UT5. Convolutional Neural Networks (CNN).

IES Abastos

CE Inteligencia Artificial y Big Data
Sistemas de Aprendizaje Automático.

Bloques de la unidad:

1. Introducción y casos de uso.
2. Convolution layers.
3. Dimensiones espaciales.
4. Max Pooling.
5. Ejercicios de dimensiones.
6. Arquitecturas CNN de visión por computador.
7. Data augmentation.
8. Transfer Learning.
9. Laboratorio 1.
10. Laboratorio 2.

1. Introducción y casos de usos.

Hemos visto cómo se comportan los hiperparámetros en el entretenimiento de una red neuronal artificial.

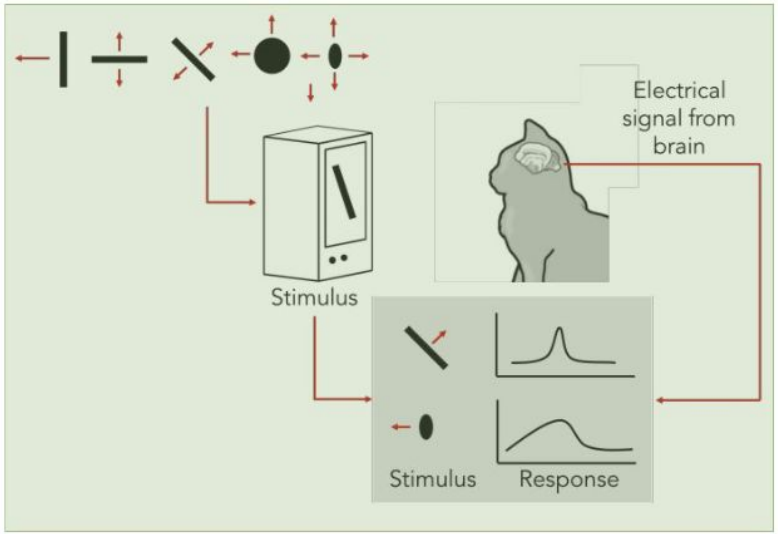
En este tema vamos a estudiar el entrenamiento en un tipo de red neuronal especializada en la visión por computador.

¿ En qué campos es aplicable la visión computador ?

1. Introducción y casos de usos.

Hubel y Wiesel (1959). Experimento que mide estímulos del cerebro de un gato ante formas y figuras.

Se descubre que las neuronas tienen cierta organización jerárquica.



Fuente de la imagen: Stanford CS231N

1. Introducción y casos de usos.

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.

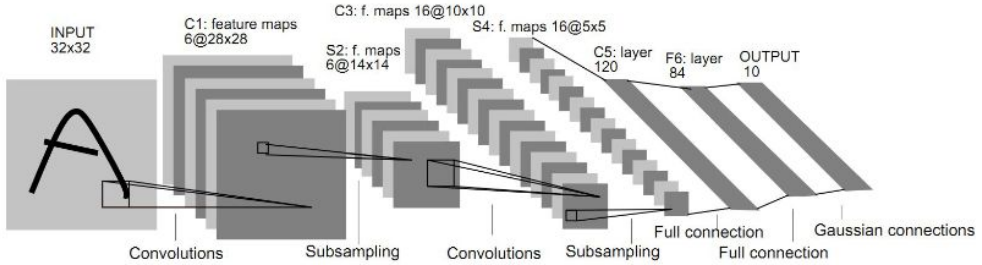
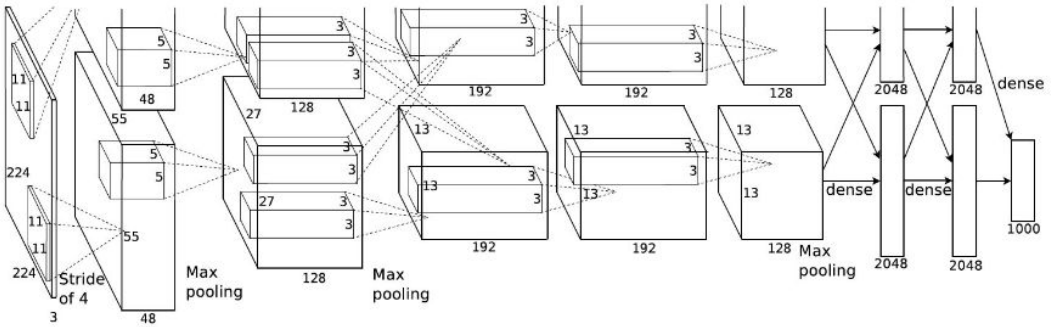


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

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

1. Introducción y casos de usos.



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.

1. Introducción y casos de usos.

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

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

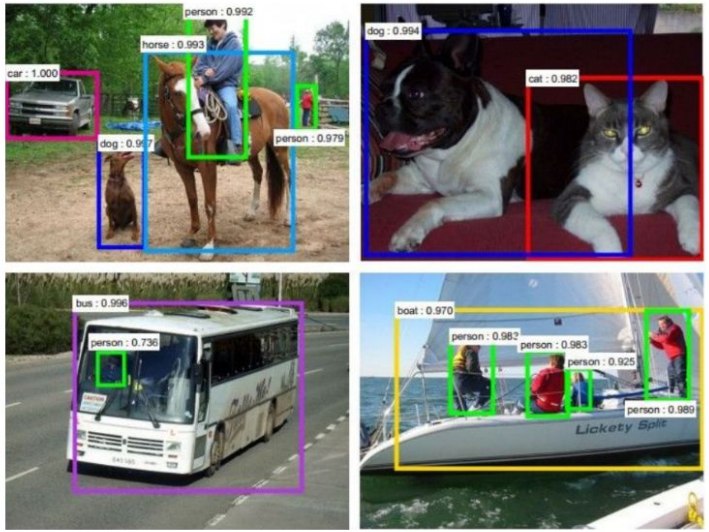
1. Introducción y casos de usos.

Retrieval de imágenes similares:

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

1. Introducción y casos de usos.

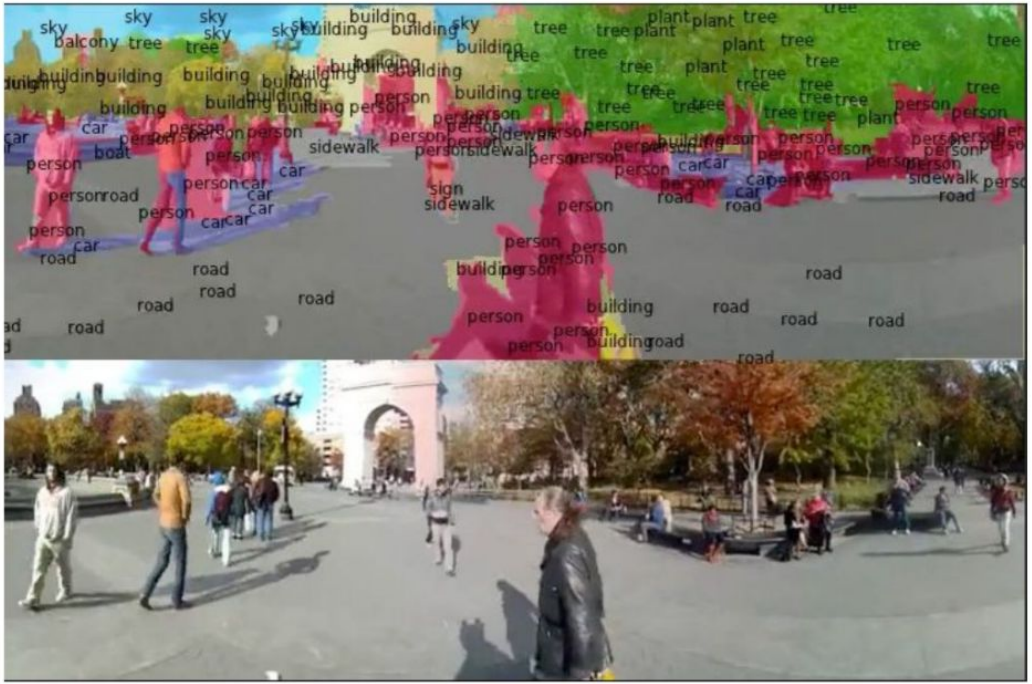
Object detection:



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

1. Introducción y casos de usos.

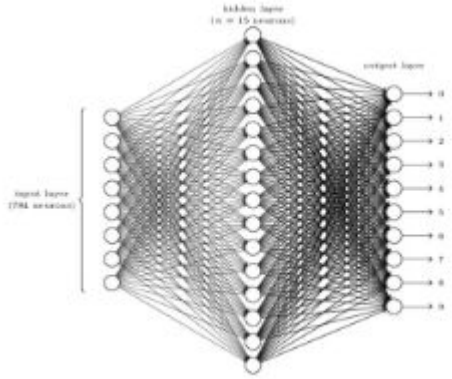
Image segmentation:



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

2. Convolution 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.



2. Convolution layers.

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.

Este gran número de parámetros empieza a ser un problema desde un punto de vista computacional.

2. Convolution layers.

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.

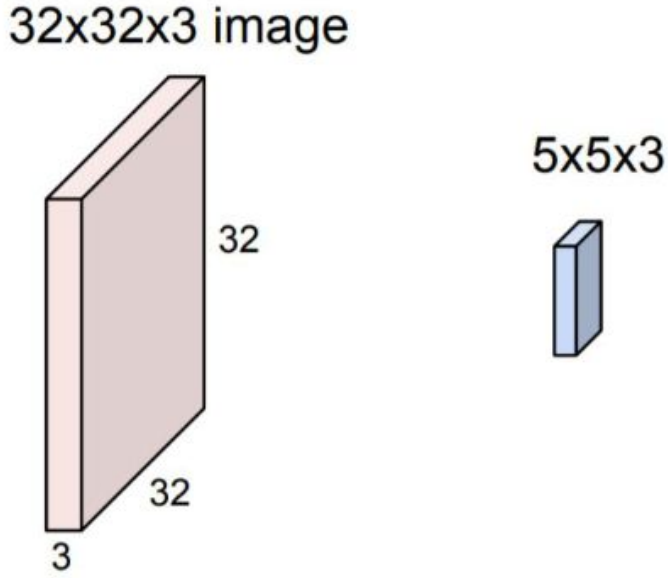
Las capas convolucionales solucionan este problema aprovechando la estructura espacial de la imagen.

2. Convolution layers.

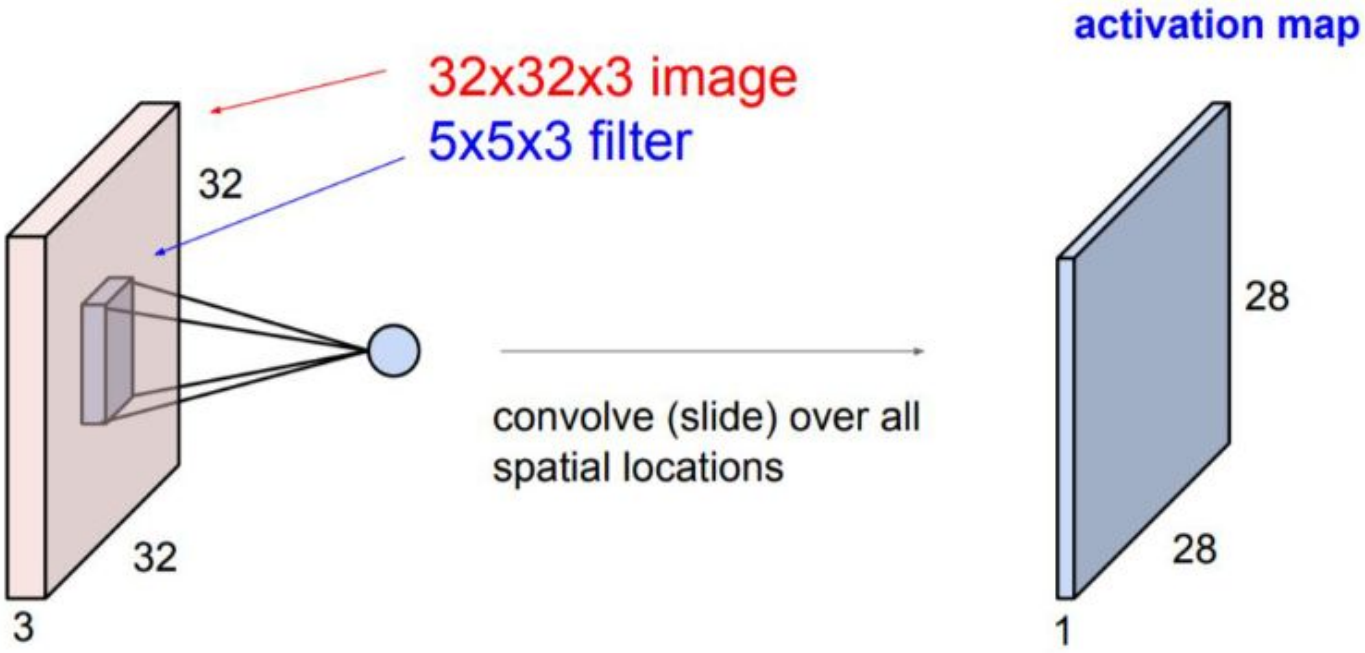
Las capas convolucionales utilizan filtros de pequeño tamaño en tres dimensiones.

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.



2. Convolution layers.

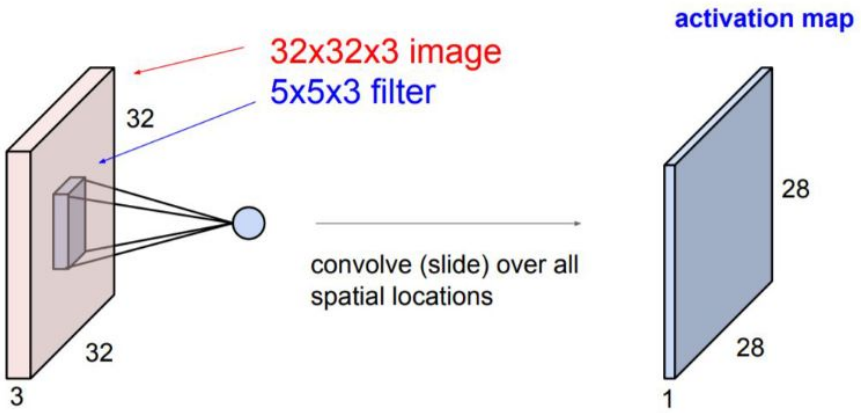


2. Convolution layers.

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 $w \cdot x + b$. Estamos multiplicando cada elemento del volumen por el correspondiente elemento del filtro, sumando esos productos y finalmente sumando el bias para obtener la salida.

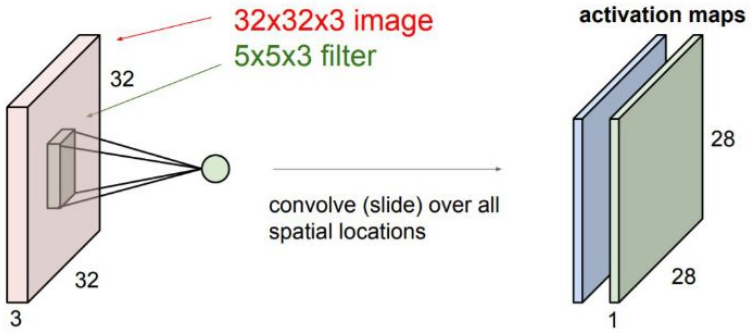
2. Convolution layers.



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)

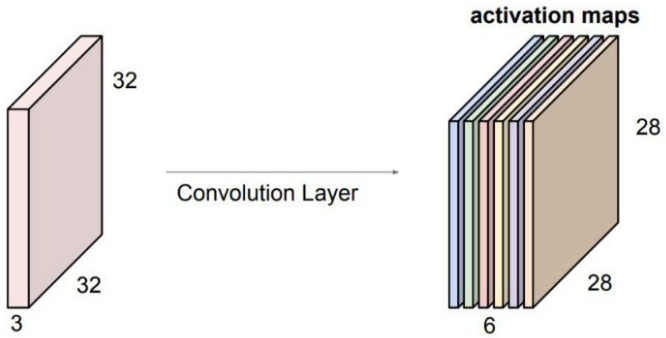
2. Convolution layers.



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.

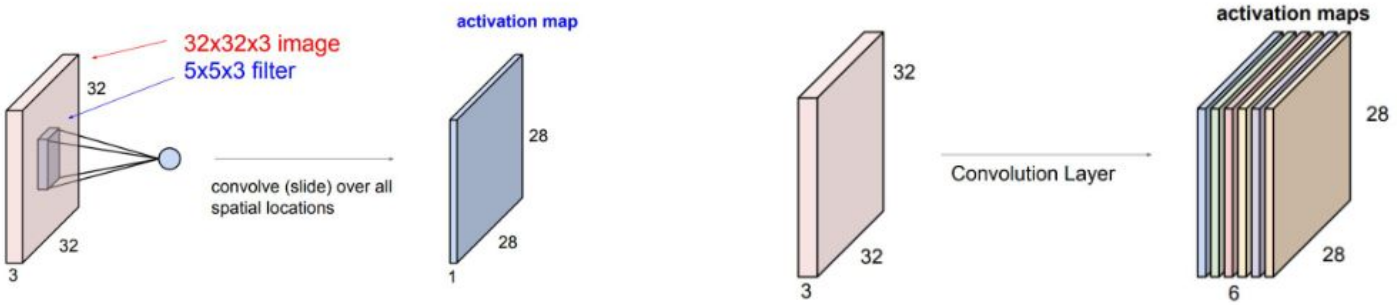
2. Convolution layers.



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

2. Convolution layers.



Número de parámetros para una convolution layer con 10 filtros de tamaño 5x5, con una imagen de entrada de tamaño 32x32x3

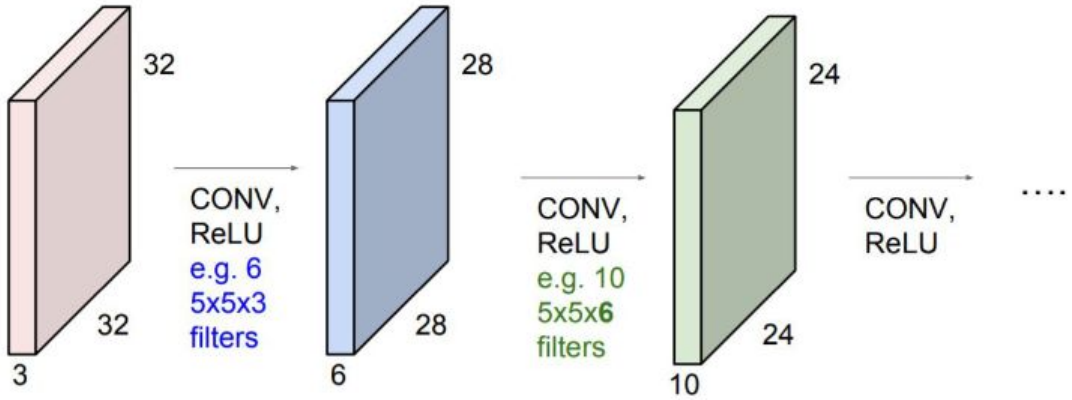
- Los filtros han de tener la misma profundidad que el volumen de entrada: 5x5x3.
- Por tanto, cada filtro tiene 5x5x3 = 75 pesos + 1 bias. Un total de 76 parámetros.
- En total, al tener 10 filtros en la capa, obtenemos un total de 760 parámetros

2. Convolution layers.

760 vs 270.000 parámetros

¡Independiente del tamaño de la imagen!

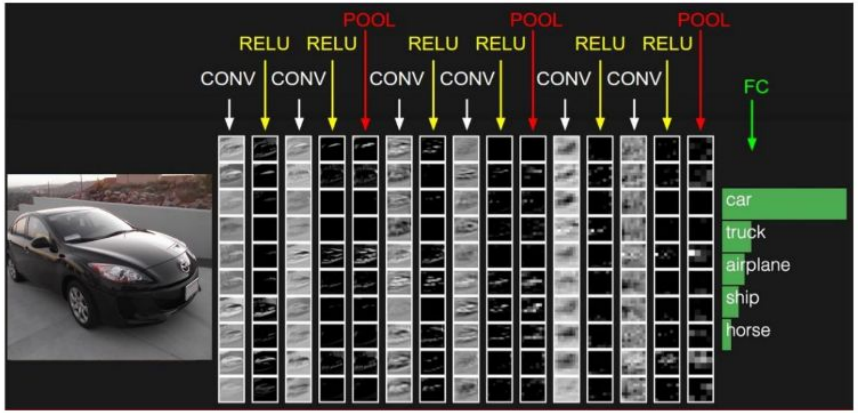
2. Convolution layers.



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. Volúmenes más pequeños y profundos.

2. Convolution layers.

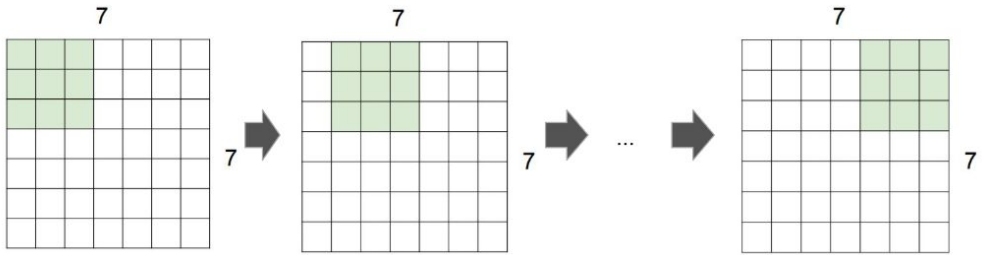


Las capas van aprendiendo una representación jerárquica de las features de la imagen, con las primeras capas reconociendo elementos más simples de una imagen y las últimas obteniendo representaciones de más alto nivel a partir de estos elementos simples.

3. 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

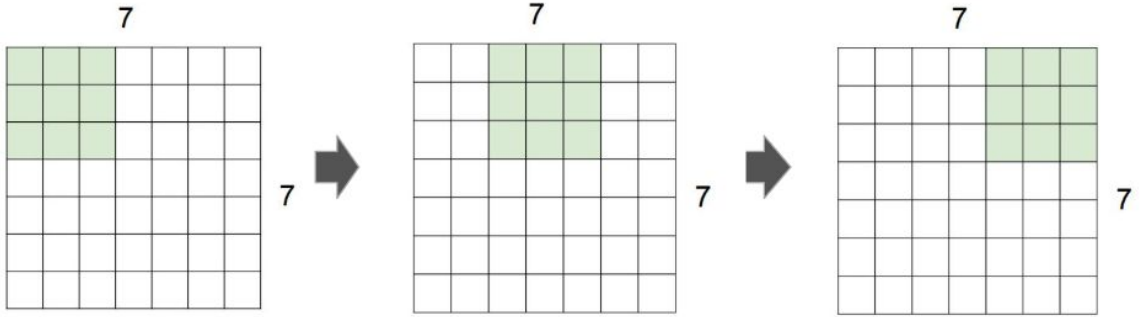


Output: 5x5x1

3. 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

3. Dimensiones espaciales

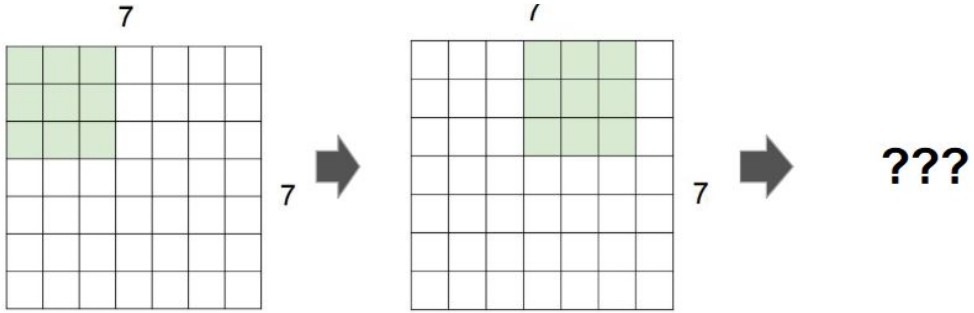
Con strides mayores, la red se fija en áreas más distantes de la imagen y obtenemos una reducción de la dimensionalidad.

Como si fuera un escaneo de menor resolución.

3. Dimensiones espaciales

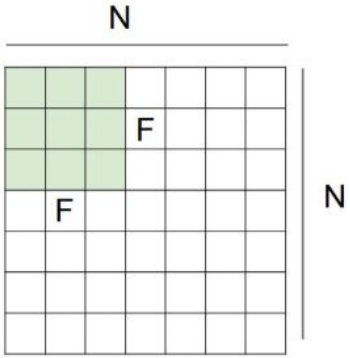
Stride: hiperparámetro que controla cómo se desplazan los filtros a través de la imagen.

Ejemplo: Input de 7x7x1, filtro de tamaño 3x3. **Stride 3**



No encaja

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

Tiene que dar un número entero para que encaje.

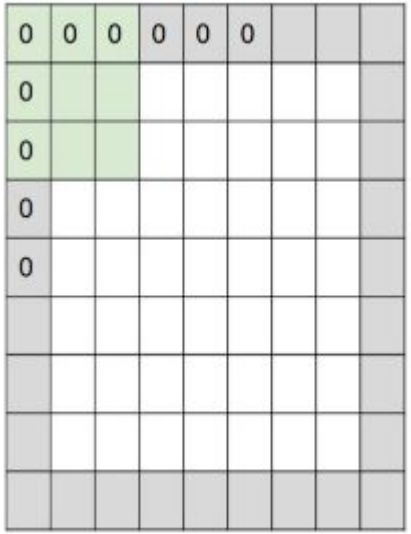
3. Dimensiones espaciales

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:

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



3. Dimensiones espaciales

En general, una capa convolucional con input un volumen de tamaño $W_1 \times H_1 \times D_1$

Tiene 4 hiperparametros:
- 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 $W_2 \times H_2 \times D_2$ donde

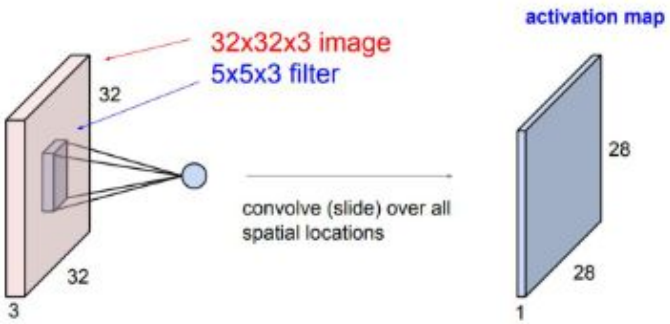
$$W_2 = (W_1 - F + 2P)/S + 1$$

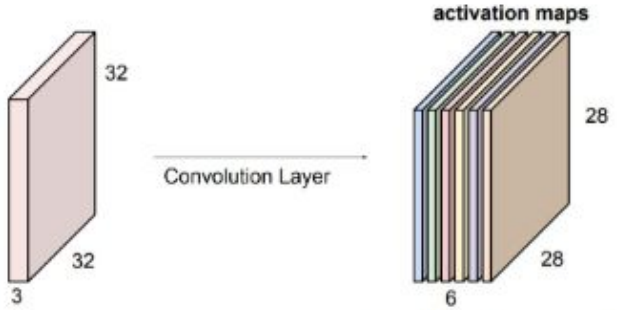
$$H_2 = (H_1 - F + 2P)/S + 1$$

$$D_2 = K$$

3. Dimensiones espaciales

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

3. Dimensiones espaciales

3. Dimensiones espaciales

Ejemplo: Input: 32x32x3, aplicamos 10 filtros de tamaño 5x5, stride 1 y padding 2.
¿ Volumen de salida y número de parámetros ?

$W_2 \times H_2 \times D_2$

$$W_2 = (W_1 - F + 2P)/S + 1$$

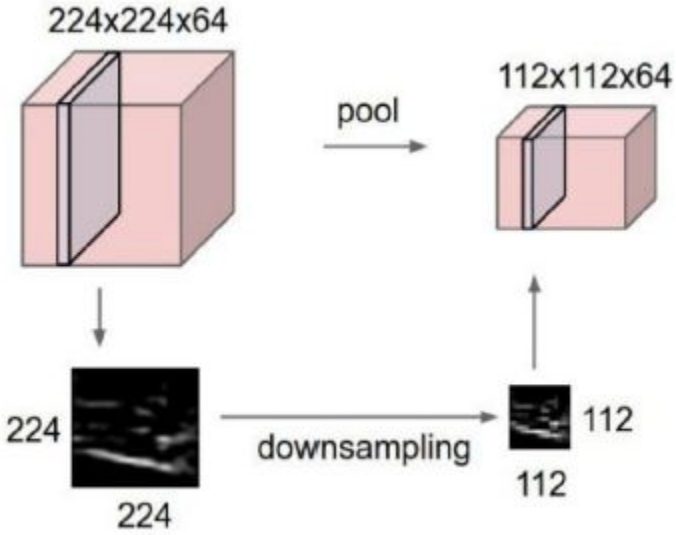
$$H_2 = (H_1 - F + 2P)/S + 1$$

$$D_2 = K$$

W= (32 - 5 +2*2)/1 +1= (27+4)/1+1= 32
H= (32 - 5 +2*2)/1+1 = (27+4)/1+1= 32
D=10

Volumen de salida: 32x32x10
Número de parámetros:
Cada filtro: 5x5x3 pesos + 1 bias = 76
76*10 = 760 parámetros

4. Capas Max Pooling



4. 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.

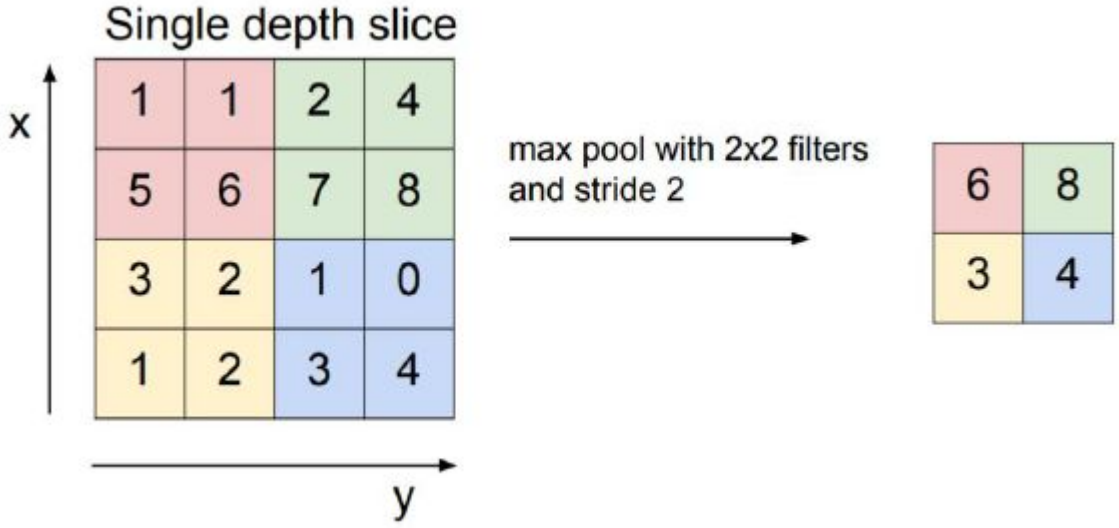
4. Capas Max Pooling

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%.

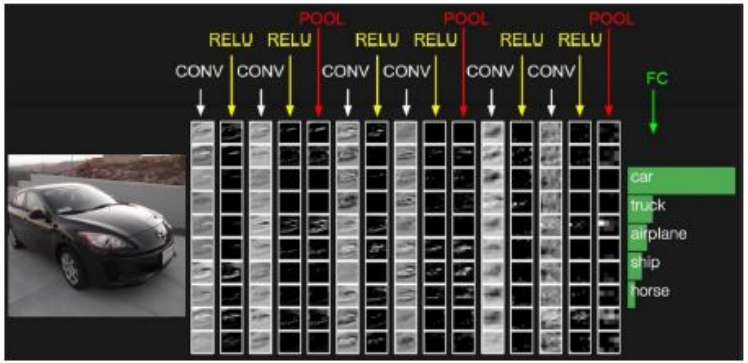
Las capas max pooling no tienen parámetros: su resultado consiste ent omar los máximos encontrados en el volumen de entrada.

4. Capas Max Pooling



4. 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.



4. Capas Max Pooling

En general, una capa max pooling aplicada a un volumen de tamaño

$W_1 \times H_1 \times D_1$

Tiene dos hiperparametros

- Tamaño de filtro o pool F
- Stride S (mismo valor horizontal y verticalmente)

4. Capas Max Pooling

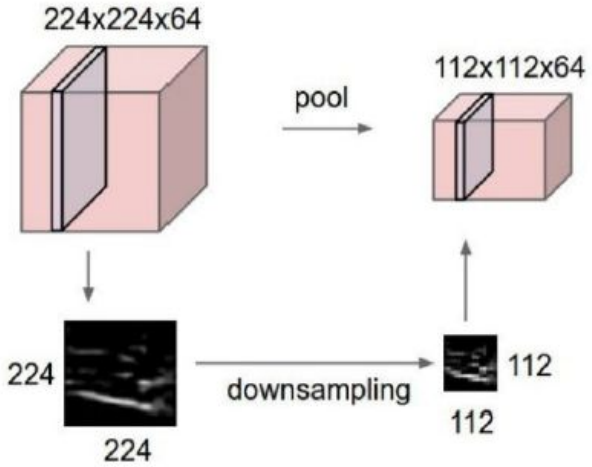
Produce un nuevo volumen $W_2 \times H_2 \times D_2$

$$W_2 = (W_1 - F)/S + 1$$

$$H_2 = (H_1 - F)/S + 1$$

$$D_2 = D_1$$

No produce parámetros



4. Capas Max Pooling

Tenemos un volumen input de 224x224x64, con un tamaño de filtro 2 y un stripe de 2. Calcular el nuevo volumen y los parámetros que produce.

$W_2 = (W_1 - F)/S + 1$ (224-2)/2+1 = 111 +1 = 112

$H_2 = (H_1 - F)/S + 1$ (224-2)/2+1 = 111 +1 = 112

$D_2 = D_1$ 64

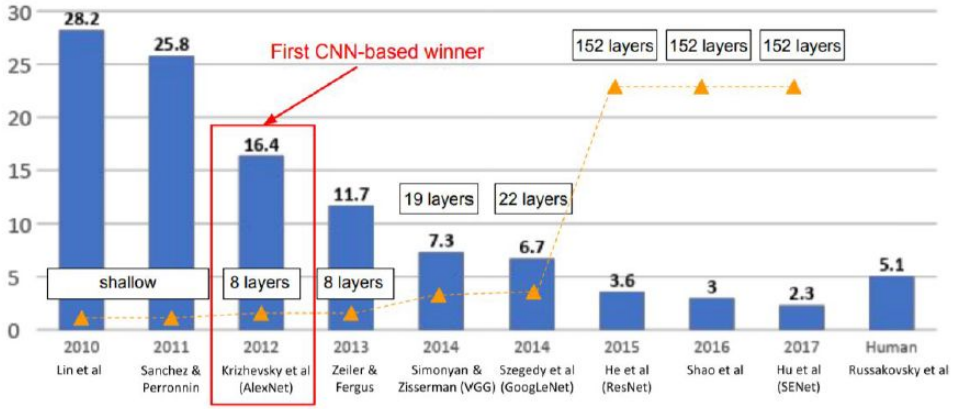
Nuevo volumen: 112x112x64 y no produce parámetros

5. Ejercicios de dimensiones

Es hora de aplicar los conceptos adquiridos de parámetros e hiperparámetros.

6. Arquitecturas CNN de visión por computador.

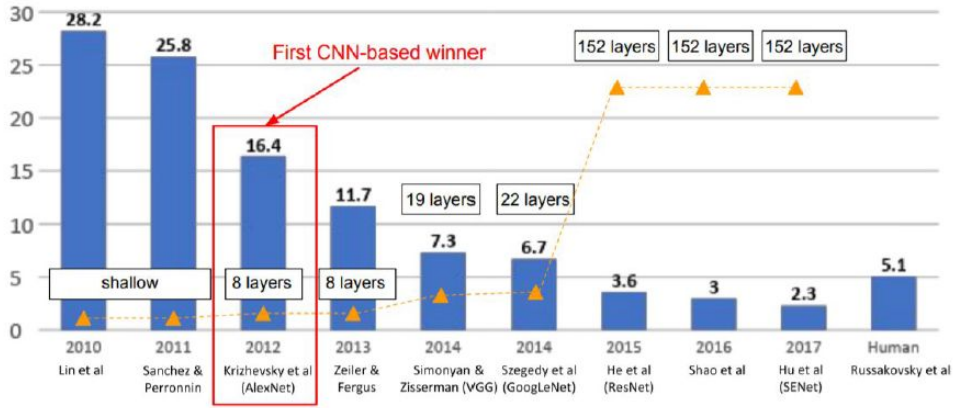
Resultados de la competición ILSVRC (ImageNet Large Scale Visual Recognition Challenge). Utiliza el dataset ImageNet 1000 posibles clases.



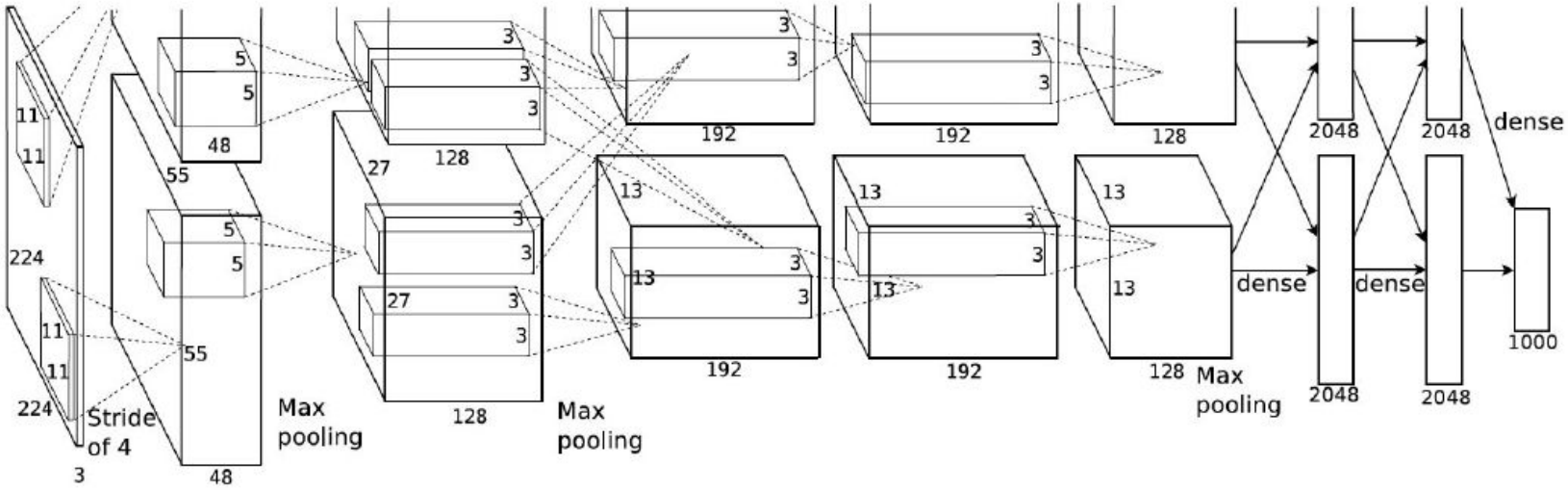
6. Arquitecturas CNN de visión por computador.

La tendencia es hacia redes más profundas.

Después de lo que hemos estudiado de parámetros e hiperparámetros vamos analizar AlexNet y VGG

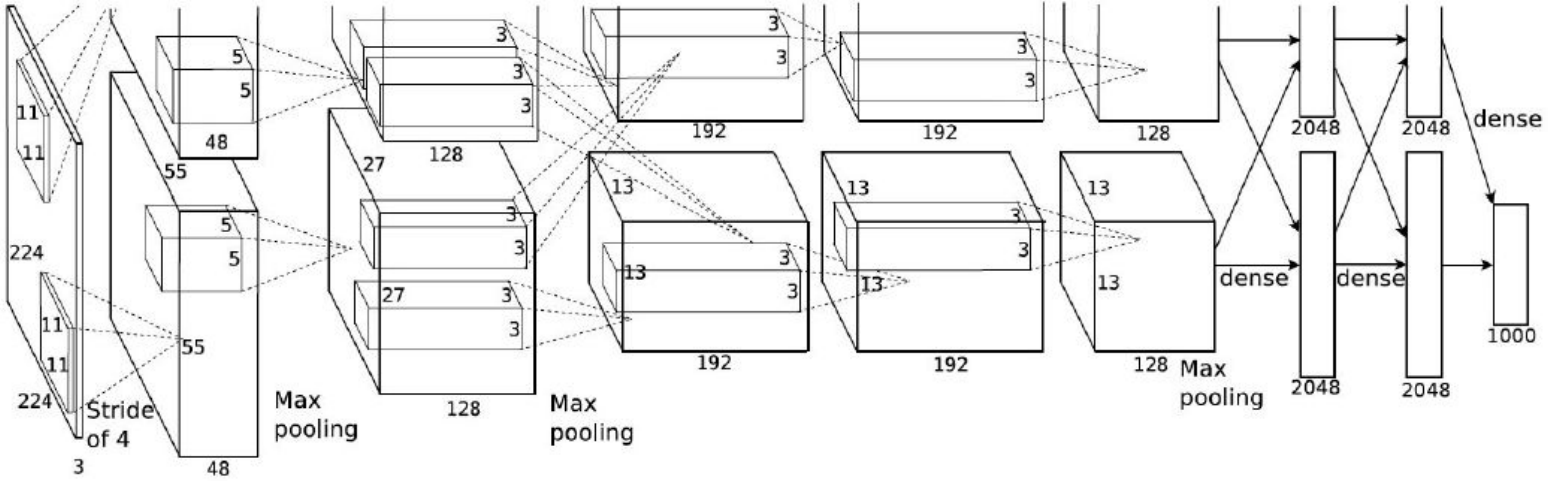


6. Arquitecturas CNN de visión por computador.



Input: imágenes 224x224x3, esta partida por 2 por qué no entraba en una GPU. Dividieron en 2 el grafo de computación y utilizaron 2 GPU. Tardaba una semana en entrenar.

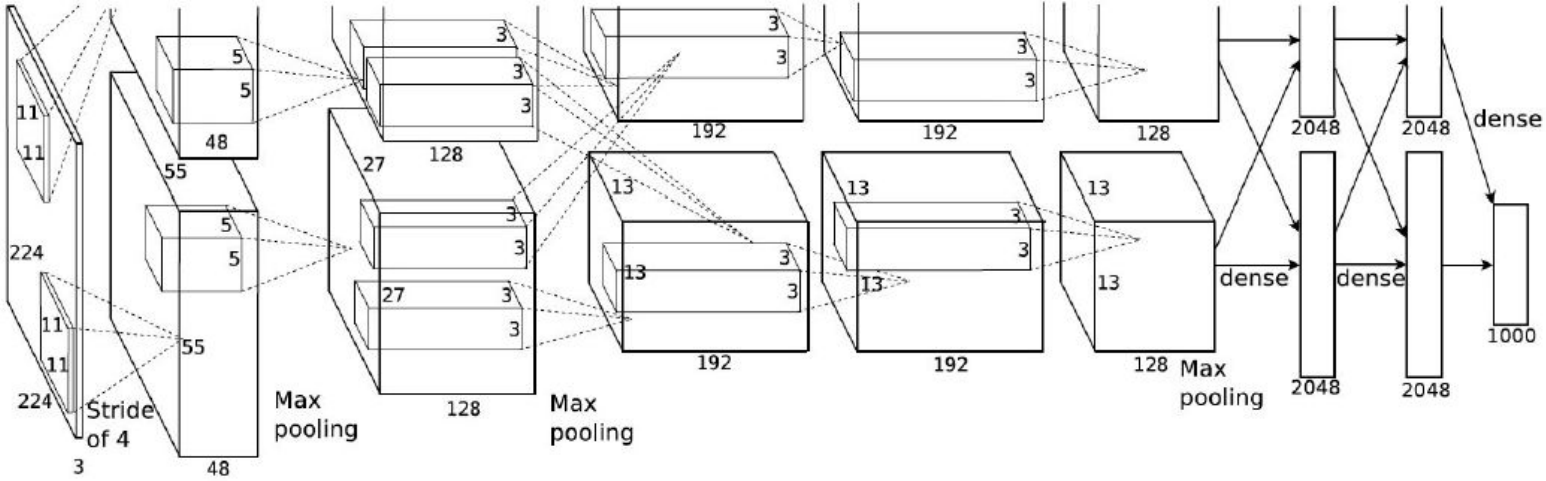
6. Arquitecturas CNN de visión por computador.



Primer bloque

- CONV 1 (96 11x11 filters, stride 4, pad 0)
- MAX POOL 1 (3x3 filters, stride 2)
- NORM 1 (Normalization layer)

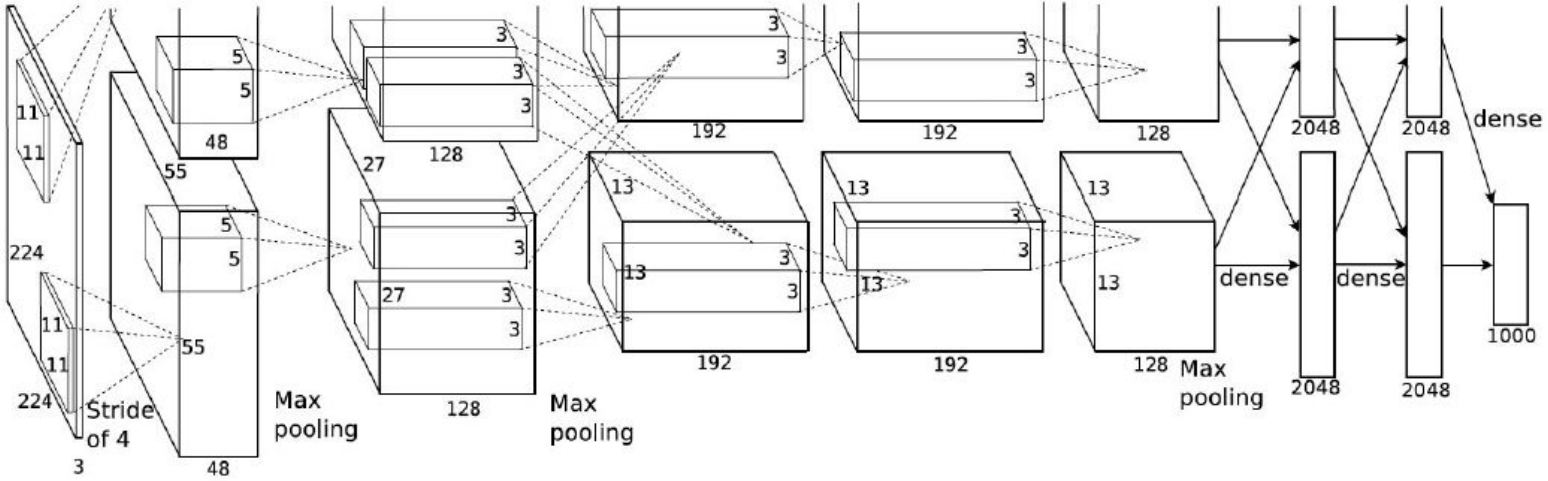
6. Arquitecturas CNN de visión por computador.



Segundo bloque

- CONV 2 (256 5x5 filters, stride 1, pad 2)
- MAX POOL 2 (3x3 filters, stride 2)
- NORM 2 (Normalization layer)

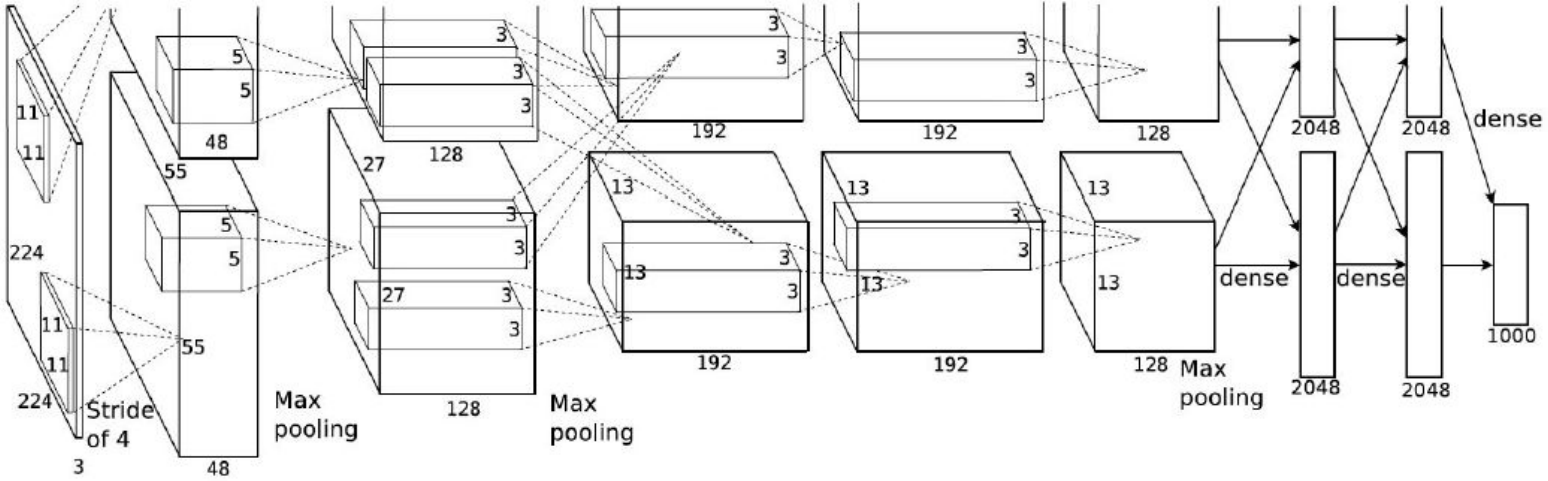
6. Arquitecturas CNN de visión por computador.



Tercer bloque

CONV 3 (384 3x3 filters, stride 1, pad 1)

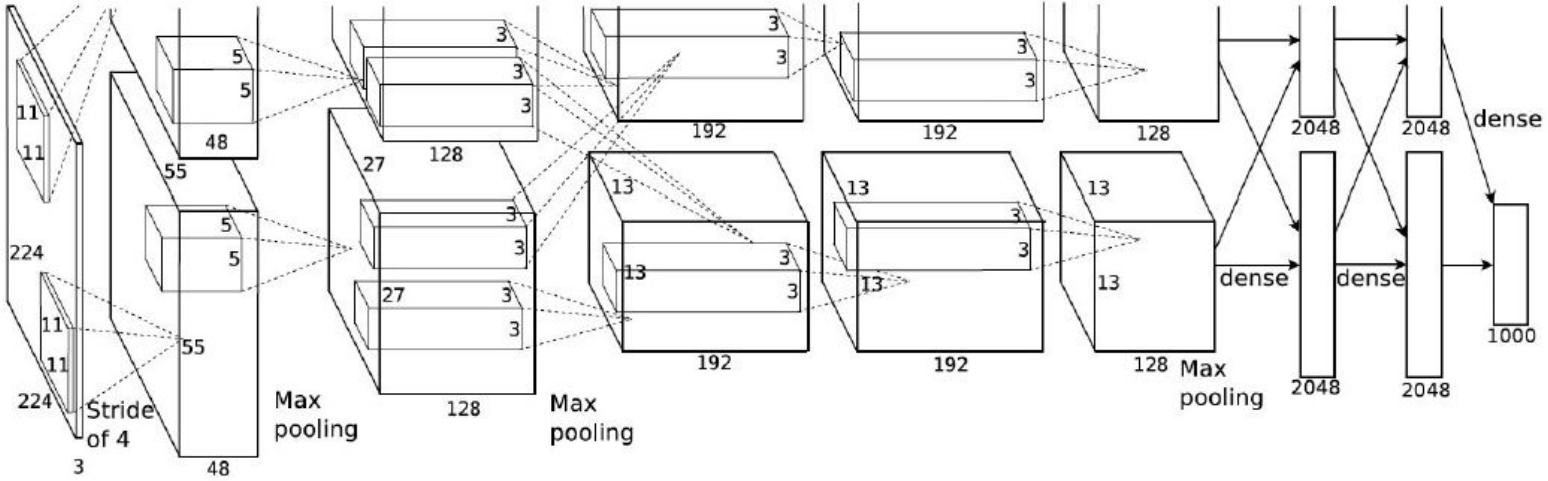
6. Arquitecturas CNN de visión por computador.



Cuarto bloque

CONV 4 (384 3x3 filters, stride 1, pad 1)

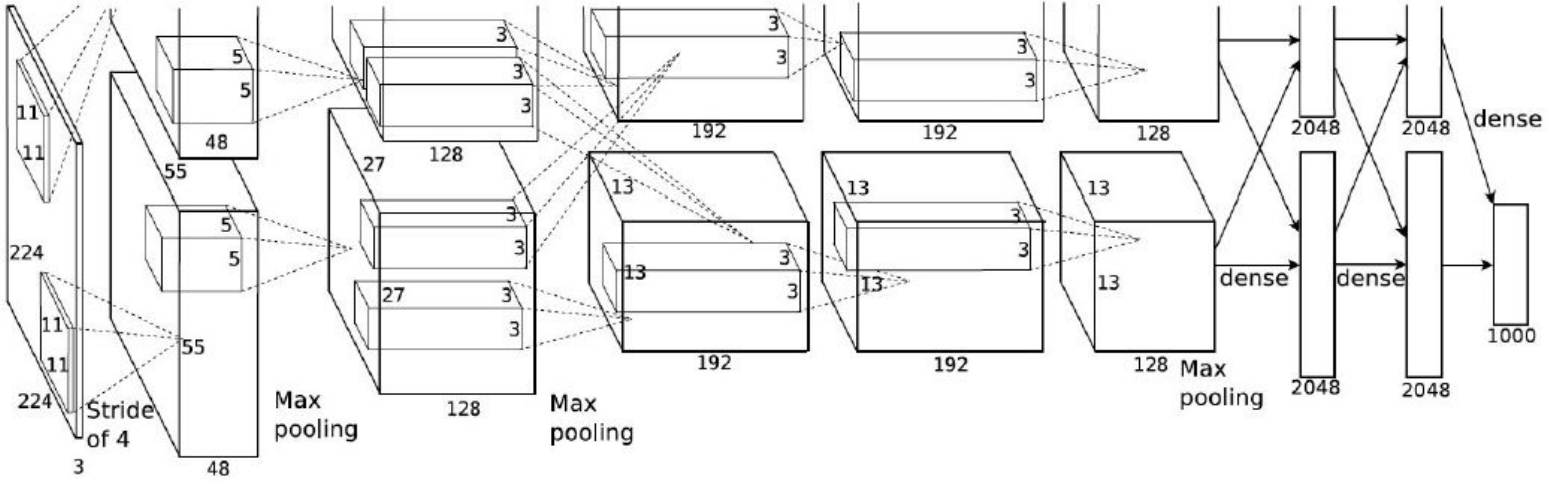
6. Arquitecturas CNN de visión por computador.



Quinto bloque

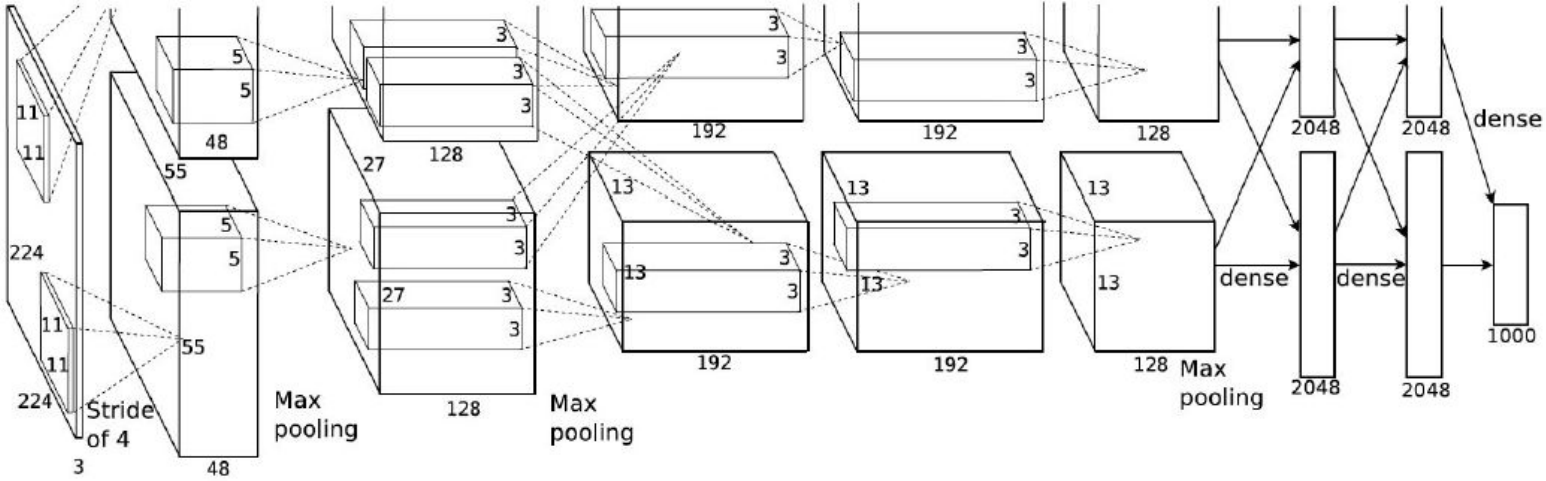
- CONV 5 (256 3x3 filters, stride 1, pad 1)
- MAX POOL 3 (3x3 filters, stride 2)

6. Arquitecturas CNN de visión por computador.



- Fully connected layer (FC6), 4096 neuronas
- Fully connected layer (FC7), 4096 neuronas
- Fully connected layer (FC8), 1000 neuronas, class scores.

6. Arquitecturas CNN de visión por computador.



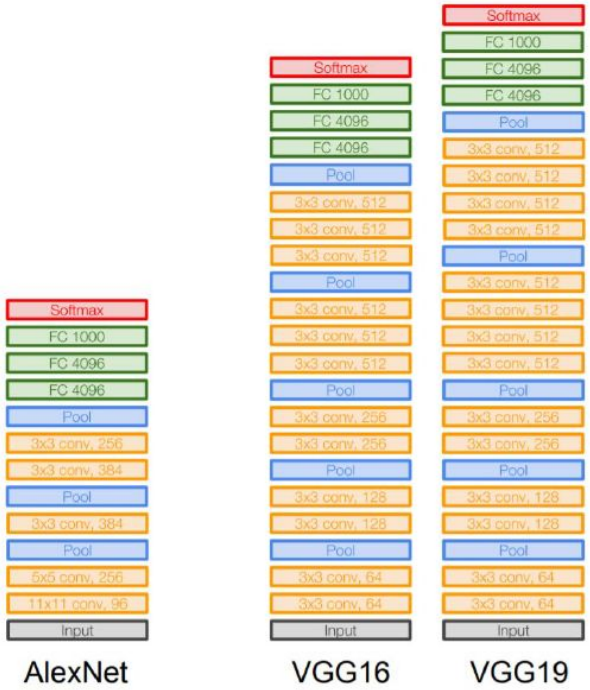
Vemos cómo la profundidad de los volúmenes va creciendo, a la vez que el tamaño espacial se reduce mediante max pooling. Queremos que la red obtenga representaciones más complejas, donde los elementos de la imagen se van componiendo de manera jerárquica.

6. Arquitecturas CNN de visión por computador.

AlexNet datos de interés:

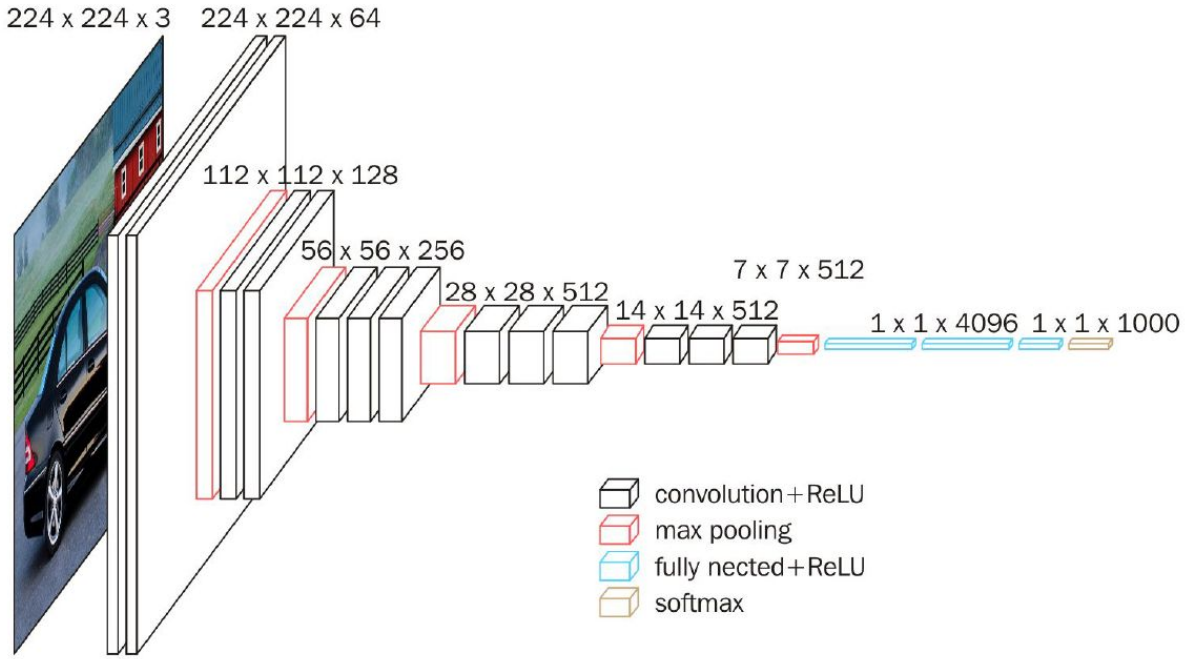
- El batch size es 128.
- El algoritmo de optimización es SGD con Momentum 0,9.
- El learning rate utilizado es 1e-2, dividido por 10 manualmente cuando el validation error deja de mejorar.
- Se usa dropout con valor 0.5.
- Se aplica regularización L2 con peso 5e-4.

6. Arquitecturas CNN de visión por computador.



► 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.

6. Arquitecturas CNN de visión por computador.



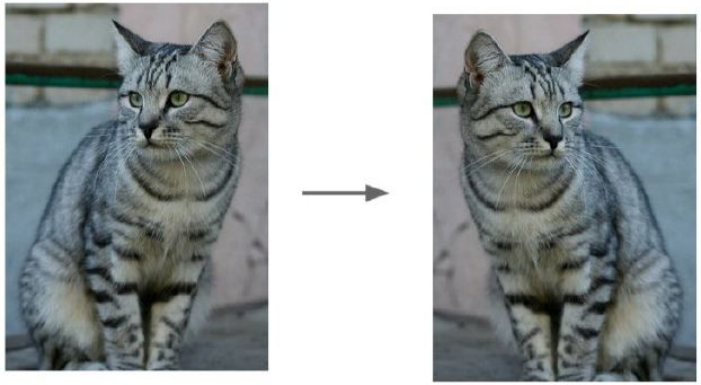
6. Arquitecturas CNN de visión por computador.

INPUT: [224x224x3] pesos: 0
CONV3-64: [224x224x64] pesos: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] pesos: (3*3*64)*64 = 36,864
POOL2: [112x112x64] pesos: 0
CONV3-128: [112x112x128] pesos: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] pesos: (3*3*128)*128 = 147,456
POOL2: [56x56x128] pesos: 0
CONV3-256: [56x56x256] pesos: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] pesos: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] pesos: (3*3*256)*256 = 589,824
POOL2: [28x28x256] pesos: 0
CONV3-512: [28x28x512] pesos: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] pesos: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] pesos: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] pesos: 0
CONV3-512: [14x14x512] pesos: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] pesos: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] pesos: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] pesos: 0
FC: [1x1x4096] pesos: 7*7*512*4096 = **102,760,448**
FC: [1x1x4096] pesos: 4096*4096 = **16,777,216**
FC: [1x1x1000] pesos: 4096*1000 = **4,096,000**

¡La mayoría de los pesos se encuentra en las fully connected layers!

La tendencia actual es tratar de eliminar esas capas para reducir el tamaño del modelo.

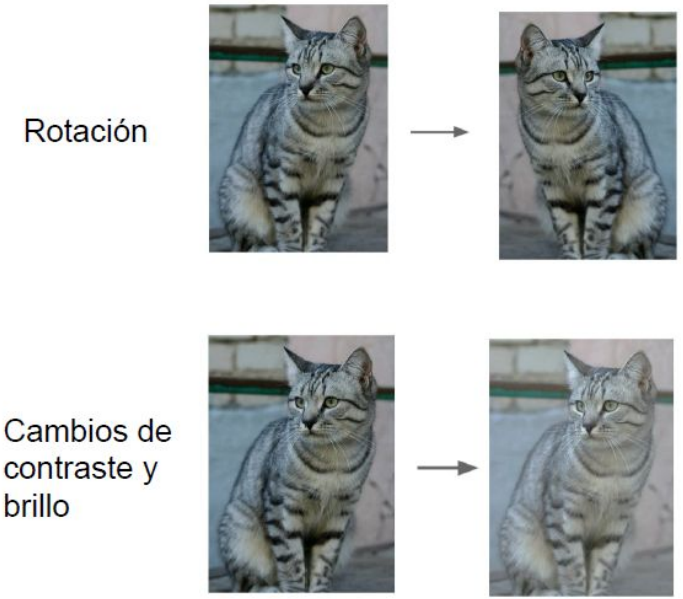
7. Data Augmentation



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.

7. Data Augmentation



Permite entrenar modelos con éxito utilizando menos imágenes.

Posibles estrategias:

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

7. Data Augmentation

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.

8. 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.

8. Transfer Learning

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.

8. Transfer Learning

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

8. 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.