GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN IN PUBLICA COMMODA SEIT 1737

COMPUTATIONAL CELL ANALYTICS

Deep Learning for Image Analysis

Segmentation, Tracking & Transfer Learning

Constantin Pape

Institut für Informatik, Georg August Universität Göttingen

@cppape.bsky.social

https://user.informatik.uni-goettingen.de/~pape41/

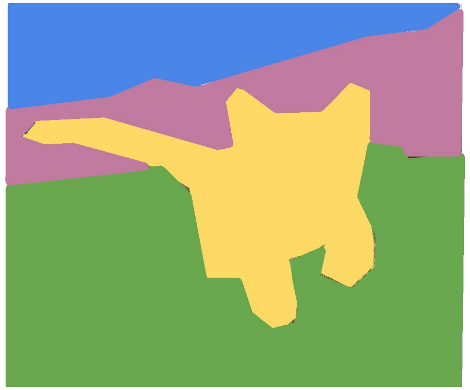
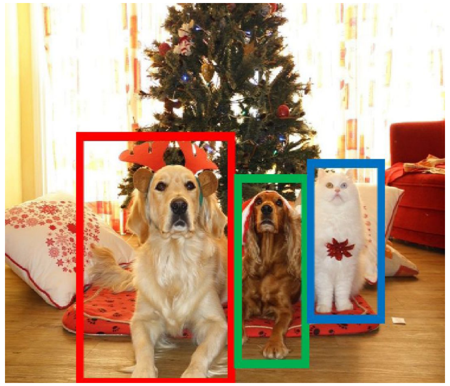
Plan for today

- 9:00 - 10:30 - Lecture, Q&A & Intro to Exercise 2
- 10:30 - 12:30 - Work on Exercise 2
- 12:00 - 13:00 - Lunch
- 13:00 - 16:00 - Work on exercise 2
- 16:00 - Recap Exercise 2

Semantic Segmentation

and the U-Net architecture

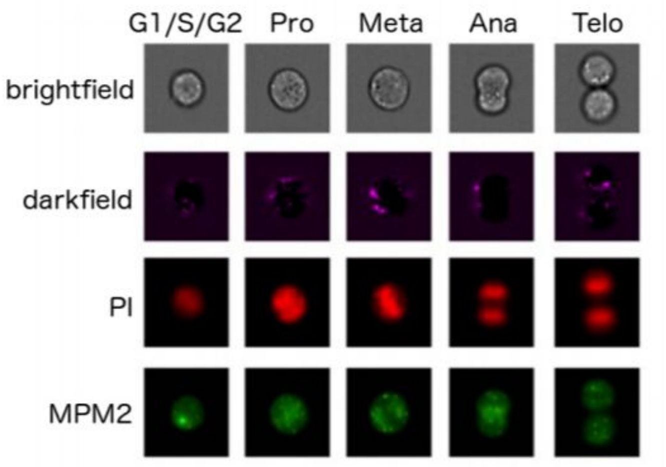
Segmentation tasks: Scene understanding

Image Classification	Semantic Segmentation	Object Detection	Instance Segmentation
			
Cat	Sky, Trees, Grass, Cat	Dog, Dog, Cat	Dog, Dog, Cat
No spatial extent	No objects, just pixels	Multiple objects	

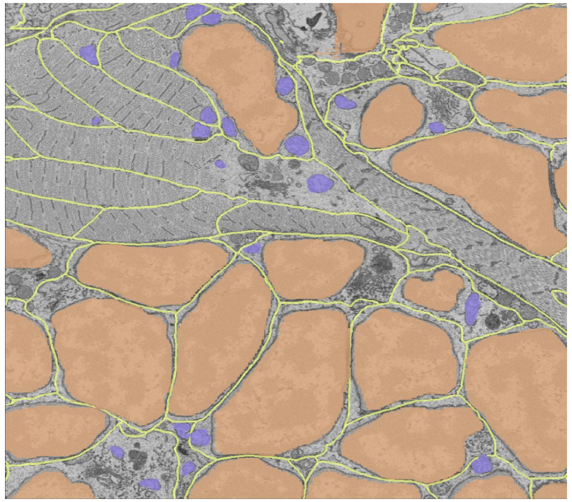
Source: Fei Fei Li, Justin Johnson, Serena Yeung

Segmentation for biomedical images

Image Classification



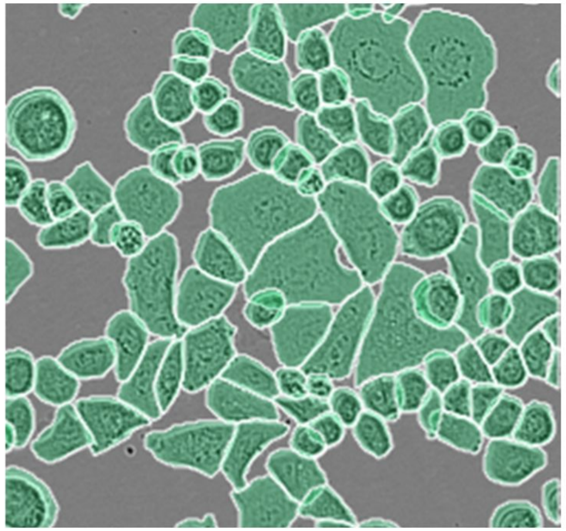
Semantic Segmentation



Cell Membrane
Nuclei
Mitochondria

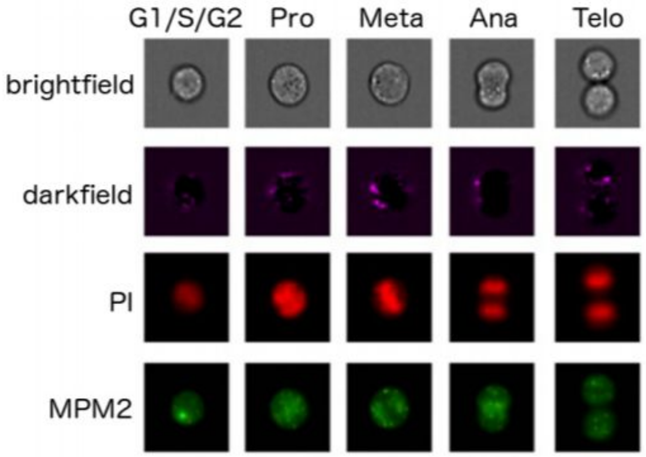
Object Detection

Instance Segmentation

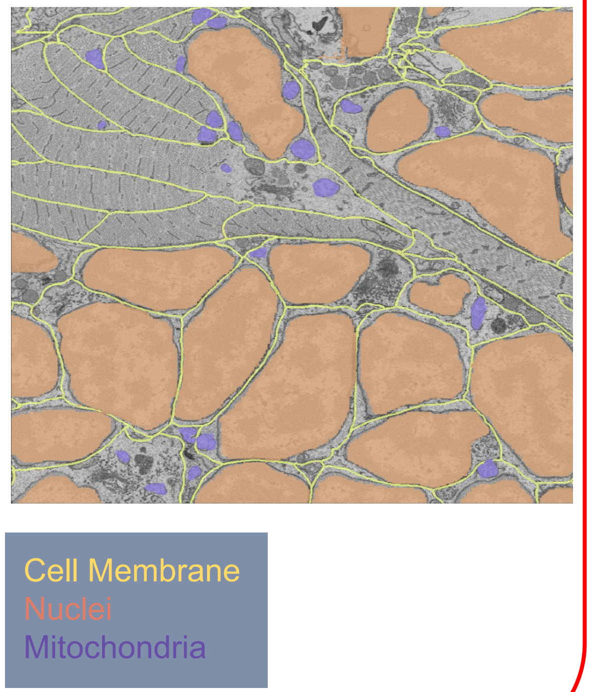


Segmentation for biomedical images

Image Classification

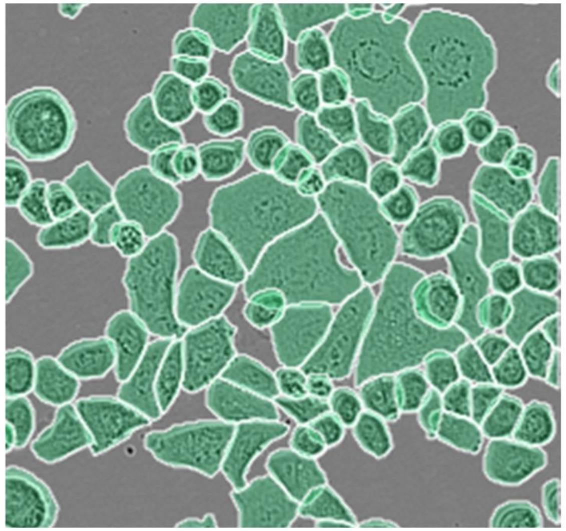


Semantic Segmentation

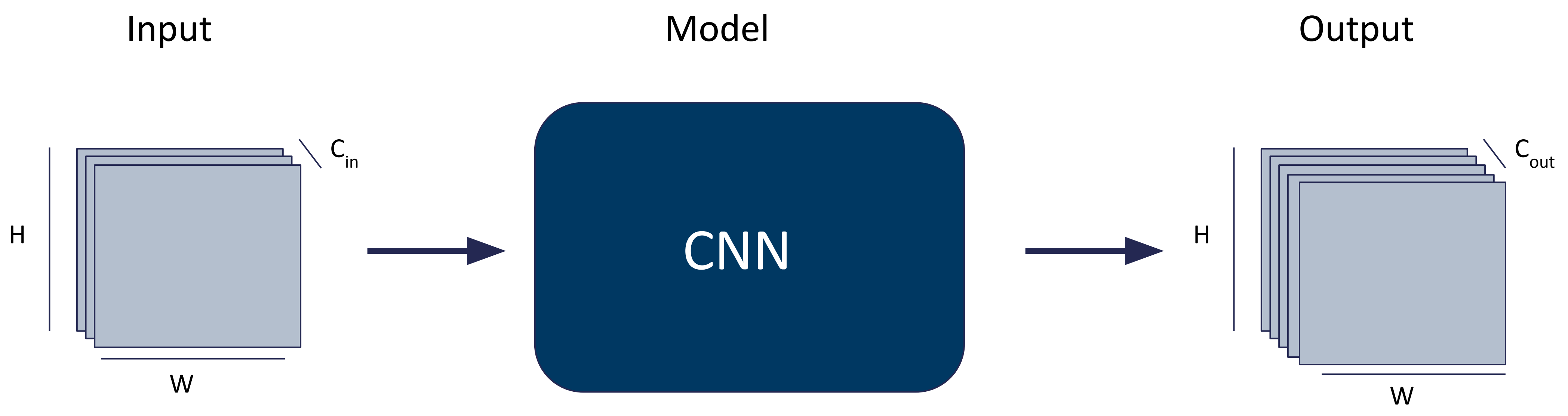


Object Detection

Instance Segmentation



Architectures for semantic segmentation

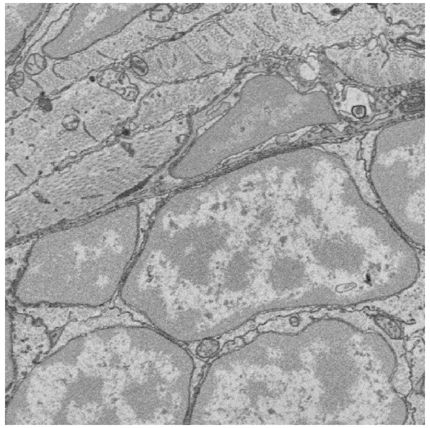


Input: image with C_{in} channels (e.g. 1 for single channel)

Output: image with C_{out} channels, C_{out} = number of classes (one-hot encoding)

Architectures for semantic segmentation

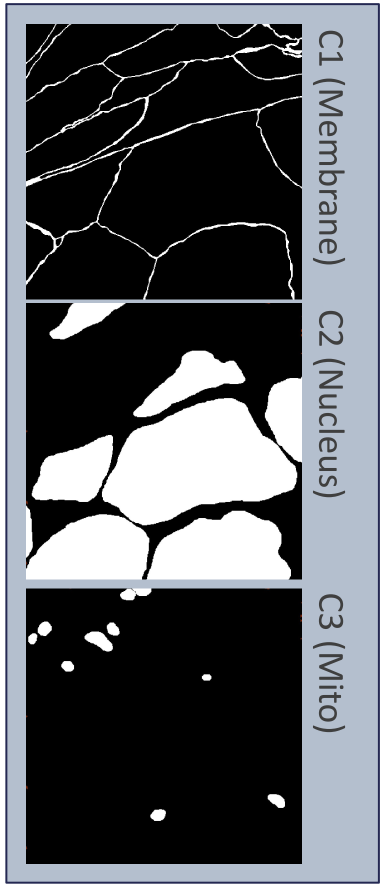
Input



Model

CNN

Output



Input: image with C_{in} channels (e.g. 1 for single channel)

Output: image with C_{out} channels, C_{out} = number of classes (one-hot encoding)

Architectures for semantic segmentation

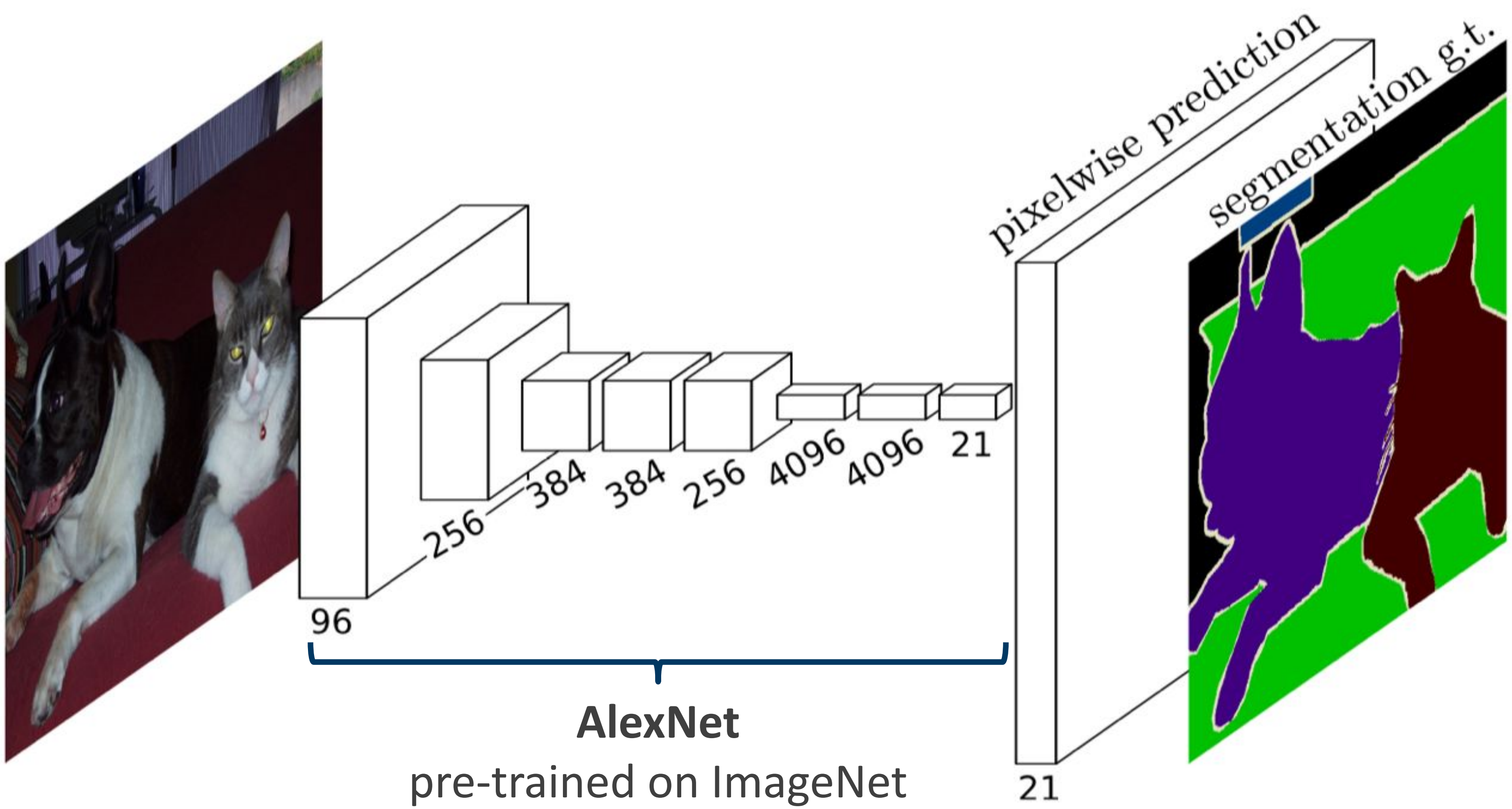
Input Model Output

Binary nucleus segmentation

Special case: foreground vs. background segmentation:

$\mathbf{C_{out} = 1}$ (binary segmentation)

Fully convolutional network



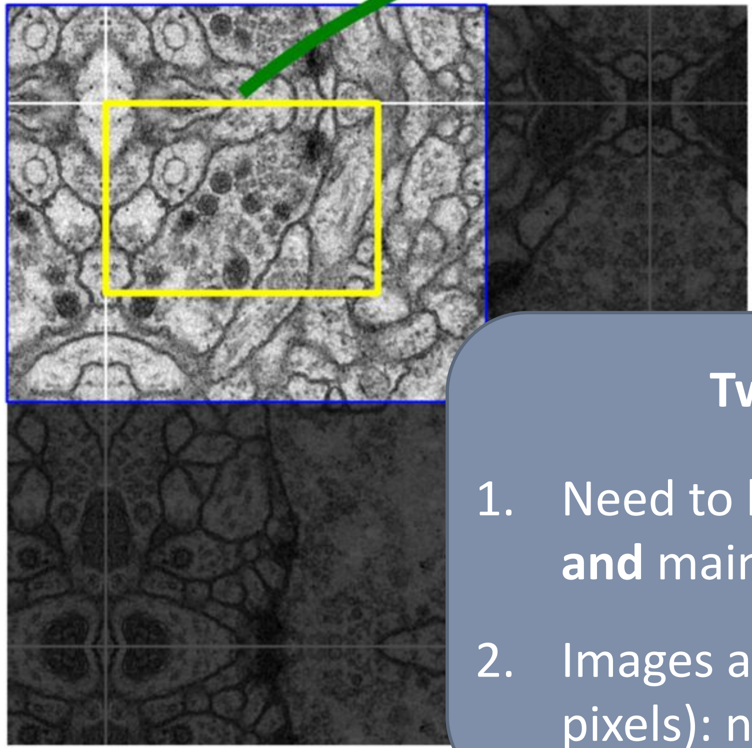
AlexNet
pre-trained on ImageNet

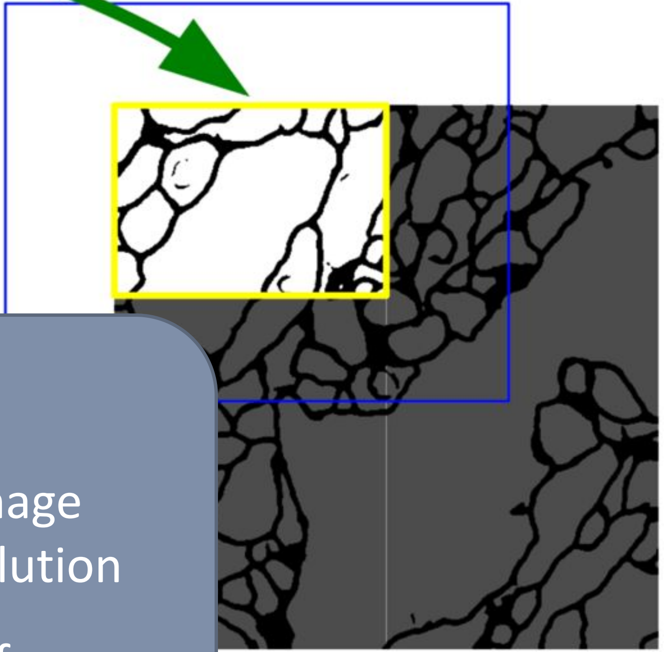
- Pretrained classification network, adapted for segmentation
  - "Transfer learning" (tomorrow)
- Outputs from convolutional layers is upsampled to predict segmentation map
- **Not high-resolution enough for biomedical images!**

Long, Shelhamer, Darrell, *CVPR* 2015

Biomedical image segmentation

Task: segment neuronal structures from electron microscopy images



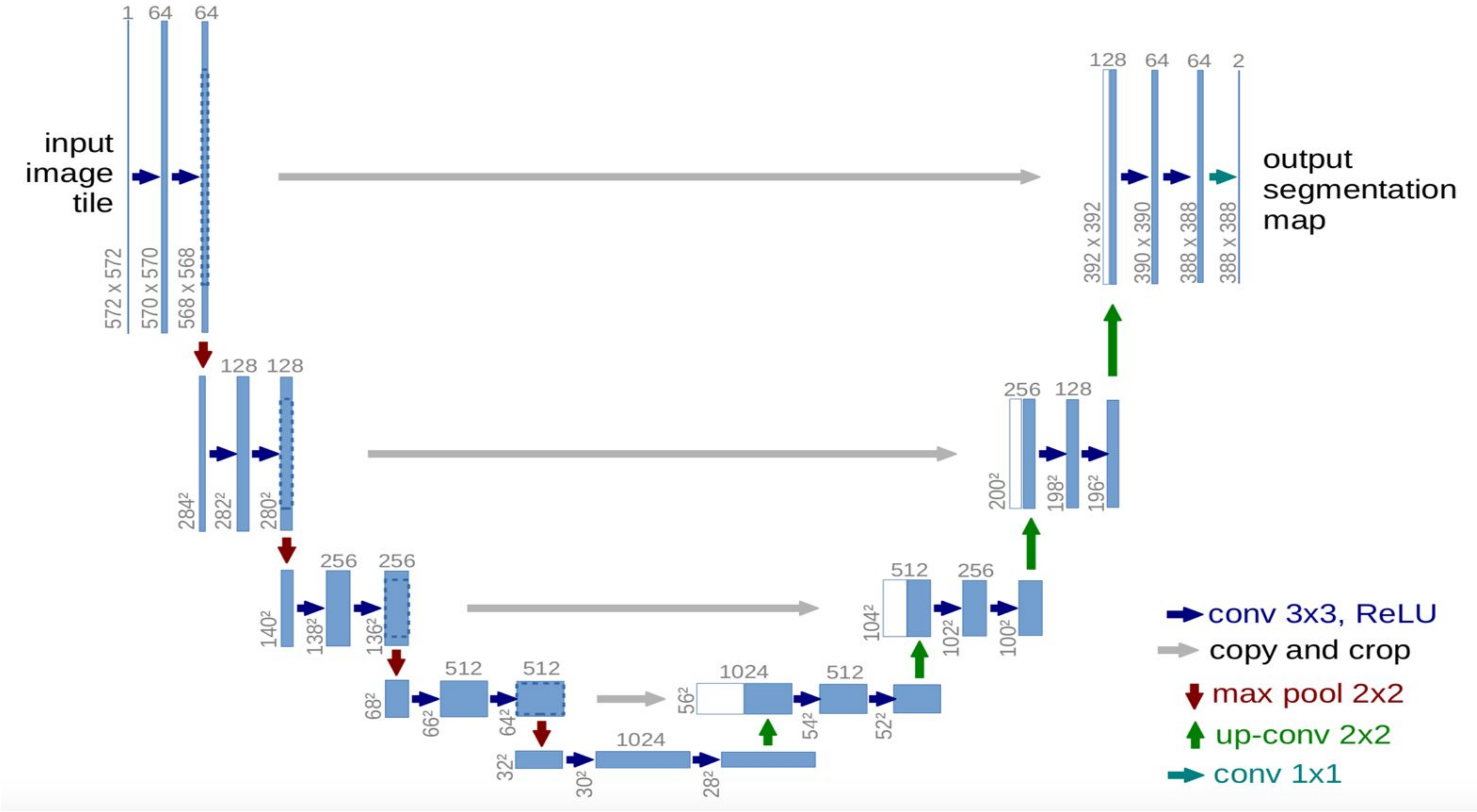


Two main challenges

1. Need to know what's in the image **and** maintain high spatial resolution
2. Images are huge (thousands of pixels): need to tile images

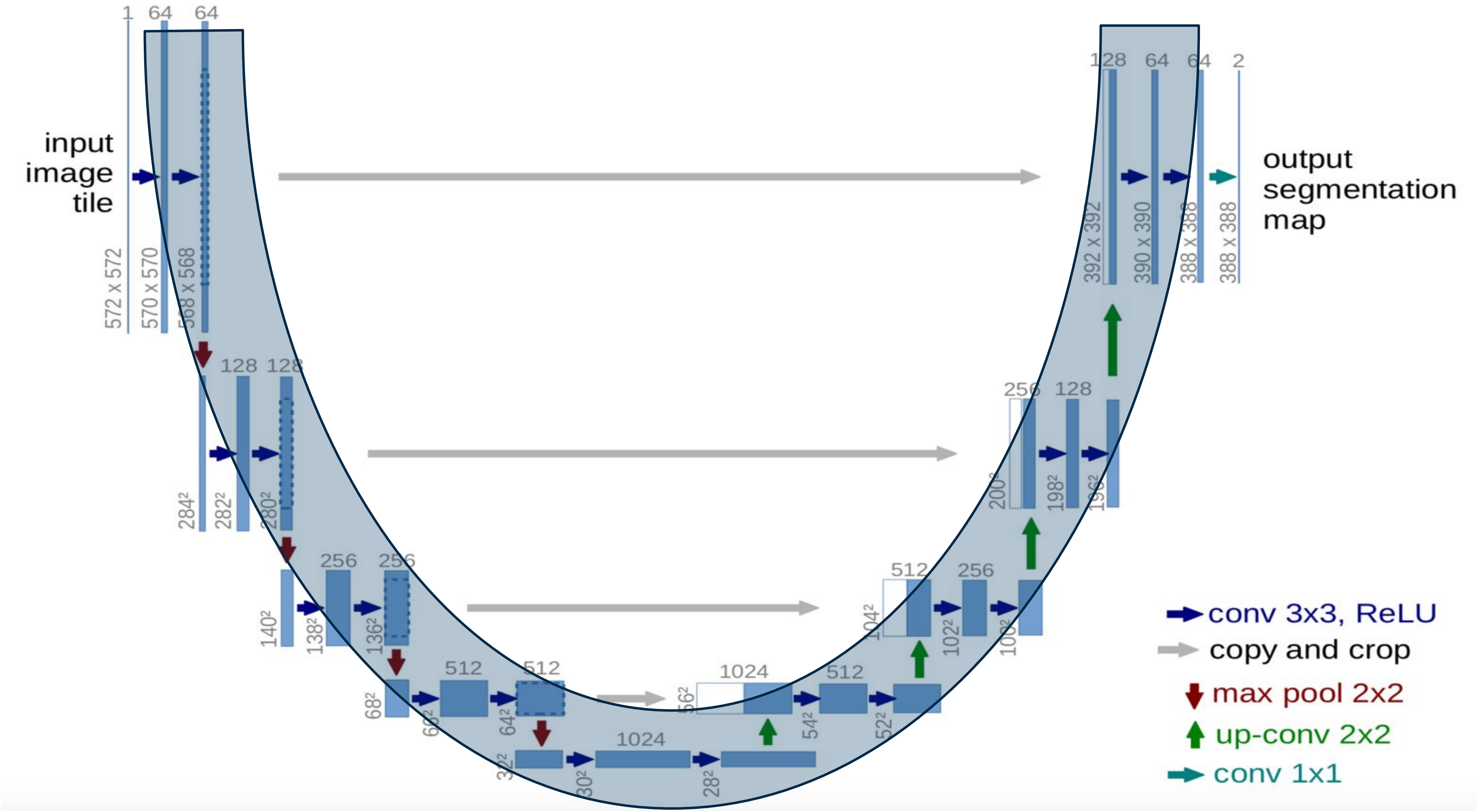
Ronneberger, Fischer, Brox, *MICCAI* 2015

U-Net



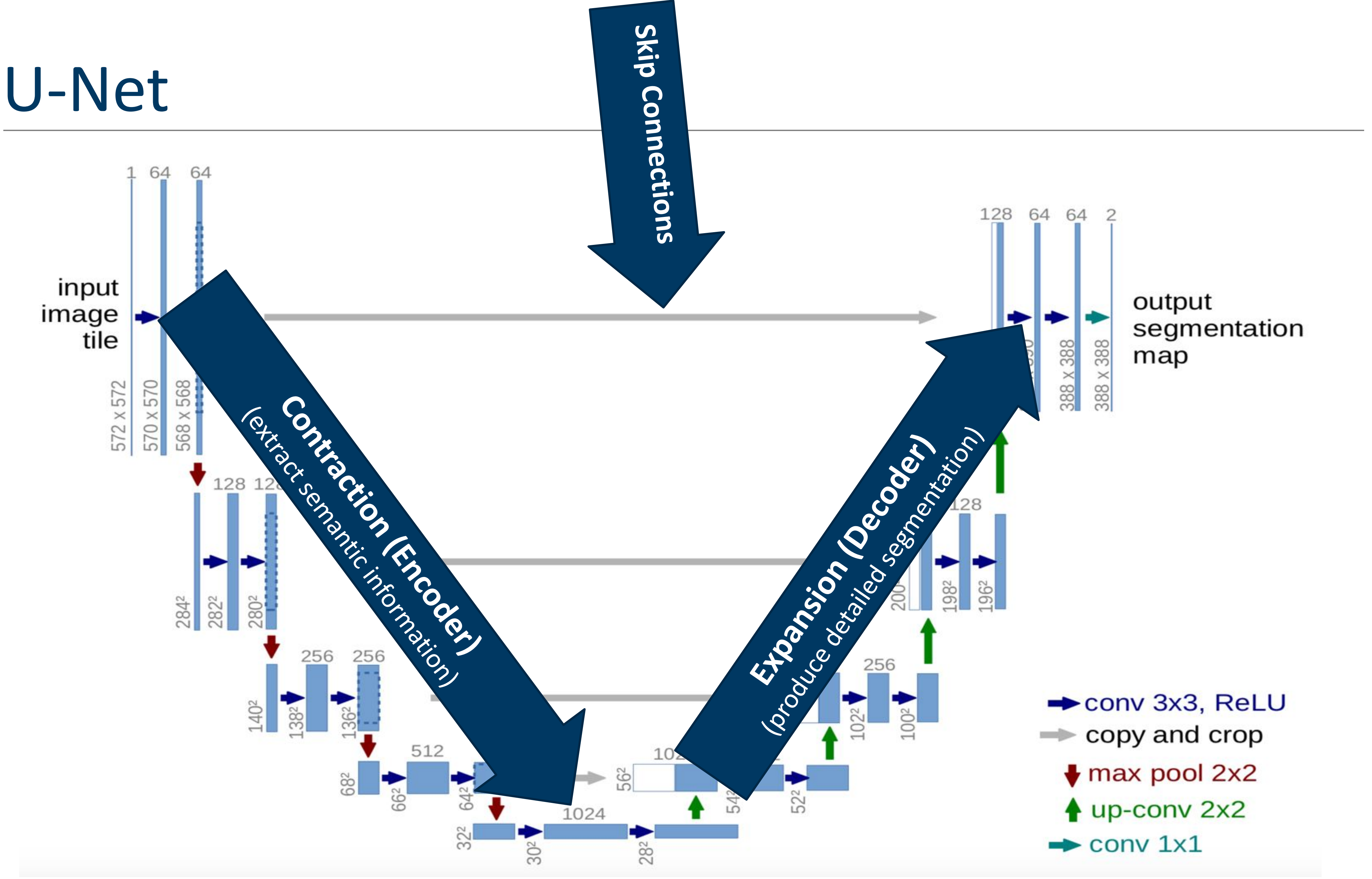
Ronneberger, Fischer, Brox, *MICCAI* 2015

U-Net



Ronneberger, Fischer, Brox, *MICCAI* 2015

U-Net



Ronneberger, Fischer, Brox, *MICCAI* 2015

U-Net: Encoder

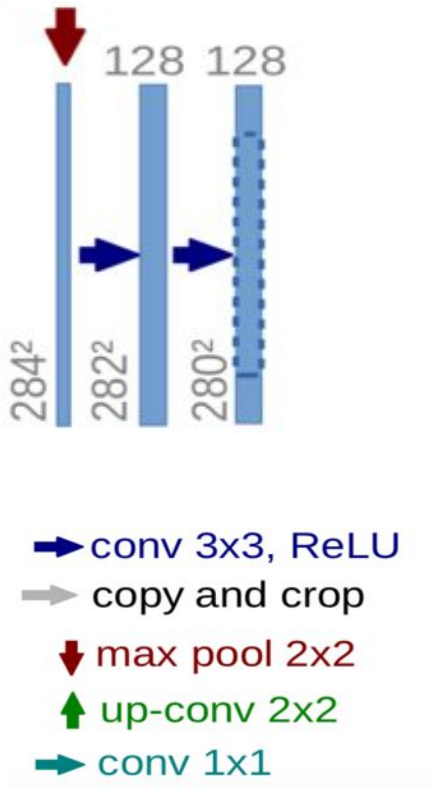
One level in the encoder:

- 2 convolutional layers with 3x3 kernel
  - Don't use padding -> lose 2 pixels per convolution (we will do this differently in the exercises)

- ReLU activation (the standard!)

- 2x2 max pooling to group features and reduce spatial dimension

We have seen all this yesterday!



U-Net: Encoder

One level in the encoder:

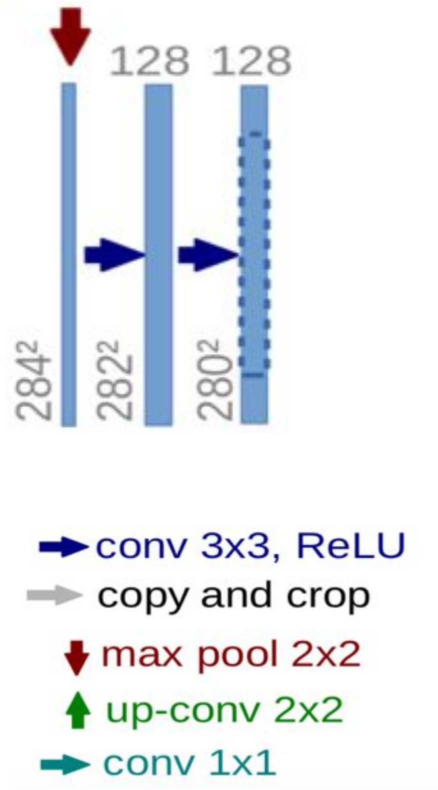
- 2 convolutional layers with 3x3 kernel
  - Don’t use padding -> lose 2 pixels per convolution (we will do this differently in the exercises)

- ReLU activation (the standard!)

- 2x2 max pooling to group features and reduce spatial dimension

We have seen all this yesterday!

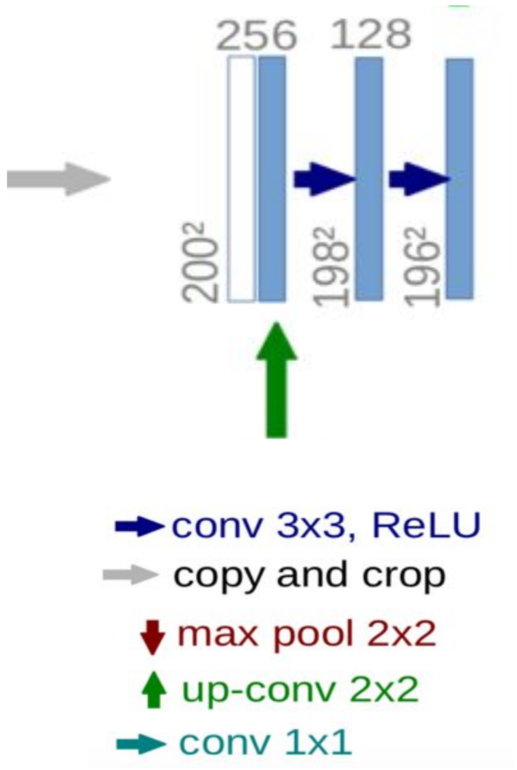
In the exercise we will also add a normalization layer. Keep the network activation in well-defined range and make training more stable.



U-Net: Decoder

One level in the decoder:

- 2 convolutional layers + ReLU (as before)
- 2x2 up-conv to (spatially upsample the features)

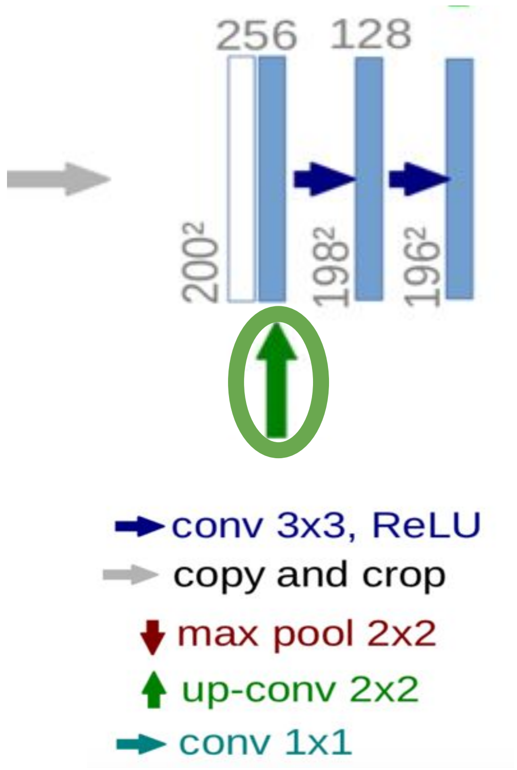


U-Net: Decoder

One level in the decoder:

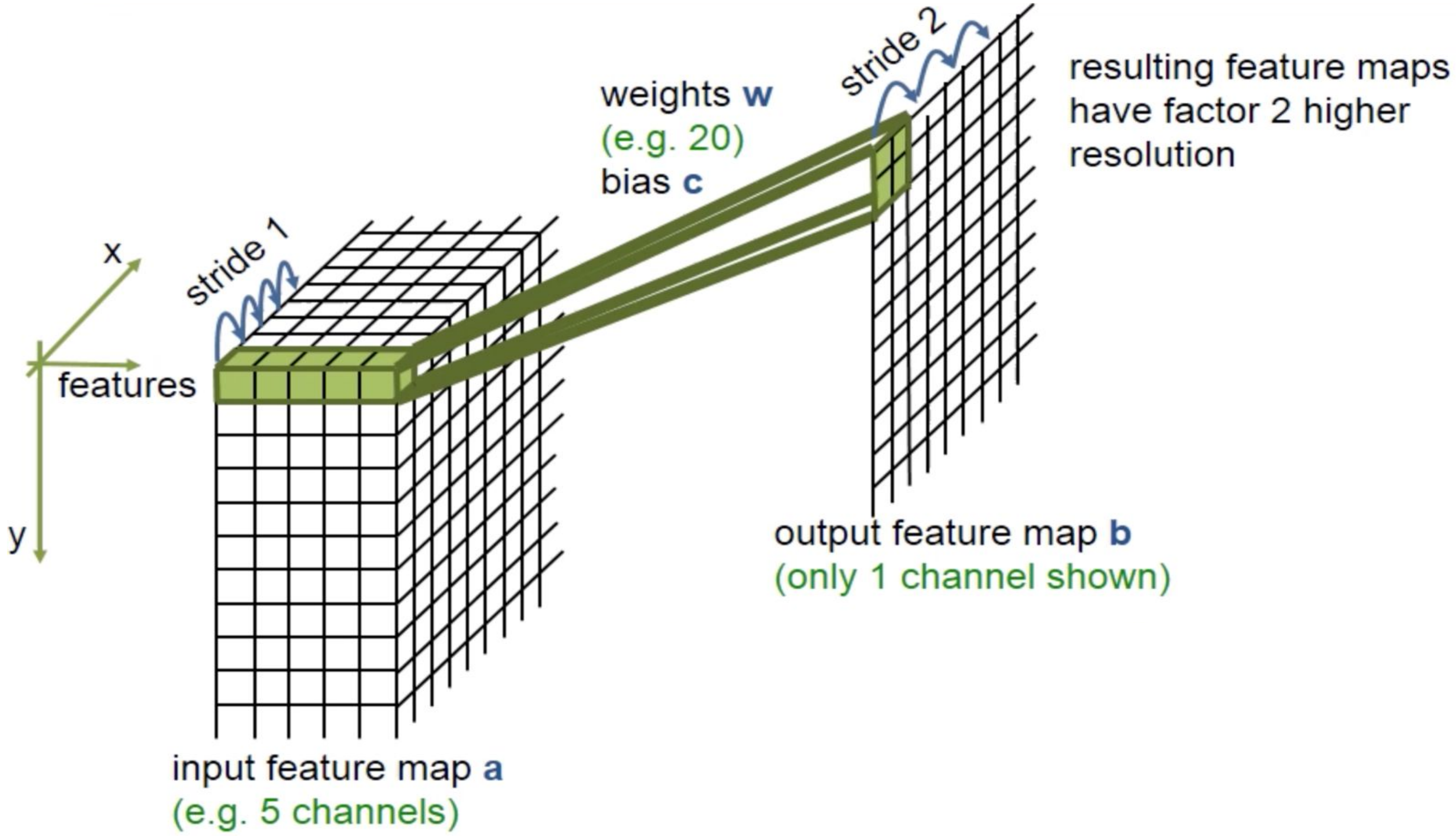
- 2 convolutional layers + ReLU (as before)
- 2x2 up-conv to (spatially upsample the features)

This is new!



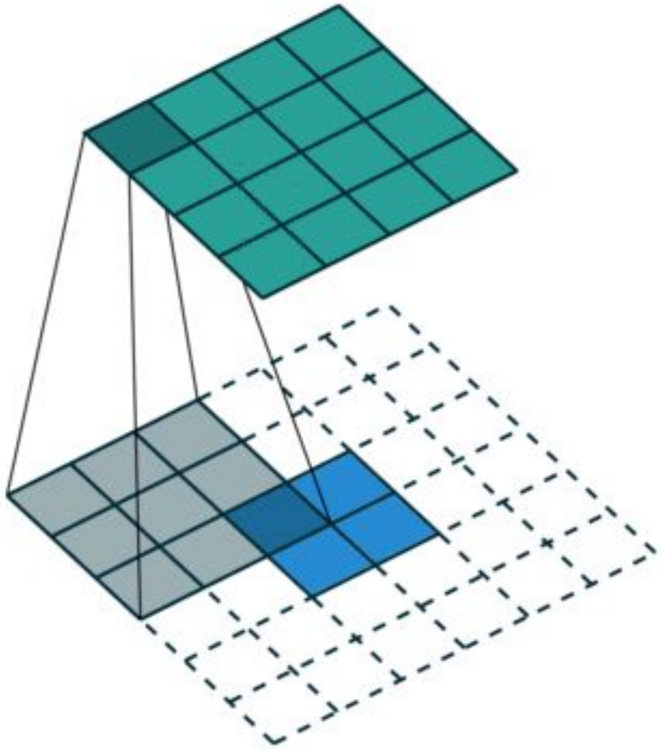
Up-convolutions

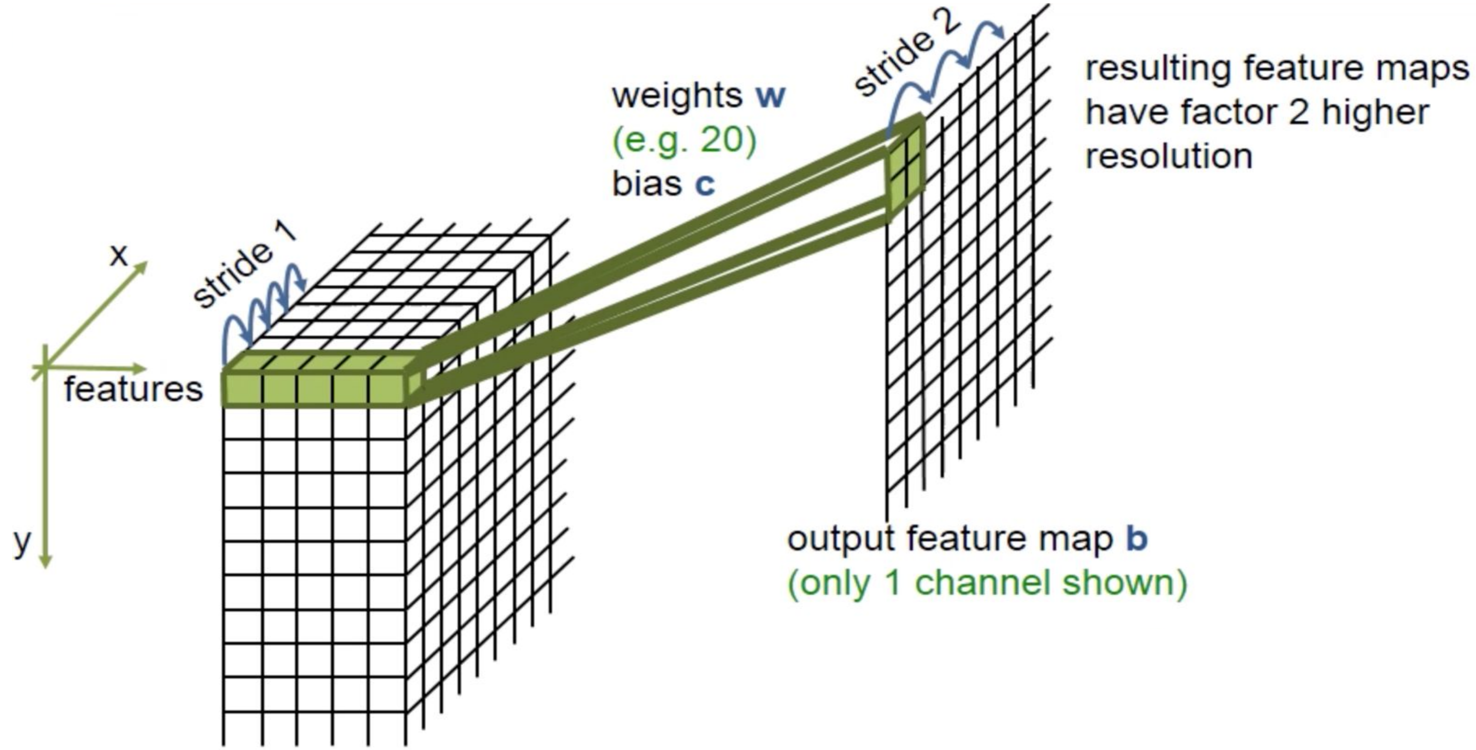
Increase the spatial dimension ("resolution") with a learned convolutional kernel.



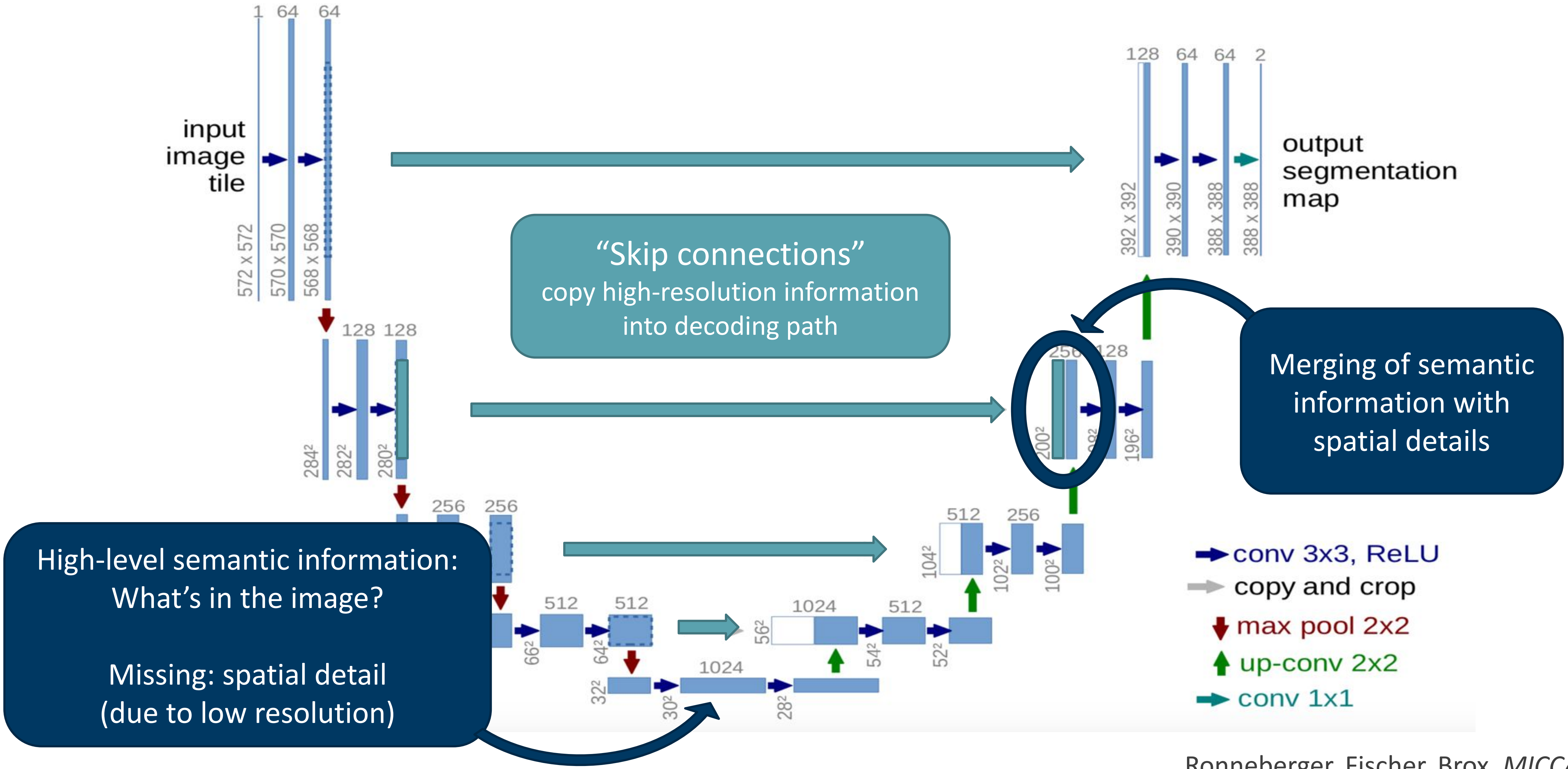
Up-convolutions

Increase the spatial dimension ("resolution") with a learned convolutional kernel.



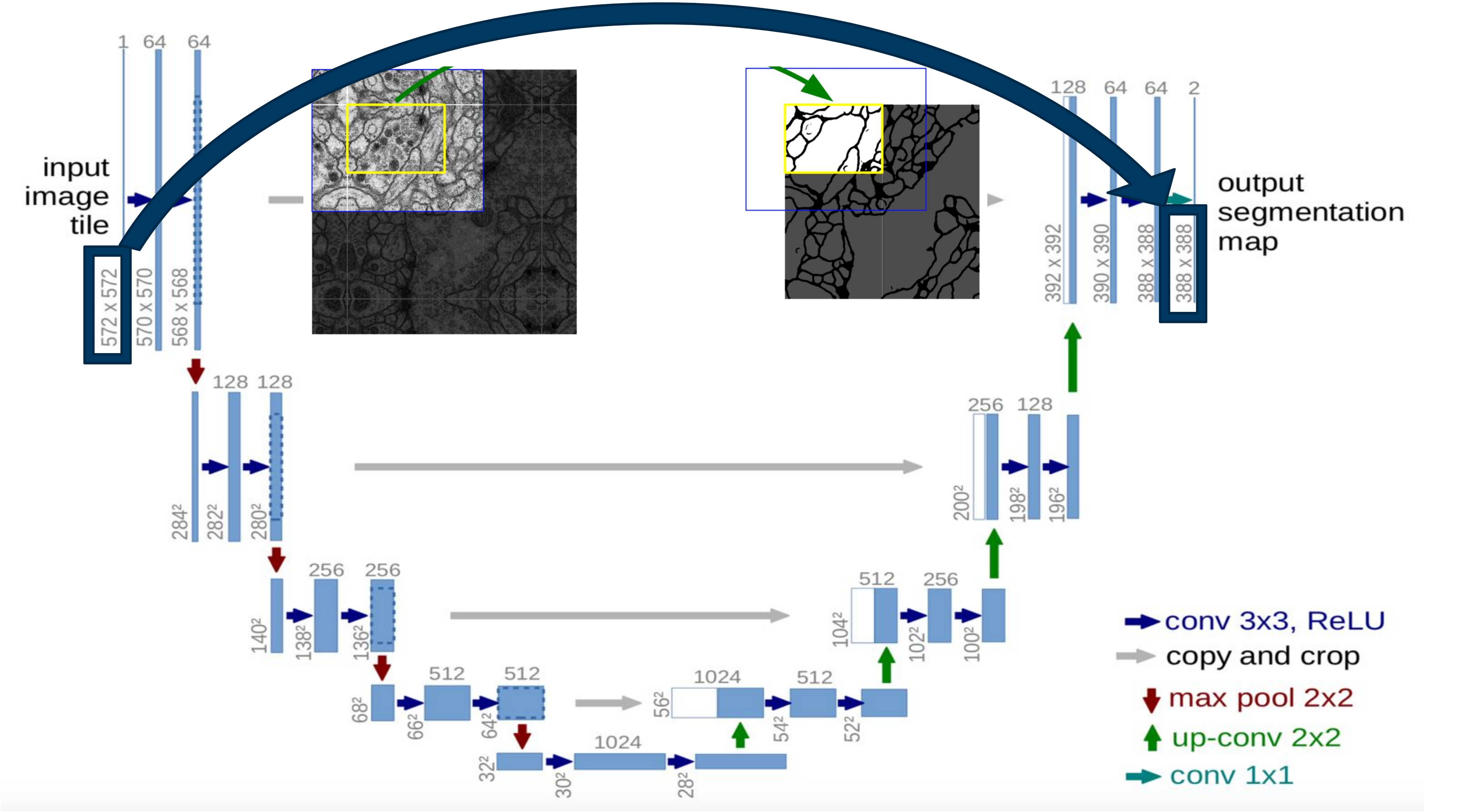


Skip connections



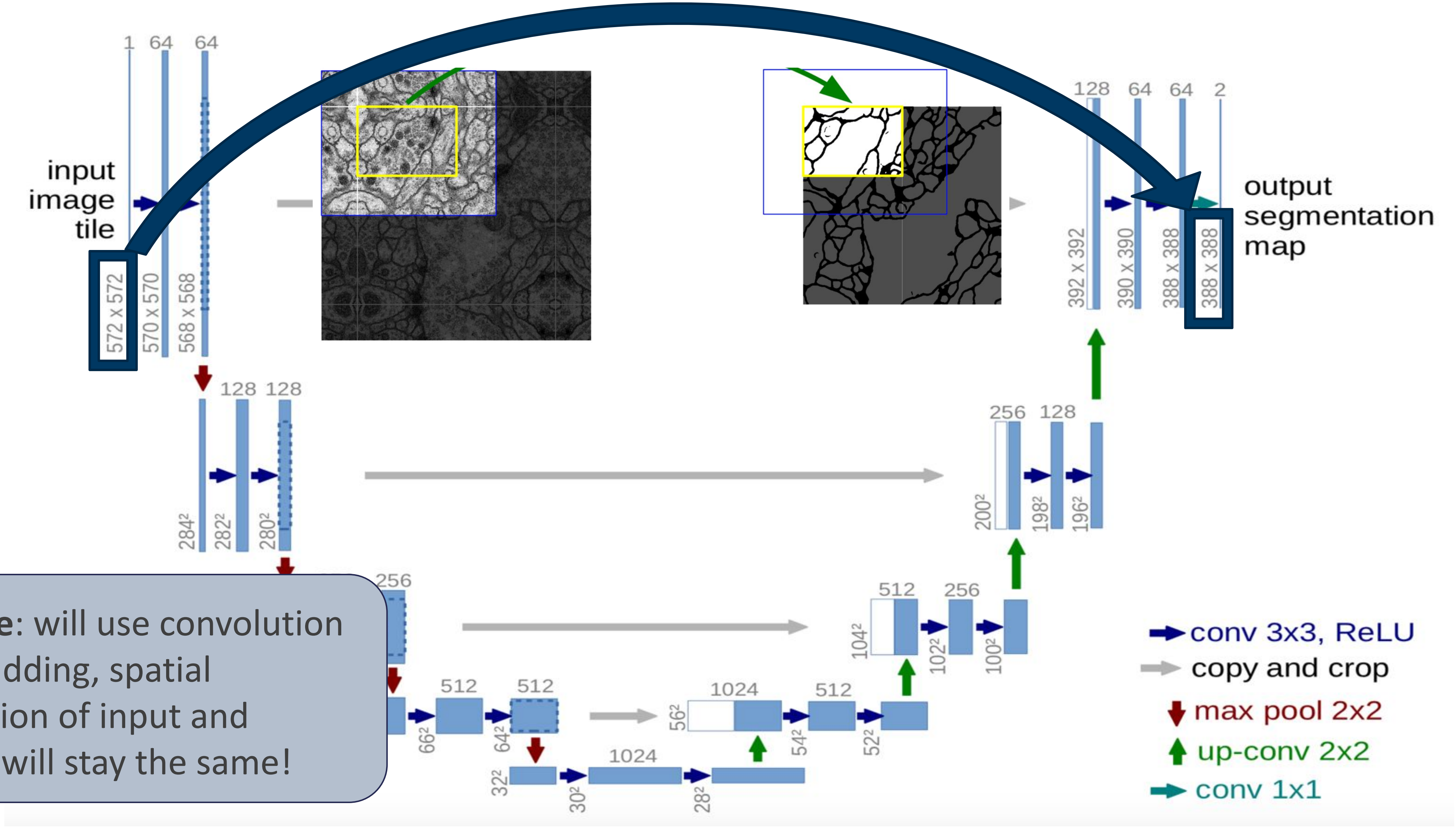
Ronneberger, Fischer, Brox, *MICCAI* 2015

Spatial dimensions



Ronneberger, Fischer, Brox, *MICCAI* 2015

Spatial dimensions



Exercise: will use convolution with padding, spatial dimension of input and output will stay the same!

Ronneberger, Fischer, Brox, *MICCAI* 2015

Loss function

Pixel-wise cross entropy: $$-\frac{1}{N}\sum_{ij}\sum_{k} \log p_{ij}^{k} \; t_{ij}^{k}$$

Output p_{ij}

i

j

Per-pixel

Loss

Target t_{ij}

i

j

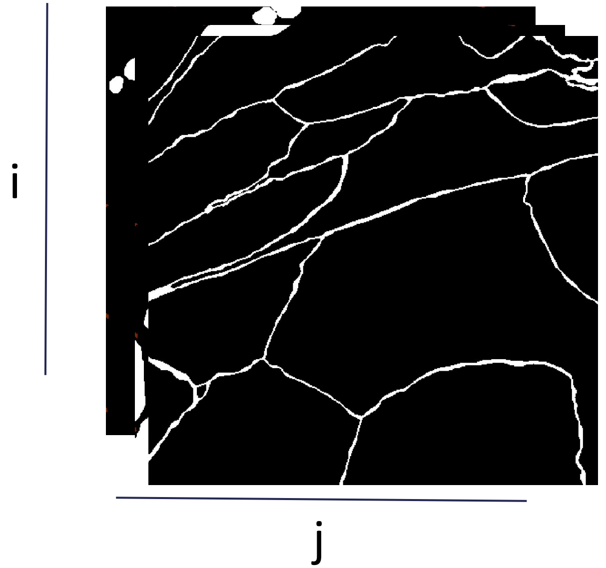
Loss function

Sum pixels

Pixel-wise cross entropy: $$-\frac{1}{N}\sum_{ij}\sum_{k}\log p_{ij}^{k}\; t_{ij}^{k}$$

Sum classes

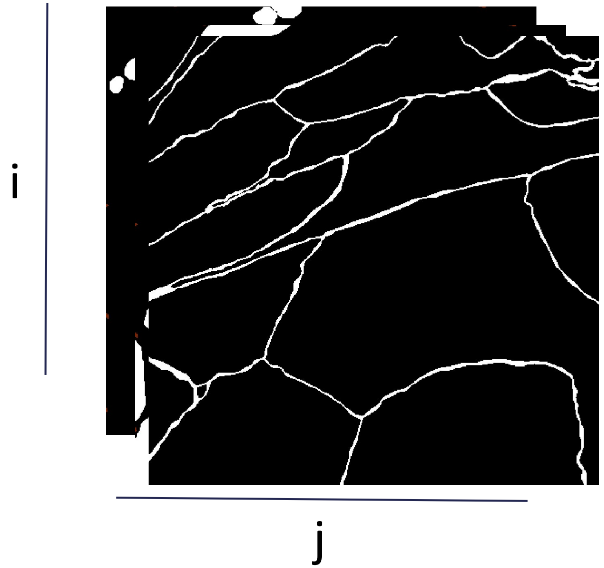
Output p_{ij}



Per-pixel

Loss

Target t_{ij}



Loss function

Sum pixels

Pixel-wise cross entropy: $$-\frac{1}{N}\sum_{ij}\sum_{k} \log p_{ij}^{k} \; t_{ij}^{k}$$

Sum classes

Output p_{ij}

i

j

Problem: target is ofen **unbalanced**.
E.g. much more background then membrane pixels!
Cross Entropy will not work well

Target t_{ij}

j

Solutions for imbalanced target

- Weight the (Binary) Cross Entropy with class frequencies.
  - How to weight exactly can be tricky.

- **Use loss function that is not sensitive to imbalance: Dice Coefficient**

$$\mathrm{DICE} = 1 - \frac{2\sum_{n}^{N} t_n p_n}{\sum_{n}^{N} t_n^2 + \sum_{n}^{N} p_n^2}$$

Solutions for imbalanced target

- Weight the (Binary) Cross Entropy with class frequencies.
  - How to weight exactly can be tricky.

- **Use loss function that is not sensitive to imbalance: Dice Coefficient**

$$\mathrm{DICE} = 1 - \frac{\boxed{2 \sum_n^N t_n p_n}}{\boxed{\sum_n^N t_n^2 + \sum_n^N p_n^2}}$$

Intersection of target and prediction

Union of target and prediction (the denominator makes it insensitive to class imbalance!)

Solutions for imbalanced target

- Weight the (Binary) Cross Entropy with class frequencies.
  - How to weight exactly can be tricky.

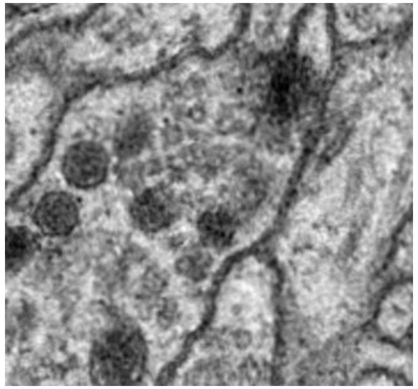
- **Use loss function that is not sensitive to imbalance: Dice Coefficient**

$$\mathrm{DICE} = \boxed{1 -} \frac{2 \sum_n^N t_n p_n}{\sum_n^N t_n^2 + \sum_n^N p_n^2}$$

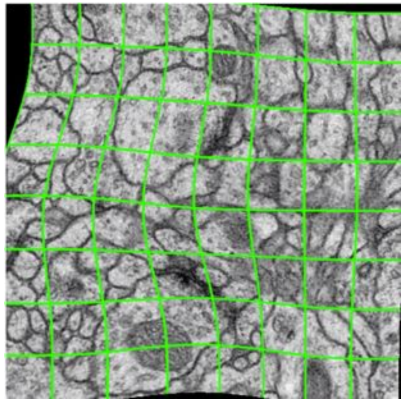
"1 - " because low values must correspond to good solution

U-net: training

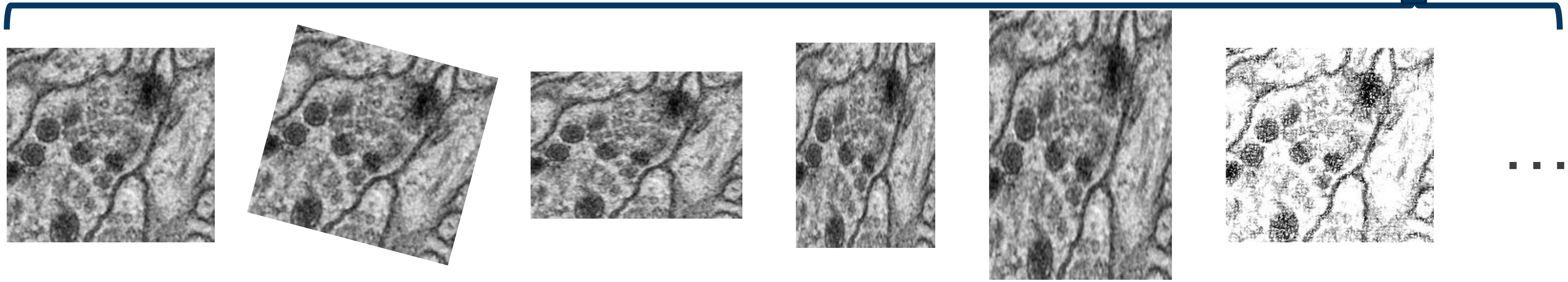
Bioimage datasets are often small!



Elastic image deformations



For best results: use data augmentation!

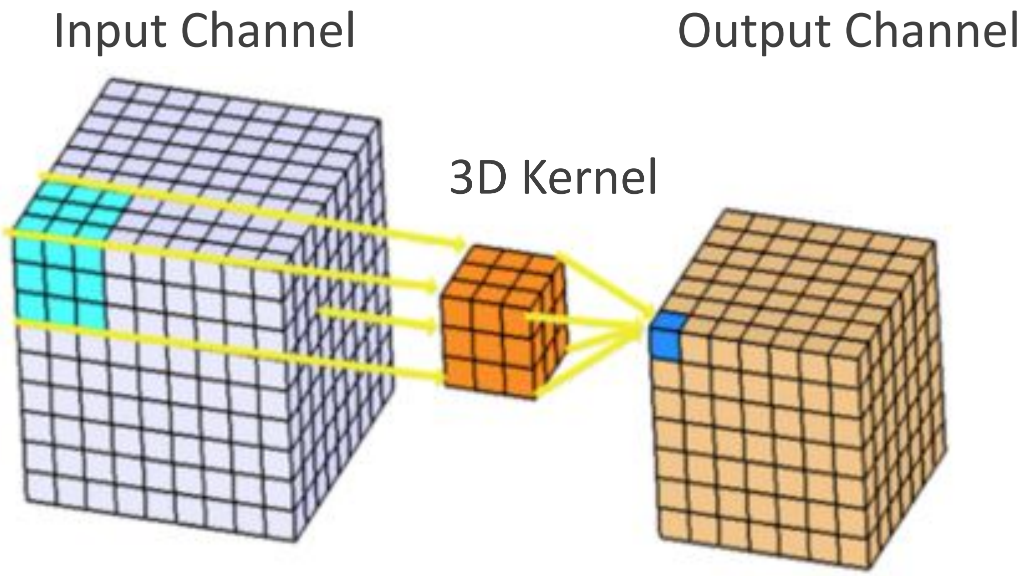


Volumetric data? Use a 3D U-Net!

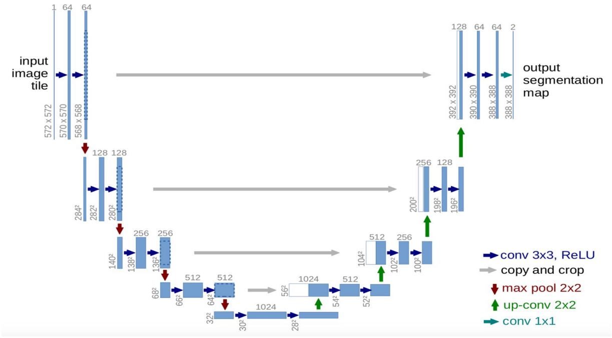
We have 3d versions for all layers used in U-Net.

- 2d convolution -> 3d convolution
  - 2d kernel *X x Y* -> 3d kernel *X x Y x Z*

- Needs more computation,
  but otherwise the same

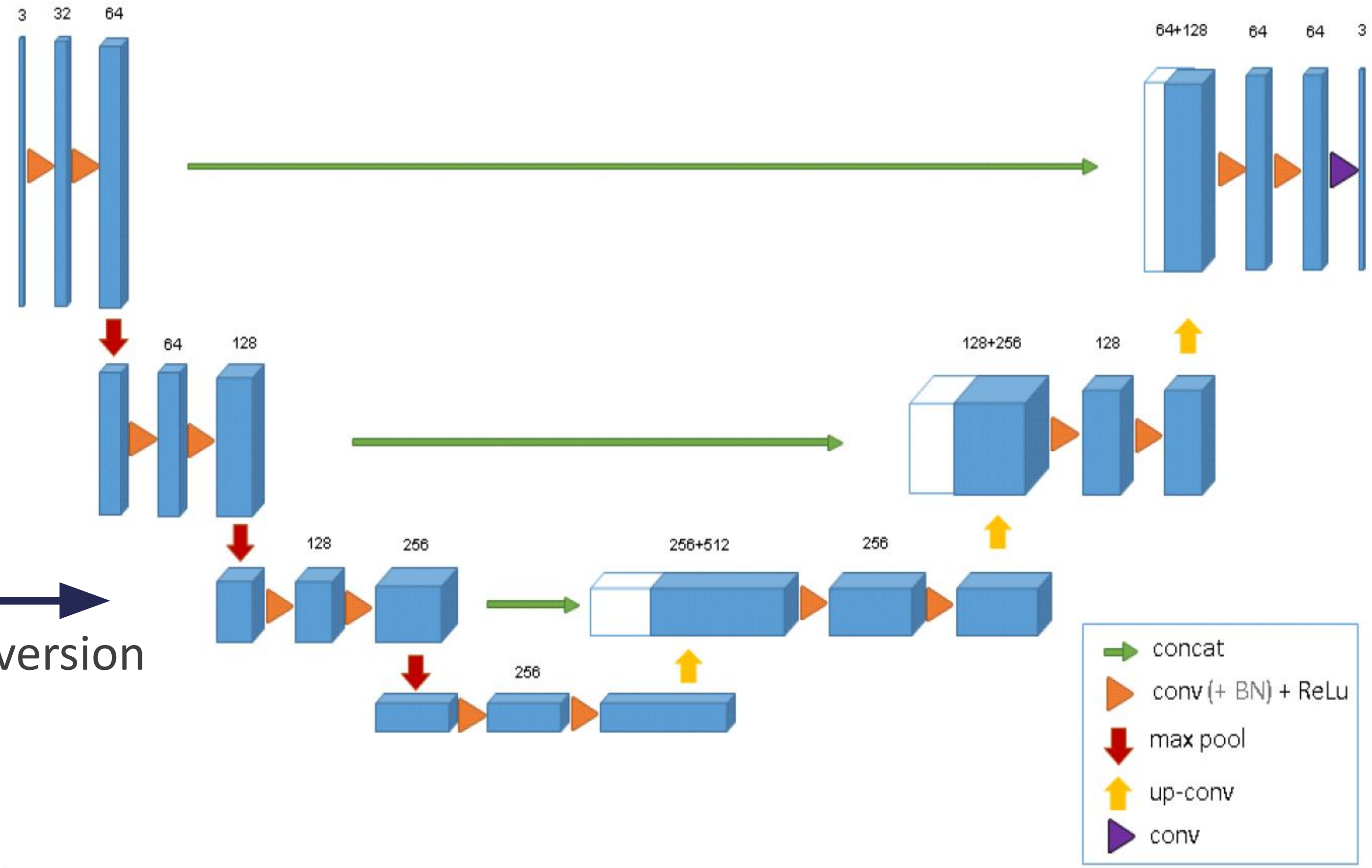


Volumetric data? Use a 3D U-Net!



Replace all

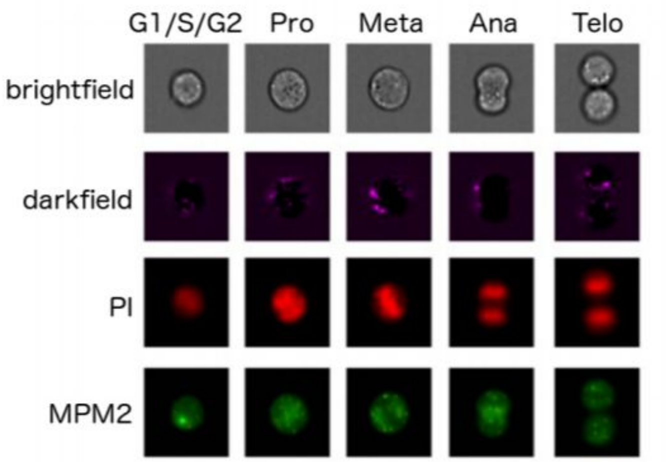
layers with 3d version



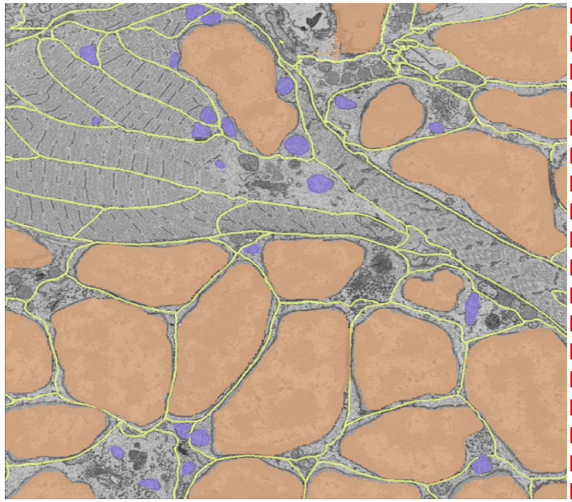
Instance Segmentation

Segmentation for biomedical images

Image Classification



Semantic Segmentation

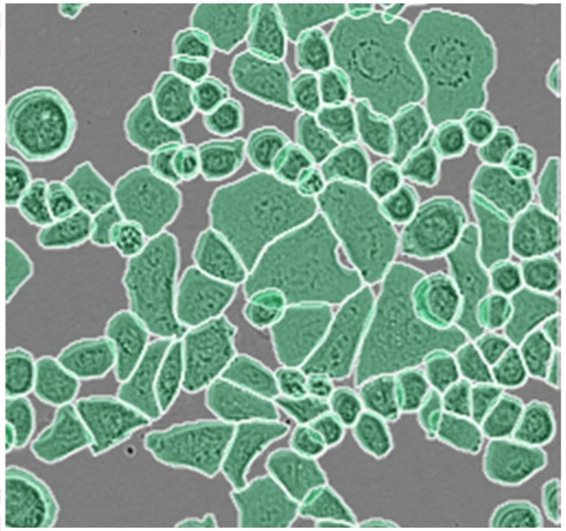


Cell Membrane
Nuclei
Mitochondria

Object Detection

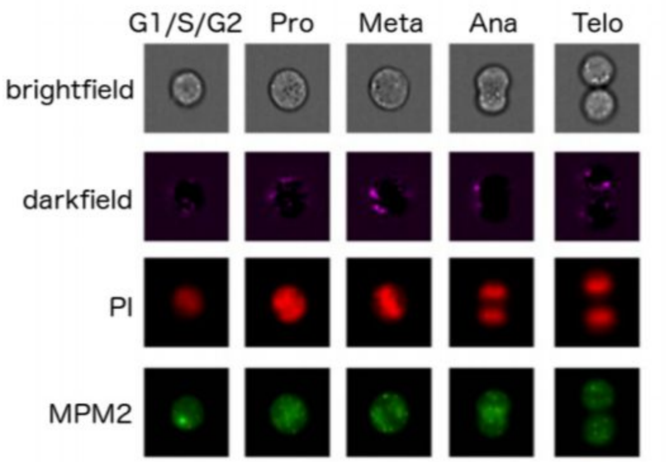
Object detection = find bounding boxes for each object -> **we skip this.**

Instance Segmentation

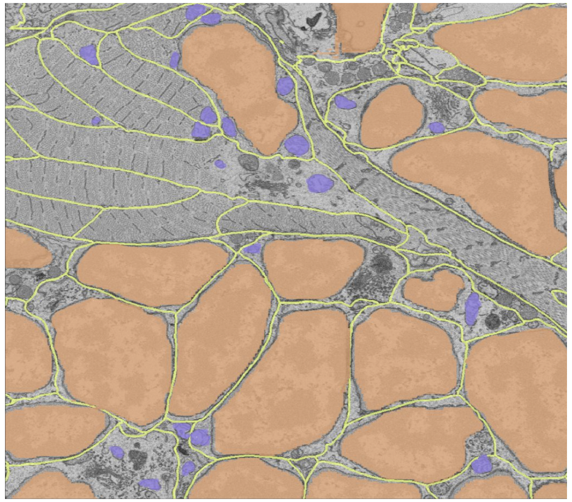


Segmentation for biomedical images

Image Classification

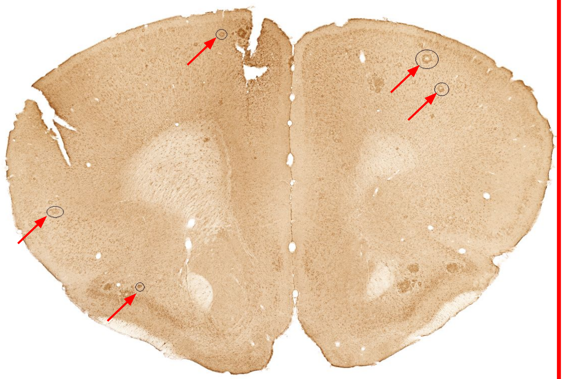


Semantic Segmentation

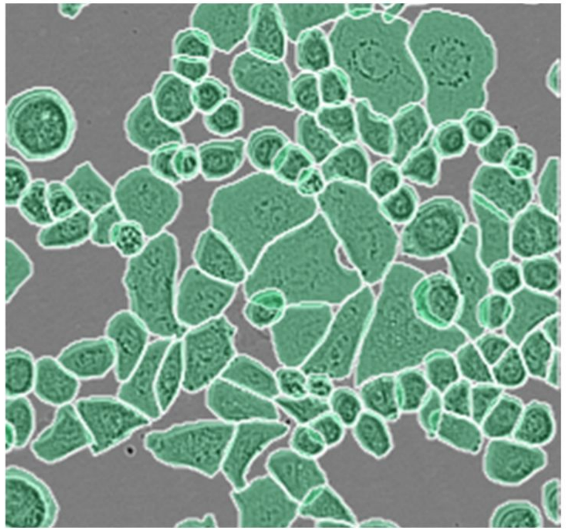


Cell Membrane
Nuclei
Mitochondria

Object Detection

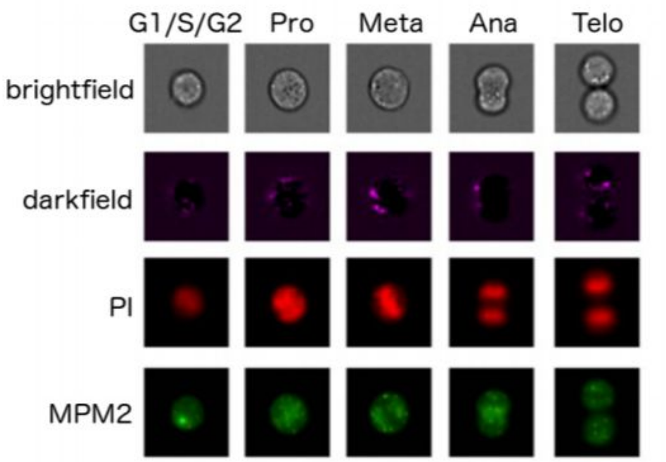


Instance Segmentation

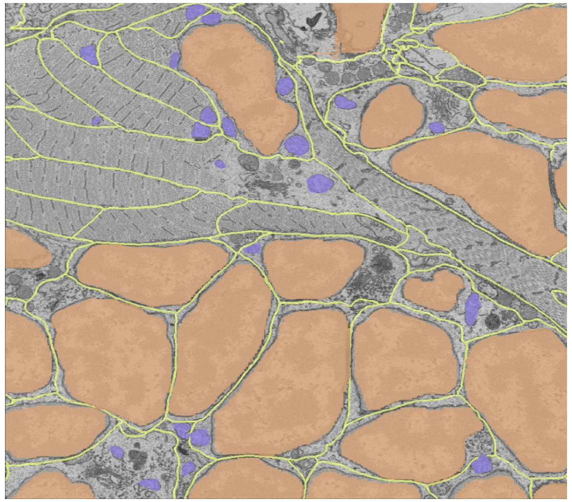


Segmentation for biomedical images

Image Classification

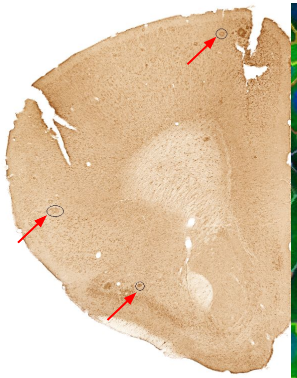


Semantic Segmentation

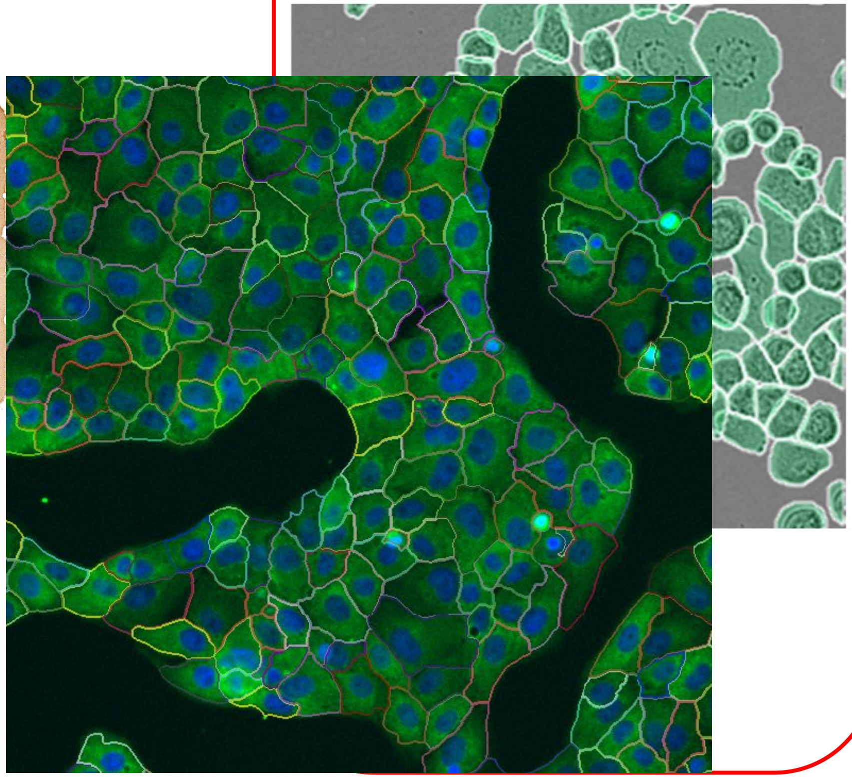


Cell Membrane
Nuclei
Mitochondria

Object Detection



Instance Segmentation

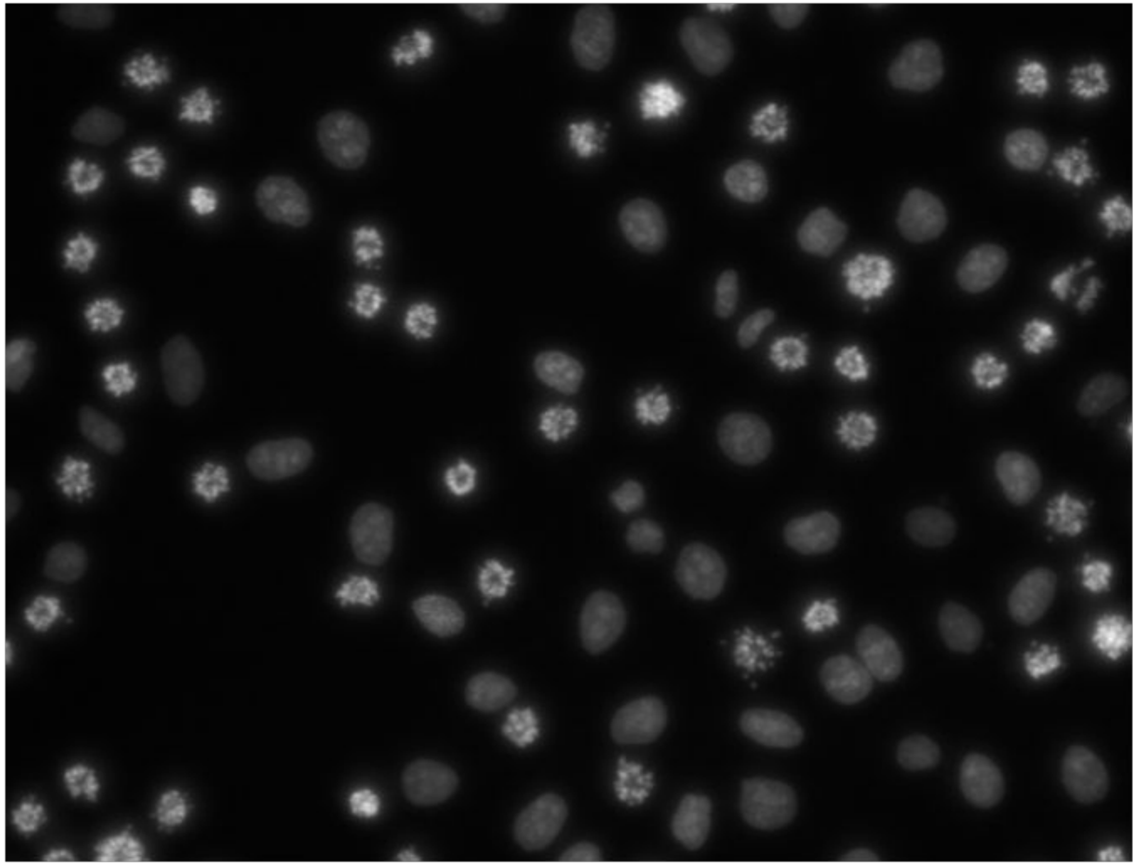


Instance segmentation

Goal: Segment individual nuclei in the image (1 mask per object)

Problem: How many objects are there?
We don't know before solving the task!

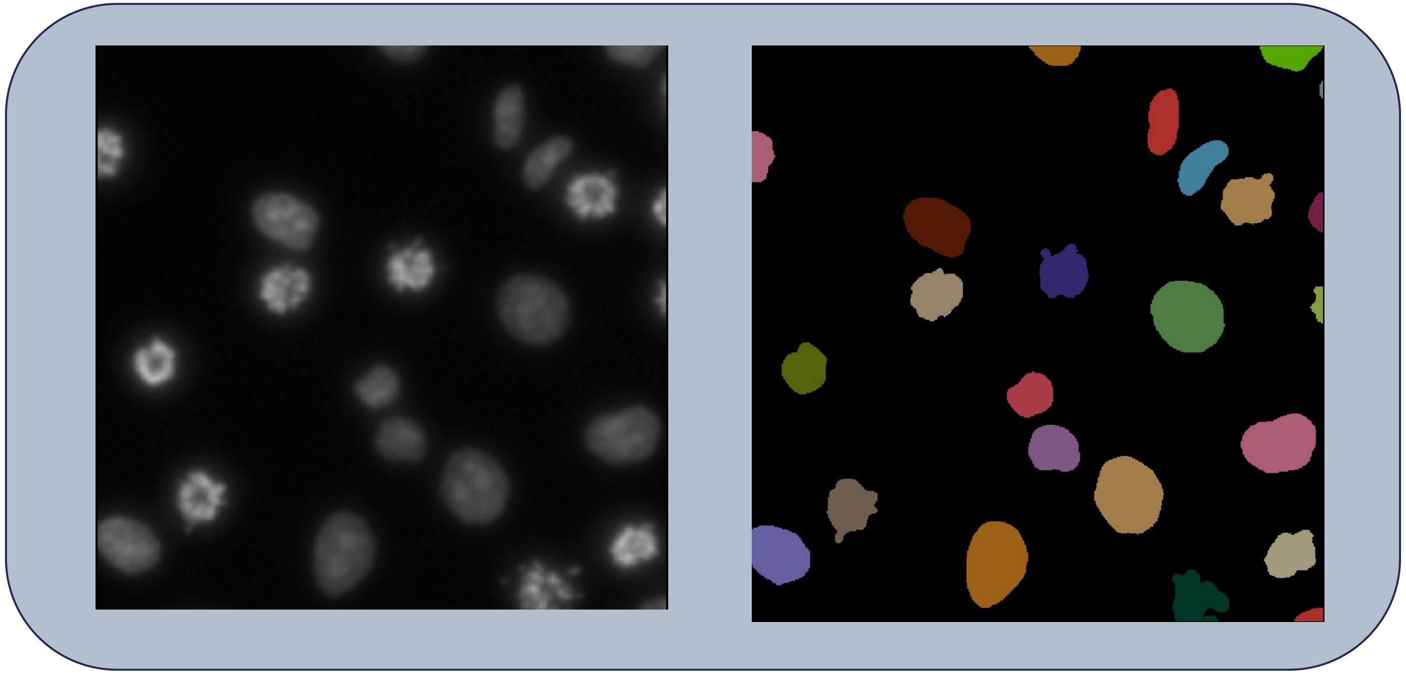
Different in semantic segmentation:
we know how many different categories
to predict in the beginning.



Instance segmentation

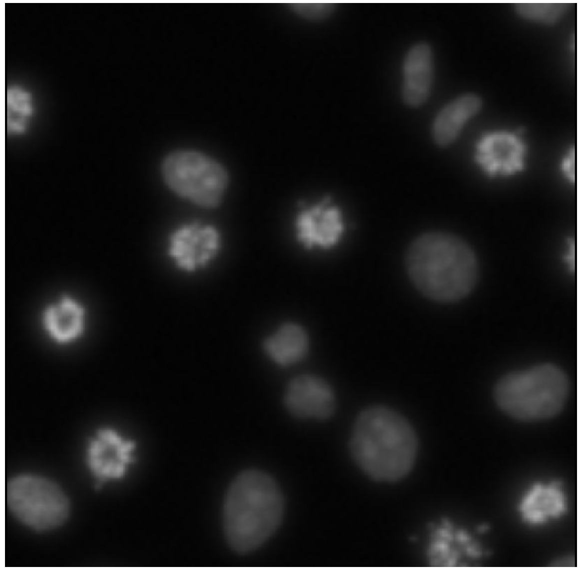
- Each instance has a unique id, all pixels belonging to instance have that id
  - Background is 0
- Visualization: random colors -> each id gets a random color

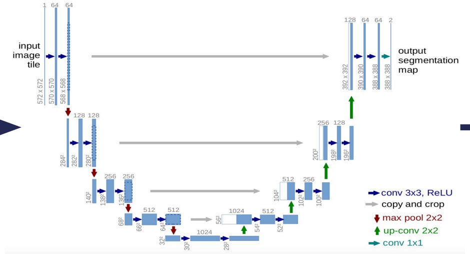
```
0000000000000000000000
0000000111111111100000
0000001111111111100000
0000001111111122222222
0000000000000000222222
0033333330000000022200
0000333333330000002000
0000003333000000000000
0000000333000000000000
```



Instance segmentation

Predict instances directly?





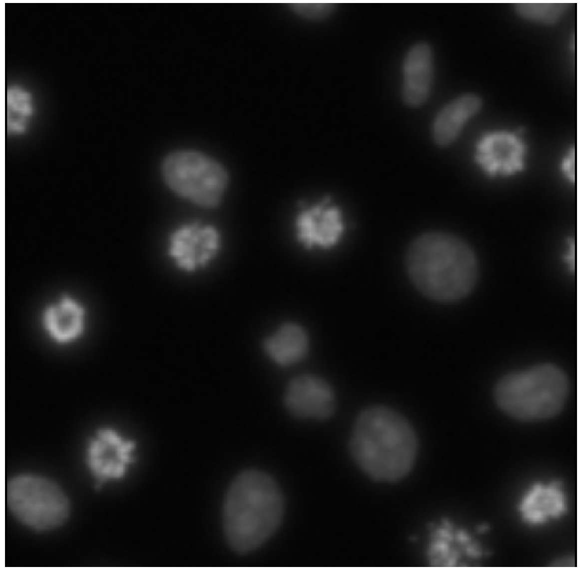
Instance segmentation

Problem: instance segmentation is invariant under relabeling:
Swapping instance labels *1* and *2* does not change the meaning!

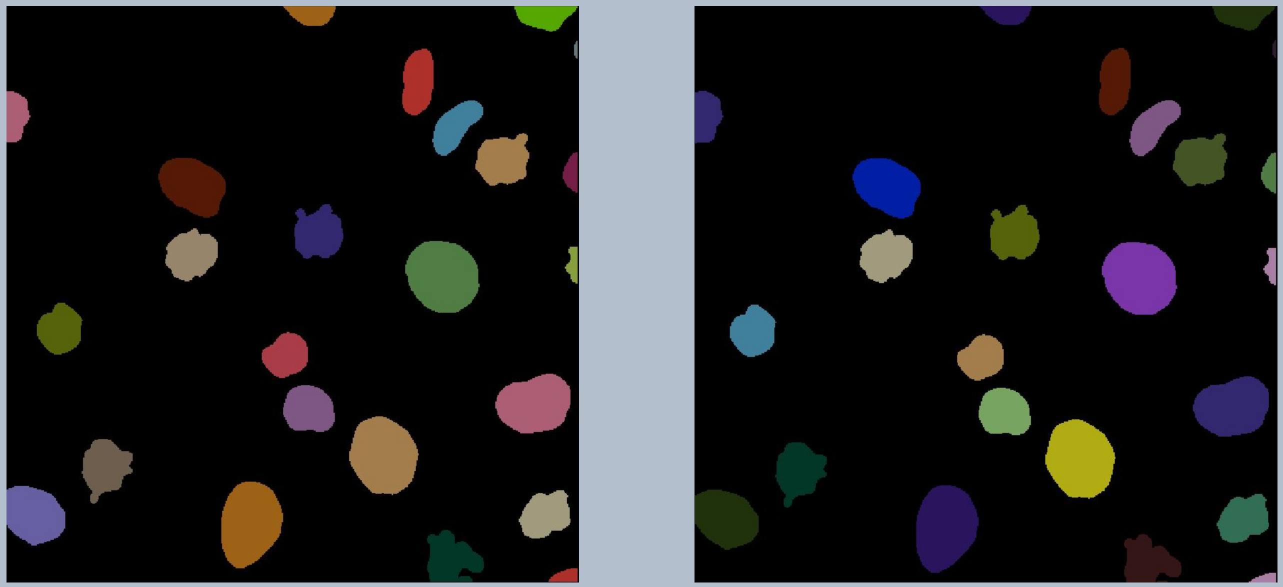
0000000000000000000000 0000000111111111100000 0000001111111111100000 0000001111111122222222 0000000000000000222222 0033333330000000022200 0000333333330000002000 0000003333000000000000 0000000333000000000000	Swap IDs 1, 2 ⟷	0000000000000000000000 0000000222222222200000 0000002222222222200000 0000002222222211111111 0000000000000000111111 0033333330000000011100 0000333333330000001000 0000003333000000000000 0000000333000000000000

Instance segmentation

Problem: instance segmentation is invariant under relabeling:
Swapping instance labels *1* and *2* does not change the meaning!



Equivalent instance segmentations!

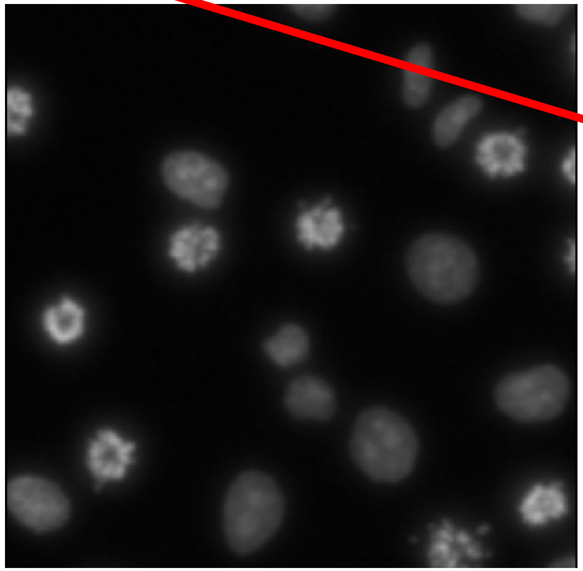


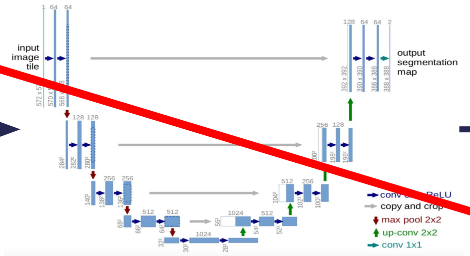
*On-going research: differentiable instance segmentation

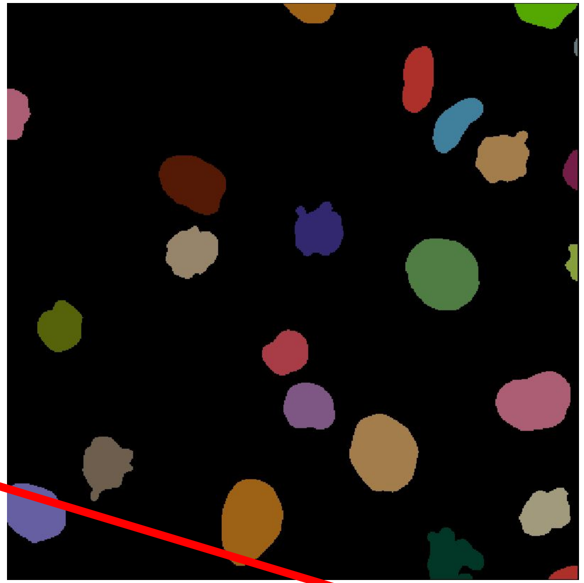
Instance segmentation

Predict instances directly?

Not possible due to relabeling invariance, can’t formulate a differentiable loss*

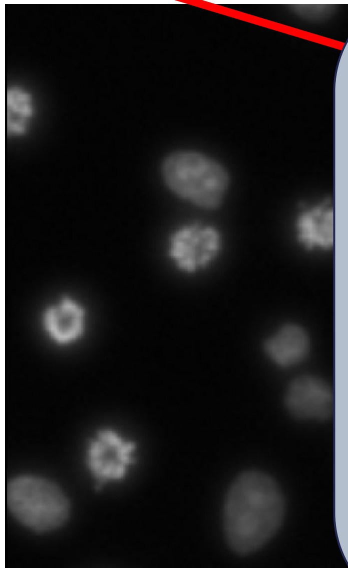






Instance segmentation

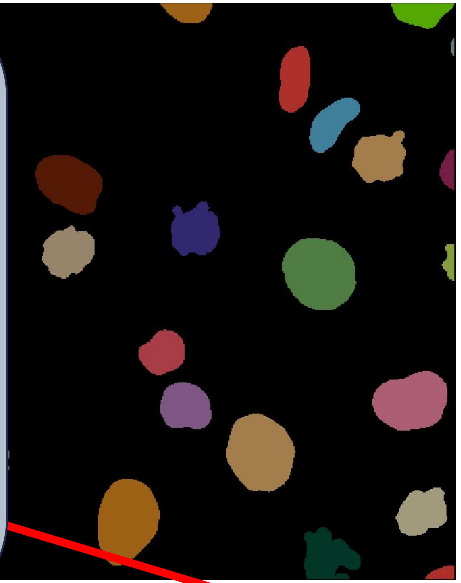
Predict instances directly?
Not possible due to relabeling invariance, can't formulate a differentiable loss



Instead:

- Semantic segmentation with deep learning (or regression)
- Post-processing to get the objects

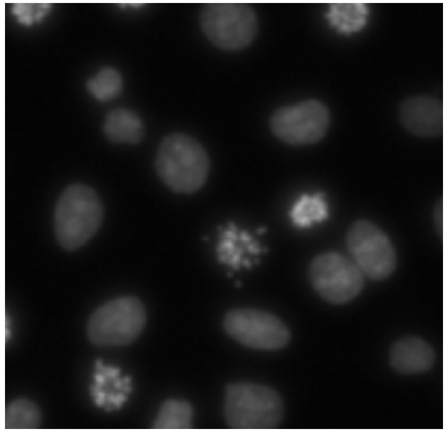
Many different approaches, will discuss three now.

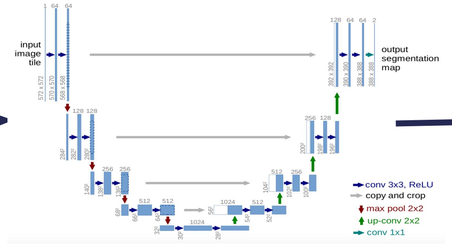


Foreground prediction

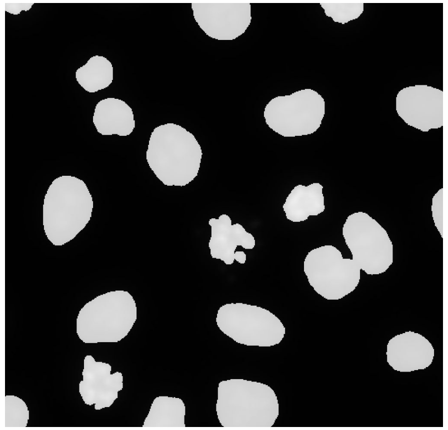
Predict foreground (binary segmentation)

Input



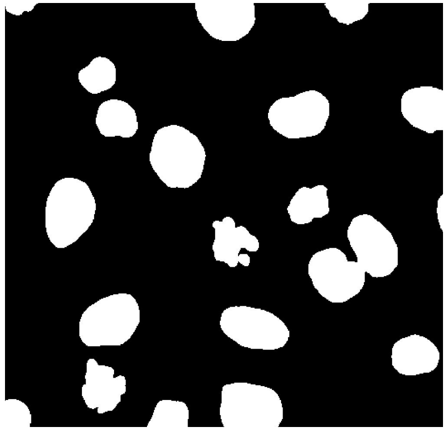


Prediction



Loss

Target

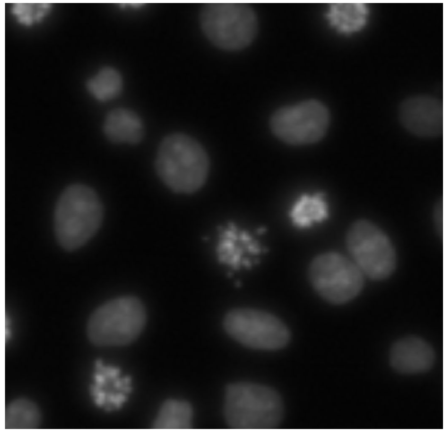


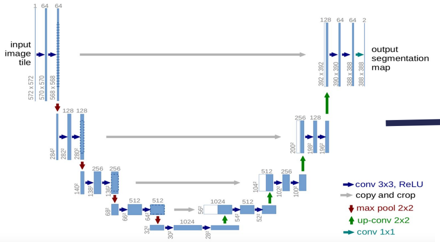
Training

Foreground prediction

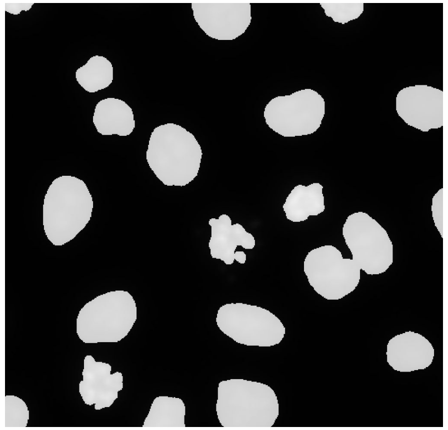
Predict foreground (binary segmentation), apply connected components

Input





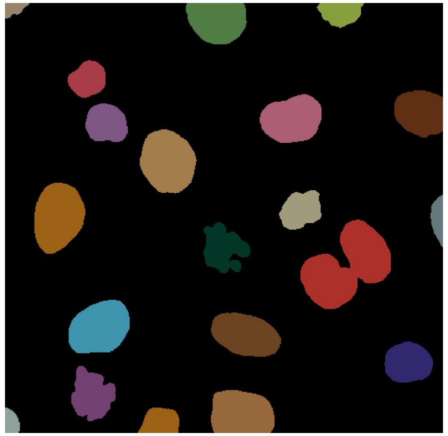
Prediction



Threshold

Connected Comp.

Instance Seg.

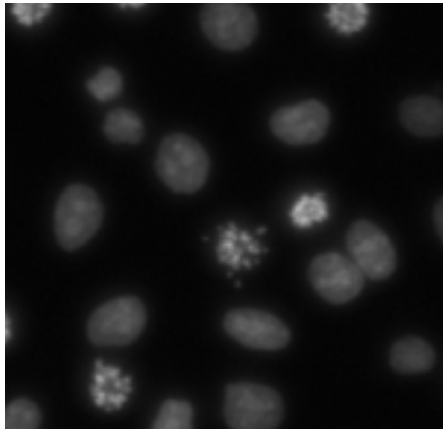


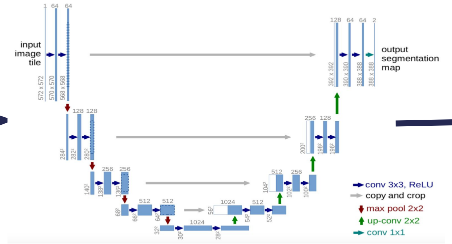
Prediction

Foreground prediction

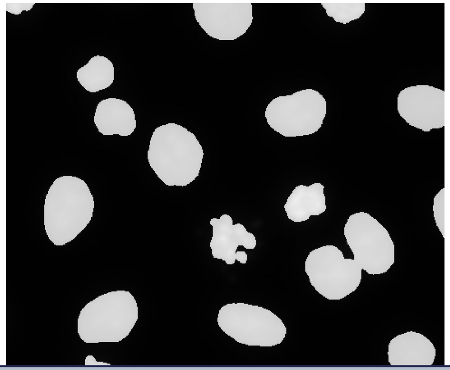
Predict foreground (binary segmentation), apply connected components

Input





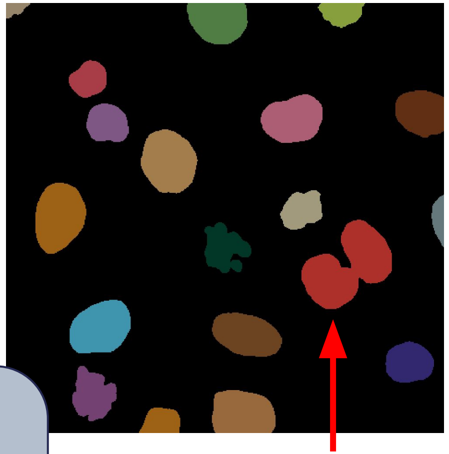
Prediction



Threshold

Connected Comp.

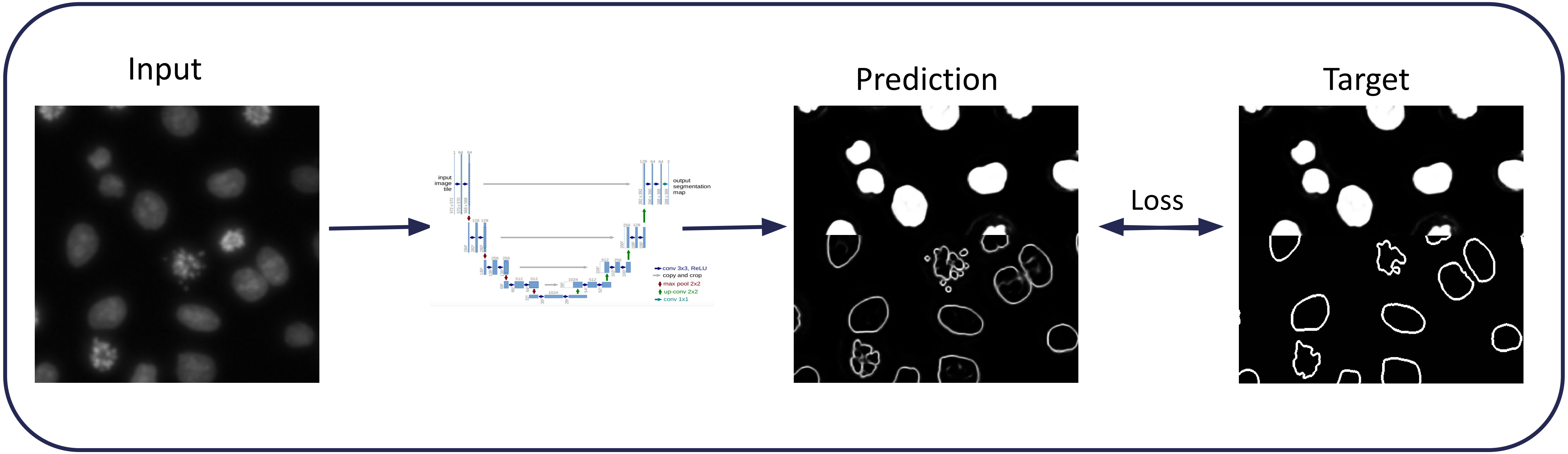
Instance Seg.



Problem: touching objects!

Foreground and boundary prediction

Predict foreground and boundaries

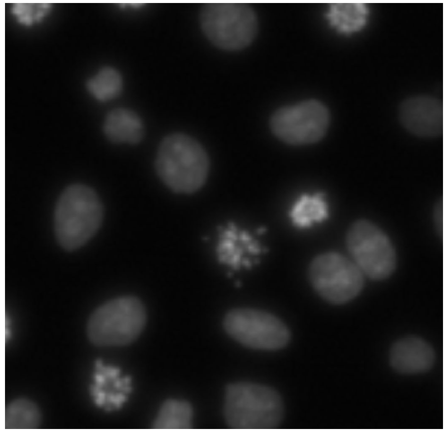


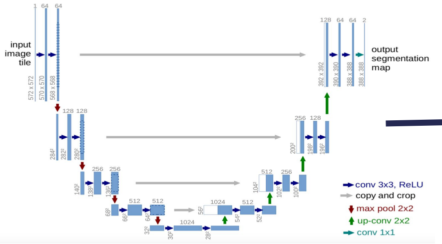
Training

Foreground and boundary prediction

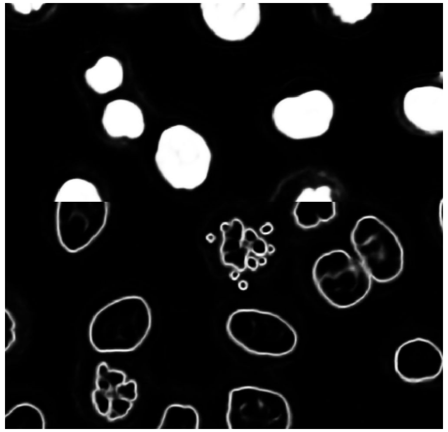
Subtract boundaries from foreground before threshold, apply watershed to get back full size

Input





Prediction



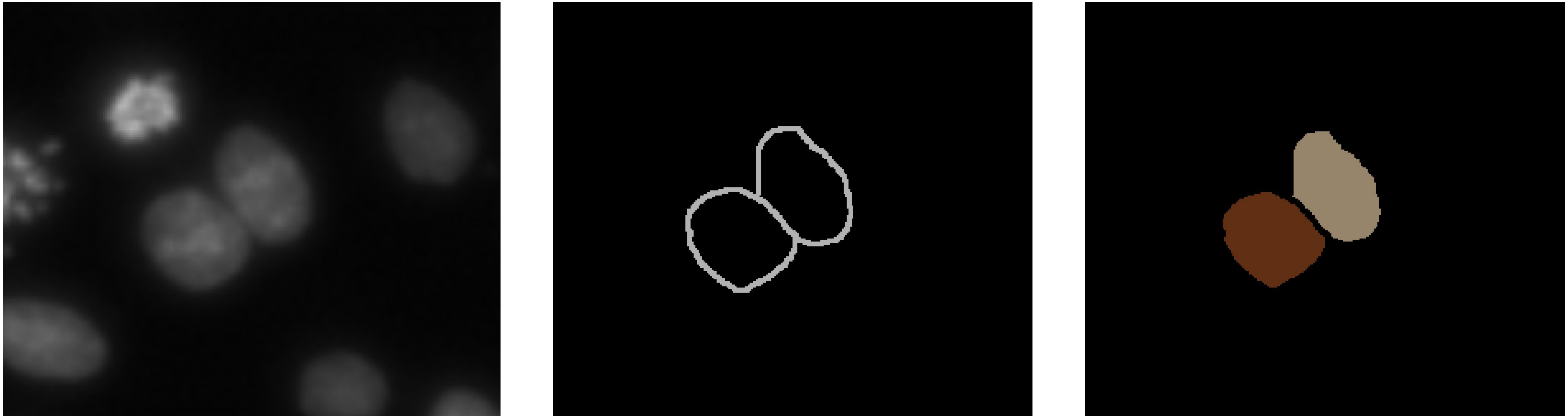
Instance Seg.

Prediction

Foreground and boundary prediction

So we can just predict foreground and boundaries and solve any segmentation problem?

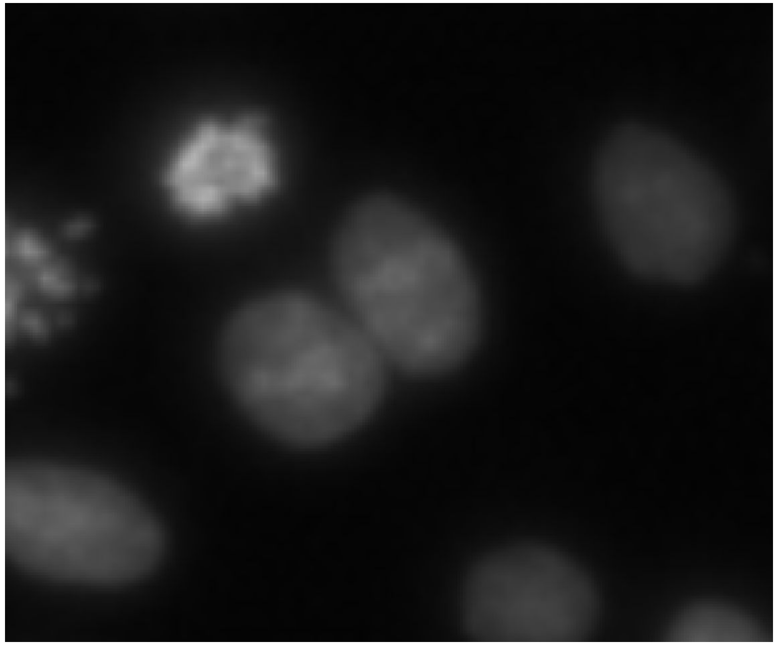
Correct boundary prediction, correct segmentation.

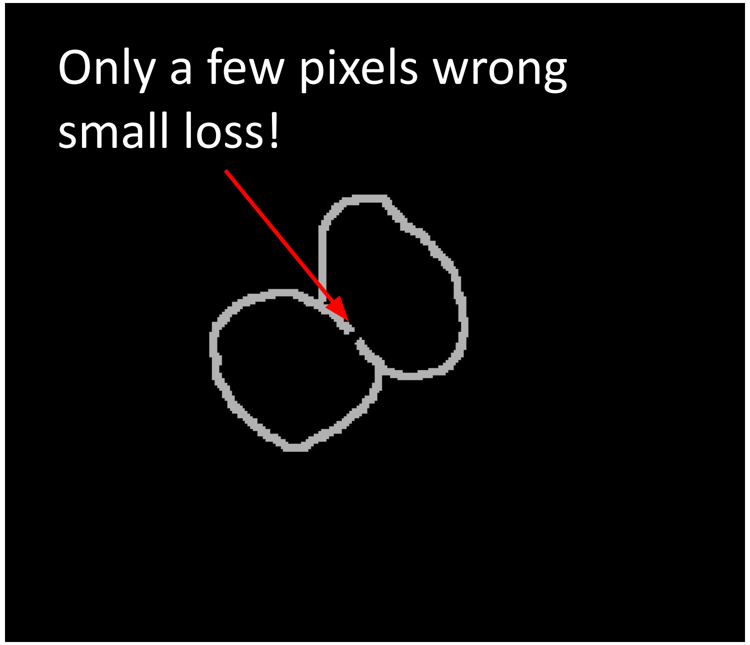


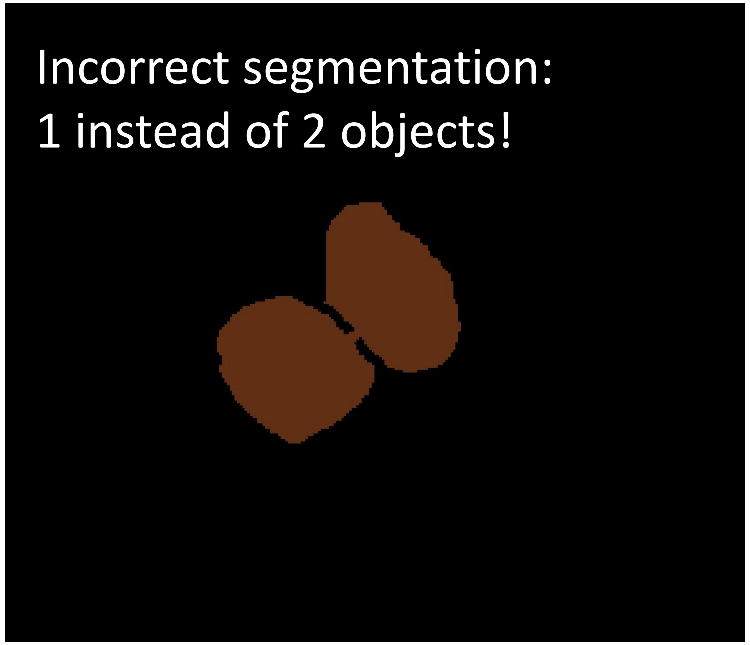
Foreground and boundary prediction

So we can just predict foreground and boundaries and solve any segmentation problem?

Small error in boundary prediction, very incorrect segmentation.



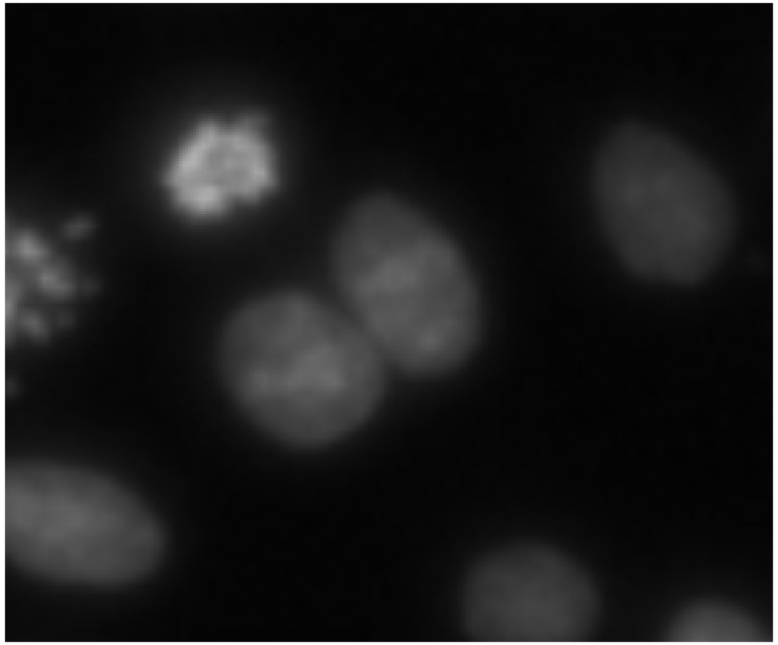


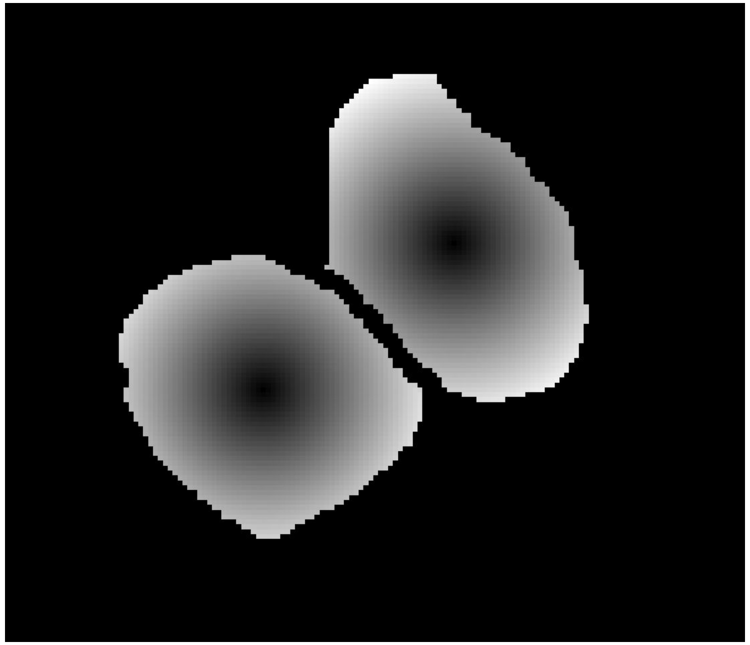


Foreground and boundary prediction

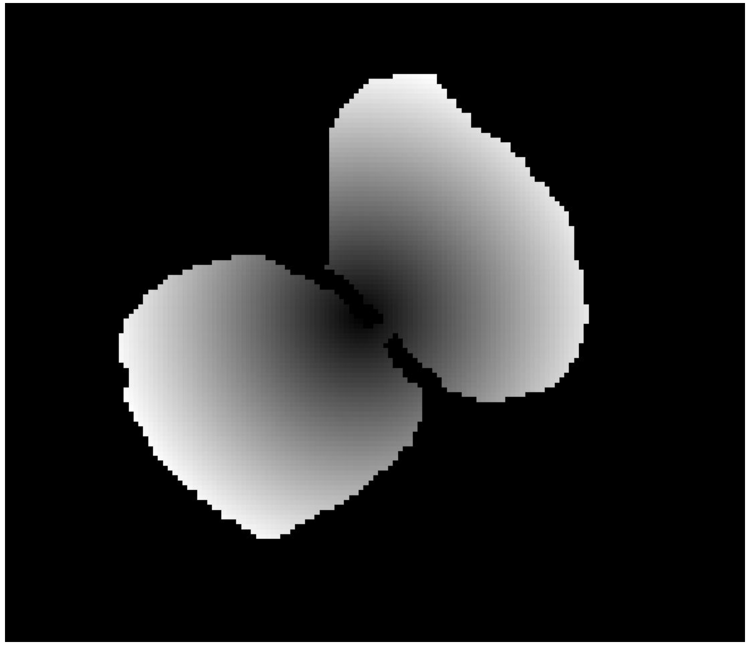
Instead: predict representation with network that changes a lot when segmentation changes.

Example: predict distance to center point.





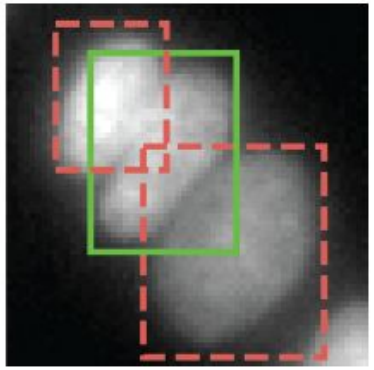
Distance to center (correct)



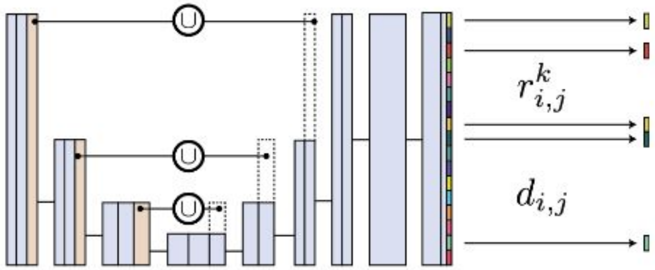
Distance to center (incorrect)

StarDist: Popular for Nucleus Segmentation

Similar idea: predict distance to center along different direction for each pixel (and foreground)

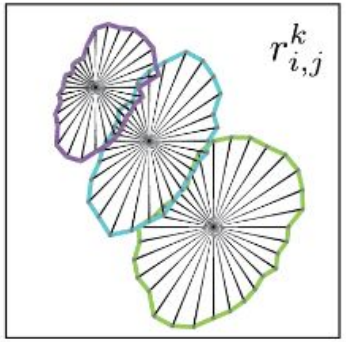


Input



Network
(U-Net)

Prediction:



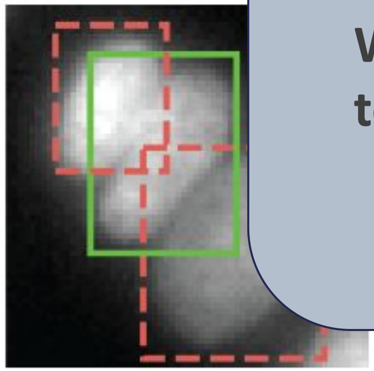
Center Distances

Foreground

Post-processing for instance segmentation

StarDist: Popular for Nucleus Segmentation

Similar idea: predict distance to center along different direction for each pixel (and foreground)



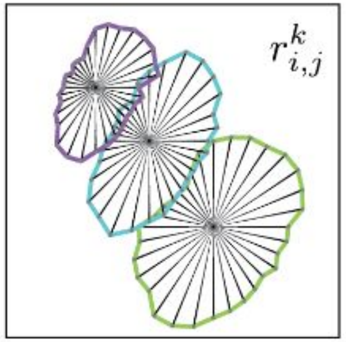
Other segmentation methods are based on similar ideas.

Will give an overview of other methods and tools for instance segmentation tomorrow!

Input

Network
(U-Net)

Prediction:



Center Distances

Foreground

Post-processing for instance segmentation

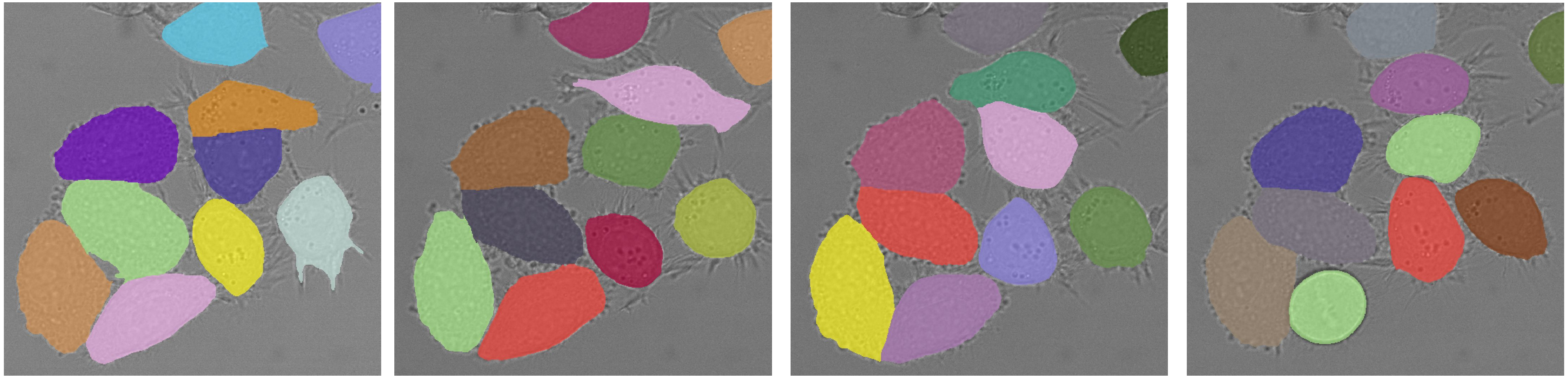
Cell Tracking

Following cells over time

Tracking by detection

Cell tracking: follow cells *and* cell divisions over time to build cell lineage graph.

Tracking by detection: first segment individual cells per frame

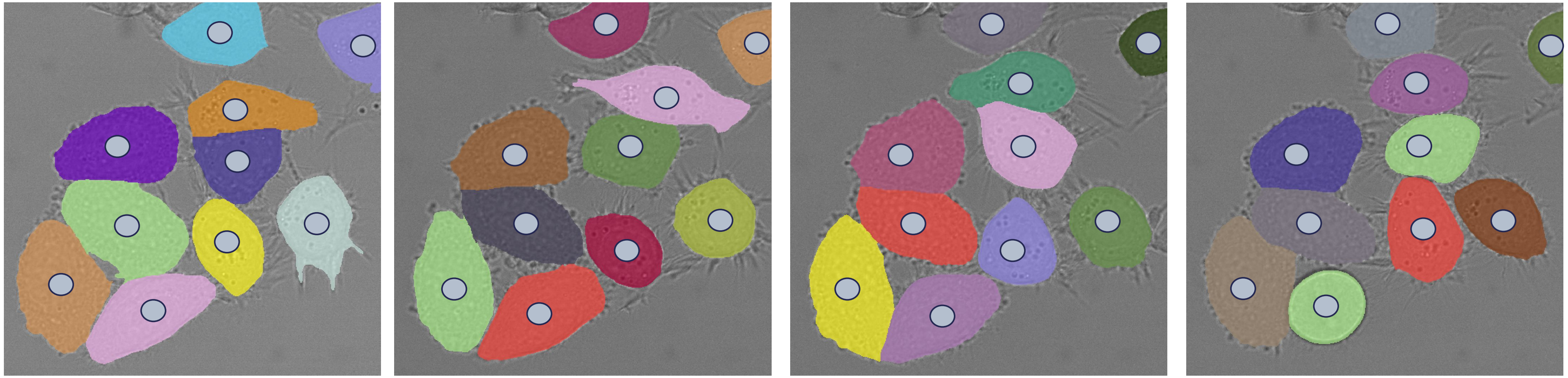


t = 0 t = 1 t = 2 t = 3

Tracking by detection

Cell tracking: follow cells *and* cell divisions over time to build cell lineage graph.

Tracking by detection: build a graph with cells as nodes possible connections (= same cell) across frames as edges

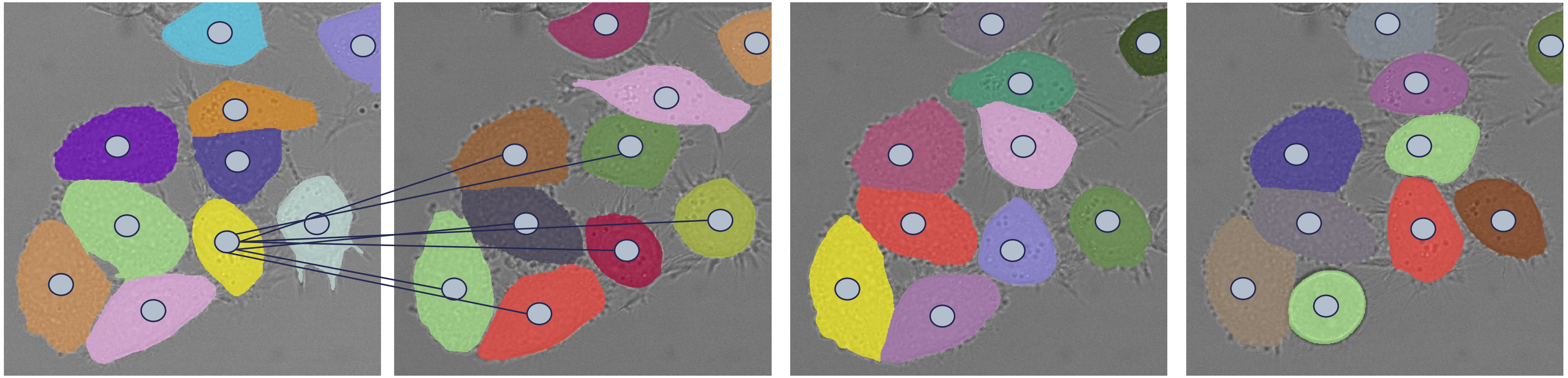


t = 0 t = 1 t = 2 t = 3

Tracking by detection

Cell tracking: follow cells *and* cell divisions over time to build cell lineage graph.

Tracking by detection: build a graph with cells as nodes possible connections (= same cell) across frames as edges

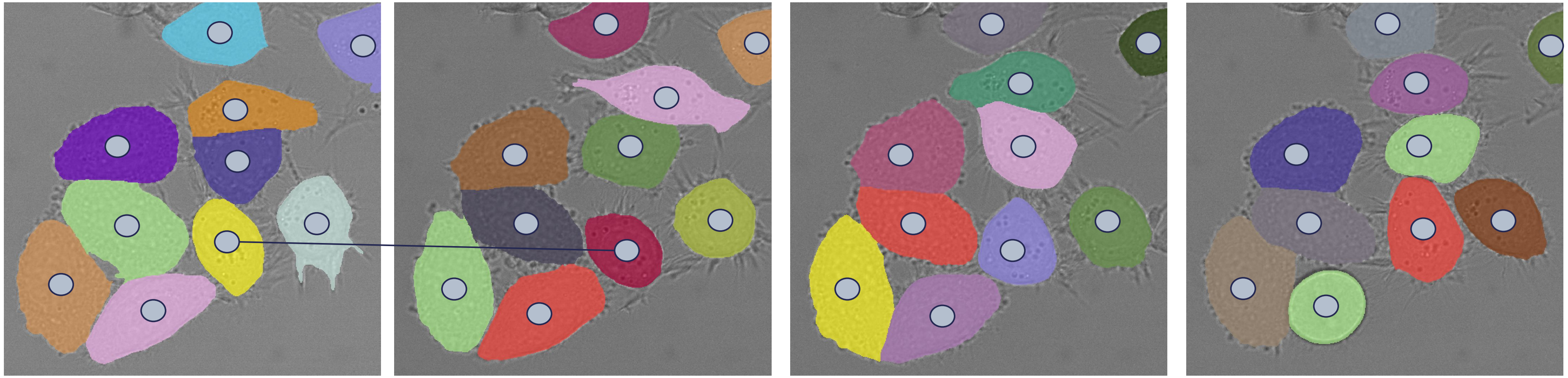


t = 0 t = 1 t = 2 t = 3

Tracking by detection

Cell tracking: follow cells *and* cell divisions over time to build cell lineage graph.

Tracking by detection: select the most likely edges / connections given the constraints that a cell may only link to one or two cells (cell division) in the next frame

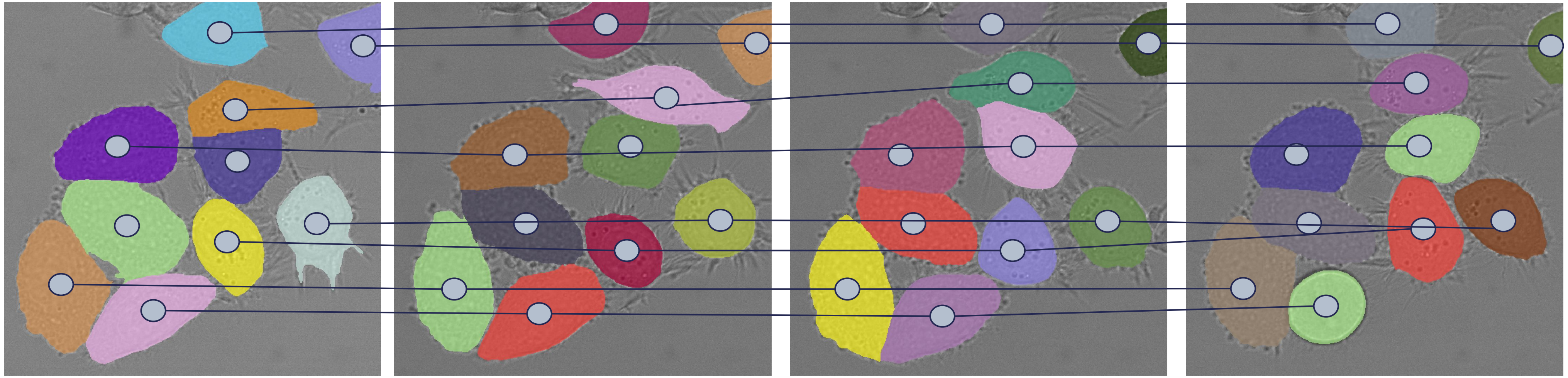


t = 0 t = 1 t = 2 t = 3

Tracking by detection

Cell tracking: follow cells *and* cell divisions over time to build cell lineage graph.

Tracking by detection: select the most likely edges / connections given the constraints that a cell may only link to one or two cells (cell division) in the next frame **for all edges** (shown for a subset here)



t = 0 t = 1 t = 2 t = 3


layer controls
olor by: track_id
olormap: turbo
lending: additive
pacity: 1,00
ail width:
ail length:
ead length:
ail:
how ID:
raph:
layer list
Tracks
lineage_segmentation
segmentation
images
ID:12
ID:11
ID:9
ID:10
ID:2
ID:1
ID:4
ID:3
ID:8
ID:7
ID:5
ID:6
acks [0 1 0]: 12

Tracking by detection – How?

"Classical" approach: Select edges according to:

- Largest mask overlap and/or smallest distance
- + motion model
- and tracking constraints

Implemented in:

- **TrackMate** (Fiji Plugin, https://www.nature.com/articles/s41592-022-01507-1)
- btrack (Python / napari plugin, https://github.com/quantumjot/btrack)
- motile (Python / napari plugin, https://github.com/funkelab/motile)

Tracking by detection – How?

Deep learning-based approach: Learn the likelihood of edge selection.

State-of-the-art: **Trackastra**

- Transformer-based architecture for cell tracking
- Pre-trained models can be applied directly to tracking problems
- Python / napari-plugin
- https://arxiv.org/abs/2405.15700

Pretrained models & Transfer learning

Re-using models

Training networks for a new task:

- requires a significant amount of annotated data (large manual effort)
- computational skills to train the network

Pre-trained networks: use network already trained on similar data!

Re-using models

Training networks for a new task:

- requires a significant amount of annotated data (large manual effort)
- computational skills to train the network

Pre-trained networks: use network already trained on similar data!

Tools like StarDist / CellPose / µSAM (more on this tomorrow):

- Provide networks for specific tasks (Nucleus / Cell Segmentation)
- What if my problem does not fit?

Re-using models

Training networks for a new task:

- requires a significant amount of annotated data (large manual effort)
- computational skills to train the network

Pre-trained networks: use network already trained on similar data!

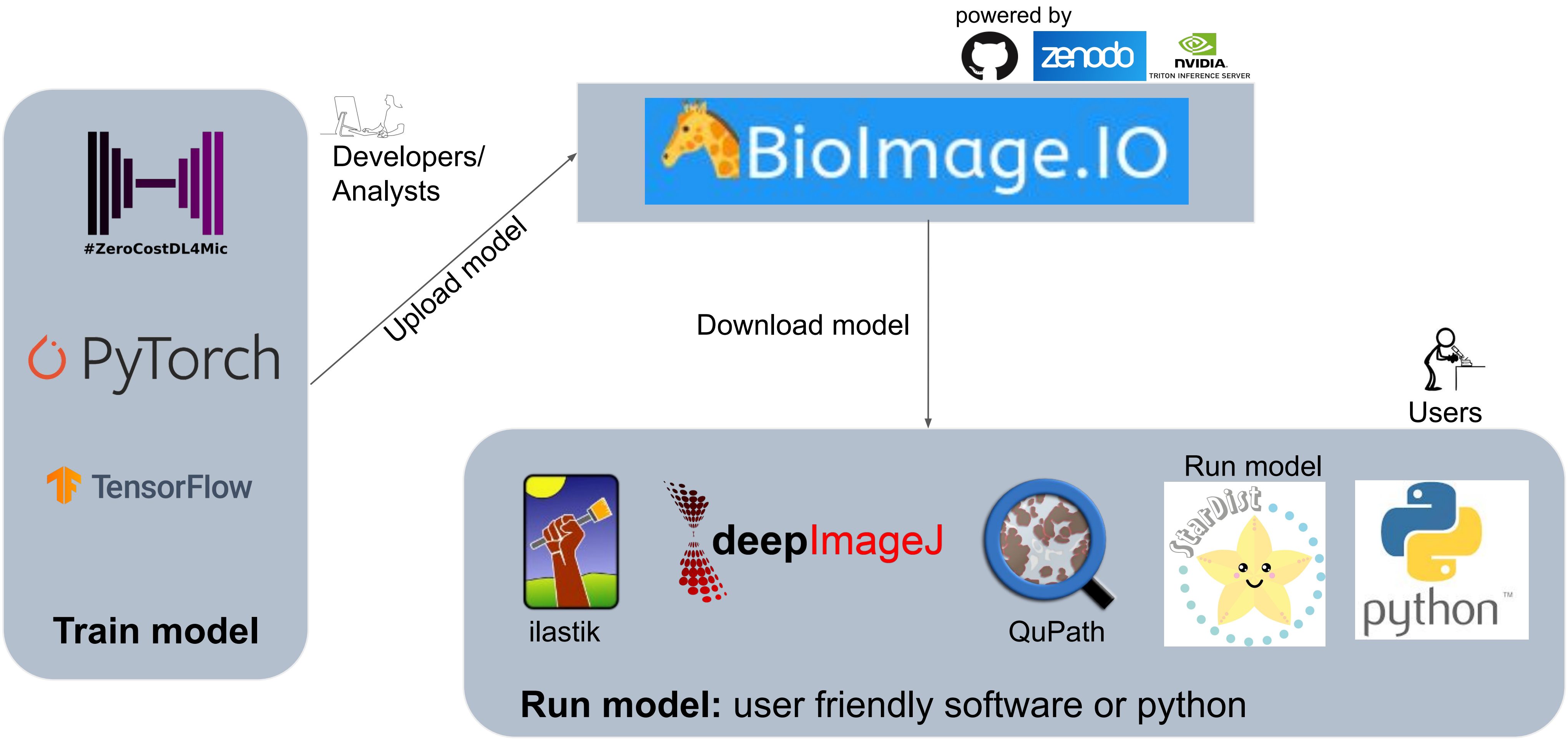
Tools like StarDist / CellPose / µSAM (more on this tomorrow):

- Provide networks for specific tasks (Nucleus / Cell Segmentation)
- What if my problem does not fit?

BioImage.IO: Ressource for sharing pre-trained models for bio-image analysis.

BioImage.IO

https://bioimage.io/#/models



Limits of pre-trained models

Pre-trained models often not good enough because of differences in training data!

Solution: Fine-tuning

- Annotate some of your data
- Update the model by training on that data

Transfer learning!

Transfer learning: the idea

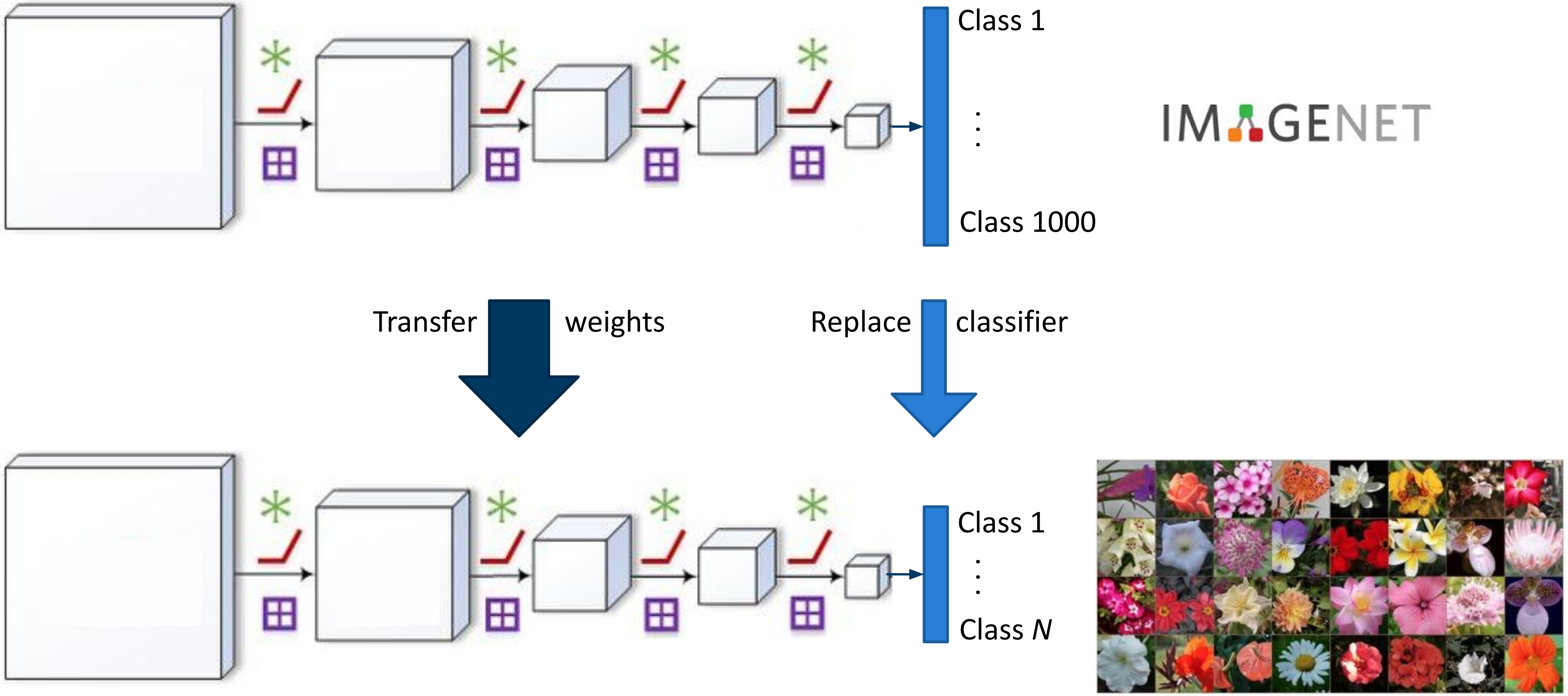
1

Pre-training
(massive data)

IMAGENET
1 million images
1000 classes

2

Fine-tuning
(much less data)



Recognize flowers

Transfer learning: the idea

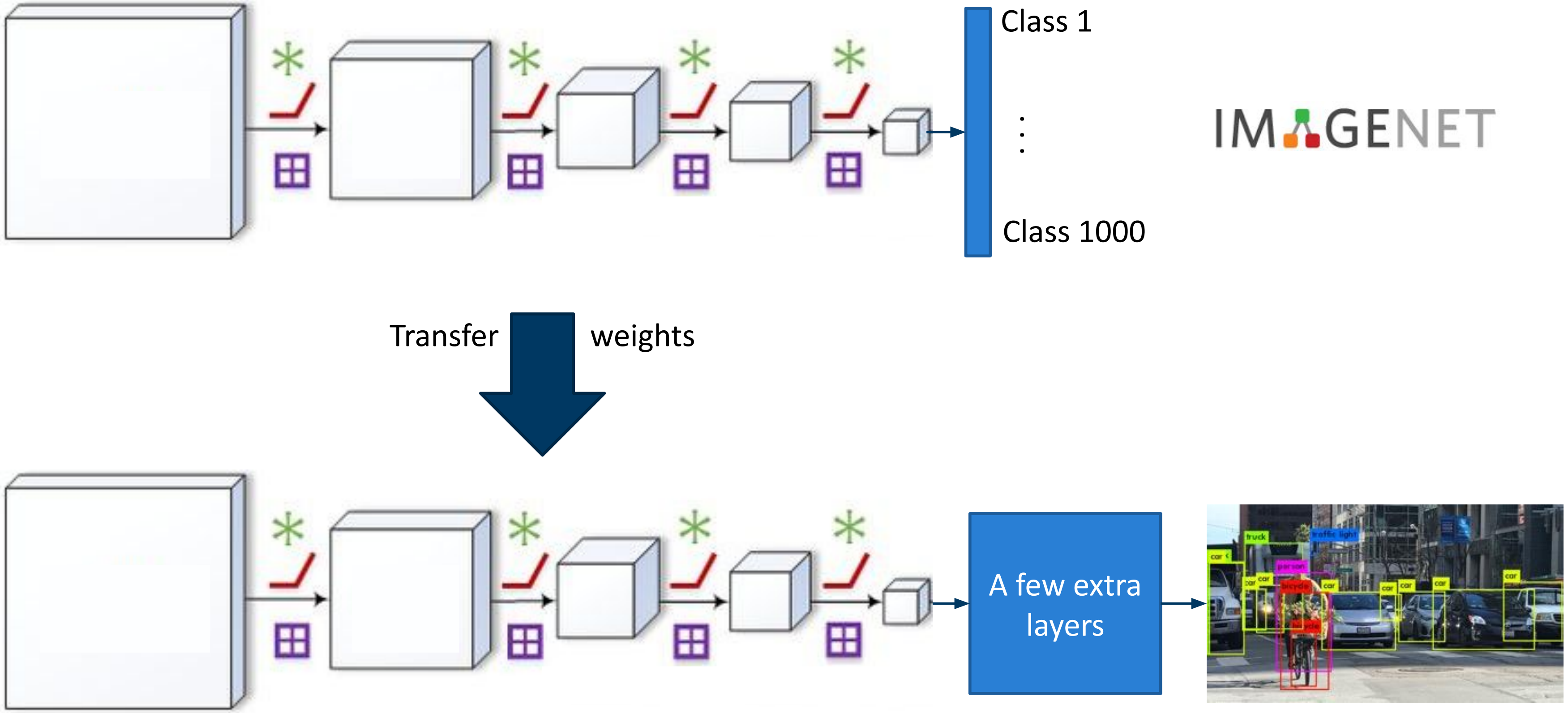
1

Pre-training
(massive data)

IMAGENET
1 million images
1000 classes

2

Fine-tuning
(much less data)



Object detection

Transfer learning for microscopy

But: natural images are quite different from microscopy data:

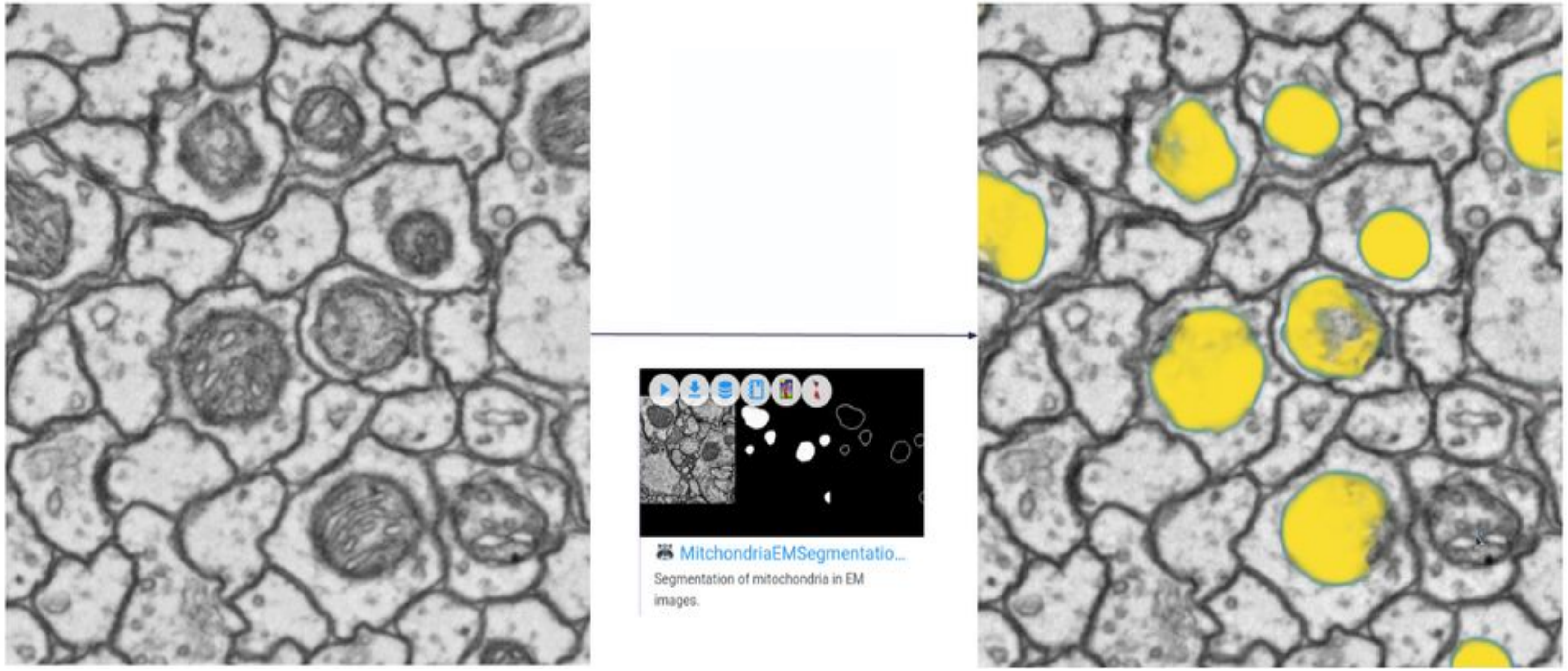
Benefits of ImageNet pretraining is limited*

≠

* not true anymore for newer, even bigger and more diverse pre-training datasets

Transfer learning for microscopy

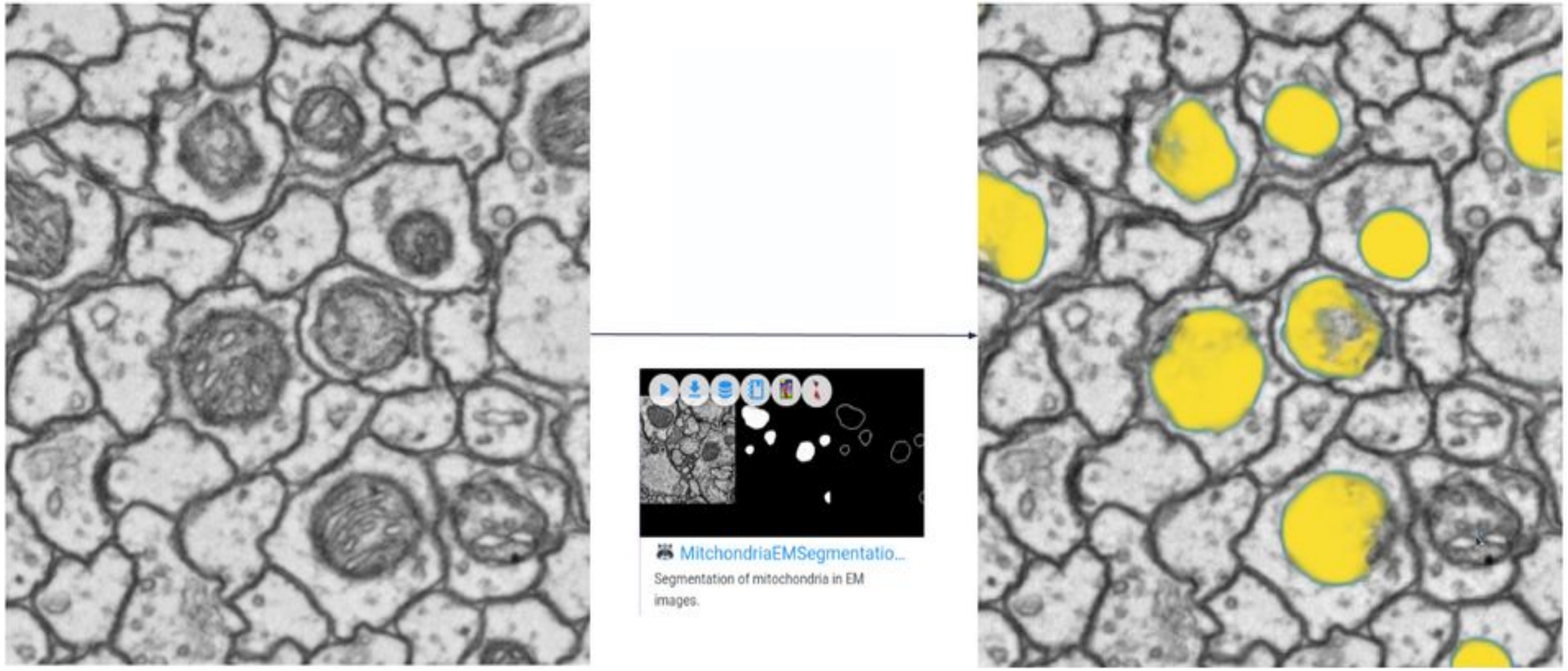
- Find the best model for your data from BioImage.IO
- Apply to your data, correct result
- Fine-tune the model on the corrected results



Transfer learning for microscopy

- Find the best model for your data from BioImage.IO
- Apply to your data, correct result
- Fine-tune the model on the corrected results

Limitations:
- No unified tool for fine-tuning
- Different architectures and frameworks (PyTorch & Tensorflow)
- **Custom python code depending no the model needed**



Deep Learning for Image Restoration

Denoising & Other Tasks

Image-to-image tasks

Many analysis tasks: transform image to a different version:

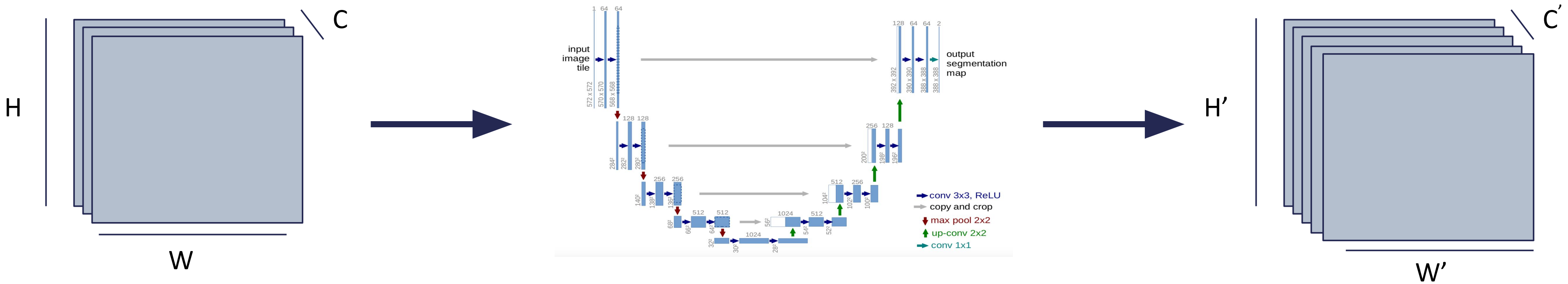
- May change spatial dimension: **H x W** -> **H’ x W’**
- May change channels: **C** -> **C’**

Image-to-image tasks

Many analysis tasks: transform image to a different version:

- May change spatial dimension: **H x W -> H’ x W’**
- May change channels: **C -> C’**

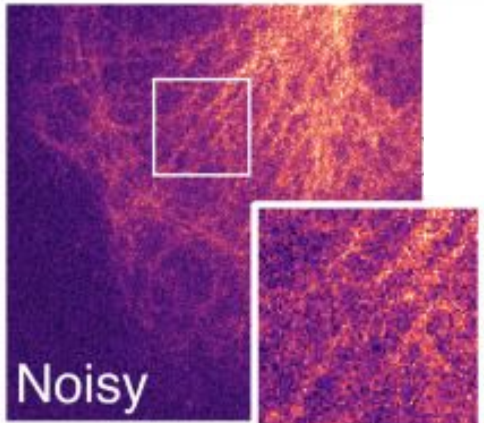
We can use a U-Net for this!

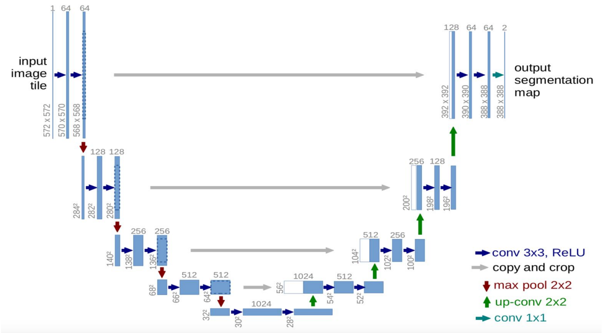


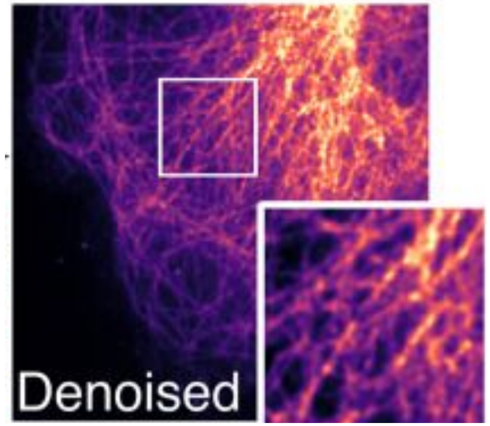
Denoising

Most common image restoration task: Denoising

- Spatial dimensions and channels stay the same
- Improve signal in the image







Denoising: CARE

https://www.biorxiv.org/content/10.1101/236463v5

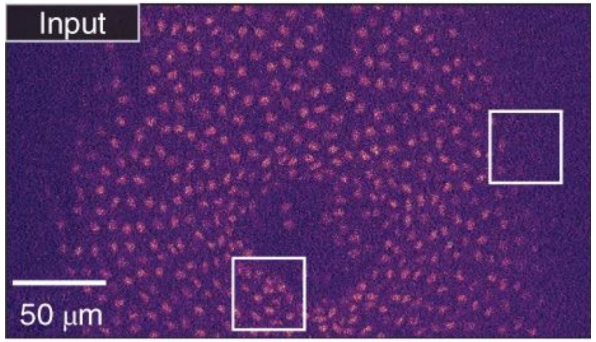
How do we obtain ground-truth for denoising? **Image in noisy and denoised conditions.**

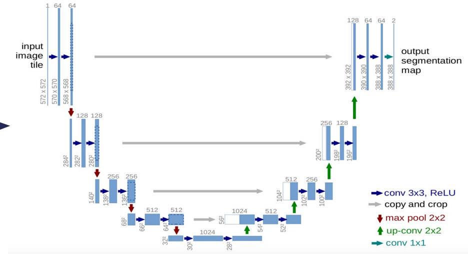
- Fluorescence microscopy: image with different laser intensities.
  - High intensity = better signal
  - Not feasible for long experiment due to bleaching / phototoxicity
- EM: image with longer electron beam dwell times.
  - Longer dwell time = better signal
  - Takes too long for large scale acquisition, may damage sample.

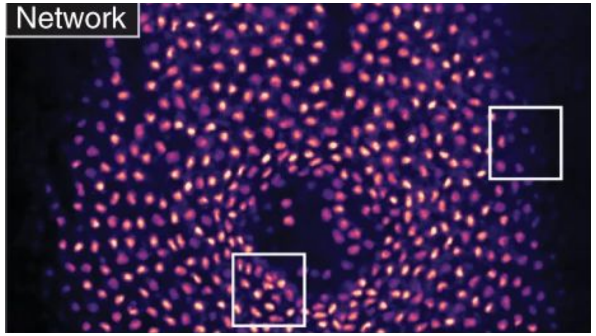
Denoising: CARE

https://www.biorxiv.org/content/10.1101/236463v5

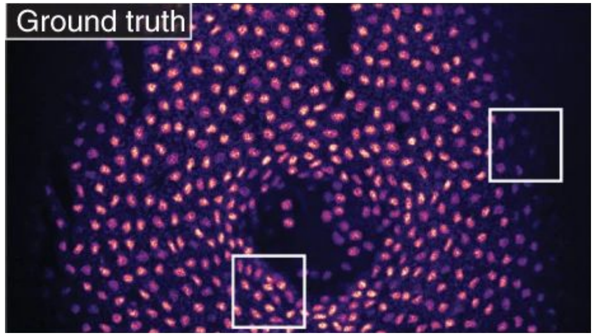
How do we obtain ground-truth for denoising? **Image in noisy and denoised conditions.**







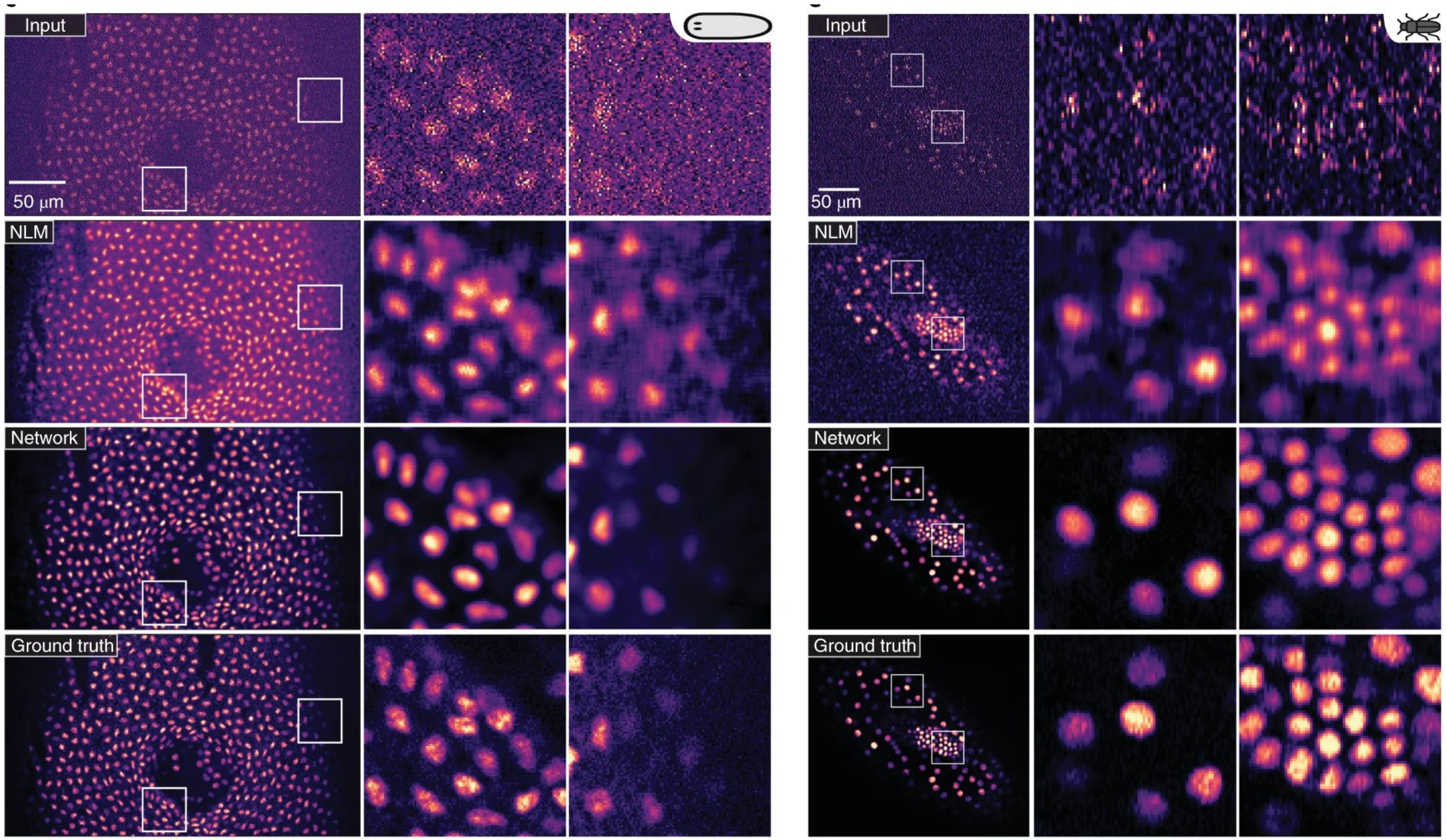
MSE Loss



MSE = Mean Squared Error (L2 Norm): Squared difference of pixel intensities, averaged over all pixels

Denoising: CARE

https://www.biorxiv.org/content/10.1101/236463v5

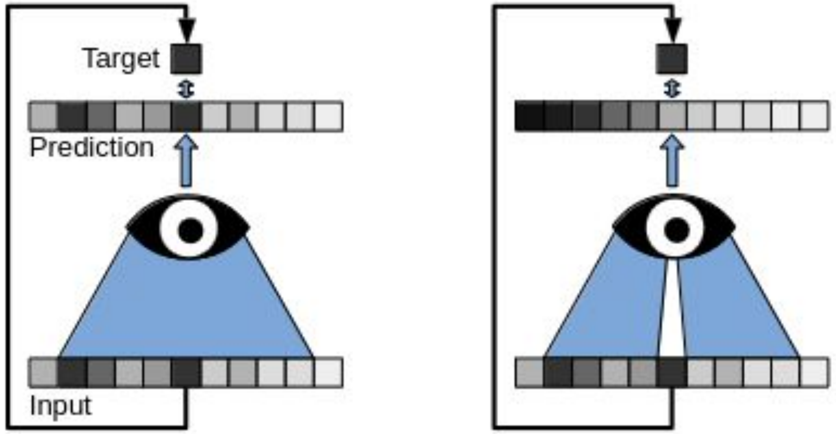


What if we can't take better images?

Can we do anything if clean targets are not available?

Yes! There are tasks we can solve for noisy data that allow denoising!

- Noise2Noise: predict a different noisy version of the image.
- **Noise2Void:** predict pixels from surrounding in noisy data. (Popular for microscopy data!)

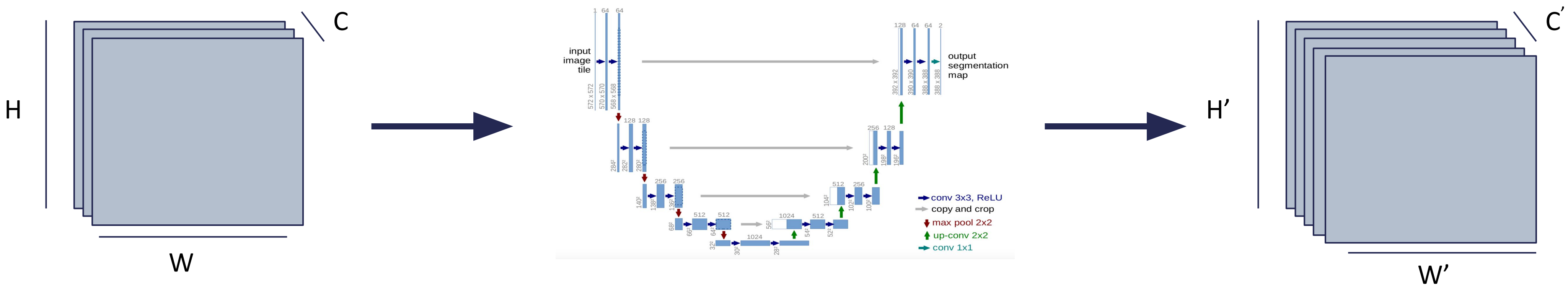


https://arxiv.org/abs/1811.10980

Other image restoration tasks

Several other image restoration tasks that can be solved with U-Net (or similar)

- “Super-resolution”: **H x W -> (s * H) x (s * W)**
  - for scale factor **s > 1**
- Slice prediction for anisotropic data: **H x W x D -> H x W x (s * D)**
  - e.g. **s = 2**: double the number of slices = double axial resolution



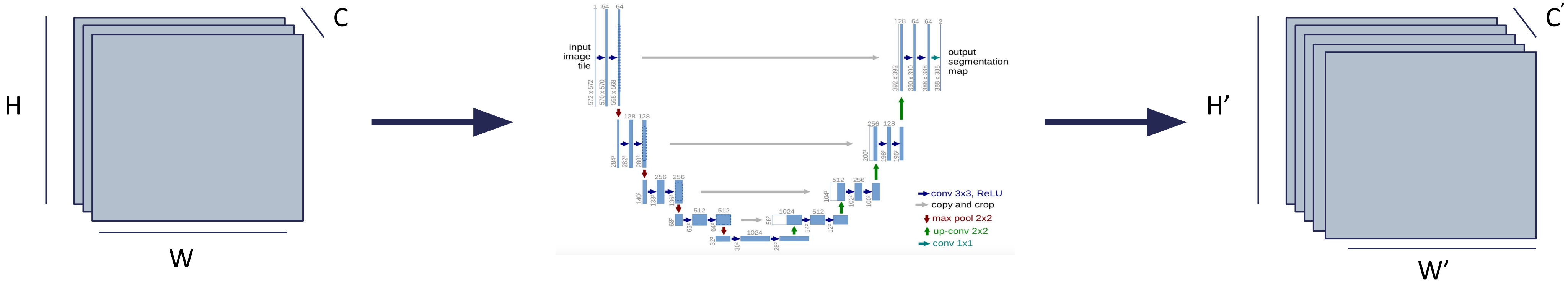
Other image restoration tasks

Several other ima

- “Super-resol
    - for scale
- Slice predicti
    - e.g. **s = 2**

Framework for image restoration tasks in microscopy:
CAREamics: https://careamics.github.io

Provides a python library and napari plugin



Second Exercise

Explain the second exercise

Explain second and third exercise