Friedrich-Alexander-Universität
Technische Fakultät

Machine Learning
Data Analytics

FAU

Machine Learning for Time Series

Dr. Emmanuelle Salin

Machine Learning and Data Analytics (MaD) Lab
Friedrich-Alexander-Universität Erlangen-Nürnberg
12.12.2024

Topics overview

1. Time series fundamentals and definitions (Part 1)
2. Time series fundamentals and definitions (Part 2)
3. Bayesian Inference and Gaussian Processes
4. State space models (Kalman Filters)
5. State space models (Particle Filters)
6. Autoregressive models
7. Data mining on time series
8. Deep Learning (DL) for Time Series (Introduction to DL)
9. DL – Convolutional models (CNNs)
10. DL – Recurrent models (RNNs and LSTMs)
11. DL – Attention-based models (Transformers)
12. DL – From BERT to ChatGPT
13. DL – New Trends in Time Series processing
14. Time series in the real world

Topics overview

1. Time series fundamentals and definitions (Part 1)
2. Time series fundamentals and definitions (Part 2)
3. Bayesian Inference and Gaussian Processes
4. State space models (Kalman Filters)
5. State space models (Particle Filters)
6. Autoregressive models
7. Data mining on time series
8. Deep Learning (DL) for Time Series (Introduction to DL)
9. **DL – Convolutional models (CNNs)**
10. DL – Recurrent models (RNNs and LSTMs)
11. DL – Attention-based models (Transformers)
12. DL – From BERT to ChatGPT
13. DL – New Trends in Time Series processing
14. Time series in the real world

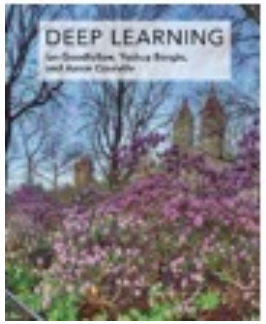
In this Lecture...

1. The Convolutional Operation
2. Convolutional Neural Networks
3. Convolutional Neural Networks for Time Series
4. Convolutional neural networks and time series imaging
5. Convolutional-based architectures

References

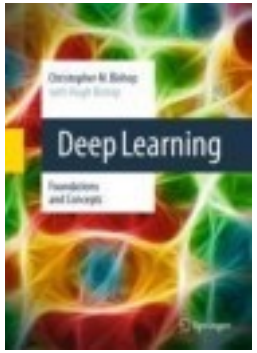
Deep Learning

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



Deep Learning: Foundations and Concepts

by C. Bishop, H. Bishop (2024)



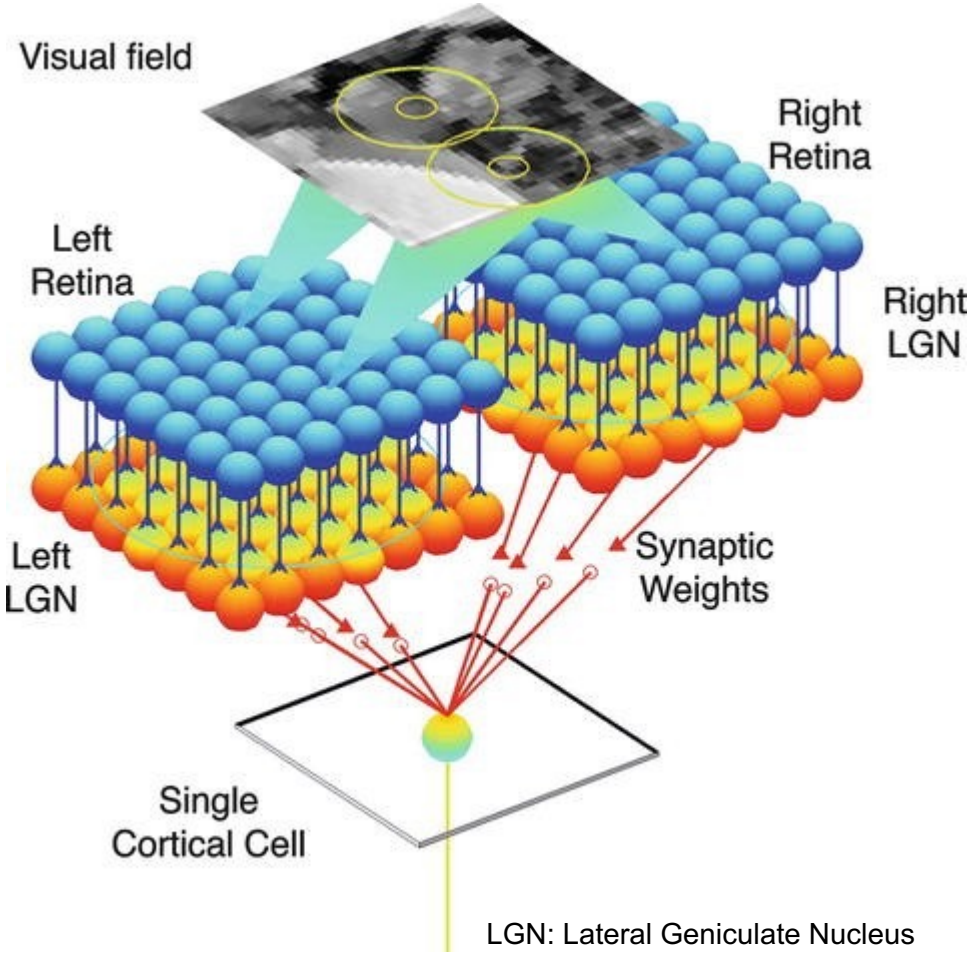
Deep Learning for Time Series – Convolutional Models
Introduction

Motivation: The Human Visual Cortex

CNNs were inspired by a specialized brain area: the **visual cortex**.

1) The visual cortex processes visual information collected by the retinae in a **hierarchical** manner.

2) The **receptive field** of cortical neurons increases the later they are in this hierarchy.
   a) Small receptive fields are stimulated by high spatial frequencies - or fine details
   b) Large receptive fields are stimulated by low spatial frequencies - or coarse details

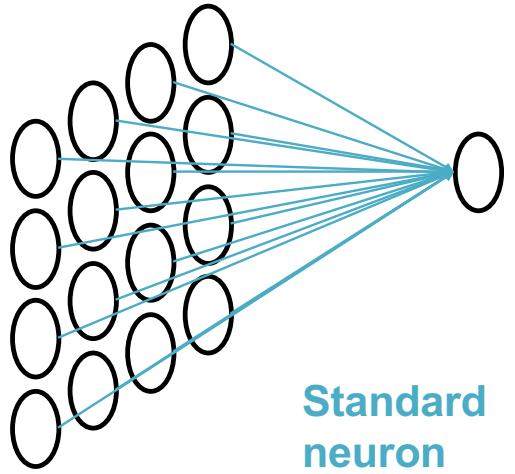


LGN: Lateral Geniculate Nucleus

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

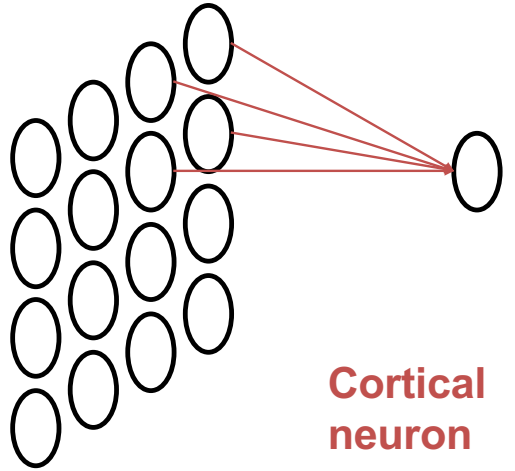
Modeling Receptive Fields

- In standard **multilayer perceptrons** (MLP), layers are **fully-connected,** i.e. all units from one layer are connected to all units from the previous layer.



- Cortical neurons have small **receptive fields**: units from one layer have **sparse** and **localized** connection with units from the previous layer.

→ This is modeled by discrete convolutional operations.



Deep Learning for Time Series – Convolutional Models
The convolutional operation

Convolutional Operation

The **continuous convolution** is a operation on two functions f and g noted $f * g$ that produces a third function.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau$$

It is the integral of the product of the two functions after one is reflected about the y-axis and shifted.

In deep learning, a **discrete** version of this operation is used.

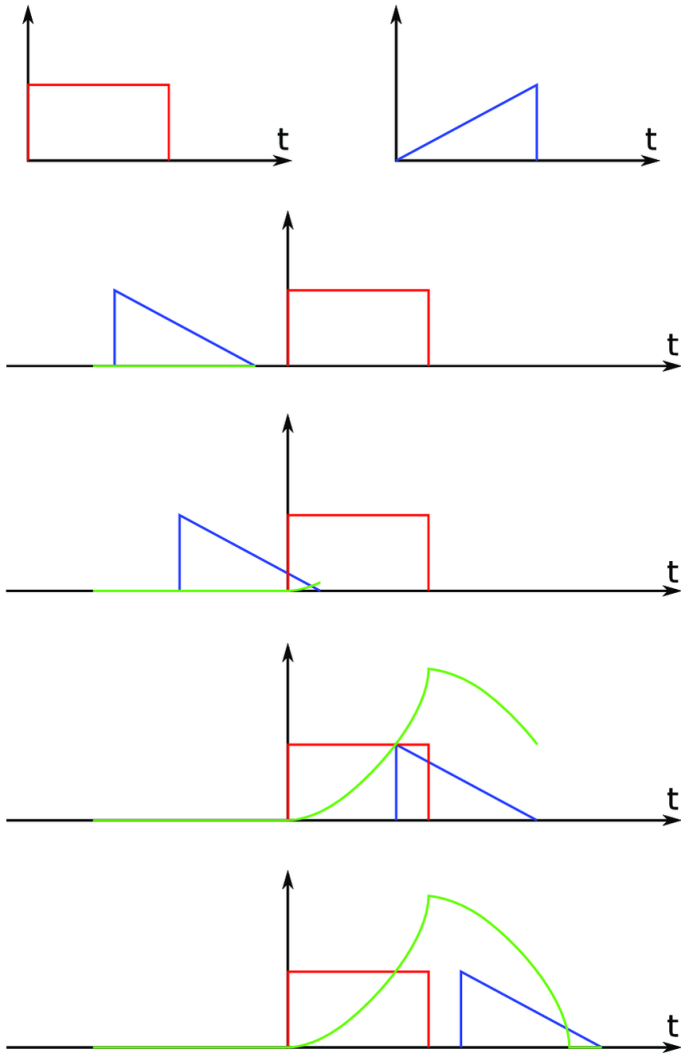


Image from Pihlajamäki, Tapani. Multi-resolution Short-time Fourier Transform Im- plementation of Directional Audio Coding.

Discrete Convolution

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

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

→ 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.

2D Convolutional Operation

Due to the small receptive field (filter), the output depends on a **subset** of inputs:

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

Input

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}

$*$

Filter

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

$=$

Output / Feature map

y_{11}		

2D Convolutional Operation

Due to the small receptive field, the output only depends on a **subset** of inputs:

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

Input

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}

$*$

Filter

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

$=$

Output / Feature map

y_{11}	y_{12}	

2D Convolutional Operation

Due to the small receptive field, the output only depends on a **subset** of inputs:

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

Input

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}

$*$

Filter

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

$=$

Output / Feature map

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

2D Convolutional Operation

Due to the small receptive field, the output only depends on a **subset** of inputs:

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

Input

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}

$*$

Filter

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

$=$

Output / Feature map

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

2D Convolutional Operation

Due to the small receptive field, the output only depends on a **subset** of inputs:

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

Input

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}

$*$

Filter

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

$=$

Output / Feature map

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

2D Convolutional Operation

Due to the small receptive field, the output only depends on a **subset** of inputs:

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

Input

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}

$*$

Filter

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

$=$

Output / Feature map

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

2D Convolutional Operation

Due to the small receptive field, the output only depends on a **subset** of inputs:

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

Input

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}

$*$

Filter

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

$=$

Output / Feature map

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

2D Convolutional Operation

Due to the small receptive field, the output only depends on a **subset** of inputs:

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

Input

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}

$*$

Filter

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

$=$

Output / Feature map

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

2D Convolutional Operation

Due to the small receptive field, the output only depends on a **subset** of inputs:

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

Input

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}

$*$

Filter

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

$=$

Output / Feature map

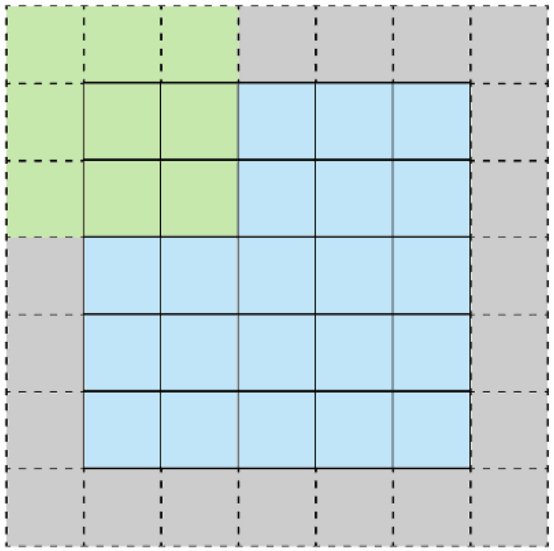
y_{11}	y_{12}	y_{13}
y_{21}	y_{22}	y_{23}
y_{31}	y_{32}	y_{33}

Padding Operation

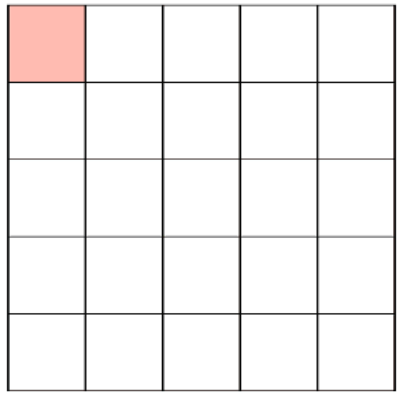
The convolution, as illustrated previously, leads to a **reduction** of the dimension.

→ 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 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

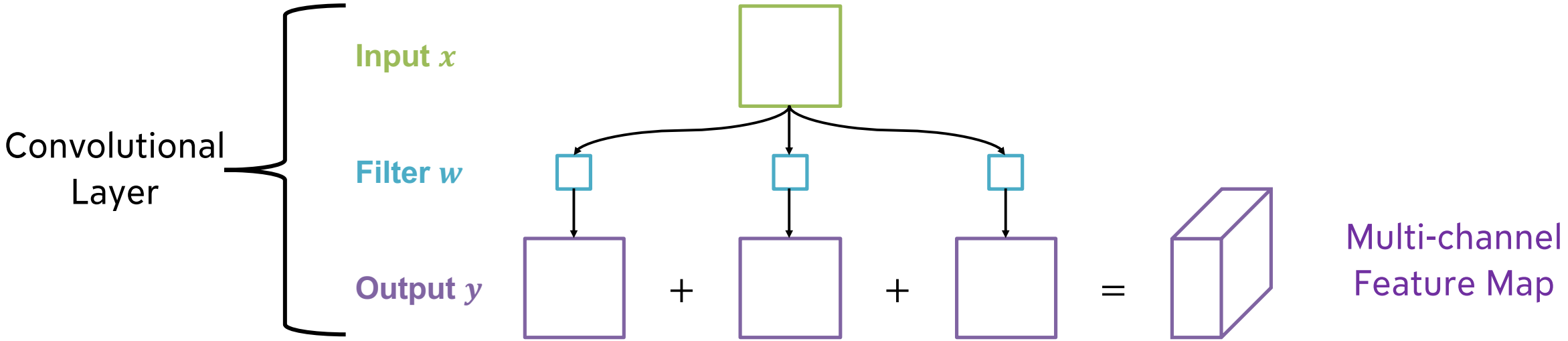
Deep Learning for Time Series – Convolutional Models
Convolutional Neural Networks

Convolutional Layer

CNNs are composed of stacked **convolutional layers**.

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 outputs are **concatenated** to create a **multi-channel feature map**.



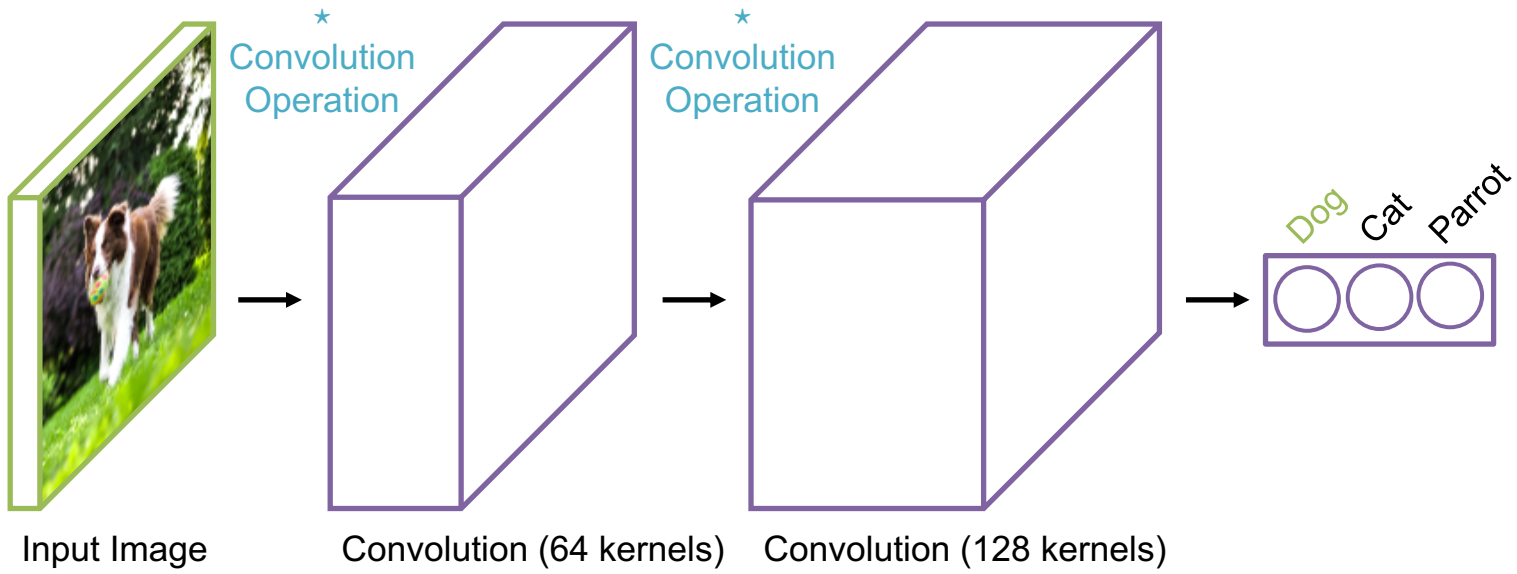
Convolutional Neural Network (CNN)

Parameters of a convolution layer in a CNN :

- **Input channels:** The number of channels of the input image
- **Output channels:** The number of channels produced by the convolution
- **kernel size**: Size of the filter matrix (the filter is usually a square matrix)
- **Stride**: Number of pixels the filter moves in one direction
- **Padding**

Convolutional Neural Network (CNN)

The **stacked** convolutional layers incrementally process relevant features in a hierarchical manner, forming a **deep convolutional neural network** (CNN).

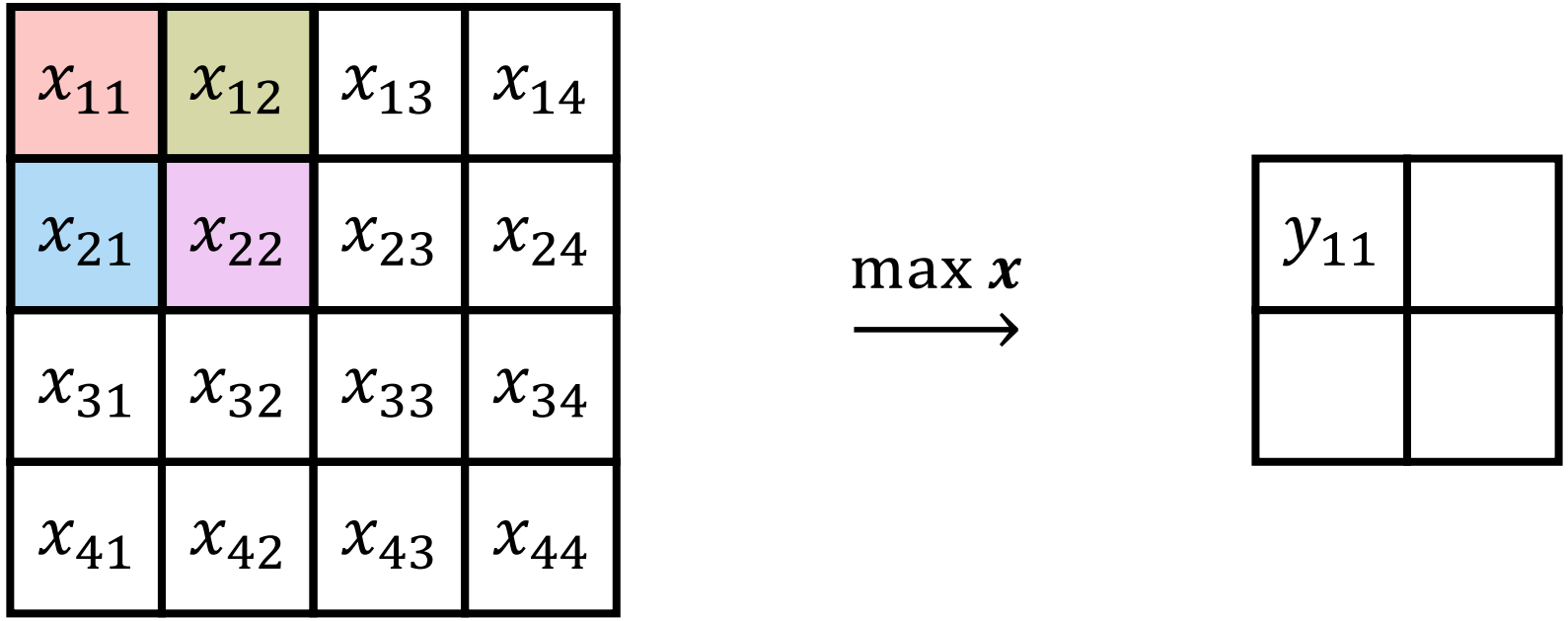


→ However, after many layer the number of features grow very quickly computationally expensive: Pooling layers were introduced to limit the computation

Pooling Operation: the Example of Max Pooling

The **pooling operation** is introduced to reduce the dimension and number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution (less focused on fine-grained details).

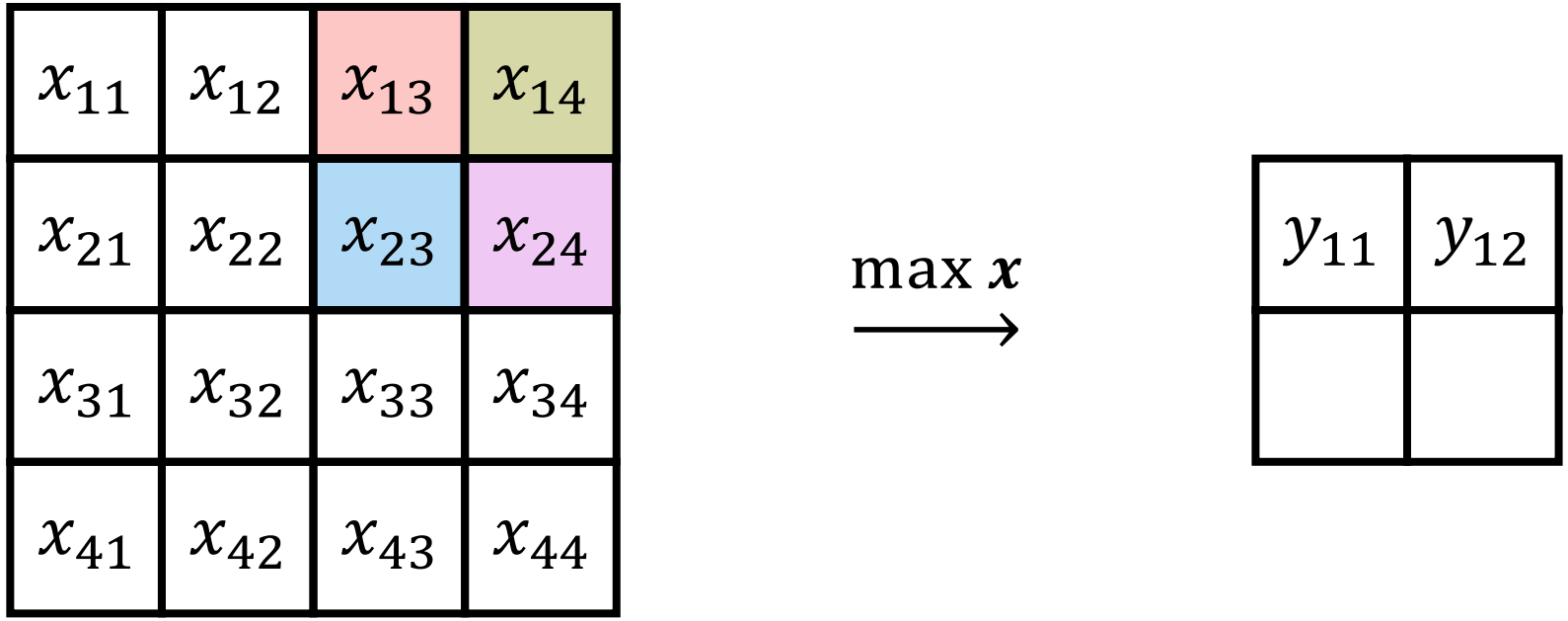
$$y_{11} = \max(x_{11}, x_{12}, x_{21}, x_{22})$$



Pooling Operation: the Example of Max Pooling

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 (less focused on fine-grained details).

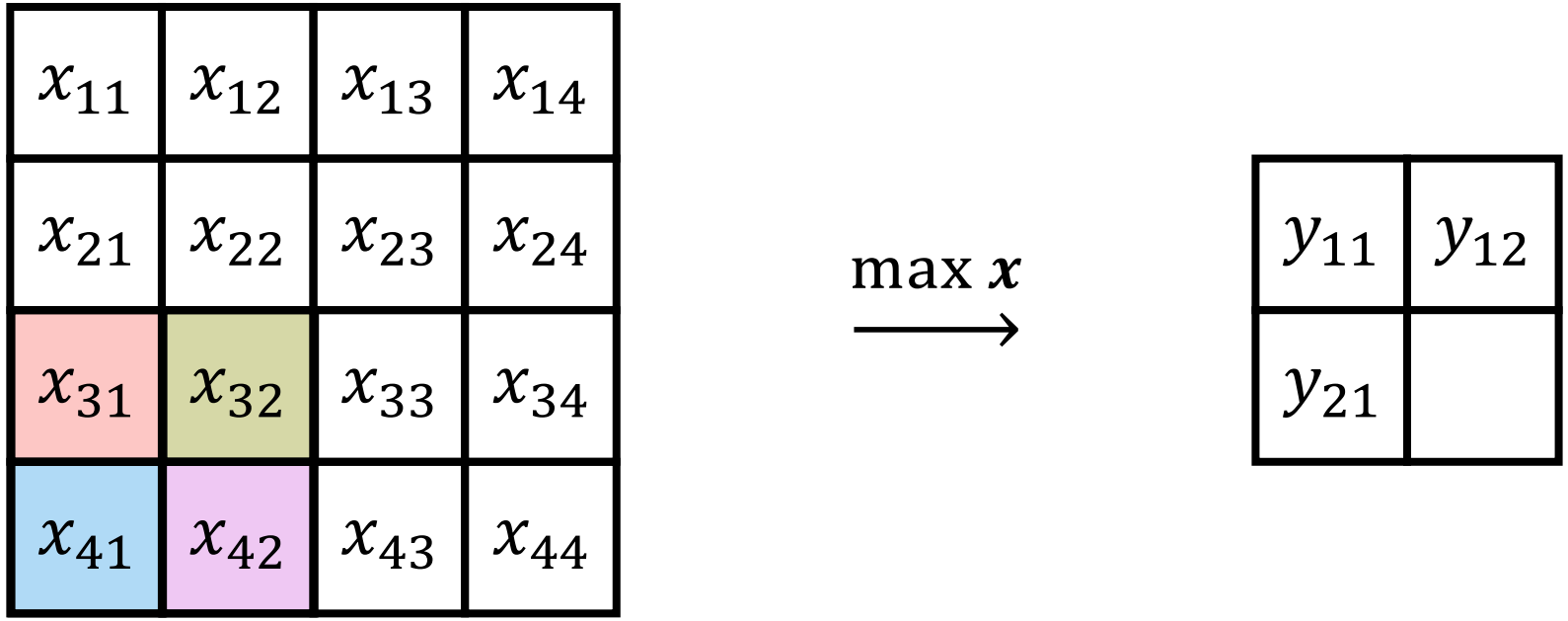
$$y_{12} = \max(x_{13}, x_{14}, x_{23}, x_{24})$$



Pooling Operation: the Example of Max Pooling

The **pooling operation** is introduced to reduce the dimension and number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution (less focused on fine-grained details).

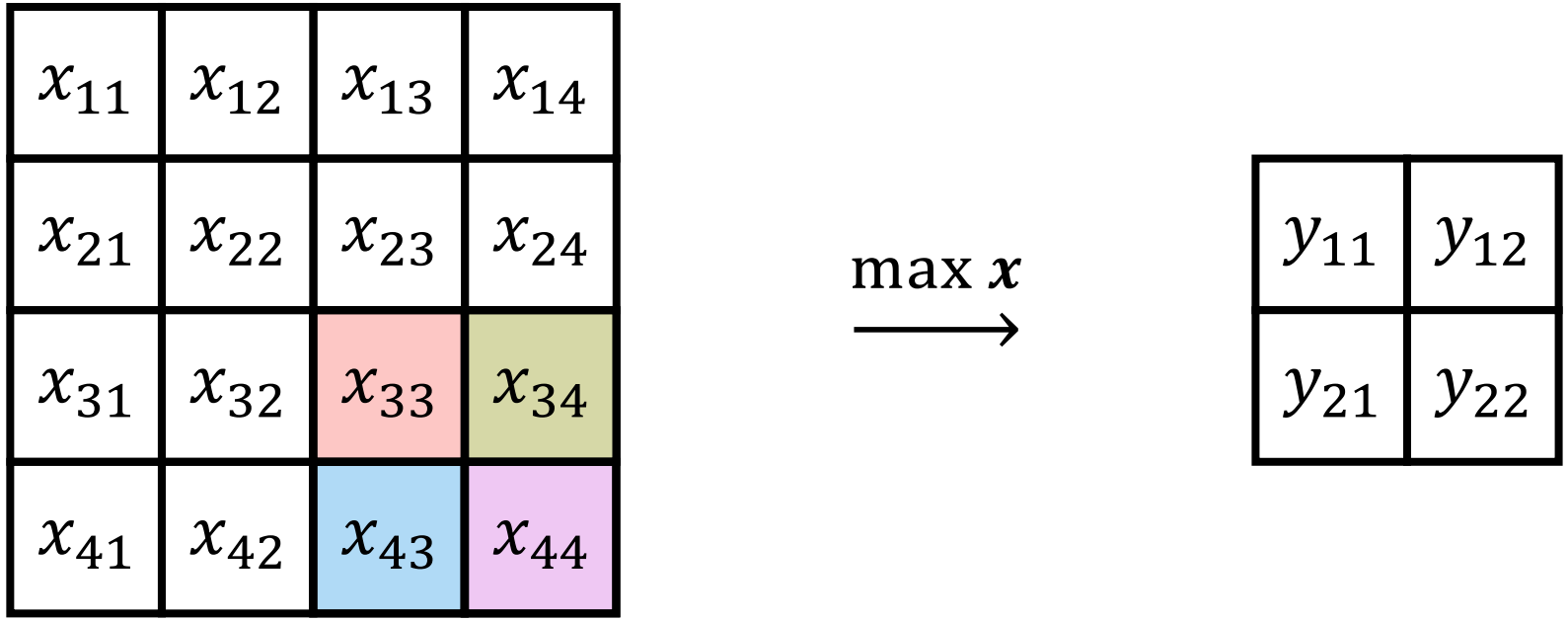
$$y_{21} = \max(x_{31}, x_{32}, x_{41}, x_{42})$$



Pooling Operation: the Example of Max Pooling

The **pooling operation** is introduced to reduce the dimension and number of computations in a CNN. From a theoretical perspective, higher representations do not require high spatial resolution (less focused on fine-grained details).

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



Pooling Operations

Other pooling operations can be used instead of the max pooling:

- **Mean** pooling
- **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

Translation Equivariance

Translation equivariance is a property of CNNs stating that the output of a CNN **remains the same** after an object within an image is **translated** (moved) by a fixed amount.

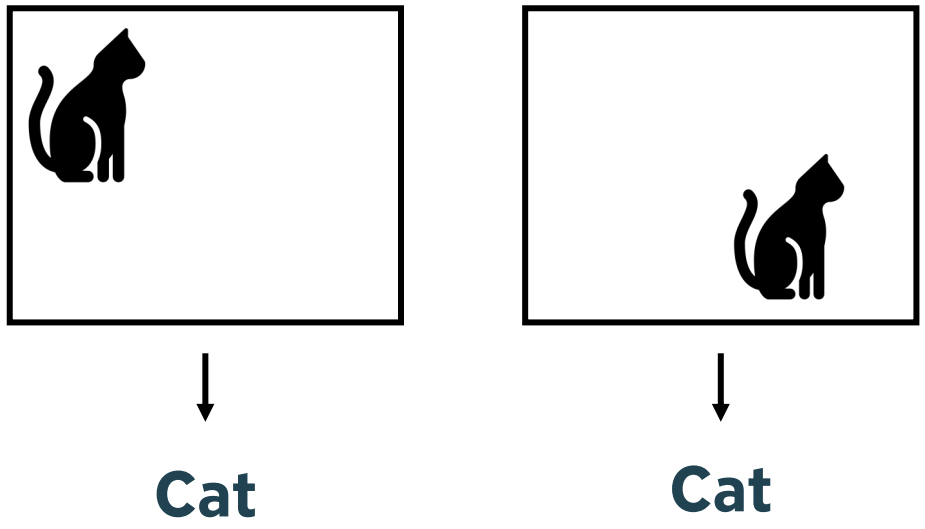
This means that if you translate an input image by a fixed number of pixels, the output of the CNN will be the same, up to the same translation.

CNNs are translation equivariant because the same kernels are 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 feature map, leading to the same output, up to the same translation.

Translation Equivariance

Translation equivariance is a property of CNNs stating that the output of a CNN **remains the same** after an object within an image is **translated** (moved) by a fixed amount.

→ This property is useful in image classification tasks, where the presence of an object in an image remains the same regardless of its location within the image.



Data Augmentation

Data augmentation is usually used to generate new data samples from existing data. The goal is to artificially increase the **data size** and improve model **generalizability**.

In **computer vision**, images can be applied random transformations with minimal impact on the relevant information contained in the image (flipping, cropping, ...)

Examples of ColorJitter augmentation from Pytorch

https://pytorch.org/vision/stable/auto_examples/transforms/plot_transforms_illustrations.html#sphx-glr-auto-examples-transforms-plot-transforms-illustrations-py

Data Augmentation

Data augmentation is usually used to generate new data samples from existing data. The goal is to artificially increase the **data size** and improve model **generalizability**.

In **computer vision**, images can be applied random transformations with minimal impact on the relevant information contained in the image (flipping, cropping, ...)

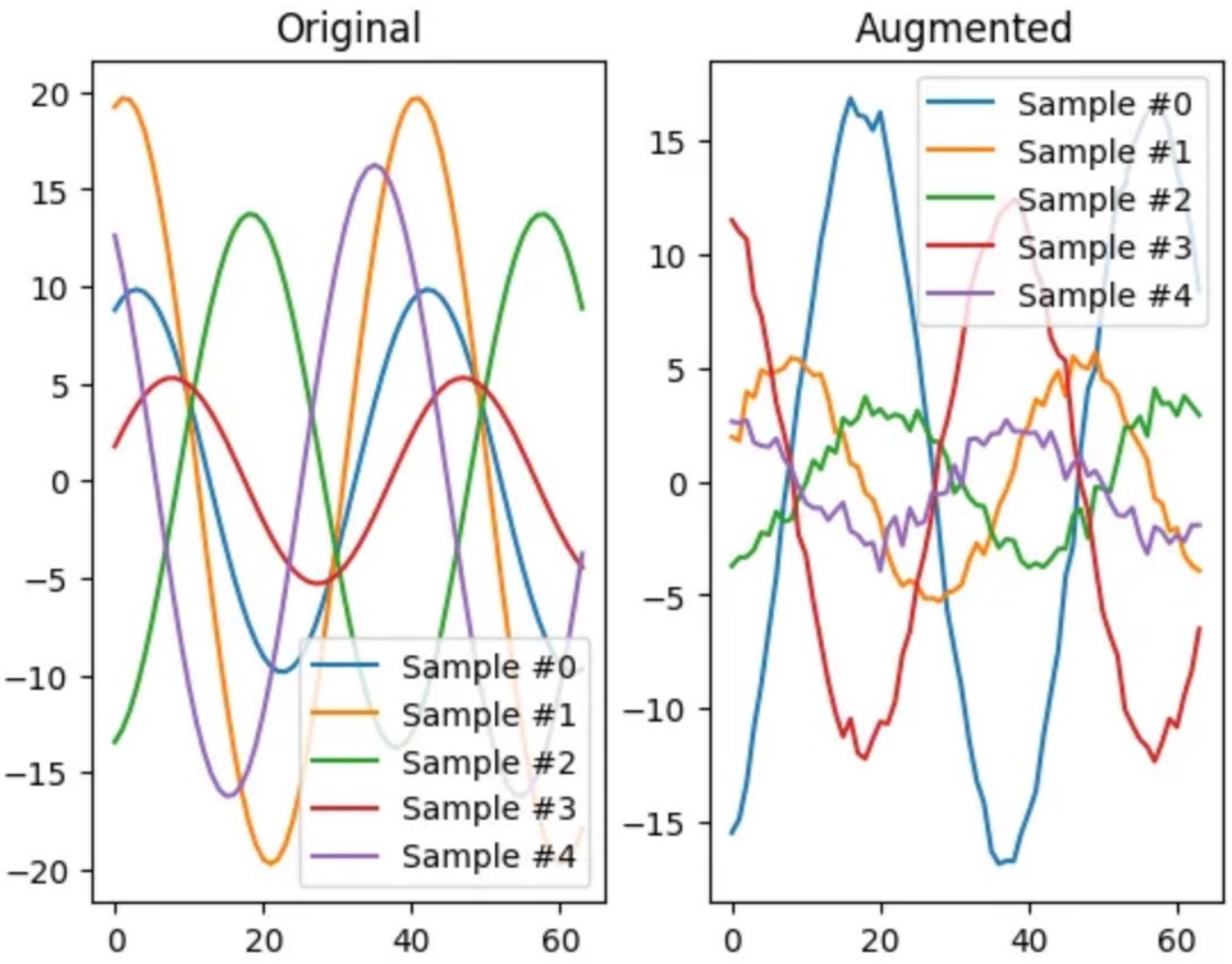
Time series data can be similarly augmented (scaling, permutation, adding noise ...)

Data Augm

Data augm existing
data. The g l
generalizab

In **compute** h minimal
impact on t pping, ...)

Time series ding noise ...)

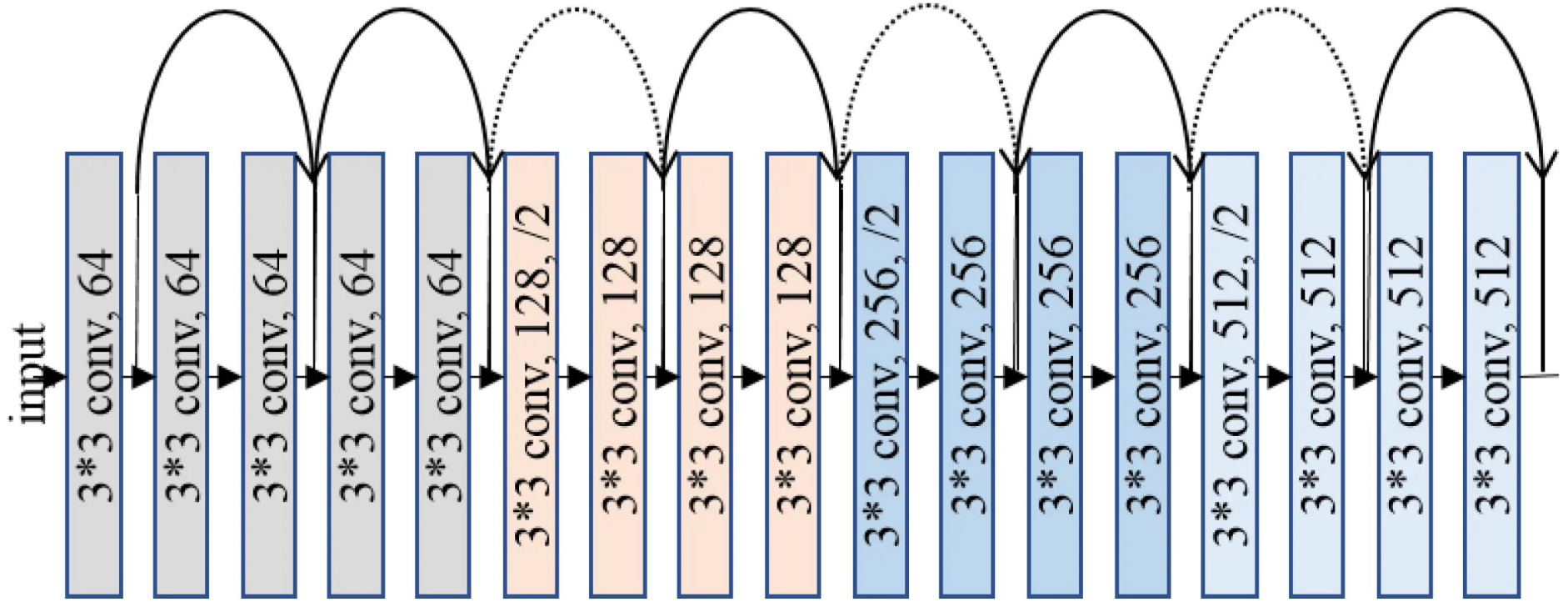


Original time series and synthetic data generated via Jittering.

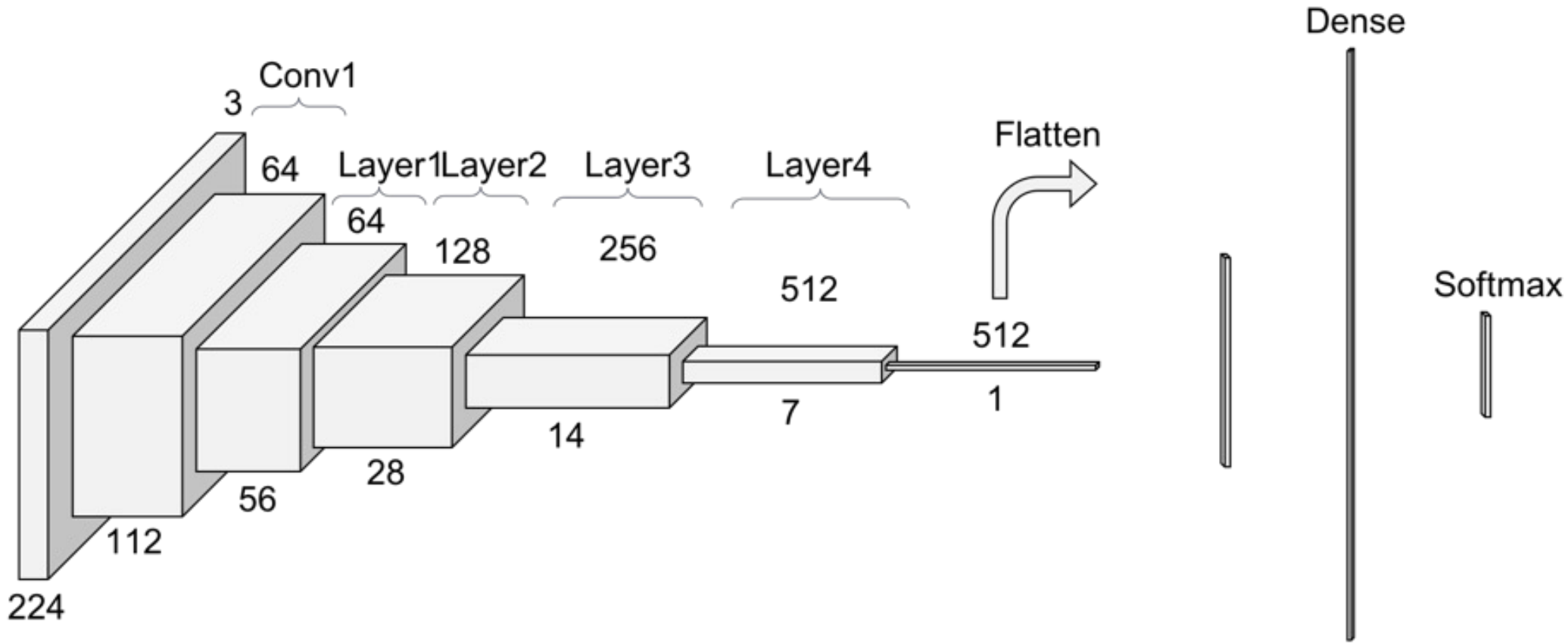
https://towardsdatascience.com/time-series-augmentations-16237134b29b

An Example of CNN: The ResNet Models

The **ResNet** models (for Residual Network) use shortcut connections (or **residual**) connections between convolution layers, which mitigates the vanishing gradient problem.



An example of CNN: The ResNet Models



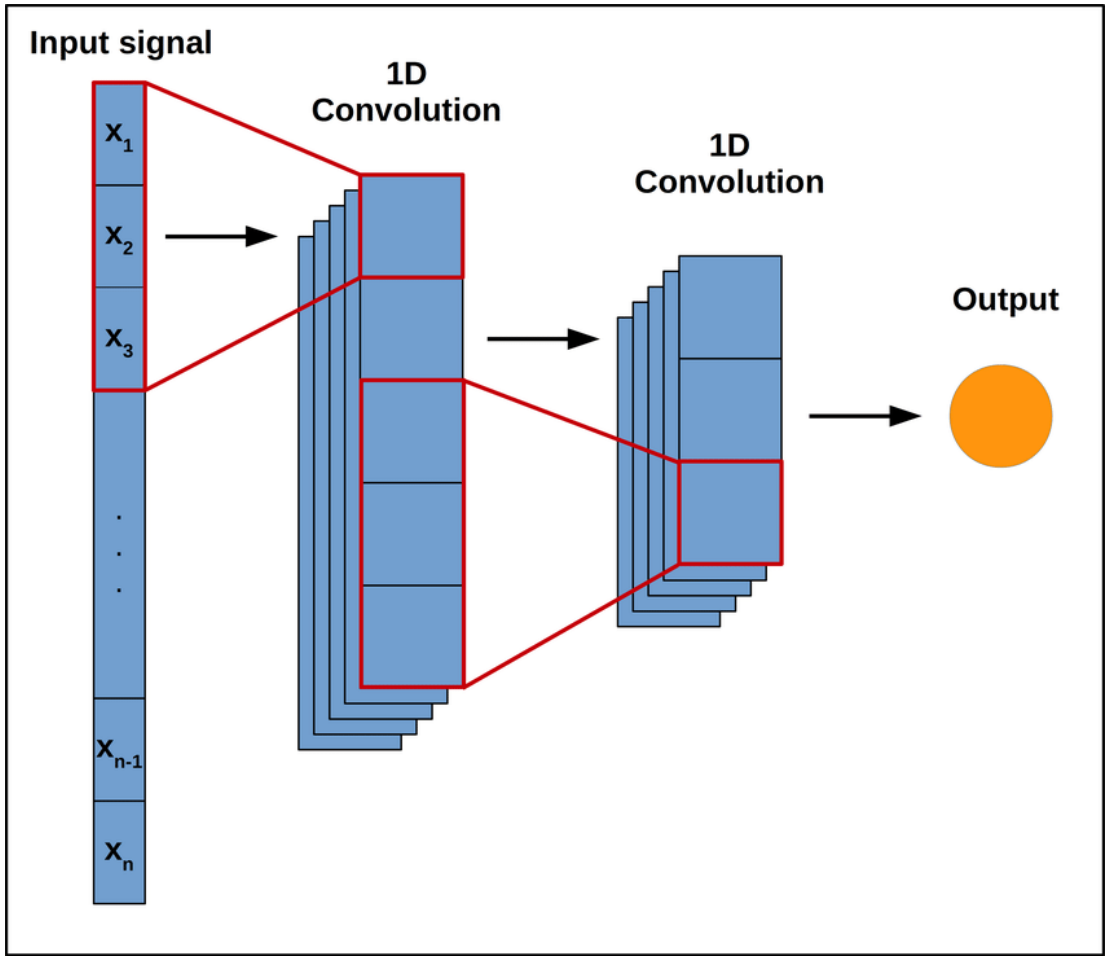
https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8

Deep Learning for Time Series – Convolutional Models
Convolutional Neural Networks for Time Series

1-D Convolutional Neural Networks

CNNs can also 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.

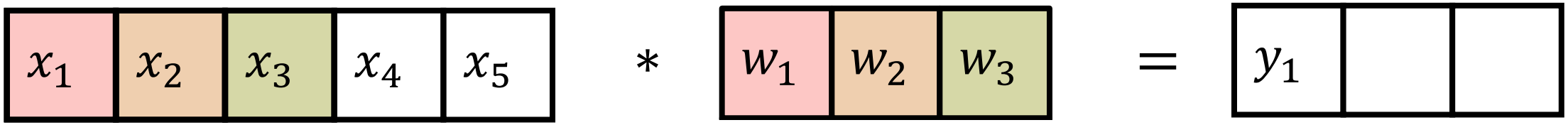


Picture from Shenfield, Alex & Howarth, Martin. (2020). A Novel Deep Learning Model for the Detection and Identification of Rolling Element-Bearing Faults. Sensors (Basel, Switzerland). 20. 10.3390/s20185112.

1D Convolutional Operation

Like the 2D operation, the output computation only depends on a **subset** of the time series:

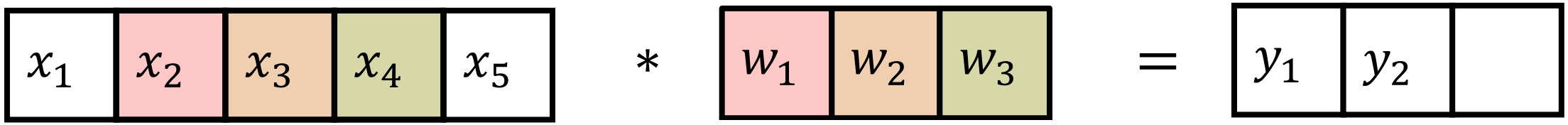
$$y_1 = w_1 x_1 + w_2 x_2 + w_3 x_3$$



1D Convolutional Operation

Like the 2D operation, the output computation only depends on a **subset** of the time series:

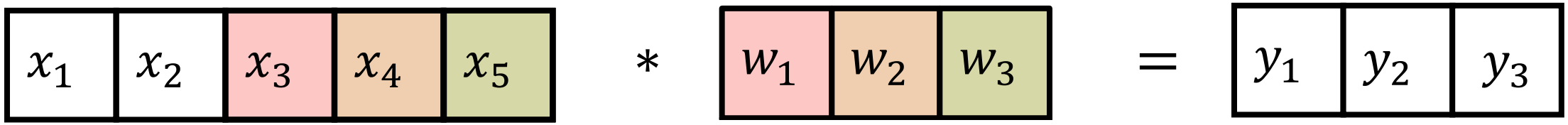
$$y_2 = w_1 x_2 + w_2 x_3 + w_3 x_4$$



1D Convolutional Operation

Like the 2D operation, the output computation only depends on a **subset** of the time series:

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



CNN Properties and Time Series

CNN properties make them very appropriate for **computer vision**. Those can also be useful for **time series**:

- Ability to extract **local patterns**: 1D CNNs can extract local patterns that occur in time series data (e.g. short term trends in stock prices, weather events)
- **Translation Equivariance**: 1D CNNs can identify those patterns regardless of their position, which is useful for tasks such as anomaly detection
- **Dimensionality reduction** of pooling layers: This can be useful when datasets are high dimensional.
- **Hierarchical** feature learning: Time series are composed of patterns at different time **scales**. (e.g. seasonal weather patterns vs sudden weather event)

Adapting CNNs for Time Series Tasks

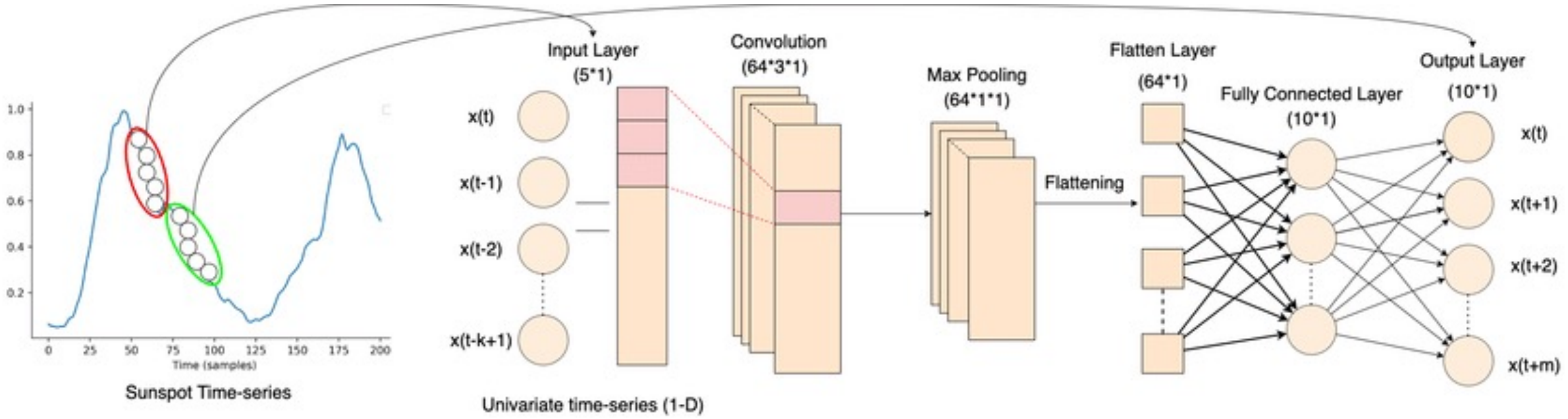
CNNs can be used for various tasks: **forecasting, classification, anomaly detection, segmentation...**

The CNN model architecture can be adapted :

- Appropriate input pre-processing (filters, time-windows ...)
- Adding fully connected layers
- Choosing relevant convolution layer parameters (kernel size, output channels...)
- Using an appropriate loss for training

CNN for Time Series Forecasting

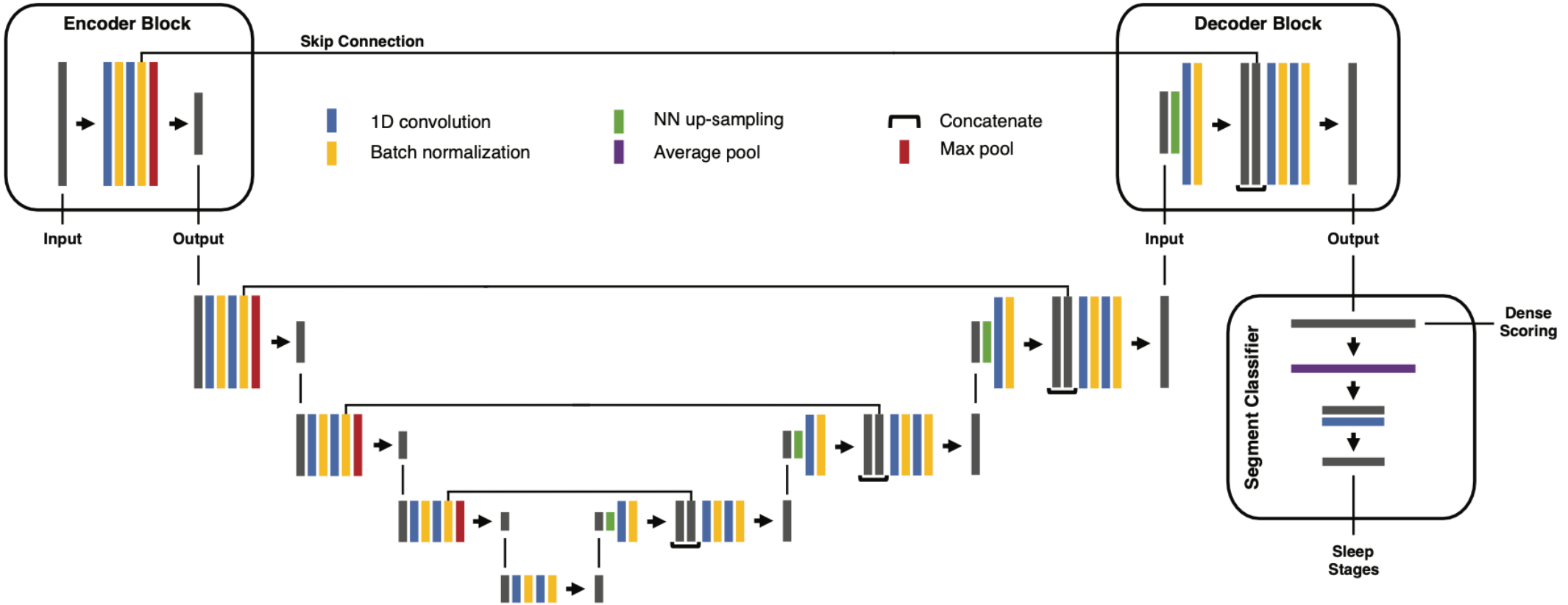
Example of CNN for univariate time series **forecasting** [1].



Chandra, Rohitash & Goyal, Shaurya & Gupta, Rishabh. (2021). Evaluation of deep learning models for multi-step ahead time series prediction. 10.48550/arXiv.2103.14250.

CNNs for Time Series Segmentation

Example of CNN model (U-Time, inspired by U-net) for time series **segmentation** [1]. It maps inputs to sequences of class labels on a freely chosen temporal scale.

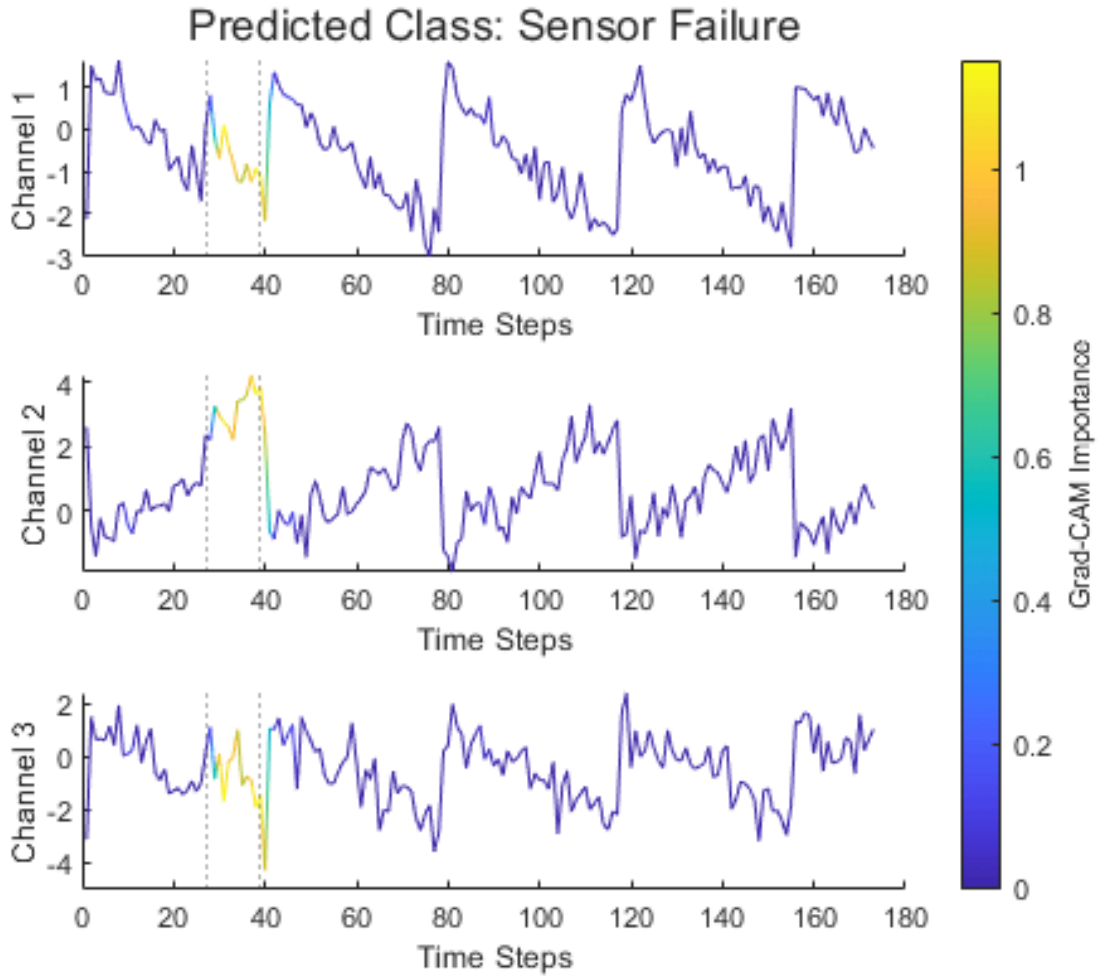


Perslev, Mathias, et al. "U-time: A fully convolutional network for time series segmentation applied to sleep staging." *Advances in Neural Information Processing Systems* 32 (2019).

CNNs and Interpretability

With the CNN architecture, **saliency maps** can be used to highlight which parts of the input most contributed to the output.

GradCAM [1] is a common interpretability method for CNNs.



[1] Selvaraju, Ramprasaath R., Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization.” International Journal of Computer Vision 128, no. 2

Image from https://de.mathworks.com/help/deeplearning/ug/interpret-time-series-classifications-with-grad-cam.html

CNNs and Interpretability

Input Image

CNN

Activation Maps

A^k

Any Network Layer Types Depending on Network Task

FC Layer

Softmax

C

Tiger Cat

y^c

(Prior to Softmax)

Compute Gradients

$\frac{\partial y^c}{\partial A^k}$

Gradient Maps

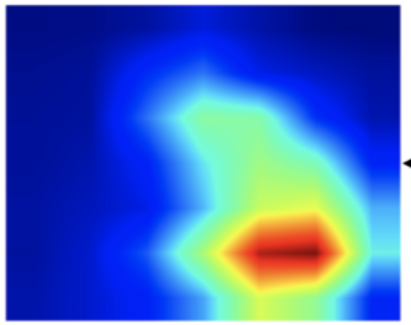
$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k}$

Global Average Pool Gradient Maps

$\alpha_k^c A^k$

Σ

ReLU



Grad-CAM Heatmap

Grad-CAM Importance

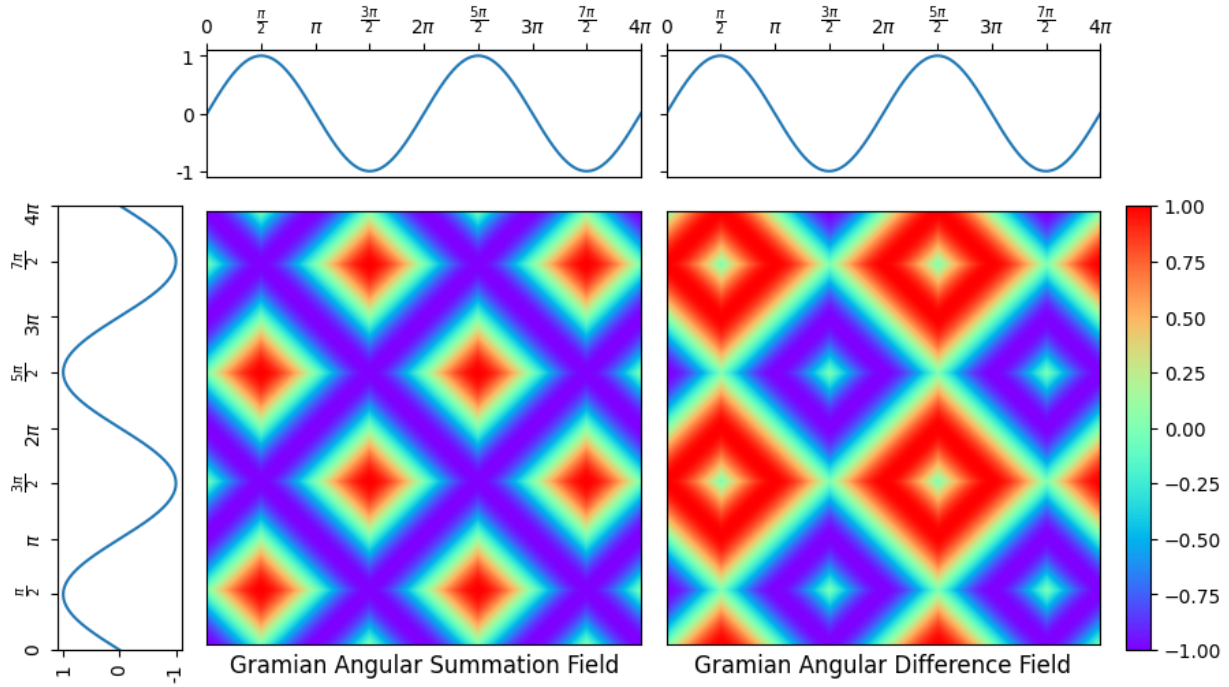
https://learnopencv.com/intro-to-gradcam/

Deep Learning for Time Series – Convolutional Models
Convolutional Neural Networks and Imaging

2D CNNs for Time Series

Beyond 1D CNN, traditional 2D CNNs can also be used with time series data by applying **imaging techniques** to the raw time series data [1]:

- Gramian Angular Fields
- Markov Transition Field
- Recurrence Plots
- Bilinear interpolation



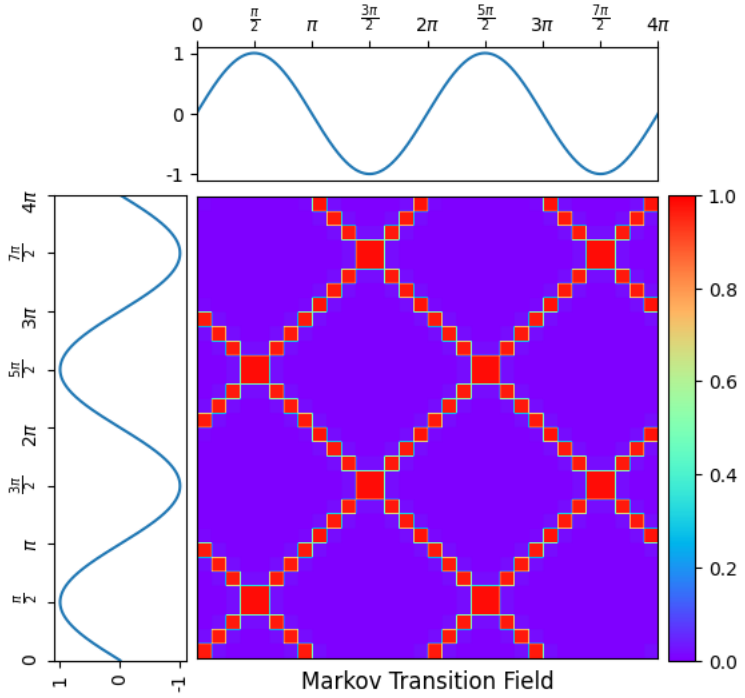
[1] Mohammadi Foumani, Navid, et al. "Deep learning for time series classification and extrinsic regression: A current survey." *ACM Computing Surveys* 56.9 (2024): 1-45.

Images fom https://pyts.readthedocs.io/en/stable/auto_examples/image/plot_single_gaf.html

2D CNNs for Time Series

Beyond 1D CNN, traditional 2D CNNs can also be used with time series data by applying **imaging techniques** to the raw time series data [1]:

- Gramian Angular Fields
- Markov Transition Field
- Recurrence Plots
- Bilinear interpolation



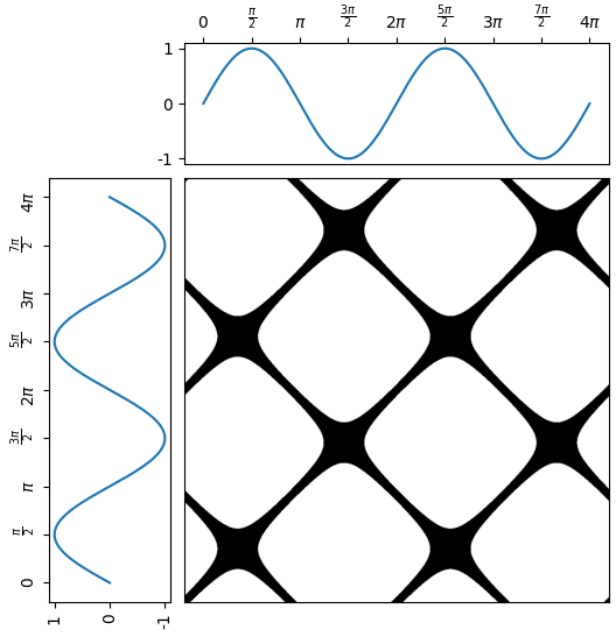
[1] Mohammadi Foumani, Navid, et al. "Deep learning for time series classification and extrinsic regression: A current survey." *ACM Computing Surveys* 56.9 (2024): 1-45.

Images fom https://pyts.readthedocs.io/

2D CNNs for Time Series

Beyond 1D CNN, traditional 2D CNNs can also be used with time series data by applying **imaging techniques** to the raw time series data [1]:

- Gramian Angular Fields
- Markov Transition Field
- Recurrence Plots
- …



[1] Mohammadi Foumani, Navid, et al. "Deep learning for time series classification and extrinsic regression: A current survey." *ACM Computing Surveys* 56.9 (2024): 1-45.

Images fom https://pyts.readthedocs.io/

2D CNNs for Time Series

Beyond 1D CNN, traditional 2D CNNs can also be used with time series data by applying **imaging techniques** to the raw time series data [1]:

- Gramian Angular Fields
- Markov Transition Field
- Recurrence Plots
- ...

This approach requires choosing an appropriate imaging technique for the given data and task (univariate vs multivariate ...).

[1] Mohammadi Foumani, Navid, et al. "Deep learning for time series classification and extrinsic regression: A current survey." *ACM Computing Surveys* 56.9 (2024): 1-45.

Images fom https://pyts.readthedocs.io/

Deep Learning for Time Series – Convolutional Models
Convolutional-based Neural Networks (CNN-based models)

CNN-based Models

CNN-based models can use the CNN architecture and adapt it in different ways, for increased performance gains.

1) Temporality

2) Combination with other architectures:
    a) Recurrent neural networks
    b) Attention mechanisms

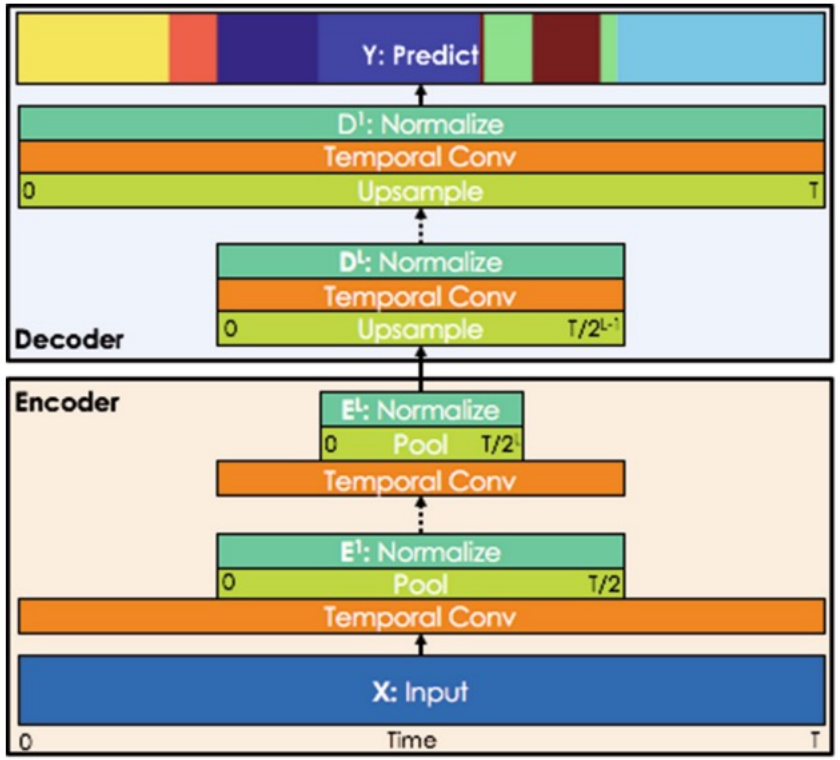
Temporal Convolutional Networks (TCNs)

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.

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

→ There is no information leakage from the future

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.

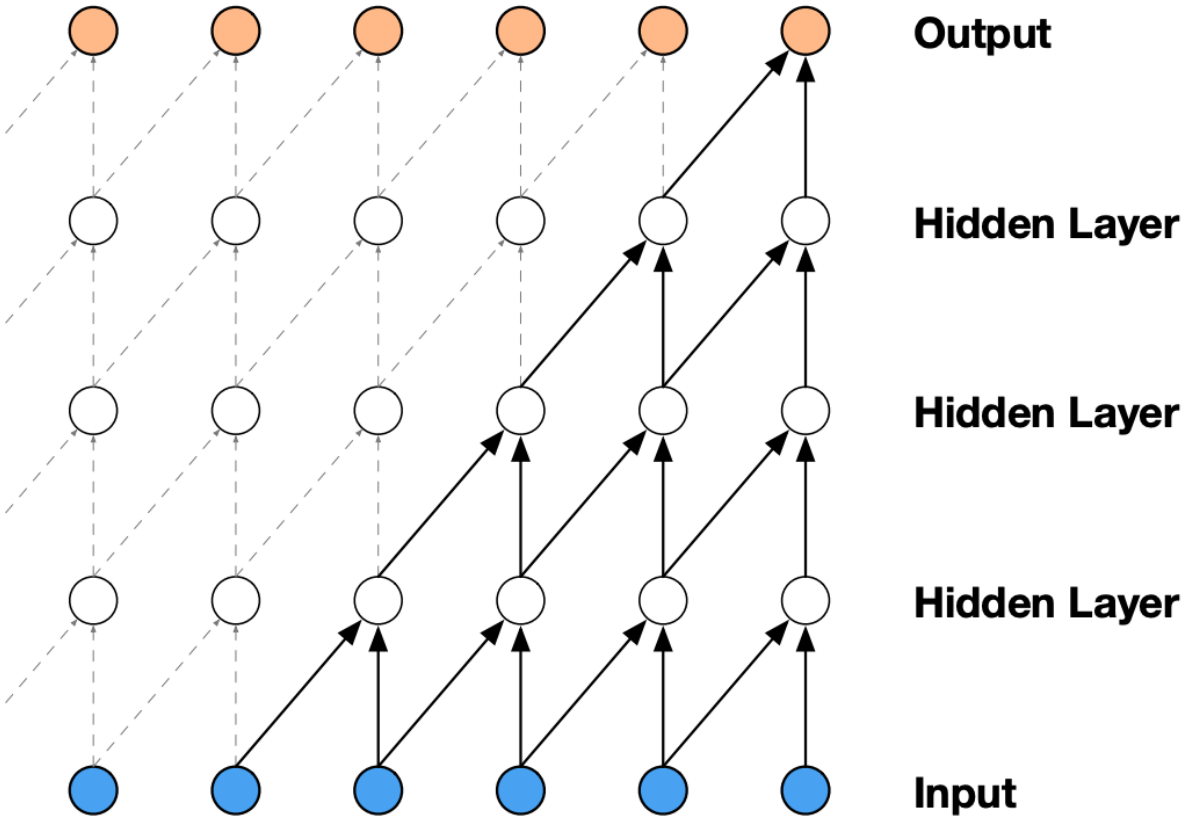


Image from Oord, A. V. D. (2016). WaveNet: A Generative Model for Raw Audio..

Applications of 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.

Deep Learning for Time Series – Convolutional Models
Recap

In this lecture

- The convolution operation
- Convolutional neural networks
- Convolutional neural networks for time series
- Convolutional neural networks and time series imaging
- Convolutional-based architectures