Friedrich-Alexander-Universität
Technische Fakultät

Machine Learning
Data Analytics

FAU

Machine Learning for Time Series

(MLTS or MLTS-Deluxe Lectures)

Dr. Dario Zanca

Machine Learning and Data Analytics (MaD) Lab
Friedrich-Alexander-Universität Erlangen-Nürnberg
18.01.2024

Topics overview

- Time series fundamentals and definitions (2 lectures)
- Bayesian Inference (1 lecture)
- Gussian processes (2 lectures)
- State space models (2 lectures)
- Autoregressive models (1 lecture)
- Data mining on time series (1 lecture)
- Deep learning on time series (4 lectures) ←
- Domain adaptation (1 lecture)

In this lecture…

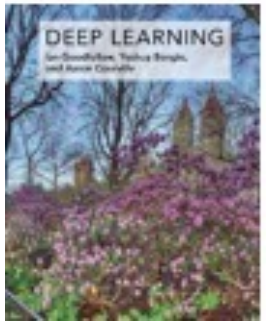
1. Convolutional neural networks

2. Convolutional-based architectures

References

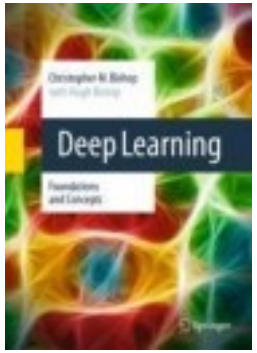
Deep Learning

by Ian Goodfellow, Yoshua Bengio, and Aaron Courville (2016)



Deep Learning: Foundations and Concepts

by C. Bishop, H. Bishop (2024)



Deep Learning for Time Series – Convolutional Models

Convolutional Neural Networks (CNNs)

Motivation

RNN / LSTM limitations:

- Non-parallelism → Long training time
- Difficulties with long sequences
  - Large memory usage
  - Difficult to train (vanishing/exploding gradients)
  - Hard to learn long-term dependencies (mitigated by LSTMs)

Some of this problems are attenuated on CNNs:

- Parallel computations
- Easy to use on large input data

Human visual cortex

CNNs were inspired by the internal functioning of a specialized brain area: the visual cortex.

The visual cortex is in charge of processing visual information collected by the retinae.

Cortical neurons only respond to a small portion of the stimuli (small receptive field).

Small receptive fields are stimulated by high spatial frequencies (fine details); large receptive fields are stimulated by low spatial frequencies (coarse details).

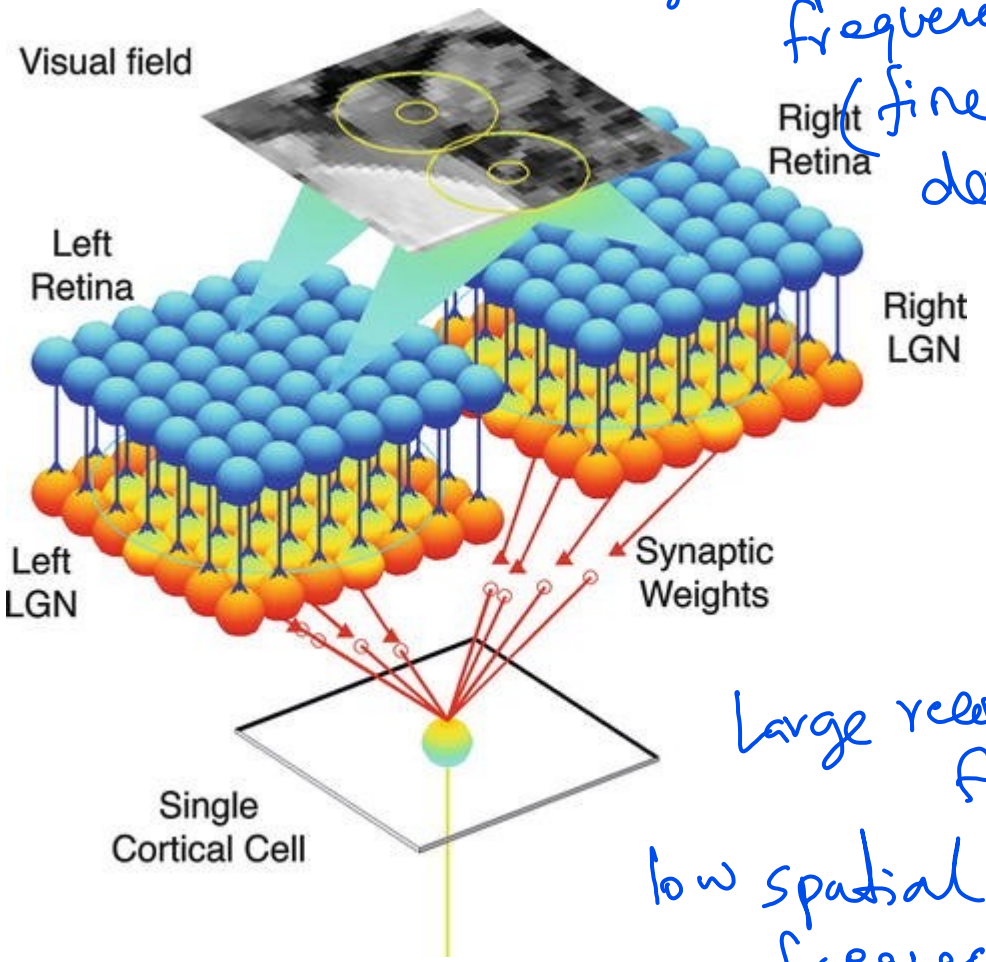
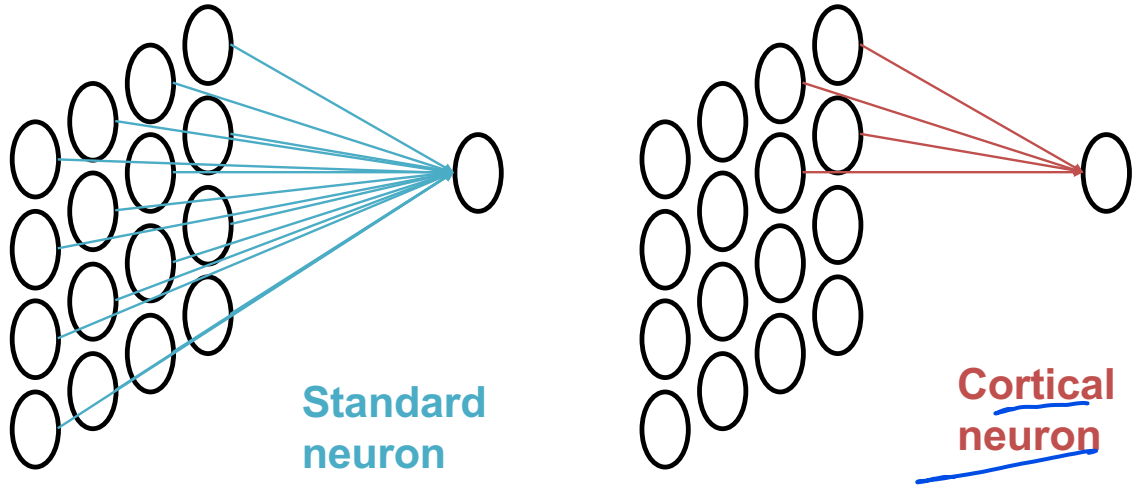


Image from: https://link.springer.com/referenceworkentry/10.1007/978-1-4614-7320-6_360-2

Modeling small receptive fields

- In standard multilayer perceptrons, neurons are connected to all units from the previous layer.
- Cortical neurons have small receptive fields: they have sparse and localized connection with units from the previous layer.
  → This is modeled by means of convolutional operations.



(sparse & localized) connections.

Convolutional operation

The (discrete) convolution is a mathematical operation on two functions (in our case, the input and a smaller filter) that produces a third function (the feature map).

i/p Smaller filter. Feature map.

$$(input * filter)[n] = \sum_{m=-\infty}^{+\infty} input[m] \cdot filter[n-m]$$

filter related 'n'

- The sum is evaluated for all values of the shift.
- The term convolution is used both to indicate the result function and process of computing it.

"result & process"

Convolutional operation (2D)

The output computation now only depends on a subset of inputs:

$$y_{11} = w_{11}x_{11} + w_{12}x_{12} + w_{13}x_{13} + w_{21}x_{21} + w_{22}x_{22} + w_{23}x_{23} + w_{31}x_{31} + w_{32}x_{32} + w_{33}x_{33}$$

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \\ x_{31} & x_{32} & x_{33} & x_{34} & x_{35} \\ x_{41} & x_{42} & x_{43} & x_{44} & x_{45} \\ x_{51} & x_{52} & x_{53} & x_{54} & x_{55} \end{bmatrix} * \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} = \begin{bmatrix} y_{11} & & \\ & & \\ & & \end{bmatrix}$$

Convolutional operation (2D)

The output computation now only depends on a subset of inputs:

$$y_{12} = w_{11}x_{12} + w_{12}x_{13} + w_{13}x_{14} + w_{21}x_{22} + w_{22}x_{23} + w_{23}x_{24} + w_{31}x_{32} + w_{32}x_{33} + w_{33}x_{34}$$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

$*$

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

$=$

y_{11}	y_{12}	

Convolutional operation (2D)

The output computation now only depends on a subset of inputs:

$$y_{13} = w_{11}x_{13} + w_{12}x_{14} + w_{13}x_{15} + w_{21}x_{23} + w_{22}x_{24} + w_{23}x_{25} + w_{31}x_{33} + w_{32}x_{34} + w_{33}x_{35}$$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

$*$

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

$=$

y_{11}	y_{12}	y_{13}

Convolutional operation (2D)

The output computation now only depends on a subset of inputs:

$$y_{23} = w_{11}x_{23} + w_{12}x_{24} + w_{13}x_{25} + w_{21}x_{33} + w_{22}x_{34} + w_{23}x_{35} + w_{31}x_{43} + w_{32}x_{44} + w_{33}x_{45}$$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

$*$

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

$=$

y_{11}	y_{12}	y_{13}
		y_{23}

Convolutional operation (2D)

The output computation now only depends on a subset of inputs:

$$y_{22} = w_{11}x_{22} + w_{12}x_{23} + w_{13}x_{24} + w_{21}x_{32} + w_{22}x_{33} + w_{23}x_{34} + w_{31}x_{42} + w_{32}x_{43} + w_{33}x_{44}$$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

$*$

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

$=$

y_{11}	y_{12}	y_{13}
	y_{22}	y_{23}

Convolutional operation (2D)

The output computation now only depends on a subset of inputs:

$$y_{21} = w_{11}x_{21} + w_{12}x_{22} + w_{13}x_{23} + w_{21}x_{31} + w_{22}x_{32} + w_{23}x_{33} + w_{31}x_{41} + w_{32}x_{42} + w_{33}x_{43}$$

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \\ x_{31} & x_{32} & x_{33} & x_{34} & x_{35} \\ x_{41} & x_{42} & x_{43} & x_{44} & x_{45} \\ x_{51} & x_{52} & x_{53} & x_{54} & x_{55} \end{bmatrix} * \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} = \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ & & \end{bmatrix}$$

Convolutional operation (2D)

The output computation now only depends on a subset of inputs:

$$y_{31} = w_{11}x_{31} + w_{12}x_{32} + w_{13}x_{33} + w_{21}x_{41} + w_{22}x_{42} + w_{23}x_{43} + w_{31}x_{51} + w_{32}x_{52} + w_{33}x_{53}$$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

$*$

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

$=$

y_{11}	y_{12}	y_{13}
y_{21}	y_{22}	y_{23}
y_{31}		

Convolutional operation (2D)

The output computation now only depends on a subset of inputs:

$$y_{32} = w_{11}x_{32} + w_{12}x_{33} + w_{13}x_{34} + w_{21}x_{42} + w_{22}x_{43} + w_{23}x_{44} + w_{31}x_{52} + w_{32}x_{53} + w_{33}x_{54}$$

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

$*$

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

$=$

y_{11}	y_{12}	y_{13}
y_{21}	y_{22}	y_{23}
y_{31}	y_{32}	

Convolutional operation (2D)

The output computation now only depends on a subset of inputs:

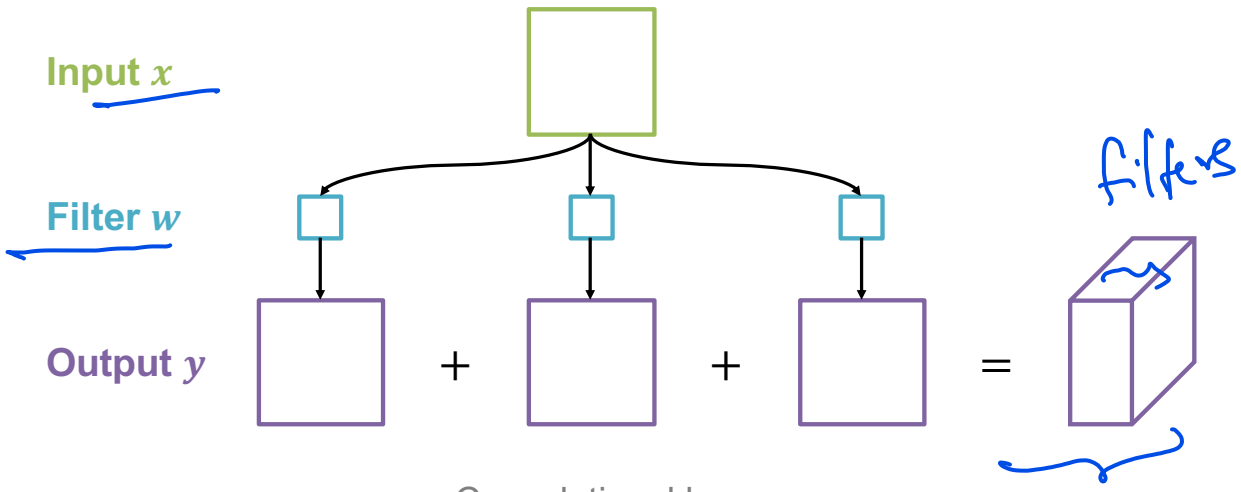
$$y_{33} = w_{11}x_{33} + w_{12}x_{34} + w_{13}x_{35} + w_{21}x_{43} + w_{22}x_{44} + w_{23}x_{45} + w_{31}x_{53} + w_{32}x_{54} + w_{33}x_{55}$$

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \\ x_{31} & x_{32} & x_{33} & x_{34} & x_{35} \\ x_{41} & x_{42} & x_{43} & x_{44} & x_{45} \\ x_{51} & x_{52} & x_{53} & x_{54} & x_{55} \end{bmatrix} * \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} = \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ y_{31} & y_{32} & y_{33} \end{bmatrix}$$

Convolutional layer

In a convolutional layer, multiple filters can be used in parallel, which result in multiple convolutional operations in parallel on the same input.

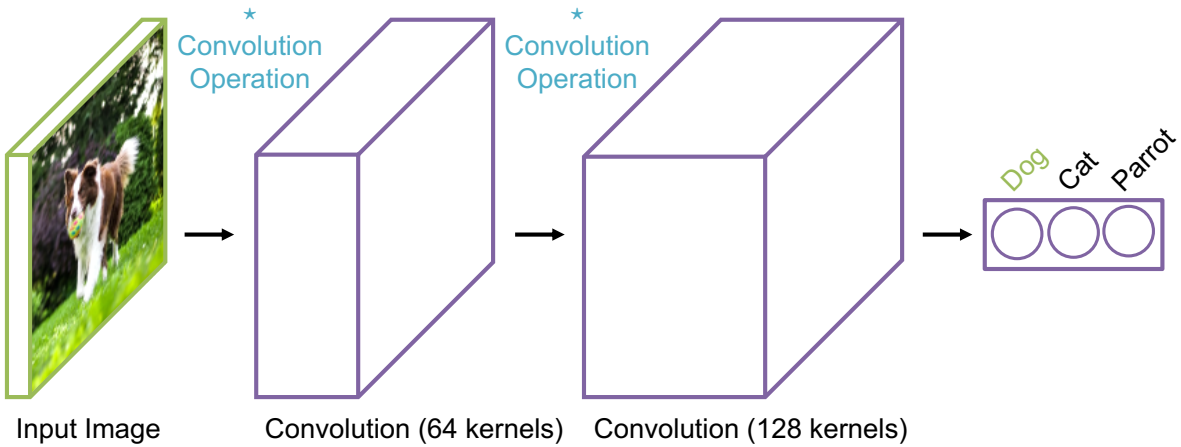
The different output are concatenated to create a multi-channel feature map.



Convolutional Neural Network (CNN)

To incrementally process relevant features, multiple layers of convolution can be stacked to create a deep convolutional neural network (CNN).

However, after many layer the number of features grow very quickly
→ Computationally expensive



Pooling operation

reduce # computations.

The pooling operation is introduced to reduce the number of computations in a CNN.
From a theoretical perspective, higher representations do not require high spatial resolution.

Higher representations → No need of high spatial resolution.

$$y_{11} = \max(x_{11}, x_{12}, x_{21}, x_{22})$$

x_{11}	x_{12}	x_{13}	x_{14}
x_{21}	x_{22}	x_{23}	x_{24}
x_{31}	x_{32}	x_{33}	x_{34}
x_{41}	x_{42}	x_{43}	x_{44}

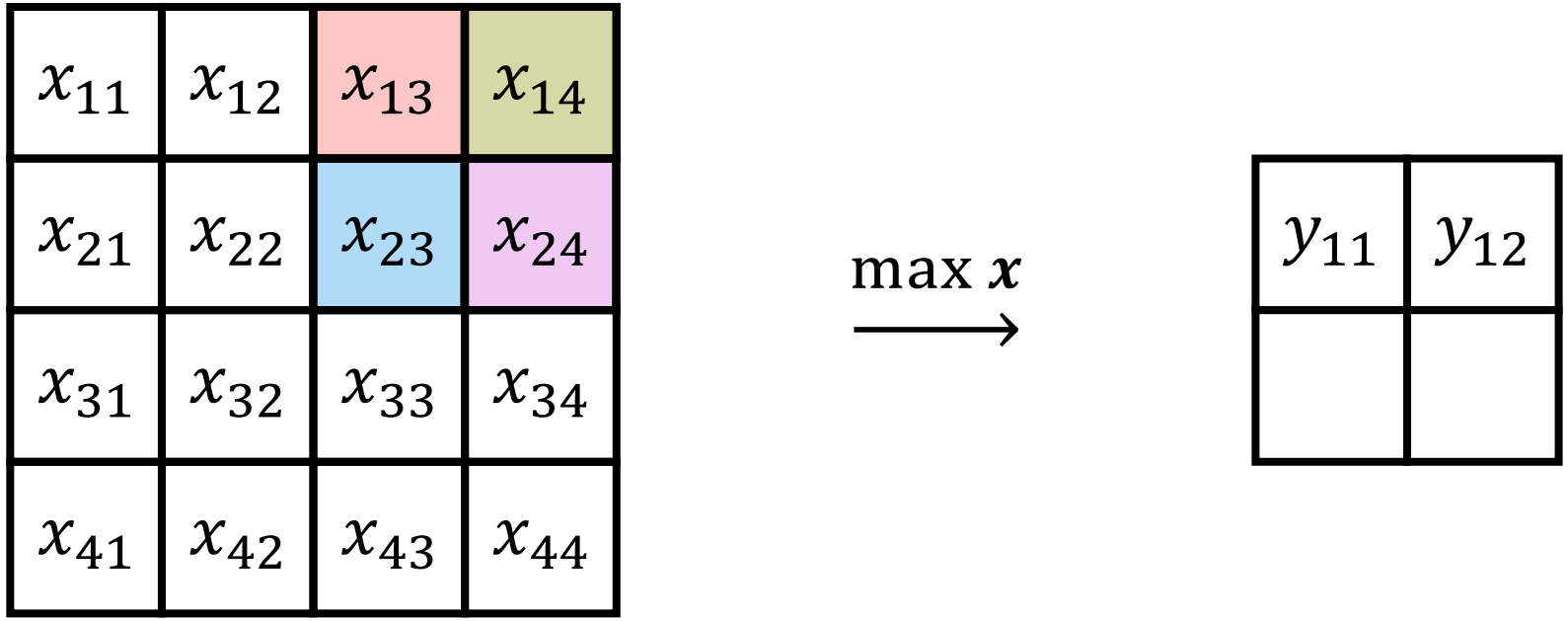
$\max \boldsymbol{x} \longrightarrow$

y_{11}	

Pooling operation

The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.

$$y_{12} = \max(x_{13}, x_{14}, x_{23}, x_{24})$$



Pooling operation

The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.

$$y_{21} = \max(x_{31}, x_{32}, x_{41}, x_{42})$$

x_{11}	x_{12}	x_{13}	x_{14}
x_{21}	x_{22}	x_{23}	x_{24}
x_{31}	x_{32}	x_{33}	x_{34}
x_{41}	x_{42}	x_{43}	x_{44}

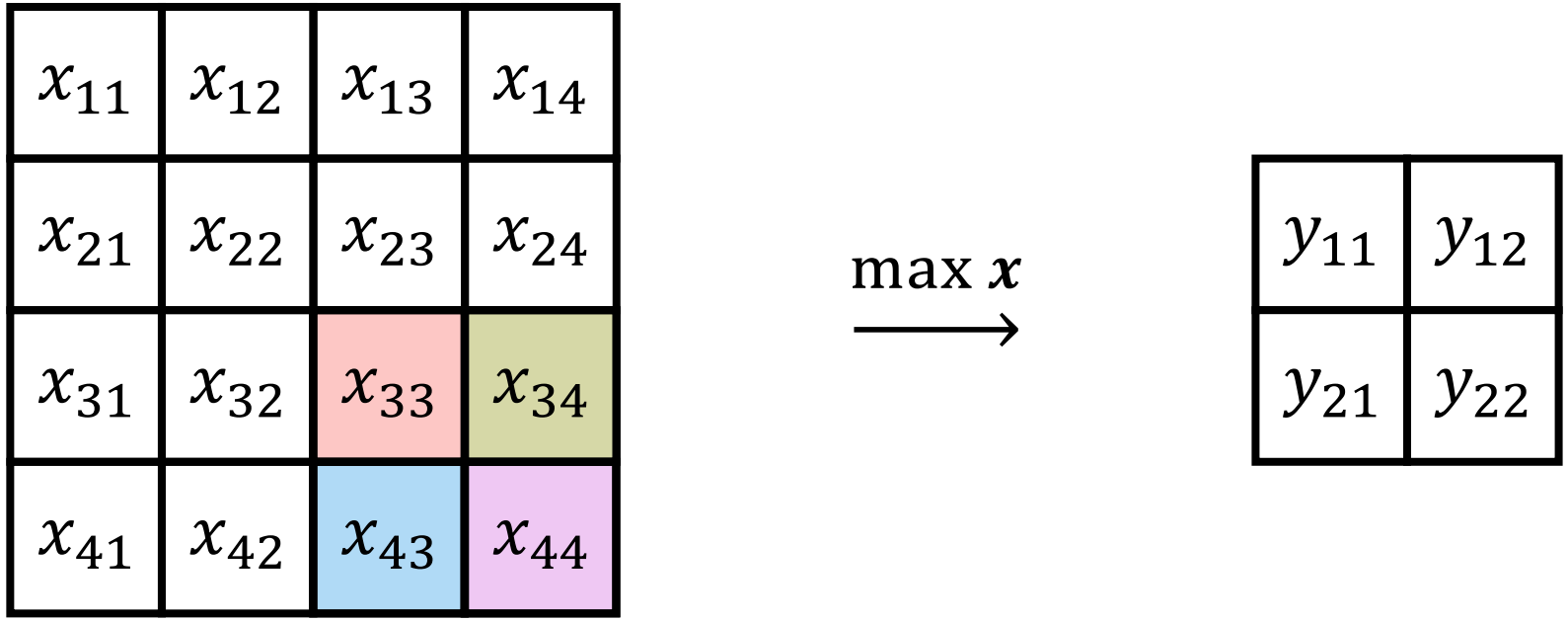
$$\xrightarrow{\max \boldsymbol{x}}$$

y_{11}	y_{12}
y_{21}	

Pooling operation

The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.

$$y_{22} = \max(x_{33}, x_{34}, x_{43}, x_{44})$$



Pooling operation

The pooling operation is introduced to reduce the number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution.

- Other operations can be used instead of the max, e.g., the mean, L^2-norm, …
- There is no general consensus on which of these operation is the best and this should be experimentally validated case by case.

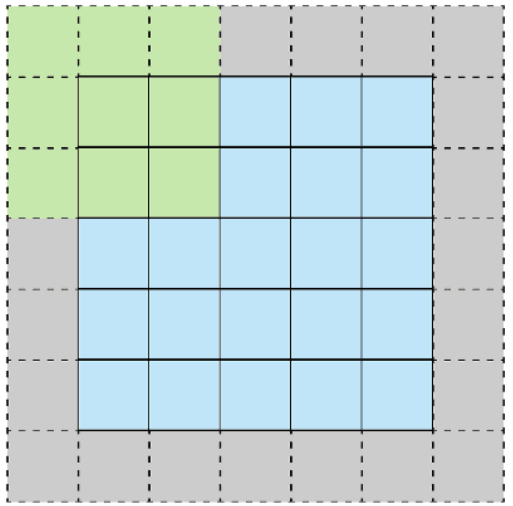
→ No fixed method
→ experimentally validate case by case.

Padding operation

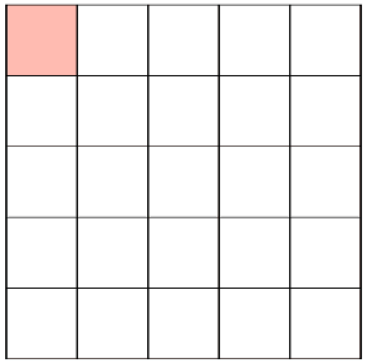
The convolution, as illustrated previously, leads to a reduction of the dimensionality.

→ Given an input of size n and a filter of size m, the resulting feature map have size $n - m + 1$.

For this reason, the padding operation is used to add symmetrically zeros to the input, such that the resulting feature map is of the same size as the input.



Stride 1 with Padding



Feature Map

https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529

1-D Convolutional Neural Networks

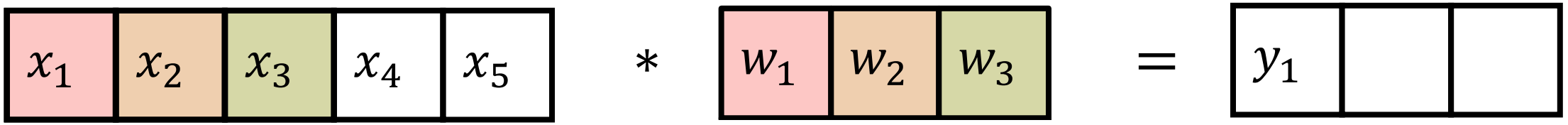
CNNs can be used to learn temporal dependencies on time series data.

- The convolution is applied along a single dimension, i.e., the temporal dimension.
- The resulting model is generally referred to as 1-D CNN.

Convolutional operation (1D)

The output computation now only depends on a subset of the time series:

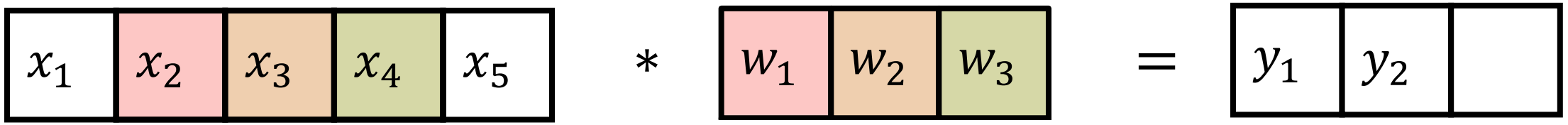
$$y_1 = w_1 x_1 + w_2 x_2 + w_3 x_3$$



Convolutional operation (2D)

The output computation now only depends on a subset of inputs:

$$y_2 = w_1 x_2 + w_2 x_3 + w_3 x_4$$

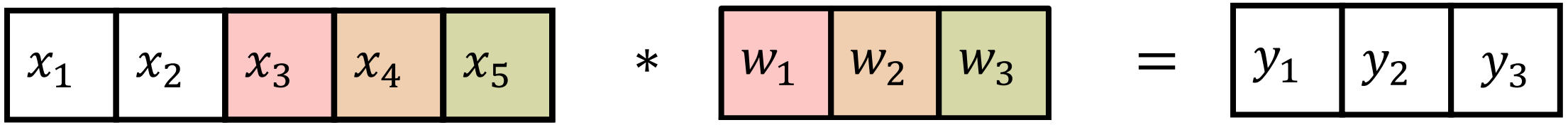


Convolutional operation (2D)

The output computation now only depends on a subset of inputs:

$$y_3 = w_1 x_3 + w_2 x_4 + w_3 x_5$$

$n - m + 1$ $\quad$ $5 - 3 + 1 = 3$



Translation equivariance

o/p of CNN → same if object in image is translated by fixed amount.

Translation equivariance is a property of Convolutional Neural Networks (CNNs) where the output of a CNN remains the same after an object within an image is translated (moved) by a fixed amount.

This means that if you translate an input image by a fixed number of pixels, the output of the CNN will be the same, up to the same translation.

object present or NOT.

→ An example of this property is seen in image classification tasks, where the presence of an object in an image remains the same regardless of its location within the image.

CNNs are translation equivariant because the same kernel is used to scan the entire image and compute the activation map: when the input image is translated, the same kernel is used to compute the activation map, leading to the same output, up to the same translation.

Same kernel to compute activation map.

Deep Learning for Time Series – Convolutional Models

Convolutional-based Neural Networks (CNNs)

CNNs for time series

CNNs can be used to process sequential data, in place of RNNs.

- CNNs look forward, while RNN models only learn from data before the timestep it needs to predict.
  - → CNNs (with shuffling) can see data from a broader perspective.
  - → RNNs can learn causality

Definition. Causality is the influence that an event (cause) contributes to a successive event (effect).

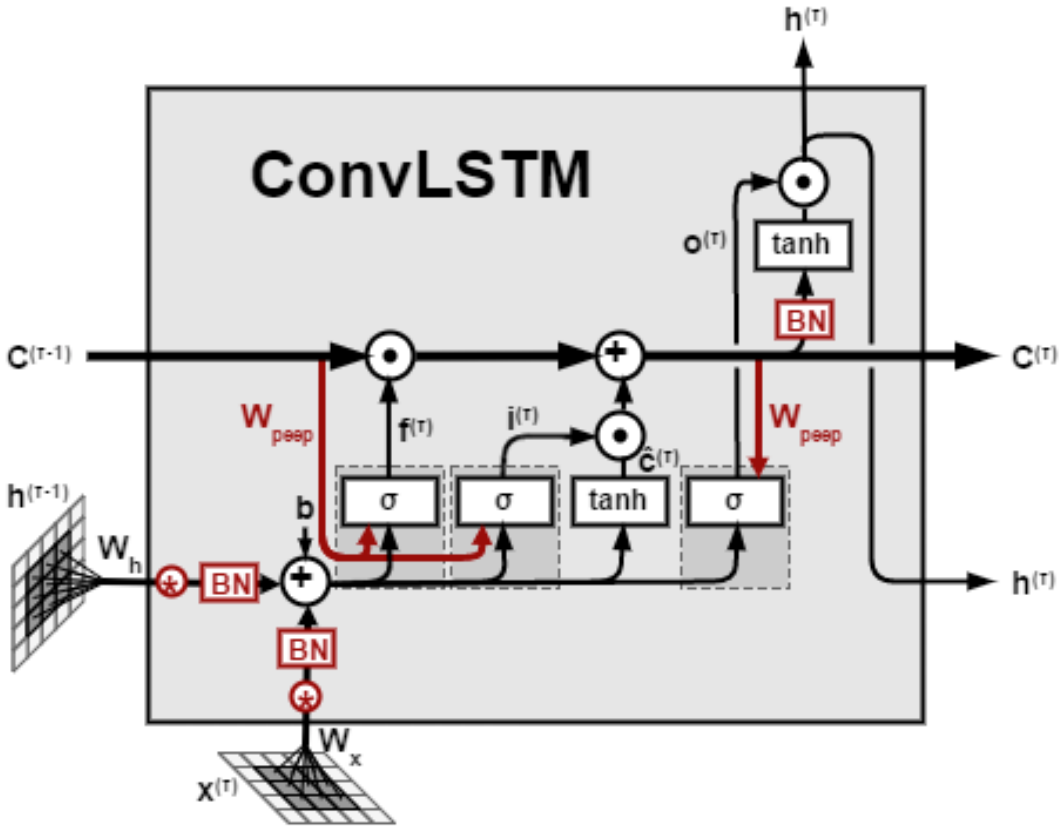
A difference with conditional statements is that causality require the antecedent to precede or coincide with the consequent in time.

ConvLSTM

The ConvLSTM is a recurrent layer, just as like as LSTM, but the matrix multiplications within the LSTM cell are replaced by convolutional operations.

→ Suited for video processing.

→ It keeps the dimensionality of the input, e.g., 3D for a single-channel video.



ConvLSTM

The ConvLSTM is defined by the following equations:

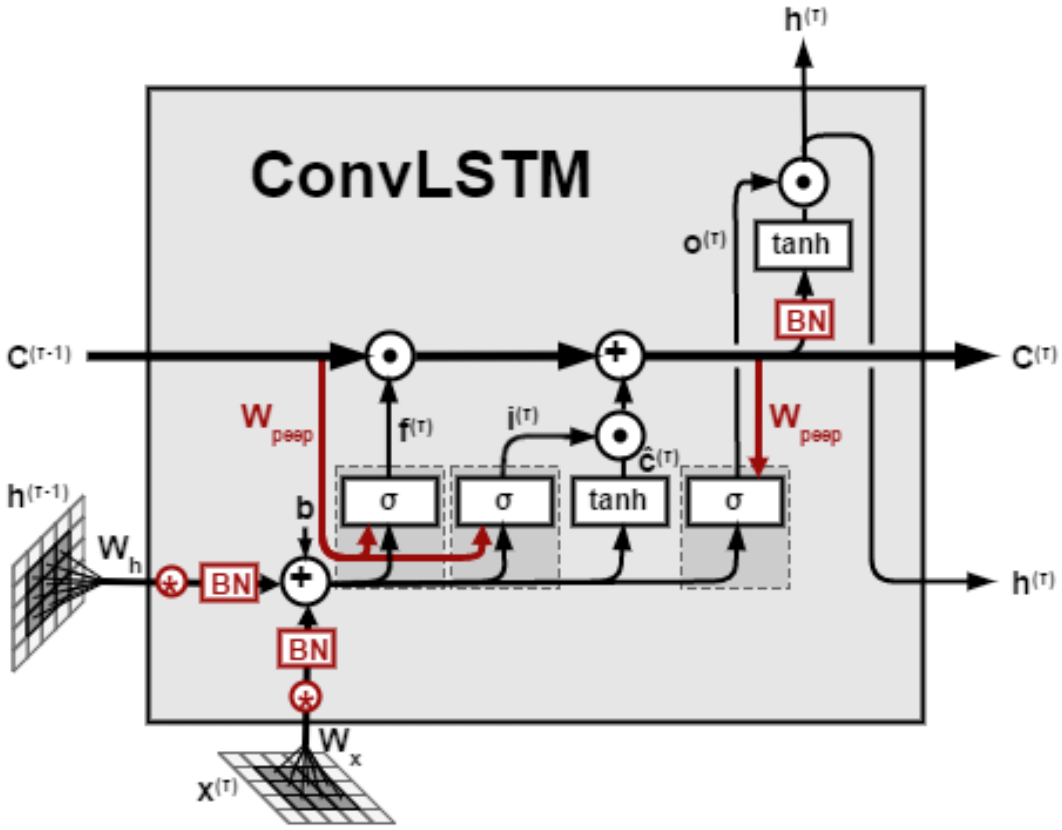
$$i_t = \sigma(W_{xi} * X_t + W_{hi} * H_{t-1} + W_{ci} \odot \mathcal{C}_{t-1} + b_i)$$

$$f_t = \sigma(W_{xf} * X_t + W_{hf} * H_{t-1} + W_{cf} \odot \mathcal{C}_{t-1} + b_f)$$

$$\mathcal{C}_t = f_t \odot \mathcal{C}_{t-1} + i_t \odot \tanh(W_{xc} * X_t + W_{hc} * \mathcal{H}_{t-1} + b_c)$$

$$o_t = \sigma(W_{xo} * X_t + W_{ho} * \mathcal{H}_{t-1} + W_{co} \odot \mathcal{C}_t + b_o)$$

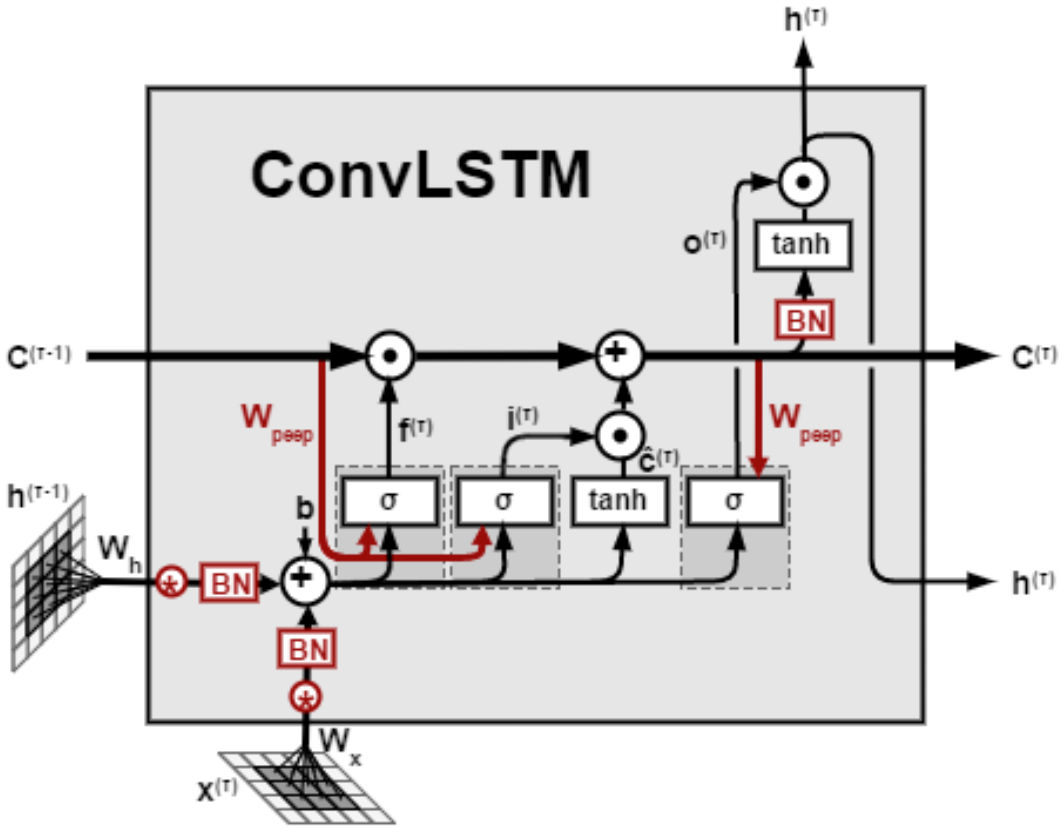
$$\mathcal{H}_t = o_t \odot \tanh(C_t)$$



ConvLSTM

ConvLSTM != Conv + LSTM
(Conv → flatten → LSTM)

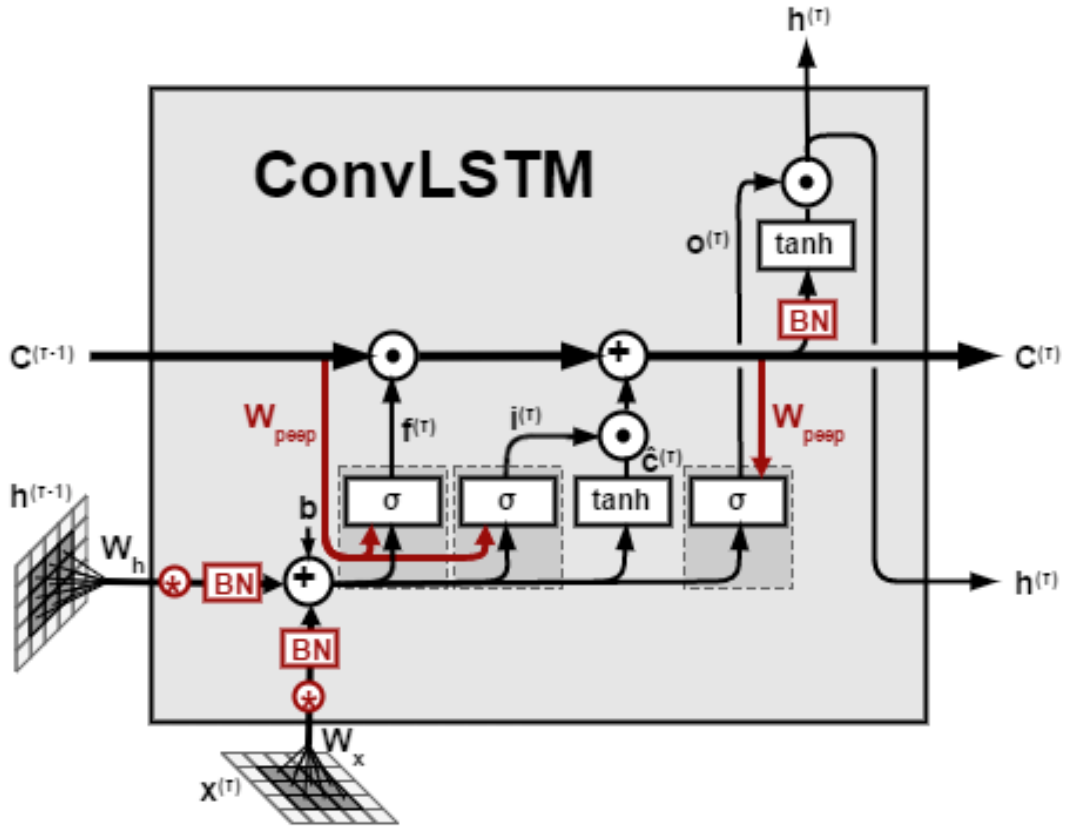
- ConvLSTM can be confused with CNN+LSTM models, i.e., a sequential combination of the two architectures which first processes an input frame through a CNN to create a flatten representation which is then fed as input to the LSTM.



ConvLSTM

ConvLSTM are successfully applied to different tasks:

- Gesture recognition [1]
- Precipitation nowcasting [2]
- Medical image segmentation [3]
- Video salient object detection [4]
- …



[1] “Attention in Convolutional LSTM for Gesture Recognition”, Zhang et al.
[2] “Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting”, Shi et al.
[3] “Multi-level Context Gating of Embedded Collective Knowledge for Medical Image Segmentation”, Asadi-Aghbolaghi et al.
[4] “Pyramid Dilated Deeper ConvLSTM for Video Salient Object Detection”, Song et al.

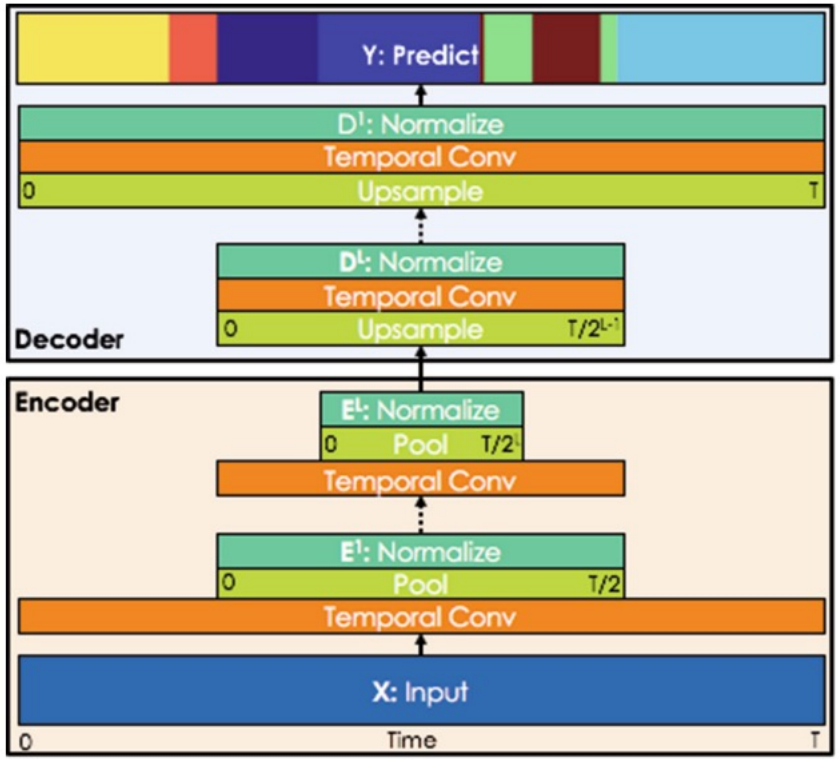
Temporal Convolutional Networks (TCNs)

Temporal Convolutional Networks (TCNs) were introduced in 2016 for video-based action segmentation [5].

It is a variation of CNN for better modeling of sequencing tasks.

The TCN provides a unified approach to capture both:

1. Low-level features encoding spatio-temporal information, and
2. High-level temporal information



[5] "Temporal convolutional networks: A unified approach to action segmentation", Lea et al.

Temporal Convolutional Networks (TCNs)

Temporal Convolutional Networks (TCNs) are based on an encoder-decoder framework.

- It takes as input a sequence of any length and output a sequence of the same length.
- The causal convolution (or, temporal convolution) main characteristic is that the output at time t is only convolved with the observation until time t.
  → There is no information leakage from the future

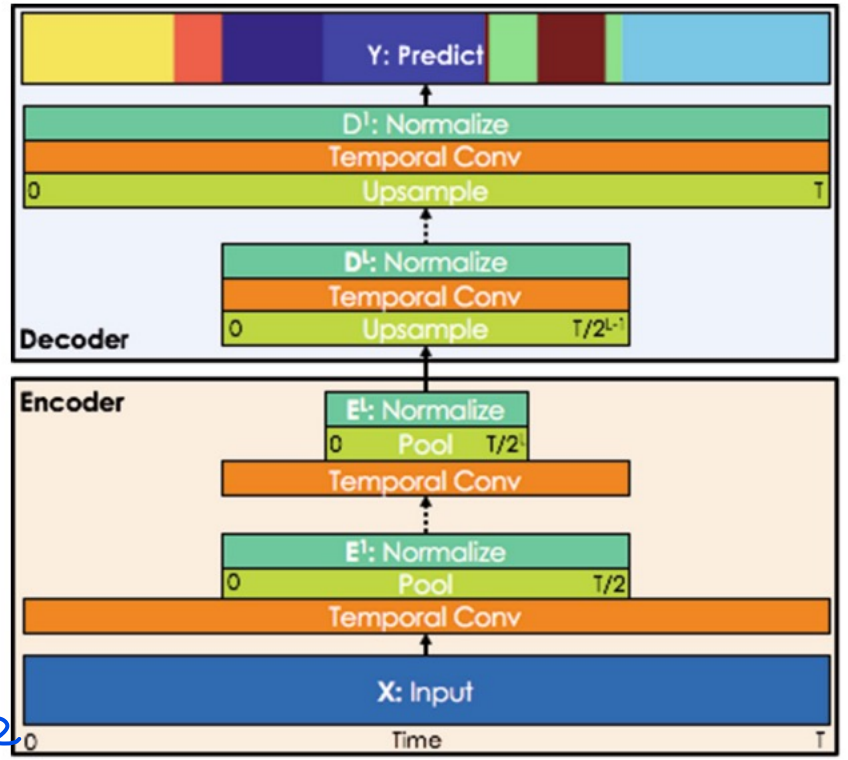


Image from the original paper: "Temporal convolutional networks: A unified approach to action segmentation", Lea et al.

Temporal Convolutional Networks (TCNs)

The causal convolution is best suited to model causality in the data.

- For images it can be implemented by "masked convolutions", i.e., a tensor mask is applied before the actual convolution takes place.
- For 1D data, e.g., audio processing, it can be more easily implemented by shifting the output of a normal convolution by a few timesteps.

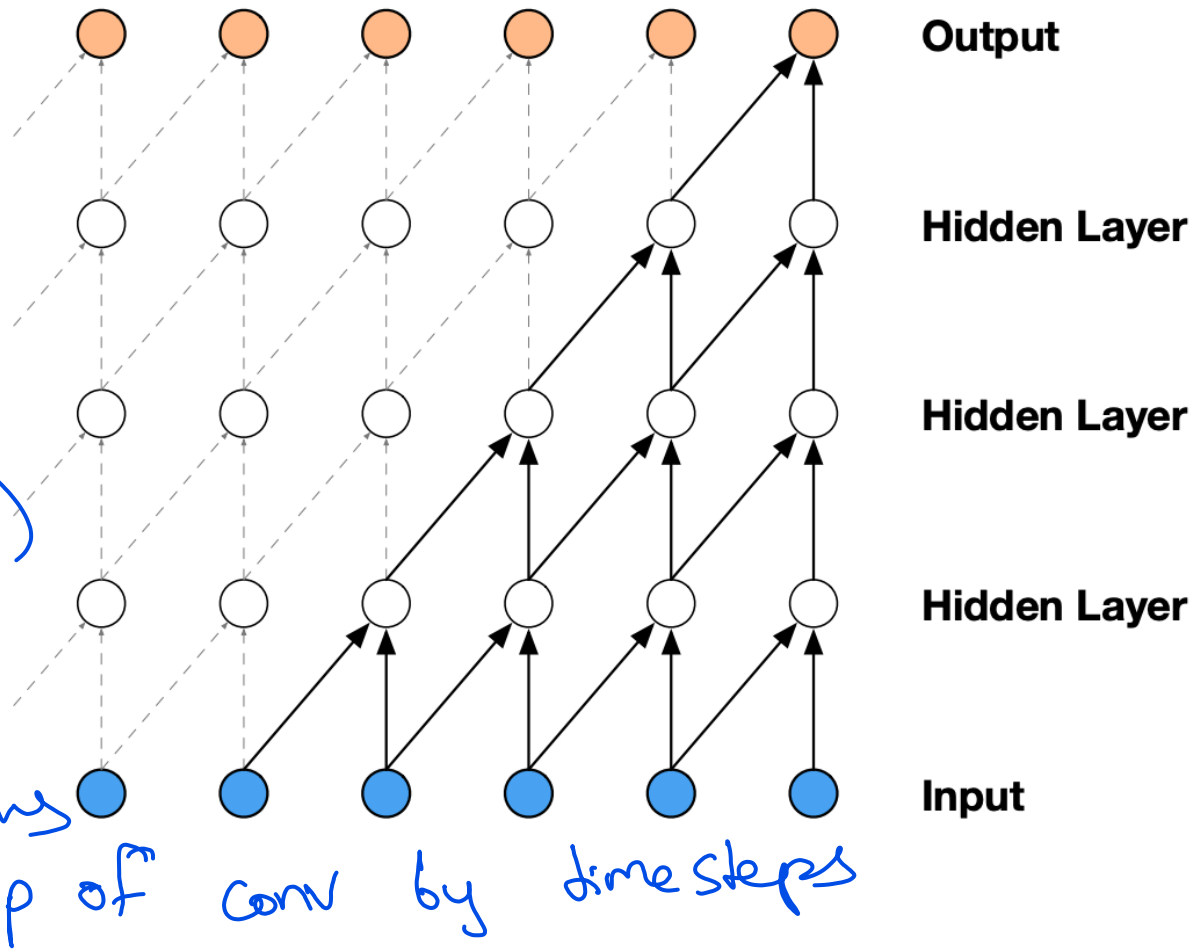
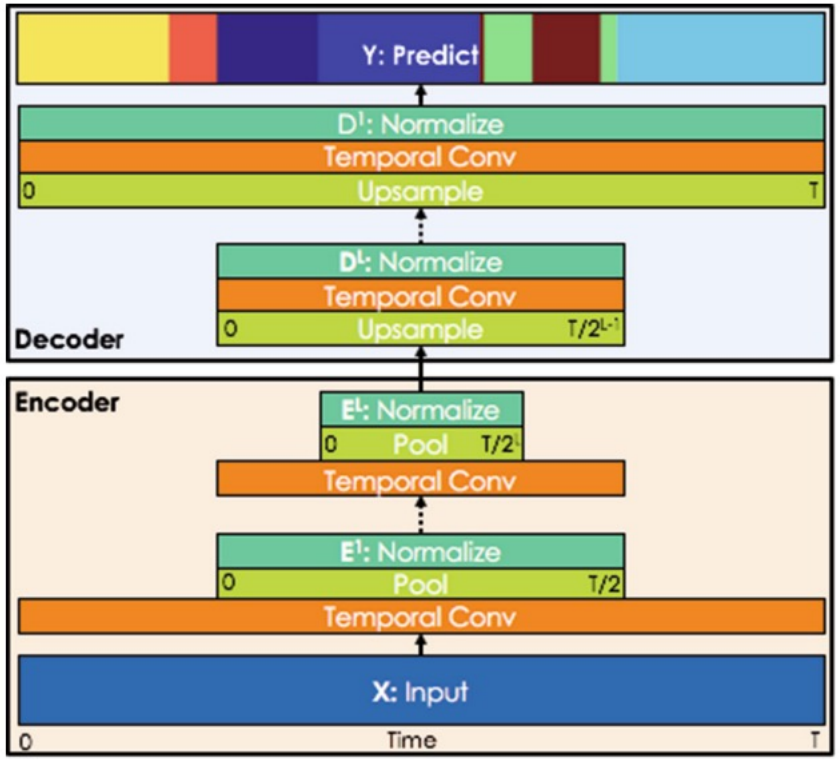


Image from the original paper: "Temporal convolutional networks: A unified approach to action segmentation", Lea et al.

Temporal Convolutional Networks (TCNs)

Among a multitude of applications, TCNs have been successfully applied to:

- Weather prediction task [6]
- Sound event localization [7]
- Action segmentation [8]
- …

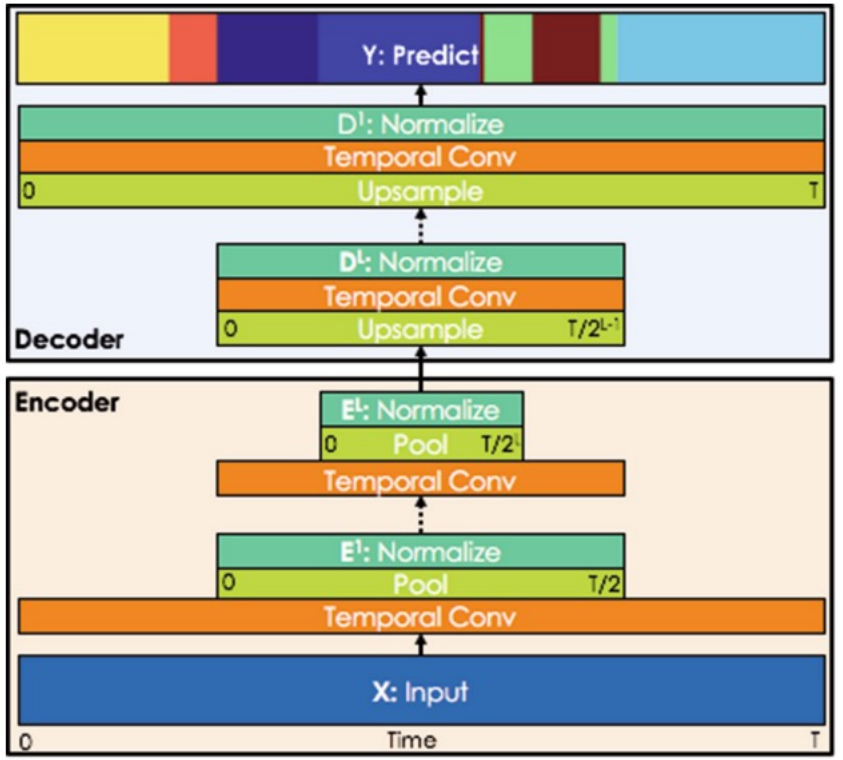


[6] “Temporal convolutional networks for the Advance prediction of enSo”, Yan, Jining, et al.
[7] “SELD-TCN: Sound Event Localization & Detection via Temporal Convolutional Networks.”, Guirguis et al.
[8] “Temporal convolutional networks: A unified approach to action segmentation.”, Colin et al.

Temporal Convolutional Networks (TCNs)

Advantages of TCNs:

- TCNs exhibit longer memory than RNNs, given the same capacity
- Parallelism of training



[6] “Temporal convolutional networks for the Advance prediction of enSo”, Yan, Jining, et al.
[7] “SELD-TCN: Sound Event Localization & Detection via Temporal Convolutional Networks.”, Guirguis et al.
[8] “Temporal convolutional networks: A unified approach to action segmentation.”, Colin et al.

Deep Learning for Time Series – Convolutional Models

Recap

In this lecture

- Convolutional neural networks (CNNs)
    - The convolution operation
    - Pooling
    - Padding
    - 1D and 2D convolutions
- Convolutional-based architectures
    - ConvLSTM
    - TCN

Critical comparison

Recurrent-based, convolutional-based, mixed

feed forward Recurrent Convolutional.

	FF-NN	RNN	CNN
Data	Tabular	Sequential	Grids
Weight-sharing	No	Yes	Yes
Recurrent connections	No	Yes	Yes
Equivariant	No	No	Yes
Parall. comp.	Yes	No	Yes
Vanishing grad.	Yes	Yes	Yes
Spatial relationships	No	No	Yes
Causality	No	Yes	No

(Spatial relationship)

(Causal)