



PONTIFICIA
UNIVERSIDAD
CATÓLICA DE
VALPARAÍSO



Introducción al Deep Learning

Convolutional Neural Network

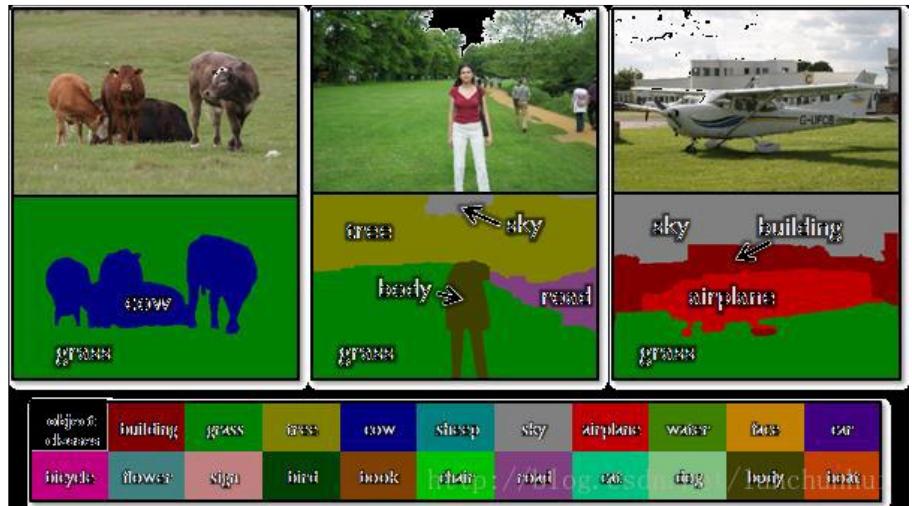
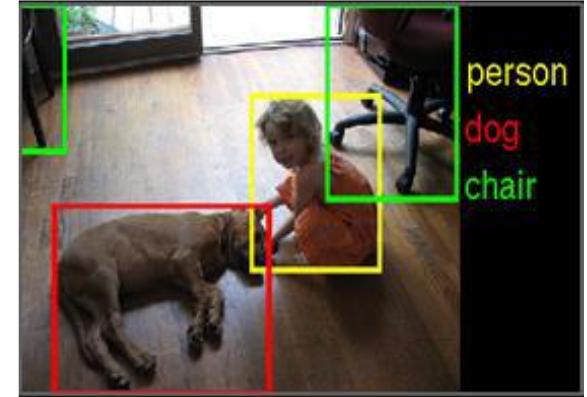
Dr. Ing. Gabriel Hermosilla Vigneau

¿Qué hace una red neuronal convolucional?

- Utiliza ejemplos (datos-etiquetas, entradas-salidas, x-y).
- Aproxima/genera una función. Dada una entrada genera una salida.
- Permite:
 - Clasificar
 - Regresión
 - Generar datos
- Con una única capa oculta y un número finito de neuronas se puede aproximar cualquier función.
- ¿Cómo lo hace? parte de un estado inicial (aleatorio), altera los pesos de manera que se minimiza el error entre las respuestas deseadas (presentadas) y lo que genera la red.

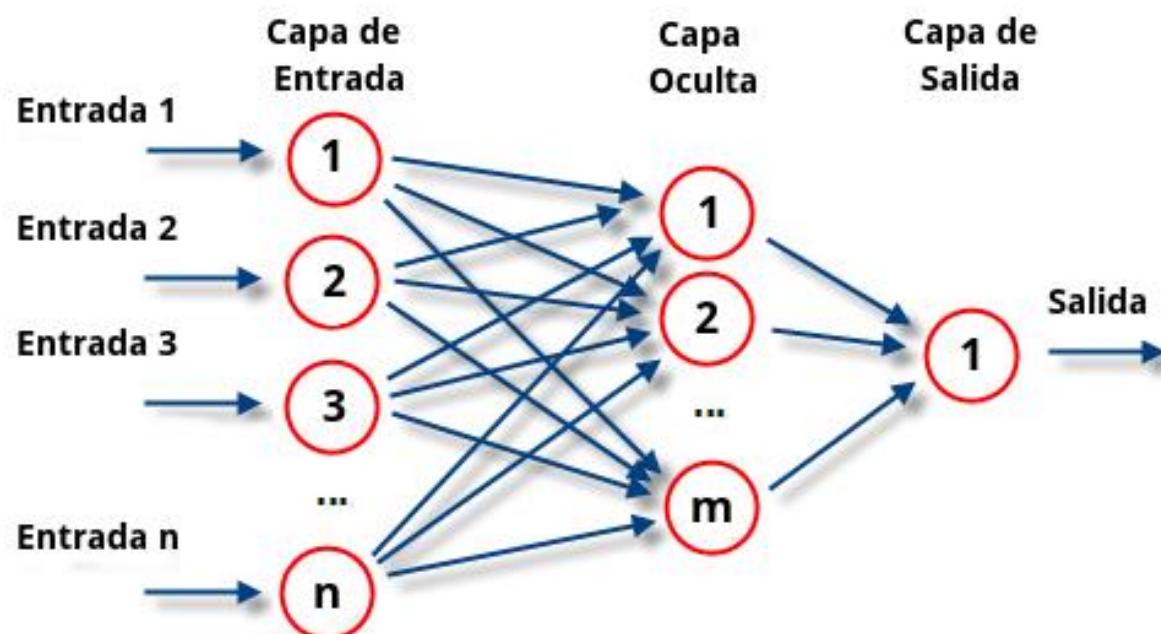
Para qué se usa en Visión por Computador

- Clasificación de imágenes: 1K–10K clases
- Detección de objetos: 20-1K clases
- Segmentación semántica: 3 –20 clases



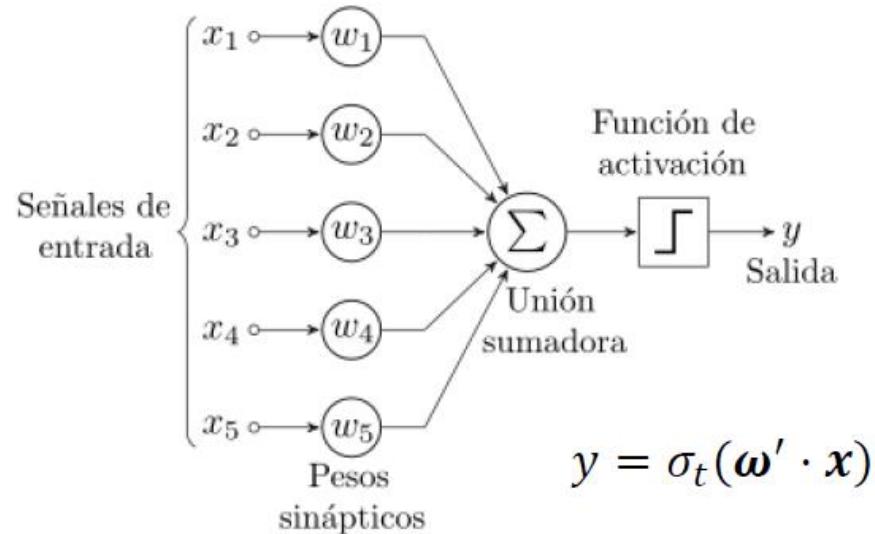
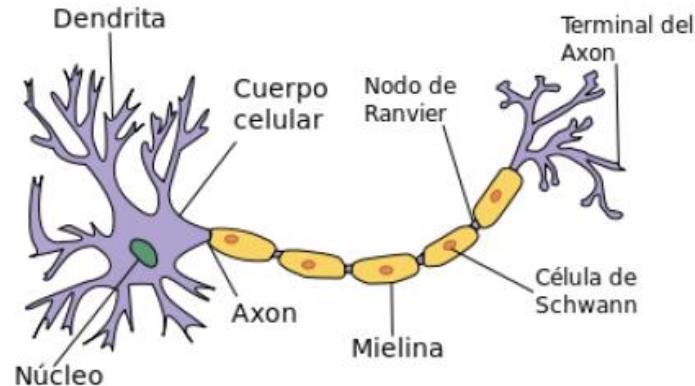
Redes Neuronales (Repasso)

- Red Clásica (feedforward)



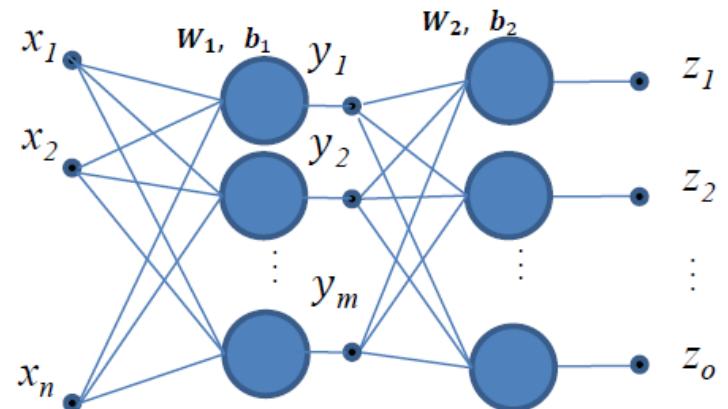
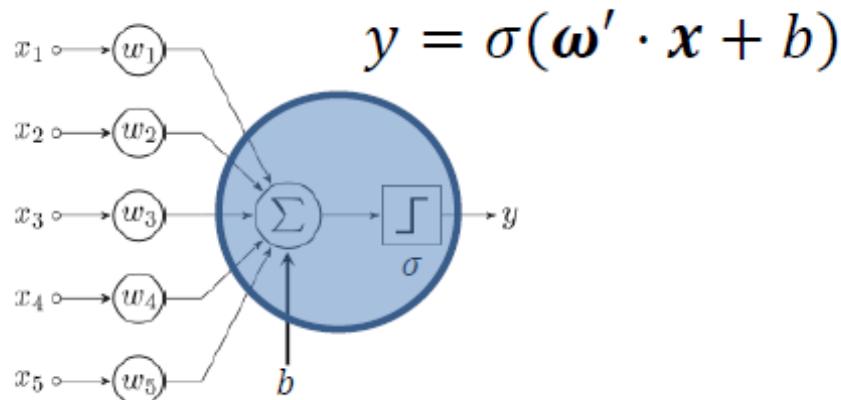
Perceptrón (Repaso)

- Neurona simple:
 - Pesos sinápticos
 - Estímulo acumulado
 - Salida (todo o nada)
- Neurona computacional:
 - n inputs y una salida
 - Parámetros (pesos w)
 - Función de activación



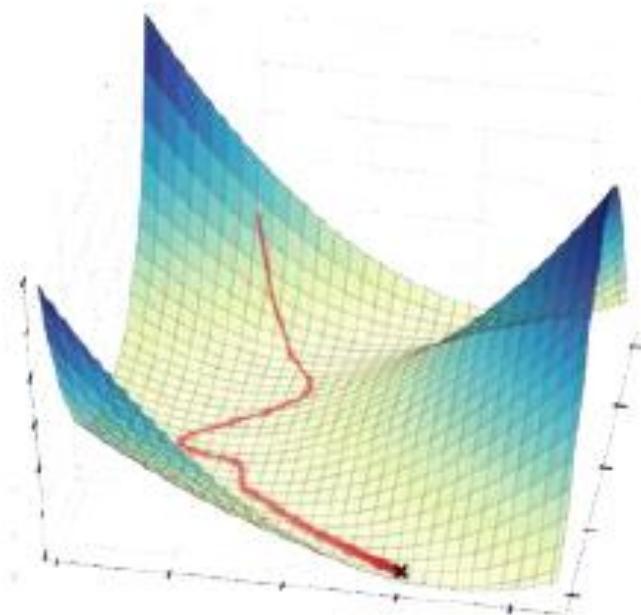
Perceptrón Multicapa (Repaso)

- Red neuronal feedforward
- Más capas => ocultas (hidden layers) => Deep
- Funciones no lineales
- Se utiliza un bias (sesgo) para no ajustar el umbral de la función de activación
- Se utilizan funciones de activación derivables (permiten el aprendizaje)



Backpropagation (Repaso)

- El objetivo es minimizar la diferencia entre la función objetivo (respuestas conocida) y la respuesta de la red (función de pérdida)
- Utilizaremos la regla del descenso del gradiente.
- El error que observamos en la salida, lo propagamos hacia el interior. Se recalcularán pesos y bias.
- Se utiliza la regla de la cadena para conocer cómo se ve afectado cada parámetro.

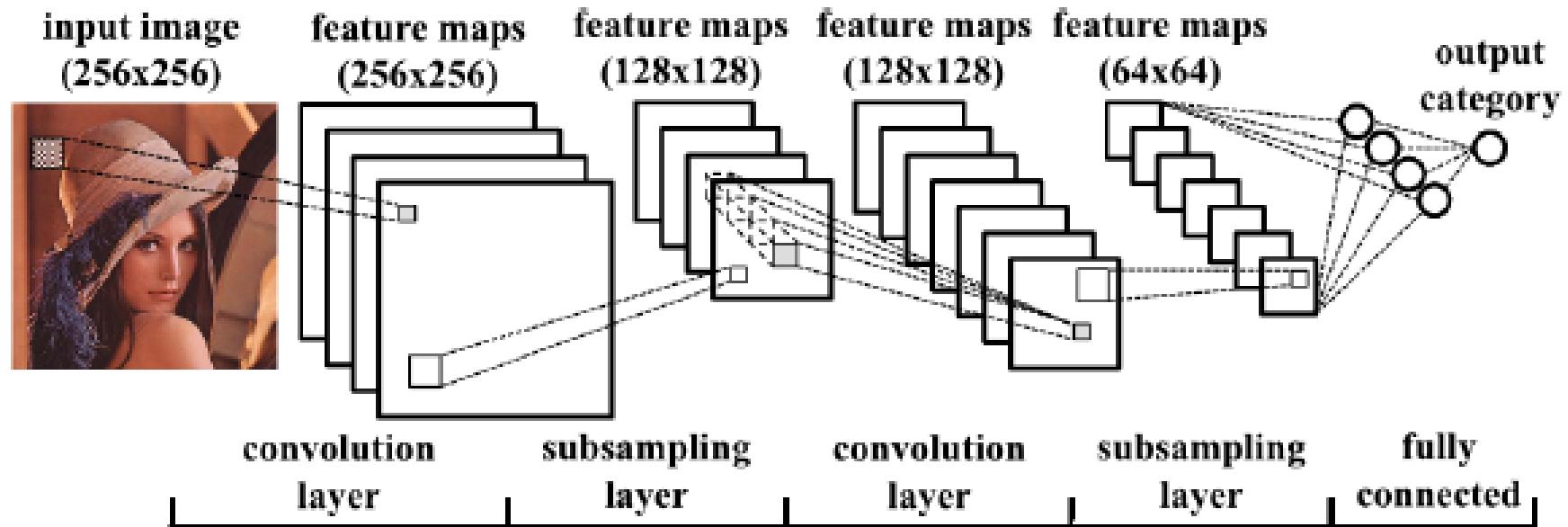


Redes Neuronales Convolucionales

- Son redes que se usan para procesar imágenes.
- Pueden aprender relaciones entrada-salida, donde la entrada es una imagen.
- Están basadas en operaciones de convolución (extracción de características)
- Permite realizar la clasificación de imágenes.
- Tareas comunes:
 - Detección/categorización de objetos
 - Clasificación de escenas
 - Clasificación de imágenes en general

Redes Neuronales Convolucionales

- Arquitectura clásica (básica).

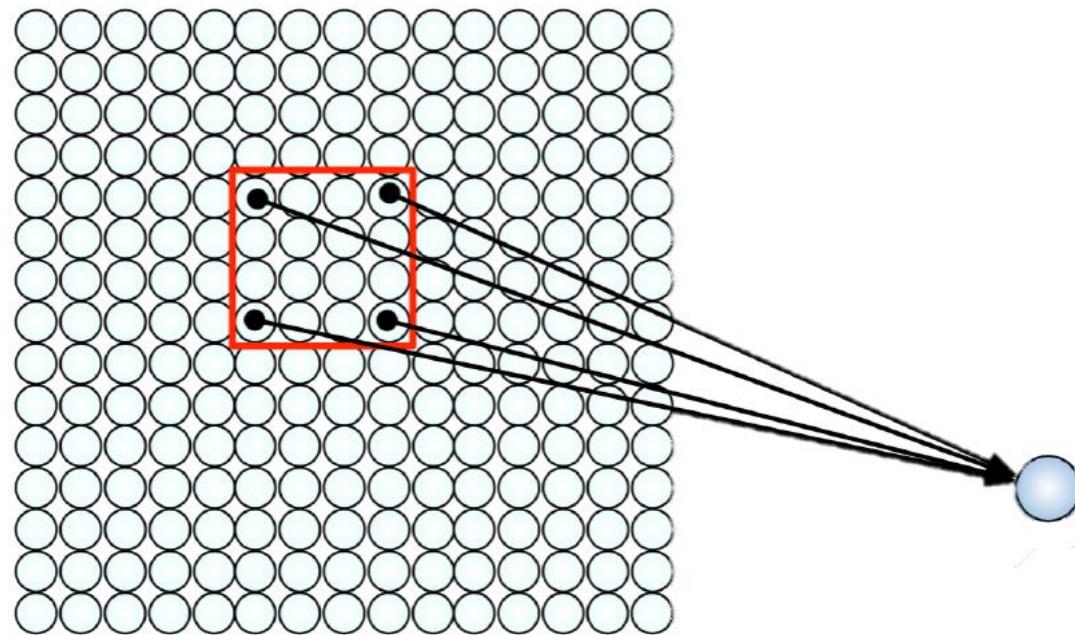


Redes Neuronales Convolucionales

- Arquitectura:
 - Cada capa es un volumen de neuronas en 3D
 - Dimensiones: Alto, Ancho, Profundidad de la capa.
- ConvNets: Se forman usando tres tipos de capas:
 - Inputs: imágenes de entrada
 - Capas Convolucionales:
 - CNN: Convolutional neural network (filtrado y ReLU), utilizan máscaras (filtros, la red aprende los valores de los filtros).
 - Capas de pooling:
 - Subsampling: Submuestreo de los datos (Pooling), la salida es el máximo de la entrada en una ventana.
 - Capas totalmente conectadas:
 - FC: Fully Connected; se aplican al final de la red, se pierde información espacial.
 - Softmax/Classification (clasificación)

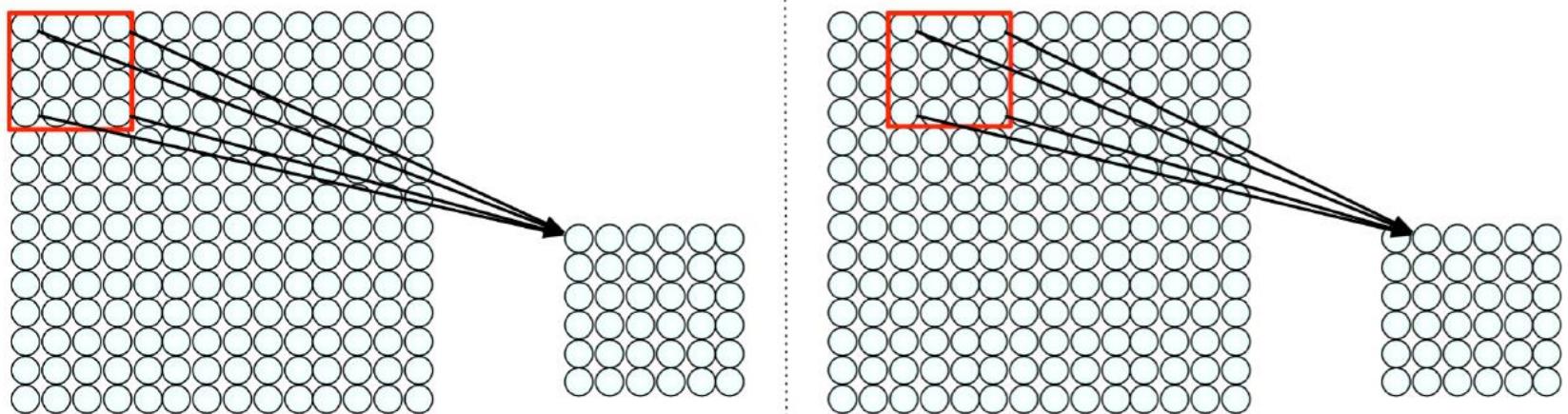
Redes Neuronales Convolucionales

- Arquitectura espacial
- Entrada: una imagen 2D. Un arreglo de valores de pixeles.
- Idea: conectar parches de entrada a las neuronas en la capa oculta.



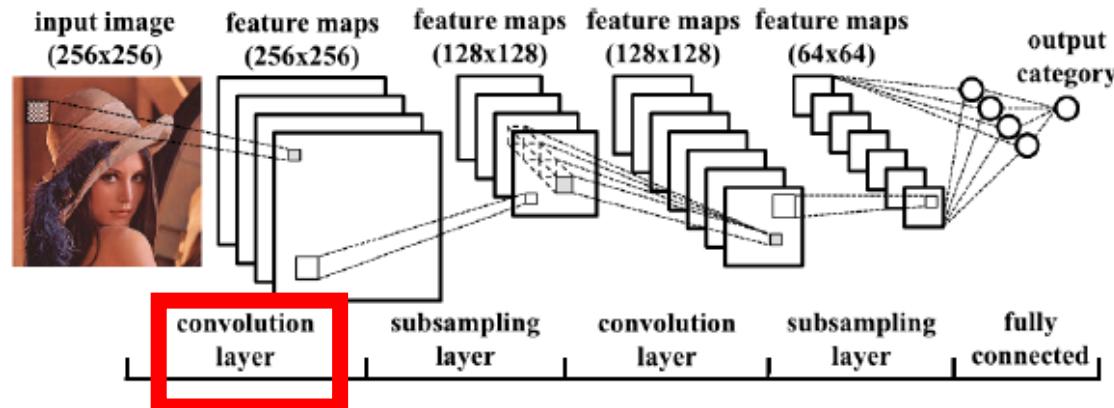
Redes Neuronales Convolucionales

- Arquitectura espacial
 - Se conecta el parche de la capa de entrada a una sola neurona en la capa siguiente.
 - Se utiliza una ventana deslizante (sliding window) para definir las conexiones.
-
- ¿Cómo podemos ponderar el parche para detectar características particulares?



Capas Convolucionales (CNN)

- La capa convolucional está constituida por una serie de filtros de convolución con capacidad de aprender y extraer características locales.
- Cada filtro usualmente posee una sección transversal pequeña, pero se extienden a lo largo de toda la profundidad del volumen de entrada.
- Por ejemplo, un filtro típico en la primera capa convolucional sería [5x5x3], durante el proceso de avance (forward) cada filtro convoluciona a través de la sección transversal del volumen de entrada, se realiza un producto punto entre los valores del filtro y el input en todas las posiciones.

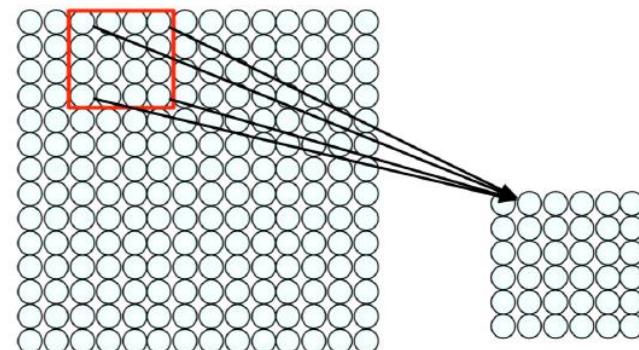


Capas Convolucionales (CNN)

- Pasos generales:
 1. Aplicar un set de filtros (pesos) para extraer características locales
 2. Usar múltiples filtros para extraer diferentes características
 3. Compartir espacialmente los parámetros de cada filtro

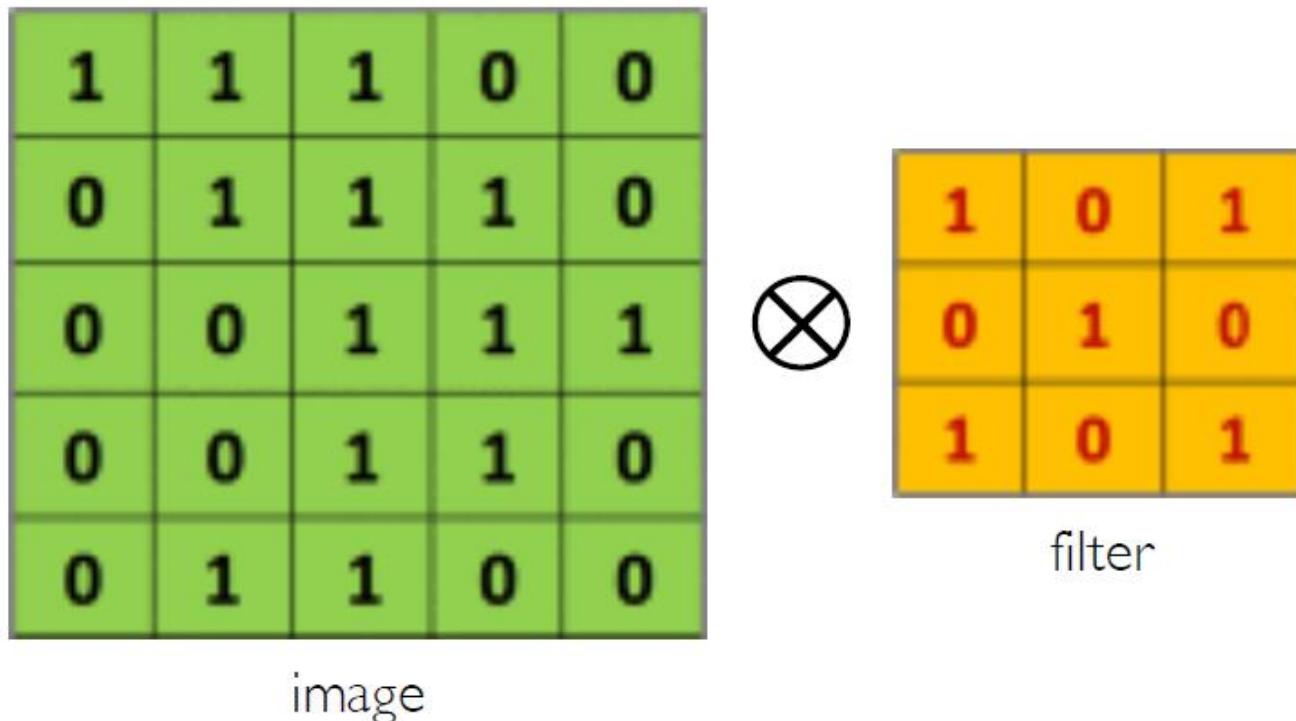
- Ejemplo:
 - Filtro de tamaño 4x4: 16 pesos diferentes
 - Aplicar el mismo filtro de 4x4 parches a la entrada
 - Desplazar cada 2 pixeles para el próximo parche.

- La operación matemática usada es la Convolución.



Operación de convolución

- Supongamos que queremos realizar la convolución de una imagen de 5x5 con un filtro de 3x3.
- Es necesario desplazar el filtro de 3x3 sobre la imagen de entrada y realizar la operación de convolución.



Operación de convolución

1 x1	1 x0	1 x1	0	0
0 x0	1 x1	1 x0	1	0
0 x1	0 x0	1 x1	1	1
0	0	1	1	0
0	1	1	0	0



1	0	1
0	1	0
1	0	1

filter



4		

feature map

Operación de convolución

1	1 x1	1 x0	0 x1	0
0	1 x0	1 x1	1 x0	0
0	0 x1	1 x0	1 x1	1
0	0	1	1	0
0	1	1	0	0



1	0	1
0	1	0
1	0	1

filter



4	3	

feature map

Operación de convolución

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0



1	0	1
0	1	0
1	0	1

filter



4	3	4

feature map

Operación de convolución

1	1	1	0	0
0	1	1	1	0
0	0	1 _{*1}	1 _{*0}	1 _{*1}
0	0	1 _{*0}	1 _{*1}	0 _{*0}
0	1	1 _{*1}	0 _{*0}	0 _{*1}



1	0	1
0	1	0
1	0	1

filter



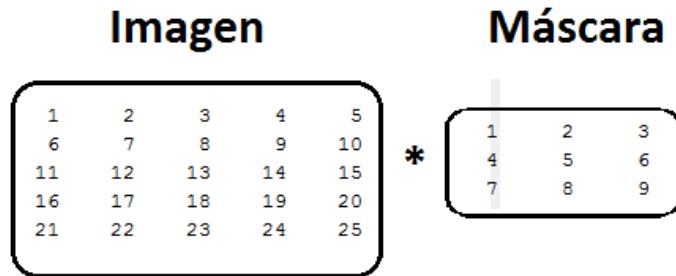
4	3	4
2	4	3
2	3	4

feature map

Convolución bidimensional:

$$x[n_1, n_2] * h[n_1, n_2] = \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} x[k_1, k_2] h[n_1 - k_1, n_2 - k_2]$$

- Ejemplo:
- Se quiere calcular la convolución de la imagen y una máscara (filtro).



- Una forma sencilla de implementar la convolución 2D es rotar la máscara de convolución en 180° y luego multiplicar cada valor de la imagen con el correspondiente en la máscara de convolución.

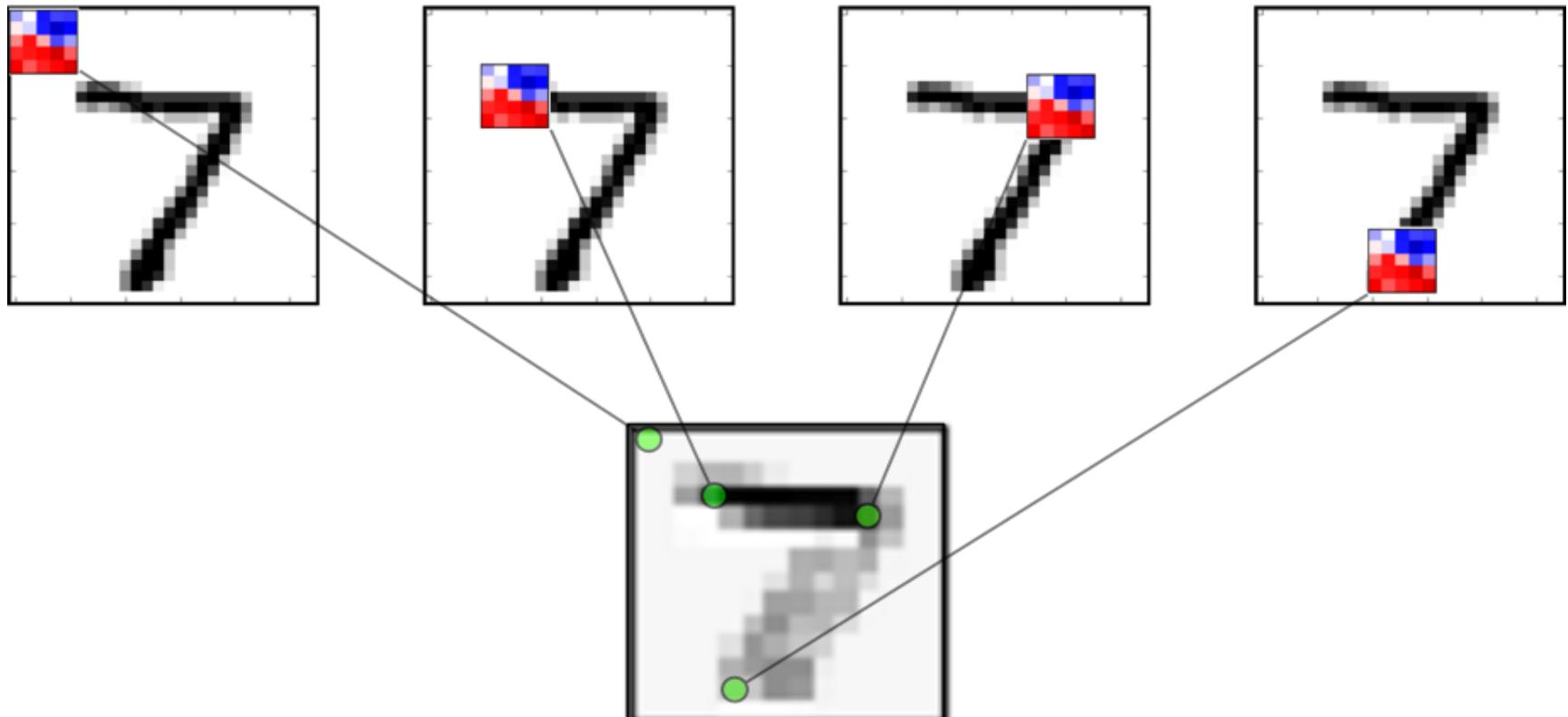
- Para el ejemplo se seleccionará solo una porción de la imagen para mostrar la convolución.

**Recorte
de la Imagen** **Máscara
Rotada en 180°**

$$\begin{pmatrix} 1 & 2 & 3 \\ 6 & 7 & 8 \\ 11 & 12 & 13 \end{pmatrix} \cdot \begin{pmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{pmatrix} = 219$$

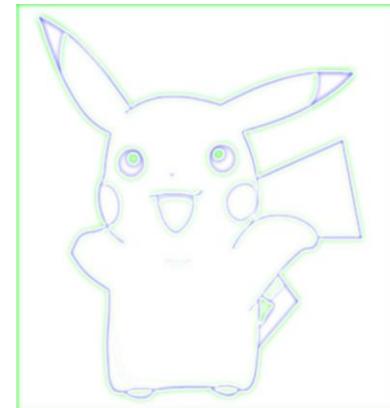
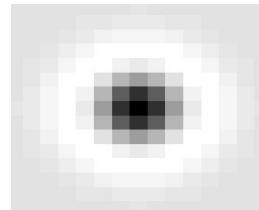
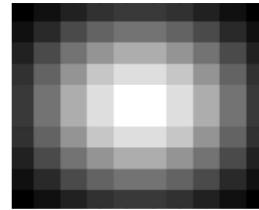
- Note que el resultado corresponde a la suma de las multiplicaciones entre el recorte y la máscara: $1 \cdot 9 + 2 \cdot 8 + 7 \cdot 3 + \dots = 219$

Input Image with Filter Overlaid (4 copies for clarity)



Result of Convolution

- Diferentes máscaras producen diferentes resultados (extracción de características)



Operación de convolución

- Observe como varían las características obtenidas al aplicar diferentes filtros de convolución



Original



Sharpen



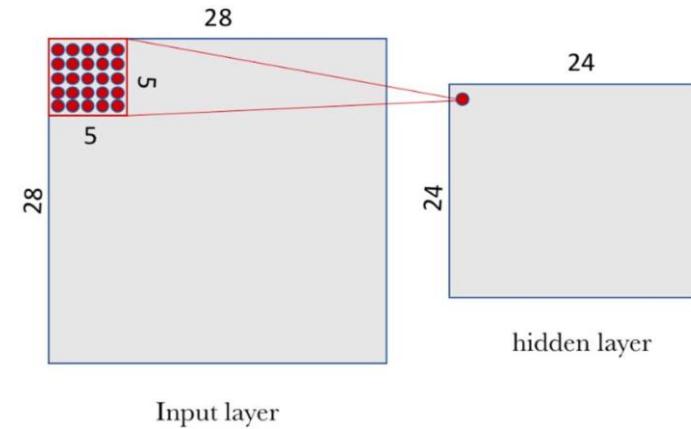
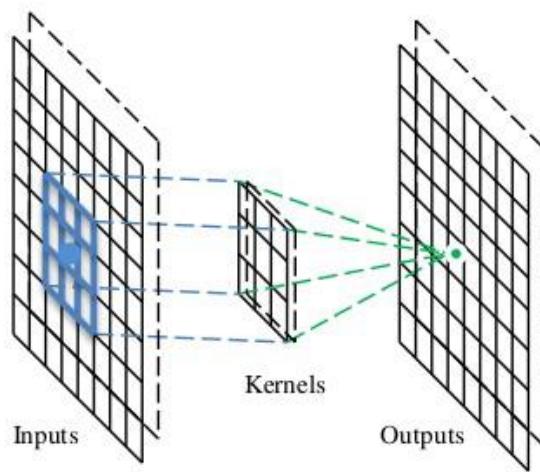
Edge Detect



“Strong” Edge
Detect

Operación de convolución

- En la convolución, cada pixel de salida es una combinación lineal de los pixeles de entrada, los cuales se calculan por medio de una ventana deslizante.
- Las máscaras representan la conectividad entre las capas sucesivas.
- Pesos = valores en las máscaras (lo que se entrena en la red).
- Las capas convolucionales pueden aprender jerarquías espaciales de patrones preservando relaciones espaciales.



Operación de convolución

- En general las capas convoluciones operan sobre tensores de 3D, llamados feature maps, con dos ejes espaciales de altura y anchura (height y width), además de un eje de canal (channels) también llamado profundidad (depth).
- En una imagen a color, la dimensión de profundidad (depth) es 3.
- En una imagen en escala de grises su depth es 1.



Imagen



Canal R



Canal G

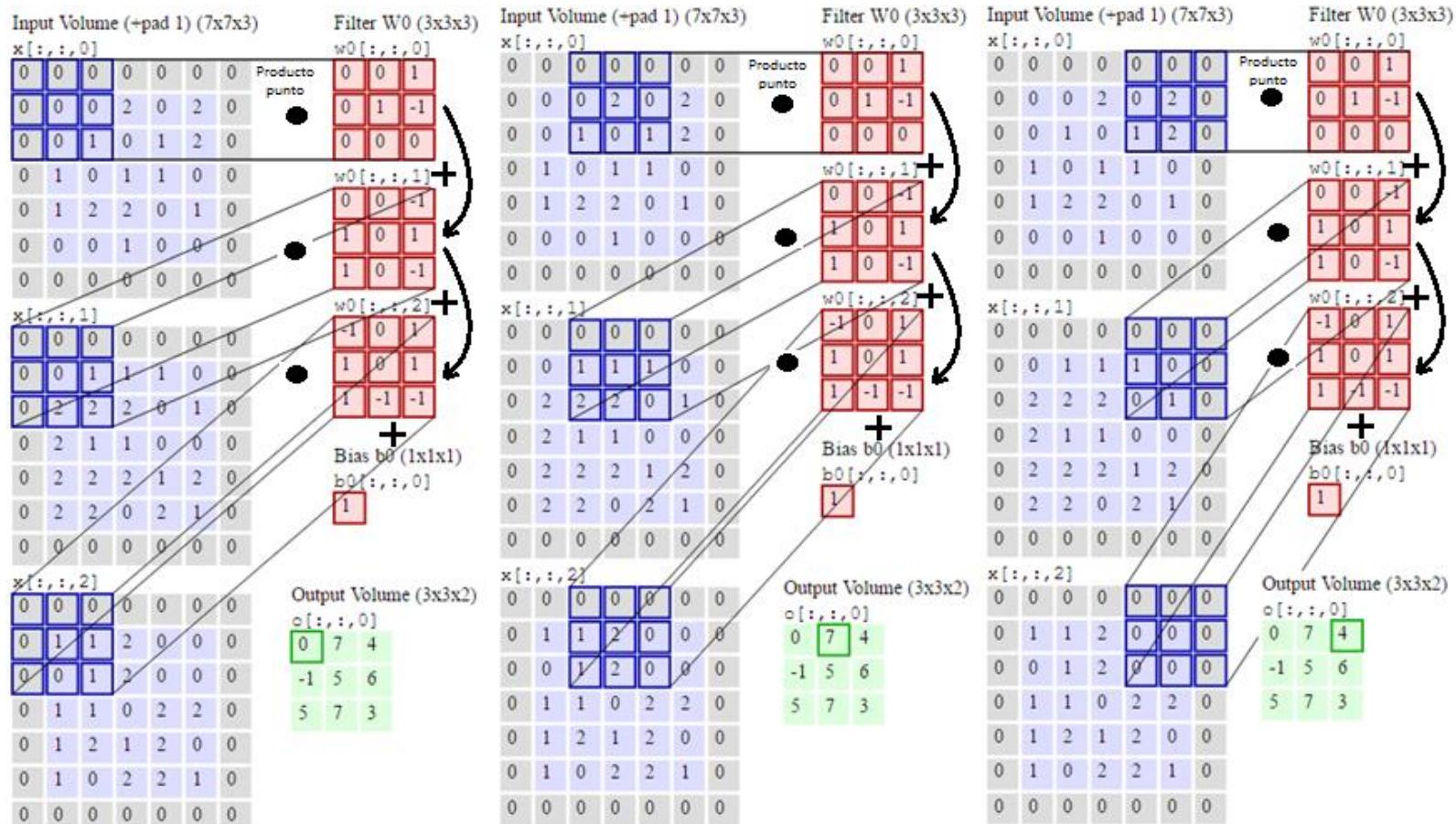


Canal B



Escala de grises

Ejemplo de convolución en RGB



La convolución efectúa producto punto elemento por elemento entre los filtros (rojo) y el input (azul), luego se suman los totales con el Bias, dando como resultado el output (verde).

Capas Convolucionales (CNN)

- Hiperparámetros:
 - **Depth:** número de filtros que queremos usar
 - **Stride:** salto entre convoluciones, si >1 se reduce el volumen espacial
 - **Zero-padding:** incrementa el borde del input con ceros, para ajustar tamaños de entradas y salidas. Puede existir diferentes valores de padding.
- Note que en el ejemplo anterior, el stride es de 2, puesto que después de la primera operación existe un corrimiento de 2 antes de continuar. Notar que si bien la imagen de entrada es de $5 \times 5 \times 3$, se usa un padding de 1, esto significa poner un contorno de ceros con grosor 1 rodeando el input, resultando en un input de $7 \times 7 \times 3$.

Capas Convolucionales (CNN)

- Hiperparámetros:

Padding

- A veces queremos obtener una imagen de salida de las mismas dimensiones que la entrada.
- Podemos añadir a la imagen de entrada una columna a la derecha, una columna a la izquierda, una fila arriba y un fila debajo de ceros.

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

0	0	0	0	0	0	0
0	1	2	3	4	5	0
0	6	7	8	9	10	0
0	11	12	13	14	15	0
0	16	17	18	19	20	0
0	21	22	23	24	25	0
0	0	0	0	0	0	0

Capas Convolucionales (CNN)

- Hiperparámetros:

Stride: número de pasos en que se mueve la ventada de los filtros.

Ej. Stride = 2

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

a

1	2	3
6	7	8
11	12	13

c

11	12	13
16	17	18
21	22	23

b

3	4	5
8	9	10
13	14	15

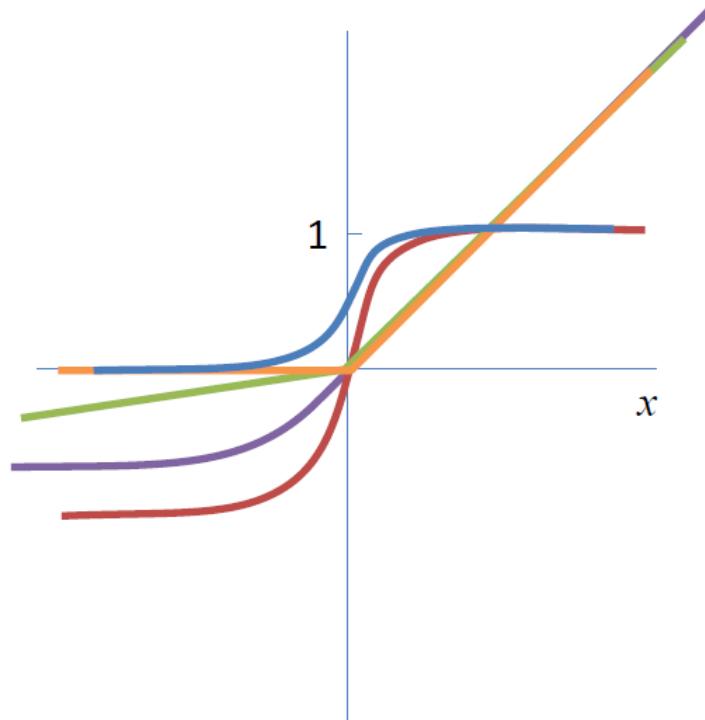
d

13	14	15
18	19	20
23	24	25

a	b
c	d

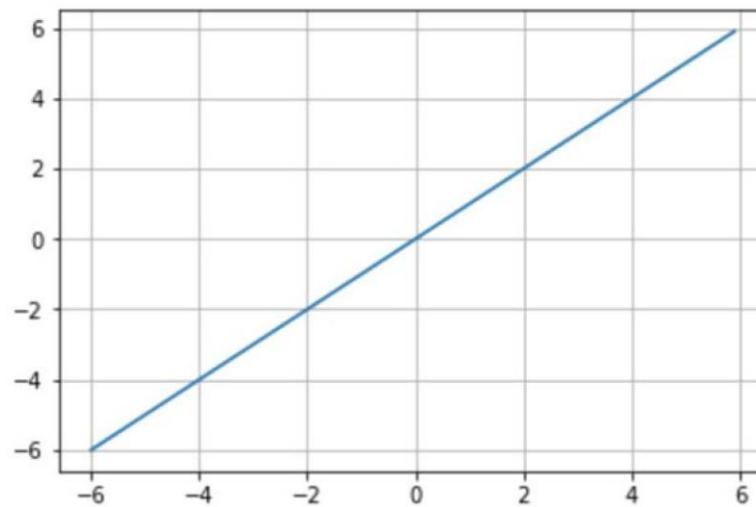
Funciones de activación típicas

- Sigmoidea: $\sigma(x) = \frac{1}{1+e^{-x}}$ 
- Tanh: $\tanh(x)$ 
- ReLU: $\max(0, x)$ 
- Leaky ReLU: $\max(0.1x, x)$ 
- ELU: $\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$ 
- Maxout: $\max(\omega'_1 x + b_1, \omega'_2 x + b_2)$



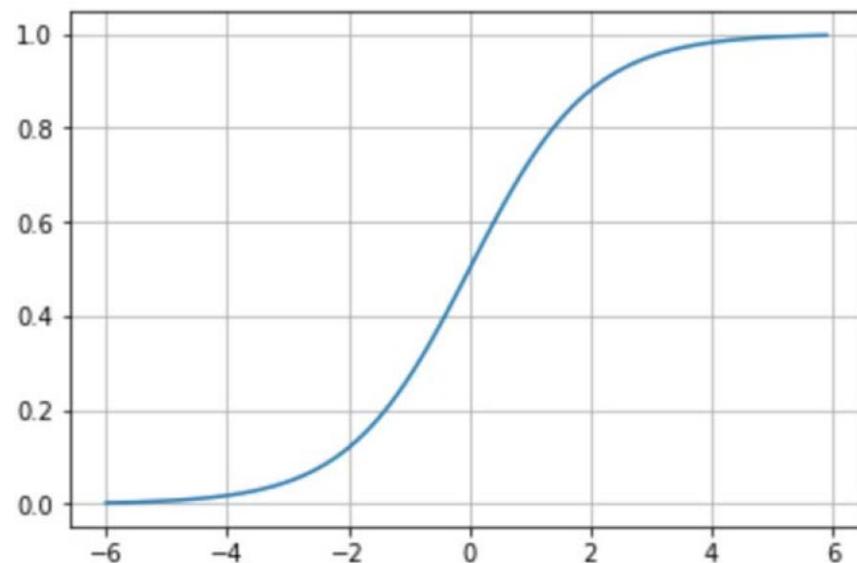
Funciones de activación típicas

- La función de activación lineal es básicamente la función identidad en la que, en términos prácticos, significa que la señal no cambia. En algunas ocasiones, se puede ver a esta función de activación en la capa de entrada de una red neuronal.



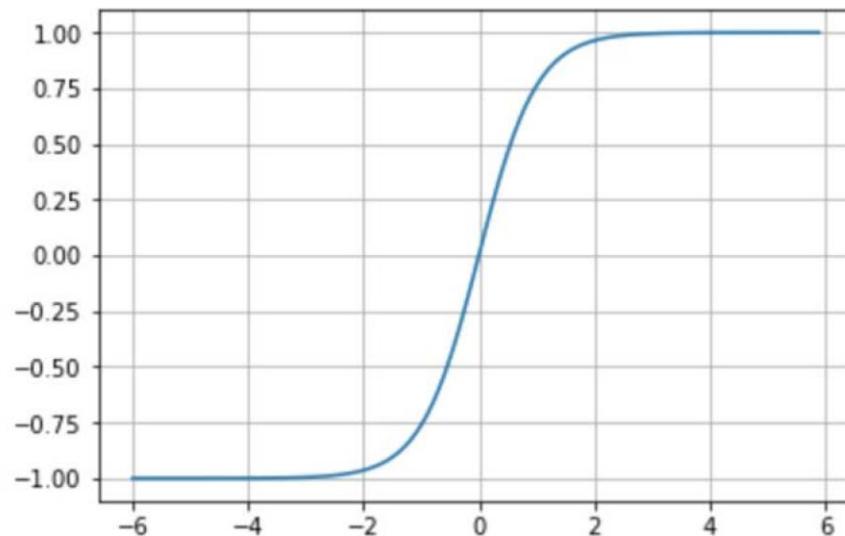
Funciones de activación típicas

- Una función sigmoidea convierte variables independientes de rango casi infinito en probabilidades simples entre 0 y 1. La mayor parte de su salida estará muy cerca de los extremos de 0 o 1 (resulta importante que los valores estén en este rango en algunos tipos de redes neuronales).



Funciones de activación típicas

- Una función tangente hiperbólica representa la relación entre el seno hiperbólico y el coseno hiperbólico: $\tanh(x) = \sinh(x)/\cosh(x)$. A diferencia de la función sigmoid, el rango normalizado de \tanh está entre -1 y 1.



Funciones de activación típicas

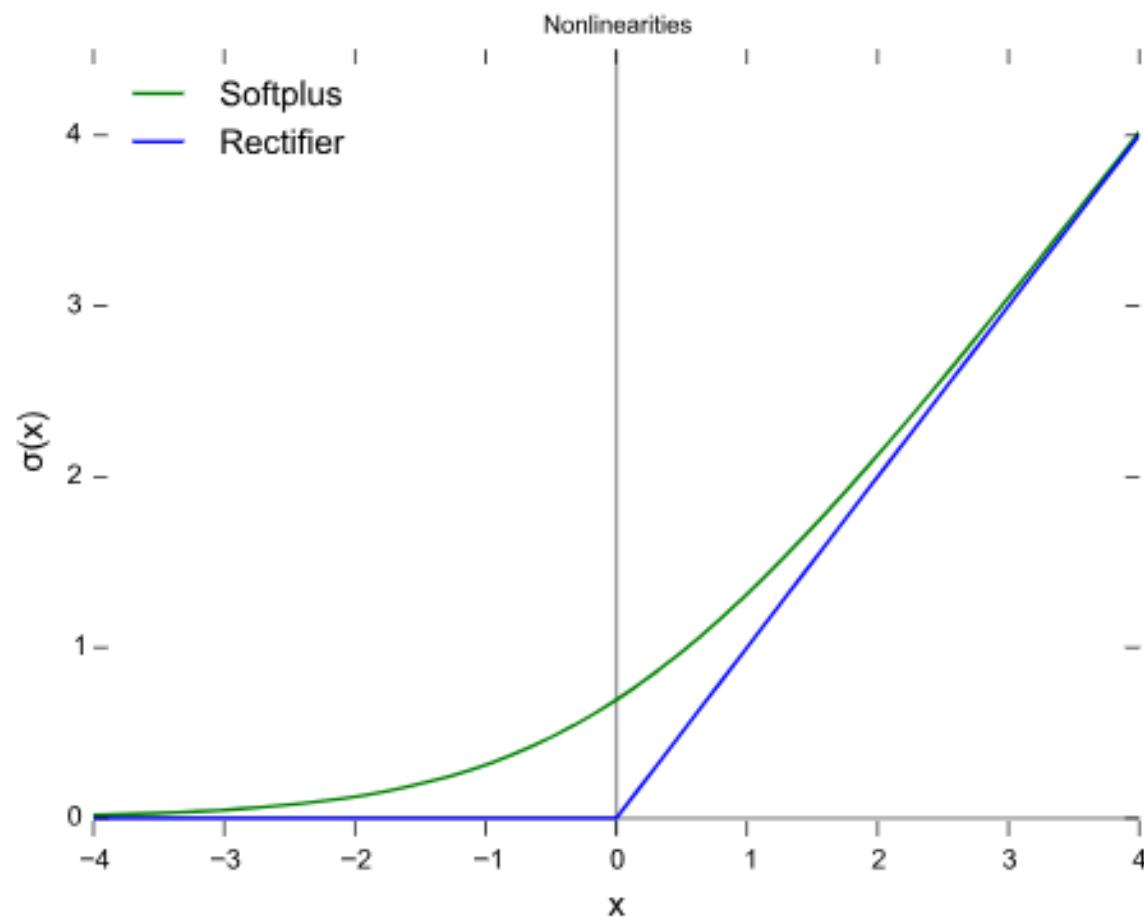
ReLU (Rectified Liner Unit)

- Para que cualquier clase de red neuronal sea poderosa, necesita contener no linealidades. Así, pasamos el resultado de la operación de convolución a través de la función de activación ReLU. Entonces, los valores en los mapas de características finales no son realmente las sumas, sino la función ReLU aplicada a ellas.
- Esta unidad se ha vuelto las más popular en la capa de activación. Se calcula con la siguiente función:

$$f(x) = \max(0, x)$$

- En otras palabras define un umbral desde cero al infinito, las ventajas importantes son:
 - Simplifica el error del Back-Propagation
 - Facilita el aprendizaje
 - Evita problemas de saturación

- ReLU



Input Feature Map



Black = negative; white = positive values

Rectified Feature Map

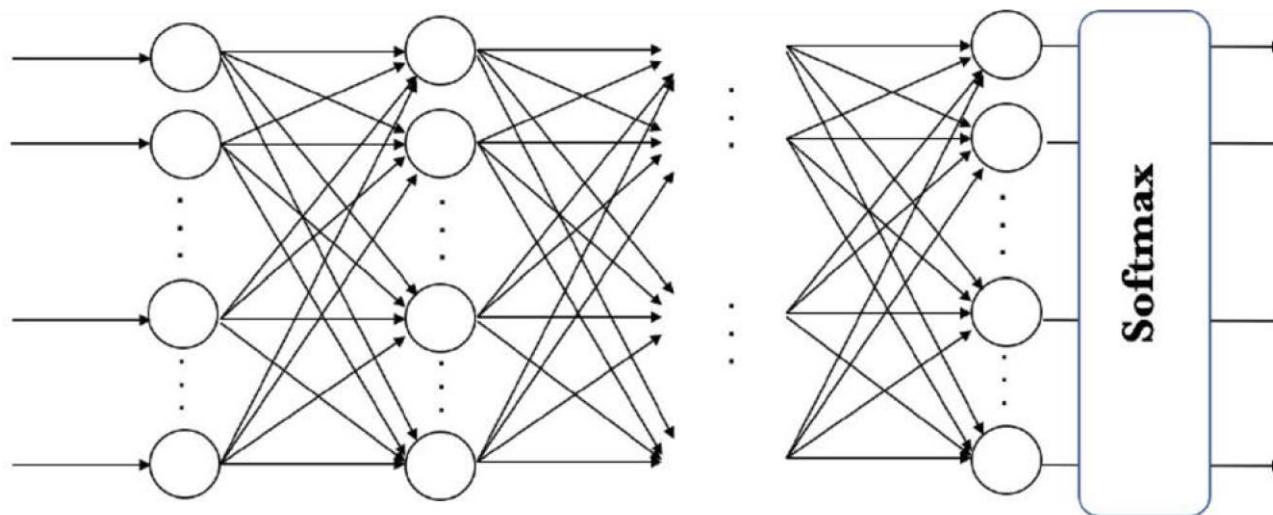


Only non-negative values

ReLU
→

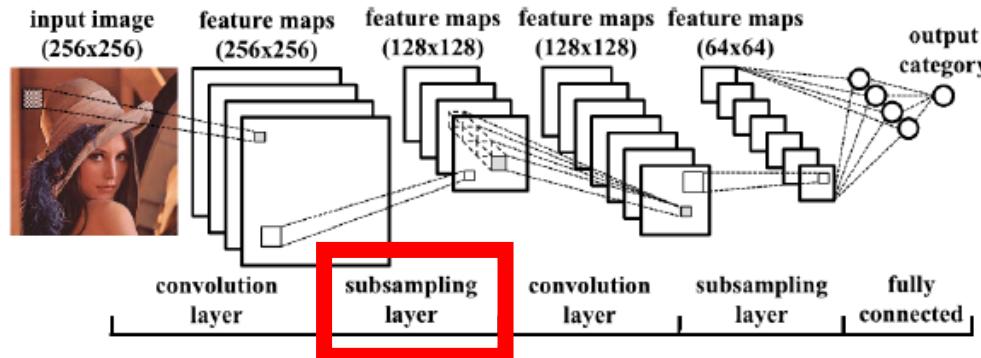
Funciones de activación típicas

- La función de activación softmax devuelve la distribución de probabilidad sobre clases de salida mutuamente excluyentes. Softmax se encontrará a menudo en la **capa de salida** de un clasificador.



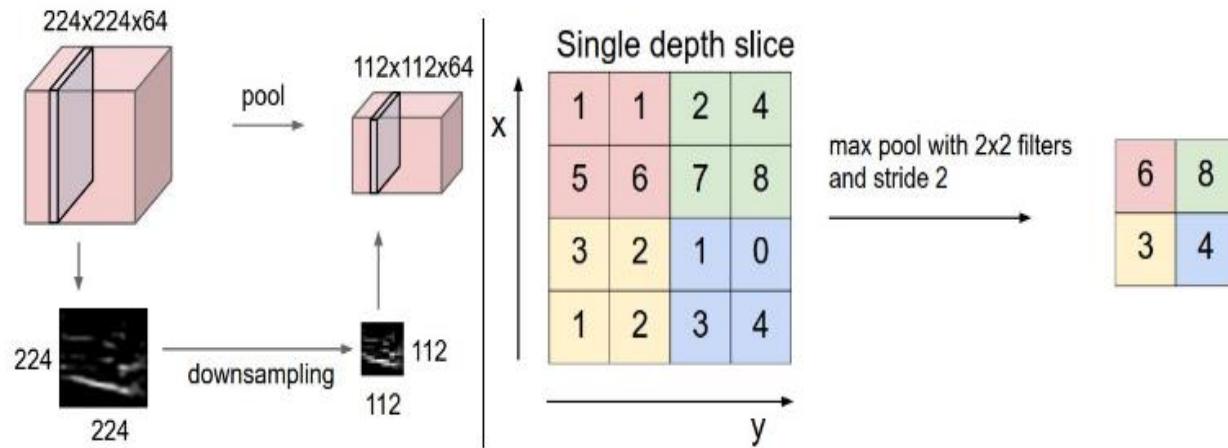
Subsampling-Pooling

- Se pueden agregar capas de pooling (submuestreo) que reducen el tamaño de la red.
- Es común el insertar periódicamente una capa Pooling entre Capas convolucionales sucesivas en una arquitectura de Red. Su función es reducir progresivamente el tamaño espacial de la representación para reducir la cantidad de parámetros y cálculos en la red, y por lo tanto, también controlar el overfitting.
- La pooling layer opera independientemente en cada pieza de profundidad de la entrada y modifica el tamaño espacial, usando por ejemplo la operación MAX.



Subsampling

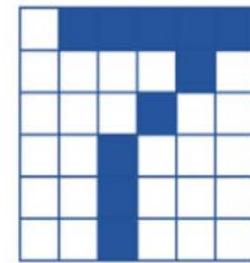
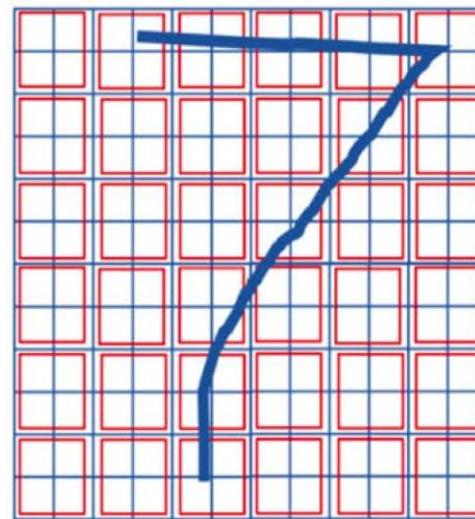
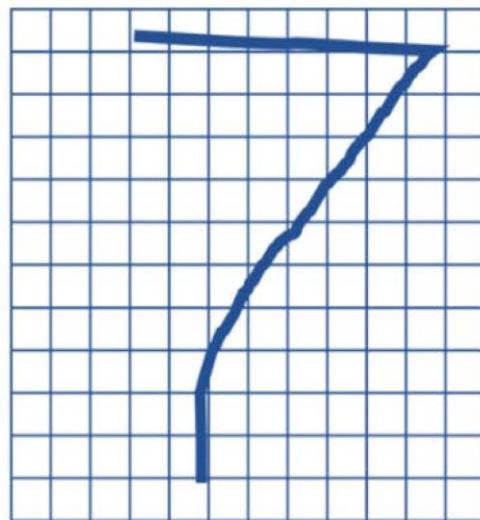
- Ejemplo:



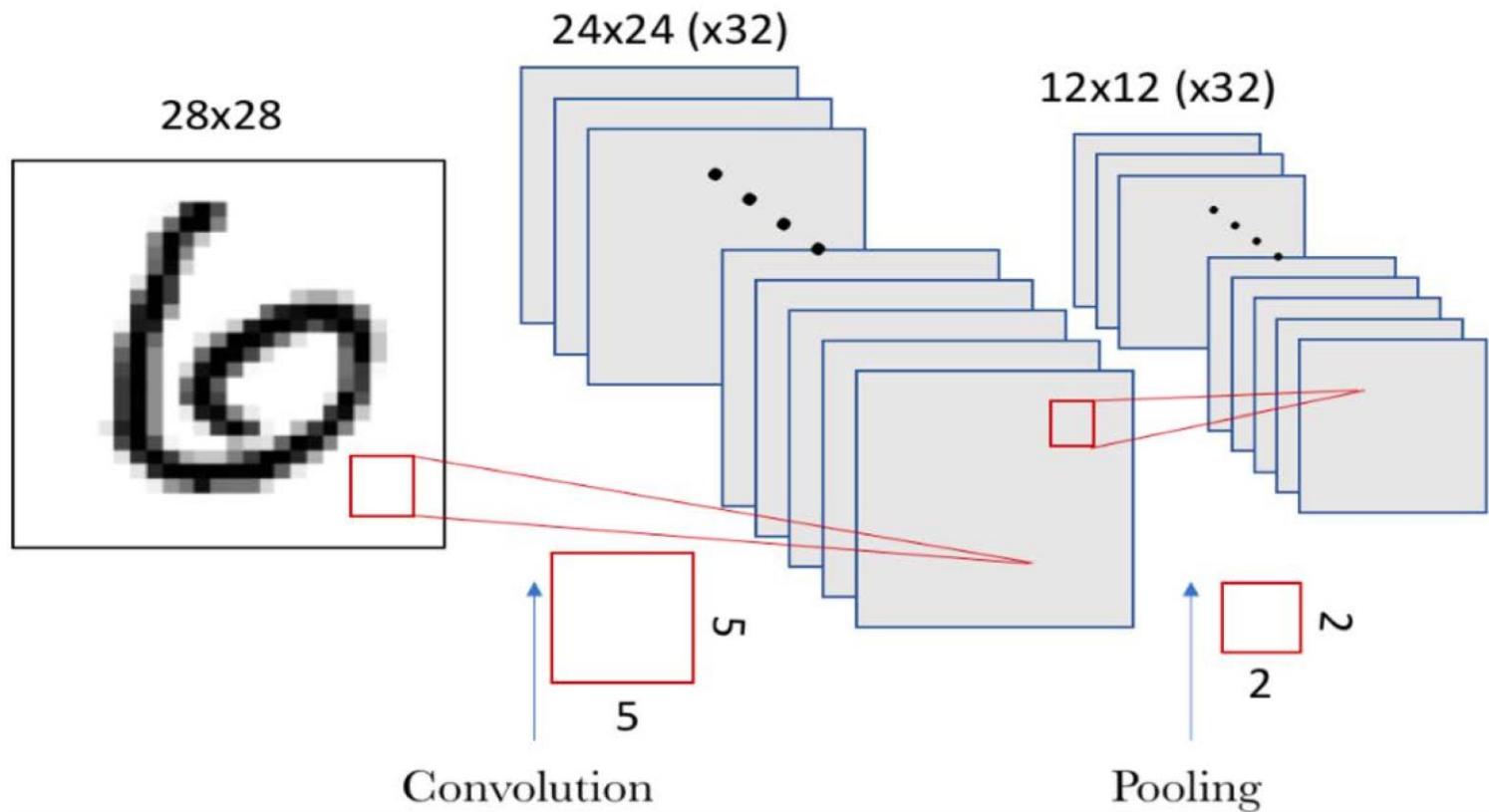
- Se aprecia claramente como se reducen las dimensiones del input gracias al pooling.
- A la derecha se ve como se extraen los valores máximos usando Max Pool.
- Se puede realizar el subsampling usando max, min, average, etc.

Subsampling

- Ejemplo:

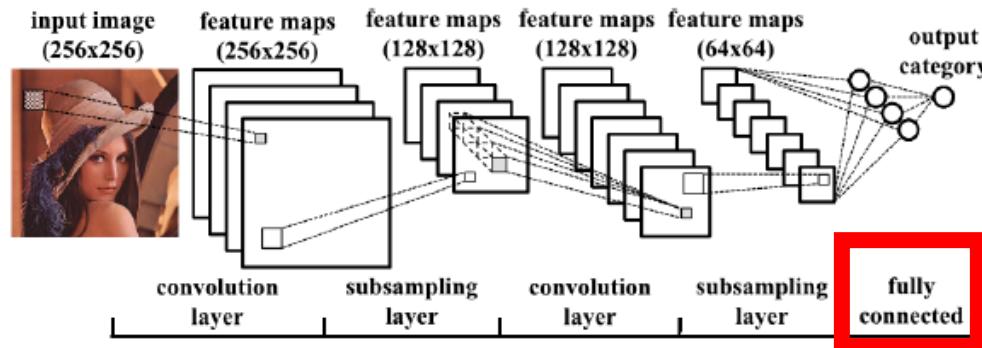


Resumen: Capa convolucional + Pooling



Fully-Connected Layer

- Neuronas con una capa totalmente conectadas que tienen conexiones completas a todas las activaciones en la capa previa, similar al caso de las redes neuronales clásicas.
- Sus activaciones pueden por tanto ser calculadas con una multiplicación de matrices seguida de un bias de compensación.
- Estas capas no buscan encontrar características, buscan clasificar.



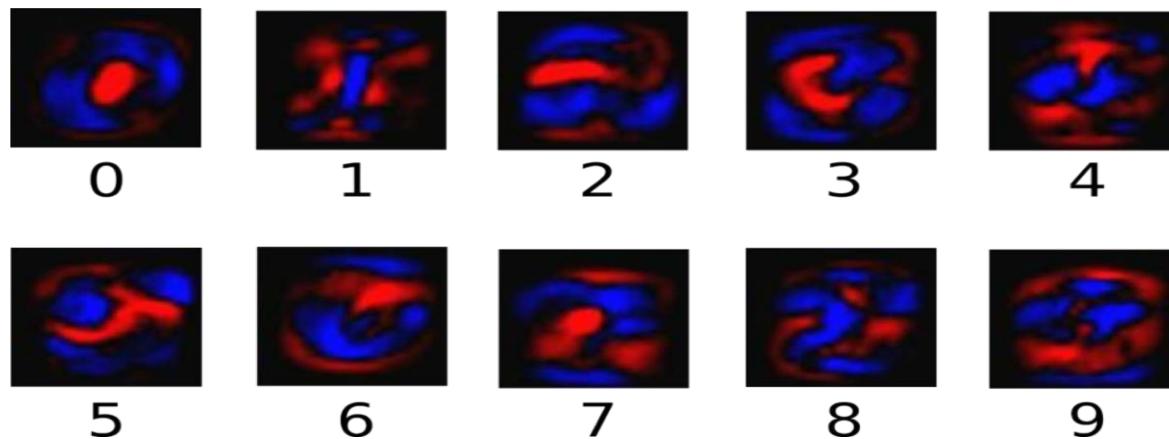
Softmax

- Los outputs de la red para el caso de un clasificador necesitamos que se adapten al rango (0, 1) y que sean una probabilidad (sumen 1).
- Para ello, utilizaremos la función softmax que ajusta las salidas de nuestra red al rango (0, 1).

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \text{ for } j = 1, \dots, K.$$

Softmax

- La función softmax se basa en calcular “las evidencias” de que una determinada imagen pertenece a una clase en particular y luego se convierten estas evidencias en probabilidades de que pertenezca a cada una de las posibles clases.
- Para medir la evidencia de que una determinada imagen pertenece a una clase en particular, una aproximación consiste en realizar una suma ponderada de la evidencia de pertenencia de cada uno de sus píxeles a esa clase.
- Ejemplo: Dígitos 0-9



Loss/Cost function

- Para poder usar backpropagation, definiremos una función: cost/loss/error function (*) que tiene que satisfacer dos propiedades:
 1. Se debe poder expresar como un promedio de costes para muestras de entrenamiento individuales.
 2. La función de costo no debe depender de ningún valor de activación por detrás de los de la última capa.

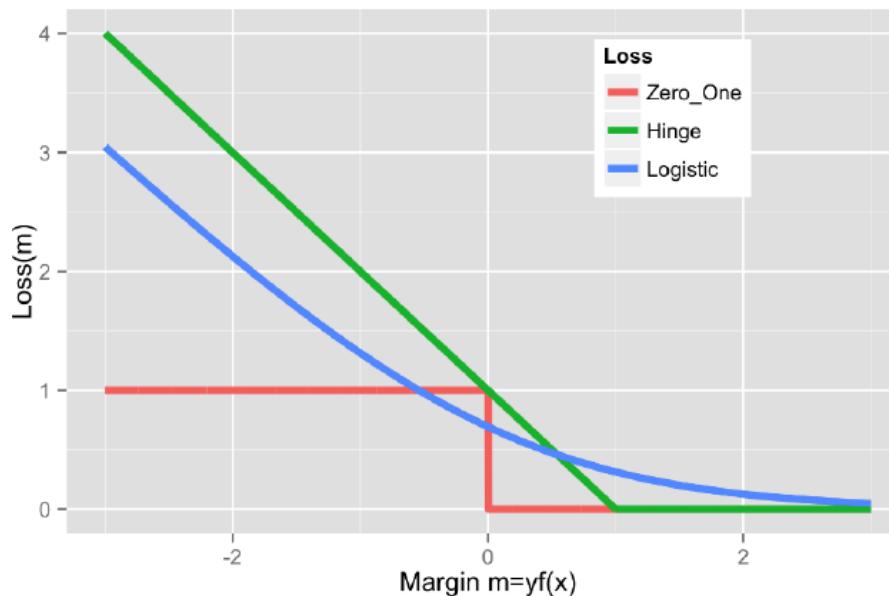
(*) en algunos contextos se distinguen éstas, siendo loss (pérdida) la función que evalúa el error para un dato concreto, y cost (costo) la que aglutina un conjunto de muestras y términos regularizadores.

Loss para clasificación

- Ejemplos:
- Logistic
- Hinge

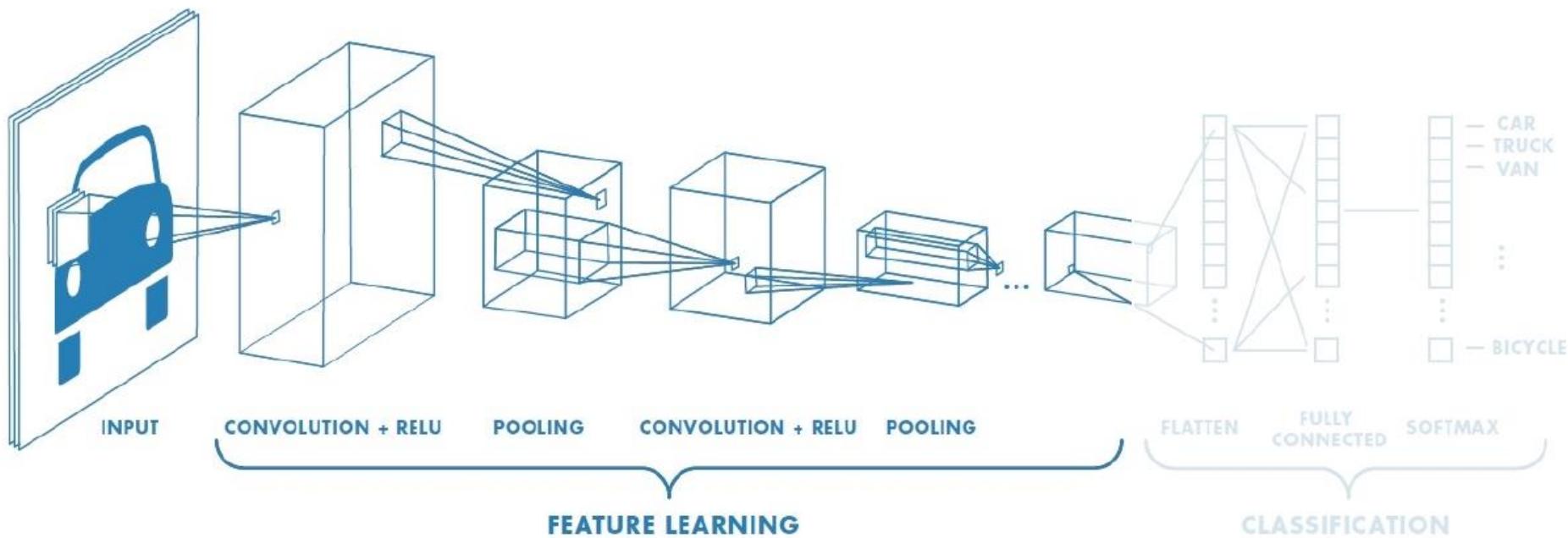
$$C = -y_j \log a_j^L - (1 - y_j) \log(1 - a_j^L)$$

$$C = \max(0, 1 - a_j^L y_j)$$



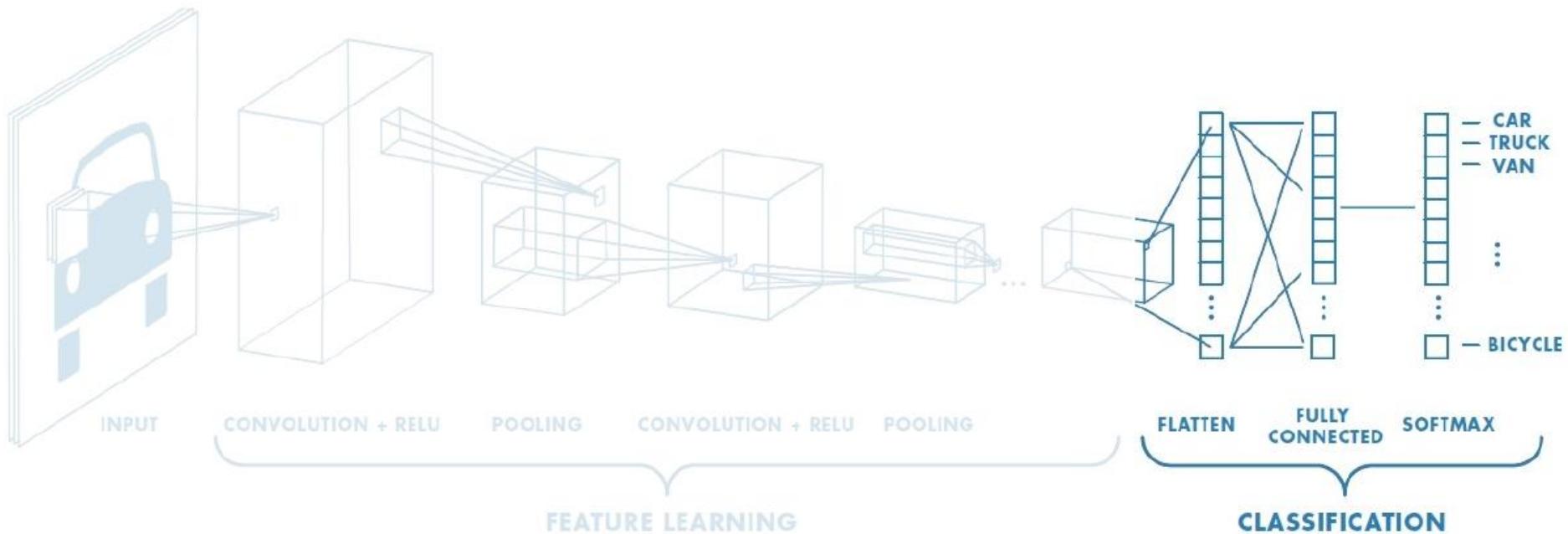
Resumen de CNN para clasificación

1. Aprender características en la imagen de entrada a través de convolución
2. Introducir la no linealidad mediante la función de activación (¡los datos del mundo real no son lineales!)
3. Reducir la dimensionalidad y preservar la invariancia espacial con la operación pooling.



Resumen de CNN para clasificación

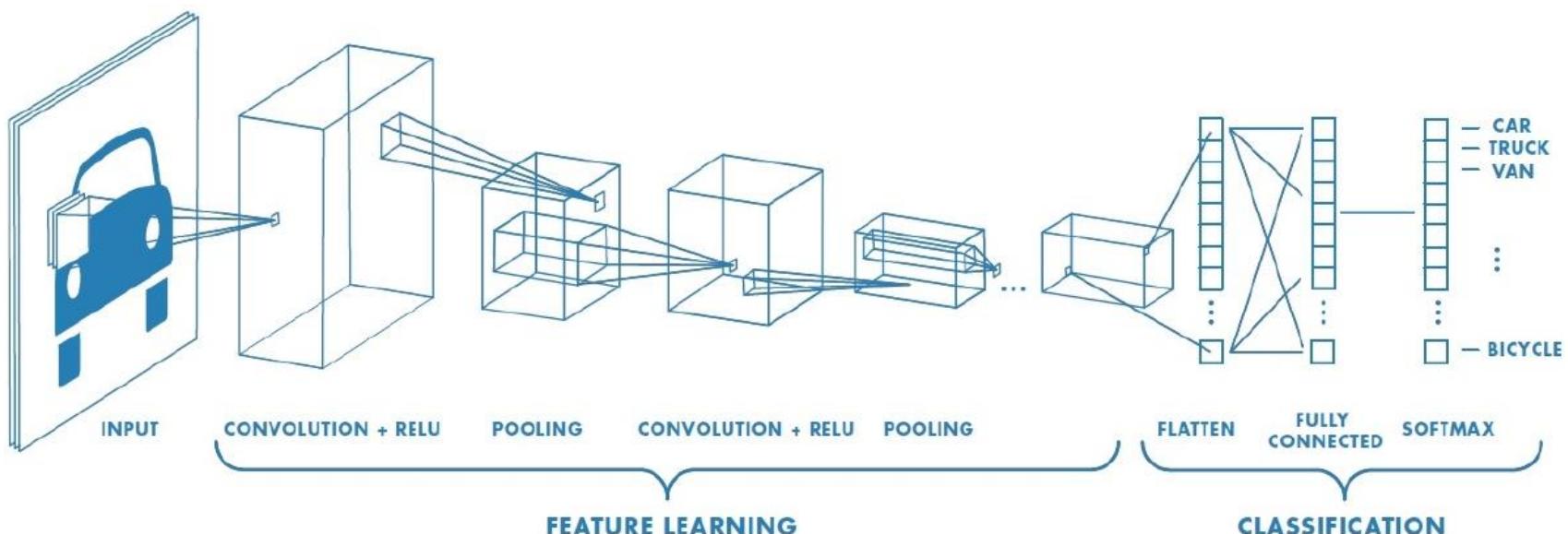
1. Las capas CONV y POOL generan características de entrada de alto nivel.
2. La capa totalmente conectada usa estas características para clasificar la imagen de entrada.
3. Se expresa la salida como probabilidad de que una imagen pertenezca a una clase en particular.



Resumen de CNN para clasificación

Entrenamiento con Backpropagation

- Aprende pesos para filtros convolucionales y capas totalmente conectadas



$$J(\theta) = \sum_i y^{(i)} \log(\hat{y}^{(i)})$$

Backpropagation: Cross-entropy Loss

¿Qué podemos clasificar?

- Existen muchas bases de datos con ejemplos etiquetados.
- Bases de datos:
- **MNIST**: 70K imágenes en grises 20x20. Representa dígitos escritos a mano 60K training, 10K testing.
- **CIFAR-10**: 60K imágenes en color 32x32, 10 clases, 6K/clase. 50K training, 10K test.
- **CIFAR-100**: lo mismo pero con 100 clases, 600/clase
- **ImageNet**: organizada según Word Net, 10M imagen en color, 10K categorías.
- **Places**: 2,44M imagen en color.



Arquitecturas actuales CNN

- AlexNet (2012):
- Entrenado con ImageNet ILSVRC-2012
 - 1.2 millones de imágenes
 - 1000 clases
 - Entrenado usando 2 GPUs durante una semana

input: 227x227

C1: 55x55x96

C2: 27x27x256

C3: 13x13x384

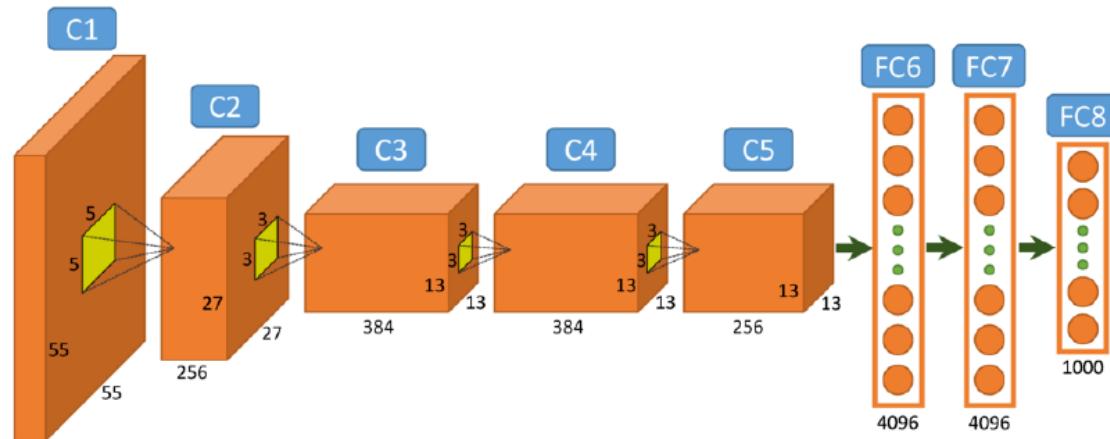
C4: 13x13x384

C5: 13x13x256

FC6: 4096

FC7: 4096

FC8 (output): 1000



Arquitecturas actuales CNN

- Zeiler (Clarifai) (2013):
- 5 Conv + 3 FC layer.

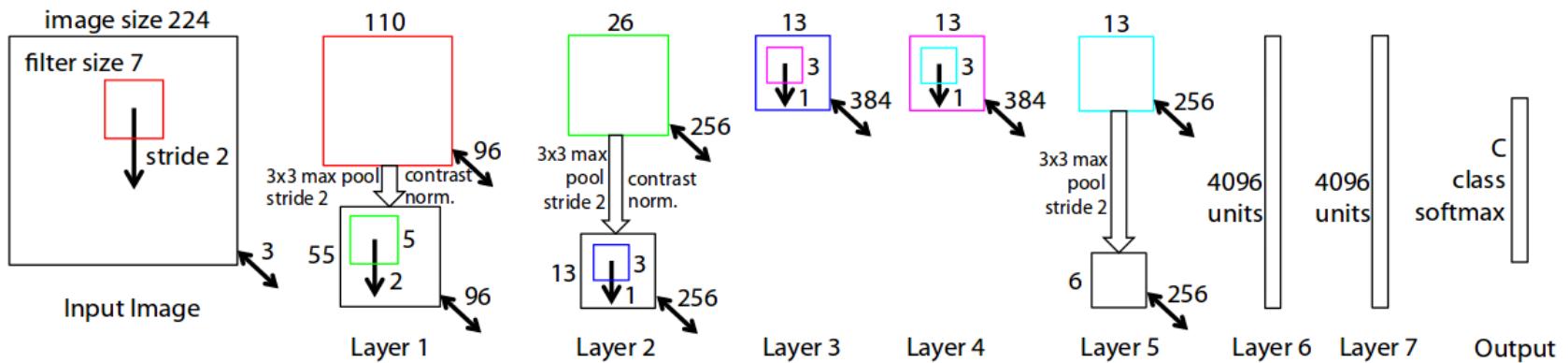
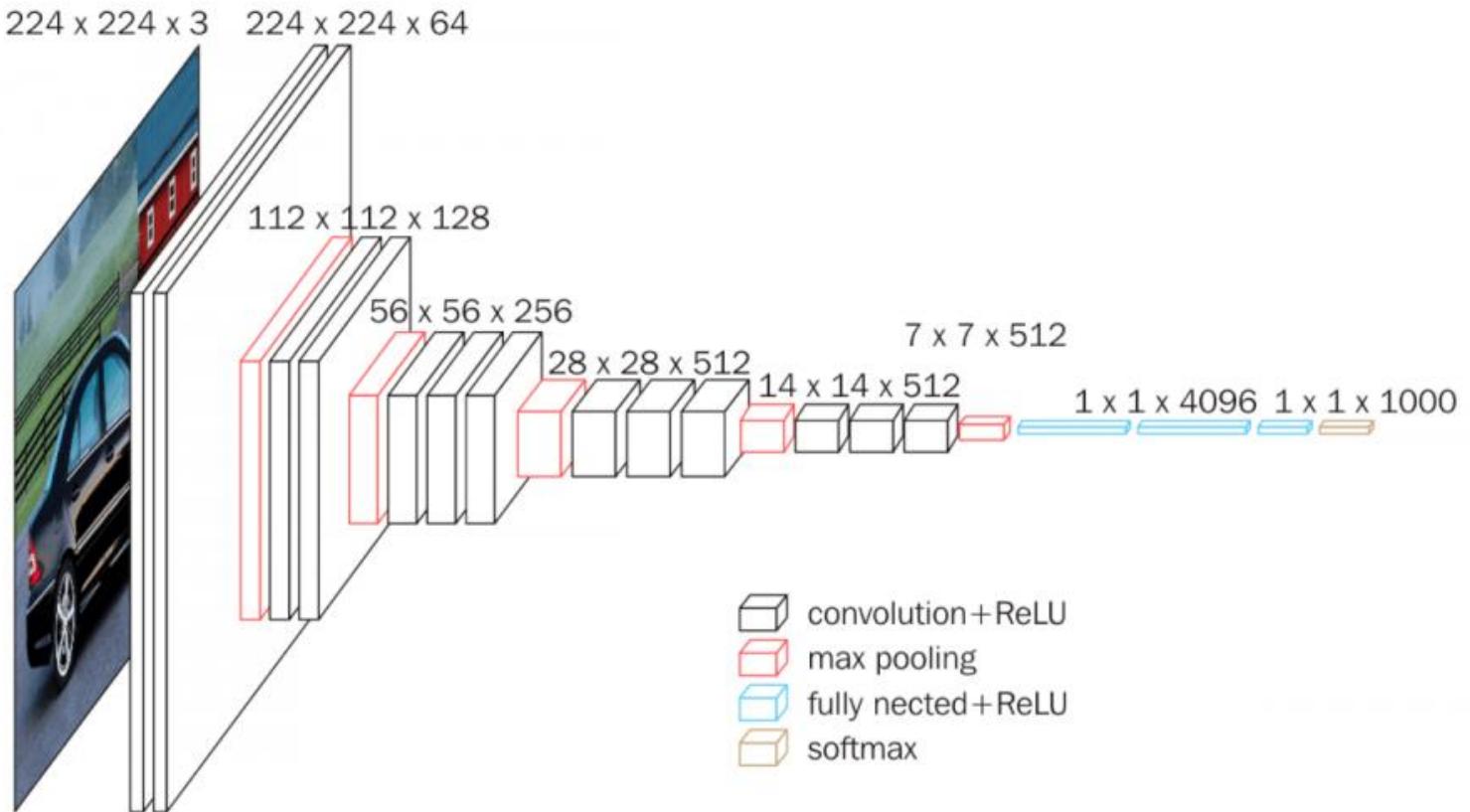


Figure 3. Architecture of our 8 layer convnet model. A 224 by 224 crop of an image (with 3 color planes) is presented as the input. This is convolved with 96 different 1st layer filters (red), each of size 7 by 7, using a stride of 2 in both x and y. The resulting feature maps are then: (i) passed through a rectified linear function (not shown), (ii) pooled (max within 3x3 regions, using stride 2) and (iii) contrast normalized across feature maps to give 96 different 55 by 55 element feature maps. Similar operations are repeated in layers 2,3,4,5. The last two layers are fully connected, taking features from the top convolutional layer as input in vector form ($6 \cdot 6 \cdot 256 = 9216$ dimensions). The final layer is a C -way softmax function, C being the number of classes. All filters and feature maps are square in shape.

Arquitecturas actuales CNN

- VGG 16.



Arquitecturas actuales CNN

- VGG 16.

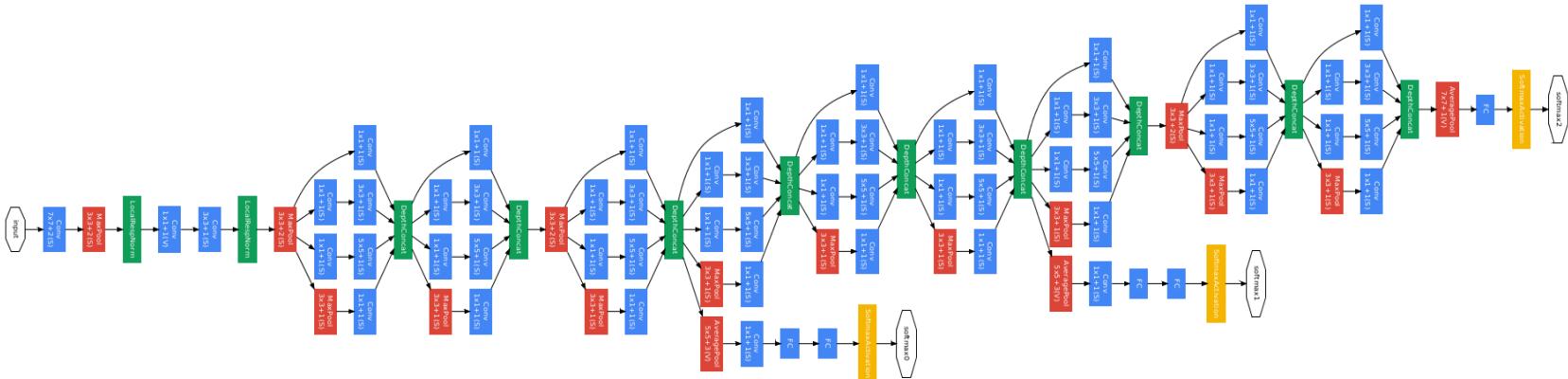
Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808

block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000

Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0

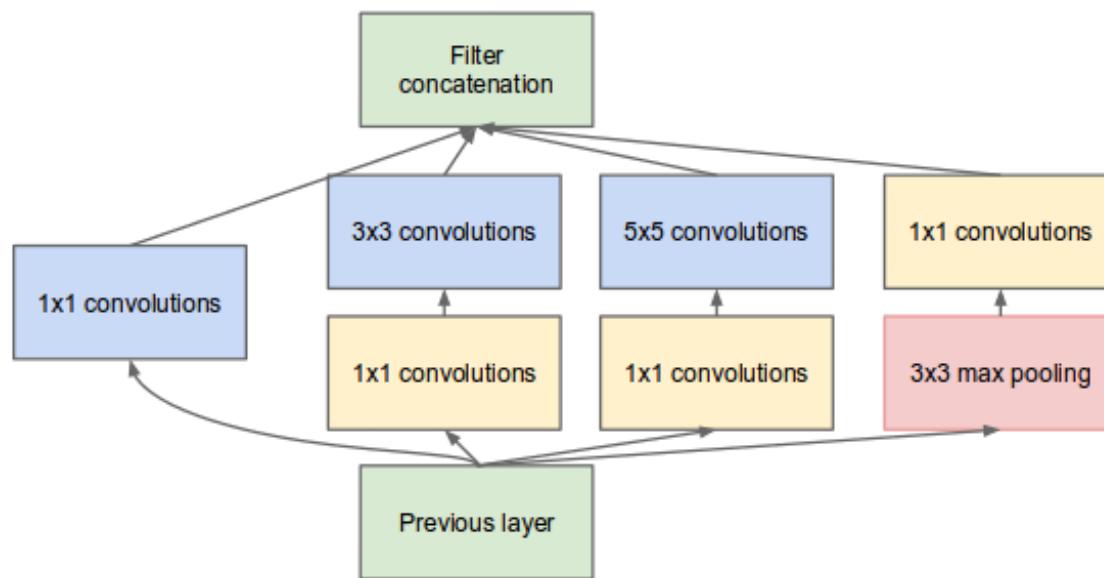
Arquitecturas actuales CNN

- GoogLeNet
- 22 layers
- Arquitectura final no es una sucesión de convoluciones.
- De este modelo derivan redes actuales como Inception, Xception.
- Neurona con modelo más complejo
 - Varios cálculos en paralelo
 - Pooling sin reducir la imagen



Arquitecturas actuales CNN

- GoogLeNet
- Se usan varias resoluciones



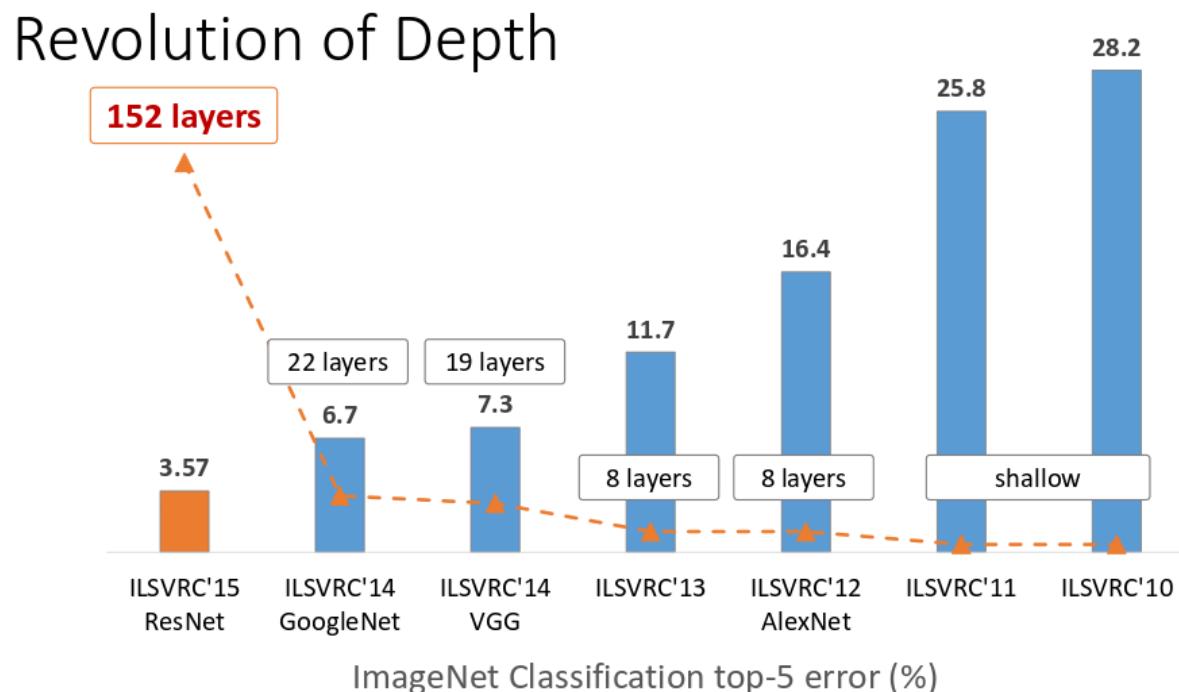
(b) Inception module with dimensionality reduction

Arquitecturas actuales CNN

- GoogLeNet

Arquitecturas actuales CNN

- ResNet (MSRA 2015)
 - Estado de arte en CNN
 - “Deep residual learning”
 - 152 layers en ICCV 2015
 - 1001 layers actualmente

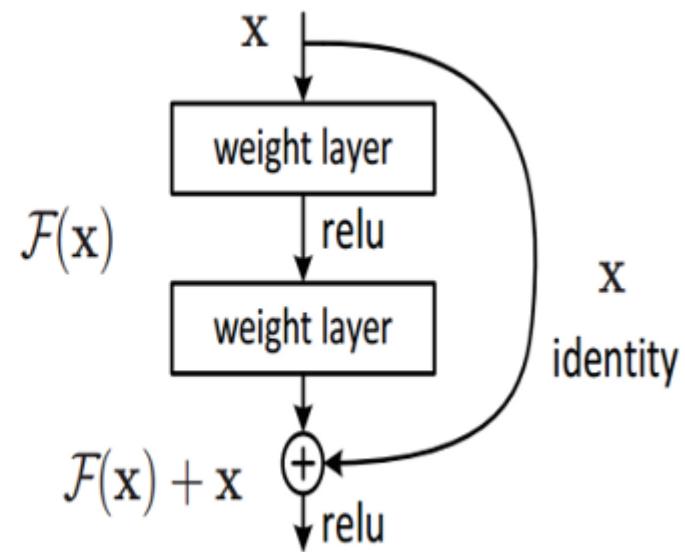
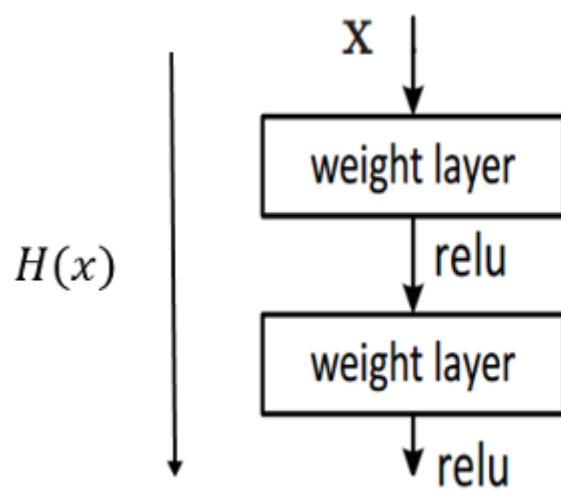


Arquitecturas actuales CNN

- ResNet (MSRA 2015)
 - Al agregar más de 30 capas, el error comienza a aumentar en vez de disminuir.
 - Esto es la causa de que haya que usar modelos complejos como GoogleNet.
 - Se encontró una solución a ese problema.
 - La idea es que las capas extras reduzcan efectivamente el error
 - Así, se trabaja con **residuales**
 - A diferencia de GoogLeNet, el diseño es simple
 - Solo se usan capas convolucionales y max pooling
 - El uso de residuales permite usar muchas capas (actualmente 1001 capas)

Arquitecturas actuales CNN

- ResNet (MSRA 2015)

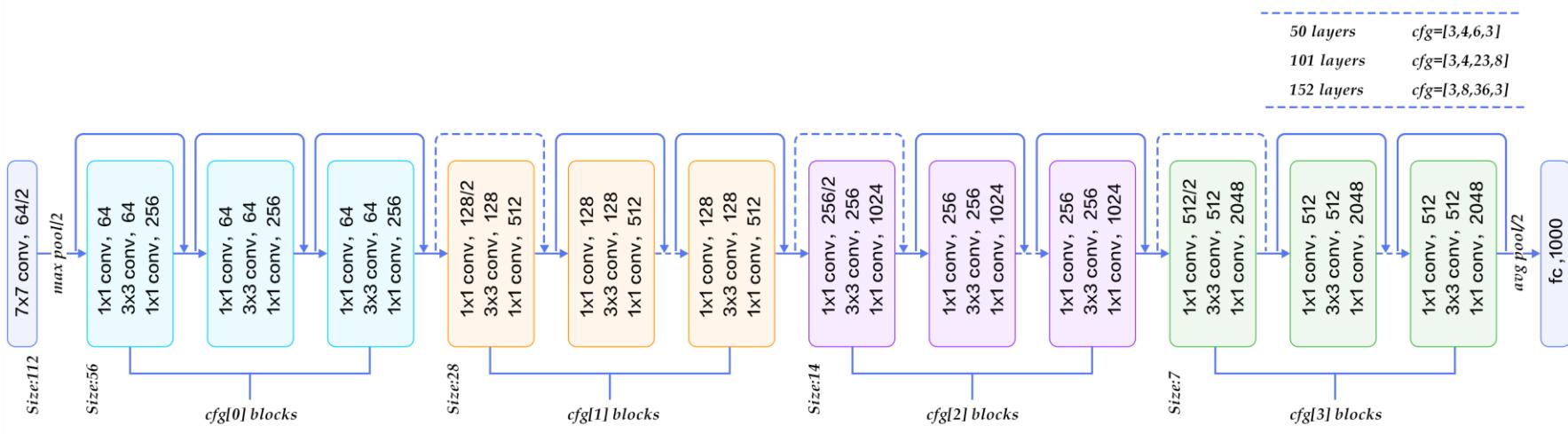


Arquitecturas actuales CNN

- ResNet
- Más capas es mejor, pero debido al problema del gradiente desvaneciente, los pesos de los modelos de las primeras capas no pueden actualizarse correctamente a través del backpropagation (la regla de la cadena multiplica los valores del gradiente de error más bajos que uno y luego, cuando el error de gradiente llega a las primeras capas su valor se va a cero).
- El objetivo de Resnet es preservar el gradiente.
- ¿Cómo? Gracias a la matriz de identidad porque: “¿qué pasaría si tuviéramos que propagar a través de la función de identidad? Entonces el gradiente simplemente se multiplicaría por 1 y no le pasaría nada”.
- Incluso se puede olvidar el problema del gradiente desvaneciente y simplemente mirar la imagen de una red Resnet: la matriz de identidad transmite los datos de entrada que evitan la pérdida de información (el problema de la fuga de datos).

Arquitecturas actuales CNN

- ResNet (MSRA 2015)

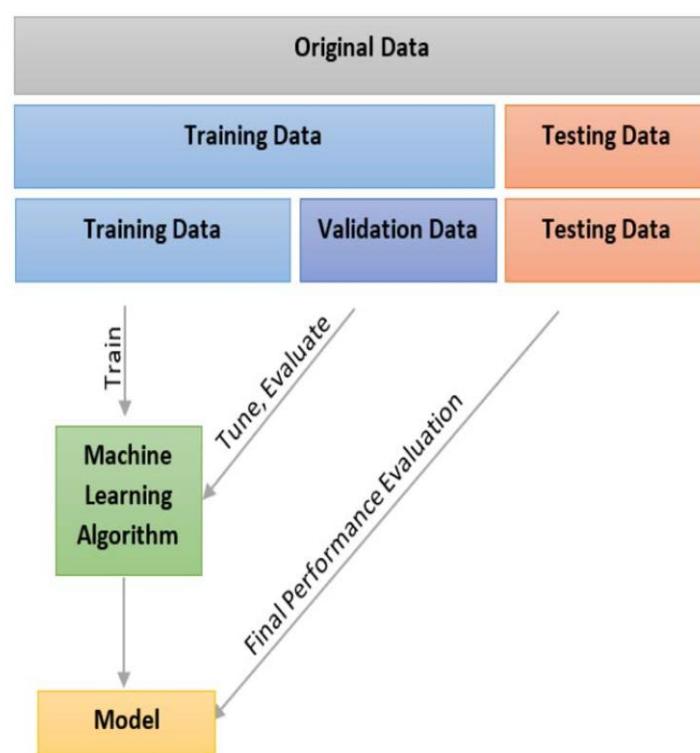


Conjuntos de: Training, Validation, Test

- **Training (entrenamiento):** datos usados para aprender. La diferencia de las respuestas de la red a estos datos con respecto a las salidas esperadas, es la diferencia que propagaremos hacia atrás.
- **Validation (validación):** la manera correcta de sintonizar los hiperparámetros sin tocar el resto de los datos. La idea es partir el training en dos: uno mayor para entrenar y otro para validar. Podemos evaluar la accuracy obtenida cada cierto tiempo y ver cómo evoluciona en el tiempo.
- **Test (testeo):** Se evalúan una única vez al final. **No podemos usar el test para aprender la red o sus hiperparámetros.**
- En el caso de tener pocas muestras y no poder partir en 3 conjuntos se puede optar por hacer una cross-validation (validación cruzada).

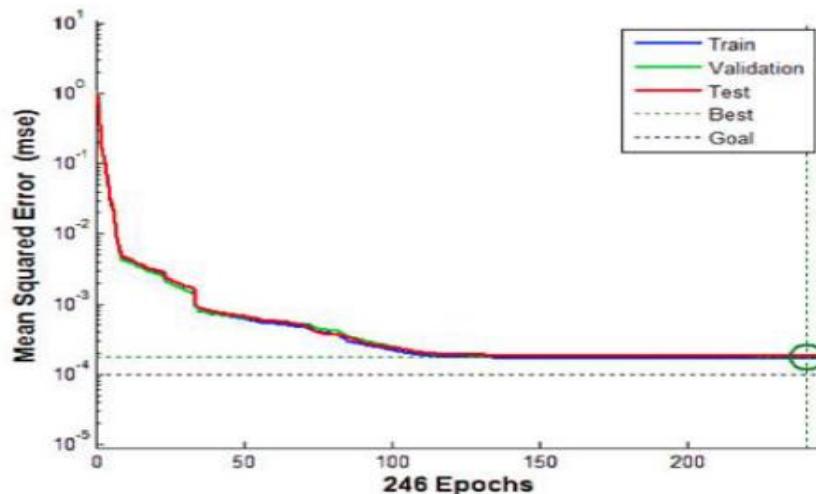
Conjuntos de: Training, Validation, Test

- Set de entrenamiento (entrenar)
- Set de validación (afinar hiperparámetros)
- Set de prueba (probar el desempeño del modelo)

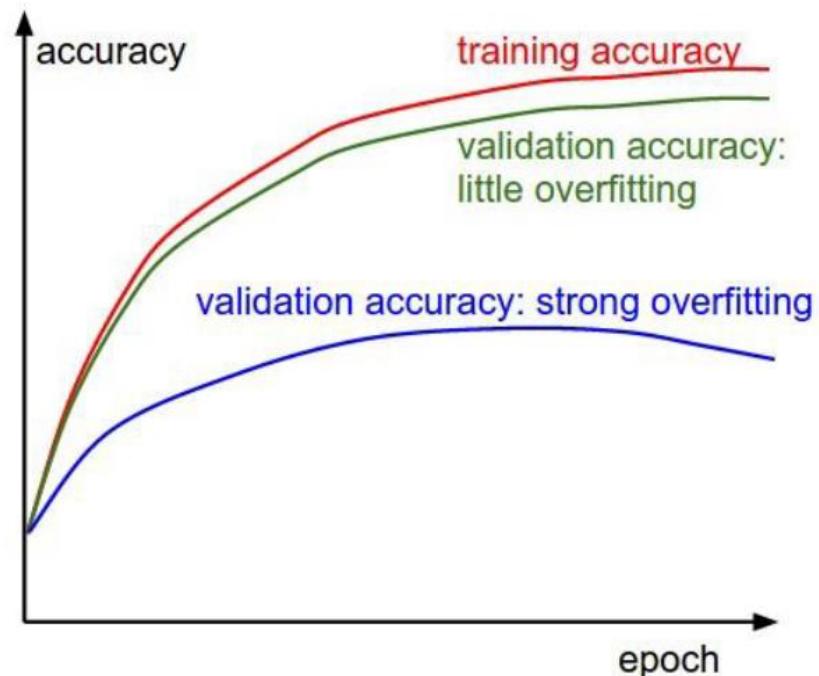
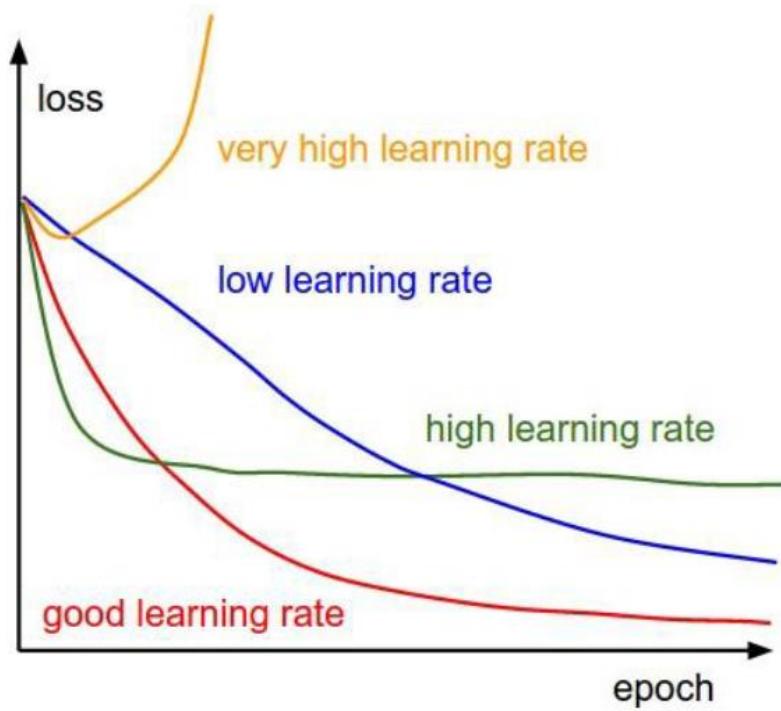


Parámetros: Epoch, batch, iterations

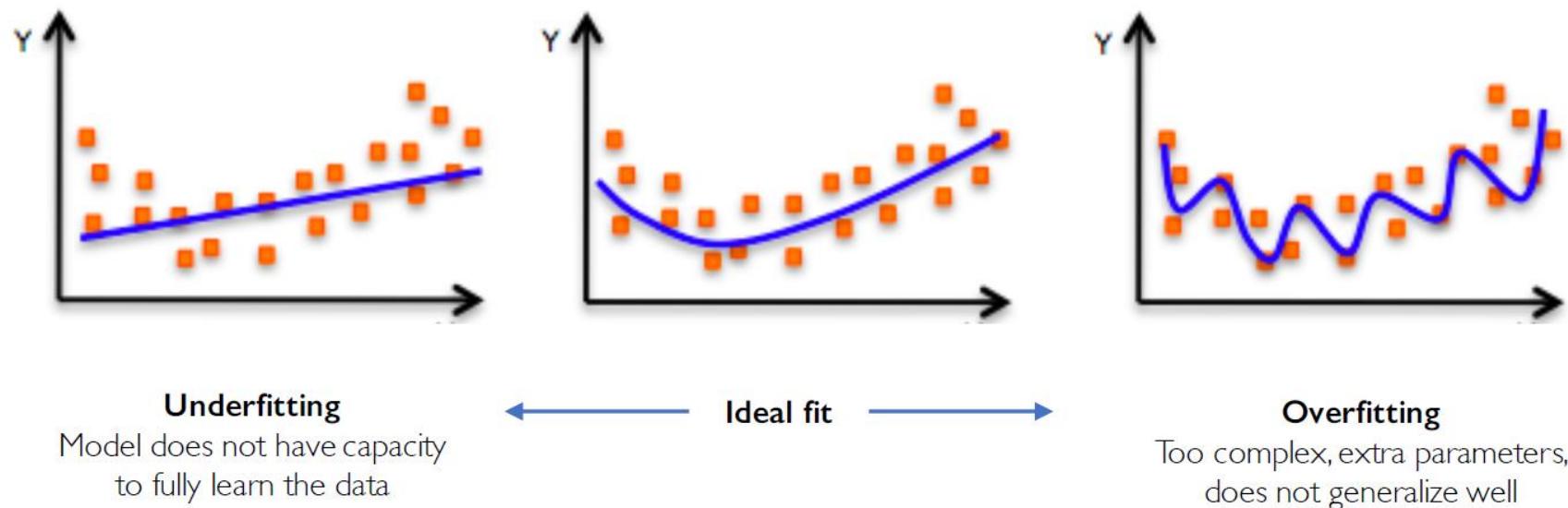
- **Epoch (épocas):** una pasada de los datos (ida y vuelta) forward y backward de todos los datos de entrenamiento. Nos indica el número de veces en las que todos los datos de entrenamiento han pasado por la red neuronal en el proceso de entrenamiento.
- **Batchsize (lotes):** número de muestras de entrenamiento en un paso forward/backward. Cuanto mayor sea este batchsize mejor representado estará el gradiente pero más memoria se necesitará.
- **Iterations (iteraciones):** número de pasos, cada uno usa batchsize número de ejemplos.



Evolución temporal de un aprendizaje

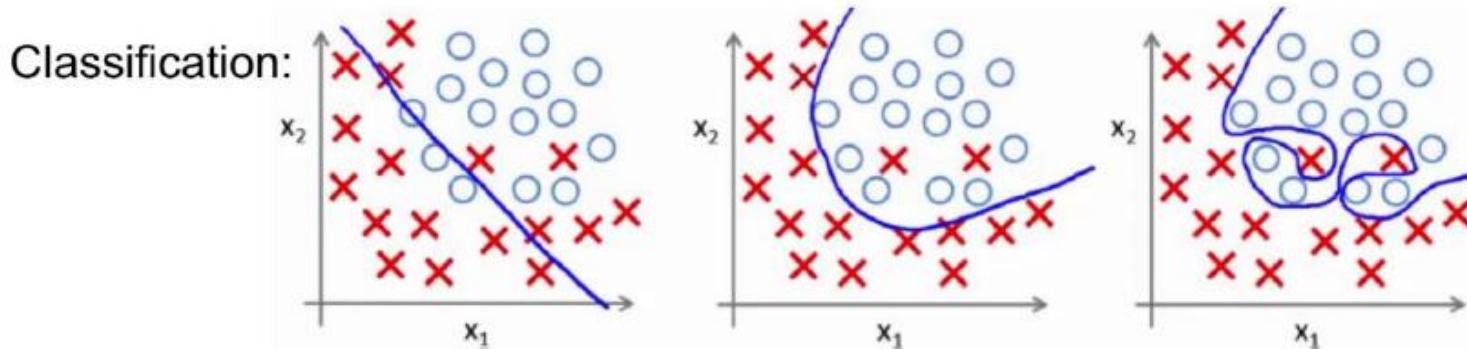


El problema del overfitting (sobre-entrenamiento)



Regularización

- Técnica que limita nuestro problema de optimización para desalentar modelos complejos. Se utiliza para mejorar la generalización del modelo ante datos que no ha visto.
- Hacer más robusto el aprendizaje, reducir el overfitting (sobreajuste en los datos). Contamos con varias técnicas:
 - Data augmentation
 - Early stopping
 - Dropout
 - Batch normalization
 - Weight decay



Data augmentation

- Generar más muestras plausibles a partir de:
 - Ligeras traslaciones (random crops)
 - Simetrías horizontales y verticales
 - Rotaciones
 - Manipulación del color
 - Distorsiones geométricas
 - Añadir ruido a los datos
 - Reconstruir nuevas imágenes



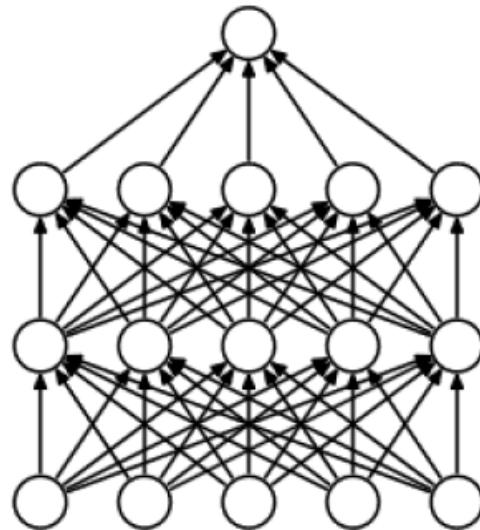
Early stopping

- Detener el entrenamiento antes de que tengamos la oportunidad de sobreajuste.
- Si el training genera overfitting de sus datos, y la red acaba ajustando muy bien (demasiado) las respuestas del training. Si esto sucede, la red se equivocará más en un caso no visto. Lo que haremos es vigilar y parar cuando el “validation error” comience a subir.

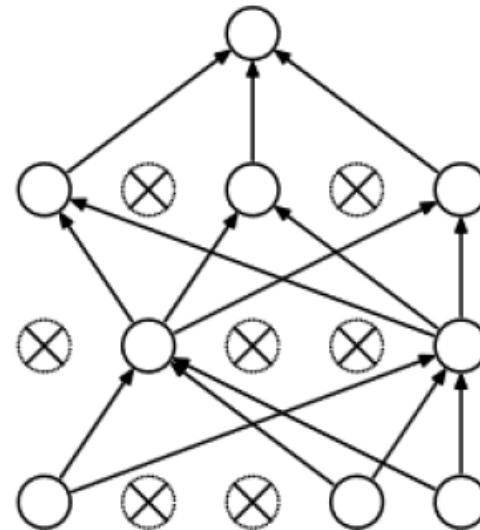


Dropout

- Es una técnica de regularización para reducir el sobreajuste en redes neuronales mediante la prevención de adaptaciones complejas en los datos de entrenamiento. Es una forma muy eficiente de realizar un promedio de modelos con redes neuronales.
- El término "abandono" se refiere al abandono de unidades (tanto ocultas como visibles) en una red neuronal.



(a) Standard Neural Net



(b) After applying dropout.

Batch normalization

- Es un método para reducir el desplazamiento de las co-variables internas en las redes neuronales, lo que conduce al posible uso de mayores tasas de aprendizaje. En principio, el método agrega un paso adicional entre las capas, en el que la salida de la capa anterior se normaliza.
- Se utiliza para normalizar las entradas en cada capa utilizando la media y la varianza en lotes pequeños.

```
Input: Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_1 \dots m\}$ ;  
Parameters to be learned:  $\gamma, \beta$   
Output:  $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$ 
```

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$
$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$
$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

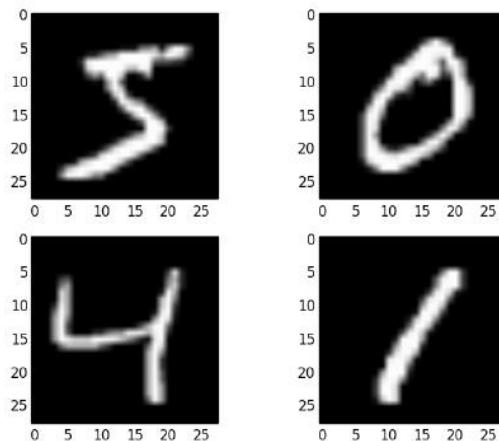
Weight decay

- Se define como la multiplicación de cada peso en el descenso del gradiente en cada época por un factor más pequeño que uno y mayor que cero.
- Esta técnica es equivalente a la introducción de un término de regularización a la función de costo que uno quiere optimizar.
- Evita que la importancia del resultado recaiga en pocas neuronas limitando el peso de éstas.
- Se utiliza para evitar el overfitting y ayudar a escapar de máximos locales.
- Añadimos a la función de coste un término que penaliza que los pesos se hagan grandes, por ejemplo:

$$\tilde{C} = C + \frac{\lambda}{2} W^2$$

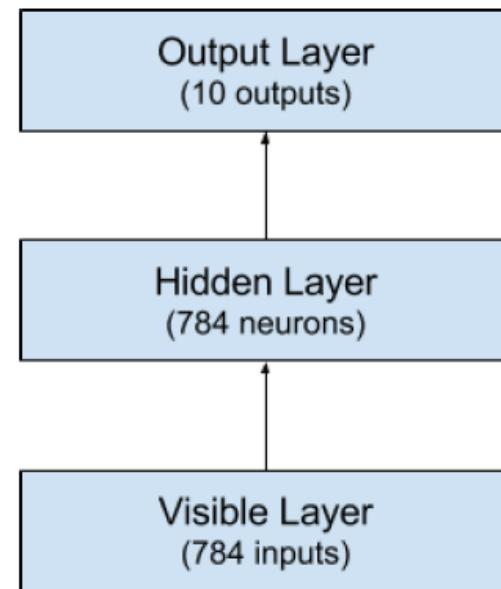
Ejemplo

- Redes Neuronales Convolucionales en Keras.
- Clasificación de MNIST database



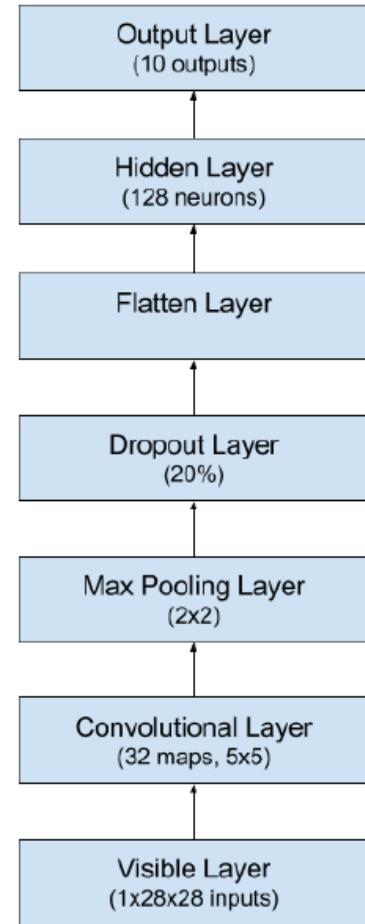
Ejemplo

- Realizar en Keras:
- Clasificación con una red neuronal clásica



Ejemplo

- Realizar en Keras:
- Clasificación con una red neuronal convolucional pequeña.



Ejemplo

- Realizar en Keras:
- Clasificación con una red neuronal convolucional larga

