Aprendizaje automático

Redes neuronales (parte 2)

FACULTAD DE INGENIERÍA | UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Agenda

- Entrenando una DNN
  - Desvanecimiento y explosión de gradiente (vanishing and exploding gradient)
  - Aprendizaje por transferencia (transfer learning)
  - Optimizadores
  - Regularización
- Redes Neuronales Convolucionales
  - Capa de convolución
  - Filtros de convolución
  - Capa de *pooling*
  - Ejemplo de CNN

Entrenando DNN

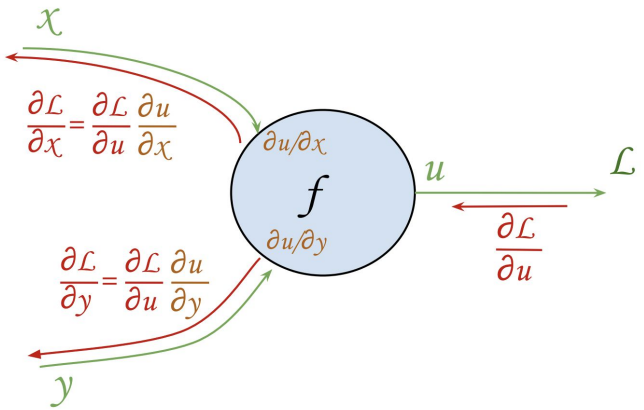
Backpropagation (BP)

$e(w)$: función error de la red

$$\nabla e(w): \frac{\partial e(w)}{\partial w_{ij}^{(\ell)}} \quad \forall i,j,\ell$$

$$w_{ij}^{(\ell)} = w_{ij}^{(\ell)} - \eta \frac{\partial e(w)}{\partial w_{ij}^{(\ell)}} \quad \forall i,j,\ell$$

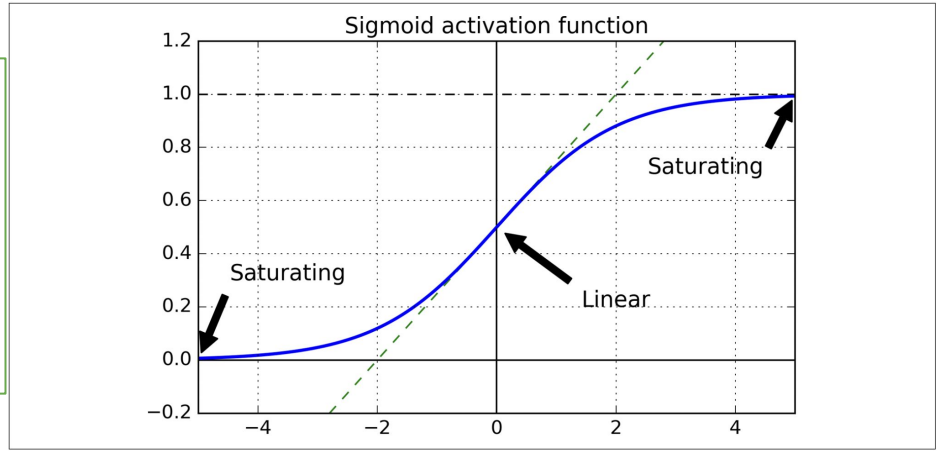
Gradiente hacia abajo: Regla de la cadena



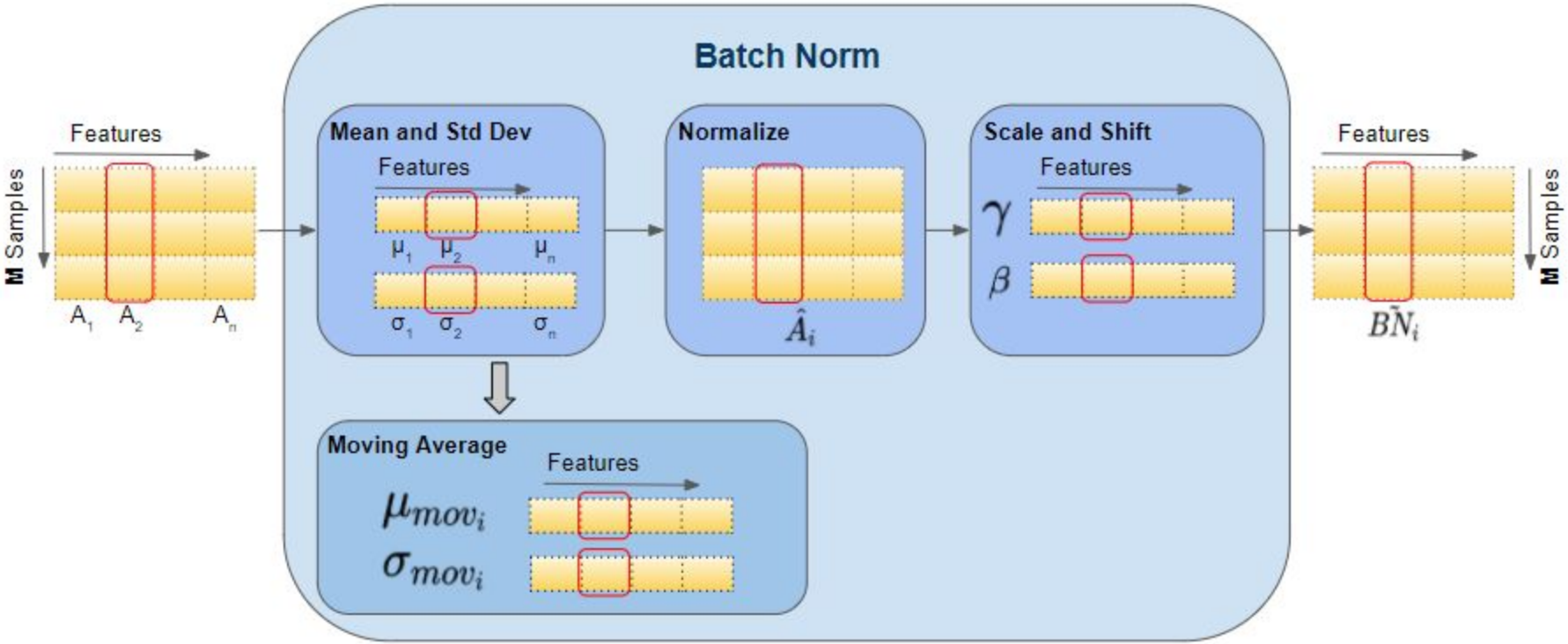
Desvanecimiento y explosión de gradiente

- BP propaga el gradiente del error de la red de atrás hacia adelante.
- BP usa el gradiente para actualizar cada parámetro de la red con un paso de GD
- Ocurre seguido
  - Gradiente decrece a medida que BP progresa → GD no actualiza las primeras capas de la red → entrenamiento no converge a una buena solución → vanishing gradient
  - Gradiente crece y crece → muy grande en primeras capas de la red → BP diverge → exploding gradient

- Inicializaciones de Glorot, He o Lecun
- Funciones de activación que no saturen → ReLU → Leaky ReLU → ELU
- Batch normalization
- Gradient clipping

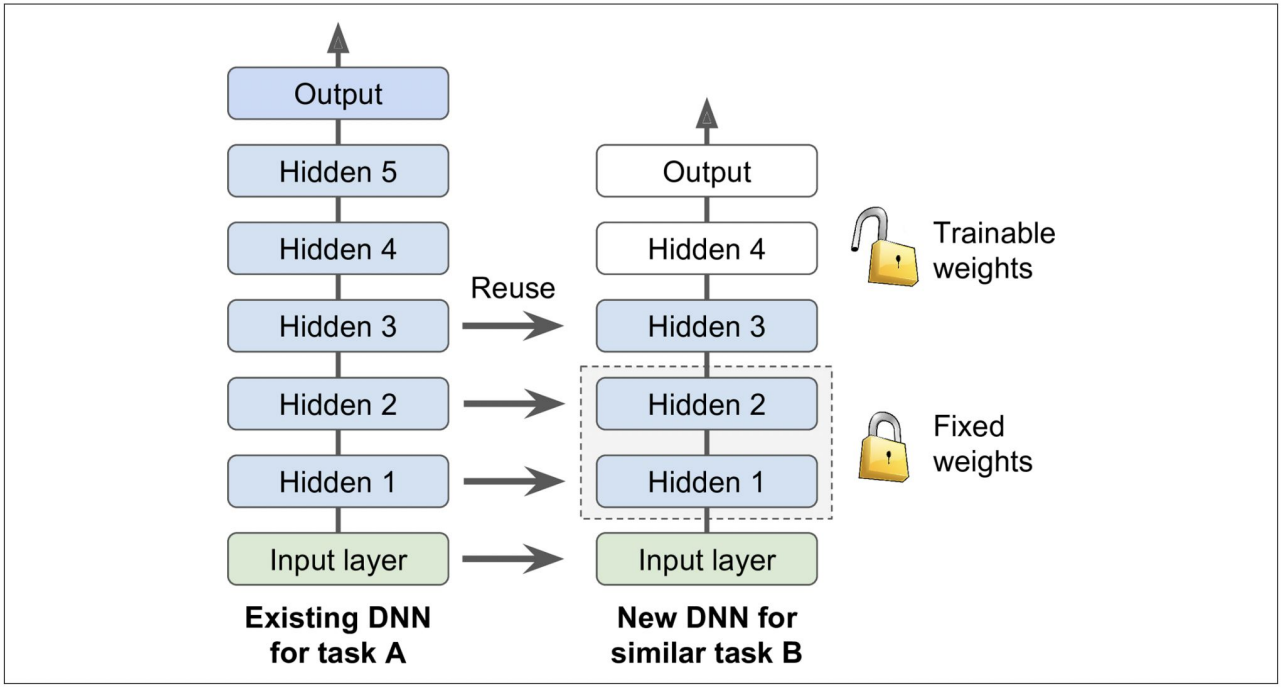


Batch normalization



Aprendizaje por transferencia

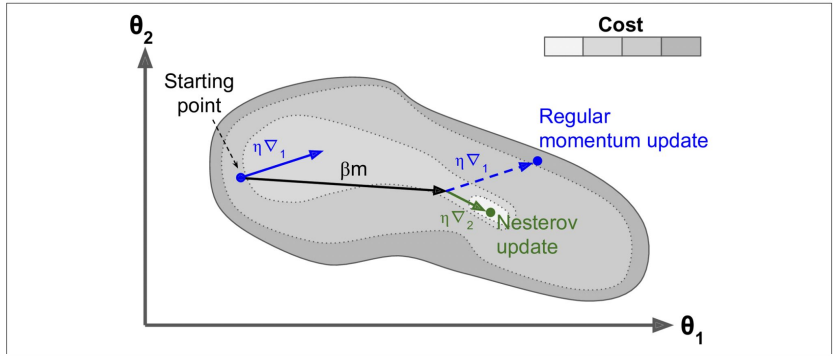
- En vez de entrenar de zero una DNN, re usar una DNN ya entrenada para una tarea similar y re usar algunas de sus capas
- Acelera la fase entrenamiento y requiere menos datos

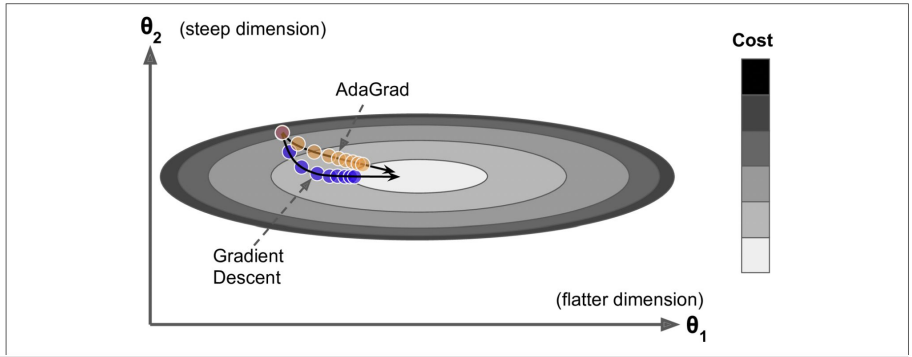


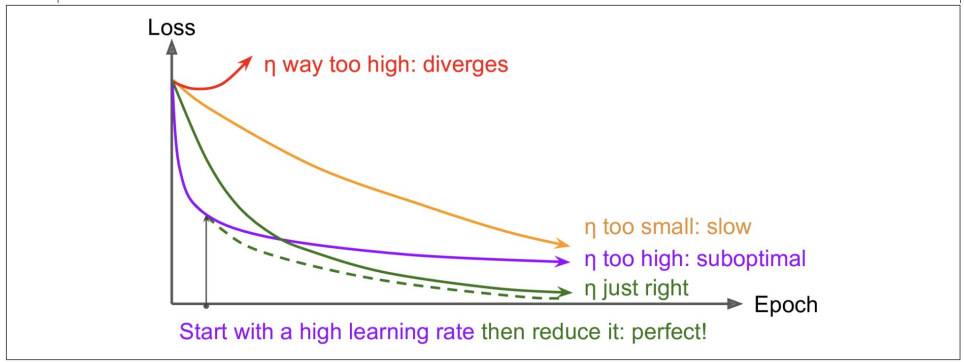
Optimizadores

- Optimizadores más rápidos que el GD regular
  - Momentum
  - Nesterov Accelerated Gradient
  - AdaGrad
  - RMSProp
  - Adam, AdaMax, Nadam

- *Learning rate scheduling*
  - *Power scheduling*
  - *Exponential scheduling*
  - *Piecewise constant*
  - *Performance scheduling*
  - *1cycle scheduling*

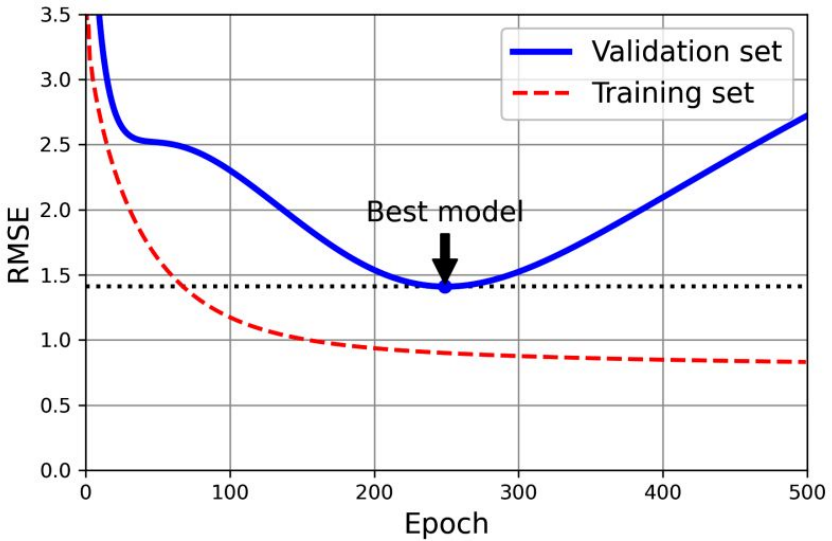


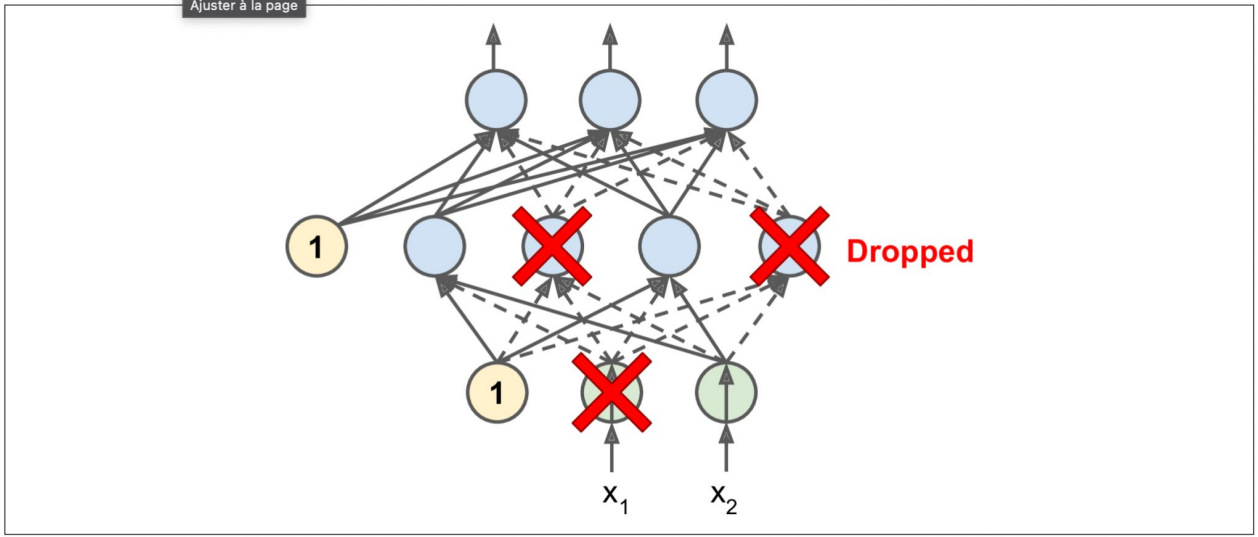




Regularización

- Modelos muy expresivos (~ entre 10^4 y 10^6 parámetros)
- Evitar sobreajuste a datos de entrenamiento
- *Early stopping*
- *Batch normalization*
- Regularización $\ell 1$ y $\ell 2$
- Max-Norm
- Dropout





Resumen

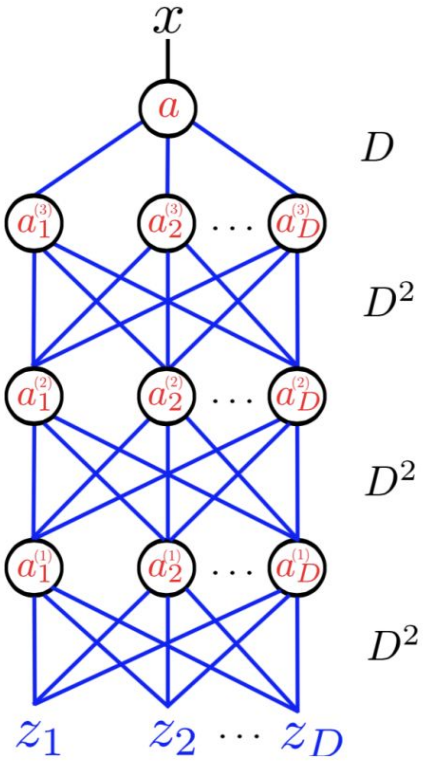
Hyperparameter	Default value
Kernel initializer	He initialization
Activation function	ELU
Normalization	None if shallow; Batch Norm if deep
Regularization	Early stopping (+ℓ_2 reg. if needed)
Optimizer	Momentum optimization (or RMSProp or Nadam)
Learning rate schedule	1cycle

Redes Neuronales Convolucionales (CNN)

FACULTAD DE INGENIERÍA
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

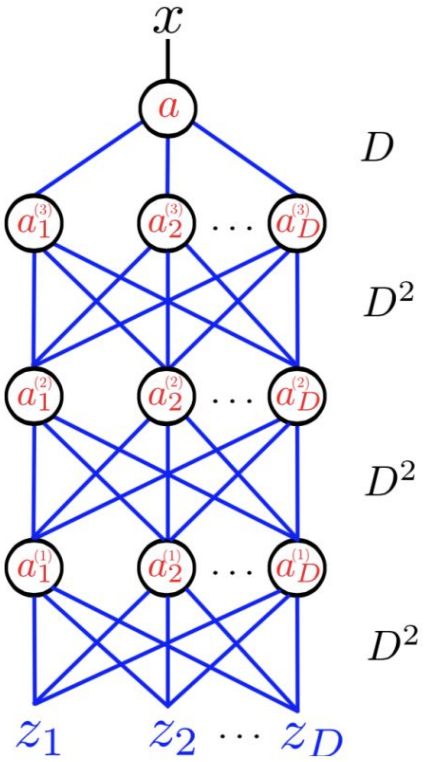
¿Cual es el problema de la redes *fully connected*?

- ¿Cuántos parámetros tiene esta red?



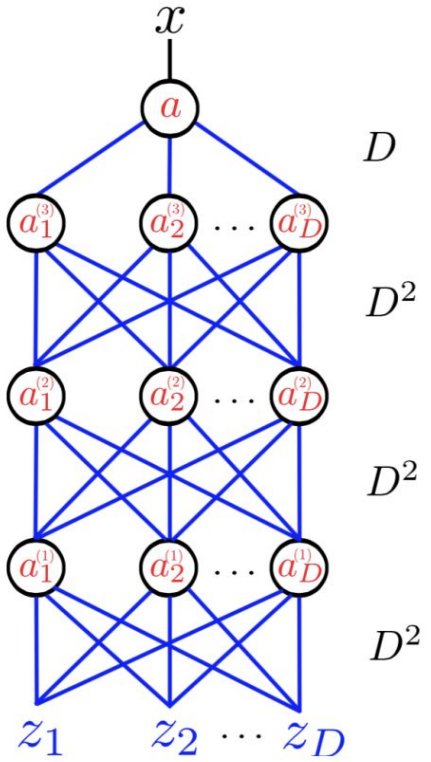
¿Cual es el problema de la redes *fully connected*?

- ¿Cuántos parámetros tiene esta red?
  - $3D^2 + D$



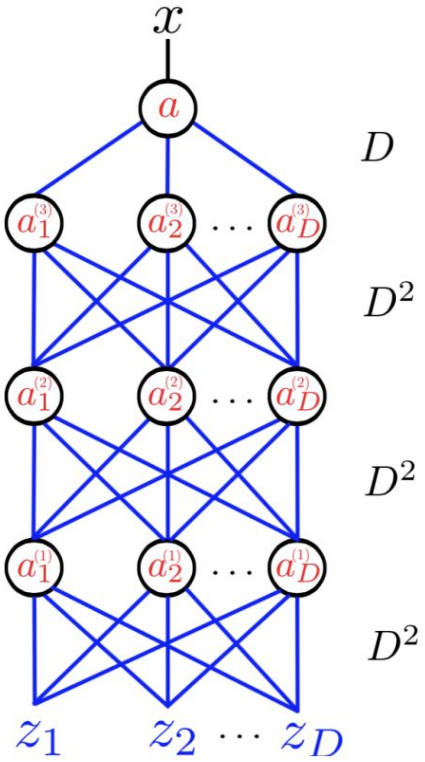
¿Cual es el problema de la redes *fully connected*?

- ¿Cuántos parámetros tiene esta red?
  - $3D^2 + D$
- Si tenemos una imagen pequeña 32x32
  - $3 \times (32^2)^2 + 32^2 \approx 3 \times 10^6$



¿Cual es el problema de la redes *fully connected*?

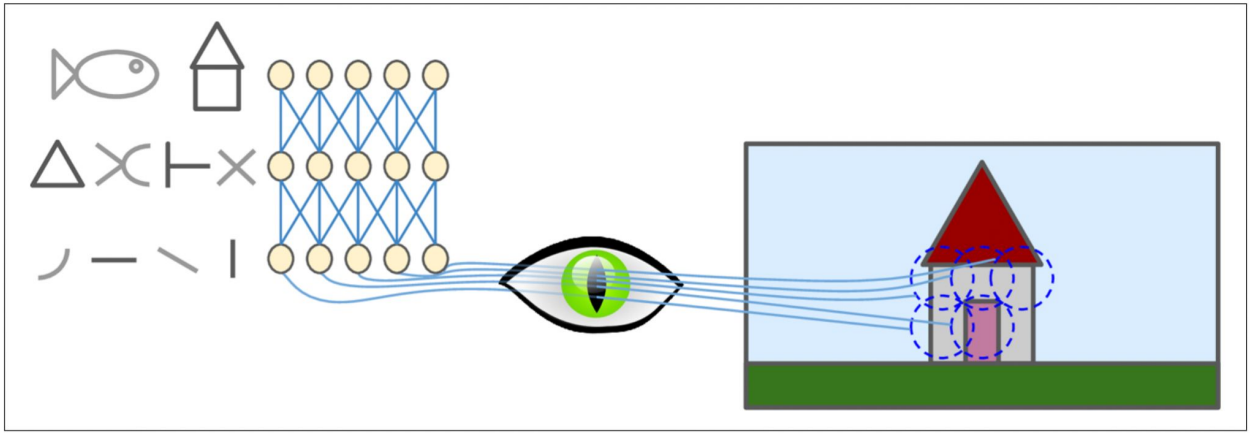
- ¿Cuántos parámetros tiene esta red?
  - $3D^2 + D$
- Si tenemos una imagen pequeña 32x32
  - $3 \times (32^2)^2 + 32^2 \approx 3 \times 10^6$



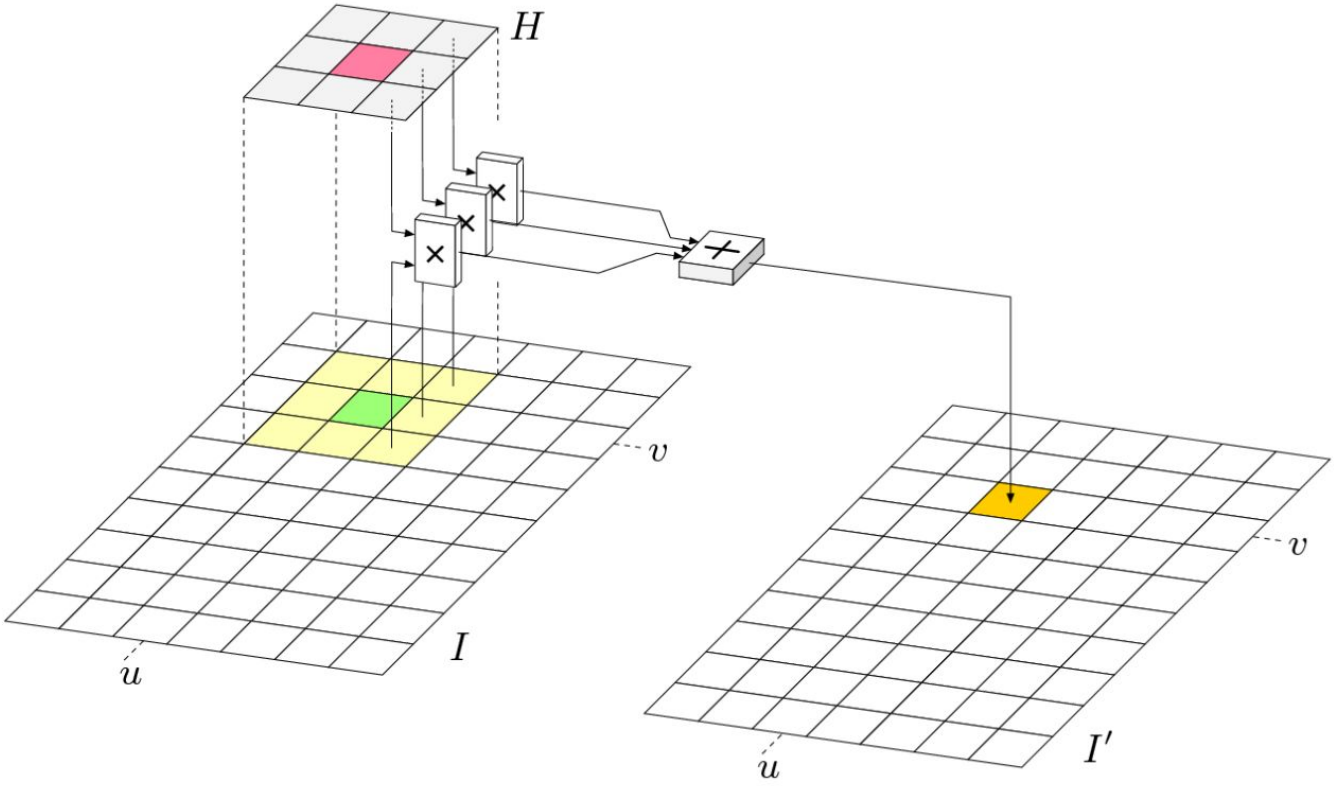
- **Difícil de entrenar**: sobreajuste, inicialización
- **Redes de Convolución:** permiten disminuir número de parámetros forzando invarianzas

Arquitectura de la corteza visual

- Hubel y Wiesel 1961
  - Varias neuronas de la corteza visual tienen un pequeño campo receptivo local
  - Los campos receptivos pueden solaparse, proporcionando el campo de visión completo
  - Algunas neuronas sólo reaccionan ante líneas con cierta orientación
  - Algunas neuronas tienen campos receptivos más grandes y reaccionan a patrones más complejos
  - Esta arquitectura puede detectar todo tipo de patrones complejos en el campo visual
- Redes Neuronales Convolucionales se inspiran de estos experimentos
  - Capas convolucionales y capas de *pooling*

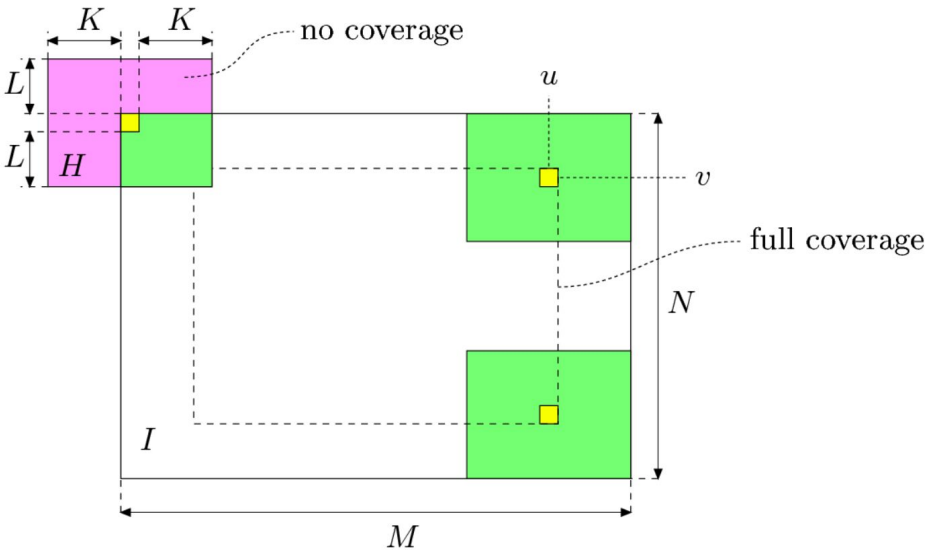


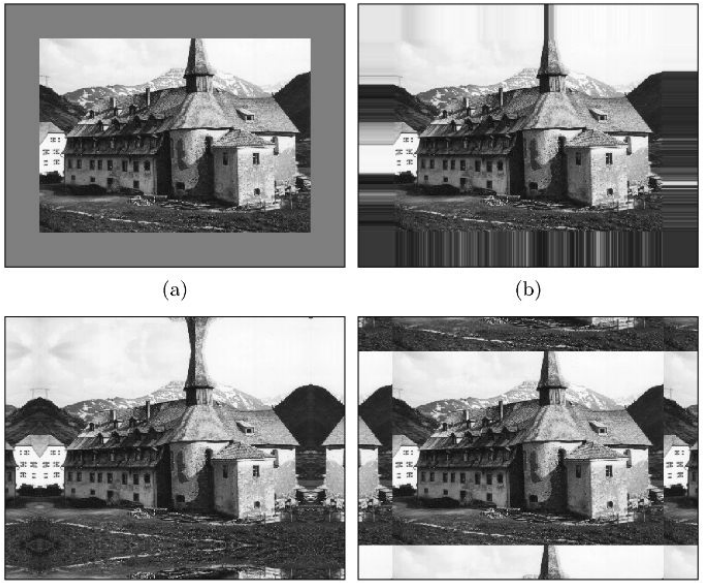
Convolución



Convolución

Padding



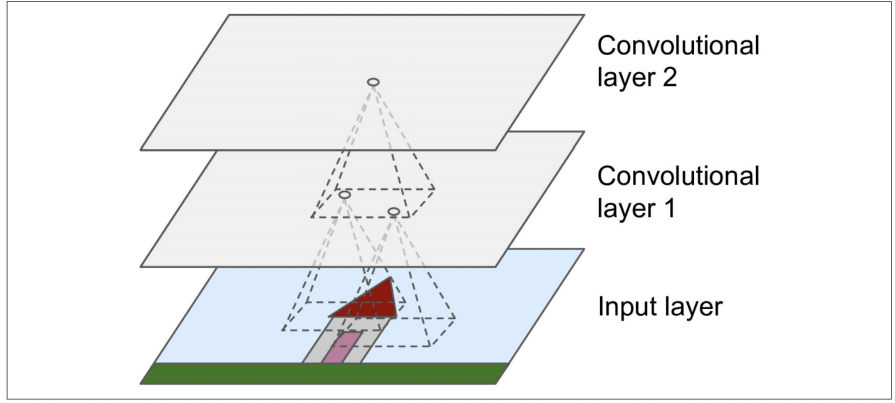


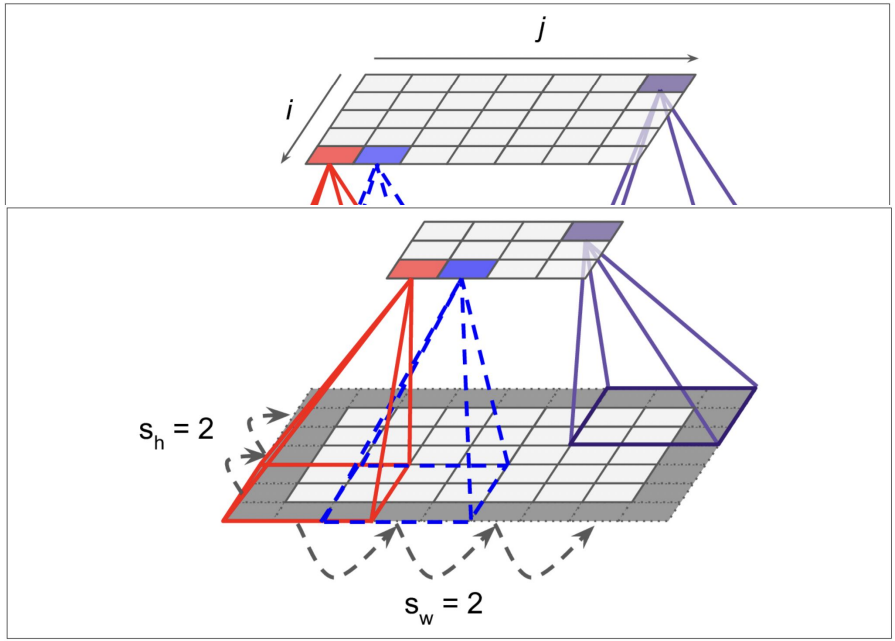
Capa de convolución

- Bloque esencial de las CNN
- Neuronas de una capa se conectan sólo a las neuronas de la capa anterior que están en el campo receptivo correspondiente

- Neurona localizada en la fila i y columna j de la capa l se conecta a las neuronas de la capa l-1 entre las filas i y $i + f_h - 1$ y entre las columnas j y $j + f_w - 1$
- El *zero padding* se usa para mantener el tamaño las capas

- Se pueden conectar capas más grandes a capas más chicas saltando de un campo receptivo a otro (*stride)*

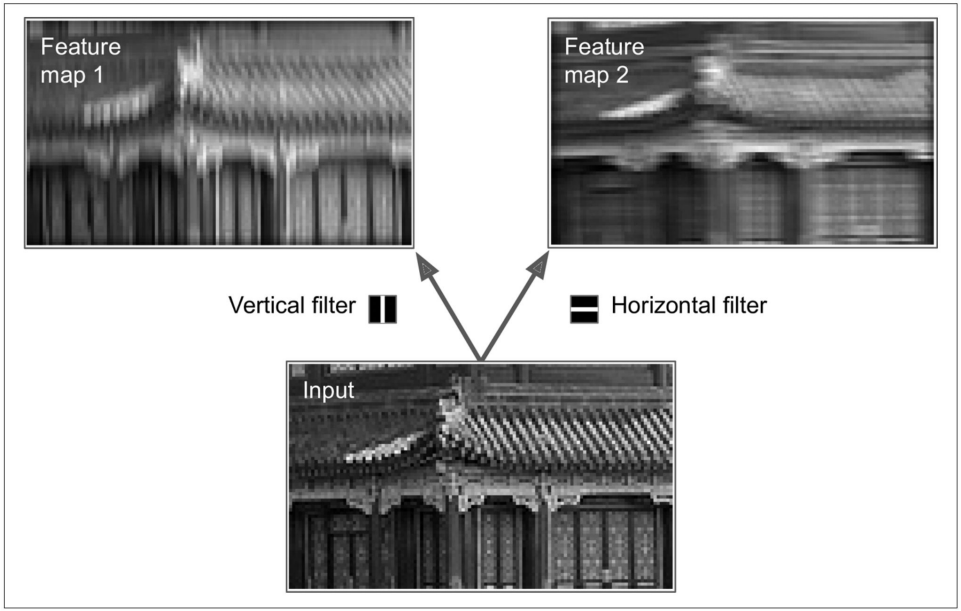




Filtros de convolución (*convolution kernel*)

- Los pesos entre neuronas se pueden representar como una pequeña imagen de tamaño igual al del campo receptivo
- Ejemplo con 2 filtros distintos

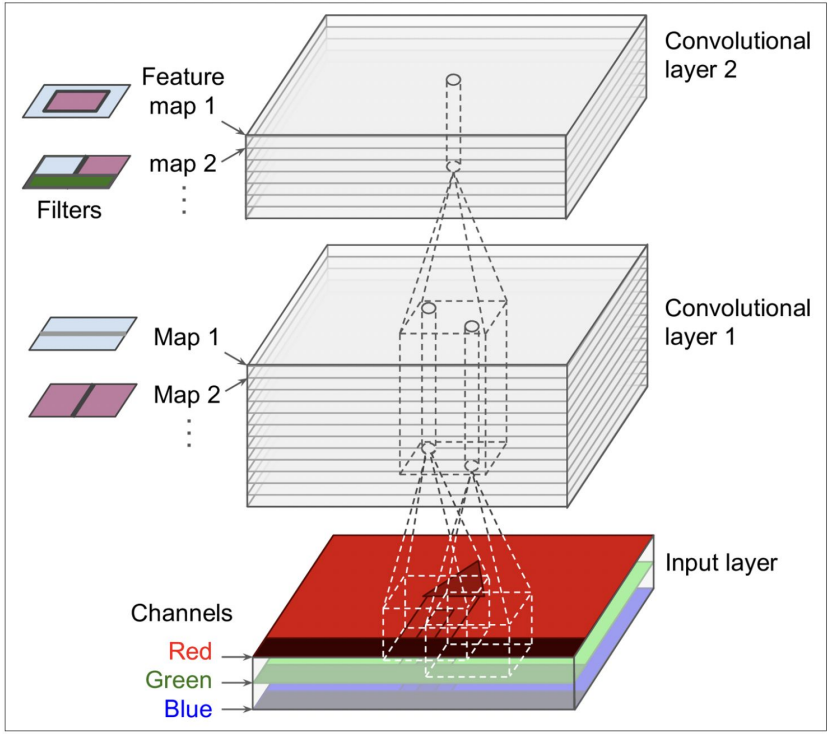
- Si todas las neuronas de una capa utilizan las mismas conexiones (el mismo filtro) con la capa anterior, entonces el resultado es la imagen filtrada con ese filtro
- Dicho resultado es el mapa de características (*feature map*) que sobresalta qué partes de la capa anterior activa los filtros
- Estos filtros es lo que aprende la CNN



Combinando varias capas de características

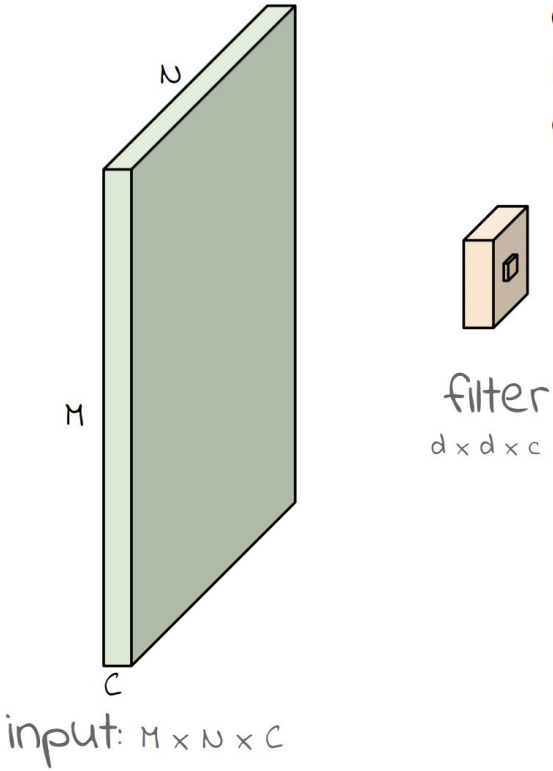
- Una capa de convolución tiene varios filtros
- Tiene tantos *feature maps* como filtros
- Hay una neurona por pixel de cada *feature map* y todas las neuronas de un *feature map* comparten los mismos pesos
- El campo receptivo de una neurona es tridimensional

- Cada neurona en la capa l y *feature map* k es el resultado de la convolución entre el k-ésimo filtro y el *tensor* comprendido dentro del campo receptivo de dicha neurona



$$z_{i,j,k} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f_{n'}-1} x_{i',j',k'} \cdot w_{u,v,k',k} \quad \text{with} \begin{cases} i' = i \times s_h + u \\ j' = j \times s_w + v \end{cases}$$

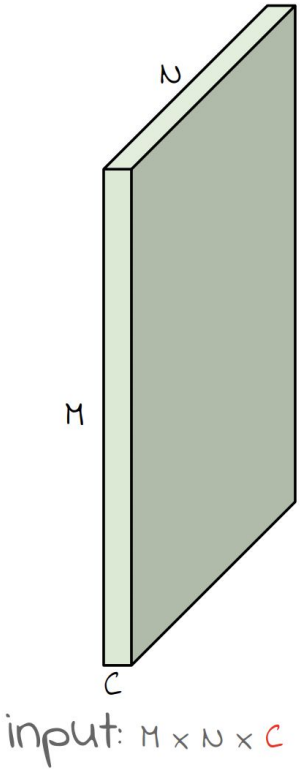
Capa de convolución

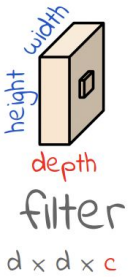


filter
d × d × c

Convolución de imagen de entrada (tensor) y filtro (productos internos en cada posición de la imagen)

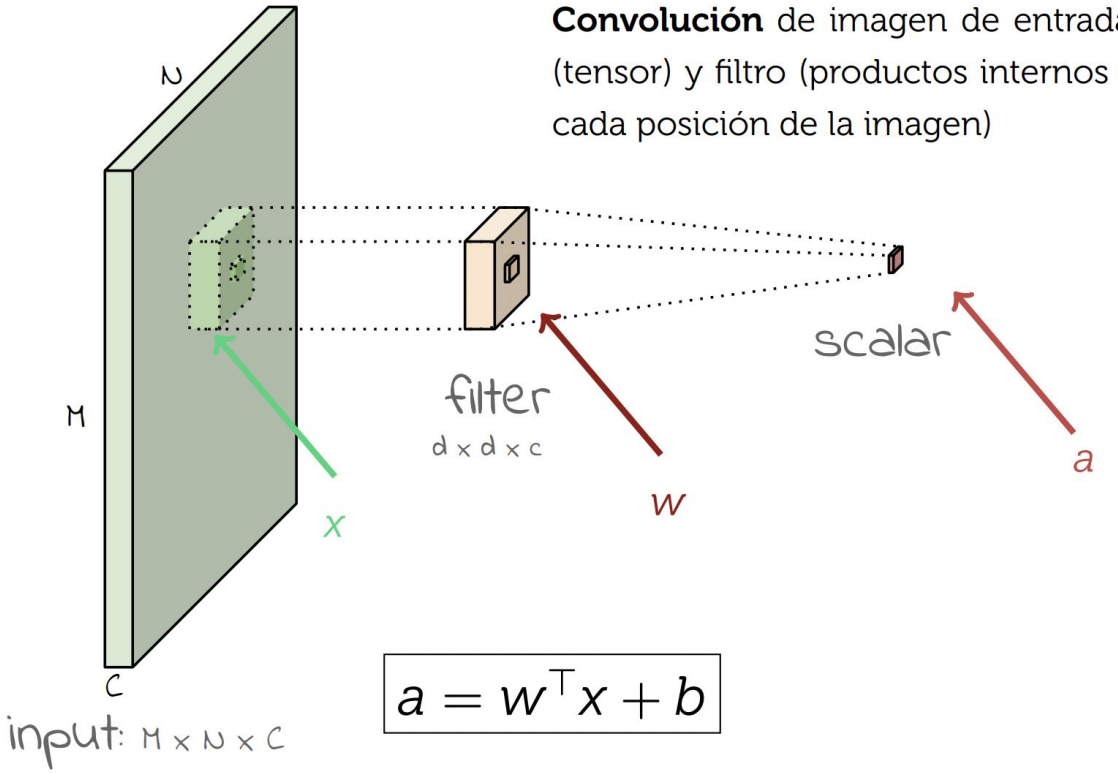
Capa de convolución



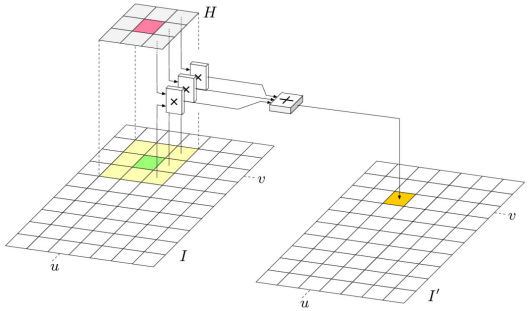


Convolución de imagen de entrada (tensor) y filtro (productos internos en cada posición de la imagen)

Capa de convolución

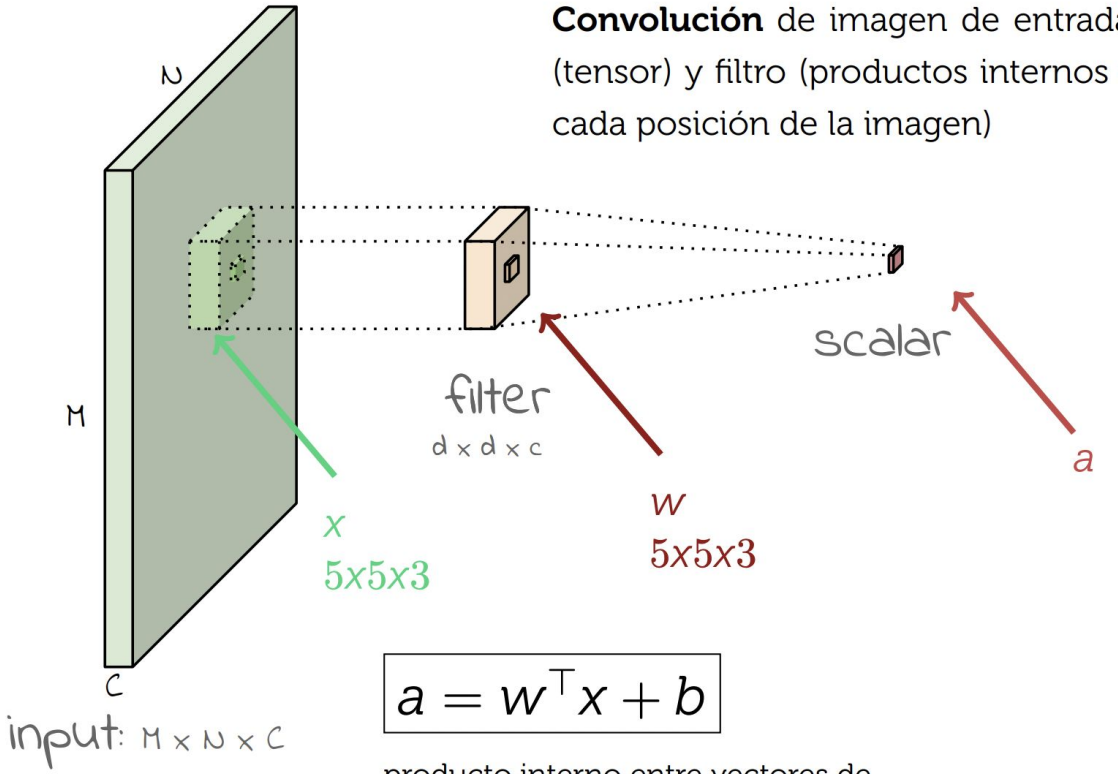


Convolución de imagen de entrada (tensor) y filtro (productos internos en cada posición de la imagen)



$$a = w^\top x + b$$

Capa de convolución

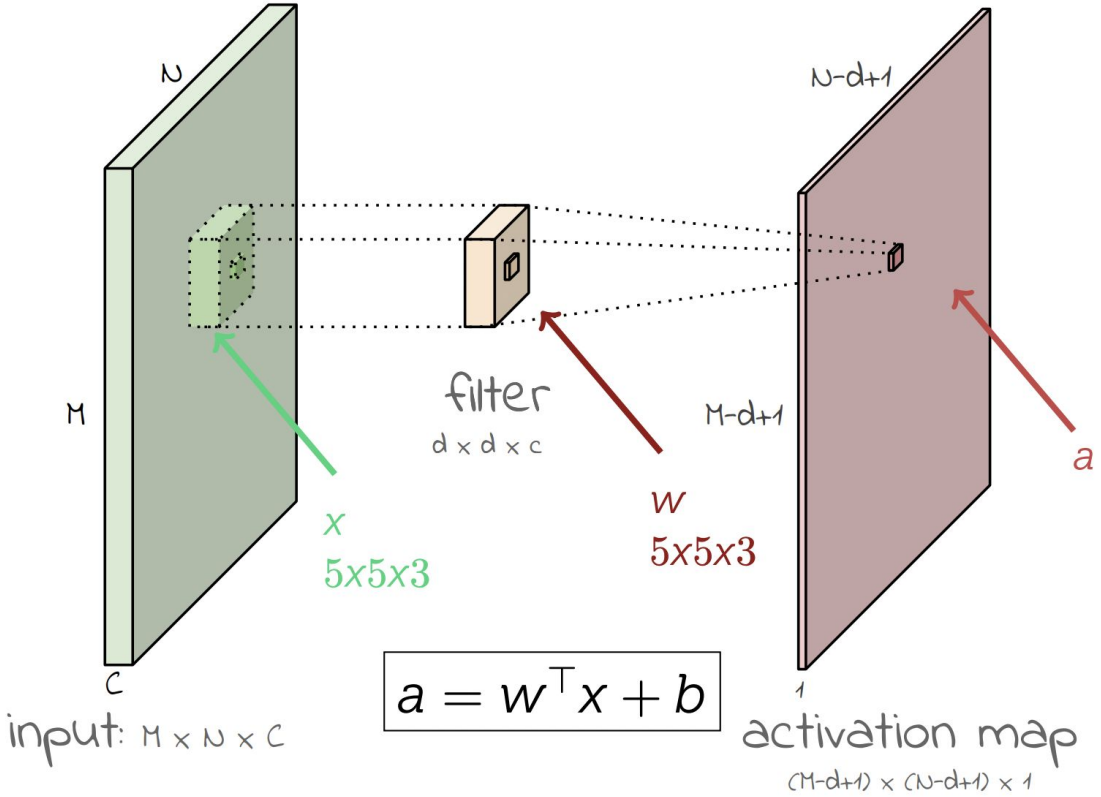


Convolución de imagen de entrada (tensor) y filtro (productos internos en cada posición de la imagen)

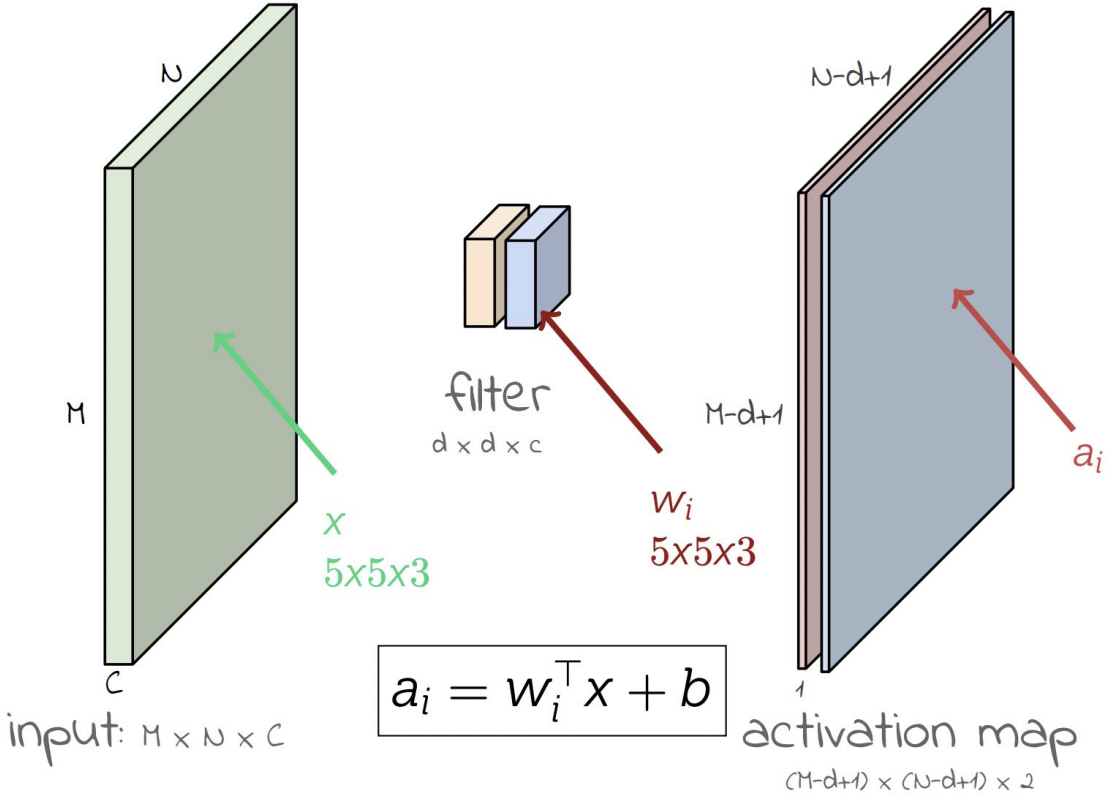
$$a = w^\top x + b$$

producto interno entre vectores de dimensión 75 $(5 \times 5 \times 3)$+ bias $\rightarrow$ **escalar**

Capa de convolución

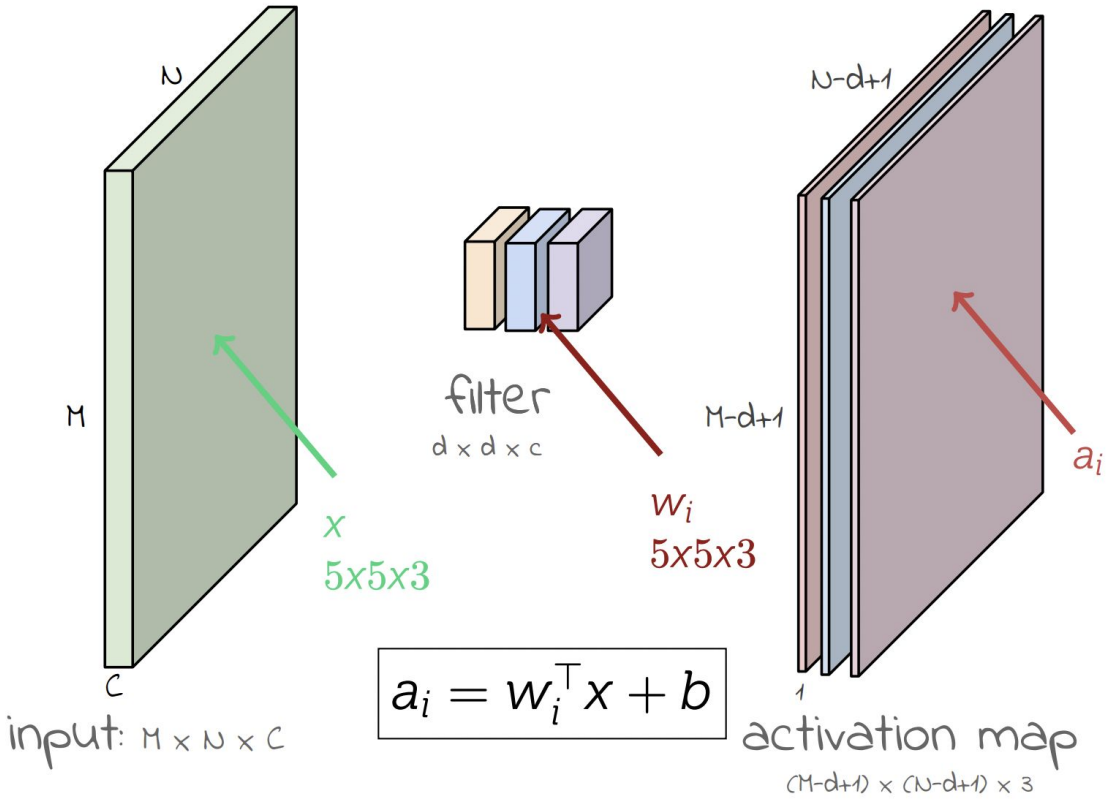


Capa de convolución



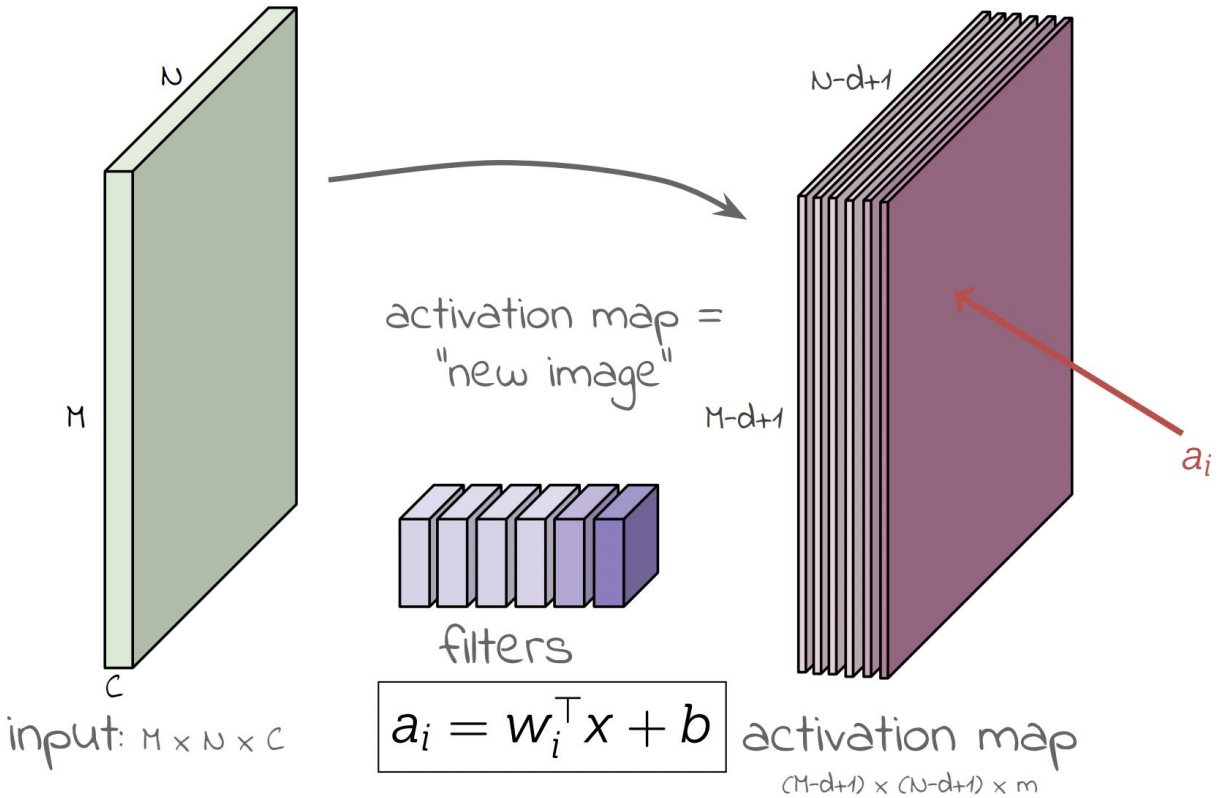
$$a_i = w_i^\top x + b$$

Capa de convolución

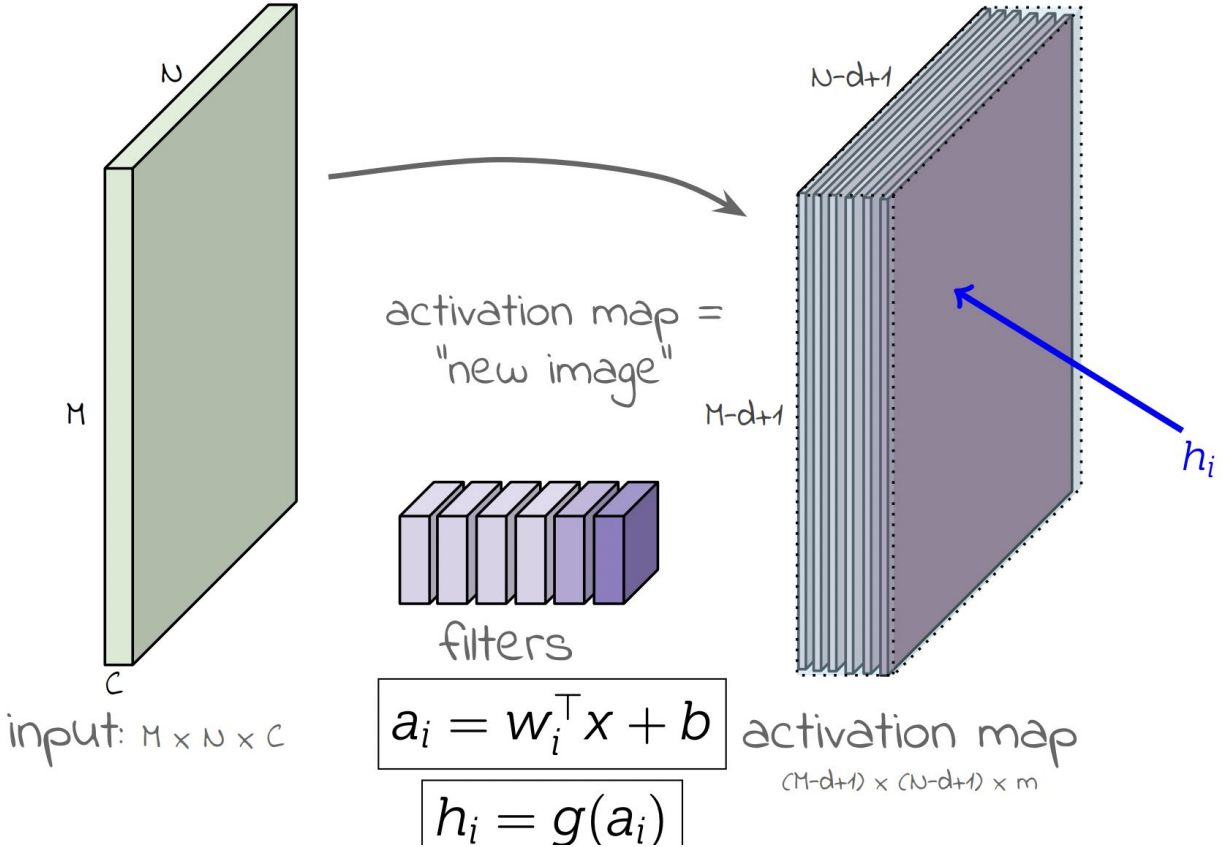


$$a_i = w_i^\top x + b$$

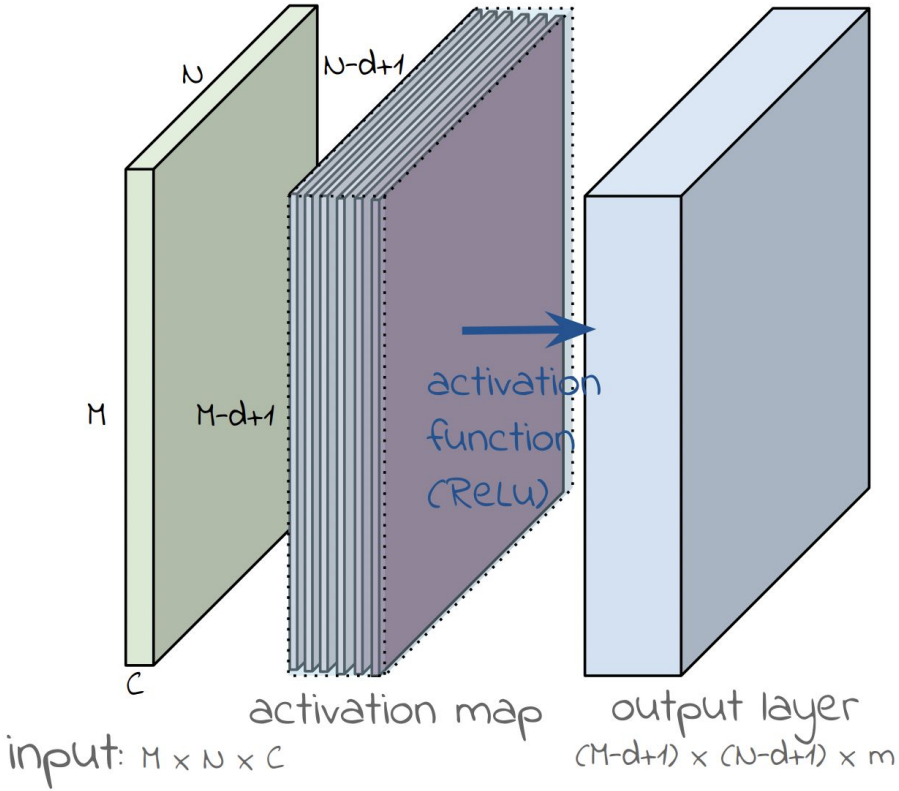
Capa de convolución



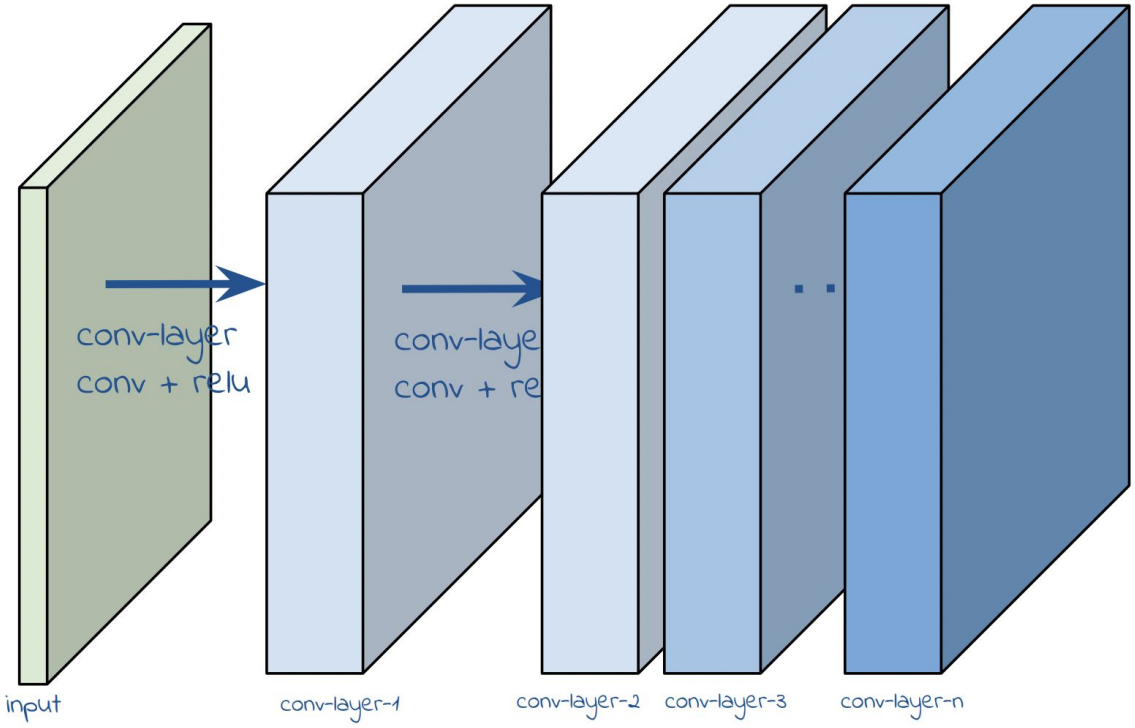
Capa de convolución: convolución



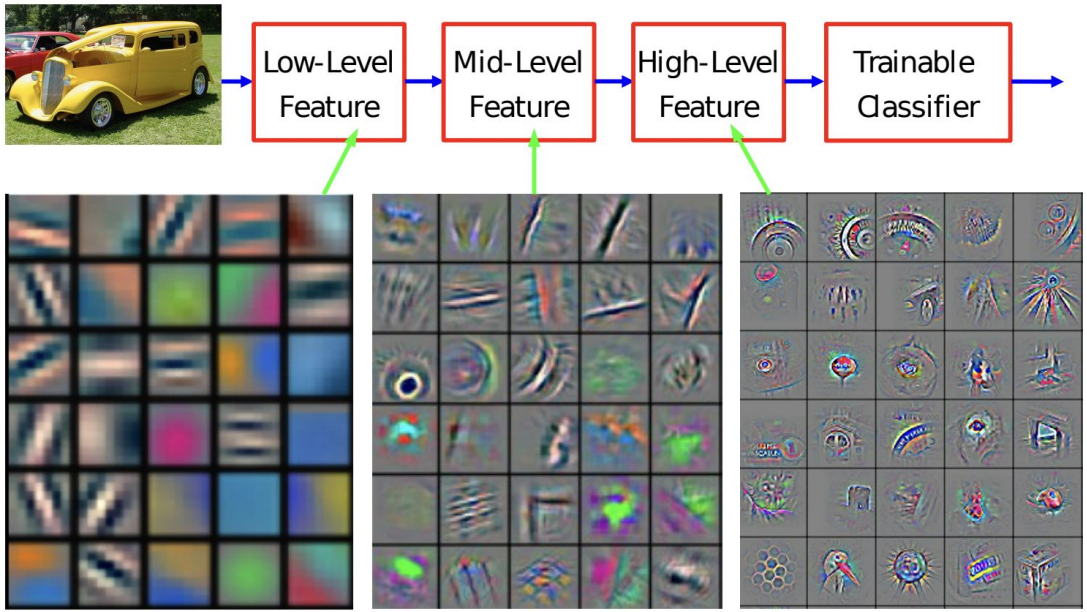
Capa de convolución: convolución + activación



Capa de convolución: convolución + activación



Visualización de los *feature maps*

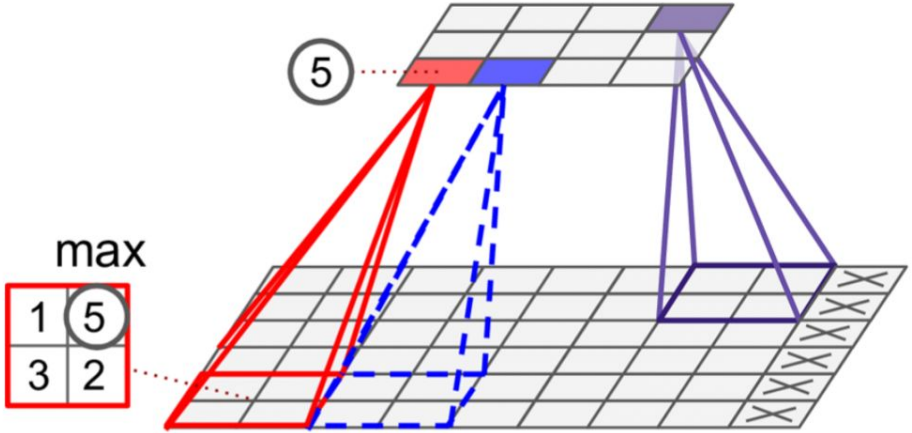


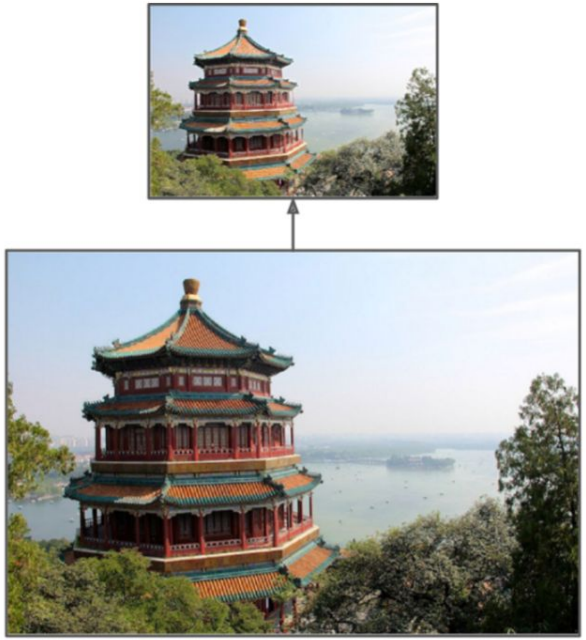
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Ejemplo tomado de charlas de Y. Lecun.

Capas de *pooling*

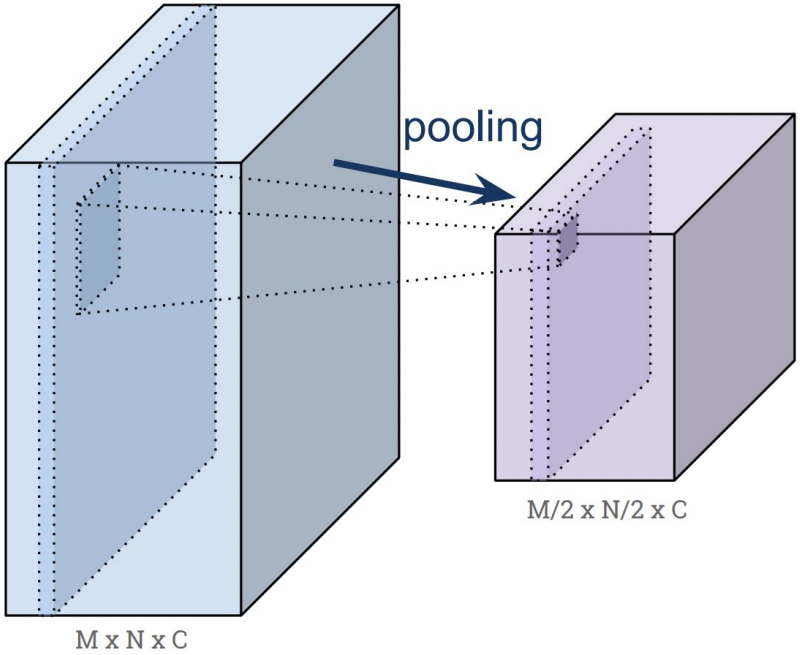
- Comprime (sub-muestreo) de la representación
- Opera en cada mapa de activación (canal) por separado





Capas de *pooling*

- Comprime (sub-muestreo) de la representación
- Opera en cada mapa de activación (canal) por separado



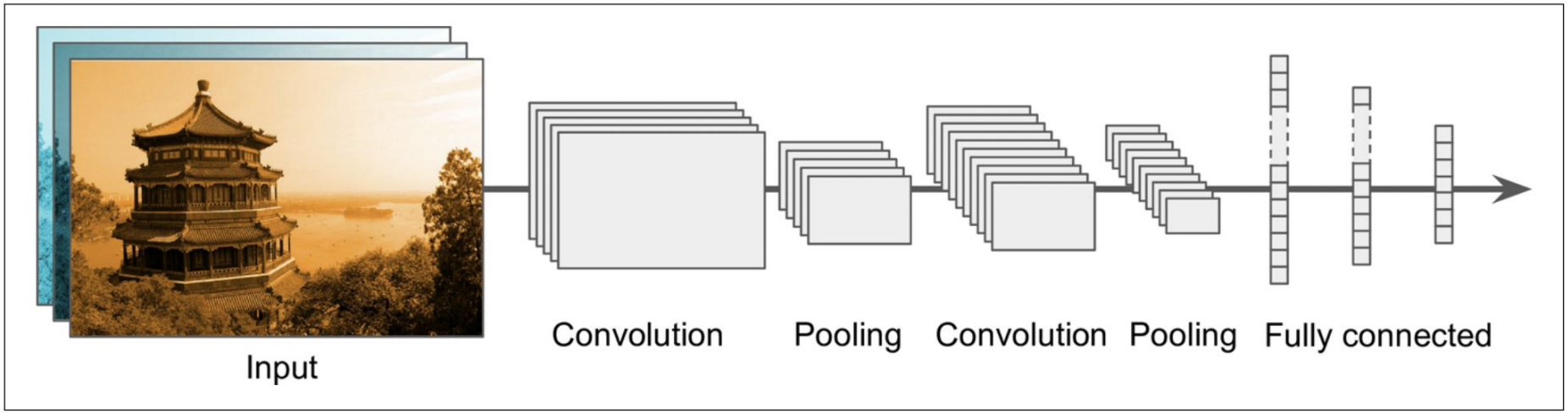
Capas de *pooling*

- Comprime (sub-muestreo) de la representación
- Opera en cada mapa de activación (canal) por separado

Capas de *pooling*

- Comprime (sub-muestreo) de la representación
- Opera en cada mapa de activación (canal) por separado

Ejemplo de CNN típica



CNN simple para el problema de clasificación de Fashion MNIST

```
model = keras.models.Sequential([
    keras.layers.Conv2D(64, 7, activation="relu", padding="same",
                        input_shape=[28, 28, 1]),
    keras.layers.MaxPooling2D(2),
    keras.layers.Conv2D(128, 3, activation="relu", padding="same"),
    keras.layers.Conv2D(128, 3, activation="relu", padding="same"),
    keras.layers.MaxPooling2D(2),
    keras.layers.Conv2D(256, 3, activation="relu", padding="same"),
    keras.layers.Conv2D(256, 3, activation="relu", padding="same"),
    keras.layers.MaxPooling2D(2),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(64, activation="relu"),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation="softmax")
])
```

LeNet-5

Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully connected	–	10	–	–	RBF
F6	Fully connected	–	84	–	–	tanh
C5	Convolution	120	1×1	5×5	1	tanh
S4	Avg pooling	16	5×5	2×2	2	tanh
C3	Convolution	16	10×10	5×5	1	tanh
S2	Avg pooling	6	14×14	2×2	2	tanh
C1	Convolution	6	28×28	5×5	1	tanh
In	Input	1	32×32	–	–	–

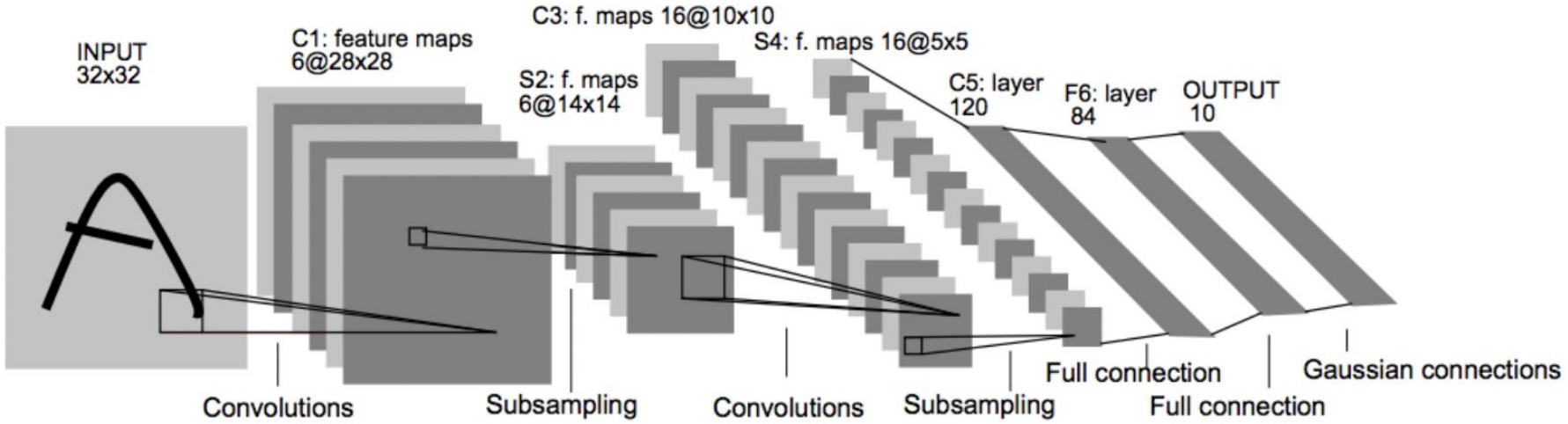
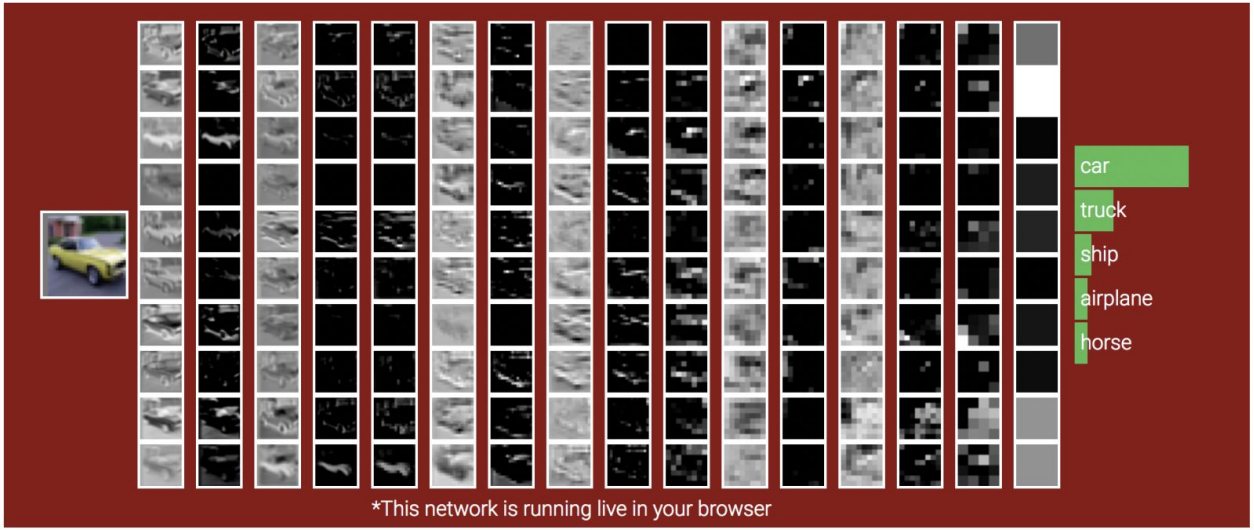


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

CNN



- (conv-relu-conv-relu-pool)x3-FC-softmax
- 17 capas 7000 parámetros, filtros 3×3, pooling 2×2
- ConvNetJS:
  `http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html`


FACULTAD DE
INGENIERÍA
UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY