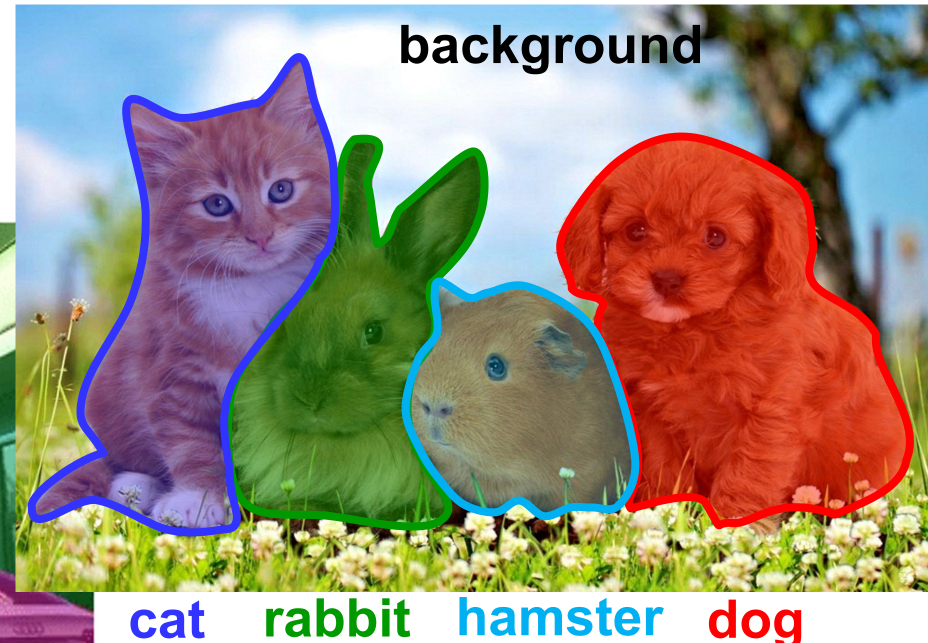
Semantic Segmentation



Slides credit: Yuri Boykov

Semantic Segmentation (outline)

- Fully-supervised CNN segmentation
  - from image labeling to **pixel labeling**
  - typical architectures
    fully convolutional networks, encoder/decoder, upsampling, skip connections, dilated (atrous) convolutions, etc.
  - training loss function (cross entropy)
  - evaluation metrics (mIoU, pixel accuracy)

input

learn to

predict

remember last topic:

image classification

<u>somewhere</u> 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**

hard to get

input

learn to

predict

target (GT mask)

pixel-level labels

person
bicycle
background

Pascal dataset

(only) 11,530 fully-labeled images

http://host.robots.ox.ac.uk/pascal/VOC

Remember:

image-net has >14,000,000 images with **image-level labels** (tags)

Fully-supervised Semantic Segmentation

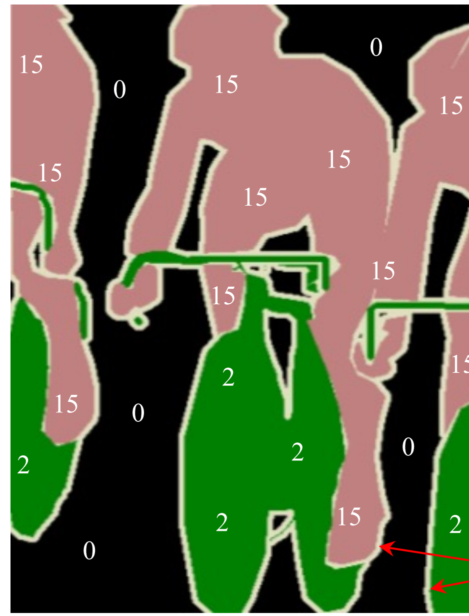
training uses **pixel-accurate Ground Truth**

input

learn to

predict

target (GT mask)



255 (void/undefined)

pixel-level labels

person
bicycle
background

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

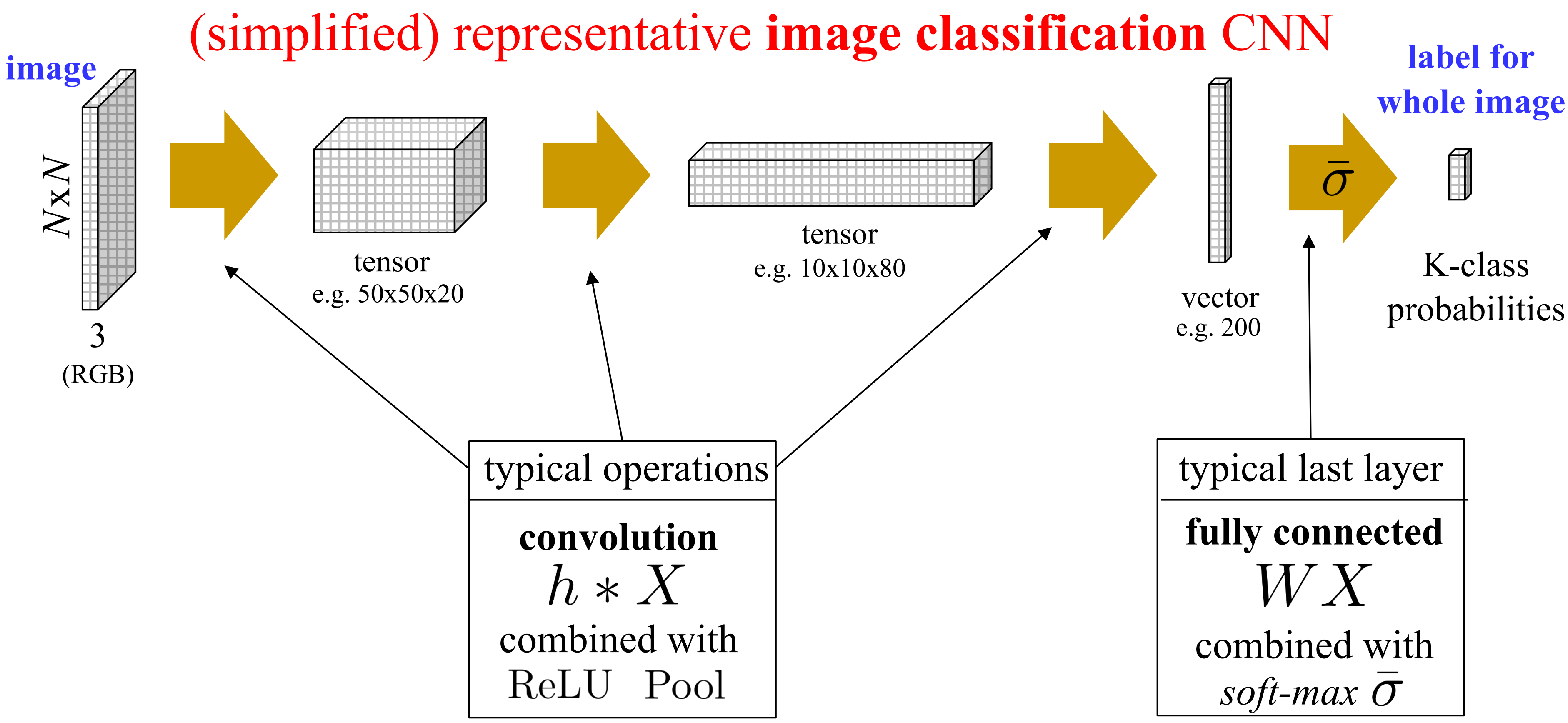
pixel labels (object classes) used in Pascal dataset:

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

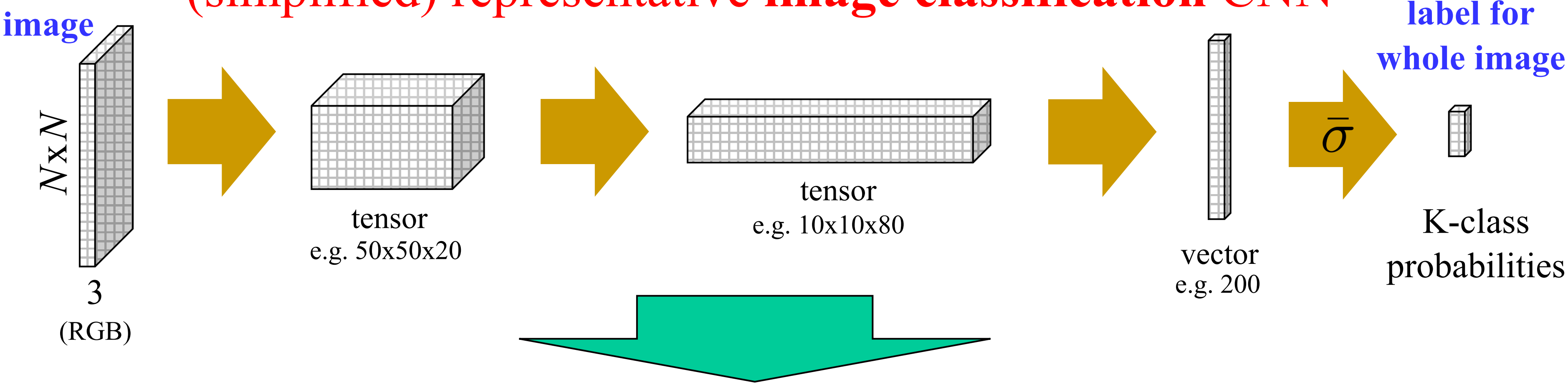


Q: How do we go from here to **image segmentation**?

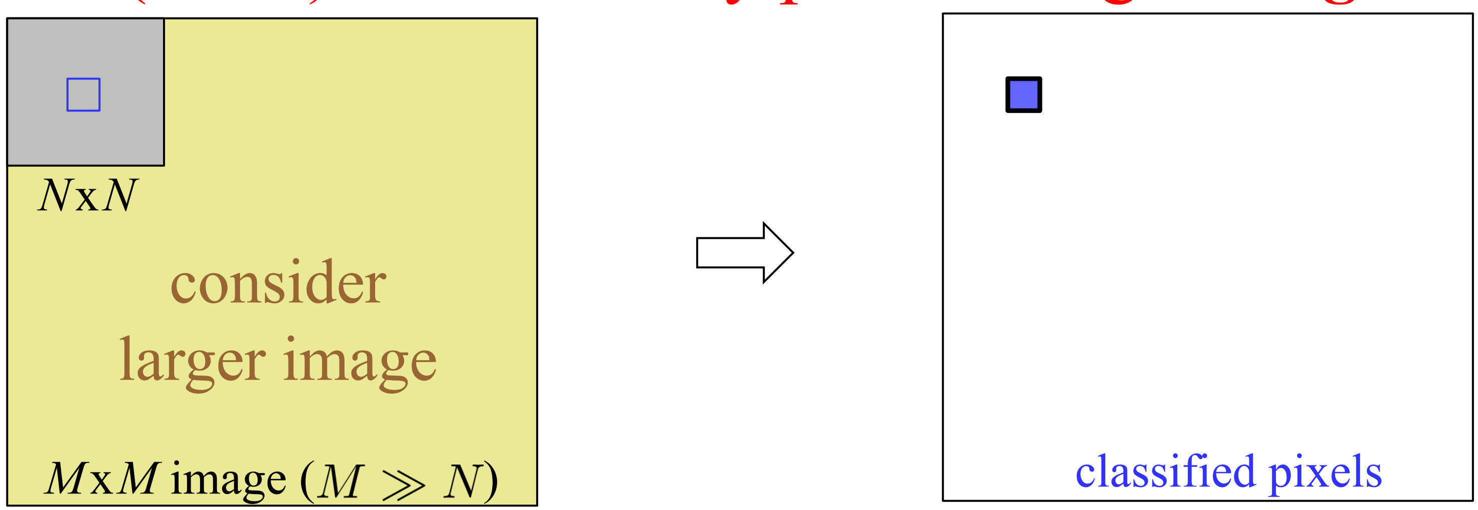
That is, how to extend NN methods for image classification to **classification of image pixels** ?

From Image to Pixel Labeling

(simplified) representative **image classification** CNN

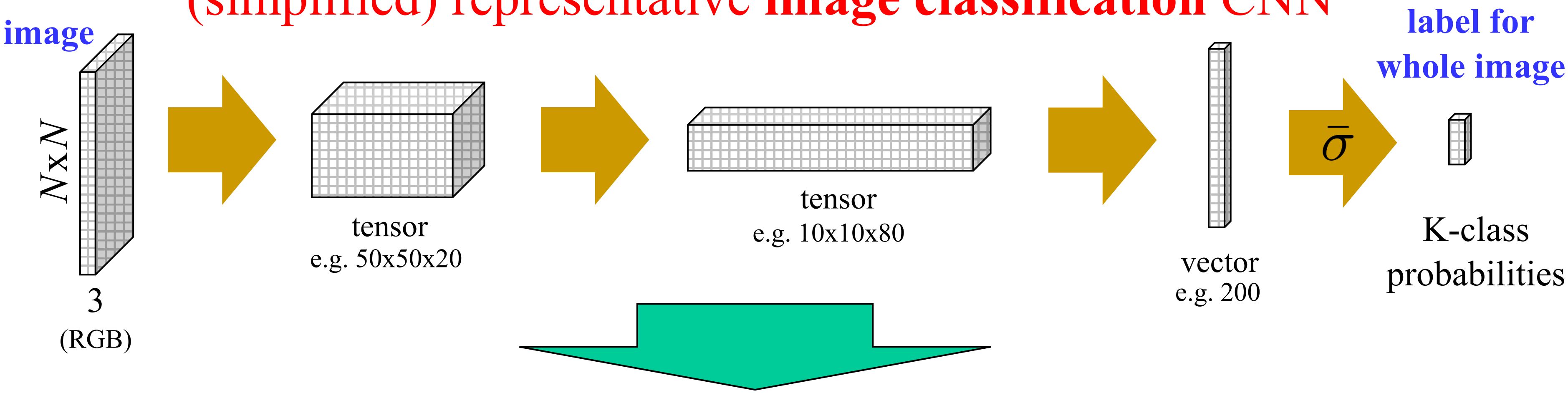


First (naïve) idea: classify pixels using ***sliding windows***

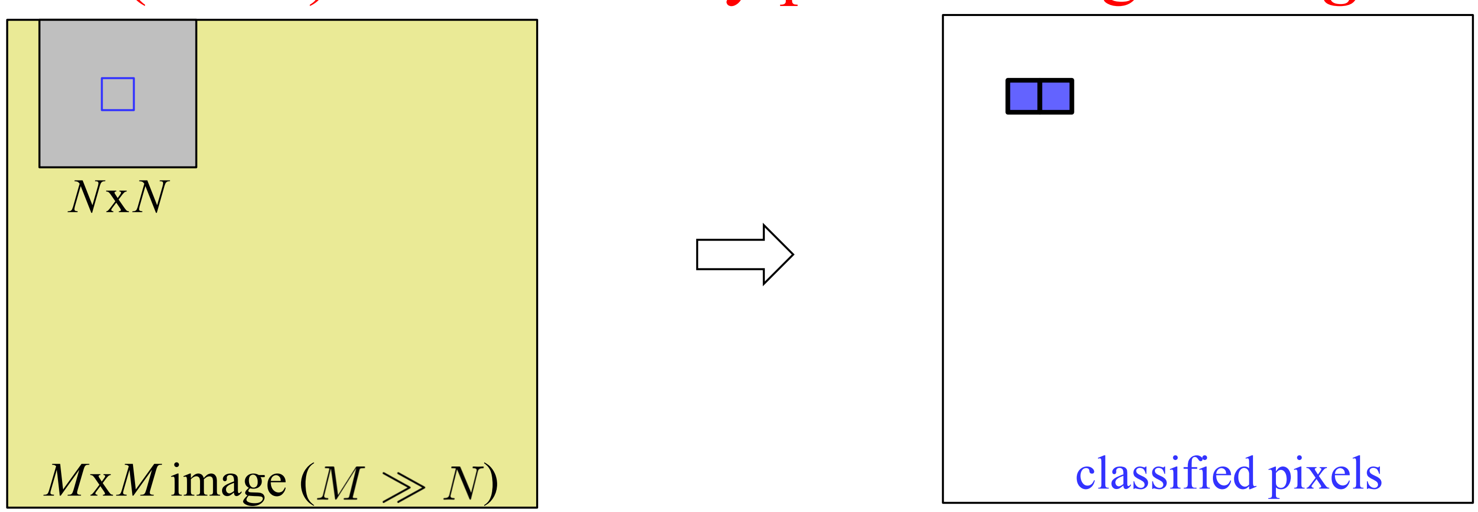


From Image to Pixel Labeling

(simplified) representative **image classification** CNN

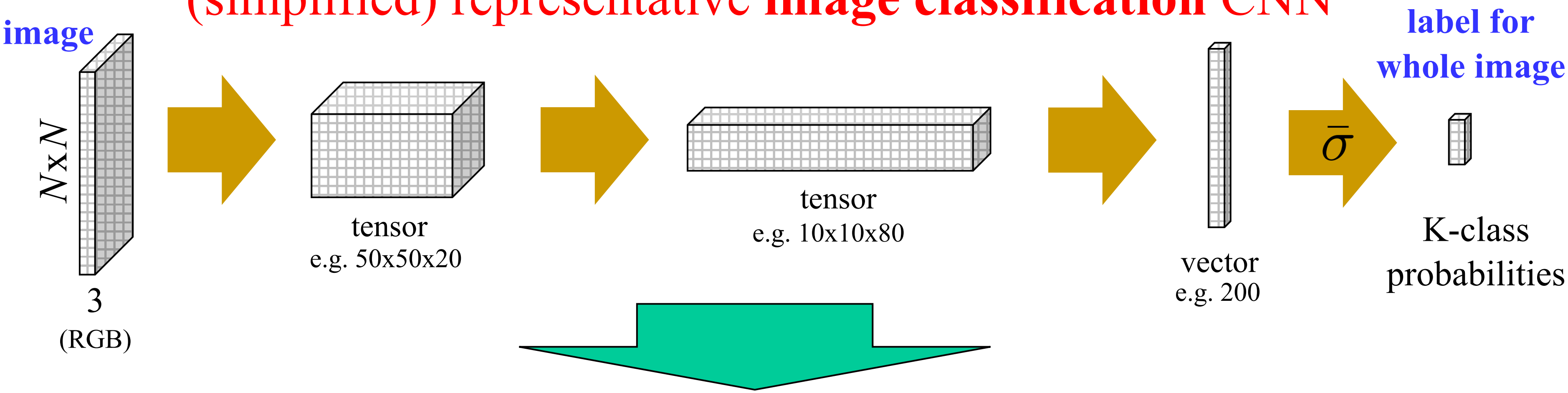


First (naïve) idea: classify pixels using ***sliding windows***

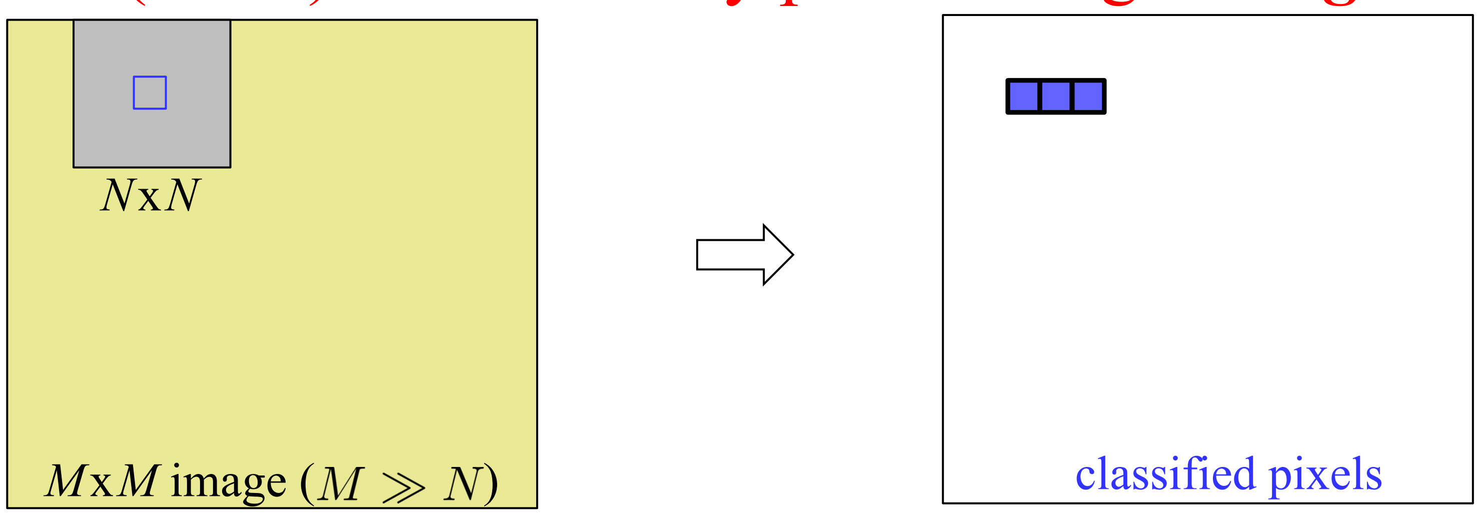


From Image to Pixel Labeling

(simplified) representative **image classification** CNN

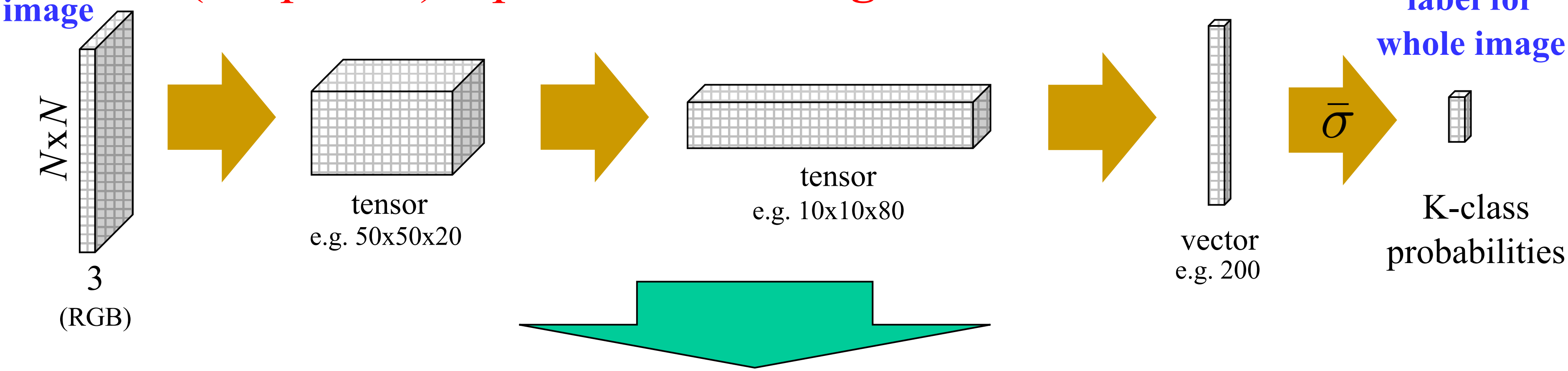


First (naïve) idea: classify pixels using ***sliding windows***

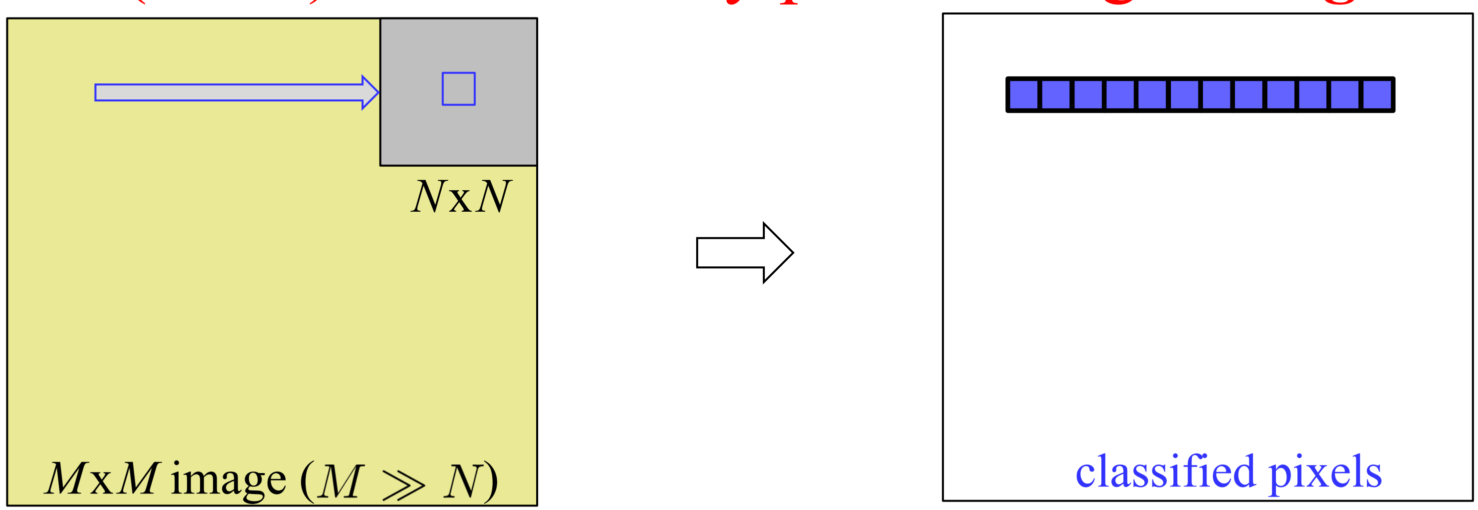


From Image to Pixel Labeling

(simplified) representative **image classification** CNN

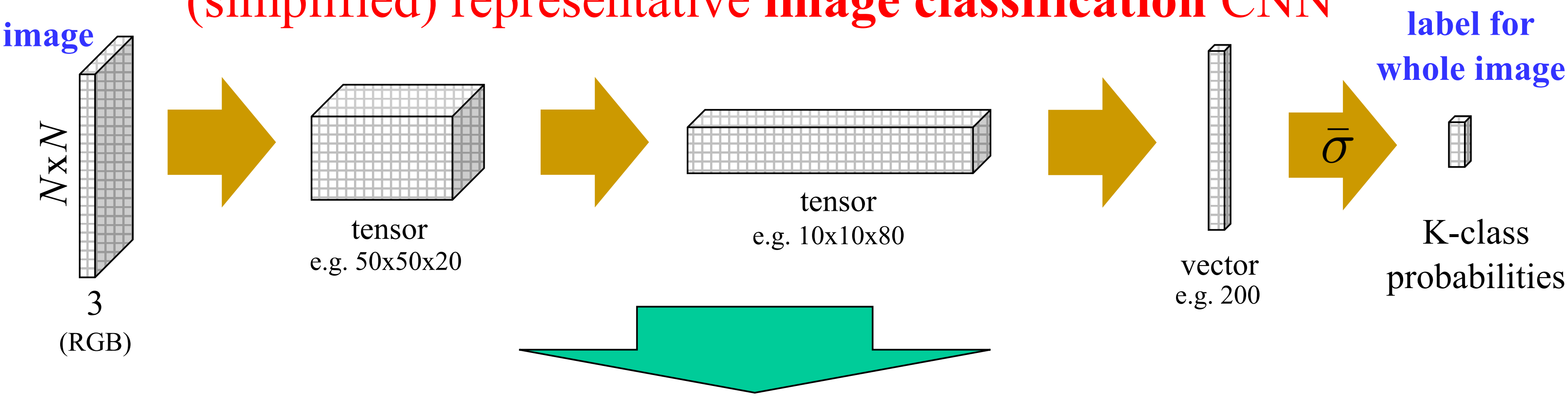


First (naïve) idea: classify pixels using ***sliding windows***

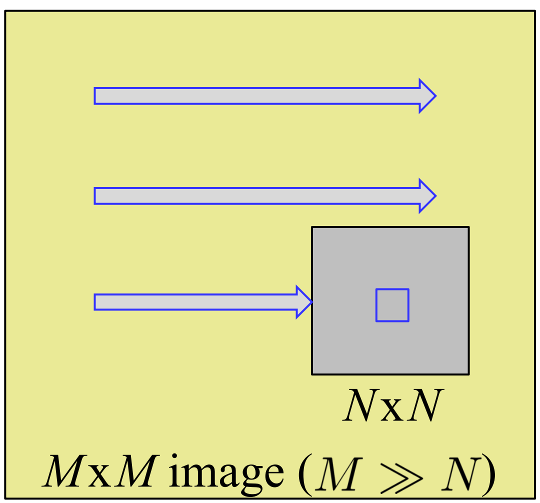


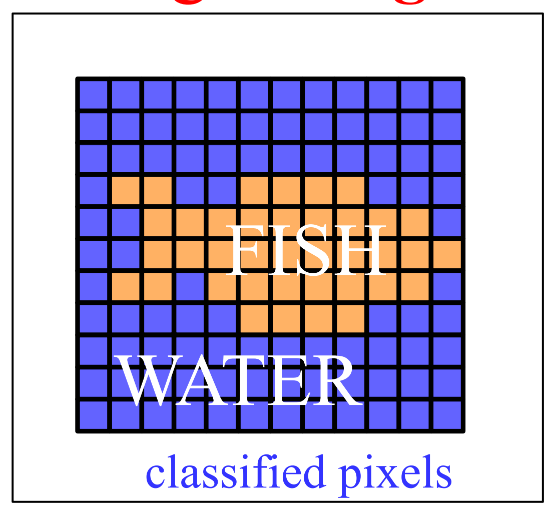
From Image to Pixel Labeling

(simplified) representative **image classification** CNN



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





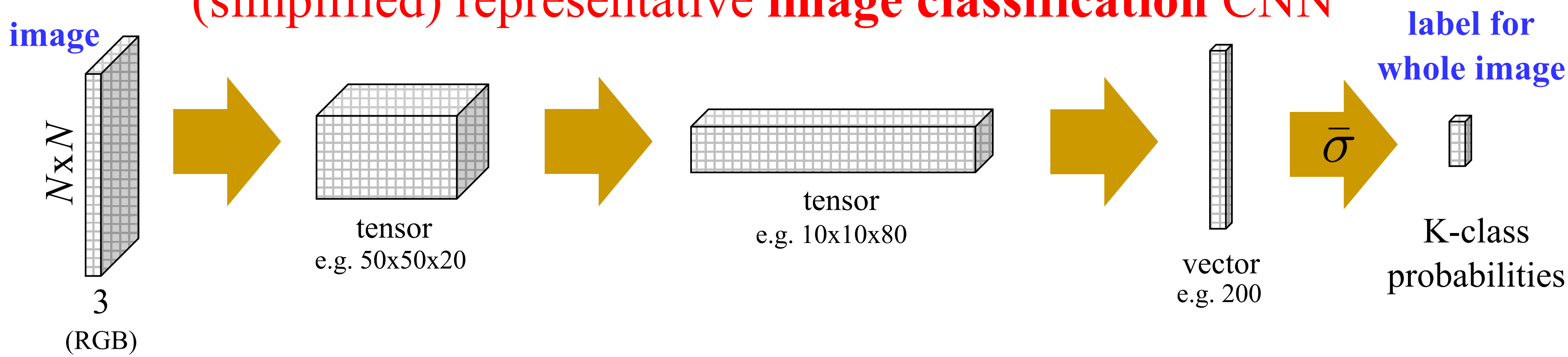
NOTE: here classification CNN <u>trained on image-level tags</u> segments image pixels

semantic segmentation

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.

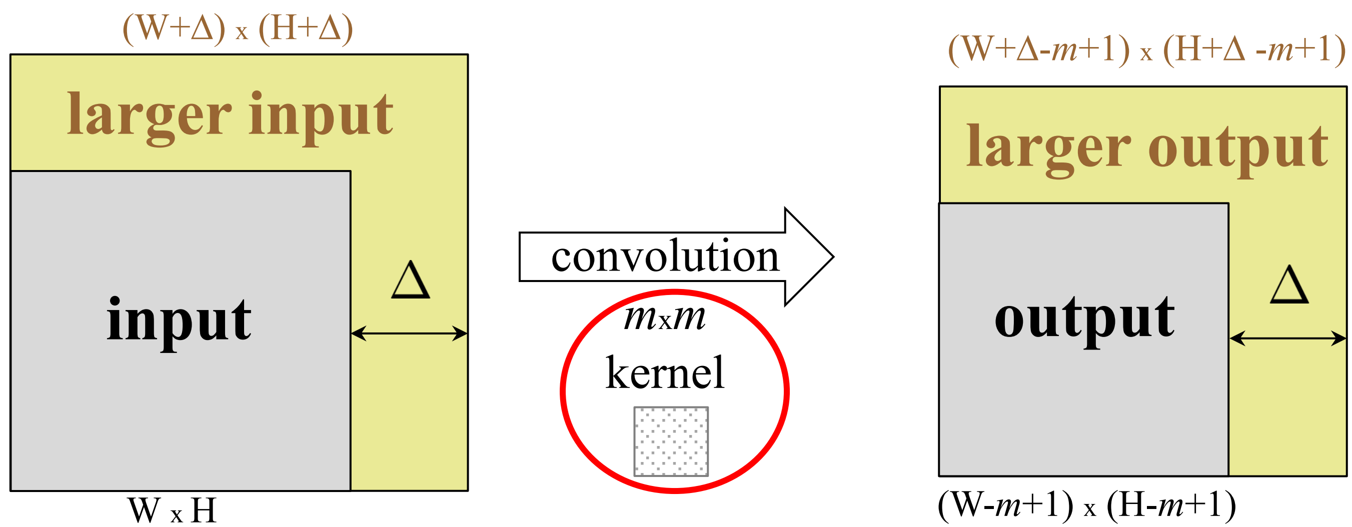
From Image to Pixel Labeling

(simplified) representative **image classification** CNN



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

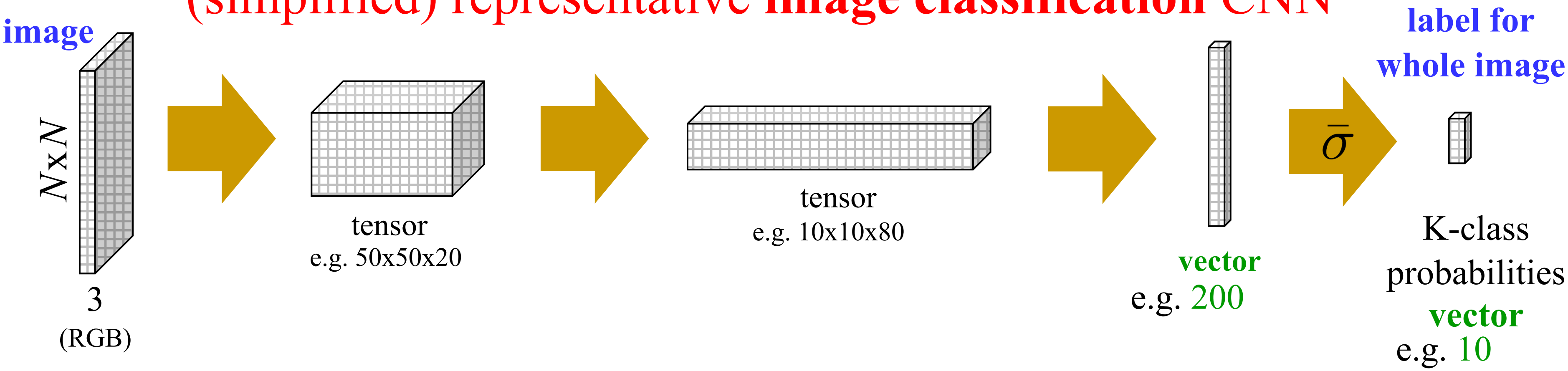
Key insight:



using the same kernel

From Image to Pixel Labeling

(simplified) representative **image classification** CNN

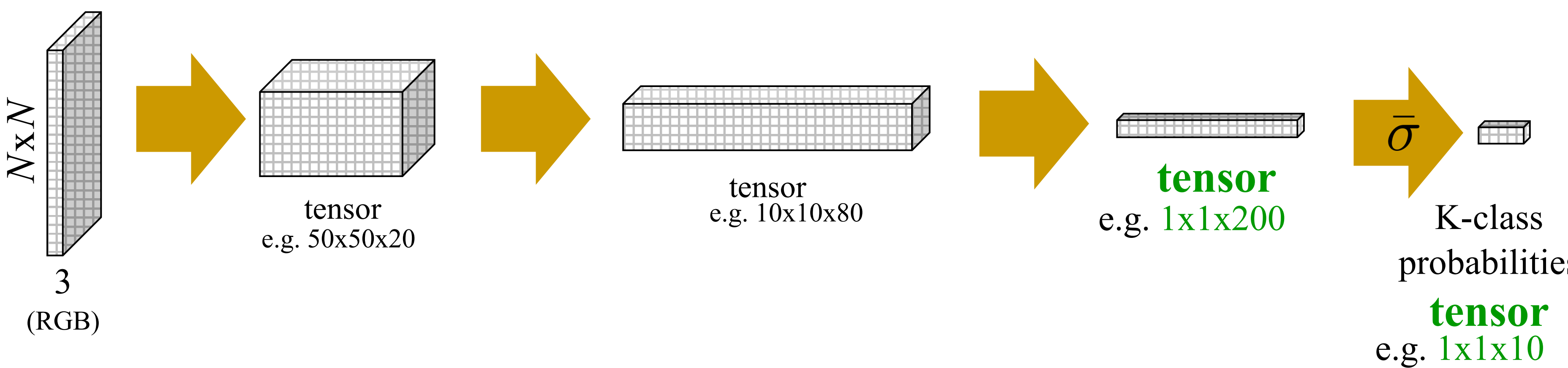


Better idea: convolutional kernel can be applied to input of any size!

Assume **all layers are convolutional**.

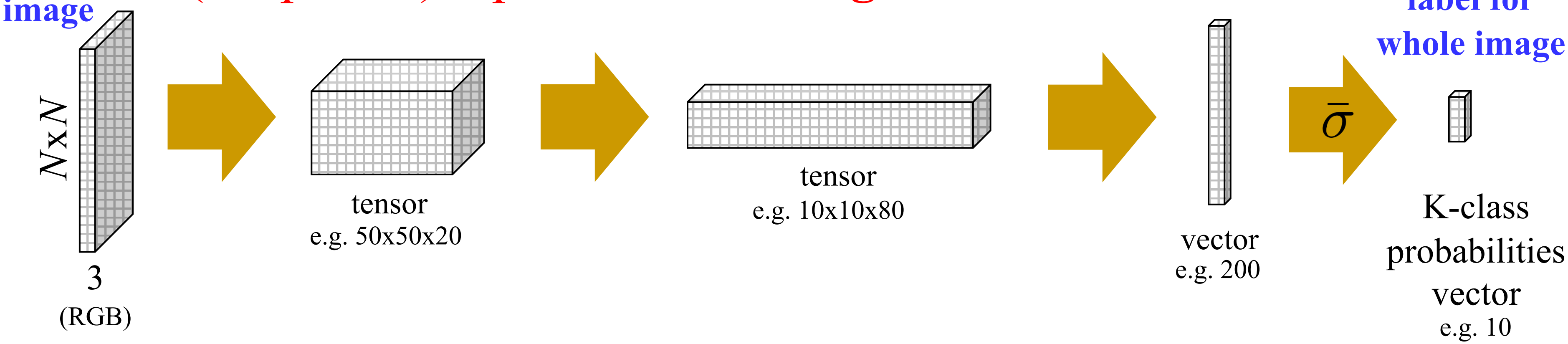
What about last (fully connected) layer? **No problem:**

$$W_{10\times 200}X \equiv h^{200\to 10}_{1\times 1} * X$$



From Image to Pixel Labeling

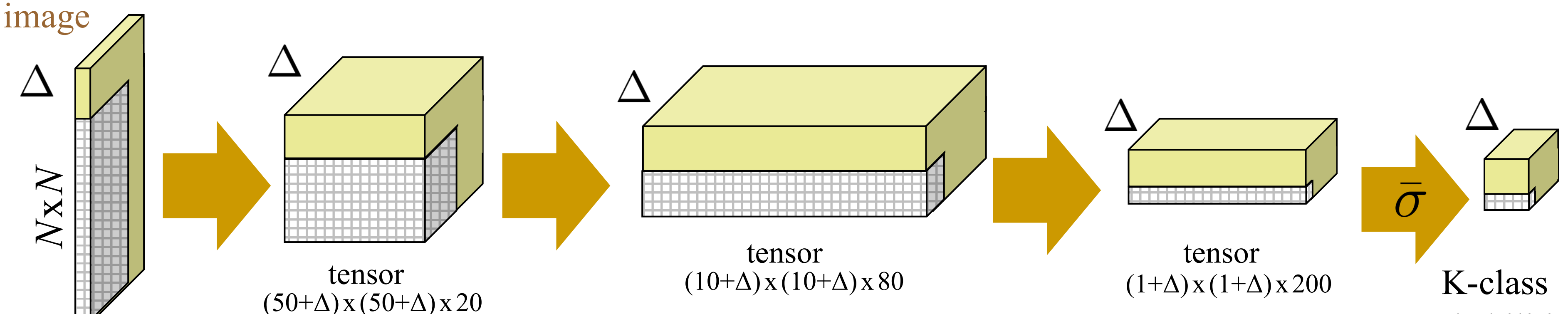
(simplified) representative **image classification** CNN



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)

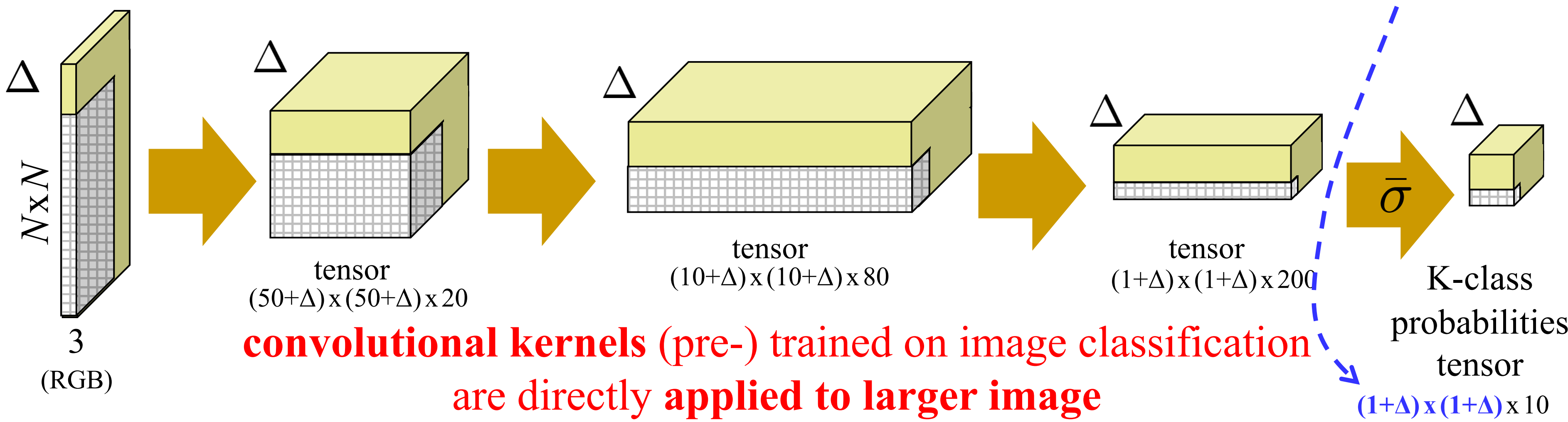


convolutional kernels (pre-) trained on image classification are directly **applied to larger image**

From Image to Pixel Labeling

Now, network output has some spatial resolution!

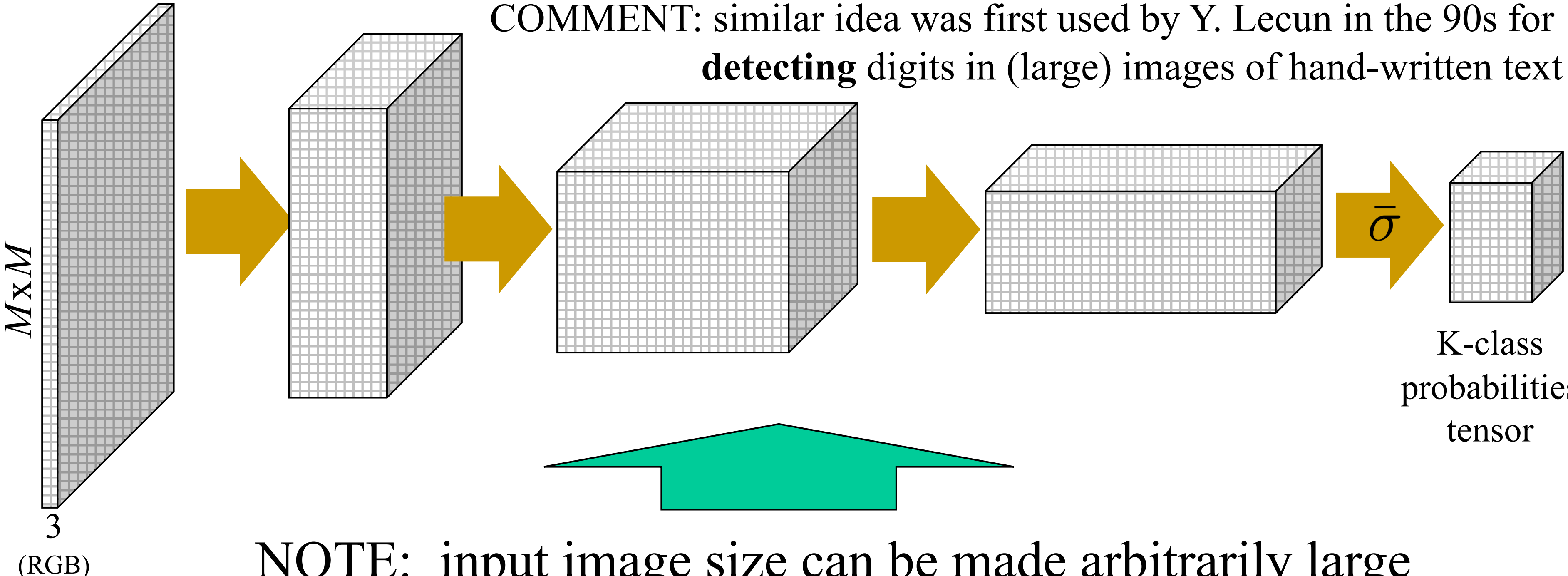
Intuition: K-class probabilities in the gray part of the output have "*receptive field*" in the gray part of the input image, while yellow output is supported by different *N*x*N* sections of the larger image



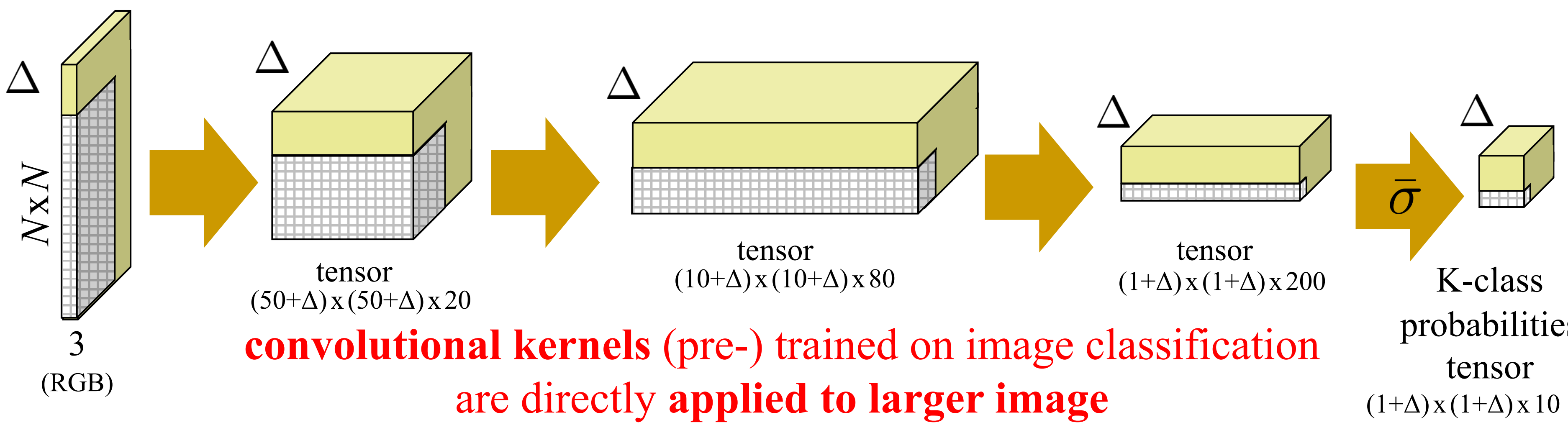
convolutional kernels (pre-) trained on image classification are directly **applied to larger image**

Fully Convolutional Network (FCN)

COMMENT: similar idea was first used by Y. Lecun in the 90s for **detecting** digits in (large) images of hand-written text

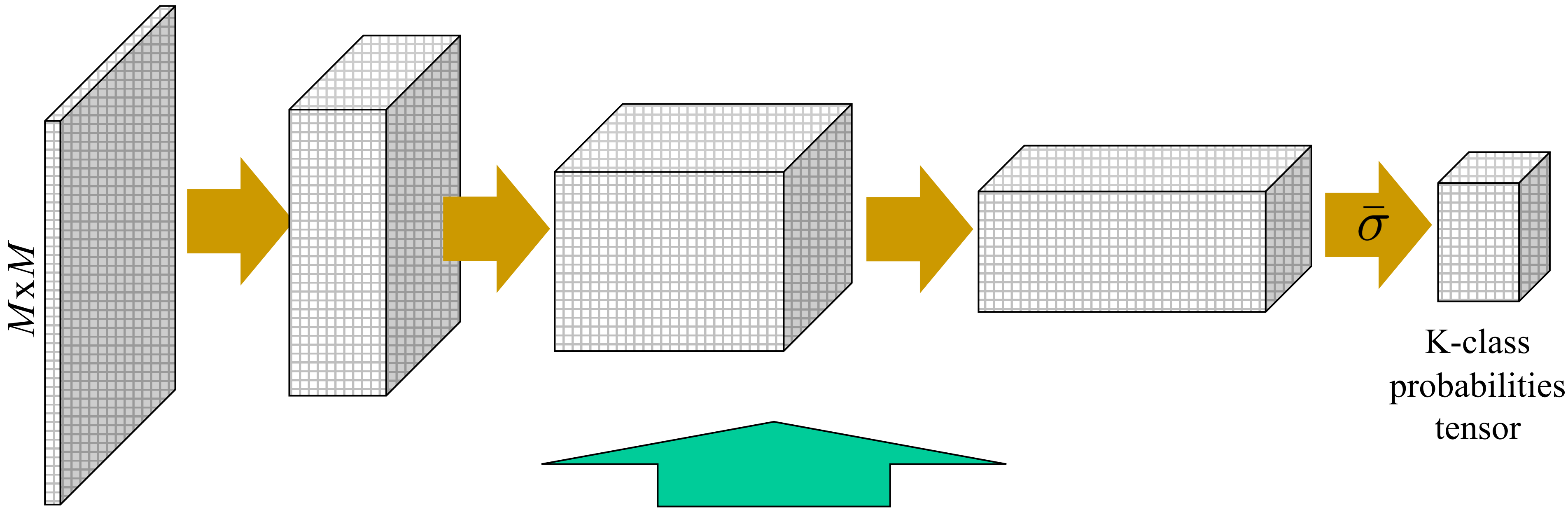


NOTE: input image size can be made arbitrarily large

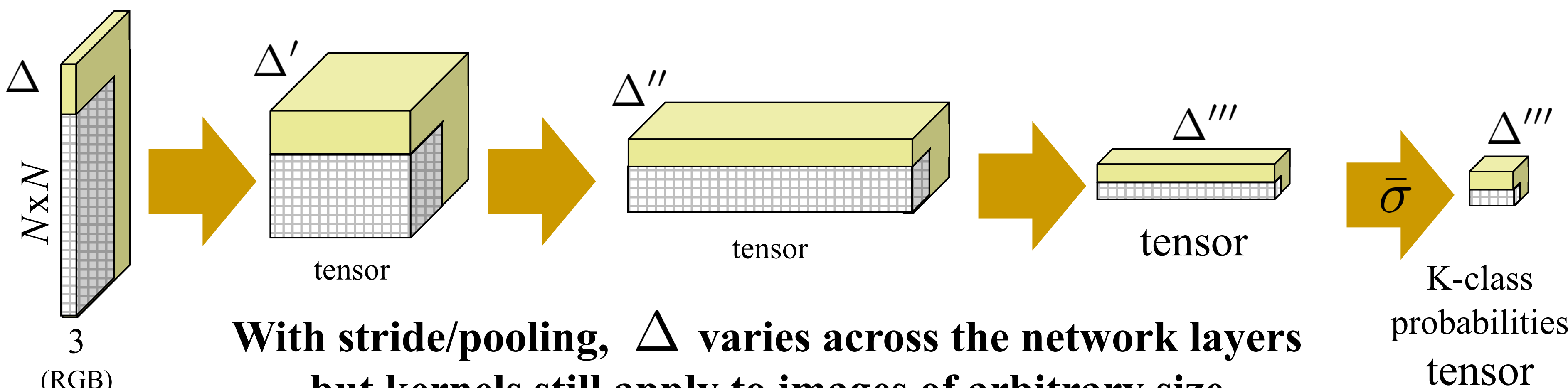


convolutional kernels (pre-) trained on image classification are directly **applied to larger image**

Fully Convolutional Network (FCN)



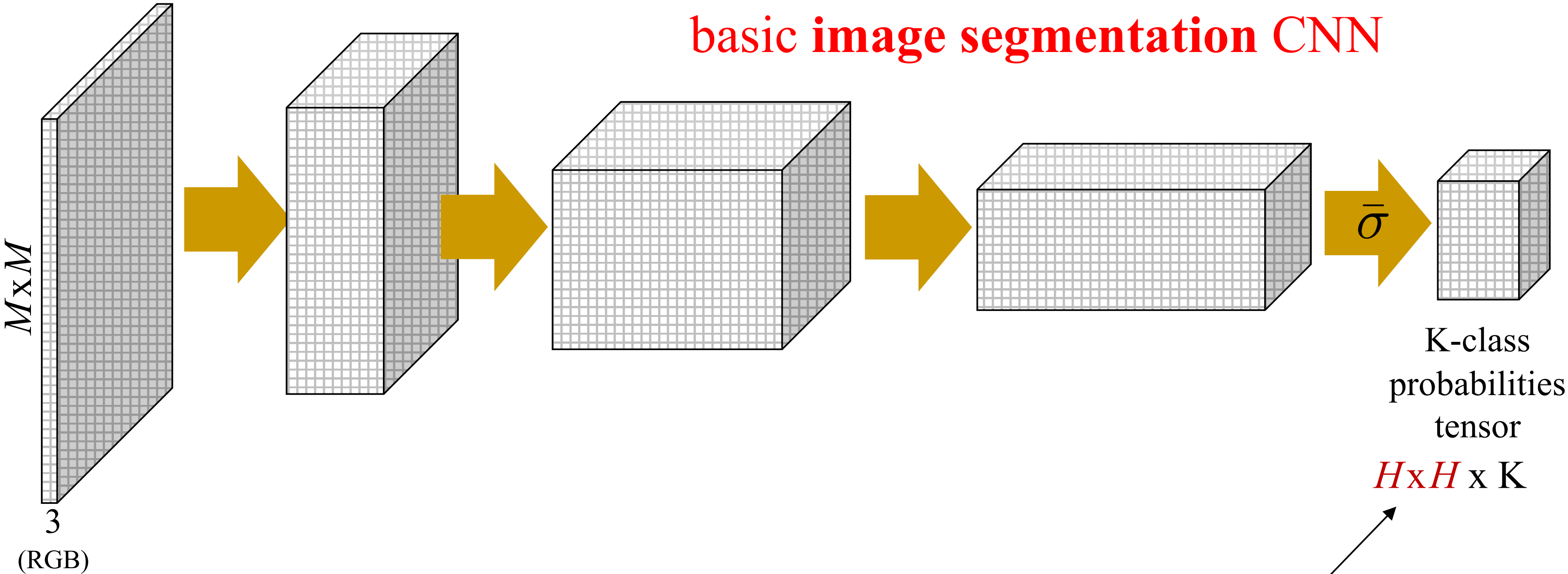
NOTE: input image size can be made arbitrarily large



With stride/pooling, Δ varies across the network layers but kernels still apply to images of arbitrary size

Fully Convolutional Network (FCN)

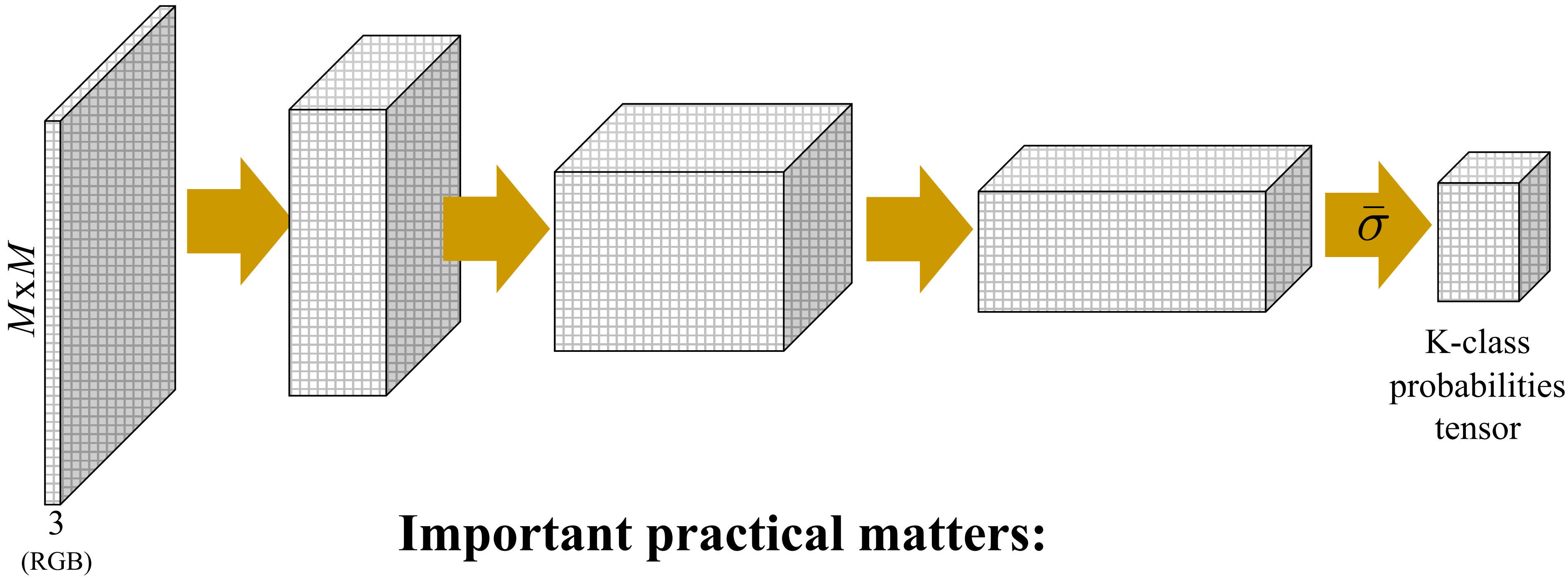
basic **image segmentation** CNN



NOTE: since this network's prediction/output has <u>spatial resolution</u>, 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)

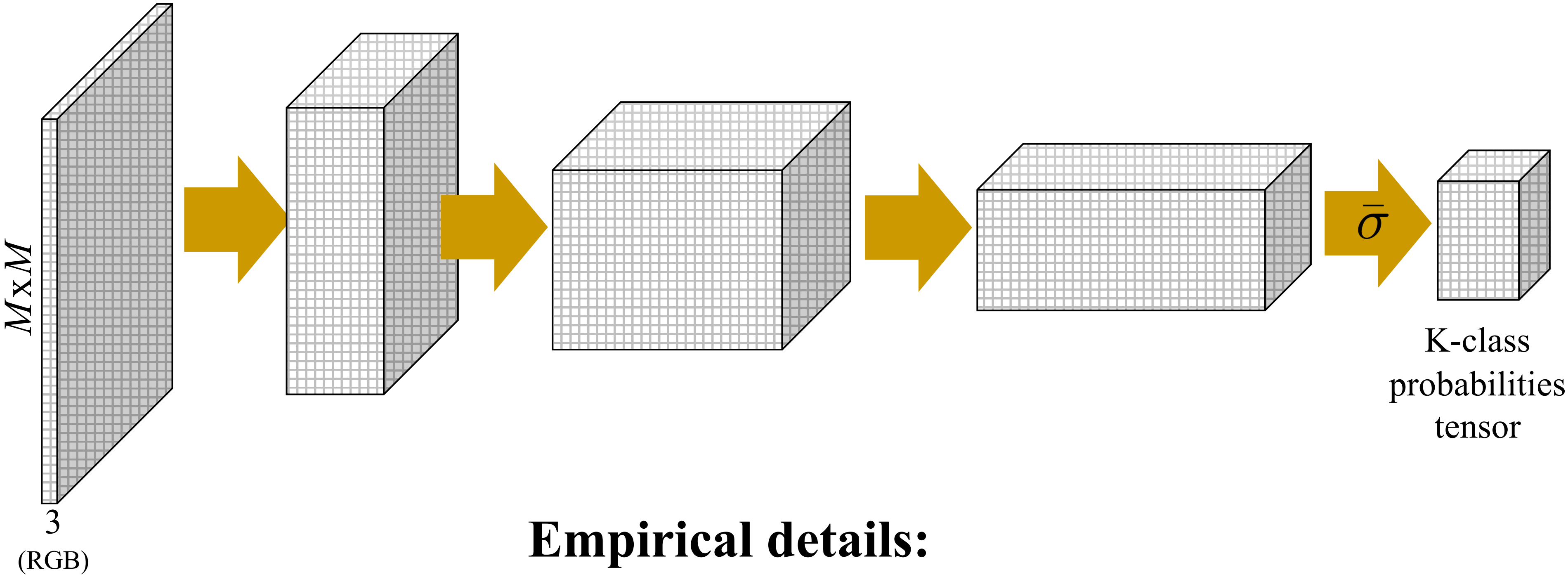


Important practical matters:

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

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

- **FCN** (2015)
  fully convolutional network for segmentation
  skip connections

  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

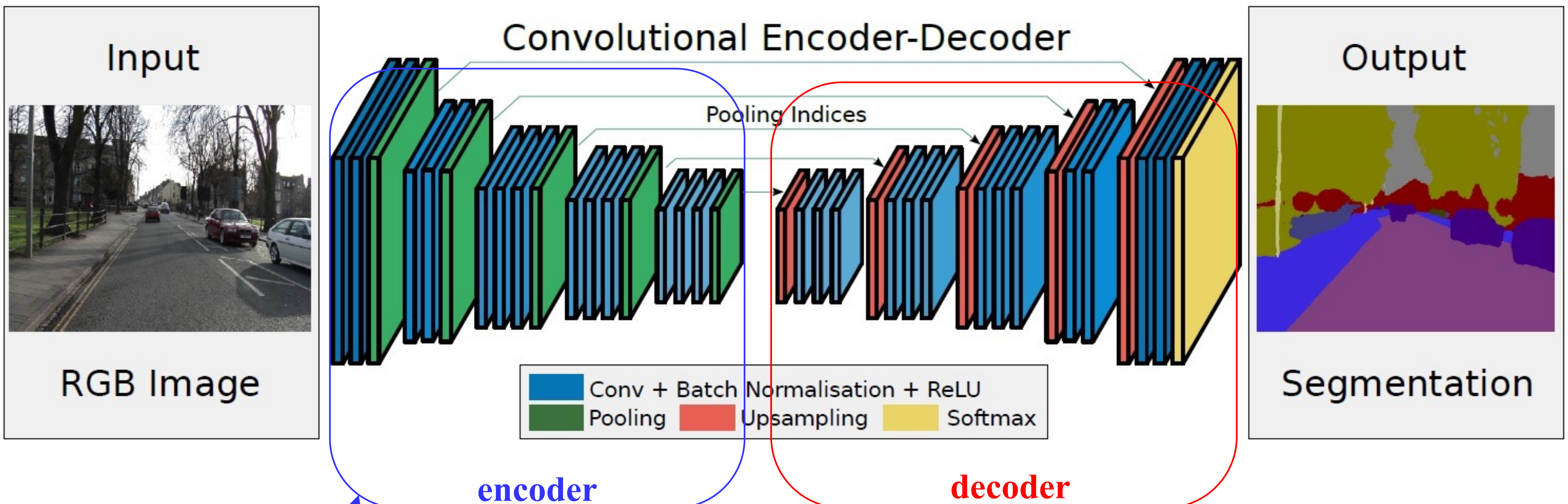
  U-net: Convolutional networks for biomedical image segmentation
  Ronneberger, Fischer, Brox - MICCAI 2015 / *Nature Methods* 2019

- **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



(upsampling part)

important:
encoder convolutional layers are typically pre-trained on *image net*
Encoder's main goal is to learn good discriminative features

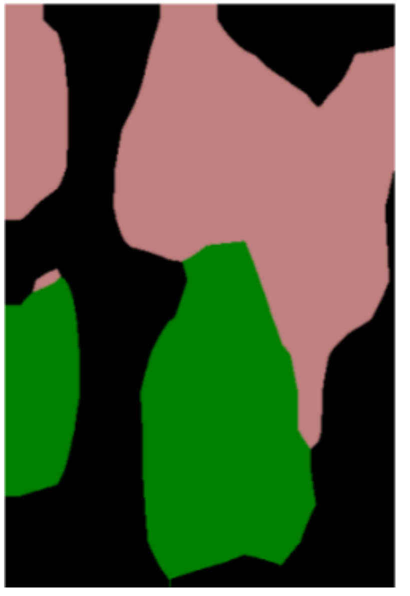
decoder upsamples encoder-generated features
(classification delayed to the network end)

Comment: feature dimensions at the encoder output should be gradually decreased with upsampling (too expensive, otherwise)

Need for upsampling

Ground truth target

Predicted segmentation



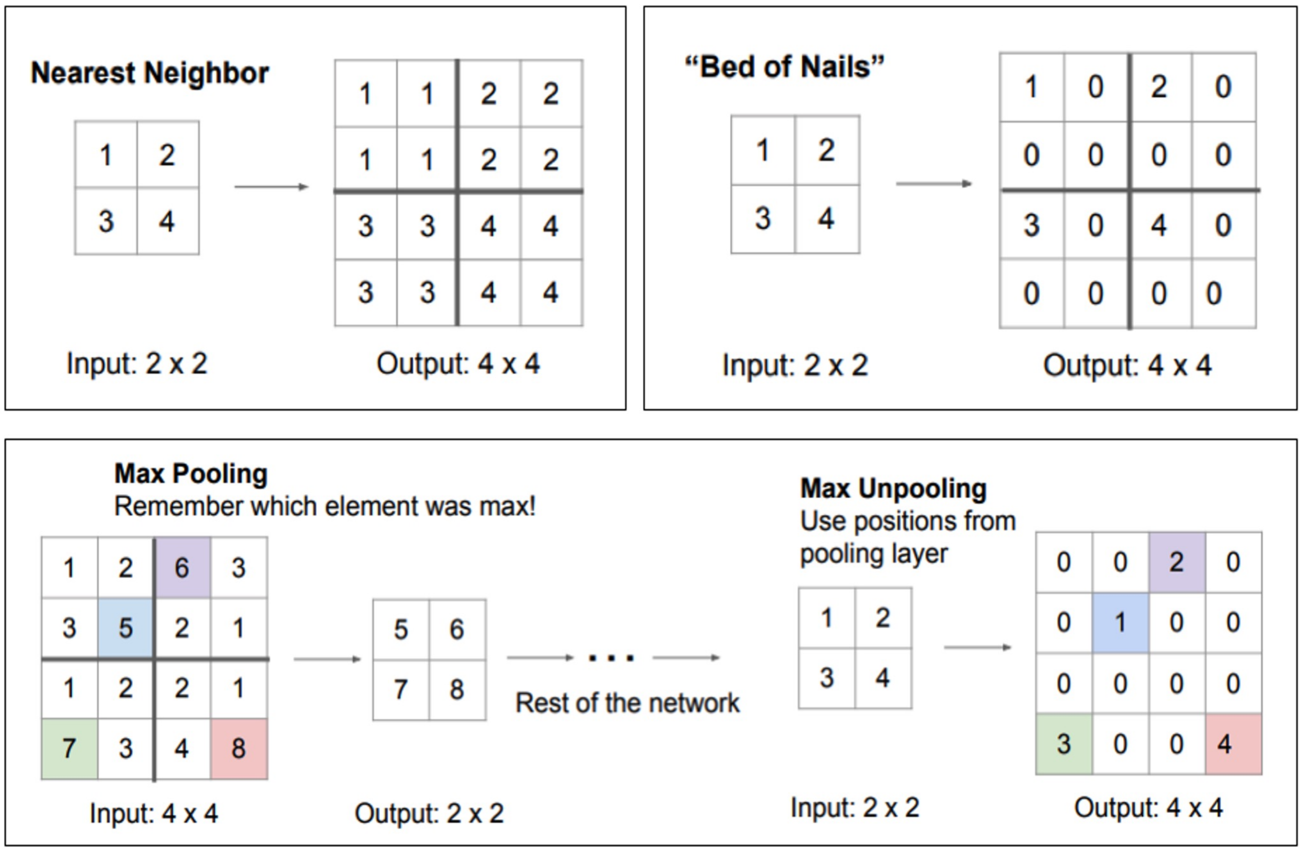
soft-max applied directly to encoder's output features

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

COMMENT: some upsampling steps in the decoder could be learned, while some are hand-engineered. (The same comment is also valid for the encoder)

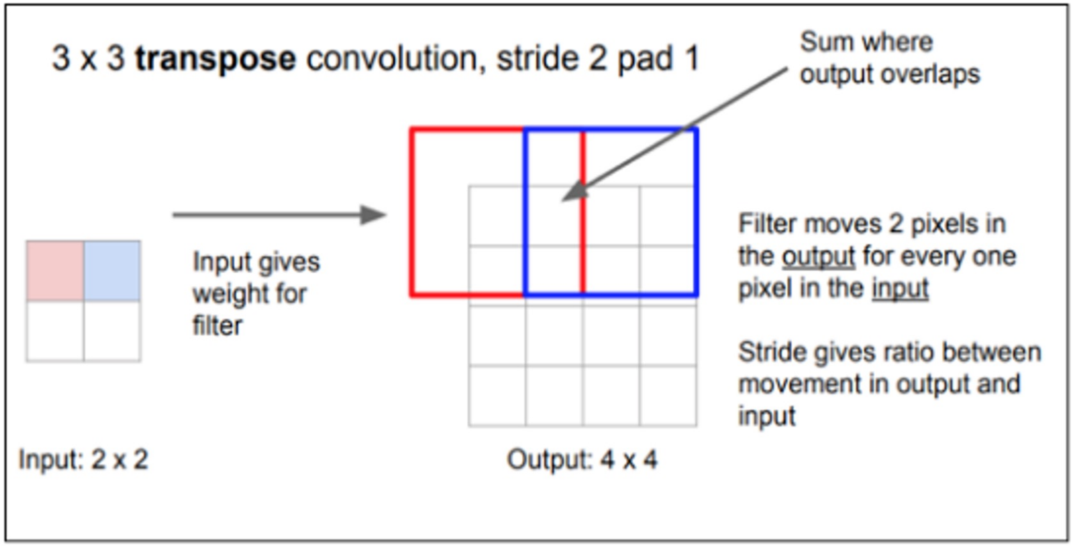
Methods for Upsampling

illustrations credit: Fei-Fei Li

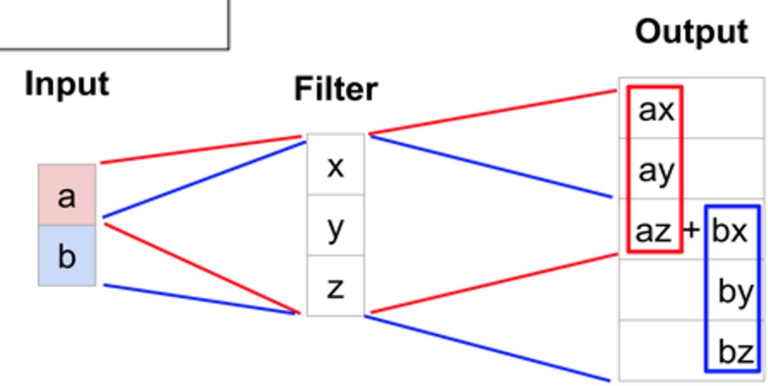


Methods for Upsampling

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

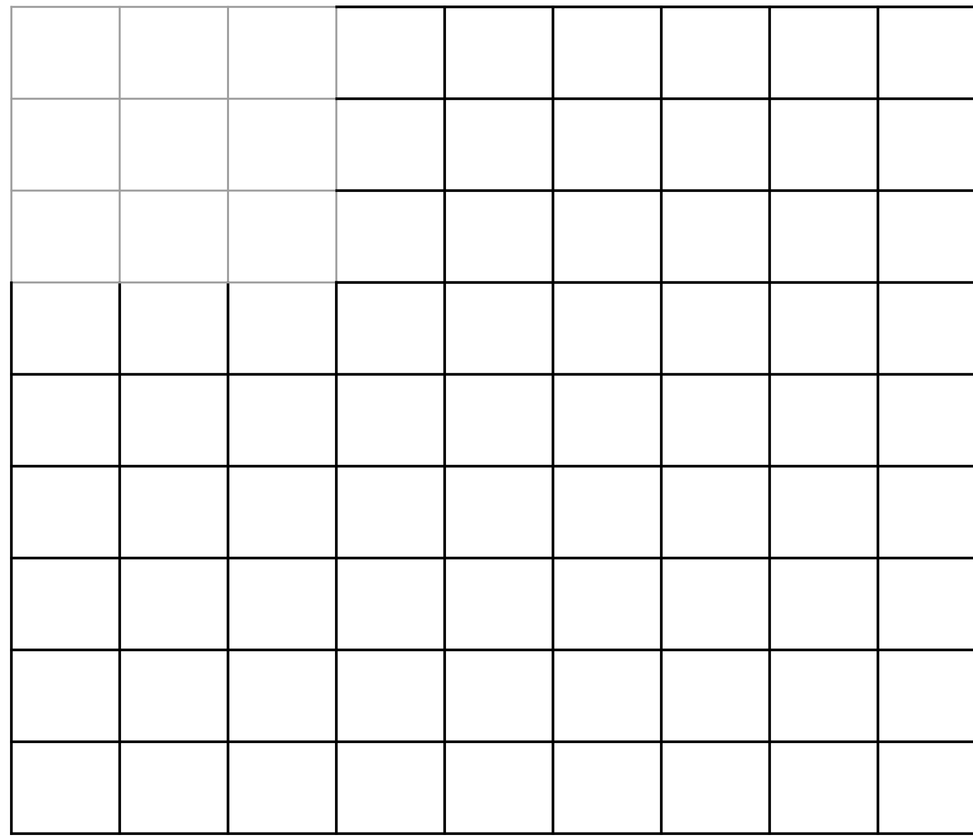
Input Image

Kernel

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

kernel=3x3
stride=2
padding=1

Output Image



illustrations credit: Soroosh Baselizadeh

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

Kernel

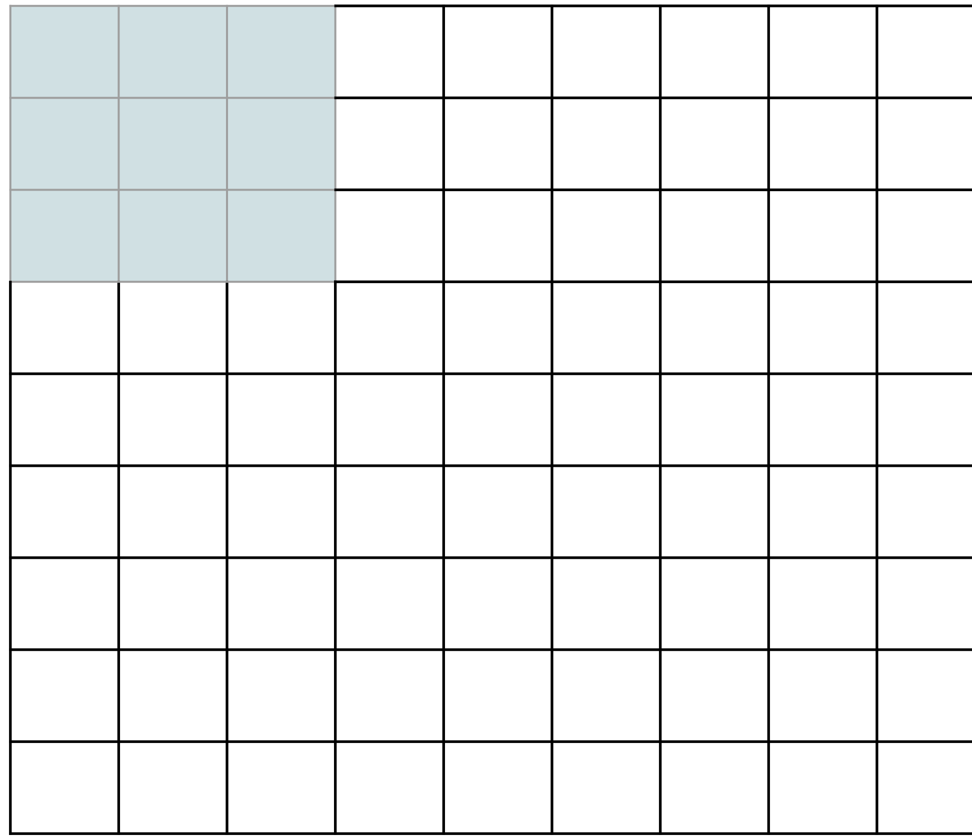
0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

Element x Kernel

0	0	0
0	0	0
0	0	0

kernel=3x3
stride=2
padding=1

Output Image



illustrations credit: Soroosh Baselizadeh

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	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						

illustrations credit: Soroosh Baselizadeh

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

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						
0	0	0						
0	0	0						

illustrations credit: Soroosh Baselizadeh

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				

illustrations credit: Soroosh Baselizadeh

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

Kernel

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				

illustrations credit: Soroosh Baselizadeh

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

0.5	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		

illustrations credit: Soroosh Baselizadeh

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

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

illustrations credit: Soroosh Baselizadeh

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

illustrations credit: Soroosh Baselizadeh

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

Kernel

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

illustrations credit: Soroosh Baselizadeh

Transpose Convolution: Example

Added Result

Input Image

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

Kernel

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						

illustrations credit: Soroosh Baselizadeh

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

1.25	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				

illustrations credit: Soroosh Baselizadeh

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		

illustrations credit: Soroosh Baselizadeh

Transpose Convolution: Example

Added Result

Input Image

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

Kernel

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

illustrations credit: Soroosh Baselizadeh

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

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						

illustrations credit: Soroosh Baselizadeh

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

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

illustrations credit: Soroosh Baselizadeh

Transpose Convolution: Example

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

Input Image

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

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~~	~~0.75~~	~~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~~

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)

V. Dumoulin, and F. Visin. "A guide to convolution arithmetic for deep learning." *arXiv preprint arXiv:1603.07285* (2016).

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 though 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 $(s\text{-}1)$ zeros between pixels, then apply regular conv using the same kernel (see **example on the next slide**).

see Sections 4 in [1] and 3.3 in [2]

[1] – Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning." *arXiv preprint arXiv:1603.07285* (2016).

[2] - Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.

illustrations credit: Soroosh Baselizadeh

Fractionally-strided Convolution

Fractional Stride

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

Standard Convolution

kernel=3x3
stride=½
(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…

illustrations credit: Soroosh Baselizadeh

Fractionally-strided Convolution

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

standard convolution

with

kernel

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

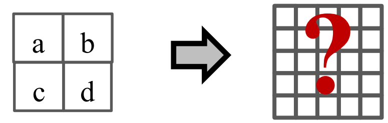
0	0.5	1	1.5	2	2.5	3
2	2.5	3	3.5	4	4.5	5
4	4.5	5	5.5	6	6.5	7
6	6.5	7	7.5	8	8.5	9
8	8.5	9	9.5	10	10.5	11
10	10.5	11	11.5	12	12.5	13
12	12.5	13	13.5	14	14.5	15

Output

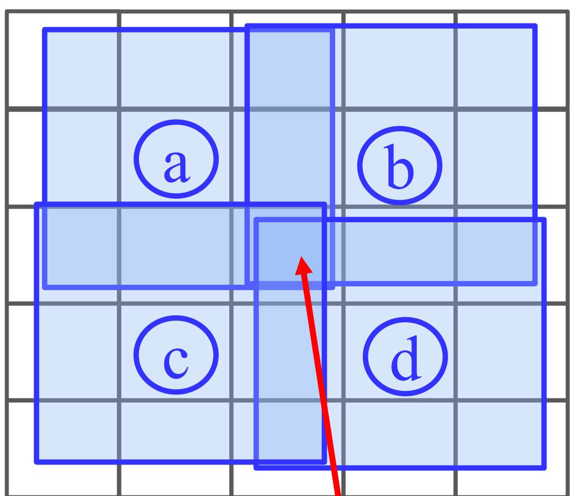
illustrations credit: Soroosh Baselizadeh

Transposed vs Fractionally-strided Convolution

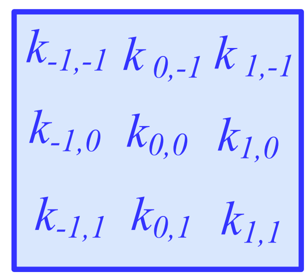
Upsampling Example:



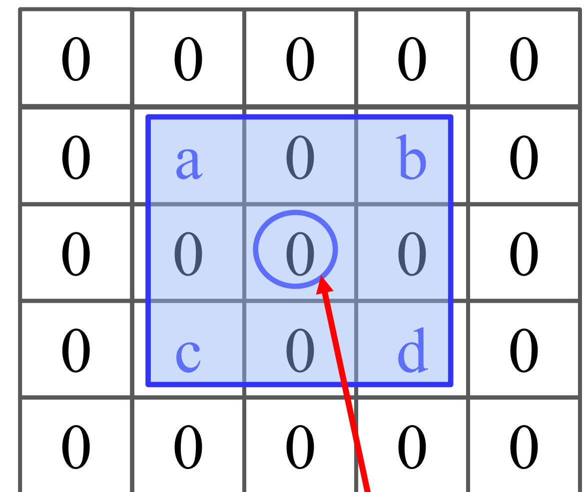
transpose convolution (slide 26)



kernel k



fractionally-strided convolution



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

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

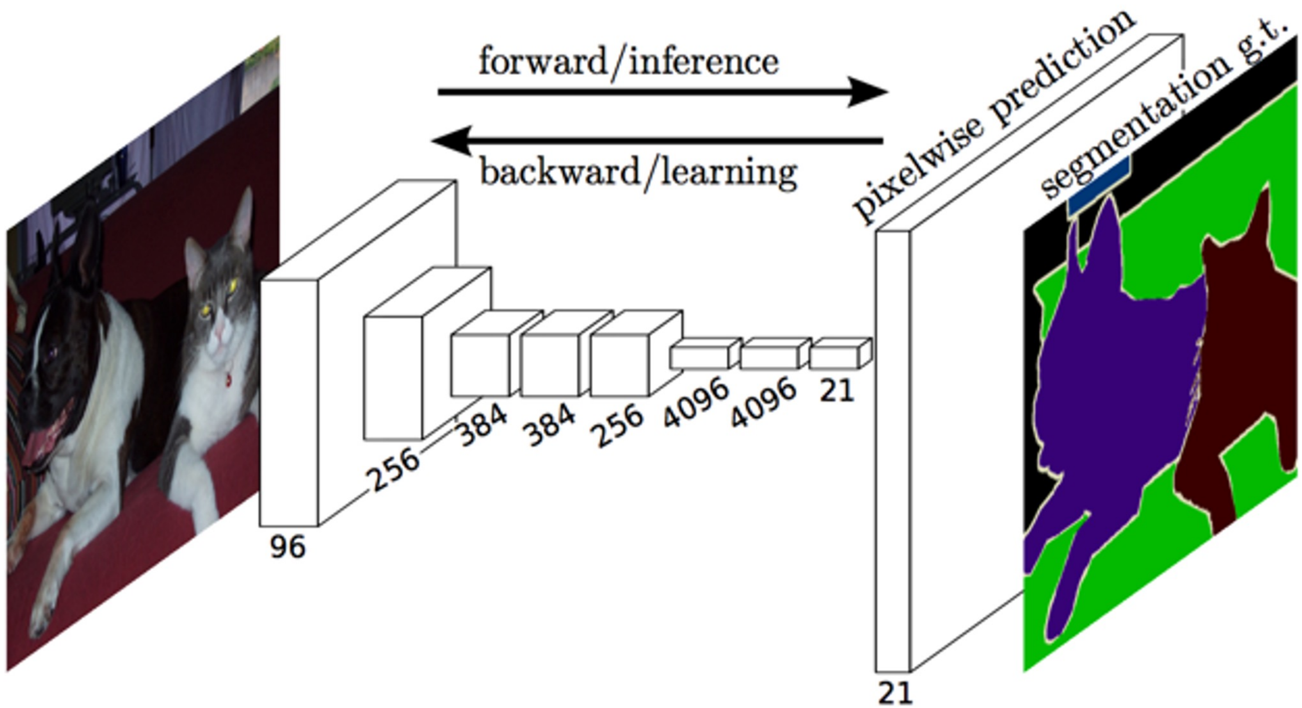
output of standard convolution using *k* with stride 1/2 for the pixel in the center

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

Homework exercise:

prove that 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.

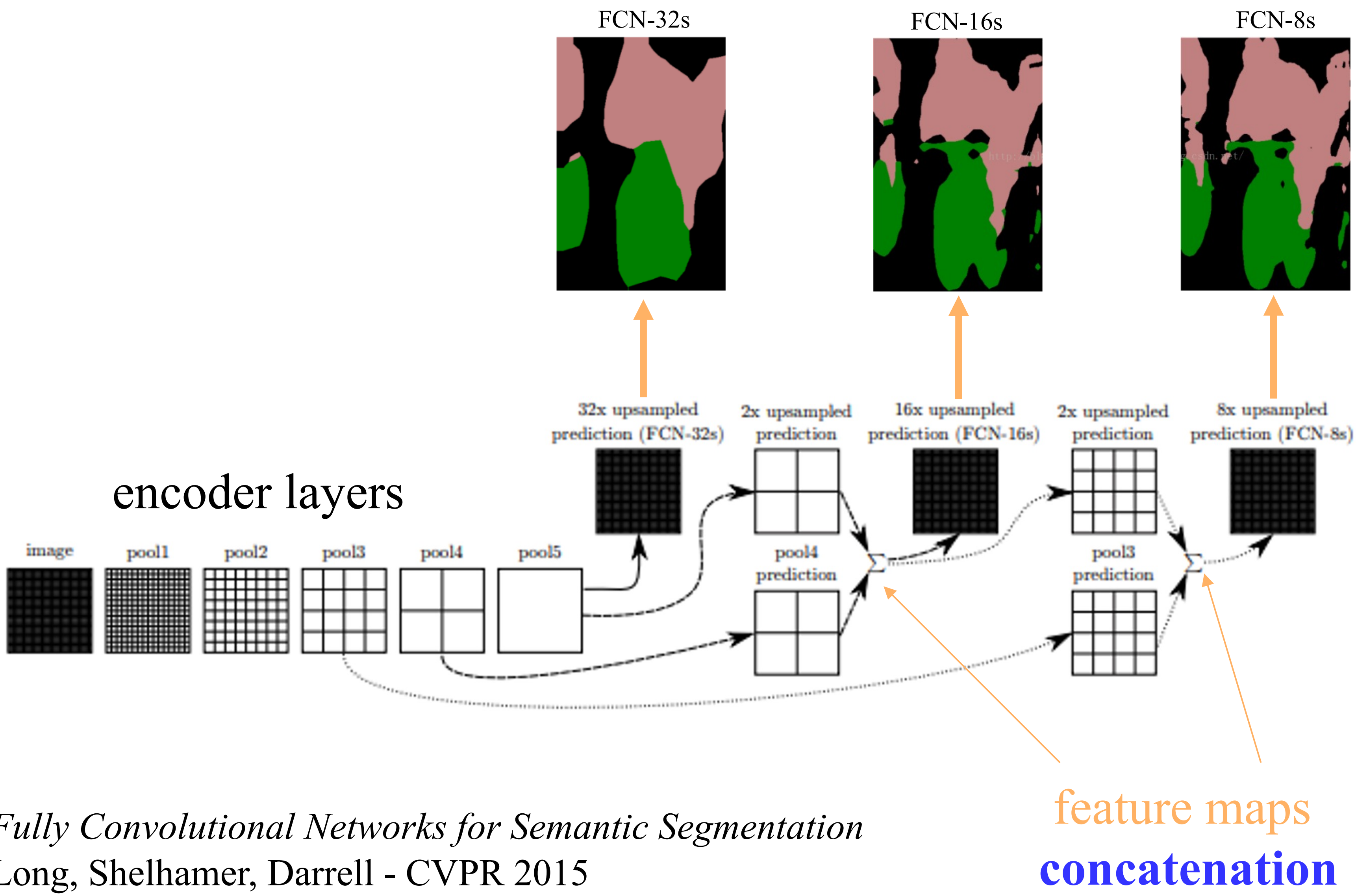
Fully Convolutional Networks (FCNs)



Upsample segmentation using "~~deconvoluton~~" *transposed convolution*

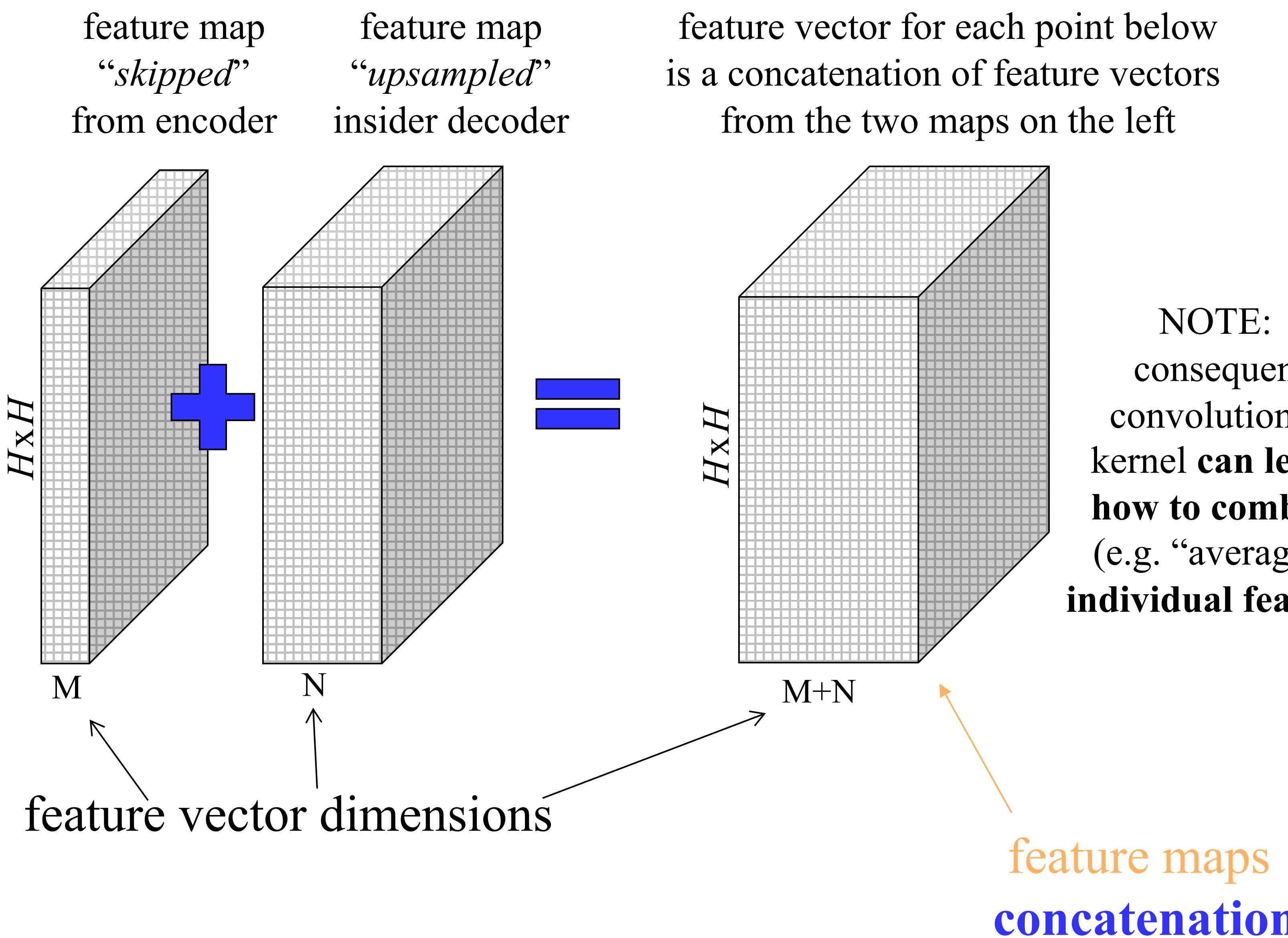
Fully Convolutional Networks for Semantic Segmentation
Long, Shelhamer, Darrell - CVPR 2015

Upsamping using **skip connections**



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

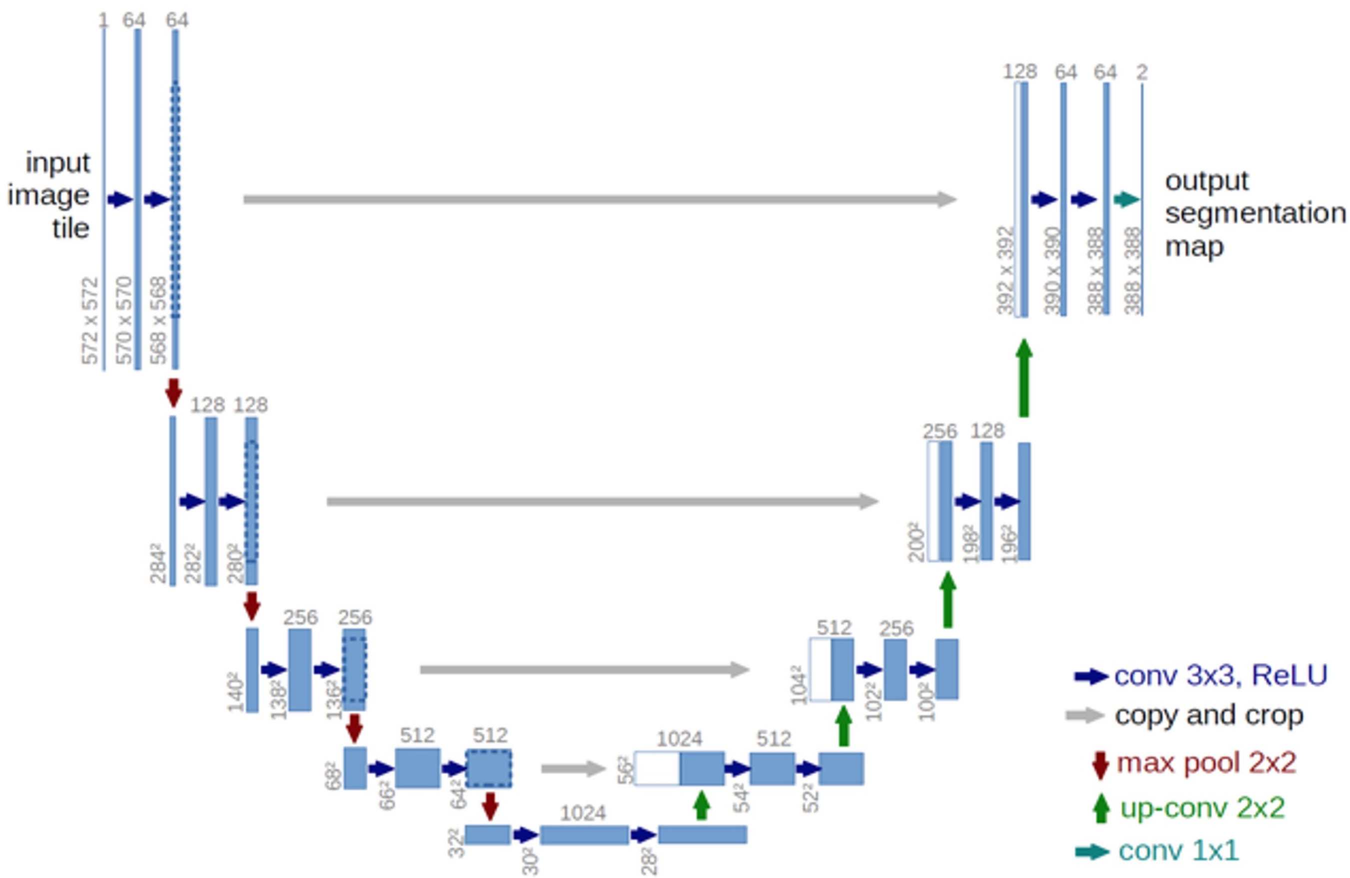
Skip connections: **concatenation**



NOTE:
consequent
convolutional
kernel **can learn**
how to combine
(e.g. "average")
individual features

U-net: expanding decoder with symmetry

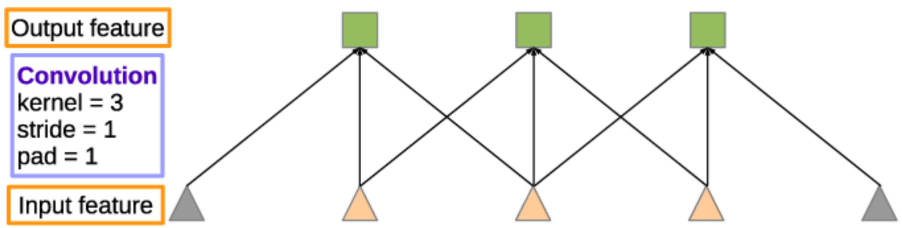
and many **skip connections**



U-net: Convolutional networks for biomedical image segmentation
Ronneberger, Fischer, Brox - MICCAI 2015 (now in *Nature Methods* 2019)

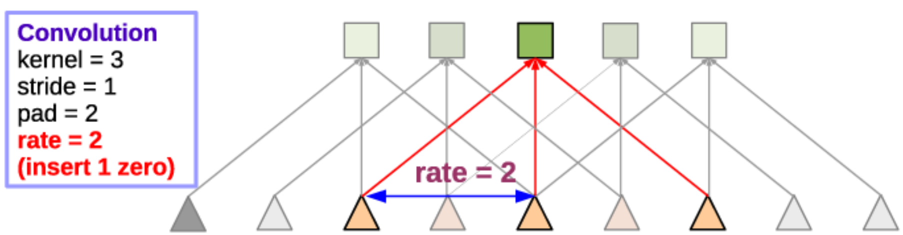
DeepLab

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



(a) Sparse feature extraction

standard 3x3 convolution



(b) Dense feature extraction

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

(GT mask)
pixel-precise target

$$\bar{\sigma}^p = \left(\bar{\sigma}_1, \bar{\sigma}_2, ..., \bar{\sigma}_K\right)$$

prediction at each pixel p

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

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

sum of
negative log-likelihoods (NLL)

Loss over image i :

$$\sum_{p \in \mathrm{I}_i} \overbrace{\sum_k -\bar{\mathbf{y}}^p_k \ln \bar{\sigma}^p_k}^{\textbf{cross entropy at p}} = \boxed{-\sum_{p \in I_i} \ln \bar{\sigma}^p_{\mathbf{y}^p}}$$

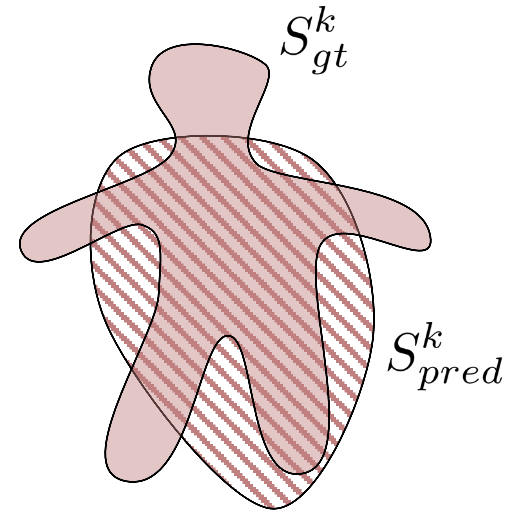
Total loss should also sum over all images i

(Validation) Quality Metrics

image sample i

network prediction

(GT mask)
pixel-precise target



- *Mean intersection over union* $\quad \text{mIoU} = \dfrac{1}{K}\sum_{k}\dfrac{|S_{gt}^{k} \cap S_{pred}^{k}|}{|S_{gt}^{k} \cup S_{pred}^{k}|} \in [0,1]$

  (focus on segments/classes, object sizes are irrelevant)

- There are also accuracy measures focused on pixels

  (what percentage of pixels is correctly classified)