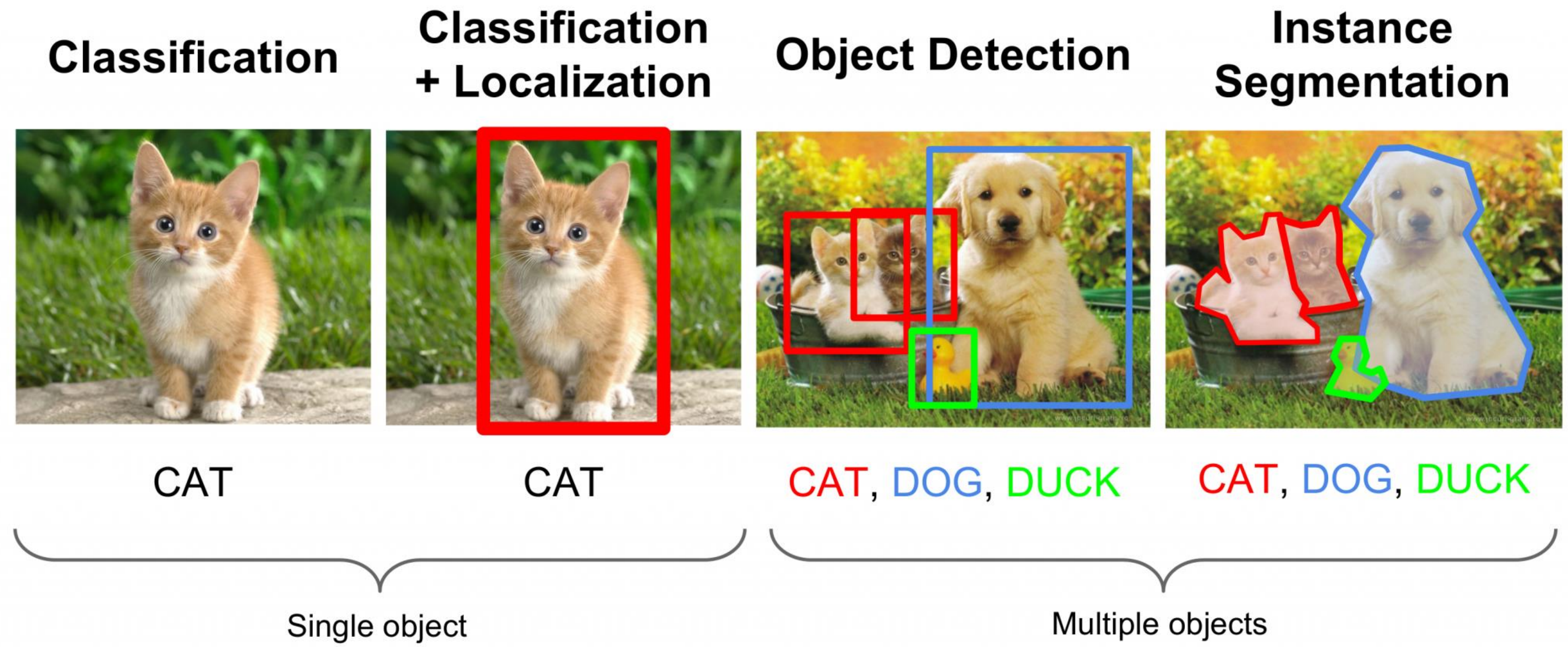
Image Segmentation

Overview and Fully Convolutional Networks

Object localization and detection



Recap: Image Classification

Class

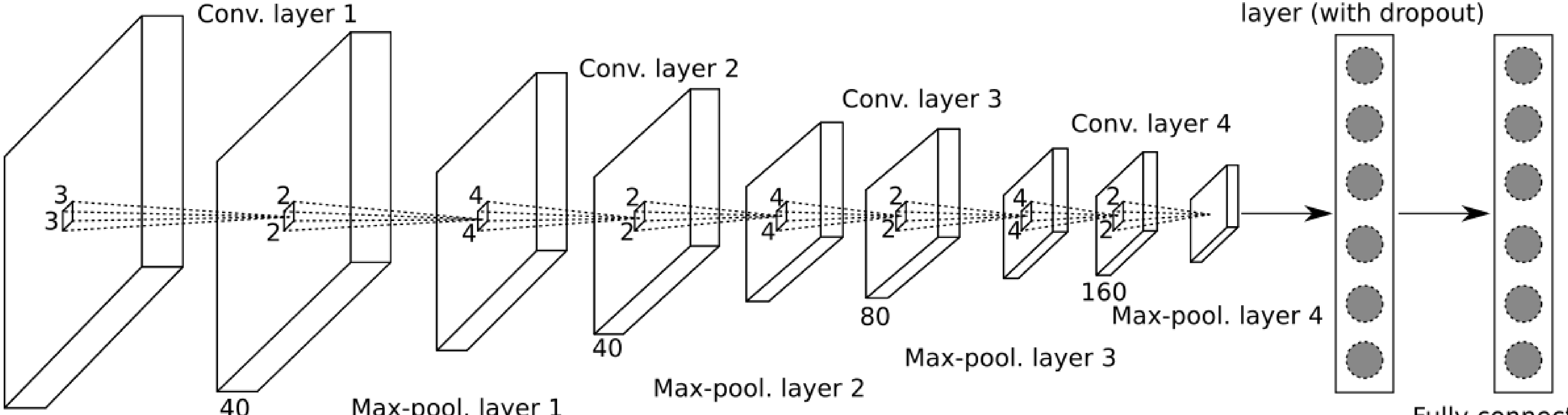


Image segmentation

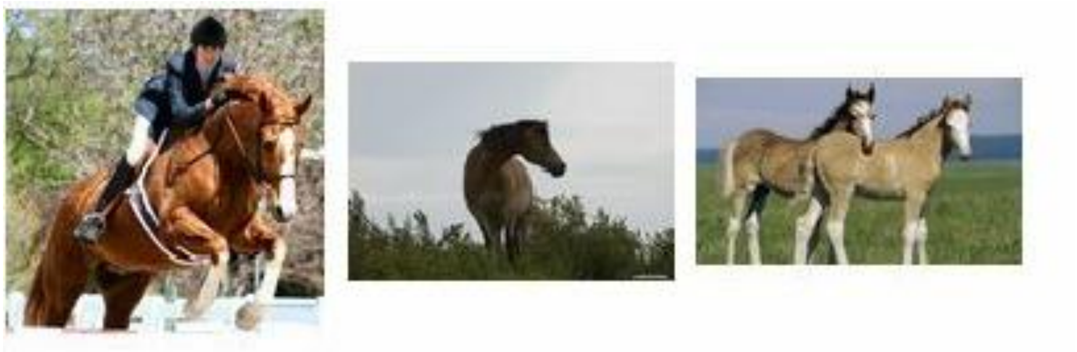
Goal:

- Identify a class for every image point/pixel
- Input: Image of arbitrary dimensions and size
- Output: segmentation mask

Image segmentation: Example

Binary classification

Input



$h \times w \times c$

Output

$h \times w \times 1$

Image segmentation: Example

Classification with several classes (non-mutual as well)

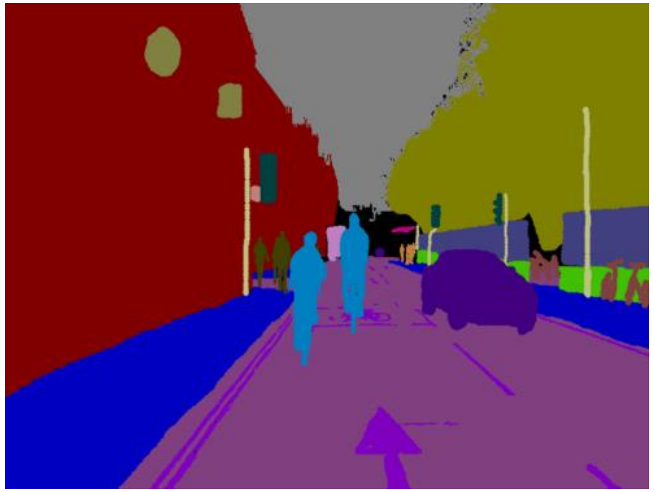
Input	Output (mapped)
	
$h \times w \times c$	$h \times w \times n_{\text{classes}}$

Table of Contents

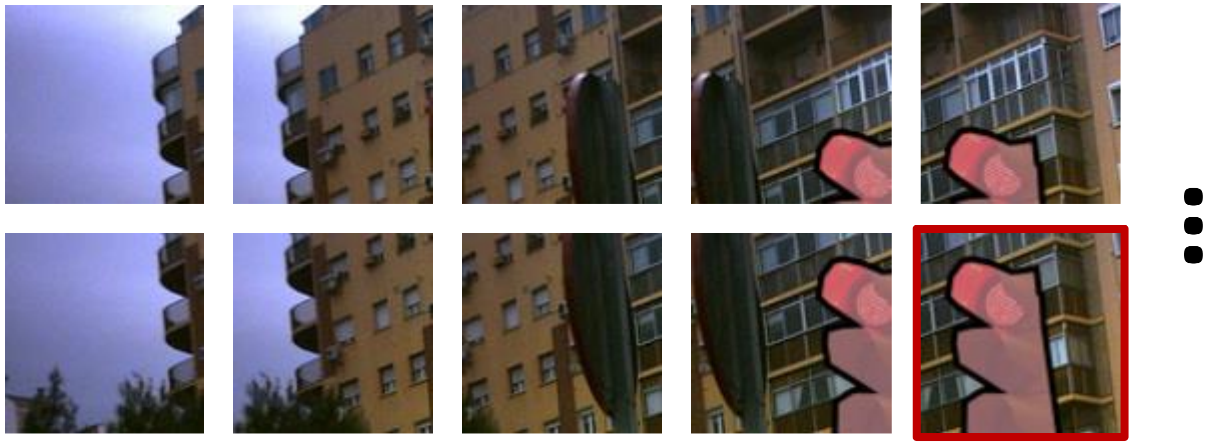
- Sliding Window
- Fully Convolutional Network
  - Up-Sampling variations
  - Training and evaluation
  - Datasets/Transfer-Learning
- Instance Segmentation

Sliding-Window

Principle

- Generate many small image patches (e.g. as a grid)
- Use a CNN to classify each one of them separately
- Center pixel is the target class

Sliding-Window (200x200 with Stride 100)



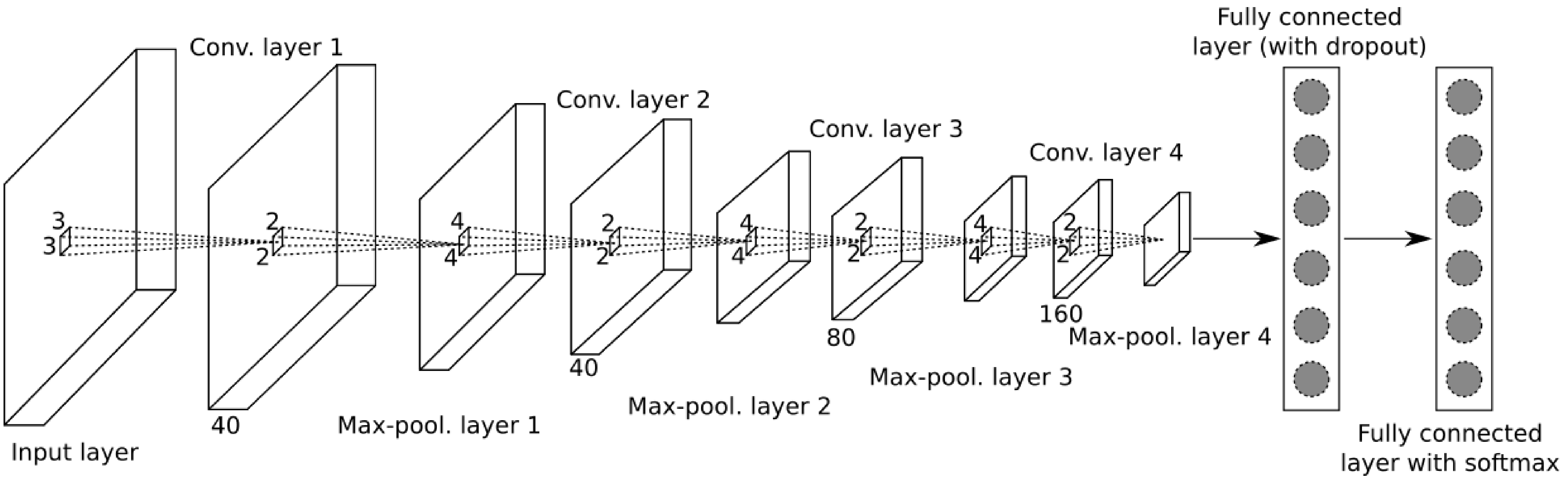
Background

Traffic light

...

Sliding-Window

Traffic light



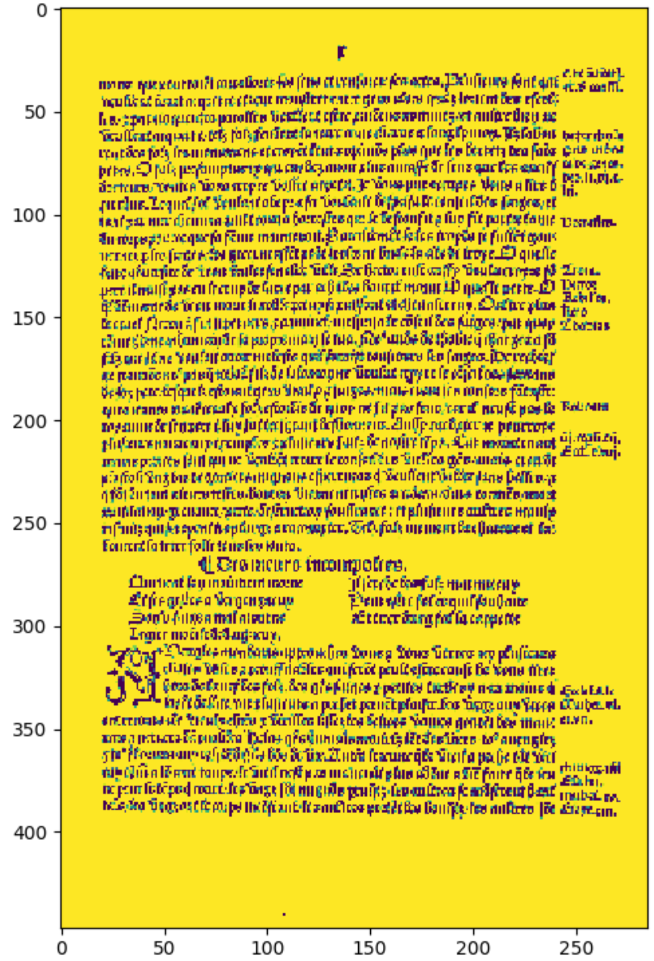
Sliding-Window: Example Page segmentation

Conv, 40, 3x3, padding=same
ReLU
MaxPool, 2x2
Conv, 40, 3x3, padding=same
ReLU
MaxPool, 2x2
Conv, 80, 3x3, padding=same
ReLU
MaxPool, 2x2
Dense(100)
ReLU
Dropout(0.5)
Dense(n_classes)
Softmax

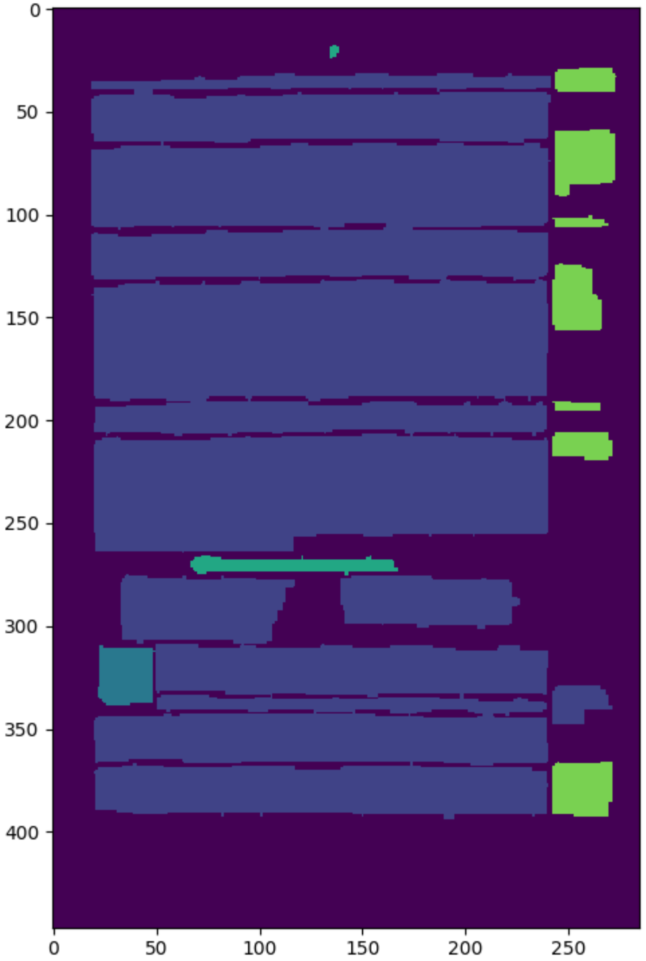
```
model = Sequential([
    Conv2D(40, (3, 3), padding='same', activation='relu')
    MaxPooling2D(pool_size=(2, 2), strides=2),
    Conv2D(40, (4, 4), padding='same', activation='relu'),
    MaxPooling2D(pool_size=(2, 2), strides=2),
    Conv2D(80, (4, 4), padding='same', activation='relu'),
    MaxPooling2D(pool_size=(2, 2), strides=2),
    Flatten(),
    Dense(100, activation='relu'),
    Dropout(dropout),
    Dense(num_classes, activation='softmax', name='softmax'),
])
```

Sliding-Window: Example Page segmentation

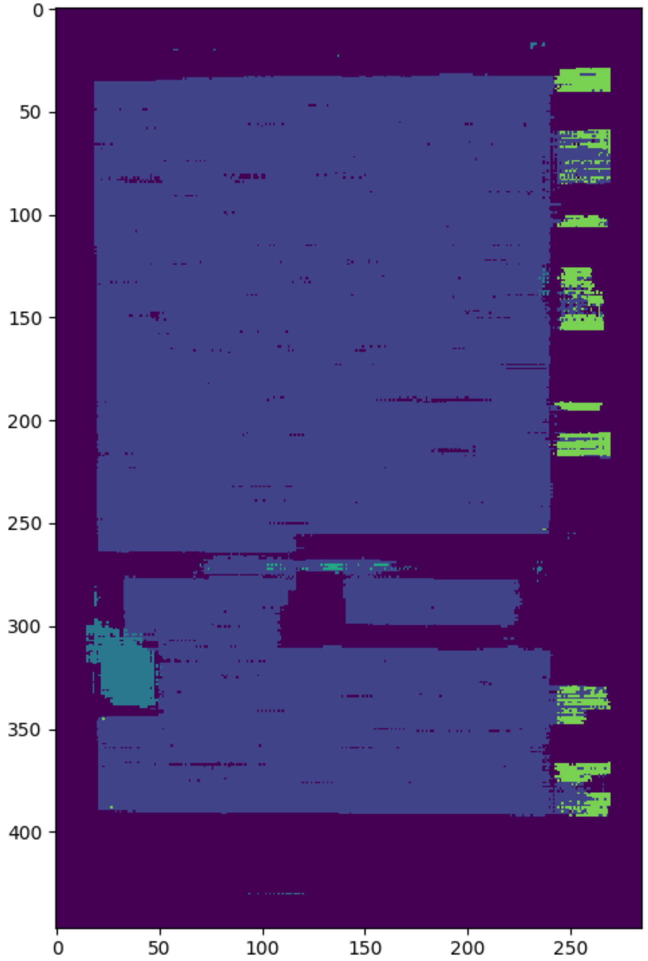
Input (Binary image)



Ground Truth (Label image)



Prediction



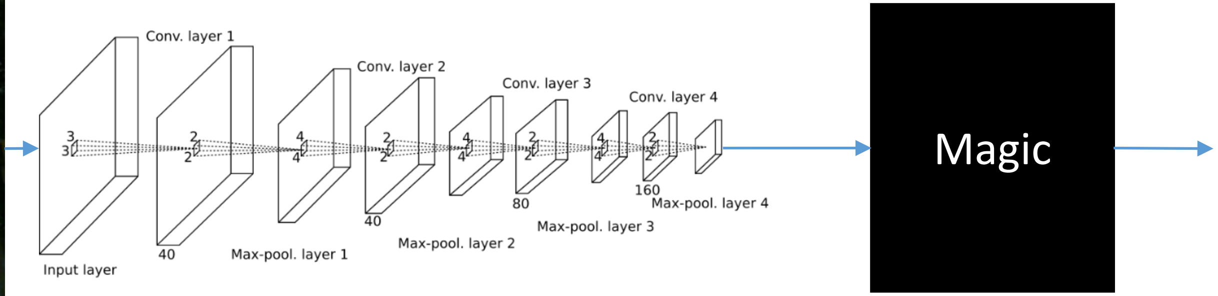
Sliding-Window

Problems

- Small surfaces:
  - Neighboring predictions have no influence on each other
- Resolution:
  - Choosing Sliding window size and stride → Scaling of original image
  - Input and output do not have matching dimensions (only with stride 1 and padding)
- Multiple computations:
  - With small stride many image patches will be sent through the CNN several times
  - Repetition of the same calculations (with the exception of the FC-Layer)

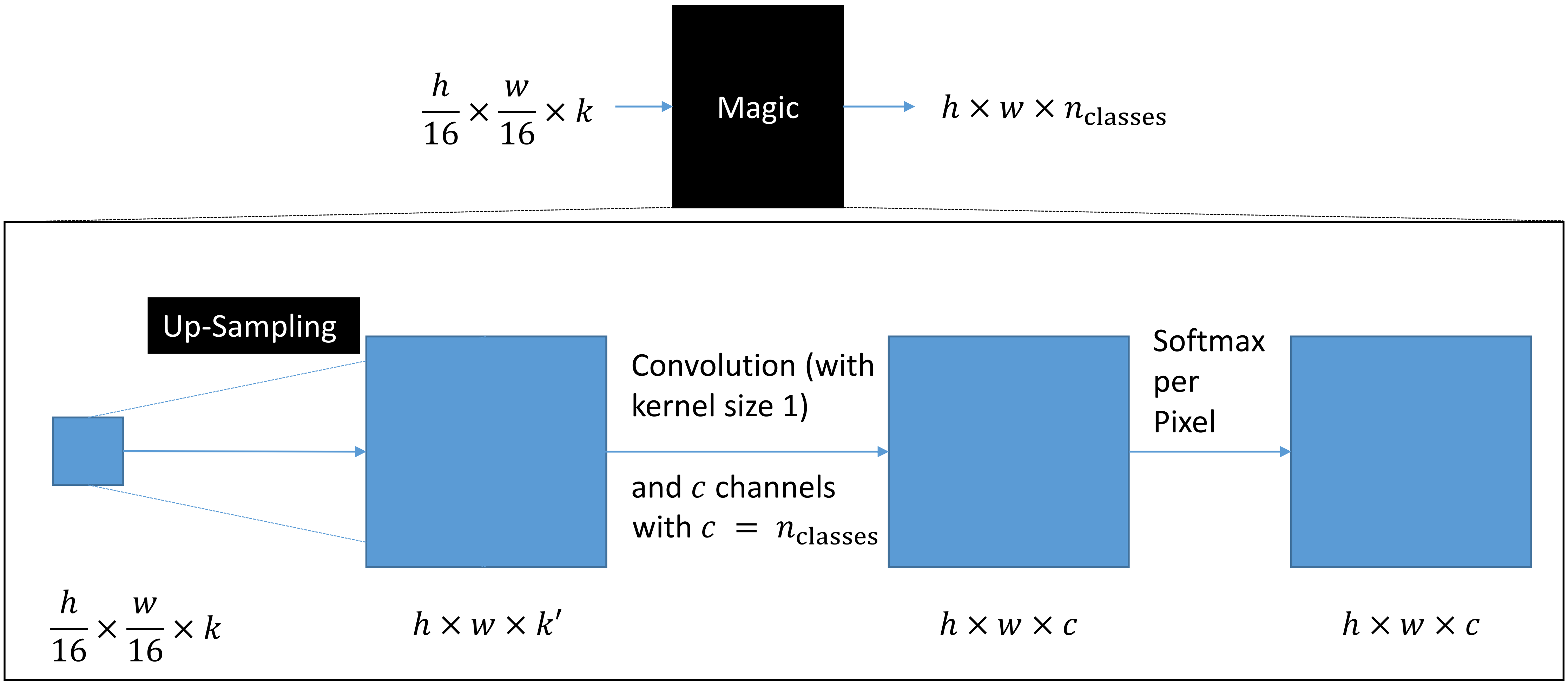
Fully Convolutional Networks

Fully Convolutional Network for segmentation



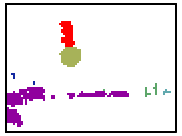
Entire image as input	CNN computes „compressed form“ **Encoder**	Reversing pooling **Decoder**	
$h \times w \times 3$	$\frac{h}{16} \times \frac{w}{16} \times k$		$h \times w \times n_{\text{classes}}$

Fully Convolutional Network for segmentation



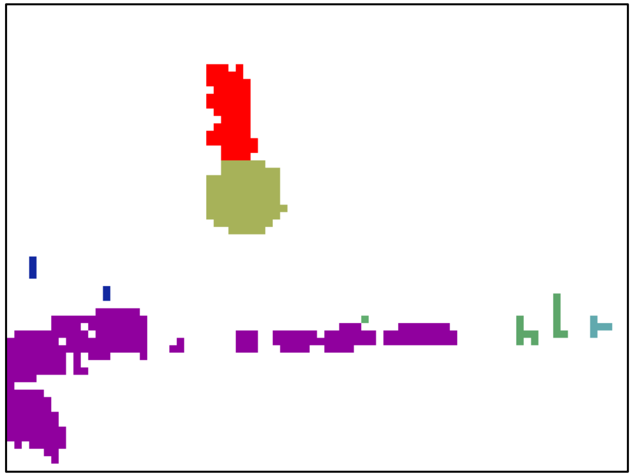
Up-Scaling-Layer

encoded input



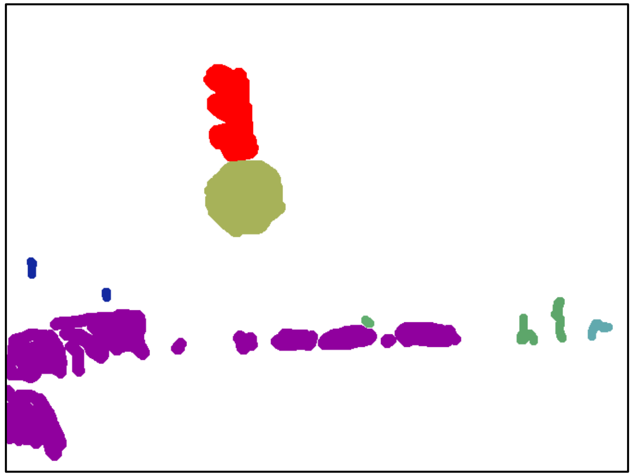
$$\frac{h}{16} \times \frac{w}{16} \times c$$

upscaled



$$h \times w \times c$$

Ground Truth



$$h \times w \times c$$

FCN-Upscaling: Example

Pseudo Code

Conv 40, 3x3, padding=same

ReLU

MaxPool 2x2

Conv 40, 3x3, padding=same

ReLU

MaxPool 2x2

Conv 80, 3x3, padding=same

ReLU

MaxPool 2x2

UpSampling 8x8

Conv2D n_classes, 1x1

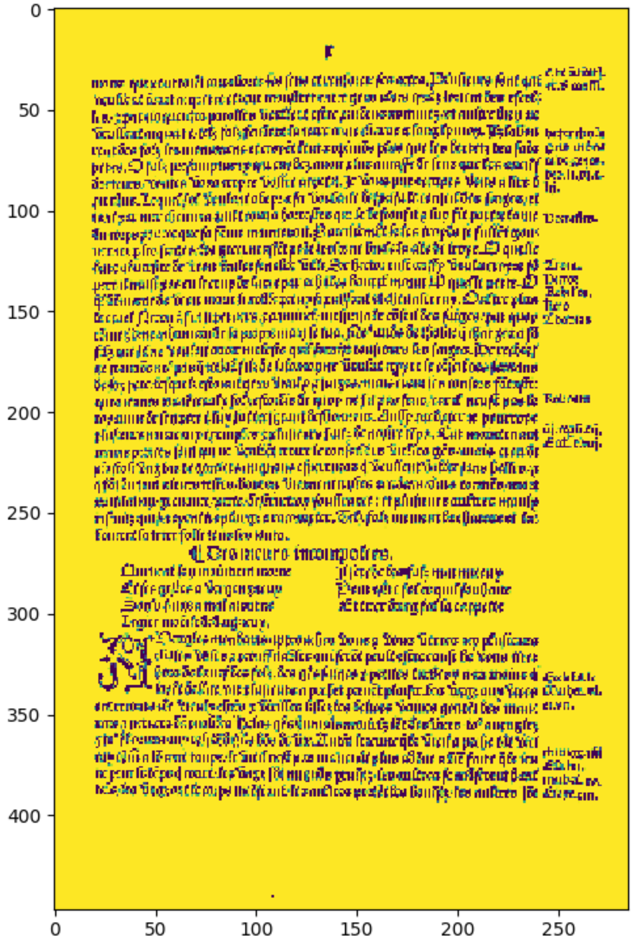
Softmax

Keras

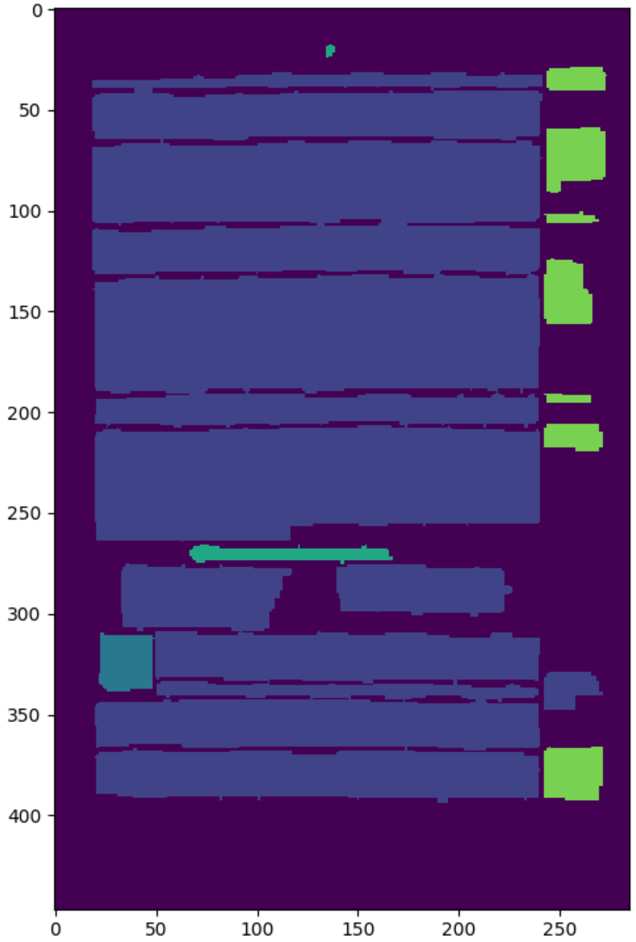
```
model = Sequential([
    Conv2D(40, (3, 3), padding='same', activation='relu',
input_shape=(None, None, 1)),
    MaxPooling2D(pool_size=(2, 2), strides=2),
    Conv2D(40, (4, 4), padding='same', activation='relu'),
    MaxPooling2D(pool_size=(2, 2), strides=2),
    Conv2D(80, (4, 4), padding='same', activation='relu'),
    MaxPooling2D(pool_size=(2, 2), strides=2),
    UpSampling2D((8, 8), interpolation='nearest'),
    Conv2D(num_classes, (1, 1)),
    Activation('softmax'),
])
```

FCN-Upscaling: Example

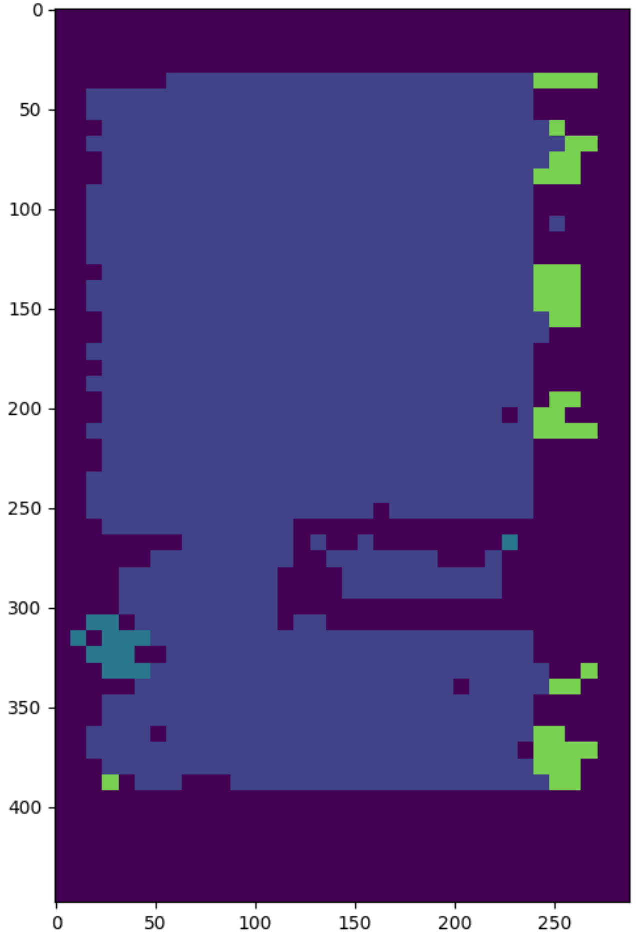
Input (binary image)



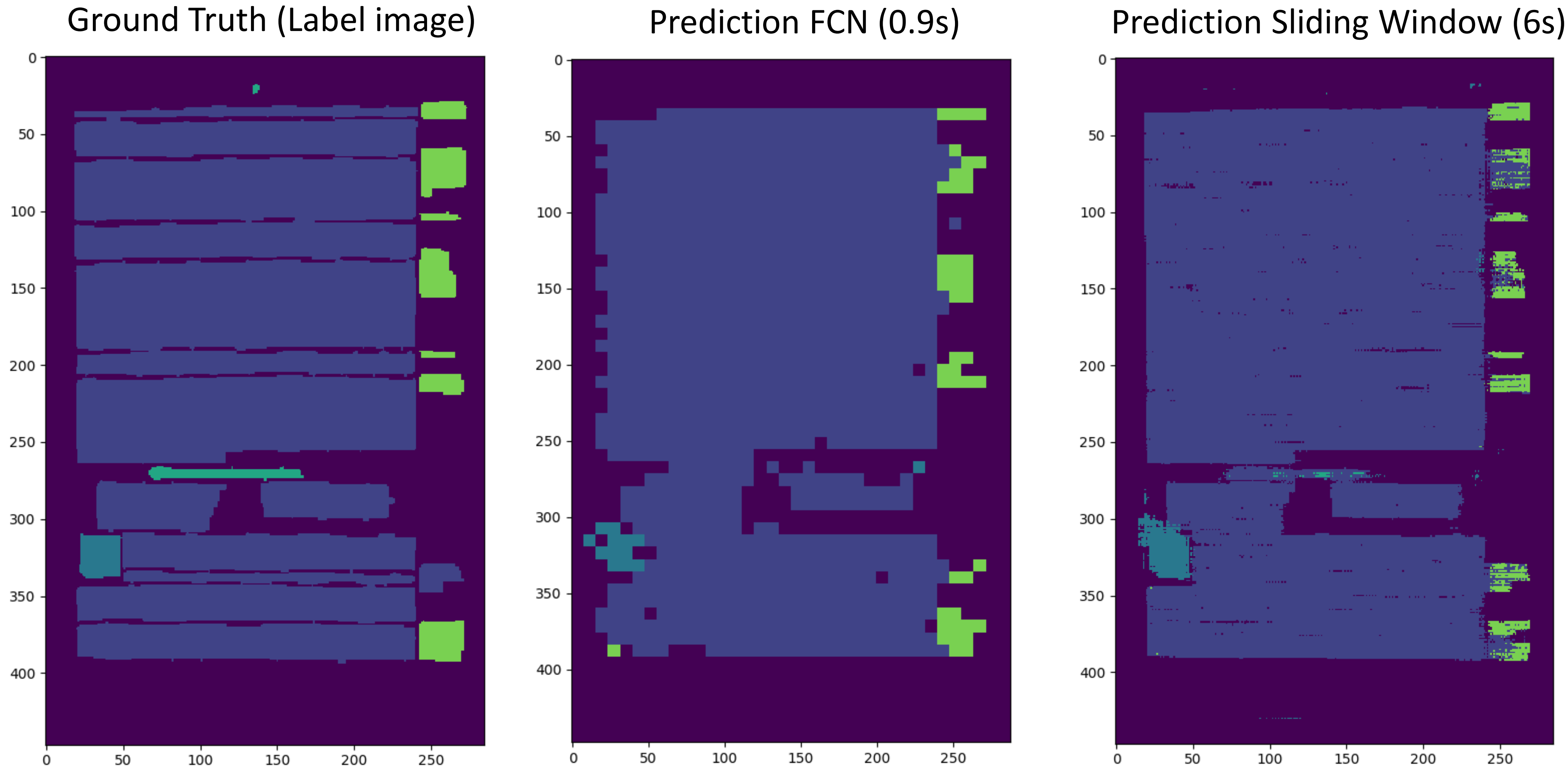
Ground Truth (Label image)



Prediction



FCN-Upscaling: Example



Up-Scaling-Layer

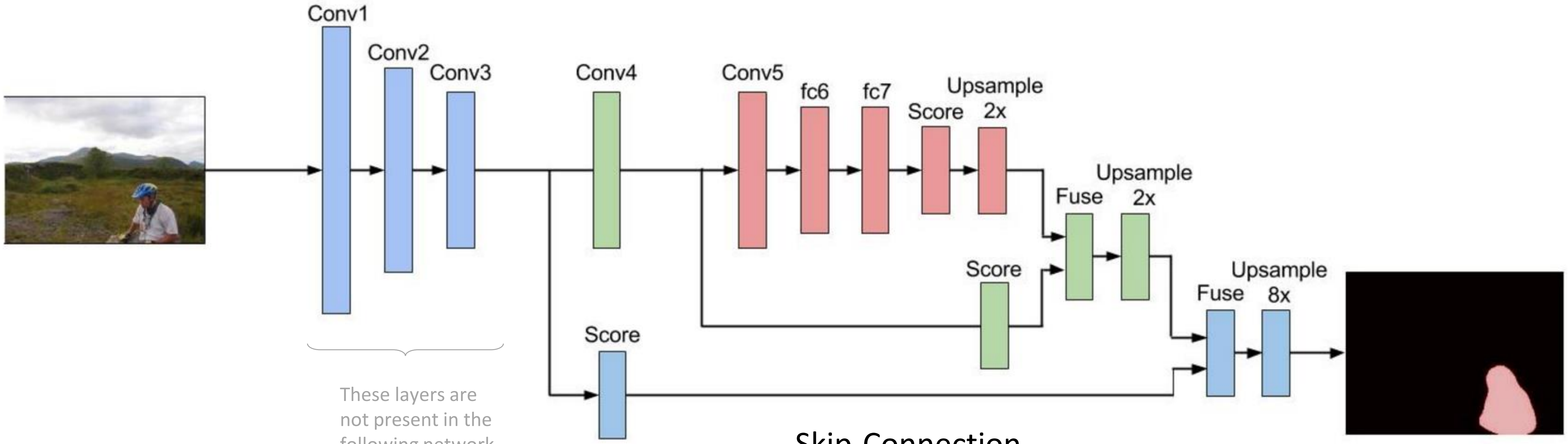
Pros

- Easy to implement (error is summed in the backward pass)
- No parametera ⇒ extremely fast

Cons

- Small objects absorbed
- Only very rough regions

FCN: Skip-Connections



These layers are not present in the following network

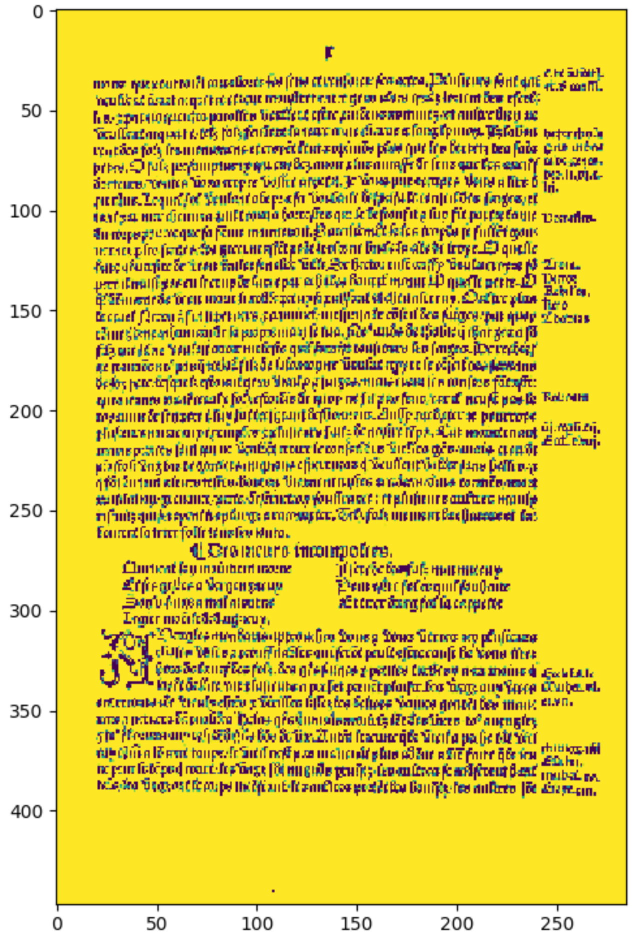
Skip-Connection

Keras: Skip-Connections

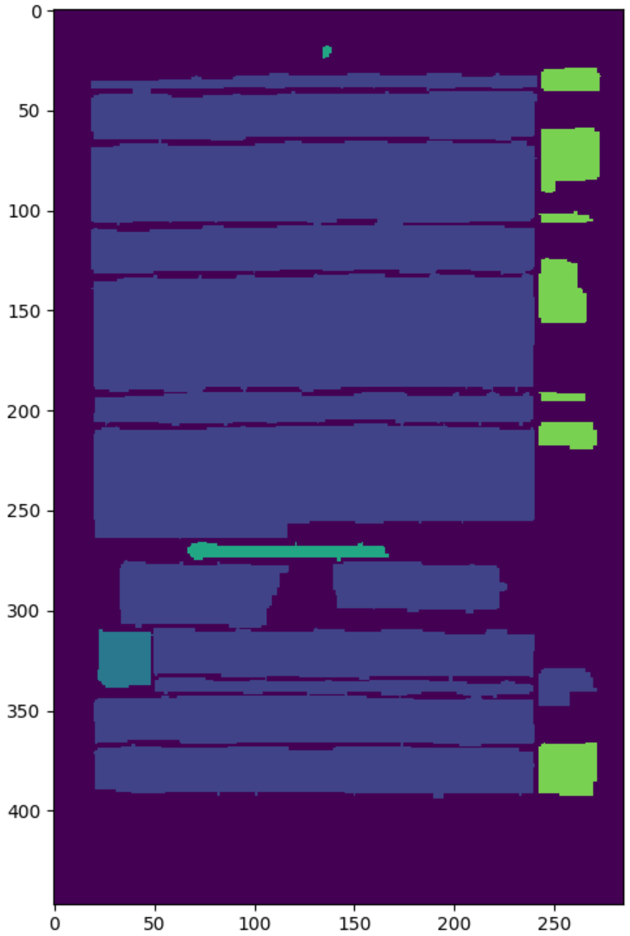
```
input = Input((None, None, 1))

conv1 = Conv2D(40, (3, 3), padding='same',
activation='relu')(input)

pool1 = MaxPooling2D(pool_size=(2, 2),
strides=2)(conv1)

conv2 = Conv2D(80, (4, 4), padding='same',
activation='relu')(pool1)

pool2 = MaxPooling2D(pool_size=(2, 2),
strides=2)(conv2)

conv3 = Conv2D(80, (4, 4), padding='same',
activation='relu')(pool2)

pool3 = MaxPooling2D(pool_size=(2, 2),
strides=2)(conv3)

fc4 = Conv2D(160, (1, 1), padding='same',
activation='relu')(pool3)

fc5 = Conv2D(160, (1, 1), padding='same',
activation='relu')(fc4)

up1 = UpSampling2D((2, 2), fc5)

fuse2 = Concatenate(axis=-1)([up1, conv3])

up2 = UpSampling2D((2, 2), fuse2)

fuse3 = Concatenate(axis=-1)([up2, conv2])

up3 = UpSampling2D((2, 2), fuse3)

fuse4 = Concatenate(axis=-1)([up3, conv1])

logits = Conv2D(num_classes, (1, 1))(fuse4)

softmax = Activation('softmax')(logits)


model = Model(inputs=input, outputs=softmax)
```

FCN-Upscaling-Skip-Connections: Example

Input (Binary Image)



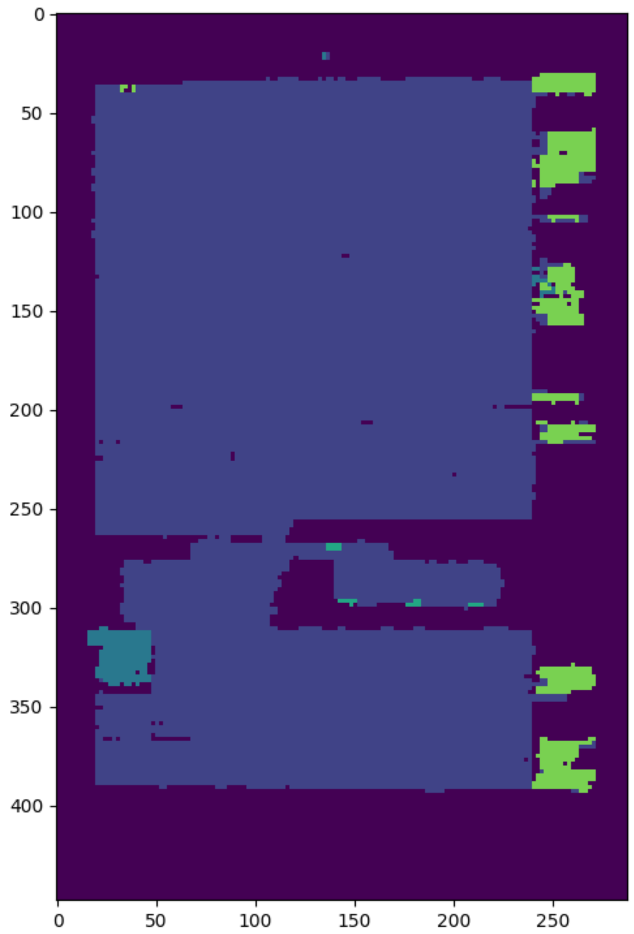
Ground Truth (Label image)



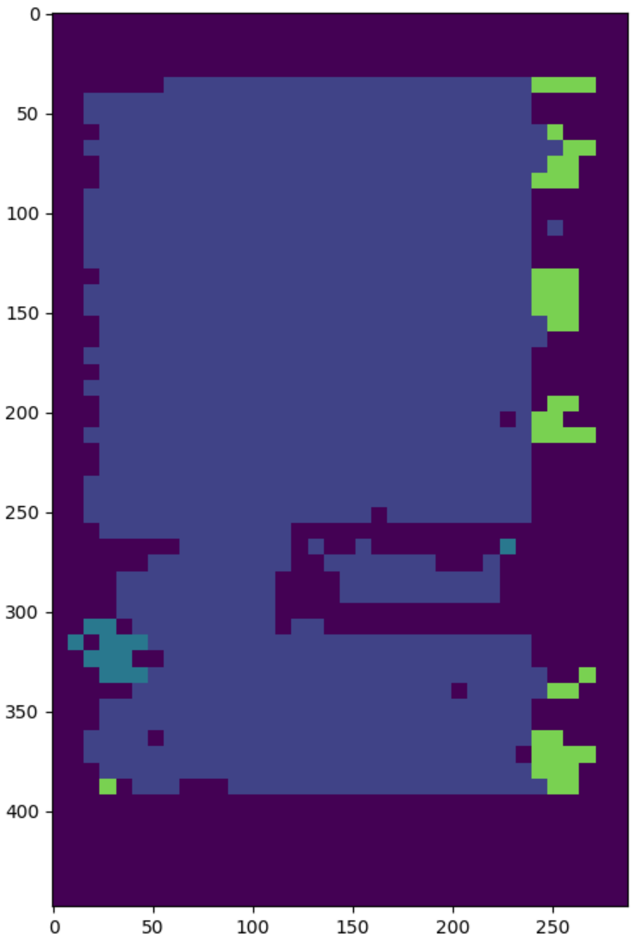
Prediction

FCN-Upscaling-Skip-Connections: Example

Prediction Skip-FCN (0.91s)



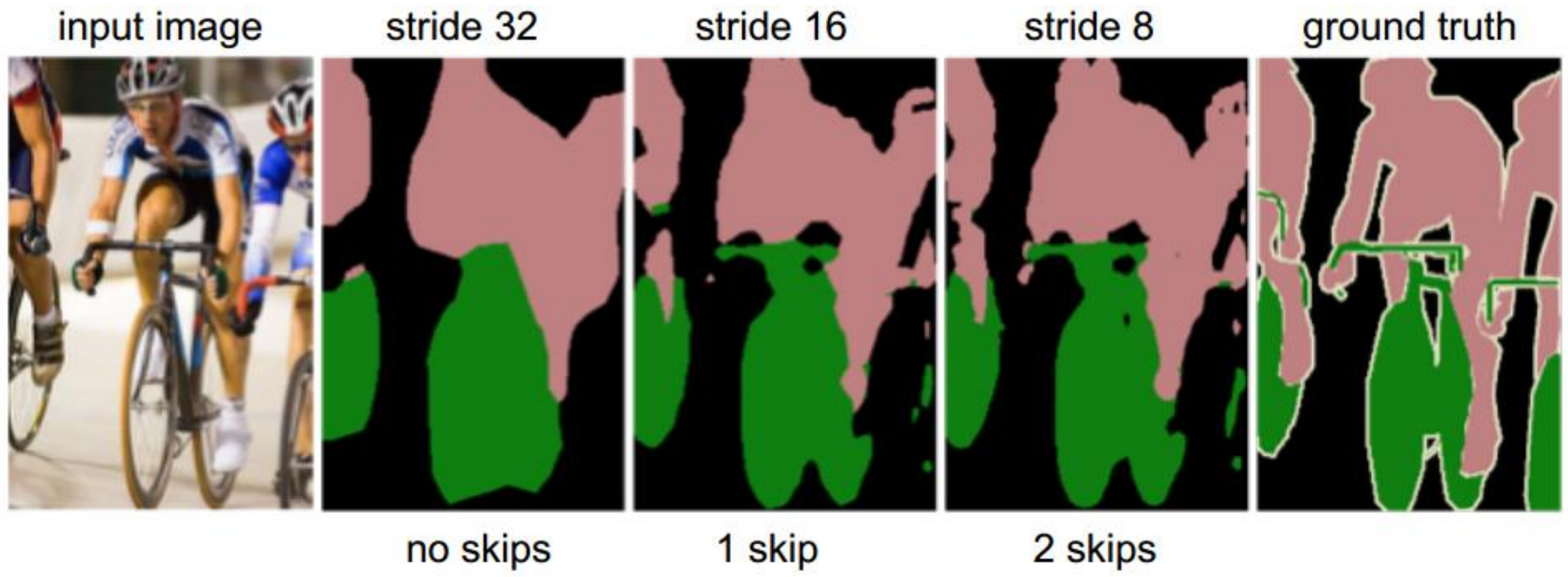
Prediction FCN (0.86s)



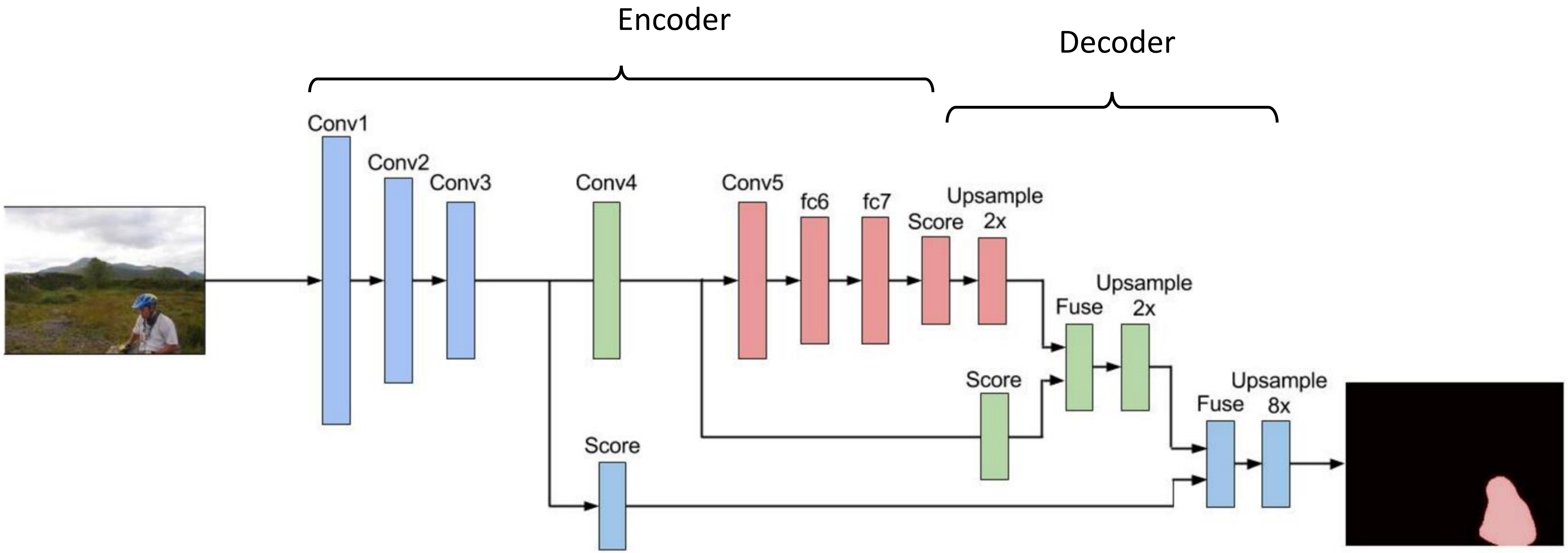
Prediction Sliding Window (6s)

FCN-Upscaling-Skip-Connections:

- Skip connections allow resolving finer structures
- The complete Encoder-Decoder generates rough structures



FCN: Encoder/Decoder naming



Insertion: FCN for compression

Input image with high resolution with full information (e.g. bmp)

Encoder algorithm

JPG

Decoder algorithmus

Decoded image as close as possible to the original

Compressed representation of the image

NN-Encoder

State

NN-Decoder

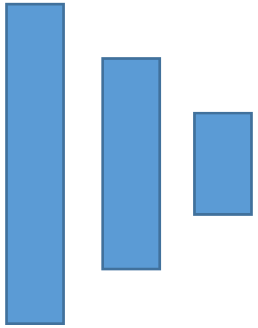
Insertion: FCN for compression

NN-Encoder

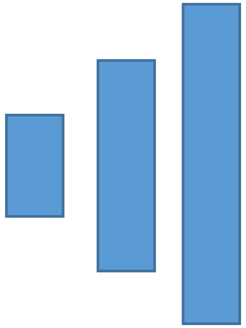
State

NN-Decoder

Conv/Pool
architecture



Conv/Up-Sampling
architecture



Up-Sampling Alternatives

Transposed Convolution, Unpooling

Up-Sampling: Methods

- Up-Scaling (no parameters)
- Transposed convolution (sometimes also deconvolution)
- Un-Pooling („Inverse pooling“)

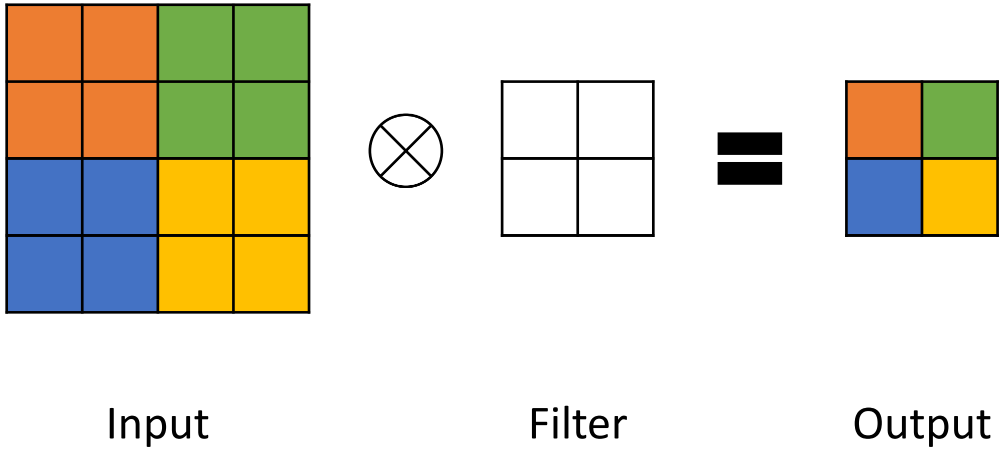
Transposed Convolution

Example U-Net

Convolution

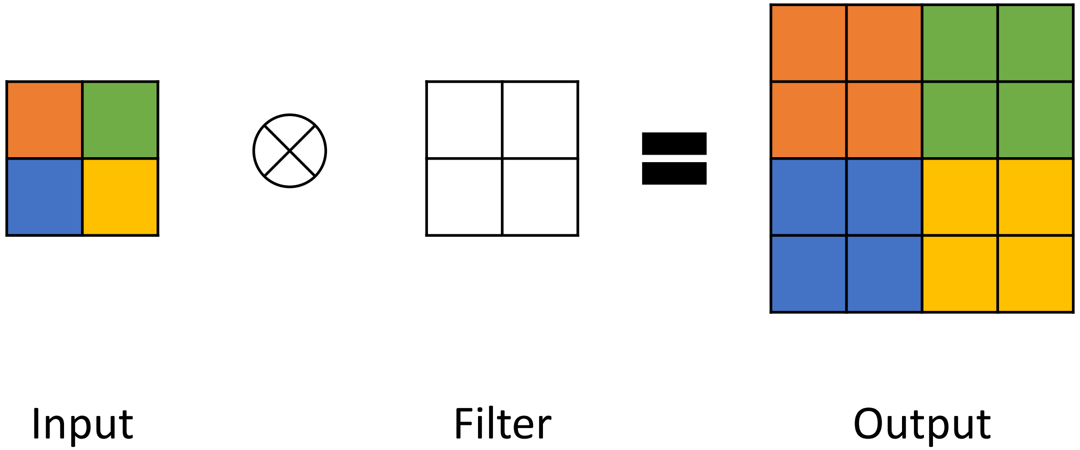
Example: Convolution

- Kernel-Size $2 \times 2 \times k$
- Stride 2



Transposed Convolution

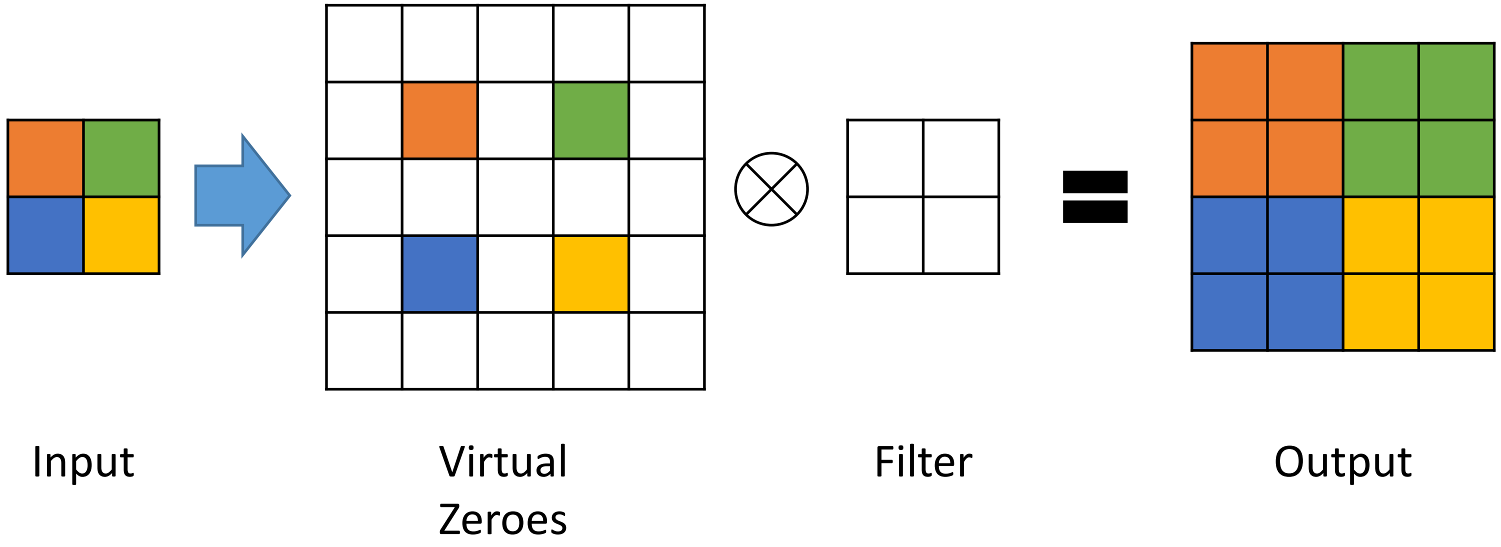
Example: Transposed Convolution
- Kernel-Size $2 \times 2 \times k$
- Stride 2



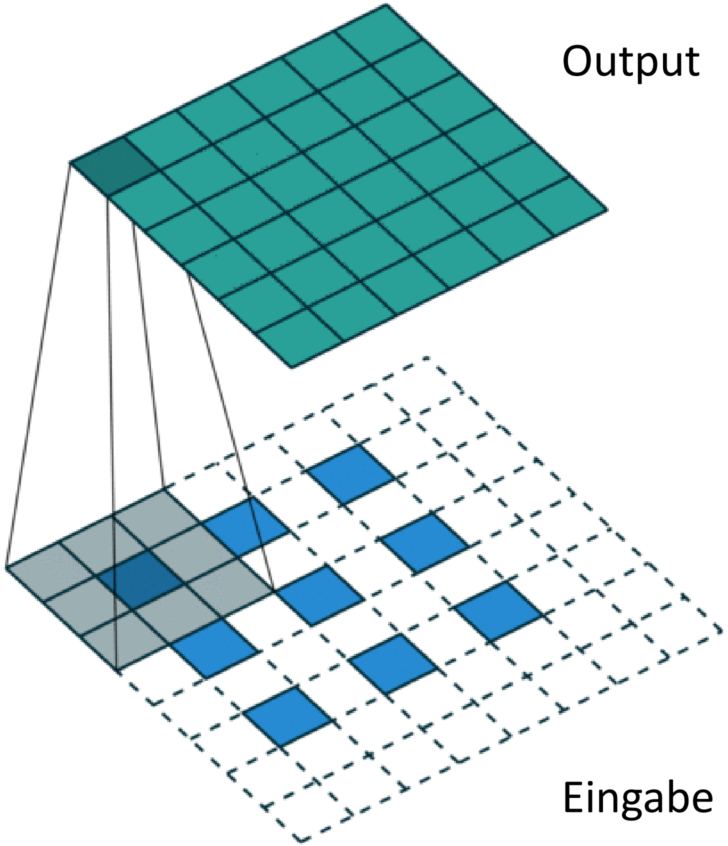
Transposed Convolution

Example: Transposed Convolution

- Kernel-Size $2 \times 2 \times k$
- Stride 2



Transposed Convolution



- Transposed convolution here with kernel size 3x3, stride 2 and padding generates an output image with doubled dimensions of the input image
- Animation should seem familiar

Comparing Convolutions

Convolution

- Padding
  - Full padding
  - No Padding
  - Half Padding
- Stride s: Resolution decreased by factor s
- Conv in Forward-Pass becomes zu transposed Conv in Backward-Pass

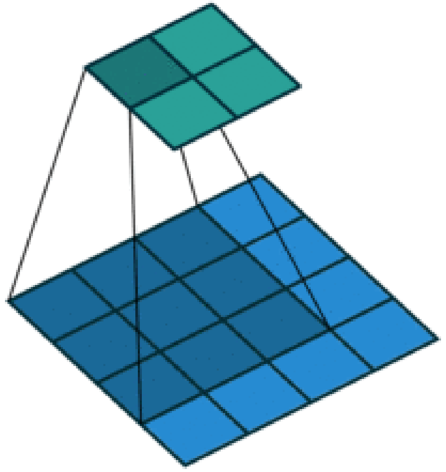
Transposed Convolution

- Padding
  - No Padding
  - Full Padding
  - Half Padding
- Stride s: Resolution increased by factor s
- Transposed Conv in Forward-Pass becomes Conv in Backward-Pass

Tranposed Convolution Examples: Standard

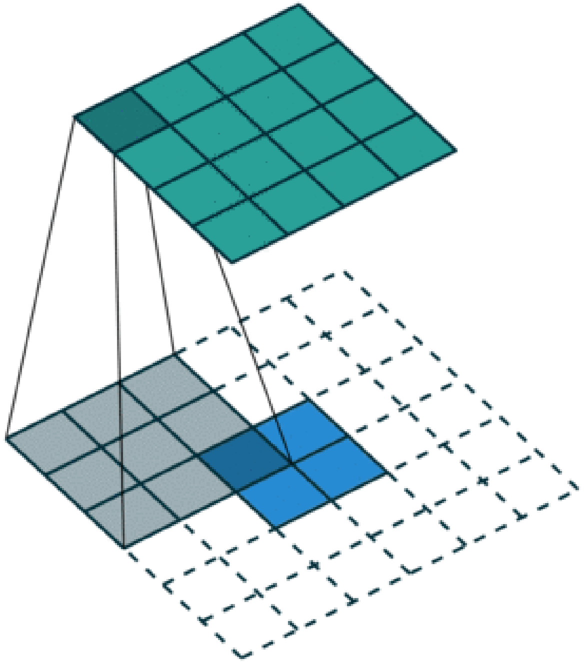
Normal Convolution

- 3x3 Filter, Stride 1
- 4x4 Input
- 2x2 Output



Transposed Convolution

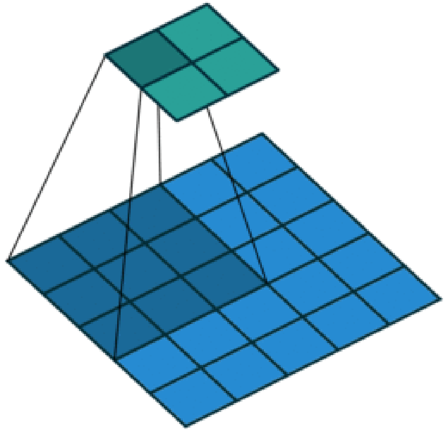
- 3x3 Filter, Stride 1
- 2x2 Input
- 4x4 Output



Tranposed Convolution Examples: Stride

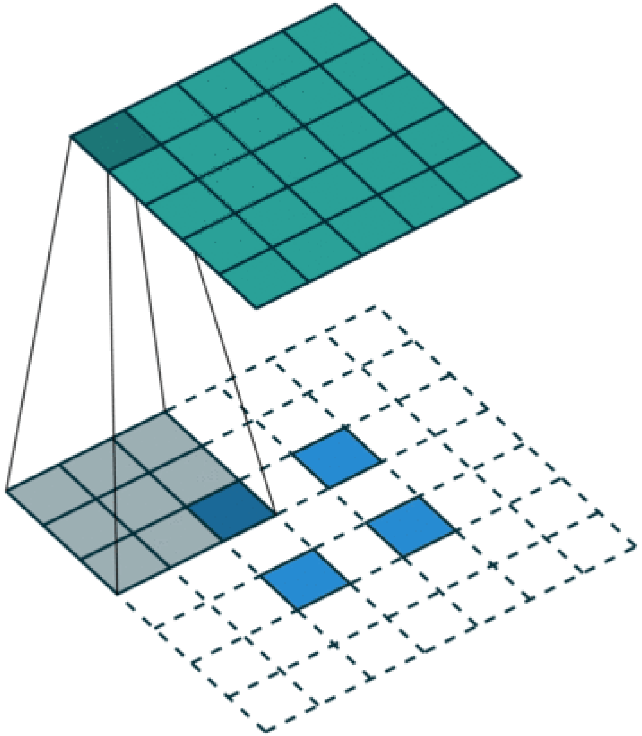
Normal Convolution

- 3x3 Filter, Stride 2
- 5x5 Input
- 2x2 Output



Transposed Convolution

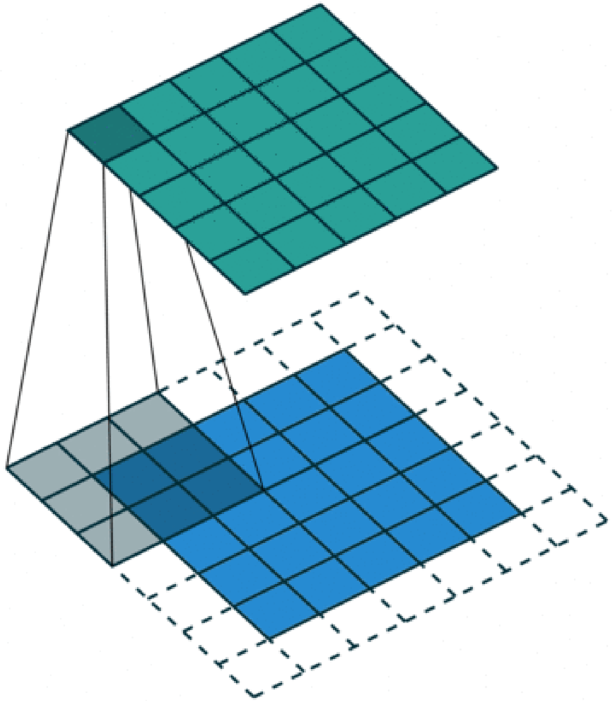
- 3x3 Filter, Stride 2
- 2x2 Input
- 5x5 Output



Tranposed Convolution Examples: Padding

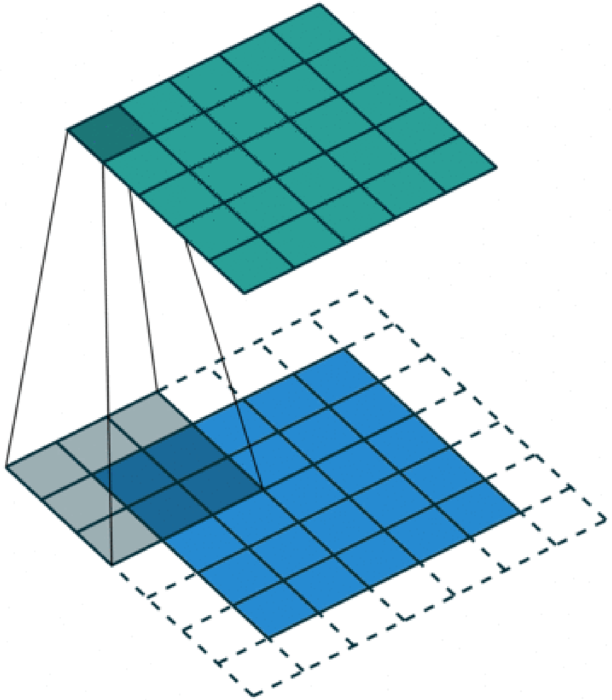
Normal Convolution

- 3x3 Filter, Stride 1
- 5x5 Input
- 5x5 Output



Transposed Convolution

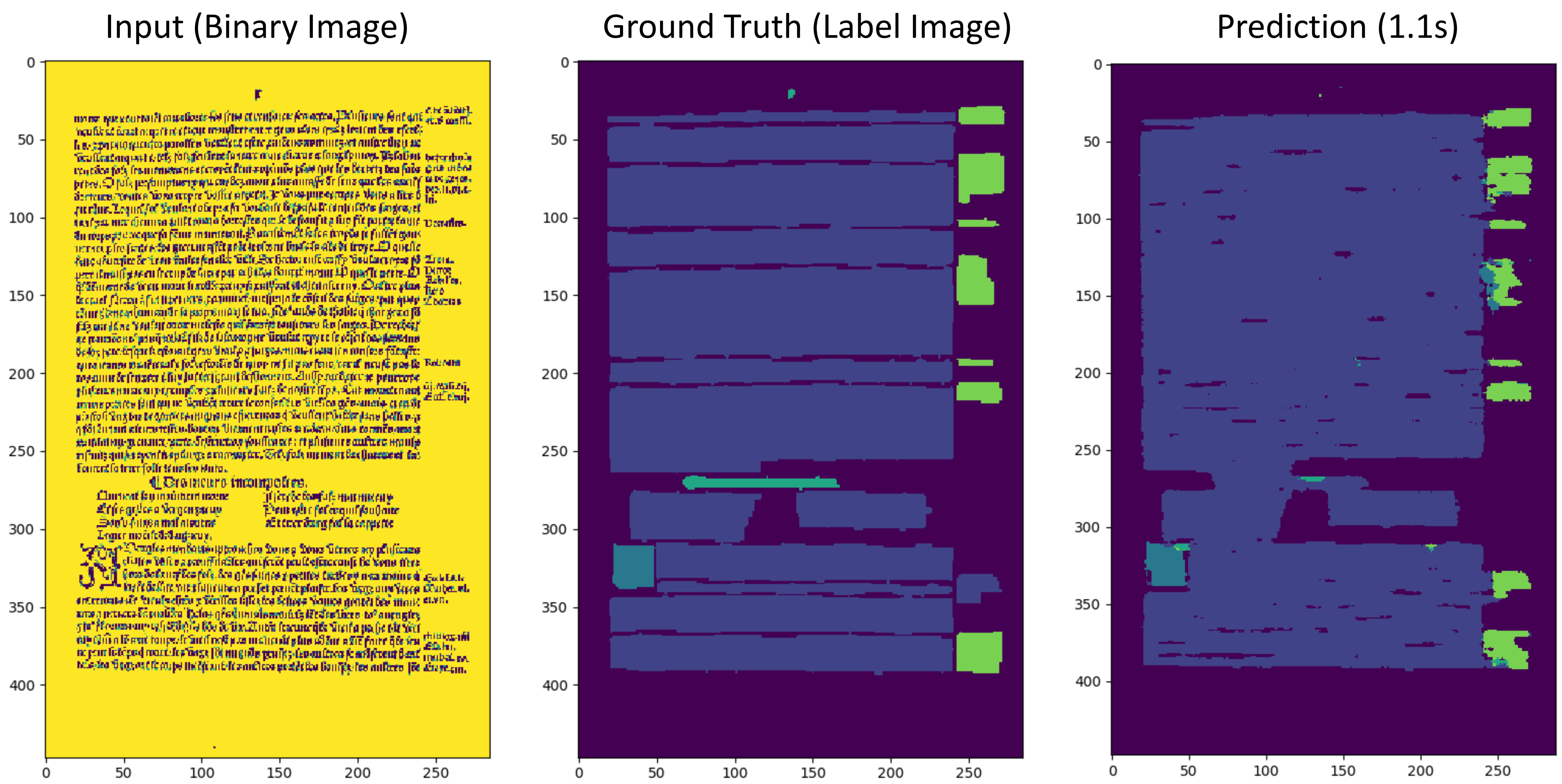
- 3x3 Filter, Stride 1
- 5x5 Input
- 5x5 Output



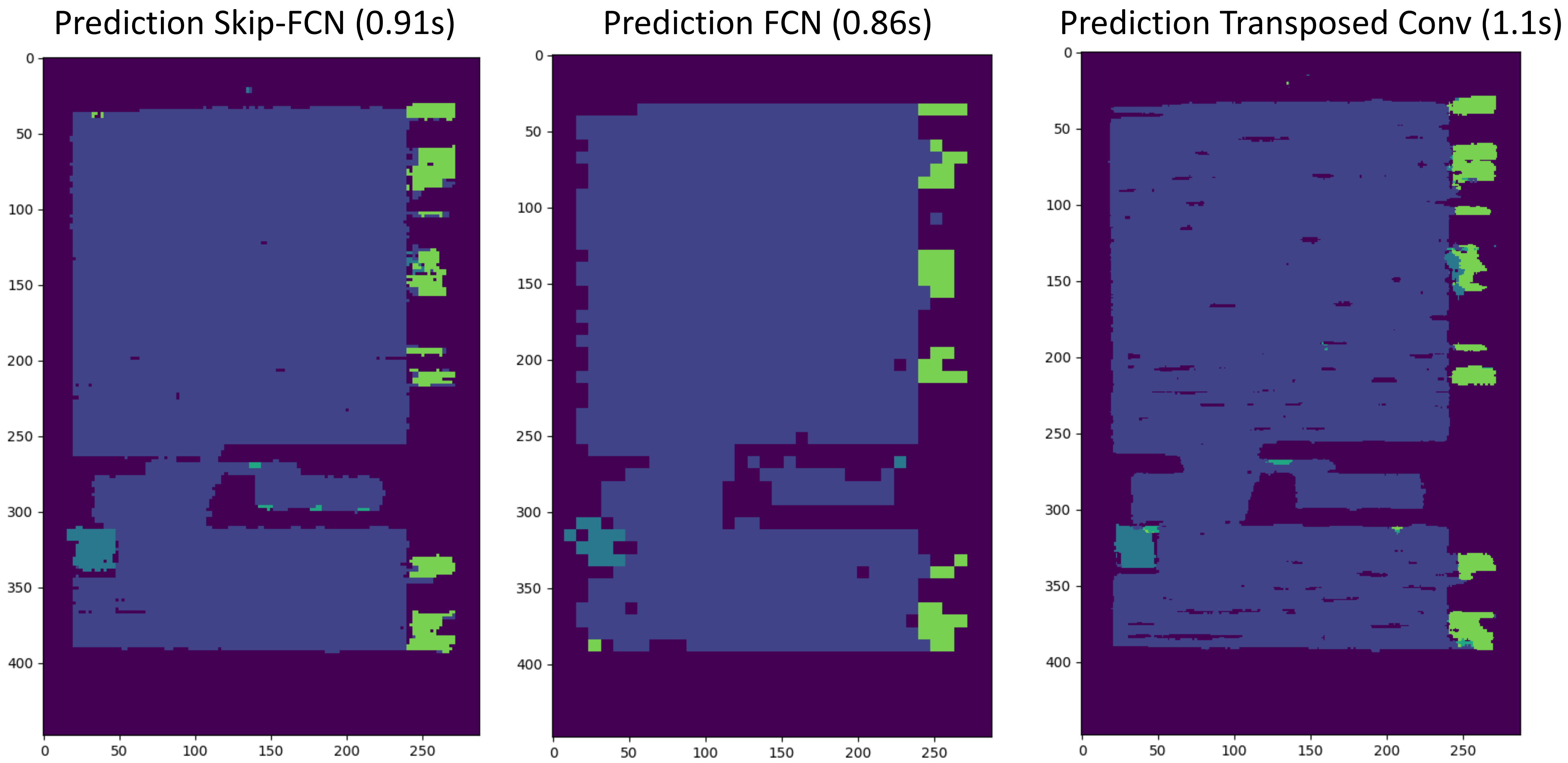
FCN-Conv^T-Skip-Connections: Architecture

```
input = Input((None, None, 1))

conv1 = Conv2D(40, (3, 3), padding='same',
activation='relu')(input)

pool1 = MaxPooling2D(pool_size=(2, 2),
strides=2)(conv1)

conv2 = Conv2D(80, (4, 4), padding='same',
activation='relu')(pool1)

pool2 = MaxPooling2D(pool_size=(2, 2),
strides=2)(conv2)

conv3 = Conv2D(80, (4, 4), padding='same',
activation='relu')(pool2)

pool3 = MaxPooling2D(pool_size=(2, 2),
strides=2)(conv3)

fc4 = Conv2D(160, (1, 1), padding='same',
activation='relu')(pool3)

fc5 = Conv2D(160, (1, 1), padding='same',
activation='relu')(fc4)

up1 = Conv2DTransposed(80, (3, 3), 2,
padding='same')(fc5)

fuse2 = Concatenate(axis=-1)([up1, conv3])

up2 = Conv2DTransposed(80, (3, 3), 2,
padding='same')(fuse2)

fuse3 = Concatenate(axis=-1)([up2, conv2])

up3 = Conv2DTransposed(40, (3, 3), 2,
padding='same')(fuse3)

fuse4 = Concatenate(axis=-1)([up3, conv1])

logits = Conv2D(num_classes, (1, 1))(fuse4)

softmax = Activation('softmax')(logits)


model = Model(inputs=input, outputs=softmax)
```

FCN-Conv^T-Skip-Connections: Example



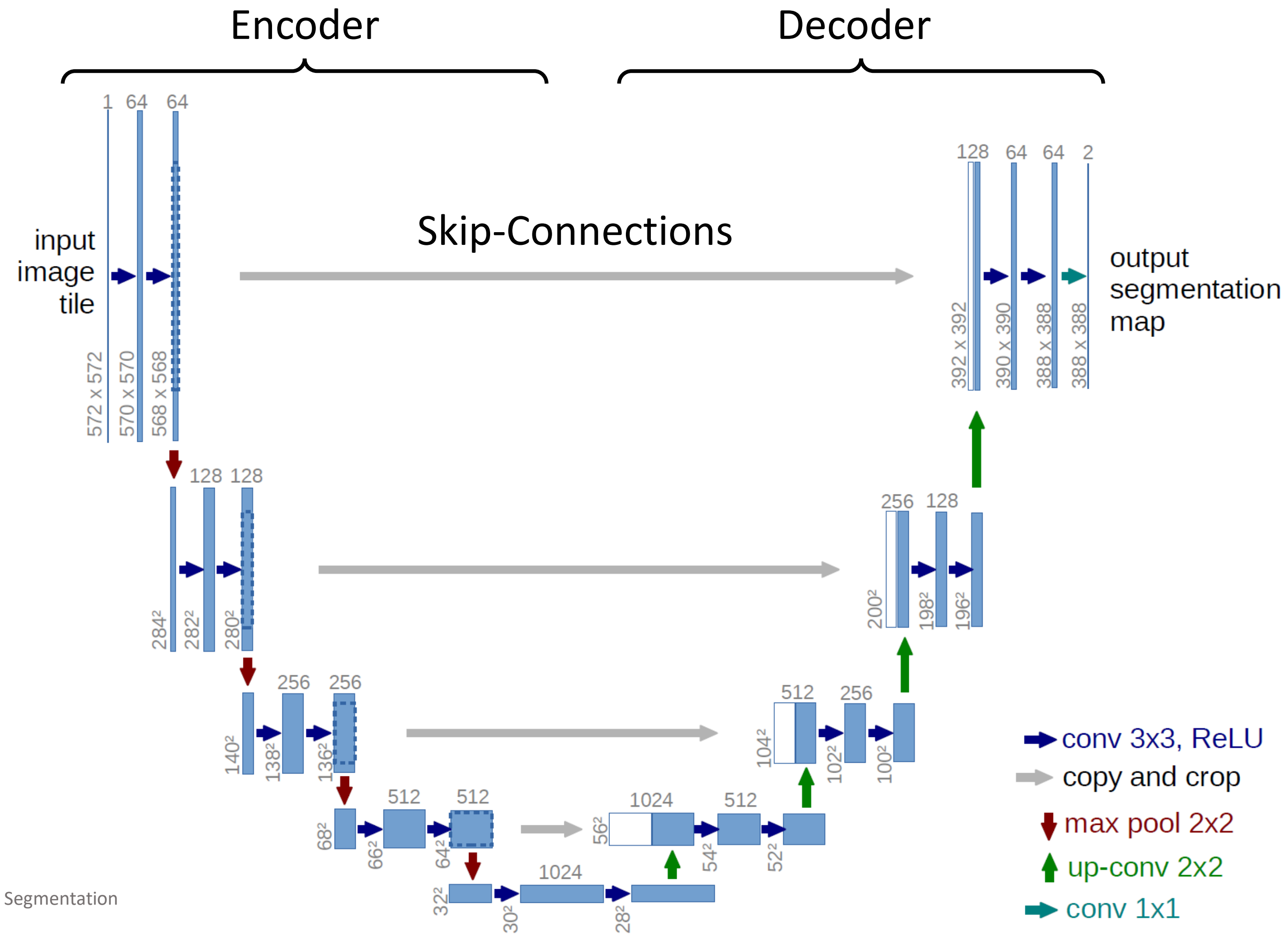
FCN-Conv^T-Skip-Connections: Example



U-Net

- So far: No additional conv layers after up-sampling (except for output layer)
- In reality (actual networks): Decoder consists of several more conv layers, analogous to the encoder
- Popular network: U-Net

U-Net



U-Net: Convolutional Networks for Biomedical Image Segmentation
Olaf Ronneberger, Philipp Fischer, Thomas Brox

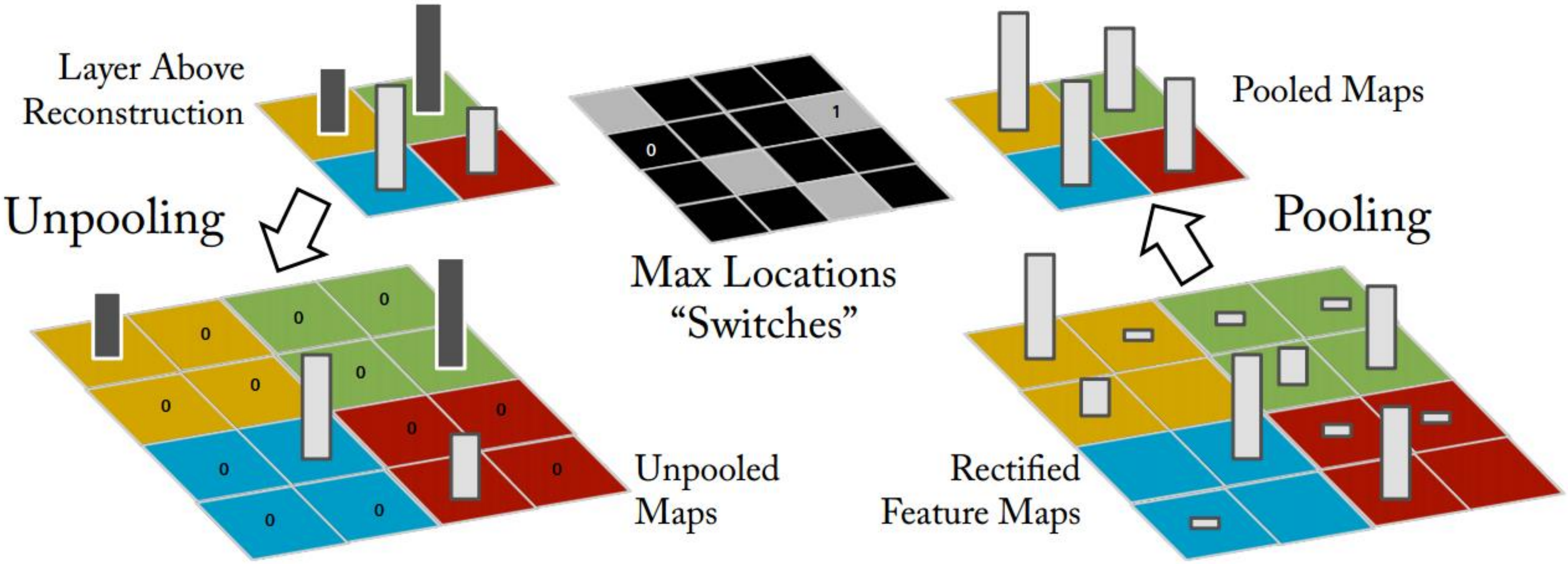
Up-Sampling Alternatives

Unpooling

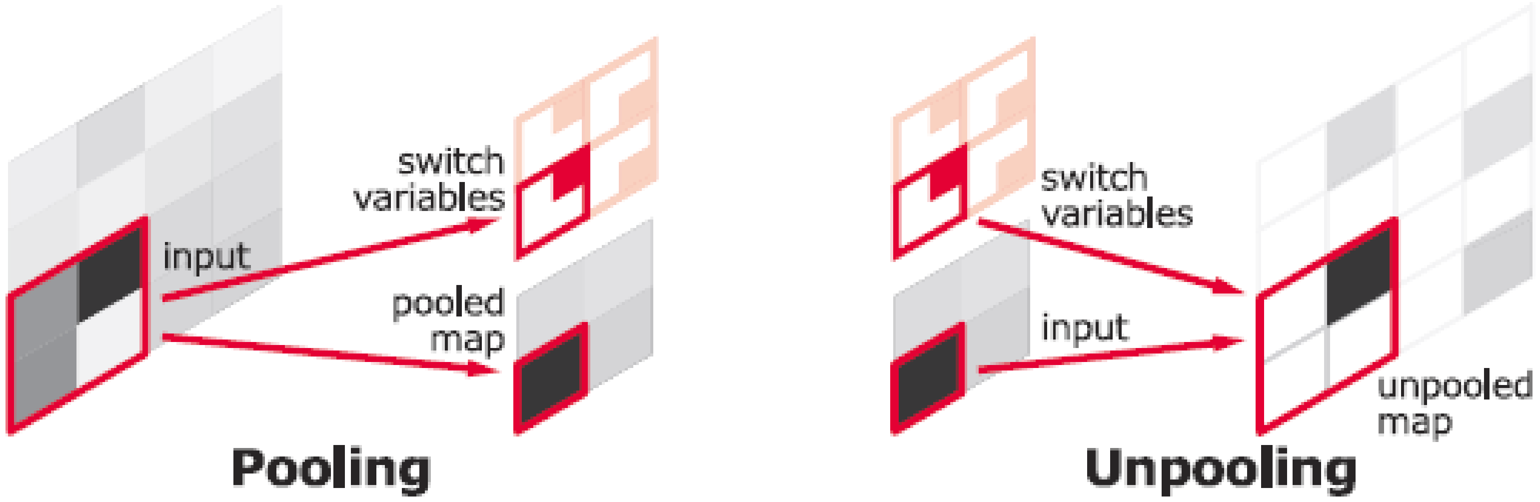
Unpooling

- Unpooling as alternative to Transposed Convolution layers
- A pooling layer is required for every unpooling layer
- Max pooling is non reversible, but analogous to backward pass:
  - Save, where the weight came from
  - Forward the input value to this position

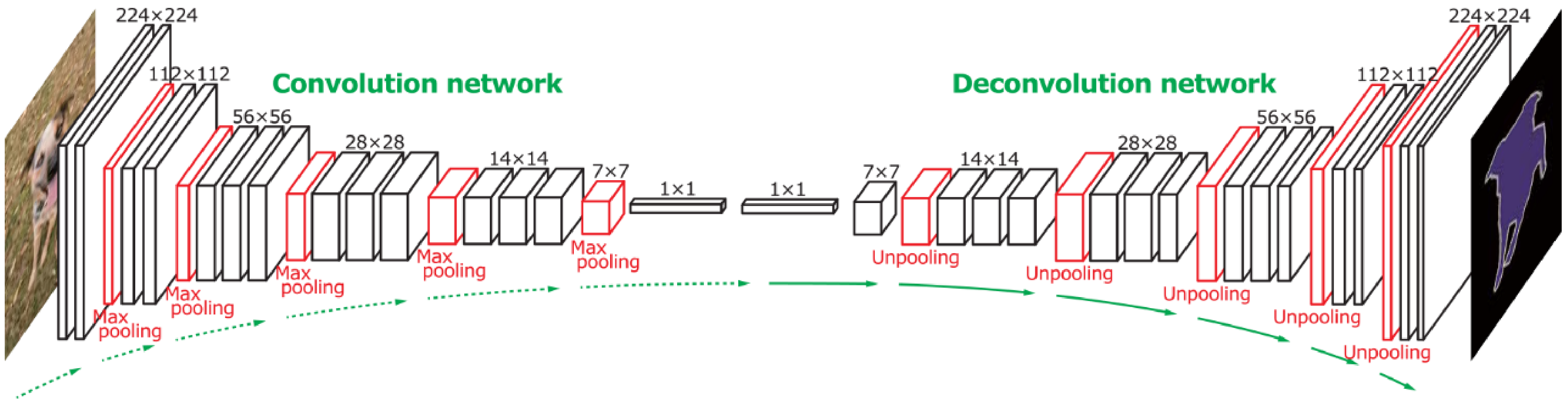
Unpooling


Layer Above
Reconstruction
Unpooling
Max Locations
“Switches”
Pooled Maps
Pooling
Unpooled
Maps
Rectified
Feature Maps

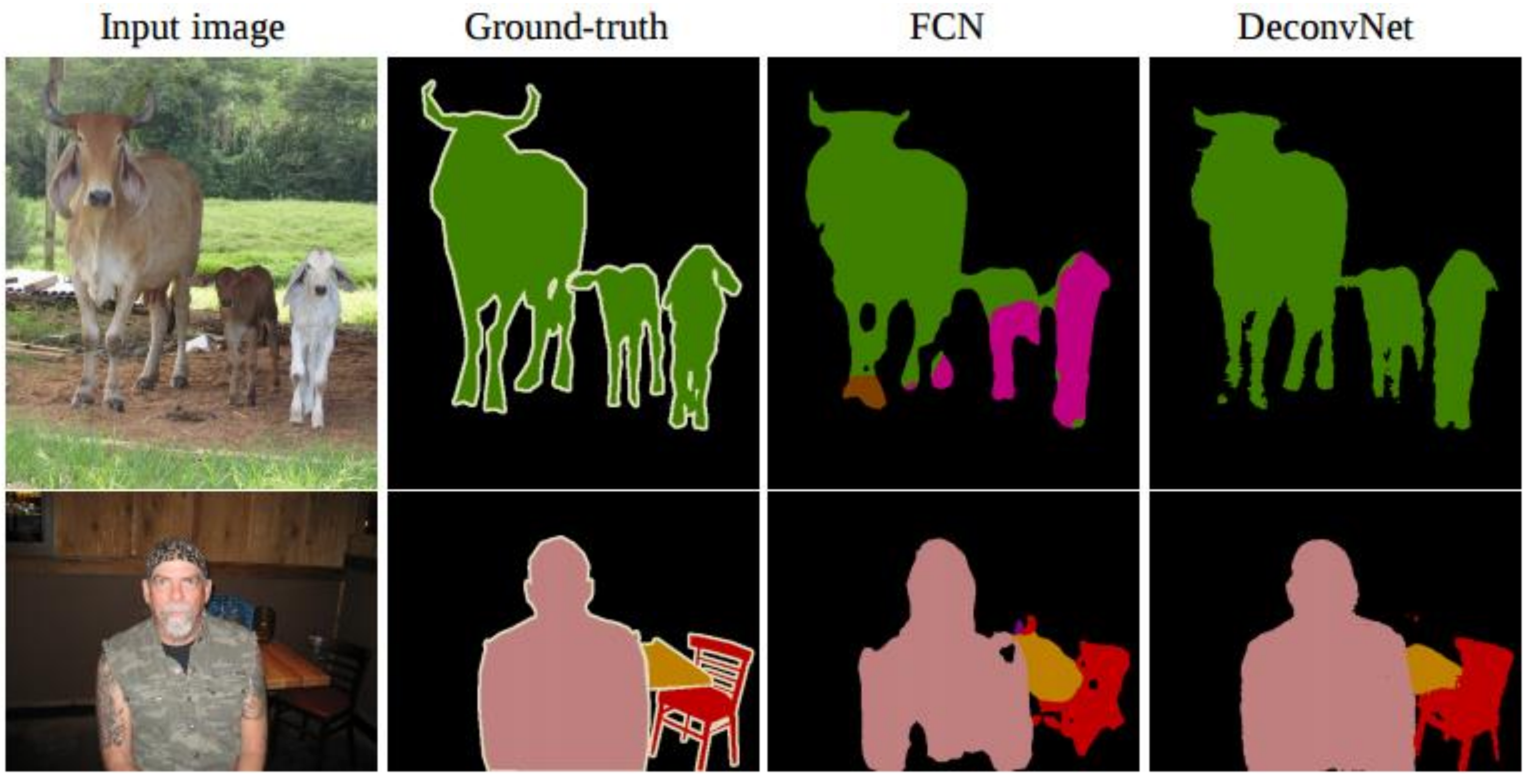
Unpooling


switch
variables
input
pooled
map
Pooling
switch
variables
input
unpooled
map
Unpooling

Example: Deconvnet



Example: Deconvnet

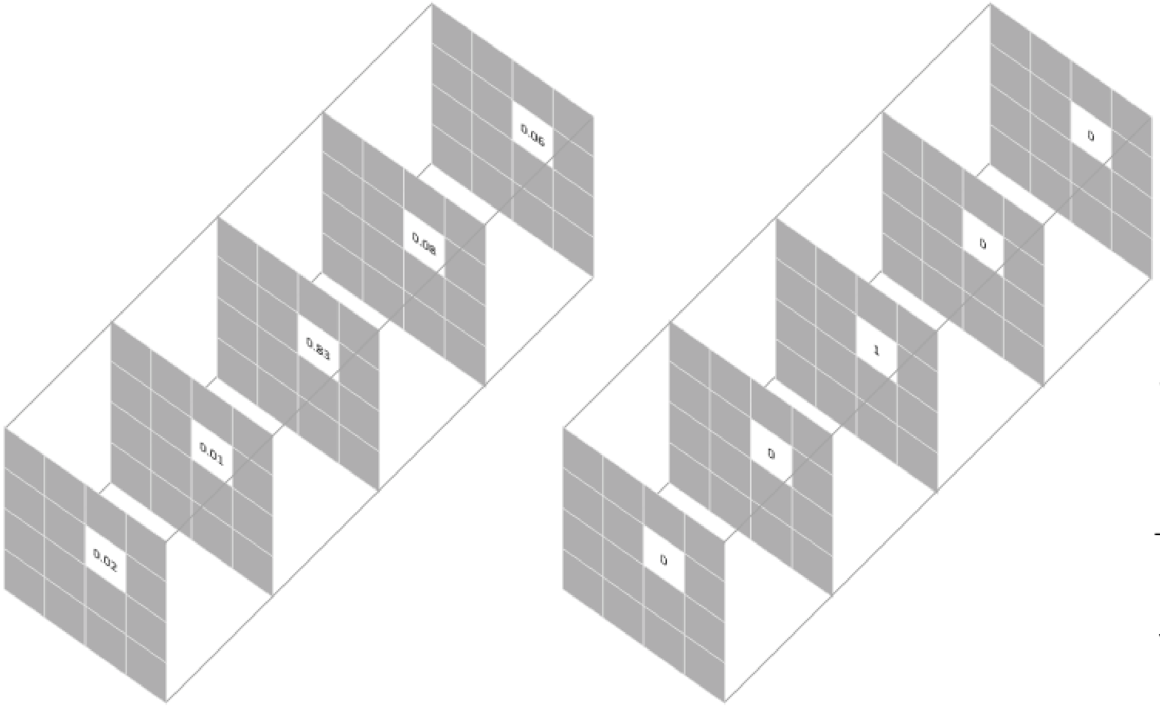


Training an FCN

Loss, Evaluation and Transfer-Learning

Loss

- Same loss function as with classification possible
- Difference: Sum and average across **all pixels**
- Problem: Some classes overrepresented (background)



Prediction for a selected pixel

Target for the corresponding pixel

Pixel-wise loss is calculated as the log loss, summed over all possible classes

$$-\sum_{classes} y_{true} \log\left(y_{pred}\right)$$

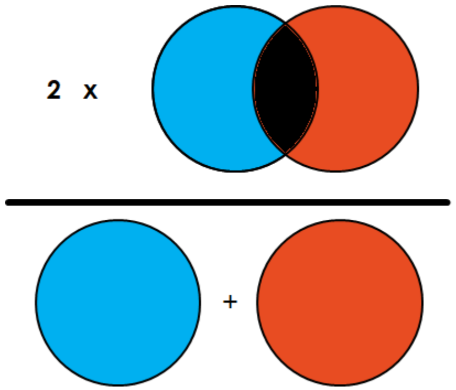
This scoring is repeated over all **pixels** and averaged

Evaluation metrics

Dice-Koefficient

- Effektively F1-Score

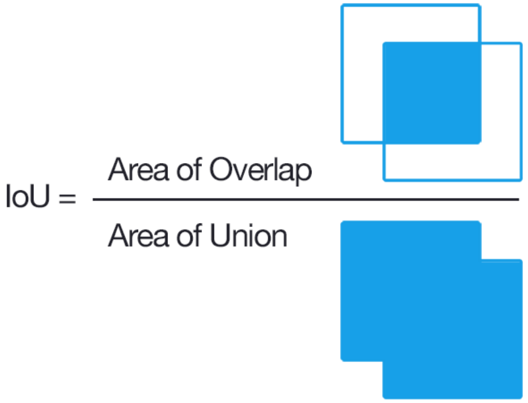
$$\frac{2 \cdot |A \cap B|}{|A| + |B|} = \frac{2 \cdot \mathrm{TP}}{2 \cdot \mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$



Intersection over Union

- Also known as Jaccard Inde

$$\frac{|A \cap B|}{|A| + |B| - |A \cap B|} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$



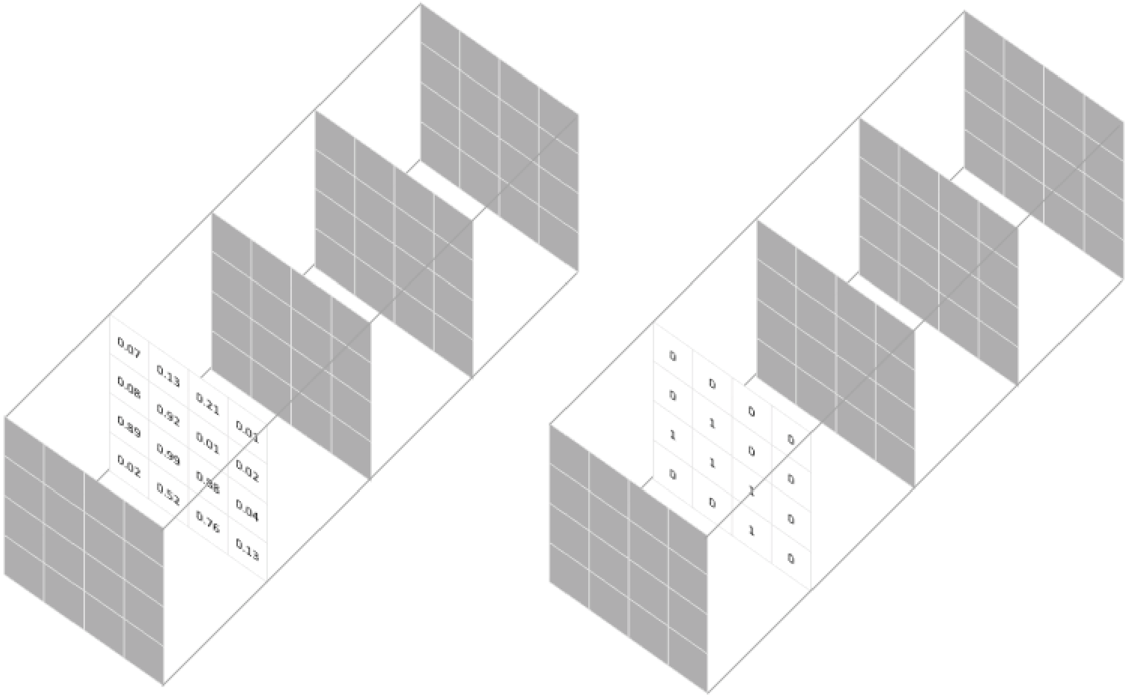
Variation: Dice-Loss

- Goal: Maximizing Dice-Score

→Formulate as differentiable loss function

- y_{true} are GT labels
- y_{pred} are predicted probabilities
- Computation happens per prediction mask

→ Better class balance



Prediction for a selected class

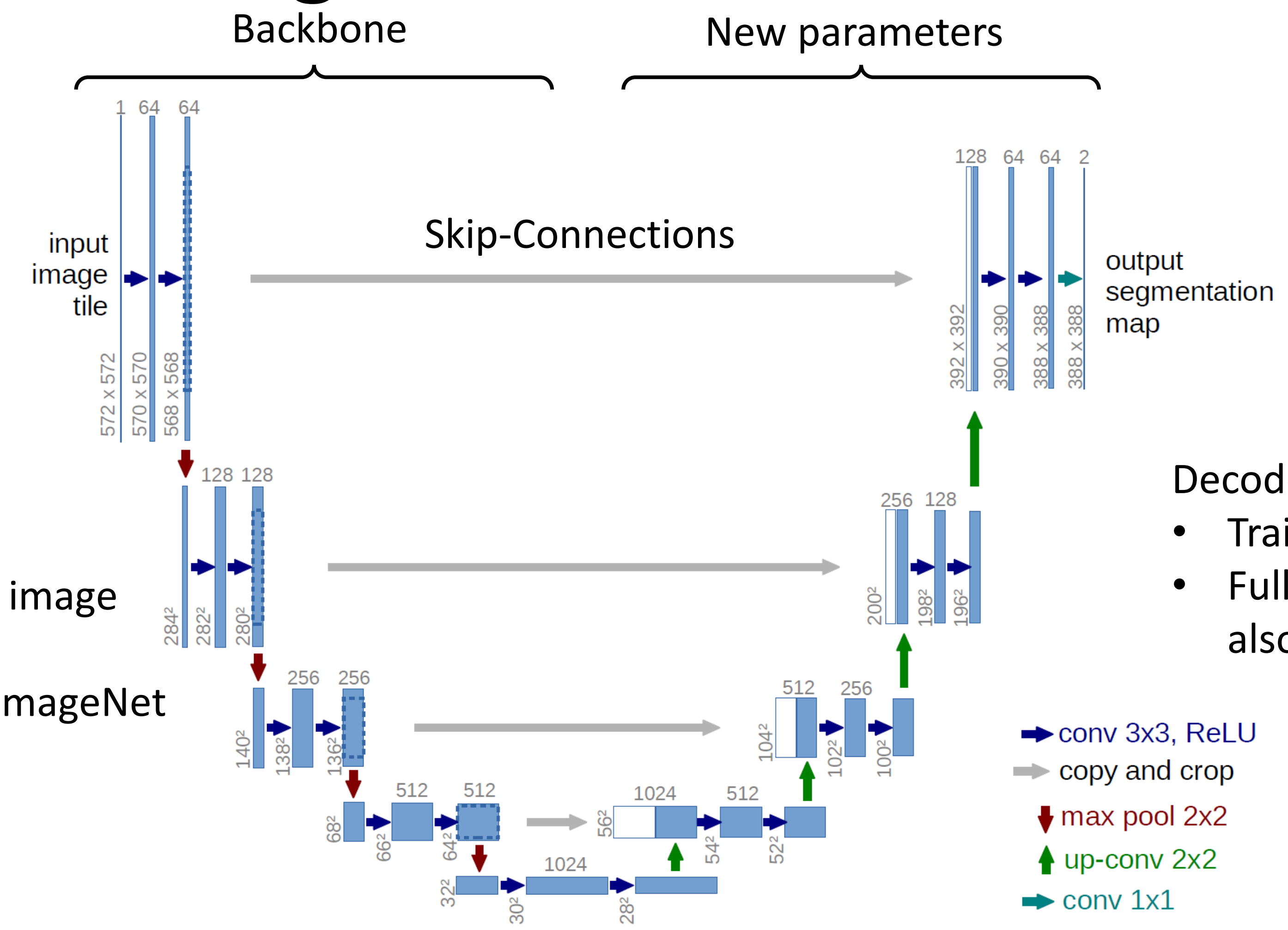
Target for the corresponding class

Soft Dice coefficient is calculated for each class mask

$$1 - \frac{2 \sum_{pixels} y_{true} y_{pred}}{\sum_{pixels} y_{true}^2 + \sum_{pixels} y_{pred}^2}$$

This scoring is repeated over all **classes** and averaged

Transfer Learning



Backbone:

- Popular networks for image classification
- Often pretrained on ImageNet

Decoder parameters:

- Training from Scratch
- Fully pretrained networks also available

Transfer-Learning: Datasets

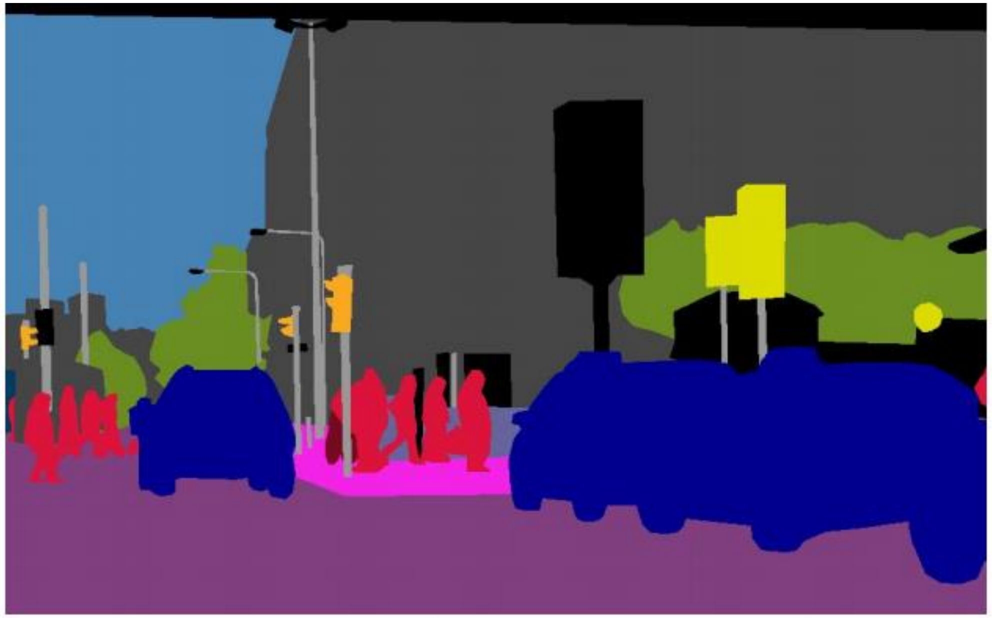
- COCO 2018 Stuff Segmentation
- BDD100K: A Large-scale Diverse Driving Video Database
- Cambridge-driving Labeled Video Database (CamVid)
- Cityscapes Dataset
- Mapillary Vistas Dataset
- Apolloscape Scene Parsing

Instance Segmentation

Mit Ausblick auf Object Detection

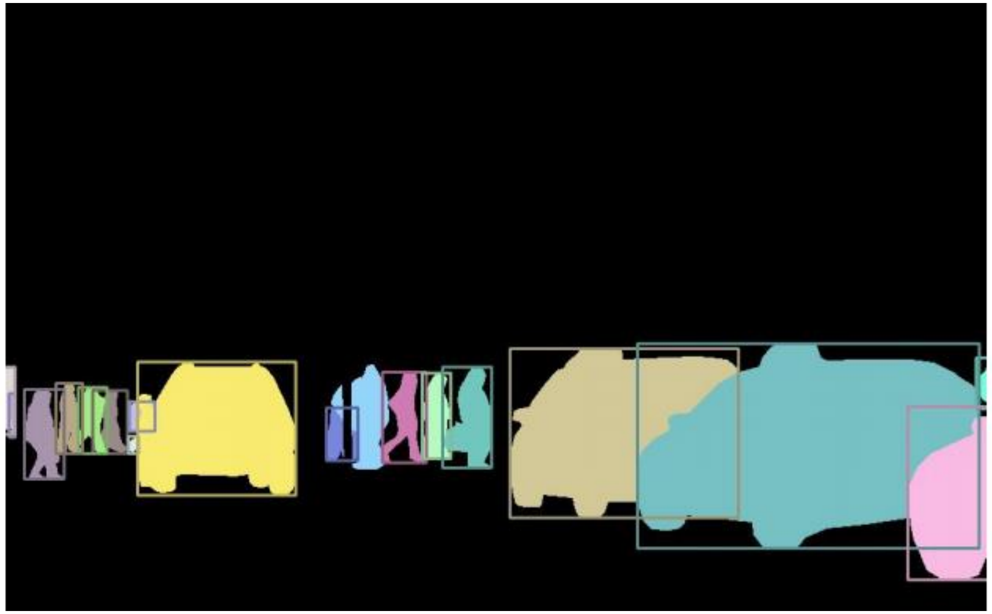
Semantic Segmentation

Only the **type/class** of an object is important



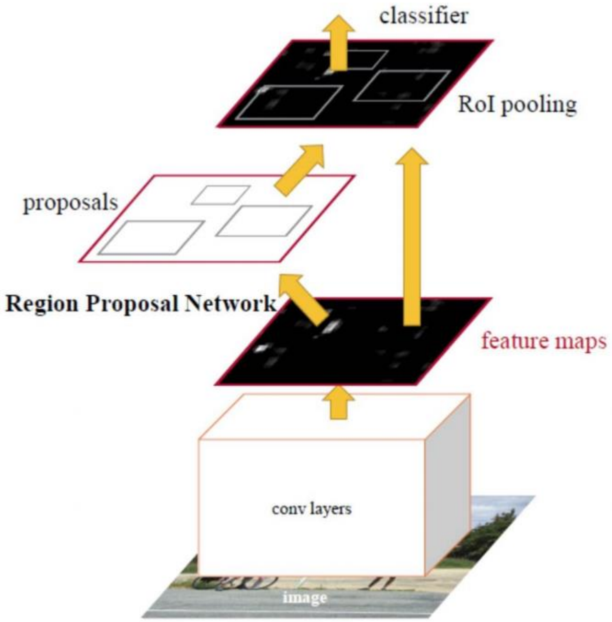
Instance Segmentation

Differentiates between **countable** objects/classes

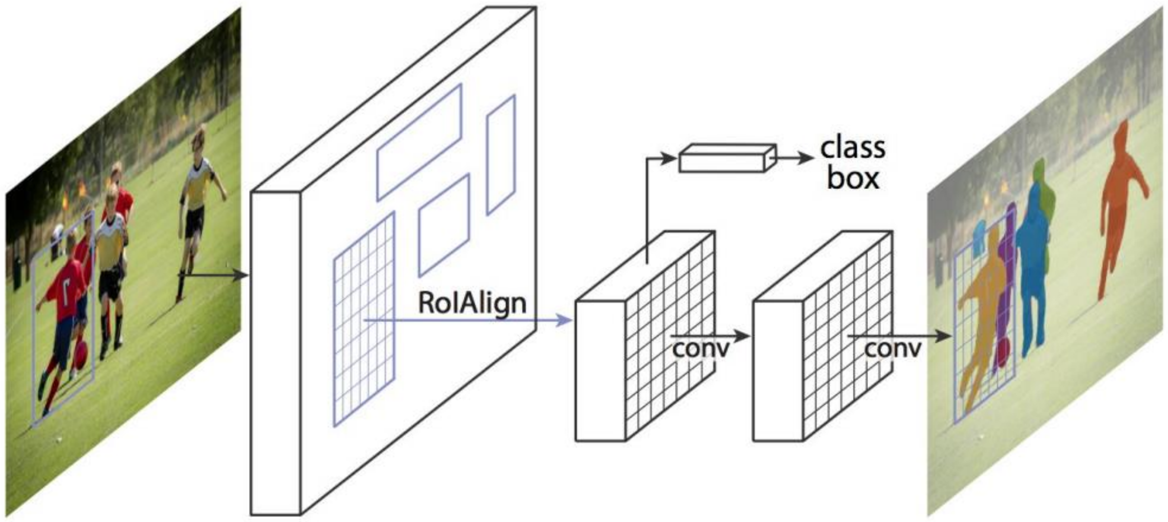


Architecture: Mask R-CNN

- Object detection module is extended by a mask module



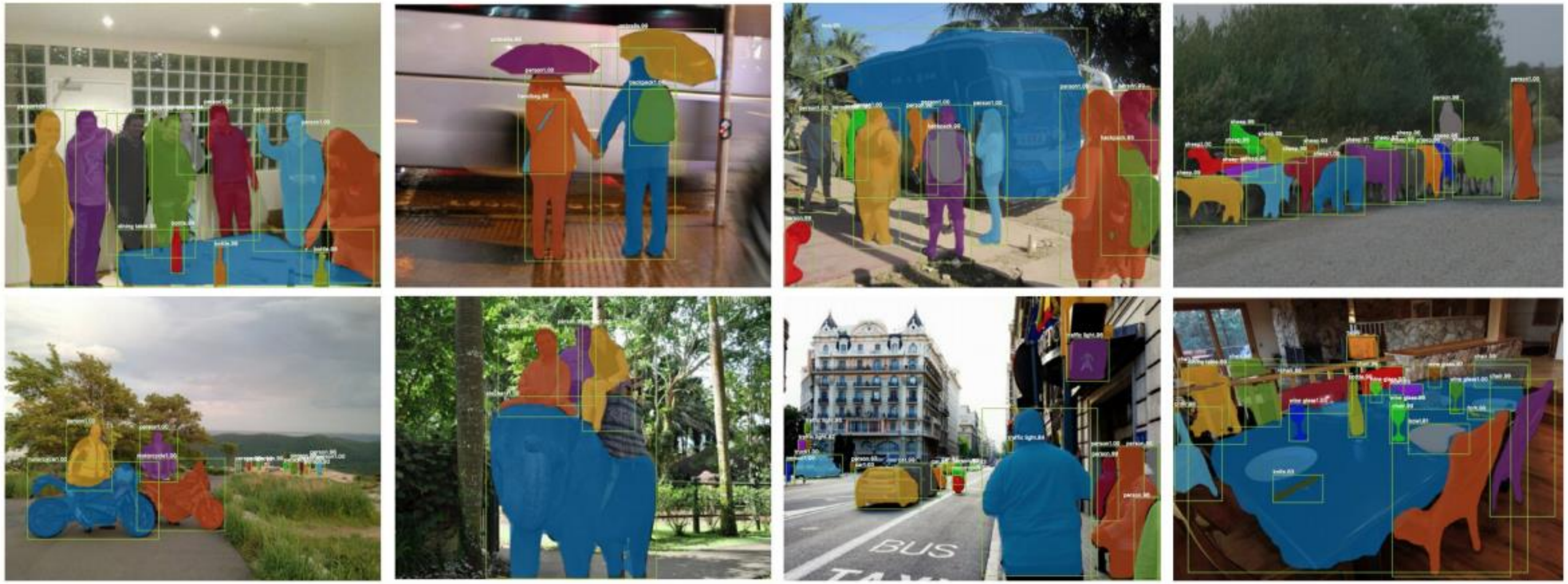
Faster R-CNN for object detection



Extension to Mask R-CNN

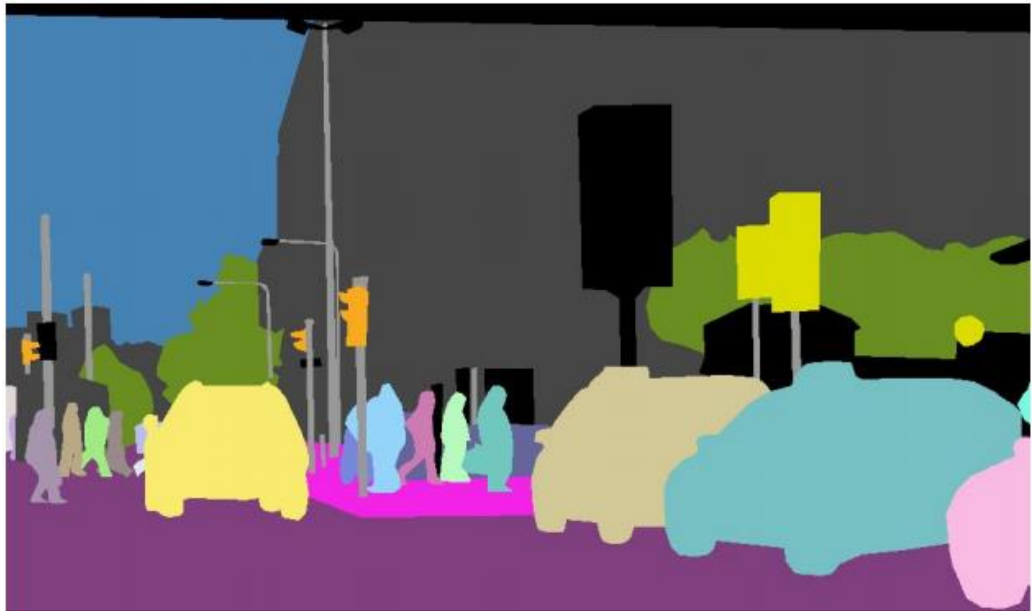
Mask R-CNN: Example

Mask R-CNN performance on COCO



Combination: Panoptic Segmentation

Segmentation of both countable („things“) and not countable („stuff“) objects



Siehe z.B. [„UPSNET: A Unified Panoptic Segmentation Network“]

Outlook

Exercise and next lecture

Outlook: Exercise

- Investigating given FCN architectures
  - Check, if certain approaches are useful
  - Pseudo-Code
  - Compute dimensions
  - Implement (with given data)

Outlook: Next lecture

Object detection:

- Two big approaches
  - Two stage detectors

  → Faster R-CNN
  - One phase/single shot detectors

  → Single Shot Detector, YOLO