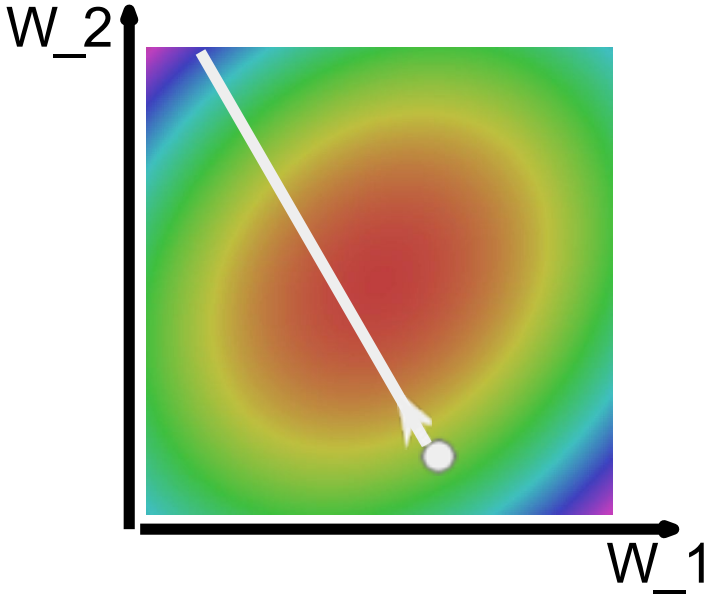
深度学习

第七讲

王胤

Optimization



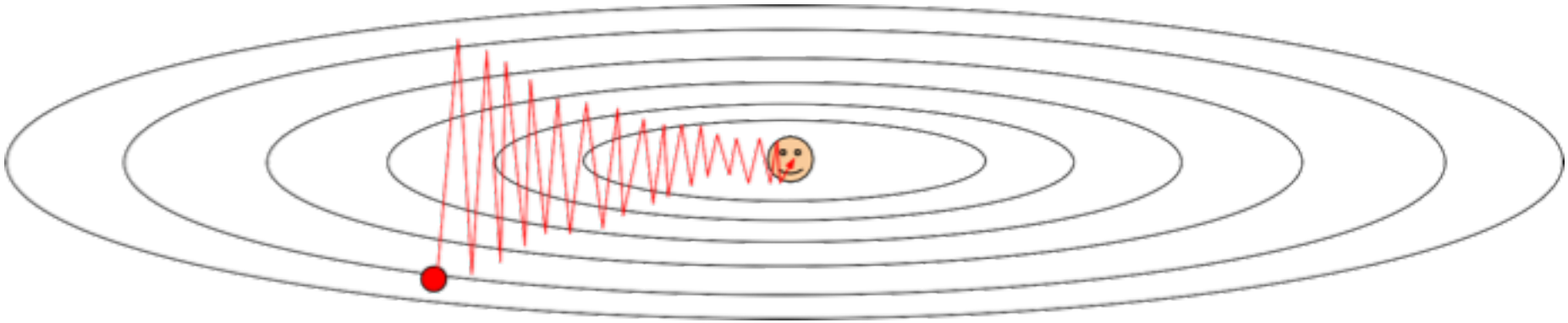

```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Optimization: Problems with SGD

What if loss changes quickly in one direction and slowly in another?

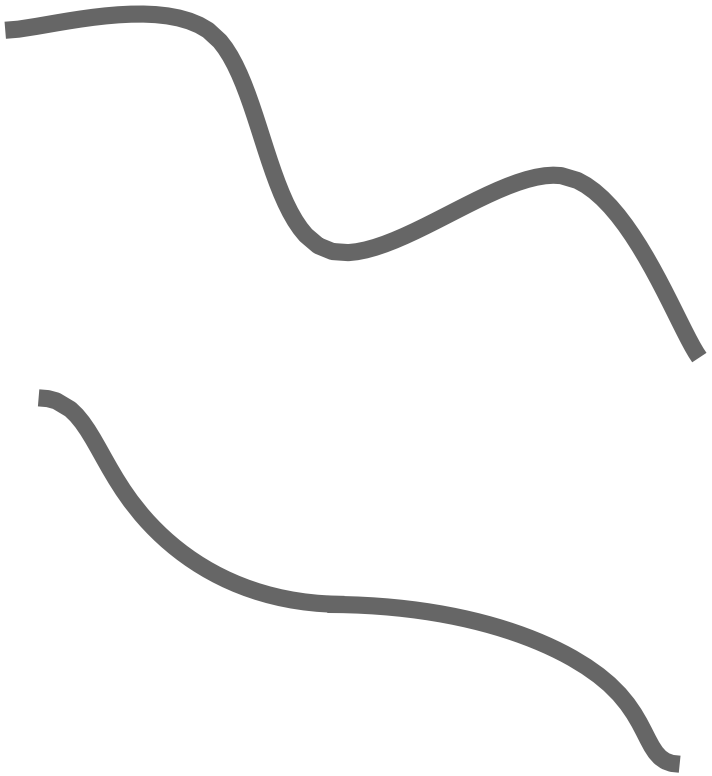
What does gradient descent do?

Very slow progress along shallow dimension, jitter along steep direction



Optimization: Problems with SGD

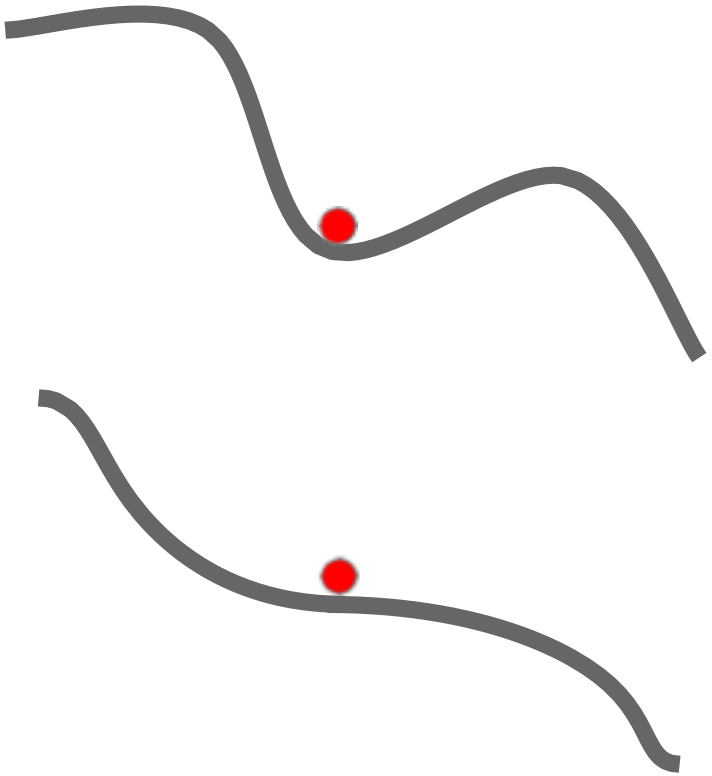
What if the loss function has a **local minima** or **saddle point**?



Optimization: Problems with SGD

What if the loss function has a **local minima** or **saddle point**?

Zero gradient, gradient descent gets stuck

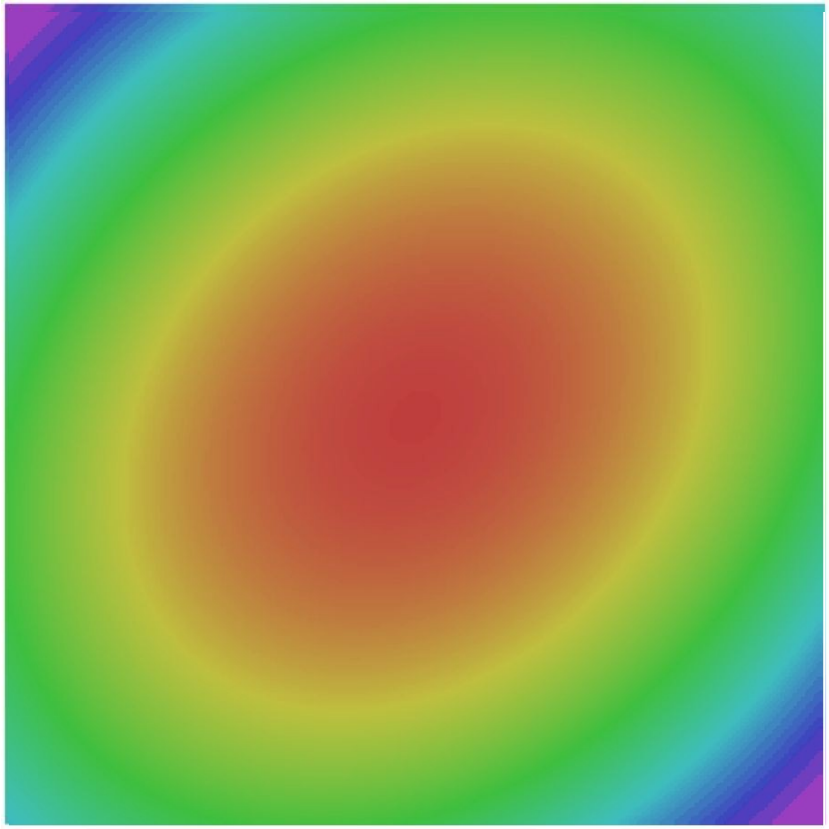


Optimization: Problems with SGD

Our gradients come from minibatches so they can be noisy!

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W)$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W)$$



SGD + Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
  dx = compute_gradient(x)
  x -= learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

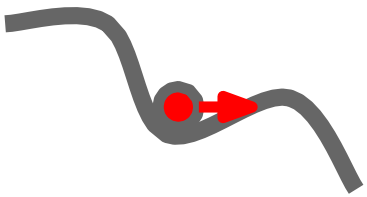
```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x -= learning_rate * vx
```

- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically rho=0.9 or 0.99

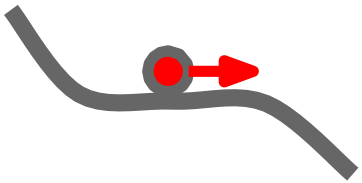
Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

SGD + Momentum

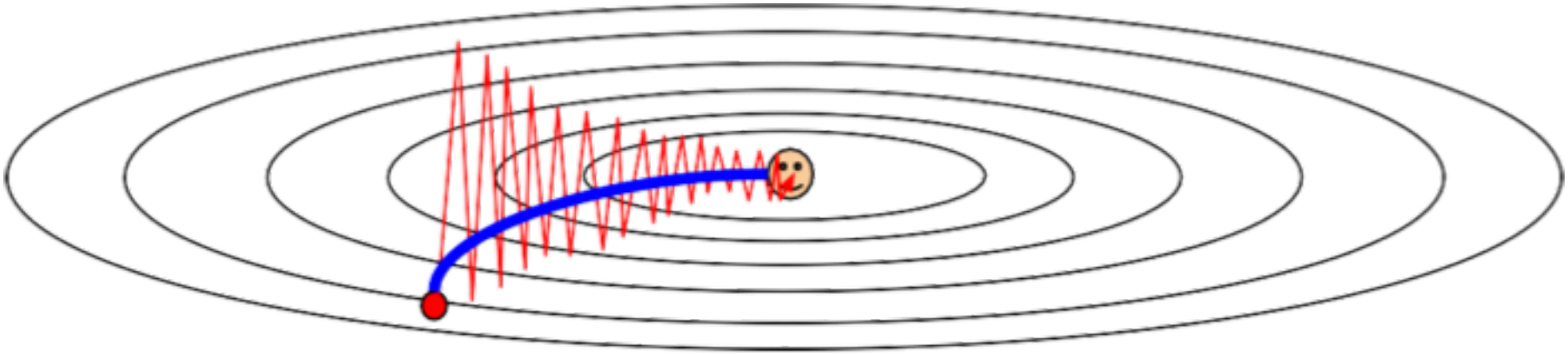
Local Minima



Saddle points

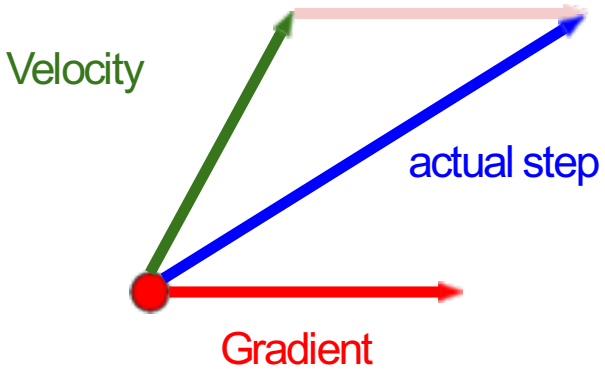


Poor Conditioning



Nesterov Momentum

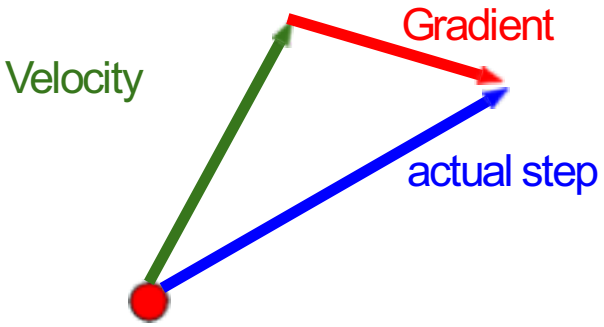
Momentum update:



Combine gradient at current point with velocity to get step used to update weights

Nesterov, "A method of solving a convex programming problem with convergence rate O(1/k^2)", 1983
Nesterov, "Introductory lectures on convex optimization: a basic course", 2004
Sutskever et al, "On the importance of initialization and momentum in deep learning", ICML 2013

Nesterov Momentum



"Look ahead" to the point where updating using velocity would take us; compute gradient there and mix it with velocity to get actual update direction

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

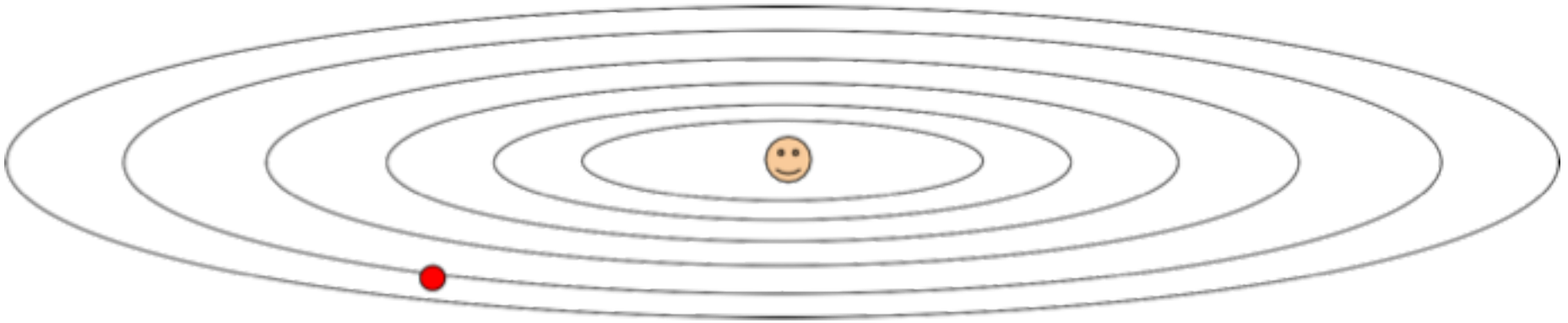
Added element-wise scaling of the gradient based on the historical sum of squares in each dimension

"Per-parameter learning rates"
or "adaptive learning rates"

Duchi et al, "Adaptive subgradient methods for online learning and stochastic optimization", JMLR 2011

AdaGrad

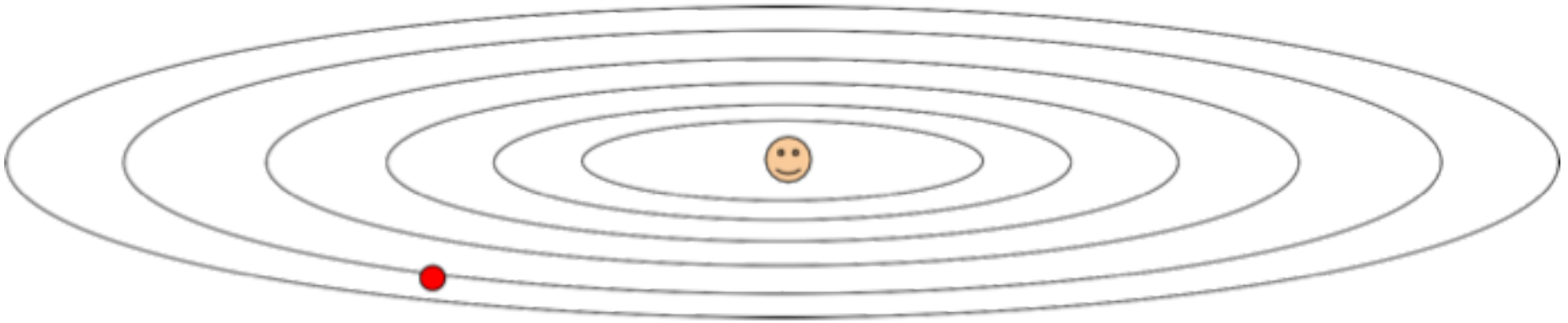
```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



Q: What happens with AdaGrad?

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



Q2: What happens to the step size over long time?

RMSProp

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

RMSProp

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Tieleman and Hinton, 2012

Adam

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

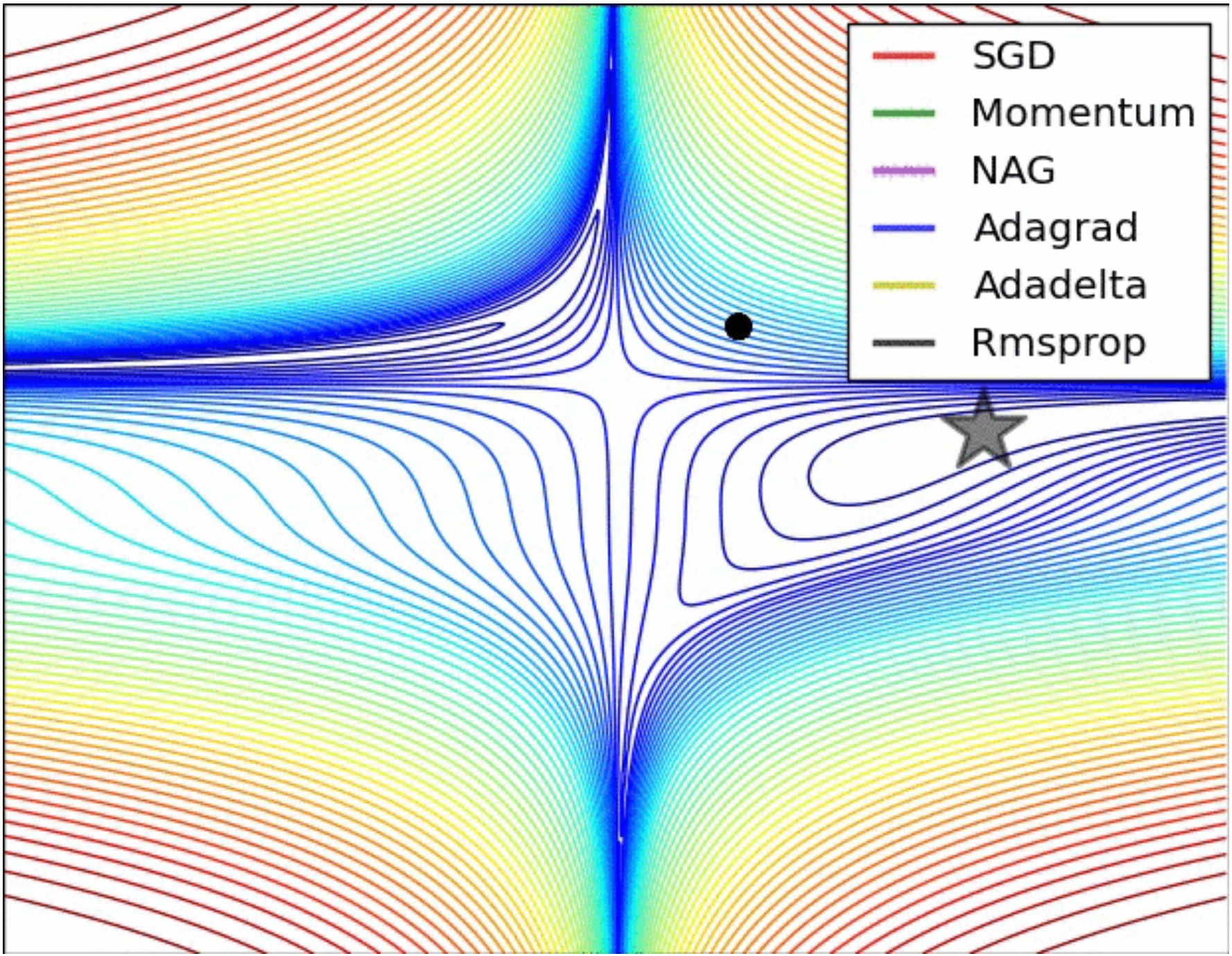
AdaGrad / RMSProp

Bias correction for the fact that first and second moment estimates start at zero

Adam with beta1 = 0.9,
beta2 = 0.999, and learning_rate = 1e-3 or 5e-4
is a great starting point for many models!

Kingma and Ba, “Adam: A method for stochastic optimization”, ICLR 2015

Optimization



In practice:

- **Adam** is a good default choice in many cases
- **SGD+Momentum** with learning rate decay often outperforms Adam by a bit, but requires more tuning

- If you can afford to do full batch updates then try out **L-BFGS** (and don't forget to disable all sources of noise)

Transfer Learning

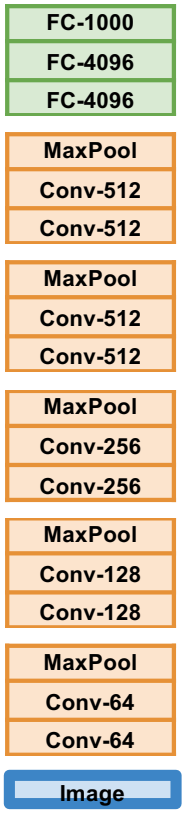
"You need a lot of a data if you want to train/use CNNs"

BUSTED

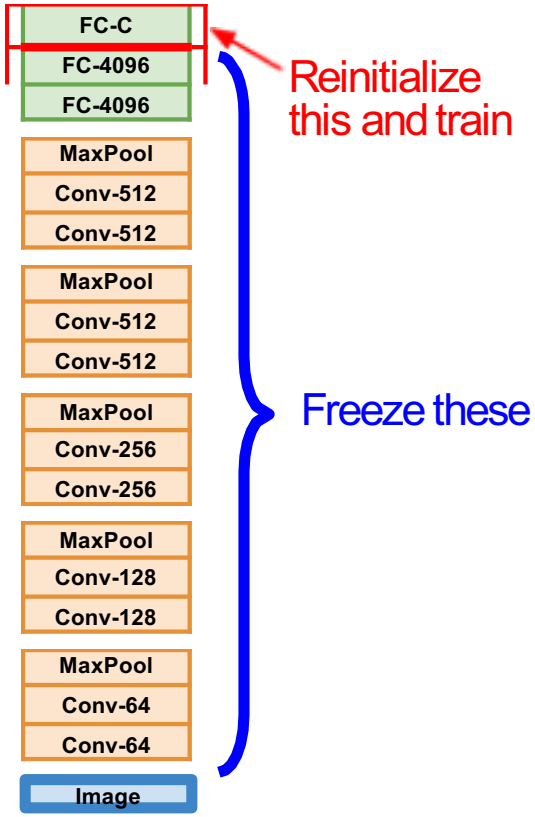
Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

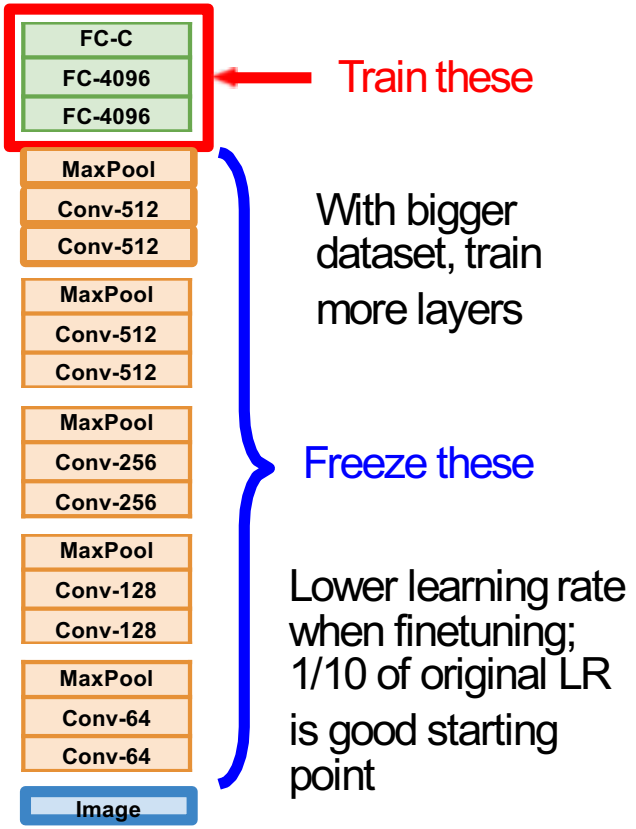
1. Train on Imagenet

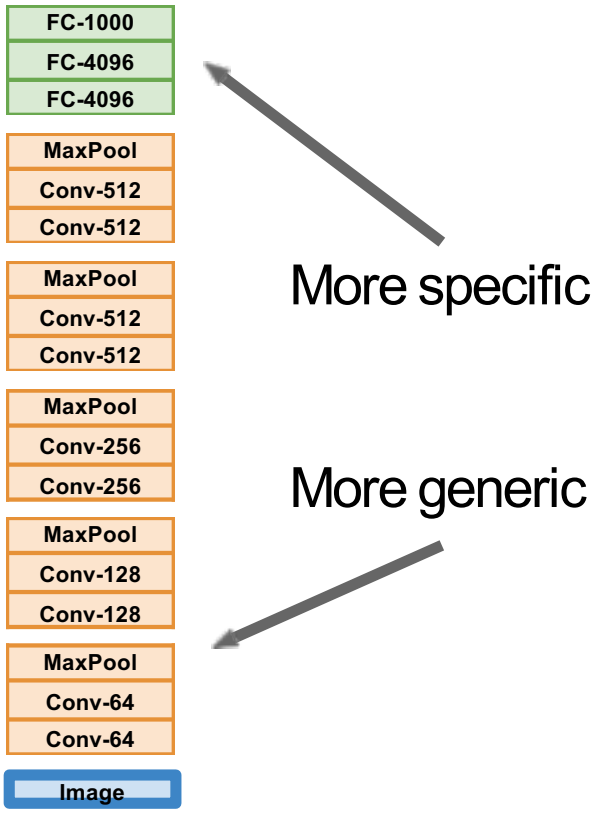


2. Small Dataset (C classes)



3. Bigger dataset





	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

Exploring the Limits of Weakly Supervised Pretraining

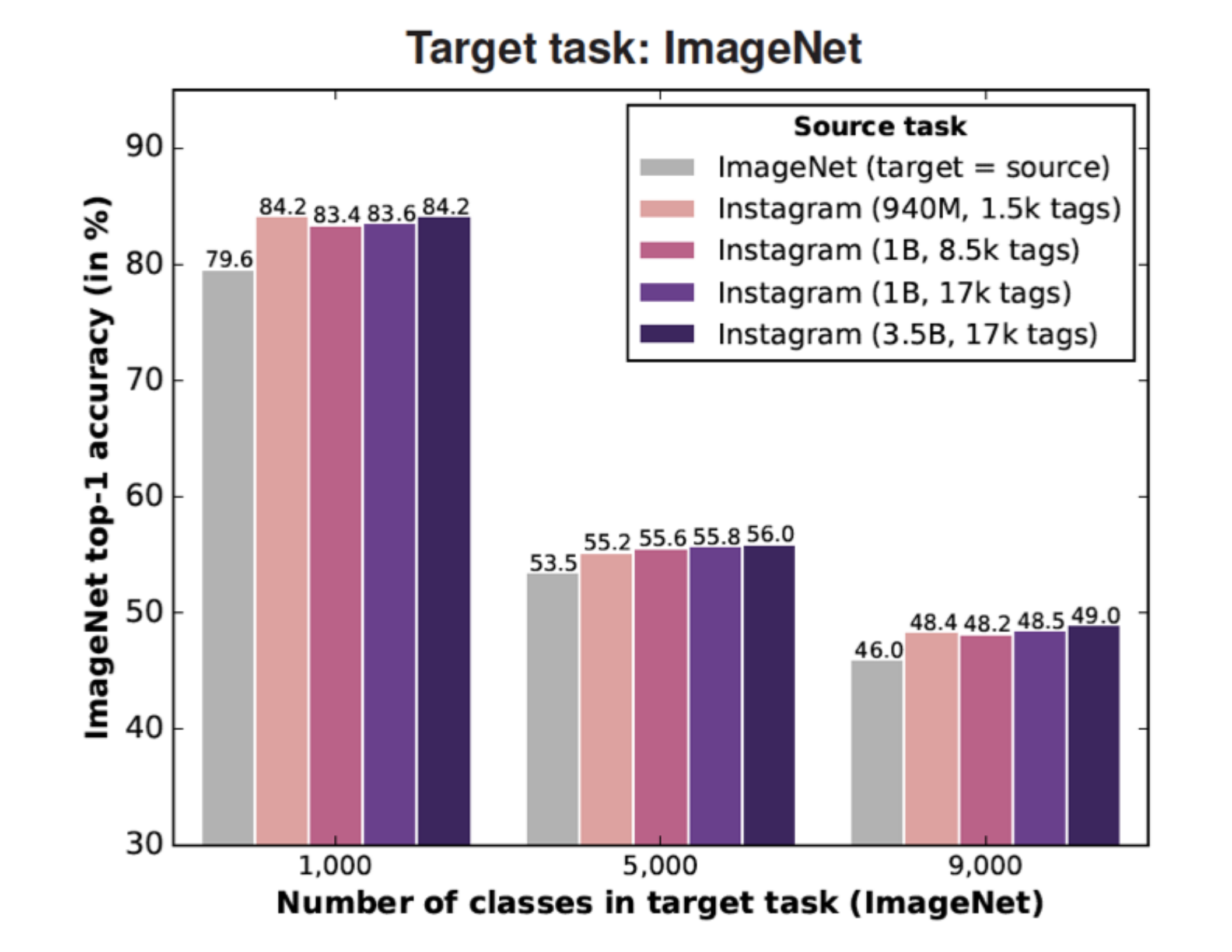
Dhruv Mahajan Ross Girshick Vignesh Ramanathan Kaiming He
Manohar Paluri Yixuan Li Ashwin Bharambe Laurens van der Maaten

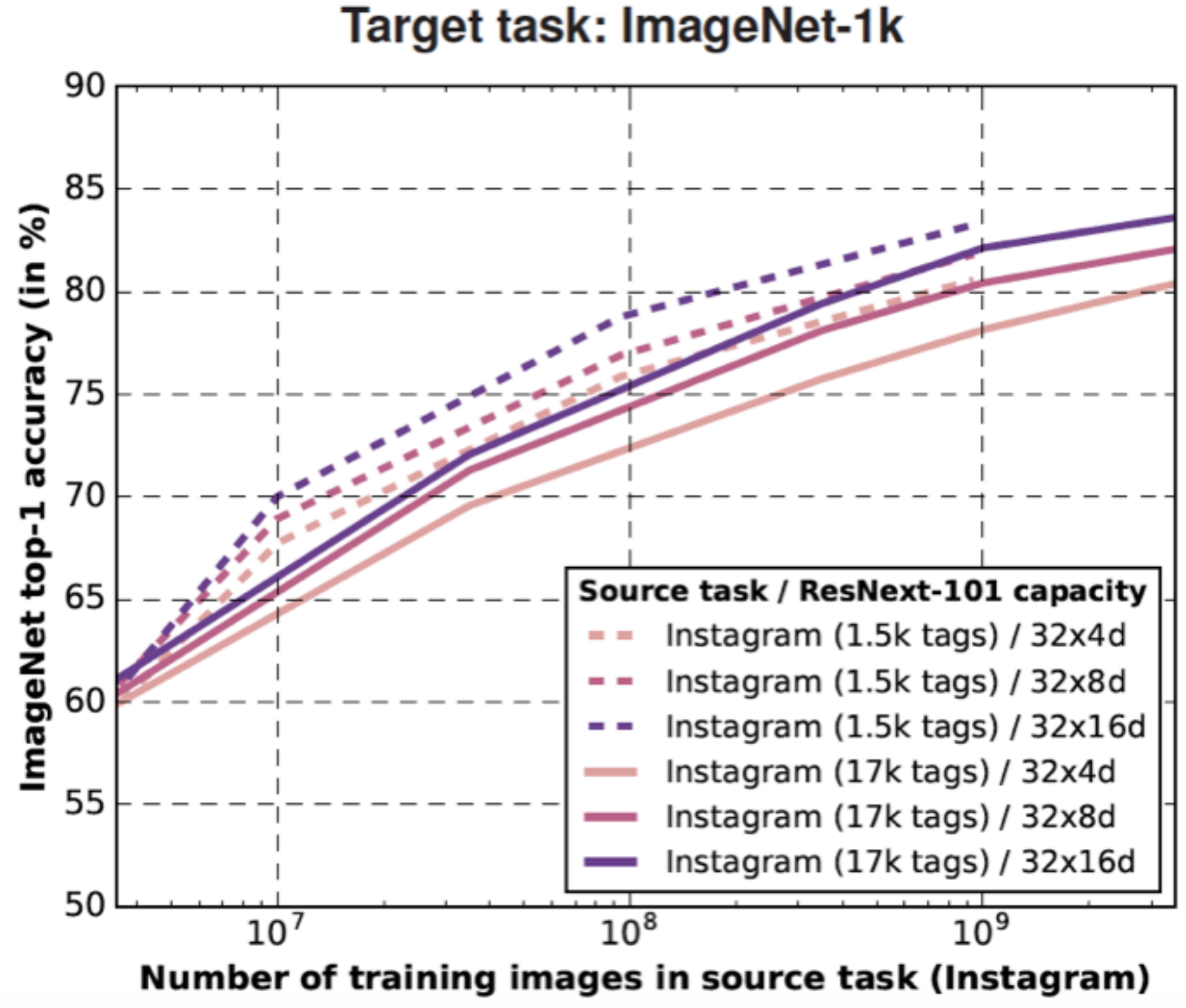
Facebook

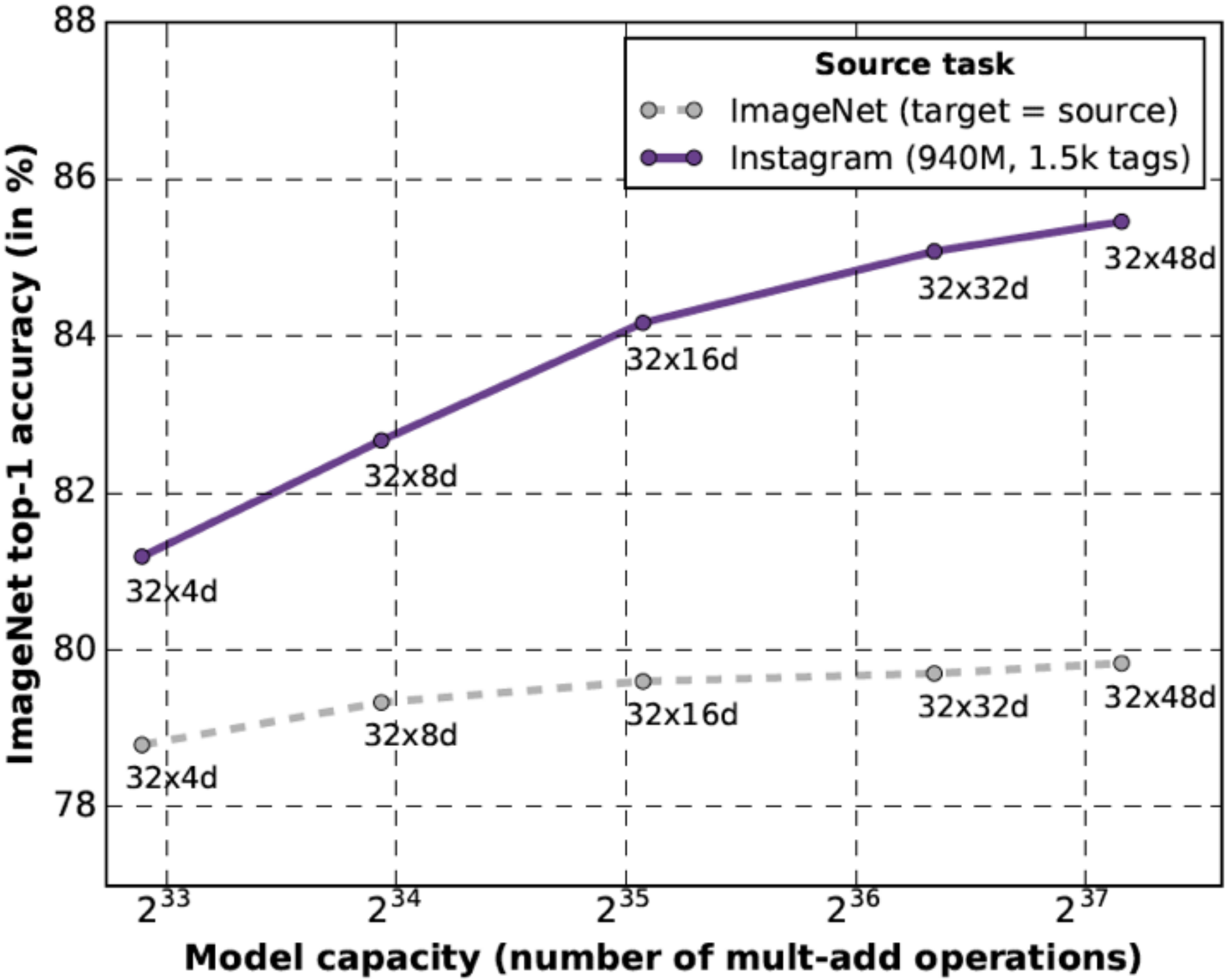
Abstract. State-of-the-art visual perception models for a wide range of tasks rely on supervised pretraining. ImageNet classification is the de facto pretraining task for these models. Yet, ImageNet is now nearly ten years old and is by modern standards "small". Even so, relatively little is known about the behavior of pretraining with datasets that are multiple orders of magnitude larger. The reasons are obvious: such datasets are difficult to collect and annotate. In this paper, we present a unique study of transfer learning with large convolutional networks trained to predict hashtags on *billions* of social media images. Our experiments demonstrate that training for large-scale hashtag prediction leads to excellent results. We show improvements on several image classification and object

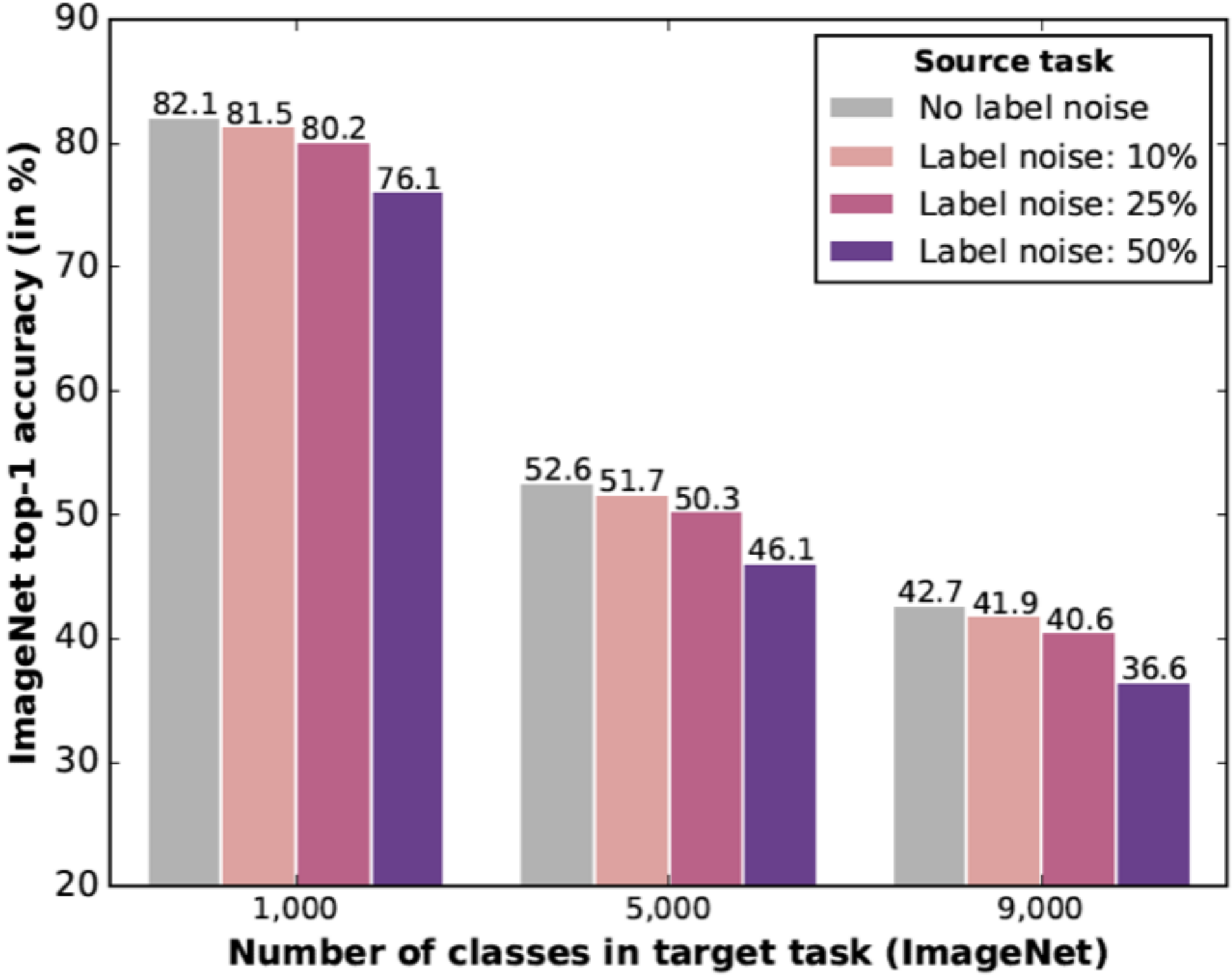
A 18 les gusta

philgonzalez Last Sunset at the Sea
philgonzalez #andaman #sea #Pukhet #Krabi #beaches #Thailand #Asia #sunset #boats #holidays #trips


Target task: ImageNet
Source task
ImageNet (target = source)
Instagram (940M, 1.5k tags)
Instagram (1B, 8.5k tags)
Instagram (1B, 17k tags)
Instagram (3.5B, 17k tags)
ImageNet top-1 accuracy (in %)
90
80
70
60
50
40
30
79.6
84.2
83.4
83.6
84.2
53.5
55.2
55.6
55.8
56.0
46.0
48.4
48.2
48.5
49.0
1,000
5,000
9,000
Number of classes in target task (ImageNet)

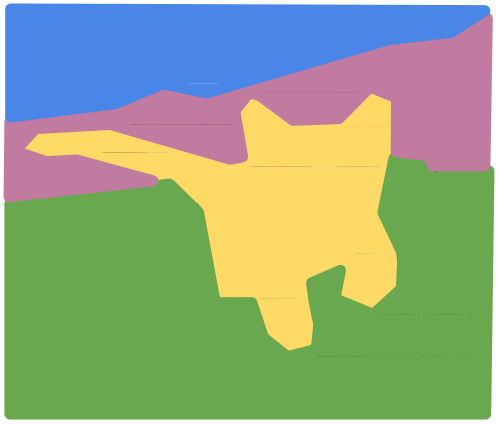

Target task: ImageNet-1k
ImageNet top-1 accuracy (in %)
90
85
80
75
70
65
60
55
50
10^7
10^8
10^9
Number of training images in source task (Instagram)
Source task / ResNext-101 capacity
Instagram (1.5k tags) / 32x4d
Instagram (1.5k tags) / 32x8d
Instagram (1.5k tags) / 32x16d
Instagram (17k tags) / 32x4d
Instagram (17k tags) / 32x8d
Instagram (17k tags) / 32x16d


Source task
ImageNet (target = source)
Instagram (940M, 1.5k tags)
ImageNet top-1 accuracy (in %)
88
86
84
82
80
78
32x4d
32x8d
32x16d
32x32d
32x48d
32x4d
32x8d
32x16d
32x32d
32x48d
2^{33}
2^{34}
2^{35}
2^{36}
2^{37}
Model capacity (number of mult-add operations)


Source task
No label noise
Label noise: 10%
Label noise: 25%
Label noise: 50%
82.1
81.5
80.2
76.1
52.6
51.7
50.3
46.1
42.7
41.9
40.6
36.6
90
80
70
60
50
40
30
20
1,000
5,000
9,000
ImageNet top-1 accuracy (in %)
Number of classes in target task (ImageNet)

Other Computer Vision Tasks

Semantic Segmentation



GRASS, CAT, TREE, SKY

No objects, just pixels

Classification + Localization

CAT

Single Object

Object Detection

DOG, DOG, CAT

Instance Segmentation

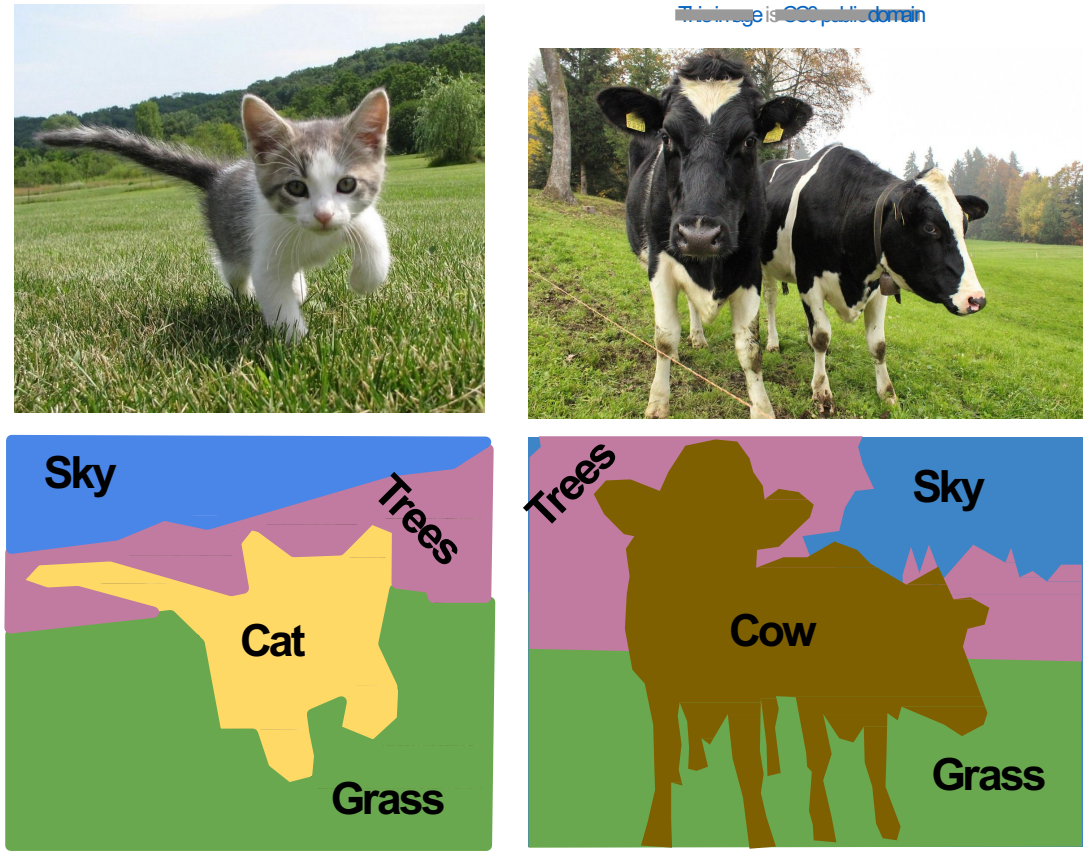
DOG, DOG, CAT

Multiple Object

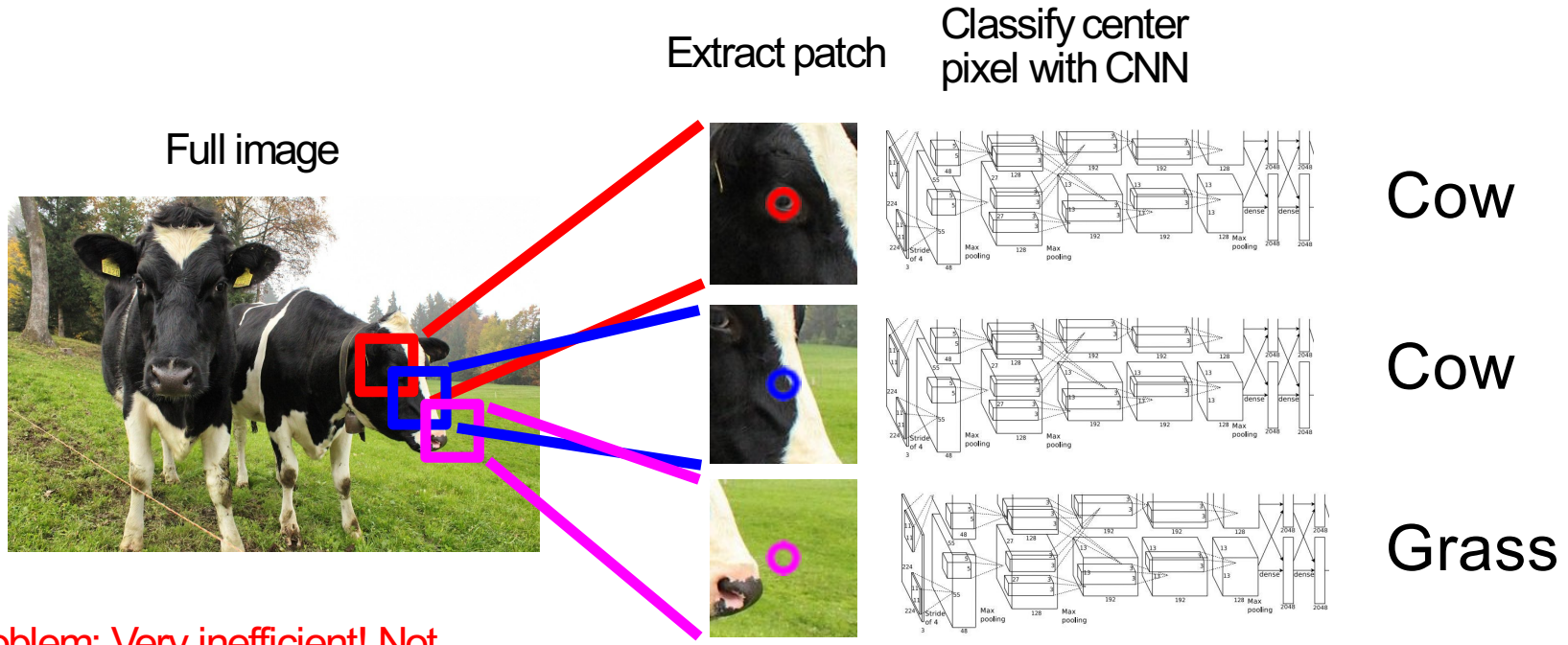
Semantic Segmentation

Label each pixel in the image with a category label

Don’t differentiate instances, only care about pixels



Semantic Segmentation Idea: Sliding Window



Problem: Very inefficient! Not reusing shared features between overlapping patches

Farabet et al, “Learning Hierarchical Features for Scene Labeling,” TPAMI 2013
Pinheiro and Collobert, “Recurrent Convolutional Neural Networks for Scene Labeling”, ICML 2014

Semantic Segmentation Idea: Fully Convolutional

Design a network as a bunch of convolutional layers to make predictions for pixels all at once!

Input:
3 x H x W

Conv

Conv

Conv

Convolutions:
D x H x W

Problem: convolutions at original image resolution will be very expensive ...

Conv

Scores:
C x H x W

argmax

Predictions:
H x W

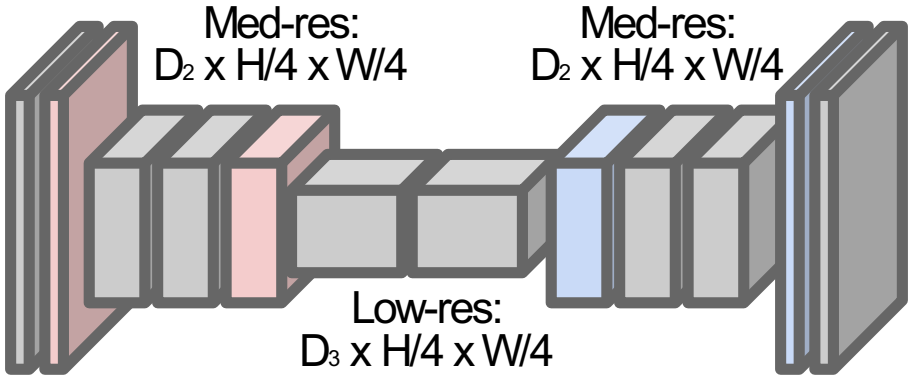
Semantic Segmentation Idea: Fully Convolutional

Downsampling: Pooling, strided convolution

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Upsampling: ???

Input:
3 x H x W



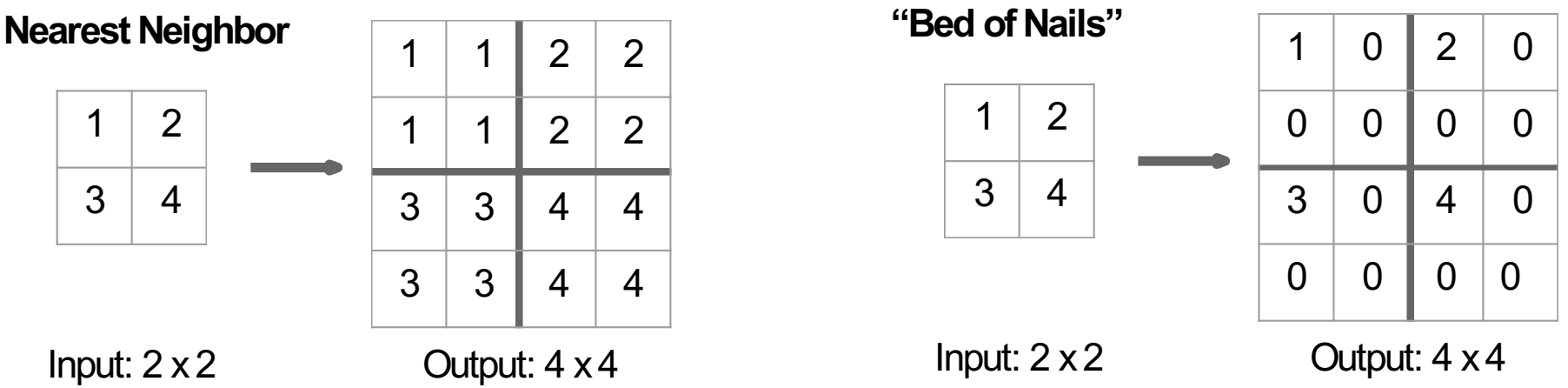
High-res:
D1 x H/2 x W/2

High-res:
D1 x H/2 x W/2

Predictions:
H x W

Long, Shelhamer, and Darrell, “Fully Convolutional Networks for Semantic Segmentation”, CVPR 2015
Noh et al, “Learning Deconvolution Network for Semantic Segmentation”, ICCV 2015

In-Network upsampling: "Unpooling"



In-Network upsampling: "Max Unpooling"

Max Pooling
Remember which element was max!

1	2	6	3
3	5	2	1
1	2	2	1
7	3	4	8

Input: 4 x 4

5	6
7	8

Output: 2 x 2

Rest of the network

Max Unpooling
Use positions from pooling layer

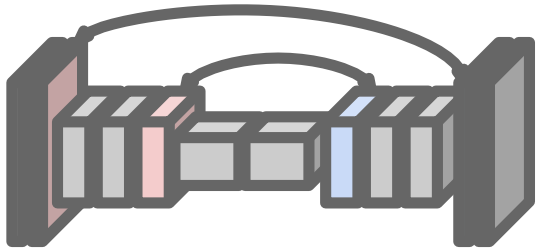
1	2
3	4

Input: 2 x 2

0	0	2	0
0	1	0	0
0	0	0	0
3	0	0	4

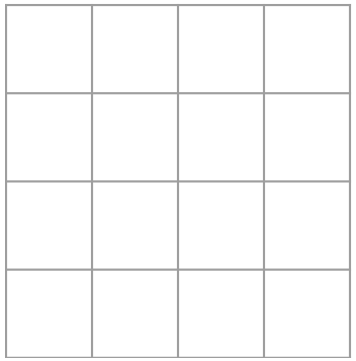
Output: 4 x 4

Corresponding pairs of downsampling and upsampling layers

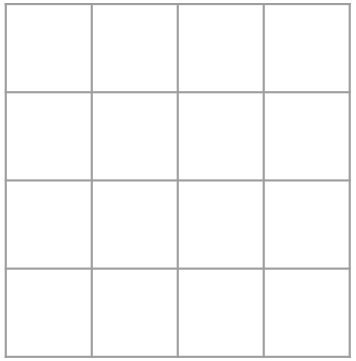


Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 1 pad 1



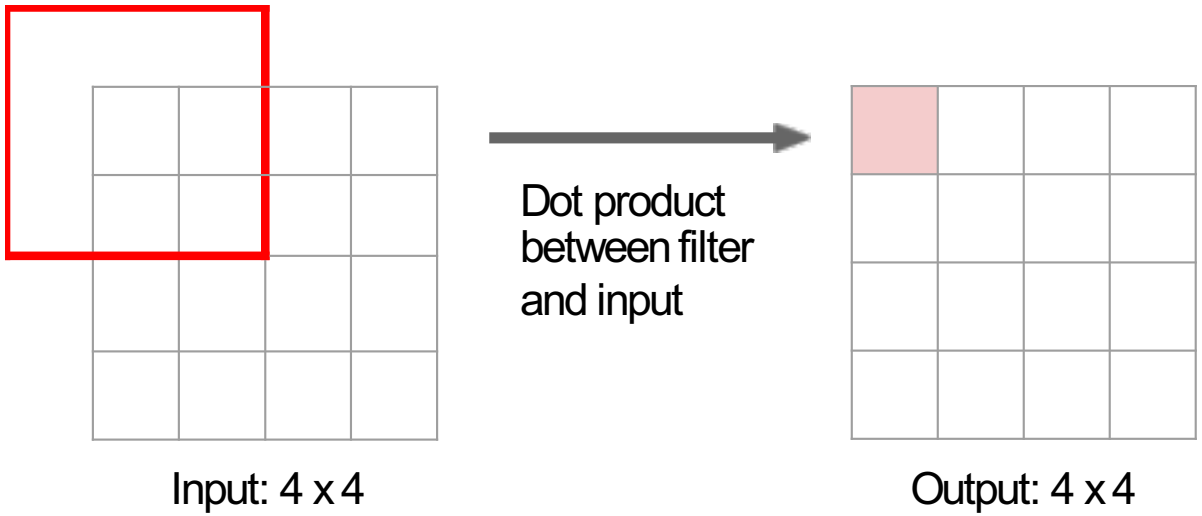
Input: 4 x 4



Output: 4 x 4

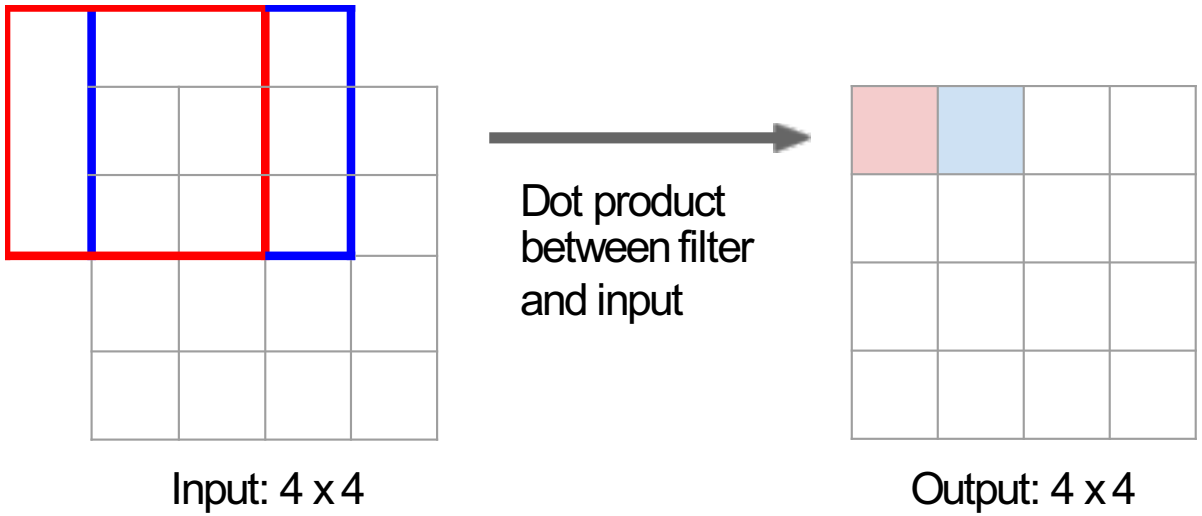
Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 1 pad 1



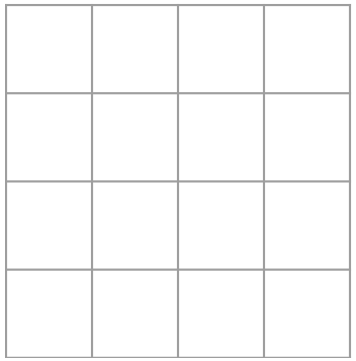
Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 1 pad 1

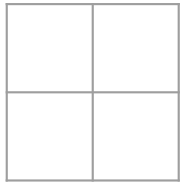


Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 2 pad 1



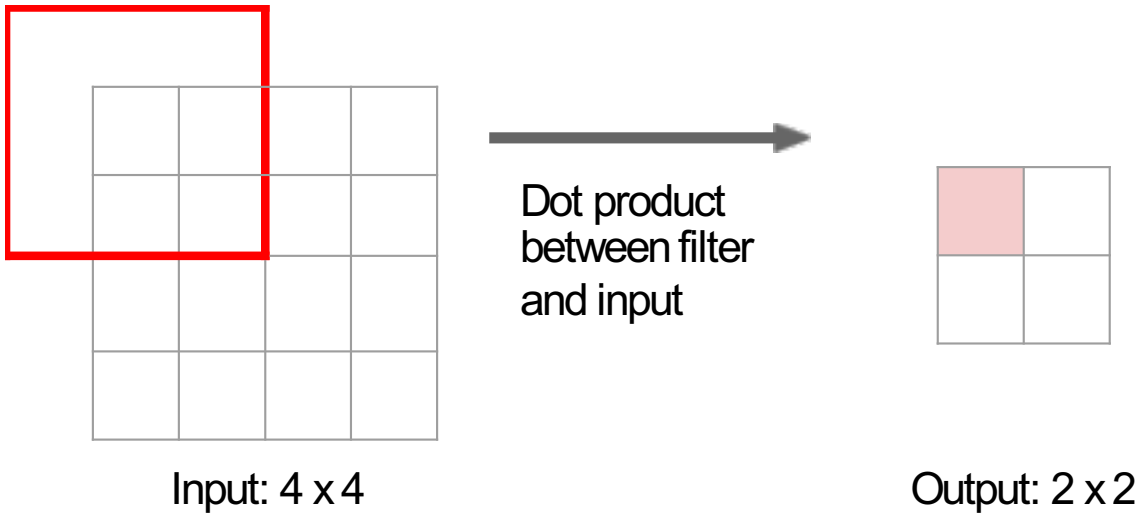
Input: 4 x 4



Output: 2 x 2

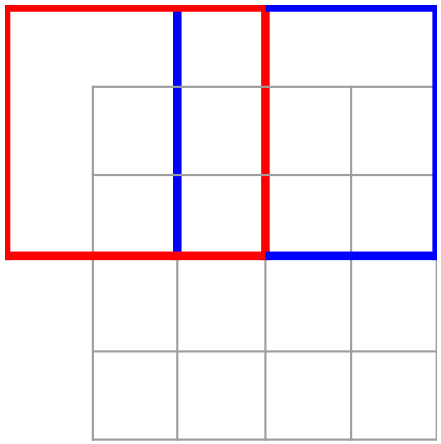
Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 2 pad 1



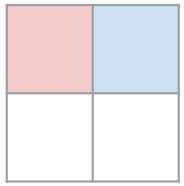
Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 2 pad 1



Input: 4 x 4

Dot product between filter and input



Output: 2 x 2

Filter moves 2 pixels in the input for every one pixel in the output

Stride gives ratio between movement in input and output

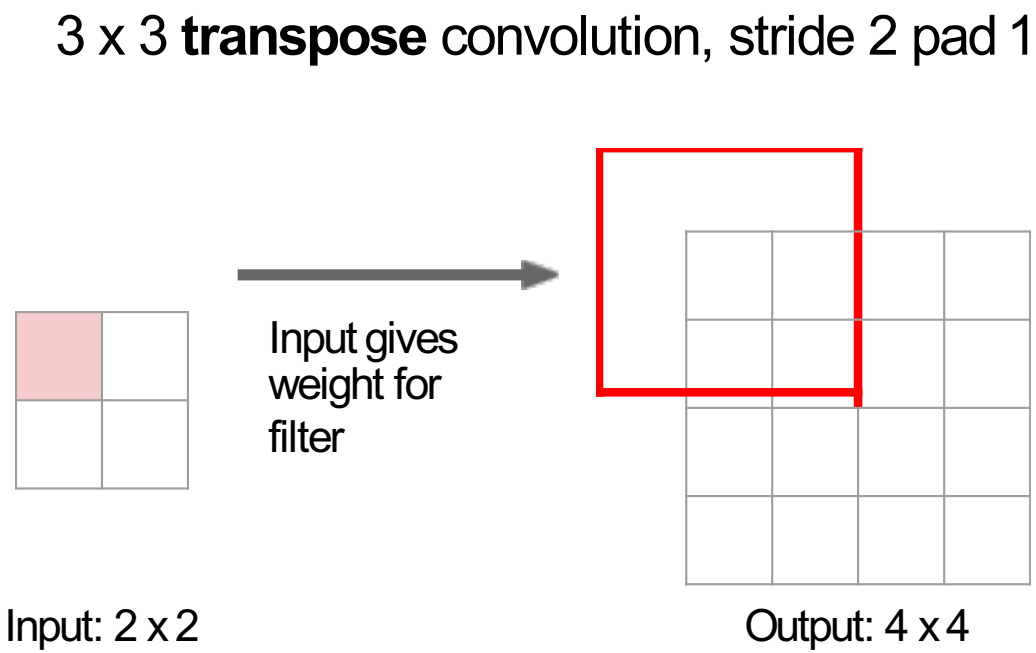
Learnable Upsampling: Transpose Convolution

3 x 3 **transpose** convolution, stride 2 pad 1

Input: 2 x 2

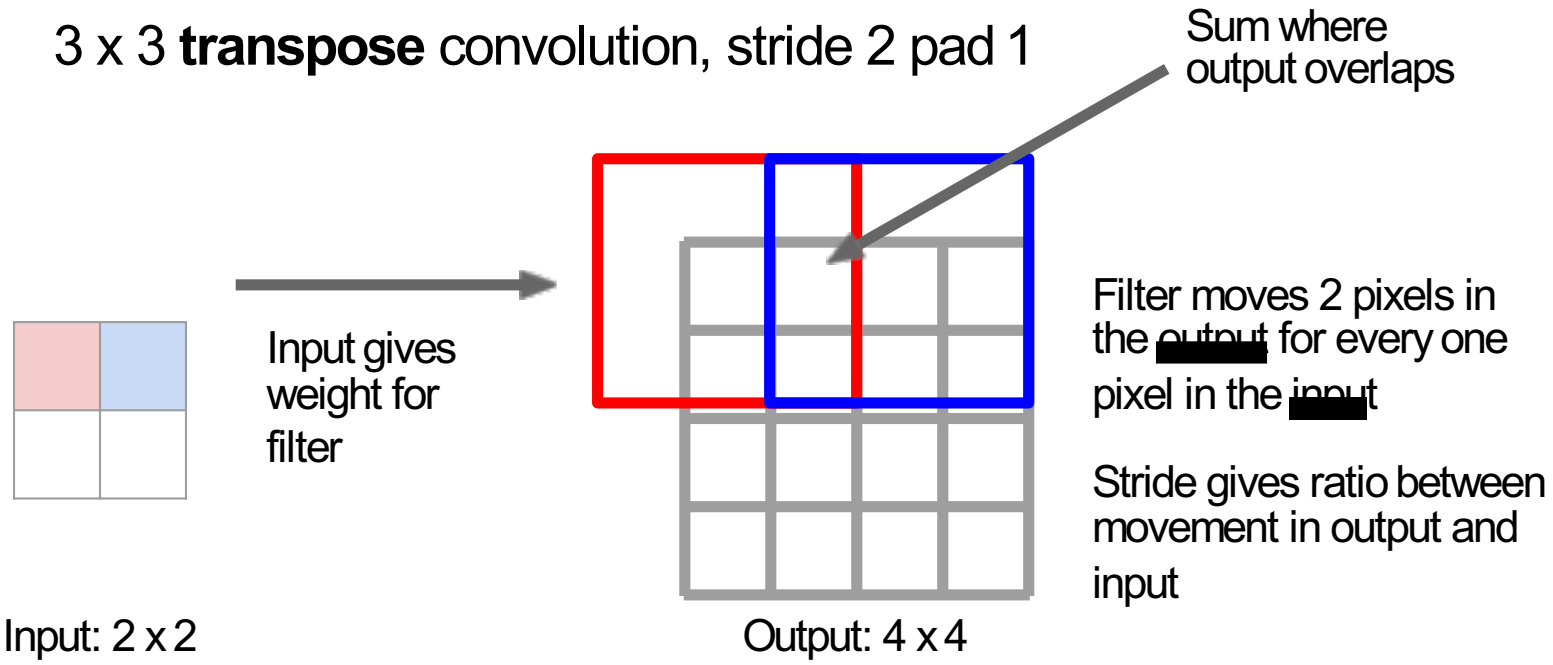
Output: 4 x 4

Learnable Upsampling: Transpose Convolution

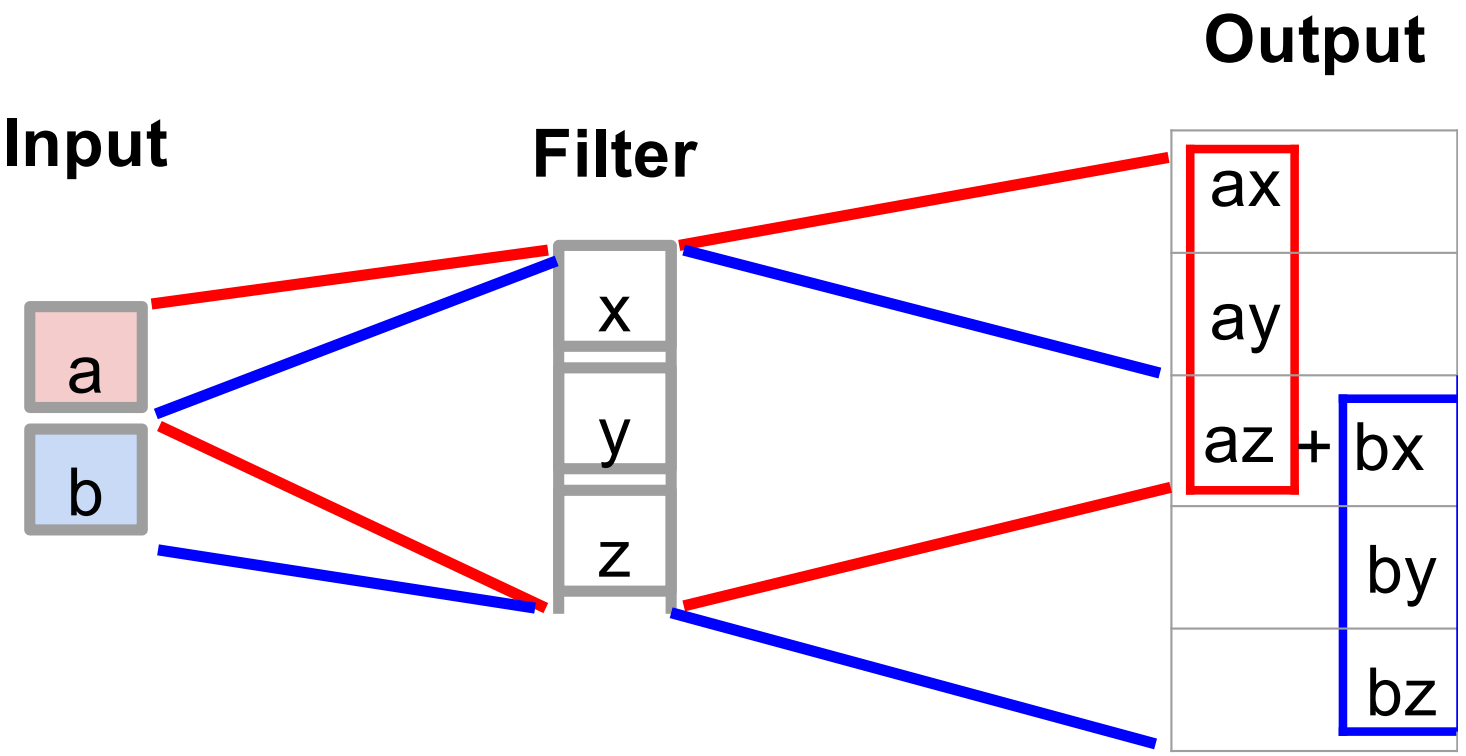


Learnable Upsampling: Transpose Convolution

Other names:
-Deconvolution (bad)
-Upconvolution
-Fractionally strided convolution
-Backward strided convolution



Learnable Upsampling: 1D Example



Output contains copies of the filter weighted by the input, summing at where at overlaps in the output

Need to crop one pixel from output to make output exactly 2x input

Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & x & y & z & 0 & 0 \\ 0 & 0 & x & y & z & 0 \\ 0 & 0 & 0 & x & y & z \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=1, padding=1

Convolution transpose multiplies by the transpose of the same matrix:

$$\vec{x} *^T \vec{a} = X^T \vec{a}$$

$$\begin{bmatrix} x & 0 & 0 & 0 \\ y & x & 0 & 0 \\ z & y & x & 0 \\ 0 & z & y & x \\ 0 & 0 & z & y \\ 0 & 0 & 0 & z \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} ax \\ ay + bx \\ az + by + cx \\ bz + cy + dx \\ cz + dy \\ dz \end{bmatrix}$$

When stride=1, convolution transpose is just a regular convolution (with different padding rules)

Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & 0 & x & y & z & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix}$$

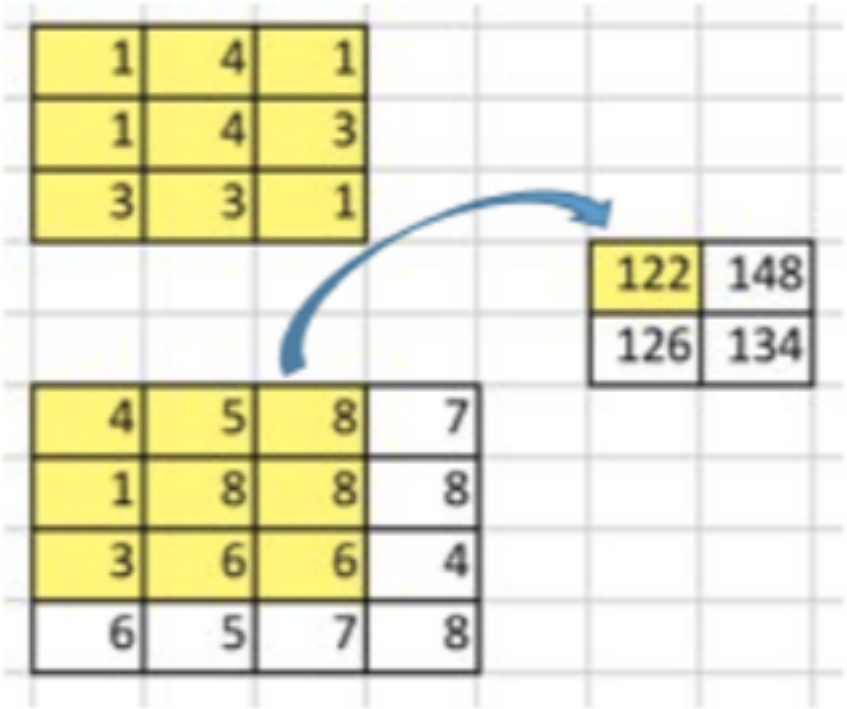
Example: 1D conv, kernel size=3, stride=2, padding=1

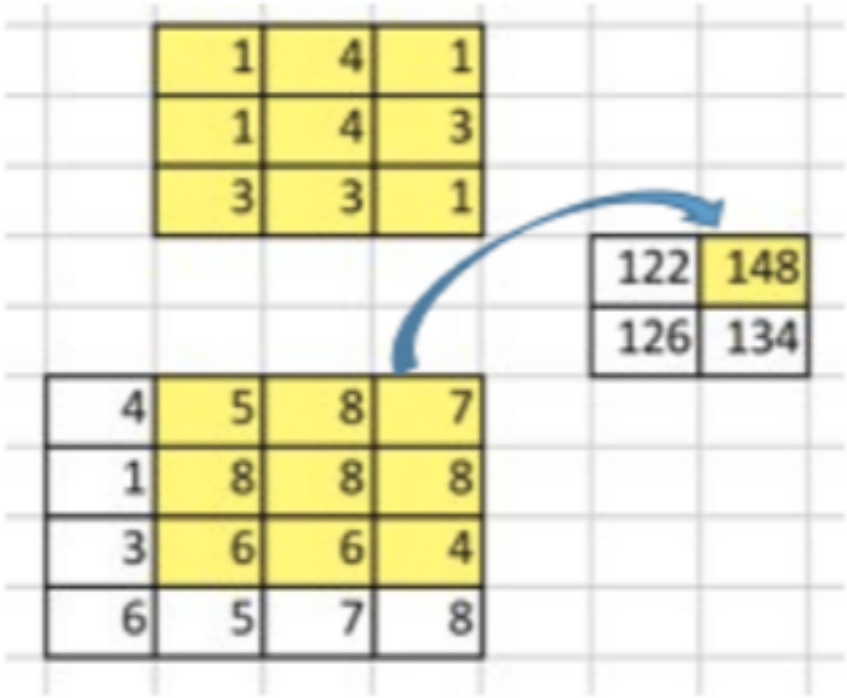
Convolution transpose multiplies by the transpose of the same matrix:

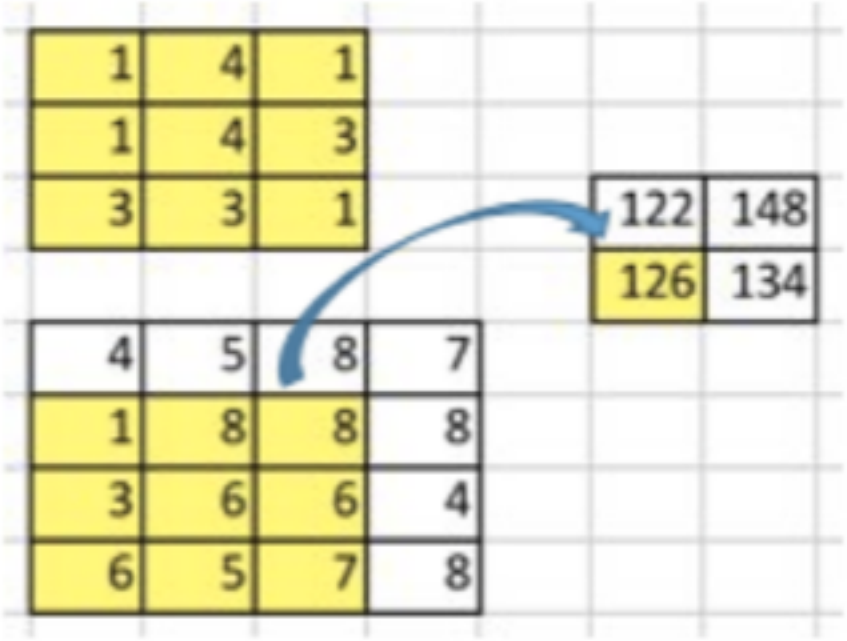
$$\vec{x} *^T \vec{a} = X^T \vec{a}$$

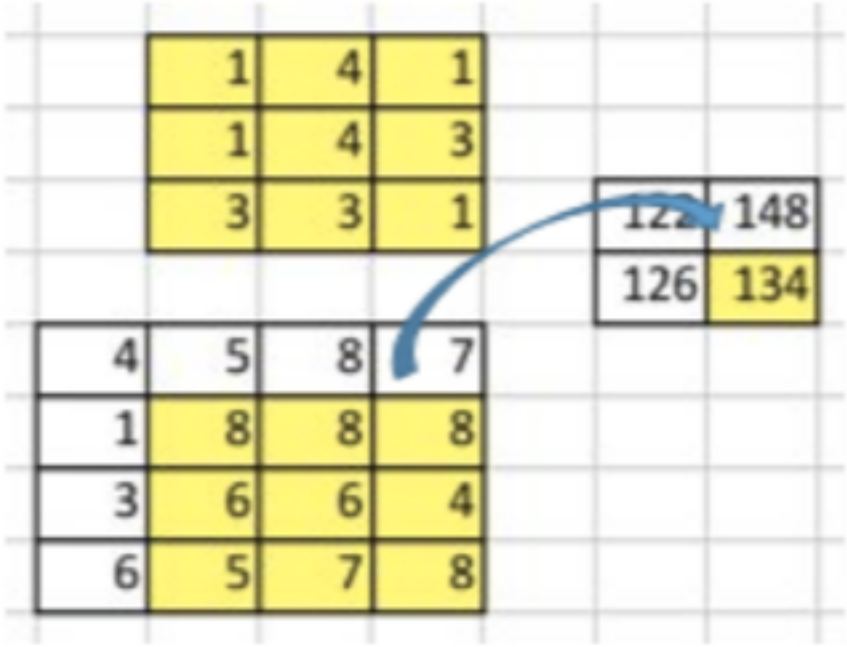
$$\begin{bmatrix} x & 0 \\ y & 0 \\ z & x \\ 0 & y \\ 0 & z \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az + bx \\ by \\ bz \\ 0 \end{bmatrix}$$

When stride>1, convolution transpose is no longer a normal convolution!

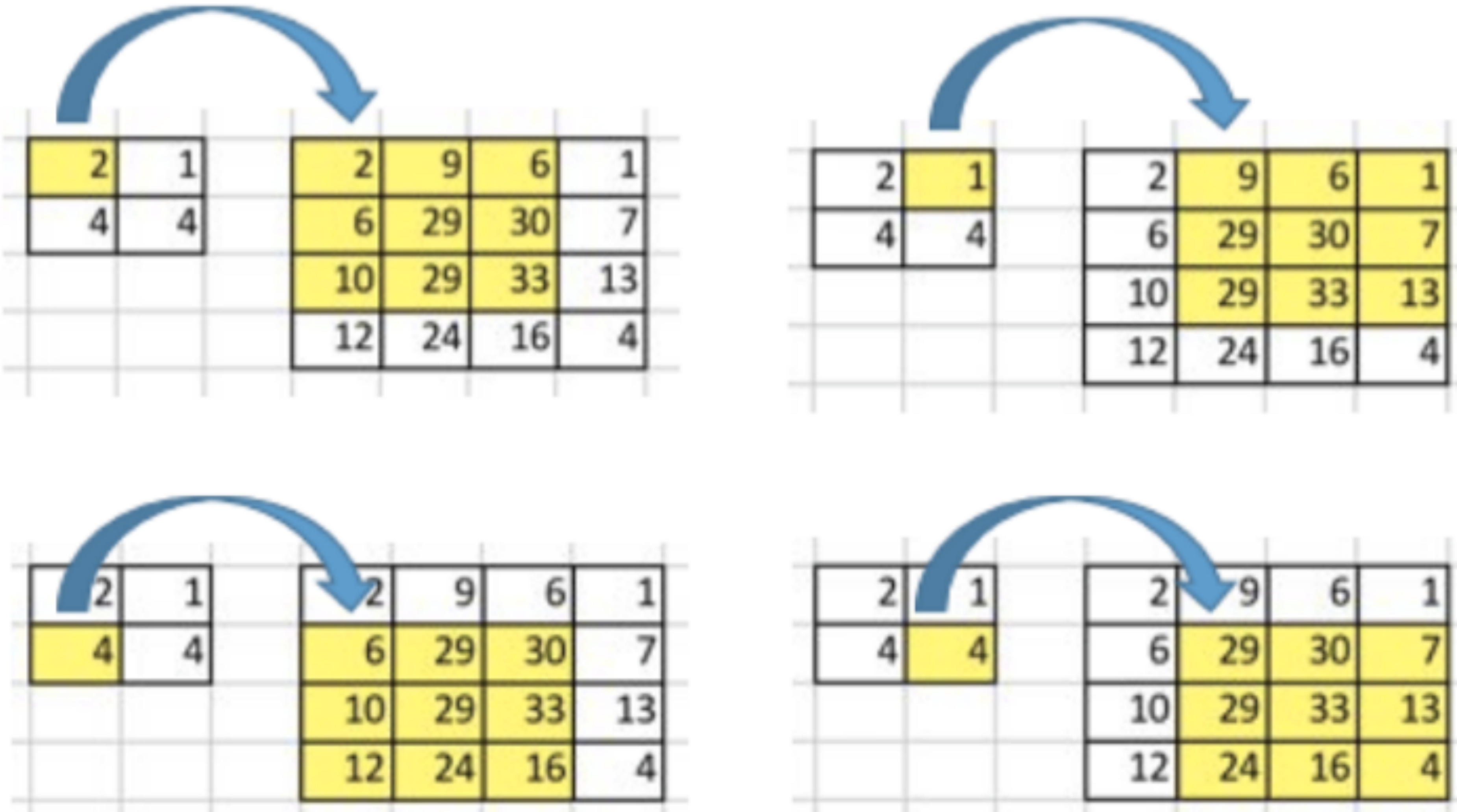

1 4 1
1 4 3
3 3 1
122 148
126 134
4 5 8 7
1 8 8 8
3 6 6 4
6 5 7 8


1 4 1
1 4 3
3 3 1
122 148
126 134
4 5 8 7
1 8 8 8
3 6 6 4
6 5 7 8

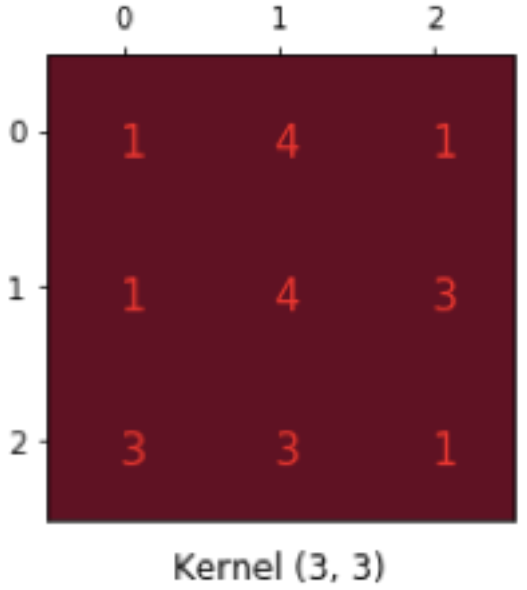

1 4 1
1 4 3
3 3 1
122 148
126 134
4 5 8 7
1 8 8 8
3 6 6 4
6 5 7 8


1 4 1
1 4 3
3 3 1
122 148
126 134
4 5 8 7
1 8 8 8
3 6 6 4
6 5 7 8

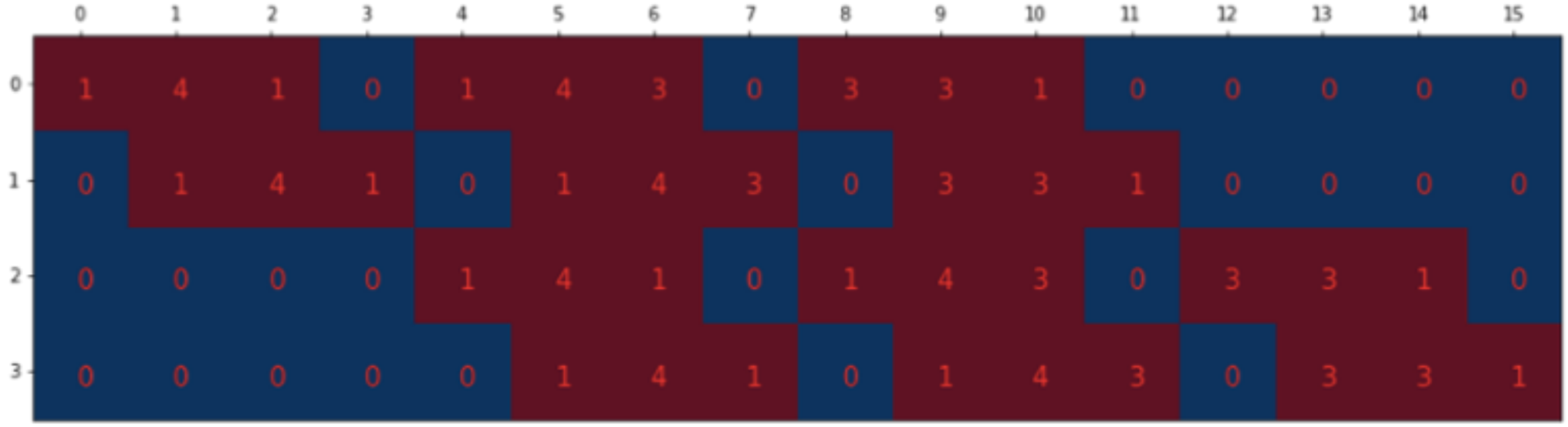
2D Transpose Convolution



Convolution as Matrix Operation

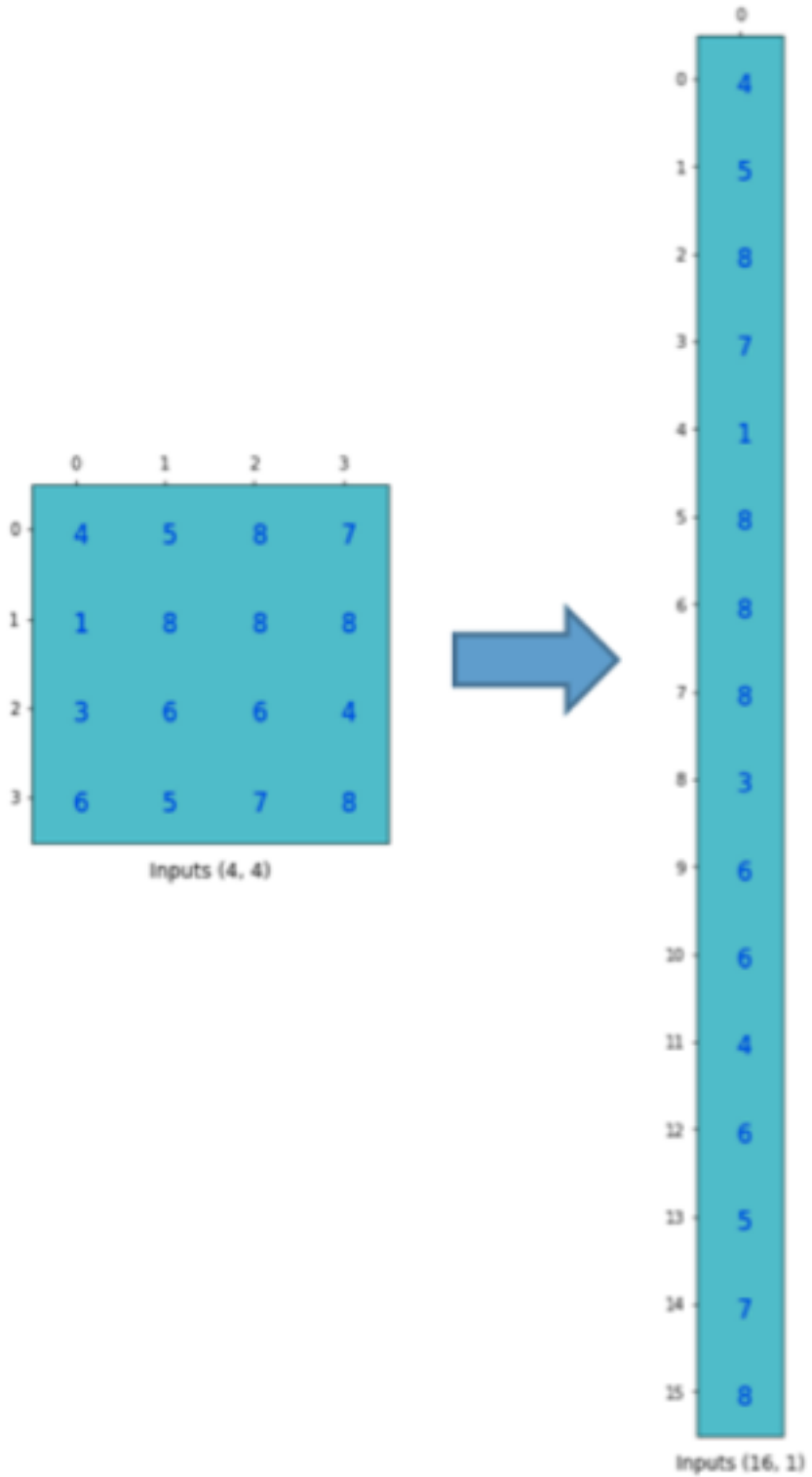


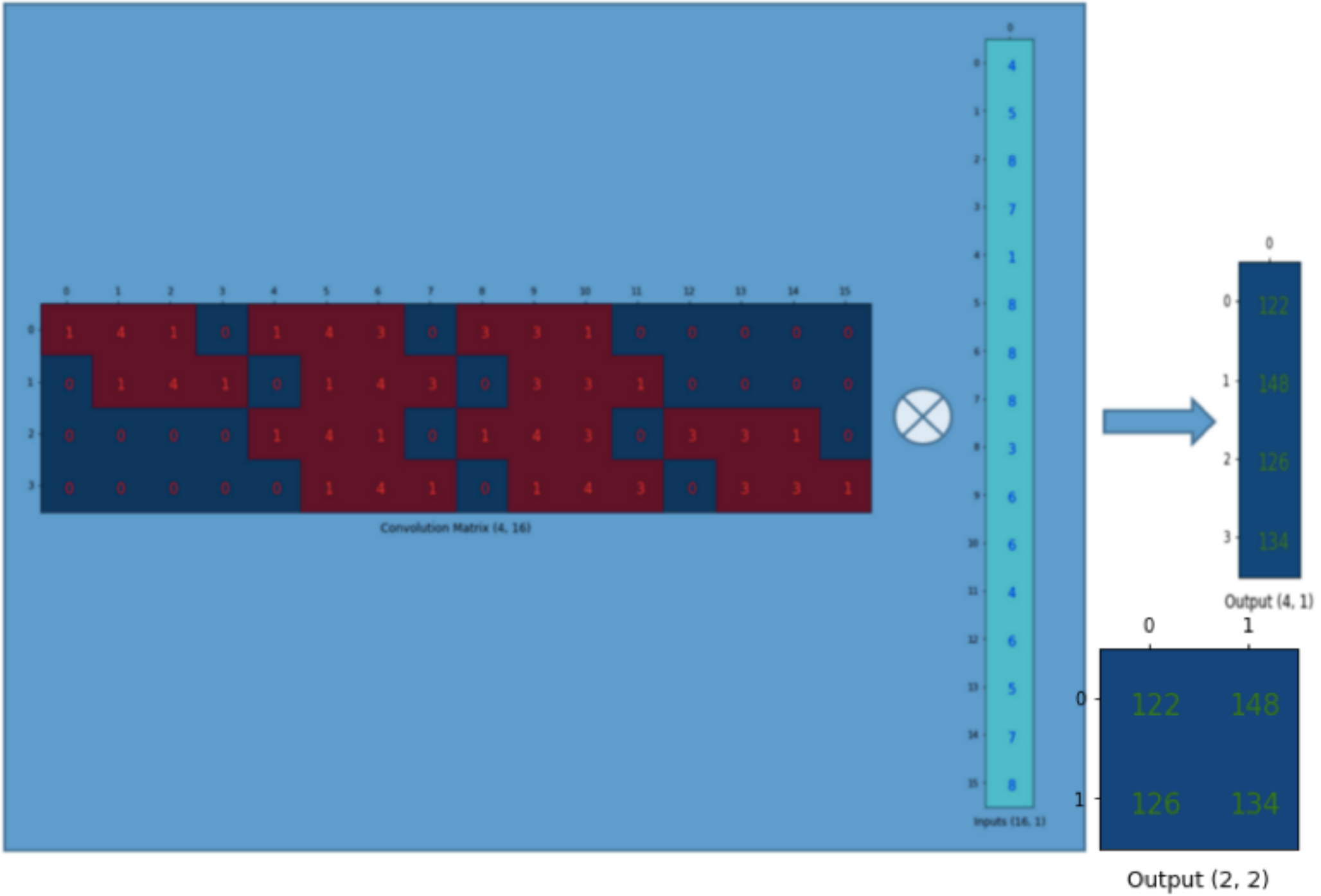
Kernel (3, 3)

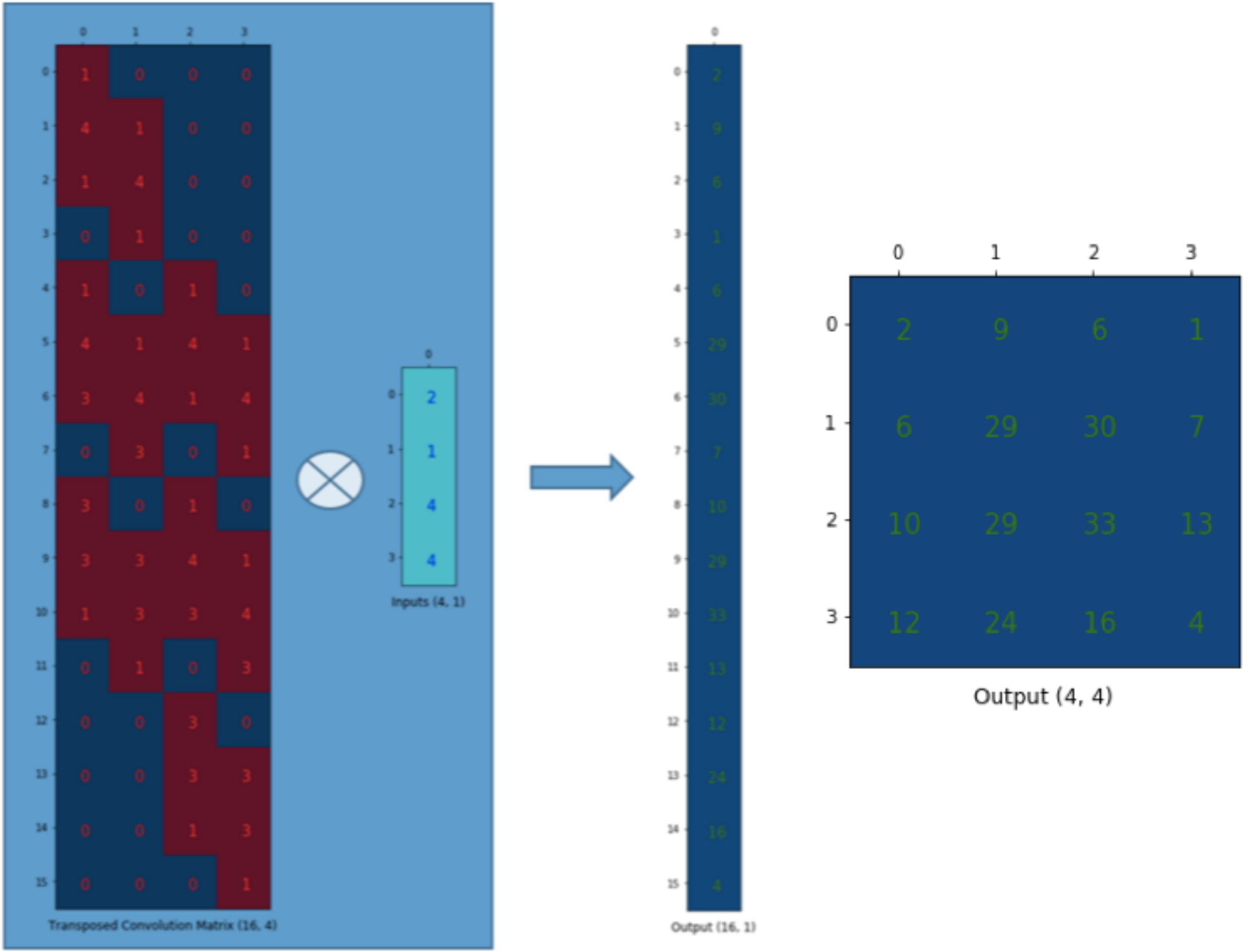


Convolution Matrix (4, 16)

Flattened Image




Convolution Matrix (4, 16)
Inputs (16, 1)
Output (4, 1)
Output (2, 2)


Transposed Convolution Matrix (16, 4)
Inputs (4, 1)
Output (16, 1)
Output (4, 4)

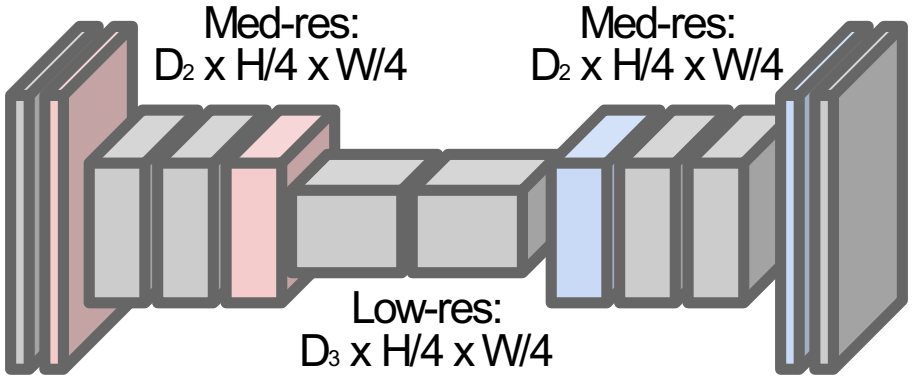
Semantic Segmentation Idea: Fully Convolutional

Downsampling:
Pooling, strided convolution

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Upsampling:
Unpooling or strided transpose convolution

Input:
3 x H x W



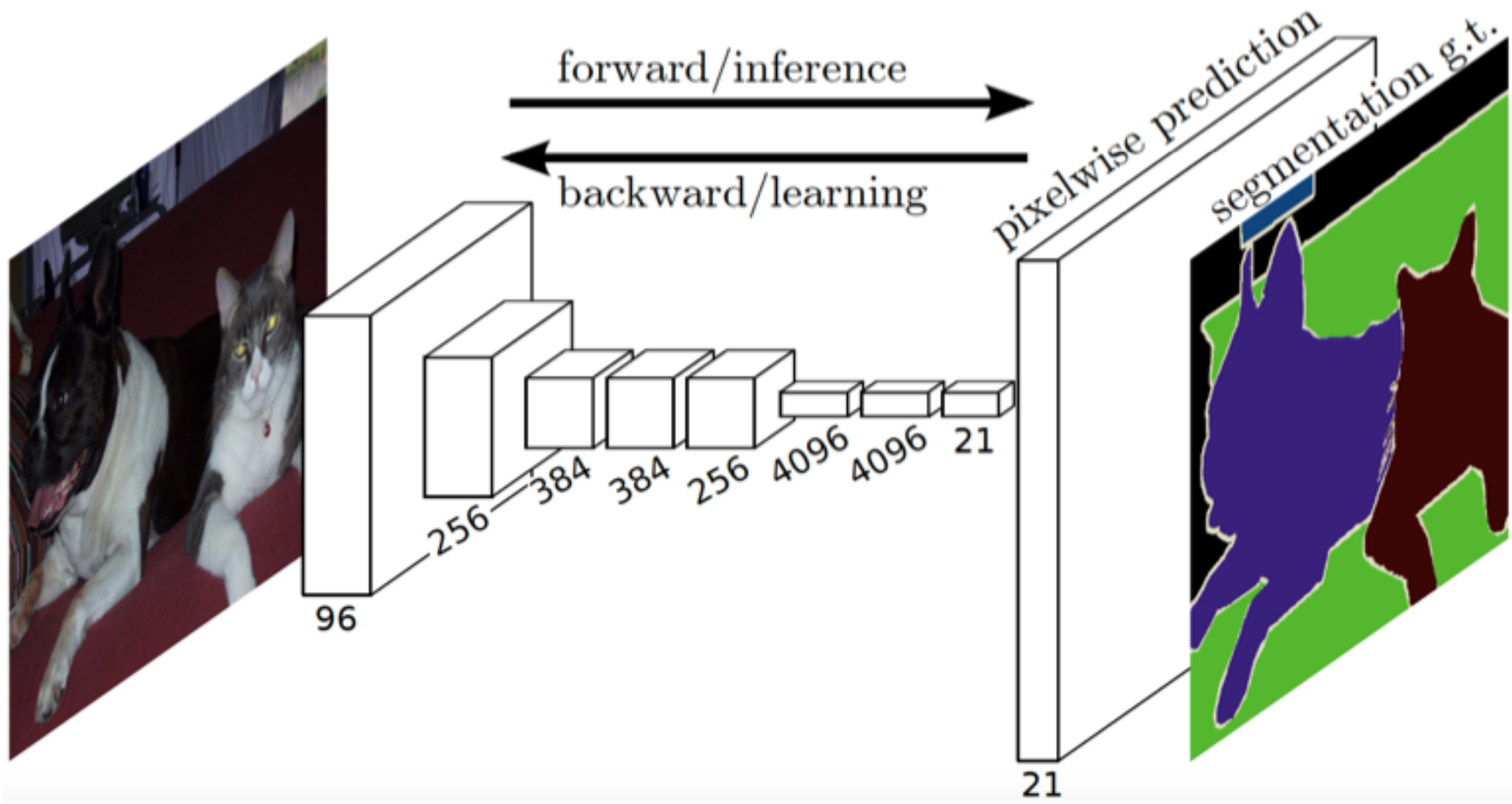
High-res:
D1 x H/2 x W/2

High-res:
D1 x H/2 x W/2

Predictions:
H x W

Long, Shelhamer, and Darrell, “Fully Convolutional Networks for Semantic Segmentation”, CVPR 2015
Noh et al, “Learning Deconvolution Network for Semantic Segmentation”, ICCV 2015

FCN [cvpr 15]


forward/inference
backward/learning
pixelwise prediction
segmentation g.t.
96
256
384
384
256
4096
4096
21
21

FCN [cvpr 15]

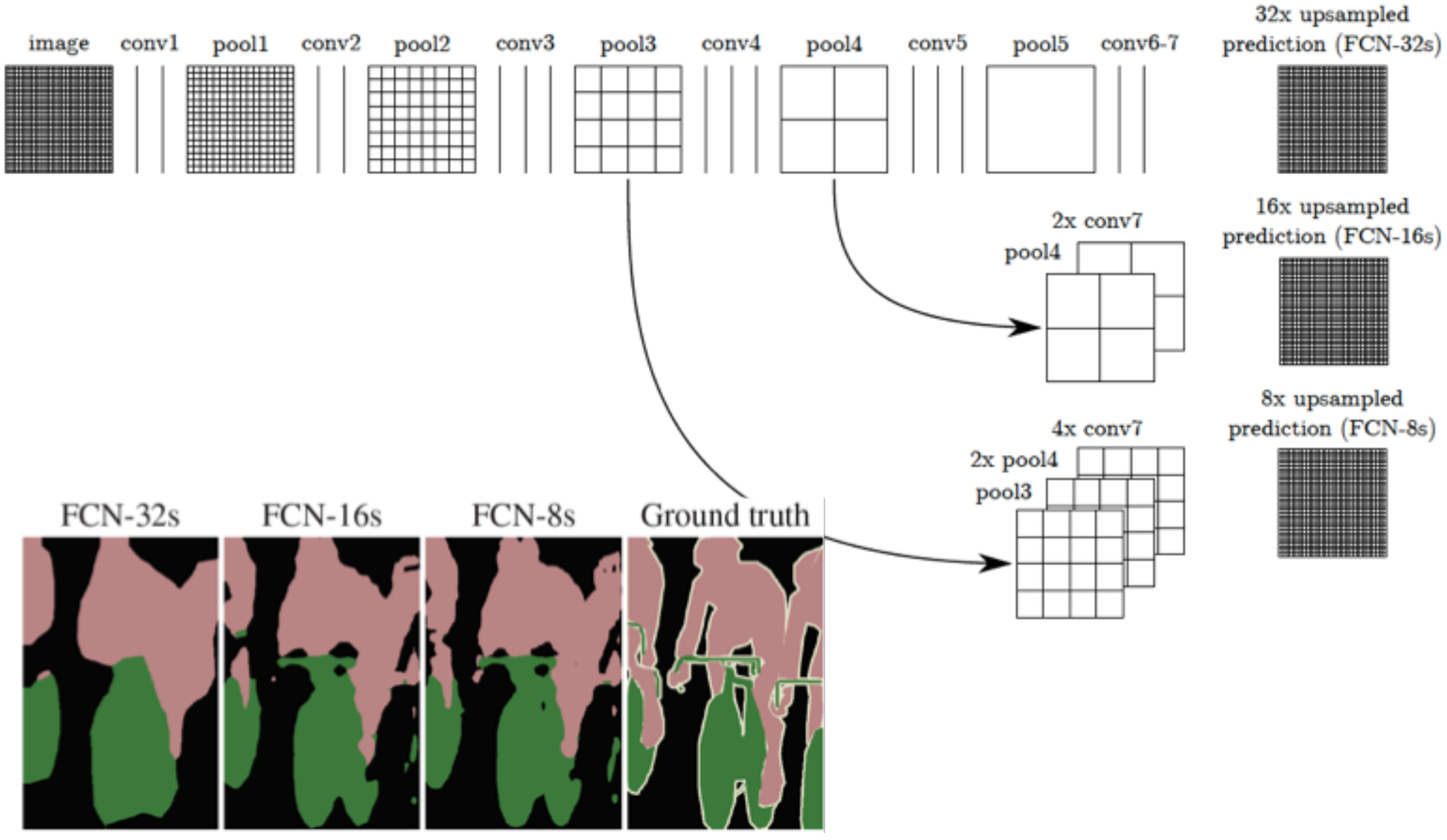
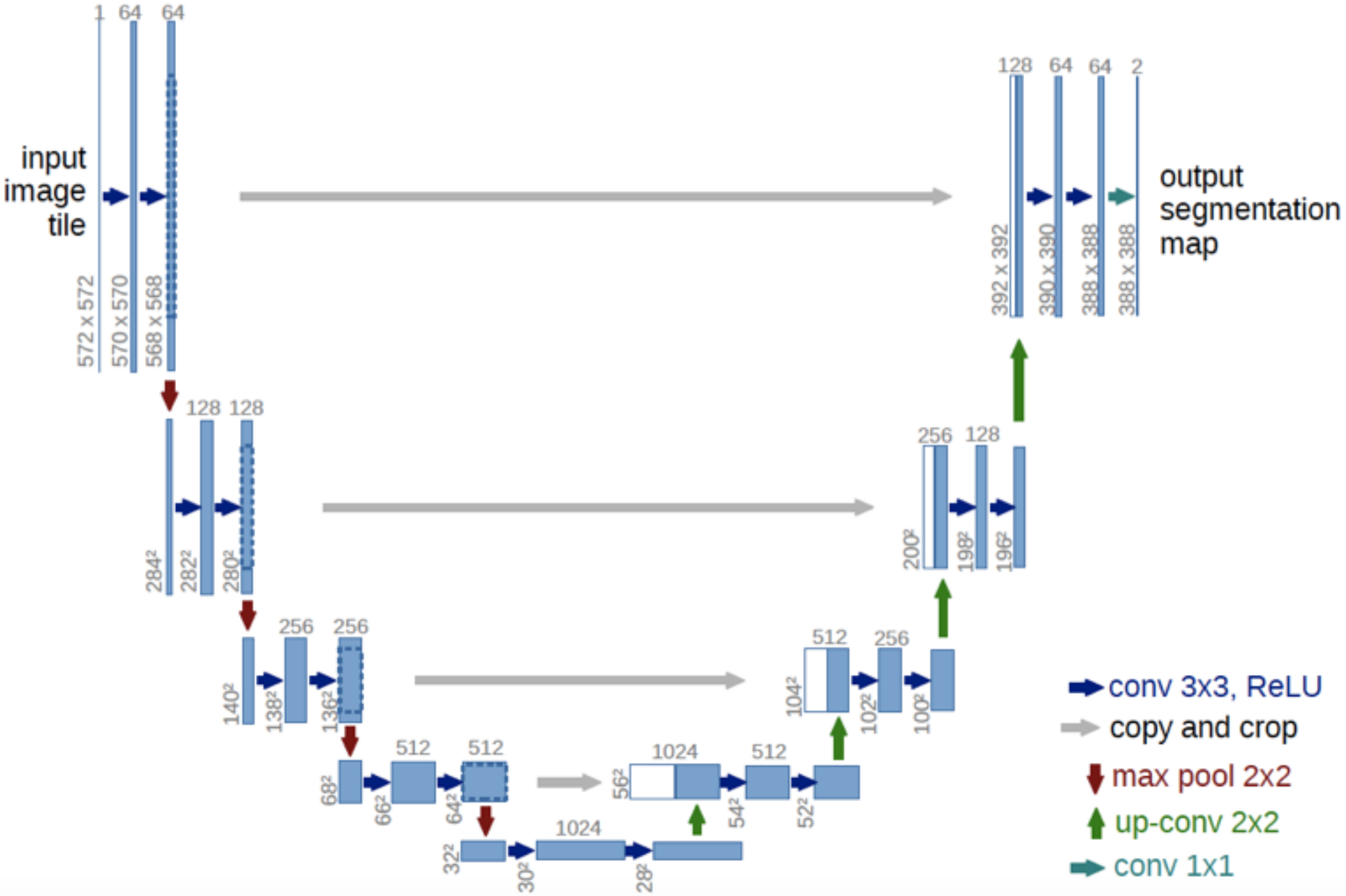


image
conv1
pool1
conv2
pool2
conv3
pool3
conv4
pool4
conv5
pool5
conv6-7
32x upsampled prediction (FCN-32s)
2x conv7
pool4
16x upsampled prediction (FCN-16s)
4x conv7
2x pool4
pool3
8x upsampled prediction (FCN-8s)
FCN-32s
FCN-16s
FCN-8s
Ground truth

U-net [ISBI 15]


input image tile
output segmentation map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

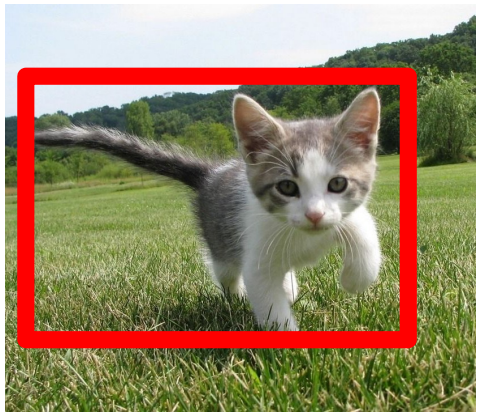
Other Computer Vision Tasks

Semantic Segmentation

GRASS, CAT, TREE, SKY

No objects, just pixels

Classification + Localization



CAT

Single Object

Object Detection

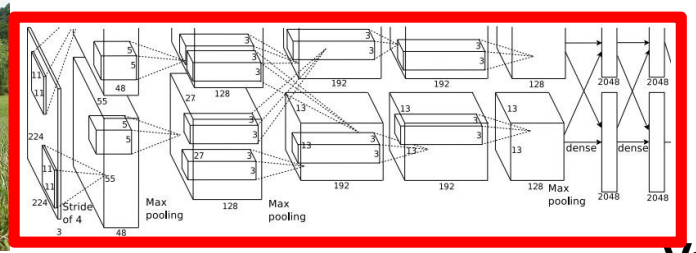
DOG, DOG, CAT

Instance Segmentation

DOG, DOG, CAT

Multiple Object

Classification + Localization



Often pretrained on ImageNet
(Transfer learning)

Vector: 4096

Fully Connected: 4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Softmax Loss

Correct label: Cat

Multitask Loss

Loss

Fully Connected: 4096 to 4

Box Coordinates (x, y, w, h)

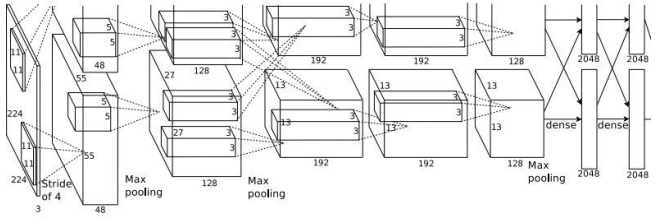
L2 Loss

Correct box: (x', y', w', h')

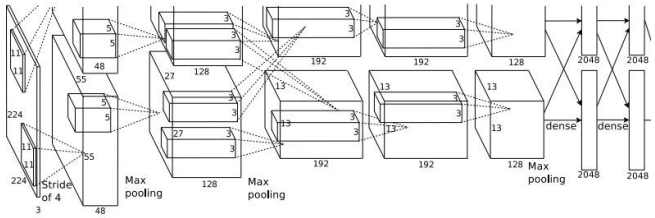
Treat localization as a regression problem!

Object Detection as Regression?

Each image needs a different number of outputs!

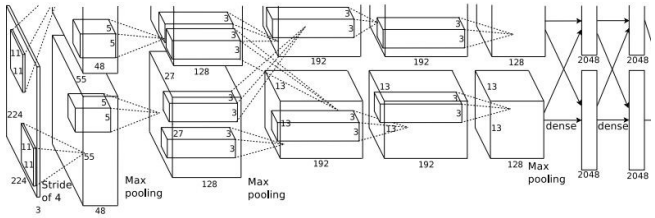


CAT: (x, y, w, h) — 4 numbers



DOG: (x, y, w, h)
DOG: (x, y, w, h)
CAT: (x, y, w, h)

16 numbers

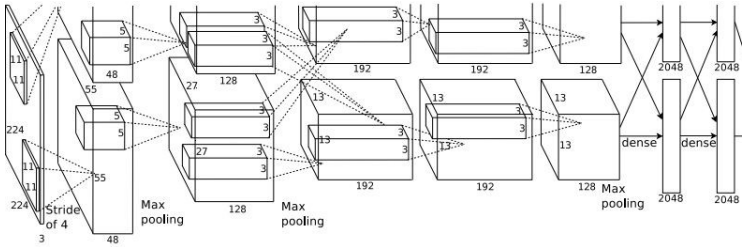


DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
….

Many numbers!
….

Object Detection as Classification: Sliding Window

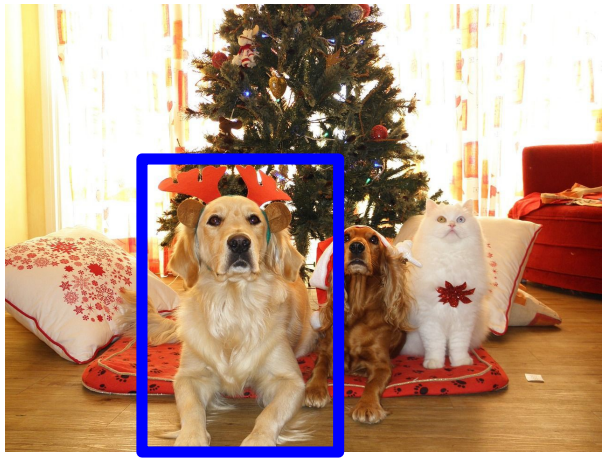
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

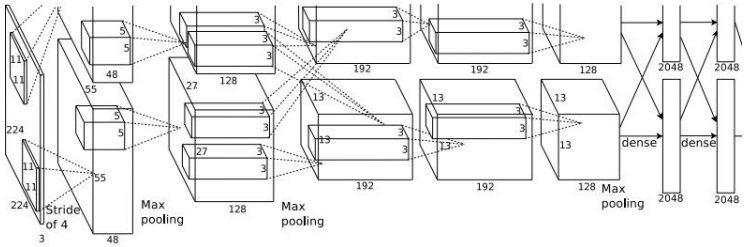


Dog? NO
Cat? NO
Background? YES

Object Detection as Classification: Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

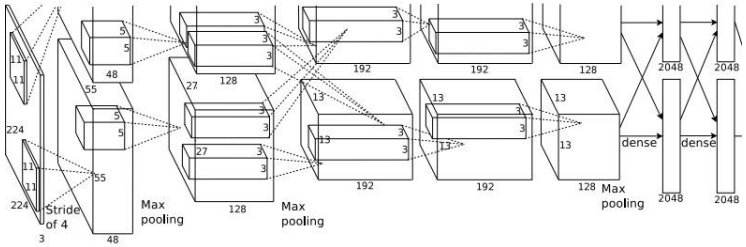




Dog? YES
Cat? NO
Background? NO

Object Detection as Classification: Sliding Window

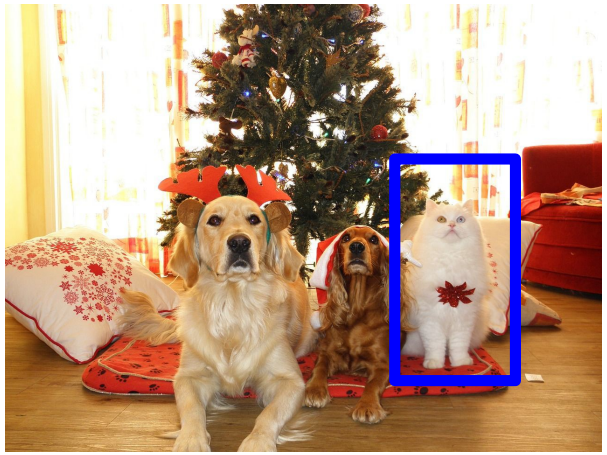
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

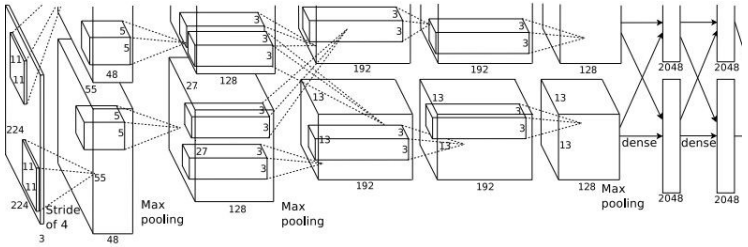


Dog? YES
Cat? NO
Background? NO

Object Detection as Classification: Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

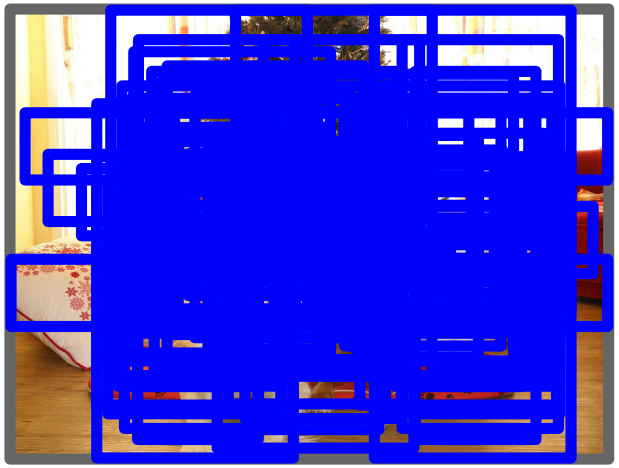


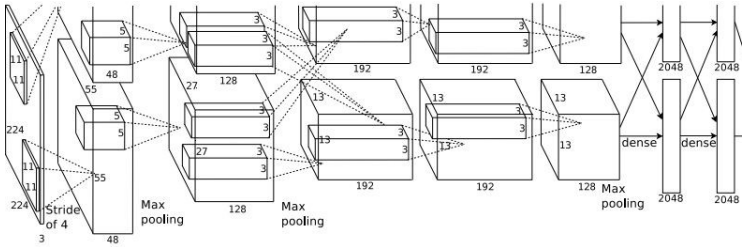


Dog? NO
Cat? YES
Background? NO

Object Detection as Classification: Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



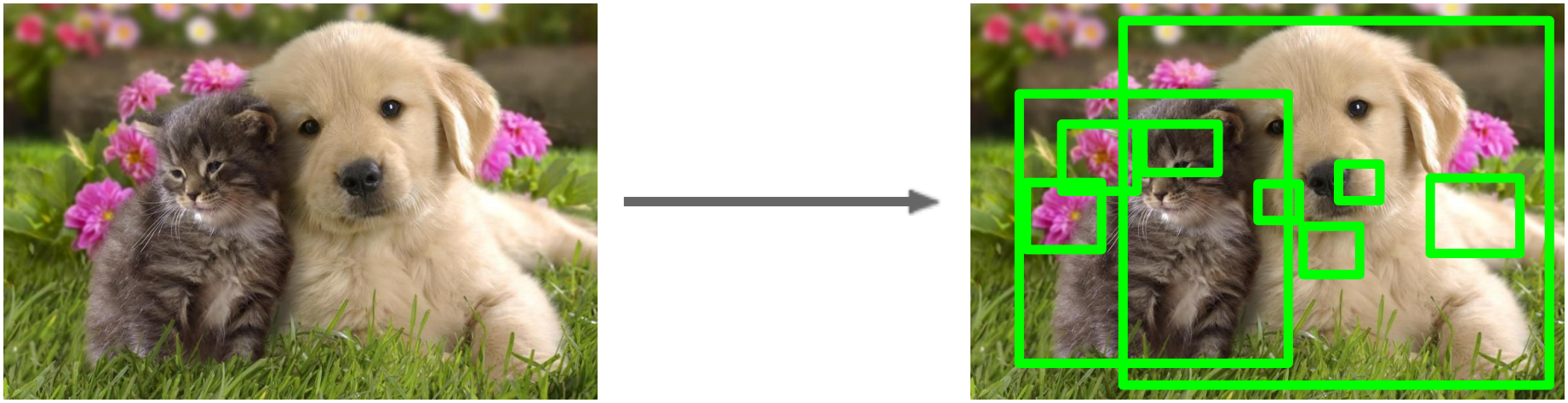


Dog? NO
Cat? YES
Background? NO

Problem: Need to apply CNN to huge number of locations, scales, and aspect ratios, very computationally expensive!

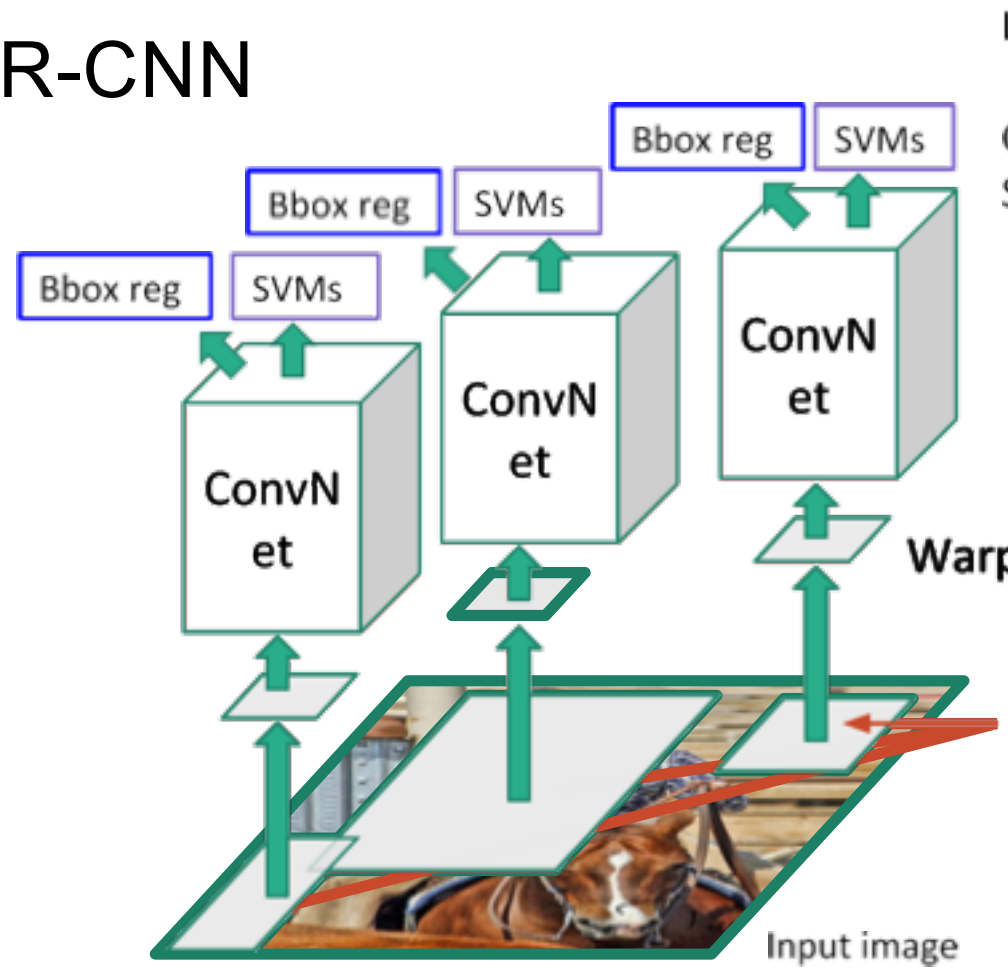
Region Proposals / Selective Search

- Find “blobby” image regions that are likely to contain objects
- Relatively fast to run; e.g. Selective Search gives 2000 region proposals in a few seconds on CPU



Alexe et al, “Measuring the objectness of image windows”, TPAMI 2012
Uijlings et al, “Selective Search for Object Recognition”, IJCV 2013
Cheng et al, “BING: Binarized normed gradients for objectness estimation at 300fps”, CVPR 2014
Zitnick and Dollar, “Edge boxes: Locating object proposals from edges”, ECCV 2014

R-CNN



Linear Regression for bounding box offsets

Classify regions with SVMs

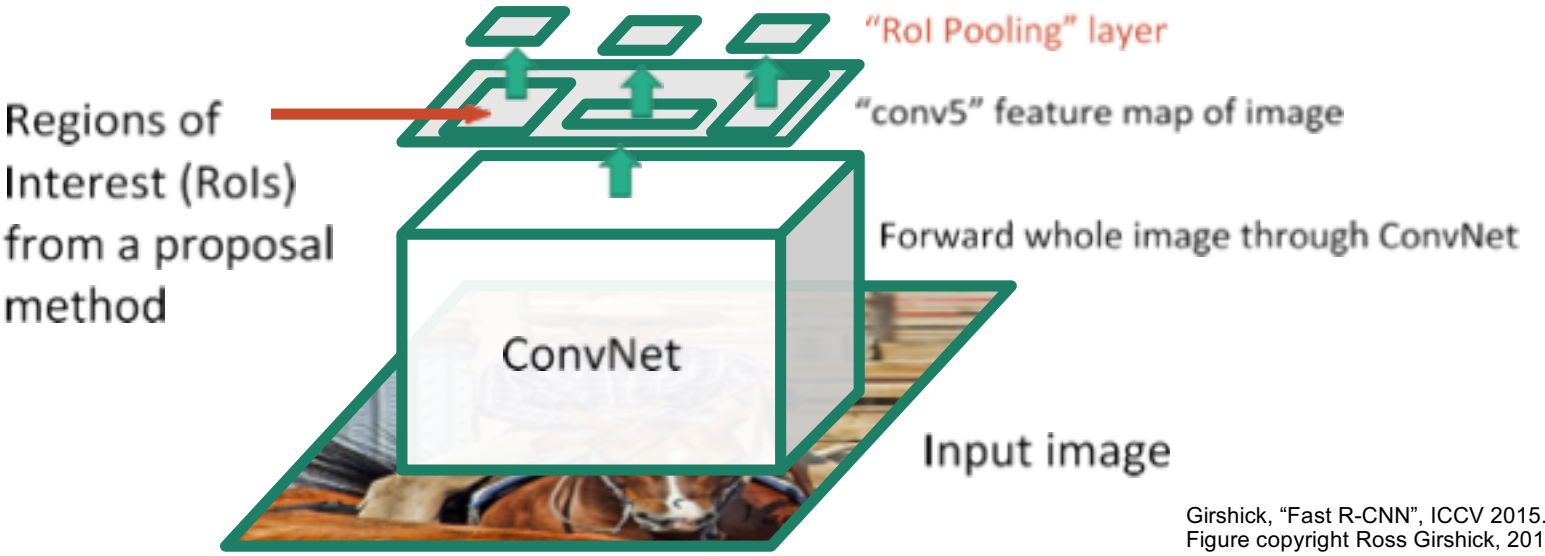
Forward each region through ConvNet

Warped image regions

Regions of Interest (RoI) from a proposal method (~2k)

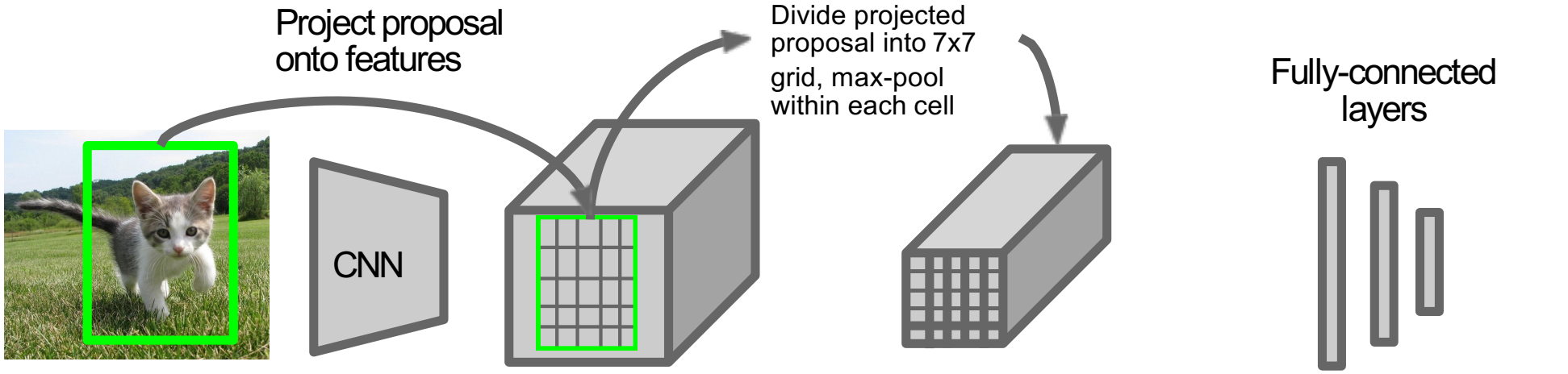
Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

Fast R-CNN



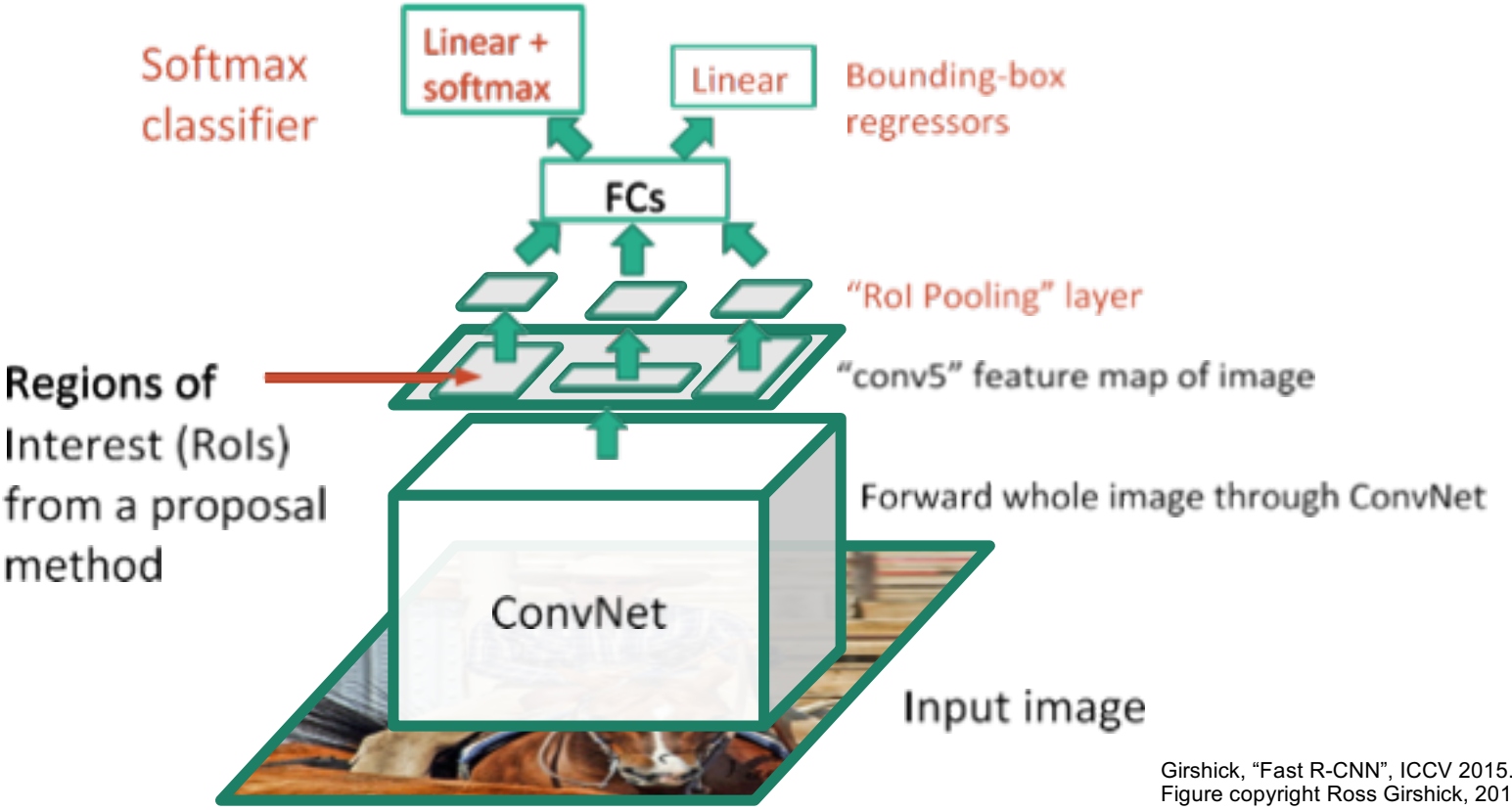
Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Fast R-CNN: RoI Pooling



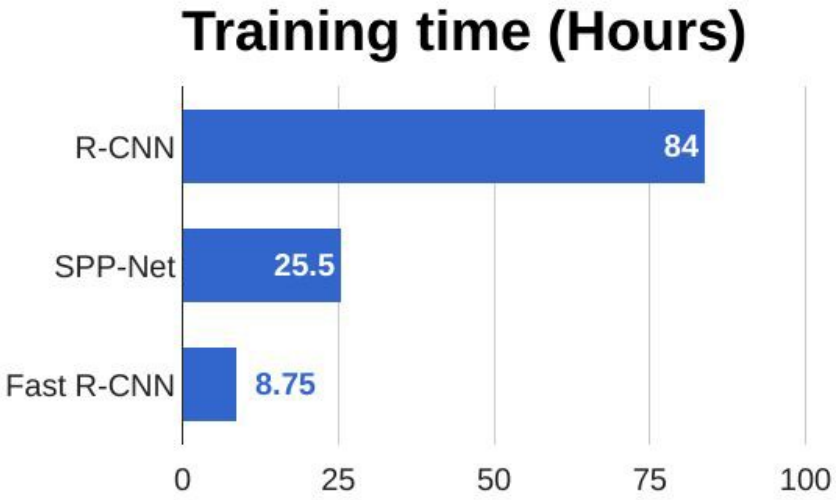
Girshick, "Fast R-CNN", ICCV 2015.

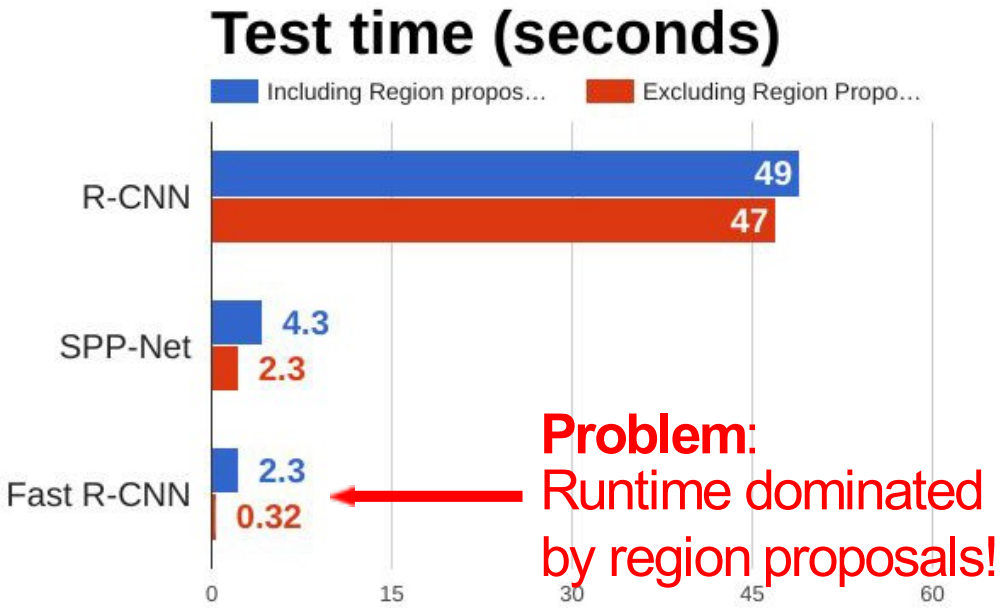
Fast R-CNN



Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

R-CNN vs SPP vs Fast R-CNN





Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.
He et al, “Spatial pyramid pooling in deep convolutional networks for visual recognition”, ECCV 2014

Girshick, “Fast R-CNN”, ICCV 2015

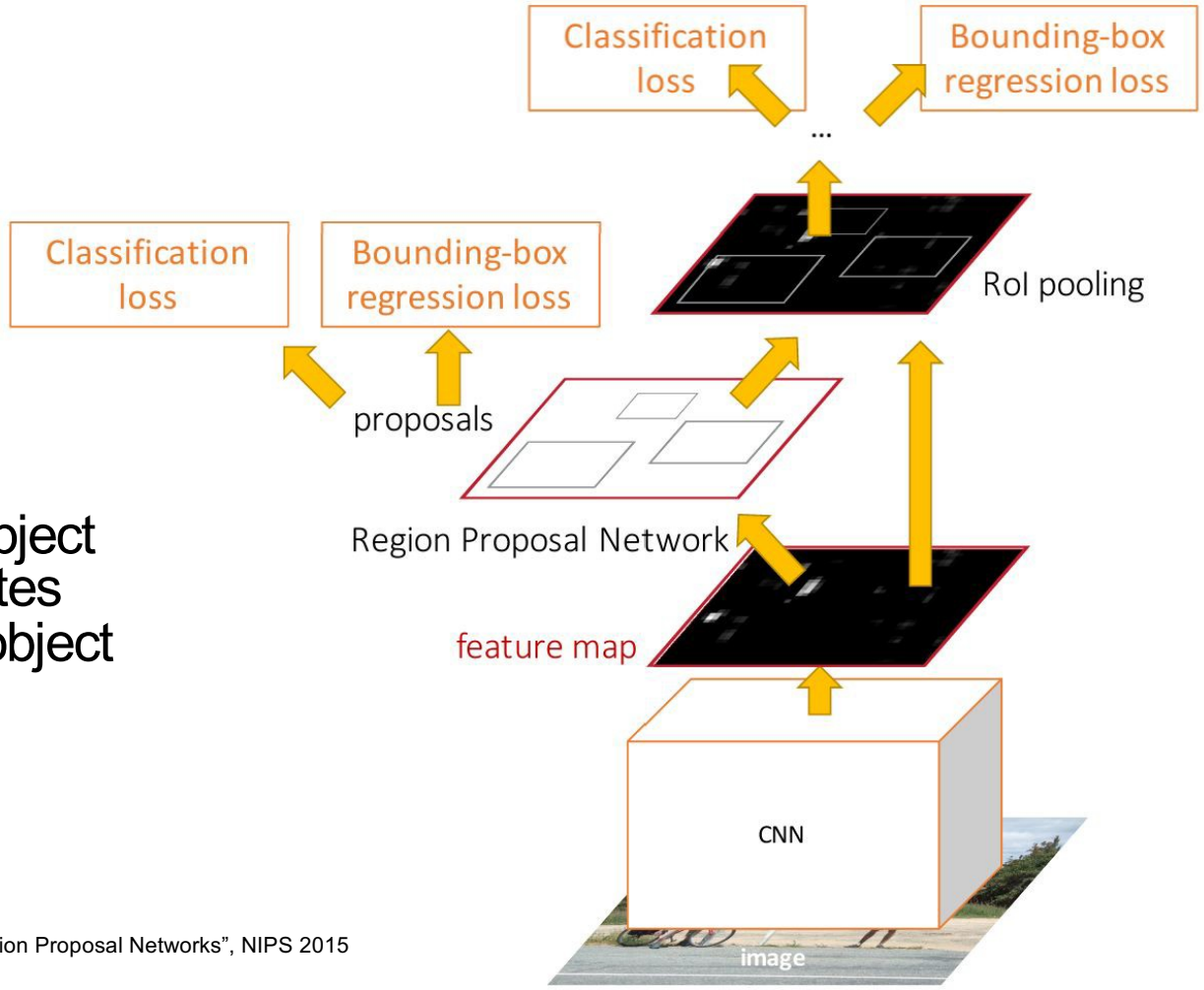
Faster R-CNN:

Make CNN do proposals

Insert **Region Proposal Network (RPN)** to predict proposals from features

Jointly train with 4 losses:

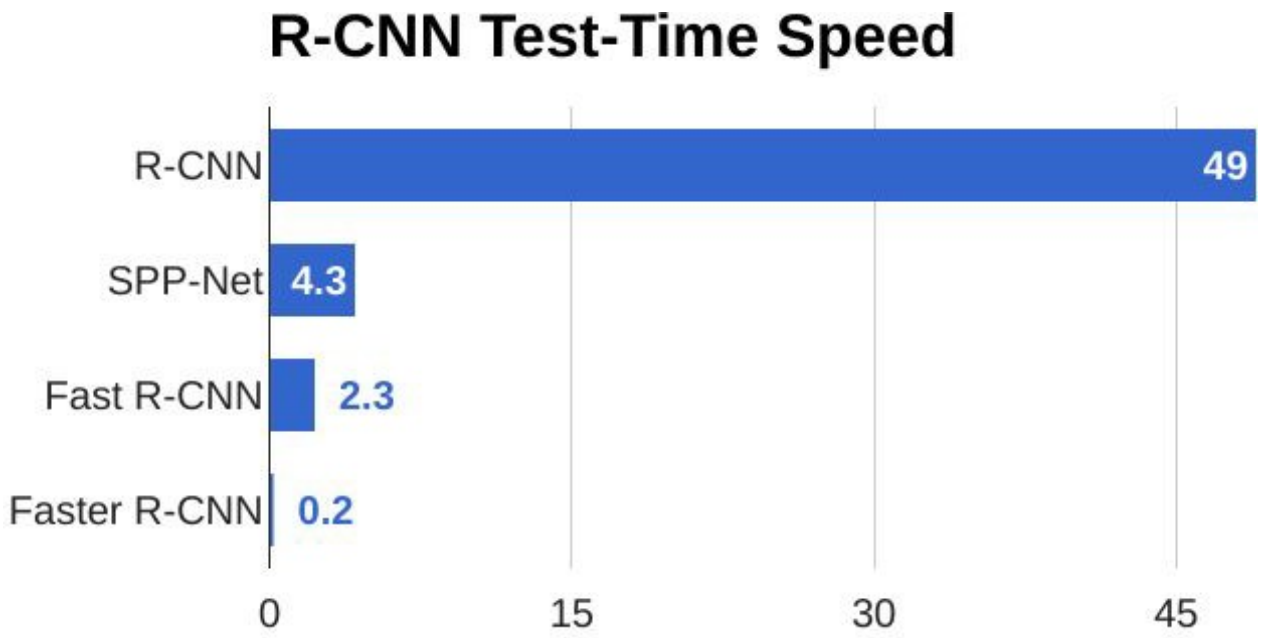
1. RPN classify object / not object
2. RPN regress box coordinates
3. Final classification score (object classes)
4. Final box coordinates



Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015

Faster R-CNN:

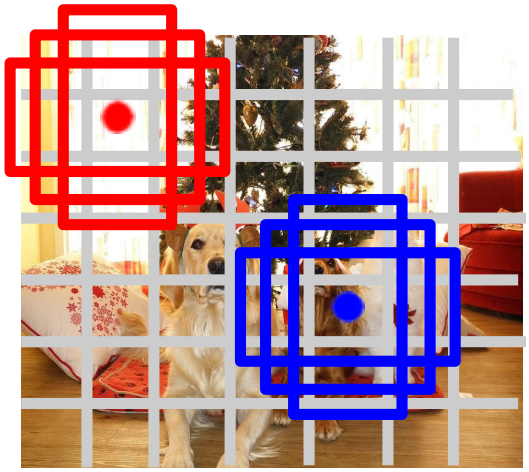
Make CNN do proposals!



Detection without Proposals: YOLO / SSD

Go from input image to tensor of scores with one big convolutional network!

Input image
3 x H x W



Divide image into grid
7 x 7

Image a set of **base boxes** centered at each grid cell
Here B = 3

Within each grid cell:

- Regress from each of the B base boxes to a final box with 5 numbers:
  (dx, dy, dh, dw, confidence)
- Predict scores for each of C classes (including background as a class)

Output:
7 x 7 x (5 * B + C)

Redmon et al, "You Only Look Once: Unified, Real-Time Object Detection", CVPR 2016

Liu et al, "SSD: Single-Shot MultiBox Detector", ECCV 2016

Object Detection: Lots of variables …

Base Network

VGG16
ResNet-101
Inception V2
Inception V3
Inception
ResNet
MobileNet

Object Detection architecture

Faster R-CNN
R-FCN
SSD

Image Size

 # Region Proposals

…

Takeaways

Faster R-CNN is slower but more accurate

SSD is much faster but not as accurate

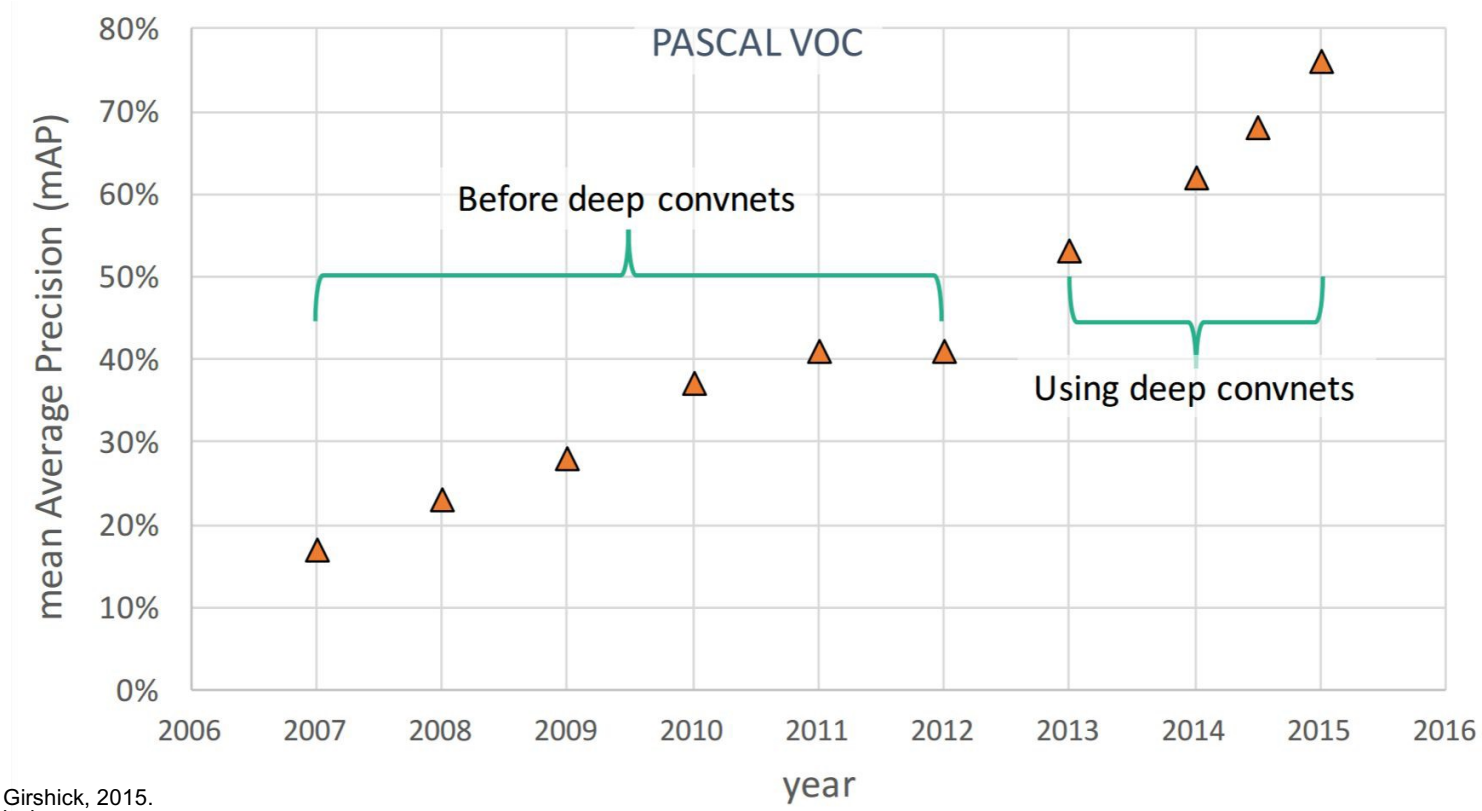
Huang et al, "Speed/accuracy trade-offs for modern convolutional object detectors", CVPR 2017

R-FCN: Dai et al, "R-FCN: Object Detection via Region-based Fully Convolutional Networks", NIPS 2016
Inception-V2: Ioffe and Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", ICML 2015

Inception V3: Szegedy et al, "Rethinking the Inception Architecture for Computer Vision", arXiv 2016
Inception ResNet: Szegedy et al, "Inception-V4, Inception-ResNet and the Impact of Residual Connections on Learning", arXiv 2016

MobileNet: Howard et al, "Efficient Convolutional Neural Networks for Mobile Vision Applications", arXiv 2017

Object Detection: Impact of Deep Learning



Other Computer Vision Tasks

Semantic Segmentation

GRASS, CAT, TREE, SKY

No objects, just pixels

Classification + Localization

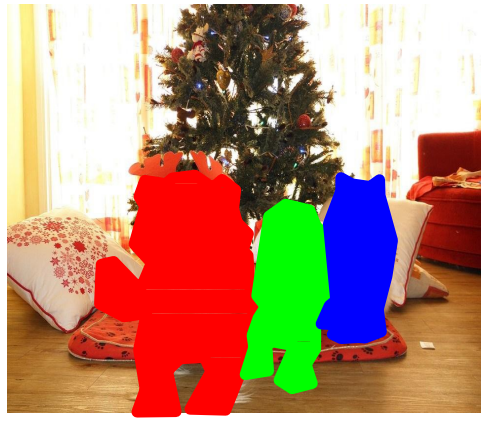
CAT

Single Object

Object Detection

DOG, DOG, CAT

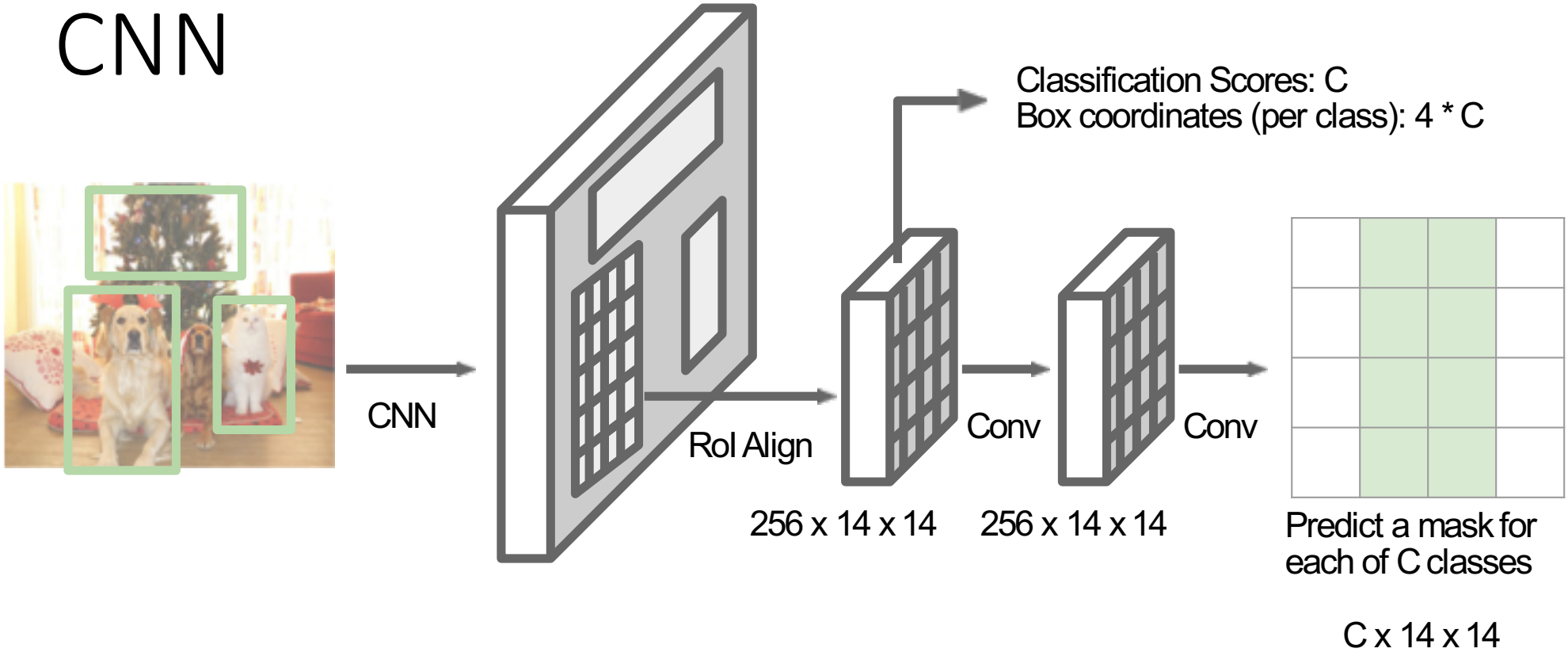
Instance Segmentation



DOG, DOG, CAT

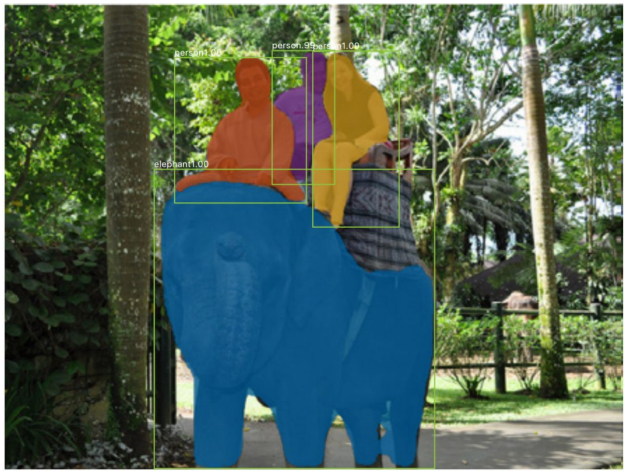
Multiple Object

Mask R-CNN



He et al, “Mask R-CNN”, arXiv 2017

Mask R-CNN: Very Good Results!



He et al, “Mask R-CNN”, arXiv 2017

Mask R-CNN
Also does pose

He et al, “Mask R-CNN”, arXiv 2017


Q&A