

# Lecture 18: Videos

# Computer Vision Tasks: 2D Recognition

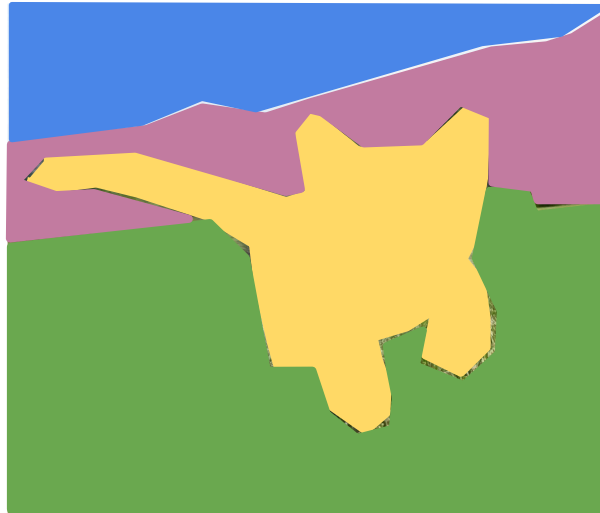
## Classification



**CAT**

No spatial extent

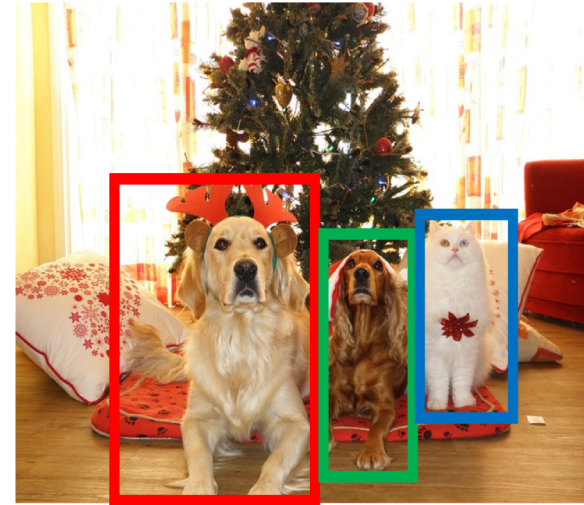
## Semantic Segmentation



**GRASS, CAT, TREE, SKY**

No objects, just pixels

## Object Detection



**DOG, DOG, CAT**

Multiple Objects

## Instance Segmentation



**DOG, DOG, CAT**

[This image is CC0 public domain](#)

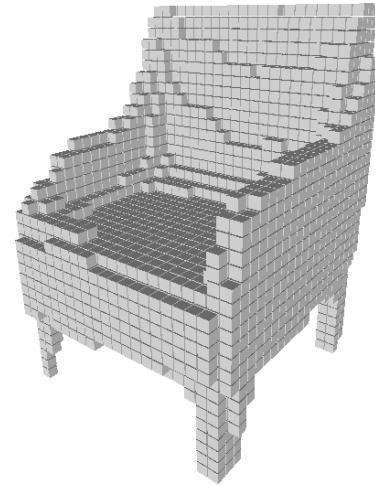


# Last Time: 3D Shapes

Predicting 3D Shapes  
from single image

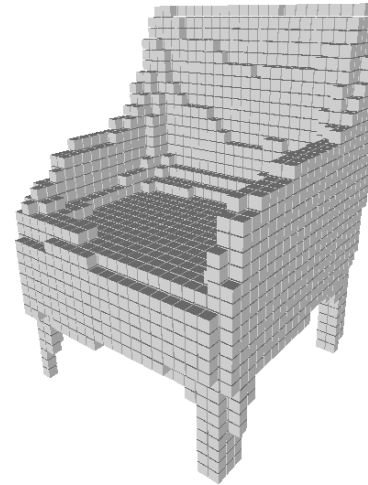


Input Image



3D Shape

Processing 3D  
input data

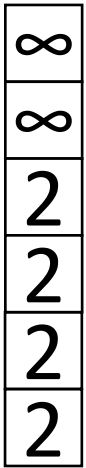


3D Shape

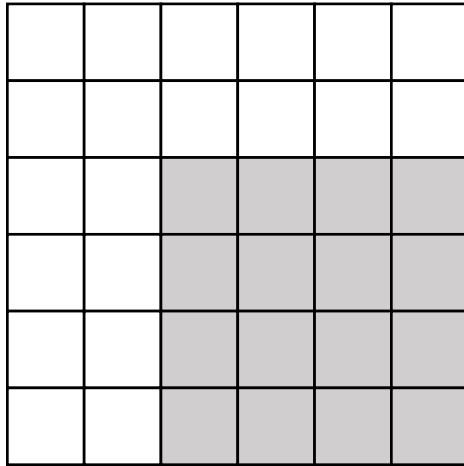


Chair

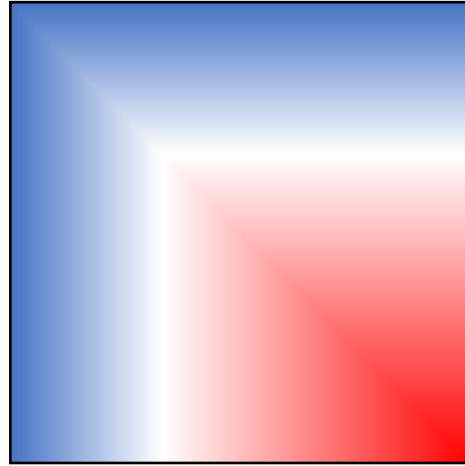
# Last Time: 3D Shape Representations



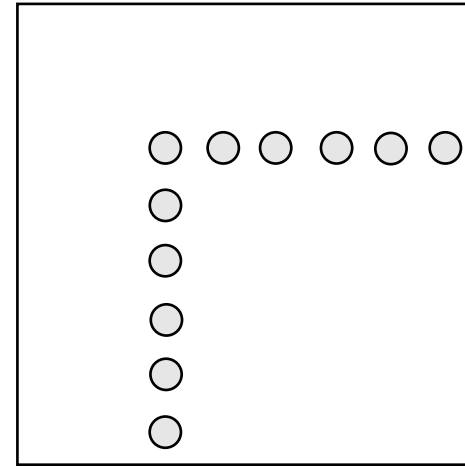
Depth  
Map



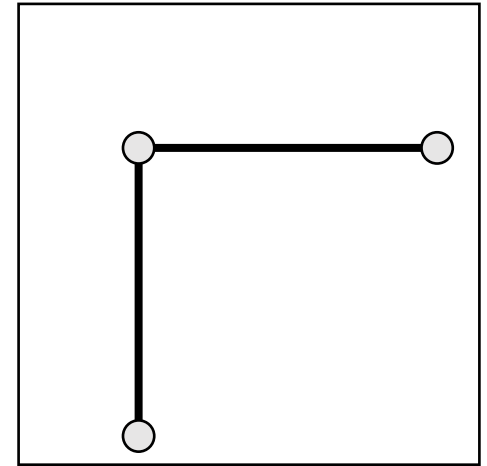
Voxel  
Grid



Implicit  
Surface



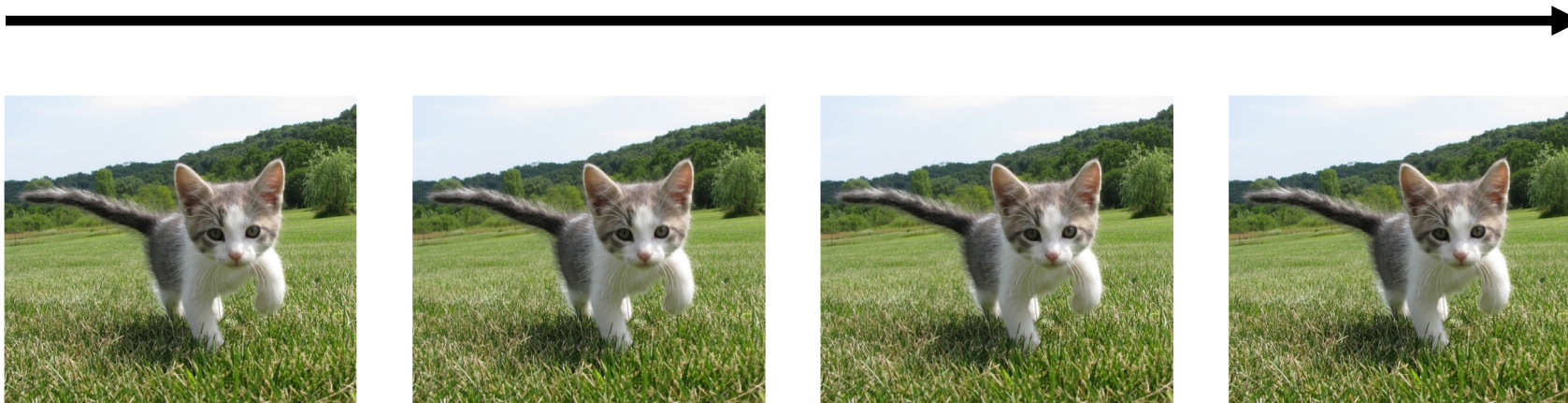
Pointcloud



Mesh

# Today: Video = 2D + Time

A video is a **sequence** of images  
4D tensor:  $T \times 3 \times H \times W$   
(or  $3 \times T \times H \times W$ )



[This image](#) is [CC0 public domain](#)

# Example task: Video Classification



Input video:  
 $T \times 3 \times H \times W$



Swimming  
**Running**  
Jumping  
Eating  
Standing

[Running video](#) is in the [public domain](#)

# Example task: Video Classification



Images: Recognize **objects**



Dog  
**Cat**  
Fish  
Truck



Videos: Recognize **actions**



Swimming  
**Running**  
Jumping  
Eating  
Standing

[Running video](#) is in the [public domain](#)



# Problem: Videos are big!

Videos are ~30 frames per second (fps)



Input video:  
 $T \times 3 \times H \times W$

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**

# Problem: Videos are big!

Videos are ~30 frames per second (fps)



Input video:  
 $T \times 3 \times H \times W$

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

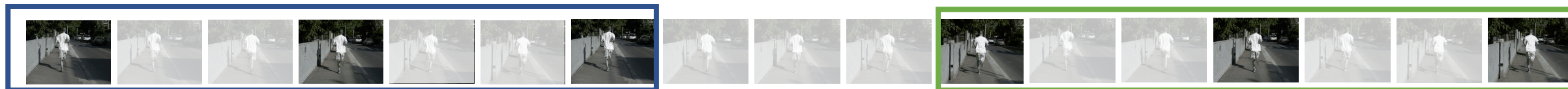


# Training on Clips

**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

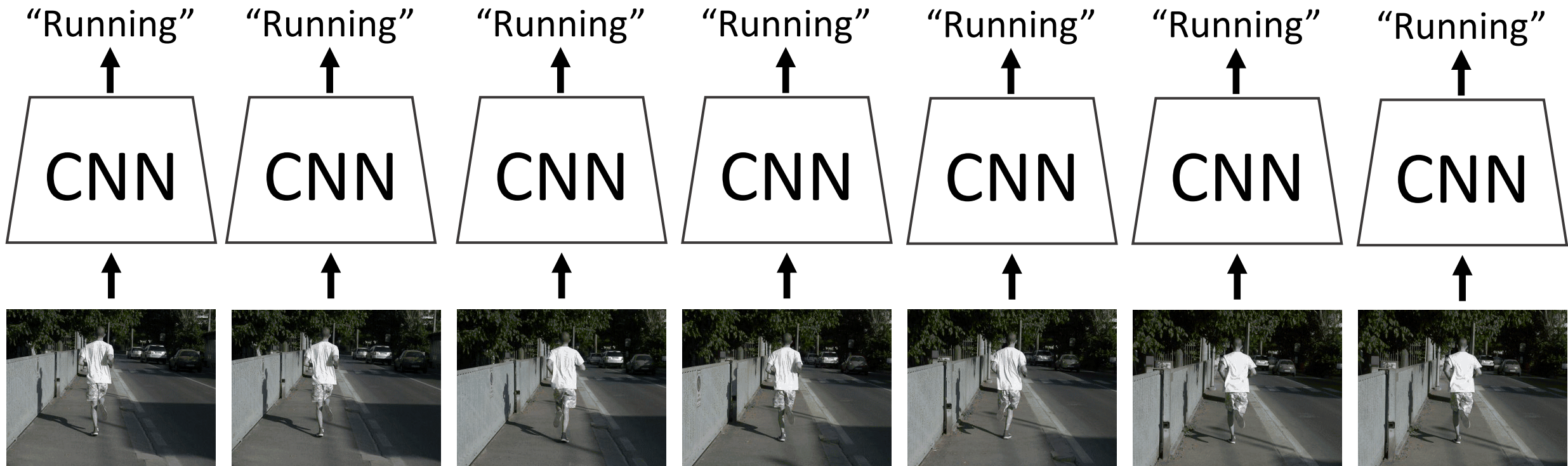




# Video Classification: Single-Frame CNN

Simple idea: train normal 2D CNN to classify video frames independently!  
(Average predicted probs at test-time)

Often a **very** strong baseline for video classification

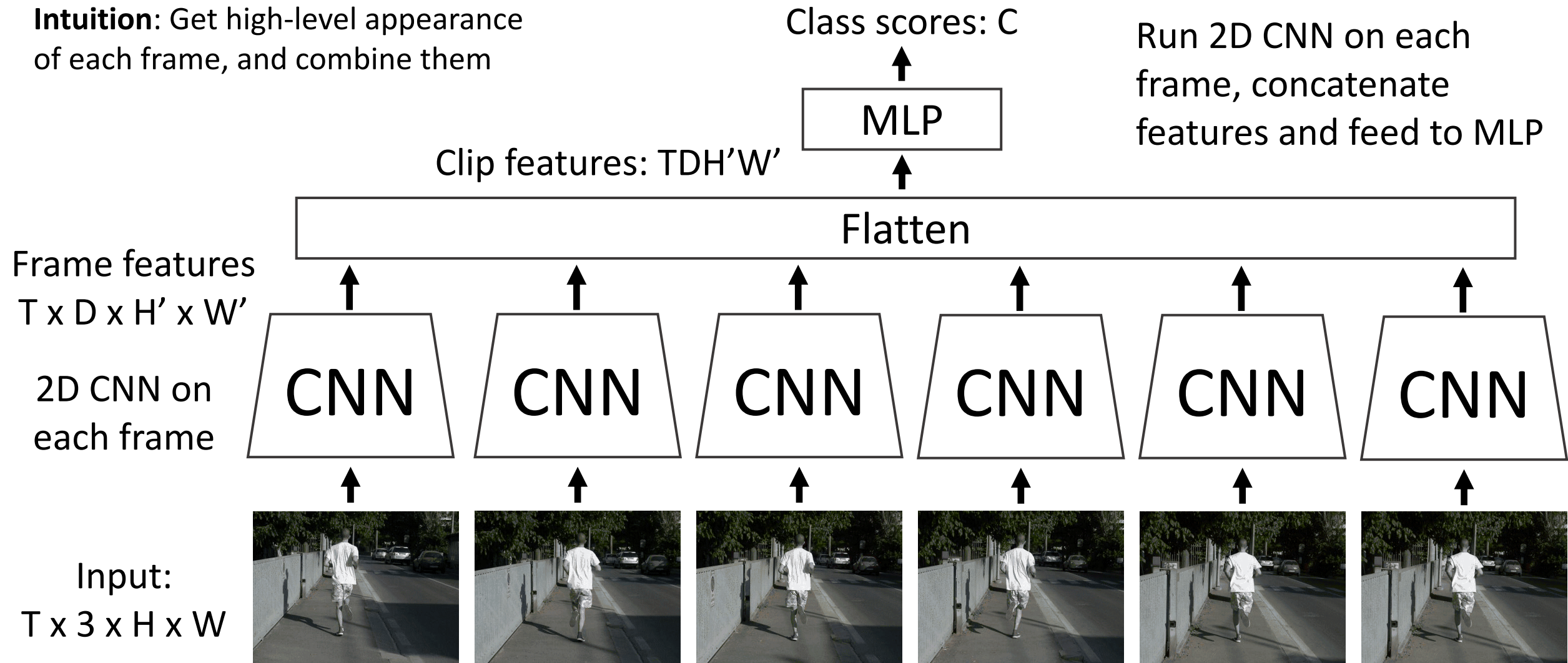


# Video Classification: Late Fusion (with FC layers)

**Intuition:** Get high-level appearance of each frame, and combine them

Class scores:  $C$

Run 2D CNN on each frame, concatenate features and feed to MLP



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

# Video Classification: Late Fusion (with pooling)

**Intuition:** Get high-level appearance of each frame, and combine them

Class scores:  $C$

Run 2D CNN on each frame, pool features and feed to Linear

Clip features:  $D$

Linear

Average Pool over space and time

Frame features  
 $T \times D \times H' \times W'$

2D CNN on  
each frame

CNN

CNN

CNN

CNN

CNN

CNN

Input:

$T \times 3 \times H \times W$



# Video Classification: Late Fusion (with pooling)

**Intuition:** Get high-level appearance of each frame, and combine them

**Problem:** Hard to compare low-level motion between frames

Class scores:  $C$

Linear

Clip features:  $D$

Run 2D CNN on each frame, pool features and feed to Linear

Average Pool over space and time

Frame features  
 $T \times D \times H' \times W'$

2D CNN on  
each frame

CNN

CNN

CNN

CNN

CNN

CNN

Input:

$T \times 3 \times H \times W$





# Video Classification: Early Fusion

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

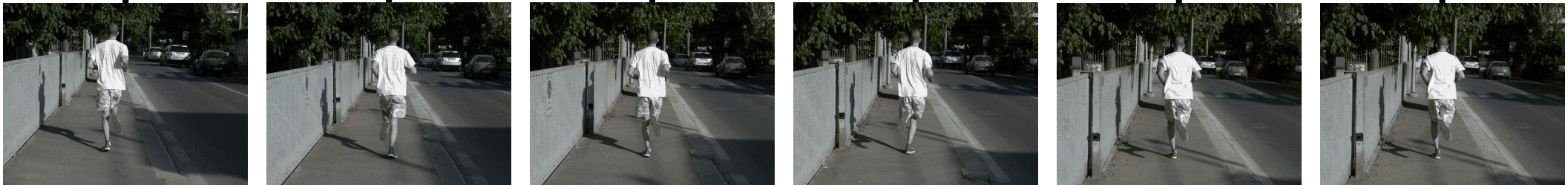
First 2D convolution collapses all temporal information:

**Input:**  $3T \times H \times W$

**Output:**  $D \times H \times W$

Reshape:  
 $3T \times H \times W$

Input:  
 $T \times 3 \times H \times W$





# Video Classification: Early Fusion

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

**Problem:** One layer of temporal processing may not be enough!

First 2D convolution collapses all temporal information:

**Input:**  $3T \times H \times W$

**Output:**  $D \times H \times W$

Reshape:

$3T \times H \times W$

Input:

$T \times 3 \times H \times W$



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

# Video Classification: 3D CNN

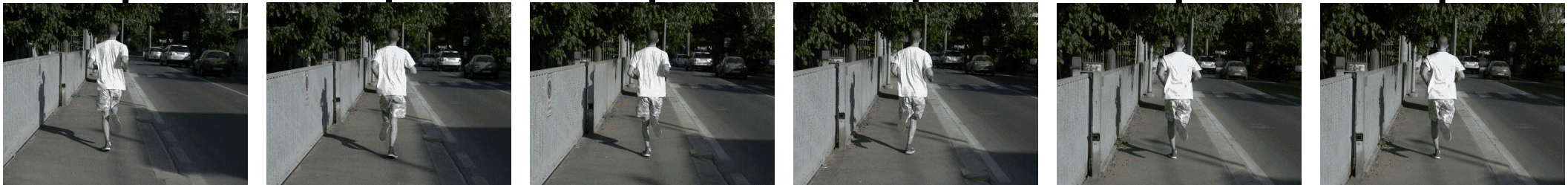
**Intuition:** Use 3D versions of convolution and pooling to slowly fuse temporal information over the course of the network

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

Class scores:  $C$

3D CNN

Input:  
 $3 \times T \times H \times W$



Ji et al, "3D Convolutional Neural Networks for Human Action Recognition", TPAMI 2010 ; Karpathy et al, "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014

# Early Fusion vs Late Fusion vs 3D CNN

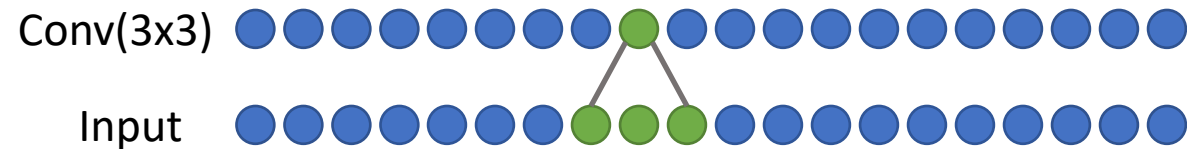
Late  
Fusion

Layer	Size (C x T x H x W)	Receptive Field (T x H x W)
Input	3 x 20 x 64 x 64	
Conv2D(3x3, 3->12)	12 x 20 x 64 x 64	1 x 3 x 3

# Early Fusion vs Late Fusion vs 3D CNN

Late  
Fusion

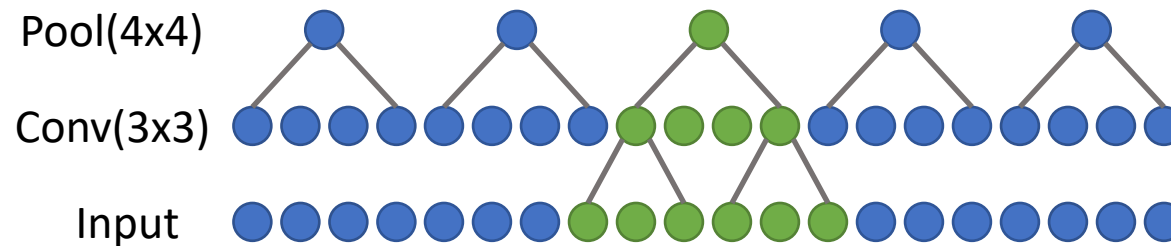
Layer	Size (C x T x H x W)	Receptive Field (T x H x W)
Input	3 x 20 x 64 x 64	
Conv2D(3x3, 3->12)	12 x 20 x 64 x 64	1 x 3 x 3



# Early Fusion vs Late Fusion vs 3D CNN

Late  
Fusion

Layer	Size (C x T x H x W)	Receptive Field (T x H x W)
Input	3 x 20 x 64 x 64	
Conv2D(3x3, 3→12)	12 x 20 x 64 x 64	1 x 3 x 3
Pool2D(4x4)	12 x 20 x 16 x 16	1 x 6 x 6



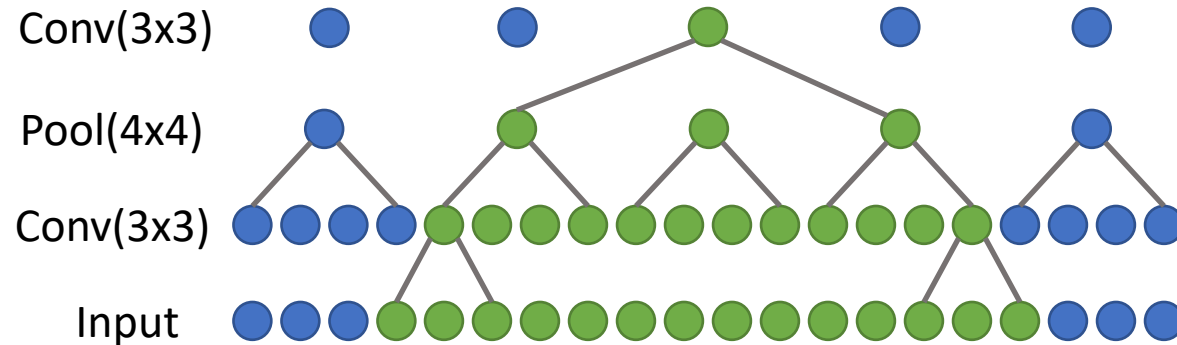


# Early Fusion vs Late Fusion vs 3D CNN

Late  
Fusion

Layer	Size (C x T x H x W)	Receptive Field (T x H x W)
Input	3 x 20 x 64 x 64	
Conv2D(3x3, 3->12)	12 x 20 x 64 x 64	1 x 3 x 3
Pool2D(4x4)	12 x 20 x 16 x 16	1 x 6 x 6
Conv2D(3x3, 12->24)	24 x 20 x 16 x 16	1 x 14 x 14

Build slowly in space

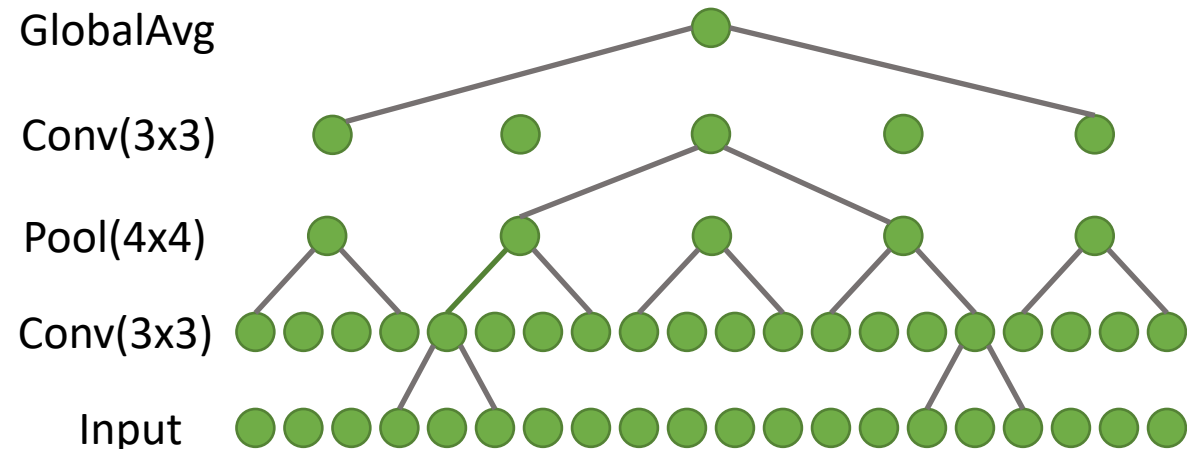


# Early Fusion vs Late Fusion vs 3D CNN

Late  
Fusion

Layer	Size (C x T x H x W)	Receptive Field (T x H x W)
Input	3 x 20 x 64 x 64	
Conv2D(3x3, 3->12)	12 x 20 x 64 x 64	1 x 3 x 3
Pool2D(4x4)	12 x 20 x 16 x 16	1 x 6 x 6
Conv2D(3x3, 12->24)	24 x 20 x 16 x 16	1 x 14 x 14
GlobalAvgPool	24 x 1 x 1 x 1	20 x 64 x 64

Build slowly in space,  
All-at-once in time at end



# Early Fusion vs Late Fusion vs 3D CNN

Late  
Fusion

Layer	Size (C x T x H x W)	Receptive Field (T x H x W)
Input	3 x 20 x 64 x 64	
Conv2D(3x3, 3->12)	12 x 20 x 64 x 64	1 x 3 x 3
Pool2D(4x4)	12 x 20 x 16 x 16	1 x 6 x 6
Conv2D(3x3, 12->24)	24 x 20 x 16 x 16	1 x 14 x 14
GlobalAvgPool	24 x 1 x 1 x 1	20 x 64 x 64
Input	3 x 20 x 64 x 64	
Conv2D(3x3, 3*10->12)	12 x 64 x 64	20 x 3 x 3
Pool2D(4x4)	12 x 16 x 16	20 x 6 x 6
Conv2D(3x3, 12->24)	24 x 16 x 16	20 x 14 x 14
GlobalAvgPool	24 x 1 x 1	20 x 64 x 64

Build slowly in space,  
All-at-once in time at end

Early  
Fusion

Build slowly in space,  
All-at-once in time at start

(Small example architectures,  
in practice much bigger)

# Early Fusion vs Late Fusion vs 3D CNN

	Layer	Size (C x T x H x W)	Receptive Field (T x H x W)	
Late Fusion	Input	3 x 20 x 64 x 64		Build slowly in space, All-at-once in time at end
	Conv2D(3x3, 3->12)	12 x 20 x 64 x 64	1 x 3 x 3	
	Pool2D(4x4)	12 x 20 x 16 x 16	1 x 6 x 6	
	Conv2D(3x3, 12->24)	24 x 20 x 16 x 16	1 x 14 x 14	
	GlobalAvgPool	24 x 1 x 1 x 1	20 x 64 x 64	
Early Fusion	Input	3 x 20 x 64 x 64		Build slowly in space, All-at-once in time at start
	Conv2D(3x3, 3*10->12)	12 x 64 x 64	20 x 3 x 3	
	Pool2D(4x4)	12 x 16 x 16	20 x 6 x 6	
	Conv2D(3x3, 12->24)	24 x 16 x 16	20 x 14 x 14	
	GlobalAvgPool	24 x 1 x 1	20 x 64 x 64	
3D CNN	Input	3 x 20 x 64 x 64		Build slowly in space, Build slowly in time "Slow Fusion"
	Conv3D(3x3x3, 3->12)	12 x 20 x 64 x 64	3 x 3 x 3	
	Pool3D(4x4x4)	12 x 5 x 16 x 16	6 x 6 x 6	
	Conv3D(3x3x3, 12->24)	24 x 5 x 16 x 16	14 x 14 x 14	
	GlobalAvgPool	24 x 1 x 1	20 x 64 x 64	

# Early Fusion vs Late Fusion vs 3D CNN

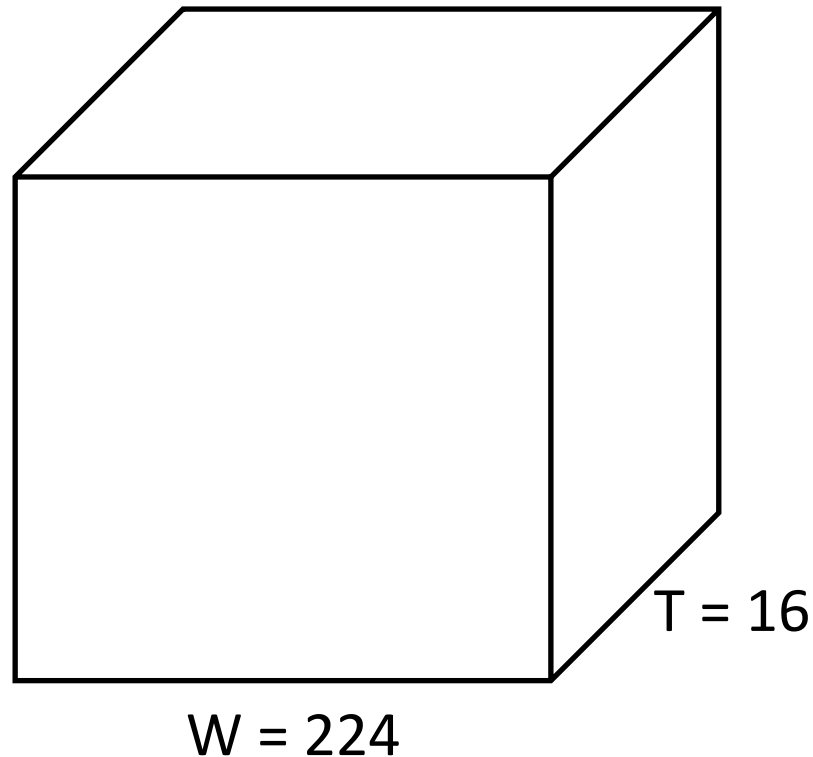
What is the difference?

	Layer	Size (C x T x H x W)	Receptive Field (T x H x W)	
Late Fusion	Input	3 x 20 x 64 x 64		Build slowly in space, All-at-once in time at end
	Conv2D(3x3, 3->12)	12 x 20 x 64 x 64	1 x 3 x 3	
	Pool2D(4x4)	12 x 20 x 16 x 16	1 x 6 x 6	
	Conv2D(3x3, 12->24)	24 x 20 x 16 x 16	1 x 14 x 14	
	GlobalAvgPool	24 x 1 x 1 x 1	20 x 64 x 64	
Early Fusion	Input	3 x 20 x 64 x 64		Build slowly in space, All-at-once in time at start
	Conv2D(3x3, 3*10->12)	12 x 64 x 64	20 x 3 x 3	
	Pool2D(4x4)	12 x 16 x 16	20 x 6 x 6	
	Conv2D(3x3, 12->24)	24 x 16 x 16	20 x 14 x 14	
	GlobalAvgPool	24 x 1 x 1	20 x 64 x 64	
3D CNN	Input	3 x 20 x 64 x 64		Build slowly in space, Build slowly in time "Slow Fusion"
	Conv3D(3x3x3, 3->12)	12 x 20 x 64 x 64	3 x 3 x 3	
	Pool3D(4x4x4)	12 x 5 x 16 x 16	6 x 6 x 6	
	Conv3D(3x3x3, 12->24)	24 x 5 x 16 x 16	14 x 14 x 14	
	GlobalAvgPool	24 x 1 x 1	20 x 64 x 64	

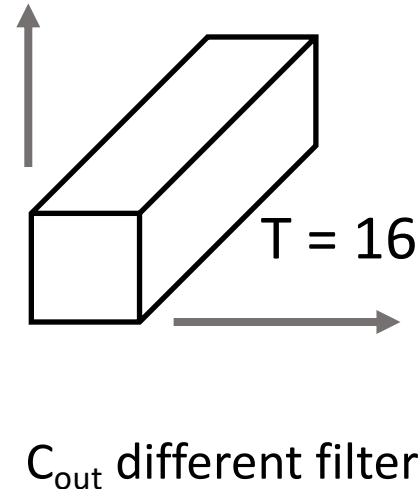


# 2D Conv (Early Fusion) vs 3D Conv (3D CNN)

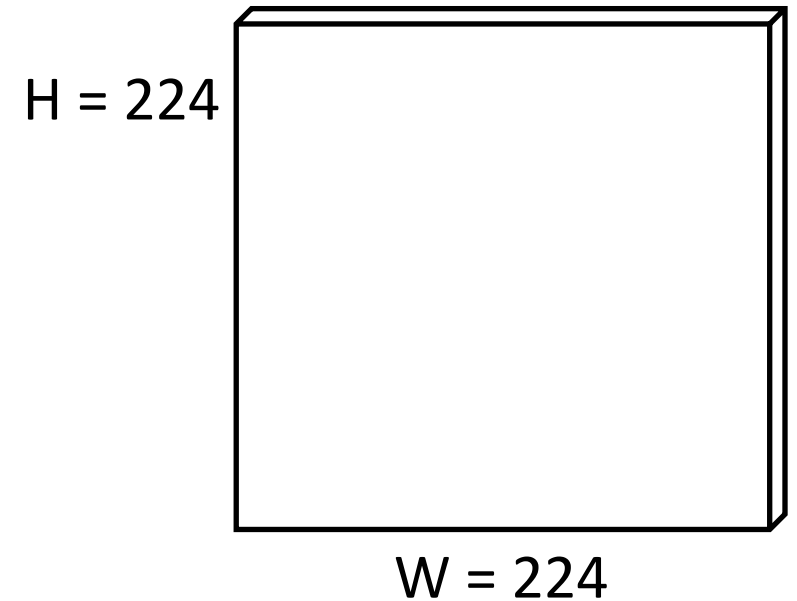
**Input:**  $C_{in} \times T \times H \times W$   
(3D grid with  $C_{in}$ -dim  
feat at each point)



**Weight:**  
 $C_{out} \times C_{in} \times T \times 3 \times 3$   
Slide over x and y

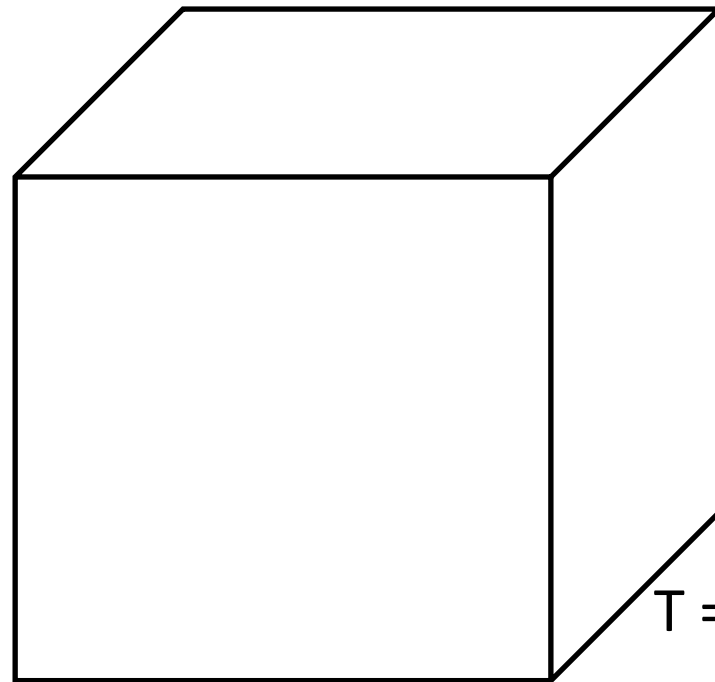


**Output:**  
 $C_{out} \times H \times W$   
2D grid with  $C_{out}$ -dim  
feat at each point



# 2D Conv (Early Fusion) vs 3D Conv (3D CNN)

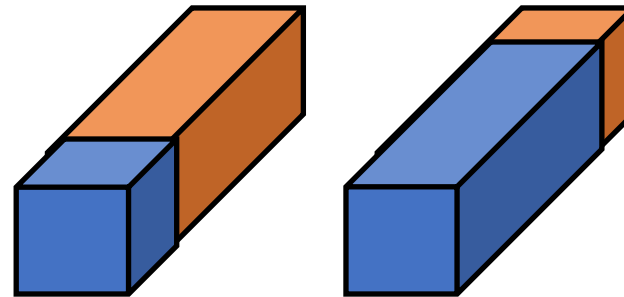
**Input:**  $C_{in} \times T \times H \times W$   
(3D grid with  $C_{in}$ -dim  
feat at each point)



**Weight:**

$C_{out} \times C_{in} \times T \times 3 \times 3$   
Slide over x and y

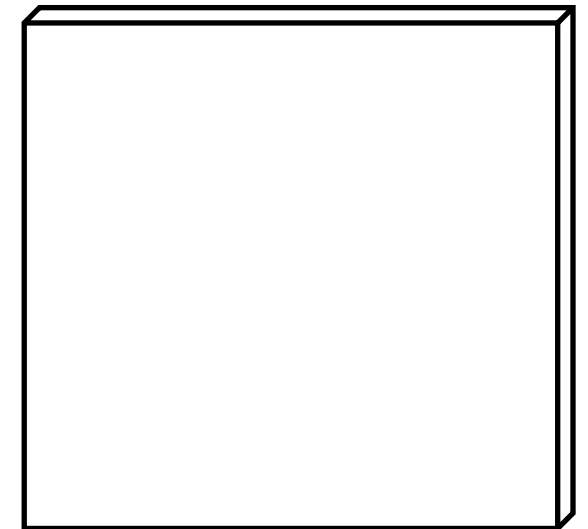
No temporal shift-invariance! Needs  
to learn separate filters for the same  
motion at different times in the clip



$C_{out}$  different filters

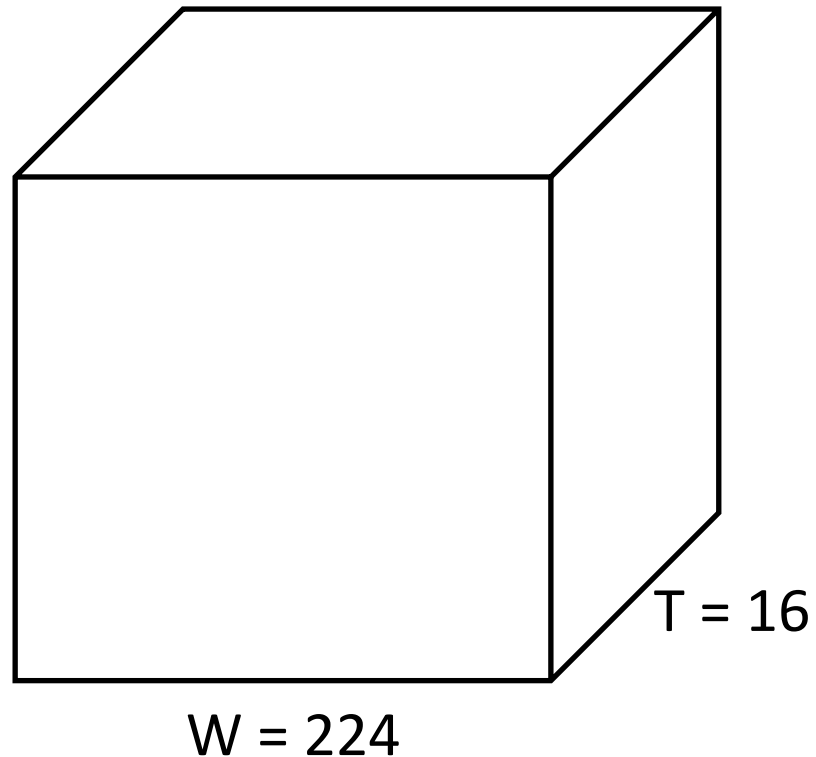
**Output:**

$C_{out} \times H \times W$   
2D grid with  $C_{out}$ -dim  
feat at each point

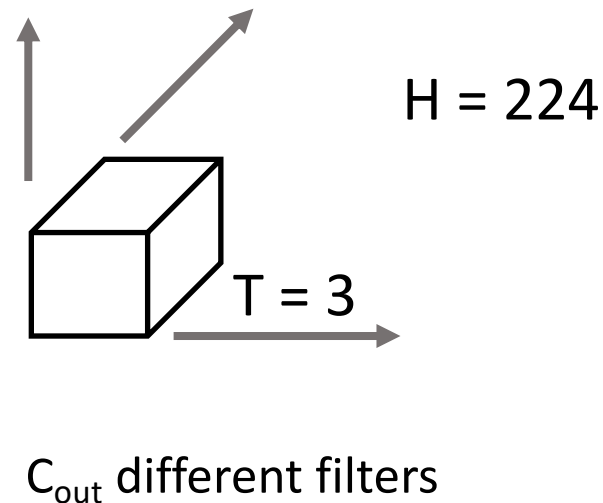


# 2D Conv (Early Fusion) vs 3D Conv (3D CNN)

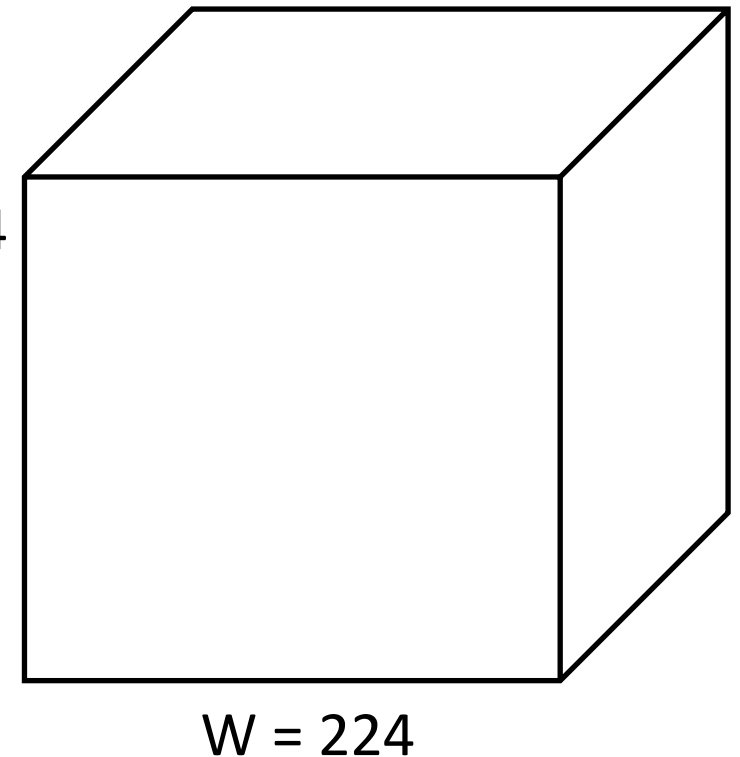
**Input:**  $C_{in} \times T \times H \times W$   
(3D grid with  $C_{in}$ -dim  
feat at each point)



**Weight:**  
 $C_{out} \times C_{in} \times 3 \times 3 \times 3$   
Slide over x and y

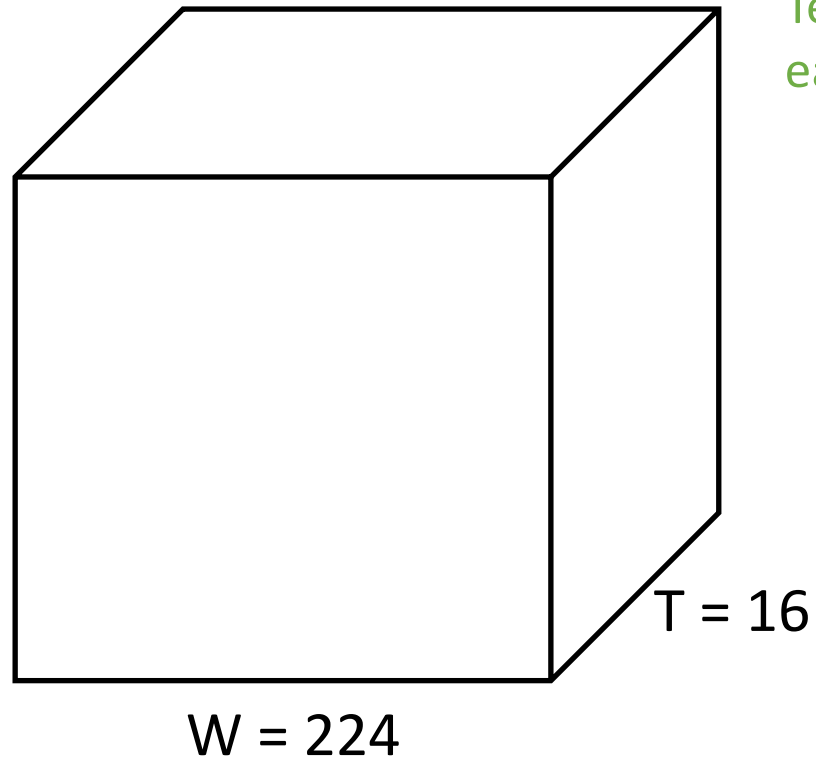


**Output:**  
 $C_{out} \times T \times H \times W$   
3D grid with  $C_{out}$ -dim  
feat at each point



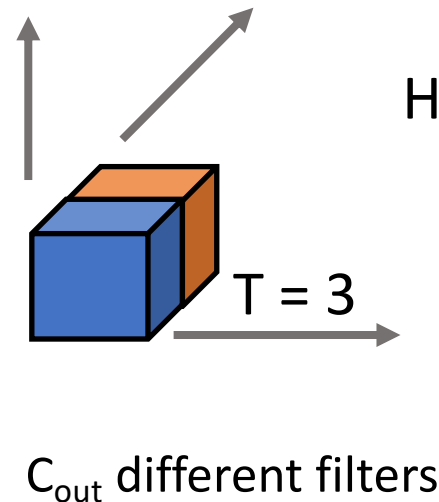
# 2D Conv (Early Fusion) vs 3D Conv (3D CNN)

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

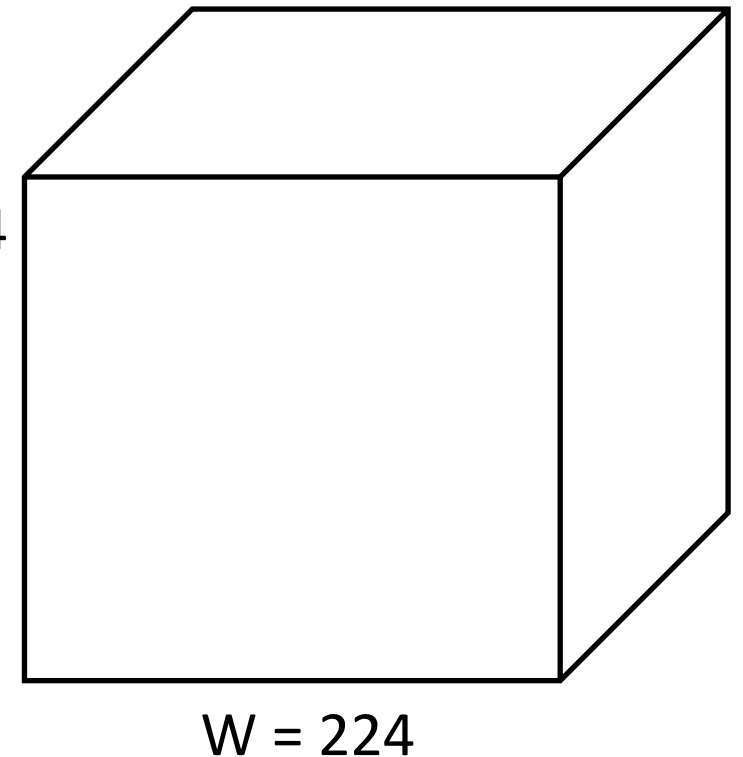


**Weight:**  
 $C_{out} \times C_{in} \times 3 \times 3 \times 3$   
Slide over x and y

Temporal shift-invariant since  
each filter slides over time!

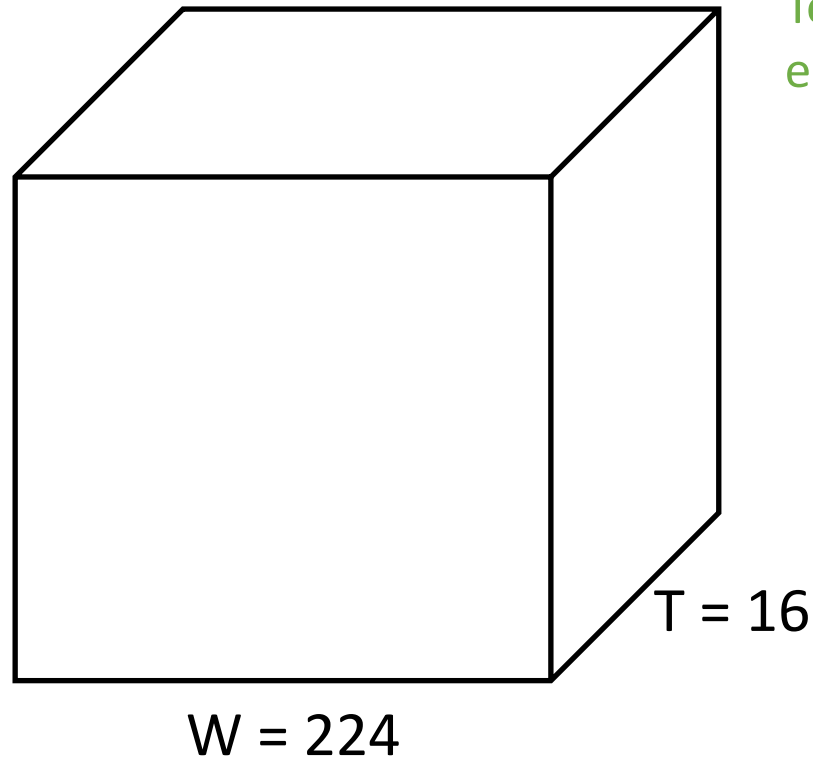


**Output:**  
 $C_{out} \times T \times H \times W$   
3D grid with  $C_{out}$ -dim  
feat at each point



# 2D Conv (Early Fusion) vs 3D Conv (3D CNN)

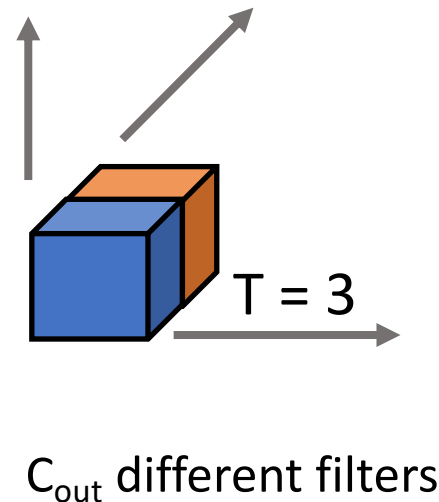
**Input:**  $C_{in} \times T \times H \times W$   
(3D grid with  $C_{in}$ -dim  
feat at each point)



**Weight:**

$C_{out} \times C_{in} \times 3 \times 3 \times 3$   
Slide over x and y

Temporal shift-invariant since  
each filter slides over time!



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



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



# Example Video Dataset: Sports-1M



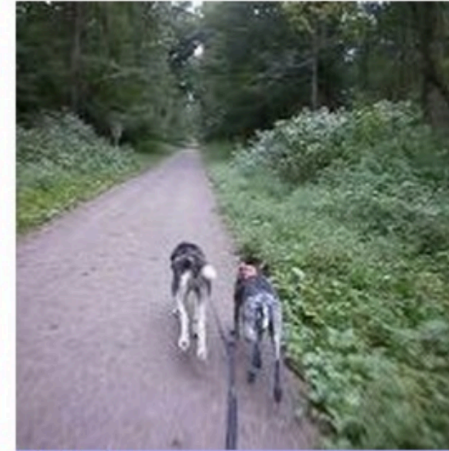
track cycling  
cycling  
track cycling  
road bicycle racing  
marathon  
ultramarathon



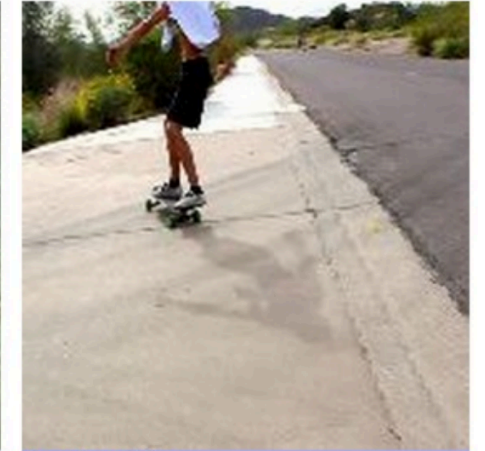
ultramarathon  
ultramarathon  
half marathon  
running  
marathon  
inline speed skating



heptathlon  
heptathlon  
decathlon  
hurdles  
pentathlon  
sprint (running)



bikejoring  
mushing  
bikejoring  
harness racing  
skijoring  
carting



longboarding  
longboarding  
aggressive inline skating  
freestyle scootering  
freeboard (skateboard)  
sandboarding

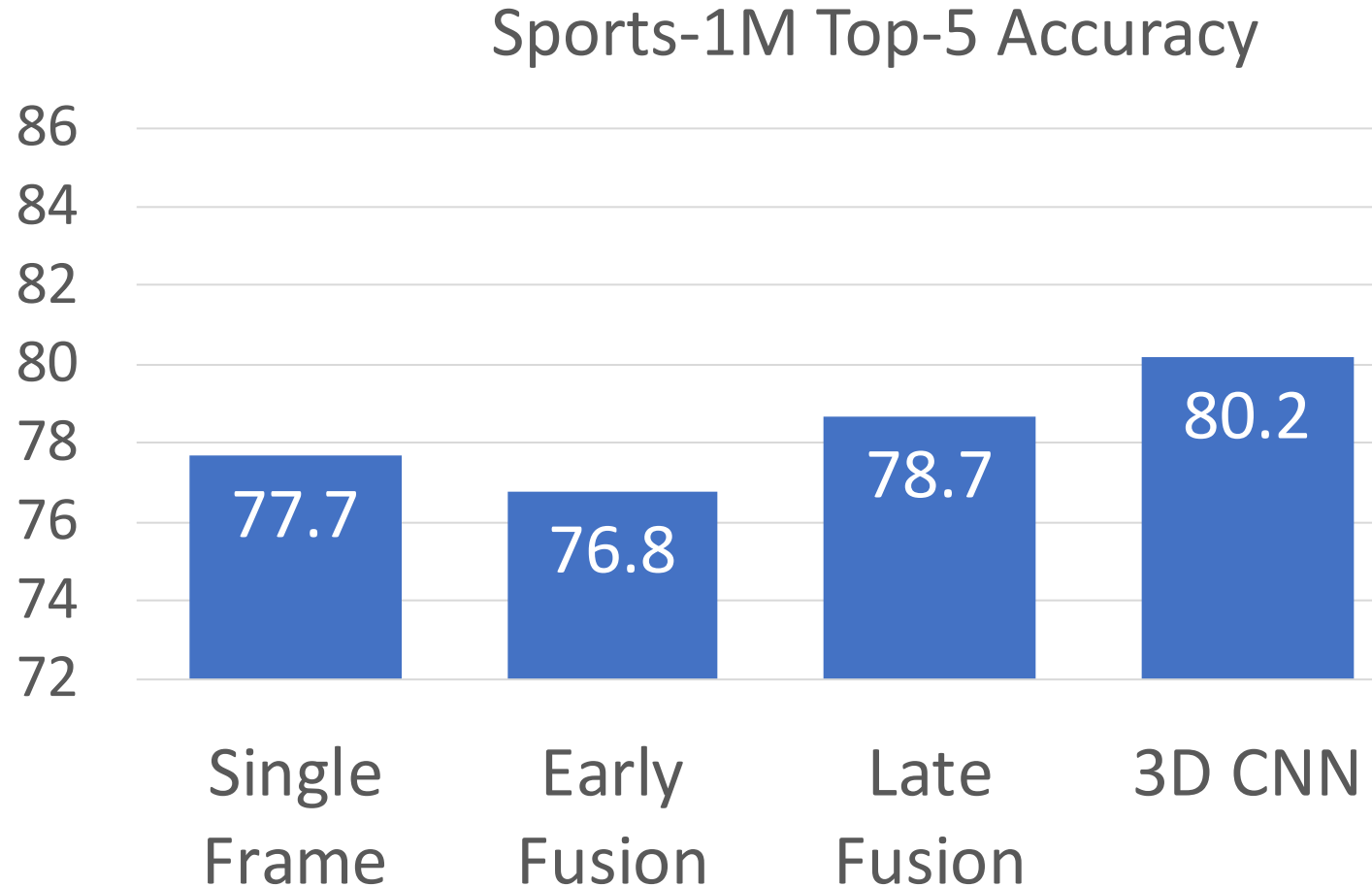
1 million YouTube videos  
annotated with labels for  
487 different types of sports

**Ground Truth**

**Correct prediction**

**Incorrect prediction**

# Early Fusion vs Late Fusion vs 3D CNN



Single Frame model works well – always try this first!

3D CNNs have improved a lot since 2014!

# 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

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

# 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

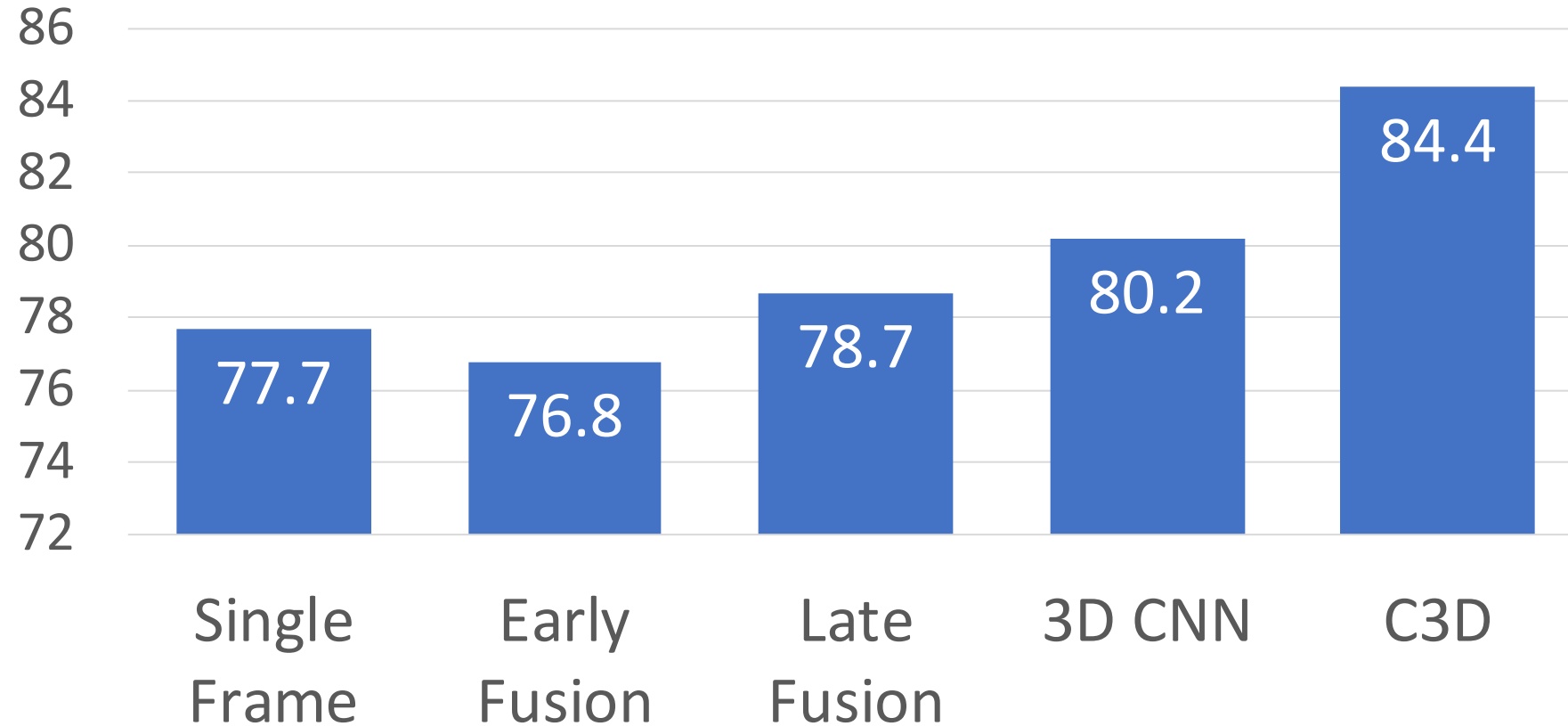
VGG-16: 13.6 GFLOP

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

# Early Fusion vs Late Fusion vs 3D CNN

Sports-1M Top-5 Accuracy



Karpathy et al, "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014  
Tran et al, "Learning Spatiotemporal Features with 3D Convolutional Networks", ICCV 2015



# Recognizing Actions from Motion

We can easily recognize actions using only **motion information**



Johansson, "Visual perception of biological motion and a model for its analysis." *Perception & Psychophysics*. 14(2):201-211. 1973.

# Measuring Motion: Optical Flow

Image at frame  $t$

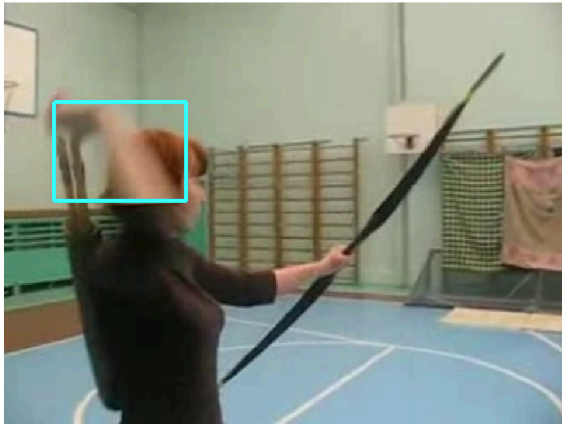
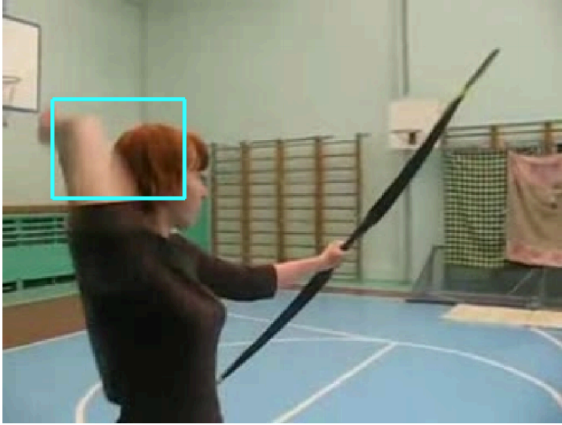
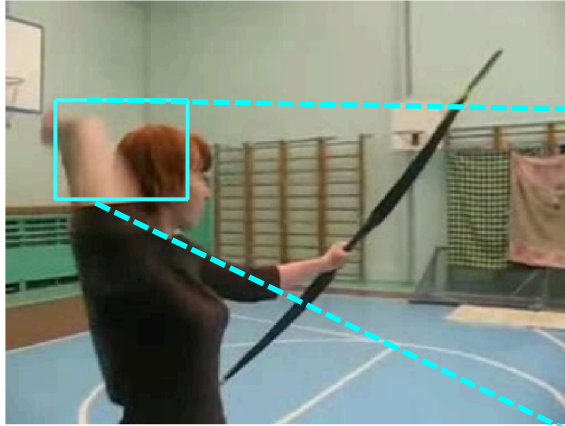


Image at frame  $t+1$

Simonyan and Zisserman, "Two-stream convolutional networks for action recognition in videos", NeurIPS 2014

# Measuring Motion: Optical Flow

Image at frame t



Optical flow gives a displacement field  $F$  between images  $I_t$  and  $I_{t+1}$

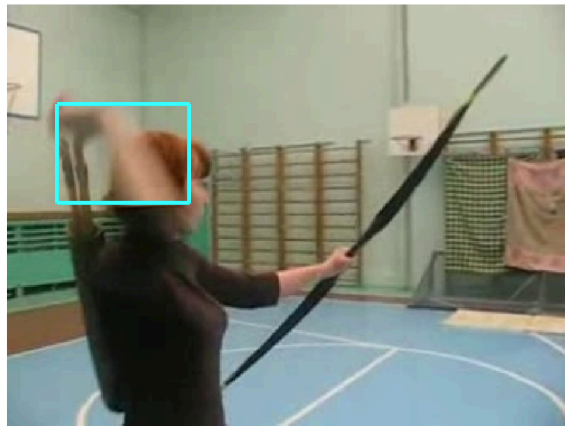
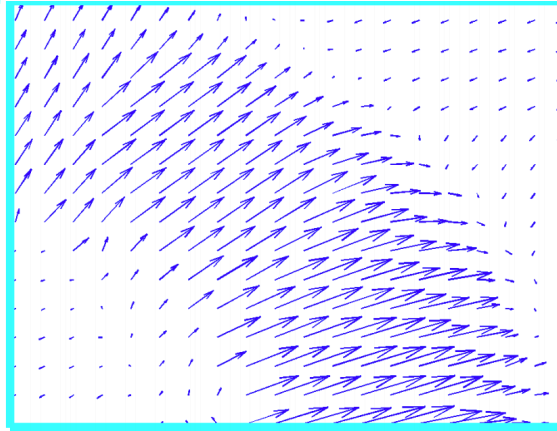


Image at frame t+1

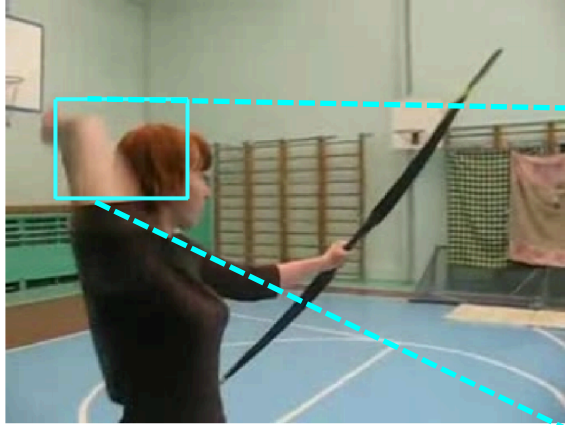
Tells where each pixel will move in the next frame:

$$F(x, y) = (dx, dy)$$

$$I_{t+1}(x+dx, y+dy) = I_t(x, y)$$

# Measuring Motion: Optical Flow

Image at frame t



Optical flow gives a displacement field  $F$  between images  $I_t$  and  $I_{t+1}$

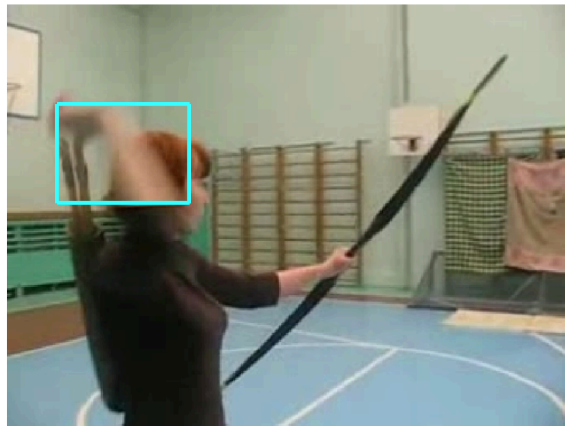
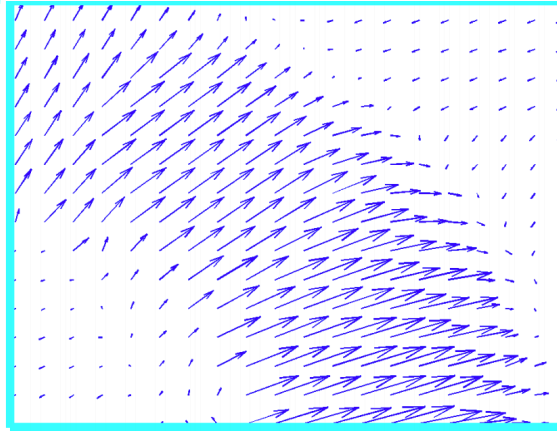
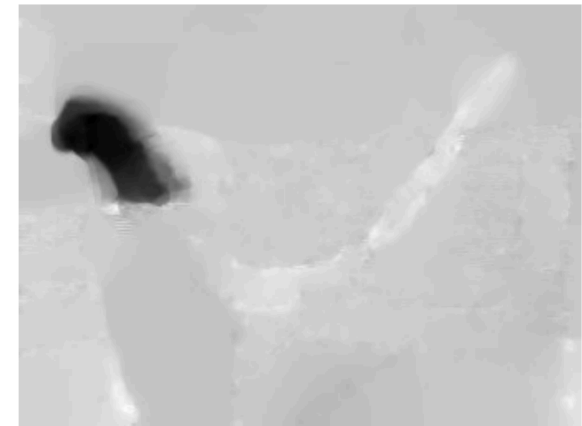
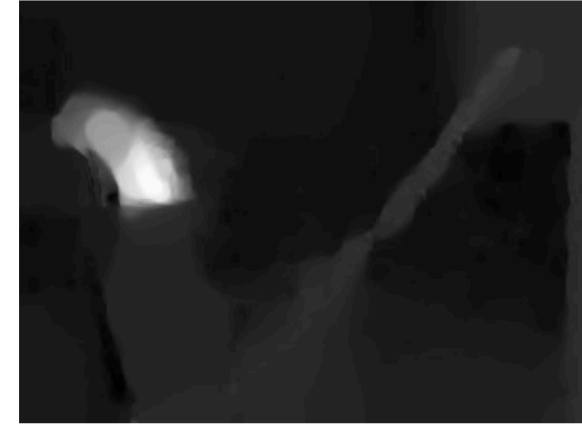


Image at frame t+1

Tells where each pixel will move in the next frame:  
 $F(x, y) = (dx, dy)$   
 $I_{t+1}(x+dx, y+dy) = I_t(x, y)$

Optical Flow highlights  
**local motion**

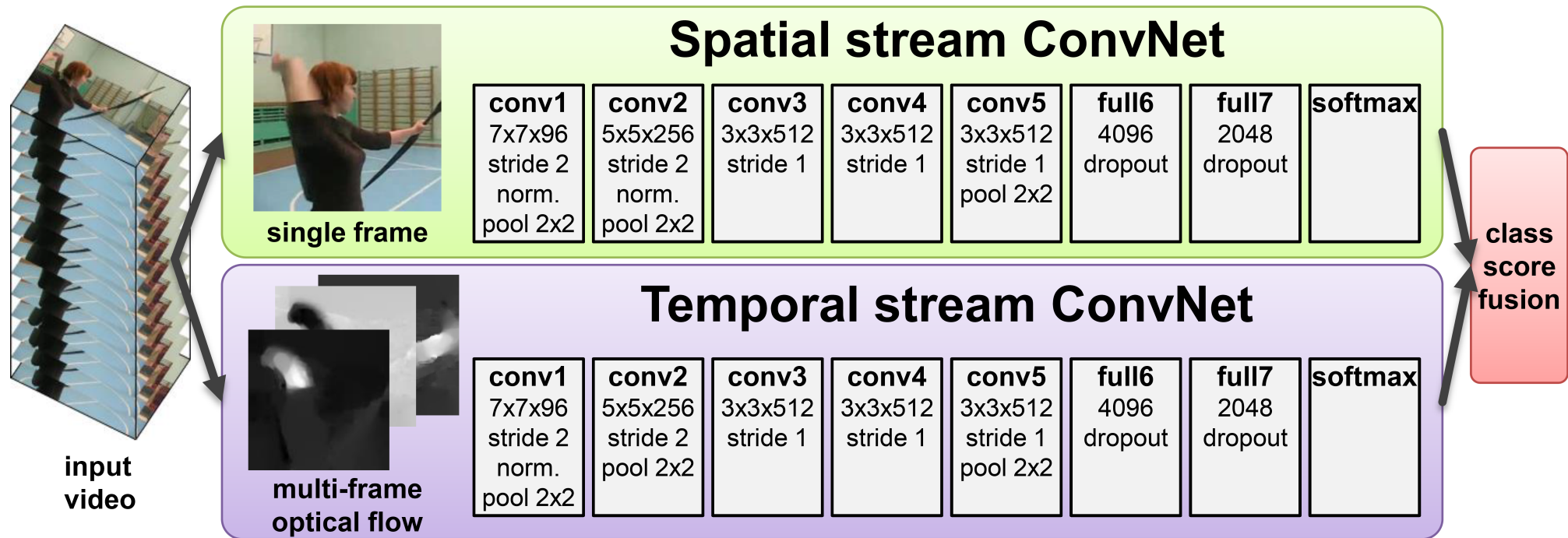
Horizontal flow  $dx$



Vertical Flow  $dy$

# Separating Motion and Appearance: Two-Stream Networks

**Input: Single Image**  
 $3 \times H \times W$

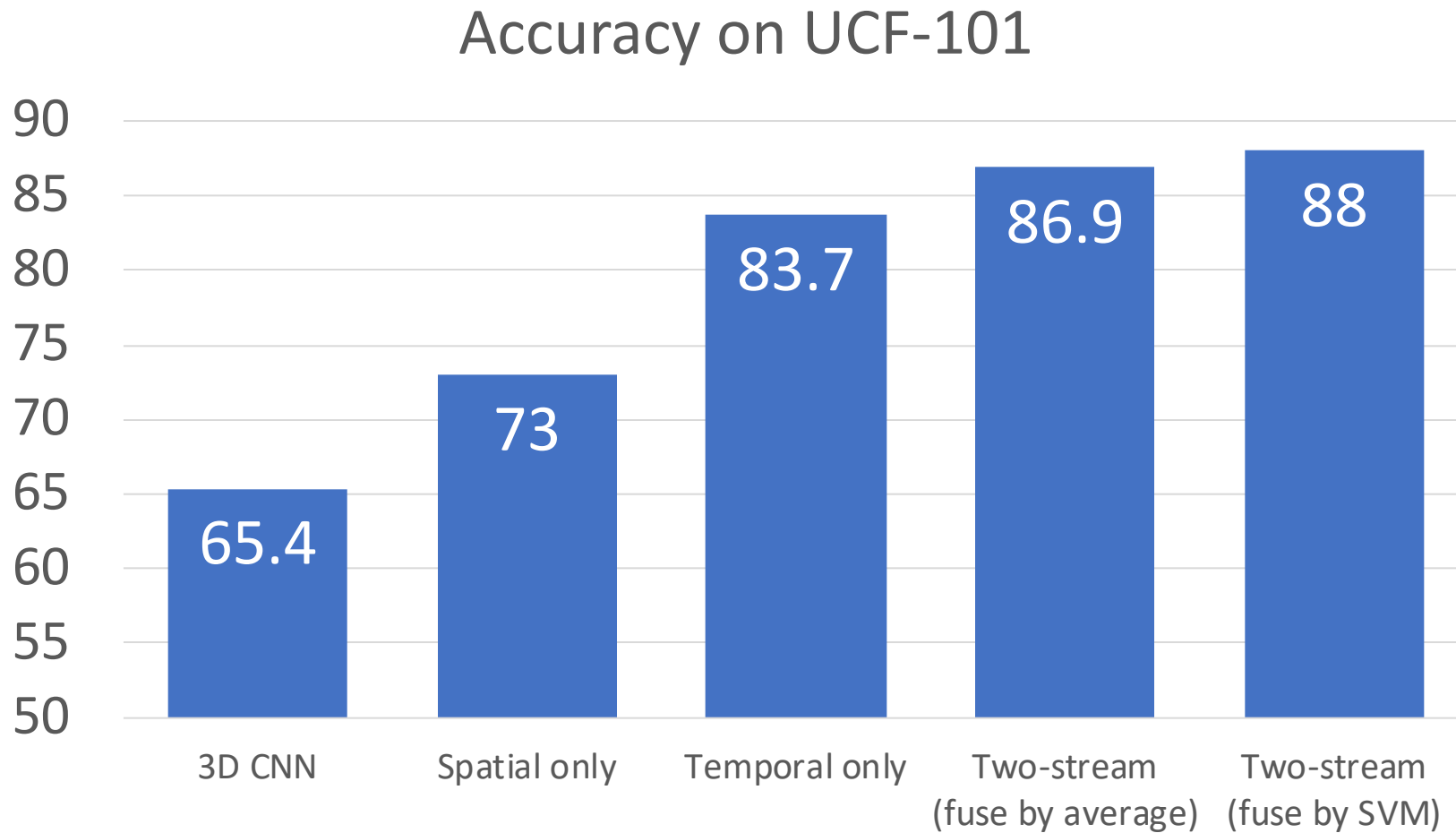


**Input: Stack of optical flow:**  
 $[2 \times (T-1)] \times H \times W$

**Early fusion:** First 2D conv processes all flow images



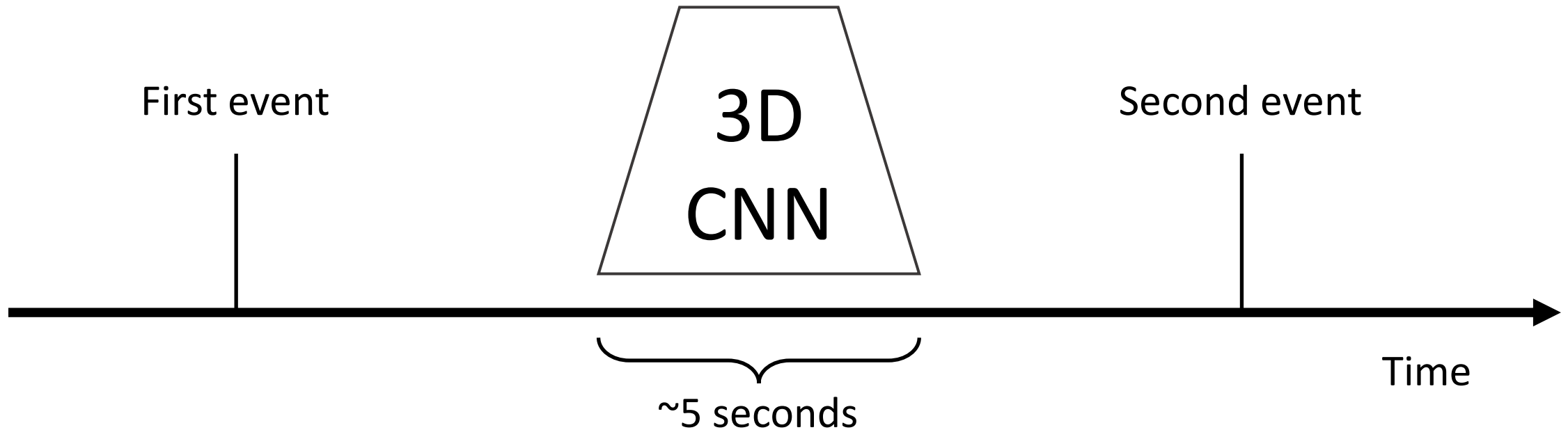
# Separating Motion and Appearance: Two-Stream Networks



Simonyan and Zisserman, "Two-stream convolutional networks for action recognition in videos", NeurIPS 2014

# Modeling long-term temporal structure

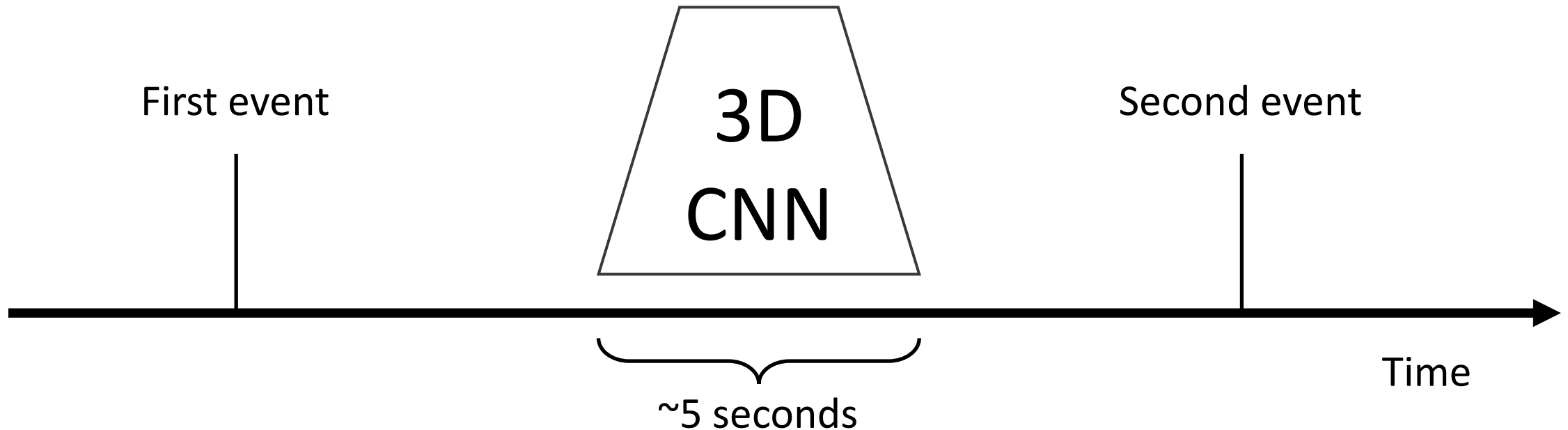
So far all our temporal CNNs only model local motion between frames in very short clips of ~2-5 seconds. What about long-term structure?



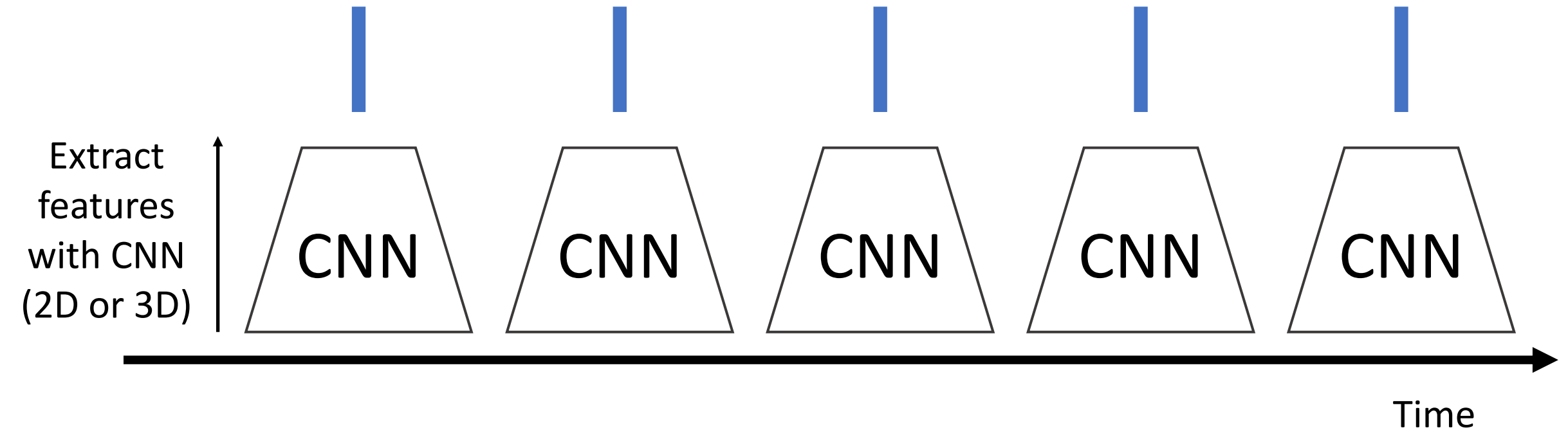
# Modeling long-term temporal structure

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?

We know how to handle sequences!  
How about recurrent networks?

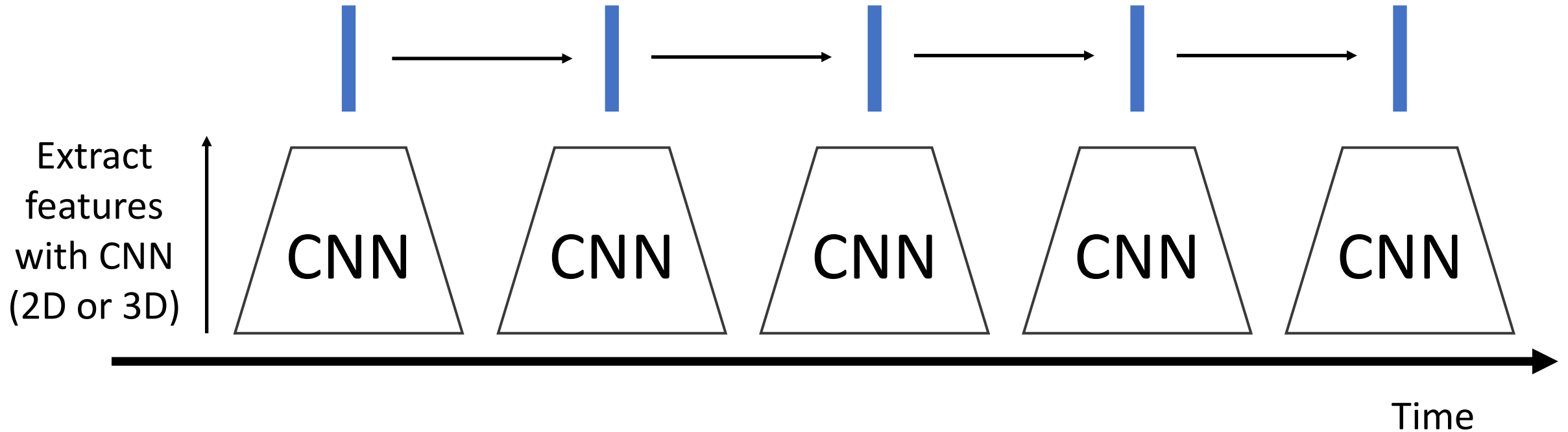


# Modeling long-term temporal structure



# Modeling long-term temporal structure

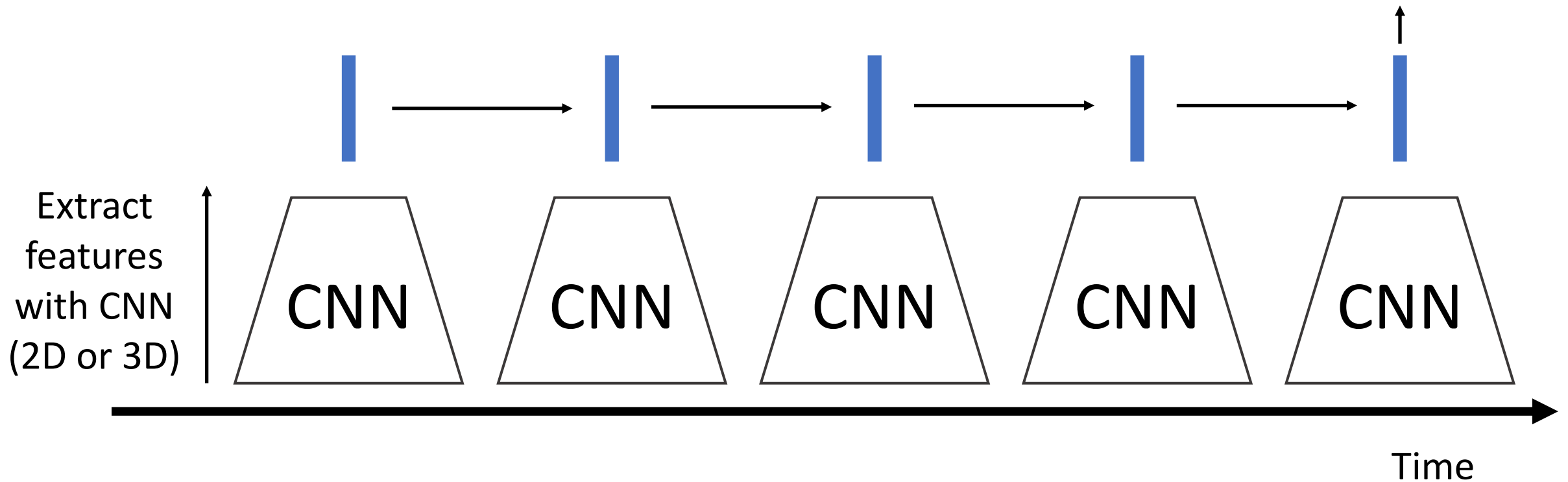
Process local features using recurrent network (e.g. LSTM)



# Modeling long-term temporal structure

Process local features using recurrent network (e.g. LSTM)

Many to one: One output at end of video

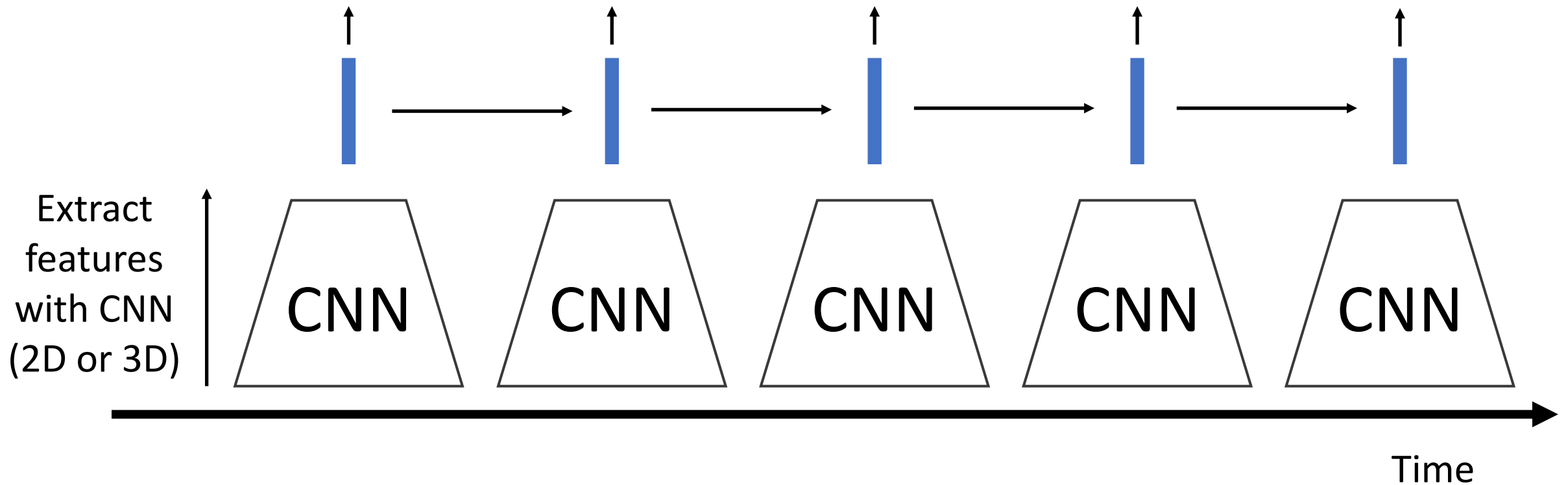




# Modeling long-term temporal structure

Process local features using recurrent network (e.g. LSTM)

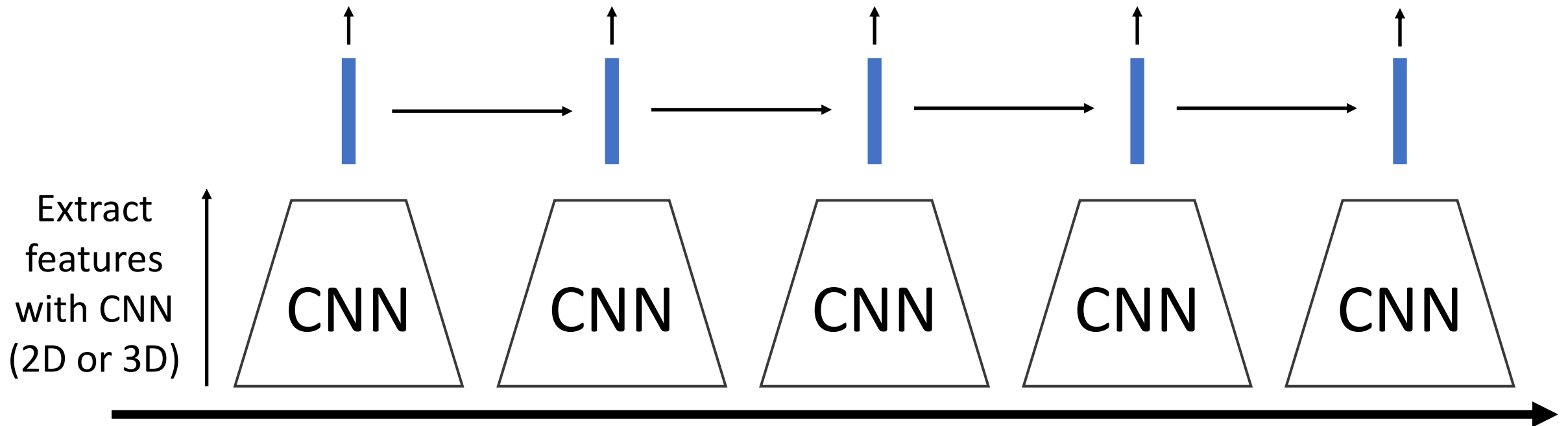
Many to many: one output per video frame



# Modeling long-term temporal structure

Process local features using recurrent network (e.g. LSTM)

Many to many: one output per video frame



Used 3D CNNs and LSTMs in 2011! Way ahead of its time

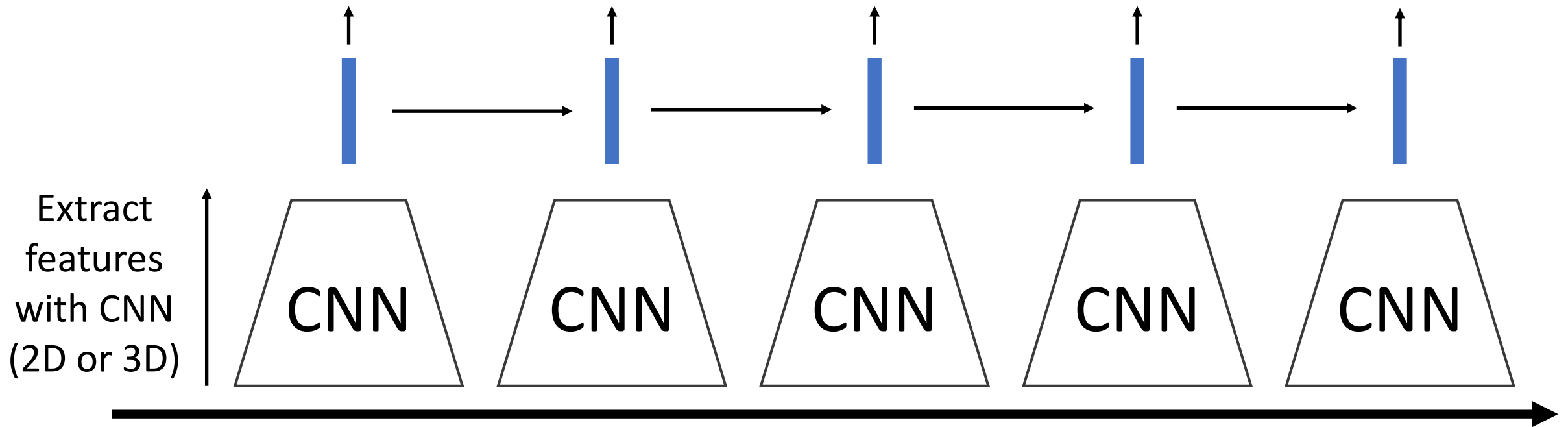
Baccouche et al, "Sequential Deep Learning for Human Action Recognition", **2011**

Time

# Modeling long-term temporal structure

Process local features using recurrent network (e.g. LSTM)

Many to many: one output per video frame



Used 3D CNNs and LSTMs in 2011! Way ahead of its time

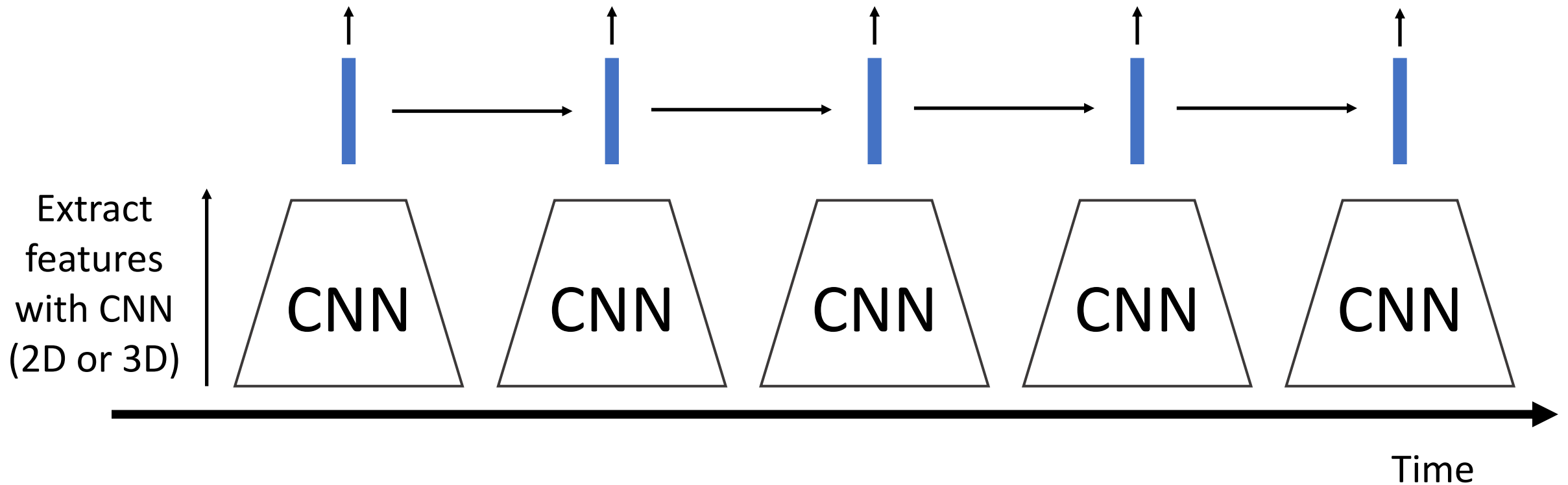
Baccouche et al, "Sequential Deep Learning for Human Action Recognition", **2011**

Time

# Modeling long-term temporal structure

Process local features using recurrent network (e.g. LSTM)

Many to many: one output per video frame

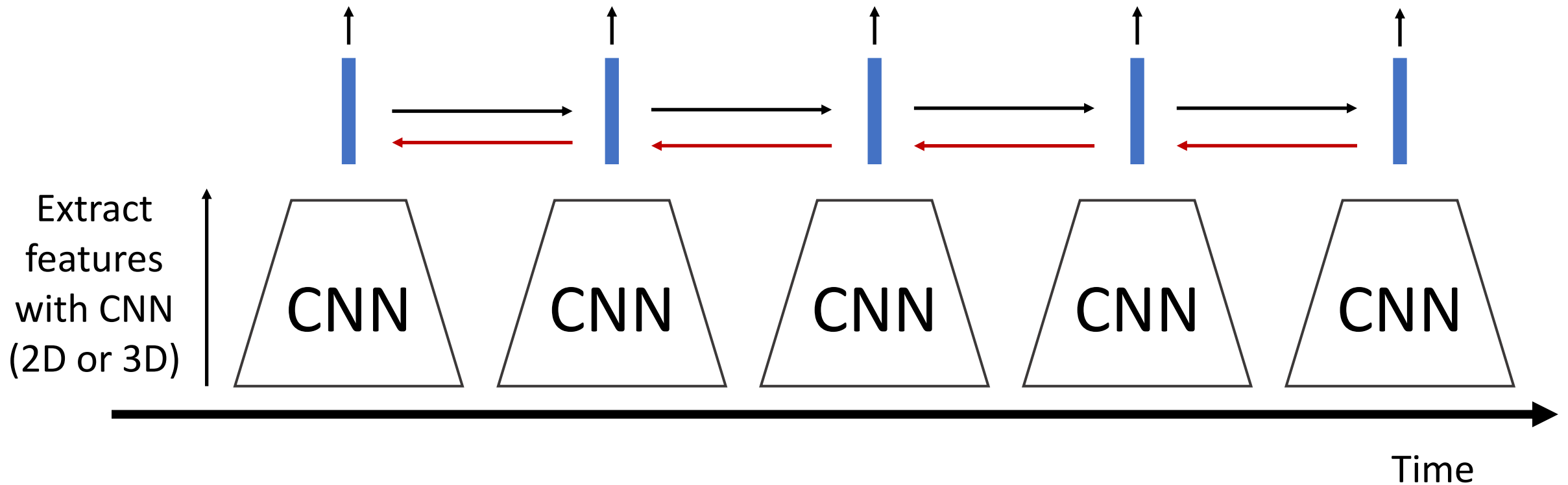


Baccouche et al, "Sequential Deep Learning for Human Action Recognition", 2011

Donahue et al, "Long-term recurrent convolutional networks for visual recognition and description", CVPR 2015

# Modeling long-term temporal structure

Sometimes don't backprop to CNN to save memory;  
pretrain and use it as a feature extractor



Baccouche et al, "Sequential Deep Learning for Human Action Recognition", 2011

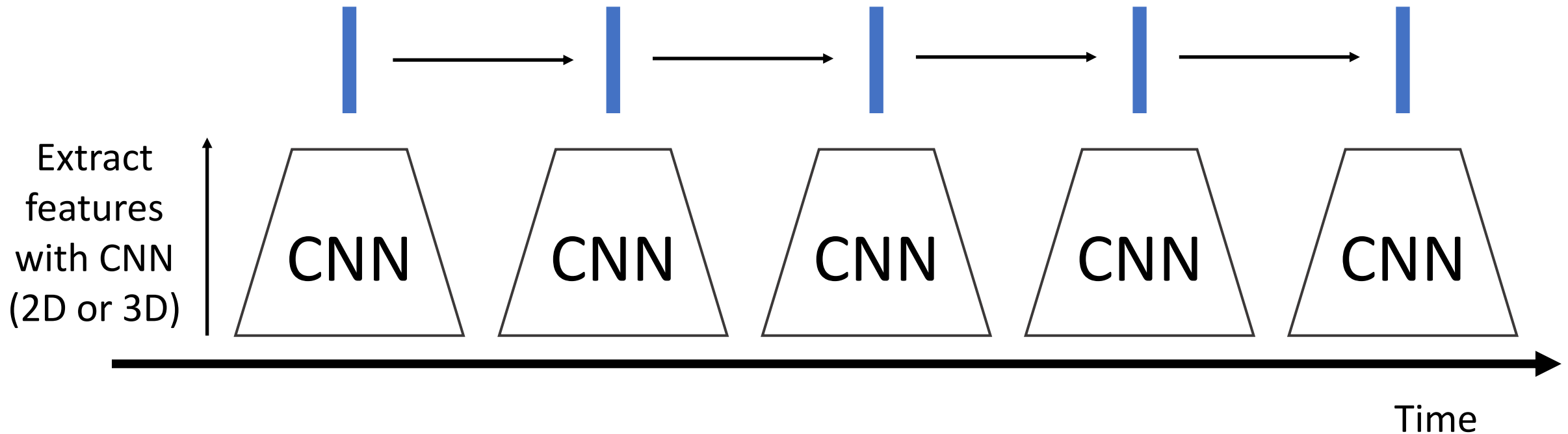
Donahue et al, "Long-term recurrent convolutional networks for visual recognition and description", CVPR 2015

# Modeling long-term temporal structure

Inside CNN: Each value 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?

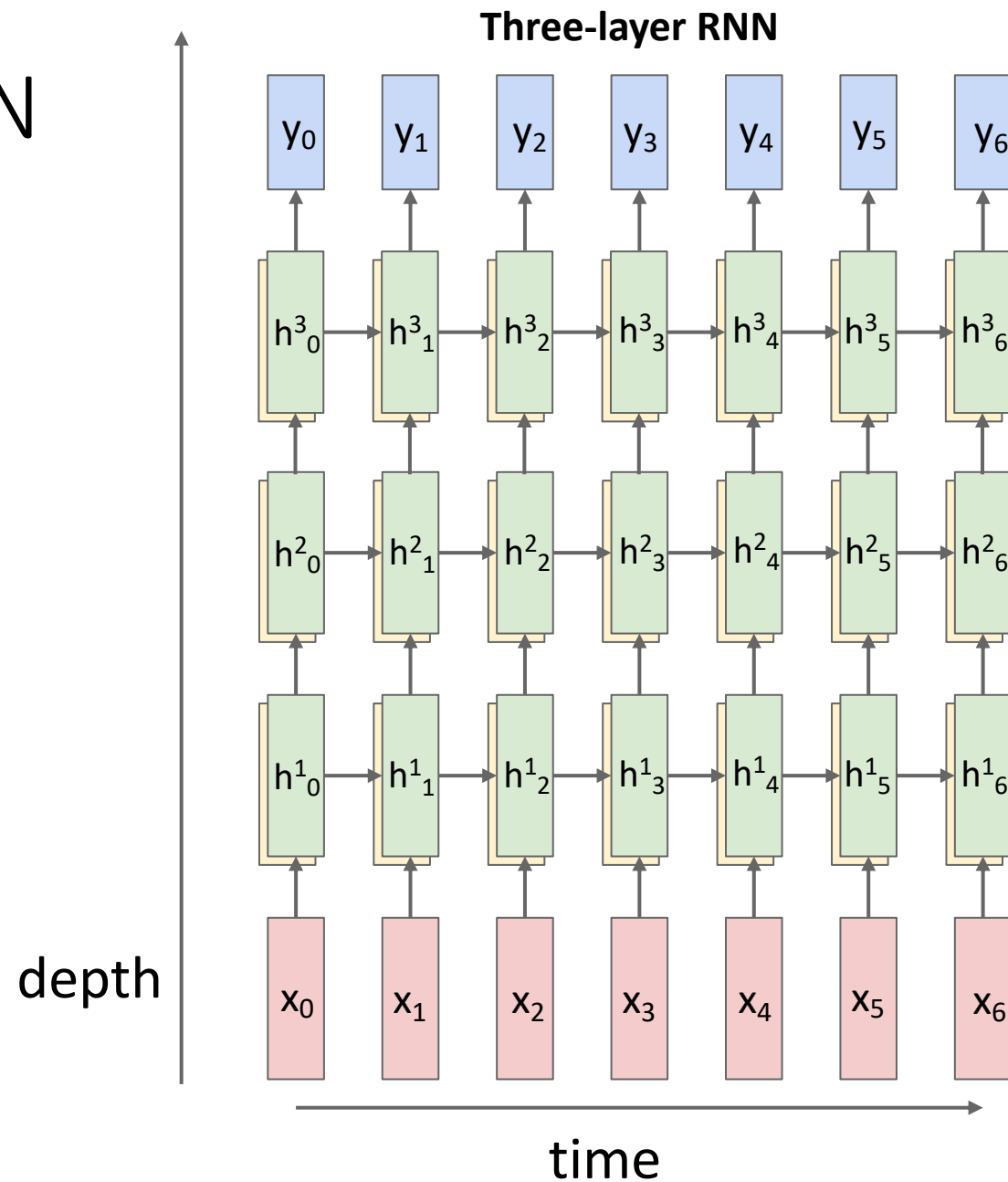


Baccouche et al, "Sequential Deep Learning for Human Action Recognition", 2011

Donahue et al, "Long-term recurrent convolutional networks for visual recognition and description", CVPR 2015

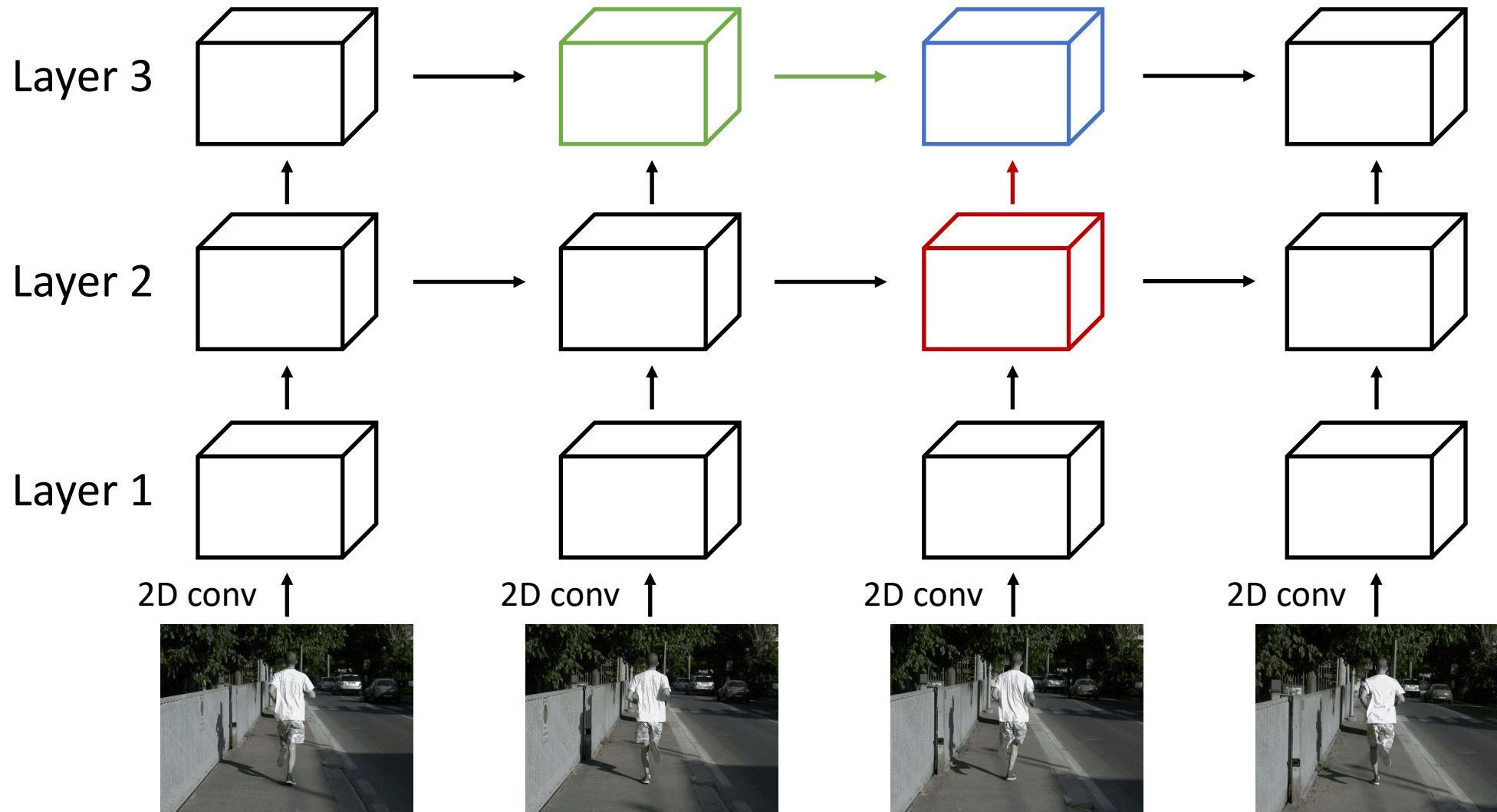
# Recall: Multi-layer RNN

We can use a similar structure to process videos!





# Recurrent Convolutional Network



Entire network  
uses 2D feature  
maps:  $C \times H \times W$

Each depends on  
two inputs:

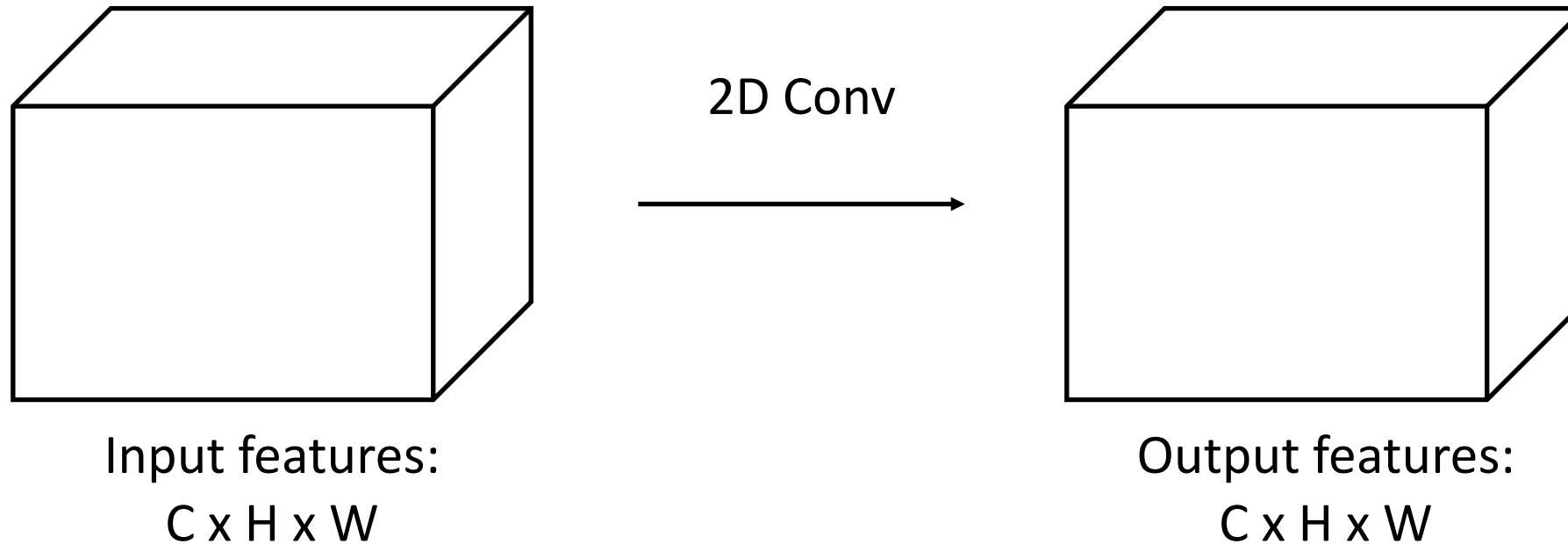
1. Same layer,  
previous timestep
2. Prev layer, same  
timestep

Use different  
weights at each  
layer, share weights  
across time

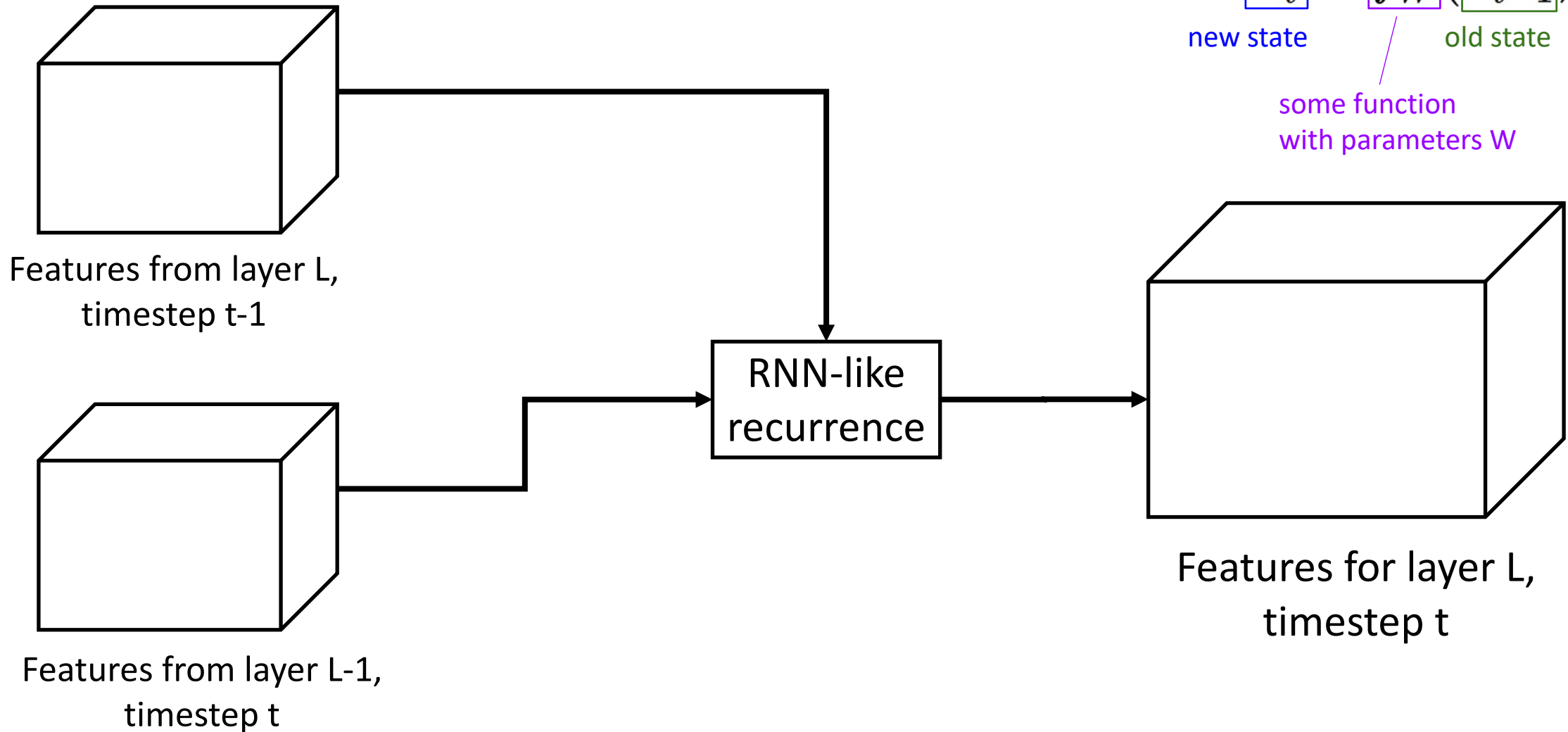
Ballas et al, "Delving Deeper into  
Convolutional Networks for Learning  
Video Representations", ICLR 2016

# Recurrent Convolutional Network

Normal 2D CNN:



# Recurrent Convolutional Network



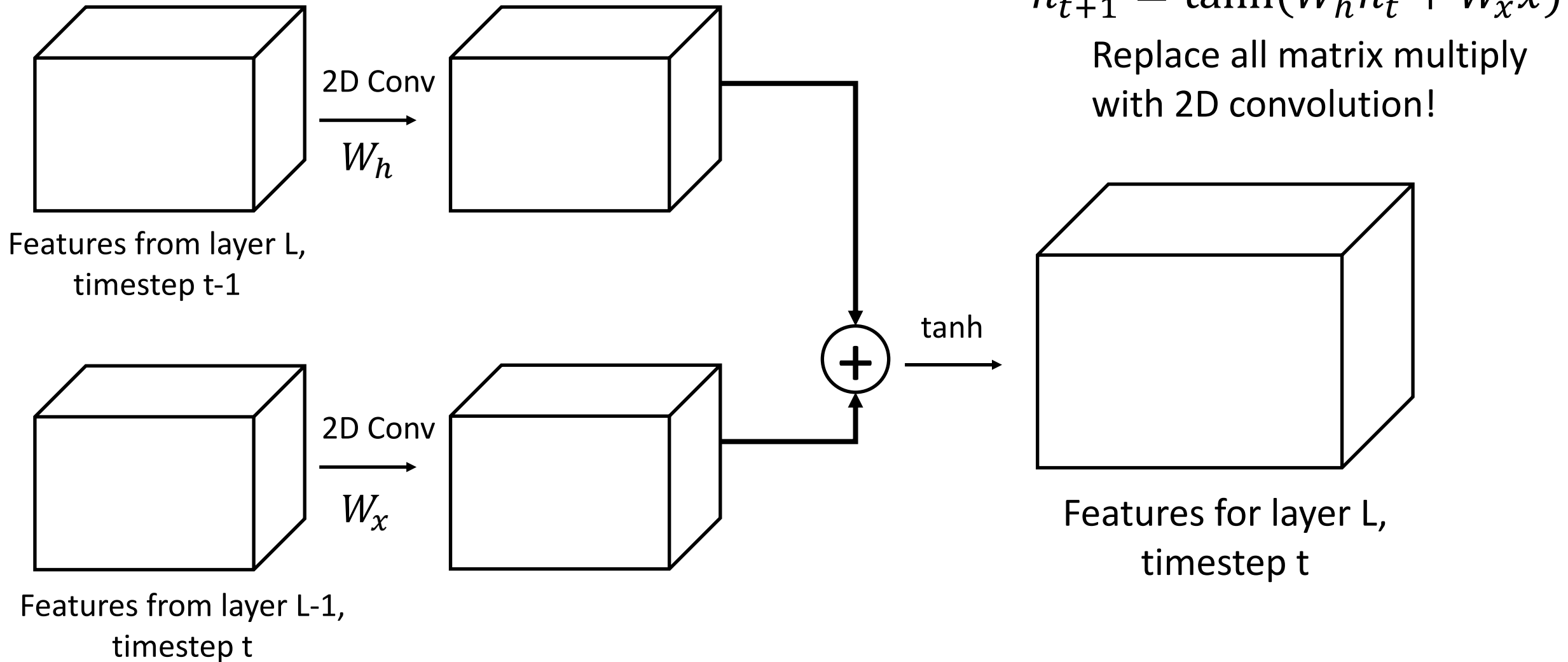
Recall: Recurrent Network

$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

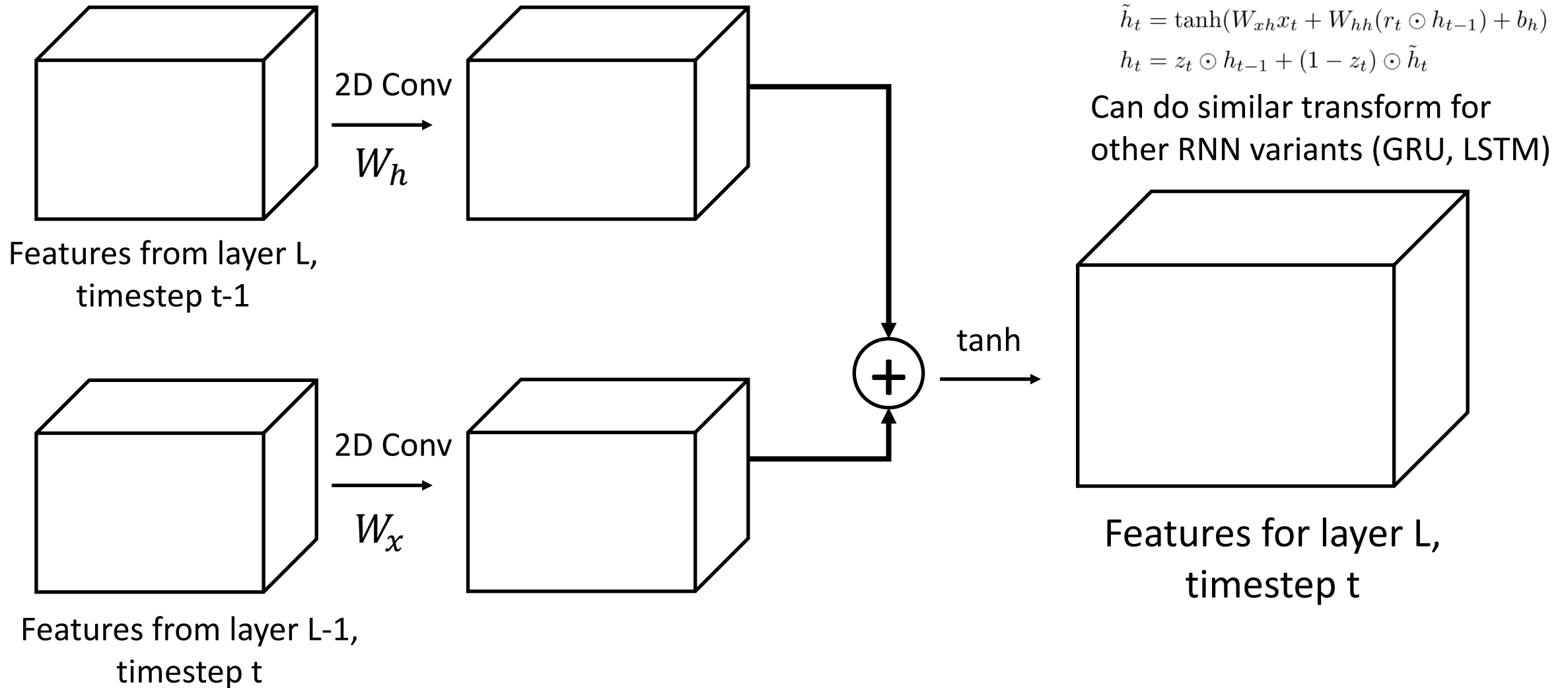
new state      old state

some function  
with parameters  $W$

# Recurrent Convolutional Network



# Recurrent Convolutional Network



## Recall: GRU

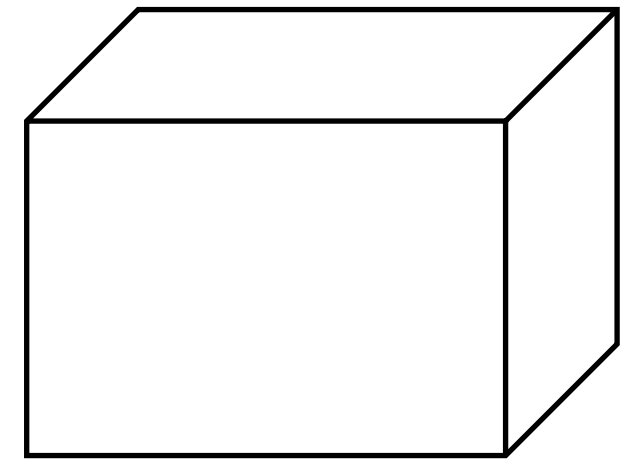
$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$$

$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$$

$$\tilde{h}_t = \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h)$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t$$

Can do similar transform for other RNN variants (GRU, LSTM)

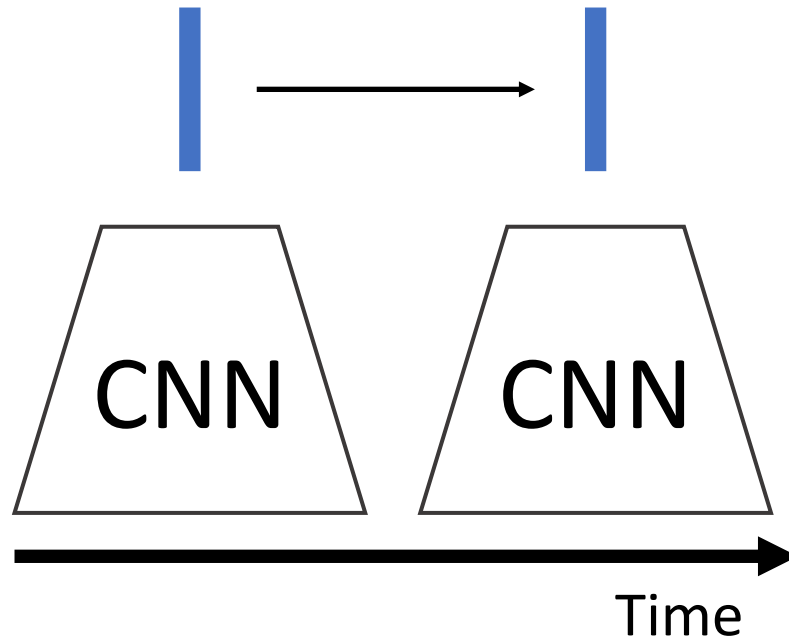


Features for layer L,  
timestep t

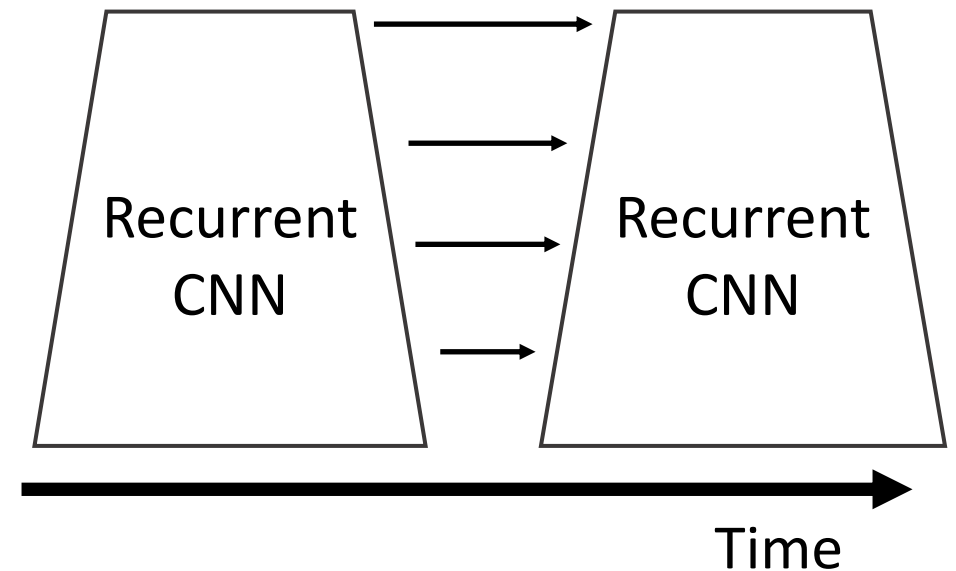
# Modeling long-term temporal structure

RNN: Infinite  
temporal extent  
(fully-connected)

CNN: finite  
temporal extent  
(convolutional)



Recurrent CNN: Infinite  
temporal extent  
(convolutional)



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

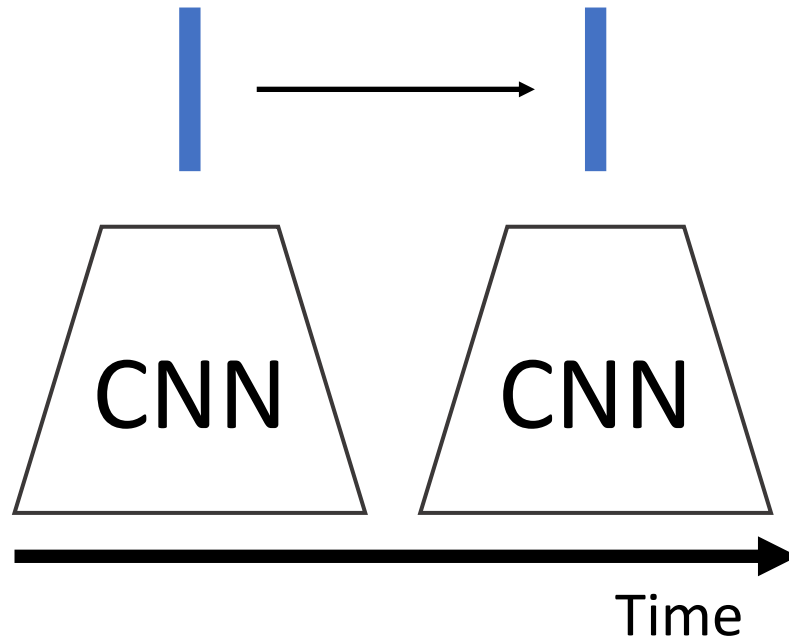
Ballas et al, "Delving Deeper into Convolutional Networks for Learning Video Representations", ICLR 2016

# Modeling long-term temporal structure

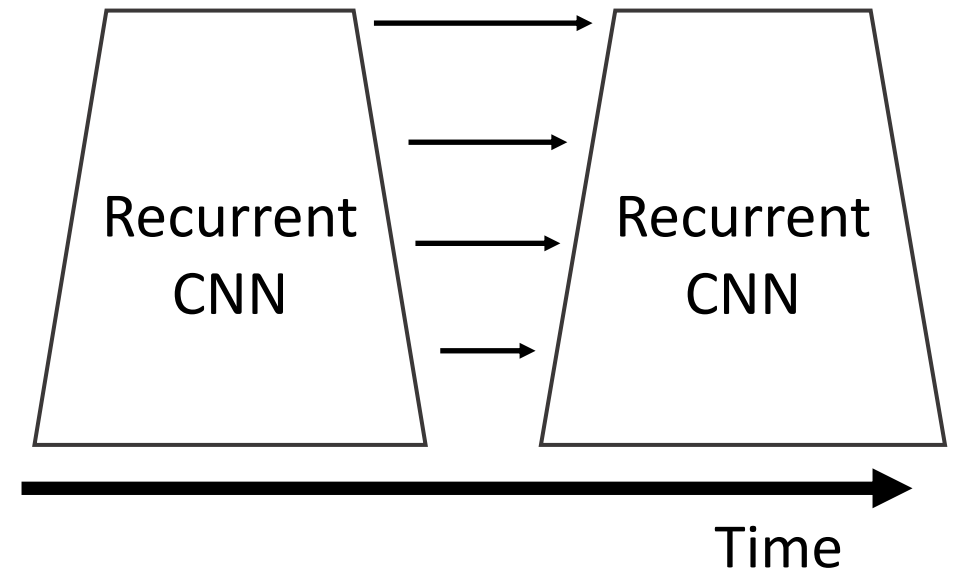
**Problem:** RNNs are slow for long sequences (can't be parallelized)

RNN: Infinite temporal extent (fully-connected)

CNN: finite temporal extent (convolutional)



Recurrent CNN: Infinite temporal extent (convolutional)



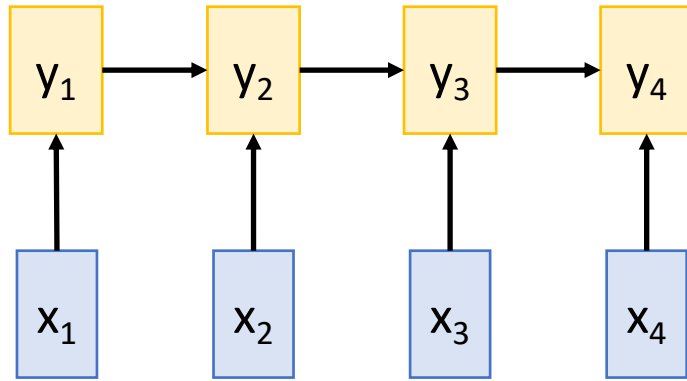
Baccouche et al, "Sequential Deep Learning for Human Action Recognition", 2011  
Donahue et al, "Long-term recurrent convolutional networks for visual recognition and description", CVPR 2015

Ballas et al, "Delving Deeper into Convolutional Networks for Learning Video Representations", ICLR 2016



# Recall: Different ways of processing sequences

## Recurrent Neural Network



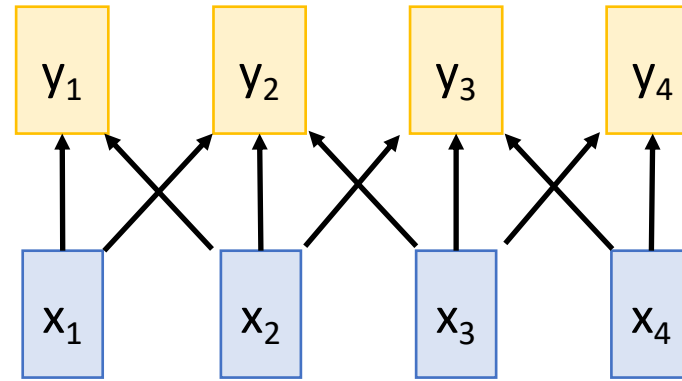
Works on **Ordered Sequences**

(+) **Good at long sequences:** After one RNN layer,  $h_T$  "sees" the whole sequence

(-) **Not parallelizable:** need to compute hidden states sequentially

In video: CNN+RNN, or recurrent CNN

## 1D Convolution



Works on **Multidimensional Grids**

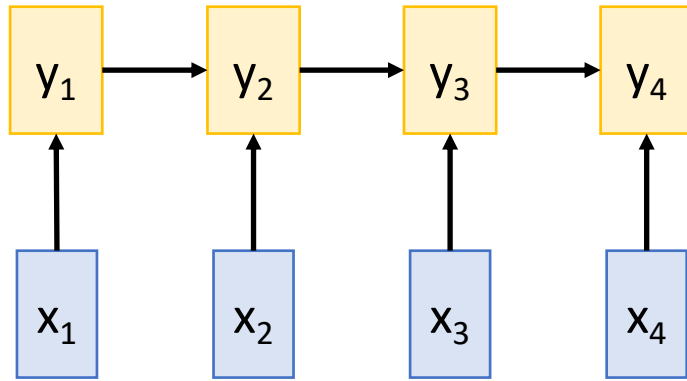
(-) **Bad at long sequences:** Need to stack many conv layers for outputs to "see" the whole sequence

(+) **Highly parallel:** Each output can be computed in parallel

In video: 3D convolution

# Recall: Different ways of processing sequences

## Recurrent Neural Network



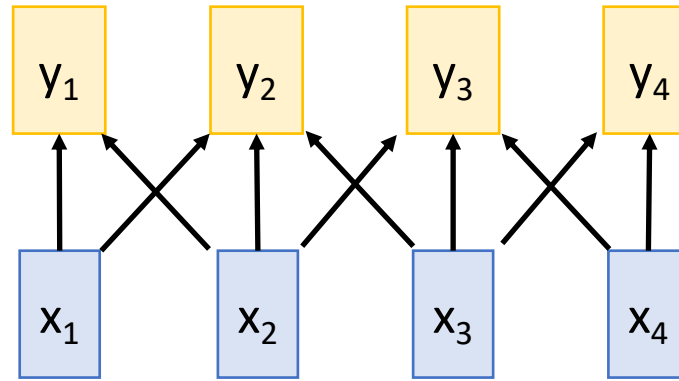
Works on **Ordered Sequences**

(+) **Good at long sequences:** After one RNN layer,  $h_T$  "sees" the whole sequence

(-) **Not parallelizable:** need to compute hidden states sequentially

In video: CNN+RNN, or recurrent CNN

## 1D Convolution



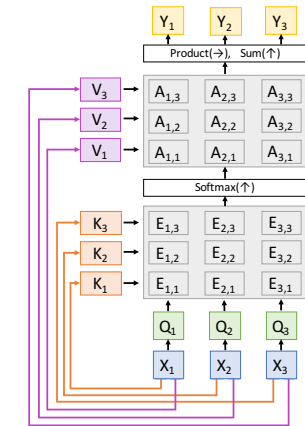
Works on **Multidimensional Grids**

(-) **Bad at long sequences:** Need to stack many conv layers for outputs to "see" the whole sequence

(+) **Highly parallel:** Each output can be computed in parallel

In video: 3D convolution

## Self-Attention



Works on **Sets of Vectors**

(-) **Good at long sequences:** after one self-attention layer, each output "sees" all inputs!

(+) **Highly parallel:** Each output can be computed in parallel

(-) **Very memory intensive**

In video: ????

# Recall: Self-Attention

**Input:** Set of vectors  $x_1, \dots, x_N$

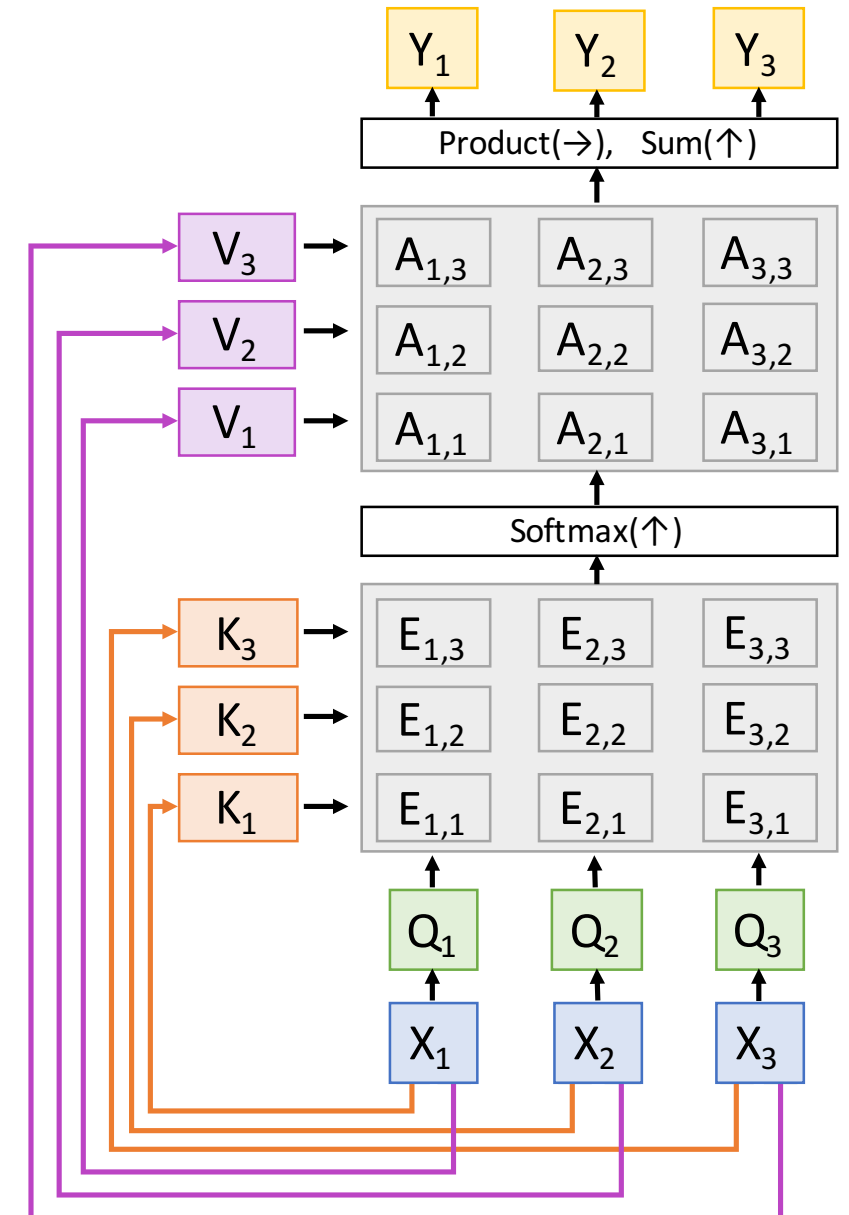
**Keys, Queries, Values:** Project each  $x$  to a key, query, and value using linear layer

**Affinity matrix:** Compare each pair of  $x$ , (using scaled dot-product between keys and values) and normalize using softmax

**Output:** Weighted sum of values, with weights given by affinity matrix

Features in 3D CNN:  $C \times T \times H \times W$

Interpret as a set of THW vectors of dim  $C$

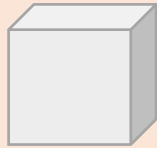


# Spatio-Temporal Self-Attention (Nonlocal Block)

Input clip



3D  
CNN



Features:  
 $C \times T \times H \times W$

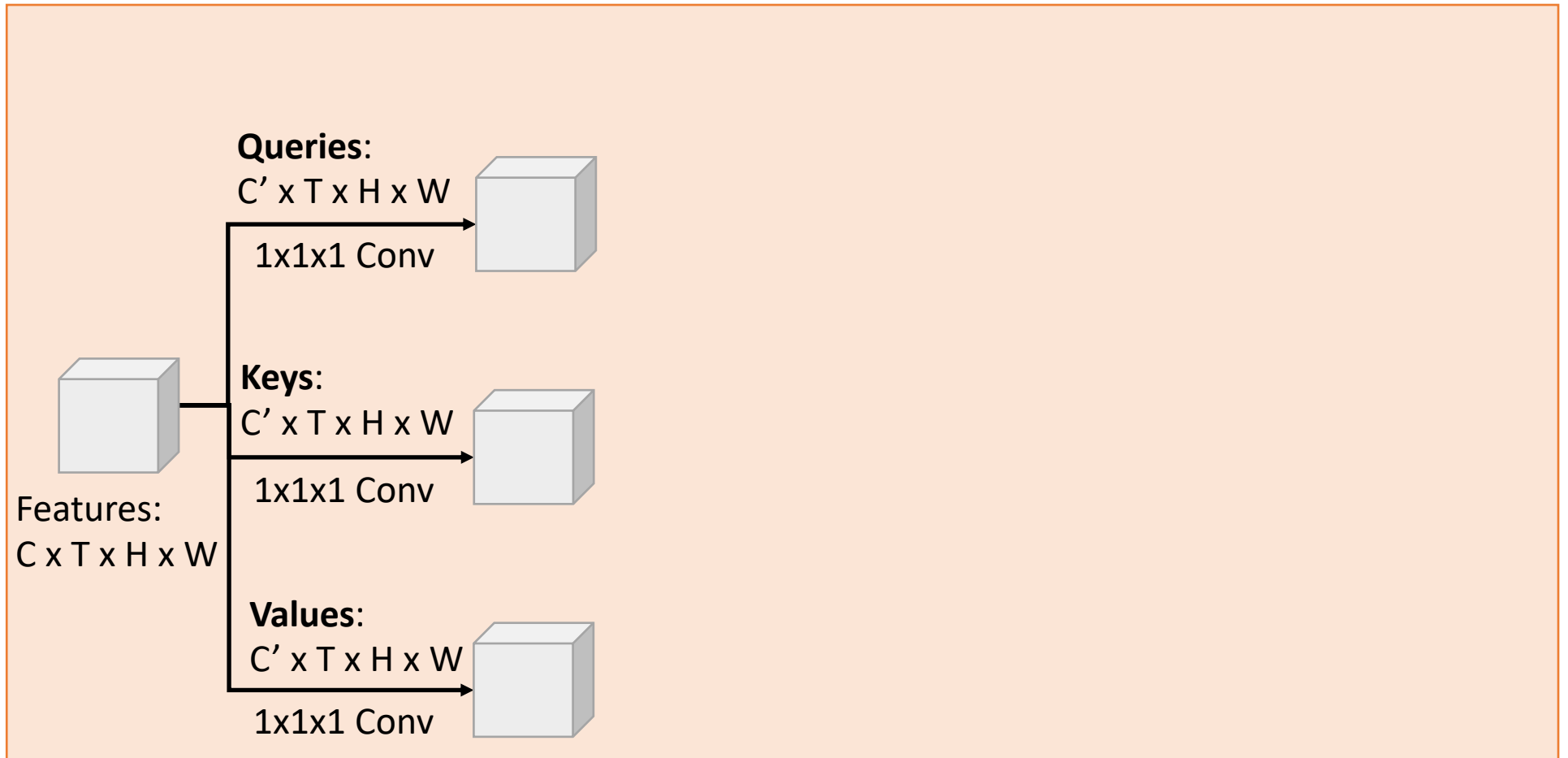
Nonlocal Block

# Spatio-Temporal Self-Attention (Nonlocal Block)

Input clip



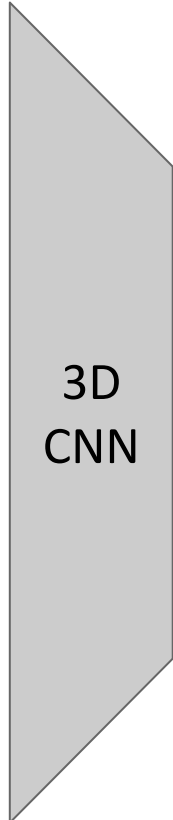
3D  
CNN



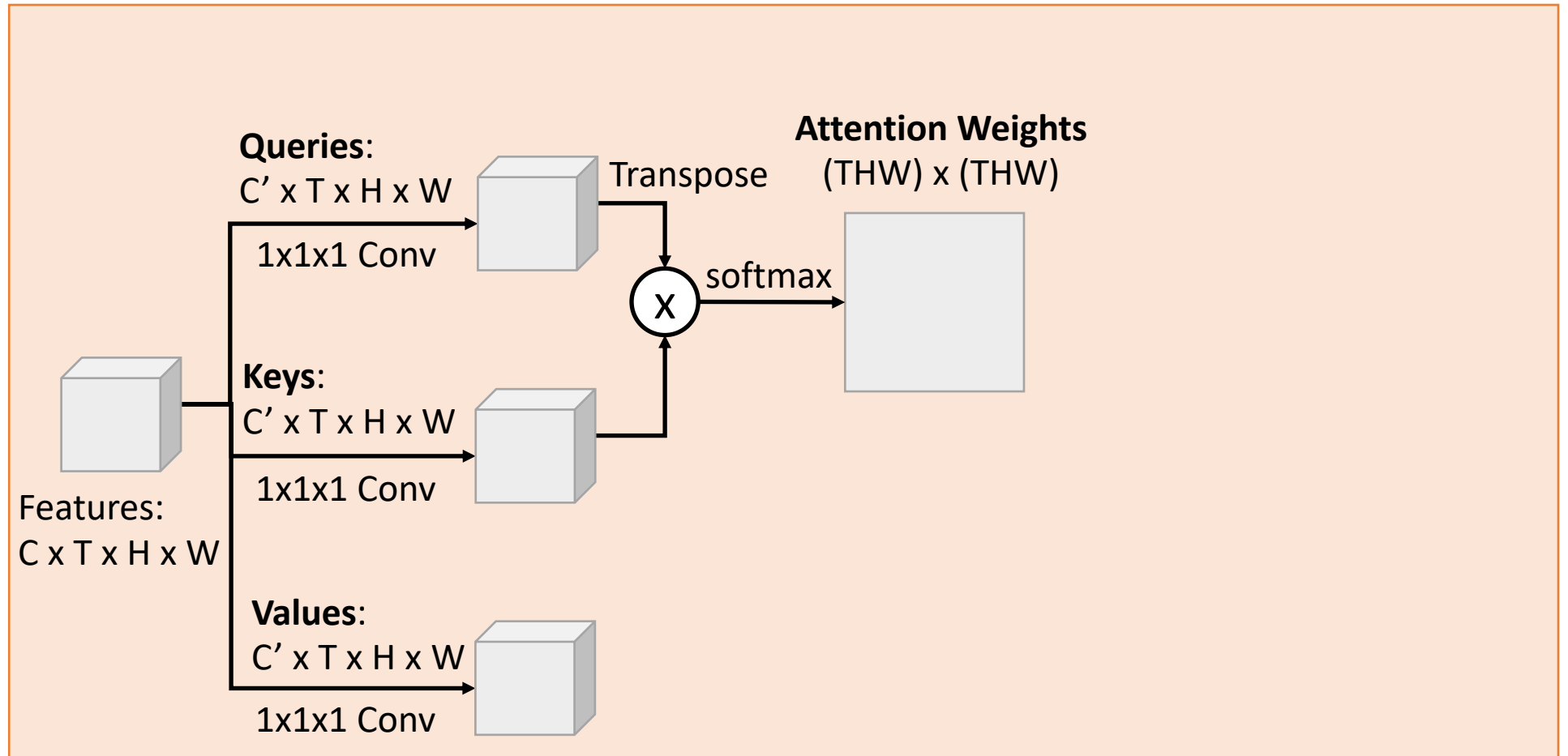
Nonlocal Block

# Spatio-Temporal Self-Attention (Nonlocal Block)

Input clip

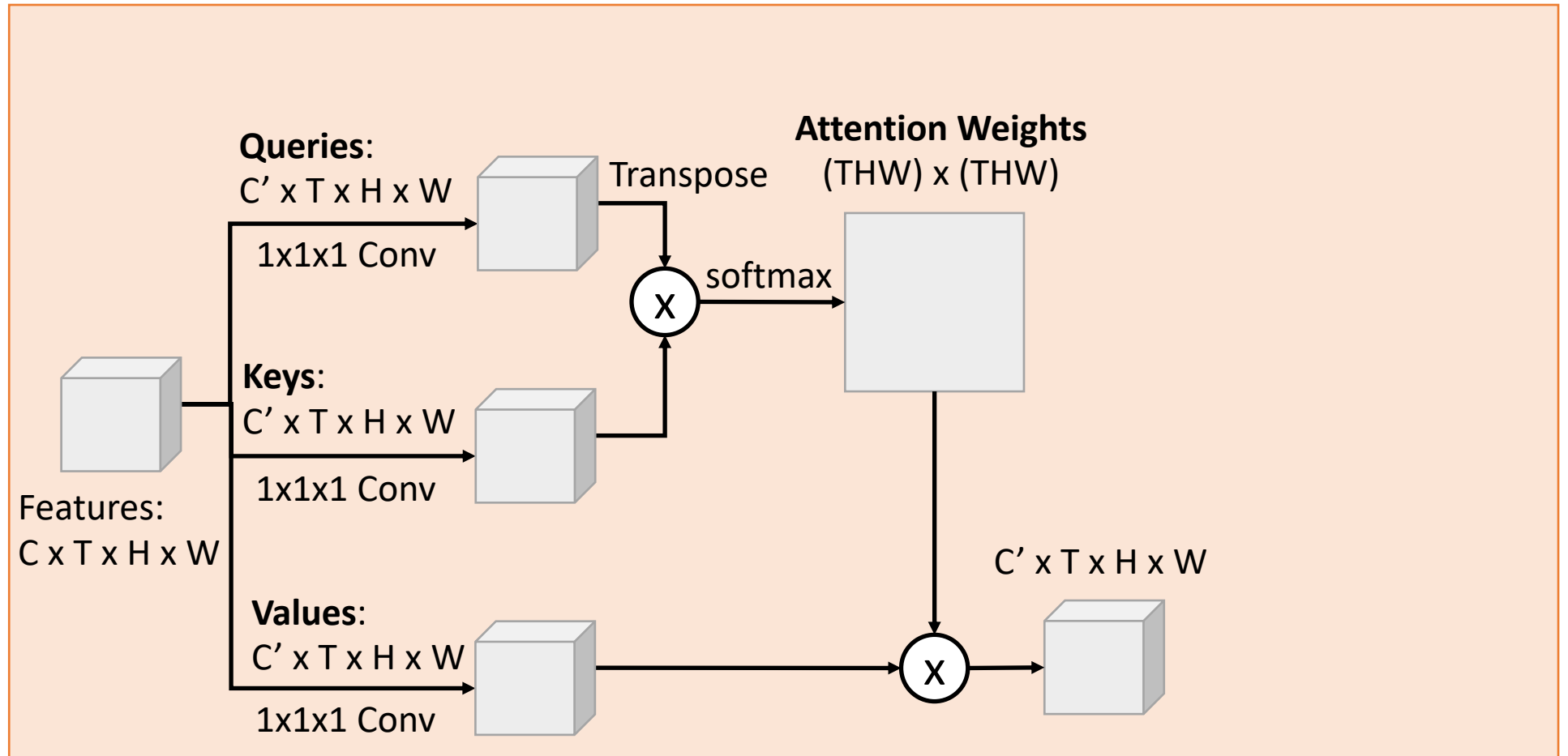
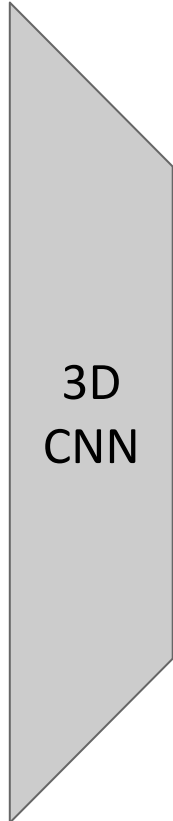
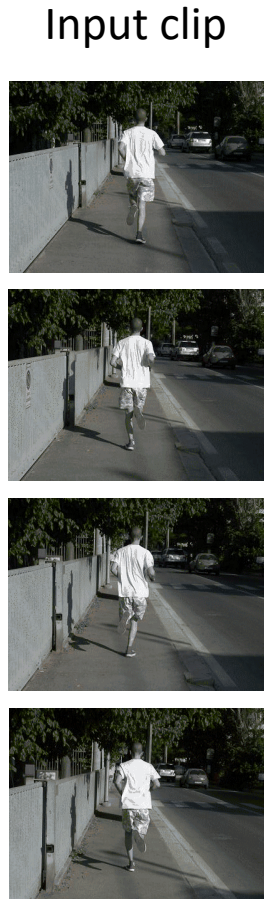


3D  
CNN



Nonlocal Block

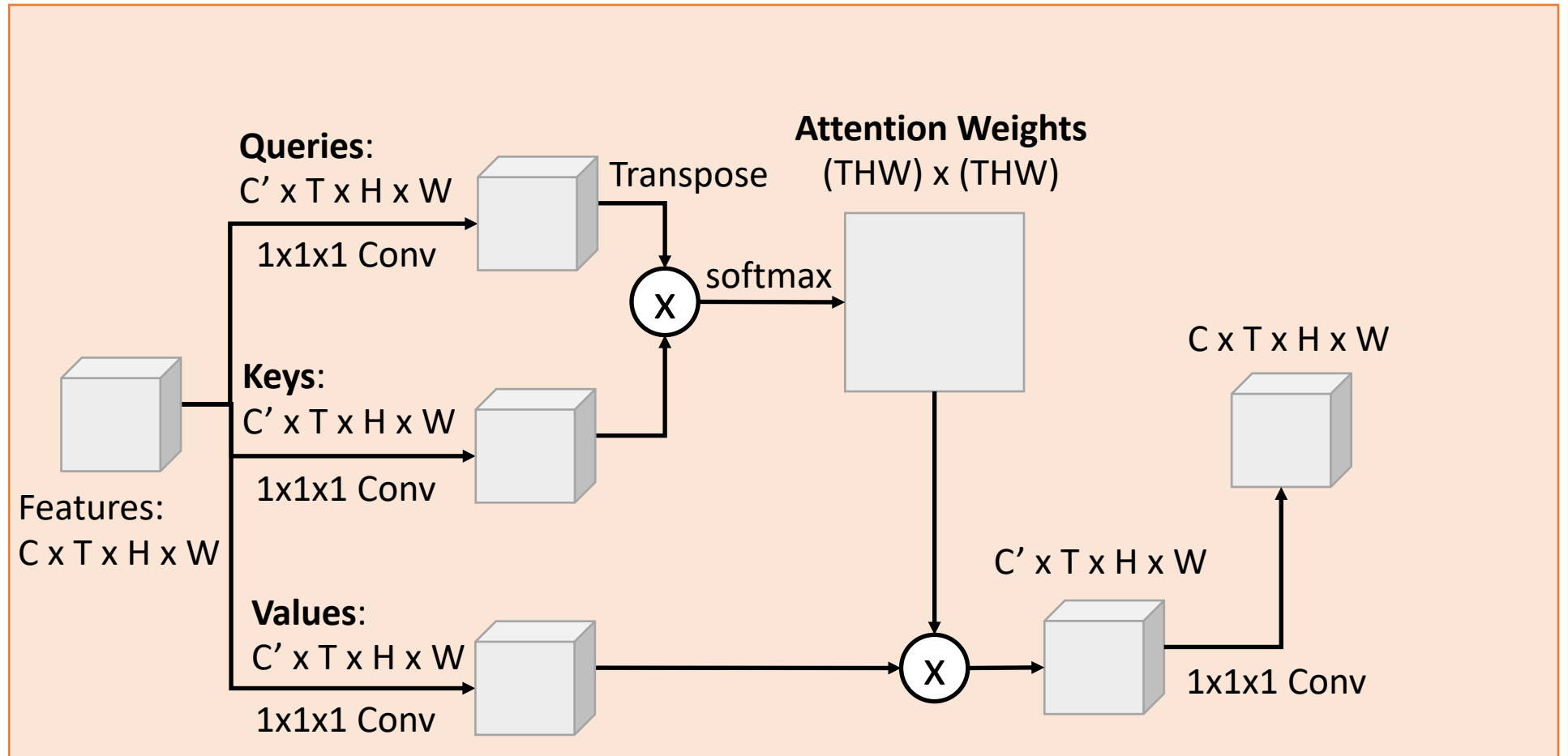
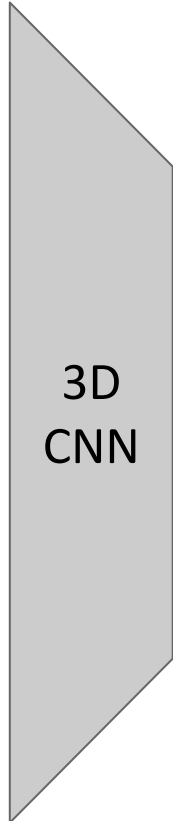
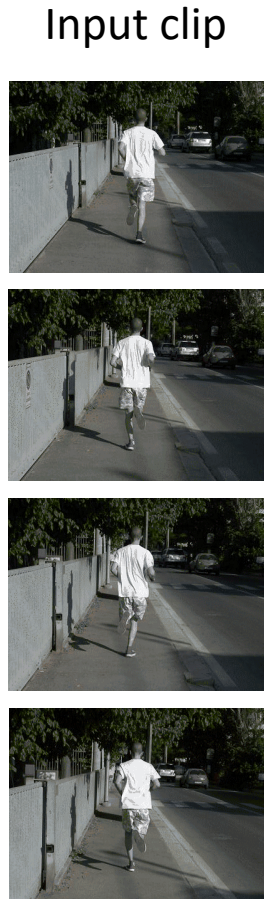
# Spatio-Temporal Self-Attention (Nonlocal Block)



Nonlocal Block

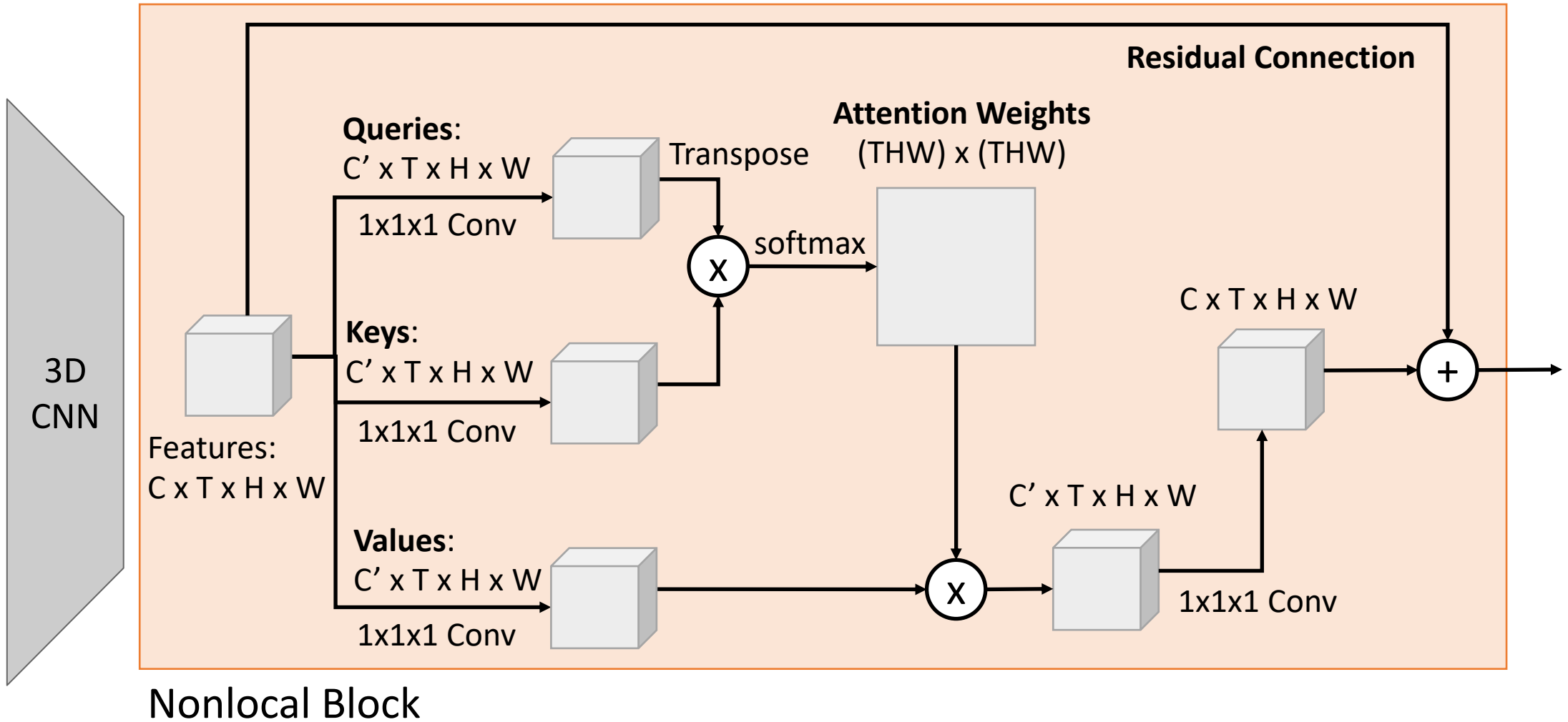


# Spatio-Temporal Self-Attention (Nonlocal Block)

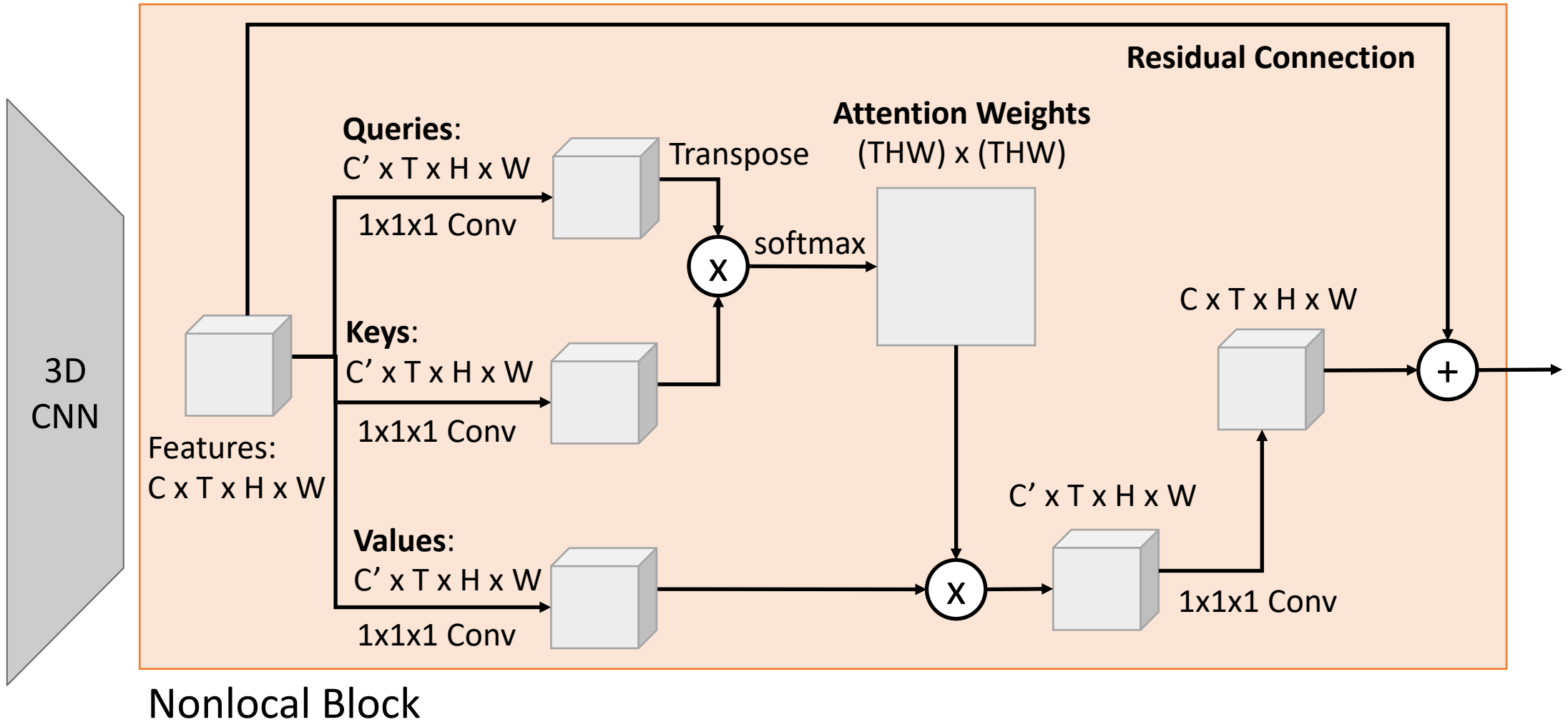
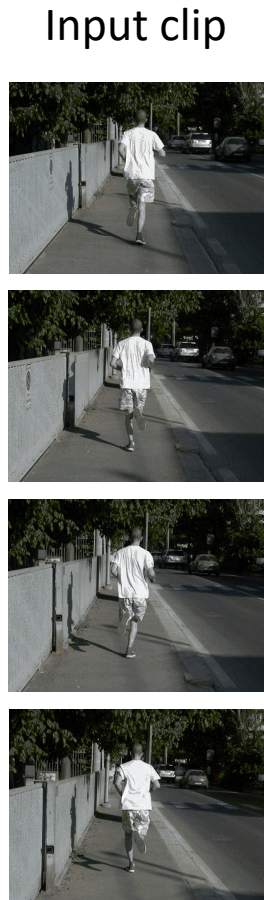


Nonlocal Block

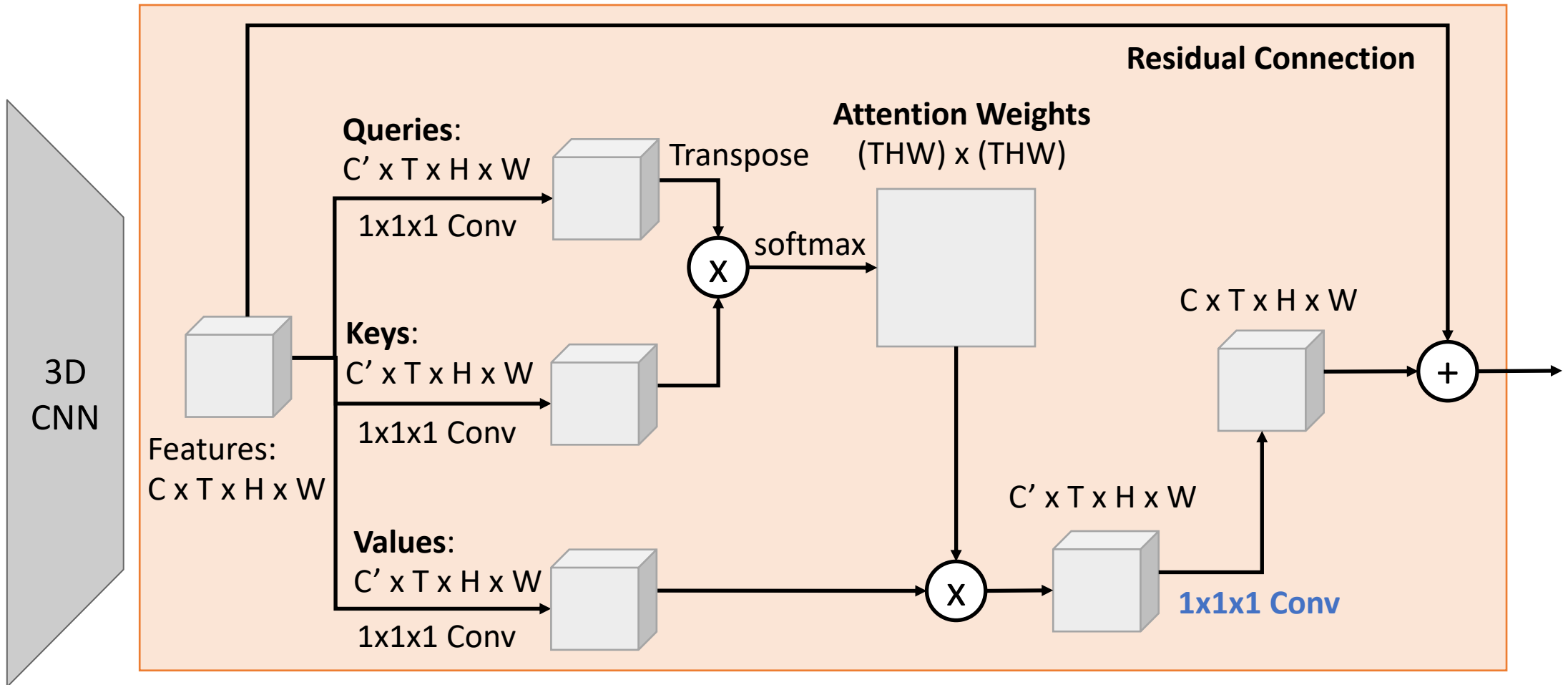
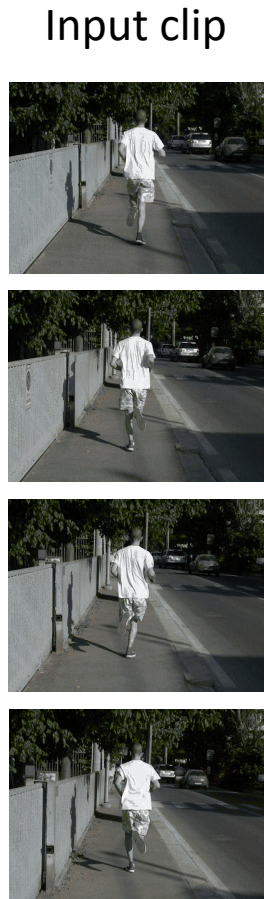
# Spatio-Temporal Self-Attention (Nonlocal Block)



# Spatio-Temporal Self-Attention (Nonlocal Block)



# Spatio-Temporal Self-Attention (Nonlocal Block)

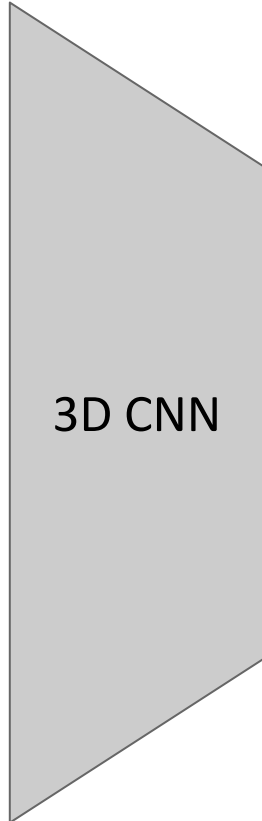


**Nonlocal Block** Trick: Initialize **last conv** to 0, then entire block computes identity. Can insert into existing 3D CNNs

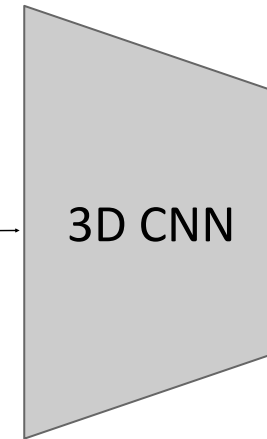
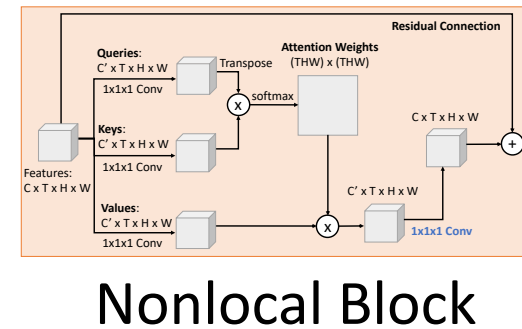
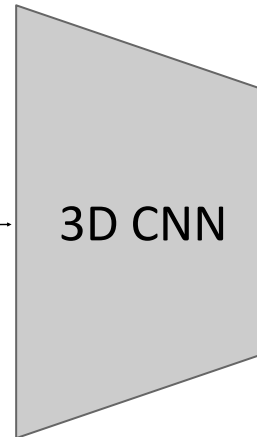
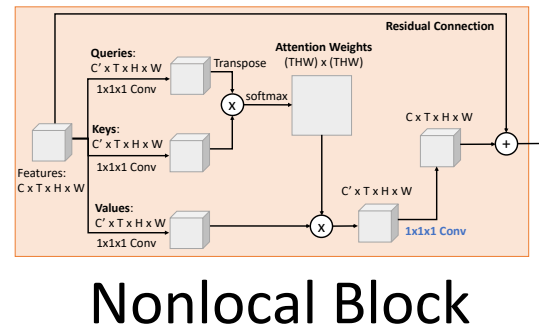
In practice, actually insert BatchNorm layer after final conv, and initialize scale parameter of BN layer to 0 rather than setting conv weight to 0

# Spatio-Temporal Self-Attention (Nonlocal Block)

Input clip



We can add nonlocal blocks into existing 3D CNN architectures.  
But what is the best 3D CNN architecture?



Running

# Inflating 2D Networks to 3D (I3D)

There has been a lot of work on architectures for images.  
Can we reuse image architectures for video?

**Idea:** take a 2D CNN architecture.

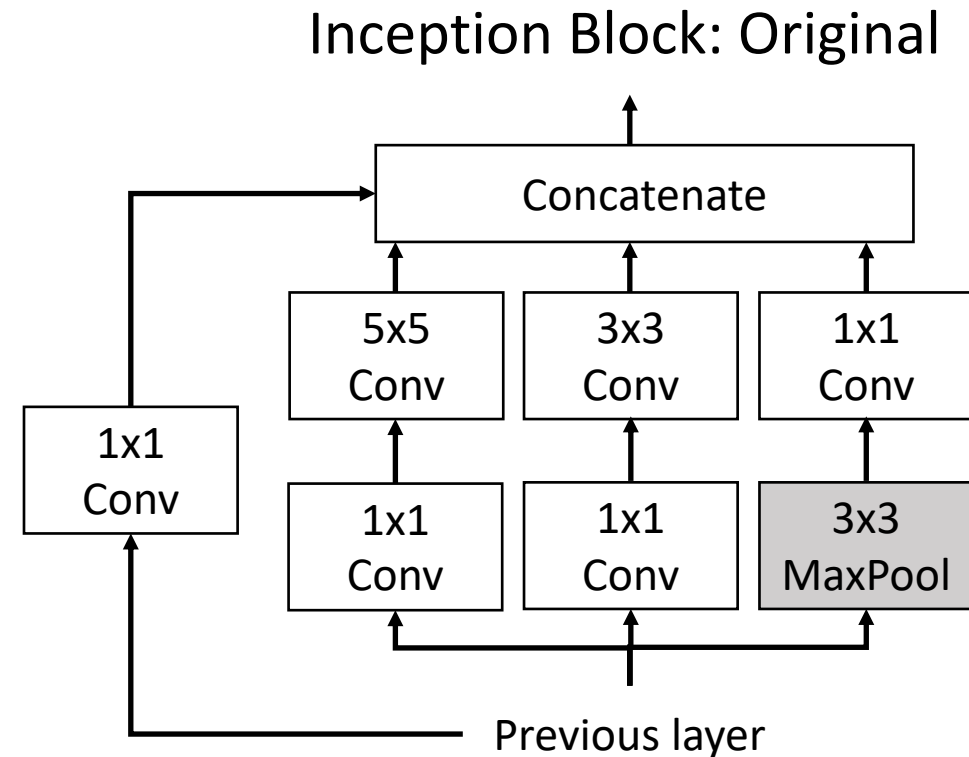
Replace each 2D  $K_h \times K_w$  conv/pool  
layer with a 3D  $K_t \times K_h \times K_w$  version

# Inflating 2D Networks to 3D (I3D)

There has been a lot of work on architectures for images.  
Can we reuse image architectures for video?

**Idea:** take a 2D CNN architecture.

Replace each 2D  $K_h \times K_w$  conv/pool layer with a 3D  $K_t \times K_h \times K_w$  version



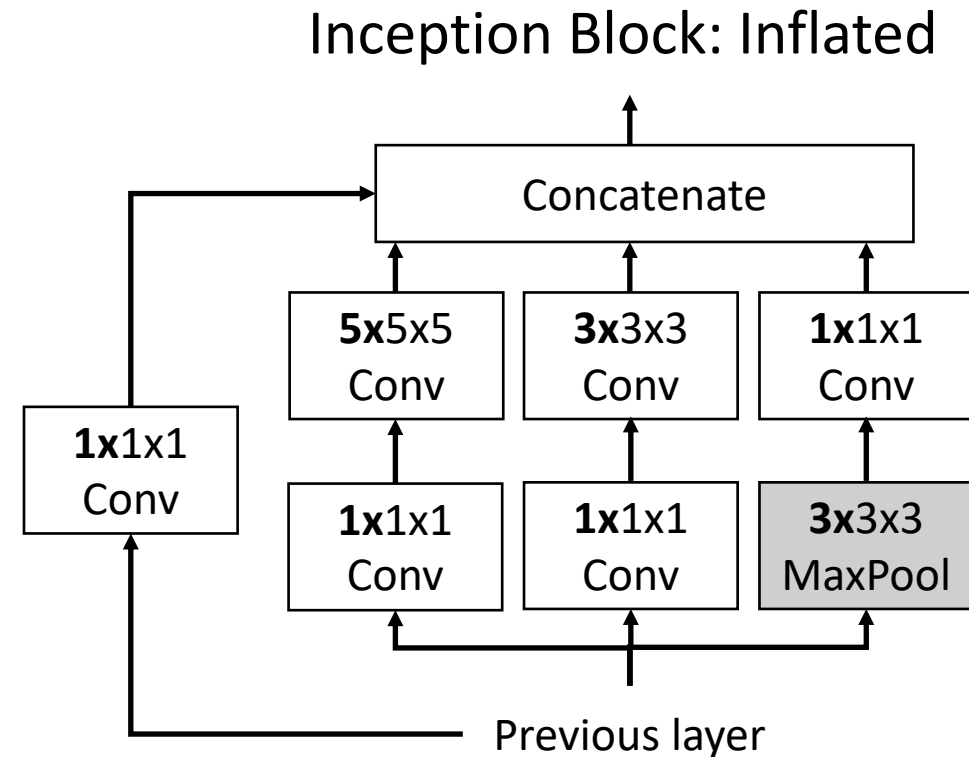


# Inflating 2D Networks to 3D (I3D)

There has been a lot of work on architectures for images.  
Can we reuse image architectures for video?

**Idea:** take a 2D CNN architecture.

Replace each 2D  $K_h \times K_w$  conv/pool layer with a 3D  $K_t \times K_h \times K_w$  version



# Inflating 2D Networks to 3D (I3D)

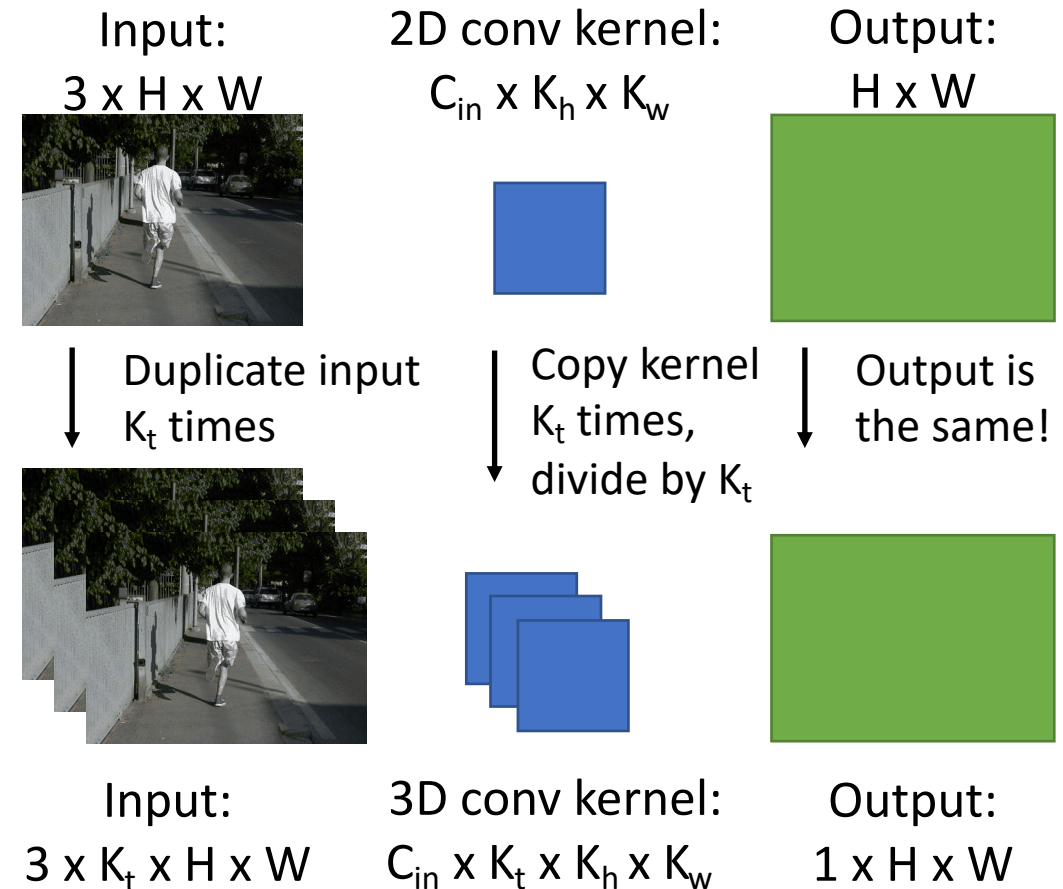
There has been a lot of work on architectures for images.  
Can we reuse image architectures for video?

**Idea:** take a 2D CNN architecture.

Replace each 2D  $K_h \times K_w$  conv/pool layer with a 3D  $K_t \times K_h \times K_w$  version

Can use weights of 2D conv to initialize 3D conv: copy  $K_t$  times in space and divide by  $K_t$

This gives the same result as 2D conv given “constant” video input



# Inflating 2D Networks to 3D (I3D)

There has been a lot of work on architectures for images.  
Can we reuse image architectures for video?

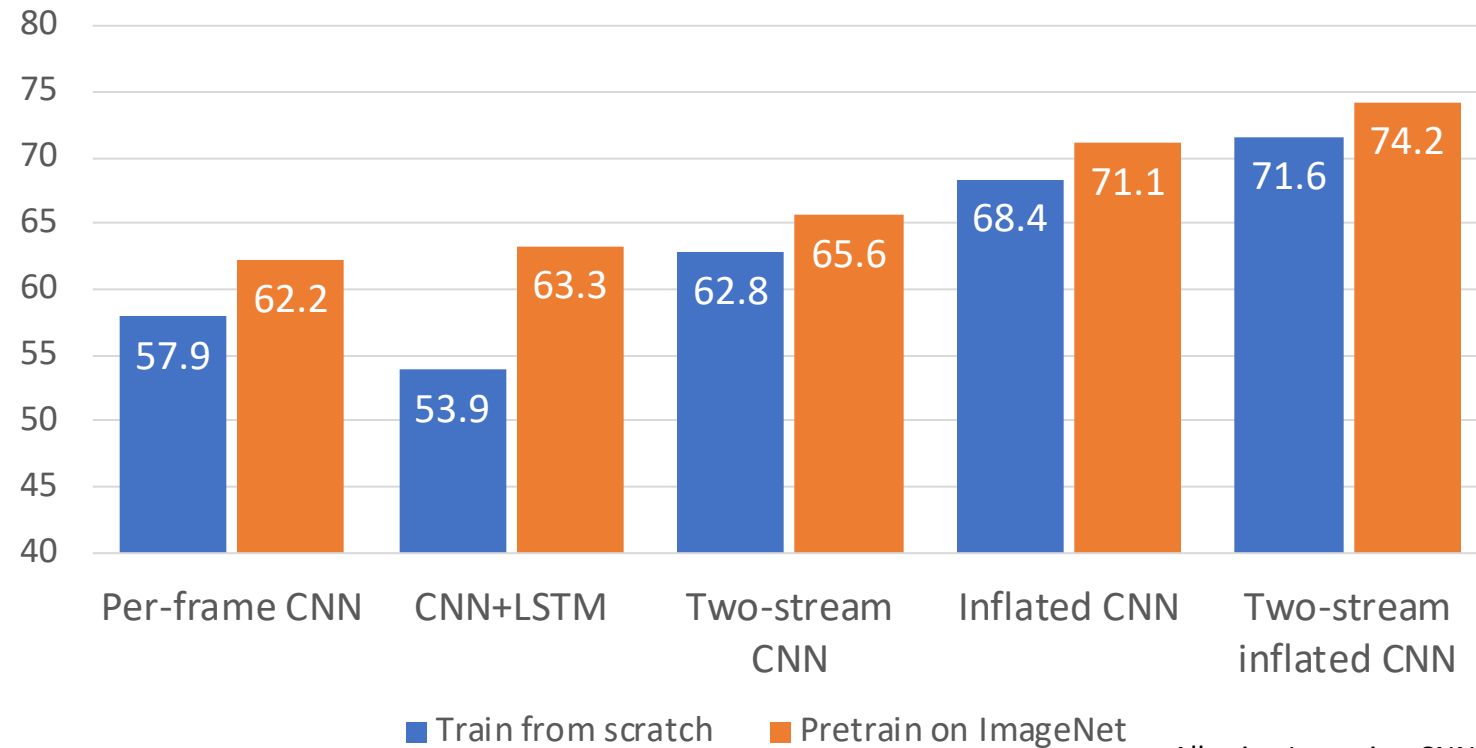
**Idea:** take a 2D CNN architecture.

Replace each 2D  $K_h \times K_w$  conv/pool layer with a 3D  $K_t \times K_h \times K_w$  version

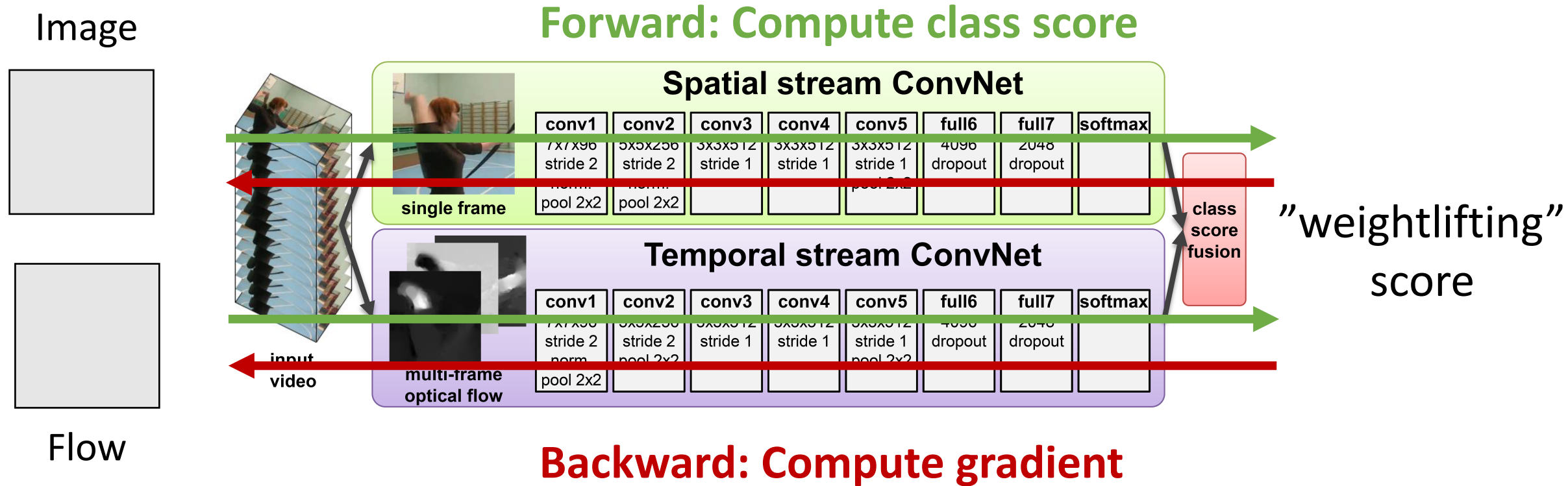
Can use weights of 2D conv to initialize 3D conv: copy  $K_t$  times in space and divide by  $K_t$

This gives the same result as 2D conv given “constant” video input

Top-1 Accuracy on Kinetics-400



# Visualizing Video Models

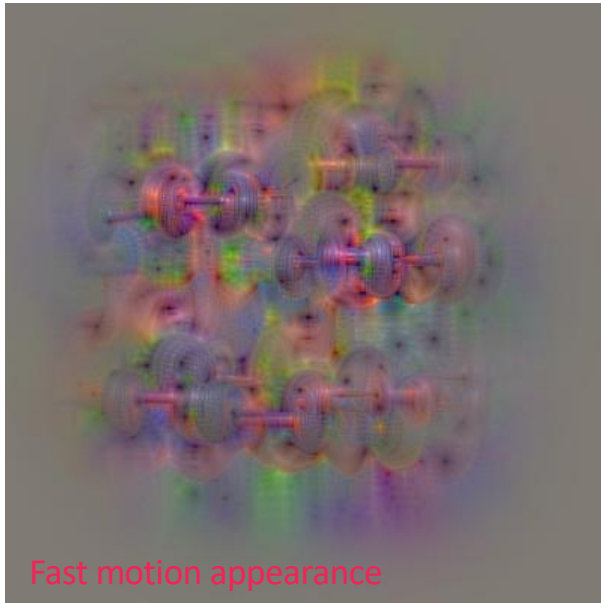


Add a term to encourage spatially smooth flow; tune penalty to pick out “slow” vs “fast” motion

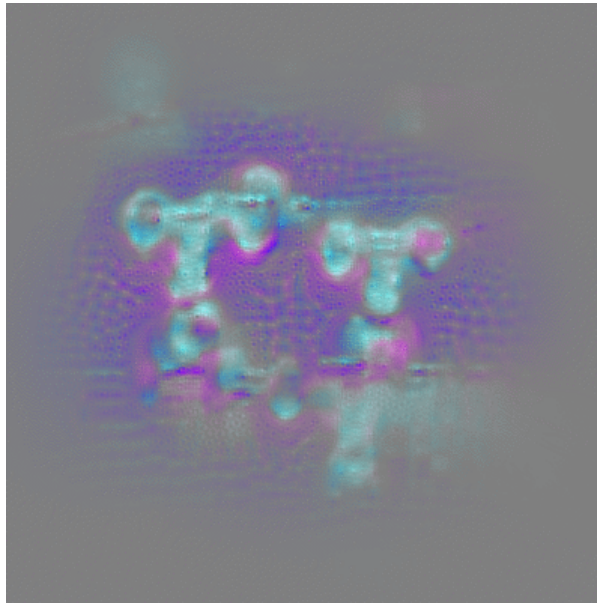
Figure credit: Simonyan and Zisserman, “Two-stream convolutional networks for action recognition in videos”, NeurIPS 2014  
Feichtenhofer et al, “What have we learned from deep representations for action recognition?”, CVPR 2018  
Feichtenhofer et al, “Deep insights into convolutional networks for video recognition?”, IJCV 2019.

# Can you guess the action?

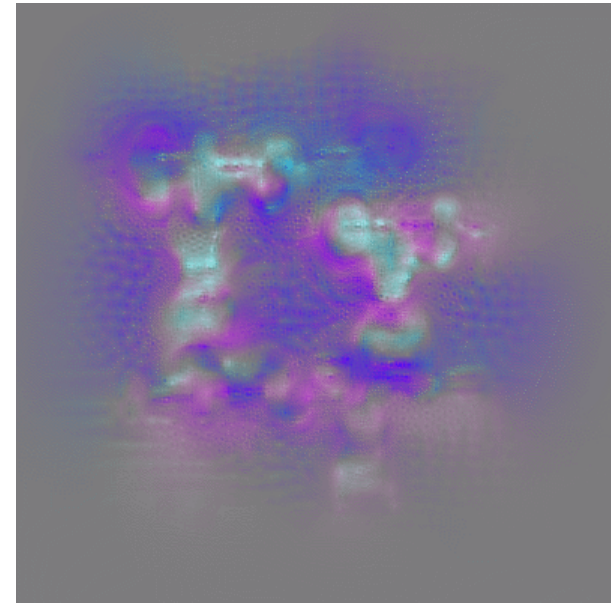
Appearance



"Slow" motion



"Fast" motion





# Can you guess the action? Weightlifting

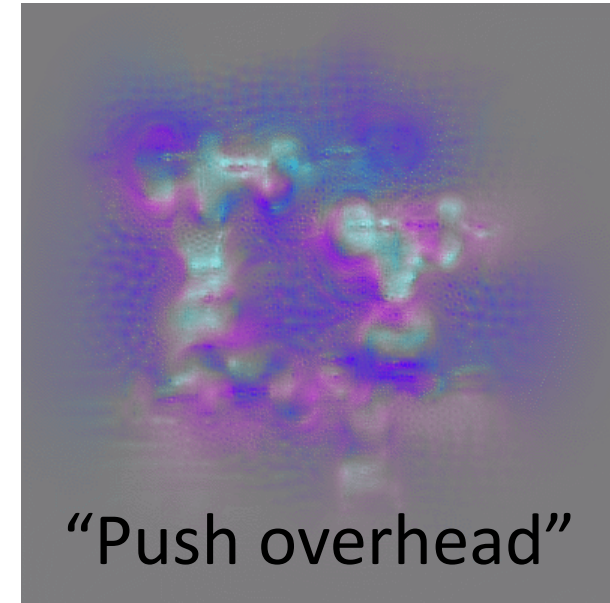
Appearance



"Slow" motion

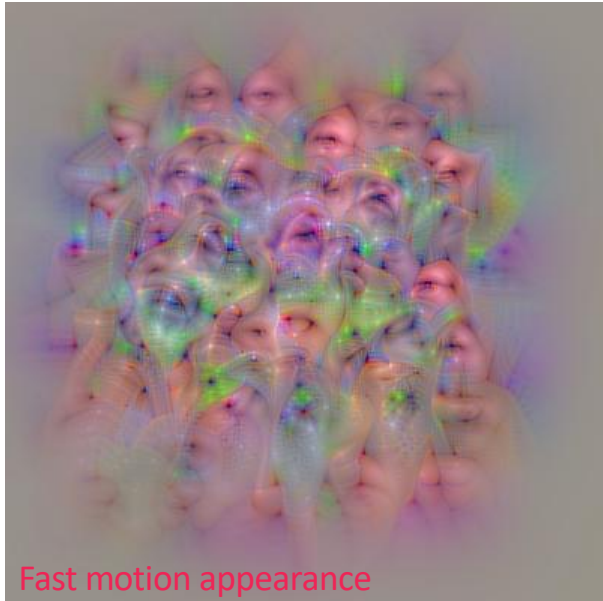


"Fast" motion

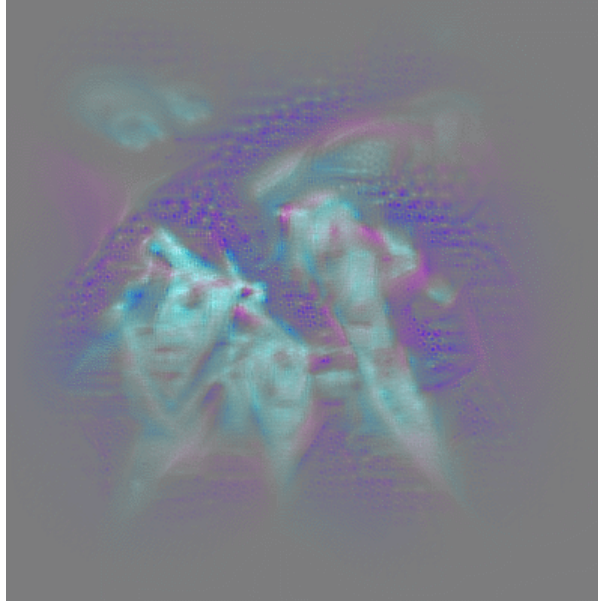


# Can you guess the action?

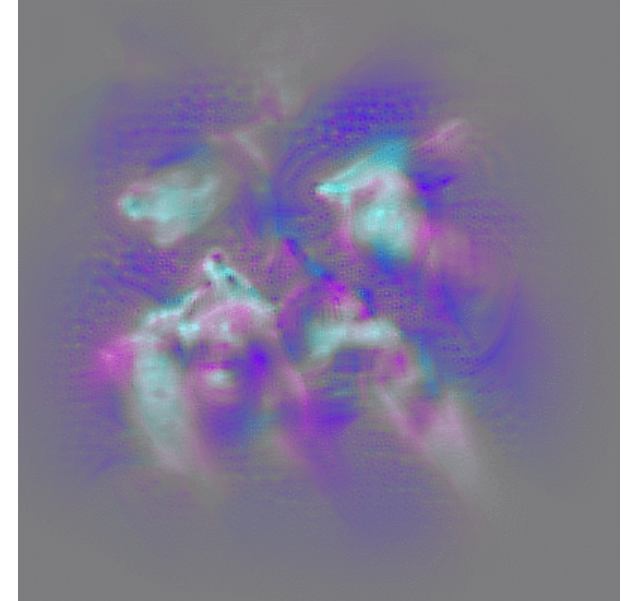
Appearance



“Slow” motion



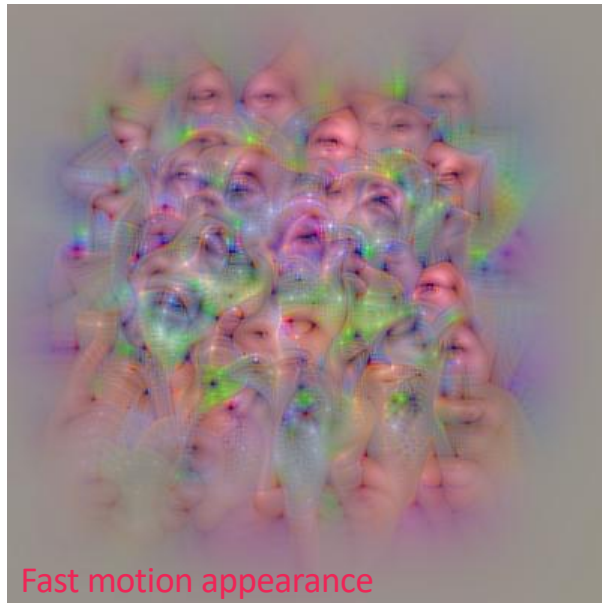
“Fast” motion



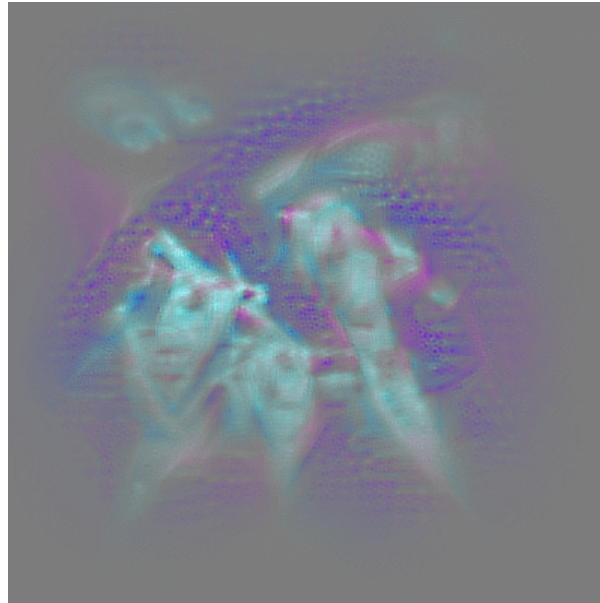


# Can you guess the action? Apply Eye Makeup

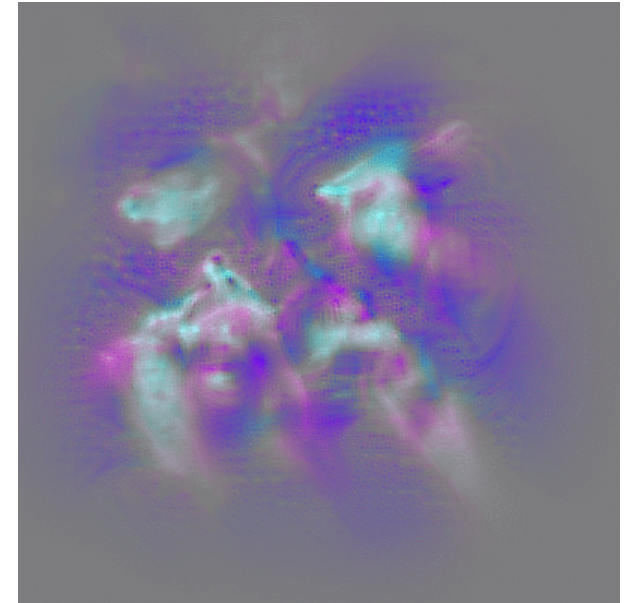
Appearance



“Slow” motion



“Fast” motion



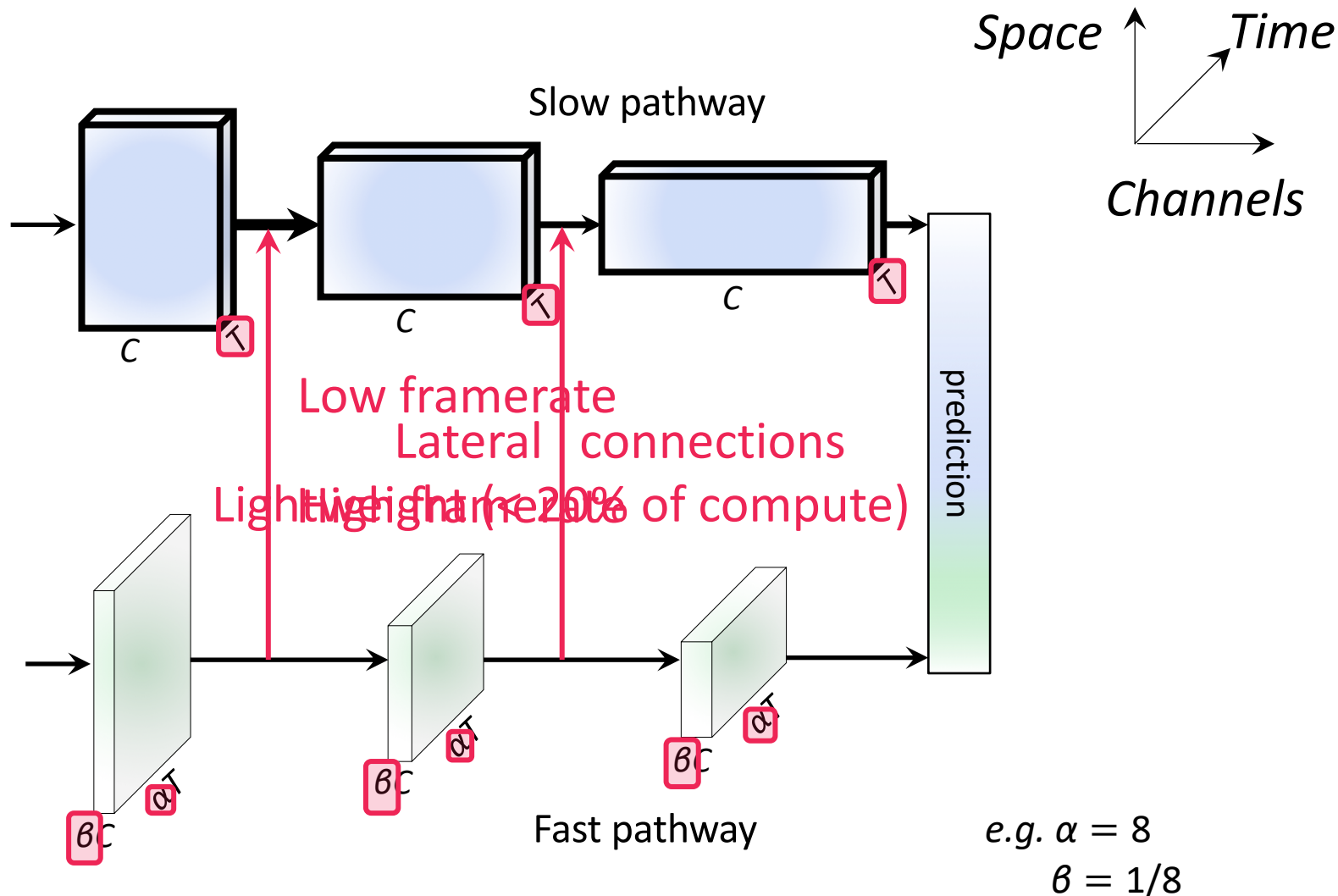


# Treating time and space differently: SlowFast Networks

Slow



Fast



# Treating time and space differently: SlowFast Networks

- Dimensions are  $\{T \times S^2, C\}$
- Strides are  $\{\text{temporal}, \text{spatial}^2\}$
- The backbone is ResNet-50
- Residual blocks are shown by brackets
- Non-degenerate temporal filters are underlined
- Here the speed ratio is  $\alpha = 8$  and the channel ratio is  $\beta = 1/8$
- **Orange** numbers mark fewer channels, for the Fast pathway
- **Green** numbers mark higher temporal resolution of the Fast pathway
- No temporal *pooling* is performed throughout the hierarchy

stage	Slow pathway	Fast pathway	output sizes $T \times S^2$
raw clip	-	-	$64 \times 224^2$
data layer	stride 16, $1^2$	stride <b>2</b> , $1^2$	Slow : $4 \times 224^2$ Fast : <b>32</b> $\times 224^2$
conv <sub>1</sub>	$1 \times 7^2, 64$ stride 1, $2^2$	<u><math>5 \times 7^2, 8</math></u> stride 1, $2^2$	Slow : $4 \times 112^2$ Fast : <b>32</b> $\times 112^2$
pool <sub>1</sub>	$1 \times 3^2$ max stride 1, $2^2$	$1 \times 3^2$ max stride 1, $2^2$	Slow : $4 \times 56^2$ Fast : <b>32</b> $\times 56^2$
res <sub>2</sub>	$\begin{bmatrix} 1 \times 1^2, 64 \\ 1 \times 3^2, 64 \\ 1 \times 1^2, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} \underline{3 \times 1^2, 8} \\ 1 \times 3^2, 8 \\ 1 \times 1^2, \text{32} \end{bmatrix} \times 3$	Slow : $4 \times 56^2$ Fast : <b>32</b> $\times 56^2$
res <sub>3</sub>	$\begin{bmatrix} 1 \times 1^2, 128 \\ 1 \times 3^2, 128 \\ 1 \times 1^2, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} \underline{3 \times 1^2, 16} \\ 1 \times 3^2, 16 \\ 1 \times 1^2, 64 \end{bmatrix} \times 4$	Slow : $4 \times 28^2$ Fast : <b>32</b> $\times 28^2$
res <sub>4</sub>	$\begin{bmatrix} \underline{3 \times 1^2, 256} \\ 1 \times 3^2, 256 \\ 1 \times 1^2, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} \underline{3 \times 1^2, 32} \\ 1 \times 3^2, 32 \\ 1 \times 1^2, 128 \end{bmatrix} \times 6$	Slow : $4 \times 14^2$ Fast : <b>32</b> $\times 14^2$
res <sub>5</sub>	$\begin{bmatrix} \underline{3 \times 1^2, 512} \\ 1 \times 3^2, 512 \\ 1 \times 1^2, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} \underline{3 \times 1^2, 64} \\ 1 \times 3^2, 64 \\ 1 \times 1^2, 256 \end{bmatrix} \times 3$	Slow : $4 \times 7^2$ Fast : <b>32</b> $\times 7^2$
global average pool, concat, fc			# classes

# So far: Classify short clips



Videos: Recognize **actions**



Swimming  
**Running**  
Jumping  
Eating  
Standing

# Temporal Action Localization

Given a long untrimmed video sequence, identify frames corresponding to different actions

Running



Jumping



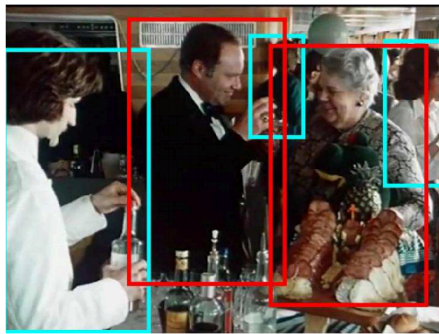
Can use architecture similar to Faster R-CNN:  
first generate **temporal proposals** then **classify**

Chao et al, " Rethinking the Faster R-CNN Architecture for Temporal Action Localization", CVPR 2018

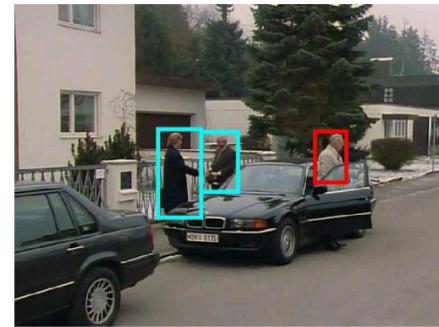


# Spatio-Temporal Detection

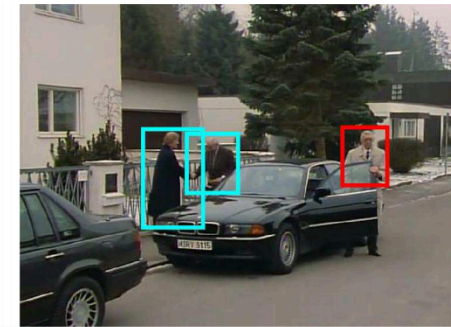
Given a long untrimmed video, detect all the people in space and time and classify the activities they are performing  
Some examples from AVA Dataset:



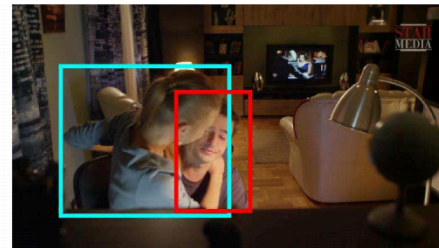
clink glass → drink



open → close



grab (a person) → hug



look at phone → answer phone



Gu et al, "AVA: A Video Dataset of Spatio-temporally Localized Atomic Visual Actions", CVPR 2018

# Recap: Video Models

## **Many video models:**

Single-frame CNN (Try this first!)

Late fusion

Early fusion

3D CNN / C3D

Two-stream networks

CNN + RNN

Convolutional RNN

Spatio-temporal self-attention

SlowFast networks (current SoTA)

Next time:

Generative Models, part 1

Generative Adversarial Networks