Sistemas Cognitivos Artificiales

Roberto Casado Vara

Tema 5: Convolutional Neural Networks (CNN)

Recuerda que…

- La actividad 1 tiene fecha de entrega 10 de mayo.
- [IMPORTANTE] Los que estéis sin grupo y no lo encontréis hare yo grupos de forma aleatoria.
- En la clase de hoy presentamos el trabajo en grupo.
- El trabajo en grupo se entrega el 24 de mayo.

Convolutional Neural Networks

- Semana 7
  - Introducción y casos de uso
  - Convolution layers
- *Semana 8*
  - Arquitecturas CNN para problemas de visión por computador
  - *Data augmentation*
  - *Transfer Learning*

Sistemas Cognitivos Artificiales

Roberto Casado Vara

Tema 5.1: Introducción y casos de uso

Un poco de historia

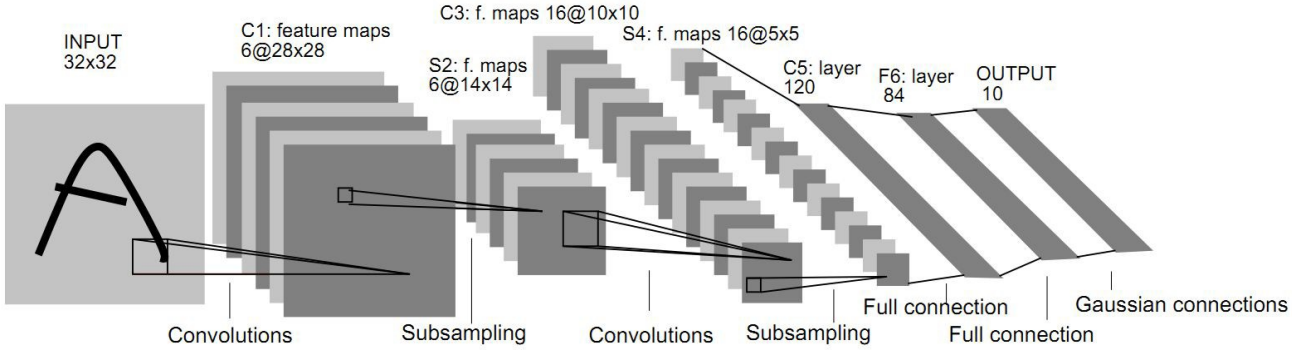
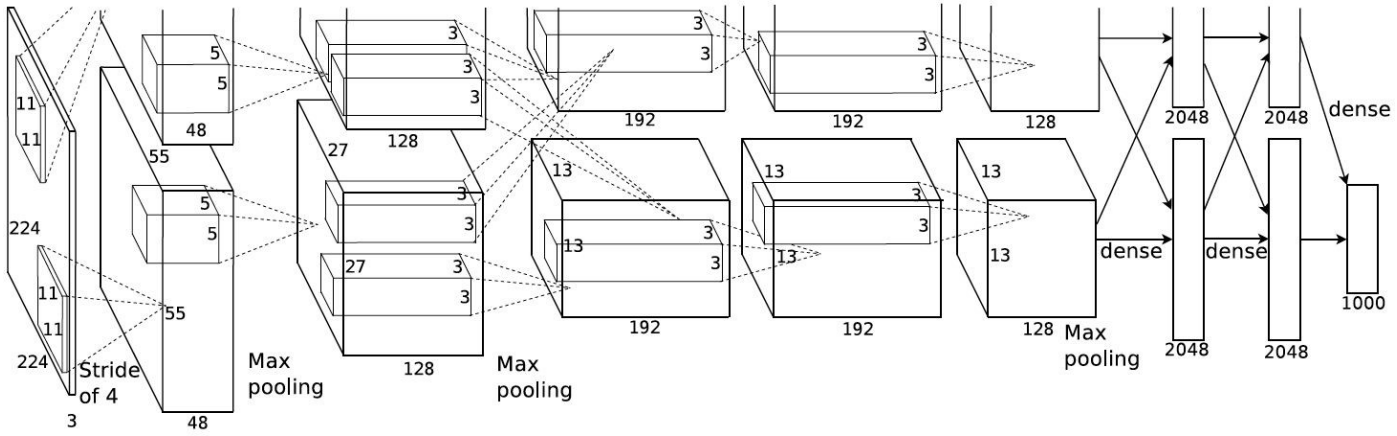


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

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

- Yann LeCun (1998) introduce el primer caso práctico de una red convolucional (*LeNet-5*) entrenada mediante gradient descent para la clasificación de dígitos para el servicio postal.

Un poco de historia



Fuente de la imagen: ImageNet Classification with Deep Convolutional Neural Networks (Krizhevsky et al., 2012)

- La verdadera explosión se produce a partir de 2012, cuando una CNN profunda (**AlexNet**) gana la competición de clasificación de imágenes ImageNet con un gran margen.

- Esta arquitectura utilizaba una estrategia de entrenamiento muy efectiva, usando elementos vistos en temas anteriores, así como GPUs para una mayor velocidad de entrenamiento.

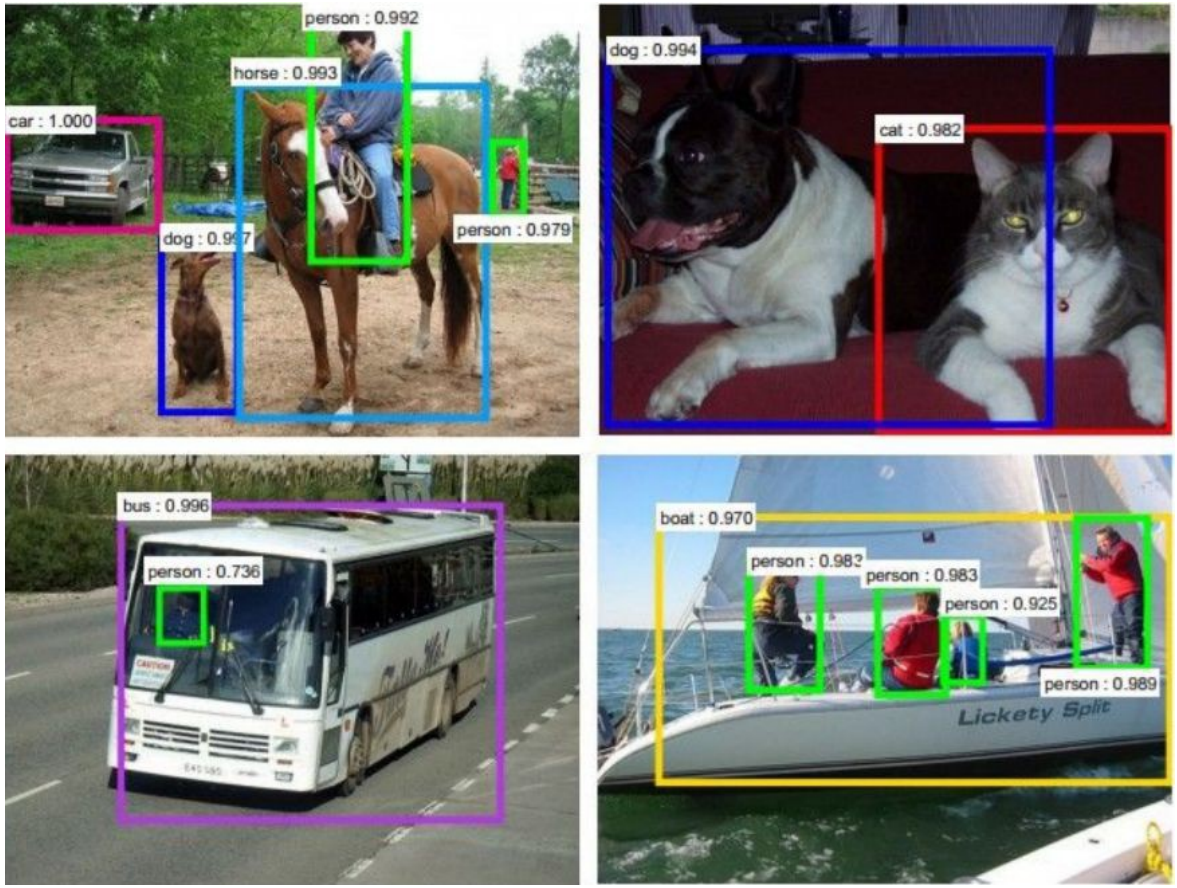
Clasificación de imágenes

Fuente de la imagen: ImageNet Classification with Deep Convolutional Neural Networks (Krizhevsky et al., 2012)

Retrieval de imágenes similares

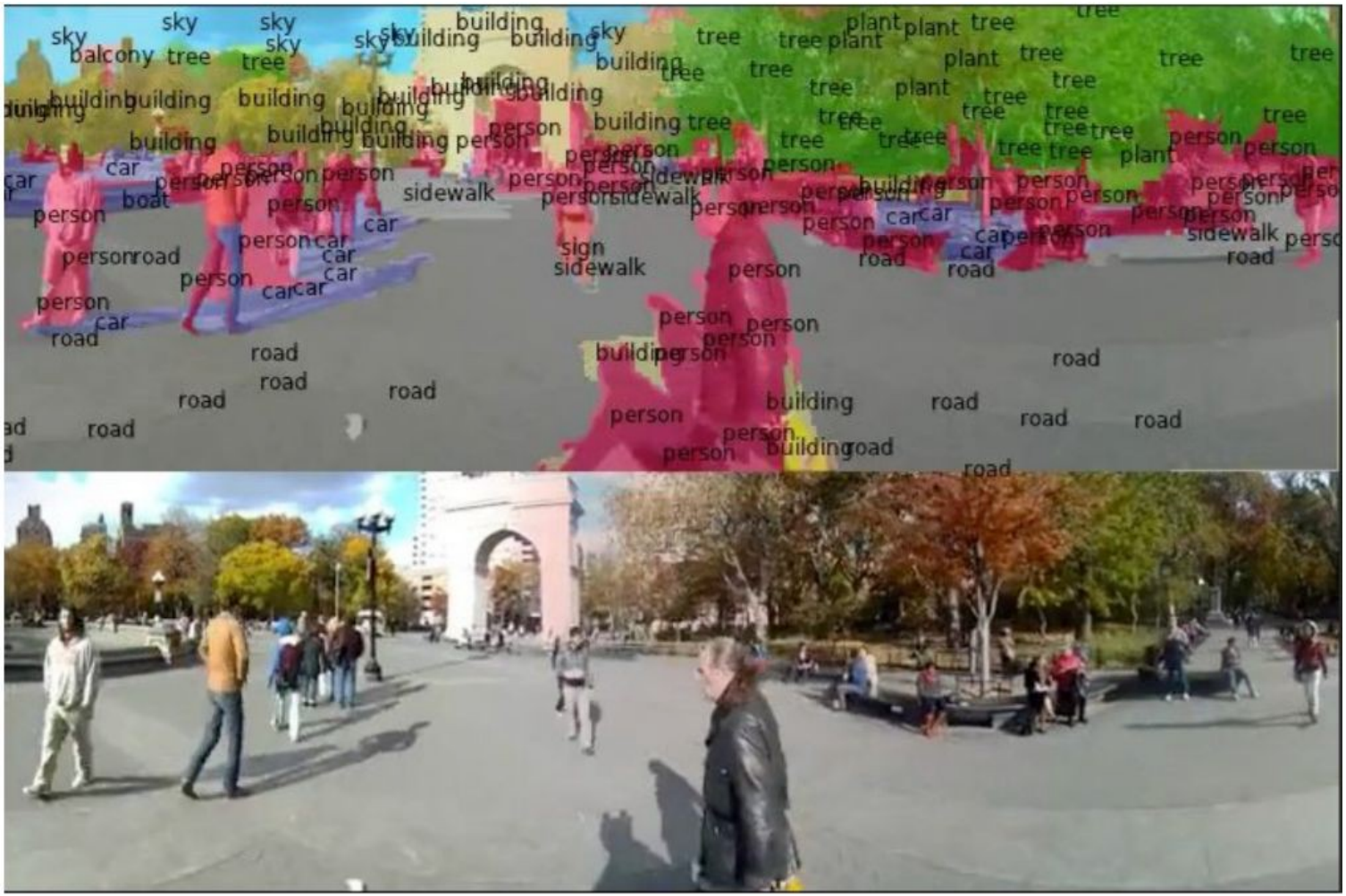
Fuente de la imagen: ImageNet Classification with Deep Convolutional Neural Networks (Krizhevsky et al., 2012)

Object detection



Fuente de la imagen: Faster R-CNN (Ren et al. 2015)

Image segmentation



Fuente de la imagen: Farabet et al (2012): Learning Hierarchical Features for Scene Labeling

Aplicaciones

- En general, cualquier problema con imágenes o vídeos es un buen candidato a ser tratado con redes convolucionales.
- Más aplicaciones en clases magistrales.

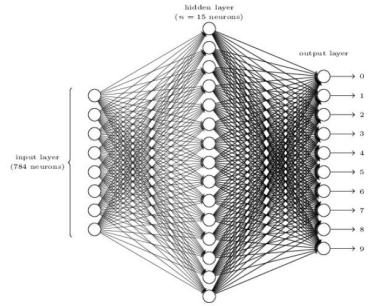
Sistemas Cognitivos Artificiales

Roberto Casado Vara

Tema 5.2: Convolution layers

Imagen con fully connected layers

- En el problema de MNIST, vimos cómo una imagen de 28x28 píxeles en blanco y negro puede representarse como un vector de 748 elementos, que se correspondía con los elementos de la *input layer* de nuestra red.



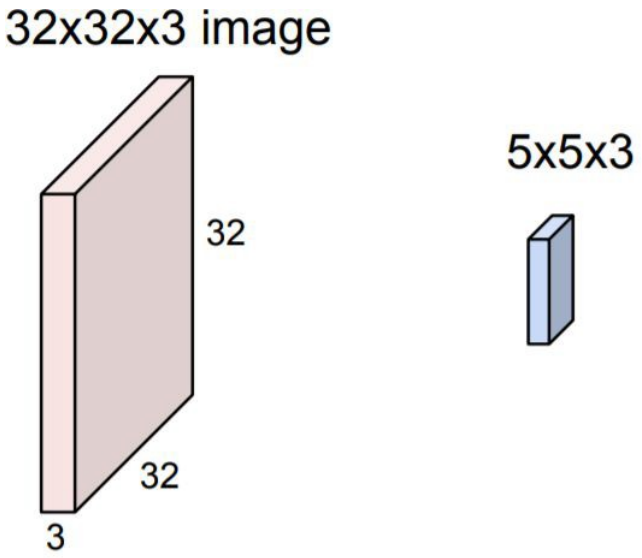
- Sin embargo, las imágenes suelen ser más grandes y además, cuando son en color, tienen 3 canales. Para una imagen de 300x300px en color tendríamos una input layer de 300·300·3 = 270.000 elementos.

- Por cada neurona de la segunda capa tenemos 270.000 parámetros.

Imagen con fully connected layers

- Este gran número de parámetros empieza a ser un problema desde un punto de vista computacional.
- Adicionalmente, puede influir en que la red caiga en *overfitting*, al tener tantos parámetros por píxel. Esto puede ayudar a la red a memorizar imágenes.
- Las capas convolucionales solucionan este problema aprovechando la estructura espacial de la imagen.

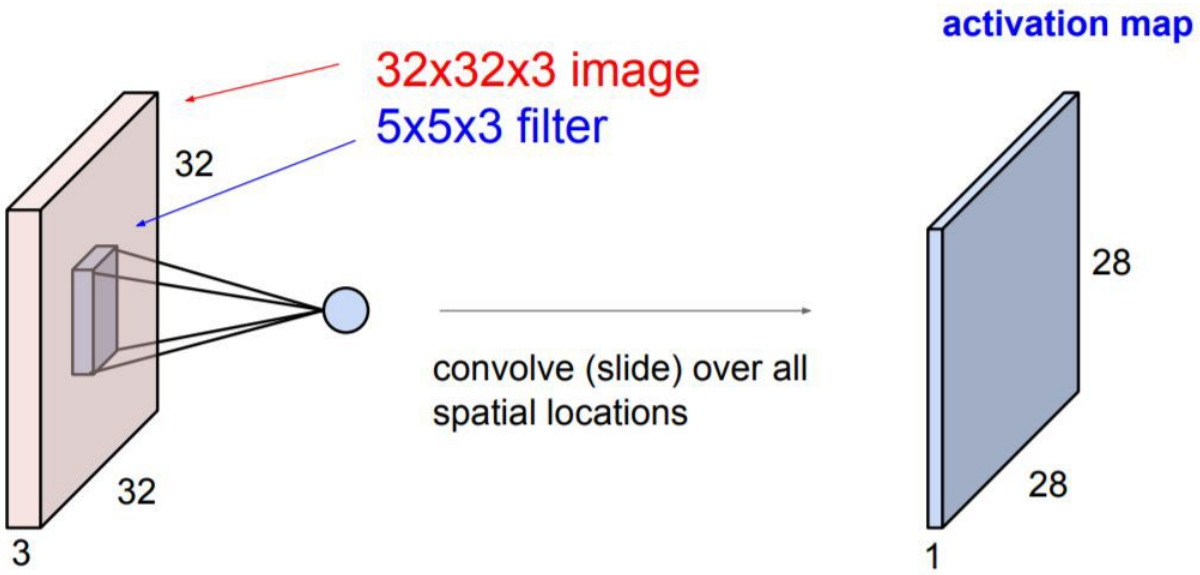
Convolution layers



- Las capas convolucionales utilizan **filtros** de pequeño tamaño en tres dimensiones.
- Los filtros recorren el volumen de entrada (la imagen, si estamos en la primera capa) produciendo **un valor de salida por cada posición**.
- Los filtros tienen la **misma profundidad** que el volumen de entrada.

Fuente de la imagen: Stanford CS231n

Convolution layers

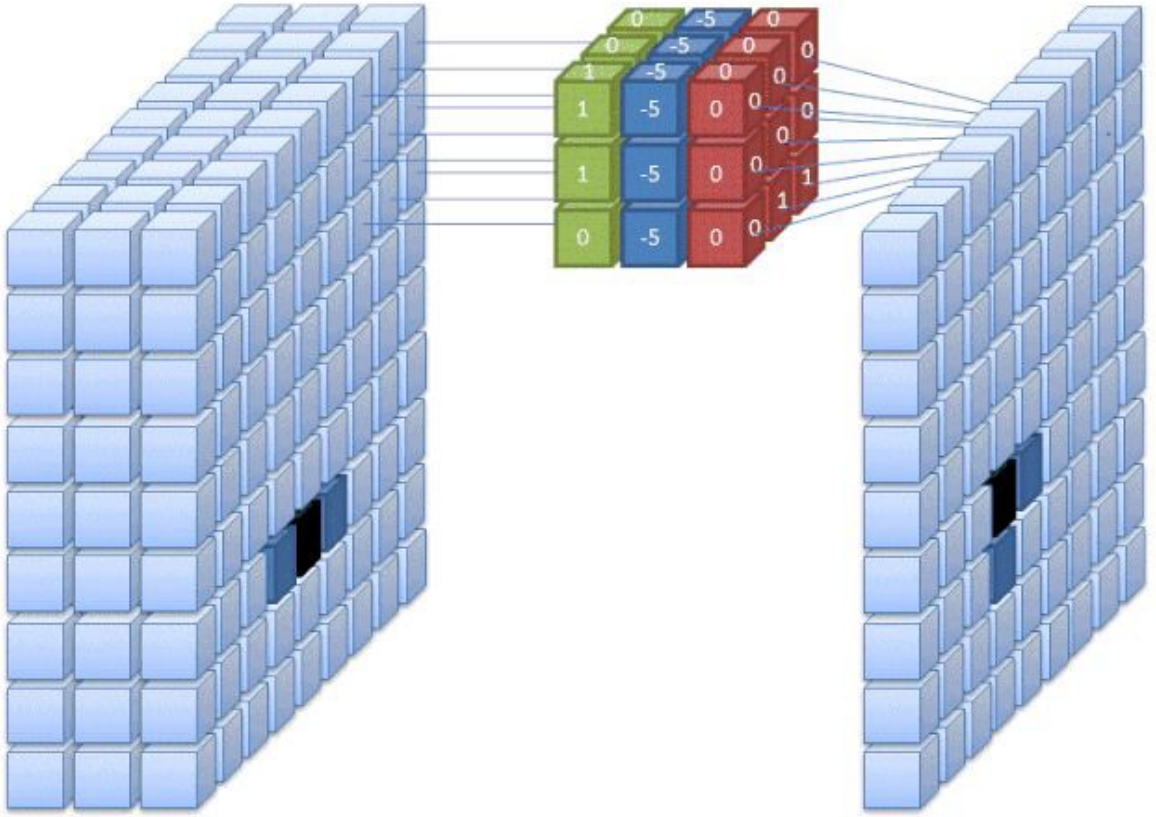


- El filtro tiene unos pesos ***w*** y un bias ***b***. Para un filtro de tamaño 5x5x3, tenemos un total de 75 *weights* y un bias.

- El resultado de aplicar el filtro en una posición de la imagen viene dado por nuestra conocida fórmula ***w·x + b***. Estamos multiplicando cada elemento del volumen por el correspondiente elemento del filtro, sumando esos productos y finalmente sumando el bias para obtener la salida.

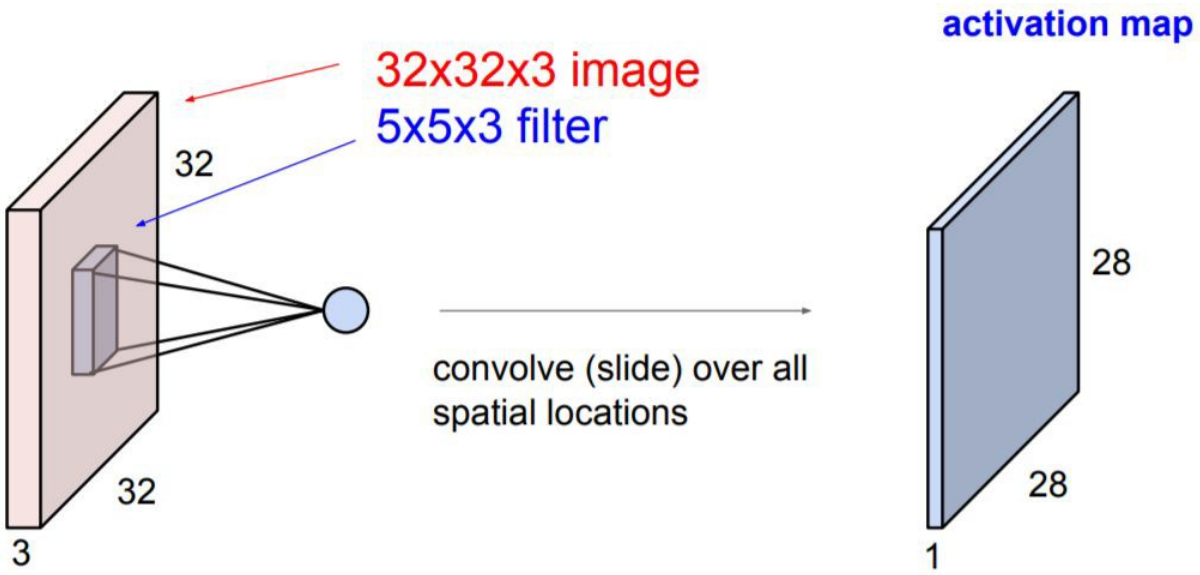
Fuente de la imagen: Stanford CS231n

Convolutional Network with Feature Layers



Fuente de la imagen: Wikimedia Commons

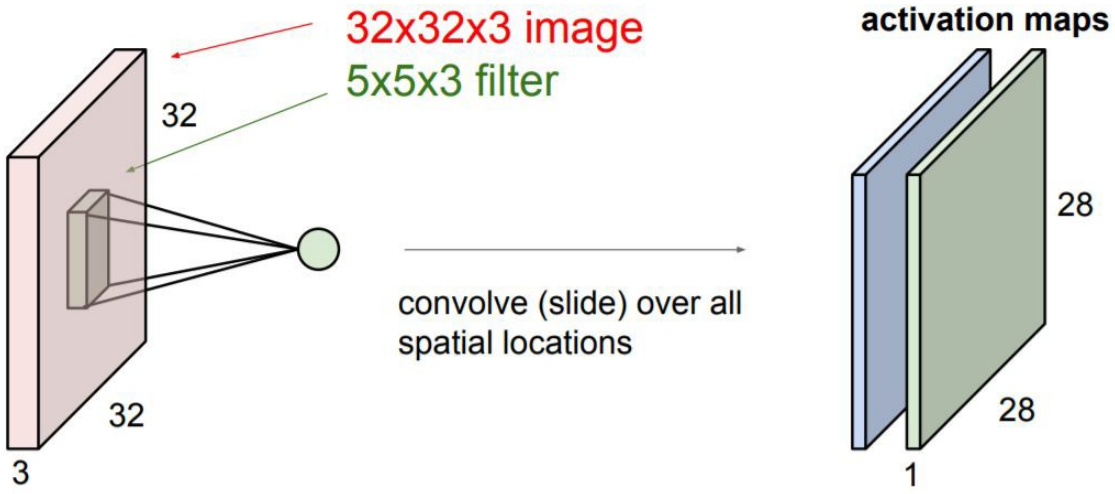
Convolution layers



- Al desplazar el filtro por toda la imagen, obtenemos un mapa de salidas conocido como **activation map**.
- El tamaño de éste es menor que el de la imagen original (salvo que apliquemos *zero-padding*).

Fuente de la imagen: Stanford CS231n

Convolution layers

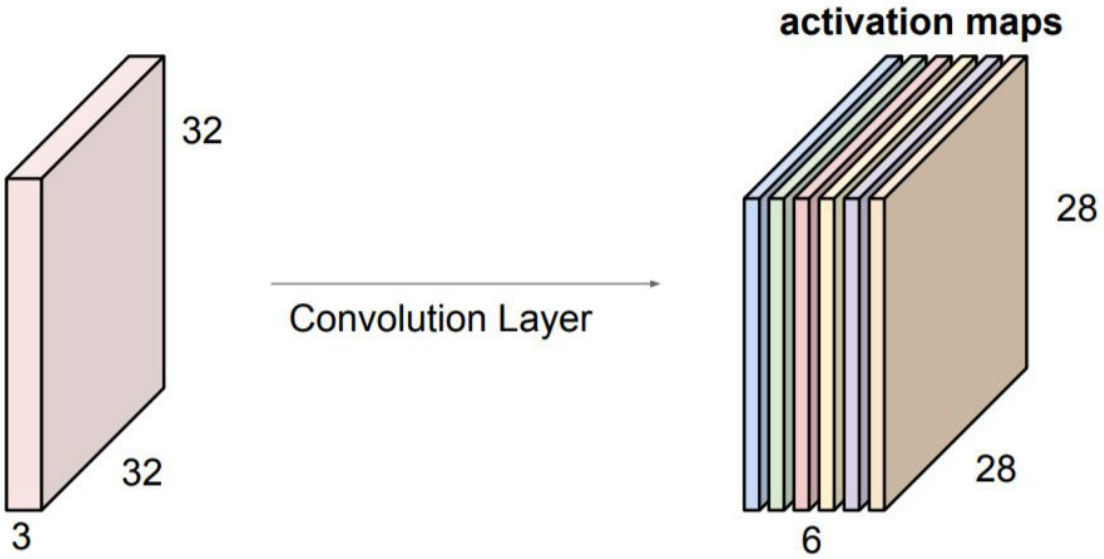


- Normalmente, se aplican varios filtros para obtener más ***features*** en cada posición de la imagen, por lo que obtenemos nuevos **activation maps**.

- La idea es que cada uno de estos filtros obtenga ciertas características de la imagen para obtener una buena representación de la misma. Por cada grupo de píxeles la red se "fija" en diferentes detalles y extrae diferentes valores.

Fuente de la imagen: Stanford CS231n

Convolution layers

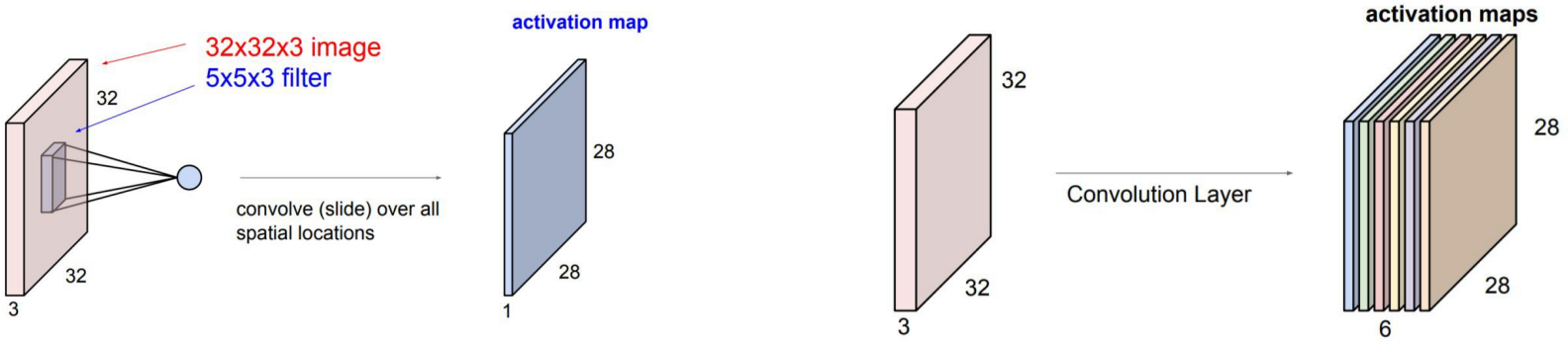


- El resultado de una convolution layer es una nueva representación en tres dimensiones de la imagen, conservando las características espaciales de ésta y añadiendo más profundidad con las features calculadas a partir de los filtros.

- En general, el input de una convolution layer es un volumen en 3D y su output es otro volumen en 3D.

Fuente de la imagen: Stanford CS231n

Convolution layers

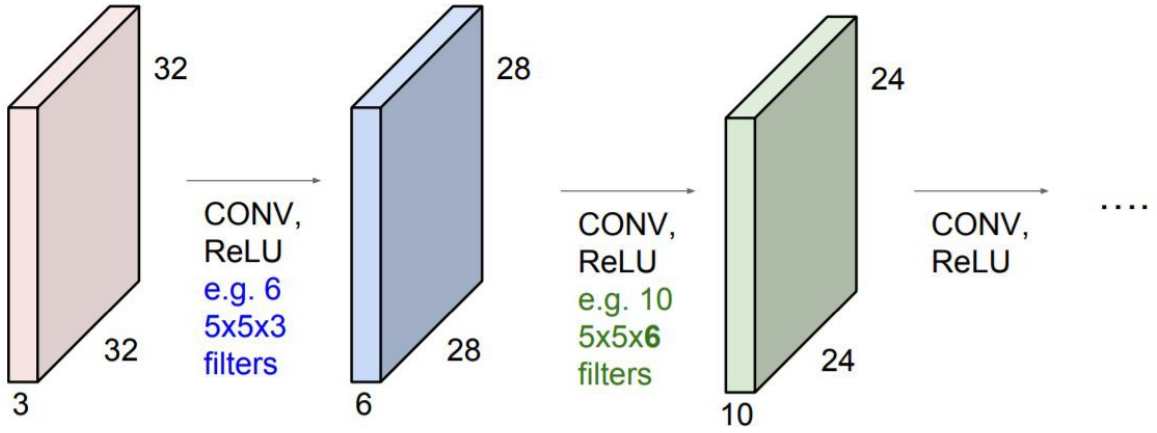


- Número de parámetros para una convolution layer con **10 filtros** de tamaño **5x5**, con una imagen de entrada de tamaño **32x32x3**.
    1. Los filtros han de tener la misma profundidad que el volumen de entrada: **5x5x3**.
    2. Por tanto, cada filtro tiene 5x5x3 = 75 pesos + 1 bias. Un total de **76 parámetros**.
    3. En total, al tener 10 filtros en la capa, obtenemos un total de **760 parámetros**.

- ¡Independiente del tamaño de la imagen!

Fuente de la imagen: Stanford CS231n

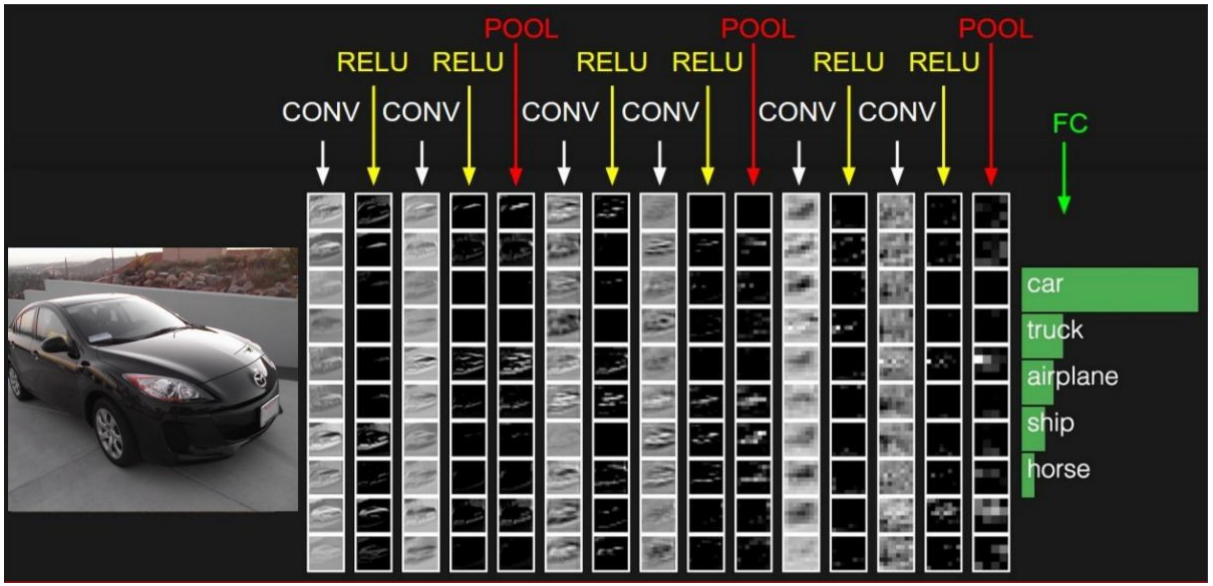
Convolutional Neural Networks



- Como hemos visto, en una *convolution layer* el input es un volumen en 3D y su output es otro volumen en 3D.
- Una **convolutional neural network** (**CNN**) se compone de una serie de *convolution layers* aplicadas una detrás de otra.
- Es muy común aplicar unidades de activación **ReLU** por cada valor de salida de la convolution layer. Esto es, aplicamos la función de activación sobre el valor *wx + b* resultante de aplicar cada filtro.

Fuente de la imagen: Stanford CS231n

Convolutional Neural Networks

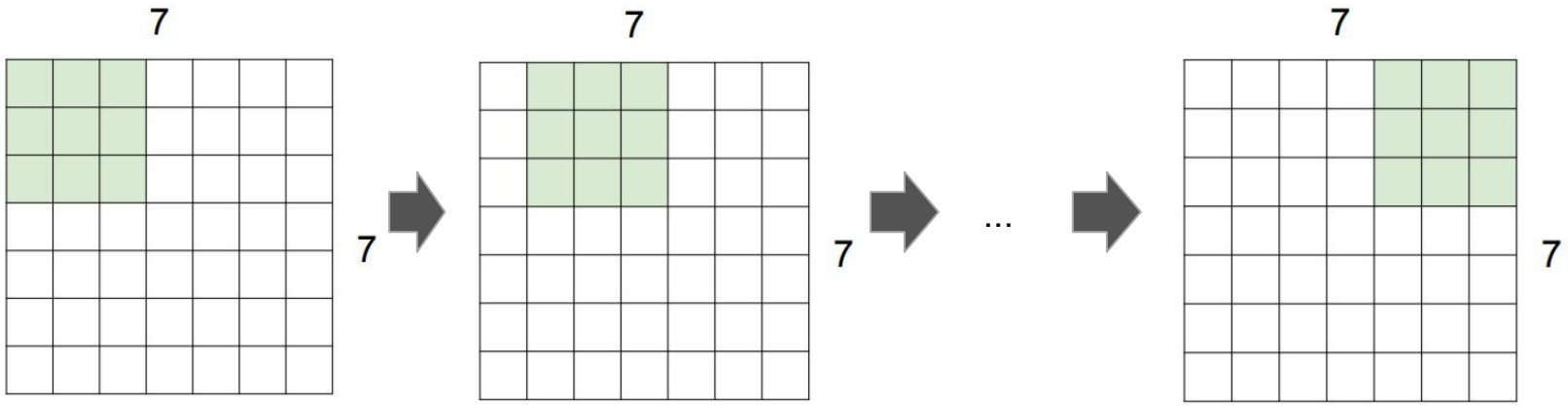


- Las capas van aprendiendo una **representación jerárquica** de las *features* de la imagen, con las primeras capas reconociendo elementos más simples de una imagen y las últimas obteniendo representaciones de más alto nivel a partir de estos elementos simples.

Fuente de la imagen: Stanford CS231n

Dimensiones espaciales

- **Stride**: hiperparámetro que controla cómo se desplazan los filtros a través de la imagen.
- Ejemplo: Input de 7x7x1, filtro de tamaño 3x3. Stride 1

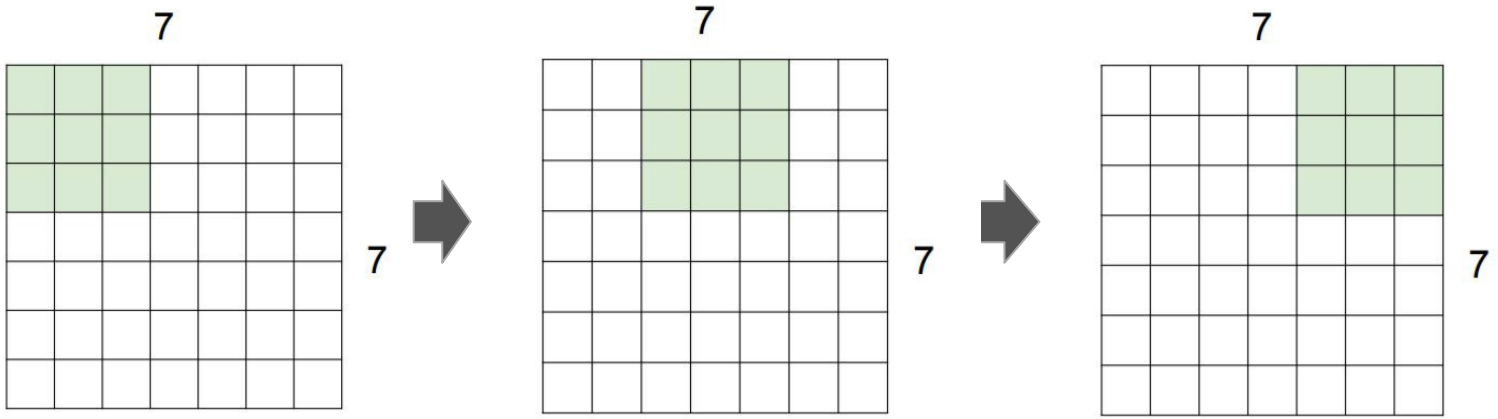


- Output: 5x5x1

Fuente de la imagen: Stanford CS231n

Dimensiones espaciales

- **Stride**: hiperparámetro que controla cómo se desplazan los filtros a través de la imagen.
- Ejemplo: Input de 7x7x1, filtro de tamaño 3x3. **Stride 2**

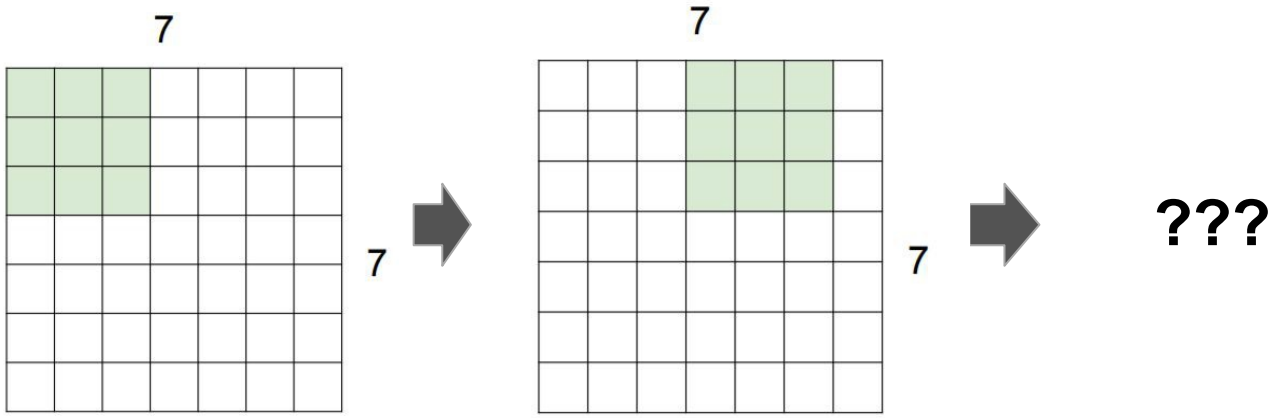


- Output: 3x3x1
- Con strides mayores, la red se fija en áreas más distantes de la imagen y obtenemos una reducción de la dimensionalidad.

Fuente de la imagen: Stanford CS231n

Dimensiones espaciales

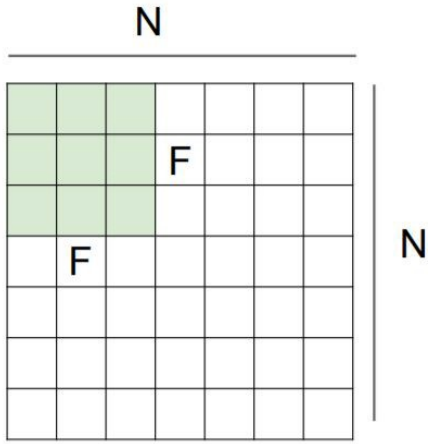
- **Stride**: hiperparámetro que controla cómo se desplazan los filtros a través de la imagen.
- Ejemplo: Input de 7x7x1, filtro de tamaño 3x3. **Stride 3?**



- No encaja

Fuente de la imagen: Stanford CS231n

Dimensiones espaciales



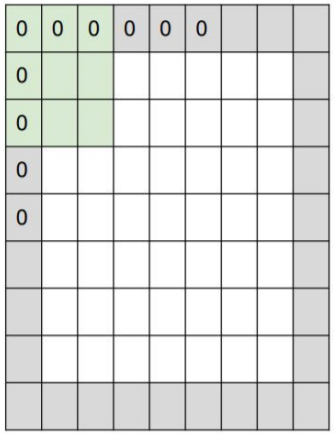
- N tamaño del input para una imagen NxN
- F tamaño del filtro
- S tamaño del stride

El tamaño resultante por cada lado es:

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

Fuente de la imagen: Stanford CS231n

Zero padding



- Es frecuente añadir un “marco” de ceros a nuestro *input* (puede ser más de uno). Esto se conoce como **zero padding**.
- Esto permite hacer que las dimensiones cuadren y/o mantener las mismas dimensiones espaciales.
- Para un marco de P ceros, nuestra fórmula cambia:

$$[(N - F + 2P) / S] + 1$$

Fuente de la imagen: Stanford CS231n

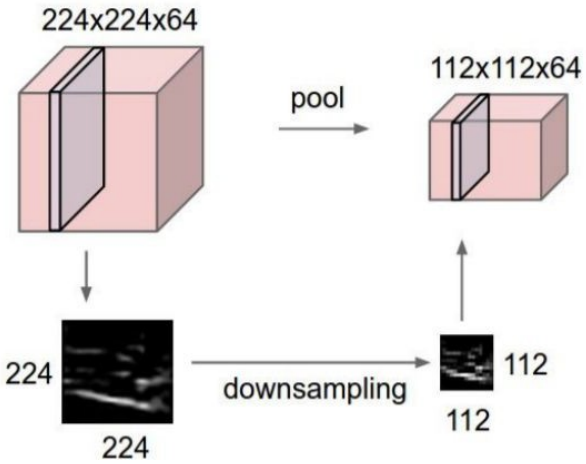
Dimensiones espaciales generales

En general, una capa convolucional con input un volumen de tamaño W_1 x H_1 x D_1:

- Tiene cuatro hiperparámetros:
  - **Número de filtros K** (comúnmente potencias de 2: 8, 16, 32, 64, etc)
  - **Tamaño de filtro F**
  - **Stride S** (mismo valor horizontal y verticalmente)
  - **Cantidad de zero-padding P**
- Produce un nuevo volumen W_2 x H_2 x D_2 donde
  - **$W_2 = (W_1 - F + 2P)/S + 1$**
  - **$H_2 = (H_1 - F + 2P)/S + 1$**
  - **$D_2 = K$**
- Introduce $F \cdot F \cdot D_1$ pesos por filtro, para un total de **$(F \cdot F \cdot D_1) \cdot K$** pesos y **K biases**.

Fuente de la imagen: Stanford CS231n

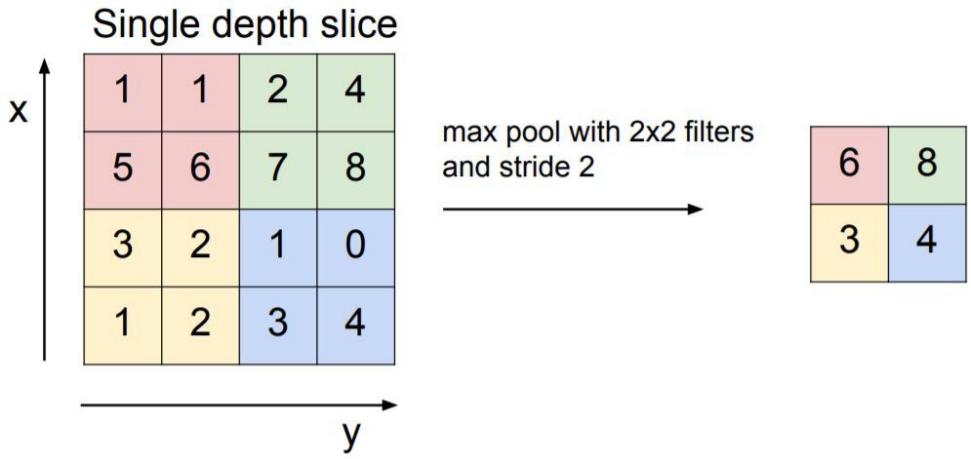
Capas Max Pooling



- Reducen el tamaño de las representaciones obtenidas, de manera que estas son más pequeñas y manejables computacionalmente.
- Se reduce el tamaño del volumen mediante la toma de máximos sobre cada nivel de profundidad. **La profundidad no varía**, por lo que tenemos el mismo número de *features*.
- Las capas *max pooling* se suelen alternar con las *convolution layers* en una CNN.

Fuente de la imagen: Stanford CS231n

Capas Max Pooling



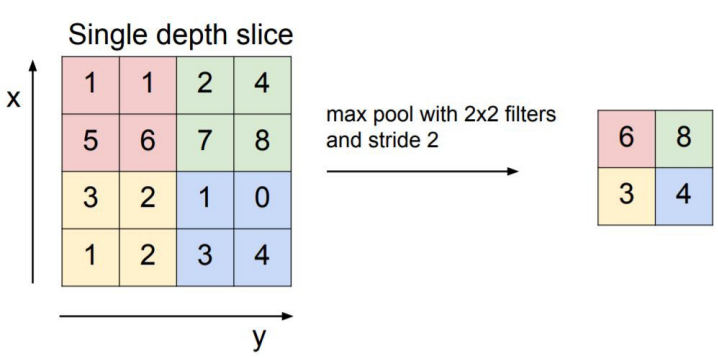
- Una capa *max pooling* aplica filtros de manera similar a una convolution layer, pero actuando sólo sobre un nivel de profundidad. La salida es el máximo valor de los inputs.

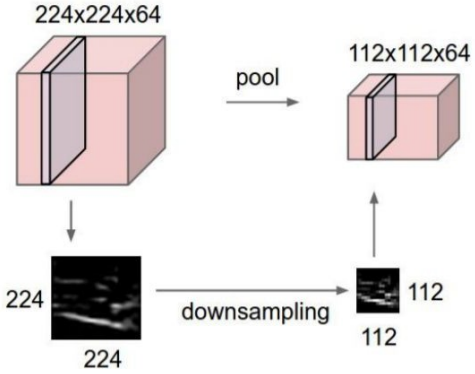
- Lo más común es aplicar filtros de tamaño 2x2 con stride 2. De este modo, el tamaño se reduce en un 75%.

- Las capas *max pooling* no tienen parámetros: su resultado consiste en tomar los máximos encontrados en el volumen de entrada.

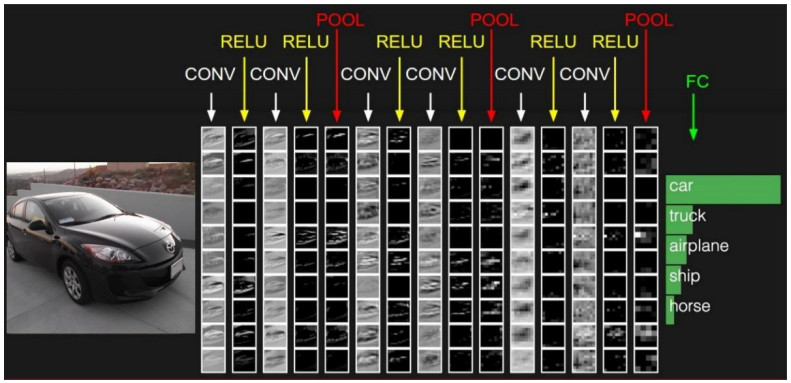
Fuente de la imagen: Stanford CS231n

Capas Max Pooling





- Max pooling selecciona las activaciones más importantes en cada zona. Otras estrategias, como tomar una media de valores, no son tan efectivas en la práctica.



Fuente de la imagen: Stanford CS231n

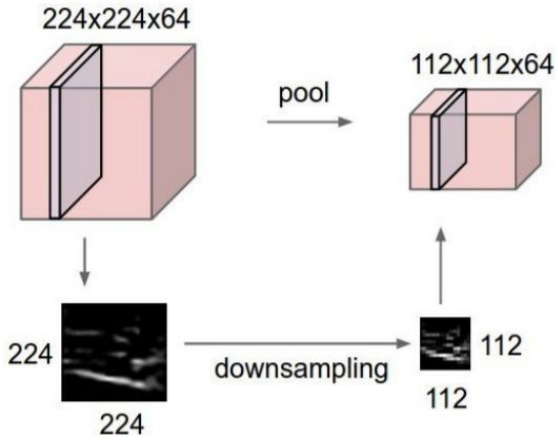
Dimensiones espaciales generales

En general, una capa max pooling aplicada a un volumen de tamaño W_1 x H_1 x D_1:

- Tiene dos hiperparámetros:
  - **Tamaño de filtro o pool F**
  - **Stride S** (mismo valor horizontal y verticalmente)

- Produce un nuevo volumen W_2 x H_2 x D_2 donde
  - $\mathbf{W_2 = (W_1 - F)/S + 1}$
  - $\mathbf{H_2 = (H_1 - F)/S + 1}$
  - $\mathbf{D_2 = D_1}$

- No introduce parámetros.

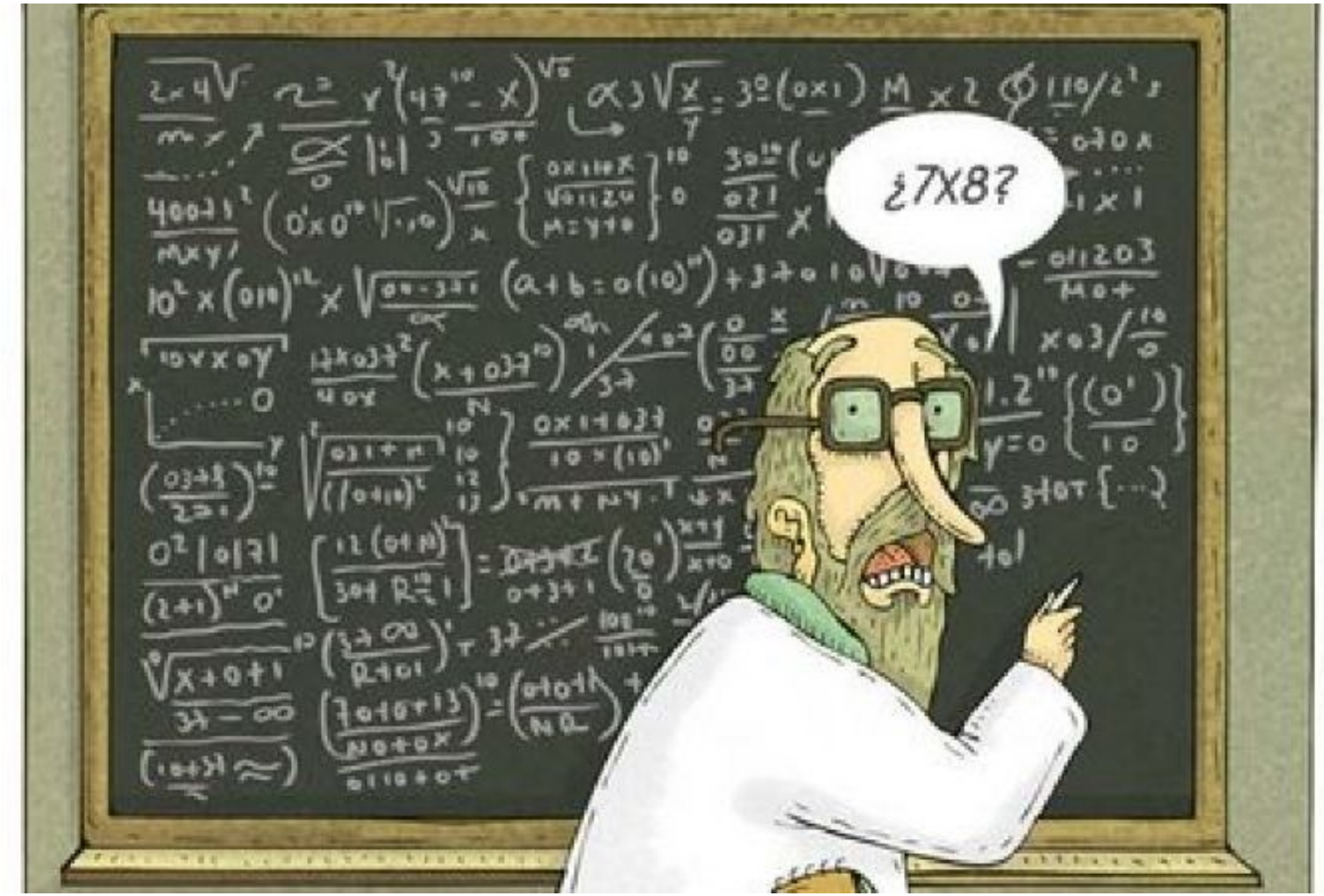


Fuente de la imagen: Stanford CS231n

Trabajo en grupo

- Ahora voy a presentar el trabajo en grupo para que podáis entender lo que estoy pidiendo y solucionar las posibles dudas.

¿Dudas?

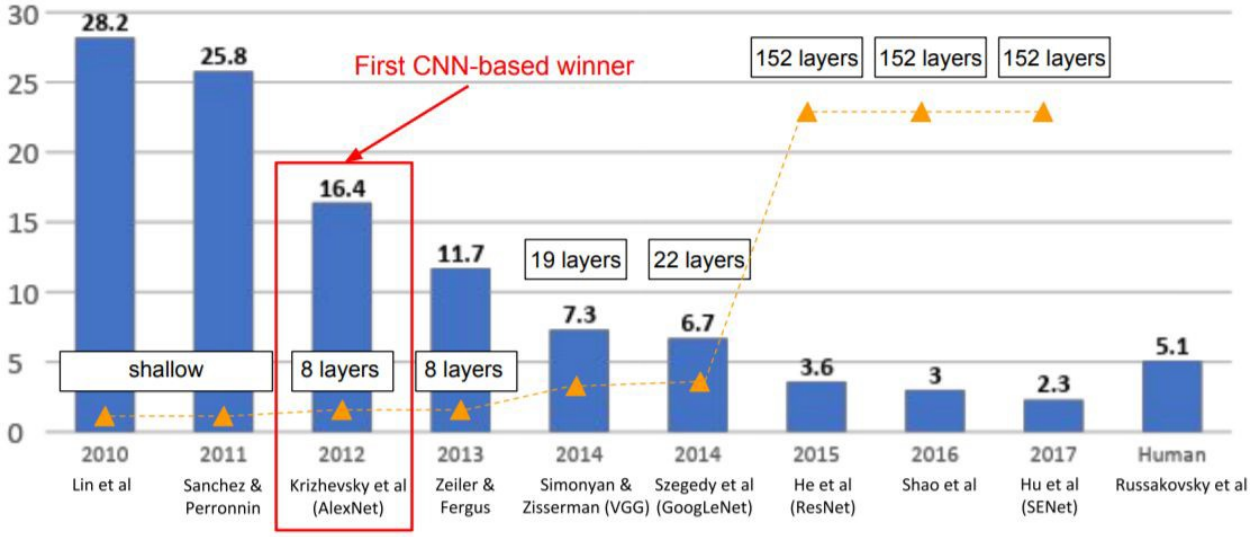


Sistemas Cognitivos Artificiales

Roberto Casado Vara

Tema 5.3: Arquitecturas CNN para problemas de visión por computador

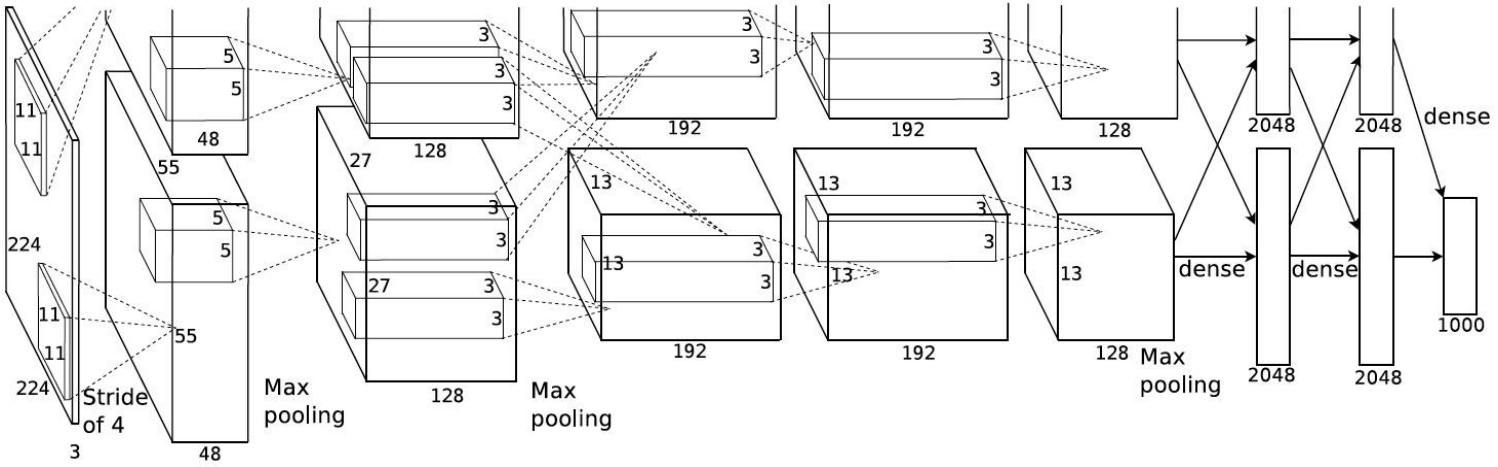
Evolución



- Resultados de la competición ILSVRC (*ImageNet Large Scale Visual Recognition Challenge*). Utiliza el dataset ImageNet con imágenes de tamaño 227x227 y 1000 posibles clases.
- Tendencia hacia redes cada vez más profundas.
- En esta clase: AlexNet y VGG.

Fuente de la imagen: Stanford CS231n

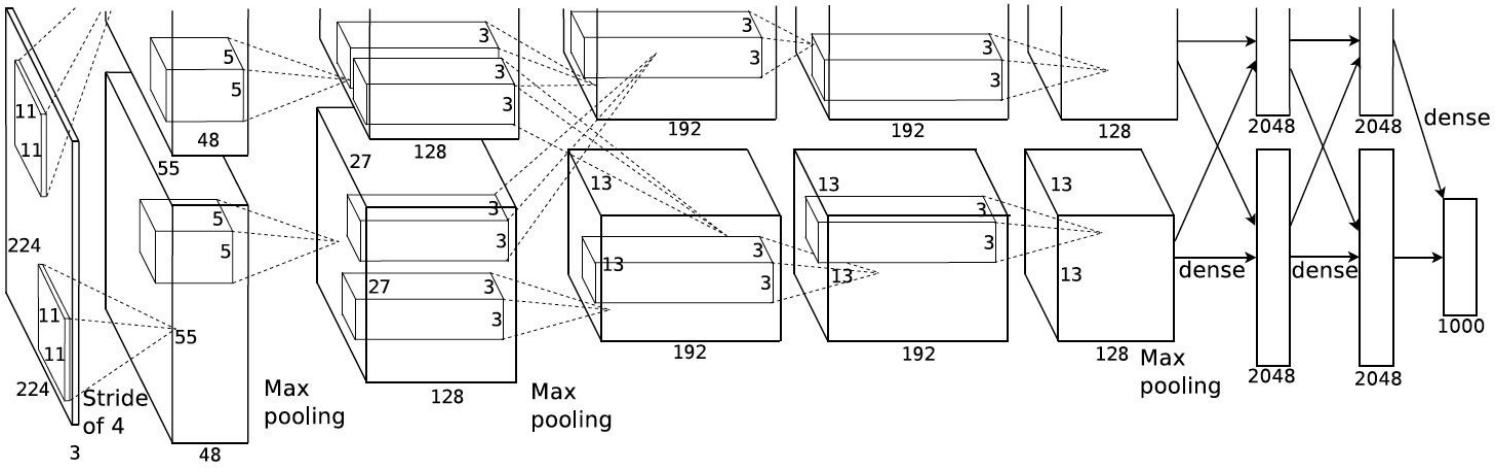
AlexNet



- **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, class scores.

Fuente de la imagen: ImageNet Classification with Deep Convolutional Neural Networks

AlexNet

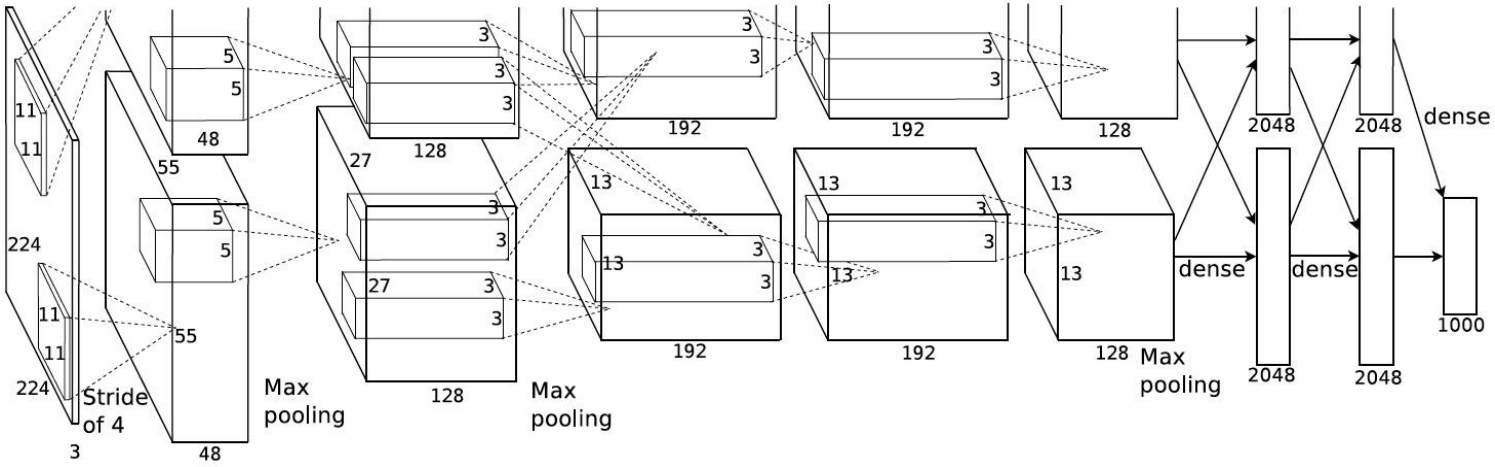


- Vemos cómo la profundidad de los volúmenes va creciendo, a la vez que el tamaño espacial se reduce mediante max pooling. Queremos que la red obtenga representaciones más complejas, donde los elementos de la imagen se van componiendo de manera jerárquica.

- Primera red en utilizar la unidad de activación ReLU.

Fuente de la imagen: ImageNet Classification with Deep Convolutional Neural Networks

AlexNet



- 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.

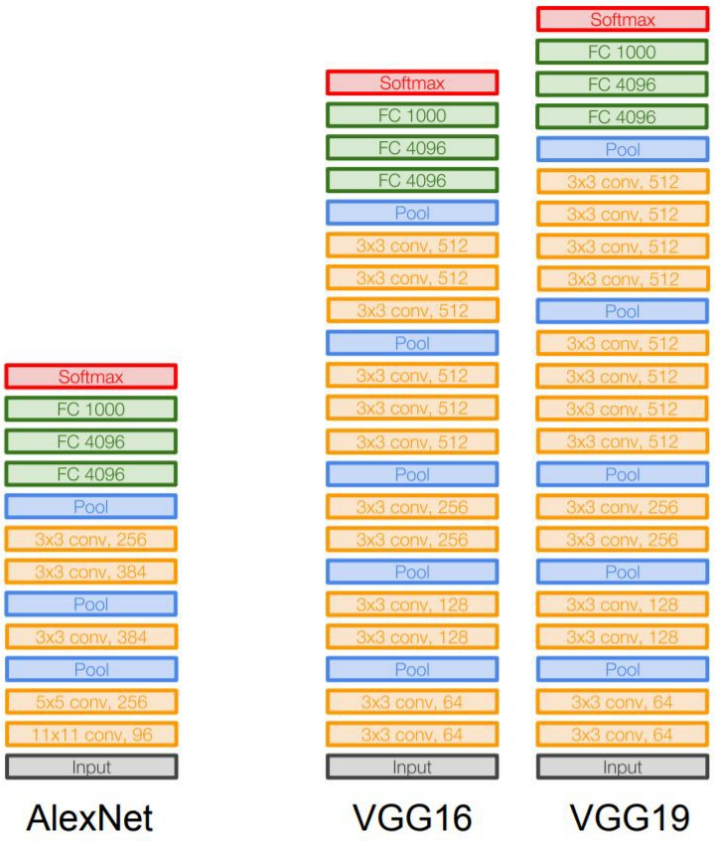
Fuente de la imagen: ImageNet Classification with Deep Convolutional Neural Networks

Dimensiones de AlexNet

- **Ejercicio**: Con lo visto en el tema, deducir las dimensiones resultantes de cada capa (las capas NORM no afectan la dimensión).

[227x227x3] INPUT
[55x55x96] **CONV1**: 96 11x11 filters at stride 4, pad 0
[27x27x96] **MAX POOL1**: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] **CONV2**: 256 5x5 filters at stride 1, pad 2
[13x13x256] **MAX POOL2**: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] **CONV3**: 384 3x3 filters at stride 1, pad 1
[13x13x384] **CONV4**: 384 3x3 filters at stride 1, pad 1
[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1
[6x6x256] **MAX POOL3**: 3x3 filters at stride 2
[4096] **FC6**: 4096 neurons
[4096] **FC7**: 4096 neurons
[1000] **FC8**: 1000 neurons (class scores)

VGG



- Mayor profundidad, filtros más pequeños (3x3, stride 1, pad 1)
- Max Pooling: siempre 2x2 con stride 2.
- Aproximadamente 140M de parámetros vs 60M de AlexNet.

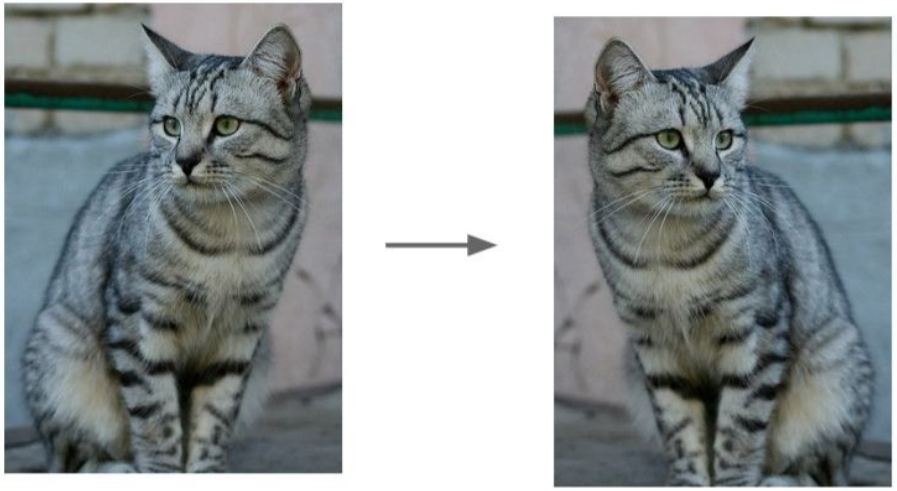
Fuente de la imagen: Stanford CS231N

Sistemas Cognitivos Artificiales

Roberto Casado Vara

Tema 5.4: Data Augmentation

Data Augmentation

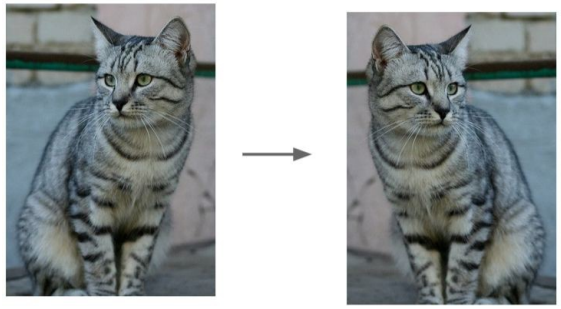


- Muy común en problemas de visión por computador.
- Consiste en realizar transformaciones o pequeñas perturbaciones aleatorias de una imagen de manera que obtenemos nuevas imágenes con las que entrenar, pero para las que el concepto a tratar o la clase final no cambia.
- Permite entrenar modelos con éxito utilizando menos imágenes.

Fuente de la imagen: Stanford CS231n

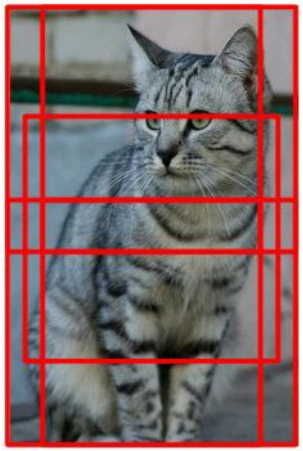
Data Augmentation - Ejemplos

Rotación



Cambios de contraste y brillo

Recortes

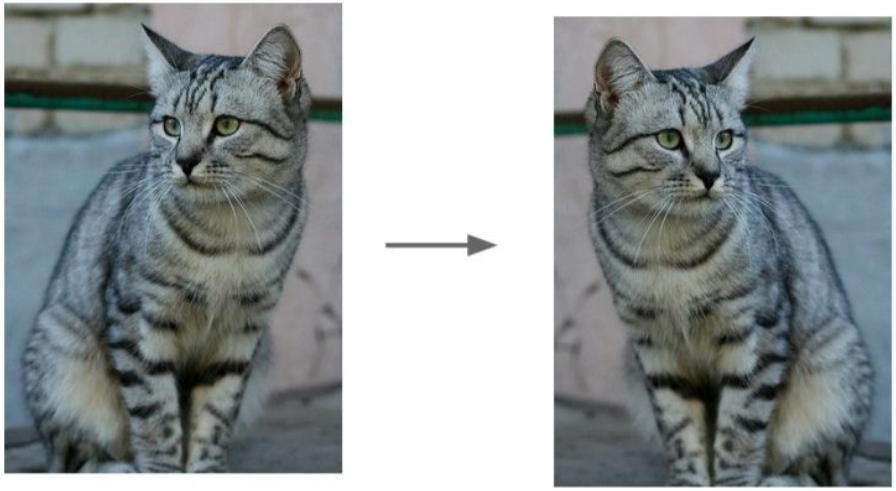


Posibles estrategias:

- Translaciones
- Rotaciones
- Recortes
- Cambios de brillo / contraste
- ...

Fuente de la imagen: Stanford CS231n

Data Augmentation - Regularización



- 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.

Fuente de la imagen: Stanford CS231n

Sistemas Cognitivos Artificiales

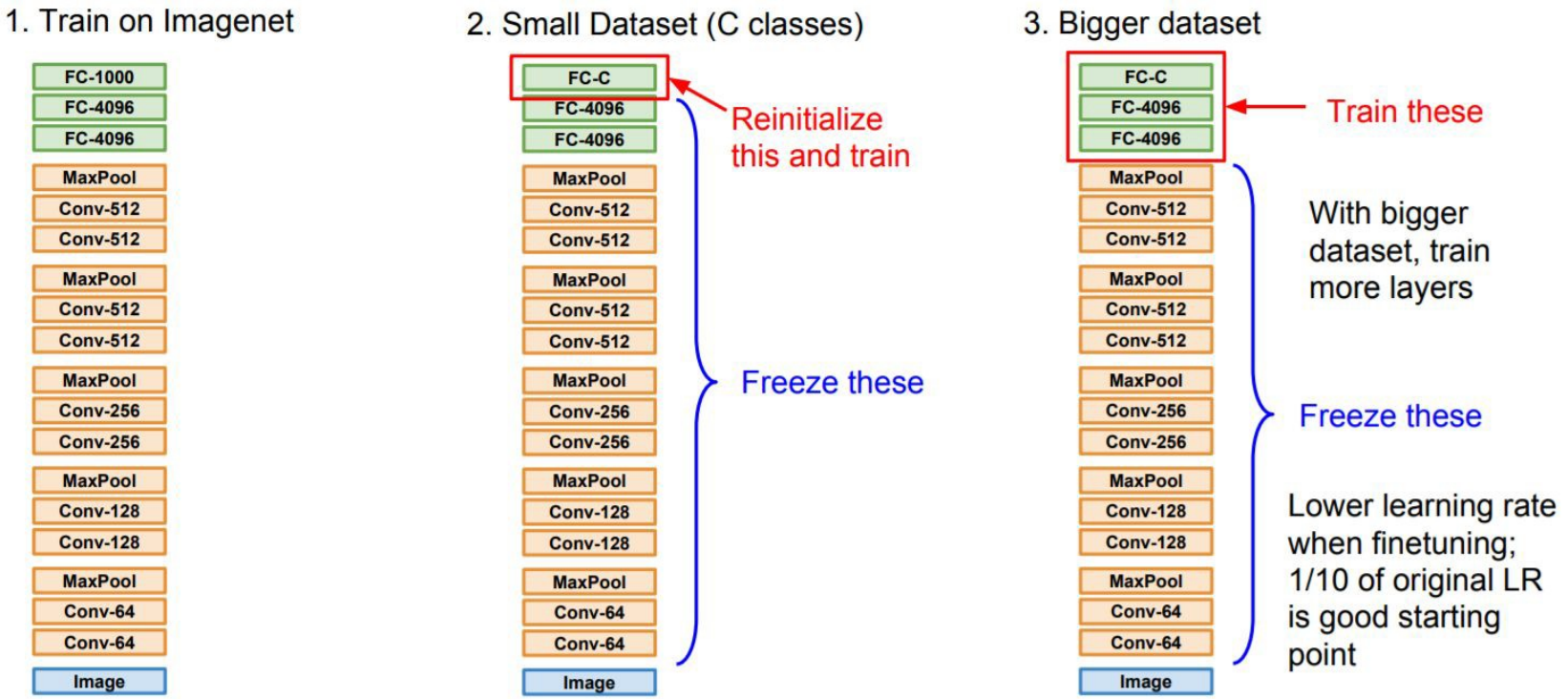
Roberto Casado Vara

Tema 5.5: Transfer Learning

Transfer Learning

- Las arquitecturas CNN que hemos visto son modelos complejos que necesitan una gran cantidad de datos para ser entrenadas con éxito.
- El dataset **ImageNet** tiene más de 14 millones de imágenes para un total de 1000 clases distintas.
- Es muy costoso hacerse con tal cantidad de datos.
- 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 es la siguiente:
  - Se parte de un modelo estándar ya entrenado (por ejemplo, VGG entrenado con ImageNet).
  - Utilizamos la misma red con nuestro training data, reinicializando y entrenando sólo las últimas capas de la red.

Transfer Learning



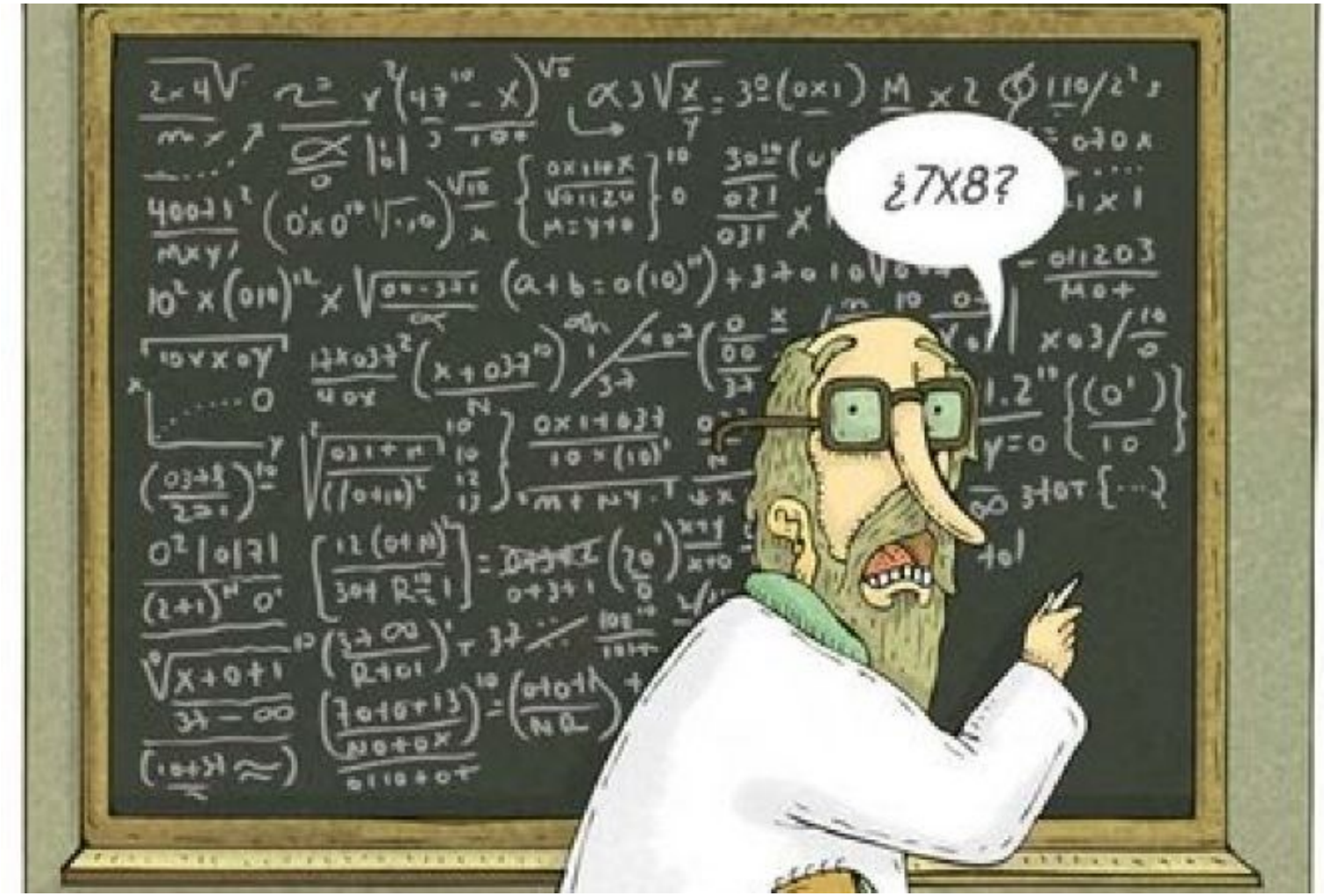
- El aprendizaje que se hizo sobre el problema original se transfiere al nuevo problema.

Fuente de la imagen: Stanford CS231n

Transfer Learning

- Transfer learning es muy común en el mundo del deep learning.
- No hay una fórmula exacta a la hora de aplicarlo. Según la cantidad de datos que dispongamos, podemos reinicializar y entrenar un mayor número de capas del modelo original.
- 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.

¿Dudas?



www.unir.net