Lecture 5

Segmentation

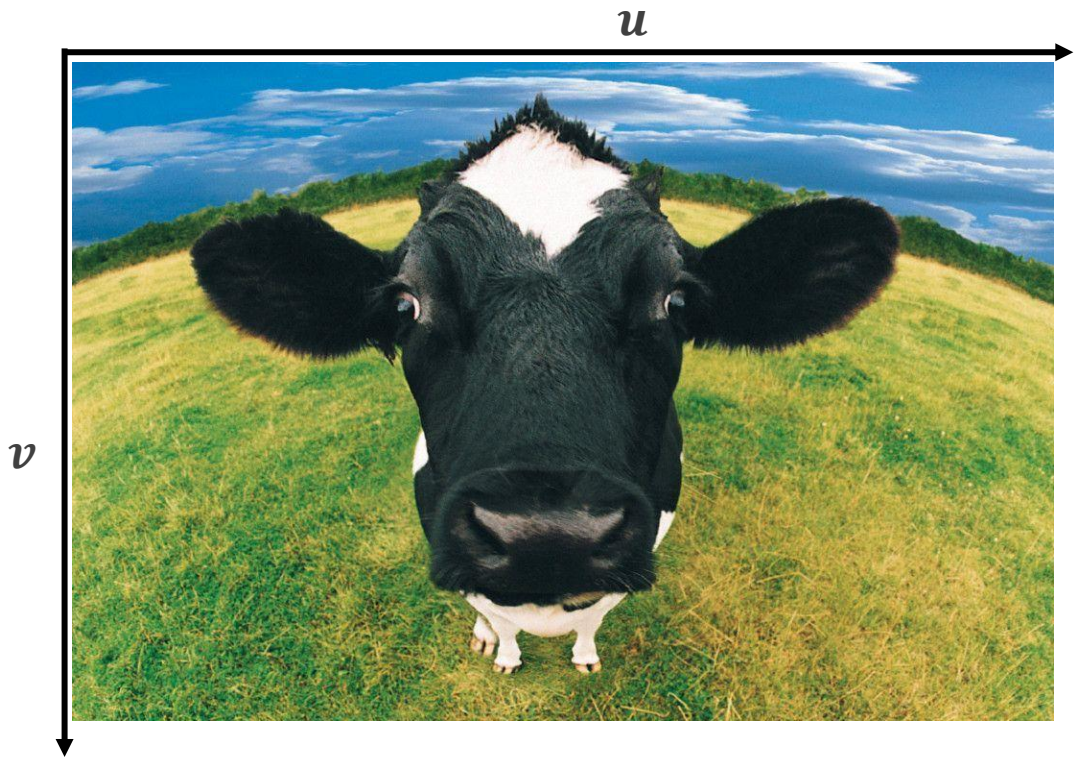
MACHINE LEARNING FOR COMPUTER VISION (ML4CV)

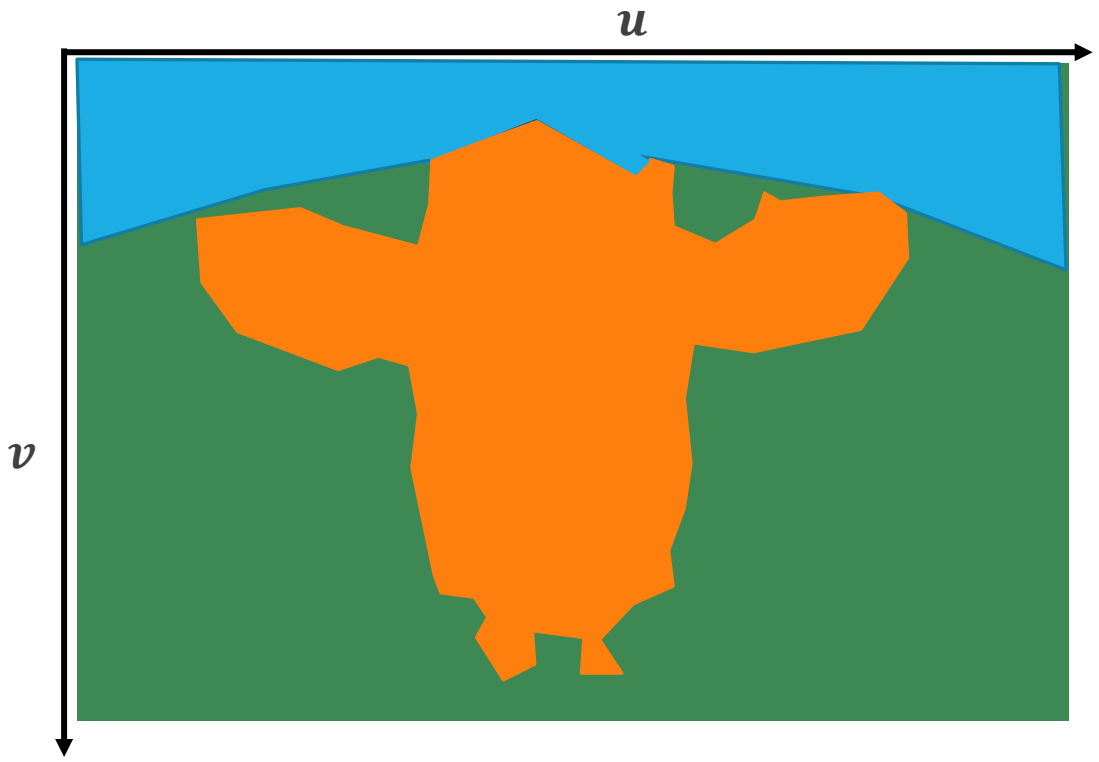
SAMUELE SALTI

Semantic segmentation

Input: RGB Image of size $W \times H$

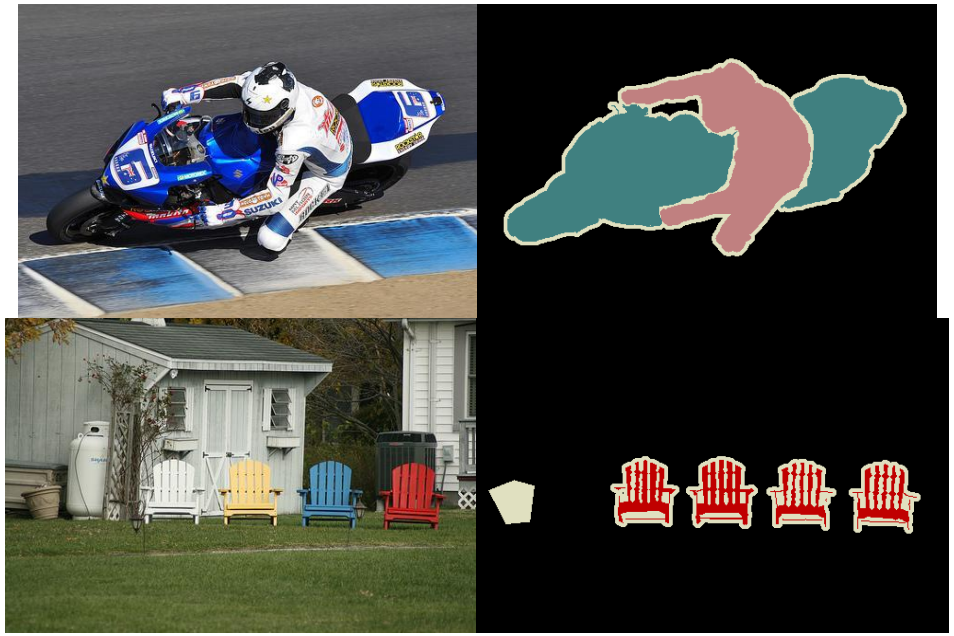
Output: a category c_{uv} for each pixel $p = (u, v)$, $c_{uv} \in [1, \ldots, C]$ (a fixed list of categories, as in image classification)





Datasets

PASCAL2
Pattern Analysis, Statistical Modelling and Computational Learning



Trainval images: 11540 (6,929 segmentation masks)
20 categories

http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html

ADE20K

train/val images: 25K/2K
150 categories

https://groups.csail.mit.edu/vision/datasets/ADE20K/

Specialized datasets

CITYSCAPES DATASET

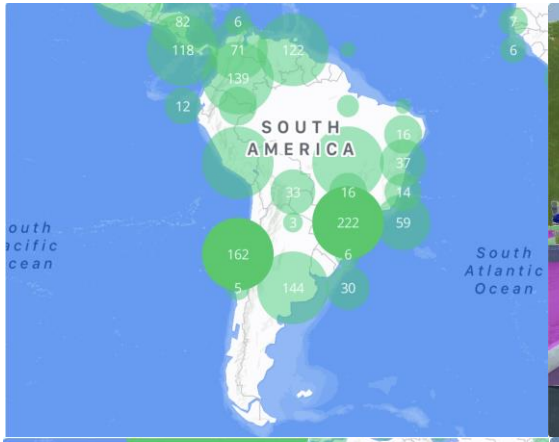
https://www.cityscapes-dataset.com/

train/val images: 2750/500
30 categories, 19 used

Mapillary Vistas

https://www.mapillary.com/dataset/vistas

train/val images: 25,000
124 categories, diverse geographies

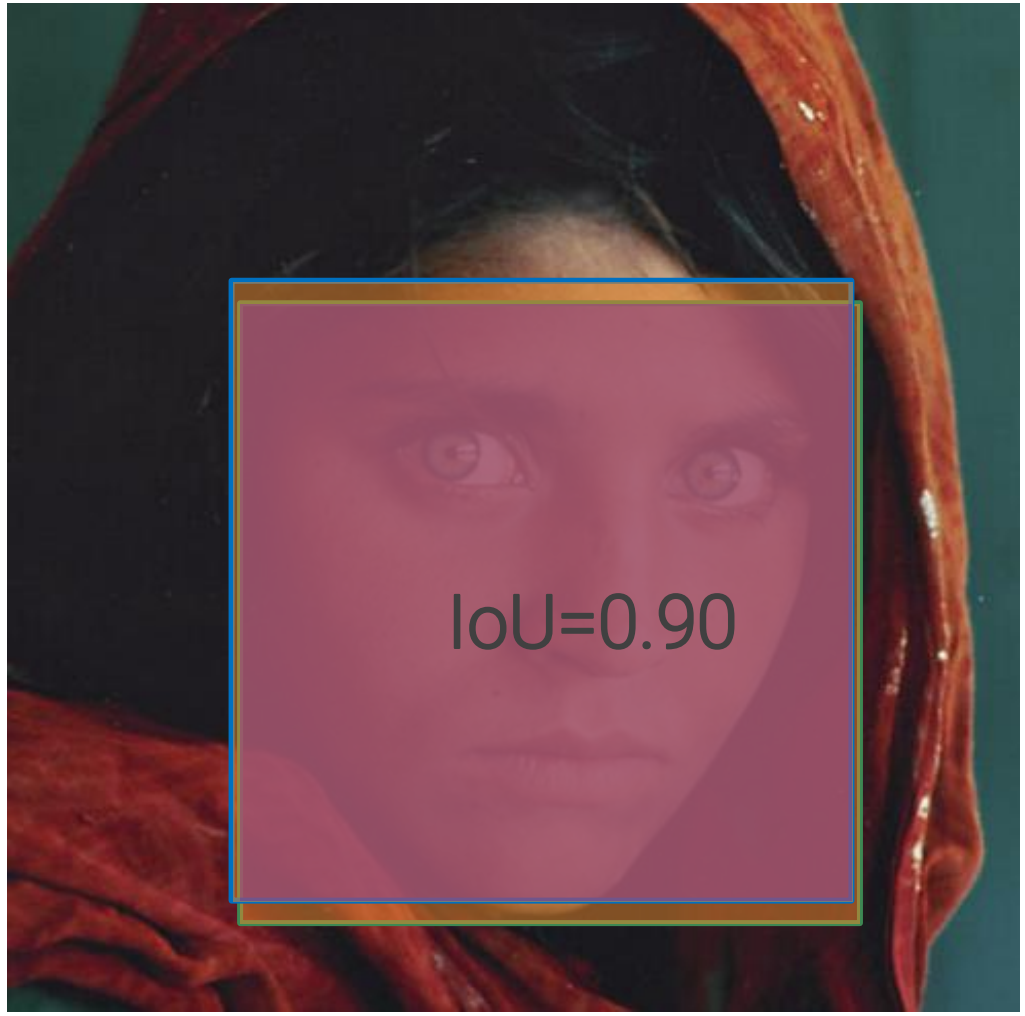


Recall: Box overlap

To check if a prediction and a GT box overlap, we measure the Intersection over Union (IoU) score (aka Jaccard index or similarity)

$$IoU(BB_i, BB_j) = \frac{area\ of\ intersection}{area\ of\ union}$$

$$= \frac{|BB_i \cap BB_j|}{|BB_i| + |BB_j| - |BB_i \cap BB_j|}$$



0.55 corresponds to a partial overlap

0.75 corresponds to a good overlap

0.90 corresponds to perfect overlap

Pixel-wise IoU

Intersection over union can be generalized to pixel-wise segmentation masks

For a class $c = 1, \dots, C$

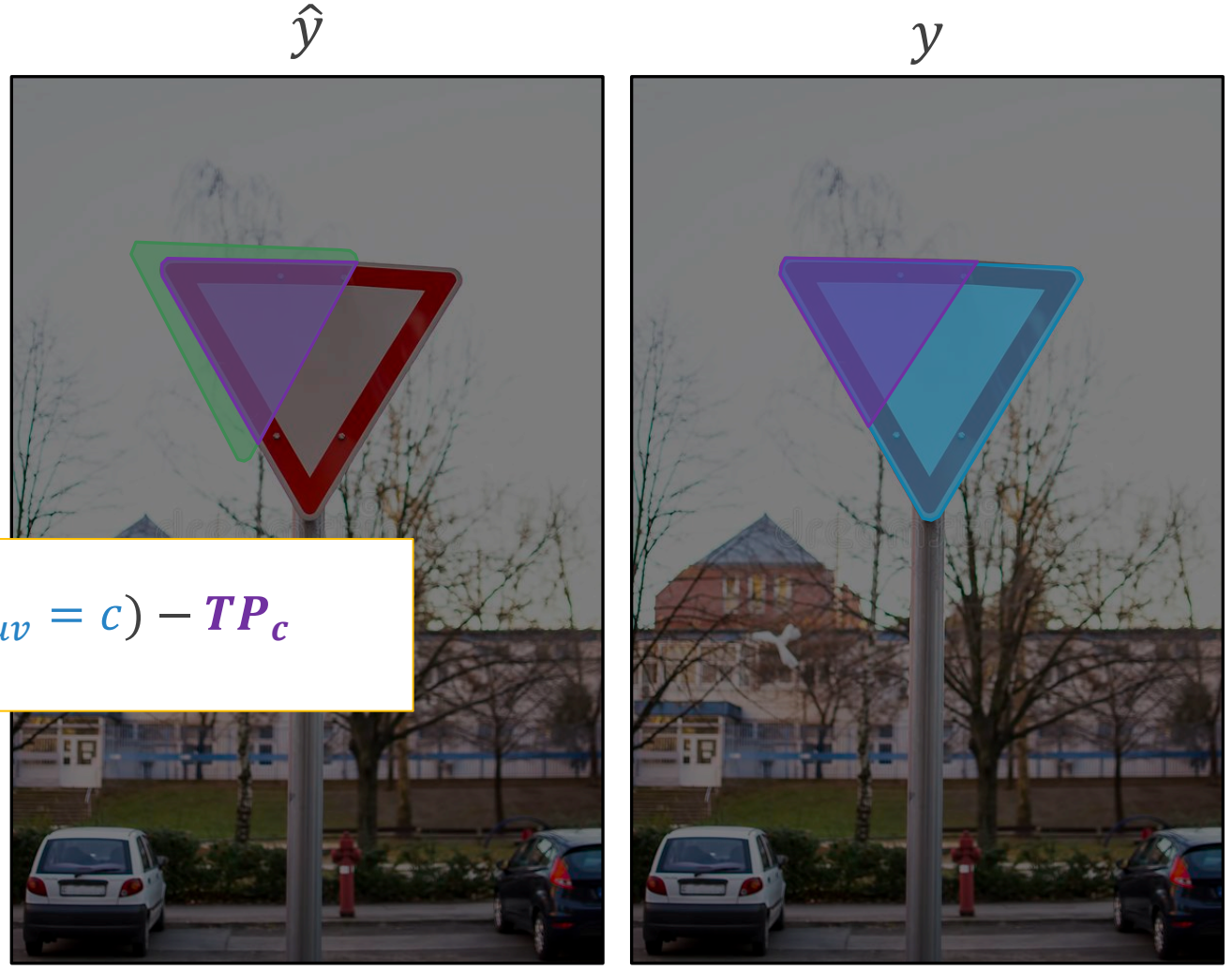
$$TP_c = \sum_{images} \#\ pixels\ where\ y_{uv} = c\ and\ \hat{y}_{uv} = c$$

$$IoU_c = \frac{area\ of\ intersection}{area\ of\ union}$$

$$\sum_{images} (\#pixels\ where\ \hat{y}_{uv} = c + \#pixels\ where\ y_{uv} = c) - \boldsymbol{TP_c}$$

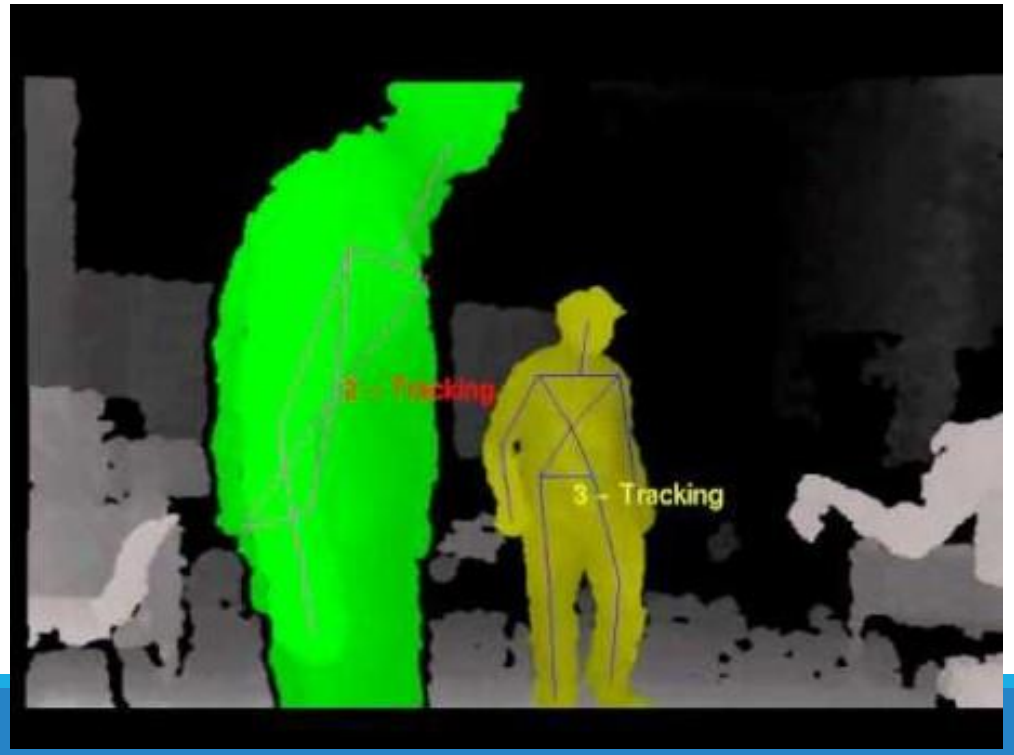
To compute mIoU score for a dataset, we average IoU_c over classes

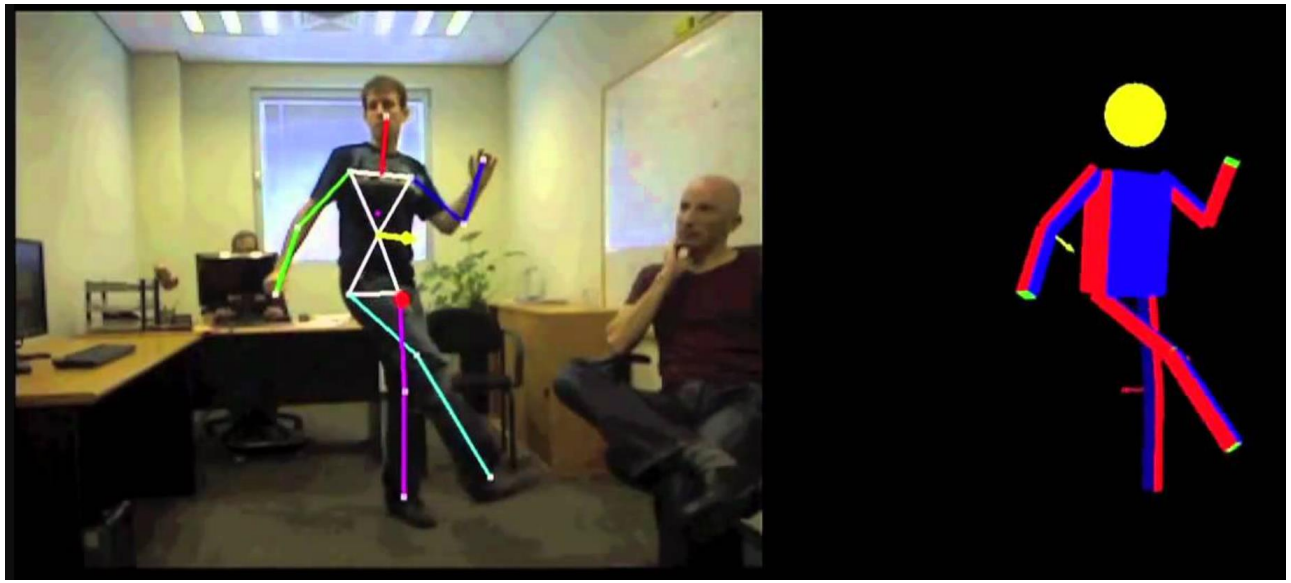
$$mIoU = \frac{1}{C}\sum_{c=1}^{C} IoU_c$$



Kinect and human pose estimation

One of the most successful applications of semantic segmentation in commercial applications has been the human pose estimation algorithm running on data collected by the Kinect controller for the Xbox gaming console.





Pipeline

Capture depth image and remove background

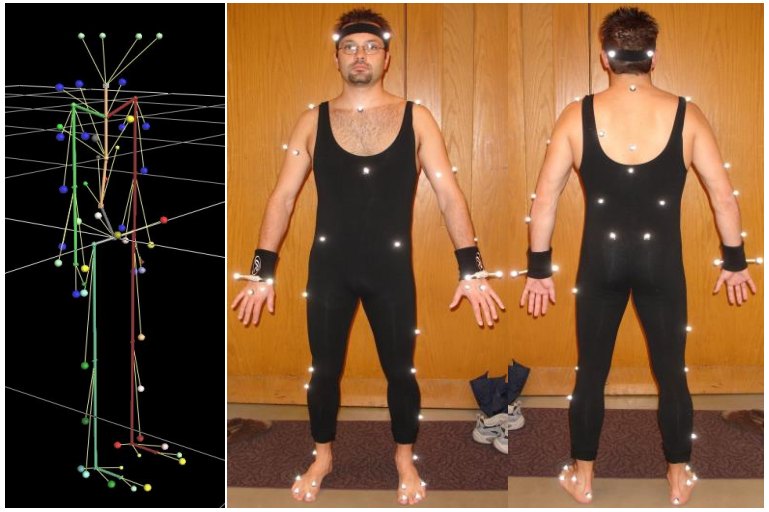
Classify each pixel into a body part with a Random Forest classifier processing depth features

Find local modes to define joint positions

Synthetic annotated data

Generating realistic RGB images to learn poses is made difficult by the huge color and texture variability induced by clothing, hair, and skin. The use of a depth camera significantly reduces the space of variations, but variations in body and clothing shape remains.

To obtain a realistic and varied set of poses, a mixture of real motion capture data (mo-cap data) augmented in a synthetic pipeline is used. Approximately 500k frames in a few hundred sequences of driving, dancing, kicking, running, navigating menus, etc, were captured… Only 100k sufficiently diverse poses were used.



http://mocap.cs.cmu.edu/

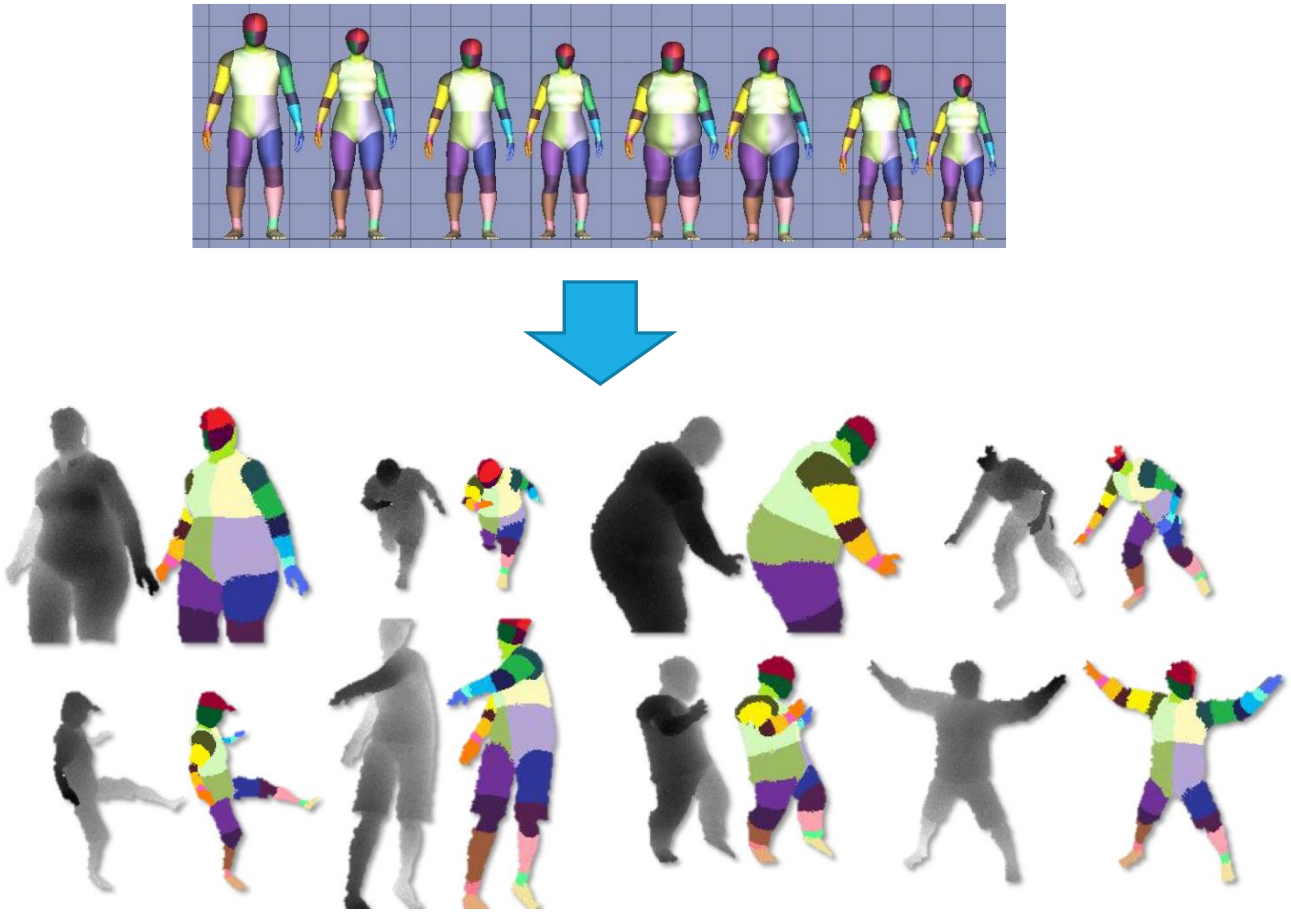
Synthetic annotated data

100k poses do not cover the full space of possible depth images

To learn invariance to camera pose, body pose, and body size and shape, standard computer graphics is used on the skeleton acquired with mocap to render a depth image and its body part labels from 15 meshes spanning a range of body shapes and sizes.

Other randomized parameters include camera pose, camera noise, clothing and hairstyle.

Gathering the right data is key: the authors found it necessary to iterate the process of motion capture, synthetic rendering, training the classifier, and testing joint prediction accuracy in order to refine the mocap database with regions of pose space that had been previously missed out.



Depth comparison features

At each pixel location x, compute simple depth comparison features given depth image D and offsets $\boldsymbol{\theta}$

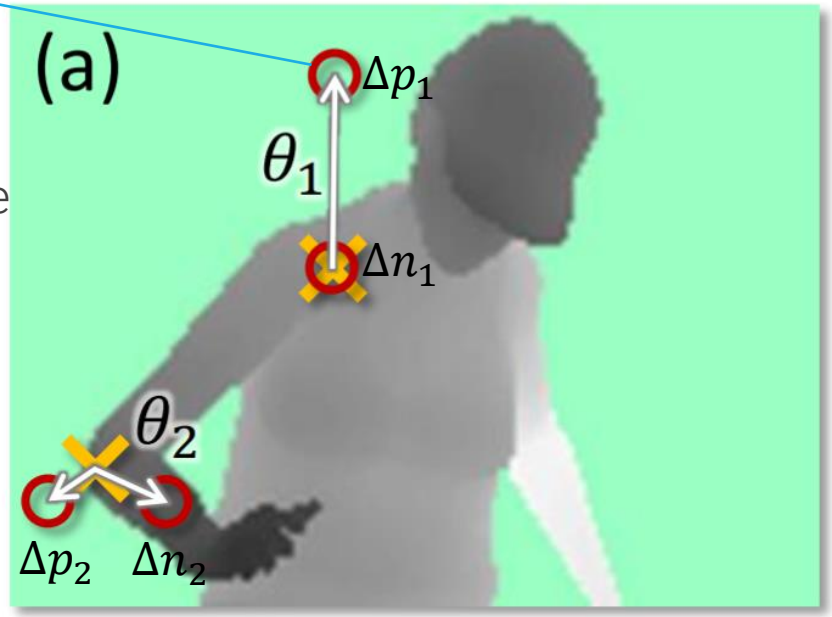
$$f(x; D, \theta = (\Delta p, \Delta n)) = D\left(x + \frac{\Delta p}{D(p)}\right) - D\left(x + \frac{\Delta n}{D(p)}\right)$$

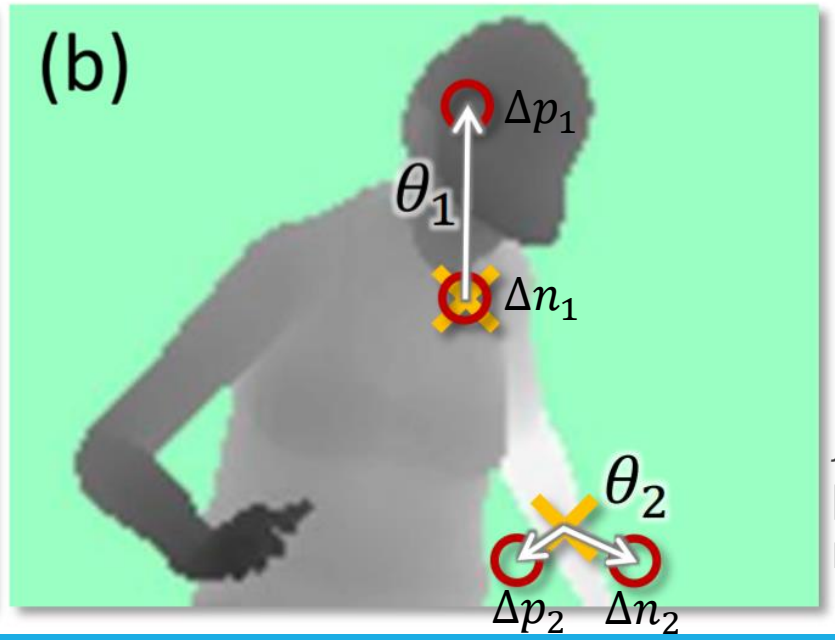
Normalized offsets ensure the feature is invariant to depth (see next slide)

A large positive depth is assigned to background pixels or pixels outside the image

$f(\theta_1)$ helps to locate pixel near the top of the body

$f(\theta_2)$ helps to find thin vertical structures, like arms



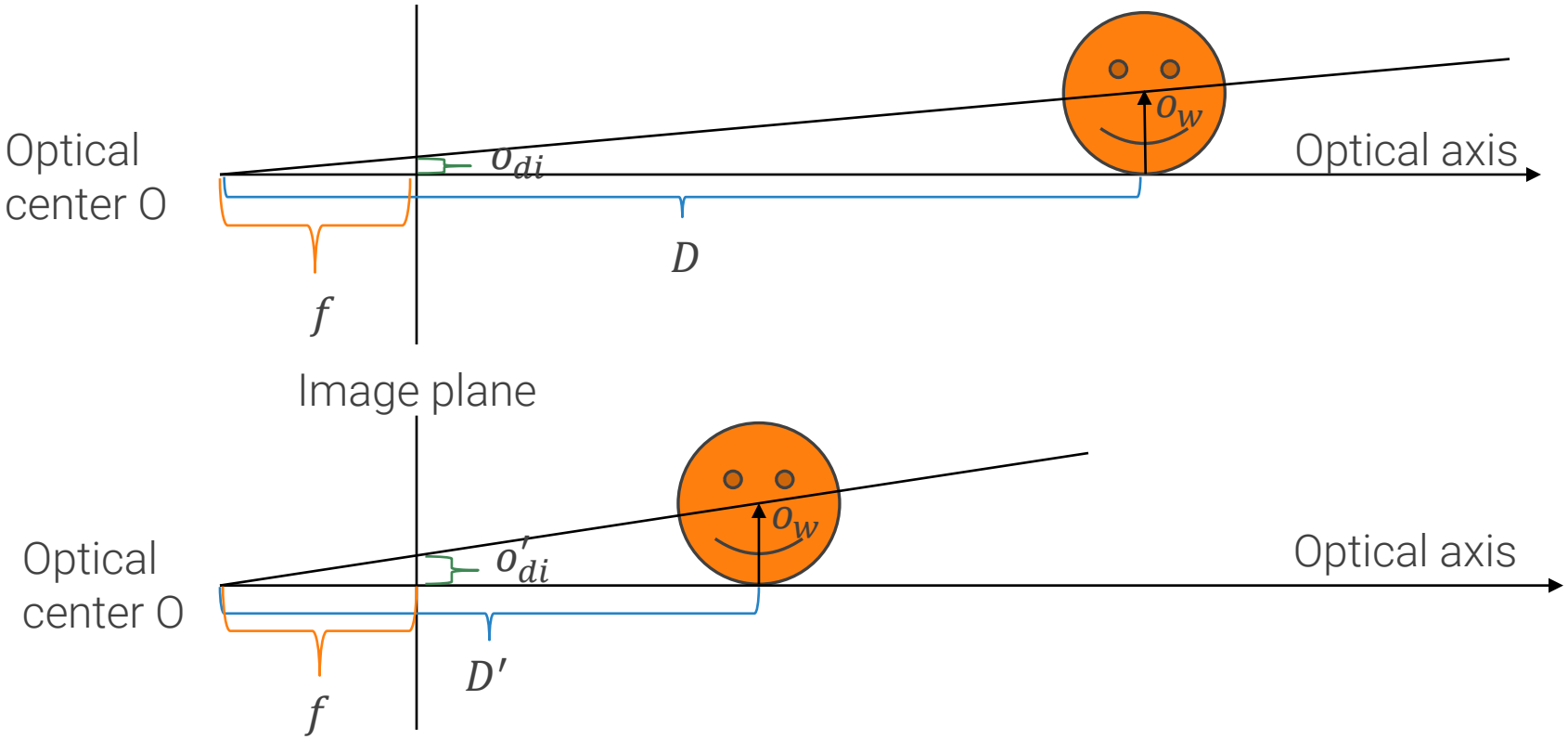


$f(\theta_1)$ does not have a strong response at all locations in the top of the body

$f(\theta_2)$ does not always provide a strong response on arms.

Depth-invariant offsets

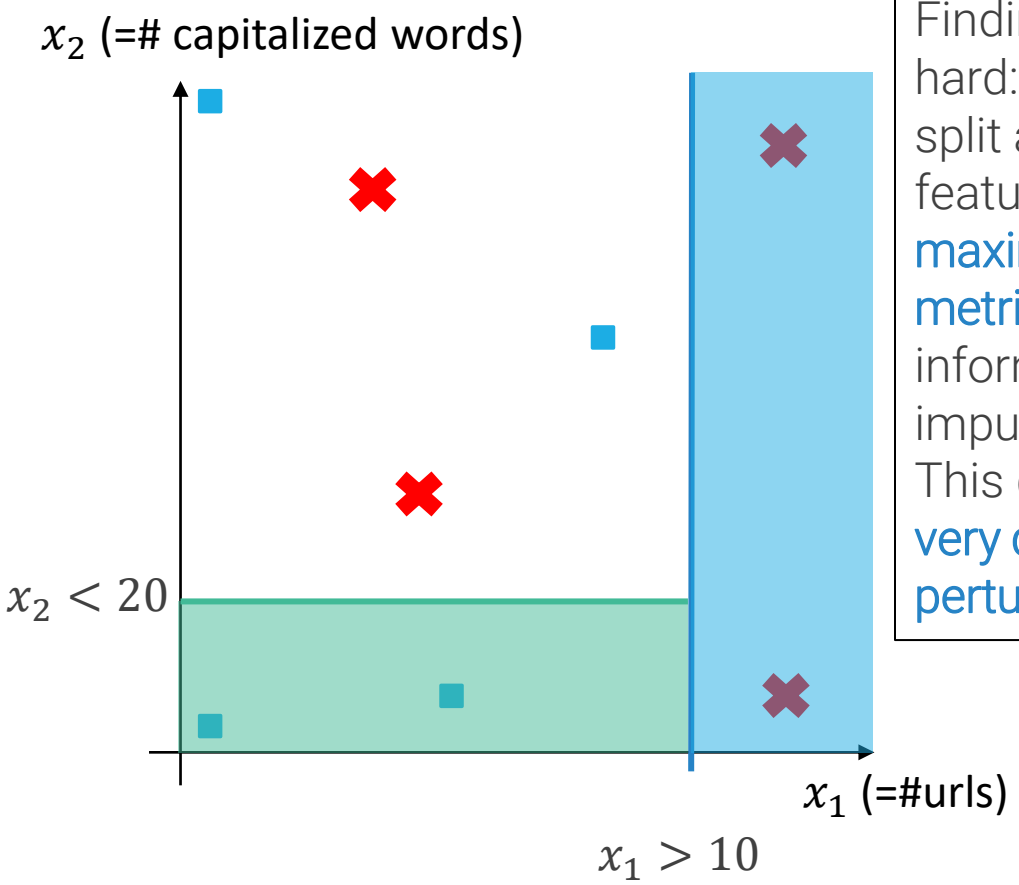
To compute the distance comparison between the chin and the nose the same world offset o_w must be applied. If the face is at different distances in two depth images D and D', the depth-invariant offset to perform the same comparison scales inversely with depth



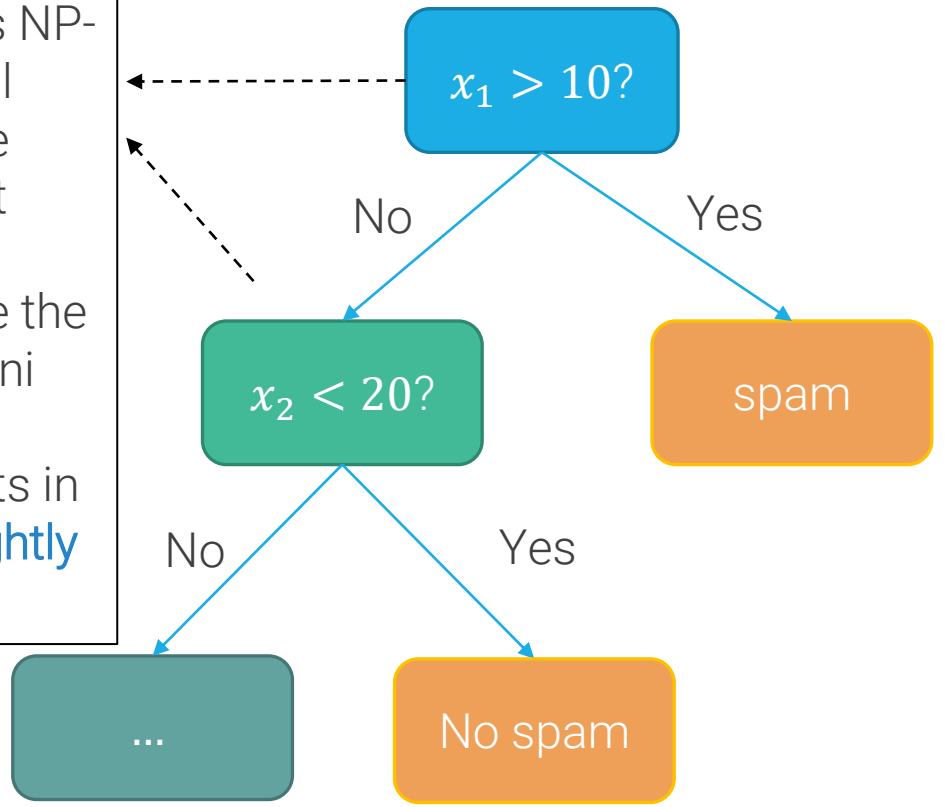
$$o_{di} : f = o_w : D$$
$$\rightarrow o_{di} = \frac{o_w f}{D} = \frac{\Delta p}{D}$$

$$o'_{di} : f = o_w : D'$$
$$\rightarrow o'_{di} = \frac{o_w f}{D'} = \frac{\Delta p}{D'}$$

Decision trees



Finding the optimal tree is NP-hard: to define the optimal split at each node, choose feature and threshold that maximize a homogeneity metric of the split data like the information gain or the Gini impurity.
This greedy training results in very different trees for slightly perturbed training sets



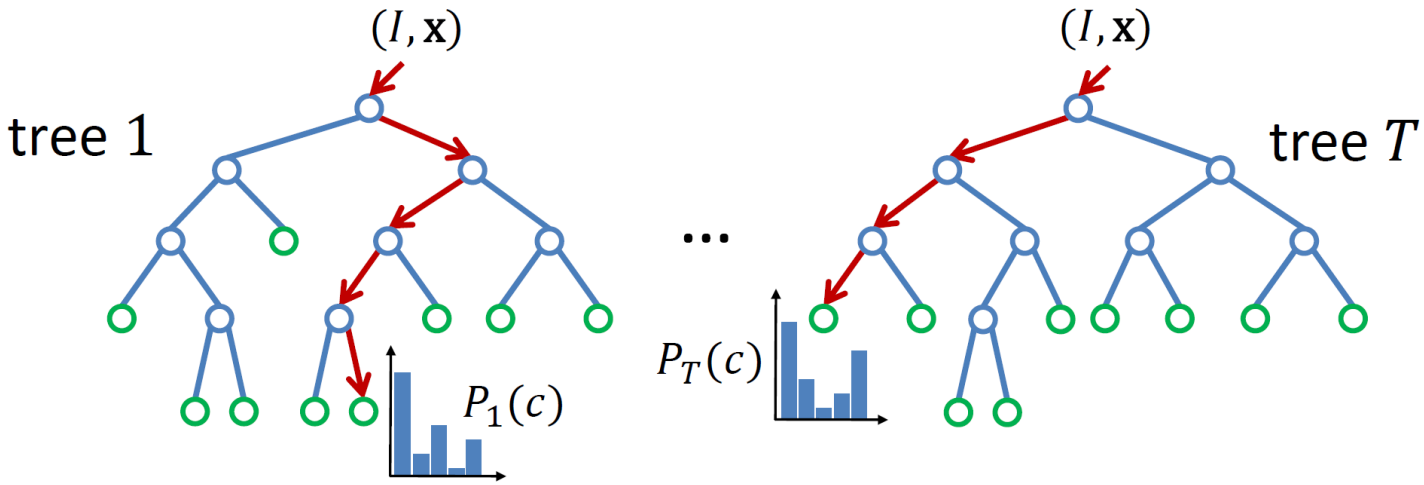
L. Breiman, J. Friedman, R. Olshen, and C. Stone, "Classification and Regression Trees", Wadsworth, Belmont, CA, 1984.
Quinlan, J. R. Induction of Decision Trees. Mach. Learn., 1986.

Random Forests

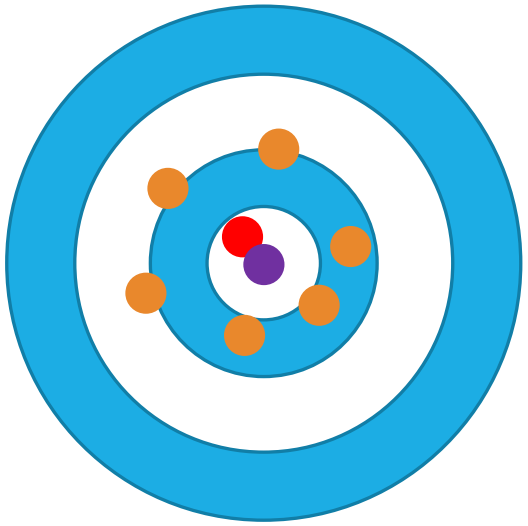
Trees are known to be robust classifiers, which easily handle missing values, categorical and numerical features, diverse ranges, etc.., but generalize poorly: when tuned to reach low bias, they in general suffer from high variance.

Random Forests are another example of ensembles of simple classifiers, in this case decision (or regression) trees, which train N trees on random views of the dataset and average their predictions.

If the trees are uncorrelated, different trees will make different errors: the average of those errors will approach zero as more trees are added to the ensemble, reducing variance.



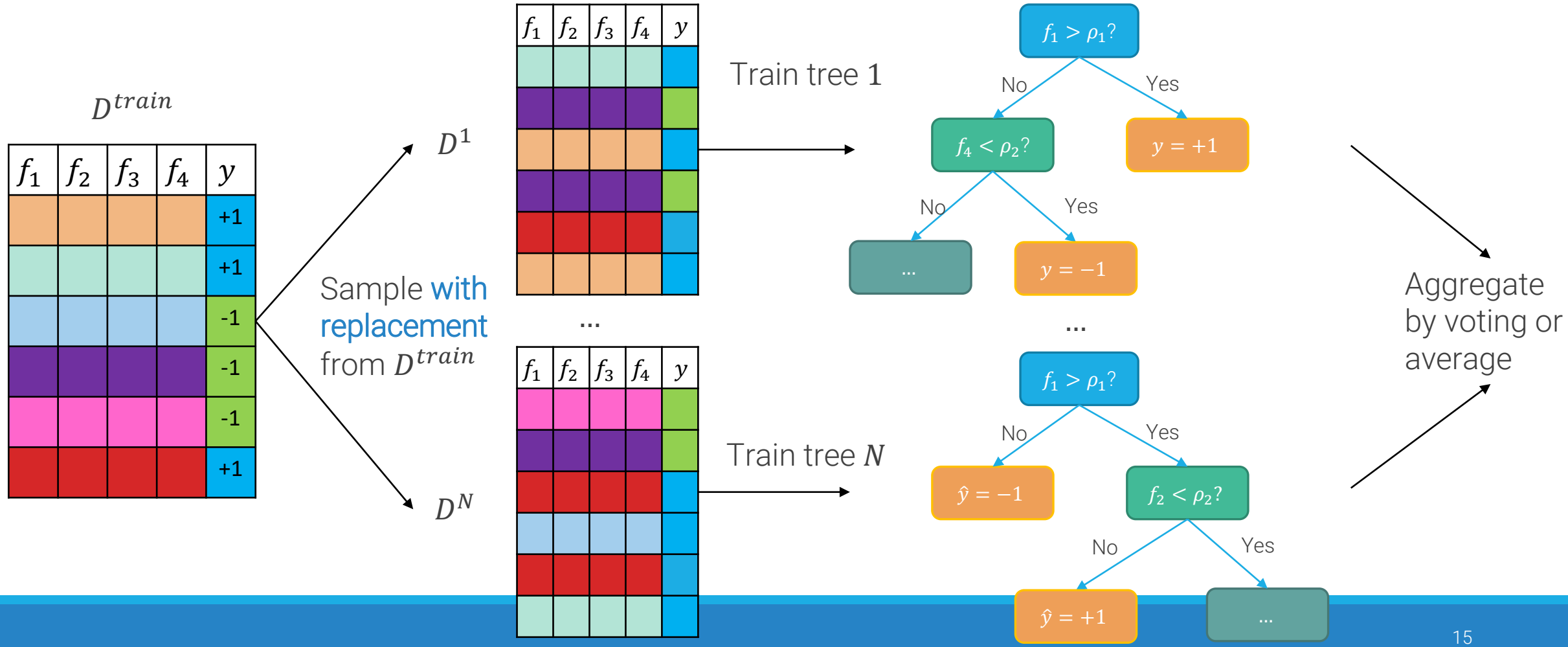
Low bias
High variance



Breiman, L. "Random Forests", Machine Learning, 45(1), 5-32, 2001.

Bagging

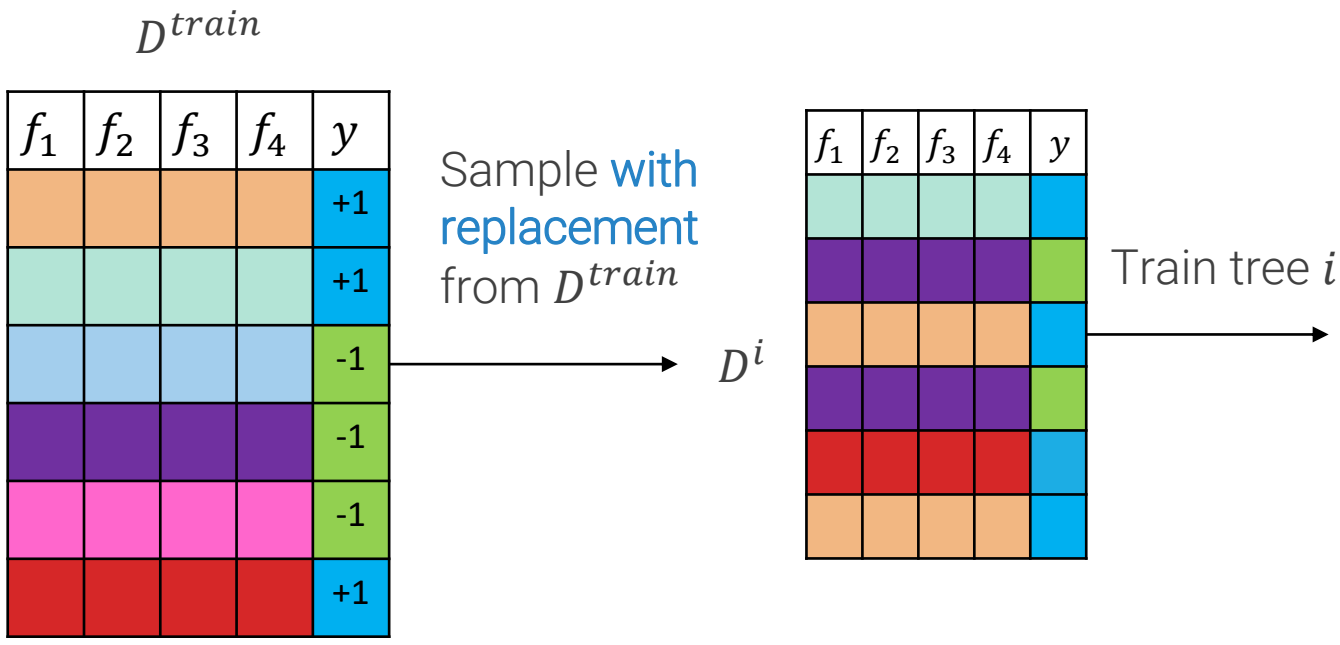
Bagging stands for "Boostrap AGGregatING": aggregating N unstable classifiers (like trees) trained on bootstrapped replicas of the same dataset improves their performance.

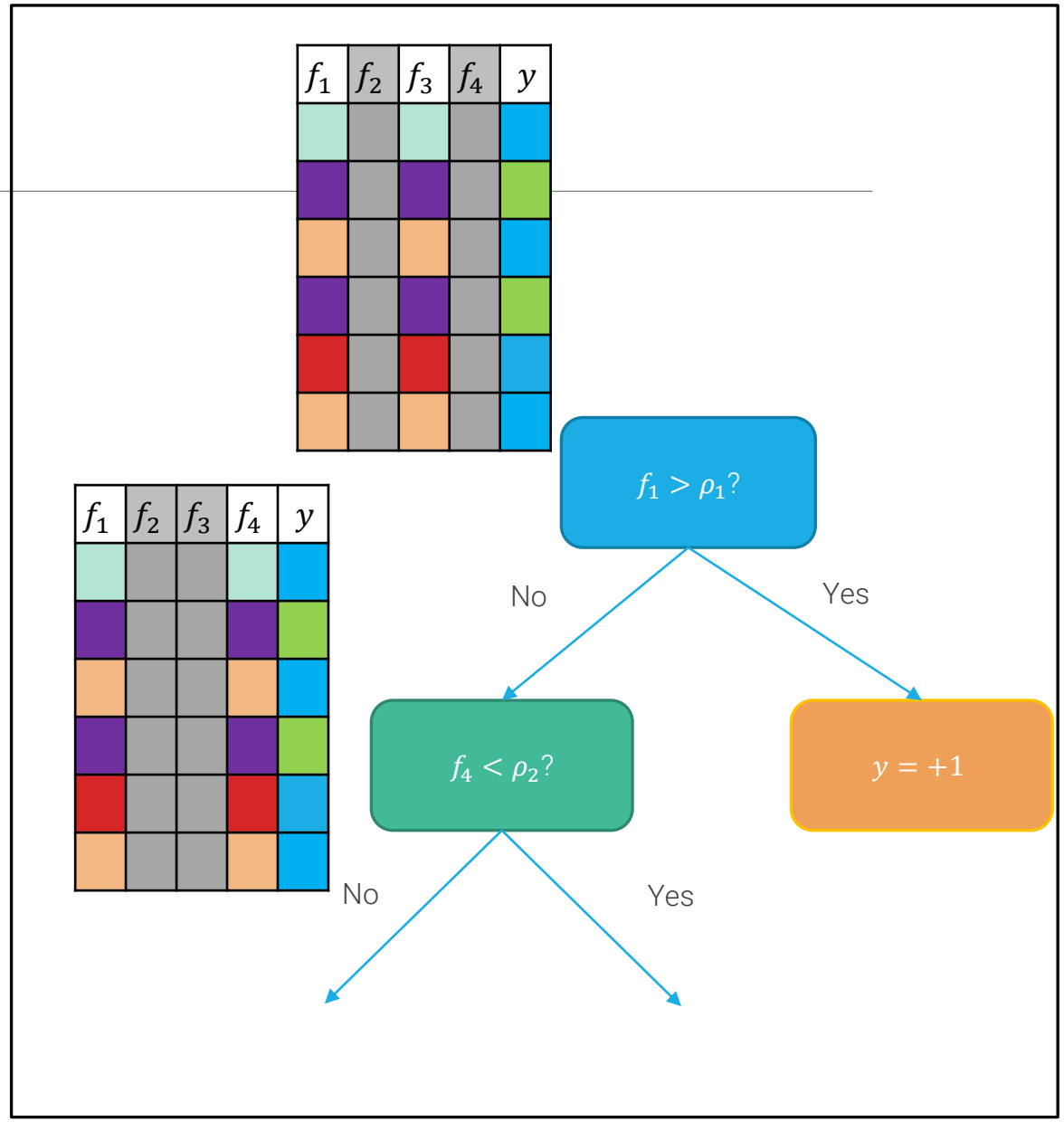


Random forests

Bagging already reduces the variance of the ensemble thanks to bootstrapped datasets. Yet, if a (subset of) feature(s) is particularly predictive, the resulting trees will likely use them and will still be highly correlated.

Random forests reduces the correlation by selecting a random subset of features to define the split at each node.





Breiman, L. "Random Forests", Machine Learning, 45(1), 5-32, 2001.

Random Forests

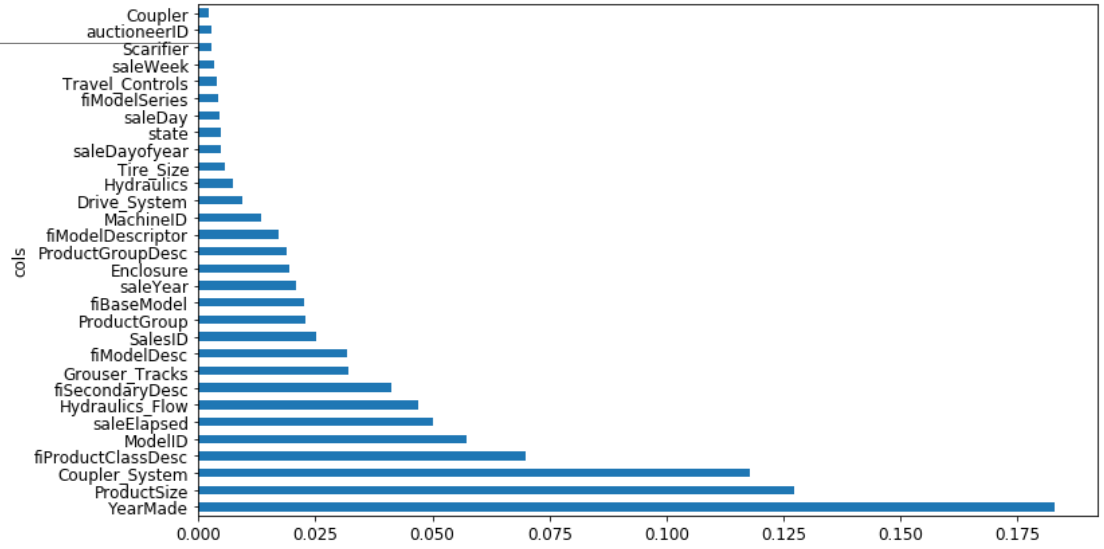
- Fast to train and to test: can be evaluated in parallel in both phases.

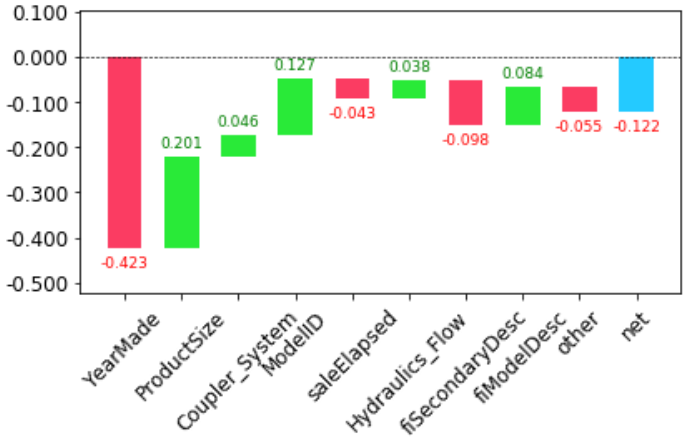
- Variance of predictions across trees provide an easy confidence on estimates

- Model interpretability, e.g. feature importance, feature contribution

- Out-of-bag data provide a free validation set

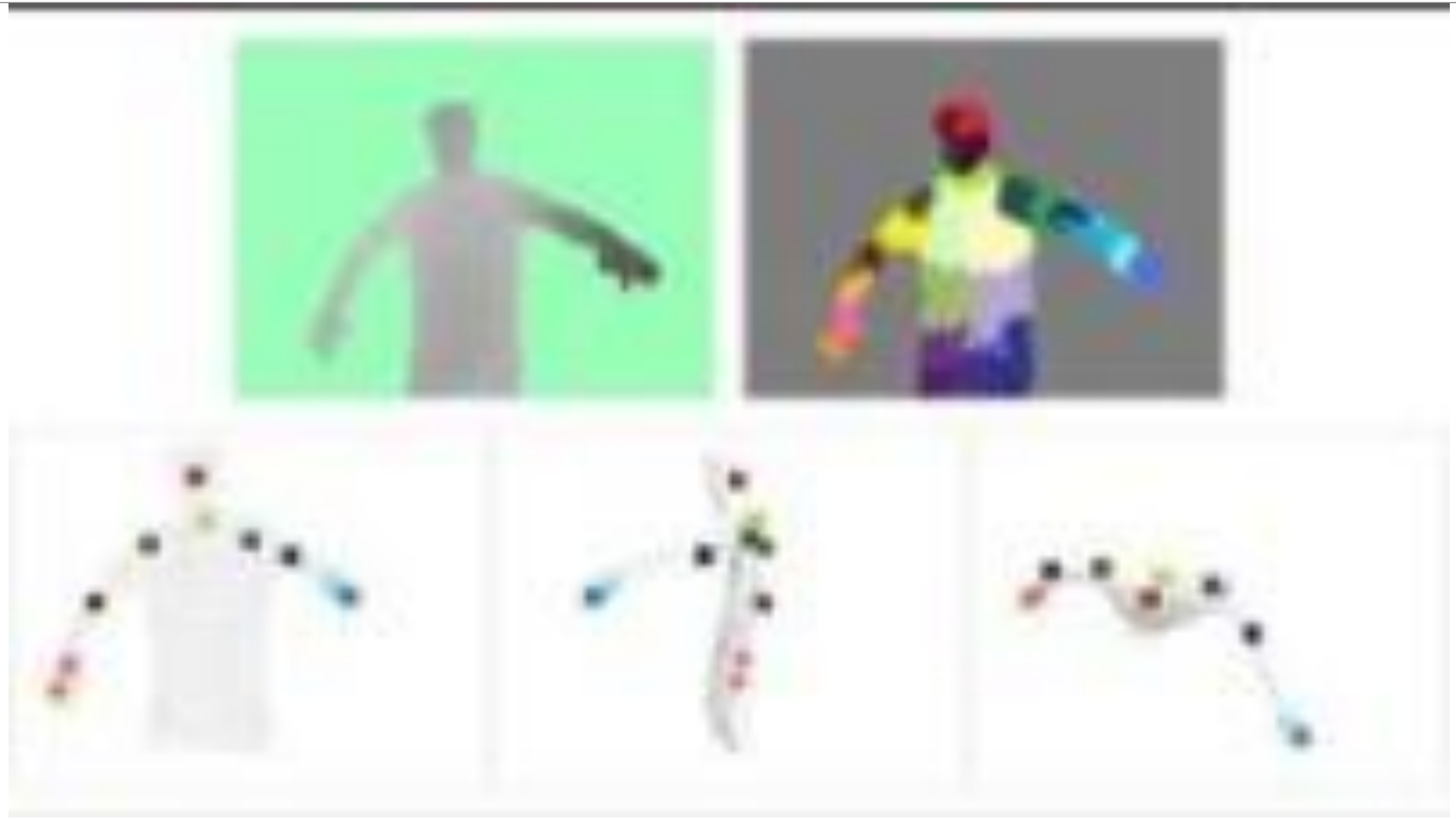
- Robust to a wide range of parameters, relatively easy to tune





https://github.com/fastai/fastbook/blob/master/09_tabular.ipynb

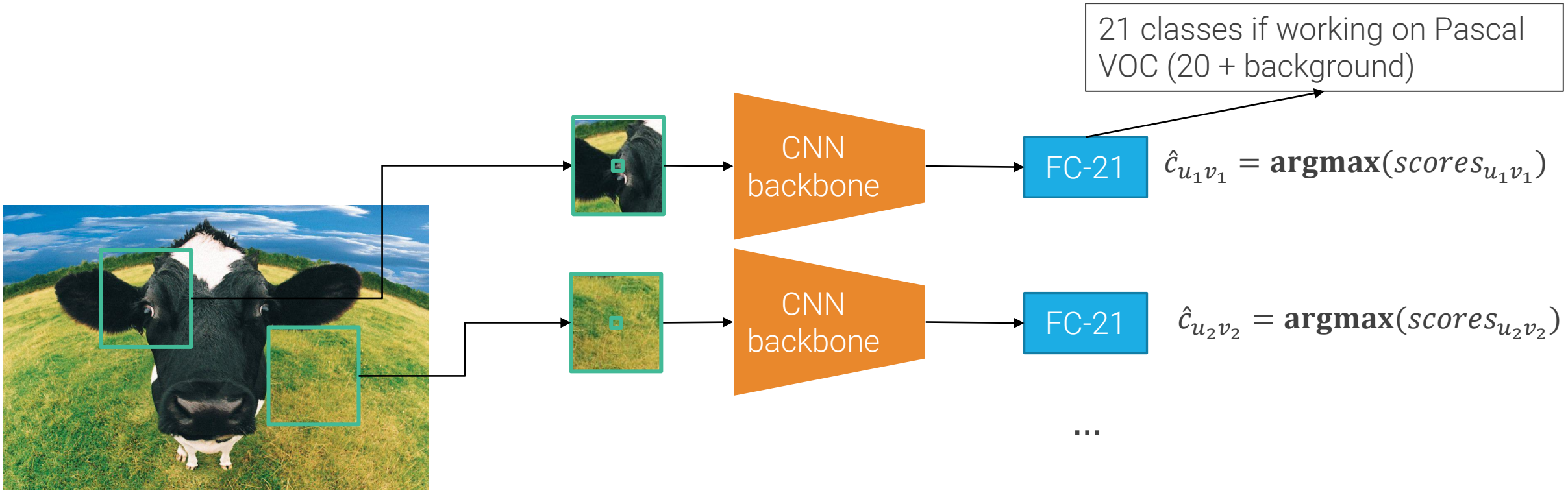
Results



Synthetic data to harvest labels

Stephan R. Richter, Zeeshan Hayder, and Vladlen Koltun, “Playing for Benchmarks”, ICCV 2017

Slow R-CNN for segmentation



Slide window at all possible positions

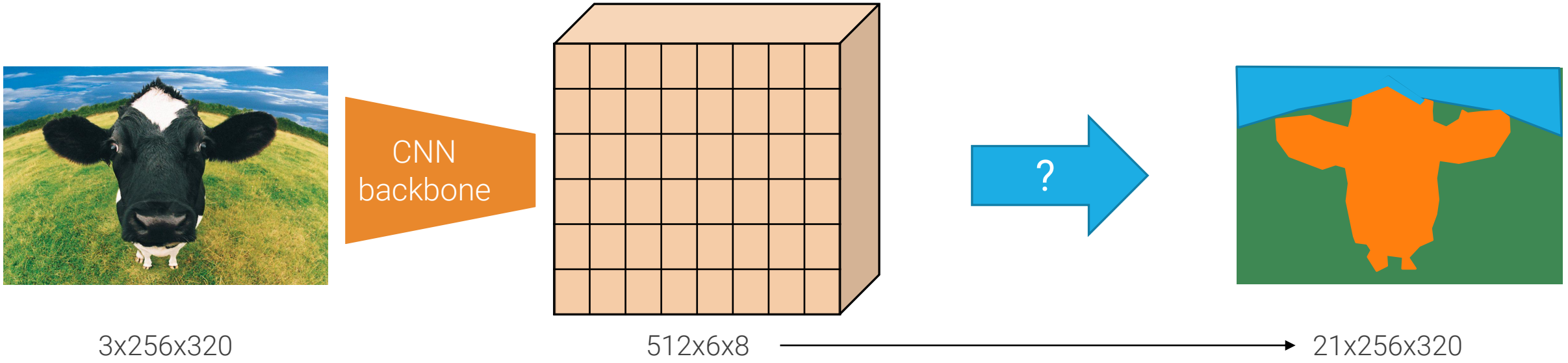
No proposals, must process each pixel, even slower than R-CNN for detection

Loss is the sum of the standard multi-class loss over all pixels

$$L\left(\theta, x^{(i)}, y^{(i)} = c^{(i)}\right) = \sum_{u,v} CE\left(\text{softmax}(scores_{uv}), \mathbb{I}\left(c_{uv}^{(i)}\right)\right)$$

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

Fully Convolutional Network (FCN)

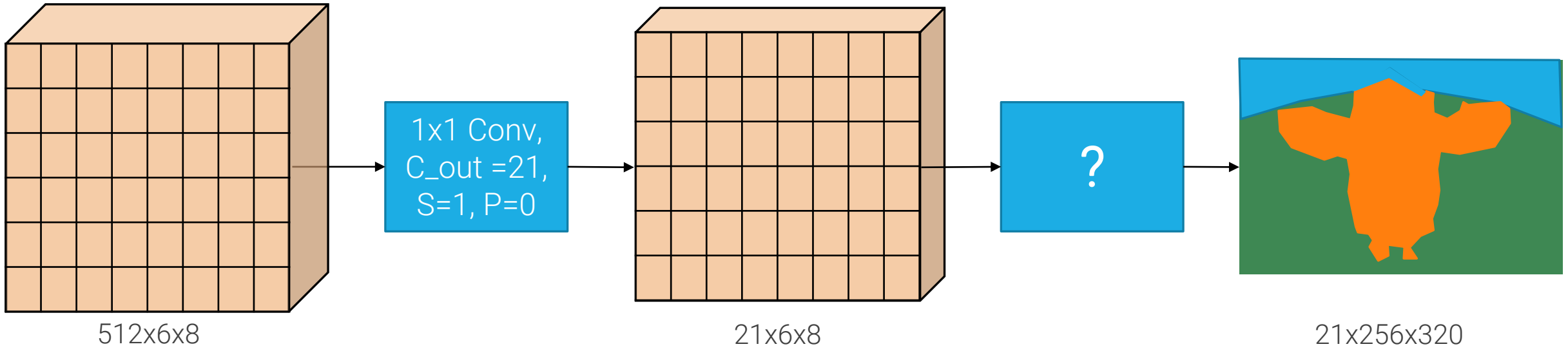


Long et al, “Fully convolutional networks for semantic segmentation”, CVPR 2015.
Evan Shelhamer, et al., “Fully convolutional networks for semantic segmentation”, PAMI 2017.

Fully Convolutional Network (FCN)

Fix channels to be equal to number of classes C

We need to convert coarse spatial class scores into fine grained scores with an upsampling operation



Long et al, "Fully convolutional networks for semantic segmentation", CVPR 2015

Upsampling

One way to perform upsampling can be to use standard, not-learned image processing operators

Input

1	2
3	4

Cx2x2

Nearest Neighbor

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

Cx4x4

Bilinear interpolation

1	1.25	1.75	2
1.50	1.75	2.25	2.5
2.5	2.75	3.25	3.5
3	3.25	3.75	4

Cx4x4

FCN-32s

3xHxW

CNN backbone with total stride 32

Fine-tuned

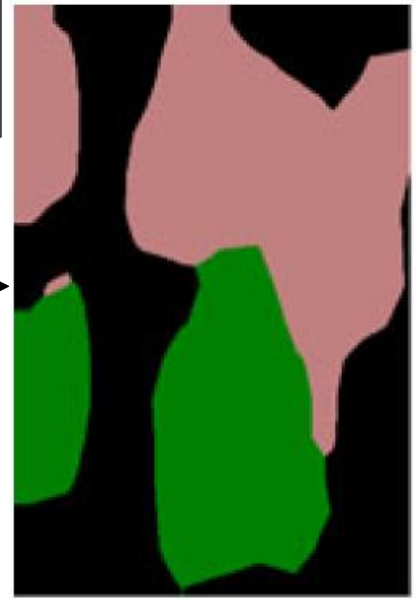
"Scoring" layer in FCN terminology: after this layer, we have a score for each class, but at a resolution that depends on the backbone stride

1x1 Conv, C_out =21, S=1, P=0

512xH/32xW/32

21xH/32xW/32

Bilinear Upsampling x32



21xHxW

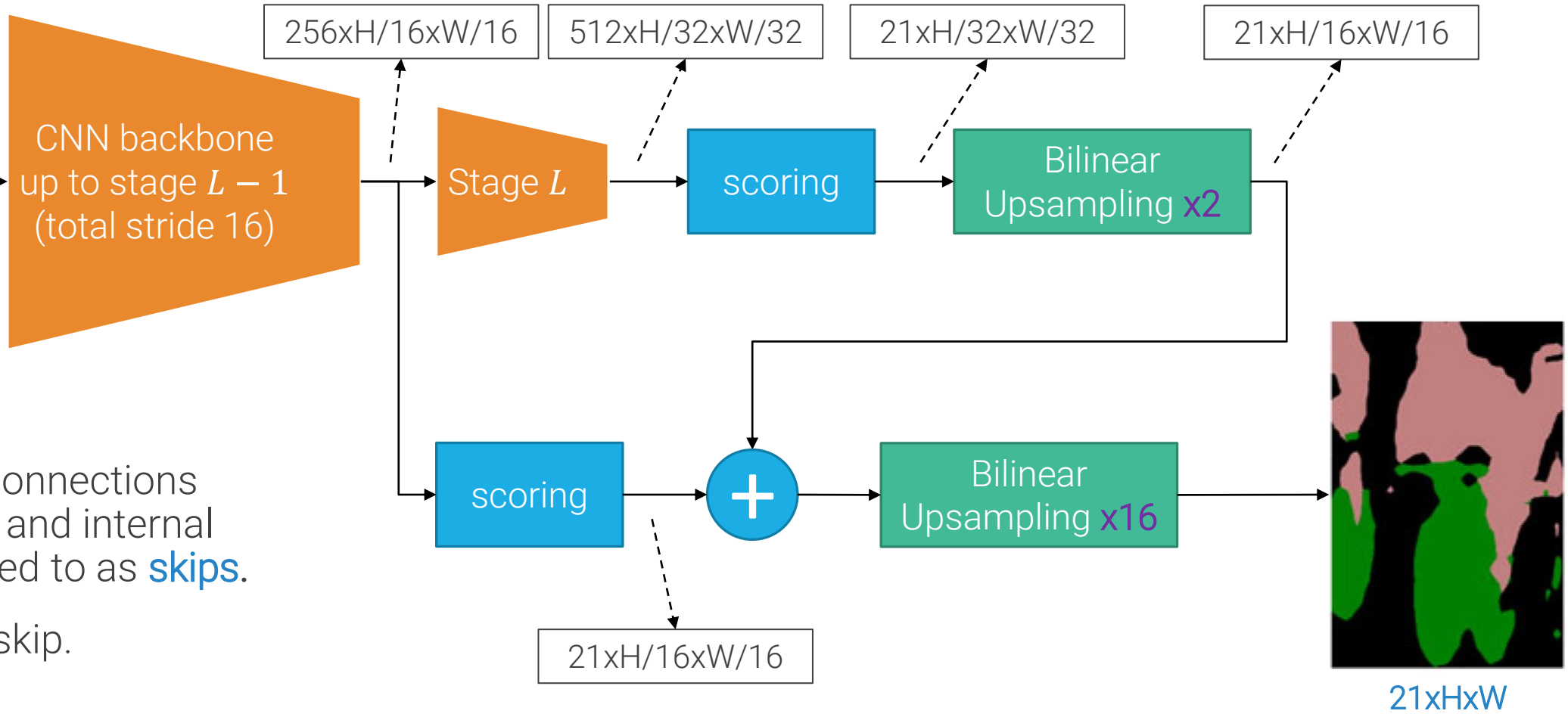
Problem: without learning a non-linear upsampling transformation, we can only uniformly spread the coarse info in the final convolutional activation, obtaining very coarse masks.

Solution: upsample multiple activations at different resolutions

Long et al, "Fully convolutional networks for semantic segmentation", CVPR 2015.
Evan Shelhamer, et al., "Fully convolutional networks for semantic segmentation", PAMI 2017.

FCN-16s

3xHxW



21xHxW

The additional connections between output and internal layers are referred to as skips.

FCN-16s has 1 skip.

Long et al, "Fully convolutional networks for semantic segmentation", CVPR 2015.
Evan Shelhamer, et al., "Fully convolutional networks for semantic segmentation", PAMI 2017.

FCN-8s

3xHxW

CNN backbone up to stage $L-2$ (total stride 8)

128xH/8xW/8

Stage $L-1$

256xH/16xW/16

Stage L

512xH/32xW/32

scoring

21xH/32xW/32

Bilinear Upsampling x2

21xH/16xW/16

scoring

21xH/16xW/16

Bilinear Upsampling x2

21xH/8xW/8

scoring

21xH/8xW/8

Bilinear Upsampling x8

21xHxW

FCN-8s has 2 skips.

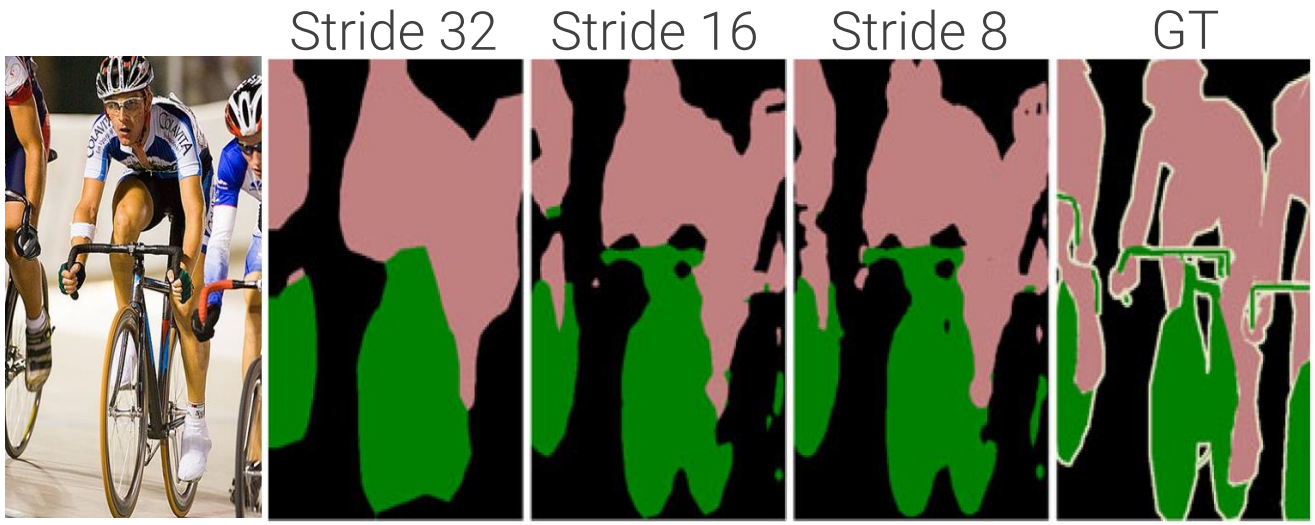
Long et al, "Fully convolutional networks for semantic segmentation", CVPR 2015.
Evan Shelhamer, et al., "Fully convolutional networks for semantic segmentation", PAMI 2017.

Ablation study

With VGG backbone, found basically no improvements after predicting from stride 8 activations (2 skips). Fine-tuning the backbone is very important (fixed row is without fine-tuning)

No difference between training end-to-end ("at-once") or coarse-to-fine (i.e. first train FCN-32s than add skips and fine-tune, "staged" in the table).

Merging different resolutions with skips is also very important: for instance, FCN-pool4 reports the (poor) quality of the predictions from strided 16 activations, if we do not merge them with coarser data

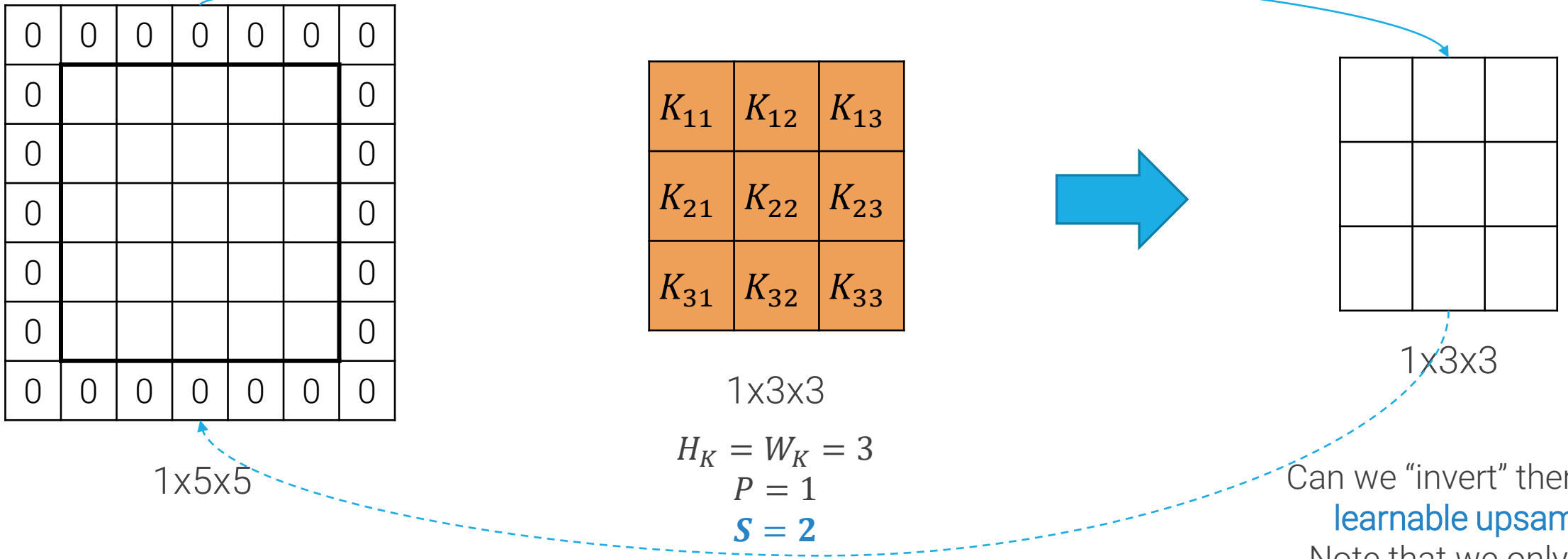


	pixel acc.	mean acc.	mean IU	f.w. IU
FCN-32s	90.5	76.5	63.6	83.5
FCN-16s	91.0	78.1	65.0	84.3
FCN-8s at-once	91.1	**78.5**	65.4	84.4
FCN-8s staged	**91.2**	77.6	**65.5**	**84.5**
FCN-32s fixed	82.9	64.6	46.6	72.3
FCN-pool5	87.4	60.5	50.0	78.5
FCN-pool4	78.7	31.7	22.4	67.0
FCN-pool3	70.9	13.7	9.2	57.6

Long et al, "Fully convolutional networks for semantic segmentation", CVPR 2015.
Evan Shelhamer, et al., "Fully convolutional networks for semantic segmentation", PAMI 2017.

Standard convolution

Convolutions with stride > 1 are a form of learnable downsampling.



Convolution are matrix products

We recall that convolutions are matrix products. For instance, the convolution in the previous slide can be realized as the (unflattened) product of the (flattened) input image with a 9x25 matrix K

K_{11}	K_{12}	K_{13}
K_{21}	K_{22}	K_{23}
K_{31}	K_{32}	K_{33}

1x3x3

$H_K = W_K = 3$

$P = 1$

$\boldsymbol{S = 2}$

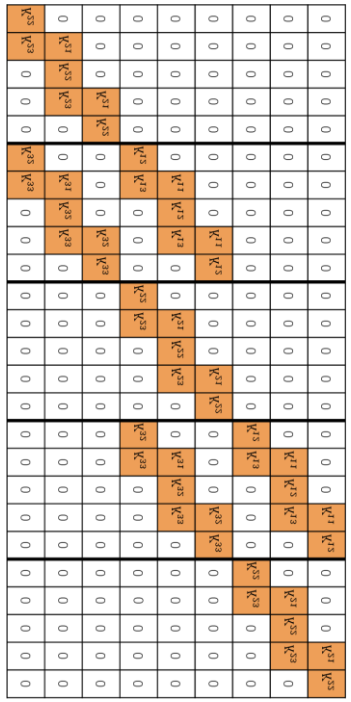
$\approx \quad K =$

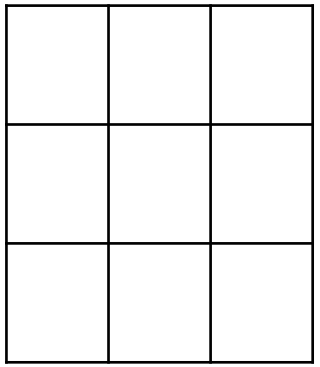
K_{22}	K_{23}	0	0	0	K_{32}	K_{33}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	K_{21}	K_{22}	K_{23}	0	0	K_{31}	K_{32}	K_{33}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	K_{21}	K_{22}	0	0	0	K_{32}	K_{33}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	K_{12}	K_{13}	0	0	0	K_{22}	K_{23}	0	0	0	K_{32}	K_{33}	0	0	0	0	0	0	0	0
0	0	0	0	0	0	K_{11}	K_{12}	K_{13}	0	0	K_{21}	K_{22}	K_{23}	0	0	K_{31}	K_{32}	K_{33}	0	0	0	0	0	0
0	0	0	0	0	0	0	0	K_{11}	K_{12}	0	0	0	K_{21}	K_{22}	0	0	0	K_{32}	K_{33}	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	K_{12}	K_{13}	0	0	0	K_{22}	K_{23}	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	K_{11}	K_{12}	K_{13}	0	0	K_{21}	K_{22}	K_{23}	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	K_{11}	K_{12}	0	0	0	K_{21}	K_{22}

Transposed convolutions

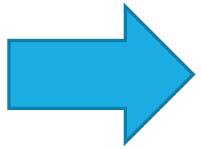
Multiplication by the transposed matrix K^T then realizes a transformation that upsamples feature maps with local connectivity and shared, learnable parameters (like a convolution). This operation is known as transposed convolution.

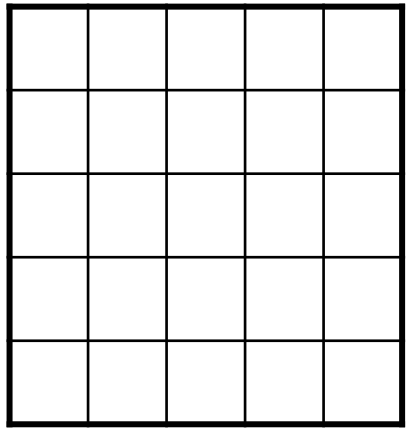
$K^T =$





1x3x3



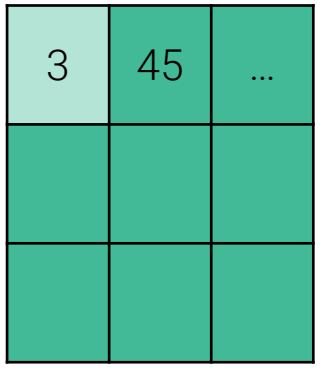


1x5x5

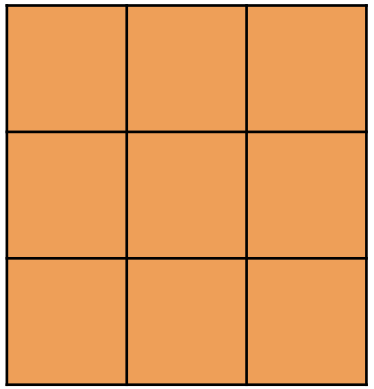
Can we realize this operation without matrix multiplication?
Yes, there always exists an equivalent convolution with fractional stride

Transposed convolutions

Transposed convolutions can be realized without materializing the transposed matrix. A transposed convolution with stride 2 can be realized by moving the kernel on the output by 2 pixels for each pixel in the input and reweighting it by the input pixel.



1x3x3
Original input image

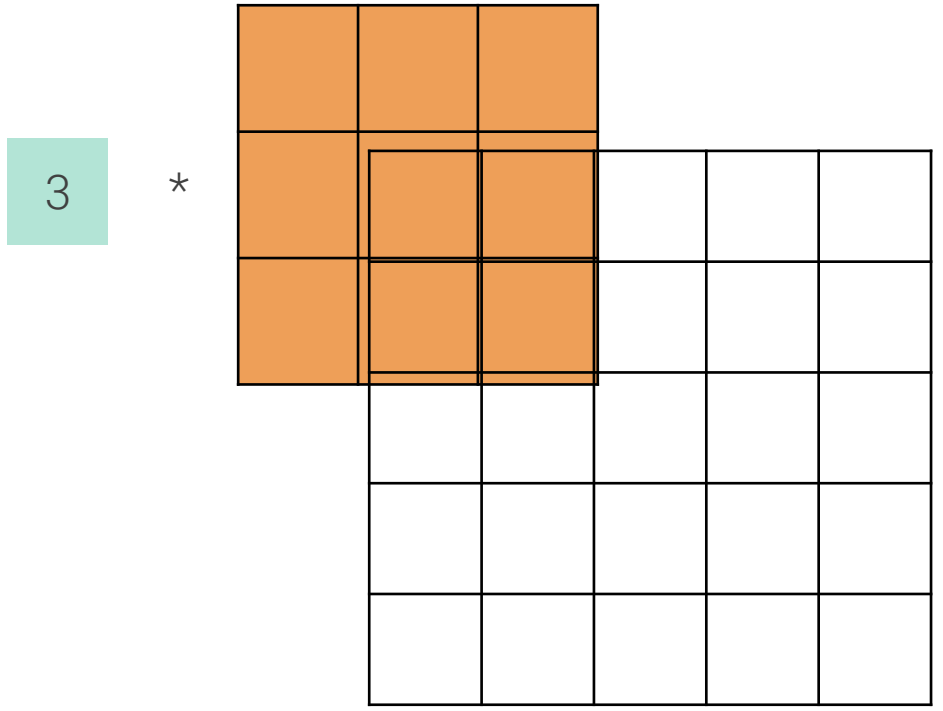


1x3x3

$H_K = W_K = 3$

$P = 1$

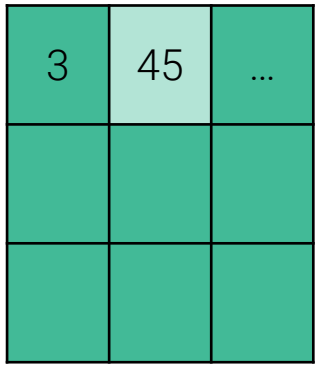
$\boldsymbol{S = 2}$



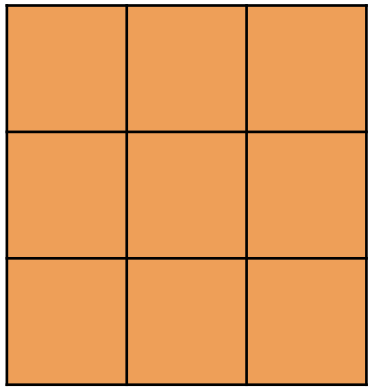
1x5x5

Transposed convolutions

Transposed convolutions can be realized without materializing the transposed matrix. A transposed convolution with stride 2 can be realized by moving the kernel on the output by 2 pixels for each pixel in the input and reweighting it by the input pixel.

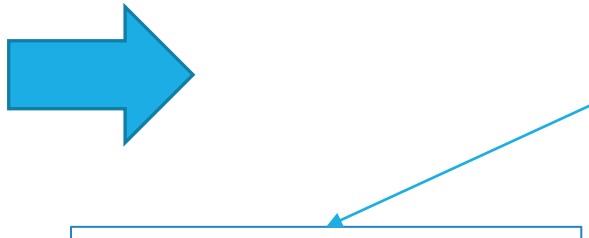


1x3x3
Original input image

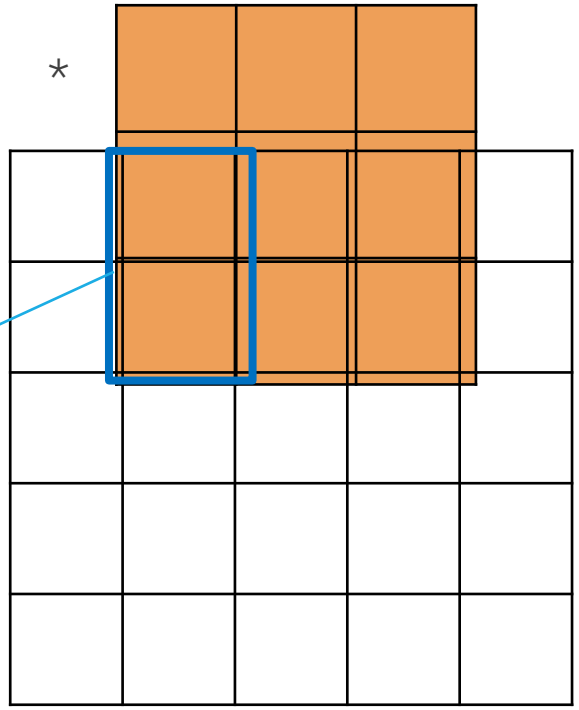


1x3x3

$H_K = W_K = 3$
$P = 1$
$\boldsymbol{S = 2}$



45 *

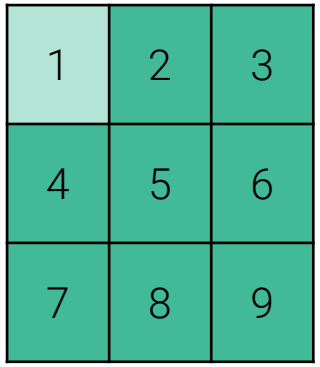


Where reweighted kernels overlap, sum the values to produce output pixel

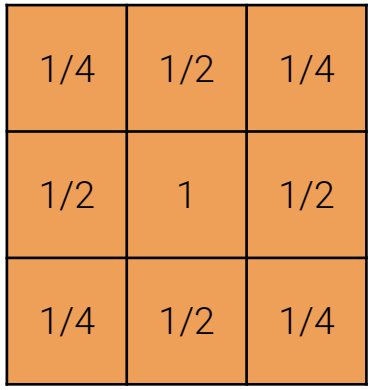
1x5x5

Transposed convolutions

Example with the bilinear interpolation kernel



1x3x3
Original input image

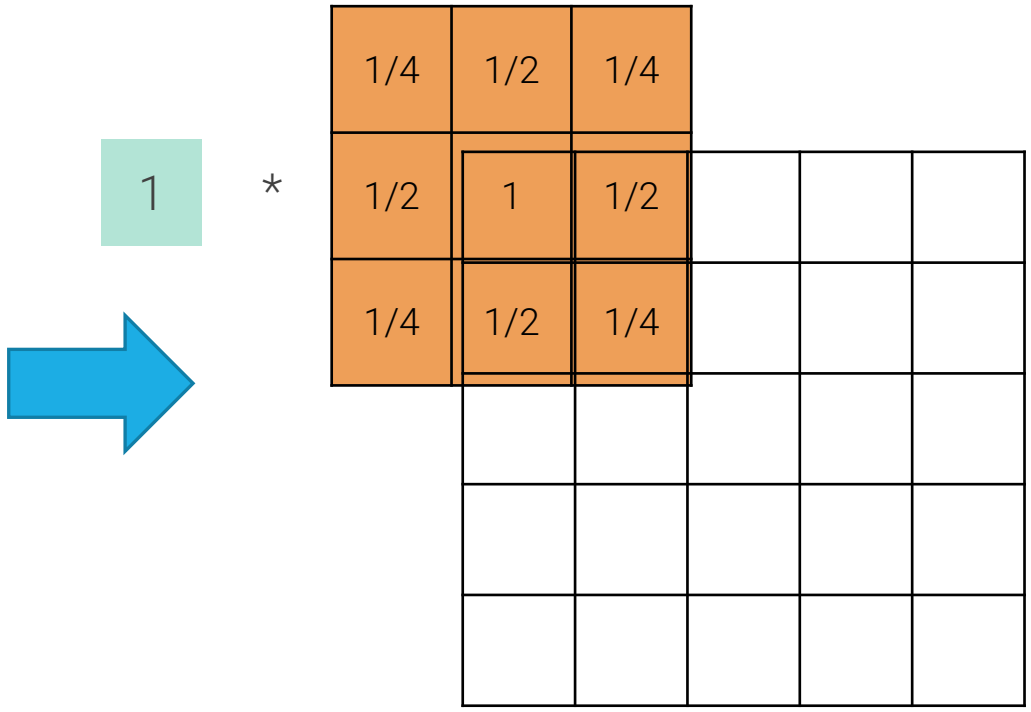


1x3x3

$H_K = W_K = 3$

$P = 1$

$\boldsymbol{S = 2}$



1x5x5

Transposed convolutions

1	2	3
4	5	6
7	8	9

1x3x3
Original input image

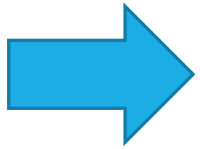
1/4	1/2	1/4
1/2	1	1/2
1/4	1/2	1/4

1x3x3

$H_K = W_K = 3$

$P = 1$

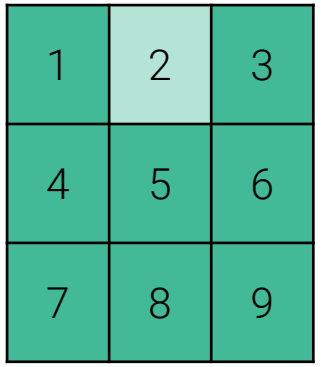
$\boldsymbol{S = 2}$



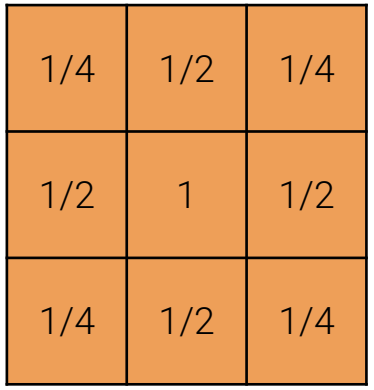
1	1/2			
1/2	1/4			

1x5x5

Transposed convolutions



1x3x3
Original input image



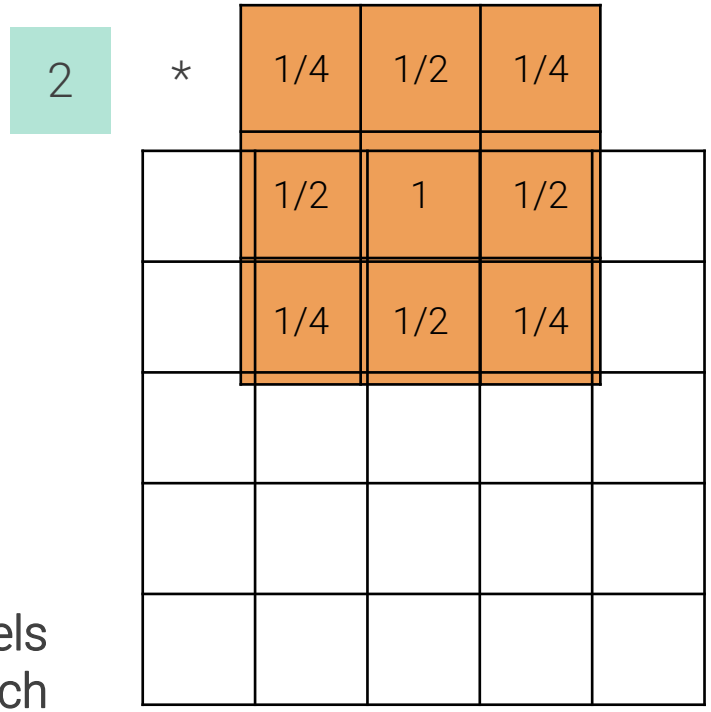
1x3x3

$H_K = W_K = 3$

$P = 1$

$\boldsymbol{S = 2}$

Kernel moves 2 pixels in the output, for each pixel in the input



1x5x5

Transposed convolutions

Highlighted output pixels already reached the final values. By sliding the kernel on the remaining input pixels we can recover the output image

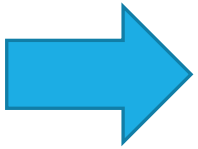
1	2	3
4	5	6
7	8	9

1x3x3
Original input image

1/4	1/2	1/4
1/2	1	1/2
1/4	1/2	1/4

1x3x3

$H_K = W_K = 3$
$P = 1$
$\boldsymbol{S = 2}$



1	1.5	2	1	
1/2	3/4	1	1/2	

1x5x5

Transposed convolutions

This kernel realizes bilinear upsampling as a transposed convolution.
This is how transposed convolutions are initialized

Depending on input/output upsampling ratio, kernel and padding varies, all rules in the usual guide

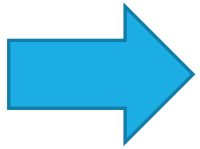
1	2	3
4	5	6
7	8	9

1x3x3
Original input image

1/4	1/2	1/4
1/2	1	1/2
1/4	1/2	1/4

1x3x3

$H_K = W_K = 3$
$P = 1$
$\boldsymbol{S = 2}$



1	1.5	2	2.5	3
2.5	3	3.5	4	4.5
4	4.5	5	5.5	6
5.5	6	6.5	7	7.5
7	7.5	8	8.5	9

1x5x5

https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md

Transposed convolutions in Pytorch

CONVTRANSPOSE2D

CLASS `torch.nn.ConvTranspose2d(`*`in_channels: int, out_channels: int, kernel_size: Union[T, Tuple[T, T]], stride: Union[T, Tuple[T, T]] = 1, padding: Union[T, Tuple[T, T]] = 0, output_padding: Union[T, Tuple[T, T]] = 0, groups: int = 1, bias: bool = True, dilation: int = 1, padding_mode: str = 'zeros'`*`)`

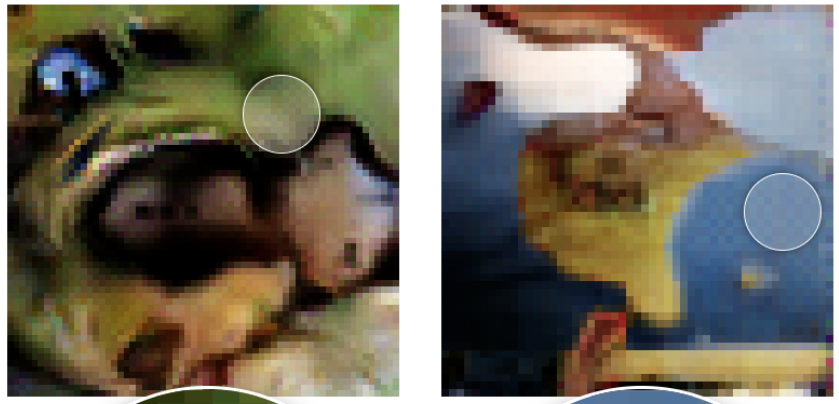
[SOURCE]

Transposed convolutions

Transposed convolutions are referred to with other names:

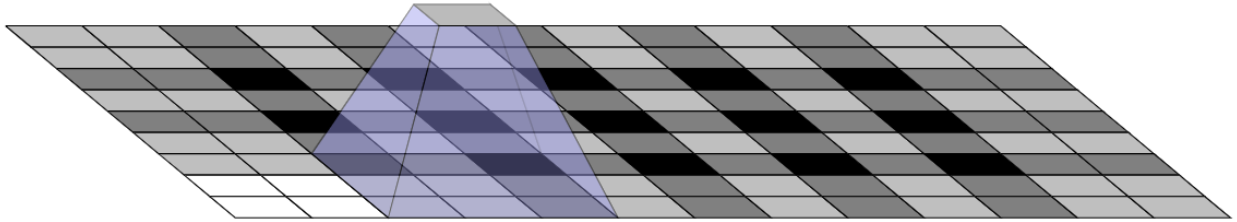
- Deconvolutions: bad choice, deconvolution is defined as the inverse of a convolution, while transposed convolutions only invert shapes
- Upconvolutions
- Fractionally strided convolutions
- Backward strided convolutions

Their use may generate checkerboard artifacts in the output for generative tasks.



Donahue, et al., 2016 [3]

Dumoulin, et al., 2016 [4]



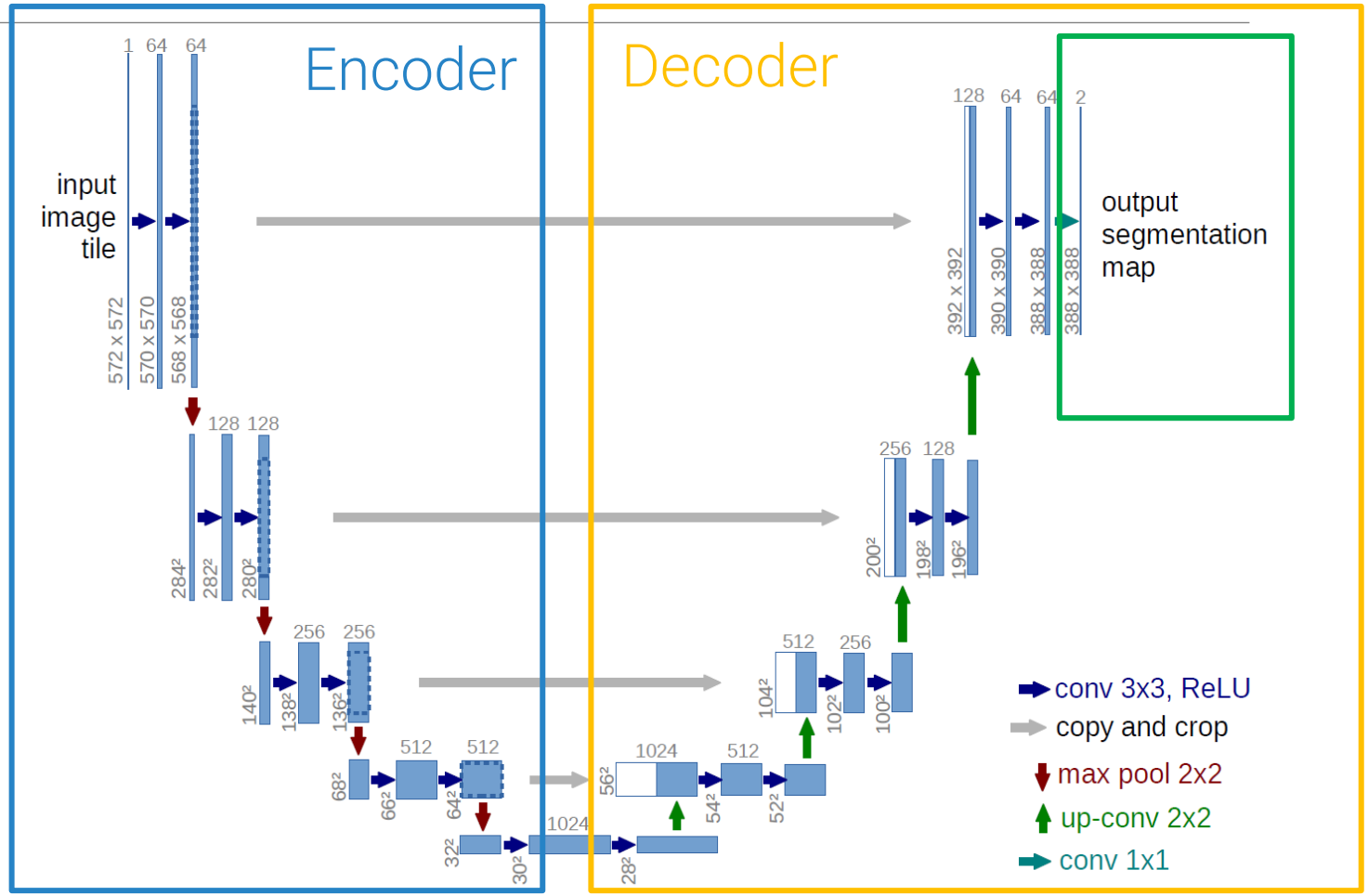
https://distill.pub/2016/deconv-checkerboard/

U-Net

Extends the idea of skips from FCNs to create a full-fledged decoder, which has roughly a symmetric structure with respect to the encoder.

Every activation produced by a stage of the backbone (or encoder, or "contracting path") has a skip connection with the corresponding level of the decoder (or "expansive path").

Scoring layer only at the end to project onto the desired number of classes



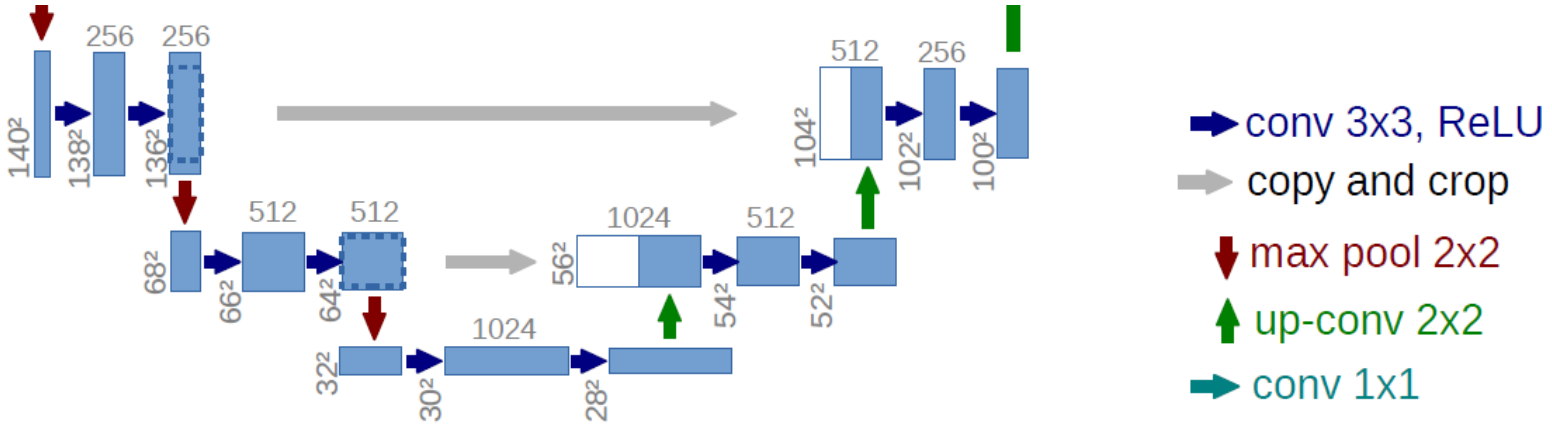
Ronneberger et al., "U-Net: Convolutional Networks for Biomedical Image Segmentation", MICCAI 2015

U-Net

Skip connections use concatenation instead of summation as in FCN.

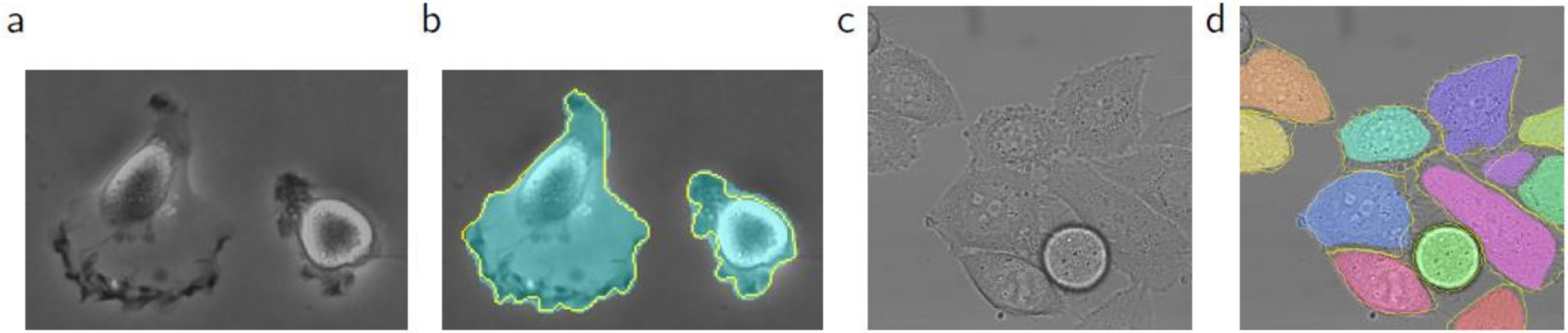
2x2 stride-2 transposed convolutions ("up-convolutions") are used to upsample the activations in the decoder, while halving the number of channels.

Normal 3x3 convolutions are used in the decoder as well: with further processing, even initial layers of the backbone can effectively contribute to the final segmentation mask, as opposed to what happened in FCN.

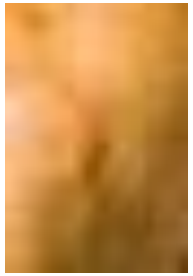


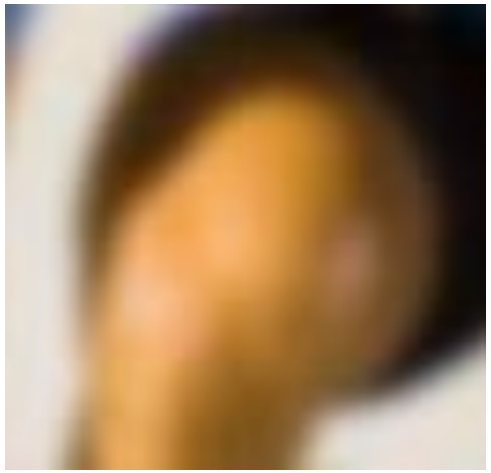
Ronneberger et al., "U-Net: Convolutional Networks for Biomedical Image Segmentation", MICCAI 2015

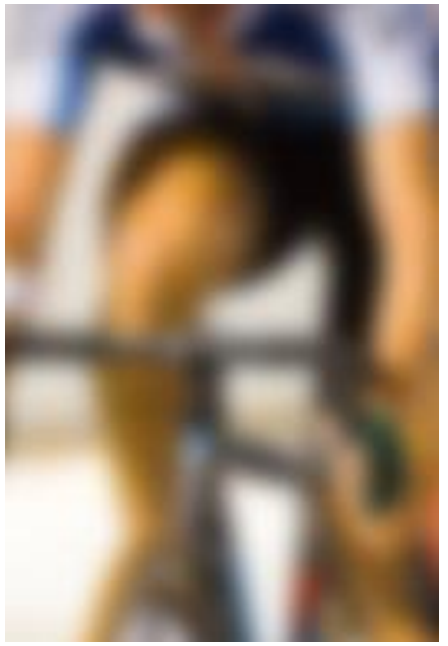
Unet - results



“What” versus “where”







Dilated convolutions

3xHxW

CNN backbone up to stage $L-1$ (total stride 16)

Stage L (stride 32)

Predicting from this activation has rich semantic info due to depth of processing and large receptive field but is spatially coarse

Computing segmentation mask from this activation has detailed resolution but smaller receptive field and not enough semantic info (compare results of FCN pool4)

Is the architecture inherited by classification backbones the best one for semantic segmentation (or dense tasks in general)?

We would like to have rich features with large spatial resolutions, large receptive fields but constant cost.

Liang-Chieh Chen et al., DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs, PAMI 2017
Fisher Yu, Vladlen Koltun., "Multi-scale context aggregation by dilated convolutions", ICLR 2016

Dilated convolutions

Dilated (or «atrous») convolutions expose an additional hyperparameter, the dilation rate $\boldsymbol{r}$. Equivalent to inserting holes ('trous' in French) between filter weights. $r = 1$ gives the usual, dense, convolution.

$$[K * I](j,i) = \sum_{n=1}^{3} \sum_{m} \sum_{l} K_n(m,l) I_n(j - \boldsymbol{r} \cdot m, i - \boldsymbol{r} \cdot l) + b$$

K_{11}	K_{12}	K_{13}
K_{21}	K_{22}	K_{23}
K_{31}	K_{32}	K_{33}

3x3 kernel
$r = 1$

K_{11}	0	K_{12}	0	K_{13}
0	0	0	0	0
K_{21}	0	K_{22}	0	K_{23}
0	0	0	0	0
K_{31}	0	K_{32}	0	K_{33}

3x3 kernel
$r = 2$

K_{11}	0	0	0	K_{12}	0	0	0	K_{13}
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
K_{21}	0	0	0	K_{22}	0	0	0	K_{23}
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
K_{31}	0	0	0	K_{32}	0	0	0	K_{33}

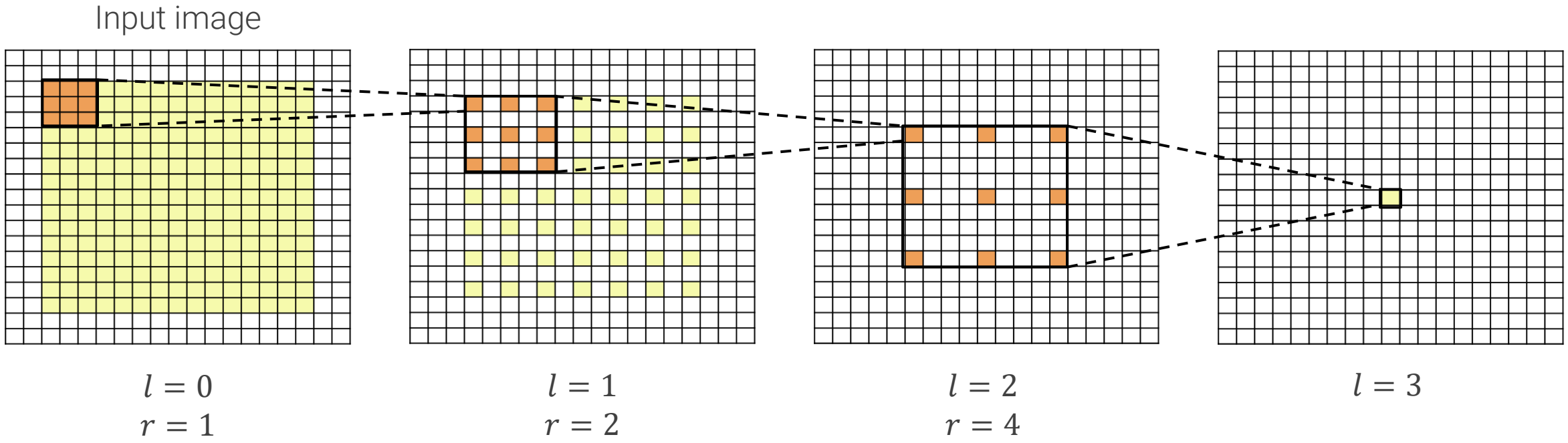
3x3 kernel
$r = 4$

Dilated convolutions

If we stack dilated convolutions with exponentially increasing dilation rate $r_l = 2^l$, the effective receptive field grows exponentially with the number of layers, while the number of parameters grows linearly, and resolution is not reduced.

In general, at level l the receptive field of an activation entry will be $(2^{l+1} - 1) \times (2^{l+1} - 1)$.

For instance, at level 3 below, the receptive field is 15x15, while with dense 3x3 convolutions it would have been 7x7.



Resnet

3xHxW

ResNet backbone up to stage 3 (total stride 16)

Stage 4

Bottleneck Residual block Stride=2

Bottleneck Residual block Stride=1

Bottleneck Residual block Stride=1

We apply changes of stride or rate to this layer

Conv2D layer 1x1, C kernels + BN

ReLU

Conv2D layer 3x3, C kernels + BN

ReLU

Conv2D layer 1x1, 4C kernels + BN

+

ReLU

Liang-Chieh Chen et al., DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs, PAMI 2017

Resnet with dilated convolutions – total stride 16

3xHxW

ResNet backbone up to stage 3 (total stride 16)

Stage 4

Bottleneck Residual block Stride=2

Bottleneck Residual block Stride=1

Bottleneck Residual block Stride=1

512xH/32xW/32

ResNet backbone up to stage 3 (total stride 16)

Stage 4 – r=2

Bottleneck Residual block Stride=1 Rate=1

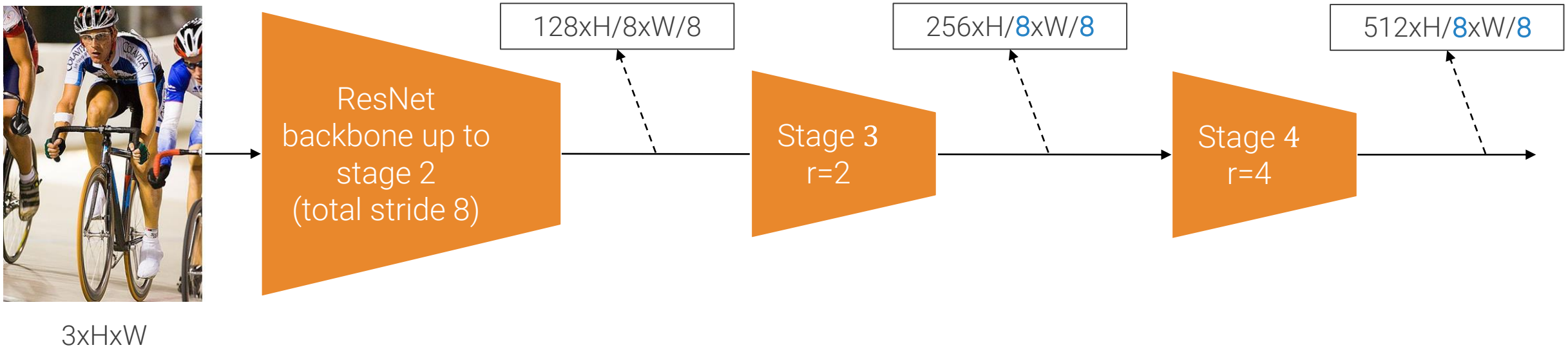
Bottleneck Residual block Stride=1 Rate=2

Bottleneck Residual block Stride=1 Rate=2

512xH/16xW/16

Liang-Chieh Chen et al., DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs, PAMI 2017

Resnet with dilated convolutions – total stride 8



In general, to create a dilated backbone, when we remove the stride from a convolution or pooling layer that was halving the spatial resolution of the activations, we double the dilation rate in all subsequent convolutions

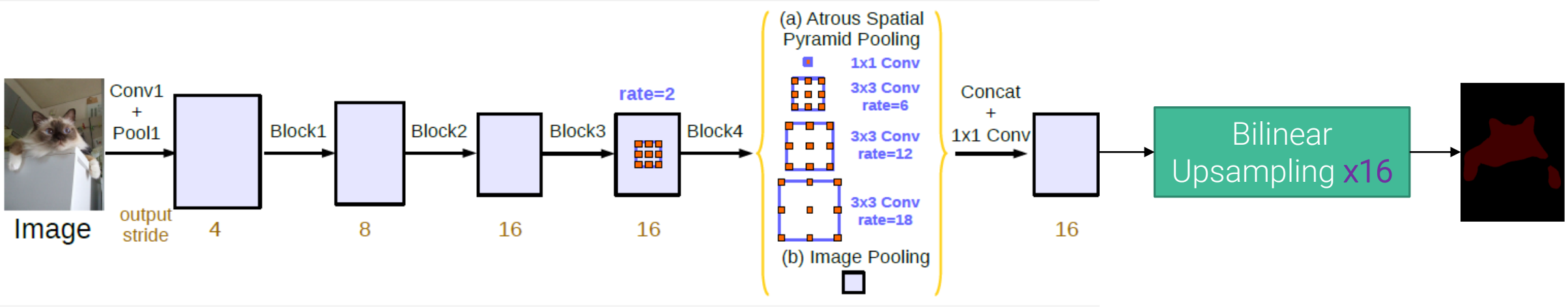
By iterating this process, we could in principle process images at full resolution: however, it becomes too computationally expensive due to the large resolution of the activations. Since initial stages do not improve significantly the quality of the segmentation (as noted in FCN) total stride 16 or 8 is usually used in practice.

Liang-Chieh Chen et al., DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs, PAMI 2017

DeepLab v3

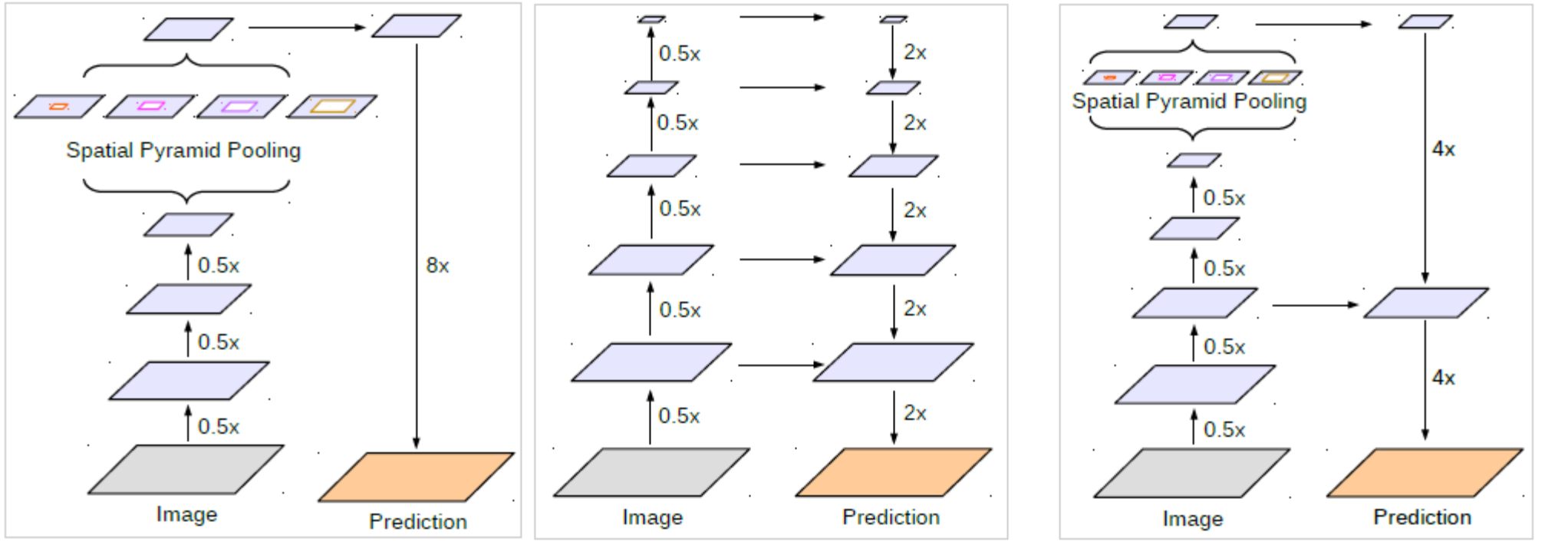
ResNet with dilated convolutions are the backbones used in DeepLab, a reference model for semantic segmentation, to control the resolution of the output feature map. Here is the architecture for DeepLab v3 as an example. Both output stride 16 and 8 are used, e.g. training starts with 16 (faster, larger batches) and then fine-tunes with 8 (more detailed output).

The second main problem in semantic segmentation is the existence of objects at multiple scales. To handle this problem DeepLab uses again atrous convolutions in the "decoder", i.e. the Atrous Spatial Pyramid Pooling (ASPP) module, which is the second major innovation of this architecture.



Liang-Chieh Chen et al., "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs", PAMI 2017
Liang-Chieh Chen et al. , "Rethinking Atrous Convolution for Semantic Image Segmentation", arXiv 2017

DeepLab V3+



(a) Spatial Pyramid Pooling (b) Encoder-Decoder (c) Encoder-Decoder with Atrous Conv

DeepLab V3 + "U-Net" ≈ DeepLab V3+

Liang-Chieh Chen et al., "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation", ECCV 2018.

Results and decoder effect

Method	mIOU
Deep Layer Cascade (LC) [82]	82.7
TuSimple [77]	83.1
Large_Kernel_Matters [60]	83.6
Multipath-RefineNet [58]	84.2
ResNet-38_MS_COCO [83]	84.9
PSPNet [24]	85.4
IDW-CNN [84]	86.3
CASIA_IVA_SDN [63]	86.6
DIS [85]	86.8
DeepLabv3 [23]	85.7
DeepLabv3-JFT [23]	86.9
DeepLabv3+ (Xception)	87.8
DeepLabv3+ (Xception-JFT)	89.0

Image w/ BU w/ Decoder

Instance segmentation

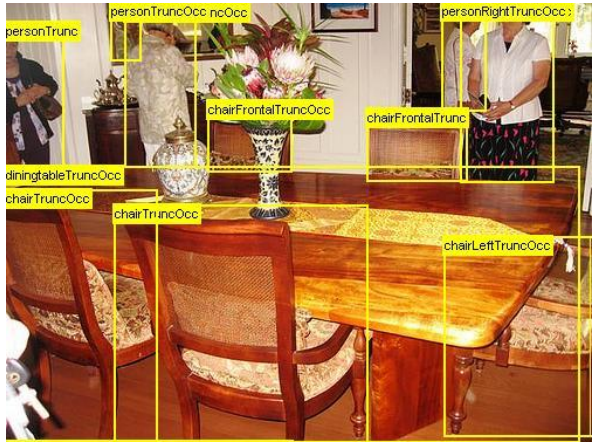
Semantic segmentation separates different classes at the pixel level but does not separate different instances of the same class.

Object detection separates instances but provides only a crude approximation of the instance shape (the box).

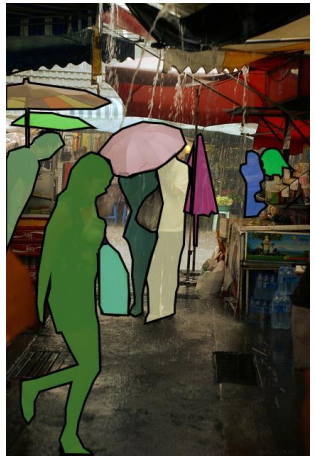
The task of instance segmentation lays at the intersection of the two. It can be defined as the task of

1. detecting all instances of the objects of interest in an image and classifying them; and
2. segmenting them from the background at the pixel level.

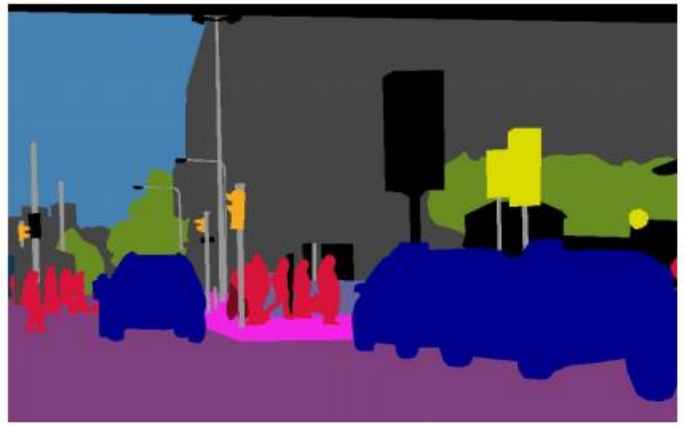
Object detection



Instance segmentation



Semantic segmentation



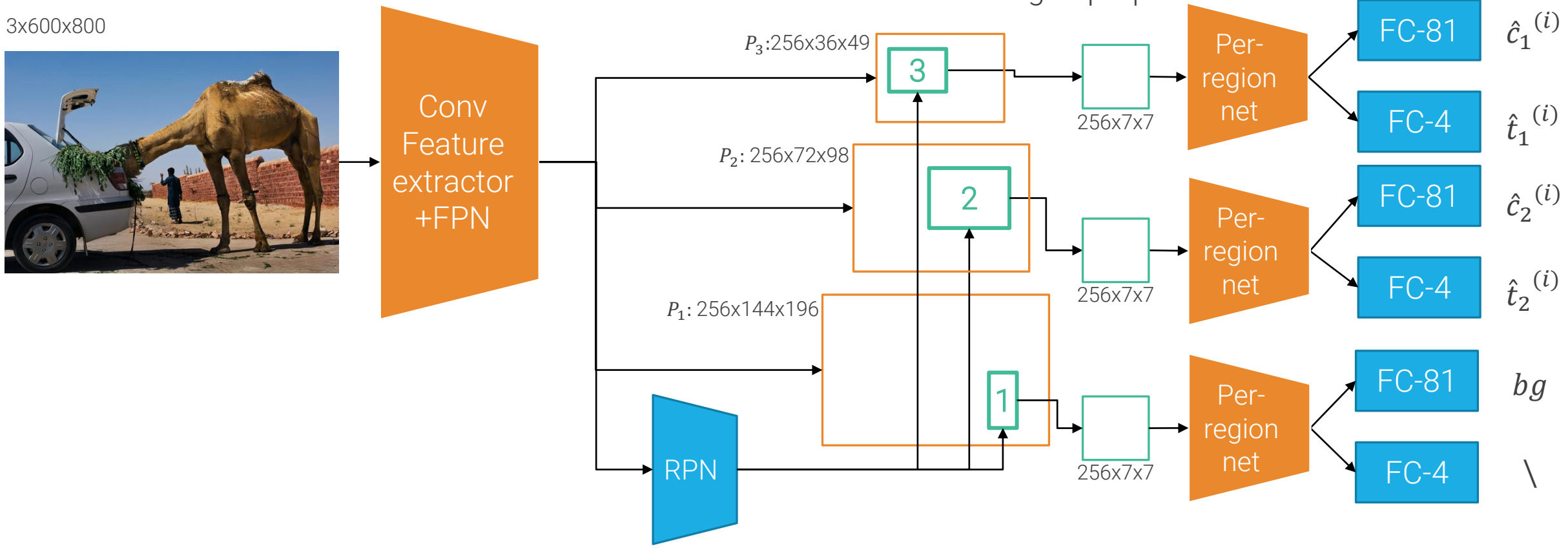
Already seen: Faster R-CNN with FPN

Run expensive backbone feature extractor once on the full image

Region proposal network generates proposals

RoIPool layer crops and warps conv features according to proposals

Per-region network computes output class and BB correction



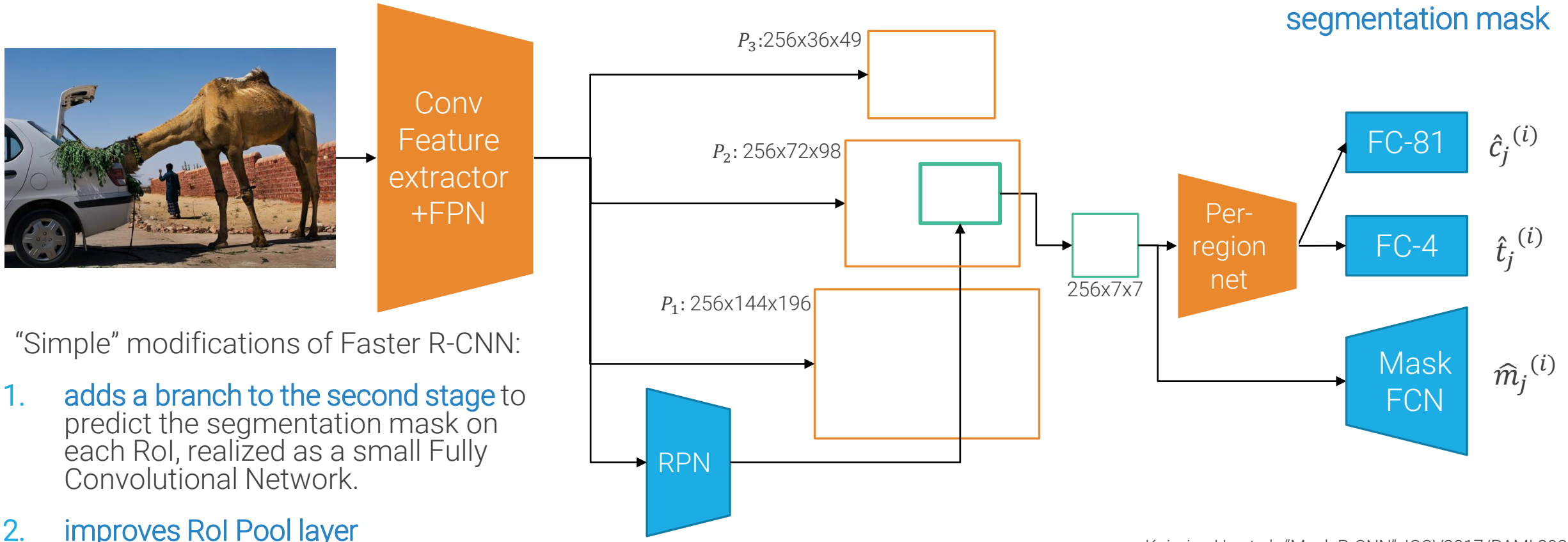
Mask R-CNN

Run expensive backbone feature extractor once on the full image

Region proposal network generates proposals

~~RoIPool~~ RoIAlign layer crops and warps conv features according to proposals

Per-region network computes output class, BB correction, and segmentation mask



"Simple" modifications of Faster R-CNN:

1. adds a branch to the second stage to predict the segmentation mask on each RoI, realized as a small Fully Convolutional Network.
2. improves RoI Pool layer

Kaiming He et al., "Mask R-CNN", ICCV2017/PAMI 2020

Recall: RoI Pool layer

1. Snap region to grid

2. Apply max pooling kernels with approx size $\lfloor H_r/H_o \rfloor \times \lfloor W_r/W_o \rfloor$ and approx stride $s = \lfloor H_r/H_o \rfloor \times \lfloor W_r/W_o \rfloor$

Two sources of quantization

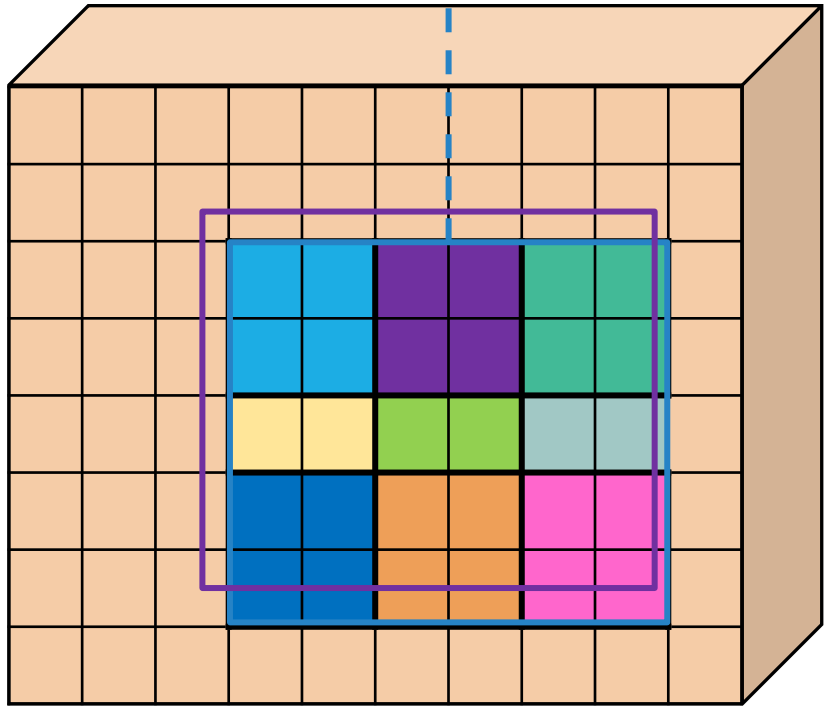
5x6
$H_r \times W_r$

Max Pool over each sub-region

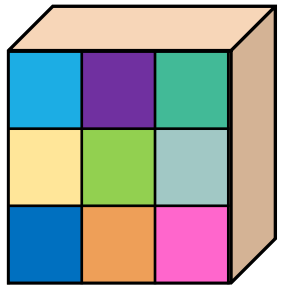
3x600x800

511x634

Convolutional feature extractor



256x8x10
$C_a \times H_a \times W_a$



256x3x3
$C_a \times H_o \times W_o$

RoI Align

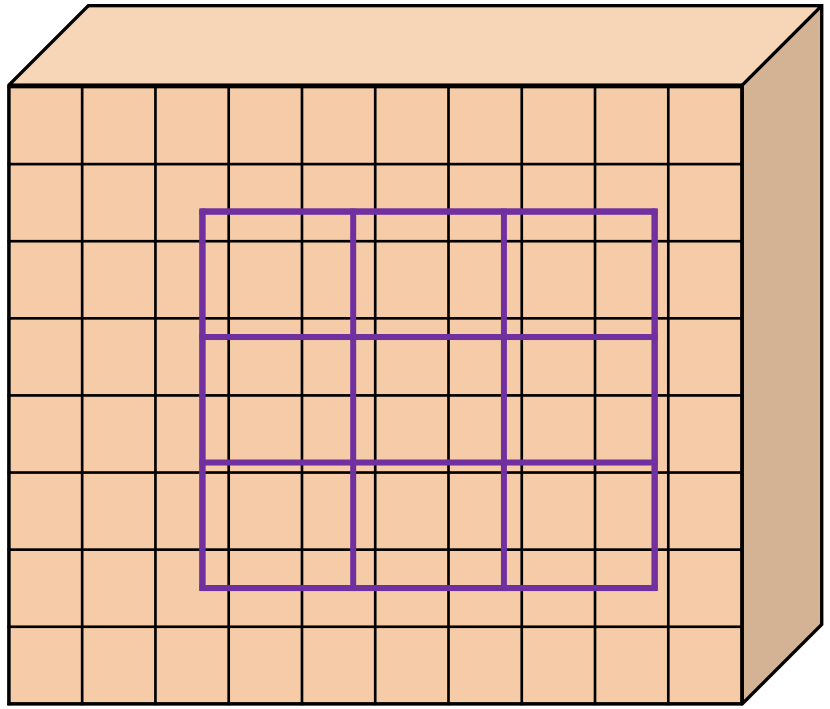
RoI Align avoids both quantizations

1. Divide proposal into equally sized subregions, without snapping RoI to grid.

3x600x800

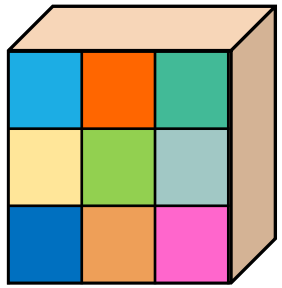
511x634

Convolutional feature extractor



256x8x10

$C_a \times H_a \times W_a$



256x3x3

$C_a \times H_o \times W_o$

RoI Align

RoI Align avoids both quantizations

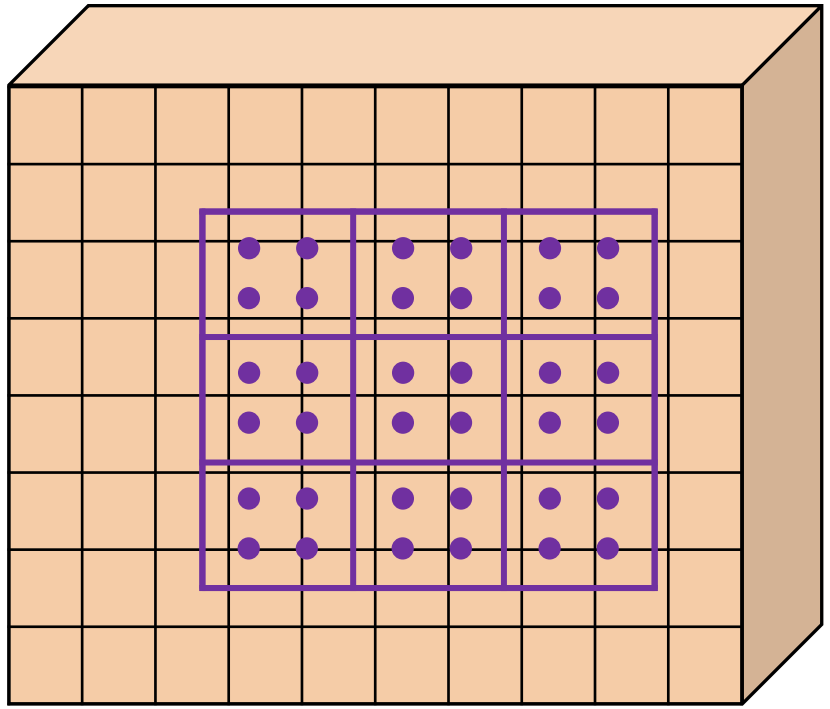
1. Divide proposal into equally sized subregions, without snapping RoI to grid.

2. Sample feature values at a regular grid of points within each RoI cell with bilinear interpolation

3x600x800

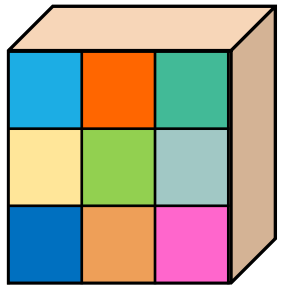
511x634

Convolutional feature extractor



256x8x10

$C_a \times H_a \times W_a$



256x3x3

$C_a \times H_o \times W_o$

RoI Align

RoI Align avoids both quantizations

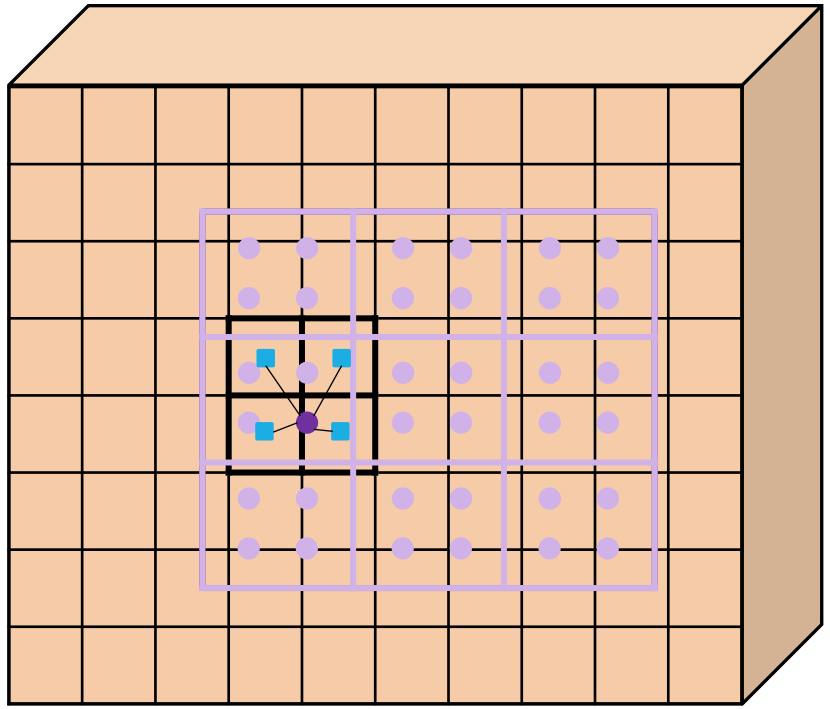
1. Divide into equally sized subregions, without snapping RoI to grid.

2. Sample feature values at a regular grid of points within each RoI cell with bilinear interpolation

3x600x800

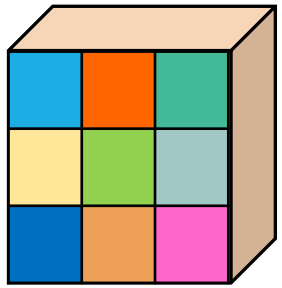
511x634

Convolutional feature extractor



256x8x10

$C_a \times H_a \times W_a$



256x3x3

$C_a \times H_o \times W_o$

RoI Align

RoI Align avoids both quantizations

1. Divide into equally sized subregions, without snapping RoI to grid.

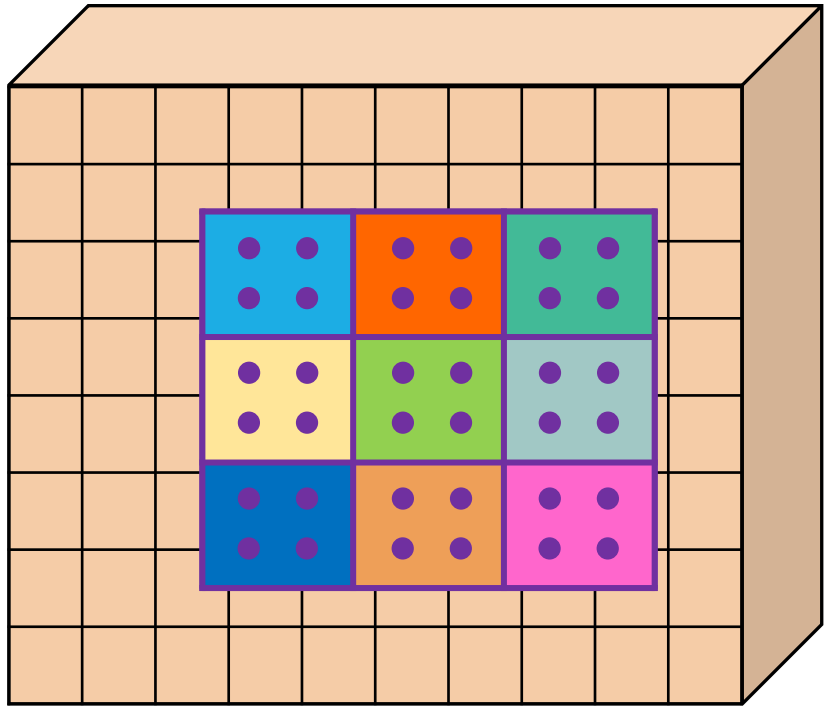
2. Sample feature values at a regular grid of points within each RoI cell with bilinear interpolation

3. Max (or average) pool sampled feature values in each sub-regions

3x600x800

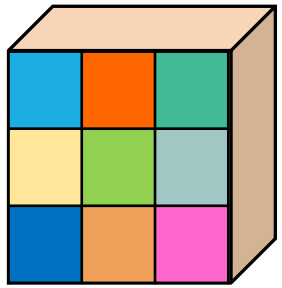
511x634

Convolutional feature extractor



256x8x10

$C_a \times H_a \times W_a$



256x3x3

$C_a \times H_o \times W_o$

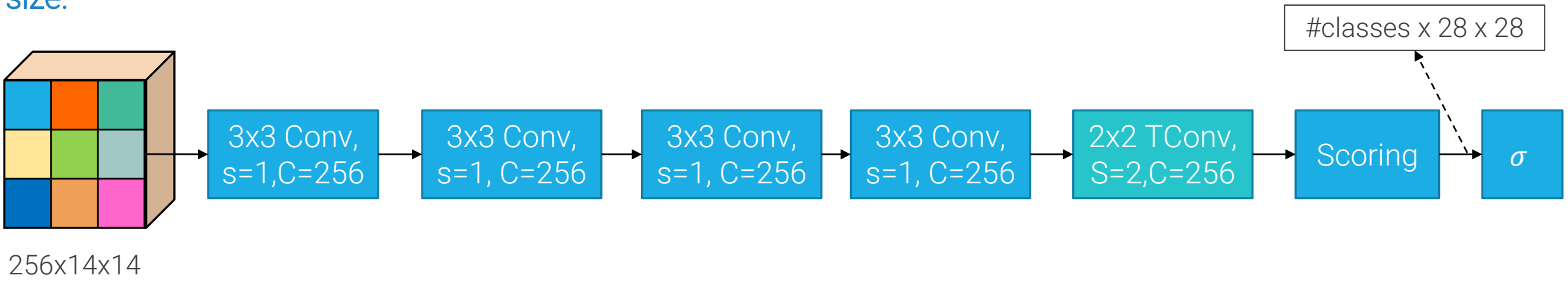
Mask head and loss

	AP	AP_{50}	AP_{75}
softmax	24.8	44.1	25.1
sigmoid	**30.3**	**51.2**	**31.5**
	+5.5	*+7.1*	*+6.4*

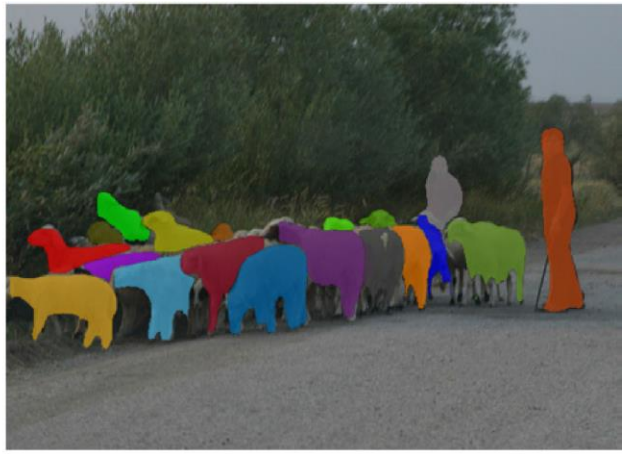
A very small, flat FCN is used to predict a binary mask for each class, with resolution 28x28: while in FCN the output is treated as multi-class problem, here only the predicted mask $\hat{m}_{c^{GT}}$ corresponding to the correct class c^{GT} contributes to the loss. A mask term is added to the usual classification and bounding box correction terms

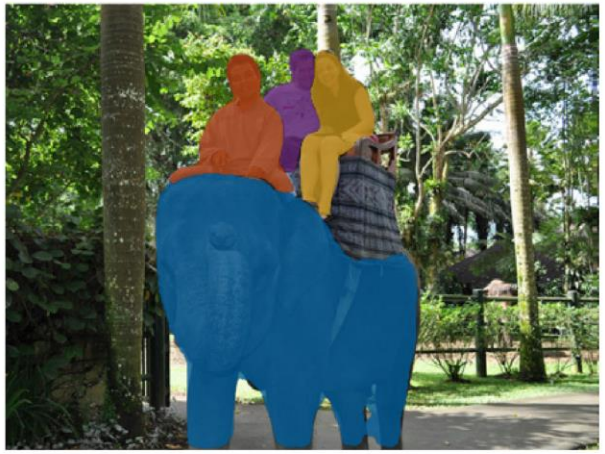
$$L_{mask}(\theta;\ c^{GT}, m^{GT}) = \frac{1}{28 \times 28} \sum_{u=0}^{27} \sum_{v=0}^{27} BCE\left(\hat{m}_{c^{GT},u,v}, m^{GT}_{c^{GT},u,v}\right)$$

At test time, only the mask of the class predicted by the classification branch is used and resampled at the RoI size.







BUS


Flexible design

The framework can easily be extended to human pose estimation.

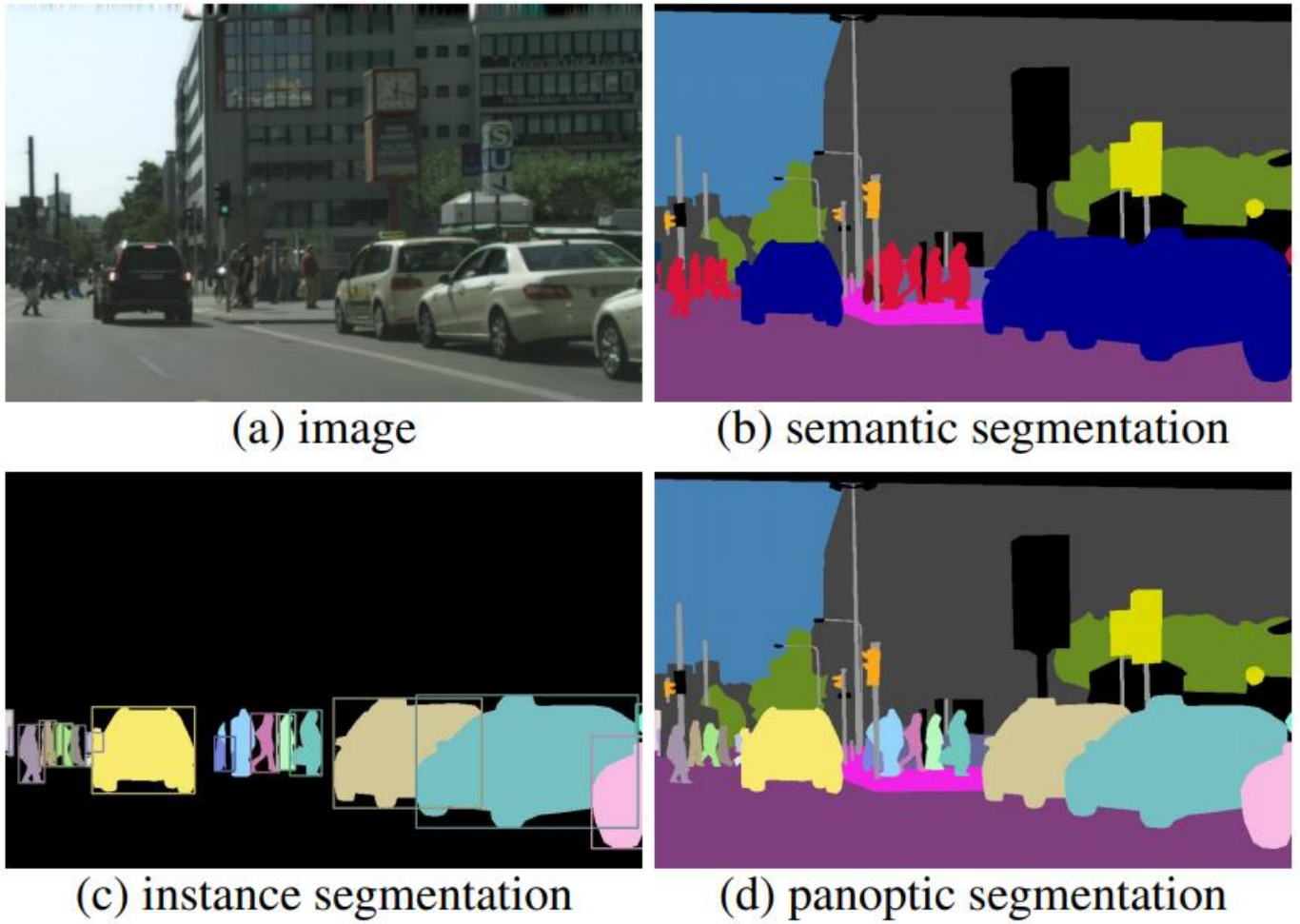
Another FCN head is added to the per region network, which is trained to reproduce K 56×56 1-hot encoded masks, which indicate the position of each keypoint.

Panoptic segmentation

Things: countable object categories, that can be separated into individual instances, such as cars, persons, chairs. The subject of object detection and instance segmentation.

Stuff: uncountable amorphous regions of similar texture or material such as grass, sky, road. Semantic segmentation recognize stuff categories but treat also “things” as “stuff”.

Panoptic segmentation unifies them: it requires to label each pixel with a category and also provide an instance id for each “thing”.



Alexander Kirillov et al., “Panoptic Segmentation”, CVPR 2019

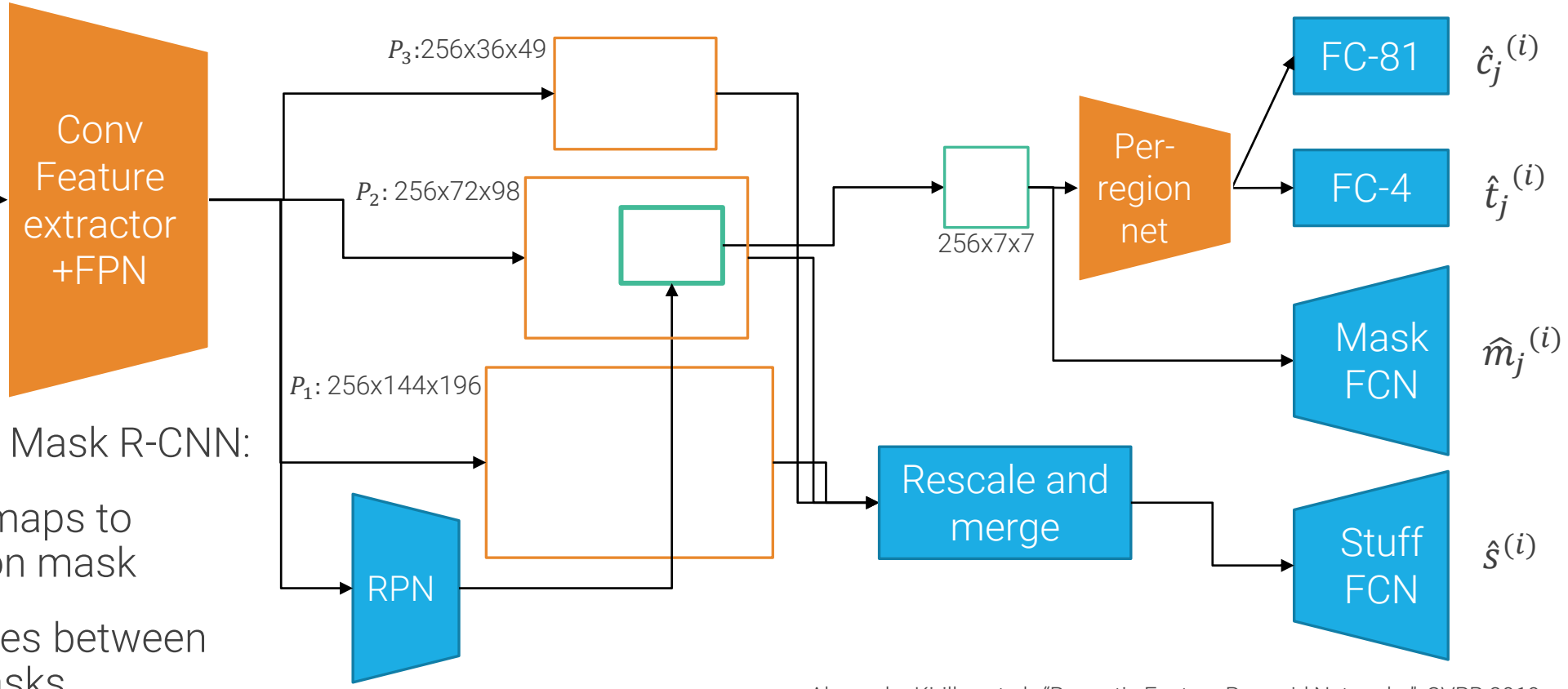
Panoptic Feature Pyramid Network

Run expensive backbone feature extractor once on the full image

Region proposal network generates proposals

RoIAlign layer crops and warps conv features according to proposals

Per-region network computes output class, BB correction, and segmentation masks

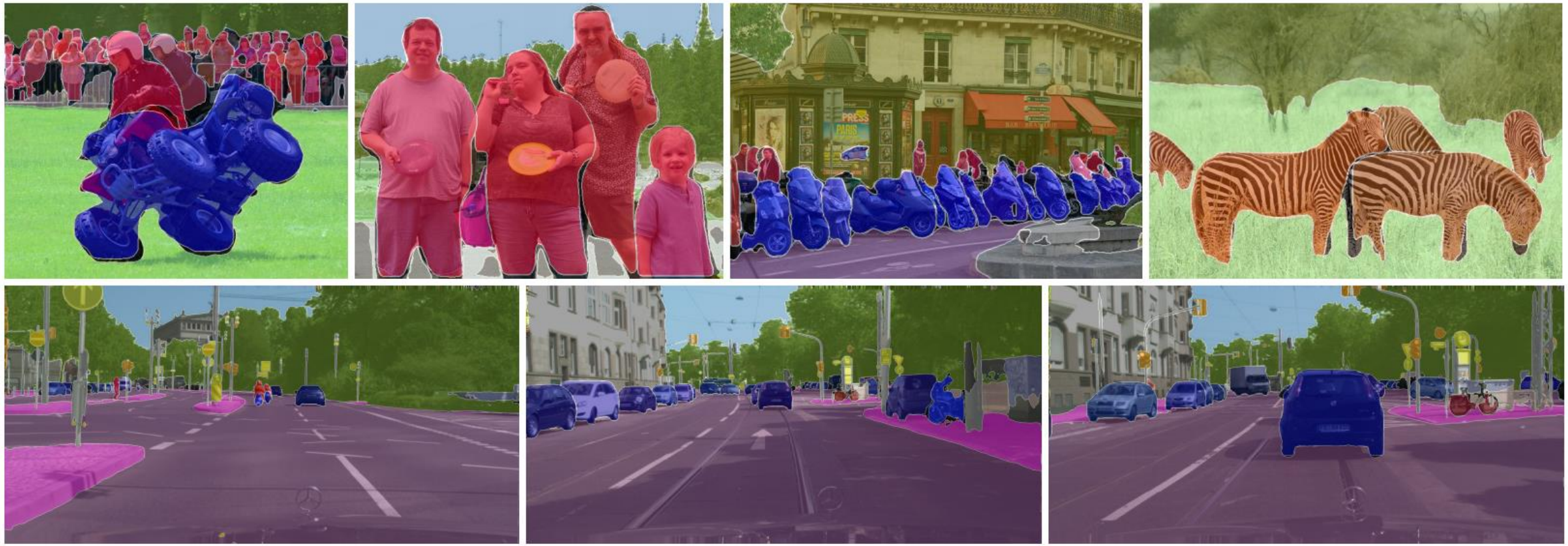


"Simple" modifications of Mask R-CNN:

1. Merges FPN feature maps to predict a segmentation mask
2. Resolve inconsistencies between instance and stuff masks

Alexander Kirillov et al., "Panoptic Feature Pyramid Networks", CVPR 2019

Panoptic Feature Pyramid Network



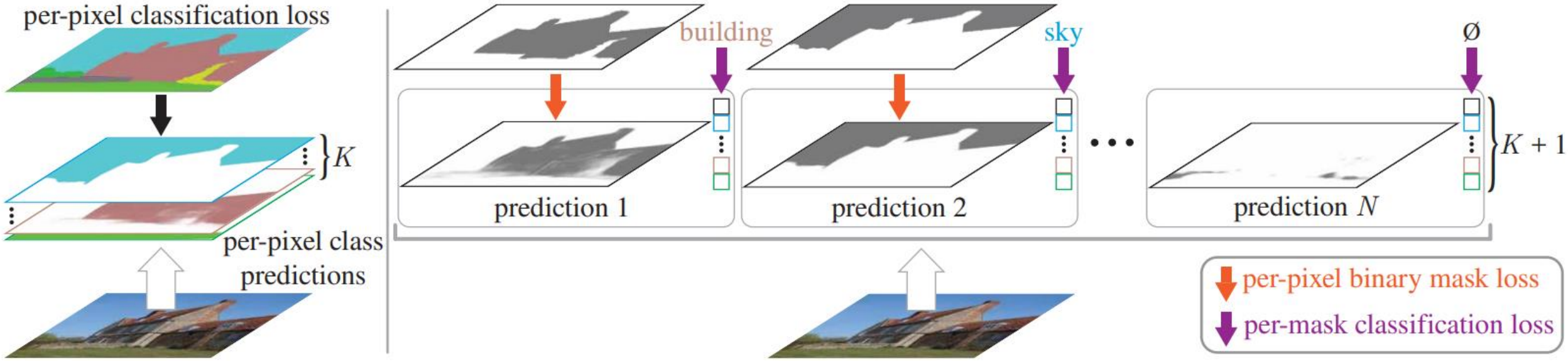
Alexander Kirillov et al., “Panoptic Feature Pyramid Networks”, CVPR 2019

MaskFormer

Semantic segmentation, since FCNs, is usually solved classifying each pixel into C classes

Instance/panoptic segmentation, instead, partitions the image into N masks which are classified.

However, mask classification (with masks as large as the image) can solve all kinds of segmentation.

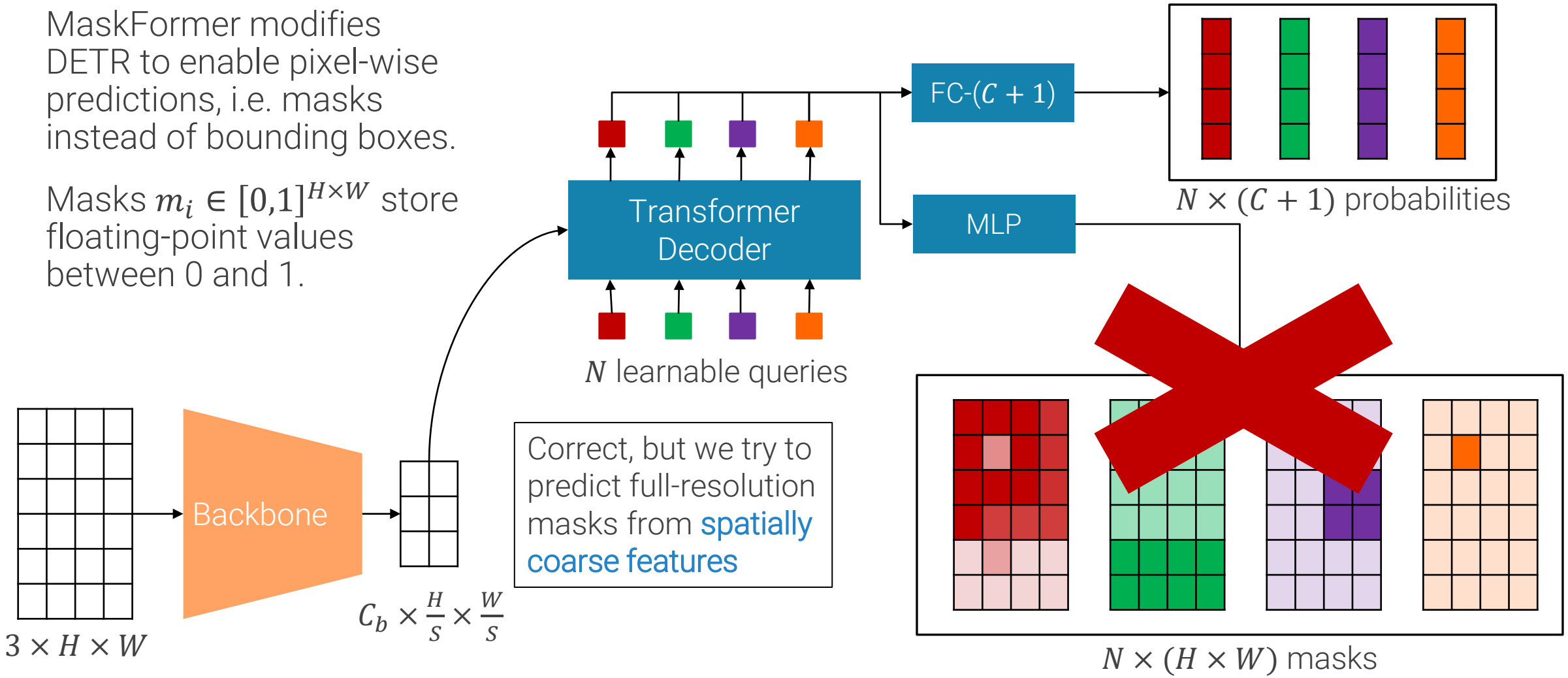


Bowen Cheng et al., "Per-Pixel Classification is Not All You Need for Semantic Segmentation", NeurIPS 2021

Architecture

MaskFormer modifies DETR to enable pixel-wise predictions, i.e. masks instead of bounding boxes.

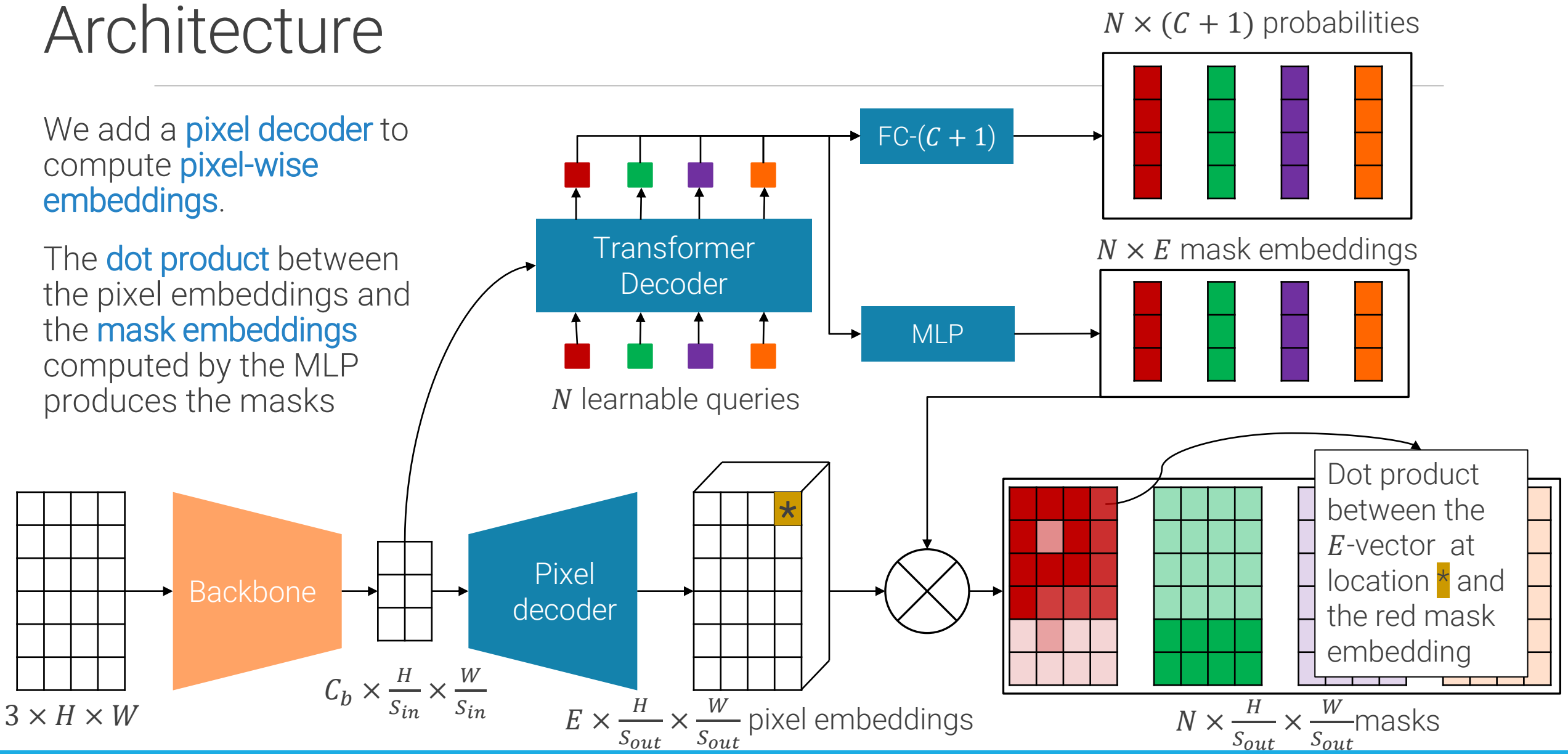
Masks $m_i \in [0,1]^{H \times W}$ store floating-point values between 0 and 1.



Architecture

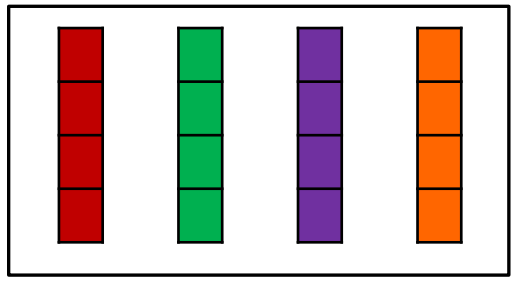
We add a **pixel decoder** to compute **pixel-wise embeddings**.

The **dot product** between the pixel embeddings and the **mask embeddings** computed by the MLP produces the masks



Inference – instance and panoptic case

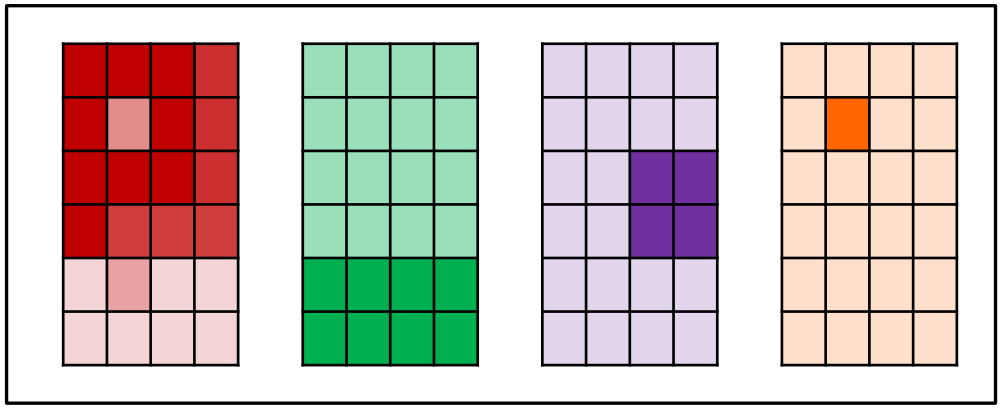
Given N pairs of probabilities and masks, at inference time MaskFormer assigns a pixel at location (u, v) to probability-mask pair i only if both the most likely class probability $p_i(c_i)$ and the mask prediction probability $m_i(u, v)$ are high.



$N \times (C + 1)$ probabilities

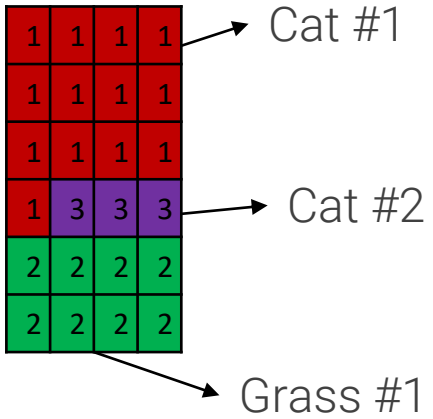
$c_i = \text{argmax}_{c \in [1,\ldots,C,\emptyset)} \; p_i(c)$

N classes with their probability



$N \times \frac{H}{S_{out}} \times \frac{W}{S_{out}}$ masks

$\text{argmax}_{i:\, c_i \neq \emptyset} \; p_i(c_i) \cdot m_i(u, v)$
+ upsample

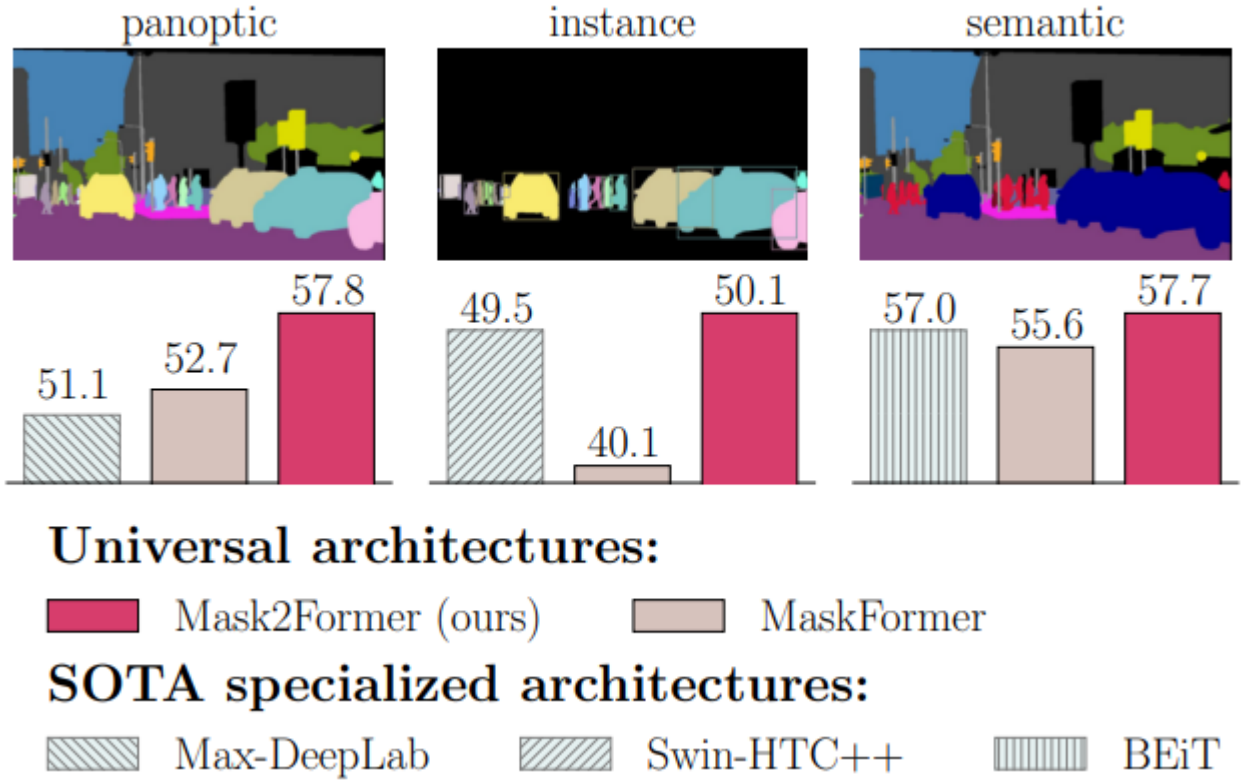


Toward a “universal” architecture

MaskFormer shows how a single architecture can handle all segmentation tasks, it is a step toward a universal architecture for segmentation.

Yet,

- its performance is not state of the art compared to specialized architectures, especially on instance segmentation, where small objects are important.
- it is also harder to train. It requires expensive hardware (32 GB of VRAM for one image) and a much longer training schedule.



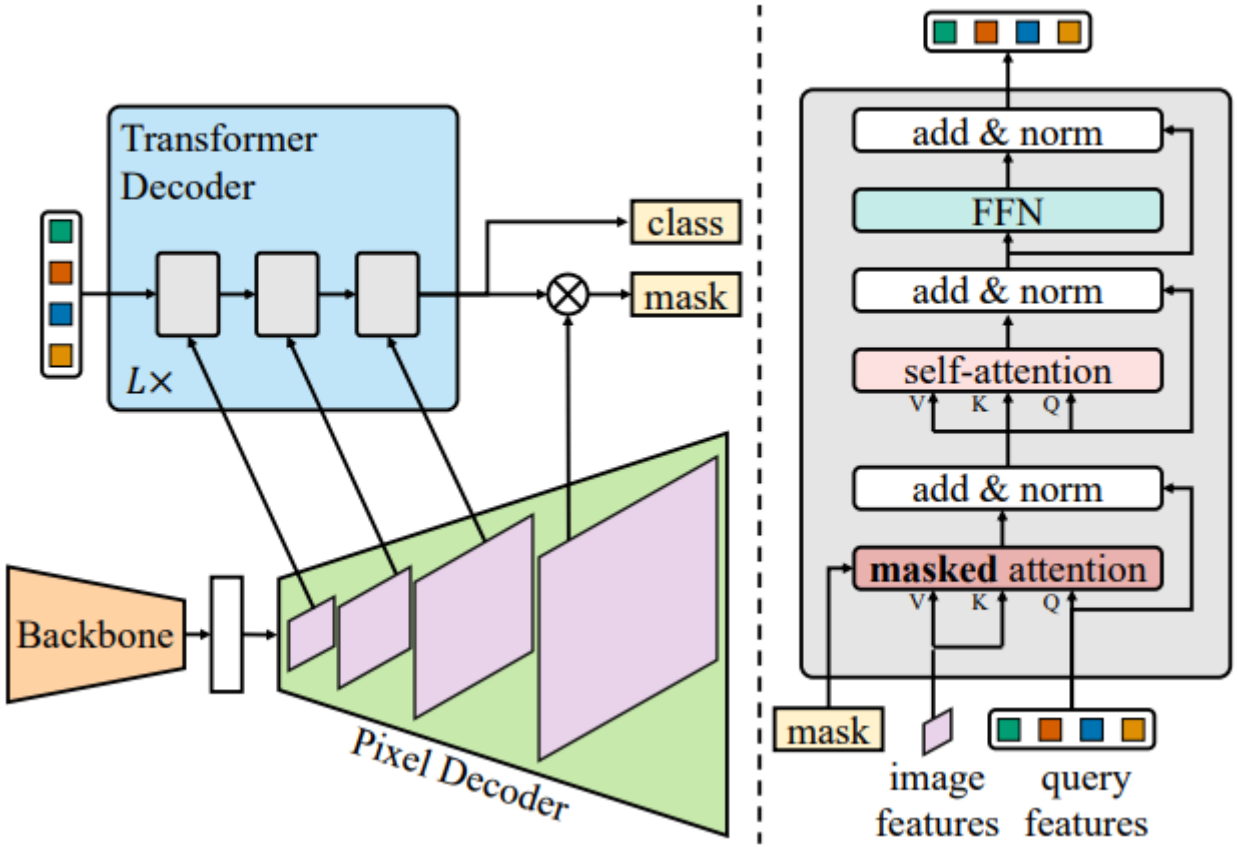
Bowen Cheng et al., “Masked-attention Mask Transformer for Universal Image Segmentation”, CVPR 2022

Mask2Former

Masked-attention Mask Transformer (Mask2Former) improves MaskFormer to address these limitations.

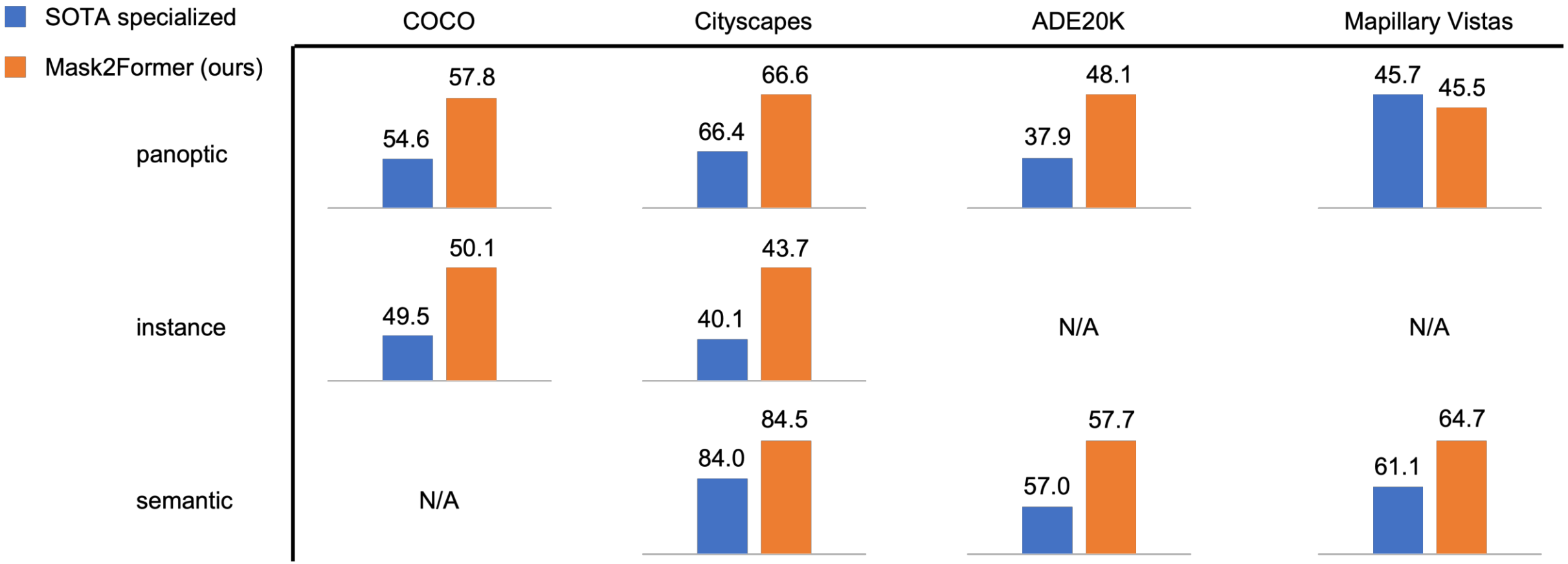
It adds:

- masked attention in the decoder, for faster training convergence and improved performance
- the use of multi-scale high-resolution features to detect small objects
- faster training by not supervising all pixels



Bowen Cheng et al., "Masked-attention Mask Transformer for Universal Image Segmentation", CVPR 2022

Results



… yet, the same “universal” architecture must be trained with task specific supervision to reach SOTA results.

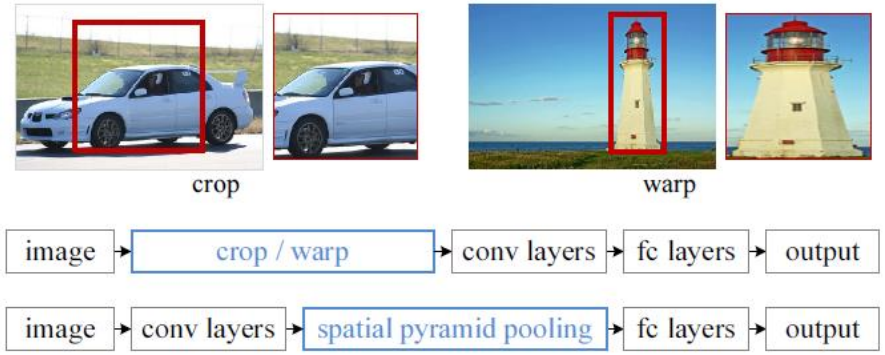
https://bowenc0221.github.io/mask2former/

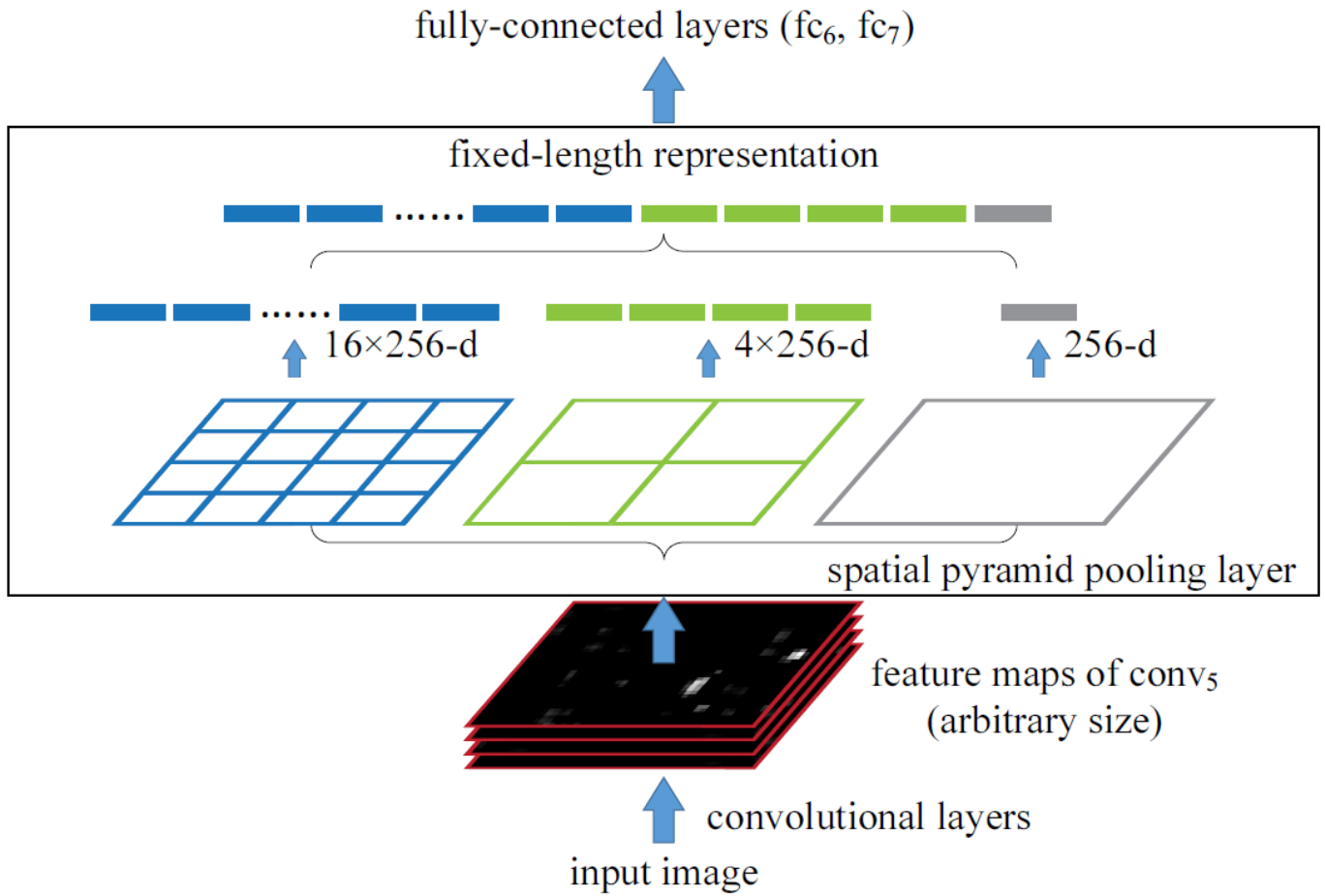
Appendix

Spatial Pyramid Pooling layer

A solution to effectively recognize objects at different scales with fixed-resolution networks was the spatial pyramid pooling layer.

To obtain a fixed-size representation to feed the fully connected layers, the variable-size convolutional feature map is max-pooled with a fixed number $n \times n$ of variable-size windows, and the resulting fixed-size flattened vectors are concatenated. It is an extension of global average pooling to preserve spatially localized information.





Kaiming He et al., "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition", PAMI 2015.

ASPP DeepLab v2

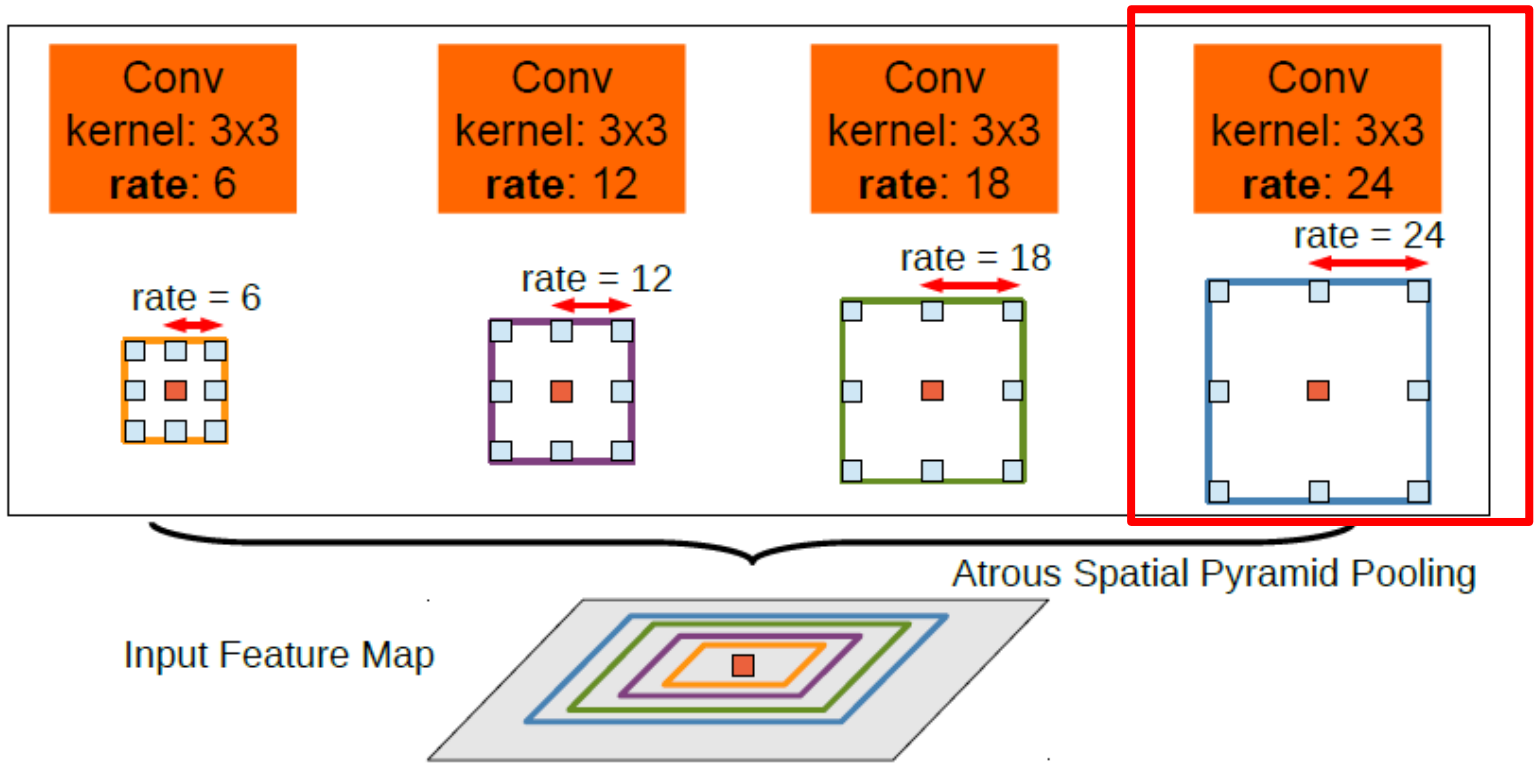
SPP layer can be seen as an extension of global average pooling that preserves spatial information.

DeepLab emulates the property of the SPP layer to capture spatial context at multiple scales by leveraging again atrous convolutions.

In DeepLab v2, the module performs 3x3 conv with increasing dilation rates.

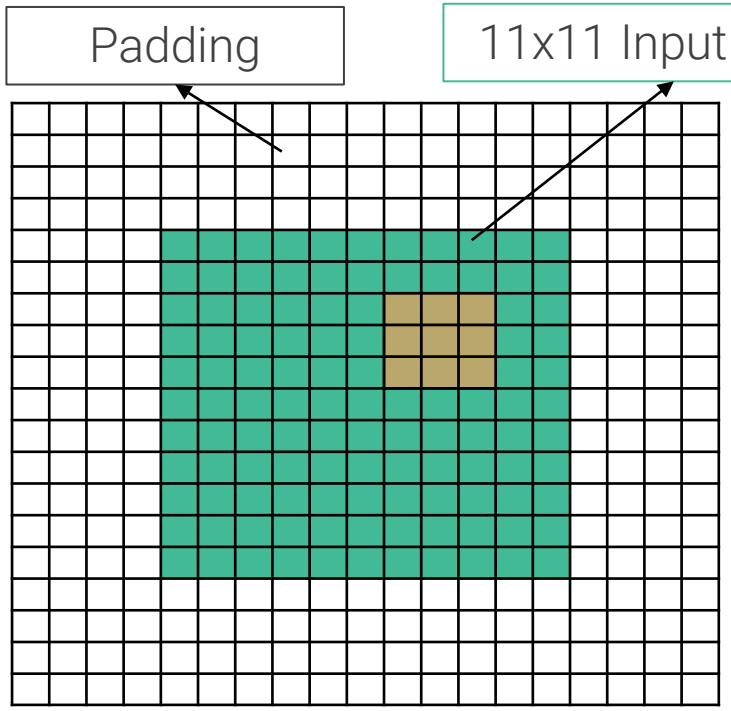
Outputs are concatenated and aggregated by a scoring 1x1 conv to produce the output scores map.

What is this layer really doing?

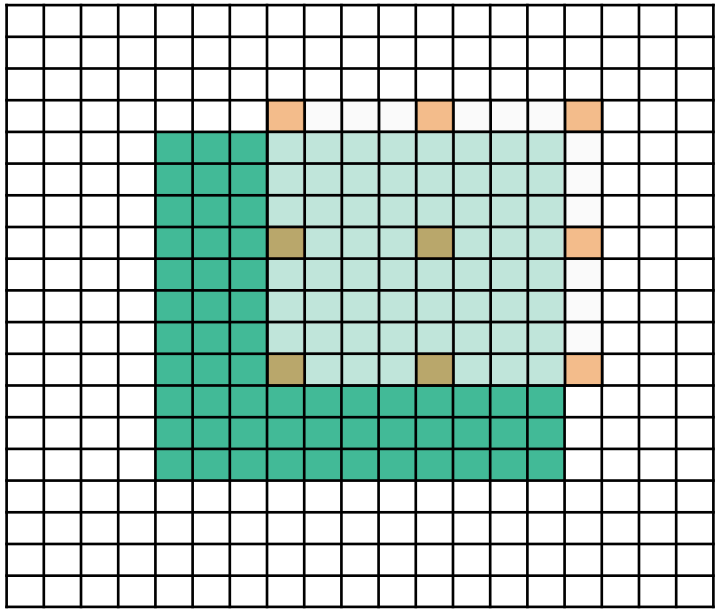


High dilation rates

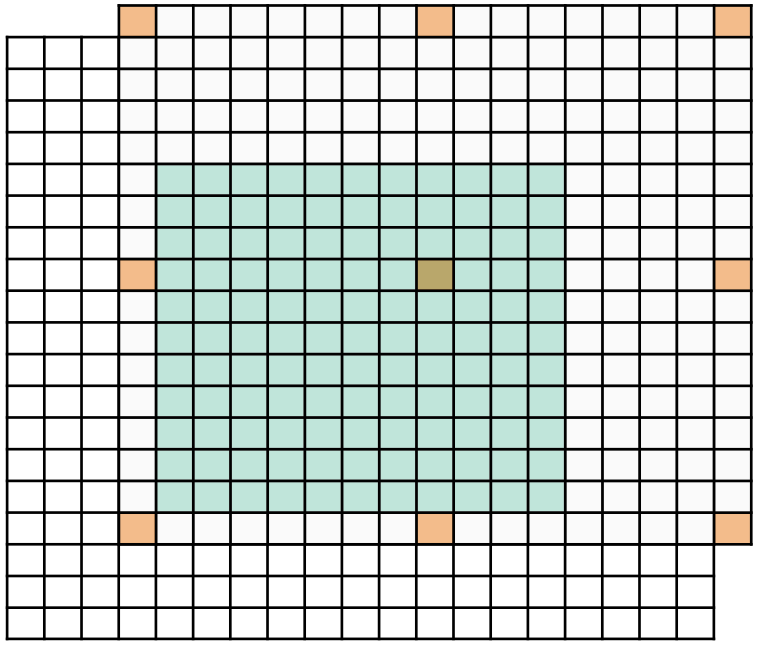
When the dilation rate grows, the number of positions where all the 9 weights of the kernel are used (i.e. are not multiplied by zero padding) shrinks.



$r = 1$
All weights used



$r = 4$
4 weights used

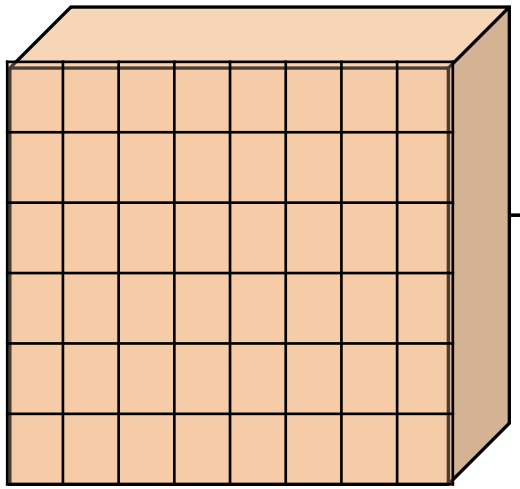


$r = 8$
1 weight used

ASPP DeepLab v3

All convolutions followed by BN and ReLU.

When backbone has output stride 8, all rates are doubled



512xH/16xW/16
Stride-16 output of dilated backbone

1x1 Conv, C=256

3x3 Conv, C=256, r=6

3x3 Conv, C=256, r=12

3x3 Conv, C=256, r=18

Global Average Pooling

1x1 Conv, C=256

Spatially replicate

Concatenate

1x1 Conv, C=256

Scoring

Global image features added to emulate larger rate convs.