

Lecture 12:

Optimizations for Processing Video

**Visual Computing Systems
Stanford CS348K, Fall 2018**

Video processing applications



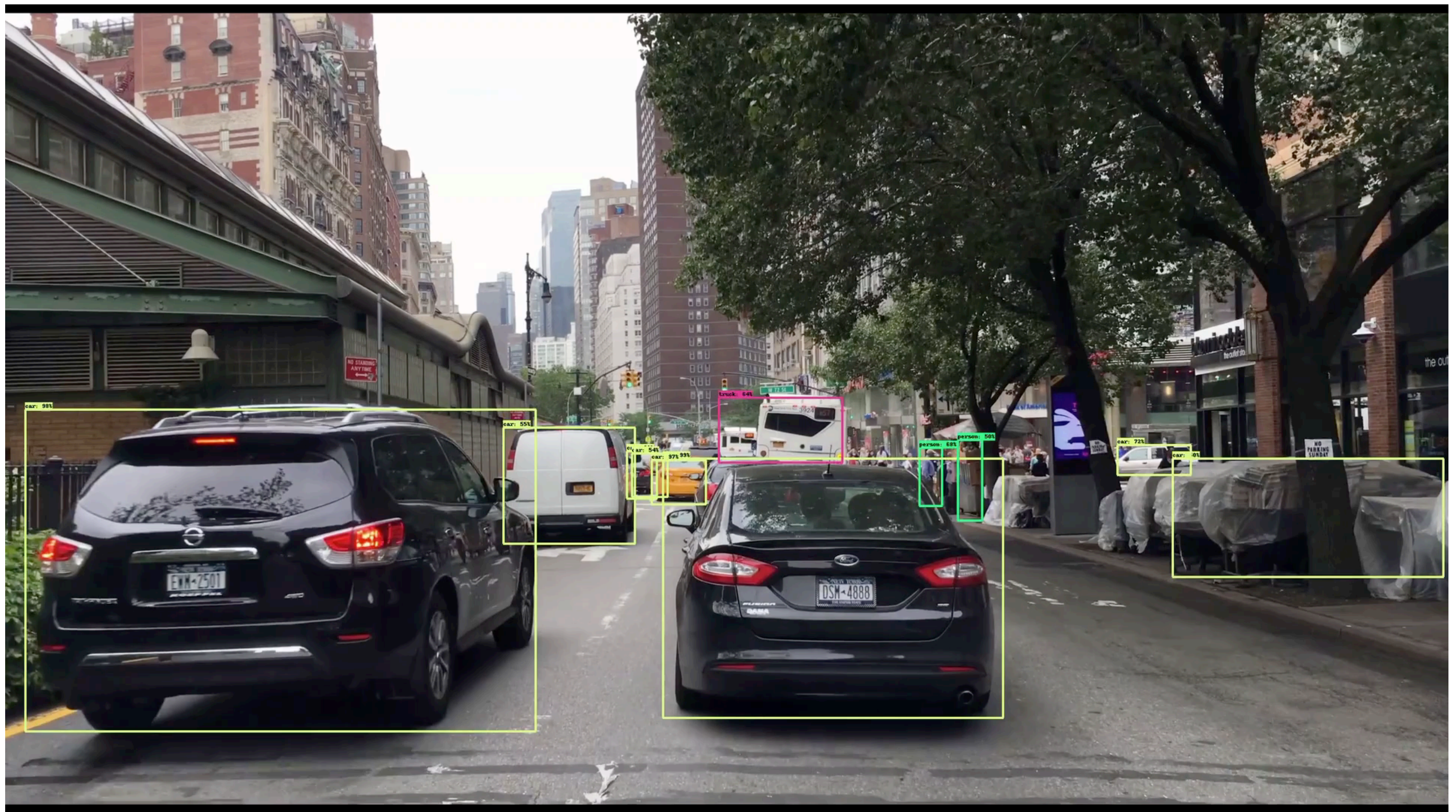
Thought experiment

Imagine we wanted to detect people/cars/bikes in a video stream



Thought experiment

Imagine we wanted to detect people/cars/bikes in a video stream



Object detection performance

600x600 input images (not particularly large)

COCO-trained models {#coco-models}

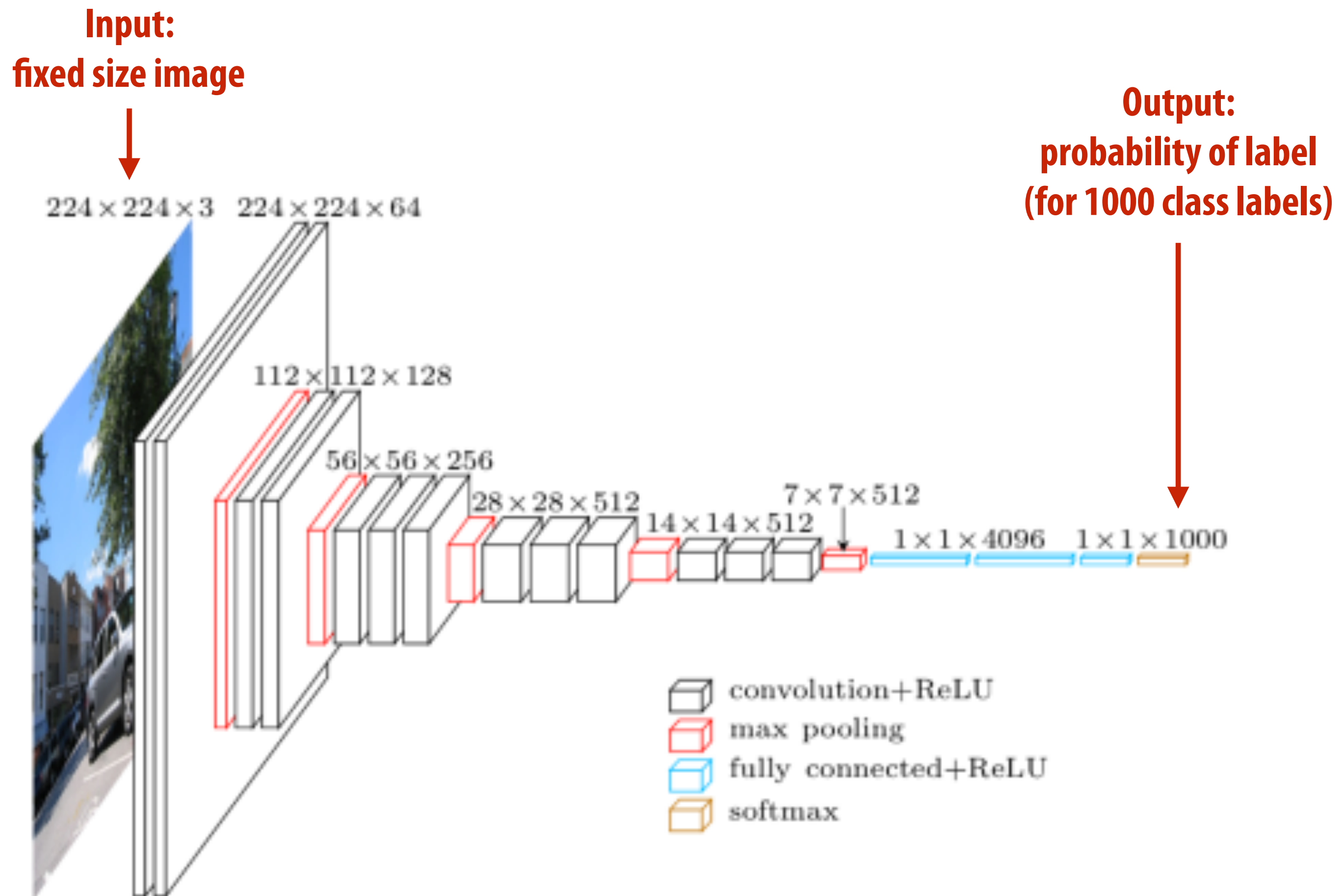
| Model name | Speed (ms) | COCO mAP[^1] | Outputs |
|--|------------|--------------|---------|
| ssd_mobilenet_v1_coco | 30 | 21 | Boxes |
| ssd_inception_v2_coco | 42 | 24 | Boxes |
| faster_rcnn_inception_v2_coco | 58 | 28 | Boxes |
| faster_rcnn_resnet50_coco | 89 | 30 | Boxes |
| faster_rcnn_resnet50_lowproposals_coco | 64 | | Boxes |
| rfcn_resnet101_coco | 92 | 30 | Boxes |
| faster_rcnn_resnet101_coco | 106 | 32 | Boxes |
| faster_rcnn_resnet101_lowproposals_coco | 82 | | Boxes |
| faster_rcnn_inception_resnet_v2_atrous_coco | 620 | 37 | Boxes |
| faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco | 241 | | Boxes |
| faster_rcnn_nas | 1833 | 43 | Boxes |
| faster_rcnn_nas_lowproposals_coco | 540 | | Boxes |

[Credit: Tensorflow detection model zoo]

Aside:
**optimizing object detection in a
single image**

VGG-16 image classification network

[Simonyan 2015]



Network assigns input image one of 1000 potential labels.

Using classification network as a “subroutine for object detection

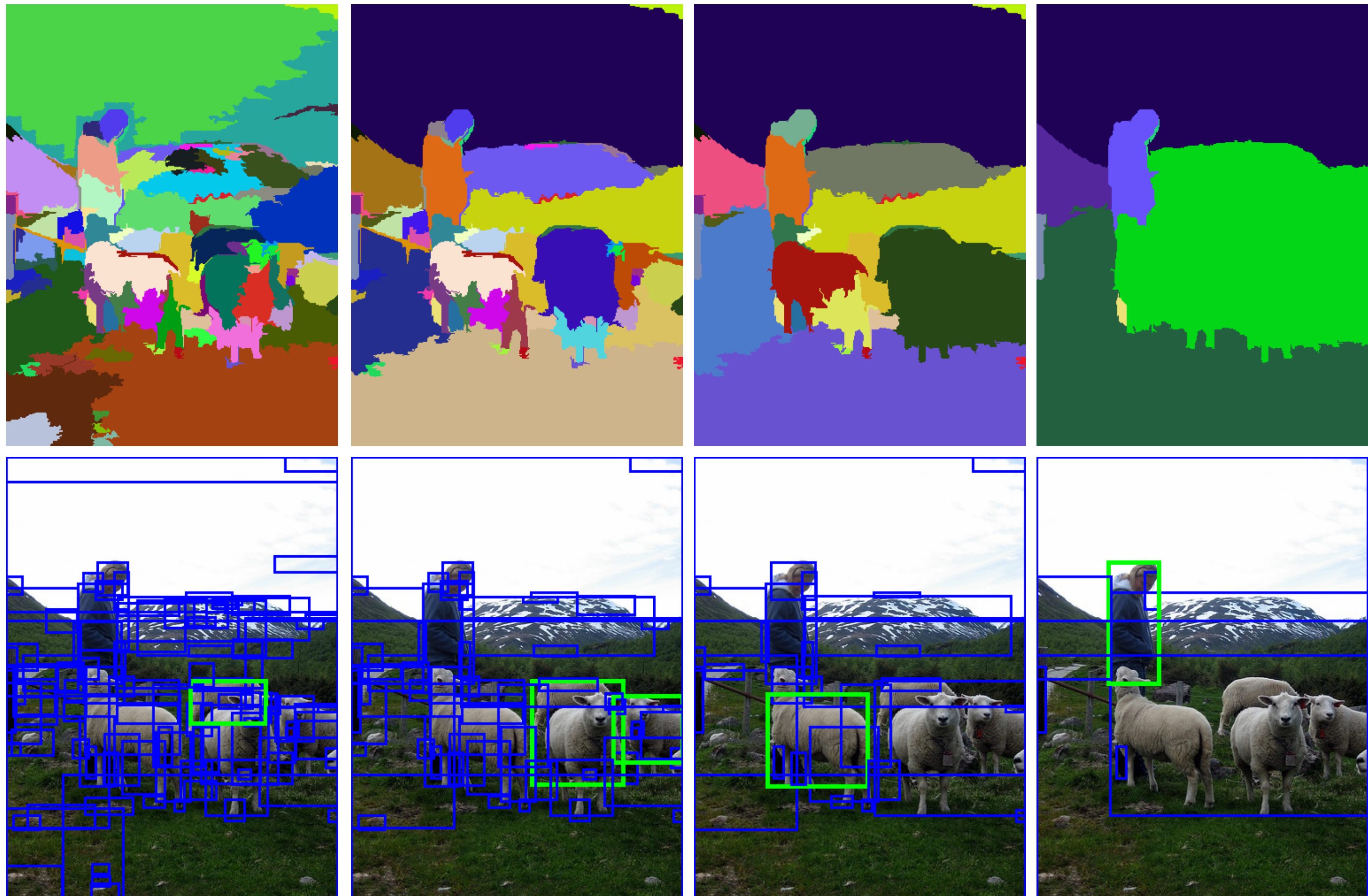
[Girshick 2014]

Search over all regions of the image and all region sizes for objects (“Sliding window” over image, repeated for multiple potential object scales)

```
for all region top-left positions (x,y):  
  for all region sizes (w,h):  
    cropped = image_crop(image, bbox(x,y,w,h))  
    resized = image_resize(227,227)  
    label = detect_object(resized)  
    if (label != background)  
      // region defined by bbox(x,y,w,h) contains object  
      // of class 'label'
```


Optimization 1: filter detection work via object proposals

Selective search [Uijlings IJCV 2013]



Input: image

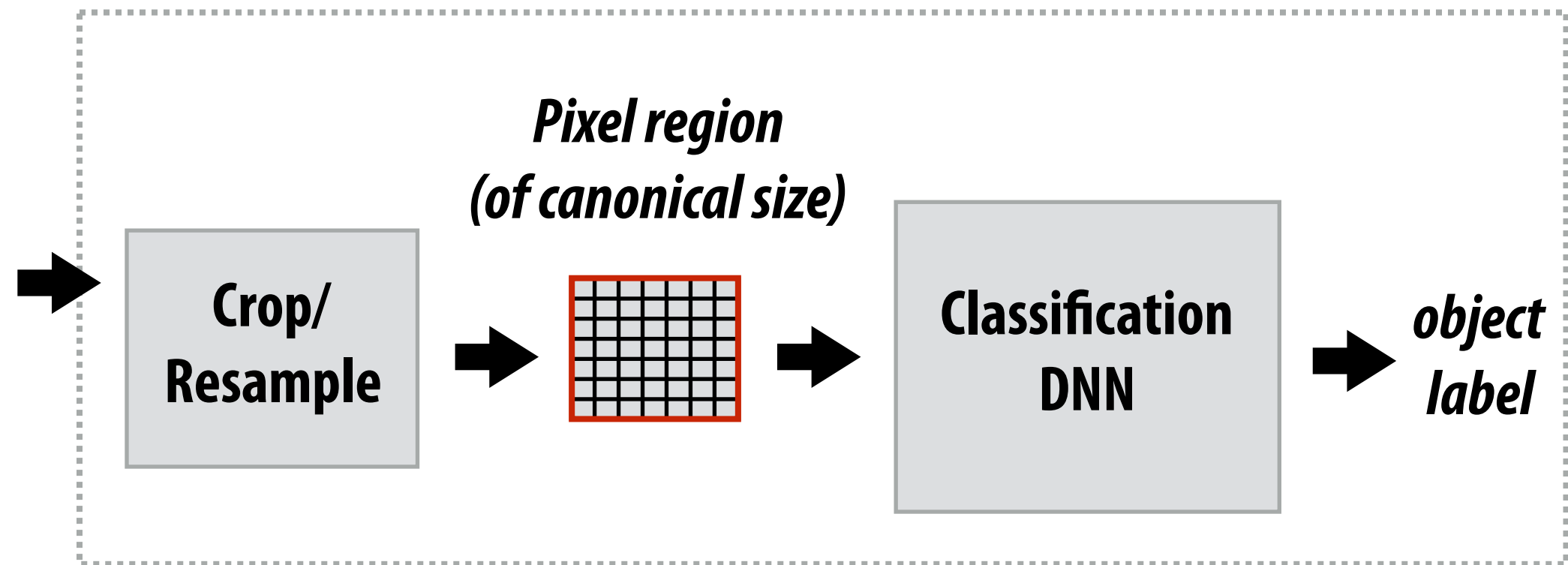
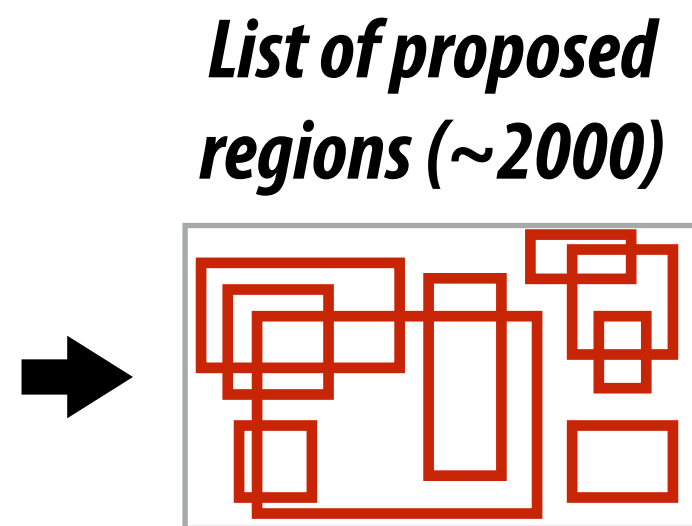
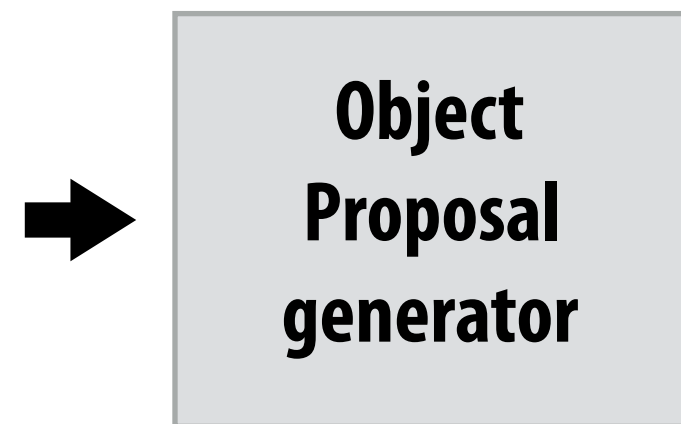
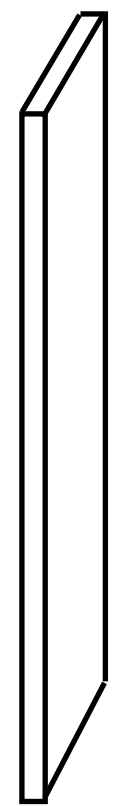
Output: list of regions (various scales) that are likely to contain objects

Idea: proposal algorithm filters parts of the image not likely to contain objects

Object detection pipeline executed only on proposed regions

[Girshick 2014]

Input image:
(of any size)



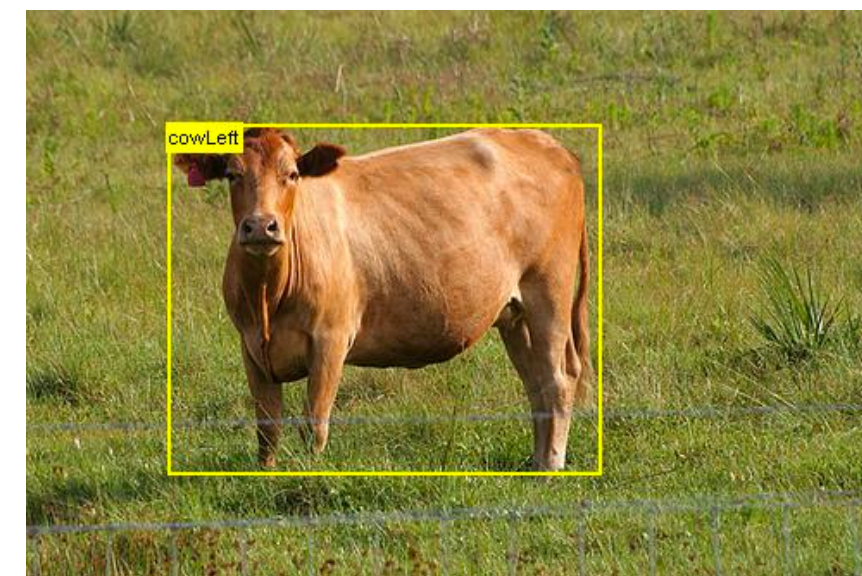
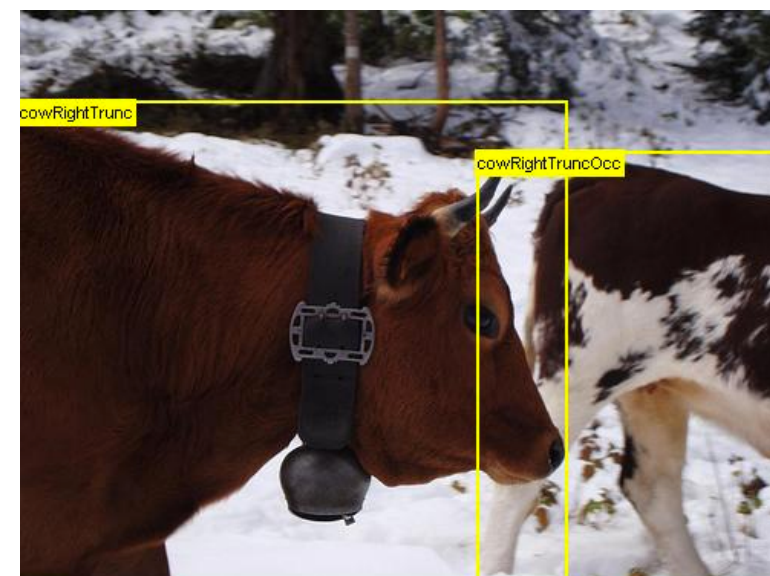
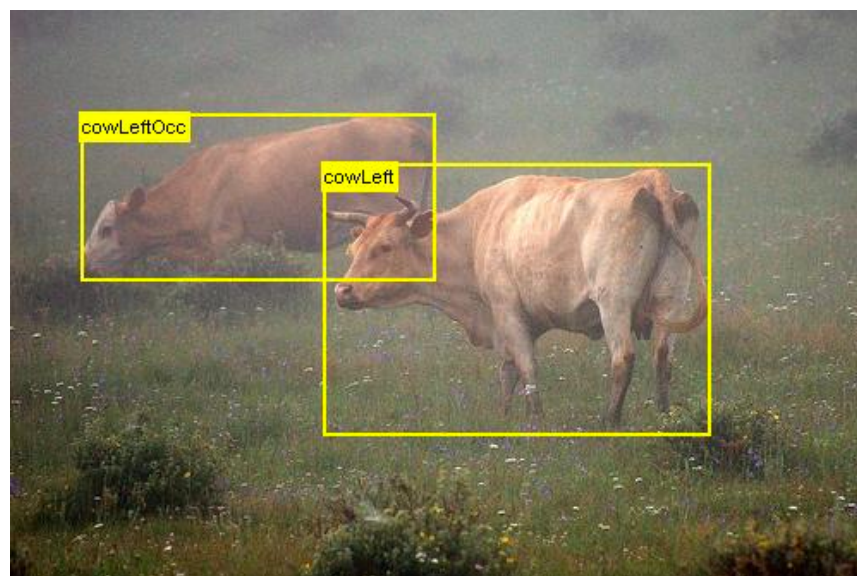
Object detection performance on Pascal VOC

Example training data

airplanes



cow



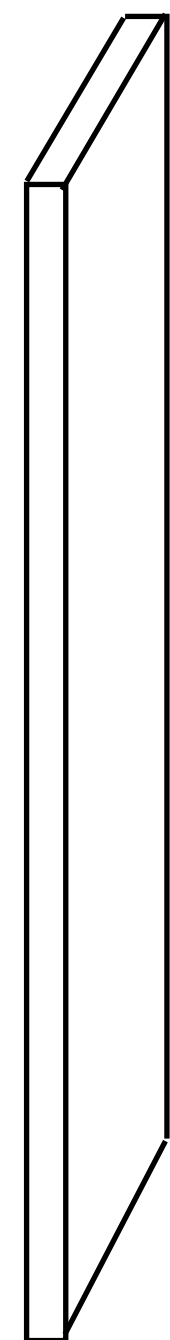
| VOC 2010 test | aero | bike | bird | boat | bottle | bus | car | cat | chair | cow | table | dog | horse | mbike | person | plant | sheep | sofa | train | tv | mAP |
|-----------------|------|------|------|------|--------|------|------|------|-------|------|-------|------|-------|-------|--------|-------|-------|------|-------|------|------|
| DPM v5 [18]† | 49.2 | 53.8 | 13.1 | 15.3 | 35.5 | 53.4 | 49.7 | 27.0 | 17.2 | 28.8 | 14.7 | 17.8 | 46.4 | 51.2 | 47.7 | 10.8 | 34.2 | 20.7 | 43.8 | 38.3 | 33.4 |
| UVA [34] | 56.2 | 42.4 | 15.3 | 12.6 | 21.8 | 49.3 | 36.8 | 46.1 | 12.9 | 32.1 | 30.0 | 36.5 | 43.5 | 52.9 | 32.9 | 15.3 | 41.1 | 31.8 | 47.0 | 44.8 | 35.1 |
| Regionlets [36] | 65.0 | 48.9 | 25.9 | 24.6 | 24.5 | 56.1 | 54.5 | 51.2 | 17.0 | 28.9 | 30.2 | 35.8 | 40.2 | 55.7 | 43.5 | 14.3 | 43.9 | 32.6 | 54.0 | 45.9 | 39.7 |
| SegDPM [16]† | 61.4 | 53.4 | 25.6 | 25.2 | 35.5 | 51.7 | 50.6 | 50.8 | 19.3 | 33.8 | 26.8 | 40.4 | 48.3 | 54.4 | 47.1 | 14.8 | 38.7 | 35.0 | 52.8 | 43.1 | 40.4 |
| R-CNN | 67.1 | 64.1 | 46.7 | 32.0 | 30.5 | 56.4 | 57.2 | 65.9 | 27.0 | 47.3 | 40.9 | 66.6 | 57.8 | 65.9 | 53.6 | 26.7 | 56.5 | 38.1 | 52.8 | 50.2 | 50.2 |

DNN weights “pre-trained” using object classification on ImageNet (lots of data, different task)

DNN weights “fine-tuned” for the 20 VOC categories + 1 “background” category (task-specific data)

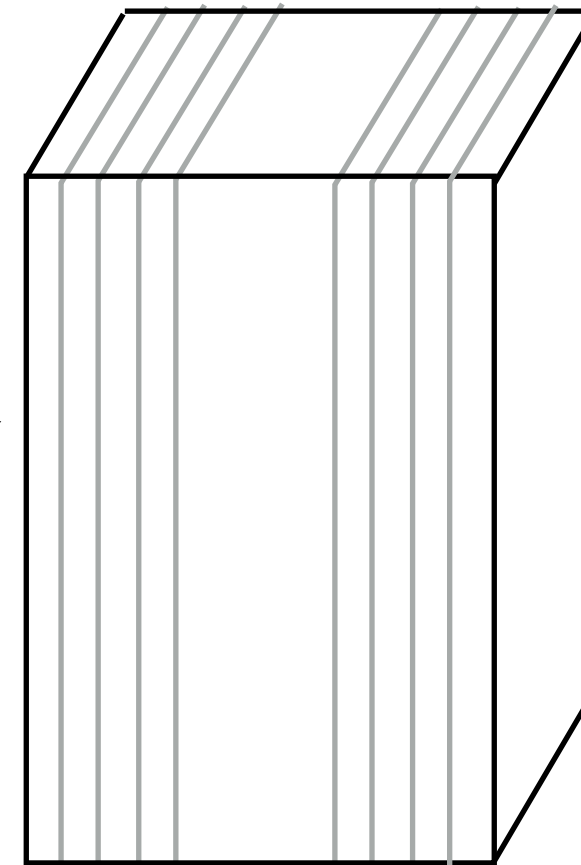
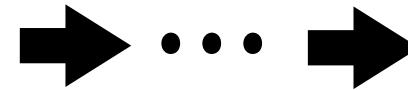
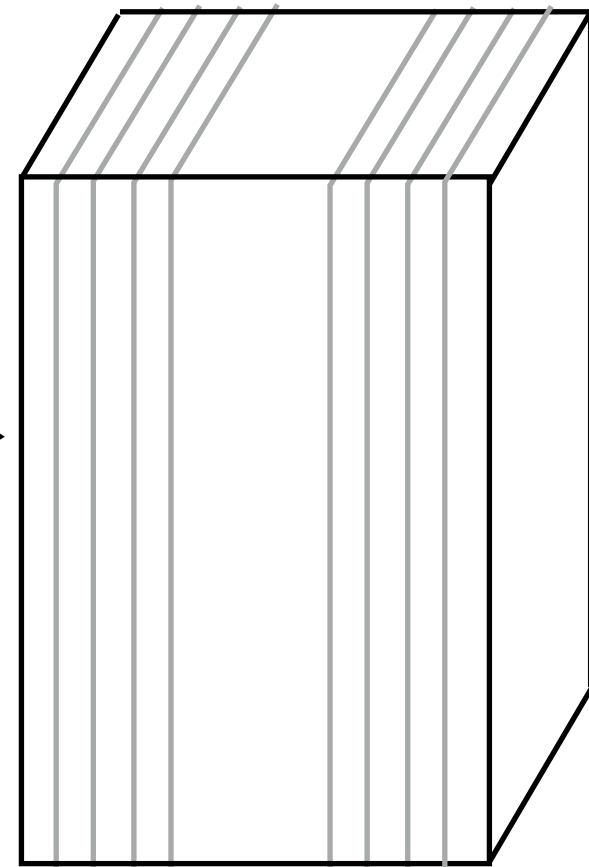
Optimization 2: region of interest pooling

RGB input image:
(of any size)

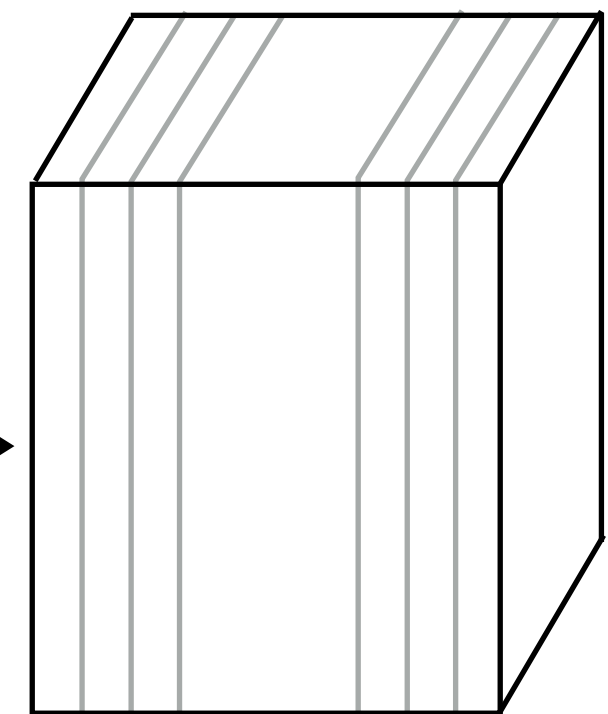


convlayer/
maxpool

“Fully convolutional network”:
sequence of convolutions and pooling steps: output size is dependent on input size



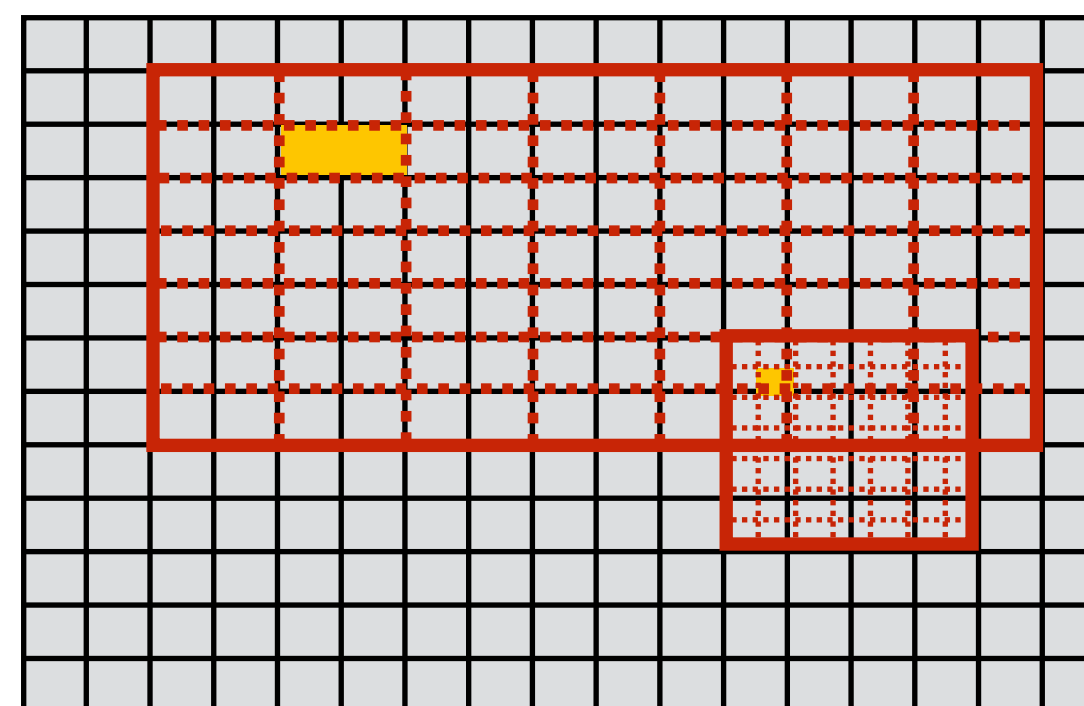
ROI pool



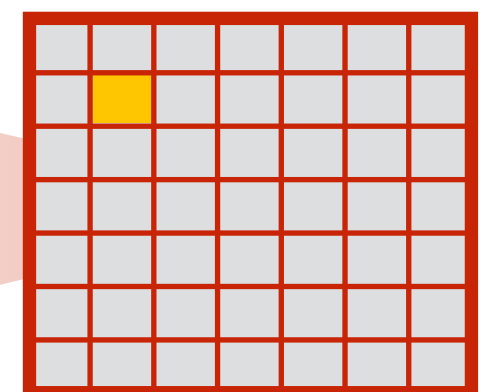
7x7xC

Idea: the output of early convolutional layers of network on downsampled input region is approximated by resampling output of fully-convolutional implementation of conv layers.

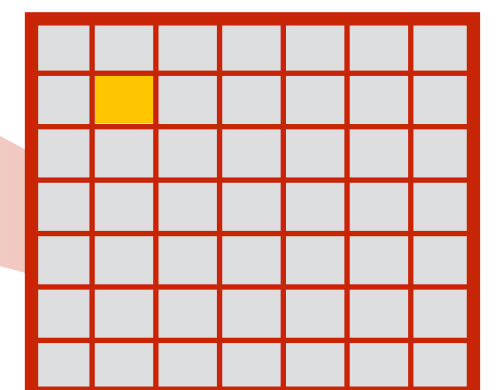
Performance optimization: can evaluate convolutional layers once on large input, then reuse intermediate output many times to approximate response of a subregion of image.



ROI pool



ROI pool

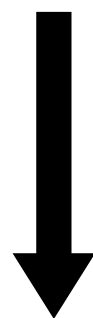


Optimization 2: region of interest pooling

This is a form of “approximate common subexpression elimination”

```
for all proposed regions (x,y,w,h): // 1000's of regions/image
    cropped = image_crop(image, bbox(x,y,w,h))
    resized = image_resize(227,227)
    label = detect_object(resized)
```

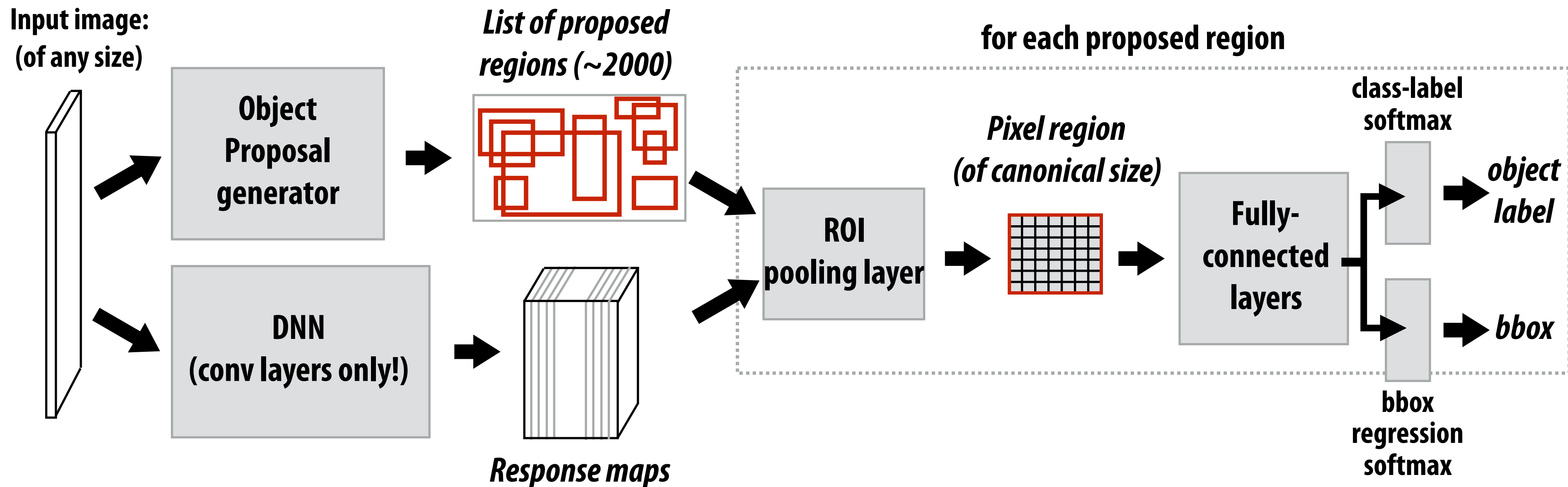
redundant work (many regions overlap, so responses at lower network layers are computed many times)



```
conv5_response = evaluate_conv_layers(image)
for all proposed regions (x,y,w,h):
    region_conv5 = roi_pool(conv5_response, bbox(x,y,w,h))
    label = evaluate_fully_connected_layers(region_conv5)
```

computed once per image

Fast R-CNN pipeline [Girshick 2015]



Evaluation speed: 146x faster than R-CNN (47sec/img → 0.32 sec/img)

[This number excludes cost of proposals]

Training speed: 9x faster than R-CNN

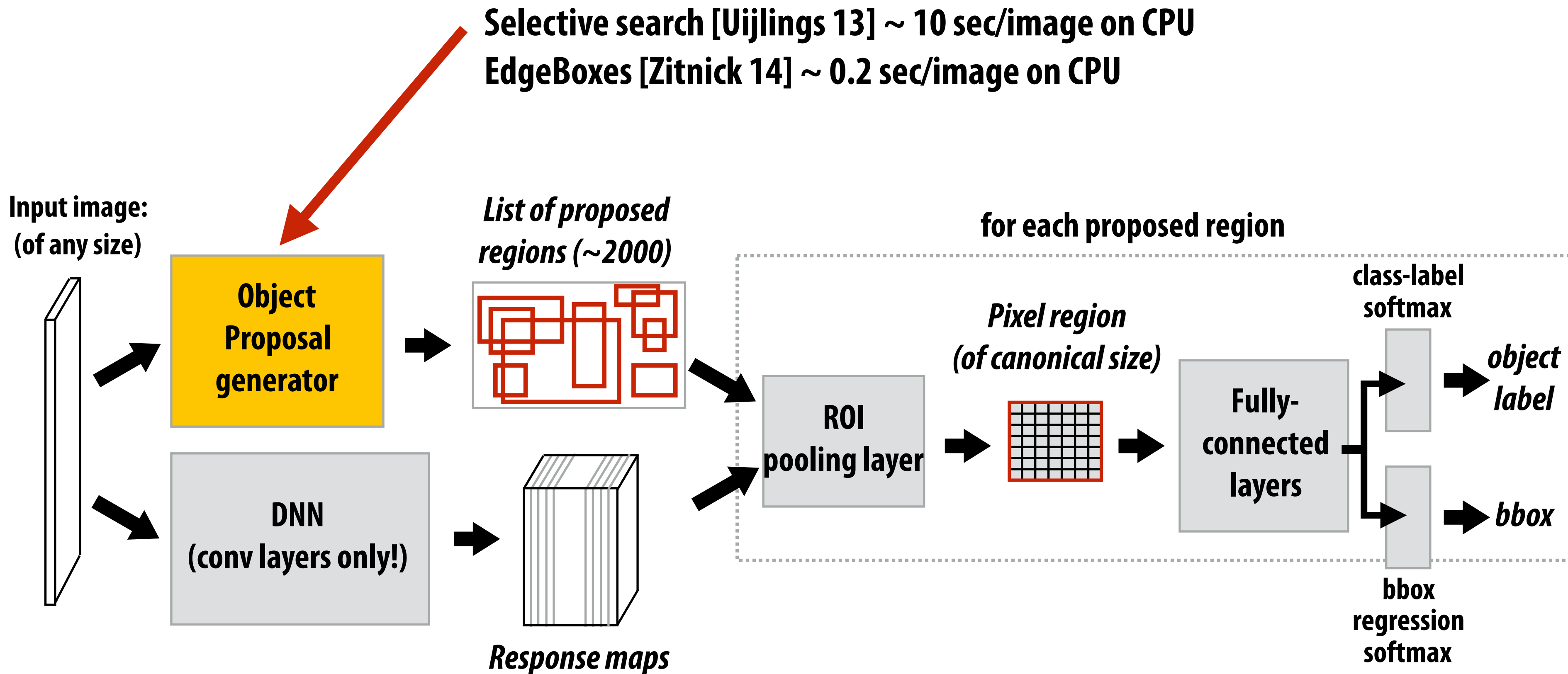
Training mini-batch: pick N images, pick 128/N boxes from each image (allows sharing of conv-layer pre-computation for multiple image-box training samples)

Simultaneously train class predictions and bbox predictions: joint loss = class label loss + bbox loss

Note: training updates weights in BOTH fully connected/softmax layers AND conv layers

| method | train set | aero | bike | bird | boat | bottle | bus | car | cat | chair | cow | table | dog | horse | mbike | persn | plant | sheep | sofa | train | tv | mAP |
|---------------|-----------|------|------|------|------|--------|------|------|------|-------|------|-------|------|-------|-------|-------|-------|-------|------|-------|------|------|
| R-CNN BB [10] | 12 | 79.3 | 72.4 | 63.1 | 44.0 | 44.4 | 64.6 | 66.3 | 84.9 | 38.8 | 67.3 | 48.4 | 82.3 | 75.0 | 76.7 | 65.7 | 35.8 | 66.2 | 54.8 | 69.1 | 58.8 | 62.9 |
| FRCN [ours] | 12 | 80.1 | 74.4 | 67.7 | 49.4 | 41.4 | 74.2 | 68.8 | 87.8 | 41.9 | 70.1 | 50.2 | 86.1 | 77.3 | 81.1 | 70.4 | 33.3 | 67.0 | 63.3 | 77.2 | 60.0 | 66.1 |

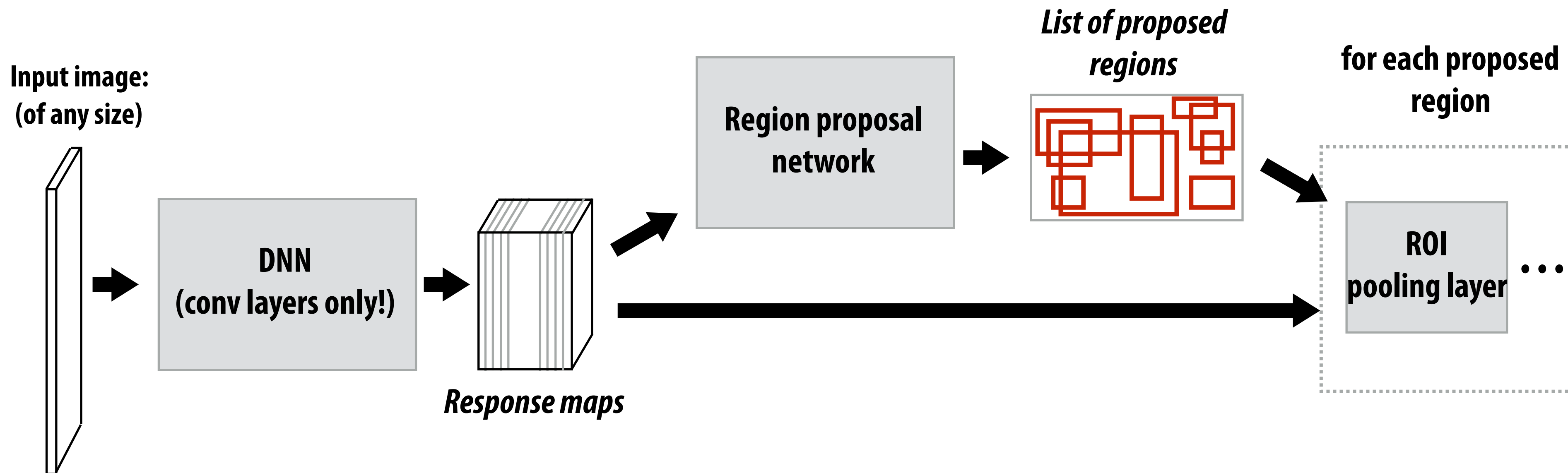
Problem: bottleneck is now generating proposals



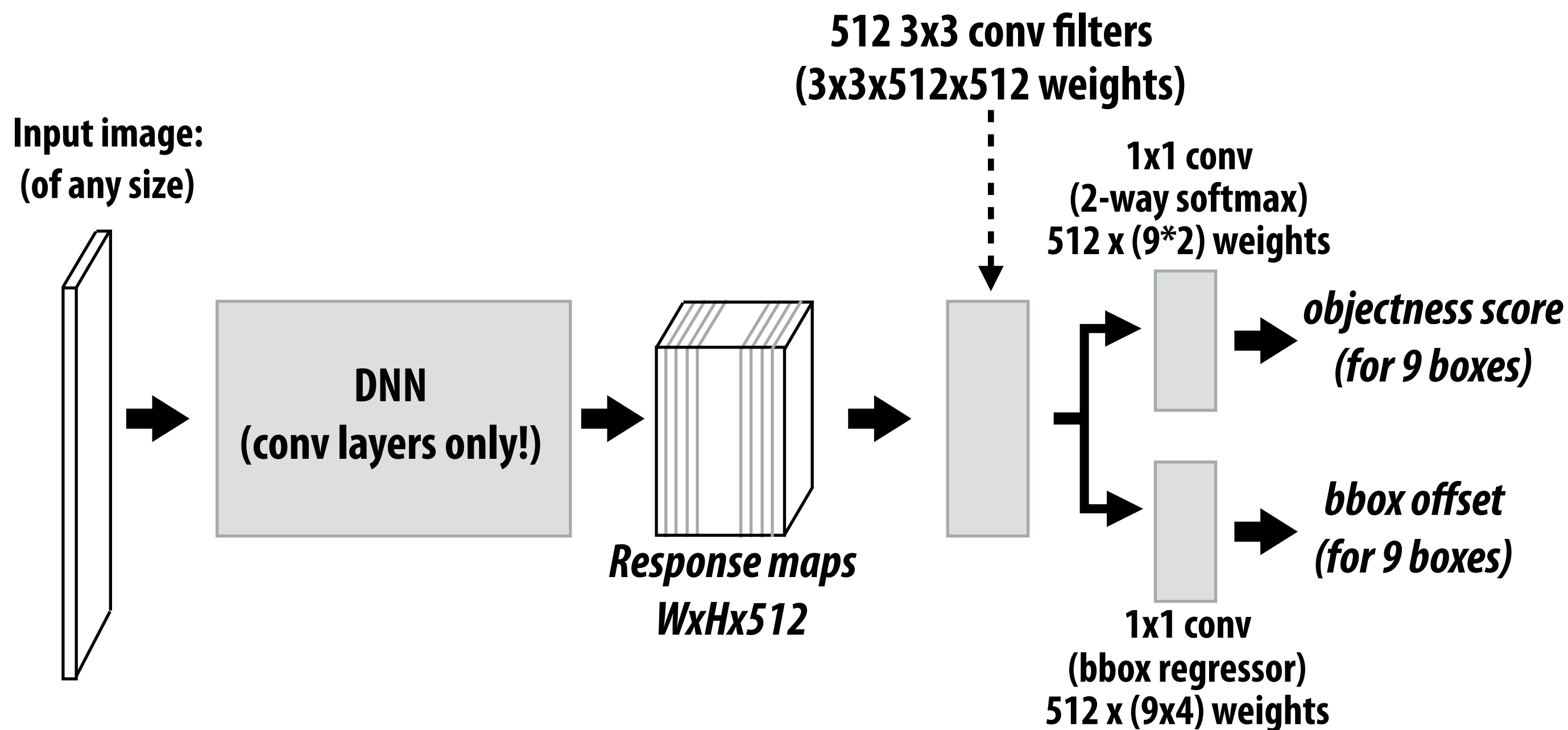
Idea: why not predict regions from the convolutional feature maps that must be computed for detection anyway? (share computation between proposals and detection)

Faster R-CNN using a region proposal network (RPN)

[Ren 2015]



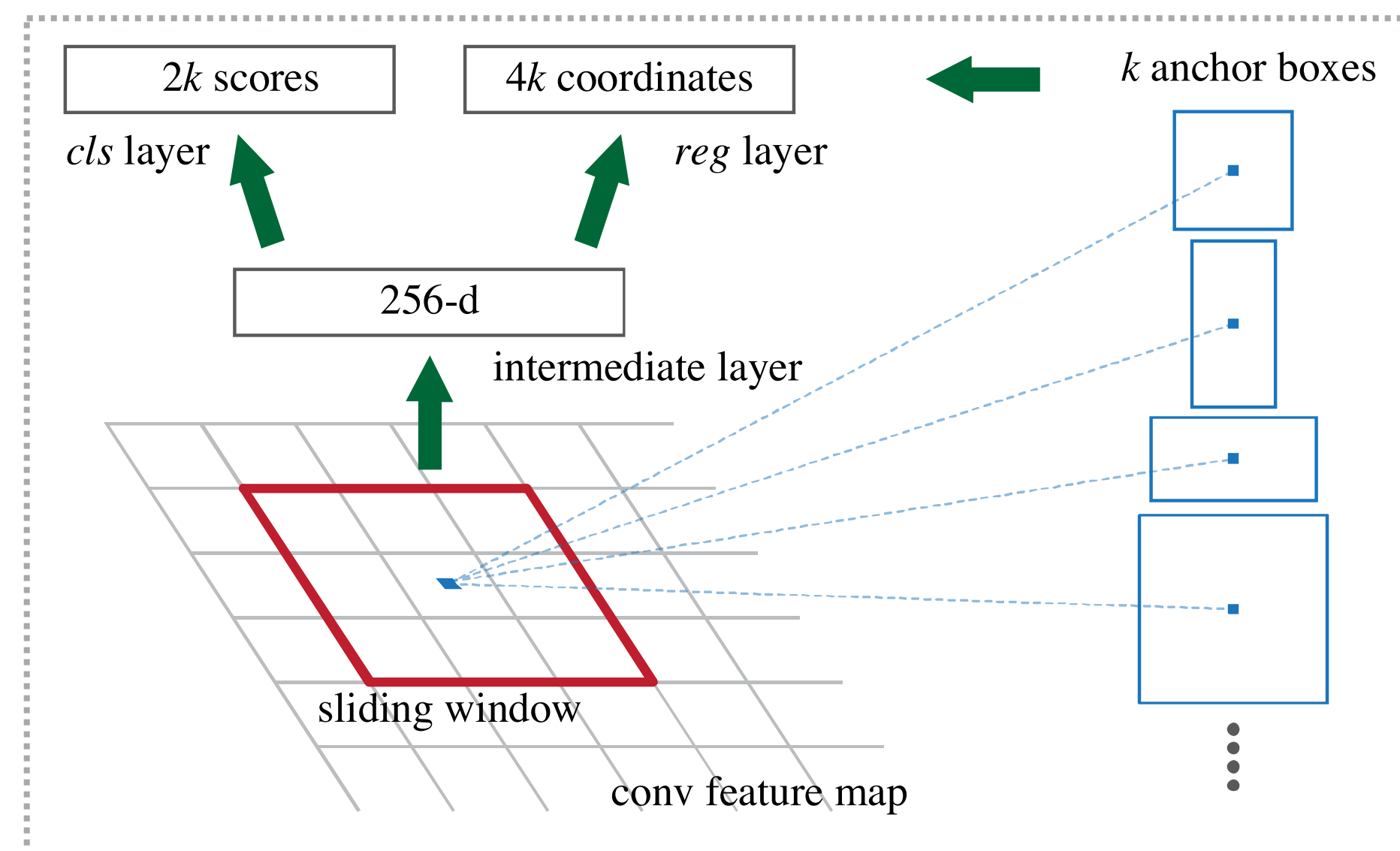
Faster R-CNN using a region proposal network (RPN)



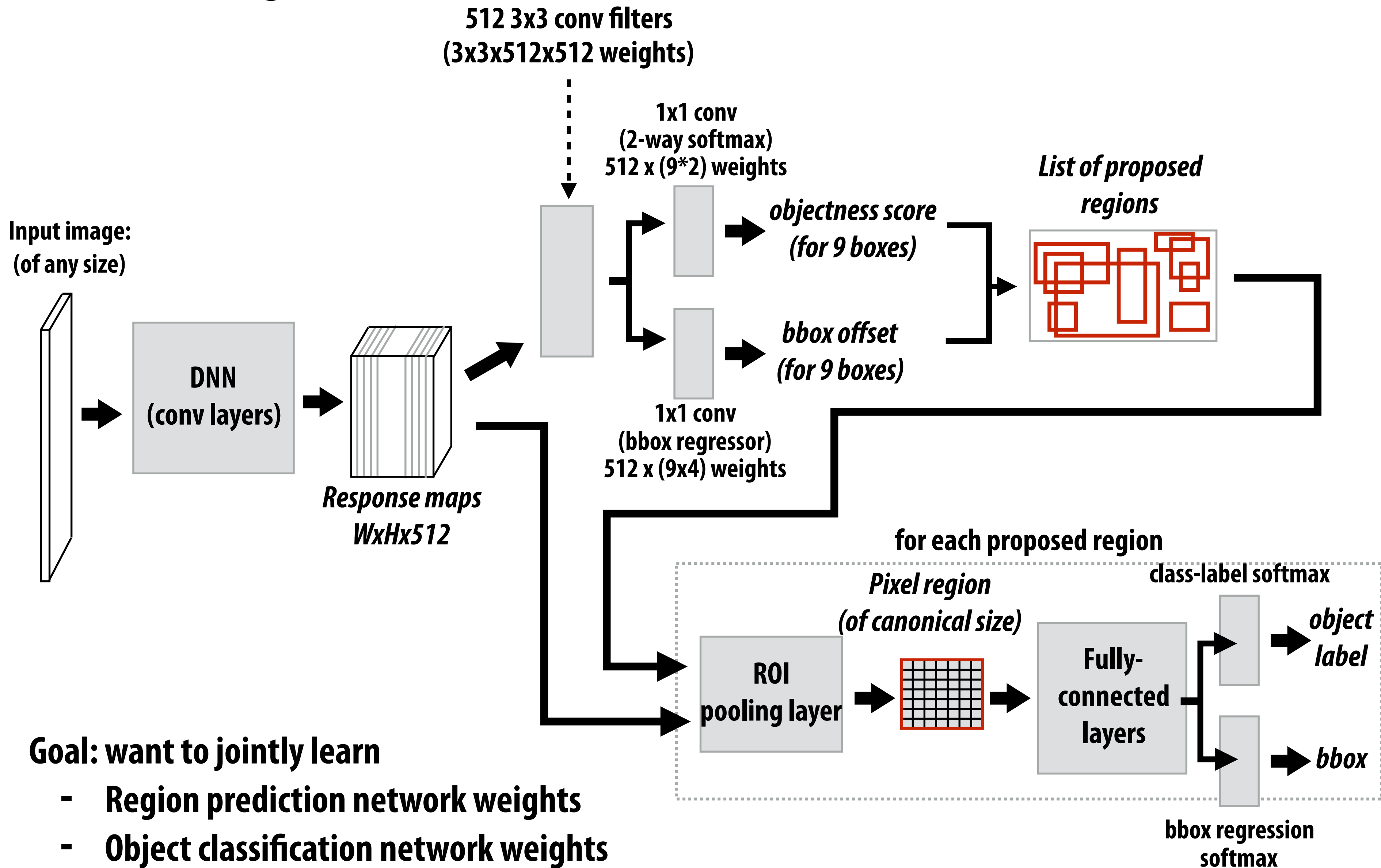
3×3 conv projects into 512-element vector per spatial position (assuming VGG input conv layers, receptive field for each output is $\sim 228 \times 228$ pixels)

At each point assume 9 “anchor boxes” of various aspect ratios and scales

Given 512-element vector predict “objectness score” of each anchor + bbox correction to anchor



Training faster R-CNN



Goal: want to jointly learn

- Region prediction network weights
- Object classification network weights
- **While constraining initial conv layers to be the same (for efficiency)**

Alternating training strategy

- **Train region proposal network (RPN)**
 - **Using loss based on ground-truth object bounding boxes**
 - **Positive example: intersection over union with ground truth box is above threshold**
 - **Negative example: intersection over union is less than threshold**
- **Then use trained RPN to train Fast R-CNN**
 - **Using loss based on detections and bbox regression**
- **Use conv layers from R-CNN to initialize RPN**
- **Fine-tune RPN**
 - **Using loss based on ground-truth boxes**
- **Use updated RPN to fine tune Fast R-CNN**
 - **Using loss based on detections and bbox regression**
- **Repeat...**

- **Notice: solution learns to predict boxes that are “good for object-detection task”**
 - **“End-to-end” optimization for object-detection task**
 - **Compare to using off-the-shelf object-proposal algorithm**

Faster R-CNN results

Specializing region proposals for object-detection task yields better accuracy.

SS = selective search for object proposals

| method | # proposals | data | mAP (%) |
|------------------------------|-------------|--------|-------------|
| SS | 2000 | 12 | 65.7 |
| SS | 2000 | 07++12 | 68.4 |
| RPN+VGG, shared [†] | 300 | 12 | 67.0 |
| RPN+VGG, shared [‡] | 300 | 07++12 | 70.4 |

Shared convolutions improve algorithm performance:

Values are times in ms

| model | system | conv | proposal | region-wise | total | rate |
|-------|------------------|------|-----------|-------------|------------|--------------|
| VGG | SS + Fast R-CNN | 146 | 1510 | 174 | 1830 | 0.5 fps |
| VGG | RPN + Fast R-CNN | 141 | 10 | 47 | 198 | 5 fps |

Summary

- **Knowledge of algorithm and properties of DNN used to gain algorithmic speedups**
 - Not just “modify the schedule of the loops”
- **Key insight: sharing results of convolutional layer computations:**
 - Between different proposed regions (proposed object bboxes)
 - Between region proposal logic and detection logic
- **Example of “end-to-end” training**
 - Back-propagate through entire algorithm to train all components at once
 - Better accuracy: globally optimize the various parts of the algorithm to be optimal for given task (Faster R-CNN: how to propose boxes learned simultaneously with detection logic)
 - Can constrain learning to preserve performance characteristics (Faster R-CNN: conv layer weights shared across RPN and detection task)

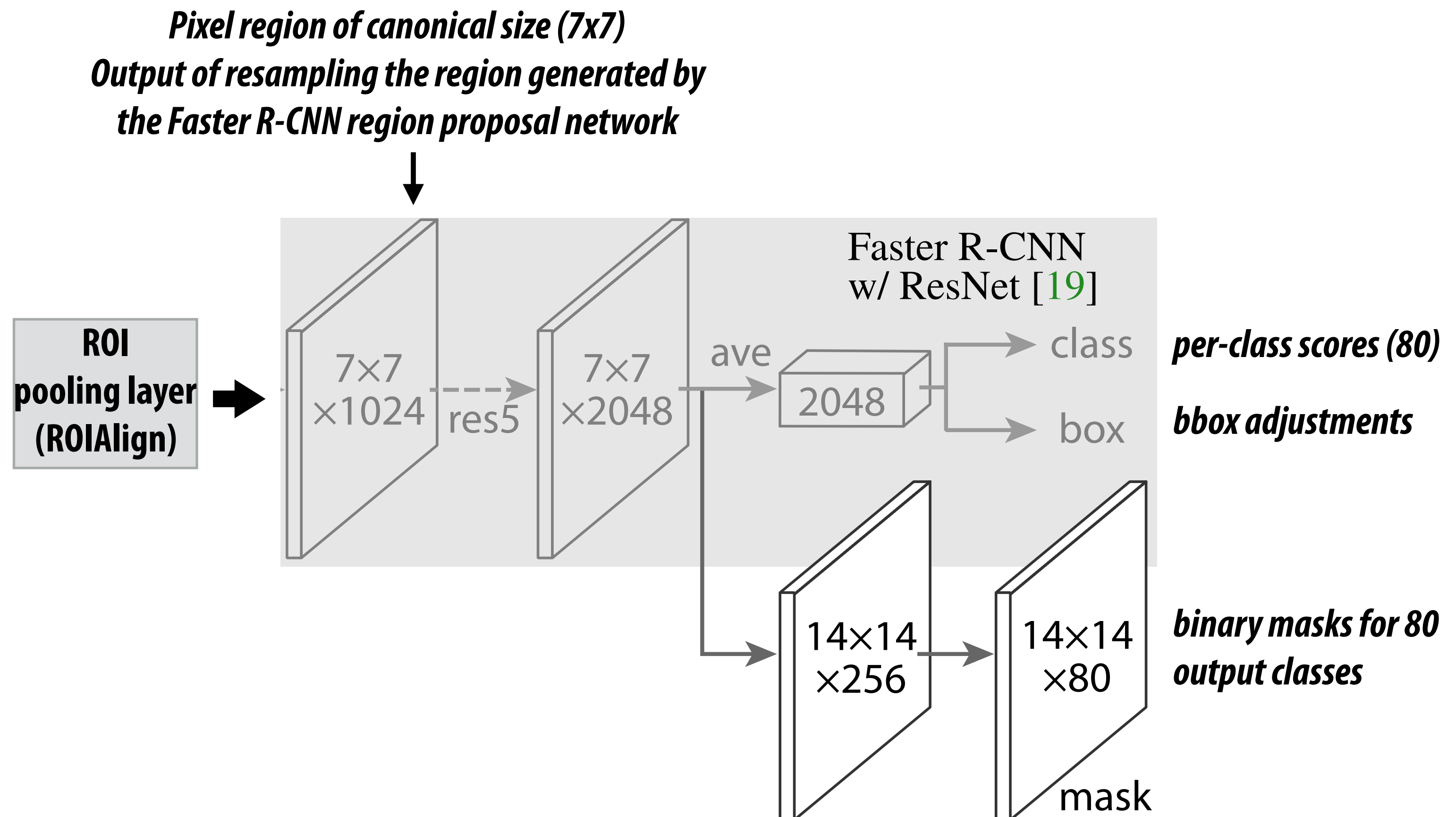
Extending to instance segmentation



[Image credit: He et al. 2017]

Mask RCNN

- **Extend Faster R-CNN to also emit a segmentation per box**
 - **Previously: box and class emitted in parallel**
 - **Now: box, class, and segmentation emitted in parallel**



Mask R-CNN for human pose

- Loss based on bitmapped with hot pixels at joint keypoint locations rather than segmentation masks



An alternative approach to object detection

Recall structure of algorithms so far: (reduce detection to classification)

```
for all proposed regions (x,y,w,h):  
    cropped = image_crop(image, bbox(x,y,w,h))  
    resized = image_resize(cropped, classifier_width, classifier_height)  
    label = classify_object(resized)  
    bbox_adjustment = adjust_bbox(resized)
```

New approach to detection:

```
for each level  $l$  of network:  
    for each (x,y) position in output:  
        use region around  $(l,x,y)$  to directly predict which anchor boxes  
        centered at (x,y) are valid and class score for that box
```

If there are B anchor boxes and C classes, then...

At each (l,x,y) , prediction network has $B \times (C + 4)$ outputs

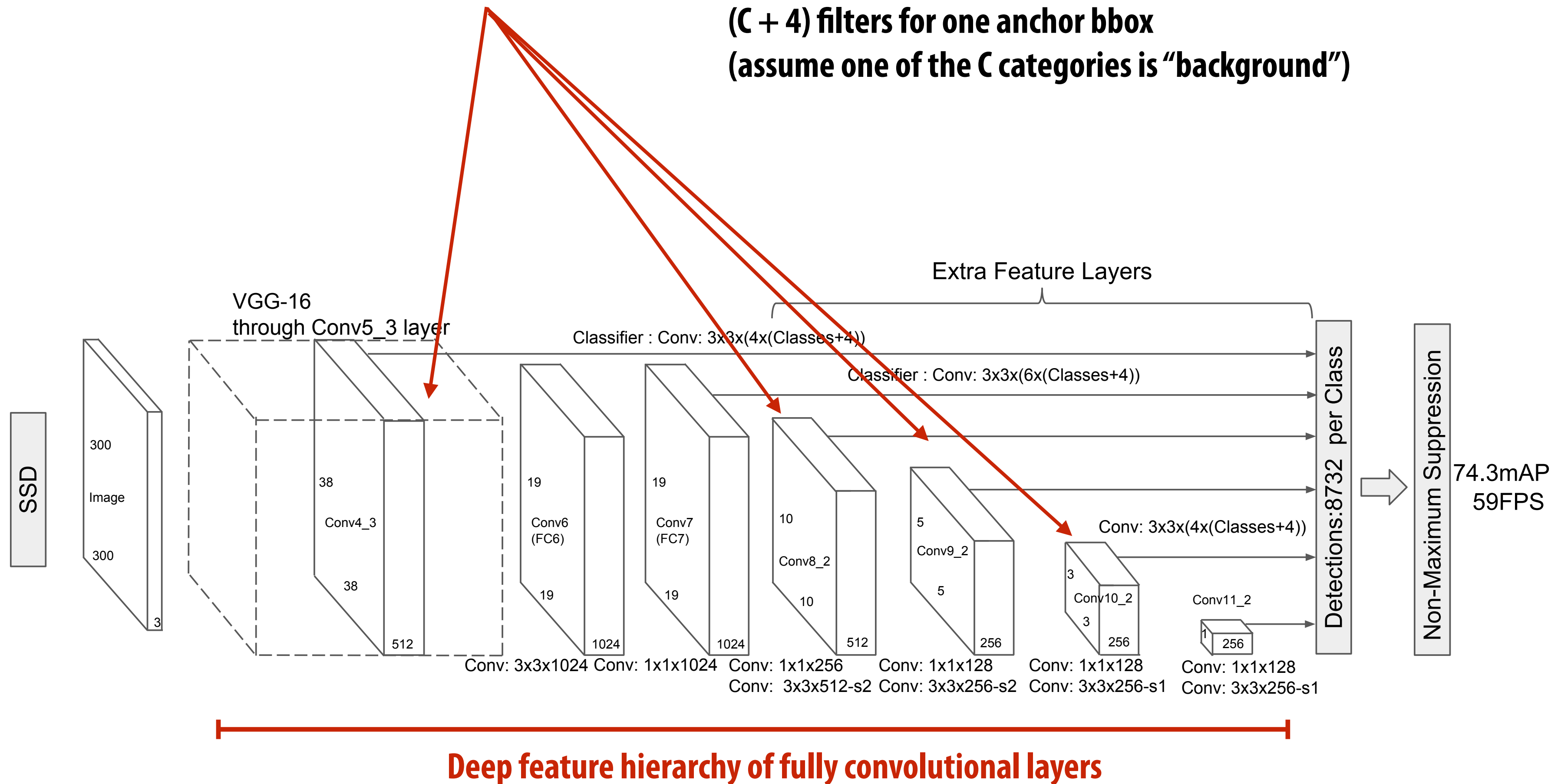
For each anchor B , there are C class probabilities + 4 values to adjust the anchor box

SSD: Single shot multi box detector

[Lui ECCV 2016]

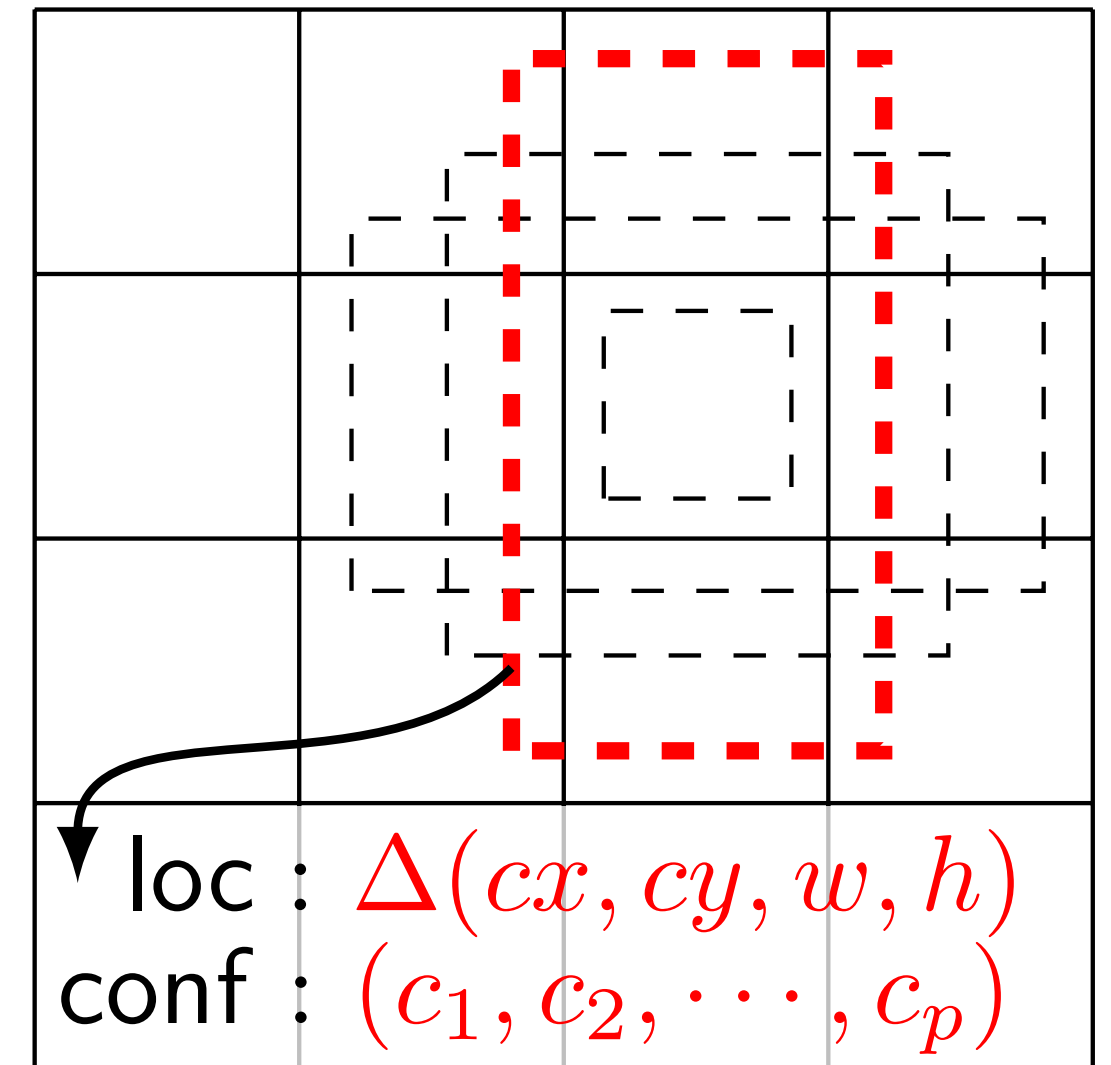
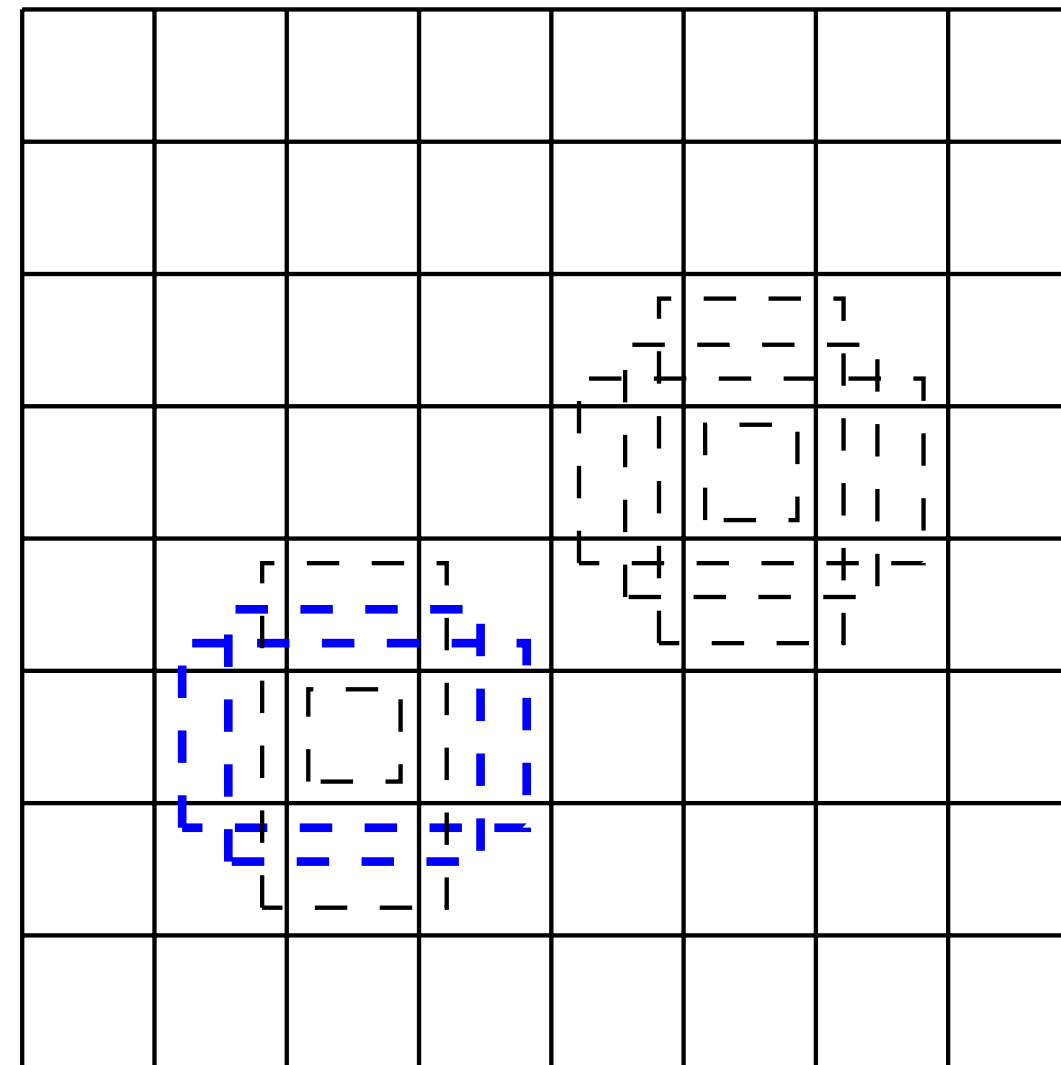
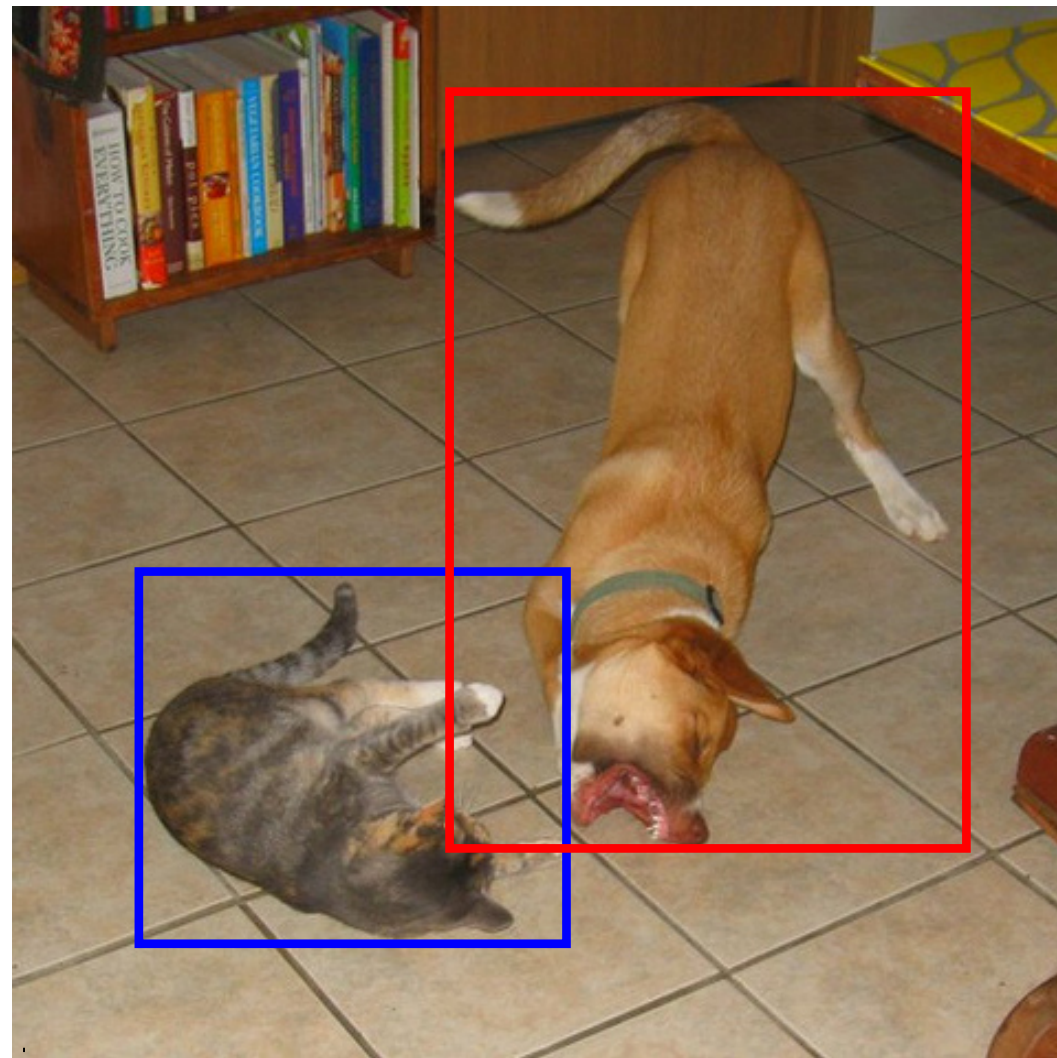
multibox detectors operating on different scales of features

If feature maps have P channels (e.g., $P=512$ and 256 below)
Each classifier is a $3 \times 3 \times P$ filter
 $(C + 4)$ filters for one anchor bbox
(assume one of the C categories is "background")



Note: diagram shows only the feature maps

SSD anchor boxes



loc : $\Delta(cx, cy, w, h)$
conf : (c_1, c_2, \dots, c_p)

(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Anchor boxes at each feature map level are of different sizes

Intuition: receptive field of cells at higher levels of the network (lower resolution feature maps) is a larger fraction of the image, have information to make predictions for larger boxes

Object detection performance

600x600 input images

COCO-trained models {#coco-models}

| Model name | Speed (ms) | COCO mAP[^1] | Outputs |
|--|------------|--------------|---------|
| ssd_mobilenet_v1_coco | 30 | 21 | Boxes |
| ssd_inception_v2_coco | 42 | 24 | Boxes |
| faster_rcnn_inception_v2_coco | 58 | 28 | Boxes |
| faster_rcnn_resnet50_coco | 89 | 30 | Boxes |
| faster_rcnn_resnet50_lowproposals_coco | 64 | | Boxes |
| rfcn_resnet101_coco | 92 | 30 | Boxes |
| faster_rcnn_resnet101_coco | 106 | 32 | Boxes |
| faster_rcnn_resnet101_lowproposals_coco | 82 | | Boxes |
| faster_rcnn_inception_resnet_v2_atrous_coco | 620 | 37 | Boxes |
| faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco | 241 | | Boxes |
| faster_rcnn_nas | 1833 | 43 | Boxes |
| faster_rcnn_nas_lowproposals_coco | 540 | | Boxes |

[Credit: Tensorflow detection model zoo]

Back to video

Interest in processing video efficiently

- **Benefits to datacenter applications:**
 - **Lower cost/frame enables processing of more streams (e.g., thousands of webcams)**
- **Benefits to edge devices:**
 - **Cheaper per frame costs, real-time performance on cheaper/lower energy computing hardware**
 - **Lower latency per frame**
 - **Example: automated braking systems target ~40ms sense to brake**

Thought experiment

Imagine we wanted to detect people/cars/bikes in a video stream



Trick 0: video stream subsampling

- **Spatial downsampling: run detector on low-resolution image**
- **Temporal subsampling: run detector at low frame rate**

Trick 1: exploit temporal coherence

Temporal differencing

- **Idea: use labels from empty frame image if similar to background image**



(a) empty frame



(b) frame with a car



(c) subtracted frames

- **Idea: use same result as previous frame if two frames are sufficiently similar**
 - **How to define sufficiently similar? (thresholds)?**
 - **Differences in feature space more robust than over pixels**

Tracking

Evaluate expensive detector sparsely in time (e.g., every 1/2 second), then use more efficient tracking algorithm to update annotations over sequence of frames

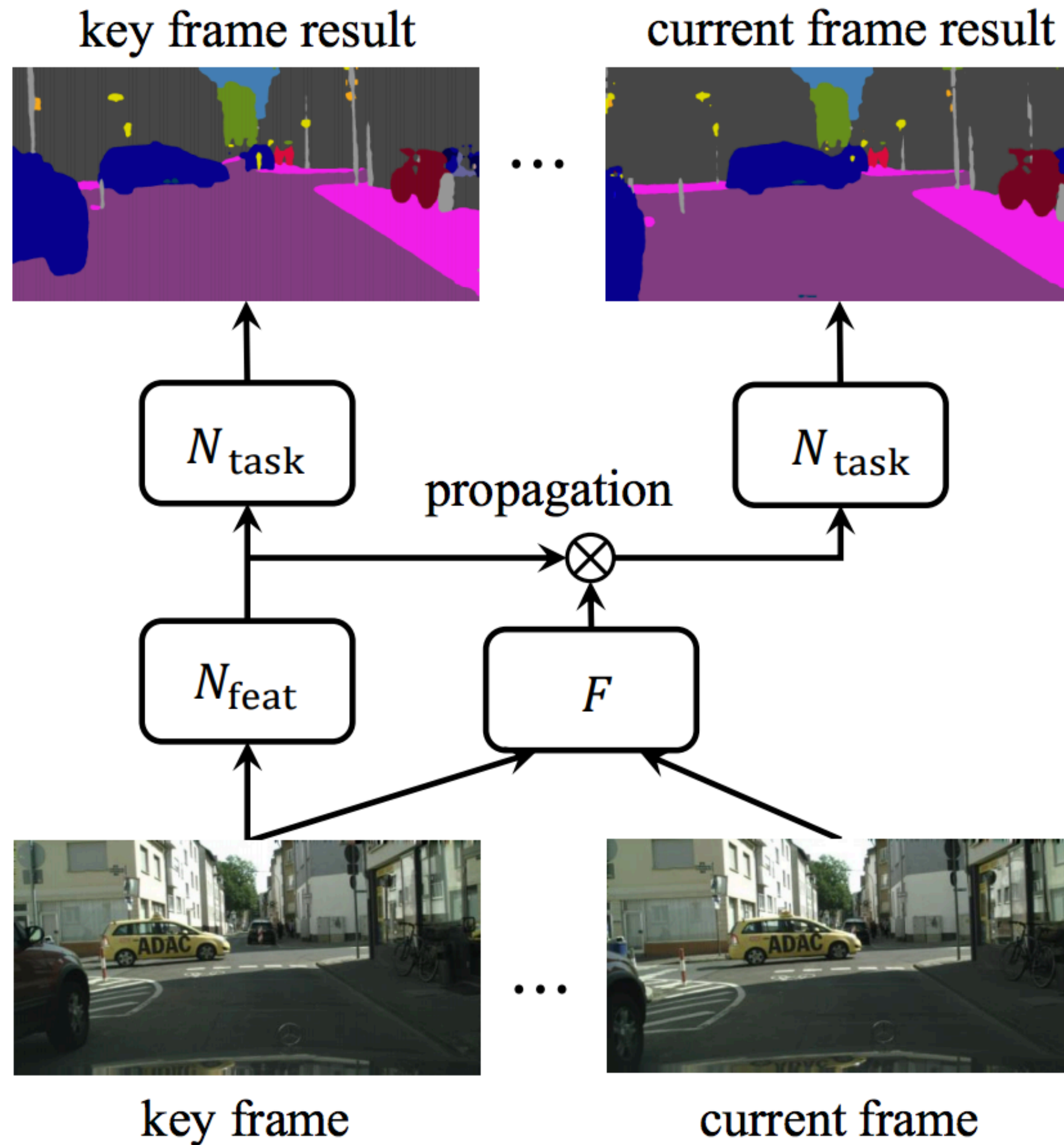


Tracking

Evaluate expensive detector sparsely in time (e.g., every 1/2 second), then use more efficient tracking algorithm to update annotations over sequence of frames



Leveraging motion in the network



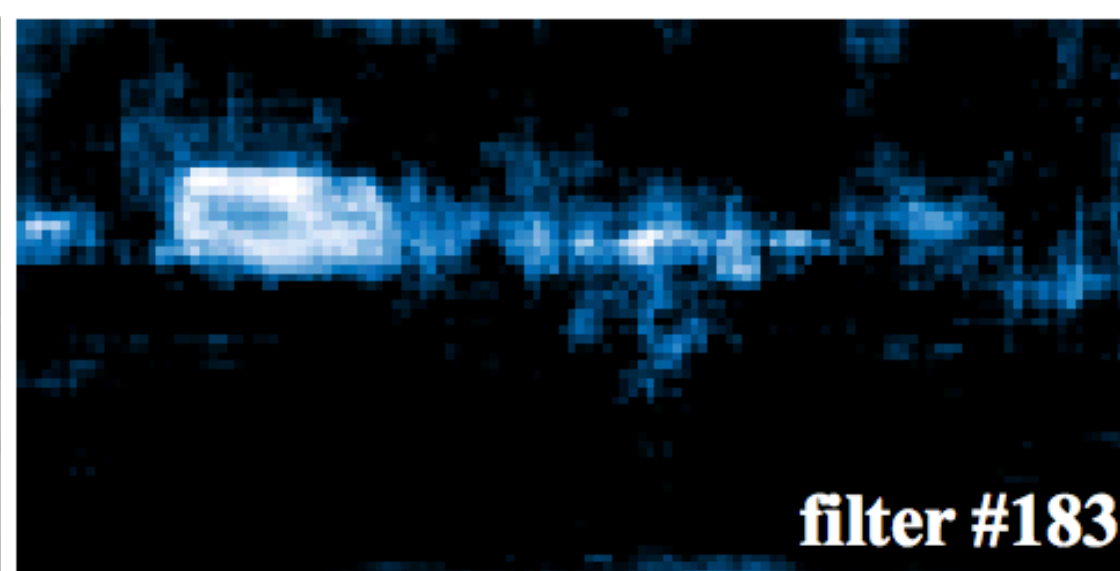
Key idea: given features (or final segmentation result from prior frame) use flow between prior and current frame to advect features (or segmentation) to new frame.

In other words: it's easier to produce the result for the current frame if you have the result from the prior frame

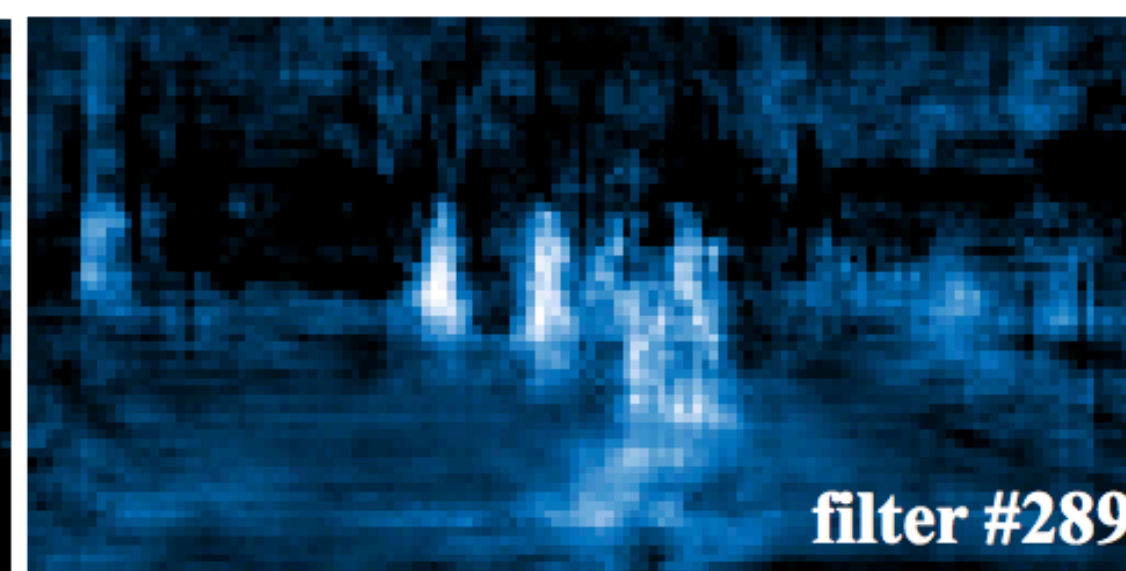
Leveraging motion in the network



key frame



filter #183

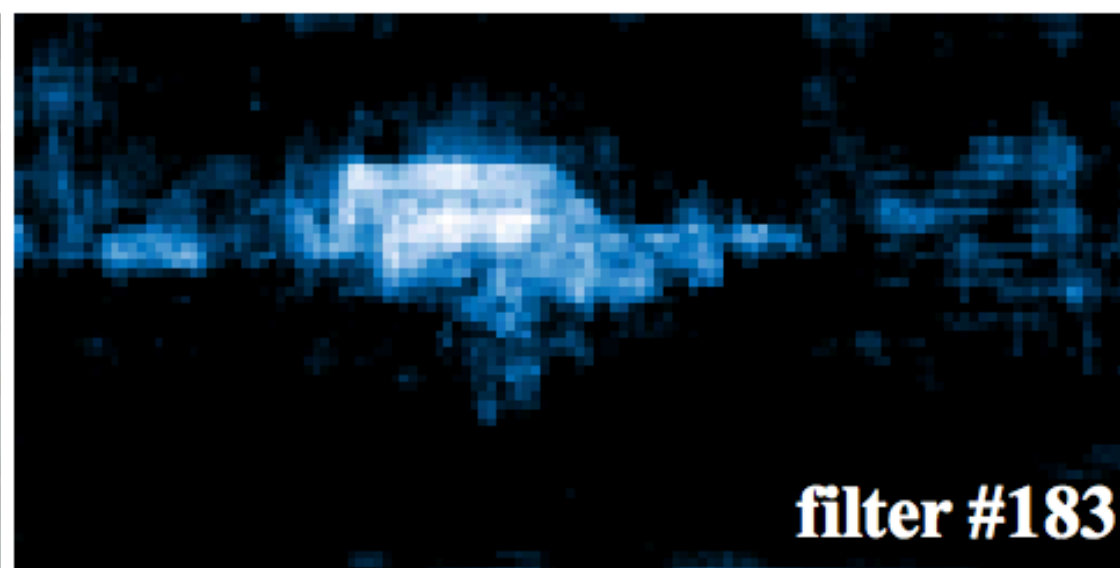


filter #289

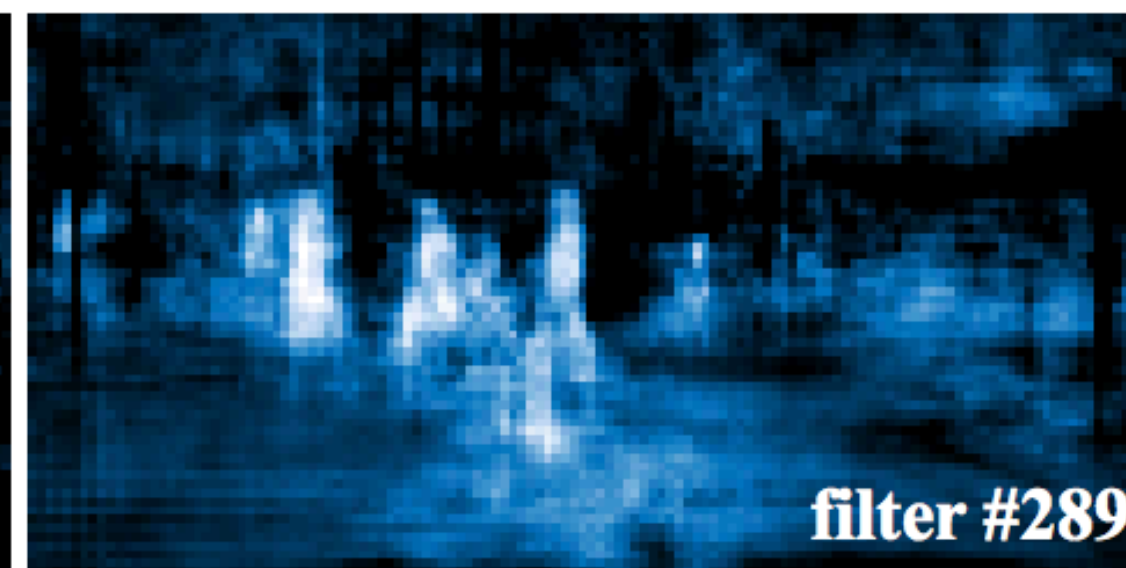
key frame feature maps



current frame



filter #183

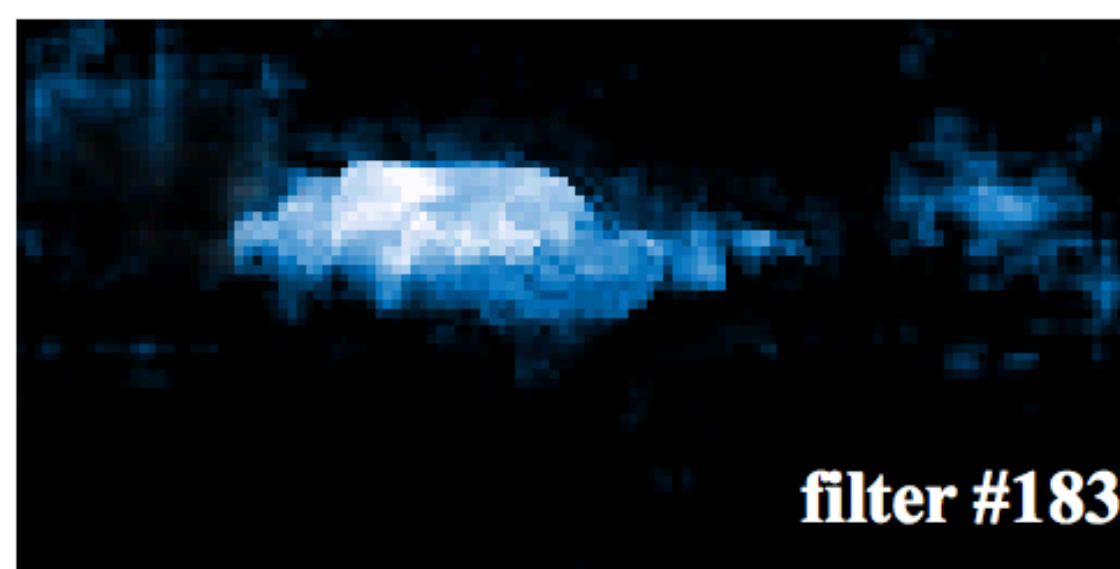


filter #289

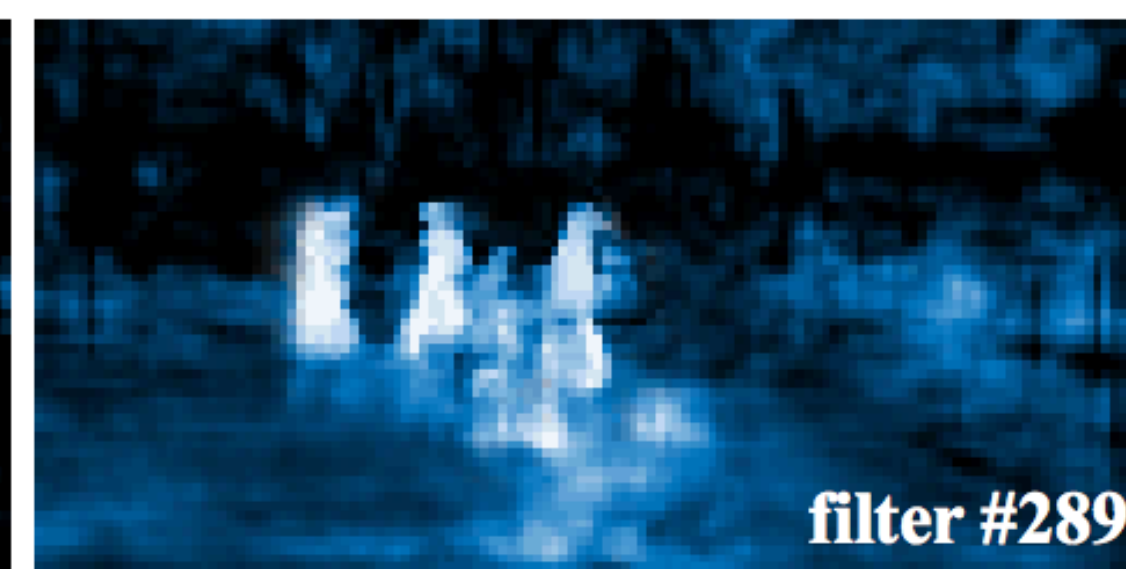
current frame feature maps



flow field



filter #183



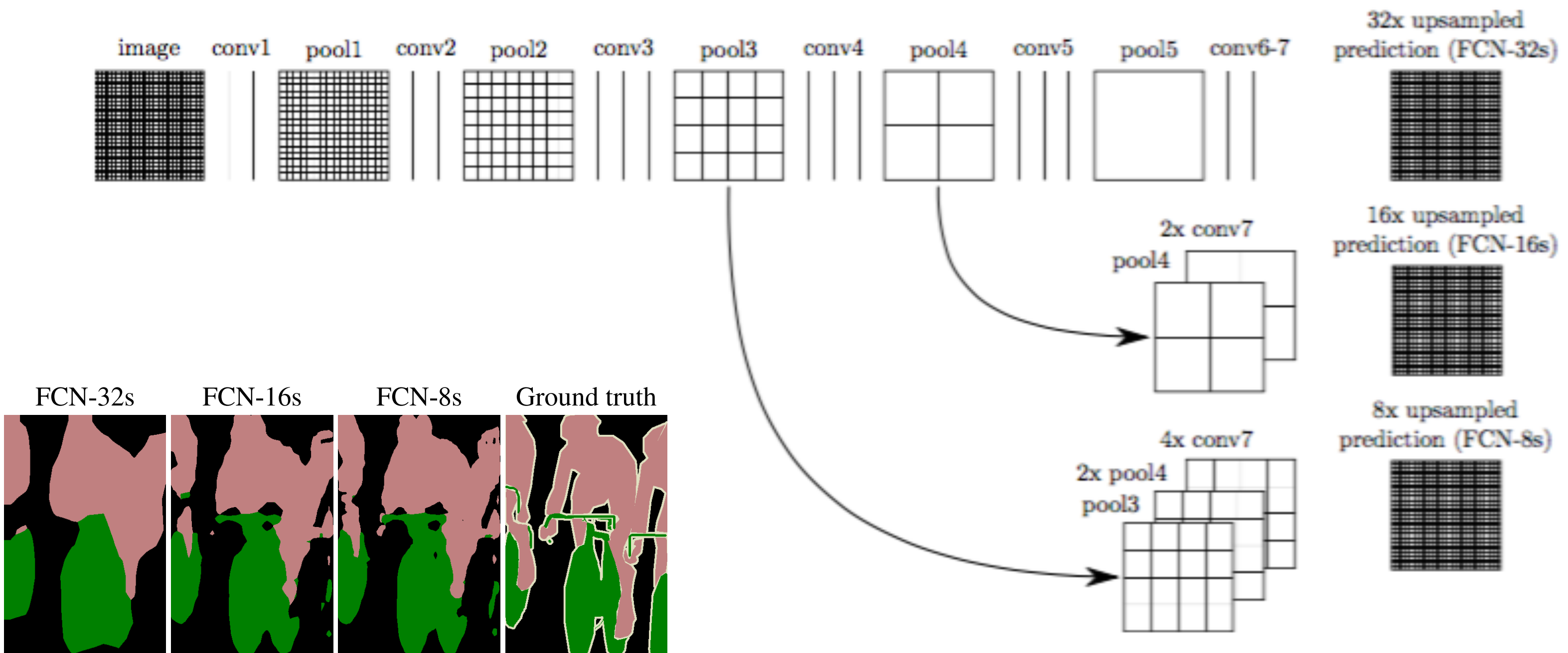
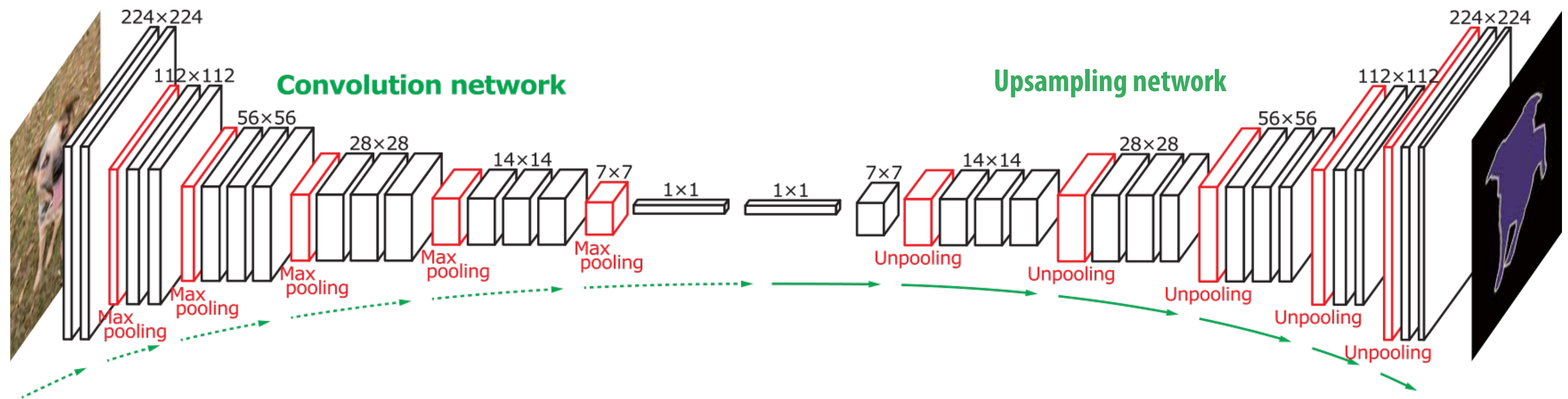
filter #289

propagated feature maps

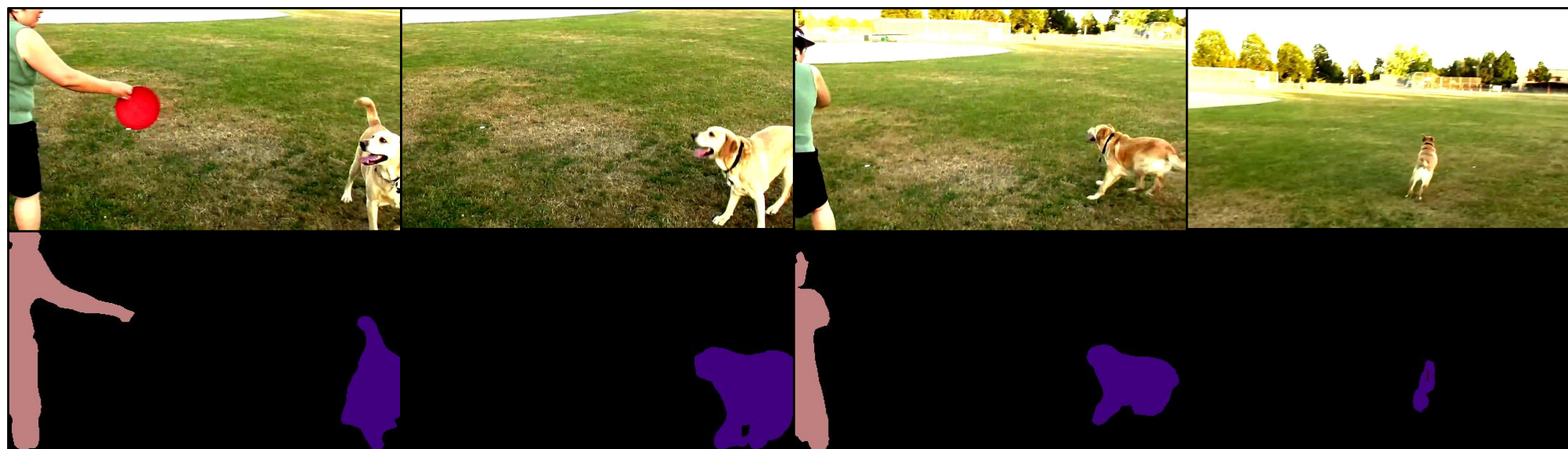
In practice: despite “intellectual appeal” of advecting features, paper results show advecting segmentation is as good as advecting features.

Trick 2: exploit temporal coherence at different scales

A fully convolutional network for image segmentation

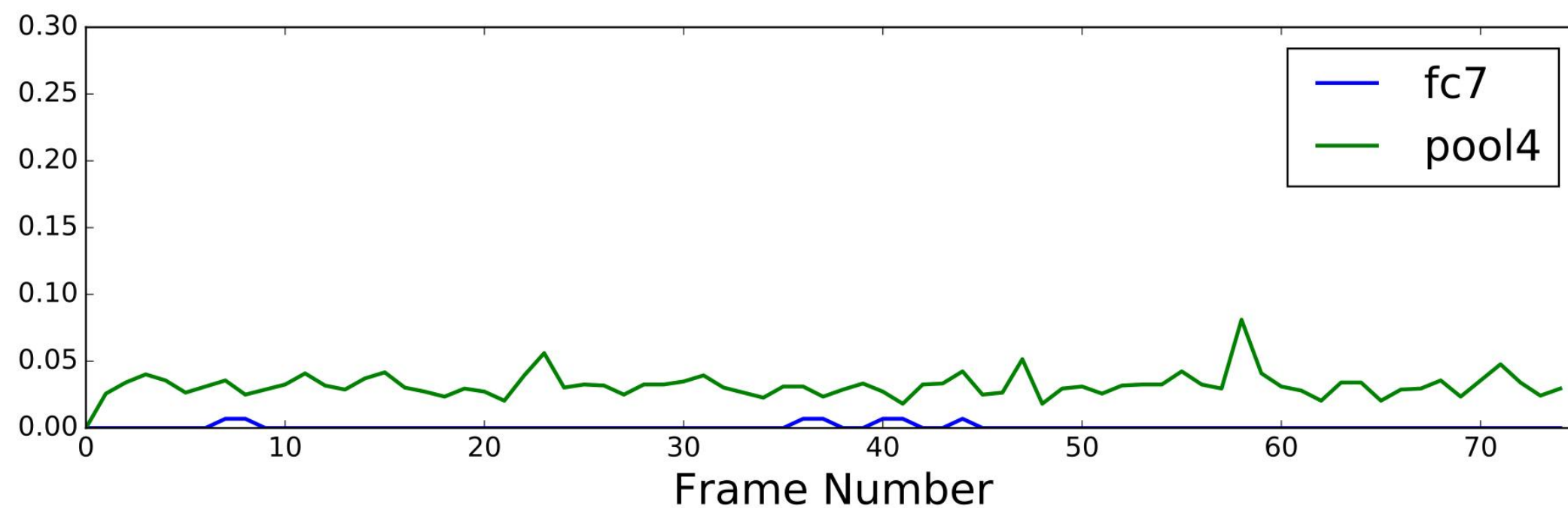
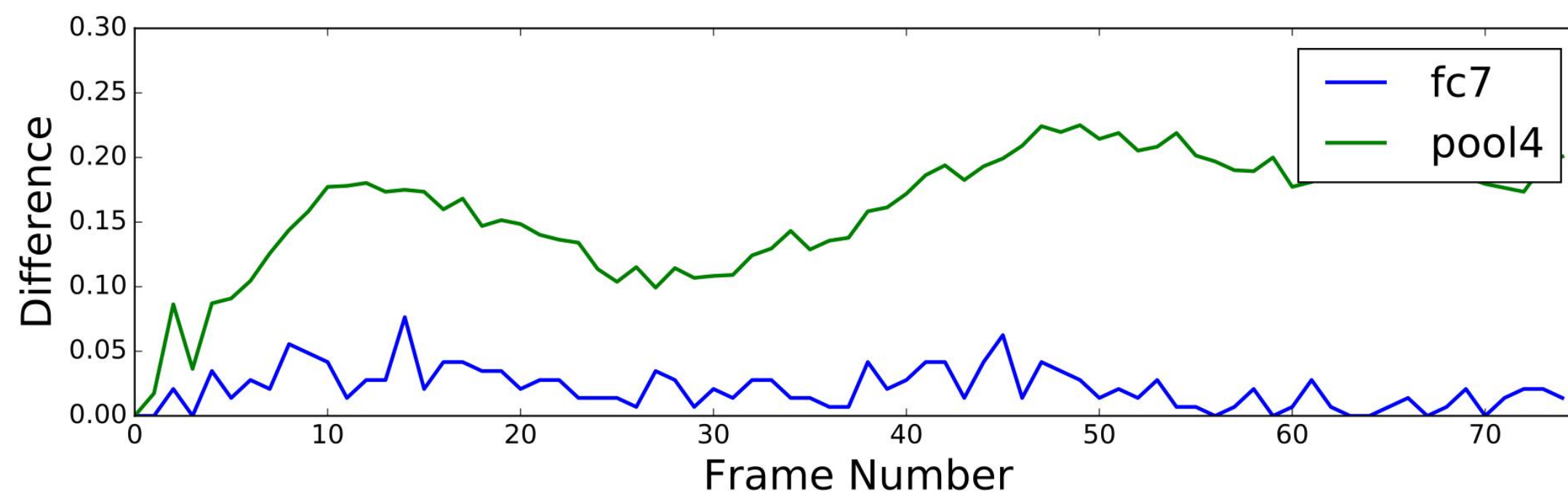


Temporal stability of deep(er) features



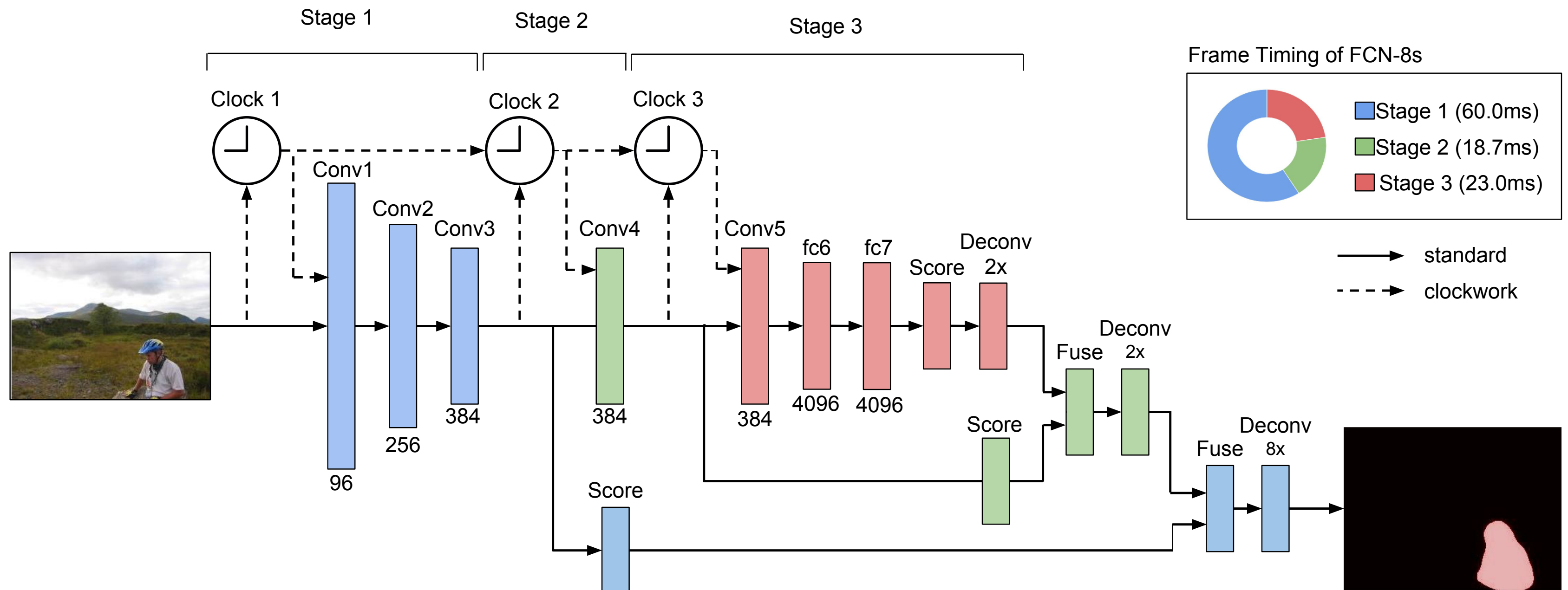
Observation:
Deeper features feature more temporal stability

(more “semantic” information changes less rapidly in a scene)



[Shelhamer ECCV16]

Clockwork network: reuse deeper layer outputs in subsequent frames

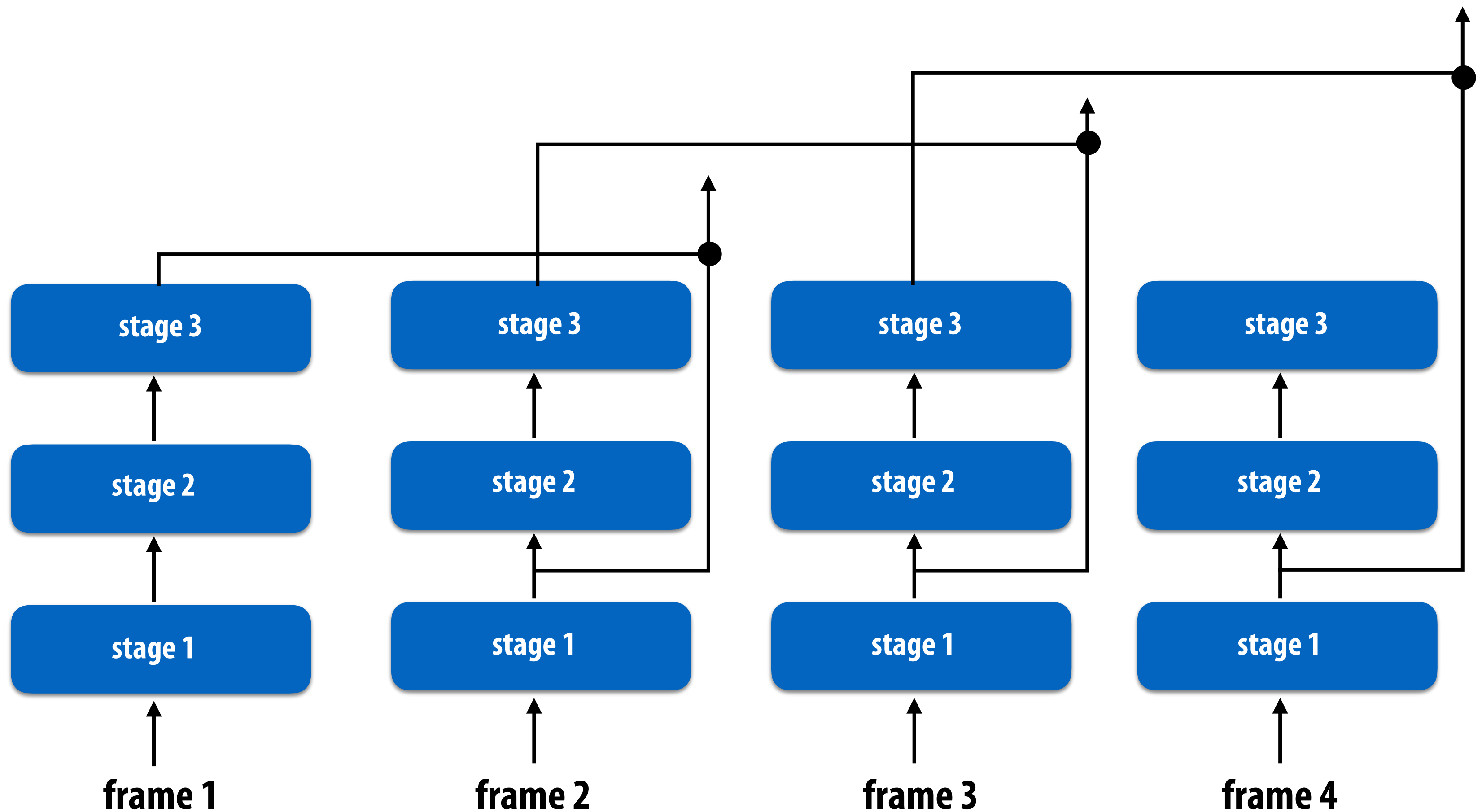


Evaluate lower (early) layers each frame

Optionally combine (fresh) output of lower layers with output of higher layers from previous frames.

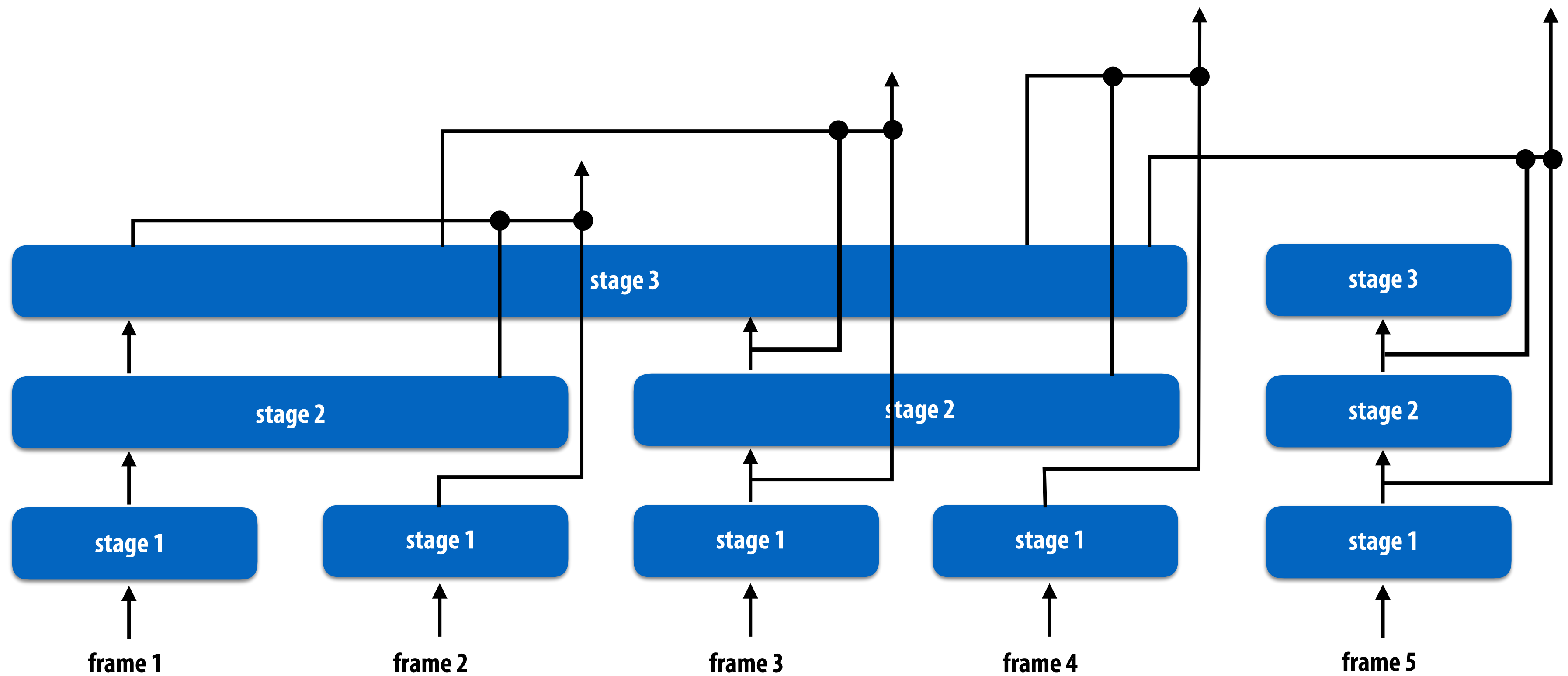
Clockwork convnet

- **Reduced latency: generate output only after evaluating first layer**

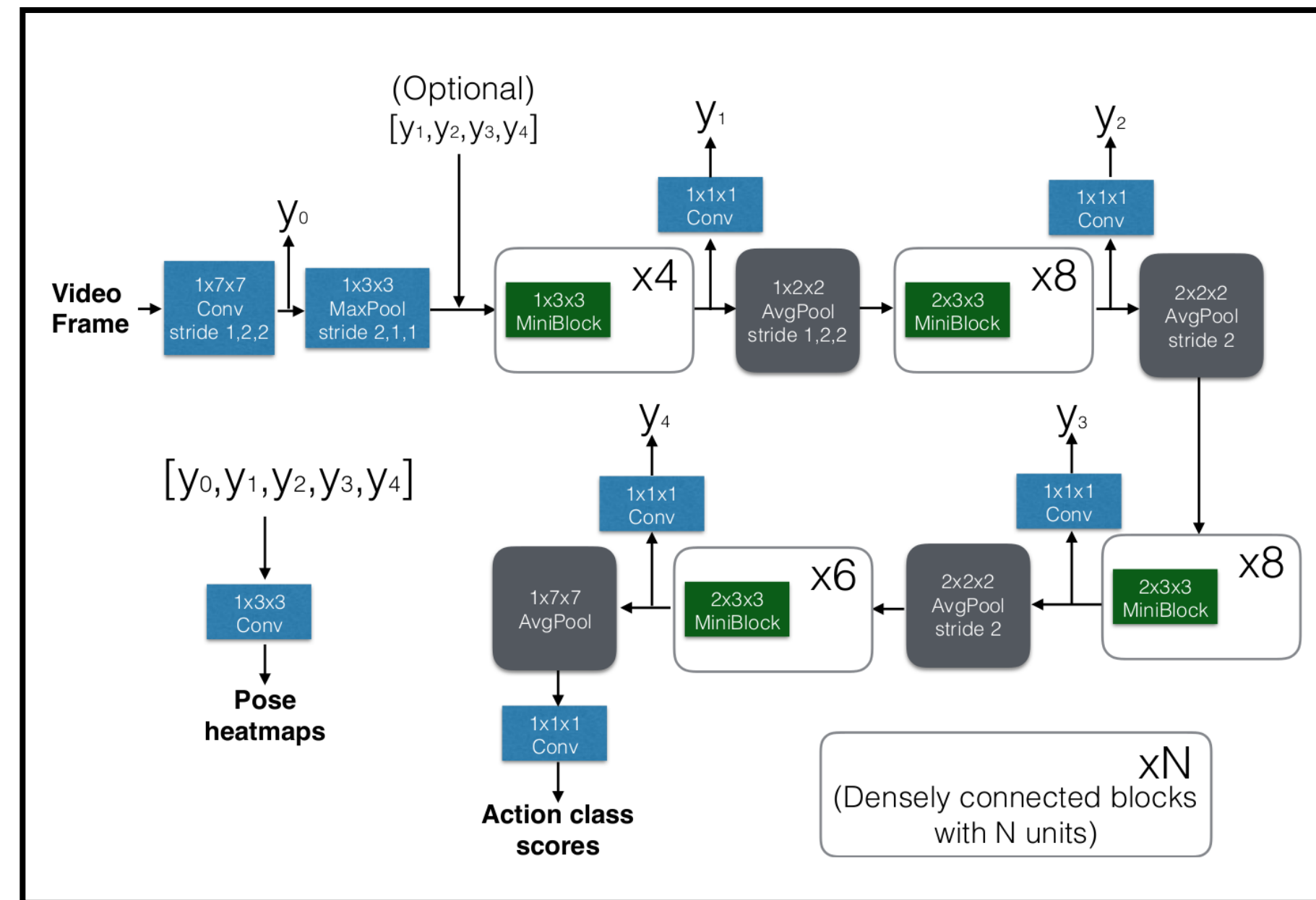
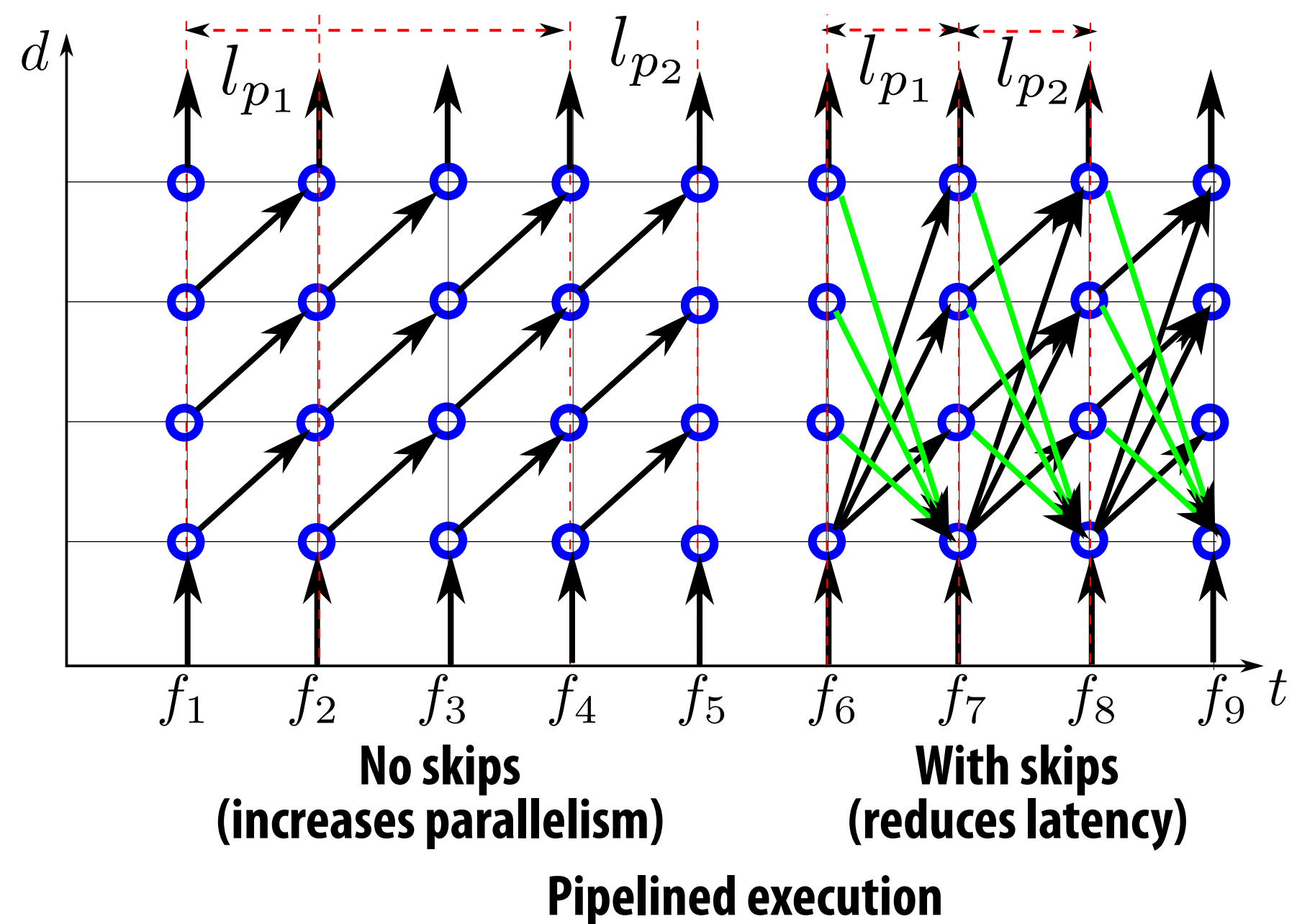
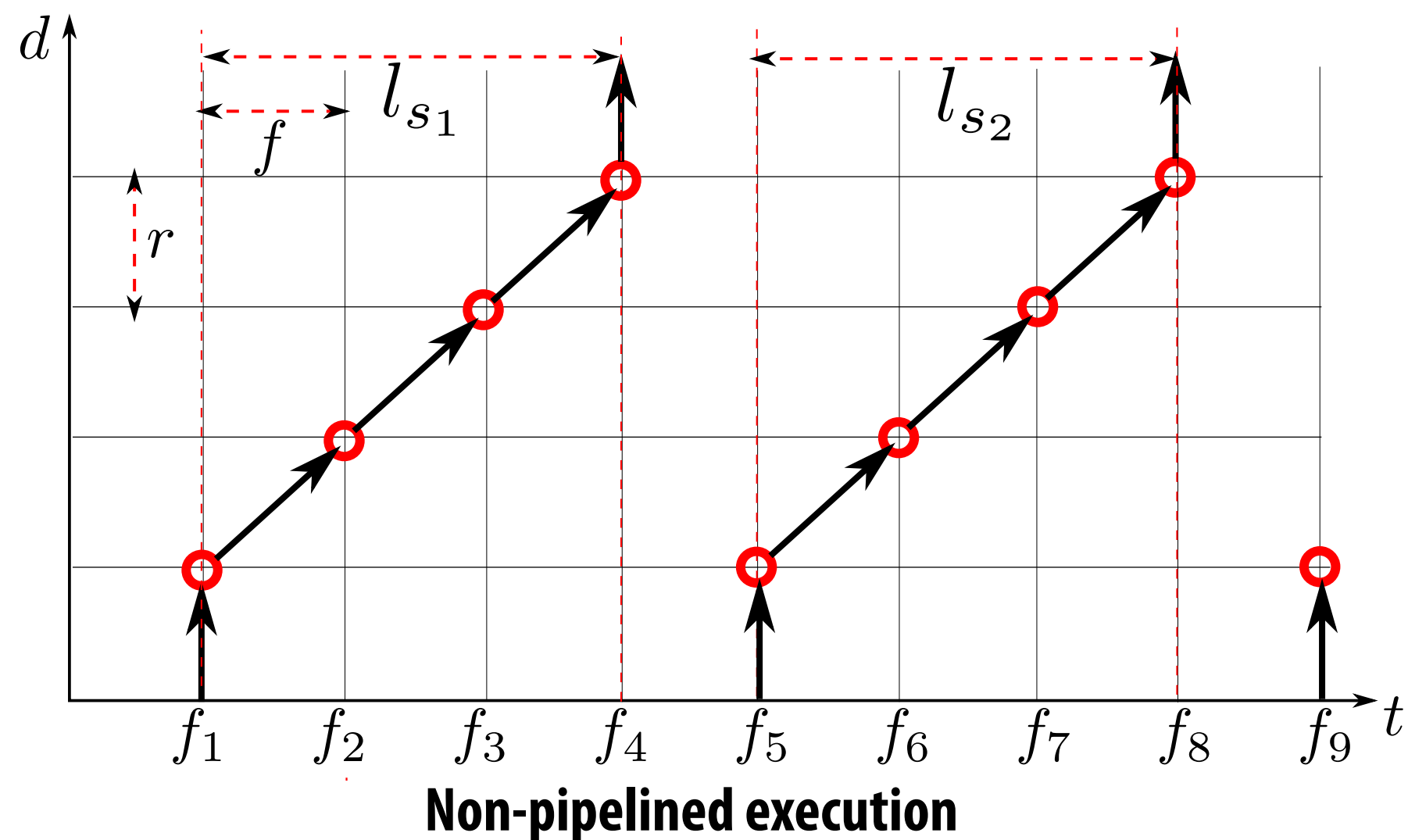


Clockwork convnet

- Increase throughput: update higher layers at a lower rate



Another example: parallel video networks



[Carriera et al. 2018]

Stanford CS348K, Fall 2018

Trick 3: specialize to content

(specialization to video content can be viewed as a form of exploiting temporal coherence, why?)

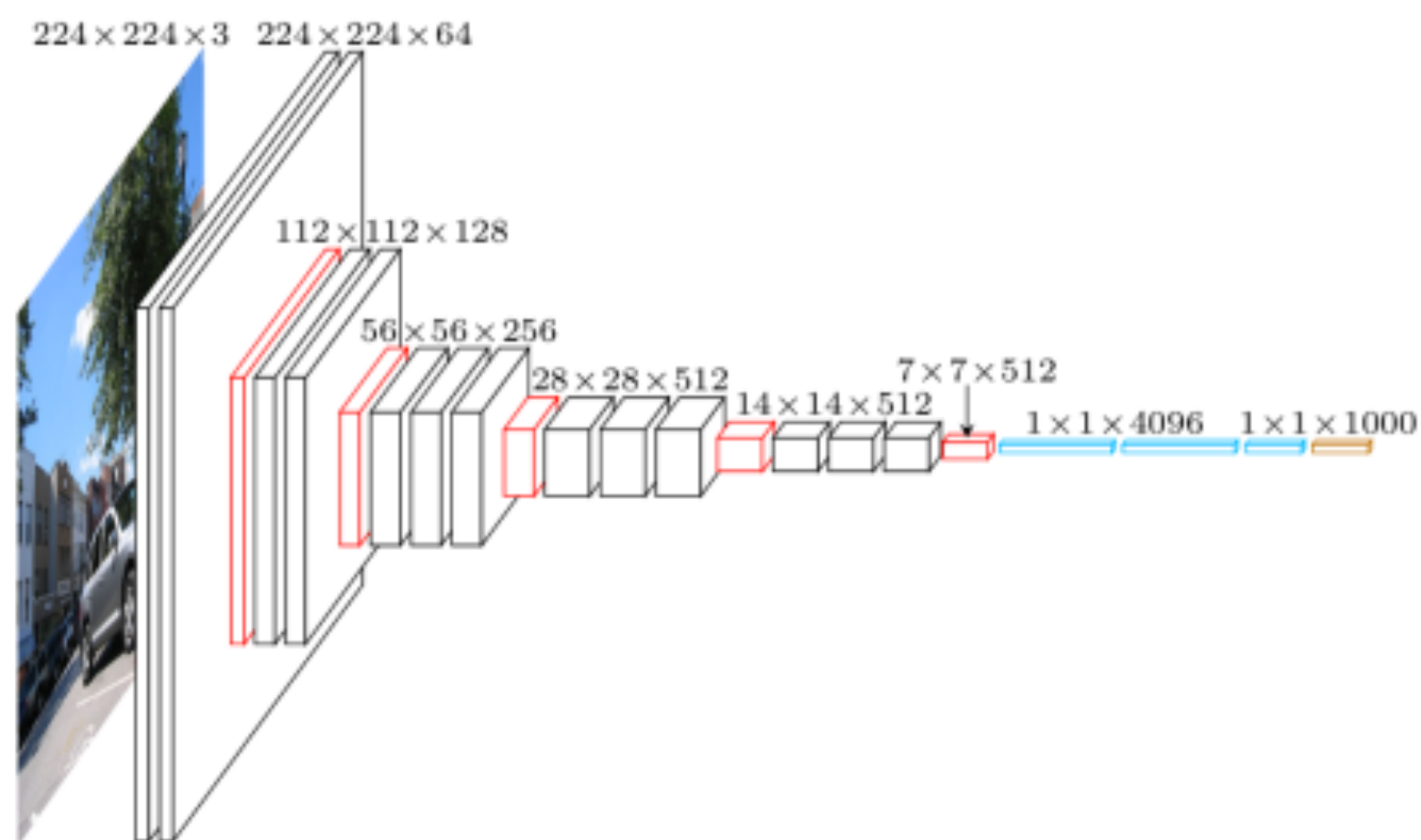
Model specialization

- **Common principle in DNN design/training is to learn most general model (via large datasets, regularization, etc.) to perform well across all instances of a task**
- **But many cameras see a very specific distribution of images**
 - **Only certain types of object classes**
 - **Always from the same/similar viewpoint**
 - **Objects appear in same regions of screen**
- **Specialization has been a major theme in this class w.r.t hardware design. Now we wish to specialize models to the contents of a video stream**
 - **“A model can be must simpler if it only needs to work for a single camera”**

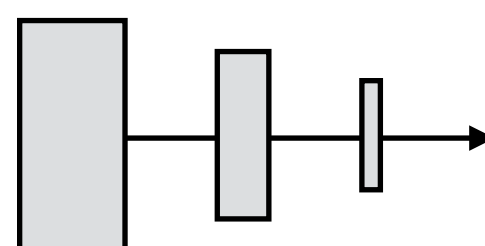
Model distillation

[Hinton 15]

- **Accurate, but expensive, model: trained on full training set**
 - **“The teacher”**



- **Smaller model (cheaper), trained to mimic the output of the teacher**
 - **“The student”**



Noscope

[Kang 17]

- Apply model distillation, but constrain training set to a specific video feed: Given an expensive network that performs a specified detection* task well on a wide range of videos, distill a highly optimized implementation for **this video stream**
- Example: binary classification task on a single class, in an traffic camera video stream



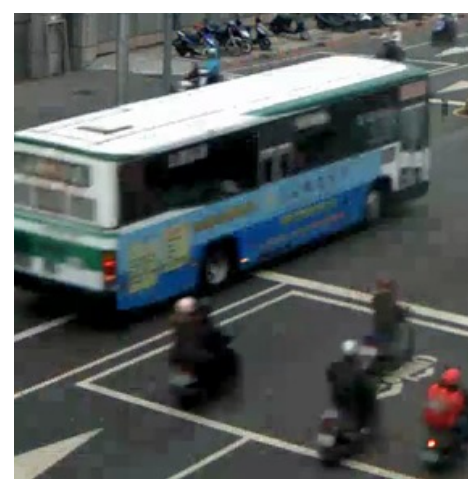
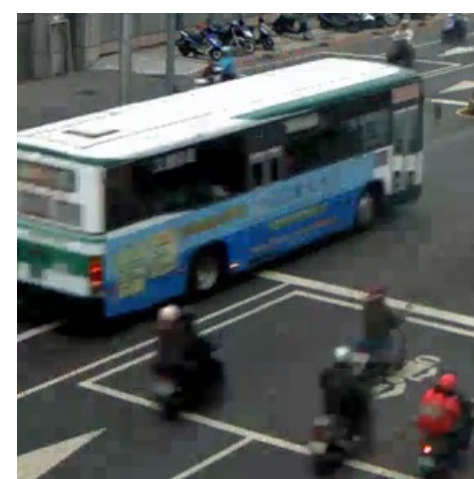
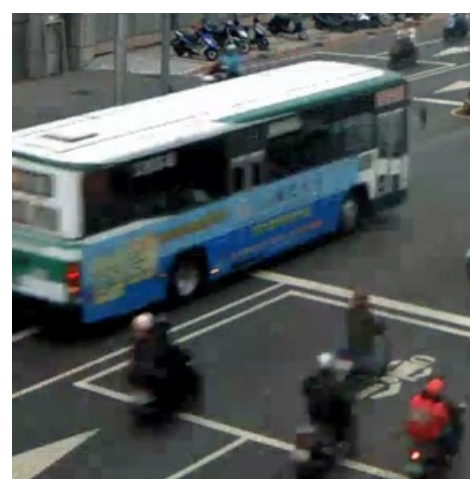
(a) empty frame



(b) frame with a car



(c) subtracted frames

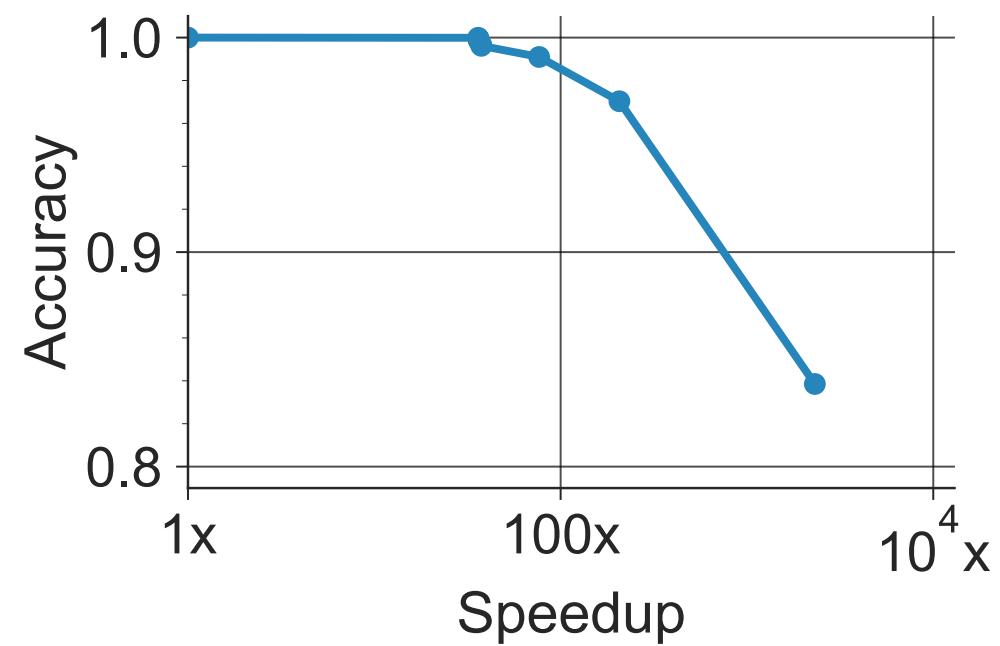


* Noscope actually performs a simpler classification task on a pre-cropped region of the viewport (not detection, which involves object location)

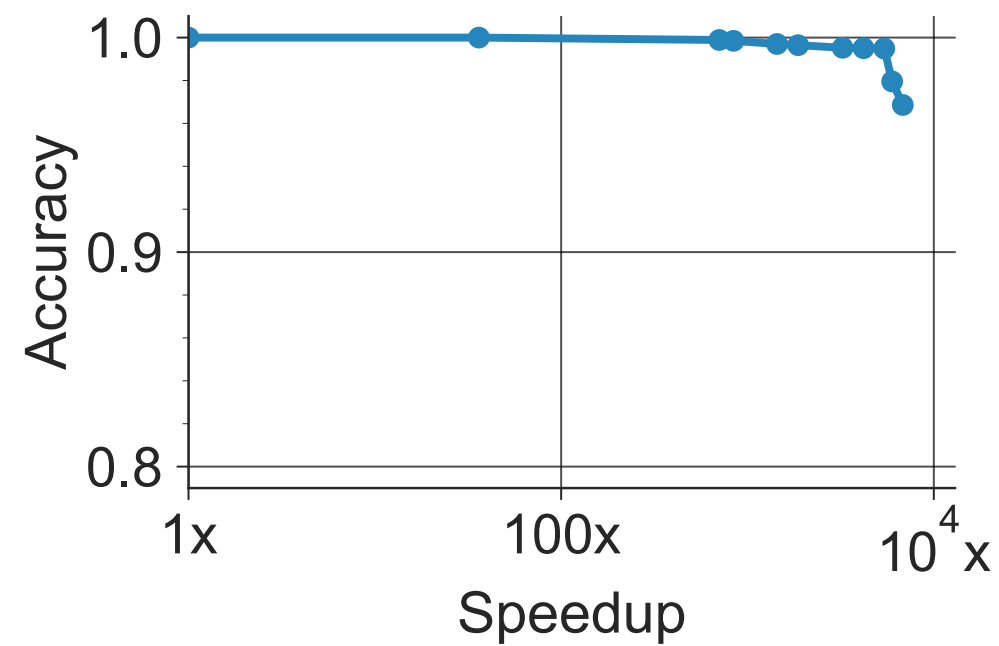
Three Noscope optimizations

- **Statically specialize model to video feed**
 - **Teacher network: Yolo object detection network**
 - **Student network: compact specialized network (2-4 conv layers)**
 - **Low cost student “learns” to mimic the teacher**
- **Dynamic: utilize frame-to-frame difference detectors with learned thresholds**
 - **“Same as background”, “same as previous frame”**
 - **Learn thresholds for how often to check for differences (in frames), and what the magnitude of a meaningful difference is**
- **Dynamic: cascades**
 - **Run cheap specialized model (student) on frame first, then run teacher model if student does not make a confident prediction**

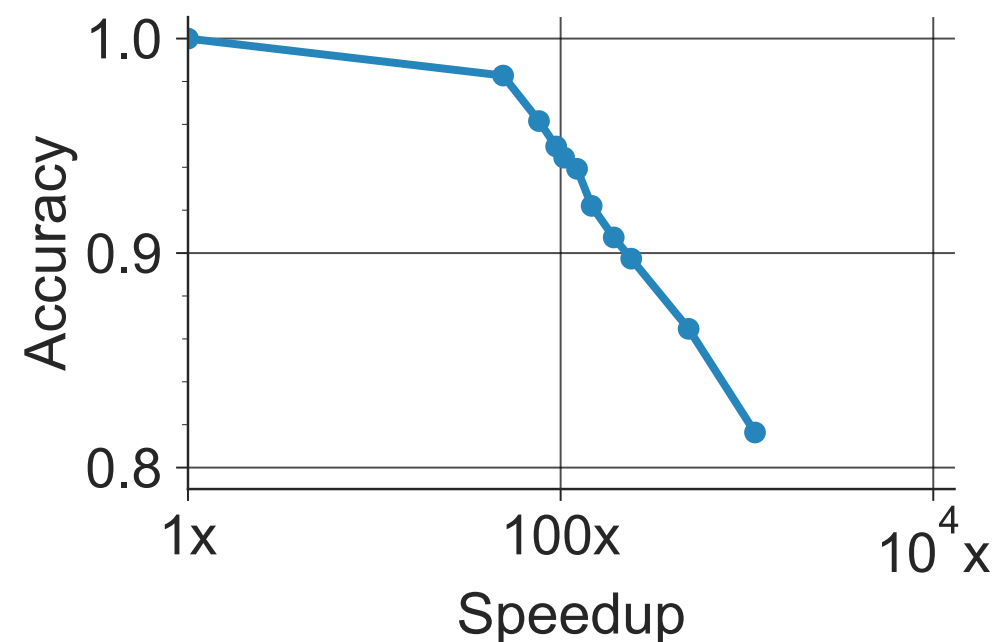
Noscope results *



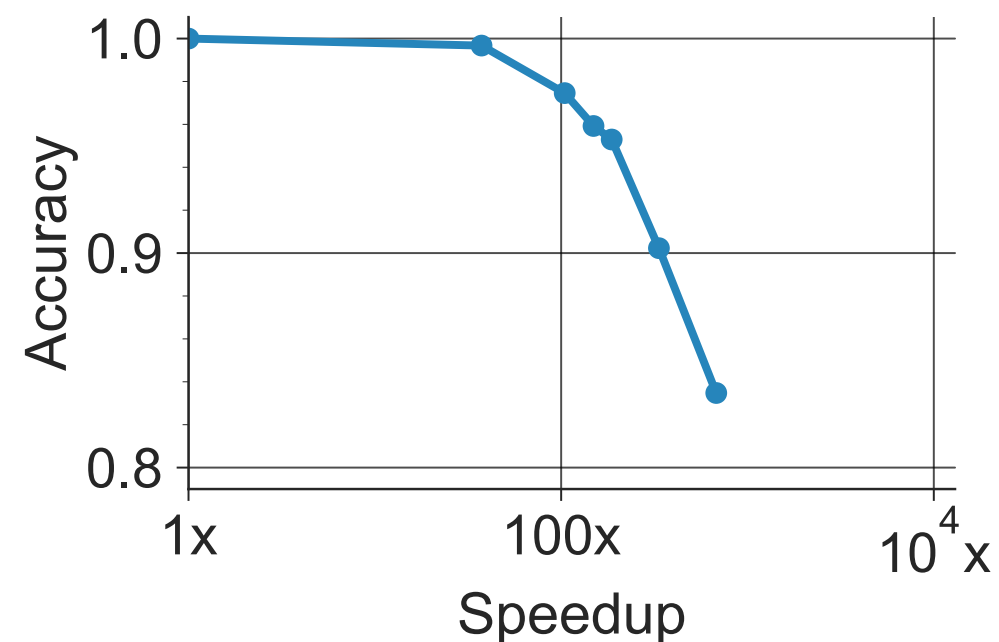
(a) taipei



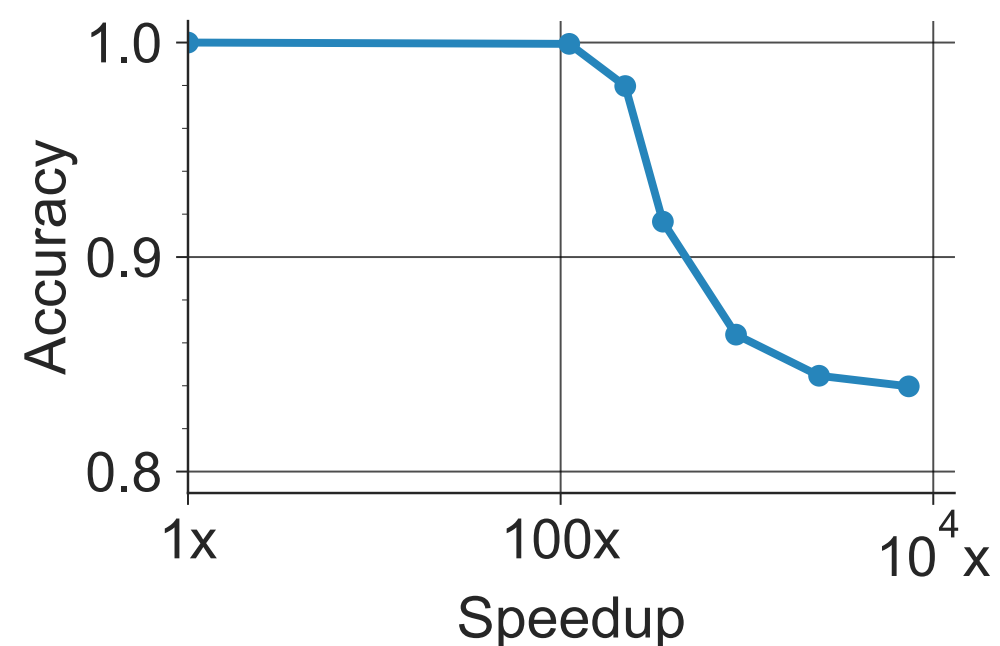
(b) coral



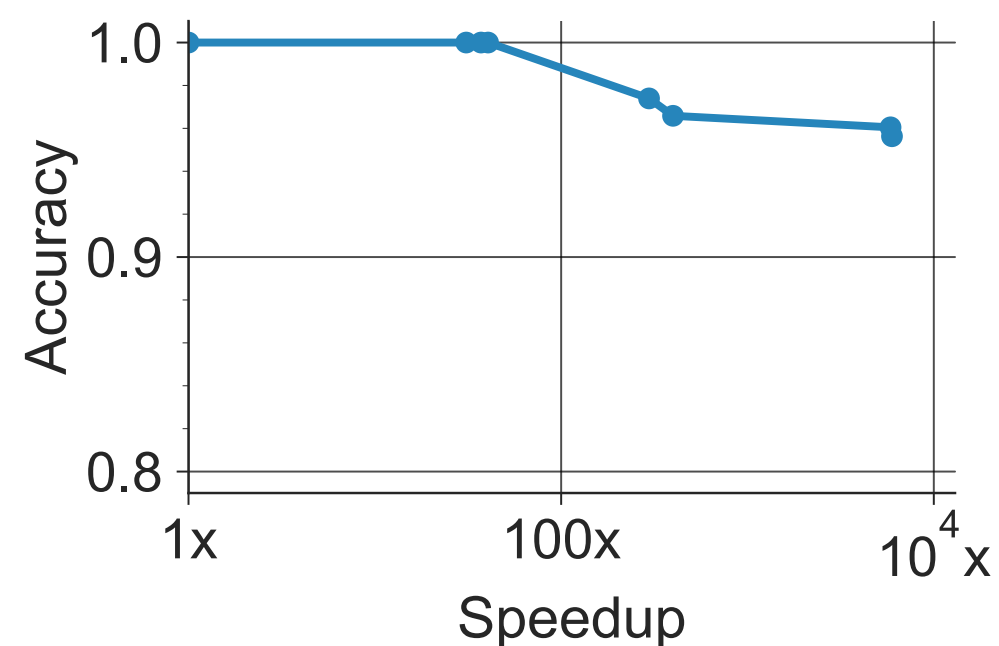
(c) amsterdam



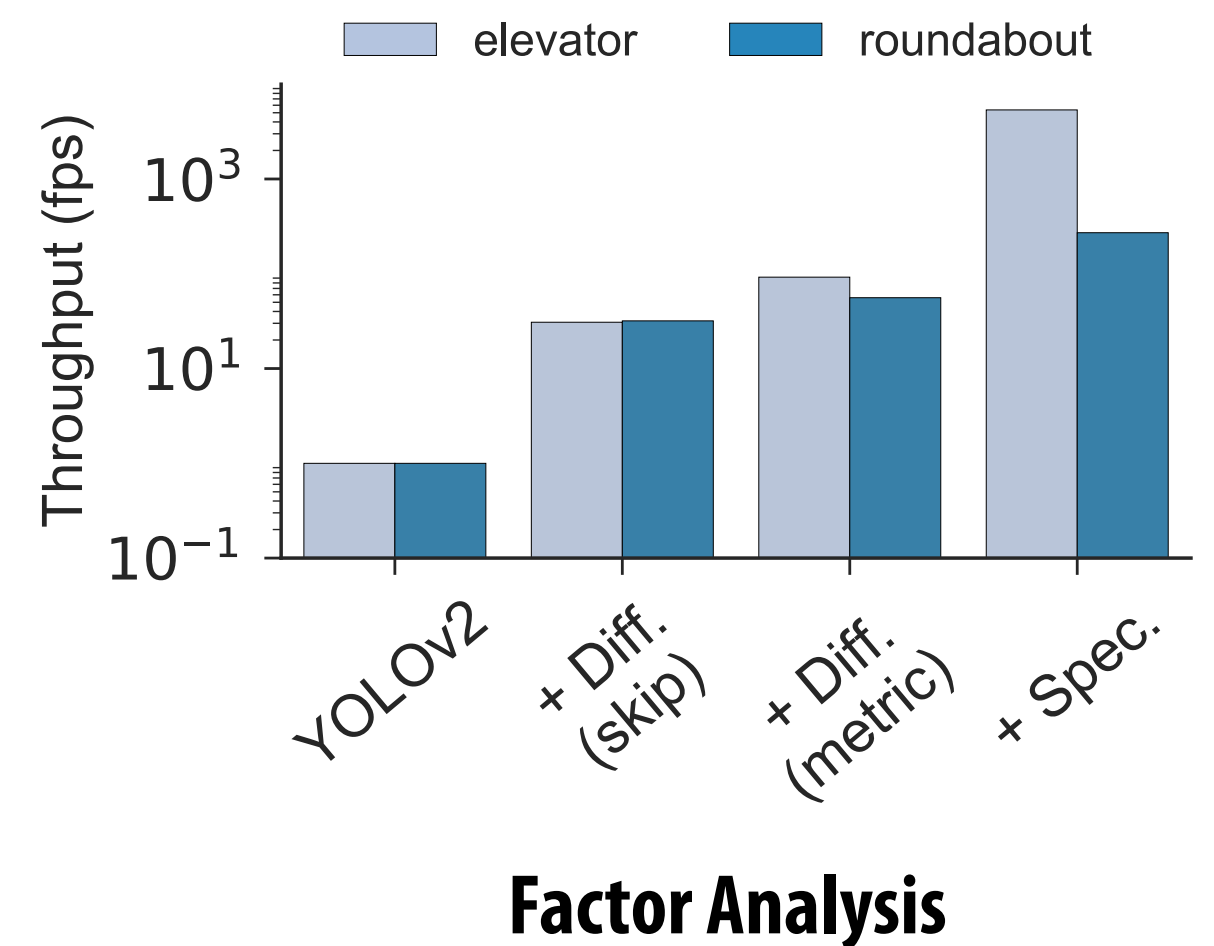
(d) night-street



(e) store



(f) elevator



Factor Analysis

* Noscope actually performs a simpler classification task on a pre-cropped region of the viewport (not detection, which involves object location)

More aggressive distillation

- Specialize to instant in time (requires constant model retraining)



**Specialized Model (specialized to scene,
camera viewpoint, and window of time)**

15 ms/frame

Expensive Model (Mask R-CNN)

250 ms/frame *

* Mask R-CNN is performing instance segmentation, specialized model is only performing segmentation (mask r-CNN is performing a slightly more complex task)

Class thought experiment



Click to open expanded view

AWS DeepLens - Deep learning enabled video camera for developers

[Amazon Web Services](#)

Price: **\$249.00** FREE Shipping for Prime members


This item will be released on June 14, 2018.

Pre-order now.


Ships from and sold by Amazon.com.

- Deep learning in your hands - A fully programmable video camera designed to expand deep learning skills.
- Start learning right away - Learn the basics of deep learning through example projects, computer vision models, tutorials, and real world, hands-on exploration on a physical device.
- A new way to learn machine learning - Allows developers of all skill levels get started with deep learning in less than 10 minutes.
- Sample projects - AWS DeepLens can run custom models from Amazon SageMaker, and comes with a collection of pre-trained models ready to run on the device with a single click.
- Fully programmable - Easy to customize and is fully programmable using AWS Lambda providing a familiar programming environment for developers to experiment with.
- Integrated with AWS - Stream video back to AWS using Amazon Kinesis Video Streams, and apply more advanced video analytics using Amazon Rekognition Video. The device also connects securely to AWS IoT, Amazon SQS, Amazon SNS, Amazon S3, Amazon DynamoDB, and more.

Share    

 Pre-order: Add to Cart


or 1-Click Checkout

 Pre-order with 1-Click

Not yet released

Free shipping once released

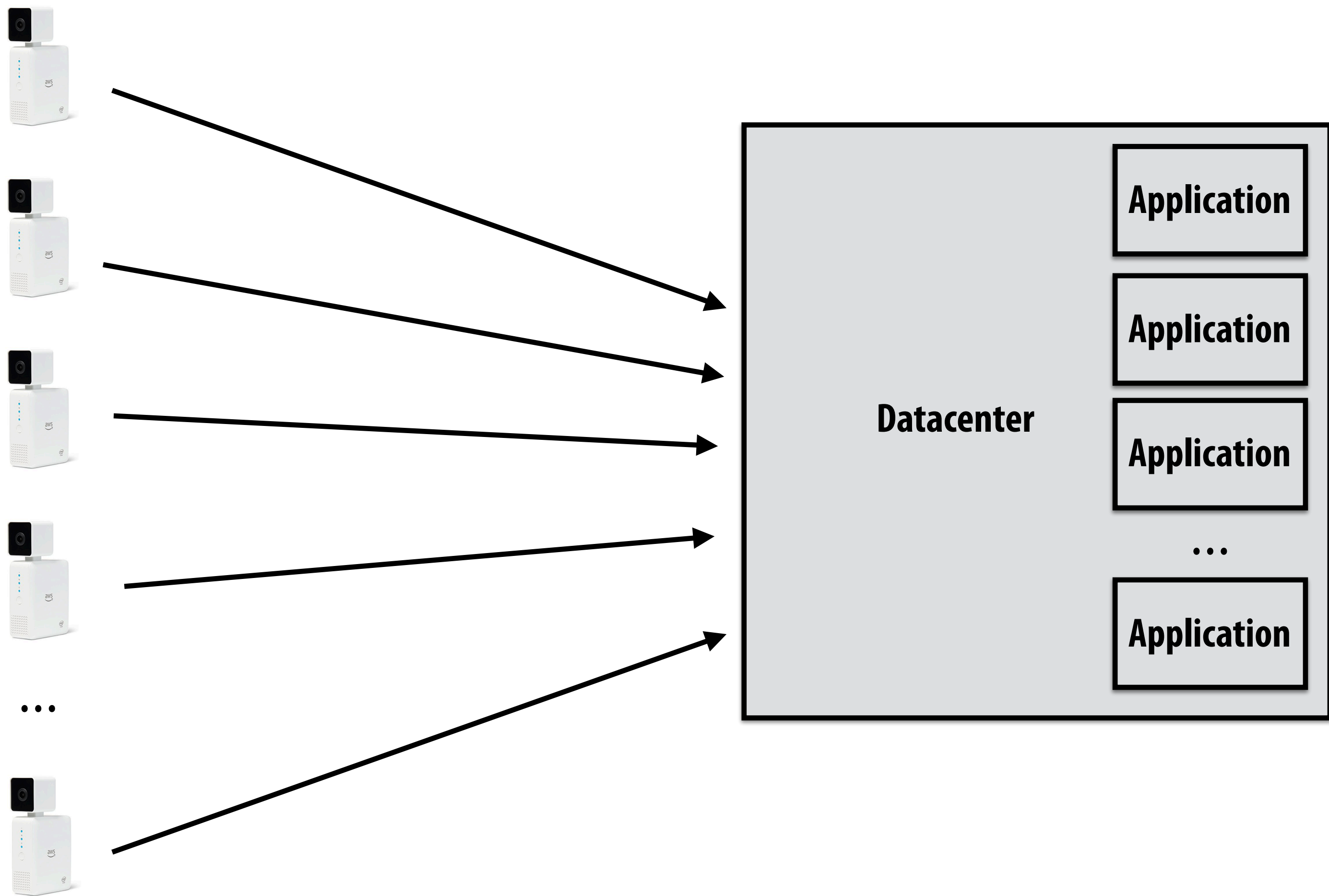
Ship to:

Kayvon Fatahalian- PALO ALTO 

Add to List

Setup

- Many cameras (e.g., traffic cameras in a major US city, cameras in future Amazon Go stores)
- Many applications needing access to streams for different tasks

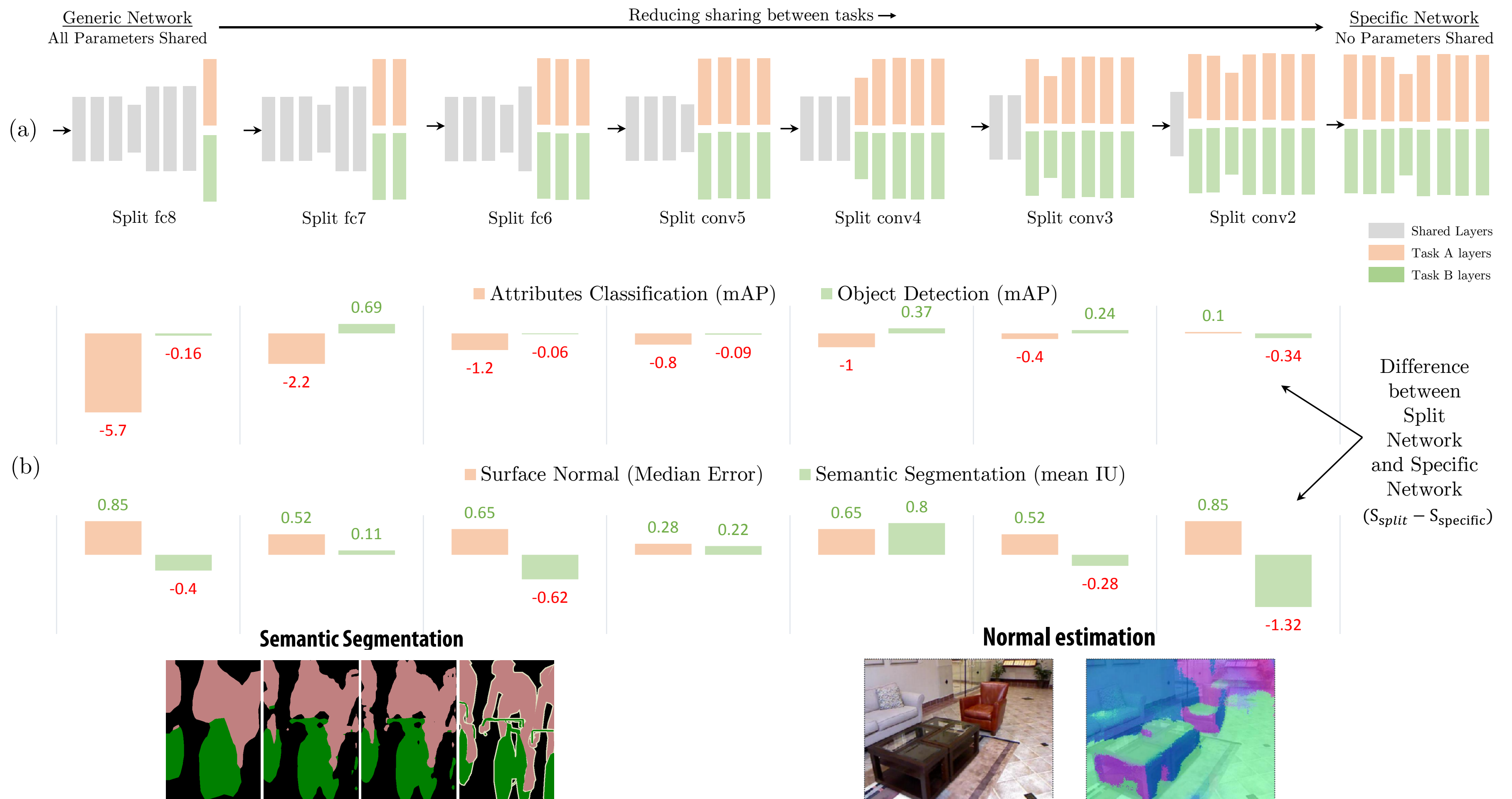


Trick 4: share processing across multiple tasks on the same stream

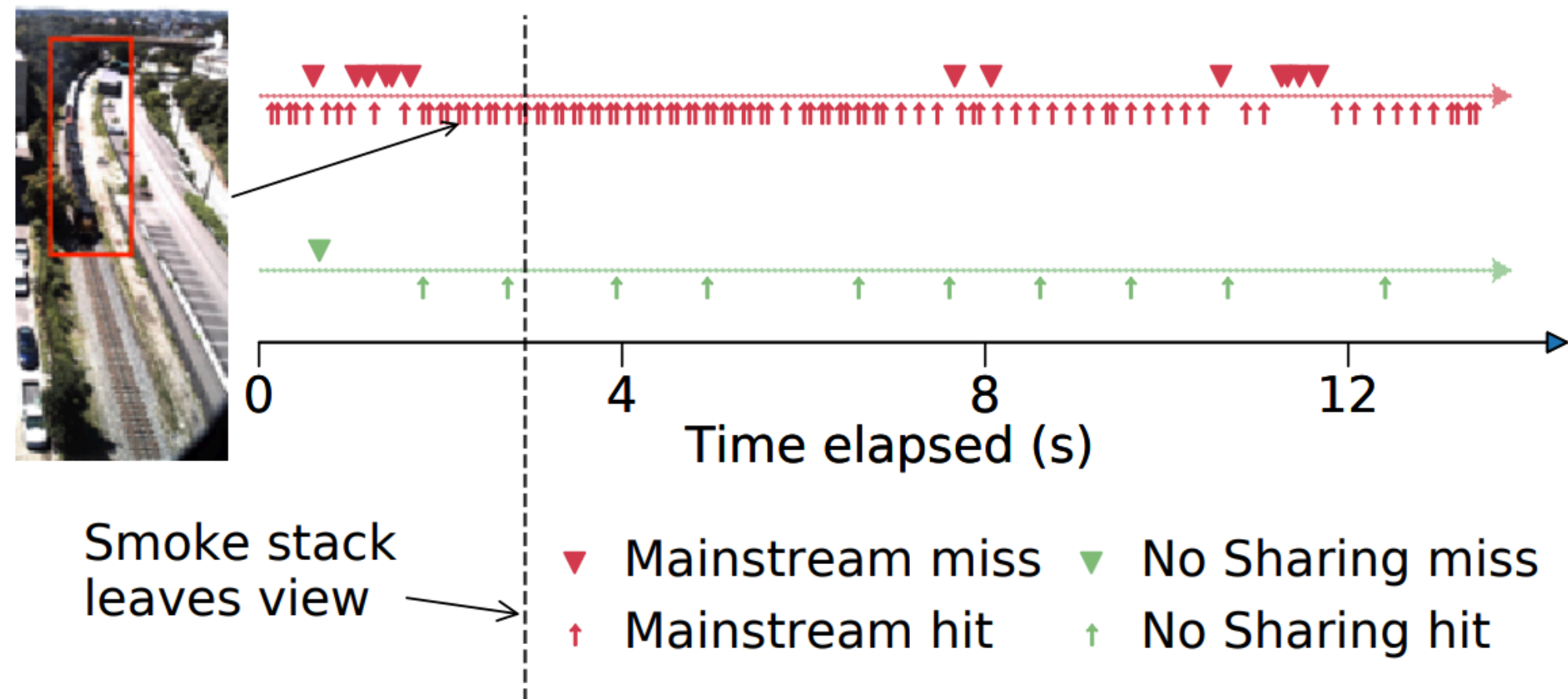
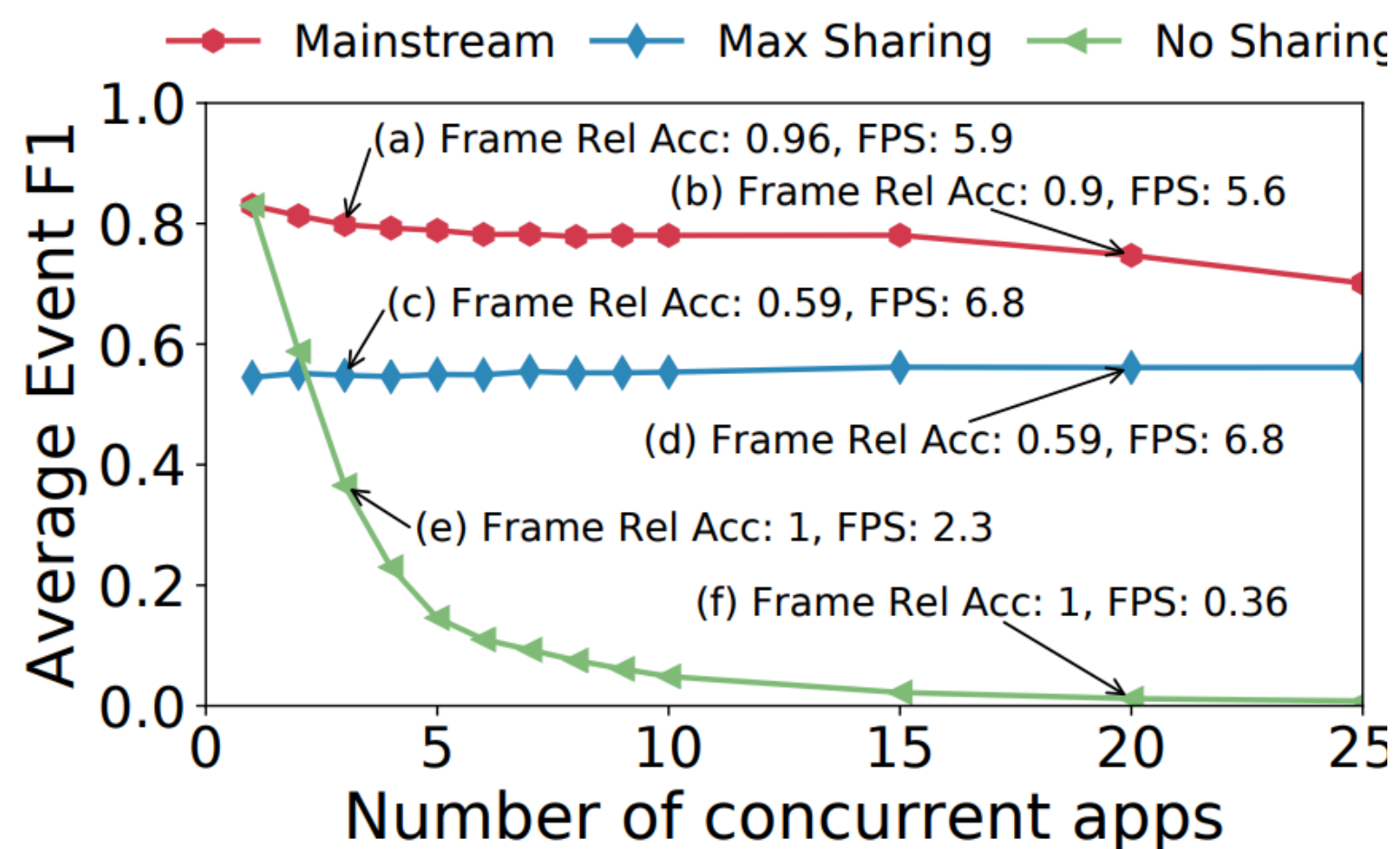
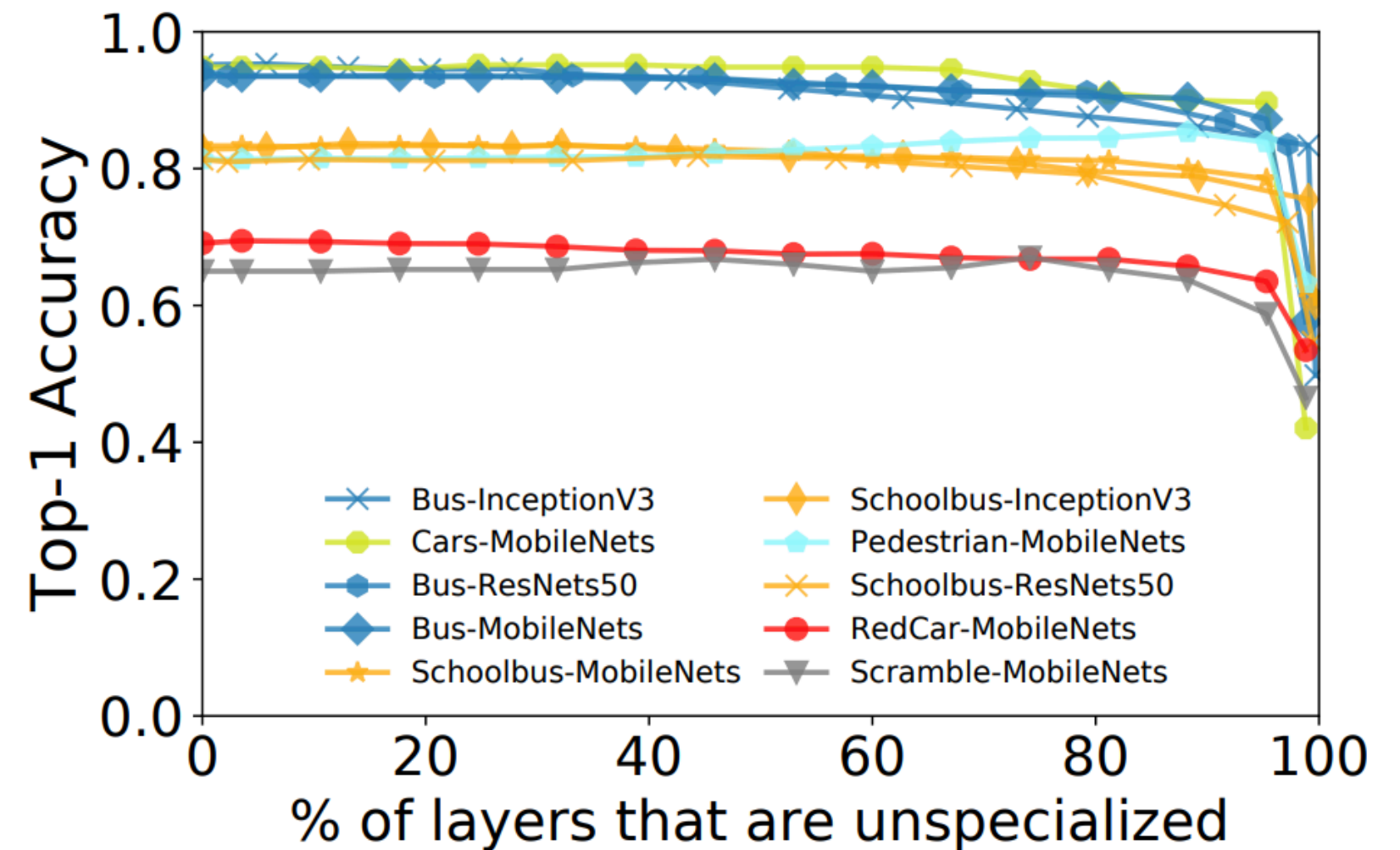
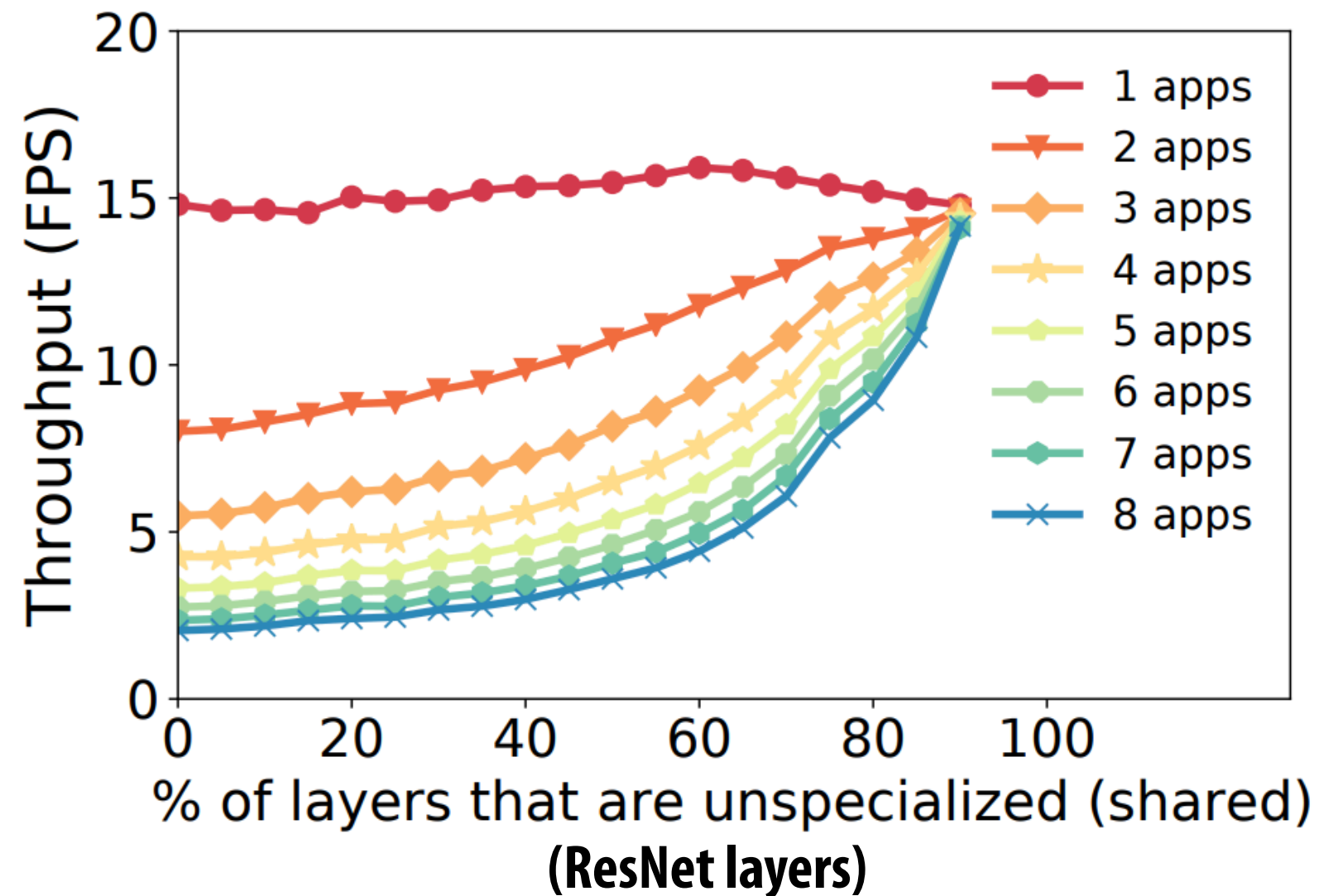
(Recall how sharing led to speedup across boxes in Mask R-CNN)

Sharing computation across tasks

- System design forces different networks to share the same trunk
- Amortize cost of early (and expensive layers) across different tasks
- Tough question: given N tasks, how much of DNN should be shared among tasks?



Emerging interest in systems that learn how to share computation across users



Less accurate per frame behavior can yield more confident event detection due to ability to sample event more times.

Summary

- **An increasing number of cameras across world will be capturing near continuous video**
- **Many applications will seek to extract value from these data streams**
 - **Implications for efficiency of cities (transportation, infrastructure monitoring), brick-and-mortar commerce, security, health-care, robotics, human-robot interactions, autonomous vehicles**
- **Need significant efficient gains to process this worldwide visual signal**
 - **Hardware specialization**
 - **Algorithmic techniques to reduce the cost of inference**
 - **Specialization to video stream or scene context**
 - **Exploit temporal coherence of video**

Discussion: privacy and ethics

Amazon's Rekognition messes up, matches 28 lawmakers to mugshots

ACLU: "And running the entire test cost us \$12.33—less than a large pizza."

CYRUS FARIVAR - 7/26/2018, 5:00 AM

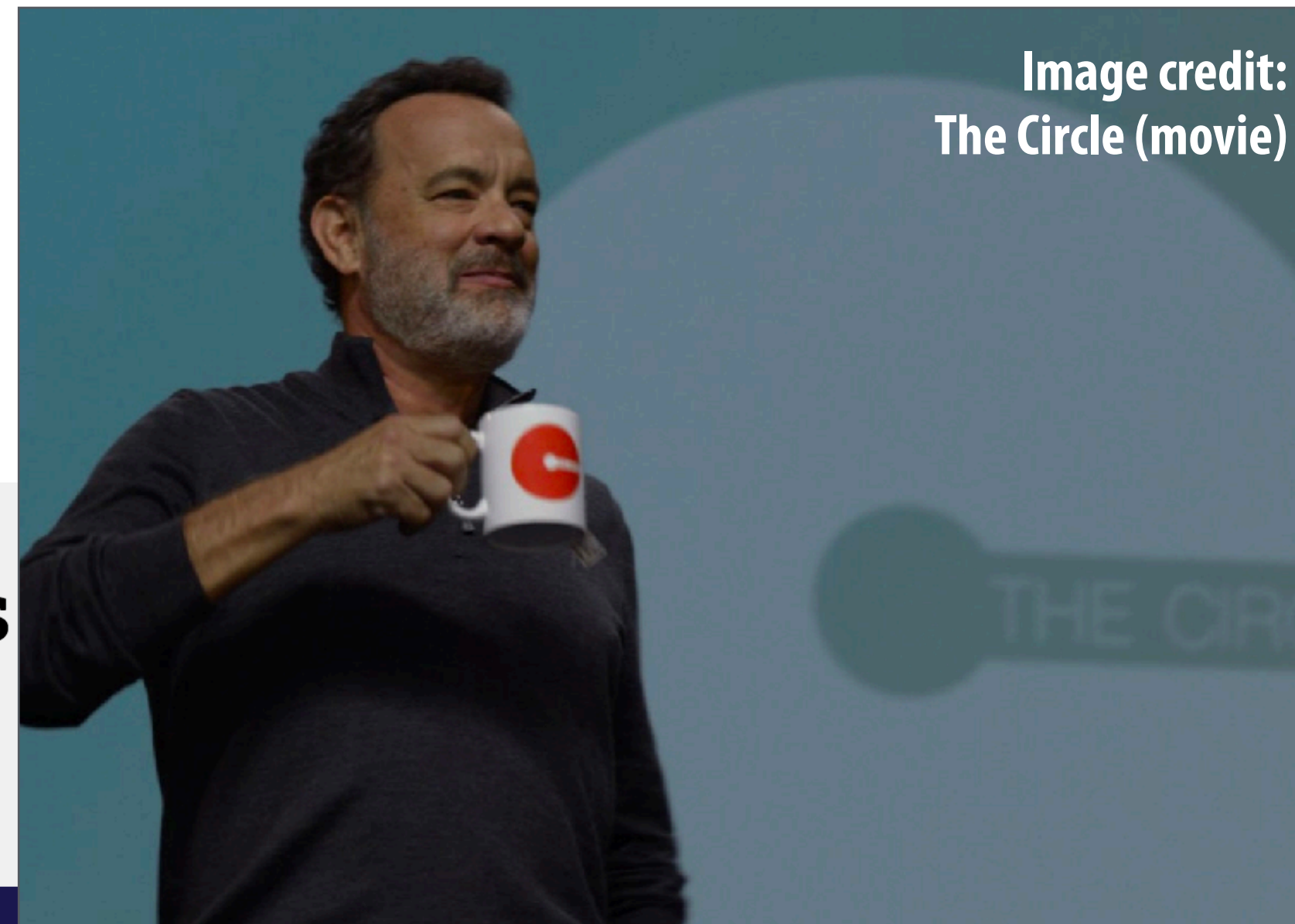


Image credit:
The Circle (movie)

Amazon Rekognition

FALSE MATCHES



28 current members of Congress

