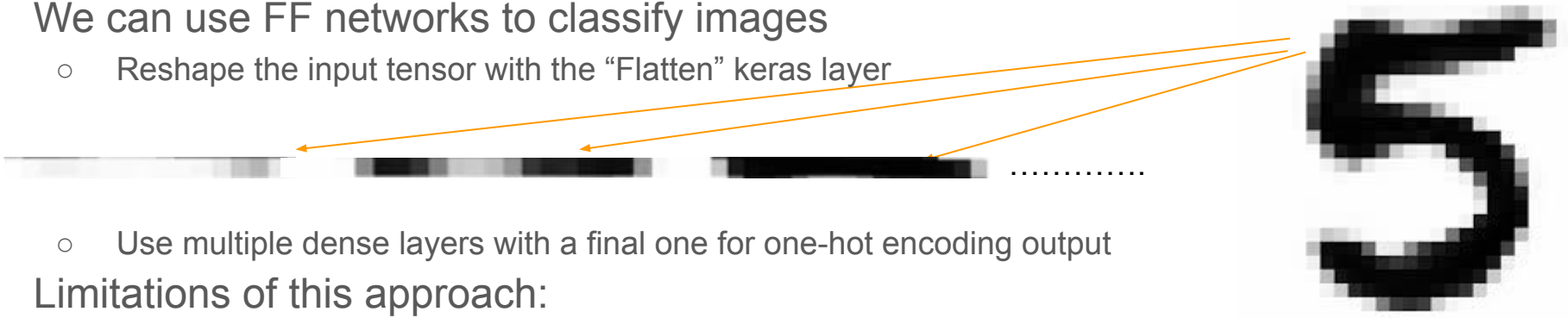
Convolutional and recurrent networks

Computing Methods for Experimental Physics and Data Analysis

Andrea.Rizzi@unipi.it

Classification of images

- Images are data structure with 2 or 3 indices: X,Y or X,Y,channel (=R,G,B)
  - Shape of the input dataset (Nsamples, Width, Height, nchannels)
  - nchannels is typically 1 (B&W), 3 (RGB) or 4 with transparency
- We can use FF networks to classify images
  - Reshape the input tensor with the “Flatten” keras layer



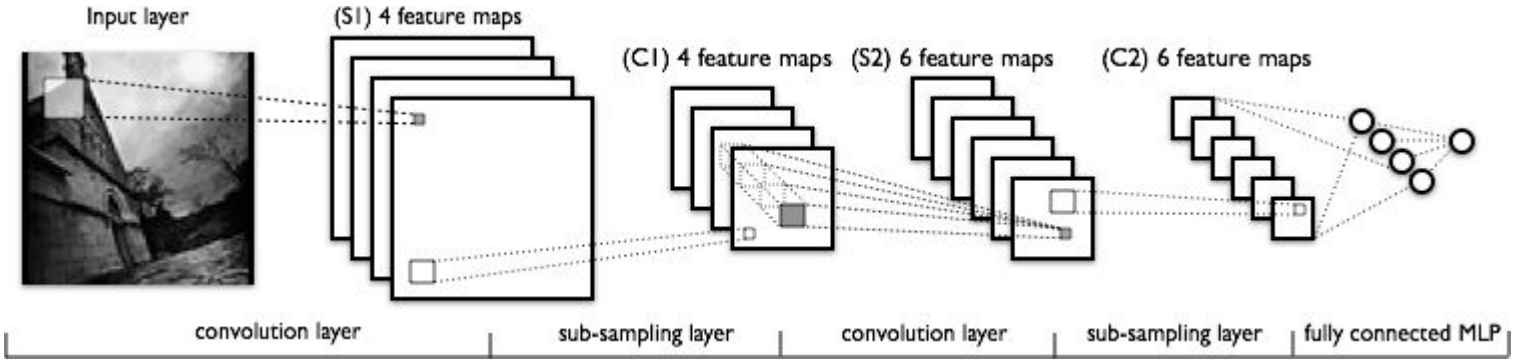
  - Use multiple dense layers with a final one for one-hot encoding output
- Limitations of this approach:
  - If the image is translated, even by a single pixel in x or y, the network may not recognize as “similar” to the untranslated image
  - Nearby pixels in “Y” (or even the same pixel but in a different color) are not treated any differently than far away pixels
- We know that our problem has some invariance
- We know that input data has some locality information

Exploit invariance and locality

- Suppose you want to count windows in a 800x600 picture with houses
  - With an MLP or DFF you have 800x600x3(RGB)=1.4M inputs
  - Each node process independently some part of the image
  - The initial “Dense” connection should converge to something with lot of “zero” weights because far away pixel points have no reason to be considered at the same time in order to detect **local** features
  - => the problem cannot be managed this way
- But the problem is translation invariant!
  - “Windows” are local features, you can just analyze a patch of the image **(locality)**
  - A window is a window no matter if it is top left or bottom right of your image **(Invariance)**
  - And actually windows are made of even more local features (some borders/frame, some uniform area, a squared shape)

Can we exploit problem invariance?

- Convolutional neural networks (CNN) attempt to exploit invariance against spatial translations
  - Smaller networks (locality !)
  - Acting on a single patch of the image
  - Stacking multiple such Convolutional Layers one after the other
  - Use “subsampling” layer to scale from local to global
- Hierarchical approach
  - Early layers learn local features
  - Subsampling reduce the information extracted from a given “patch”
  - A final flatten+one or more dense layers is used to reach the final target

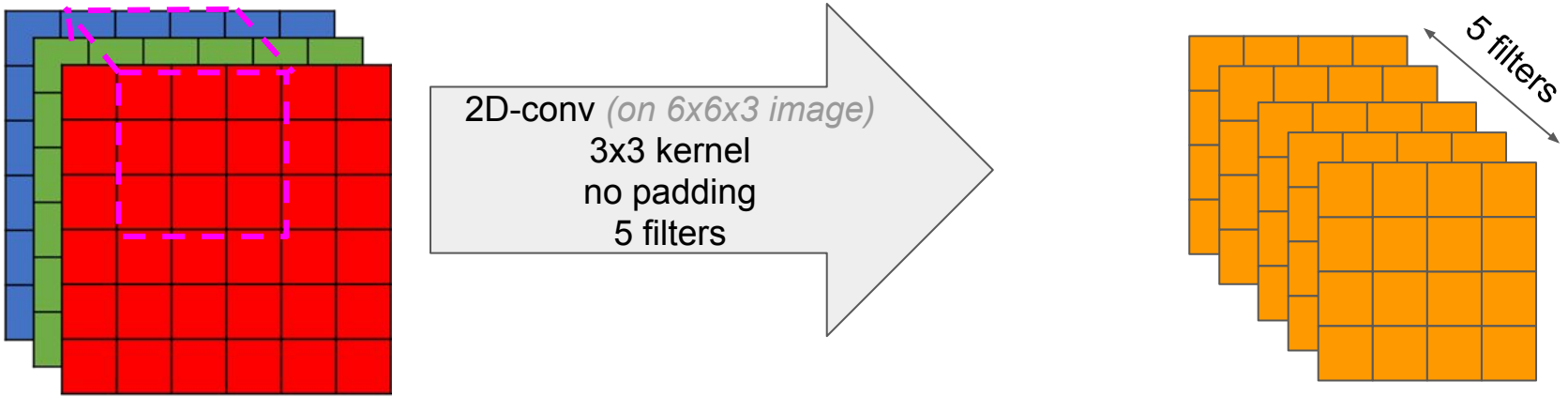


Limitations

- The linear algebra formalism we use can handle nicely images, hence implement nicely CNN (translation invariance along x and y)
- There are more invariances out there!
  - Rotation
  - Scale
  - Luminosity
  - … you name it…
- So currently the networks have to learn them all
  - We can do tricks to increase the number of samples in our datasets with augmentation techniques (i.e. apply random transformations of scale, rotation etc..)
  - “Built-in” invariance (such as the x-y one) has the advantage of reducing by orders of magnitude the number of weights to learn

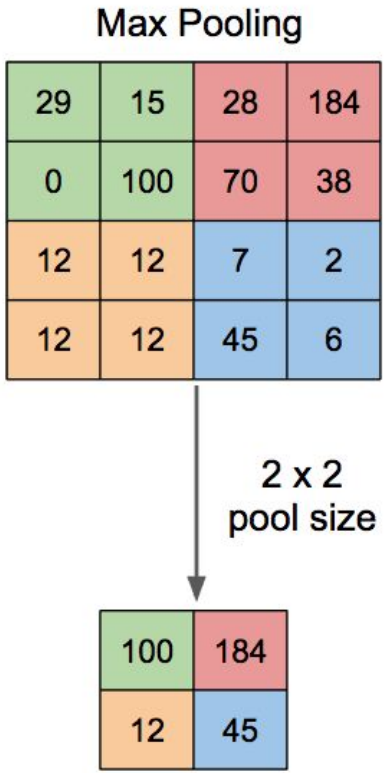
Understanding the dimensions of the convolution

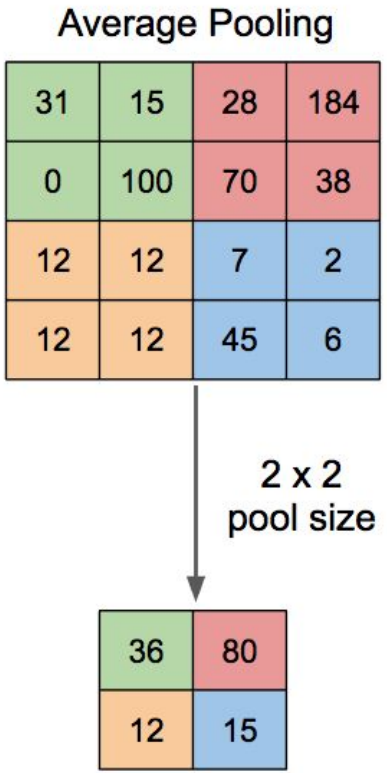
- Convolution can be 1D, 2D, 3D
- Kernel size, typically square (MxM) with M odd (but can be any shape)
- *Padding*: how to we handle borders? We can do only “valid” windows (no padding) or process borders as if there were zeros (or other values) outside
- Each “point” in the 1D, 2D, 3D matrix can have multiple features (e.g. R,G,B)
- Each Convolutional layer have mutiple outputs (filters) for every “patch” it scans on (one optimized to detect if the patch is uniformly filled, one looking for vertical lines, etc..)



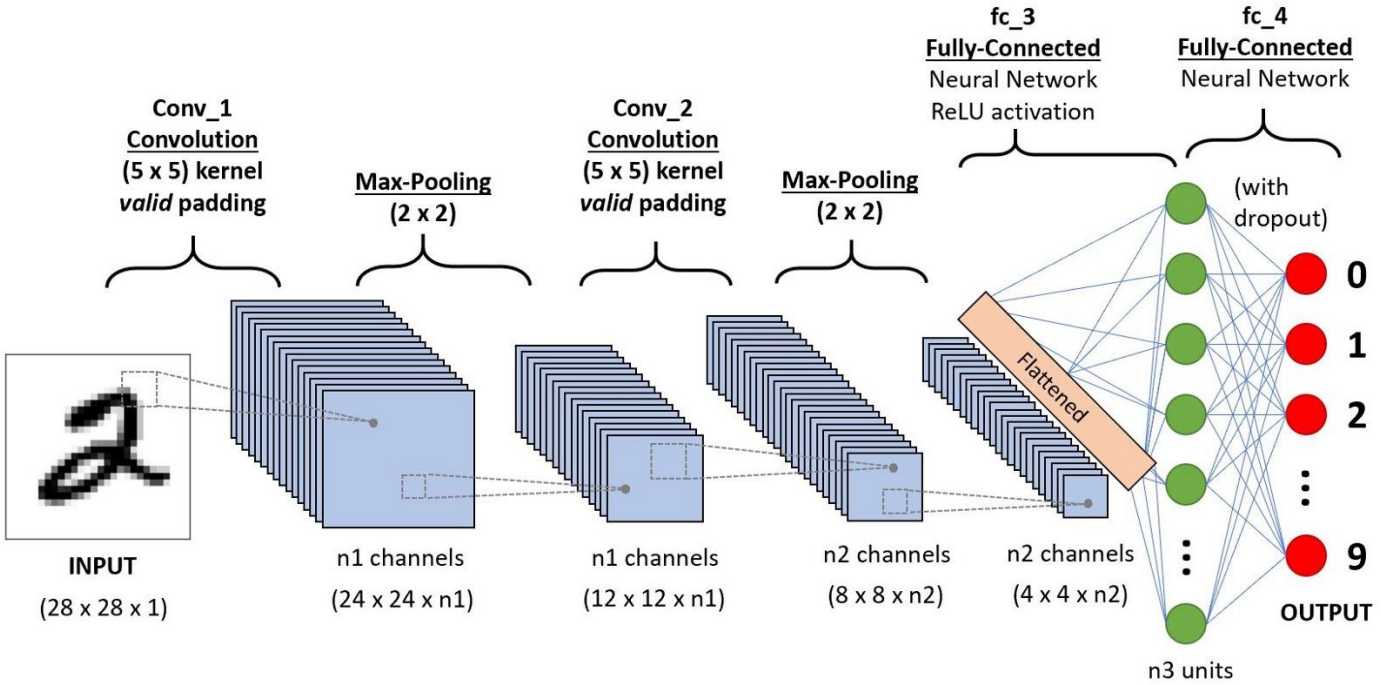
Pooling (subsampling)

- Pooling layers are simply finding maxima or computing average in patches of some convolution layer output
- Pooling is used to reduce the space dimensionality after a convolutional layer
  - The Conv “filters” look for features (e.g. a filter may look for cats eyes)
  - The Pooling layer checks if in a given region some filtered fired (there was a cat eye somewhere in this broader region)



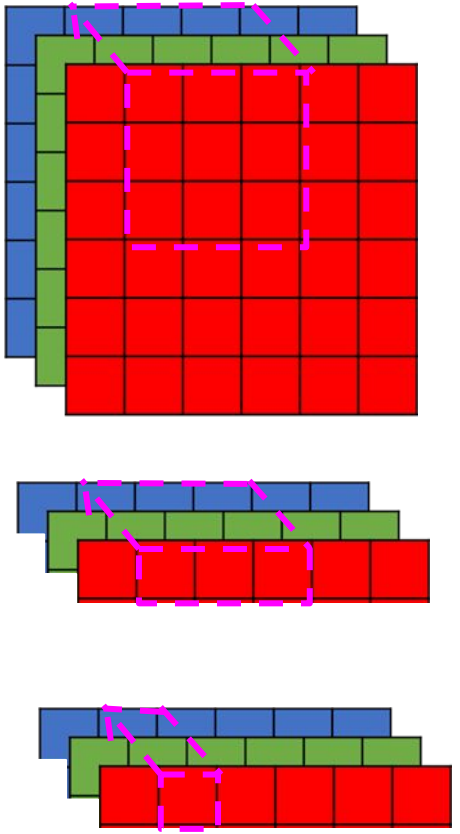


Typical CNN architecture



More on convolution

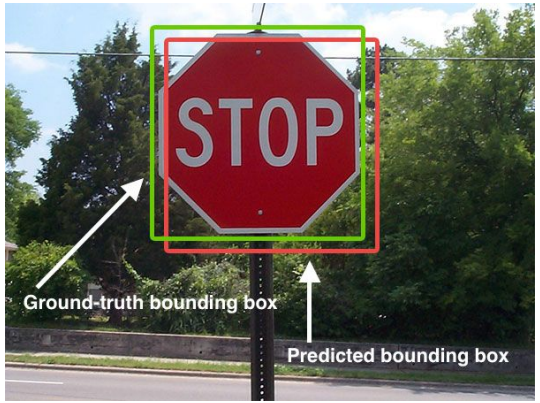
- Convolution is a way to correlate **local** input information and to reduce the NN size by sharing the weights of the nodes across all repeated patches.
- What if I have multiple objects, with no local correlation, but with multiple features (like R,G,B channels) and I want to process them all in the same way?
  - 1x1 convolution!
  - Conv1D is usually enough (as the x-y coordinates have no meaning here)
  - The symmetry here is that all “objects” are the same
- Example :
  - Particles in a detector with information about 4-vector, tracking hits, calorimeter deposits, p-ID etc… and want to preprocess them one by one before using them for some higher level task



Bounding Box

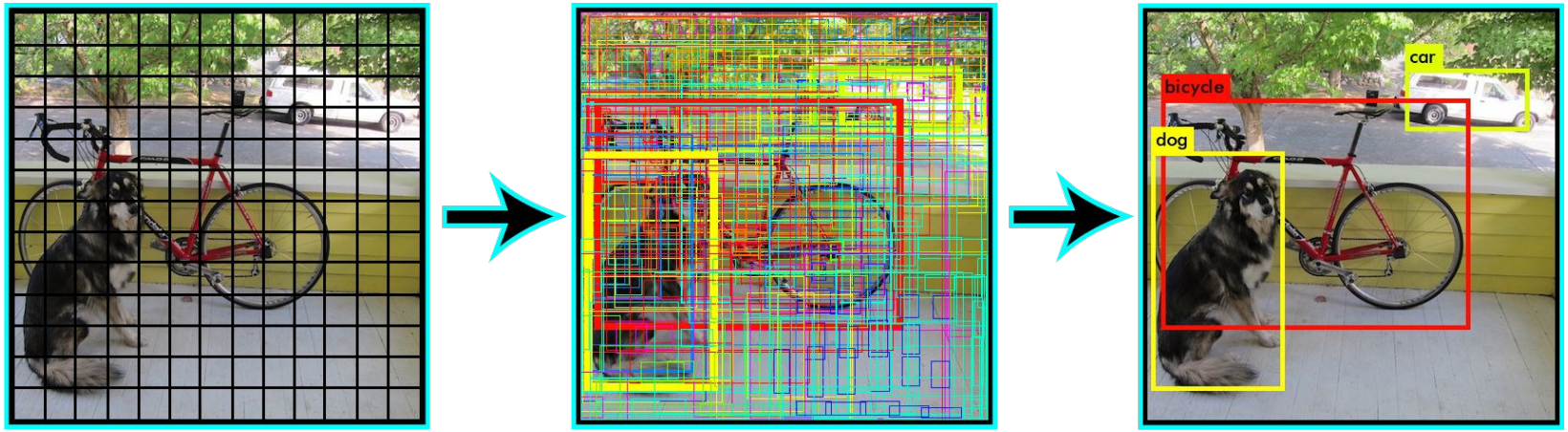
In order to predict “where” an object is a “bounding box” is defined

- Coordinates of two opposite corners
- Essentially a “regression” problem



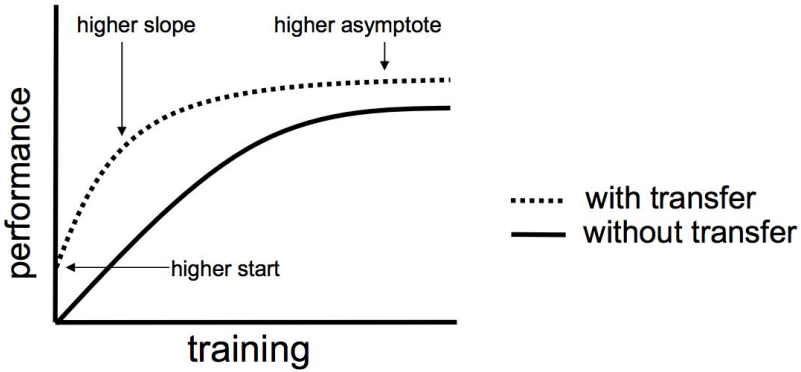
Not simple to extend to multiple objects in a single image, YOLO (You Only Look Once) algorithm is an option https://pjreddie.com/darknet/yolo/

- Divide the image in cells, in each cell you predict up to N bounding box corners (relative to the cell position)
- Pick only cells with high score (and cluster multiple predictions of the same bb)



Transfer learning

- If learn to process images of a given size, can we apply that to different size
  - If the “scale” is the same, the convolutional part can work unchanged
  - The dense (when present) anyhow need to be adapted/retrained
- Transfer learning is a technique to reuse a network training for a task to perform **another task** with reduced retraining
  - E.g. a Conv2D network meant for image processing have initial layers processing “local features”... that is not very domain specific (if you trained on flowers images it may work on animals too)
  - Very useful when the available sample of the proper domain is small
    - E.g. annotated medical images are harder to get than labelled real world pictures



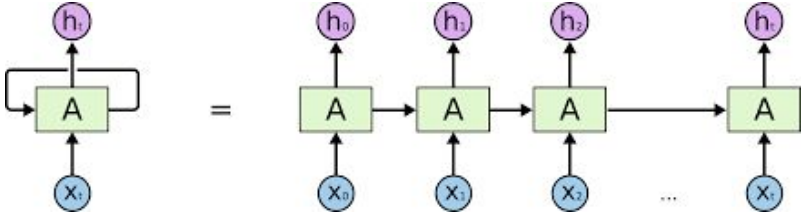
Variable length, sequences and causality

- What if the input size has a variable length? For example
  - Text translations
  - Identification of “jets” of particle in High Energy Physics
- In many case sequences have still a concept of locality and translation invariance
  - “A cat” or “the cat” are two sentences, both containing “cat” but in different position
- Sequences often also have implied ordering
  - “The cat eat a mouse” and “The mouse eat a cat” have different meanings

Exploiting time invariance

- Some problems are “time invariant”
  - E.g. recognize words in a sentence (written or spoken)
- Order matters and some causality is implied in the sequence
- Length of the inputs or the output may not be fixed

Recurrent Networks (RNN)

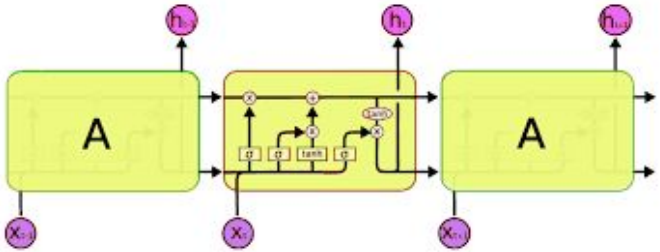


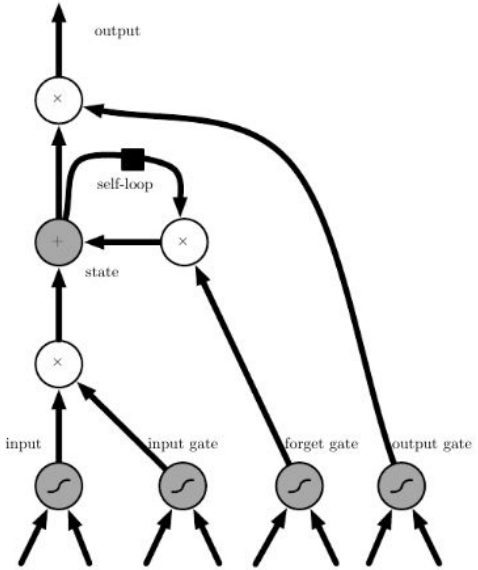
- Iterative networks with output passed again as input
  - Allow some “memory” of the previous inputs and/or some internal “state” of what the network understood so far in the sequence
- Most commonly used RNN are LSTM (Long Short Term Memory) and GRU (Gated Recurrent Unit)

LSTM and GRU

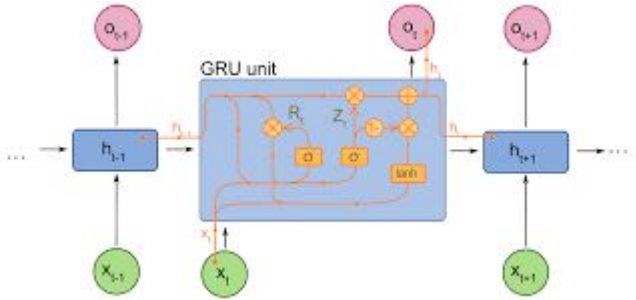
- LSTM and GRU are RNN units with additional features to *control* their “memory”
- “Gates” allow to control (keep or drop) *input*, *output* and *internal state*
- The advantage of gated units is that they *can forget* so that when processing a sequence they focus on the relevant part (e.g. when processing a text we may know that each time we encounter a space the word is over)

LSTM



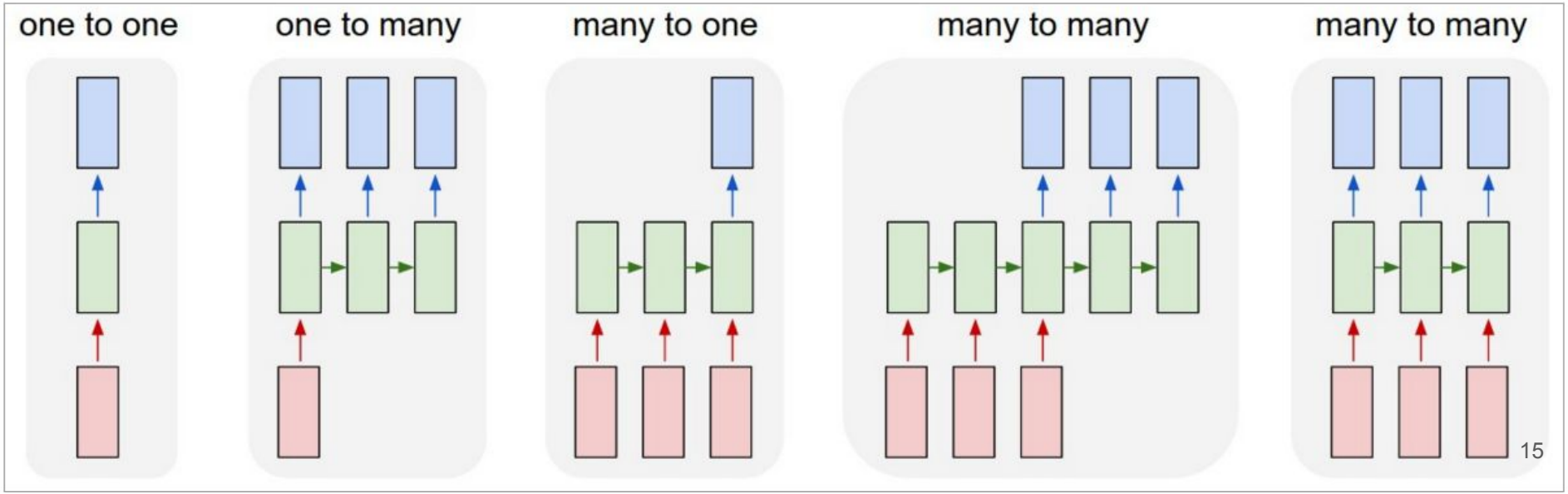


GRU



Different ways of processing time series

- Recurrent Networks can be used to implement networks with variable number of inputs and outputs
  - Encoding, Decoding, Sequence2Sequence



Keras basic layers

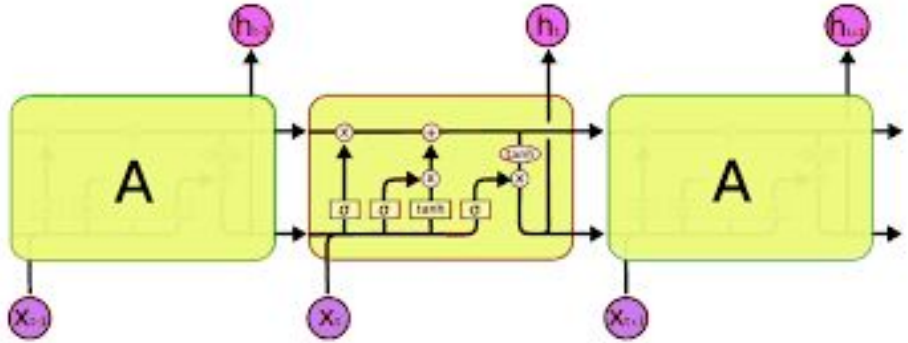
- Convolutional layers
  - Flatten
  - Conv1D/2D/3D
  - ConvTranspose or “Deconvolution”
  - UpSampling and ZeroPadding
  - MaxPooling, AveragePooling
  - Flatten
- Recurrent layers
  - LSTM
  - GRU
  - SimpleRNN
  - TimeDistributed
  - ConvLSTM2D

channels_first vs channels_last
Clarifies which indices are part of the convolution and which indices are the “channels”

(#sample, X, Y, channels) **<- default**
VS
(#sample, channels, X, Y)

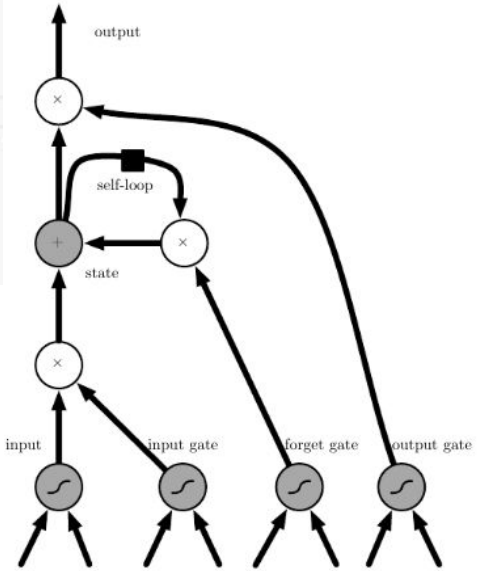
More on LSTM

- LSTM layers in keras can return
  - Just the output of the last iteration
  - The whole sequence of output
  - The gated output of the memory
  - The cell state

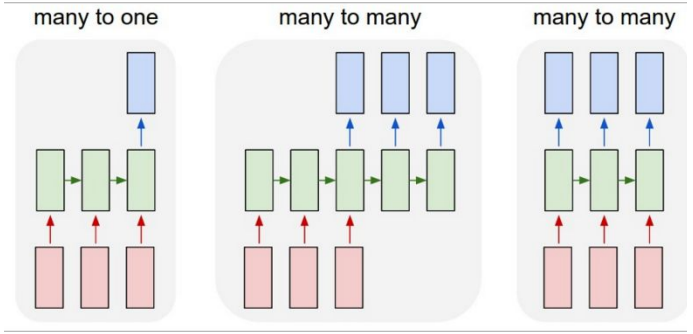



```
from keras.models import Model
from keras.layers import LSTM
from numpy import array
inputs1 = Input(shape=(5, 1))
lstm1, state_h, state_c = LSTM(1, return_state=True, return_sequences=True)(inputs1)
model = Model(inputs=inputs1, outputs=[lstm1, state_h, state_c])
data = array([0.1, 0.2, 0.3, 0.4, 0.5]).reshape((1,5,1))
print(model.predict(data))
```

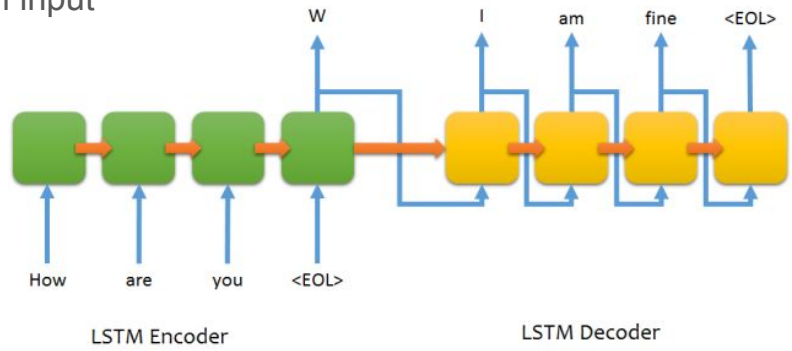
```
[array([[[0.02106816],
         [0.05576485],
         [0.09626514],
         [0.13520567],
         [0.16713278]]], dtype=float32),
 array([[0.16713278]], dtype=float32),
 array([[0.41894686]], dtype=float32)]
```



Using LSTM



- Many to one configuration:
  - Just use a LSTM layer with default config
    - No need to know the full sequence
    - Optionally request also the cell state
- Many to Many (synchronous)
  - Set return_sequence=True to get exactly one output for each input
- Many to many (async, different length)
  - Need two LSTM: A encoder + a decoder
  - *Sequence2Sequence* or *Encode-Decode* architecture
  - The cell state of the encoder can be used as initial state for the decoder
  - Need to define a STOP character to receive when the decoding sequence is over

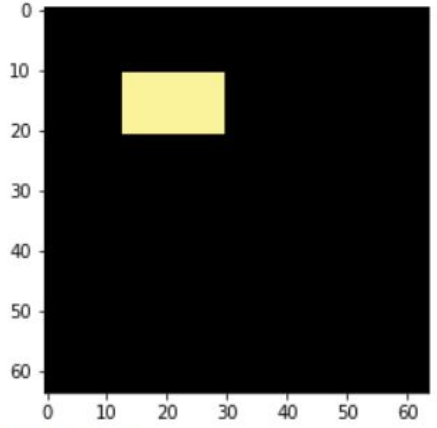


- Inputs with variable length should be “padded”
  - Masking layers exist in keras to avoid “learning from padding”
  - Reversing the sentence order (so that padding is at the beginning also helps)
  - Often with LSTM useful to provide most important information at the end

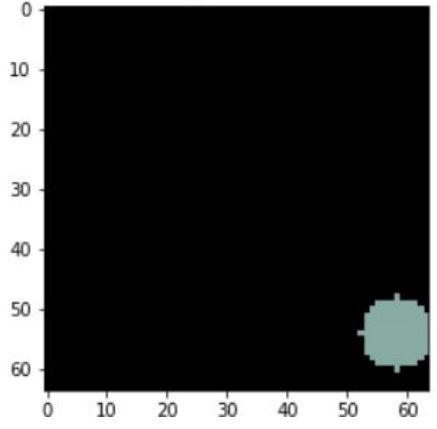
Assignment 3

Create a CNN that recognize squares and circles in an image. Let's try three variations:

1. Classify: does it contain a rectangle or a circle?
2. Count circles and rectangles when there is more than one in the dataset
3. Find the position (bounding box) of the circle or rectangle




```
(50000, 4)
[[13 11 29 20]
 [52 48 64 60]]
```



https://colab.research.google.com/drive/1kRP1NfbL3hj9xIHAnfMEx9ug76ozGeqR

Solution

Assignment 4

Try building from scratch a LSTM that find the maximum length and its position in a sequence of two dimensional vectors.

- Generate some data
- Build a network with one LSTM layer followed by a Dense one

Solution