S02.2: Clasificación no lineal

Redes Neuronales y *Deep Learning*

Dr. Juan Bekios Calfa

Magíster en Inteligencia Artificial

UNIVERSIDAD ADOLFO IBÁÑEZ

Información de Contacto

- Juan Bekios Calfa
  - email: juan.bekios@edu.uai.cl, juan.bekios@ucn.cl
  - Web page: http://jbekios.ucn.cl
  - Teléfono: 235(5162) - 235(5125)

UNIVERSIDAD ADOLFO IBÁÑEZ

Contenidos

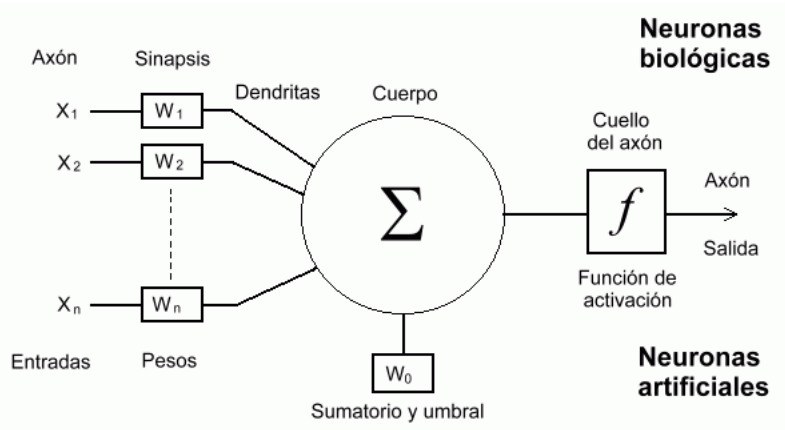
Introducción

- ***Deep learning*** es un conjunto de métodos de aprendizaje que intentan modelar datos con arquitecturas complejas que combinan diferentes transformaciones no lineales.
- Los elementos principales del *deep learning* son las redes neuronales, que se combinan para formar las redes neuronales profundas.
- Las técnicas desarrolladas para implementar diferentes arquitecturas de redes neuronales, y en especial el *deep learning*, han permitido el progreso significativo de diferentes áreas de la ingeniería y la inteligencia artificial.

UNIVERSIDAD ADOLFO IBÁÑEZ

Contenidos

Modelo de una neurona simple



https://www.um.es/LEQ/Atmosferas/Ch-VI-3/F63s4p3.htm

Hipótesis lineal y aprendizaje de características

Hasta ahora, hemos considerado (mayormente) algoritmos de aprendizaje automático donde la hipótesis es lineal.

$$h_\theta(x) = \theta^T \phi(x)$$

Donde $\theta : \mathbb{R}^n \to \mathbb{R}^k$, un conjunto de características (*features*) típicamente no lineales.

Ejemplo: Polinomios, funciones de base radial, características personalizadas como TF-IDF (en muchos dominios cada 10 años más o menos habrá nuevos tipos de características).
El rendimiento de estos algoritmos depende crucialmente de la obtención de buenas características.

Pregunta

¿Podemos llegar a un algoritmo que aprenda automáticamente las características en sí mismas?

(Mejora del modelo neuronal)

UNIVERSIDAD ADOLFO IBÁÑEZ

Contenidos

Aprendizaje de características

En lugar de un **simple clasificador lineal**, consideremos una clase de hipótesis de dos etapas donde una función lineal crea las características y otra produce el final hipótesis.

$$h_\theta(x) = W_2\phi(x) + b_2 = W_2(W_1x + b_1) + b_2$$
$$\theta = \{W_1 \in \mathbb{R}^{k\times n}, b_1 \in \mathbb{R}^k, W_2 \in \mathbb{R}^{1\times k}, b_2 \in \mathbb{R}\}$$

Pero hay problema:

$$h_\theta(x) = W_2(W_1x + b_1) + b_2 = \widetilde{W}x + \widetilde{b}$$

Aprendizaje de características

En lugar de un **simple clasificador lineal**, consideremos una clase de hipótesis de dos etapas donde una función lineal crea las características y otra produce el final hipótesis.

$$h_\theta(x) = W_2\phi(x) + b_2 = W_2(W_1x + b_1) + b_2$$
$$\theta = \{W_1 \in \mathbb{R}^{k\times n}, b_1 \in \mathbb{R}^k, W_2 \in \mathbb{R}^{1\times k}, b_2 \in \mathbb{R}\}$$

Pero hay problema:

$$h_\theta(x) = W_2(W_1x + b_1) + b_2 = \widetilde{W}x + \widetilde{b}$$

¡Todavía estamos usando un clasificador lineal! (la aparente complejidad añadida es realidad no cambia la función de la hipótesis subyacente)

Ejercicio

¿Podemos convertir la clasificación lineal de una neurona a una no lineal?

`https://playground.tensorflow.org/`

Redes Neuronales

Las redes neuronales son una simple extensión de esta idea, donde adicionalmente aplicamos una **función no lineal** después de cada transformación lineal.

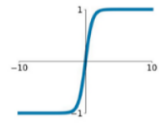
$$h_\theta(x) = f_2(W_2 f_1(W_1 x + b_1) + b_2)$$

Donde $f_1, f_2 : \mathbb{R}^n \to \mathbb{R}^k$

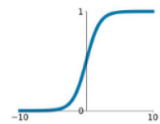
Redes Neuronales

Elecciones comunes de f_i:

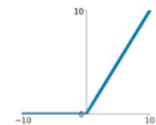
Tangente hiperbólica: $f(x) = tanh(x) = \frac{e^{2x-1}}{e^{2x-1}}$



Sigmoide: $f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$

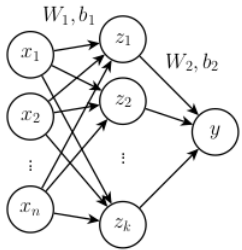


Rectified linear unit (ReLU): $f(x) = max\{x, 0\}$



Modelo gráfico de una Red Neuronal

Podemos representar la forma de las redes neuronales con el siguiente grafo dirigido.



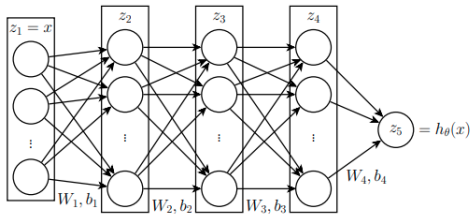
La capa intermedia z se conoce como la capa oculta o activaciones.

UNIVERSIDAD ADOLFO IBÁÑEZ

Contenidos

Deep Learning

Deep Learning (Aprendizaje profundo) se refiere (casi siempre) al aprendizaje automático utilizando modelos de red neuronal con múltiples capas ocultas.



Función de hipótesis para redes de k-capas.

$$z_{i+1} = f_i(W_i z_i + b_i),\ z_1 = x\ h_\theta(x) = z_k$$

Propiedades de las redes neuronales

- Una **red neuronal** tendrá una sola capa oculta (y suficientes unidades ocultas) es una “aproximador” de funciones universales, puede aproximar cualquier función sobre las entradas.

Propiedades de las redes neuronales

- Una **red neuronal** tendrá una sola capa oculta (y suficientes unidades ocultas) es una “aproximador” de funciones universales, puede aproximar cualquier función sobre las entradas.
- En la práctica, no es tan relevante (similar a como los polinomios pueden representar en cualquier función), y el aspecto más importante es que parecen funcionar muy bien en la práctica para muchos dominios.

Propiedades de las redes neuronales

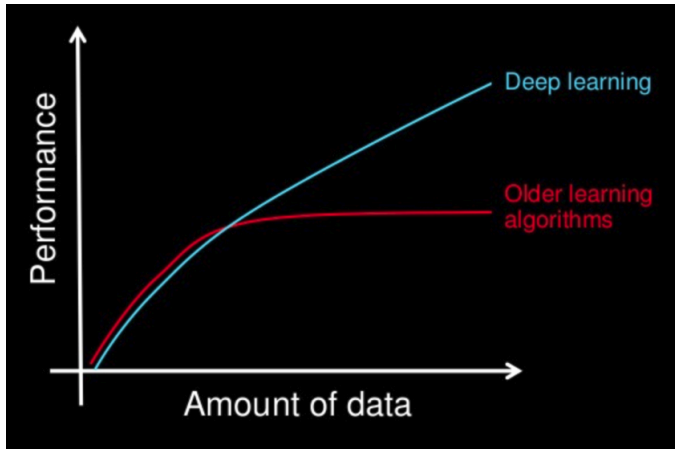
- Una **red neuronal** tendrá una sola capa oculta (y suficientes unidades ocultas) es una “aproximador” de funciones universales, puede aproximar cualquier función sobre las entradas.
- En la práctica, no es tan relevante (similar a como los polinomios pueden representar en cualquier función), y el aspecto más importante es que parecen funcionar muy bien en la práctica para muchos dominios.
- La hipótesis $h_\theta(x)$ no es una función convexa de parámetros $\Theta = \{W_i, b_i\}$, por lo que es posible encontrar un óptimo local.

Propiedades de las redes neuronales

- Una **red neuronal** tendrá una sola capa oculta (y suficientes unidades ocultas) es una "aproximador" de funciones universales, puede aproximar cualquier función sobre las entradas.
- En la práctica, no es tan relevante (similar a como los polinomios pueden representar en cualquier función), y el aspecto más importante es que parecen funcionar muy bien en la práctica para muchos dominios.
- La hipótesis $h_\theta(x)$ no es una función convexa de parámetros $\Theta = \{W_i, b_i\}$, por lo que es posible encontrar un óptimo local.
- Las elecciones arquitectónicas (cuántas capas, cómo están conectadas, etc.), se convierten en importantes elecciones de diseño algorítmico (**hiperparámetros**).

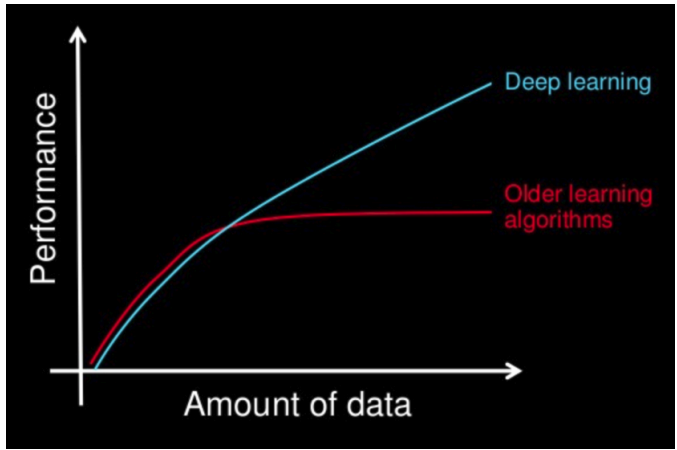
¿Porqué ahora?

- Muchos datos.



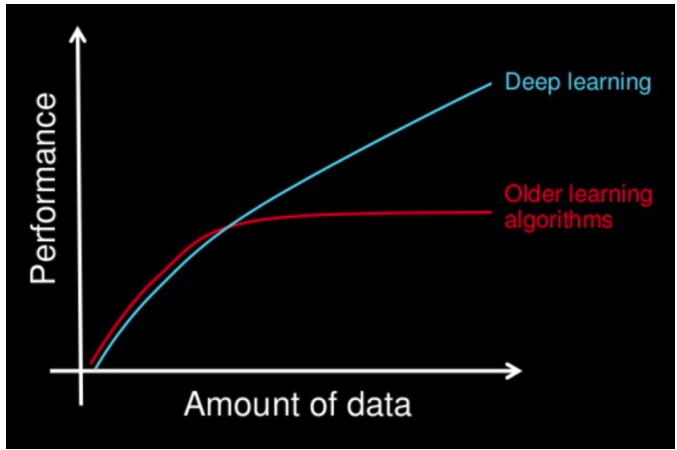
¿Porqué ahora?

- Muchos datos.
- Alto poder de cómputo.



¿Porqué ahora?

- Muchos datos.
- Alto poder de cómputo.
- Mejores algoritmos y modelos.



Propiedades esperadas de las redes neuronales

¿Qué ventajas esperarías de aplicar una red profunda a alguna máquina problema de aprendizaje frente a un clasificador lineal (puro)?

1. Menos posibilidades de *overfitting* (sobre ajuste) de los datos.

Propiedades esperadas de las redes neuronales

¿Qué ventajas esperarías de aplicar una red profunda a alguna máquina problema de aprendizaje frente a un clasificador lineal (puro)?

1. Menos posibilidades de *overfitting* (sobre ajuste) de los datos.
2. Poder capturar funciones de predicción más complejas.

Propiedades esperadas de las redes neuronales

¿Qué ventajas esperarías de aplicar una red profunda a alguna máquina problema de aprendizaje frente a un clasificador lineal (puro)?

1. Menos posibilidades de *overfitting* (sobre ajuste) de los datos.
2. Poder capturar funciones de predicción más complejas.
3. Mejor rendimiento del **conjunto de pruebas** cuando el número de datos es pequeño.

Propiedades esperadas de las redes neuronales

¿Qué ventajas esperarías de aplicar una red profunda a alguna máquina problema de aprendizaje frente a un clasificador lineal (puro)?

1. Menos posibilidades de *overfitting* (sobre ajuste) de los datos.
2. Poder capturar funciones de predicción más complejas.
3. Mejor rendimiento del **conjunto de pruebas** cuando el número de datos es pequeño.
4. Mejor rendimiento del **conjunto de entrenamiento** cuando el número de datos es pequeño.

Propiedades esperadas de las redes neuronales

¿Qué ventajas esperarías de aplicar una red profunda a alguna máquina problema de aprendizaje frente a un clasificador lineal (puro)?

1. Menos posibilidades de *overfitting* (sobre ajuste) de los datos.
2. Poder capturar funciones de predicción más complejas.
3. Mejor rendimiento del **conjunto de pruebas** cuando el número de datos es pequeño.
4. Mejor rendimiento del **conjunto de entrenamiento** cuando el número de datos es pequeño.
5. Mejor rendimiento del **conjunto de pruebas** cuando el número de puntos de datos es grande.

UNIVERSIDAD ADOLFO IBÁÑEZ

Contenidos

Redes Neuronales para Aprendizaje Automático

Función de Hipótesis: Red Neuronal.

Redes Neuronales para Aprendizaje Automático

Función de Hipótesis: Red Neuronal.

Función de Pérdida (*loss function*)**:** Función de pérdida "tradicional". Por ejemplo, función de pérdida logística para clasificación binaria (*logistic loss*).

$$\ell(h_\theta(x), y) = log(1 + e^{(-y \cdot h_\theta(x))})$$

Redes Neuronales para Aprendizaje Automático

Función de Hipótesis: Red Neuronal.

Función de Pérdida (*loss function*)**:** Función de pérdida "tradicional". Por ejemplo, función de pérdida logística para clasificación binaria (*logistic loss*).

$$\ell(h_\theta(x), y) = log(1 + e^{(-y \cdot h_\theta(x))})$$

Optimización: ¿Cómo resolvemos el problema de la optimización?

Redes Neuronales para Aprendizaje Automático

Función de Hipótesis: Red Neuronal.

Función de Pérdida (*loss function*)**:** Función de pérdida "tradicional". Por ejemplo, función de pérdida logística para clasificación binaria (*logistic loss*).

$$\ell(h_\theta(x), y) = log(1 + e^{(-y \cdot h_\theta(x))})$$

Optimización: ¿Cómo resolvemos el problema de la optimización?

Se usa el descenso de gradiente, una variación de la que hemos visto hasta ahora.

Descenso de gradiente estocástico

Un desafío importante para las redes neuronales: Cuando se tiene un gran número de muestras, es calcular los **gradientes** ser intensivos en cuanto su cómputo.

Descenso de gradiente estocástico

Un desafío importante para las redes neuronales: Cuando se tiene un gran número de muestras, es calcular los **gradientes** ser intensivos en cuanto su cómputo.

El descenso de gradiente: Visto hasta ahora, se calcula respecto a la suma sobre todos los ejemplos. Luego, se ajusta a los parámetros:

$$\theta := \theta - \alpha \sum_{i=1}^{m} \nabla_\theta \ell(h_\theta(x^{(i)}, y^{(i)})$$

Descenso de gradiente estocástico

Un desafío importante para las redes neuronales: Cuando se tiene un gran número de muestras, es calcular los **gradientes** ser intensivos en cuanto su cómputo.

El descenso de gradiente: Visto hasta ahora, se calcula respecto a la suma sobre todos los ejemplos. Luego, se ajusta a los parámetros:

$$\theta := \theta - \alpha \sum_{i=1}^{m} \nabla_\theta \ell(h_\theta(x^{(i)}, y^{(i)})$$

Enfoque alternativo: Descenso de gradiente estocástico (SGD), ajustar los parámetros en base a sobre una sola muestra.

$$\theta := \theta - \alpha \nabla_\theta \ell(h_\theta(x^{(i)}, y^{(i)}))$$

y luego repetir estas actualizaciones para todas las muestras.

Descenso de gradiente vs SGD *(Stochastic gradient descent)*

Descenso de gradiente:

- For $i = 1, ..., m$;

$$g^{(i)} \leftarrow \nabla_\theta \ell(h_\theta(x^{(i)}), y^{(i)})$$

Descenso de gradiente estocástico (SGD):

- For $i = 1, ..., m$;

$$\theta \leftarrow \theta - \nabla_\theta \ell(h_\theta(x^{(i)}), y^{(i)})$$

En la práctica, el descenso de gradiente estocástico utiliza una pequeña colección de muestras, no sólo uno, llamado *mini-batch*.

Descenso de gradiente vs SGD *(Stochastic gradient descent)*

Descenso de gradiente:

- For $i = 1, ..., m$;

$$g^{(i)} \leftarrow \nabla_\theta \ell(h_\theta(x^{(i)}), y^{(i)})$$

- Actualizar parámetros:

$$\theta \leftarrow \theta - \alpha \sum_{i=0}^{m} g^{(i)}$$

Descenso de gradiente estocástico (SGD):

- For $i = 1, ..., m$;

$$\theta \leftarrow \theta - \nabla_\theta \ell(h_\theta(x^{(i)}), y^{(i)})$$

En la práctica, el descenso de gradiente estocástico utiliza una pequeña colección de muestras, no sólo uno, llamado *mini-batch*.

Calculando gradientes: *Backpropagation*

¿Cómo calculamos el gradiente $\nabla_\theta \ell(h_\theta(x^{(i)}), y^{(i)})$?.

- Recordar que θ aquí denota un conjunto de parámetros, así que estamos realmente computando gradientes con respecto a todos los elementos de ese conjunto.

Calculando gradientes: *Backpropagation*

¿Cómo calculamos el gradiente $\nabla_\theta \ell(h_\theta(x^{(i)}), y^{(i)})$?.

- Recordar que θ aquí denota un conjunto de parámetros, así que estamos realmente computando gradientes con respecto a todos los elementos de ese conjunto.
- Esto se logra a través del algoritmo de retropropagación.

Calculando gradientes: *Backpropagation*

¿Cómo calculamos el gradiente $\nabla_\theta \ell(h_\theta(x^{(i)}), y^{(i)})$?.

- Recordar que θ aquí denota un conjunto de parámetros, así que estamos realmente computando gradientes con respecto a todos los elementos de ese conjunto.
- Esto se logra a través del algoritmo de retropropagación.
- No cubriremos el algoritmo en detalle, pero la retropropagación es sólo una aplicación de la regla de la cadena (multivariante) del cálculo, más los términos intermedios de ”caching”. Por ejemplo, se producen en el gradiente tanto de W_1 como de W_2

UNIVERSIDAD ADOLFO IBÁÑEZ

Contenidos

Ejemplo

Feedfordward

(Red prealimentada y su uso)

Red poco profunda

Supongamos que tenemos una red la cual tiene tres capas:

- **Capa de entrada:** Con dos neuronas de entradas (dos atributos de entrada).

Red poco profunda

Supongamos que tenemos una red la cual tiene tres capas:

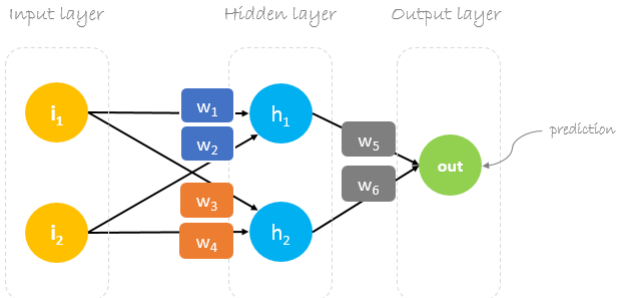
- **Capa de entrada:** Con dos neuronas de entradas (dos atributos de entrada).
- **Una capa oculta:** Una capa oculta con dos neuronas.

Red poco profunda

Supongamos que tenemos una red la cual tiene tres capas:

- **Capa de entrada:** Con dos neuronas de entradas (dos atributos de entrada).
- **Una capa oculta:** Una capa oculta con dos neuronas.
- **Capa de salida:** Una neurona como capa de salida.

Red poco profunda



Arquitectura de red propuesta

https://mefics.org/es/backpropagation-paso-a-paso/

Configuración de los pesos sinápticos

- **Entrenamiento** busca encontrar los pesos sinápticos que minimizan la función de pérdida.

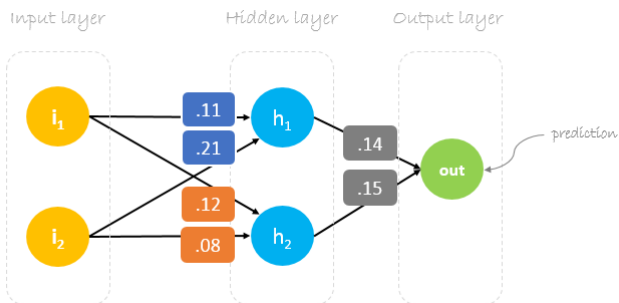
Configuración de los pesos sinápticos

- **Entrenamiento** busca encontrar los pesos sinápticos que minimizan la función de pérdida.
- **Inicialmente**, cuando la **red neuronal artificial no ha sido entrenada**, los pesos sinápticos puede tener asignado cualquier valor. Existen muchas formas de asignar los valores de los pesos iniciales, generalmente, son por azar con diferentes estrategias.

Configuración de los pesos sinápticos

- **Entrenamiento** busca encontrar los pesos sinápticos que minimizan la función de pérdida.
- **Inicialmente**, cuando la **red neuronal artificial no ha sido entrenada**, los pesos sinápticos puede tener asignado cualquier valor. Existen muchas formas de asignar los valores de los pesos iniciales, generalmente, son por azar con diferentes estrategias.
- Los **pesos sinápticos** de la red son los que permiten hacer predicciones en la red.

Pesos sinápticos: Red poco profunda



Red poco profunda con pesos sinápticos

https://mefics.org/es/backpropagation-paso-a-paso/

Conjunto de datos

- Una muestra:

Conjunto de datos

- Una muestra:
  - Dos **atributos** de entrada.

Conjunto de datos

- Una muestra:
  - Dos **atributos** de entrada.
  - Una **etiqueta** de entrada.

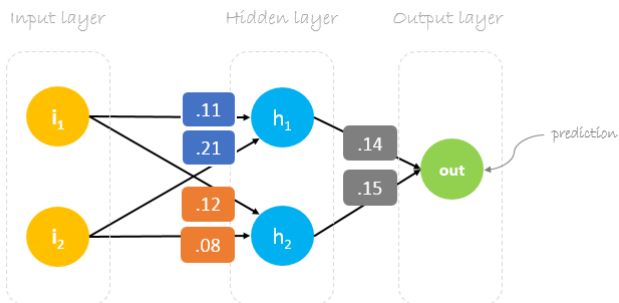
Conjunto de datos

- Una muestra:
  - Dos **atributos** de entrada.
  - Una **etiqueta** de entrada.

Base de datos ficticia

https://mefics.org/es/backpropagation-paso-a-paso/

Red neuronal con pesos sinápticos



Red poco profunda con pesos sinápticos

https://mefics.org/es/backpropagation-paso-a-paso/

Paso *forward*

- Calcular las salidas para **cada neurona** en la **capa oculta**.

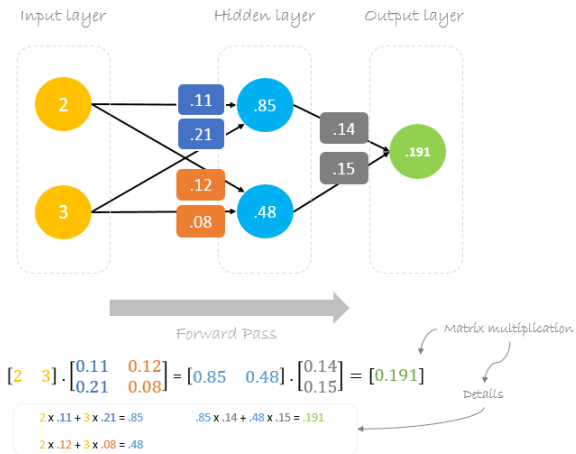
Paso *forward*

- Calcular las salidas para **cada neurona** en la **capa oculta**.
- Calcular la **salida de la red** (predicción) para la única **neurona de salida**.

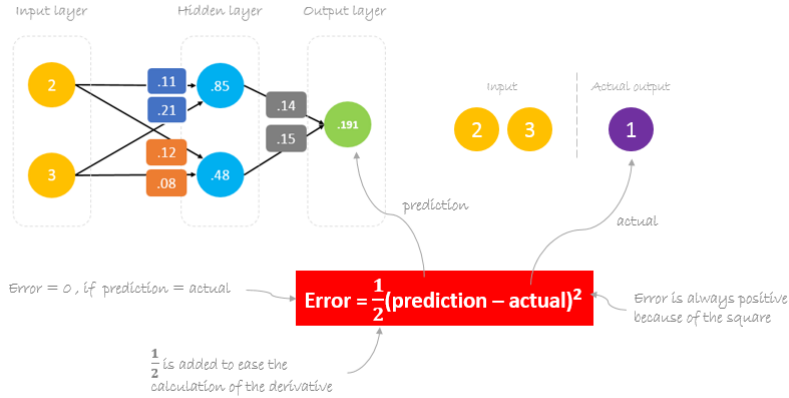
Paso *forward*

- Calcular las salidas para **cada neurona** en la **capa oculta**.
- Calcular la **salida de la red** (predicción) para la única **neurona de salida**.
- Utilizar la salida de la red para **calcular el error** comparando la salida real.

Paso *forward*: Cálculo de la salida predicha



Calculamos el error

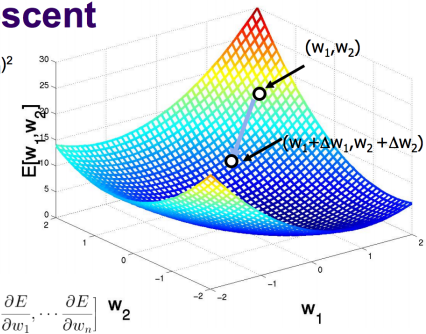


$$\text{Error} = \frac{1}{2}(0.191 - 1.0)^2 = 0.327$$

Reducir el error: ¿Descenso de gradiente?

Gradient Descent

$E[w_1,\ldots,w_n] = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$



Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Reducir el error

$$\text{prediction} = \text{out}$$

$$\text{prediction} = (h_1)\, w_5 + (h_2)\, w_6$$

$h_1 = i_1 w_1 + i_2 w_2$

$h_2 = i_1 w_3 + i_2 w_4$

$$\text{prediction} = (i_1 w_1 + i_2 w_2)\, w_5 + (i_1 w_3 + i_2 w_4)\, w_6$$

to change ***prediction*** value,
we need to change ***weights***

https://mefics.org/es/backpropagation-paso-a-paso/

Actualización de pesos en una red neuronal artificial

Backpropagation

David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams: *Learning representations by back-propagating errors*

Backpropagation (Backward propagation of errors)

- *Backpropagation* es un mecanismo para actualizar los sinápticos de una red multicapa o que contenga más de una neurona.

Backpropagation (Backward propagation of errors)

- *Backpropagation* es un mecanismo para actualizar los sinápticos de una red multicapa o que contenga más de una neurona.
- Es una versión mejorada del **descenso de gradiente**.

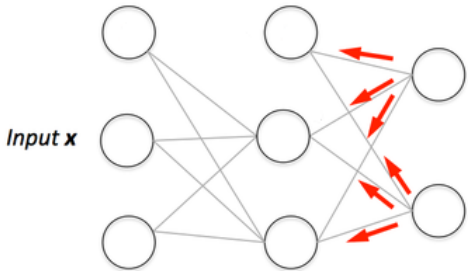
Backpropagation (Backward propagation of errors)

- *Backpropagation* es un mecanismo para actualizar los sinápticos de una red multicapa o que contenga más de una neurona.
- Es una versión mejorada del **descenso de gradiente**.
- Calcula el error de la salida de la red y la propaga hacia atrás, a las neuronas más internas.

Backpropagation (Backward propagation of errors)

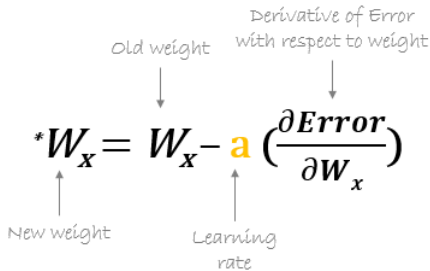
- *Backpropagation* es un mecanismo para actualizar los sinápticos de una red multicapa o que contenga más de una neurona.
- Es una versión mejorada del **descenso de gradiente**.
- Calcula el error de la salida de la red y la propaga hacia atrás, a las neuronas más internas.
- Se busca encontrar una combinación de pesos sinápticos que minimice el error entre predicha y la salida real.

Backpropagation



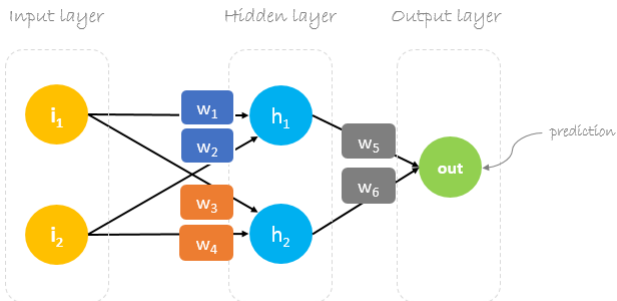
https://mefics.org/es/backpropagation-paso-a-paso/

Actualización de pesos con descenso de gradiente



https://mefics.org/es/backpropagation-paso-a-paso/

Recordemos la notación de la red



Arquitectura de red propuesta

https://mefics.org/es/backpropagation-paso-a-paso/

Actualizamos los pesos w_5 y w_6

- Por ejemplo, para w_6.

$$^{*}W_6 = W_6 - a\left(\frac{\partial Error}{\partial W_6}\right)$$

Calculamos la actualización de w_6

$$\frac{\partial Error}{\partial W_6} = \frac{\partial Error}{\partial prediction} * \frac{\partial prediction}{\partial W_6}$$

chain rule

Error = $\frac{1}{2}$(prediction – actual)^2

prediction = $(i_1 w_1 + i_2 w_2) w_5 + (i_1 w_3 + i_2 w_4) w_6$

$$\frac{\partial Error}{\partial W_6} = \frac{\frac{1}{2}(predictoin - actula)^2}{\partial prediciton} * \frac{\partial (i_1 w_1 + i_2 w_2) w_5 + (i_1 w_3 + i_2 w_4) w_6}{\partial W_6}$$

$$\frac{\partial Error}{\partial W_6} = 2 * \frac{1}{2}(\mathbf{predictoin - actula}) \frac{\partial (predictoin - actula)}{\partial prediciton} * (i_1 w_3 + i_2 w_4)$$

$h_2 = i_1 w_3 + i_2 w_4$

$$\frac{\partial Error}{\partial W_6} = (\mathbf{predictoin - actula}) * (h_2)$$

Δ = prediction – actual ← delta

$$\frac{\partial Error}{\partial W_6} = \Delta h_2$$

Actualizamos los pesos sinápticos w_5 y w_6

$$^{*}W_6 = W_6 - a\,\Delta h_2$$

$$^{*}W_5 = W_5 - a\,\Delta h_1$$

Actualizar los pesos sinápticos w_1, w_2, w_3 y w_4

$$\frac{\partial Error}{\partial W_1} = \frac{\partial Error}{\partial prediction} * \frac{\partial prediction}{\partial h_1} * \frac{\partial h_1}{\partial W_1}$$

chain rule

$$\frac{\partial Error}{\partial W_1} = \frac{\partial \frac{1}{2}(predictoin - actula)^2}{\partial prediciton} * \frac{\partial (h_1)\ w_5 + (h_2)\ w_6}{\partial h_1} * \frac{\partial\ i_1\ w_1 + i_2\ w_2}{\partial w_1}$$

$$\frac{\partial Error}{\partial W_1} = 2 * \frac{1}{2}(\mathbf{predictoin - actula}) \frac{\partial (predictoin - actula)}{\partial prediciton} * (w_5) * (i_1)$$

$$\frac{\partial Error}{\partial W_1} = (\mathbf{predictoin - actula}) * (w_5 i_1)$$

$$\frac{\partial Error}{\partial W_1} = \mathbf{\Delta}\ w_5 i_1$$

Error = $\frac{1}{2}$(prediction – actual)^2

prediction = $(h_1)\ w_5 + (h_2)\ w_6$

$h_1 = i_1 w_1 + i_2 w_2$

Δ = prediction – actual ← delta

Formulación matricial

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - a\,\Delta \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - \begin{bmatrix} a h_1 \Delta \\ a h_2 \Delta \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - a\,\Delta \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} . \begin{bmatrix} w_5 & w_6 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - \begin{bmatrix} a\, i_1 \Delta w_5 & a\, i_1 \Delta w_6 \\ a\, i_2 \Delta w_5 & a\, i_2 \Delta w_6 \end{bmatrix}$$

Paso *backward*

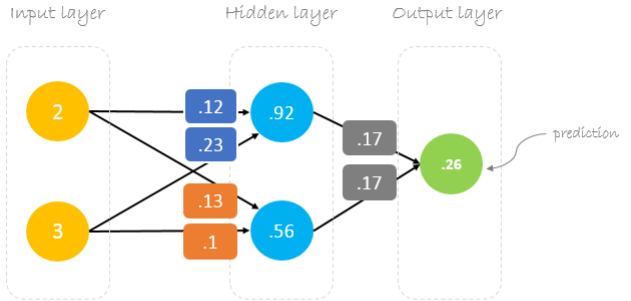
$\Delta = 0.191 - 1 = -0.809$ ← **Delta** = prediction - actual

$a = 0.05$ ← **Learning rate**, we smartly guess this number

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - 0.05(-0.809)\begin{bmatrix} 0.85 \\ 0.48 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - \begin{bmatrix} -0.034 \\ -0.019 \end{bmatrix} = \begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - 0.05(-0.809)\begin{bmatrix} 2 \\ 3 \end{bmatrix}.[0.14 \quad 0.15] = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - \begin{bmatrix} -0.011 & -0.012 \\ -0.017 & -0.018 \end{bmatrix} = \begin{bmatrix} .12 & .13 \\ .23 & .10 \end{bmatrix}$$

Nueva predicción con pesos actualizados



Entrenado redes neuronales artificiales

Lenguajes de programación y librerías

Python y pytorch

Entrenando de redes neuronales en la práctica

- La otra buena noticia es que rara vez tendrás que implementar la algoritmo de retropropagación.

Entrenando de redes neuronales en la práctica

- La otra buena noticia es que rara vez tendrás que implementar la algoritmo de retropropagación.
- Muchas bibliotecas ofrecen métodos para que solo se especifique la red neuronal “forward”, y calcular automáticamente los gradientes necesarios.

Entrenando de redes neuronales en la práctica

- La otra buena noticia es que rara vez tendrás que implementar la algoritmo de retropropagación.
- Muchas bibliotecas ofrecen métodos para que solo se especifique la red neuronal “forward”, y calcular automáticamente los gradientes necesarios.
- Ejemplo: Tensorflow, Pytorch.

UNIVERSIDAD ADOLFO IBÁÑEZ

Contenidos

Arquitecturas especializadas

- Hasta el momento hemos visto que las redes totalmente conectadas (*fully connected*) pueden implementar una red profunda. En la práctica, el potencial de las redes profundas **no se debe al uso de redes neuronales totalmente conectadas**.

Arquitecturas especializadas

- Hasta el momento hemos visto que las redes totalmente conectadas (*fully connected*) pueden implementar una red profunda. En la práctica, el potencial de las redes profundas **no se debe al uso de redes neuronales totalmente conectadas**.
- Por otro lado, el mayor éxito del uso de esta redes se debe a dos tipos de arquitecturas:

Arquitecturas especializadas

- Hasta el momento hemos visto que las redes totalmente conectadas (*fully connected*) pueden implementar una red profunda. En la práctica, el potencial de las redes profundas **no se debe al uso de redes neuronales totalmente conectadas**.
- Por otro lado, el mayor éxito del uso de esta redes se debe a dos tipos de arquitecturas:
  - **Redes neuronales convolucionales**.

Arquitecturas especializadas

- Hasta el momento hemos visto que las redes totalmente conectadas (*fully connected*) pueden implementar una red profunda. En la práctica, el potencial de las redes profundas **no se debe al uso de redes neuronales totalmente conectadas**.
- Por otro lado, el mayor éxito del uso de esta redes se debe a dos tipos de arquitecturas:
  - **Redes neuronales convolucionales**.
  - **Redes neuronales recurrentes**.

Problema: Redes neuronales totalmente conectadas

- Si consideramos una imagen de entrada 256×256 (RGB), significa que la dimensión del espacio de entrada es aproximadamente 200,000.

Problema: Redes neuronales totalmente conectadas

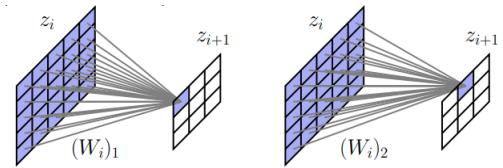
- Si consideramos una imagen de entrada 256×256 (RGB), significa que la dimensión del espacio de entrada es aproximadamente 200,000.
- Una **red profunda completamente conectada** requeriría un gran número de parámetros, y que probablemente se sobreajuste (*overfitting*).

Problema: Redes neuronales totalmente conectadas

- Si consideramos una imagen de entrada 256×256 (RGB), significa que la dimensión del espacio de entrada es aproximadamente 200,000.
- Una **red profunda completamente conectada** requeriría un gran número de parámetros, y que probablemente se sobreajuste (*overfitting*).
- Una red profunda genérica tampoco capta las invariancias "naturales" que existen en las imágenes (ubicación, escala) u otro tipo de señales.

Problema: Redes neuronales totalmente conectadas

- Si consideramos una imagen de entrada 256×256 (RGB), significa que la dimensión del espacio de entrada es aproximadamente 200,000.
- Una **red profunda completamente conectada** requeriría un gran número de parámetros, y que probablemente se sobreajuste (*overfitting*).
- Una red profunda genérica tampoco capta las invariancias "naturales" que existen en las imágenes (ubicación, escala) u otro tipo de señales.



Redes neuronales convolucionales (I)

- Una de aplicaciones más comunes donde se utilizan redes neuronales convolucionales es el área de la visión por computador.

`https://setosa.io/ev/image-kernels/`

Redes neuronales convolucionales (I)

- Una de aplicaciones más comunes donde se utilizan redes neuronales convolucionales es el área de la visión por computador.
- La naturaleza de la red convulucional se base en el concepto de correlación y convolución utilizada para imágenes y señales.

`https://setosa.io/ev/image-kernels/`

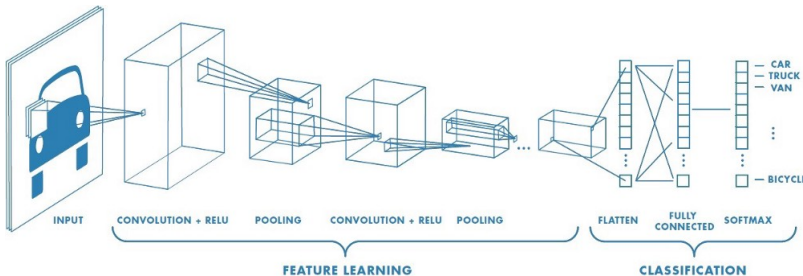
Redes neuronales convolucionales (I)

- Una de aplicaciones más comunes donde se utilizan redes neuronales convolucionales es el área de la visión por computador.
- La naturaleza de la red convulucional se base en el concepto de correlación y convolución utilizada para imágenes y señales.
- Es importante entender el concepto de *kernel* sobre una imagen.

`https://setosa.io/ev/image-kernels/`

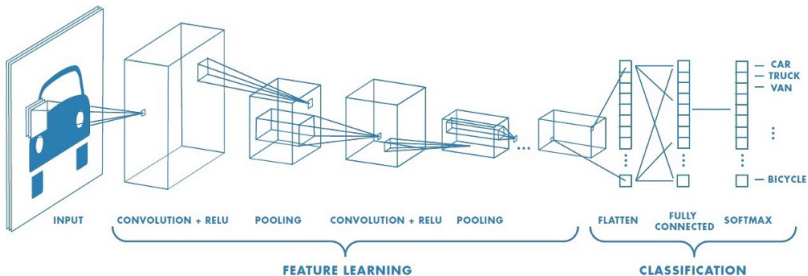
Redes neuronales convolucionales (I)

- La arquitectura de una **red neuronal convolucional** es análoga a la del patrón de conectividad de las neuronas del cerebro humano y se inspiró en la organización de la Corteza Visual.



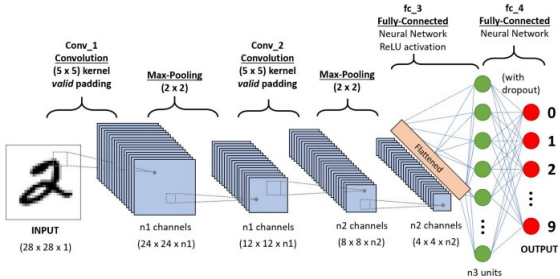
Redes neuronales convolucionales (I)

- La arquitectura de una **red neuronal convolucional** es análoga a la del patrón de conectividad de las neuronas del cerebro humano y se inspiró en la organización de la Corteza Visual.
- Las neuronas individuales responden a los estímulos **sólo en una región restringida** del campo visual conocida como Campo Receptivo.



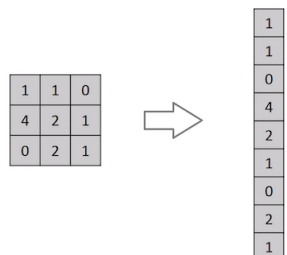
Redes neuronales convolucionales(II)

- Las diferentes arquitecturas de una red neuronal profunda están compuestas por diferentes capas de neuronas especializadas como las: Convolucionales, *Pooling* o totalmente conectadas.



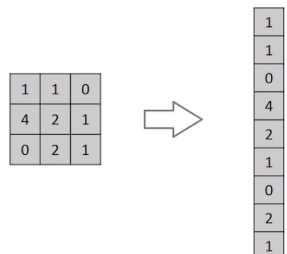
Diferencias entre utilizar una imagen o vector

- Una **red neuronal convolucional** es capaz de capturar con éxito las dependencias espaciales y temporales en una imagen mediante la aplicación de los filtros.



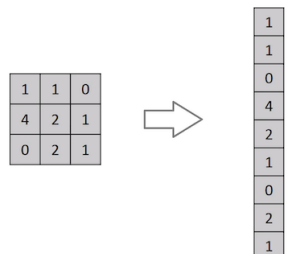
Diferencias entre utilizar una imagen o vector

- Una **red neuronal convolucional** es capaz de capturar con éxito las dependencias espaciales y temporales en una imagen mediante la aplicación de los filtros.
- La arquitectura se ajusta mejor al conjunto de datos de la imagen debido a la reducción del número de parámetros implicados y a la reutilización de los pesos.



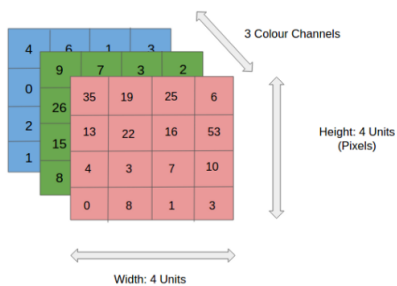
Diferencias entre utilizar una imagen o vector

- Una **red neuronal convolucional** es capaz de capturar con éxito las dependencias espaciales y temporales en una imagen mediante la aplicación de los filtros.
- La arquitectura se ajusta mejor al conjunto de datos de la imagen debido a la reducción del número de parámetros implicados y a la reutilización de los pesos.
- La red neuronal convolucional captura las variabiliades de posición y estructura.



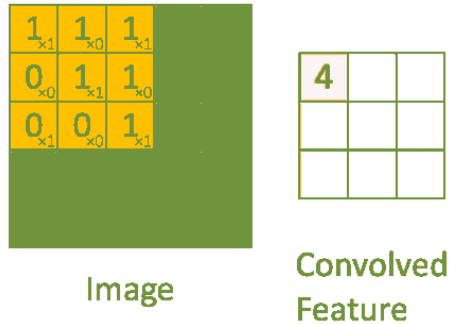
Capa convolucional: Imagen

- La capa convolucional reduce las imágenes a una forma que sea más fácil de procesar, sin perder las características que son críticas para obtener una buena predicción.



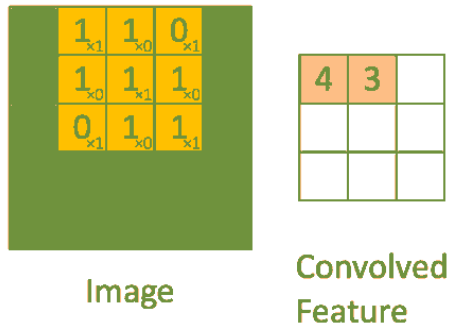
Capa convolucional: Filtros (*kernels*)

- Dimensiones de la imagen = 5 (altura) x 5 (anchura) x 1 (número de canales, por ejemplo RGB).
- En la figura, la matriz verde se asemeja a una imagen de entrada de 5x5x1, I.
- La matriz amarilla simula un kernel/filtro, K.
- La matriz blanca muestra el resultado de aplicar el filtro.



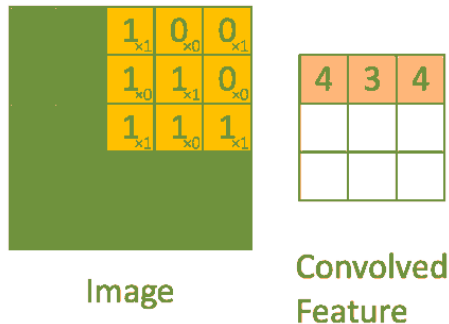
Capa convolucional: Filtros (*kernels*)

- Dimensiones de la imagen = 5 (altura) x 5 (anchura) x 1 (número de canales, por ejemplo RGB).
- En la figura, la matriz verde se asemeja a una imagen de entrada de 5x5x1, I.
- La matriz amarilla simula un kernel/filtro, K.
- La matriz blanca muestra el resultado de aplicar el filtro.



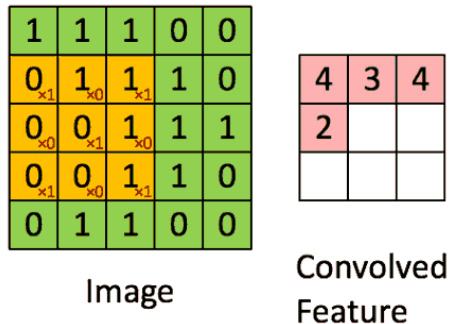
Capa convolucional: Filtros (*kernels*)

- Dimensiones de la imagen = 5 (altura) x 5 (anchura) x 1 (número de canales, por ejemplo RGB).
- En la figura, la matriz verde se asemeja a una imagen de entrada de 5x5x1, I.
- La matriz amarilla simula un kernel/filtro, K.
- La matriz blanca muestra el resultado de aplicar el filtro.



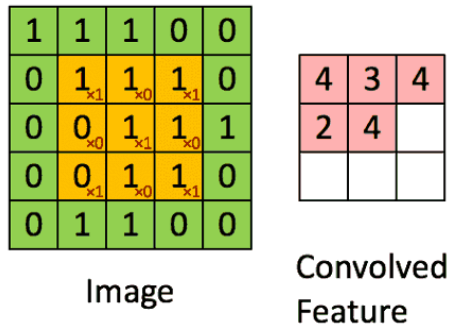
Capa convolucional: Filtros (*kernels*)

- Dimensiones de la imagen = 5 (altura) x 5 (anchura) x 1 (número de canales, por ejemplo RGB).
- En la figura, la matriz verde se asemeja a una imagen de entrada de 5x5x1, I.
- La matriz amarilla simula un kernel/filtro, K.
- La matriz blanca muestra el resultado de aplicar el filtro.



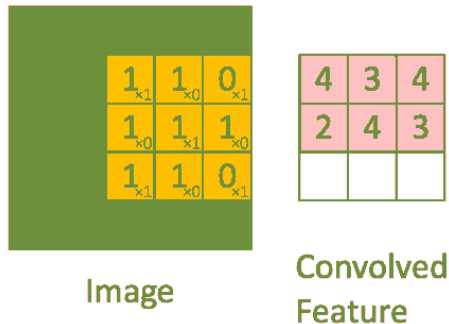
Capa convolucional: Filtros (*kernels*)

- Dimensiones de la imagen = 5 (altura) x 5 (anchura) x 1 (número de canales, por ejemplo RGB).
- En la figura, la matriz verde se asemeja a una imagen de entrada de 5x5x1, I.
- La matriz amarilla simula un kernel/filtro, K.
- La matriz blanca muestra el resultado de aplicar el filtro.



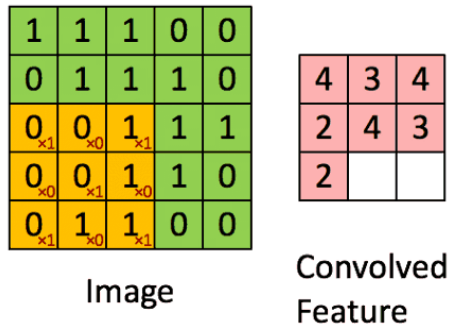
Capa convolucional: Filtros (*kernels*)

- Dimensiones de la imagen = 5 (altura) x 5 (anchura) x 1 (número de canales, por ejemplo RGB).
- En la figura, la matriz verde se asemeja a una imagen de entrada de 5x5x1, I.
- La matriz amarilla simula un kernel/filtro, K.
- La matriz blanca muestra el resultado de aplicar el filtro.



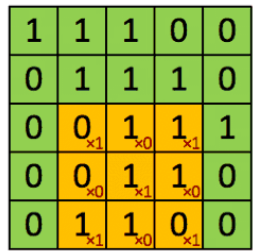
Capa convolucional: Filtros (*kernels*)

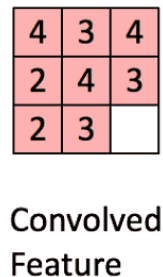
- Dimensiones de la imagen = 5 (altura) x 5 (anchura) x 1 (número de canales, por ejemplo RGB).
- En la figura, la matriz verde se asemeja a una imagen de entrada de 5x5x1, I.
- La matriz amarilla simula un kernel/filtro, K.
- La matriz blanca muestra el resultado de aplicar el filtro.



Capa convolucional: Filtros (*kernels*)

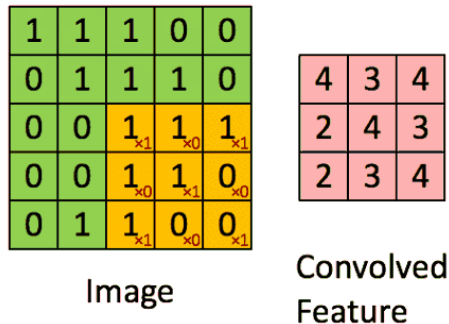
- Dimensiones de la imagen = 5 (altura) x 5 (anchura) x 1 (número de canales, por ejemplo RGB).
- En la figura, la matriz verde se asemeja a una imagen de entrada de 5x5x1, I.
- La matriz amarilla simula un kernel/filtro, K.
- La matriz blanca muestra el resultado de aplicar el filtro.



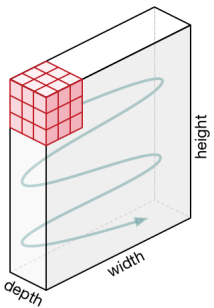


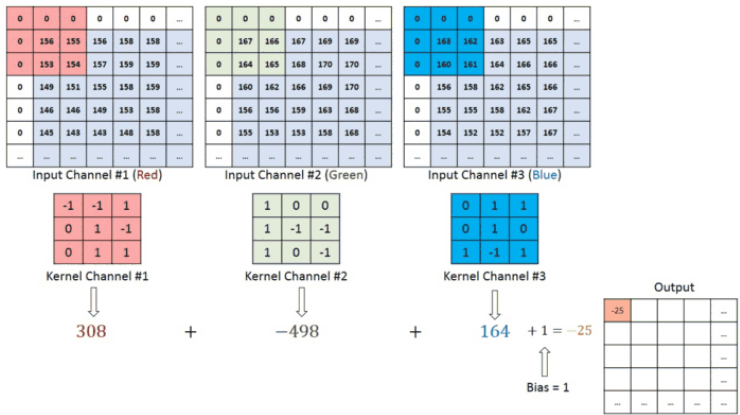
Capa convolucional: Filtros (*kernels*)

- Dimensiones de la imagen = 5 (altura) x 5 (anchura) x 1 (número de canales, por ejemplo RGB).
- En la figura, la matriz verde se asemeja a una imagen de entrada de 5x5x1, I.
- La matriz amarilla simula un kernel/filtro, K.
- La matriz blanca muestra el resultado de aplicar el filtro.

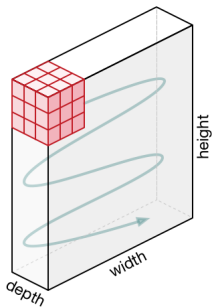


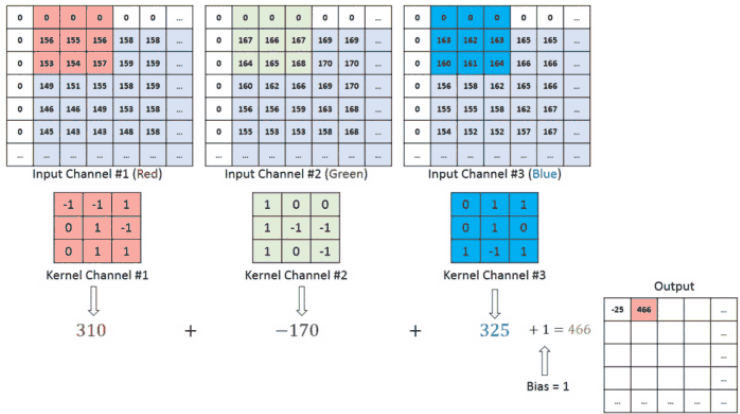
Capa convolucional: Imagen RGB



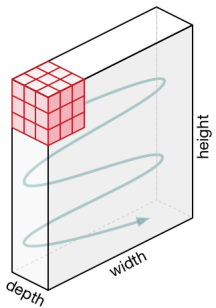


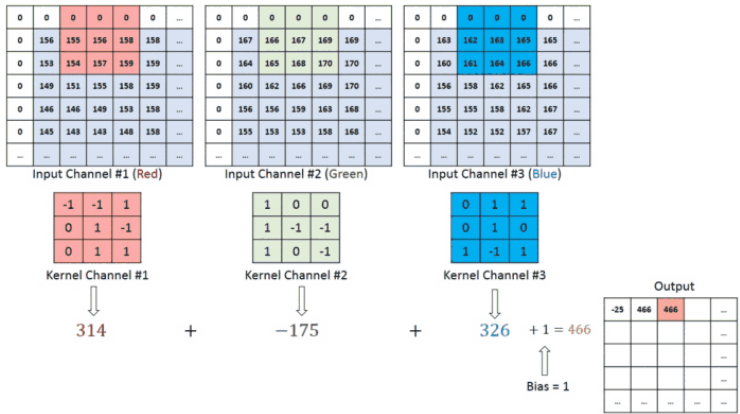
Capa convolucional: Imagen RGB



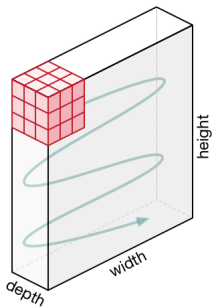


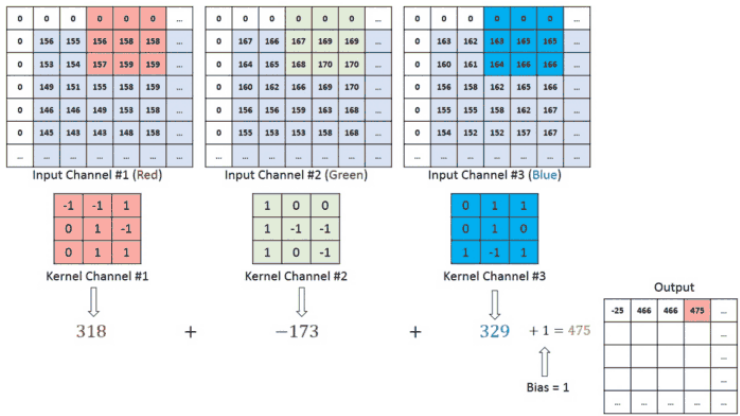
Capa convolucional: Imagen RGB



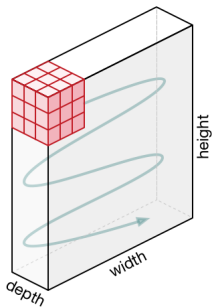


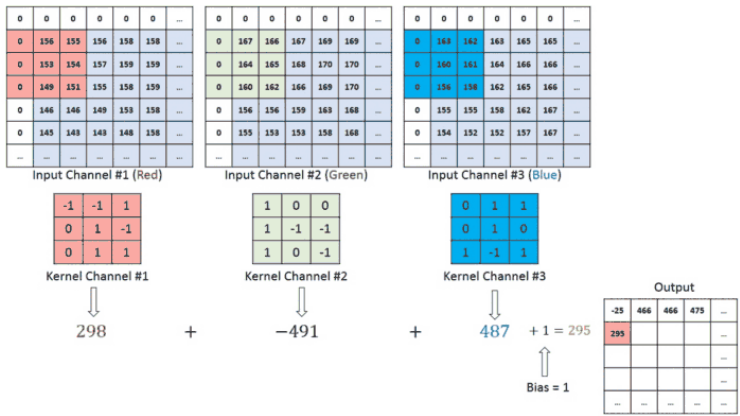
Capa convolucional: Imagen RGB



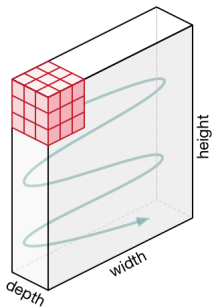


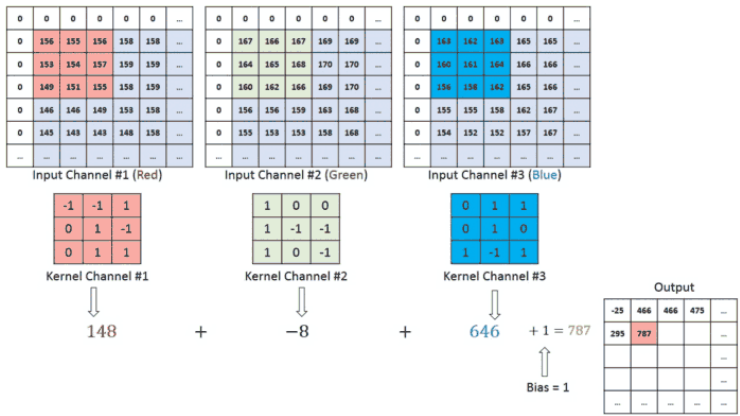
Capa convolucional: Imagen RGB



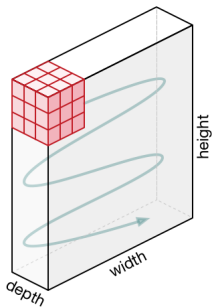


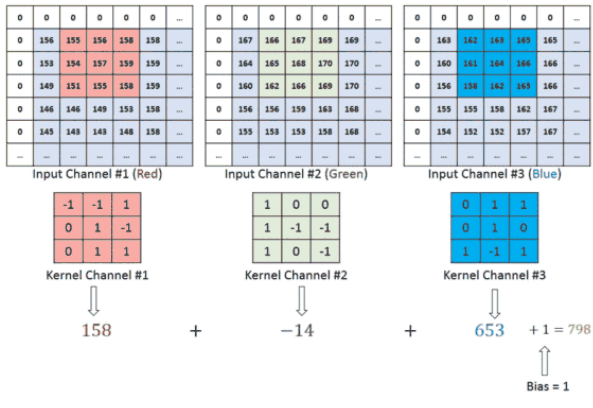
Capa convolucional: Imagen RGB



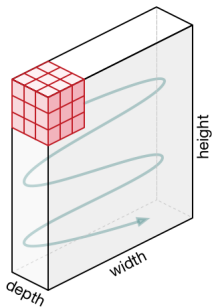


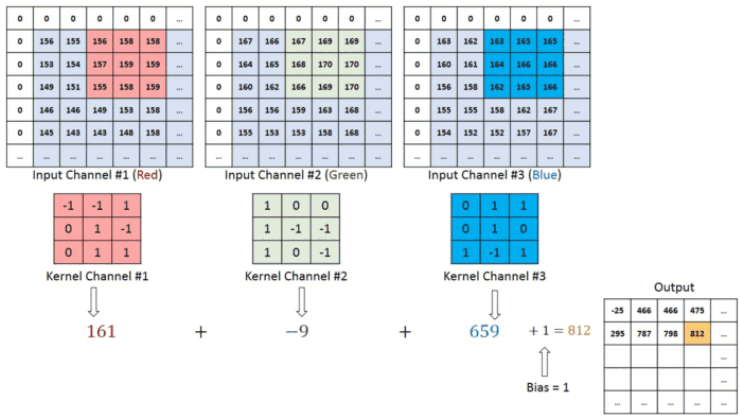
Capa convolucional: Imagen RGB





Capa convolucional: Imagen RGB

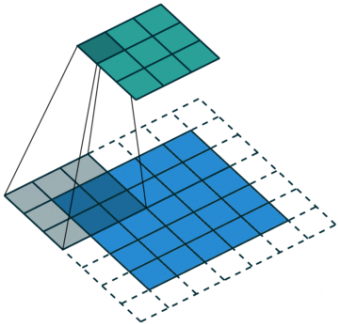




Capa convolucional: Resultados

- El objetivo de la operación de convolución es extraer las características de alto nivel, como los bordes, de la imagen de entrada.
- La capa convolucional no se limita a solo una. Generalmente, la **primera capa convolucional** es responsable de capturar las características de bajo nivel como los bordes, el color, la orientación del gradiente, etc.
- En la medida que se agregan más capas convolucionales, la arquitectura se adapta para obtener características de alto nivel.

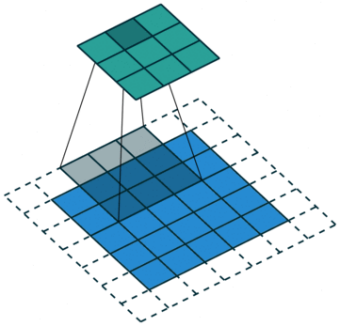
Capa convolucional: *Stride* y *padding*



Stride

Padding

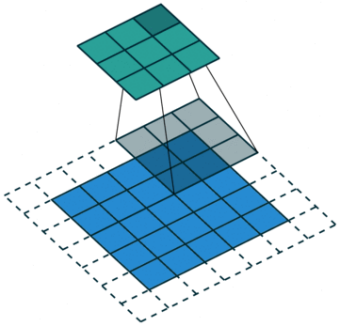
Capa convolucional: *Stride* y *padding*



Stride

Padding

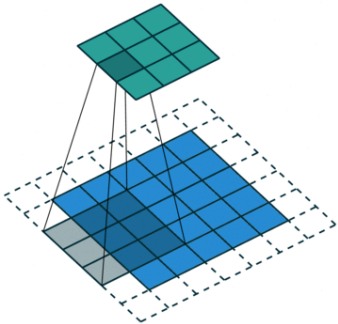
Capa convolucional: *Stride* y *padding*



Stride

Padding

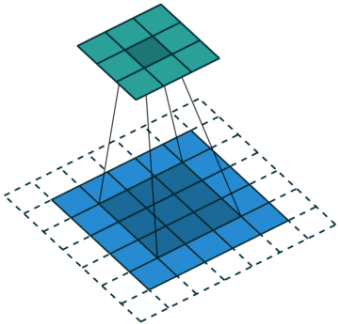
Capa convolucional: *Stride* y *padding*



Stride

Padding

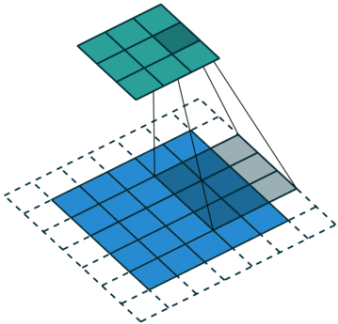
Capa convolucional: *Stride* y *padding*



Stride

Padding

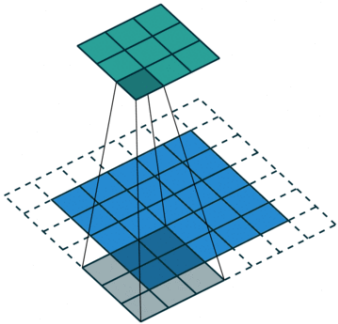
Capa convolucional: *Stride* y *padding*



Stride

Padding

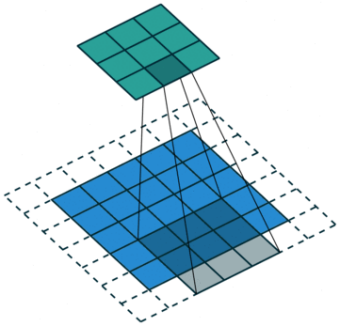
Capa convolucional: *Stride* y *padding*



Stride

Padding

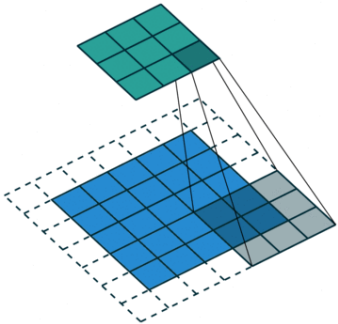
Capa convolucional: *Stride* y *padding*



Stride

Padding

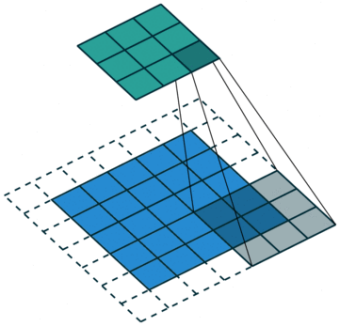
Capa convolucional: *Stride* y *padding*



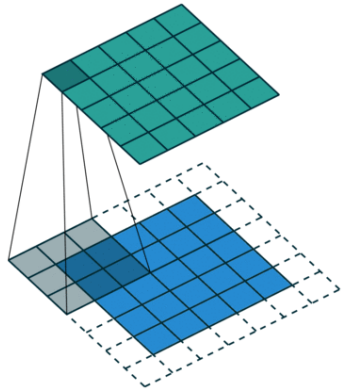
Stride

Padding

Capa convolucional: *Stride* y *padding*

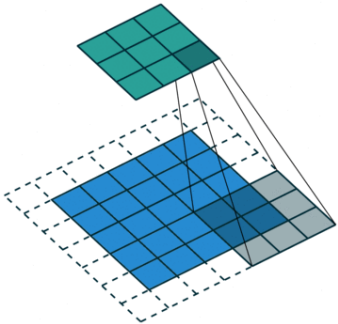


Stride

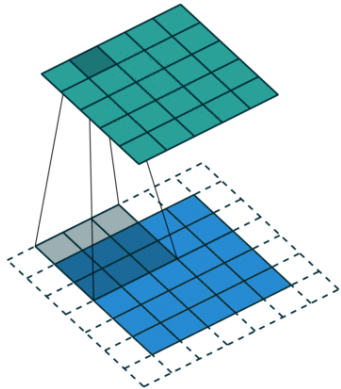


Padding

Capa convolucional: *Stride* y *padding*

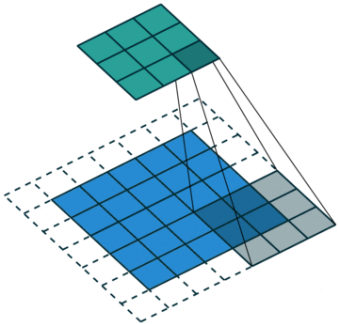


Stride

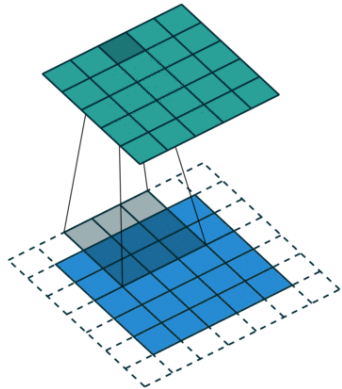


Padding

Capa convolucional: *Stride* y *padding*

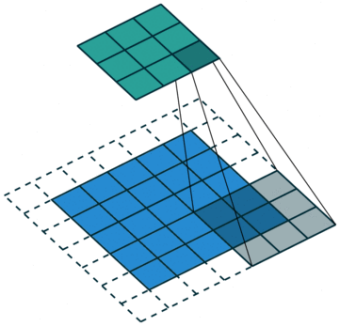


Stride

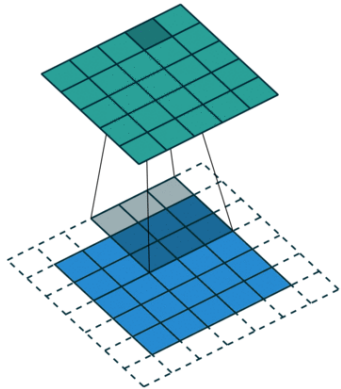


Padding

Capa convolucional: *Stride* y *padding*

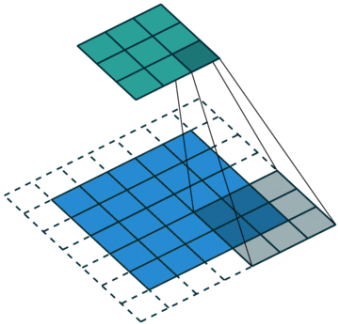


Stride

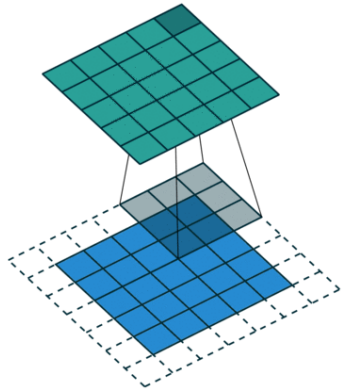


Padding

Capa convolucional: *Stride* y *padding*

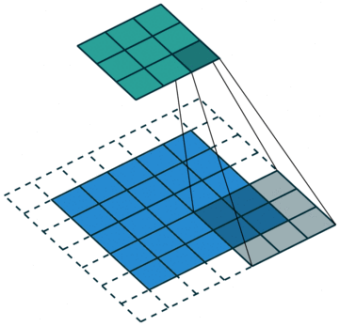


Stride

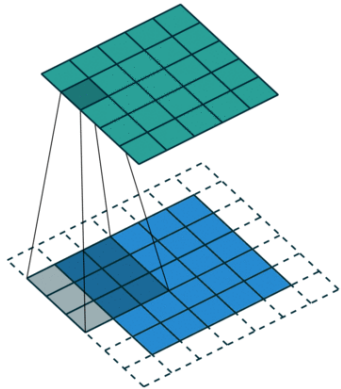


Padding

Capa convolucional: *Stride* y *padding*

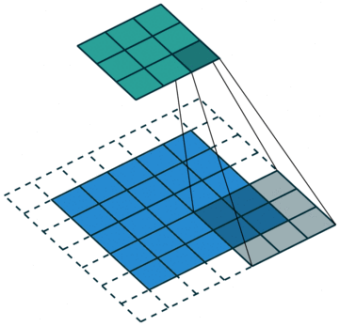


Stride

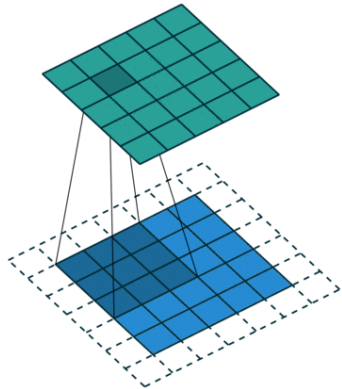


Padding

Capa convolucional: *Stride* y *padding*

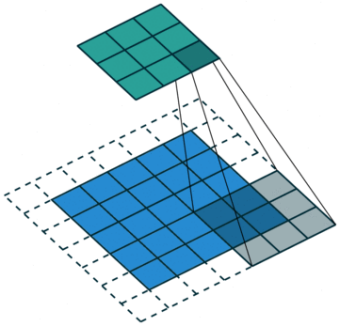


Stride

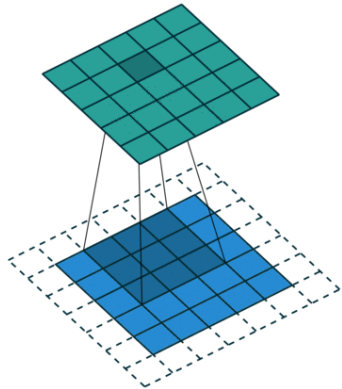


Padding

Capa convolucional: *Stride* y *padding*

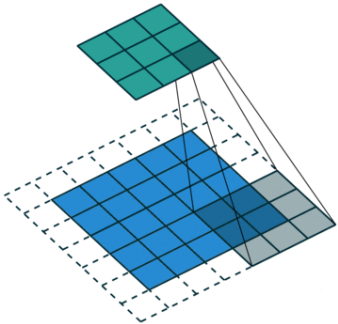


Stride

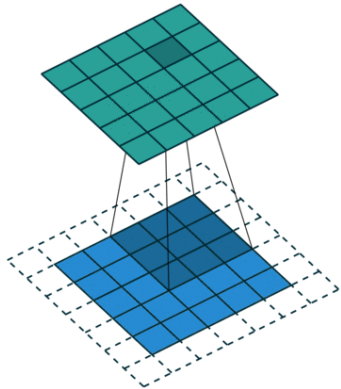


Padding

Capa convolucional: *Stride* y *padding*

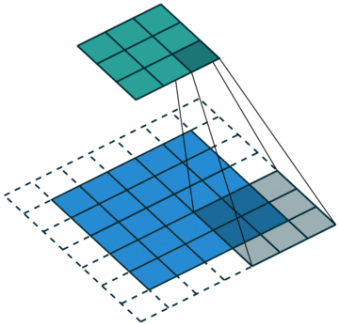


Stride

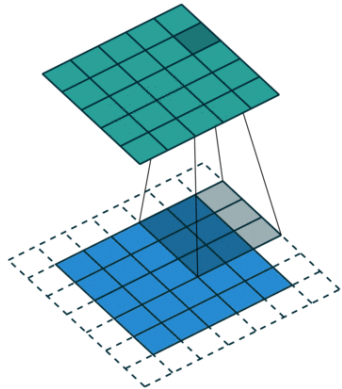


Padding

Capa convolucional: *Stride* y *padding*

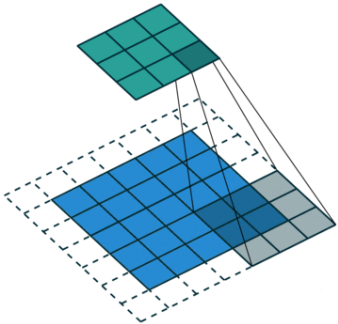


Stride

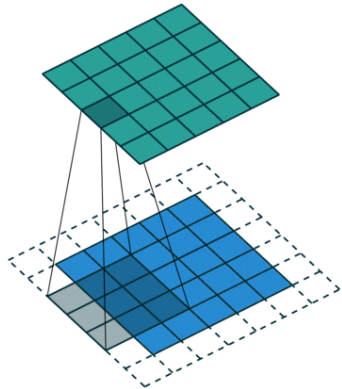


Padding

Capa convolucional: *Stride* y *padding*

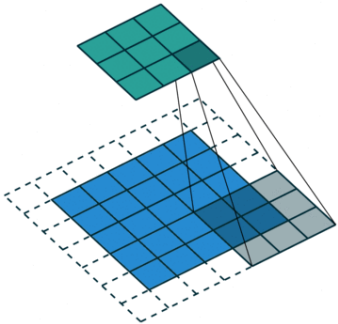


Stride

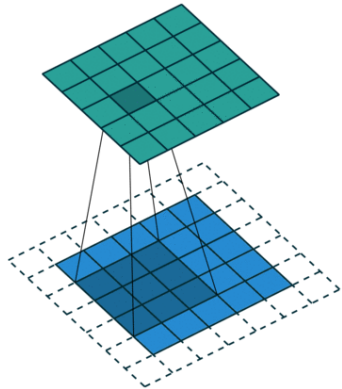


Padding

Capa convolucional: *Stride* y *padding*

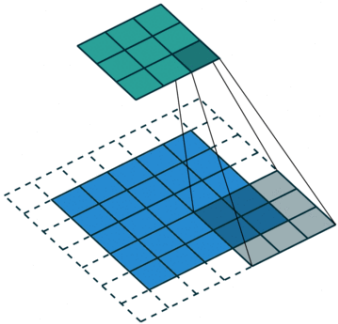


Stride

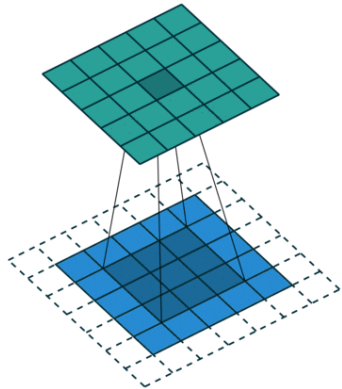


Padding

Capa convolucional: *Stride* y *padding*

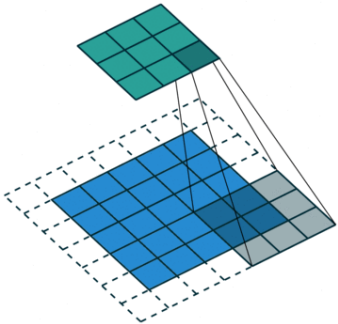


Stride

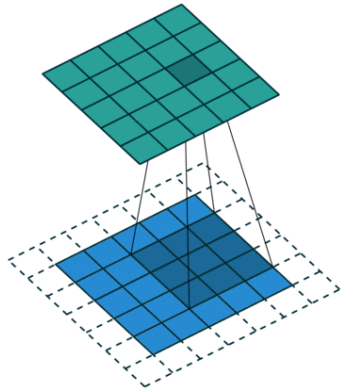


Padding

Capa convolucional: *Stride* y *padding*

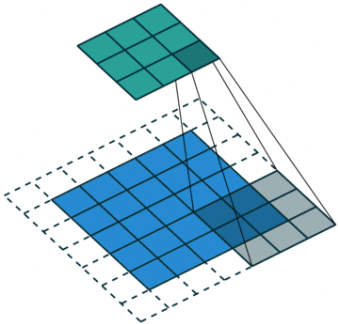


Stride

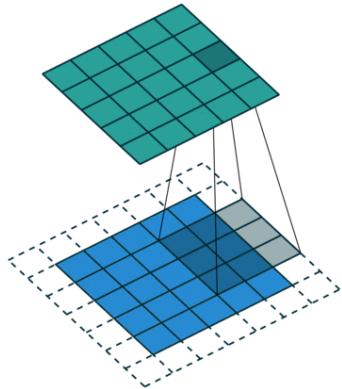


Padding

Capa convolucional: *Stride* y *padding*

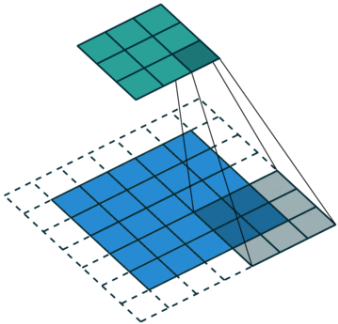


Stride

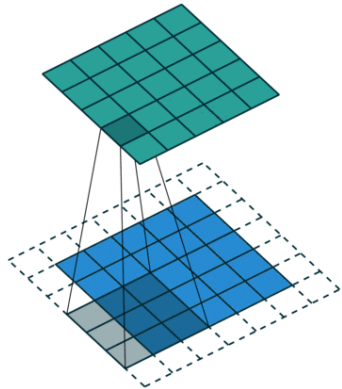


Padding

Capa convolucional: *Stride* y *padding*

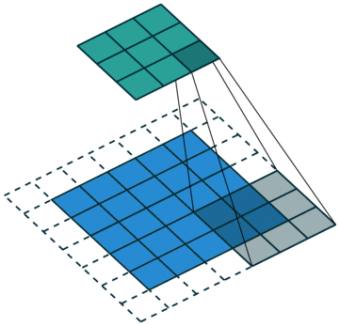


Stride

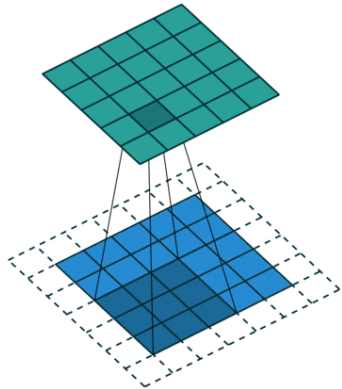


Padding

Capa convolucional: *Stride* y *padding*

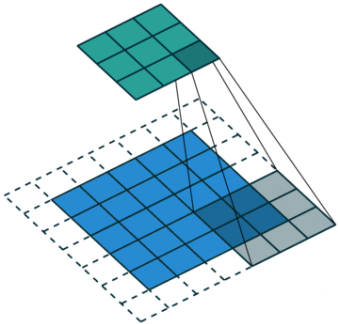


Stride

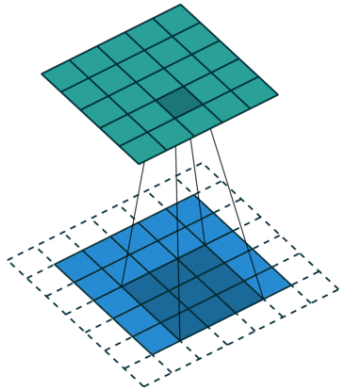


Padding

Capa convolucional: *Stride* y *padding*

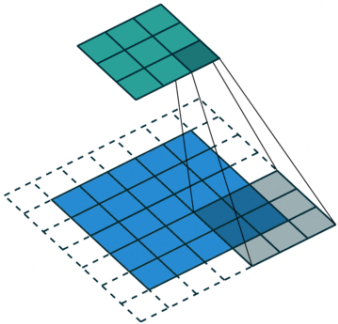


Stride

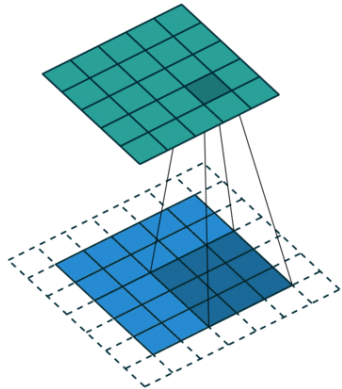


Padding

Capa convolucional: *Stride* y *padding*

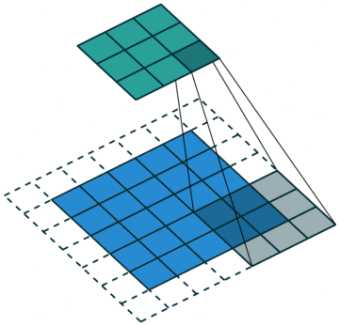


Stride

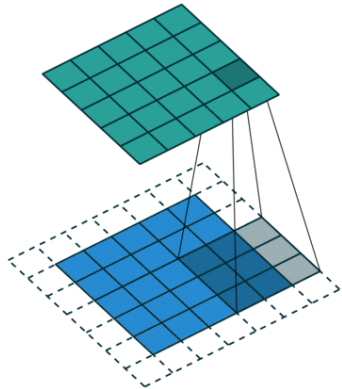


Padding

Capa convolucional: *Stride* y *padding*

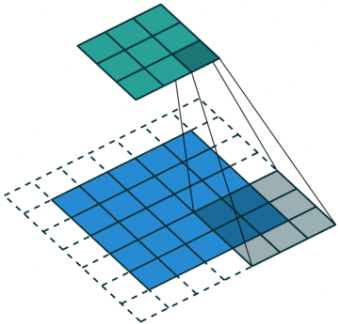


Stride

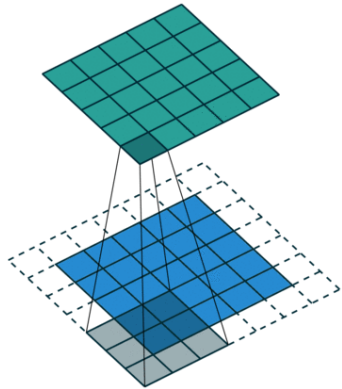


Padding

Capa convolucional: *Stride* y *padding*

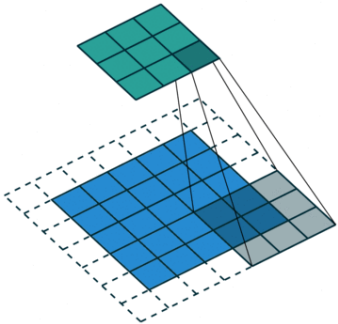


Stride

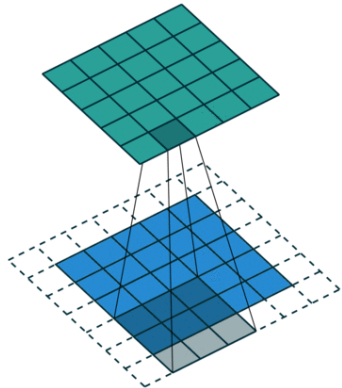


Padding

Capa convolucional: *Stride* y *padding*

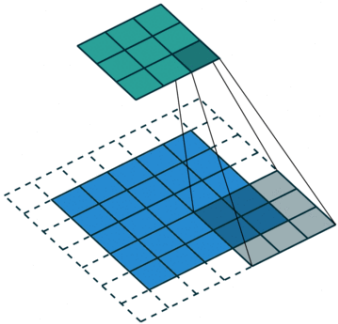


Stride

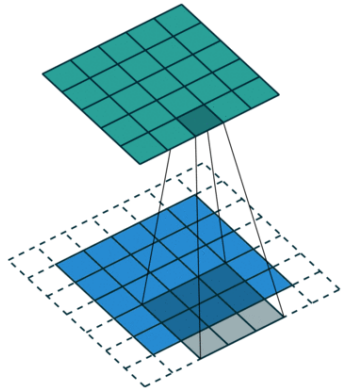


Padding

Capa convolucional: *Stride* y *padding*

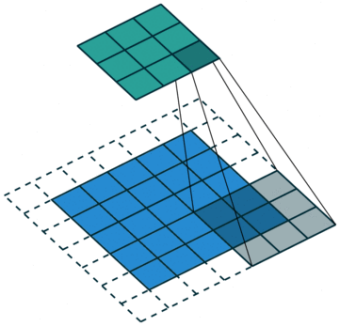


Stride

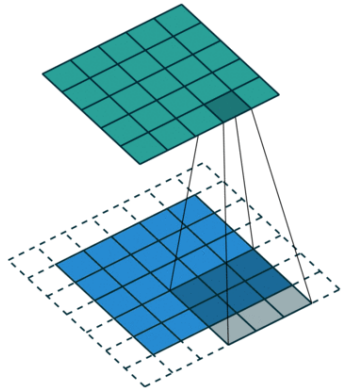


Padding

Capa convolucional: *Stride* y *padding*

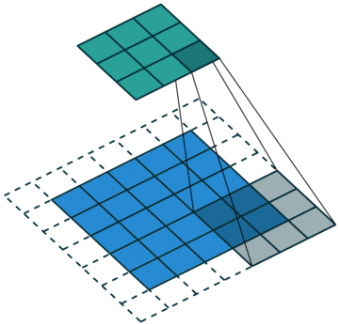


Stride

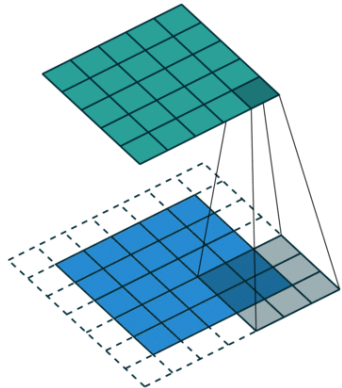


Padding

Capa convolucional: *Stride* y *padding*



Stride



Padding

Capa de agrupación (*Pooling layer*): Definición

- Similar a la **Capa Convolucional**, la **Capa de agrupación** es responsable de reducir el tamaño espacial de las características convolucionales.

Capa de agrupación (*Pooling layer*): Definición

- Similar a la **Capa Convolucional**, la **Capa de agrupación** es responsable de reducir el tamaño espacial de las características convolucionales.
- Esto es para disminuir la potencia de cálculo necesaria para procesar los datos por medio de la reducción de la dimensionalidad.

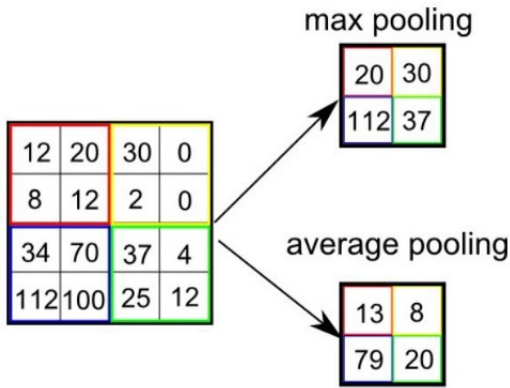
Capa de agrupación (*Pooling layer*): Definición

- Similar a la **Capa Convolucional**, la **Capa de agrupación** es responsable de reducir el tamaño espacial de las características convolucionales.
- Esto es para disminuir la potencia de cálculo necesaria para procesar los datos por medio de la reducción de la dimensionalidad.
- Además, es útil para extraer los características dominantes que son invariantes a rotacionales y translaciones, mejorando la eficiencia del proceso de entrenamiento.

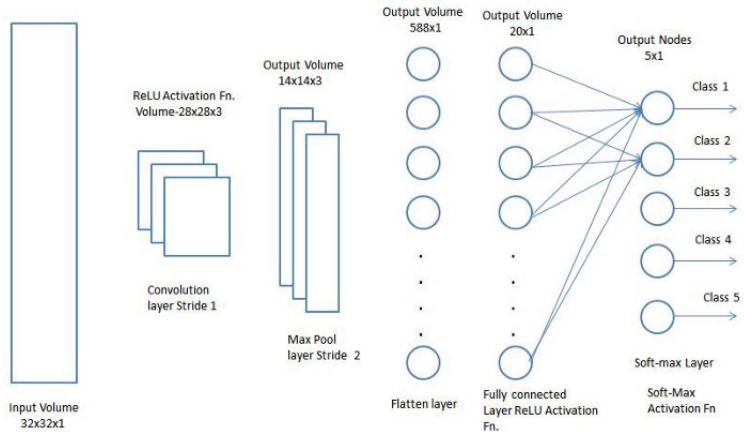
Capa de agrupación (*Pooling layer*): Definición

- Similar a la **Capa Convolucional**, la **Capa de agrupación** es responsable de reducir el tamaño espacial de las características convolucionales.
- Esto es para disminuir la potencia de cálculo necesaria para procesar los datos por medio de la reducción de la dimensionalidad.
- Además, es útil para extraer los características dominantes que son invariantes a rotacionales y translaciones, mejorando la eficiencia del proceso de entrenamiento.
- Hay dos tipos de Pooling: *Max Pooling* y *Average Pooling*.

Capa de agrupación (*Pooling layer*): *Max pooling* y *average pooling*



Red neuronal profunda general



UNIVERSIDAD ADOLFO IBÁÑEZ

Referencias

- **Pattern Recognition and Machine Learning.** Christopher M. Bishop. Springer. 2006.
- **The Elements of Statistical Learning: Data Mining, Inference, and Prediction.** Trevor Hastie, Robert Tibshirani, Jerome Friedman. Springer. 2016.
- **Practical Data Science: Deep learning**. J. Zico Kolter. `http://www.datasciencecourse.org/slides/deep_learning.pdf`
- **A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way**. Sumit Saha. `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53`

UNIVERSIDAD ADOLFO IBÁÑEZ

¿Preguntas?