Advanced Deep Learning

DATA.ML.230

Processing videos

Processing videos

A video is a sequence of images
4D tensor: T x 3 x H x W
(or 3 x T x H x W)

Image

Image vs. video classification

Images: Recognize objects

Dog
Cat
Fish
Truck

Videos: Recognize actions

Swimming
Running
Jumping
Eating
Standing

Video

Videos are big!

Input video:
T x 3 x H x W

Videos are ~30 frames per second (fps)

Size of uncompressed video
(3 bytes per pixel):

SD (640 x 480): ~1.5 GB per minute
HD (1920 x 1080): ~10 GB per minute

Videos are big!

Input video:
T x 3 x H x W

Videos are ~30 frames per second (fps)

Size of uncompressed video
(3 bytes per pixel):

SD (640 x 480): ~1.5 GB per minute
HD (1920 x 1080): ~10 GB per minute

Solution: Train on short clips: low fps and low spatial resolution
e.g. T = 16, H=W=112
(3.2 seconds at 5 fps, 588 KB)

Training on clips

Raw video: Long, high FPS

Training on clips

Raw video: Long, high FPS

Training: Train model to classify short clips with low FPS

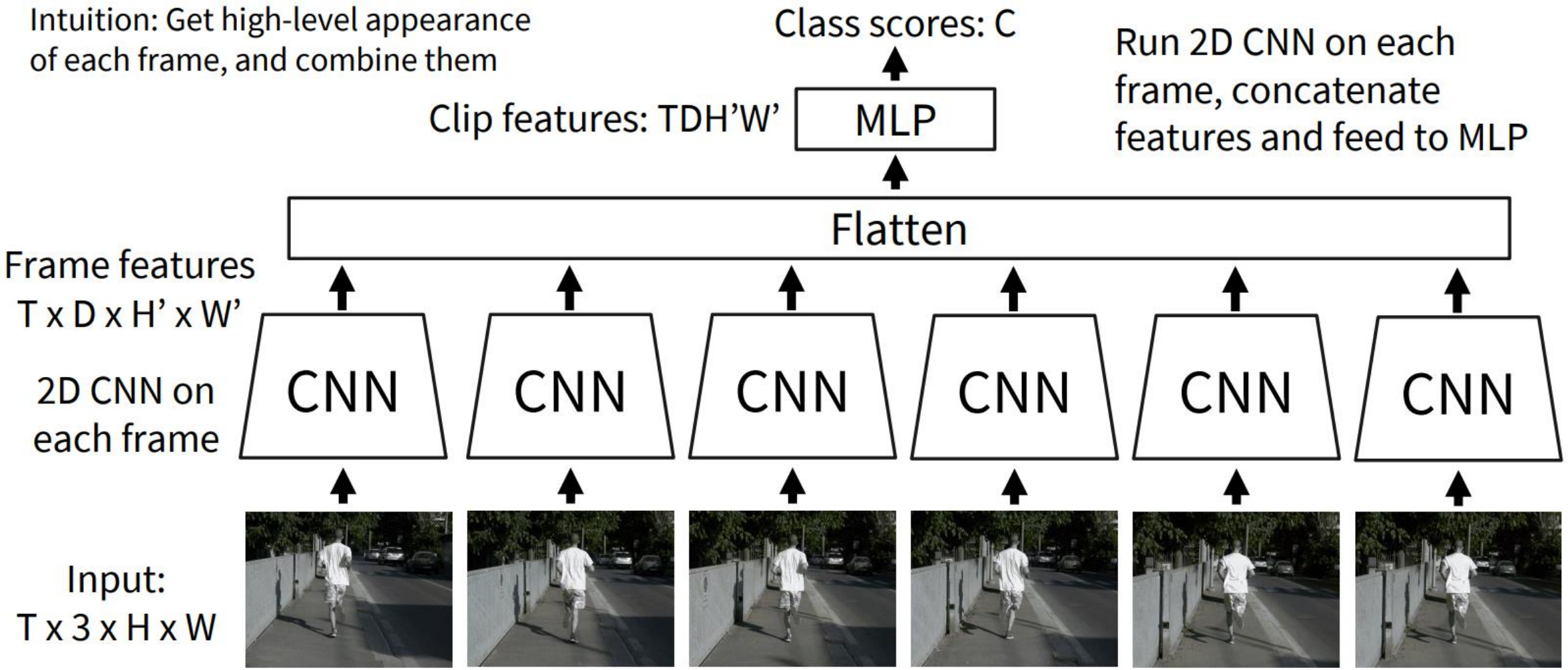
Training on clips

Raw video: Long, high FPS

Training: Train model to classify short clips with low FPS

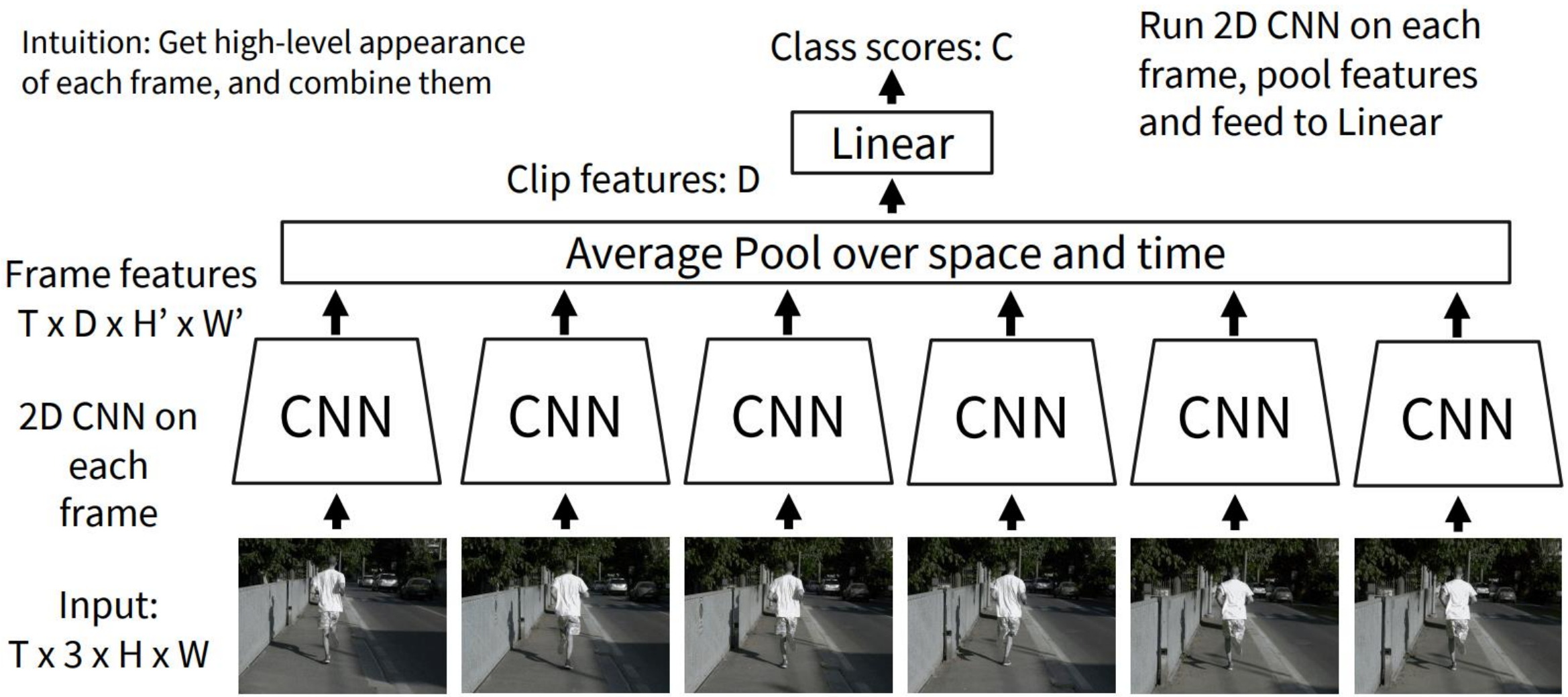
Testing: Run model on different clips, average predictions

Video classification: late fusion



Karpathy et al, "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014

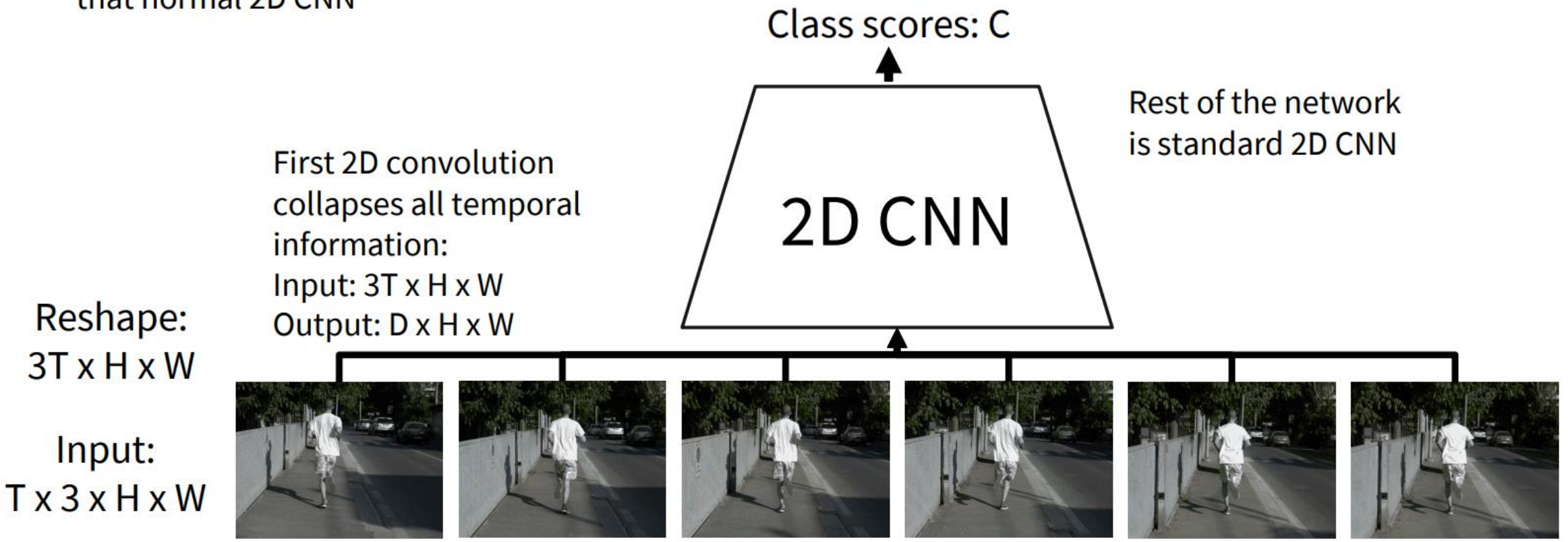
Video classification: late fusion



Karpathy et al, "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014

Video classification: early fusion

Intuition: Compare frames with very first conv layer, after that normal 2D CNN

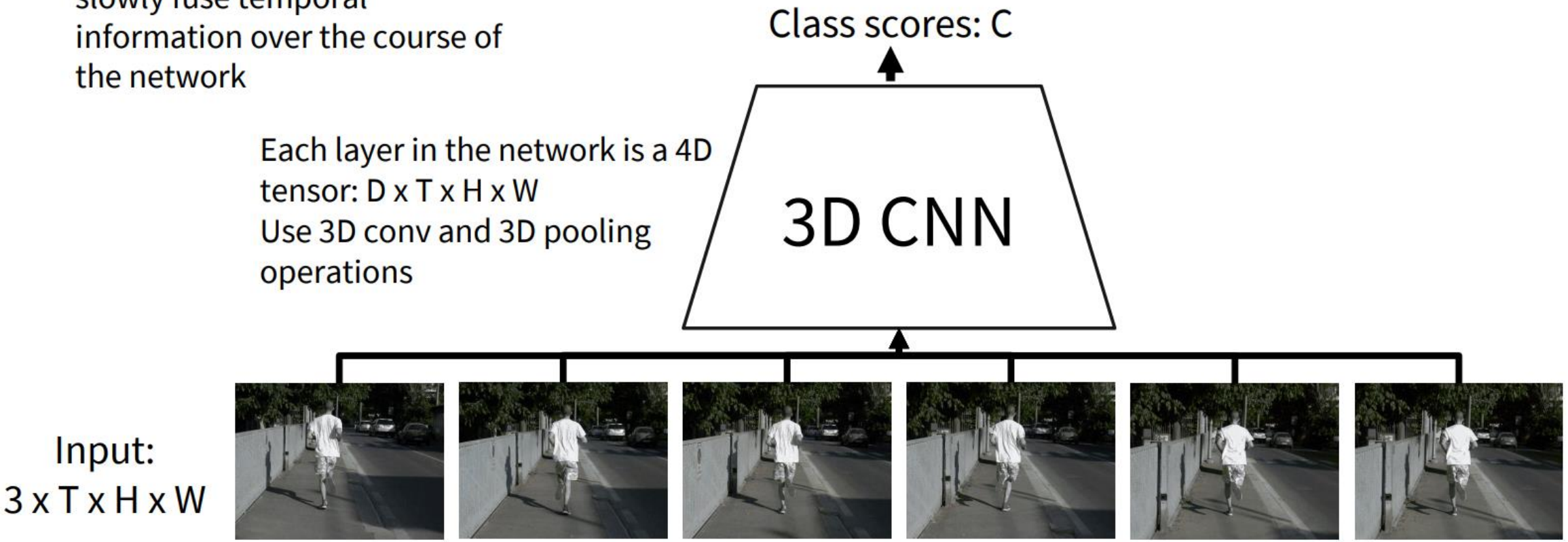


Karpathy et al, “Large-scale Video Classification with Convolutional Neural Networks”, CVPR 2014

Video classification: 3D CNN

Intuition: Use 3D versions of convolution and pooling to slowly fuse temporal information over the course of the network

Each layer in the network is a 4D tensor: D x T x H x W
Use 3D conv and 3D pooling operations

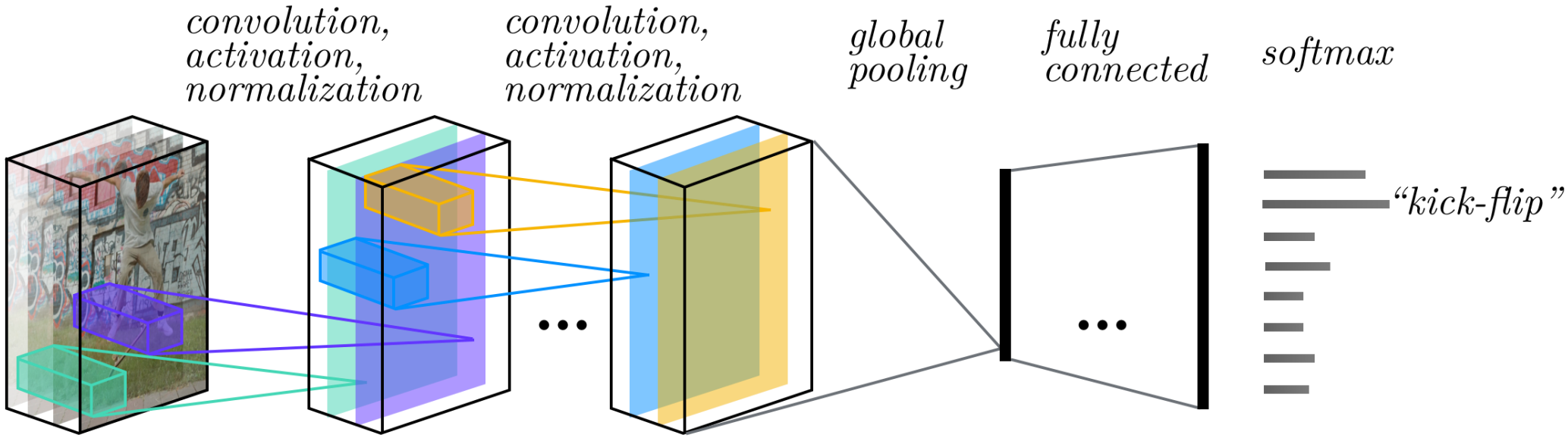


Karpathy et al, “Large-scale Video Classification with Convolutional Neural Networks”, CVPR 2014

3D CNN

3D-extension of 2D CNNs (for image recognition)

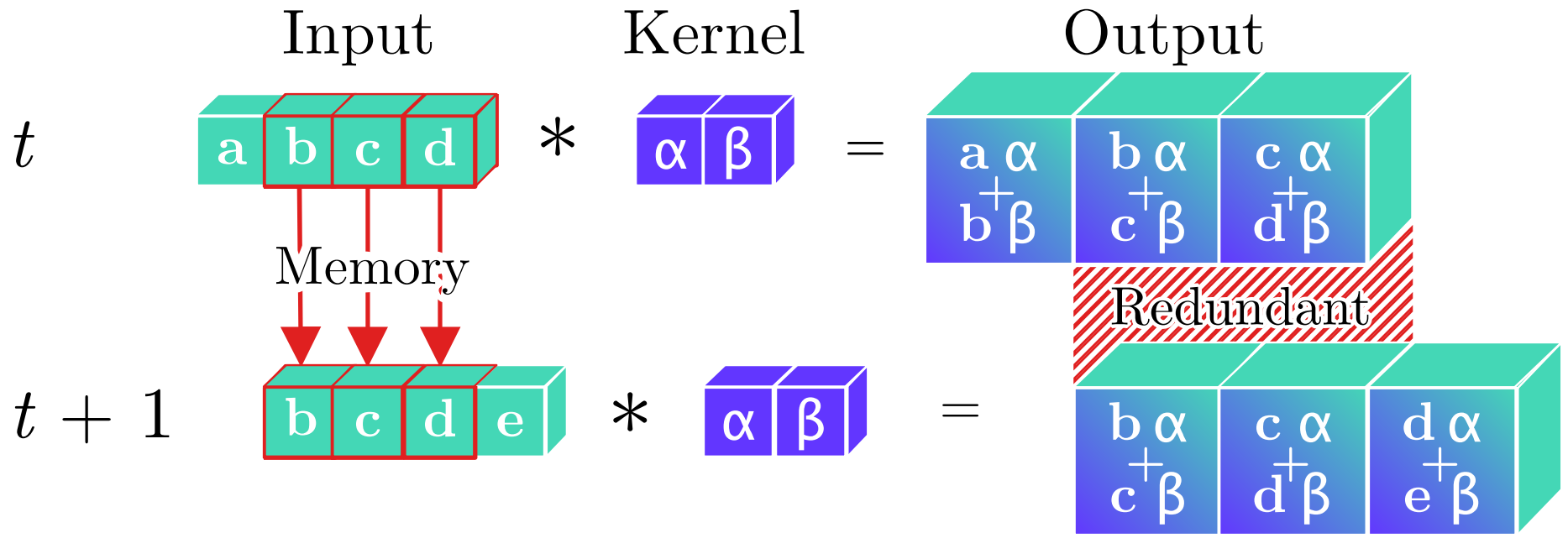
- Time-series data
- Volumetric data



Taylor et al., *"Convolutional learning of spatio-temporal features"*, 2010 (ECCV)
Tran et al., *"Learning Spatiotemporal Features with 3D Convolutional Networks"*, 2015 (ICCV)
Tran et al., *"Long-term temporal convolutions for action recognition"*, 2017 (TPAMI)
Carreira & Zisserman, *"Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset"*, 2018 (CVPR)
Tran et al., "A Closer Look at Spatiotemporal Convolutions for Action Recognition", 2018 (CVPR)
Feichtenhofer et al., *"SlowFast Networks for Video Recognition"*, 2019 (ICCV)
Feichtenhofer, *"X3D: Expanding Architectures for Efficient Video Recognition"*, 2020 (CVPR)

3D CNN

Regular temporal convolution

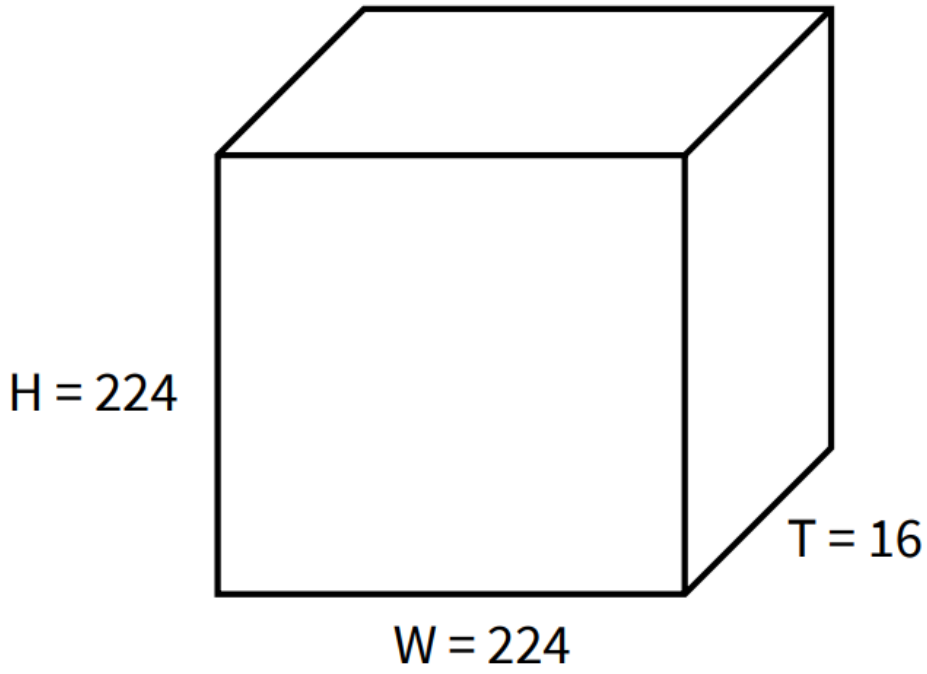


3D Conv:

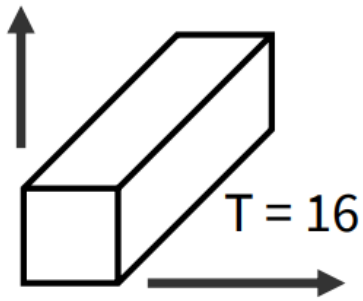
$$(W * X)^{(t,h,w)} \stackrel{\text{def}}{=} \sum_{k_t=0}^{K_T-1} \sum_{k_h=0}^{K_H-1} \sum_{k_w=0}^{K_W-1} W^{(k_t,k_h,k_w)} X^{(t-k_t,h-k_h,w-k_w)}$$

2D Convolution vs. 3D Convolution

Input: C_{in} x T x H x W
(3D grid with C_{in}-dim feat at each point)

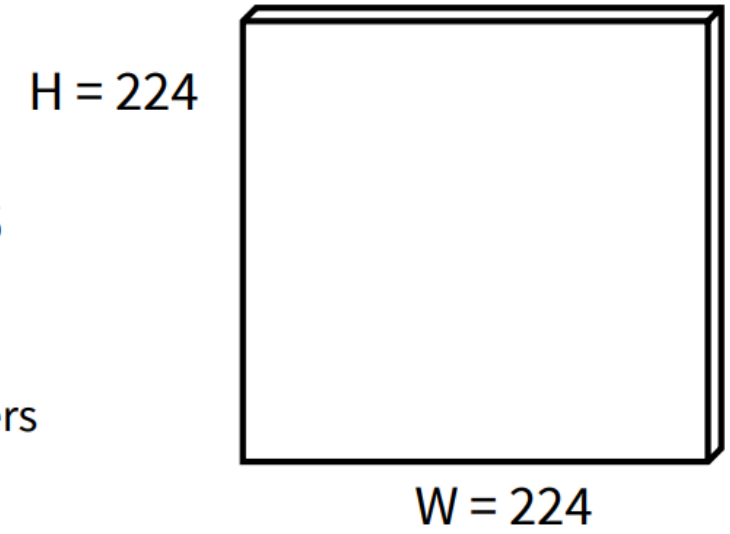


Weight:
C_{out} x C_{in} x T x 3 x 3
Slide over x and y



C_{out} different filters

Output:
C_{out} x H x W
2D grid with C_{out} – dim feat at each point



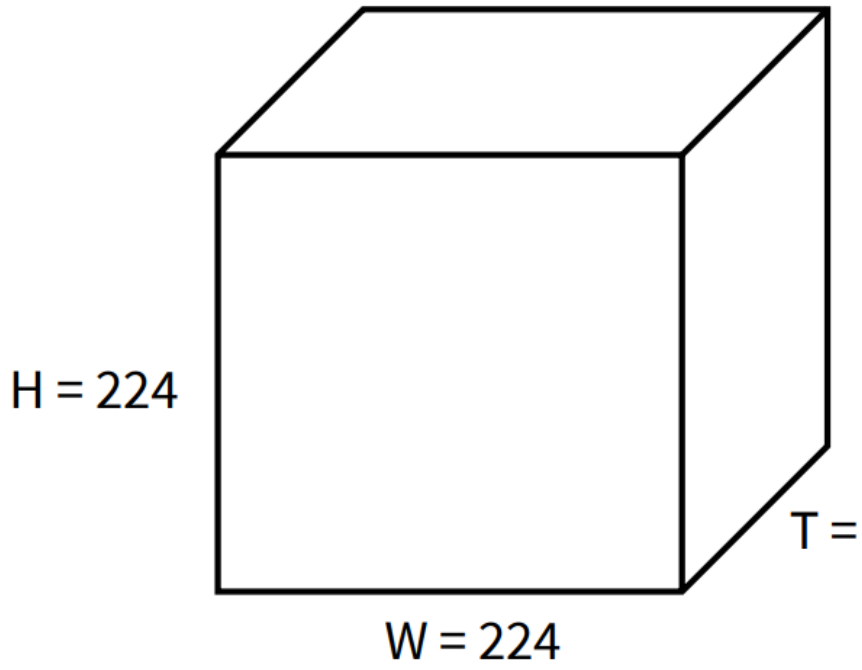
2D Convolution vs. 3D Convolution

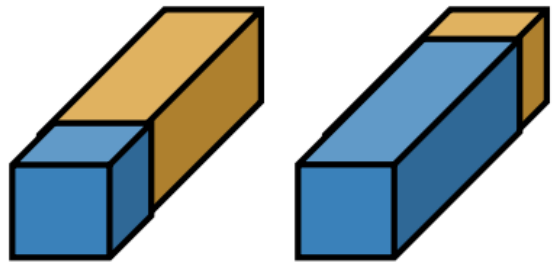
Input: C_{in} x T x H x W
(3D grid with C_{in}-dim feat at each point)

Weight:
C_{out} x C_{in} x T x 3 x 3
Slide over x and y

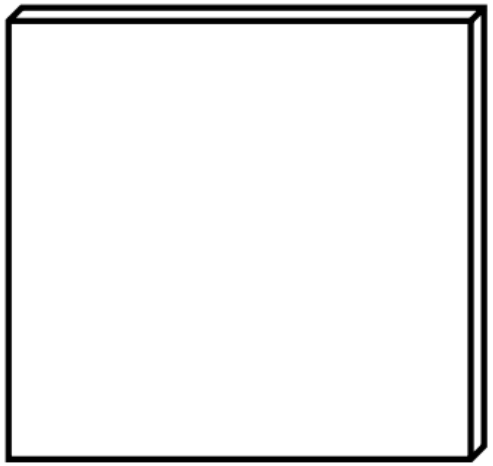
No temporal shift-invariance!
Needs to learn separate filters for the same motion at different times in the clip

Output:
C_{out} x H x W
2D grid with C_{out} -dim feat at each point





C_{out} different filters



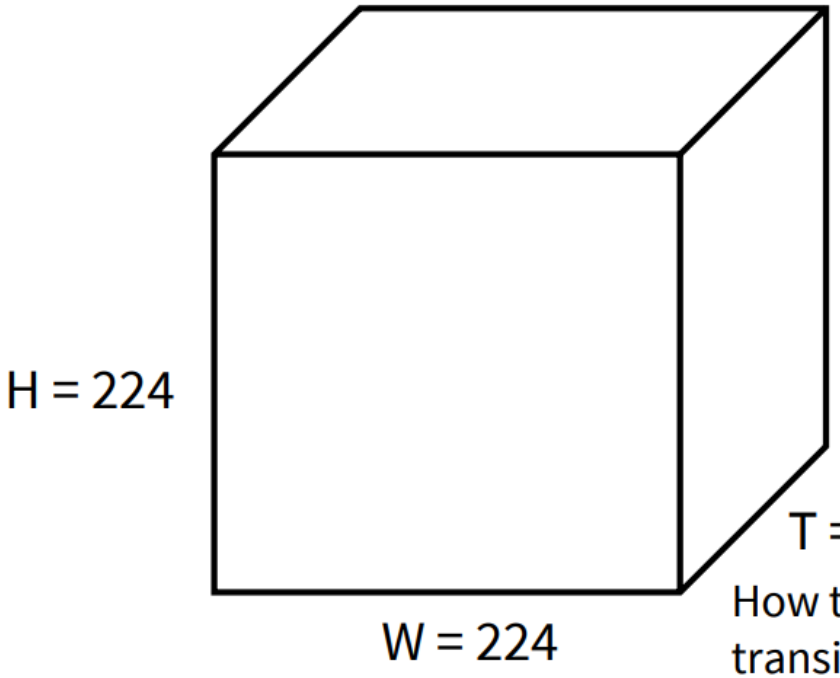
2D Convolution vs. 3D Convolution

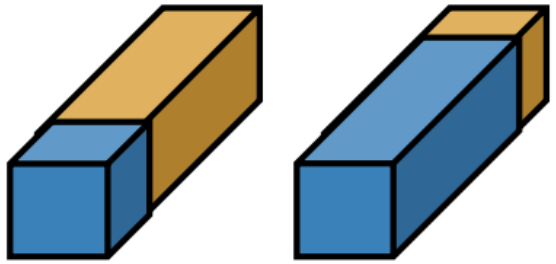
Input: C_{in} x T x H x W
(3D grid with C_{in}-dim feat at each point)

Weight:
C_{out} x C_{in} x T x 3 x 3
Slide over x and y

Output:
C_{out} x H x W
2D grid with C_{out} –dim feat at each point

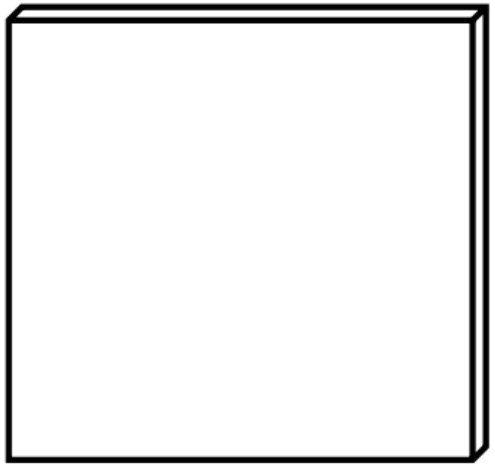
No temporal shift-invariance!
Needs to learn separate filters for the same motion at different times in the clip





C_{out} different filters

How to recognize blue to orange transitions anywhere in space and time?

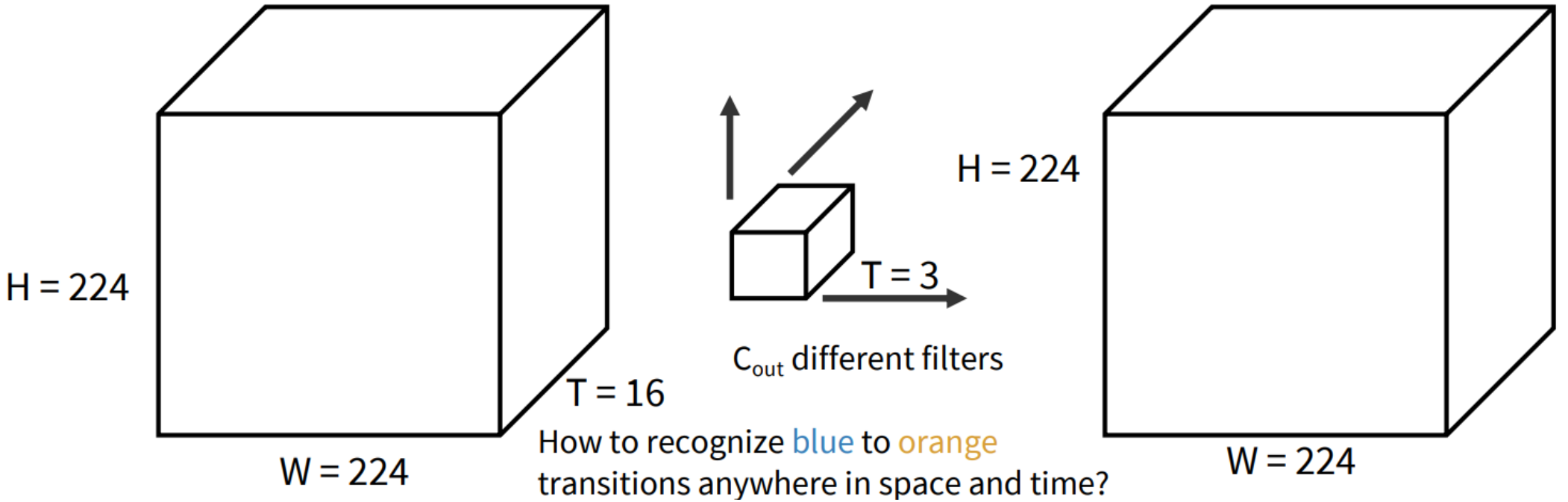


2D Convolution vs. 3D Convolution

Input: C_{in} x T x H x W
(3D grid with C_{in}-dim feat at each point)

Weight:
C_{out} x C_{in} x 3 x 3 x 3
Slide over x and y

Output:
C_{out} x T x H x W
3D grid with C_{out}–dim feat at each point



C_{out} different filters

How to recognize blue to orange transitions anywhere in space and time?

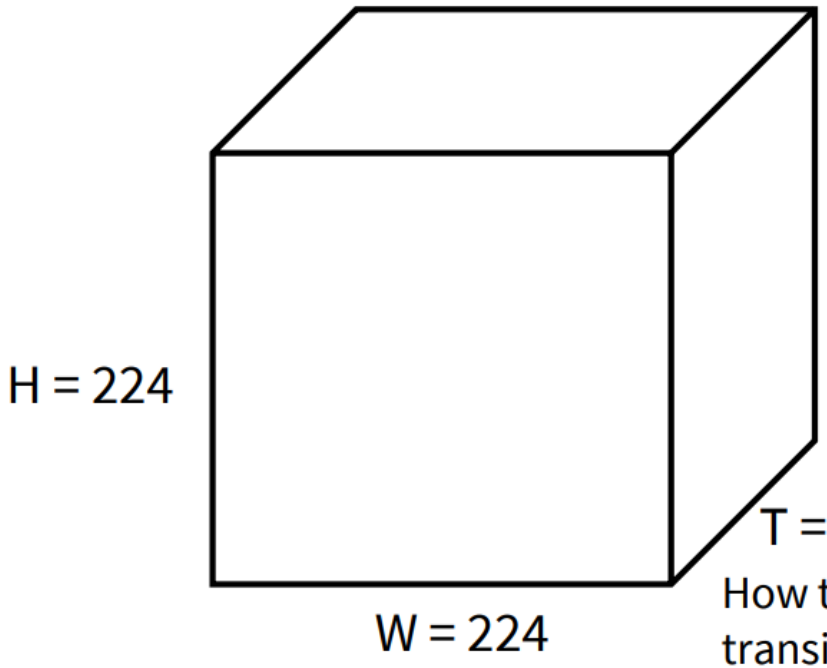
2D Convolution vs. 3D Convolution

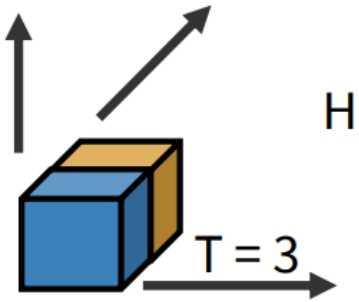
Input: C_{in} x T x H x W
(3D grid with C_{in}-dim feat at each point)

Weight:
C_{out} x C_{in} x 3 x 3 x 3
Slide over x and y

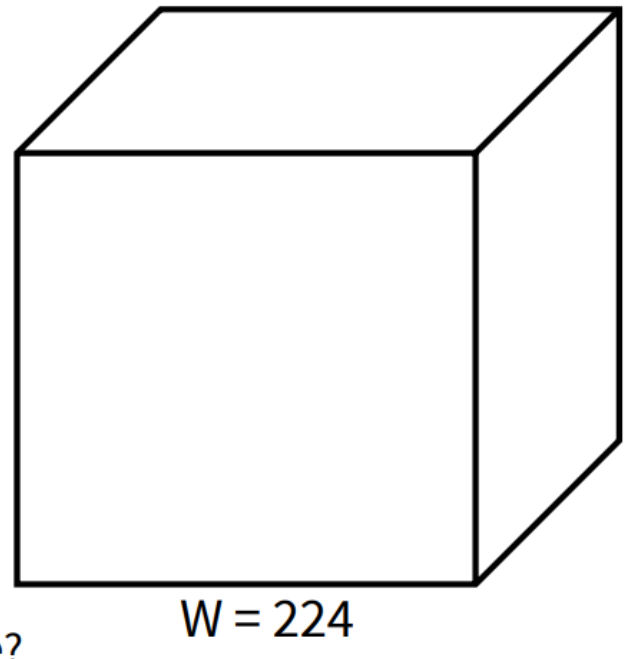
Output:
C_{out} x T x H x W
3D grid with C_{out}–dim feat at each point

Temporal shift-invariant since each filter slides over time!





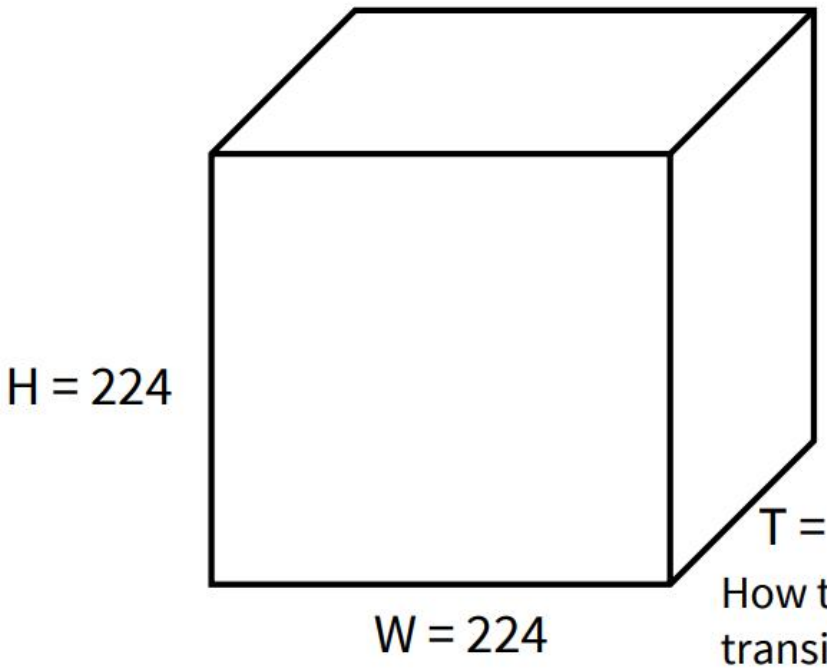
C_{out} different filters



How to recognize blue to orange transitions anywhere in space and time?

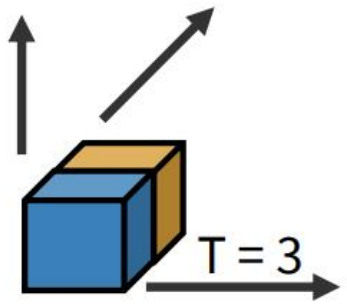
2D Convolution vs. 3D Convolution

Input: C_{in} x T x H x W
(3D grid with C_{in}-dim feat at each point)



Weight:
C_{out} x C_{in} x 3 x 3 x 3
Slide over x and y

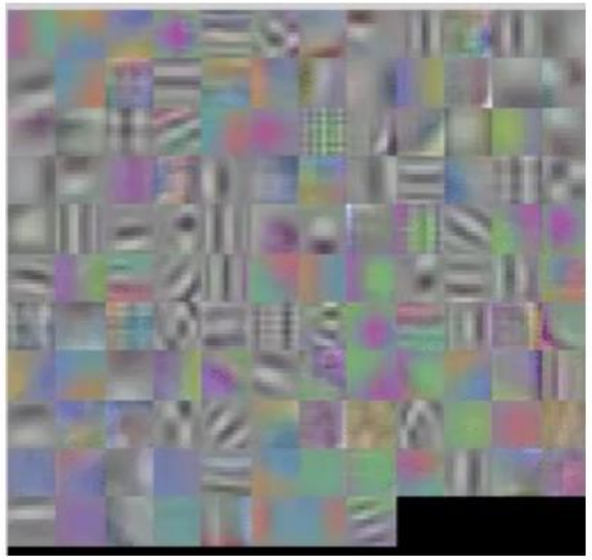
Temporal shift-invariant since each filter slides over time!



C_{out} different filters

How to recognize blue to orange transitions anywhere in space and time?

First-layer filters have shape 3 (RGB) x 4 (frames) x 5 x 5 (space)
Can visualize as video clips!



C3D: The VGG of 3D CNNs

3D CNN that uses all 3x3x3 conv and 2x2x2 pooling
(except Pool1 which is 1x2x2)

Released model pretrained on Sports-1M: Many people used this as a video feature extractor

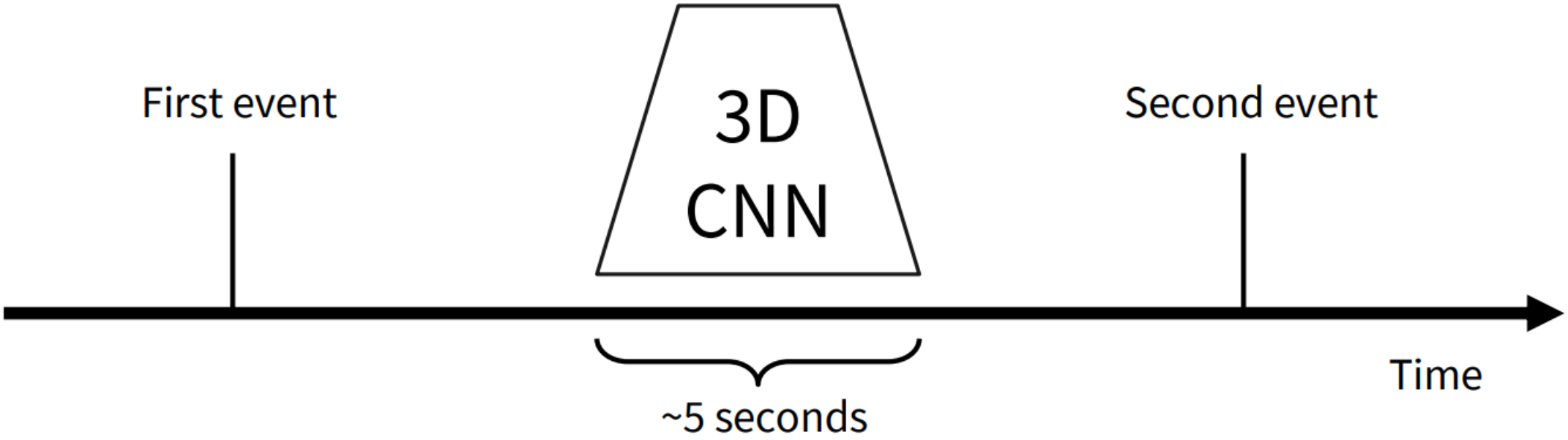
Problem: 3x3x3 conv is very expensive!
AlexNet: 0.7 GFLOP
VGG-16: 13.6 GFLOP
C3D: 39.5 GFLOP (2.9x VGG!)

Layer	Size	MFLOPs
Input	3 x 16 x 112 x 112	
Conv1 (3x3x3)	64 x 16 x 112 x 112	1.04
Pool1 (1x2x2)	64 x 16 x 56 x 56	
Conv2 (3x3x3)	128 x 16 x 56 x 56	11.10
Pool2 (2x2x2)	128 x 8 x 28 x 28	
Conv3a (3x3x3)	256 x 8 x 28 x 28	5.55
Conv3b (3x3x3)	256 x 8 x 28 x 28	11.10
Pool3 (2x2x2)	256 x 4 x 14 x 14	
Conv4a (3x3x3)	512 x 4 x 14 x 14	2.77
Conv4b (3x3x3)	512 x 4 x 14 x 14	5.55
Pool4 (2x2x2)	512 x 2 x 7 x 7	
Conv5a (3x3x3)	512 x 2 x 7 x 7	0.69
Conv5b (3x3x3)	512 x 2 x 7 x 7	0.69
Pool5	512 x 1 x 3 x 3	
FC6	4096	0.51
FC7	4096	0.45
FC8	C	0.05

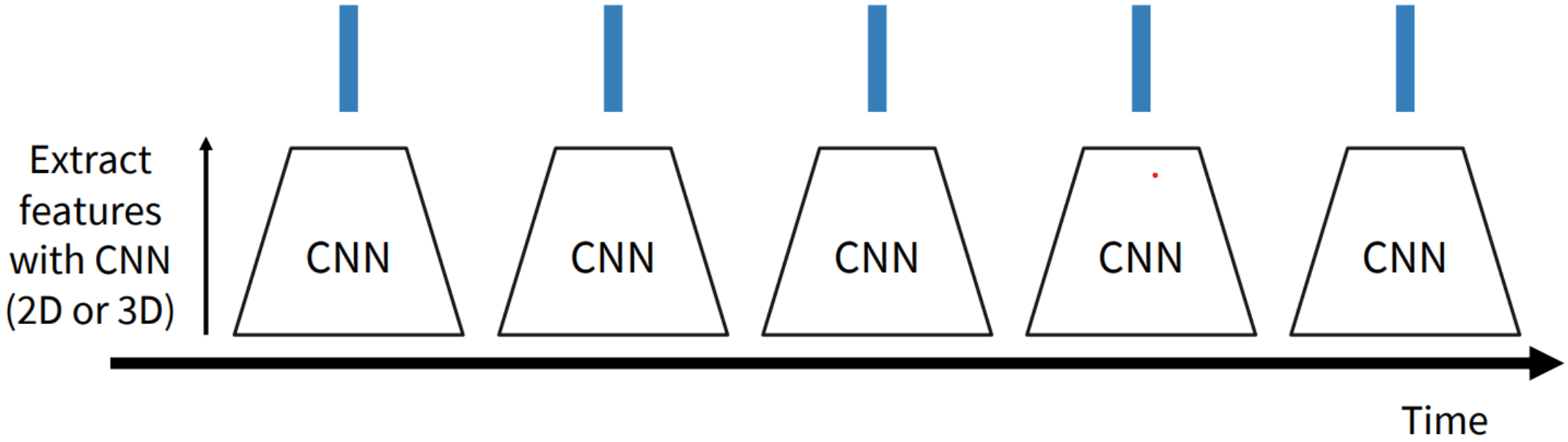
Tran et al, “Learning Spatiotemporal Features with 3D Convolutional Networks”, ICCV 2015

3D CNNs as feature extractors

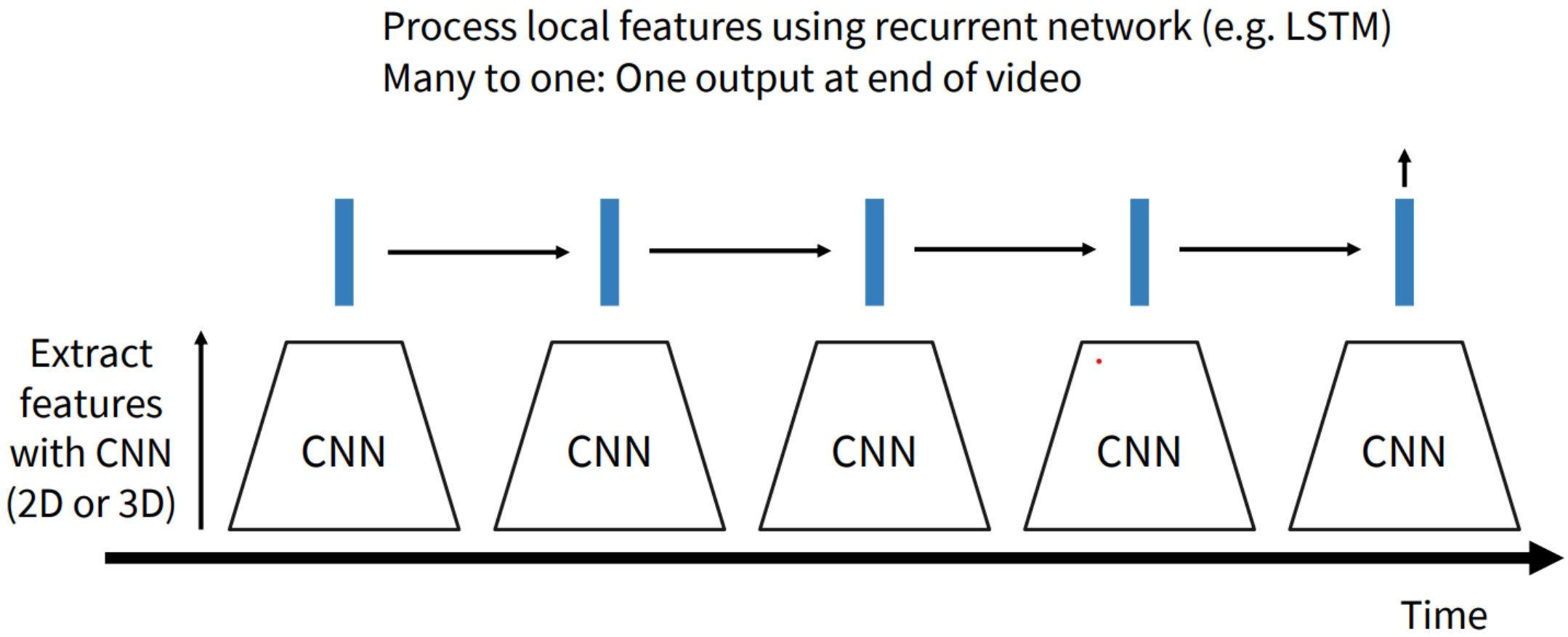
So far all our temporal CNNs only model local motion between frames in very short clips of ~2-5 seconds. What about long-term structure?



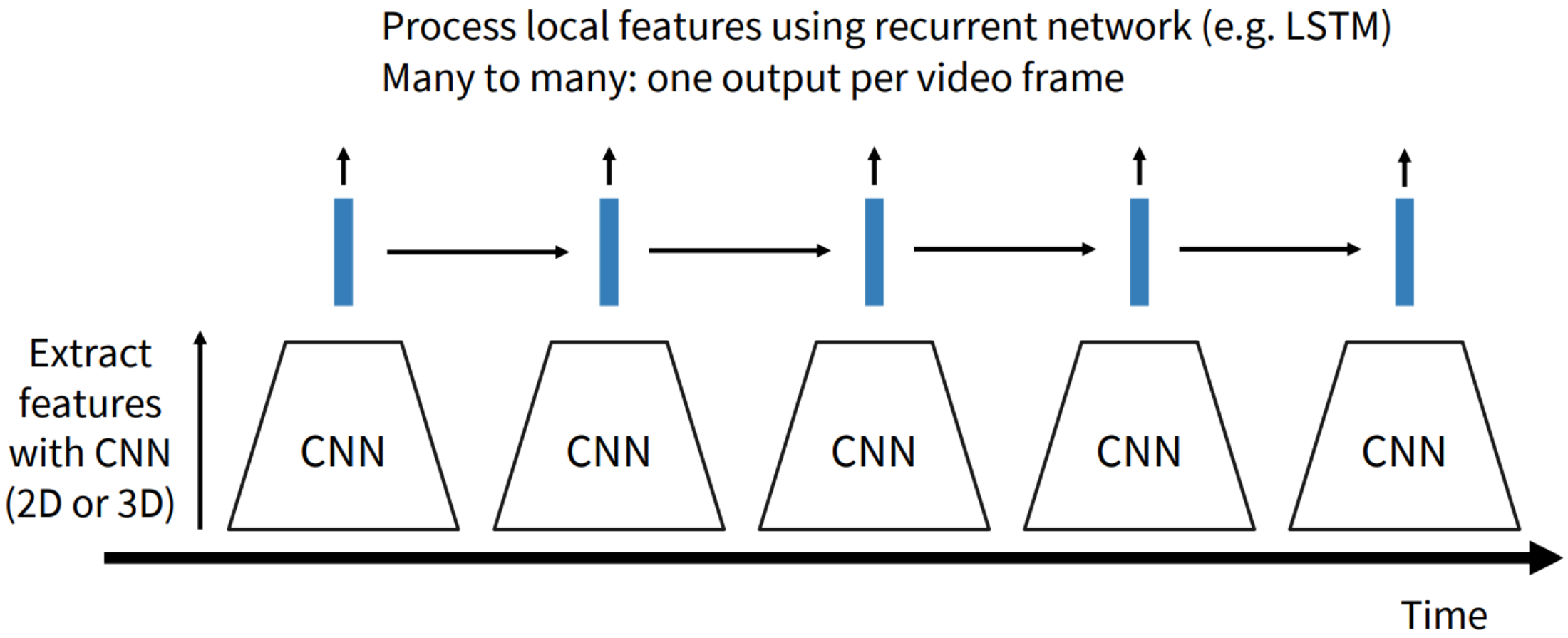
3D CNNs as feature extractors



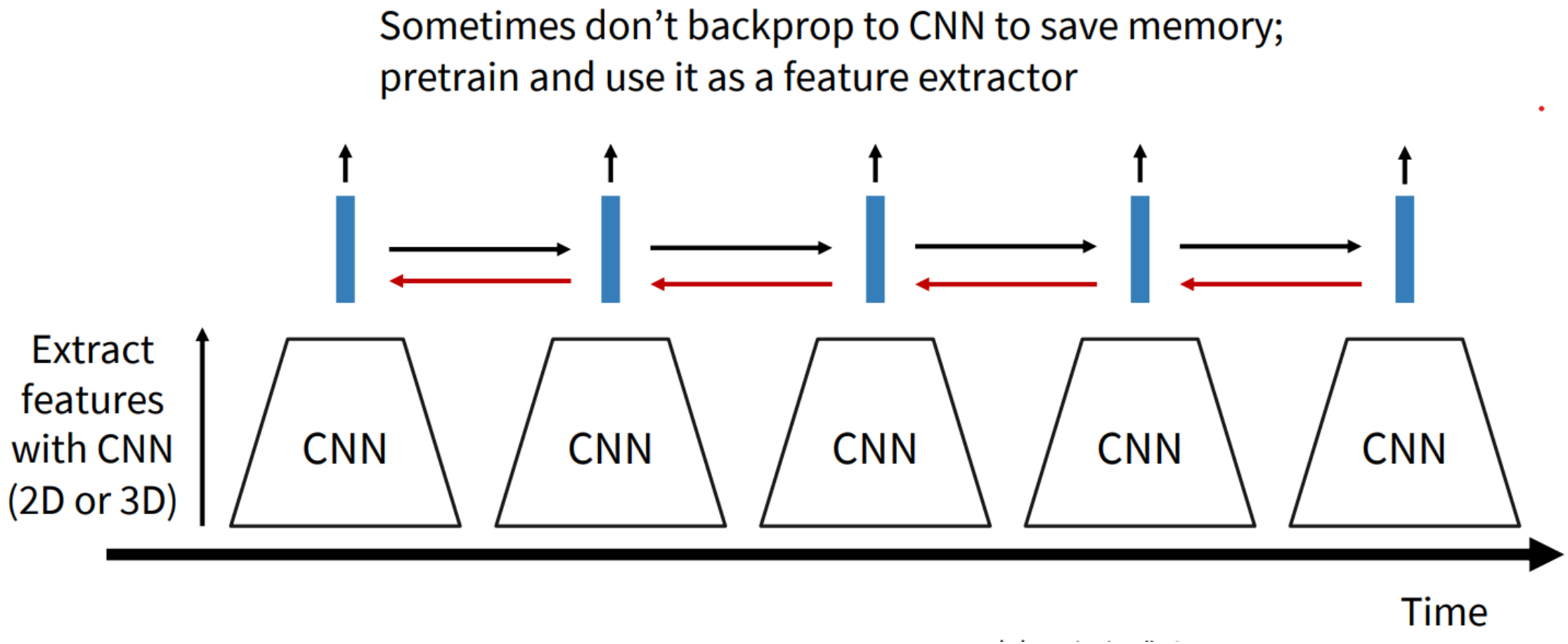
3D CNNs as feature extractors



3D CNNs as feature extractors



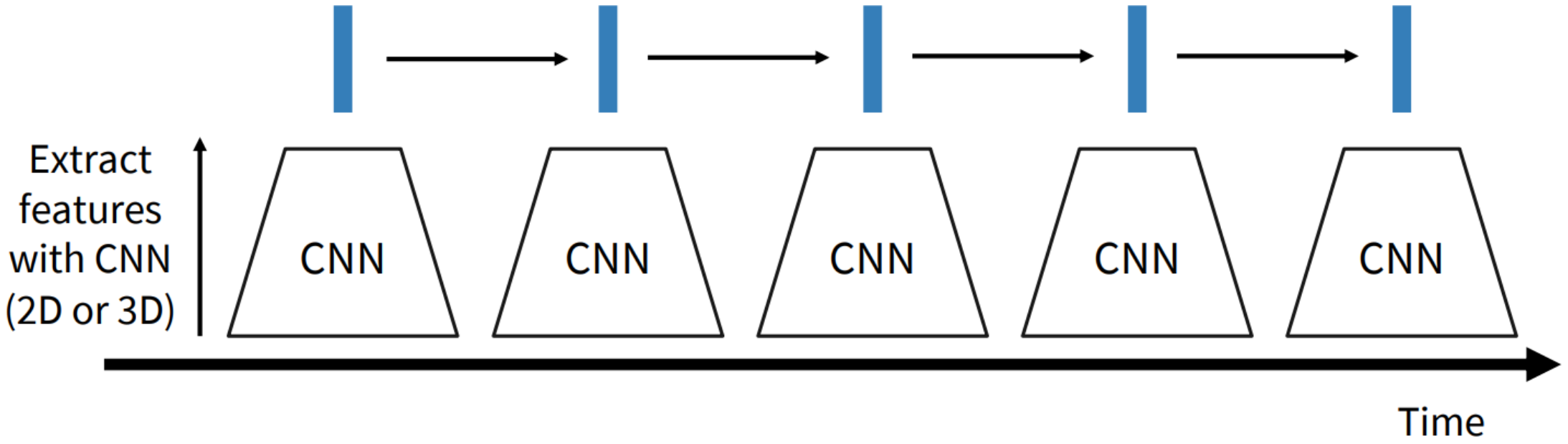
3D CNNs as feature extractors



Baccouche et al, "Sequential Deep Learning for Human Action Recognition", 2011
Donahue et al, "Long-term recurrent convolutional networks for visual recognition and description", CVPR 2015

3D CNNs as feature extractors

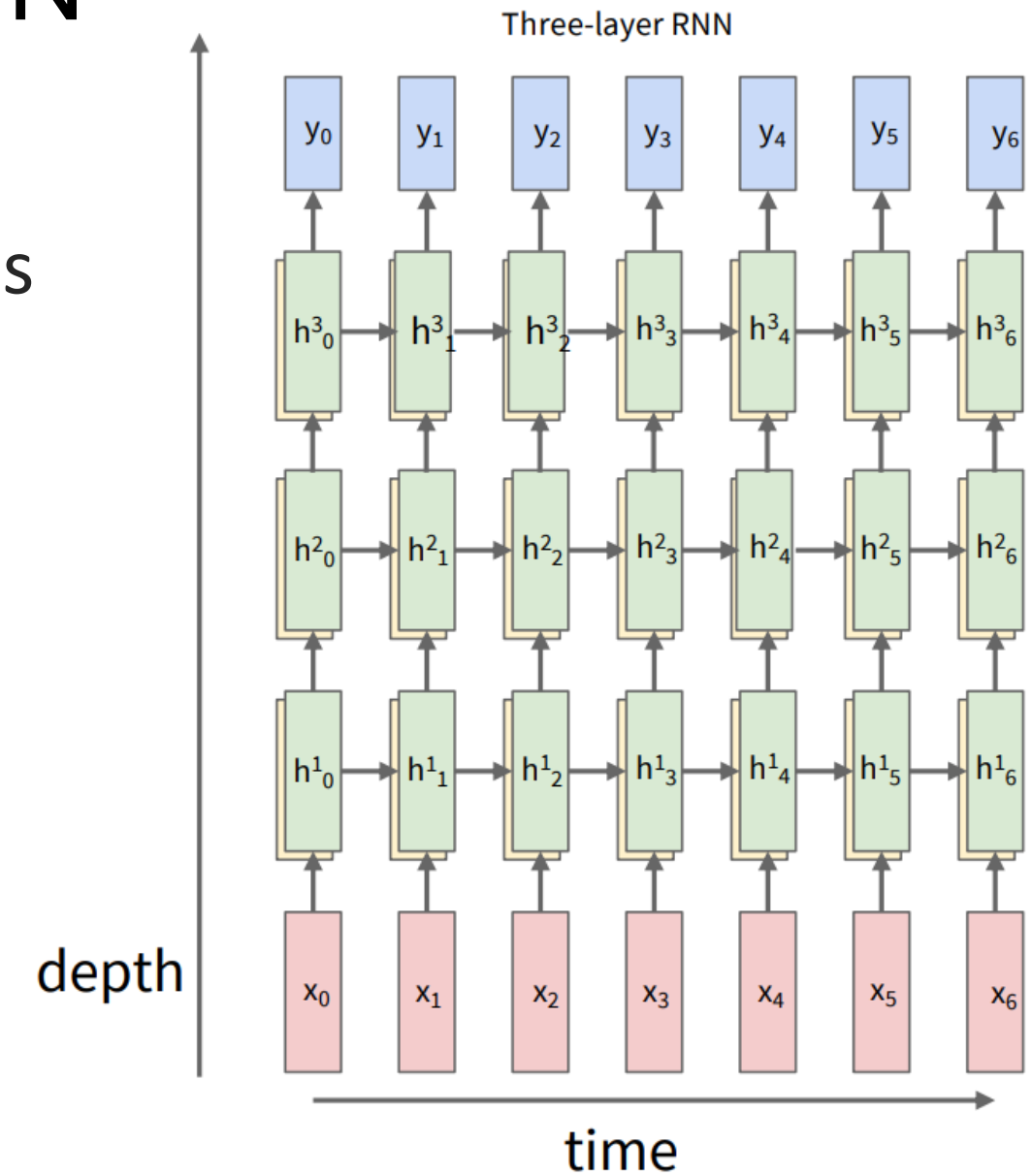
Inside CNN: Each value is a function of a fixed temporal window (local temporal structure)
Inside RNN: Each vector is a function of all previous vectors (global temporal structure)
Can we merge both approaches?



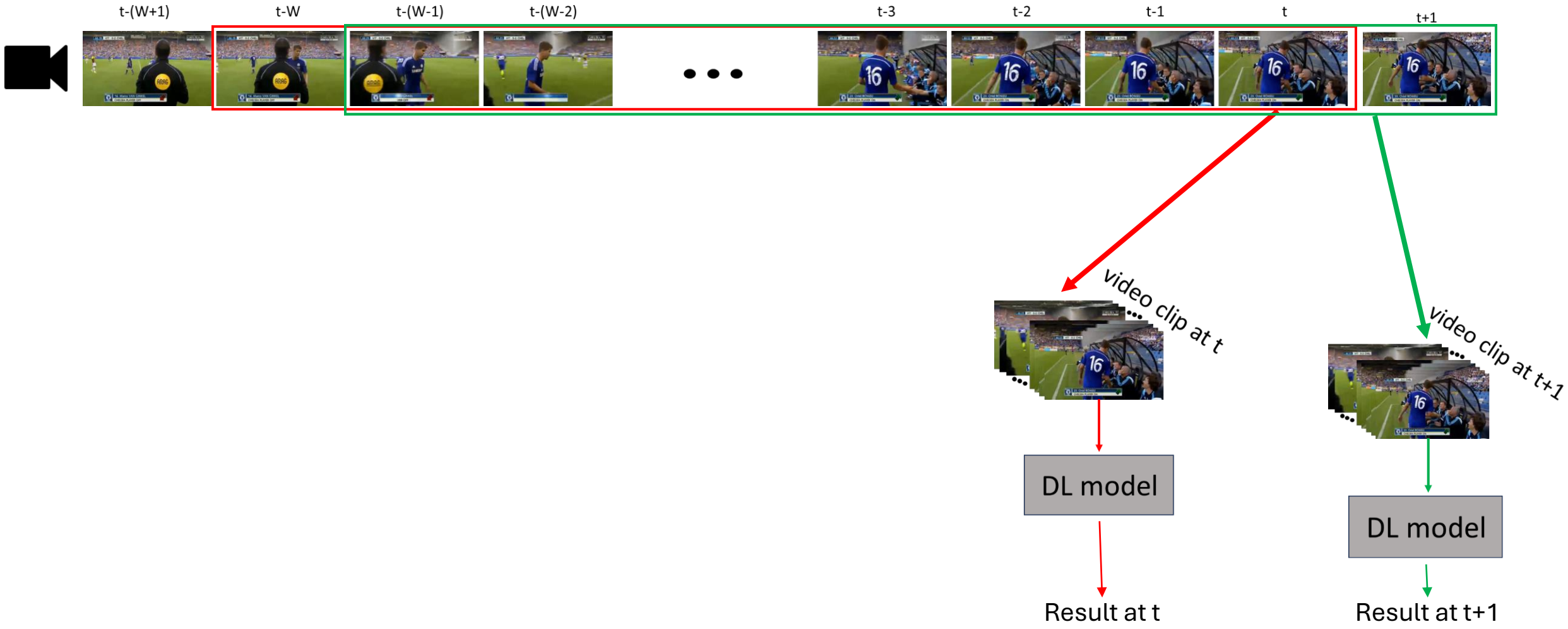
Baccouche et al, "Sequential Deep Learning for Human Action Recognition", 2011
Donahue et al, "Long-term recurrent convolutional networks for visual recognition and description", CVPR 2015

Reminder: Multi-layer RNN

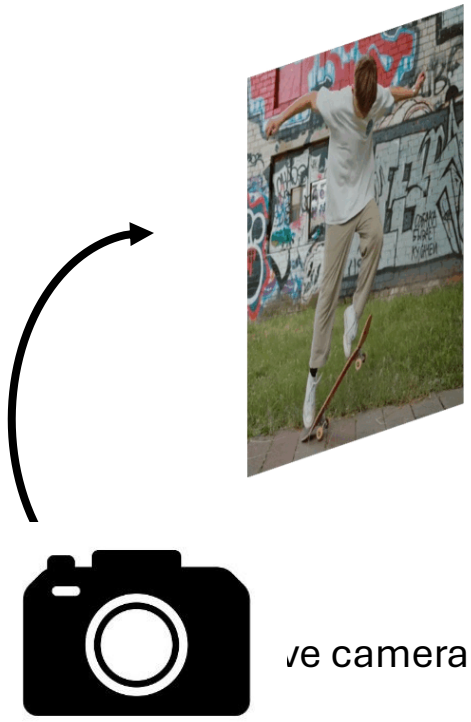
We can use a similar RNN for processing the video representations described above



Online video classification



Continual Inference



Compute features once

Retain the features needed
to compute future elements

ve camera

Input is a *continual* stream

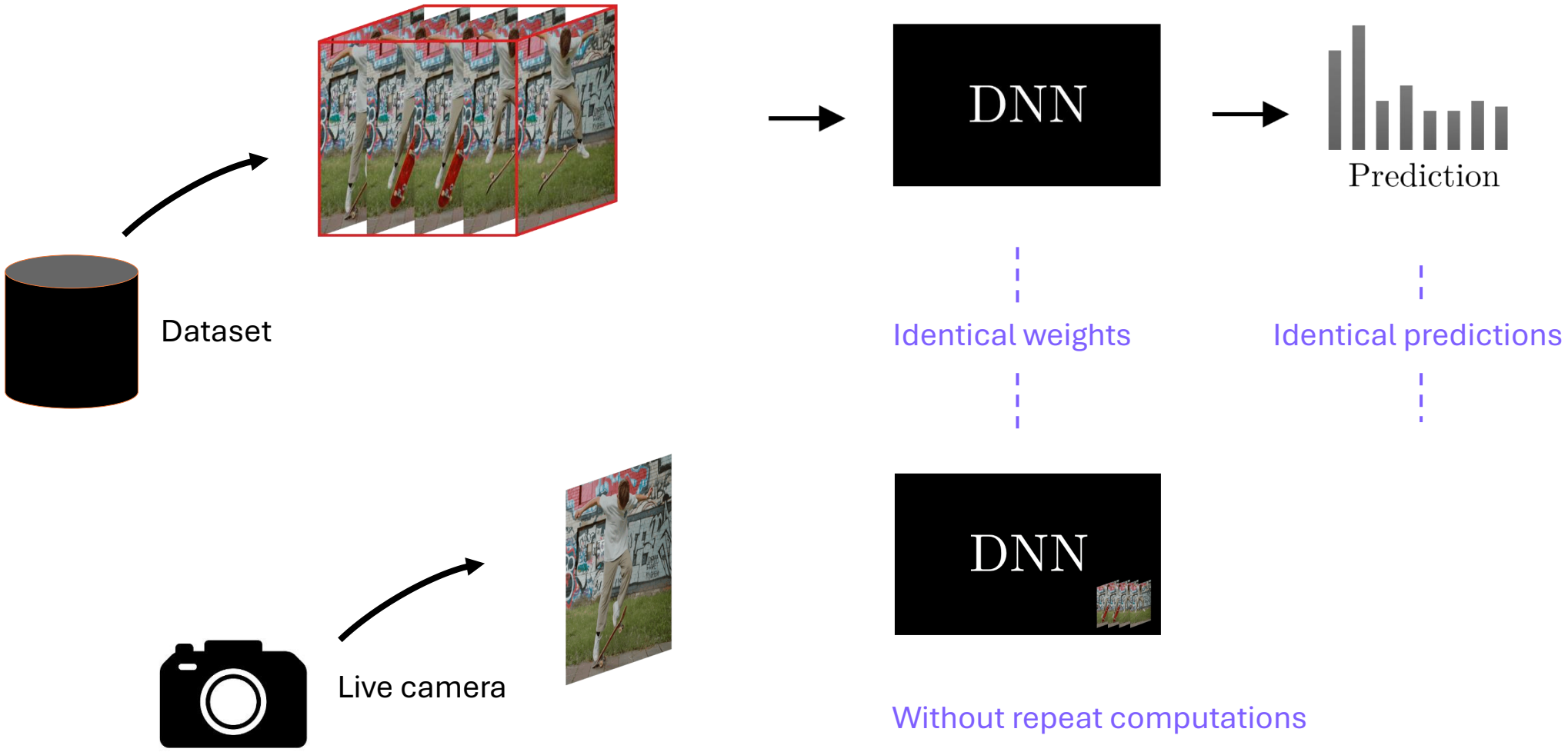
Continual Inference Networks

A deep neural network, which:

- is capable of continual step inference without computational redundancy,
- is capable of batch inference corresponding to a non-continual neural network,
- produces identical outputs for batch- and step inference given identical receptive fields,
- uses one set of trainable parameters for both batch and step inference

Hedegaard et al, ”Continual Inference: A Library for Efficient Online Inference with DNNs in PyTorch”, ECCV-W, 2022

Continual Inference Network

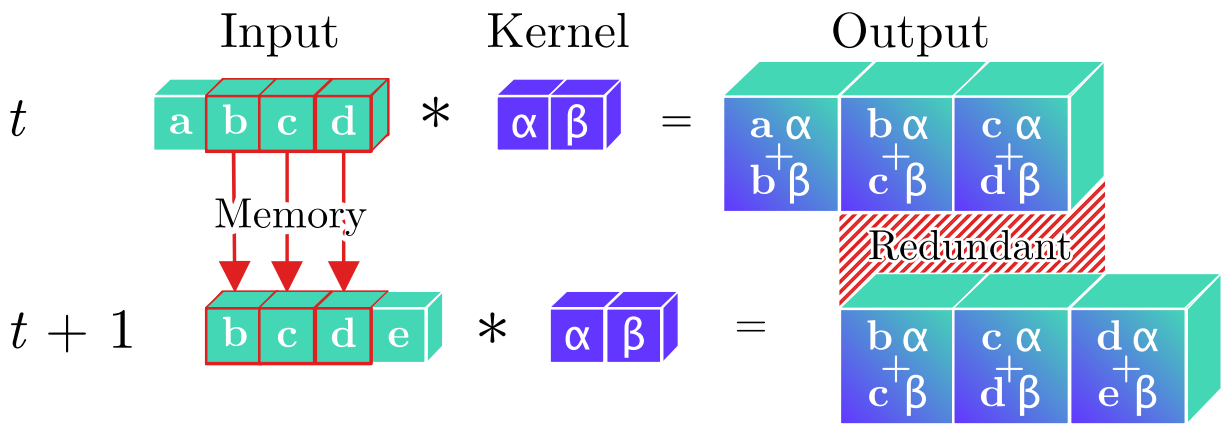


Hedegaard et al, ”Continual Inference: A Library for Efficient Online Inference with DNNs in PyTorch”, ECCV-W, 2022
Hedegaard, Iosifidis, “Continual 3D CNNs for Real-time Processing of Videos”, ECCV 2022

Continual Inference

Standard 3D Convolution:

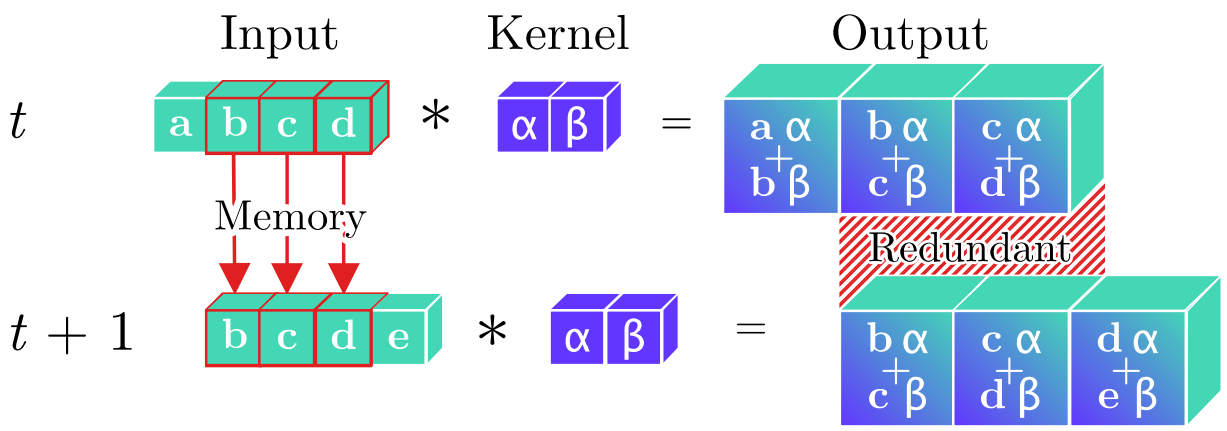
$$(W * X)^{(t,h,w)} \stackrel{\text{def}}{=} \sum_{k_t=0}^{K_T-1} \sum_{k_h=0}^{K_H-1} \sum_{k_w=0}^{K_W-1} W^{(k_t,k_h,k_w)} X^{(t-k_t,h-k_h,w-k_w)}$$



Continual Inference

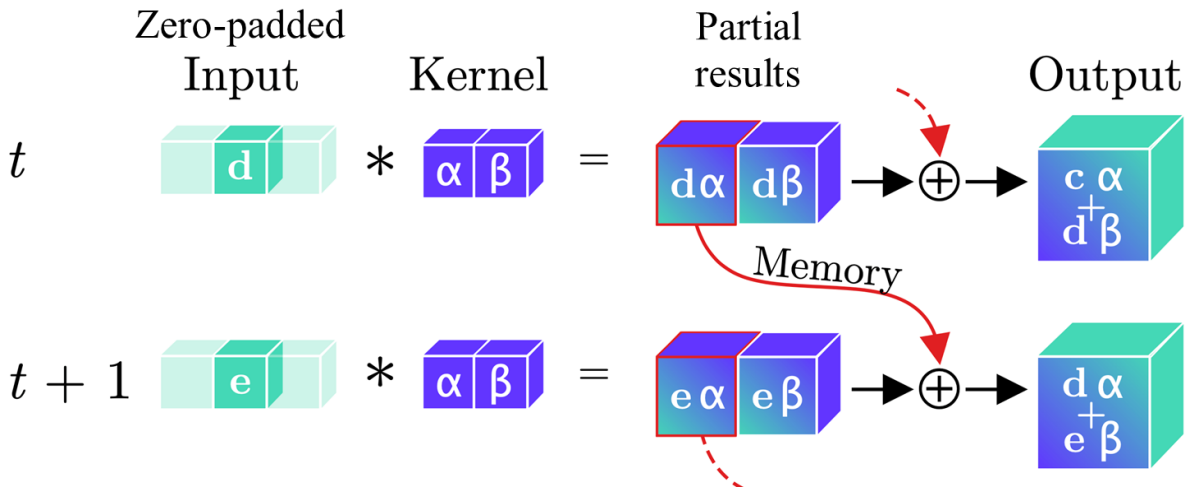
Standard 3D Convolution:

$$(W * X)^{(t,h,w)} \stackrel{\text{def}}{=} \sum_{k_t=0}^{K_T-1} \sum_{k_h=0}^{K_H-1} \sum_{k_w=0}^{K_W-1} W^{(k_t,k_h,k_w)} X^{(t-k_t,h-k_h,w-k_w)}$$



Continual 3D Convolution at time t:

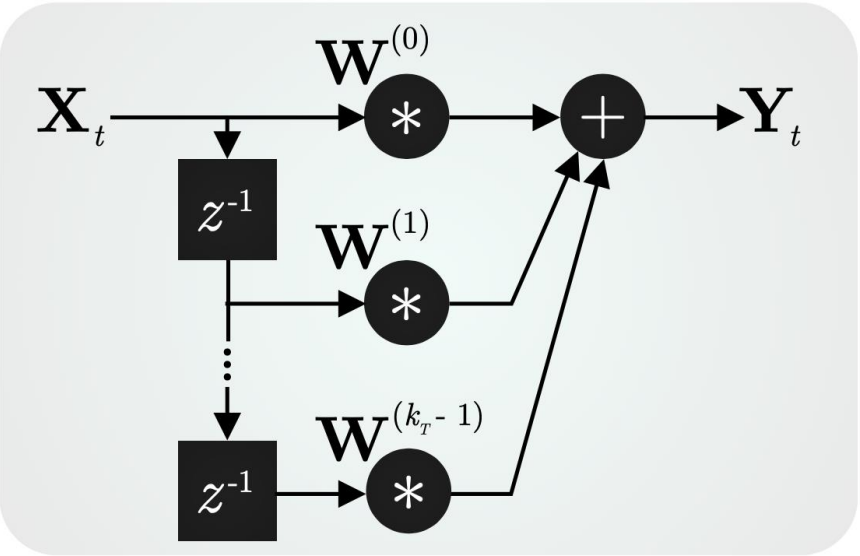
$$(\mathbf{W} * \mathbf{X})^{(t)} = \sum_{k_t=0}^{K_T-1} \mathbf{W}^{(k_t)} * \mathbf{X}^{(t-k_t)}$$



Hedegaard, Iosifidis, “Continual 3D CNNs for Real-time Processing of Videos”, ECCV 2022

Continual temporal convolution

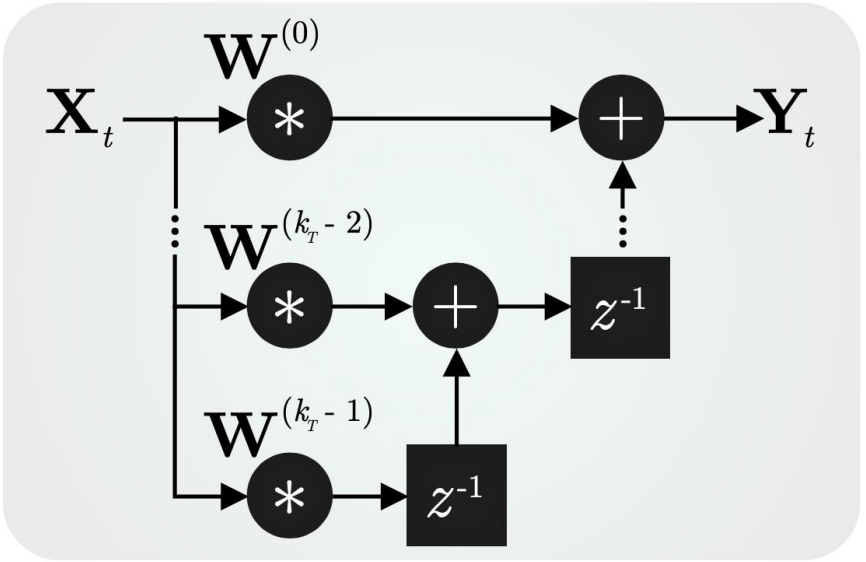
Implementation options



(a) Pre-cached

Direct form FIR-filter

Uses least memory if $C_{\text{in}} < C_{\text{out}}$



(b) Post-cached

Transposed form FIR-filter

Uses least memory if $C_{\text{in}} > C_{\text{out}}$

Continual temporal convolution (Delay)

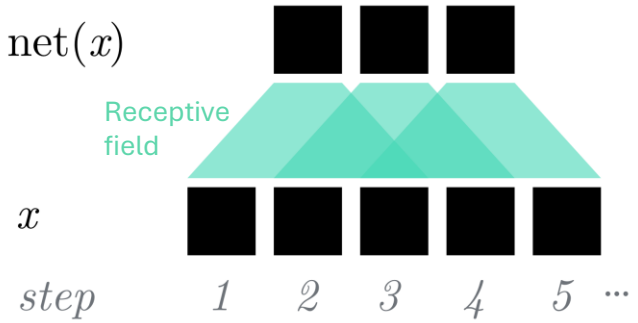
CINs are not allowed to peek into the future

Delay is occasionally needed:

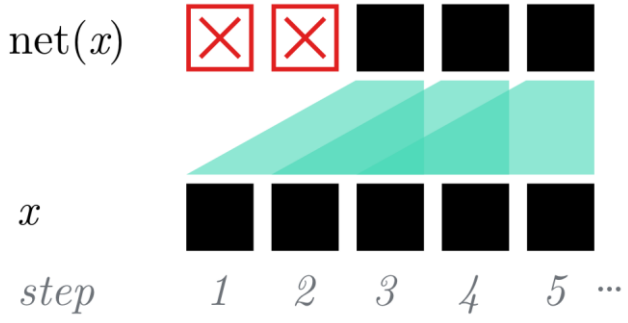
Kernel size Dilation Padding

$$\underbrace{K_T + (K_T - 1)(D_T - 1)}_{\text{Receptive field}} - P_T - 1$$

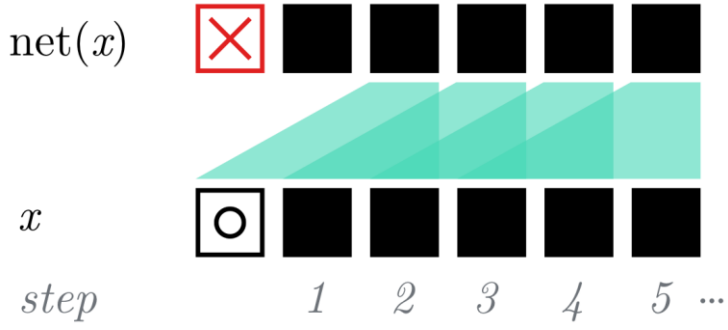
Offline processing



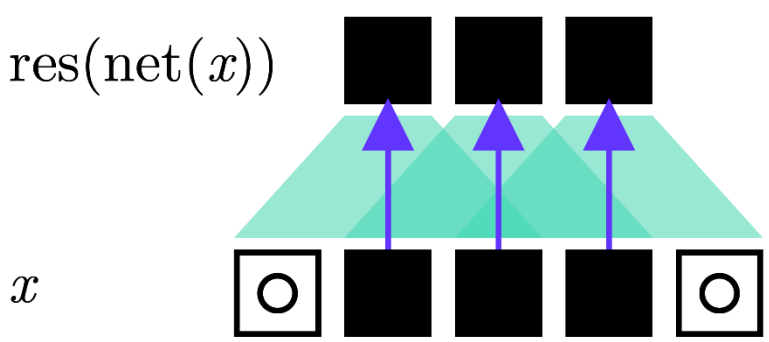
Online processing



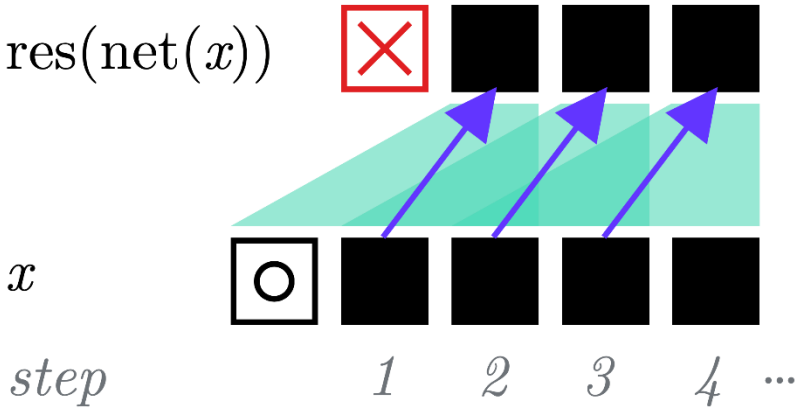
Online with padding



Continual temporal convolution (Delay residuals)



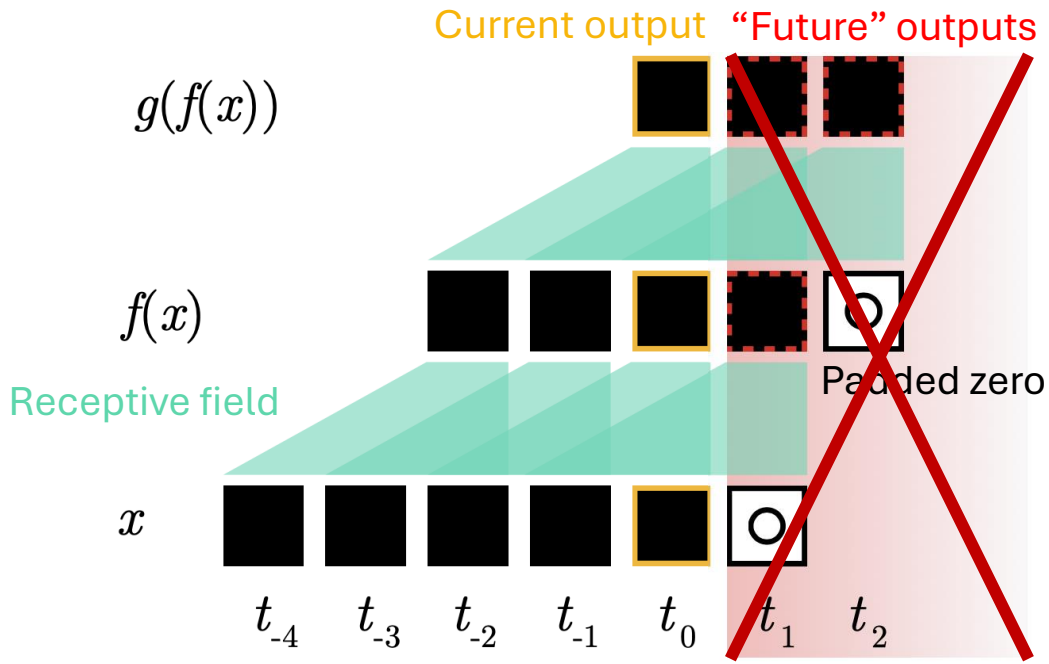
Offline inference



Online inference

Residual connections ↑ over a module with receptive field of size ▲ and padding one (“equal padding”) . ☒ are empty outputs.

Continual temporal convolution (zero-padding)



Zero-padding is used for:
- Avoiding spatio-temporal shrinkage at multiple layers
- It prevents information at the borders from "washing out"

These issues do not appear in Continual 3D CNNs

Removal of zero-padding → *"Model shift"* wrt. source network

Continual temporal convolution (Extended receptive field)

Method approach:

1. Reuse existing 3D CNN
2. Transfer source 3D CNN to Co3D CNN
3. Extend receptive field

Computational complexity of CoConv is independent of *T*.

↓

Longer effective clip size is **cheap**

↓

Trick :
Increase length continual global-average pooling layer