

DDA4220 · Fall 2026

Lecture 8

Deep Learning and Applications

Dense prediction and semantic segmentation

Zhen Liu

CUHK-Shenzhen

Lecture outline

01 Pixel prediction

02 Upsampling and decoders

03 Training and evaluation

04 Spatial context

* denotes optional slides

Semantic segmentation

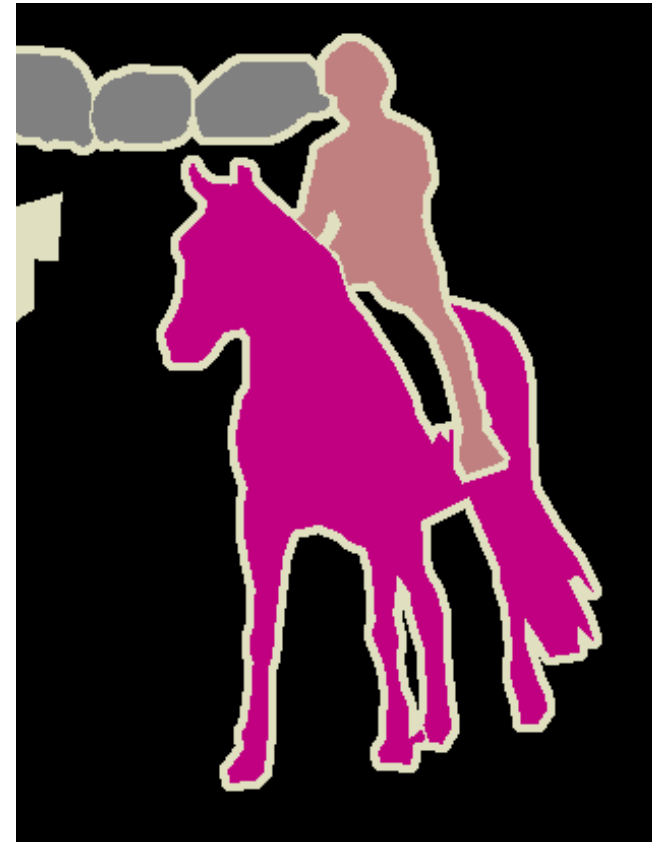
Input image



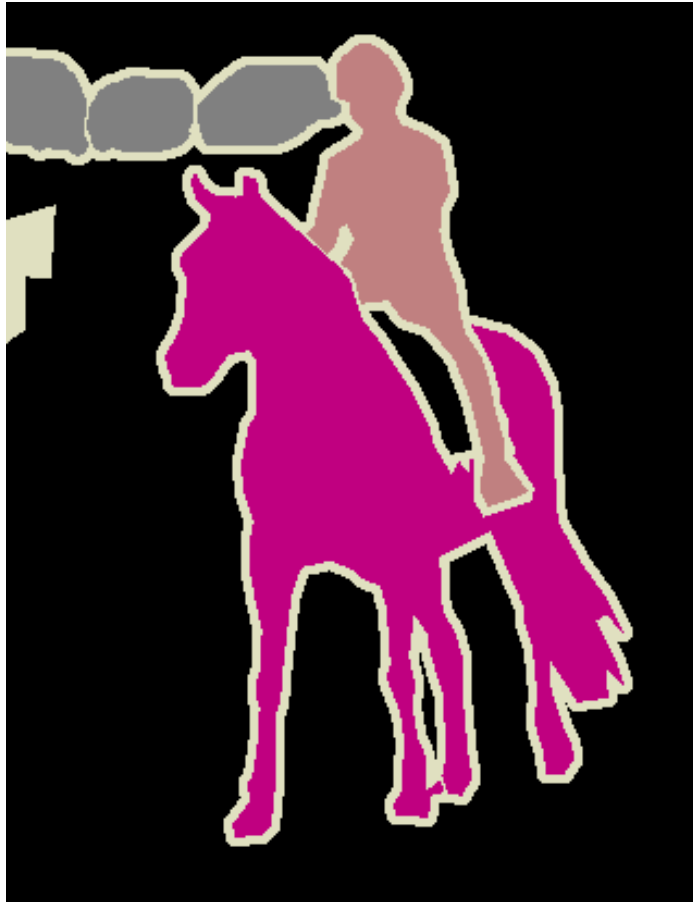
CNN



Desired pixel labels



Reading a label mask

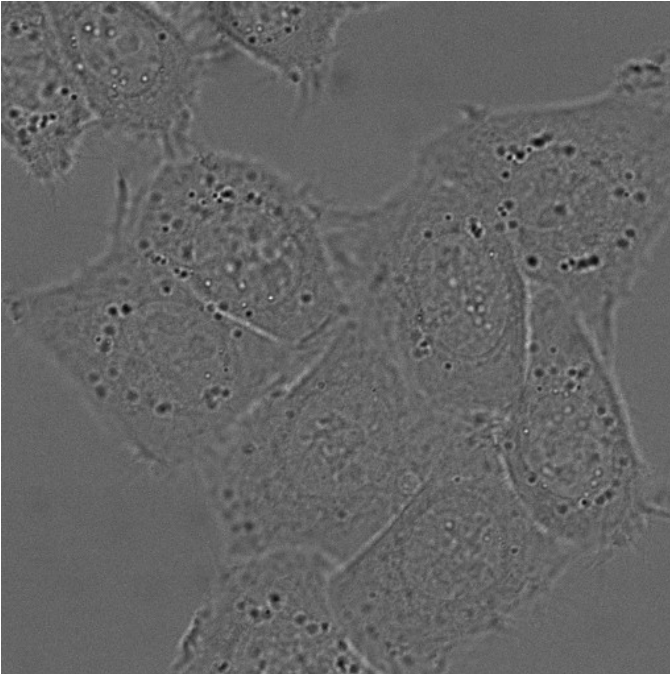


-  Horse
-  Person
-  Car
-  Background
-  Ignored pixels

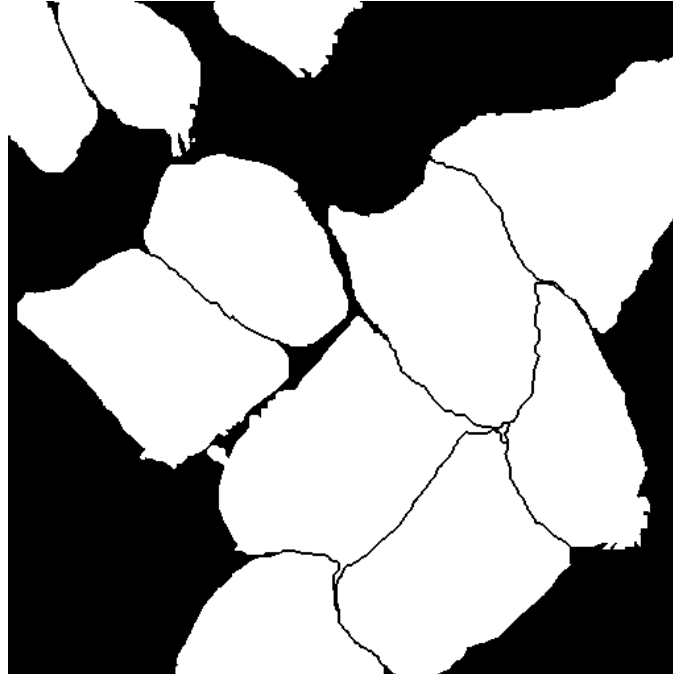
One class ID per pixel

Semantic and instance masks

Image

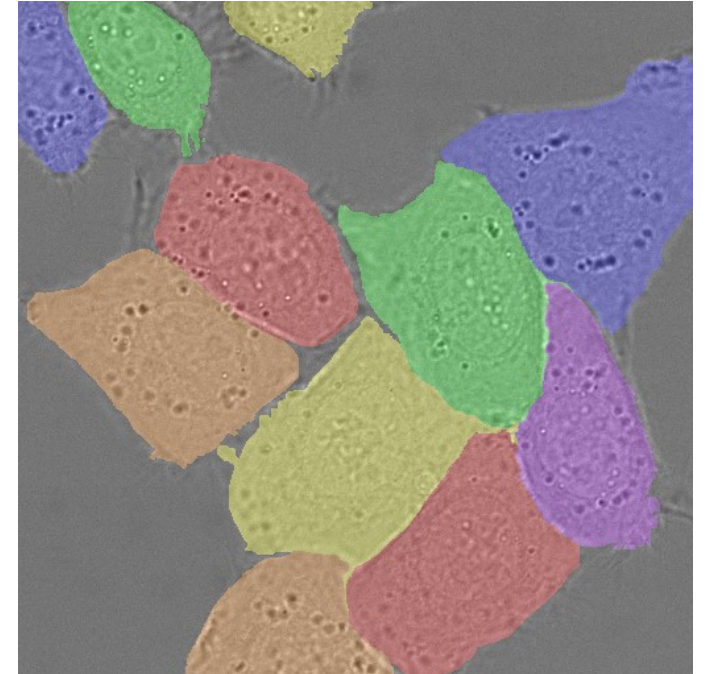


Cell / background



Semantic labels

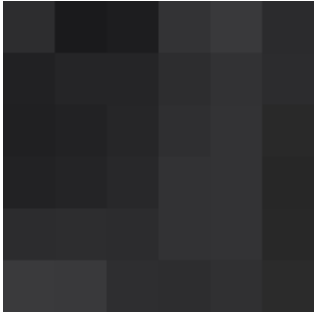
Individual cells



Instance IDs

Pixel labels need context

Small patch



Larger neighborhood



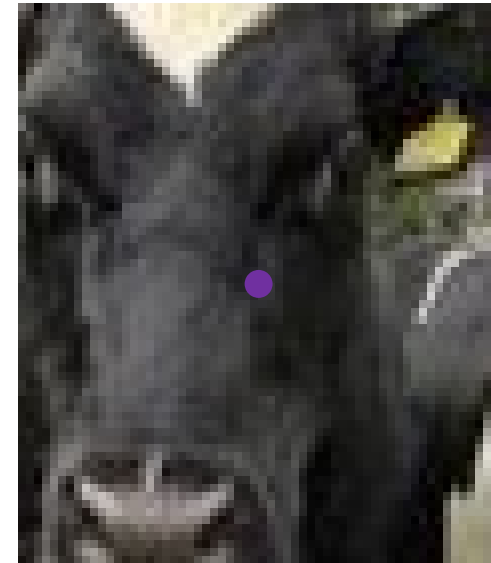
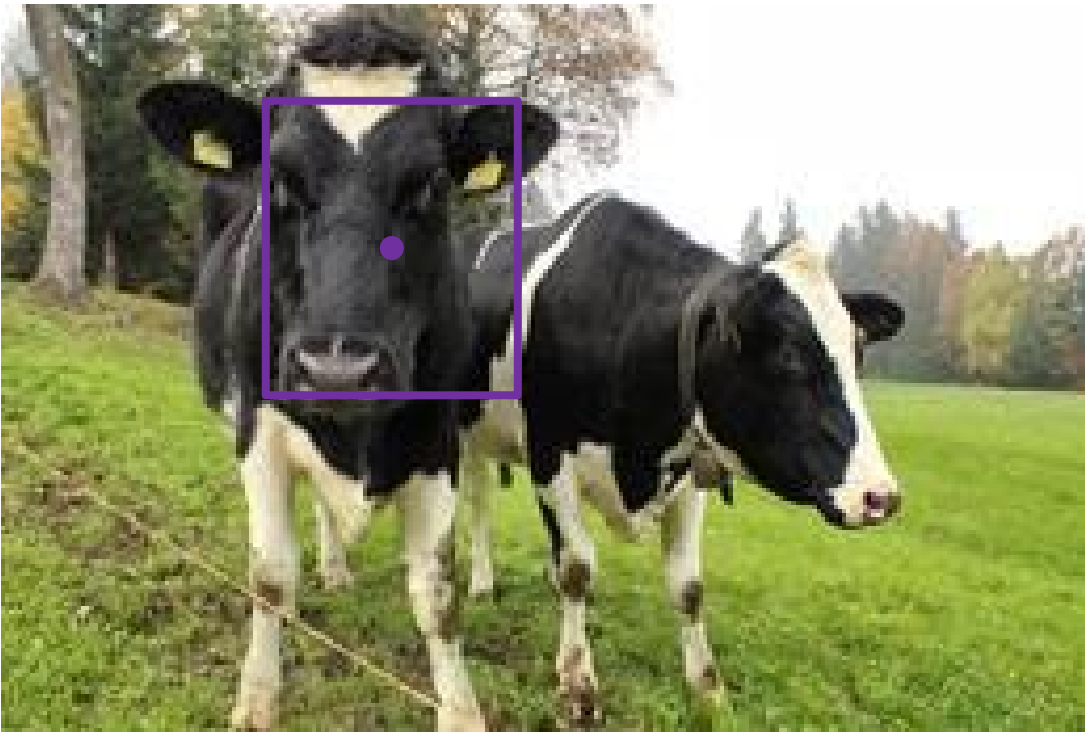
Surrounding scene



Same target pixel

Classifying image patches

Move the window, reuse the same CNN.

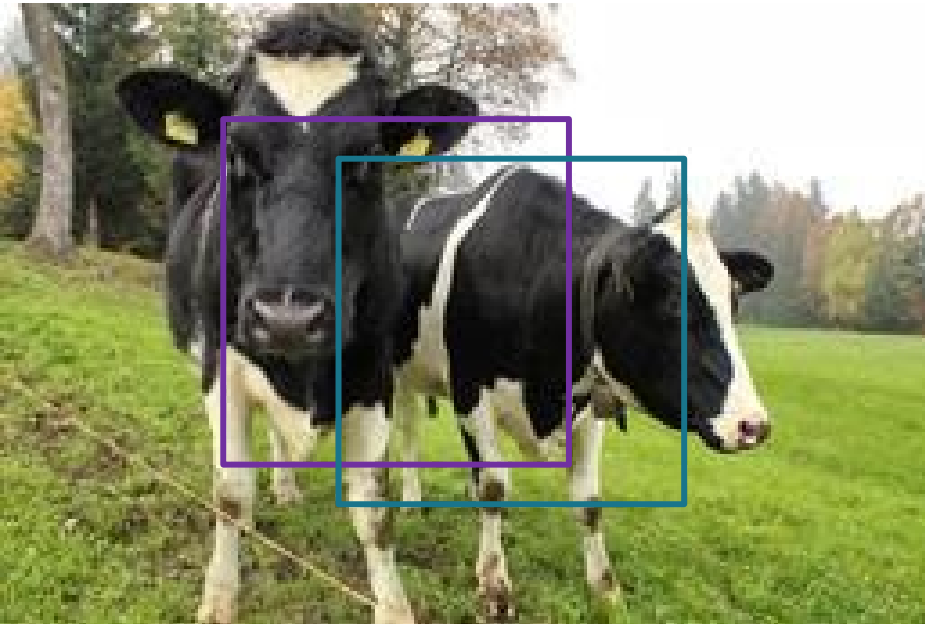


Input: a neighborhood

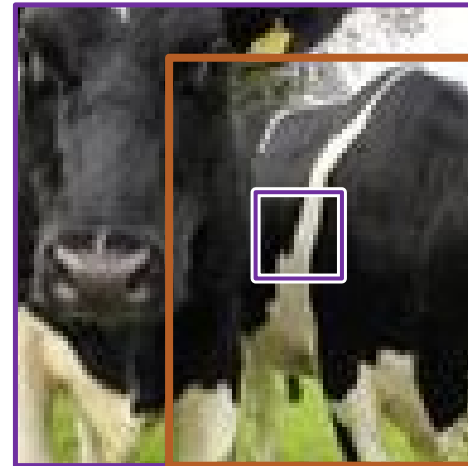
Target: the center pixel's class

Overlapping patches repeat computation

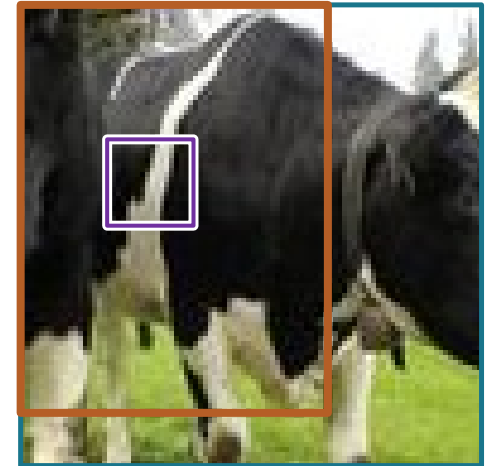
Overlapping input windows



Patch 1



Patch 2

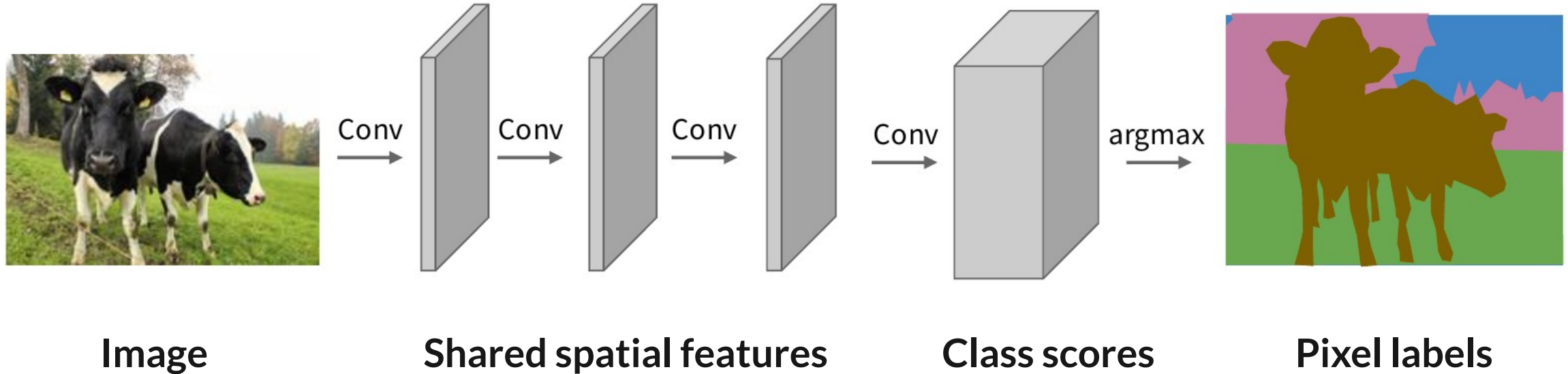


Same image region in both crops

Each crop runs the convolutional layers again.

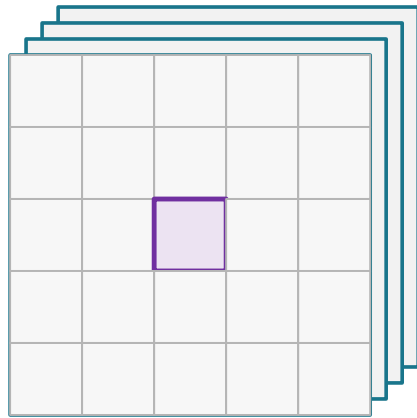
Shared convolutional features

Compute features once for the whole image.



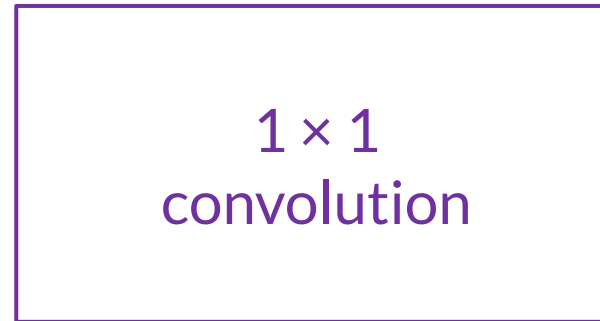
Class scores at each location

Learned feature maps

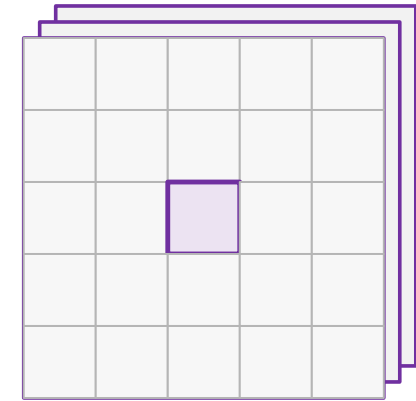


C channels

$h \times w$ locations



One filter per category



K class-score channels

Same $h \times w$ grid

Features and class scores

At one location (i, j)

Feature vector

$$\mathbf{z}_{ij} \in \mathbb{R}^C$$

Feature values



$$\mathbf{s}_{ij} = W\mathbf{z}_{ij} + \mathbf{b}$$

Same W and b at every location



Class scores



Horse



Person



Background



...

From scores to pixel labels

Compare categories at one location

$$p_{ij,k} = \frac{\exp(s_{ij,k})}{\sum_{c=1}^K \exp(s_{ij,c})}$$

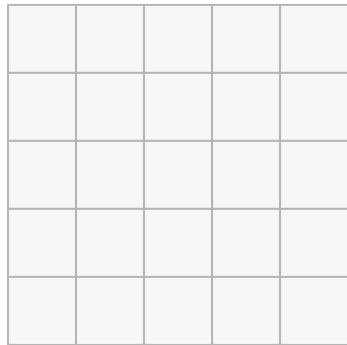
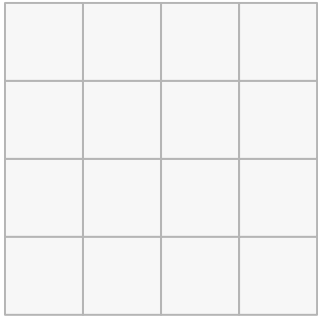
Softmax across the K classes

$$\hat{y}_{ij} = \arg \max_k s_{ij,k}$$

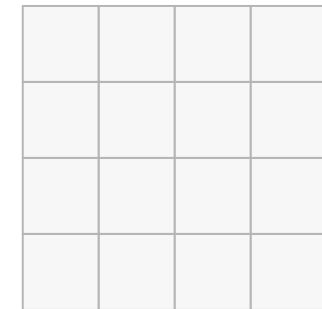
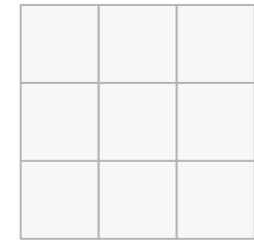
One category ID at this location

Fully convolutional networks (FCNs)

Different input sizes



Corresponding output grids



From a linear layer to a convolution

One feature channel and one output shown

Linear layer

Flattened features

x_1	x_2	x_3	x_4
-------	-------	-------	-------

Linear weights

w_1	w_2	w_3	w_4
-------	-------	-------	-------

Convolution

Feature grid

x_1	x_2
x_3	x_4

Kernel weights

w_1	w_2
w_3	w_4

Match positions across the full grid

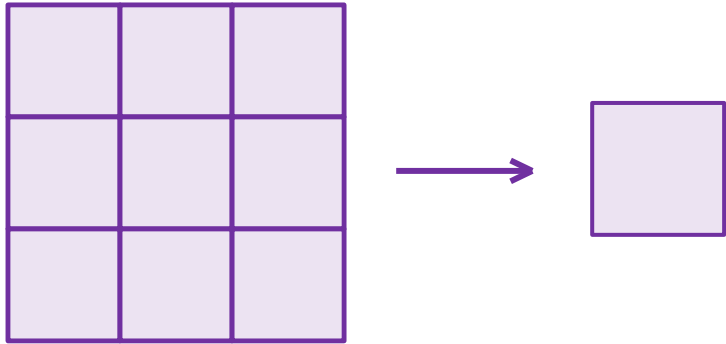
Both compute

$$y = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$

For C channels, each kernel spans all $h \times w \times C$ input values.

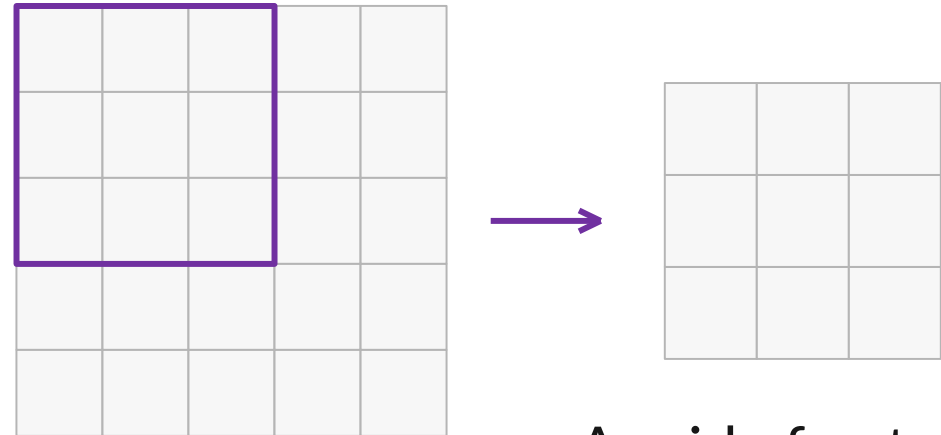
Applying the classifier across an image

Original feature grid



One output location

Larger feature grid



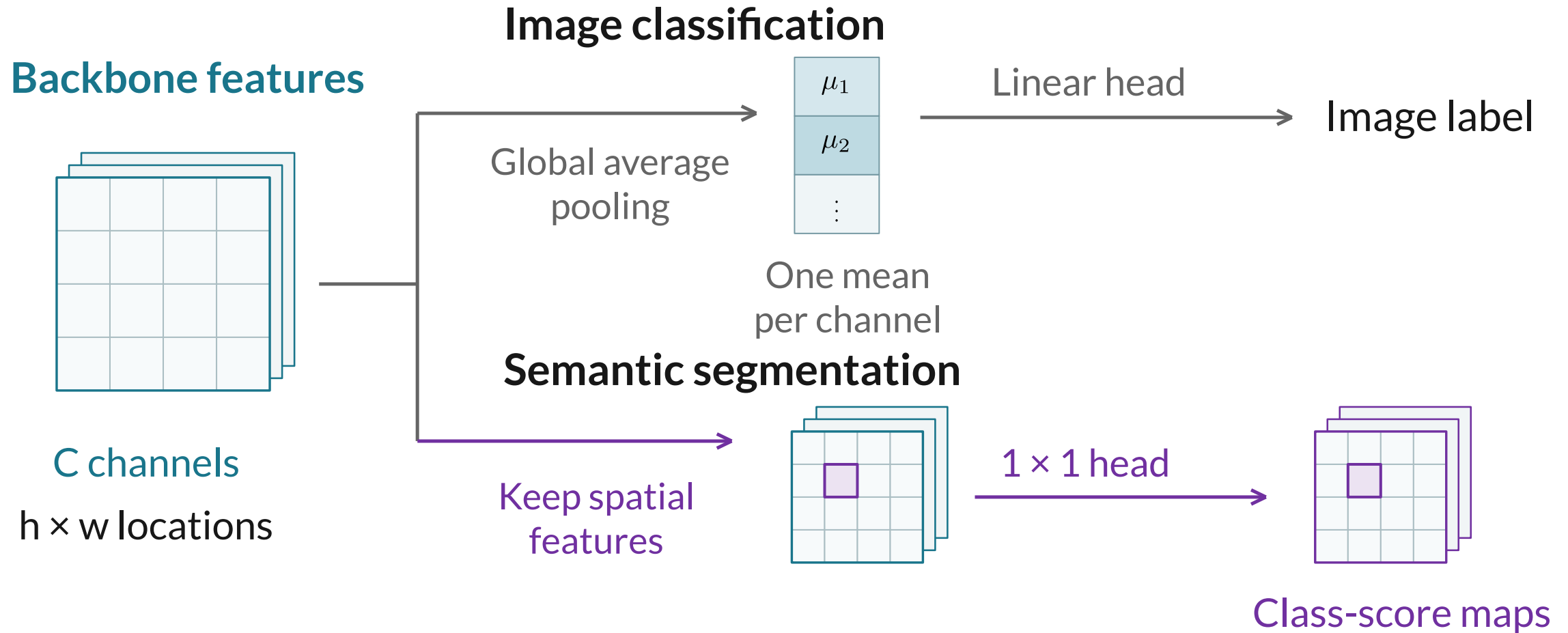
A grid of outputs

Same 3×3 kernel

Later 1×1 layers preserve this grid

Backbones with global pooling

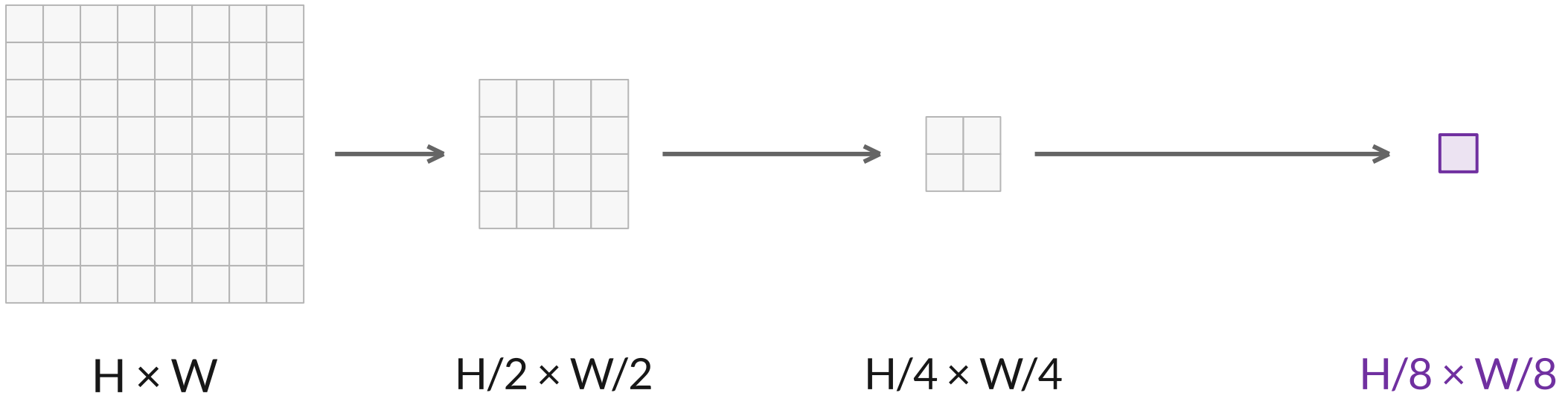
Take features before global pooling



Output stride

Three stride-2 reductions

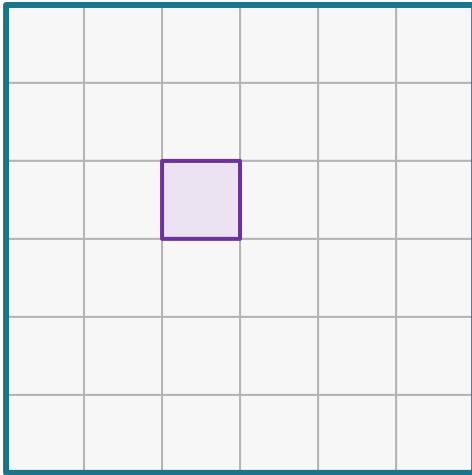
Output stride: spacing measured in input-image pixels



$$\text{Output stride} = 2 \times 2 \times 2 = 8$$

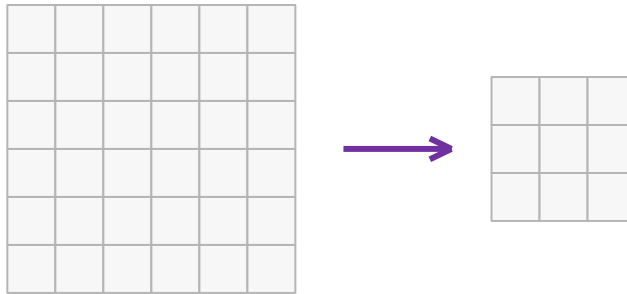
Context, computation and resolution

Context



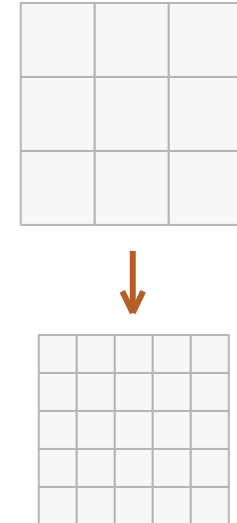
A prediction needs
its surroundings

Computation



Smaller maps make
later layers cheaper

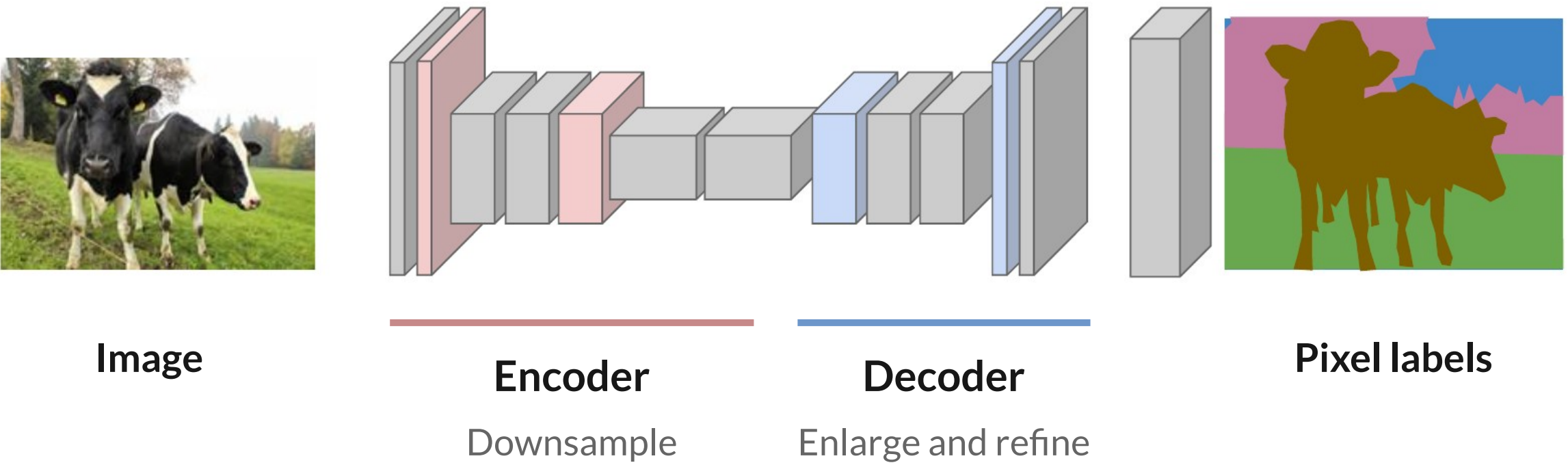
Dense output



Finer decisions
need more locations

Encoder and decoder

Use a coarse grid for context, then build finer pixel predictions.



Lecture outline

01 Pixel prediction

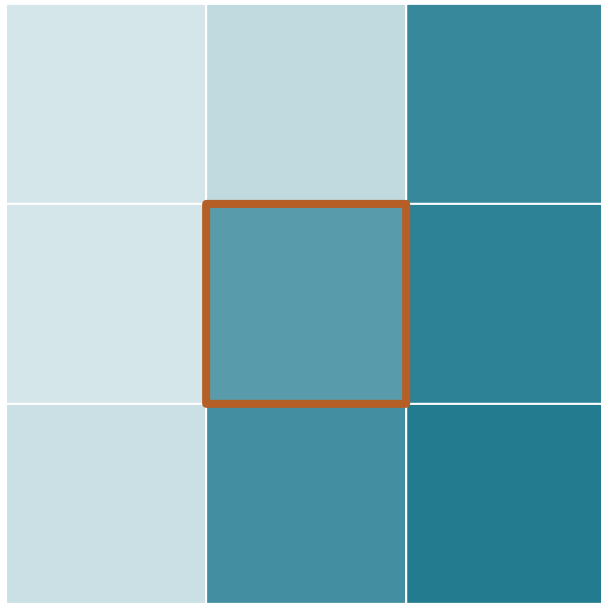
02 Upsampling and decoders

03 Training and evaluation

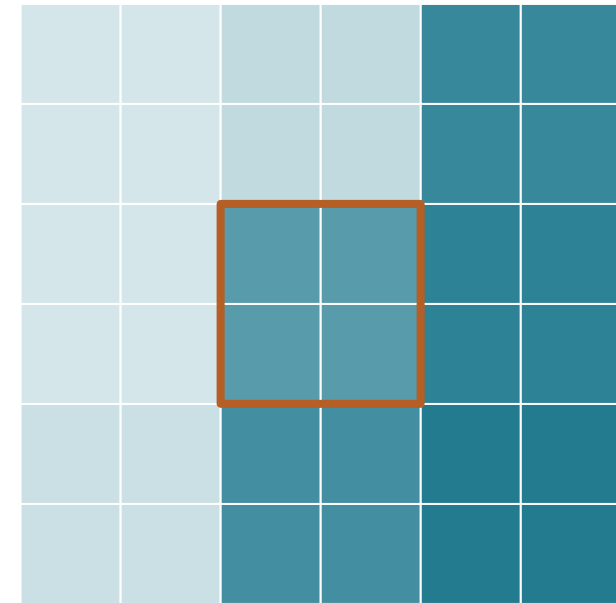
04 Spatial context

Nearest-neighbor upsampling

Coarse map • 3×3



Enlarged map • 6×6



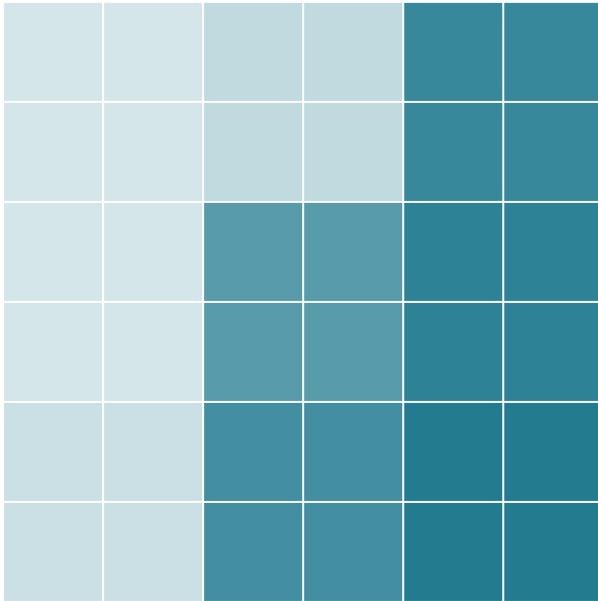
Copy the nearest
coarse value

Darker blue = larger feature value

No learned weights

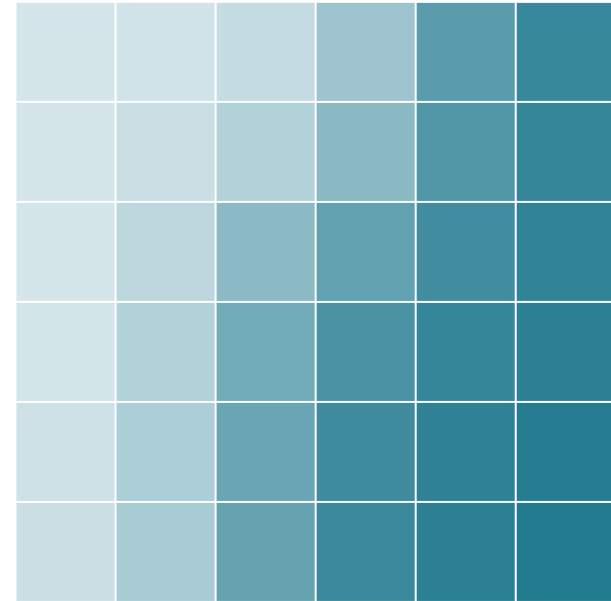
Bilinear interpolation

Nearest neighbor



Copies one nearby sample

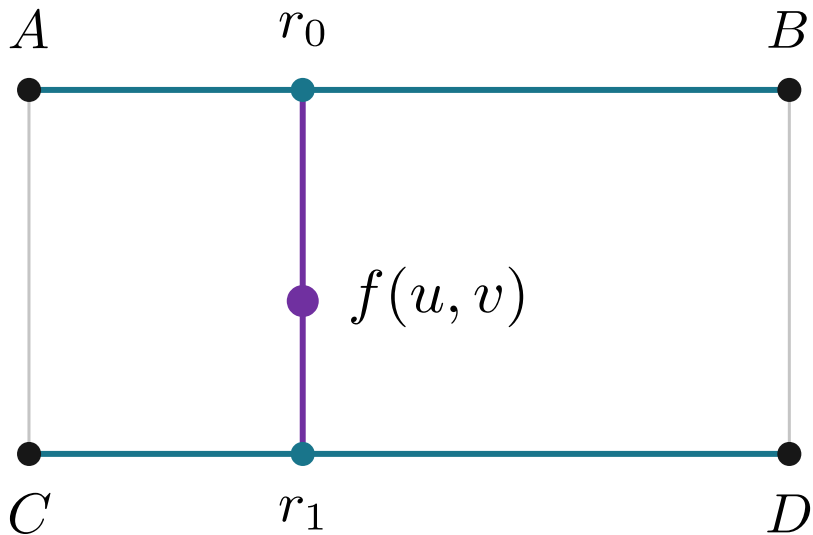
Bilinear



Average of four neighbors,
weighted by distance

Bilinear interpolation

A, B, C, D: four neighboring feature values



u : fraction from left to right

v : fraction from top to bottom

$$0 \leq u, v \leq 1$$

1. Interpolate along each row

$$r_0 = (1 - u)A + uB$$

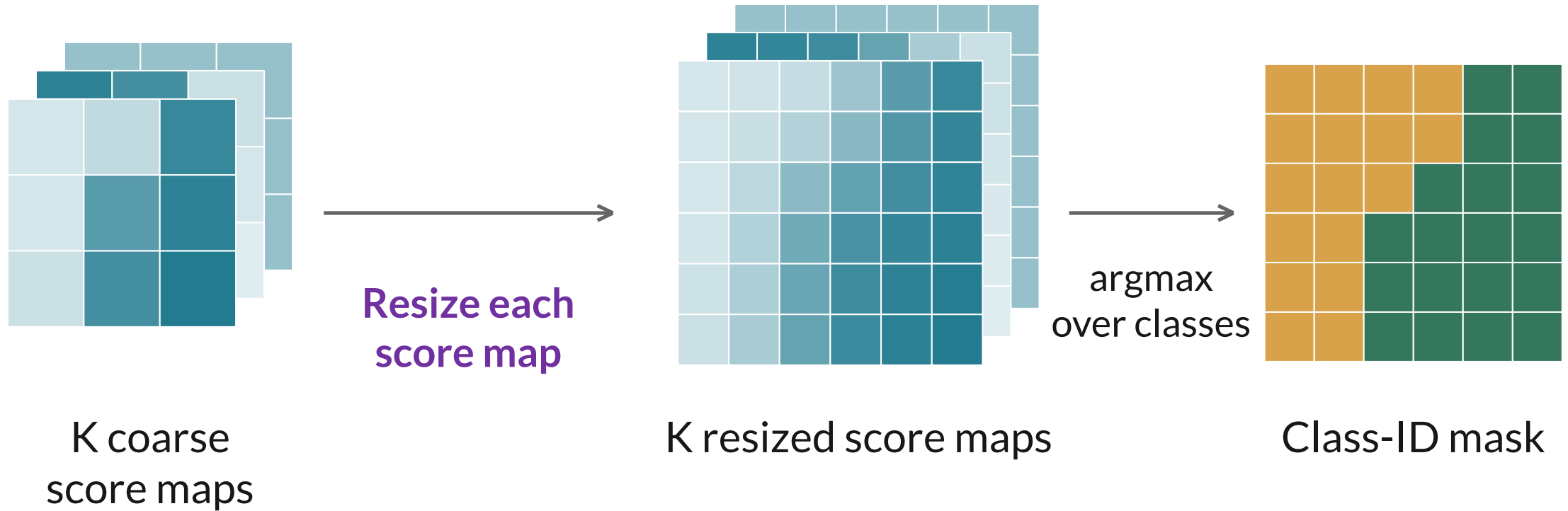
$$r_1 = (1 - u)C + uD$$

2. Interpolate between the rows

$$f(u, v) = (1 - v)r_0 + vr_1$$

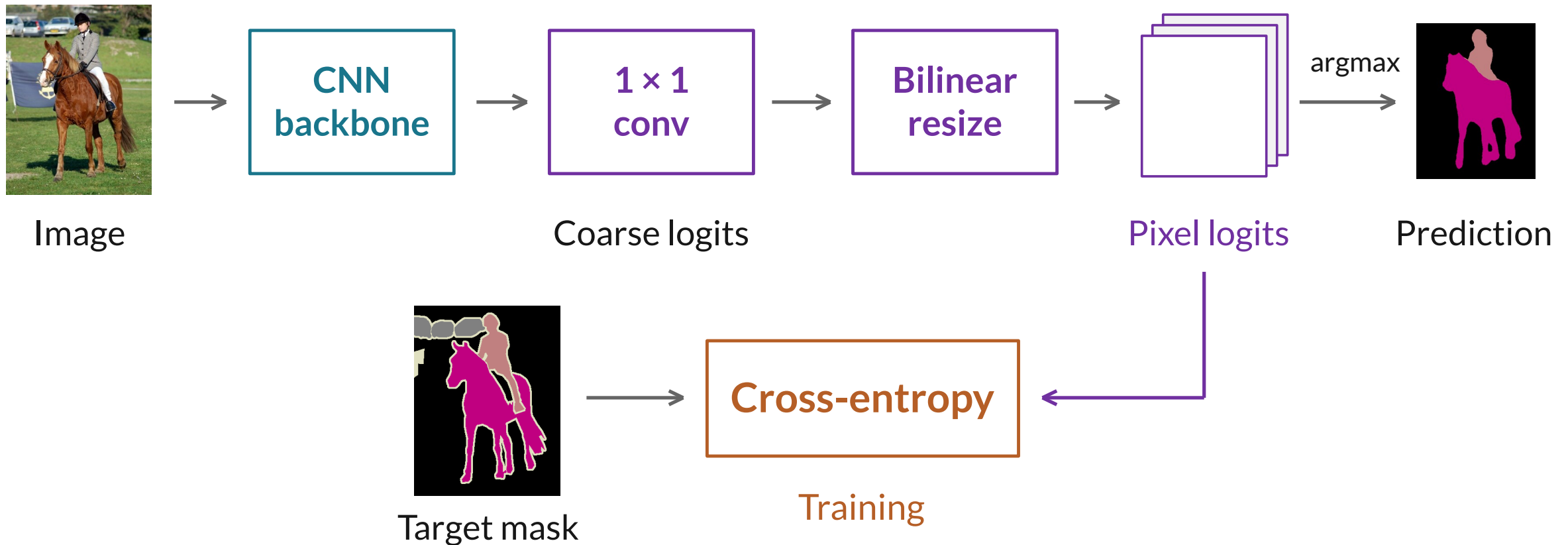
$$\begin{aligned} f(u, v) = & (1 - u)(1 - v)A + u(1 - v)B \\ & + (1 - u)vC + uvD \end{aligned}$$

Upsampling class scores



FCN with bilinear upsampling

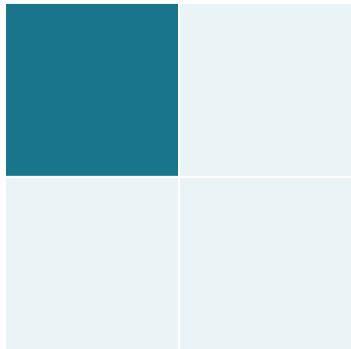
Baseline: fixed bilinear resizing of the class scores



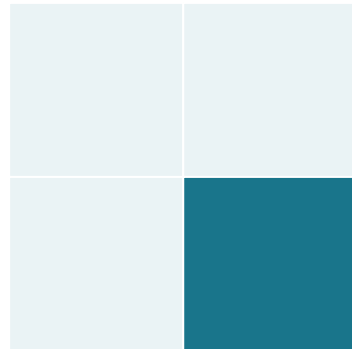
Information lost during downsampling

Different fine-scale inputs

Strong response 1 · other cells 0



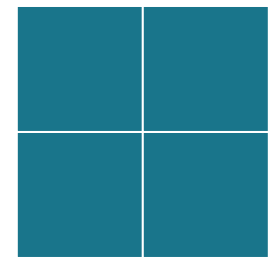
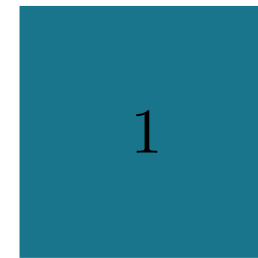
Upper-left



Lower-right



2×2
max pooling



Nearest resize

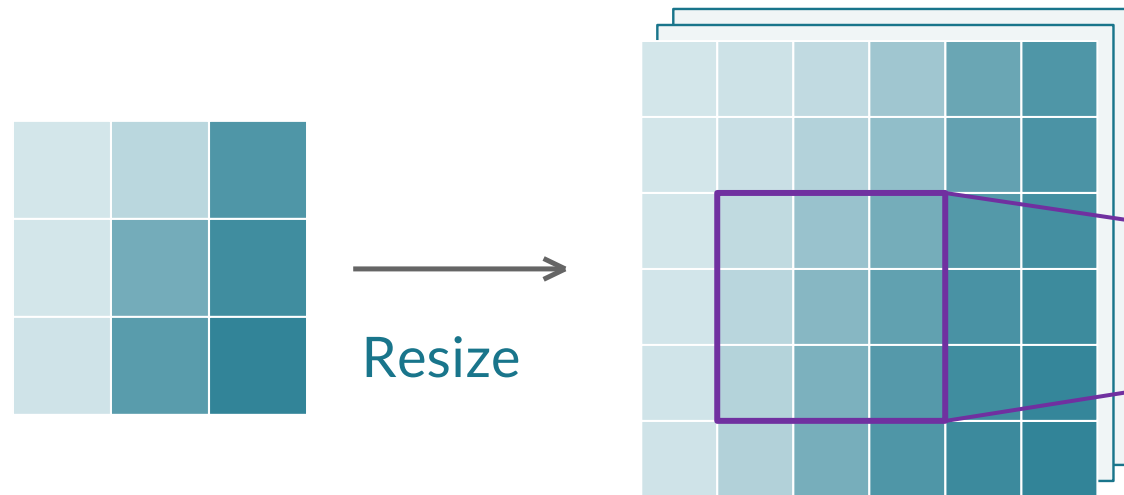
Same coarse value

After pooling, the original location is no longer available

Resize followed by convolution

Refine the features before predicting the classes

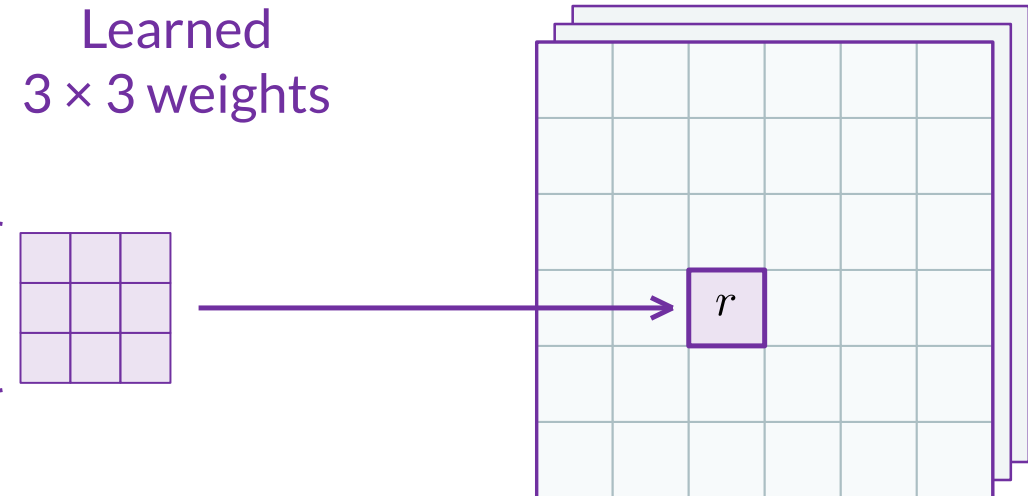
Fixed enlargement



Coarse features

Enlarged features

Learned refinement

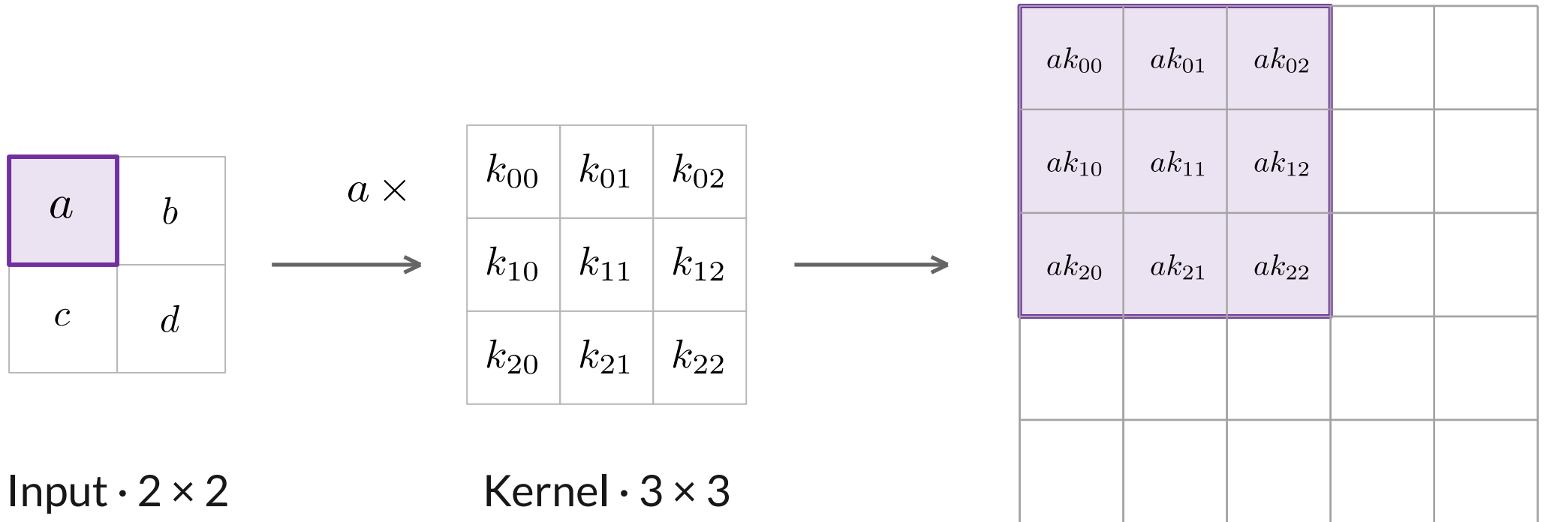


One output feature

The class head then maps refined features to class scores.

Transposed convolution

Learn an upsampling kernel for features or class scores



Input $\cdot 2 \times 2$

Kernel $\cdot 3 \times 3$

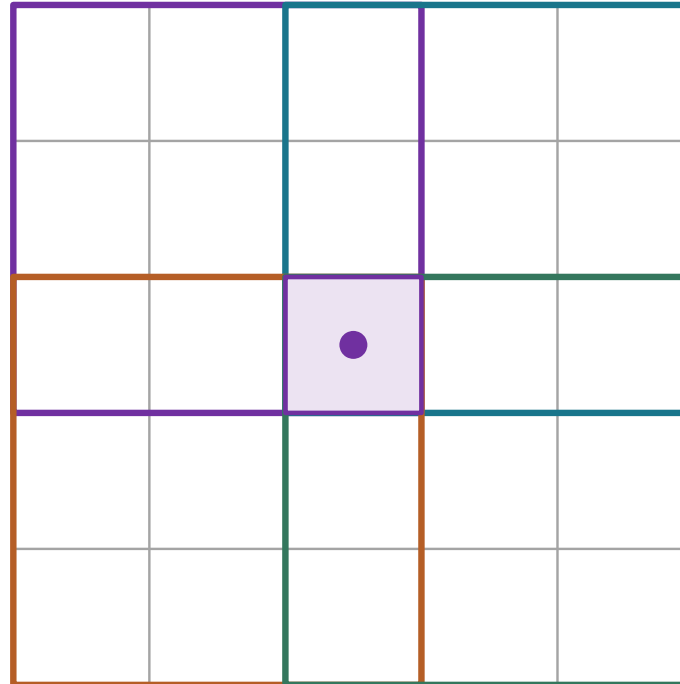
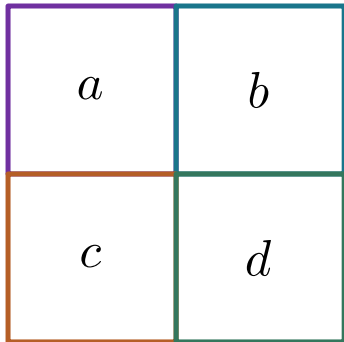
Subscripts: kernel row, column

Stride 2 $\cdot$ no padding

Contribution to a 5×5 output

Kernel placement and overlap

Input



$a \times \text{kernel}$

$b \times \text{kernel}$

$c \times \text{kernel}$

$d \times \text{kernel}$

Stride 2

Add contributions where the kernel placements overlap

Highlighted cell: $ak_{22} + bk_{20} + ck_{02} + dk_{00}$

Convolution and its transpose

Convolution · 5 values → 2

$$M = \begin{bmatrix} k_0 & k_1 & k_2 & 0 & 0 \\ 0 & 0 & k_0 & k_1 & k_2 \end{bmatrix}$$

Each row gathers a neighborhood

1D example · 3 kernel weights · stride 2

Transpose · 2 values → 5

$$M^T = \begin{bmatrix} k_0 & 0 \\ k_1 & 0 \\ k_2 & k_0 \\ 0 & k_1 \\ 0 & k_2 \end{bmatrix}$$

Each column spreads a contribution

Transposed convolution: output size

One spatial axis

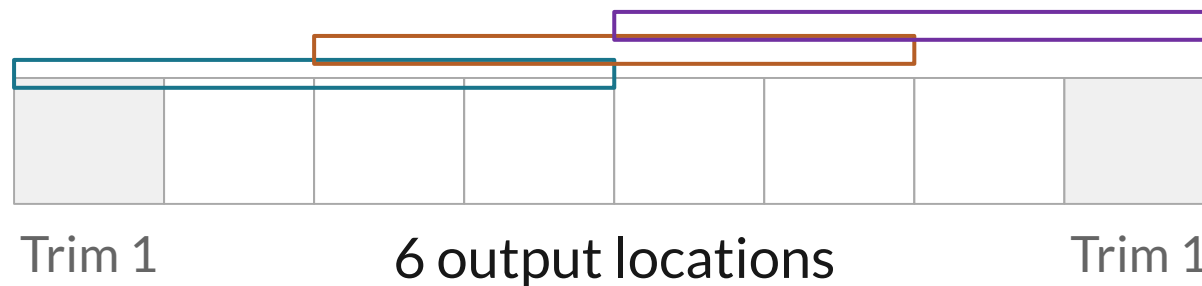
Adjacent kernel entries, no extra output padding

$$n_{\text{out}} = (n_{\text{in}} - 1)s - 2p + k$$

n input / output size s stride p padding k kernel size

Kernel 4 • stride 2 • padding 1

$$(3 - 1) \times 2 - 2 \times 1 + 4 = 6$$

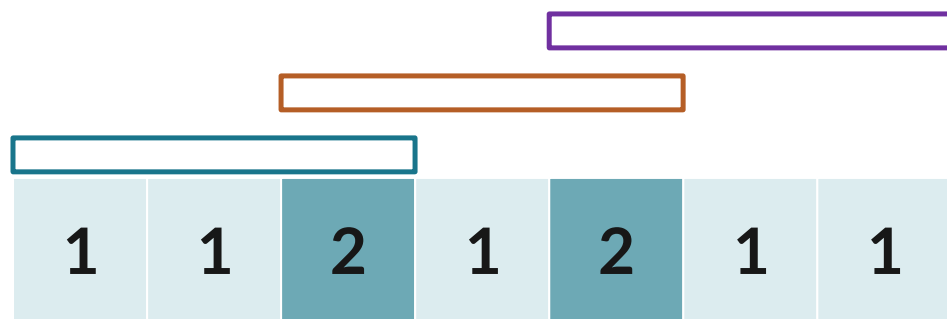


3 × 3 input
6 × 6 output

Uneven overlap

Kernel 3 • stride 2

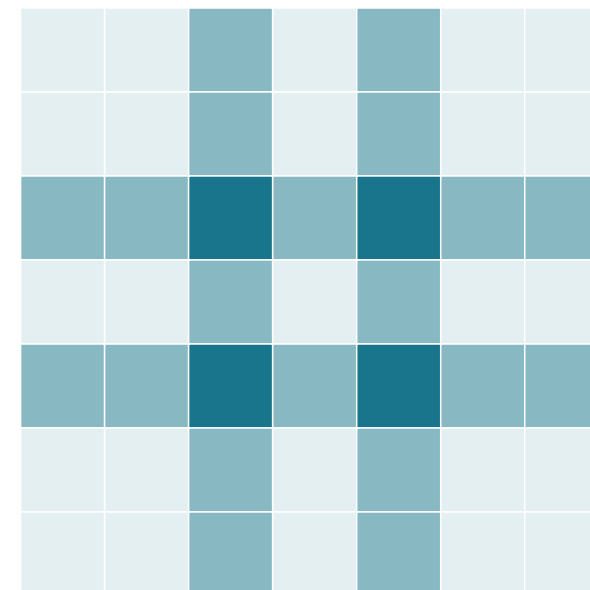
Number of overlapping contributions



Along one axis



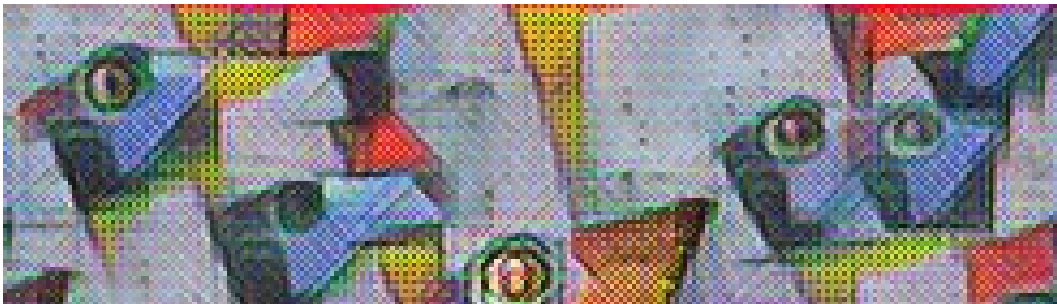
More overlap can amplify a repeated pattern



Two axes combine

Checkerboard artifacts

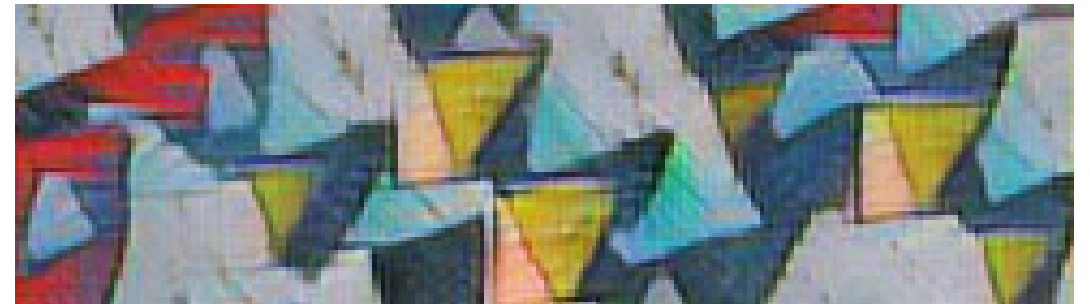
Transposed convolution



Style-transfer outputs

Repeated grid pattern

Resize + convolution



No repeated grid pattern

Remembering pooling locations

Encoder features

1	6	2	3
4	5	8	1
7	2	1	9
3	4	5	6

2×2 max pooling

Pooled values

6	8
7	9

Keep the maximum

Pooling indices

	●		
		●	
●			●

Save selected positions

and

Max unpooling

Current decoder values

a	b
c	d



Saved locations

	●
●	

●	
	●



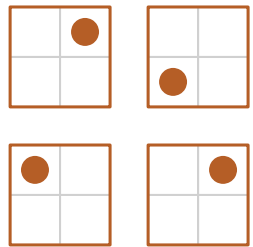
Unpooled map

0	a	0	0
0	0	b	0
c	0	0	d
0	0	0	0

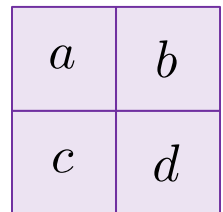
Place each value at its saved position

Unpooling followed by convolution

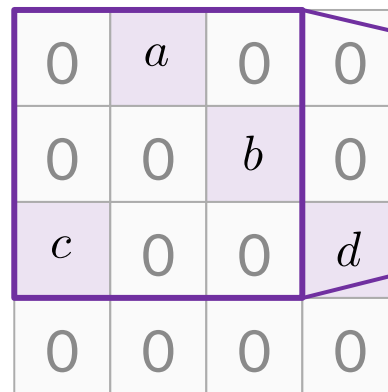
Saved pooling positions



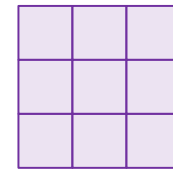
Decoder values



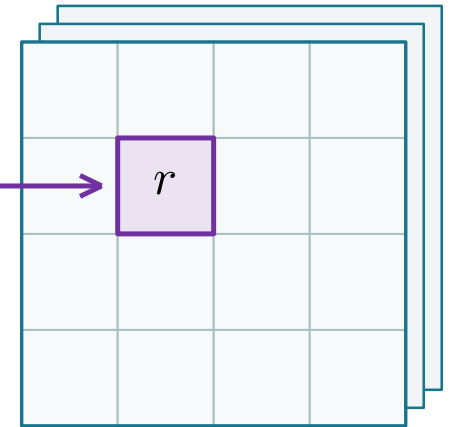
Sparse map



Convolution BatchNorm + ReLU



Refined features



Max unpooling

Learned weights mix nearby values and channels.

Upsampling choices

Resize + convolution

Enlarge with a fixed rule, then learn local refinement.

Transposed convolution

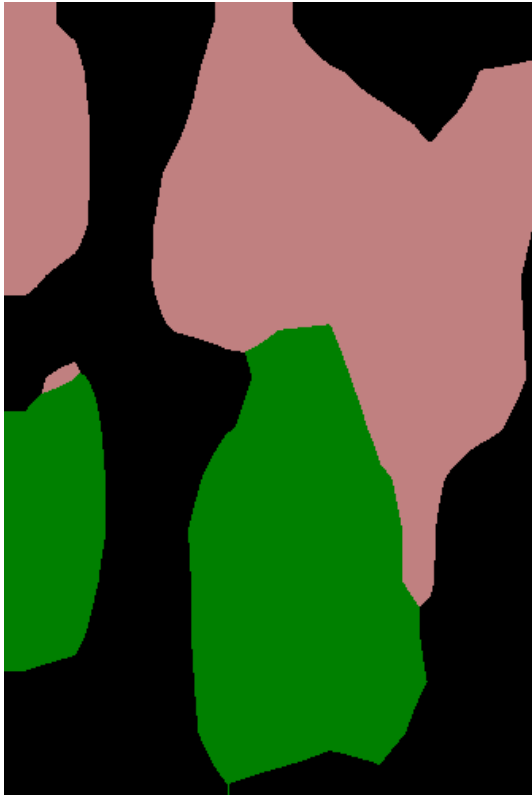
Learn the kernel contributions placed on a finer grid.

Max unpooling + convolution

Use saved pooling positions, then learn local refinement.

Recovering boundary detail

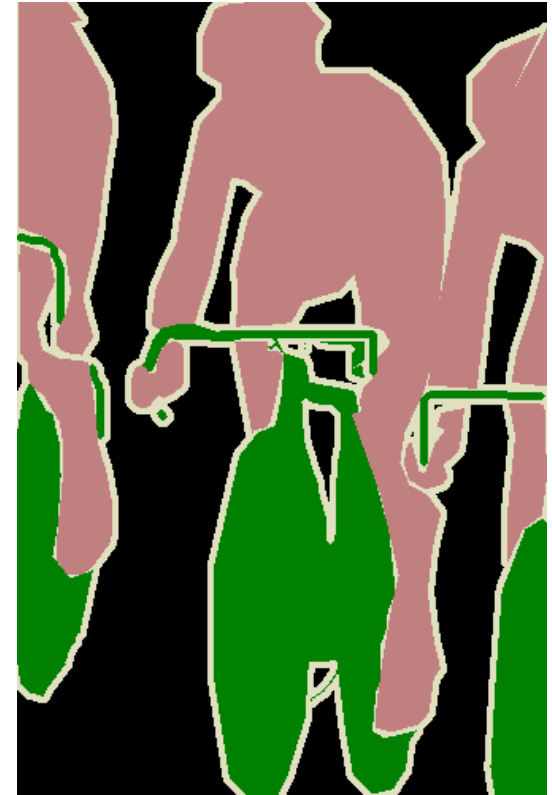
Coarse prediction, enlarged



Missing boundary detail
needs finer spatial
evidence

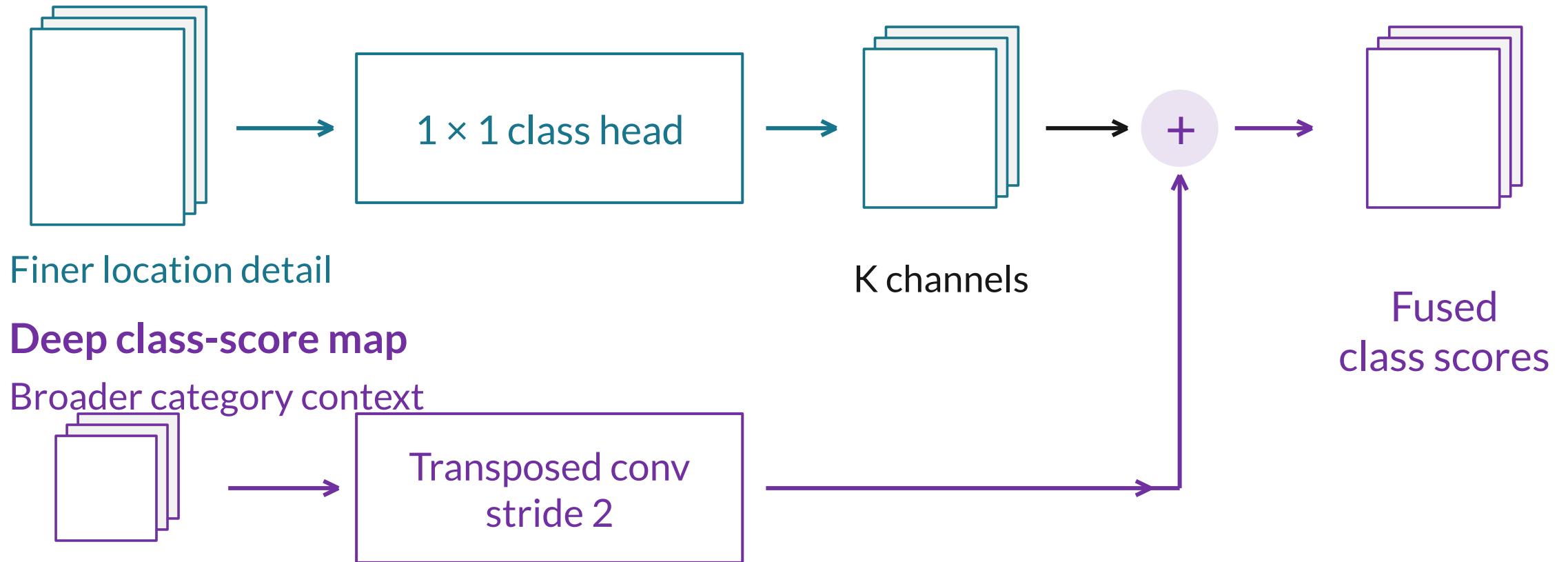
- Person
- Bicycle

Ground truth



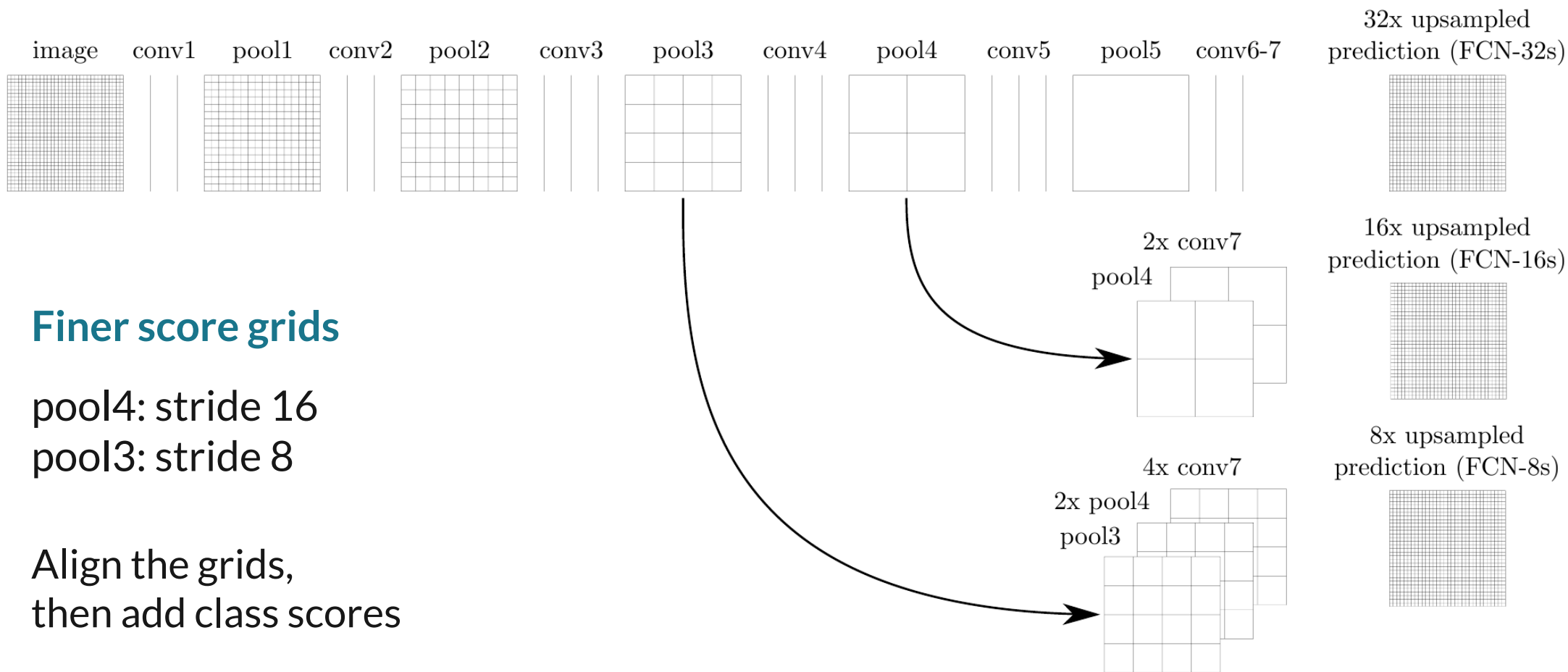
FCN skip connections

Earlier feature map



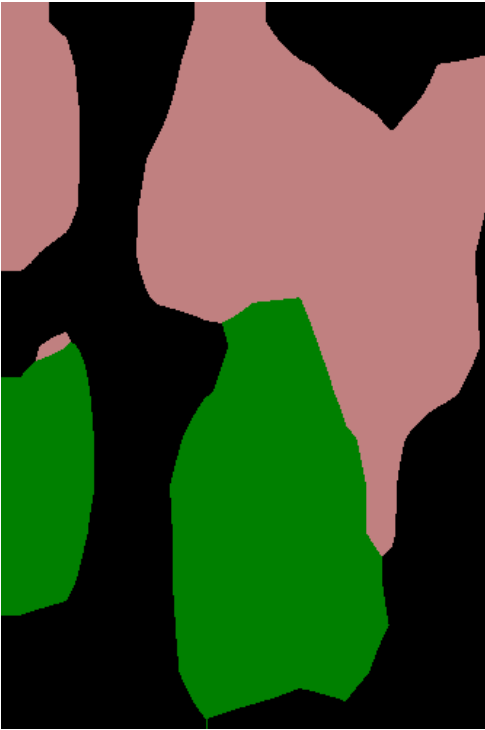
Match spatial positions and class channels

FCN-32s, FCN-16s and FCN-8s



Refining FCN predictions

FCN-32s



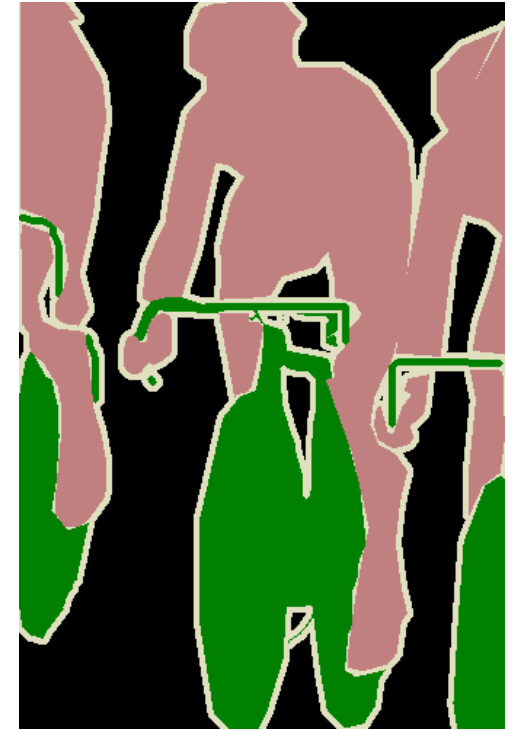
FCN-16s



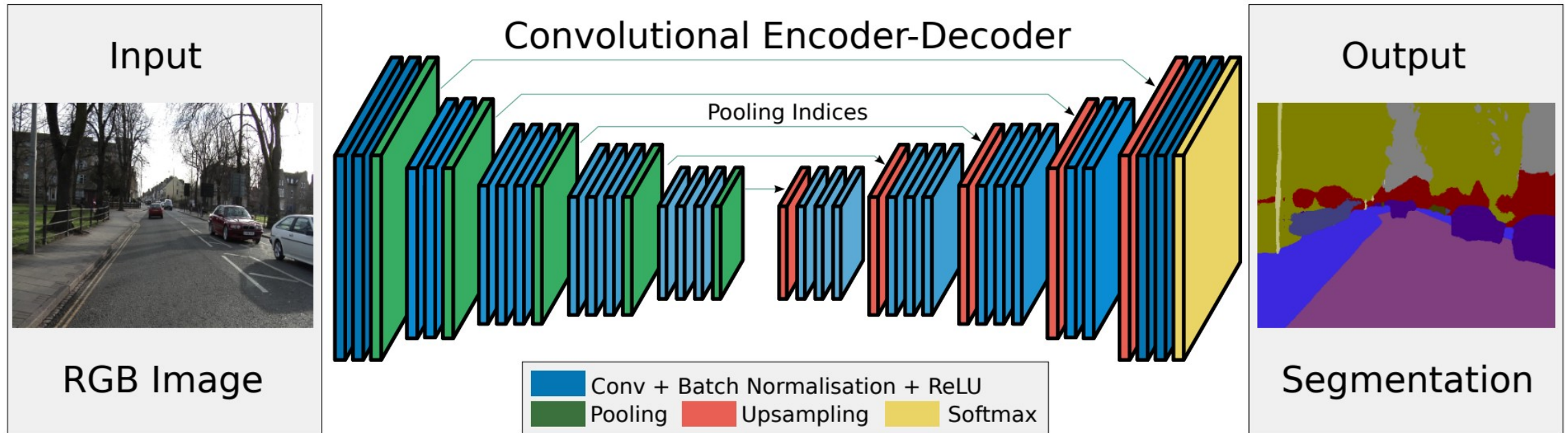
FCN-8s



Ground truth



SegNet, 2015



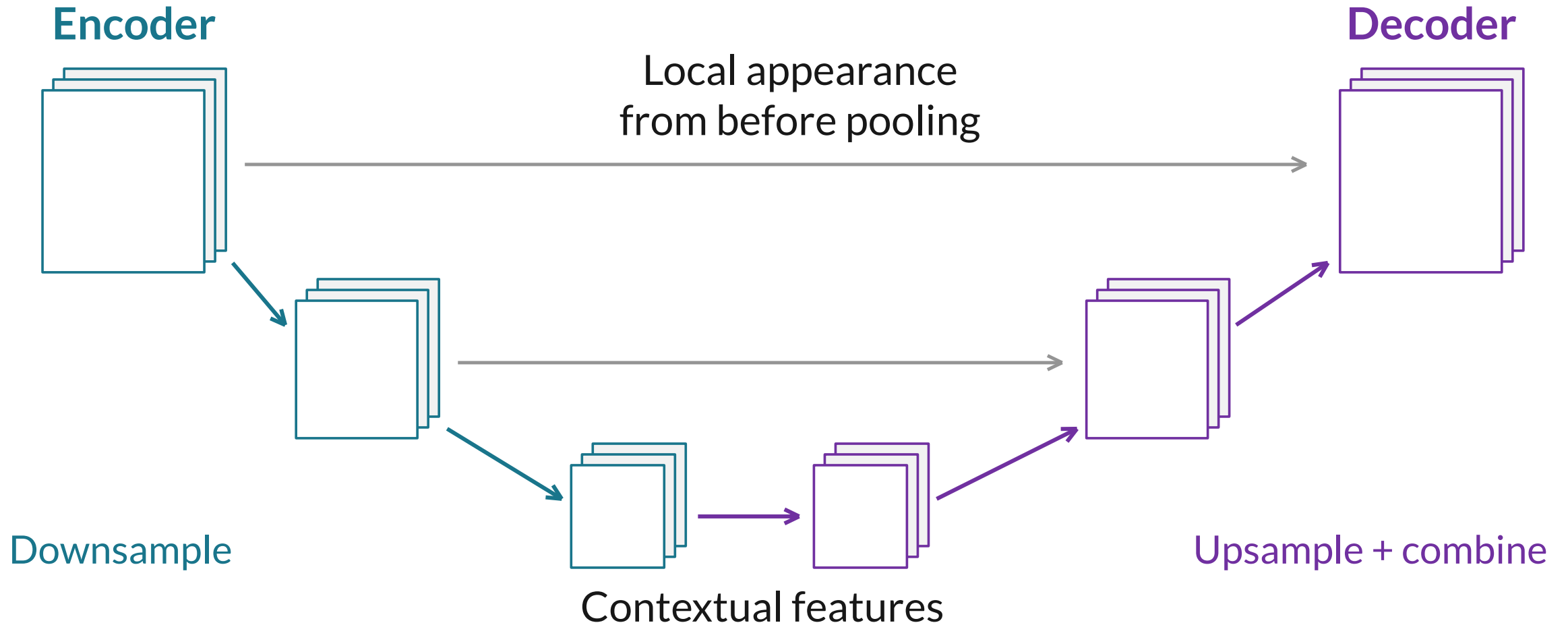
Location records

Compact information for upsampling

Earlier feature values

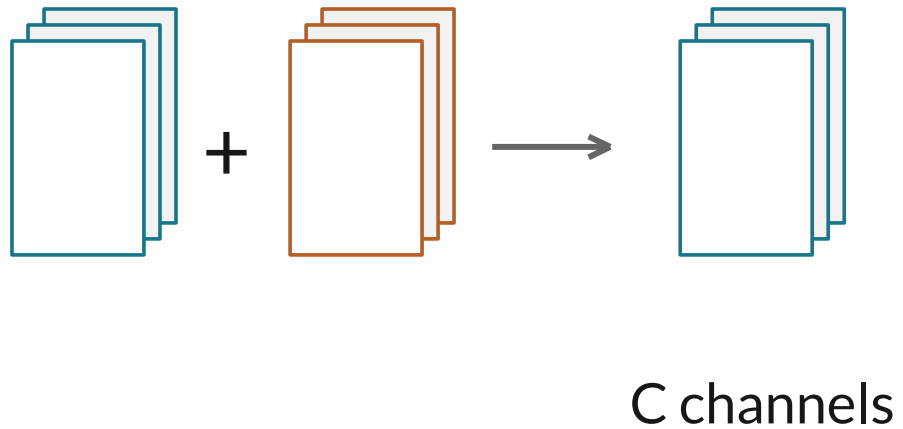
Not passed through these skips

U-Net: an encoder and a decoder



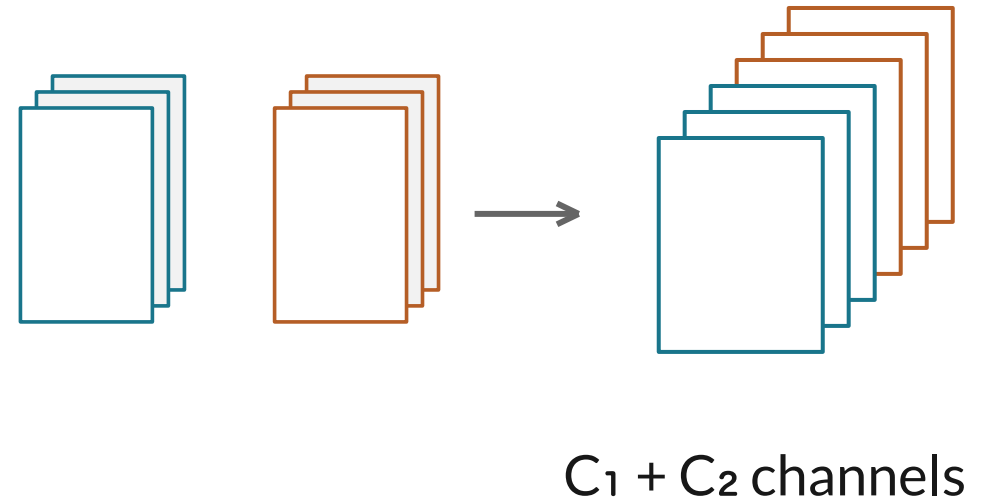
Concatenation and addition

Addition



Sum corresponding values

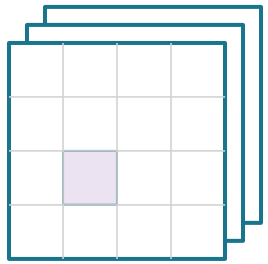
Concatenation



Keep both sets of features

One U-Net decoder stage

Encoder features

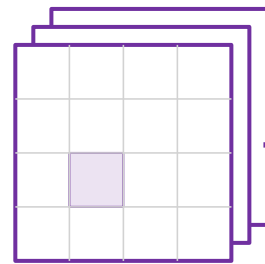


Finer spatial grid

Coarse decoder features

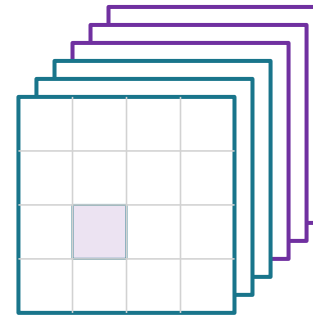


Transposed conv
 2×2 , stride 2

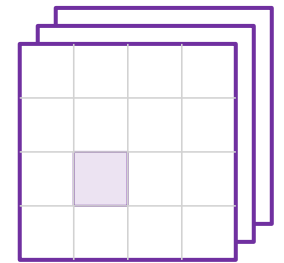


Same spatial grid

Concatenate channels



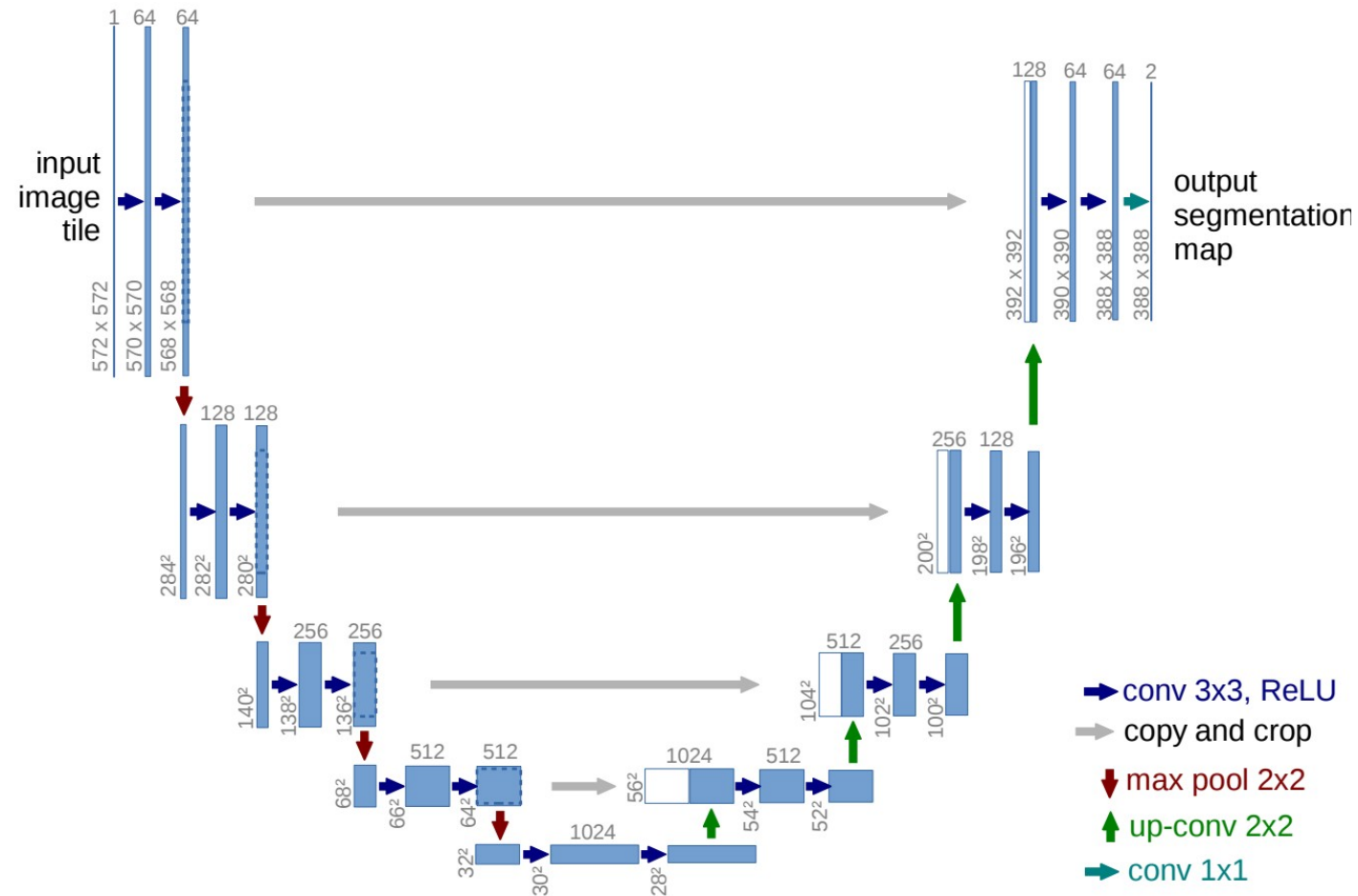
Convolutions
+ ReLU



Refined features

Keep both feature vectors
at each matching location

U-Net, 2015



Convolutional features

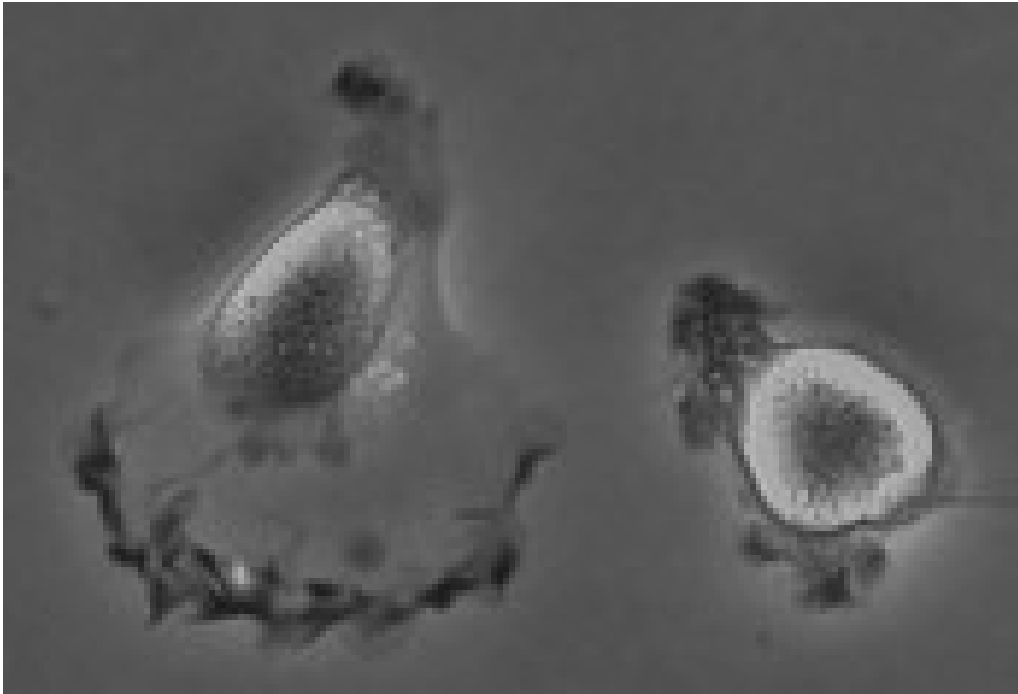
Gray arrows copy
encoder features

Green arrows:
transposed convolution

Original design:
no padding
cropped skip features

Cell segmentation

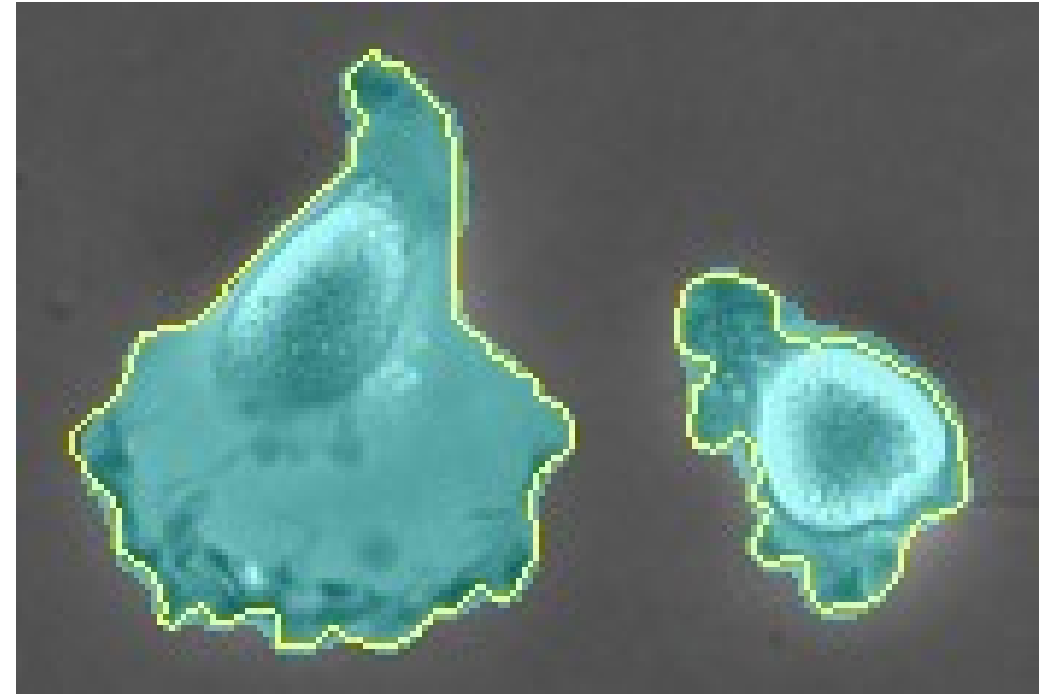
Microscopy input



Segmentation result

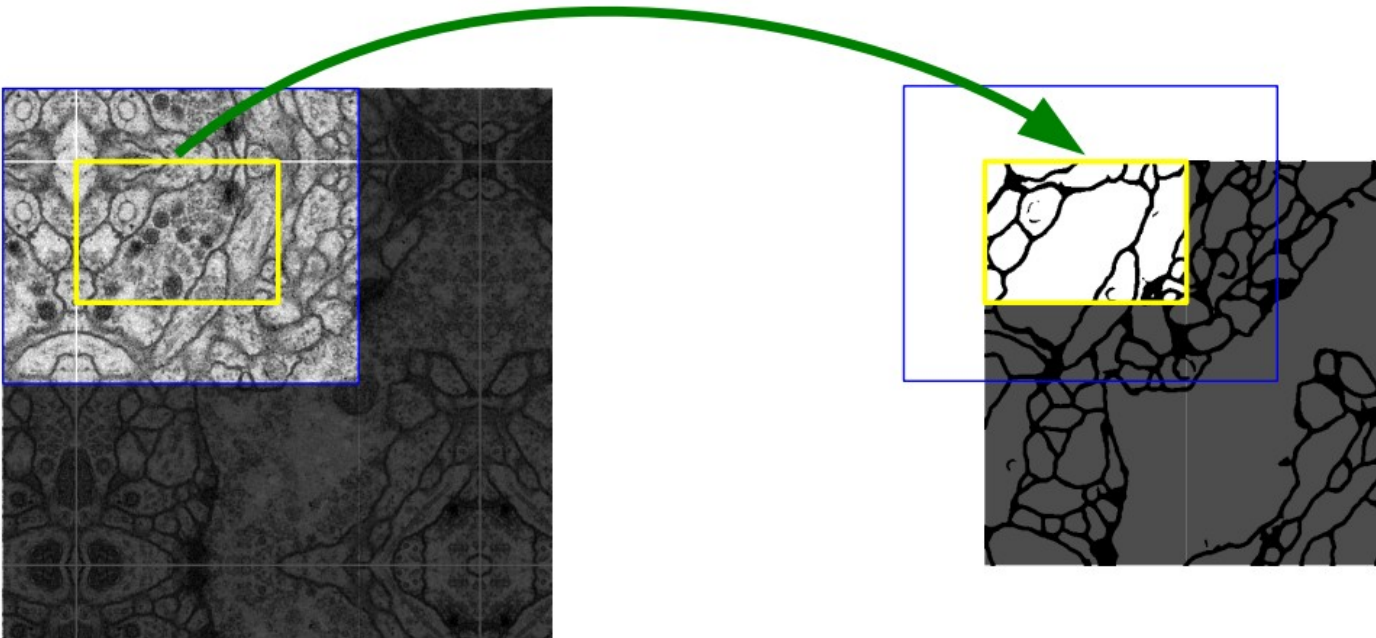
■ Prediction

■ Annotation



Predicting large images in tiles

Process a large image in tiles that fit in memory



Input tile

Border pixels supply context for the center

Retained output

Keep the center, then join adjacent outputs

Human pose estimation



Predicted joint locations

Input

Image crop around one person

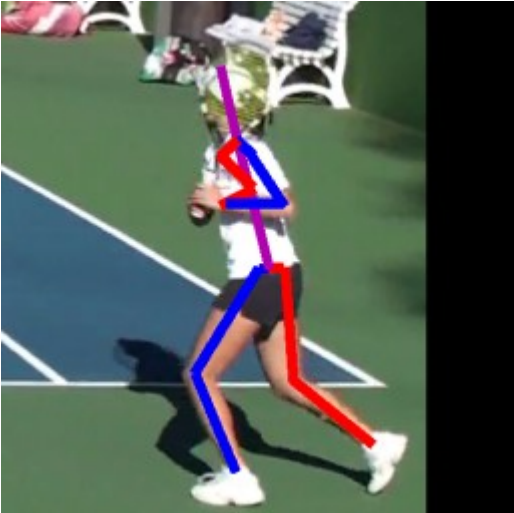
Output

Locations of joints:
elbows, wrists,
knees, ankles...

Encoder-decoders can also predict joint locations (keypoints).

One heatmap per joint

Predicted pose



Neck



Left elbow



Left wrist



Bright area: higher predicted joint score

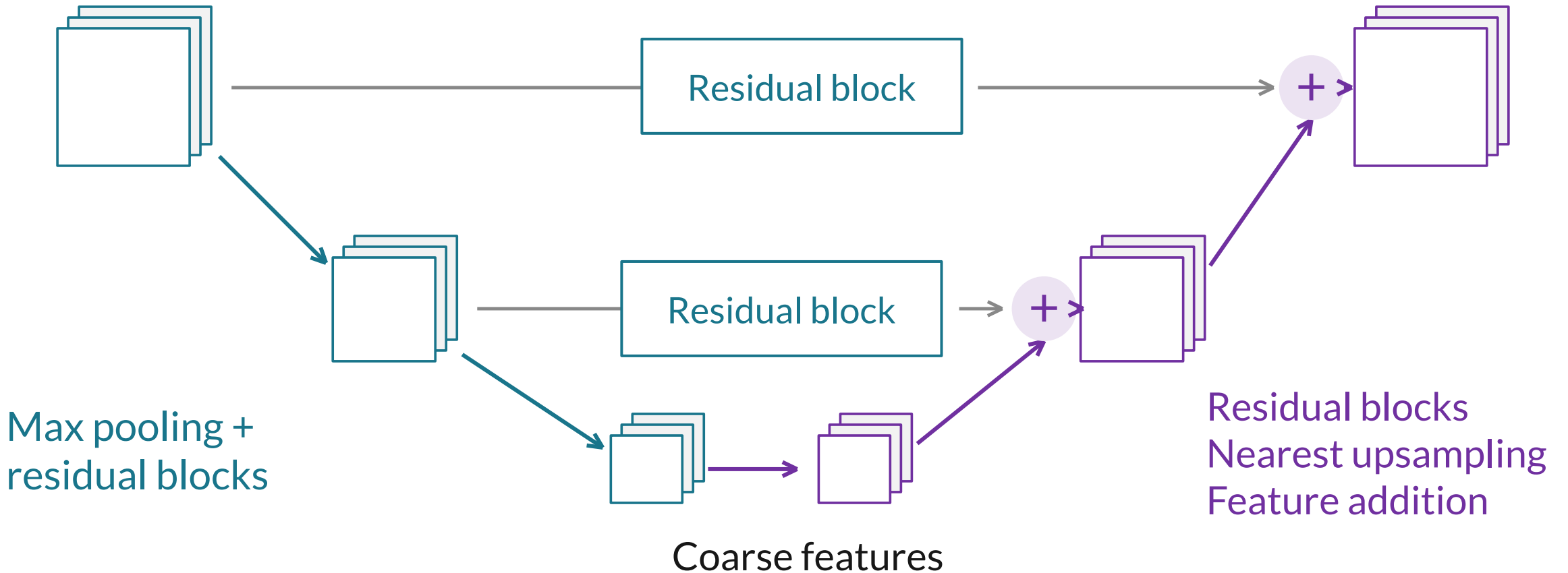
Find the spatial peak in each joint's heatmap.

Hourglass encoder-decoder

Decoded features predict one heatmap per joint.

Downsampling

Upsampling

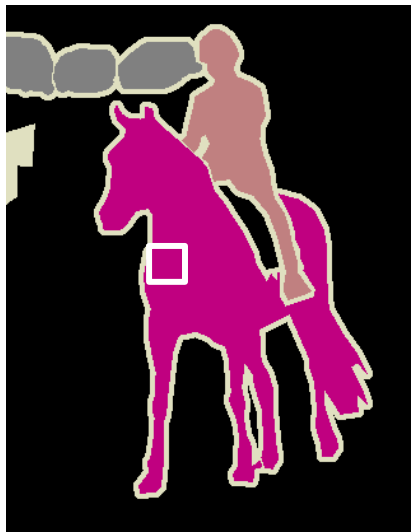


Lecture outline

- 01 Pixel prediction
- 02 Upsampling and decoders
- 03 Training and evaluation**
- 04 Spatial context

Pixel-wise cross-entropy

One labeled location



Class scores

Background s_{bg}

Horse s_{horse}

Person s_{person}

Target class y : horse

Across labeled locations

$$L = -\frac{1}{|\Omega|} \sum_{(i,j) \in \Omega} \log p_{ij, y_{ij}}$$

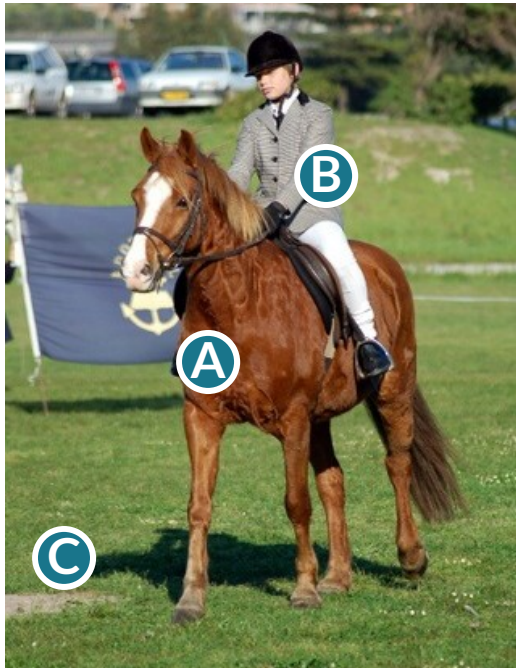
Softmax across classes
Select the probability for target y

Ω : labeled locations

$|\Omega|$: number of labeled locations

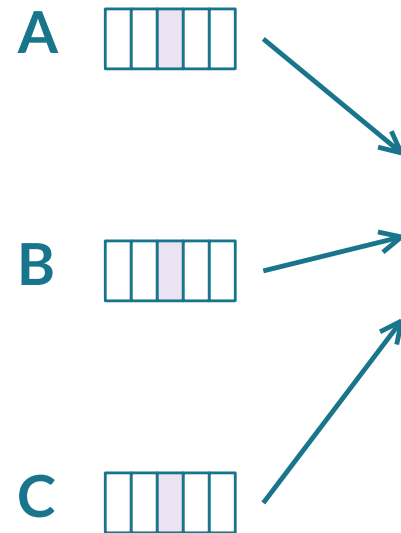
Shared weights, many supervised locations

One image



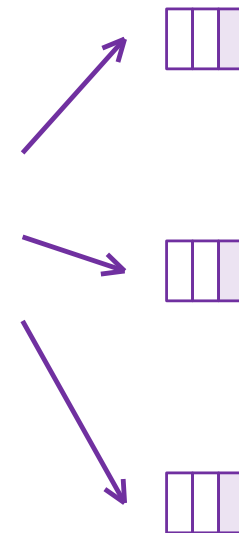
One parameter set
for every location

CNN feature vectors

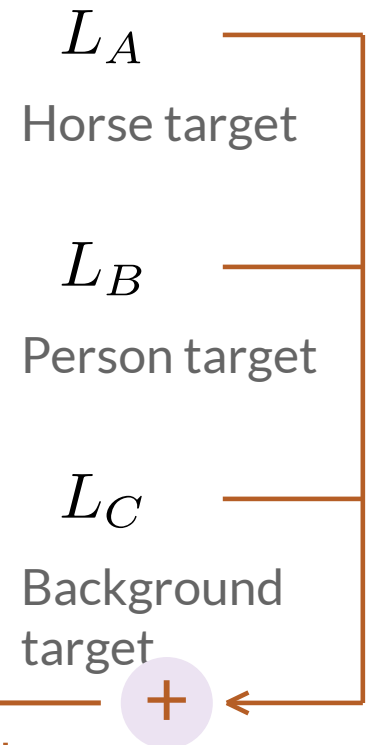


One shared
 1×1
classifier

Class scores



Local losses



Local gradients add for one update to the shared weights

Dense classification loss in PyTorch

Class scores

$$N \times K \times H \times W$$

Floating-point logits

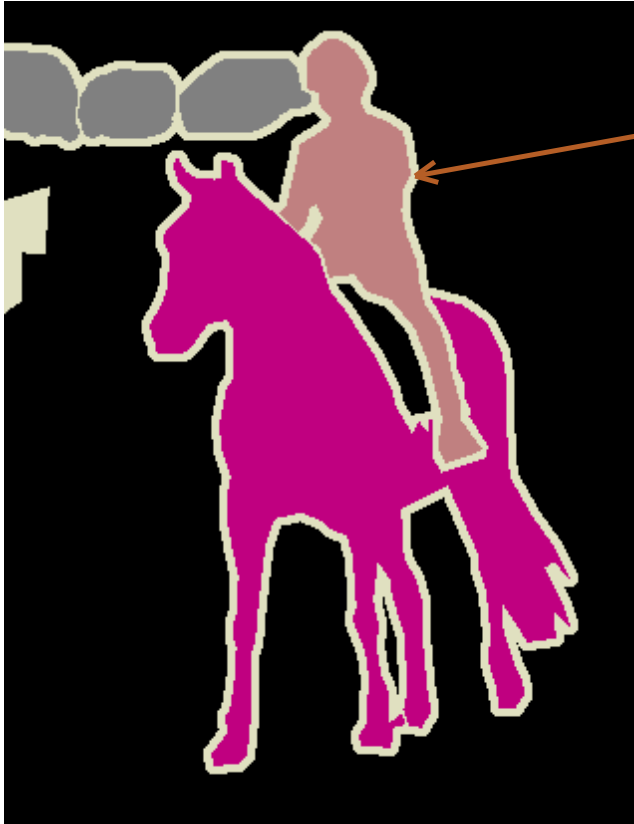
Target class IDs

$$N \times H \times W$$

torch.long

```
loss = F.cross_entropy(logits, target) # training loss
prediction = logits.argmax(dim=1)      # predicted class IDs
```

Unlabeled regions



Pale boundary: no class label

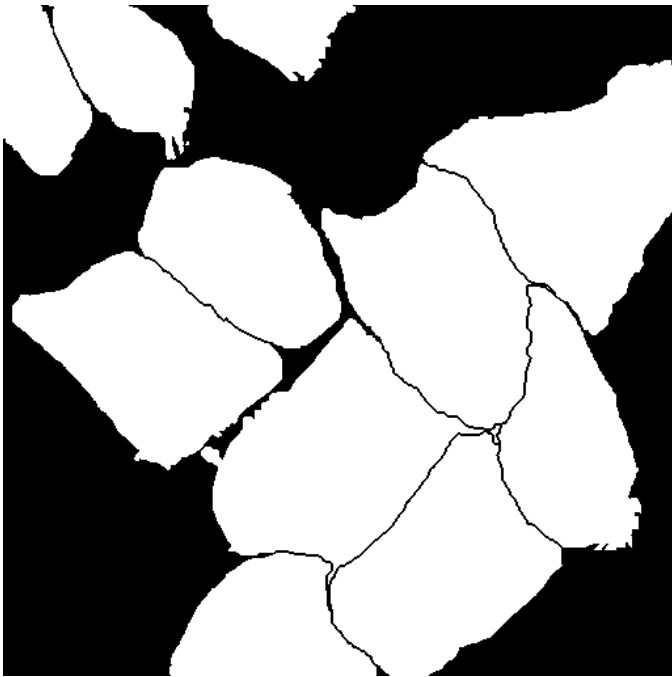
Background is a valid category.
Ignored pixels are excluded from the loss.

```
loss = F.cross_entropy(  
    logits, target, ignore_index=255  
)
```

Ignored target value: 255

Binary segmentation

Binary target



Foreground 1 · Background 0

One logit per pixel

$$p = \sigma(s)$$

s: logit p: foreground probability

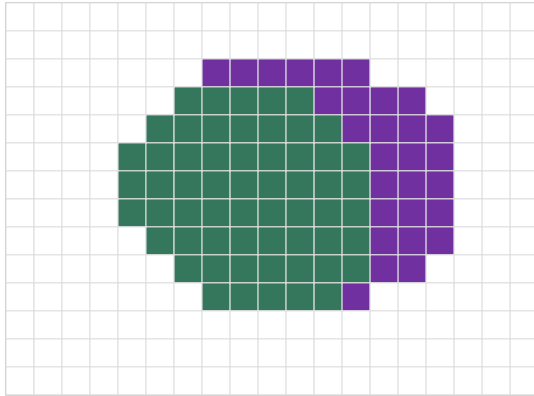
$$\hat{y} = \begin{cases} 1 & p \geq 0.5 \\ 0 & p < 0.5 \end{cases}$$

Train with binary cross-entropy (BCE)

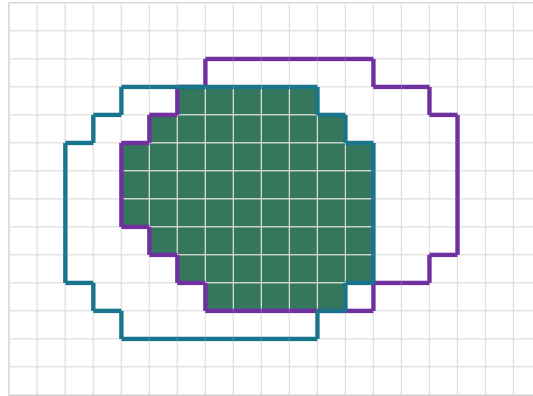
```
loss = F.binary_cross_entropy_with_logits(  
    logits, target # float mask, same shape  
)
```

Dice overlap

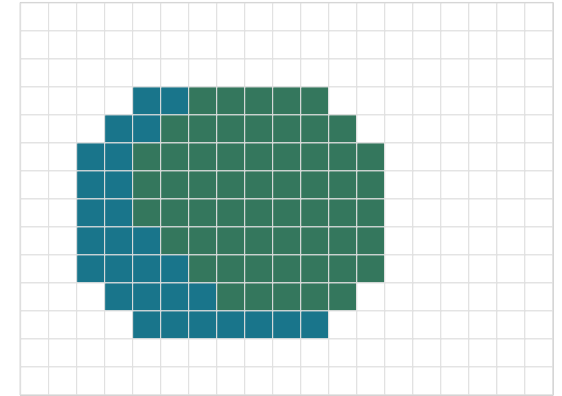
Prediction P



Shared foreground



Target G



$$\text{Dice}(P, G) = \frac{2|P \cap G|}{|P| + |G|}$$

Bars count pixels

The green pixels occur in both region sizes

Soft Dice loss

Keep probabilities so the loss is differentiable

$$L_{\text{Dice}} = 1 - \frac{2 \sum_{ij} p_{ij} y_{ij} + \epsilon}{\sum_{ij} p_{ij} + \sum_{ij} y_{ij} + \epsilon}$$

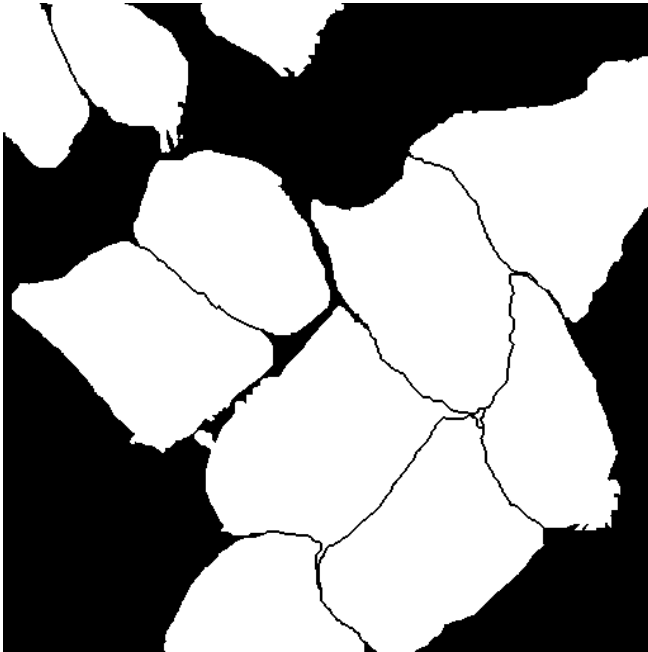
p: foreground probability

y: binary target

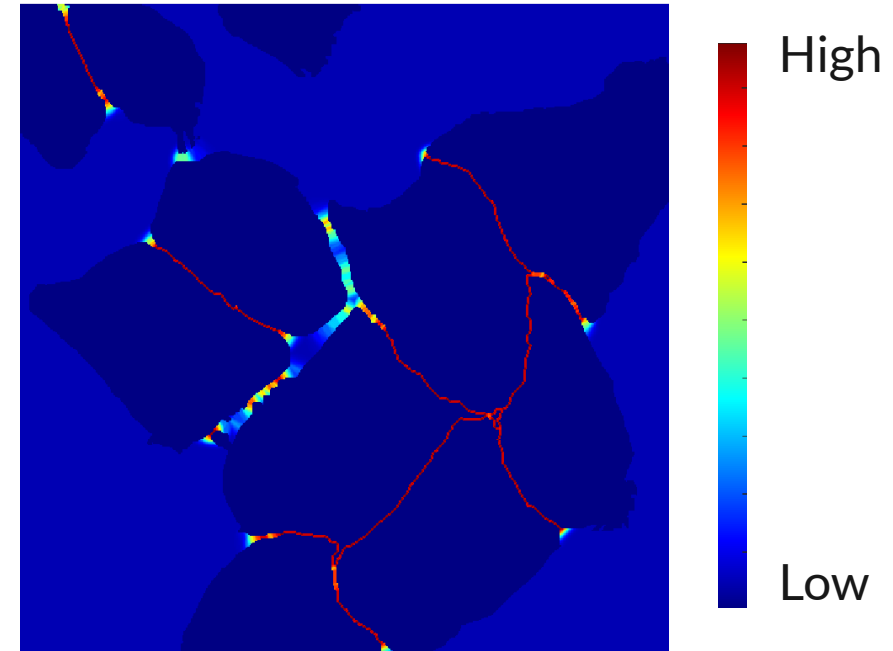
ϵ : small stabilizing constant

Weighted pixel losses

Cell / background target



Larger loss weights at separators



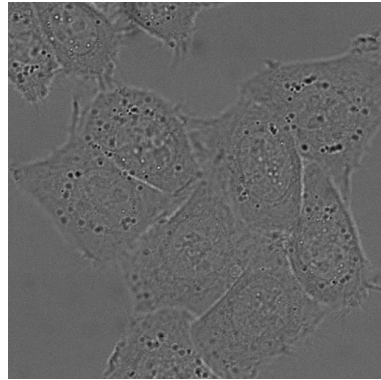
Multiply each pixel's loss by its map value before averaging

Paired geometric augmentation

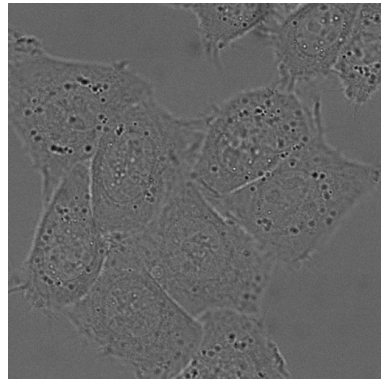
Image

Target mask

Original

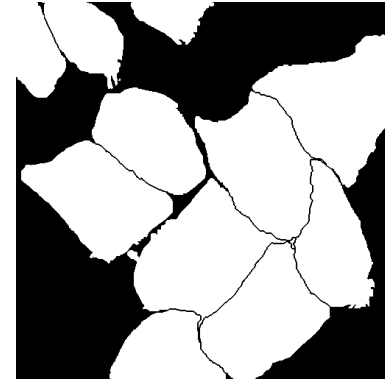


Flipped



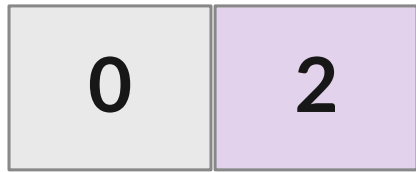
Same crop / flip /
rotation parameters

Color changes apply
only to the image



Resizing categorical masks

Class IDs at a boundary



Nearest neighbor



Average at the midpoint



Copy existing labels

Class 1 was never present at this boundary

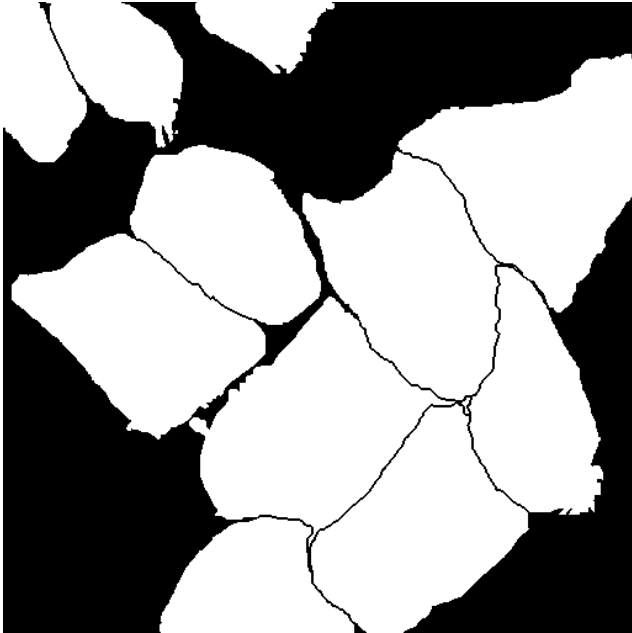
Matching the target resolution

Resize continuous class-score maps to the label grid

```
logits = F.interpolate(  
    logits, size=target.shape[-2:], # target height and width  
    mode="bilinear"  
)  
loss = F.cross_entropy(logits, target)
```

Resolution and fine structures

Original target



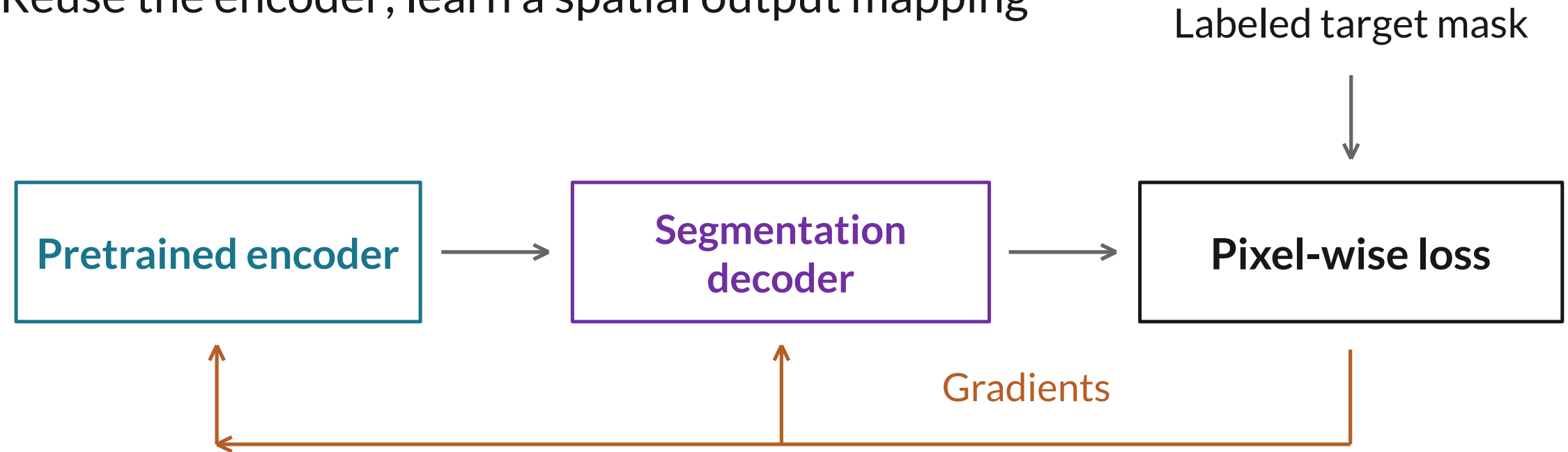
Coarser target, enlarged



Coarsening can remove thin gaps

Finetuning for segmentation

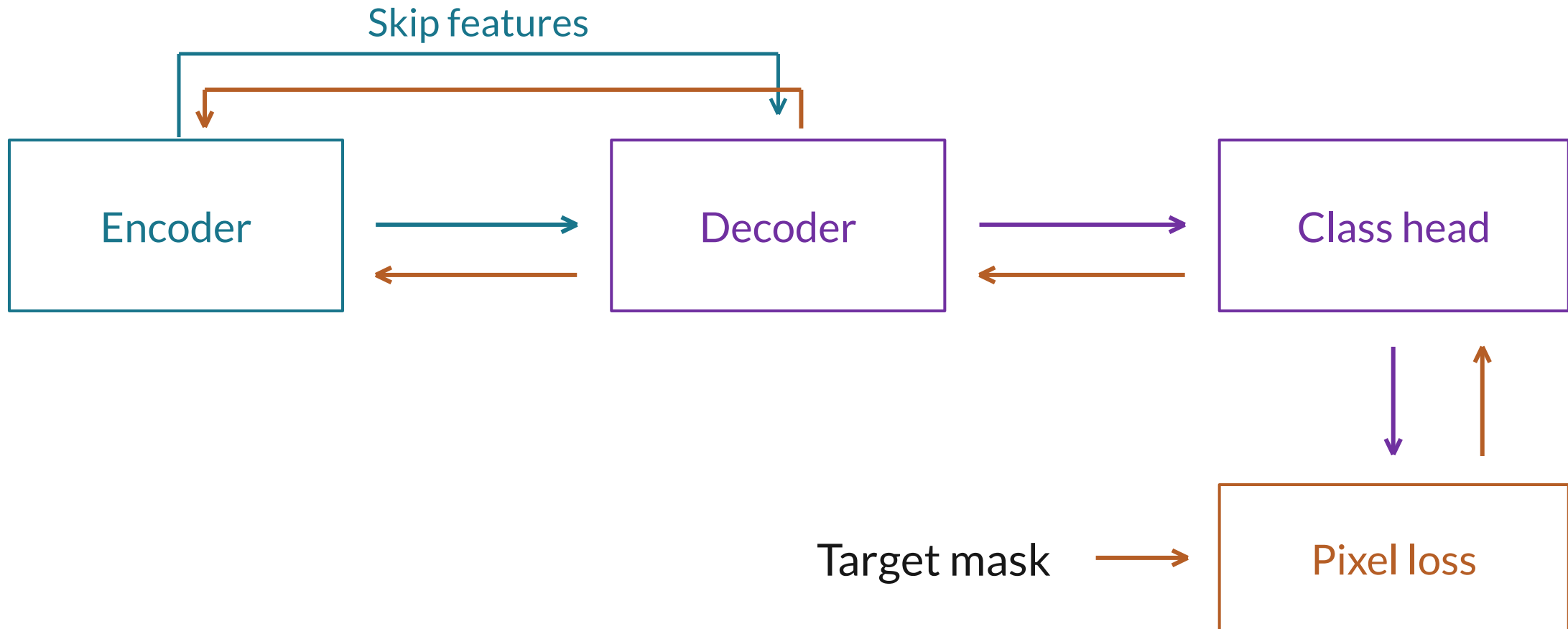
Reuse the encoder; learn a spatial output mapping



Small LR for pretrained layers

Train a new decoder for the task

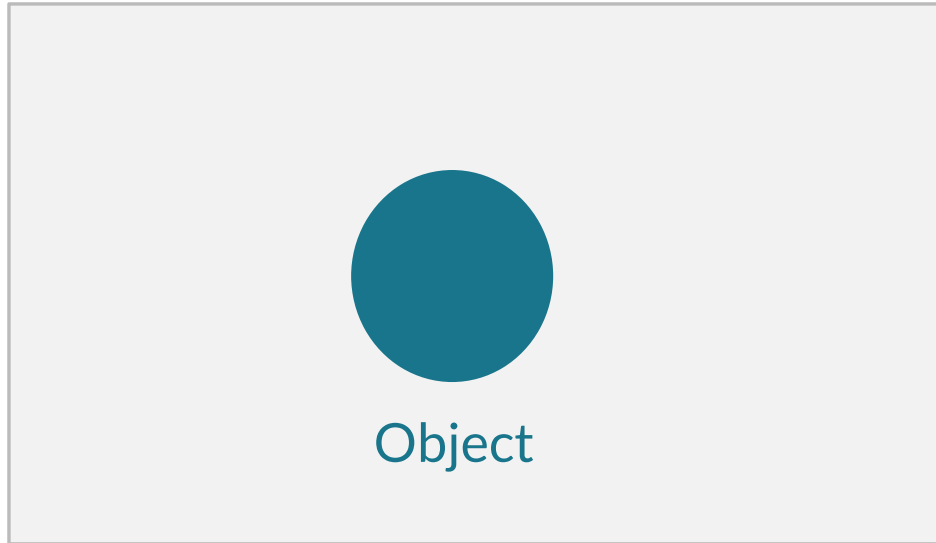
Training an encoder-decoder



Gradients follow both encoder routes

Foreground imbalance

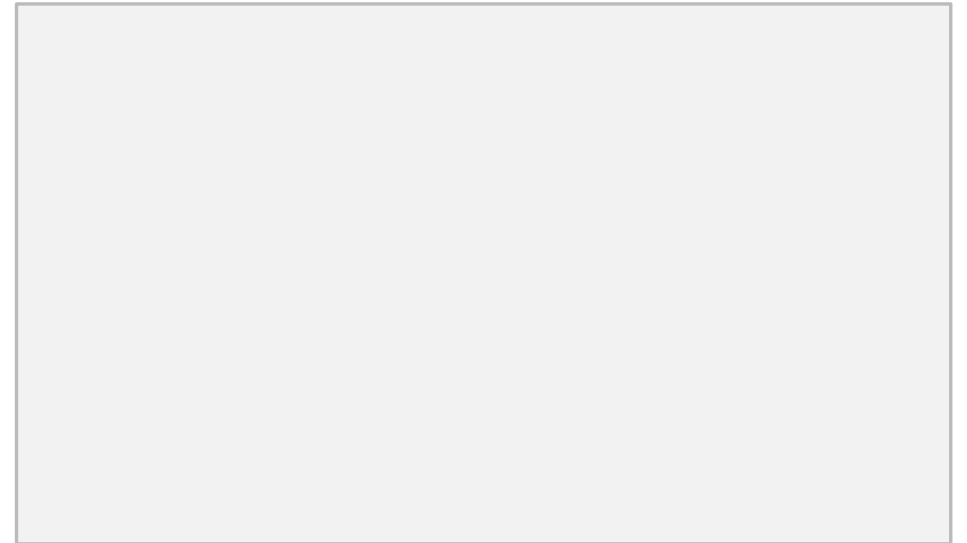
Target



Most pixels are background

Pixel accuracy: fraction of correctly labeled pixels

Always predict background

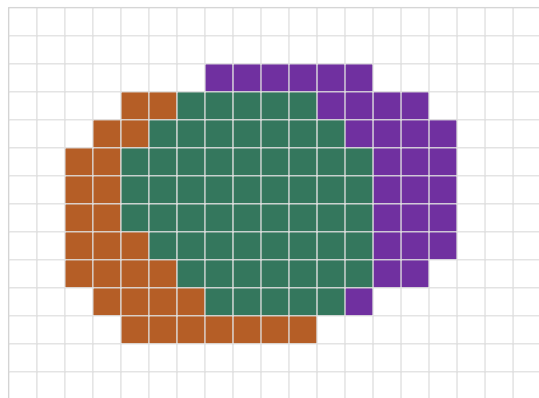


Many pixels correct

Foreground entirely missed

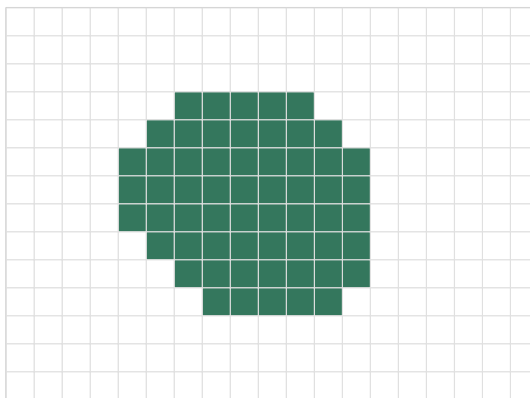
Intersection over Union

Region comparison

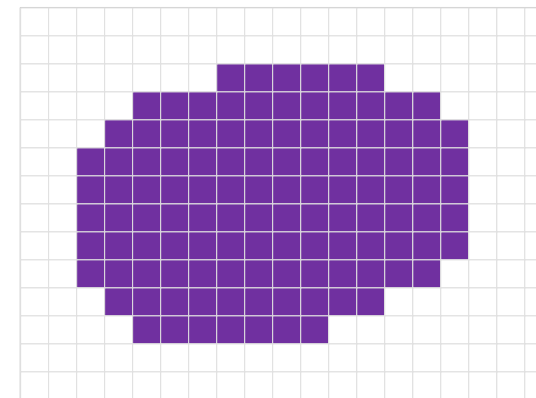


- Agreed foreground
- Extra prediction
- Missed target

Intersection



Union



$$\text{IoU}(P, G) = \frac{|P \cap G|}{|P \cup G|}$$

P: prediction G: ground truth $|\cdot|$: pixel count

Mean Intersection over Union

Accumulate each class's intersection and union across the test set



Person

$\text{IoU}_{\text{person}}$



Horse

$\text{IoU}_{\text{horse}}$



Car

IoU_{car}

$$\text{mIoU} = \frac{1}{K} \sum_{k=1}^K \text{IoU}_k$$

Reading segmentation results

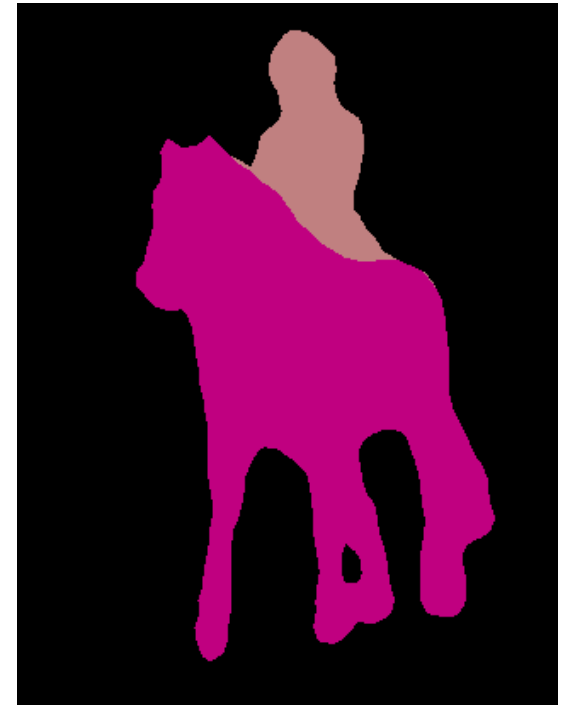
Input



Ground truth



FCN prediction



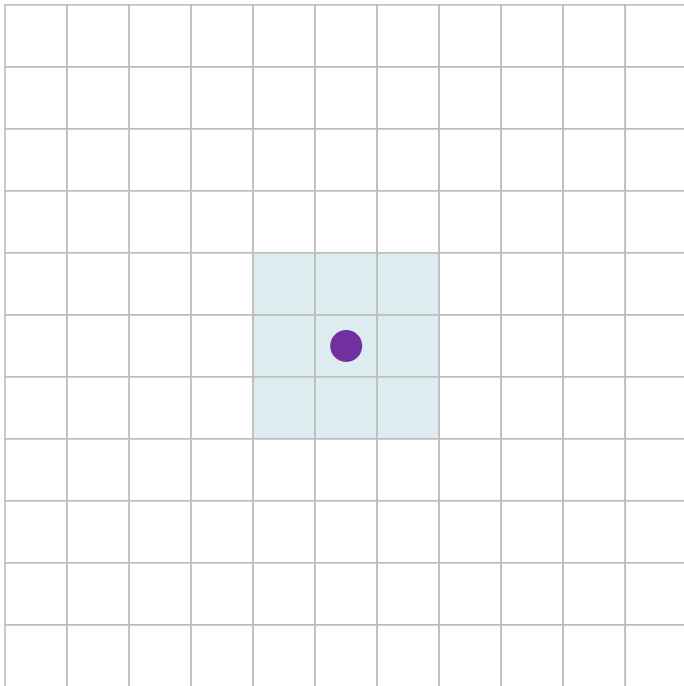
Lecture outline

- 01 Pixel prediction
- 02 Upsampling and decoders
- 03 Training and evaluation
- 04 Spatial context**

Context at high resolution

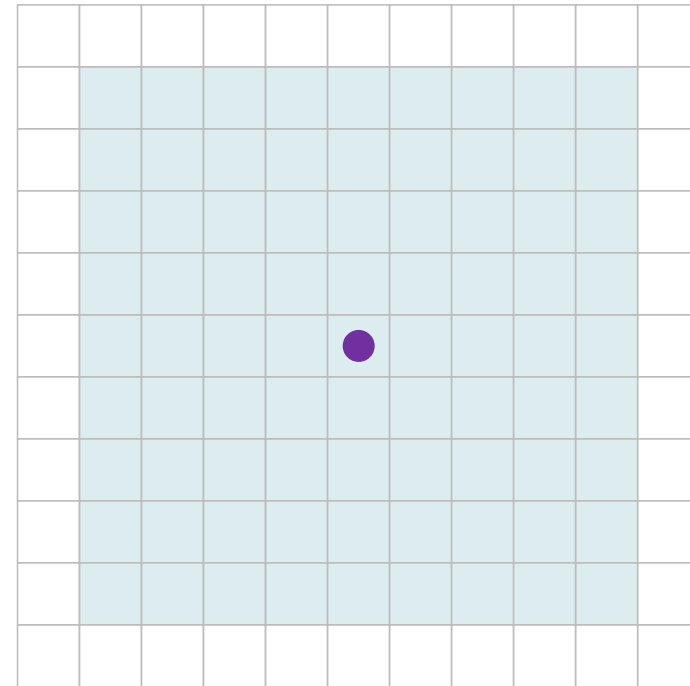
Input region used for one output

Local detail



● Output center

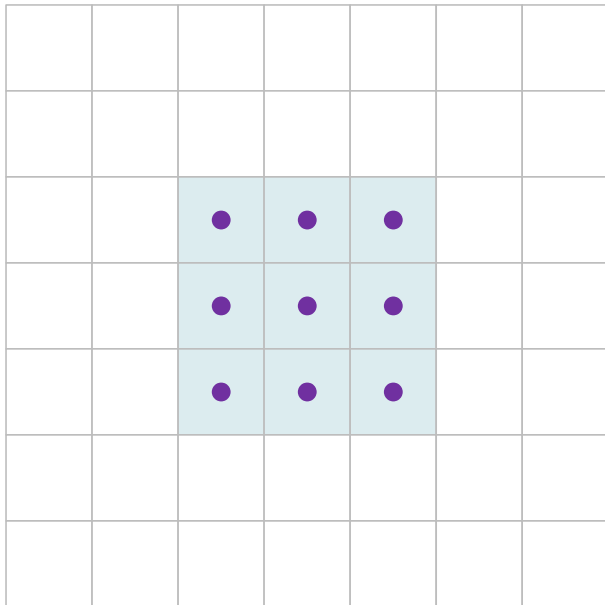
Wider context



Widen the view without another
downsampling step

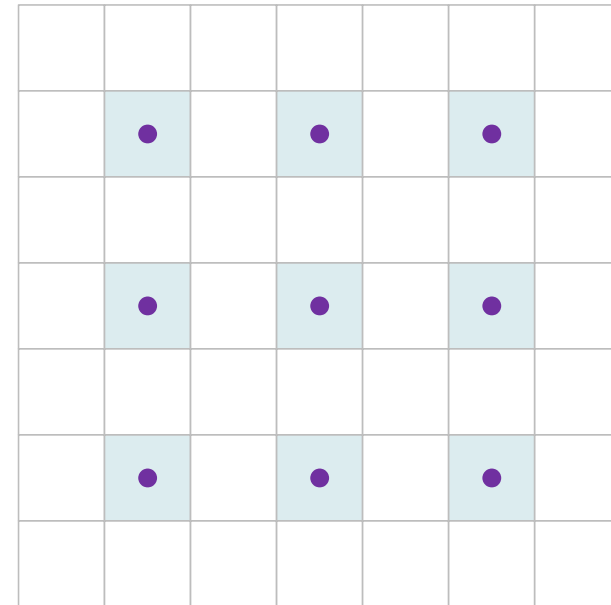
Dilated (atrous) convolution

Ordinary 3×3



Adjacent sample positions

Dilation rate 2



Spaced sample positions

Nine learned weights in both filters

Stride and dilation

Stride

Spacing of output centers



Stride 1



Stride 2

Dilation

Spacing of samples within a kernel



Rate 1



Rate 2

The dilated convolution operation

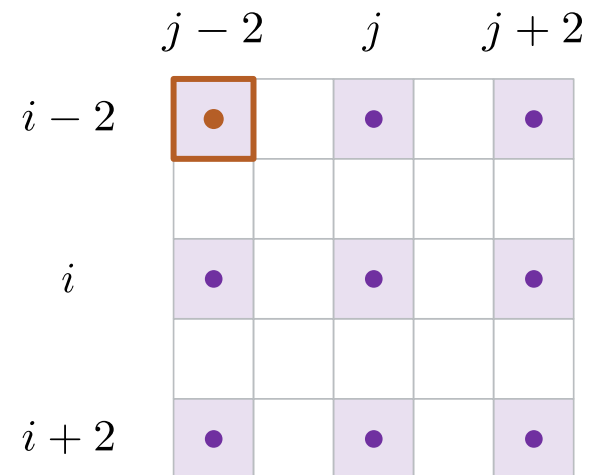
$$z_{i,j} = \sum_{a=-1}^1 \sum_{b=-1}^1 w_{a,b} x_{i+ra, j+rb}$$

- $z_{i,j}$ Output at row i , column j
- x, w Input values and kernel weights
- r Dilation rate
- a, b Kernel offsets: $-1, 0$, or 1

Upper-left sampled term

$$w_{-1,-1} x_{i-2,j-2}$$

Rate 2: sampled coordinates

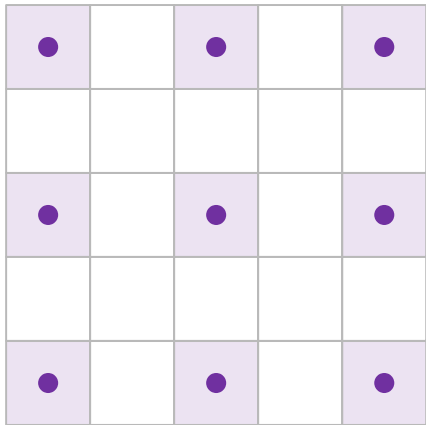


One input/output channel; bias omitted

Output size with dilation

Dilation changes the effective kernel size.

3 × 3 kernel, rate 2



Effective size: 5 × 5

$$k_{\text{eff}} = r(k - 1) + 1$$

$$H_{\text{out}} = \left\lfloor \frac{H_{\text{in}} + 2p - k_{\text{eff}}}{s} \right\rfloor + 1$$

$$W_{\text{out}} = \left\lfloor \frac{W_{\text{in}} + 2p - k_{\text{eff}}}{s} \right\rfloor + 1$$

k: kernel size

r: dilation rate

p: padding per side

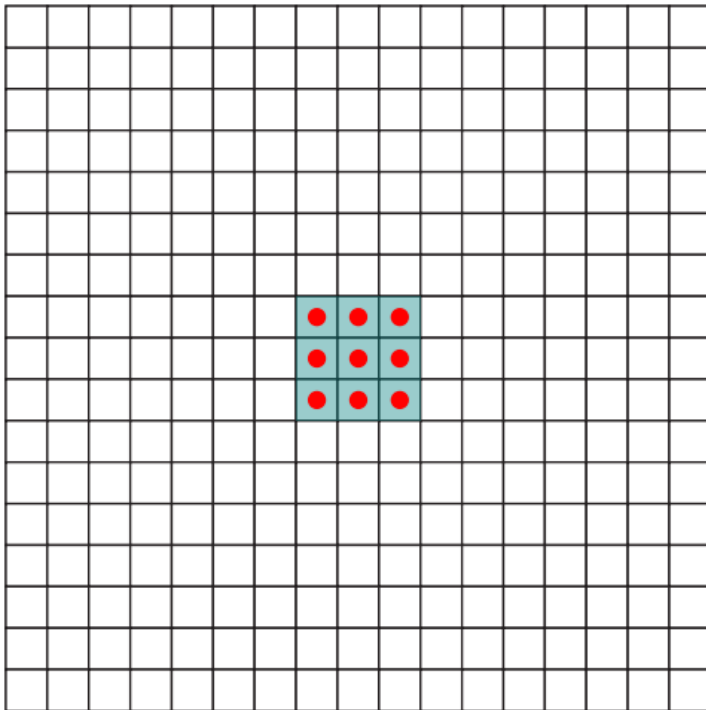
s: stride

H and W: height and width • $\lfloor \rfloor$ rounds down

Stacked dilated layers

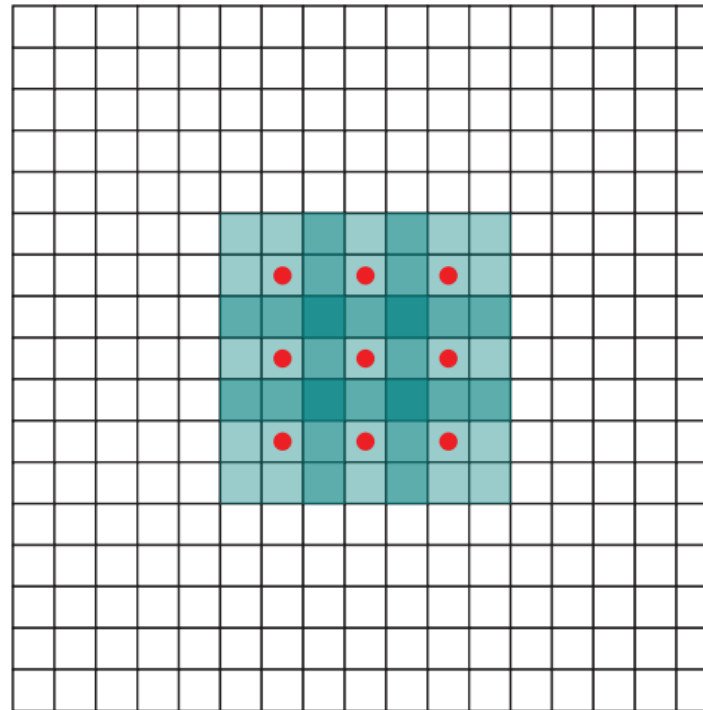
Dots: preceding-layer samples · shading: input coverage

Rate 1



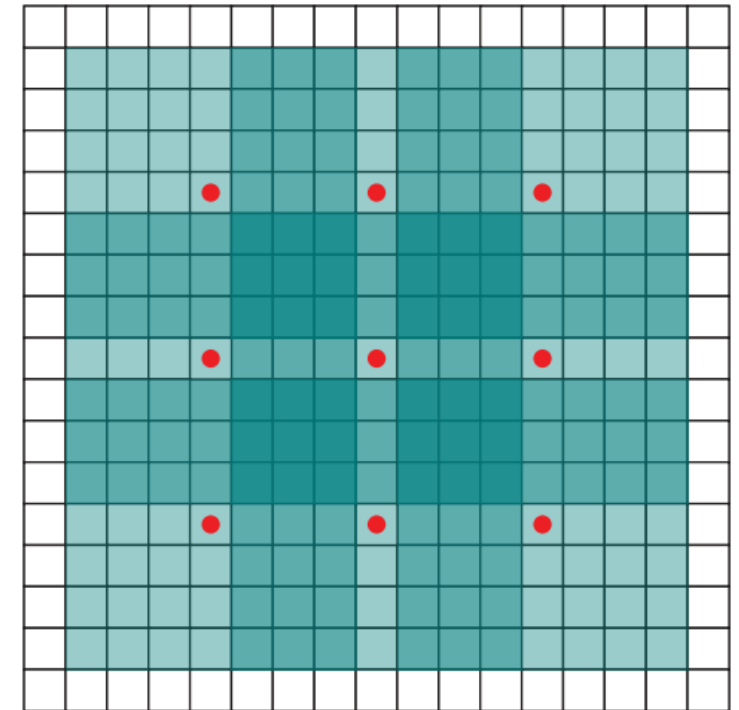
3×3 input coverage

Then rate 2



7×7 input coverage

Then rate 4



15×15 input coverage

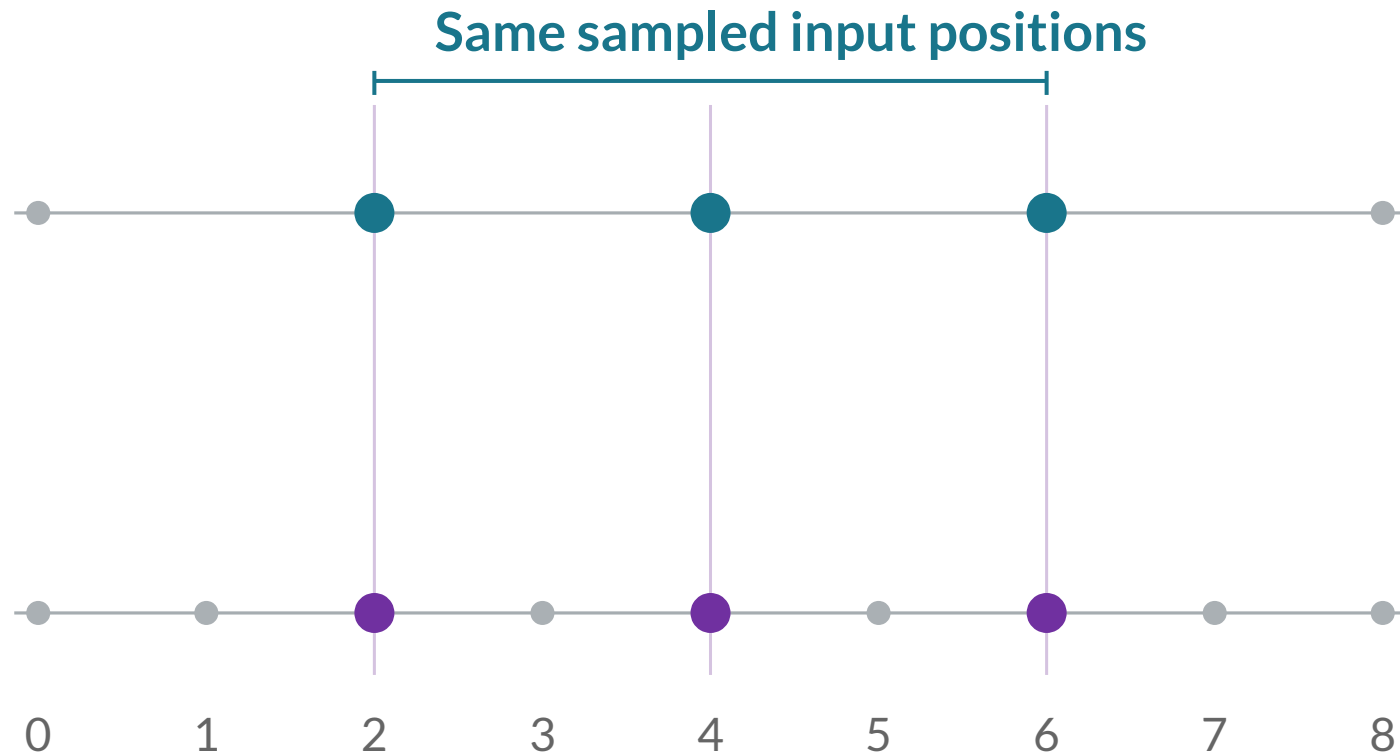
Adapting the output stride

Removing a stride-2 step makes the feature grid denser.

One spatial row

Stride-2 stage retained

Later convolution: rate 1



Stride-2 stage removed

Later convolution: rate 2

Positions on the finer grid

Denser predictions cost more computation and memory.

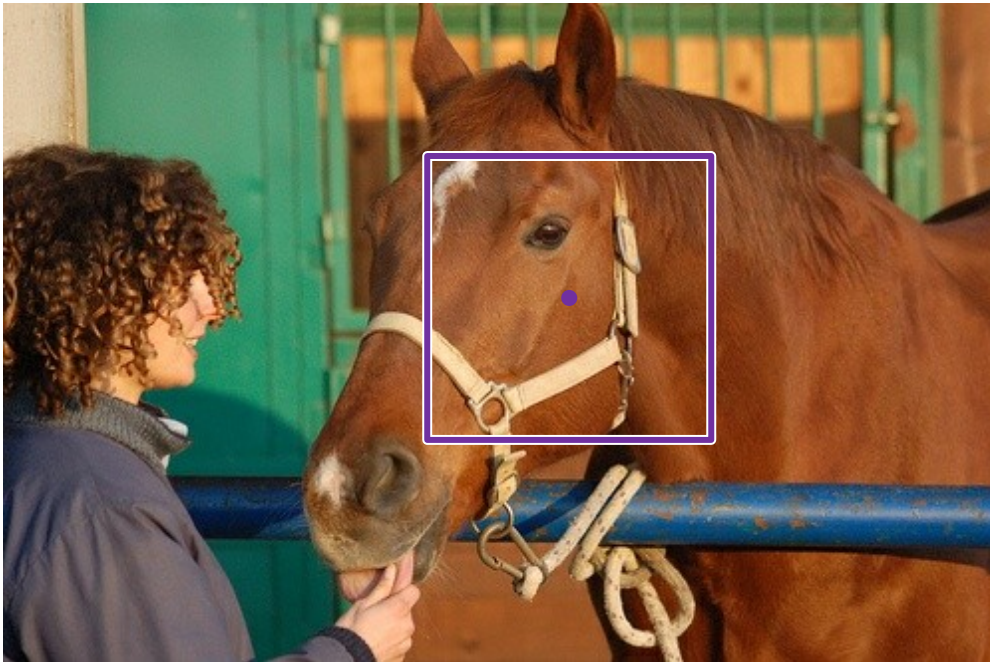
Dilated convolution in PyTorch

Input x: $N \times C \times H \times W$ D: output channels

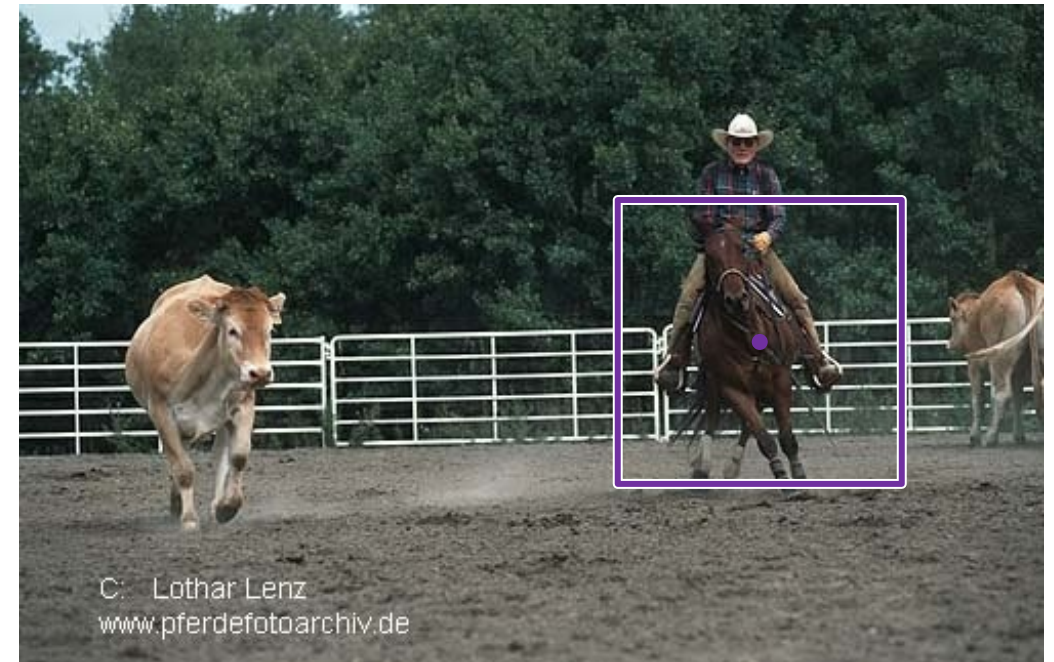
```
layer = nn.Conv2d(  
    C, D, kernel_size=3,  
    stride=1, dilation=2, padding=2  
)  
features = layer(x)  # output:  $N \times D \times H \times W$ 
```

Objects at different scales

Nearby object



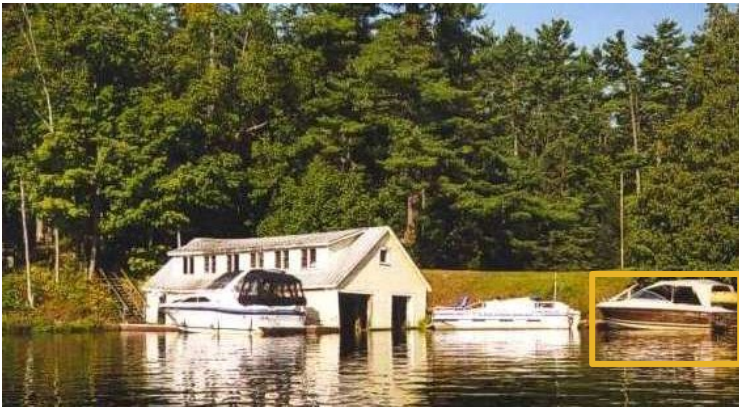
Distant objects



The same-sized neighborhood covers less of a large object.

Scene context

Input image



FCN baseline



PSPNet



A boat can resemble a car in a small image region.

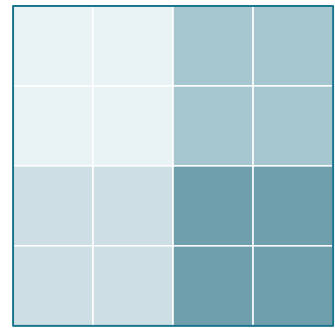
 Car label

 Boat label

Image-level context

One feature channel

Spatial features

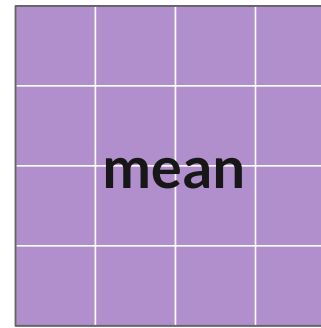


Global average

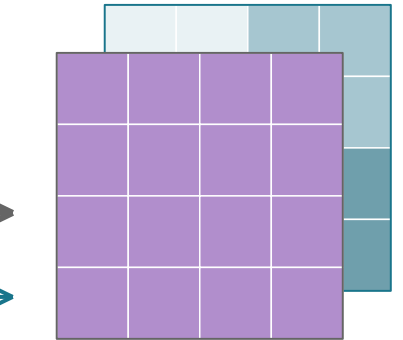
mean

Repeat

Repeated mean



Keep the original spatial features



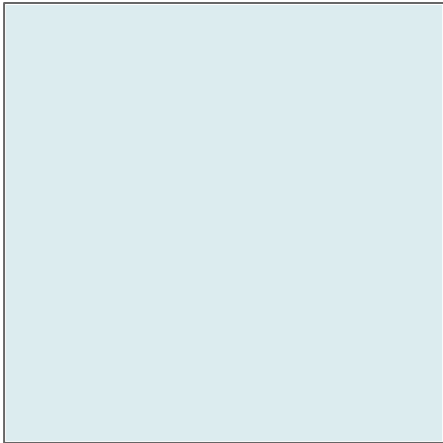
Concatenate
+ convolution

A global summary supplies context; local features retain position.

Pyramid pooling

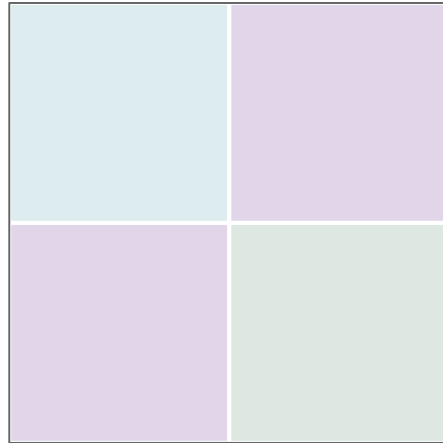
Pool separate regions of the same input feature map

1×1



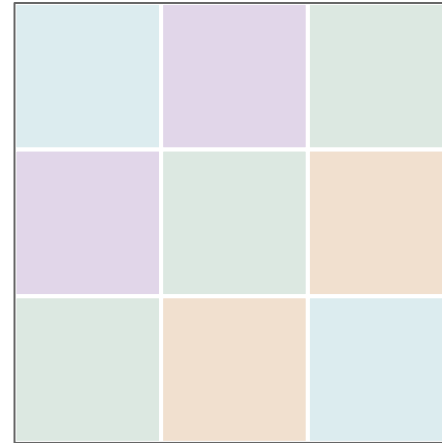
Whole-image
summary

2×2

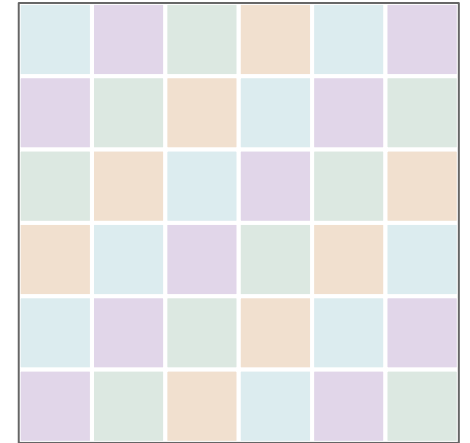


One pooled channel vector per region

3×3



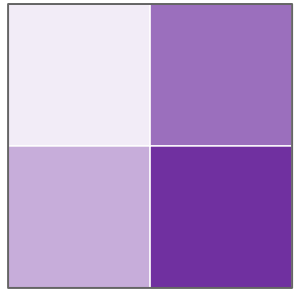
6×6



Combining pooled context

One pooling branch

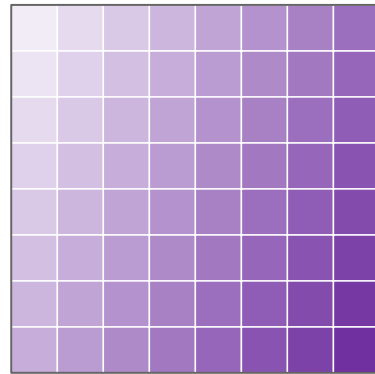
Pooled channel



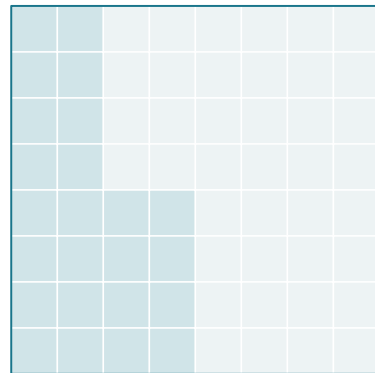
After the 1×1
channel transform

Bilinear resize

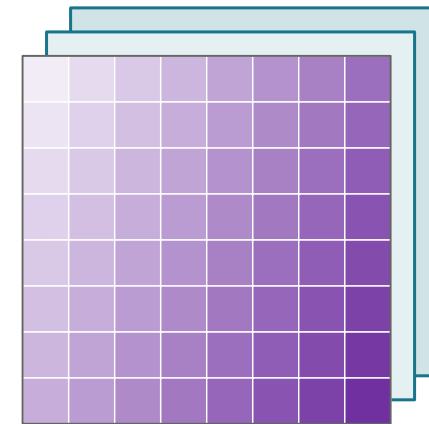
Resized context



Original features



Same spatial grid

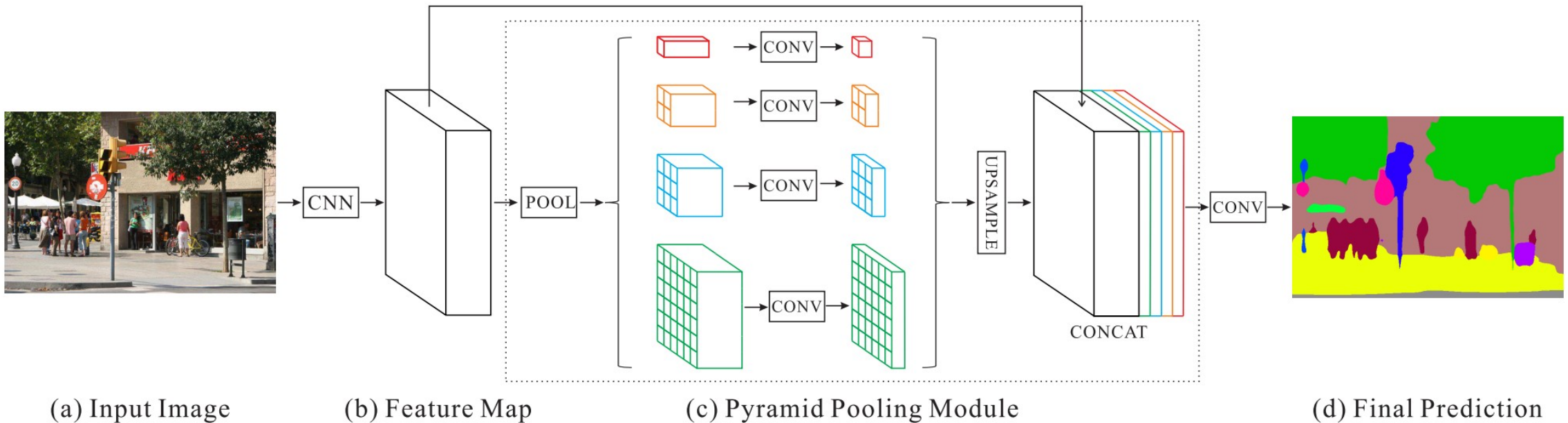


Concatenate channels

Convolution learns
their combination

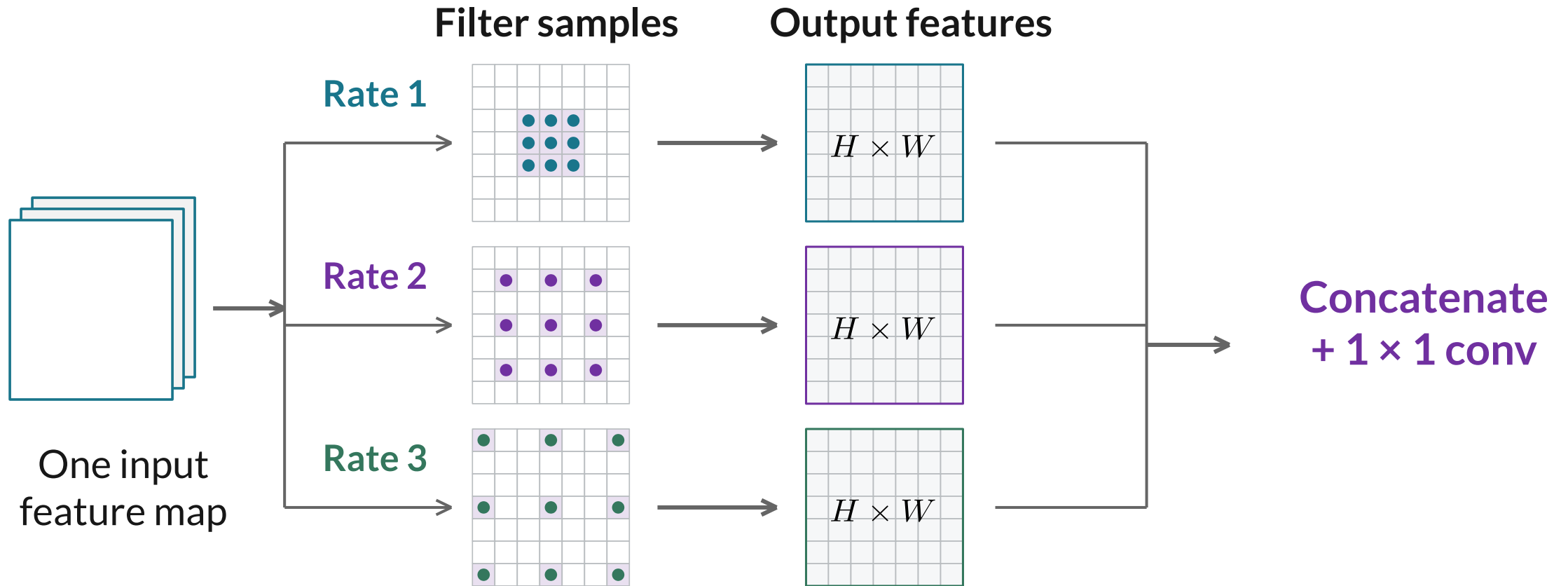
PSPNet

Pyramid Scene Parsing Network



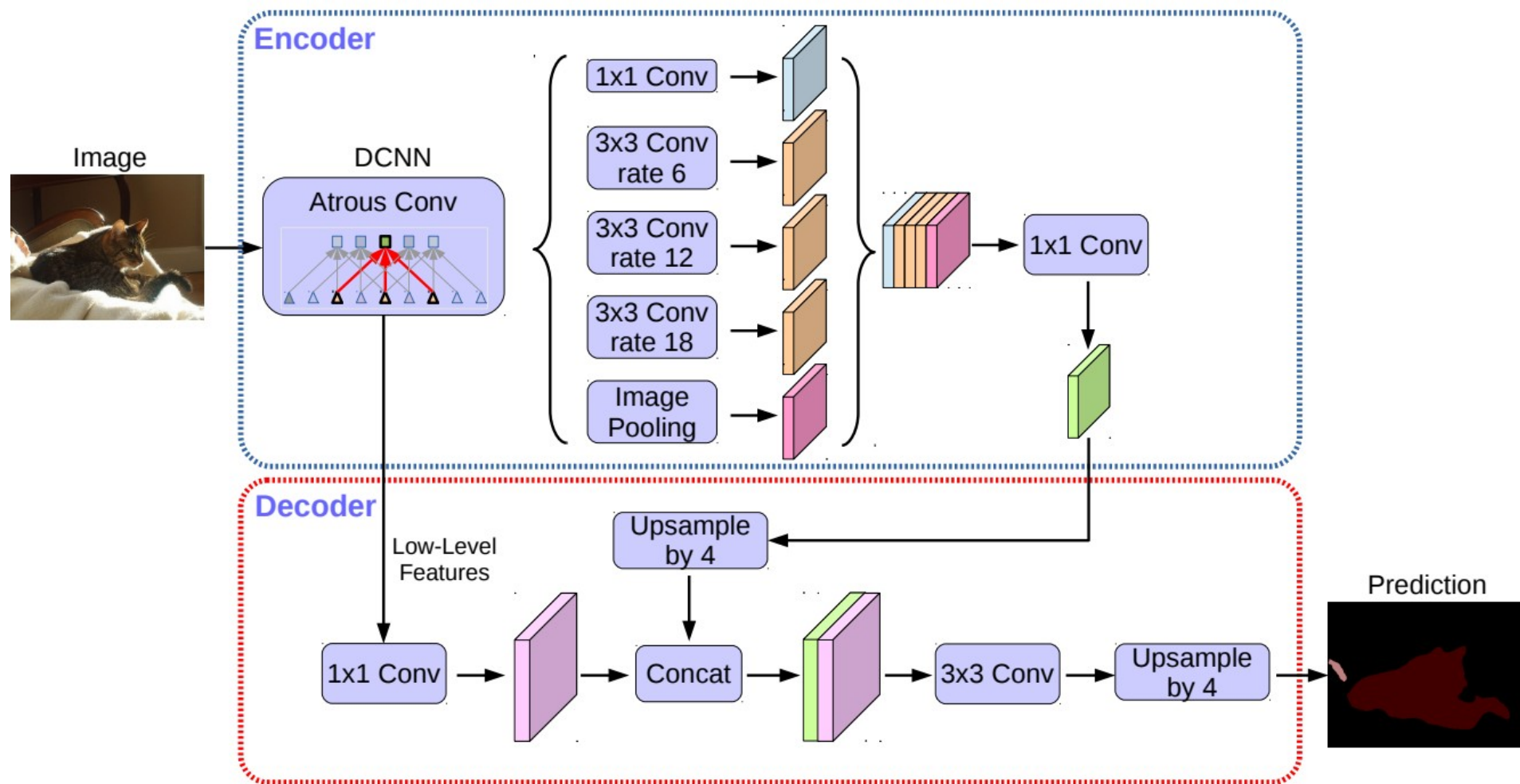
Dilated branches of ASPP

Atrous spatial pyramid pooling: parallel 3×3 filters



The full model also uses 1×1 and image-pooling branches.

DeepLabv3+



Decoder refinement

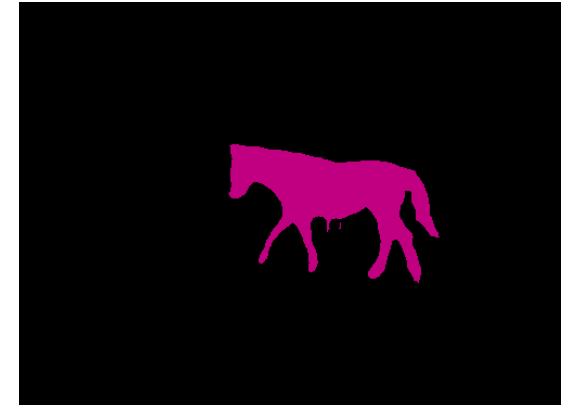
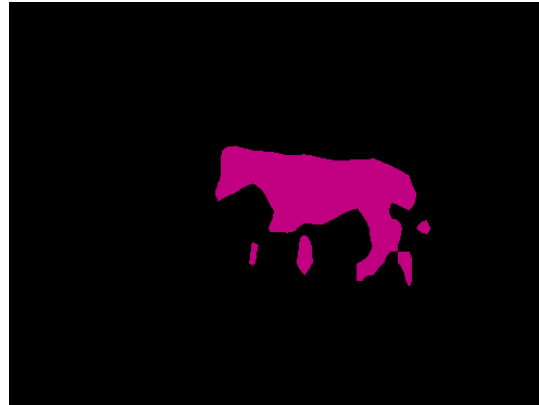
Input image



Bilinear upsampling



Learned decoder



Open-vocabulary segmentation: SAM 3

Fixed-class models use predefined categories.

Text or examples specify what to segment.

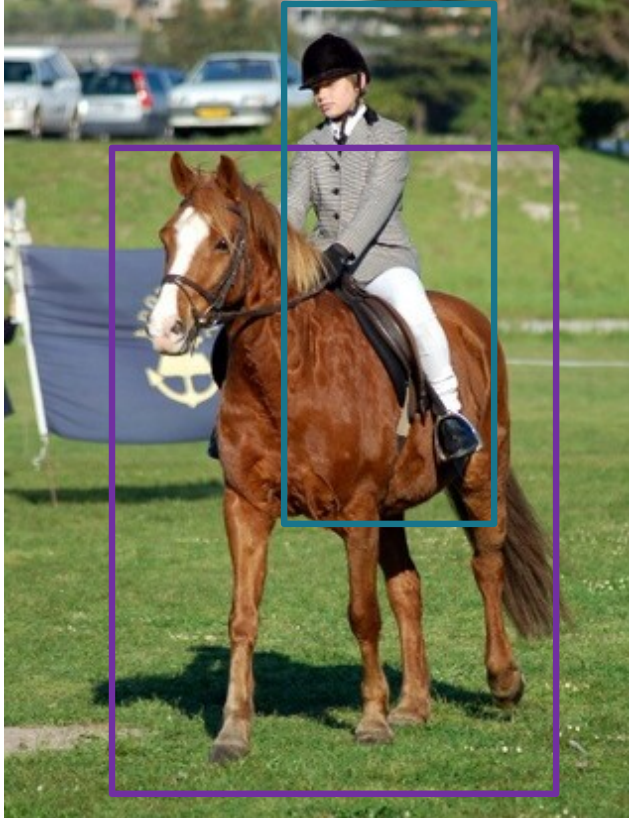
Prompt: “a fish” + example



Output masks



Next: object detection



Horse

Person

Predict a box and category
for each object

The number of objects can vary