



INTRODUCCIÓN AL FRAMEWORK KERAS Y TENSORFLOW PARA EL DESARROLLO DE REDES NEURONALES DE DEEP LEARNING

LOURDES DURÁN LÓPEZ (lduran2@us.es)

JUAN PEDRO DOMÍNGUEZ MORALES (jpdominguez@us.es)

UNIVERSIDAD DE SEVILLA
ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES



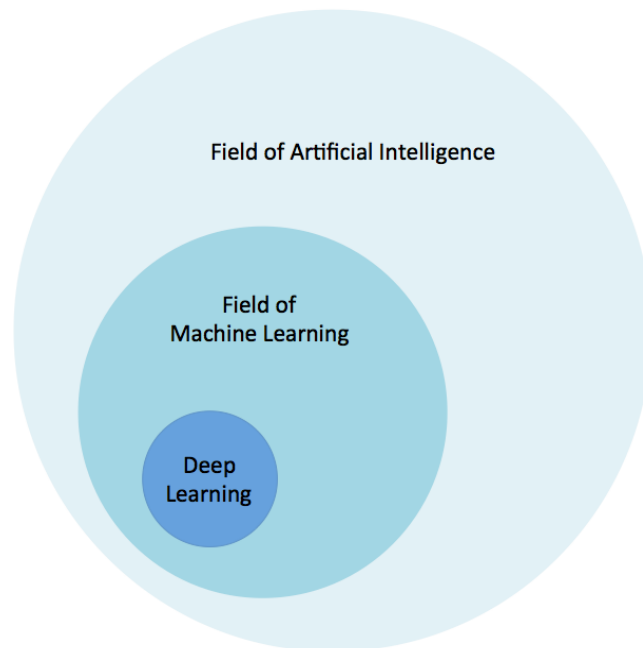
ÍNDICE

- Deep Learning y CNNs
- Tensorflow y Keras
- Instalación
- Ejemplos

DEEP LEARNING

- Redes neuronales de aprendizaje profundo (Deep Learning Neural Networks, DLNN):

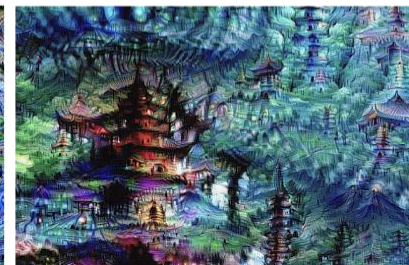
“Arquitecturas de redes neuronales artificiales (ANN) capaces de analizar datos, identificar patrones y/o modelar sistemas no lineales”



DEEP LEARNING

- Tradicionalmente se han desarrollado clasificadores basados en características/reglas definidas por un programador.
- Las DLNN aprenden a partir de un dataset:
 - Capacidad de abstracción y generalización.
- Están mostrando muy buenos resultados. Algunas aplicaciones son:
 - Visión por computador.
 - Reconocimiento automático del habla.
 - Procesamiento natural de lenguaje.
 - Filtrado y recomendación en redes sociales.
 - Conducción autónoma de vehículos.
 - Operaciones financieras.





Ethereal



HDR















Melancholy

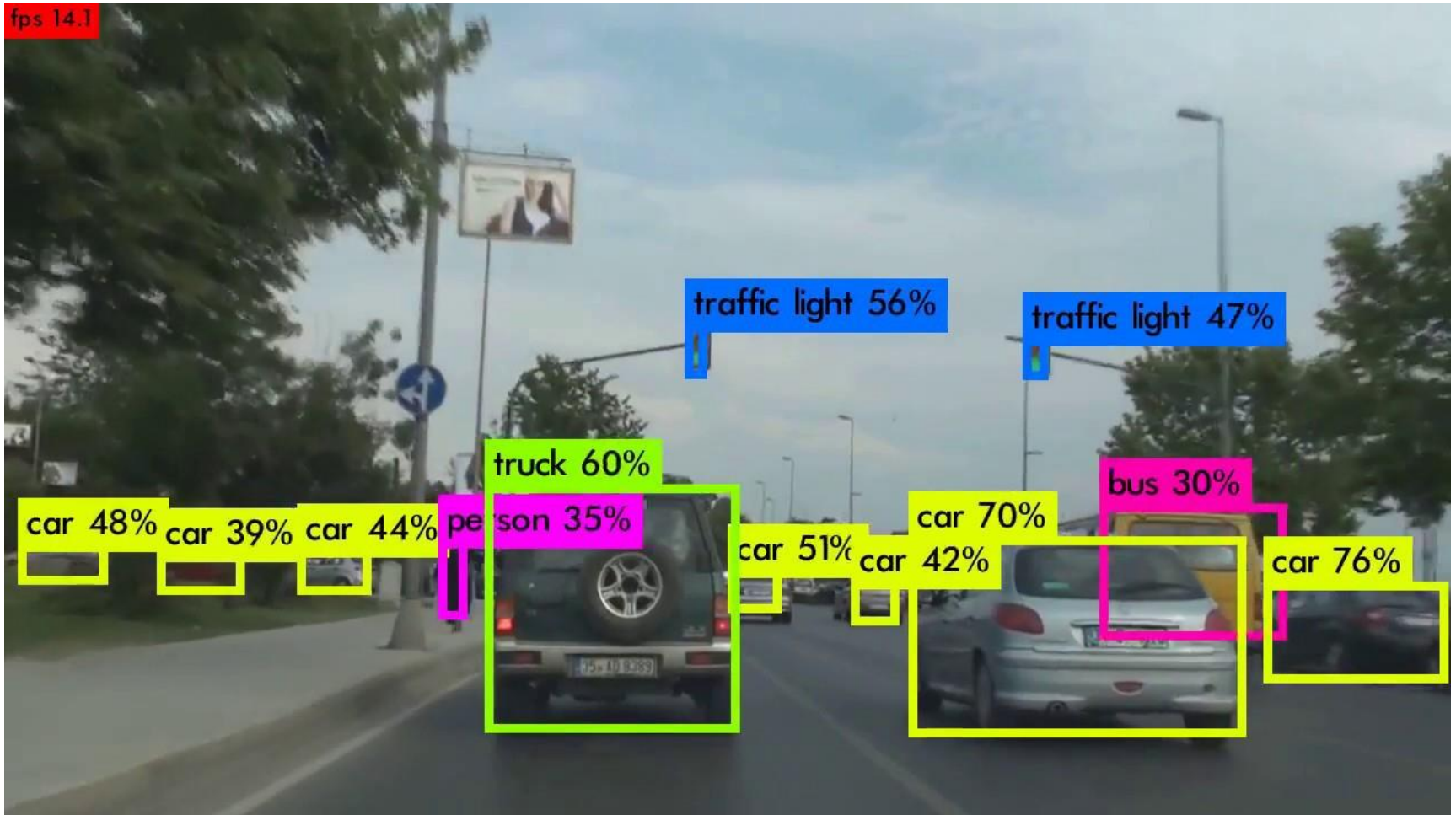


Minimal



Ethereal	HDR	Melancholy	Minimal
			
			
			

fps 14.1

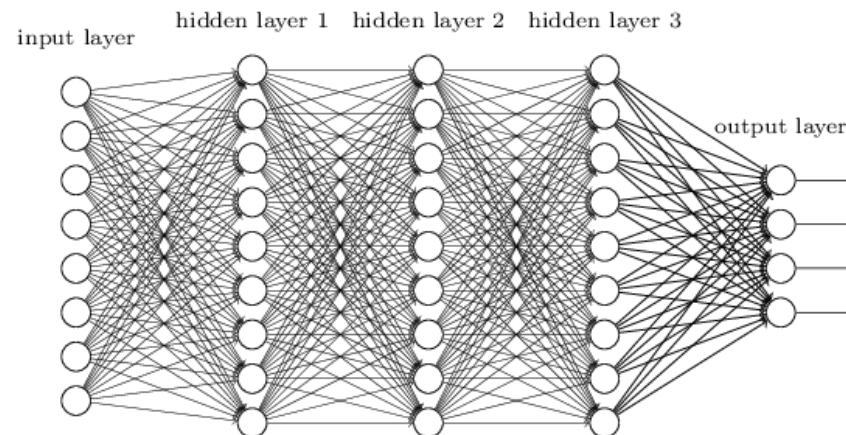
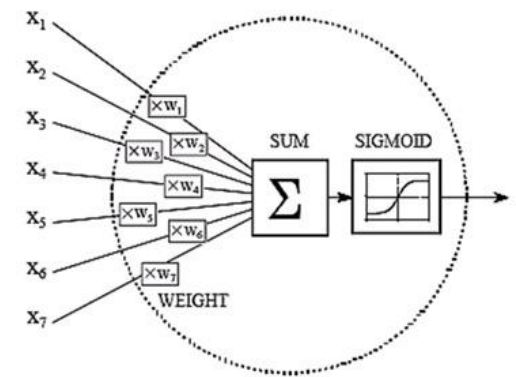
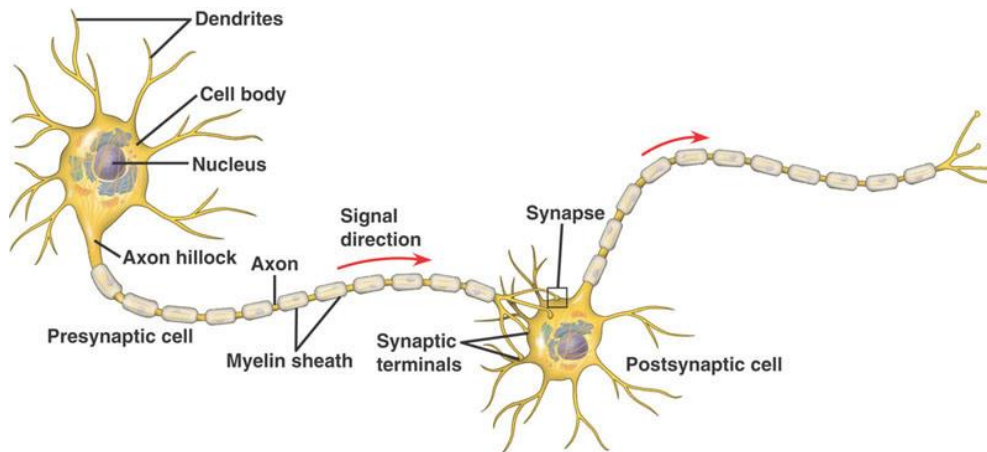


¿ESTO ES NUEVO?

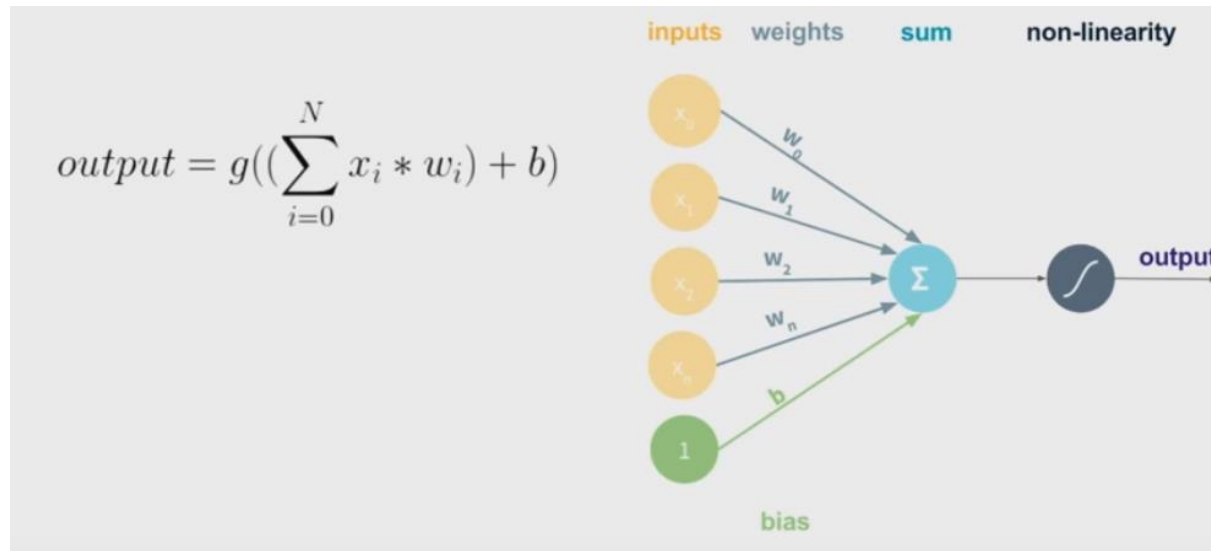
- Las redes neuronales se empezaron a usar en los años 80.
- ¿Por qué ahora este tipo de redes comienzan a ser tan populares?
 - Ampliación de la capacidad de computación: CPU + GPU, Cloud Computing...
 - Disponibilidad de datos para entrenamiento: grandes datasets de imágenes, sonidos, datos financieros...
 - Gran diversidad de frameworks libres: Caffe, Keras, TensorFlow, Theano, PyTorch...
 - Apoyo de grandes empresas: Google, Facebook, Flickr, Amazon...



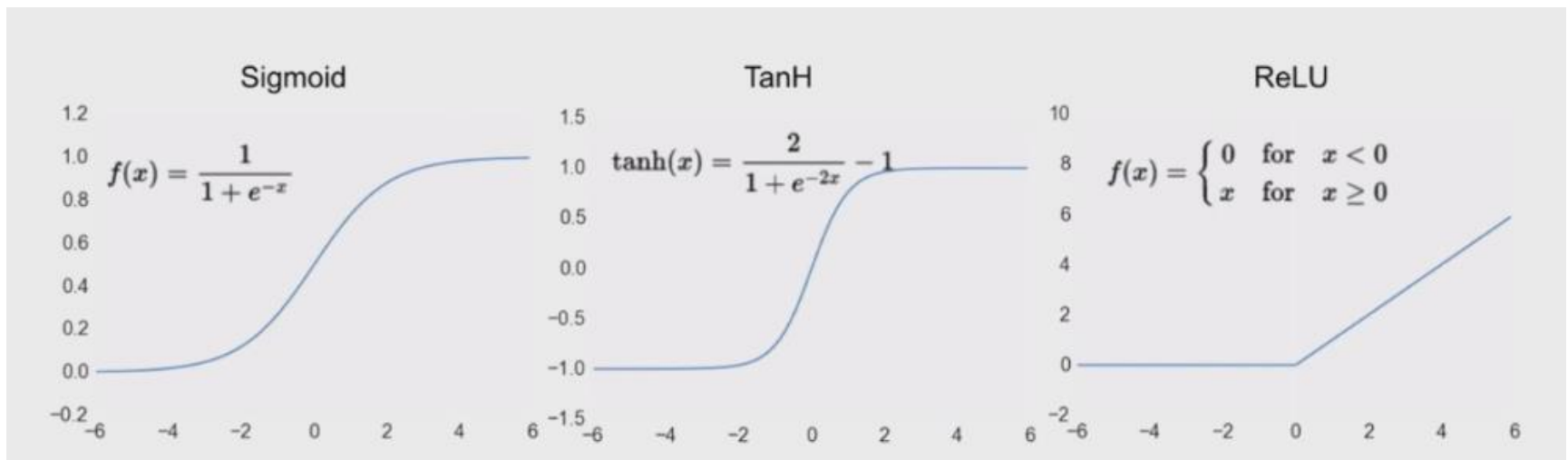
REDES NEURONALES



- Neurona Basada en el Perceptrón:



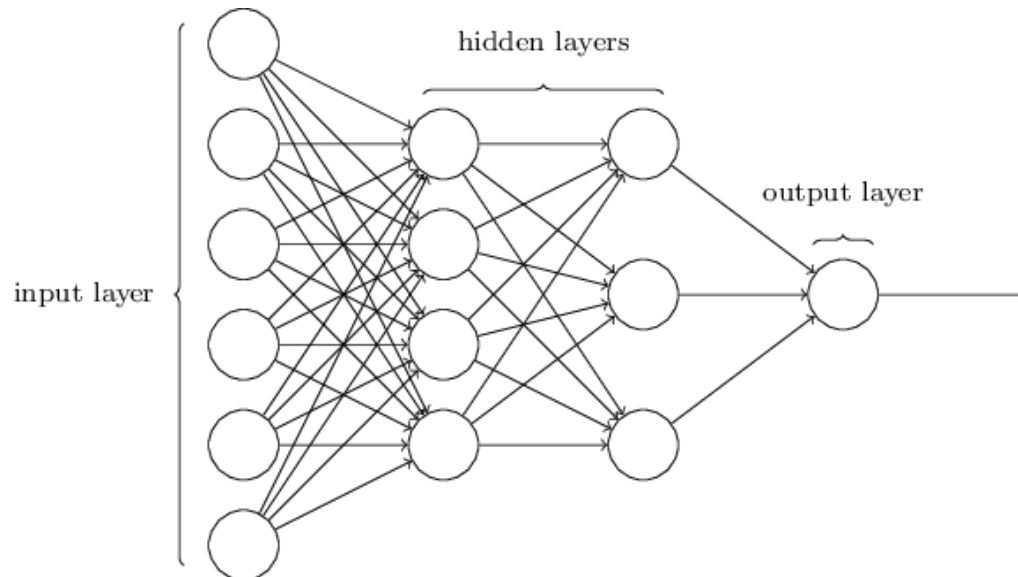
- Funciones de activación No Lineales:



- Las funciones de activación añaden un componente No Lineal al Perceptrón, y en conjunto a la red completa.
- La realidad no suele ser lineal:



- En general las NN están compuestas por una serie de capas de neuronas conectadas en cascada.
- La primera y última capa se denominan capas de entrada y de salida respectivamente, y las capas intermedias se conocen como capas ocultas.
- Existen diversas arquitecturas de redes:
 - Convolutional Deep Neural Networks (CNN).
 - Regional Convolutional Neural Network (RCNN).
 - Deep Belief Neural Networks (DBNN).
 - Recurrent Neural Networks (RNN).
 - Long-Short Term Memory (LSTM).





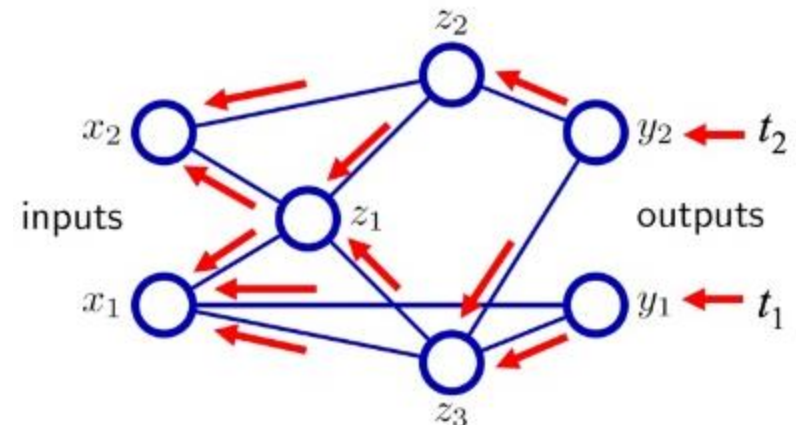
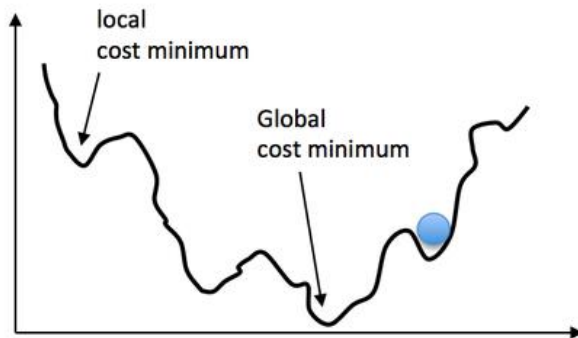
```
graph LR; A[Entrenamiento] --> B[Validación]; B --> C[Test];
```

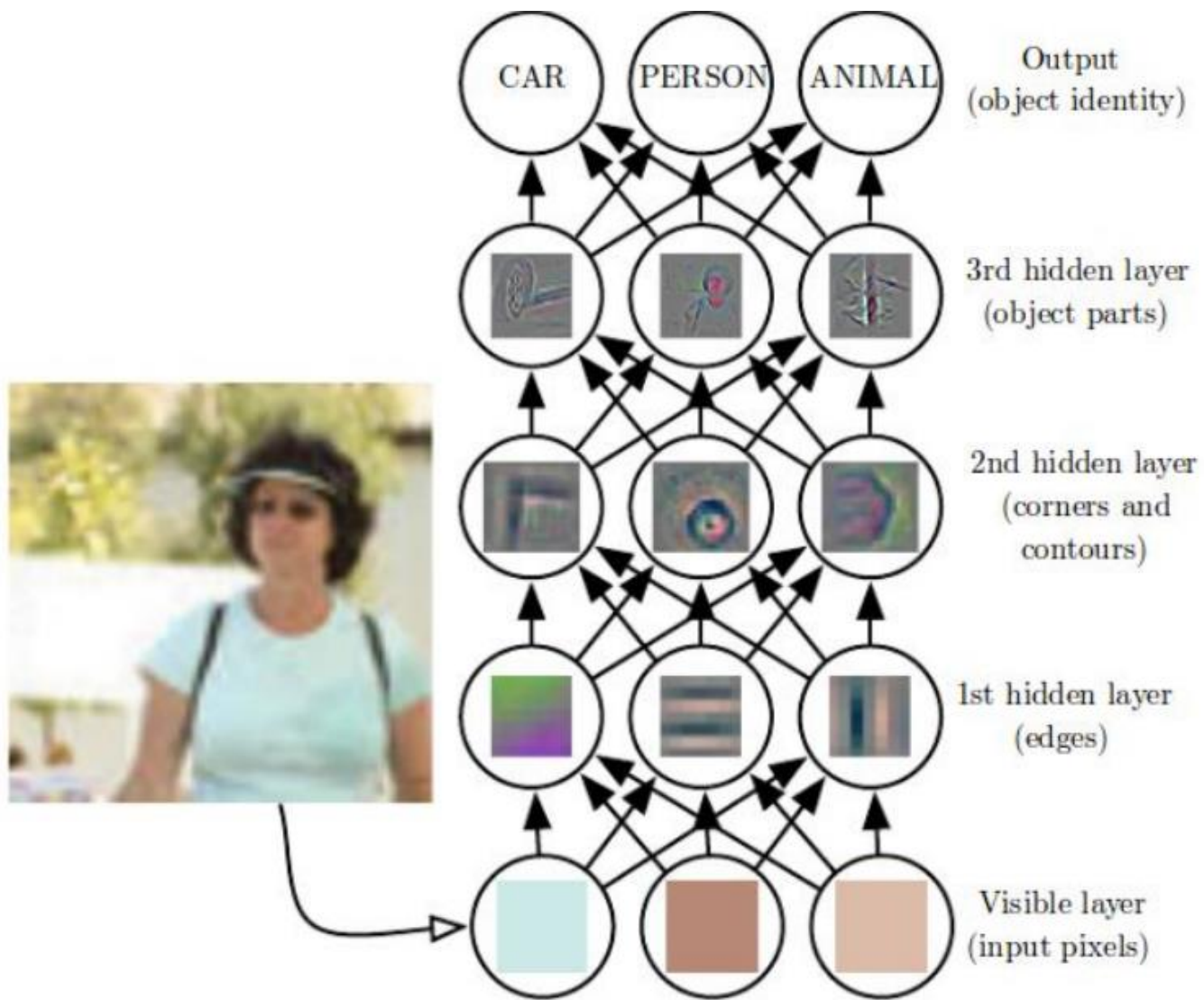
Entrenamiento

Validación

Test

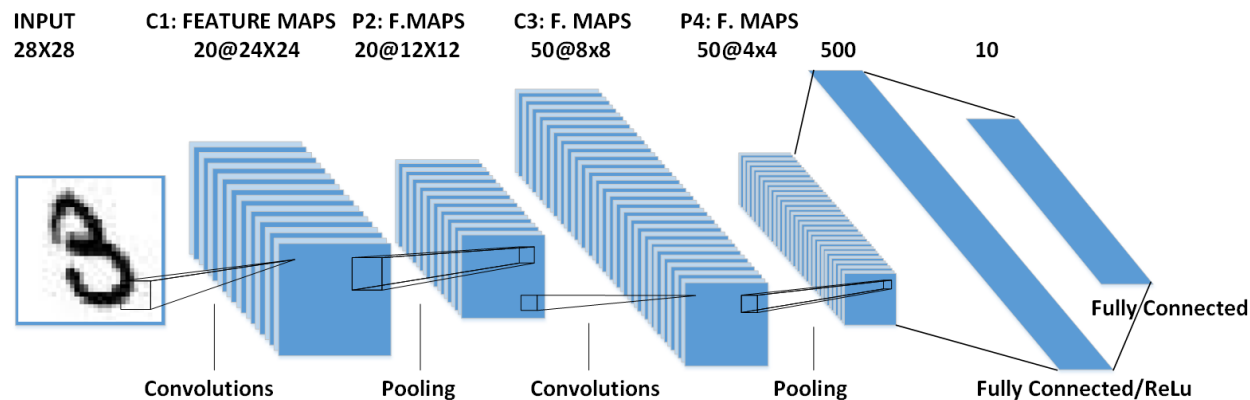
- Un algoritmo muy común para el entrenamiento de NN es el de propagación hacia atrás (backpropagation):
 - Se calcula el error total en la capa de salida.
 - Se van ajustando los pesos de las capas anteriores en base a su contribución al error. Este ajuste responde a diversas estrategias.
- Se intenta evitar el problema de los mínimos locales.
- Existen dos tipos de aprendizaje:
 - Supervisado: datos etiquetados.
 - No supervisado: datos sin etiquetar.





REDES NEURONALES DE CONVOLUCIÓN (CNN)

- Las CNN se estructuran en capas que realizan diferentes operaciones.
- Las capas más habituales en una CNN son:
 - Convolución
 - Pooling
 - Dropout
 - Activación
 - Fully connected



CAPA DE CONVOLUCIÓN

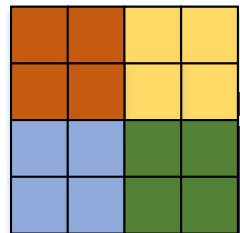
- Requiere uno o varios filtros con los que convolucionar los datos de entrada a la capa, generando una serie de datos de salida.
- Los datos de entrada y de salida se conocen como feature maps.
- Para definir una capa de convolución basta con especificar el número y el tamaño de los filtros.

1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

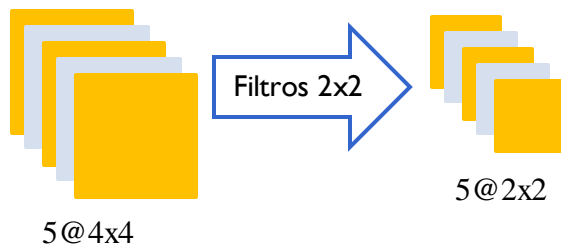
4		

Convolved
Feature



CAPA DE POOLING

- Esta capa se utiliza para acelerar la computación de las CNN, quedándose con la información más relevante de los feature maps.
- Agrupa los datos de entrada localmente reduciendo el tamaño y resolución de los feature maps.
- Esta capa no se ve afectada en la fase de entrenamiento, ya que siempre aplica la misma función que no posee ningún parámetro variable que ajustar.



Max pooling

12	20	30	0		
8	12	2	0	2 x 2 Max-Pool	20 30
35	70	37	6		99 37
99	80	25	12		

Avg pooling

12	20	30	0		
8	12	2	0	2 x 2 Avg-Pool	13 8
35	70	37	6		71 20
99	80	25	12		



Capa de Dropout

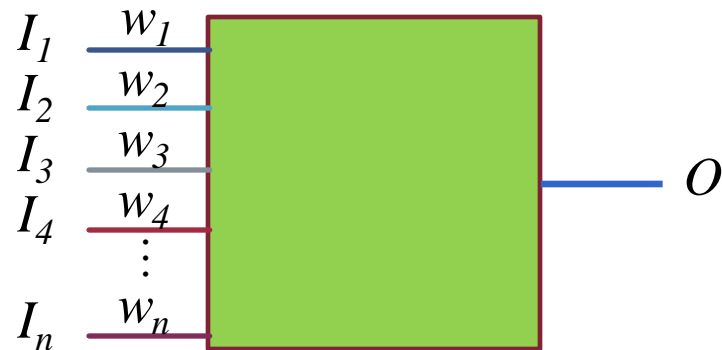
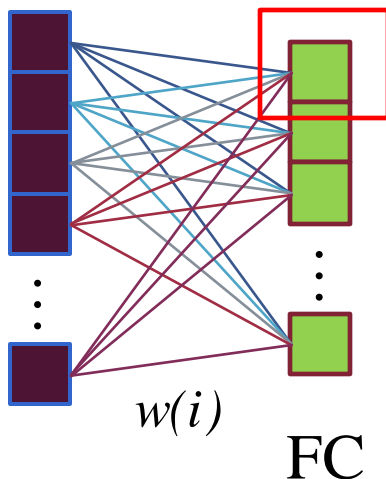
- Esta capa suprime el número de neuronas en función de una probabilidad con el fin de acelerar la computación en las sucesivas capas y reducir el sobreajuste (overfitting).
- Solo se aplica en la fase de entrenamiento.

Capa de Activación

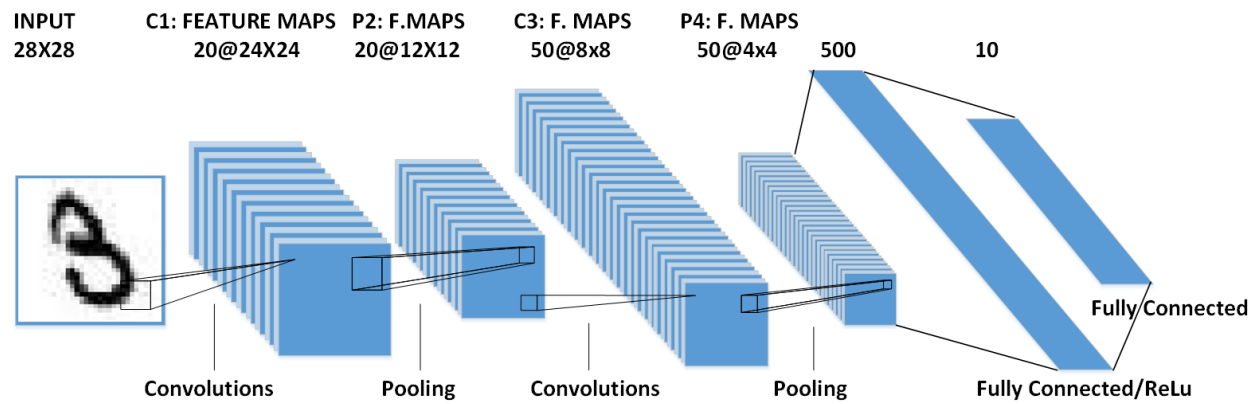
- Se aplican funciones no lineales a los elementos de la entrada.

FULLY CONNECTED / DENSE

- Este tipo de capa es conocida como la etapa de decisión, y se utiliza para hacer la clasificación.
- La función de esta capa es la de relacionar todos los datos de entrada entre sí, haciendo uso de unos pesos que son calculados y ajustados en la fase de entrenamiento.
- Para definir esta capa basta solo con indicar el número de unidades que se desea tener en la capa.



$$O = I_1 * w_1 + I_2 * w_2 + I_3 * w_3 + I_4 * w_4 + \dots + I_n * w_n$$



¿QUÉ ES TENSORFLOW?

- Tensorflow es una librería open source multiplataforma de computación numérica.
- Desarrollada por Google
- Es cross-platform. Permite ejecución en GPU, CPU, incluyendo dispositivos móviles y plataformas embebidas.

¿QUÉ ES KERAS?

- API de alto nivel de redes neuronales escrita en Python.
- Se ejecuta encima de Theano o TensorFlow.
- Una de las más sencillas y potentes para desarrollar y evaluar modelos basados en Deep Learning.

¿POR QUÉ USAR KERAS?

- Permite el diseño, entrenamiento y testeo de modelos de forma rápida y sencilla.
- Soporta diversos tipos de redes (no sólo CNNs).
- Se ejecuta perfectamente en CPU y GPU.

INSTALACIÓN (CPU)

1. Descargar Anaconda e instalar
<https://www.anaconda.com/products/individual>
2. `conda create -n cursoDL python=3.7 numpy scipy matplotlib pillow spyder==4`
3. `conda activate cursoDL`
4. `conda install tensorflow==2.1`
5. `conda install keras`
6. `conda install tensorflow-estimator=2.1.0`

INSTALACIÓN (GPU)

1. Descargar Anaconda e instalar
<https://www.anaconda.com/products/individual>
2. `conda create -n cursoDL python=3.7 numpy scipy matplotlib pillow spyder==4`
3. `conda activate cursoDL`
4. Instalar CUDA 10.1 y cuDNN 7.6.5
5. `conda install tensorflow-gpu==2.1`
6. `conda install keras`
7. `conda install tensorflow-estimator=2.1.0`

<https://haroonshakeel.medium.com/installing-tensorflow-2-1-with-cuda-and-cudnn-on-windows-10-c90fa309536a>

EJEMPLOS PRÁCTICOS

- EJEMPLO 1: MNIST
- EJEMPLO 2: Custom dataset

EJEMPLO I: MNIST



EJEMPLO DE PRUEBA

1. Definir dataset
2. Diseñar arquitectura de red
3. Compilar
4. Entrenar y validar

I. DEFINIR DATASET

*** Librerías a importar:**

```
import keras
from keras.datasets import mnist
```

Descarga del dataset MNIST:

```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

Adecuamiento de los datos:

```
# Adaptar las dimensiones:
train_images = train_images.reshape(train_images.shape[0], 28, 28, 1)
test_images = test_images.reshape(test_images.shape[0], 28, 28, 1)
```

I. DEFINIR DATASET

* Librerías a importar:

```
import keras
from keras.datasets import mnist
```

Descarga del dataset MNIST:

```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

Adecuamiento de los datos:

```
# Adaptar las dimensiones:
train_images = train_images.reshape(train_images.shape[0], 28, 28, 1)
test_images = test_images.reshape(test_images.shape[0], 28, 28, 1)
# Normalizar los valores entre 0 y 1:
train_images = train_images.astype('float32')
test_images = test_images.astype('float32')
train_images /= 255
test_images /= 255
```

I. DEFINIR DATASET

* Librerías a importar:

```
import keras
from keras.datasets import mnist
```

Descarga del dataset MNIST:

```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

Adecuamiento de los datos:

```
# Adaptar las dimensiones:
train_images = train_images.reshape(train_images.shape[0], 28, 28, 1)
test_images = test_images.reshape(test_images.shape[0], 28, 28, 1)
# Normalizar los valores entre 0 y 1:
train_images = train_images.astype('float32')
test_images = test_images.astype('float32')
train_images /= 255
test_images /= 255
# Convertir las etiquetas en formato one-hot:
train_labels = keras.utils.to_categorical(train_labels, 10)
test_labels = keras.utils.to_categorical(test_labels, 10)
```

2. DISEÑAR ARQUITECTURA DE RED

* Librerías a importar:

```
from keras import models  
from keras import layers
```

2.1. Crear un modelo

```
model = models.Sequential()
```

2. DISEÑAR ARQUITECTURA DE RED

* Librerías a importar:

```
from keras import models
from keras import layers
```

2.1. Crear un modelo

```
model = models.Sequential()
```

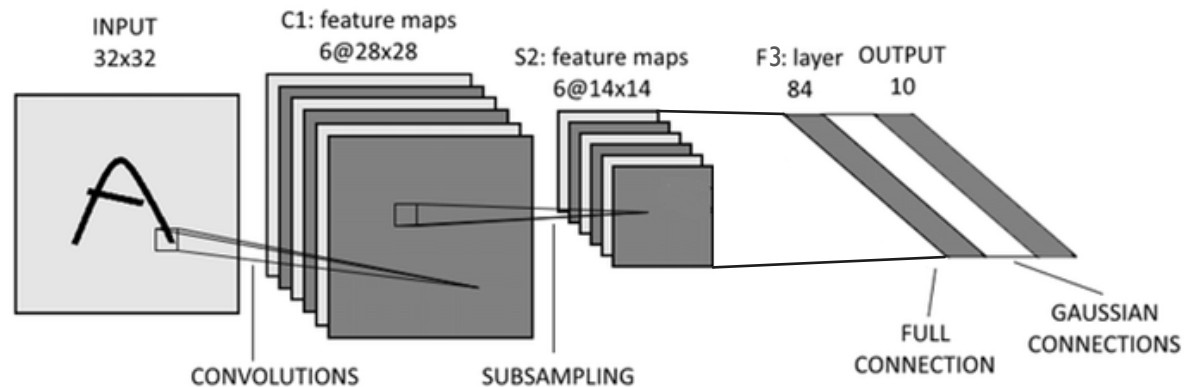
2.2. Añadir capas

- **Covolución:** Conv2D, ...
- **Pooling:** MaxPooling2D, AveragePooling2D, ...
- **Core:** Dense, Dropout, Flatten, ...
- Etc

Tipos de capas en Keras:
<https://keras.io/api/layers/>

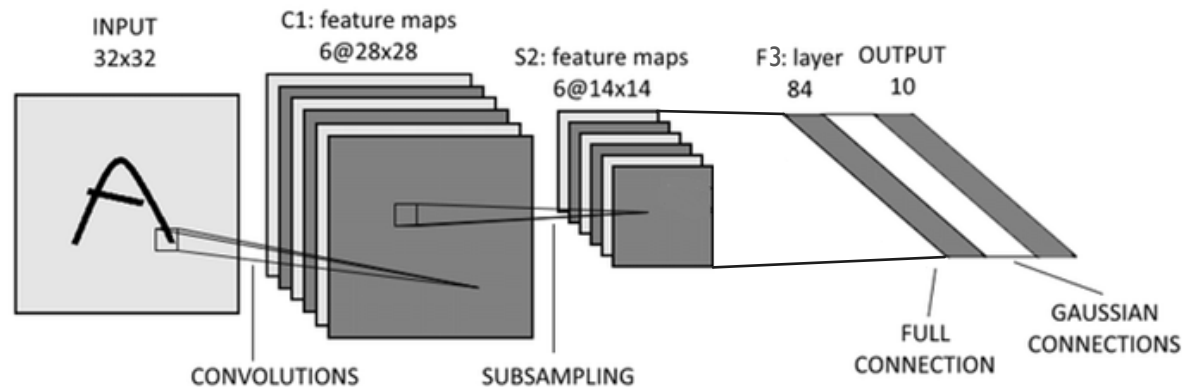
2. DISEÑAR ARQUITECTURA DE RED

2.2. Añadir capas



2. DISEÑAR ARQUITECTURA DE RED

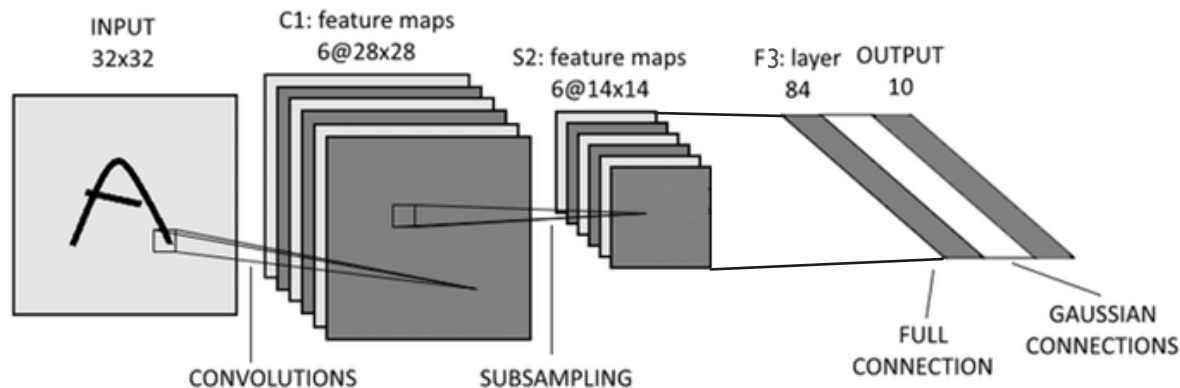
2.2. Añadir capas



```
model.add(layers.Conv2D(filters=6, kernel_size=(5, 5), input_shape=(28,28,1)))
```

2. DISEÑAR ARQUITECTURA DE RED

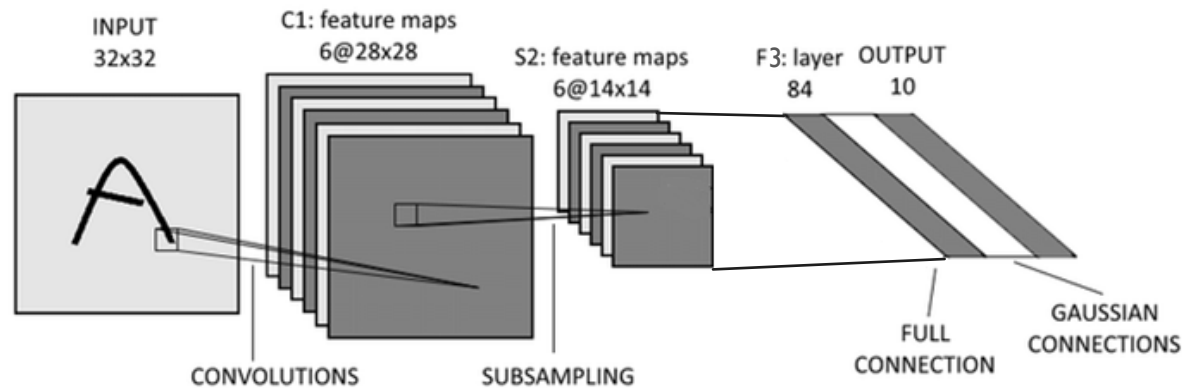
2.2. Añadir capas



```
model.add(layers.Conv2D(filters=6, kernel_size=(5, 5), input_shape=(28,28,1)))  
model.add(layers.Activation('relu'))
```

2. DISEÑAR ARQUITECTURA DE RED

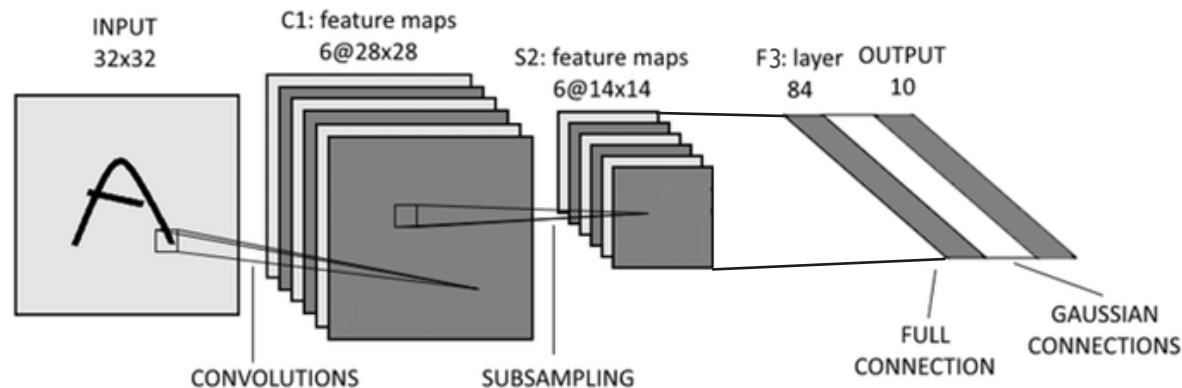
2.2. Añadir capas



```
model.add(layers.Conv2D(filters=6, kernel_size=(5, 5), input_shape=(28,28,1)))  
model.add(layers.Activation('relu'))  
model.add(layers.MaxPooling2D(pool_size=2))
```

2. DISEÑAR ARQUITECTURA DE RED

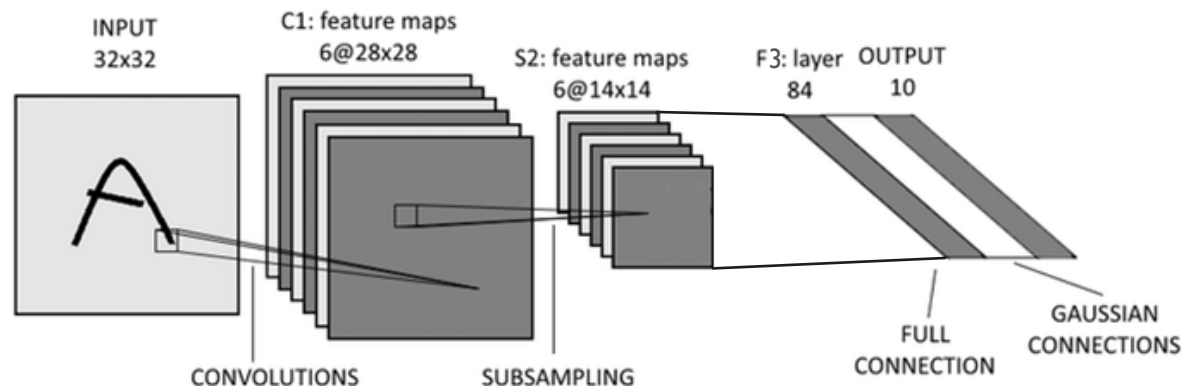
2.2. Añadir capas



```
model.add(layers.Conv2D(filters=6, kernel_size=(5, 5), input_shape=(28,28,1)))  
model.add(layers.Activation('relu'))  
model.add(layers.MaxPooling2D(pool_size=2))  
model.add(layers.Flatten())
```

2. DISEÑAR ARQUITECTURA DE RED

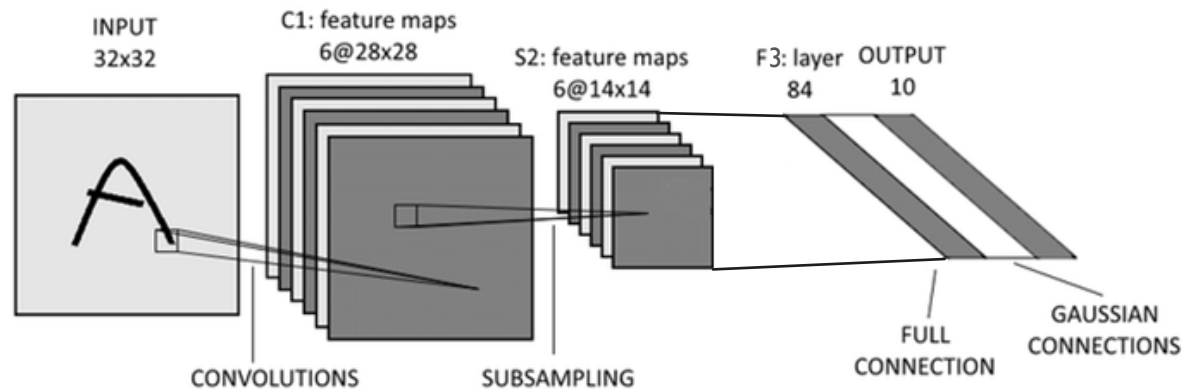
2.2. Añadir capas



```
model.add(layers.Conv2D(filters=6, kernel_size=(5, 5), input_shape=(28,28,1)))
model.add(layers.Activation('relu'))
model.add(layers.MaxPooling2D(pool_size=2))
model.add(layers.Flatten())
model.add(layers.Dense(units=84))
```

2. DISEÑAR ARQUITECTURA DE RED

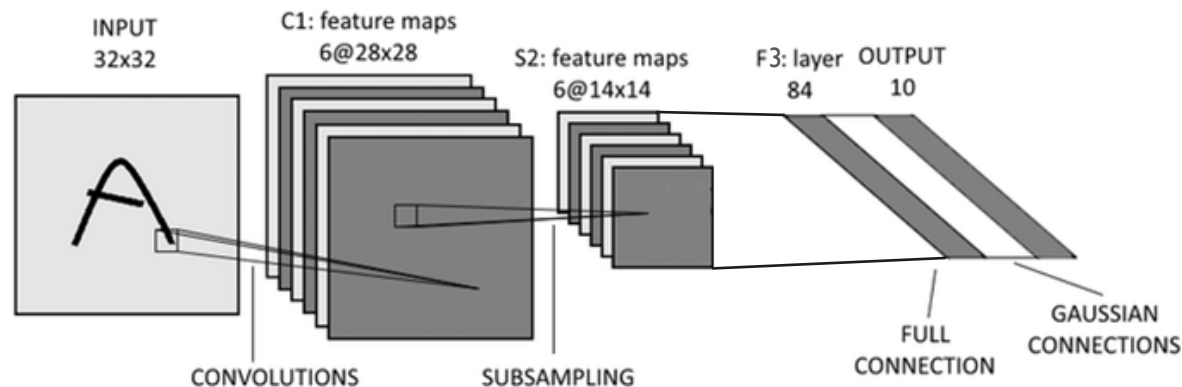
2.2. Añadir capas



```
model.add(layers.Conv2D(filters=6, kernel_size=(5, 5), input_shape=(28,28,1)))
model.add(layers.Activation('relu'))
model.add(layers.MaxPooling2D(pool_size=2))
model.add(layers.Flatten())
model.add(layers.Dense(units=84))
model.add(layers.Activation('relu'))
```


2. DISEÑAR ARQUITECTURA DE RED

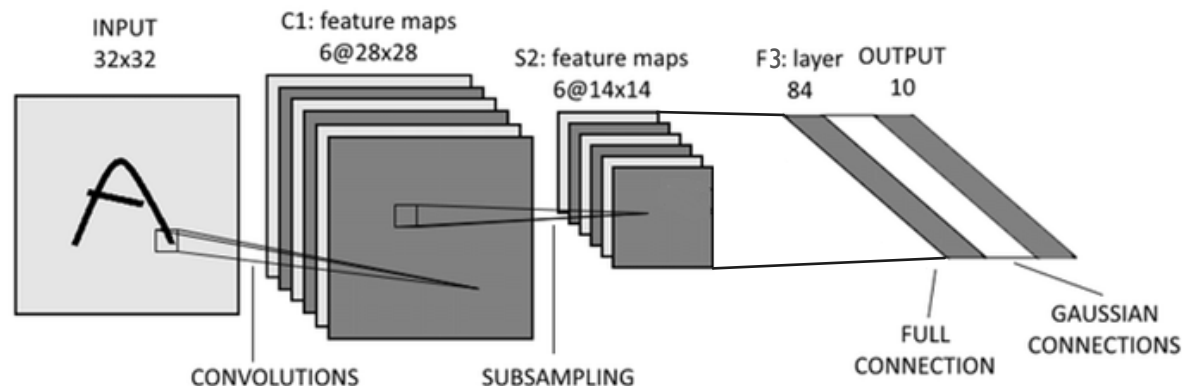
2.2. Añadir capas



```
model.add(layers.Conv2D(filters=6, kernel_size=(5, 5), input_shape=(28,28,1)))
model.add(layers.Activation('relu'))
model.add(layers.MaxPooling2D(pool_size=2))
model.add(layers.Flatten())
model.add(layers.Dense(units=84))
model.add(layers.Activation('relu'))
model.add(layers.Dense(units=10))
```

2. DISEÑAR ARQUITECTURA DE RED

2.2. Añadir capas



```
model.add(layers.Conv2D(filters=6, kernel_size=(5, 5), input_shape=(28,28,1)))
model.add(layers.Activation('relu'))
model.add(layers.MaxPooling2D(pool_size=2))
model.add(layers.Flatten())
model.add(layers.Dense(units=84))
model.add(layers.Activation('relu'))
model.add(layers.Dense(units=10))
model.add(layers.Activation('softmax'))
```

3. COMPILAR

Configurar proceso de aprendizaje con `.compile()`

```
model.compile(optimizer='sgd',  
              loss='categorical_crossentropy',  
              metrics=['accuracy'])
```

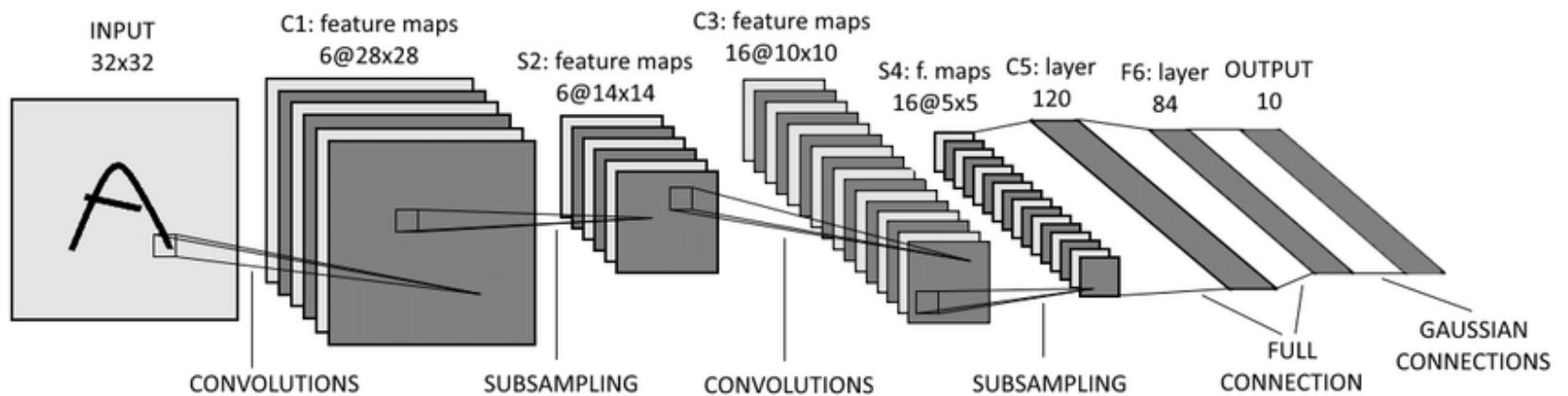
- **Optimizer:** SGD, Adam, RMSProp, ...
- **Loss:** mean_squared_error, categorical_crossentropy, ...
- **Metrics** (opcional)

4. ENTRENAR Y VALIDAR

Entrenar con `.fit()`

```
train = model.fit(train_images,  
                  train_labels,  
                  epochs=5,  
                  batch_size=32,  
                  validation_data=(test_images, test_labels))
```

EJERCICIO



1. Implementar arquitectura LeNet
2. Realizar un entrenamiento

EJEMPLO 2: CUSTOM DATASET

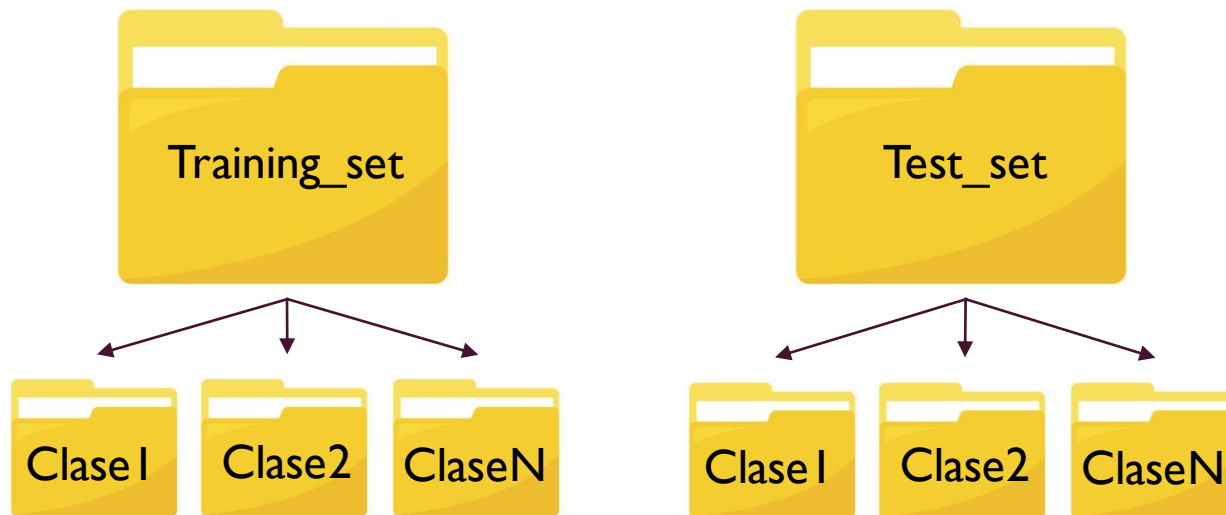


Enlaces de interés para más datasets:
<https://www.kaggle.com/>

EJEMPLO REAL

1. Definir dataset
2. Diseñar arquitectura de red
3. Compilar
4. Entrenar y validar
5. Testear

I. DEFINIR DATASET



I. DEFINIR DATASET

* Librerías a importar:

```
from keras.preprocessing.image import ImageDataGenerator
```

```
train_datagen = ImageDataGenerator(rescale=1./255, horizontal_flip=True,  
vertical_flip=True)
```

```
val_datagen = ImageDataGenerator(rescale=1./255)
```

```
test_datagen = ImageDataGenerator(rescale=1./255)
```

Documentación sobre ImageDataGenerator:

https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

I. DEFINIR DATASET

Train:

```
train_generator = train_datagen.flow_from_directory( path_train,  
    target_size=(128, 128),  
    batch_size=32)
```

Validación:

```
val_generator = val_datagen.flow_from_directory( path_val,  
    target_size=(128, 128),  
    batch_size=32)
```

Test:

```
test_generator = test_datagen.flow_from_directory( path_test,  
    target_size=(128, 128),  
    batch_size=32)
```

2. DISEÑAR ARQUITECTURA DE RED

Partimos de LeNet:

```
model = models.Sequential()  
model.add(layers.Conv2D(filters=6, kernel_size=(5, 5), input_shape=(128,  
128,3)))  
model.add(layers.Activation('relu'))  
model.add(layers.MaxPooling2D(pool_size=2))  
  
model.add(layers.Conv2D(filters=16, kernel_size=(5, 5)))  
model.add(layers.Activation('relu'))  
model.add(layers.MaxPooling2D(pool_size=2))  
  
model.add(layers.Flatten())  
model.add(layers.Dense(units=120))  
model.add(layers.Activation('relu'))  
model.add(layers.Dense(units=84))  
model.add(layers.Activation('relu'))  
model.add(layers.Dense(units=2))  
model.add(layers.Activation('softmax'))
```

3. COMPILAR

Configurar proceso de aprendizaje con .compile()

```
model.compile(optimizer='sgd',  
              loss='categorical_crossentropy',  
              metrics=['accuracy'])
```


4. ENTRENAR Y VALIDAR

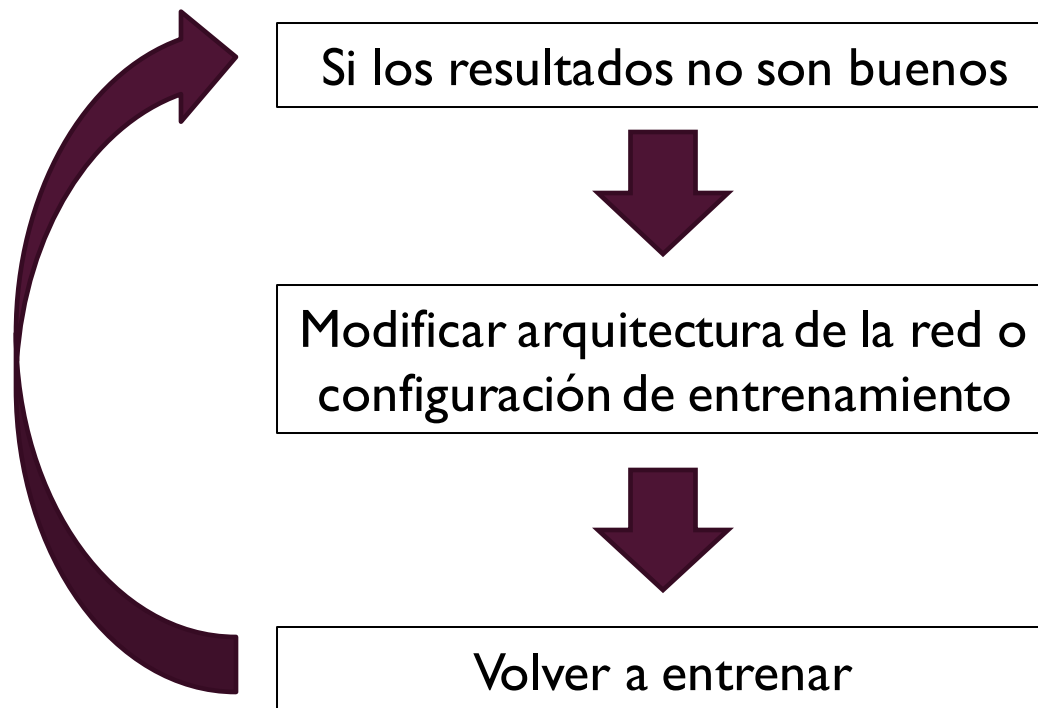
* Librerías a importar:

```
import matplotlib
import matplotlib.pyplot as plt
```

Representación de la evolución del accuracy y loss:

```
ent_acc = train.history['acc']
val_acc = train.history['val_acc']
ent_loss = train.history['loss']
val_loss = train.history['val_loss']
epochs = range(len(ent_acc))
plt.plot(epochs, ent_acc, 'bo', label='Entrenamiento')
plt.plot(epochs, val_acc, 'b', label='Validación')
plt.title('Accuracy Entrenamiento y Validación')
plt.legend()
plt.figure()
plt.plot(epochs, ent_loss, 'bo', label='Entrenamiento')
plt.plot(epochs, val_loss, 'b', label='Validación')
plt.title('Loss Entrenamiento y Validación')
plt.legend()
plt.show()
```

4. ENTRENAR Y VALIDAR



5. TESTEAR

Test con `.evaluate_generator()`

```
test_loss, test_acc = model.evaluate_generator(test_generator)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

EJERCICIO:

1. Definir dataset
2. Implementar una CNN
3. Entrenar
4. Obtener representación de la evolución del accuracy y loss