



Visión por Computador II

CEAI, FIUBA

Profesores:

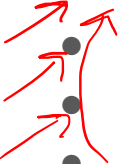
- Javier A. Kreiner, javkrei@gmail.com 
- Andrés F. Brumovsky, abrumov@gmail.com 

Plan del curso:

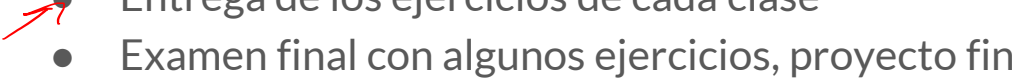
- Clase 1: Problemas en visión por computador: Clasificación de imágenes, detección de objetos, segmentación. Redes neuronales convolucionales. Orígenes. Estructura. Tipos de capas. Parámetros. Entrenamiento. Ejemplos básicos en keras.
- Clase 2: Data Augmentation. Ejemplos de arquitecturas clásicas: LeNet-5, AlexNet. Entrenamiento de una red para resolver un dataset en Kaggle.
- Clase 3: Residual Networks. Algunas arquitecturas: ResNet, Inception, VGGNet, UNet. Transfer Learning y su para dataset de Kaggle.
- Clase 4: Localización y detección de objetos. Algoritmos: R-CNN, Fast R-CNN, Faster R-CNN, YOLO.
- Clase 5: Implementación. Entrenamiento en un dataset. Segmentación semántica, segmentación panóptica. U-Net, EfficientNet. Implementación de segmentación.
- Clase 6: Aplicaciones: Estimación de pose, denoising, estimación de profundidad, segmentación de imágenes médicas.
- Clase 7: Aplicaciones en la industria. Algunos papers interesantes.
- Clase 8: Examen y proyecto final.



Dinámica de las clases

- Segmento teórico
 - (Pausa)
 - Segmento práctico: programación
 - Loop
- 

Evaluación:

- Entrega de los ejercicios de cada clase
 - Examen final con algunos ejercicios, proyecto final
- 




Herramientas

- Python
- Librerías comunes: Numpy, Pandas, Matplotlib
- Librería para Deep Learning: Tensorflow/Keras
- Github
- Entorno local recomendado: Anaconda
- Colab para compartir más fácilmente



Bibliografía:

- Computer Vision: Algorithms and Applications, 2nd ed. : <https://szeliski.org/Book/>, R. Szeliski
- Deep Learning: <https://www.deeplearningbook.org/>, I. Goodfellow, J. Bengio, A. Courville



Para profundizar, no es necesaria para las clases.

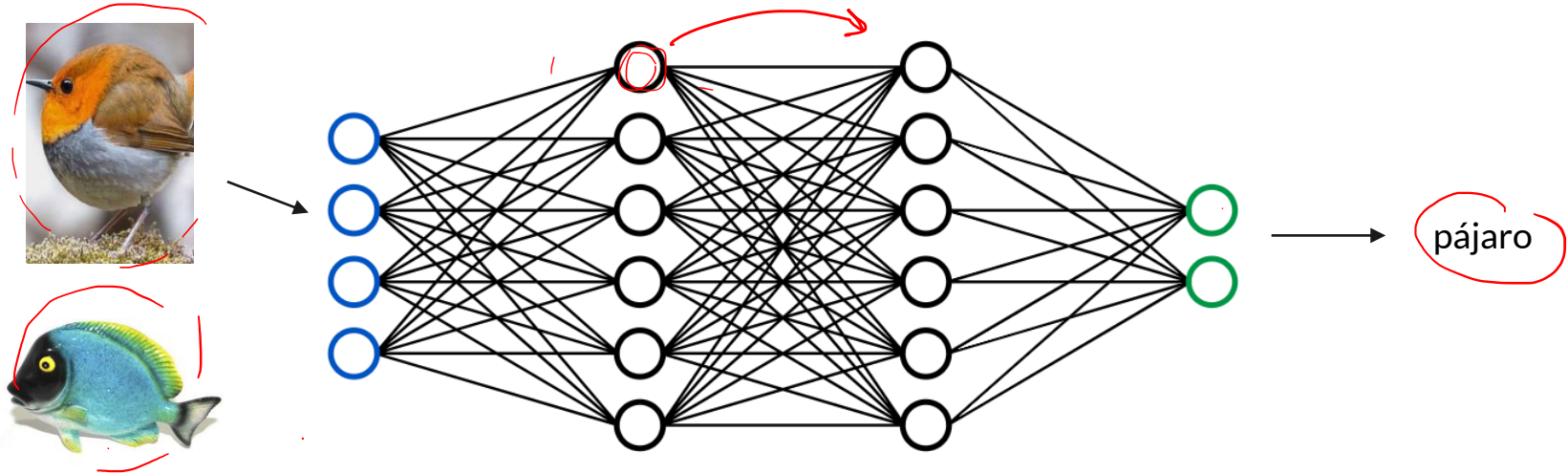
Primera clase:



- Tareas básicas en visión por computador: Clasificación de imágenes, detección de objetos, segmentación. Otras tareas: Transferencia de estilo, navegación autónoma, predicción de necesidad de riego, predicción de producción, diagnóstico médico, tracking de objetos, etc.
- Redes convolucionales
 - Orígenes
 - Tipos de capas
 - Operación de convolución
 - Estructura
 - Tipos de datos adecuados
- Implementación de capas
- Ejemplos básicos en keras

Clasificación (Image Classification)

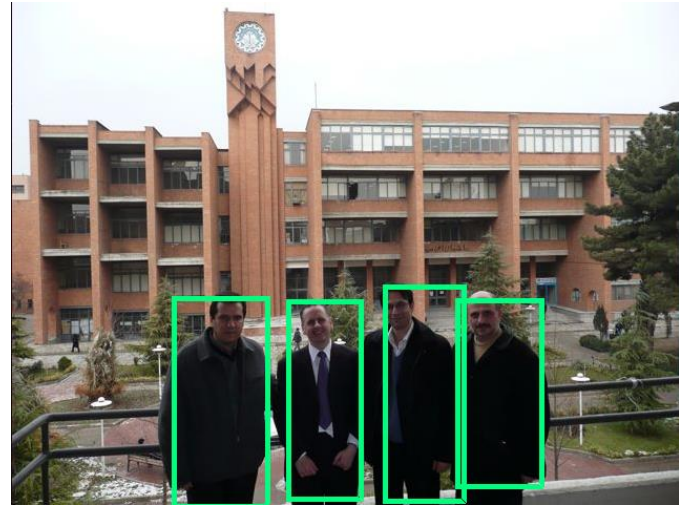
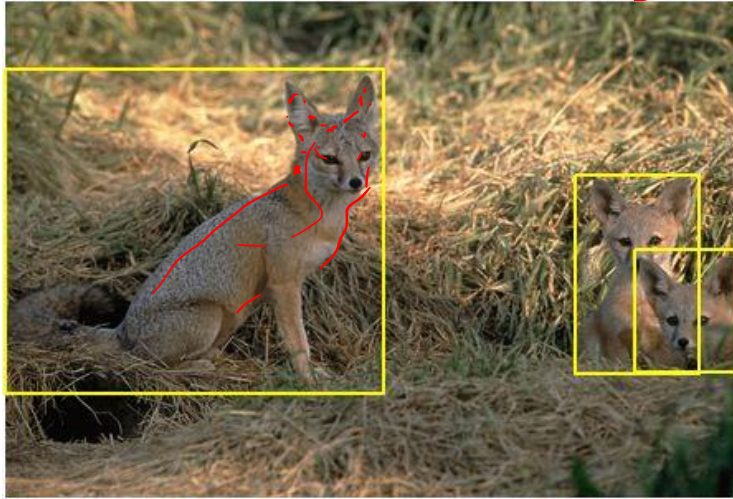
- Se trata de identificar en cuál categoría cae una imagen.
- Ejemplo:



Datasets clásicos: MNIST, CIFAR-10, ImageNet (usado para ILSVRC, ImageNet Large Scale Visual Recognition Challenge) y más en este [link](#)

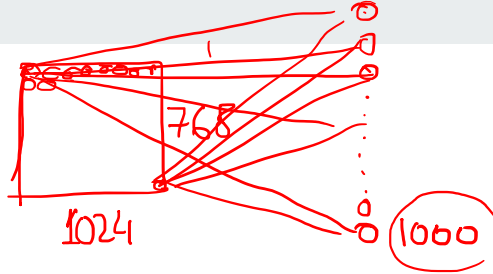
Localización de objetos (Object localization)

- El output son las 'bounding boxes' o recuadros que determinan los objetos/entidades de interés en la imagen. Ejemplos:



- En vez de recuadros el output podría ser clasificar cuáles píxeles son parte de un objeto y cuáles no (Object Segmentation). Dataset: Imagenet.

Algunos desafíos:



- Tamaño del espacio de las entradas. Para una imagen de 1024x768, el tamaño es de 786.432 números por cada canal. En general son tres canales (Rojo, Verde, Azul). Entonces serían 2.359.296 inputs.
- Si consideramos que la cantidad de pesos entre capas de una red neuronal (si son fully connected) es [cantidad de unidades de la capa de C-1] x [cantidad de entradas de la capa C]. Supongamos que la segunda capa tiene 1000 unidades, entonces tenemos ya ~ dos mil millones de parámetros entre la entrada y la primera capa.
- Esta enorme cantidad de parámetros hace que sea más fácil overfitear.
- Enorme necesidad de memoria y poder de cálculo.
- Una solución para este problema es utilizar redes convolucionales.



Redes convolucionales

- Redes neuronales compuestas por capas con ciertas características
- Inspiración biológica: basado en el procesamiento visual de animales
- Inspiración 'teórica': consideraciones de invarianzas (por ejemplo, un objeto debe ser reconocido como tal independientemente de la posición, orientación o escala en el campo visual)
- Necesidad de reducir la cantidad de parámetros de una red neuronal:
 - disminuir tiempo de cómputo
 - disminuir probabilidad de overfitting

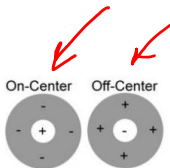
Inspiración biológica

- Experimentos demostraron una organización jerárquica de la corteza visual en la que áreas iniciales en el procesamiento detectan características (features) simples y luego esta información es subsecuentemente procesada por regiones posteriores para obtener características más complejas. -> imagen del flujo de esta información
- Se descubrió en experimentos con gatos que la primera parte del procesamiento (neuronas en la corteza V1) codifican orientación y posición en el campo visual
- Los llamados receptive fields capturan información similar en cada posición del campo visual
- En particular en la corteza V1 de gatos se observaron:
 - Células simples (simple cells): detectan regiones del campo visual en que la imagen es on-center u off-center
 - Bordes y orientación
- Hubel and Wiesel, 1962 (Receptive fields, binocular interaction and functional architecture in the cat's visual cortex)
- <https://www.youtube.com/watch?v=jw6nBW021Zk>,

Tipos de células en la corteza V1

Células simples y complejas son sensibles a:

- Center-surround (diferencia de gaussianas)



- Bordes

- Rectángulos de diferentes formas, mitades del campo visual

También hay otras neuronas sensibles a:

- Movimiento

- Color

- Otras cosas

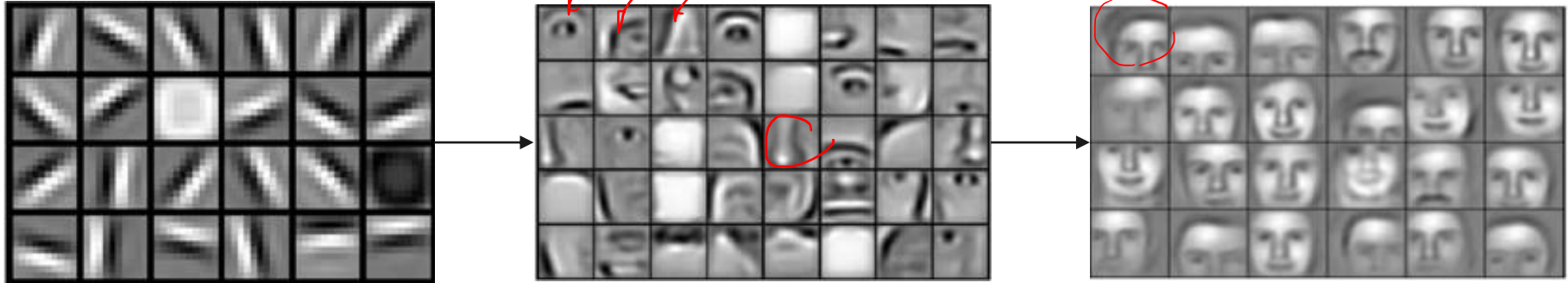
Video de Wiesel contando esto: <https://www.youtube.com/watch?v=aqzWy-zALzY>

Cantidad de parámetros e invarianza

- Parámetros compartidos: invarianza
- Disminución de cantidad de params
- Origen de la operación de convolución
- Conexiones esparzas
- Invarianza a la translación
- Disminución de la probabilidad de overfitting

Redes neuronales para visión por computador

- Capa detectora de bordes
- Capa detectora de partes de caras
- Capa detectora de caras enteras



Ejemplo de convolución

$1 \rightarrow 2 + 1$

$3 + 5 + 8 - 7 - 4 - 1$

5	0	3	3	7
9	3	5	2	4
7	6	8	8	1
6	7	7	8	1
6	9	8	9	4

*

1	0	-1
1	0	-1
1	0	-1

=

5	-4	4

- colab: https://colab.research.google.com/drive/1bVNiRii_e8eKPYYZc7bHg_UJGISBtdnj?usp=sharing

Ejemplo: detección de bordes verticales

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

1	0	-1
1	0	-1
1	0	-1

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

- colab: https://colab.research.google.com/drive/1bVNiRii_e8eKPYYZc7bHg_UJGISBtdnj?usp=sharing

Ejemplo: detección de bordes verticales

0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

*

1	0	-1
1	0	-1
1	0	-1

=

0	-30	-30	0
0	-30	-30	0
0	-30	-30	0
0	-30	-30	0

Ejemplo: detección de bordes horizontales

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

*

1	1	1
0	0	0
-1	-1	-1

=

0	0	0	0
30	10	-10	-30
30	10	-10	-30
0	0	0	0

Otros posibles detectores de bordes

1	1	1
0	0	0
-1	-1	-1

1	0	-1
1	0	-1
1	0	-1

Sobel/Sobel-Feldman

1	2	1
0	0	0
-1	-2	-1

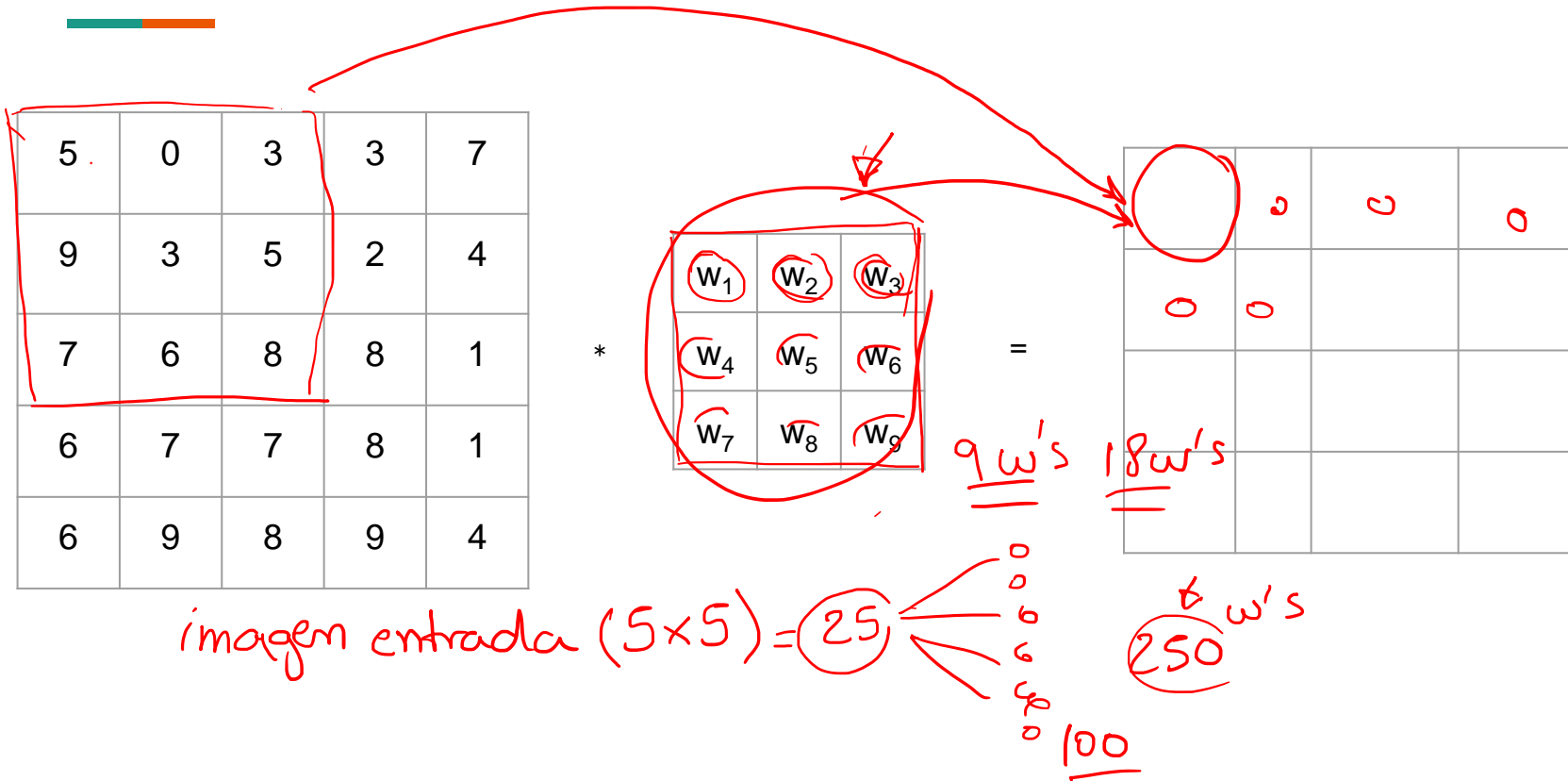
1	0	-1
2	0	-2
1	0	-1

Scharr

3	10	3
0	0	0
-3	-10	-3

3	10	3
0	0	0
-3	-10	-3

Aprender de los datos:



Capas fundamentales de redes convolucionales







Hay tres capas fundamentales:

- Capa convolucional (CONV)
- Capa de pooling (POOL)
- Capa Fully Connected (FC)



Hiperparámetros:

- padding (p) 
- stride (s) 
- tamaño de los filtros (w, h) 
- cantidad de filtros 

Zero-Padding (completar con ceros)



0	0	0				
0	5	0	3	3	7	
0	9	3	5	2	4	
	7	6	8	8	1	
	6	7	7	8	1	
	6	9	8	9	4	

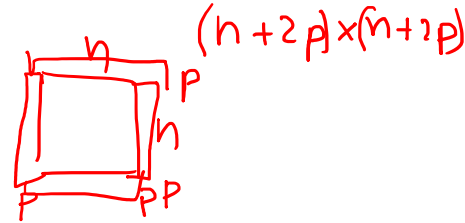
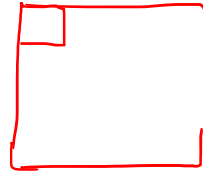


p
↔

0	0	0	0	0	0	0
0	5	0	3	3	7	0
0	9	3	5	2	4	0
0	7	6	8	8	1	0
0	6	7	7	8	1	0
0	6	9	8	9	4	0
0	0	0	0	0	0	0

Zero-Padding

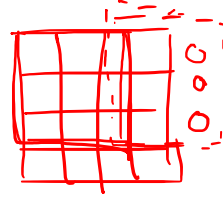
Cuando no hay padding:



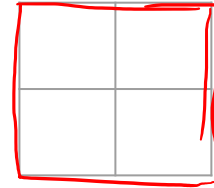
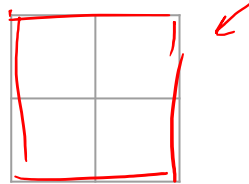
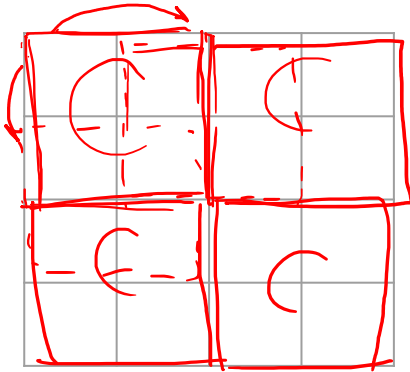
- la imagen se achica de $n \times n$ a $(n-f+1) \times (n-f+1)$
- los píxeles en los bordes son utilizados menos veces que los píxeles en el medio
- solución -> completar con ceros
- si p es el tamaño del 'padding', n el tamaño de la imagen y f el del filtro el tamaño de la salida será: $(n-f+2p+1) \times (n-f+2p+1)$
- dos tipos comunes: 'valid' y 'same'
- 'valid' -> sin padding
- 'same' -> p tal que la salida tiene el mismo tamaño que la entrada: $p = \text{floor}((f-1)/2)$, en general f es impar (el filtro tiene un centro bien definido)



Stride (zancada)



- stride es la 'zancada' que se da al desplazar el filtro
- n, f, p, s -> tamaño de la salida: $\text{floor}[(n + 2p - f) / s + 1] \times \text{floor}[(n + 2p - f) / s + 1]$
- divide el tamaño de la imagen por la zancada



$$n = 4, f = 2, p = 0, s = 2$$

Volumen de entrada y salida

- en general el filtro tiene la misma cantidad de canales que la entrada
- $n \times n \times n_c * f \times f \times n_c \rightarrow (n-f+1) \times (n-f+1) \times n_c$ (donde n_c es el número de filtros)
- bias + no linealidad

• $f^{[l]} :=$ tamaño del filtro

• $p^{[l]} :=$ padding

• $s^{[l]} :=$ stride

• $n_c^{[l]} :=$ cantidad de filtros

• entrada: $n_h^{[l-1]} \times n_w^{[l-1]} \times n_c^{[l-1]}$


• salida: $n_h^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

• $n_h^{[l]} = \text{floor}[(n_h^{[l-1]} + 2p^{[l]} - f^{[l]}) / s^{[l]} + 1]$

• cantidad de pesos: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$

• cantidad de bias: $n_c^{[l]}$

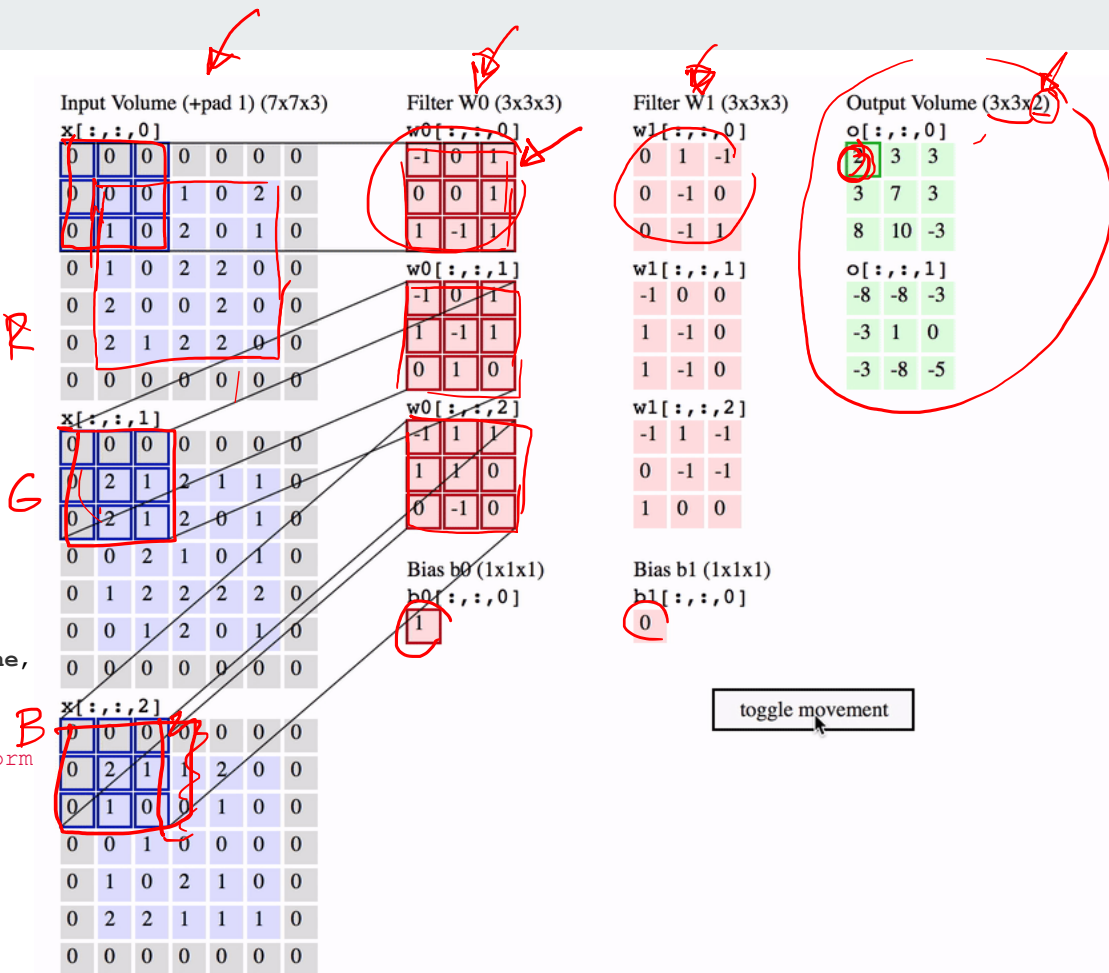
• ejemplo de calculo de cantidad de parámetros: filtros de $3 \times 3 \times 3$ + bias, 10 filtros: 280 parámetros,
fijos independientemente del tamaño de la imagen


$$\times n_c^{[l-1]} \times n_c^{[l]}$$

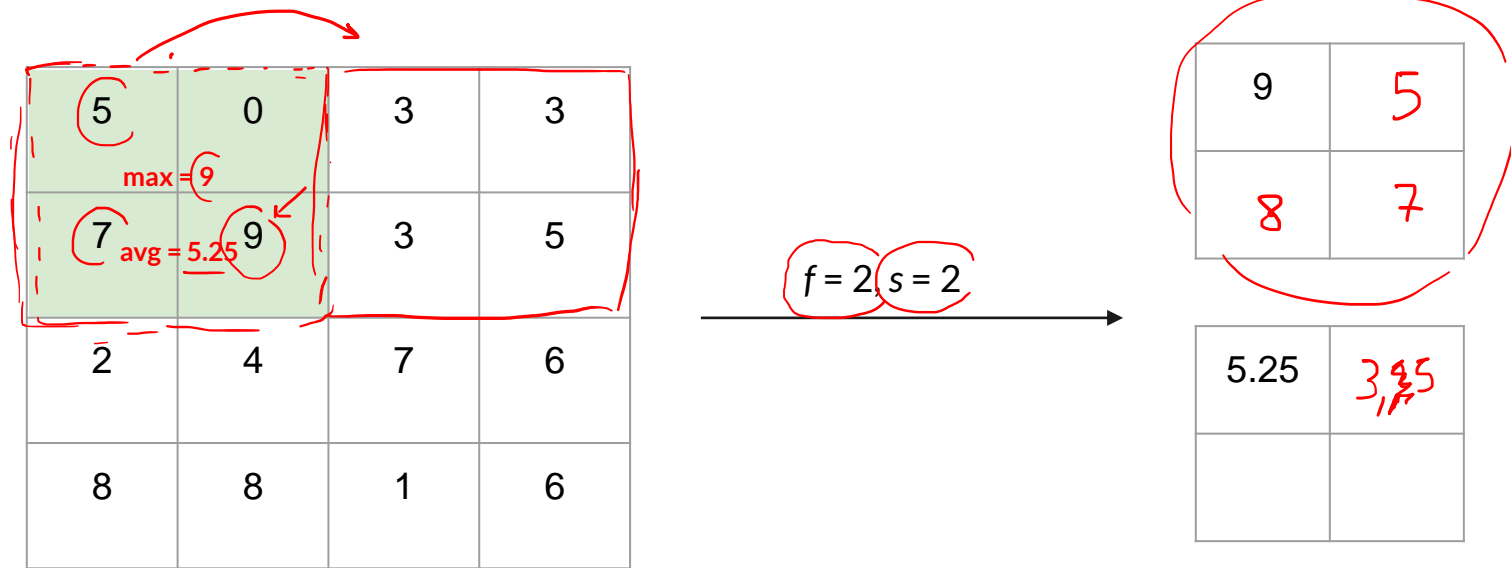
Capa convolucional

Parámetros:

- tamaño de los filtros: (n_w, n_h)
- tamaño del stride: (s_w, s_h)
- cantidad de filtros
- en keras:
 - `keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid', data_format=None, dilation_rate=(1, 1), activation=None, use_bias=True, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None, kernel_constraint=None, bias_constraint=None)`




Capa Pooling (max o average)



- colab: https://colab.research.google.com/drive/1JAuLgPj9ExCvNV7Sb8Ge-7_iPVS5Xco3?usp=sharing


Ejemplo capa Max Pooling:



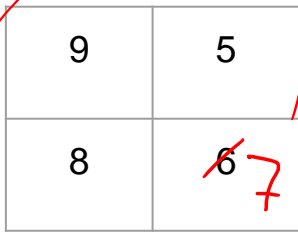
5	0	3	3
7	9	3	5
2	4	7	6
8	8	1	6



$f=2, s=2$



9	5
8	6 7





Ejercicio de programación:

- **Implementar una capa convolucional y una capa de tipo pool:**

<https://colab.research.google.com/drive/1HbcnXtyk4lzIUHJAAPNM-4ljMgUgagr?usp=sharing>

Cómo combinar estos layers para hacer una red simple



- Código en keras:

https://colab.research.google.com/drive/1Z7Fb_LXwxSiUYbugR83QGEEn0i4x85WZZ?usp=sharing

Ejercicio: Hacer una red convolucional que procese los mismos datos y tenga:

1. Capa convolucional, 8 filtros de 3x3
2. Capa max pool
3. Capa convolucional, 16 filtros de 3x3
4. Capa max pool
5. Capa convolucional, 32 filtros de 3x3
6. Capa max pool
7. Flatten
8. Capa densa
9. Salida

Calcular: ¿cuántos parámetros tiene esta red?



Ejemplo de utilizar una CNN simple para entrenar un modelo para reconocer señas

- https://colab.research.google.com/drive/1x_YcuuprDB7n3Pk4pbEqSaM6vrMTO9bZ?usp=sharing

Ejercicio: completar la tabla

	Dimensiones de activación	Tamaño de activación	# de parámetros
Entrada	(64, 64, 3)	12288	0
Conv2D(f=4, s=1,c=8, 'same')	(64, 64, 8)		
MaxPool(f=8, s=8)	(8,8,8)		
Conv2D(f=2, s=1, c=16, 'same')	(8,8,16)		
MaxPool(f=4, s=4)	(2,2,16)		
Dense(salida=6)			



Tarea

- completar las funciones `conv_forward` y `pool_forward` el colab:
<https://colab.research.google.com/drive/1HbcnXtyk4IzIUHJAAPNM-4IjMgUgagr?usp=sharing>
- visualizar los filtros (https://www.tensorflow.org/api_docs/python/tf/keras):
https://colab.research.google.com/drive/1Z7Fb_LXwxSiUYbugR83QGEEn0i4x85WZZ?usp=sharing
- completar la última celda:
https://colab.research.google.com/drive/1x_YcuuprDB7n3Pk4pbEqSaM6vrMTO9bZ?usp=sharing
- sin usar `summary()` completar la tabla de la slide anterior