Sesión 2:

Redes Neuronales Convolucionales y Redes Residuales para la clasificación de imágenes

---

Elvin Mark Munoz Vega

Table of contents

Intro

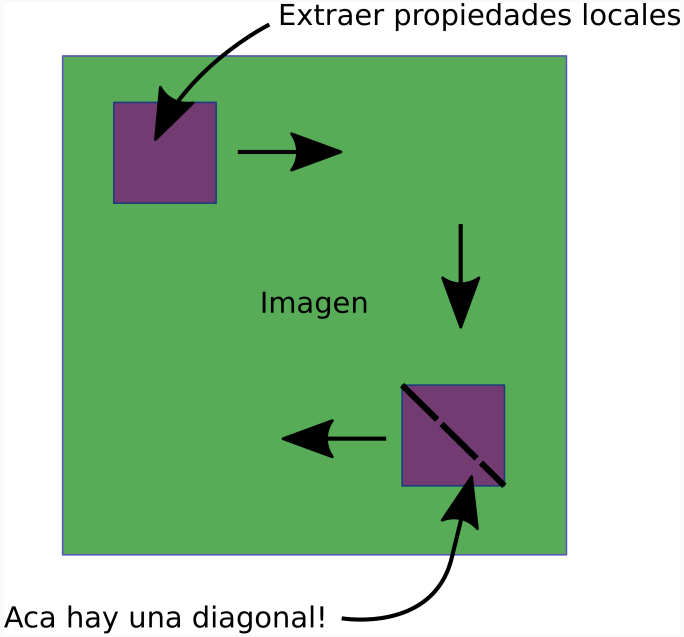
Objetivos

1. Aprender sobre convolución y su importancia en computer vision.
2. Aprender sobre las diferentes capas muy utilizadas en computer vision: Pooling layers, Dropout layers, BatchNorm Layers.
3. Entender el problema del desvanecimiento de la gradiente y como las redes residuales ayudan a solucionar este problema.
4. Aprender sobre la clasificación de imágenes.

Convolución

Convolución (Correlación)

Toda figura en una imagen esta formada de ciertas formas básicas (lineas, diagonales, etc). Reconociendo estas formas básicas podriamos clasificar correctamente la figura que se encuentra en la imagen. Por ejemplo, sabemos que un triangulo esta formado por 3 esquinas, si en la imagen ubicamos estas 3 esquinas, podriamos intuir que en la imagen hay un triangulo. La operación de convolución proporciona esta capacidad a las redes neuronales.



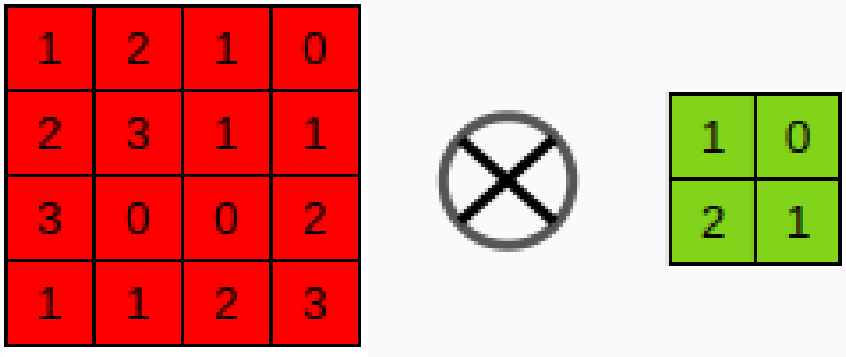
Convolución (Correlación)

Sea la imagen $I = [I_{ij}] \in \mathcal{R}^{H \times W}$ con ancho (width) W y alto (height) H y el filtro $F = [F_{ij}] \in \mathcal{R}^{K \times K}$ con dimension $K \times K$. La convolución de I con F se define como sigue.

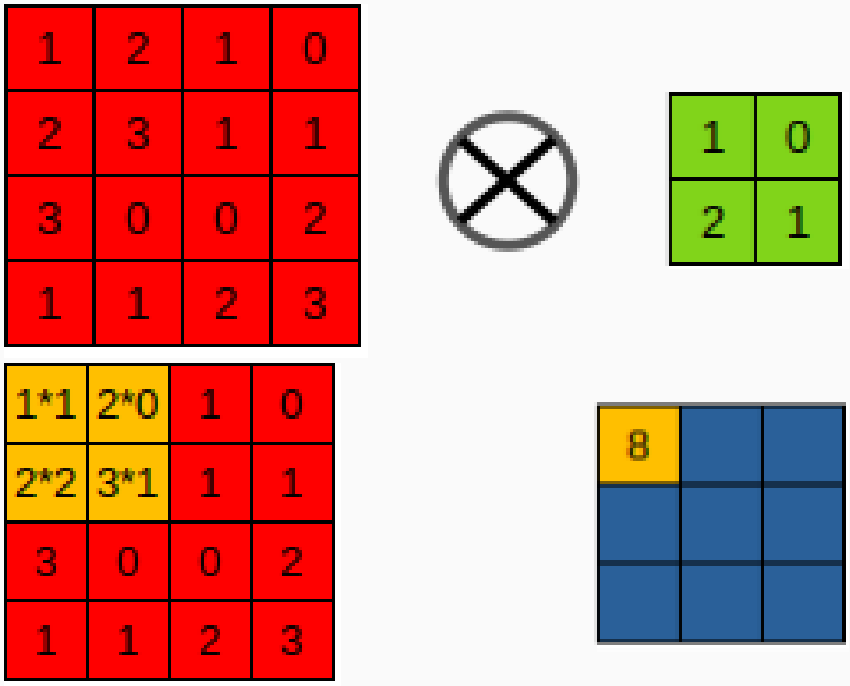
$O = I \bigotimes F = [O_{ij}] \in \mathcal{R}^{(H-K)\times(W-K)}$ donde cada elemento se calcula de la siguiente forma:

$$O_{ij} = \sum_{p=1}^{K} \sum_{q=1}^{K} I_{i+p,j+q} F_{p,q}$$

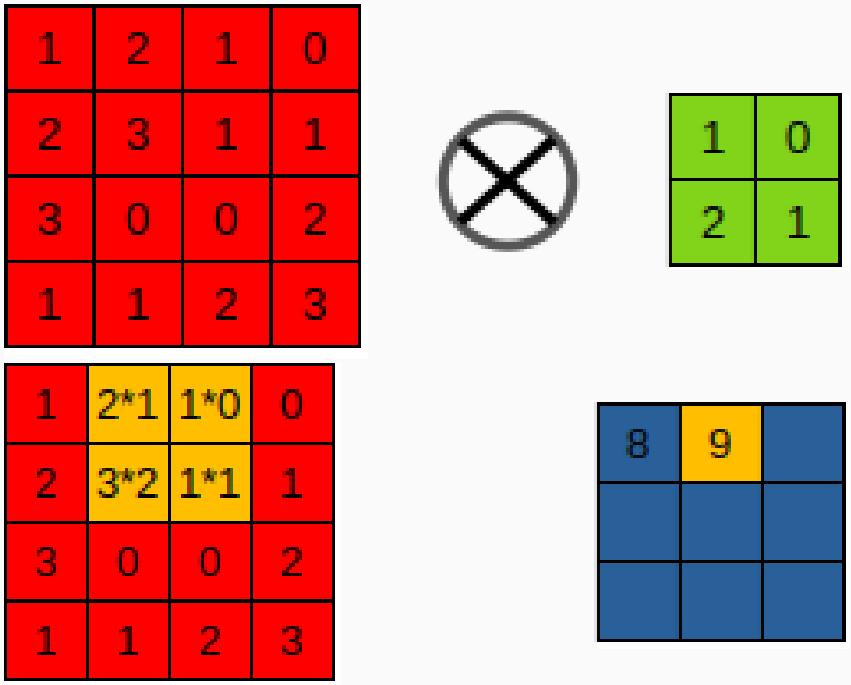
Convolución: Ejemplo



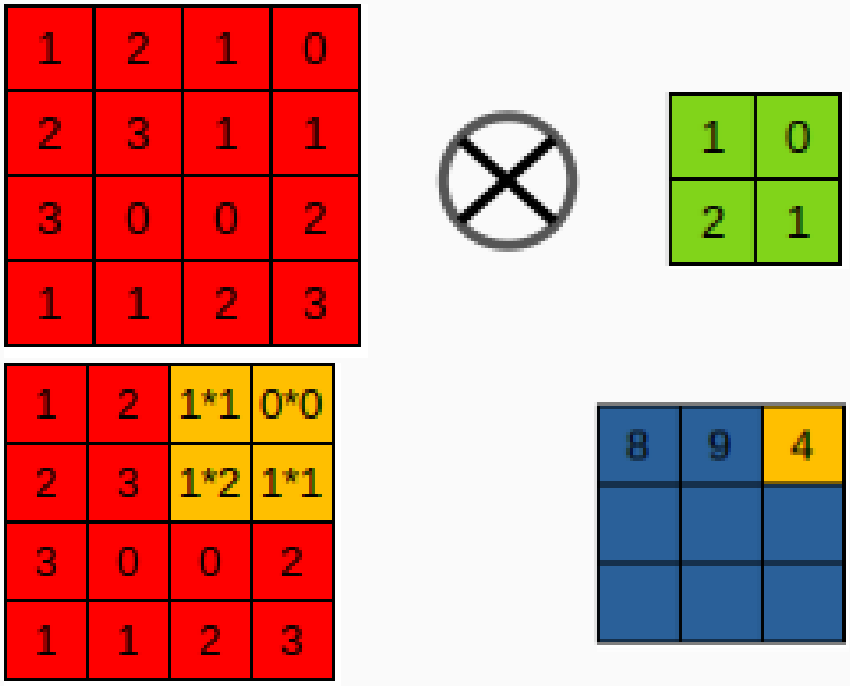
Convolución: Ejemplo



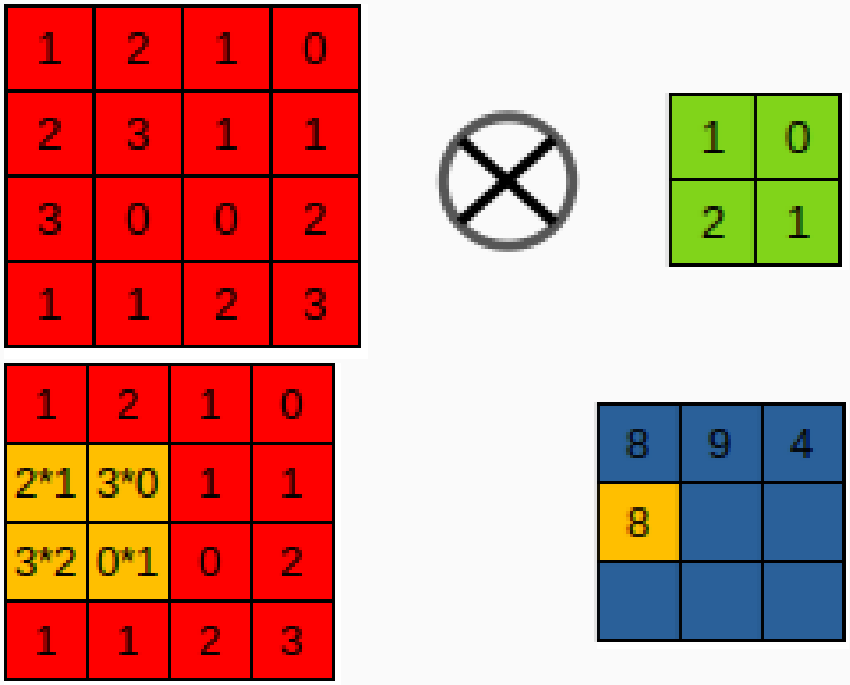
Convolución: Ejemplo



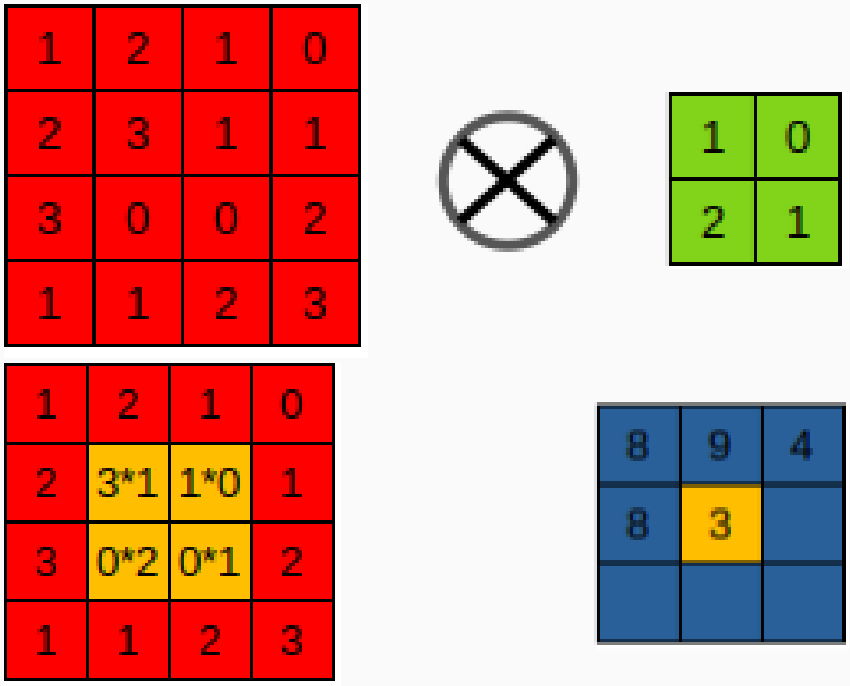
Convolución: Ejemplo



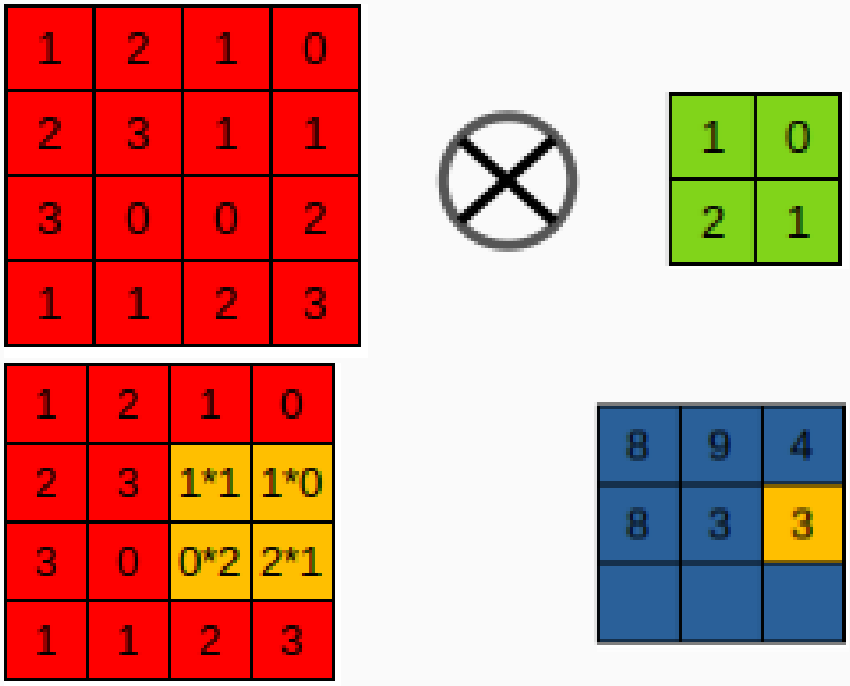
Convolución: Ejemplo



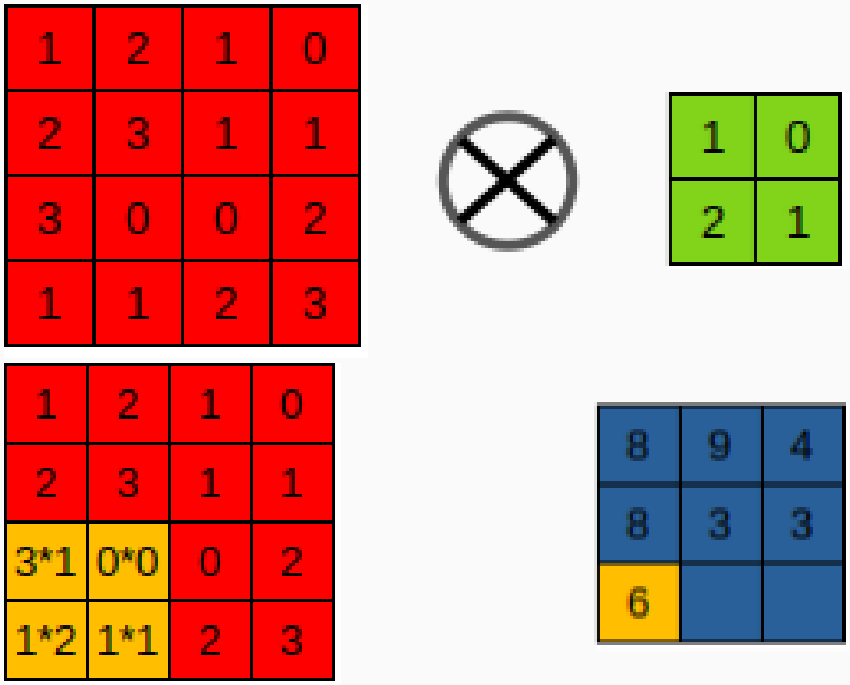
Convolución: Ejemplo



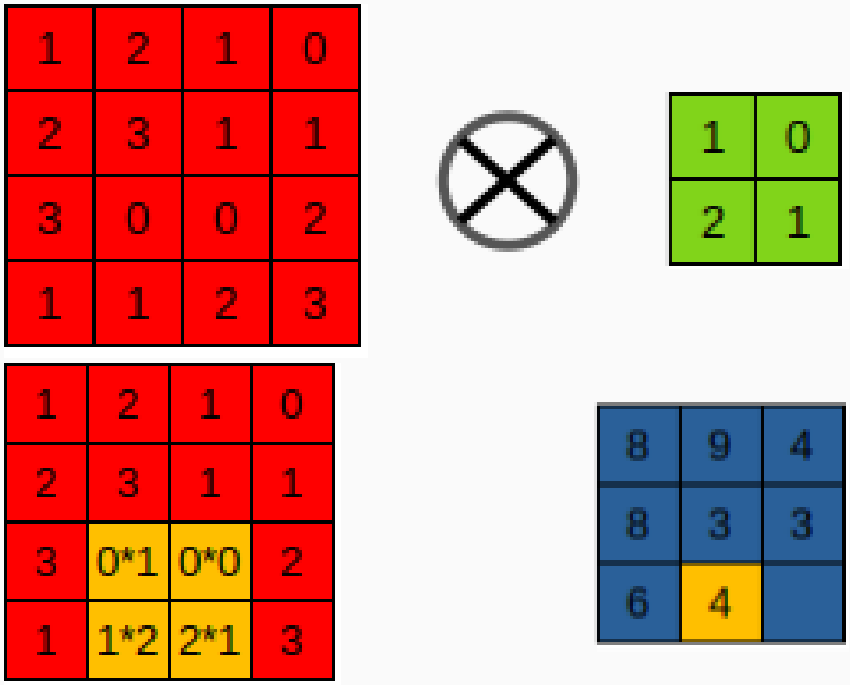
Convolución: Ejemplo



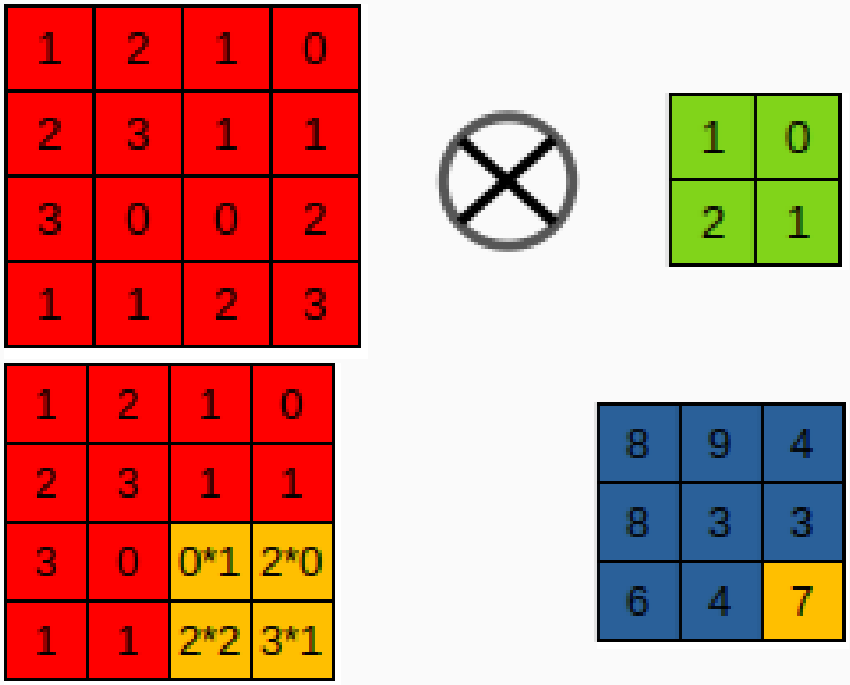
Convolución: Ejemplo



Convolución: Ejemplo



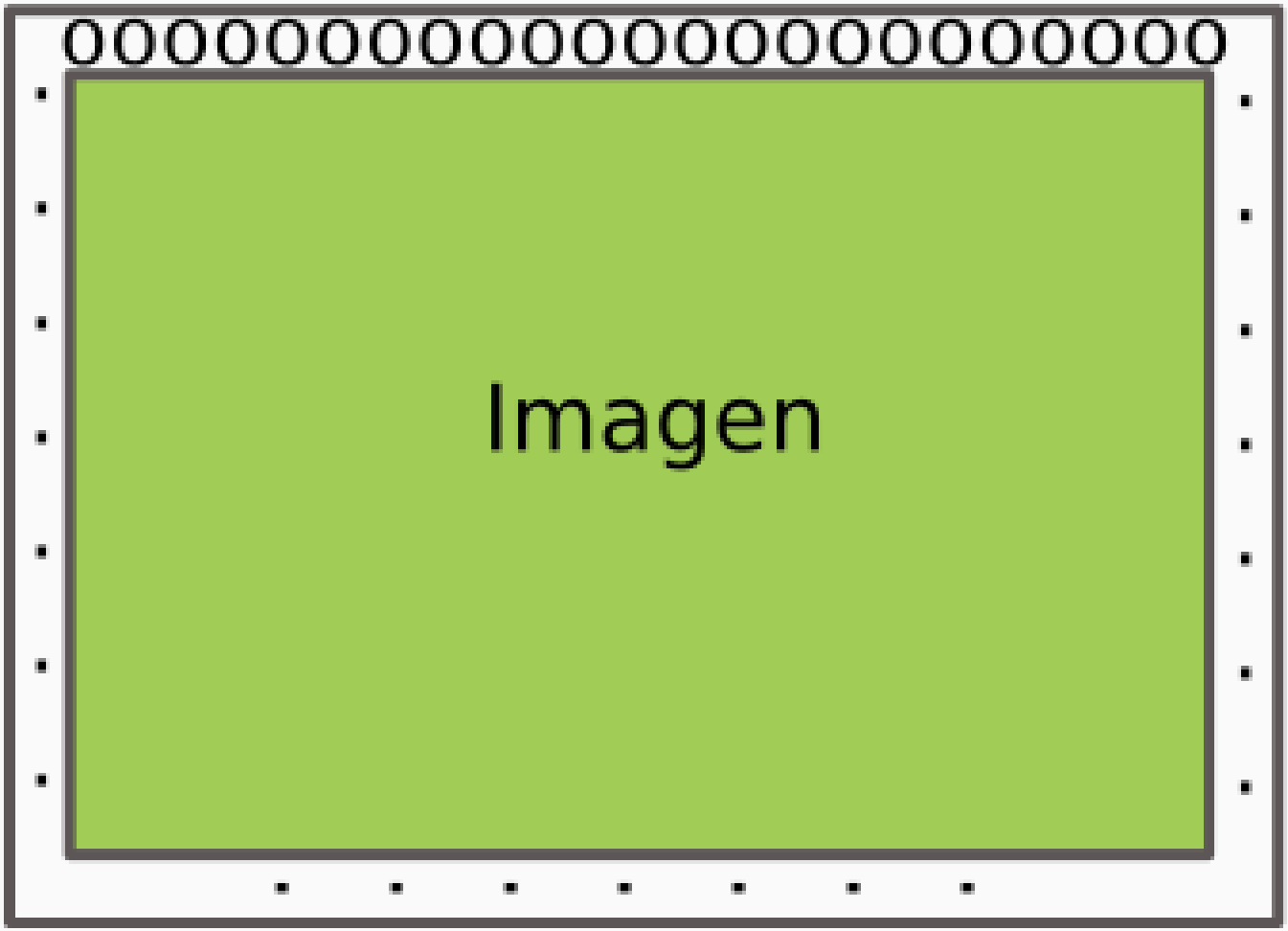
Convolución: Ejemplo



Padding y Striding

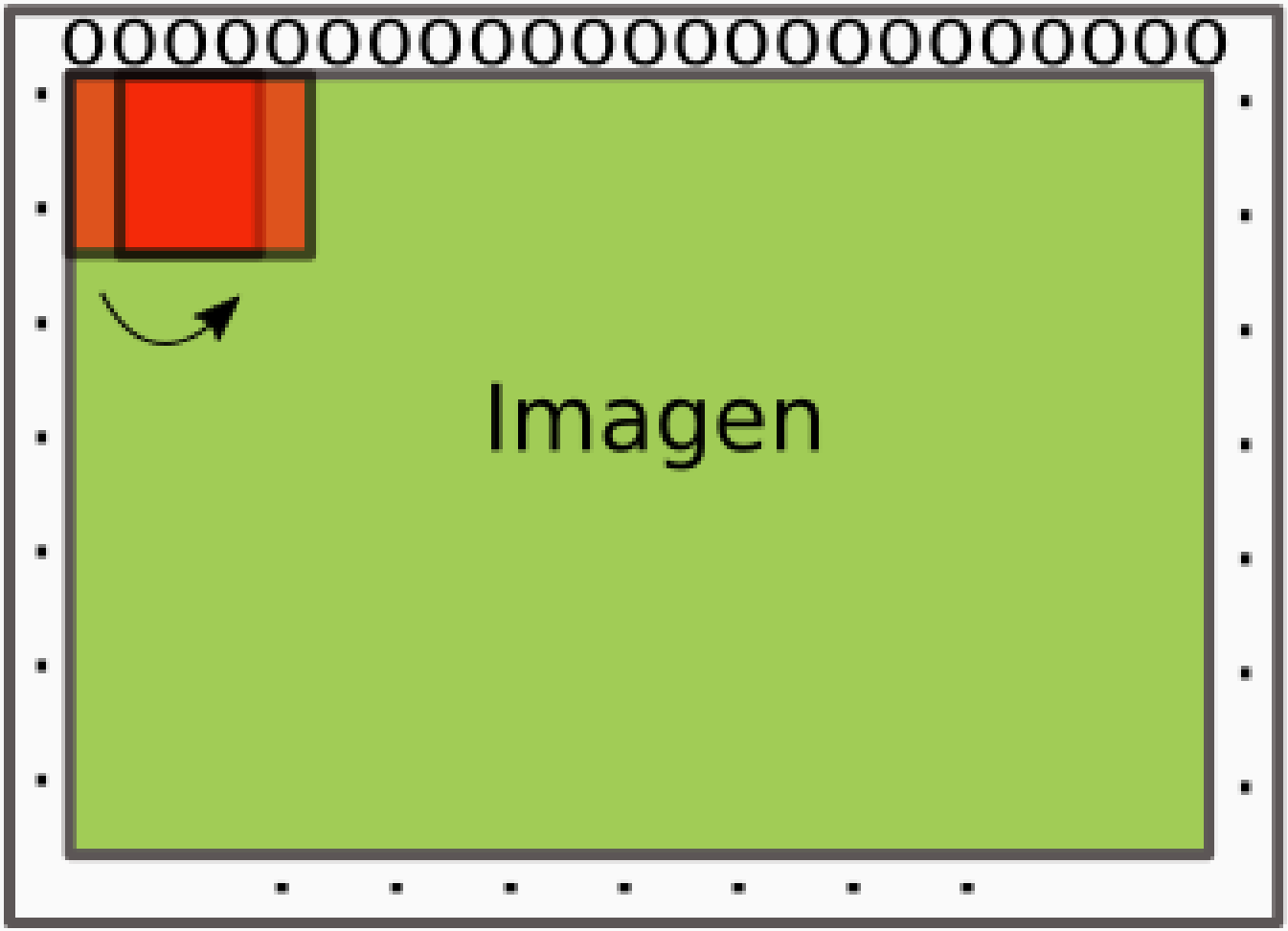
Padding y Striding

- Padding: Añadir ceros alrededor.



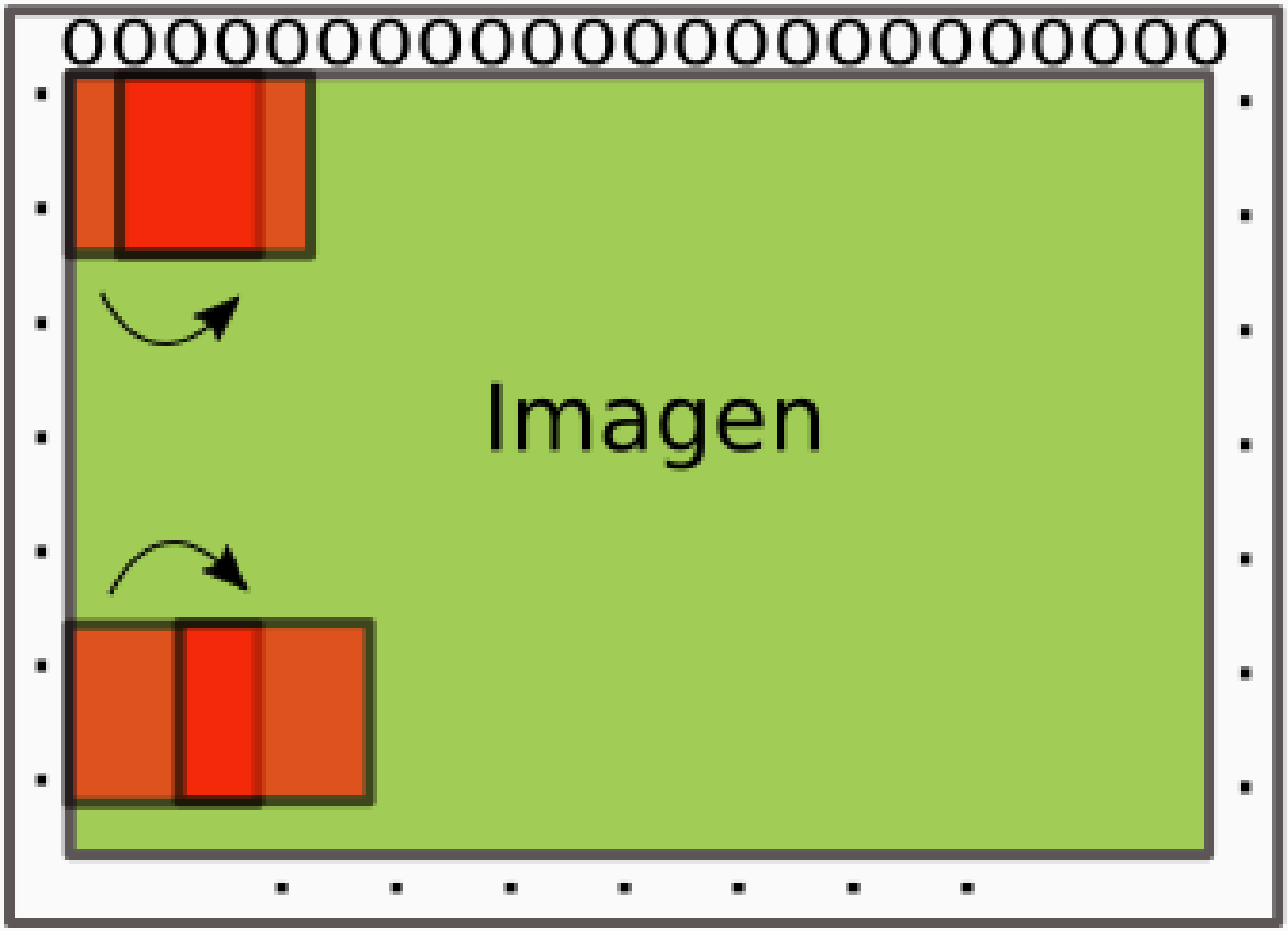
Padding y Striding

- Padding: Añadir ceros alrededor.
- Striding: Step para mover la "ventana".



Padding y Striding

- Padding: Añadir ceros alrededor.
- Striding: Step para mover la "ventana".



Padding y Striding

Sea la imagen $I = [I_{ij}] \in \mathcal{R}^{H \times W}$ con ancho (width) W y alto (height) H y el filtro $F = [F_{ij}] \in \mathcal{R}^{K \times K}$ con dimension $K \times K$. La convolución de I con F usando padding P y striding S se define como sigue.

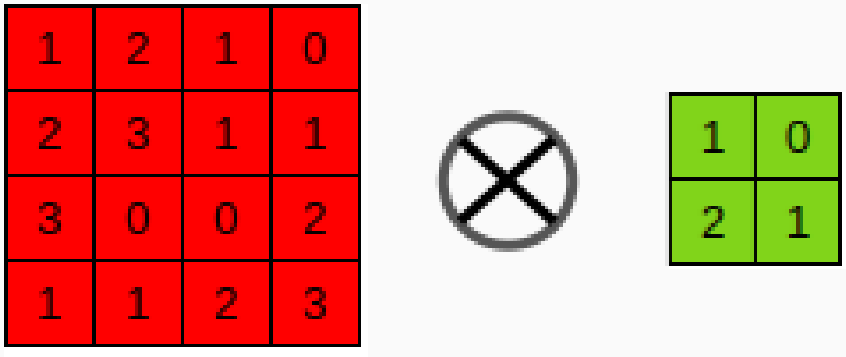
$O = I \bigotimes F = [O_{ij}] \in \mathcal{R}^{H' \times W'}$ con $H' = \frac{H-K+2P}{S} + 1$y $W' = \frac{W-K+2P}{S} + 1$, donde cada elemento se calcula de la siguiente forma:

$$O_{ij} = \sum_{p=1}^{K} \sum_{q=1}^{K} I'_{Si+p-P, Sj+q-P} F_{p,q}$$

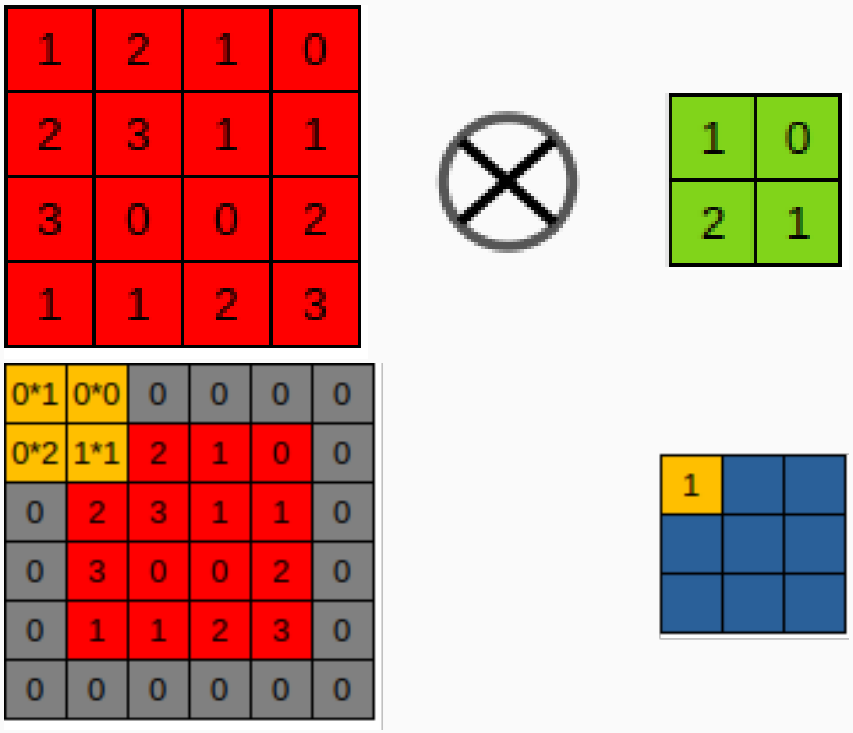
, donde I' se define de la siguiente manera:

$$I'_{i,j} = \begin{cases} I_{i,j} & ; i >= 1 \wedge i <= H \wedge j >= 1 \wedge j <= W \\ 0 & ; otherwise \end{cases}$$

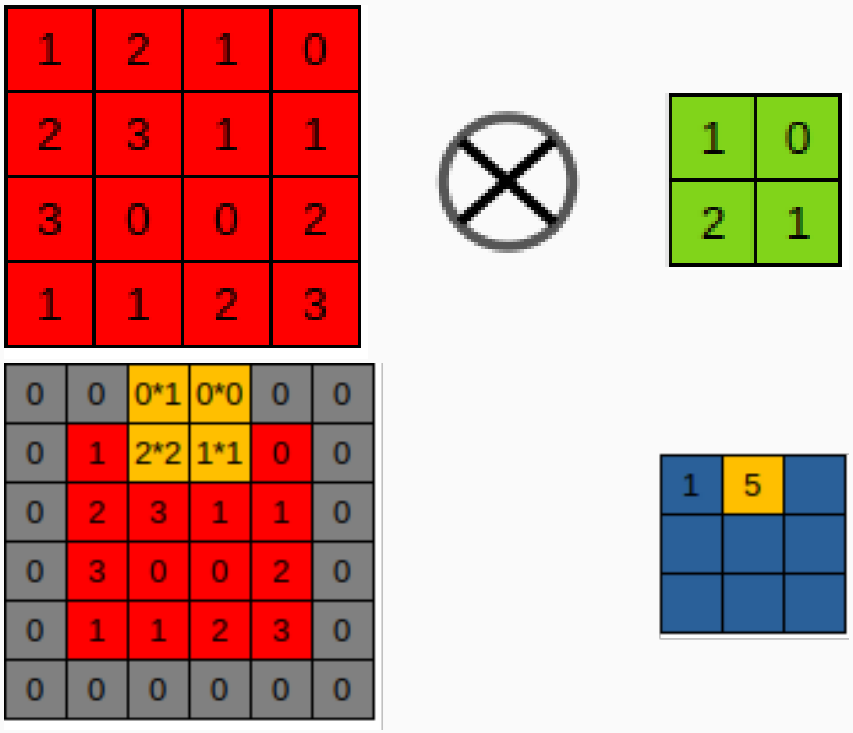
Convolución con padding y stride: Ejemplo



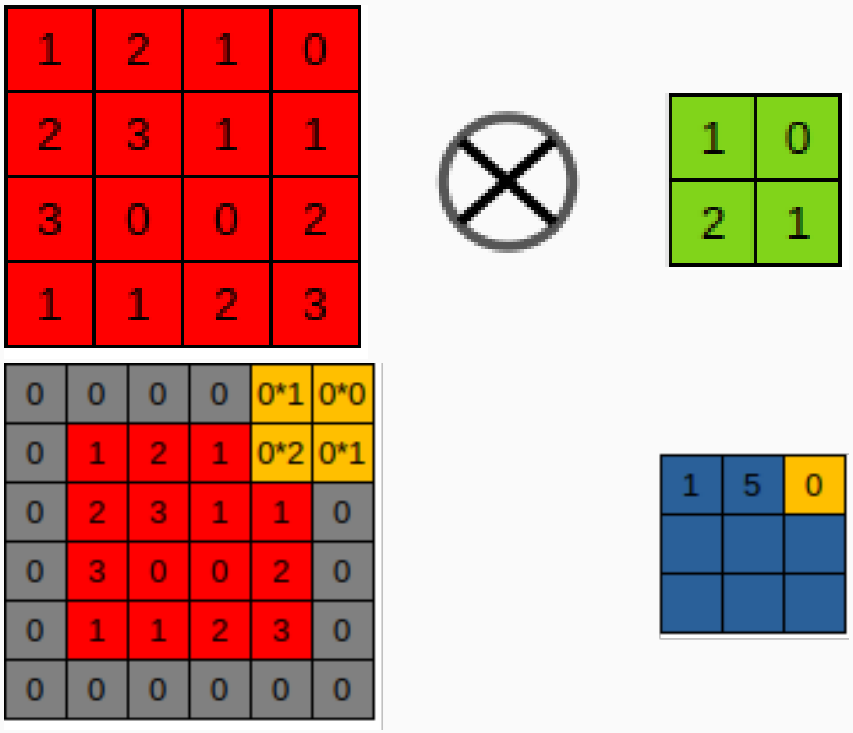
Convolución con padding y striding: Ejemplo



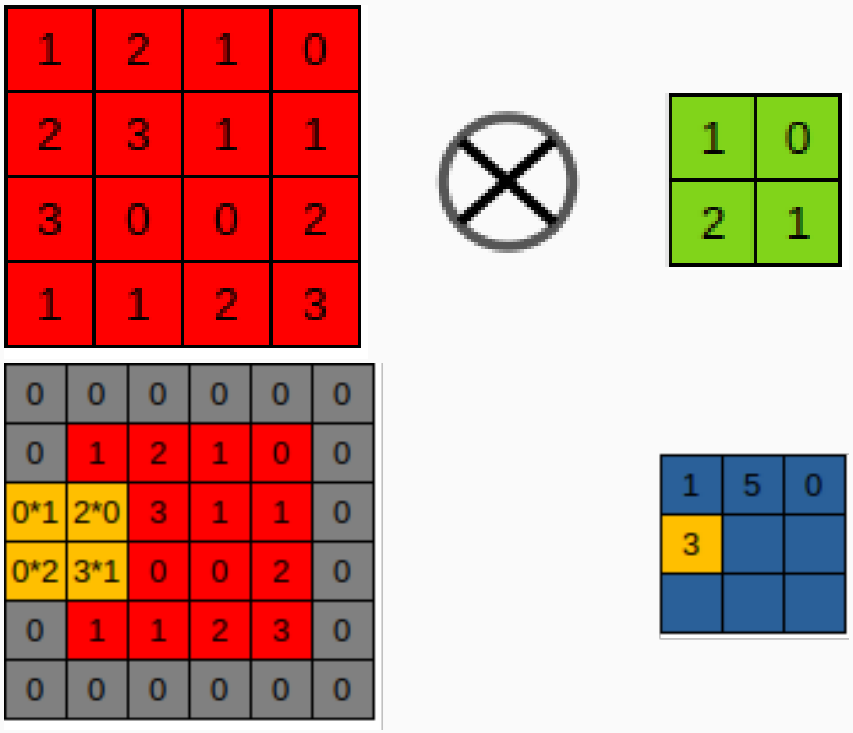
Convolución con padding y striding: Ejemplo



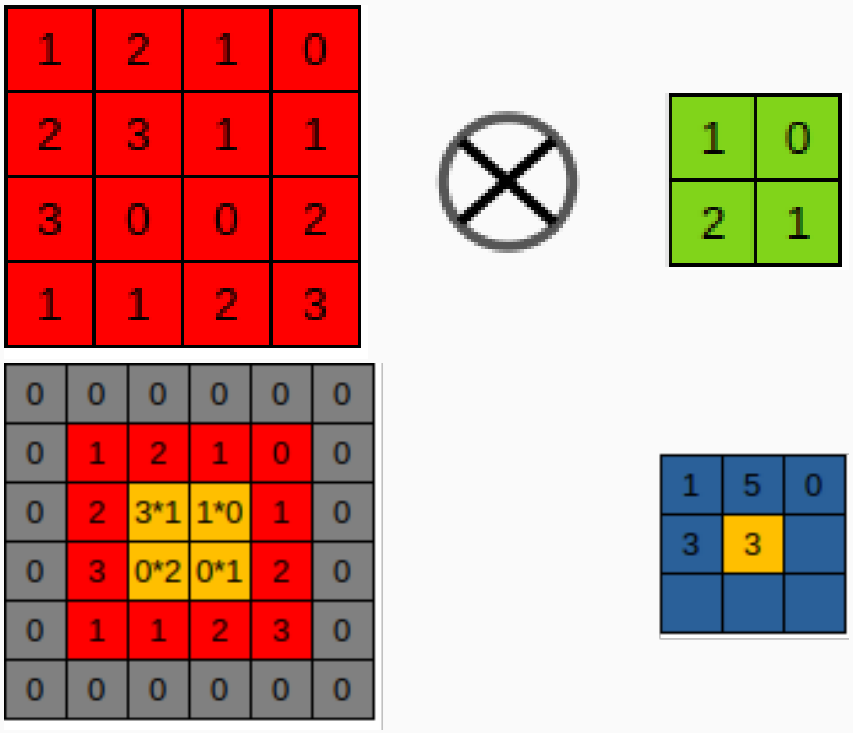
Convolución con padding y striding: Ejemplo



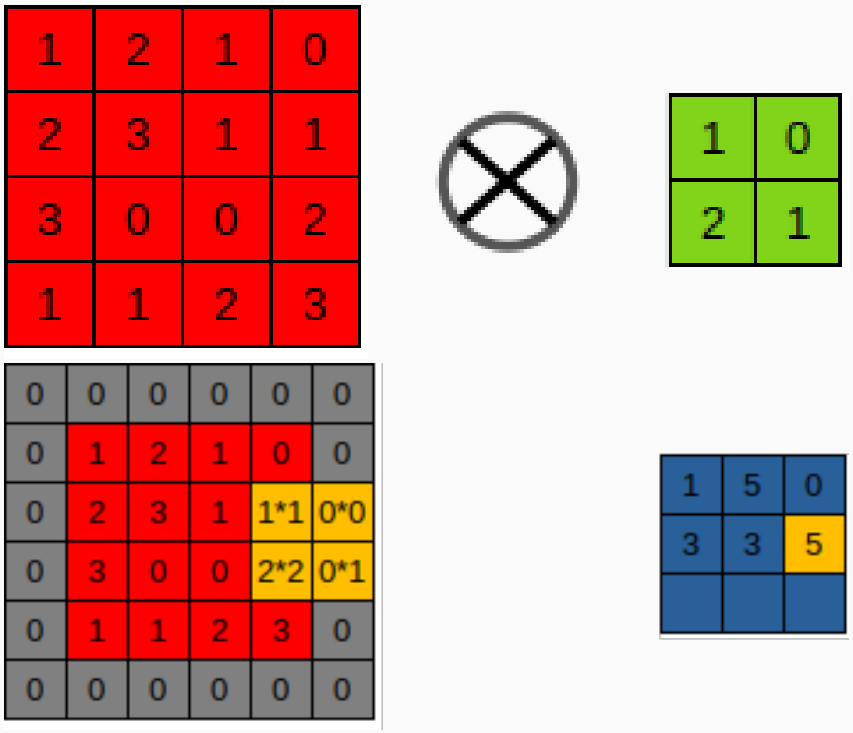
Convolución con padding y striding: Ejemplo



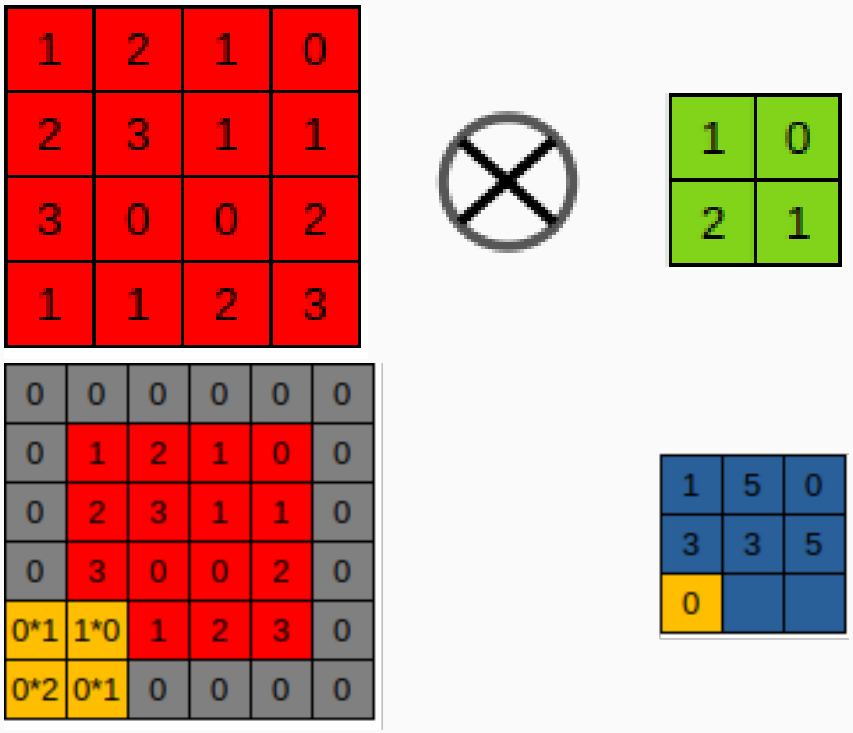
Convolución con padding y striding: Ejemplo



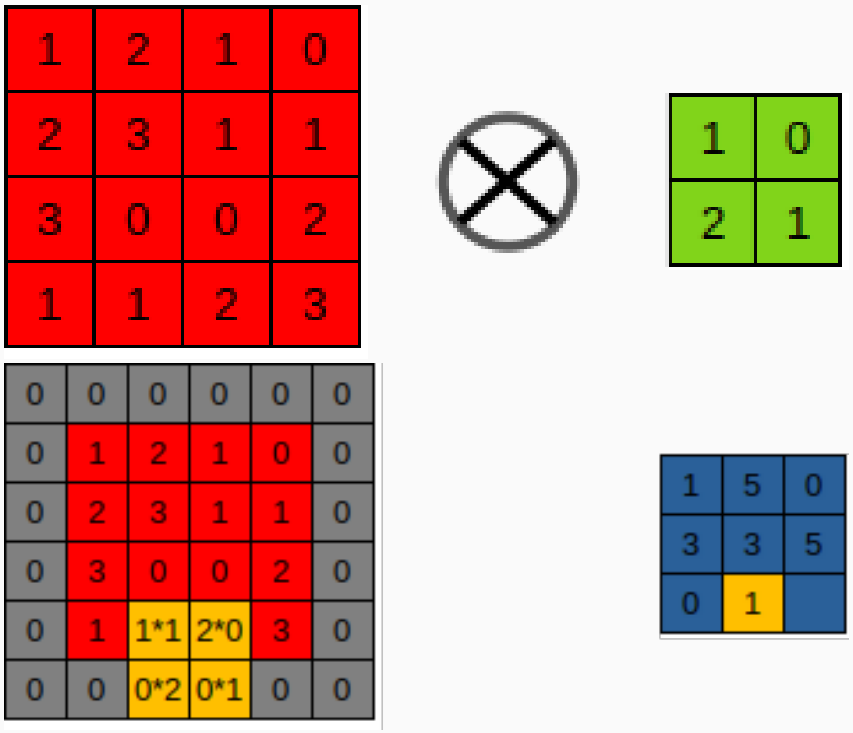
Convolución con padding y striding: Ejemplo



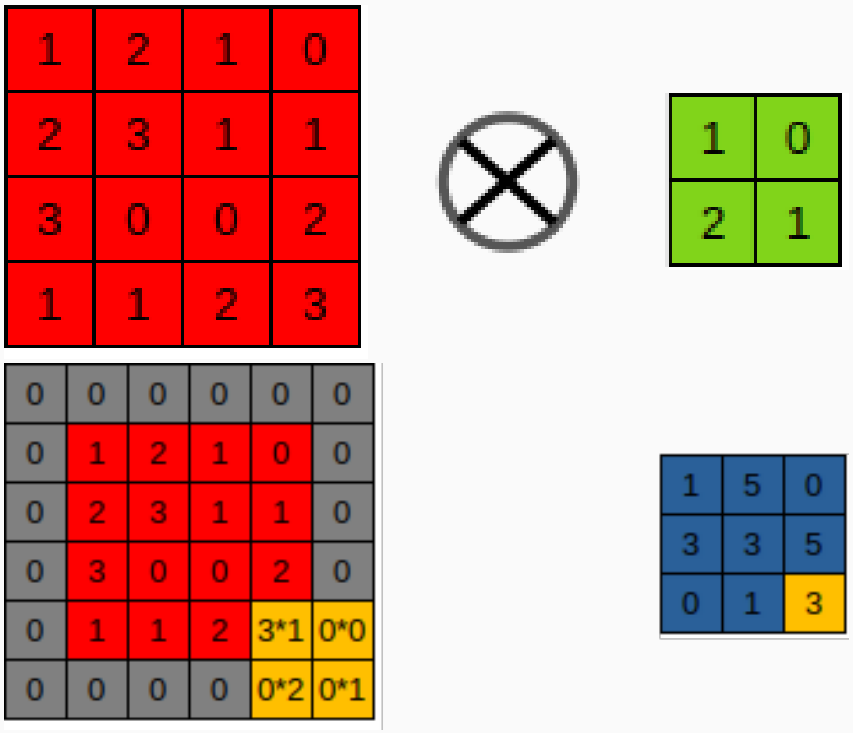
Convolución con padding y striding: Ejemplo



Convolución con padding y striding: Ejemplo



Convolución con padding y striding: Ejemplo



Convolución: Forwarding

En general no trabajaremos con una sola imágen pero con N imágenes al mismo tiempo, y cada "imágen" no es solo una matriz pero una coleccion de C (canales o channels) matrices (RGB). Por lo tanto, no estaremos trabajando con un matriz $I \in \mathcal{R}^{H \times W}$ sino con un tensor 4D $I = [I_{ijkl}] \in \mathcal{R}^{N \times C \times H \times W}$. Nuestros filtros tambien cambiaran de dimension. En lugar de ser una matriz ahora será un tensor 4D $F = [F_{ijkl}] \in \mathcal{R}^{C_{out} \times C \times K \times K}$. La convolución sigue siendo similar a como lo definimos anteriormente: $O = I \bigotimes F = [O_{ijkl}] \in \mathcal{R}^{N \times C_{out} \times H' \times W'}$ con $H' = \frac{H-K+2P}{S} + 1$ y $W' = \frac{W-K+2P}{S} + 1$, donde

$$O_{ijkl} = \sum_{p=1}^{C} \sum_{q=1}^{K} \sum_{r=1}^{K} I'_{i,p,Sk+q-P,Sl+r-P} F_{j,p,q,r}$$

y I' esta definidio de la siguiente manera:

$$I'_{ijkl} = \begin{cases} I_{ijkl}, k >= 1 \wedge k <= H \wedge l >= 1 \wedge l <= W \\ 0, otherwise \end{cases}$$

Conv: Backwarding

Calculemos la gradiente del loss function con respecto al input I.

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial I_{ijkl}} &= \sum_{pqrs} \frac{\partial \mathcal{L}}{\partial O_{pqrs}} \frac{\partial O_{pqrs}}{\partial I_{ijkl}} \\
&= \sum_{pqrs} \sum_{abc} \frac{\partial \mathcal{L}}{\partial O_{pqrs}} F_{qabc} \delta_{p,i} \delta_{a,j} \delta_{Sr+b-P,k} \delta_{Ss+c-P,l} \\
&= \sum_{qrs} \sum_{bc} \frac{\partial \mathcal{L}}{\partial O_{iqrs}} F_{qjbc} \delta_{Sr+b-P,k} \delta_{Ss+c-P,l} \\
&= \sum_{qrs} \frac{\partial \mathcal{L}}{\partial O_{iqrs}} F_{q,j,k-Sr+P,l-Ss+P}
\end{aligned}$$

Conv: Backwarding

Calculemos la gradiente del loss function con respecto a los filtros F.

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial F_{ijkl}} &= \sum_{pqrs} \frac{\partial \mathcal{L}}{\partial O_{pqrs}} \frac{\partial O_{pqrs}}{\partial F_{ijkl}} \\
&= \sum_{pqrs} \sum_{abc} \frac{\partial \mathcal{L}}{\partial O_{pqrs}} I'_{p,a,Sr+b-P,Ss+c-P} \delta_{q,i} \delta_{a,j} \delta_{b,k} \delta_{c,l} \\
&= \sum_{prs} \frac{\partial \mathcal{L}}{\partial O_{pirs}} I'_{p,j,Sr+k-P,Ss+l-P}
\end{aligned}$$

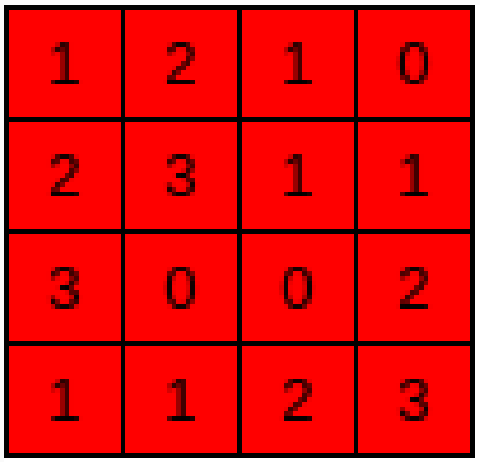
Layers muy utilizados en Computer Vision

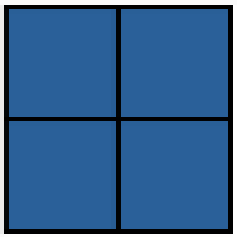
Pooling Layers: Max Pooling

Sea nuestro input $I = [I_{ijkl}] \in \mathcal{R}^{N \times C \times H \times W}$, la operación del Max Pool se define como $O = MaxPool_K(I) = [O_{ijkl}] \in \mathcal{R}^{N \times C \times \frac{H}{K} \times \frac{W}{K}}$, donde K es el "tamaño" del kernel, y cada elemento O_{ijkl} se calcula de la siguiente manera.

$$O_{ijkl} = max\{I_{i,j,Kk+p,Kl+q}, p >= 0 \wedge p <= K \wedge q >= 0 \wedge q <= K\}$$

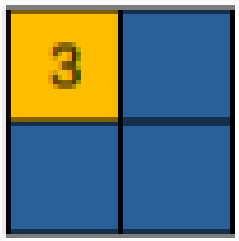
Max Pooling: Example



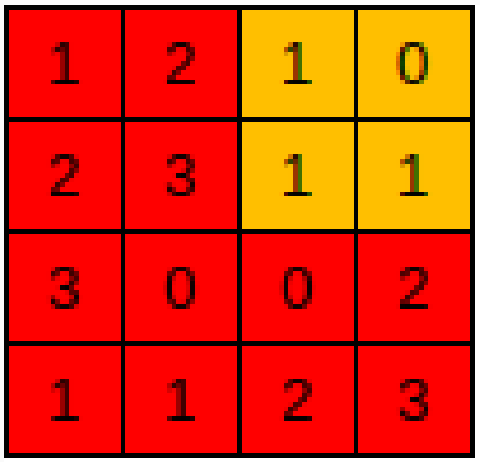


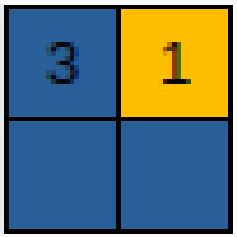
Max Pooling: Example

1	2	1	0
2	3	1	1
3	0	0	2
1	1	2	3

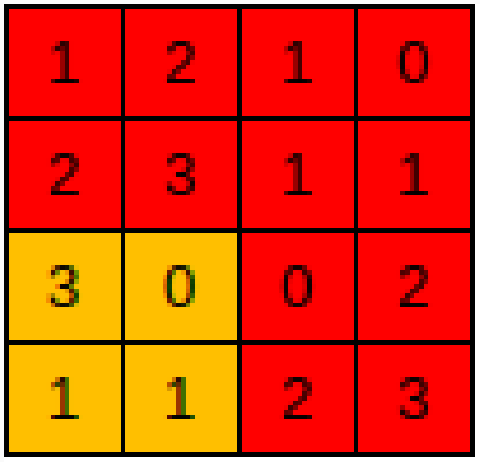


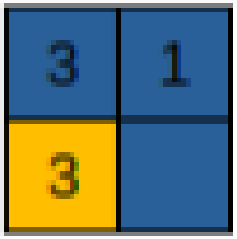
Max Pooling: Example



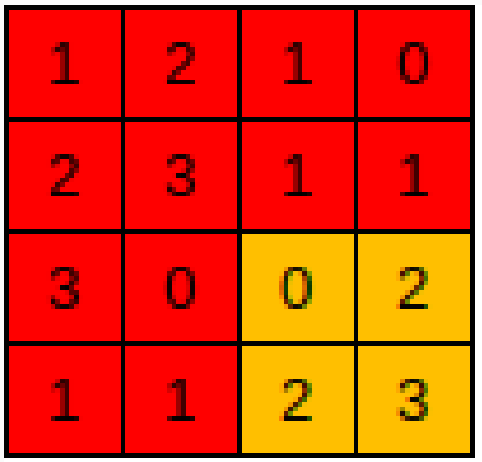


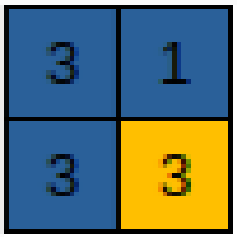
Max Pooling: Example





Max Pooling: Example





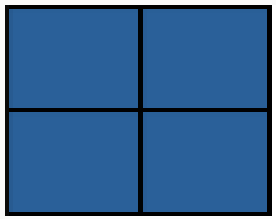
Pooling Layers: Average Pooling

Sea nuestro input $I = [I_{ijkl}] \in \mathcal{R}^{N \times C \times H \times W}$, la operación del Average Pool se define como $O = AvgPool_K(I) = [O_{ijkl}] \in \mathcal{R}^{N \times C \times \frac{H}{K} \times \frac{W}{K}}$, donde K es el "tamaño" del kernel, y cada elemento O_{ijkl} se calcula de la siguiente manera.

$$O_{ijkl} = \frac{1}{K^2} \sum_{p=1}^{K} \sum_{q=1}^{K} I_{i,j,Kk+p,Kl+q}$$

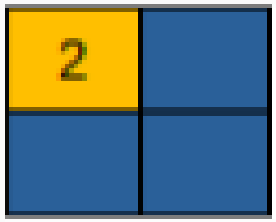
Average Pooling: Example

1	2	1	0
2	3	1	1
3	0	0	2
1	1	2	3

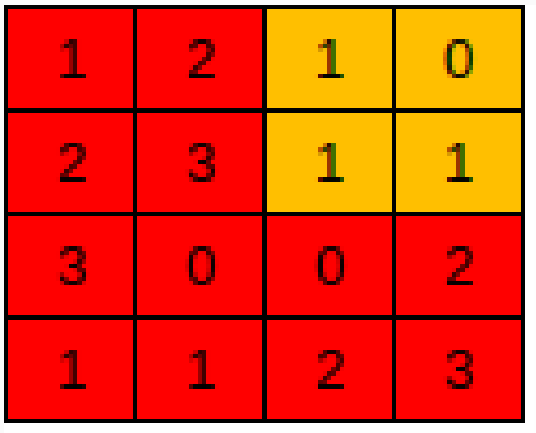


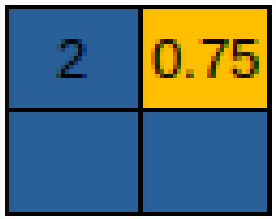
Average Pooling: Example

1	2	1	0
2	3	1	1
3	0	0	2
1	1	2	3

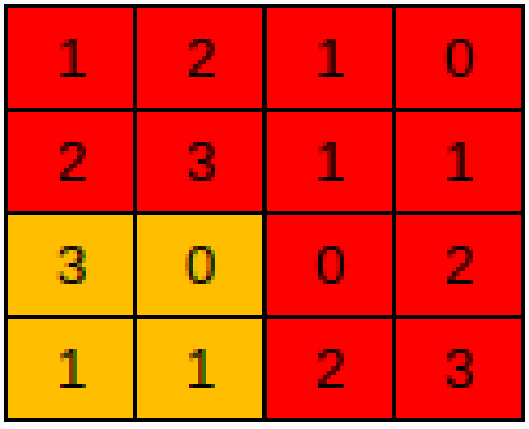


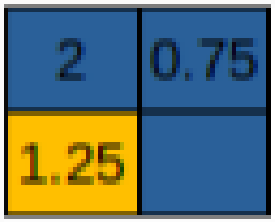
Average Pooling: Example



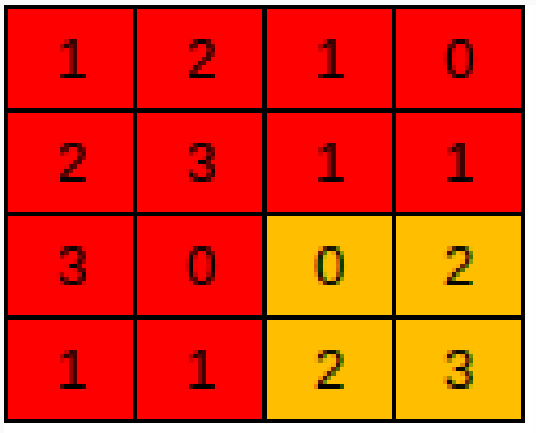


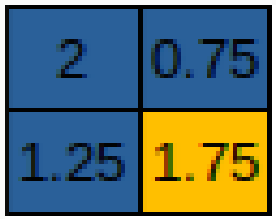
Average Pooling: Example





Average Pooling: Example





Pooling Layers: Adaptive Average Pooling

El adaptive average pooling, básicamente es un average pooling donde la "ventana" se define con los siguientes indices:

$$p_0 = floor(i\frac{H}{k})$$

$$p_f = ceil((i+1)\frac{W}{k})$$

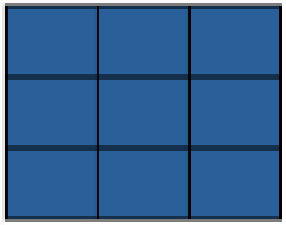
$$q_0 = floor(j\frac{H}{k})$$

$$q_f = ceil((j+1)\frac{W}{k})$$

$$O_{ij} = \frac{1}{(p_f - p_0)(q_f - q_0)} \sum_{a=p_0}^{p_f-1} \sum_{b=q_0}^{q_f-1} I_{ab}$$

Adaptive Average: Example

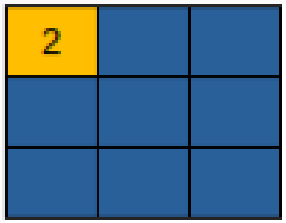
1	2	1	0
2	3	1	1
3	0	0	2
1	1	2	3



Adaptive Average: Example

1	2	1	0
2	3	1	1
3	0	0	2
1	1	2	3

$$i = 0, j = 0$$
$$p_0 = floor(i\tfrac{4}{3}) = 0$$
$$p_f = ceil((i+1)\tfrac{4}{3}) = 2$$
$$q_0 = floor(j\tfrac{4}{3}) = 0$$
$$q_f = ceil((j+1)\tfrac{4}{3}) = 2$$



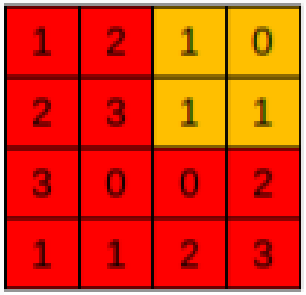
Adaptive Average: Example

1	2	1	0
2	3	1	1
3	0	0	2
1	1	2	3

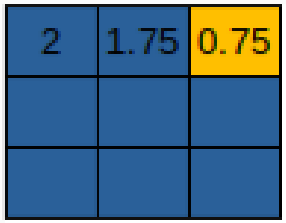
$$i = 0, j = 1$$
$$p_0 = floor(i\tfrac{4}{3}) = 0$$
$$p_f = ceil((i+1)\tfrac{4}{3}) = 2$$
$$q_0 = floor(j\tfrac{4}{3}) = 1$$
$$q_f = ceil((j+1)\tfrac{4}{3}) = 3$$

2	1.75	

Adaptive Average: Example



$$i = 0, j = 2$$
$$p_0 = floor(i\tfrac{4}{3}) = 0$$
$$p_f = ceil((i+1)\tfrac{4}{3}) = 2$$
$$q_0 = floor(j\tfrac{4}{3}) = 2$$
$$q_f = ceil((j+1)\tfrac{4}{3}) = 4$$



Adaptive Average: Example

1	2	1	0
2	3	1	1
3	0	0	2
1	1	2	3

$$i = 1, j = 0$$
$$p_0 = floor(i\tfrac{4}{3}) = 1$$
$$p_f = ceil((i+1)\tfrac{4}{3}) = 3$$
$$q_0 = floor(j\tfrac{4}{3}) = 0$$
$$q_f = ceil((j+1)\tfrac{4}{3}) = 2$$

2	1.75	0.75
2		

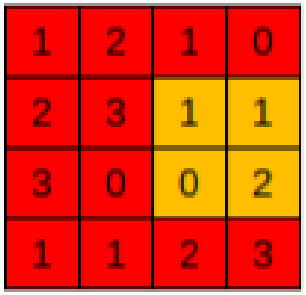
Adaptive Average: Example

1	2	1	0
2	3	1	1
3	0	0	2
1	1	2	3

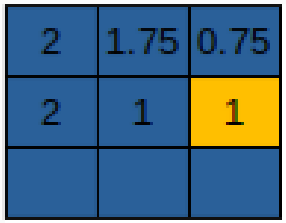
$$i = 1, j = 1$$
$$p_0 = floor(i\tfrac{4}{3}) = 1$$
$$p_f = ceil((i+1)\tfrac{4}{3}) = 3$$
$$q_0 = floor(j\tfrac{4}{3}) = 1$$
$$q_f = ceil((j+1)\tfrac{4}{3}) = 3$$

2	1.75	0.75
2	1	

Adaptive Average: Example



$$i = 1, j = 2$$
$$p_0 = floor(i\tfrac{4}{3}) = 1$$
$$p_f = ceil((i+1)\tfrac{4}{3}) = 3$$
$$q_0 = floor(j\tfrac{4}{3}) = 2$$
$$q_f = ceil((j+1)\tfrac{4}{3}) = 4$$



Adaptive Average: Example

1	2	1	0
2	3	1	1
3	0	0	2
1	1	2	3

$$i = 2, j = 0$$
$$p_0 = floor(i\tfrac{4}{3}) = 2$$
$$p_f = ceil((i+1)\tfrac{4}{3}) = 4$$
$$q_0 = floor(j\tfrac{4}{3}) = 0$$
$$q_f = ceil((j+1)\tfrac{4}{3}) = 2$$

2	1.75	0.75
2	1	1
1.25		

Adaptive Average: Example

1	2	1	0
2	3	1	1
3	0	0	2
1	1	2	3

$$i = 2, j = 1$$
$$p_0 = floor(i\tfrac{4}{3}) = 2$$
$$p_f = ceil((i+1)\tfrac{4}{3}) = 4$$
$$q_0 = floor(j\tfrac{4}{3}) = 1$$
$$q_f = ceil((j+1)\tfrac{4}{3}) = 3$$

2	1.75	0.75
2	1	1
1.25	0.75	

Adaptive Average: Example

1	2	1	0
2	3	1	1
3	0	0	2
1	1	2	3

$$i = 2, j = 2$$
$$p_0 = floor(i\frac{4}{3}) = 2$$
$$p_f = ceil((i+1)\frac{4}{3}) = 4$$
$$q_0 = floor(j\frac{4}{3}) = 2$$
$$q_f = ceil((j+1)\frac{4}{3}) = 4$$

2	1.75	0.75
2	1	1
1.25	0.75	1.75

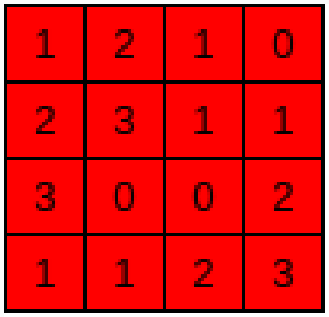
Previniendo el sobreentrenamiento: Dropout Layers

Sea nuestro input $I = [I_{ijkl}] \in \mathcal{R}^{N \times C \times H \times W}$, la operación del Dropout se define como $O = Dropout_p(I) = [O_{ijkl}] \in \mathcal{R}^{N \times C \times H \times W}$, donde cada elemento O_{ijkl} se calcula de la siguiente manera.

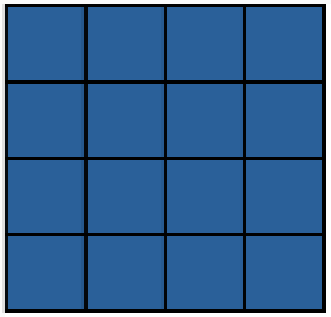
$$O_{ijkl} = \begin{cases} 0 & ; random(0,1) < p \\ \frac{I_{ijkl}}{p} & ; otherwise \end{cases}$$

$random(0,1)$ indica la generación de un número random entre 0 y 1.

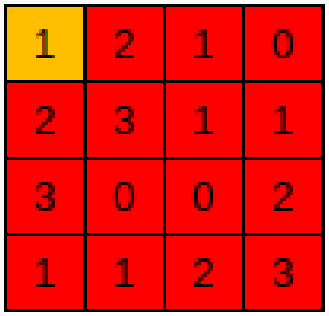
Dropout: Example



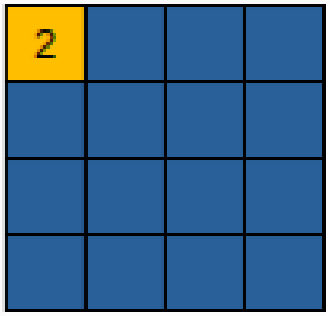
$$p = 0.5$$



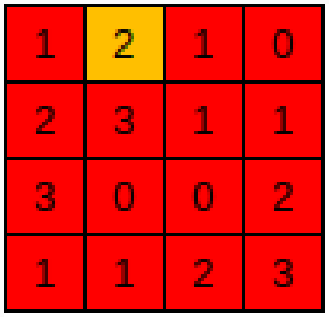
Dropout: Example



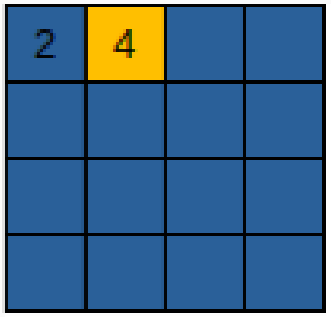
$$p = 0.5$$
$$random(0, 1) = 0.6$$



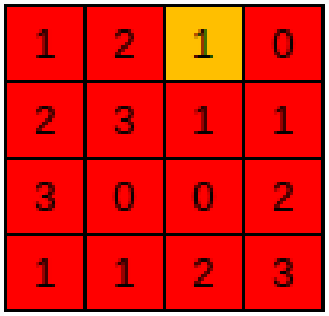
Dropout: Example



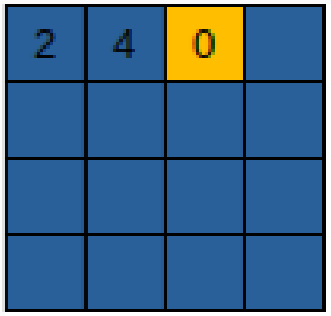
$$p = 0.5$$
$$random(0, 1) = 0.8$$



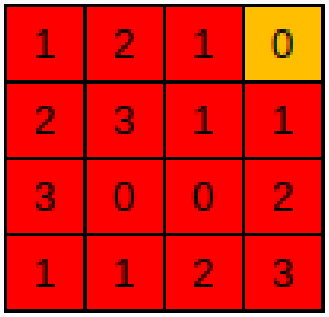
Dropout: Example



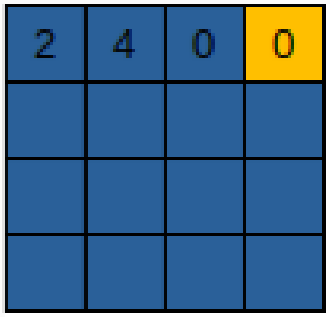
$$p = 0.5$$
$$random(0, 1) = 0.2$$



Dropout: Example



$$p = 0.5$$
$$random(0, 1) = 0.1$$



Dropout: Example

1	2	1	0
2	3	1	1
3	0	0	2
1	1	2	3

2	4	0	0
4	0	6	2
6	0	0	4
0	2	4	0

BatchNorm Layers: Lidiando con el "covariate shift"

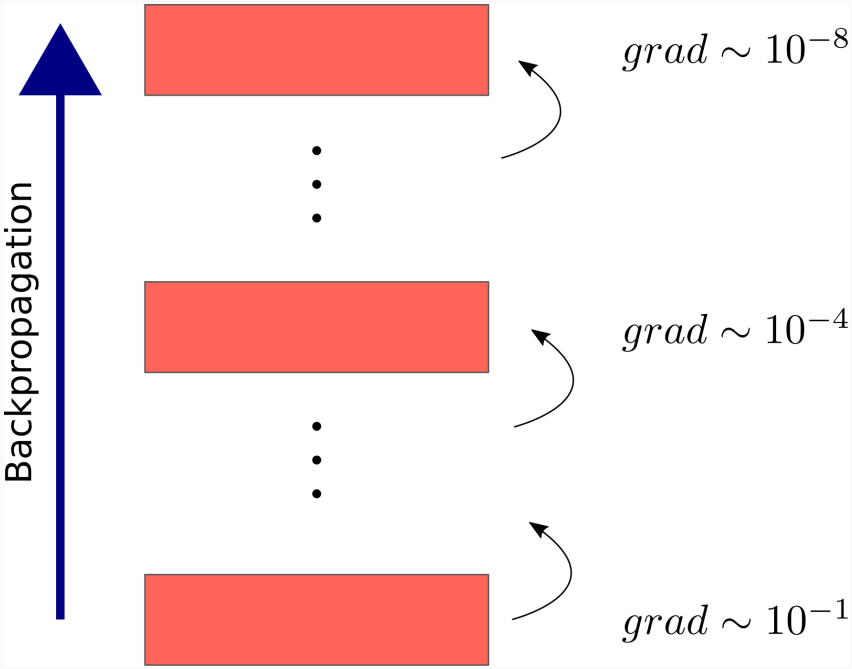
Sea nuestro input $I = [I_{ijkl}] \in \mathcal{R}^{N \times C \times H \times W}$, podemos calcular el promedio $\mu \in \mathcal{R}^C$ y la varianza $\sigma^2 \in \mathcal{R}^C$ atraves de los canales C de nuestro input de la siguiente manera: $\mu_j = \frac{1}{NHW} \sum_{ikl} I_{ijkl}$, $\sigma_j^2 = \frac{1}{NHW} \sum_{ikl} (I_{ijkl} - \mu_j)^2$. Usando esta media y varianza podemos reescalar (Z-score + transformación lineal) nuestro input de la siguiente forma:

$O_{ijkl} = \frac{I_{ijkl} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}} \gamma_j + \beta_j$

Donde $\gamma \in \mathcal{R}^C$ y $\beta \in \mathcal{R}^C$ son valores que se van a ir aprendiendo.

Problema del desvanecimiento de la gradiente y Redes Residuales

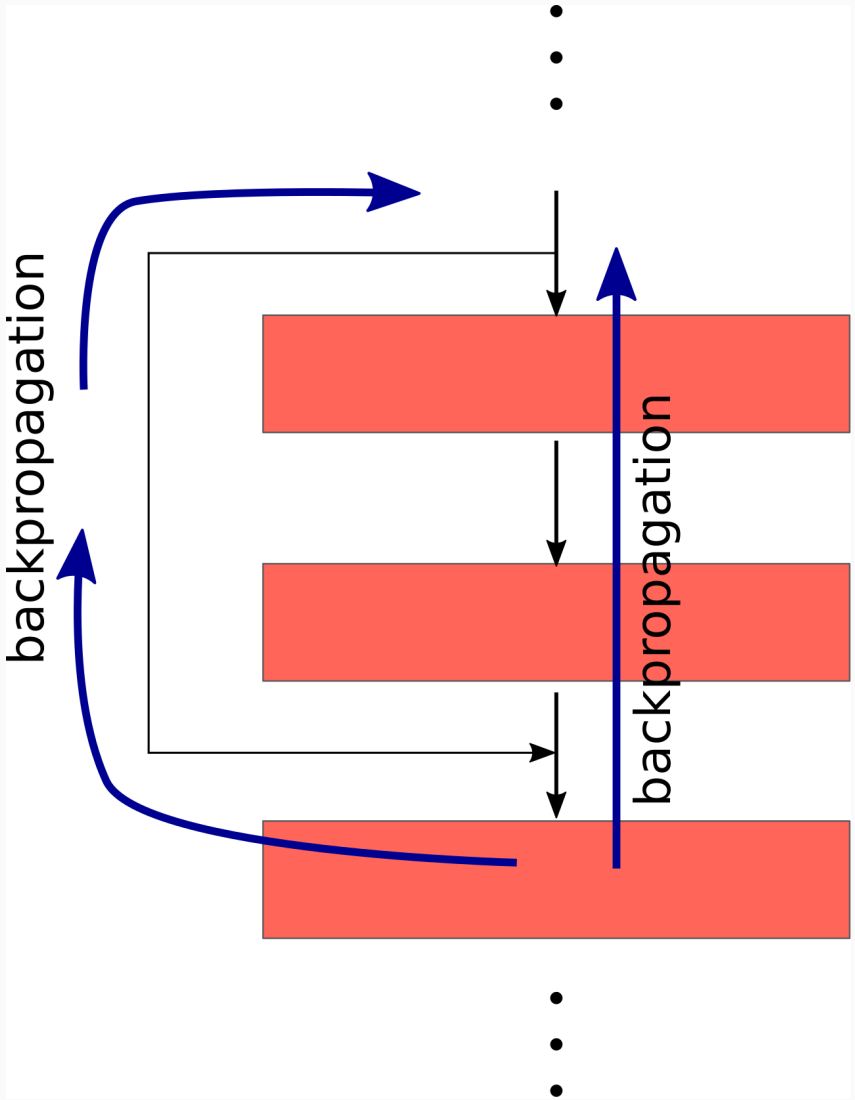
Que es el desvanecimiento de la gradiente?



- Mientras más profunda sea (i.e. más layers tenga) nuestra red, la gradiente que se "propaga" se vuelve mucho más pequeña.
- Por lo que los parámetros que se encuentran en las primeras capas no se pueden actualizar apropiadamente o su aprendizaje se vuelve muy lento.
- Debido a la limitada precisión de las computadoras, puede que la gradiente sufra de underflowing.

Skip Connections y Redes Residuales

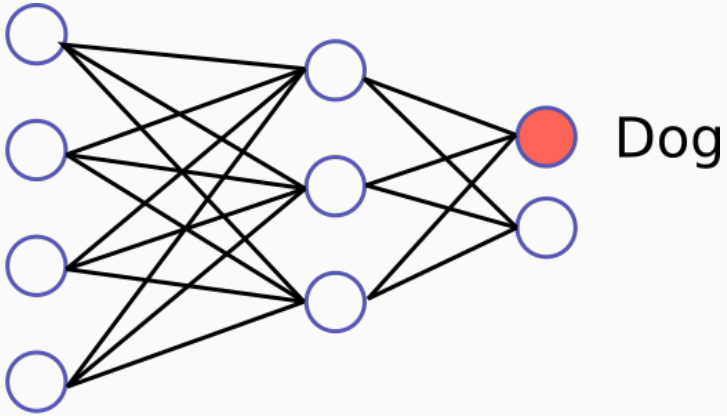
- Los “skip connections” o “shortcuts” proveen de un “camino alterno” para la gradiente que se propaga.
- Eso previene que la gradiente disminuya abruptamente.



Clasificación de Imágenes

Que es la clasificación de imágenes?

Esta tarea consiste en entrenar una red neuronal de tal forma que pueda discernir correctamente el tipo de imagen que se le esta proveyendo como input.



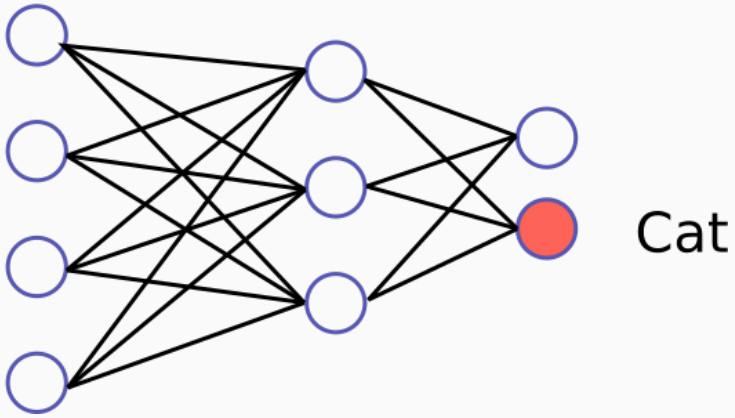
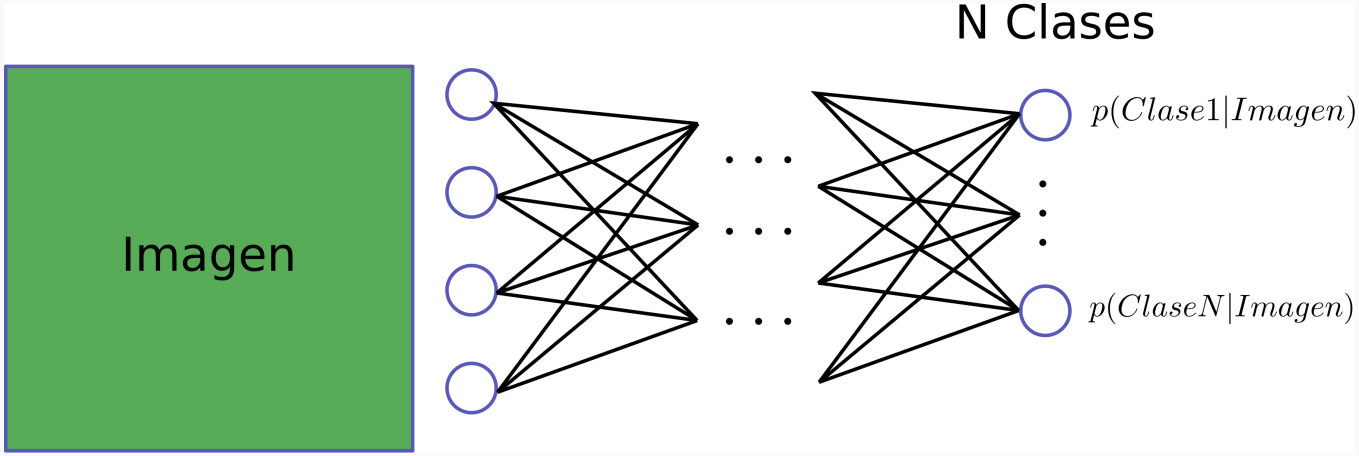


Image Classification Networks

En general nuestra red neuronal de clasificación debera devolvernos una distribución de probabilidad. Donde cada salida representa la probabilidad de que la imagen pertenezca a dicha clase. En el mejor de los casos, quisieramos una probabilidad de 1 en la clase que corresponde y 0 en las demás probabilidades.



$$\sum_{k=1}^{N} p(Clasek|Imagen) = 1$$

Funciones de activación: Sigmoid y Softmax

Normalmente la salida de las redes neuronales no siguen una distribución de probabilidad. Por lo que usualmente se utiliza o una función sgmoidea o softmax en la ultima capa.

Clasificación Binaria:

$$\begin{aligned} y &= Sigmoid(x) \\ &= \frac{1}{1+e^{-x}} \end{aligned}$$

Clasificación de multiples clases:

$$\begin{aligned} \vec{y} &= Softmax(\vec{x}) \\ y_j &= \frac{e^{x_j}}{\sum_k e^{x_k}} \\ \sum_j y_j &= 1 \end{aligned}$$

Binary Cross Entropy y Cross Entropy Loss: Divergencia KL (Kullback-Leibler)

La divergencia KL nos indica que tan alejado una distribucion de probabilidad q_i esta de otra distribución de probabilidad p_i.

$$KL(p|q) = \sum_i p_i \log \frac{p_i}{q_i}$$
$$= \sum_i p_i \log p_i - \sum_i p_i \log q_i$$

En nuestro caso, p_i representaría la distribucion de probabilidad correcta (e.g. [0,0,1, ...,0]) mientras que q_i representaría la distribución de probabilidad obtenida por la red neuronal (e.g. [0.1,0.01,0.7,...,0.001]) que queremos que se asemeje a la distribución $p_i i$ (i.e. queremos disminuir $KL(p|q)$). El termino $\sum_i p_i \log p_i$ se mantendrá constante durante el entrenamiento por lo que minimizar $KL(p|q)$ seria equivalente a minimizar $-\sum_i p_i \log q_i$.

$$\mathcal{L} = -\sum p_i \log q_i$$

Binary Cross Entropy Loss

Nuestro input solo puede pertencer a dos posibles clases: C_1 y C_2. Nuestra red neuronal nos retornará un solo valor y que nos indicará la probabilidad de que la clase pertenezca a la clase C_1. Ya que solo tenemos 2 clases, la probabilidad de que el input pertenezca a la clase C_2 sería $1 - y$. Lo mismo pasa con la distribución de probabilidad que es proporcionada para el enterenamiento: t. El valor de t nos indicará si el input pertence a la clase C_1 mientras que $1 - t$ nos indicará si el input pertenece a la clase C_2.

$$\mathcal{L} = -t \log y - (1 - t) \log(1 - y)$$

Cross Entropy Loss

Nuestro input puede pertenecer a multiples clases: $C_1, C_2, ..., C_N$.
Nuestra red neuronal, por ende, debe retornarnos N valores ($\vec{y}$) representando la probabilidad de pertenencia a cada clase. Y la distribución de probabilidad usada para el entrenamiento $\vec{t}$ tambien debe de tener N elementos, donde la mayoria son zeros y solo un valor es 1 (la clase correcta).

$$\mathcal{L} = -\sum_i t_i \log y_i$$