Clase20 IMA539

Alejandro Ferreira Vergara

November 22, 2022

1 Red Neuronal Convolucional (CNN)

Publicado en 1989 por Yann LeCun y sus colegas, donde propusieron una novedosa arquitectura de red neuronal para clasificar dígitos escritos a mano a partir de imágenes (Handwritten Digit Recognition with a Back-Propagation Network, Y LeCun, and others, 1989, publicado en la conferencia Neural Information Processing Systems.(NIPS)).

Son una familia de **modelos que se inspiran en el funcionamiento del córtex visual del cerebro humano** cuando se trata de reconocer objetos.

1.1 Aprendizaje Jerárquico de Características

Podemos considerar una red neuronal como un motor de **extracción de características.** Son capaces de aprender automáticamente las características más útiles para una tarea concreta, a partir de los datos en bruto.

- Las primeras capas extraen **características de bajo nivel.**

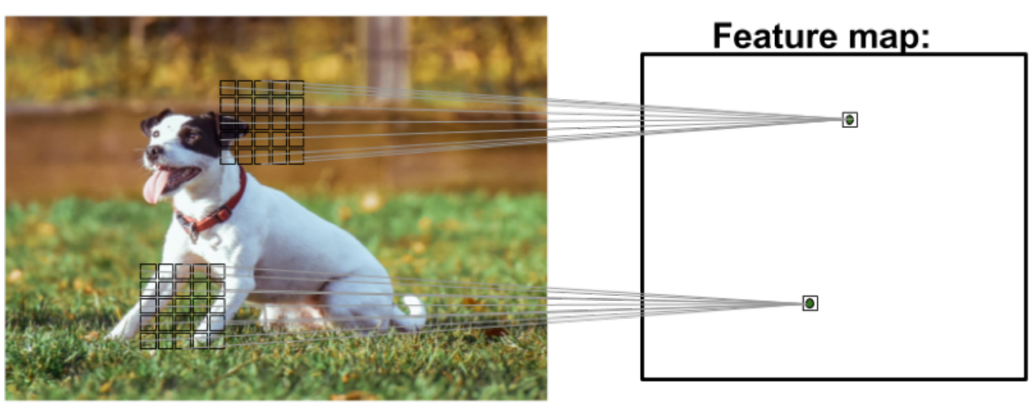
Las redes profundas multicapas y las CNNs combinan las características de bajo nivel en capas para formar **características de alto nivel.** A esto se le llama **Jerarquía de Características**.

Por ejemplo, si pensamos en imágenes, las características de bajo nivel podrían ser los bordes y las manchas, y se extraen en las primeras capas, luego se combinan para formar características de alto nivel, como las formas de los objetos (perro, gato, estrella de neutrones).

[1]:
```
from IPython.display import Image

Image(filename=r'clase20/20_1.png', width=600)
```

[1]:



- Este parche local de píxeles se denomina **campo receptivo local**.

Las CNNs suelen funcionar muy bien para tareas relacionadas con imágenes, y eso se debe principalmente a dos ideas importantes:

1) **Conectividad dispersa (o escasa)**: Un solo elemento del mapa de características está conectado sólo a un pequeño parche de píxeles.

2) **Compartir parámetros**: Se utilizan los mismos pesos para diferentes parches de la imagen de entrada.

Lo anterior implica que **el número de pesos (parámetros) de la red disminuye drásticamente**, y se observa una mejora en la **capacidad de captar características destacadas**.

Normalmente, las CNNs se componen de varias **capas convolucionales (conv)** y capas de submuestreo, también conocido como **capas pooling o de agrupación (P)**. Estas, van seguidas de una o más **capas totalmente conectadas o Fully Connected (FC)** al final.

- Las capas pooling no tienen parámetros que aprender.

1.2 Convolución Discreta en una Dimensión

Una **convolución discreta** o simplemente **convolución** es la operación fundamental en una CNN.

Si x y w son tensores de orden 1 (por ejemplo, vectores de $\mathbb{R}^n$ y $\mathbb{R}^m$, respectivamente, con $m \leq n$), denotamos por $x * w$ a la **convolución entre x y w**. El vector x es la entrada (a veces llamada **señal**) y w se llama el **filtro** o **kernel**.

Matemáticamente, la convolución discreta en una dimensión entre x y w está definida como:

$$y = x * w \to y[i] = \sum_{k=-\infty}^{+\infty} x[i-k]w[k]$$

- Los corchetes [] denotan la indexación de los elementos del vector.
- El índice i recorre cada elemento del vector de salida y. **¿$-\infty$, $+\infty$, indexación negativa?**

Para calcular correctamente la suma mostrada en la fórmula anterior, se hace la suposición de que fuera del rango de características de x y w, los valores están llenos de ceros.

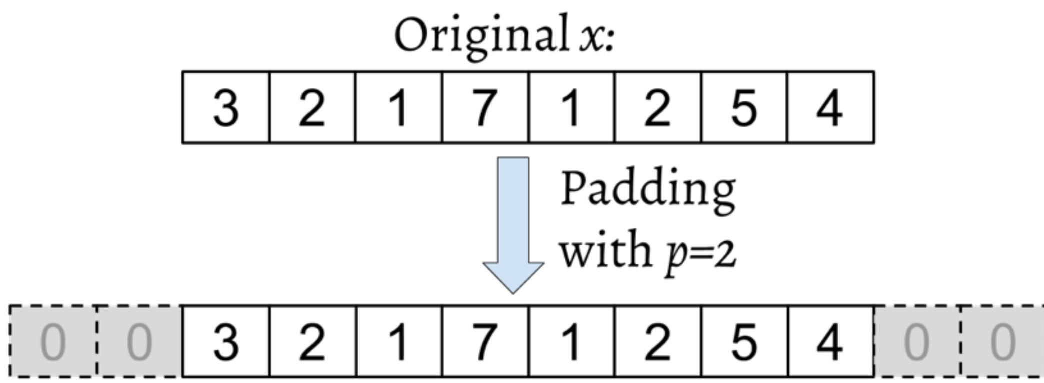
Esto dará como resultado un vector de salida y que tiene un tamaño infinito con muchos ceros.

Como esto no es útil en situaciones prácticas, x se rellena sólo con un número finito de ceros.

Este último proceso se denomina **relleno de ceros (zero-padding) o simplemente relleno**. Aquí, el número de ceros rellenados en cada lado se denota por p.

[2]:
```
Image(filename=r'clase20/20_2.png', width=500)
```

[2]:



Si $x \in \mathbb{R}^n$ y $w \in \mathbb{R}^m$, con $m \leq n$, entonces el **vector acolchado** x^p tendrá tamaño igual a $n + 2p$.

De esta forma, una expresión más práctica para la **convolución discreta en una dimensión entre x y w** es:

$$y = x * w \rightarrow y[i] = \sum_{k=0}^{k=m-1} x^p[i + m - k]w[k]$$

El punto importante a tener en cuenta aquí es que, en la suma anterior, x **y** w **están indexados en diferentes direcciones**.

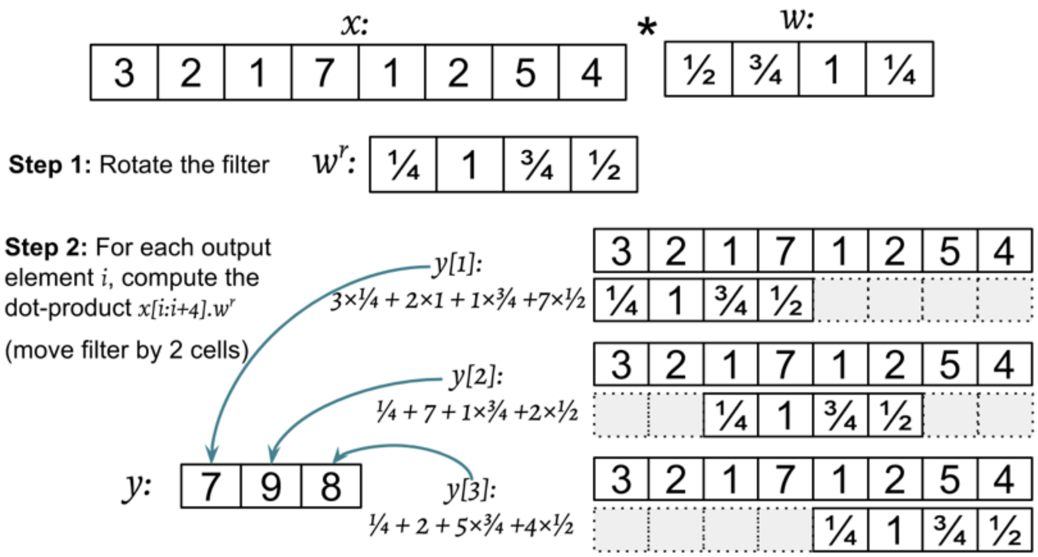
Si volteamos el kernel w y lo denotamos por w^r, entonces la operación dentro de la suma sería simplemente un producto punto.

En este sentido, se calcula el **producto punto entre** $x[i : 1+m]$ **con** w^r **y se obtiene** $y[i]$, donde $x[i : 1 + m]$ es simplemente un parche de x, de tamaño m.

Esta operación se repite como en un enfoque de ventana deslizante para obtener todos los elementos de salida.

[3]:
```
Image(filename=r'clase20/20_3.png', width=600)
```

[3]:



Al número de celdas que se corre (o desplaza) el kernel, se le llama **Strides** y es un hiperparámetro, al igual que el relleno.

Técnicamente, el relleno se puede aplicar con cualquier $p \geq 0$. Dependiendo de la elección de p, las celdas de los límites pueden ser tratadas de forma diferente a las celdas situadas en el centro del vector x.

En la práctica, existen tres modos de relleno que se utilizan habitualmente: **completo**, **igual** y **válido** (**full, same y valid**).

1) En el modo **completo (full)**, el parámetro de acolchado p se establece en $p = m - 1$. El modo completo aumenta las dimensiones de la salida.

2) El modo **igual (same)** se suele utilizar si se quiere que el tamaño de la salida sea el mismo que el del vector de entrada x. En este caso, el parámetro de acolchado p se calcula en función del tamaño del kernel, junto con el requisito de que el tamaño de la entrada y el de la salida sean iguales.

3) El cálculo de una convolución en el modo **válido (valid)** se refiere al caso en que $p = 0$ (sin relleno).

El **tamaño de la salida de una convolución** estará dada por el número total de veces que desplazamos el kernel w a lo largo del vector de entrada. Supongamos que el vector de entrada tiene un tamaño n y el kernel es de tamaño m, con $m \leq n$. Entonces, el tamaño de la salida resultante de $x * w$, con relleno p y zancada s se determina como:

$$o = \left\lfloor \frac{n + 2p - m}{s} \right\rfloor + 1$$

[9]:
```
import numpy as np
```

```
def conv1d(x, w, p=0, s=1):
    w_rot = np.array(w[::-1])
    x_padded = np.array(x)
    if p > 0:
        zero_pad = np.zeros(shape=p)
        x_padded = np.concatenate([zero_pad, x_padded, zero_pad])
    res = []
    for i in range(0, int(len(x)/s),s):
        res.append(np.sum(x_padded[i:i+w_rot.shape[0]] * w_rot))
    return np.array(res)

## Testing:
x = [1, 3, 2, 4, 5, 6, 1, 3]
w = [1, 0, 3, 1, 2]
print('Conv1d Implementation: ', conv1d(x, w, p=2, s=1))
print('Numpy Results:         ', np.convolve(x, w, mode='same'))
```

```
Conv1d Implementation:  [ 5. 14. 16. 26. 24. 34. 19. 22.]
Numpy Results:          [ 5 14 16 26 24 34 19 22]
```

1.3 Convolución Discreta en dos Dimensiones

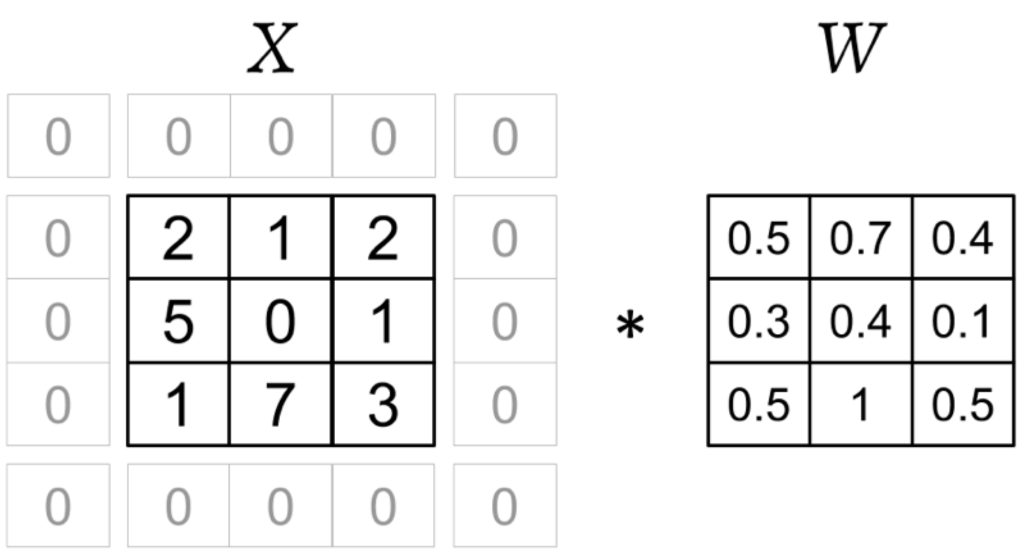
Para una matriz $X_{n_1 \times n_2}$ y la matriz de kernel $W_{m_1 \times m_2}$, donde $m_1 \leq n_1$ y $m_2 \leq n_2$, denotamos a la matriz $Y = X * W$ como el resultado de la **convolución** $2D$ **de** X **con** W.

Matemáticamente, lo anterior se define como:

$$Y = X * W \rightarrow Y[i,j] = \sum_{k_1=-\infty}^{+\infty} \sum_{k_2=-\infty}^{+\infty} X[i-k_1, j-k]W[k_1,k_2]$$

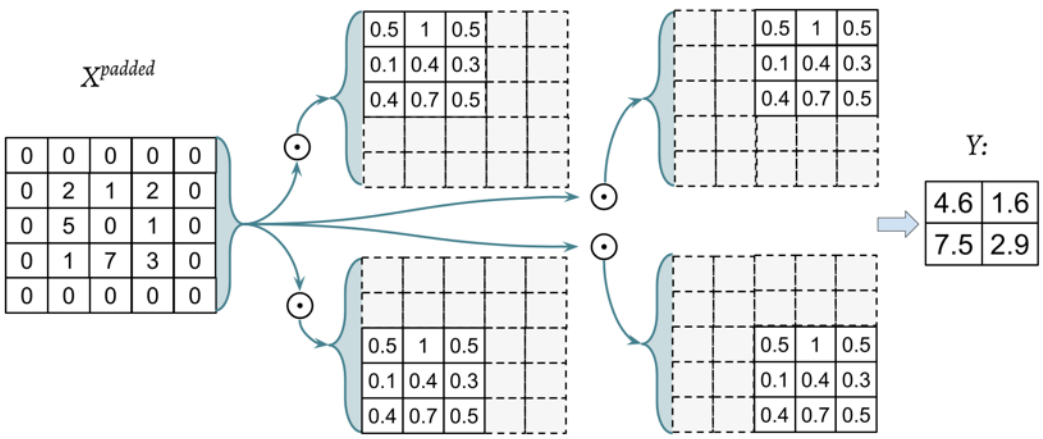
```
[4]: Image(filename=r'clase20/20_4.png', width=400)
```

[4]:



```
[5]: Image(filename=r'clase20/20_5.png', width=650)
```

[5]:



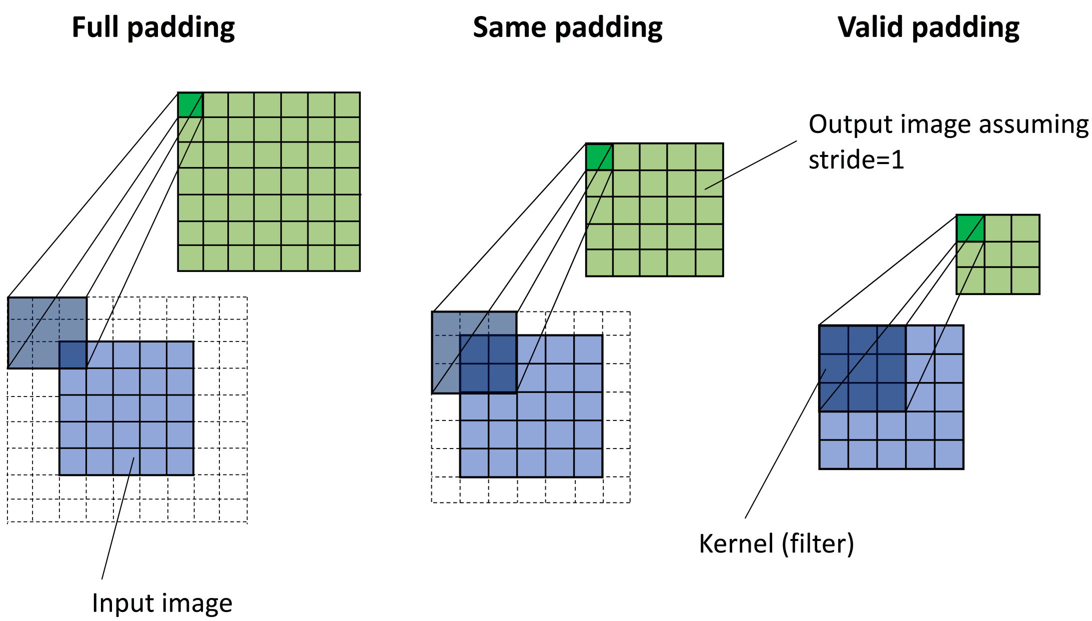
0	0	0	0	0
0	2	1	2	0
0	5	0	1	0
0	1	7	3	0
0	0	0	0	0

0.5	1	0.5
0.1	0.4	0.3
0.4	0.7	0.5

4.6	1.6
7.5	2.9

```
[6]: Image(filename=r'clase20/20_8.png', width=600)
```

[6]:




```
[10]: import scipy.signal

def conv2d(X, W, p=(0,0), s=(1,1)):
    W_rot = np.array(W)[::-1,::-1]
```

```python
    X_orig = np.array(X)
    n1 = X_orig.shape[0] + 2*p[0]
    n2 = X_orig.shape[1] + 2*p[1]
    X_padded = np.zeros(shape=(n1,n2))
    X_padded[p[0]:p[0] + X_orig.shape[0],p[1]:p[1] + X_orig.shape[1]] = X_orig

    res = []
    for i in range(0, int((X_padded.shape[0]-W_rot.shape[0])/s[0])+1, s[0]):
        res.append([])
        for j in range(0, int((X_padded.shape[1]-W_rot.shape[1])/s[1])+1, s[1]):
            X_sub = X_padded[i:i+W_rot.shape[0], j:j+W_rot.shape[1]]
            res[-1].append(np.sum(X_sub * W_rot))
    return(np.array(res))

X = [[1, 3, 2, 4], [5, 6, 1, 3], [1 , 2,0, 2], [3, 4, 3, 2]]
W = [[1, 0, 3], [1, 2, 1], [0, 1, 1]]
print('Conv2d Implementation: \n',conv2d(X, W, p=(1,1), s=(1,1)))

print('Scipy Results:         \n',scipy.signal.convolve2d(X, W, mode='same'))
```

```
Conv2d Implementation:
 [[11. 25. 32. 13.]
 [19. 25. 24. 13.]
 [13. 28. 25. 17.]
 [11. 17. 14.  9.]]
Scipy Results:
 [[11 25 32 13]
 [19 25 24 13]
 [13 28 25 17]
 [11 17 14  9]]
```

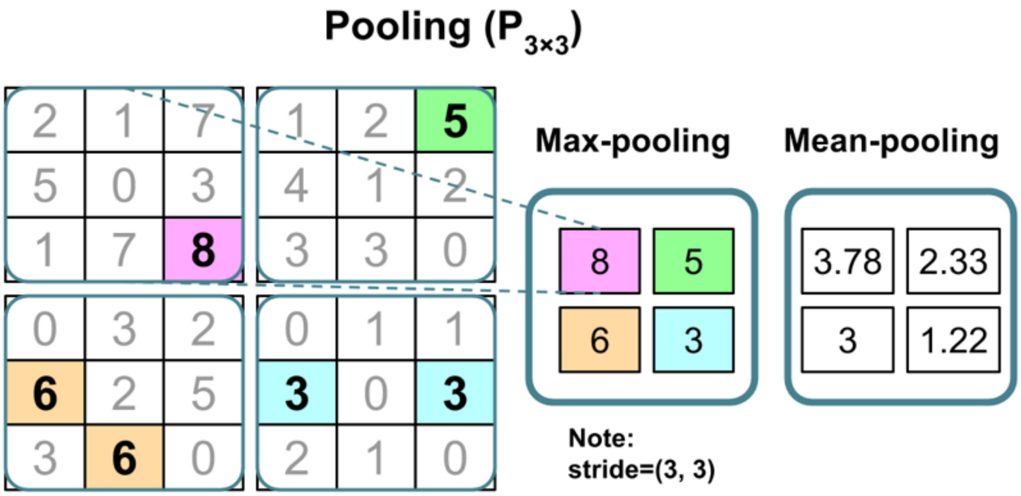
1.4 Capas Submuestreo

En las CNNs, el submuestreo se aplica normalmente en dos formas de operaciones de pooling: **max-pooling** y **average-pooling**.

Para el caso de convoluciones $2D$, las capas pooling se pueden denotar como $P_{n_1 \times n_2}$. Aquí, el subíndice determina el **tamaño del vecindario** (en el caso de imágenes, el número de píxeles adyacentes en cada dimensión), donde se realiza la operación *mximo* o *media*.

[7]:
```python
Image(filename=r'clase20/20_6.png', width=500)
```

[7]:



La agrupación **disminuye el tamaño de las características**, lo que se traduce en una mayor eficiencia computacional.

Además, la reducción del número de características puede **reducir también el grado de sobreajuste**.

La agrupación max-pooling introduce un tipo de invariancia local.

1.5 Múltiples canales de entrada

Una muestra de entrada a una capa convolucional puede contener una o más matrices o arreglos $2D$ con dimensiones $N_1 \times N_2$. Estas matrices $N_1 \times N_2$ se denominan **canales**. Por lo tanto, el uso de múltiples canales como entrada a una capa convolucional nos obliga a utilizar un tensor de rango 3: $X_{N_1 \times N_2 \times C_{in}}$, donde C_{in} **es el número de canales de entrada**.

Ahora, realizamos la operación de convolución para cada canal por separado y luego sumamos los resultados utilizando la suma matricial.

La convolución asociada a cada canal tiene su propia matriz kernel, $W[:, :, c]$.

De esta forma, la **salida de una capa de convolución** se calcula como:

$$\begin{array}{c} Dada\ una\ muestra\ X_{n_1 x n_2 x c_{in}} \\ una\ Matriz\ Kernel\ W_{m_1 x m_2 x c_{in}} \\ y\ coeficiente\ sesgo\ b \end{array} \Rightarrow \begin{cases} Y^{conv} = \sum_{c=1}^{C_{in}} W[:, :, c] * X[:, :, c] \\ pre-activation: \quad A = Y^{conv} + b \\ Featuremap: \quad H = \phi(A) \end{cases}$$

El resultado final, H, se llama **mapa de características**.

Normalmente, **una capa convolucional de una CNN tiene más de un mapa de características**. Si utilizamos múltiples mapas de características, el tensor núcleo se convierte en cuatridimensional:

$$width \times height \times C_{in} \times C_{out}$$

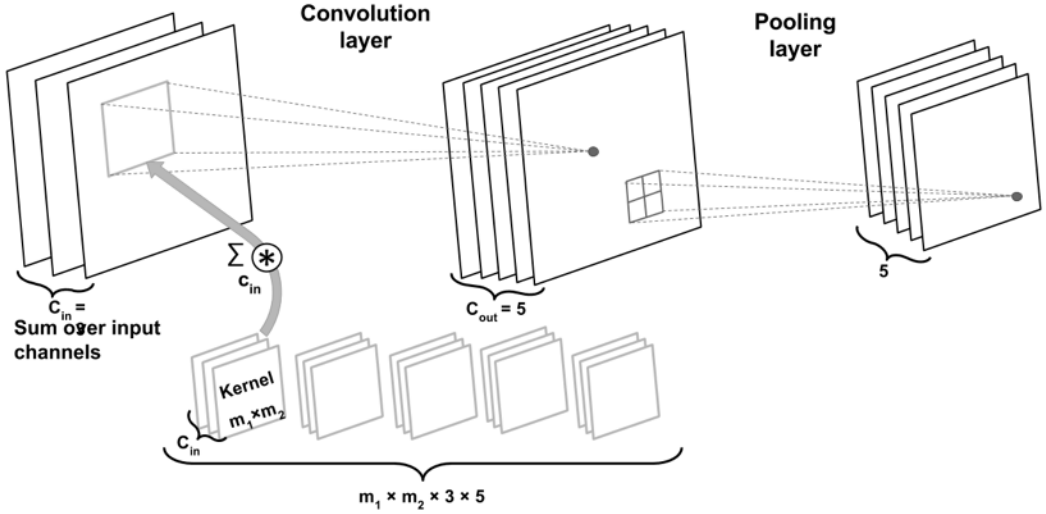
- C_{in} es el número de canales de entrada
- C_{out} es el número de mapas de características de salida.

Por lo tanto, incluyendo el número de mapas de características de salida en la fórmula anterior:

$$\begin{array}{c} Dada\ una\ muestra\ X_{n_1 x n_2 x C_{in}} \\ una\ Matriz\ Kernel\ W_{m_1 x m_2 x C_{in} x C_{out}} \\ y\ coeficiente\ sesgo\ b_{C_{out}} \end{array} \Rightarrow \begin{cases} Y^{conv}[:,:,k] = \sum_{c=1}^{C_{in}} W[:,:,c,k] * X[:,:,c] \\ A[:,:,k] = Y^{conv}[:,:,k] + b[k] \\ H[:,:,k] = \phi(A[:,:,k]) \end{cases}$$

[8]:
```
Image(filename=r'clase20/20_7.png', width=600)
```

[8]:



[ ]: