

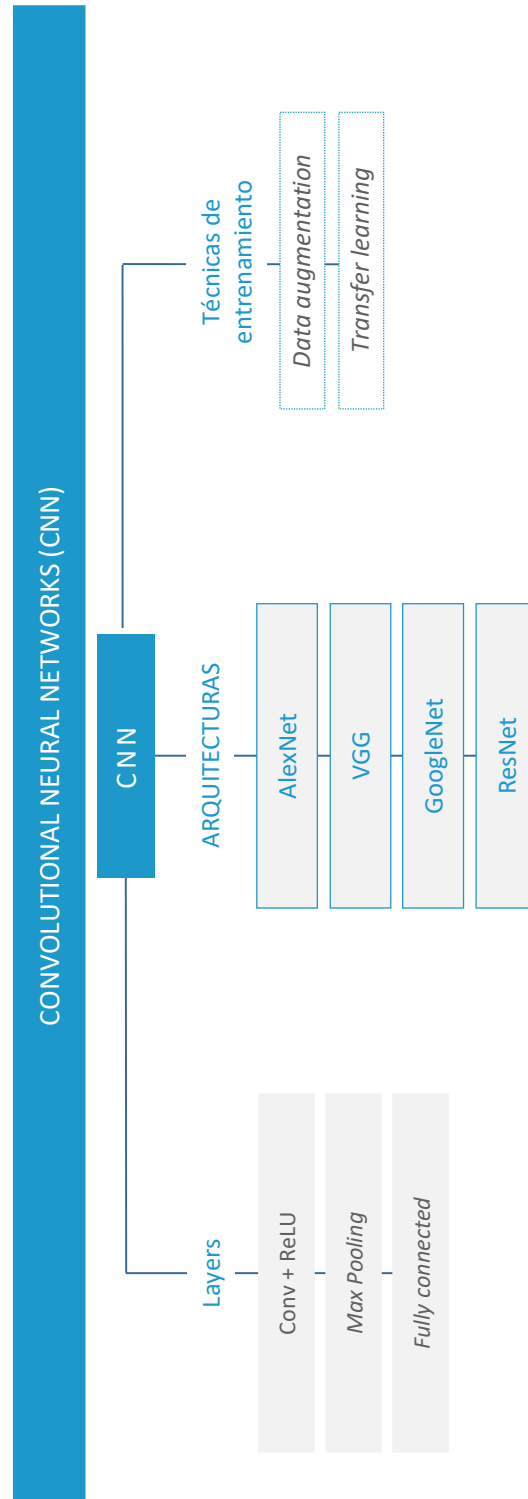
Sistemas Cognitivos Artificiales

Convolutional Neural Networks (CNN)

Índice

Esquema	3
Ideas clave	4
5.1. ¿Cómo estudiar este tema?	4
5.2. Introducción a las CNN	4
5.3. <i>Convolution layers</i>	11
5.4. Arquitecturas CNN para problemas de visión por computador	21
5.5. <i>Data augmentation</i>	26
5.6. <i>Transfer Learning</i>	28
5.7. Referencias bibliográficas	30
Lo + recomendado	31
+ Información	34
Test	35

Esquema



5.1. ¿Cómo estudiar este tema?

Para estudiar este tema deberás leer las **Ideas clave** que se desarrollan a continuación.

En este tema estudiaremos las *Convolutional Neural Networks* (CNN), un tipo de red neuronal muy popular para problemas de visión por computador. Veremos cómo funcionan en detalle estas redes, así como varias arquitecturas reales que han revolucionado el área de *computer vision* en los últimos años.

Es importante en este tema entender cómo funcionan las capas de una CNN y cómo se aplican a las imágenes. También es necesario comprender la dimensionalidad de entrada y de salida de estas capas y tener una idea intuitiva de cómo las redes convolucionales obtienen *features* de alto nivel. Asimismo, es importante entender cómo los conceptos de *Transfer Learning* y *Data Augmentation* ayudan a entrenar redes neuronales a partir de imágenes. Sobre las arquitecturas CNN que estudiaremos (AlexNet y VGG), no es necesario saberse de memoria sus detalles y parámetros, pero sí es importante conocer los ingredientes que las hicieron exitosas.

5.2. Introducción a las CNN

En el tema 2 vimos cómo aplicar una red neuronal a las imágenes del dataset MNIST. En aquel caso, se colocaban todos los píxeles de la imagen concatenados en un vector, que constituía la primera capa de nuestra red neuronal. Esta se encargaba después de clasificar las imágenes mediante el uso de una *hidden layer*.

Esto es una forma correcta de atacar un problema como MNIST. Las imágenes de MNIST son de apenas 28x28 píxeles y en blanco y negro, por lo que el tamaño de la *input layer* era de 784 elementos. Como sabemos, en *fully connected layers*, el número de parámetros crece con el producto del número de neuronas en una capa y el de la anterior (¡importante pensar mentalmente por qué esto es cierto si no lo tenemos claro!). En el caso concreto de MNIST, los números eran manejables.

Sin embargo, como sabemos, las imágenes suelen tener un número mucho mayor de píxeles y pueden estar en color, por lo que hemos de tener en cuenta los canales RGB. En este caso, una imagen de 300x300 píxeles en color tendría un total de $300 \times 300 \times 3 = 270\,000$ valores para representarla. Si esto fuera la *input* de nuestra red neuronal, tendríamos un total de 270 000 parámetros ¡solo por cada neurona en la primera *hidden layer*! Claramente estos números son demasiados altos desde un punto de vista computacional y, además, muy probablemente llevarían a la red neuronal a sufrir de *overfitting* (recordemos que a más parámetros, más capacidad de representación de la red y, por tanto, más capacidad de «memorizar» el dataset con el que se entrena).

Las *convolutional neural networks* o **redes neuronales convolucionales**, o simplemente CNN para los amigos, intentan solucionar este problema. De manera similar a las redes neuronales convencionales, las neuronas de una red convolucional:

- ▶ Toman valores de entrada.
- ▶ Realizan un producto escalar entre sus parámetros y esos valores de entrada.
- ▶ Suman un *bias* y aplican después una *non-linearity*.
- ▶ Igualmente, tendremos una *loss function* a minimizar mediante un algoritmo de optimización.

Nada de esto cambia aquí. Sin embargo, las CNN asumen ciertas características espaciales de los *inputs* que permiten simplificar las arquitecturas, reduciendo en gran medida el número de parámetros.

Las redes neuronales convolucionales surgieron en el **contexto de la visión por computador** y son ubicuas en todo problema que implique el uso de imágenes. Como tal, este tema estará dedicado enteramente al tratamiento de imágenes, y las *inputs* a tratar serán siempre imágenes salvo que se diga lo contrario. No obstante, es importante destacar que las CNN se utilizan también en otros dominios diferentes al de visión, si bien en menor medida.

Historia

Veamos brevemente la historia de las CNN y cómo se han erigido en la solución estándar para problemas de visión por computador. De nuevo, podemos ver cierta inspiración en el campo de la **neurociencia**.

En 1959, Hubel y Wiesel realizaron un experimento por el cual midieron los estímulos del cerebro de un gato ante ciertas formas y figuras. Esto llevó a descubrir que había ciertas áreas en el córtex relacionadas con áreas en el campo visual. Igualmente, se descubrió que las neuronas tienen cierta organización jerárquica, con neuronas sencillas encargándose de respuestas ante la orientación de la luz y otras neuronas más complejas relacionadas con el movimiento y las formas.

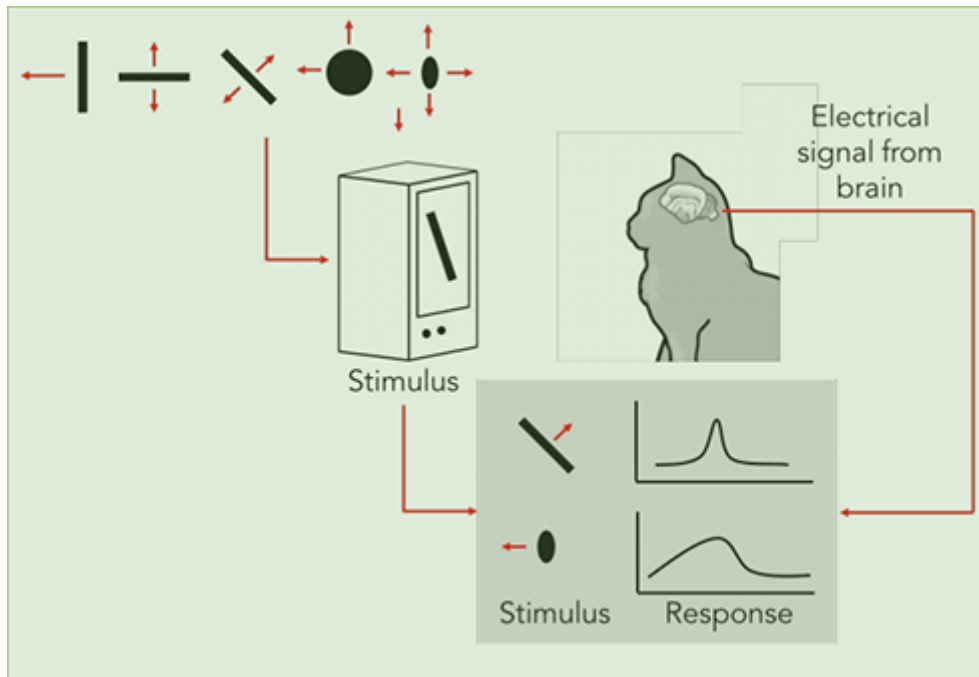


Figura 1. Reacción a estímulos en el cerebro de un gato.

Fuente: http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture01.pdf

En 1998, Yann LeCun, considerado uno de los padres del aprendizaje profundo moderno, introdujo el primer caso práctico de una red convolucional entrenada con *gradient descent* (LeCun, Bottou, Bengio y Haffner, 1998). Esta red era muy efectiva en reconocer dígitos para el servicio postal.

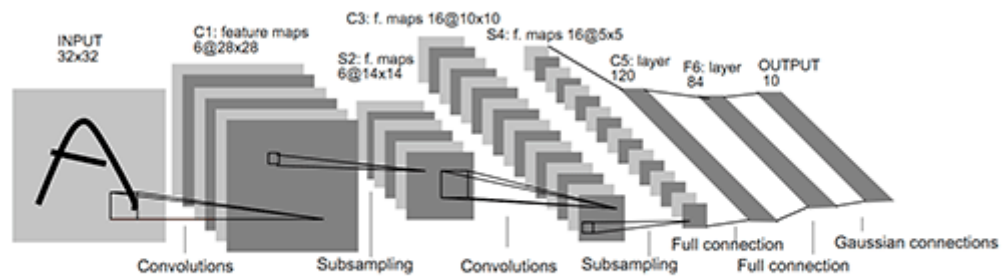


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Figura 2. Arquitectura LeNet-5.

Fuente: <https://world4jason.gitbooks.io/research-log/content/deepLearning/CNN/Model%20&%20ImgNet/lenet.html>

Sin embargo, la verdadera explosión en el uso de las CNN vino, como ya vimos en el primer tema, cuando una red de este tipo ganó en 2012 la competición de clasificación de imágenes ImageNet por un gran margen. Esta CNN era más profunda

que las vistas hasta el momento y presentaba una estrategia de entrenamiento mucho más efectiva, utilizando varios de los elementos que hemos visto en los temas anteriores. Asimismo, se emplearon GPU para una mayor velocidad de entrenamiento.

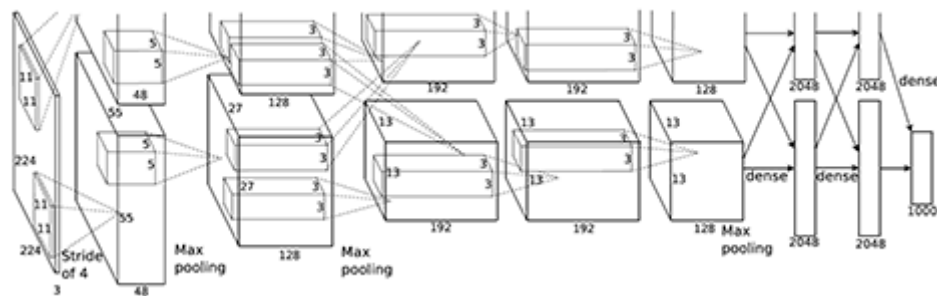


Figura 3. Arquitectura AlexNet.
Fuente: Krizhevsky, Sutskever y Hinton, 2012.

Desde ese momento, las redes convolucionales empezaron a romper récords en tareas de visión por computador, revolucionando prácticamente el campo y permitiendo nuevos sistemas de inteligencia artificial impensables hasta el momento.

Ejemplos de uso de redes convolucionales en la actualidad

Veamos ahora varios de los problemas que están siendo resueltos con CNN en la actualidad.

La clasificación de imágenes

En primer lugar, el problema que lo empezó todo: la clasificación de imágenes. Aplicando una CNN a problemas de clasificación, podemos saber qué objeto o categoría tiene una imagen:



Figura 4. Clasificación de imágenes mediante CNN.

Fuente: Krizhevsky, Sutskever y Hinton, 2012.

Imágenes similares

Las CNN pueden utilizarse en el campo de *image retrieval* para obtener imágenes similares a otras imágenes. Las *features* aprendidas por la red convolucional pueden ser utilizadas para buscar similitudes entre aquellas.



Figura 5. Búsqueda de imágenes similares mediante CNN.

Fuente: Krizhevsky, Sutskever y Hinton, 2012.

Detección de objetos

Otra área donde las CNN han tenido gran éxito es en la detección de objetos (*object detection*). El problema consiste en ser capaz de localizar y situar objetos en imágenes.

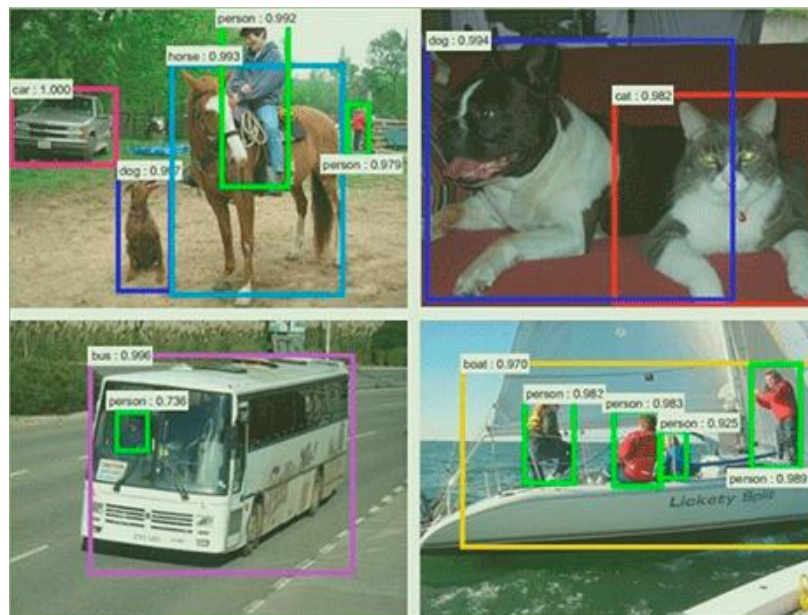


Figura 6. Detección de objetos mediante CNN.

Fuente: Ren, He, Girshick, y Sun, 2015.

Segmentación

Del mismo modo, el problema de *segmentation* también está siendo tratado con CNN. En resumidas cuentas, el objetivo es separar la imagen en todos sus constituyentes píxel a píxel.

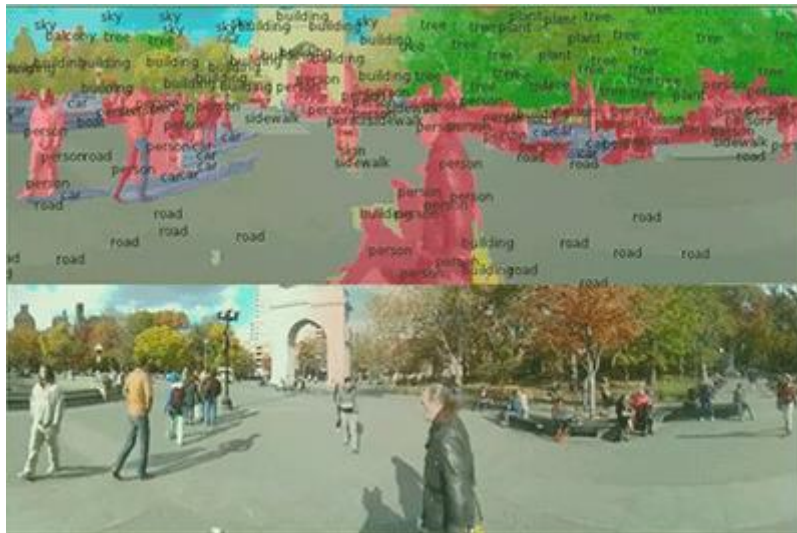


Figura 7. *Segmentation* mediante CNN.

Fuente: Farabet, Couprie, Najman y LeCun, 2012.

Con ejemplos como los vistos hasta ahora, queda bastante claro por qué las CNN son una de las tecnologías clave para los ***self-driving cars* o coches sin conductor**.

El número de aplicaciones no se reduce solo a las vistas hasta ahora. Ciertamente, casi cualquier problema de inteligencia artificial donde estén presentes imágenes o vídeos es un gran candidato a ser tratado con redes convolucionales.

5.3. *Convolution layers*

Las CNN utilizan distintos tipos de capas o *layers*. La capa más importante, y la que da nombre a la red, es la **capa convolucional**. Esta *layer* funciona a partir de unos filtros de tres dimensiones de pequeño tamaño, que van desplazándose por la imagen obteniendo las salidas de la capa.

En la siguiente imagen podemos observar uno de estos filtros. La imagen en este caso tiene un tamaño de 32x32x3, mientras que el filtro es más pequeño, 5x5x3. **Los filtros siempre tienen la misma profundidad (*depth*) que la imagen** (en este caso, 3), ya que se desplazarán a través de la primera y segunda dimensión.

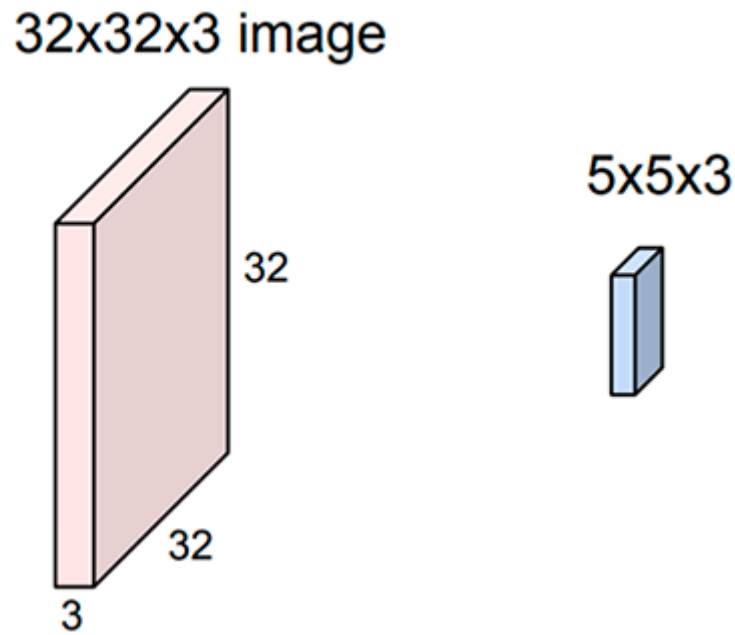


Figura 8. Imagen vista como un volumen en 3 dimensiones (la tercera dimensión corresponde a los canales RGB). A su derecha, un filtro de tamaño 5x5x3.

Fuente: <http://cs231n.github.io/>

En la figura 9, el filtro se mueve por todas las posiciones posibles en la imagen (yendo primero de izquierda a derecha y luego «pasando» a la siguiente línea), y por cada posición obtenemos una activación o un valor de salida. La idea aquí es que el filtro va recorriendo la imagen y obteniendo *features* relevantes. Como se ve, la activación resultante (*activation map*) pasa de tener un tamaño 32x32 a 28x28.

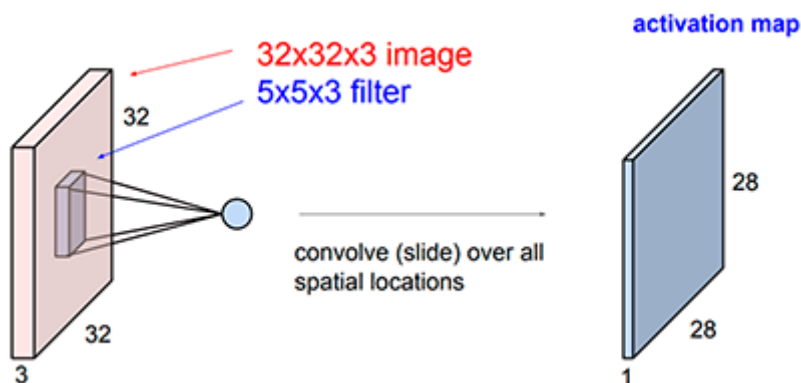


Figura 9. Convolución a partir de la imagen.

Fuente: <http://cs231n.github.io/>

El valor de salida por cada convolución aplicada con un filtro se obtiene mediante la multiplicación de todos los elementos de la imagen en los que el filtro está actuando por cada parámetro correspondiente del filtro. El filtro, de tamaño $5 \times 5 \times 3$, se aplica sobre un área de tamaño $5 \times 5 \times 3$ en la imagen, por tanto, multiplicamos cada número de la imagen con el peso correspondiente del filtro.

Otra forma de verlo es pensar que concatenamos los $5 \times 5 \times 3 = 75$ elementos del filtro en un vector de parámetros w y hacemos el producto escalar con los $5 \times 5 \times 3 = 75$ elementos de la imagen sobre los que el filtro está siendo aplicado. Con esto podemos ver por qué el filtro tiene que tener la misma profundidad que la imagen, ya que si no, no podríamos aplicar esta operación. Igualmente, a esta multiplicación de valores se le aplica del mismo modo un término *bias* b .

Es importante comprender que los parámetros de la red convolucional, los pesos w que aprenderemos durante el entrenamiento, están en los filtros en el caso de las capas convolucionales.

No es común aplicar solo un filtro en una capa convolucional; lo más normal es tener varios filtros para obtener más *features* en cada posición de la imagen. La idea es que cada uno de estos filtros obtenga ciertas características de la imagen que serán importantes a la hora de obtener una representación suficientemente expresiva de la misma. En el siguiente diagrama, vemos cómo se aplica un segundo filtro y se obtiene un nuevo «mapa de activación» de tamaño $28 \times 28 \times 1$.

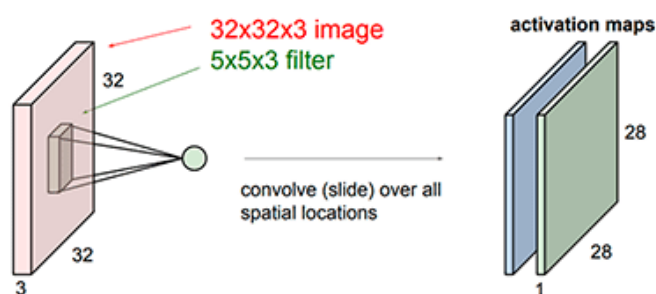


Figura 10. Aplicando un segundo filtro a la convolución.

Fuente: <http://cs231n.github.io/>

Del mismo modo, aplicando más filtros acabamos obteniendo una nueva representación en tres dimensiones de la imagen, conservando sus características espaciales y, a la vez, añadiendo más profundidad con las nuevas *features* calculadas a partir de los filtros.

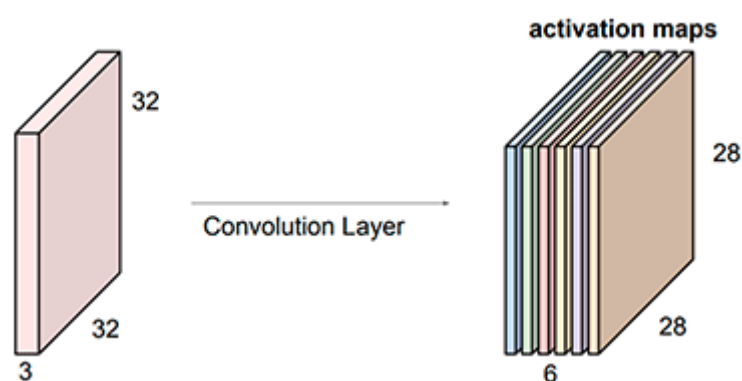


Figura 11. Las redes convolucionales aplican un gran número de filtros para obtener nuevas *features*.

Fuente: <http://cs231n.github.io/>

Es fácil ver cómo hay muchos menos parámetros en estas redes (la suma de los parámetros de cada filtro) para tratar una imagen en comparación con una *fully connected layer*. Dejamos como ejercicio para el alumno hacer esta comprobación.

Creando una red convolucional a partir de las capas convolucionales

Como vemos, las capas convolucionales aplican distintos filtros sobre un volumen de entrada y crean nuevos volúmenes, por lo que las propiedades espaciales de la imagen se mantienen. Lo más común es aplicar capas convolucionales seguidas de funciones de activación ReLU. La unidad ReLU se aplicaría sobre cada valor de activación salido de un filtro aplicado sobre un área. Dicho de otro modo, por cada valor del volumen de salida aplicamos la *non-linearity*.

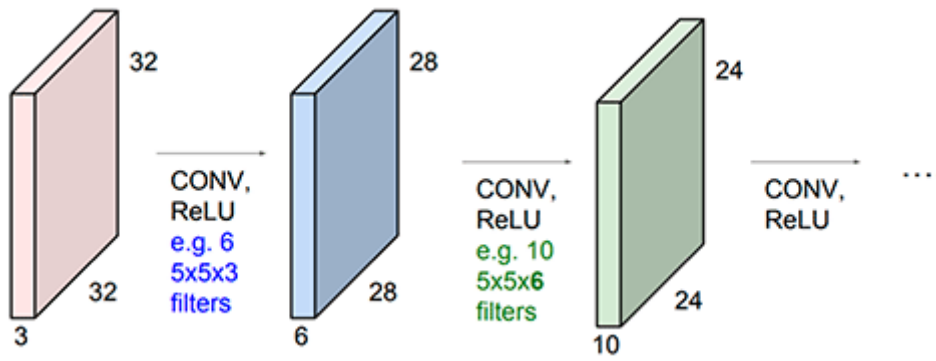


Figura 12. Las CNN suelen estar compuestas por capas convolucionales + ReLUs aplicadas de manera sucesiva.

Fuente: <http://cs231n.github.io/>

La idea aquí es que, al configurar una red de esta forma, las distintas capas van obteniendo una representación jerárquica de las *features*, con las primeras capas reconociendo elementos más simples en una imagen y las siguientes obteniendo representaciones de más alto nivel a partir de estos elementos simples. En la siguiente imagen, podemos ver un ejemplo típico de una red convolucional (con sus distintas capas) aplicada a un problema de clasificación. La capa de *pooling*, POOL en la imagen, será vista más adelante en este tema.

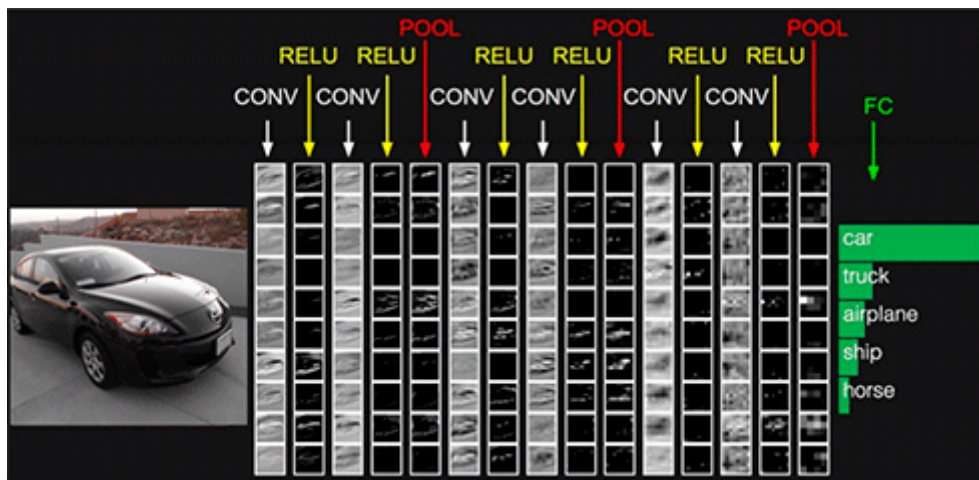


Figura 13. CNN aplicada a un problema de clasificación vista capa a capa.

Fuente: <http://cs231n.github.io/>

Stride y zero-padding

Stride

Veamos unos ejemplos más claros de cómo se realizan las convoluciones espacialmente. Aparte del tamaño del filtro, un elemento importante es el *stride*, que **indica cuánto se desplaza el filtro a través de la imagen**.

Por ejemplo, con un *stride* de tamaño 1, podemos ver en la siguiente imagen cómo un filtro 3x3 se mueve desplazándose un cuadro a la derecha (de arriba a abajo sería similar). En este ejemplo no estamos considerando la tercera dimensión (*depth*) del *input* ni del filtro que, como sabemos, tienen que coincidir.

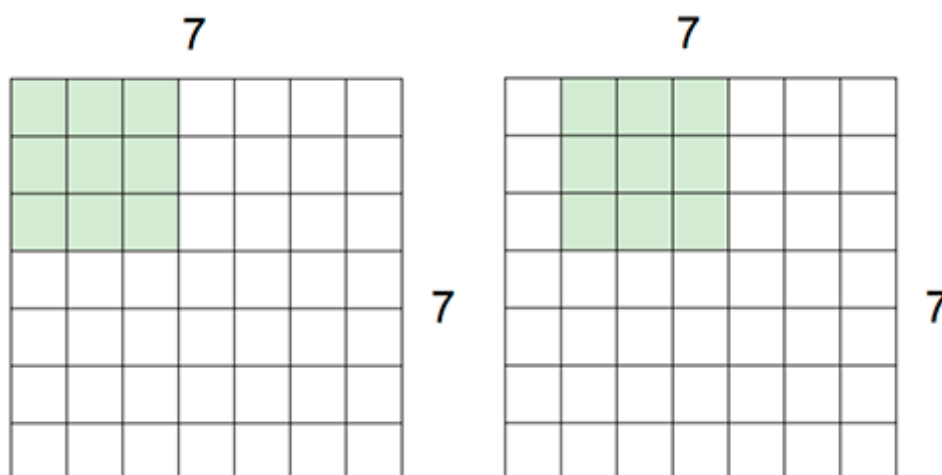


Figura 14. Movimiento de un filtro de izquierda a derecha con stride 1.

Fuente: <http://cs231n.github.io/>

De manera equivalente, un *stride* de 2 haría que el filtro se desplazara dos cuadritos a la derecha. Aumentar el tamaño del *stride* hace que las *features* que obtenemos en la red convolucional se fijen en áreas más distantes de la imagen, pero también implican una **reducción en la dimensionalidad**.

Si:

- ▶ N es el tamaño del *input* para una imagen $N \times N$.
- ▶ F el tamaño del filtro.
- ▶ S el tamaño del *stride*.

Tenemos que la dimensión resultante de cada lado de nuestro *output* sería:

$$[(N - F) / S] + 1.$$

Zero-padding

Para evitar la reducción de dimensionalidad, es común aplicarlo y que el volumen resultante sea del mismo tamaño que el *input*. El *zero-padding* consiste en **añadir ceros a los lados para que las dimensiones cuadren**.

0	0	0	0	0	0			
0								
0								
0								
0								

Figura 15. *Zero-padding* en una imagen.

Fuente: <http://cs231n.github.io/>

Con *zero-padding*, la fórmula anterior cambia un poco. En general, una capa convolucional con *input* y un volumen de tamaño $W_1 \times H_1 \times D_1$:

- ▶ Tiene cuatro hiperparámetros:
 - Número de filtros K .
 - Tamaño de filtro F .
 - Stride S .
 - Cantidad de *zero-padding* P .
- ▶ Produce un nuevo volumen $W_2 \times H_2 \times D_2$ donde:
 - $W_2 = (W_1 - F + 2P)/S + 1$
 - $H_2 = (H_1 - F + 2P)/S + 1$
 - $D_2 = K$

Capas *Max pooling*

Las *max pooling layers* son un tipo de capa que está presente en una gran cantidad de arquitecturas CNN. La idea es **reducir (*downsample*) las representaciones** obtenidas de manera que estas se hagan más pequeñas y sean más manejables computacionalmente, reduciendo el número de parámetros necesarios y, por tanto, ayudando también a reducir el *overfitting*.

Las capas *max pooling* actúan de manera independiente sobre cada nivel de profundidad del volumen de entrada y reducen su tamaño mediante la aplicación de máximos.

La forma más común de utilizar *max pooling* es mediante filtros de tamaño 2x2 y *stride* 2. De este modo, la salida de la capa *pooling* por cada filtro de tamaño 2x2 es el máximo valor en el recuadro donde se aplica. En la figura 16 podemos ver con mayor claridad cómo al aplicar *pooling* el tamaño se reduce en un 75 %.

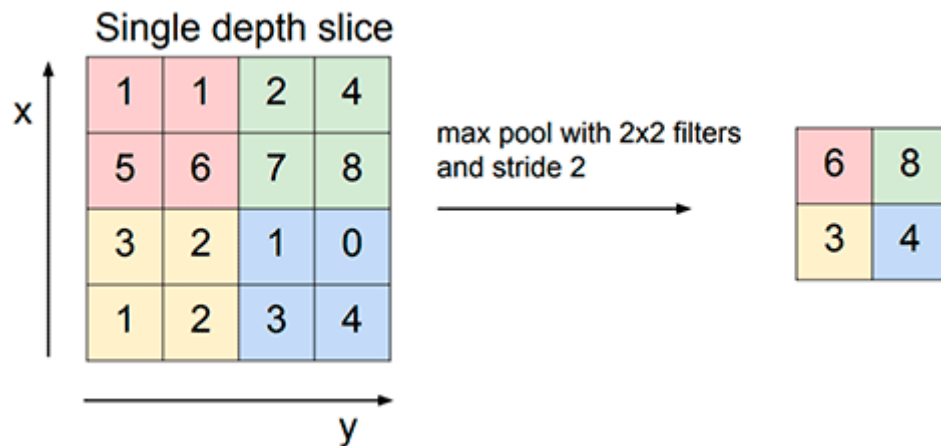


Figura 16. *Max pooling* aplicado en un solo nivel de profundidad.

Fuente: <http://cs231n.github.io/>

Las capas *max pooling* no afectan la profundidad del volumen de entrada. La reducción de la representación ocurre en el **plano espacial**, pero no en el número de *features* distintas obtenidas en cada posición.

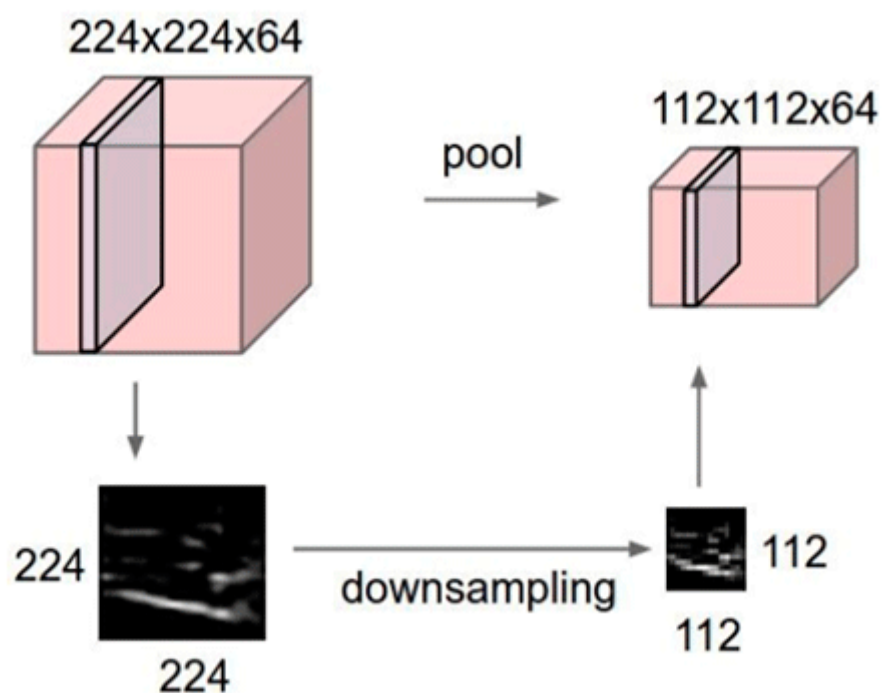


Figura 17. *Max pooling* aplicado en un solo nivel de profundidad.

Fuente: <http://cs231n.github.io/>

Las capas *max pooling* se ubican normalmente entre capas convolucionales, como vimos en una imagen anterior. Una interpretación del funcionamiento mediante la

toma de máximos es que estas capas seleccionan las activaciones más importantes en cada región de una imagen o *input*. Otras posibles estrategias de reducción de dimensionalidad, por ejemplo, obtener la media de valores en el filtro en vez del máximo, no resultan tan efectivas en la práctica.

En resumen, una capa de *max pooling* aplicada a un *input* de tamaño $W_1 \times H_1 \times D_1$:

- ▶ Tiene dos hiperparámetros:
 - Tamaño de filtro o *pool* F .
 - Stride S .
- ▶ Produce un nuevo volumen $W_2 \times H_2 \times D_2$ donde:
 - $W_2 = (W_1 - F)/S + 1$
 - $H_2 = (H_1 - F)/S + 1$
 - $D_2 = D_1$
- ▶ No añade parámetros a la red neuronal ya que calcula una función fija (max).

Fully connected layers

La última capa de una red convolucional para problemas de clasificación es una *fully connected layer*, ya que necesitamos una neurona de salida para cada clase. Normalmente, esto se hace mediante una capa *softmax*. Muchas CNN llevan varias *fully connected layers* como últimas capas para obtener las representaciones finales después de las capas de *convolutions + pooling*.

Volviendo a la imagen que vimos anteriormente (figura 13), una arquitectura CNN común sería una serie de capas convolucionales seguidas de ReLU y *max pooling layers*. La última o últimas capas serían *fully connected layers*.

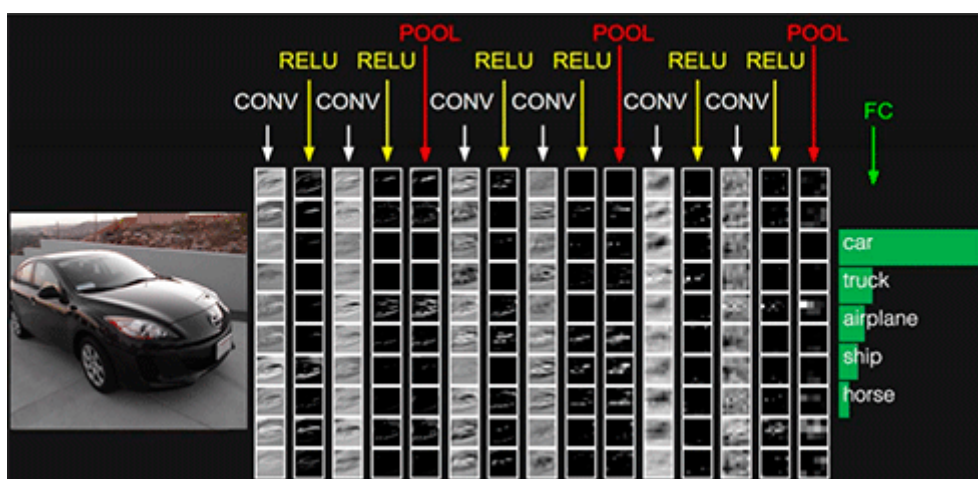


Figura 18. CNN aplicada a un problema de clasificación vista capa a capa.

Fuente: <http://cs231n.github.io/>

5.4. Arquitecturas CNN para problemas de visión por computador

Una vez vistos los ingredientes básicos del funcionamiento de las CNN, veremos aquí algunas de las arquitecturas más conocidas con más detalle. Estas arquitecturas han ido poco a poco obteniendo nuevos récords en la competición ILSVRC (*ImageNet Large Scale Visual Recognition Challenge*). Esta competición evalúa algoritmos para detección de objetos y clasificación de imagen, y utiliza el famoso dataset ImageNet que consiste en 1000 categorías a clasificar.

Como ya dijimos al principio de este tema, AlexNet fue el primer modelo de aprendizaje profundo que ganó esta competición, obteniendo una gran diferencia con respecto a los ganadores de ediciones anteriores. A partir de este hito, nuevas

CNN, cada vez más sofisticadas, han ido batiendo año a año el récord anterior, hasta el punto de que estas redes son capaces de resolver el problema mejor que un humano según varios *benchmarks*.

En la siguiente imagen, podemos ver los ganadores de ILSVRC desde el inicio de la competición. Puede apreciarse el salto cualitativo de AlexNet respecto a los anteriores ganadores, así como la aparición de nuevas arquitecturas CNN cada vez más profundas y efectivas:

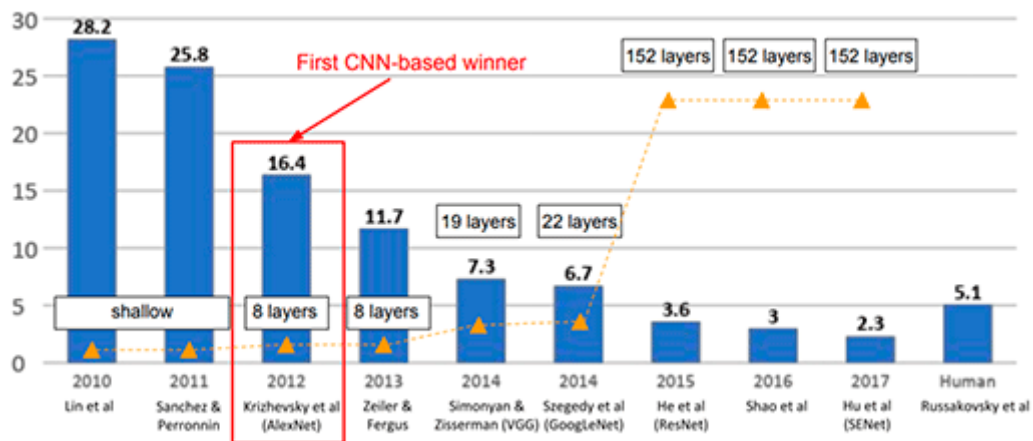


Figura 19. Ganadores de ILSVRC (2010-2017) y comparación con el rendimiento de humanos en la tarea.

Fuente: <http://cs231n.github.io/>

AlexNet

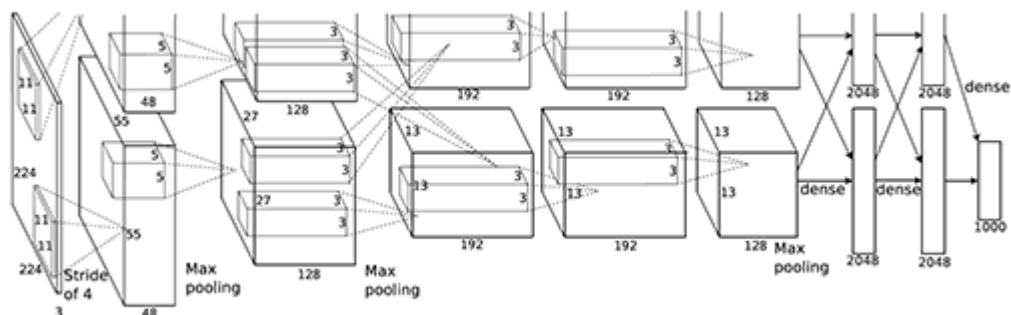


Figura 20. Arquitectura AlexNet.

Fuente: Krizhevsky, Sutskever y Hinton, 2012.

La **arquitectura por capas de AlexNet** es la siguiente:

- ▶ Input: imágenes 227x227x3.
- ▶ Primer bloque:
 - CONV 1 (96 11x11 filters, stride 4, pad 0).
 - MAX POOL 1 (3x3 filters, stride 2).
 - NORM 1 (Normalization layer).
- ▶ Segundo bloque:
 - CONV 2 (256 5x5 filters, stride 1, pad 2).
 - MAX POOL 2 (3x3 filters, stride 2).
 - NORM 2 (Normalization layer).
- ▶ Tercer bloque:
 - CONV 3 (384 3x3 filters, stride 1, pad 1).
- ▶ Cuarto bloque:
 - CONV 4 (384 3x3 filters, stride 1, pad 1).
- ▶ Quinto bloque:
 - CONV 5 (256 3x3 filters, stride 1, pad 1).
 - MAX POOL 3 (3x3 filters, stride 2).
- ▶ Fully connected layer (FC6), 4096 neuronas.
- ▶ Fully connected layer (FC7), 4096 neuronas.
- ▶ Fully connected layer (FC8), 1000 neuronas, una por clase final a predecir.

El único elemento que no hemos visto aquí son las *normalization layers*, una capa específica de AlexNet que no es muy común en la actualidad y que consiste en realizar una normalización que según los autores dio buenos resultados en esta arquitectura.

Como vemos, una característica de AlexNet es que la profundidad de los volúmenes (dicho de otro modo, el número de filtros) va aumentando capa por capa, a la vez que el tamaño espacial se reduce mediante *max pooling*. Si bien esto no es una regla escrita, es algo muy común en arquitecturas CNN. Se puede entender intuitivamente que queremos que la red vaya obteniendo representaciones más complejas, donde los elementos de la imagen se van componiendo de una manera jerárquica.

Otro punto importante de AlexNet es que fue la primera arquitectura en utilizar la **unidad de activación ReLU**. Esta *non-linearity* es la más utilizada en la actualidad en el mundo del aprendizaje profundo y, como vemos, su origen proviene de las arquitecturas profundas para visión por computador.

Como ejercicio, sería interesante que el alumno compruebe, bloque por bloque de la arquitectura, que las dimensiones tienen sentido a partir de lo que hemos aprendido en los puntos anteriores.

Sin embargo, hay algo que resulta extraño en el esquema de la arquitectura: **¿por qué los bloques están partidos en dos trozos a lo largo de la profundidad del volumen?** Por ejemplo, los 96 filtros del primer bloque se reparten en dos partes de 48 elementos de profundidad cada una. Pues bien, esto se debe a que esta arquitectura fue entrenada utilizando GPU para acelerar el entrenamiento. Las tarjetas gráficas utilizadas solo disponían de 3GB de memoria (en la actualidad los números son mayores), por lo que era imposible mantener toda la arquitectura en una sola de ellas. De este modo, el entrenamiento se realizó distribuyendo la red en dos GPU, lo cual es una estrategia que veremos en temas posteriores.

Otros detalles interesantes sobre AlexNet son:

- ▶ Input: imágenes 227x227x3
- ▶ Se usa *dropout* con valor 0.5.
- ▶ El *batch size* es 128.
- ▶ El algoritmo de optimización es SGD con Momentum 0,9.
- ▶ El *learning rate* utilizado es 1e-2, dividido por 10 manualmente cuando el *validation error* deja de mejorar.
- ▶ Se aplica regularización L2 con peso 5e-4.
- ▶ El resultado final es un *ensemble* de 7 modelos AlexNet, lo cual permite reducir el *error rate* notablemente.

VGGNet

VGGNet es una arquitectura similar a AlexNet que se caracteriza por ser más profunda y utilizar filtros más pequeños, de tamaño constante 3x3. Fue ganadora de ILSVRC en 2014 junto con la arquitectura GoogLeNet. Hay dos versiones de esta arquitectura, VGG16 y VGG19, con 16 y 19 capas respectivamente.

Podemos ver la arquitectura en las siguientes imágenes. Como vemos, de manera similar a AlexNet, los bloques son cada vez más profundos y las dimensiones espaciales se reducen. Todas las convoluciones utilizan filtros 3x3 con *stride* 1 y pad 1, mientras que las capas *max pooling* utilizan 2x2 *pools* con *stride* 2.

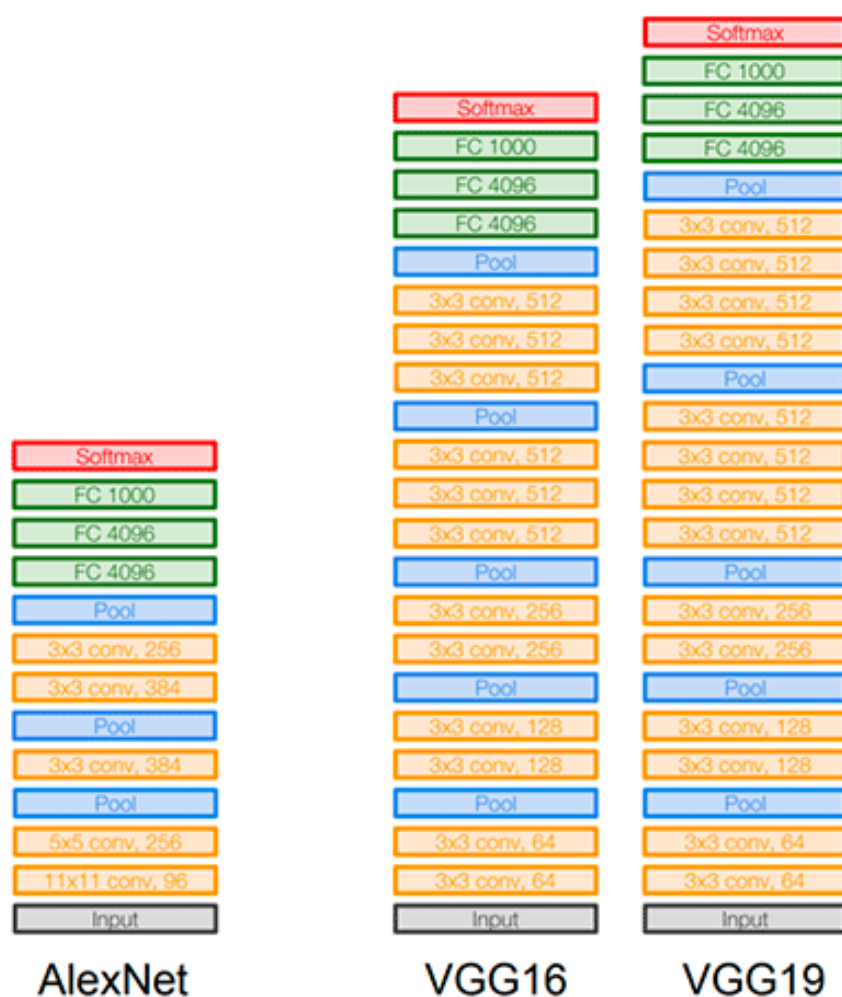


Figura 21. Arquitecturas VGG16 y VGG19 junto con AlexNet.

Fuente: <http://cs231n.github.io/>

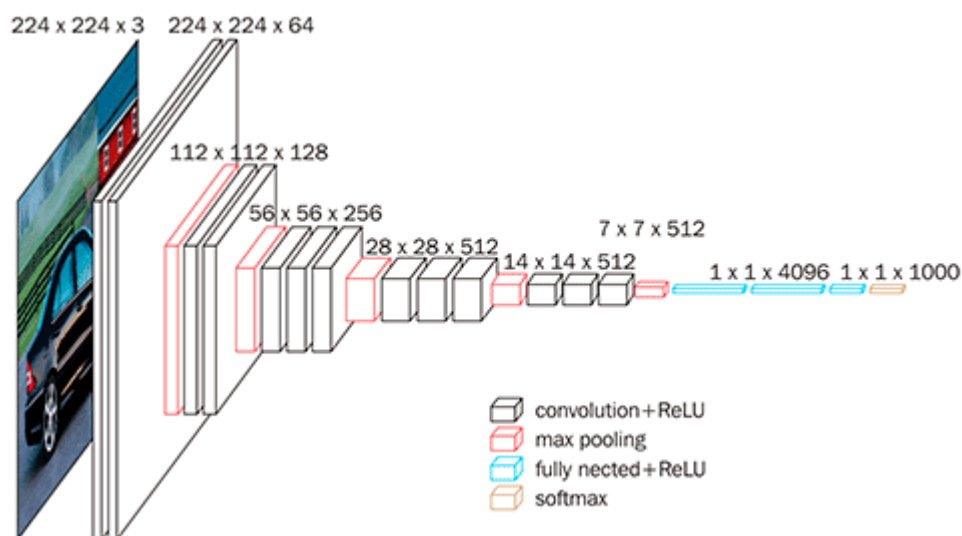


Figura 22. Arquitectura VGG.

Fuente: <https://www.safaribooksonline.com/library/view/machine-learning-with/9781786462961/21266fa5-9e3b-4f9e-b3c6-2ca27a8f8c12.xhtml>

El número de parámetros total de esta red es de aproximadamente 138 millones, en comparación con los aproximadamente 60 millones de AlexNet.

Un ejercicio interesante para el alumno sería obtener estos números.

Este enorme tamaño de VGG la hace un poco más compleja de entrenar y utilizar en la práctica. Arquitecturas más recientes tienden a eliminar las *fully connected layers* del final, lo cual no parece afectar mucho el rendimiento.

5.5. Data augmentation

Una técnica muy aplicada en problemas de visión por computador es la de *data augmentation*. Consiste en realizar transformaciones o pequeñas perturbaciones aleatorias de la imagen de manera que obtenemos nuevas imágenes con las que entrenar, pero utilizando la misma clase o *label*. Esto permite que nuestra red vea nuevos (pero ligeramente distintos) ejemplos que

consisten en el mismo objeto que queremos clasificar, obteniendo así nuevos puntos de vista y permitiendo reducir la cantidad de datos que necesitamos para entrenar.

Un ejemplo de una modificación de la imagen sería utilizar la **imagen simétrica**.

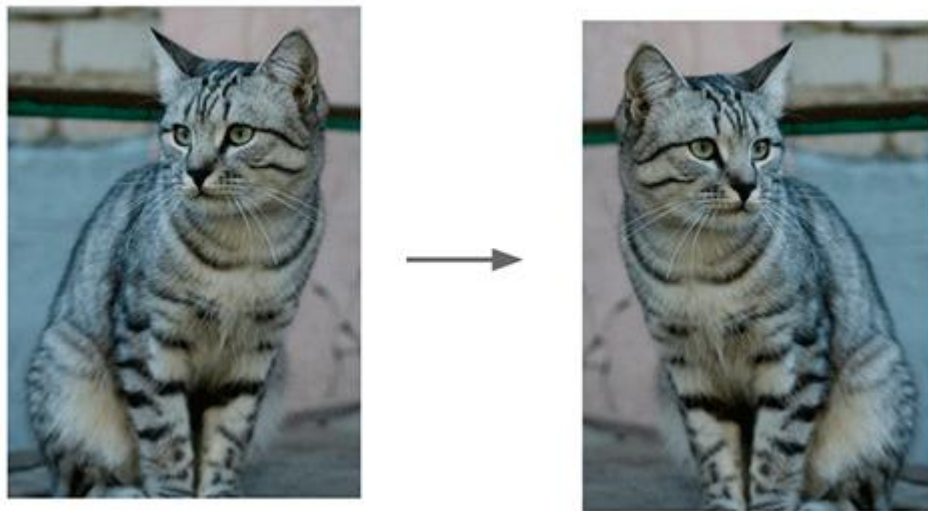


Figura 23. Ejemplo de imagen simétrica.

Fuente: <http://cs231n.github.io/>

Otra opción es cambiar aleatoriamente **el contraste y el brillo** de la imagen:



Figura 24. Ejemplo de modificación de contraste en una imagen.

Fuente: <http://cs231n.github.io/>

En general, cualquier modificación donde la imagen siga teniendo la misma clase es válida. Por ejemplo: rotaciones, translaciones, estiramientos de la imagen... Otra forma muy común de aplicar *data augmentation* es mediante la obtención aleatoria de **pequeñas partes recortadas de la imagen**.

El proceso de obtener más *training data* mediante *data augmentation* puede también verse como una forma de **regularización**. Al añadir cierta aleatoriedad y variaciones en las imágenes, estamos impidiendo en cierta medida que la red aprenda ciñéndose a elementos particulares de las imágenes originales.

5.6. Transfer Learning

Una situación común a la hora de entrenar modelos de visión por computador es que no contamos con una cantidad de datos suficiente para entrenar una CNN. Como hemos visto, las CNN son modelos complejos con muchos parámetros y, por lo tanto, necesitan una gran cantidad de datos para ser entrenados con éxito. Una arquitectura tan grande aplicada a pocos datos llevará con gran probabilidad a *overfitting*.

Es muy difícil y costoso hacerse con una cantidad de datos como la que tiene ImageNet. *Transfer learning* es una técnica que intenta mitigar este problema mediante la **transferencia de lo aprendido con grandes datasets a problemas relativamente similares**. La idea consiste en utilizar un modelo ya entrenado (por ejemplo, VGG entrenado en ImageNet) y utilizar el *training data* de nuestro problema modificando solo los parámetros de las últimas capas del modelo, congelando el resto. De este modo, el aprendizaje que se hizo sobre el problema original se transfiere en parte al nuevo problema, intentando mantener elementos que probablemente son comunes a ambos (recordemos que en las primeras capas las redes neuronales tienden a aprender representaciones sencillas como formas e interacciones básicas).

Por ejemplo, si queremos entrenar un clasificador de razas de gatos con 10 clases, podemos coger una red entrenada con las 1000 clases de ImageNet y volver a entrenar reiniciando la última capa y permitiendo que solo esta pueda ser entrenada. De este modo, únicamente las combinaciones de representaciones de alto nivel cambian, adaptándose al nuevo problema.

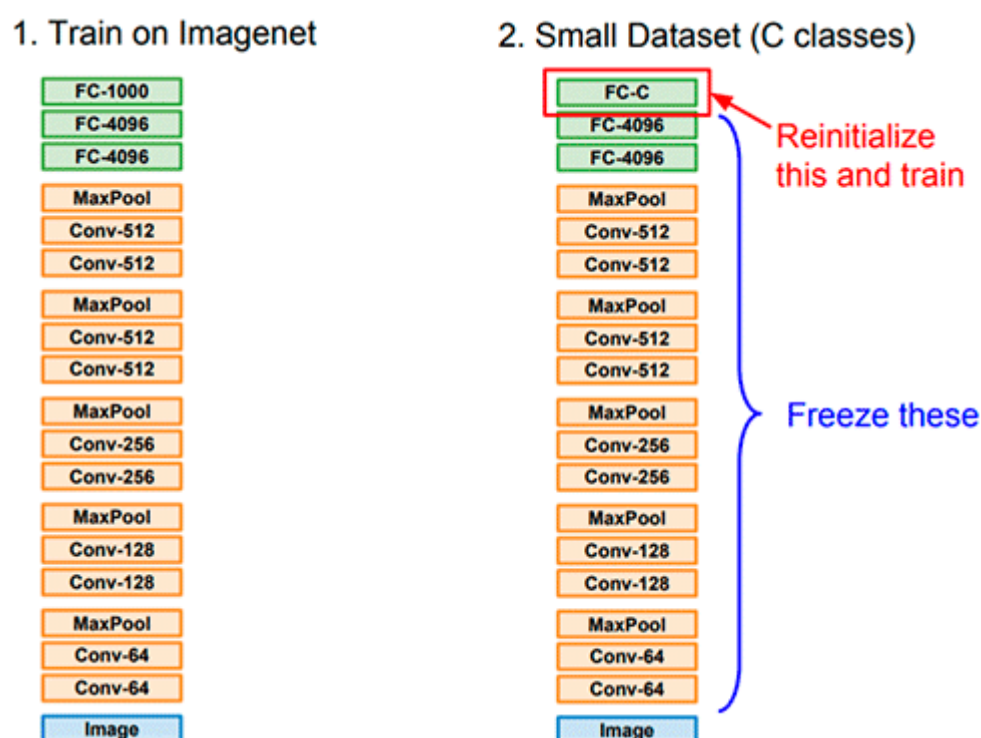


Figura 25. *Transfer learning*.
Fuente: <http://cs231n.github.io/>

Transfer learning es extremadamente común en el mundo del *deep learning*. No hay realmente una fórmula exacta a la hora de aplicarlo. Según la cantidad de datos de la que dispongamos, podemos reinicializar y entrenar un mayor número de capas del modelo original. Por otro lado, la efectividad del *transfer learning* puede verse comprometida si el problema a tratar es muy distinto del problema con el que se entrenó la red original.

5.7. Referencias bibliográficas

Farabet, C., Couprie, C., Najman, L. y LeCun, Y. (2012). Learning Hierarchical Features for Scene Labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8), 1915-1929. DOI: 10.1109/TPAMI.2012.231. Recuperado de <http://yann.lecun.com/exdb/publis/pdf/farabet-pami-13.pdf>

Krizhevsky, A., Sutskever, I. y Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in neural information processing systems*, 25(2). DOI: 10.1145/3065386. Recuperado de <https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>

LeCun, Y., Bottou, L., Bengio, Y. y Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278-2324. DOI: 10.1109/5.726791. Recuperado de http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf

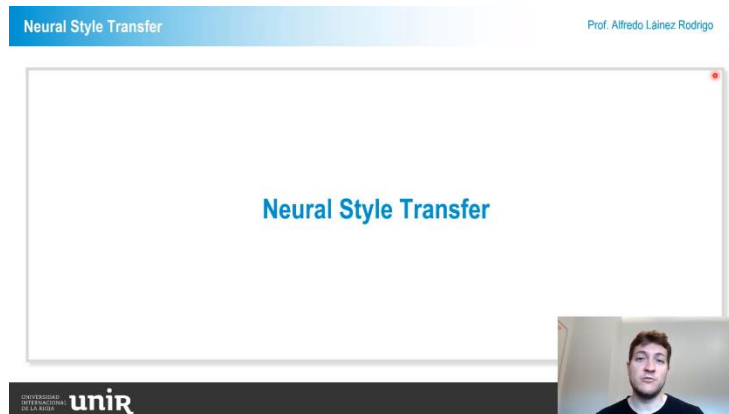
Ren, S., He, K., Girshick, R. y Sun. J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6), 1137-1149. DOI: 10.1109/TPAMI.2016.2577031. Recuperado de <https://arxiv.org/pdf/1506.01497.pdf>

Lo + recomendado

Lecciones magistrales

Neural Style Transfer

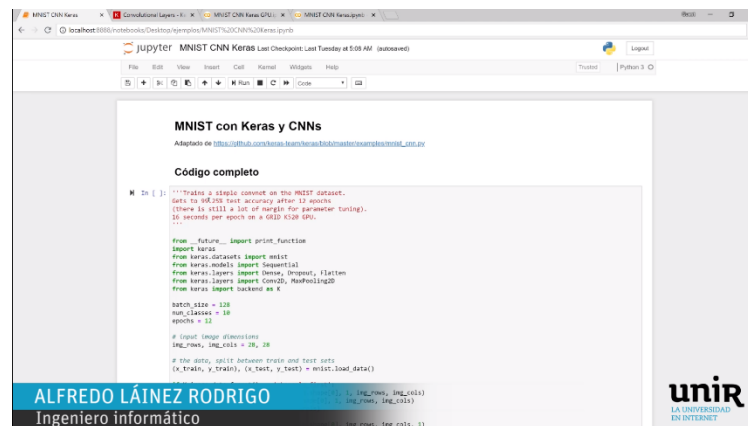
En esta sesión profundizaremos en una de las aplicaciones más interesantes de las redes convolucionales, es de tipo artístico cuyo objetivo es cambiar el estilo de una imagen (*content image*) aplicándole el de una segunda imagen (*style image*).



Accede a la lección magistral a través del aula virtual

Entrenamiento con CNN

En esta ocasión vamos a resolver un problema sencillo de MNIST utilizando Keras y Convolutional Neural Networks (CNN), con lo que deberíamos obtener unos valores de acierto de casi un 99 %, lo que asegura su efectividad.



```
from keras import print_function
import keras
from keras.callbacks import ModelCheckpoint
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

batch_size = 128
num_classes = 10
epochs = 10

# input image dimensions
img_rows, img_cols = 28, 28

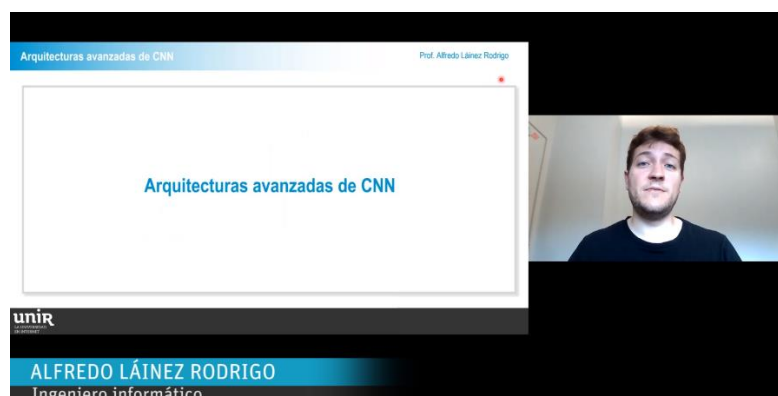
# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

img_train, img_test, y_train, y_test = mnist.load_data()
img_train, img_test, y_train, y_test = mnist.load_data()
```

Accede a la lección magistral a través del aula virtual

Arquitecturas avanzadas de CNN

En esta magistral veremos con detalle Google Net y Residual Networks, dos arquitecturas avanzadas de CNN ganadoras de la competición de ImageNet.



Accede a la lección magistral a través del aula virtual

No dejes de leer

ImageNet Classification with Deep Convolutional Neural Networks

Krizhevsky, A., Sutskever, I. y Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in neural information processing systems*, 25(2). DOI: 10.1145/3065386.

El *paper* original de AlexNet. Interesante echarle un vistazo para ver muchos de los conceptos estudiados en este tema y durante el curso en una arquitectura real.

Accede al artículo a través del aula virtual o desde la siguiente dirección web:

<https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>

Webgrafía

Convolutional Neural Networks en Stanford CS231N

Estos apuntes cubren gran parte de los contenidos que hemos visto hasta ahora. Este gran recurso nos servirá también para profundizar con más detalle acerca de las redes convolucionales. Especialmente interesante es el esquema en movimiento donde se explica el funcionamiento de las convoluciones. Para este tema recomendaremos una serie de enlaces

CS231n Convolutional Neural Networks for Visual Recognition

Accede a la página web a través del aula virtual o desde las siguientes direcciones:

Stanford CS231N: <http://cs231n.github.io/convolutional-networks/>

Entender las CNN: <http://cs231n.github.io/understanding-cnn/>

Cómo utilizar *Transfer Learning*: <http://cs231n.github.io/transfer-learning/>

Bibliografía

Simonyan, K. y Zisserman, A. (2014). Very Deep Convolutional Neural Networks for Large-Scale Image Recognition. Recuperado de <https://arxiv.org/abs/1409.1556v6>

1. Clasificar imágenes mediante el uso de redes neuronales convencionales con *fully connected layers* (marca la respuesta correcta):
 - A. Se vuelve complicado computacionalmente al aumentar el tamaño de las imágenes.
 - B. No es buena idea porque no respetamos las características espaciales de las imágenes.
 - C. Es algo bastante común. Muchos problemas de *computer vision* se solucionan con *fully connected neural networks*.
 - D. Es común entrelazar *fully connected layers* con *convolutional layers*.
2. Las CNN (marca todas las respuestas correctas):
 - A. Empezaron a tener una gran importancia a partir de 1998.
 - B. Se han convertido en una tecnología clave en problemas de visión por computador.
 - C. Son utilizadas en *self-driving cars* para problemas como detectar peatones y otros coches a través de cámaras.
 - D. Empezaron a tener una gran importancia en la década de 2010.
3. Cuando movemos un filtro por la imagen aplicando convoluciones (marca la respuesta correcta):
 - A. Los parámetros del filtro son parámetros distintos en cada posición de la imagen sobre la que se aplica.
 - B. Los parámetros del filtro son parámetros distintos en ciertas posiciones de la imagen detectadas por la red neuronal.
 - C. Los parámetros de un filtro son siempre los mismos para toda la imagen.

4. Las *max pooling layers* (marca la respuesta correcta):
- A. Tienen dos parámetros y dos hiperparámetros.
 - B. Tienen dos hiperparámetros y no tienen parámetros.
 - C. No tienen parámetros ni hiperparámetros.
 - C. Tienen dos hiperparámetros y el número de parámetros de la capa depende del tamaño de la imagen.
5. Durante *backpropagation*, en la *max pooling layer* (marca la respuesta correcta):
- A. El gradiente no se propaga hacia atrás ya que no hay parámetros.
 - B. El gradiente no se propaga hacia atrás ya que *max pooling* solo reduce dimensionalidad.
 - C. El gradiente se propaga de igual manera para todas las *inputs*.
 - D. El gradiente se propaga solo al valor de entrada que tenía el máximo valor durante el *forward pass*.
6. El volumen resultante de aplicar *max pooling* con pools de 2x2 y *stride* 2 sobre un volumen de tamaño 24x24x24 es (marca la respuesta correcta):
- A. 12x12x24.
 - B. 12x12x12.
 - C. 24x24x12.
 - D. 24x24x24.
 - E. 24x12x12.
 - F. 12x24x12.
7. El volumen resultante de aplicar una capa convolucional con 16 filtros de tamaño 2x2 y *stride* 1 sobre un volumen de tamaño 5x5x4 es (marca la respuesta correcta):
- A. 5x5x4.
 - B. 3x3x16.
 - C. 4x4x16.
 - D. 3x3x4.
 - E. 4x4x4.
 - F. Ninguna de las anteriores.

8. El volumen resultante de aplicar una capa convolucional con 16 filtros de tamaño 2×2 y *stride* 2 sobre un volumen de tamaño $5 \times 5 \times 4$ es (marca la respuesta correcta):
- A. $2,5 \times 2,5 \times 16$
 - B. $3 \times 3 \times 16$
 - C. $4 \times 4 \times 16$
 - D. $3 \times 3 \times 4$
 - E. $4 \times 4 \times 4$
 - F. Ninguna de las anteriores, las dimensiones no tienen sentido.
9. ¿Cuál de las siguientes son transformaciones válidas para *data augmentation*? (Marca todas las respuestas correctas)
- A. Rotar la imagen.
 - B. Obtener el recorte de una parte de la imagen. Por ejemplo: en una imagen con un gato, obtener la cara del gato.
 - C. Pasar una imagen a blanco y negro.
 - D. Utilizar un objeto similar en la misma pose. Por ejemplo, en una imagen de un gato, cambiar al gato por un perro en el mismo sitio.
 - E. Colorear la imagen con nuevos colores aleatorios.
10. *Transfer learning* (marca todas las respuestas correctas):
- A. Es apropiado cuando no disponemos de suficientes datos y tenemos un problema similar al de datasets como ImageNet.
 - B. Permite entrenar CNN complejas con relativamente pocos datos.
 - C. Aprovecha las representaciones obtenidas en otro problema rico en datos.