

Convolutional Neural Networks for Semantic Segmentation

Giacomo Boracchi

giacomo.boracchi@polimi.it

Artificial Neural Networks and Deep Learning

Politecnico di Milano, AY2023/2024

<https://boracchi.faculty.polimi.it/>

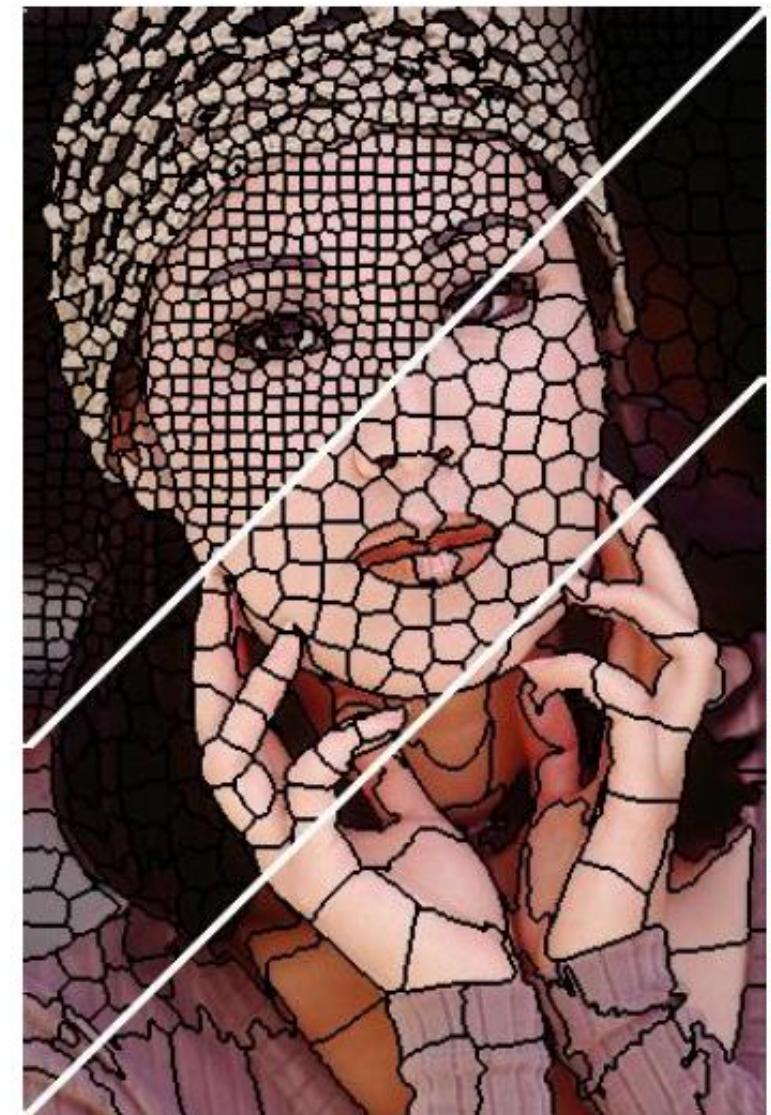
Image Segmentation

Image Segmentation

Goal: identify groups of pixel that “go together”

- Group together similar-looking pixel for efficiency
- Separate images into coherent objects

One way of looking at segmentation is **clustering**



Achanta, et al *SLIC superpixels compared to state-of-the-art superpixel methods.* TPAMI 2012

Problem Formulation: Image Segmentation

Given an image $I \in \mathbb{R}^{R \times C \times 3}$, having as domain \mathcal{X} , the goal of image segmentation consists in estimating a partition $\{R_i\}$ such that

$$\bigcup_i R_i = \mathcal{X}$$

and $R_i \cap R_j = \emptyset, \quad i \neq j$

There are two types of sementation:

- Unsupervised (which we do not address here)
- Supervised (or Semantic)

Semantic Segmentation

Objects appearing in the image:

Boat

Dining table

Person



Semantic Segmentation, the problem

The goal of semantic segmentation is:

Given an image I , associate to each pixel (r, c) a label from Λ .

The result of segmentation is a map of labels containing in each pixel the estimated class.

Remark: In this image there is no distinction among persons.

⚠ Segmentation does not separate different instances belonging to the same class.
That would be instance segmentation.



<http://www.robots.ox.ac.uk/~szheng/crfasrnndemo>

Semantic Segmentation, the problem

Assign to each pixel of an image $I \in \mathbb{R}^{R \times C \times 3}$:

- a label $\{l_i\}$ from a fixed set of categories
 $\Lambda = \{"wheel", "cars", \dots, "castle", "baboon"\},$
 $I \rightarrow S \in \Lambda^{R \times C}$

where $S(x, y) \in \Lambda$ denotes the class associated to the pixel (x, y)

To this purpose, you are given a training set

$$TR = \{(I, GT)_i\}$$

which have been (manually) annotated and that can be used for training

Deep Neural Networks for Semantic Segmentation

Predicting dense classification outputs on images

Training Set

The training set is made of pairs (I, GT) , where the GT is a pixel-wise annotated image over the categories in Λ



I



GT

The Training Set



COCO Explorer

COCO 2017 train/val browser (123,287 images, 886,284 instances). Crowd labels not shown.



The Training Set $TR = \{(I, GT)_i\}$

I is the input image
 GT is the annotation
(shown in overlays)



Lin, Tsung-Yi, et al. "Microsoft coco: Common objects in context." ECCV 2014.

The Training Set $TR = \{(I, GT)_i\}$

I is the input image
 GT is the annotation
(shown in overlays)



Lin, Tsung-Yi, et al. "Microsoft coco: Common objects in context." ECCV 2014.

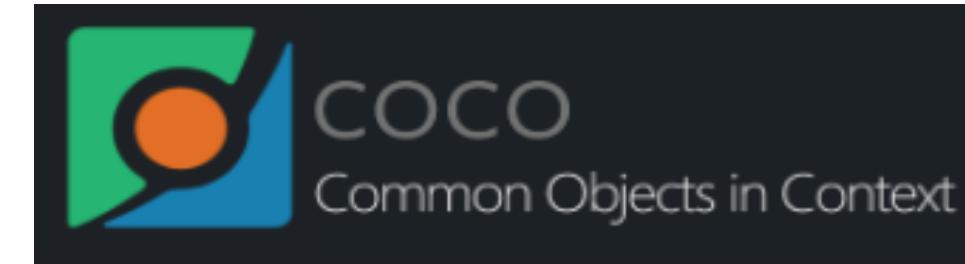
The Training Set $TR = \{(I, GT)_i\}$

I is the input image

GT is the annotation
(shown in overlays)



The Training Set

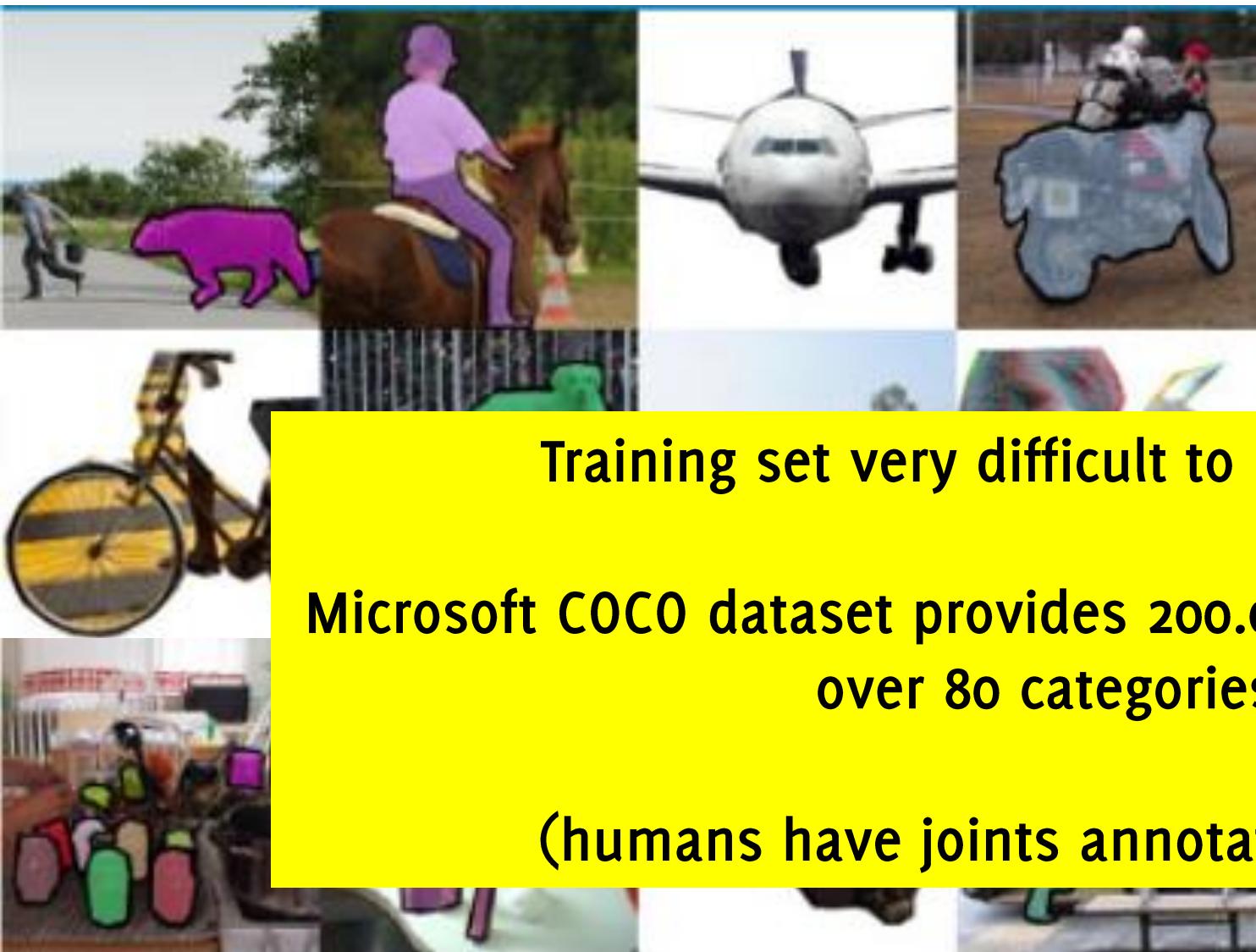


COCO is a large-scale object detection, segmentation, and captioning dataset.

COCO has several features:

- ✓ Object segmentation
- ✓ Recognition in context
- ✓ Superpixel stuff segmentation
- ✓ 330K images (>200K labeled)
- ✓ 1.5 million object instances
- ✓ 80 object categories
- ✓ 91 stuff categories
- ✓ 5 captions per image
- ✓ 250,000 people with keypoints

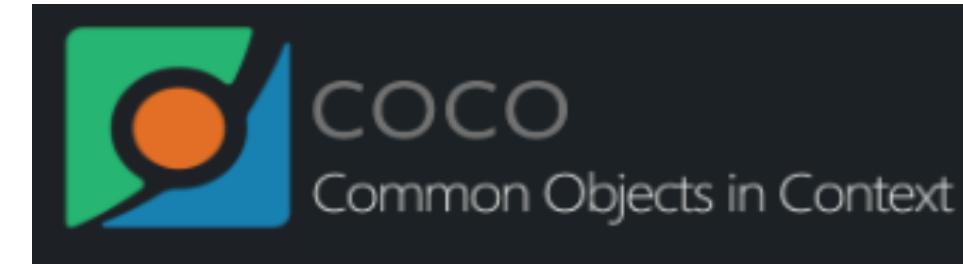
The Training Set



Training set very difficult to annotate 😊

Microsoft COCO dataset provides 200.000 annotated images
over 80 categories

(humans have joints annotated as well)



COCO is a large-scale object detection, segmentation, and captioning dataset. COCO has several features:

Object segmentation

Text

Segmentation

OK labeled)

Instances

es

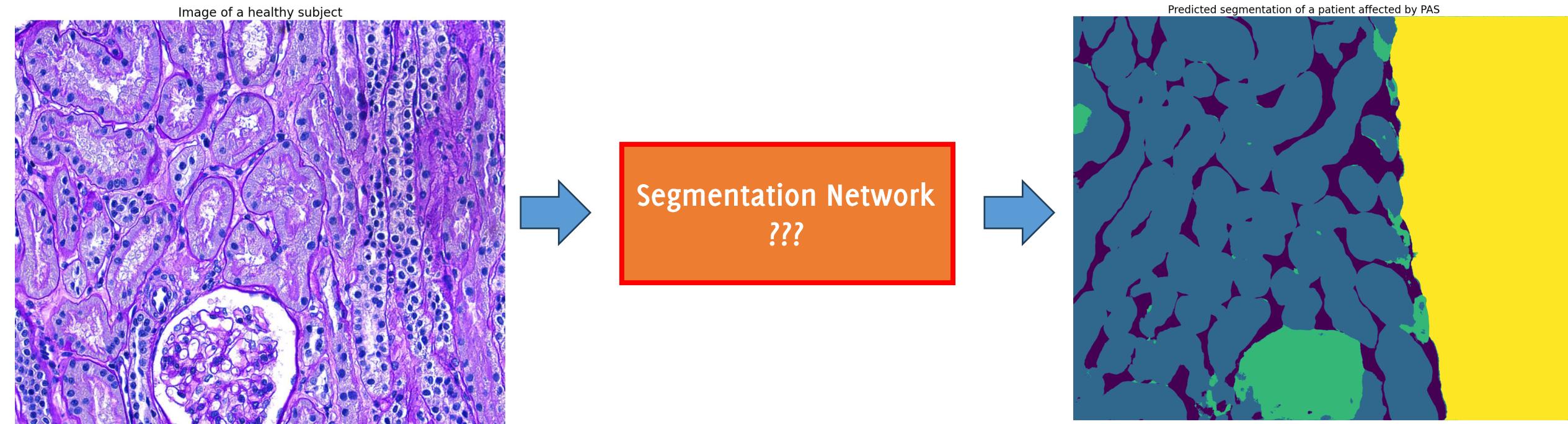
ge

with keypoints

Semantic Segmentation by Fully Convolutional Neural Networks

Predicting dense outputs for arbitrary-sized inputs

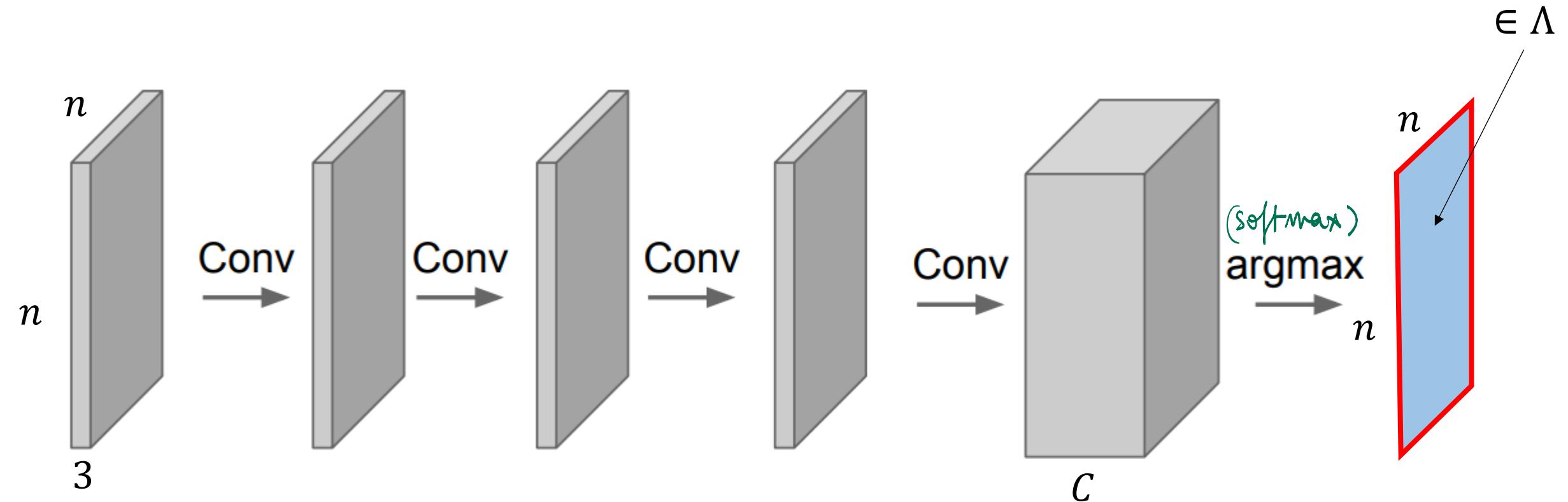
How to train an end-to-end model for semantic segmentation?



Semantic Segmentation by Convolutions Only

What if we avoid any pooling (just conv2d and activation layers)?

- Very small receptive field
- Very inefficient



Drawbacks of convolutions only

- On the one hand we need to “go deep” to extract high level information on the image
- On the other hand we want to stay local not to loose spatial resolution in the predictions

Semantic segmentation faces an inherent tension between semantics and location:

- *global information resolves what*, while
- *local information resolves where*

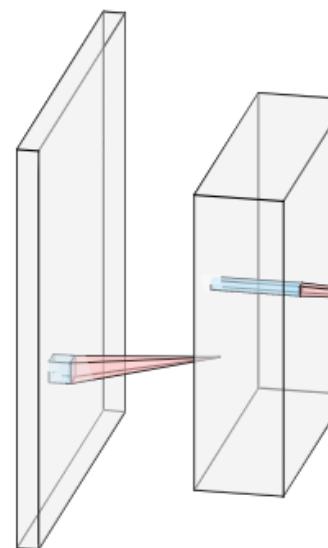
Combining fine layers and coarse layers lets the model make local predictions that respect global structure.

↳ of next slide

Low-dimensional representation and upsampling

An architecture like the following would probably be more suitable for semantic segmentation

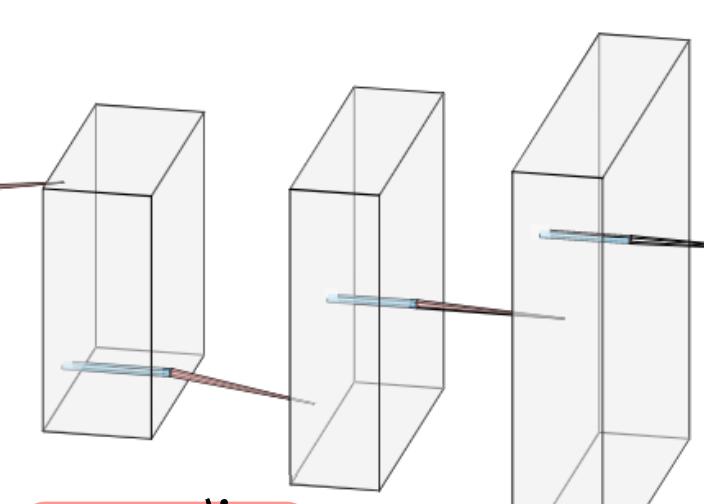
$n \times n \times 3$



$m \times m \times D, m \ll n$



$n \times n \times L$



$\text{argmax}(\cdot)$

1 label by pixel.

$n \times n \times 1$

contractive
"like a classifier"

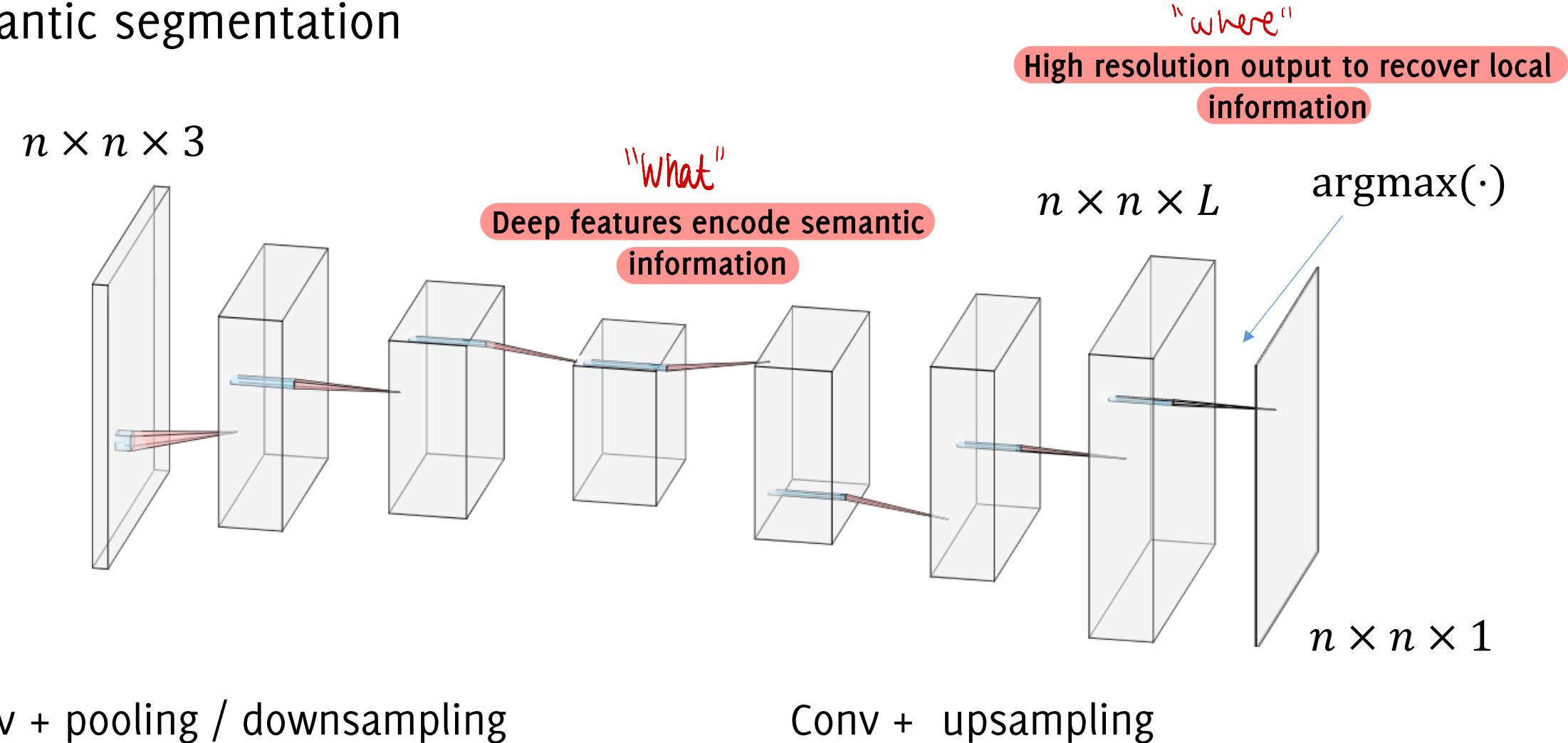
Conv + pooling / downsampling

expanding

Conv + upsampling

Low-dimensional representation and upsampling

An architecture like the following would probably be more suitable for semantic segmentation

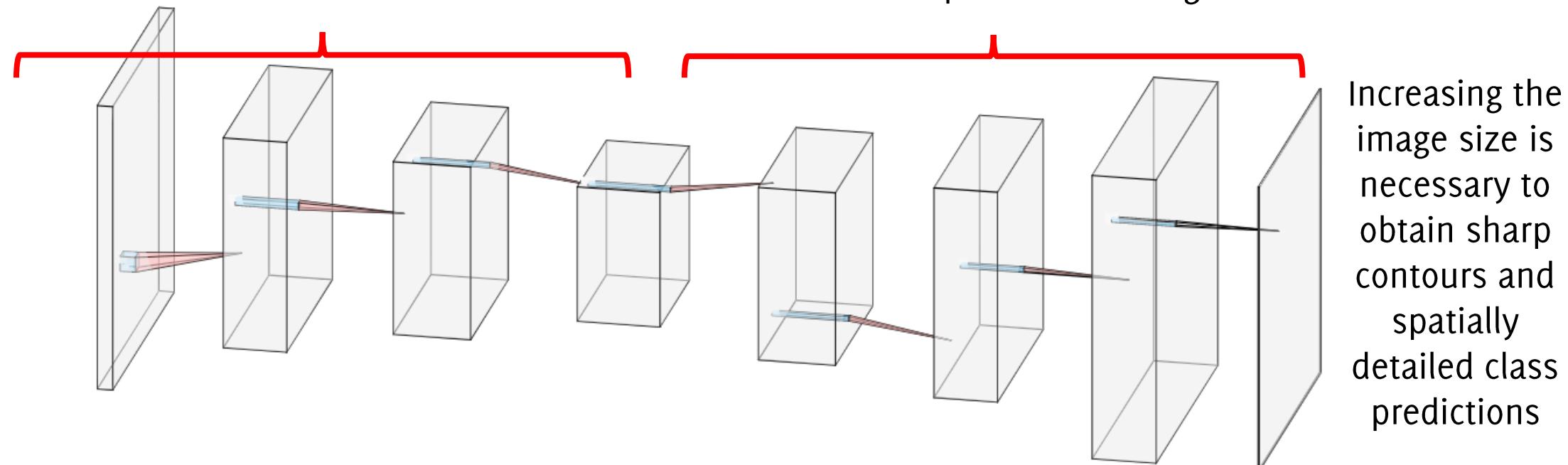


Low-dimensional representation and upsampling

An architecture like the following would probably be more suitable for semantic segmentation

The first half is the same of a classification network

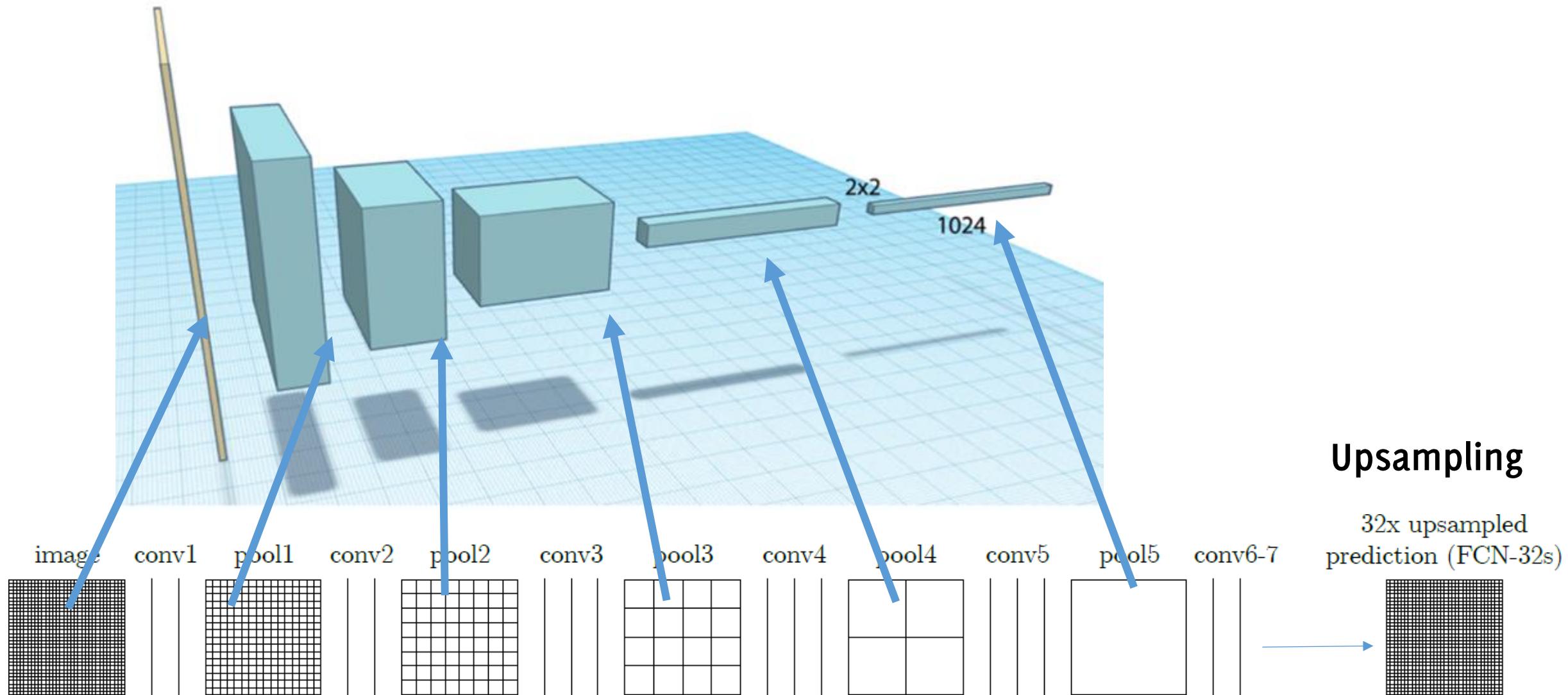
The second half is meant to upsample the predictions to cover each pixel in the image



Conv + pooling / downsampling

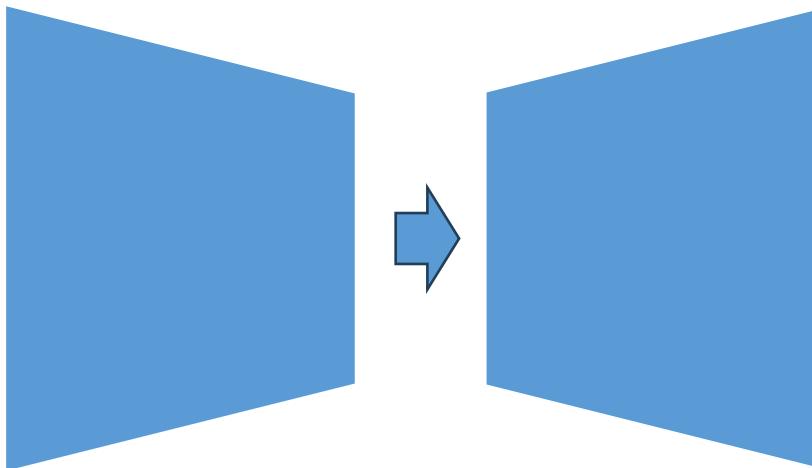
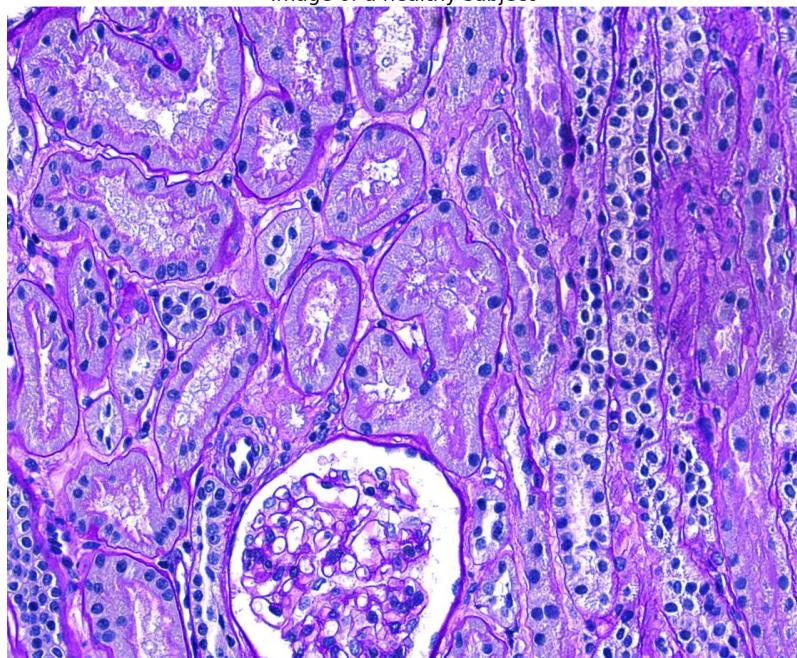
Conv + upsampling

Low-dimensional representation and upsampling

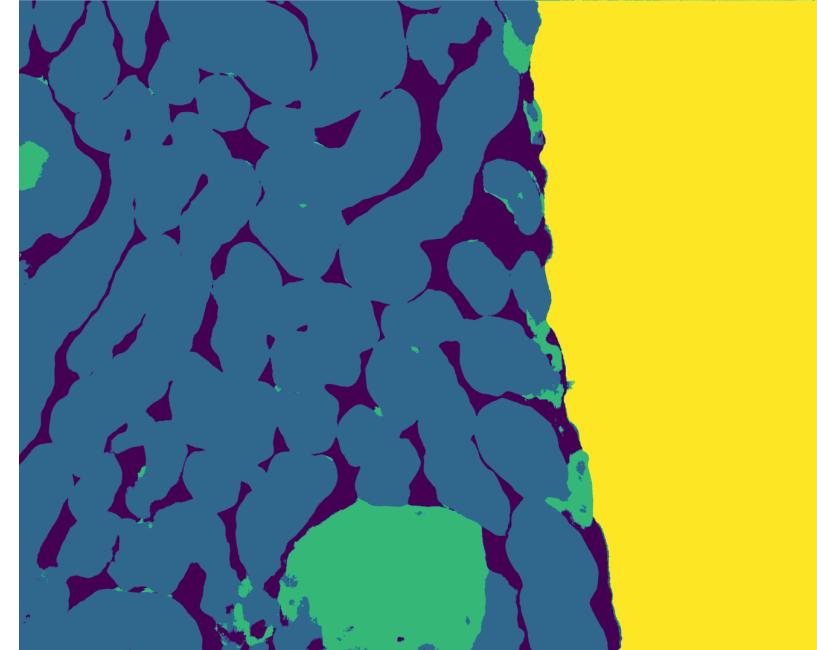


Semantic Segmentation Networks

Image of a healthy subject

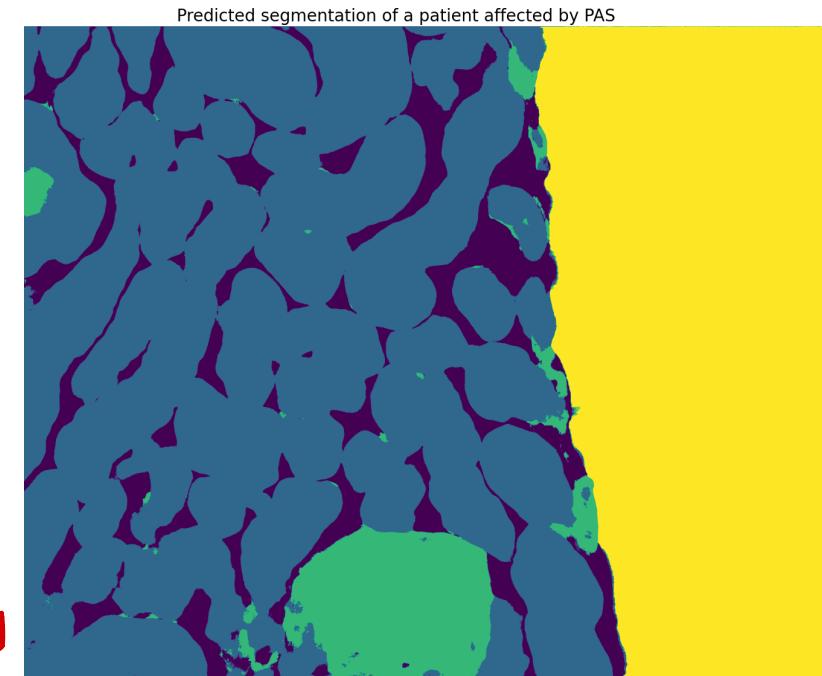
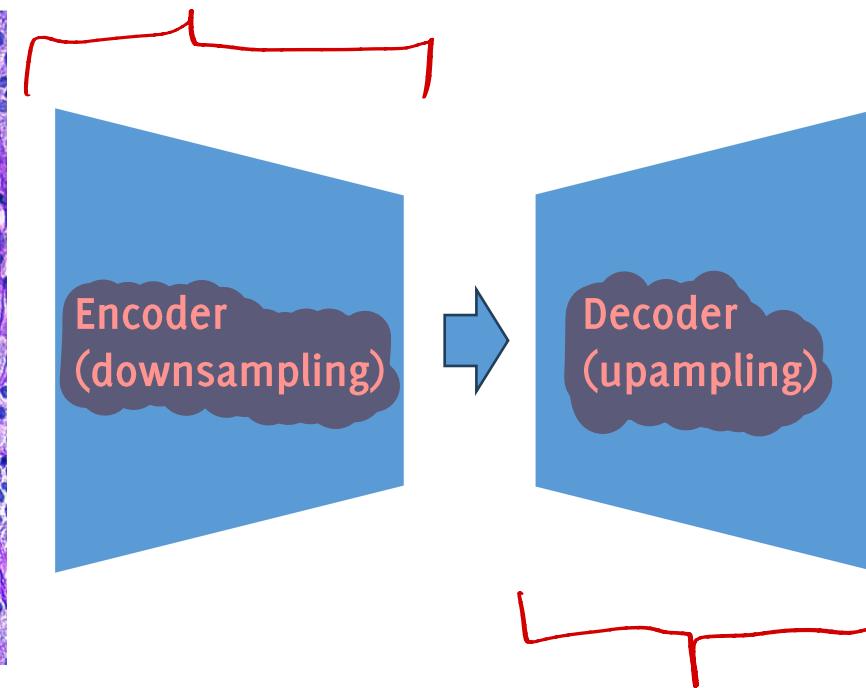
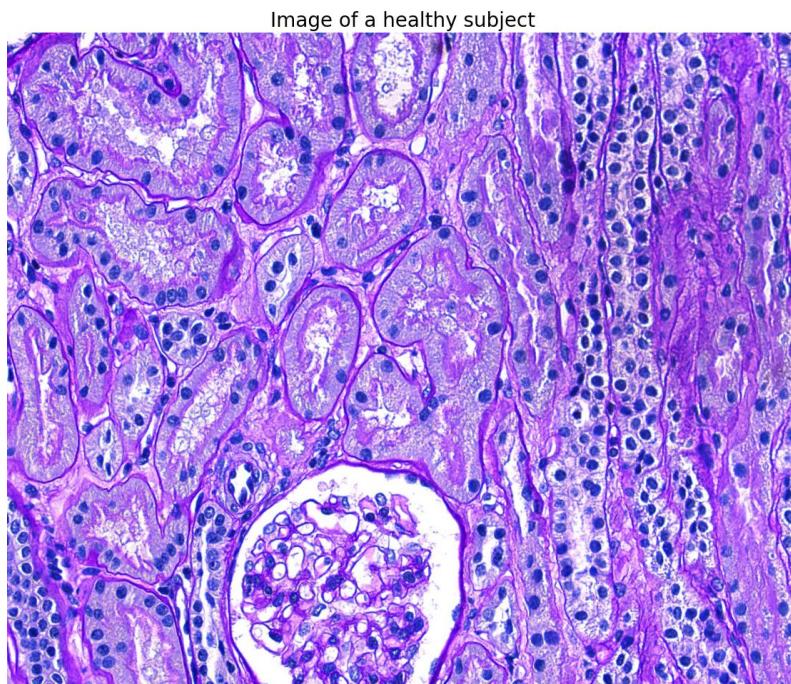


Predicted segmentation of a patient affected by PAS



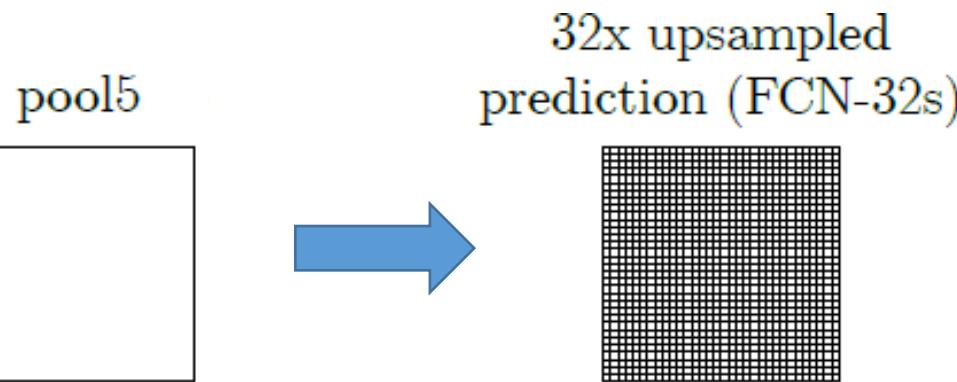
Semantic Segmentation Networks

This is a standard CNN,
we know how to
implement this network



What about this?

How to perform upsampling?



Nearest Neighbor

1	2
3	4



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

Input: 2 x 2

Output: 4 x 4

“Bed of Nails”

1	2
3	4



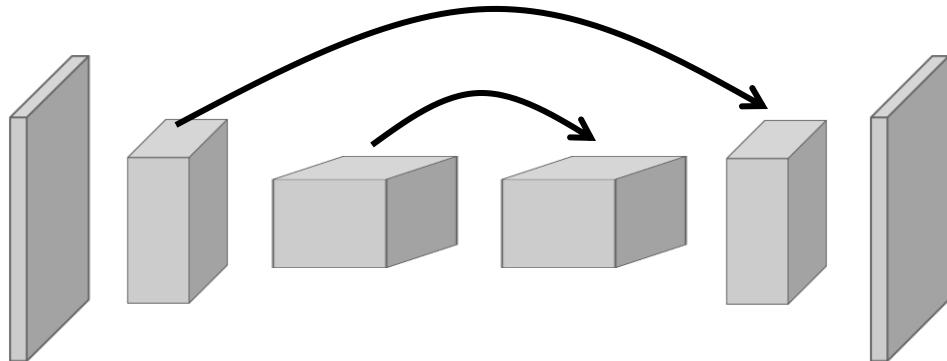
1	0	2	0
0	0	0	0
3	0	4	0
0	0	0	0

Input: 2 x 2

Output: 4 x 4

Max Unpooling

You have to keep track of the locations of the max during maxpooling



Max Pooling

Remember which element was max!

You need to keep
track of where the max elt
was.

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

Input: 4 x 4

Output: 2 x 2

Max Unpooling

Use positions from
pooling layer

1	2
3	4

Rest of the network

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

Input: 2 x 2

Output: 4 x 4

Transpose Convolution

most POWERFUL.

See example

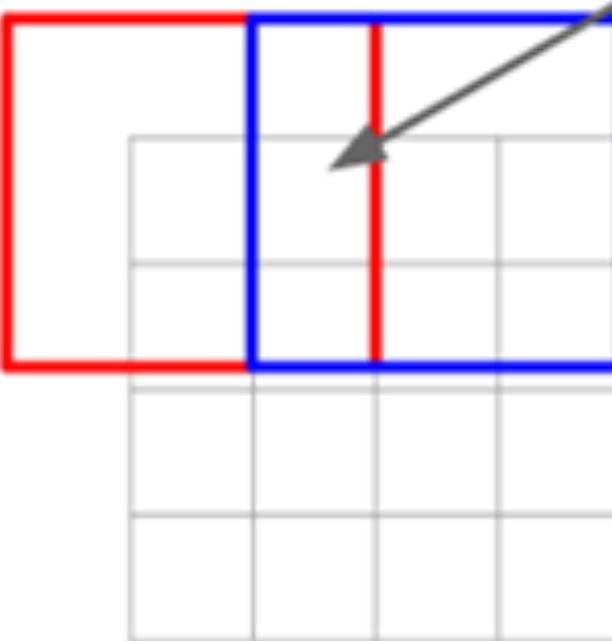
3 x 3 transpose convolution, stride 2 pad 1



Input gives weight for **a 3x3 filter**



Input: 2 x 2



Output: 4 x 4

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

Stride gives ratio between movement in output and input

Sum where output overlaps

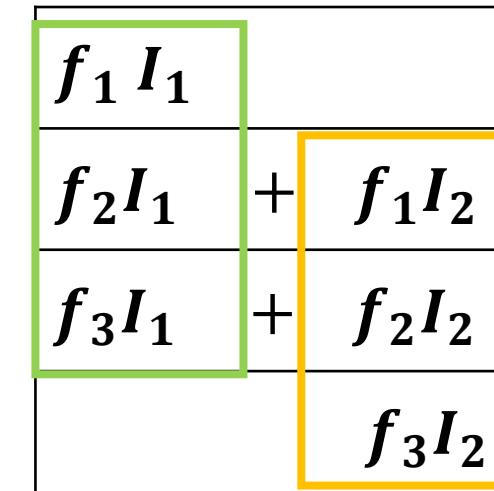
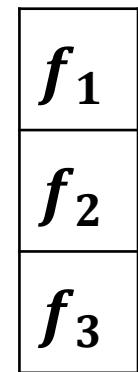
Transpose Convolution

Transpose convolution with **stride 1**

Input 2×1



Filter 3×1



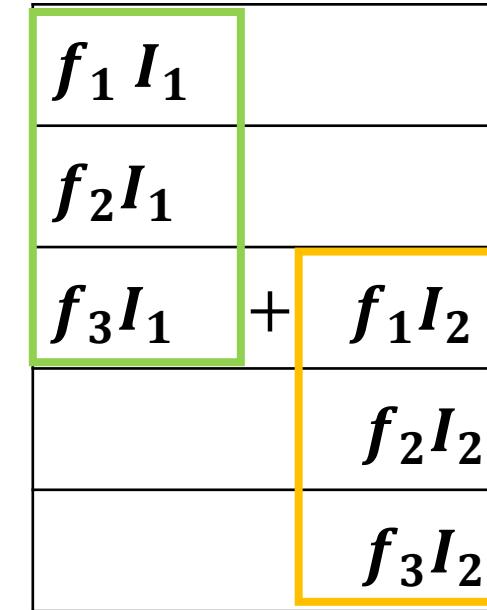
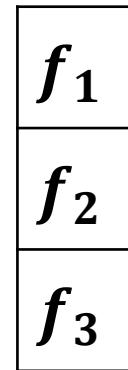
Transpose Convolution

Transpose convolution with stride 2, even larger expansion

Input 2×1



Filter 3×1



Transpose Convolution

Transpose Convolution and upsampling:

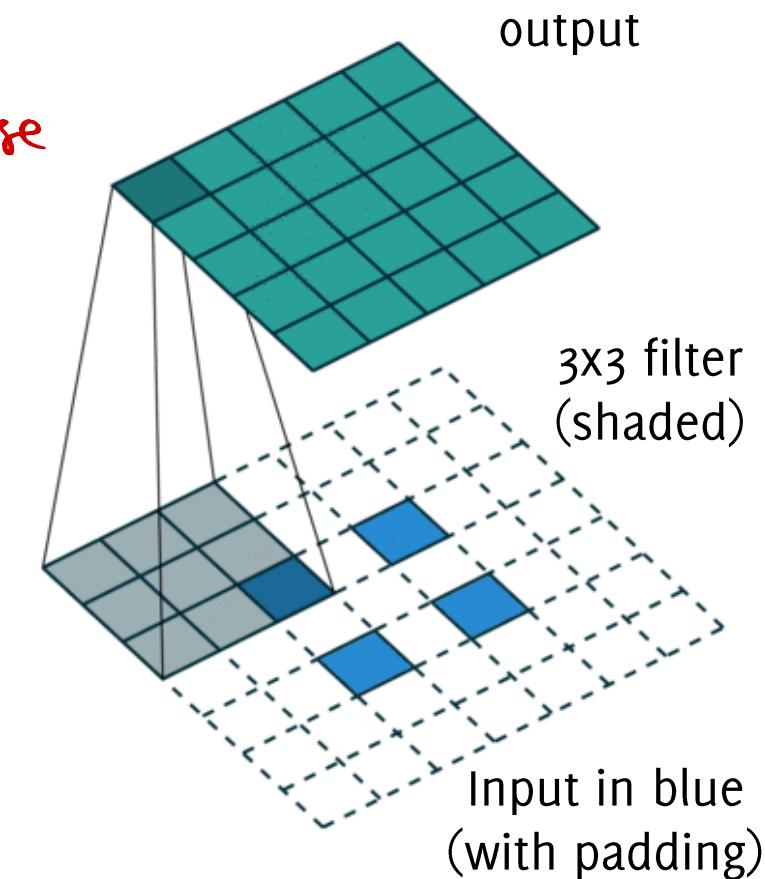
Transpose Convolution can be replaced by a pair of layers:

- Upsampling (e.g. with bed of nails)
- Convolution

Same as transpose convolution!

All in all

- **Upsampling based on convolution** gives more degrees of freedom, since the **filters can be learned**
- Many names for transpose convolution: fractional strided convolution, backward strided convolution, deconvolution (very misleading!!!)



https://github.com/vdumoulin/conv_arithmetic

Semantic Segmentation Networks Training

The categorical cross-entropy loss can be assessed in each and every pixel of the input.

The loss function becomes the sum of pixel-wise losses!

$$\hat{\theta} = \operatorname{argmin}_{x_j \in I} \sum_{x_j \in R} \ell(x_j, \theta)$$

Where x_j are all the pixels in a region R of the input image and the loss is evaluated over the corresponding labels in the annotation for semantic segmentation. Therefore, each image/region provides already a mini-batch estimate for computing gradient.

U-Net: Convolutional Networks for Biomedical Image Segmentation

Olaf Ronneberger, Philipp Fischer, and Thomas Brox

Computer Science Department and BIOSS Centre for Biological Signalling Studies,
University of Freiburg, Germany

ronneber@informatik.uni-freiburg.de,

WWW home page: <http://lmb.informatik.uni-freiburg.de/>

U-Net

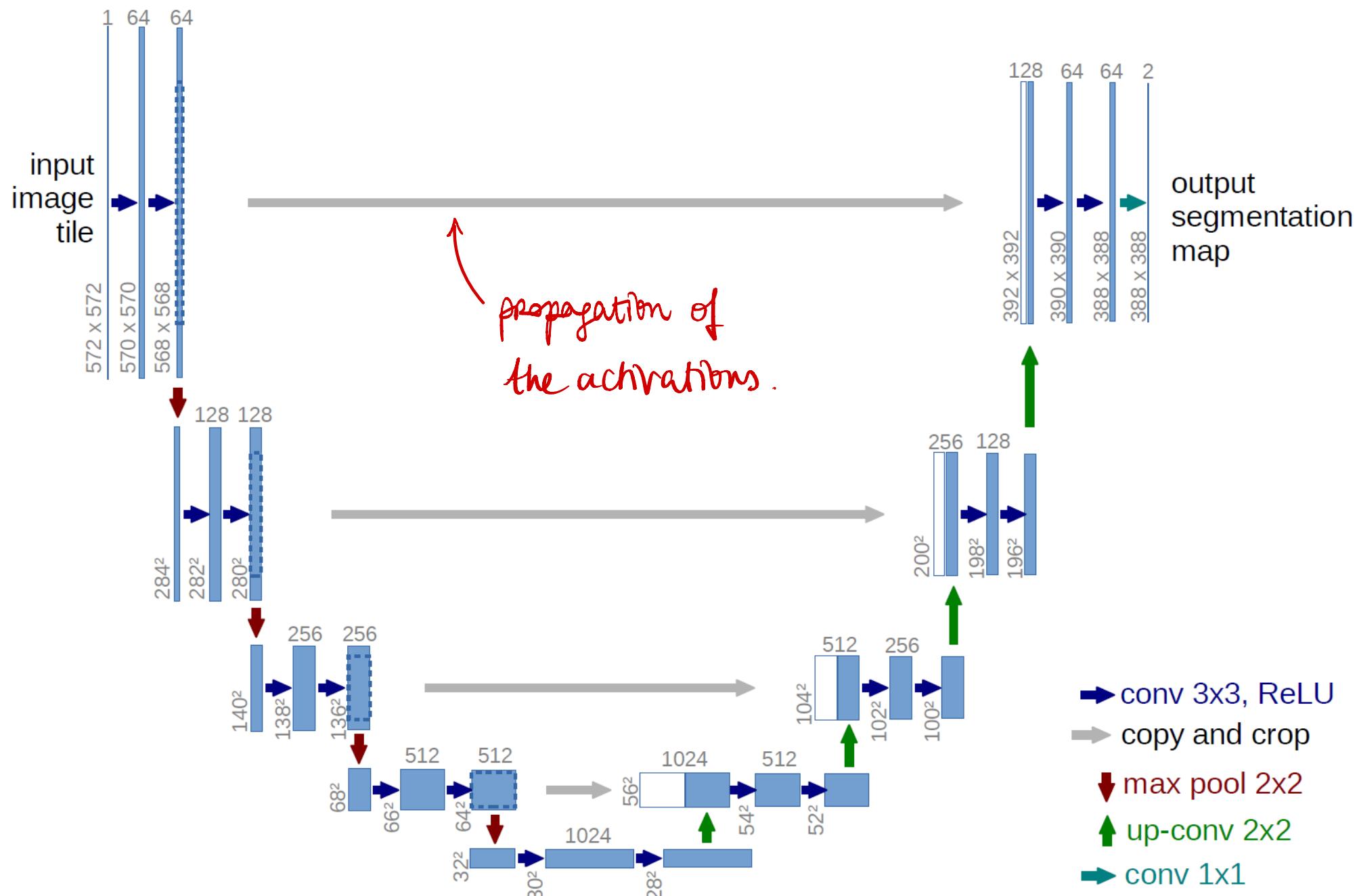
Network formed by:

- A contracting path
- An expansive path

No fully connected layers

Major differences w.r.t. (Long et al. 2015):

- use a large number of feature channels in the upsampling part, while in (long et al. 2015) there were a few upsampling. The network become symmetric
- Use excessive data-augmentation by applying elastic deformations to the training images



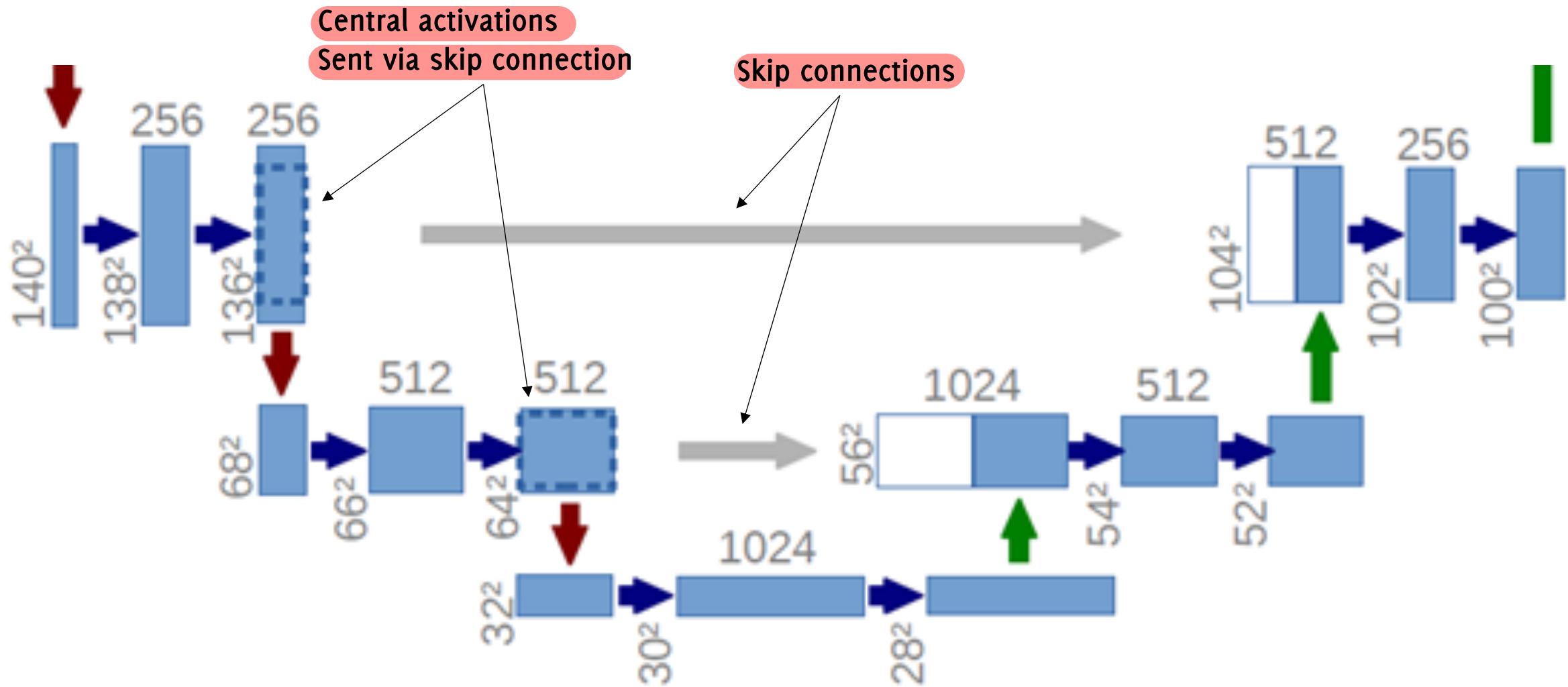
U-Net: Contracting path

Repeats blocks of:

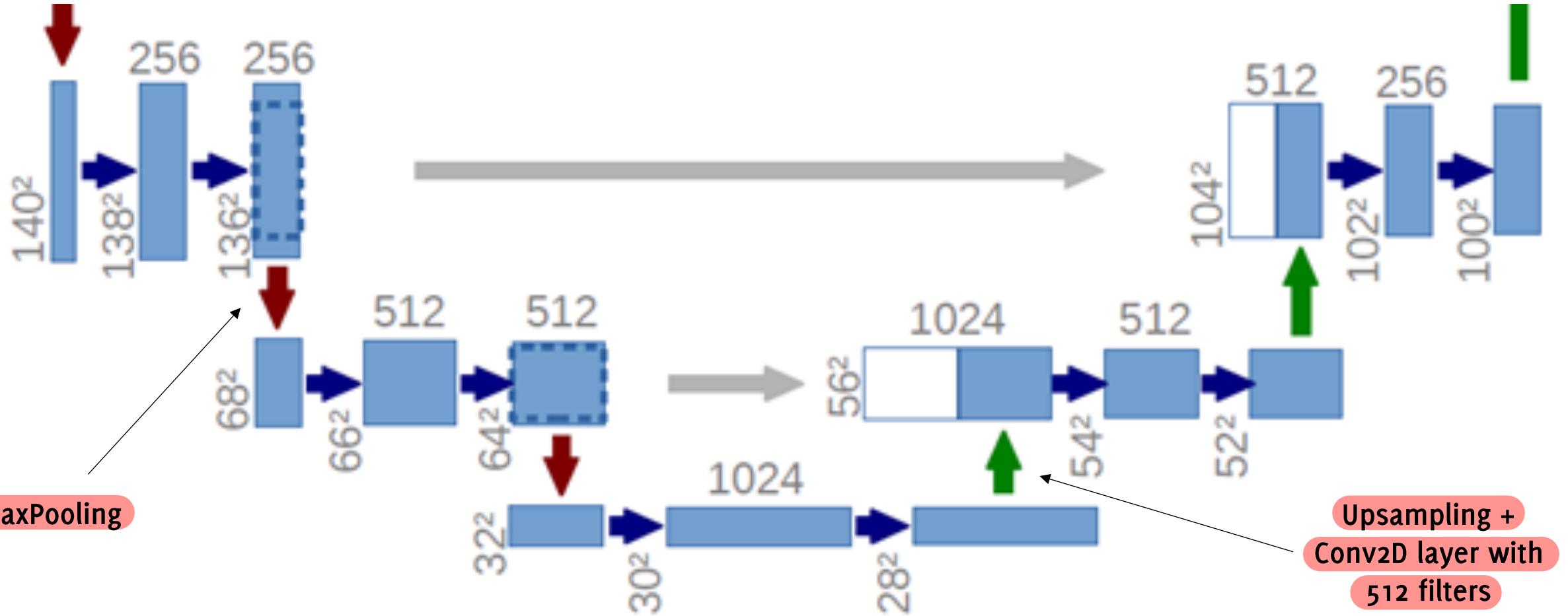
- 3×3 convolution + ReLU ('valid' option, no padding)
- 3×3 convolution + ReLU ('valid' option, no padding)
- Maxpooling 2×2

At each downsampling the number of feature maps is doubled

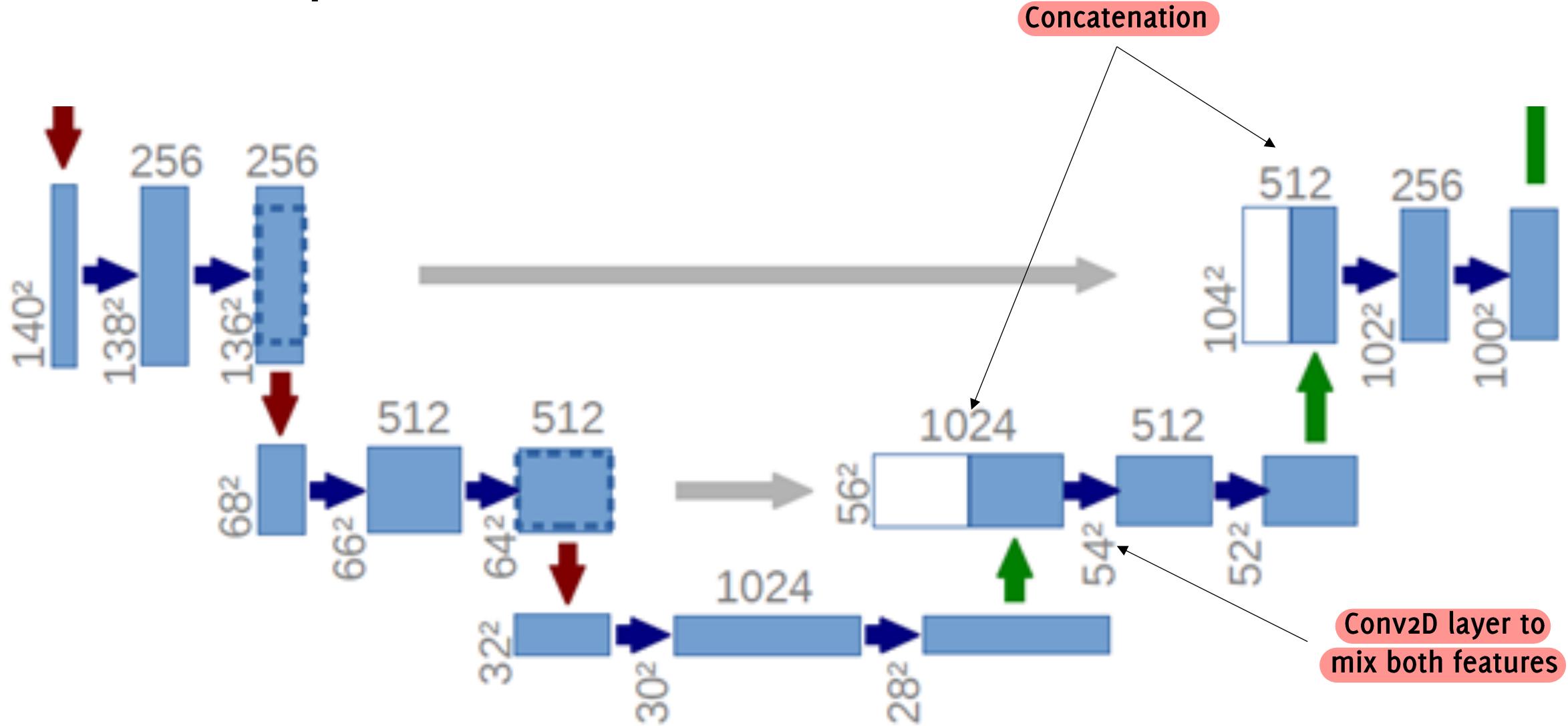
U-Net: Skip connections



U-Net: Skip connections



U-Net: Skip connections



U-Net: Expanding path

Repeats blocks of:

- 2×2 transpose convolution, halving the number of feature maps (but doubling the spatial resolution)
 - Concatenation of corresponding cropped features
 - 3×3 convolution + ReLU
 - 3×3 convolution + ReLU
- 
- Aggregation during
upsampling

U-Net: Network Top

No fully connected layers: there are L convolutions against filters $1 \times 1 \times N$, to yield predictions out of the convolutional feature maps
Output image is smaller than the input image by a constant border

U-Net: Training

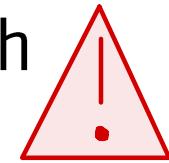
Full-image training by a weighted loss function

$$\hat{\theta} = \min_{\theta} \sum_{x_j} w(x_j) \ell(x_j, \theta)$$

where the weight

$$w(x) = w_c(x) + w_0 e^{-\frac{(d_1(x)+d_2(x))^2}{2\sigma^2}}$$

- w_c is used to balance class proportions (remember no patch resampling in full-image training)
- d_1 is the distance to the border of the closest cell
- d_2 is the distance to the border of the second closest cell



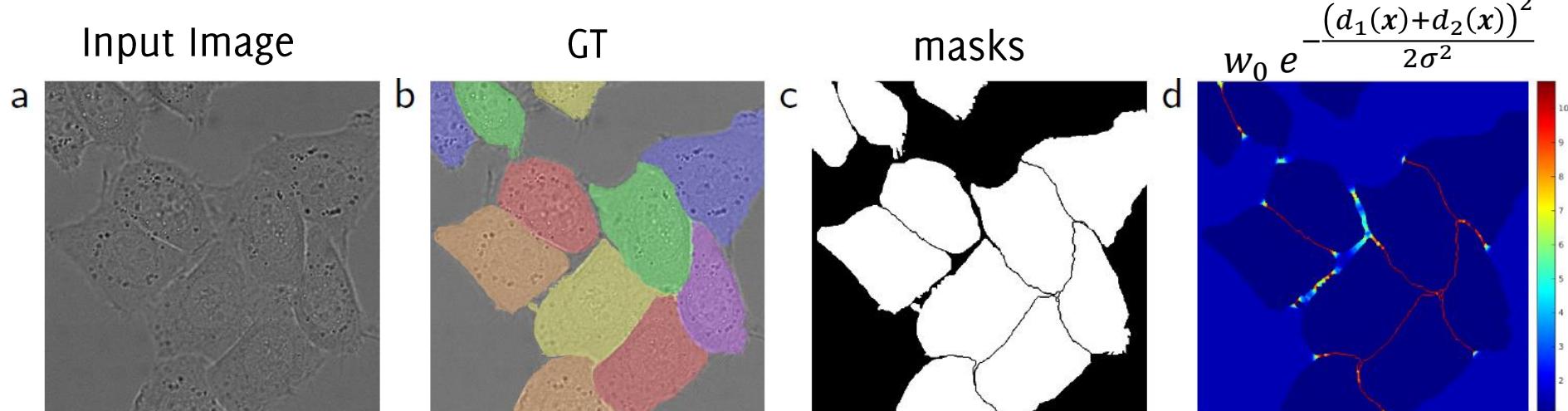
U-Net: Training

Full-image training by a weighted loss function

$$w(x) = w_c(x) + w_0 e^{-\frac{(d_1(x)+d_2(x))^2}{2\sigma^2}}$$

Takes into account class unbalance in the training set

Enhances classification performance at borders of different objects

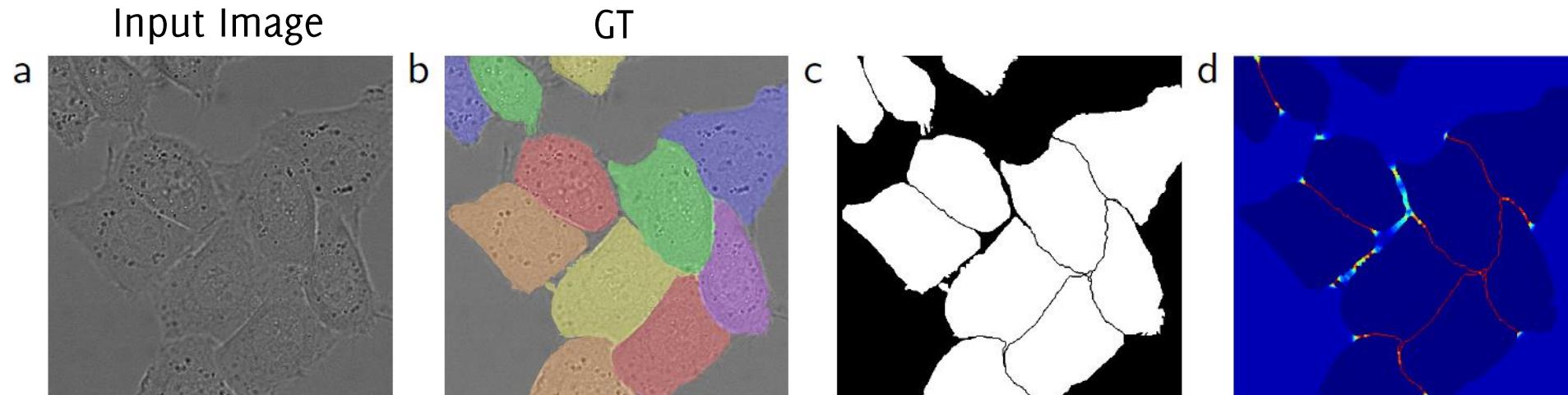


U-Net: Training

Full-image training by a weighted loss function

$$w(x) = w_c(x) + w_0 e^{-\frac{(d_1(x)+d_2(x))^2}{2\sigma^2}}$$

- w_c is used to balance class proportions (remember no patch resampling in full-image training)
- d_1 is the distance to the border of the closest cell
- d_2 is the distance to the border of the second closest cell



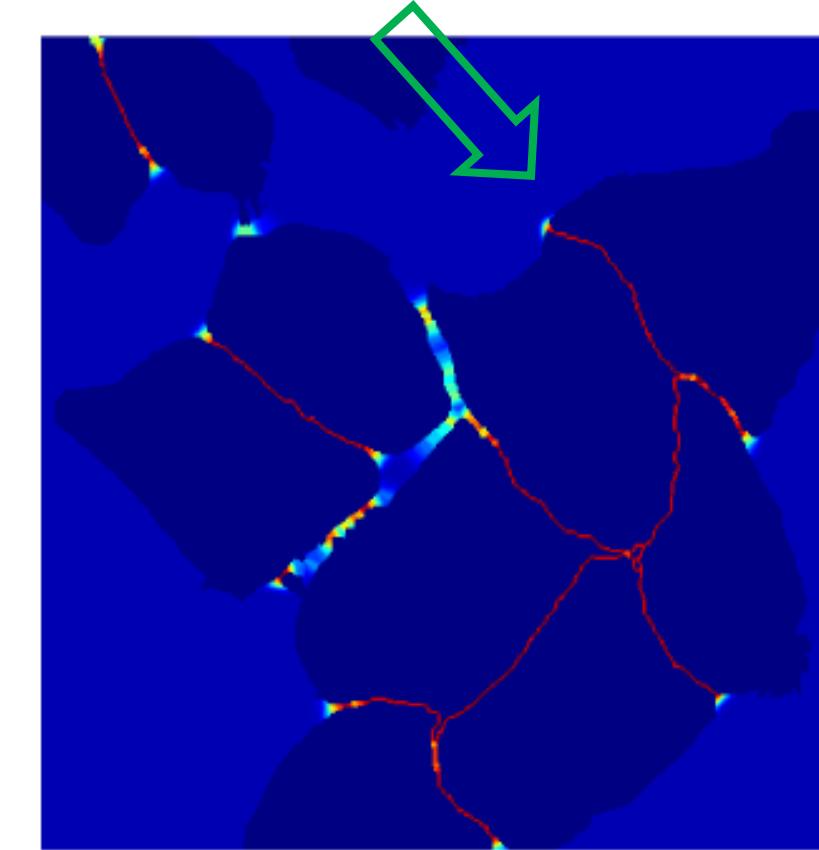
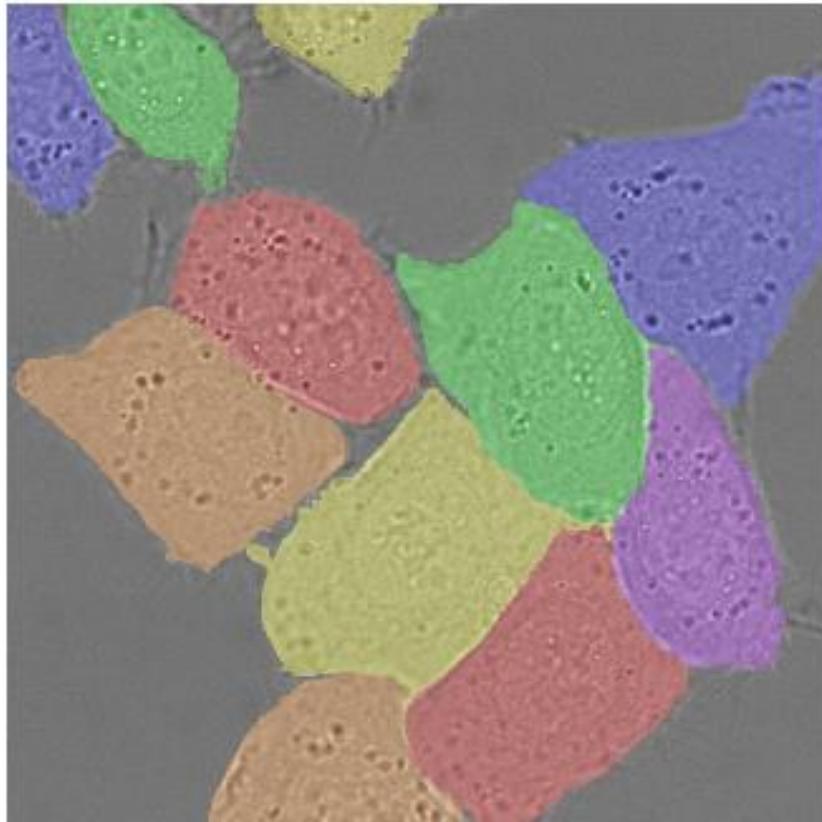
Weights are large when the distance to the first two closest cells is small

U-Net: Training

Full-image training by a weighted loss function

$$w(x) = w_c(x) + w_0 e^{-\frac{(d_1(x)+d_2(x))^2}{2\sigma^2}}$$

This term is large at pixels close to borders delimiting objects of different cells (note that thin borders are red, maximum score)



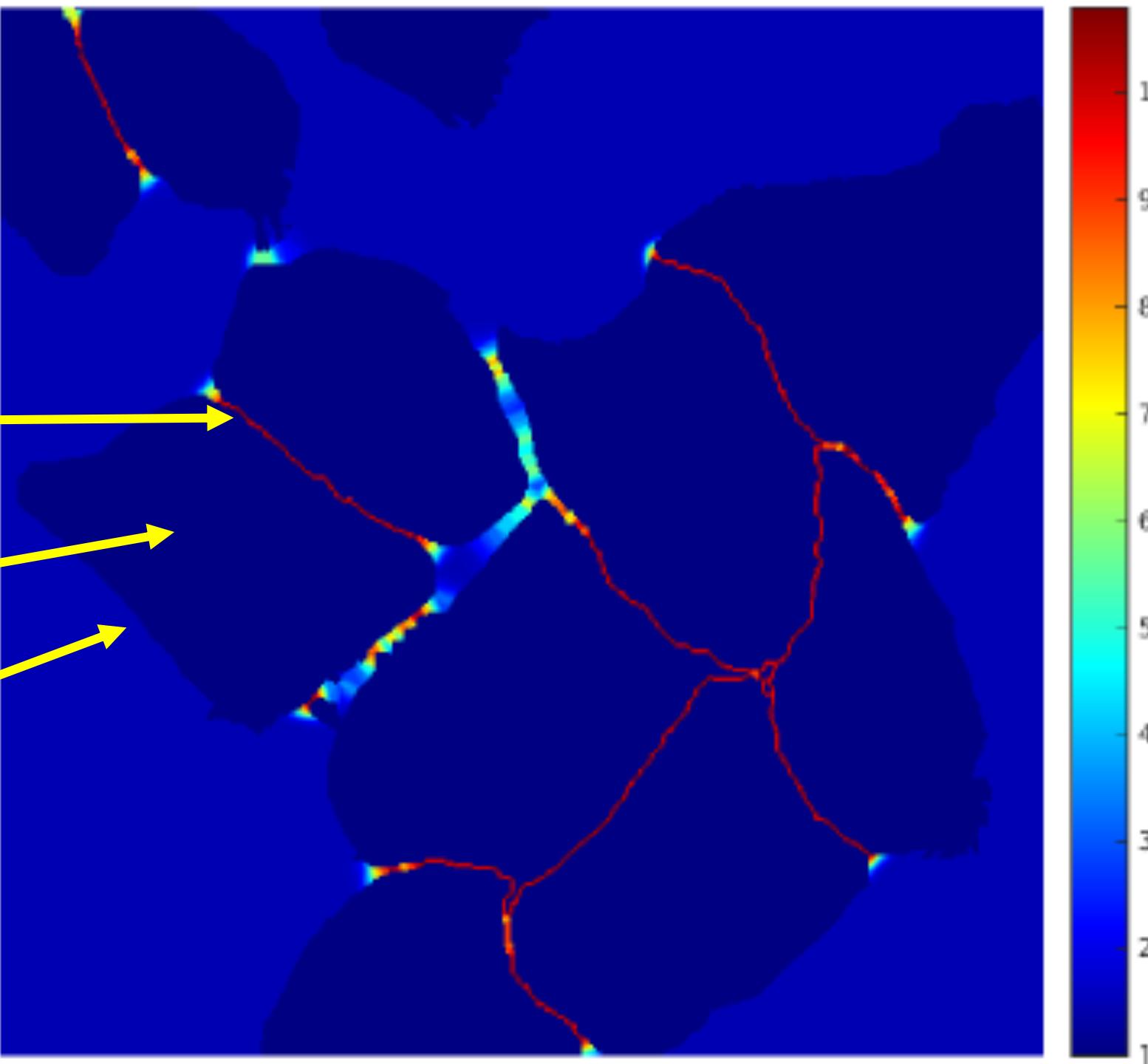
U-Net: Training

$$w(\boldsymbol{x}) = w_c(\boldsymbol{x}) + w_0 e^{-\frac{(d_1(\boldsymbol{x})+d_2(\boldsymbol{x}))^2}{2\sigma^2}}$$

small $d_1(\boldsymbol{x})$ and small $d_2(\boldsymbol{x})$

large $d_1(\boldsymbol{x})$ and $d_2(\boldsymbol{x})$

small $d_1(\boldsymbol{x})$ and large $d_2(\boldsymbol{x})$



U-Net Block in Keras

```
# make sure that within this block spatial sizes are preserved

def unet_block(input_tensor, filters, kernel_size=3, activation='relu', name=''):
    # 2D convolution

    x = tfkl.Conv2D(filters, kernel_size=3, padding='same', name=name+'conv1')(input_tensor)

    # batch normalization (optional)

    x = tfkl.BatchNormalization(name=name+'bn1')(x)

    # activation after batchnorm (this is kind of debated)

    x = tfkl.Activation(activation, name=name+'activation1')(x)

    # 2D convolution

    x = tfkl.Conv2D(filters, kernel_size=3, padding='same', name=name+'conv2')(x)

    # batch normalization (optional)

    x = tfkl.BatchNormalization(name=name+'bn2')(x)

    # activation

    x = tfkl.Activation(activation, name=name+'activation2')(x)

return x
```

U-Net Block in Keras

```
# Forth Downsampling
    down_block_4 = unet_block(d3, 256, name='down_block4_')
    d4 = tfkl.MaxPooling2D()(down_block_4)
    d4 = tfkl.Dropout(0.2, seed=seed)(d4)

# Bottleneck
    bottleneck = unet_block(d4, 512)

# First Upsampling layer
    # add 2D upsampling layer (this has learnable weights)
    u1 = tfkl.UpSampling2D()(bottleneck)
    # add dropout
    u1 = tfkl.Dropout(0.2, seed=seed)(u1)

    # concatenate the output of down_block_4 (the deepest layer in the contractive path) with the first layer in the expanding path.

    # Spatial sizes need to be consistent, otherwise it is not possible to concatenate
    u1 = tfkl.Concatenate()([u1, down_block_4])
    # add a convolutional block (Exactly same architecture as in the contractive path)
    u1 = unet_block(u1, 256, name='up_block1_')
```

Concatenation is meant along the channel dimension, then values are mixed by forthcoming convolutional layers

Example of Research Projects on Semantic Segmentation

In collaboration with

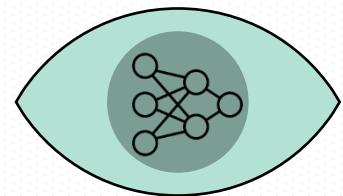


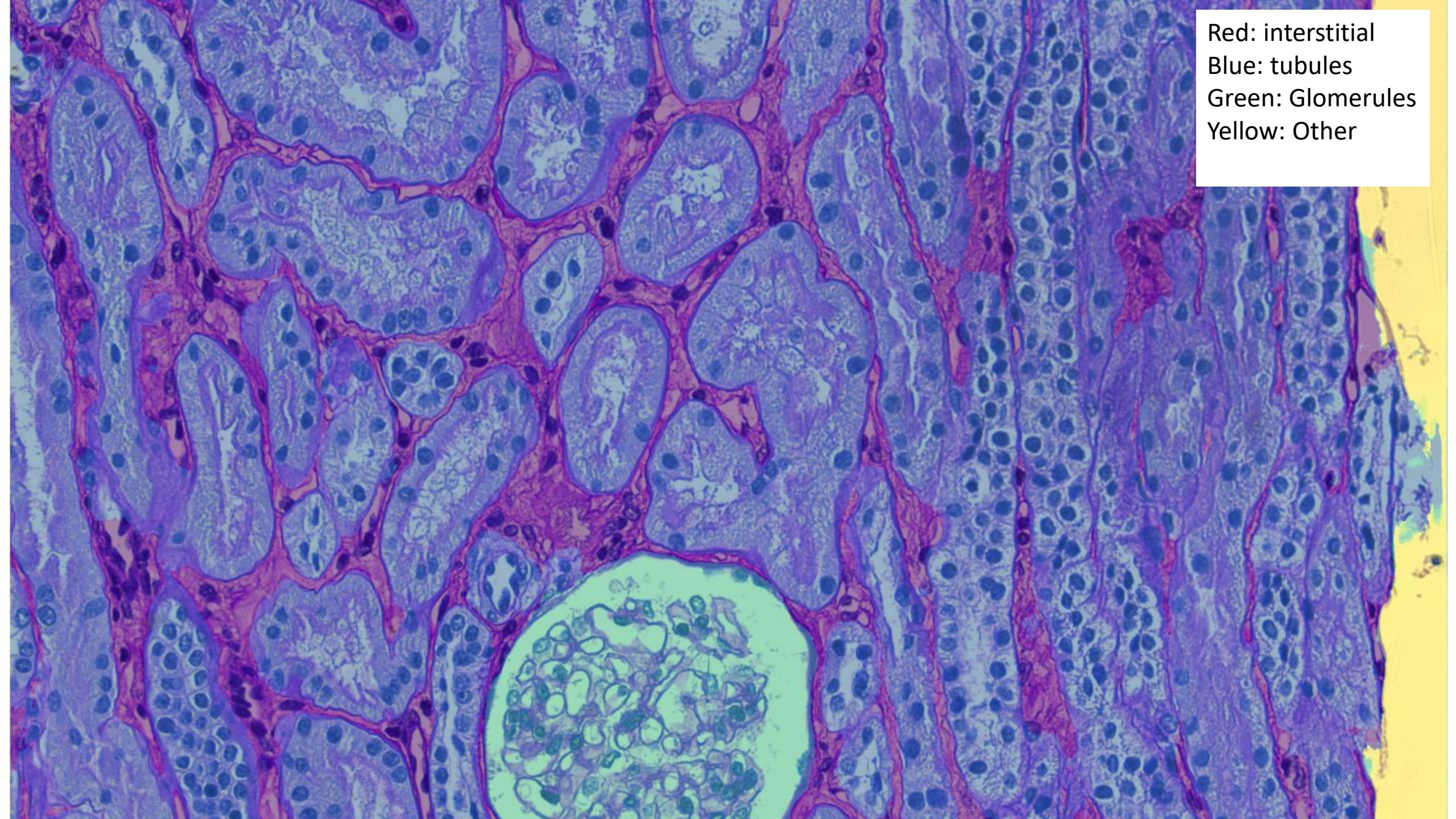
Interstitial Segmentation in Kidney's Biopsies



ISTITUTO DI RICERCHE
FARMACOLOGICHE
MARIO NEGRI · IRCCS

Deep Learning for Visual Recognition





Red: interstitial
Blue: tubules
Green: Glomerules
Yellow: Other

Context

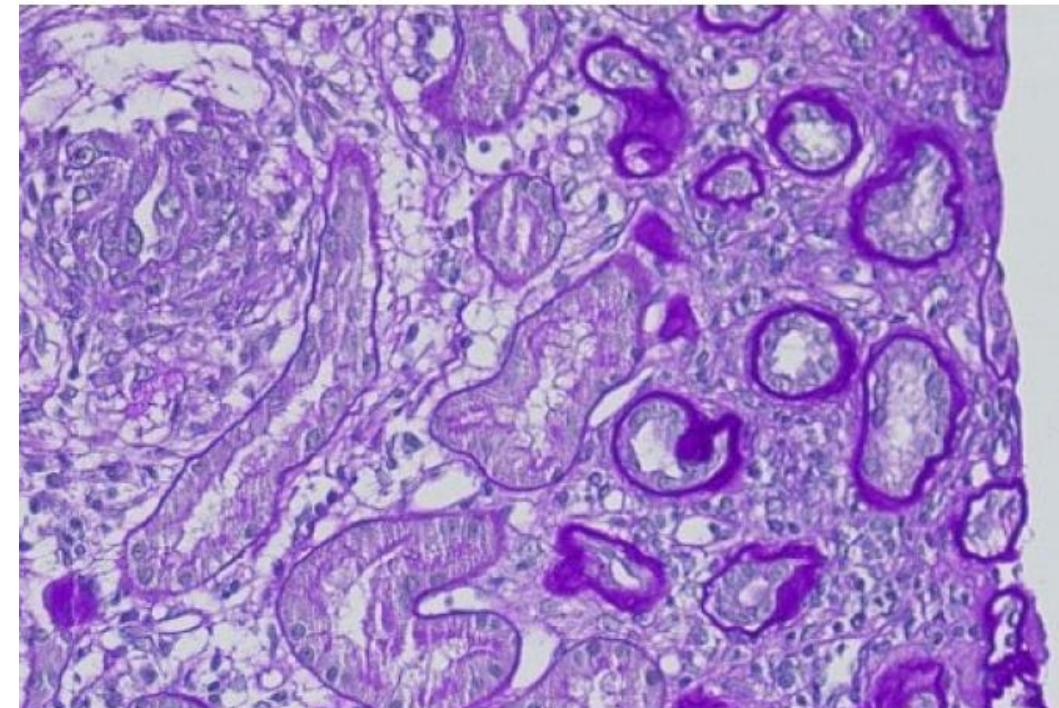
The interstitial area between cells is a biomarker to assess the progression of tumors and pathologies

Goal: Design automatic tools to segment the interstitial region, tubules, nuclei and “other”

Challenges:

Tissues might vary their appearance in different pathologies and different stages

Dense annotations are very difficult and time demanding (more than 2hrs)



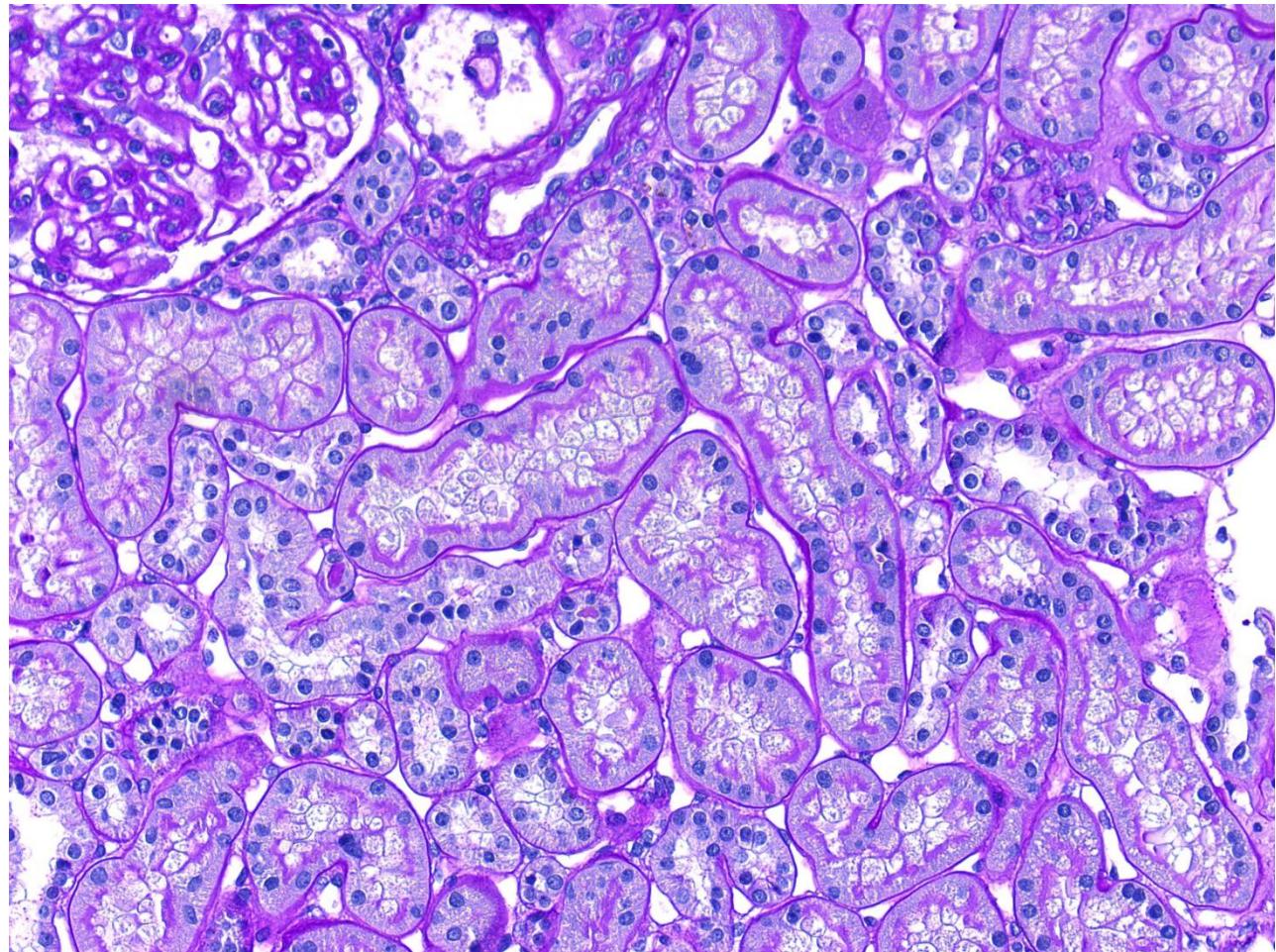
Semantic Segmentation in Medical Images

In medical images (more than natural images) it is often important to quantitatively assess areas (nr of pixels) covered by a specific class.

Here, we want to assess **renal peritubular interstitium** volume in **kidney's biopsies**.

Why?

- Area is correlated with the kidney's chronic pathologies evolution.
- Could be considered as a biomarker to monitor the effectiveness of some drug on kidney's diseases.



In collaboration with



ISTITUTO DI RICERCHE
FARMACOLOGICHE
MARIO NEGRI · IRCCS

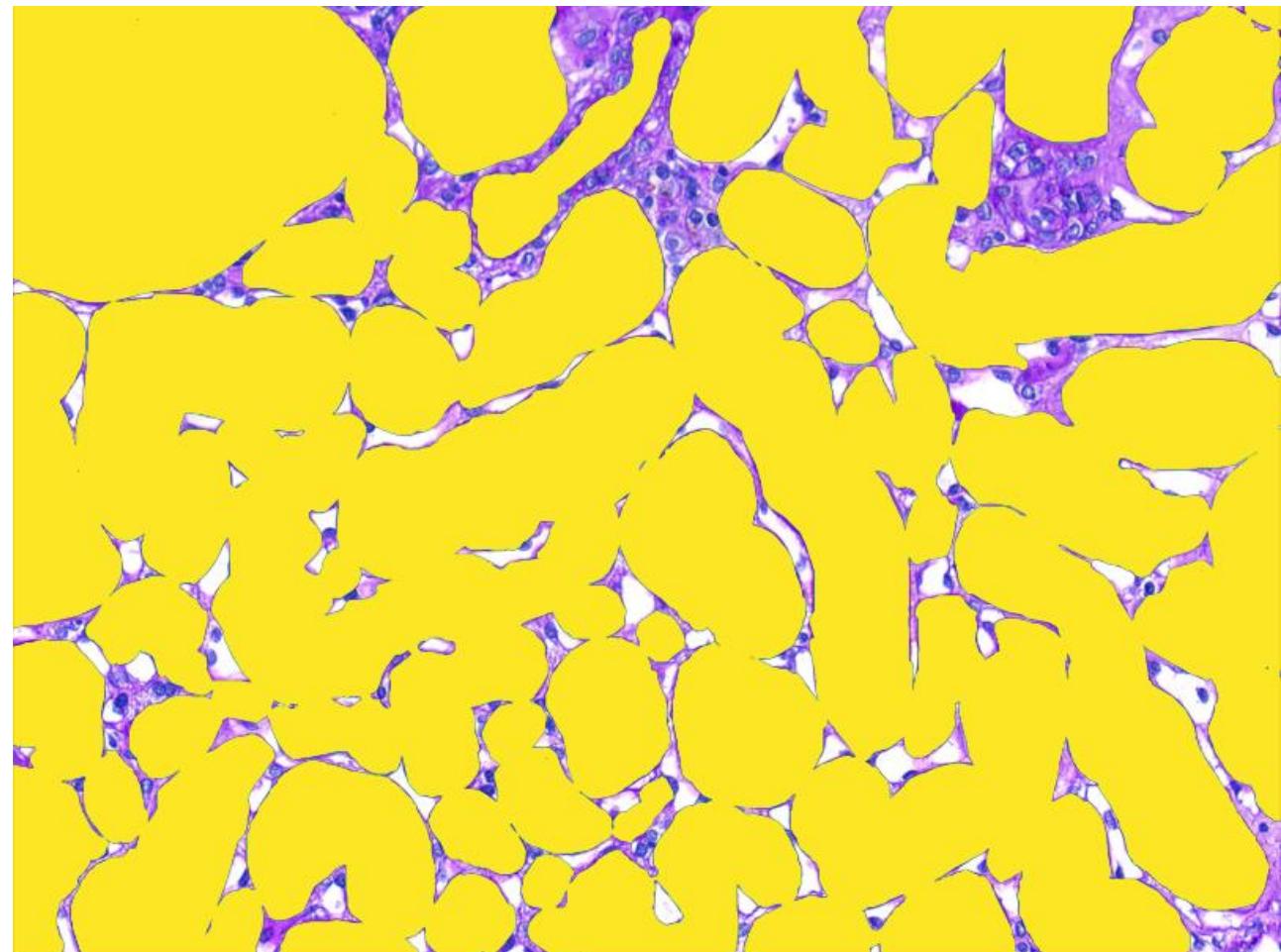
Semantic Segmentation in Medical Images

In medical images (more than natural images) it is often important to quantitatively assess areas (nr of pixels) covered by a specific class.

Here, we want to assess **renal peritubular interstitium** volume in **kidney's biopsies**.

Why?

- Area is correlated with the kidney's chronic pathologies evolution.
- Could be considered as a biomarker to monitor the effectiveness of some drug on kidney's diseases.



In collaboration with

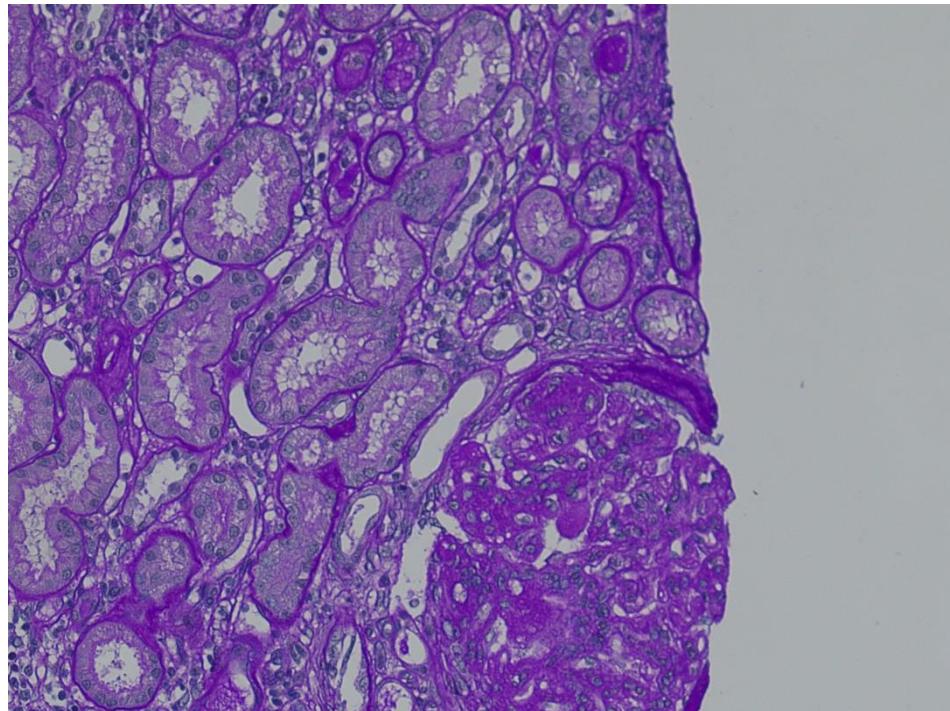


ISTITUTO DI RICERCHE
FARMACOLOGICHE
MARIO NEGRI · IRCCS

Problem

- Semantic segmentationation of kidney biopsises (1920 x 2560 x 3 images)
- 4 classes: ‘Interstitial’ (black), ‘Tubules’ (red), ‘Glomerulus’ (orange), ‘Other’ (yellow)

Original image



Ground truth

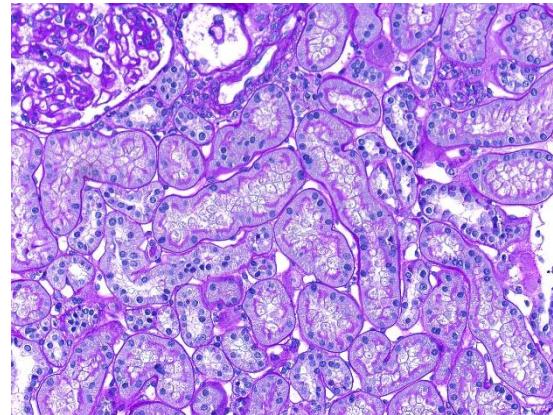


Intrinsic Challenges

SCARSE ANNOTATIONS

Few images available with the correspondent **dense annotations**.
In our application the average annotation time is 2 hours for each image.

Image



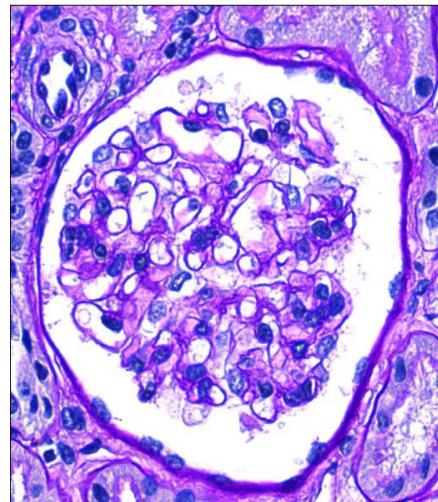
Dense Annotations



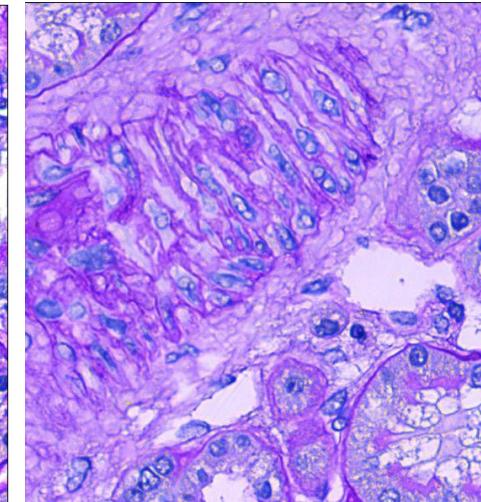
HIGH ANATOMICAL VARIABILITY

Different patients suffering from different pathologies, great anatomical variability even within the same group.

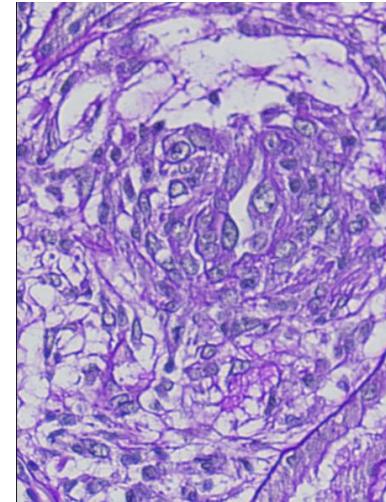
Healthy Subject



Healthy Subject

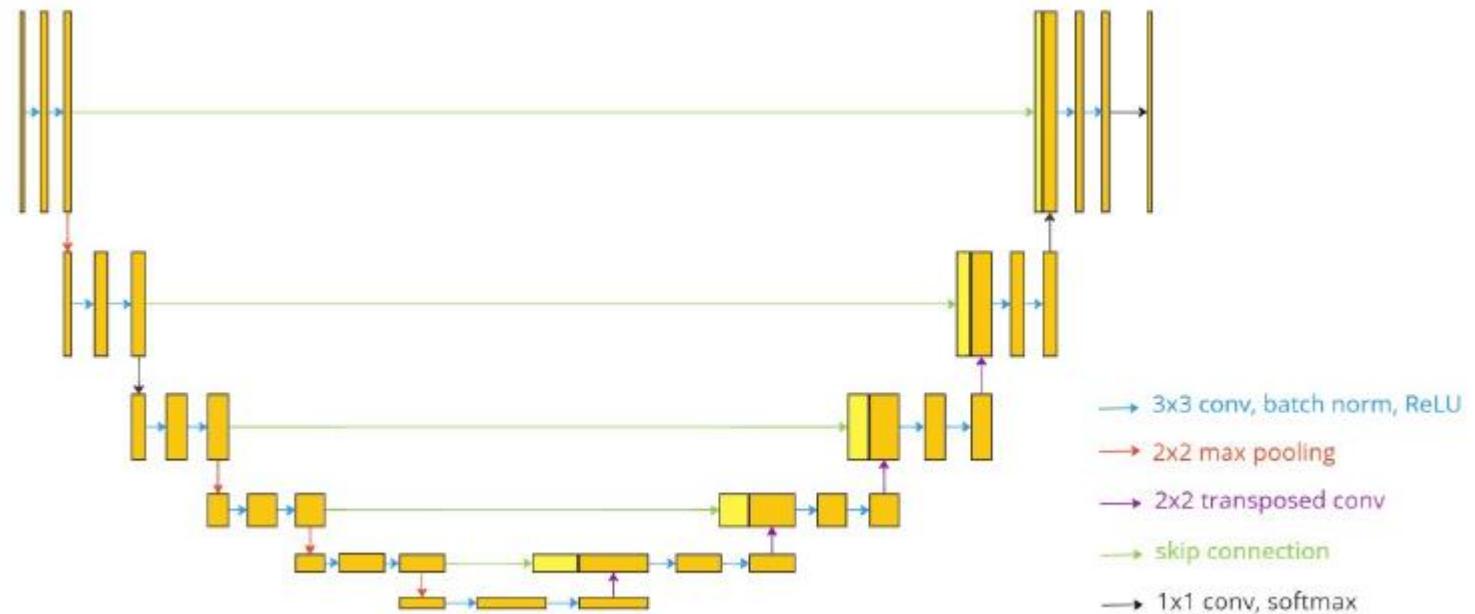


Patient



Semantic Segmentation Network

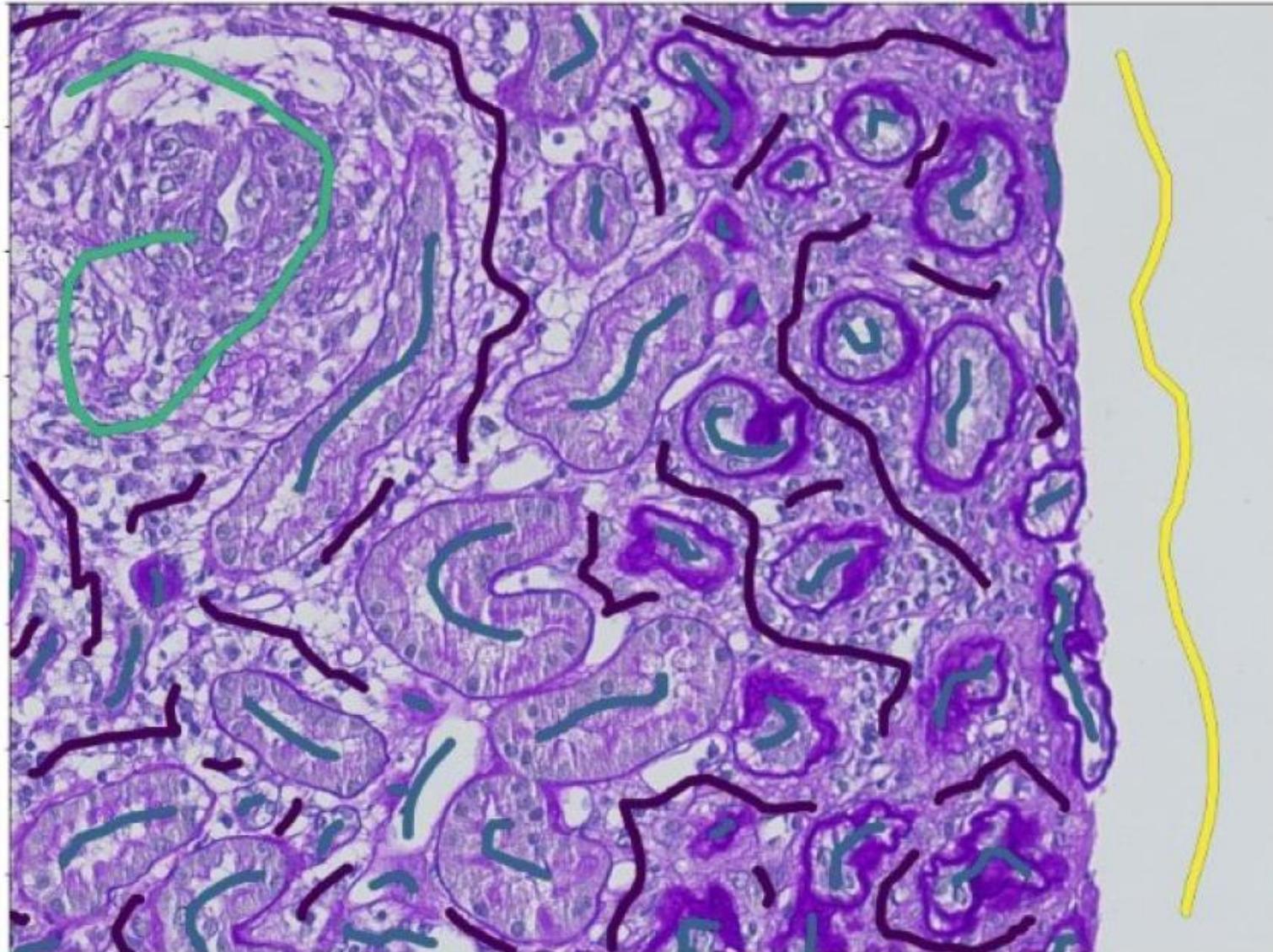
Training a semantic segmentation model can be very demaning when it comes to prepare the annotations



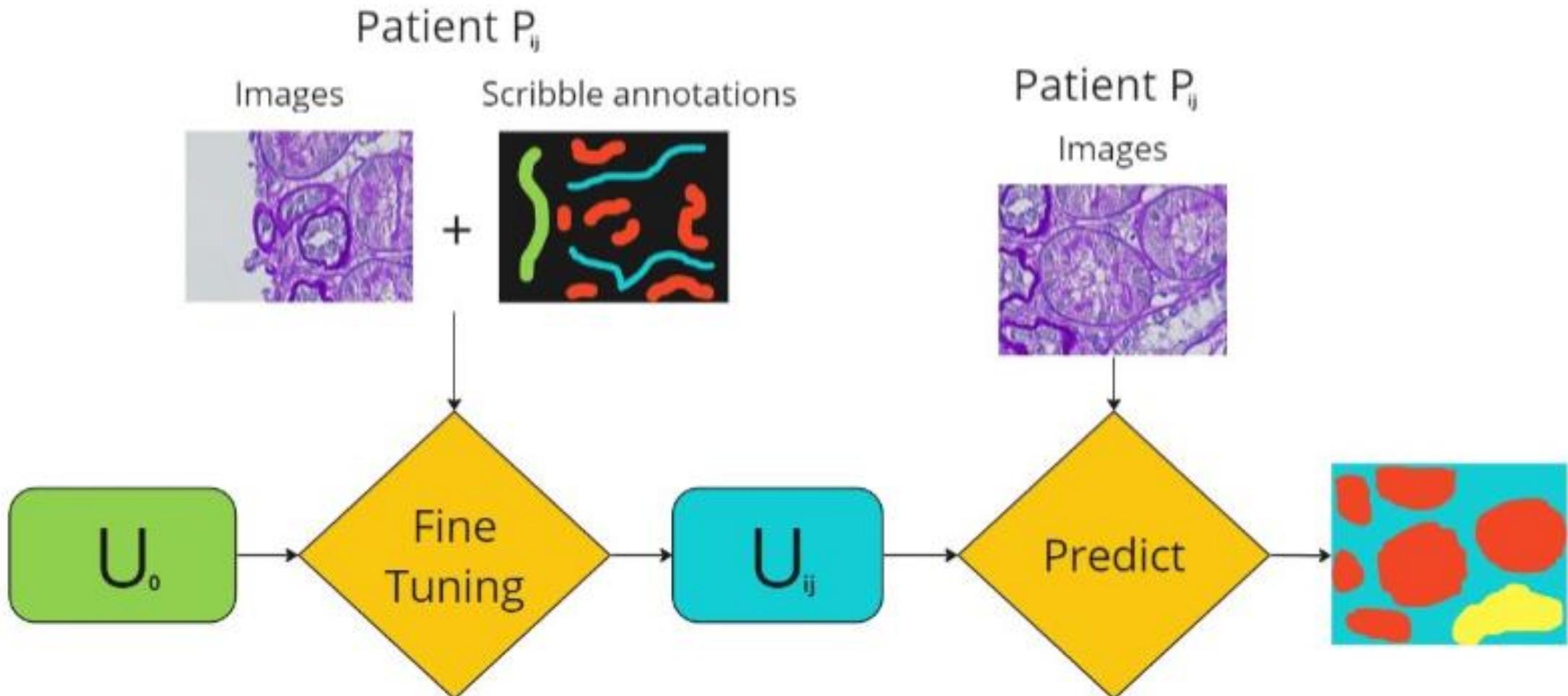
Solution

Train segmentation networks for kidney biopsies using **sparse and fast-to-obtain annotations**.

Scribbles can take less than 5 minutes to prepare (different colors correspond to different classes)

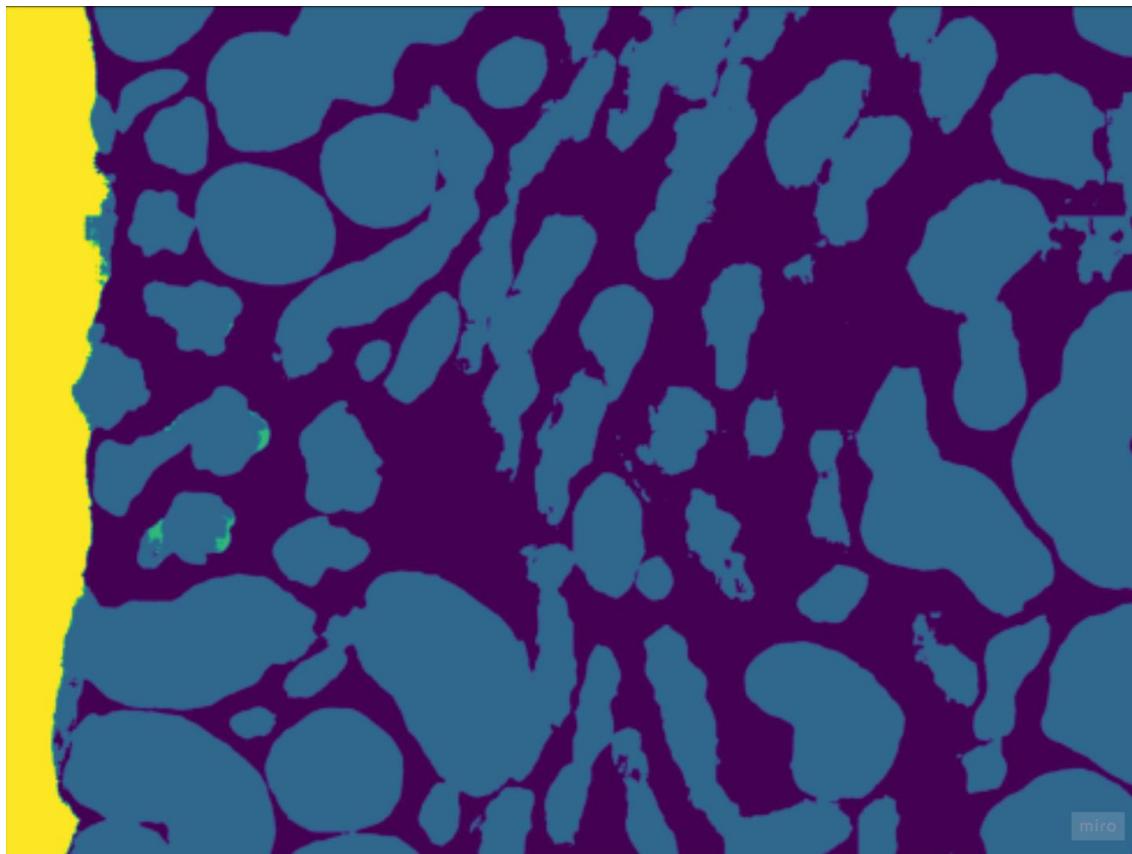


Pathology-Specific fine tuning from spare annotations

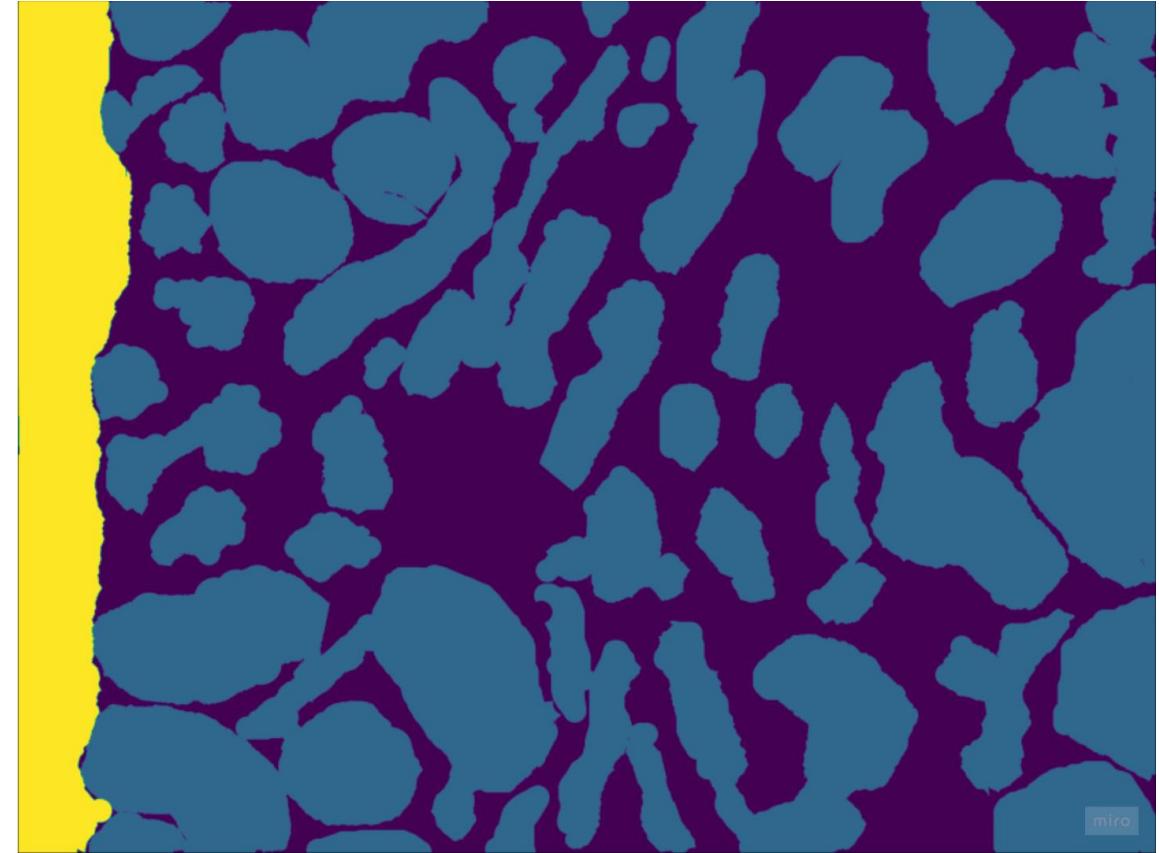


Qualitative Performance

Ground Truth

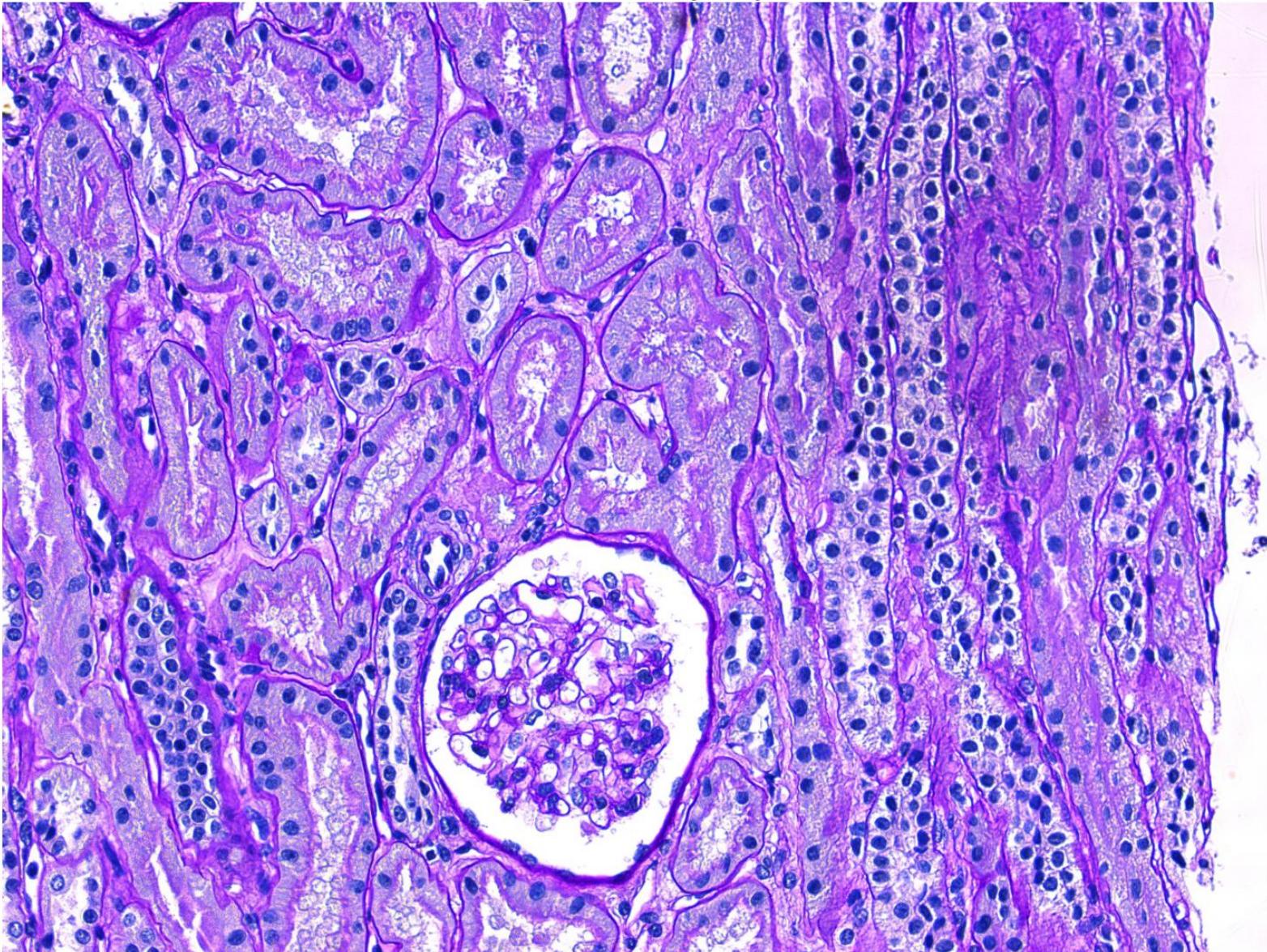


Predictions after Tuning using
sparse annotations



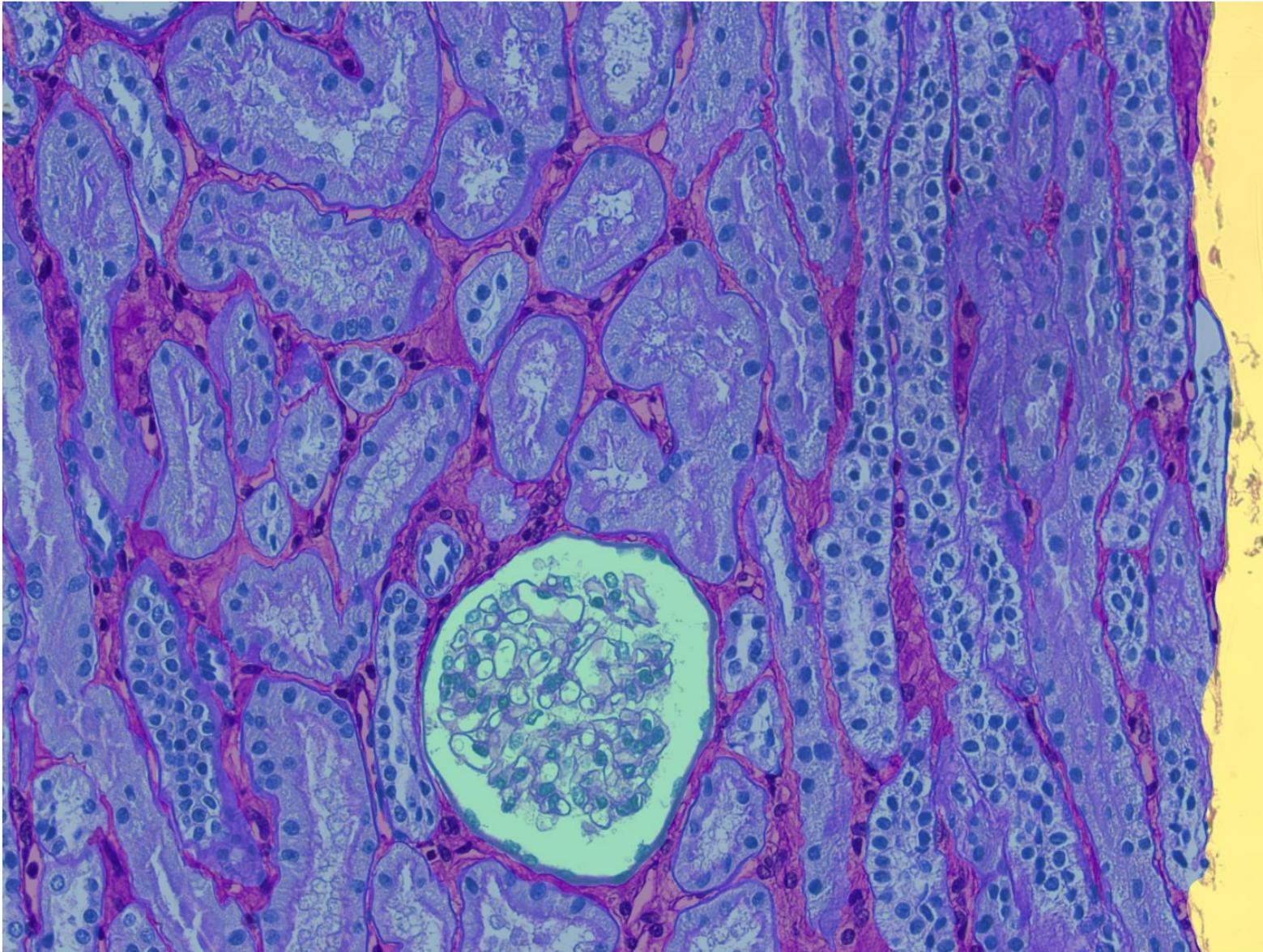
Input Image of a Healty Subject

Image of a healthy subject



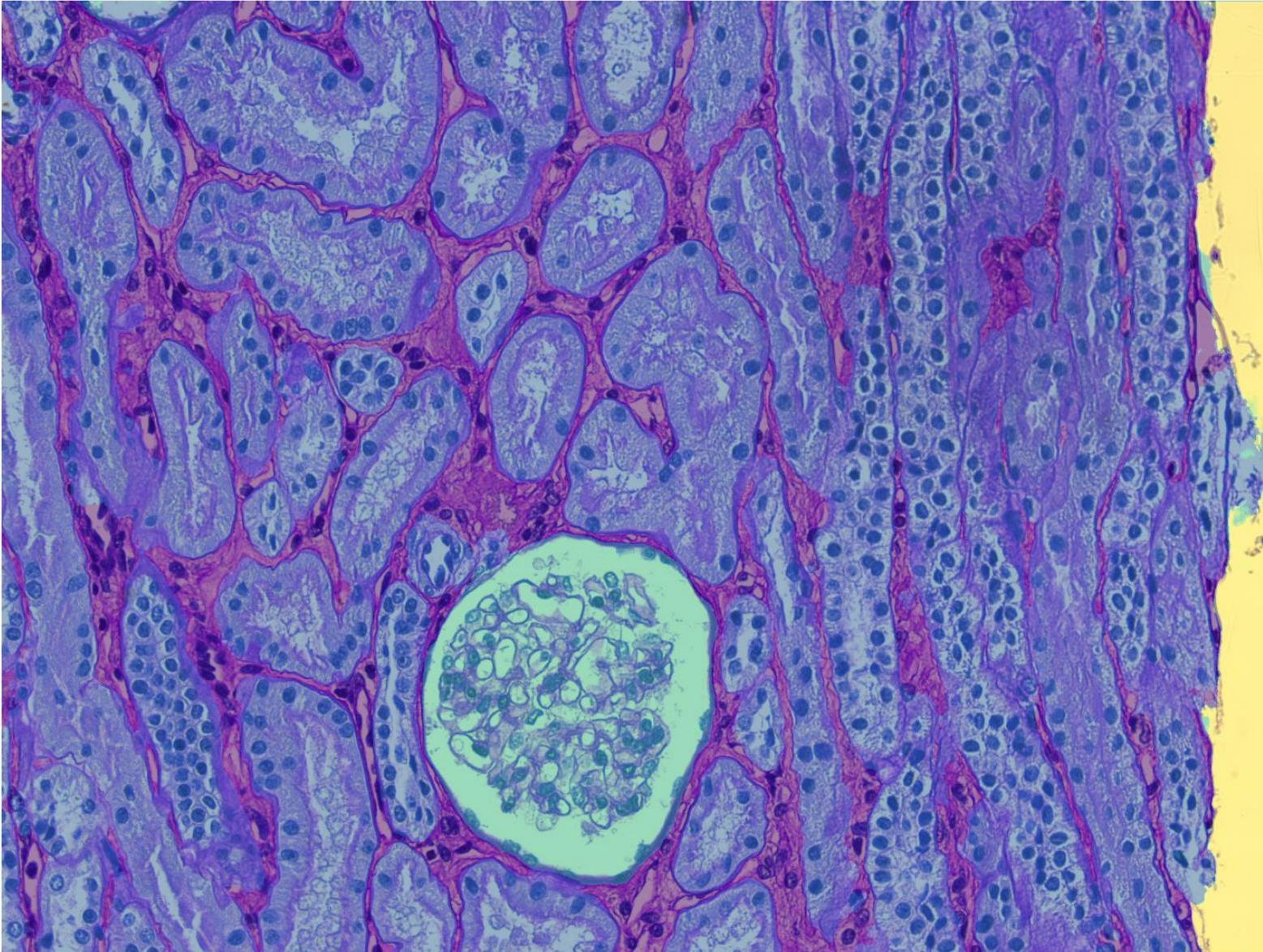
GT Segmentation of a Healthy Subject

Ground truth segmentation overlapped with image of a healthy subject



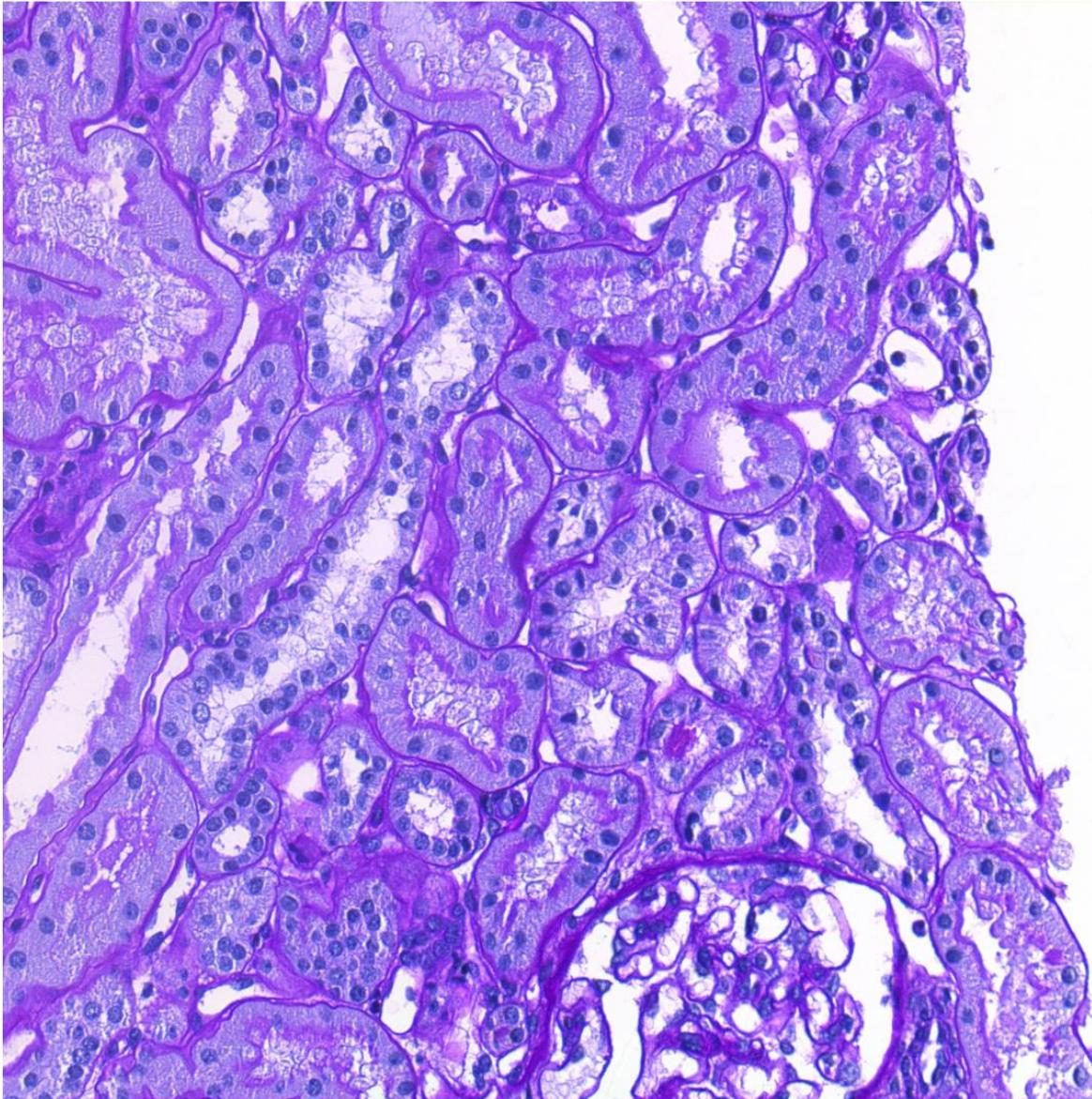
Predicted Segmentation of a Healthy Subject

Predicted segmentation overlapped with image of a healthy subject



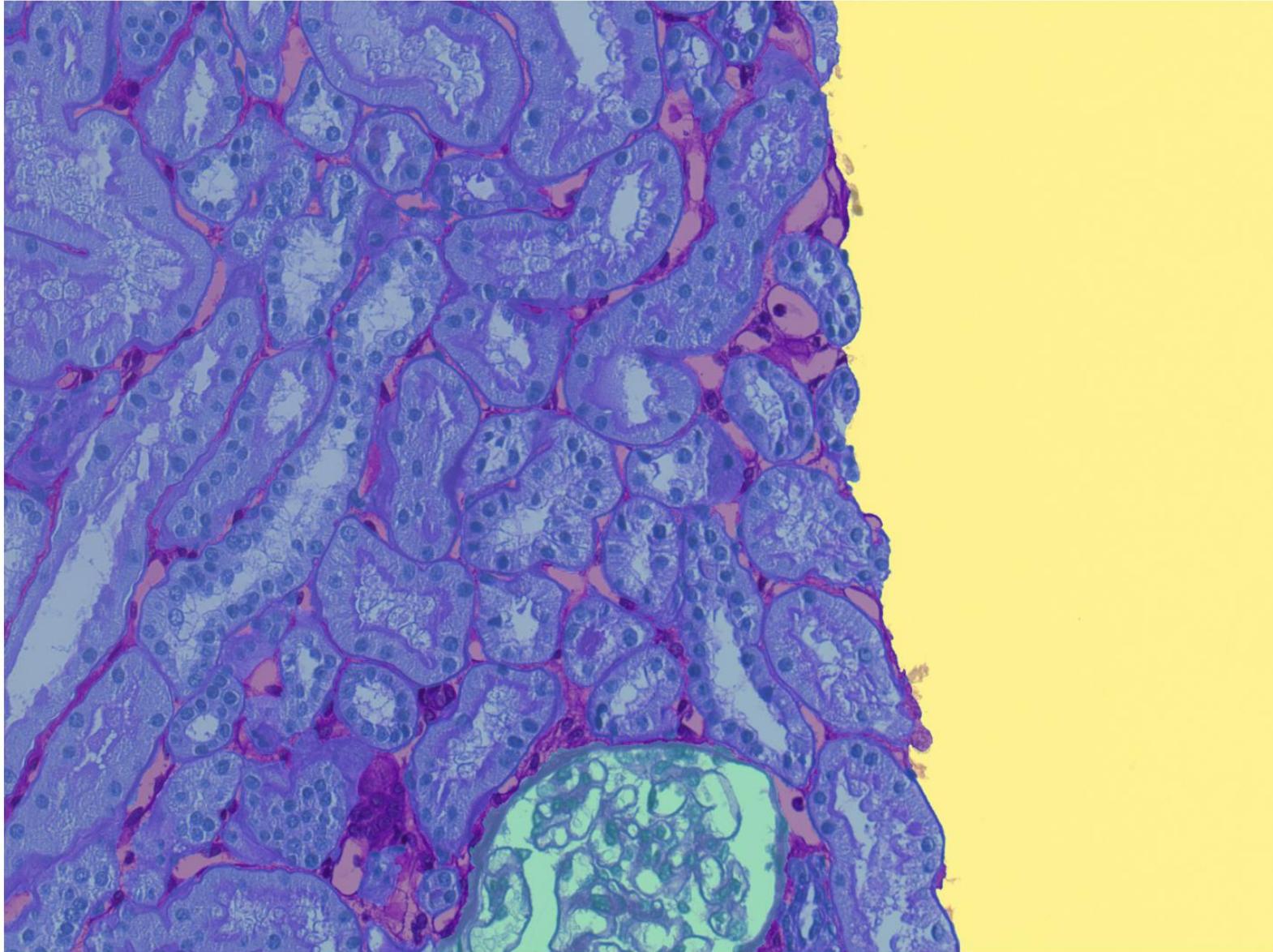
GT Segmentation of a Patient

Image of a patient affected by PAS



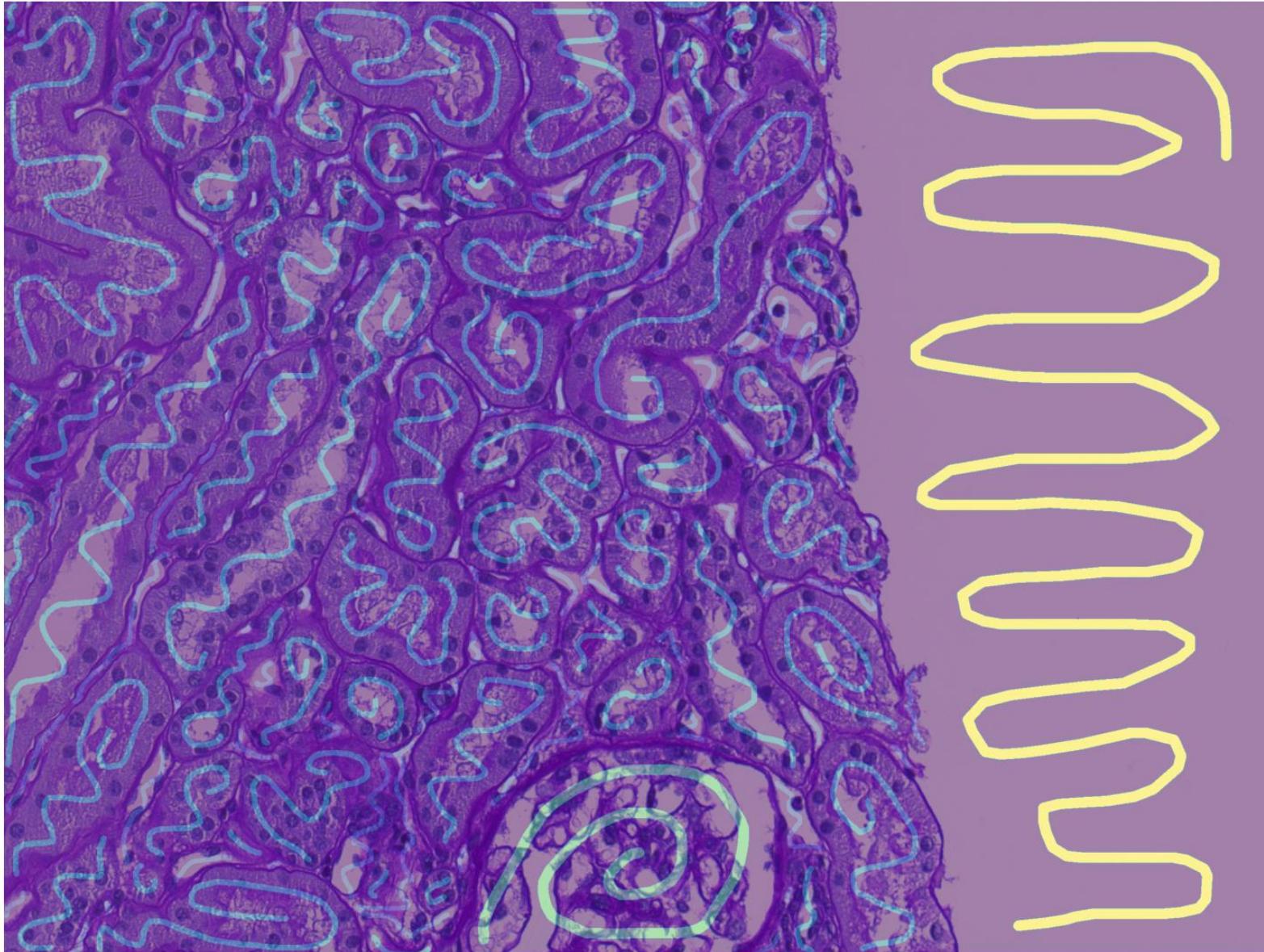
GT Segmentation of a Patient

Ground truth segmentation overlapped with image of a patient affected by PAS



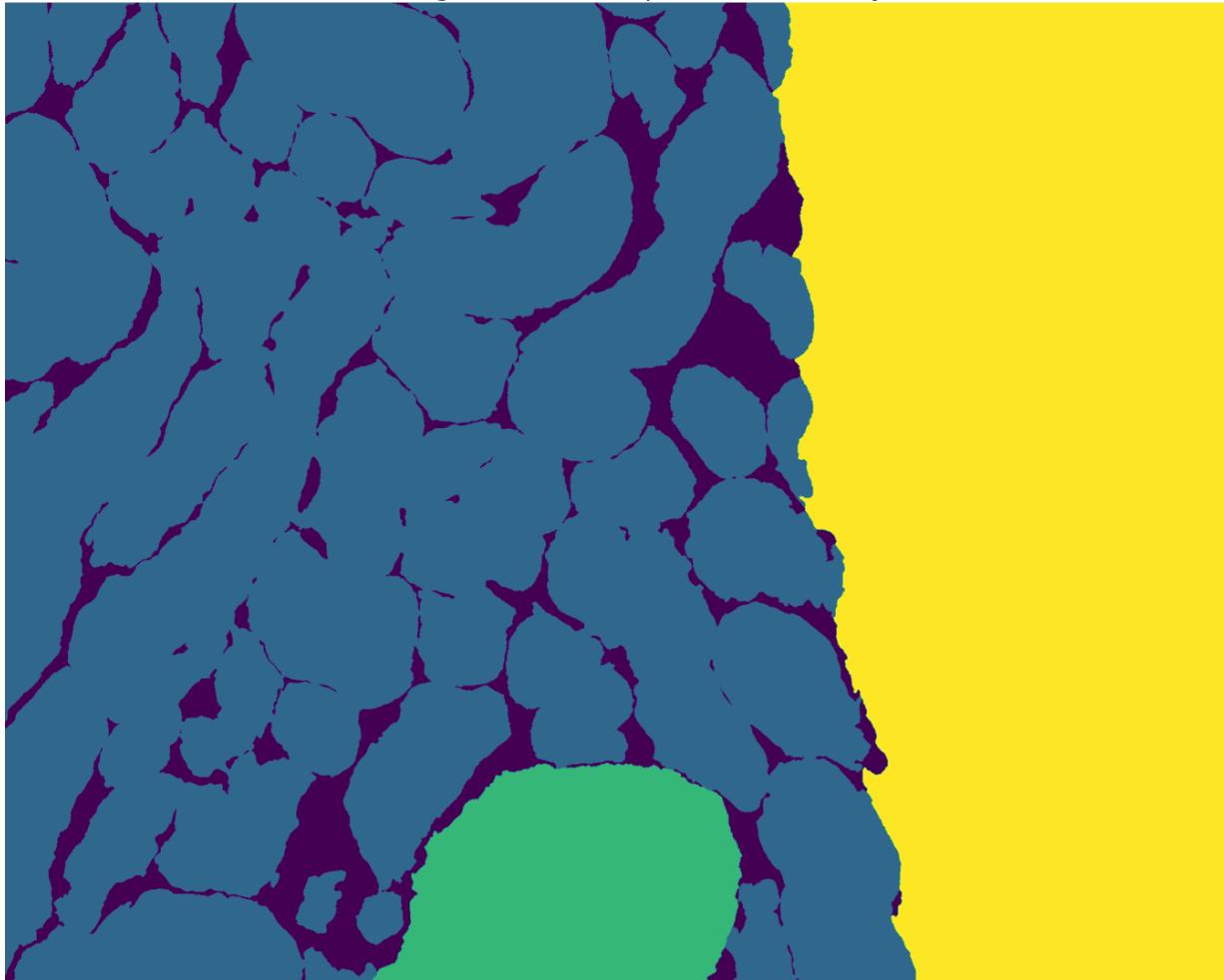
Scribble Annotations for a Patient

Scribble annotation overlapped with image of a patient affected by PAS

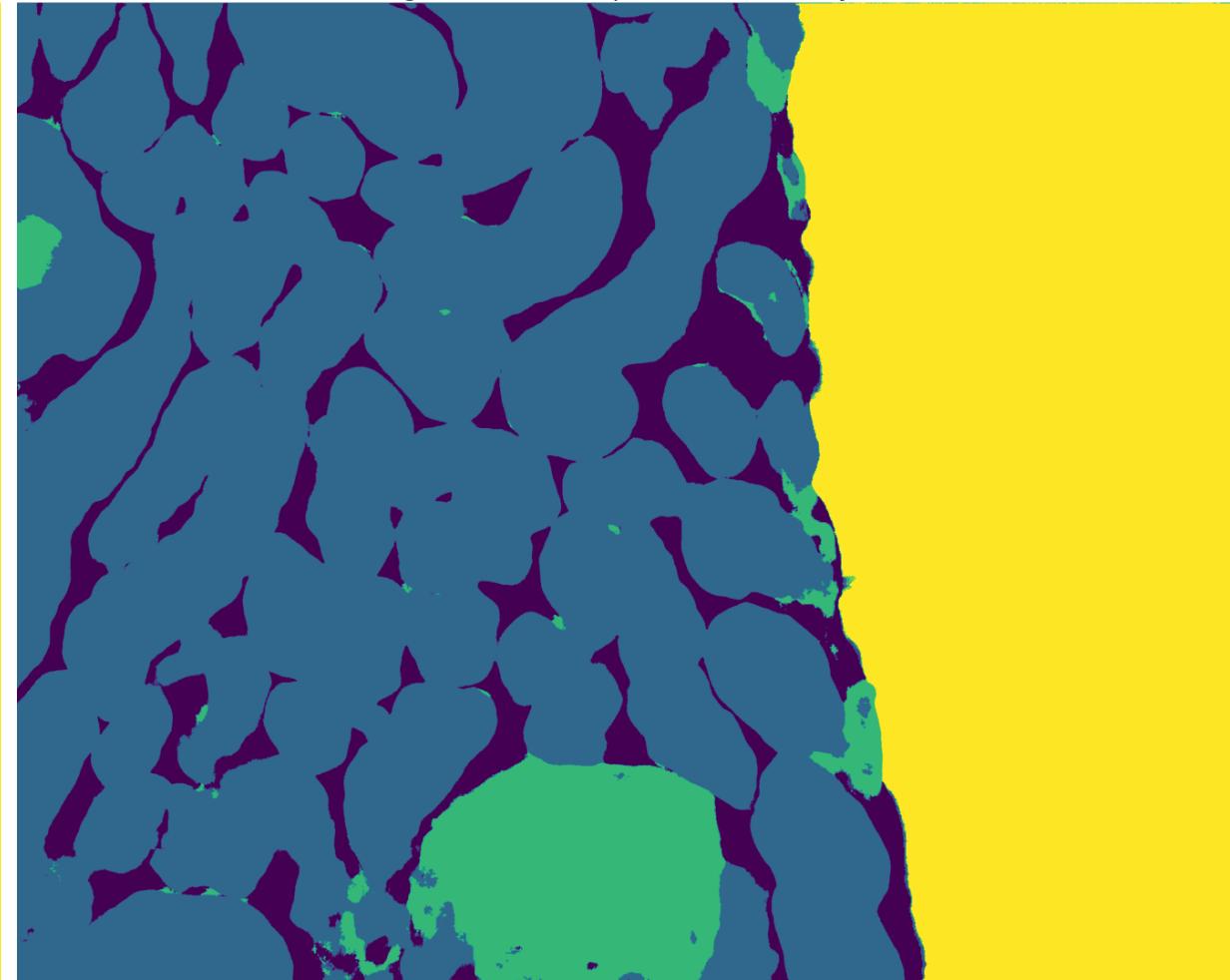


Predictions vs GT for a Patient

Ground truth segmentation of a patient affected by PAS



Predicted segmentation of a patient affected by PAS

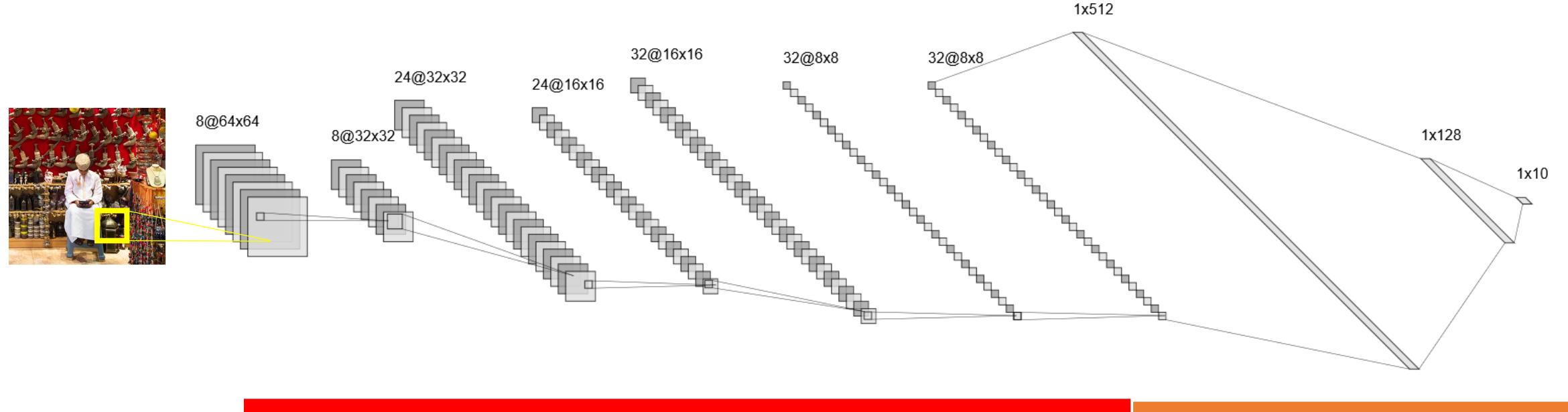


Fully Convolutional Networks

What happens when feeding the network with
an image having different size?

Convolutional Neural Networks (CNN)

The typical architecture of a convolutional neural network



CNNs are meant to process input of a fixed size (e.g. 200 x 200).

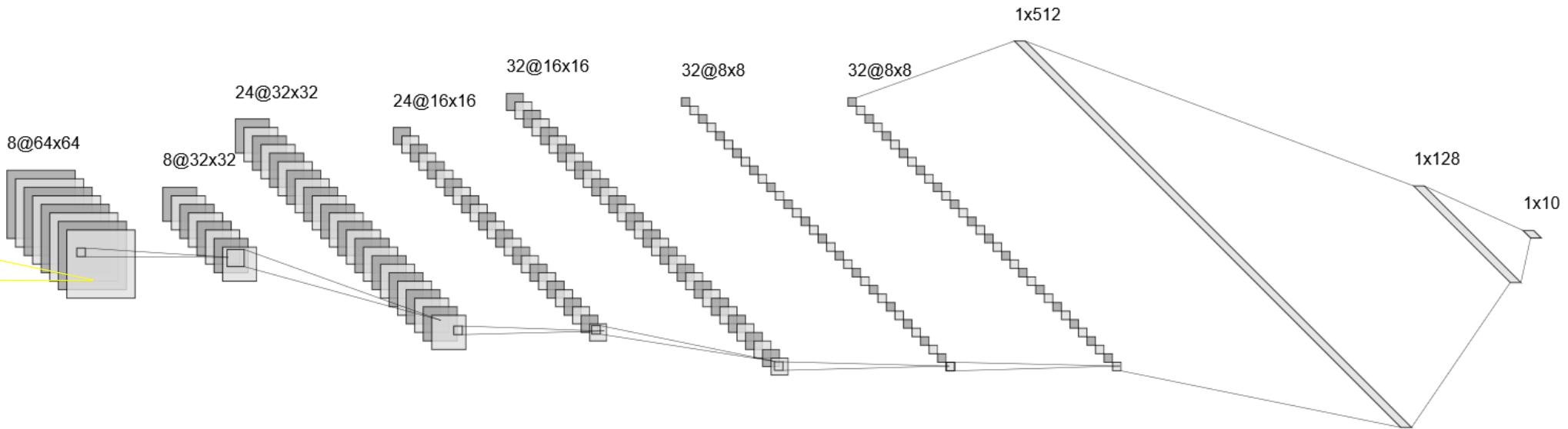
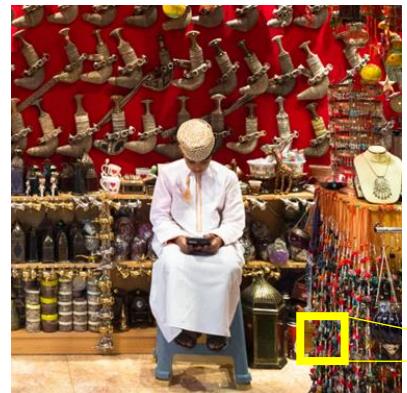
The **convolutional and subsampling layers** operate in a sliding manner over image having arbitrary size

The **fully connected** layer constrains the input to a fixed size.

Convolutional Neural Networks (CNN)

Assume the latent space ends in 1×1 spatial extent, no flattening needed (will remove this assumption later)

$\text{size} = 1 \times 1 \times N$



?

What happens when we feed a larger image to the network?

Convolutional Neural Networks (CNN)

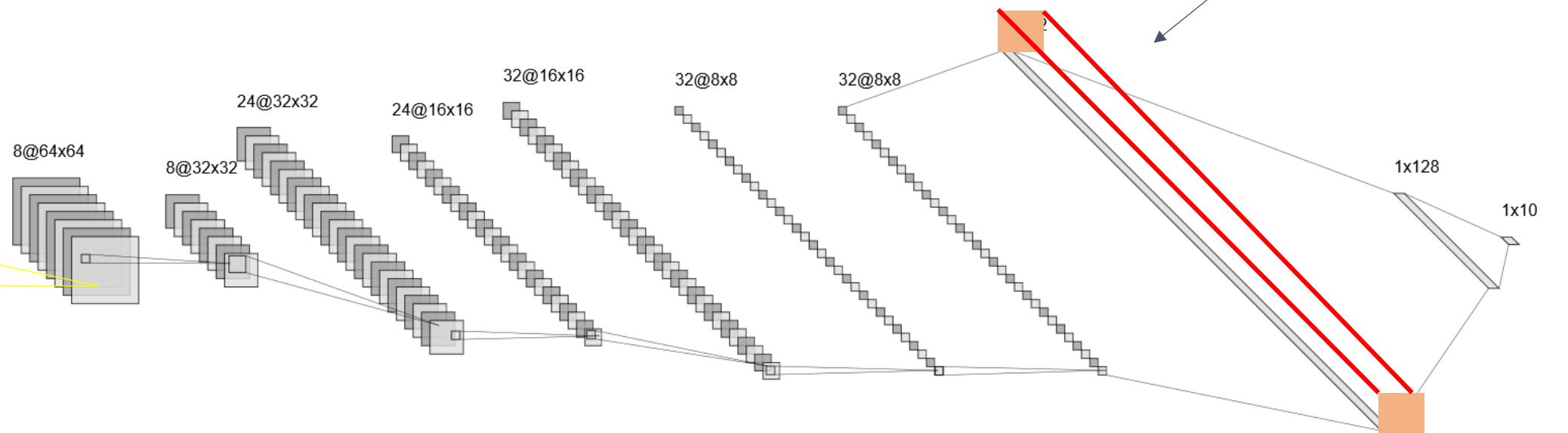
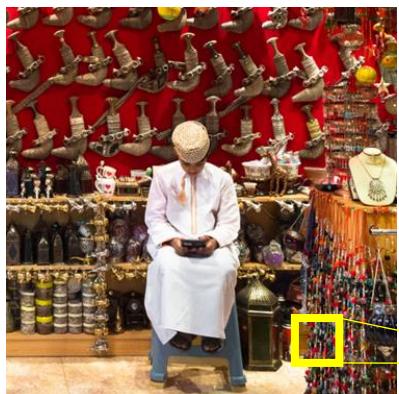
Convolutional filters can be applied to volumes of any size, yielding larger volumes in the network until the FC layer.

The FC network however does require a fixed input size

Thus, CNN cannot compute class scores, yet can extract features!



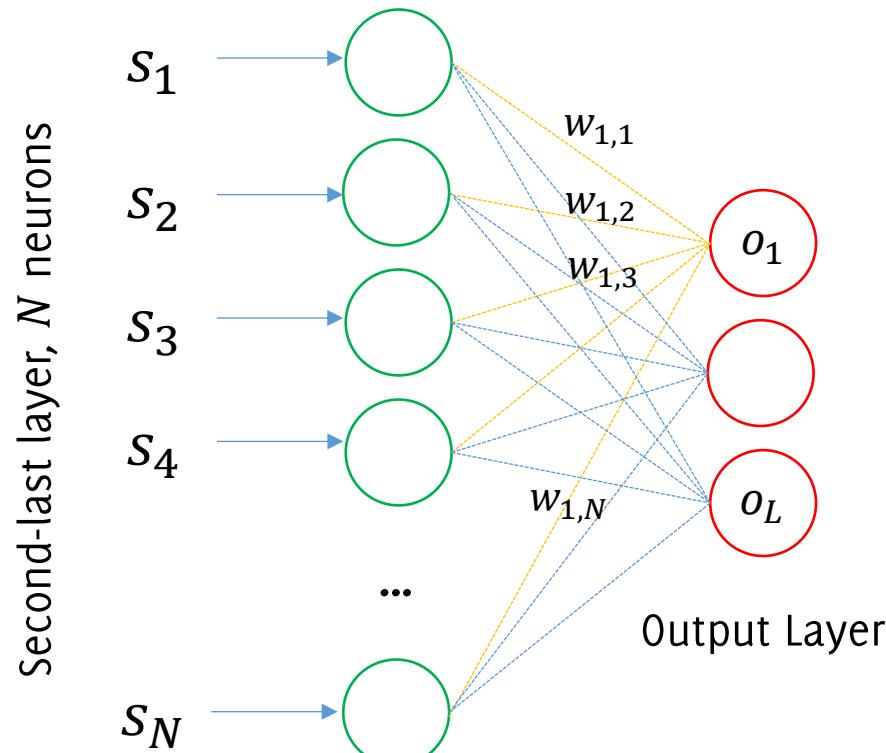
$$\text{size} = M_1 \times M_2 \times N$$



Fully Convolutional Neural Networks (FC-CNN)

However, since the FC is linear, it can be represented as convolution!

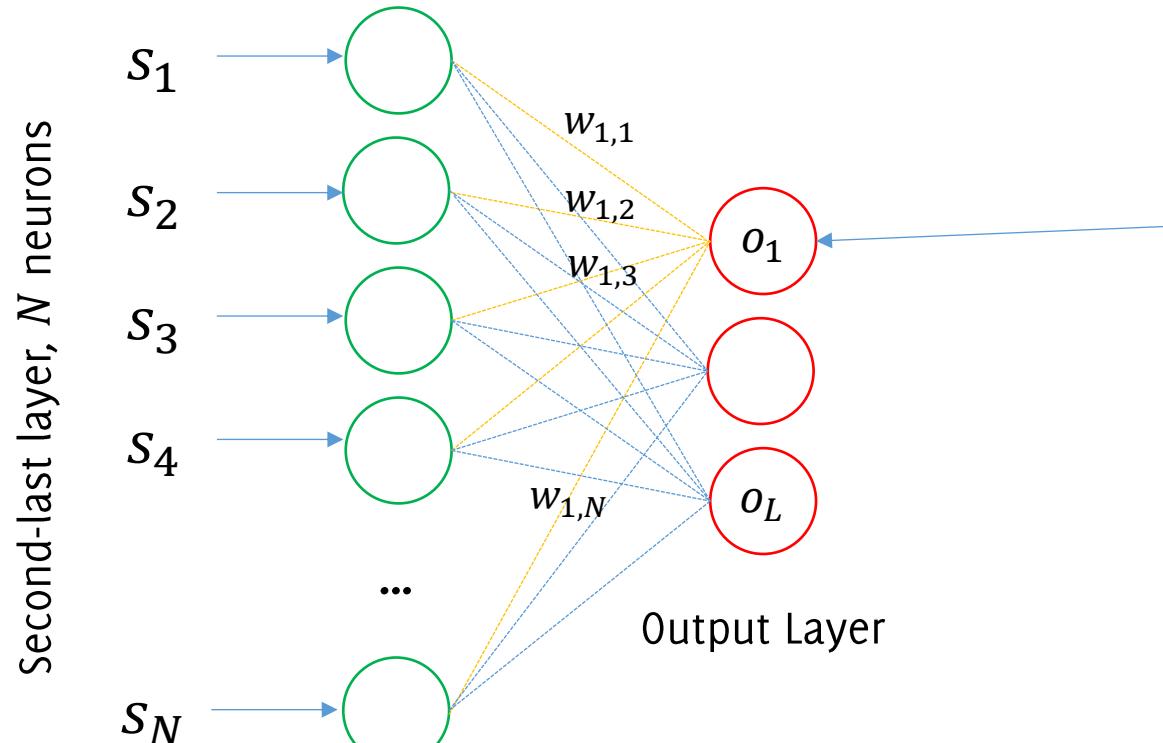
Weights associated to output neuron i : $\mathbf{w}_i = [w_{i,j}]_{j=1:N}$



Fully Convolutional Neural Networks (FC-CNN)

However, since the FC is linear, it can be represented as convolution!

Weights associated to output neuron i : $\mathbf{w}_i = [w_{i,j}]_{j=1:N}$.



$$o_i = \sum_{j=1:N} w_{i,j} s_j + b_i$$

$$o_i = (\mathbf{w}_i)' \mathbf{s} + b_i$$

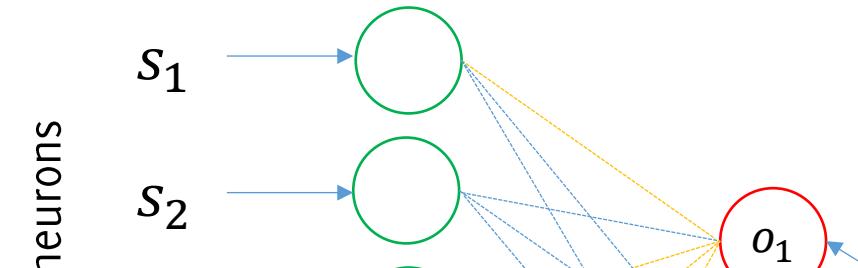
$$o_i = (\mathbf{w}_i \circledast \mathbf{s})(0,0) + b_i$$

A FC layer of L outputs NN corresponds to a 2DConv layer having L filters with size $1 \times 1 \times N$

Fully Convolutional Neural Networks (FC-CNN)

However, since the FC is linear, it can be represented as convolution!

Weights associated to output neuron i : $\mathbf{w}_i = [w_{i,j}]_{j=1:N}$



$$o_i = \sum w_{i,j} s_j + b_i$$

This transformation can be applied to each hidden layer of a FC network placed at the CNN top.

A FC layer
 L filters with size $1 \times 1 \times N$

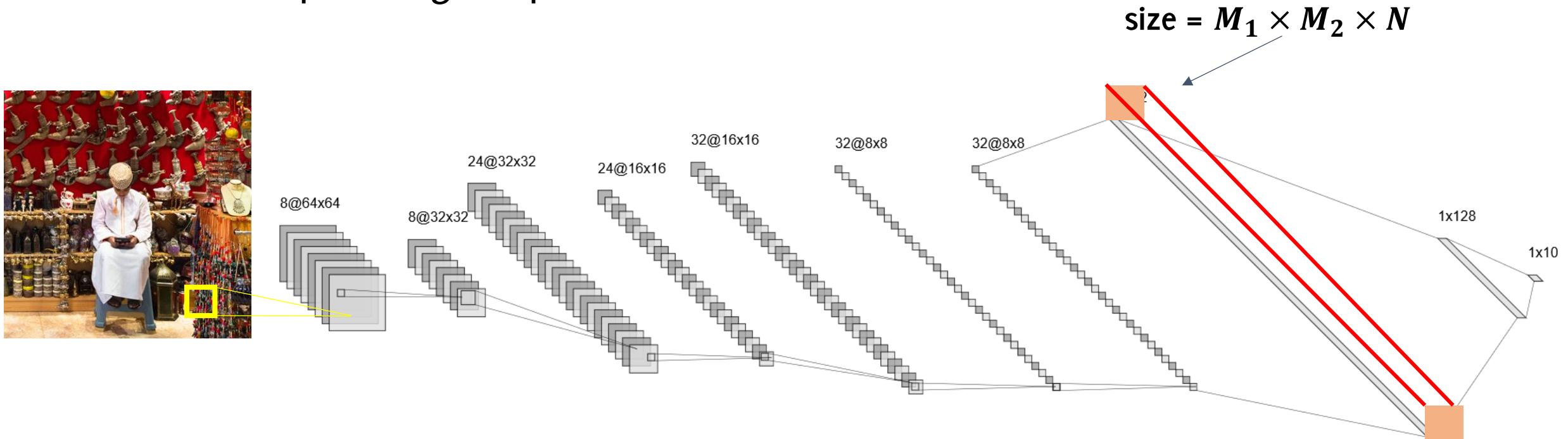
This transformation consists in reading weights from the neuron in the dense layer and recycling these together with bias in the filters of a new 1×1 convolutional layer.

w_i

Fully Convolutional Neural Networks (FC-CNN)

However, since the FC is linear, it can be represented as convolution against L filters of size $1 \times 1 \times N$.

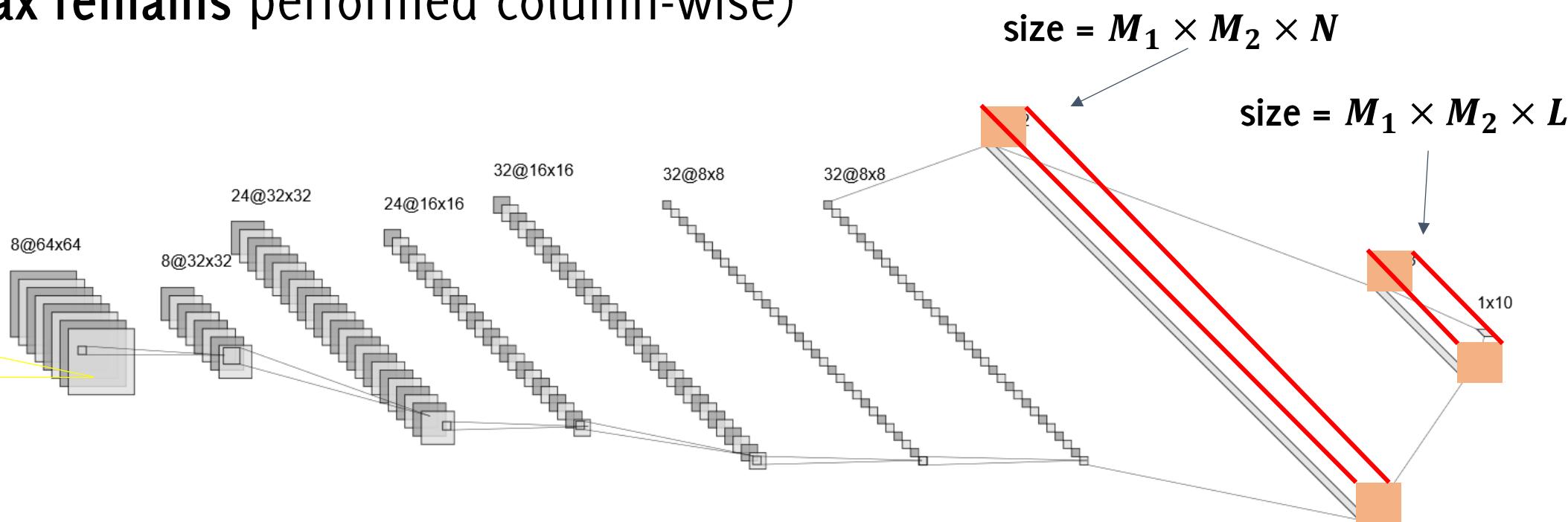
Each of these convolutional filters contains the weights of the FC for the corresponding output neuron.



Fully Convolutional Neural Networks (FC-CNN)

For each output class we obtain an image, having:

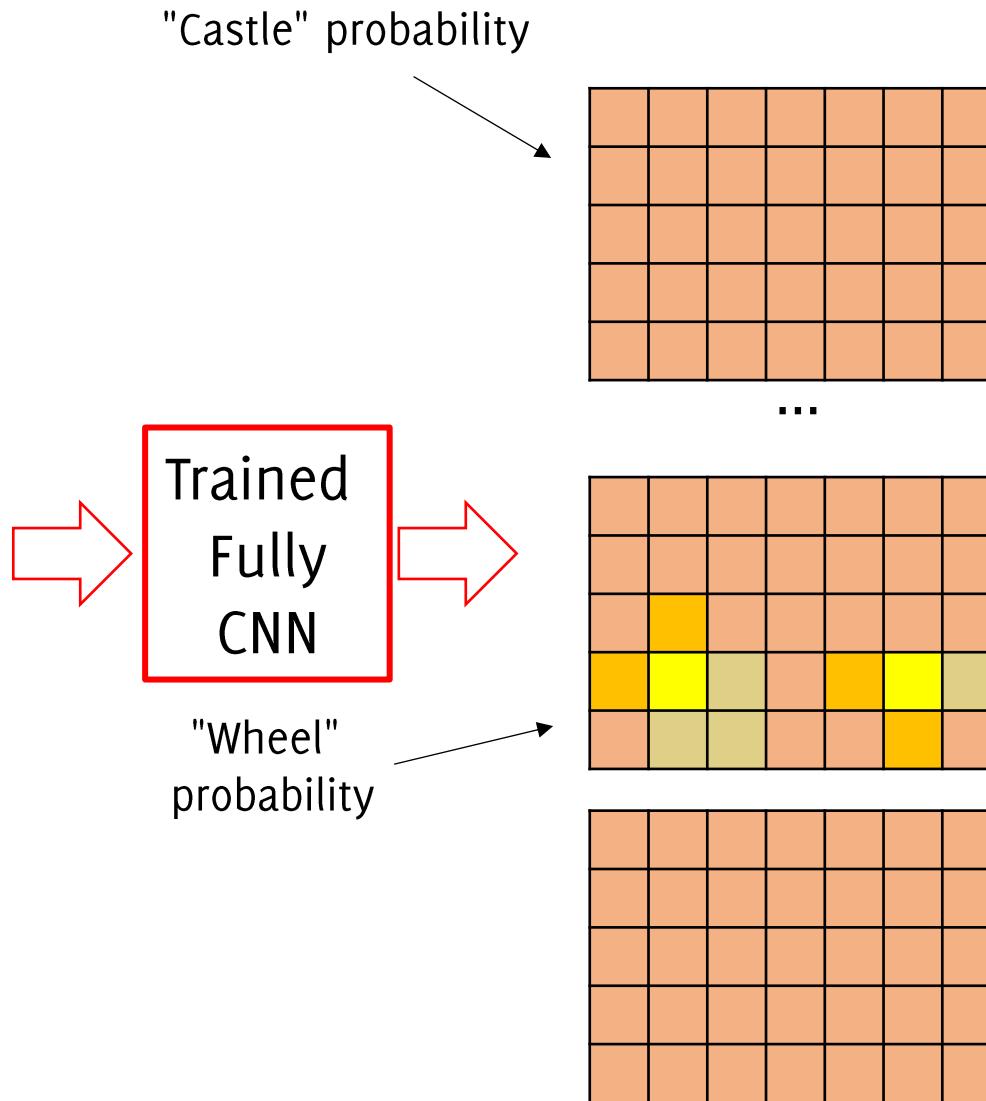
- Lower resolution than the input image
- Class probabilities for the receptive field of each pixel (assuming softmax remains performed column-wise)



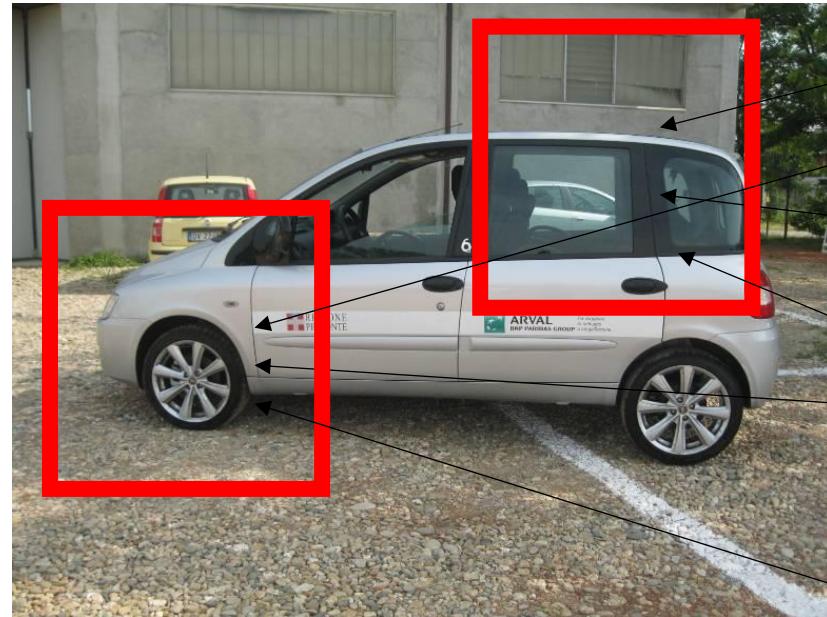
Output of a FC-CNN as heatmaps



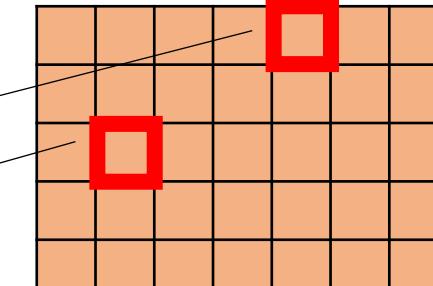
A larger image than
those used for training
the network



Output of a FCNN as heatmaps

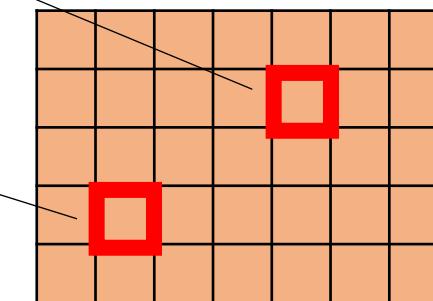
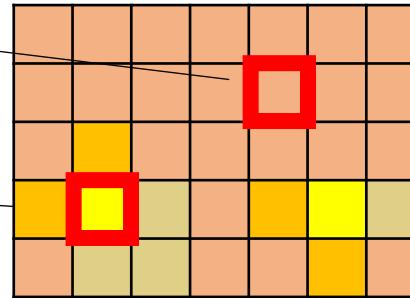


"Castle" probability



...

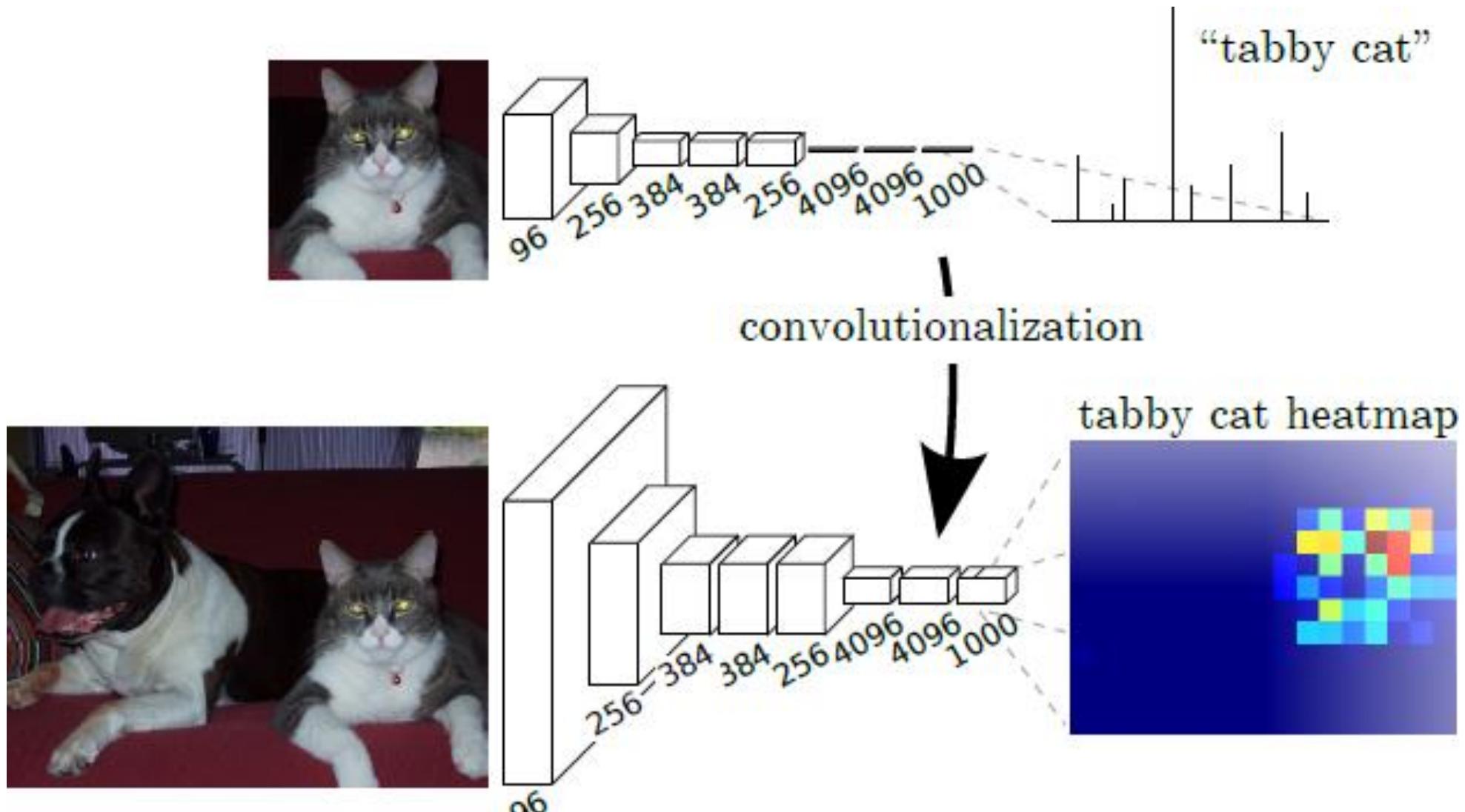
"Wheel" probability



Each pixel in the heatmap corresponds to a "receptive field" in the input image

Output heatmaps computed after softmax

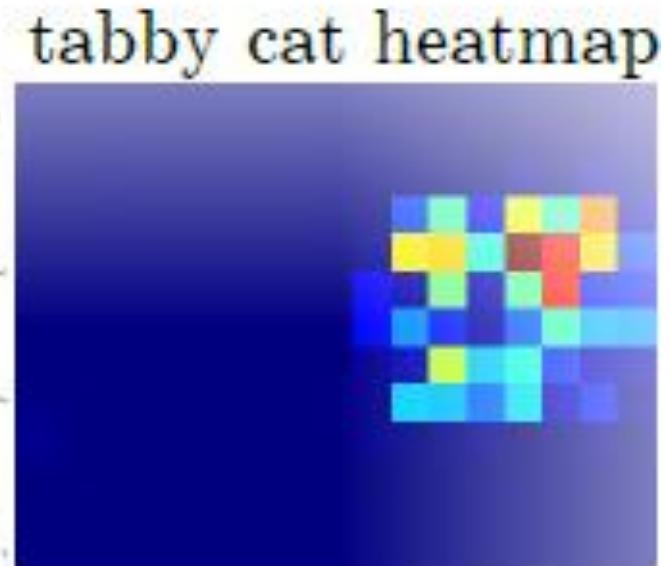
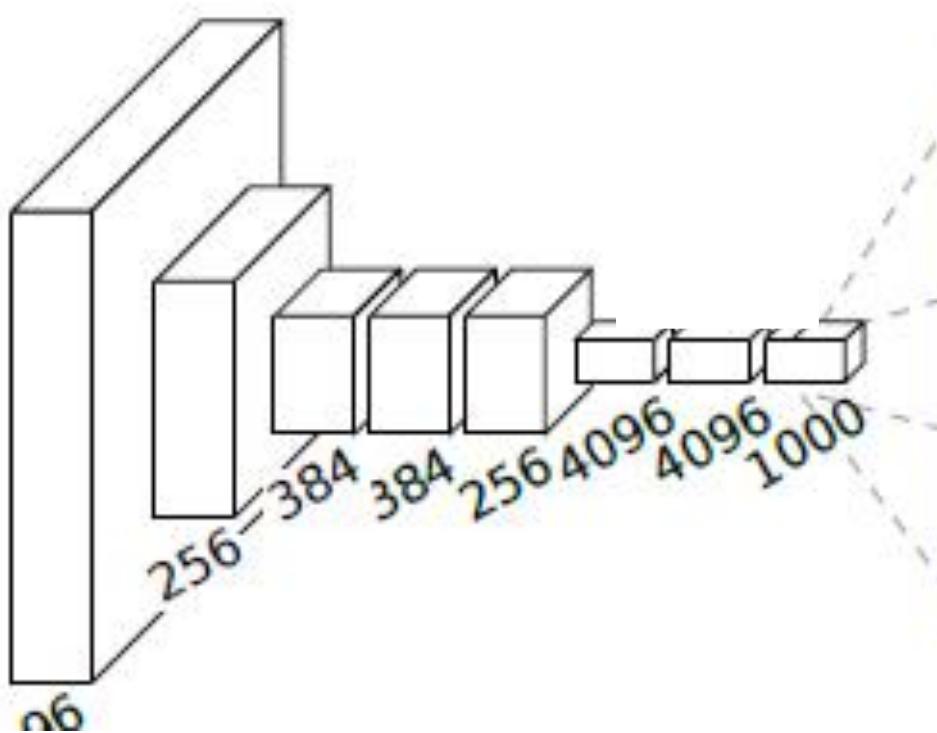
Migration to FCNN of a pretrained model



Migration to FCNN of a pretrained model

This stack of convolutions operates on the whole image as a powerful, nonlinear, filter which operates as a classifier sliding on the latent space.

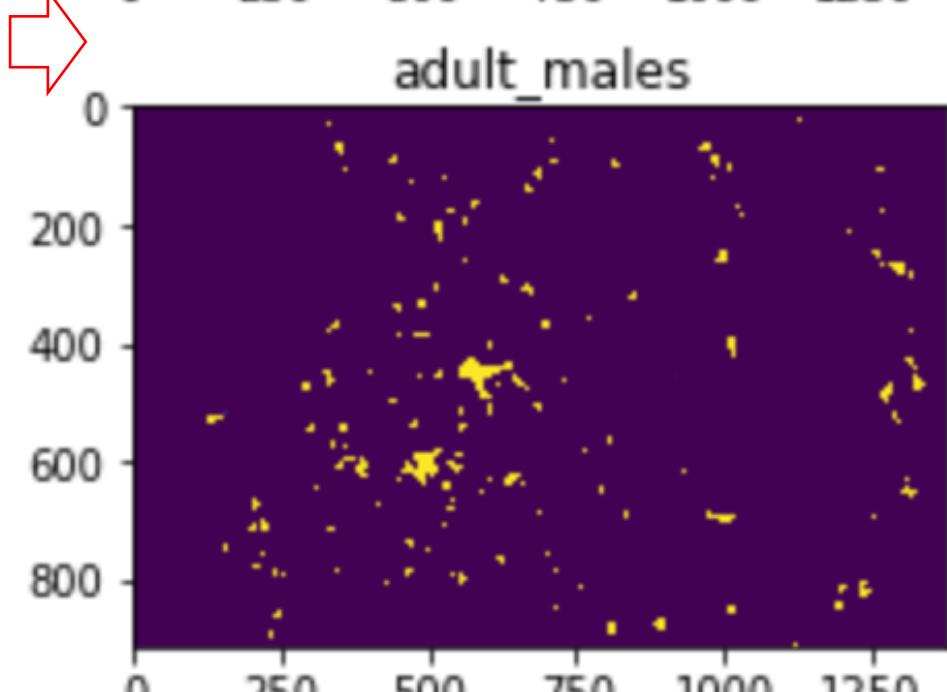
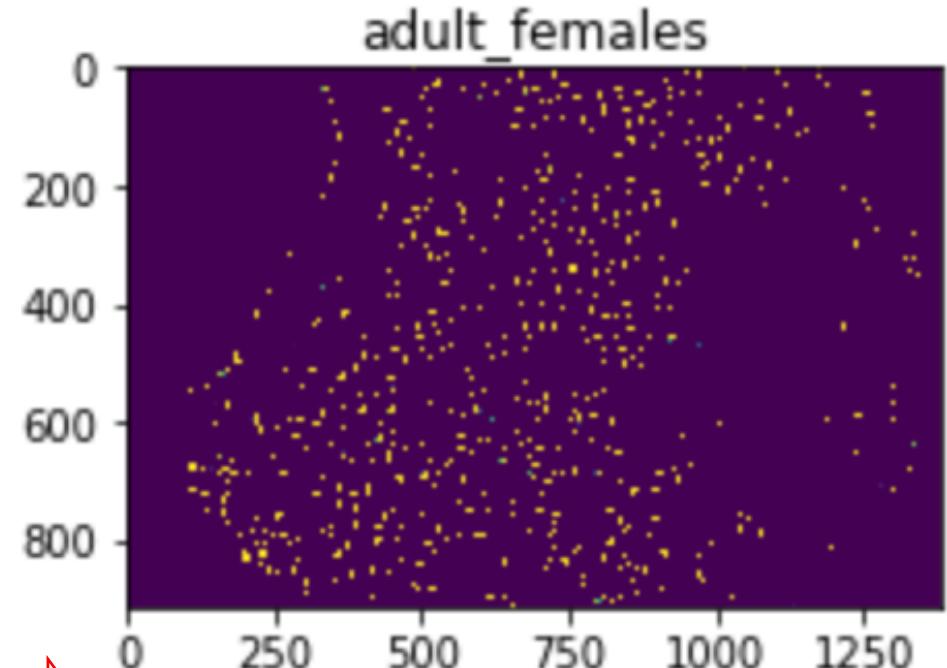
Significantly more efficient than patch extraction and classification: this avoids multiple repeated computations within overlapping patches



Sealion HeatMaps



FCNN



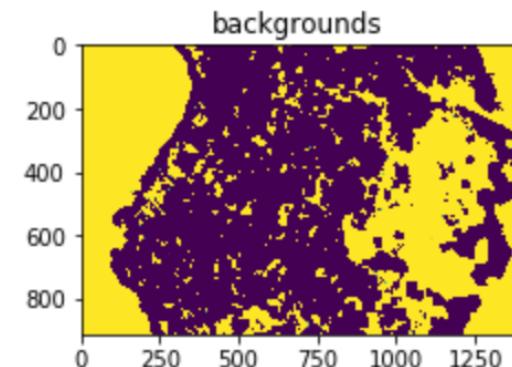
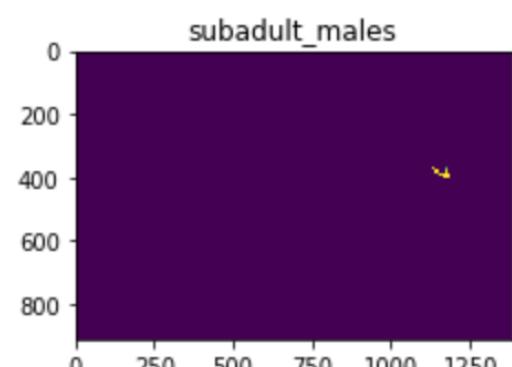
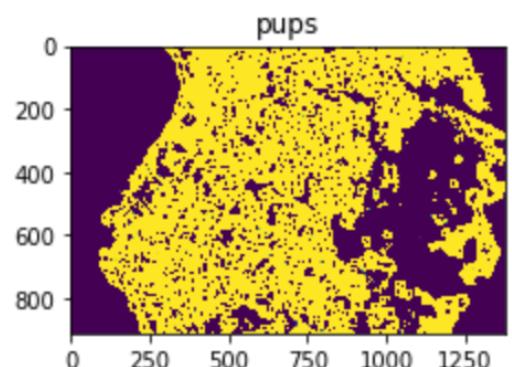
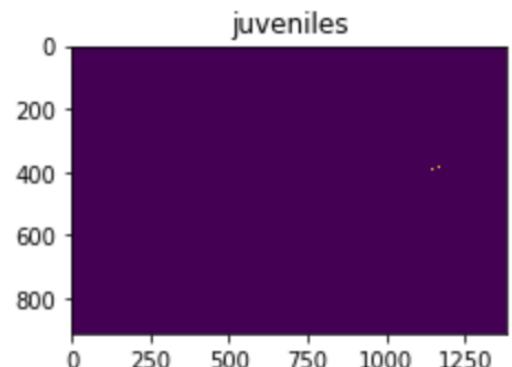
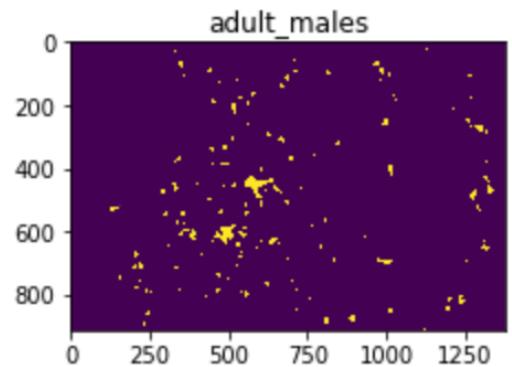
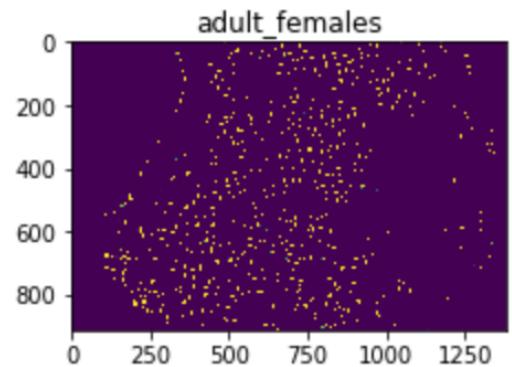
Credits Yinan Zhou

<https://github.com/marioZYN/FC-CNN-Demo>

Sealion HeatMaps



→ FCNN →

A sequence of three red arrows pointing from left to right. The first arrow points from the aerial photograph to a white rectangular box containing the text "FCNN". The second arrow points from the "FCNN" box to the heatmaps.

Credits Yinan Zhou

<https://github.com/marioZYN/FC-CNN-Demo>

Fully convolutional networks in keras

Assume **cnn** is a pre-trained CNN classifier and the model **cnn2** is the FC-CNN variant that is initialized as **cnn** in all the layers, until the first FC in **cnn**. Then, **cnn2** has suitably defined conv2D layers instead of FCs. Assume the FC in **cnn** has L output neurons.

Get and set the weights of networks by means of the methods **get_weights** and **set_weights**

- get the weights of the trained CNN (assume the FC is layer 7)
`w7, b7 = cnn.layers[7].get_weights()`
- reshape these weights to become 1×1 convolutional filters having depth equal to C (the input size of the dense layer) with L filters.
`w7.reshape(1, 1, C, L)`
- assign these weights to the FCNN architecture
`cnn2.layers[7].set_weights(w7, b7)`

Fully convolutional networks in kears

What happens when the classification network has a flattening layer?

Layer (type)	Output Shape	Param #
Input (InputLayer)	[(None, 96, 96, 3)]	0
preprocessing (Sequential)	(None, 96, 96, 3)	0
conv1 (Conv2D)	(None, 96, 96, 16)	448
mp1 (MaxPooling2D)	(None, 48, 48, 16)	0
conv2 (Conv2D)	(None, 48, 48, 32)	4640
mp2 (MaxPooling2D)	(None, 24, 24, 32)	0
conv3 (Conv2D)	(None, 24, 24, 64)	18496
mp3 (MaxPooling2D)	(None, 12, 12, 64)	0
conv4 (Conv2D)	(None, 12, 12, 128)	73856
flatten (Flatten)	(None, 18432)	0
dropout_12 (Dropout)	(None, 18432)	0
dense2 (Dense)	(None, 256)	4718848
Output (Dense)	(None, 6)	1542

Fully convolutional networks in kears

Include a **convolutional layer** having the **same spatial size** as the activation before flattening

```
fully_conv_1 = tfkl.Conv2D(  
    filters=256,  
    kernel_size=(12, 12),  
    padding='valid',  
    activation='relu',  
    name='fully_conv_1'  
) (conv4)
```

Layer (type)	Output Shape	Param #
Input (InputLayer)	[None, None, None, 3]	0
preprocessing (Sequential)	(None, None, None, 3)	0
conv1 (Conv2D)	(None, None, None, 16)	448
mp1 (MaxPooling2D)	(None, None, None, 16)	0
conv2 (Conv2D)	(None, None, None, 32)	4640
mp2 (MaxPooling2D)	(None, None, None, 32)	0
conv3 (Conv2D)	(None, None, None, 64)	18496
mp3 (MaxPooling2D)	(None, None, None, 64)	0
conv4 (Conv2D)	(None, None, None, 128)	73856
fully_conv_1 (Conv2D)	(None, None, None, 256)	4718848
fully_conv_2 (Conv2D)	(None, None, None, 6)	1542

Fully convolutional networks in kears

Include a **convolutional layer** having the **same spatial size** as the activation before flattening

```
fully_conv_1 = tfkl.Conv2D(  
    filters=256,  
    kernel_size=(12,12),  
    padding='valid',  
    activation='relu',  
    name='fully_conv_1'  
) (conv4)
```

Convolution options need ot be valid to give a single response

Layer (type)	Output Shape	Param #
Input (InputLayer)	[None, None, None, 3]	0
preprocessing (Sequential)	(None, None, None, 3)	0
conv1 (Conv2D)	(None, None, None, 16)	448
mp1 (MaxPooling2D)	(None, None, None, 16)	0
conv2 (Conv2D)	(None, None, None, 32)	4640
mp2 (MaxPooling2D)	(None, None, None, 32)	0
conv3 (Conv2D)	(None, None, None, 64)	18496
mp3 (MaxPooling2D)	(None, None, None, 64)	0
conv4 (Conv2D)	(None, None, None, 128)	73856
fully_conv_1 (Conv2D)	(None, None, None, 256)	4718848
fully_conv_2 (Conv2D)	(None, None, None, 6)	1542

Fully convolutional networks in kears

Include a **convolutional layer** having the **same spatial size** as the activation before flattening

```
fully_conv_1 = tfkl.Conv2D(  
    filters=256,  
    kernel_size=(12,12),  
    padding='valid',  
    activation='relu',  
    name='fully_conv_1'  
) (conv4)
```

Convolution options need ot be valid to give a single response

The number of filters corresponds to the number of output neurons of the filter

Layer (type)	Output Shape	Param #
Input (InputLayer)	[None, None, None, 3]	0
preprocessing (Sequential)	(None, None, None, 3)	0
conv1 (Conv2D)	(None, None, None, 16)	448
mp1 (MaxPooling2D)	(None, None, None, 16)	0
conv2 (Conv2D)	(None, None, None, 32)	4640
mp2 (MaxPooling2D)	(None, None, None, 32)	0
conv3 (Conv2D)	(None, None, None, 64)	18496
mp3 (MaxPooling2D)	(None, None, None, 64)	0
conv4 (Conv2D)	(None, None, None, 128)	73856
fully_conv_1 (Conv2D)	(None, None, None, 256)	4718848
fully_conv_2 (Conv2D)	(None, None, None, 6)	1542

Fully convolutional networks in kears

Make sure each convolutional layer have the same number of parameters as the dense layer it is replacing

Layer (type)	Output Shape	Param #
Input (InputLayer)	[None, 96, 96, 3]	0
preprocessing (Sequential)	(None, 96, 96, 3)	0
conv1 (Conv2D)	(None, 96, 96, 16)	448
mp1 (MaxPooling2D)	(None, 48, 48, 16)	0
conv2 (Conv2D)	(None, 48, 48, 32)	4640
mp2 (MaxPooling2D)	(None, 24, 24, 32)	0
conv3 (Conv2D)	(None, 24, 24, 64)	18496
mp3 (MaxPooling2D)	(None, 12, 12, 64)	0
conv4 (Conv2D)	(None, 12, 12, 128)	73856
flatten (Flatten)	(None, 18432)	0
dropout_12 (Dropout)	(None, 18432)	0
dense2 (Dense)	(None, 256)	4718848
Output (Dense)	(None, 6)	1542

Layer (type)	Output Shape	Param #
Input (InputLayer)	[None, None, None, 3]	0
preprocessing (Sequential)	(None, None, None, 3)	0
conv1 (Conv2D)	(None, None, None, 16)	448
mp1 (MaxPooling2D)	(None, None, None, 16)	0
conv2 (Conv2D)	(None, None, None, 32)	4640
mp2 (MaxPooling2D)	(None, None, None, 32)	0
conv3 (Conv2D)	(None, None, None, 64)	18496
mp3 (MaxPooling2D)	(None, None, None, 64)	0
conv4 (Conv2D)	(None, None, None, 128)	73856
fully_conv_1 (Conv2D)	(None, None, None, 256)	4718848
fully_conv_2 (Conv2D)	(None, None, None, 6)	1542

Fully convolutional networks in kears

Build the network to take as input any spatial size

```
fc_input_shape = (None, None, 3)
fc_output_shape = (None, None, 6)
input_layer = tfkl.Input(shape=input_shape, name='Input')
...
output_layer = tfkl.Conv2D(
    filters=output_shape[-1],
    kernel_size=(1, 1),
    padding='valid',
    activation='softmax',
    name='fully_conv_2'
) (fully_conv_1)
```

Layer (type)	Output Shape	Param #
Input (InputLayer)	[None, None, None, 3]	0
preprocessing (Sequential)	(None, None, None, 3)	0
conv1 (Conv2D)	(None, None, None, 16)	448
mp1 (MaxPooling2D)	(None, None, None, 16)	0
conv2 (Conv2D)	(None, None, None, 32)	4640
mp2 (MaxPooling2D)	(None, None, None, 32)	0
conv3 (Conv2D)	(None, None, None, 64)	18496
mp3 (MaxPooling2D)	(None, None, None, 64)	0
conv4 (Conv2D)	(None, None, None, 128)	73856
fully_conv_1 (Conv2D)	(None, None, None, 256)	4718848
fully_conv_2 (Conv2D)	(None, None, None, 6)	1542

Check The Fully Convolutional Lab on sealions

https://drive.google.com/drive/folders/1-Uv2XpQcwhuEpXromxlqyfyUsMcEaBTh?usp=drive_link



This CVPR2015 paper is the Open Access version, provided by the Computer Vision Foundation.
The authoritative version of this paper is available in IEEE Xplore.

Fully Convolutional Networks for Semantic Segmentation

Jonathan Long*

Evan Shelhamer*

Trevor Darrell

UC Berkeley

{jonlong, shelhamer, trevor}@cs.berkeley.edu

From Classification to Segmentation

Setup:

- You are given a pre-trained CNN for classification
- You possibly performed transfer learning to the problem at hand
- You have «convolutionalized» it to extract heatmaps

Limitation:

- Heatmaps are very low-resolution

Goal:

- Obtain a Semantic Segmentation network for images of arbitrary size

Simple Solution (1): Direct Heatmap Predictions

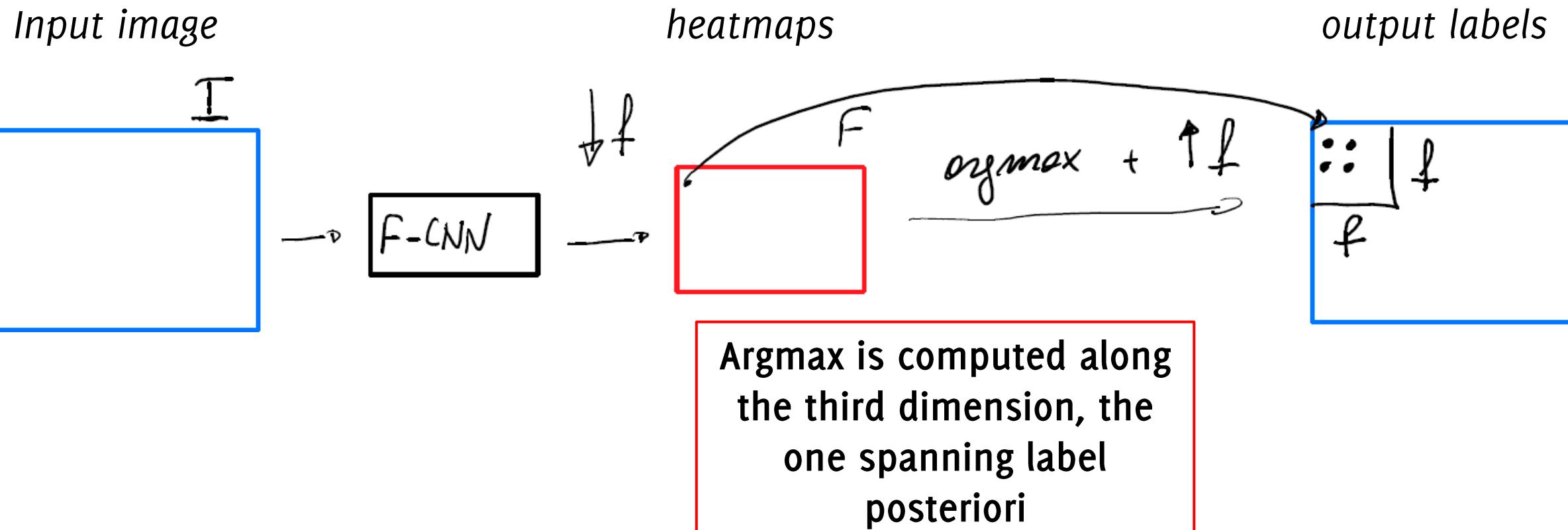
We can assign the predicted label in the heatmap to the whole receptive field, however that would be a **very coarse estimate**



We need to upsample the estimated map!

Simple Solution (1): Direct Heatmap Upsampling

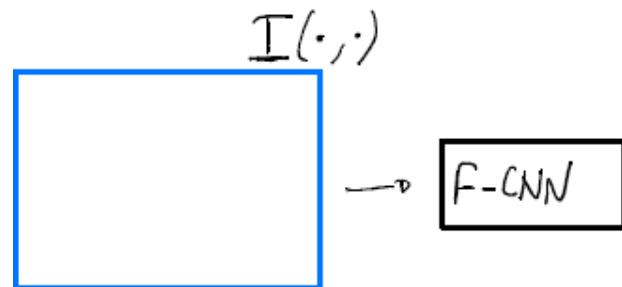
Very coarse estimates



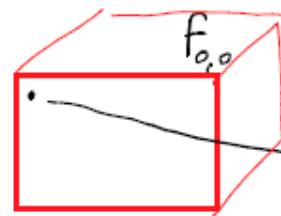
Simple Solution (2): The Shift and Stich

Shift and Stitch: Assume there is a downsampling ratio f between the size of input and of the output heatmap

Input image

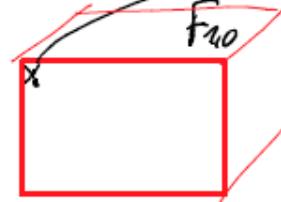
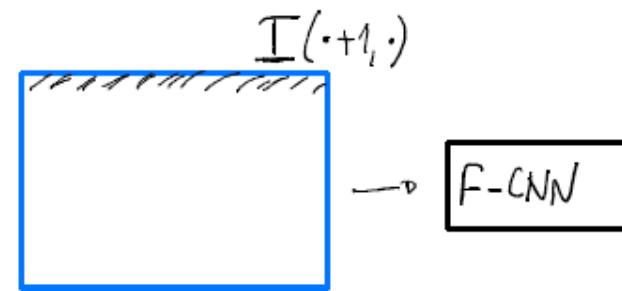


heatmaps

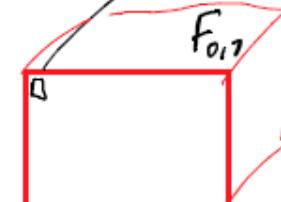
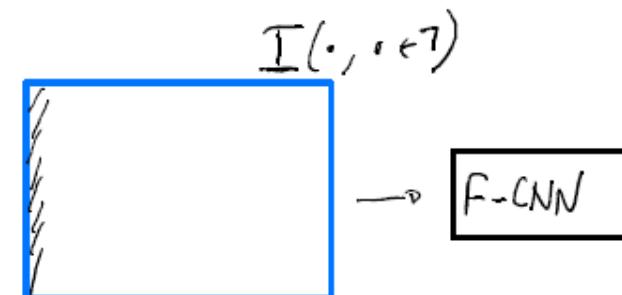


output labels

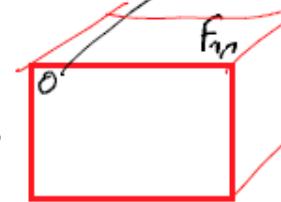
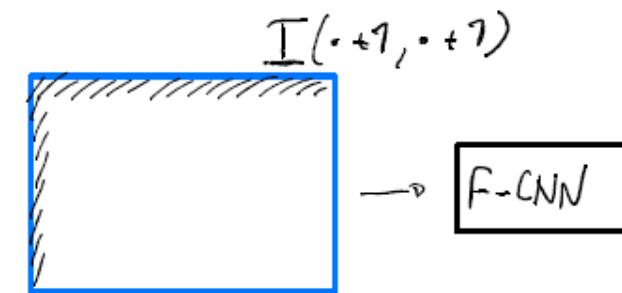
$$\text{softmax}(F_{0,0}, 3)$$



$$\text{softmax}(f_{1,0}, 3)$$



$$\text{softmax}(F_{0,e7}, 3)$$



That's a very rigid form of upsampling!

Simple Solution (2): The Shift and Stich

Shift and Stitch: Assume there is a downsampling ratio f between the size of input and of the output heatmap

- Compute **heatmaps** for all f^2 possible shifts of the input ($0 \leq r, c < f$)
- **Map predictions from the f^2 heatmaps to the image:** each pixel in the heatmap provides prediction of the central pixel of the receptive field
- **Interleave the heatmaps** to form an image as large as the input

Shift and Stitch

- **exploits the whole depth of the network**
- **efficient implementation** via the à trous algorithm as in Wavelets (dilating filters rather than repeating operations)

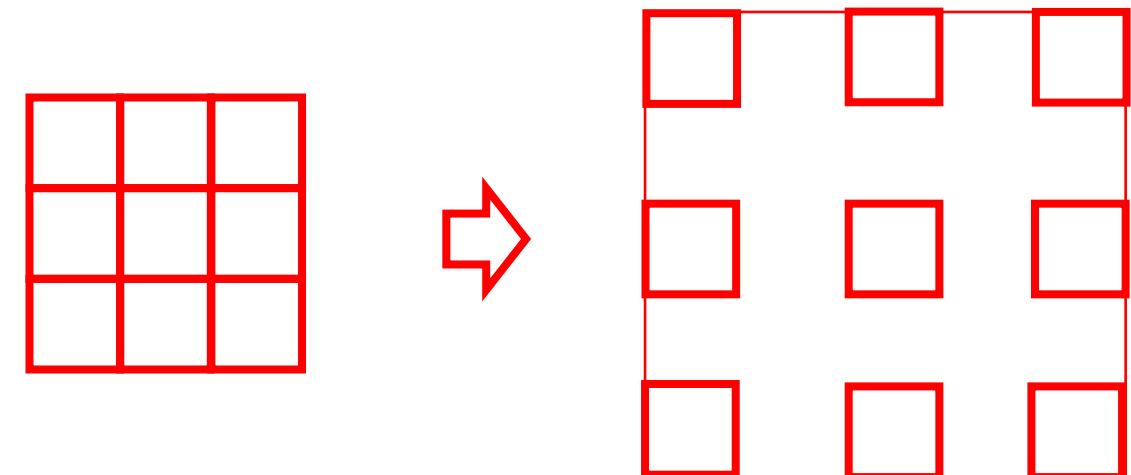
However, **the upsampling method is very rigid**

Simple Solution (2): The Shift and Stich

The same effect as shift and stich can be obtained by

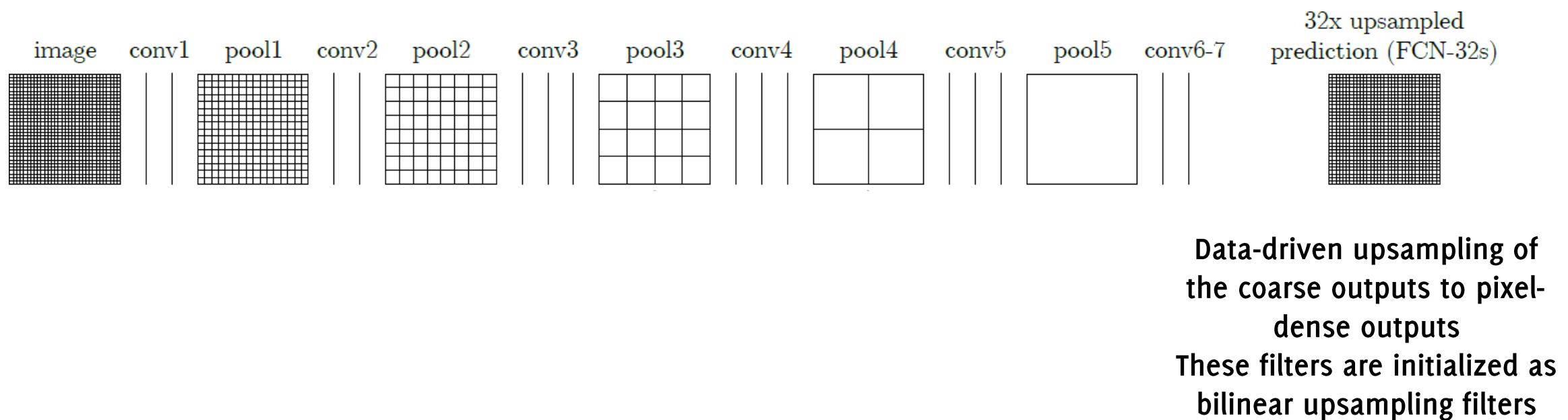
- Remove strides in pooling layer (conv or mp). This is equivalent as computing the output of all the shifted versions at once
- *Rarefy* the filters of the following convolution layer by upsampling and zero padding

$$f'_{ij} = \begin{cases} f_{i/s, j/s} & \text{if } s \text{ divides both } i \text{ and } j; \\ 0 & \text{otherwise,} \end{cases}$$



- Repeat the same operation along each channel dimension
- Repeat for each subsampling layer

Solution (3): Learn Upsampling in a FC-CNN



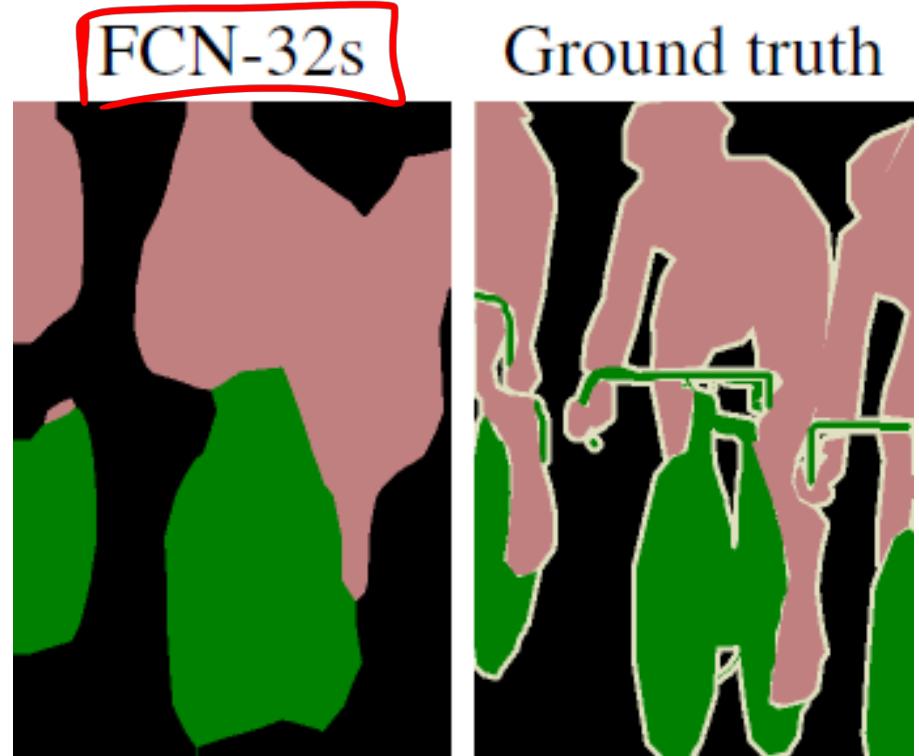
Prediction Upsampling

Linear upsampling of a factor f can be implemented as a convolution against a filter with a fractional stride $1/f$.

Upsampling filters can thus be learned during network training (provided a segmentation training set).

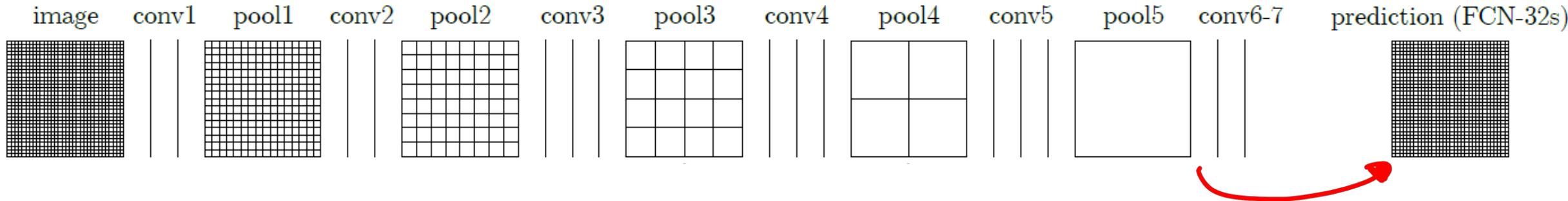
These predictions however are **very coarse**

Upsampling filters are learned with initialization equal to the bilinear interpolation

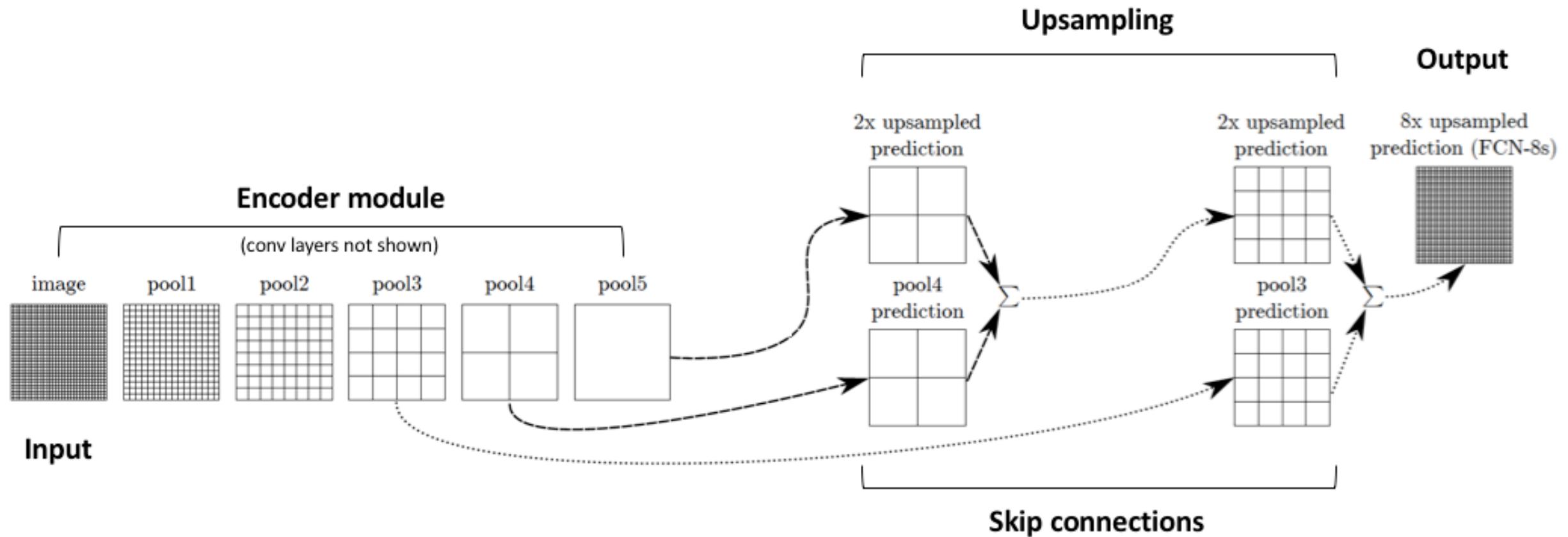


Upsampling filters

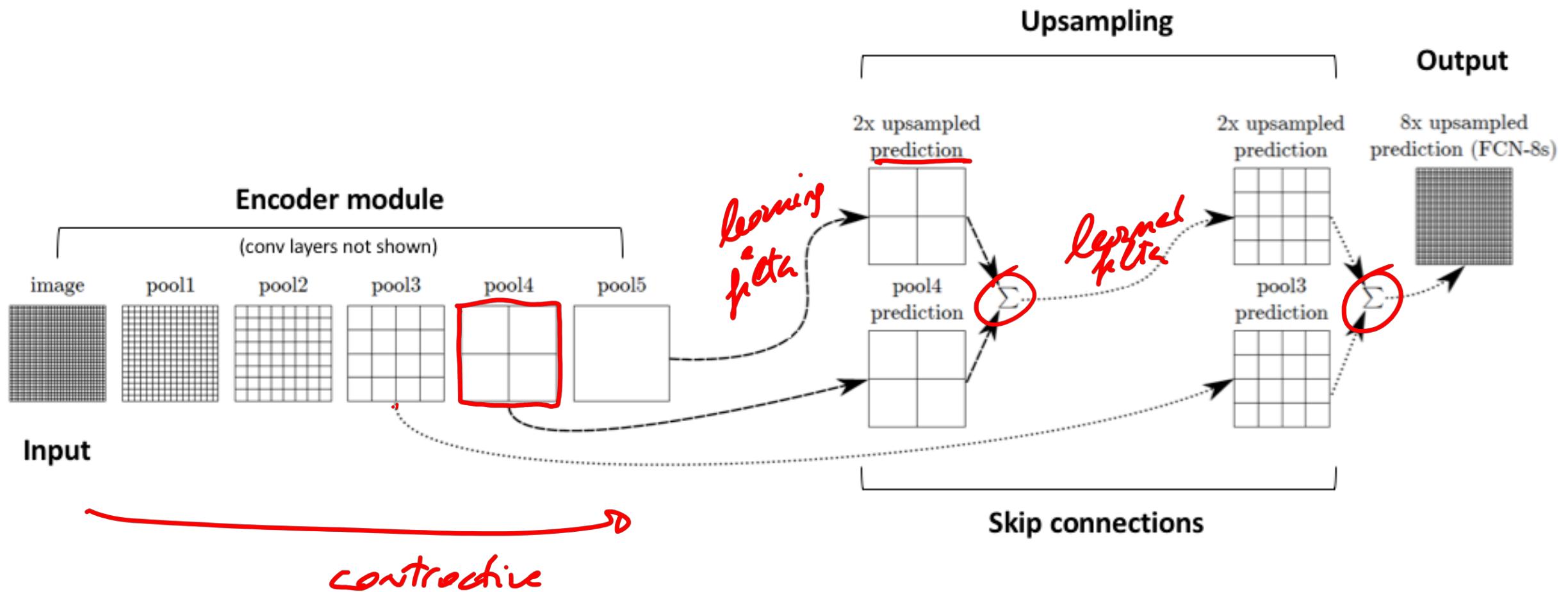
32x upsampled
prediction (FCN-32s)



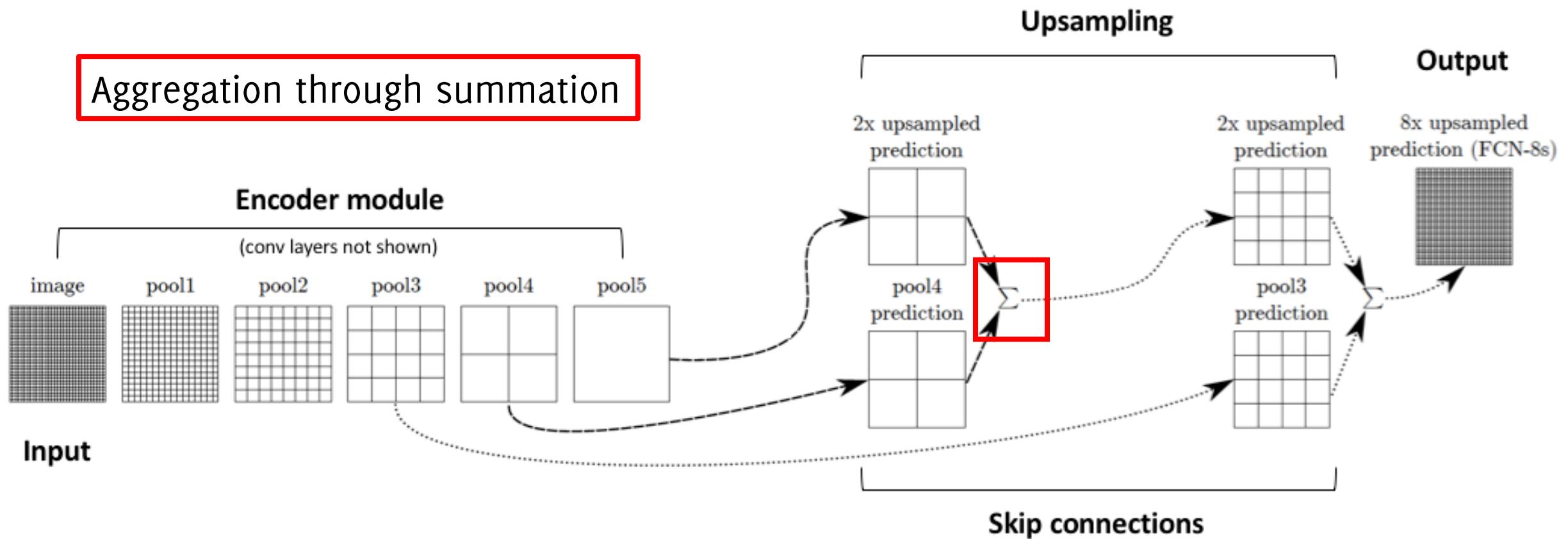
Improve segmentation: Skip Connections!



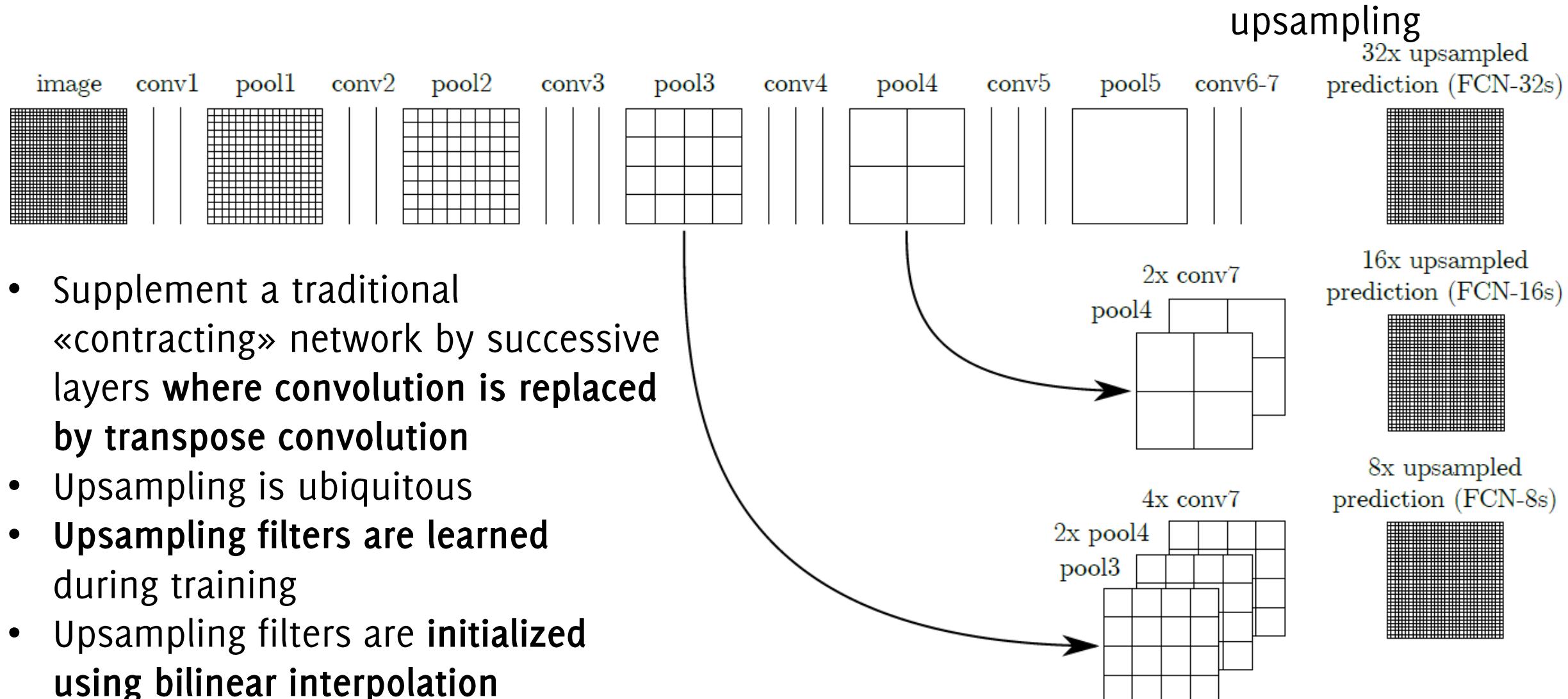
Improve segmentation: Skip Connections!



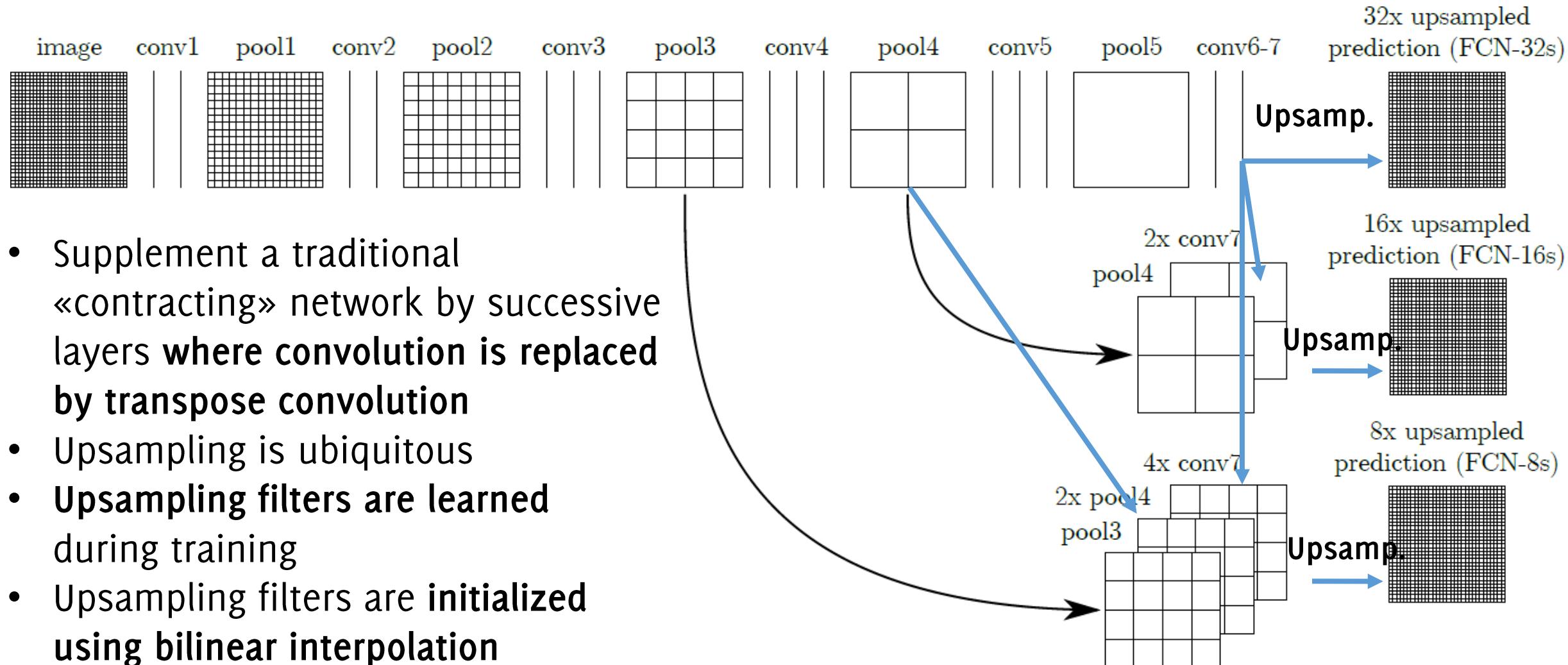
Solution: Skip Connections!



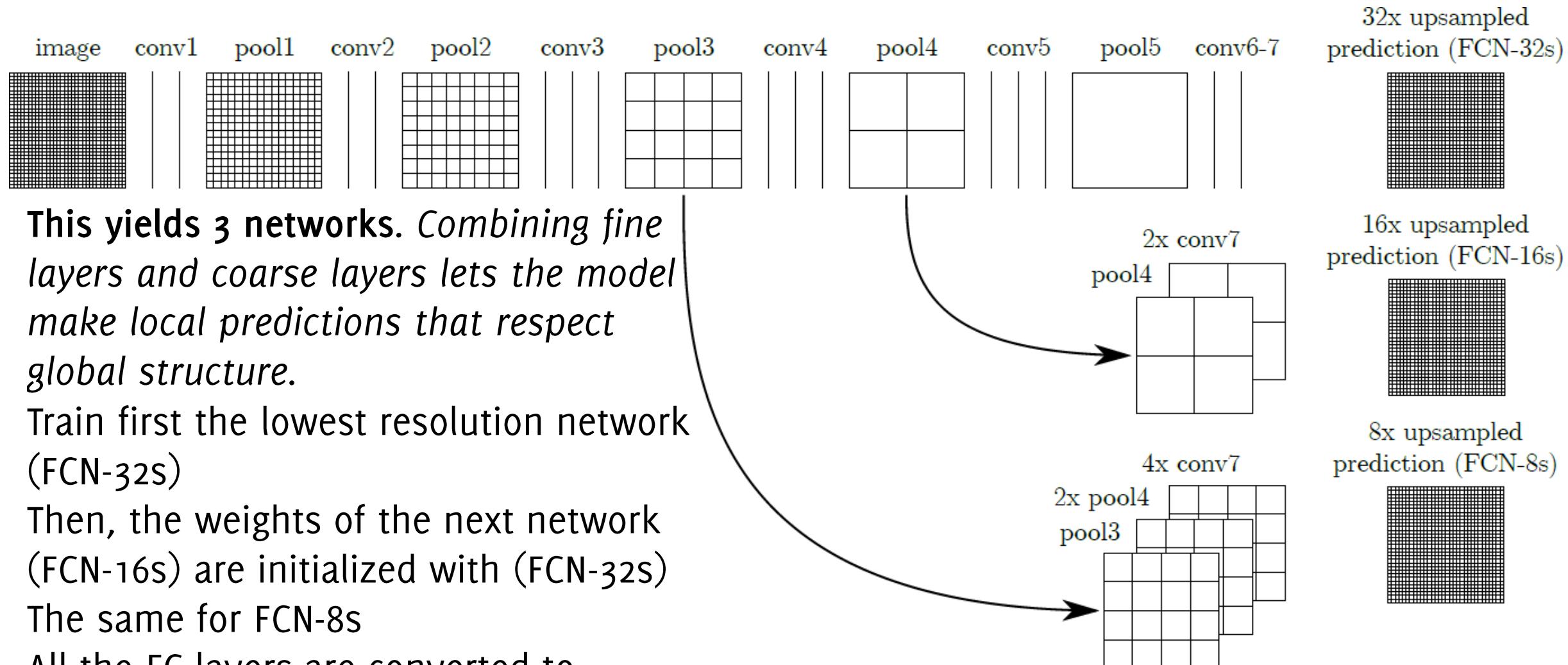
Solution: Skip Connections!



Solution: Skip Connections!

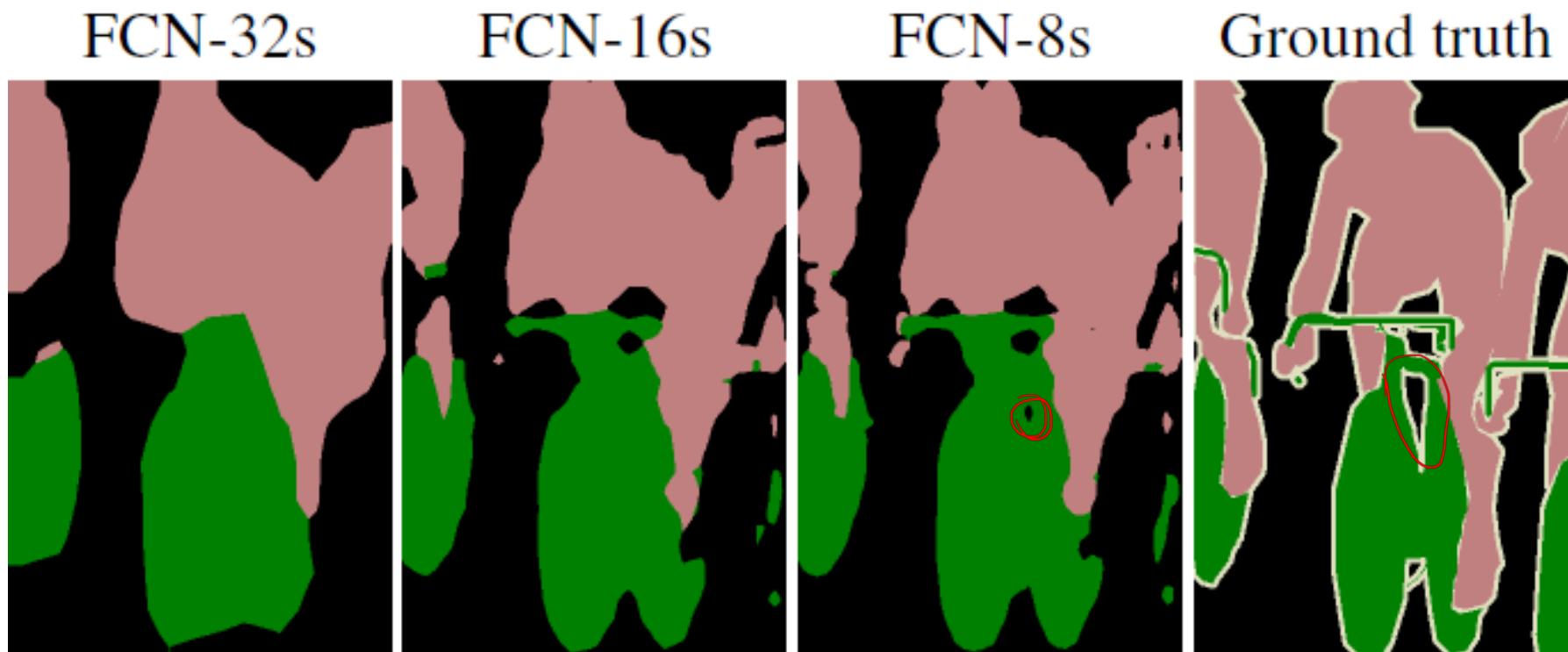


Solution: Skip Connections!

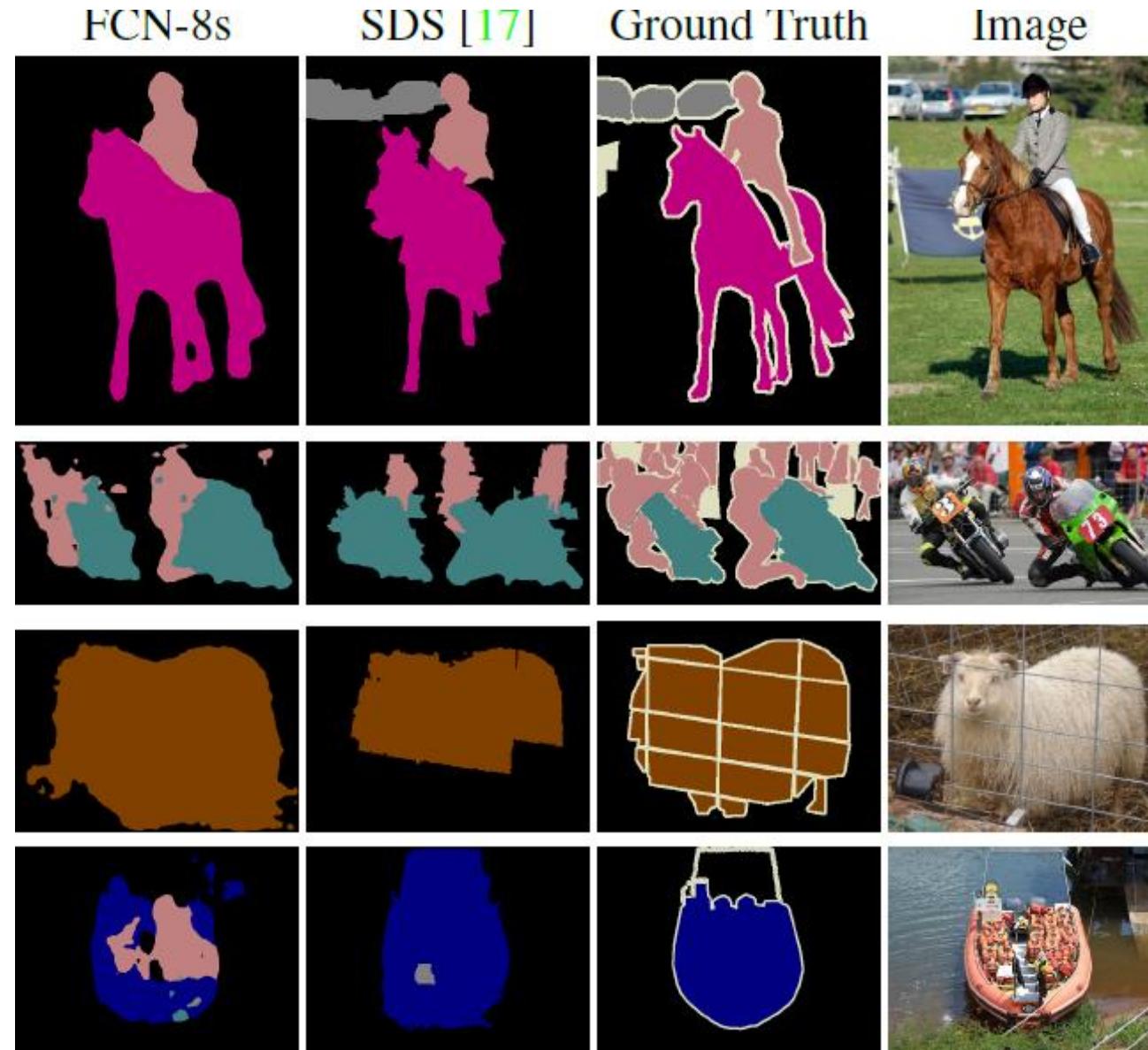


Semantic Segmentation

Retaining intermediate information is beneficial, the deeper layers contribute to provide a better refined estimate of segments



Semantic Segmentation Results



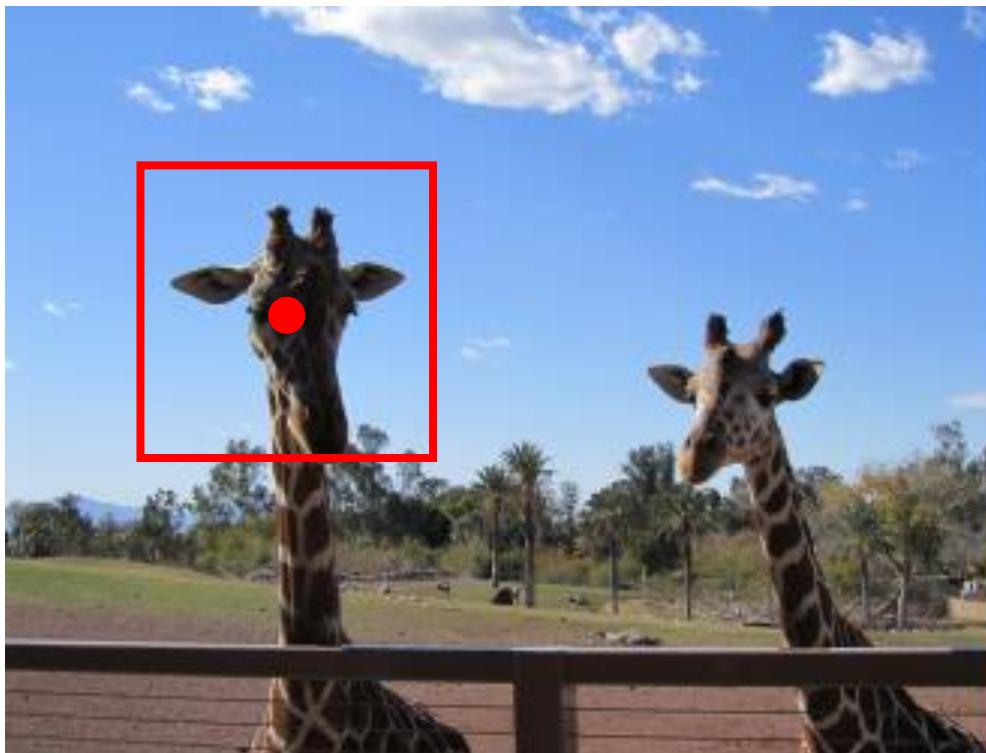
Comments

- Both learning and inference can be performed on the whole-image-at-a-time
- It is possible to **perform transfer learning/fine tuning of pre-trained classification models** (segmentation typically requires fewer labels than classification)
- **Accurate pixel-wise prediction** is achieved by upsampling layers
- Outperforms state-of the art in 2015
- Being fully convolutional, this network **handles arbitrarily sized input**

Patch-wise Training vs Full-image Training

«Patch-based» training

- Prepare a training set for a classification network
- Crop as many patches x_i from annotated images and assign to each patch, the label corresponding to the patch center



FC-CNN Training Options

The «patch-based» way:

- Prepare a training set for a classification network:
 - crop many patches x_i from annotated images
 - assign to each patch, the label corresponding to the patch center
- Train a CNN for classification from scratches, or fine tune a pre-trained model over the segmentation classes.
- Convolutionalization: once trained the network, move the FC layers to 1x1 convolutions.
- (Design and train the upsampling side of the network)

FC-CNN Training Options

The «patch-based» way:

- The classification network is trained to minimize the classification loss ℓ over a mini-batch B

$$\hat{\theta} = \operatorname{argmin}_{\theta} \sum_{x_j \in B} \ell(x_j, \theta)$$

where x_j belongs to a mini-batch B

- Batches of patches are **randomly assembled during training**
- It is possible to **resample patches for solving class imbalance**
- It is **very inefficient**, since convolutions on overlapping patches are repeated multiple times

FC-CNN Training Options: The «full-image» way

When full-image annotations are provided, x_j are all the pixels in a region R of the input image and the loss is evaluated over the corresponding labels in the annotation for semantic segmentation.

$$\hat{\theta} = \operatorname{argmin}_{x_j \in I} \sum_{x_j \in R} \ell(x_j, \theta)$$

Therefore, each region provides already a mini-batch estimate for computing gradient.

Whole image fully convolutional training is identical to patchwise training where each batch consists of all the receptive fields of the units below the loss for an image [...]. While this is more efficient than uniform sampling of patches, it reduces the number of possible batches.

FCNN Training Options

The «full-image» way:

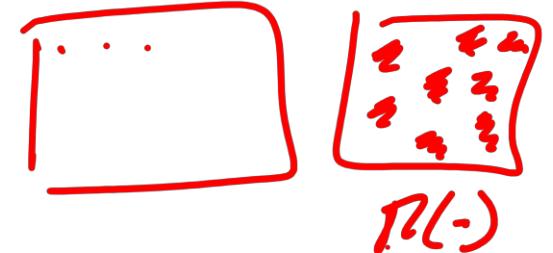
- **FCNN are trained in an end-to-end manner to predict the segmented output $S(\cdot, \cdot)$.**
- This loss is the sum of losses over different pixels. Derivatives can be easily computed through the whole network, and this can be trained through backpropagation.
- No need to pass through a classification network first.
- Takes **advantage of FC-CNN efficiency**, does not have to re-compute convolutional features in overlapping regions.
- **End-to-end training is more efficient than patch-wise training**

Limitations of full-image training and solutions

Limitations of full-image training and solutions:

- **Minibatches in patch-wise training are assembled randomly.** Image regions in full-image training are not. To make the estimated loss a bit stochastic, adopt random mask

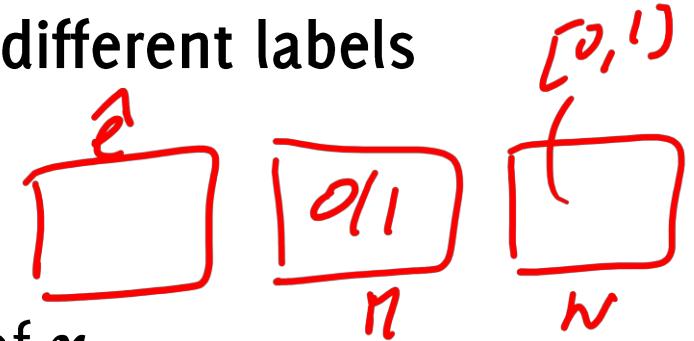
$$\underset{x_j}{\text{minimize}} \sum M(x_j) \ell(x_j, \theta)$$



being $M(x_j)$ a binary random variable

- It is not possible to perform patch resampling to compensate for class imbalance. One should go for **weighting the loss over different labels**

$$\underset{x_j}{\text{minimize}} \sum w(x_j) \ell(x_j, \theta)$$



being $w(x_j)$ a weight that depends on the true label of x_j

Example of Research Projects on Semantic Segmentation

Credits Sebastiano Rossi

Looking for Champions to address next challenges...

In collaboration with

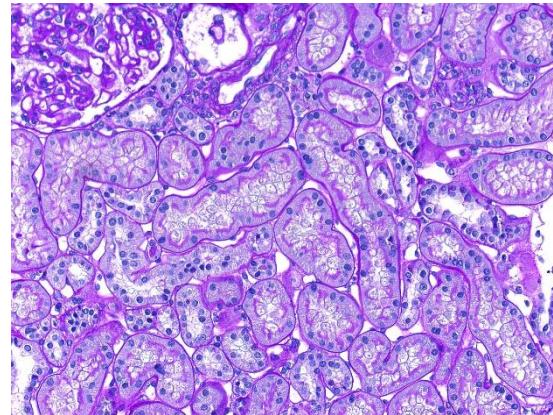


Intrinsic Challenges

SCARSE ANNOTATIONS

Few images available with the correspondent **dense annotations**.
In our application the average annotation time is 2 hours for each image.

Image



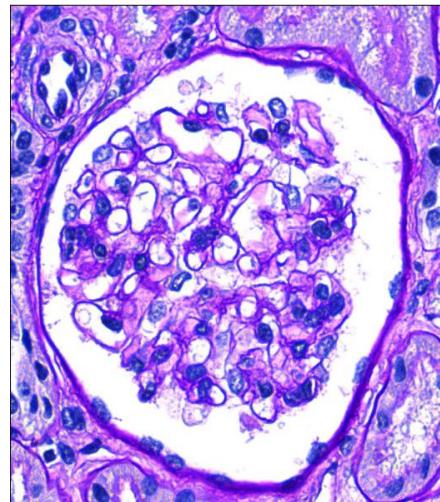
Dense Annotations



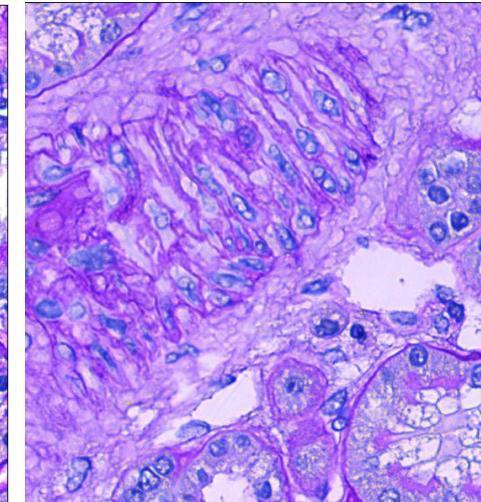
HIGH ANATOMICAL VARIABILITY

Different patients suffering from different pathologies, great anatomical variability even within the same group.

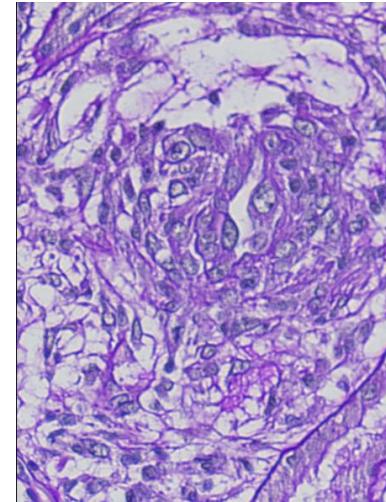
Healthy Subject



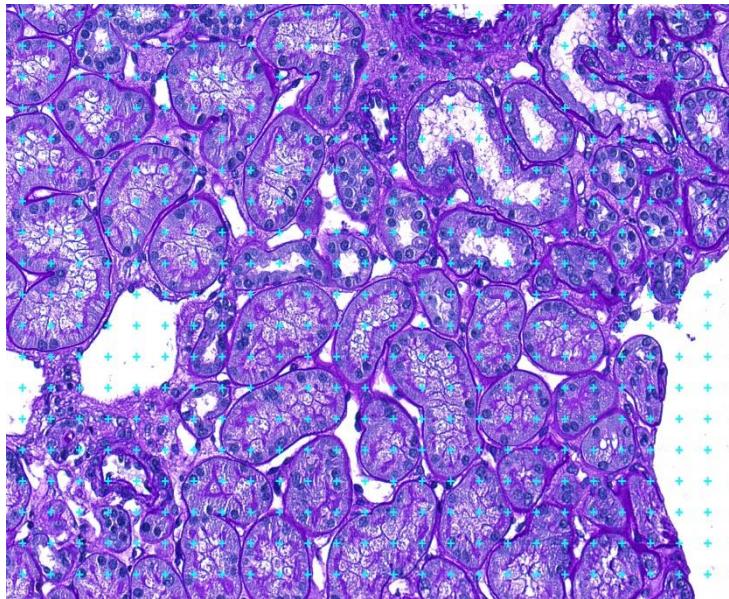
Healthy Subject



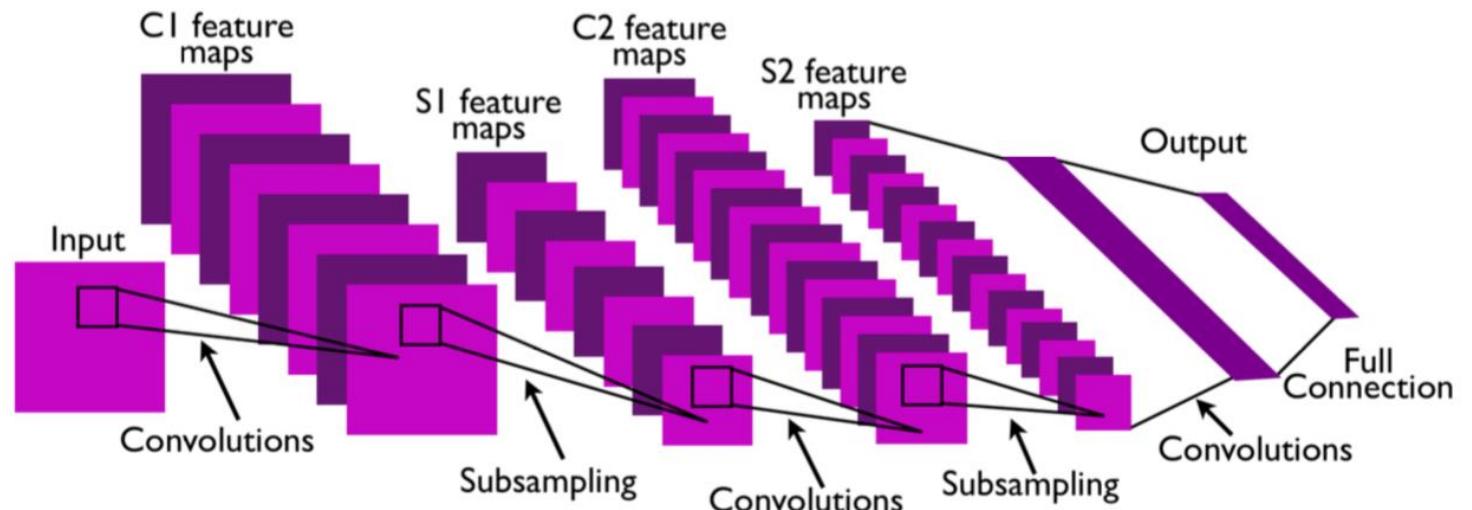
Patient



Fully Convolutional CNN from Grid annotations



- Crop the image around the grid points and assign to each patch the class of the central pixel
- Train a classification network with the patches



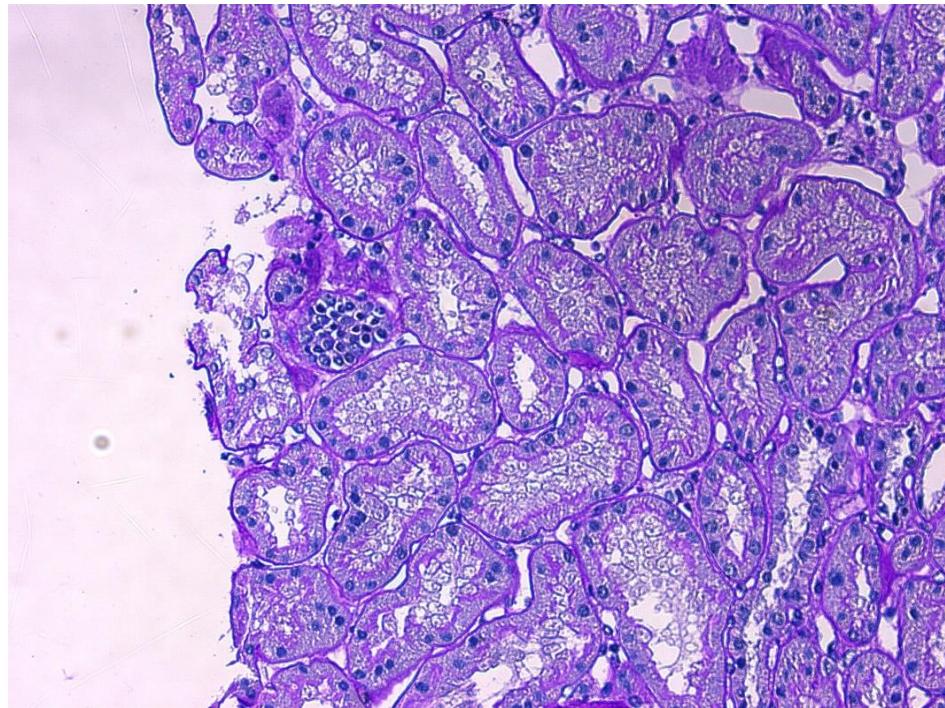
Fully Convolutionalization

- Modify the network to make it fully convolutional (substitute the dense layers with 1×1 convolutions)
- It is now possible to feed the network with images of any size, receiving in output an heatmap instead of a simple class prediction
- Apply a technique to upsample the heatmap and get a segmentation mask of the same size of the original image (for example shift and stitch)

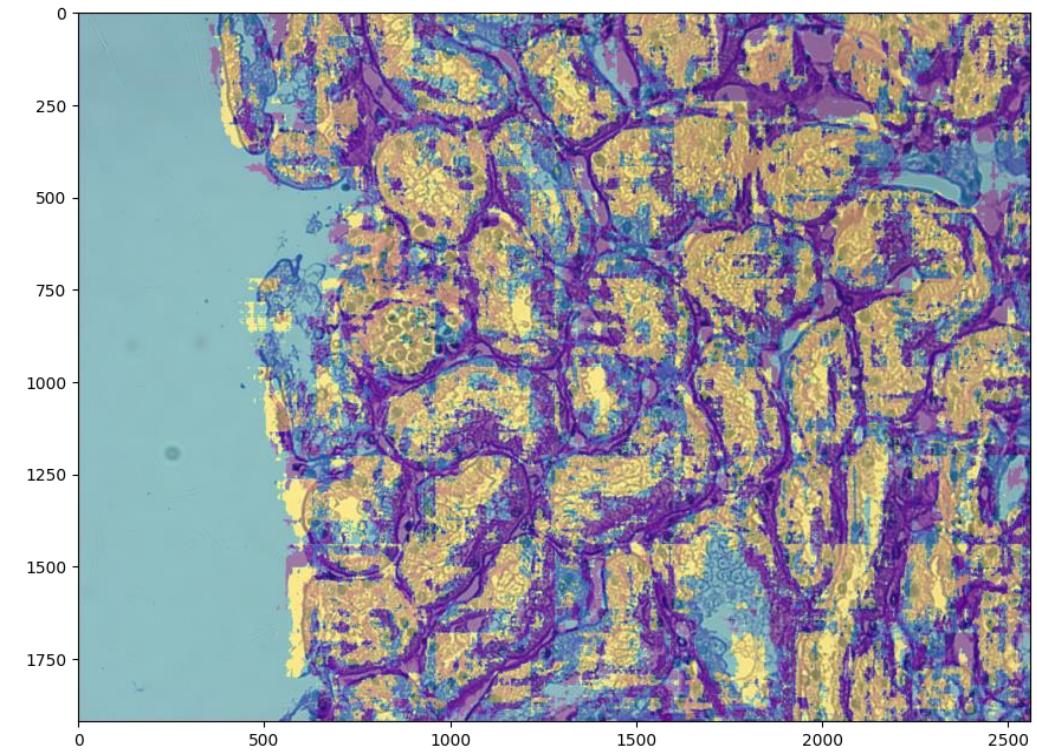
Shift and Stitch Results

Efficient inference by a vanilla classification network
trained on 30917 patches of size 200x200

Original image



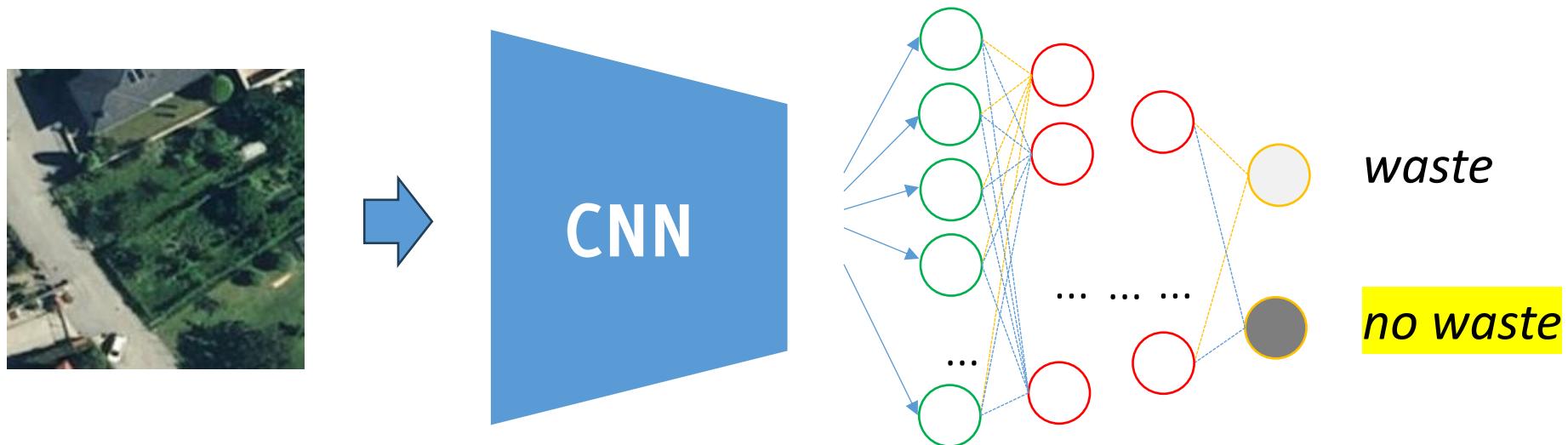
Shift and stitch result



Deep Learning to inspect Remote Sensing Images

EU funded project aiming at fighting environmental crime

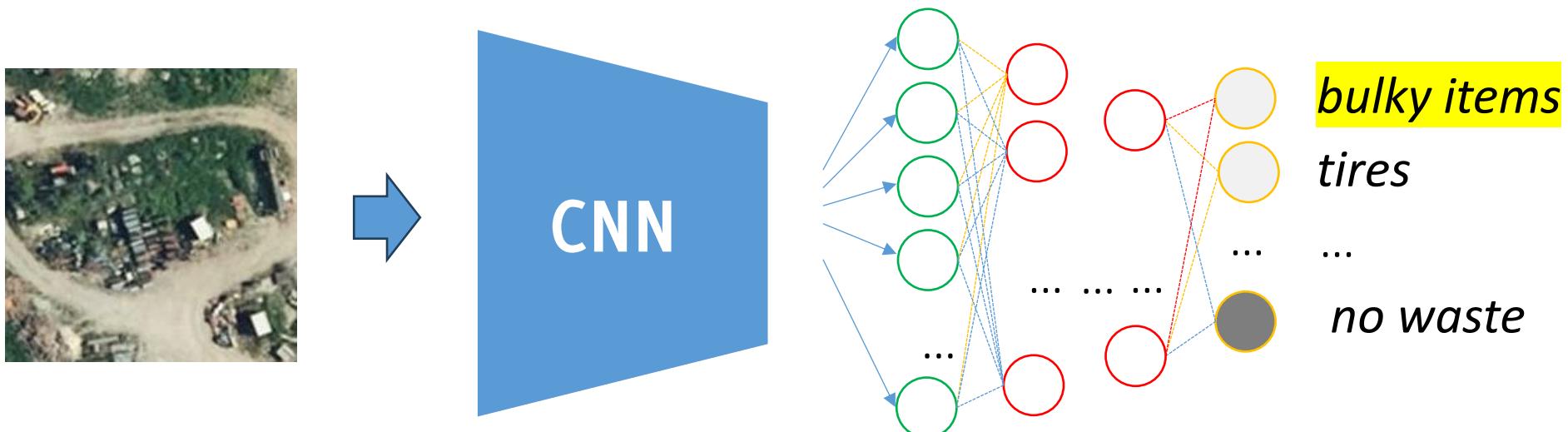
- Illegal landfills detection by DL models analyzing remote sensing images (VHR satellite images)
- Sparse annotations provided (landfills locations)

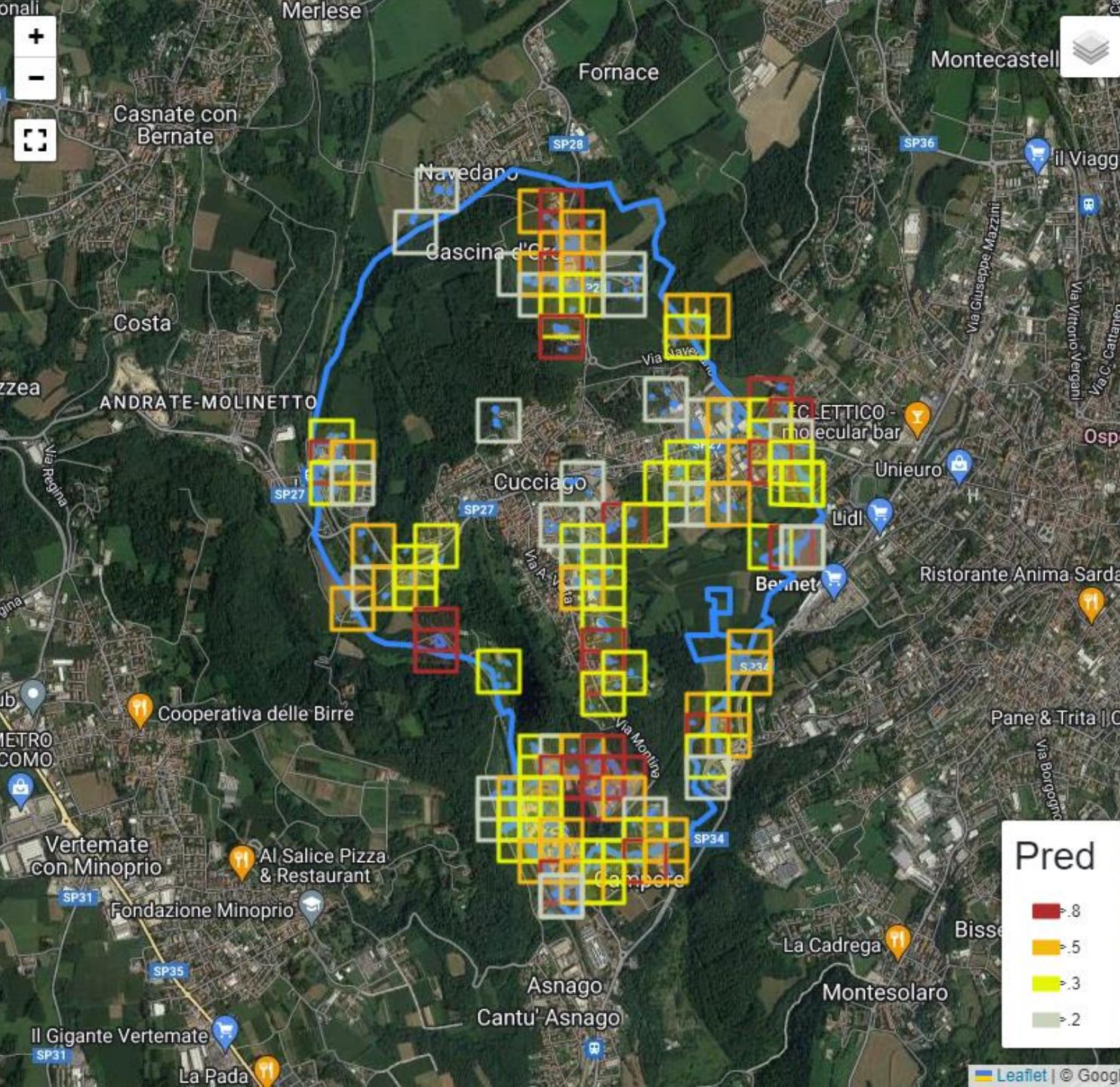


Deep Learning to inspect Remote Sensing Images

Addressing classification waste detection by patch classification

- extensions to segmentation by shift and stitch and *saliency maps* (see *next lecture*)

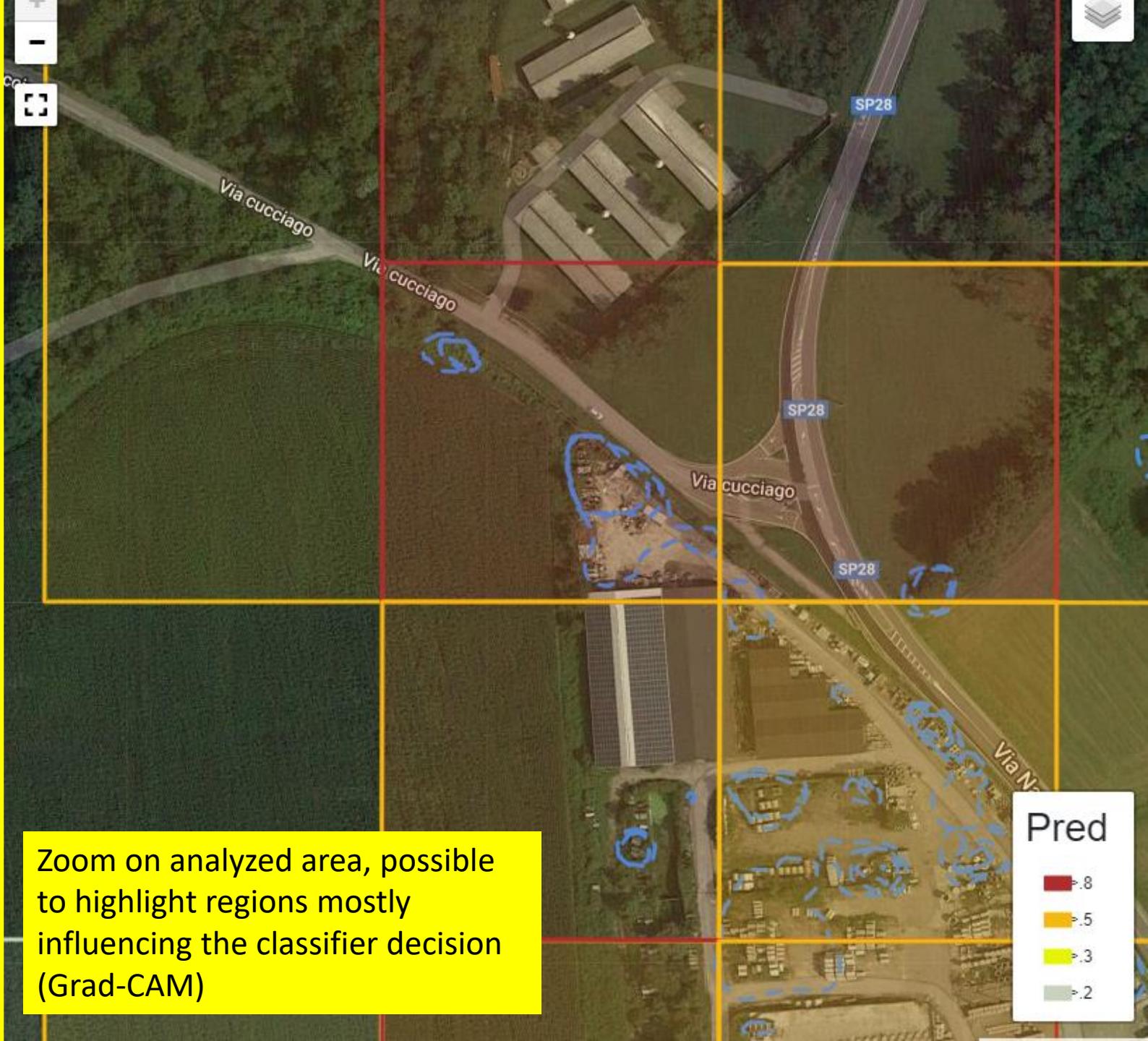
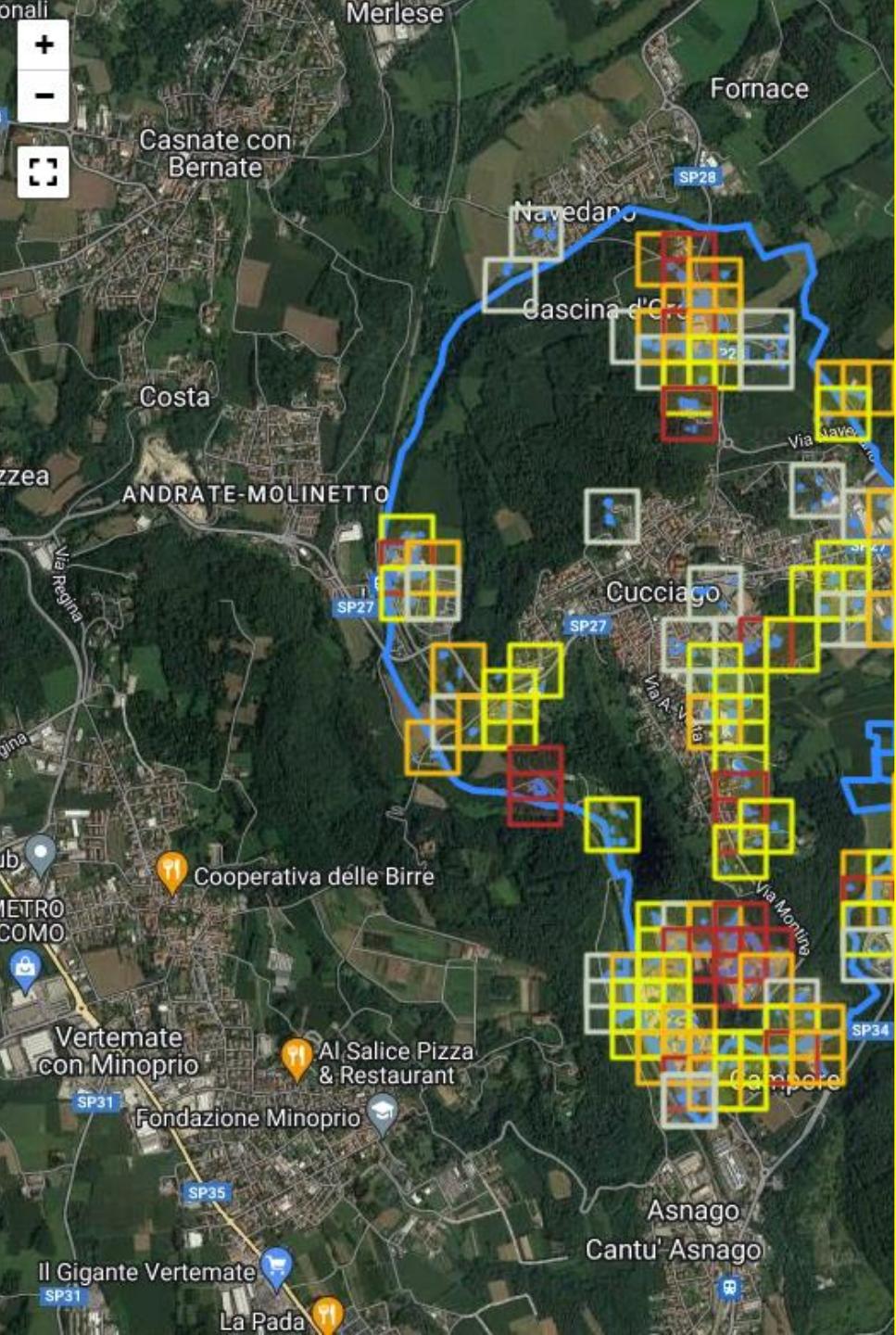




The ODIN Platform within Perivallon

- Tight collaboration with ARPA Lombardia to get annotations
- 100+ municipalities automatically scanned by our system on Google Earth images
- Regular surveys are planned to detect both existing and new waste disposal sites

T. R. Nahime, F. Milani, and P. Fraternali. "ODIN: Pluggable Meta-annotations and Metrics for the Diagnosis of Classification and Localization." *Int. Conf. on Machine Learning, Optimization, and Data Science*.



Zoom on analyzed area, possible
to highlight regions mostly
influencing the classifier decision
(Grad-CAM)

Pred

>.8

>.5

>.3

>.2

Examples of segmentation

