



EECS 230 Deep Learning

Lecture 8: Image Segmentation

From last lecture: Convolutional Layer

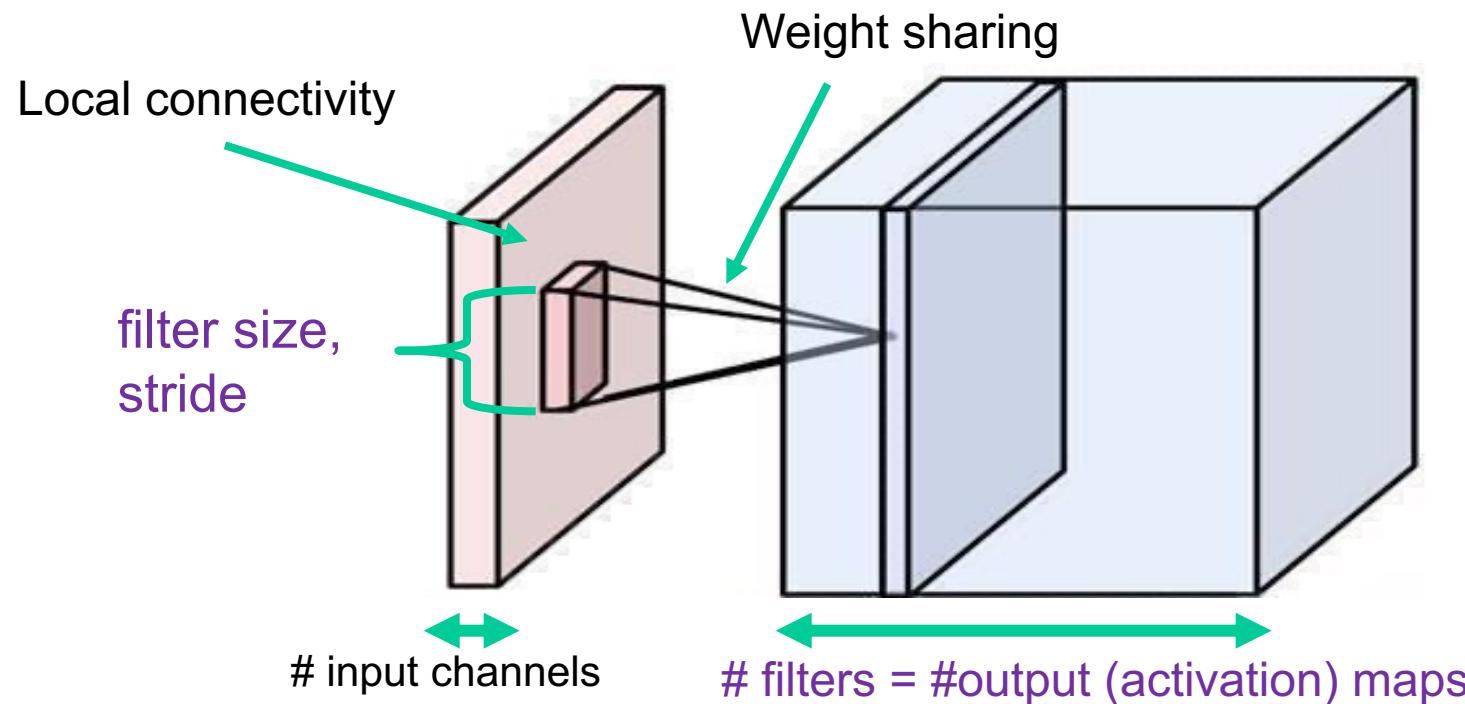
Local connectivity

Weight sharing

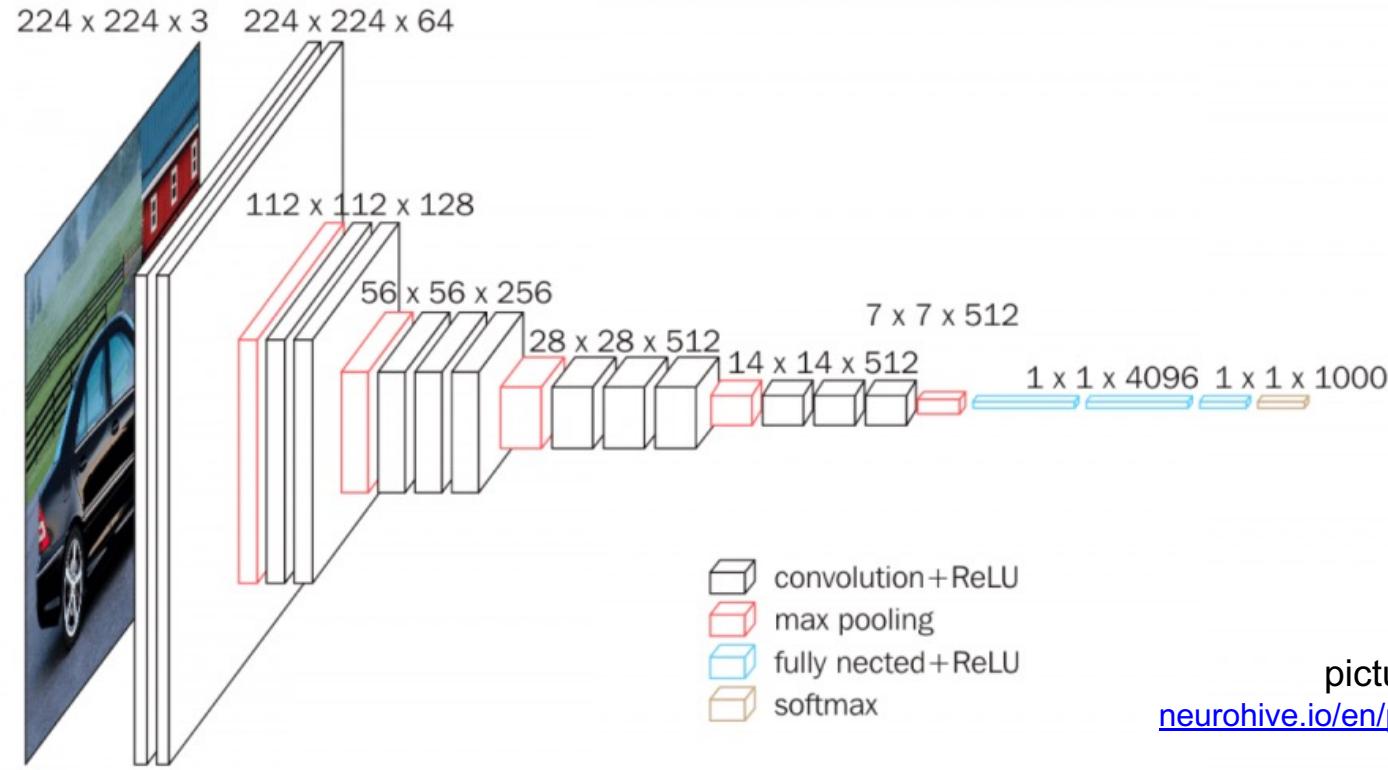
Handling multiple input/output channels

Retains location associations

Transforms 3D tensor into 3D tensor



From last lecture: CNN e.g. VGG -16



picture credits
neurohive.io/en/popular-networks/vgg16/

Input →

Conv 1-1
Conv 1-2
Pooing

Conv 2-1
Conv 2-2
Pooing

Conv 3-1
Conv 3-2
Conv 3-3
Pooing

Conv 4-1
Conv 4-2
Conv 4-3
Pooing

Conv 5-1
Conv 5-2
Conv 5-3
Pooing

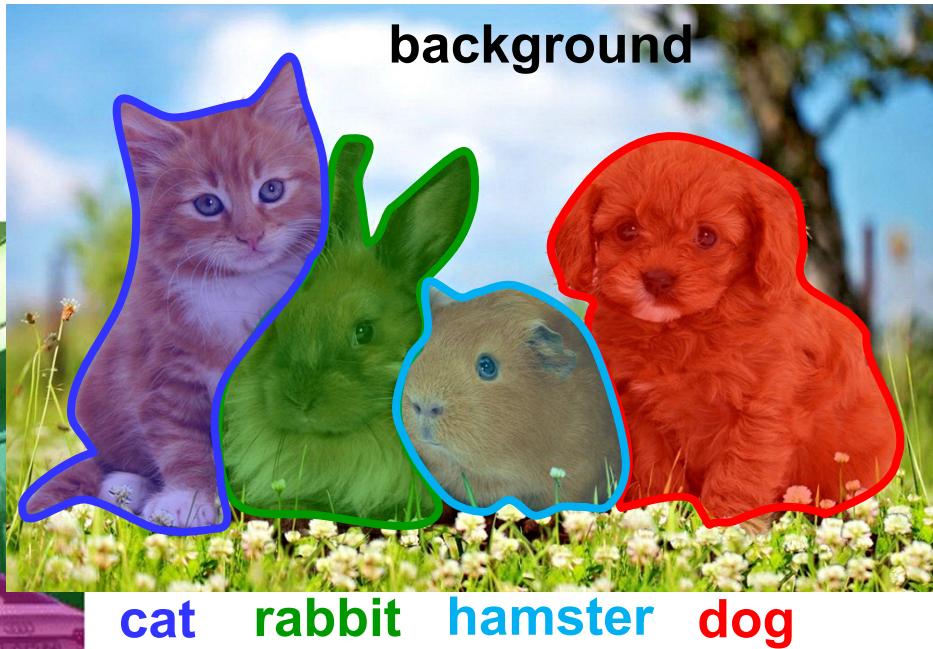
Dense
Dense
Dense

→ Output



CNN for Image Segmentation

Semantic Segmentation



Semantic Segmentation (outline)

- From image labeling to **pixel labeling**
- Architectures
- Training loss function
- Evaluation metrics
- Weakly supervised segmentation

input



learn to
predict

remember last topic:

image classification

somewhere in the image
there is a **bicycle** and a **person**

image **tags**
(image-level labels)

Semantic Segmentation

input



learn to
predict



pixel-level labels

person
bicycle
background

Fully-supervised Semantic Segmentation

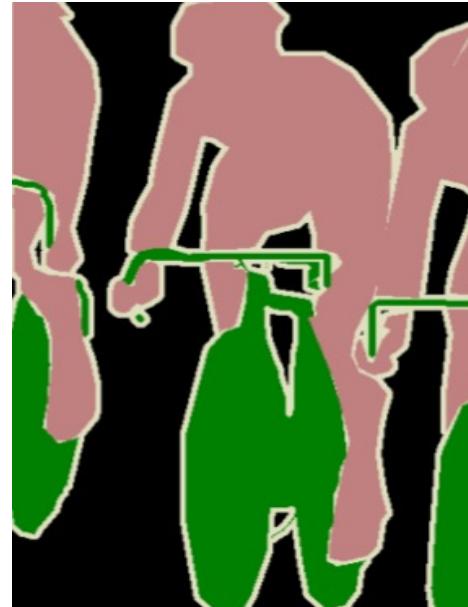
training uses pixel-accurate Ground Truth

input



learn to
predict

target (GT mask)



pixel-level labels

person
bicycle
background

Fully-supervised Semantic Segmentation

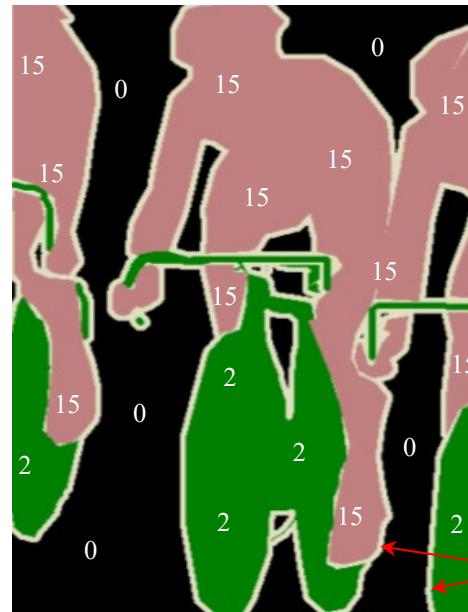
training uses pixel-accurate Ground Truth

input



learn to
predict

target (GT mask)



pixel-level labels

person
bicycle
background

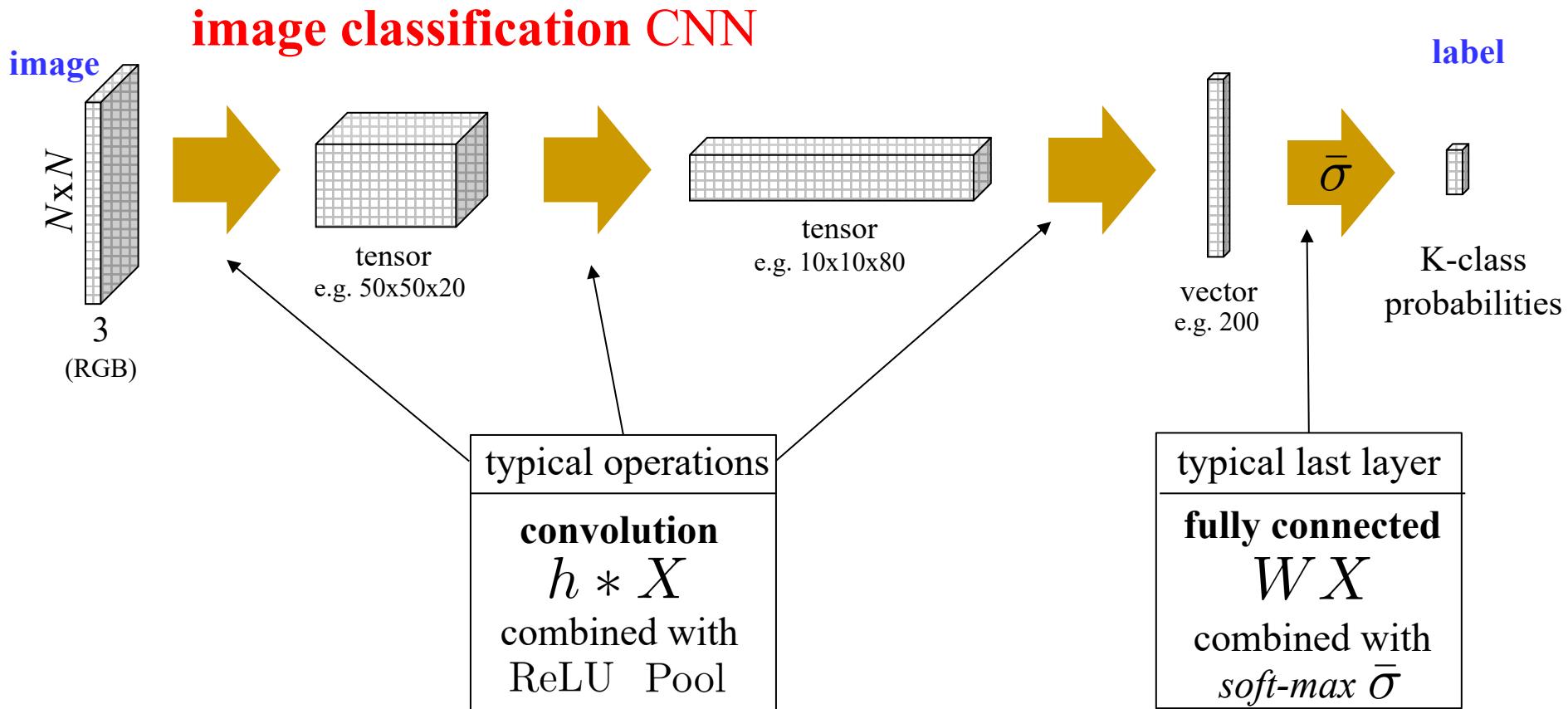
255 (void/undefined)

$\mathbf{y}^p \in [0, 1, 2, 3, \dots]$ - class label at each pixel p

pixel labels (object classes) used in Pascal dataset:

- 0 - *background*
- 1-20 - airplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, dining table, dog, horse, motorbike, person, potted plant, sheep, sofa, train, TV monitor
- 255 - *void* (class for pixel is undefined)

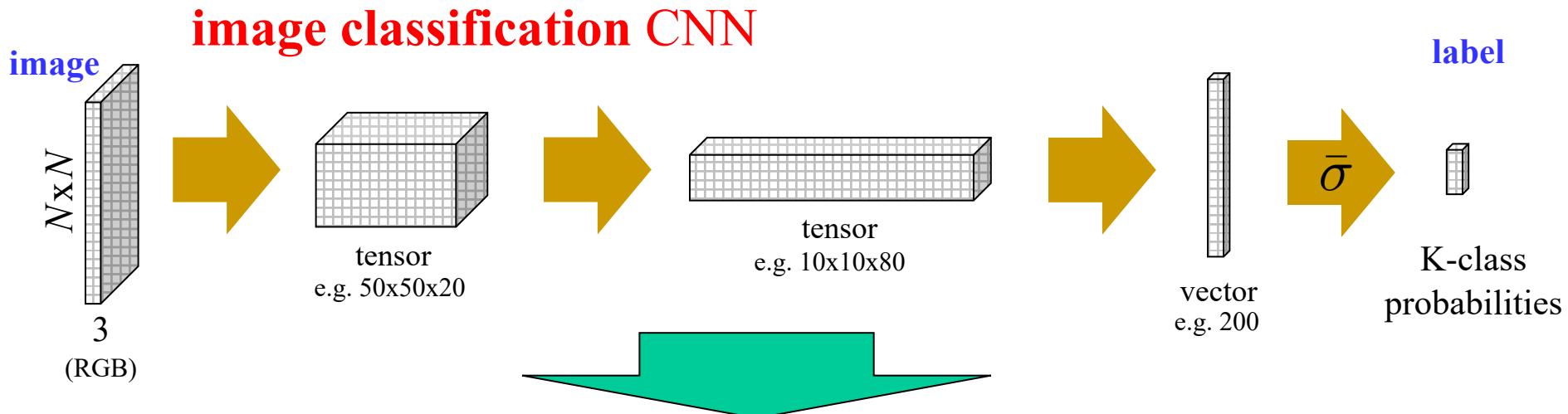
From Image to Pixel Labeling



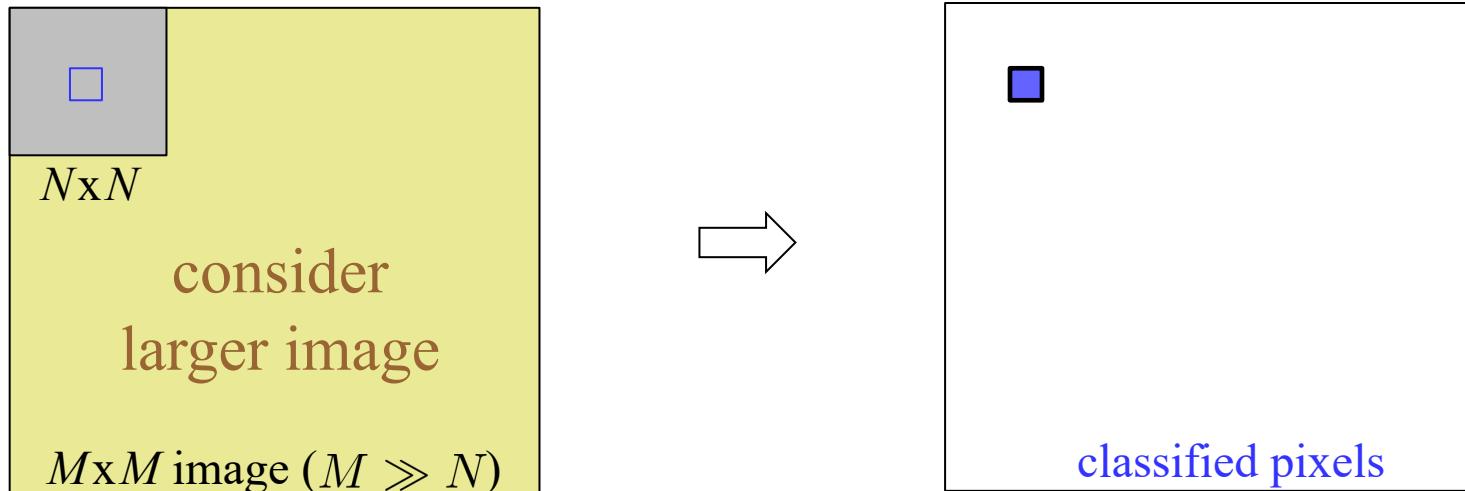
Q: How do we go from here to image segmentation?

That is, how to extend NN methods for image classification to **classification of image pixels** ?

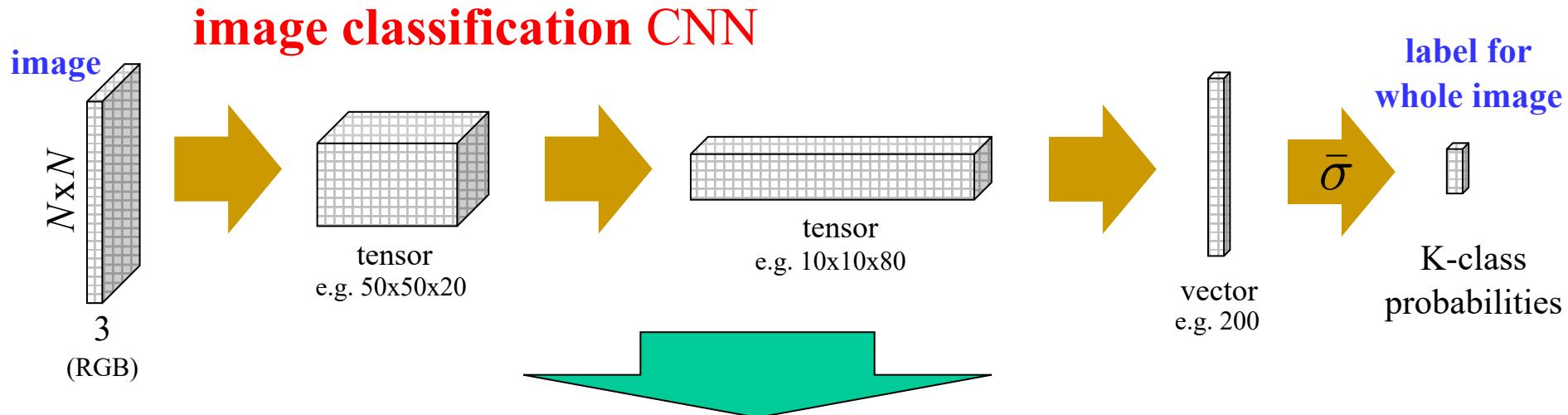
From Image to Pixel Labeling



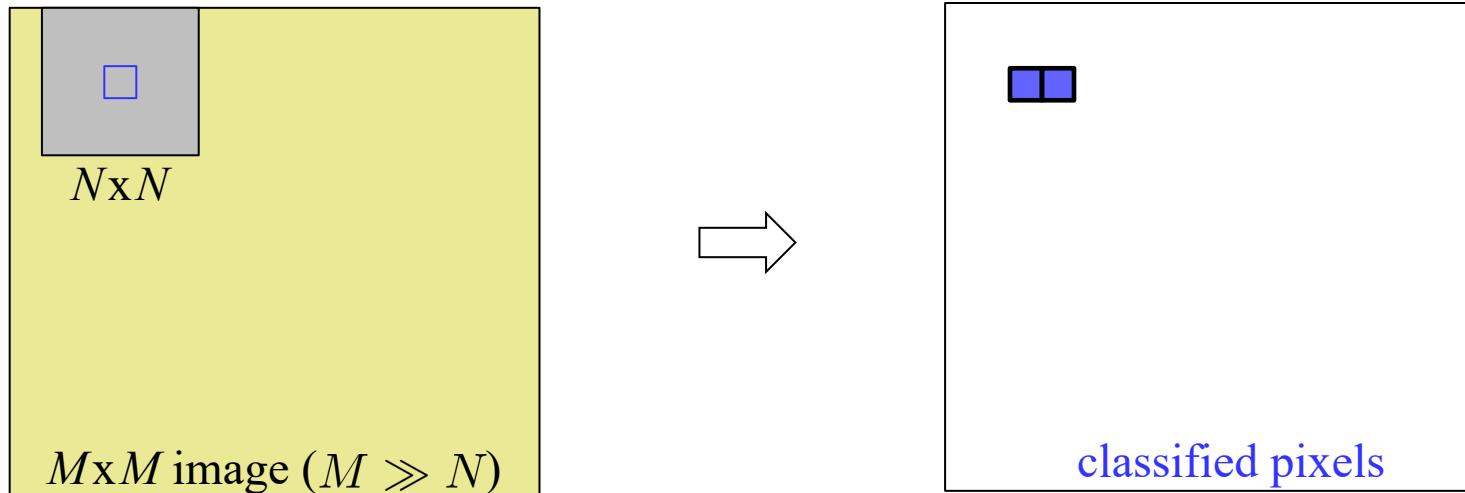
First (naïve) idea: classify pixels using *sliding windows*



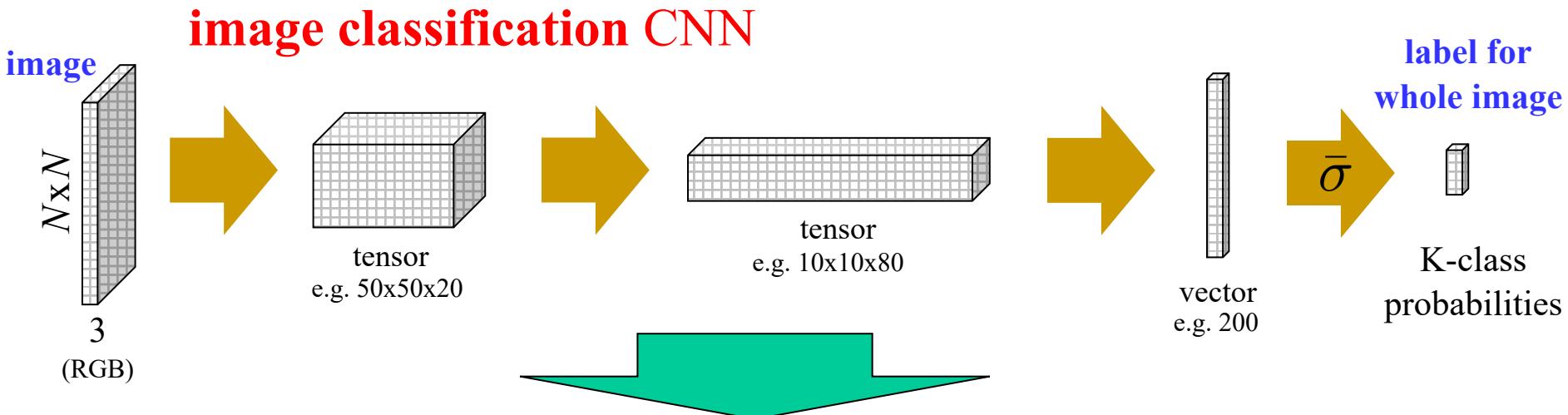
From Image to Pixel Labeling



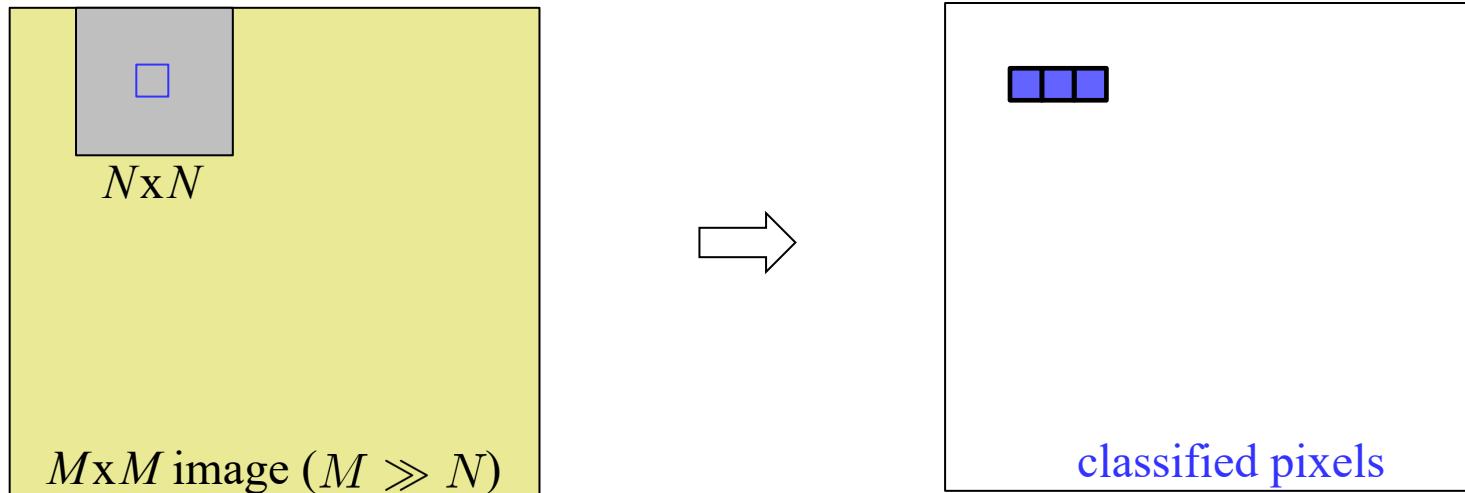
First (naïve) idea: classify pixels using *sliding windows*



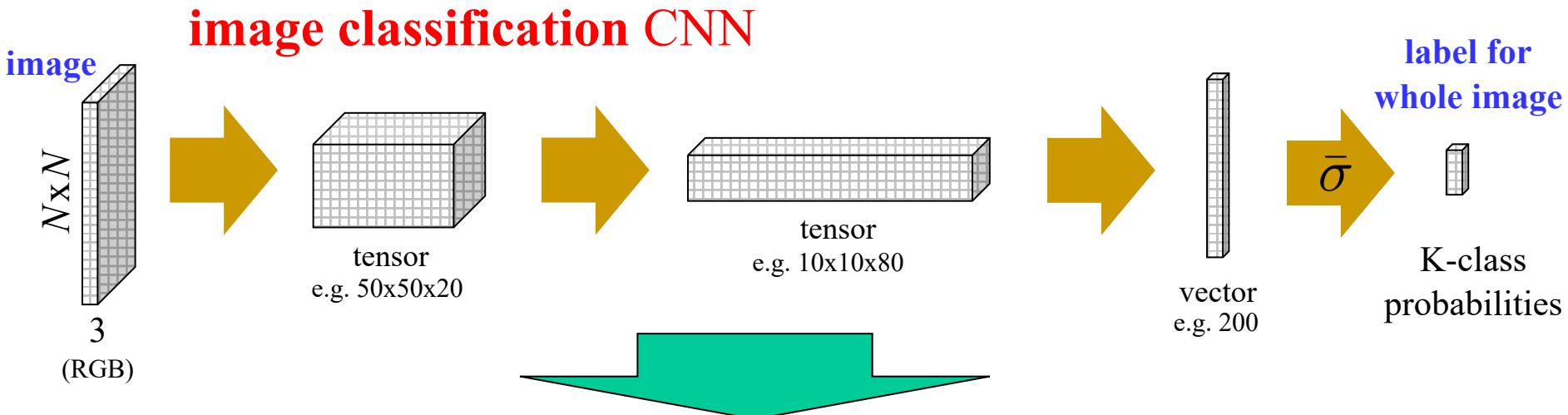
From Image to Pixel Labeling



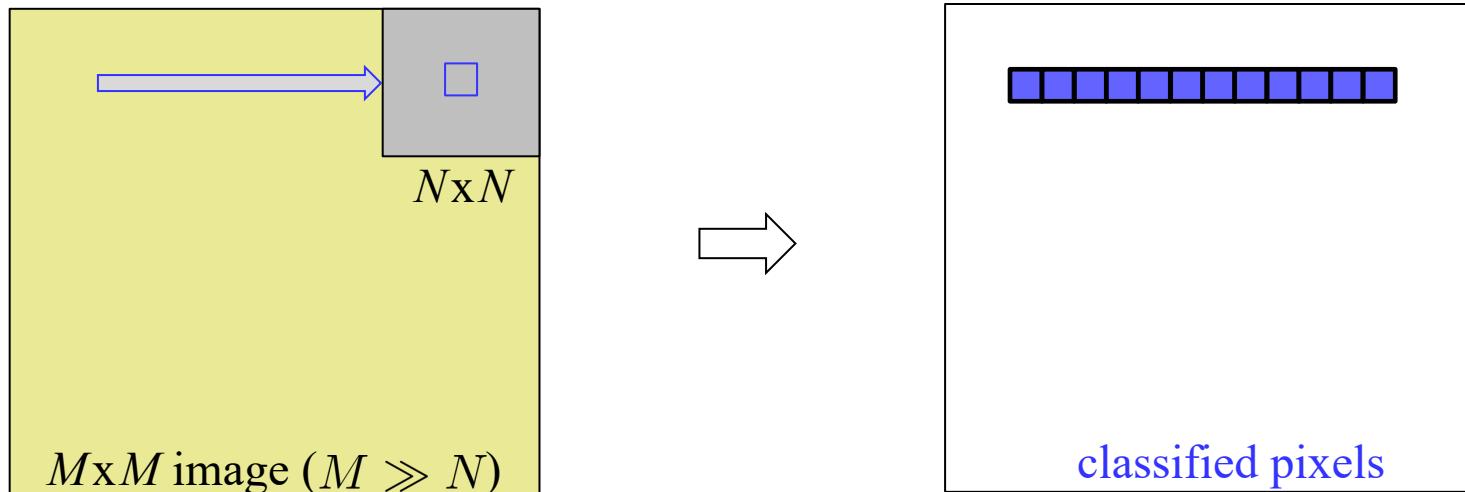
First (naïve) idea: classify pixels using *sliding windows*



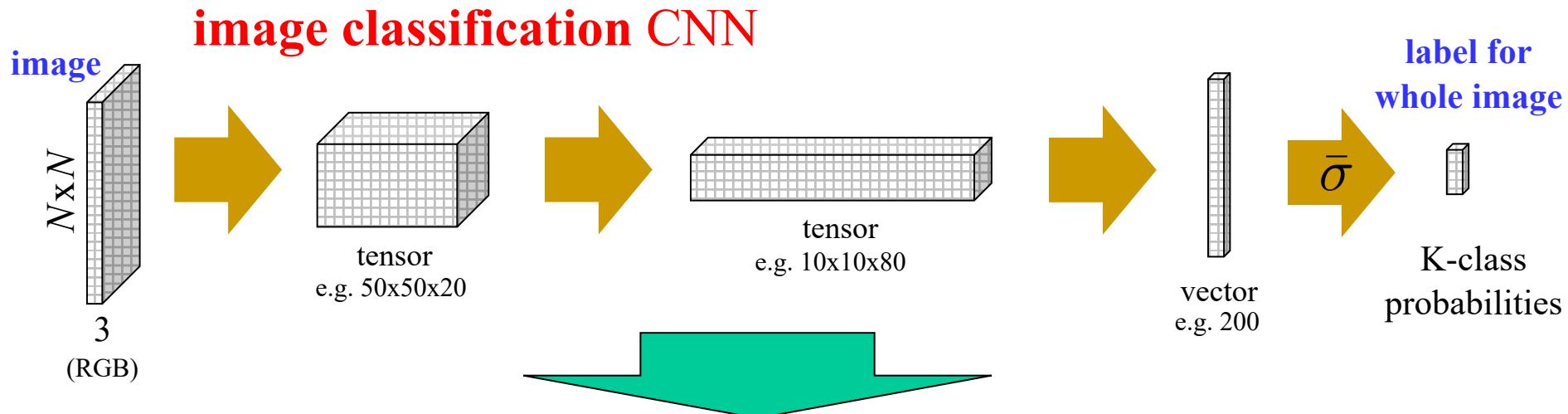
From Image to Pixel Labeling



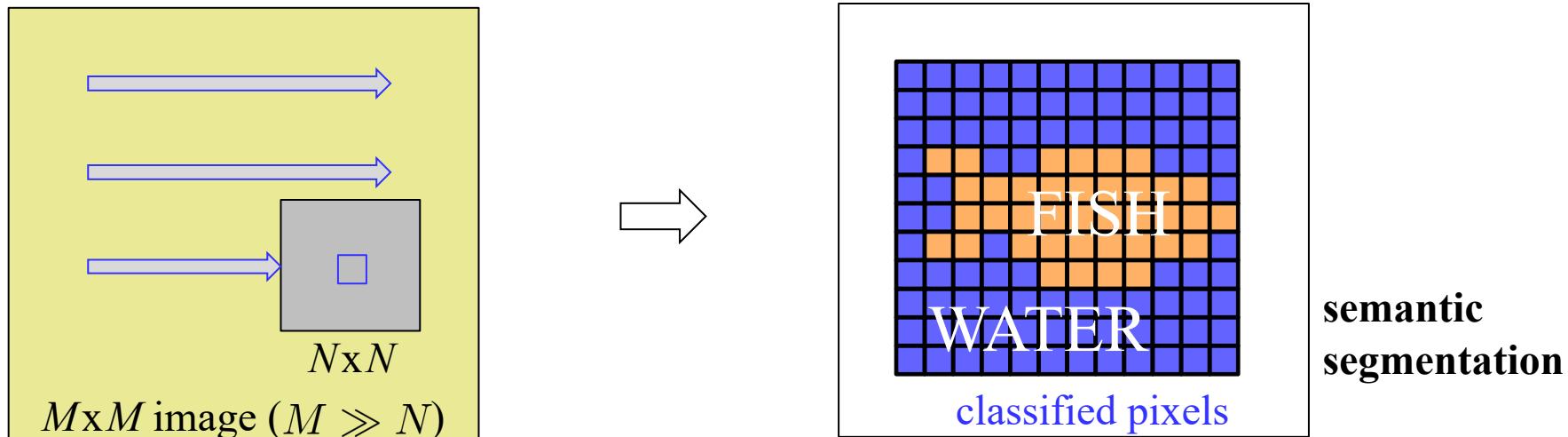
First (naïve) idea: classify pixels using *sliding windows*



From Image to Pixel Labeling



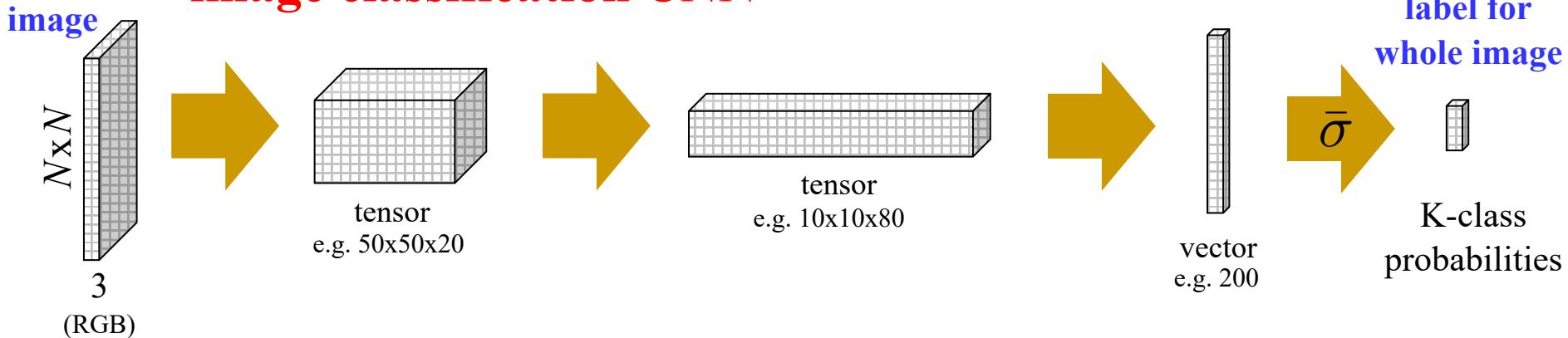
First (naïve) idea: classify pixels using *sliding windows*



Not bad for a start, but pixels are classified independently (one-at-a-time). For example, such one-pixel classifying network can NOT learn spatial pattern of the whole GT segmentation mask.

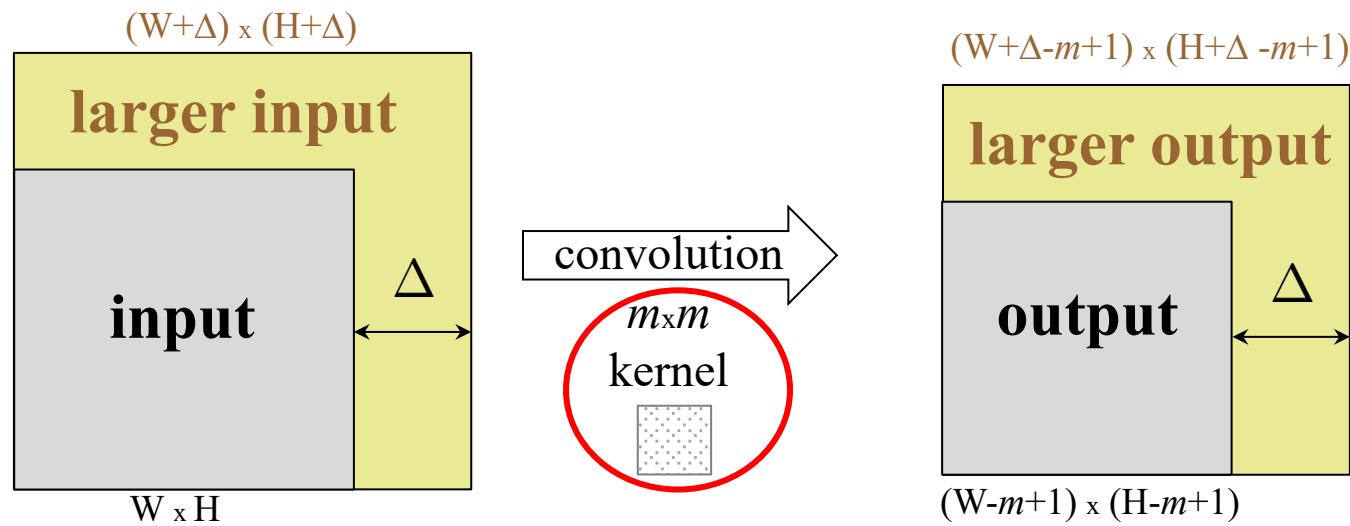
From Image to Pixel Labeling

image classification CNN



Better idea: convolutional kernel can be applied to input of any size!

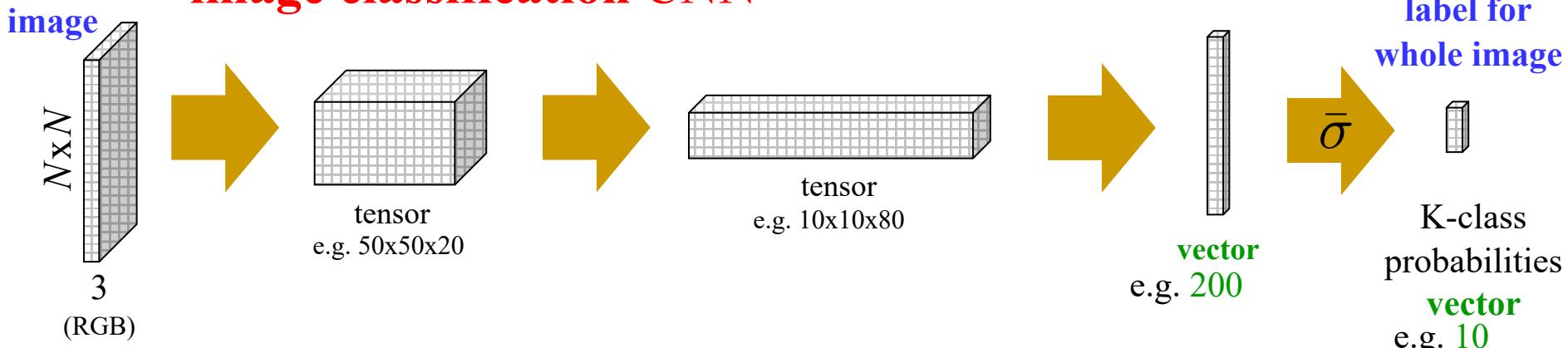
Key insight:



using the same kernel

From Image to Pixel Labeling

image classification CNN



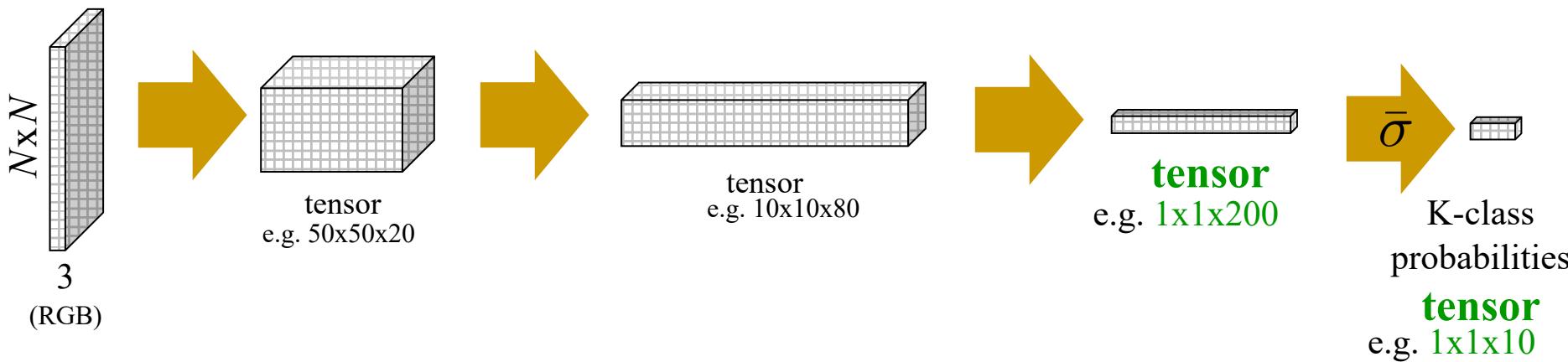
Better idea: convolutional kernel can be applied to input of any size!

Assume all layers are convolutional.

What about last (fully connected) layer?

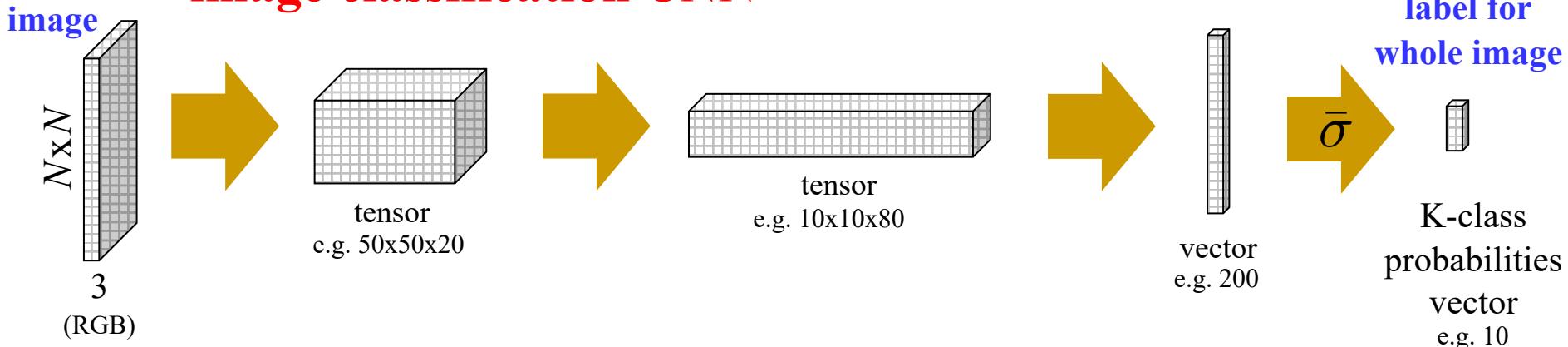
No problem:

$$W_{10 \times 200} X \equiv h_{1 \times 1}^{200 \rightarrow 10} * X$$



From Image to Pixel Labeling

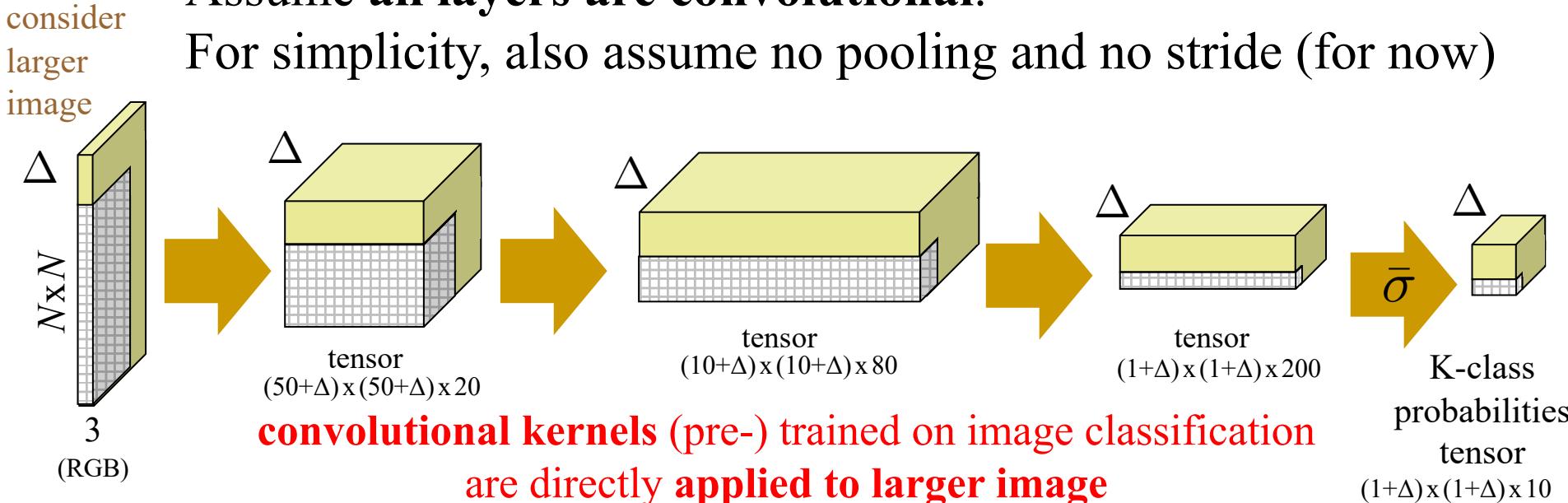
image classification CNN



Better idea: convolutional kernel can be applied to input of any size!

Assume **all layers are convolutional**.

For simplicity, also assume no pooling and no stride (for now)

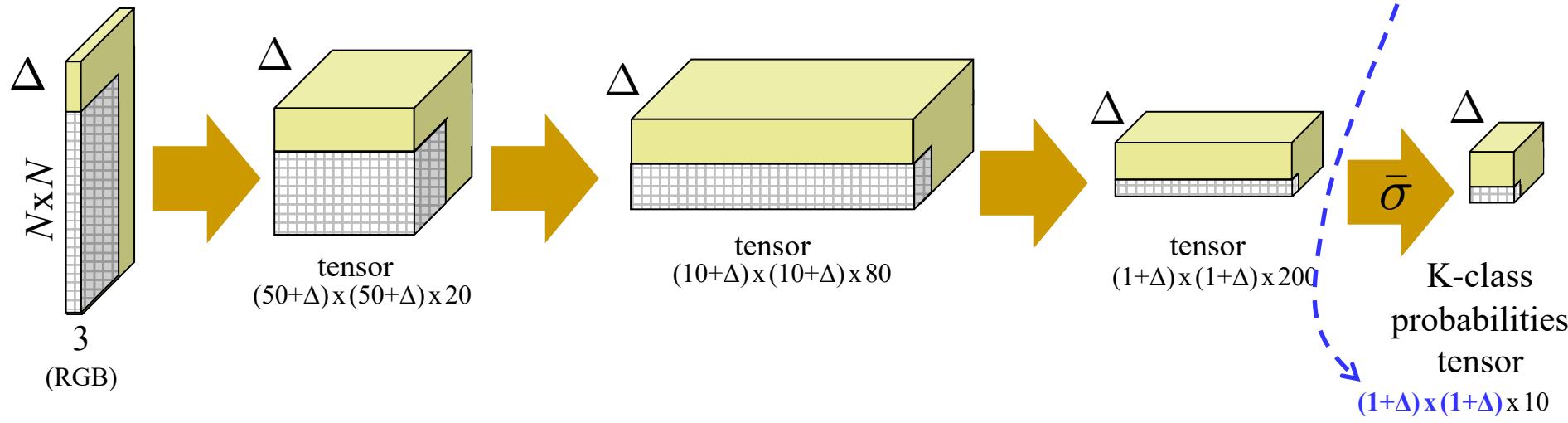


**convolutional kernels (pre-) trained on image classification
are directly applied to larger image**

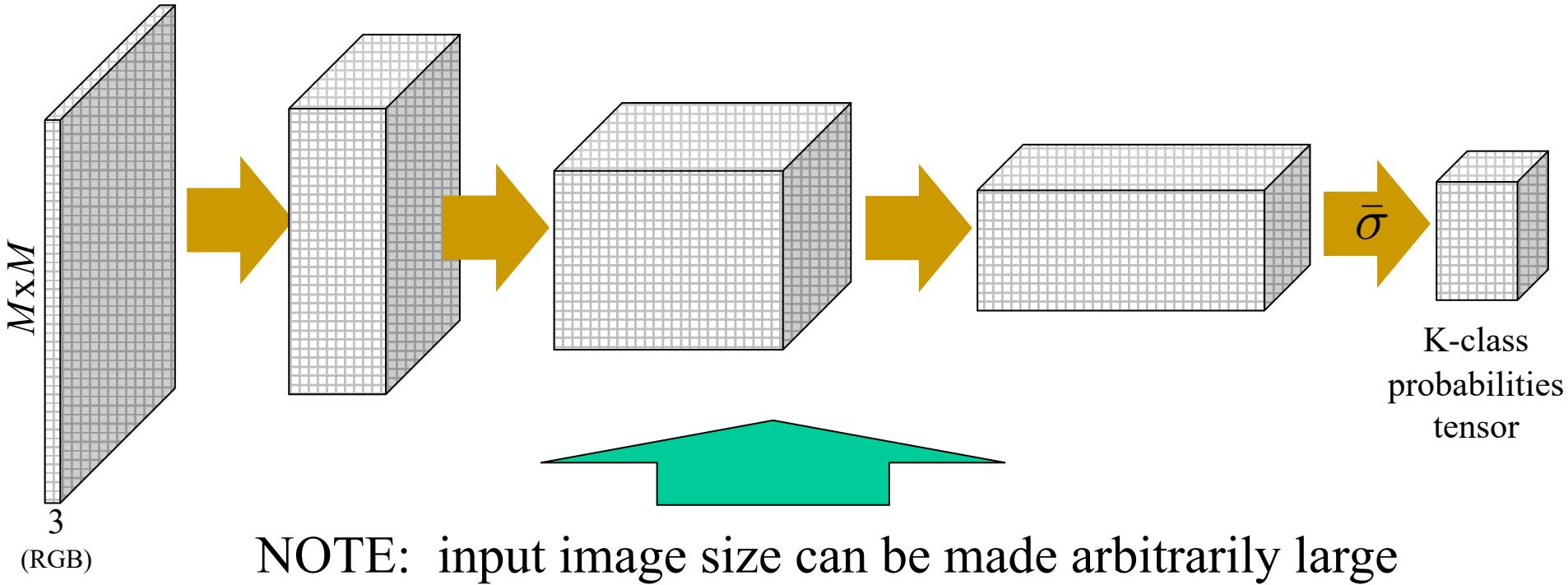
From Image to Pixel Labeling

Now, network output has some spatial resolution!

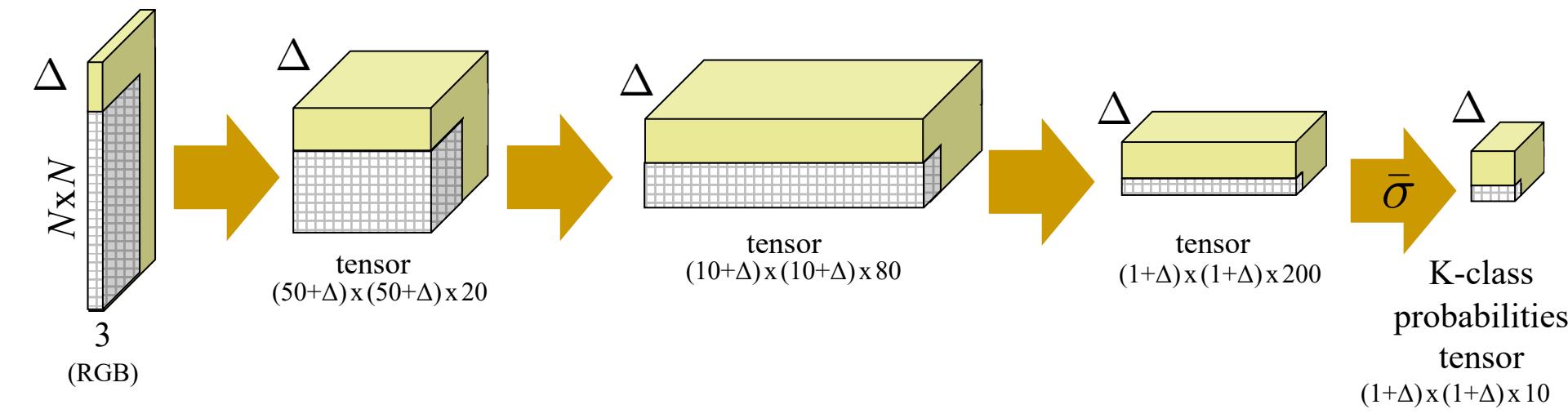
Intuition: K-class probabilities in the gray part of the output have “*receptive field*” in the gray part of the input image, while yellow output is supported by different $N \times N$ sections of the larger image



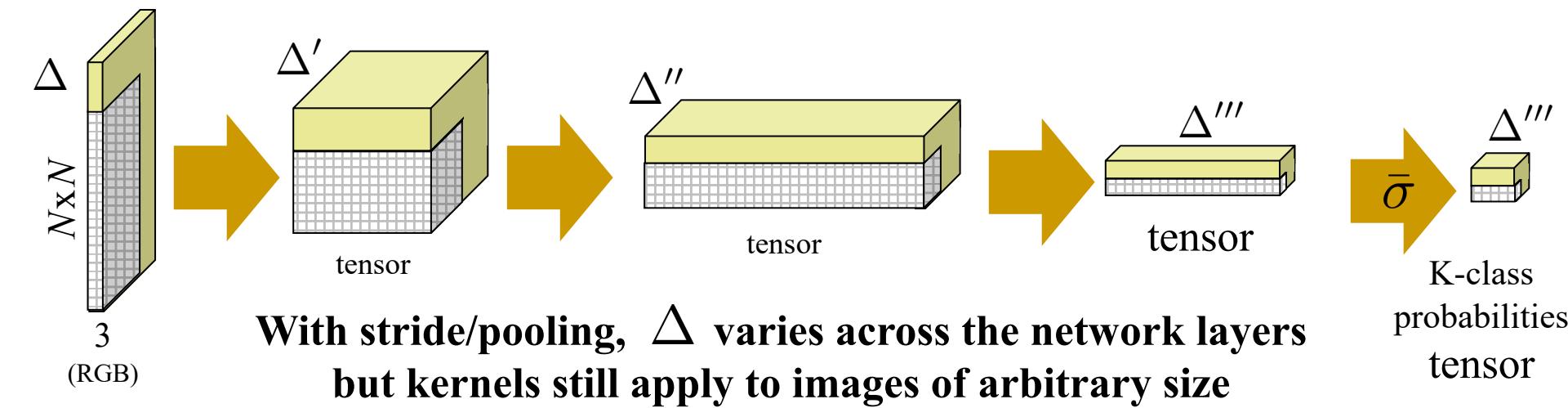
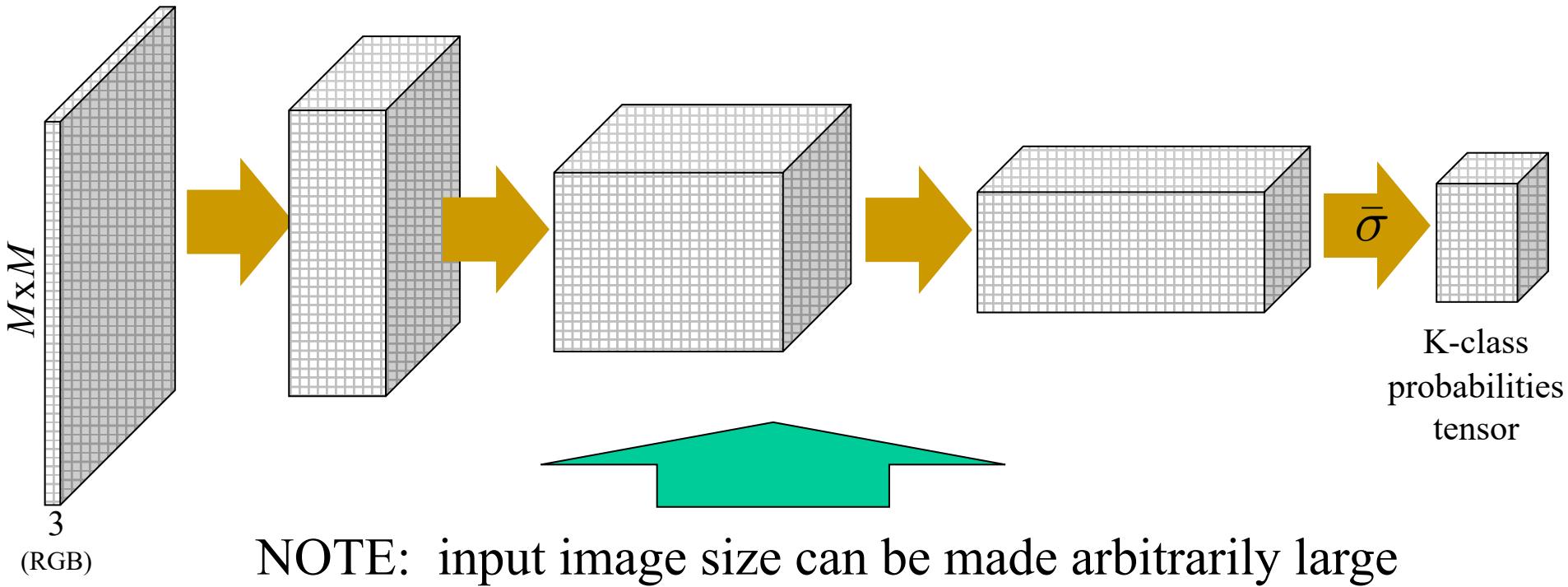
Fully Convolutional Network (FCN)



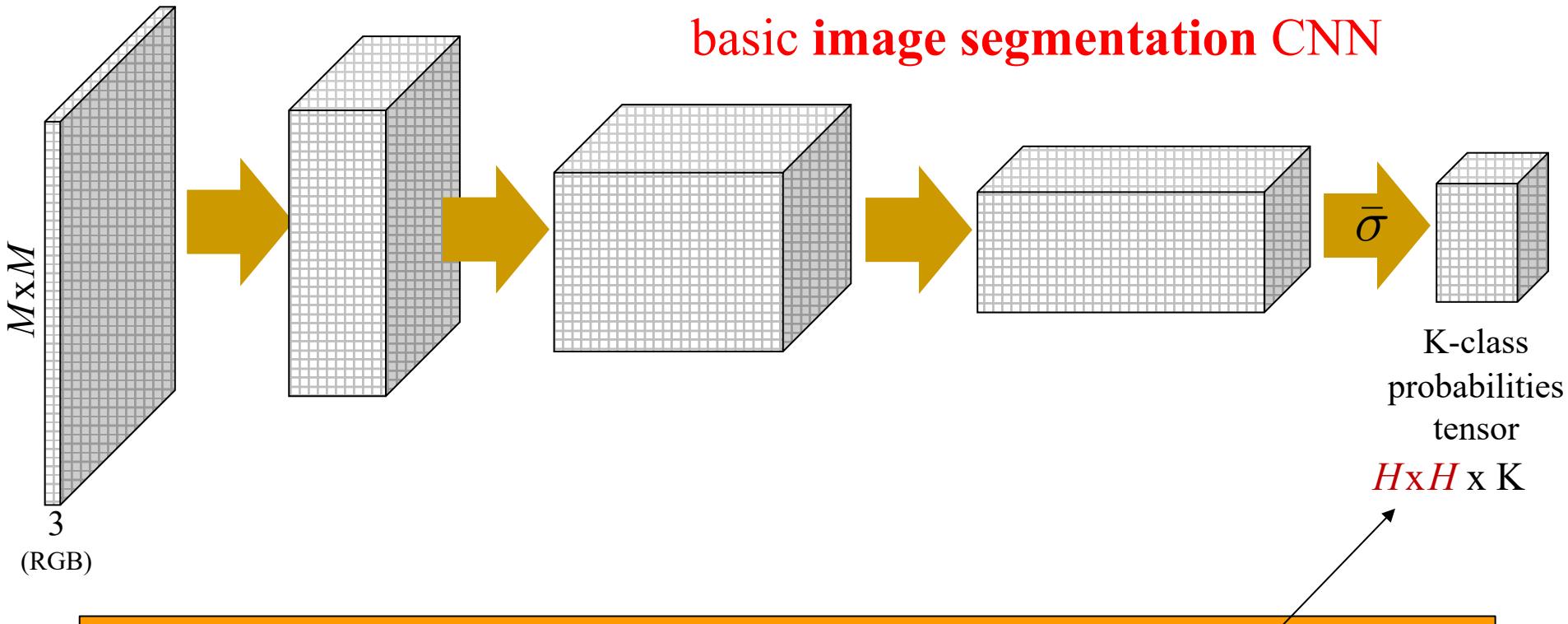
NOTE: input image size can be made arbitrarily large



Fully Convolutional Network (FCN)



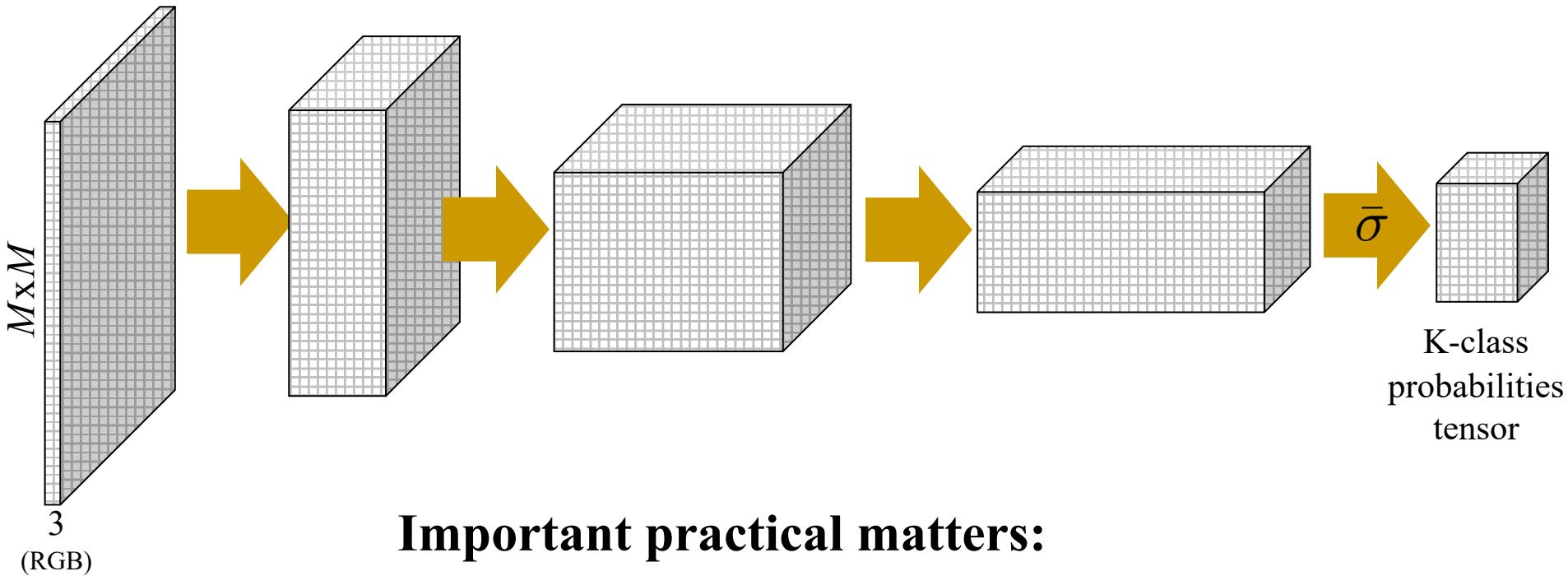
Fully Convolutional Network (FCN)



NOTE: since this network's prediction/output has spatial resolution,
it can be trained directly using (**whole**) segmentation masks/targets
(unlike our earlier naïve one-pixel classifying network)

Our first “proper” segmentation CNN
end-to-end trainable for image segmentation task

Fully Convolutional Network (FCN)

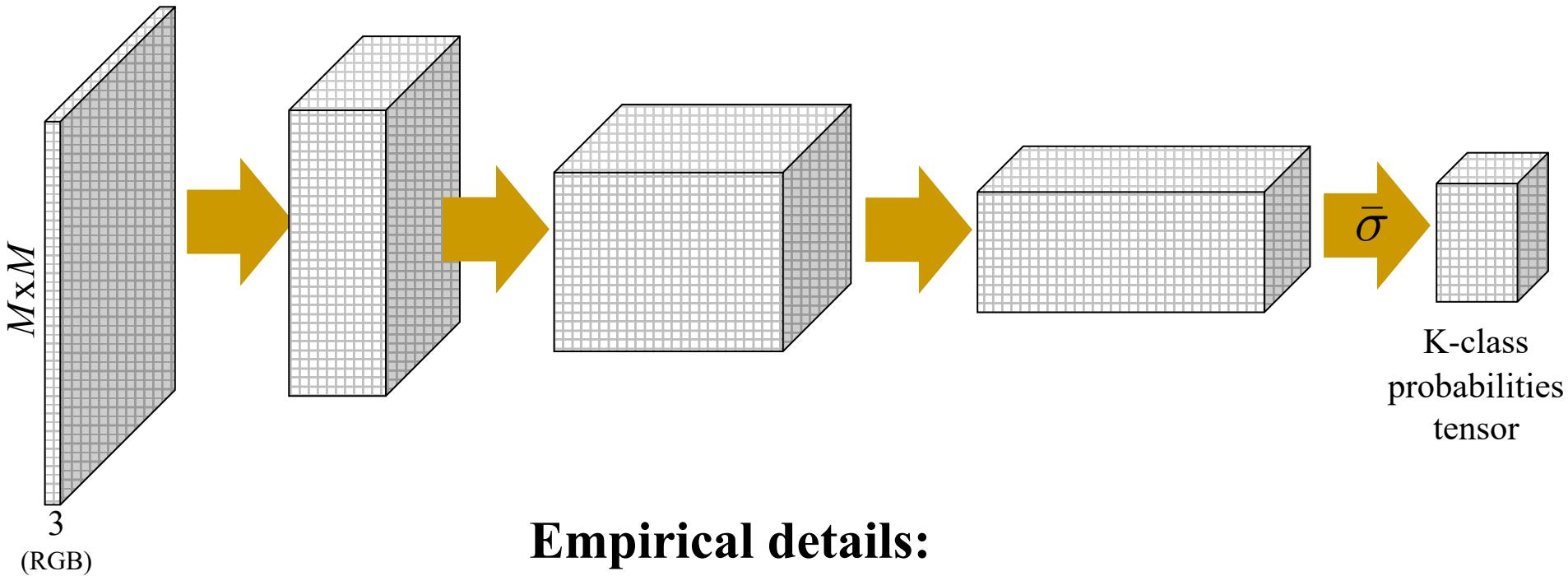


Important practical matters:

FCN can be initialized from network (kernels) **pre-trained on huge image classification training datasets** (e.g. *ResNet* trained on *image net*) learning good high-dimensional features (embedding) at later layers

Then can be **re-trained** (*domain adaptation*) to any specific segmentation dataset **based on full segmentation masks** (targets)

Fully Convolutional Network (FCN)



Empirical details:

works better (after re-training) with **pooling, stride, dilation** giving wider “*receptive field*” for output layer elements/pixels

... even though such operations generally decrease output resolution therefore, requiring **output up-sampling** to improve it

Popular CNN architectures for segmentation

- **FCN** (2015)

fully convolutional network for segmentation

skip connections

various ideas/details on
pooling, stride, dilation
and **upsampling**

Fully Convolutional Networks for Semantic Segmentation
Long, Shelhamer, Darrell - CVPR 2015

- **SegNet** (2015)

encoder / decoder

Segnet: A deep convolutional encoder-decoder architecture for image segmentation

Badrinarayanan, Kendall, Cipolla – TPAMI 2017

- **UNet** (2015)

encoder / decoder with symmetric skip connections

U-net: Convolutional networks for biomedical image segmentation
Ronneberger, Fischer, Brox - MICCAI 2015 / Nature Methods 2019

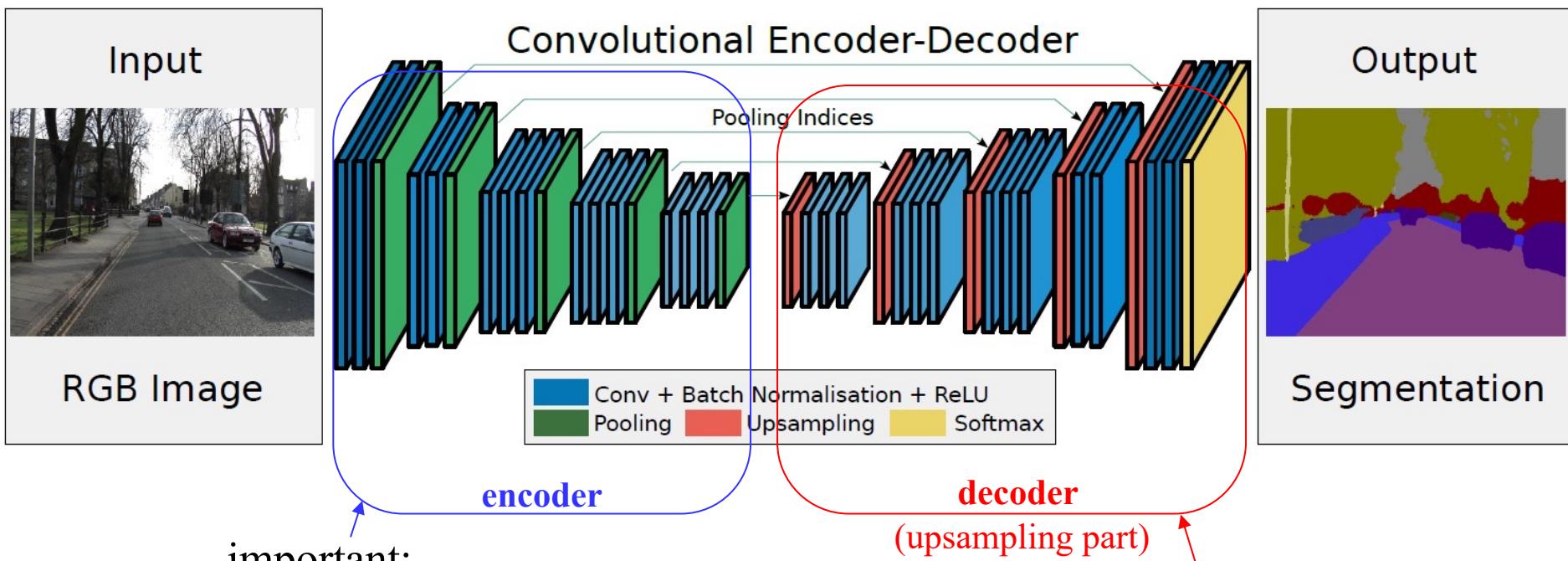
- **DeepLab** (2015)

atrous convolutions, spatial pyramid pooling, etc.

DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolutions, and Fully Connected CRFs
Chen, Papandreou, Kokkinos, Murphy, Yuille – TPAMI 2018 / ICLR 2015

Common Structure: Encoder/Decoder

Segnet: A deep convolutional encoder-decoder architecture for image segmentation
Badrinarayanan, Kendall, Cipolla – TPAMI 2017



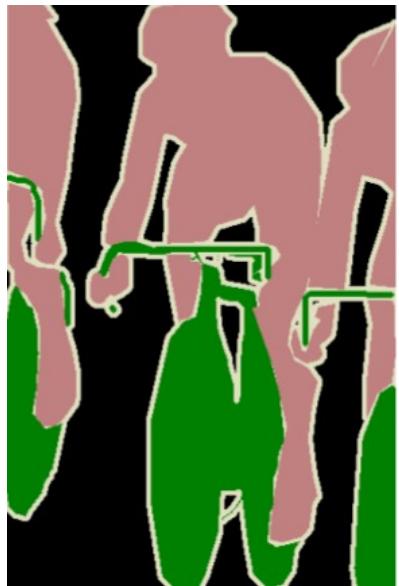
important:

encoder convolutional layers are typically pre-trained on *image net*

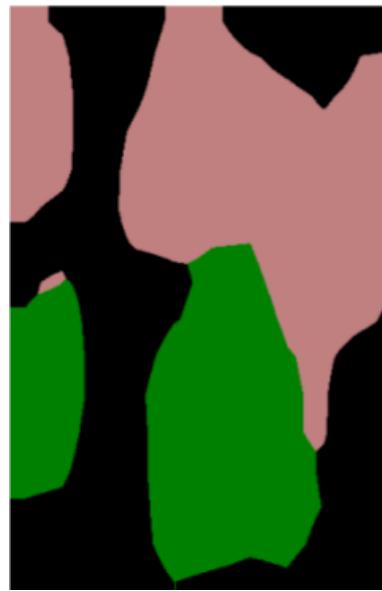
decoder upsamples encoder-generated features

Need for upsampling

Ground truth target



Predicted segmentation

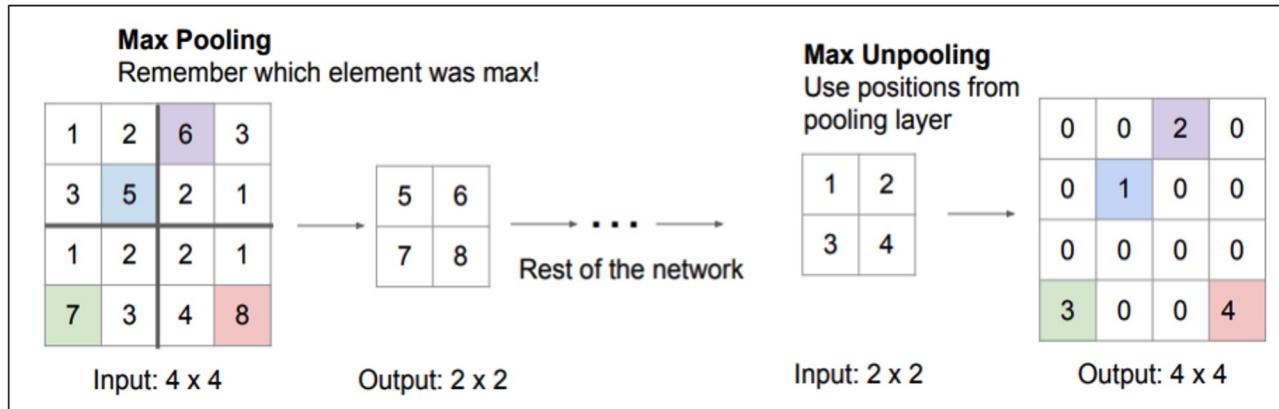
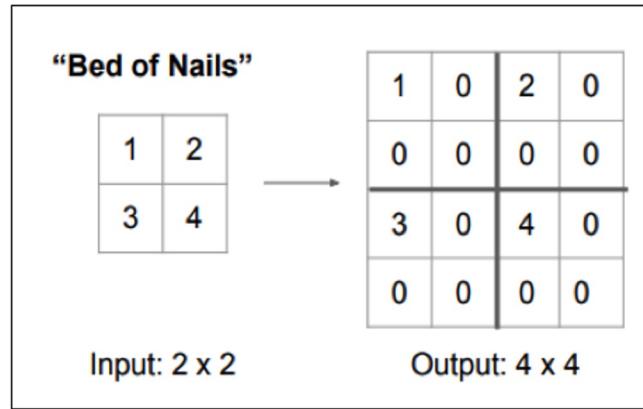
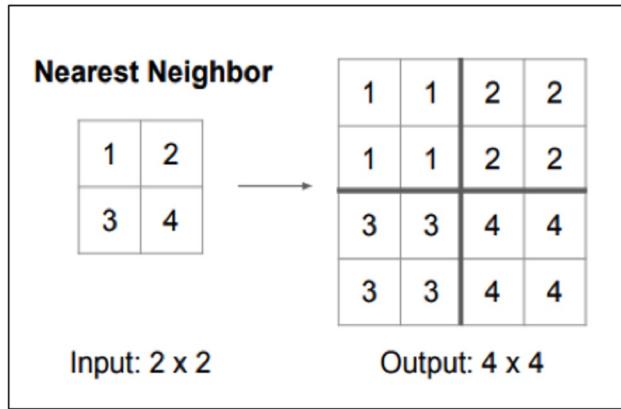


soft-max applied directly to
encoder's output features

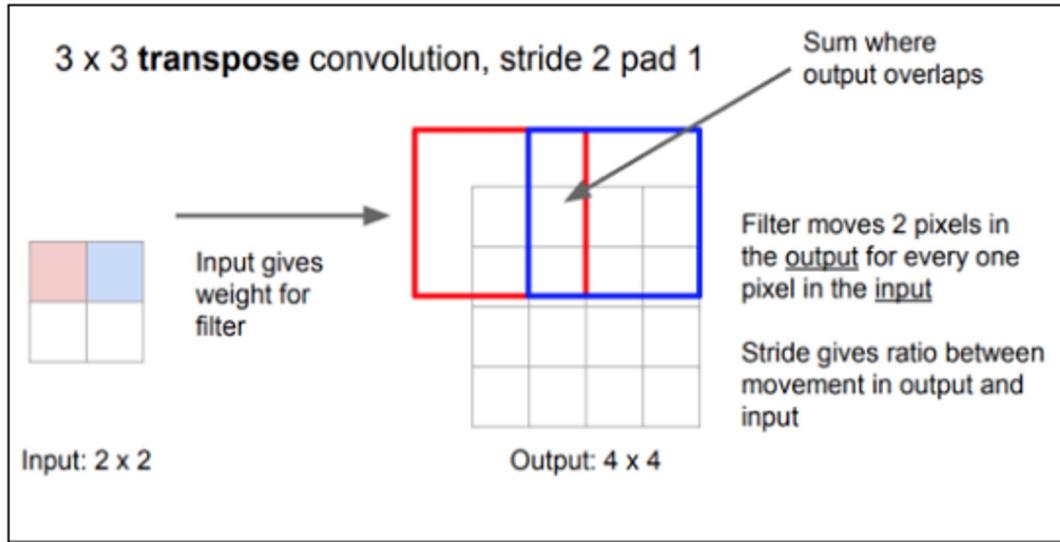
Primary goal of the **decoder** is (to learn) **to upsample**

Methods for Upsampling

illustrations credit: Fei-Fei Li

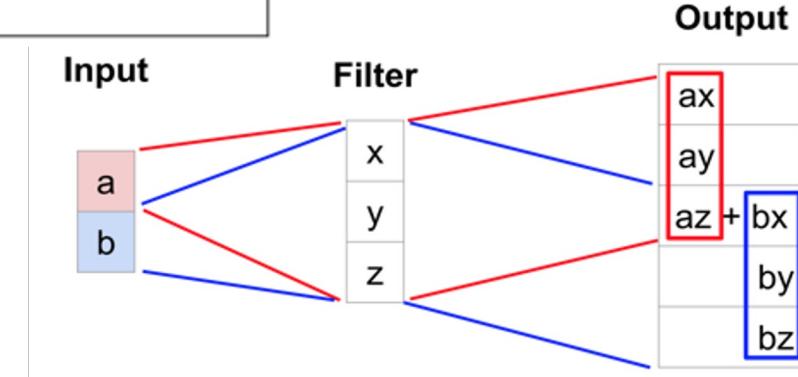


Methods for Upsampling



illustrations credit: Fei-Fei Li

Simpler 1D illustration:



Weights for such **transpose convolution** kernel (filter) **can be learned**.

Why should transpose convolution work well for upsampling?

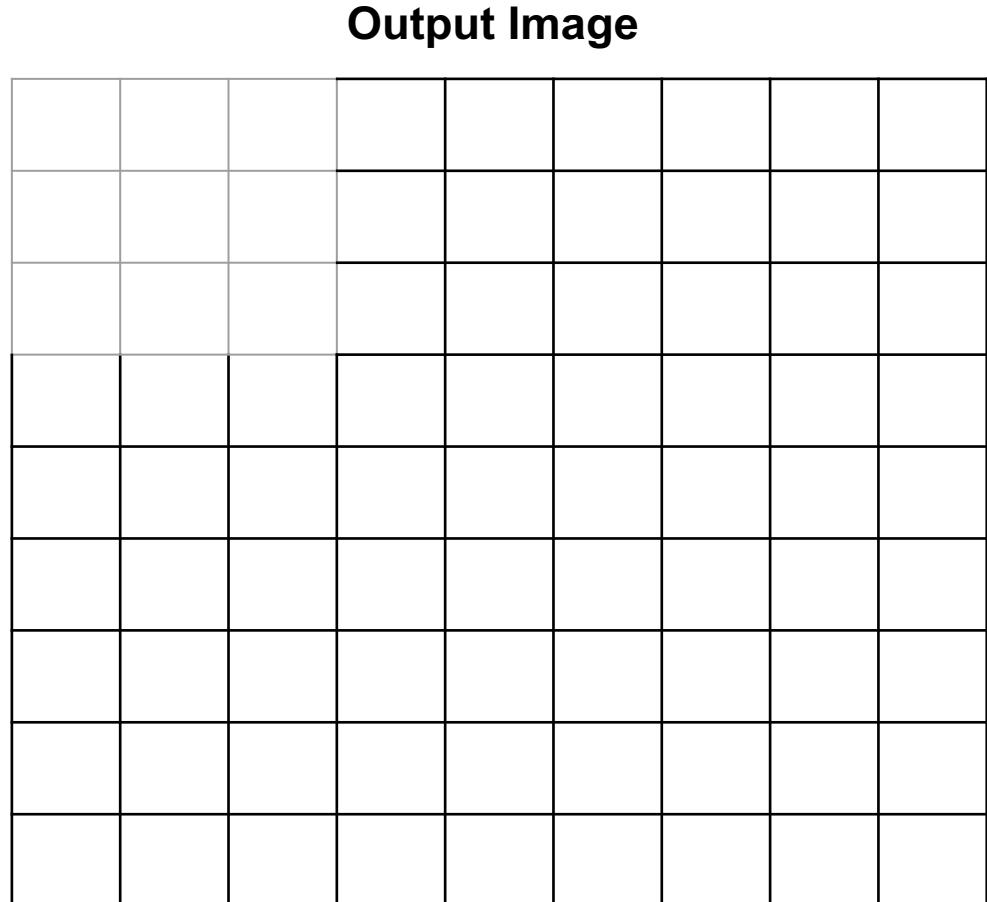
Transpose Convolution: Example

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

Kernel		
0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

kernel=3x3
stride=2
padding=1



Transpose Convolution: Example

First Element x Kernel

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

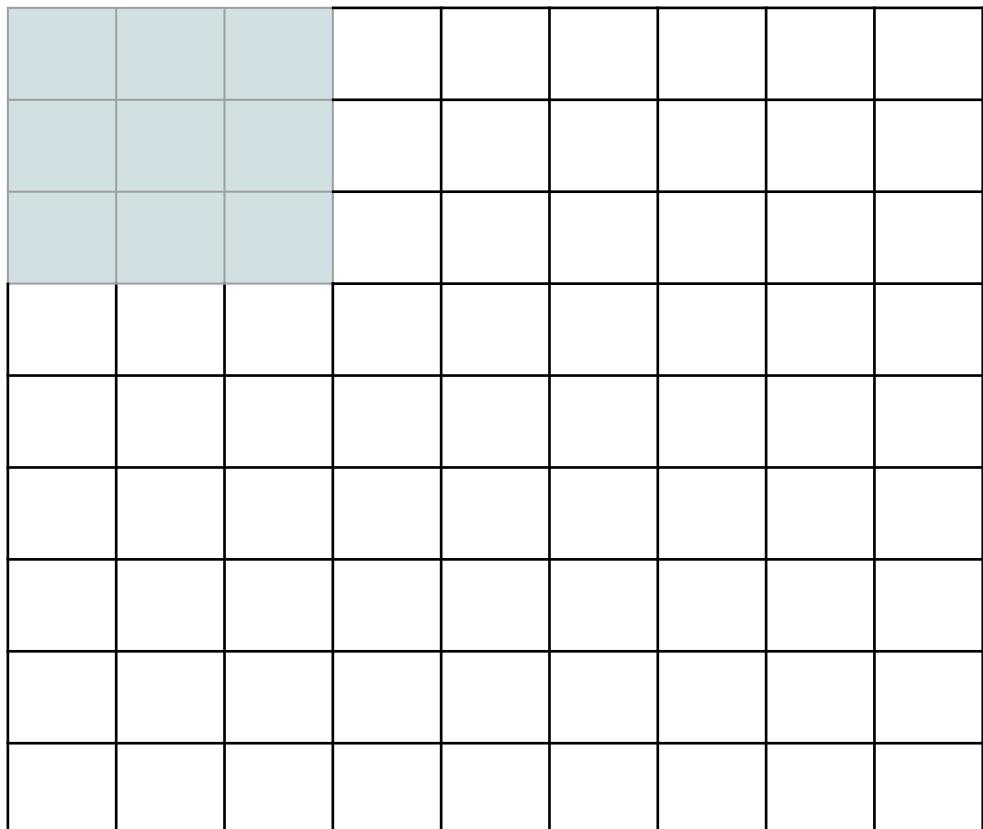
0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

0	0	0
0	0	0
0	0	0

kernel=3x3
stride=2
padding=1

Output Image



Transpose Convolution: Example

Added Result

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

0	0	0
0	0	0
0	0	0

kernel=3x3
stride=2
padding=1

Output Image

0	0	0						
0	0	0						
0	0	0						

Transpose Convolution: Example

Next Element x Kernel

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

kernel=3x3
stride=2
padding=1

Output Image

0	0	0						
0	0	0						
0	0	0						

Transpose Convolution: Example

Added Result

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

Kernel

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

kernel=3x3
stride=2
padding=1

Output Image

0	0	0.25	0.5	0.25				
0	0	0.5	1	0.5				
0	0	0.25	0.5	0.25				

Transpose Convolution: Example

Next Element x Kernel

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

0.5	1	0.5
1	2	1
0.5	1	0.5

kernel=3x3
stride=2
padding=1

Output Image

0	0	0.25	0.5	0.25				
0	0	0.5	1	0.5				
0	0	0.25	0.5	0.25				

Transpose Convolution: Example

Added Result

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

0.5	1	0.5
1	2	1
0.5	1	0.5

kernel=3x3
stride=2
padding=1

Output Image

0	0	0.25	0.5	0.75	1	0.5		
0	0	0.5	1	1.5	2	1		
0	0	0.25	0.5	0.75	1	0.5		

Transpose Convolution: Example

Next Element x Kernel

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

0.75	1.5	0.75
1.5	3	1.5
0.75	1.5	0.75

kernel=3x3
stride=2
padding=1

Output Image

0	0	0.25	0.5	0.75	1	0.5		
0	0	0.5	1	1.5	2	1		
0	0	0.25	0.5	0.75	1	0.5		

Transpose Convolution: Example

Added Result

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

Kernel

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

0.75	1.5	0.75
1.5	3	1.5
0.75	1.5	0.75

kernel=3x3
stride=2
padding=1

Output Image

0	0	0.25	0.5	0.75	1	1.25	1.5	0.75
0	0	0.5	1	1.5	2	2.5	3	1.5
0	0	0.25	0.5	0.75	1	1.25	1.5	0.75

Transpose Convolution: Example

Next Element x Kernel

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

1	2	1
2	4	2
1	2	1

kernel=3x3
stride=2
padding=1

Output Image

0	0	0.25	0.5	0.75	1	1.25	1.5	0.75
0	0	0.5	1	1.5	2	2.5	3	1.5
0	0	0.25	0.5	0.75	1	1.25	1.5	0.75

Transpose Convolution: Example

Added Result

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

1	2	1
2	4	2
1	2	1

kernel=3x3
stride=2
padding=1

Output Image

0	0	0.25	0.5	0.75	1	1.25	1.5	0.75
0	0	0.5	1	1.5	2	2.5	3	1.5
1	2	1.25	0.5	0.75	1	1.25	1.5	0.75
2	4	2						
1	2	1						

Transpose Convolution: Example

Added Result

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

1.25	2.5	1.25
2.5	5	2.5
1.25	2.5	1.5

kernel=3x3
stride=2
padding=1

Output Image

0	0	0.25	0.5	0.75	1	1.25	1.5	0.75
0	0	0.5	1	1.5	2	2.5	3	1.5
1	2	2.5	3	2	1	1.25	1.5	0.75
2	4	4.5	5	2.5				
1	2	2.5	2.5	1.5				

Transpose Convolution: Example

Added Result

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

Kernel

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

1.5	3	1.5
3	6	3
1.5	3	1.5

kernel=3x3
stride=2
padding=1

Output Image

0	0	0.25	0.5	0.75	1	1.25	1.5	0.75
0	0	0.5	1	1.5	2	2.5	3	1.5
1	2	2.5	3	3.5	4	2.75	1.5	0.75
2	4	4.5	5	5.5	6	3		
1	2	2.5	2.5	2.75	3	1.5		

Transpose Convolution: Example

Added Result

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

1.75	3.5	1.75
3.5	7	3.5
1.75	3.5	1.75

kernel=3x3
stride=2
padding=1

Output Image

0	0	0.25	0.5	0.75	1	1.25	1.5	0.75
0	0	0.5	1	1.5	2	2.5	3	1.5
1	2	2.5	3	3.5	4	4.5	5	2.5
2	4	4.5	5	5.5	6	6.5	7	3.5
1	2	2.5	2.5	2.75	3	3.25	3.5	1.75

Transpose Convolution: Example

Added Result

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

2	4	2
4	8	4
2	4	2

kernel=3x3
stride=2
padding=1

Output Image

0	0	0.25	0.5	0.75	1	1.25	1.5	0.75
0	0	0.5	1	1.5	2	2.5	3	1.5
1	2	2.5	3	3.5	4	4.5	5	2.5
2	4	4.5	5	5.5	6	6.5	7	3.5
3	6	4.25	2.5	2.75	3	3.25	3.5	1.75
4	8	4						
2	4	2						

Transpose Convolution: Example

Added Result

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

3.75	7.5	3.75
7.5	15	7.5
3.75	7.5	3.75

kernel=3x3
stride=2
padding=1

Output Image

0	0	0.25	0.5	0.75	1	1.25	1.5	0.75
0	0	0.5	1	1.5	2	2.5	3	1.5
1	2	2.5	3	3.5	4	4.5	5	2.5
2	4	4.5	5	5.5	6	6.5	7	3.5
3	6	6.5	7	7.5	8	8.5	9	4.5
4	8	8.5	9	9.5	10	10.5	11	5.5
5	10	10.5	11	11.5	12	12.5	13	6.5
6	12	12.5	13	13.5	14	14.5	15	7.5
3	6	6.25	6.5	6.75	7	7.25	7.5	3.75

Transpose Convolution: Example

Note: this result is equivalent to **Bilinear Interpolation**

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image



0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

kernel=3x3
stride=2
padding=1

0	0	0.25	0.5	0.75	1	1.25	1.5	1.75
0	0	0.5	1	1.5	2	2.5	3	3.5
4	2	2.5	3	3.5	4	4.5	5	5.5
2	4	4.5	5	5.5	6	6.5	7	7.5
3	6	6.5	7	7.5	8	8.5	9	9.5
4	8	8.5	9	9.5	10	10.5	11	11.5
5	10	10.5	11	11.5	12	12.5	13	13.5
6	12	12.5	13	13.5	14	14.5	15	15.5
3	6	6.25	6.5	6.75	7	7.25	7.5	7.75

Bilinear Interpolation is a special case of transpose convolution.

The corresponding transpose convolution kernels exists for any stride (code <https://gist.github.com/mjstevens777/9d6771c45f444843f9e3dce6a401b183>)

Transpose Convolution and Bilinear Interpolation

Thus...

the transpose convolution should be at least as good as bilinear interpolation.

In particular, transpose convolution kernel can be initialized to replicate bilinear interpolation, but one might learn a “better” upsampling kernel during training.

Transpose Convolution: other names

- ***Deconvolution***: not a very good name as it is commonly used for the inverse of convolution. Moreover, in image analysis, “*deconvolution*” also stands for a standard non-linear image reconstruction problem.
- ***Backward convolution***: If we think about convolution of an input image as a matrix multiplication operation, then transposed convolution could be related to the backward pass when the loss gradient is backpropagated through the standard convolutional layer.
- ***Fractionally-strided convolution***: transposed convolution with stride s is equivalent to a standard convolution with stride $1/s$, as follows: insert $(s-1)$ zeros between pixels, then apply regular conv using the same kernel (see **example on the next slide**).

Fractionally-strided Convolution

Fractional Stride

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image

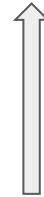
0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Standard Convolution

kernel=3x3

stride= $\frac{1}{2}$

(inserting one zero between pixels, then apply conv with stride=1)
padding=1



Transposed Convolution

kernel=3x3

stride=2

padding=1

Zero-interleaved Image (also zero-padded)

0	0	0	0	0	0	0	0	0
0	0	0	1	0	2	0	3	0
0	0	0	0	0	0	0	0	0
0	4	0	5	0	6	0	7	0
0	0	0	0	0	0	0	0	0
0	8	0	9	0	10	0	11	0
0	0	0	0	0	0	0	0	0
0	12	0	13	0	14	0	15	0
0	0	0	0	0	0	0	0	0

Now, apply standard convolution...

Fractionally-strided Convolution

Zero-interleaved Image

(also zero-padded)

0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	2	0	3	0	0
0	0	0	0	0	0	0	0	0	0
0	4	0	5	0	6	0	7	0	0
0	0	0	0	0	0	0	0	0	0
0	8	0	9	0	10	0	11	0	0
0	0	0	0	0	0	0	0	0	0
0	12	0	13	0	14	0	15	0	0
0	0	0	0	0	0	0	0	0	0

standard convolution

with
kernel

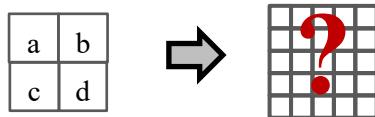
0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

0	0.5	1	1.5	2	2.5	3
2	2.5	3	3.5	4	4.5	5
4	4.5	5	5.5	6	6.5	7
6	6.5	7	7.5	8	8.5	9
8	8.5	9	9.5	10	10.5	11
10	10.5	11	11.5	12	12.5	13
12	12.5	13	13.5	14	14.5	15

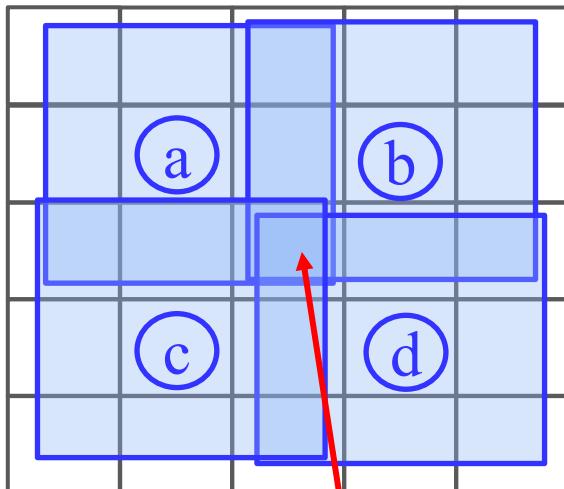
Output

Transposed vs Fractionally-strided Convolution

Upsampling Example:



transpose convolution (slide 26)



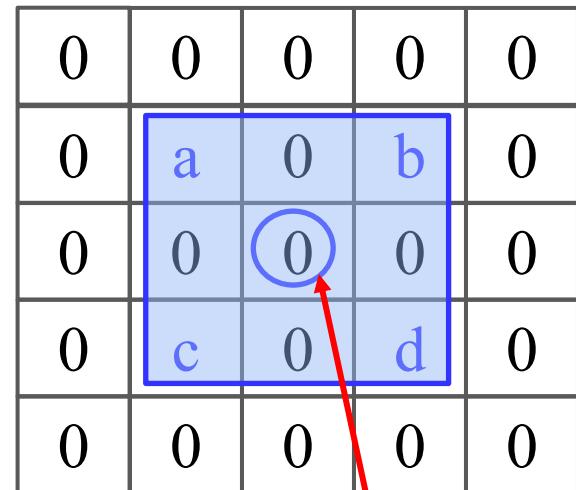
kernel k

$$\begin{matrix} k_{-1,-1} & k_{0,-1} & k_{1,-1} \\ k_{-1,0} & k_{0,0} & k_{1,0} \\ k_{-1,1} & k_{0,1} & k_{1,1} \end{matrix}$$

output of transpose convolution using k
with stride 2 for the **pixel in the center**

$$ak_{1,1} + bk_{-1,1} + ck_{1,-1} + dk_{-1,-1}$$

fractionally-strided convolution

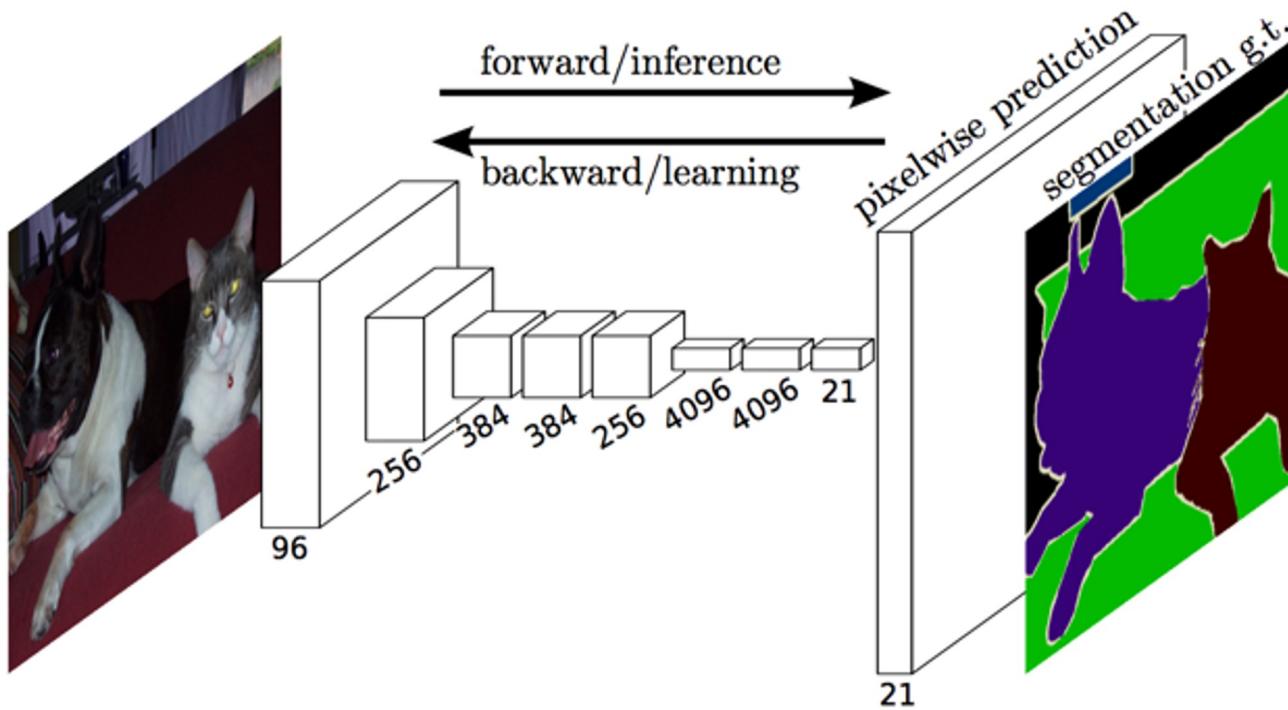


output of standard convolution using k
with stride 1/2 for the **pixel in the center**

$$ak_{-1,-1} + bk_{1,-1} + ck_{-1,1} + dk_{1,1}$$

For non-symmetric kernels one must use a “transposed” version of the kernel (flipped both horizontally & vertically) to get equivalence between the transposed convolution (as on slide 26) and the fractionally-strided convolution.

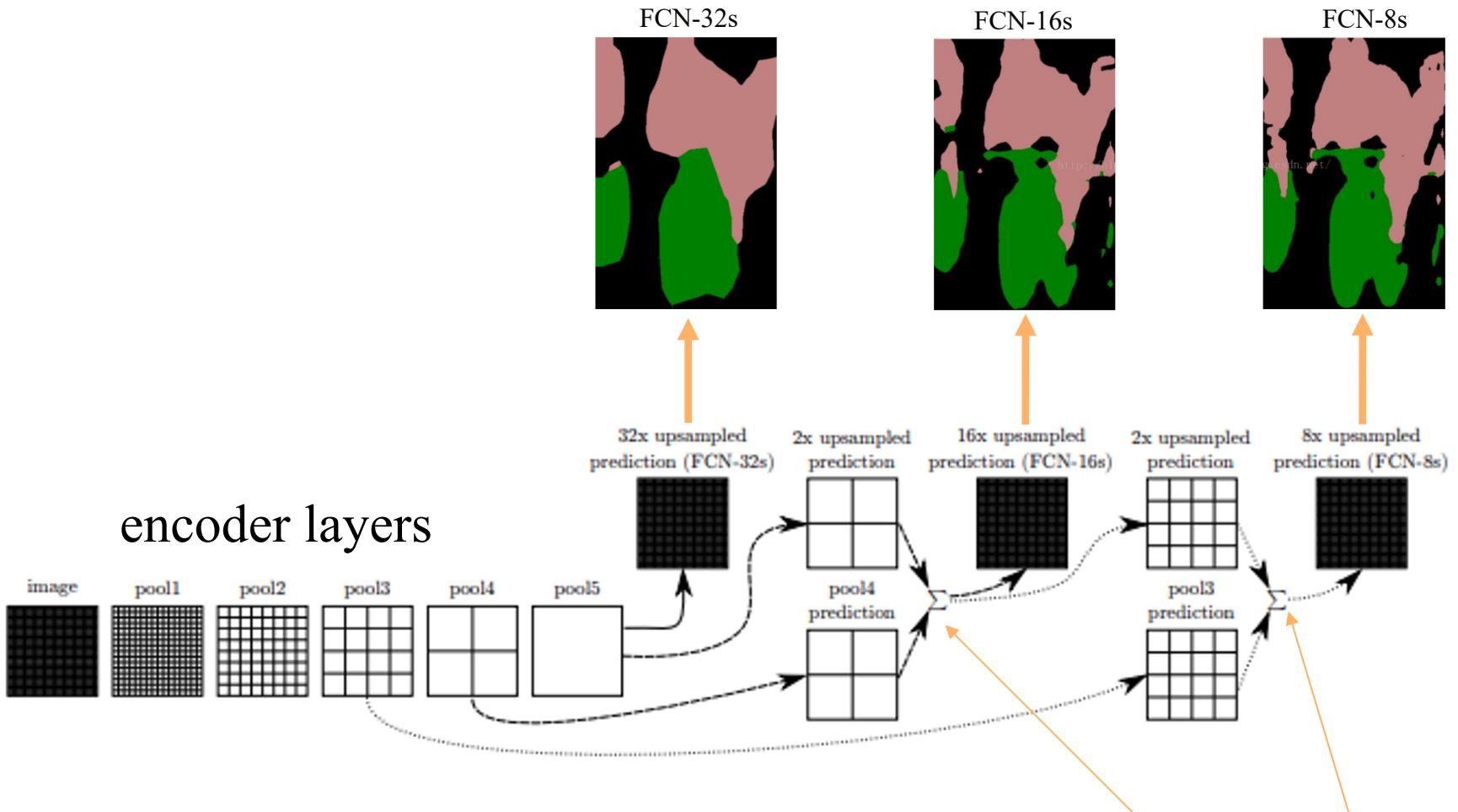
Fully Convolutional Networks (FCNs)



Upsample segmentation using “**deconvoluton**” *transposed convolution*

Fully Convolutional Networks for Semantic Segmentation
Long, Shelhamer, Darrell - CVPR 2015

Upsampling using skip connections

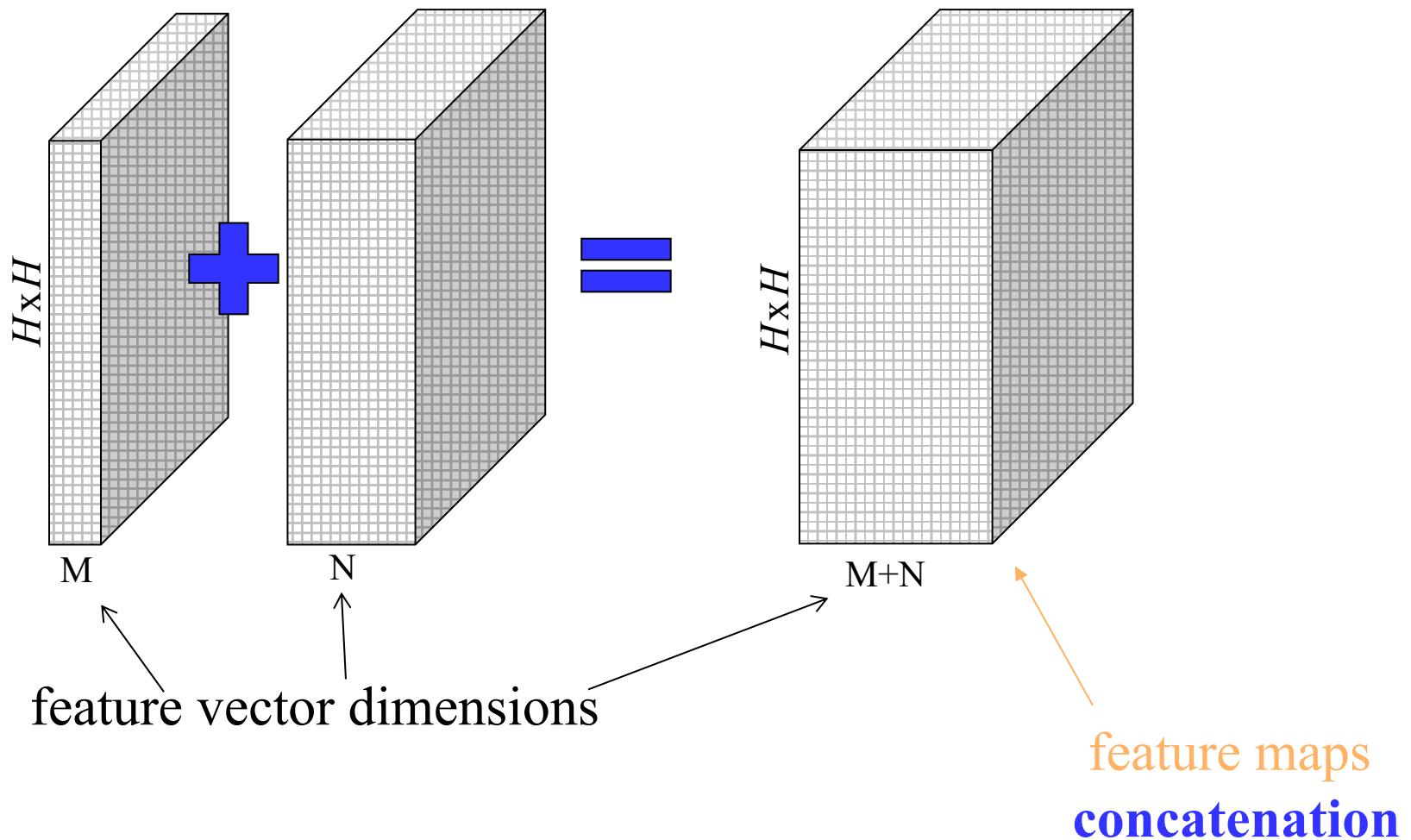


Fully Convolutional Networks for Semantic Segmentation
Long, Shelhamer, Darrell - CVPR 2015

feature maps
concatenation

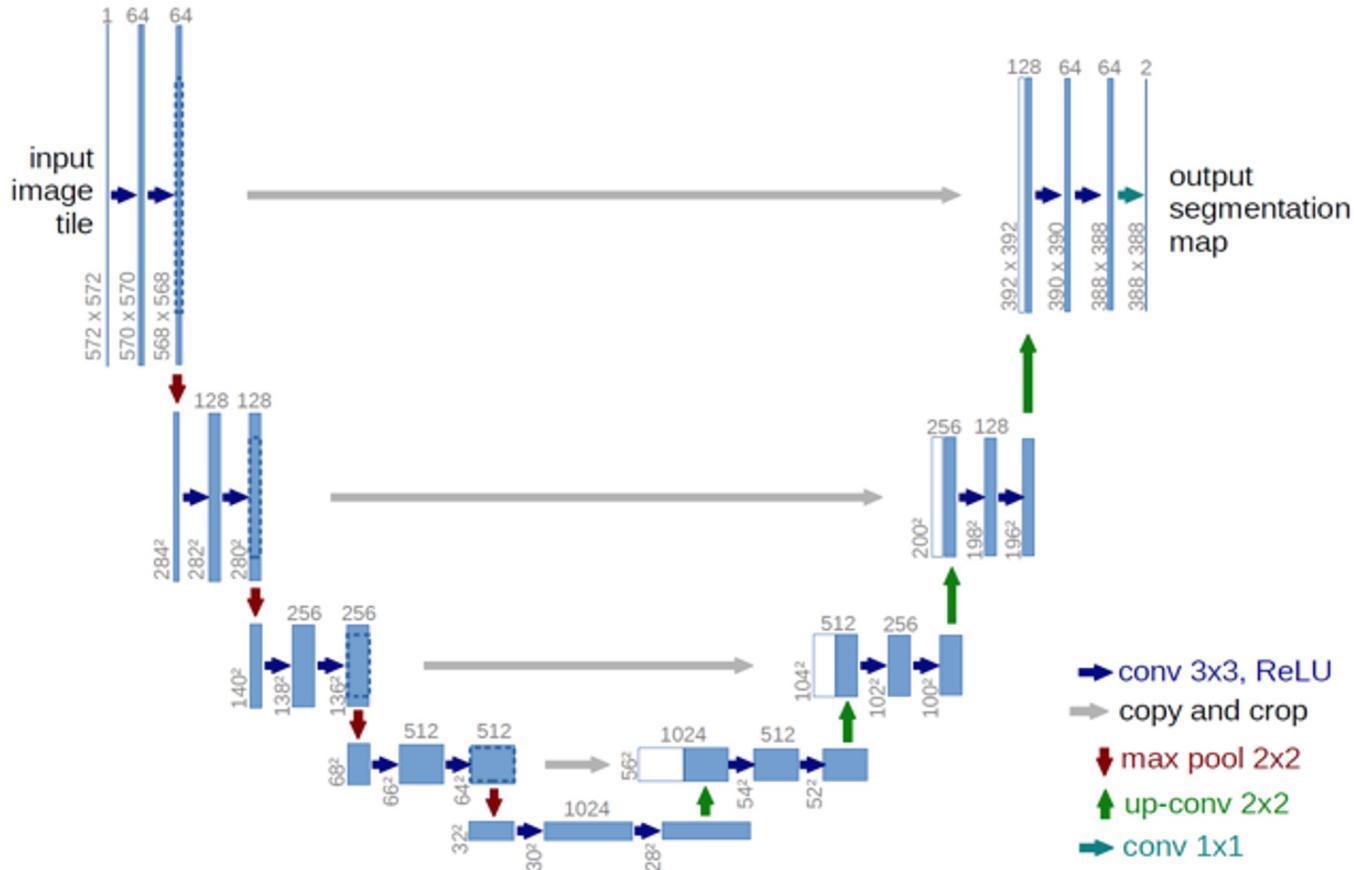
Skip connections: concatenation

feature map
“skipped”
from encoder feature map
“upsampled”
insider decoder



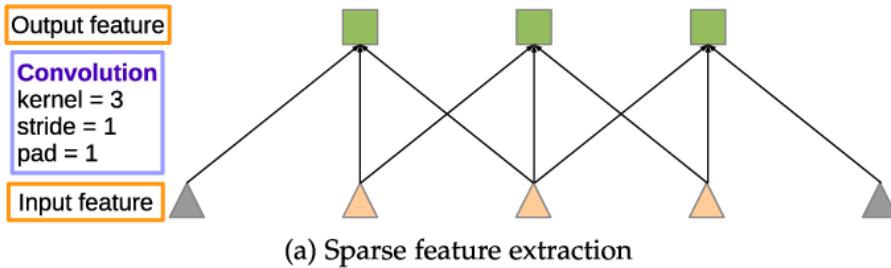
U-net: expanding decoder with symmetry

and many skip connections

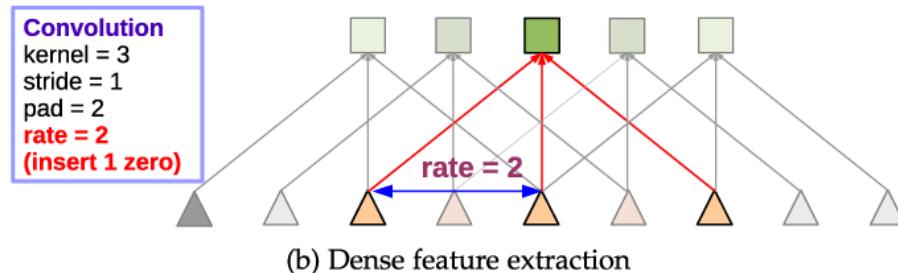


DeepLab

- encoder uses *atrous convolutions* (a.k.a. *dilation*)
increasing *receptive field* without increase in kernel size
(or significant decrease in output resolution)



standard 3x3 convolution



atrous 3x3 convolution
i.e. convolution with
holes or gaps (Fr. *trous*)

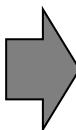
Key insight: encoder can still use any standard kernels pre-trained on *image-net* classification (e.g. from *ResNet*)
For example, pre-trained 3x3 kernels can be “dilated” into 5x5 kernels (as above) by adding “holes”

DeepLab

- encoder uses *atrous convolutions* (a.k.a. *dilation*)
increasing *receptive field* without significant loss of resolution
(unlike stride and pooling)
- decoder uses *bilinear interpolation* (see topic 4)
for upsampling
- other ideas

(Training) Loss: Cross-Entropy

image sample i



network prediction



$$\bar{\sigma}^p = (\bar{\sigma}_1, \bar{\sigma}_2, \dots, \bar{\sigma}_K)$$

prediction at each pixel p

(GT mask)

pixel-precise target



$$\mathbf{y}^p \in [0, 1, 2, 3, \dots] \quad - \text{ class label at each pixel } p$$

$$\bar{\mathbf{y}}^p = (0, 0, 1, 0, \dots, 0) \quad - \text{ one-hot distribution at } p$$

Loss over
image i :

$$\overbrace{\sum_{p \in I_i} \sum_k -\bar{\mathbf{y}}_k^p \ln \bar{\sigma}_k^p}^{\text{cross entropy at } p}$$

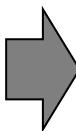
sum of
negative log-likelihoods (NLL)

$$= - \sum_{p \in I_i} \ln \bar{\sigma}_{\mathbf{y}^p}^p$$

Total loss should also sum over all images i

(Validation) Quality Metrics

image sample i



network prediction



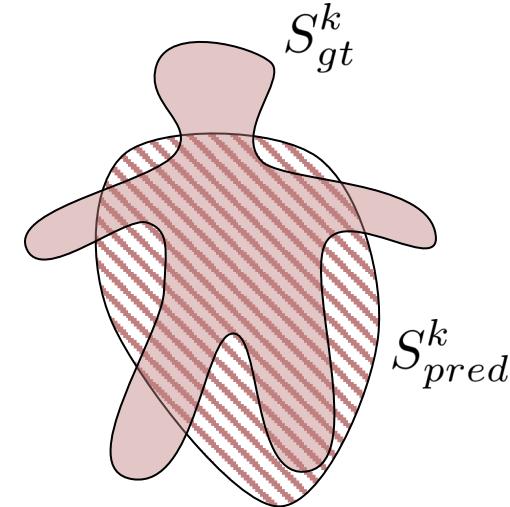
(GT mask)

pixel-precise target



S_{gt}^k

S_{pred}^k



- *Mean intersection over union* $mIoU = \frac{1}{K} \sum_k \frac{|S_{gt}^k \cap S_{pred}^k|}{|S_{gt}^k \cup S_{pred}^k|} \in [0, 1]$
(focus on segments/classes, object sizes are irrelevant)
- There are also accuracy measures focused on pixels
(what percentage of pixels is correctly classified)



Weakly-supervised Semantic Segmentation

Weakly Supervised Semantic Segmentation

bounding



scribbles



clicks



polygons

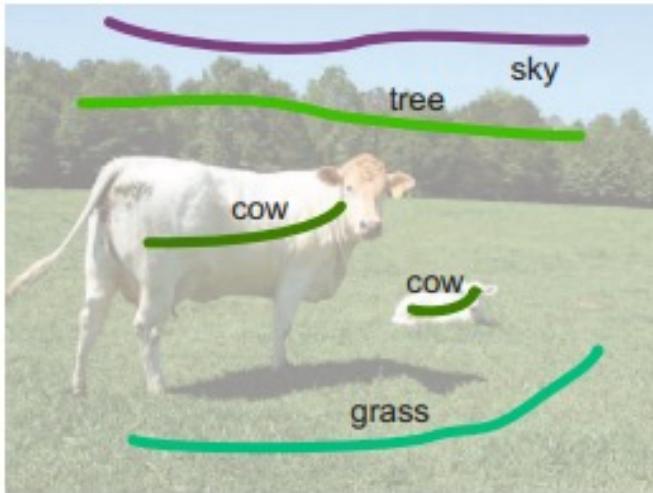


**image-level
labels**

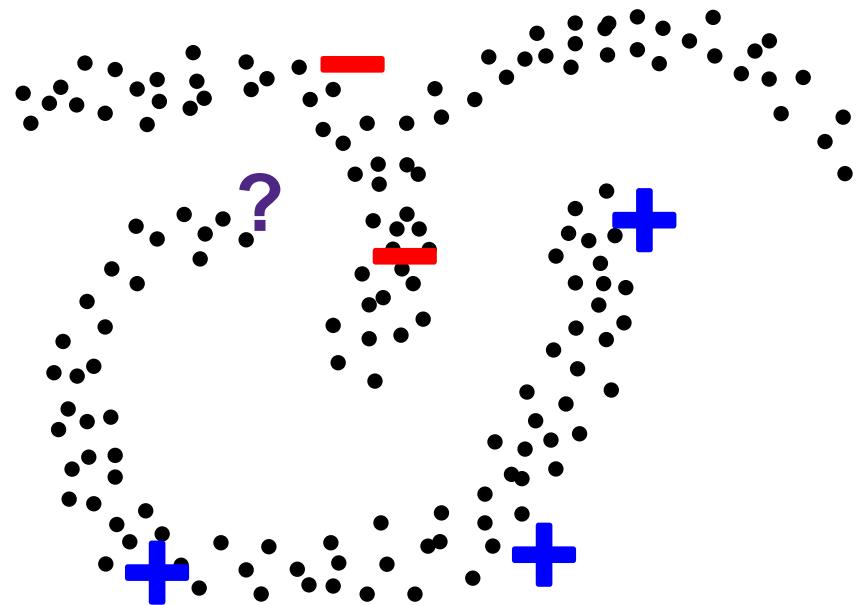


Key Idea

Weakly-supervised
segmentation

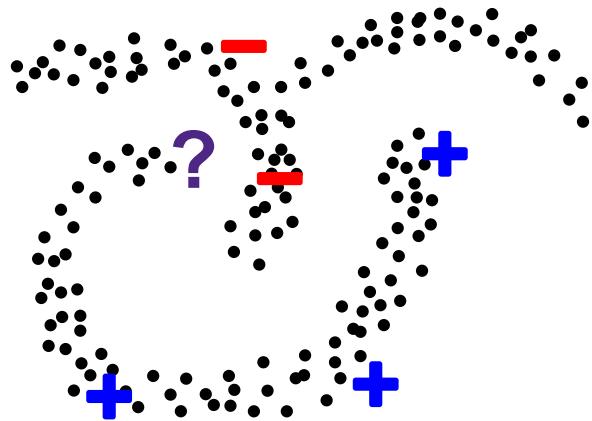


Semi-supervised learning



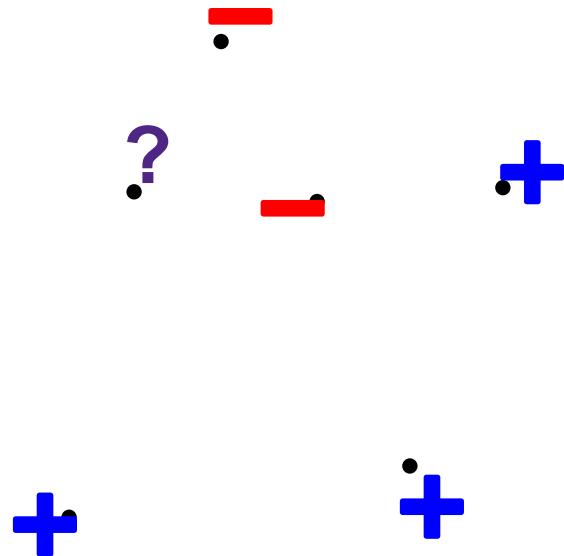
Semi-supervised learning

Definition Given M labeled data $(x_i, y_i) \in (\mathcal{X}, \mathcal{Y}), i = 1, \dots, M$ and U unlabeled data $x_i, i = M + 1, \dots, M + U$, learn $f(x) : \mathcal{X} \rightarrow \mathcal{Y}$.

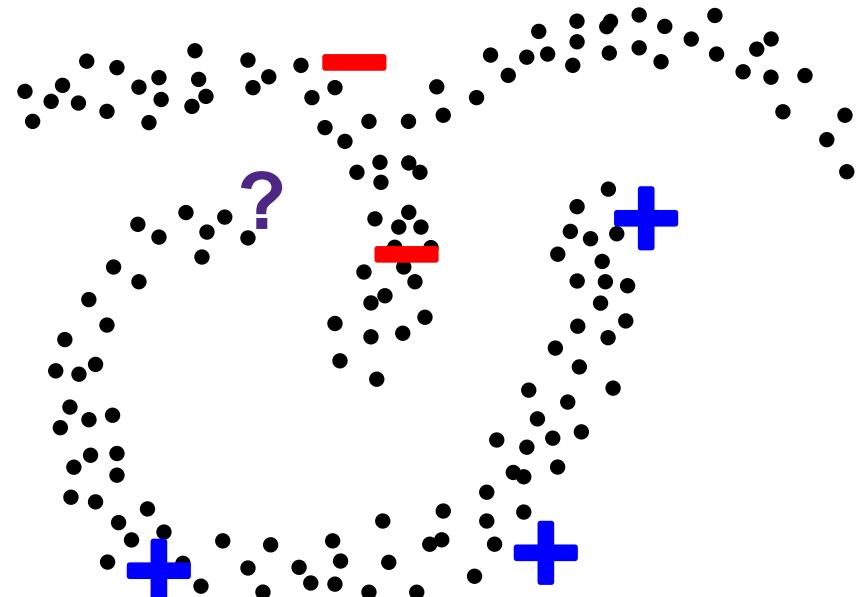


[Zhu & Goldberg, “Introduction to semi-supervised learning”, 2009]
[Chapelle, Scholkopf & Zien, “Semi-supervised learning”, 2009]

Does unlabeled data matter?



w/o unlabeled data



w/ unlabeled data

Semi-supervised Learning Methods

Self-training

Graph-based Semi-supervised learning

Entropy minimization

Many others...

[Zhu & Goldberg, “Introduction to semi-supervised learning”, 2009]
[Chapelle, Scholkopf & Zien, “Semi-supervised learning”, 2009]

Graph-Based Semi-supervised Learning

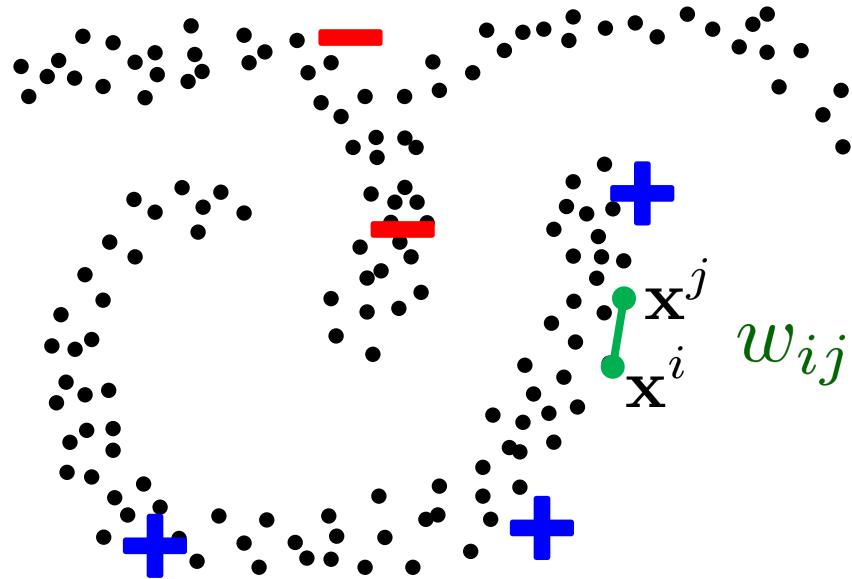
Loss function ?

- labelled points should have **consistency with the target**
e.g.

$$\sum_{i=1}^M \delta(f(\mathbf{x}^i) \neq \mathbf{y}^i)$$

- unlabeled points should be labeled so that there is some agreement between neighbors
i.e. **pairwise regularization**:

$$\sum_{ij \in \mathcal{N}} w_{ij} ||f(\mathbf{x}^i) - f(\mathbf{x}^j)||^2$$

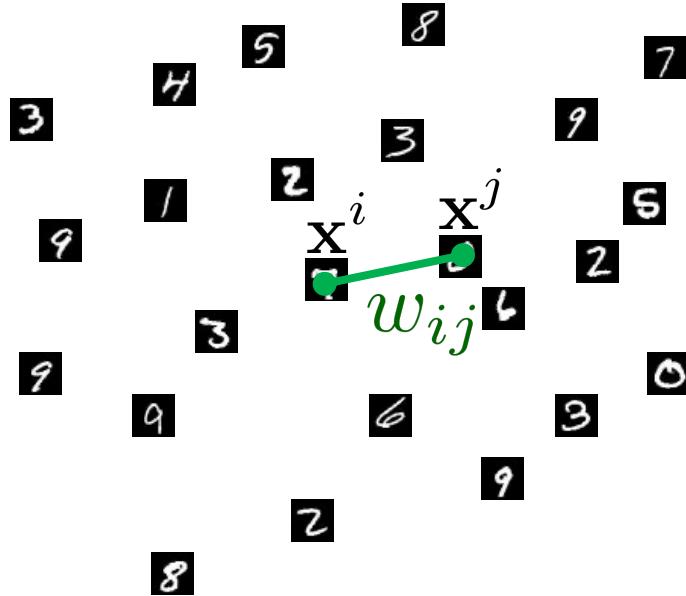


w_{ij} - pre-computed penalty,
e.g. based on distance
between feature vectors
 \mathbf{x}^i and \mathbf{x}^j

Deep Semi-supervised Learning

Classification

(Weston et al. 2012)



e.g. for **classification CNN** output

$$f(\mathbf{x}^i) = \bar{\sigma}^i \equiv (\bar{\sigma}_1^i, \dots, \bar{\sigma}_K^i)$$

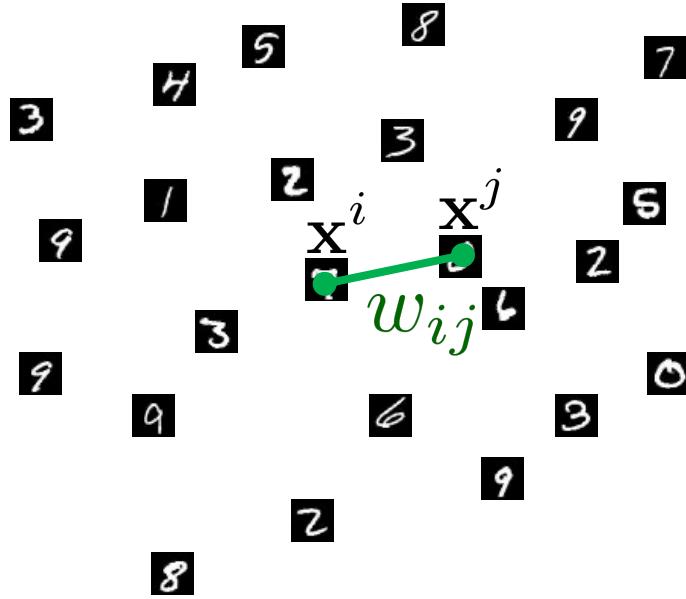
class probabilities at point i

$$\sum_{ij \in \mathcal{N}} w_{ij} \quad ||\bar{\sigma}^i - \bar{\sigma}^j||^2$$

Deep Semi-supervised Learning

Classification

(Weston et al. 2012)



e.g. for **classification CNN** output

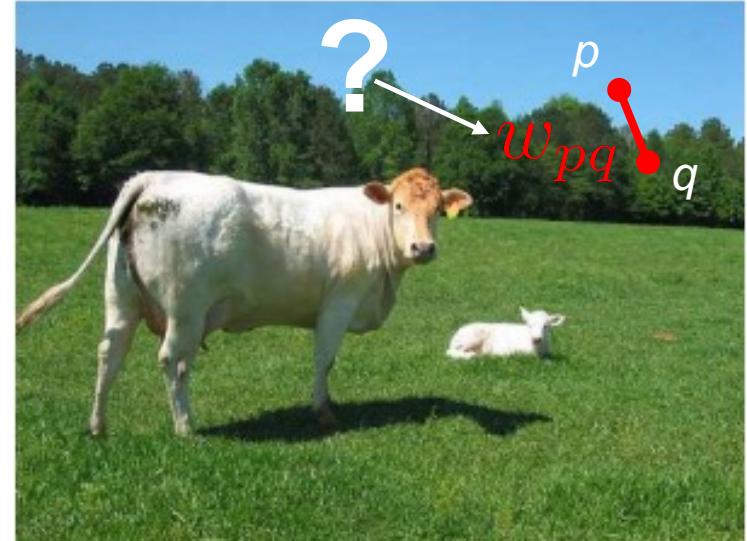
$$f(\mathbf{x}^i) = \bar{\sigma}^i \equiv (\bar{\sigma}_1^i, \dots, \bar{\sigma}_K^i)$$

class probabilities at point i

$$\sum_{ij \in \mathcal{N}} w_{ij} \quad \|\bar{\sigma}^i - \bar{\sigma}^j\|^2$$

Segmentation

(Tang et al. CVPR18, ECCV18)



e.g. for **segmentation CNN** output

$$\bar{\sigma}^p \equiv (\bar{\sigma}_1^p, \dots, \bar{\sigma}_K^p)$$

class probabilities at pixel p

$$\sum_{pq \in \mathcal{N}} w_{pq} \quad \|\bar{\sigma}^p - \bar{\sigma}^q\|^2$$

Regularized Loss Functions

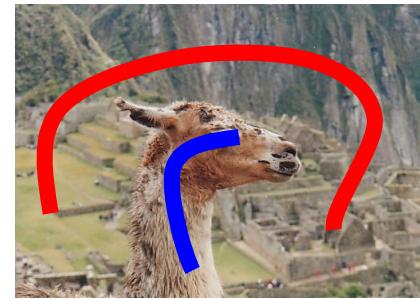
We can use regularization ideas from
**unsupervised and interactive
segmentation**

to exploit low-level segmentation cues

(contrast alignment, boundary regularity, regional color consistency, etc.)
for unlabeled parts of an image

low-level segmentation

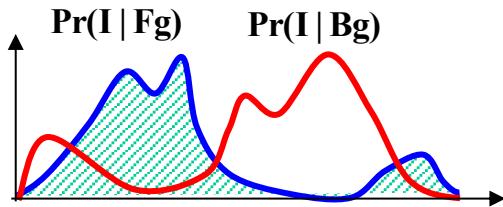
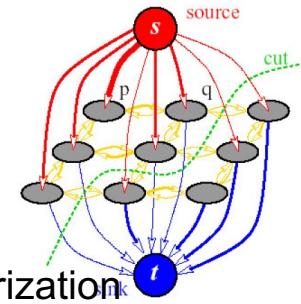
Markov Random Field for Segmentation



Without Regularization



With Regularization



$$E(S, \theta_0, \theta_1) = \sum_{k=0,1} \sum_{p \in S^k} -\ln P(I_p | \theta_k) + \lambda \cdot \sum_{pq \in \mathcal{N}} w_{pq} \cdot [s_p \neq s_q]$$

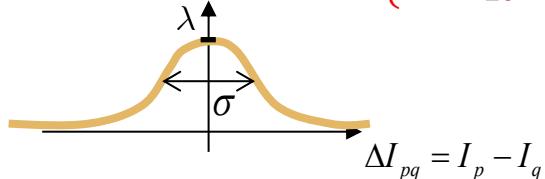
MRF regularization

[Boykov, Jolly, ICCV 2001]

Regularization energies

$$w_{pq} = \lambda \exp \left\{ -\frac{\|I_p - I_q\|^2}{2\sigma^2} \right\}$$

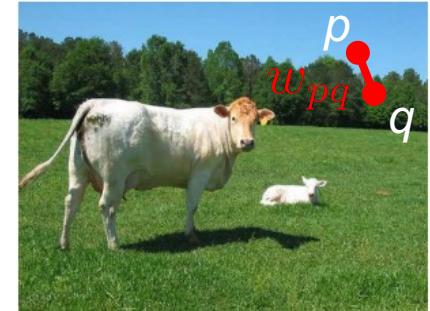
- contrast weights w_{pq} from topic 9



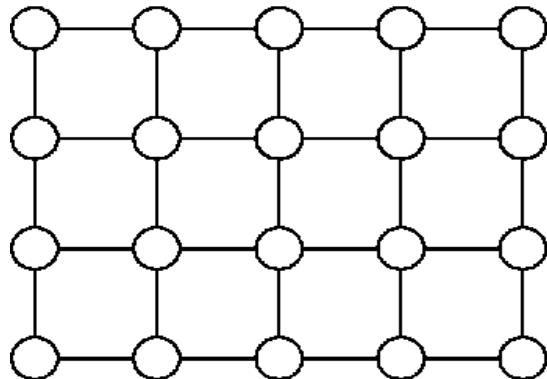
coherence between
discrete labels
at pixels p and q

$$\sum_{pq \in \mathcal{N}} w_{pq} [S^p \neq S^q]$$

Iverson brackets

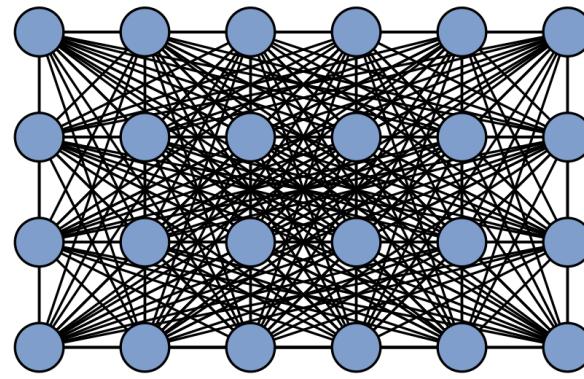


Examples of neighborhood systems \mathcal{N} on pixel grid



sparsely connected

[Geman&Geman'81, BVZ PAMI'01, B&J ICCV'01]



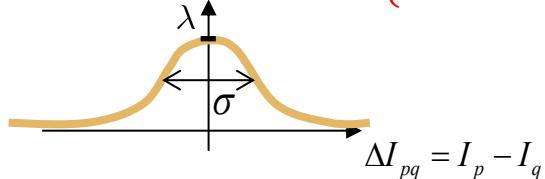
densely connected

[Dense CRF, Krähenbühl & Koltun, NIPS 2011]

Regularization Loss

$$w_{pq} = \lambda \exp \left\{ -\frac{\|I_p - I_q\|^2}{2\sigma^2} \right\}$$

- contrast weights w_{pq} from topic 9



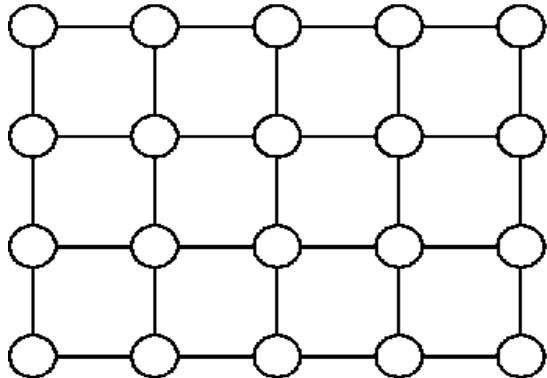
coherence between
probabilistic predictions
at pixels p and q

$$\sum_{pq \in \mathcal{N}} w_{pq} \ ||\bar{\sigma}^p - \bar{\sigma}^q||^2$$

relaxation of Iverson
brackets for probabilistic
predictions

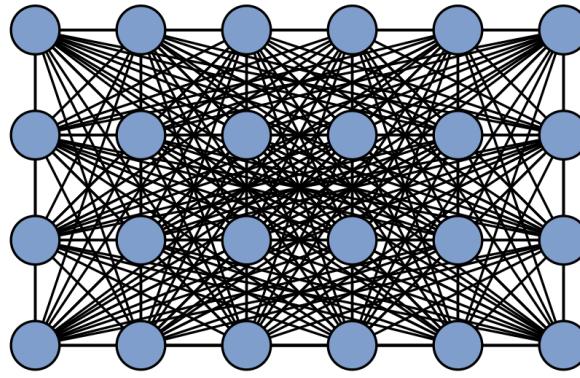


Examples of neighborhood systems \mathcal{N} on pixel grid



sparsely connected

[Geman&Geman'81, BVZ PAMI'01, B&J ICCV'01]



densely connected

[Dense CRF, Krähenbühl & Koltun, NIPS 2011]

weakly-supervised CNN segmentation:

Partial Cross Entropy Loss

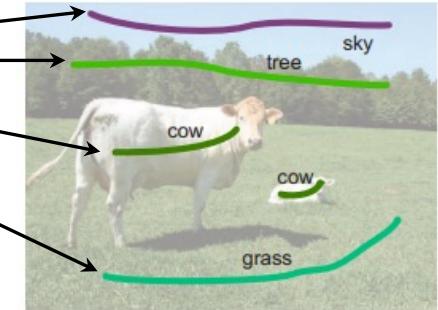


**cross entropy
over seeds only**

$$-\sum_{p \in \text{seeds}} \ln \bar{\sigma}_{\mathbf{y}^p}^p$$

$$\bar{\sigma}^p \equiv (\bar{\sigma}_1^p, \dots, \bar{\sigma}_K^p)$$

predicted “probabilities” for p
to be in each class, e.g. $(0,0,\dots,1,\dots)$ in **one-hot** case



NOTE: if prediction is one-hot
then cross entropy at seed p
is equivalent to $0/\infty$ hard constraint
(as in interactive graph cut, Topic 9)

$$\sum_{p \in \text{seeds}} \delta(\bar{\sigma}^p \neq \bar{\mathbf{y}}^p)$$

hard constraint
on seed p

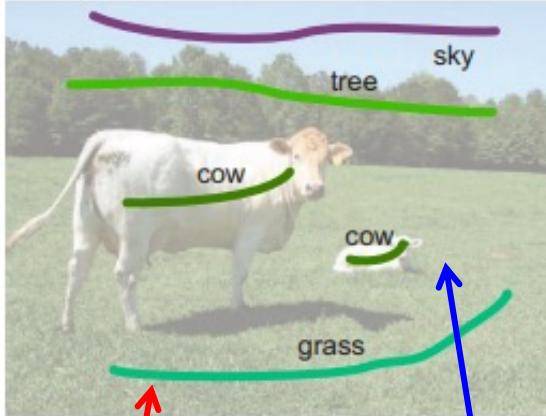
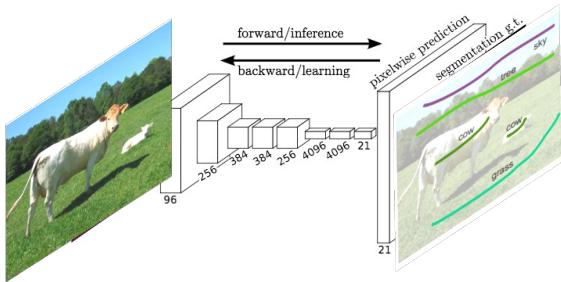
Implications:

- Cross entropy is a relaxation of hard constraints for probabilistic predictions.
- Cross entropy is a bad idea for pixels where targets \mathbf{y}^p could be wrong.

Remember “fake” ground truths - network tries hard to learn all their mistakes.

weakly-supervised CNN segmentation:

Total Regularized Loss



$$L(\bar{\sigma}) = - \sum_{p \in seeds} \ln \bar{\sigma}_{\mathbf{y}^p}^p$$

Partial Cross Entropy (PCE)

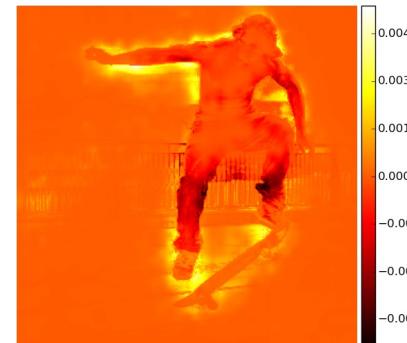
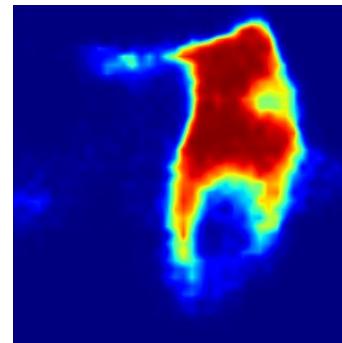
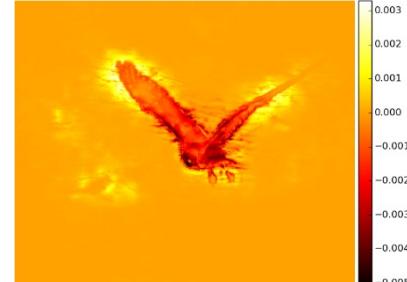
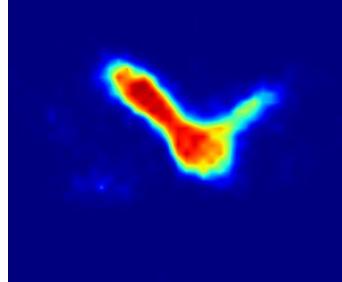
$$+ \sum_{\substack{pq \in \mathcal{N} \\ n-links}} w_{pq} \ ||\bar{\sigma}^p - \bar{\sigma}^q||^2$$

Regularization Loss

scribbles / seeds

unlabeled pixels

Regularization Loss Gradients



input

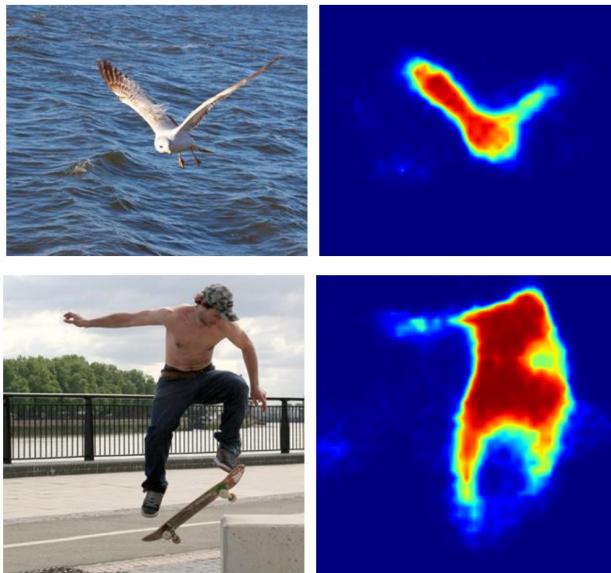
network prediction for
class k during training

$$\bar{\sigma}_k^p$$

regularization loss
gradient $\frac{\partial R(\sigma)}{\partial \sigma_k}$

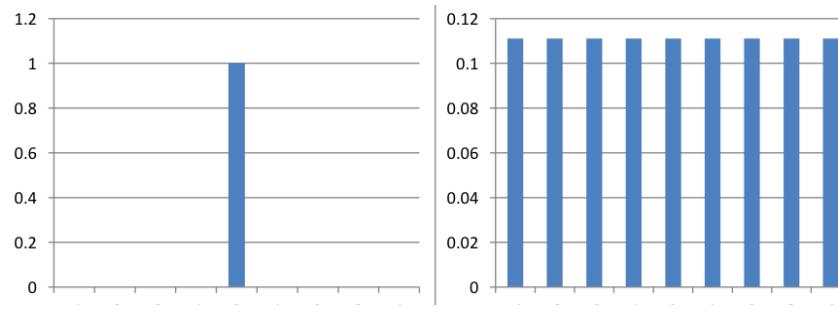
$$R(\sigma) = \sum_{pq \in \mathcal{N}} w_{pq} \cdot \|\bar{\sigma}^p - \bar{\sigma}^q\|^2$$

CNN Segmentation may be blurred



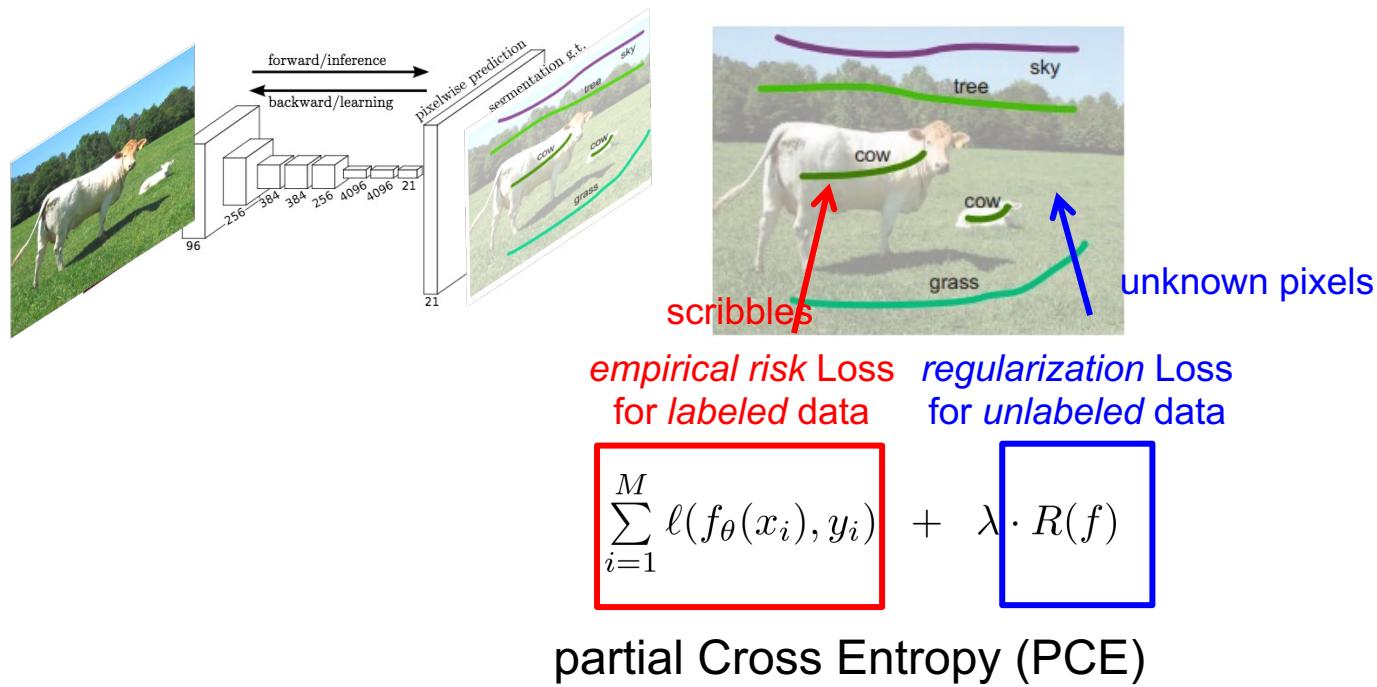
Pointwise Entropy Regularization

$$\sum_i H(f(x_i))$$



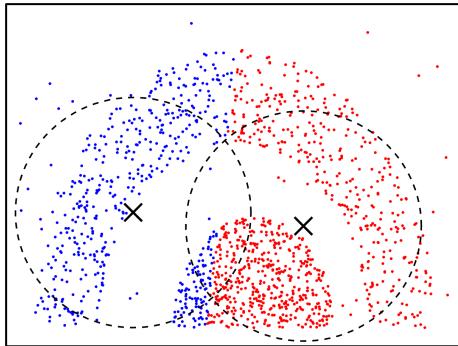
$$H(P) = \sum_{k=0}^K -P_k \cdot \log P_k$$

Regularized loss for weakly-supervised CNN segmentation

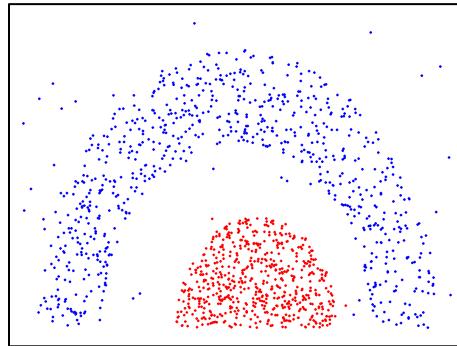


Clustering and Segmentation are Largely Synonym

Linear Clustering



Nonlinear Clustering



Normalized Cut
Segmentation

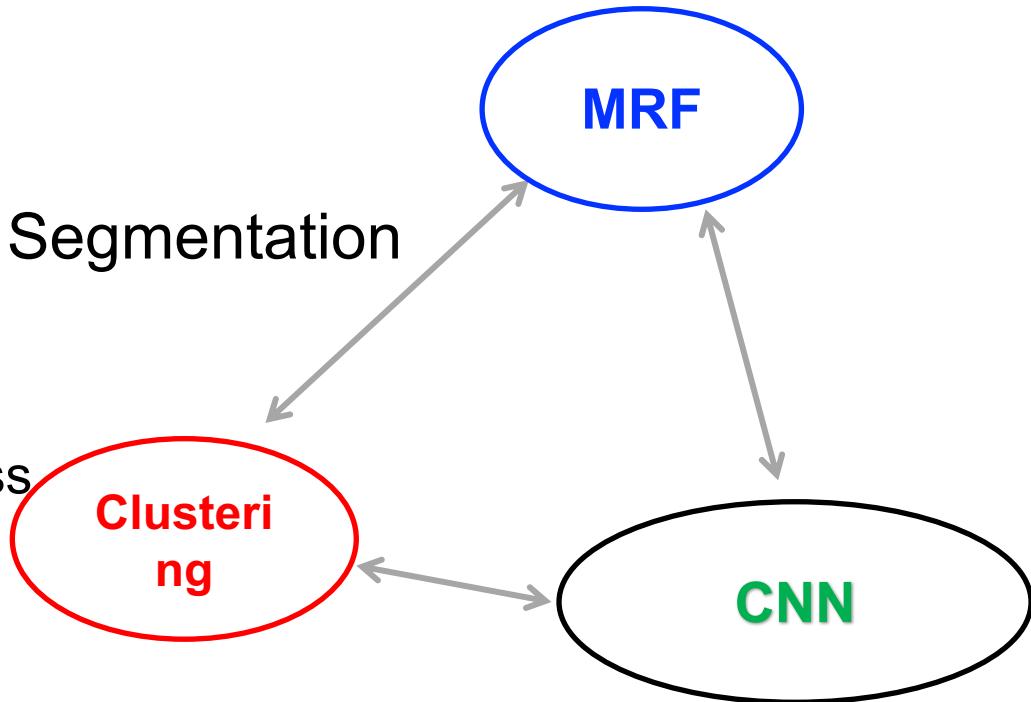
Kernel K-means

$$\begin{aligned} & \sum_{p \in S} \|\phi(I_p) - \mu_S\|^2 + \sum_{p \in \bar{S}} \|\phi(I_p) - \mu_{\bar{S}}\|^2 \\ & \stackrel{c}{=} -\frac{\sum_{p,q \in S} k(I_p, I_q)}{|S|} - \frac{\sum_{p,q \in \bar{S}} k(I_p, I_q)}{|\bar{S}|} \end{aligned}$$

Regularized Losses

Regularized Loss for **CNN** Segmentation

- Pointwise entropy loss
- Pairwise **MRF** loss
- High-order **Clustering** loss



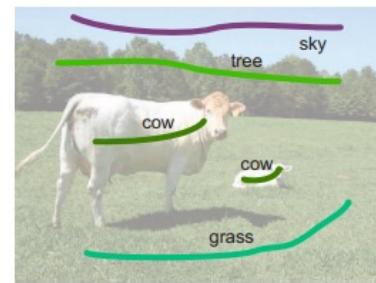
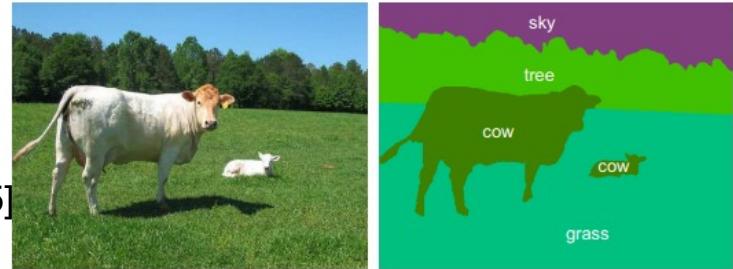
Experiments

PASCAL VOC 2012 Segmentation Dataset

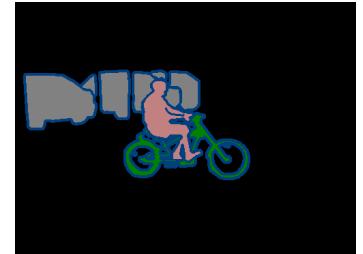
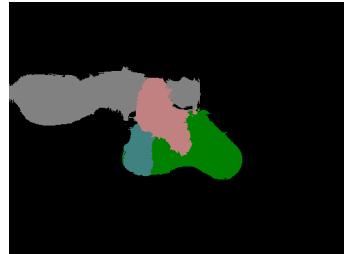
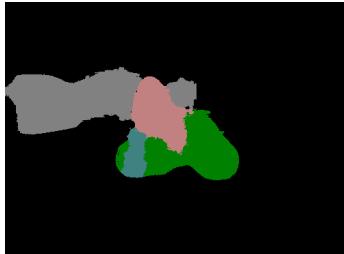
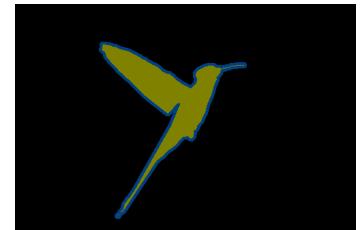
- 10K training images (full masks)
- 1.5K validation images
- 1.5K test images

ScribbleSup Dataset [Dai *et al.* ICCV 2015]

- scribbles for each object
- ~3% of pixels labelled



Training with combination of losses



Test image

pCE loss

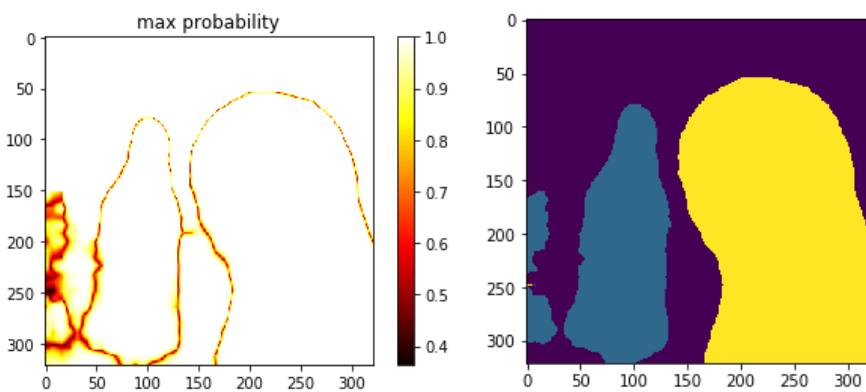
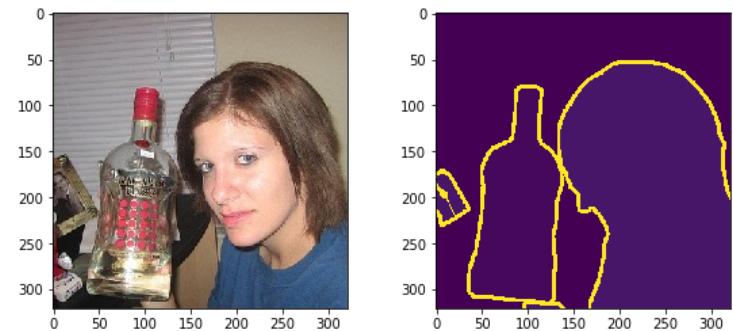
+ clustering loss

+ clustering loss
+ MRF loss

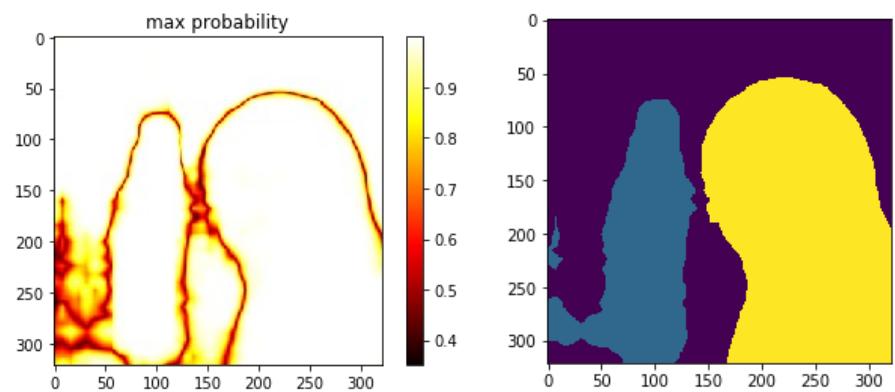
Ground truth

better color clustering better edge alignment

Peakedness of distribution



w/ entropy regularization



w/o entropy regularization

Compare weak and full supervision

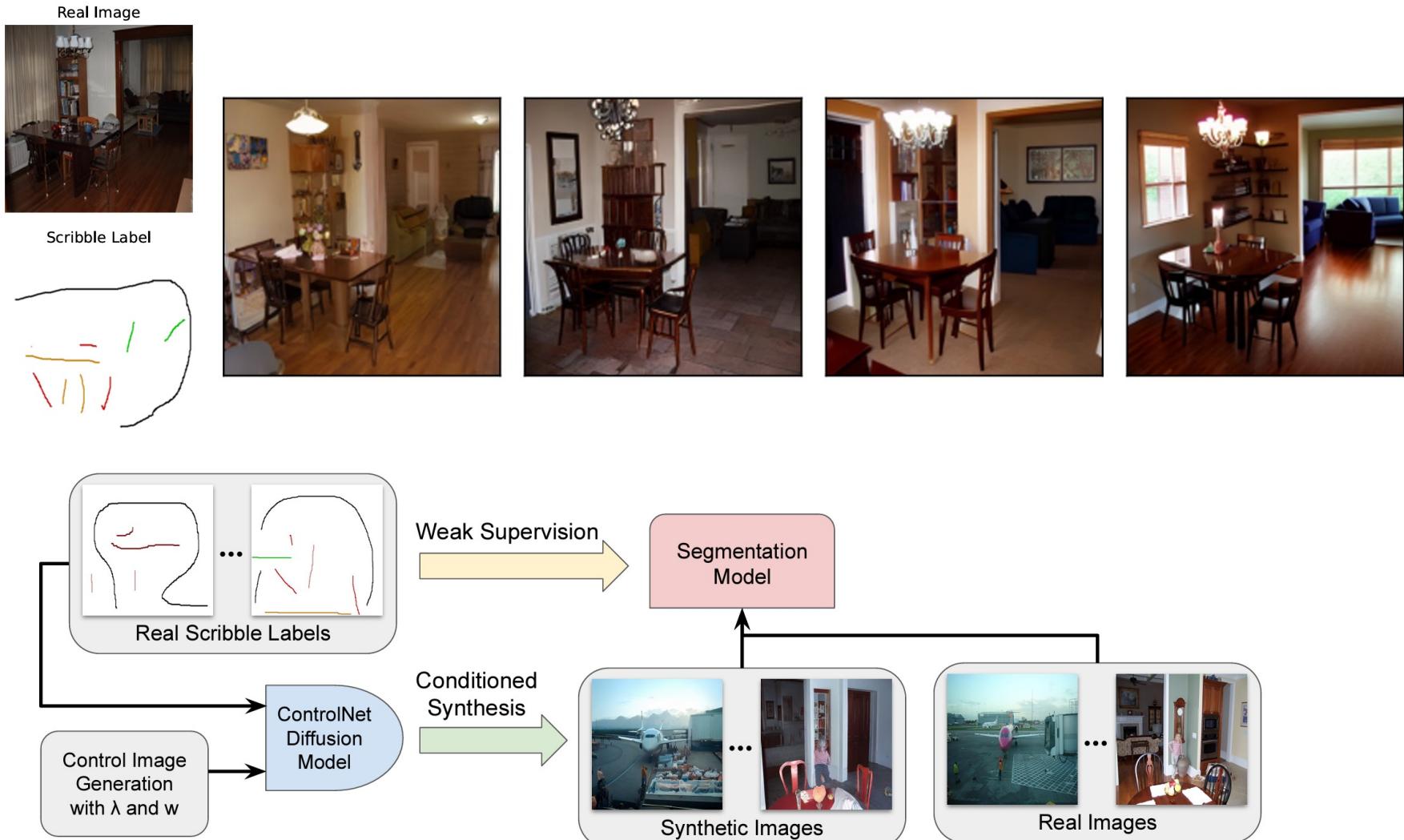
Almost as good as
full supervision!

network	Full supervision	Weak supervision			
		PCE	PCE+CRF [1]	PCE+ENTROPY	PCE+CRF+ENTROPY
Deeplab2-largeFOV	63.0	55.8	62.2	59.9	63.0
Deeplab2-Msc-largeFOV	64.1	56.0	63.1	n/a	63.5
Deeplab2-VGG16	68.8	60.4	64.4	63.3	65.5
Deeplab2-Resnet101	75.6	69.5	72.9	73.1	74.4
Deeplab3 ⁺ -Resnet101	78.6	71.9	74.6	74.0	75.6

PCE: partial cross entropy. CRF: pairwise conditional random field

[1] Tang et al., “On Regularized Losses for Weakly-supervised CNN Segmentation”, in *ECCV 2018*.

Generative Data Augmentation



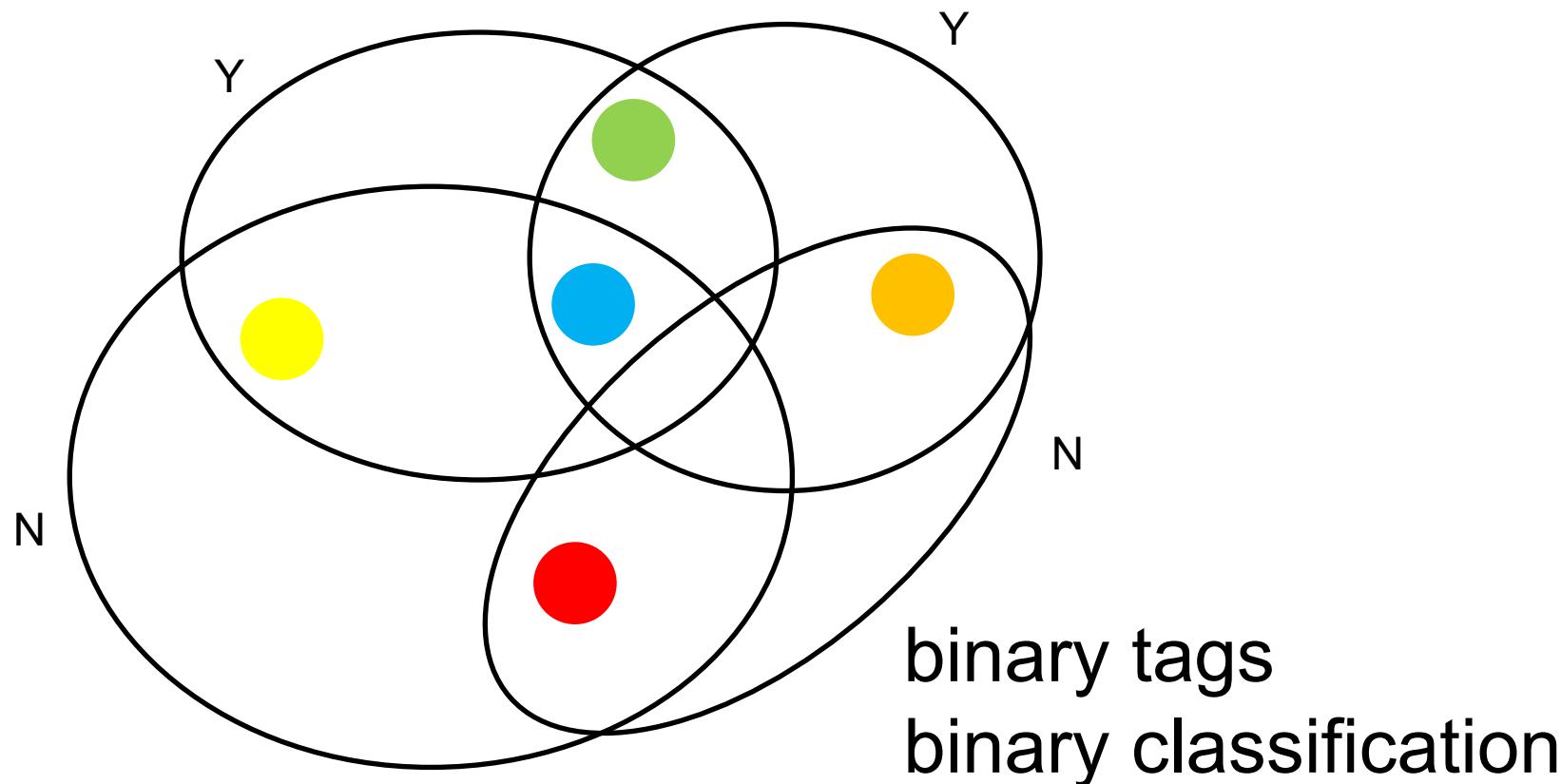
Schnell, et al. "Generative Data Augmentation Improves Scribble-supervised Semantic Segmentation." *arXiv preprint arXiv:2311.17121* (2023).

What if image-level labels only ?

First, consider a simple related example:

find working molecule (drug discovery)

instead of individual examples,
training labels are available
only for sets (bags) of examples



Multiple Instance Learning (MIL)

What if image-level labels only ?

For simplicity, assume pixel colors are discriminative enough features.

To segment, we have to learn **what color is sky, grass, and sand**

From these three images, we can segment pixels by matching **green to grass**, **blue to sky**, and **beige to sand**



{ sky, grass, sand }

{ sky, sand }

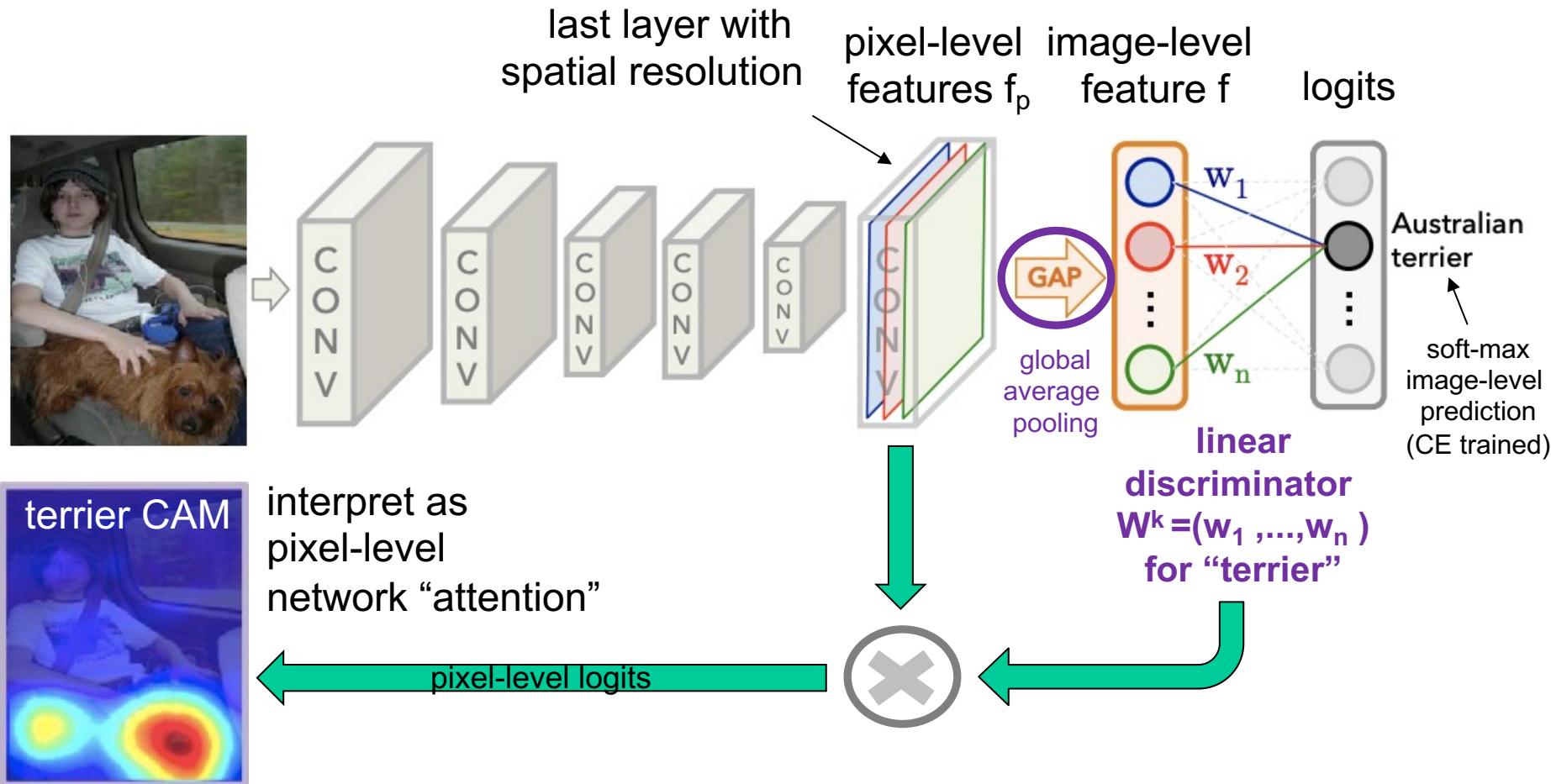
{ grass, sand }

image-level tags

multi-class tags
multi-class classification

In general, segmentation network must learn BOTH
(deep) **discriminative pixel-level features** AND their **match with class tags**

Class-activation Map (CAM)



CVPR 2016: “Learning Deep Features for Discriminative Localization”
B.Zhou, A.Khosla, A. Lapedriza, A.Oliva, A.Torralba

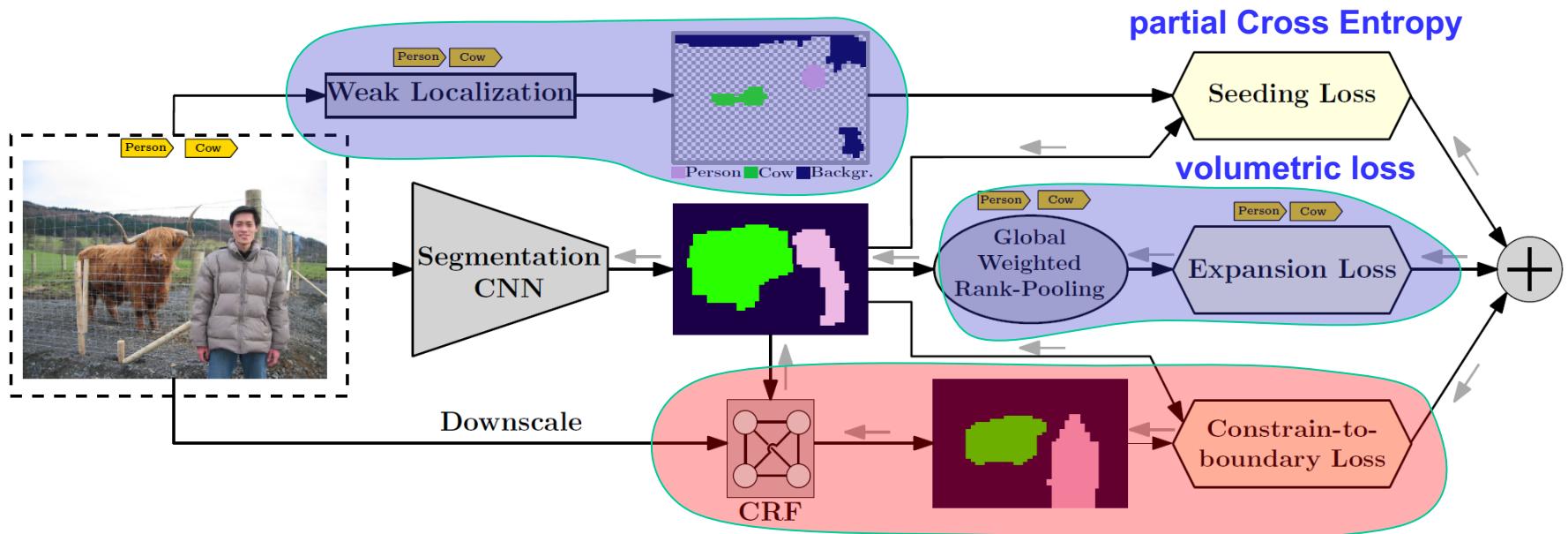
NOTE: motivates ideas for **object localization**, as well as
image-level supervision for semantic segmentation

What if image-level labels only ?

Some ideas: [Kolesnikov & Lampert ECCV 2016]

seeds from “network attention”

see CAM at the end of Topic 10



Can be simplified using
regularization loss
in the previous slides