Redes convolucionales II

Visión por Computador, curso 2024-2025

Silvia Martín Suazo, silvia.martin@u-tad.com

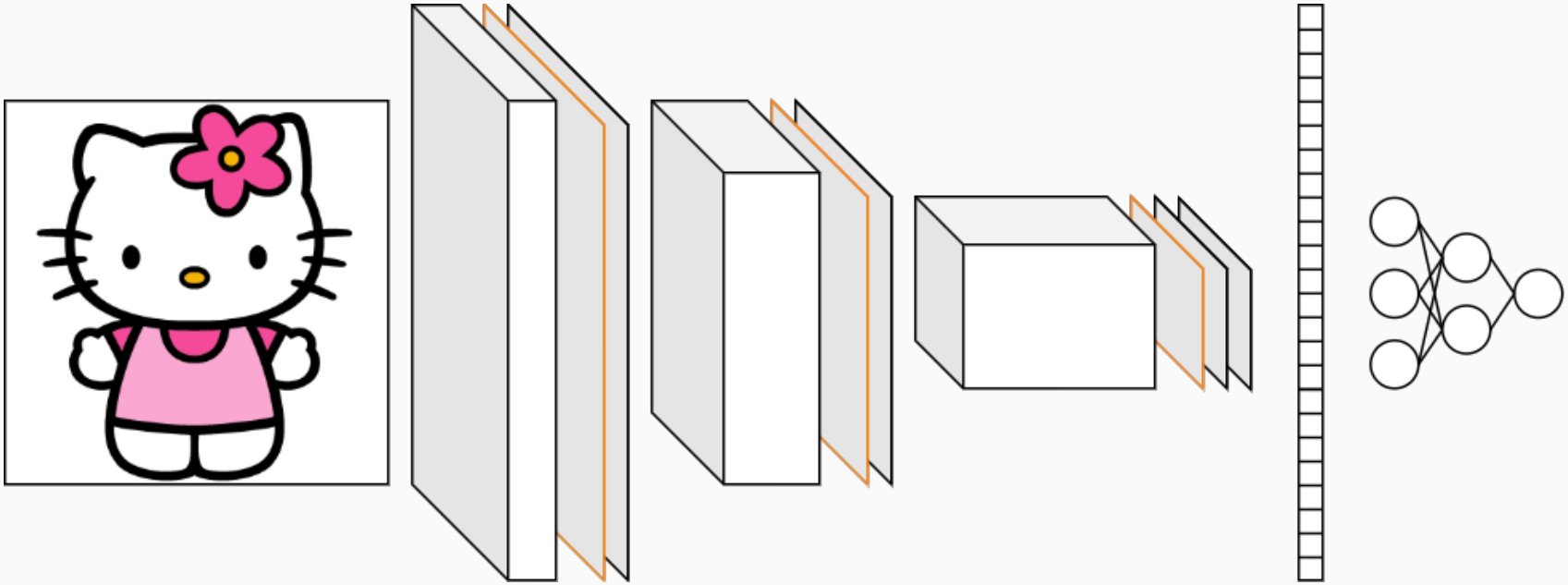
6 de noviembre de 2024

U-tad | Centro Universitario de Tecnología y Arte Digital

Activación de la capa convolucional

Activación de la capa convolucional

- Input
- Conv2D
- Activation
- Pooling
- Flatten
- Dense
- Output



Activación de la capa convolucional

La función de activación de las capas convolucionales se escoge de la misma manera que para las redes neuronales tradicionales. Se utiliza para introducir no-linealidad en los datos tras la convolución (que es una operación lineal) y de esta forma aprender relaciones complejas.

Generalmente, se utiliza ReLU, ya que evita problemas derivados del gradiente, y LeakyReLU, que previene la neurona muerta.

Existen otras capas como la ELU, cuya principal desventaja es su lenta computación en comparación con las anteriores.

Se recomienda el siguiente cheatsheet para estudiar las distintas funciones de activación.

Activación de la capa convolucional en Keras

Como se vio anteriormente, en Keras esta función de activación se indica como un parámetro de la capa de convolución correspondiente.

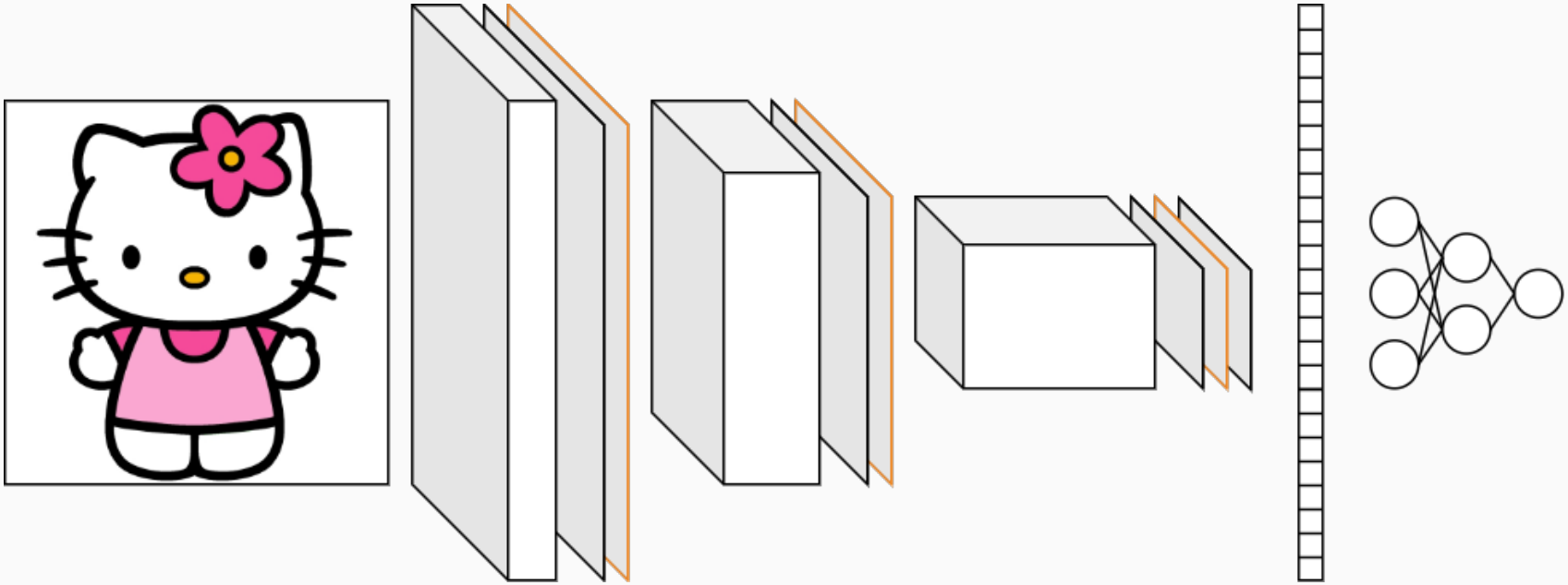
Otra forma de realizar la activación sin indicarla como parámetro de la capa de convolución, es con una capa de activación.

```
Ex: keras.layers.Activation('relu')
```

Pooling

Esquema de construcción de redes convolucionales

- Input
- Conv2D
- Activation
- Pooling y dropout
- Flatten
- Dense
- Output



Pooling

Una vez los datos pasan por las capas convolucionales, se realiza la operación denominada como pooling. Su principal función es la reducción de dimensionalidad de los mapas de características. De esta forma conseguimos:

1. Reducir el número de parámetros de la red. Es decir, disminuimos la capacidad del modelo para que el modelo aprenda exclusivamente las características más destacables y evitar el sobreajuste.
2. Reducir el tiempo de procesamiento y carga computacional.

Además, conseguimos que la red sea invariante a la traslación gracias a que se encuentra el valor más representativo de cada región sin importar la ubicación exacta.

Reducción dimensional con Pooling

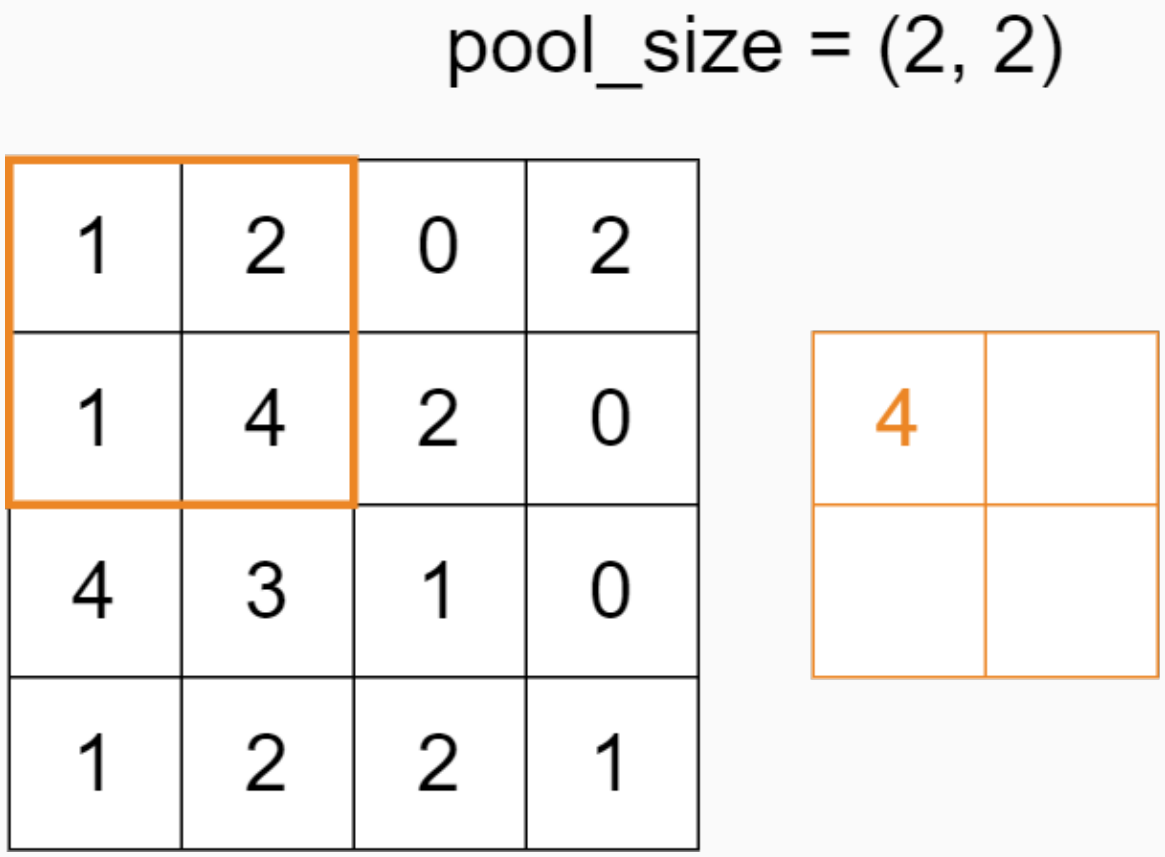
MaxPooling 2D La capa de MaxPooling2D reduce la dimensión de un vector cogiendo el máximo de la ventana definida.

pool_size = (2, 2)

1	2	0	2
1	4	2	0
4	3	1	0
1	2	2	1

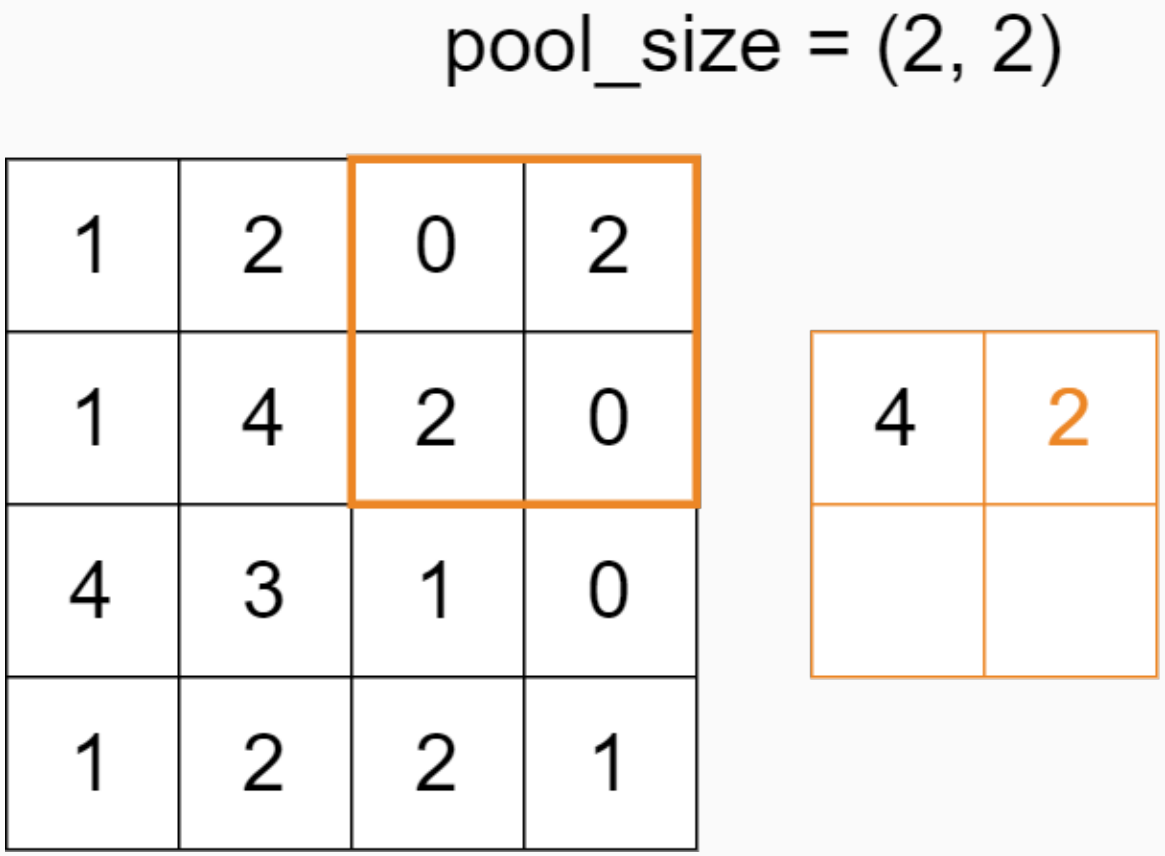
Reducción dimensional con Pooling

MaxPooling 2D La capa de MaxPooling2D reduce la dimensión de un vector cogiendo el máximo de la ventana definida.



Reducción dimensional con Pooling

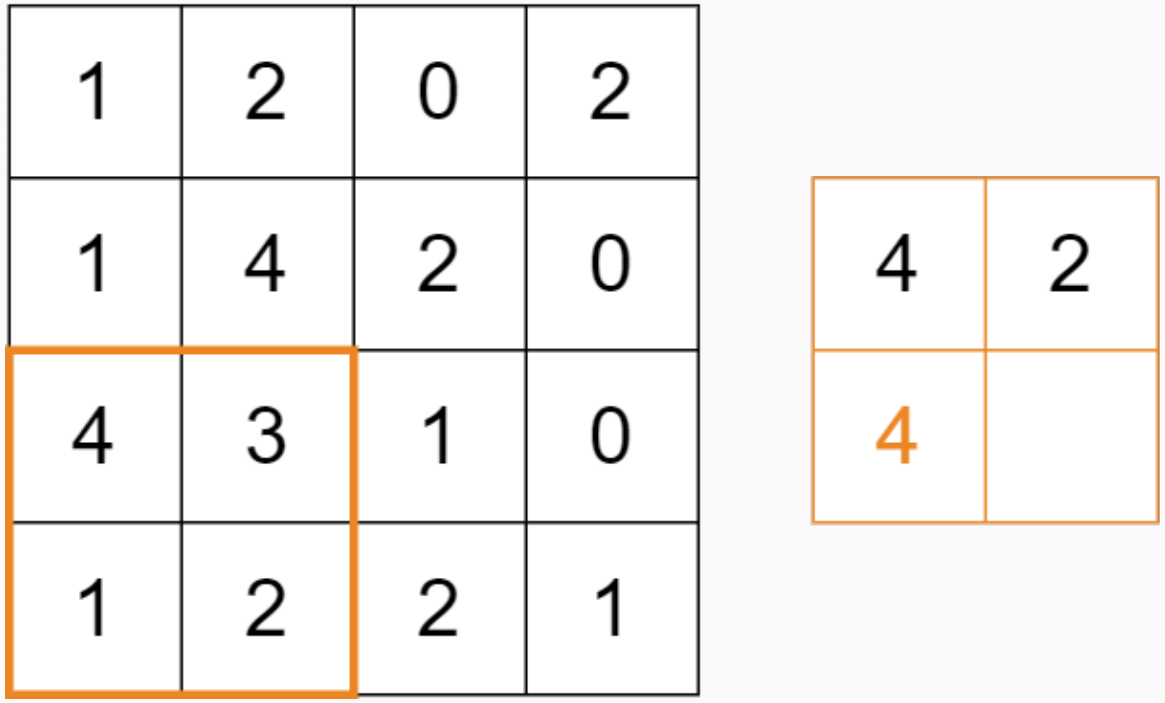
MaxPooling 2D La capa de MaxPooling2D reduce la dimensión de un vector cogiendo el máximo de la ventana definida.



Reducción dimensional con Pooling

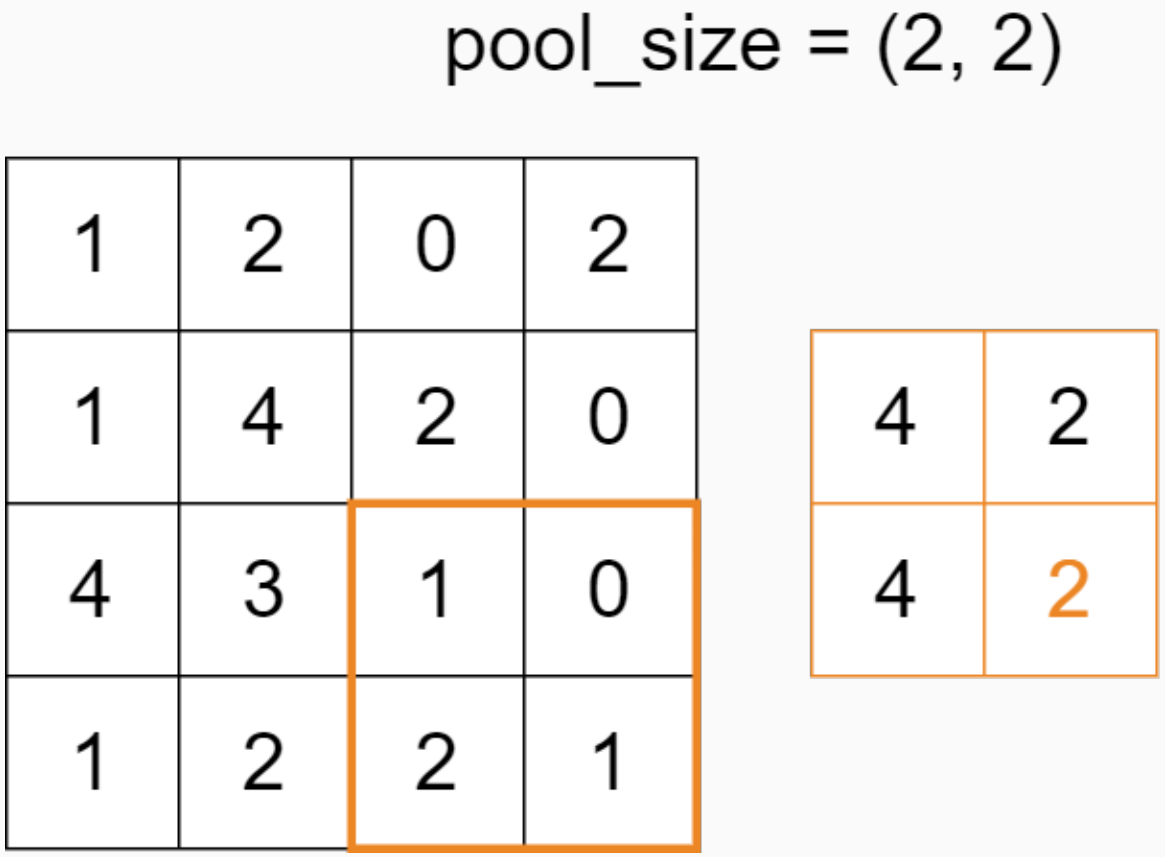
MaxPooling 2D La capa de MaxPooling2D reduce la dimensión de un vector cogiendo el máximo de la ventana definida.

pool_size = (2, 2)



Reducción dimensional con Pooling

MaxPooling 2D La capa de MaxPooling2D reduce la dimensión de un vector cogiendo el máximo de la ventana definida.



Reducción dimensional con Pooling

MaxPooling 2D La capa de MaxPooling2D reduce la dimensión de un vector cogiendo el máximo de la ventana definida.

pool_size = (2, 2)

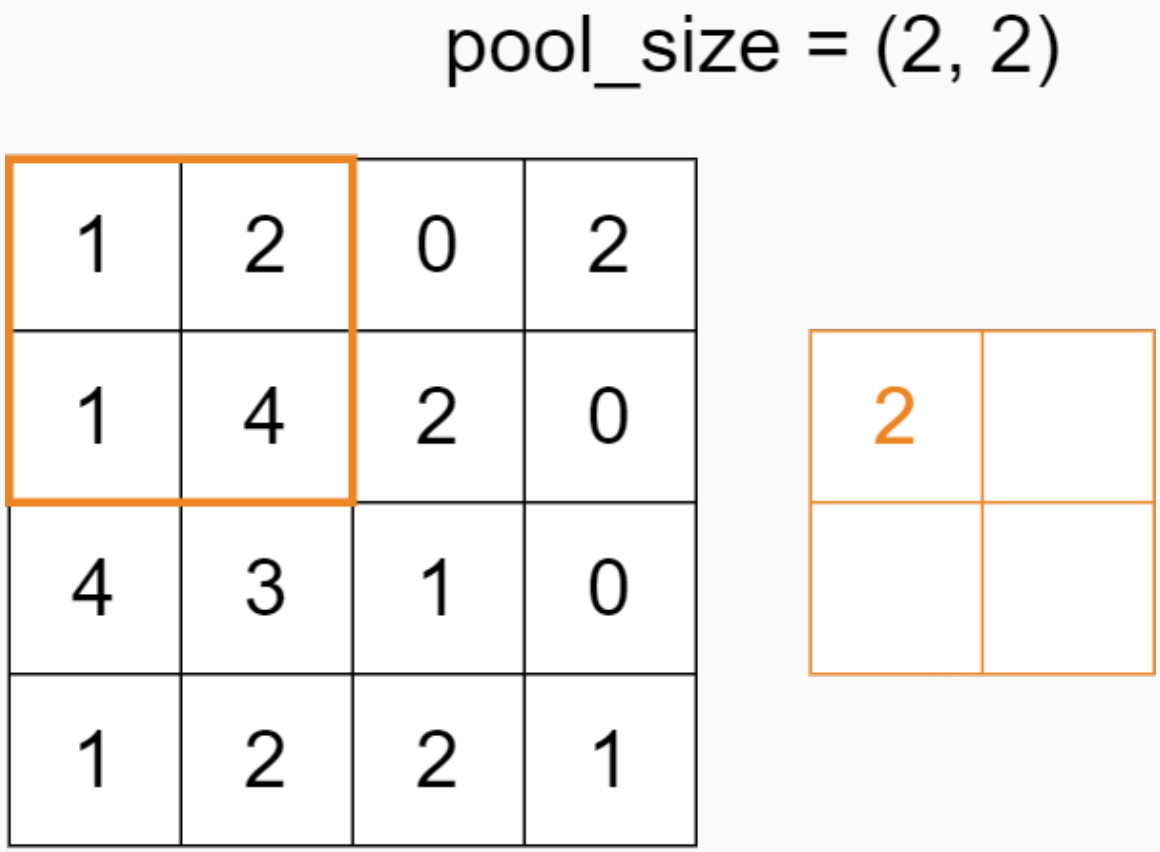
1	2	0	2
1	4	2	0
4	3	1	0
1	2	2	1

4	2
4	2

**cabe destacar que el máximo se encarga de preservar la característica más importante*

Reducción dimensional con Pooling

AveragePooling 2D La capa de AveragePooling2D reduce la dimensión de un vector cogiendo el promedio de la ventana definida.



Reducción dimensional con Pooling

AveragePooling 2D La capa de AveragePooling2D reduce la dimensión de un vector cogiendo el promedio de la ventana definida.

pool_size = (2, 2)

1	2	0	2
1	4	2	0
4	3	1	0
1	2	2	1

2	1

Reducción dimensional con Pooling

AveragePooling 2D La capa de AveragePooling2D reduce la dimensión de un vector cogiendo el promedio de la ventana definida.

pool_size = (2, 2)

1	2	0	2
1	4	2	0
4	3	1	0
1	2	2	1

2	1
2.5	1

Reducción dimensional con strides

Otra alternativa para la reducción dimensional es el uso de convoluciones con strides que no sean (1, 1). Cierta literatura apunta a esta aproximación como "más inteligente" ya que se le permite a la red que sea ella la que escoja cómo hacer la reducción.

El principal inconveniente es que usando este método se aumenta el número de parámetros de la red.

**Normalmente el stride elegido es de (2, 2) pero hay libertad para adaptarlo a distintos casos.*

Pooling en Keras

Para implementar ambos tipos de pooling en keras, se utilizan las siguientes funciones:

```
Ex: keras.layers.MaxPooling2D(pool_size=(2, 2)
Ex: keras.layers.AveragePooling2D(pool_size=(2, 2)
```

Donde el parámetro *pool_size* indica el tamaño de ventana del pooling.

Regularización

La regularización de redes neuronales es un proceso común a cualquier tipo de problema. El principal objetivo de esta es ayudar a evitar el sobreajuste y aumentar la capacidad de generalización mediante distintas restricciones. Su utilización suele considerarse opcional

En el ámbito de la visión por computador gracias a la regularización se consigue aumentar la invarianza del modelo ante cambios en las imágenes.

Las principales técnicas de regularización que encontramos son:

- Regularización L1
- Regularización L2
- Capas de dropout
- Batch normalization

Regularización L1 y L2

Estos tipos de regularización se utilizan en conjunto o por separado. Ambas consisten en añadir una penalización a la función de pérdida, y suelen especificarse en las capas convolucionales de Keras.

La regularización L1 (Lasso) agrega una penalización proporcional a la magnitud absoluta de los parámetros. Esto puede forzar algunos de los parámetros a 0.

```
Ex: layers.Conv2D(filters,k_size,
kernel_regularizer=regularizers.l2(1)
```

La regularización L2 (Ridge) agrega una penalización proporcional al cuadrado de la magnitud de los parámetros. Esto no forzará a los parámetros a ser exactamente 0, pero tenderá a reducir su magnitud.

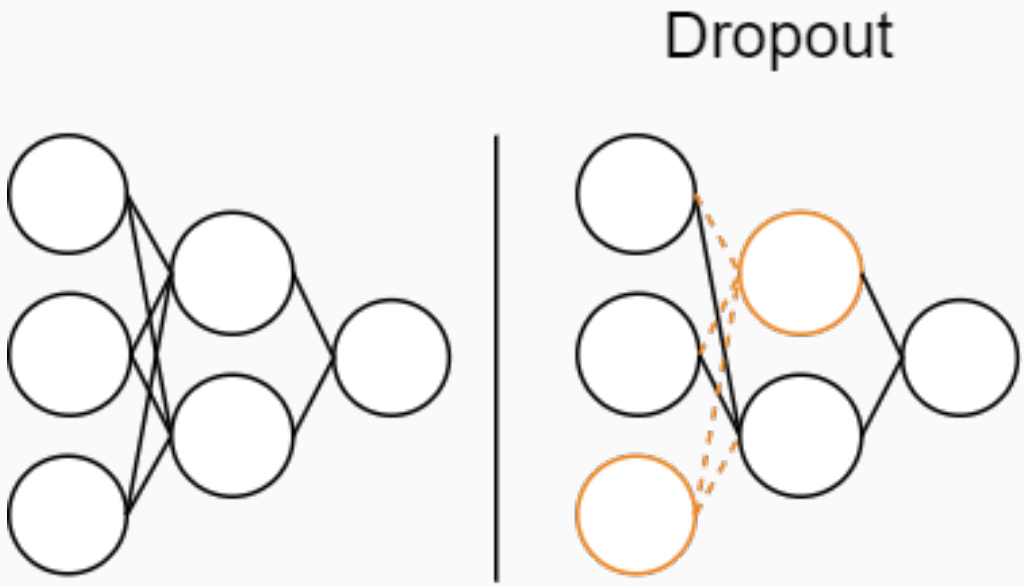
```
Ex: layers.Conv2D(filters,k_size,
kernel_regularizer=regularizers.l1(1)
```

Capa de dropout

La capa de dropout es una capa de regularización que desactiva aleatoriamente la activación de ciertas neuronas durante el entreno.

El dropout se utiliza normalmente antes de una capa con neuronas, y en Keras tiene la siguiente forma, donde el parámetro indica el porcentaje de neuronas a desactivar:

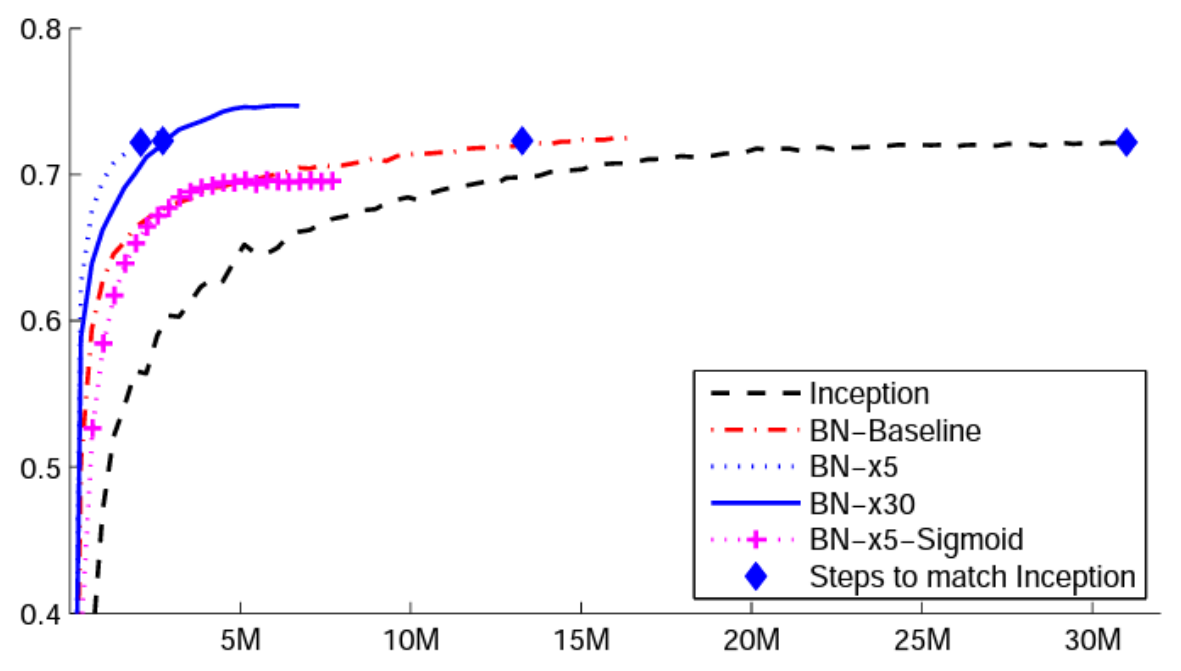
```
Ex: layers.Dropout(0.5)
```



Batch normalization

Las capa de normalización del batch tienen como objetivo mantener la media de las activaciones de una capa en torno al 0 con desviación estándar de 1. Generalmente se utiliza tras la convolución.

El uso de esta capa ayuda enormemente a la estabilidad y rendimiento de las redes neuronales. Se utiliza prácticamente en todas las investigaciones actuales.



[1]

Batch normalization

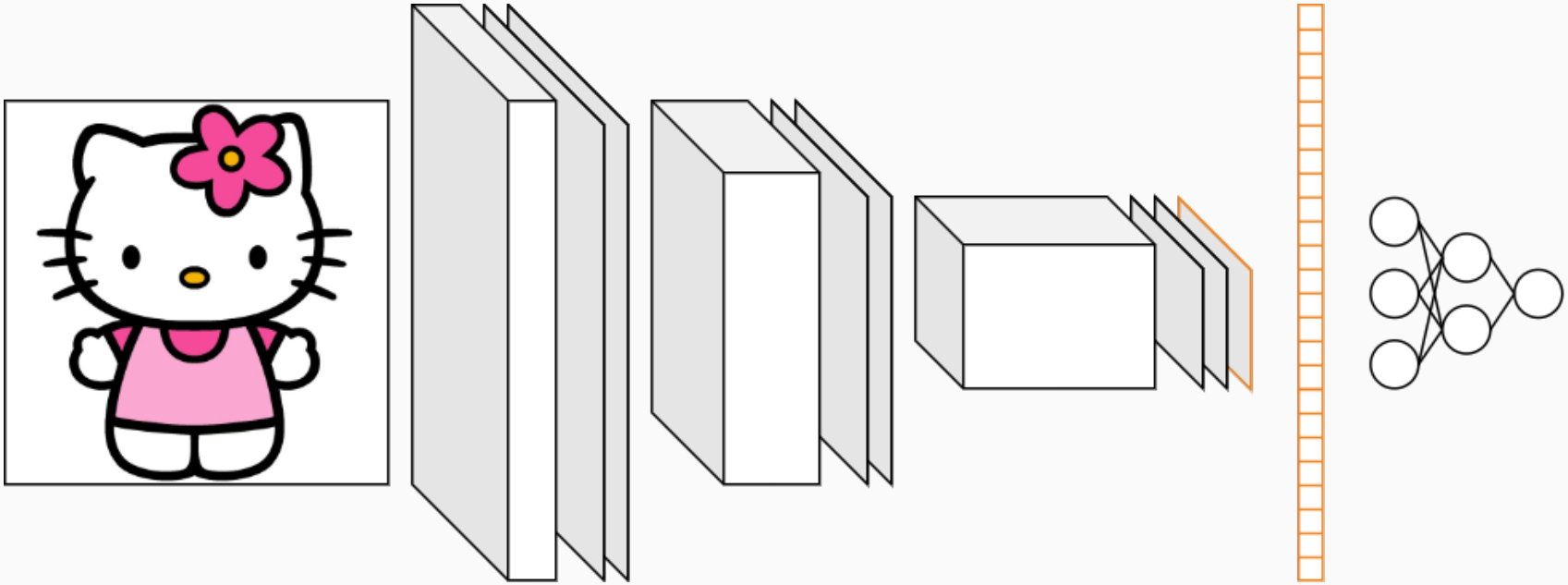
Suele localizarse antes o después de la no-linealidad.

Ex: `layers.BatchNormalization()`

Capa Flatten

Esquema de construcción de redes convolucionales

- Input
- Conv2D
- Activation
- Pooling y dropout
- Flatten
- Dense
- Output



Flatten

Una vez las características principales han sido extraídas en sucesivas capas convolucionales, no-linealidades, poolings y regularizaciones, es necesario realizar el aprendizaje. Para ello se utilizan redes neuronales básicas, o completamente conectadas.

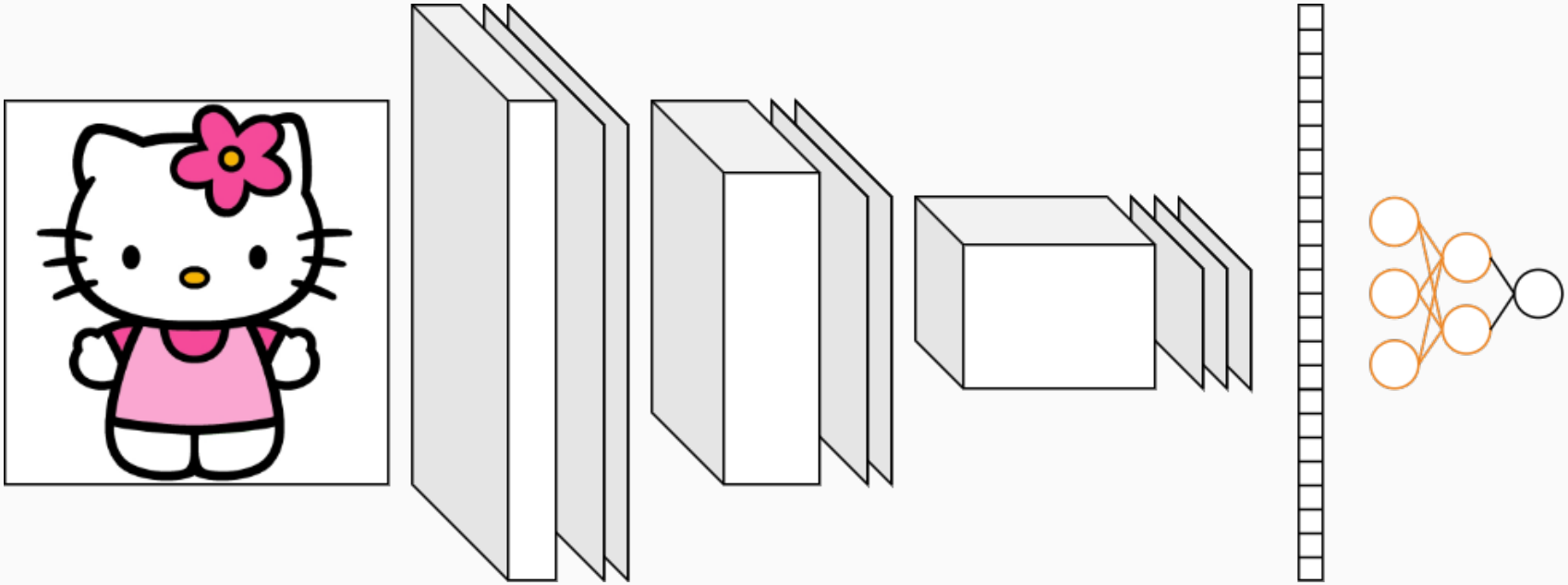
En este punto los datos aún se encuentran en formato bidimensional (mapas de características). Como se vio anteriormente, para alimentar una red básica con imágenes es necesario transformarlas a unidimensionales. Para ello se hace uso de una capa flatten en Keras:

```
Ex: layers.Flatten()
```

Capa completamente conectada

Esquema de construcción de redes convolucionales

- Input
- Conv2D
- Activation
- Pooling y dropout
- Flatten
- Dense
- Output



Capas completamente conectadas

La capas completamente conectadas de una CNN es la encargada de procesar la información de las características extraídas y generar una salida.

Como ya se vio durante los fundamentos de redes neuronales, las capas completamente conectadas están compuestas por neuronas donde cada una de ellas está conectada a todas las de la capa anterior.

En keras esta capa se implementa de la siguiente forma para un problema de clasificación:

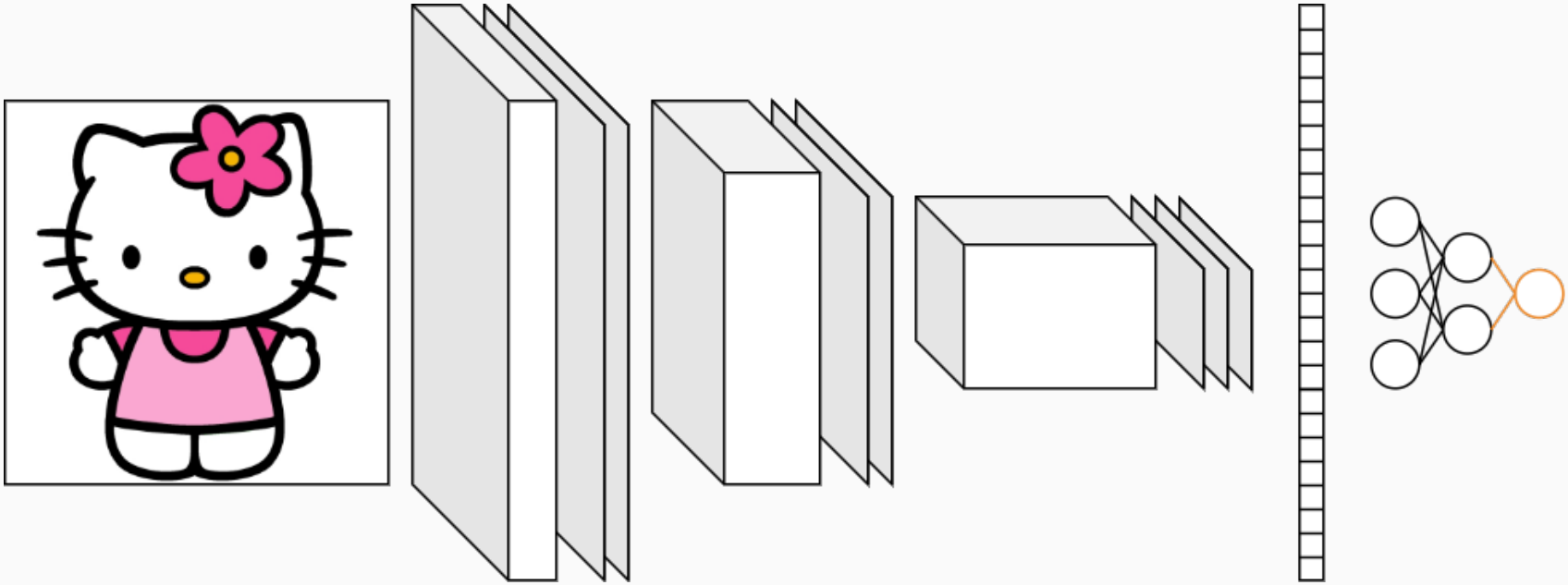
```
Ex: layers.Dense(num_classes,activation='softmax')
```

Donde en el primer parámetro se indica el número de posibles clases y en el segundo la función de activación de salida.

Capa de salida

Esquema de construcción de redes convolucionales

- Input
- Conv2D
- Activation
- Pooling
- Flatten
- Dense
- Output



Salida

De forma análoga a la activación de la capa de convolución, la activación de salida puede indicarse como parámetro de la capa completamente conectada o como una capa de activación.

Esta función de activación se elegirá dependiendo de la tarea que queramos llevar a cabo:

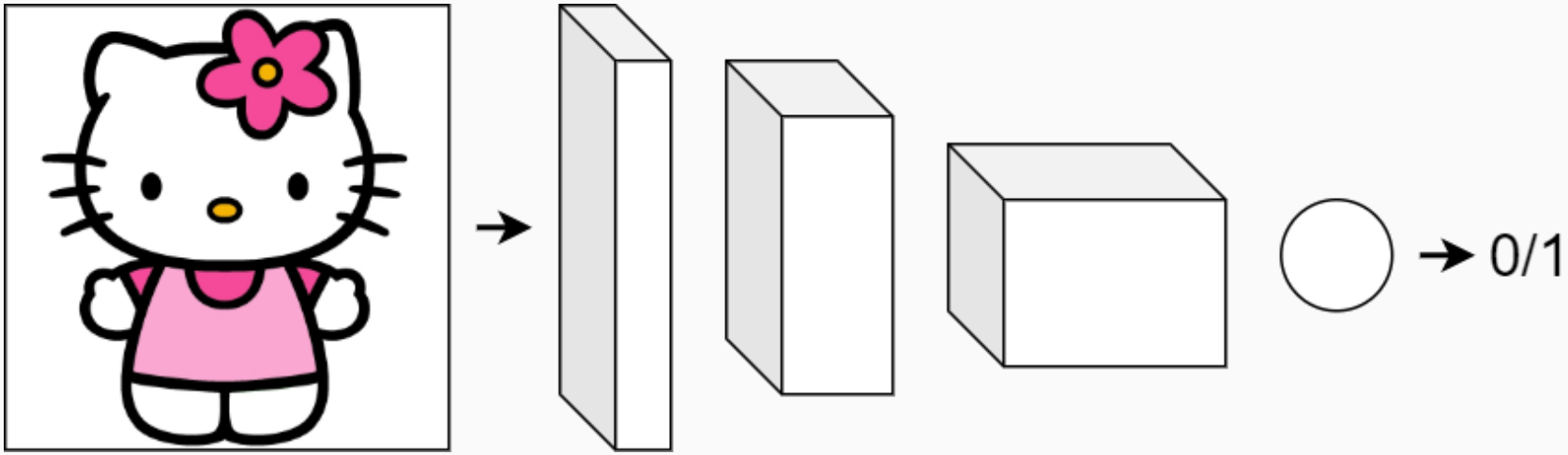
- Regresión
- Clasificación
  - Binaria
  - Multiclase

En todos los casos de clasificación, el resultado de esta capa es un vector de probabilidades de pertenencia a cada categoría.

Entrenamiento de una CNN

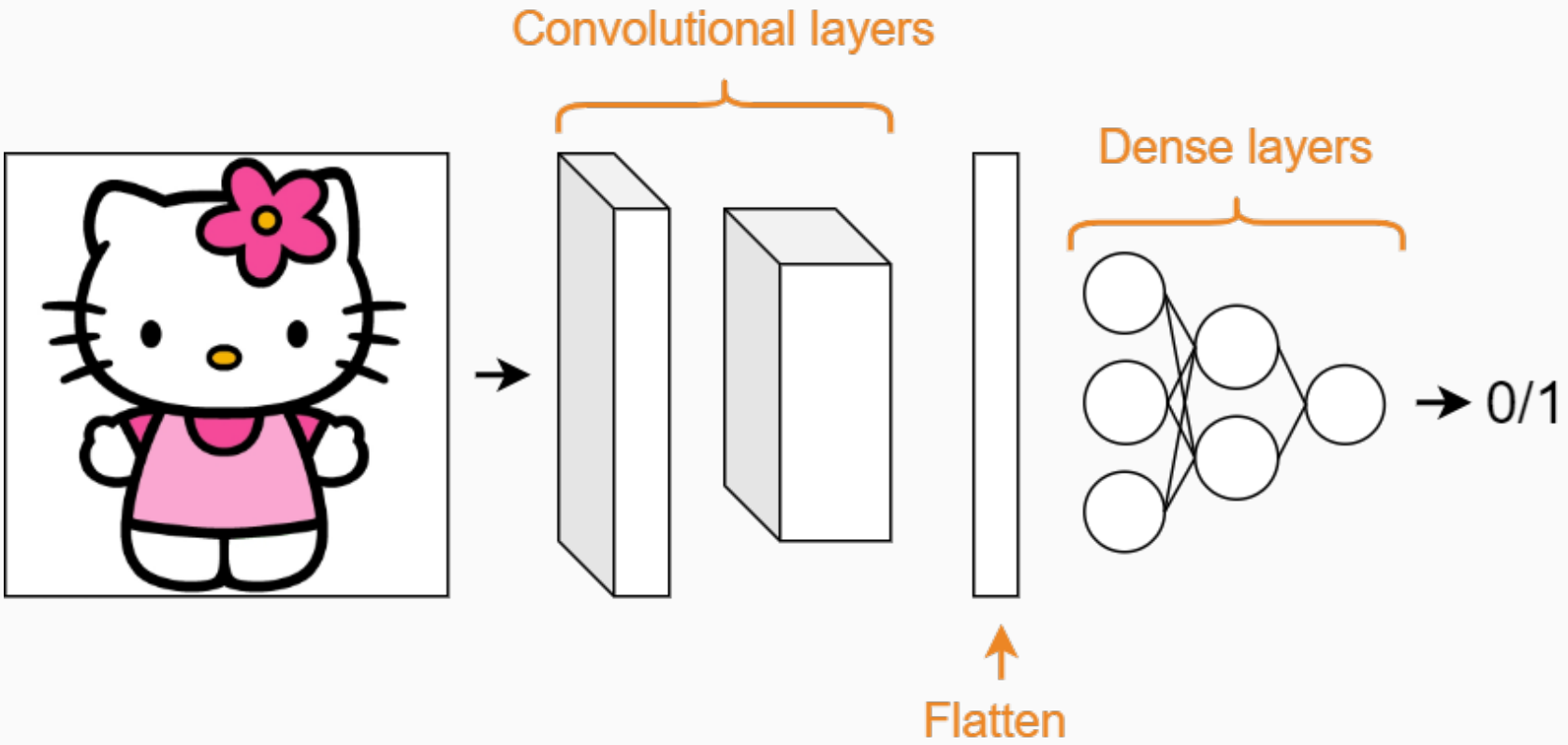
Construcción de una CNN

La estructura de embudo típica de las redes neuronales clasificadoras también se aplica a Convolutional Neural Network (CNN). Para ello el objetivo es reducir la dimensión de la imagen hasta generar una salida.



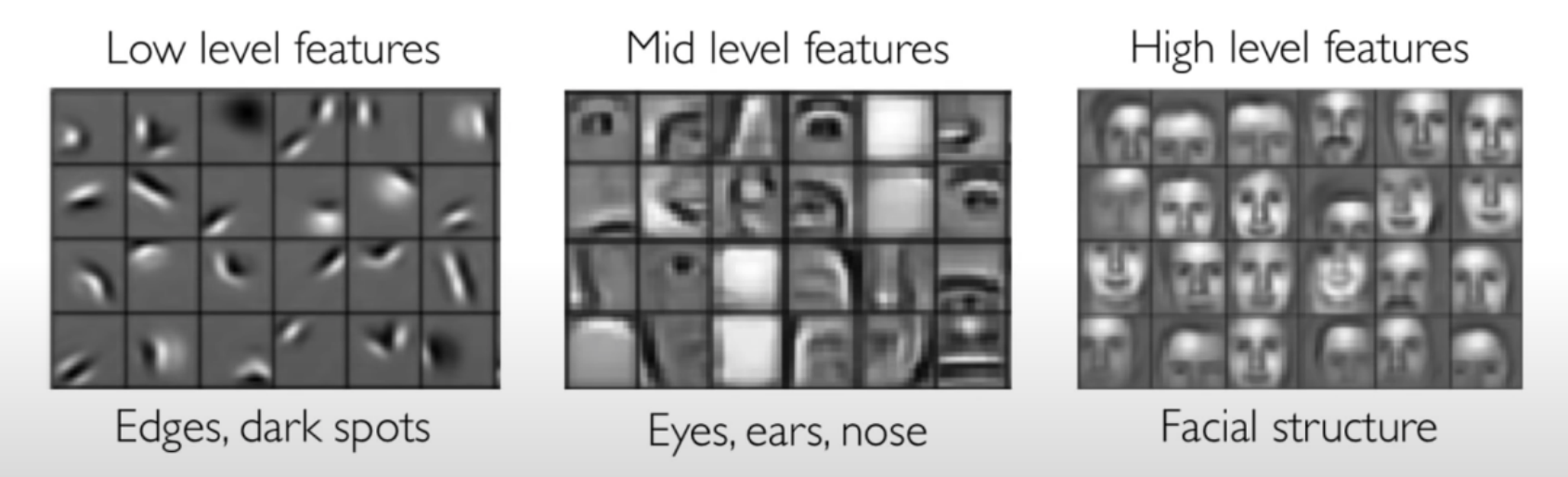
Construcción de una CNN

Las primeras capas convolucionales de la red se encargan de la extracción de características de la imagen. Posteriormente un perceptrón se encarga de clasificar las características extraídas para generar la salida deseada.



Construcción de una CNN

Es importante recordar que la jerarquía de capas de una red convolucional detecta características a alto nivel en las capas más profundas.



[2]

Los hiperparámetros en una red

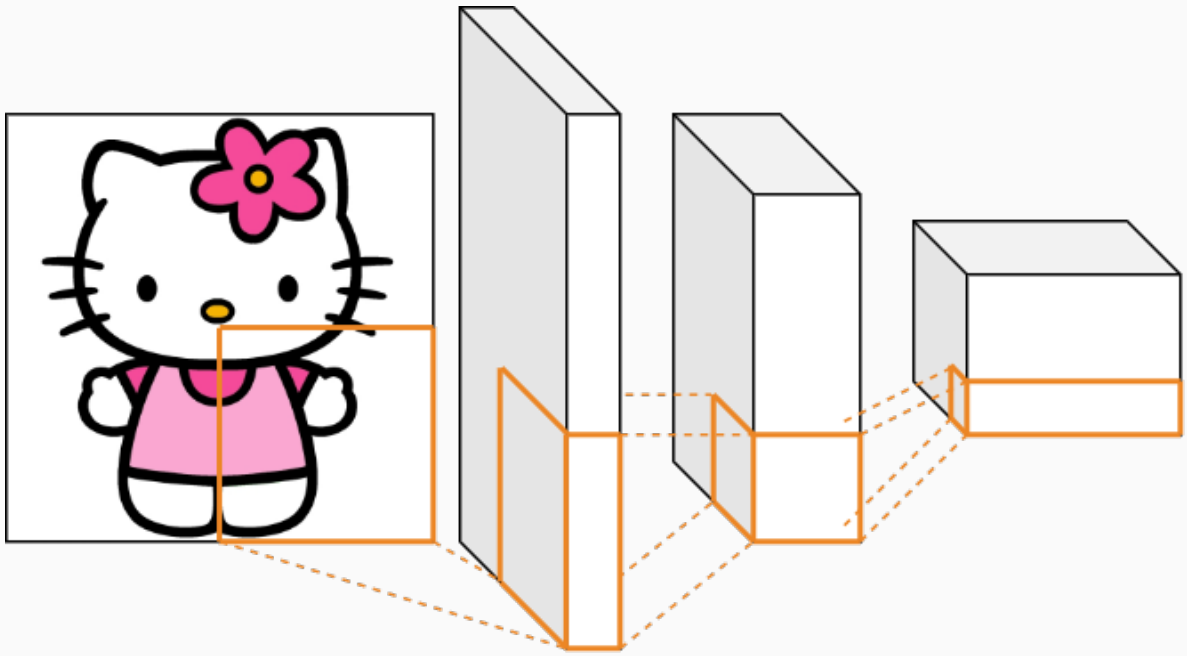
Uno de los mayores inconvenientes a la hora de realizar entrenamientos con redes neuronales artificiales es su difícil configuración. Debido a la cantidad inmensa de hiperparámetros a escoger.

Sin embargo, existen una serie de prácticas comunes a la hora de tratar con CNNs.

Tamaño de imagen y filtros

A la hora de escoger el número de filtros de cada capa convolucional este va ligado al tamaño de la matriz de datos.

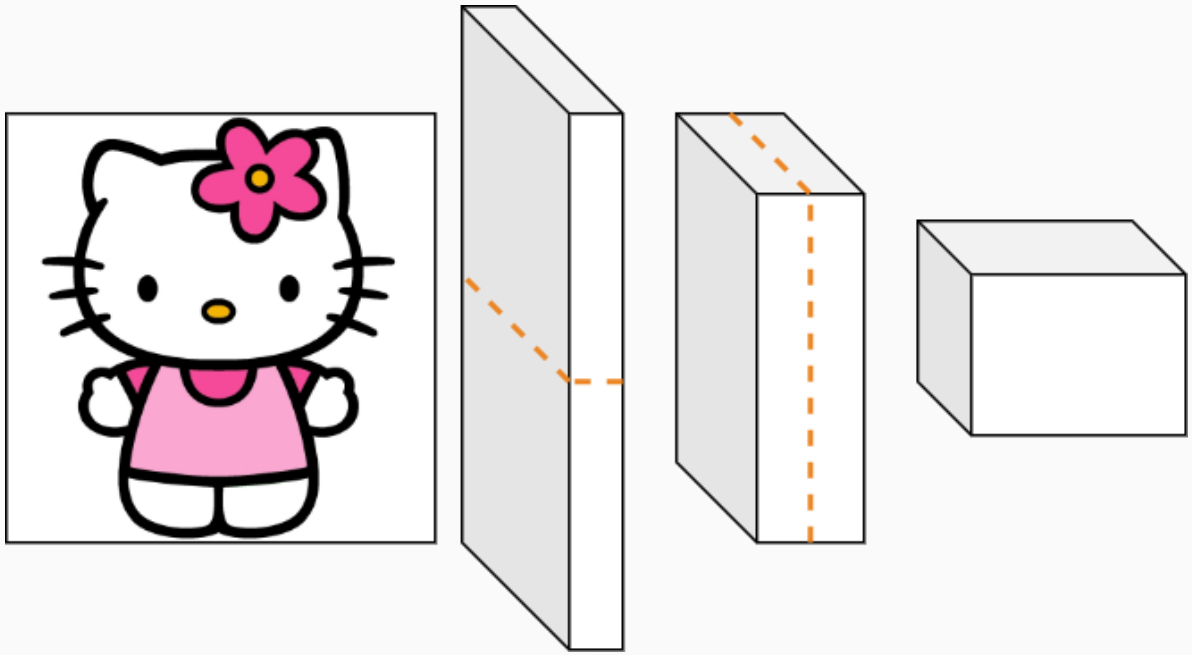
A medida que la imagen de entrada va reduciendo su tamaño, el número de filtros aumenta. Con esto se pretende extraer más características de alto nivel cada vez cubriendo zonas más amplias de la imagen original.



Tamaño de imagen y filtros

Al mismo tiempo, a medida que el número de filtros de multiplica por 2 las dimensiones de la matriz de datos se reducen a la mitad.

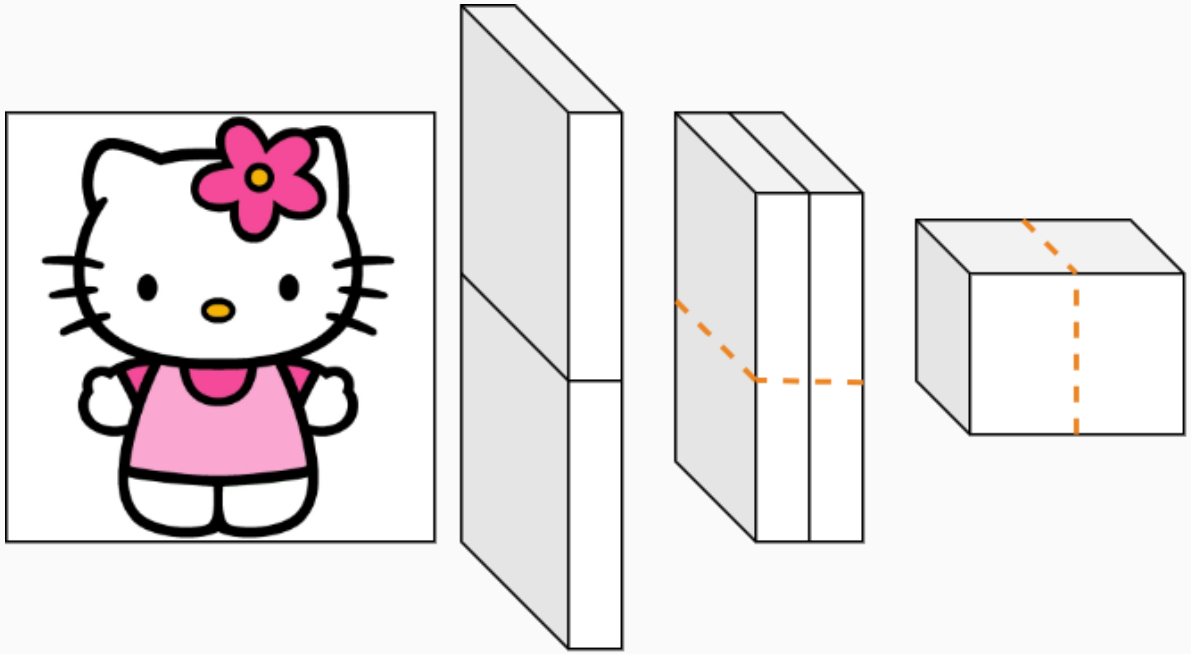
El objetivo de este intercambio es mantener la misma cantidad de información, pero tratada por la red.



Tamaño de imagen y filtros

Al mismo tiempo, a medida que el número de filtros de multiplica por 2 las dimensiones de la matriz de datos se reducen a la mitad.

El objetivo de este intercambio es mantener la misma cantidad de información, pero tratada por la red.



Otros hiperparámetros

kernel_size

El tamaño del kernel habitualmente es de (3, 3) o (5, 5), en caso de imágenes muy grandes puede llegar a (7, 7).

Para matrices de datos más grandes se utilizan kernels más grandes, en casos combinando kernels de (5, 5) para las primeras capas y posteriormente (3, 3) para capas más profundas.

strides

El paso de la convolución se mantiene a (1, 1) a no ser que se desee una reducción dimensional.

Otros hiperparámetros

padding

El padding de una convolución suele ser same para controlar las dimensiones de la matriz de datos, pero no es extraño encontrar casos con padding valid.

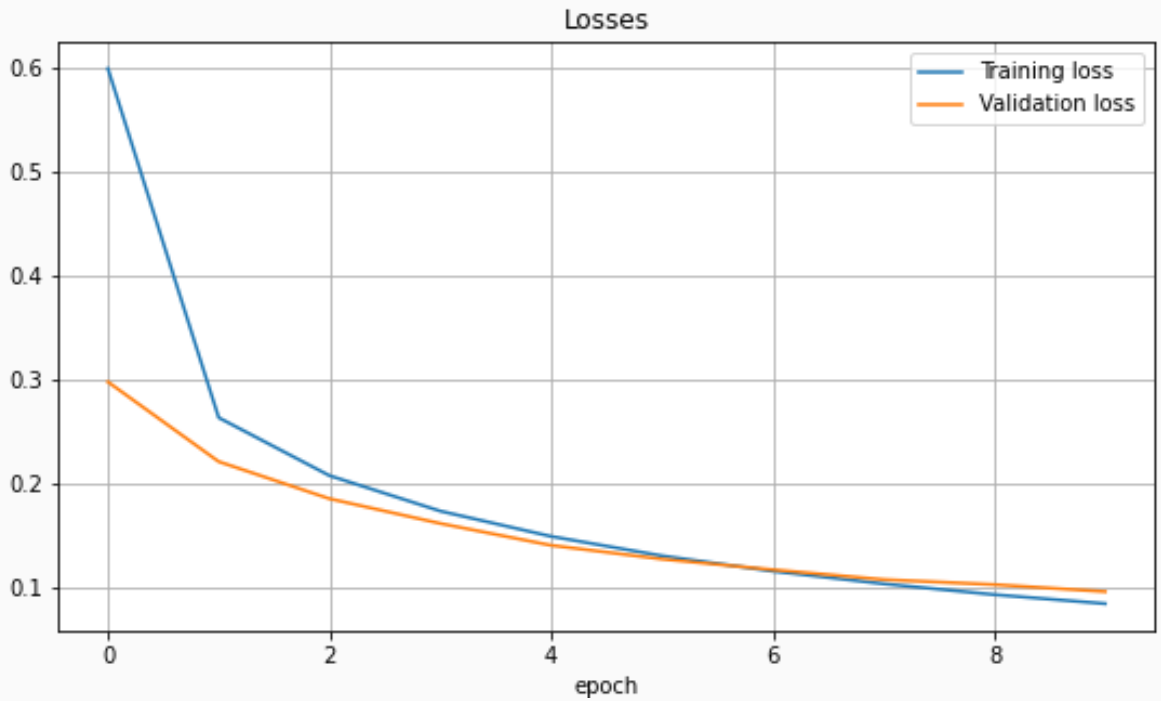
activation

Para las capas ocultas se suele utilizar la función ReLU o LeakyReLU, para la capa de salida la activación depende del problema concreto.

Valor de pérdidas

De la misma manera que para cualquier red neuronal una de las maneras más sencillas y directas de evaluar un entrenamiento es gracias a sus pérdidas.

Dependiendo del problema en concreto el valor de las pérdidas puede variar.



Notebook de ejemplo, resultado de capas CNN

El siguiente notebook contiene un breve código para explorar el resultado de las capas CNN.

- 03.03-CNNLayers.ipynb

Referencias i

[1] Sergey Ioffe and Christian Szegedy.
Batch normalization: Accelerating deep network training by reducing internal covariate shift.
In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[2] Kathrin Melcher (Medium).
Convolutional hierarchy image.
[Online; accessed August, 2022].