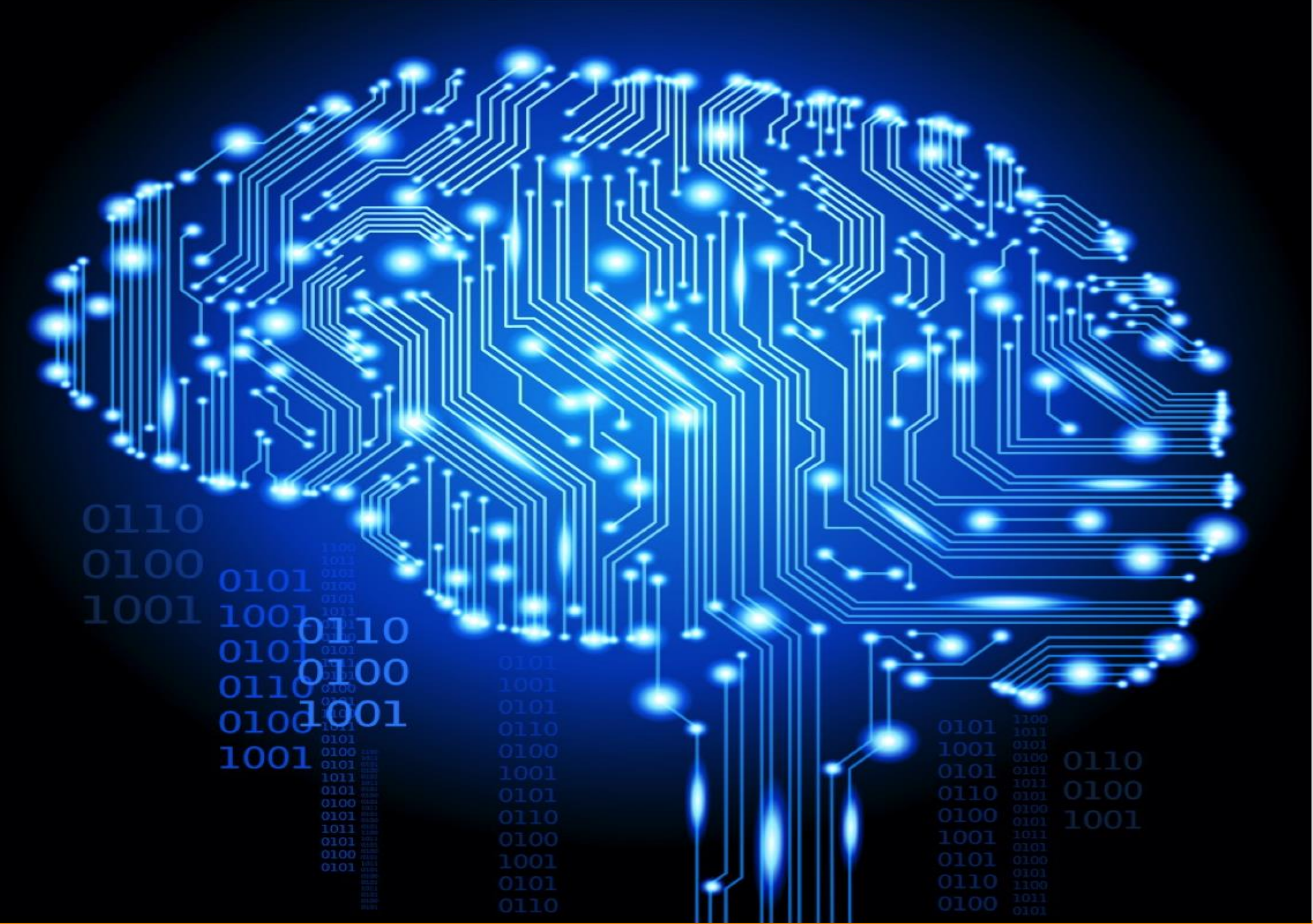


Lecture 5: Modern ConvNets

Efstratios Gavves

Lecture overview

- Popular Convolutional Neural Networks architectures
- Go deeper on what makes them tick
  - what makes them different

VGGnet

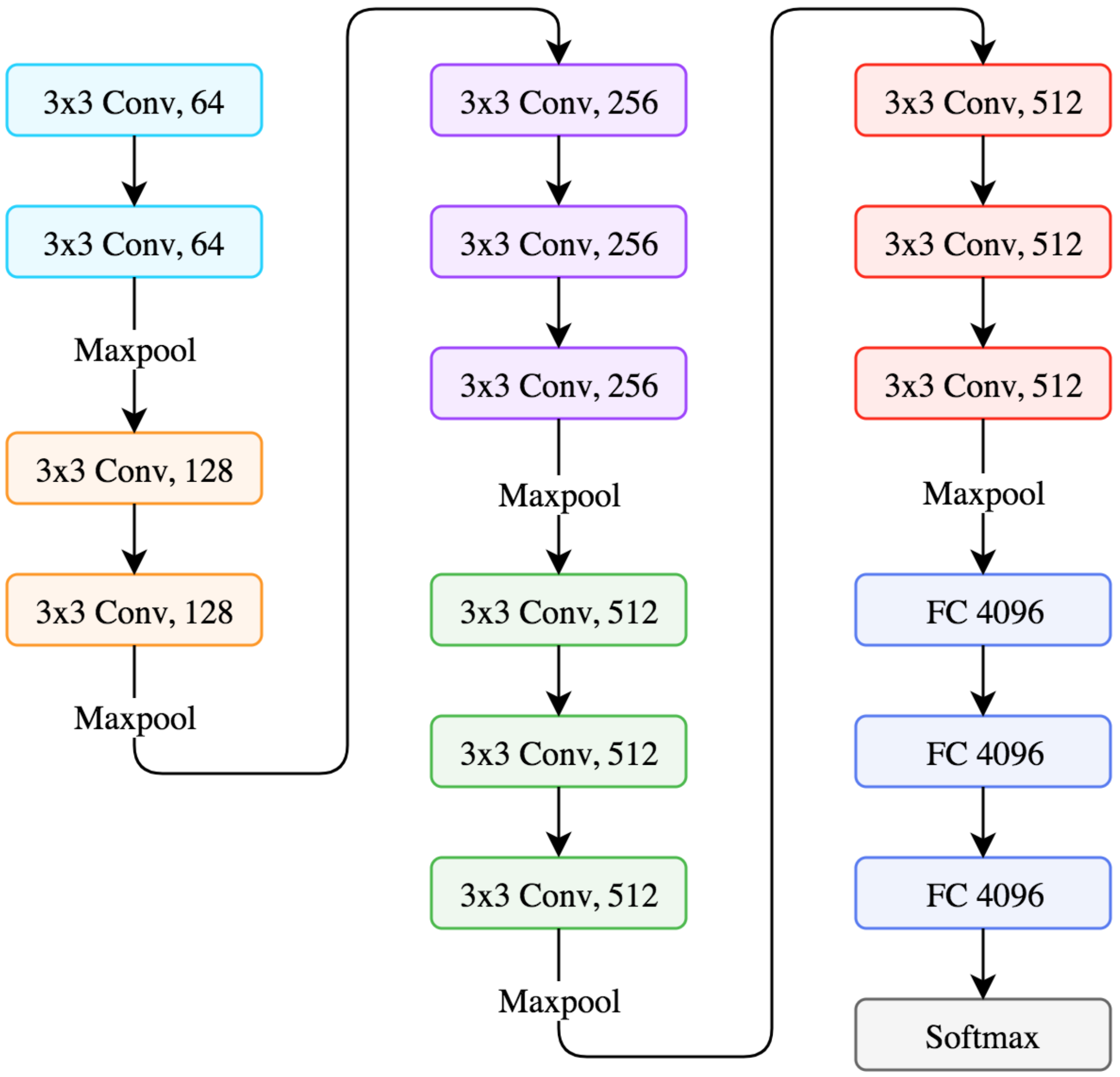
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

VGG16

- 7.3% error rate in ImageNet
- Compared to 18.2% of AlexNet



Picture credit: [Arden Dertat]

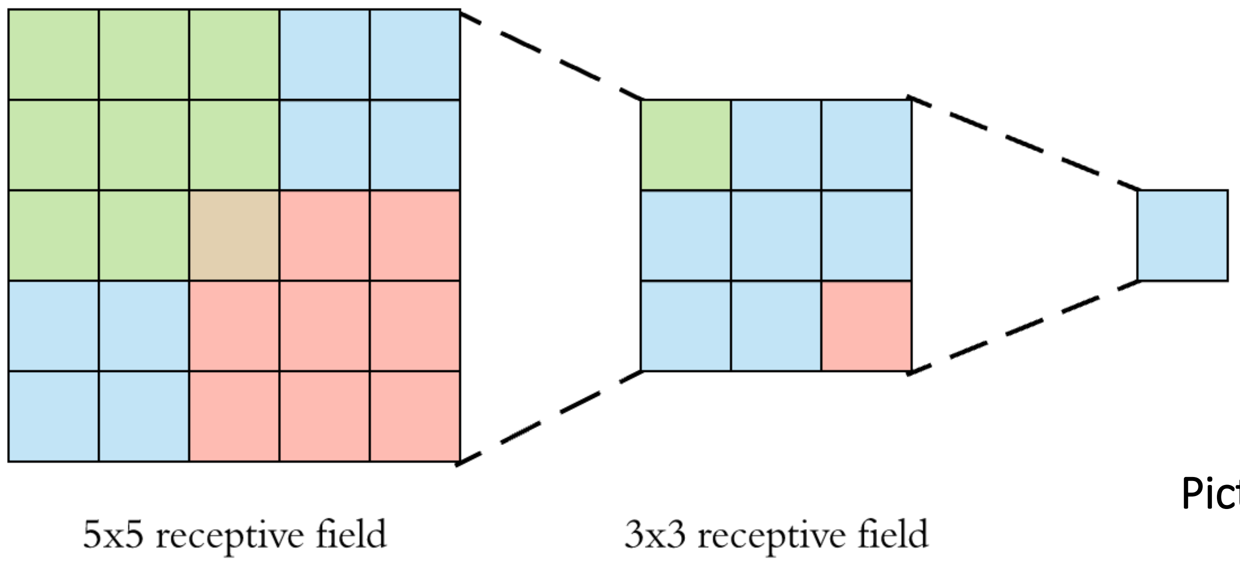
Characteristics

- Input size: 224×224
- Filter sizes: 3×3
- Convolution stride: 1
  - Spatial resolution preserved
- Padding: 1
- Max pooling: 2×2 with a stride of 2
- ReLU activations
- No fancy input normalizations
  - No Local Response Normalizations
- Although deeper, number of weights is not exploding

Why 3×3 filters?

Why 3×3 filters?

- The smallest possible filter to captures the “up”, “down”, “left”, “right”
- Two 3×3 filters have the receptive field of one 5×5
- Three 3×3 filters have the receptive field of …



Picture credit: Arden Dertat

Why 3×3 filters?

- The smallest possible filter to captures the “up”, “down”, “left”, “right”
- Two 3×3 filters have the receptive field of one 5×5
- Three 3×3 filters have the receptive field of one 7×7
- 1 large filter can be replaced by a deeper stack of successive smaller filters
- Benefit?

Why 3×3 filters?

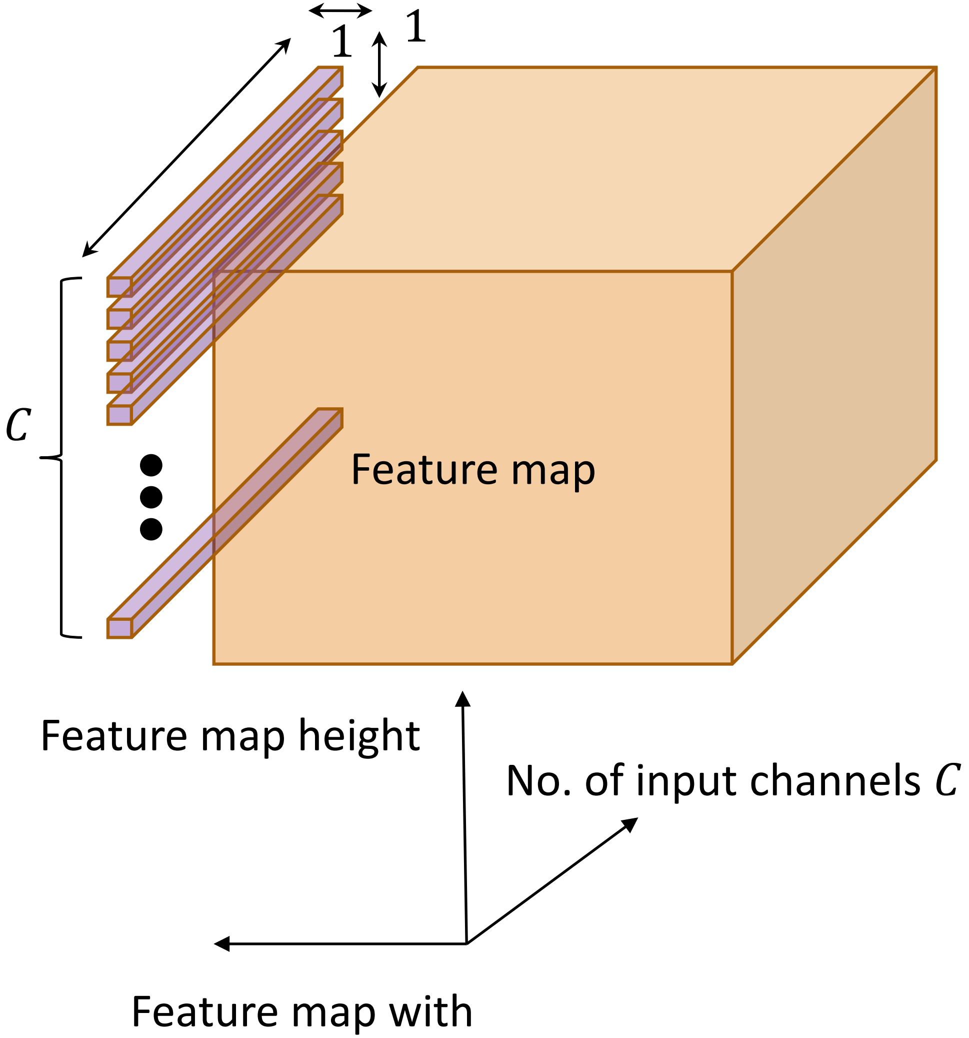
- The smallest possible filter to captures the “up”, “down”, “left”, “right”
- Two 3×3 filters have the receptive field of one 5×5
- Three 3×3 filters have the receptive field of one 7×7
- 1 large filter can be replaced by a deeper stack of successive smaller filters
- Benefit?
- Three more nonlinearities for the same “size” of pattern learning
- Also fewer parameters and regularization

$$(3 \times 3 \times C) \times 3 = 27 \cdot C, 7 \times 7 \times C \times 1 = 49 \cdot C$$

- **Conclusion: 1 large filter can be replaced by a deeper, potentially more powerful, stack of successive smaller filters**

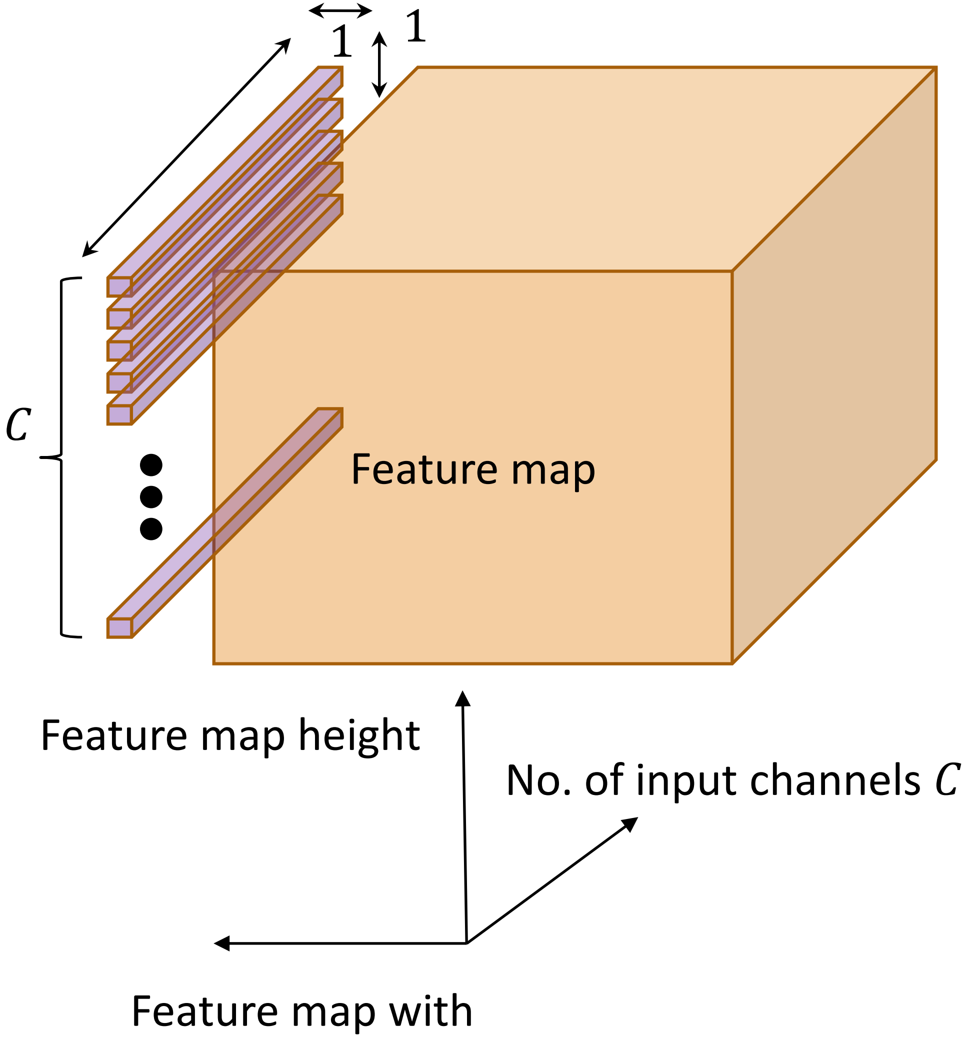
Even smaller filters?

- Also $1x1$ filters are used
- Followed by a nonlinearity
- Why?



Even smaller filters?

- Also $1x1$ filters are used
- Followed by a nonlinearity
- Why?
- Increasing nonlinearities without affecting receptive field sizes
  - Linear transformation of the input channels



Training

- Batch size: 256
- SGD with momentum=0.9
- Weight decay $\lambda = 5 \cdot 10^{-4}$
- Dropout on first two fully connected layers
- Learning rate $\eta_0 = 10^{-2}$, then decreased by factor of 10 when validation accuracy stopped improving
  - Three times this learning rate decrease
- Faster training
  - Smaller filters →
  - Depth also serves as regularization

Inception

type	patch size/ stride	output size	depth	$\#1\times1$	$\#3\times3$ reduce	$\#3\times3$	$\#5\times5$ reduce	$\#5\times5$	pool proj	params	ops
convolution	$7\times7/2$	$112\times112\times64$	1							2.7K	34M
max pool	$3\times3/2$	$56\times56\times64$	0								
convolution	$3\times3/1$	$56\times56\times192$	2		64	192				112K	360M
max pool	$3\times3/2$	$28\times28\times192$	0								
inception (3a)		$28\times28\times256$	2	64	96	128	16	32	32	159K	128M
inception (3b)		$28\times28\times480$	2	128	128	192	32	96	64	380K	304M
max pool	$3\times3/2$	$14\times14\times480$	0								
inception (4a)		$14\times14\times512$	2	192	96	208	16	48	64	364K	73M
inception (4b)		$14\times14\times512$	2	160	112	224	24	64	64	437K	88M
inception (4c)		$14\times14\times512$	2	128	128	256	24	64	64	463K	100M
inception (4d)		$14\times14\times528$	2	112	144	288	32	64	64	580K	119M
inception (4e)		$14\times14\times832$	2	256	160	320	32	128	128	840K	170M
max pool	$3\times3/2$	$7\times7\times832$	0								
inception (5a)		$7\times7\times832$	2	256	160	320	32	128	128	1072K	54M
inception (5b)		$7\times7\times1024$	2	384	192	384	48	128	128	1388K	71M
avg pool	$7\times7/1$	$1\times1\times1024$	0								
dropout (40%)		$1\times1\times1024$	0								
linear		$1\times1\times1000$	1							1000K	1M
softmax		$1\times1\times1000$	0								

Table 1: GoogLeNet incarnation of the Inception architecture

Basic idea

- Problem?

Picture credit: [Bharath Raj]

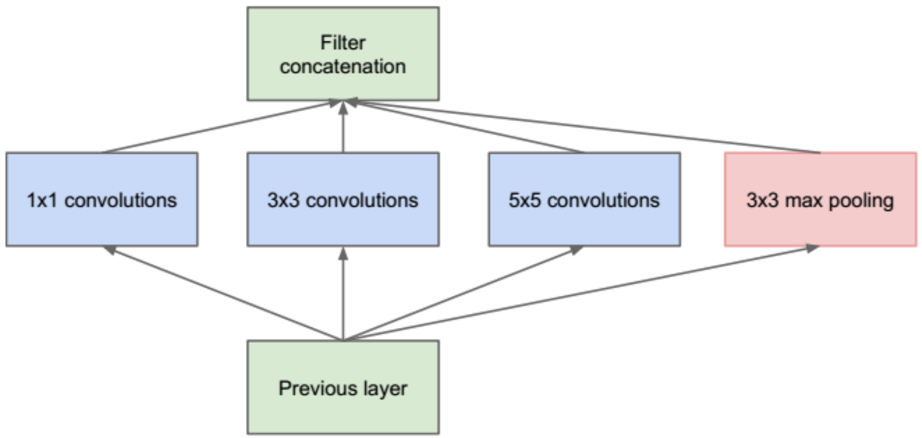
Basic idea

- Salient parts have great variation in sizes
- Hence, the receptive fields should vary in size accordingly
- Naively stacking convolutional operations is expensive
- Very deep nets are prone to overfitting

Picture credit: Bharath Raj

Inception module

- Multiple kernel filters of different sizes $(1 \times 1, 3 \times 3, 5 \times 5)$
  - Naïve version
- Problem?

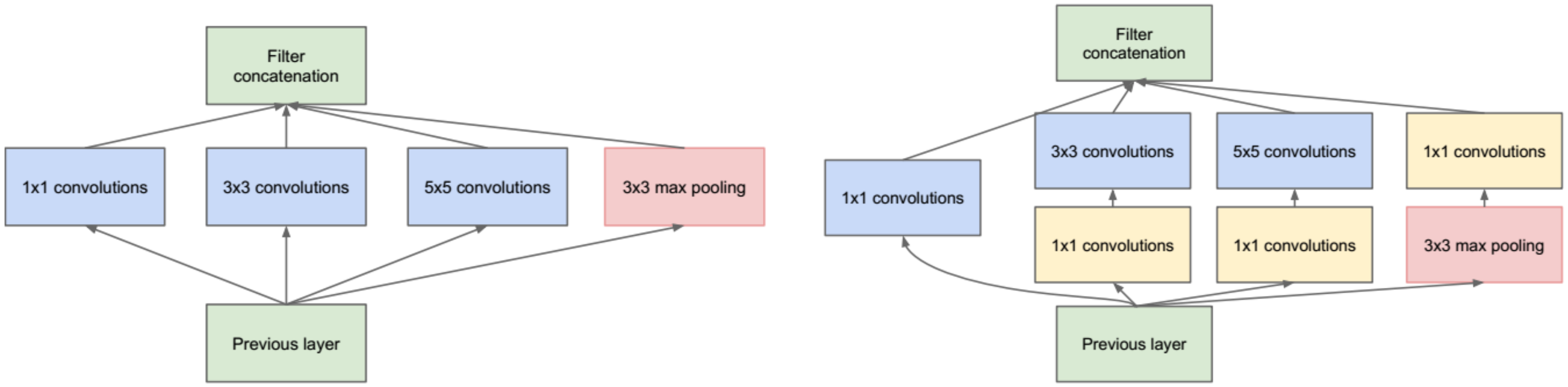


(a) Inception module, naïve version

Picture credit: [Bharath Raj]

Inception module

- Multiple kernel filters of different sizes $(1 \times 1, 3 \times 3, 5 \times 5)$
  - Naïve version
- Problem?
  - Very expensive!
- Add intermediate 1×1 convolutions

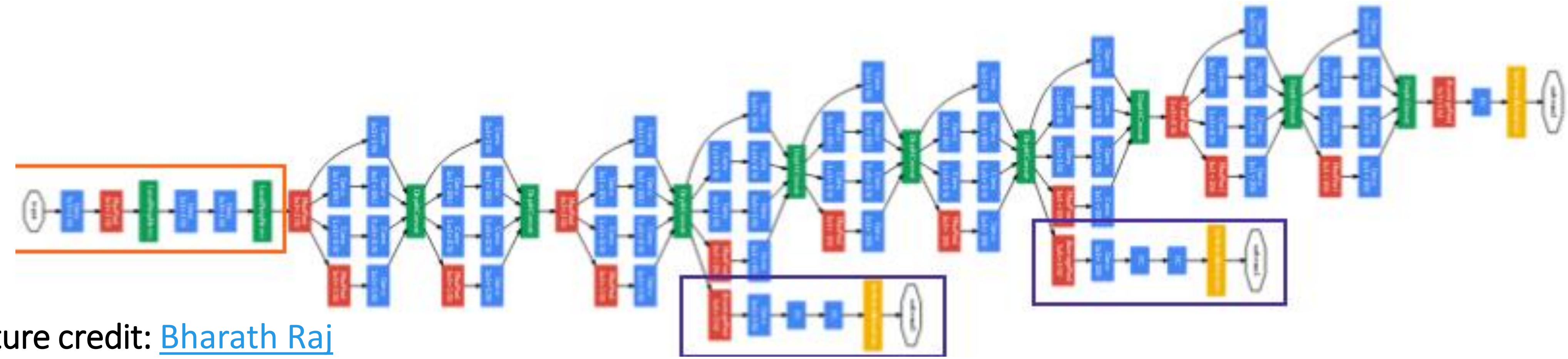


(a) Inception module, naïve version (b) Inception module with dimension reductions

Picture credit: [Bharath Raj]

Architecture

- 9 Inception Modules
- 22 layers deep (27 with the pooling layers)
- Global average pooling at the end of last Inception Module
- 6.67% Imagenet error, compared to 18.2% of Alexnet



Picture credit: [Bharath Raj]

Houston, we have a problem

Problem: Vanishing gradients

- The network was too deep (at the time)
- Roughly speaking, backprop is lots of matrix multiplications

$$\frac{\partial \mathcal{L}}{\partial w^l} = \frac{\partial \mathcal{L}}{\partial a^L} \cdot \frac{\partial a^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial a^{L-2}} \cdot \ldots \cdot \frac{\partial a^l}{\partial w^l}$$

- Many of intermediate terms < 1 → the final $\frac{\partial \mathcal{L}}{\partial w^l}$ gets extremely small
- Extremely small gradient → ?

Picture credit: [Anish Singh Walia]

Problem: Vanishing gradients (more details later)

- The network was too deep (at the time)
- Roughly speaking, backprop is lots of matrix multiplications

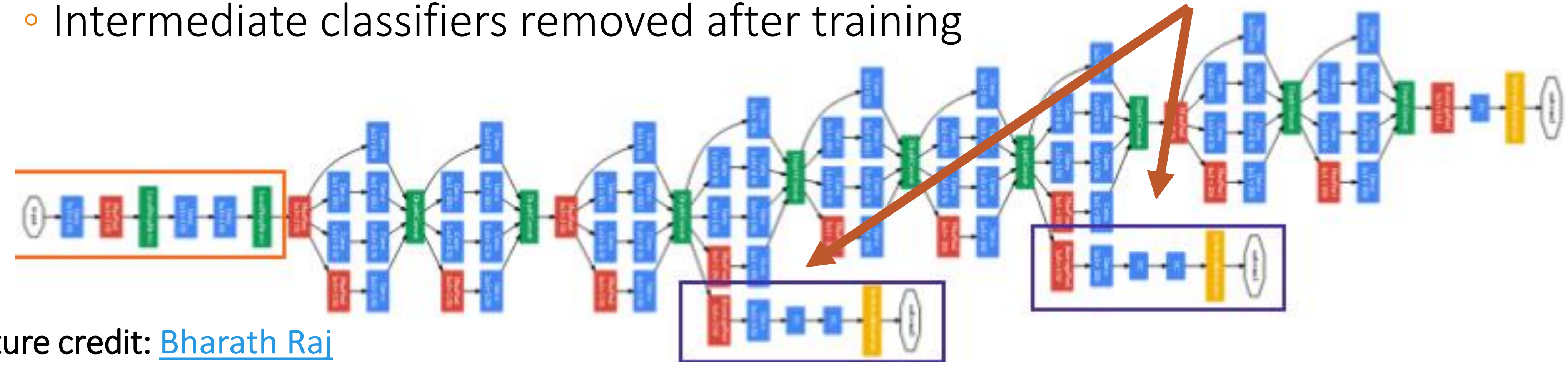
$$\frac{\partial \mathcal{L}}{\partial w^l} = \frac{\partial \mathcal{L}}{\partial a^L} \cdot \frac{\partial a^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial a^{L-2}} \cdot \ \dots \cdot \frac{\partial a^l}{\partial w^l}$$

- Many of intermediate terms < 1 → the final $\frac{\partial \mathcal{L}}{\partial w^l}$ gets extremely small
- Extremely small gradient → Extremely slow learning

Picture credit: Anish Singh Walia

Architecture

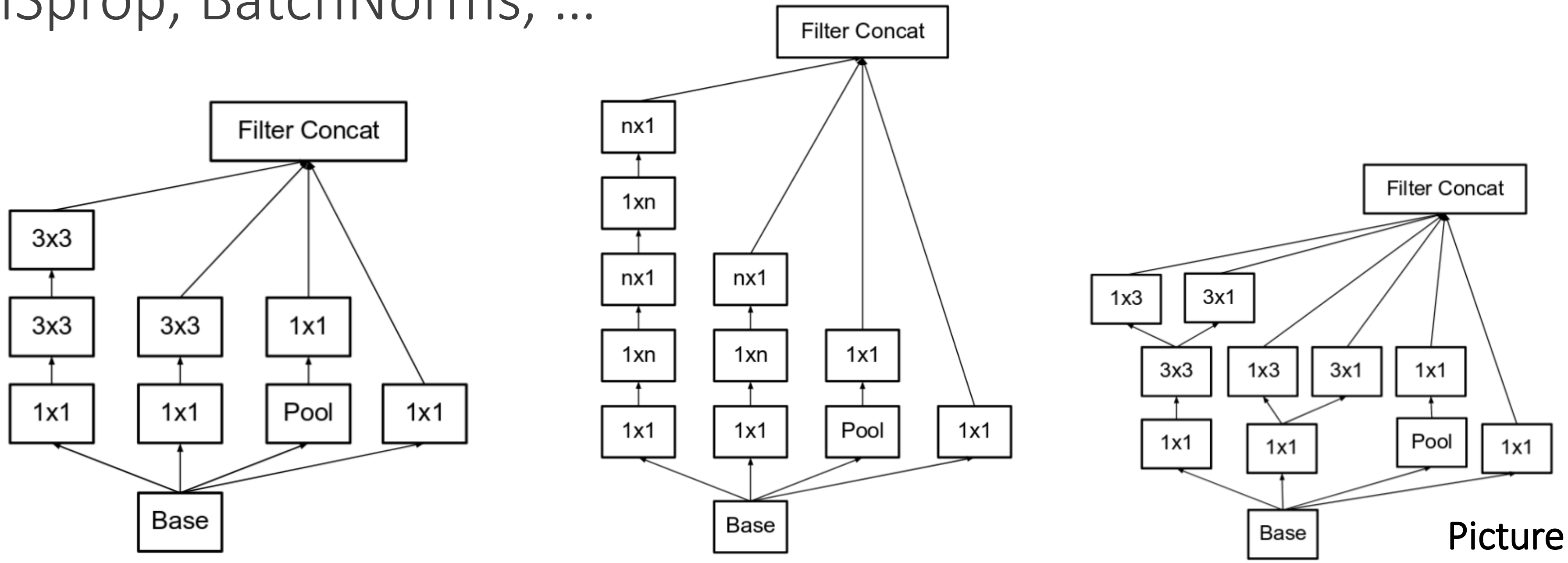
- 9 Inception Modules
- 22 layers deep (27 with the pooling layers)
- Global average pooling at the end of last Inception Module
- Because of the increased depth → **Vanishing gradients**
- Inception solution to vanishing gradients: **intermediate classifiers**
    - Intermediate classifiers removed after training



Picture credit: [Bharath Raj]

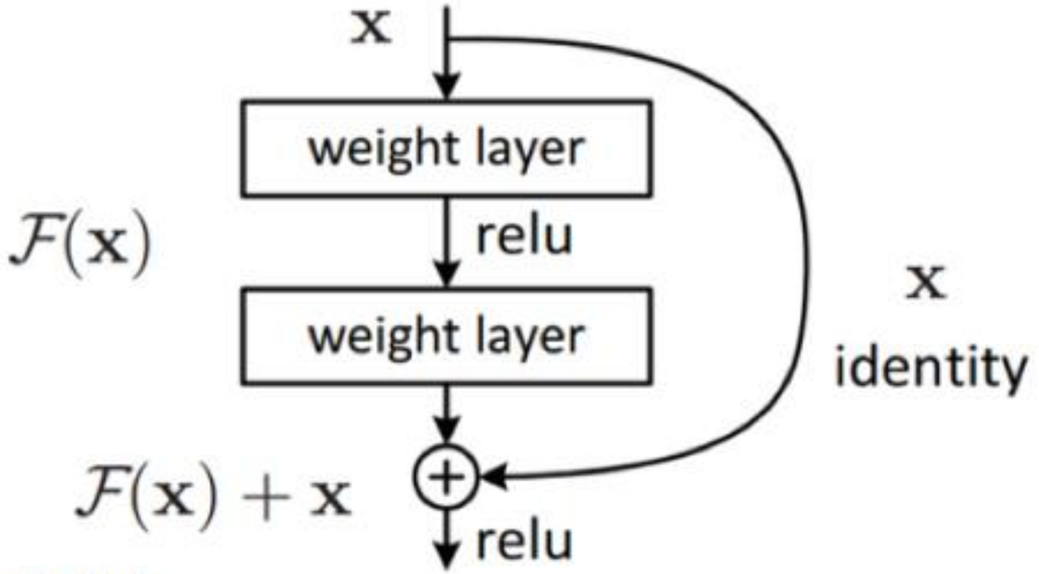
Inceptions v2, v3, v4, ….

- Factorize 5×5 in two 3×3 filters
- Factorize $n \times n$ in two $n \times 1$ and $1 \times n$ filters (quite a lot cheaper)
- Make nets wider
- RMSprop, BatchNorms, …



Picture credit: Bharath Raj

ResNets
DenseNets
HighwayNets



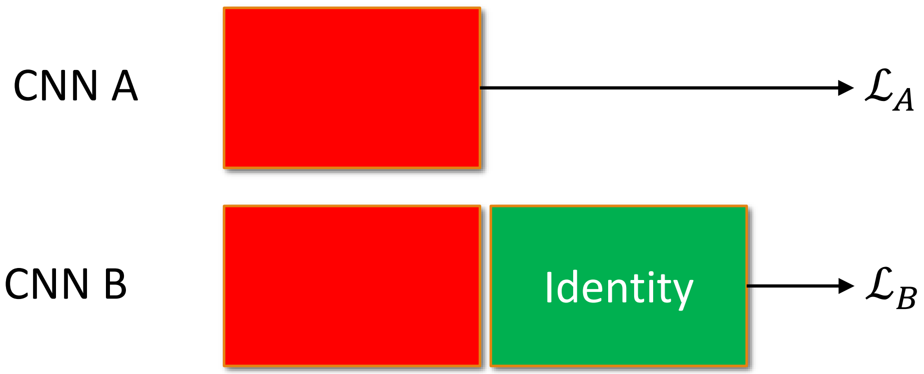
A residual block

Some facts

- The first truly Deep Network, going deeper than 1,000 layers
- More importantly, the first Deep Architecture that proposed a novel concept on how to gracefully go deeper than a few dozen layers
  - Not simply getting more GPUs, more training time, etc
- Smashed Imagenet, with a 3.57% error (in ensembles)
- Won all object classification, detection, segmentation, etc. challenges

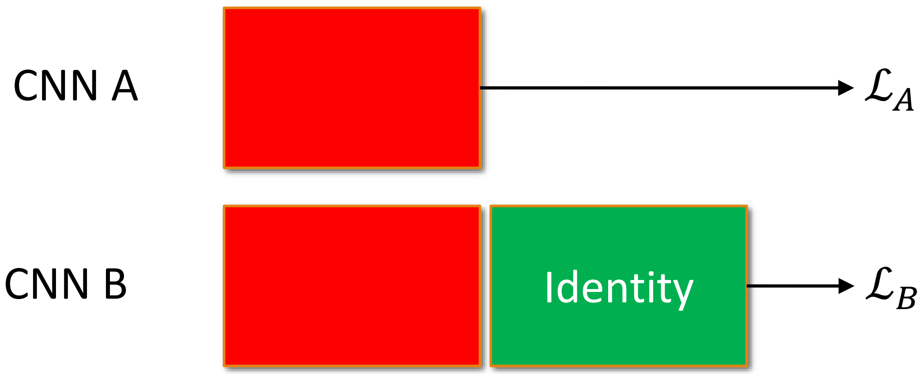
Hypothesis

- **Hypothesis:** Is it possible to have a very deep network at least as accurate as averagely deep networks?
- **Thought experiment:** Let's assume two Convnets A, B. They are almost identical, in that B is the same as A, with extra "identity" layers. Since identity layers pass the information unchanged, the errors of the two networks should …



Hypothesis

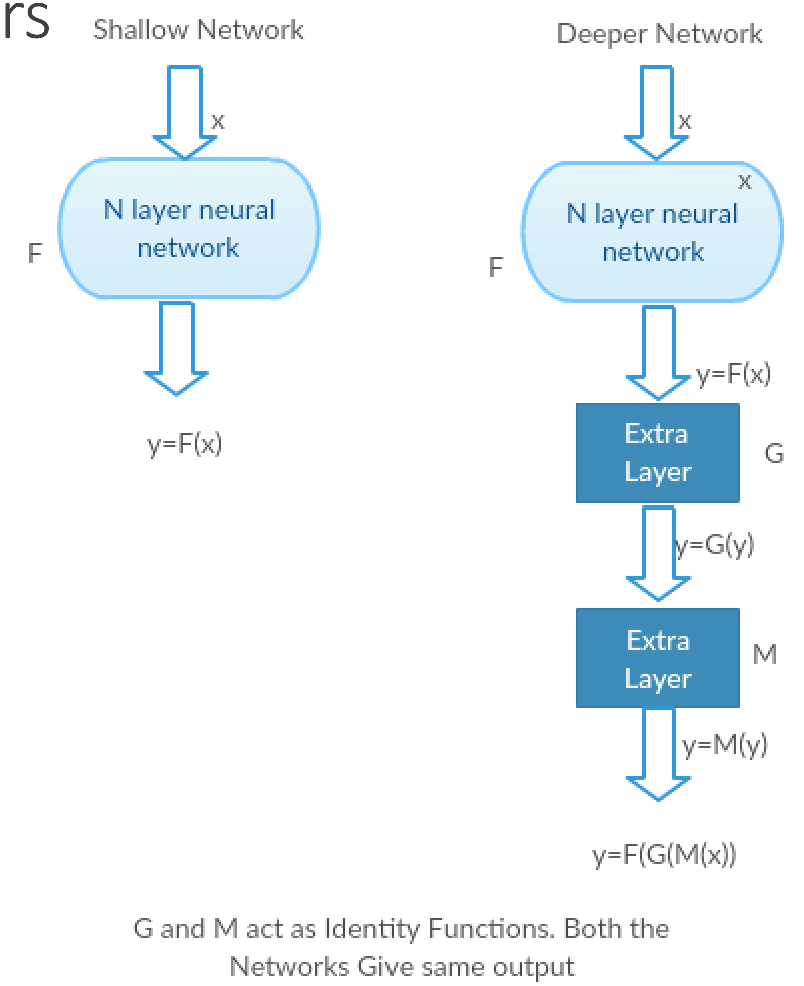
- **Hypothesis:** Is it possible to have a very deep network at least as accurate as averagely deep networks?
- **Thought experiment:** Let's assume two Convnets A, B. They are almost identical, in that B is the same as A, with extra "identity" layers. Since identity layers pass the information unchanged, the errors of the two networks should be similar. Thus, there is a Convnet B, which is at least as good as Convnet A w.r.t. training error



Hypothesis

- We add a few extra layers on top of the N layers
- These few extra layers are identify mappings

$$y = x$$



Picture credit: [Prakash Jay]

Quiz: What looks weird?

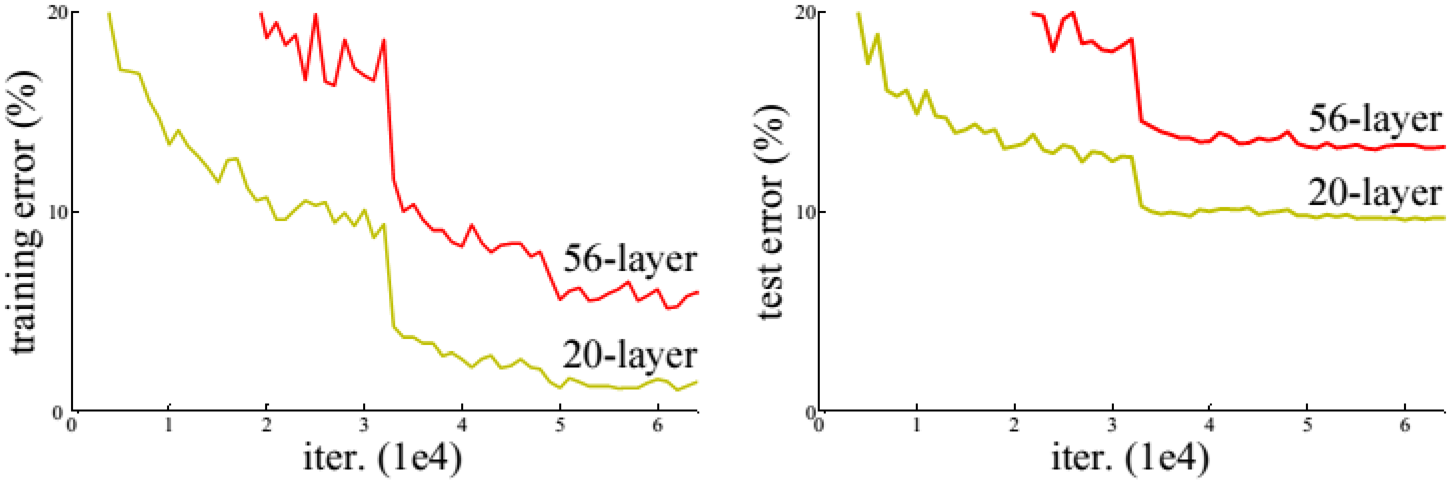
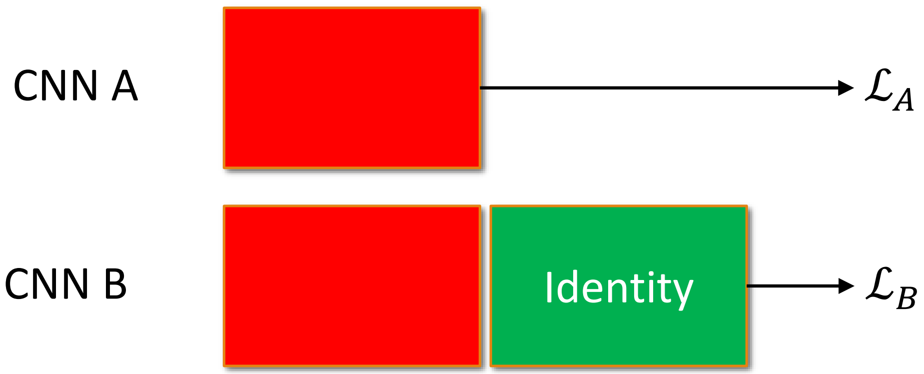


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Testing the hypothesis

- Adding identity layers increases **training error**!!
  - Training error, not testing error
- **Performance degradation** not caused by overfitting
  - Just the optimization task is harder
- Assuming optimizers are doing their job fine, it appears that not all networks are the same as easy to optimize



$\mathcal{L}_A^{train} < \mathcal{L}_B^{train}$!!!

Observation

- Very deep networks stop learning after a bit
  - An accuracy is reached, then the network saturates and starts unlearning
- Signal gets lost through so many layers
- Models start failing

The "residual idea", intuitively

- Let's say we have the neural network nonlinearity $\mathrm{a} = F(x)$
- Perhaps easier to learn a function $\mathrm{a} = F(x)$ to model differences $\mathrm{a} \sim \delta \mathrm{y}$ than to model absolutes $\mathrm{a} \sim \mathrm{y}$
  - Otherwise, you may need to model both the magnitude as well as the direction of activations
  - Think of it like in input normalization → you normalize around 0
  - Think of it like in regression → you model differences around the mean value
- "Residual idea": Let neural networks explicitly model difference mappings

$$F(x) = H(x) - x \Rightarrow H(x) = F(x) + x$$

- $F(x)$ are the stacked nonlinearities
- x is the input to the nonlinear layer

ResNet block

- $H(x) = F(x) + x$
- If dimensions don't match
  - Either zero padding
  - Or a projection layer to match dimensions

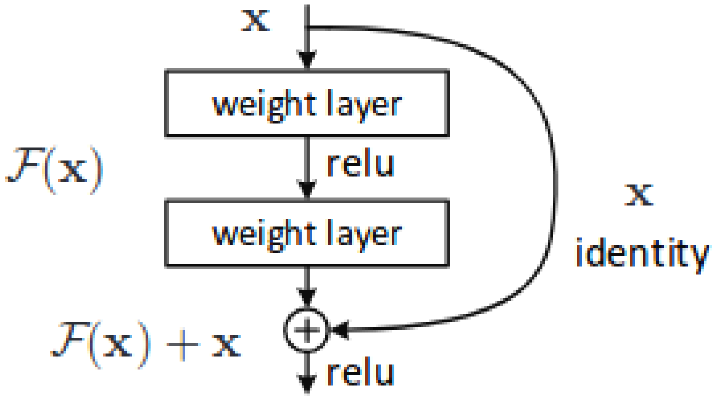
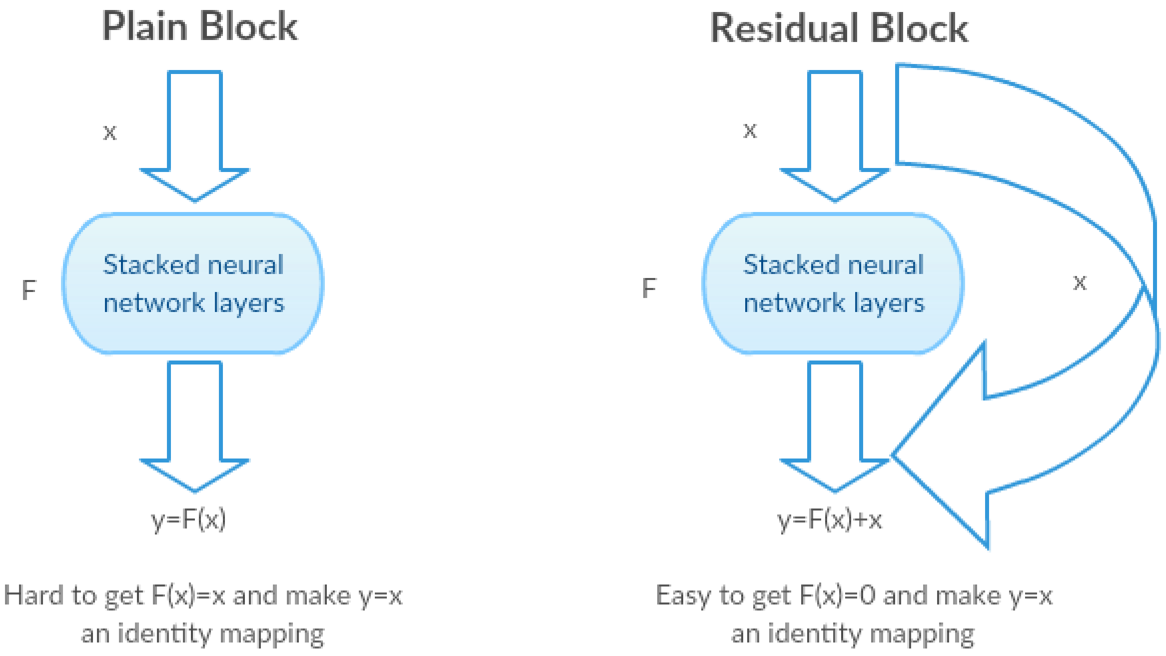


Figure 2. Residual learning: a building block.



No degradation anymore

- Without residual connections deeper networks attain worse scores

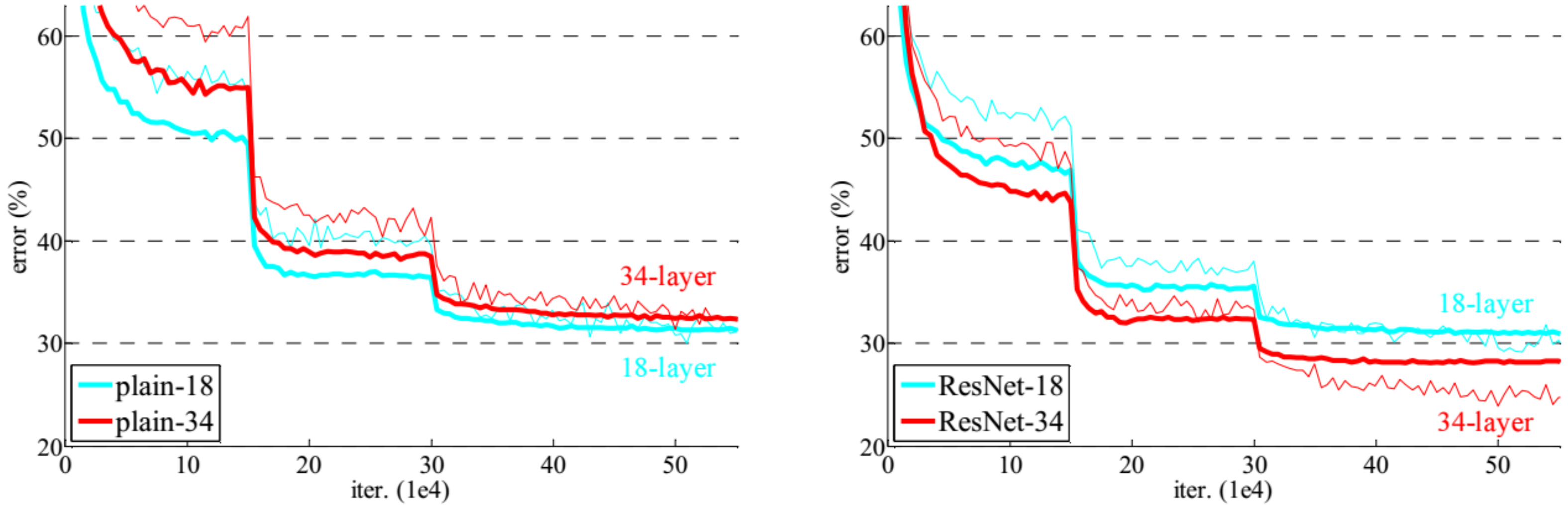


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

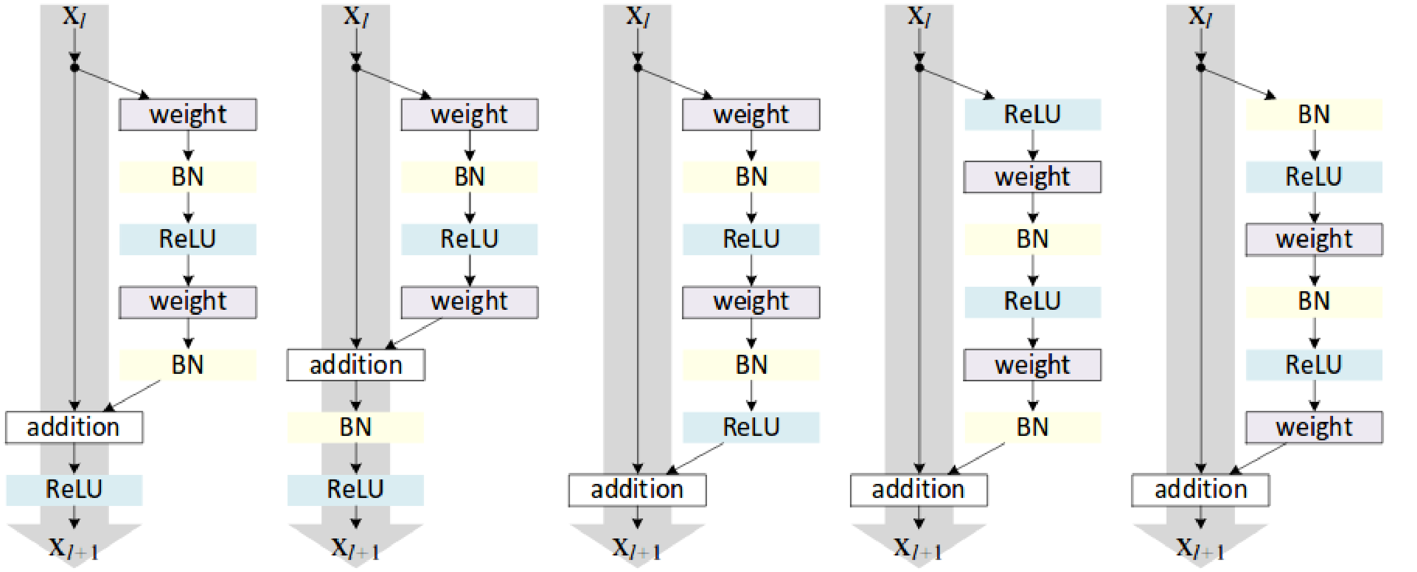
ResNet breaks records

- Ridiculously low error in ImageNet
- Up to 1000 layers ResNets trained
  - Previous deepest network ~30-40 layers on simple datasets

method	top-5 err. (**test**)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

Table 5. Error rates (%) of **ensembles**. The top-5 error is on the test set of ImageNet and reported by the test server.

ResNet architectures & ResNeXt



(a) original (b) BN after addition (c) ReLU before addition (d) ReLU-only pre-activation (e) **full pre-activation**

case	Fig.	ResNet-110	ResNet-164
original Residual Unit [1]	Fig. 4(a)	6.61	5.93
BN after addition	Fig. 4(b)	8.17	6.50
ReLU before addition	Fig. 4(c)	7.84	6.14
ReLU-only pre-activation	Fig. 4(d)	6.71	5.91
full pre-activation	Fig. 4(e)	**6.37**	**5.46**

ResNeXt

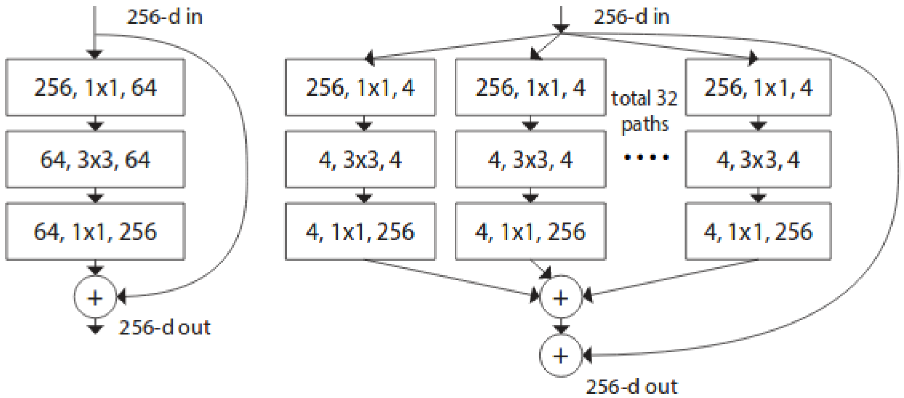


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

	setting	top-1 err (%)	top-5 err (%)
1× complexity references:			
ResNet-101	1 × 64d	22.0	6.0
ResNeXt-101	32 × 4d	21.2	5.6
2× complexity models follow:			
ResNet-**200** [15]	1 × 64d	21.7	5.8
ResNet-101, wider	1 × **100**d	21.3	5.7
ResNeXt-101	**2** × 64d	20.7	5.5
ResNeXt-101	**64** × 4d	**20.4**	**5.3**

Table 4. Comparisons on ImageNet-1K when the number of FLOPs is increased to 2× of ResNet-101's. The error rate is evaluated on the single crop of 224×224 pixels. The highlighted factors are the factors that increase complexity.

[Aggregated Residual Transformations for Deep Neural Networks, Xie et al., 2016]

Some observations

- BatchNorms absolutely necessary because of vanishing gradients
- Networks with skip connections (like ResNets) converge faster than the same network without skip connections
- Identity shortcuts cheaper and almost equal to project shortcuts
- Hopefully, more on Neural Network dynamics later

HighwayNet

- Similar to ResNets, only introducing a gate with learnable parameters on the importance of each skip connection

$$y = H(x, W_H) \cdot T(x, W_T) + x \cdot (1 - T(x, W_T))$$

- Similar to …

HighwayNet

- Similar to ResNets, only introducing a gate with learnable parameters on the importance of each skip connection

$$y = H(x, W_H) \cdot T(x, W_T) + x \cdot (1 - T(x, W_T))$$

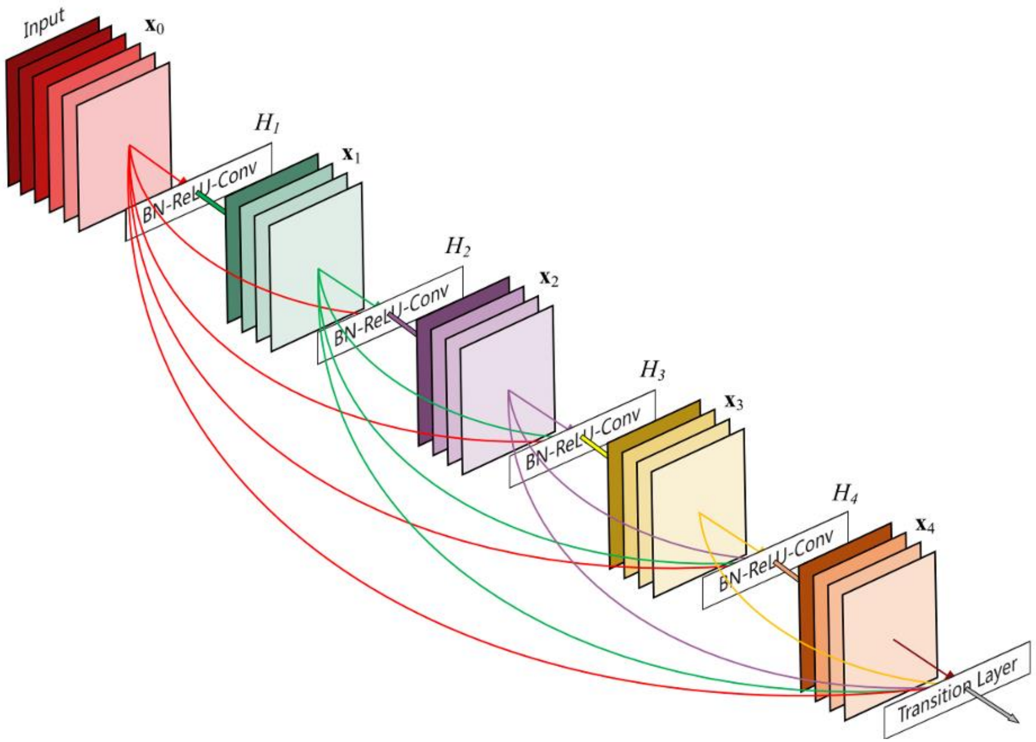
- Similar to ... LSTMs as we will say later

DenseNet

- Add skip connections to multiple forward layers

$$y = h(x_l, x_{l-1}, \dots, x_{l-n})$$

- Why?

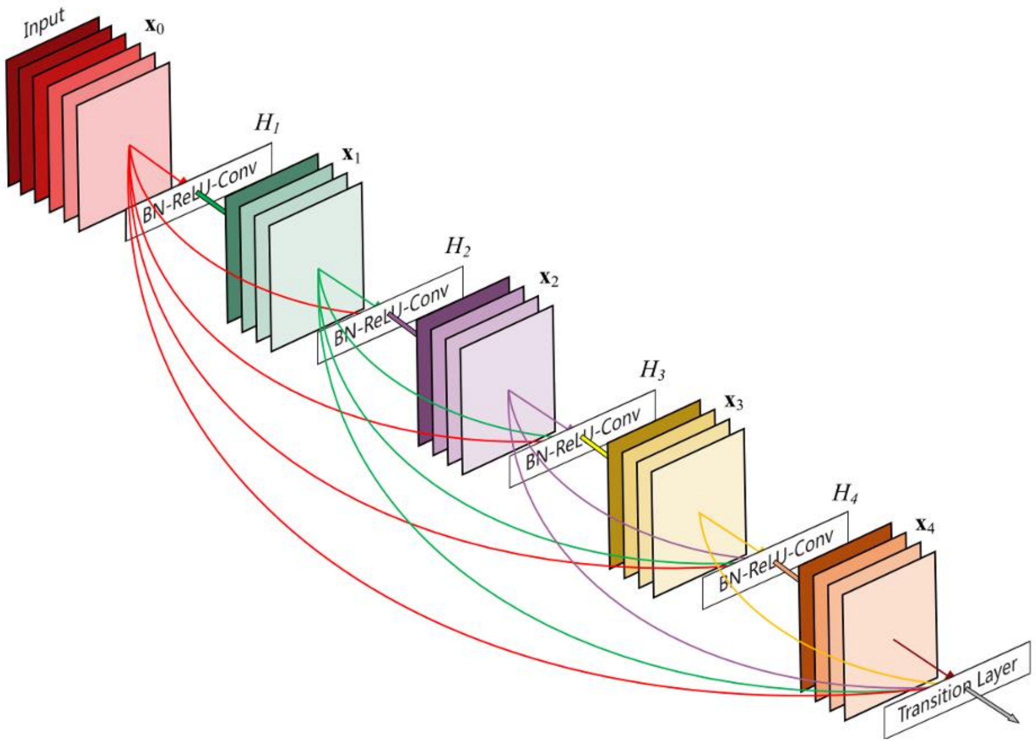


DenseNet

- Add skip connections to multiple forward layers

$$y = h(x_l, x_{l-1}, \ldots, x_{l-n})$$

- Assume layer 1 captures edges, while layer 5 captures faces (and other stuff)
- Why not have a layer that combines both faces and edges (e.g. to model a scarred face)
- Standard ConvNets do not allow for this
  - Layer 6 combines only layer 5 patterns, not lower



Neural Architecture Search

- It is also possible to learn the neural architecture

- Problem?

Neural Architecture Search

- It is also possible to learn the neural architecture

- Problem?
- Architectures/graphs are discrete structures → Backprop?

- Still, some very interesting workarounds have been proposed in practice

- Will it work for you? If you are Facebook or Google, yes!

Evolutionary Search for NAS

- DARTS: Differentiable Architecture Search, Liu et al., 2018
- Efficient Neural Architecture Search via Parameter Sharing, Pham et al., 2018
- Evolving Space-Time Neural Architectures for Videos, Piergiovanni et al. 2018
- Regularized Evolution for Image Classifier Architecture Search, Real et al., 2019

Algorithm 1 Evolutionary search algorithm

function SEARCH
  Randomly initialize the population, P
  Evaluate each individual in P
  for $i <$ number of evolutionary rounds **do**
    $S =$ random sample of 25 individuals
    $parent =$ the most fit individual in S
    $child = parent$
    for $\max(\lceil d - \frac{i}{r} \rceil, 1)$ **do**
      $child = mutate(child)$
    end for
    evaluate $child$ and add to population
    remove least fit individual from population
  end for
end function

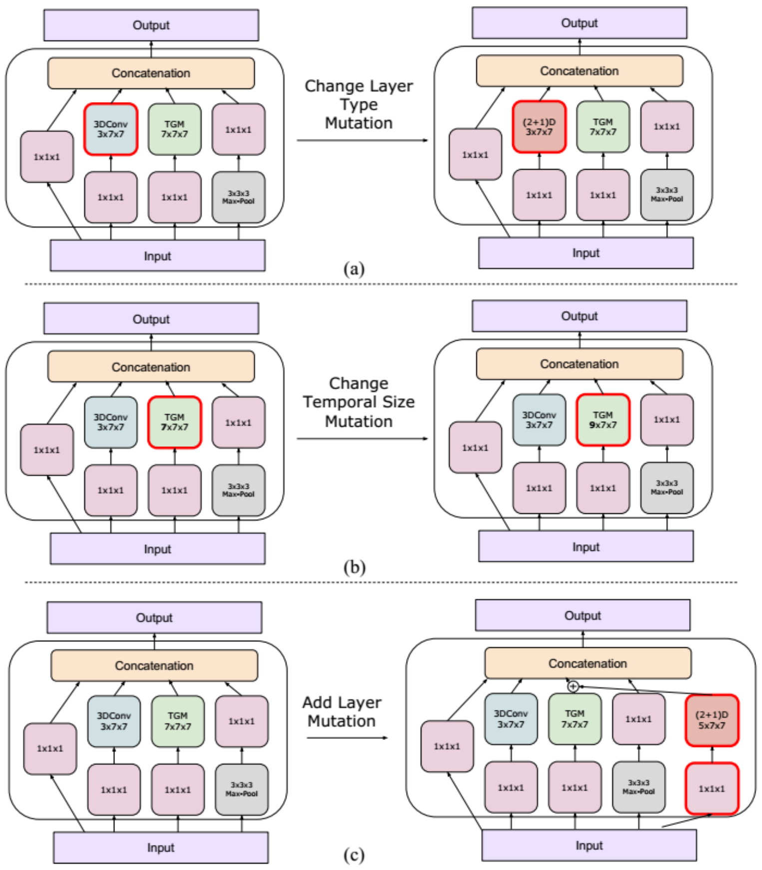
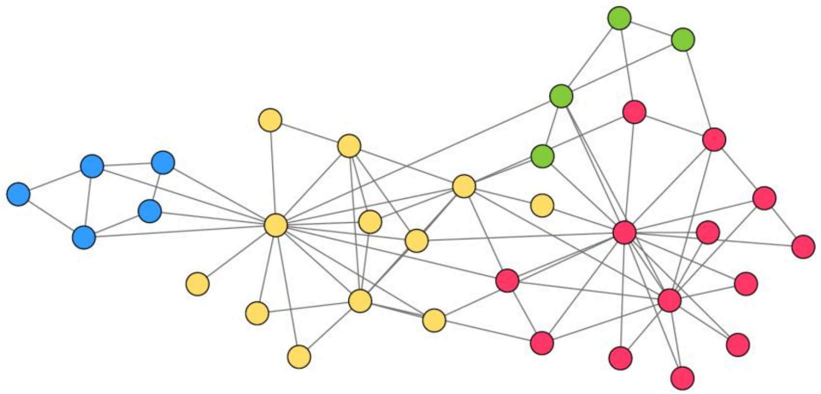
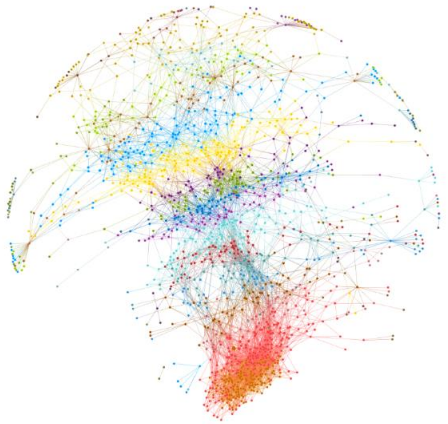


Figure 5. Example mutations applied to a module, including (a) layer type change, (b) filter length change, and (c) layer addition.

Combinatorial Bayesian Optimization for NAS



- Find an optimum of f over a search space S
  $$x_{minimum} \;=\; argmin_{x \in S} \; f(x)$$
- f is a black box function
  - No parametric form of f is known
  - Gradient of f is not calculable
  - Only accessible information is an evaluation at a given point

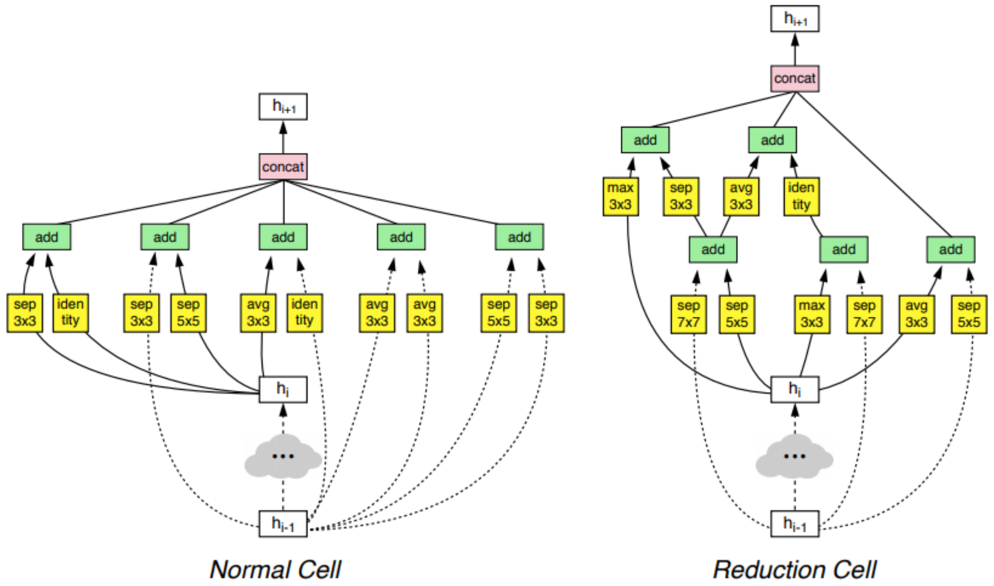


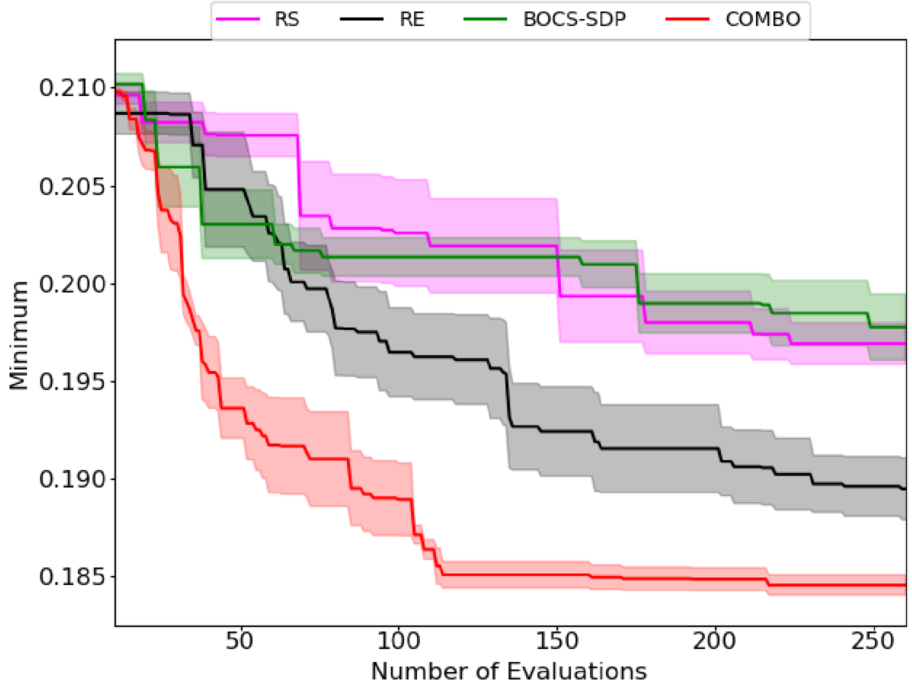
- Minimize f while minimizing # of evaluations
  - In expensive evaluation cost cases, Bayesian Optimization is the most effective method.

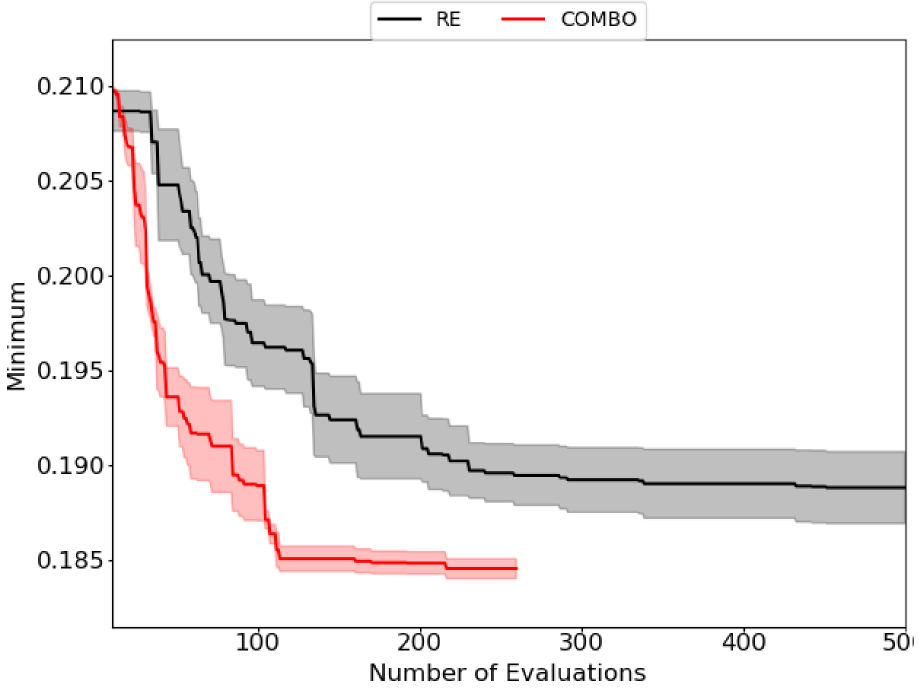
Combinatorial Bayesian Optimization using the Graph Cartesian Product, to appear in NeurIPS 2019
C. Oh, J. Tomczak, E. Gavves, M. Welling, to appear in NeurIPS 2019

On Neural Architecture Search

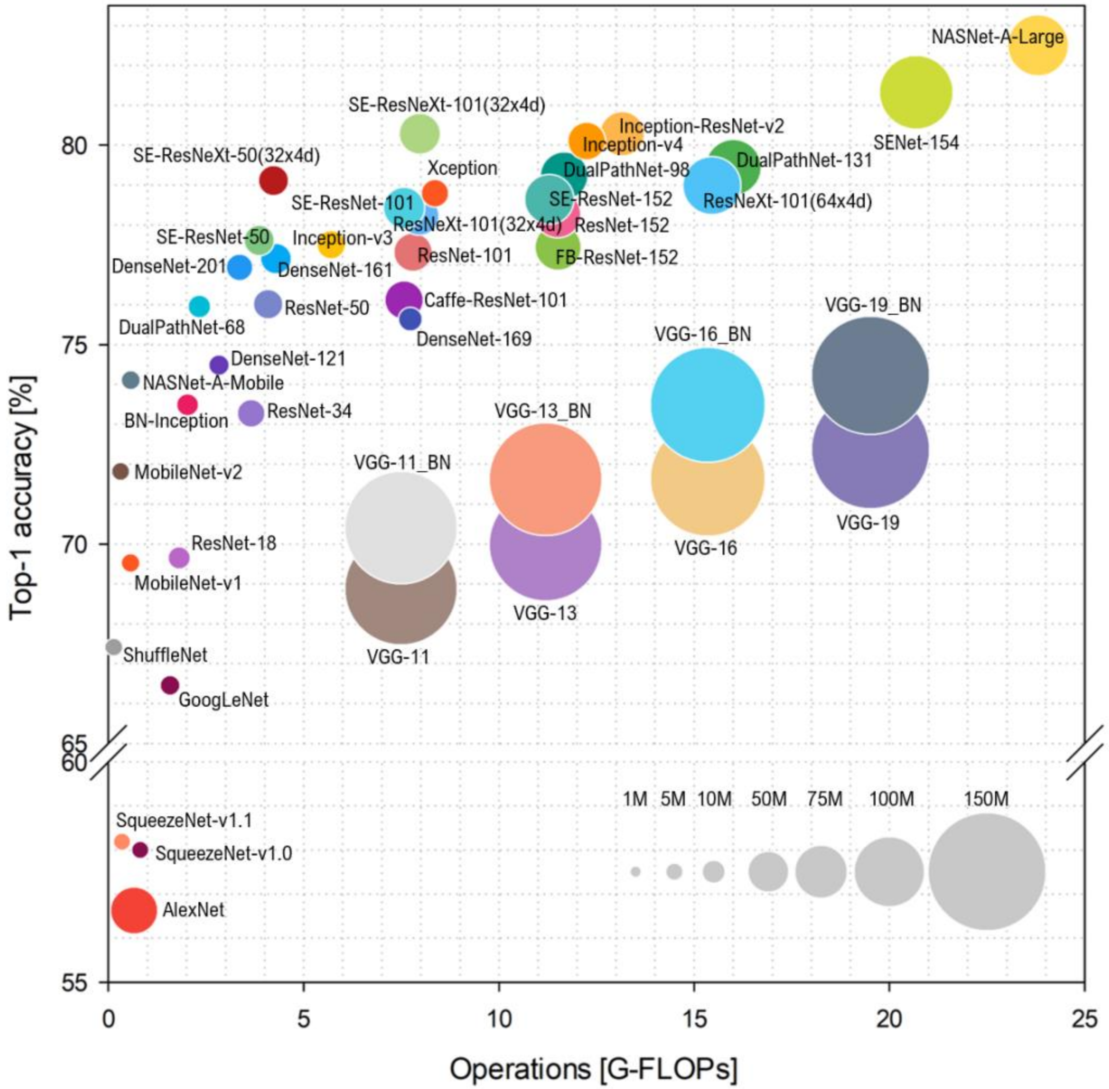
- Competitive to recent state-of-the-art specifically designed for NAS



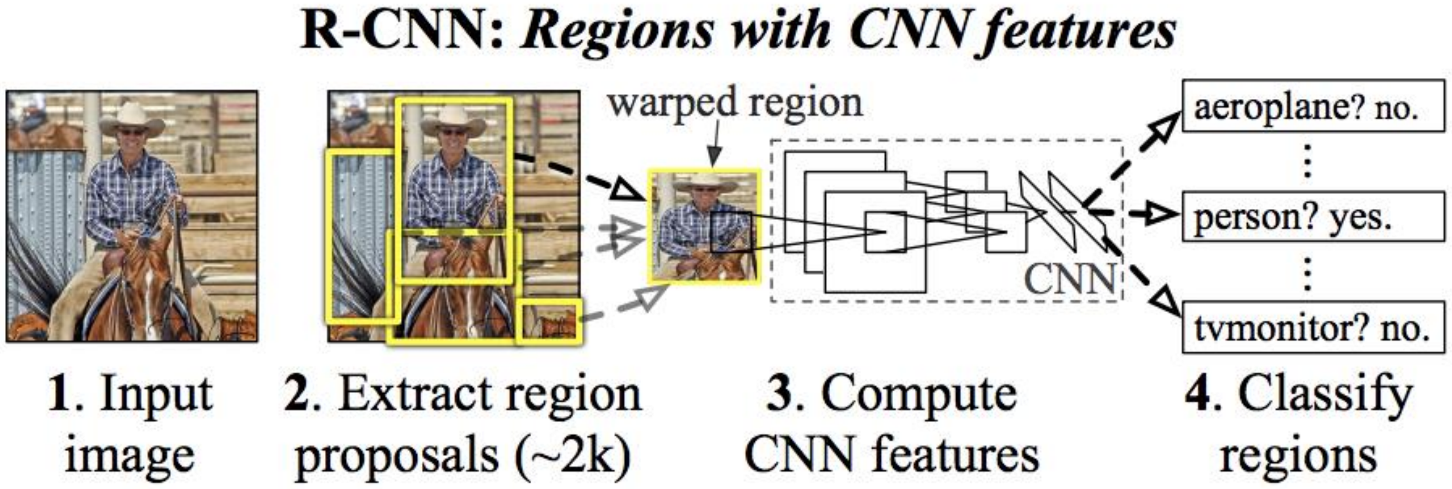




State-of-the-Art

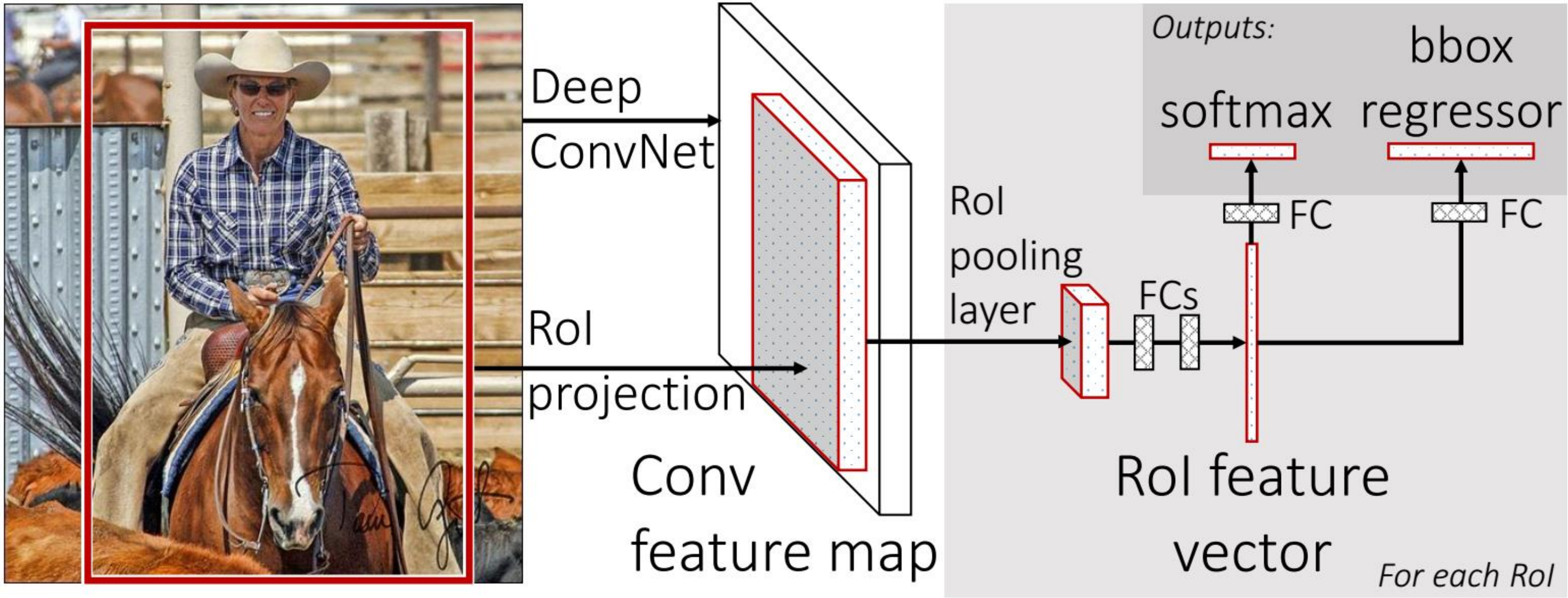


R-CNNs
Fully Convolutional Siamese Nets for Tracking



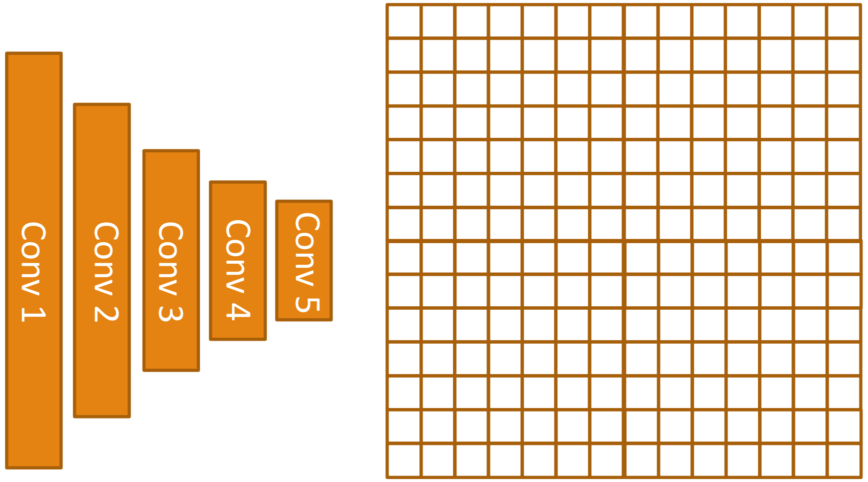
Sliding window on feature maps

- SPPnet [He2014]
- Fast R-CNN [Girshick2015]



Fast R-CNN: Steps

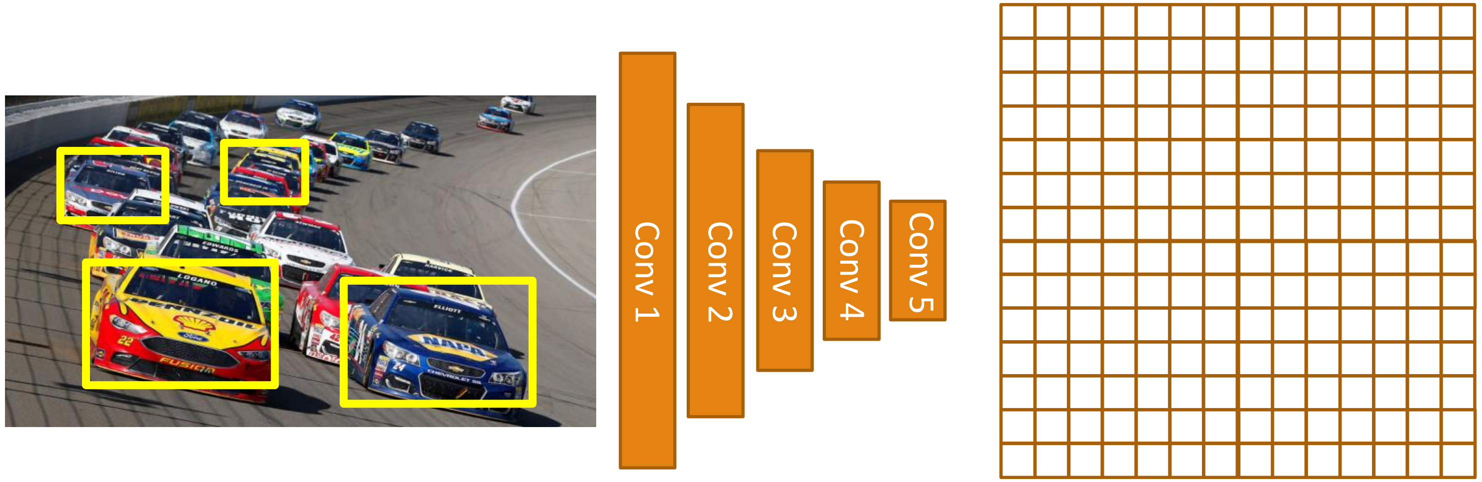
- Process the whole image up to conv5



Conv 5 feature map

Fast R-CNN: Steps

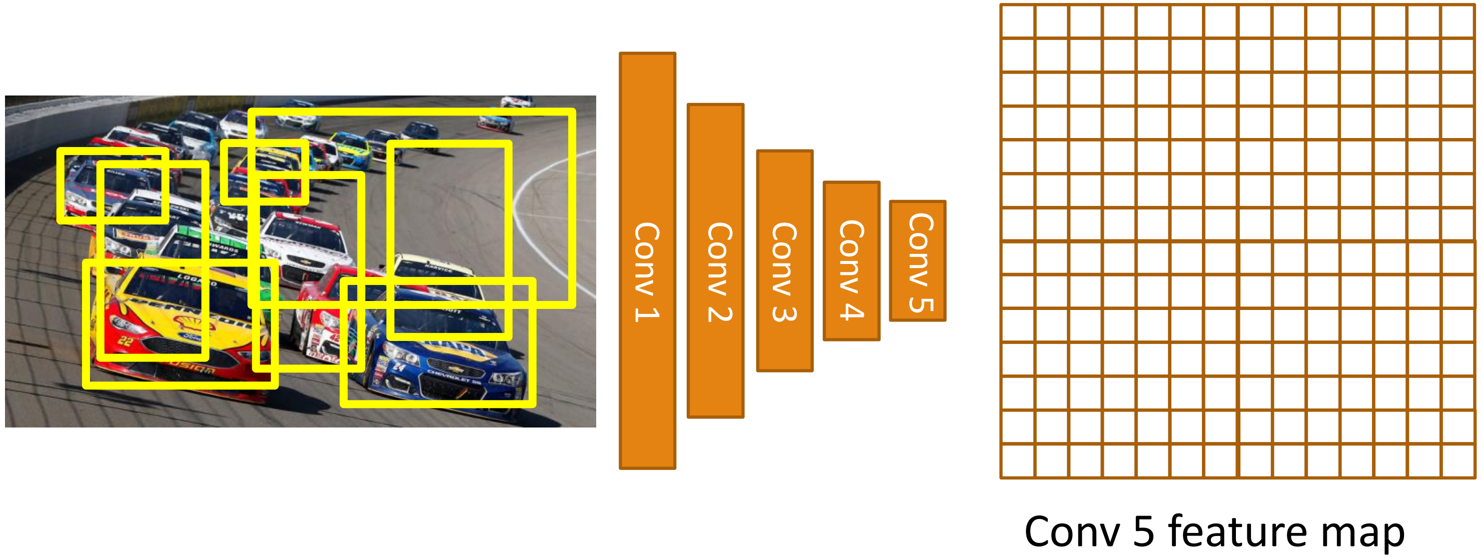
- Process the whole image up to conv5
- Compute possible locations for objects



Conv 5 feature map

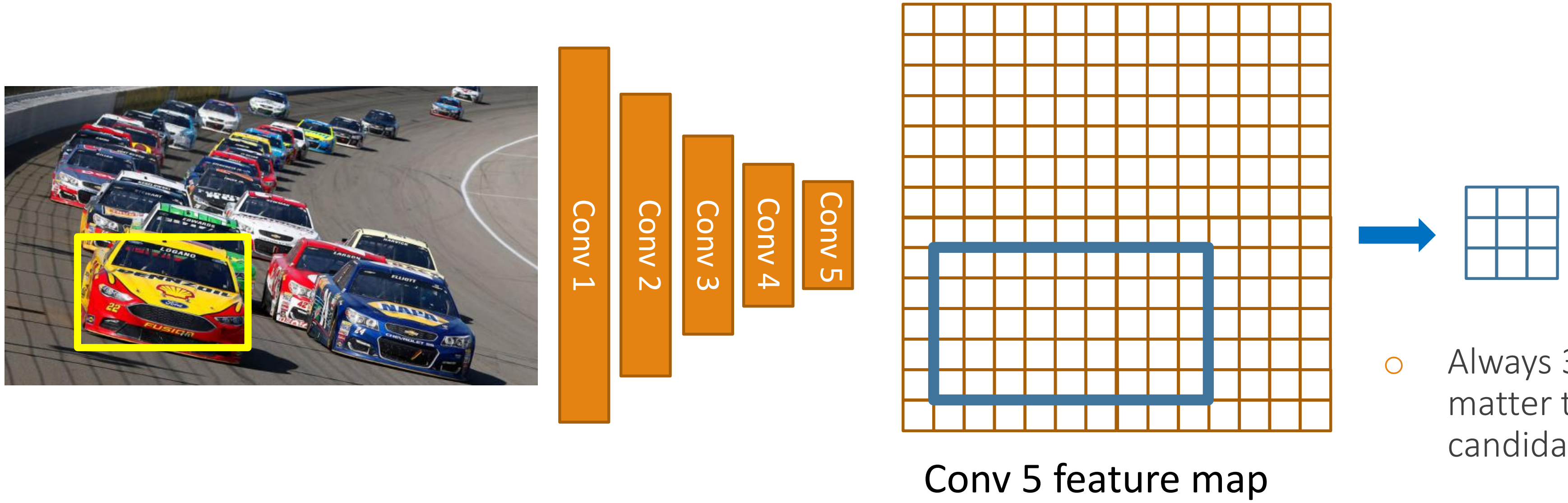
Fast R-CNN: Steps

- Process the whole image up to conv5
- Compute possible locations for objects (some correct, most wrong)



Fast R-CNN: Steps

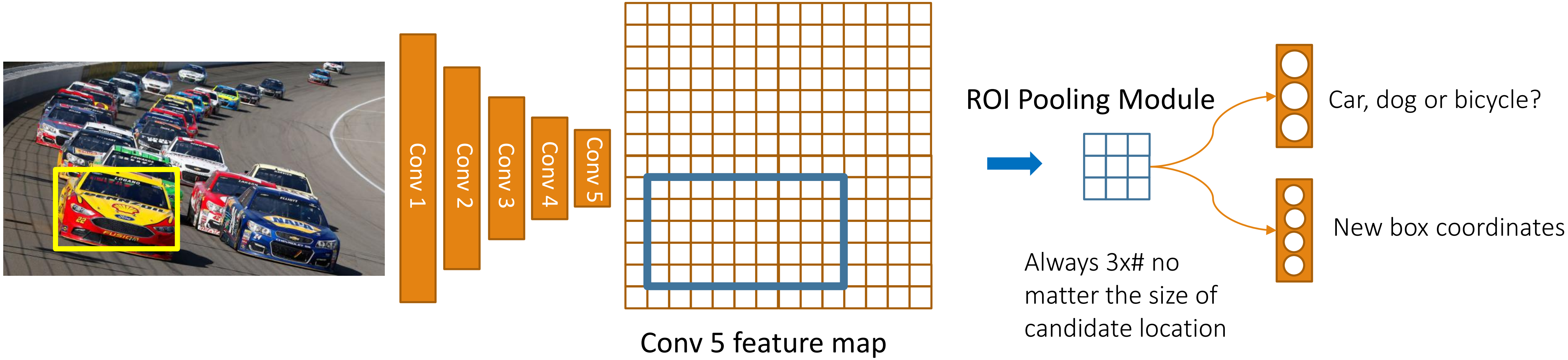
- Process the whole image up to conv5
- Compute possible locations for objects
- Given single location → ROI pooling module extracts fixed length feature



- Always 3x3 no matter the size of candidate location

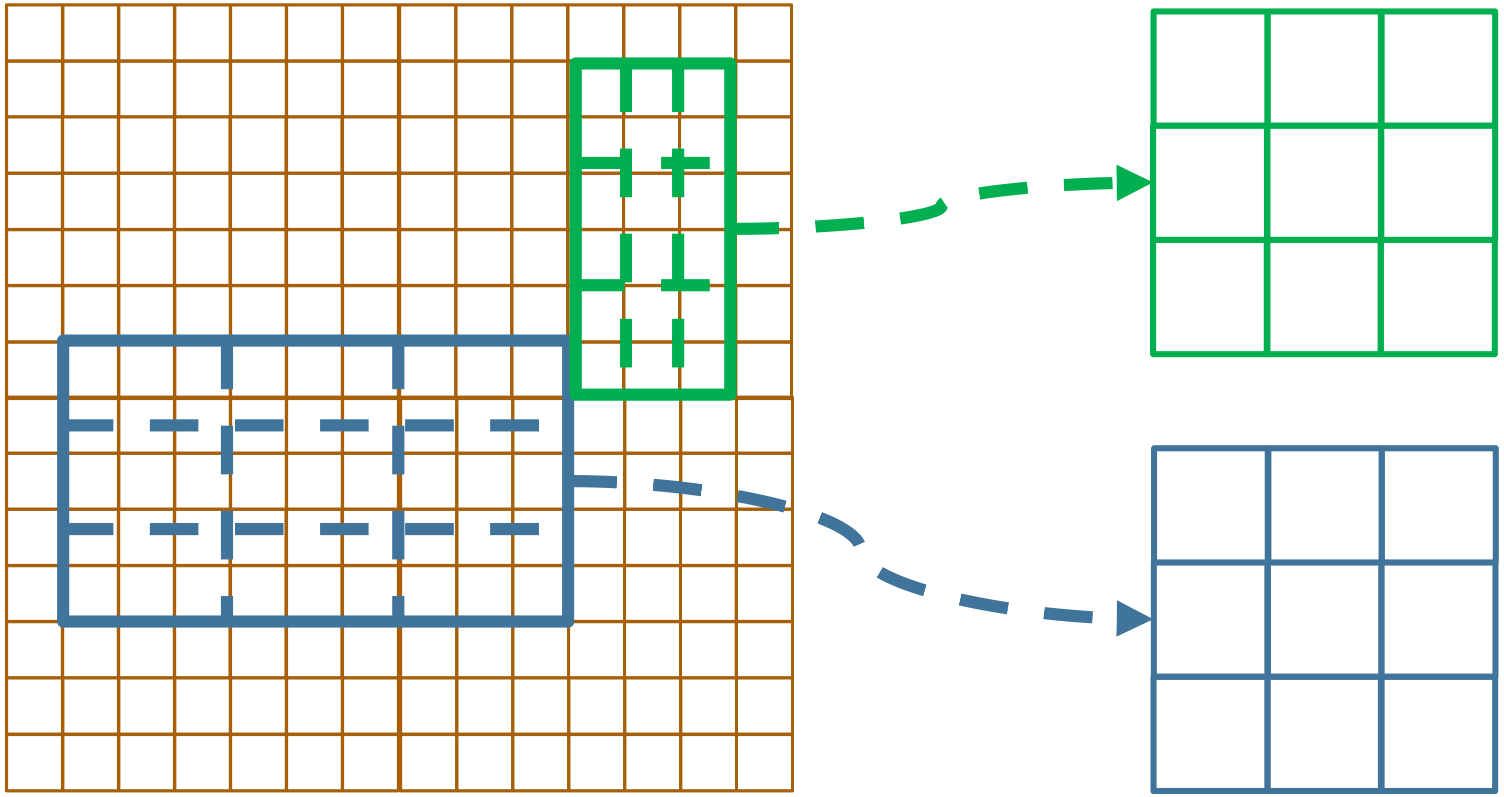
Fast R-CNN: Steps

- Process the whole image up to conv5
- Compute possible locations for objects
- Given single location → ROI pooling module extracts fixed length feature
- Connect to two final layers, 1 for classification, 1 for box refinement



Region-of-Interest (ROI) Pooling Module

- Divide feature map in TxT cells
  - The cell size will change depending on the size of the candidate location

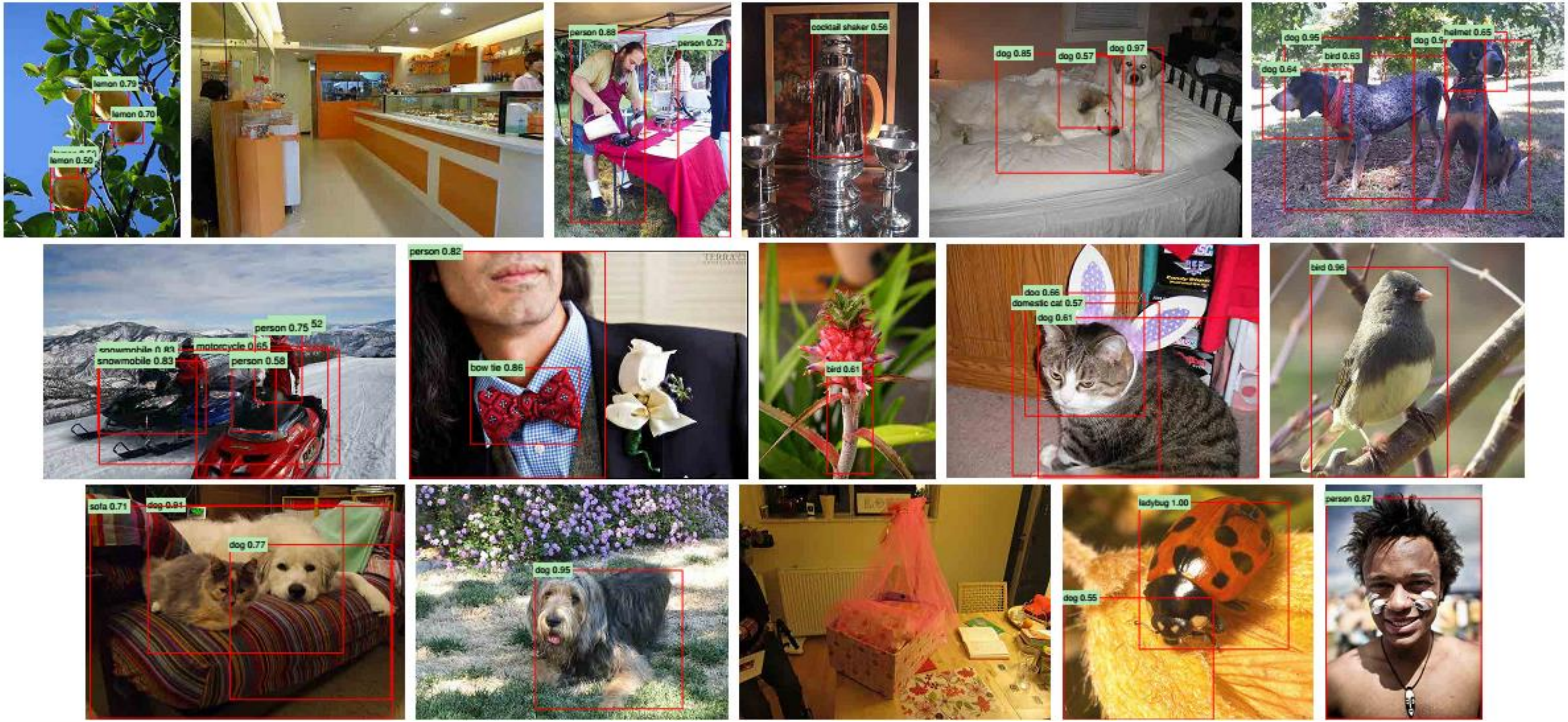


Always 3x3 no matter the size of candidate location

Smart fine-tuning

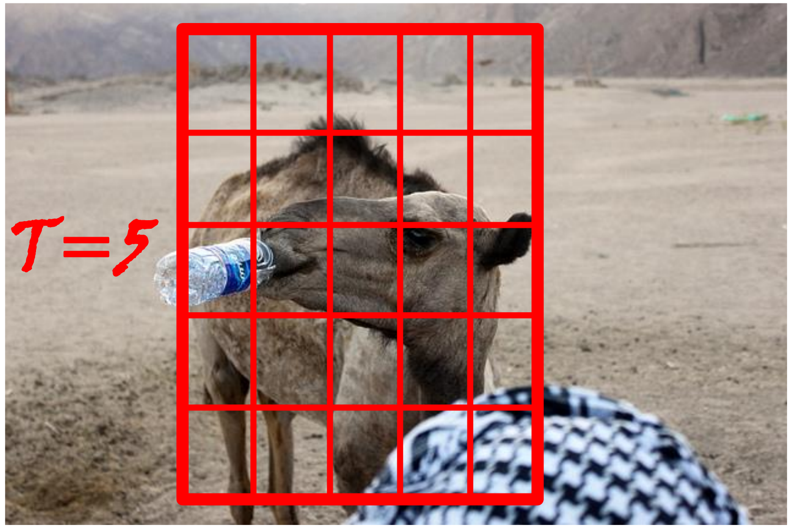
- Normally samples in a mini-batch completely random
- Instead, organize mini-batches by ROIs
- 1 mini-batch = N (images) $\times \frac{R}{N}$ (candidate locations)
- Feature maps shared → training speed-up by a factor of $\frac{R}{N}$
- Mini-batch samples might be correlated
  - In Fast R-CNN that was not observed

Some results



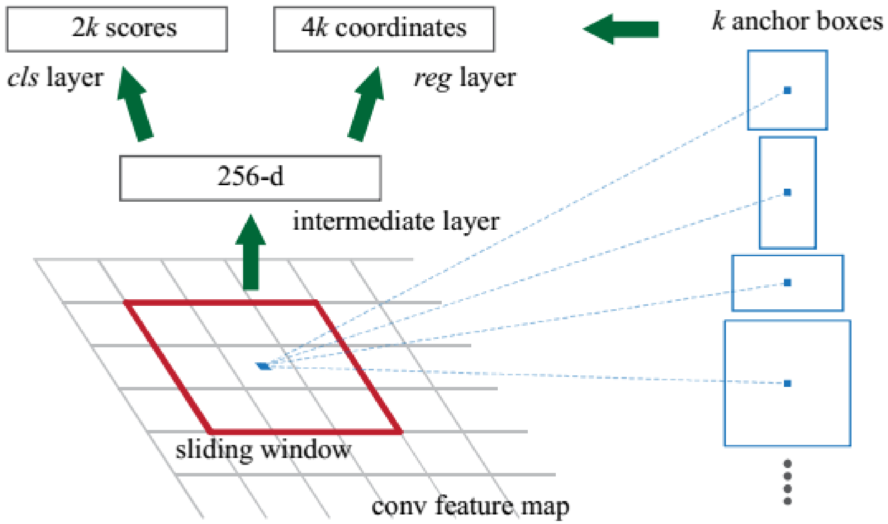
Fast-RCNN

- Reuse convolutions for different candidate boxes
  - Compute feature maps only once
- Region-of-Interest pooling
  - Define stride relatively → box width divided by predefined number of “poolings” T
  - Fixed length vector
- End-to-end training!
- (Very) Accurate object detection
- (Very) Faster
  - Less than a second per image
- External box proposals needed



Faster R-CNN [Girshick2016]

- Fast R-CNN → external candidate locations
- Faster R-CNN → deep network proposes candidate locations
- Slide the feature map → k anchor boxes per slide



Region Proposal Network

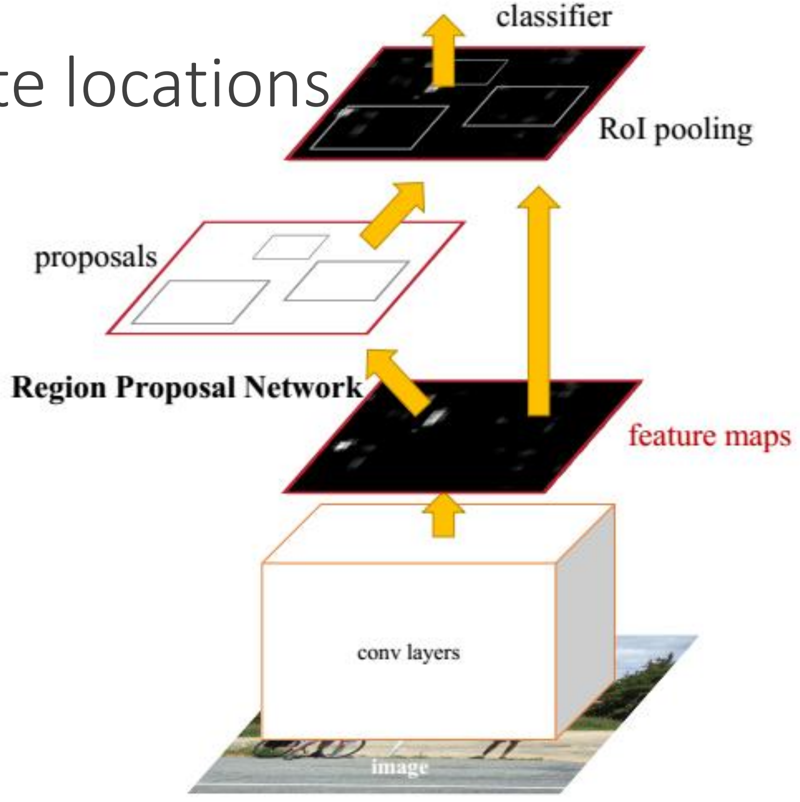
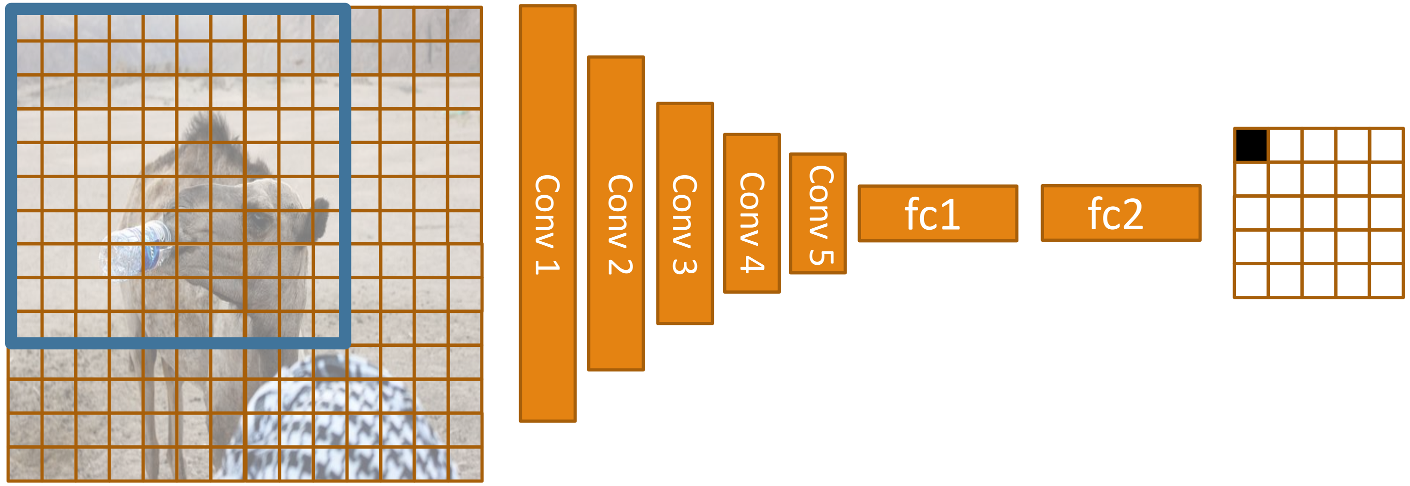


Figure 2: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network.

Going Fully Convolutional [LongCVPR2014]

- Image larger than network input → slide the network

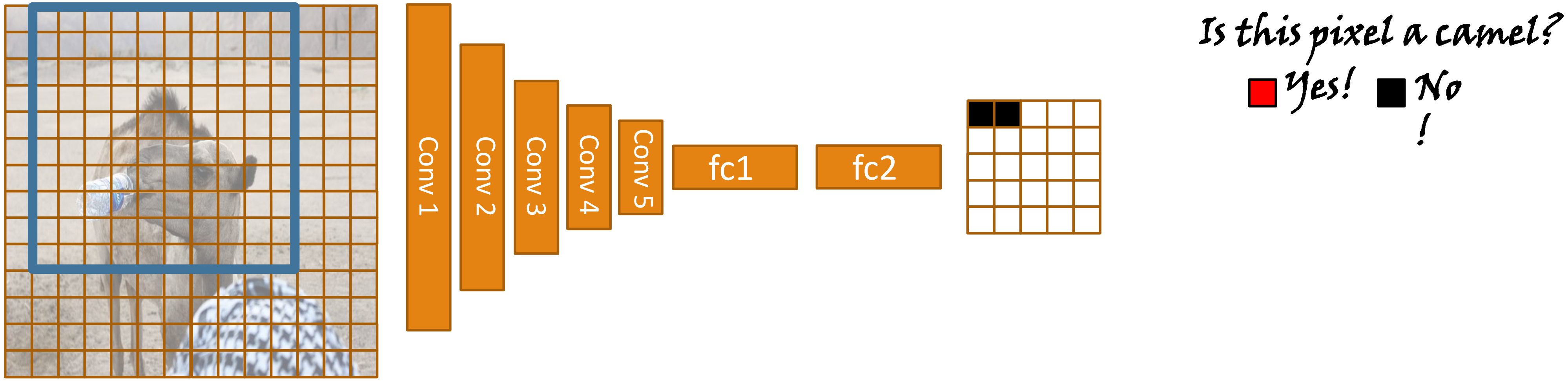


Is this pixel a camel?

Yes! No!

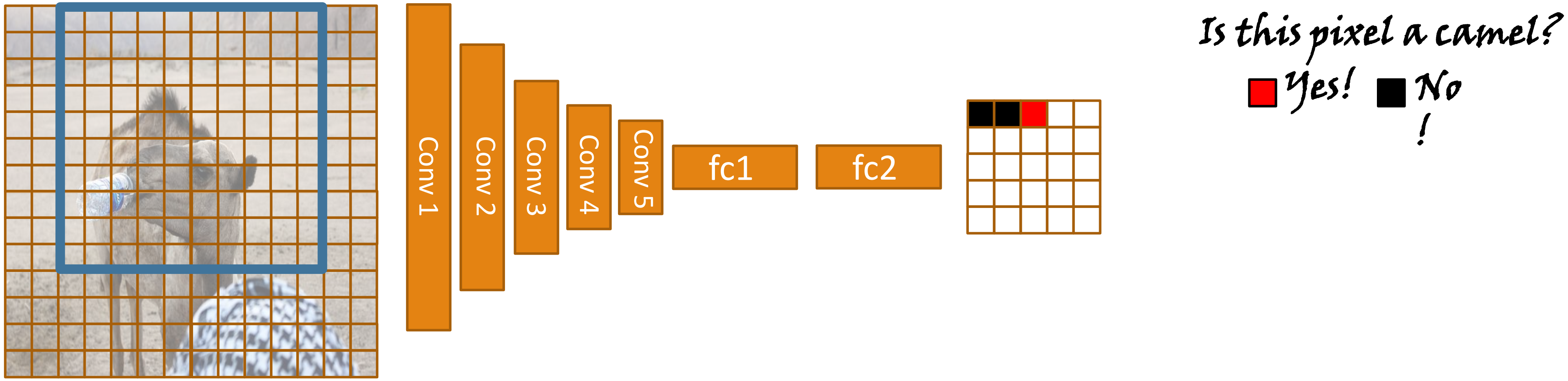
Going Fully Convolutional [LongCVPR2014]

- Image larger than network input → slide the network



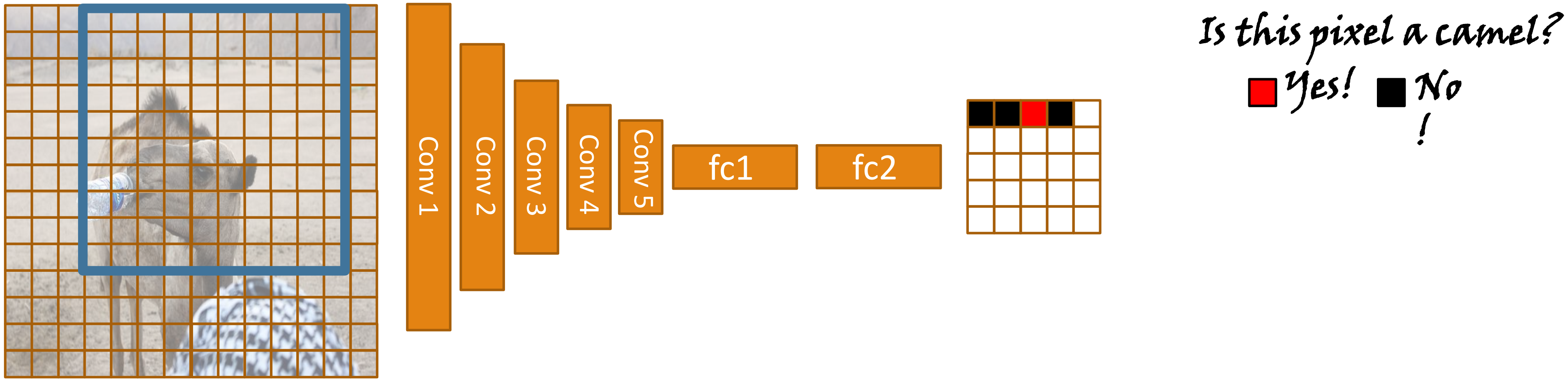
Going Fully Convolutional [LongCVPR2014]

- Image larger than network input → slide the network



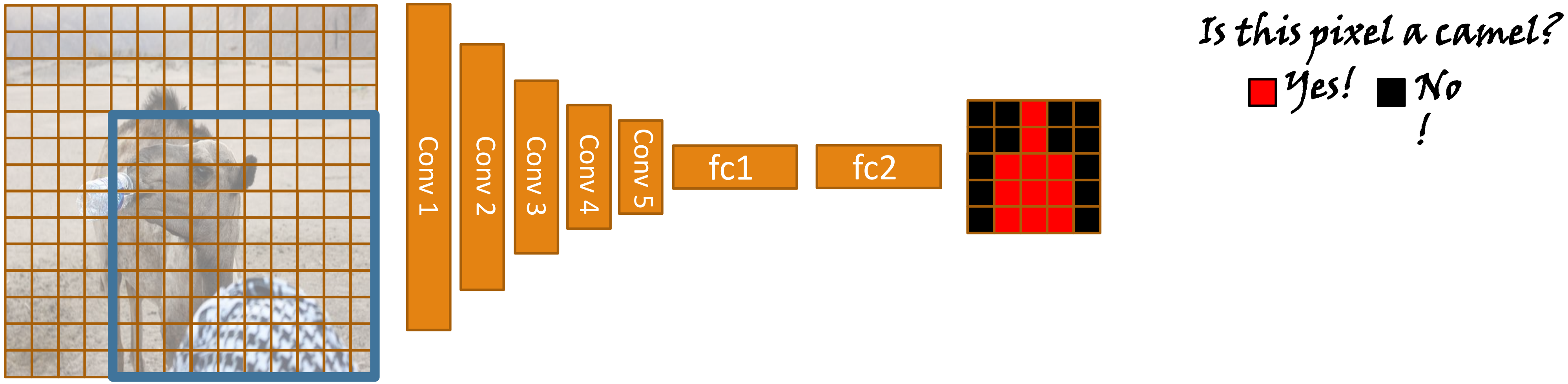
Going Fully Convolutional [LongCVPR2014]

- Image larger than network input → slide the network

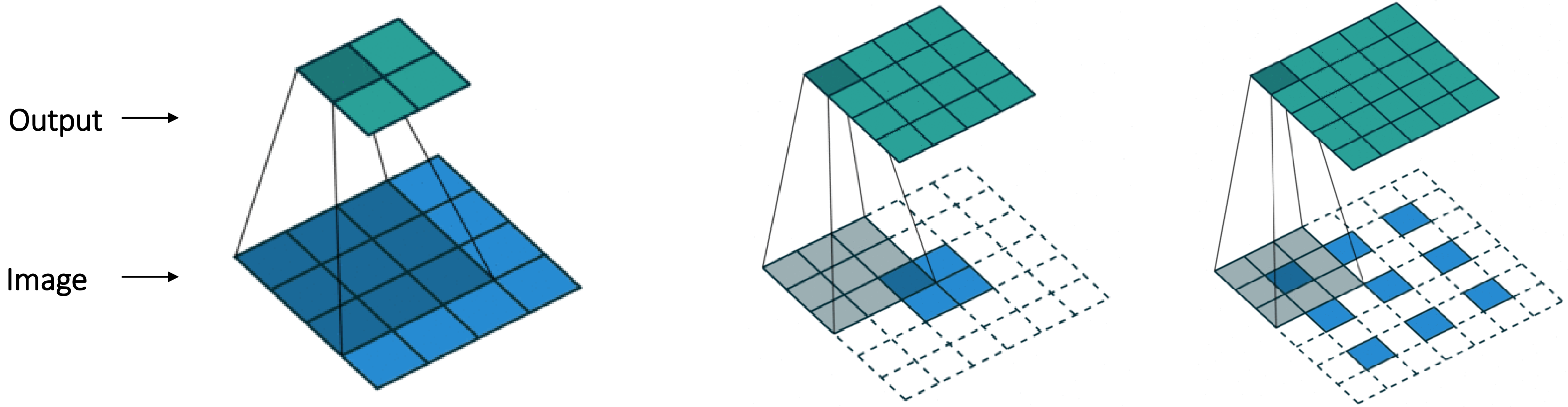


Going Fully Convolutional [LongCVPR2014]

- Image larger than network input → slide the network



Deconvolutional modules



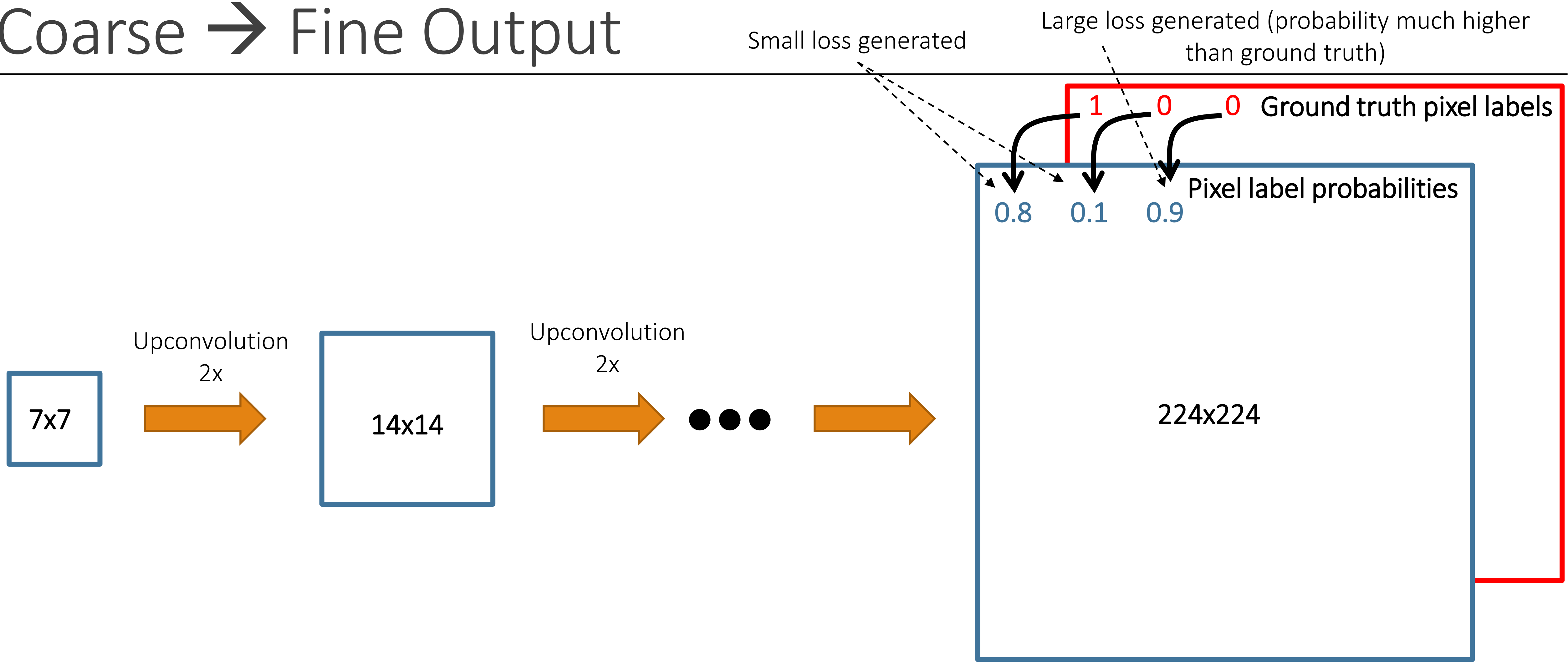
<u>Convolution</u>
No padding, no strides

<u>Upconvolution</u>
Padding, no strides

<u>Upconvolution</u>
Padding, strides

More visualizations: https://github.com/vdumoulin/conv_arithmetic

Coarse → Fine Output



Siamese Networks for Tracking

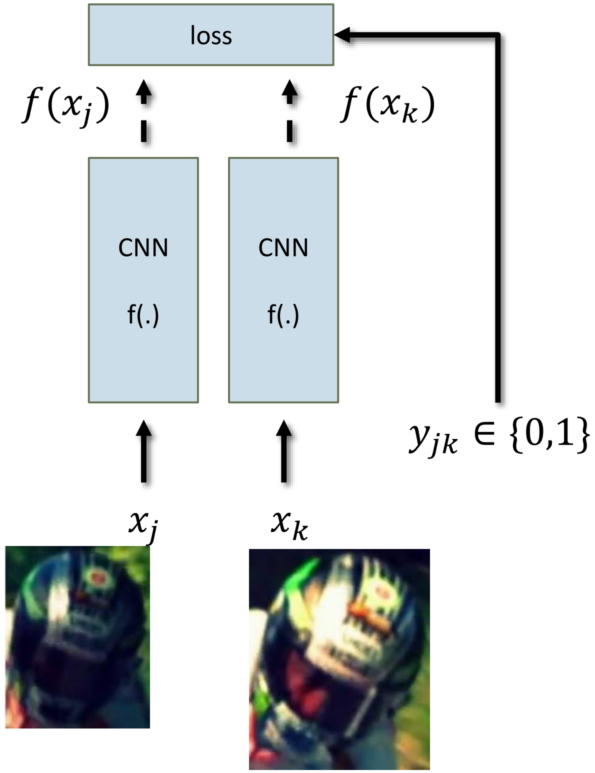
- Siamese Instance Search for Tracking, R. Tao, E. Gavves, A. Smeulders, CVPR 2016

- While tracking, the only definitely correct training example is the first frame
  - All others are inferred by the algorithm
- If the "inferred positives" are correct, then the model is already good enough and no update is needed
- If the "inferred positives" are incorrect, updating the model using wrong positive examples will eventually destroy the model
- Basically, the backbone for most trackers nowadays, especially for long-term tracking

Basic idea

- No model updates through time to avoid model contamination
- Instead, learn invariance model $f(\boldsymbol{dx})$
  - invariances shared between objects
  - reliable, external, rich, category-independent, data
- Assumption
  - The appearance variances are shared amongst object and categories
  - Learning can accurate enough to identify common appearance variances
- Solution: Use a Siamese Network to compare patches between images
  - Then "tracking" equals finding the most similar patch at each frame (no temporal modelling)

Training



Marginal Contrastive Loss:

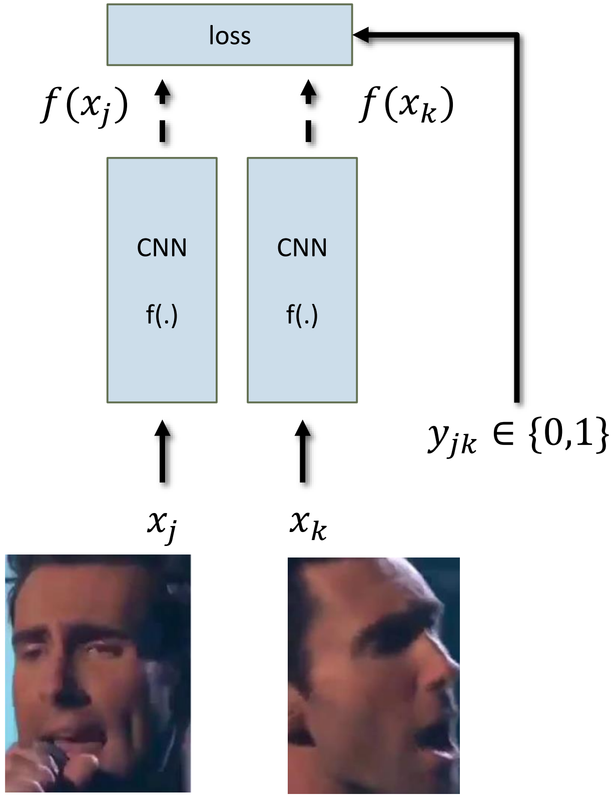
$$L(x_j, x_k, y_{jk}) = \frac{1}{2} y_{jk} D^2 + \frac{1}{2}(1 - y_{jk}) \max(0, \sigma - D^2)$$

$$D = \left\| f(x_j) - f(x_k) \right\|_2$$

Matching function (after learning):

$$m(x_j, x_k) = f(x_j) \cdot f(x_k)$$

Training



Marginal Contrastive Loss:

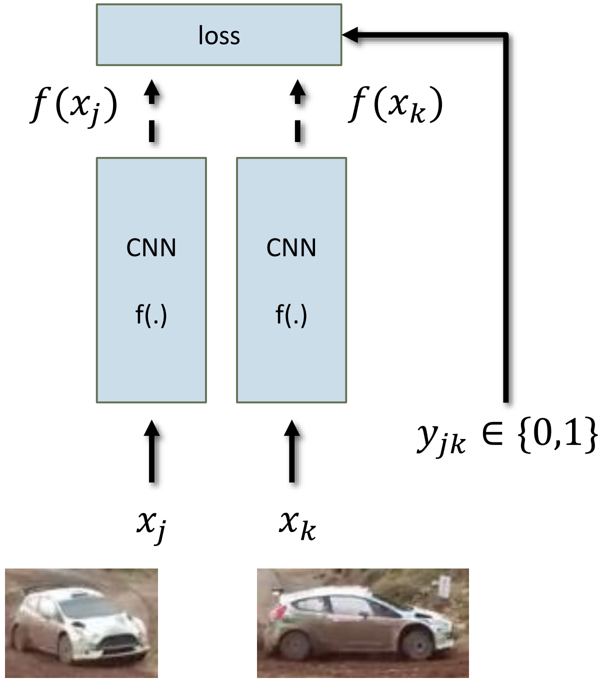
$$L(x_j, x_k, y_{jk}) = \frac{1}{2} y_{jk} D^2 + \frac{1}{2}(1 - y_{jk}) \max(0, \sigma - D^2)$$

$$D = \left\| f(x_j) - f(x_k) \right\|_2$$

Matching function (after learning):

$$m(x_j, x_k) = f(x_j) \cdot f(x_k)$$

Training



Marginal Contrastive Loss:

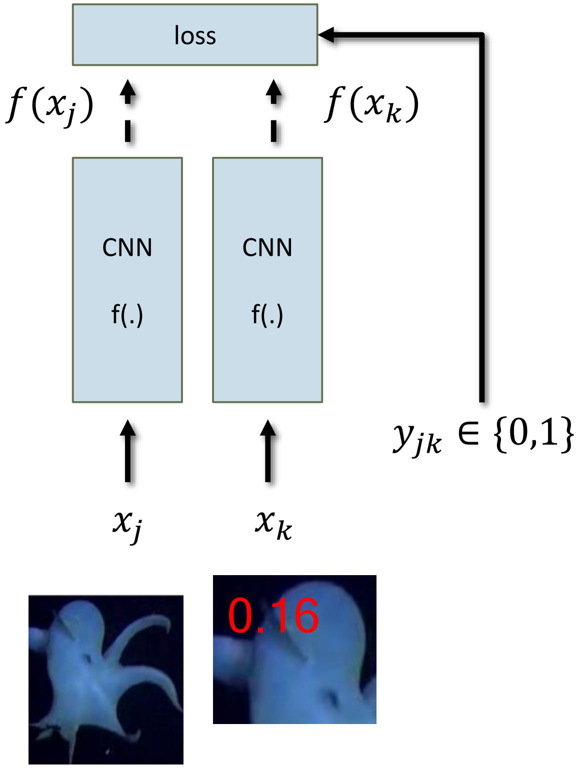
$$L(x_j, x_k, y_{jk}) = \frac{1}{2} y_{jk} D^2 + \frac{1}{2}(1 - y_{jk}) \max(0, \sigma - D^2)$$

$$D = \left\| f(x_j) - f(x_k) \right\|_2$$

Matching function (after learning):

$$m(x_j, x_k) = f(x_j) \cdot f(x_k)$$

Testing

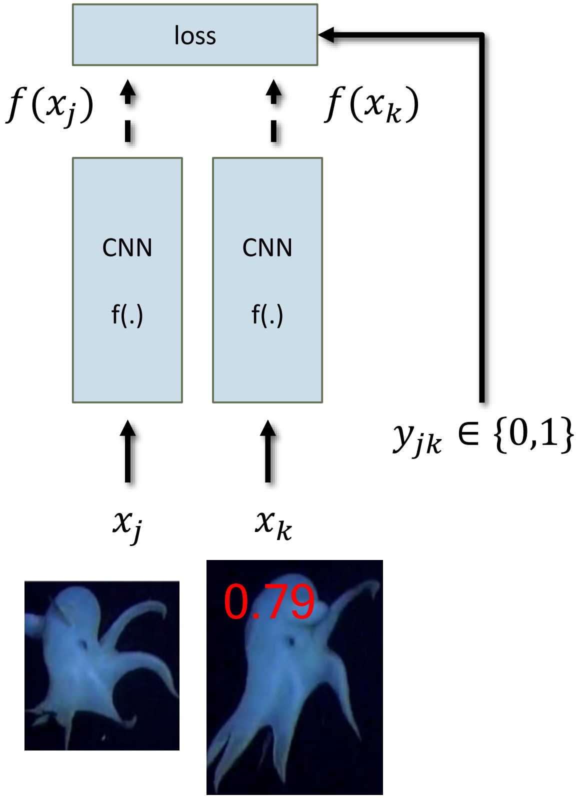


Predicting the next location

1. Define query x_0 at $t = 0$
2. Set current target location x_t
3. Measure similarity $\mathrm{s}_{\mathrm{t}+1}^{k} = s(x_0, x_{t+1}^{k})$ of x_0 with multiple boxes ${x'}_{t+1}$ sampled around x_t
4. Select next target location with maximum similarity $\mathrm{s}_{\mathrm{t}+1}^{k}$
5. Go to 2

 VISUAL OBJECT TRACKING BY NATURAL LANGUAGE SPECIFICATION, *TO APPEAR IN IEEE CVPR 2017*

Testing

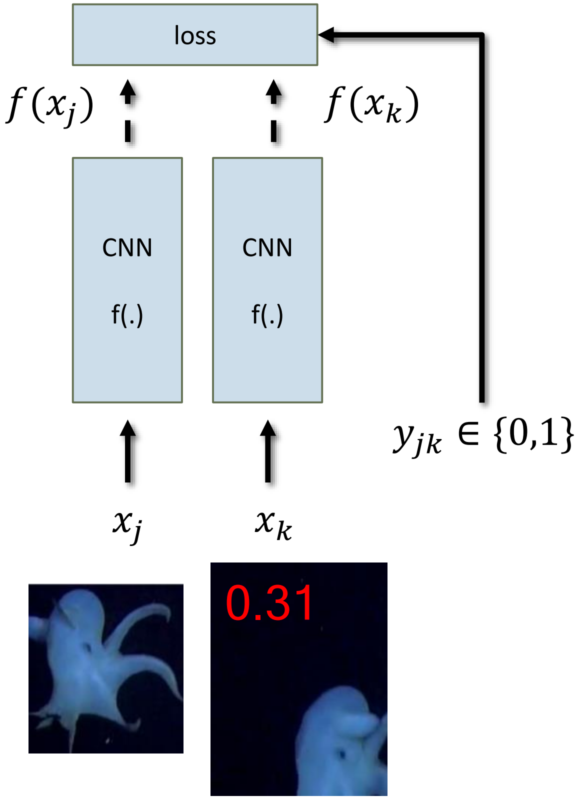


Predicting the next location

1. Define query x_0 at $t = 0$
2. Set current target location x_t
3. Measure similarity $\mathrm{s}_{\mathrm{t}+1}^{k} = s(x_0, x_{t+1}^{k})$ of x_0 with multiple boxes x'_{t+1} sampled around x_t
4. Select next target location with maximum similarity $\mathrm{s}_{\mathrm{t}+1}^{k}$
5. Go to 2

 VISUAL OBJECT TRACKING BY NATURAL LANGUAGE SPECIFICATION, *TO APPEAR IN IEEE CVPR 2017*

Testing



Predicting the next location

1. Define query x_0 at $t = 0$
2. Set current target location x_t
3. Measure similarity $\mathrm{s}_{\mathrm{t}+1}^{k} = s(x_0, x_{t+1}^{k})$ of x_0 with multiple boxes x'_{t+1} sampled around x_t
4. Select next target location with maximum similarity $\mathrm{s}_{\mathrm{t}+1}^{k}$
5. Go to 2

 VISUAL OBJECT TRACKING BY NATURAL LANGUAGE SPECIFICATION, *TO APPEAR IN IEEE CVPR 2017*

Video Neural Networks

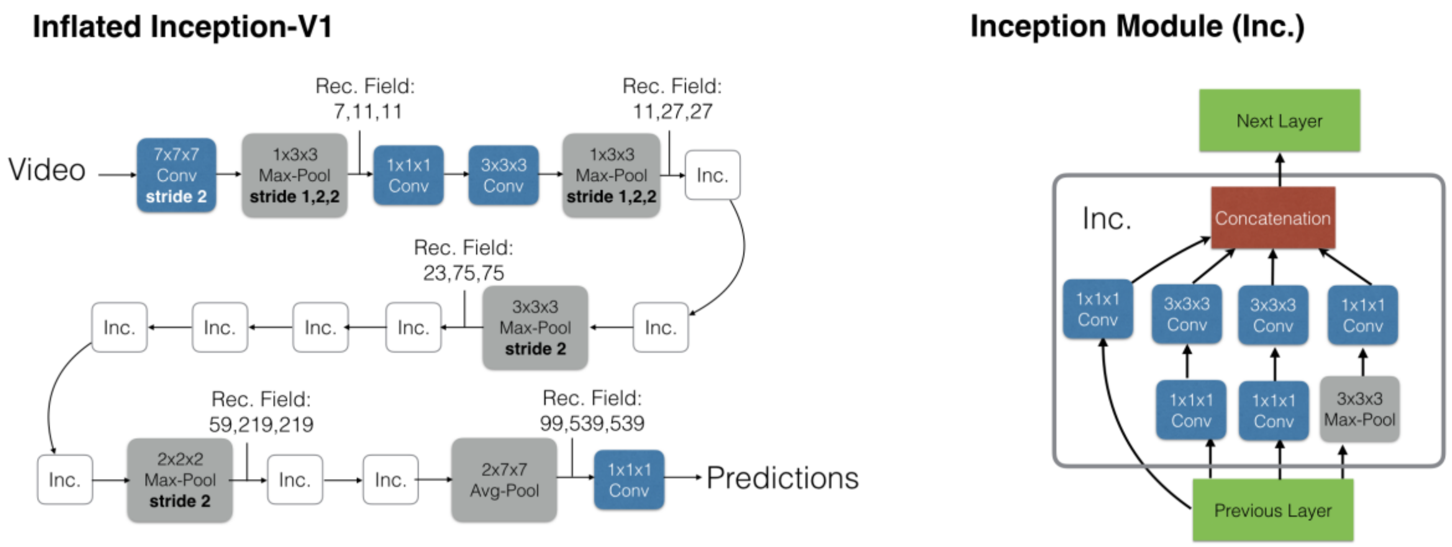


Figure 3. The Inflated Inception-V1 architecture (left) and its detailed inception submodule (right). The strides of convolution and pooling operators are 1 where not specified, and batch normalization layers, ReLu's and the softmax at the end are not shown. The theoretical sizes of receptive field sizes for a few layers in the network are provided in the format "time,x,y" – the units are frames and pixels. The predictions are obtained convolutionally in time and averaged.

Basic idea

- Replace 2D convolutions with 3D convolutions
- Train on same domain data
  - Videos

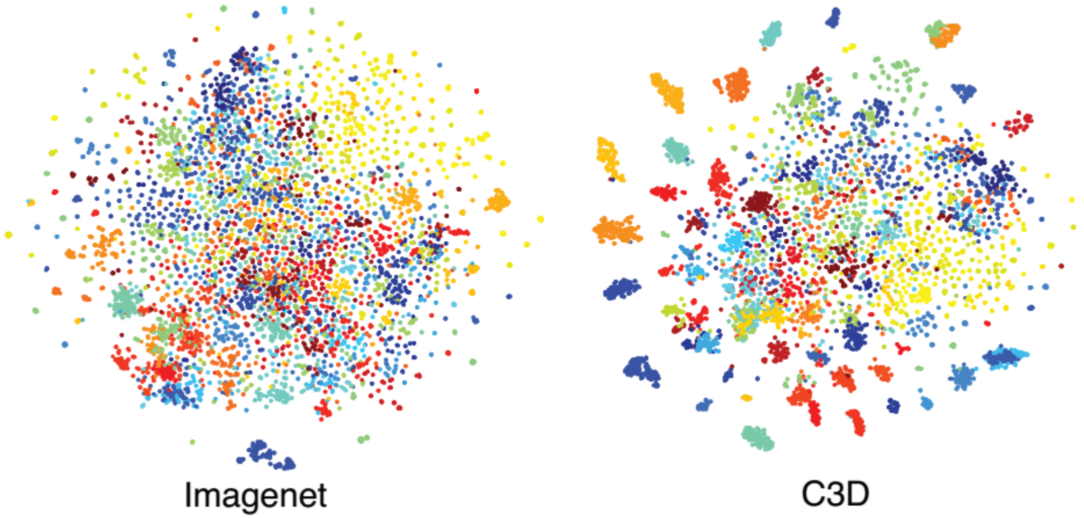
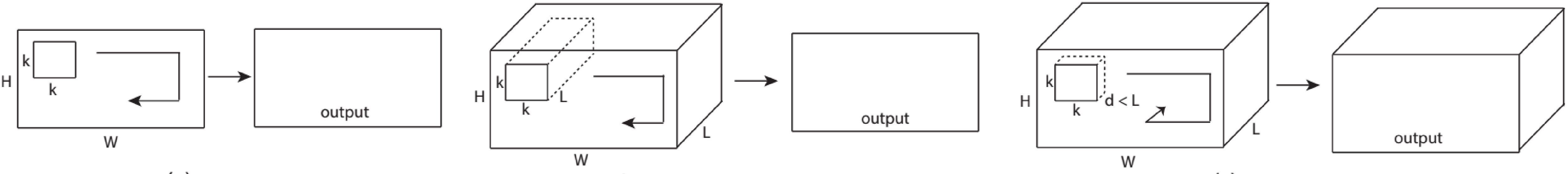


Figure 6. **Feature embedding**. Feature embedding visualizations of Imagenet and C3D on UCF101 dataset using t-SNE [43]. C3D features are semantically separable compared to Imagenet suggesting that it is a better feature for videos. Each clip is visualized as a point and clips belonging to the same action have the same color. Best viewed in color.



(a) 2D convolution (b) 2D convolution on multiple frames (c) 3D convolution

Some results

- Generally, it works pretty nicely
- Not for all temporal tasks though, as we will see later on in the course

Method	Accuracy (%)
Imagenet + linear SVM	68.8
iDT w/ BoW + linear SVM	76.2
Deep networks [18]	65.4
Spatial stream network [36]	72.6
LRCN [6]	71.1
LSTM composite model [39]	75.8
C3D (1 net) + linear SVM	82.3
C3D (3 nets) + linear SVM	**85.2**
iDT w/ Fisher vector [31]	87.9
Temporal stream network [36]	83.7
Two-stream networks [36]	88.0
LRCN [6]	82.9
LSTM composite model [39]	84.3
Conv. pooling on long clips [29]	88.2
LSTM on long clips [29]	88.6
Multi-skip feature stacking [25]	89.1
C3D (3 nets) + iDT + linear SVM	**90.4**

Table 3. **Action recognition results on UCF101**. C3D compared with baselines and current state-of-the-art methods. Top: simple features with linear SVM; Middle: methods taking only RGB frames as inputs; Bottom: methods using multiple feature combinations.

i3D

- i3D = C3D + Inception
  - Plus some neat tricks
- Take 2D filters and inflate them so that they become 3D filters
- Then, use them as initialization

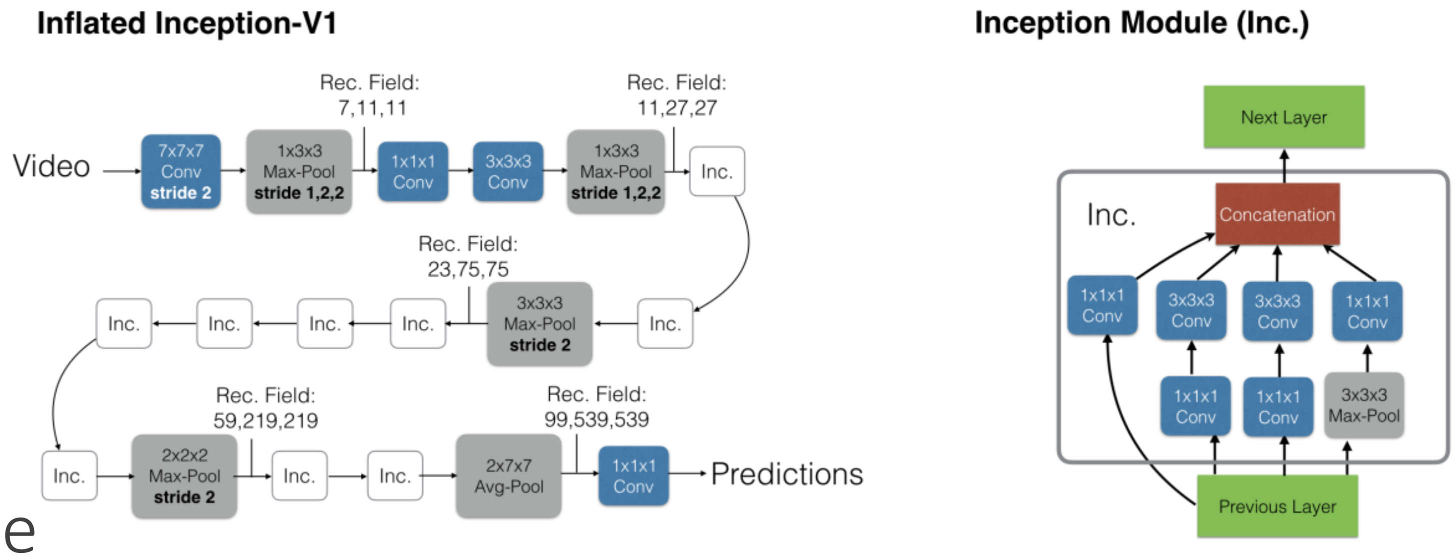


Figure 3. The Inflated Inception-V1 architecture (left) and its detailed inception submodule (right). The strides of convolution and pooling operators are 1 where not specified, and batch normalization layers, ReLu's and the softmax at the end are not shown. The theoretical sizes of receptive field sizes for a few layers in the network are provided in the format "time,x,y" – the units are frames and pixels. The predictions are obtained convolutionally in time and averaged.

	UCF-101			HMDB-51			Kinetics		
Architecture	RGB	Flow	RGB + Flow	RGB	Flow	RGB + Flow	RGB	Flow	RGB + Flow
(a) LSTM	81.0	–	–	36.0	–	–	63.3	–	–
(b) 3D-ConvNet	51.6	–	–	24.3	–	–	56.1	–	–
(c) Two-Stream	83.6	85.6	91.2	43.2	56.3	58.3	62.2	52.4	65.6
(d) 3D-Fused	83.2	85.8	89.3	49.2	55.5	56.8	–	–	67.2
(e) Two-Stream I3D	**84.5**	**90.6**	**93.4**	**49.8**	**61.9**	**66.4**	**71.1**	**63.4**	**74.2**

Table 2. Architecture comparison: (left) training and testing on split 1 of UCF-101; (middle) training and testing on split 1 of HMDB-51; (right) training and testing on Kinetics. All models are based on ImageNet pre-trained Inception-v1, except 3D-ConvNet, a C3D-like [31] model which has a custom architecture and was trained here from scratch. Note that the Two-Stream architecture numbers on individual RGB and Flow streams can be interpreted as a simple baseline which applies a ConvNet independently on 25 uniformly sampled frames then averages the predictions.

Timeception

Timeception for complex action recognition
CVPR 2019

N. Hussein

E. Gavves

A. Smeulders

University of Amsterdam

PROBLEM

Problem Complex Actions

Problem Complex Actions

Preparing Breakfast

Complex Action

Stirring Food

One-action

Problem **Complex Actions**

Preparing Breakfast

Complex Action

1. Long-range
2. Temporal Extent
3. Temporal Dependency

Problem 1. Long-range

get cook put wash

One-action ~2 sec.

Problem 1. Long-range

get cook put wash

One-action ~2 sec.

Complex Action ~30 sec.

Problem 2. Temporal Extent

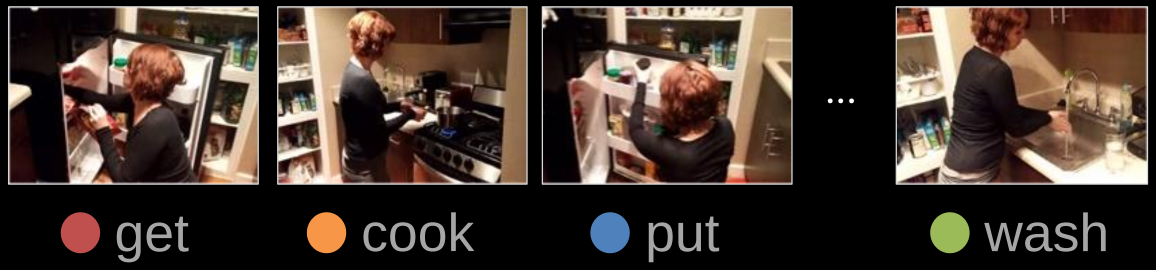
get cook put wash

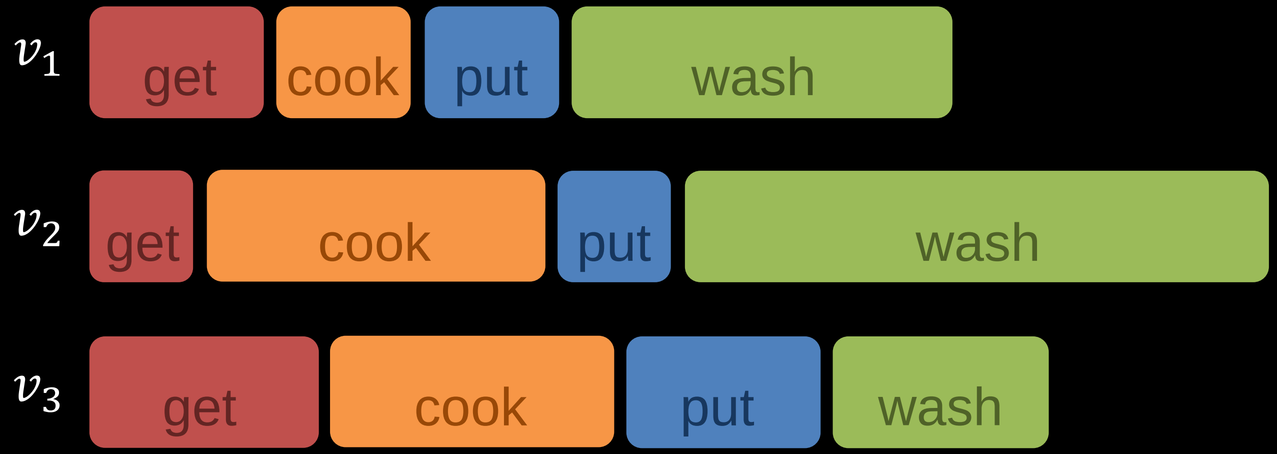
v_1 get | cook | put | wash

v_2 get | cook | put | wash

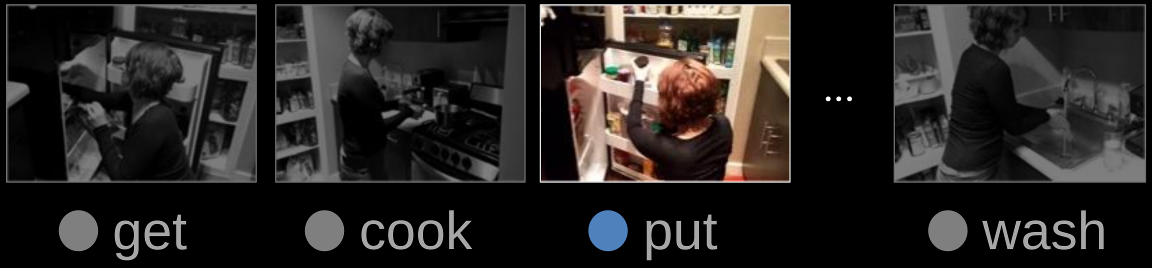
v_3 get | cook | put | wash

Problem 2. Temporal Extent



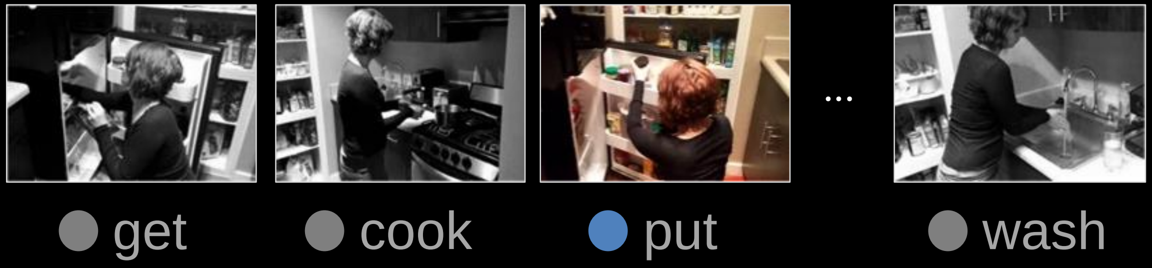


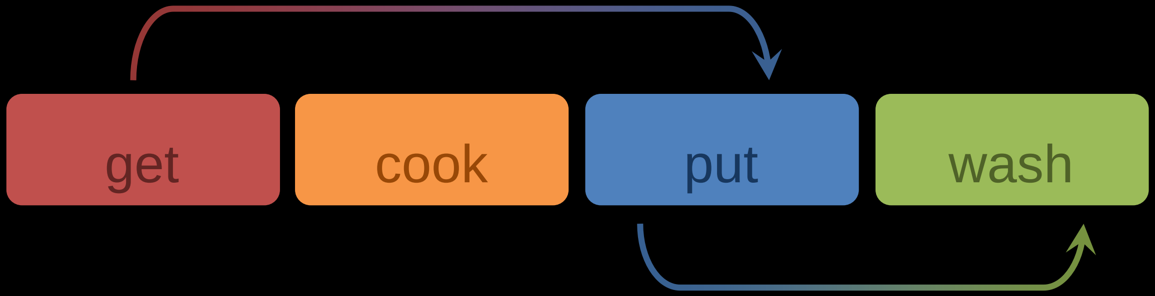
Problem 3. Temporal Dependency



get | cook | put | wash

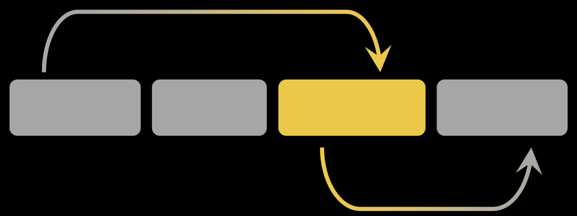
Problem 3. Temporal Dependency





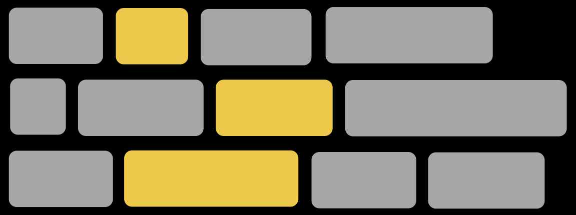
Problem Complex Actions

1. Dependency

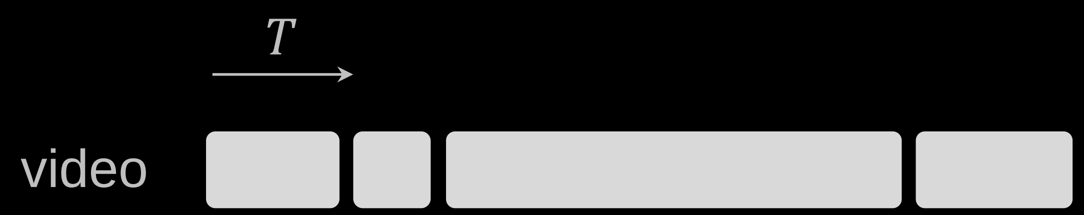


2. Long-range

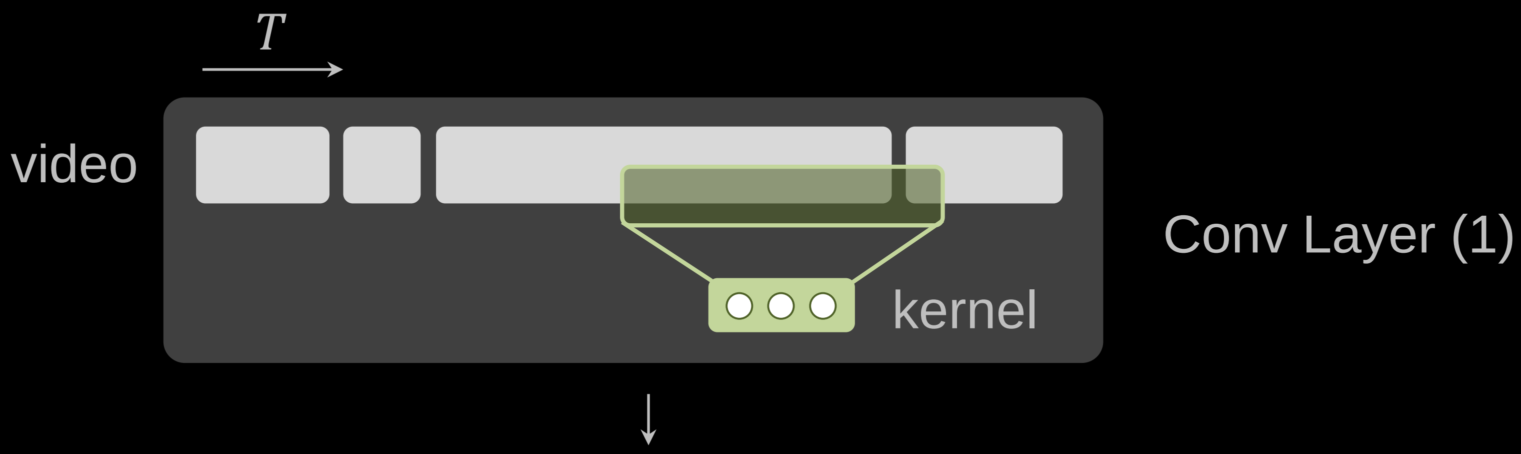
3. Temporal Extent

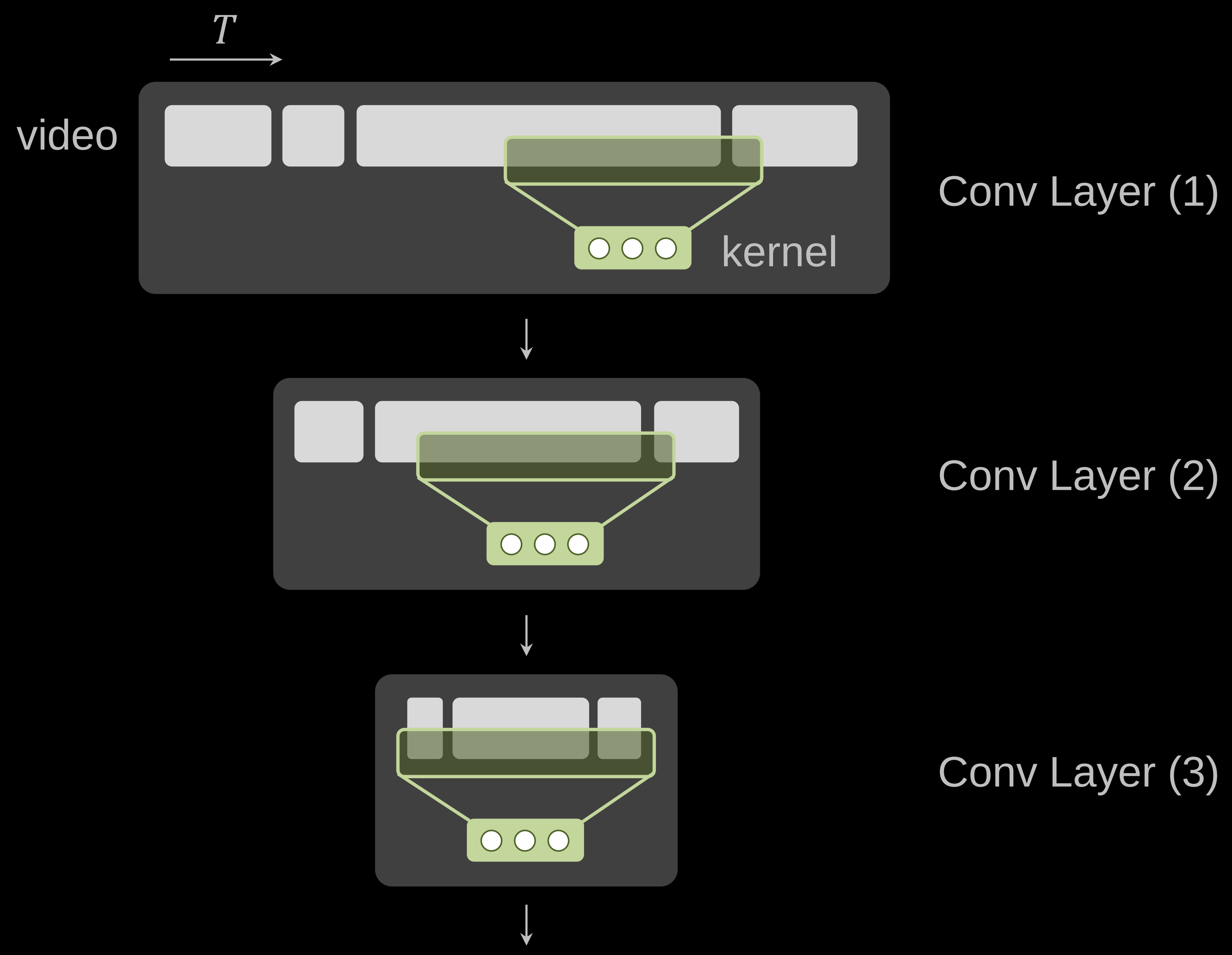


METHOD

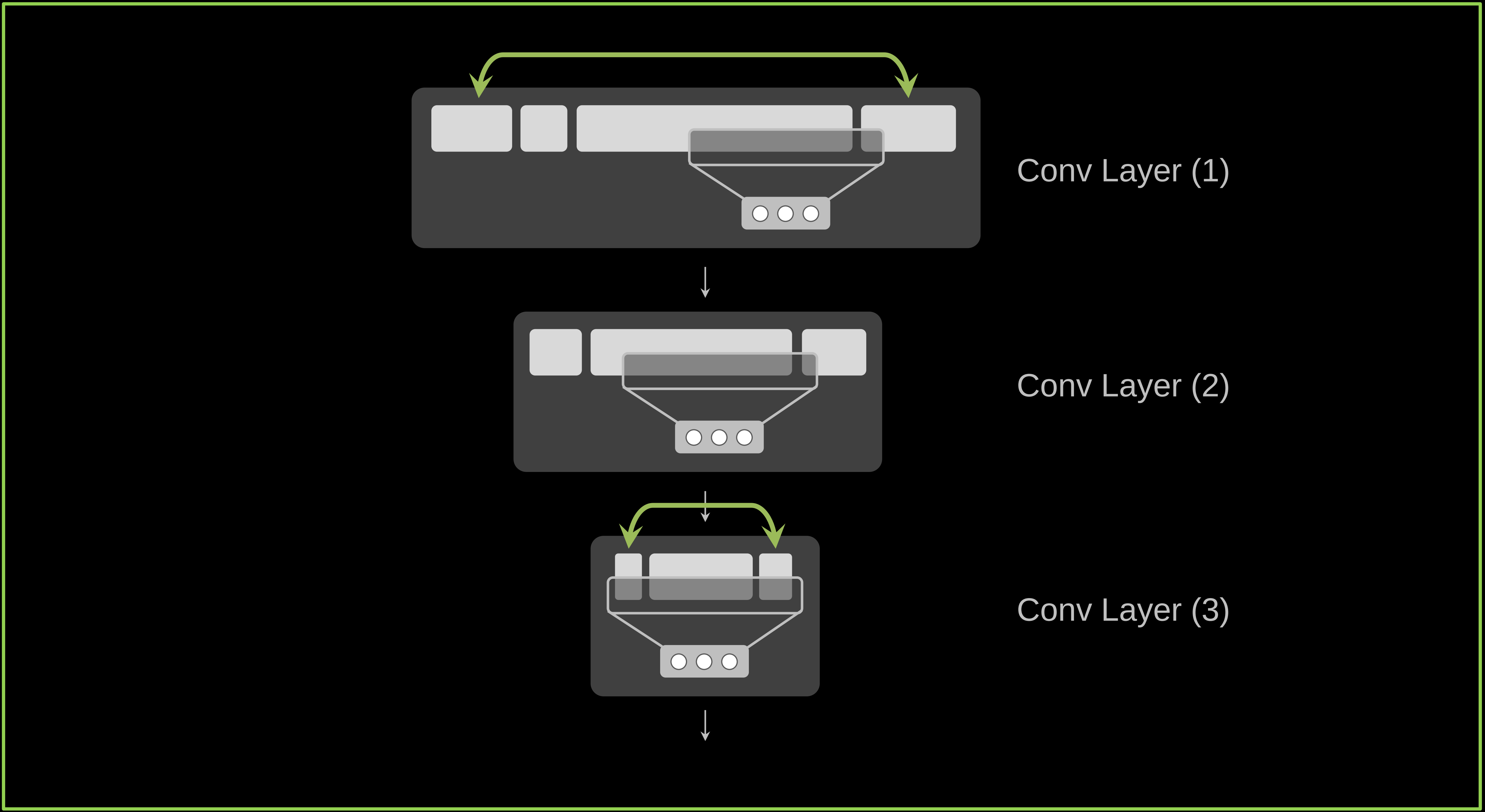


T
video

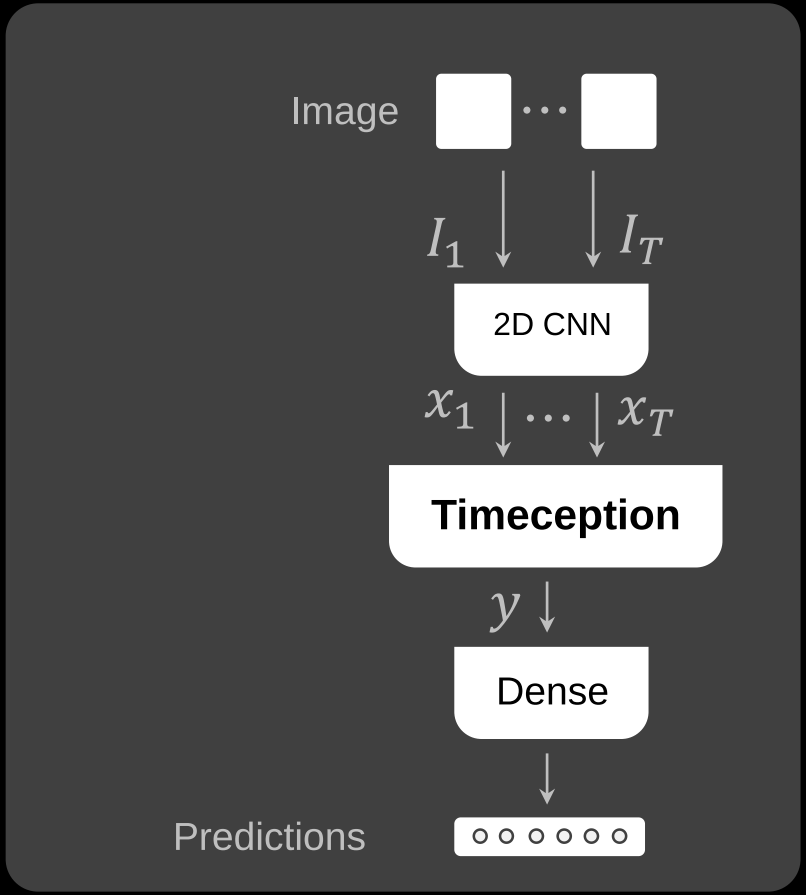

T
video
kernel
Conv Layer (1)



T
video
kernel
Conv Layer (1)
Conv Layer (2)
Conv Layer (3)

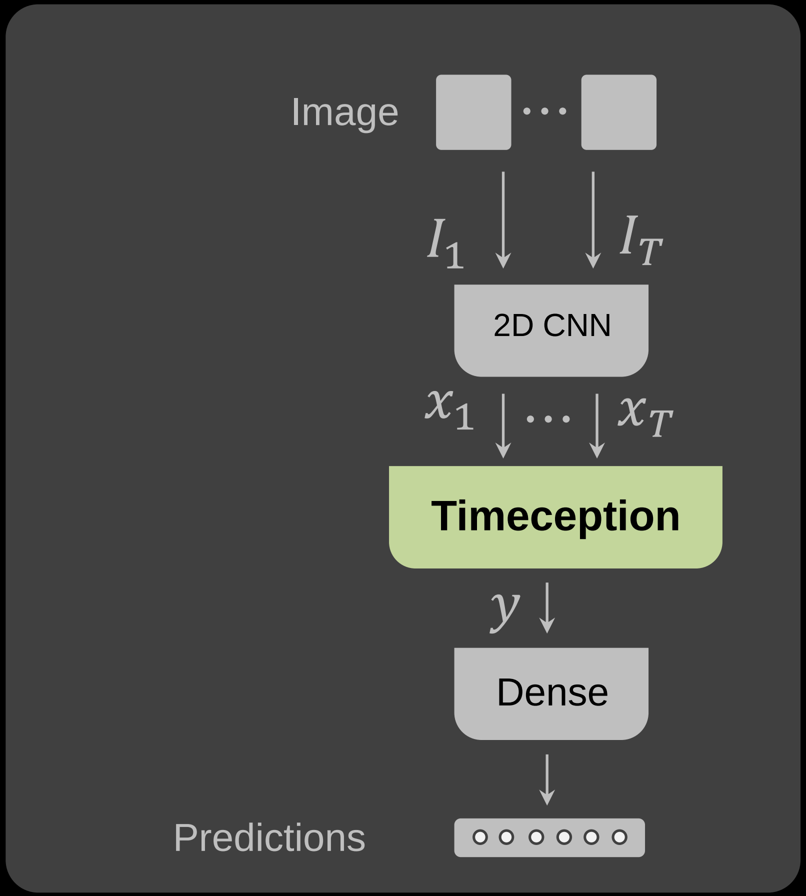

Conv Layer (1)
Conv Layer (2)
Conv Layer (3)

Method Timeception



Model Overview

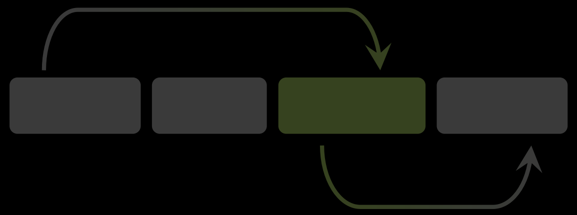
Method Timeception



Model Overview

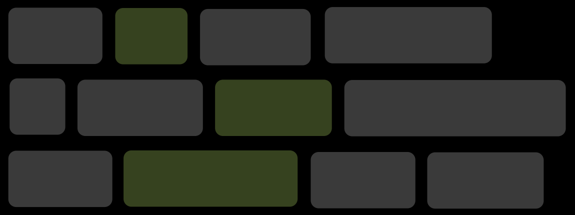
Method Timeception

1. Dependency

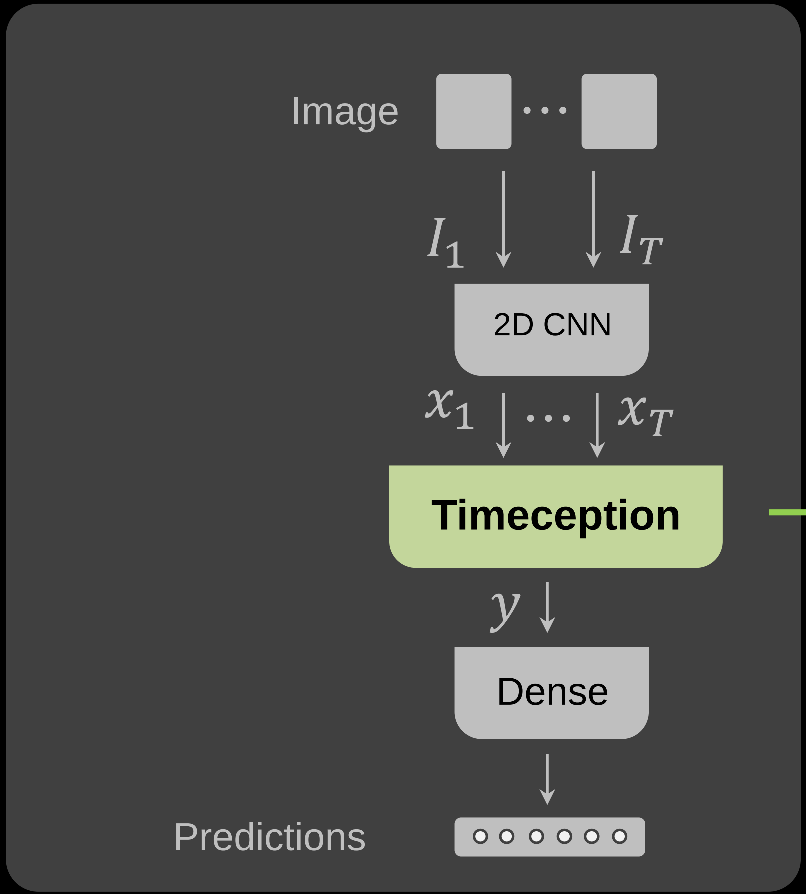


2. Long-range

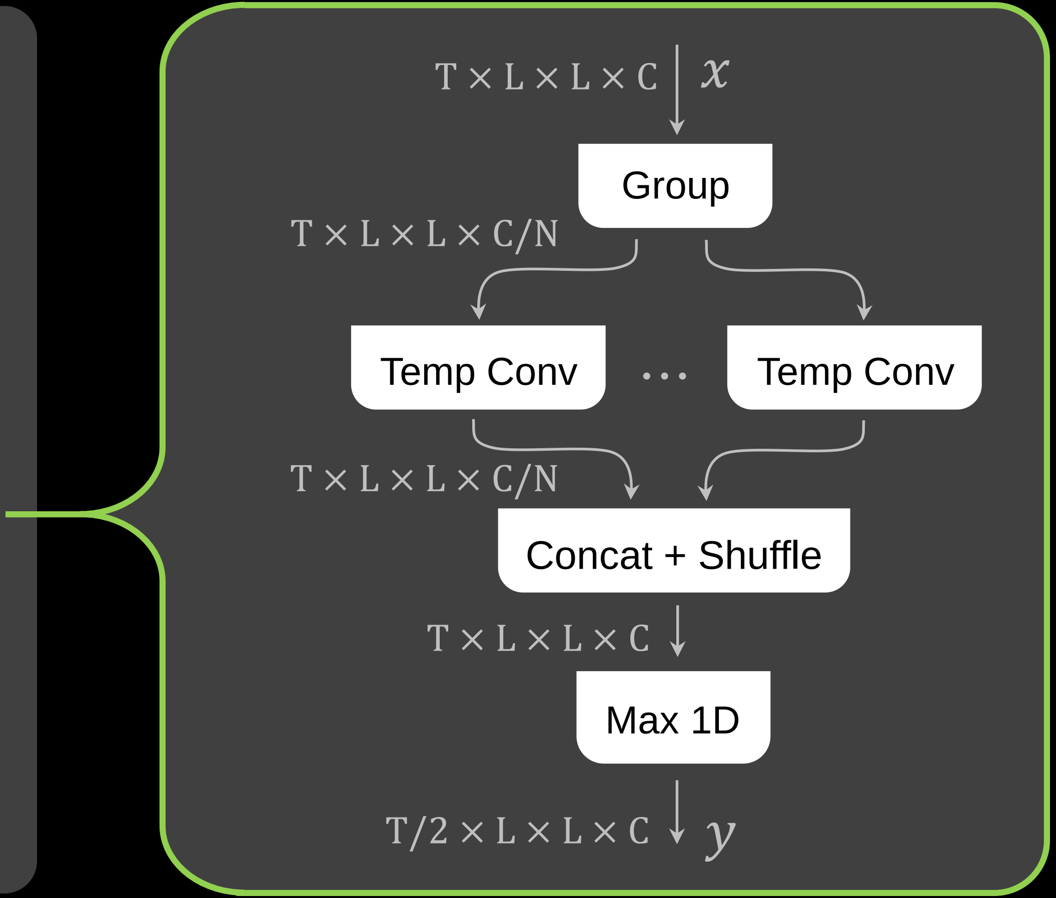
3. Temporal Extent



Method **Efficiency**

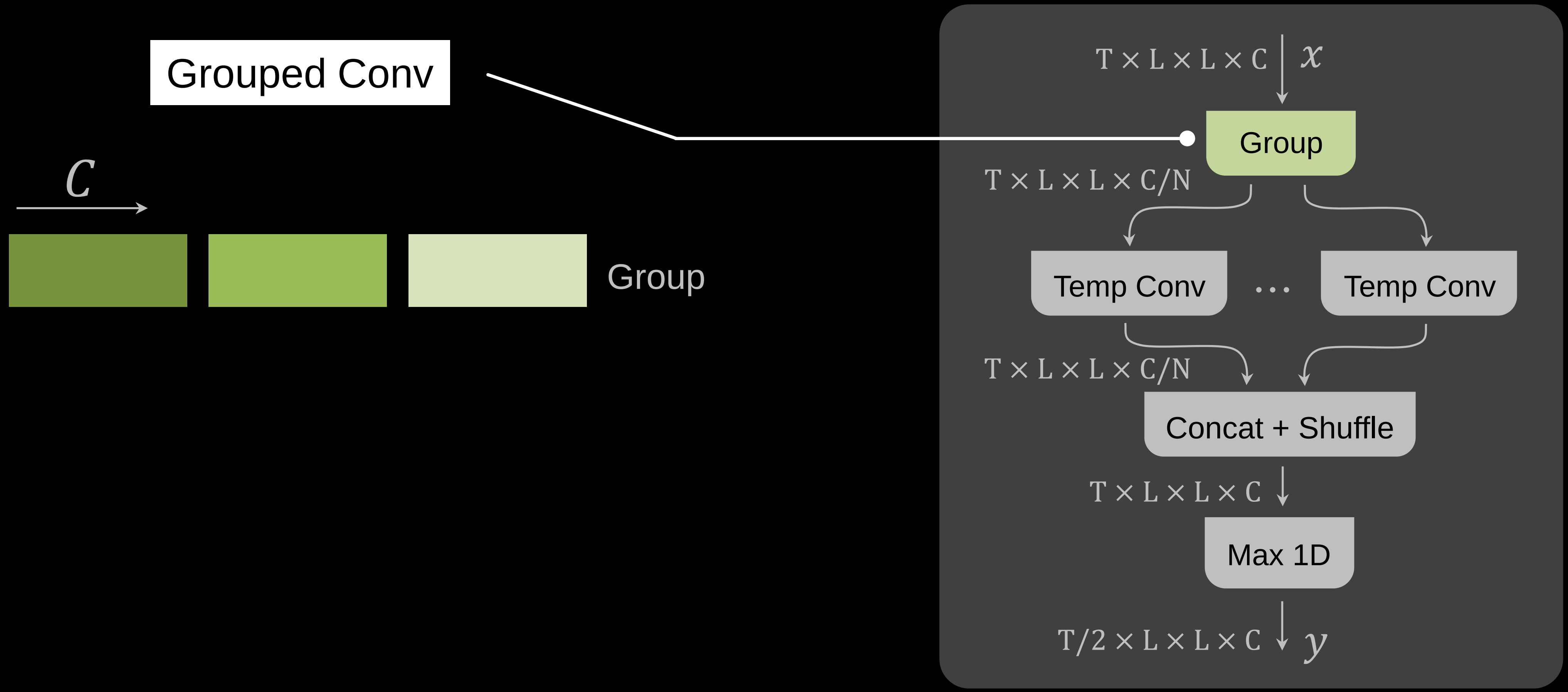


Model Overview



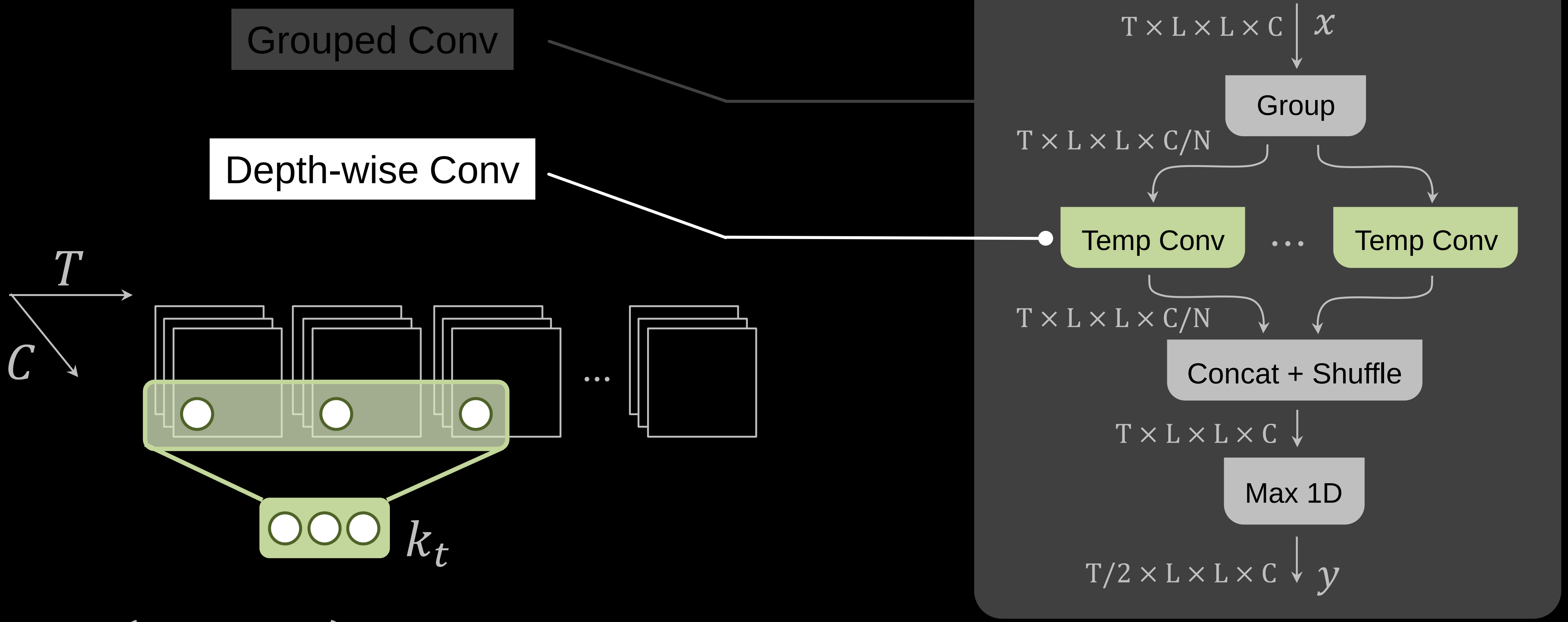
Timeception Layer

Method **Efficiency**



Timeception Layer

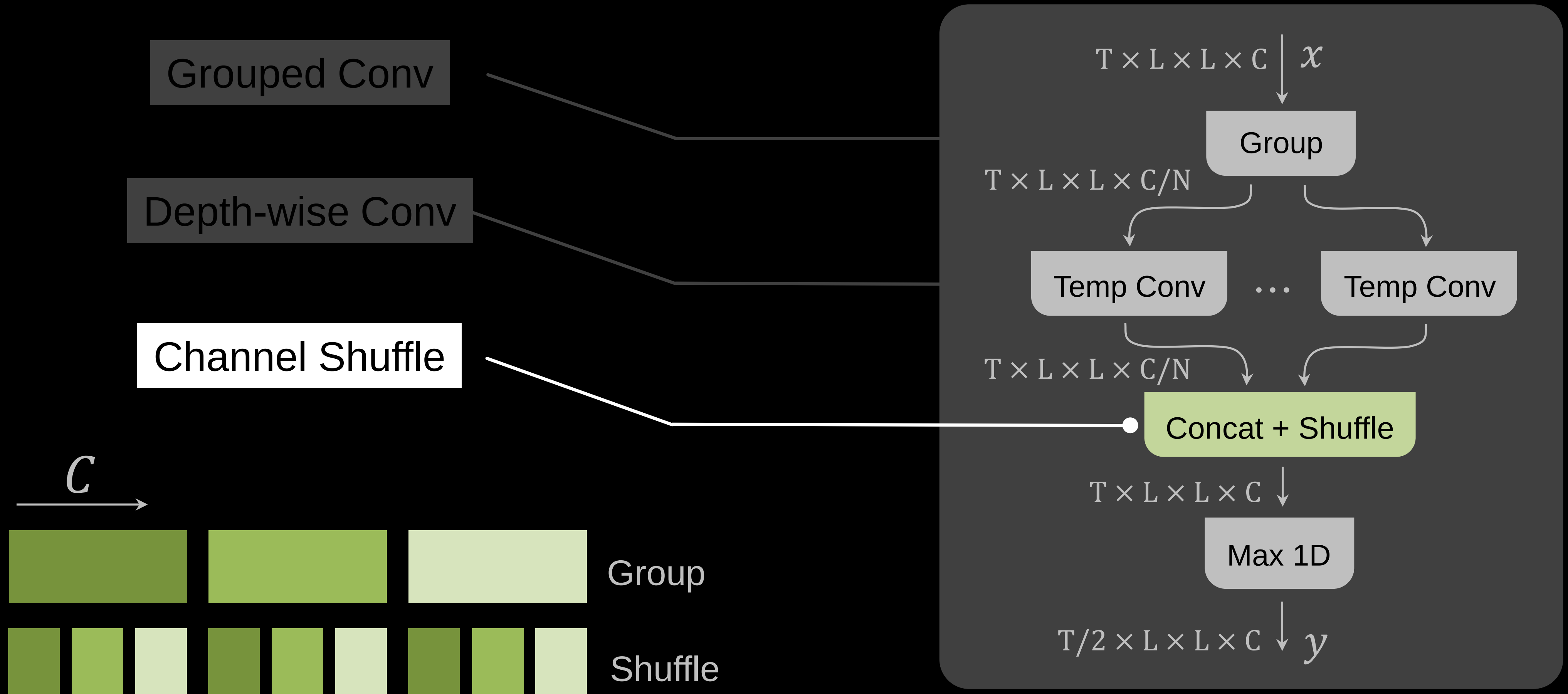
Method Efficiency



$$O(t \times c \times c) \rightarrow O(t \times c)$$

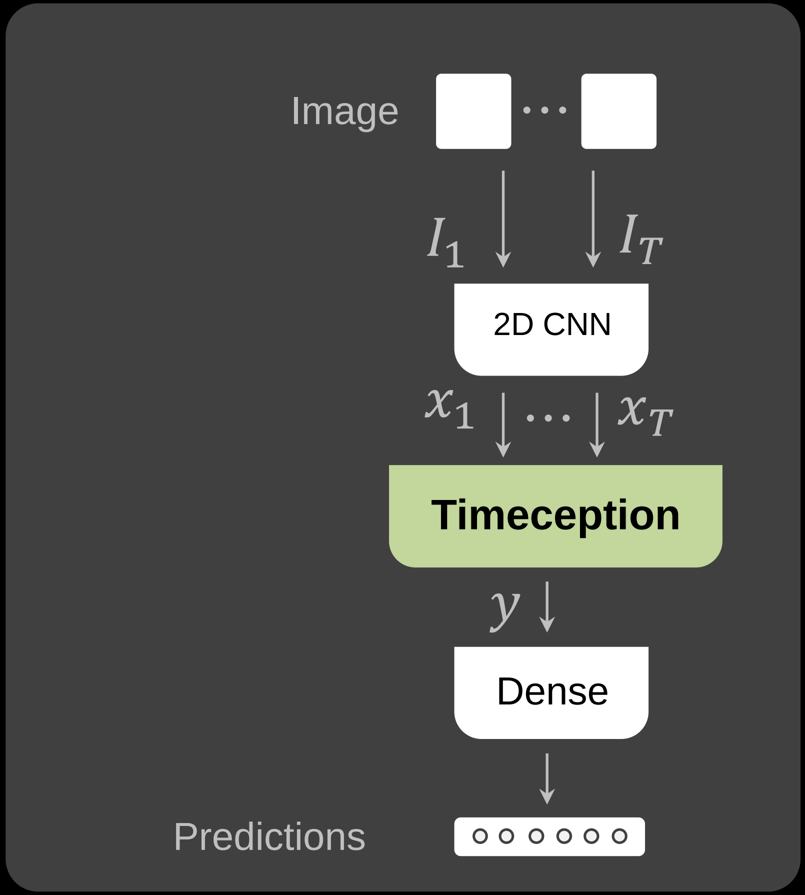
Timeception Layer

Method Efficiency

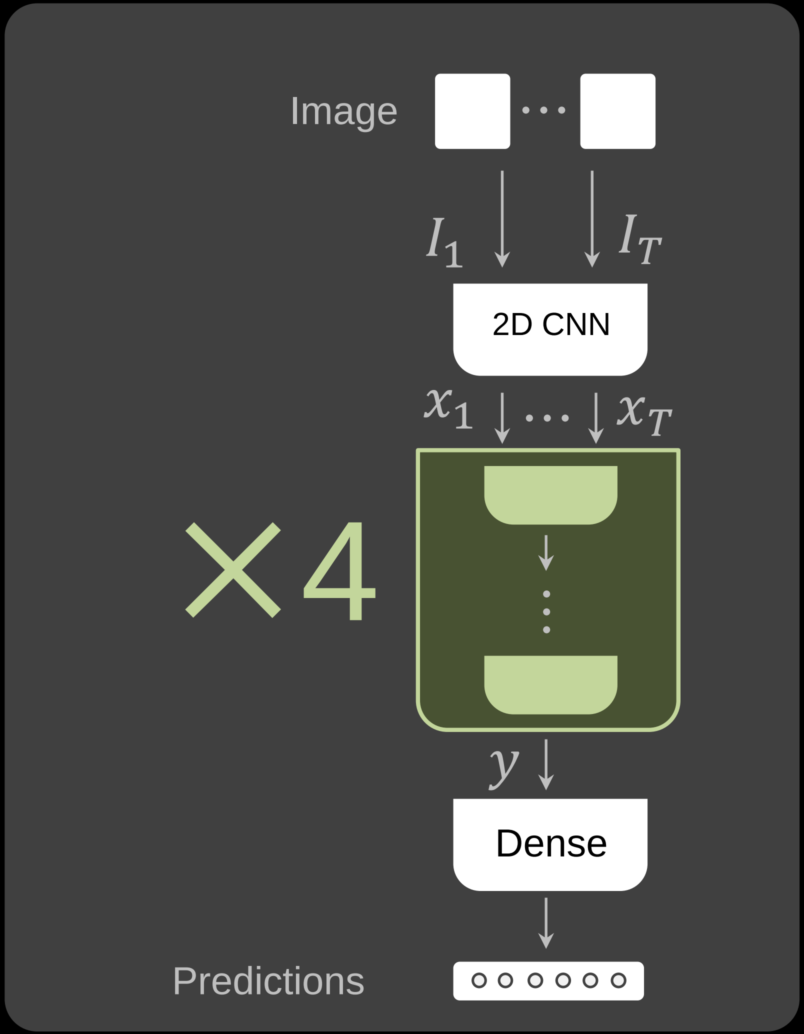


Timeception Layer

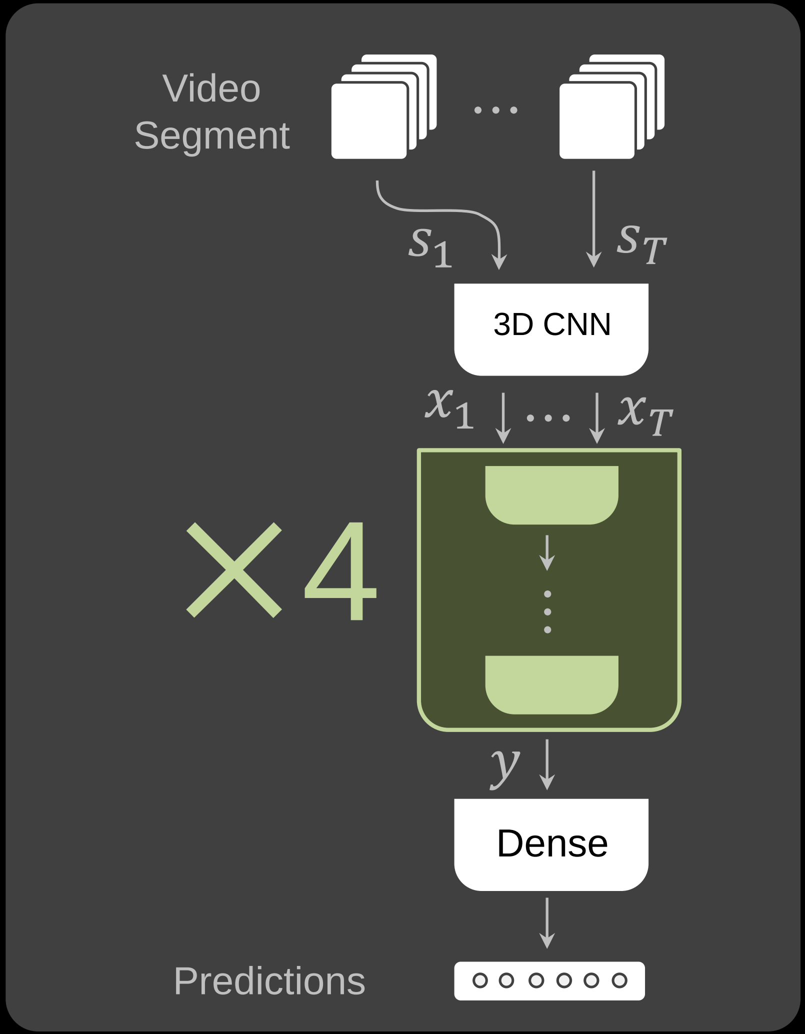
Method Timeception



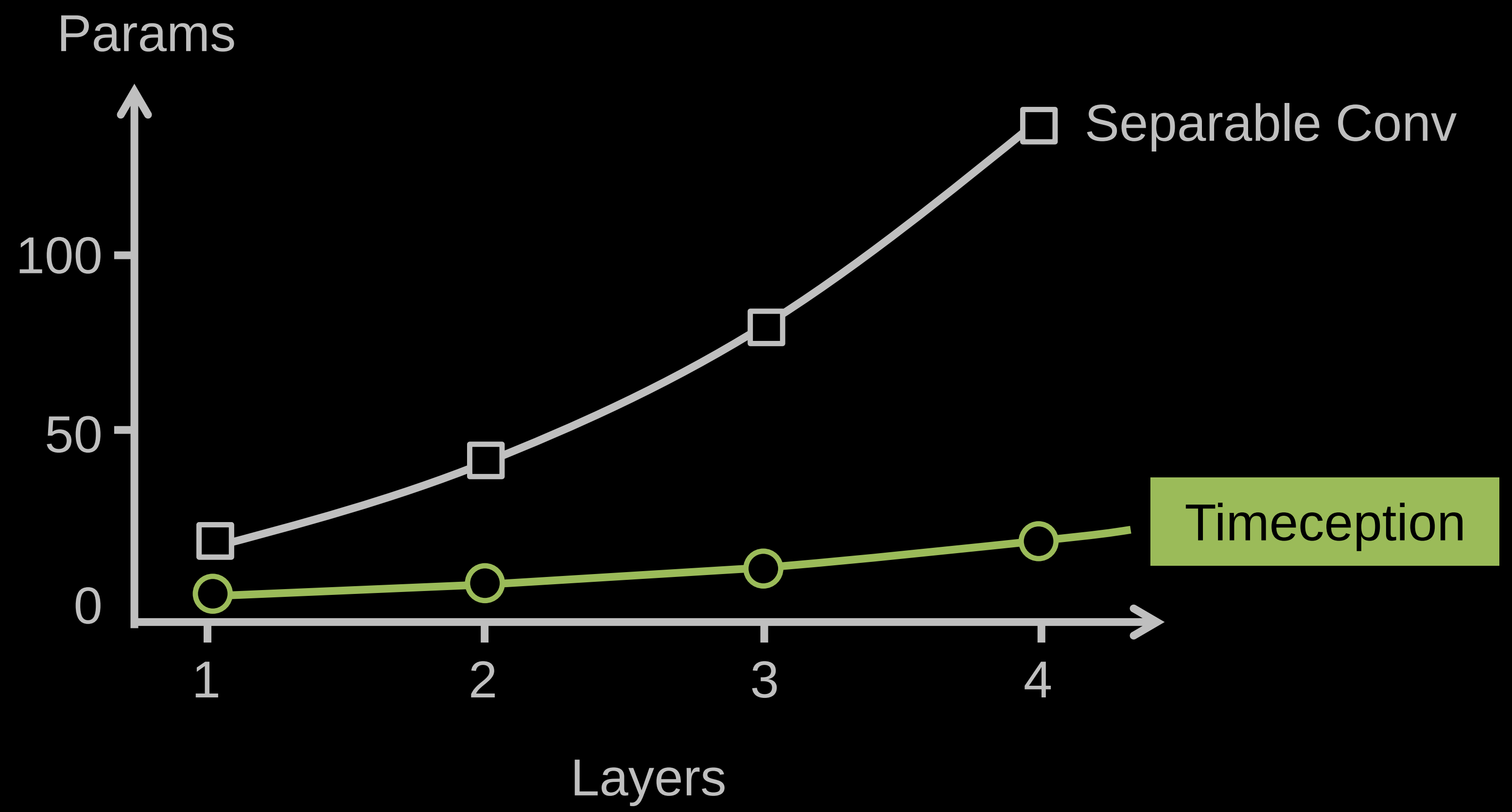
Method Timeception



Method Timeception

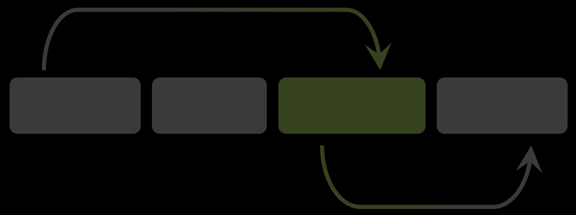


Method **Efficiency**



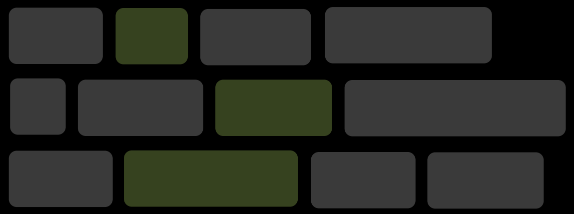
Method Timeception

1. Dependency

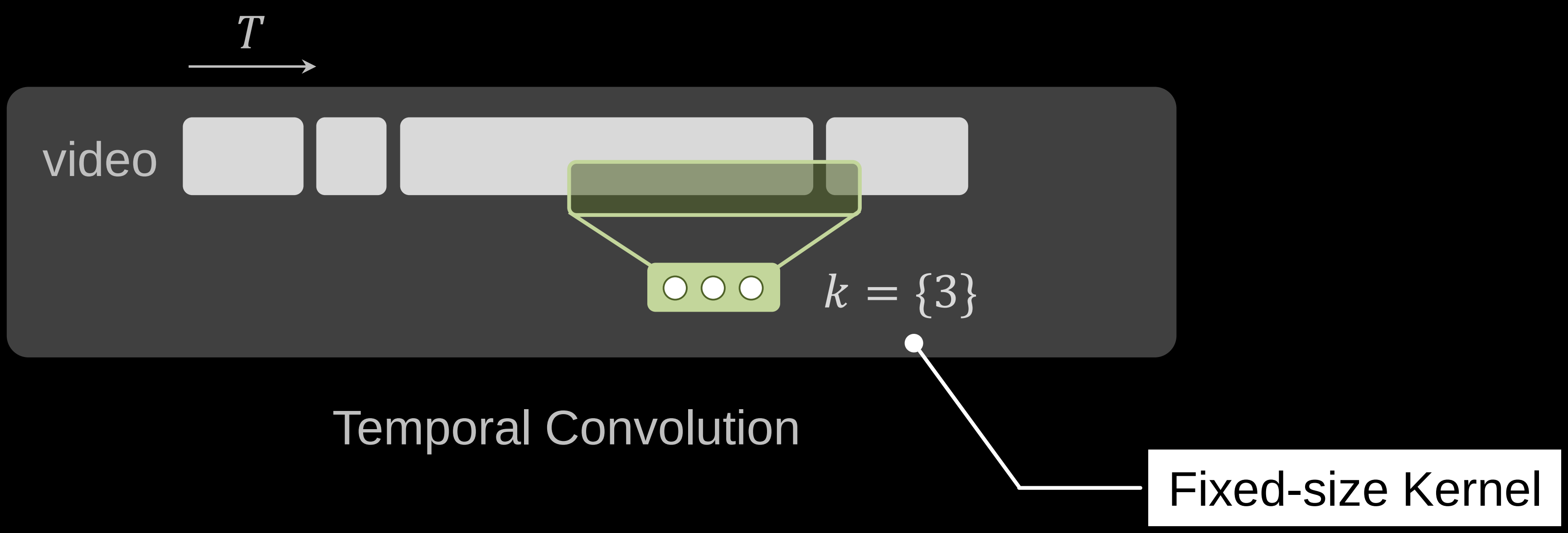


2. Long-range

3. Temporal Extent

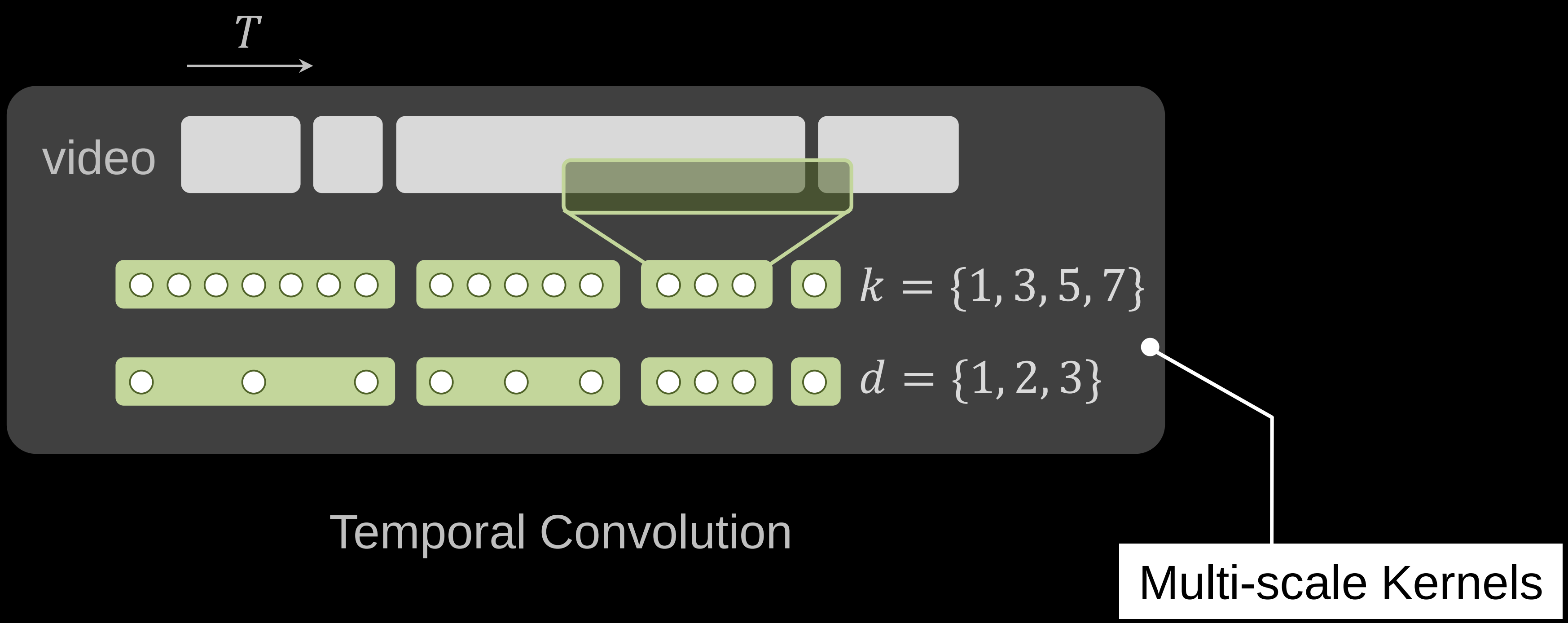


Method Tolerating Temporal Extents



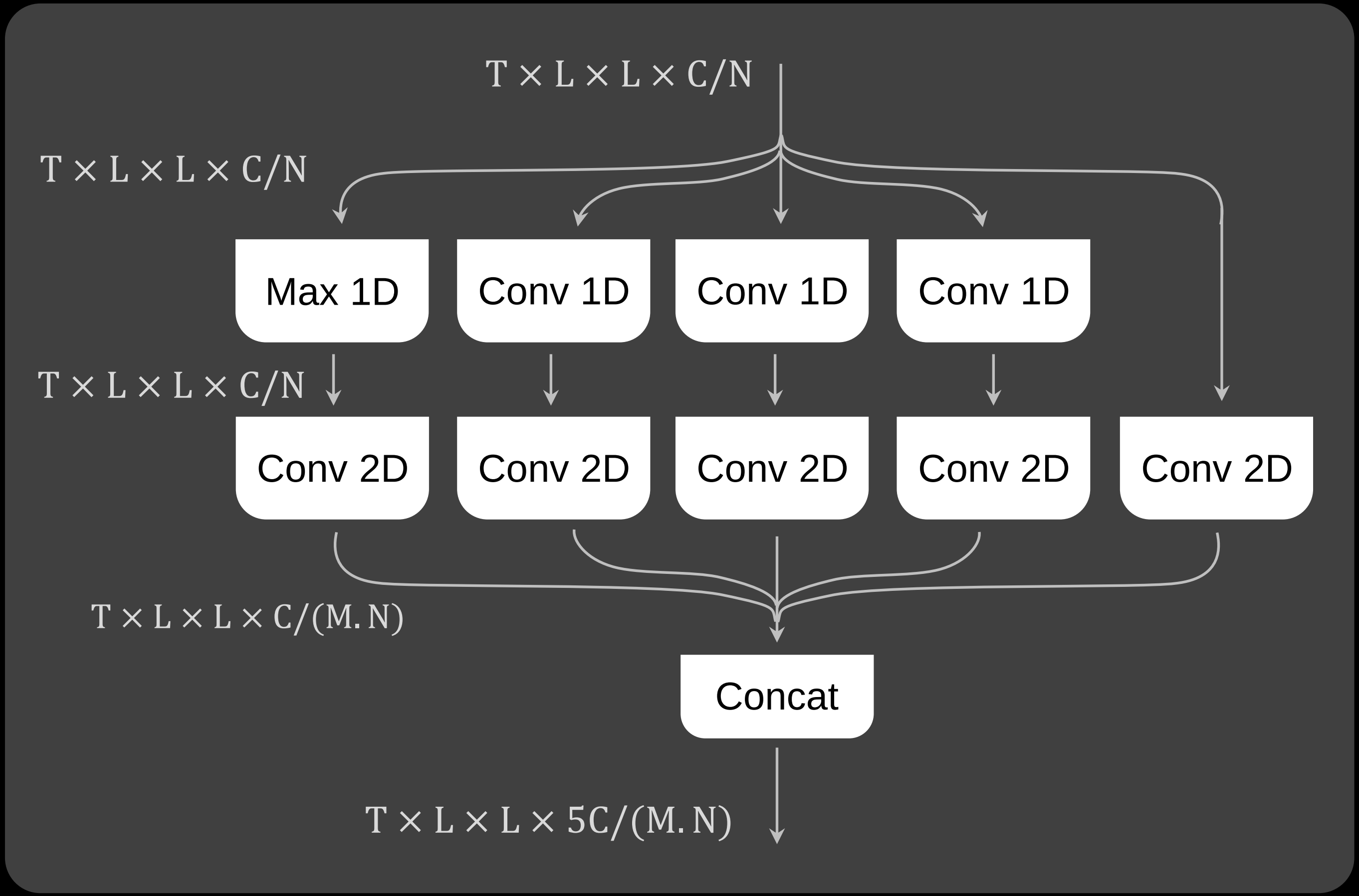
Temporal Convolution

Method Tolerating Temporal Extents



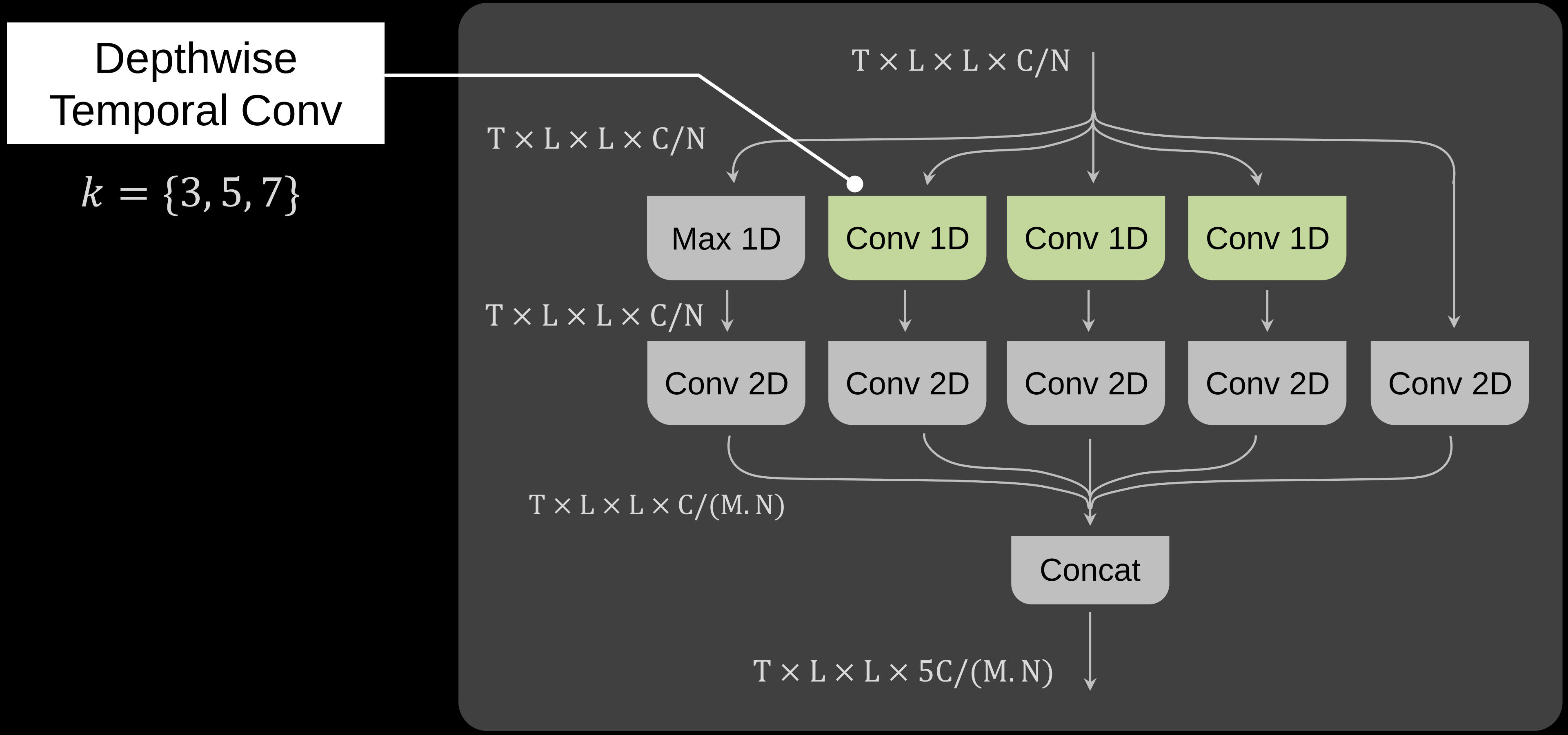
Temporal Convolution

Method Tolerating Temporal Extents



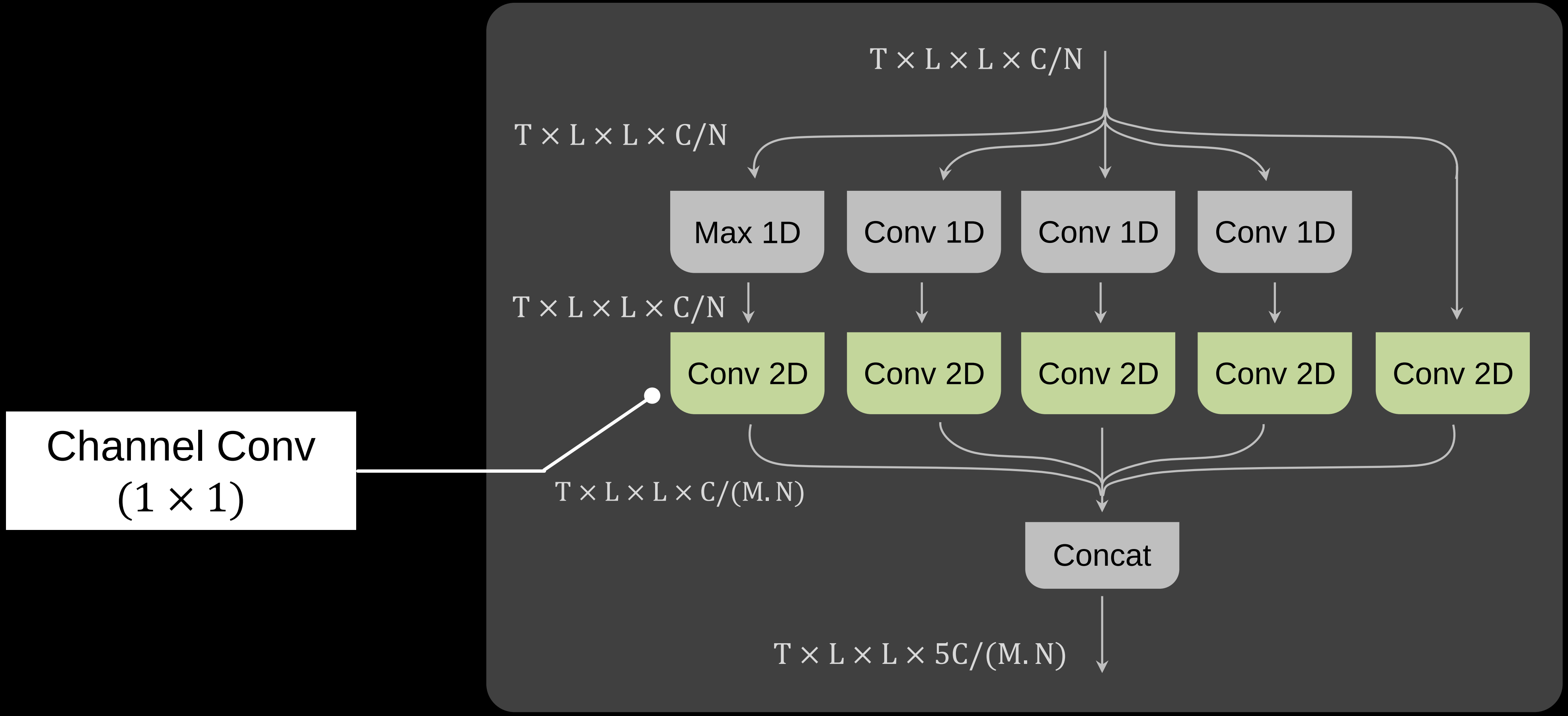
Temporal Conv Module

Method Tolerating Temporal Extents



Temporal Conv Module

Method Tolerating Temporal Extents



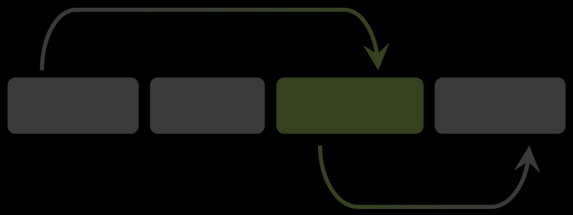
Temporal Conv Module

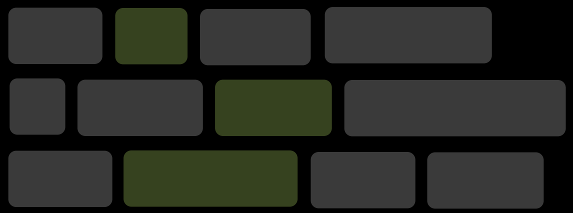
Method Timeception

1. Dependency

2. Long-range

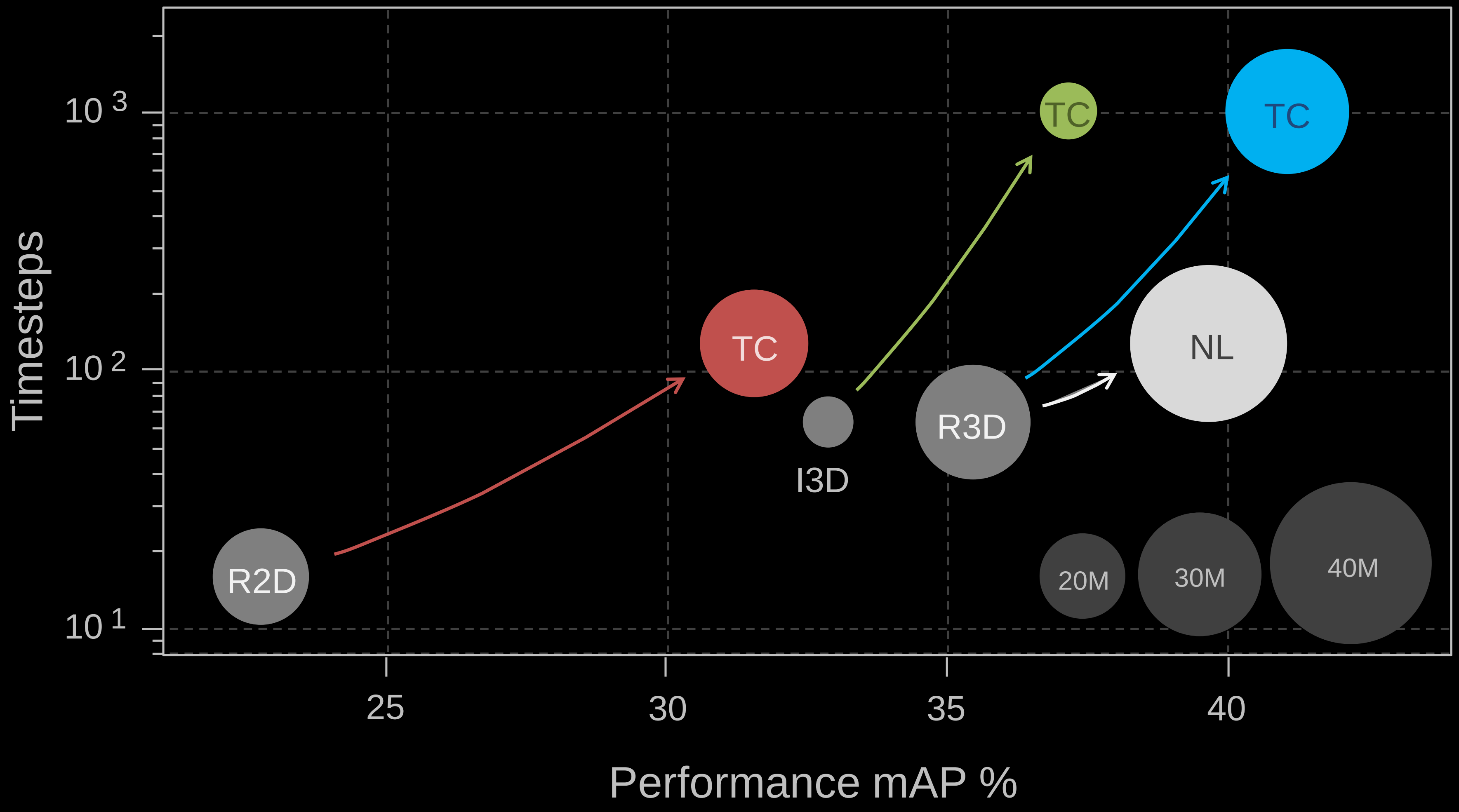
3. Temporal Extent



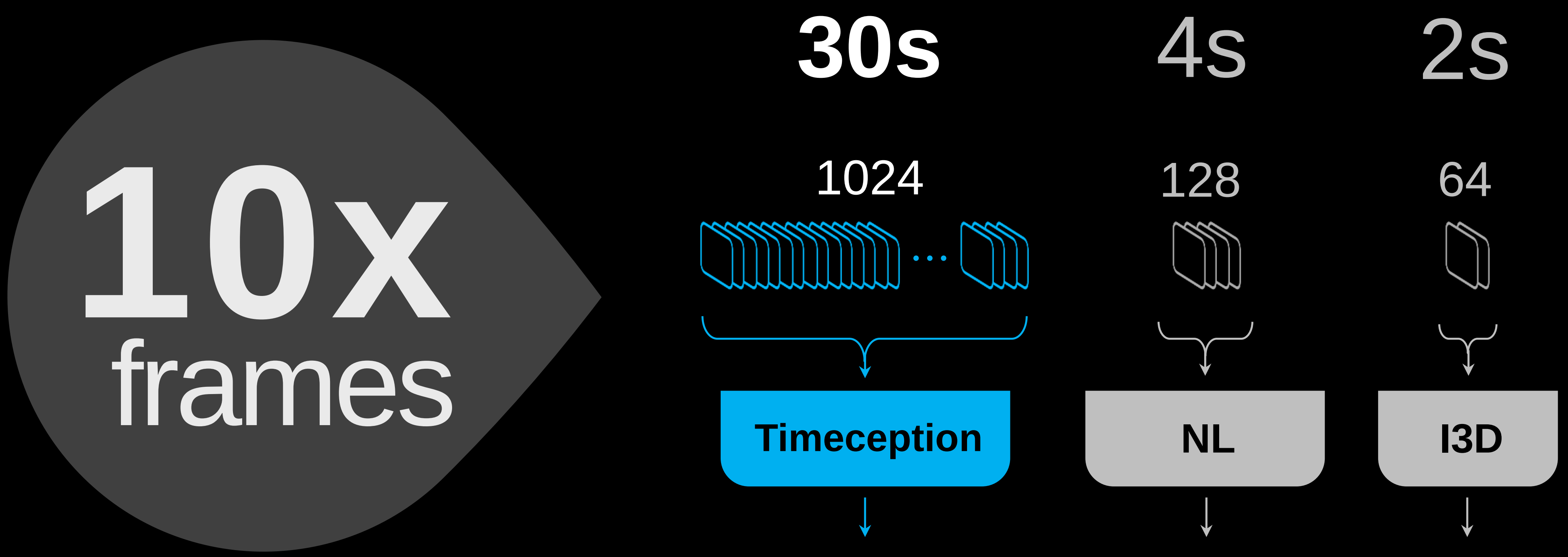


RESULTS

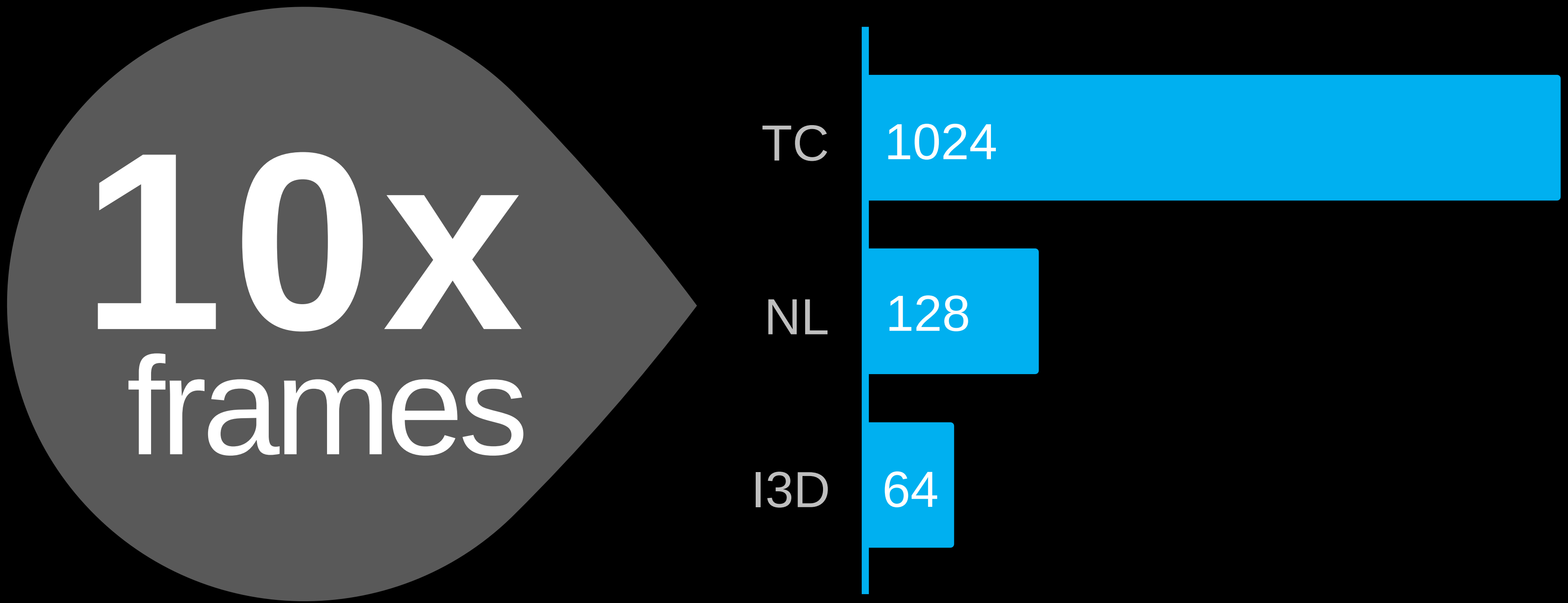
Results Charades Dataset



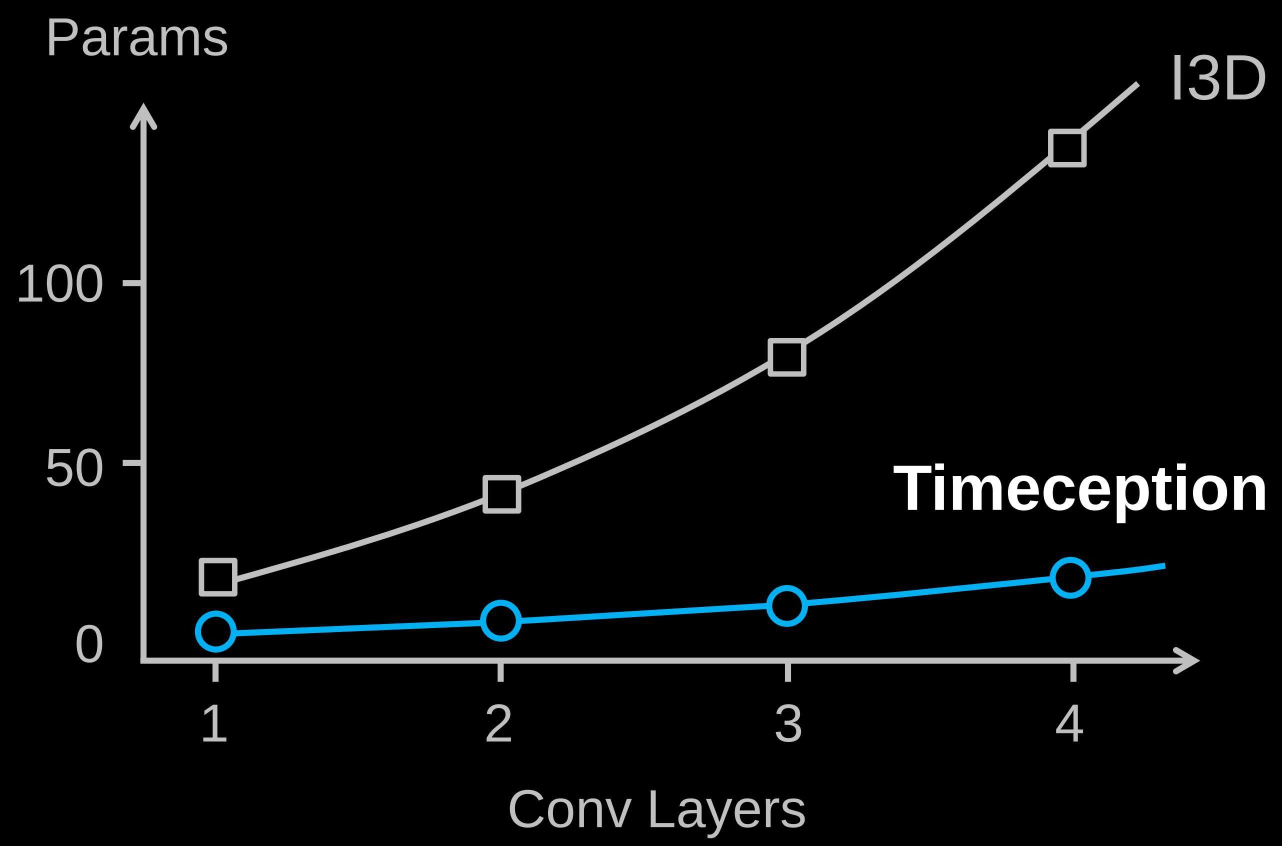
Results Temporal Footprint



Results Temporal Footprint

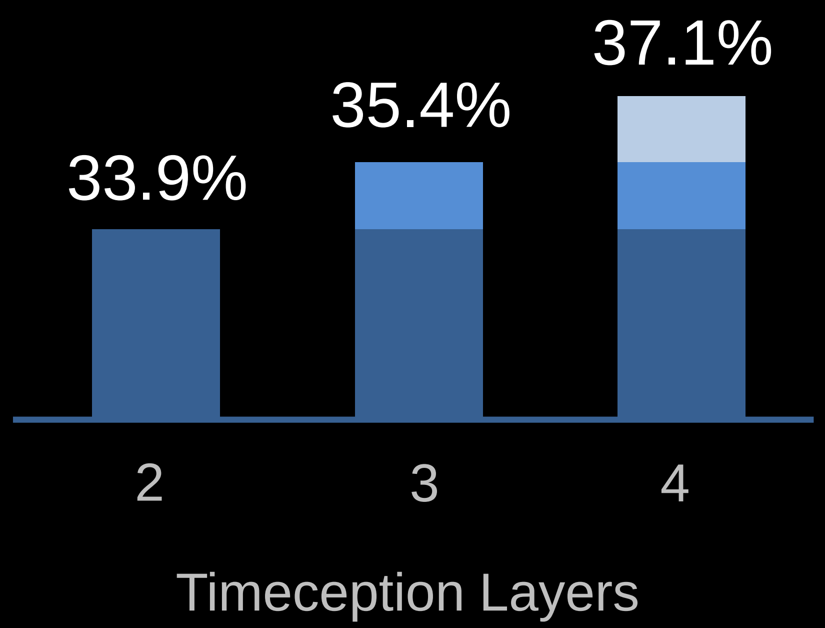


Results Layer Efficiency

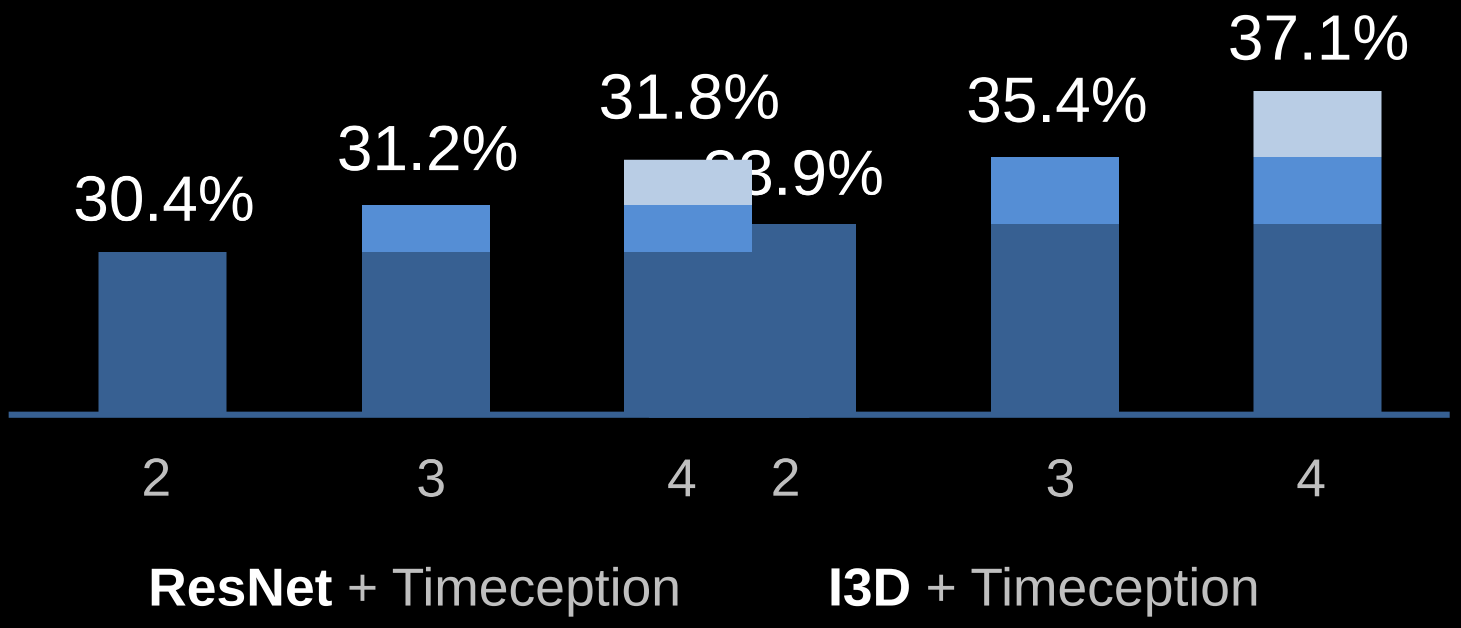


2.8m params

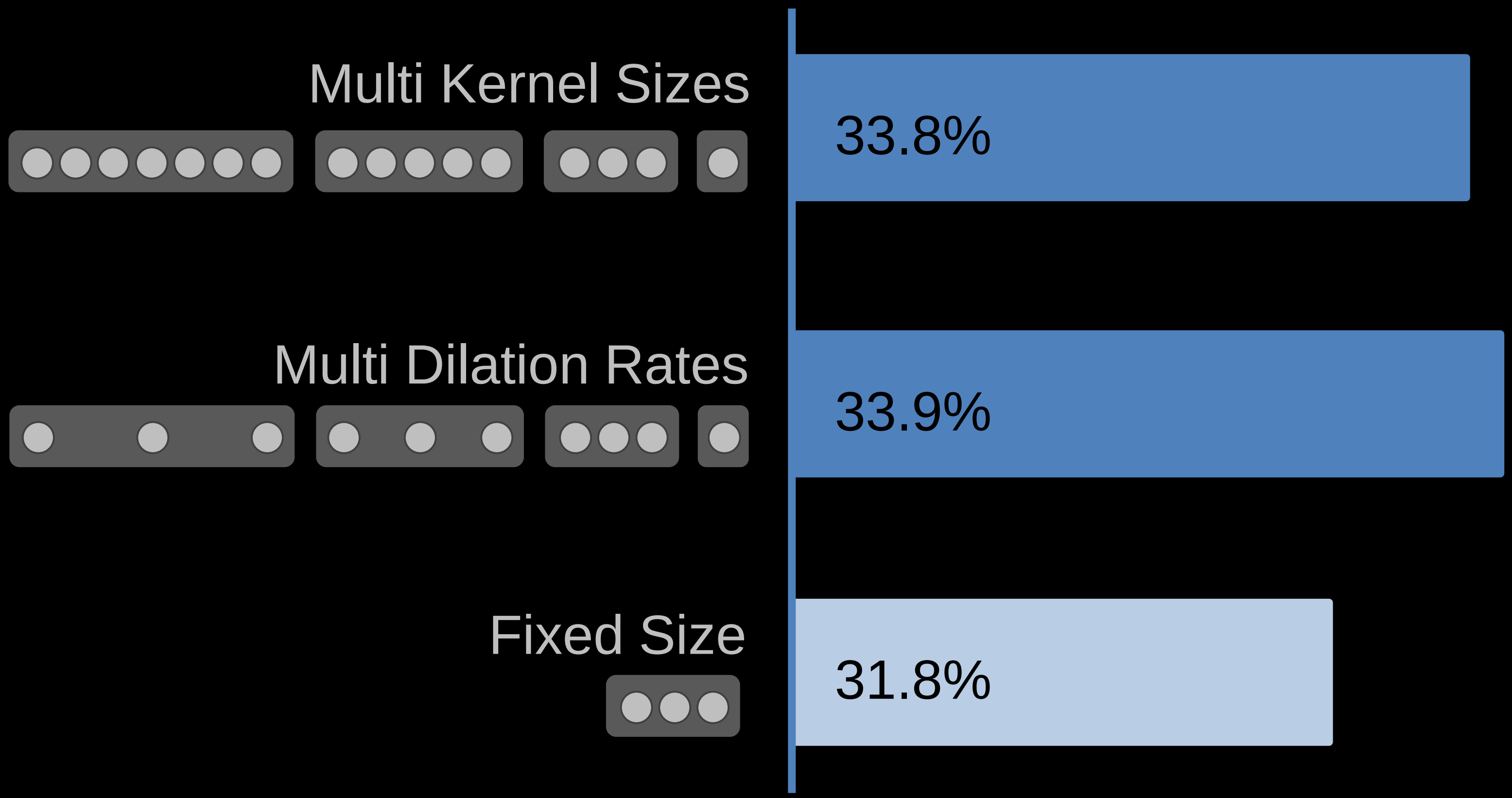
Results Layer Effectiveness



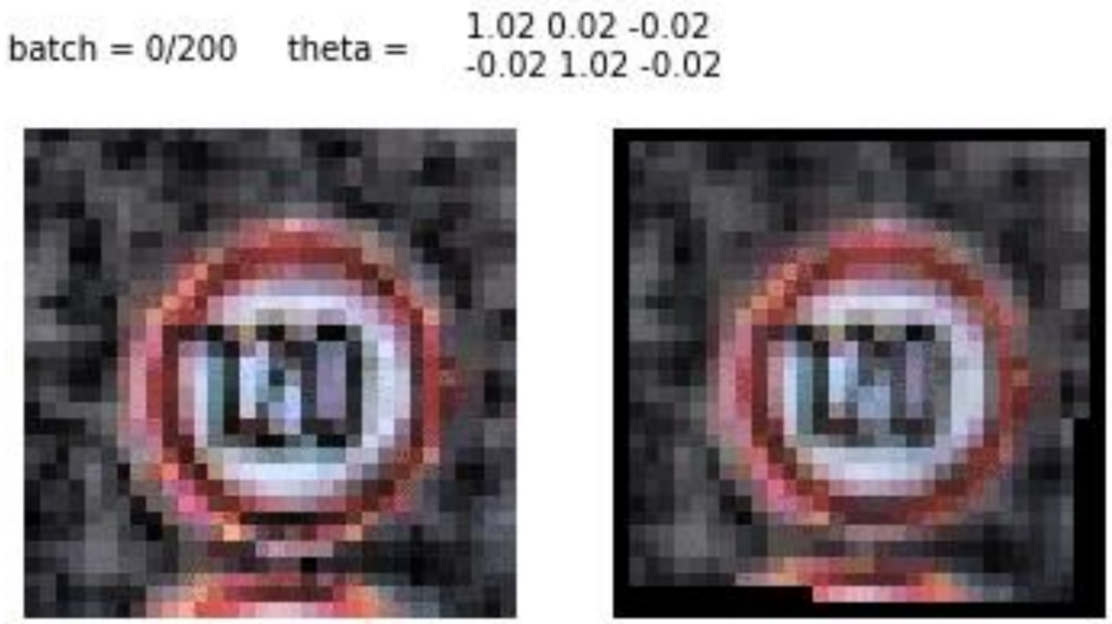
Results Layer Effectiveness



Results Multi-Scale Kernels

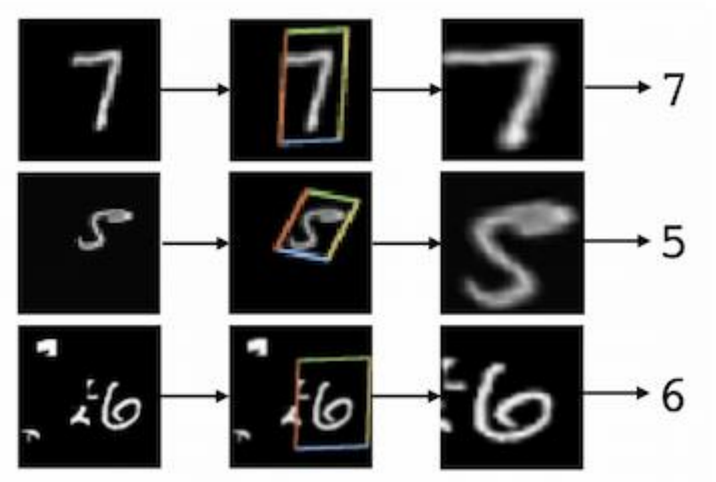


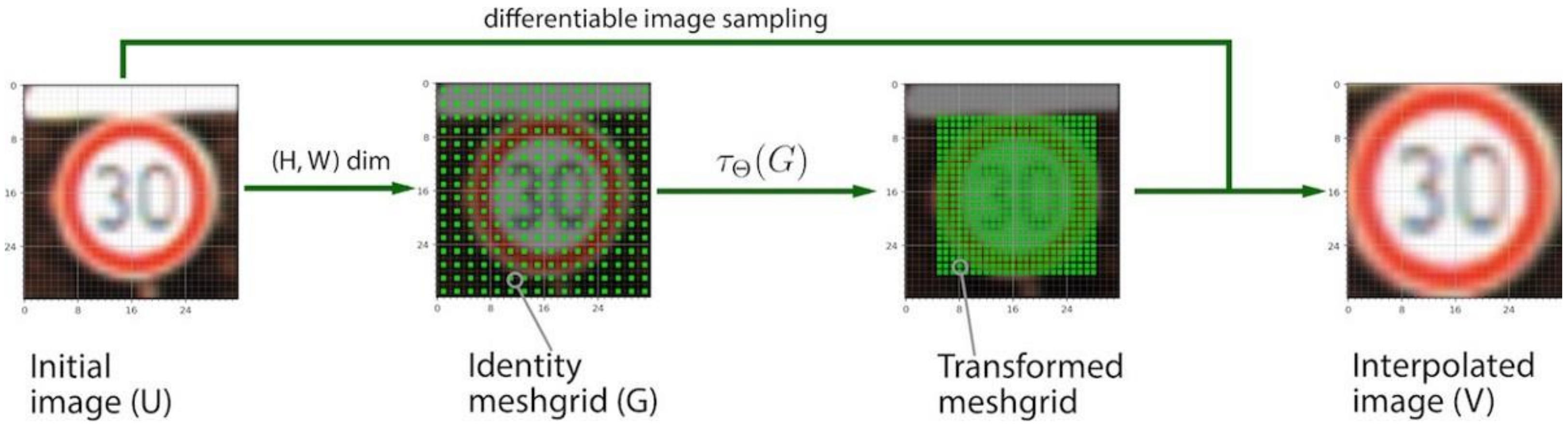
Spatial Transformer Network



Problem

- ConvNets sometimes are robust enough to input changes
  - While pooling gives some invariance, only in deeper layers the pooling receptive field is large enough for this invariance to be noteworthy
  - One way to improve robustness: Data augmentation
- Smarter way: Spatial Transformer Networks





Basic idea

- Define a geometric transformation matrix

$$\Theta = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix}$$

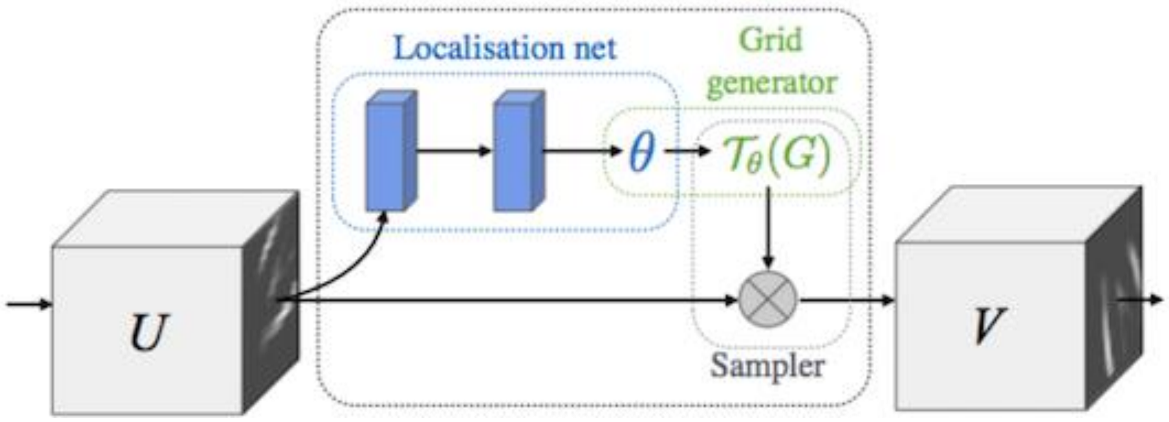
- Four interesting transformations
  - Identity, i.e. $\Theta = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$
  - Rotation, e.g., $\Theta \approx \begin{bmatrix} 0.7 & -0.7 & 0 \\ 0.7 & 0.7 & 0 \end{bmatrix}$ for 45^o, as $\cos(\frac{\pi}{4}) \approx 0.7$
  - Zooming in, e.g. $\Theta \approx \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \end{bmatrix}$ for 2X zooming in
  - Zooming in, e.g. $\Theta \approx \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix}$ for 2X zooming out

Basic idea

- Then, define a mesh grid (x_i^t, y_i^t) on the original image and apply the geometric transformations

$$\begin{bmatrix} x_i^s \\ y_i^s \end{bmatrix} = \Theta \cdot \begin{bmatrix} x_i^t \\ y_i^t \\ 1 \end{bmatrix}$$

- Produce the new image using the transformation above and an interpolation method
- Learn the parameters Θ and the meshgrid from the data
- A localization network learns to predict Θ given a new image



Summary

- Popular Convolutional Neural Networks architectures
- Go deeper on what makes them tick & what makes them different

Reading material

- All the papers from the models presented