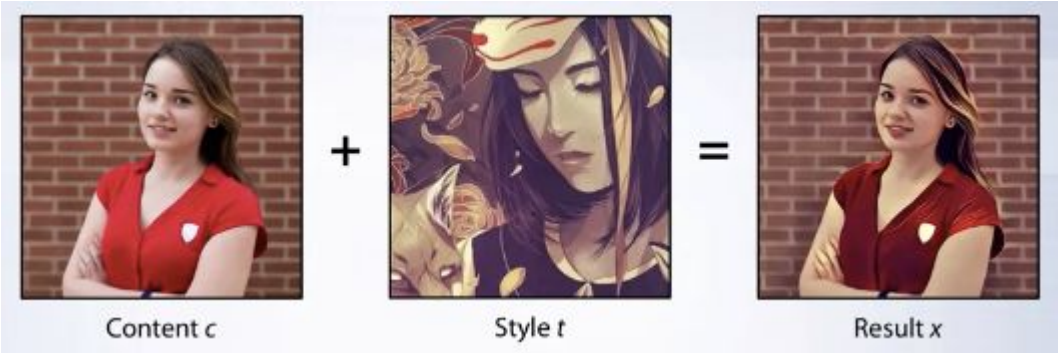
Computer vision: Segmentation and friends

Taras Khakhulin
Deep Learning Engineer
Samsung AI
MIPT and Skoltech Master

https://twitter.com/t_khakhulin
https://www.linkedin.com/in/taras-khakhulin/

Style Transfer Formulation



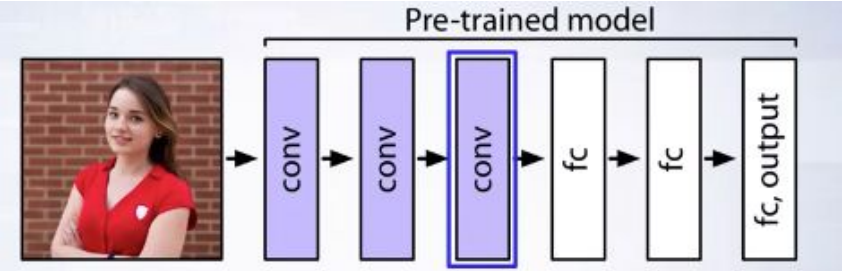
· Content c and style t are presented

· The goal: produce output x matching (to some extent) the content and (to some extent) the style

How to represent content and style similarity?

- Consider feature activations in a convolutional network!

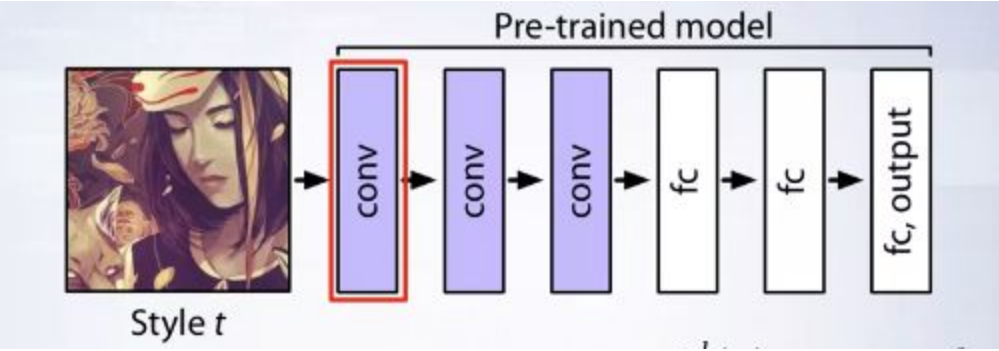
Style Transfer Loss



Extract content targets $F^l(x)$ from layer l : e.g. at conv $5 \rightarrow 13 \times 13 \times 128$ array
Evaluate content similarity between x and c using

$$L_{\text{content}}(x; c) = \left\| F^l(x) - F^l(c) \right\|_2^2$$

Style Transfer Loss

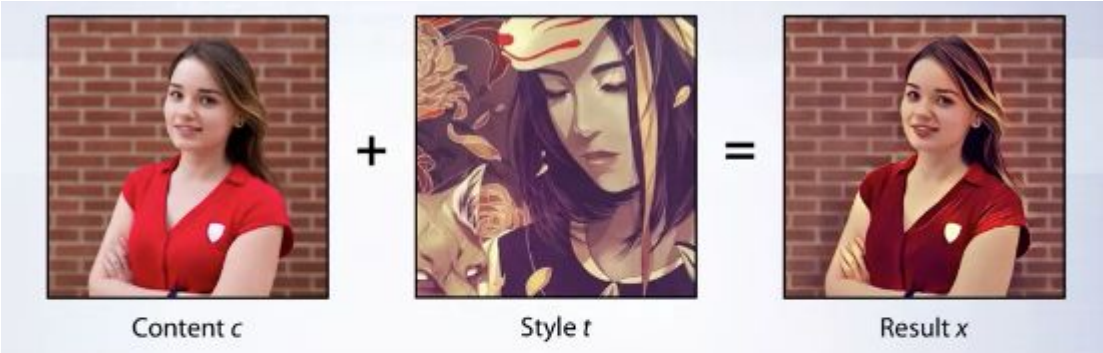


Extract style targets (Gram matrices) $G^l(x)$ from layer l
e.g. at conv1 with $224 \times 224 \times 64$ array of activations $\rightarrow 64 \times 64$ Gram matrix

$$G^l_{i,k} = \sum_k F^l_{i,k} F^l_{j,k}$$

Evaluate style similarity between x and t using

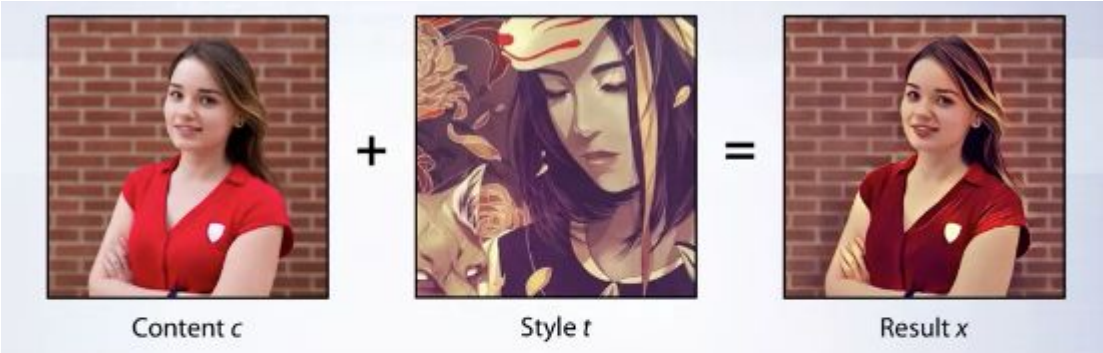
$$L_{style}(x;t) = \left\| G^l(x) - G^l(t) \right\|_2^2$$



style loss $L_{\text{style}}(x;t) = \left\|G^l(x) - G^l(t)\right\|_2^2$

Content loss : $L_{\text{content}}(x;c) = \left\|F^l(x) - F^l(c)\right\|_2^2$

Total loss : $L_{\text{total}}(x;c,t) = \alpha L_{\text{content}}(x;c) + \beta L_{\text{style}}(x;t)$

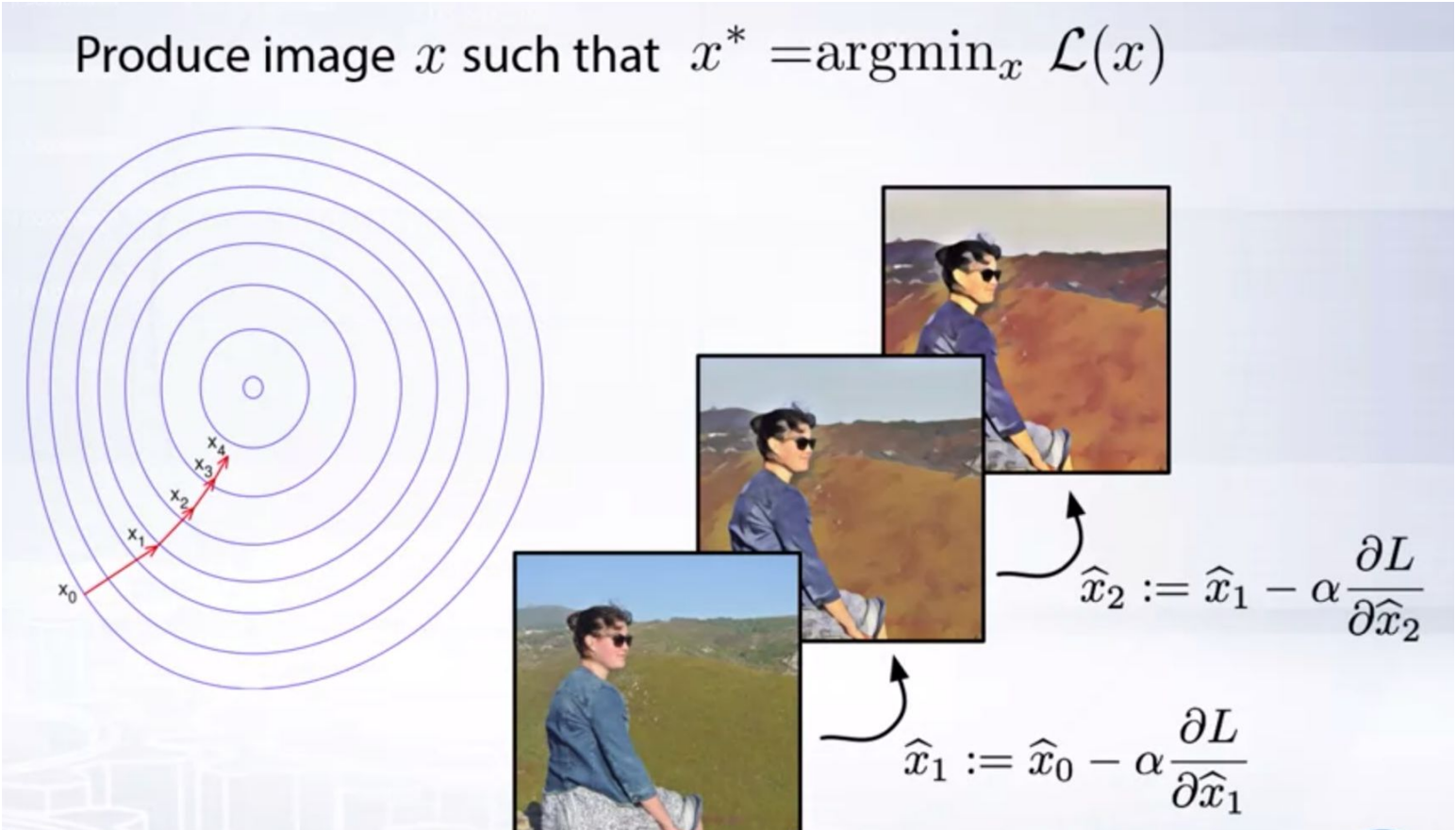


style loss $L_{\text{style}}(x;t) = \left\|G^l(x) - G^l(t)\right\|_2^2$

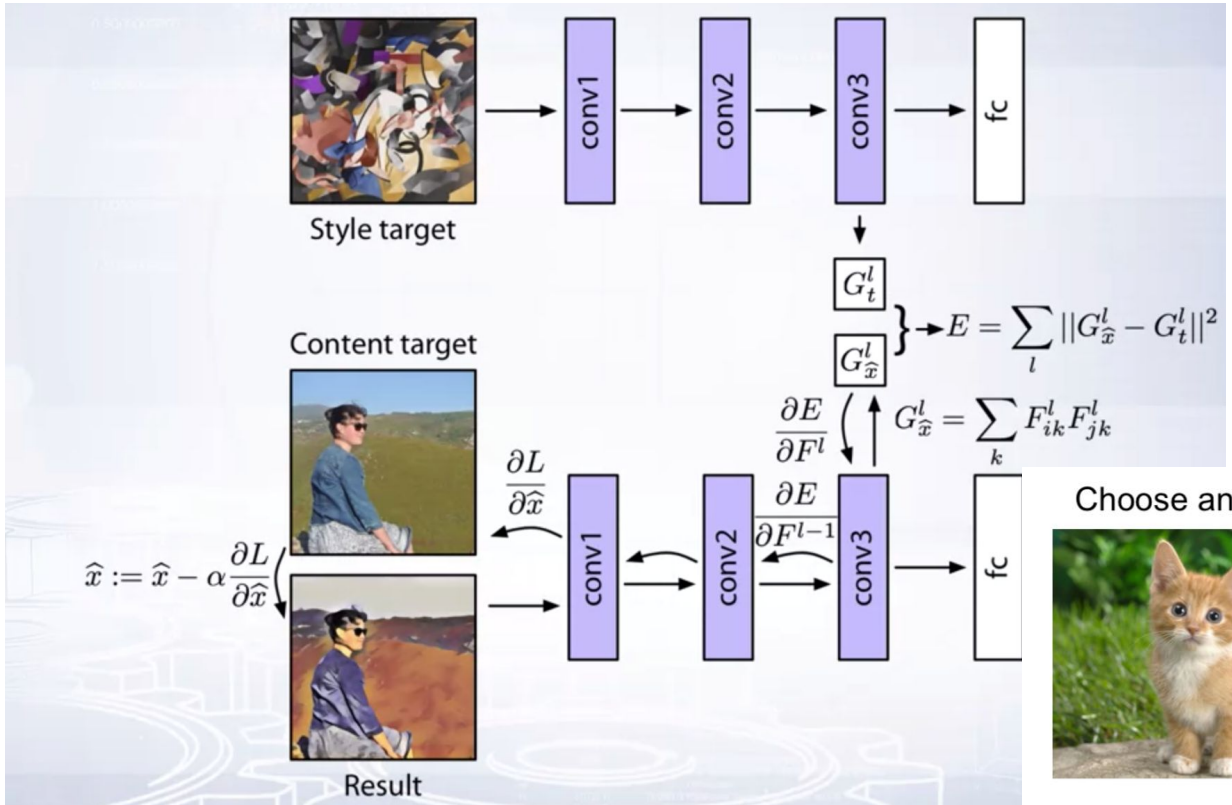
Content loss : $L_{\text{content}}(x;c) = \left\|F^l(x) - F^l(c)\right\|_2^2$ Enforce good-looking image

Total loss : $L_{\text{total}}(x;c,t) = \alpha L_{\text{content}}(x;c) + \beta L_{\text{style}}(x;t)$

Illustration



Illustration



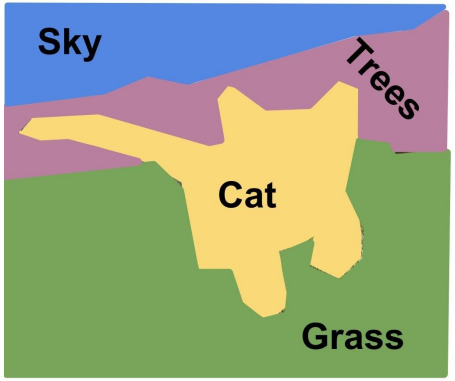
Style transfer is an optimization technique used to paint the natural images using a style of an artwork:

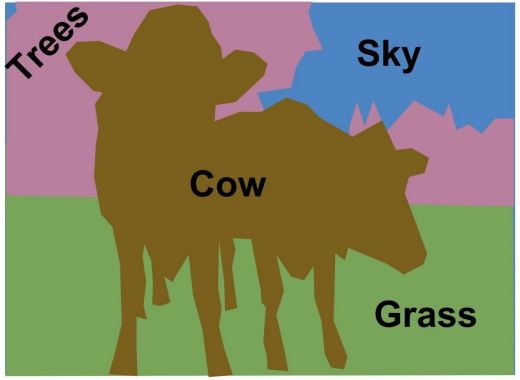
- it aims to produce an image close to the two target images
- the style and the content targets.
- we match feature activations induced by the generated and the target images at two different layers in a pretrained CNN.
- The image is generated via gradient descent in the pixel space.

Segmentation Outline

- Segmentation task
- Simple solutions
- Upsampling methods
  - Unpooling
  - Transposed convolution
- FCN
- DeconvNet
- SegNet
- U-Net
- Mask R-CNN

Semantic Segmentation





Input image (C x H x W)

Output

mask of classes (1 x H x W)

Assumption

just class label for each pixel

Instance Segmentation

Input image (C x H x W)

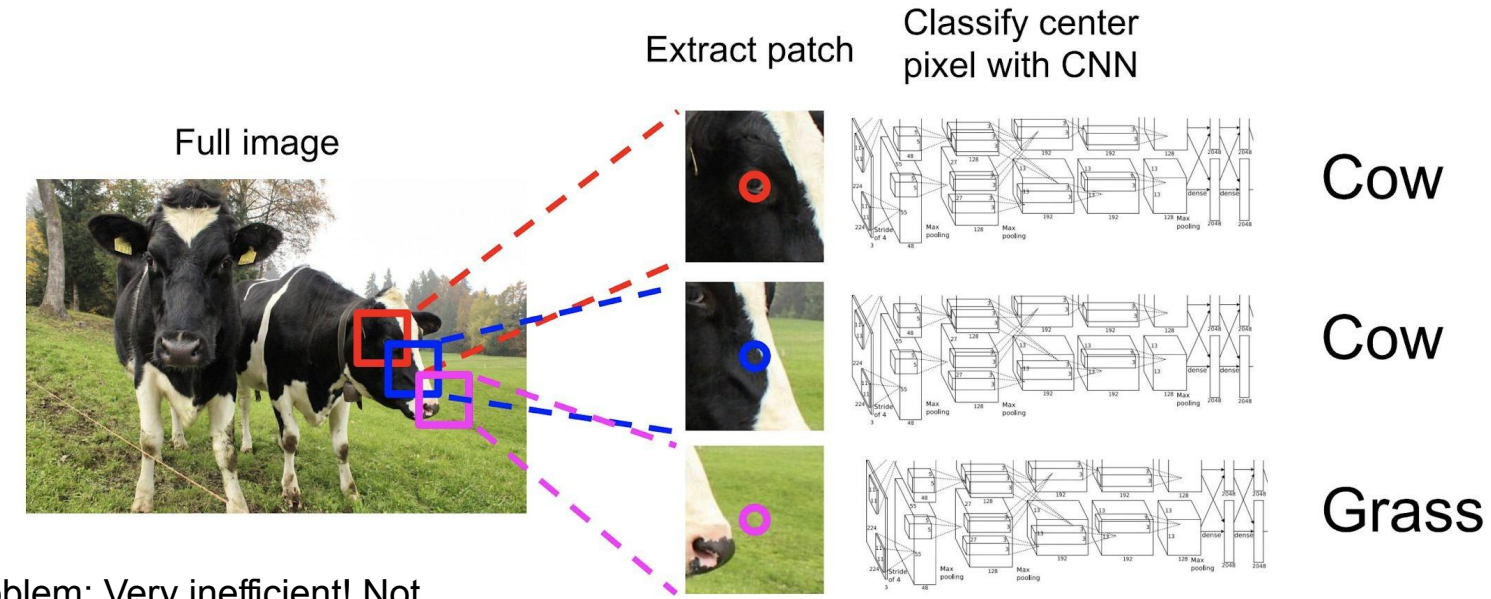
Output

mask of class instances
(detection + 1 x H x W)

Assumption

separated masks for each distinct object

Naive solution

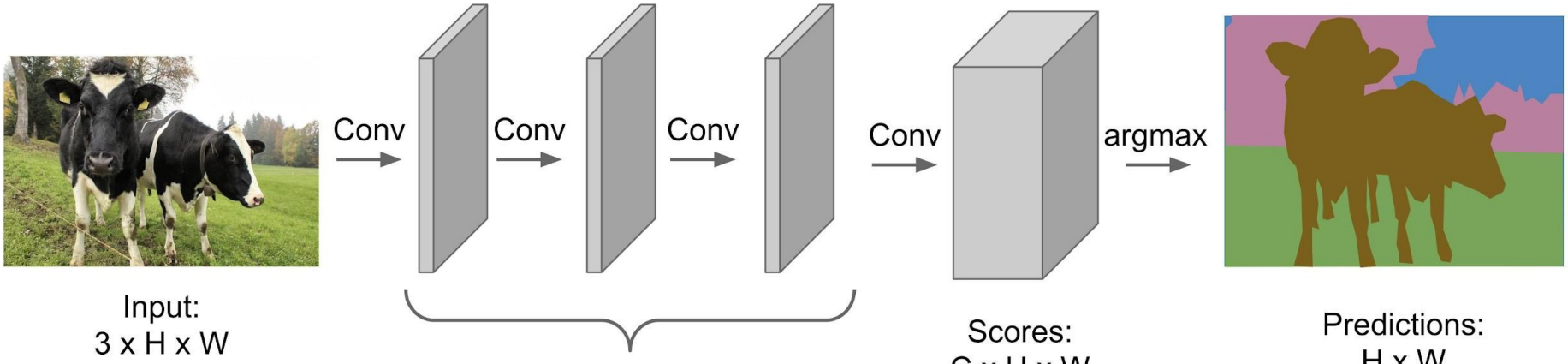


Problem: Very inefficient! Not reusing shared features between overlapping patches (same as

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

Design a network as a bunch of convolutional layers to make predictions for pixels all at once!

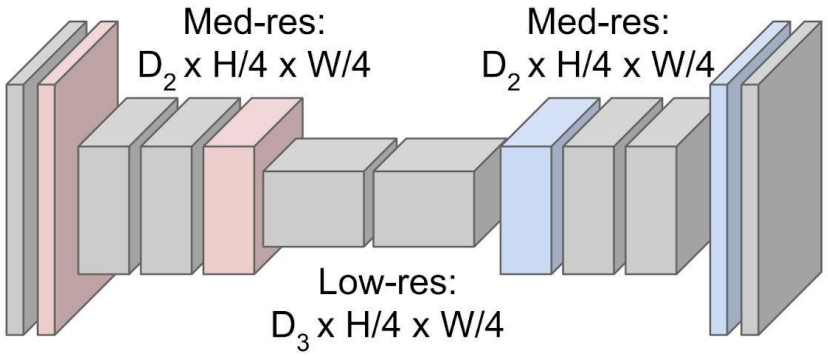


Problem: convolutions at original image resolution will be very expensive (remember VGG architecture)

Downsampling and Upsampling

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Input:
3 x H x W

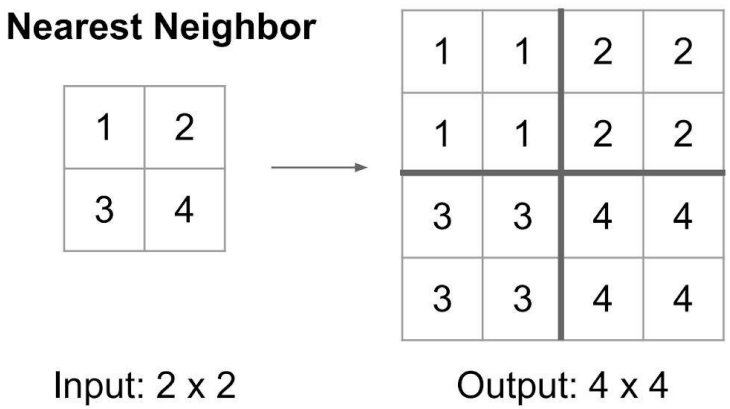


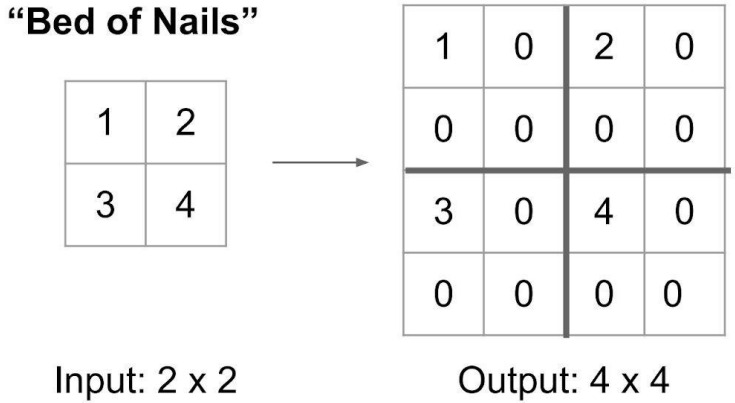
High-res:
D_1 x H/2 x W/2

High-res:
D_1 x H/2 x W/2

Predictions:
H x W

Simple unpooling





Max Unpooling

Max Pooling
Remember which element was max!

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

Input: 4 x 4

5	6
7	8

Output: 2 x 2

...

Rest of the network

Max Unpooling
Use positions from pooling layer

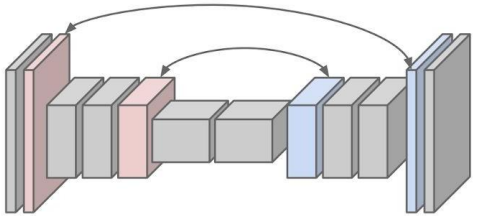
1	2
3	4

Input: 2 x 2

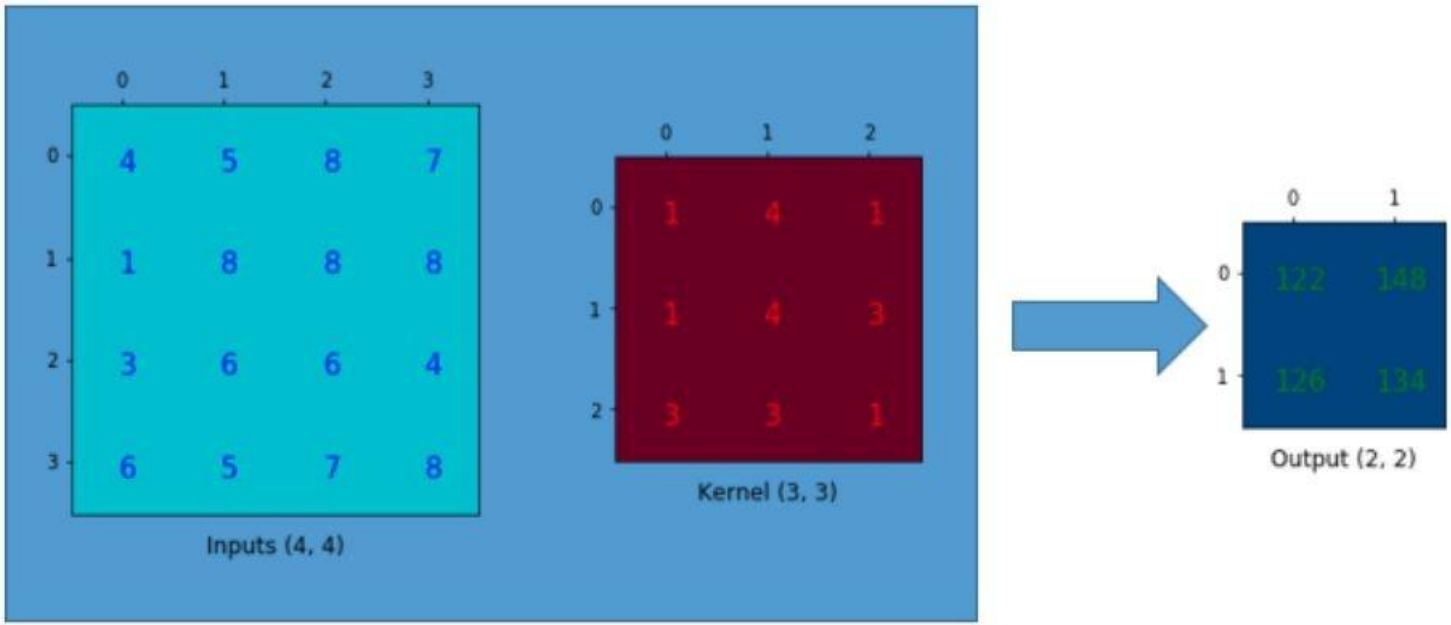
0	0	2	0
0	1	0	0
0	0	0	0
3	0	0	4

Output: 4 x 4

Corresponding pairs of downsampling and upsampling layers

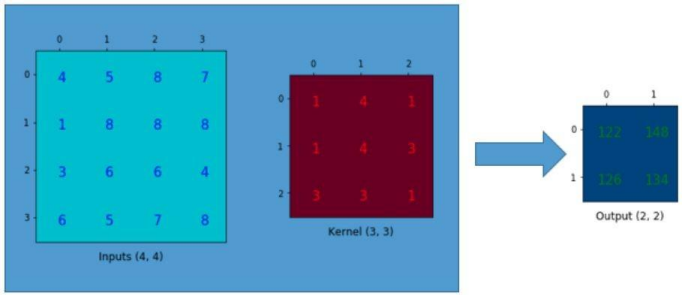


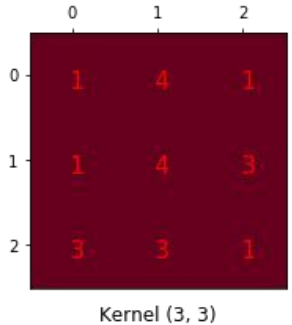
Regular convolution

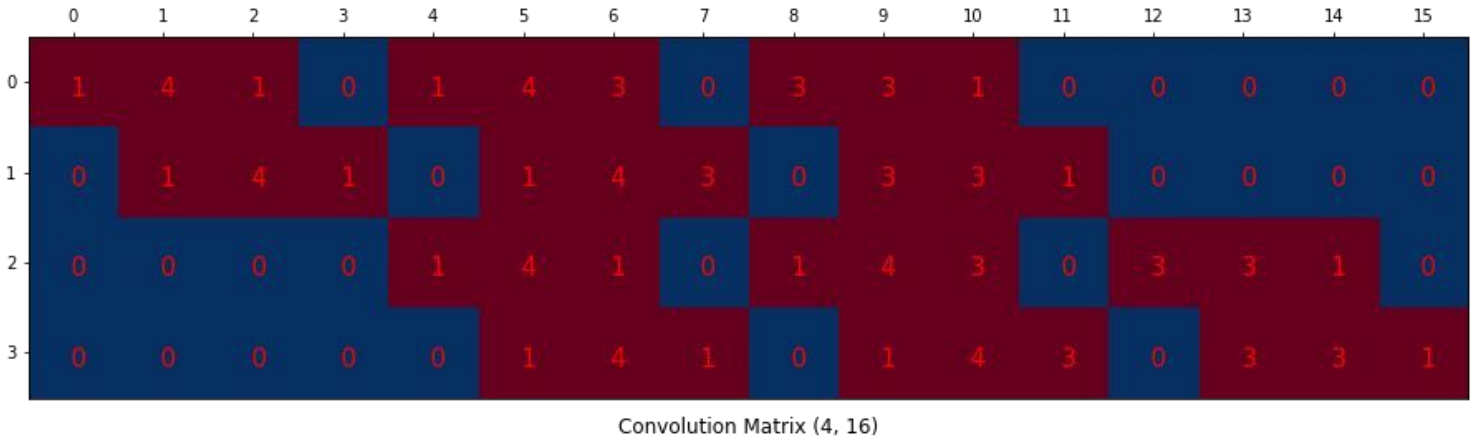


[Images credentials]

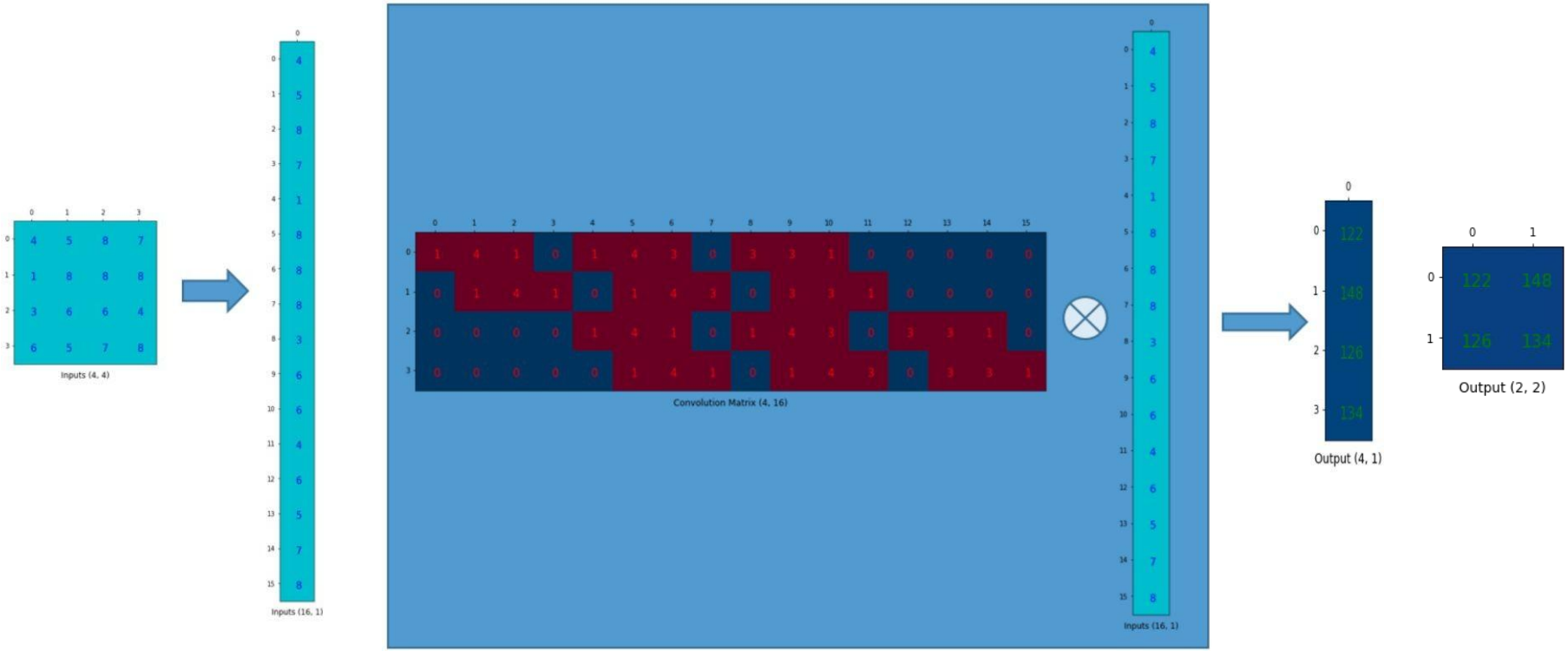
Regular convolution: matrix form



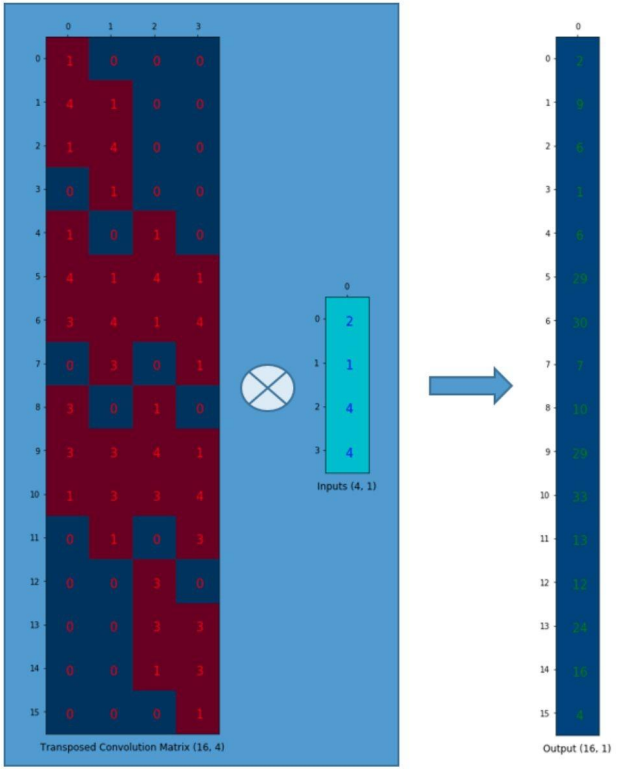


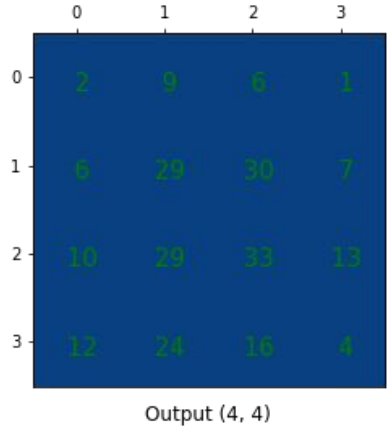


Regular convolution: matrix form



Transposed convolution matrix





Convolution as Matrix Multiplication (1D)

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & x & 0 & 0 & 0 \\ 0 & x & y & x & 0 & 0 \\ 0 & 0 & x & y & x & 0 \\ 0 & 0 & 0 & x & y & x \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=1, padding=1

Convolution transpose multiplies by the transpose of the same matrix:

$$\vec{x} *^T \vec{a} = X^T\vec{a}$$

$$\begin{bmatrix} x & 0 & 0 & 0 \\ y & x & 0 & 0 \\ z & y & x & 0 \\ 0 & z & y & x \\ 0 & 0 & z & y \\ 0 & 0 & 0 & z \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} ax \\ ay + bx \\ az + by + cx \\ bz + cy + dx \\ cz + dy \\ dz \end{bmatrix}$$

When stride=1, convolution transpose is just a regular convolution (with different padding rules)

Terms

Some researchers denote Transposed convolution with term “Deconvolution”. However:

deconvolution is an algorithm-based process used to reverse the effects of convolution on recorded data [Wikipedia]

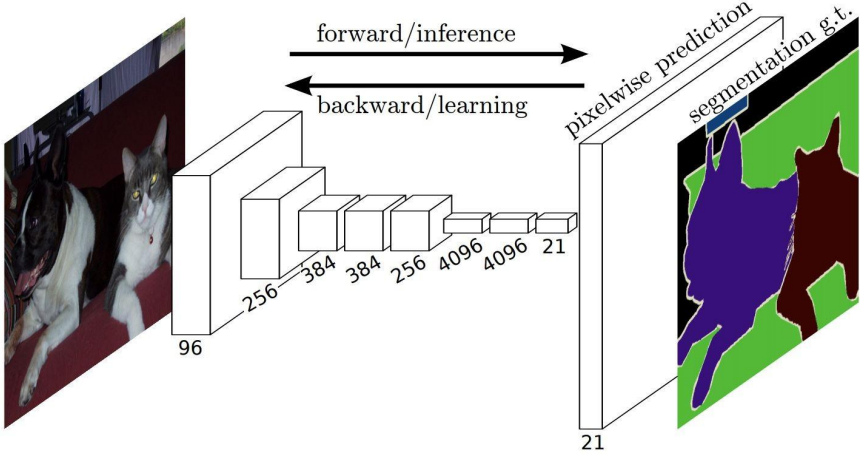
What we do is not a deconvolution.

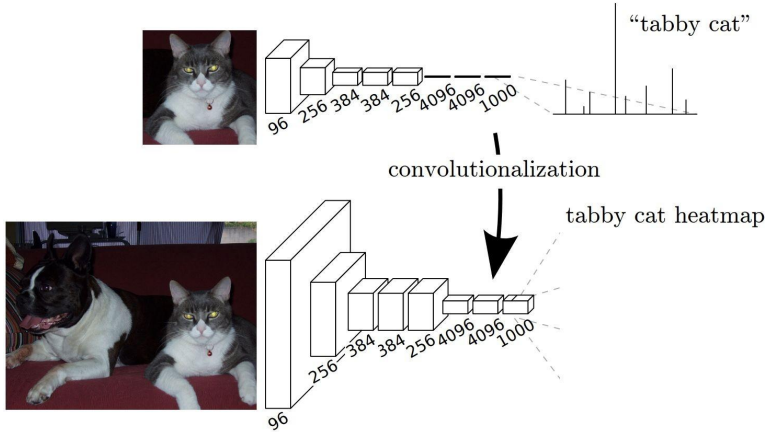
Please be precise in terms

Right terms:

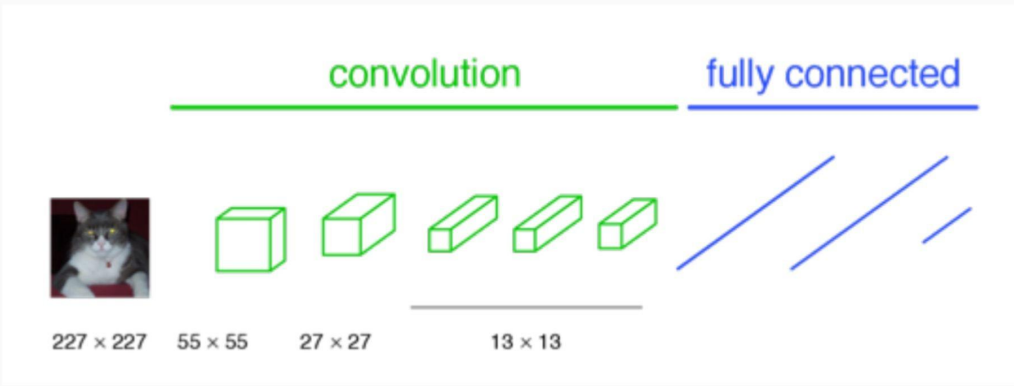
- Transposed convolution
- Upconvolution

FCN - Fully Convolutional Network

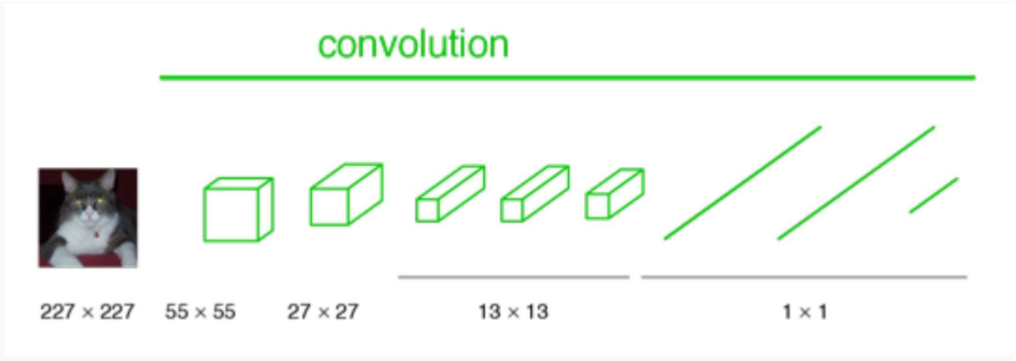


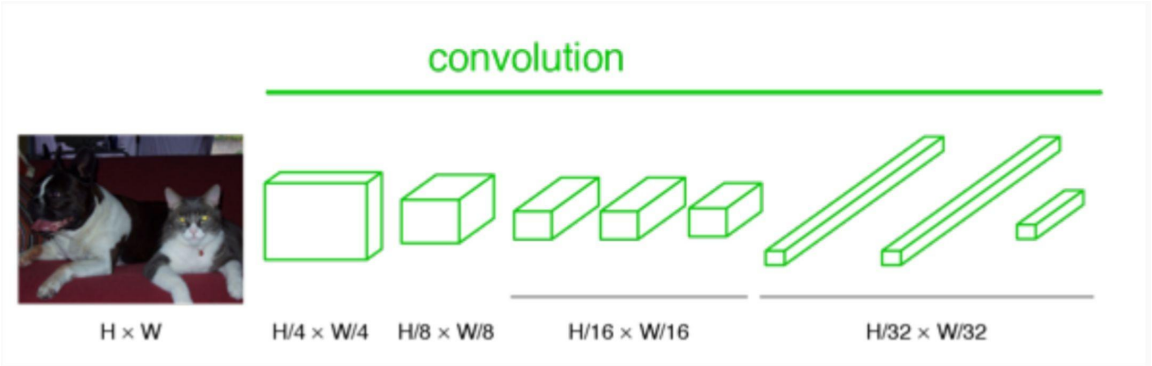


Regular CNN to FCN

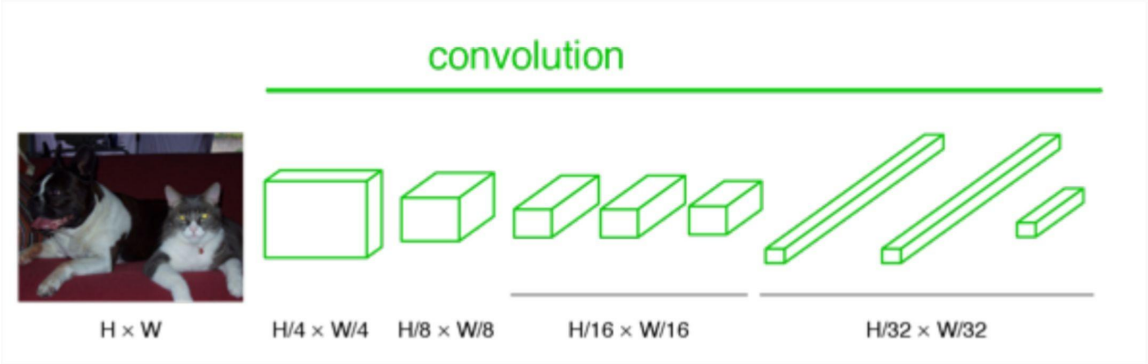


Regular CNN to FCN



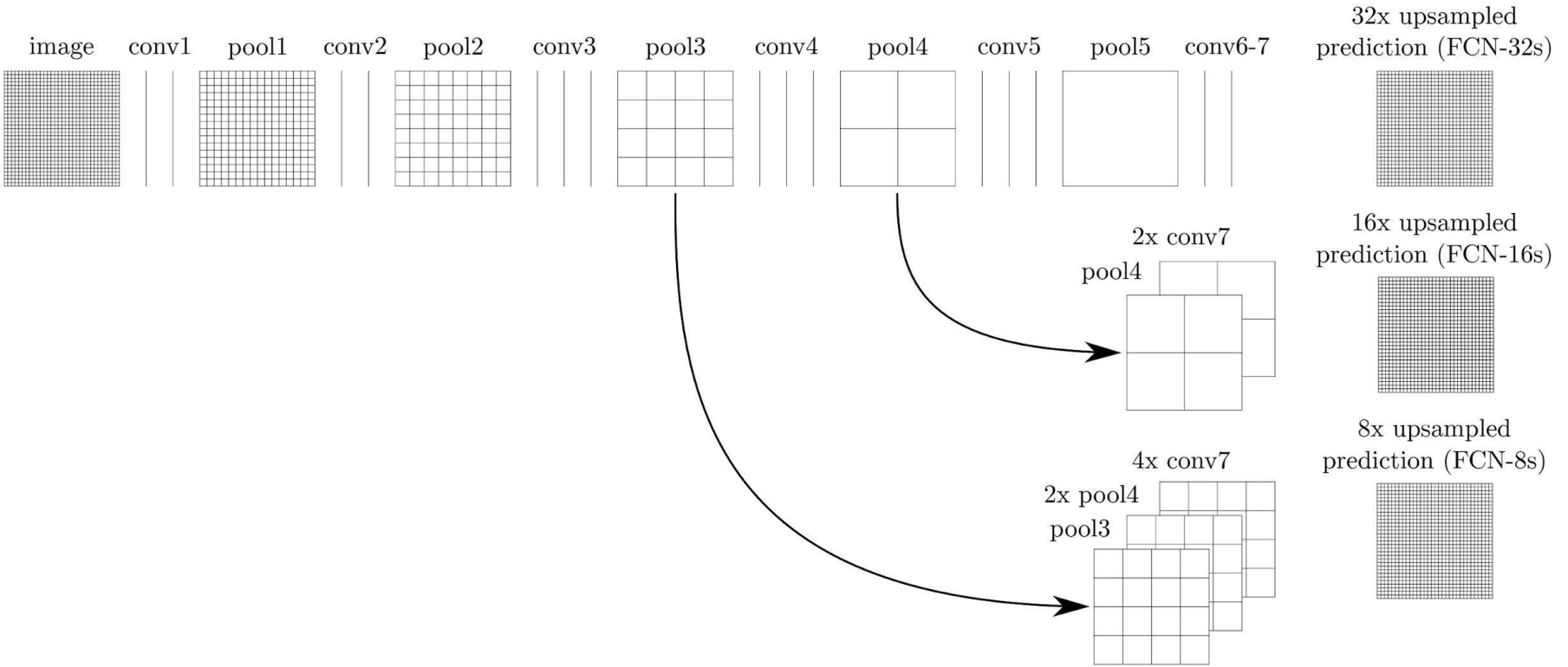


Regular CNN to FCN

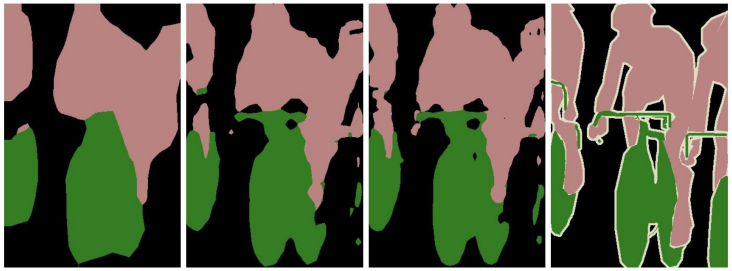


Problem: we lose resolution

FCN: Introduce skip connections

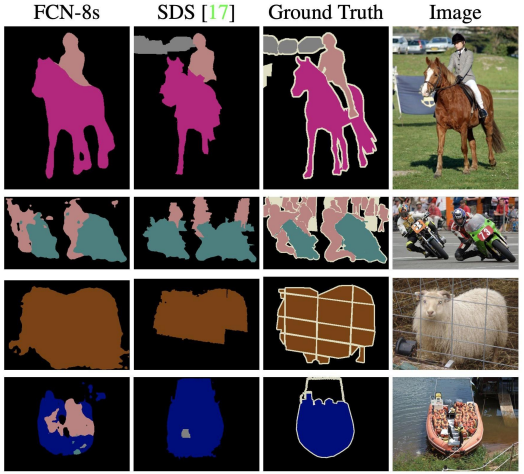


FCN: Results

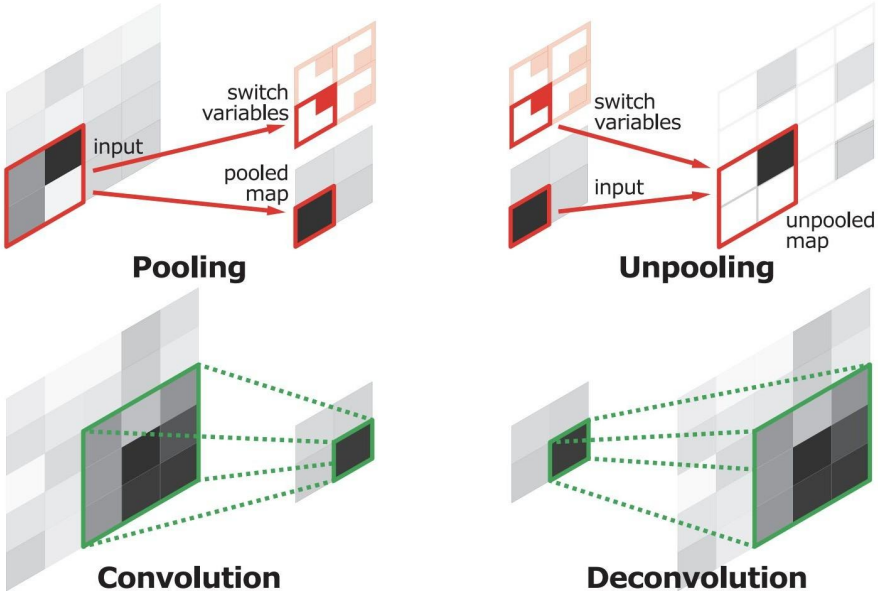


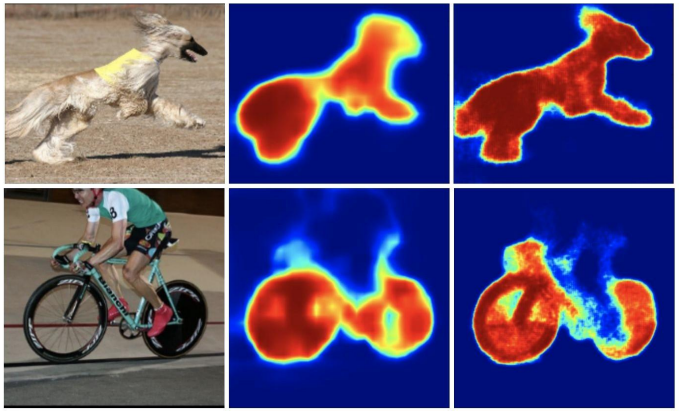
	pixel acc.	mean acc.	mean IU	f.w. IU
FCN-32s-fixed	83.0	59.7	45.4	72.0
FCN-32s	89.1	73.3	59.4	81.4
FCN-16s	90.0	75.7	62.4	83.0
FCN-8s	**90.3**	**75.9**	**62.7**	**83.2**

	FCN-AlexNet	FCN-VGG16	FCN-GoogLeNet[4]
mean IU	39.8	**56.0**	42.5
forward time	50 ms	210 ms	59 ms
conv. layers	8	16	22
parameters	57M	134M	6M
rf size	355	404	907
max stride	32	32	32



DeconvNet: use Max Unpooling

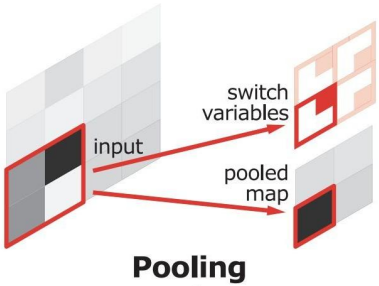


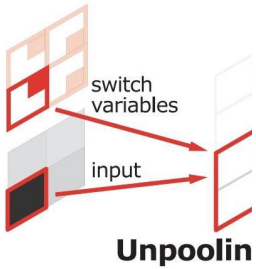


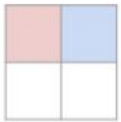
(a) Input image (b) FCN-8s (c) Ours

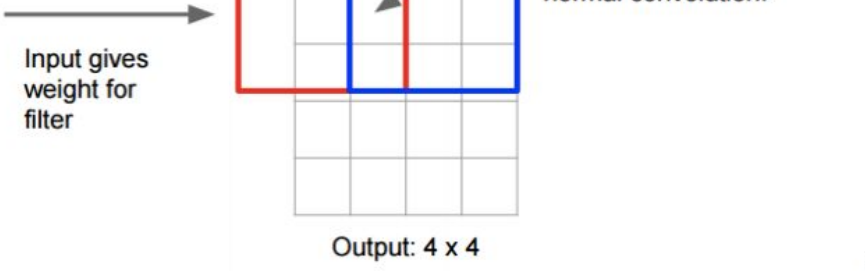
Figure 5. Comparison of class conditional probability maps from FCN and our network (top: dog, bottom: bicycle).

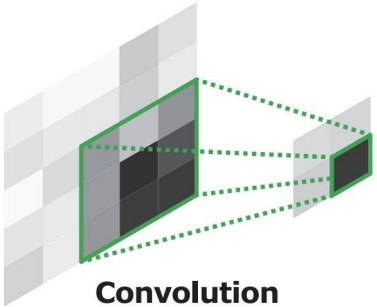
DeconvNet: use Max Unpooling

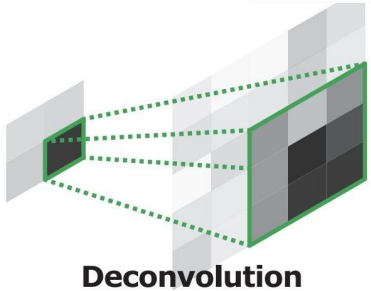












DeconvNet: use Max Unpooling

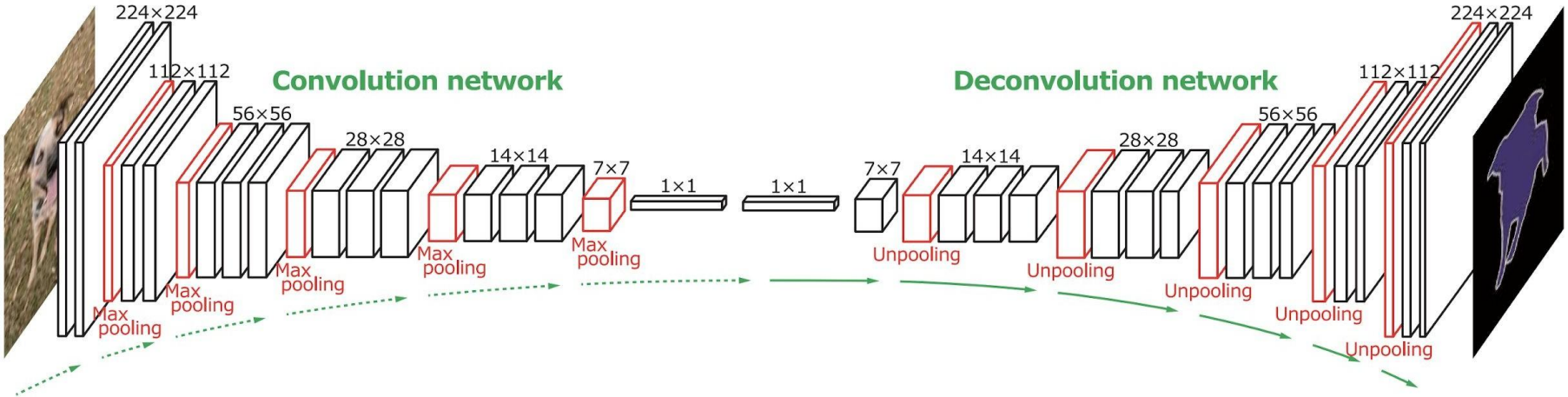
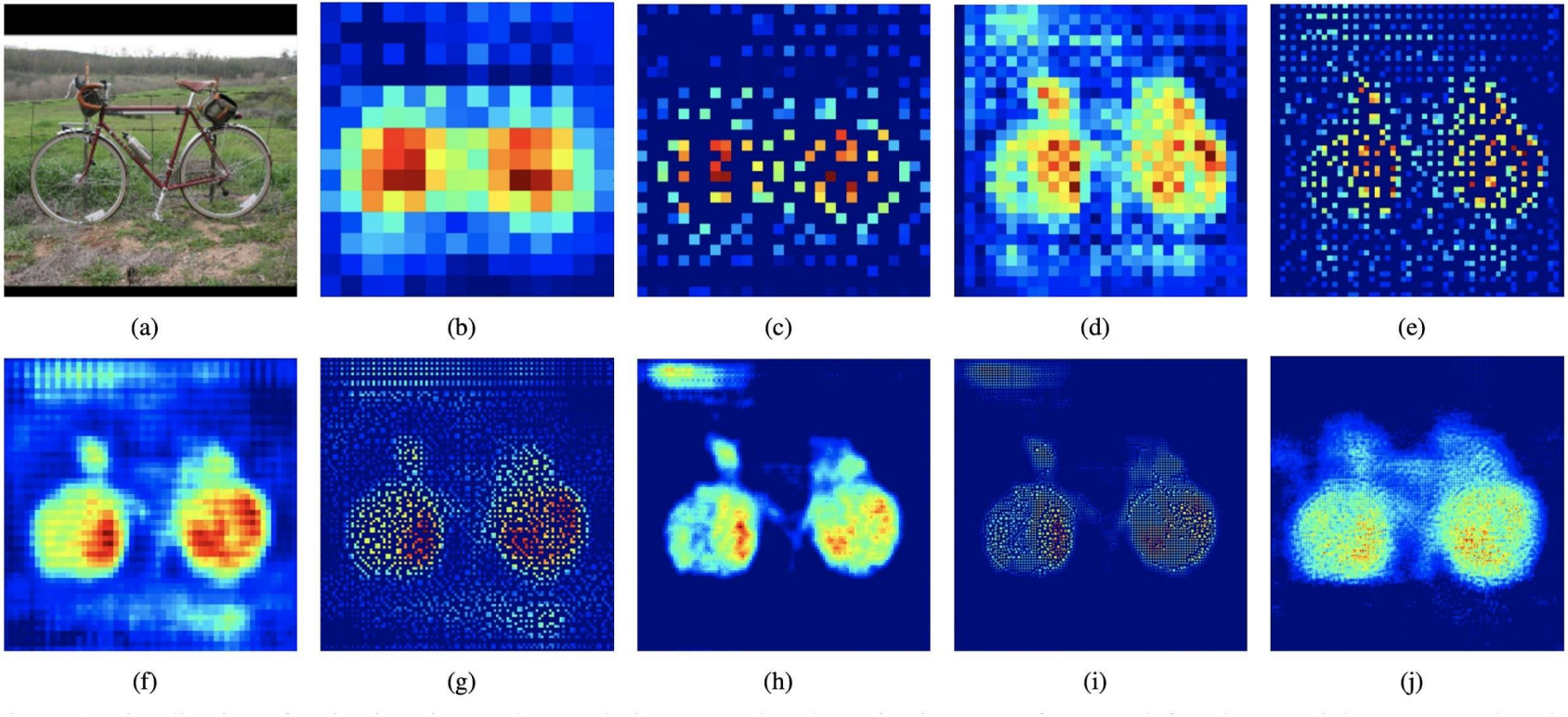


Figure 2. Overall architecture of the proposed network. On top of the convolution network based on VGG 16-layer net, we put a multi-layer deconvolution network to generate the accurate segmentation map of an input proposal. Given a feature representation obtained from the convolution network, dense pixel-wise class prediction map is constructed through multiple series of unpooling, deconvolution and rectification operations.

DeconvNet: use Max Unpooling



SegNet: Further development

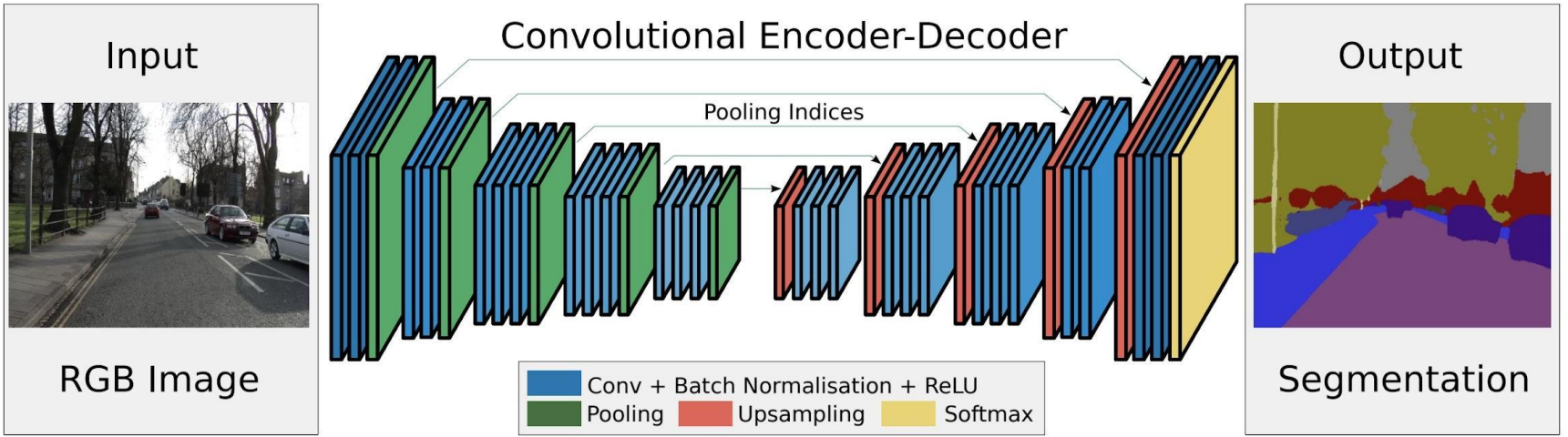
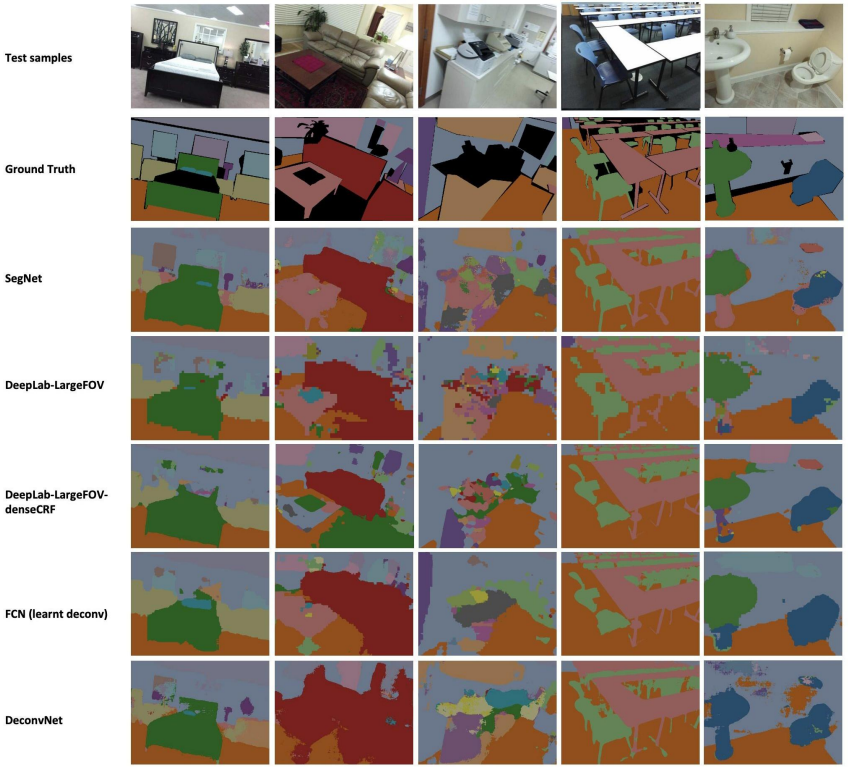
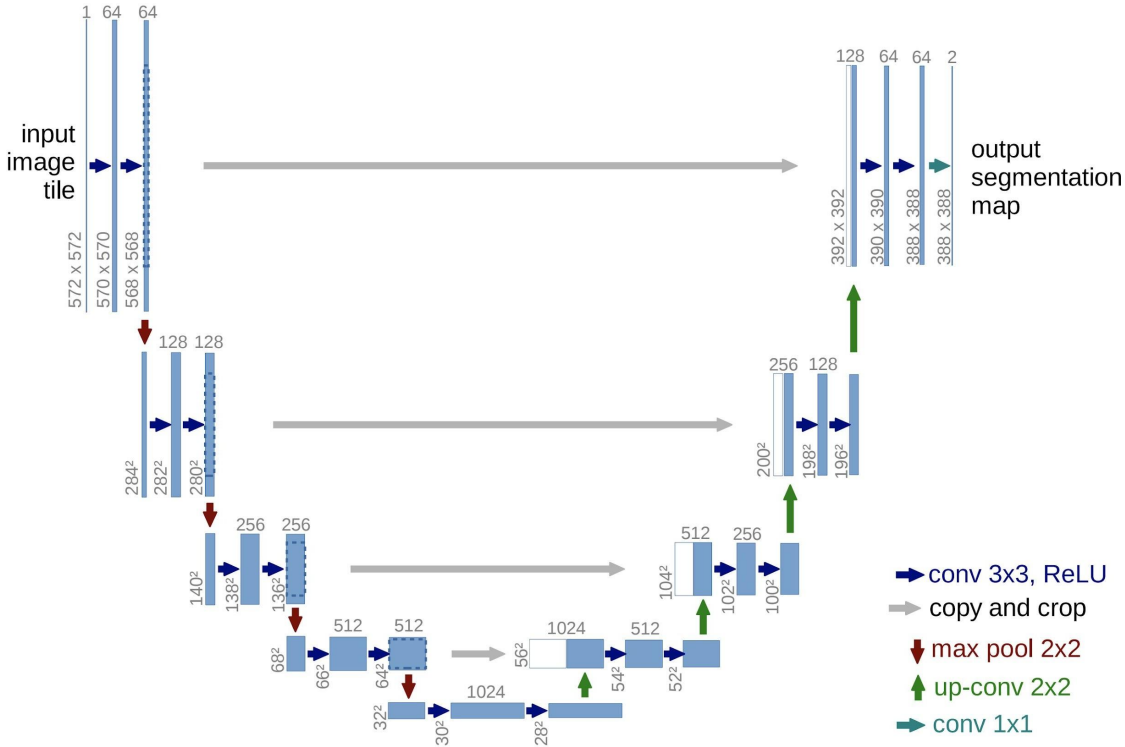


Fig. 2. An illustration of the SegNet architecture. There are no fully connected layers and hence it is only convolutional. A decoder upsamples its input using the transferred pool indices from its encoder to produce a sparse feature map(s). It then performs convolution with a trainable filter bank to densify the feature map. The final decoder output feature maps are fed to a soft-max classifier for pixel-wise classification.

SegNet: Further development



U-Net


input image tile
output segmentation map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

U-Net

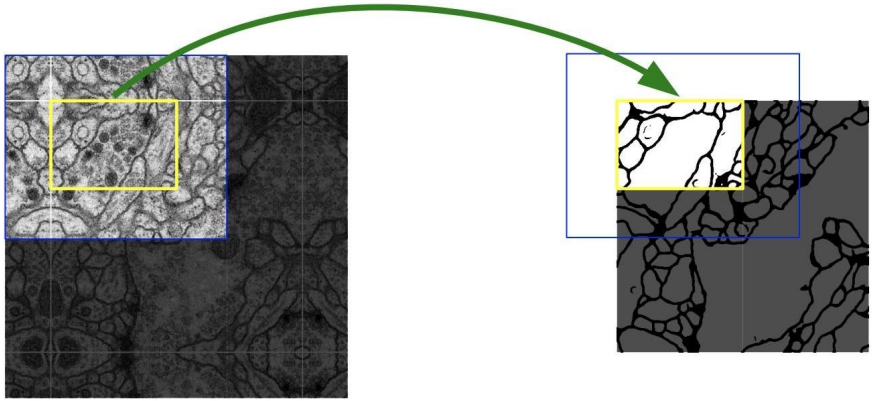
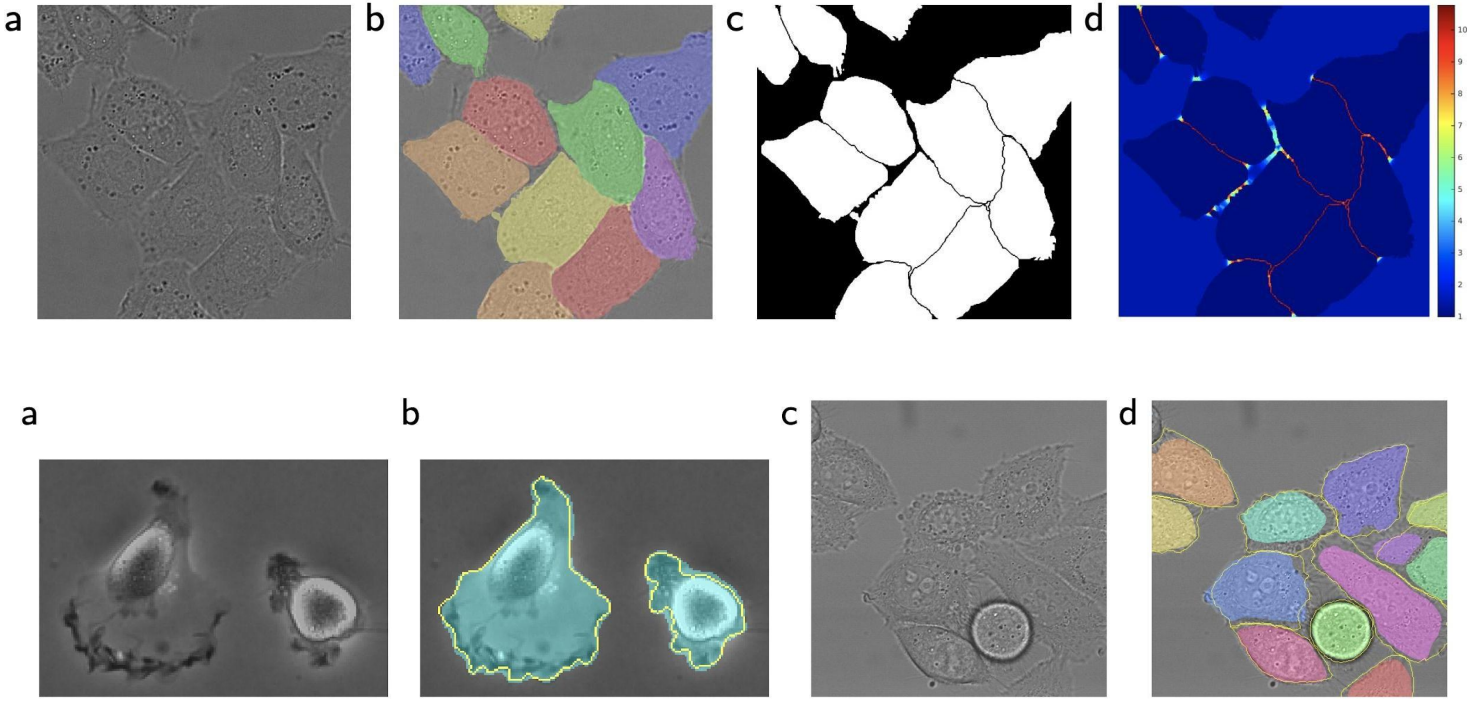


Fig. 2. Overlap-tile strategy for seamless segmentation of arbitrary large images (here segmentation of neuronal structures in EM stacks). Prediction of the segmentation in the yellow area, requires image data within the blue area as input. Missing input data is extrapolated by mirroring

U-Net



If you don’t understand something, we will implement all these models in the seminar

DeTR: Attention really is all you need

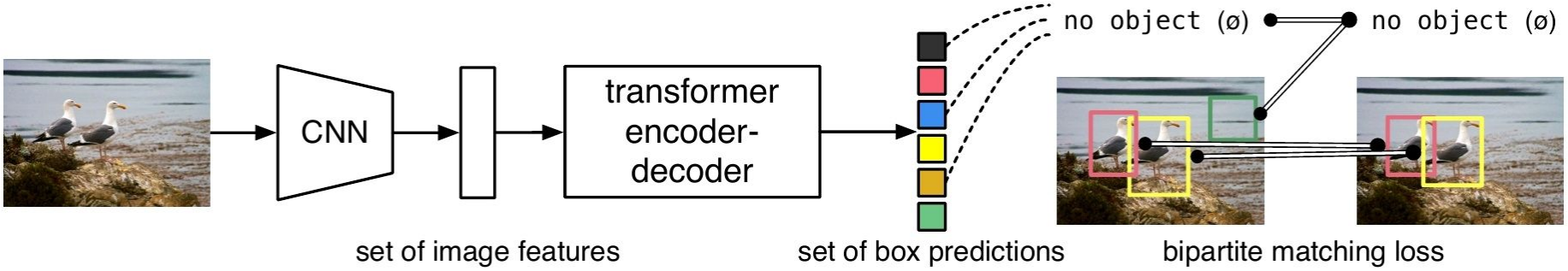


Fig. 1: DETR directly predicts (in parallel) the final set of detections by combining a common CNN with a transformer architecture. During training, bipartite matching uniquely assigns predictions with ground truth boxes. Prediction with no match should yield a "`no object`" () class prediction.

ArXiv: arxiv.org/abs/2005.12872

DeTR: Attention really is all you need

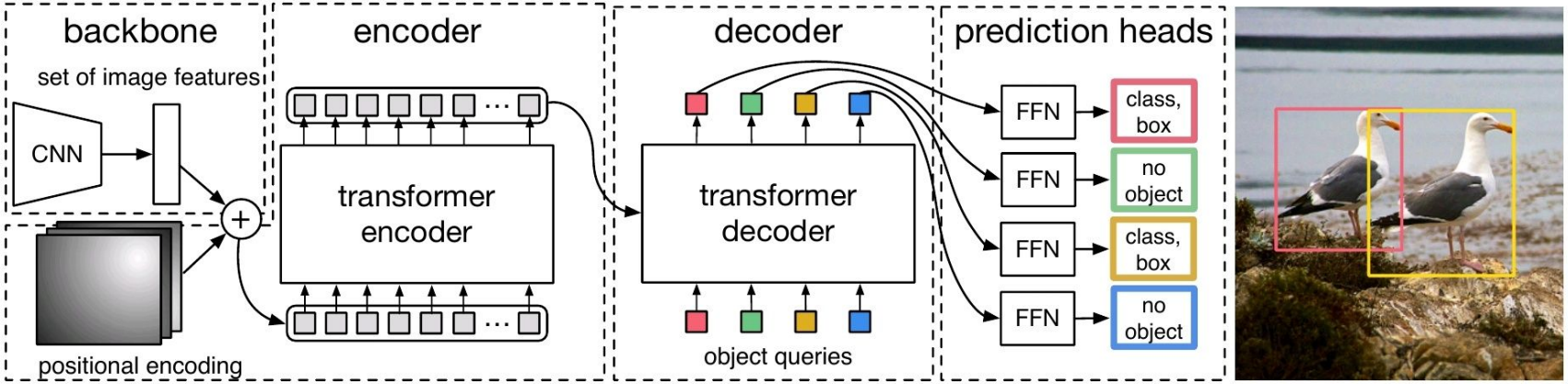
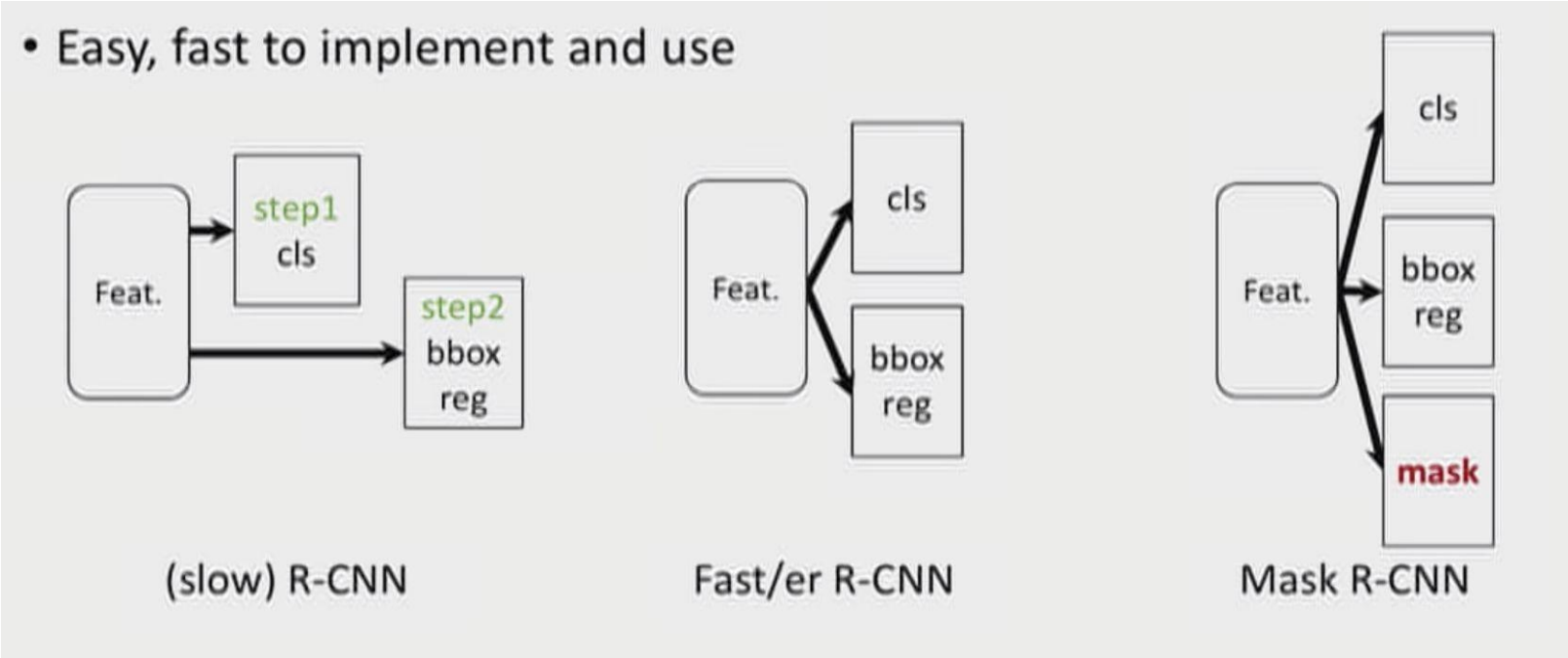
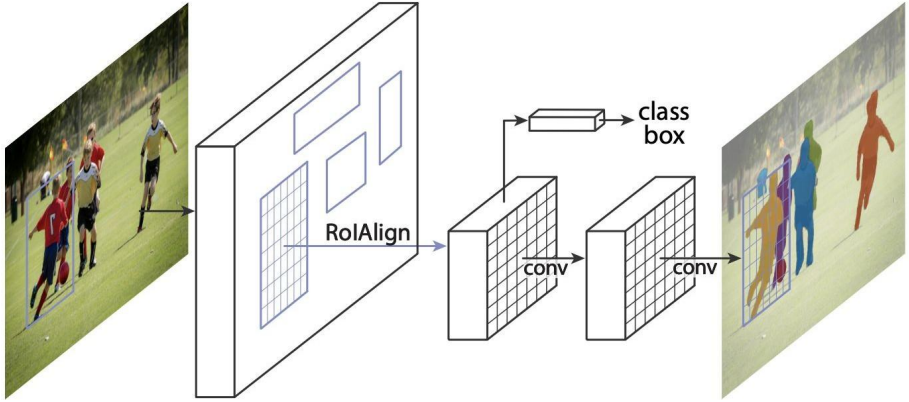


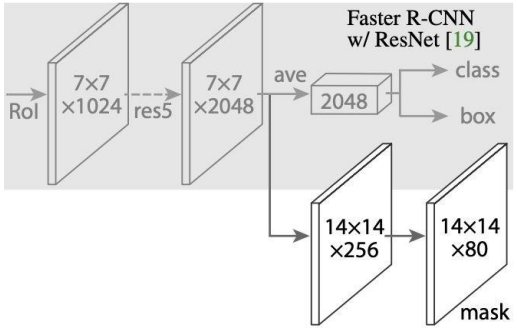
Fig. 2: DETR uses a conventional CNN backbone to learn a 2D representation of an input image. The model flattens it and supplements it with a positional encoding before passing it into a transformer encoder. A transformer decoder then takes as input a small fixed number of learned positional embeddings, which we call *object queries*, and additionally attends to the encoder output. We pass each output embedding of the decoder to a shared feed forward network (FFN) that predicts either a detection (class and bounding box) or a "`no object`" class.

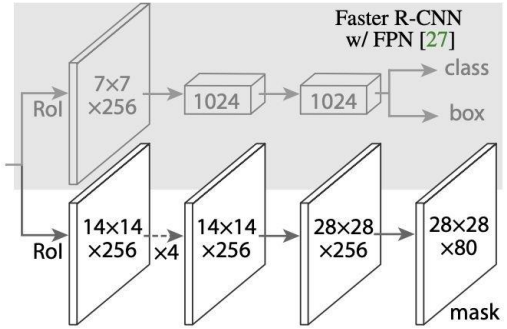
Mask R-CNN: Instance segmentation



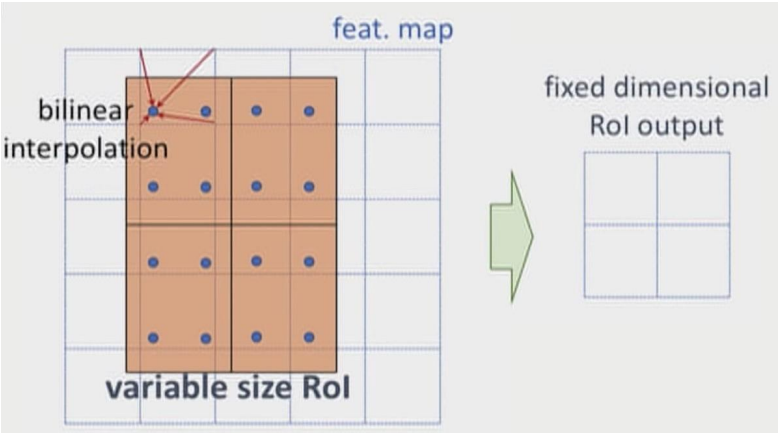
Mask R-CNN

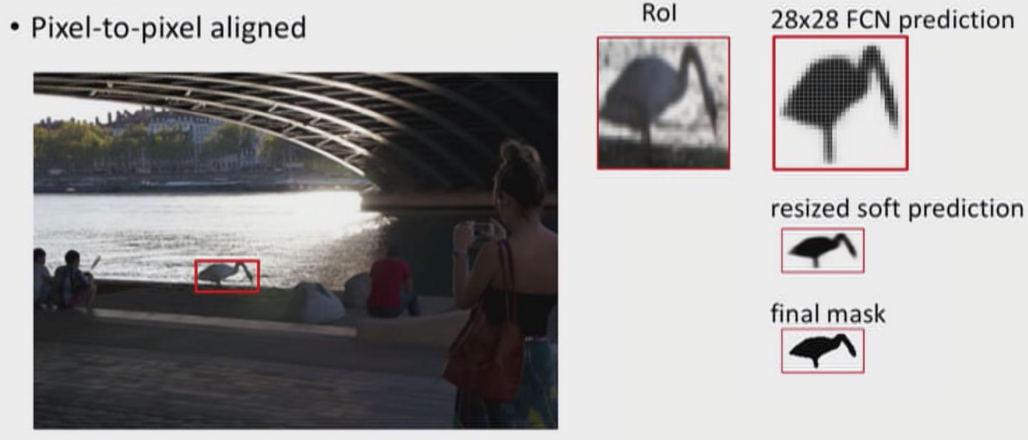

RoIAlign
class
box
conv
conv


Faster R-CNN
w/ ResNet [19]
RoI
7×7
×1024
res5
7×7
×2048
ave
2048
class
box
14×14
×256
14×14
×80
mask


Faster R-CNN
w/ FPN [27]
RoI
7×7
×256
1024
1024
class
box
RoI
14×14
×256
×4
14×14
×256
28×28
×256
28×28
×80
mask

Mask R-CNN: RoI Align





Mask R-CNN: Results