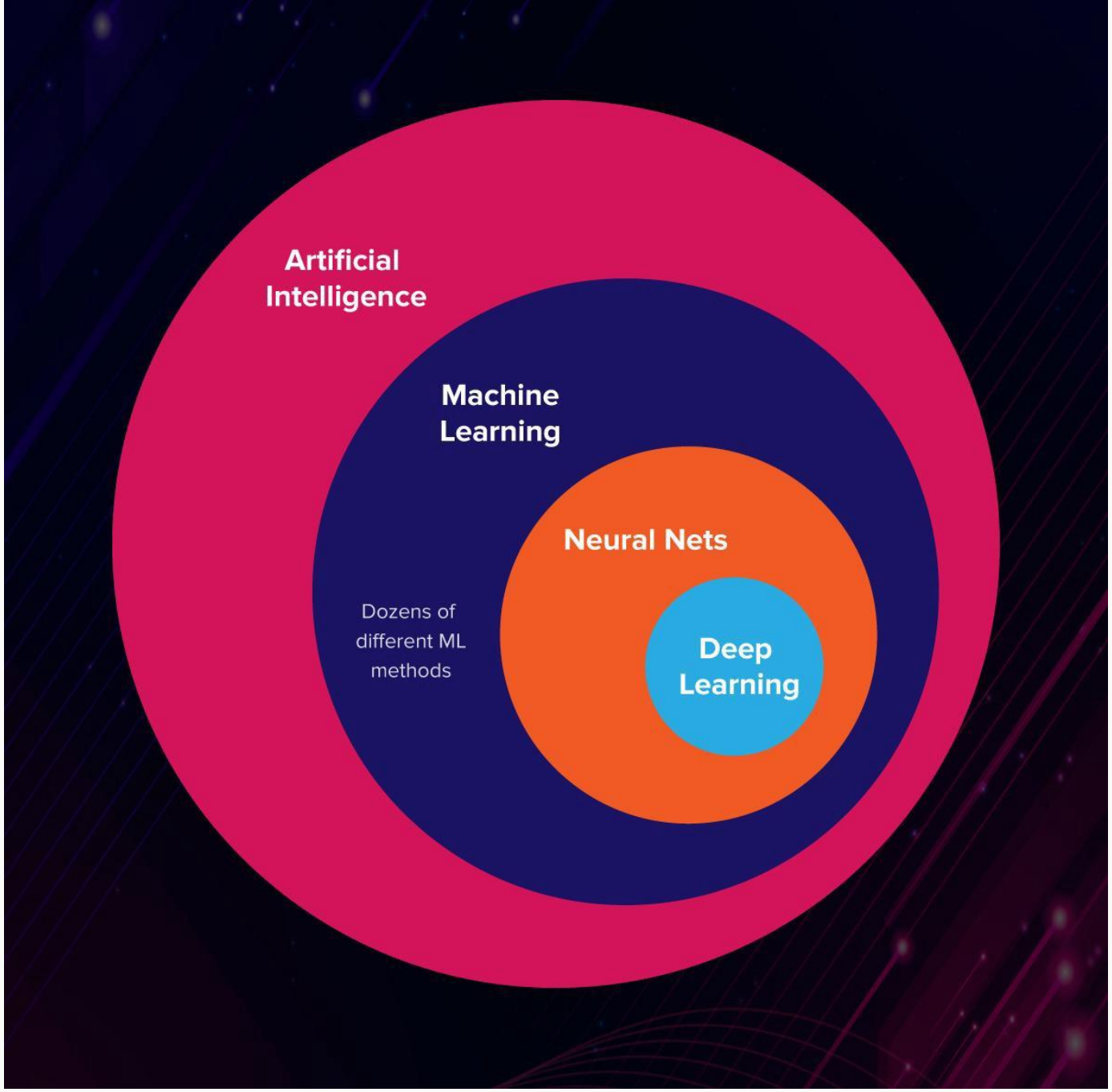
DEEP LEARNING


Artificial Intelligence
Machine Learning
Neural Nets
Deep Learning
Dozens of different ML methods

PROFUNDIDAD

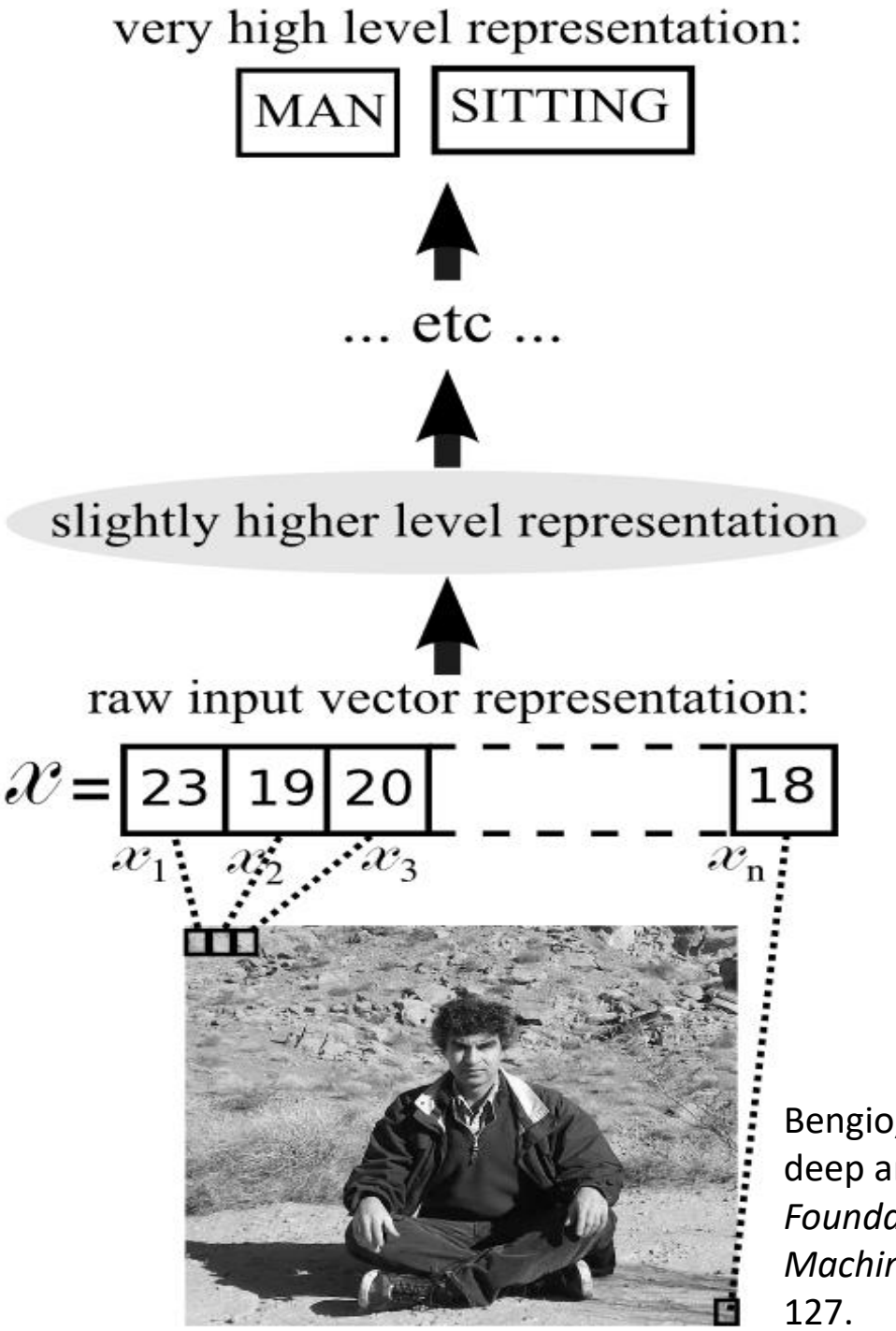
¿Qué es y por qué es necesaria?

APRENDIZAJE PROFUNDO

El Aprendizaje Profundo (*Deep Learning*) es una nueva área de investigación de Machine Learning, cuyo objetivo es acercar el aprendizaje automático a la Inteligencia Artificial.

Resultados teóricos sugieren fuertemente que para aprender el tipo de funciones complicadas que pueden representar abstracciones de alto nivel, uno necesita arquitecturas profundas.

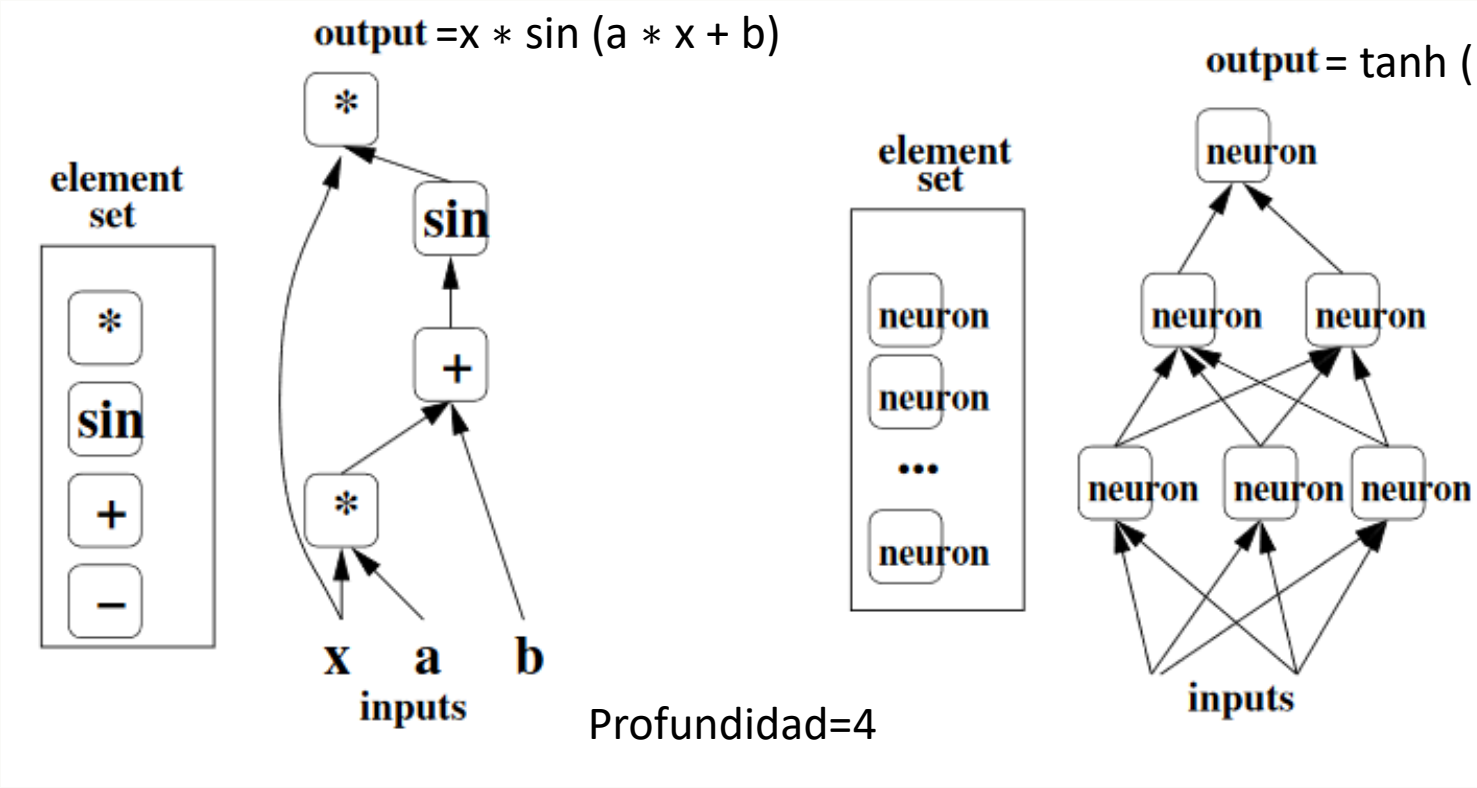
Las arquitecturas profundas se componen de múltiples niveles de operaciones no lineales, como en las redes neuronales con muchas capas ocultas.



Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and trends® in Machine Learning*, *2*(1), 1-127.

PROFUNDIDAD (CONJUNTO DE ELEMENTOS COMPUTACIONALES)

- Una función puede expresarse mediante la composición de elementos de este conjunto, utilizando un grafo que formaliza esta composición, con un nodo por elemento computacional.



Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and trends® in Machine Learning*, *2*(1), 1-127.

MÁS PROFUNDIDAD → MENOS ELEMENTOS DE CÓMPUTO

- Las máquinas de vectores de soporte (SVM) tienen una profundidad de 2 (una para las salidas del kernel o el espacio de características, y otra para la combinación lineal que produce la salida):

$$f(x) = b + \sum_i \alpha_i K(x, x_i)$$

- La profundidad 2 es suficiente en muchos casos (por ejemplo, compuertas lógicas, unidades de función de base radial [RBF] como en SVM) para representar cualquier función con una precisión objetivo determinada.

MÁS PROFUNDIDAD → MENOS ELEMENTOS DE CÓMPUTO

- Sin embargo, resultados teóricos demuestran que una arquitectura con profundidad insuficiente, puede requerir mucho (exponencialmente) más elementos computacionales (con respecto al tamaño de la entrada), que arquitecturas cuya profundidad empata bien con la tarea.

MÁS PROFUNDIDAD → REPRESENTACIONES COMPACTAS

- Si una función puede representarse de manera compacta con k niveles usando una elección particular de conjunto de elementos computacionales, puede requerir una gran cantidad de elementos computacionales para representarla con k − 1 o menos niveles (usando ese mismo conjunto de elementos computacionales)

MÁS PROFUNDIDAD→ REPRESENTACIONES DISTRIBUÍDAS

- Se pueden aprovechar las arquitecturas profundas para extraer múltiples niveles de representaciones distribuidas, donde el conjunto de configuraciones de valores en cada nivel del grafo de cálculo puede ser muy grande. Esto nos permitiría representar de manera compacta una función complicada de la entrada.

VARIABILIDAD DE LA ENTRADA

- La profundidad de la arquitectura está conectada a la noción de funciones altamente variables.
- En general, las arquitecturas profundas pueden representar de manera compacta funciones muy variables que de otro modo requerirían un tamaño muy grande para representarse con una arquitectura inapropiada.
- Decimos que una función es muy variable cuando una aproximación por partes (por ejemplo, constante por partes o lineal por partes) de esa función requeriría una gran cantidad de piezas.

EN RESUMEN

- Las arquitecturas profundas son útiles para:
  - *Reducir la cantidad de elementos de cómputo en el cálculo de funciones complejas.*
  - *Generar representaciones distribuídas del espacio de entrada.*
  - *Producir modelos que permitan generalizar funciones con entradas de alta variabilidad.*

PLATAFORMAS MÁS COMUNES PARA DEEP LEARNING

Keras es una librería de código abierto escrita en Python que funciona como una API (Application Programming Interface) de TensorFlow. Está diseñada para permitir una rápida experimentación con redes neuronales profundas.

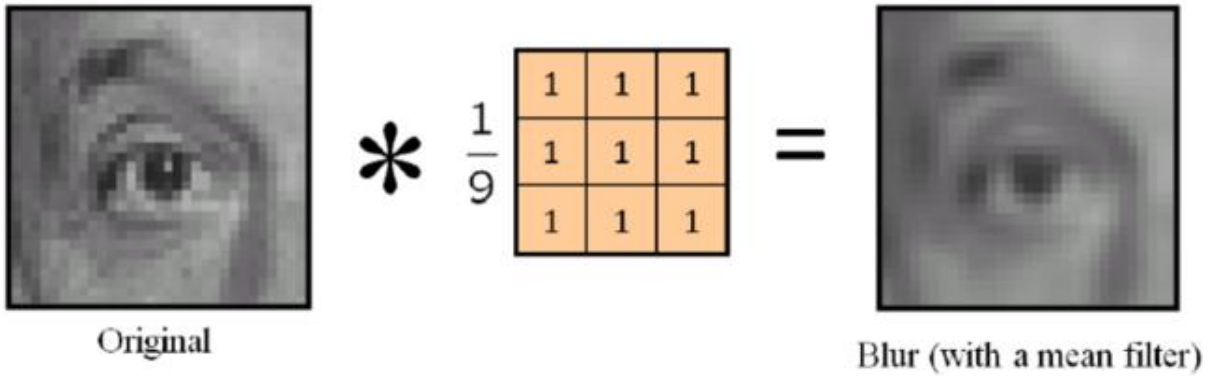
TensorFlow es una plataforma de código abierto para el aprendizaje automático.

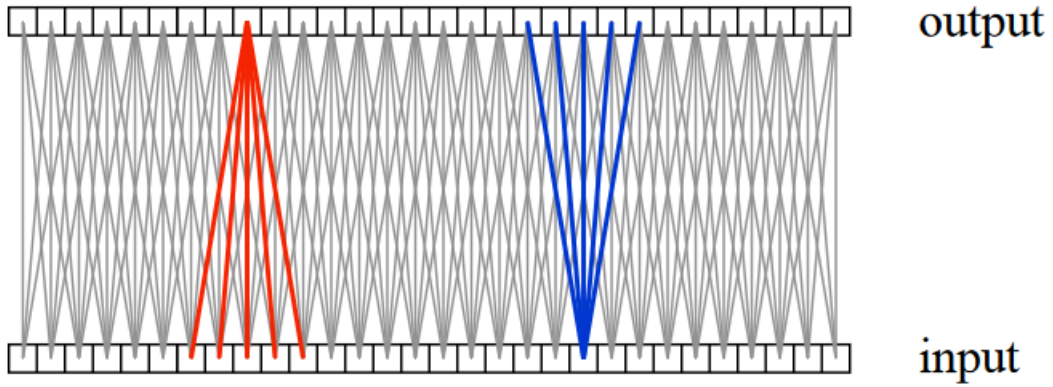
PyTorch es una librería de aprendizaje automático de código abierto para Python, basada en Torch. Se utiliza para aplicaciones como el procesamiento del lenguaje natural y fue desarrollado por el grupo de investigación de IA de Facebook.

REDES NEURONALES DE CONVOLUCIÓN (CNN)

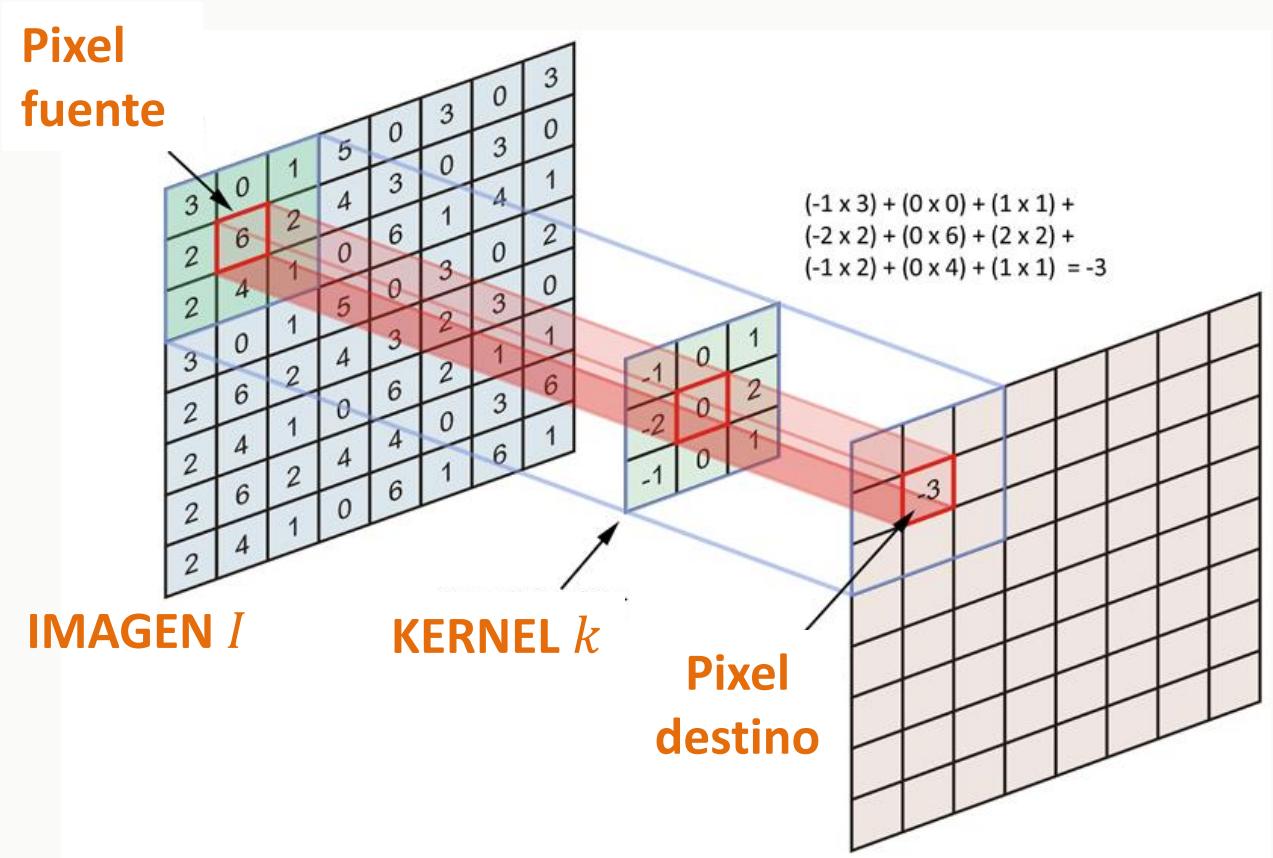
ASPECTOS BÁSICOS DEL APRENDIZAJE PROFUNDO

CONVOLUCIÓN COMO UNA TRANSFORMACIÓN





CONVOLUCIÓN



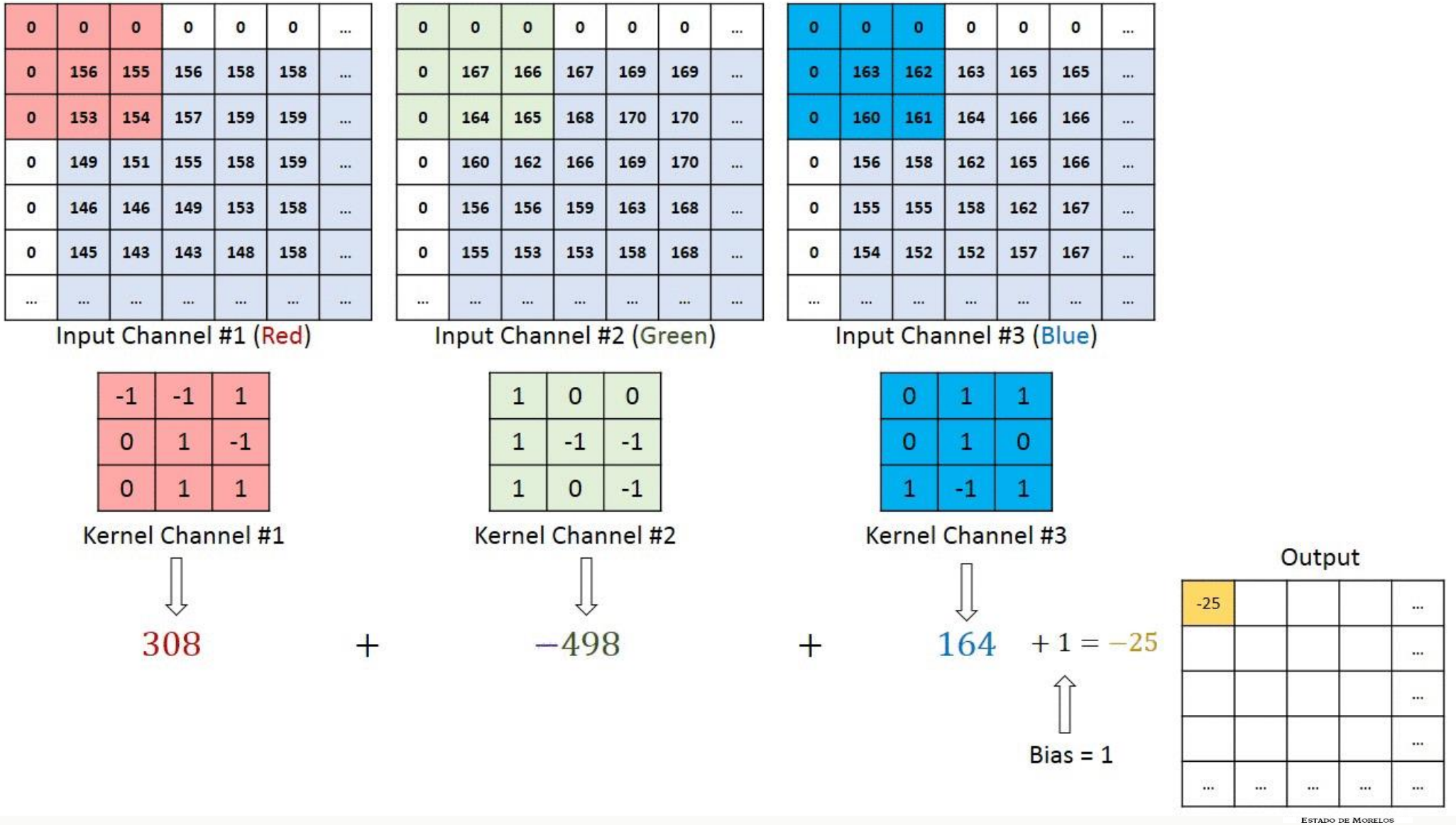
$$Z = I * k[x, y] = \sum_{i,j} I[x - i, y - j]k[i, j]$$

$$I[M \times N] * k[m \times n] \rightarrow [M - m + 1, N - n + 1]$$

Padding

- Zeros:
  $[5|4|2|3|7] \rightarrow [0|5|4|3|2|7|0]$
- Extended:
  $[5|4|2|3|7] \rightarrow [5|5|4|3|2|7|7]$
- Cyclic:
  $[5|4|2|3|7] \rightarrow [7|5|4|3|2|7|5]$
- Undefined:
  $[5|4|2|3|7] \rightarrow [?|5|4|3|2|7|?]$

RELLENO CERO (ZERO-PADDING) Y PASO (STRIDE)



CONVOLUCIÓN VS CORRELACIÓN

Convolución

- $I * k[x, y] = \sum_{i,j} I[x - i, y - j]k[i, j]$
- Aplicaciones:
  - *Suavizado*
  - *Afilado*

Correlación

- $I \circ k[x, y] = \sum_{i,j} I[x + i, y + j]k[i, j]$
- Aplicaciones:
  - *Comparación de plantillas*
  - *Detección de bordes*

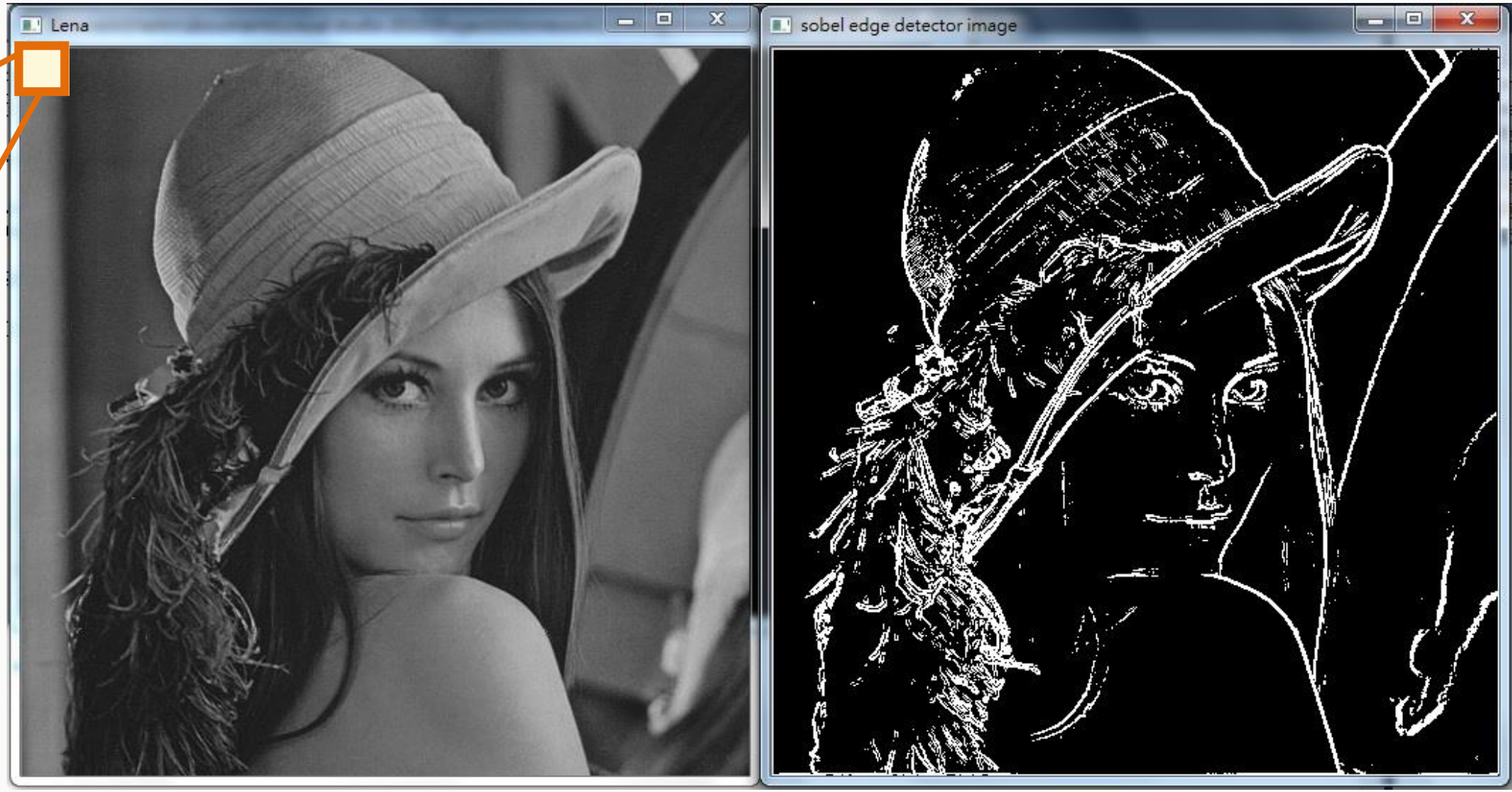
DETECCIÓN DE BORDES CON UN FILTRO SOBEL

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

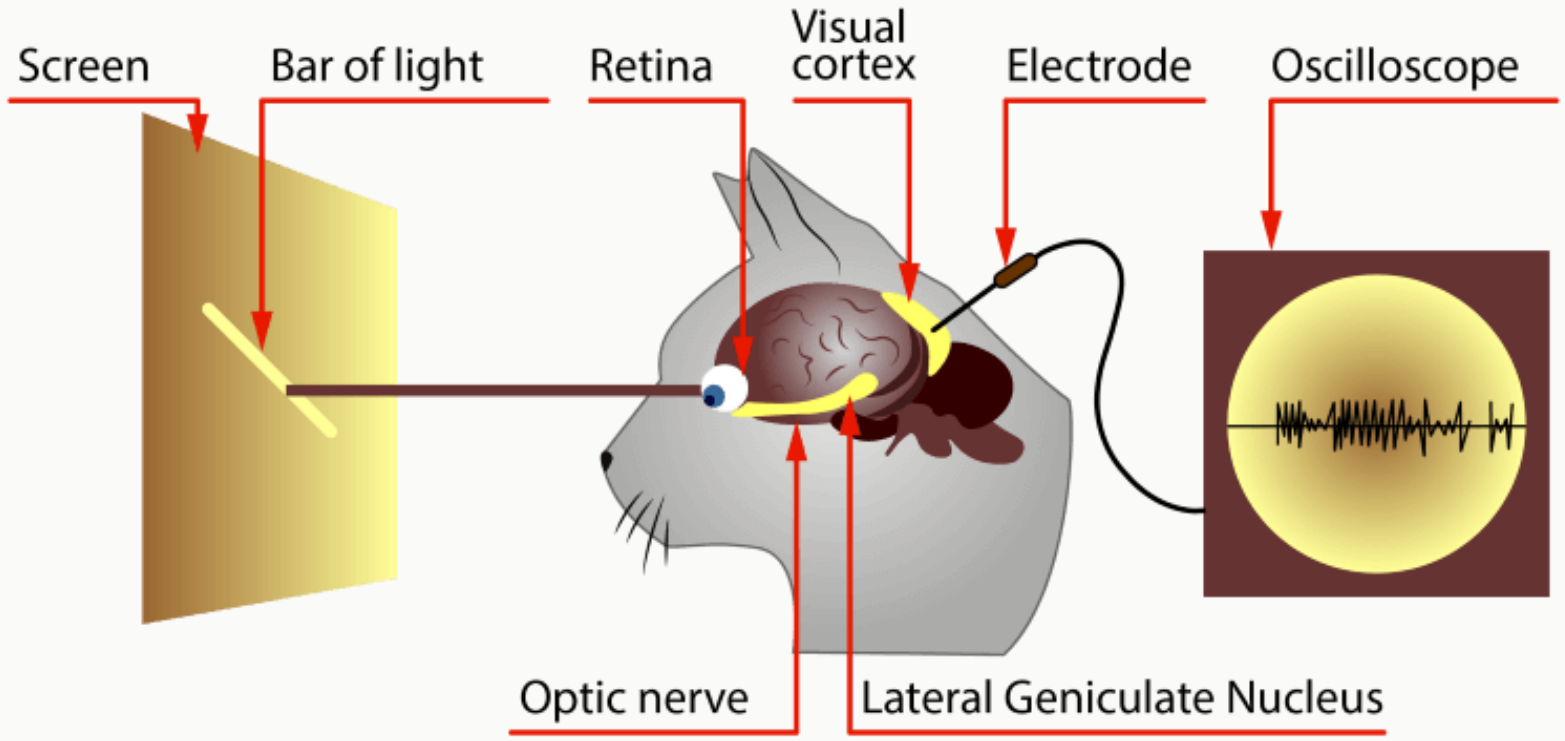
x filter

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

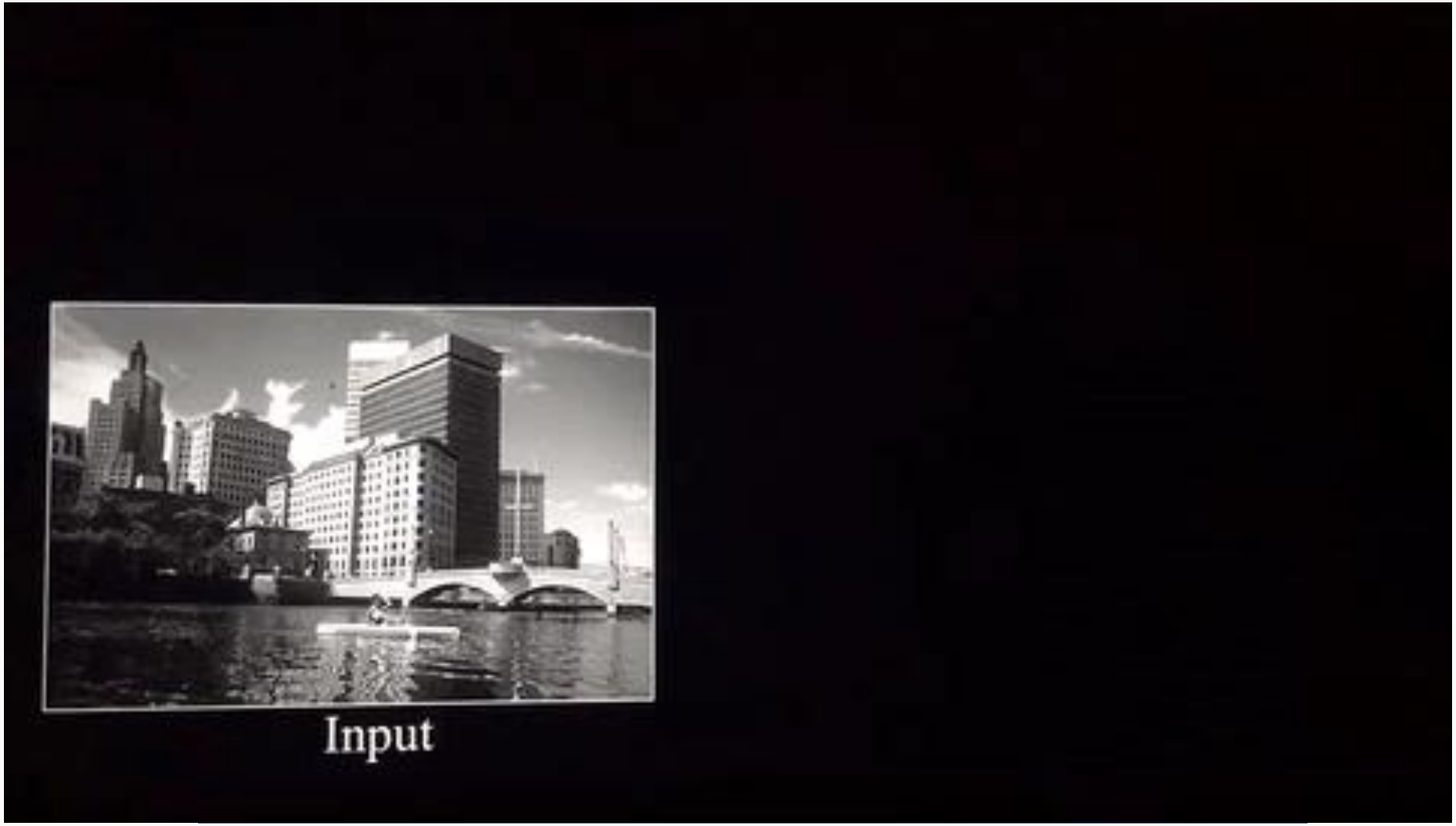
y filter



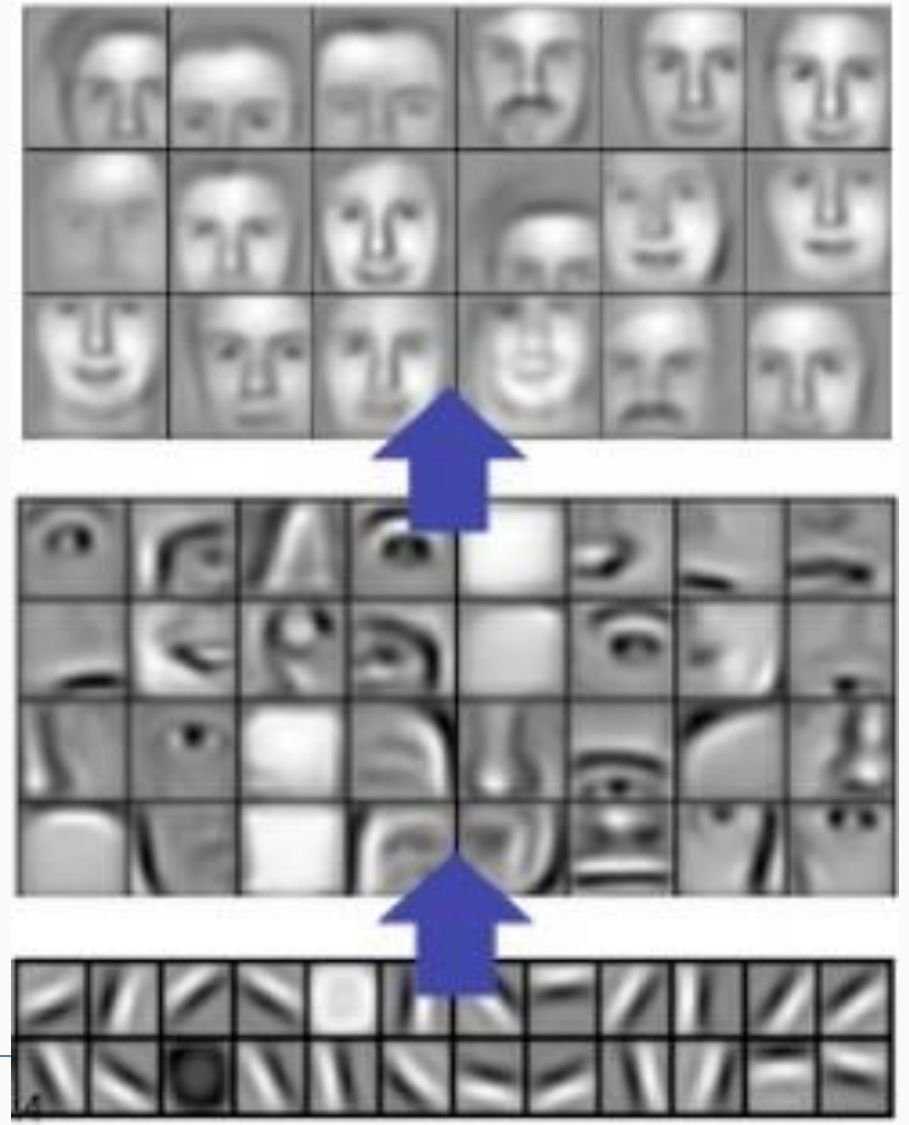
INSPIRACIÓN BIOLÓGICA DE LAS CNN



OPERACIÓN COMPUTACIONAL: LA CONVOLUCIÓN



OBJETIVO DE CNN: APRENDER LOS FILTROS (KERNELS)

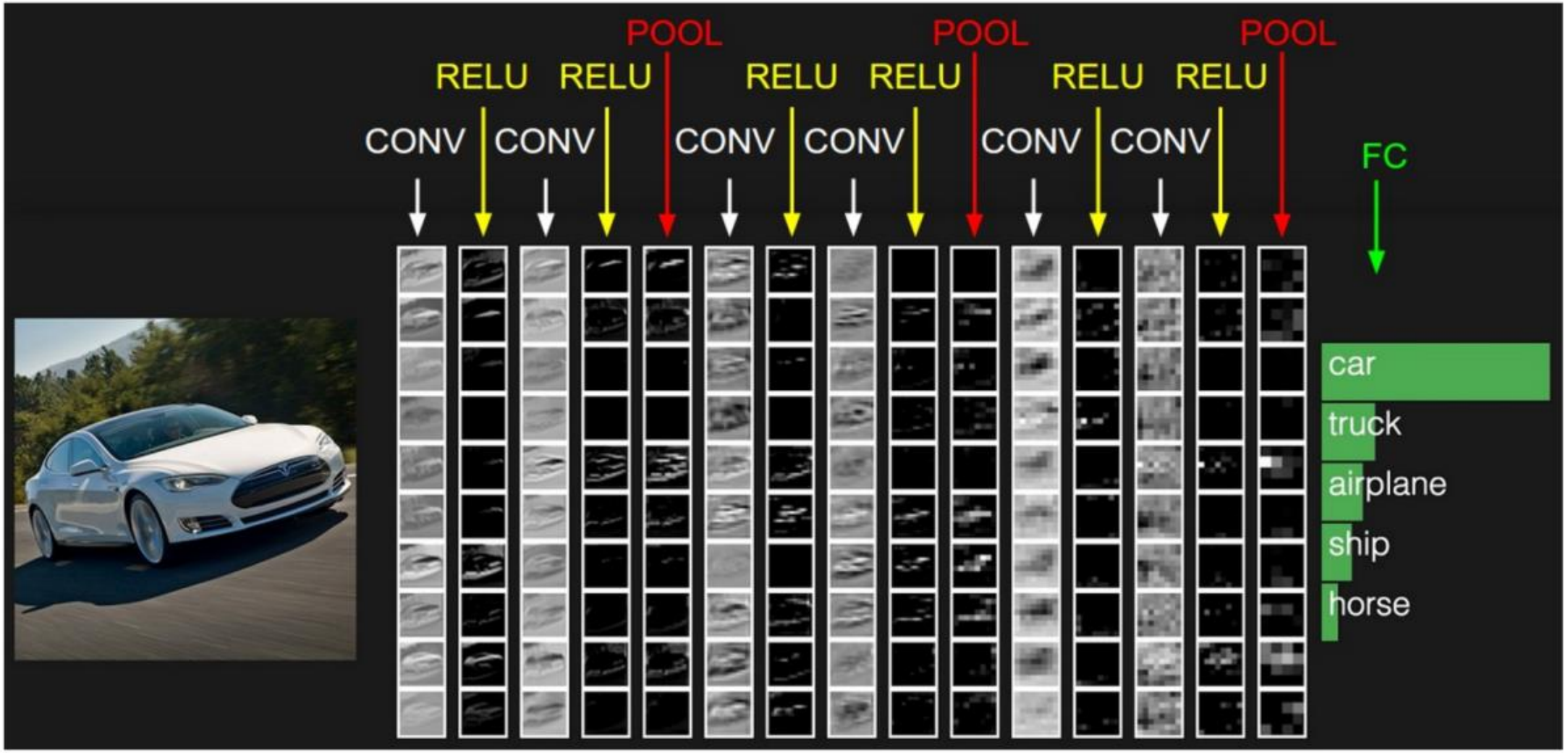


DETECCIÓN DE BORDES CON OPENCV Y CNN

[Holistically-Nested Edge Detection with OpenCV and Deep Learning]

Post de Adrian Rosebrock, 2019

CLASIFICACIÓN USANDO CNN

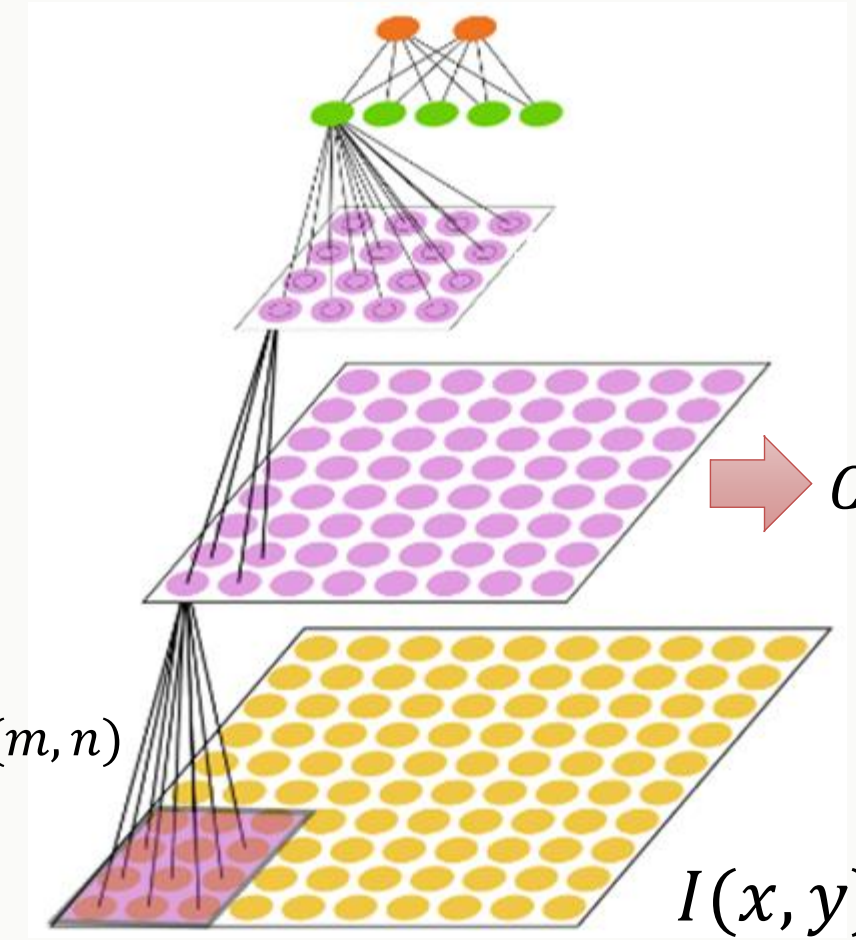


ARQUITECTURA YOLO

Detección usando un modelo pre-entrenado:
- git clone https://github.com/pjreddie/darknet
- cd darknet
- make

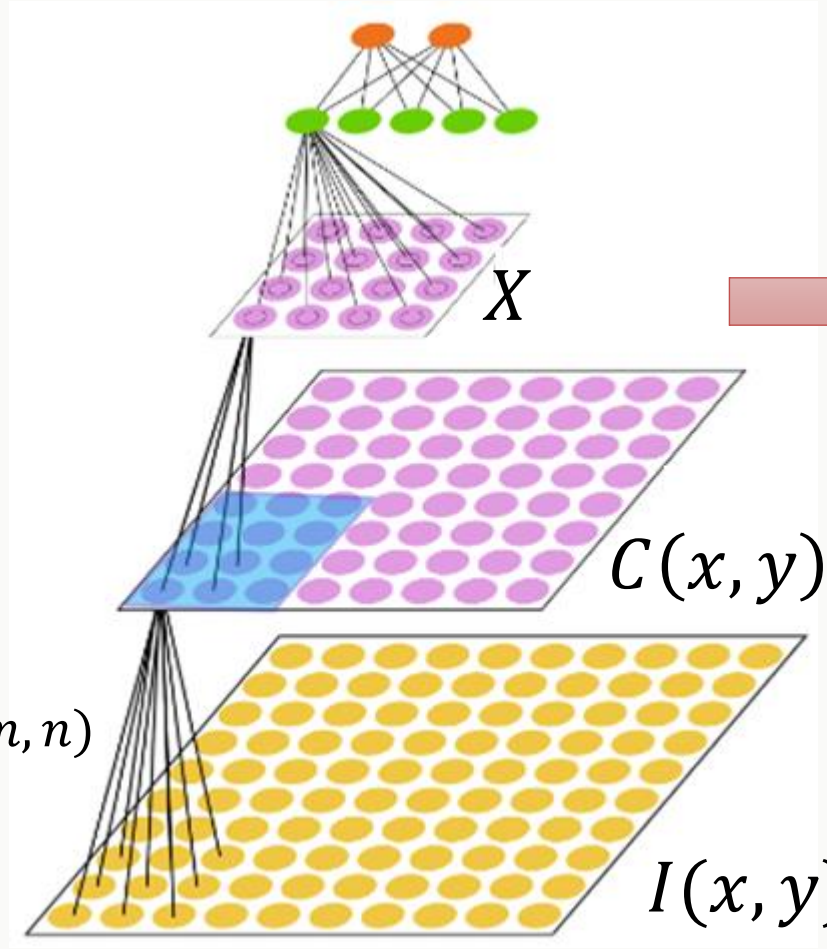
You Only Look Once: Unified, Real-Time Object Detection

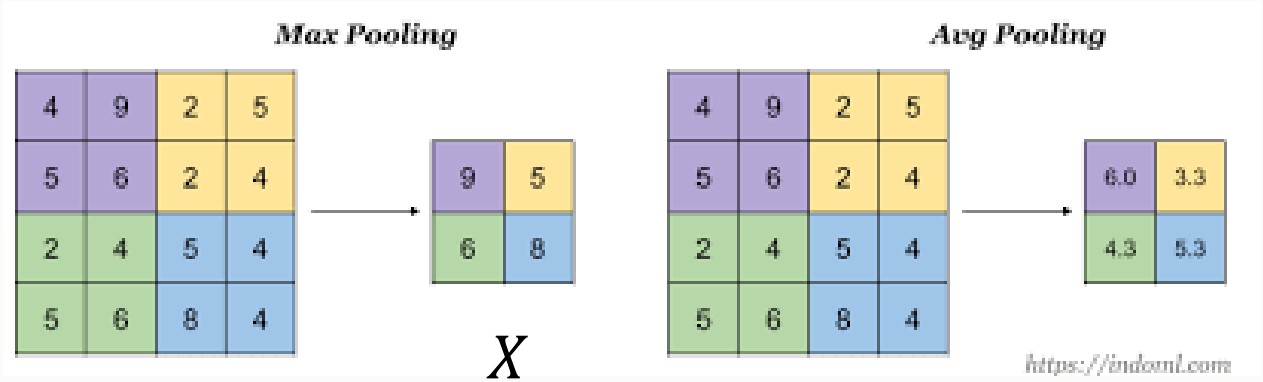
ARQUITECTURA BÁSICA: CONVOLUCIÓN



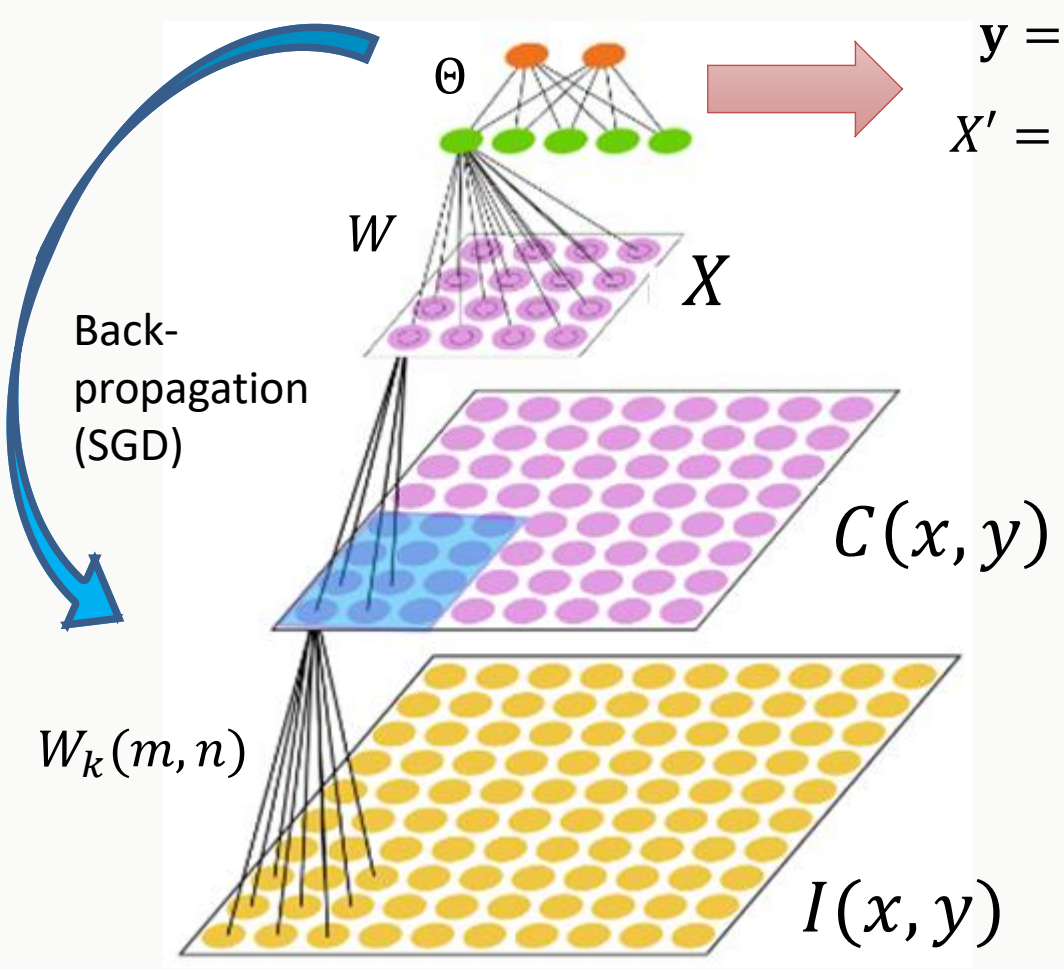
$$C(x,y) = g\left(\sum_{m=0}^{k-1}\sum_{n=0}^{k-1} I(x-m, y-n)\, W_k(m,n) + Bias\right)$$

ARQUITECTURA BÁSICA: POOLING





ARQUITECTURA BÁSICA: CLASIFICADOR



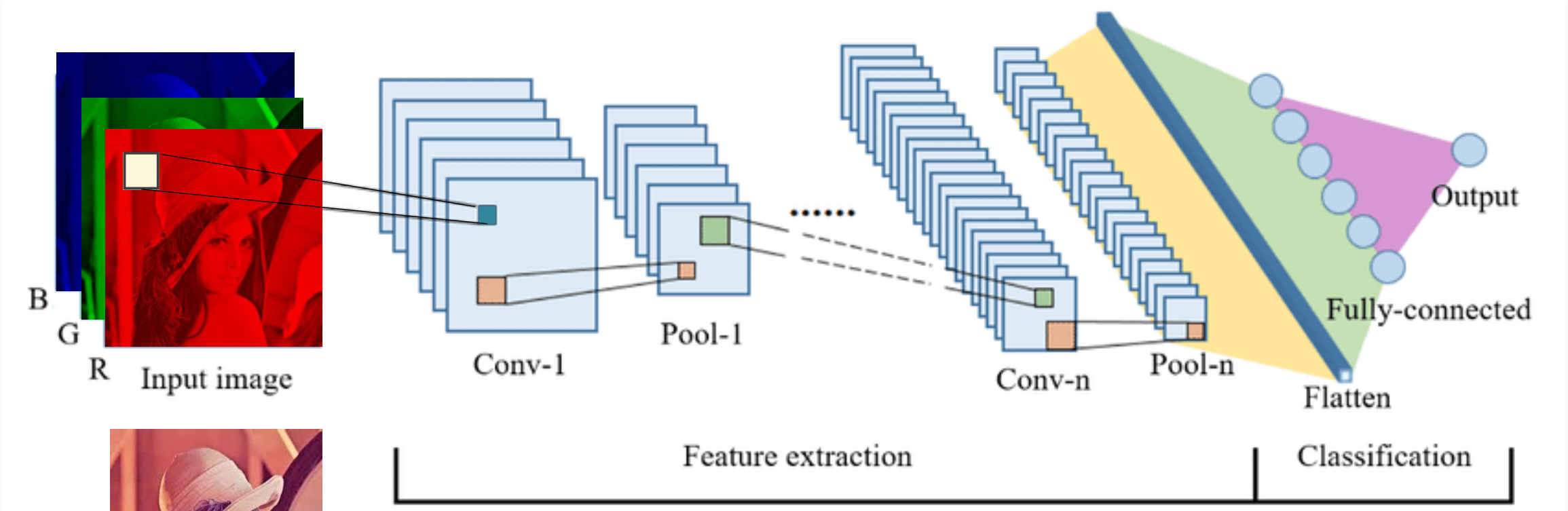
$$\mathbf{y} = h_\Theta(X')$$

$$X' = g(W^T X)$$

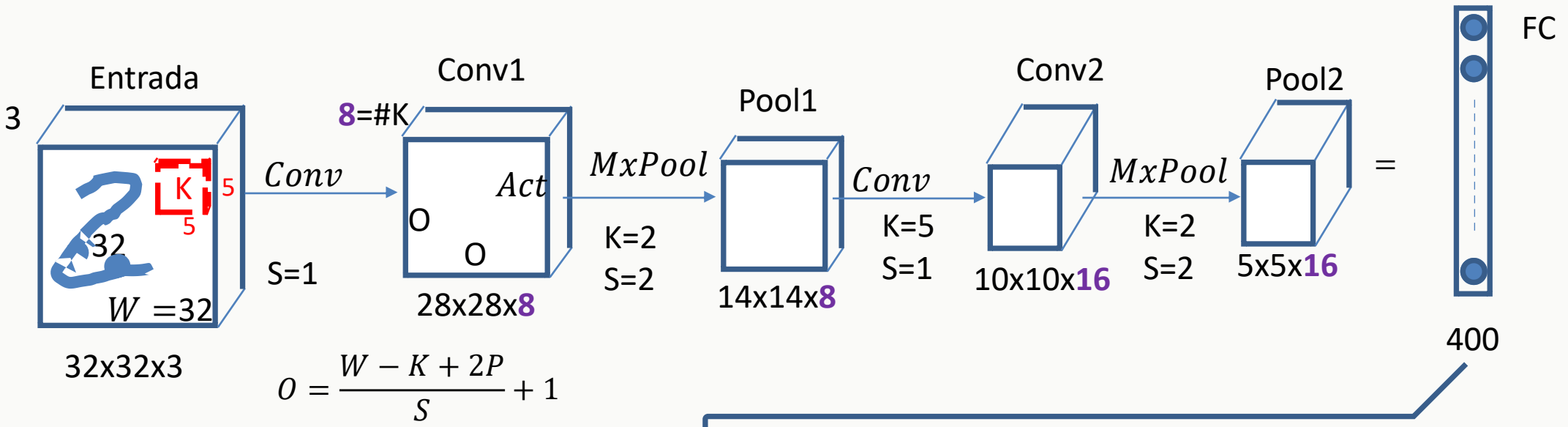
Prácticas comunes:

- Capa de entrada divisible entre 2
- Filtros pequeños; e.g., 3x3 a 9x9
- Zero padding
- Activación ReLu
- Stride = 2 vs Maxpooling
- BatchNorm vs regularización L2
- BatchNorm vs Dropout

ARQUITECTURA TÍPICA DE UNA CNN

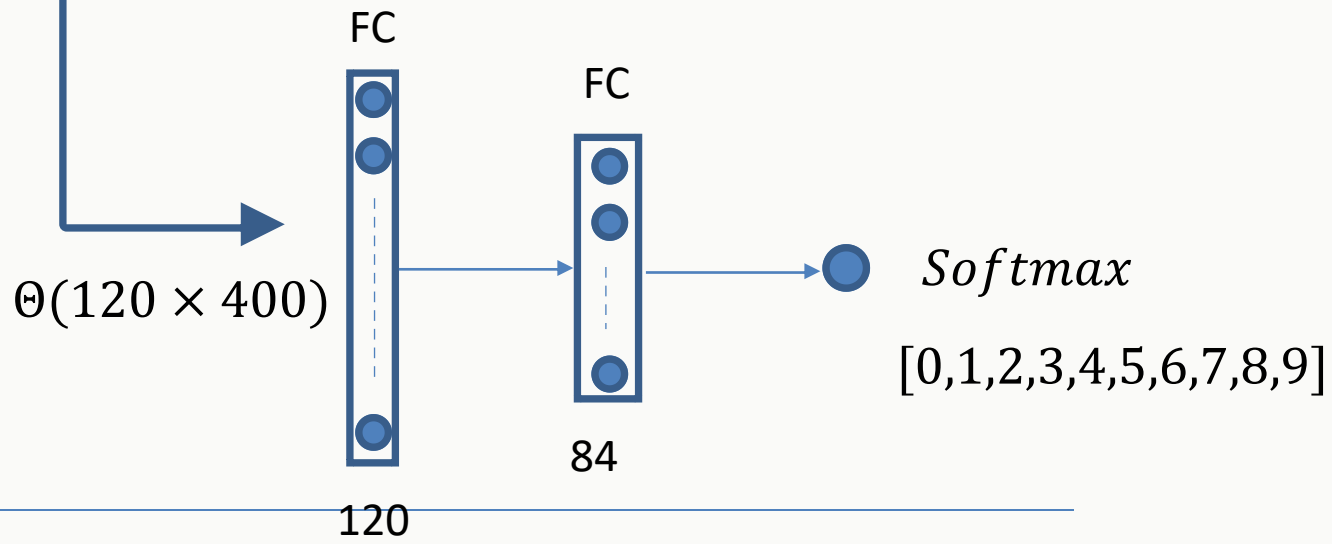


DIMENSIONES DE UNA CNN: EJEMPLO LeNet-5



$$O = \frac{W - K + 2P}{S} + 1$$

Donde P es el padding



NOTA:

- Reducción de la dimensión de las capas
- Aumento del número de canales

Número de parámetros

Definamos:

$W_c =$ Número de pesos en la capa Conv.

$B_c =$ Número de sesgos en la capa Conv.

$P_c =$ Número de parámetros en la capa Conv.

$K =$ Tamaño (ancho) de los kernels utilizados en la capa Conv.

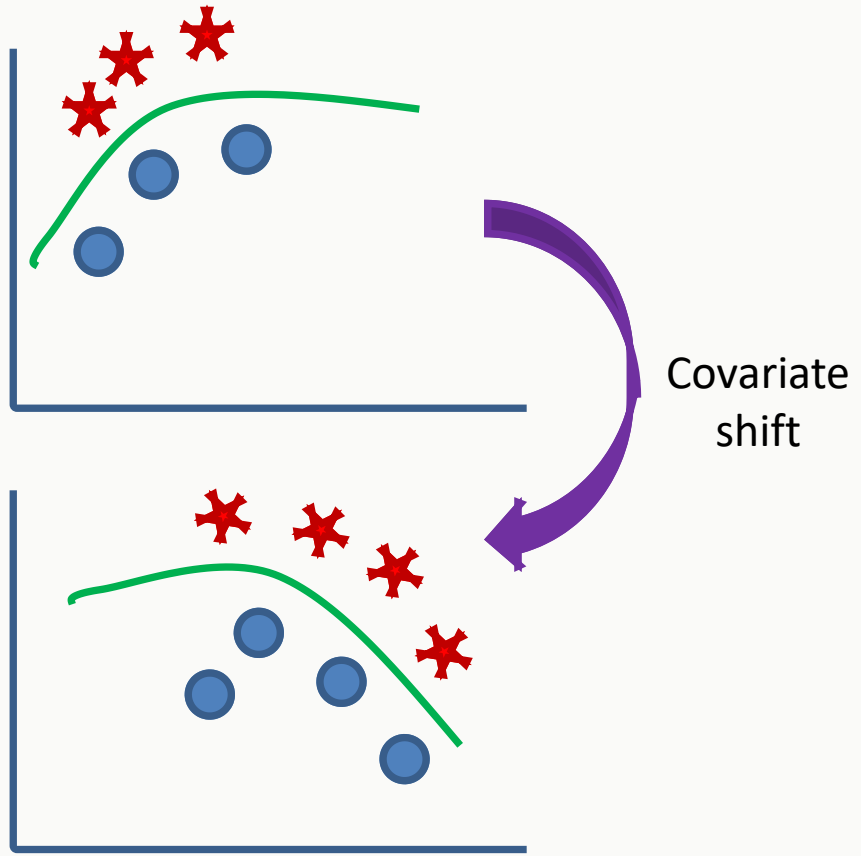
$N =$ Número de kernels.

$C =$ Número d canales de la imagen de entrada

$$W_c = K^2 \times C \times N$$
$$B_c = N$$
$$P_c = W_c + B_c$$

En la capa Conv., la profundidad de cada kernel es siempre igual al número de canales de la imagen de entrada. Así que cada kernel tiene $K^2 \times C$ parámetros, y hay N de esos kernels.

NORMALIZACIÓN DEL BATCH: “COVARIATE SHIFT”



NORMALIZACIÓN DEL BATCH

Entrada: Valores de x sobre un mini-batch: $\mathcal{B} = \{x_{1,\dots m}\}$;

Parámetros a ser aprendidos: γ, β

Salida: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \#\mathrm{mini-batch\ mean}$$

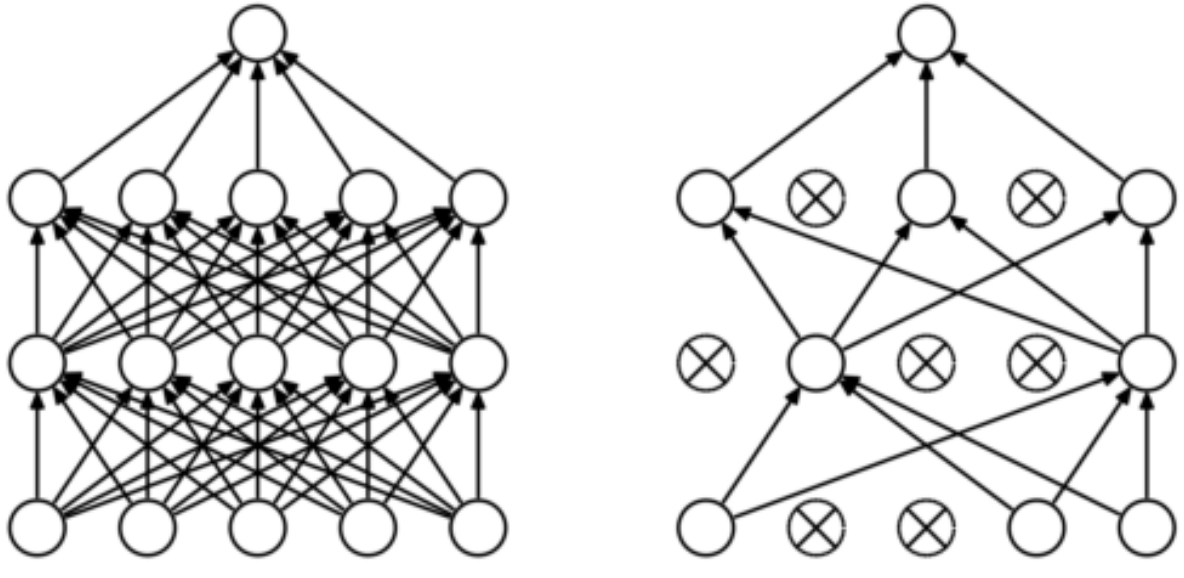
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \#\mathrm{mini-batch\ variance}$$

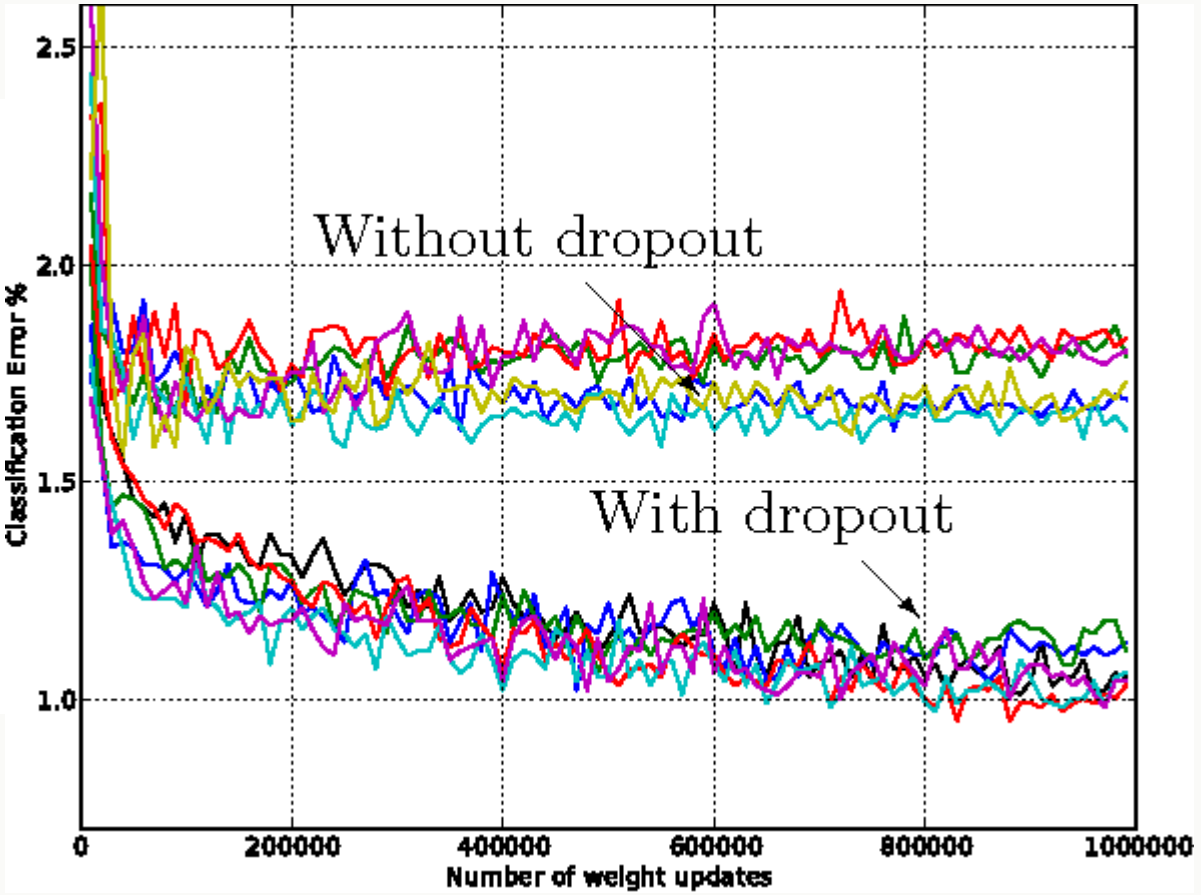
$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \#\mathrm{normaliza}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \#\mathrm{escala\ y\ traslada}$$

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37* (*ICML'15*). JMLR.org, 448–456.

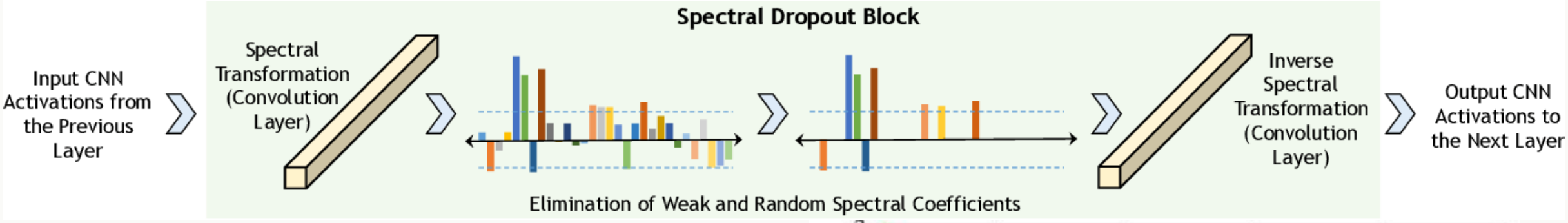
DROPOUT



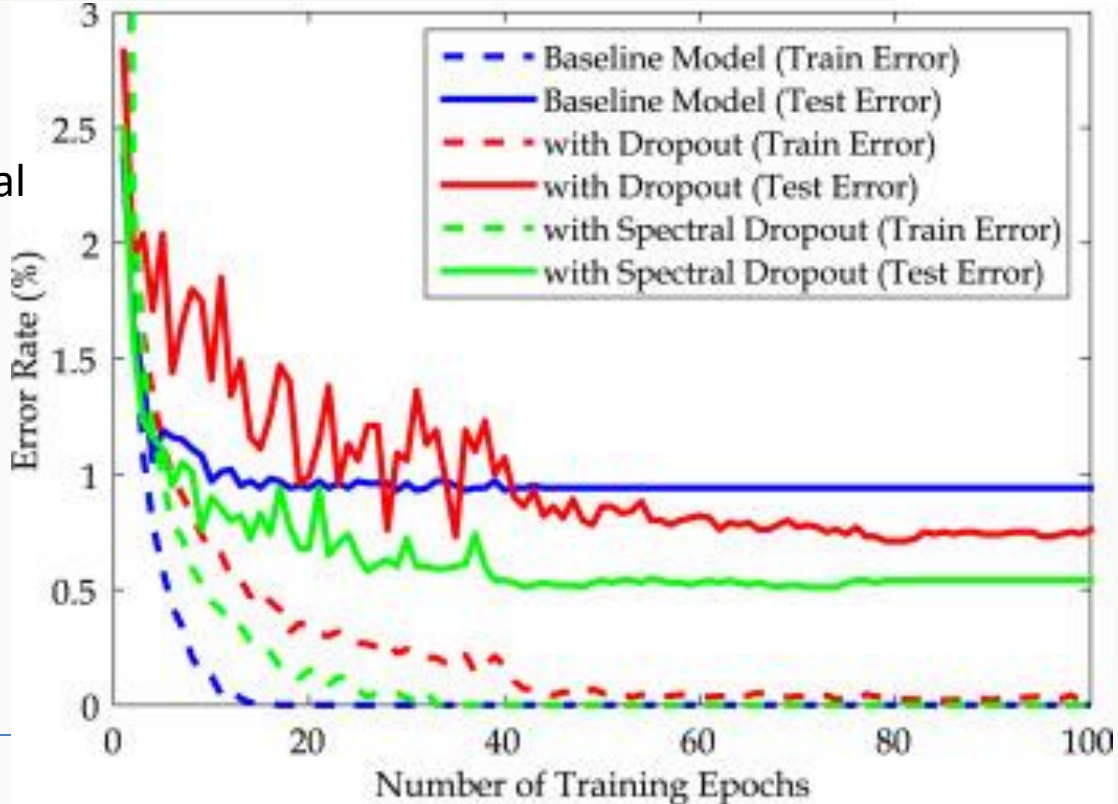


Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15, 1 (January 2014), 1929–1958.

SPECTRAL DROPOUT



Khan SH, Hayat M, Porikli F. Regularization of deep neural networks with spectral dropout. Neural Networks : the Official Journal of the International Neural Network Society. 2019 Feb;110:82-90. DOI: 10.1016/j.neunet.2018.09.009.

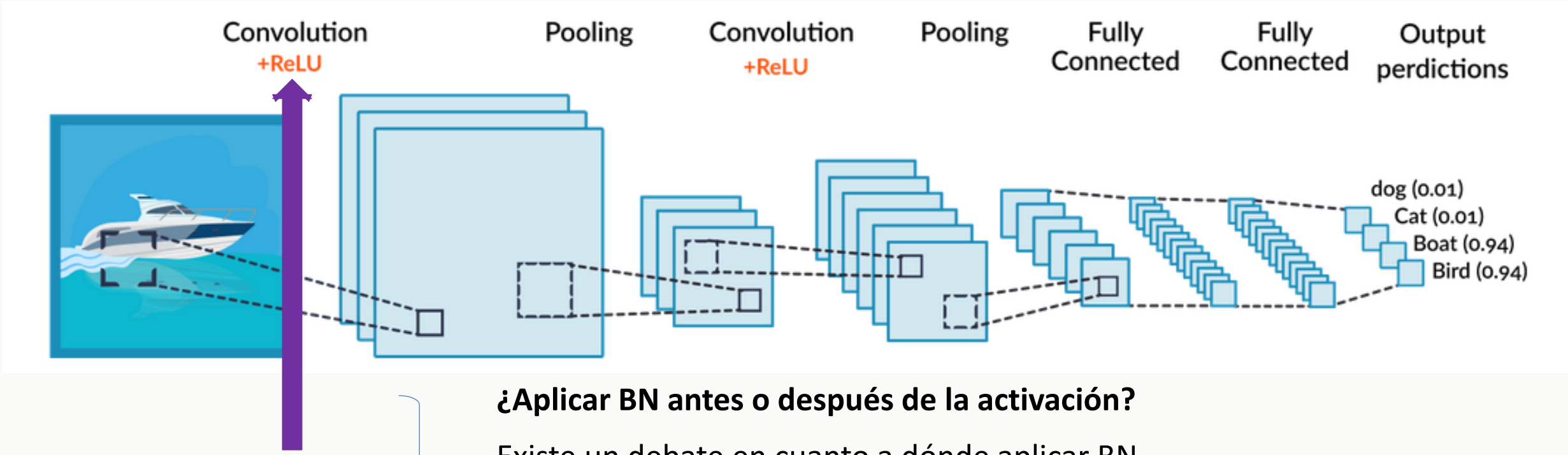


¿DROPOUT O BATCHNORM?

- "**Overfitting and long training time** are two fundamental challenges in multilayered neural network learning and deep learning in particular. **Dropout and batch normalization are two well-recognized approaches to tackle these challenges**.
- […]numerous research results have shown that **they have unique strengths to improve deep learning**.
- […] It is **not clear when** users should consider **using dropout and/or batch normalization**, and how they should be combined (or used alternatively) to achieve optimized deep learning outcomes.
- The empirical study showed that a **non-adaptive optimizer (e.g. SGD) can outperform adaptive optimizers**, but only **at the cost of a significant amount of training times** to perform hyperparameter tuning, while an adaptive optimizer (e.g. RMSProp) performs well without much tuning.
- Finally, it showed that dropout and batch normalization should be used in CNNs only with caution and experimentation (**when in doubt and short on time to experiment, use only batch normalization**)."

Garbin, C., Zhu, X. & Marques, O. Dropout *vs.* batch normalization: an empirical study of their impact to deep learning. *Multimed Tools Appl* **79,** 12777–12815 (2020). https://doi.org/10.1007/s11042-019-08453-9

ARQUITECTURA PROFUNDA



BATCH-NORM

¿Aplicar BN antes o después de la activación?

Existe un debate en cuanto a dónde aplicar BN.
El artículo original sugiere hacer BN antes de la activación (e.g. ReLu).
Pero hay otros autores que sugieren hacerlo después de la activación.

EJEMPLO: PROCESAMIENTO DE TEXTO

- A sentence is a sequence $w_1, w_2, \cdots, w_T$ of words $w_i \in V$
- Every word is represented by a feature vector $x_i \in \mathbb{R}^m$
- x_i **is randomly initialized**.
- Each sentence is represented by a matrix $(n \times m)$:

$$S = \begin{bmatrix} X \\ P \end{bmatrix}$$

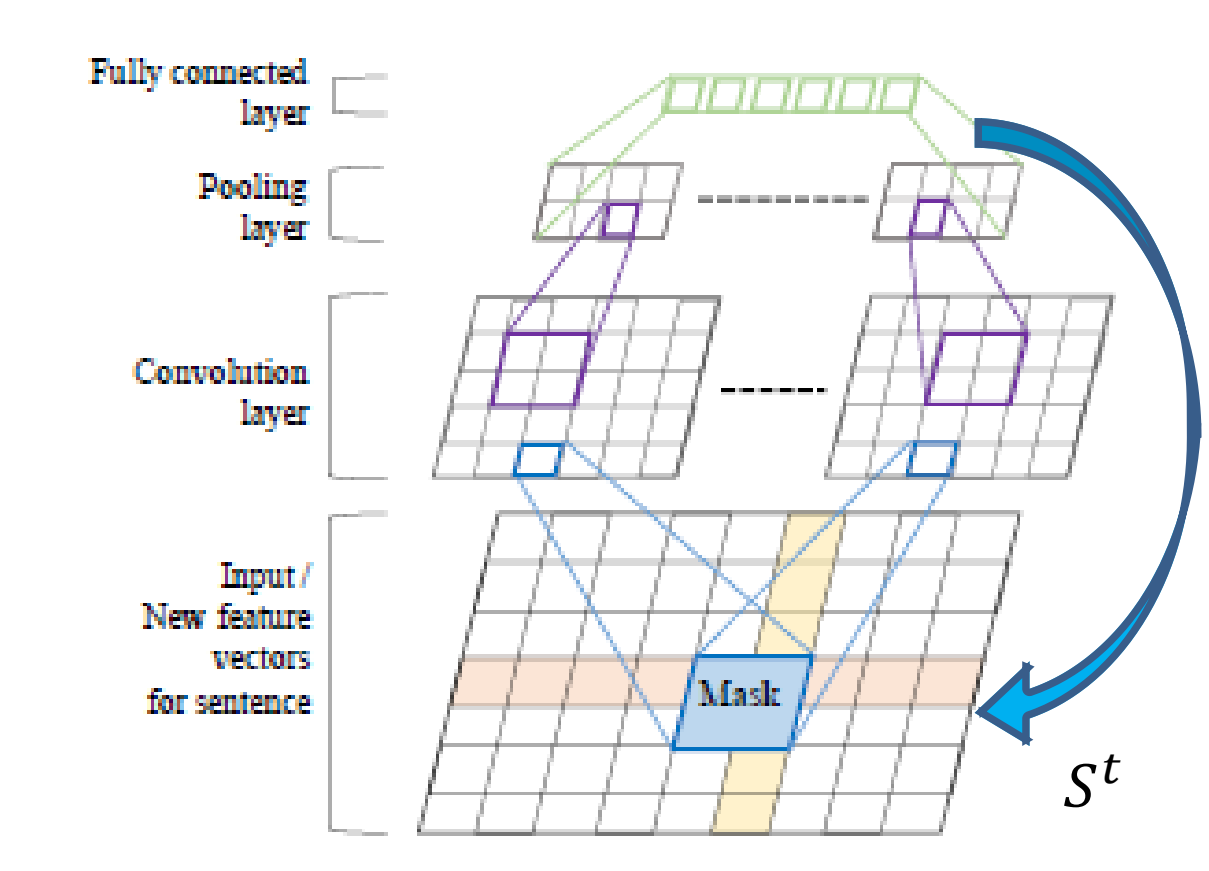
X is the feature matrix $(T \times m)$, P is a padding matrix $((\mathrm{n} - T) \times m)$

$$S = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \cdots & \vdots \\ x_{T,1} & x_{T,2} & \cdots & x_{T,m} \\ d & d & \cdots & d \\ \vdots & \vdots & \vdots & \vdots \\ d & d & \cdots & d \end{bmatrix} \begin{matrix} w_1 \\ w_2 \\ \vdots \\ w_T \\ p_1 \\ \\ p_2 \\ p_{n-T} \end{matrix}$$

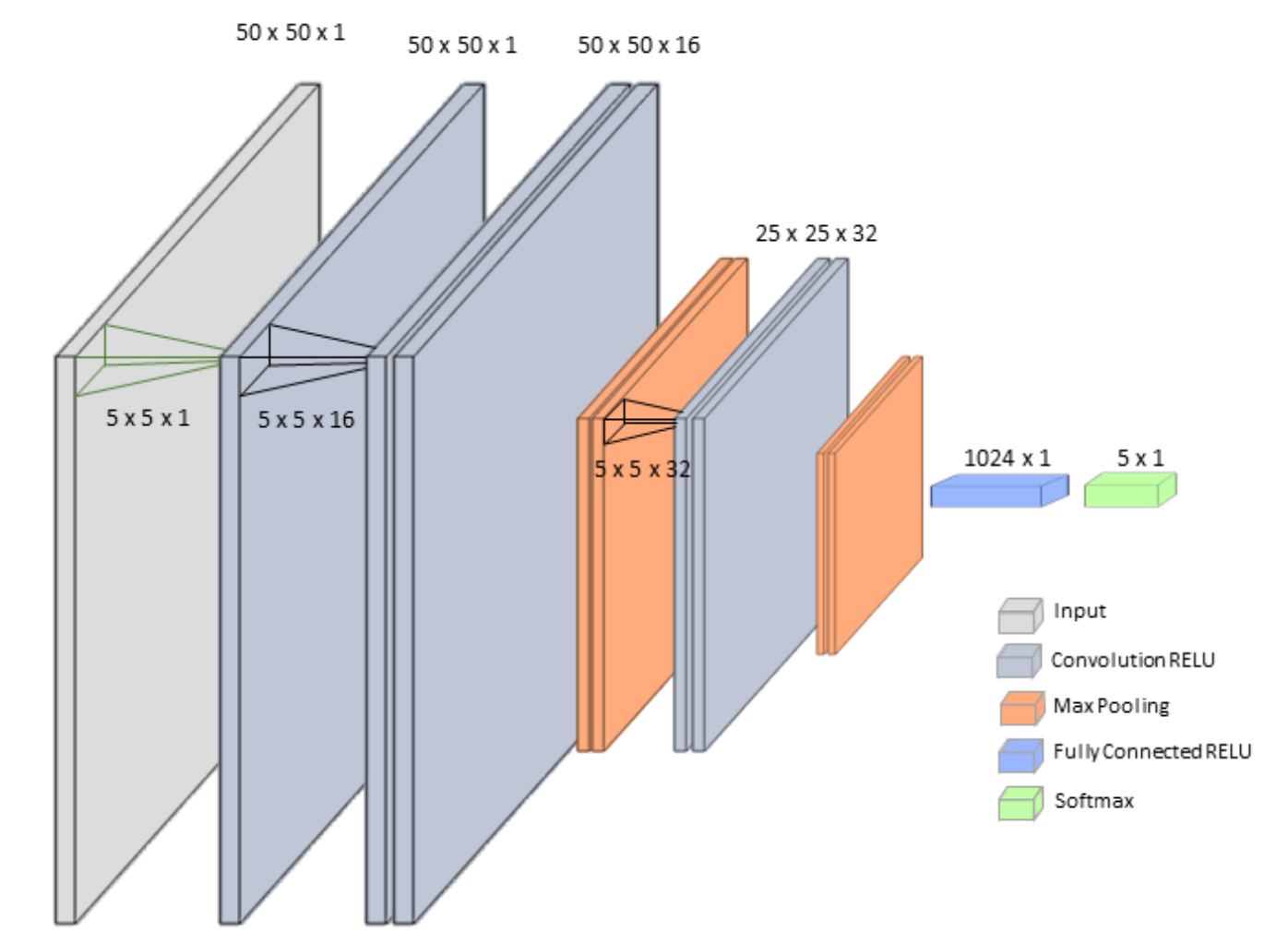
G. Vargas-Ocampo, E. Román-Rangel, y J. Hermosillo-Valadez (2018). *Learning word and sentence embeddings using a Convolutional Neural Network.* MCPR (2018) Puebla, México.

EJEMPLO: PROCESAMIENTO DE TEXTO

- Key idea: use the input layer as an embedding generative layer



EJEMPLO: PROCESAMIENTO DE TEXTO



- Input layer: a single convolutional filter with single filter of size 5x5 and ReLU activation.
- 2 convolutional-pooling layers, of 16 and 32 convolutional filters of size 5x5 (stride=1 and zero-padding).
- Max-pooling of size=2x2.
- Fully-connected layer of 1024 ReLU units.
- Softmax layer.
- Cross-entropy loss function, and Adam optimizer.

MÁSCARAS RESULTANTES