Convolutional Neural Networks (CNN, ConvNets)

Computer Vision / ITBA

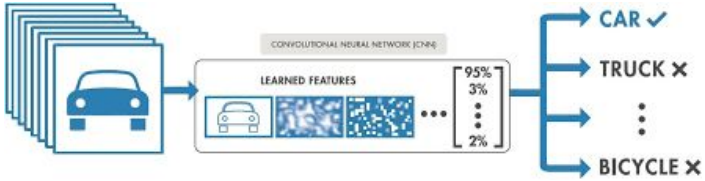
https://github.com/deeplearning-itba/cnn

Para qué se usan las CNN:

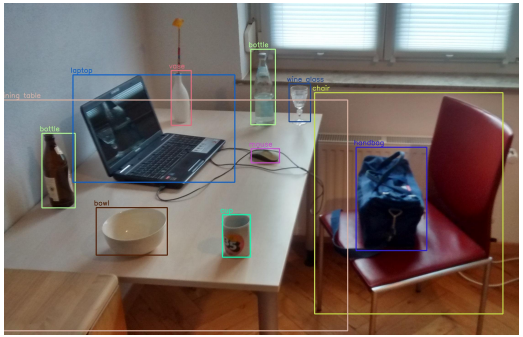
- **Image classification**

Binary: Cat? (0/1)

Multiclass



- **Object detection**



- **Face detection**
- **Face recognition**
- **Image segmentation**

Usos fuera de las imágenes:

- **NLP: text classification**
- **Audio classification**
- **Time series forecasting**
- **Game playing (extracción de features para Reinforcement Learning)**

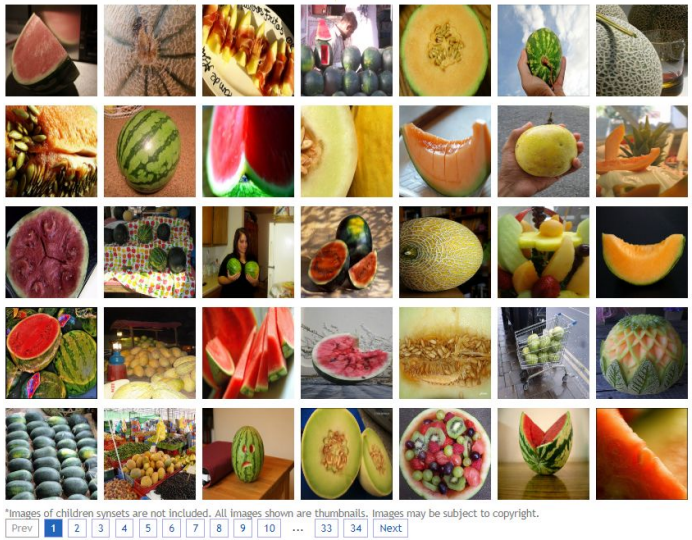
Problemas con las FCN

Ej: MNIST / Fashion MNIST

- Imágenes de 28x28x1 => Dimensión de entrada = 784
- 1 capa densa de 1000 => 784.784 parámetros

Ej: Imágenes más reales:

- Imágenes de 227x227x3 => Dim. de entrada: 154.587
- 1 capa densa de 1000 => +150 millones de parámetros



Problemas que surgen:

1. La cantidad de parámetros hace que sea computacionalmente caro/imposible de manejar.

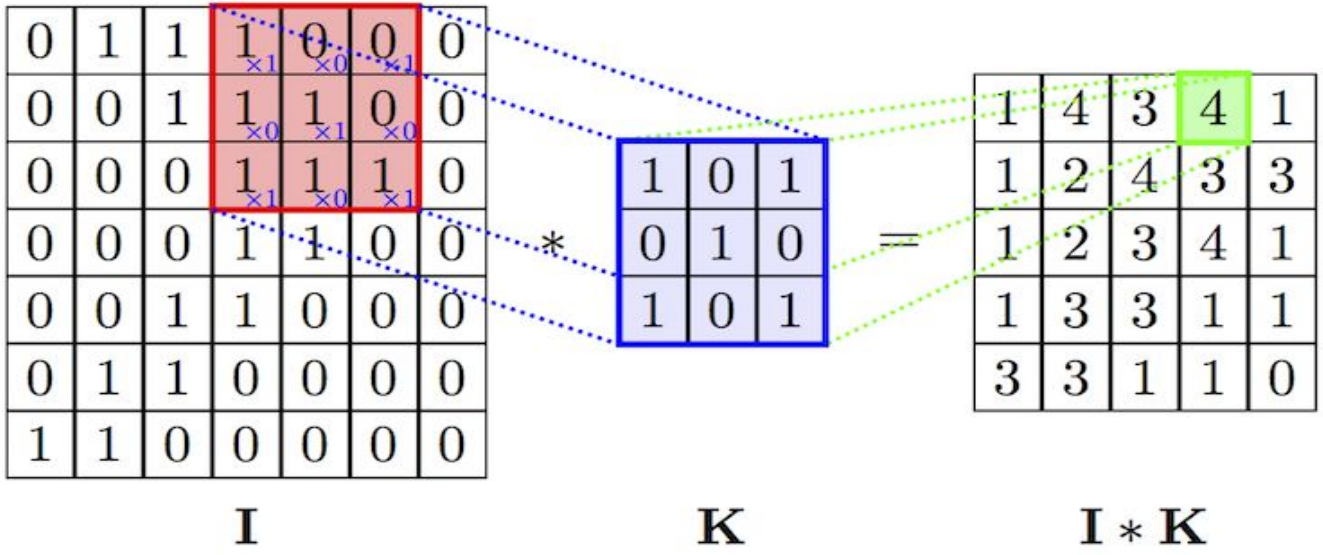
2. El exceso de parámetros tiende al overfitting.

3. Las imágenes tienen una estructura espacial que la red tiene que aprender, pero que sería mucho más sencillo dárselo resuelto de antemano.

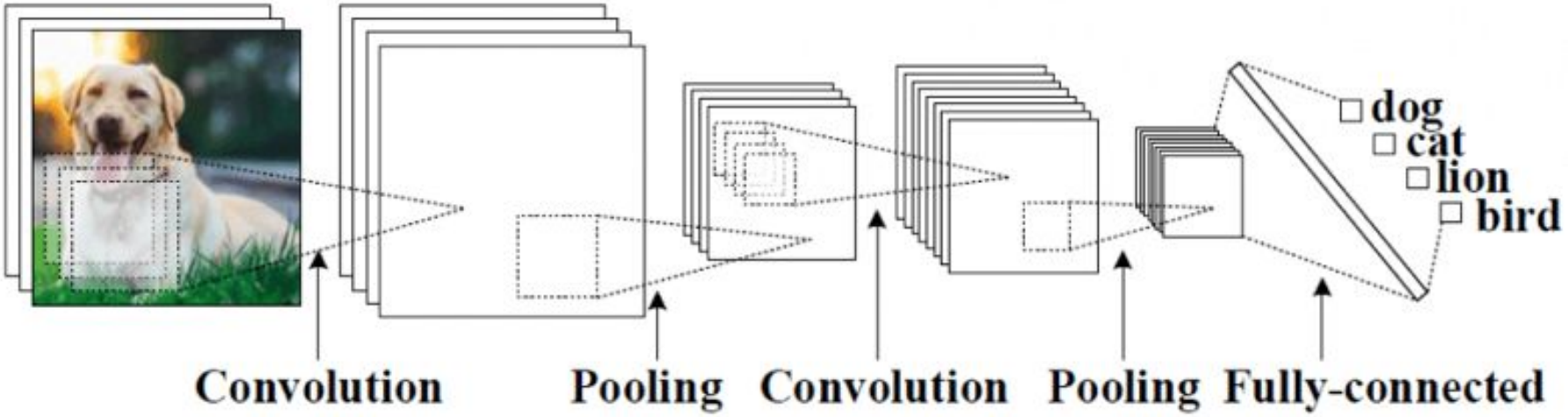
Capa convolucional / Filtros

- La idea es transformar la información espacial en características (feature maps).
- Para eso se utilizan filtros o “kernels”. Cada filtro será capaz de extraer patrones de la imagen, los cuales en sucesivas capas se van convirtiendo en los feature maps.

Funcionamiento de un filtro sobre una imagen monocromática (1 canal):



Estructura básica de una CNN

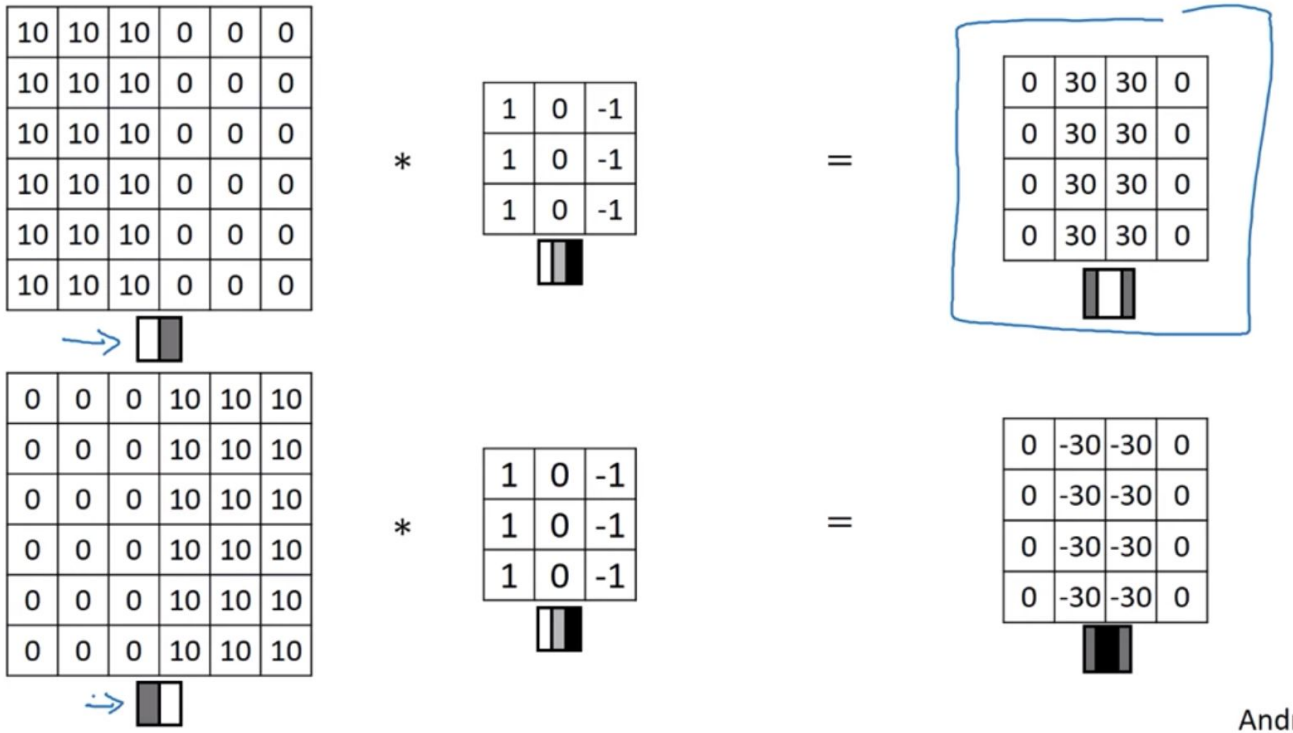


Capas:

- Convolucional (con o sin función de activación): feature extraction
- Pooling: reducción de la dimensionalidad
- Fully-connected: interpreta los feature maps y realiza la clasificación

Efecto de los filtros

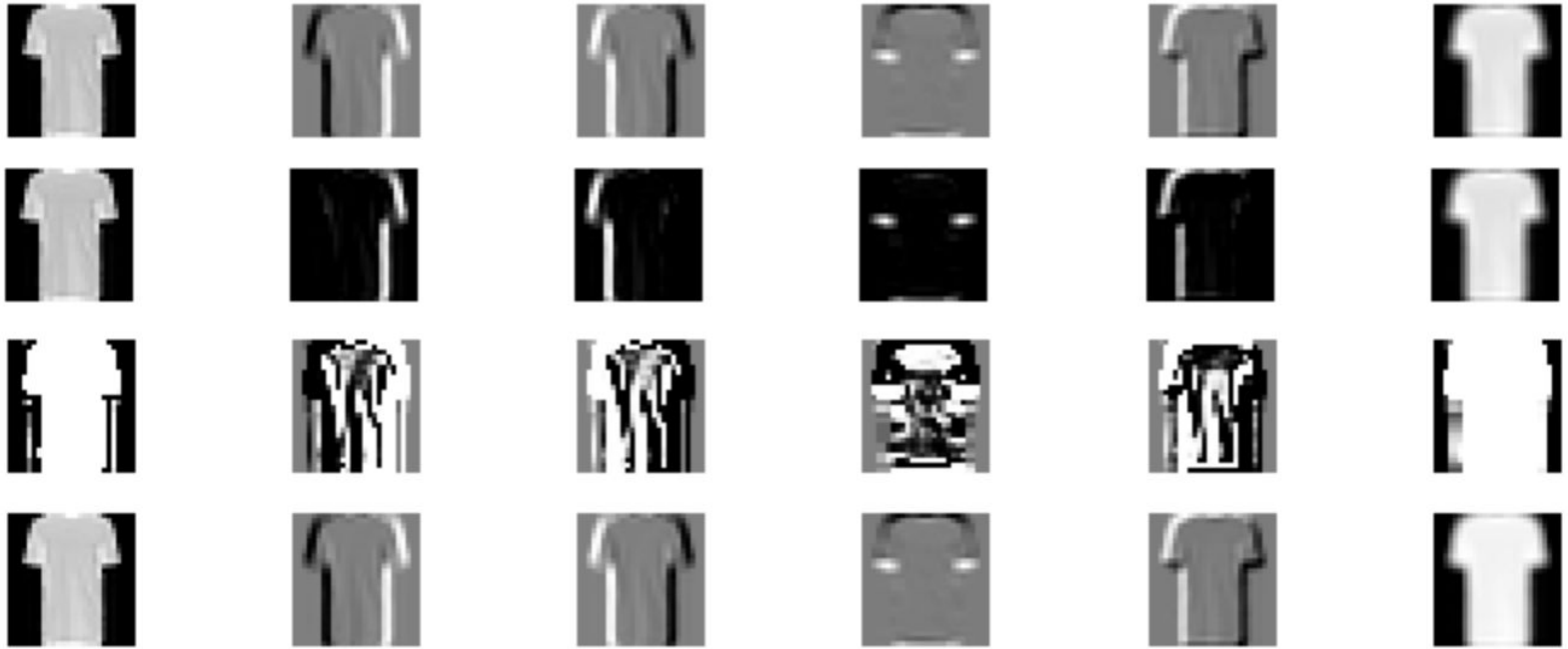
Ejemplo de detección de bordes:



Andrew Ng

Efecto de la función de activación

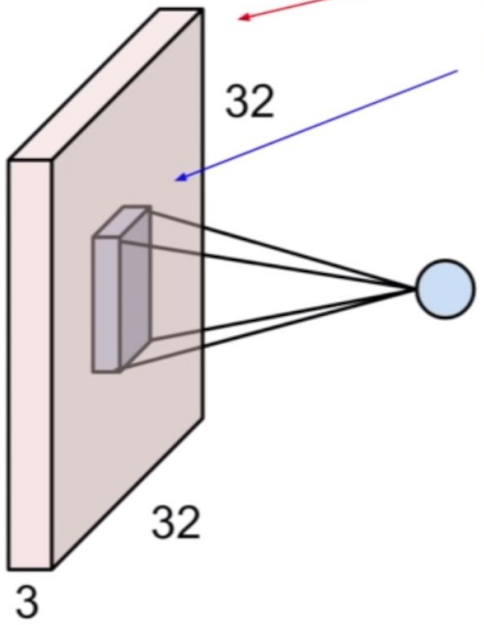
Los filtros pueden tener a su salida una función de activación y los efectos pueden verse acá:



Esto está visualizado mejor en la notebook: https://github.com/deeplearning-itba/cnn/blob/master/1-CNN-Teoria.ipynb

Un filtro visto como una neurona

En una CNN los valores de los filtros son los pesos (w_n) que la red “aprende” durante la etapa de back propagation del entrenamiento.



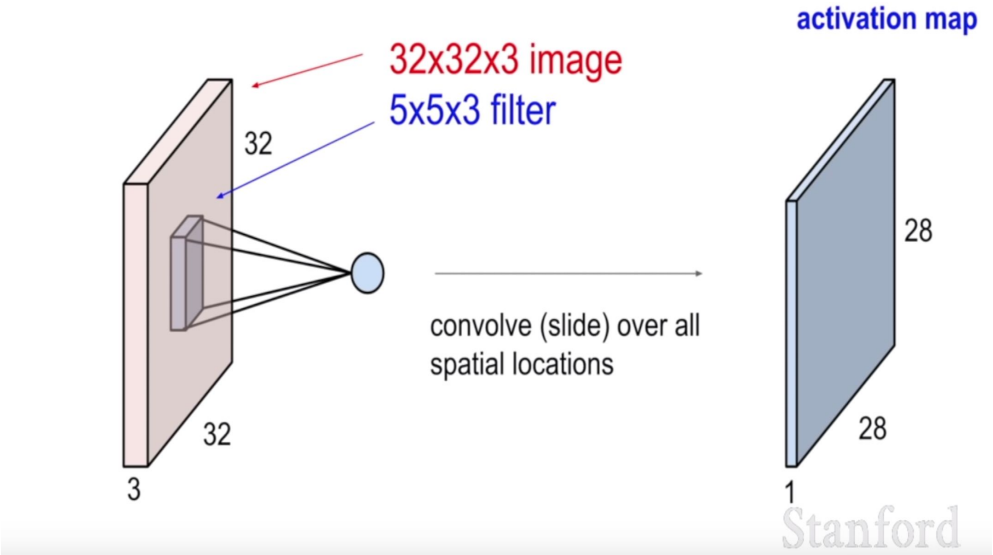
32x32x3 image

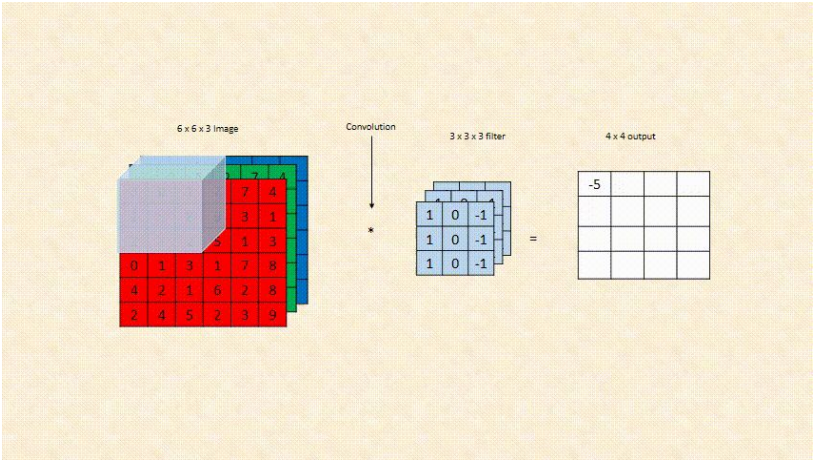
5x5x3 filter w

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

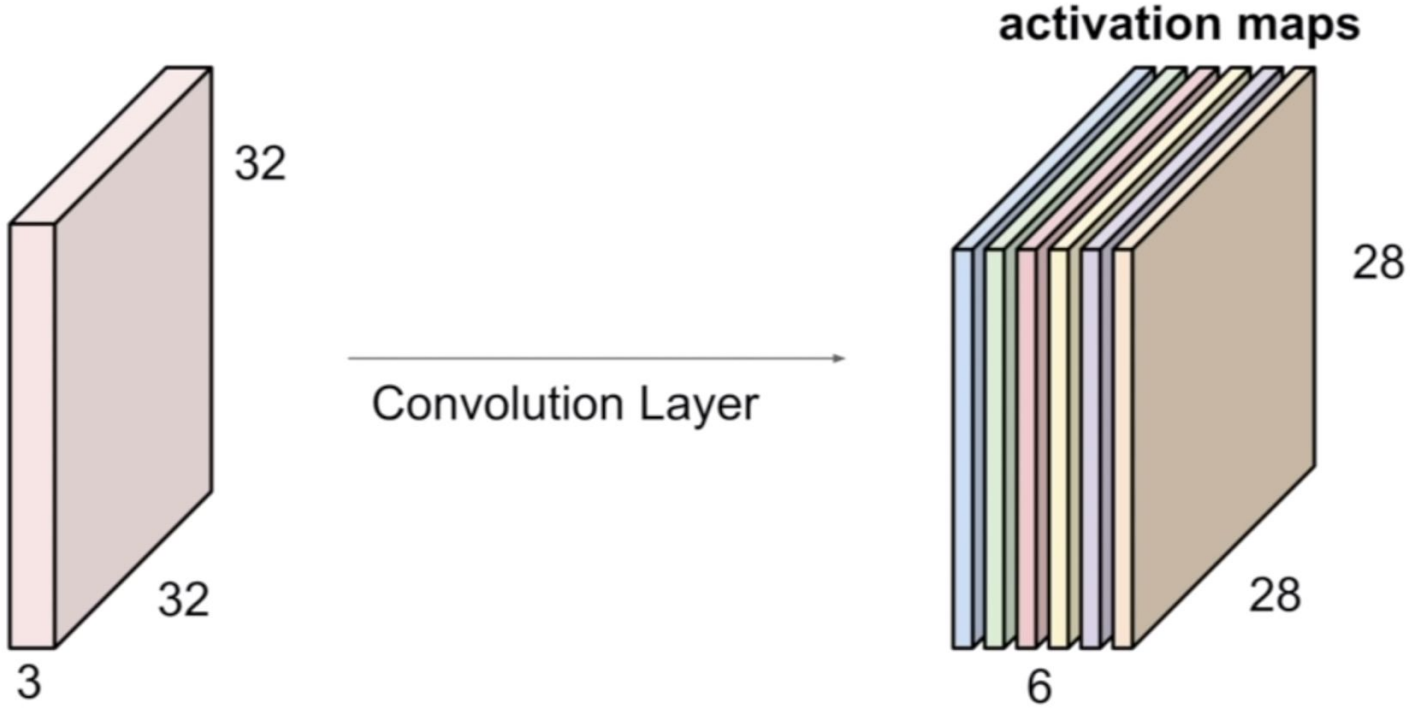
$$w^T x + b$$

Resultados de la convolución de 1 filtro





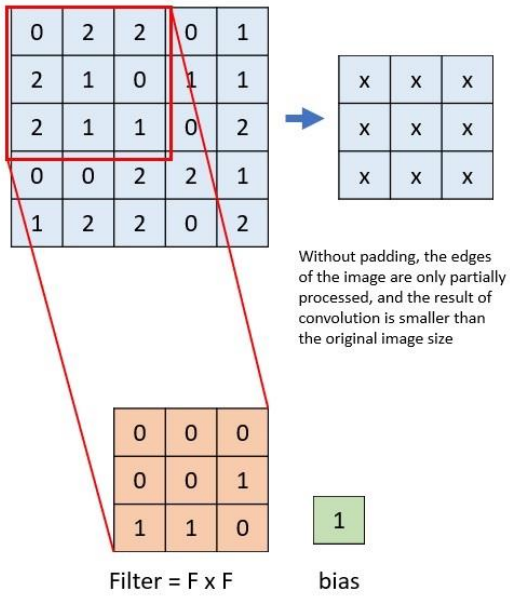
Ejemplo con 6 filtros

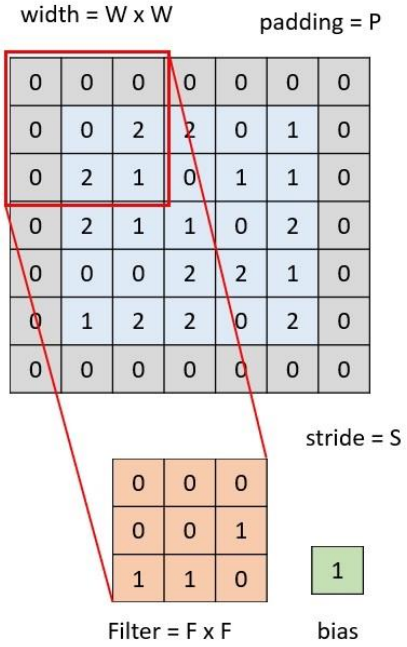


El resultado es equivalente a una “imagen” de 6 canales, donde cada canal/feature guarda características espaciales de la imagen original.

Hiperparámetros de la capa convolucional

filters: cantidad de filtros o “kernels” (K)
kernel_size: tamaño de cada filtro (F), ej: (3,3)
padding: ‘valid’ (p=0) o ‘same’ (p=1)
strides: normalmente (1, 1)
activation: None (‘linear’), ‘relu’, etc.





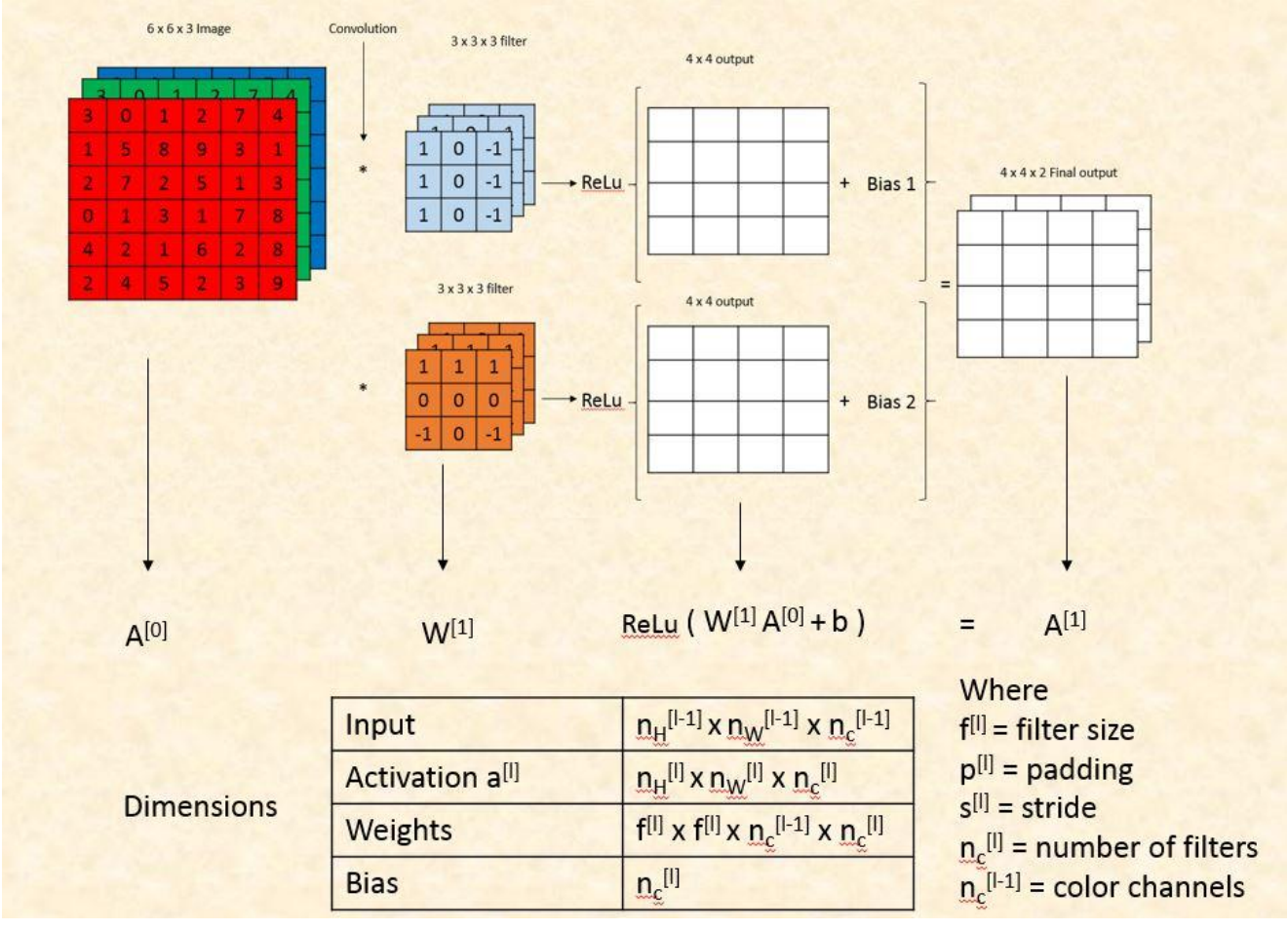
Dimensión de salida: $W_o \times H_o \times K$

$$W_o = \frac{(W - F + 2p)}{S} + 1$$

$$H_o = \frac{(H - F + 2p)}{S} + 1$$

K = cantidad de kernels

Dimensiones de la capa convolucional



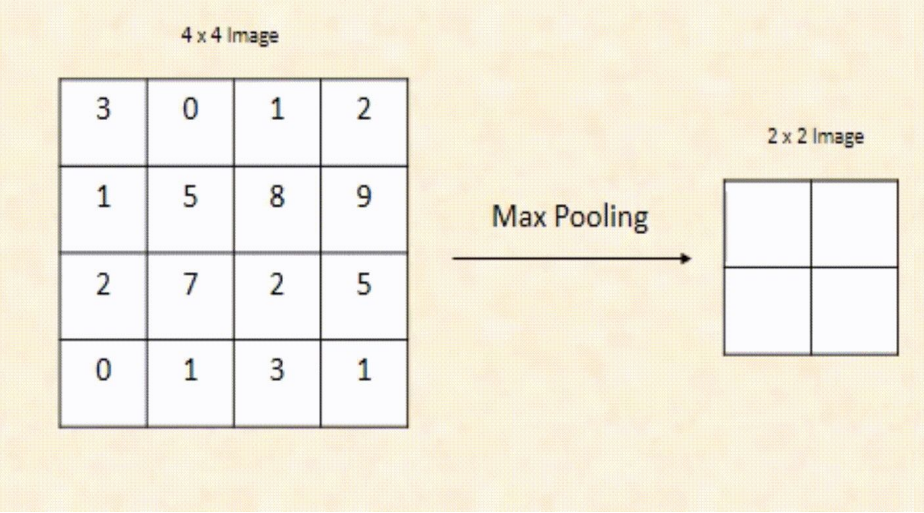
Dimensions

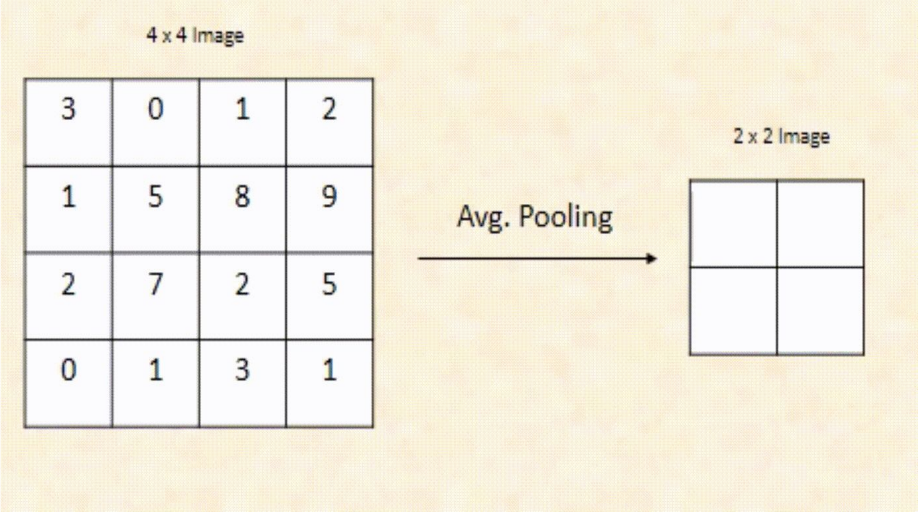
Input	$n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$
Activation $a^{[l]}$	$n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$
Weights	$f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$
Bias	$n_c^{[l]}$

Where
$f^{[l]}$ = filter size
$p^{[l]}$ = padding
$s^{[l]}$ = stride
$n_c^{[l]}$ = number of filters
$n_c^{[l-1]}$ = color channels

Capa pooling

Su función es reducir la cantidad de parámetros, bajar la dimensionalidad.
Tipos de pooling: Max, Average, Global Max, Global Average (explicar flatten).



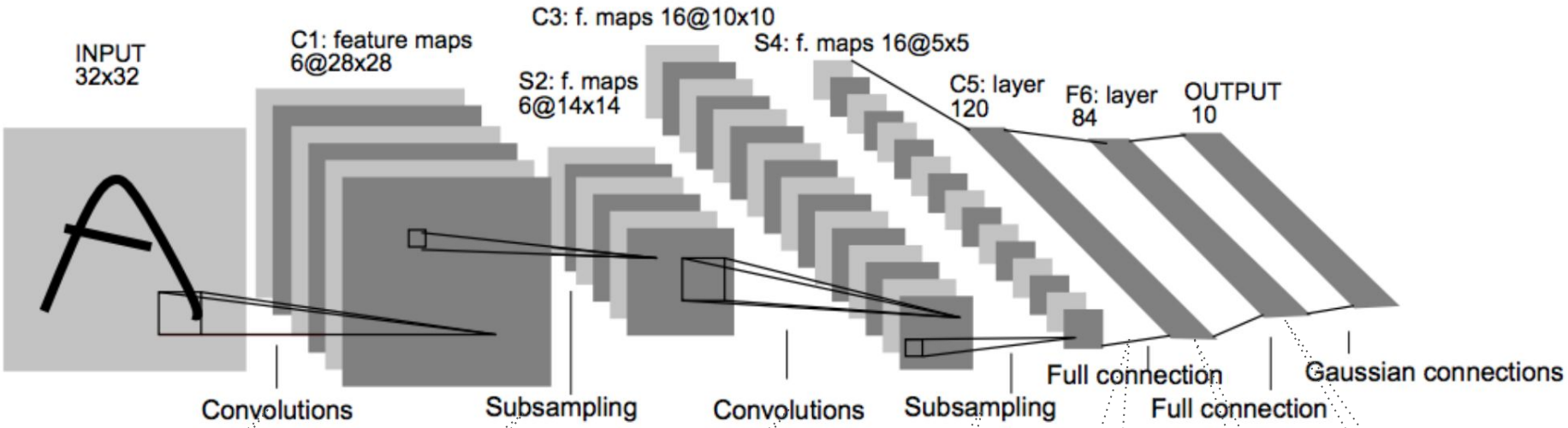


Hyperparámetros para Max y Average Pooling:
- Pool size
- Strides: en None por default, lo que significa que strides=pool_size

La capa de pooling no tienen ningún parámetro entrenable. Se aplica a cada canal por separado, es decir, no cambia la cantidad de canales a su salida, sólo el ancho y el alto.

Ejemplo completo de CNN: LeNet-5 (1998)

LeNet-5 fue desarrollado por Yann Lecun y presentado en 1998, y fue usado por el servicio postal para reconocer códigos postales escritos a mano. Este modelo fue pionero al introducir el concepto de red neuronal convolucional y es la base sobre la cual se diseñaron otras arquitecturas.



LeNet-5 Architecture. Credit: LeCun et al., 1998

6 filtros
F = 5x5
P = 0
S = 1

Average Pooling
2x2
S = 2

16 filtros
F = 5x5
P = 0
S = 1

Average Pooling
2x2
S = 2

Flatten => 400

FC: 120

FC: 84

Implementación de LeNet-5 con Keras

```
model = Sequential()

model.add(Conv2D(filters=6, kernel_size=(5, 5), activation='tanh', input_shape=(32,32,1)))

model.add(AveragePooling2D(pool_size=(2, 2)))

model.add(Conv2D(filters=16, kernel_size=(5, 5), activation='tanh'))

model.add(AveragePooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(units=120, activation='tanh'))

model.add(Dense(units=84, activation='tanh'))

model.add(Dense(units=10, activation = 'softmax'))

model.compile(loss='categorical_crossentropy', optimizer='Adam', metrics=['accuracy'])
```

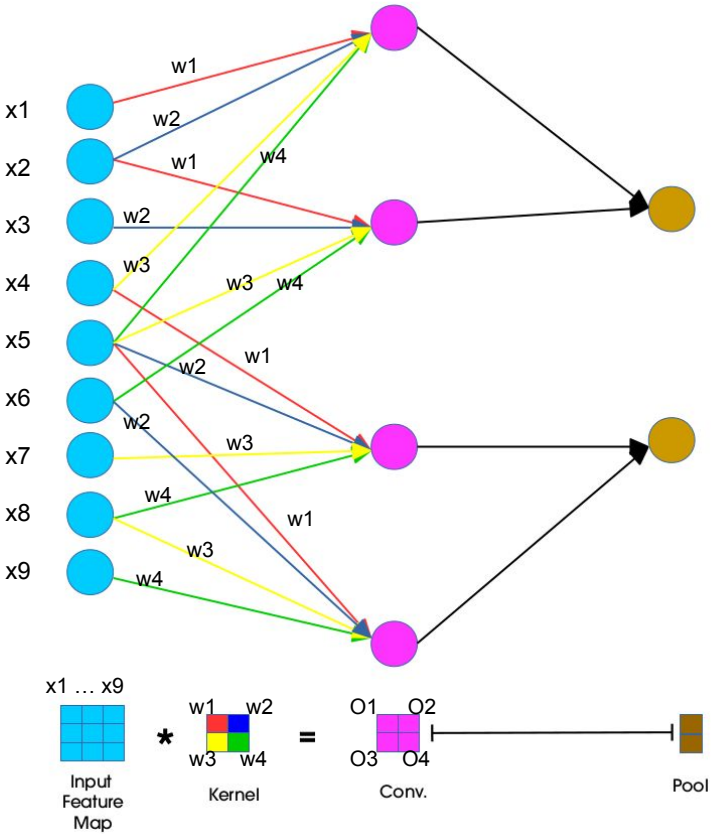
Implementación de LeNet-5 con Keras (cont.)

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 28, 28, 6)         156
_________________________________________________________________
average_pooling2d_3 (Average (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
average_pooling2d_4 (Average (None, 5, 5, 16)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_4 (Dense)              (None, 120)               48120
_________________________________________________________________
dense_5 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_6 (Dense)              (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
```

Link a la Notebook

Back propagation



Sin weight sharing:

De backpropagation:

$$\frac{\partial J}{\partial w_1} = -2 \sum_j \bar{e}_j \frac{\partial O_j}{\partial w_1}$$

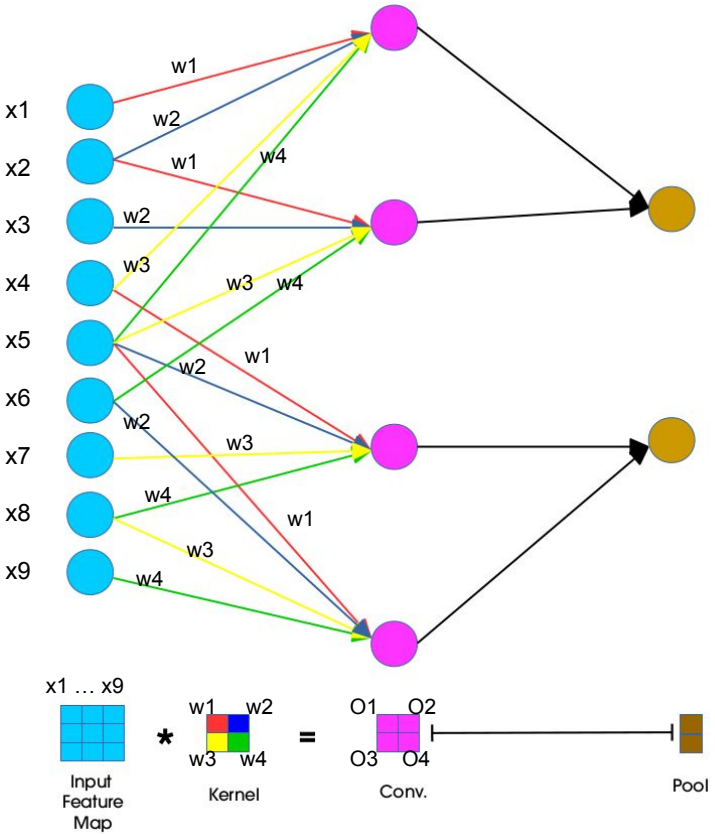
$$O_1 = x_1 . w_1 + x_2 . w_2 + x_4 . w_3 + x_5 . w_4$$
$$O_2 = x_2 . w_5 + x_3 . w_6 + x_5 . w_7 + x_6 . w_8$$
$$O_3 = x_4 . w_9 + x_5 . w_{10} + x_7 . w_{11} + x_8 . w_{12}$$
$$O_4 = x_5 . w_{13} + x_6 . w_{14} + x_8 . w_{15} + x_9 . w_{16}$$

$$\frac{\partial J}{\partial w_1} = -2 \bar{e}_j x_1$$

Como me quiero mover en dirección contraria al gradiente, con un learning rate η:

$$\Delta w_1 = 2 \eta \bar{e}_j x_1$$

Back propagation



Con weight sharing:

De backpropagation:

$$\frac{\partial J}{\partial w_1} = -2 \sum_j \bar{e}_j \frac{\partial O_j}{\partial w_1}$$

$$O_1 = x_1 . w_1 + x_2 . w_2 + x_4 . w_3 + x_5 . w_4$$
$$O_2 = x_2 . w_1 + x_3 . w_2 + x_5 . w_3 + x_6 . w_4$$
$$O_3 = x_4 . w_1 + x_5 . w_2 + x_7 . w_3 + x_8 . w_4$$
$$O_4 = x_5 . w_1 + x_6 . w_2 + x_8 . w_3 + x_9 . w_4$$

$$\frac{\partial J}{\partial w_1} = -2\bar{e}_j(x_1 + x_2 + x_4 + x_5)$$

Como me quiero mover en dirección contraria al gradiente, con un learning rate η:

$$\Delta w_1 = 2\eta\bar{e}_j(x_1 + x_2 + x_4 + x_5)$$

Image Viz

- https://blog.keras.io/how-convolutional-neural-networks-see-the-world.html
- http://cs231n.github.io/understanding-cnn/
- http://yosinski.com/deepvis

Bibliografía

1. Coursera: Convulutional Neural Networks, Andrew Ng
   https://www.coursera.org/learn/convolutional-neural-networks/
2. Stanford University: Lecture 5 | Convolutional Neural Networks
   https://www.youtube.com/watch?v=bNb2fEVKeEo
3. Convolutional Neural Networks, Muhammad Rizwan
   https://engmrk.com/convolutional-neural-network-3/
4. Deep Learning Book, Ian Goodfellow and Yoshua Bengio and Aaron Courville
   http://www.deeplearningbook.org/contents/convnets.html
1. Backpropagation in CNN
   https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/