Convolutional Networks CNN’s

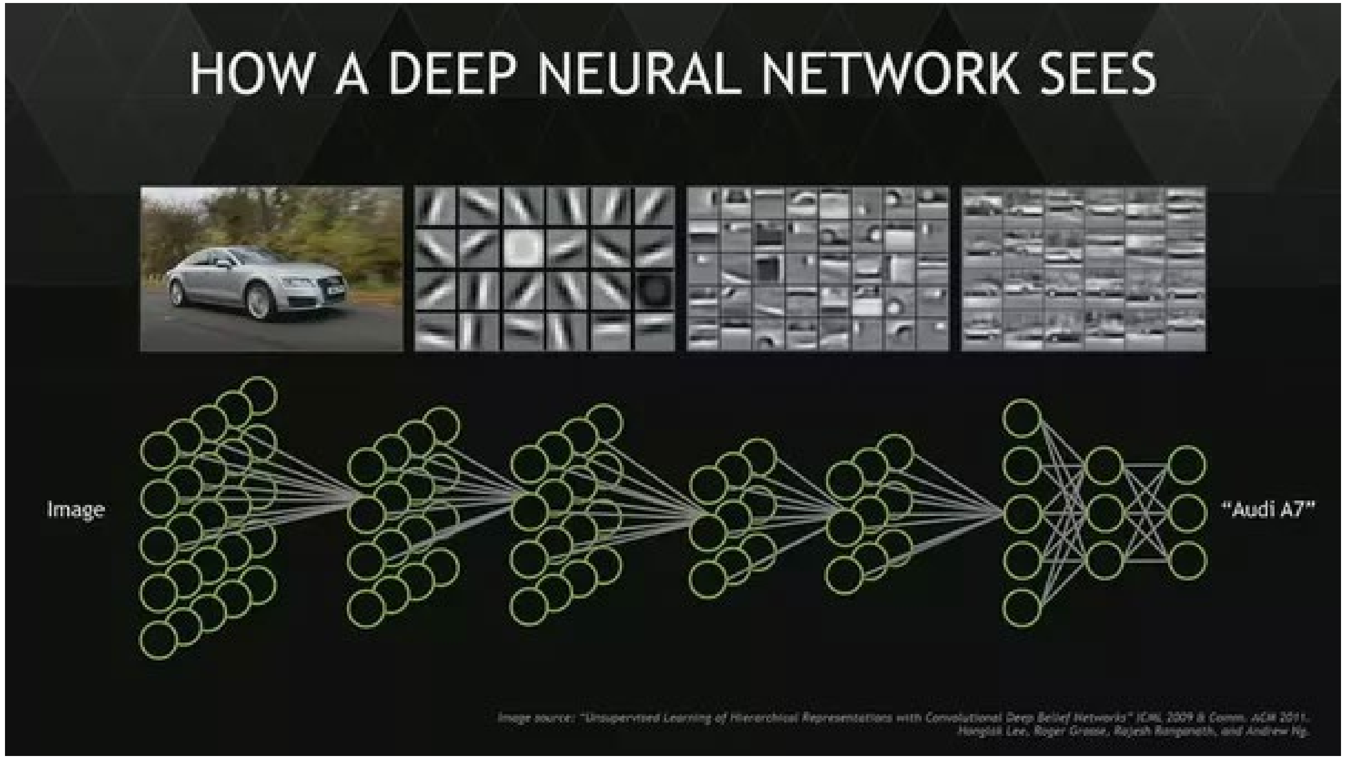
Silverio García Cortés
Universidad de Oviedo
Naples. May 2023, Univ. Federico II

How computers see?

Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

25 43 11 04 70 87 12 31 43 10 05 77 12 06 45 09 29 30 02
56 22 75 03 22 96 45 12 23 03 77 67 81 45 22 04 90 22 21
32 45 41 91 87 62 35 02 00 11 62 25 43 11 04 70 87 12 61
31 43 10 05 77 12 06 45 09 29 30 56 22 75 03 22 96 45 05
12 23 03 77 67 81 45 22 04 90 22 32 45 41 91 87 62 35 44
02 00 11 62 25 43 11 04 70 87 12 31 43 10 05 77 12 06 10
45 09 29 30 56 22 75 03 22 96 45 12 23 03 77 67 81 45 55
22 04 90 22 32 45 41 91 87 62 35 02 00 11 62 25 43 11 80
04 70 87 12 31 43 10 05 77 12 06 45 09 29 30 56 22 75 08
03 22 96 45 12 23 03 77 67 81 45 22 04 90 22 32 45 41 99
91 87 62 35 02 00 11 62 22 01 00 72 65 23 01 00 22 04 30
90 22 32 45 41 91 87 62 35 02 00 11 62 25 43 11 04 70 42
87 12 31 43 10 05 77 12 06 45 09 29 30 56 22 75 03 22 91
96 45 12 23 03 77 67 81 45 22 04 90 22 32 45 41 91 87 40
62 35 02 00 11 62 22 01 00 72 65 23 01 00 56 22 75 03 67
22 96 45 12 23 03 77 67 81 45 22 04 90 22 32 45 41 91 22

What we see **What computers see**



- A machine will detect the features from the input image. First, it will try to identify the edges. From the edges that it identified it will try to form shapes i.e. a tire and from the tyre the full car shape!!!!.

[CNN and Regularization Techniques with TensorFlow and Keras]()

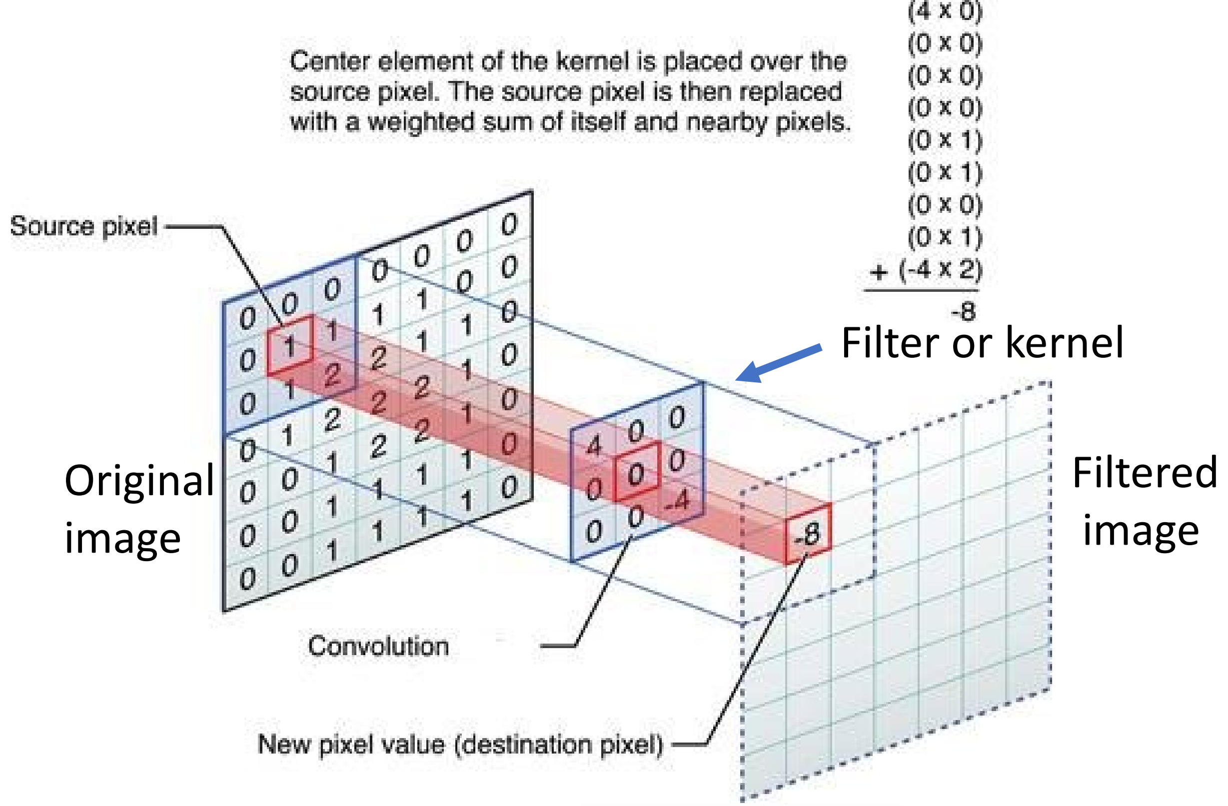
Convolution Operation. Filters, kernels

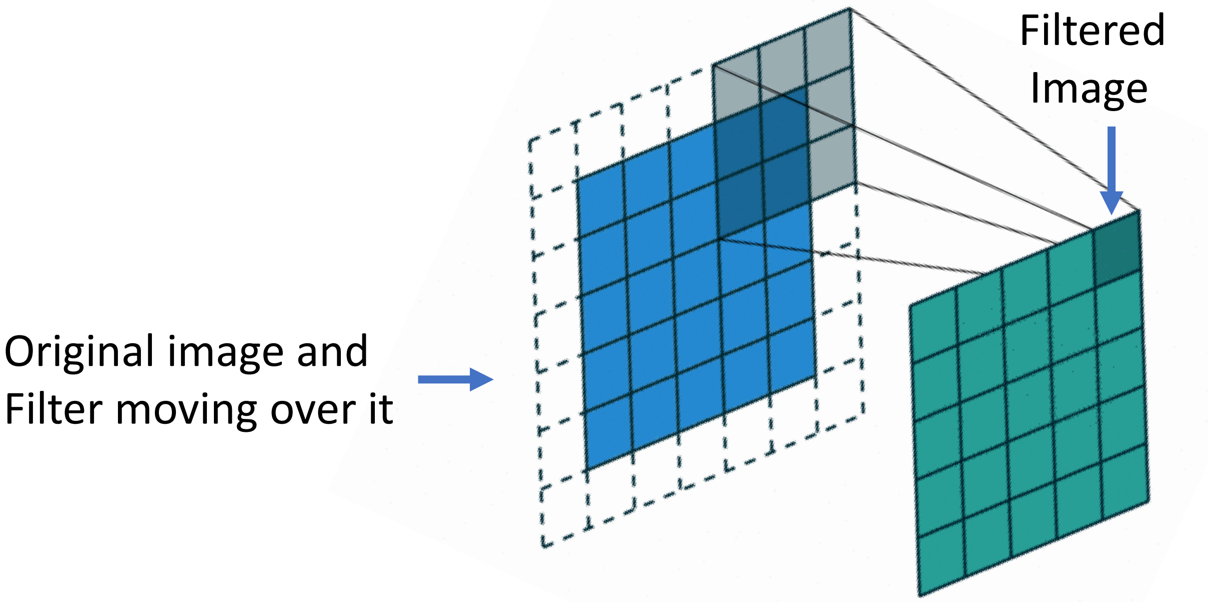
Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

- The procedure, known as convolution, uses a matrix (also called kernel, mask, template,filter...) of much smaller dimensions than the original image (e.g. 3x3, 5x5, 7x7,...).
- This filter is sequentially superimposed on each pixel of the original image
- The filter contains the weights with which the ND of each pixel under it will influence the resulting value for the ND of the pixel of the new image located under the central pixel of the mask.
- For each filter position the following calculation is performed:

$$G = F \star W \qquad G[i,j] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} F[i-k, j-l]W[k,l]$$

Image　　Filter





Some Spatial Filtering results…

Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

Blurring

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Gaussian Blurring

0	0	0	5	0	0	0
0	5	18	32	18	5	0
0	18	64	100	64	18	0
5	32	100	100	100	32	5
0	18	64	100	64	18	0
0	5	18	32	18	5	0
0	0	0	5	0	0	0

Line detection

-1	-1	-1
2	2	2
-1	-1	-1

Horizontal lines

-1	2	-1
-1	2	-1
-1	2	-1

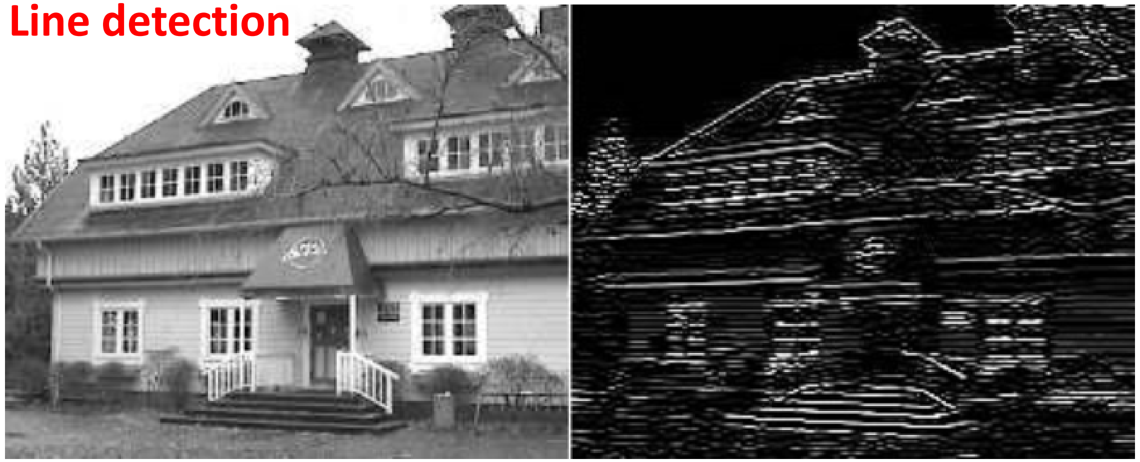
Vertical lines

-1	-1	2
-1	2	-1
2	-1	-1

45 degree lines

2	-1	-1
-1	2	-1
-1	-1	2

135 degree lines



More filters …

Sobel operator

-1	-2	-1
0	0	0
1	2	1

Horizontal

-1	0	1
-2	0	2
-1	0	1

Vertical

Laplacian operator

0	-1	0
-1	4	-1
0	-1	0

he laplacian operator

-1	-1	-1
-1	8	-1
-1	-1	-1

The laplacian operator (include diagonals)

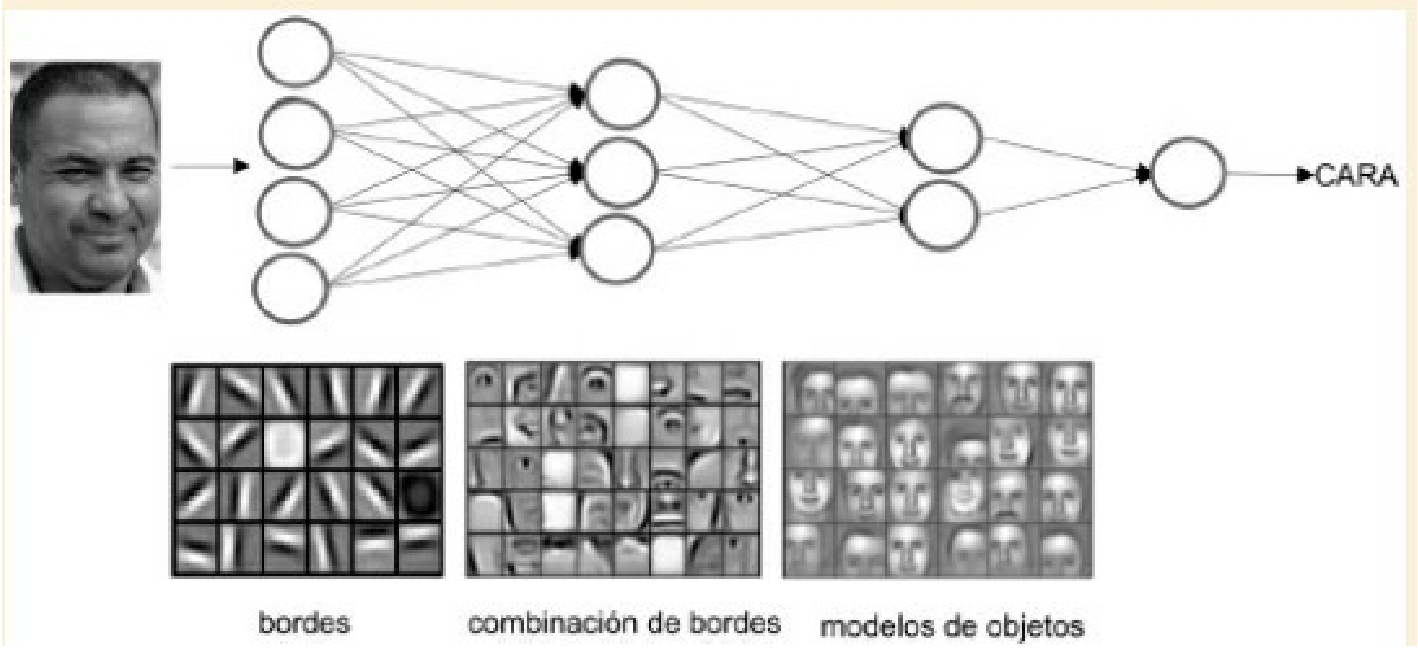
LoG: Laplacian of Gaussian

0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

And many more!!!!!!

Deep Learning: Convolutional Networks: CNN's, ConvNets

- CNNs are networks based on the **convolution operation** and explicitly assume that the inputs are images. Each layer learns basic elements, (e.g. the first layer will learn edges, the second one, more complex patterns of grouping those edges and so on, until it learns very complex patterns such as a face).

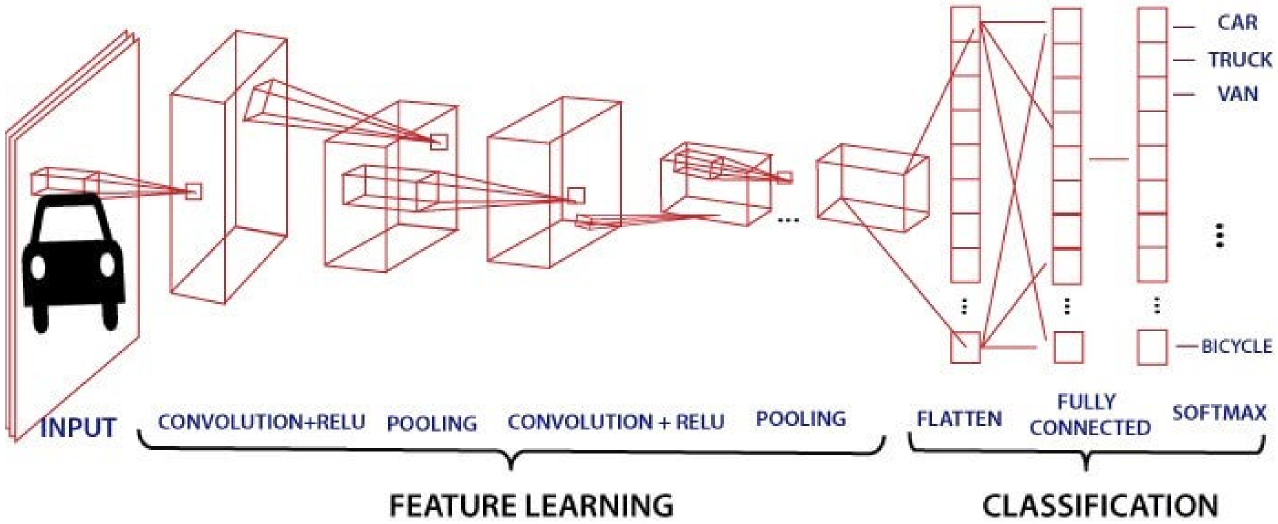


- The number of parameters of these networks is much smaller than that of MLP (multilayer perceptron) networks, which is an advantage over them.

- In addition, CNNs can naturally capture spatial patterns no matter where they are located in the image space, instead of memorizing the pattern and the location in pixels.

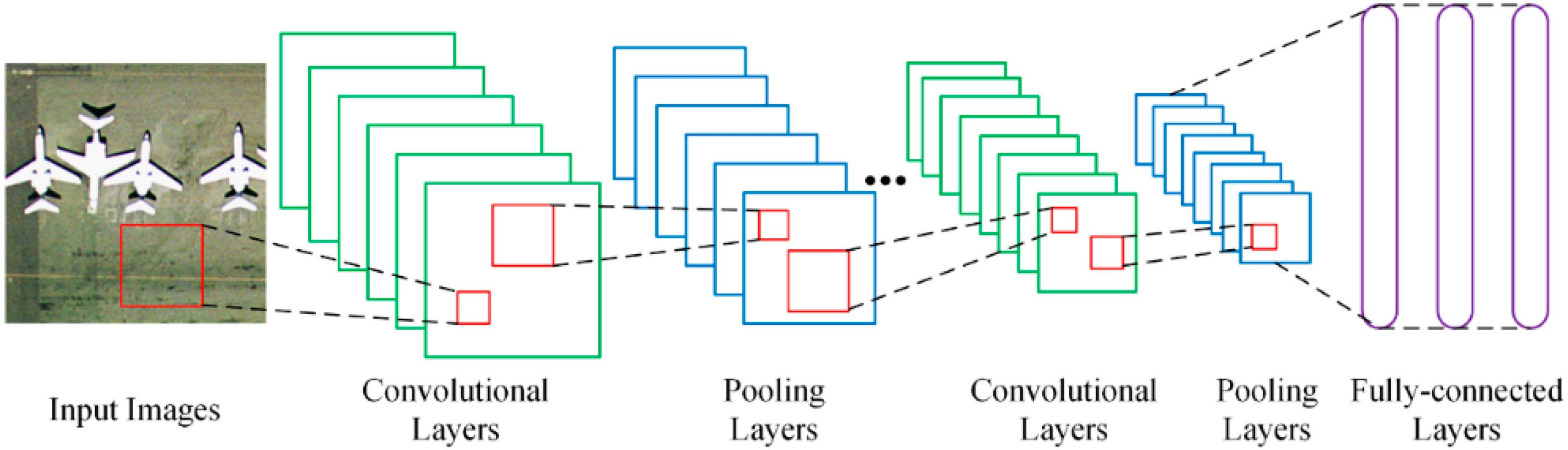
Characteristics of CNN's

- The parameters learned in a CNN are **the kernels (filters) of the convolution operations**. In each layer a number of filters are learned and each filter generates a new image which is called a "feature map". The filters together with their associated outputs are known as channels.
- In an MLP , a weight is associated with each pixel but **in a CNN the weights are associated with the filter elements** which can extract features anywhere in the image. Example:A MLP working with an image of 32x32 pixels if it has a first layer of 256 neurons will have to determine: 32x32x256+256 = 262400 parameters. In other words, each neuron receives 32x32 inputs with its weights and each neuron has an additional bias that must be determined. In a CNN that has 256 filters in the first layer and each filter has 3x3=9 parameters, it will be necessary to determine: 9x256+256 = 2560, i.e. 9 parameters per filter plus a bias per filter. The number of parameters is therefore much smaller than in MLP and yet in CNNs we learn filters that extract patterns that can appear at different positions in the image.
- We can say that the purpose of CNNs is to learn to detect visual features that can then appear at any position in the image. **A fully connected network such as the MLP would have to learn the feature each time it was presented in a different location**.

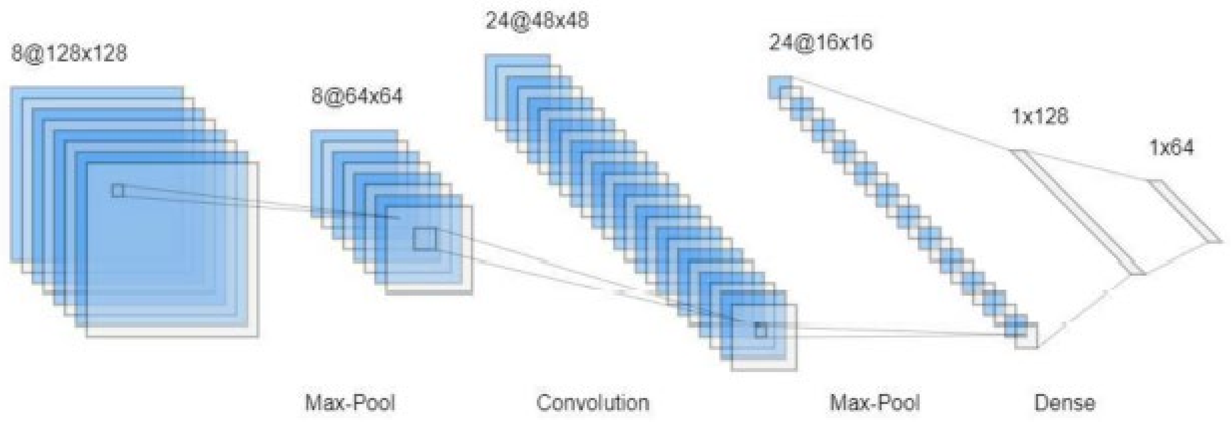


CNN's: Structure and components

- In the typical structure of convolutional NN's, **convolutional layers** are combined with layers of **subsampling** operations and finally **fully connected** (dense) layers.
- In the first layers the filters learn to detect very basic features (edges, lines, dots) the following layers learn to detect combinations of the previous features (leaf, wheel, etc.) which constitute higher level concepts.
- The **deeper layers** lose special information due to subsampling while the **number of channels increases**. Finally the content of the feature maps feeds into dense (fully connected) layers for classification.



CNN's: Structure and components in Keras (I)



Tensor Input data dimensions:
[number of. Images (num samples), height (rows), width (cols), channels].
Example:
Tensor dims (25000, 300, 450, 3)

Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

- Convolutional layers: They are declared with the module keras.layers.Conv2D.

```
from keras.layers import Conv2D
```

- The number of filters in the layers and the size of the filter kernel are indicated, either with a tuple or an integer if the kernel is square.

```
Conv2D(8, 5, activation='relu')
```

- In the **first layer** of a CNN, the **size of the input (height, width, channels)** must be indicated, not flattened as was the case in MLP's. The form of the input matrices (tensor) will be: **(height, width, channels)**.The rest of the convolutional layers automatically adjust their inputs to the outputs of the previous one.

```
Conv2D(16, 3, strides = (2,2), activation='relu', padding = 'same', input_shape = (28,28,1))
```

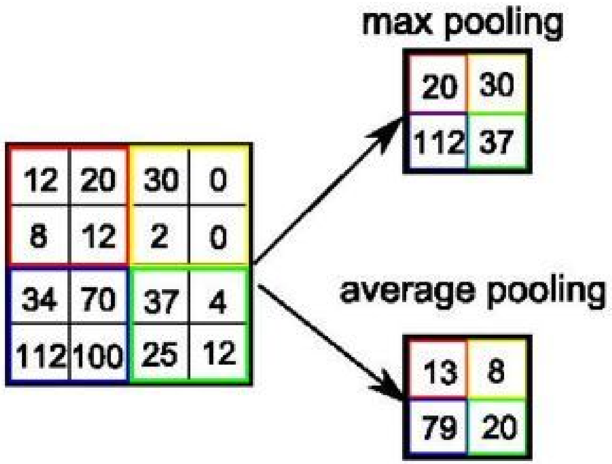
- The displacement of the kernel over the image in the convolution is variable and is adjusted with the parameter **"strides".** It can be specified with a tuple or an integer (h,w).

- **Padding** is the option to apply zeros to the edges of the image so that the filtered image is the same size as the input image. In Keras there are two padding options: 'valid', 'same', valid adds no zeros and same adds so that the output is the same as the input divided by the stride if it is greater than 1.

CNN´s: Componentes y estructura en Keras (II)

Pooling:

- These pooling layers are used in combination with the convolutional layers and apply a scrolling window over the feature maps that serve as input and return an output statistic (e.g. maximum or mean). This is where the stride is applied and produces a dimensional reduction of the maps.

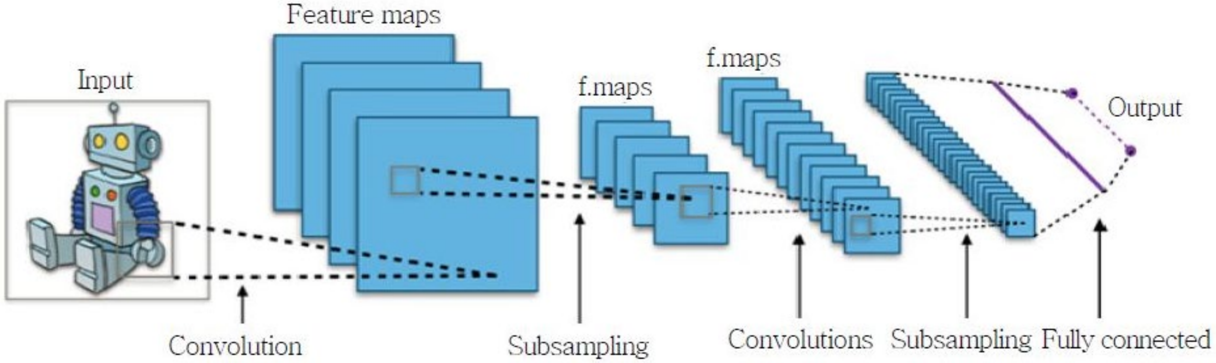


Max pooling and average pooling size (2,2) stride (2,2)

- The application of pooling improves the invariance to translations and spatial distortions as well as reducing the dimensionality. In this way it is possible to increase the depth of the network since the number of parameters to be trained is reduced.
- The two most commonly used types of pooling are **maxpooling** and **average pooling** (maximum and average respectively). In Keras it is necessary to indicate the size of the pooling window and the stride, which by default is equal to the size of the pooling window..

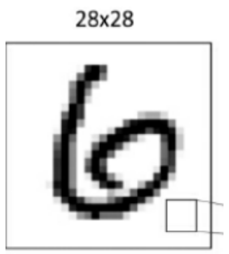
```
MaxPooling2D(pool_size=2)
```

AveragePooling2D()



CNN example in Keras: Written character recognizing (I)

- We will use the MNIST (National Institute of Standards and Technology) dataset consisting of 60000 8-bit 28x28 pixel images with handwritten numeric characters.



- The problem is a multi-class classification. We want to give CNN an image and have it recognise the digit.

- The network is created with a Sequential model and we import Convolution, Pooling, Dense. Also the Flatten layer. The flatten layer will be used to change the dimensions of the outputs of the last convolutional layer to adapt them to the inputs of the Dense layer.

- Recall that the outputs are one-hot vectors (with the probabilities of each image being each number from 0 to 9) and in addition it is necessary to normalise inputs.

- Data loading, one-hot encoding, normalisation, addition of number of channels as a dimension

- Some types of network layers

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Conv2D
from keras.layers import MaxPooling2D, Flatten
```

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
#Codificación one-hot
y_train = keras.utils.to_categorical(y_train)
y_test = keras.utils.to_categorical(y_test)
#Normalización de entradas
x_train = x_train/255
x_test = x_test/255
x_train = x_train.reshape(60000, 28, 28, 1)
x_test = x_test.reshape(10000, 28, 28, 1)
```

CNN example in Keras: Written character recognizing (II)

- Network design:

```
cnn = Sequential()
cnn.add(Conv2D(16, 3, activation='relu', input_shape=(28, 28, 1)))
cnn.add(MaxPooling2D(pool_size=2))
cnn.add(Conv2D(32, 3, activation='relu'))
cnn.add(MaxPooling2D(pool_size=2))
cnn.add(Flatten())
cnn.add(Dense(32, activation='relu'))
cnn.add(Dense(10, activation='softmax'))
```

- This network will have 2 convolutional layers without stride and with padding valid (default),
- Number of filters: layer1 16, layer2 32, kernels 3x3
- After each layer there is a maxpooling, 2x2, and stride (2,2) (default).
- The Flatten layer is used for reshaping the pooling outputs.
- At the end there are 2 Dense layers, the last one is the output layer and has to have as many neurons as classes of this multiple classification problem. And also softmax activation function (the right one for multi classification problems).

```
cnn.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 16)        160
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 32)        4640
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 800)               0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                25632
_________________________________________________________________
dense_2 (Dense)              (None, 10)                330
=================================================================
Total params: 30,762
```

If padding is valid (default in pooled layers)The output of the conv_2d_1 layer to the pool layer is smaller than the input (goes from 26x26 to 13x13)

Flatten transforms the output layer max pooling (5,5,32) in (800) (5x5x32=800)

CNN example in Keras: Written character recognizing (III)

Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

- **Compiling model and storing históry**:

```
cnn.compile(loss='categorical_crossentropy',
            optimizer='adam',
            metrics=['acc'])
```

```
hist = cnn.fit(x_train, y_train, batch_size=128, epochs=10)
```

- **Model Validation:**

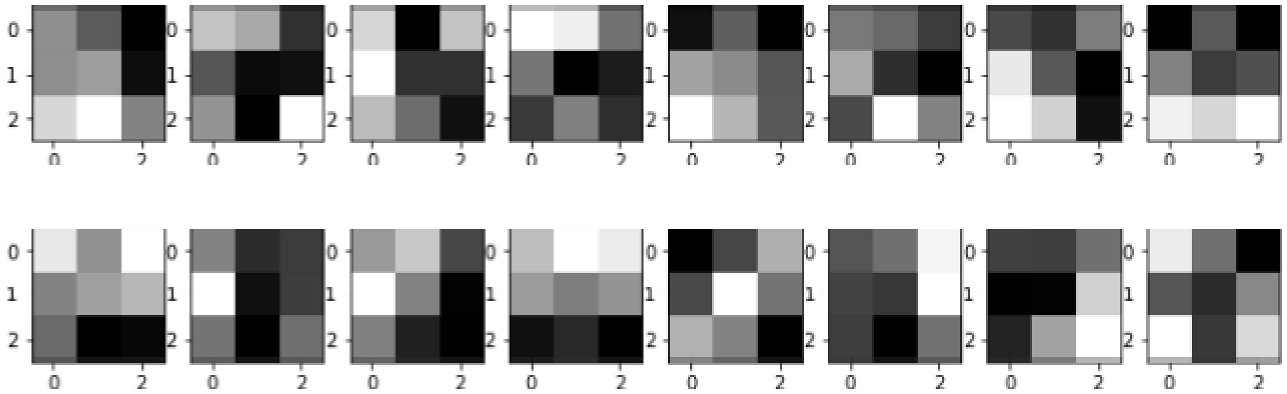
```
acc_train = cnn.evaluate(x_train, y_train)[1]*100
acc_test = cnn.evaluate(x_test, y_test)[1]*100
```

```
print('Exactitud entrenamiento: ', format(acc_train,'.2f'), '%')
print('Exactitud test: ', format(acc_test, '.2f'), '%')
```

```
60000/60000 [==============================] - 2s 35us/step
10000/10000 [==============================] - 0s 35us/step
Exactitud entrenamiento:  99.48 %
Exactitud test:  98.92 %
```

- **Filter visualization**

```
fig, axs = plt.subplots(2,8, figsize=(12, 4))
axs = axs.ravel()
for i,ax in enumerate(axs):
    axs[i].imshow(filtros_capa1[0][:,:,0,i], cmap='binary')
```



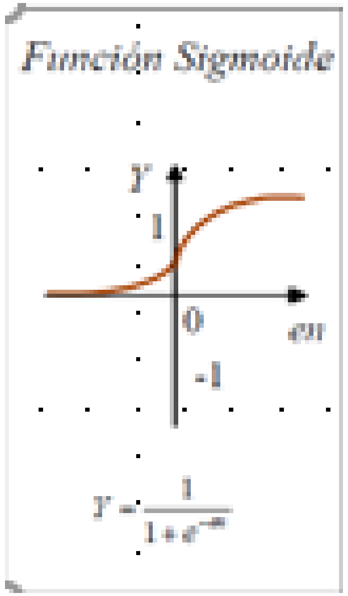
Each filter of the first layer will be activated in the presence of the patterns shown by its kernels.

CNN´s: Batch Normalization and Spatial Dropout 2D

- Batch Normalization is a technique that increases the speed and stability of neural network training by allowing higher learning rates to be employed.
- It is based on the normalisation of the activations of each training batch that are used as input to successive layers. The mean is made to be 0 and the variance 1.
- Normalising the activations also introduces some regularisation as it decreases the "expressiveness" of the model.
- To compensate for this, new parameters are added to be optimised. These parameters multiply the activations once normalised. If we use images with ND's 255 the input of the activation function can be very large. At these high values the logistic (sigmoid) function gives outputs in the asymptotic zone which has almost zero slope.
- **KERAS coding fro Batch Normalization**

```
from keras.layers import BatchNormalization
```

```
cnn.add(Conv2D(16, 5, activation='relu',
               input_shape=(64, 64, 3)))
cnn.add(BatchNormalization())
cnn.add(Conv2D(16, 5, activation='relu')
```

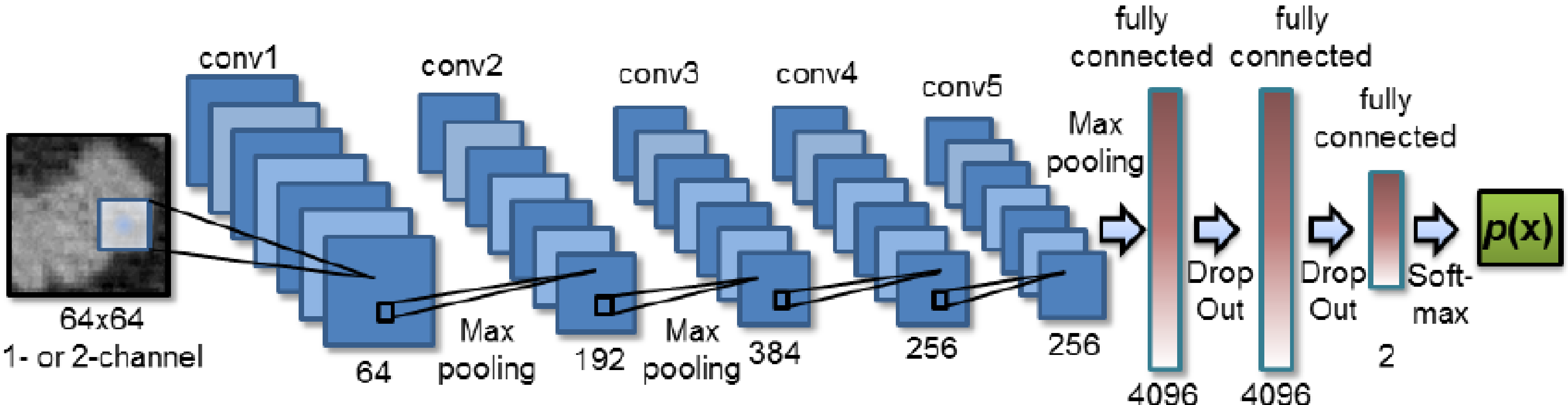


- **SPATIAL DROPOUT 2D**
  - Technique that allows the elimination of some complete channels (similar to the dropout of dense layers).
  - It is used after convolutional layers.

```
from keras.layers import SpatialDropout2D
```

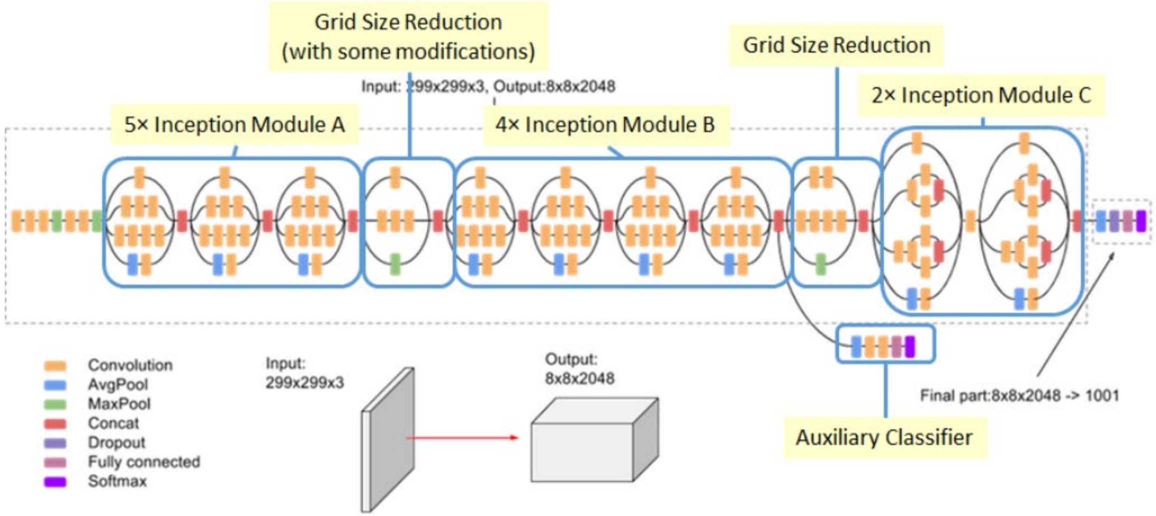
Convnets (CNN´s) Design common practices

- The size of the kernels is usually odd (most often 3x3, 5x5, 7x7) and influences the size of the patterns it will detect.
- The larger the kernel size, the larger the patterns. The first layers use a small number of channels. As they go deeper, they tend to increase the number of channels containing less spatial (situational) and more abstract (high-level) information.
- It is common to use 2x2 strides and then double the number of filters in later layers.
- Convolutional and pooling layers are stacked by combining one of each in turn (or two convolutions before pooling).
- Batch normalisation adds regularisation, so the learning rate can be increased, and the dropout can be decreased.
- Dropout is used in fully connected networks (in special convolutions).
- Memory problems (OOM) can be addressed by: reducing batch size, decreasing channels and layers, subsampling using strides (more times or with higher values).



CNN's: Transfer Learning

- Transfer learning consists of taking advantage of a model that has already learned to generate representations in one task to achieve a result in another with a much smaller amount of data and time.
- It is developed from computer vision competitions such as: ImageNet Large Scale Visual Recognition Challenge (ILSVRC) http://image-net.org/challenges/LSVRC/
- or large image databases such as: Image OpenData.Since 2010, popular CNN network architectures have been developed that improve the performance and work with an image database that has 1000 classes (ImageNet) and more than a million images.
- The architectures are different designs of CNN neural networks that have been tried and tested and give good results. It is possible to import already trained weights from these networks, it is also possible to slightly modify the network design.
- Not all architectures are available at keras..... Available architectures can be found at: https://keras.io/api/applications/. These models can be used for prediction, pattern extraction, or fine tuning of that model in another type of task with much less training.



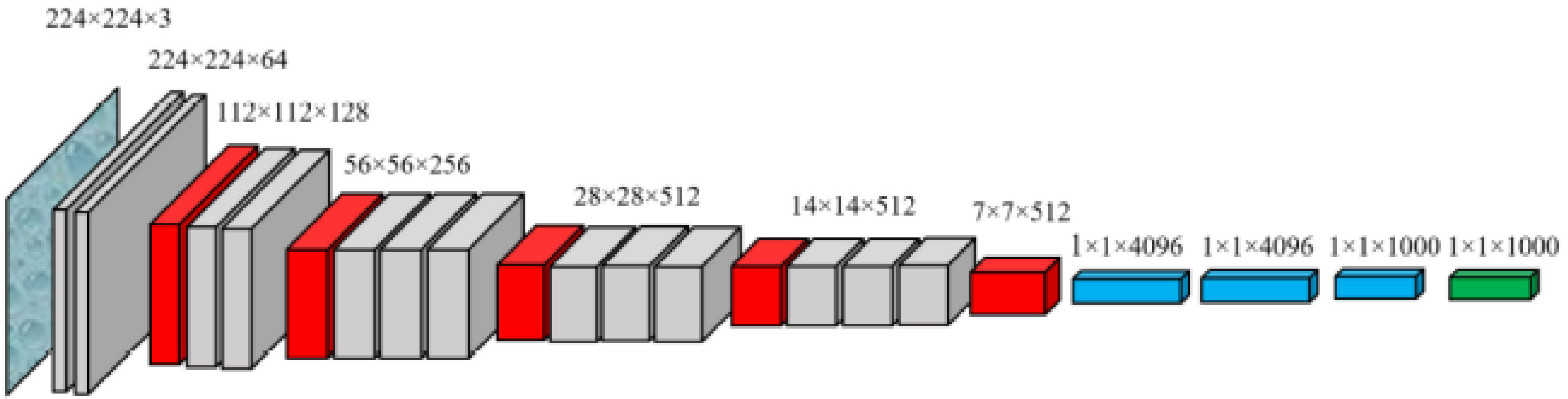
InceptionV3 Architecture

CNN's: Transfer Learning (II)

There are different ways to take advantage of transfer learning:

- **Extract features from an image** using the values of intermediate layers of the network, and use these features as inputs for other Machine Learning models (other than RN's), for example a SVM (Supported Vector Machines). (Feature extraction)
- **Train only a part of the model** (architecture), i.e. only a few layers of the network to perform a specific task that might be different from the one used for training the whole network. This saves a lot of computational time and effort. This method is done by "freezing" some layers of the network that will not be re-trained. (Fine tuning)
- **Re-training the entire model:** This is more expensive and is carried out in phases, re-training only the final layers (the rest of the network is frozen so that the error propagating backwards does not undo the acquired knowledge). Once the final layers are trained, the rest of the network is unfrozen and the whole model is trained together.

- Model weights can be saved for use in predictions without the need to train the entire model each time.
- A commonly used format to store model weights is "*.h5".



VGG16 architecture

Arquitectura VGG, las capas grises son convoluciones, las rojas *pooling*, las azules capas totalmente conectadas y la verde *softmax*.

Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

References:

- Raschka Sebastian, Mirjalili Vahid, 2017. Python Machine Learning. 2^{nd} ed. Packt Birmingham-Mubai
- Brownlee Jason. (2019). Deep Learning with Python: Develop Learning Models on Theano and TensorFlow using Keras. (Machine Learning Mastery)
- Torres Jordi. 2020. Python Deep Learning: Introducción práctica con Keras y Tensorflow 2. 1ª ed. Marcombo.
- Chollet, Francois. 2020. Deep Learning con Python. Ed ANAYA
- The MNIST dataset. http://yann.lecun.com/exdb/mnist/ . Accessed Dec. 2020
- German Research Center for Artificial Inteliggence. Multimedia análisis and Data Minign. Dowloads. Datasets for Machine Learning. Downloads — Multimedia Analysis and Data Mining Research Group (dfki.de) . Accessed Dec. 2020.