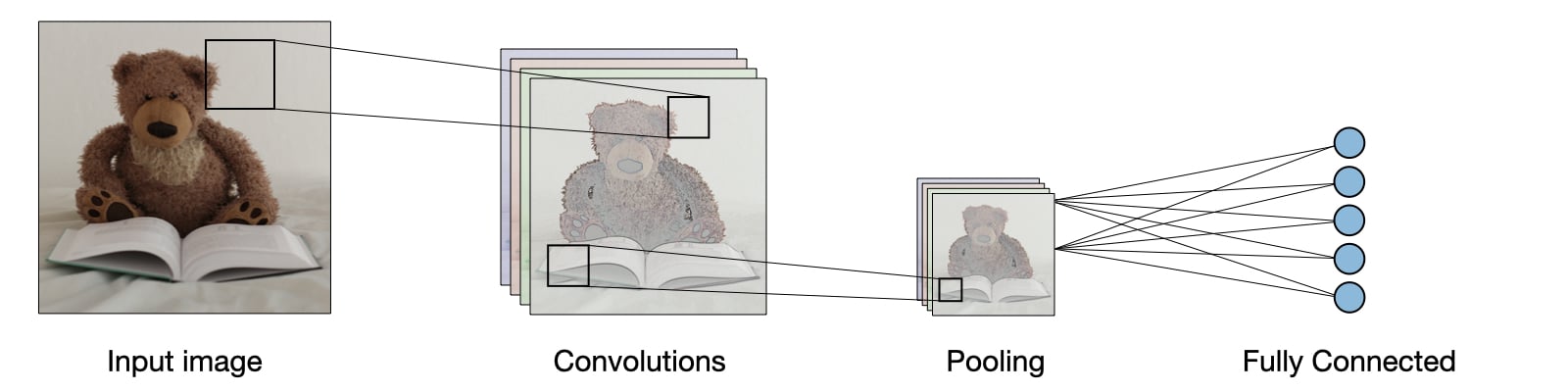
**Overview**

**Architecture of a traditional CNN**

Convolutional neural networks, also known as CNNs, are a specific type of neural networks that are generally composed of the following layers:



The convolution layer and the pooling layer can be fine-tuned with respect to hyperparameters.

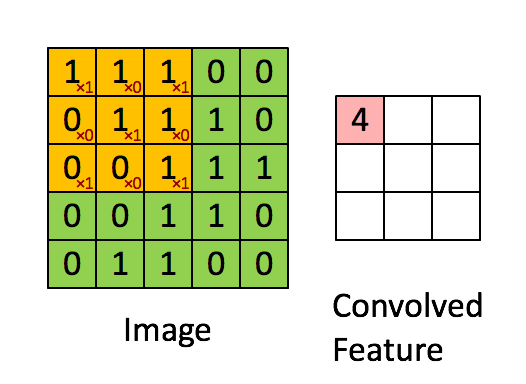
CNN as an artificial neural network that has some type of specialization for being able to pick out or detect patterns and make sense of them.

Difference from other ones, hidden layers call convolutional layer.

<https://www.youtube.com/watch?v=ZjM_XQa5s6s>

<https://www.youtube.com/watch?v=pDdP0TFzsoQ>

'CNN learns also filters as weights'



**Types of layer**

**Convolution layer (CONV)**

The convolution layer **(CONV)** uses filters that perform convolution operations as it is scanning the input I with respect to its dimensions. Its hyperparameters include the filter size $F$ and stride $S$. The resulting output $O$ is called feature map or activation map.

**Pooling (POOL)**

The pooling layer **(POOL)** is a downsampling operation, typically applied after a convolution layer, which does some spatial invariance. In particular, max and average pooling are special kinds of pooling where the maximum and average value is taken, respectively.

**Fully Connected (FC)**

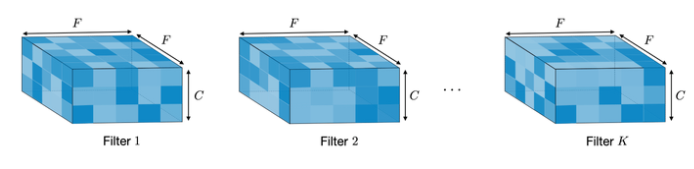
The fully connected layer **(FC)** operates on a flattened input where each input is connected to all neurons. If present, FC layers are usually found towards the end of CNN architectures and can be used to optimize objectives such as class scores.

**Filter hyperparameters**

The convolution layer contains filters for which it is important to know the meaning behind its hyperparameters

**Dimensions of a filter**

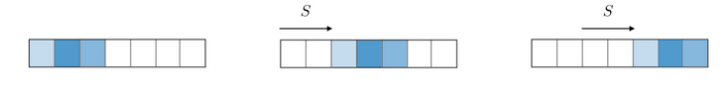
A filter of size $F×F$ applied to an input containing $C$ channels is a $F×F×C$ volume that performs convolutions on an input of size $I×I×C$ and produces an output feature map (also called activation map) of size $O×O×1$.



Remark: the application of $K$ filters of size $F×F$ results in an output feature map of size $O×O×K$.

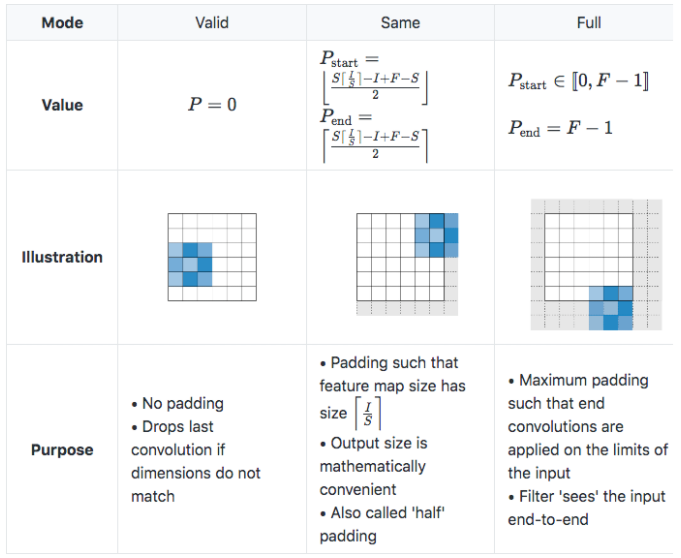
**Stride**

For a convolutional or a pooling operation, the stride SSS denotes the number of pixels by which the window moves after each operation.



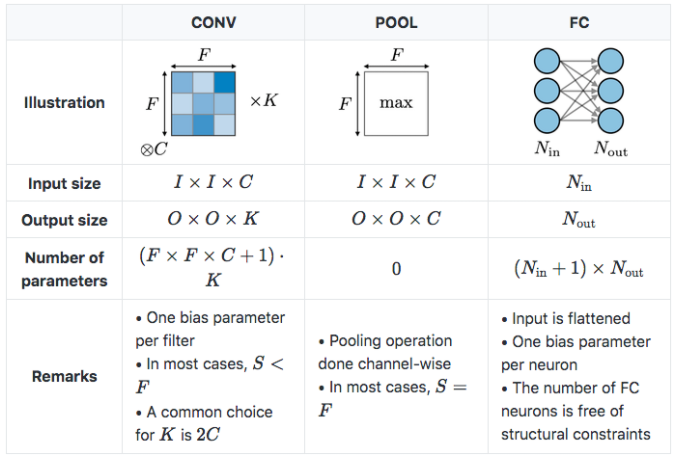
**Zero-padding**

Zero-padding denotes the process of adding PPP zeroes to each side of the boundaries of the input. This value can either be manually specified or automatically set through one of the three modes detailed below:



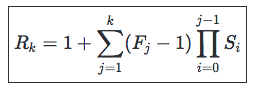
**Understanding the complexity of the model**

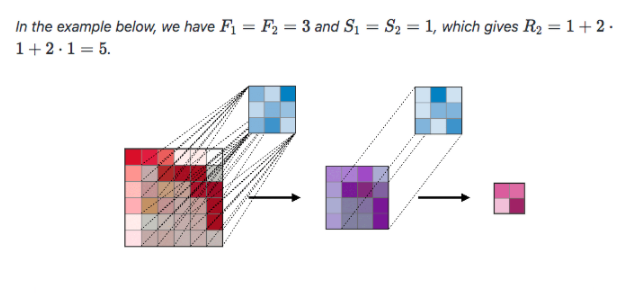
In order to assess the complexity of a model, it is often useful to determine the number of parameters that its architecture will have. In a given layer of a convolutional neural network, it is done as follows:

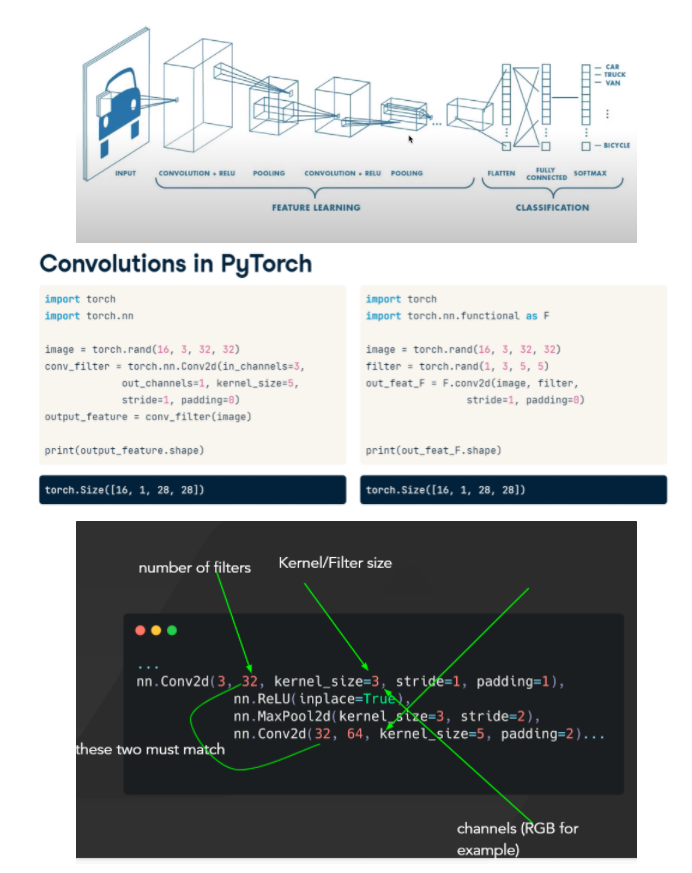


**Receptive field**

The receptive field at layer $k$ is the area denoted $Rk×Rk$ of the input that each pixel of the $k$-th activation map can 'see'. By calling $Fj$ the filter size of layer $j$ and $Si$ the stride value of layer $i$ and with the convention $S0=1$, the receptive field at layer $k$ can be computed with the formula:



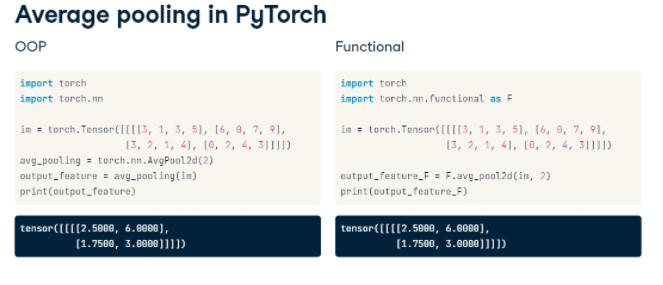




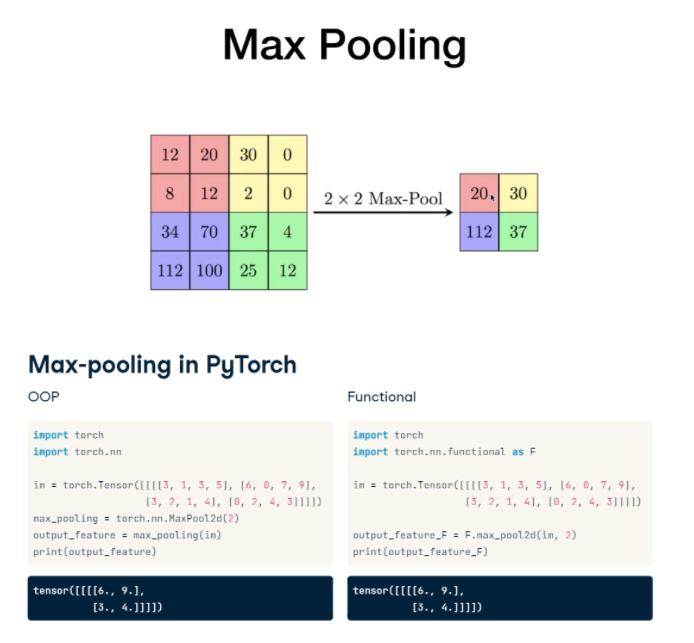
channels is the first 3!!

**Pooling**

**Average Pooling**



**Max Pooling**



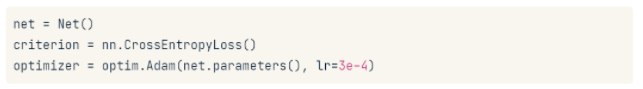
**Creating CNN**

self.fc=nn.Linear(128\*4\*4, num\_classes) → 128 is the number of units the last layer had (for depth or number of channels) 4 is for height 4 is for width. 4 comes from dividing 32 by 2 three times! for each of the pooling we apply after conv filters

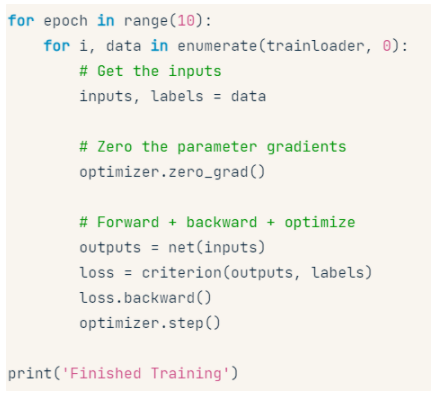


def forward → first we apply the first conv filter to the input followed by relu. we are doing this for 3 conv layer. Finally we prepare the net for the fully connected layer by squeezing all there dimensions of depth (128) , height (4) and width (4) in one dimension. then we apply to fully connected layer

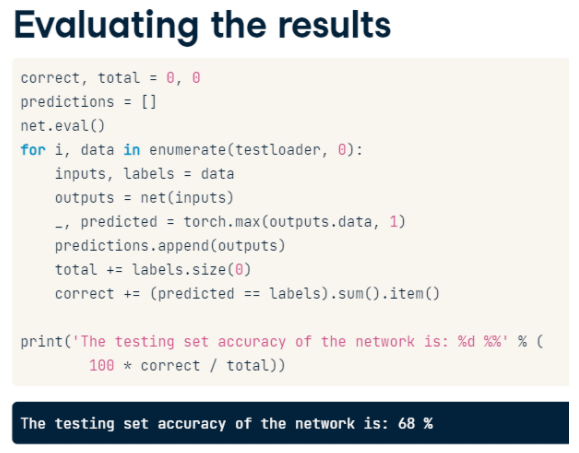
then we define the optimizer and loss function



then we create for loop!



**Evaluating**



**Dropout**

When to use it? Always!

<https://camo.githubusercontent.com/a2d68dd60749daebec06601c14c871009f9c3146/68747470733a2f2f6769746875622e636f6d2f546f6d6a6f686e736f6e656c6c69732f7374726976652d776f726b2f626c6f622f6d61696e2f646565702d6c6561726e696e672f696d672f64726f706f75742e6769663f7261773d74727565>



