.4.

Componentes de Keras para FCNN (menú a la izquierda [↗](#))

- *Convolutional layers:*

Conv1D, Conv2D

- *Core layers:*

dropout, flatten

- *Pooling layers:*

MaxPooling1D, MaxPooling2D, AveragePooling1D, AveragePooling2D.

- Probar los modelos preconstruidos en Keras, disponibles en la sección *Applications* de la documentación de Keras [↗](#)