MODELOS COMPUTACIONALES: CUARTO CURSO DEL GRADO DE ING. INFORMÁTICA EN COMPUTACION

REDES NEURONALES CONVOLUCIONALES

César Hervás-Martínez/ Pedro A. Gutierrez
Grupo de Investigación AYRNA

Departamento de Informática y Análisis Numérico
Universidad de Córdoba
Campus de Rabanales. Edificio Einstein.
Email: chervas@uco.es

2018-2019

REDES CONVOLUCIONALES

Se emplean en múltiples aplicaciones prácticas

Reconocimiento de imágenes, reconocimiento del habla, etiquetas de fotos de Google o Baidu

Han ganado varias competiciones

ImageNet, Kaggle Facial Expression, Kaggle Multimodal Learning, German Traffic Signs, Connectomics, Handwriting

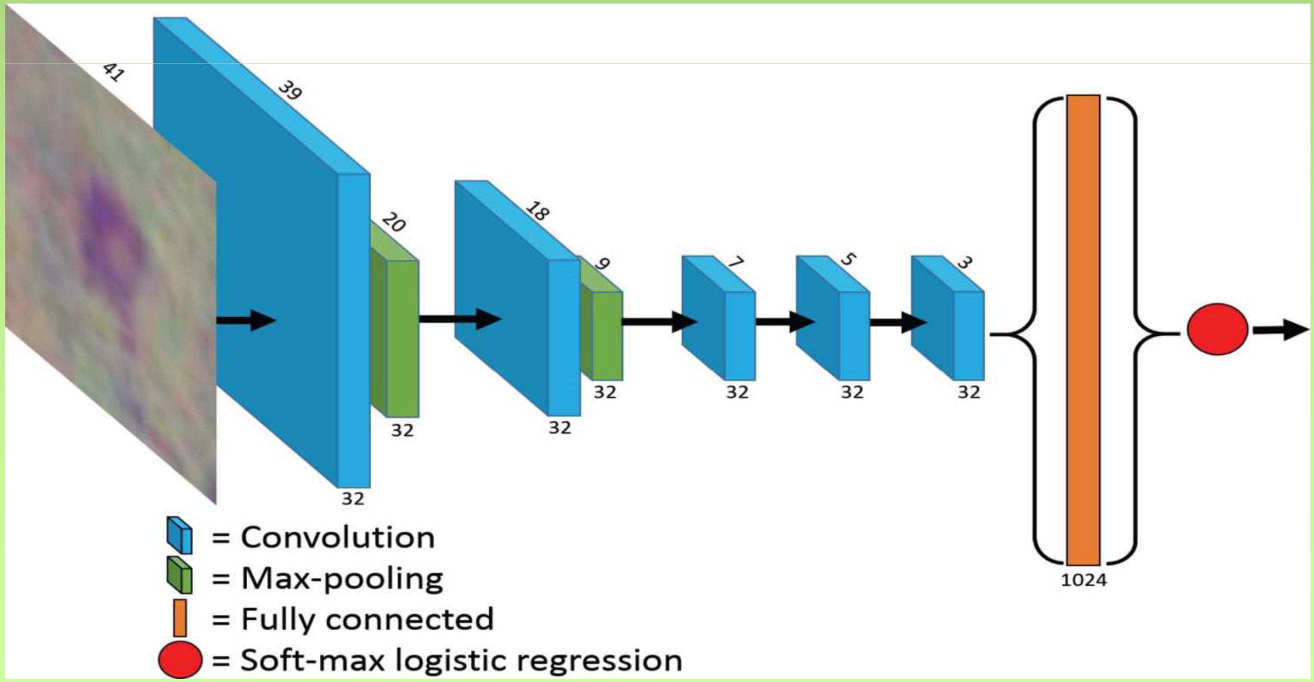
Se aplican a un array de datos donde los valores cercanos están correlados

Imagen, Sonido, Video, Imágenes volumétricas, 3D, Imágenes RGB-Depth

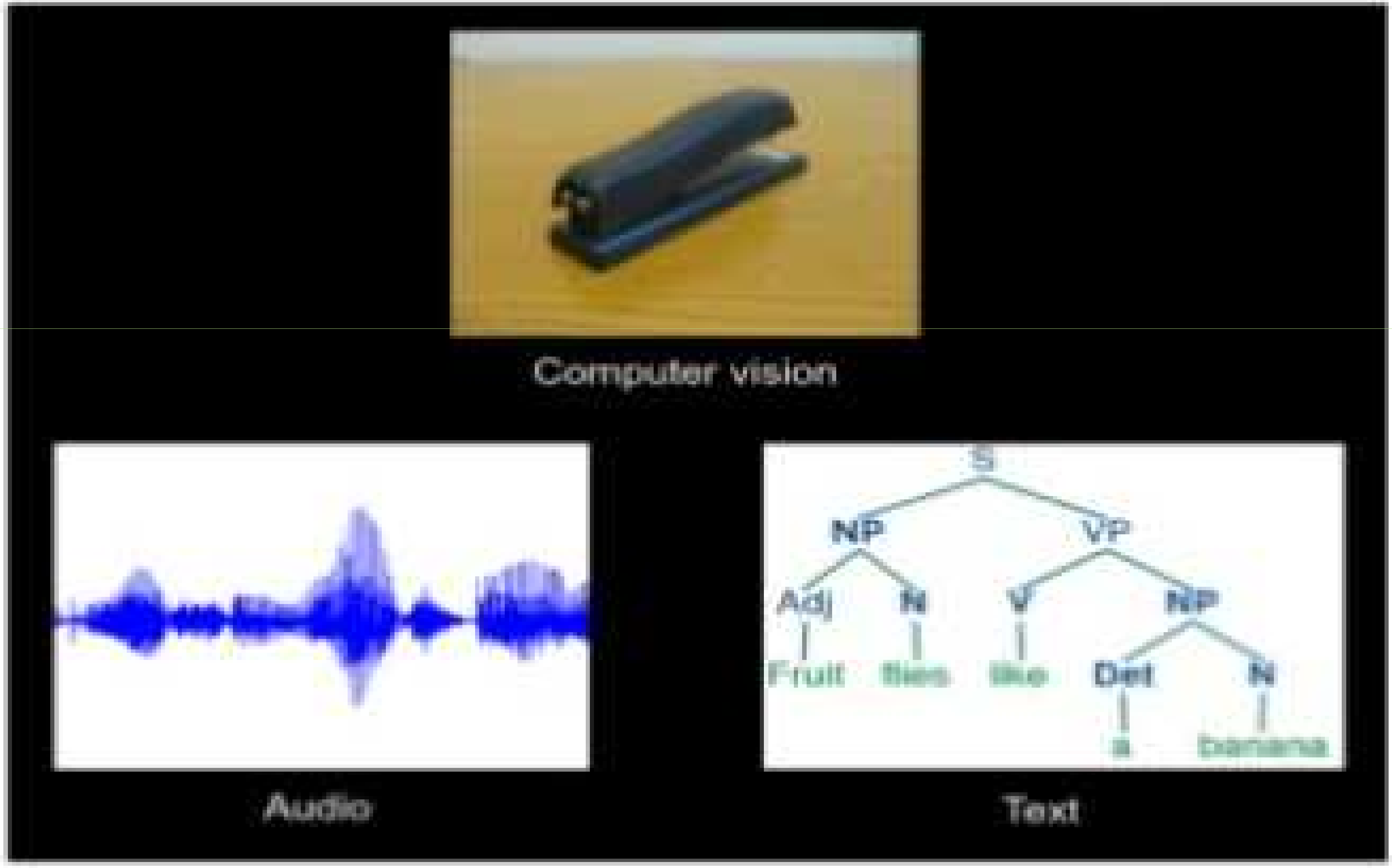
Es uno de los pocos modelos que puede ser entrenado de modo puramente supervisado

¿Que son las Redes Neuronales Convolucionales?

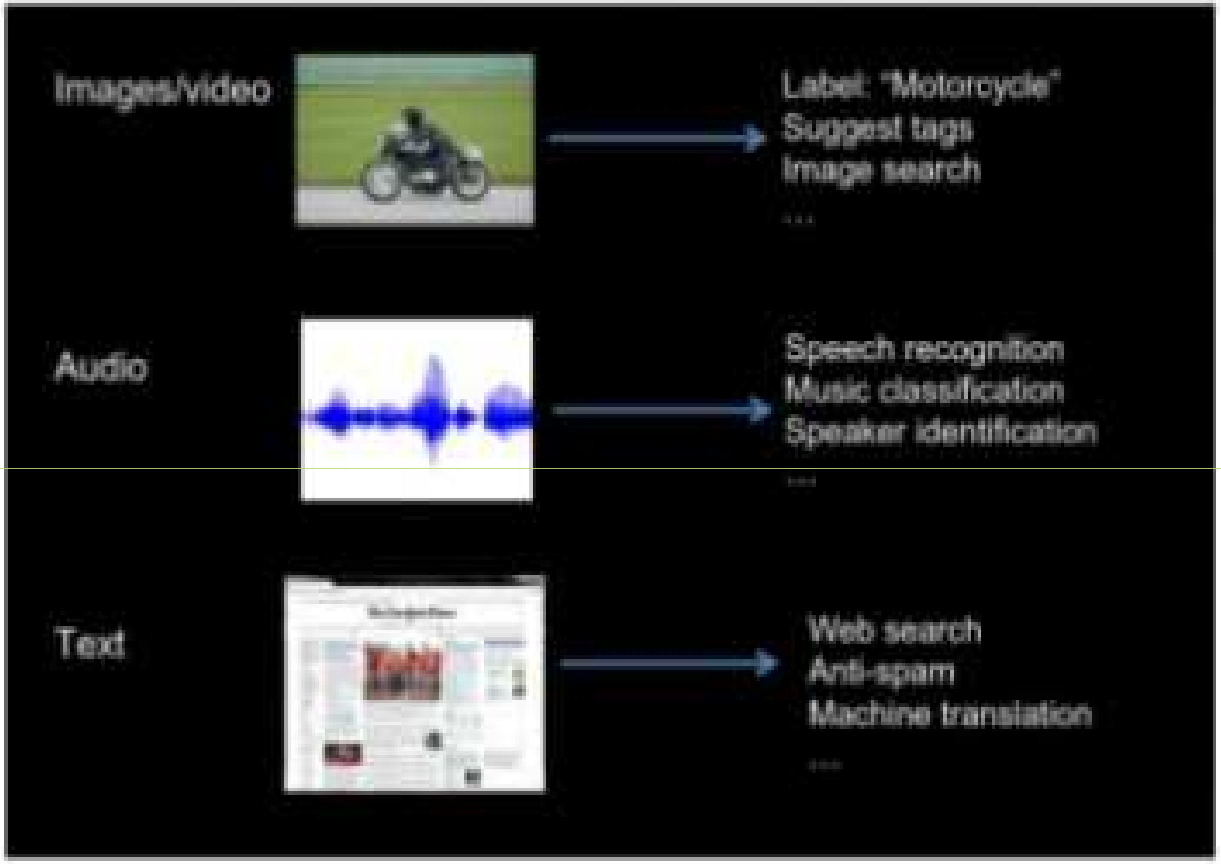
Son esencialmente redes neuronales que utilizan convolución en lugar de matrices generales multiplicativas en al menos una de sus capas ocultas



Se utilizan para la obtención de clasificadores potentes en visión, señales de audio, reconocimiento de textos, etc

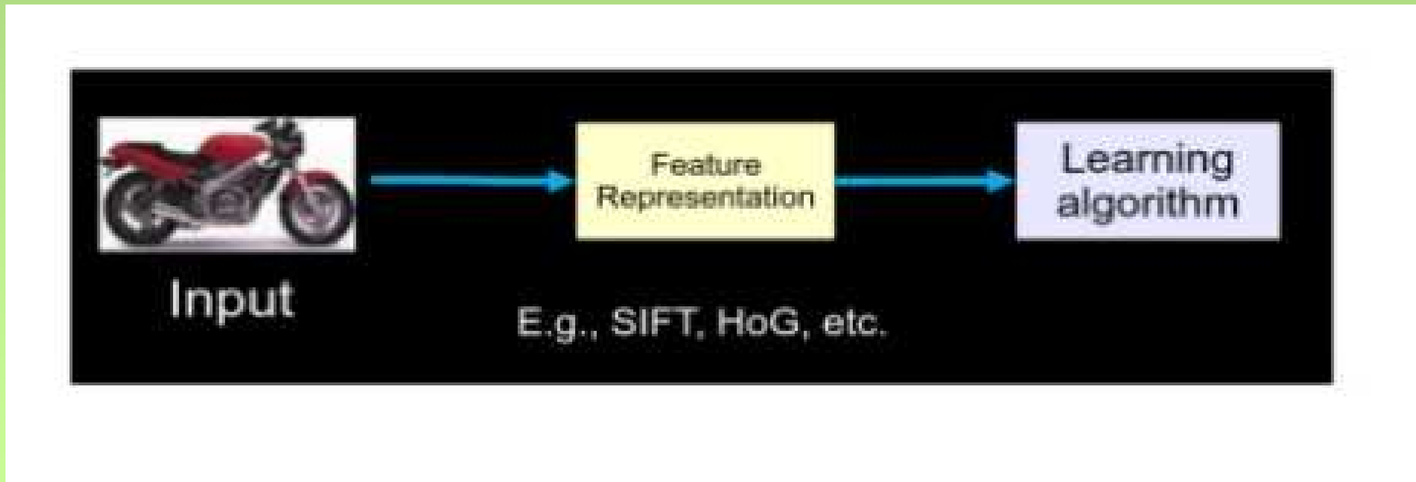


¿Que podemos hacer con estos datos para construir modelos de reconocimiento de patrones?



Etiquetar imágenes, sugerir etiquetas, buscar imágenes, reconocimiento del habla, clasificación de música, identificación de locutores, búsquedas en la web, detección de emails no deseados, traducción automática, etc

Representación de características



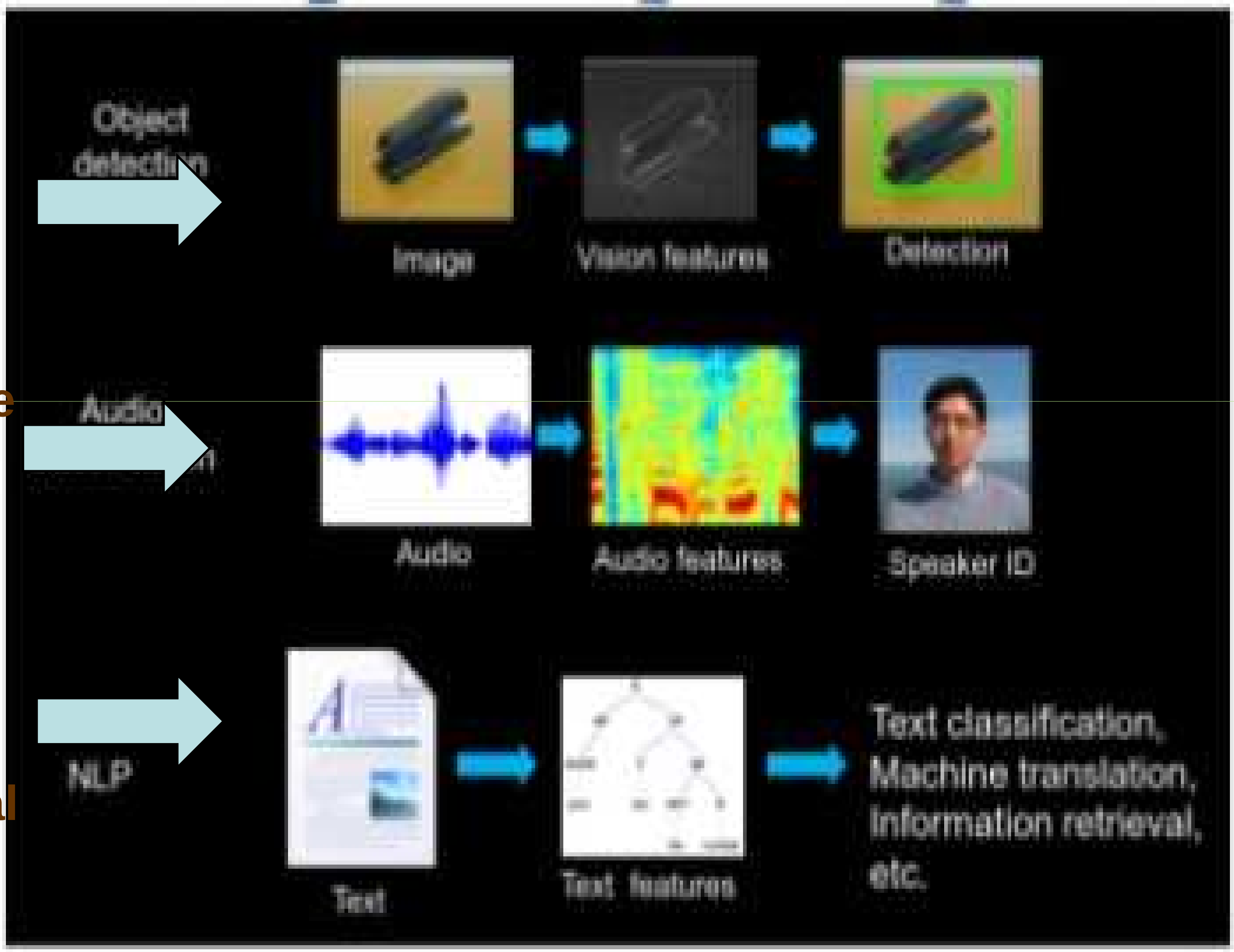
En 1999, David Lowe publicó su primer artículo sobre Scale-invariant feature transform (SIFT), donde describió un algoritmo de detección de características invariante tanto a rotaciones como al escalado de las imágenes

¿Como realiza la percepción la computadora?

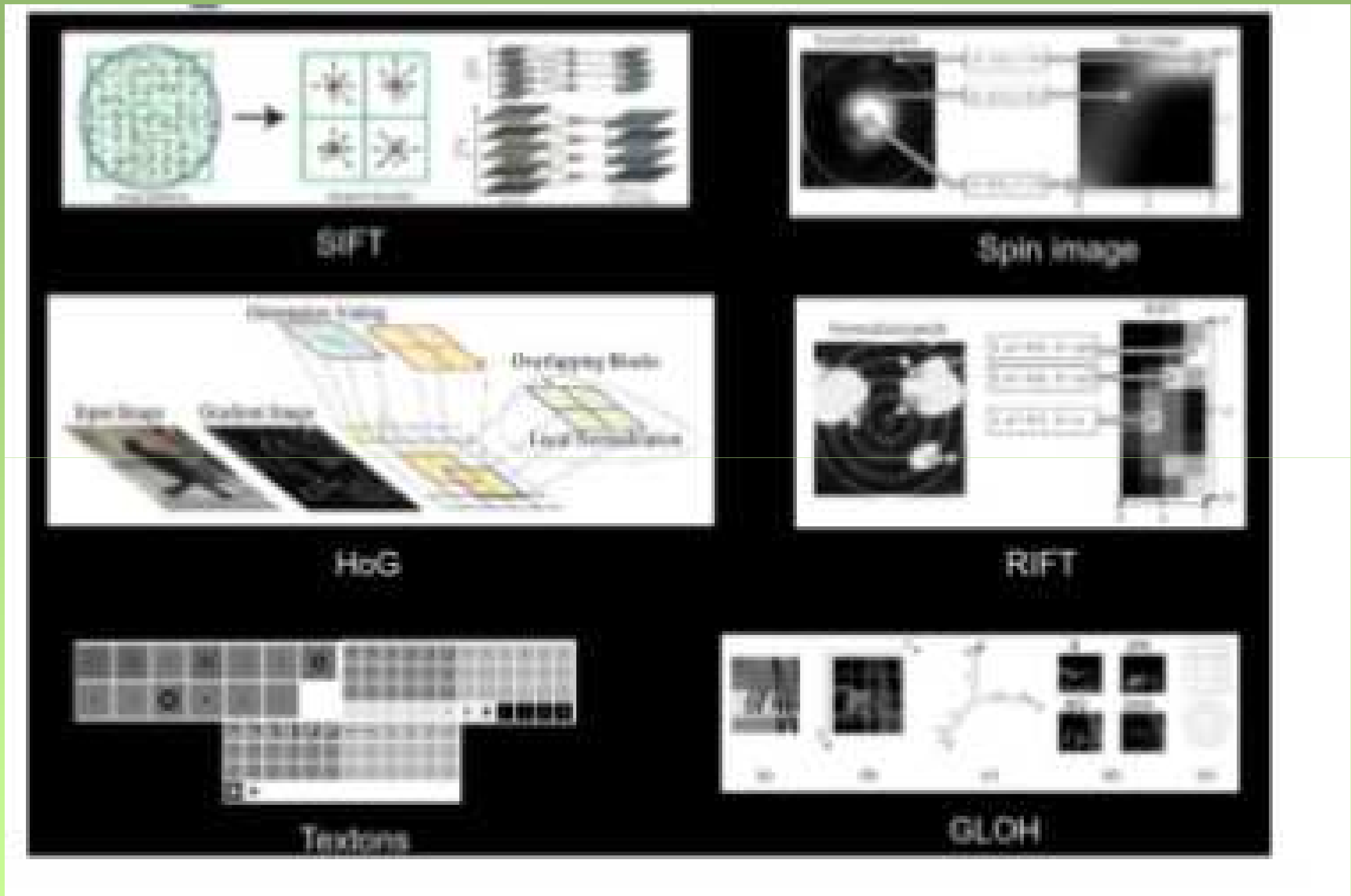
Detección de objetos

Clasificación de Audio

Procesamiento del lenguaje natural

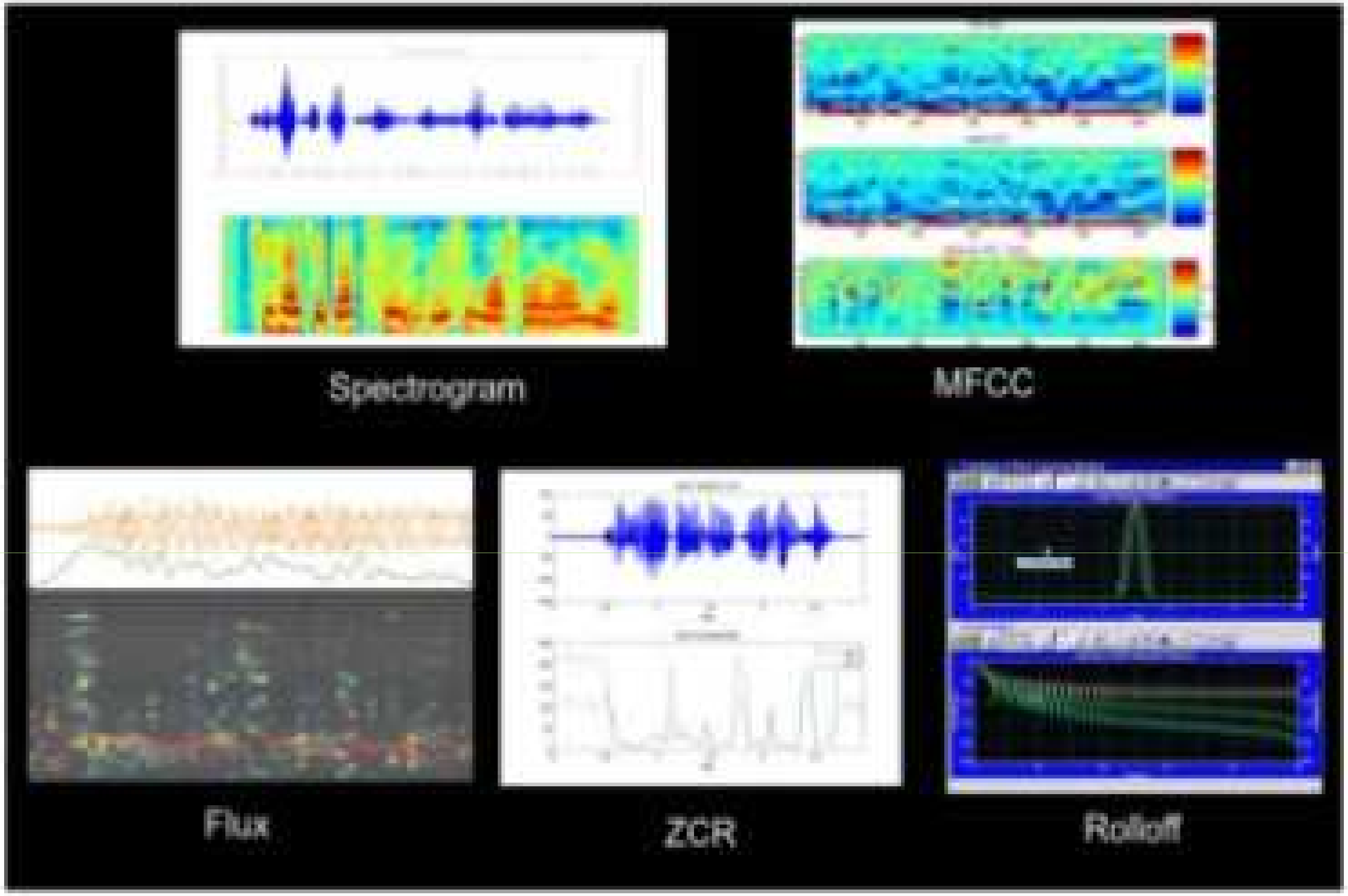


Características en visión



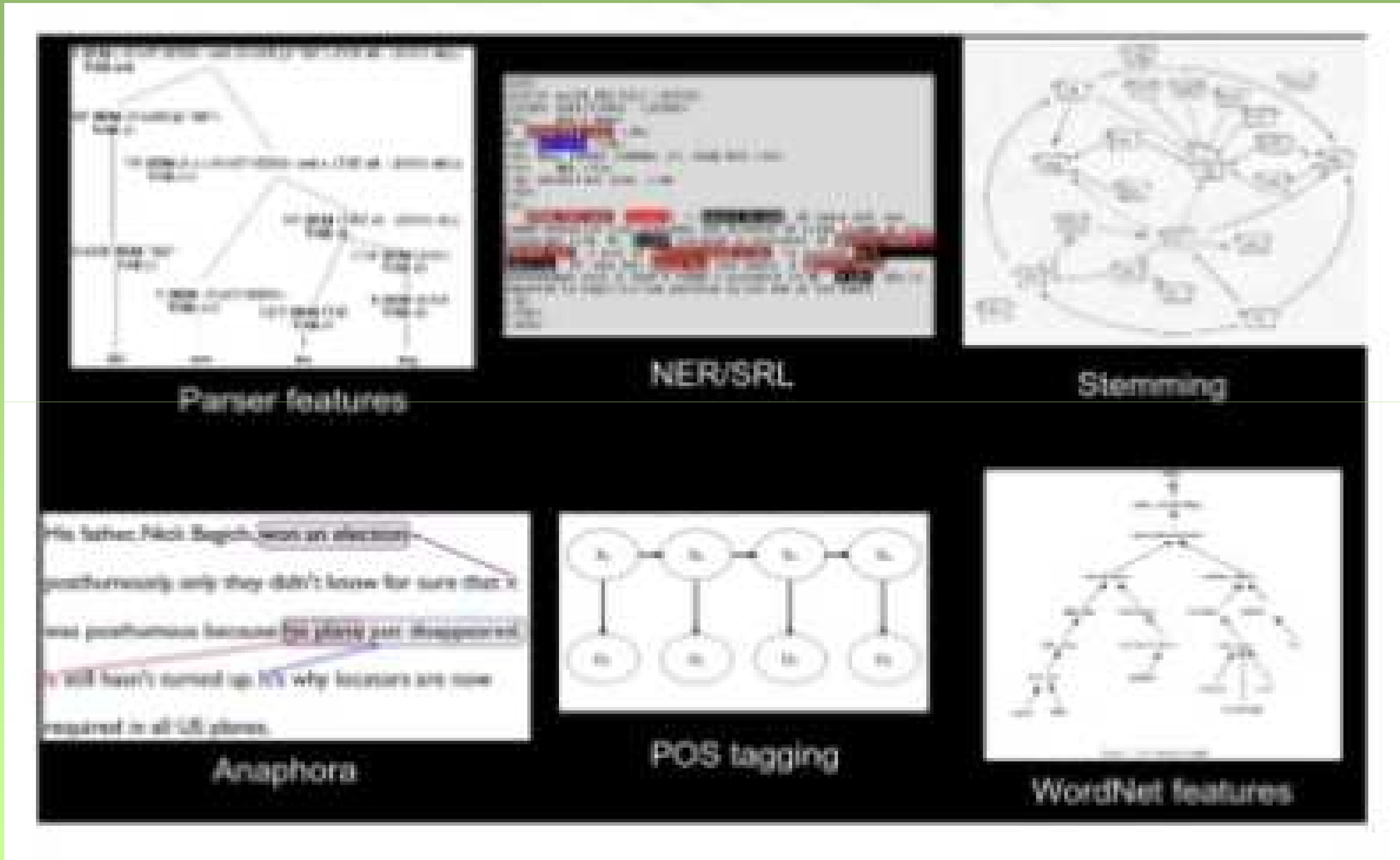
Obtener estas características es difícil,
y requiere mucho tiempo y conocimiento experto

Características de audio



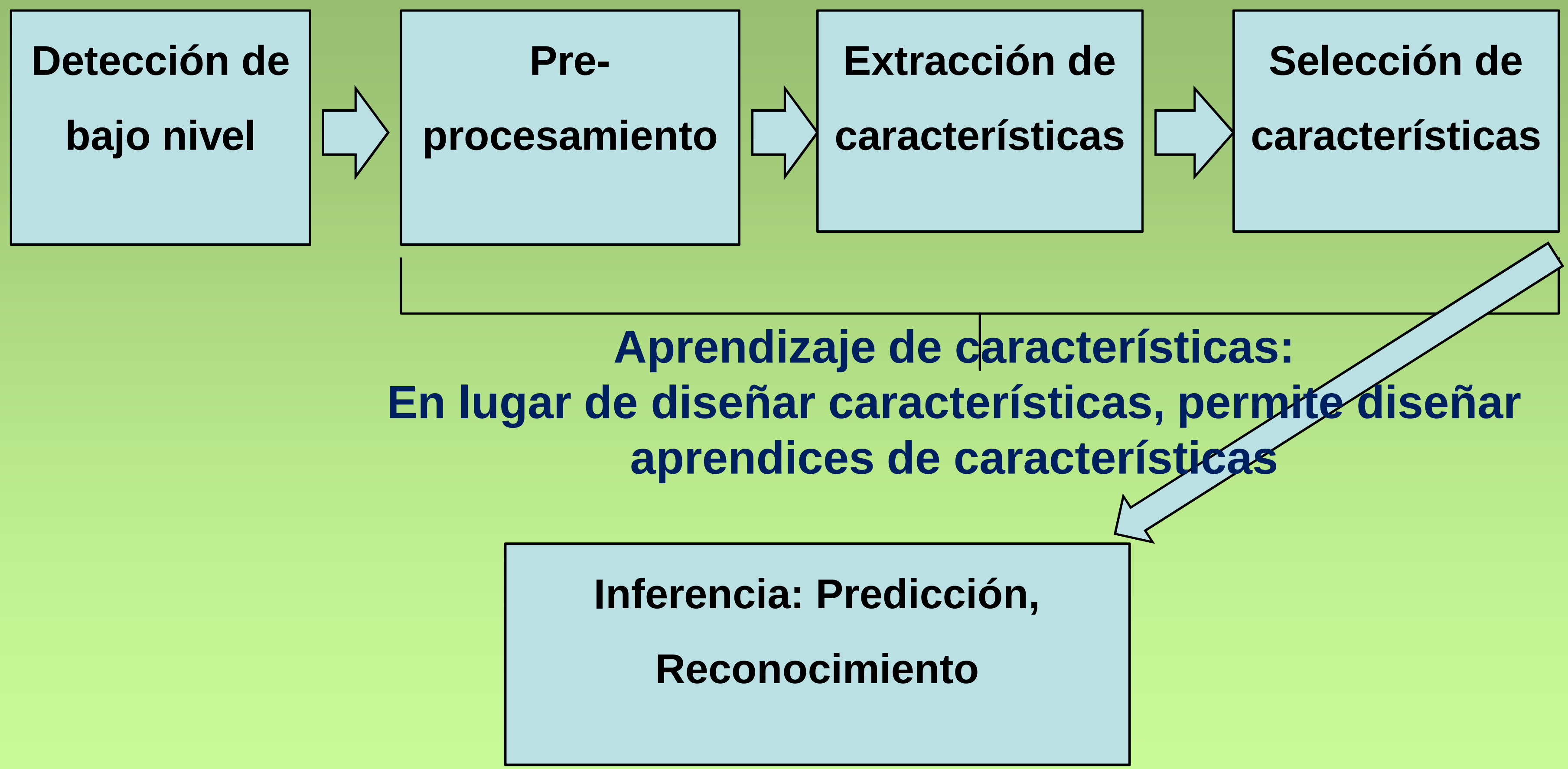
zero-crossing rate, ZCR es la tasa de cruce por el cero; o lo que es igual a la tasa de cambio de signo a lo largo de una señal, es decir, la velocidad a la que la señal cambia de positivo a negativo o viceversa.

Características del Procesamiento del Lenguaje Natural, NLP



Obtener estas características es difícil, y requiere mucho tiempo y conocimiento experto

Representación de las características



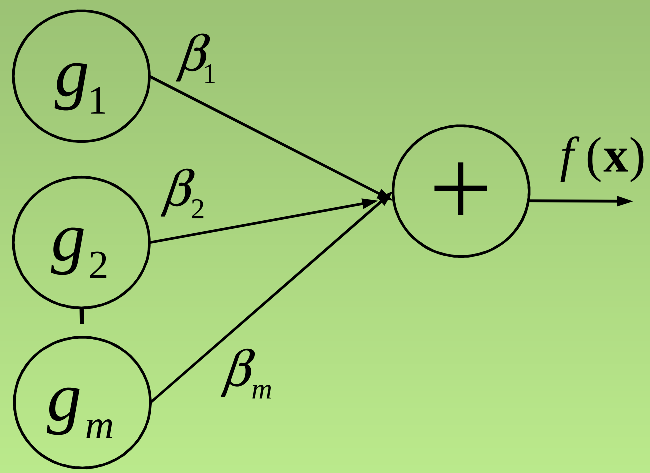
Aprendizaje de funciones no lineales

Dado un conjunto de funciones no lineales simples: $\{g_1,\ldots,g_k\}$

Podemos hacer dos propuestas:

a) Hacer una combinación lineal de dichas funciones de base, esto se llama un aprendizaje superficial.

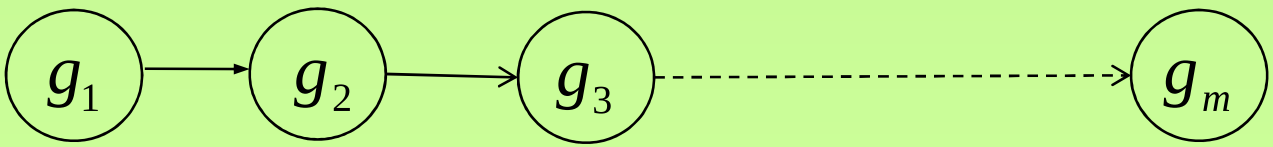
$$f(\mathbf{x}) = \sum_{j=1}^{k} \beta_k g_k(\mathbf{x})$$



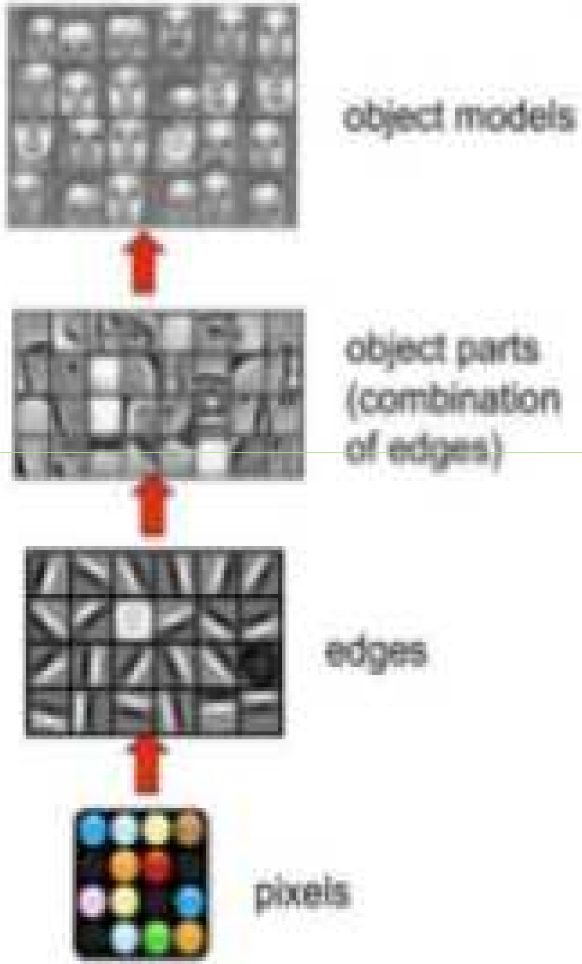
b) Hacer una composición de estas funciones.

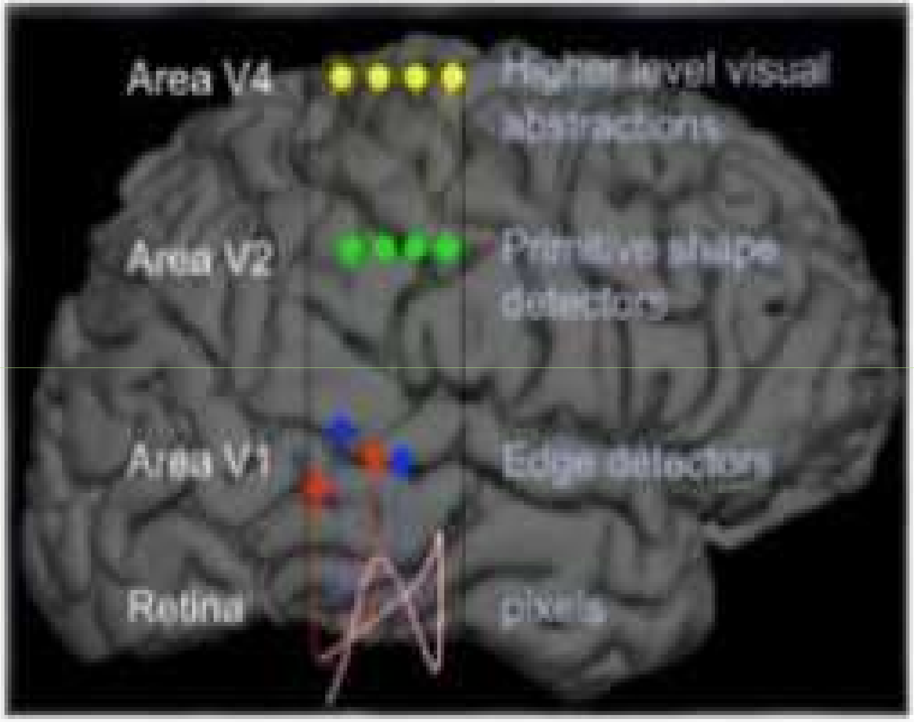
A esto le llamaremos aprendizaje profundo

$$f(\mathbf{x}) \approx g_1(g_2(\ldots g_m(\mathbf{x}))$$

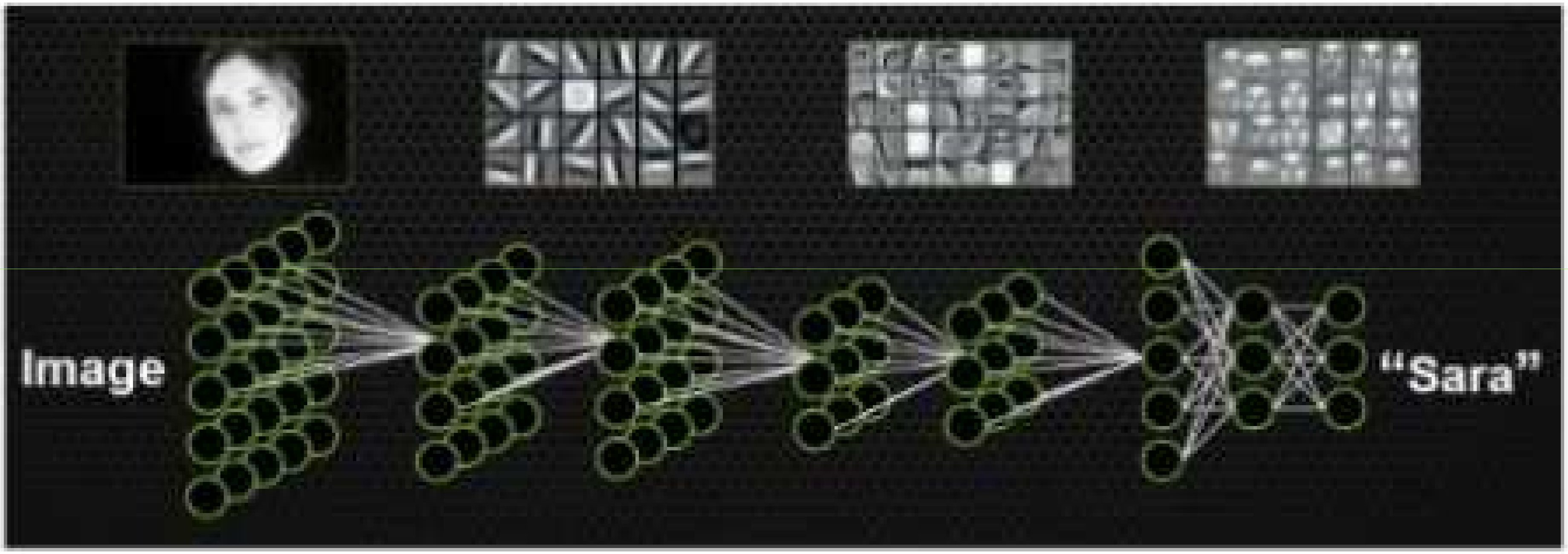


Inspiración biológica

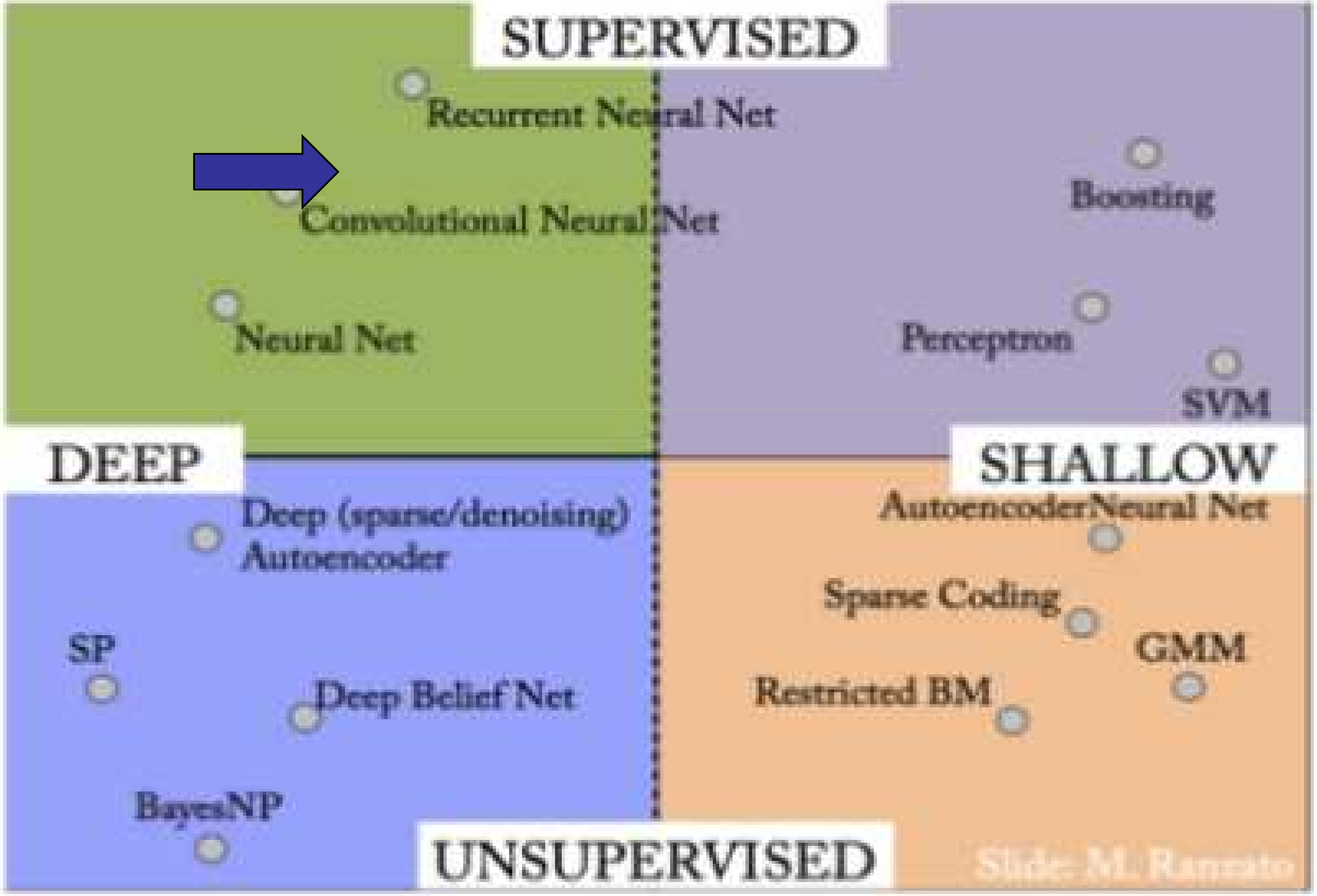


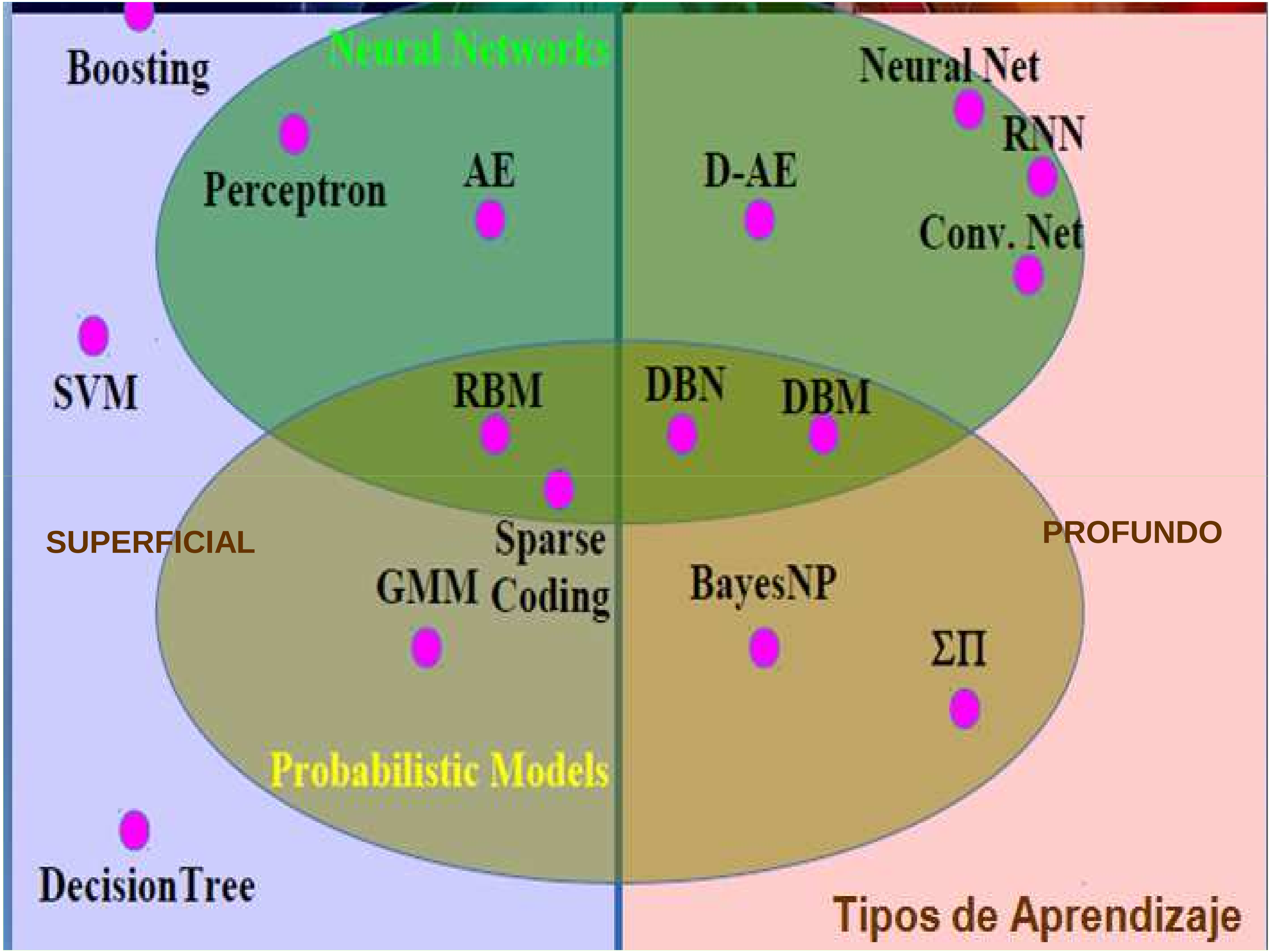


Aprendizaje de características mediante entrenamiento profundo

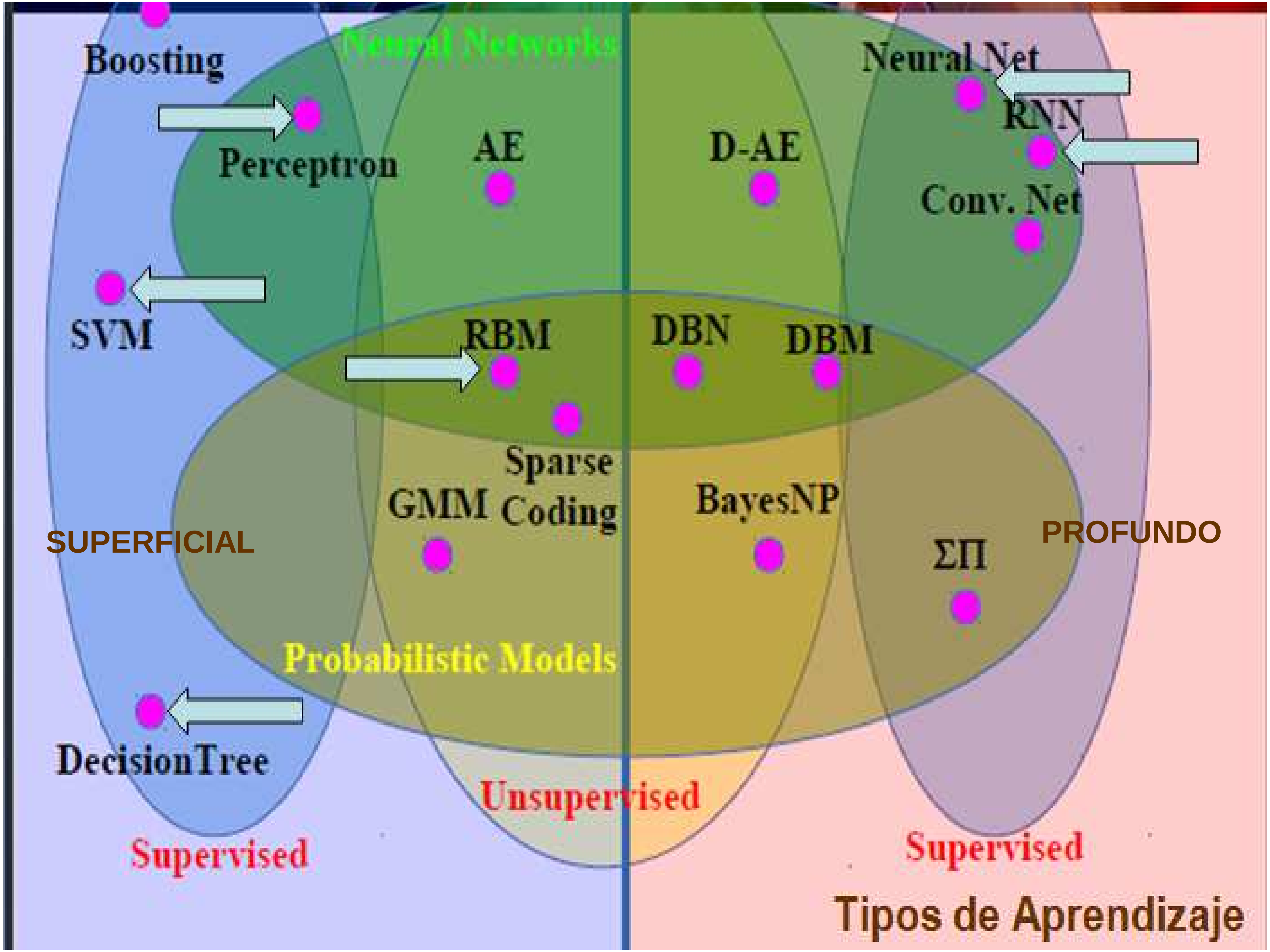


Tipos de Aprendizaje





Boosting
Neural Networks
Neural Net
RNN
Perceptron
AE
D-AE
Conv. Net
SVM
RBM
DBN
DBM
SUPERFICIAL
PROFUNDO
Sparse Coding
GMM
BayesNP
ΣΠ
Probabilistic Models
Decision Tree
Tipos de Aprendizaje


Boosting
Neural Networks
Neural Net
RNN
Perceptron
AE
D-AE
Conv. Net
SVM
RBM
DBN
DBM
Sparse
Coding
GMM
BayesNP
SUPERFICIAL
PROFUNDO
ΣΠ
Probabilistic Models
DecisionTree
Unsupervised
Supervised
Supervised
Tipos de Aprendizaje

Etapas de una CNN

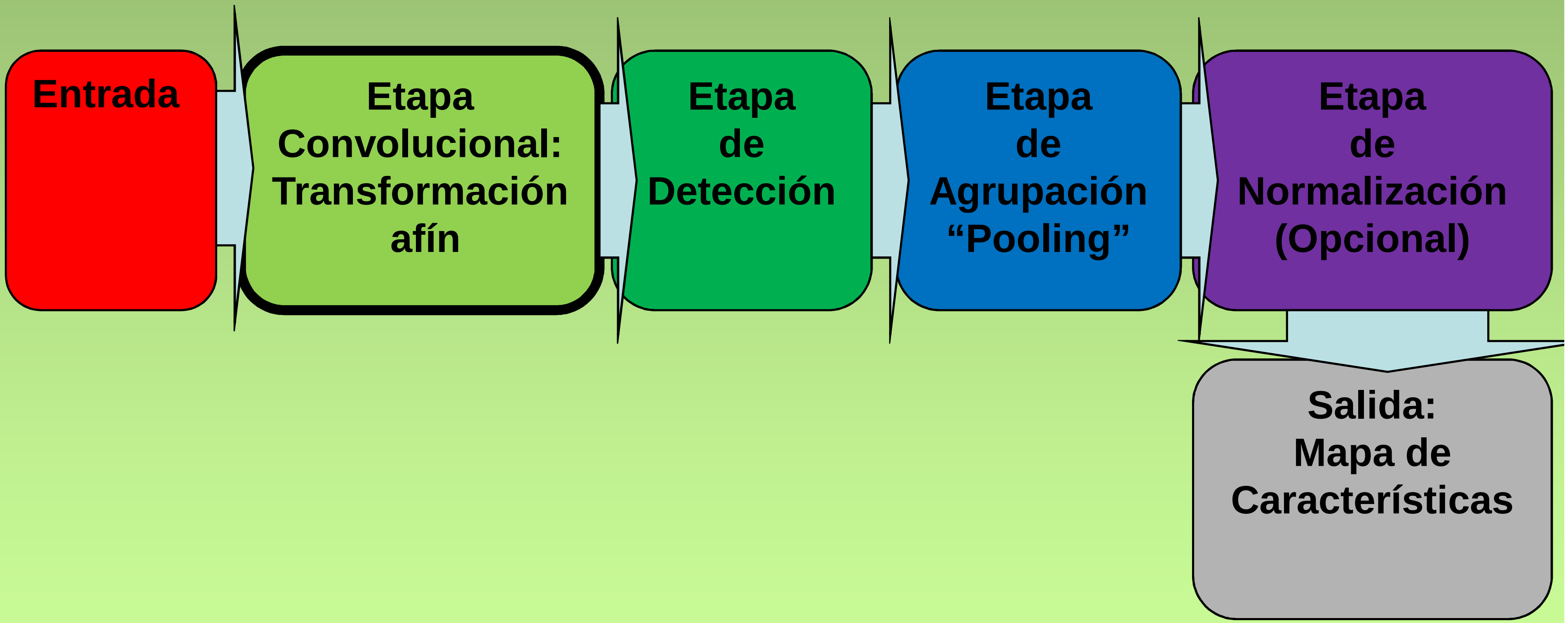
1) Extracción de Características

2) Campos Localmente Receptivos

3) Pesos Compartidos

4) Sub-muestreo espacial o temporal

Estructura de las capas de una CNN



Convolución

La convolución de las funciones f y g, escrita en la forma f*g se define como la integral del producto de las dos funciones después de que una se invierte y se desplaza, esto es

$$(f * g)(t) = \int_{-\infty}^{\infty} f(z)g(t-z)dz$$

Puede ser considerada como una operación de promediado ponderado en cada momento (para ello es necesario que g sea una función de densidad de probabilidad).

El intervalo de integración dependerá del dominio sobre el que estén definidas las funciones. En el caso de un rango de integración finito, *f* y *g* se consideran a menudo como extendidas, periódicamente en ambas direcciones, tal que el término *g*(*t*- z) no implique una violación del rango.

La convolución es conmutativa

Convolución discreta

Cuando se trata de hacer un procesamiento digital de señal no tiene sentido hablar de convoluciones aplicando estrictamente la definición ya que solo disponemos de valores en instantes discretos de tiempo. Es necesario, pues, una aproximación numérica.

Para realizar la convolución entre dos señales, se evaluará el área de la función: x(z)*h(t-z). Para ello, disponemos de muestreos de ambas señales en los instantes de tiempo nt , que llamaremos x[k] y h[n-k] (donde n y k son enteros). El área es, por tanto,

$$y[n] = \sum_{k=-\infty}^{\infty} t.x[k].h[n-k] = t.\left[\sum x[k].h[n-k] \right]$$

La convolución discreta se determina para un intervalo de muestreo t= 1

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k].h[n-k]$$

Correlación cruzada

En procesamiento de señales, la correlación cruzada (o a veces denominada "covarianza cruzada") es una medida de la similitud entre dos señales, frecuentemente usada para encontrar características relevantes en una señal desconocida por medio de la comparación con otra que sí se conoce. Es función del tiempo relativo entre las señales, a veces también se la llama *producto escalar desplazado*, y tiene aplicaciones en el reconocimiento de patrones y en criptoanálisis. Para funciones continuas se define como

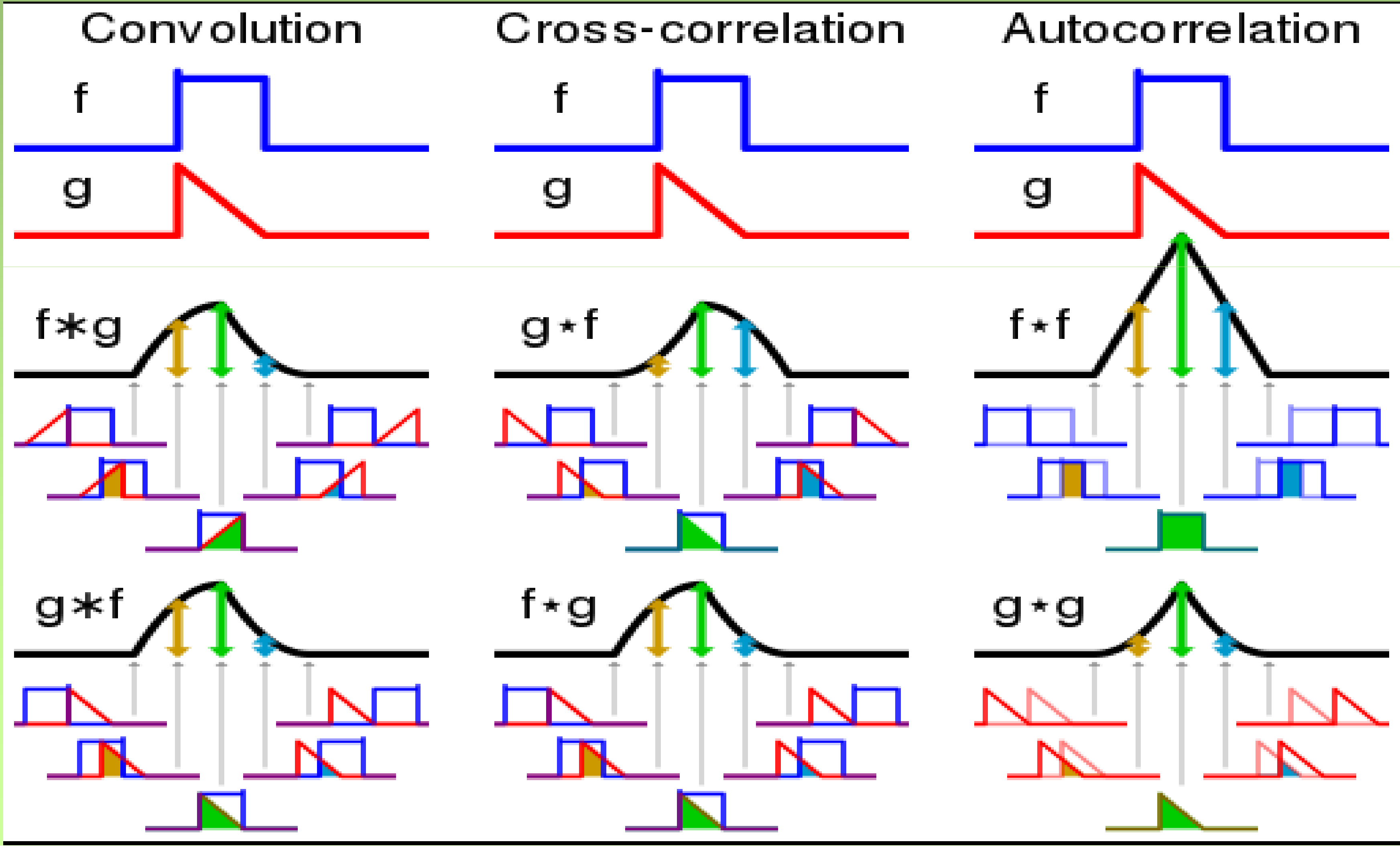
$$(f * g)(\mathrm{x}) \overset{def}{=} \int_{-\infty}^{\infty} f^*(t) g(x+t) dt$$

Donde f* es la conjugada compleja de f y x es el retardo

Para funciones discretas se define como

$$(f * g)[n] \overset{def}{=} \sum_{m=-\infty}^{\infty} f^*[m] g[m+n]$$

Convolución y correlación: Ejemplos. Hay que tener en cuenta que la simetría de la función f hace que f*g y g*f sean operaciones idénticas con resultados idénticos



Convolución y correlación cruzada en imágenes

Para una imagen H en 2-D y para un *kernel* F en 2-D,

El operador de Convolución G= H*F, es de la forma

$$G_{ij} = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H(u,v) F(i-u, j-v)$$

Y el operador de Correlación G= H⊗F

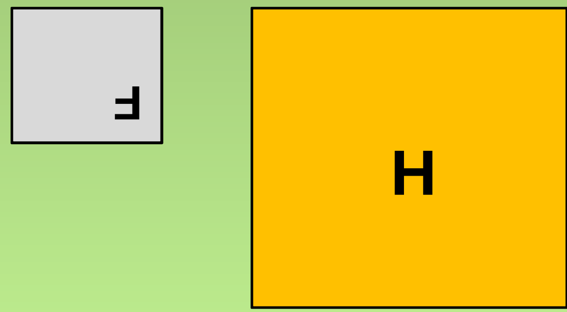
$$G_{ij} = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H(u,v) F(i+u, j+v)$$

¿Como se diferencian la convolución y la correlación cruzada?

La convolución es equivalente a invertir el filtro en ambas dimensiones (de abajo hacia arriba, de derecha a izquierda) y aplicar correlación cruzada.

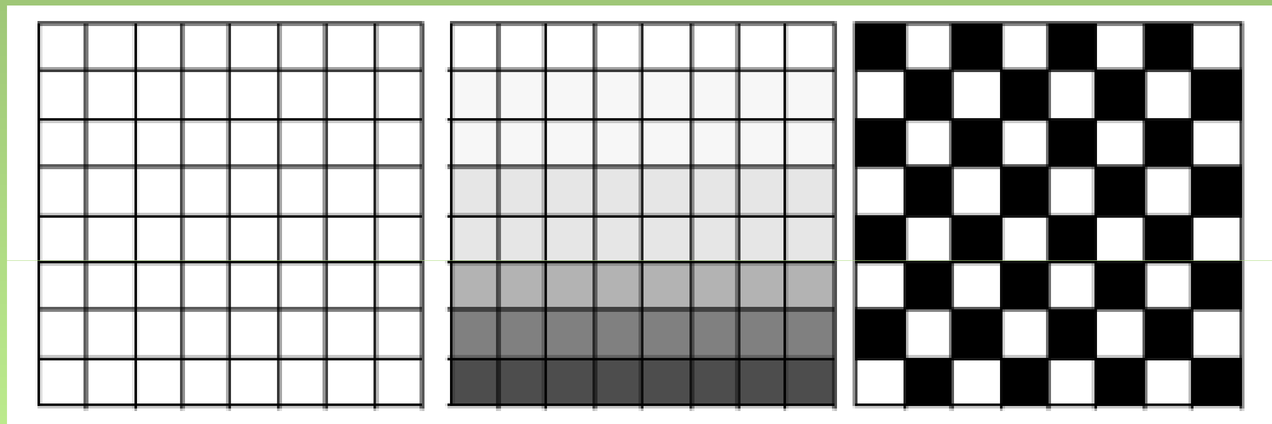
Para *kernels* simétricos, ambos operadores dan como resultado la misma salida.

Muchas bibliotecas de aprendizaje automático implementan la correlación cruzada, pero lo llaman convolución.



Frecuencias espaciales

El filtrado por convolución se utiliza para modificar la frecuencia espacial de las características de una imagen



Frecuencia espacial nula

Frecuencia espacial baja

Frecuencia espacial alta

¿Que es una convolución?

Una operación de convolución es el efecto de un filtro de propósito general para imágenes.

Consta de una matriz aplicada a una imagen y una operación matemática que consta de números enteros.

Trabaja determinando el valor de un pixel central añadiendo los valores ponderados de todos sus vecinos juntos

La salida es una nueva imagen filtrada y modificada

El proceso de convolución de una imagen

Una convolución se realiza multiplicando un pixel y sus pixeles vecinos con un valor de color o nivel de gris por una matriz

***Kernel*: Un *kernel* es, por lo general, una matriz pequeña de números que se utiliza en la convolución de imágenes.**

Diferentes tamaños de *kernel* contienen diferentes patrones de números, lo que produce diferentes resultados bajo la convolución.

El tamaño de un *kernel* es arbitrario pero a menudo es de 3x3

Ejemplo de *kernel*

0	1	0
1	1	1
0	1	0

Formula de la convolución

$$V = \left| \frac{\sum_{i=1}^{q} \sum_{j=1}^{q} f_{ij} d_{ij}}{F} \right|$$

Donde

- f_{ij} es el coeficiente de un *kernel* de convolución en la posición ij (en el *kernel*)
- d_{ij} el valor del dato del pixel que corresponde a f_{ij}
- q es la dimensión del *kernel.* suponiendo un *kernel* cuadrado (si q=3, entonces el *kernel* es de 3x3
- F es la suma de los coeficientes del *kernel* o 1 si la suma de los coeficientes es 0
- V es el valor de salida del pixel

En el caso en el que V sea menor que 0, V se sitúa a 0

Ejemplo

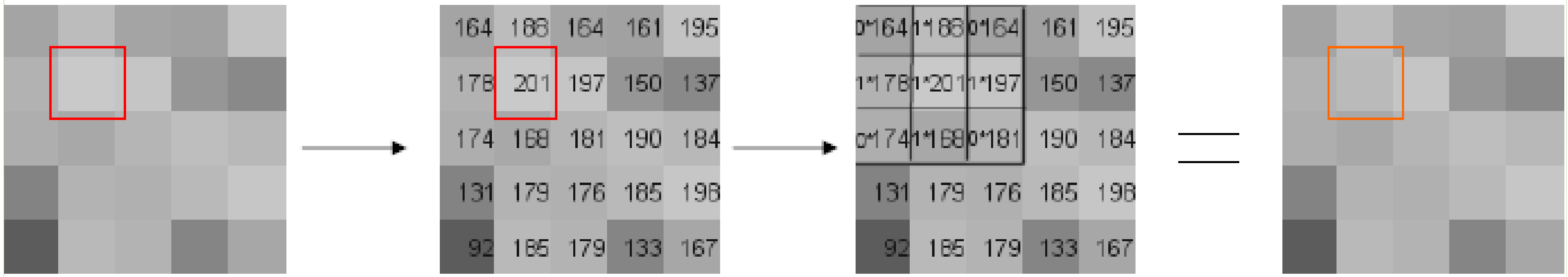


Imagen original

Imagen con valores del color de los pixeles situados sobre ella

Imagen con un *kernel* de 3x3 situado sobre ella

Imagen de salida

Suma 188+178+201+197+168=932

164	188	164
178	201	197
174	168	181

×

0	1	0
1	1	1
0	1	0

Valores de color

Kernel

Dividiendo por la suma del kernel 932/5= nuevo valor de color del pixel

Mas Ejemplos

Dato de entrada

2	9	6	6	6
2	8	6	6	6
2	2	8	6	6
2	2	2	8	6
2	2	2	2	8

Kernel

-1	-1	-1
-1	16	-1
-1	-1	-1

Parte entera de [(-1x8)+(-1x6) +(-1x6)+(-1x2)+(16x8)+(-1x6)+(-1x2)+(-1x2)+(-1x8)/(-1-1-1-1+16-1-1-1-1)]= Parte entera de (128-40)/(16-8)= Parte entera de 11= 11. Es el nuevo valor del pixel

Que hacemos con los pixeles del borde

a) Envolver la imagen

b) Ignorar los pixeles del borde y sólo calcular aquellos pixeles con todos los vecinos completos.

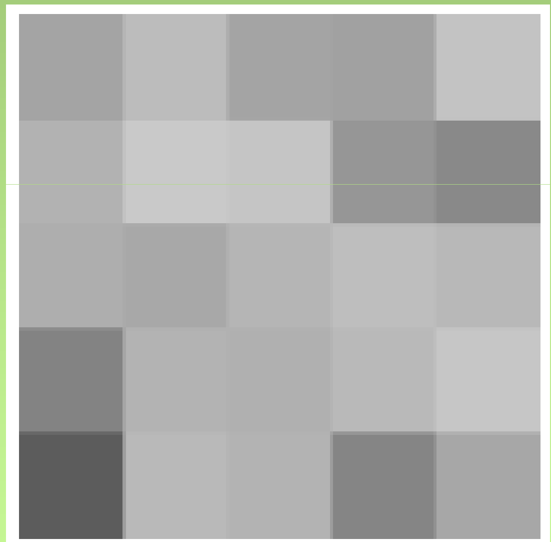
c) Duplicar los pixeles del borde de forma tal que el pixel en (2,n) (donde n seria no positivo) tendrá un valor de (2,1)

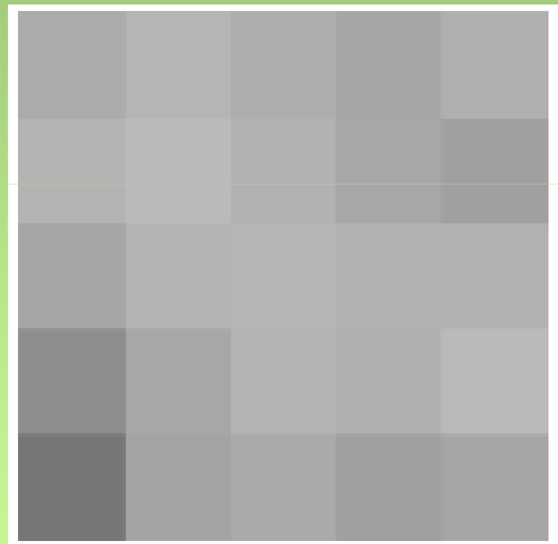
Donde -1 es no existir dato

	-1	-1	-1	-1	-1	
-1	164	188	164	161	195	-1
-1	178	201	197	150	137	-1
-1	174	168	181	190	184	-1
-1	131	179	176	185	198	-1
-1	92	185	179	133	167	-1
	-1	-1	-1	-1	-1	

	1	1	1	1	1	
1	164	188	164	161	195	1
1	178	201	197	150	137	1
1	174	168	181	190	184	1
1	131	179	176	185	198	1
1	92	185	179	133	167	1
	1	1	1	1	1	

Imagen original **Imagen modificada alisada**





Otros ejemplos de kernel

1	1	1
1	1	1
1	1	1

Kernel de alisamiento no ponderado de 3x3

0	1	0
1	4	1
0	1	0

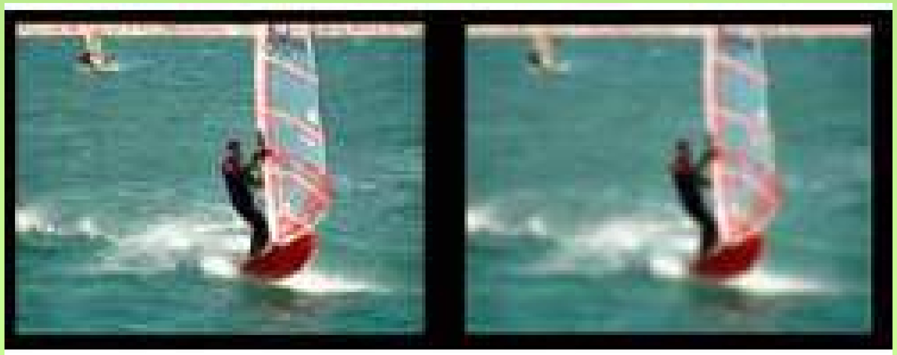
Kernel de alisamiento ponderado de 3x3 con difuminación Gaussiana

0	-1	0
-1	5	-1
0	-1	0

Kernel para agudizar la imagen

-1	-1	-1
-1	9	-1
-1	-1	-1

Imagen agudizada intensificada



Difuminación Gaussiana

Imagen agudizada

Ejemplo de *kernel* de alisamiento

Cuando se utiliza un *kernel* de suavizado un área más grande del núcleo aumenta el área de suavizado

0	1	2	1	0
1	4	8	4	1
2	8	16	8	2
1	4	8	4	1
0	1	2	1	0

***Kernel* 5x5 de suavizado**

Puntos principales

a) Empezar con una imagen

b) Elegir el *kernel* que afecta a la imagen de salida

c) Basar la elección del *kernel* en los resultados deseados para la imagen (alisar, difuminar, mejorar, afilar)

Convolución 2-D (sin invertir el *kernel*)

Entrada

a	b	c	d
e	f	g	h
i	j	k	l

Kernel

w	x
y	z

Salida

aw+bx +ey+fz	bw+cx +fy+gz	cw+dx +gy+hz
ew+fx +iy+jz	fw+gx +jy+kz	gw+hx +ky+lz

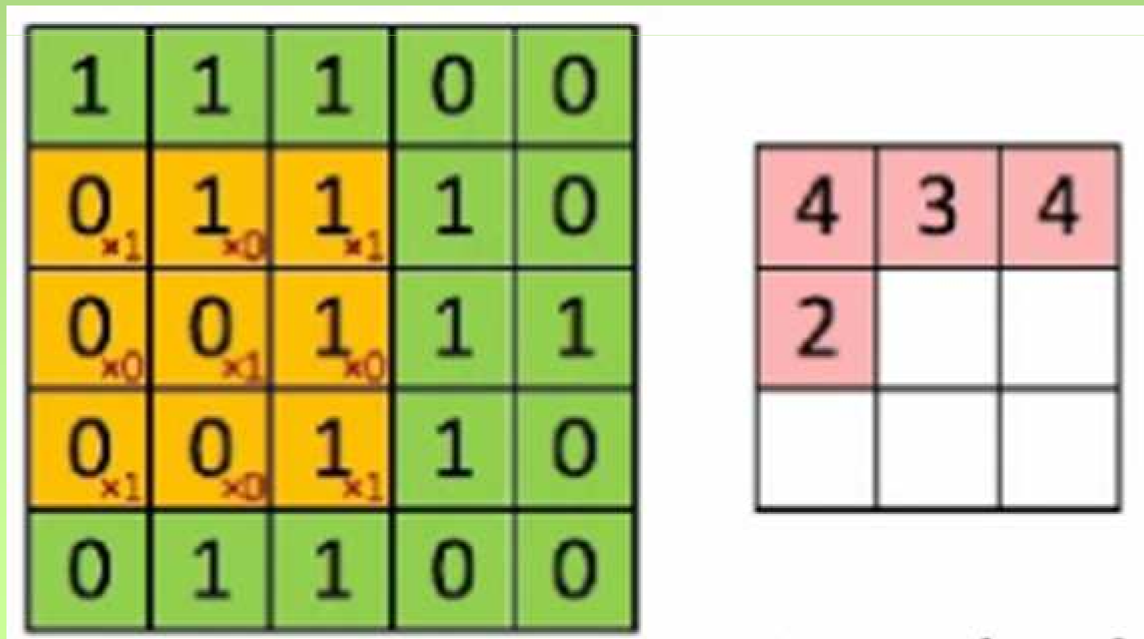
Ejemplo de una convolución 2-D válida (sin invertir el kernel) donde una matriz de 3x4 convoluciona con un kernel de 2x2 dando como salida una matriz de dimensión (3-1)x(4-1)

Convolución 2-D

kernel

1	0	1
0	1	0
1	0	1

Imagen **Función Convolvente**



Variantes

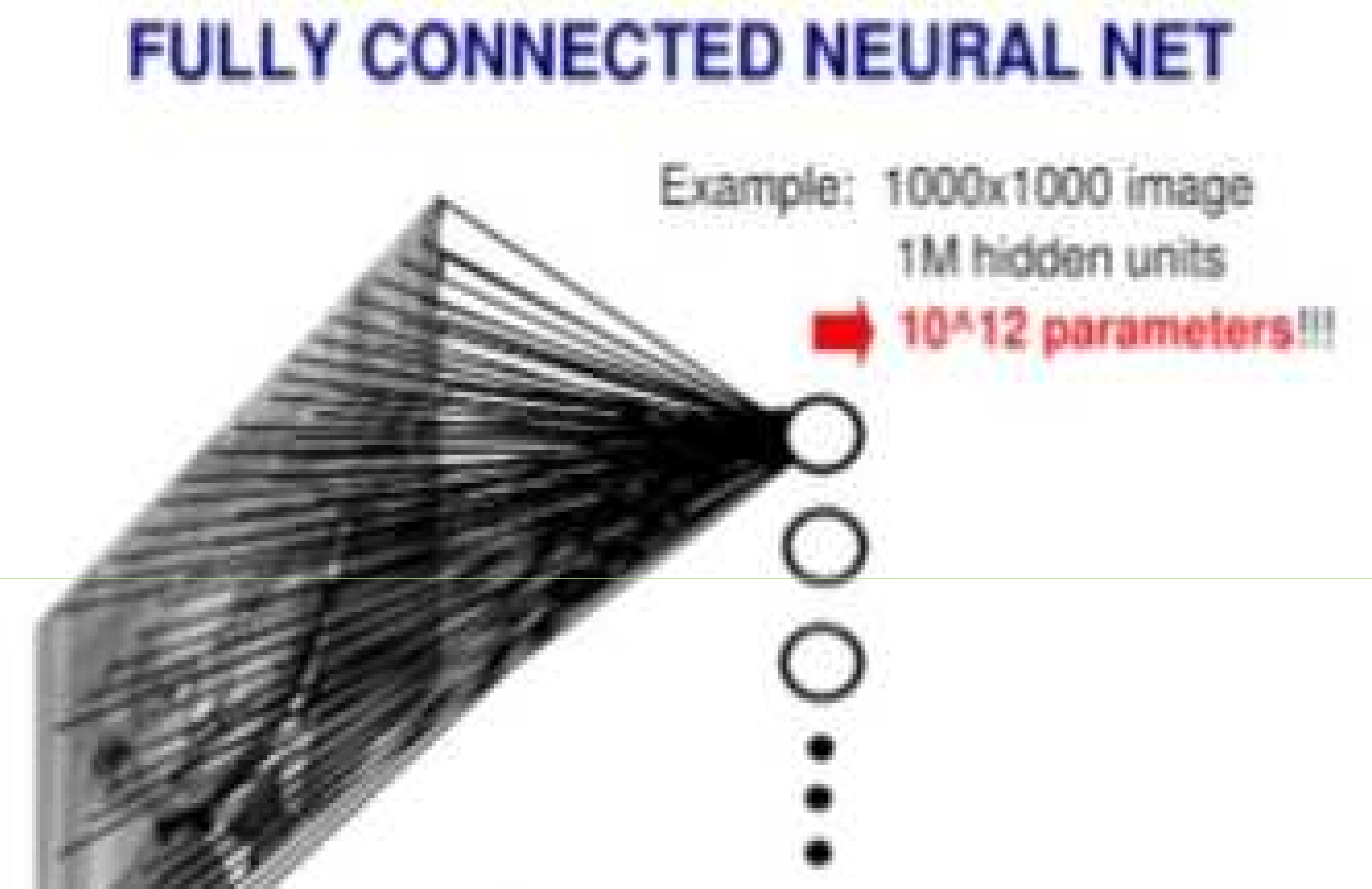
Completa: Añade suficientes ceros de relleno a la imagen para que cada píxel sea visitado k veces en cada dirección, con tamaño de salida: (m + k-1) x (m + k-1)

Válida. Sin rellenar con ceros, el *kernel* es restringido a girar horizontalmente sólo dentro de la imagen, con tamaño de salida : (m-k+1) x (m-k+1)

Idéntica: Añade ceros de relleno a la imagen para que la salida tenga el mismo tamaño de la imagen, por ejemplo, m x m.

Paso o *Stride*: Muestreo descendente de la salida de convolución muestreando solo cada s píxeles en cada dirección. Por ejemplo, la salida de convolución "válida" con paso s da como resultado una salida de tamaño (m-k+s)/s x (m-k+s)/s.

Porque convolución



La correlación espacial es local
Mejor poner recursos en otro sitio

Redes completamente o localmente conectadas de alta dimensión

Ejemplo: Imagen de 200 x 200

Completamente conectada ,

400.000 unidades ocultas=

$$4\times10^{4}\times4\times10^{5}=16\times10^{9}$$

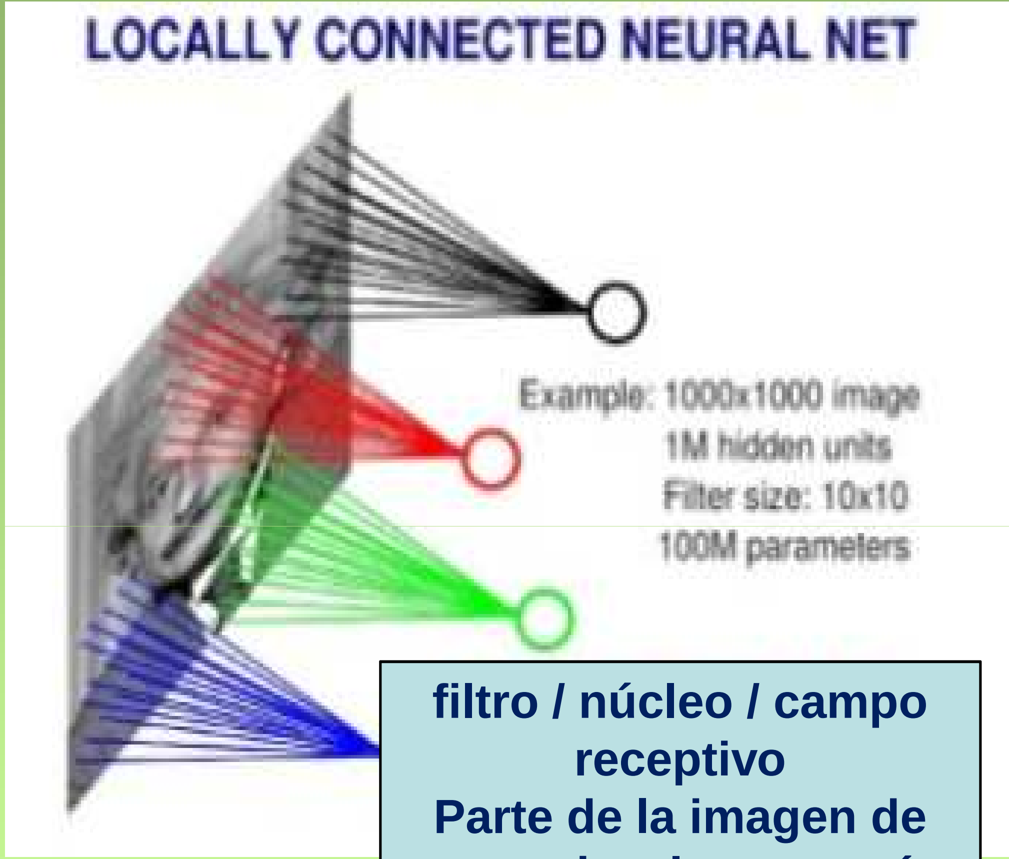
= 16 billones de parámetros

Localmente conectada, 10 x 10 campos o localizaciones de 400.000 unidades ocultas =

$$10^{2}\times4\times10^{5}=4\times10^{7}$$

= 40 millones de parámetros

Las conexiones locales capturan dependencias locales



filtro / núcleo / campo receptivo
Parte de la imagen de entrada a la que está conectada la unidad oculta

Múltiples convoluciones con diferentes *kernels*

Detecta múltiples patrones en cada localización

La colección de unidades mirando a la misma subimagen es similar a un vector de características para cada subimagen-

El resultado es un *array* en 3D, donde cada rebanada es un mapa de características

Campos Localmente Receptivos/ Conectividad dispersa

La Convolución explota la propiedad de correlaciones locales espaciales en la imagen reforzando los patrones de conectividad local entre neuronas de capas adyacentes.

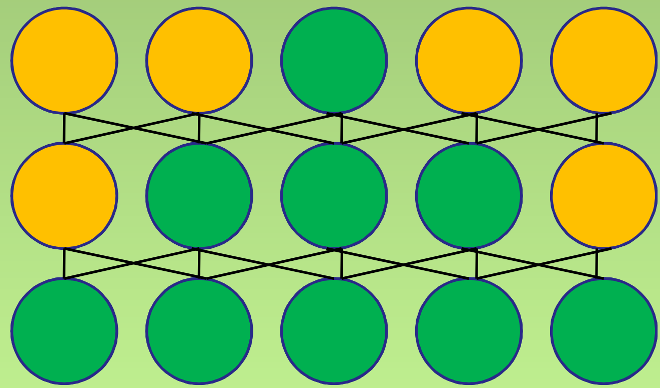
Reduce drásticamente el número de parámetros libres en comparación con las redes completamente conectadas reduciendo el sobre-entrenamiento y lo más importante, reduciendo la complejidad computacional de la red

Conectividad global indirecta

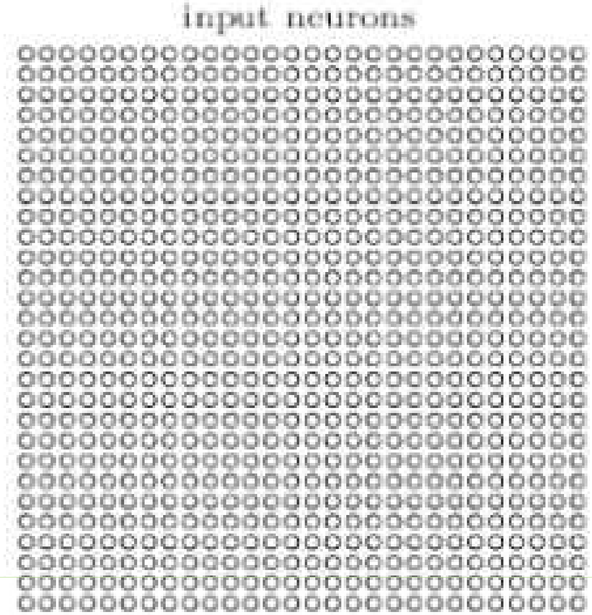
Los campos receptivos de unidades en capas más profundas son más grandes que las capas superficiales.

Aunque las conexiones directas son muy dispersas, las capas profundas indirectamente conectan la mayoría de las entradas de la imagen.

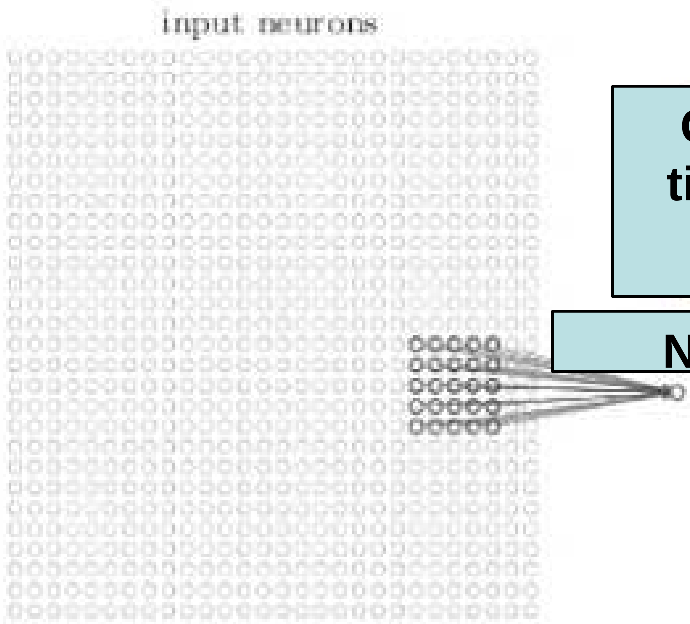
La efectividad aumenta con la convolución con saltos, *stride* o con agrupamiento, *pooling*.



Ejemplo



Las neuronas de entrada representan una imagen de 28x28 de la base de datos MNIST



Cada neurona de la capa oculta tiene un campo receptivo de una región de 5x5 pixeles

Neurona oculta

Ejemplo

Así sucesivamente se va construyendo la primera capa oculta

(28-4)x(28-4)= 24x24 neuronas en la capa oculta son la convolución válida

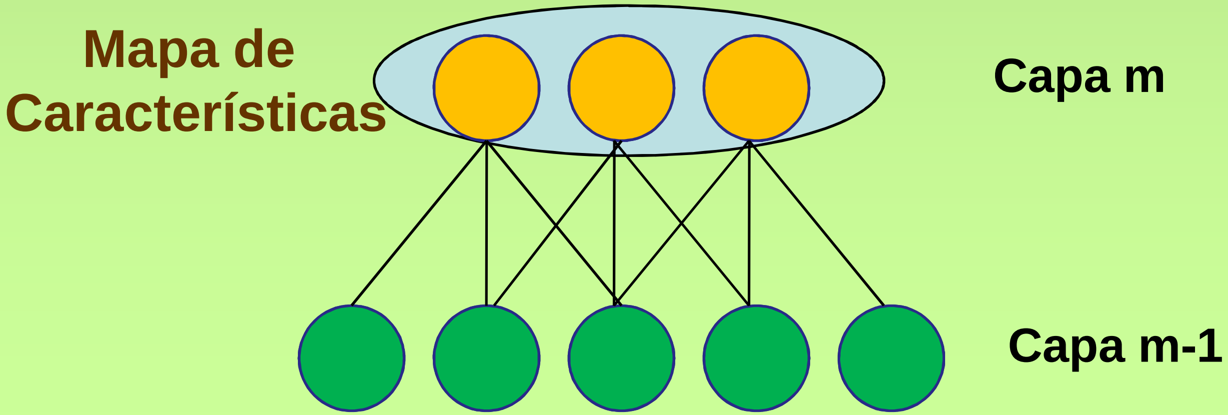
El tamaño de la capa oculta se puede cambiar utilizando otra variante de convolución

Compartir pesos y sesgos

Todas las neuronas en la capa oculta comparten la misma parametrización (vector de pesos y sesgos) formando un Mapa de Características

(Compartir pesos y sesgos) → *Kernel* o Filtro

Ahora, el gradiente de un peso compartido es la suma de los gradientes de los pesos que están siendo compartidos



Invarianza de imágenes: Transformaciones afines

Permiten detectar características independientemente de su posición en el campo visual. (**La característica es un tipo de patrón de entrada que hará que una neurona se active, por ejemplo, el borde de un objeto**).

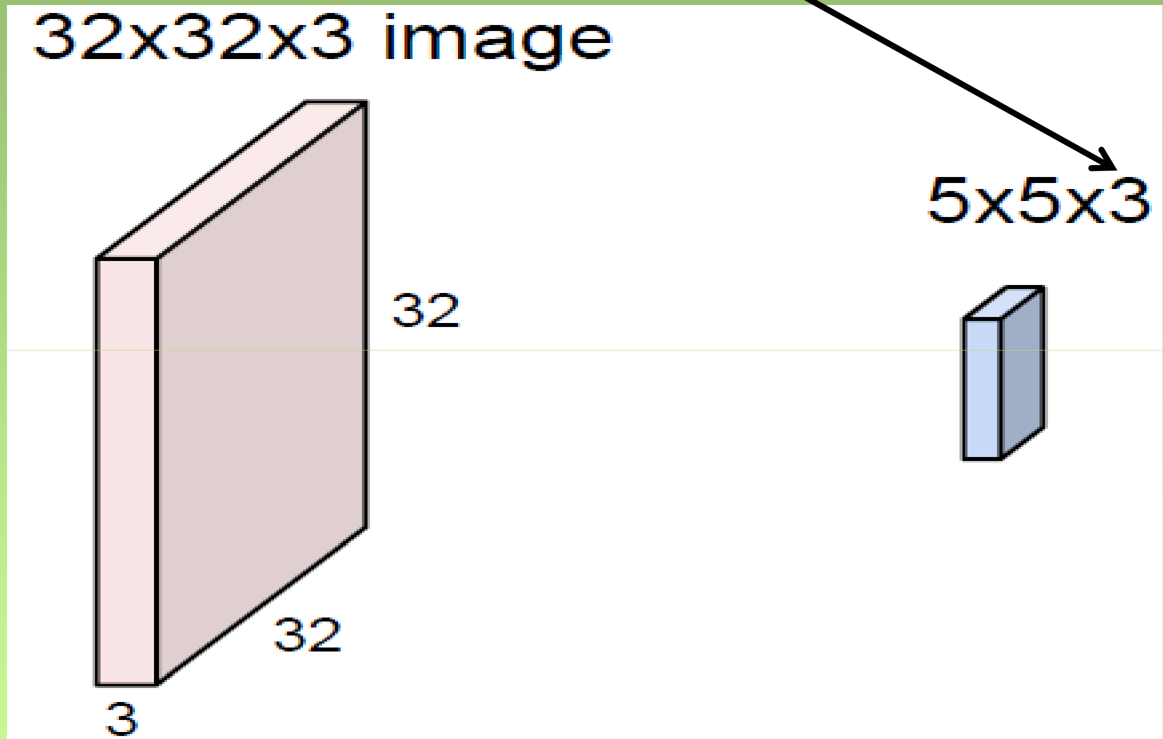
Todas las neuronas en la primera capa oculta detectan exactamente la misma característica, solo que en diferentes ubicaciones.

Los CNN están bien adaptadas a la invariancia de imágenes: movemos la imagen de un objeto, ¡y sigue siendo la imagen de un objeto!

Reduce aún más el número de parámetros libres, logrando una mejor generalización y rendimiento computacional.

Capa Convolucional

Los filtros siempre extienden la profundidad completa del volumen de entrada



filtro

Convolver el filtro con la imagen, es decir, "deslizarse sobre la imagen espacialmente, calculando productos escalares"

Capa Convolucional

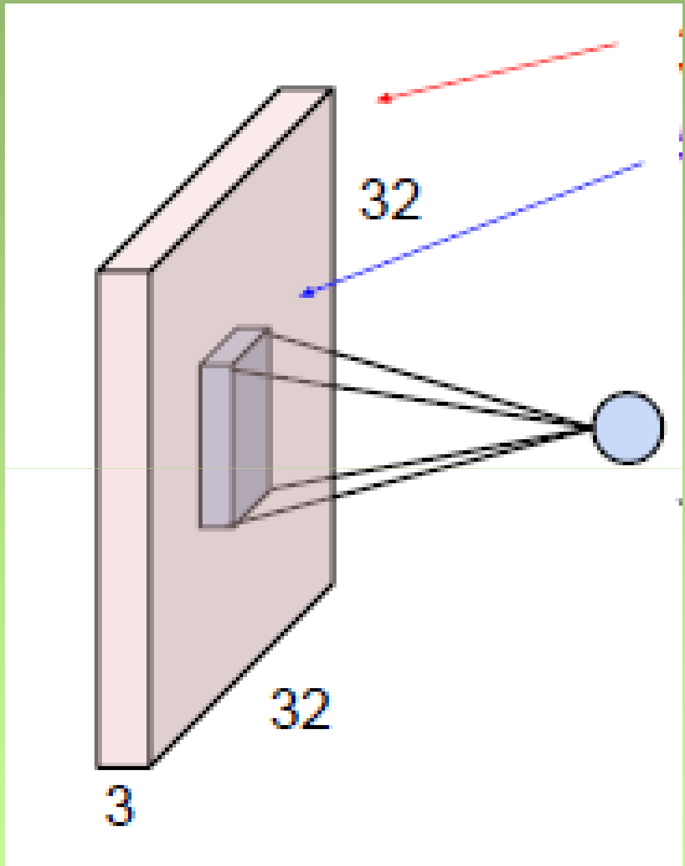


Imagen de 32x32x3

Filtro de 5x5x3 , w

Número 1: Resultado de calcular un producto escalar entre el filtro y un pequeño fragmento de 5x5x3 de la imagen (es decir, 5*5*3 = producto escalar de 75 dimensiones + sesgo)

$$\mathbf{w}^{\mathrm{T}}\mathbf{x} + b$$

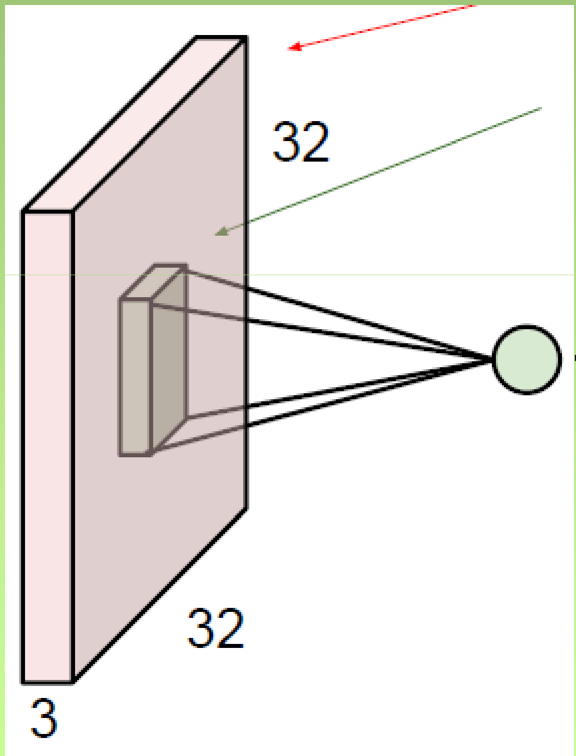
Capa Convolucional

Consideremos un segundo filtro verde

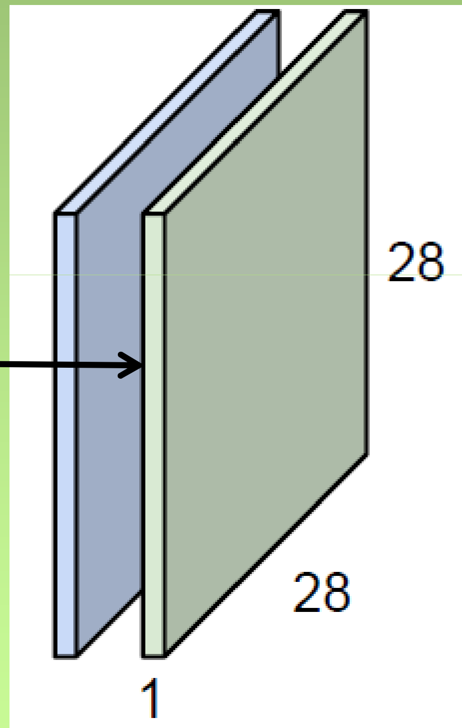
Mapas de características o de activación

Imagen de 32x32x3

Filtro de 5x5x3

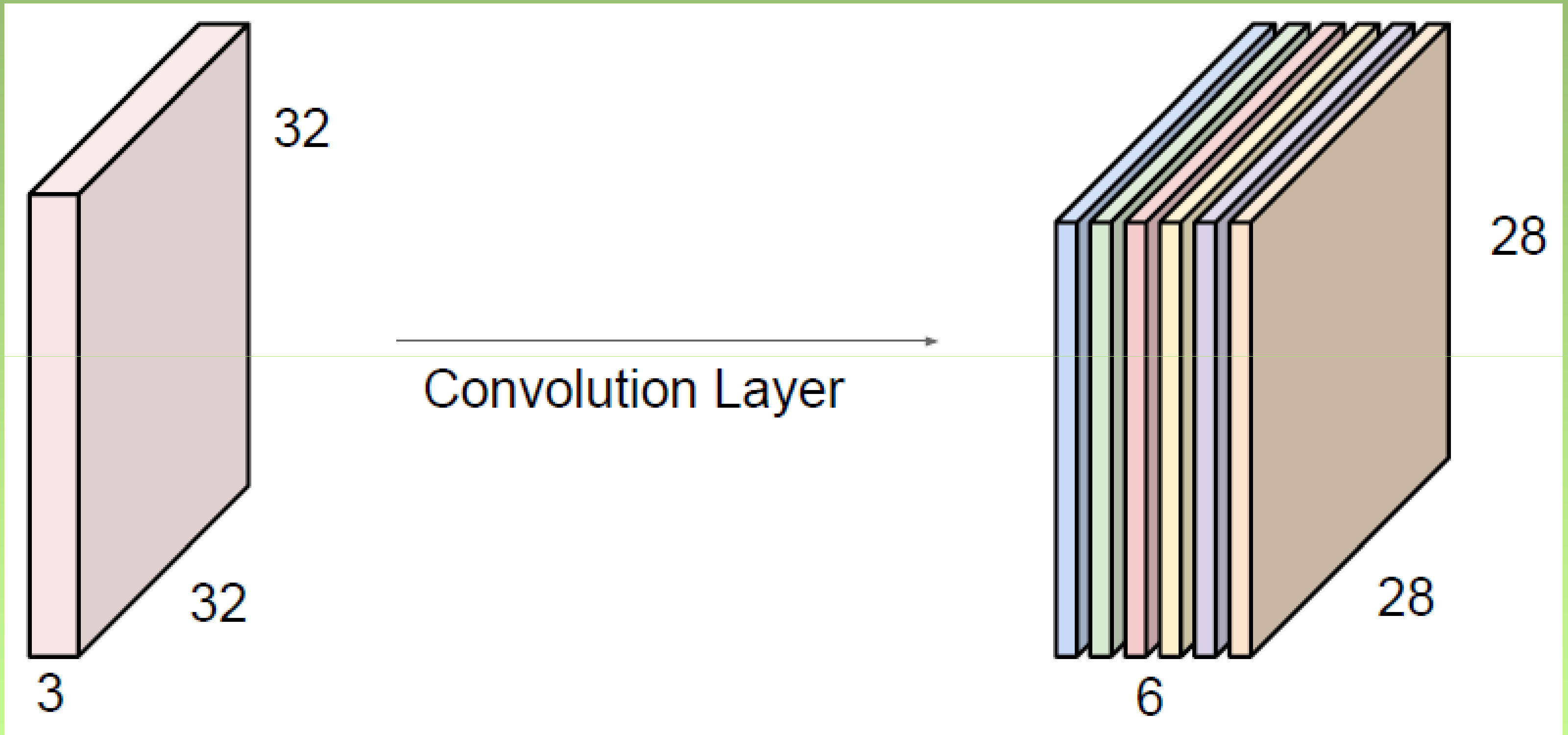


coevoluciona (se desliza) en todas las localizaciones espaciales



Capa Convolucional

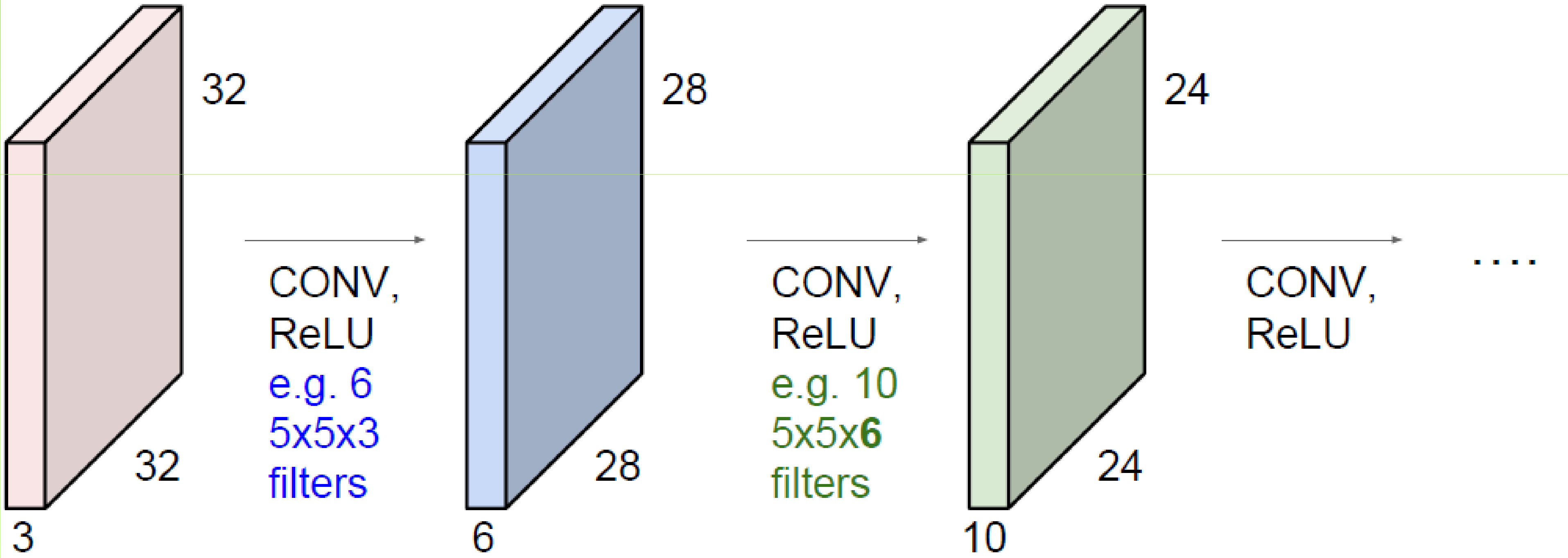
Si por ejemplo, tuviéramos 6 filtros de 5x5, obtendríamos 6 mapas de activación separados



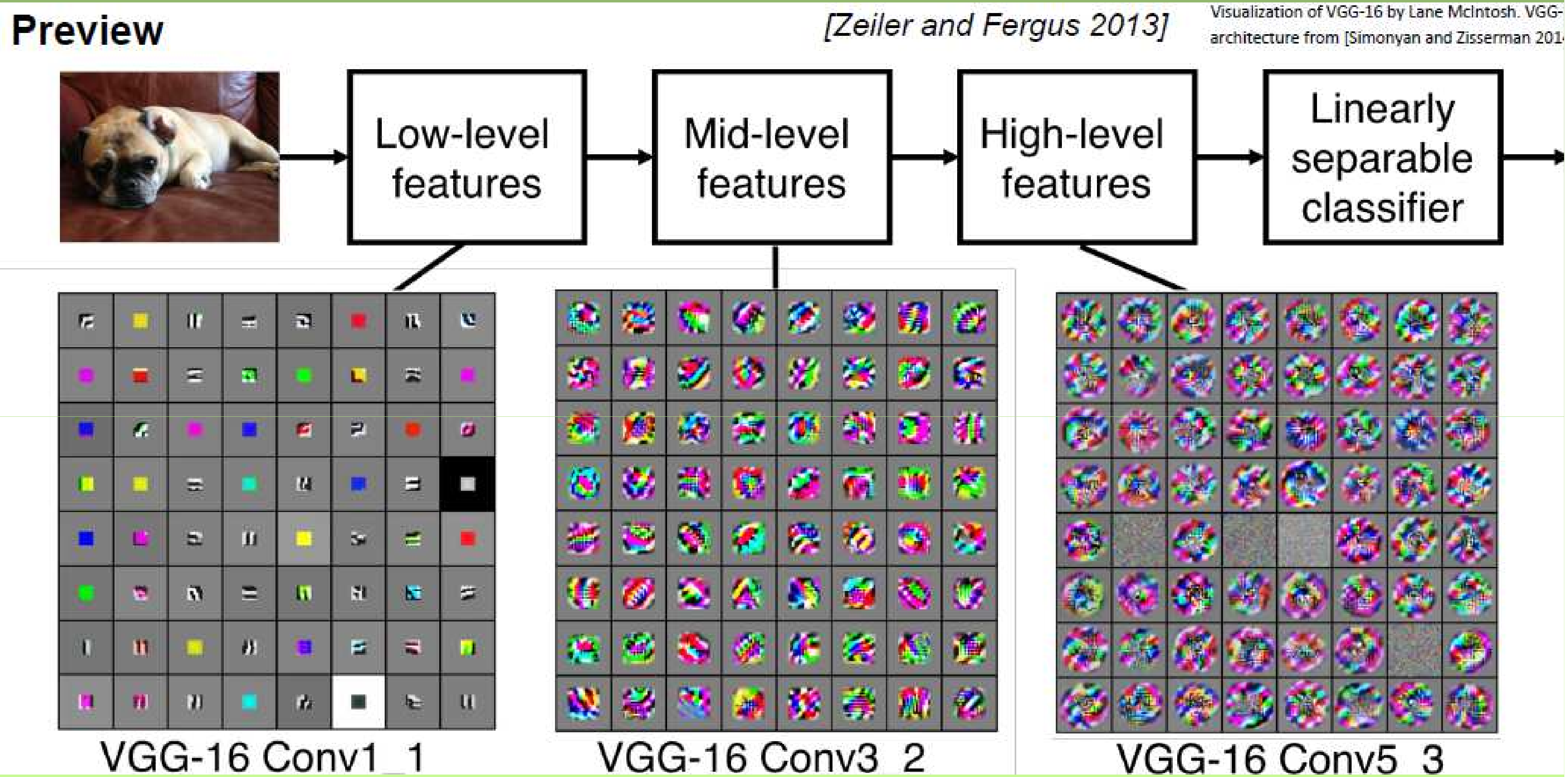
Apilamos estos mapas para obtener una "nueva imagen" de tamaño 28x28x6

Capa Convolucional

Vista previa: ConvNet es una secuencia de Capas Convolucionales, intercaladas con funciones de activación

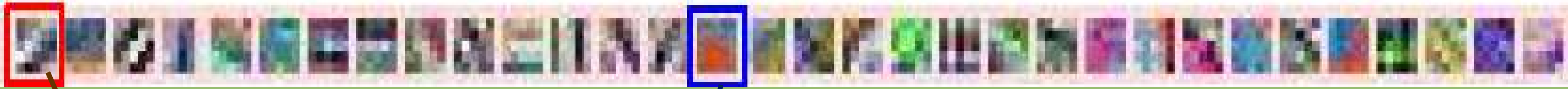


Capa Convolucional



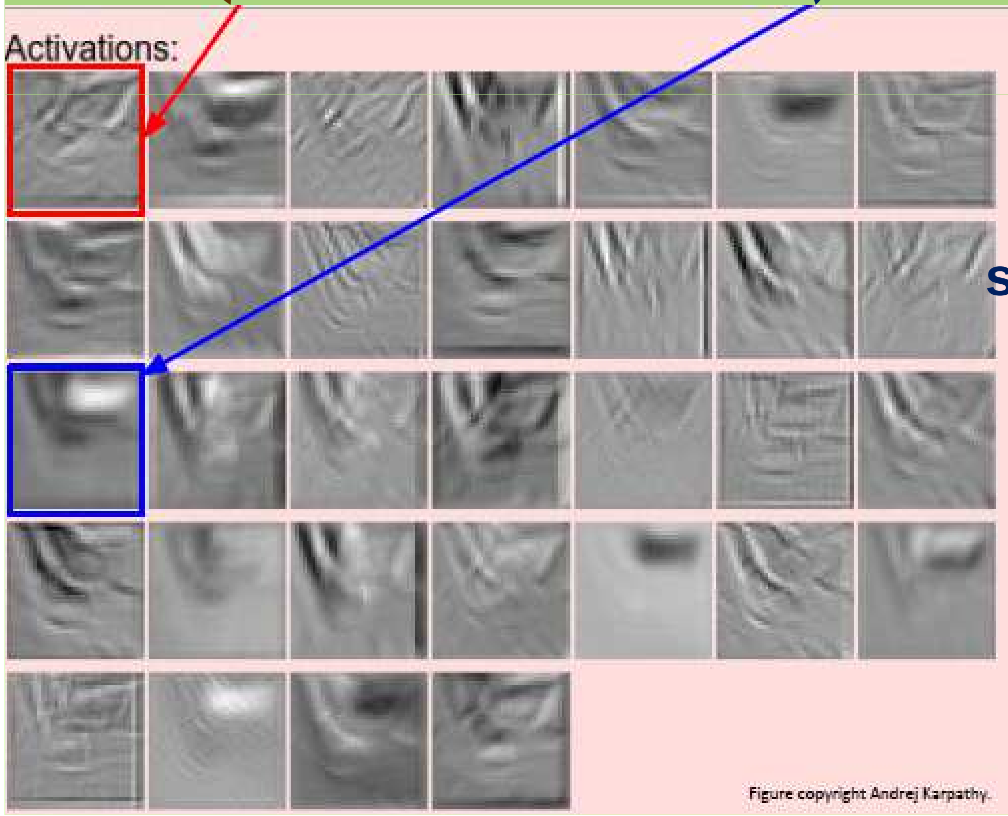
Capa Convolucional

Ejemplos de filtros 5x5 (32 en total)



Un filtro ⇨ **Un mapa de activación**

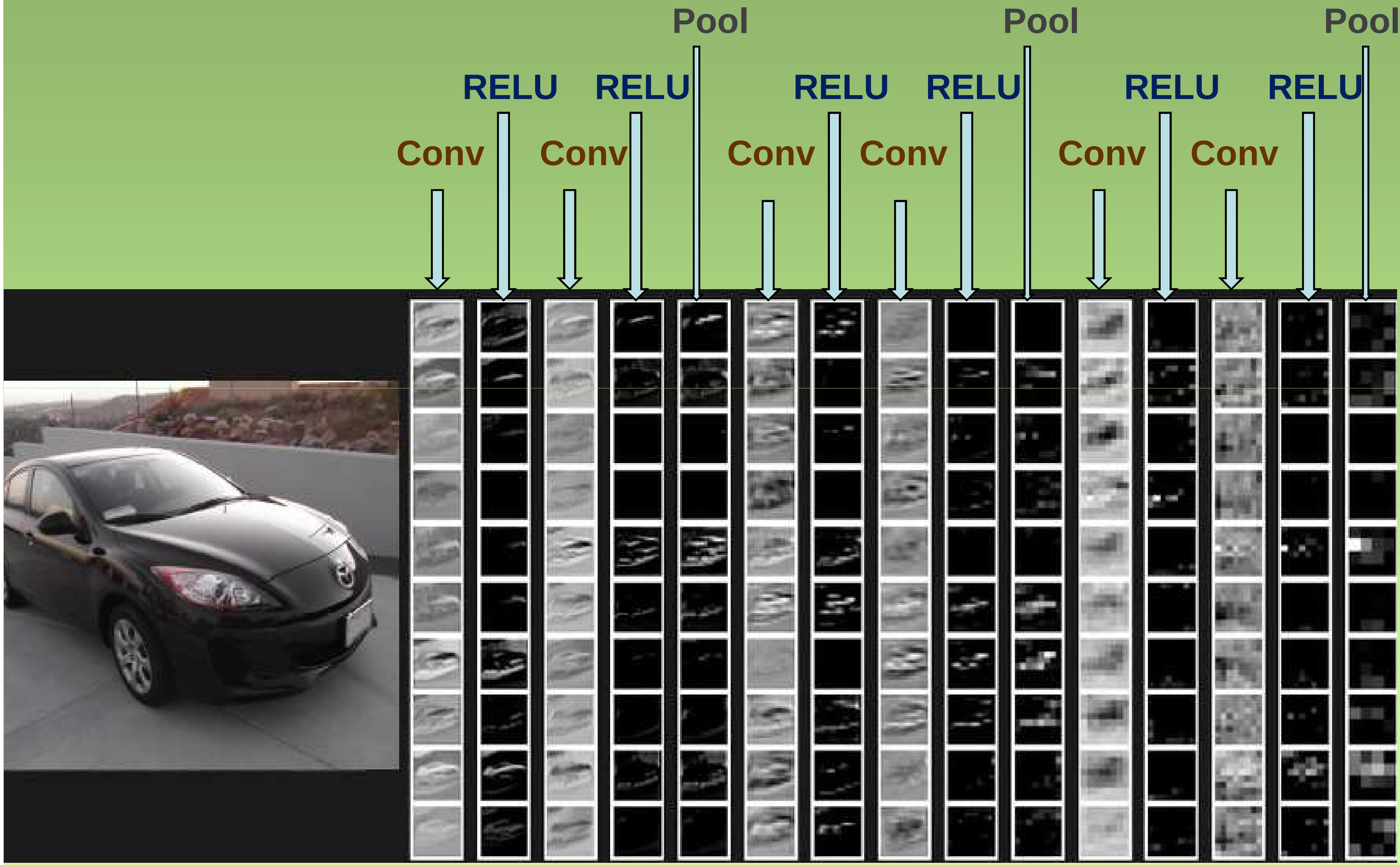
$$f[x,y]*g[x,y]=\sum_{n_1=-\infty}^{\infty}\sum_{n_2=-\infty}^{\infty} f[n_1,n_2].g[x-n_1,y-n_2]$$



multiplicación por elementos y suma de un filtro y la señal (imagen)

Llamamos a la capa convolucional porque está relacionada con la convolución de dos señales

Avance o vista anticipada



Salida: Coche FC 50%, Camión 25%, Barco 15%, Caballo 10%

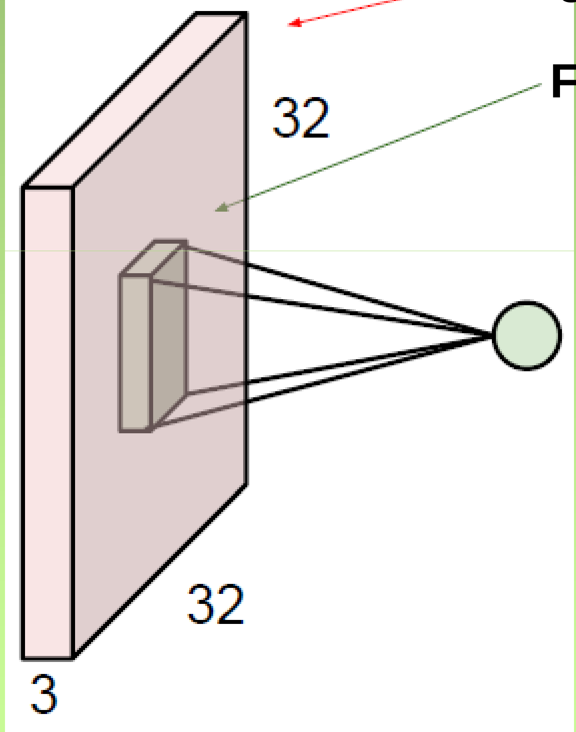
Capa Convolucional

Una mirada más cercana a las dimensiones espaciales:

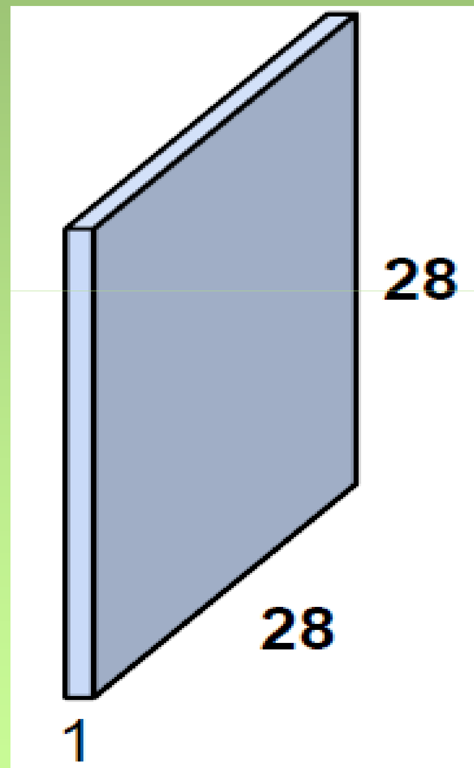
Mapa de activación

Imagen de 32x32x3

Filtro de 5x5x3

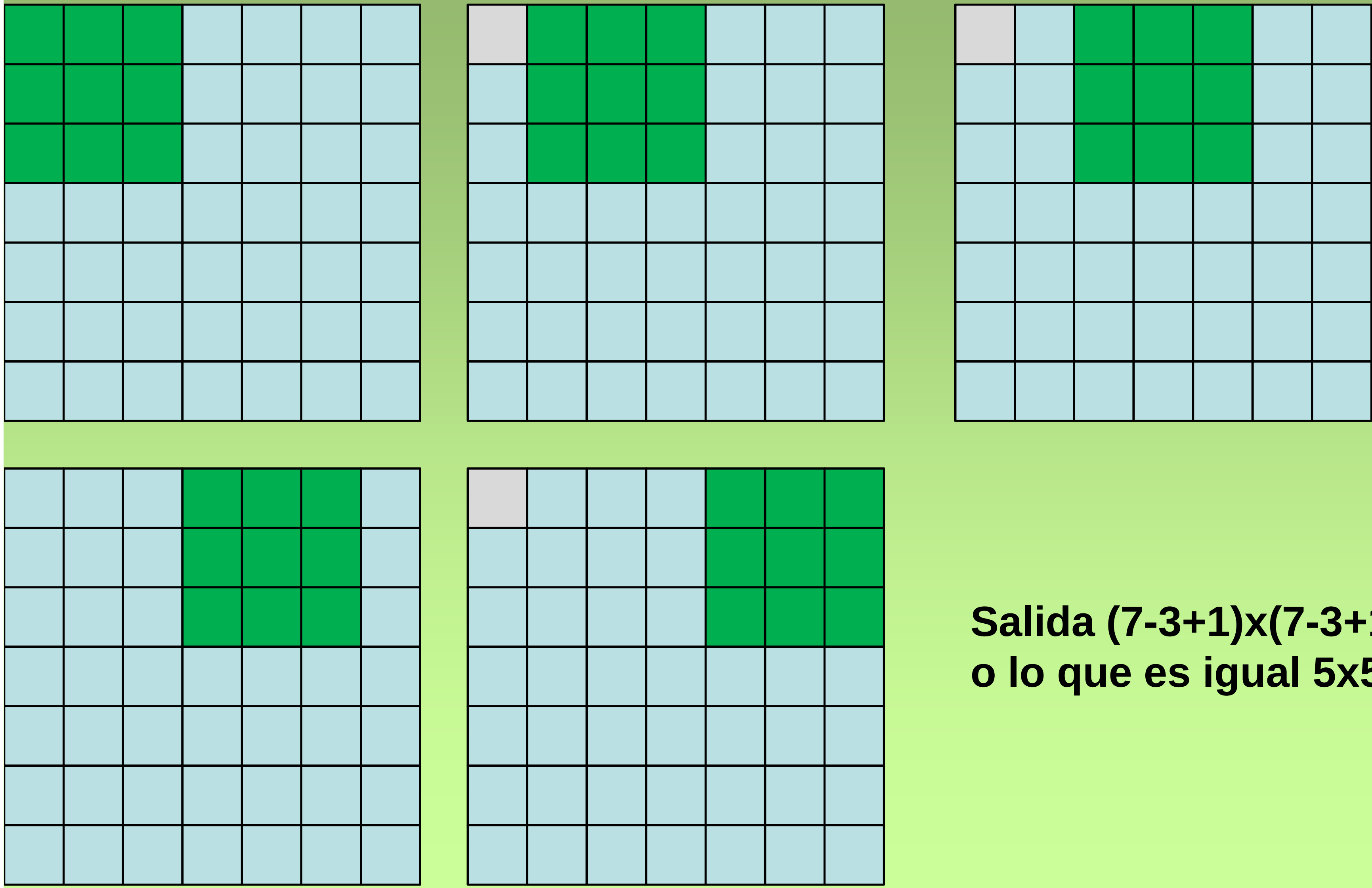


convoluciona (se desliza) en todas las localizaciones espaciales



Una mirada más cercana a las dimensiones espaciales:

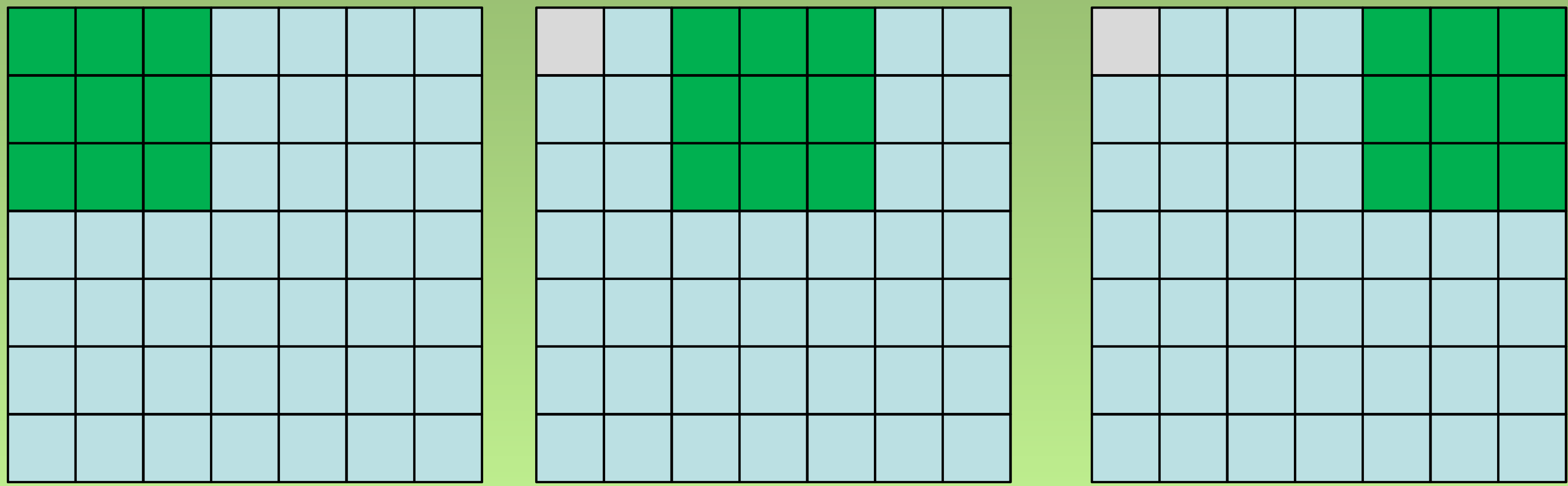
La entrada de 7x7 (espacialmente) se le aplica un filtro de 3x3



Salida (7-3+1)x(7-3+1)
o lo que es igual 5x5

Una mirada más cercana a las dimensiones espaciales:

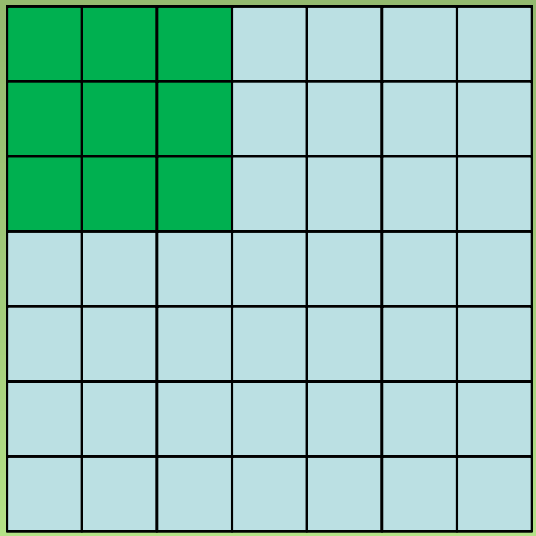
Si a la entrada de 7x7 (espacialmente) se le aplica un filtro de 3x3 con espaciado 2



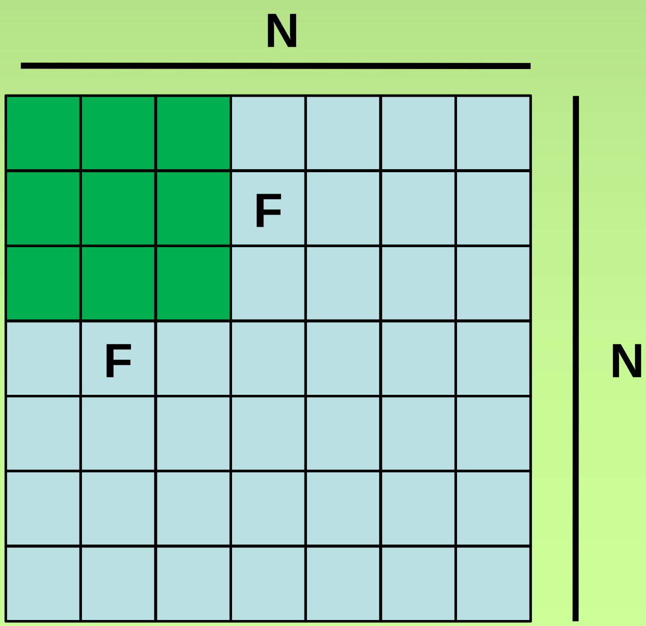
La entrada de 7x7 (espacialmente) supone un filtro de 3x3 aplicado con espaciado 2
=> Salida ((7-3)/2)+1)x((7-3)/2)+1)=3x3

Una mirada más cercana a las dimensiones espaciales:

La entrada de 7x7 (espacialmente) se le puede aplicar un filtro de 3x3 con paso S=3?



No es posible. No se puede aplicar el filtro 3x3 a la entrada 7x7 con *stride* 3.



Tamaño de salida $\frac{N-F}{S}+1$

Ejemplo, si N=7 y F=3, tenemos

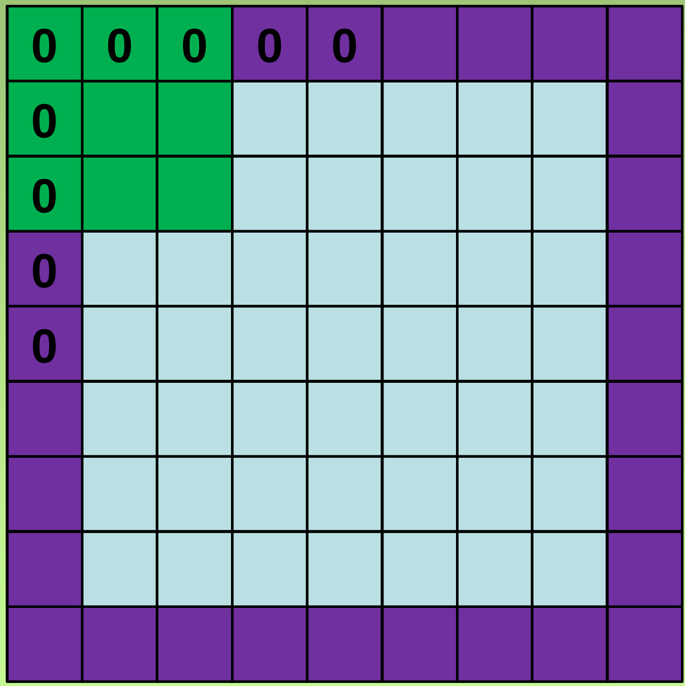
((7-3)/1)+1=5

((7-3)/2)+1=3

((7-3)/3)+1=2.33; no es un nº entero

Una mirada más cercana a las dimensiones espaciales:

En la práctica: Rellenamos el borde con ceros



Por ejemplo. Para una entrada 7x7 con un filtro 3x3, aplicado con un stride de 1 y con relleno de un pixel de borde => ¿Cuál es la salida?

La salida es de 7x7; pues tenemos [((9-3)/1)+1]x[((9-3)/1)+1]

En general, es común ver capas CONV con paso 1, filtros de tamaño FxF y cero relleno con (F-1) / 2. (preservará el tamaño espacialmente)

p.ej. F = 3 => relleno cero con paso de 1

F = 5 => relleno cero con paso de 2

F = 7 => relleno cero con paso de 3

Ejemplos

Recordemos

Una entrada de 32x32 convolucionada repetidamente con filtros de 5x5 reduce los volúmenes espacialmente y se pasa de (32 -> 28 -> 24 ...).

Reducir demasiado rápido no es bueno, no funciona bien.

Volumen de entrada: 32x32x3
10 filtros de 5x5 con paso 1, y relleno de 2

Tamaño del volumen de salida:
(32 + 2*2-5) / 1 + 1 = 32 espacialmente,
entonces la salida es 32x32x10

Volumen de entrada: 32x32x3
10 filtros de 5x5 con paso 1, relleno 2

¿Número de parámetros en esta capa?
cada filtro tiene 5*5*3 + 1 = 76 pesos (+1 para el sesgo) => 76*10 = 760

Resumen de la Capa Convolucional o CONV

1) Se acepta un volumen de tamaño W_1 x H_1 x D_1

2) Introducir los hiperparámetros

Número de filtros K,

Tamaño espacial del filtro F,

Paso o *stride* S,

La cantidad de relleno con ceros P.

Ajustes comunes:

K=(potencia de 2, p.e. 32, 64, 128,512

F=3, S=1, P=1

F=5, S=1, P=2

F=5, S=2, P= Cualquier fijación

F=1, S=1, P=0

3) Producir un volumen de tamaño W_2 x H_2 x D_2 donde

$$W_2 = \frac{W_1 - F + 2P}{S} + 1; \qquad H_2 = \frac{H_1 - F + 2P}{S} + 1,$$

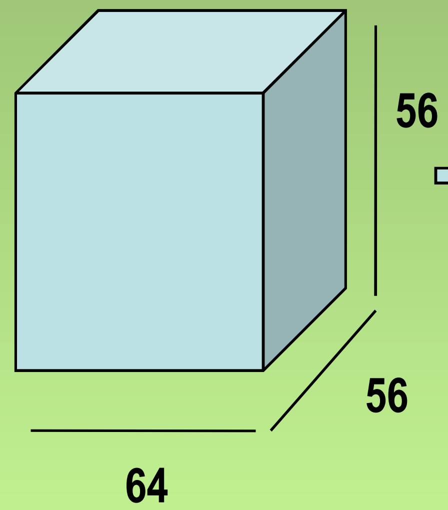
$$(esto\ es\ el\ ancho\ y\ el\ alto\ se\ computan\ igual\ por\ simetría)$$

$$D_2 = K$$

4) Con parámetros compartidos , se introducen F*F*D_1 pesos por filtro, con lo que tenemos (F*F*D_1)*K pesos y K sesgos (un sesgo para cada filtro).

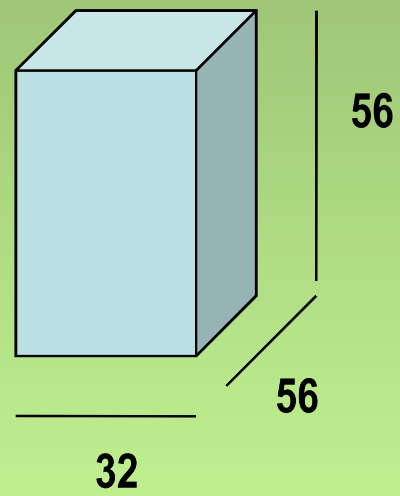
5) Como volumen de salida tenemos, una rebanada de profundidad d (de tamaño W_2 x H_2) como resultado de realizar una convolución válida del d-ésimo filtro sobre el volumen de entrada con un paso de S, y acompañado por el d-ésimo sesgo.

(Las capas de convolución 1x1 tienen perfecto sentido)



1x1 CONV con 32 filtros

(cada filtro tiene un tamaño de 1x1x64 y realiza un producto escalar de 64 dimensiones)



Ejemplo: Capa CONV enTorch

SpatialConvolution

module=nn.SpatialConvolution (nInputPlane, nOutputPlane, KW, KH, [dW], [dH], [padW], [padH]

Applies a 2D convolution over an input image composed of several input planes. The input tensor in forward (input) is expected to be a 3D tensor (nInputPlane x height x width)

The parameters are the following:

. nInputPlane: The number of expected input planes in the image given info format()

. nOutputPlane: The number of output planes the convolution layer will produce.

KW: The kernel width of the convolution

KH: The kernel height of the convolution

dW: The step of the convolution in the width dimension. Default is 1

dH: The step of the convolution in the height dimension. Default is 1

psdW: The additional zeros added per width to the input planes. Default is 0, a good number is (KW-1)/2

psdH: The additional zeros added per height to the input planes. Default is psdW, a good number is (KW-1)/2

Ejemplo: Capa CONV enTorch Continuación

Note that depending of the size of your kernel, several (of the last) columns or rows of the input image might be lost. It is up to the user to add proper padding in images.

If the input image is a 3D tensor (nInputPlane x height x width), the output image size will be (nOutputPlane x oheight x owidth), where

owidth = floor((width+2*padW-kW)/dW +1)

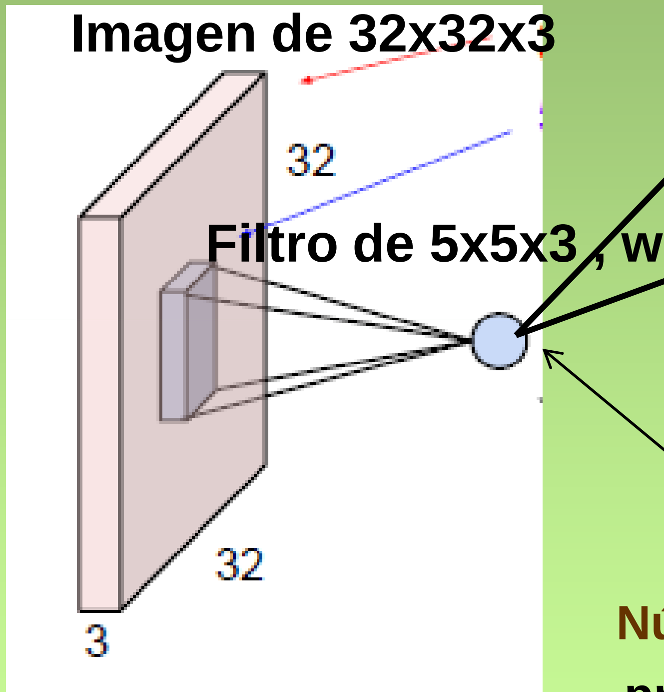
oheight = floor((height+2*padH-kH)/dH +1)

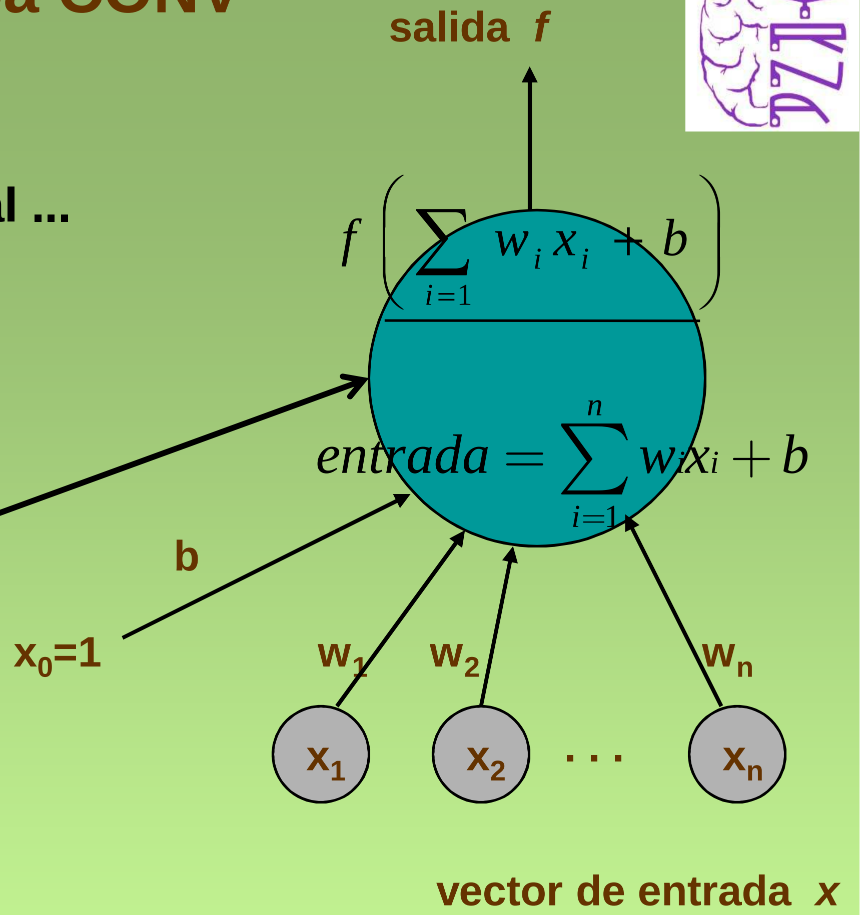
Caffe: Capa de convolución

```
layers {
          name: "conv1"
          type:        "CONVOLUTION"
          bottom: "data"
          top: "conv1"
          # learning rate and decay multipliers for the filters
          param {lr_mult: 1 decay_mult:1}
          # learning rate and decay multipliers for the biases
          param {lr_mult: 2 decay_mult:0}
          #         (o de forma alternativa)
          #         blobs_lr: 1
          #         blobs_lr: 2
          #         weight_decay: 1
          #         weight_decay: 0
          convolution_ param {
                    num_output: 96         # learn 96 filters
                    kernel_size:7                    # each filter is 7x7
                    stride: 4              # step 4  pixels between each filter application
                    weight_filter {
                              type: " gaussian"       # initialize the filters from a Gaussian
                              std: 0.01               # distribution with stdev 0.01 (default mean: 0)
                    }
                    bias_filter {
                              type: "constant"        # initialize the biases to zero (0)
                              value: 0
                    }
          }
}
```

Neurona de la capa CONV

Es solo una neurona con conectividad local ...

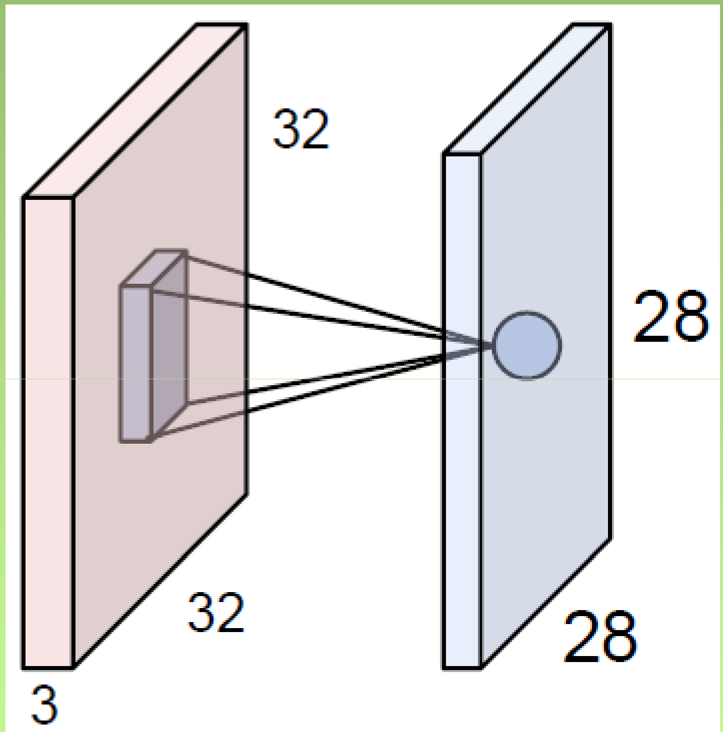




Número 1: Resultado de calcular un producto escalar entre el filtro y un pequeño fragmento de 5x5x3 de la imagen (es decir, 5*5*3 = producto escalar de 75 dimensiones + sesgo)

$$\mathbf{w}^{\mathrm{T}}\mathbf{x} + b$$

Neurona de la capa CONV



Un mapa de activación es una hoja de 28x28 de salidas de neuronas:

1. Cada una está conectada a una pequeña región de la entrada

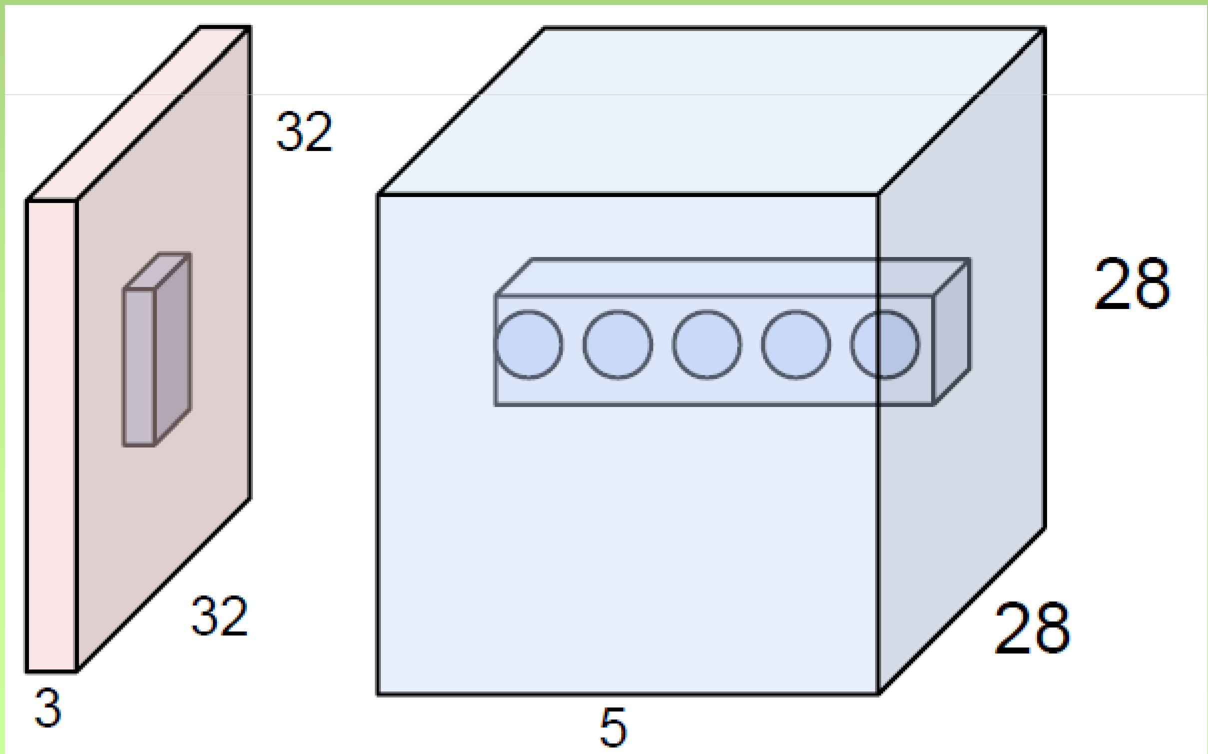
2. Todas ellas comparten parámetros

"Filtro 5x5" -> "campo receptivo 5x5 para cada neurona"

Neurona de la capa CONV

Por ejemplo con 5 filtros, la capa CONV consta de neuronas dispuestas en una cuadrícula 3D (28x28x5)

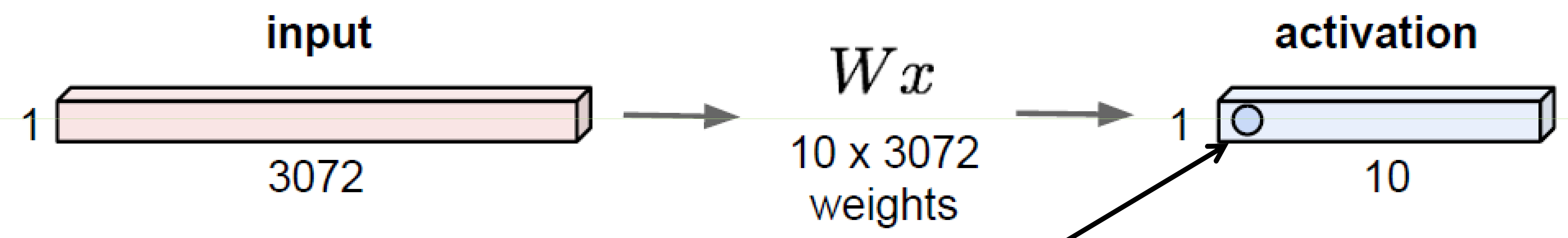
Habrá 5 neuronas diferentes todas mirando a la misma región en el volumen de entrada



Capa completamente conectada

Cada neurona mira el volumen de entrada completo

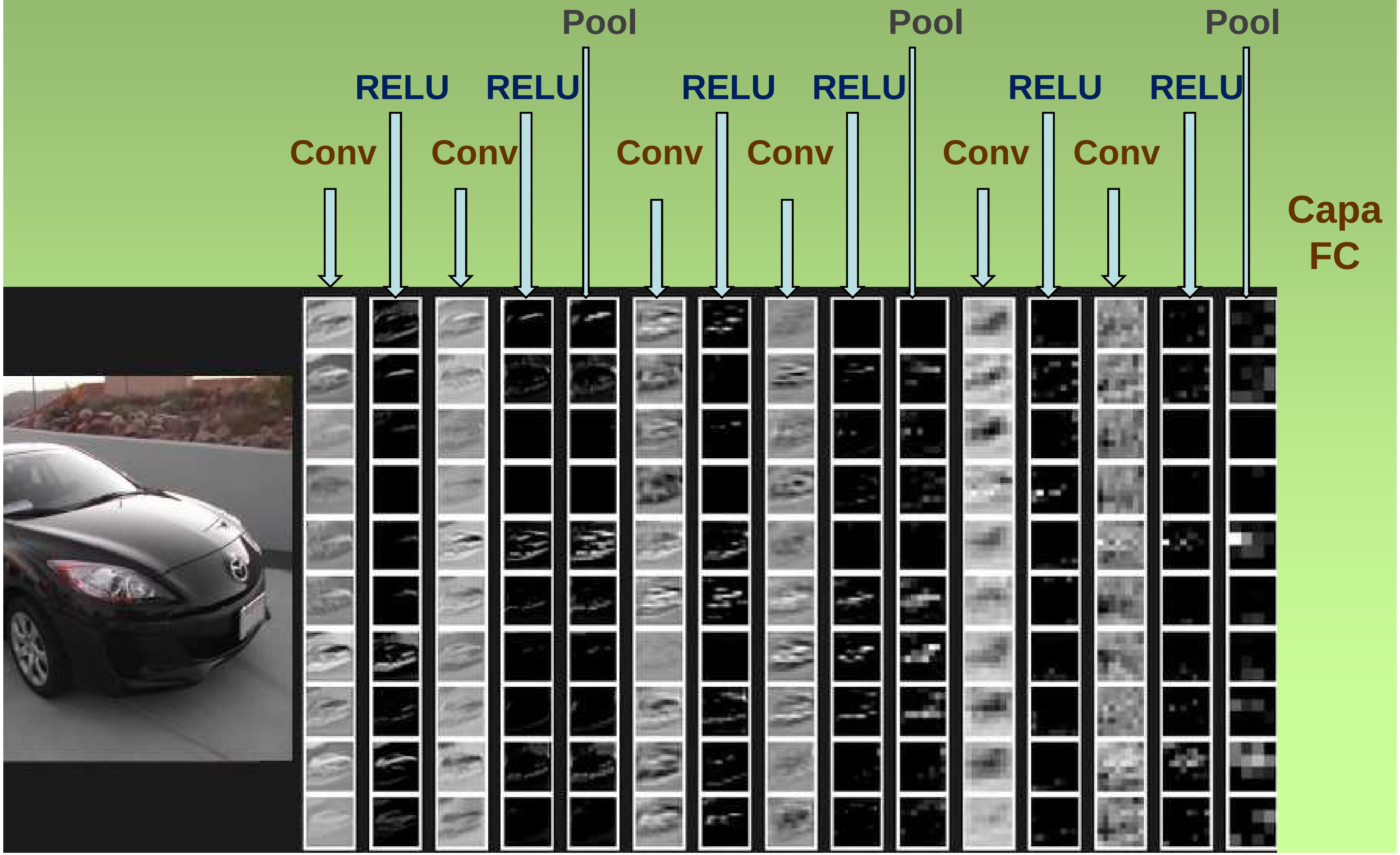
La imagen de 32x32x3 es transformada a la dimensión 1 x 3072



Número 1: Es el resultado de realizar un producto escalar entre una fila de W y la entrada x (un producto escalar de dimensión 3072)

Capa completamente conectada (capa FC)

- Contiene neuronas que se conectan a todo el volumen de entrada, como en Redes Neurales ordinarias



Resumen

- ConvNets stack CONV, POOL, FC capas

- Tendencia hacia filtros más pequeños y arquitecturas más profundas

- Tendencia a deshacerse de las capas POOL / FC (solo CONV)

- Las arquitecturas típicas parecen ser

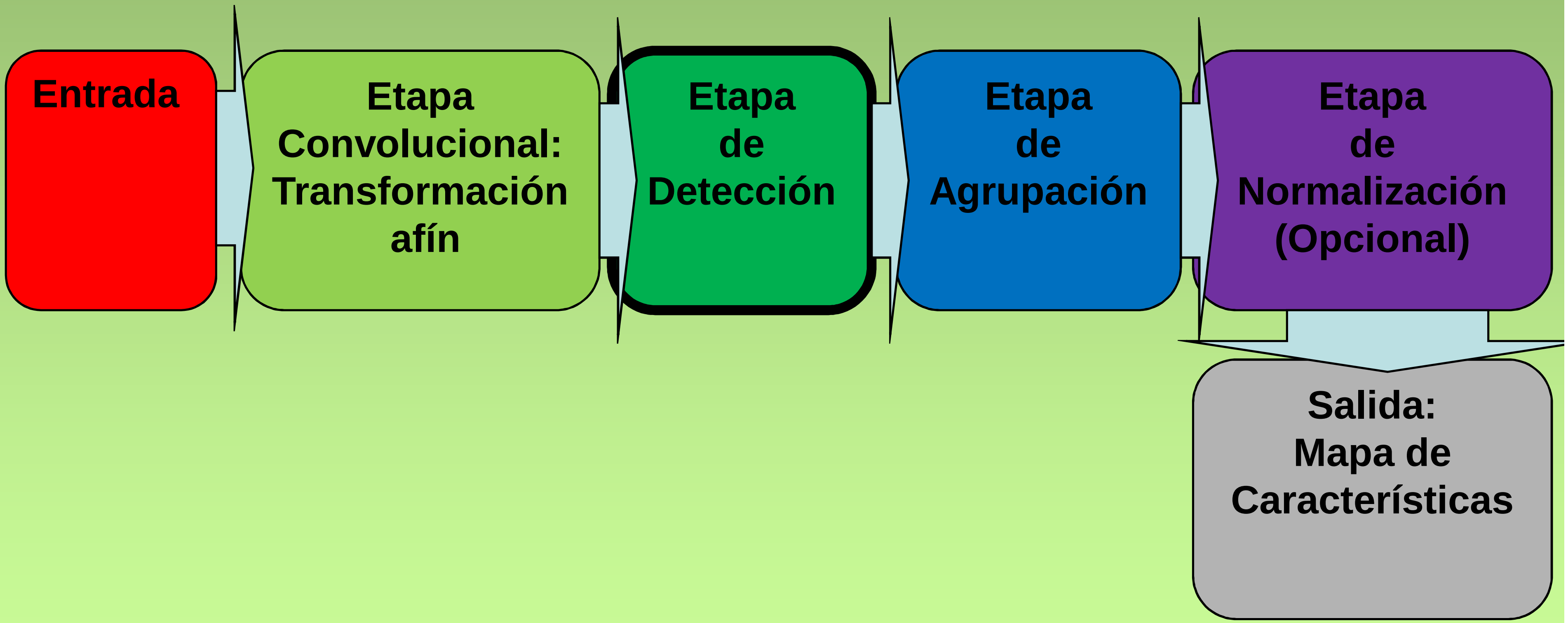
[(CONV-RELU) * N-POOL?] * M- (FC-RELU) * K, SOFTMAX

donde N es usualmente de hasta ~ 5, M es grande, 0 <= K <= 2.

-Pero recientes avances como ResNet / GoogLeNet desafian este paradigma

-[ConvNetJS demo: training on CIFAR-10]

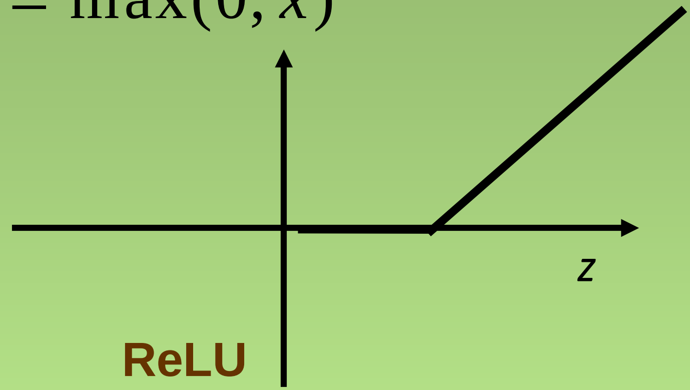
Estructura de las capas de una CNN



Funciones de Activación No Lineales

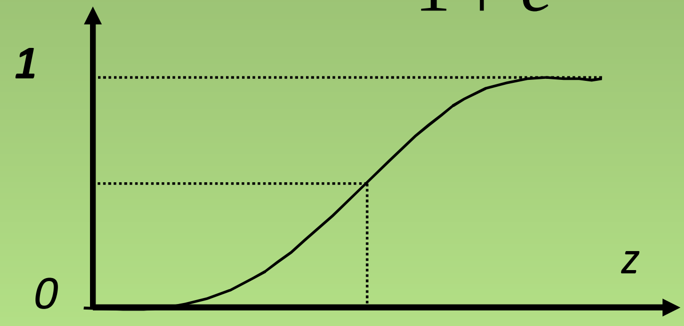
Unidad Lineal Rectificada

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



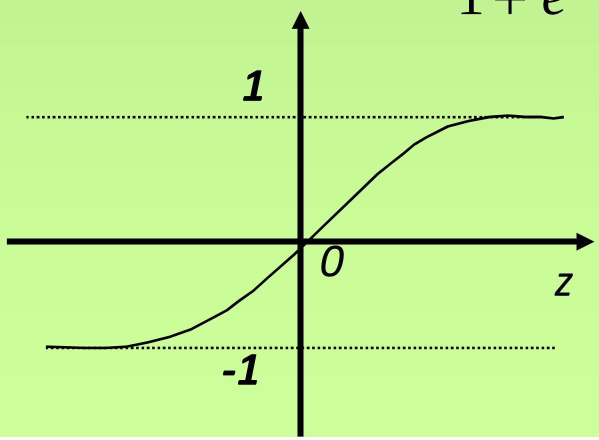
Activación Logistica

$$\phi(z) = \frac{1}{1+e^{-z}}$$



Activación Tangente Hiperbólica

$$\phi(u) = tanh(u) = \frac{1-e^{-u}}{1+e^{-u}}$$



La función ReLU es la función de activación más popular para DNN desde el año 2015, evita saturaciones y hace más rápido el aprendizaje

Compartir pesos y convoluciones: Explota la estacionariedad espacial

Características que son útiles en una parte de la imagen son probablemente útiles en otra parte

Todas las unidades comparten el mismo conjunto de pesos

Cambio de equivalencia de procesamiento
Cuando las entradas cambian, las salidas también cambian pero en otro caso permanecen sin cambios.

Convolución
Con un kernel de aprendizaje (o filtro)
No-linearidad: ReLU (rectificado lineal)

$$A_{ij} = \sum_{k,l} W_{kl} X_{i+j,k+l}$$

A la imagen filtrada Z se le llama mapa de características "feature map"

$$Z_{ij} = \max(0, A_{ij})$$

Etapa de Detección

Mapa de Características.- Se obtiene por convolución de la imagen mediante un filtro lineal, añadiendo un término de sesgo y aplicando una función de transferencia no lineal

Se necesita un número de mapas de característica en cada capa para capturar un número suficiente de características en la imagen

Sea el k-ésimo mapa de características en una capa dada, x^k cuyos filtros están determinados por $\mathbf{W}_k$ y sesgo $\mathbf{b}_k$, entonces se obtiene x^k con función sigmoide σ para la no linealidad y un filtro de tamaño mxm en la forma

$$x_{ij}^{k} = \sigma\left((\mathbf{W}^{k} * a)_{ij} + b_k\right) = \sigma\left[\left(\sum_{a=0}^{m-1}\sum_{b=0}^{m-1} w_{ab}\, y_{(i+a)(j+b)}^{k-1}\right)_{ij} + b_k\right]$$

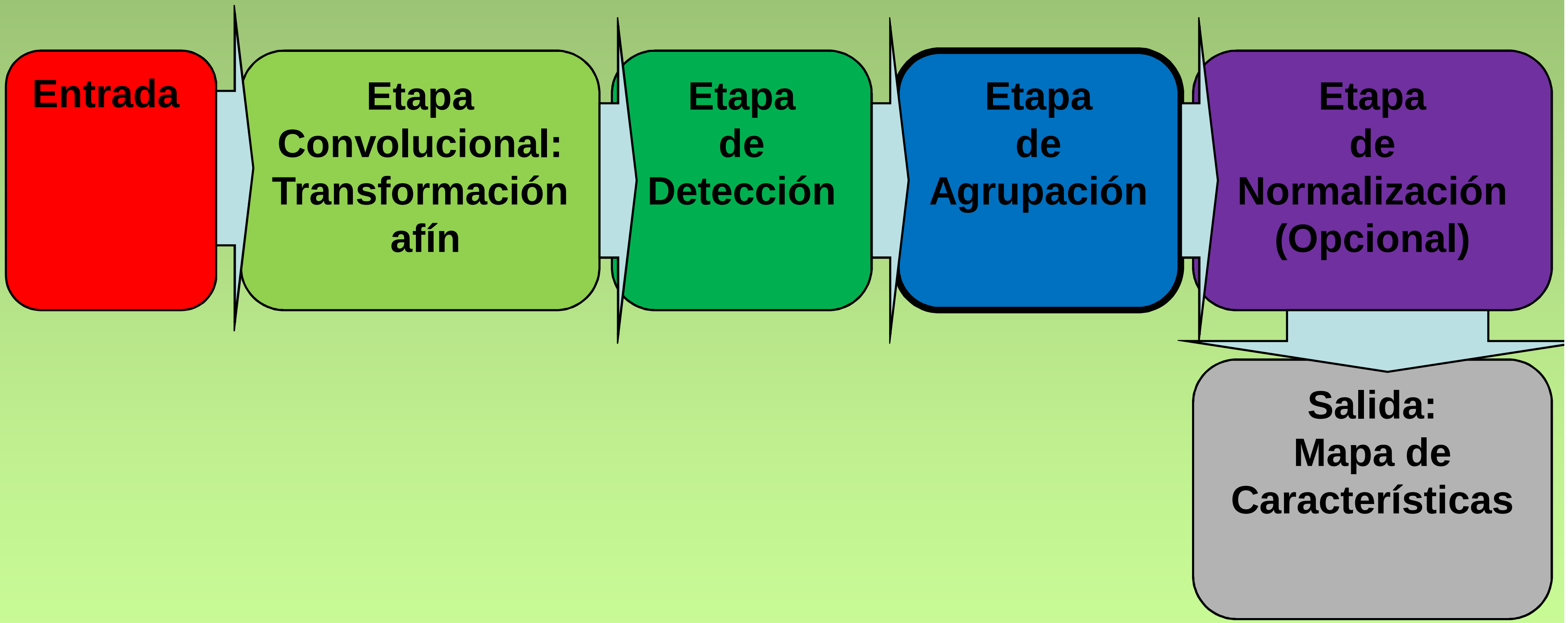
Etapa de Detección

Cada capa oculta está formada por múltiples mapas de características, $\{x^k, k = 0, \ldots, K\}$

La matriz de pesos, W, de una capa oculta puede representarse mediante un tensor de dimensión 4D conteniendo elementos de cada combinación de (mapas de características de destino, mapa de características fuente, posición vertical fuente y posición horizontal fuente)

Sesgos, b, los cuales pueden representarse mediante un vector que contenga un elemento por cada mapa de características destino

Estructura de las capas de una CNN



Reducción “Pooling”

Muestreo no lineal para simplificar la información en la salida de la capa convolucional

Variantes:

- Máximo agrupamiento (es el más polular)
- Media ponderada basada en distancia
- Norma L_2 de los vecinos

Reduce el cálculo de las capas superiores mediante informes resumidos de estadísticas (solo con *stride* > 1).

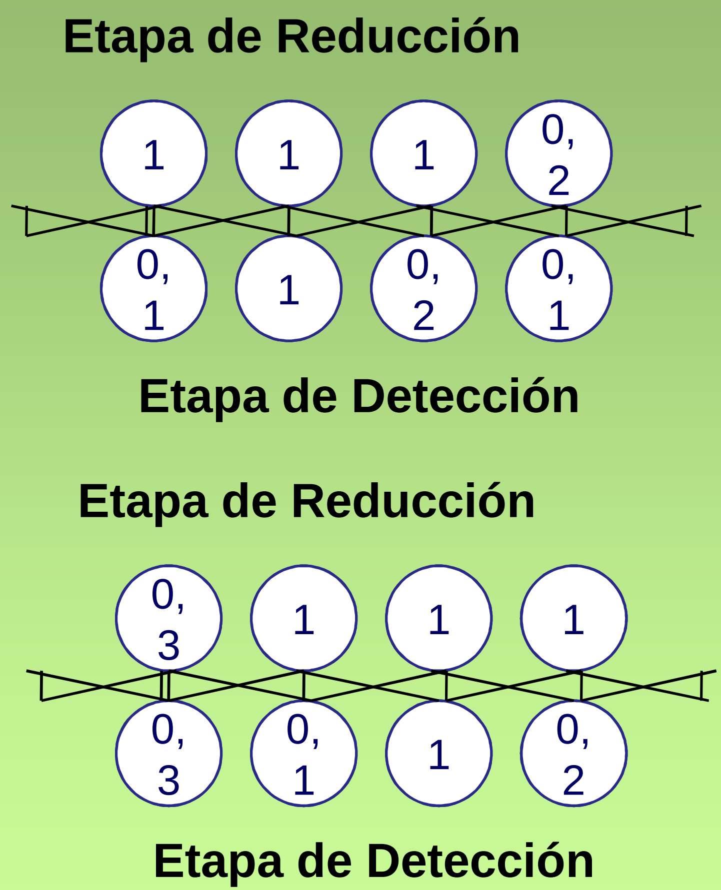
Proporciona invarianza a la traslación (dado que antes del aprendizaje debe ser invariante a las traslaciones pequeñas)

Propiedad útil, dado que nos preocupamos más por si algunas características están presentes que exactamente donde se encuentran, por lo tanto, se agrega robustez a la posición.

Reducción: *Pooling*

La vista inferior ha sido desplazada por 1 píxel versus la vista superior.

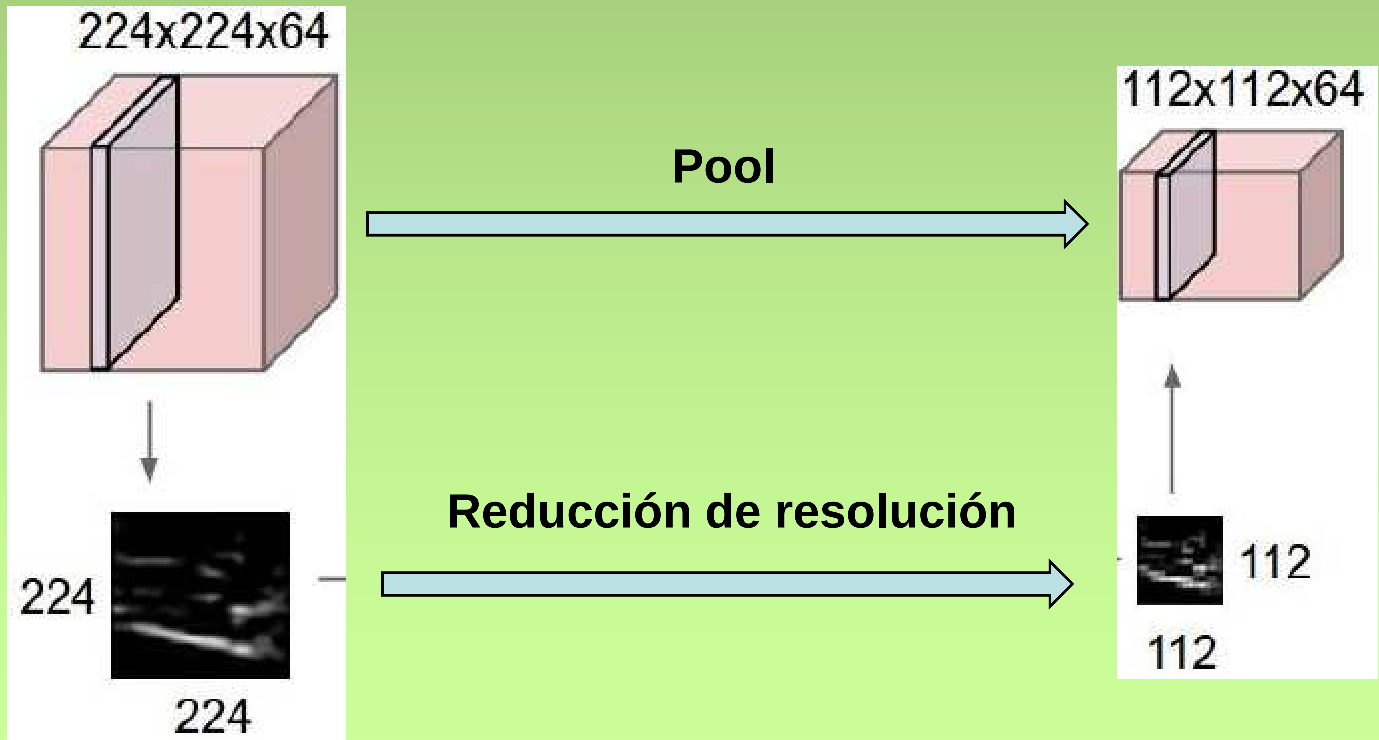
¡Cada valor en la fila inferior ha cambiado, pero solo la mitad de los valores en la fila superior ha cambiado!



Capa de reducción: *Pooling*

-**Hace las representaciones más pequeñas y más manejables**

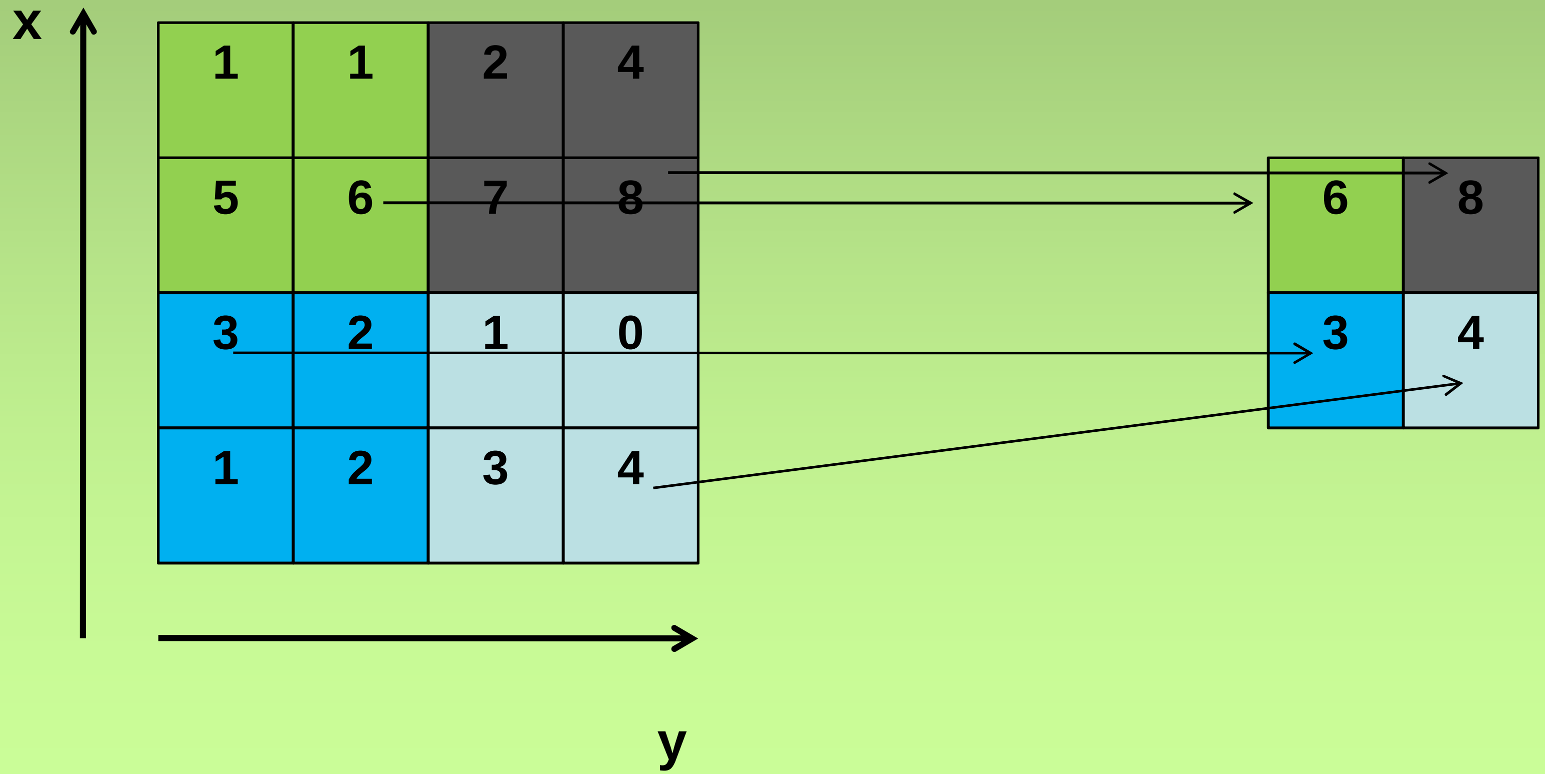
- Opera en cada mapa de activación de forma independiente:



Max Pool

Máx Pool con filtros 2x2
y paso de 2

Mapa de activación
Rebanada de profundidad 1



Pooling

1) Se acepta un volumen de tamaño W_1 x H_1 x D_1

2) Introducir los tres hiperparámetros

Tamaño espacial del filtro F,

Paso o *stride* S,

Configuración común:
F = 2, S = 2
F = 3, S = 2

3) Produce un volumen de tamaño W_2 x H_2 x D_2 donde

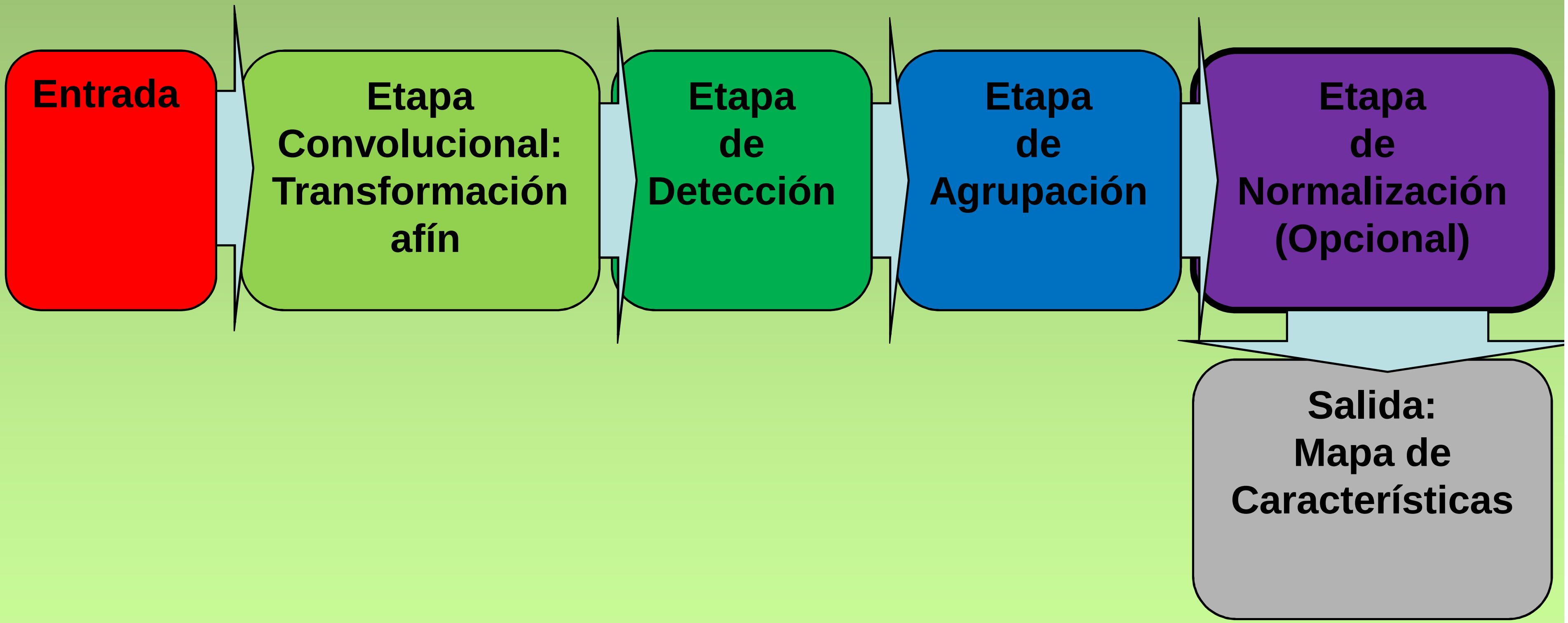
$$W_2 = \frac{W_1 - F}{S} + 1; \qquad H_2 = \frac{H_1 - F}{S} + 1,$$

$$D_2 = D_1; \qquad P = 0$$

Introduce cero parámetros ya que calcula una función fija de la entrada

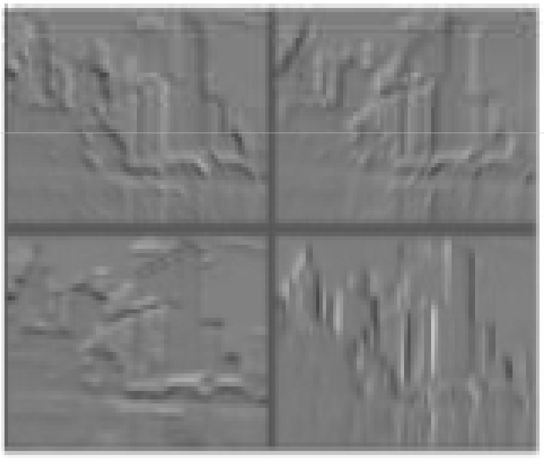
Hay que tener en cuenta que no es común usar relleno cero para agrupar capas

Estructura de las capas de una CNN

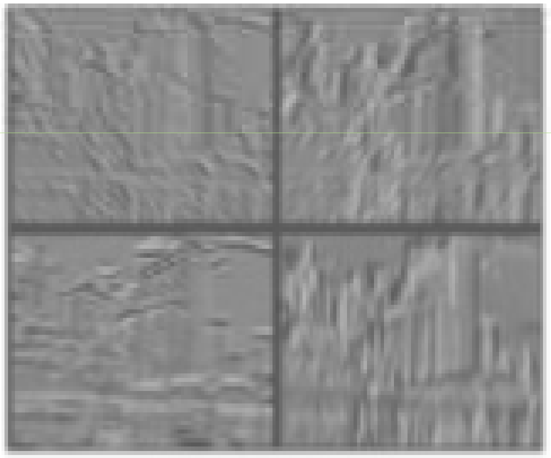


Normalización (Opcional)

La respuesta es normalizada localmente utilizando alguna distancia basada en una función promedio ponderada

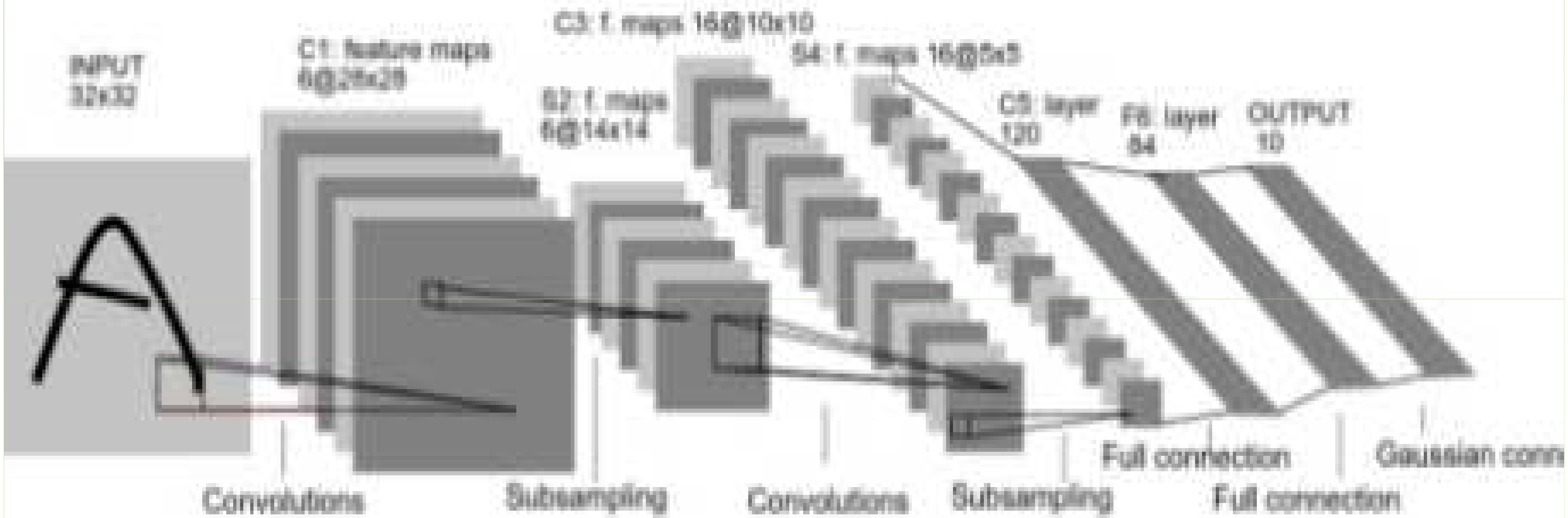


Mapa de características

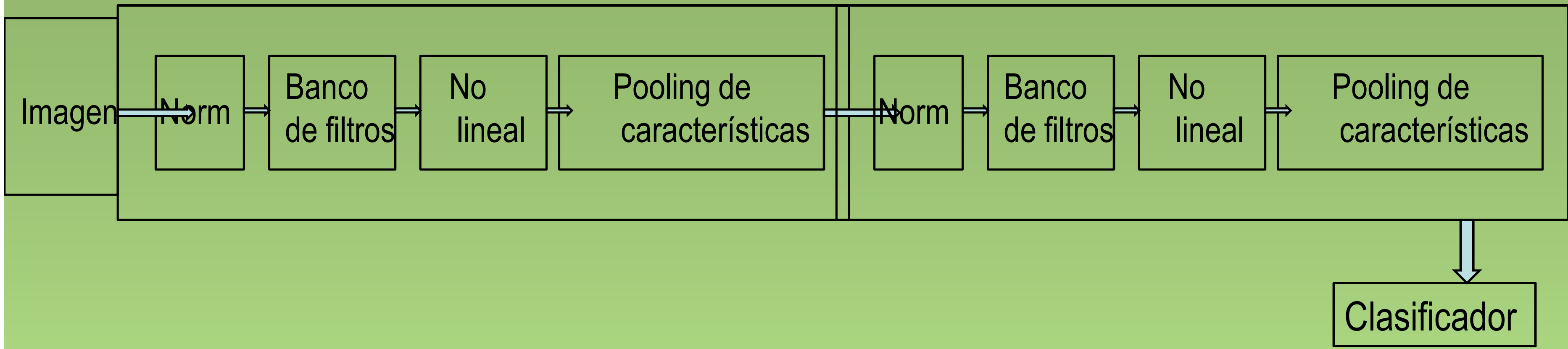


Mapa de características después de una normalización de contraste

Situar todas las capas juntas



Transformación de características:
Normalización, Banco de filtros, No linearidad, *pooling*



Banco de Filtros → **No linearidad= No linearidad embebida en alta dimensión**

Pooling de características= contracción, reducción de dimensionalidad, alisado

Aprendizaje del banco de filtros en cada etapa

Crear una jerarquía de características

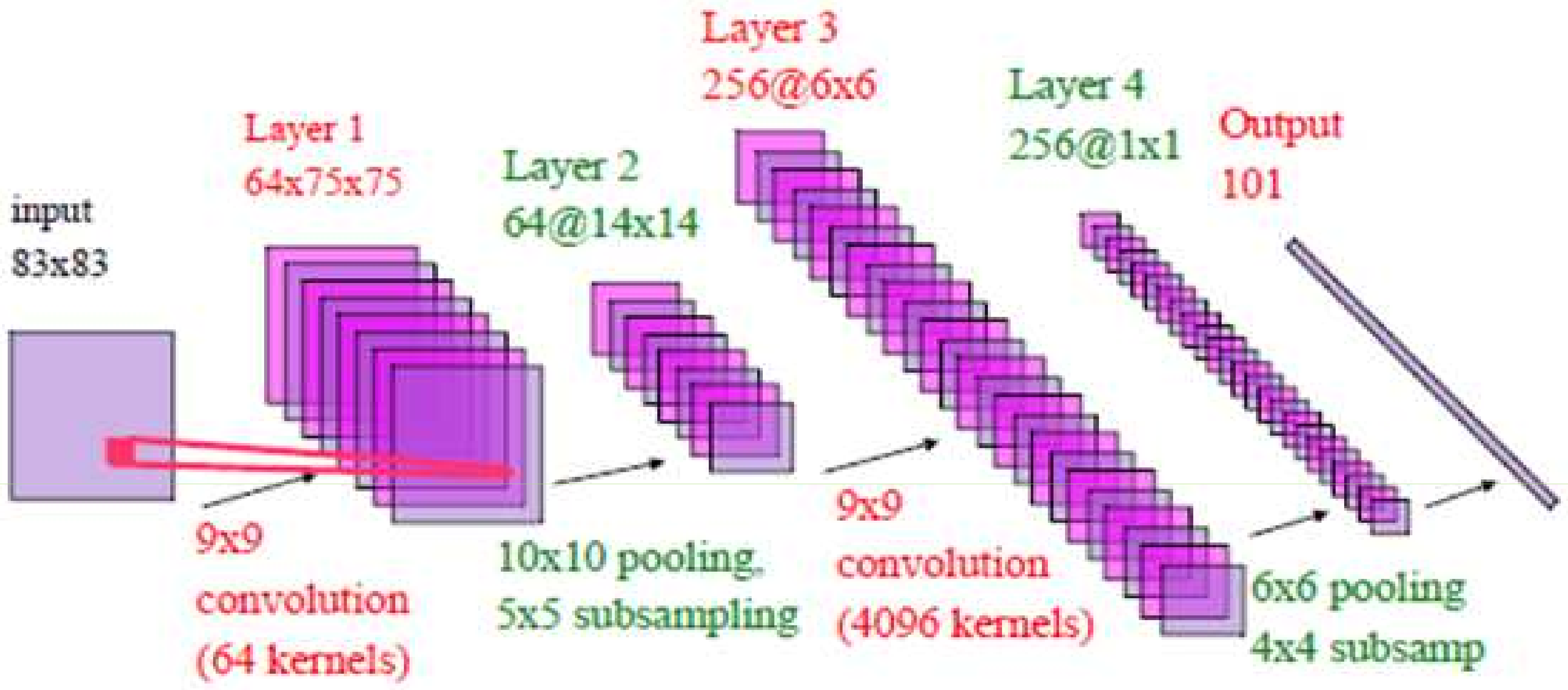
Los elementos básicos están inspirados en modelos de la corteza visual (y auditiv

Hubel and Wiesel 1962

Fufushima 1974-1982, LeCun 1988 y sucesivos

Desde el 2000, Hinton, Seung, Poggio, Ng, etc

Ejemplo: CovNet



Tipos de No-Linearidad: Media onda, función de contracción, sigmoide

Tipos de pooling: Media, L1, L2, Max

Tipos de entrenamiento: Supervisado (1988-2006), No supervisado +Supervisado (2006, hasta ahora)

Retropropagación del error

Funciones de perdida para clasificación multiclase

Función de decisión de tipo softmax

$$p\left(y^{(l)}=1 \middle| \mathbf{x},\boldsymbol{\theta}\right)=\frac{\exp f_l\left(\mathbf{x},\boldsymbol{\theta}_l\right)}{\sum_{j=1}^{J}\exp f_j\left(\mathbf{x},\boldsymbol{\theta}_j\right)},\ l=1,2,\ldots,J$$

la regla de clasificación es $C(x)=\hat{l}$, siendo $\hat{l}=\arg\max_l\ f_l(\mathbf{x},\hat{\boldsymbol{\theta}}_l)$ para j=1,...,J

Para regresión

Error Cuadrático Medio, $\text{MSE}=\frac{1}{n}\sum_{i=1}^{n}(t_i-y_i)^2$

Cambio de pesos

Para cada peso

$$\omega_i=\omega_i-\eta\frac{\partial E}{\partial w_i}+\alpha\omega_i-\lambda\eta\omega_i$$

donde η es el parámetro de aprendizaje

α es el parámetro de momento

λ es el parámetro de reducción del valor del peso

Retropropagación

Capa convolucional

Con la función de error E, y el filtro de salida x^t

$$\frac{\partial E}{\partial w_{ab}} = \sum_{i=0}^{N-m}\sum_{j=0}^{N-m}\frac{\partial E}{\partial x_{ij}^t}\frac{\partial x_{ij}^t}{\partial w_{ab}} = \sum_{i=0}^{N-m}\sum_{j=0}^{N-m}\frac{\partial E}{\partial x_{ij}^t} y_{(i+a)(j+b)}^{t-1}$$

$$\frac{\partial E}{\partial x_{ij}^t} = \frac{\partial E}{\partial y_{ij}^t}\frac{\partial y_{ij}^t}{\partial x_{ij}^t} = \frac{\partial E}{\partial y_{ij}^t}\frac{\partial}{\partial x_{ij}^t}\left(\sigma(x_{ij}^t)\right) = \frac{\partial E}{\partial y_{ij}^t}\sigma'(x_{ij}^t)$$

$$\frac{\partial E}{\partial y_{ij}^{t-1}} = \sum_{a=0}^{m-1}\sum_{b=0}^{m-1}\frac{\partial E}{\partial x_{(i-a)(j-b)}^t}\frac{\partial x_{(i-a)(j-b)}^t}{\partial y_{ij}^{t-1}} = \sum_{a=0}^{m-1}\sum_{b=0}^{m-1}\frac{\partial E}{\partial x_{(i-a)(j-b)}^t} w_{ab}$$

Entonces, el error es propagado a la capa previa

Capa de Reducción o *pooling*

En realidad se aprende en sí mismo, solo reducen el tamaño del problema al introducir la dispersión.

Reduce la región de tamaño kxk a un valor único durante la propagación hacia adelante.

El error se propagó al lugar de donde vino, por lo que los errores son bastante dispersos

¿Que es Theano?

Theano es un compilador de expresiones matemáticas basado en Python cuya sintaxis es bastante similar a NumPy.

Es un proyecto de código abierto desarrollado y mantenido por el grupo Ml de la Universidad de Montréal.

Utiliza expresiones matemáticas compuestas en una descripción de alto nivel imitando la sintaxis de NumPy y su semántica permitiendo a Theano proporcionar diferenciación simbólica

Características clave

Implementación sencilla compatible con CPU y GPU

Theano tiene sus propios optimizadores utilizando CUDA C++ para GPU

Fácil de implementar el algoritmo de retropropagación en CNN, de forma tal que se calculan automáticamente todas las transformaciones que conlleva el procedimiento.

Crea un grafo con las diferentes entradas involucradas.

La diferenciación se realiza utilizando la regla de la cadena

2017/11/15: Release of Theano 1.0.0. Everybody is encouraged to update.

Implementaciones basadas en Theano para Deep Learning

Caffe
Torch
Keras

Otros marcos de trabajo

cuDNN
DIGITS

Características claves de Caffe

Marco de trabajo (esencialmente para entrenamiento de CNN) desarrollado por Berkeley Vision Learning Center (BVLC)

Velocidad: Capaz de procesar sobre 60M imágenes por día con una sola tarjeta Nvidia K40 GPU, esta considerada la más rápida implementación para CNN disponible

Arquitectura expresiva: Permite que los modelos y la optimización se definan como ficheros de configuración más que código rígido, con habilidad para intercambiar entre CPU y GPU mediante una sencilla bandera.

Caffe: Capa de convolución

```
Layers {
        name: “conv1”
        type:       CONVOLUTION
        bottom: “data”
        top: “conv1”
        blobs_lr: 1
        blobs_lr: 2
        weight_decay: 1
        weight_decay: 0
        convolution_ param {
                num_output: 96
                kernel_size:7
                stride: 4
                weight_filter {
                        type: “ gaussian”
                        std: 0.01
                }
                bias_filter {
                        type: “constant”
                        value: 0
                }
        }
}
```

Caffe: Capa de reducción máxima

```
Layers {
        name: “pool1”
        type:     POOLING
        bottom: “conv1”
        top: “pool1”
        pooling_param {
                pool: MAX
                kernel_size: 3
                stride: 2
        }
}
```

Caffe: Resolución

Scrip

net: “trainer.prototxt”

test_iter: 1000

test_interval: 1000

base_ lr: 0.001

lr_policy: “step”

gamma: 0.1

stepsize: 10000

display: 20

max_iter: 50000

momentum: 0.9

weight_decay: 0.0005

snapshot: 1000

snapshot_prefix: “snaps/age_train”

solver_mode: GPU

Alguna tareas para las cuales las redes convolucionales profundas son las mejores

- Traffic sign recognition (2011) GTSRB competition (IDSIA, NYU)
- Pedestrian Detection (2013): INRIA datasets and others (NYU)
- Human Action Recognition (2011) Hollywood II dataset (Stanford)
- Object Recognition (2012) ImageNet competition
- Scene Parsing (2012) Standford bgd, SiftFlow, Barcelona (NYU)
- Scene parsing from depth images (2013) NYU TGB-D dataset (NYU))
- Sèech Recognition (2012) Acoustic modeling (IBM and Google)
- Breast cancer cell mitosis detection (2011) MITOS (IDSIA)
- Age and Gender clasification using Convolutional Neural Networks 2015
- Ordinal Regression with Multiple Output CNN for Age Estimation. 2016
- Fast Convolutional Neural Network Training Using Selective Data Sampling: Application to Hemorrhage Detection in Color Fundus Images. 2016

MODELOS COMPUTACIONALES: CUARTO CURSO DEL GRADO DE ING. INFORMÁTICA EN COMPUTACION

REDES NEURONALES CONVOLUCIONALES

César Hervás-Martínez/ Pedro A. Gutierrez
Grupo de Investigación AYRNA

Departamento de Informática y Análisis Numérico
Universidad de Córdoba
Campus de Rabanales. Edificio Einstein.
Email: chervas@uco.es

2018-2019