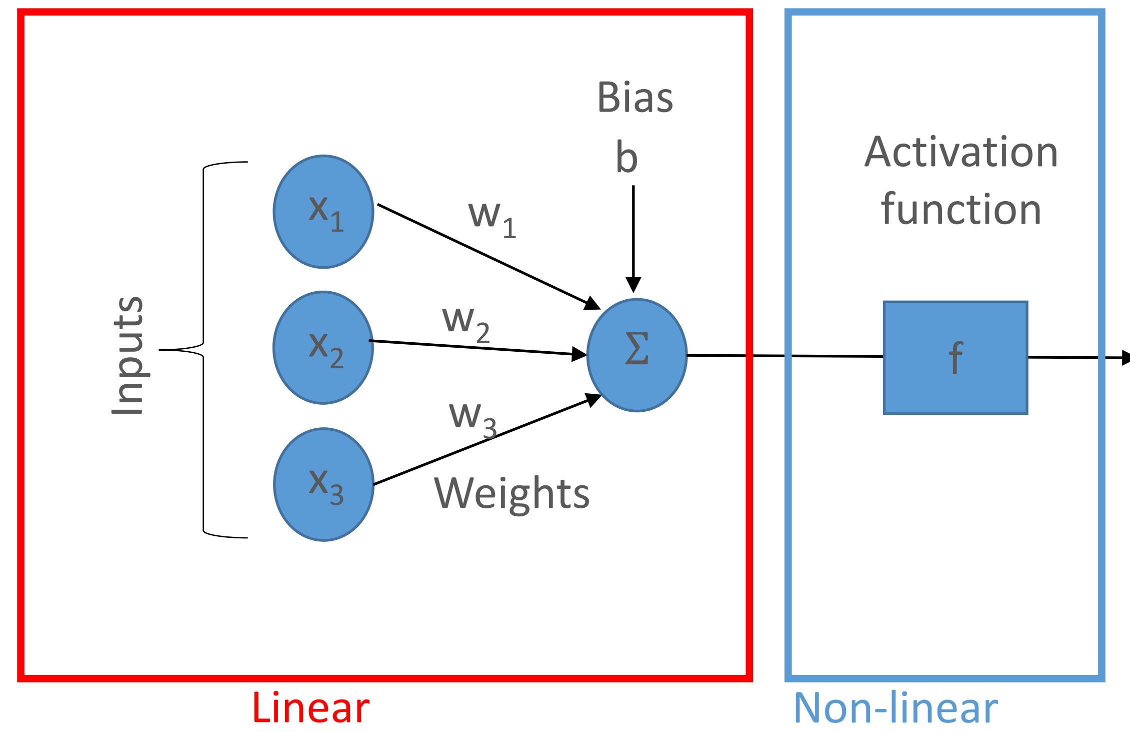
Convolutional neuronal networks

Johannes Soltwedel

With Material from
Robert Haase, PoL
Alex Krull, MPI CBG
Martin Weigert, EPFL Lausanne
Uwe Schmidt, MPI CBG
Ignacio Arganda-Carreras, Universidad del Pais Vasco

Convolutional neural networks



Single neuron output calculation

$$y = w_1x_1 + w_2x_2 + w_3x_3 + b = w^Tx + b$$

Goal: Change w_i so that $y = \hat{y}$

Prediction **Gound truth**

For image data, the values x_1, x_2,… would be

Pixel intensities

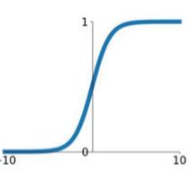
Pixel coordinates

Filter kernel entries

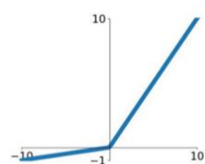
@jm_mightypirate

Activation functions

Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$

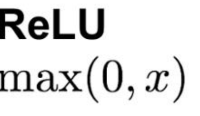


Leaky ReLU
$\max(0.1x, x)$

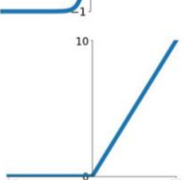


tanh
$\tanh(x)$

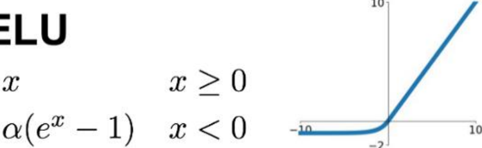
Maxout
$\max(w_1^Tx + b_1, w_2^Tx + b_2)$



ReLU
$\max(0, x)$

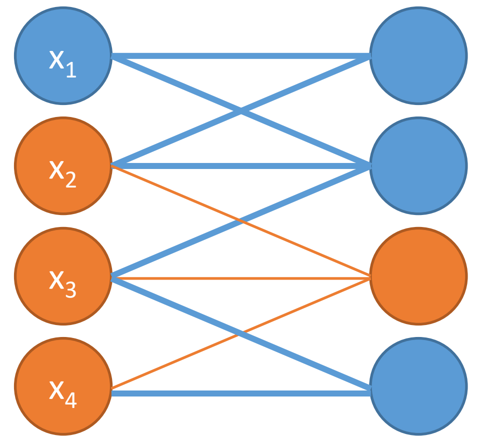


ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$

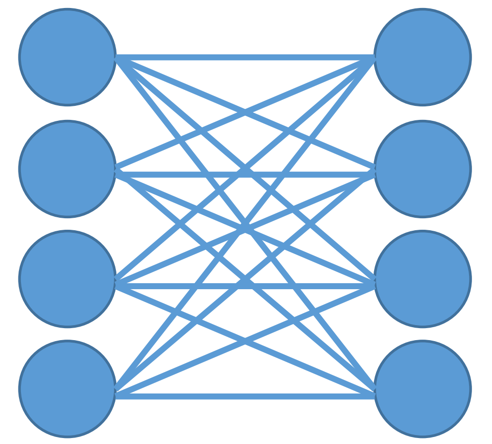


Convolutional neural networks

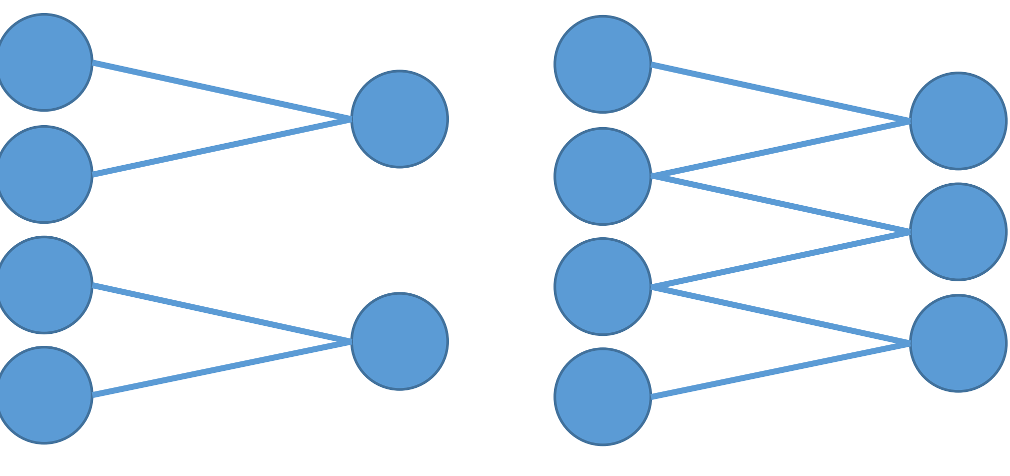
- Layers



Convolutional layer



Fully connected layer



Pooling layer
("Max pool", "Average pool")

Previously:
Defined filter kernels

1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

Now:
Undefined filter kernels

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

3	15	1	13
9	7	0	10
11	5	5	3
1	8	9	6

Pooling maximal values

15	13
11	9

Averaging values

8.5	6.0
6.3	5.8

@jm_mightypirate

Learning: Back propagation

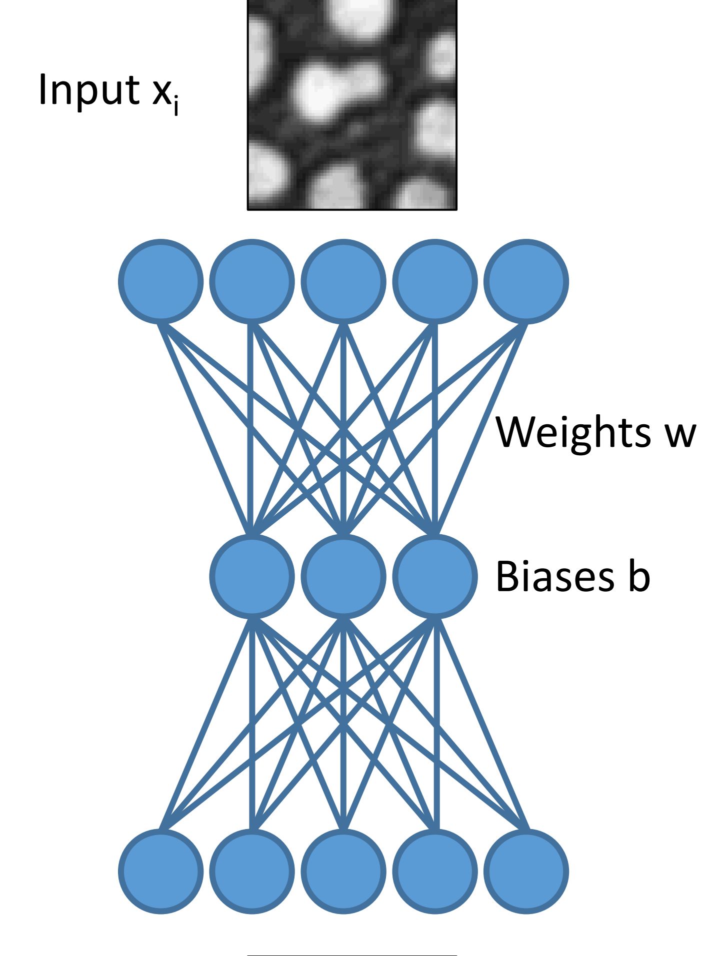
- Learning is an optimization problem

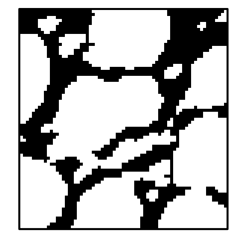
- Step 0: Initialize the network randomly
  - Weights
  - Bias
- Step 1: Forward pass the input through the network, get an initial prediction (Images 0…M)

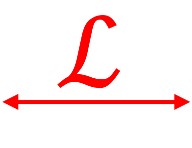
- Step 2: Compare the output with the ground truth, computer the error (loss function)
  - The loss function can be freely defined.
  - Mean squared error:

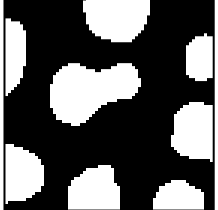
$$\mathcal{L}(y, \hat{y}) = \frac{1}{M}\sum_{i=1}^{M}(\hat{y}_i - y_i)^2$$

- Step 3: Update weights



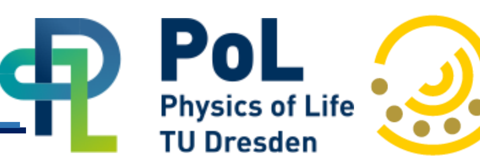






Slide adapted from: Mahmood Nazari, TU Dresde

Back-Propagation Algorithm



The loss function can be expanded from

$$\mathcal{L}(y, \hat{y}) = \frac{1}{M}\sum_{i=1}^{M}(\hat{y}_i - y_i)^2$$

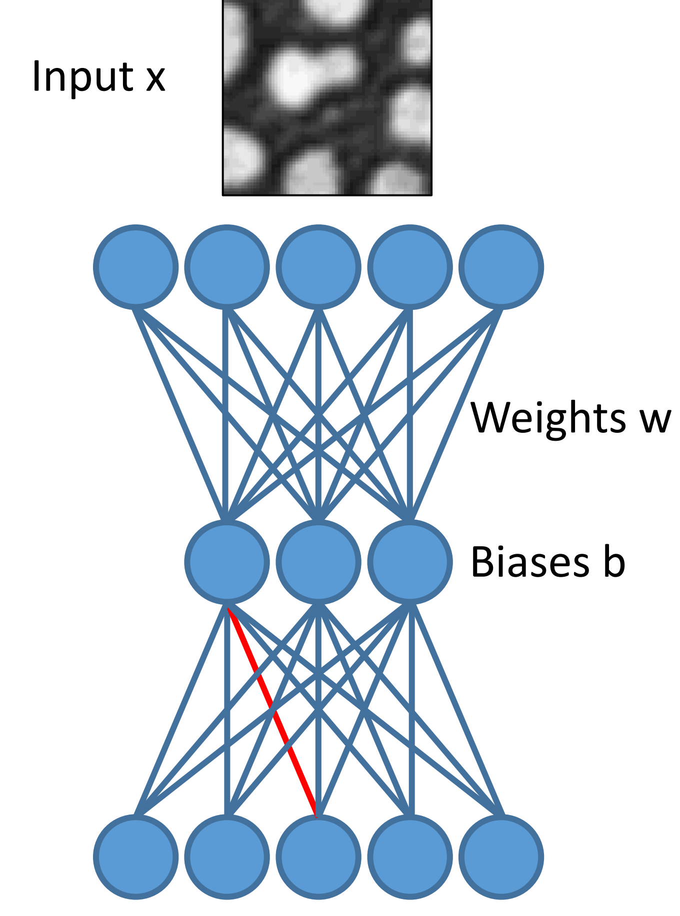
as the prediction depends on inputs x weights w and bias b

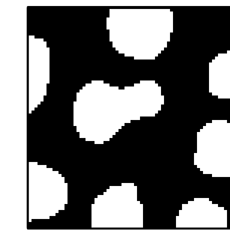
$$\mathcal{L}(\hat{y}, x, w) = \frac{1}{M}\sum_{i=1}^{M}(\hat{y}_i - (w^T x_i + b))^2$$

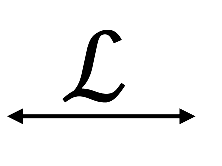
We can calculate derivatives with respect to w and b to find their optimal values

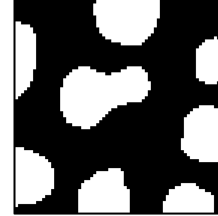
→ Derivatives tell us how to change w & b in order to improve the prediction

Repeat this n times, each time update weights w









@jm_mightypirate

Slide adapted from : Mahmood Nazari, TU Dresden

Image segmentation

Encoder → Decoder →

Where — Where

crop and copy

crop and copy

crop and copy

crop and copy

What

3x3 convolution, ReLu activation
2x2 max pool
2x2 up conv.
1x1 conv.

- The **U-net** is the most used network architecture in biological image processing using CNNs.
  - Encoder: Increase the “What”, decrease the “Where”
  - Decoder: Increase the “Where”, decrease the “What”

@jm_mightypirate

Ronneberger et al. (2015): U-Net: Convolutional Networks for Biomedical Image Segmentation

Object detection: YOLO & classification

YOLO: You only look once

https://github.com/ultralytics/yolov5

Annotations:

Class	x	y	w	h
0	0.09	0.48	0.09	0.22
0	0.22	0.53	0.2	0.27
1	0.56	0.49	0.21	0.30
1	0.68	0.49	0.09	0.12
2	0.9	0.57	0.17	0.23

Currently supported:

- Object detection
- Mask segmentation
- Object tracking

- Easy to train (only bounding boxes necessary)
- Currently restricted to 2D → still interesting for object detection, e.g. in smart microscopy
- Speed: < 10ms/frame (Yolo v5x) → Almost real-time for some cameras

Image denoising: CARE

- **CARE**: content-aware restoration
- Image acquisition of pairs of images: A high-quality and a low-quality image.
- Problem: Shot noise, Biology moves!
- Trained model only applicable to image data of the same conditions (biological system, microscope, etc)

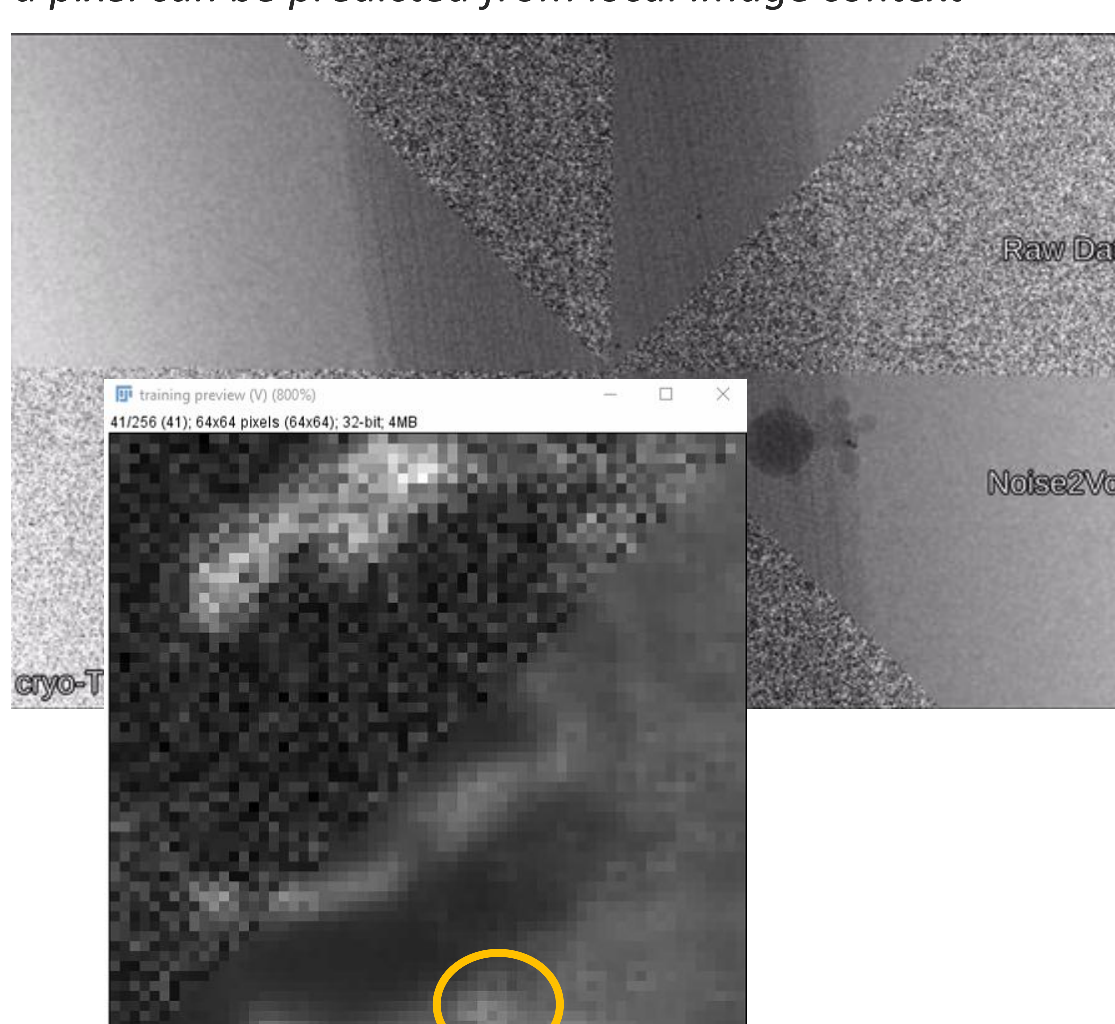
5 example validation patches
top row: input (source), middle row: target (ground truth), bottom row: predicted from source

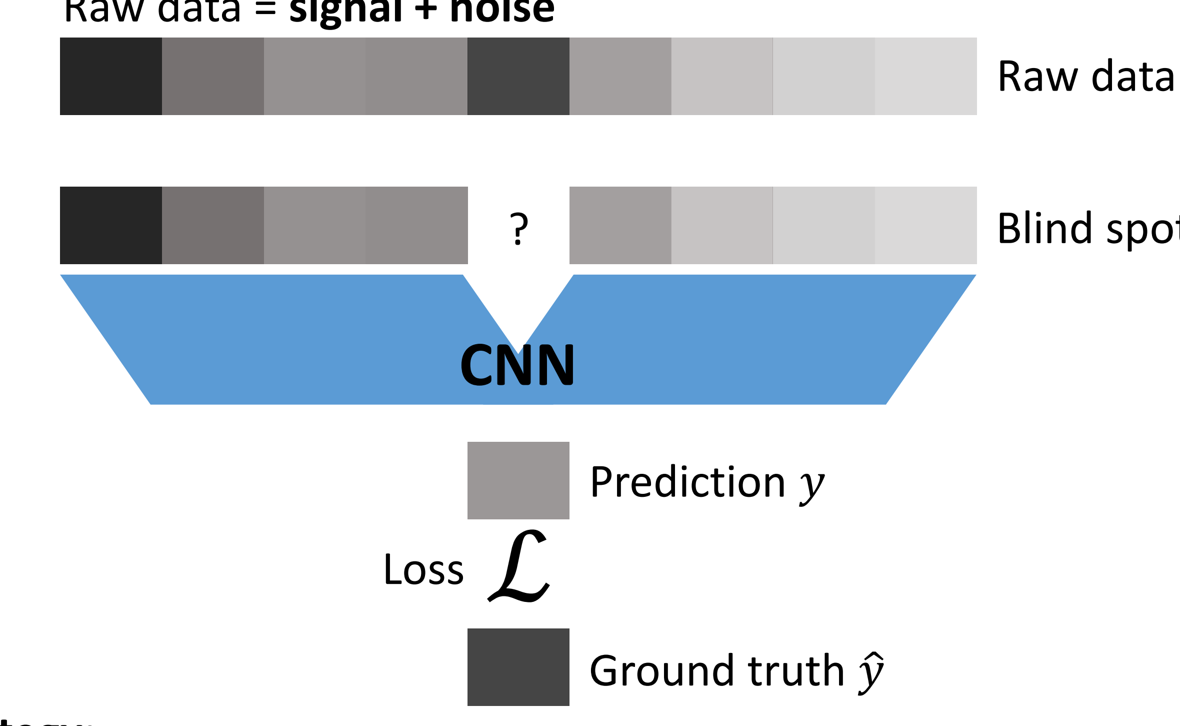
@jm_mightypirate

https://csbdeep.bioimagecomputing.com/

Image denoising: Noise2Void

“Self-supervised training assumes that the noise is pixel-wise independent and that the true intensity of a pixel can be predicted from local image context”





Strategy:

→ Try to predict intensity of pixel y from surrounding pixels x

→ CNN fails to predict noise component → N2V can only reproduce signal from the surroundings of **y**

→ Only **random/uncorrelated** noise can be removed, otherwise artifacts occur

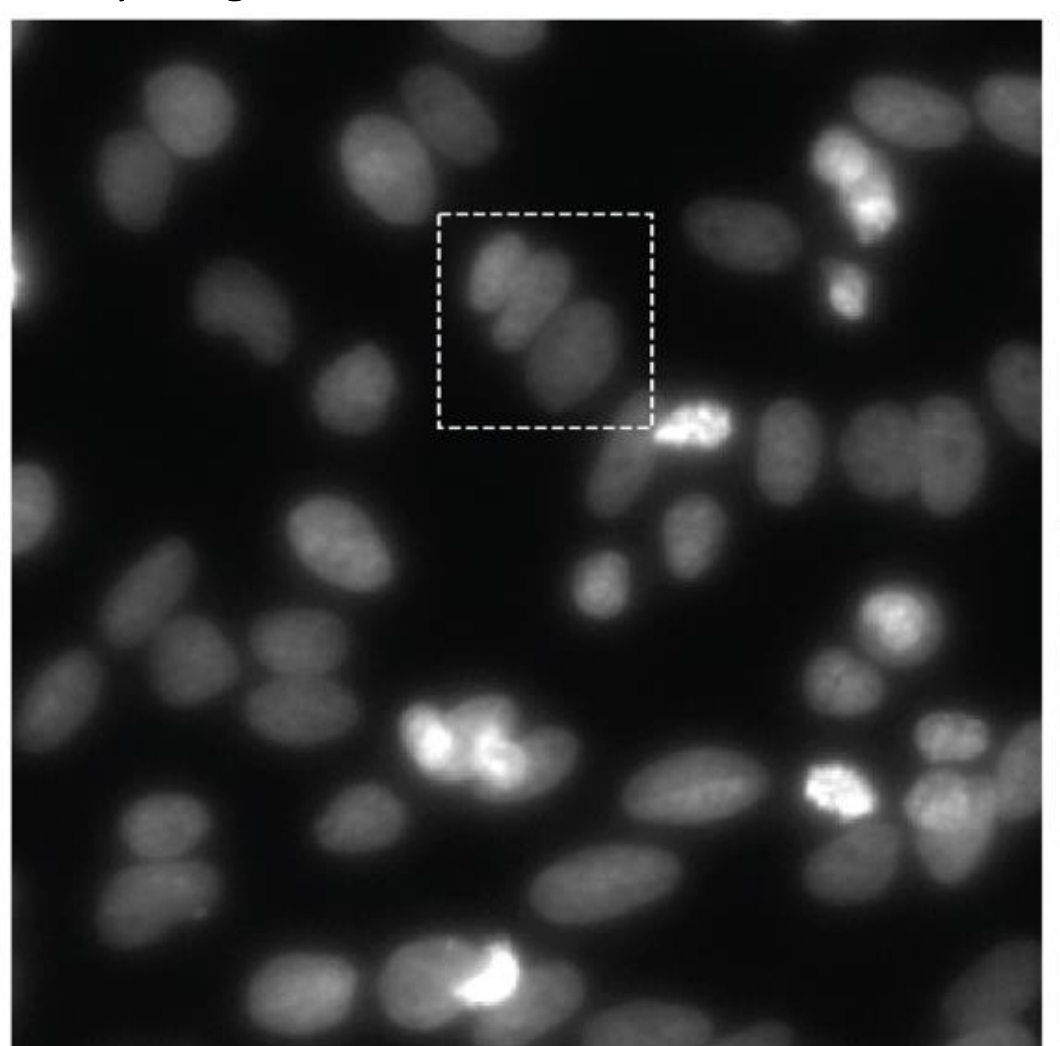
https://github.com/juglab/n2v

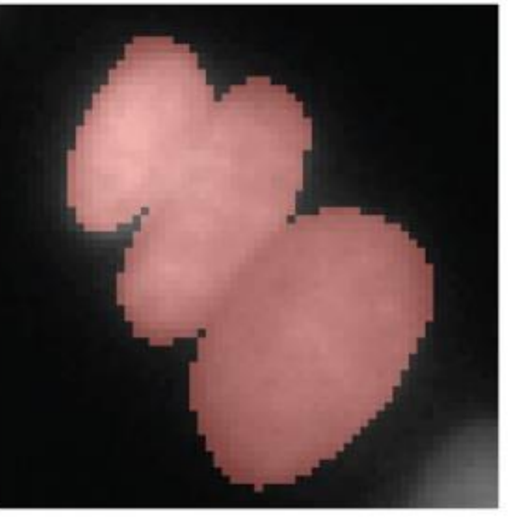
https://forum.image.sc/t/n2v-artefacts-in-training-data/70686

@jm_mightypirate

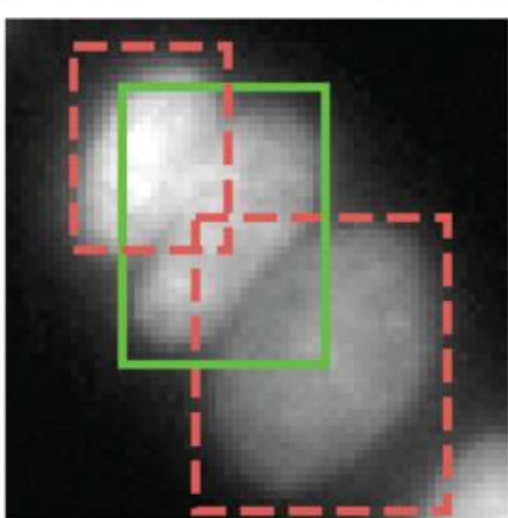
Stardist: Nucleus segmentation

Noisy images + Crowded cells = Common source of segmentation errors





Dense Segmentation
(e.g. U-Net)



Bounding box based methods
(e.g. Mask-RCNN)

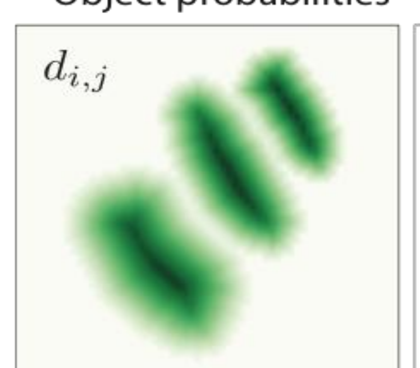
@jm_mightypirate

Schmid et al., MICCAI (2018), https://github.com/stardist/stardist

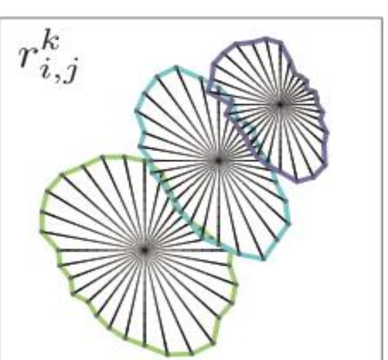
Stardist: Nucleus segmentation

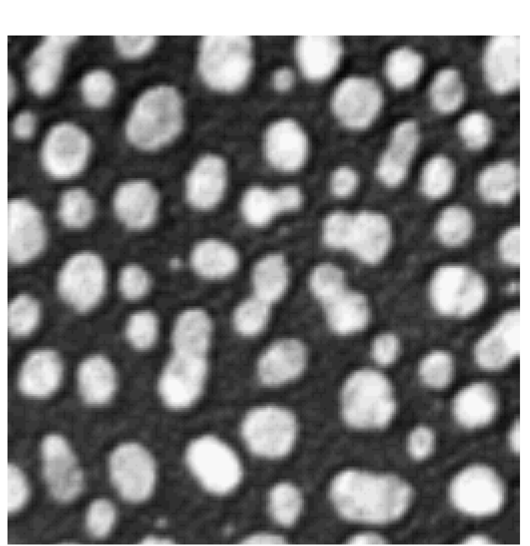
Strategy:
→Add additional information to prediction
→Member pixels of objects (nuclei) can be reached via a straight line from the center

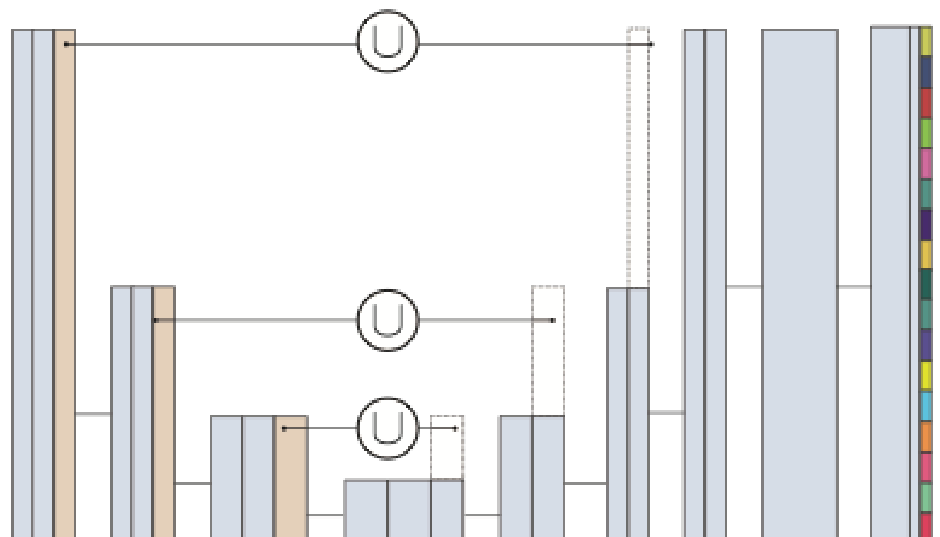
Object probabilities

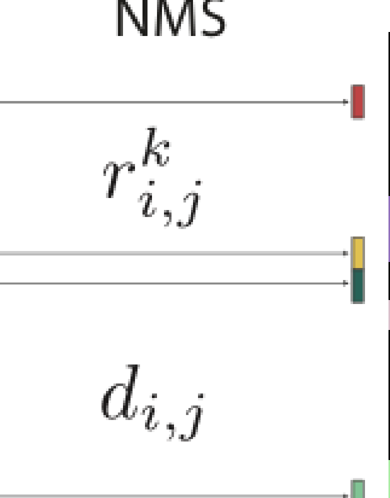


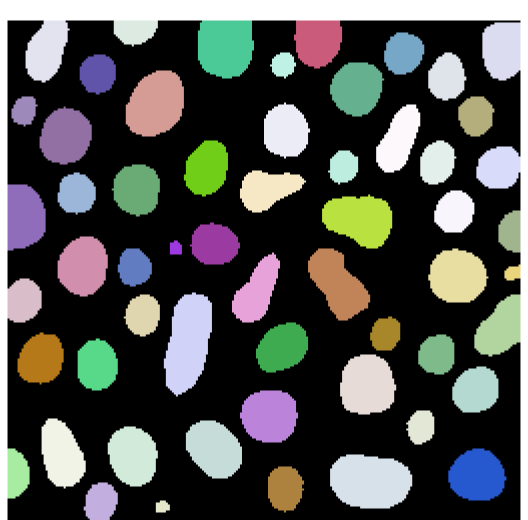
Radial Distances











Dense Polygon Prediction
(e.g. U-Net, ResNet)

Polygon Selection
(Non-Maximum Suppression NMS)

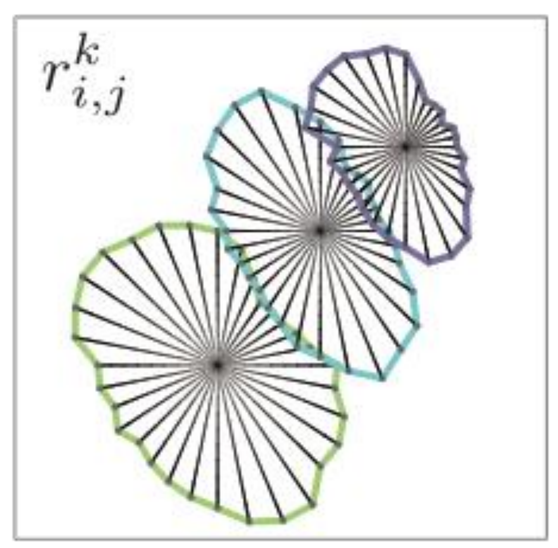
Schmid et al., MICCAI (2018), https://github.com/stardist/stardist

@jm_mightypirate

Stardist: Nucleus segmentation

Object probabilities

Radial Distances



Problem:

→ Multiple candidate points for nucleus center

→ Overlapping instance predictions

Before NMS

After NMS

Non-maximum-suppression (NMS):

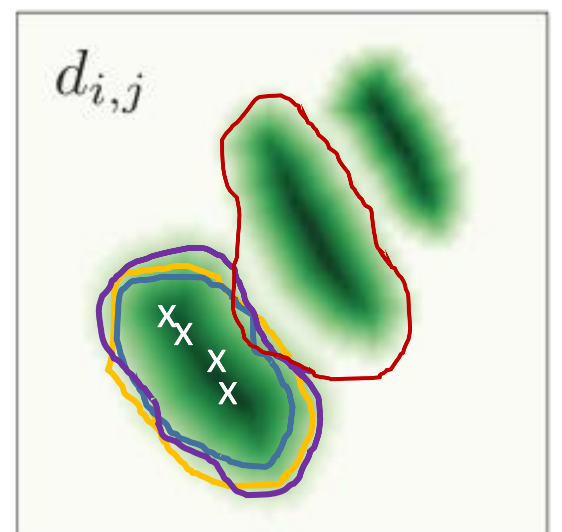
→Intersection over Union (IoU) threshold τ determines „conservativeness“:

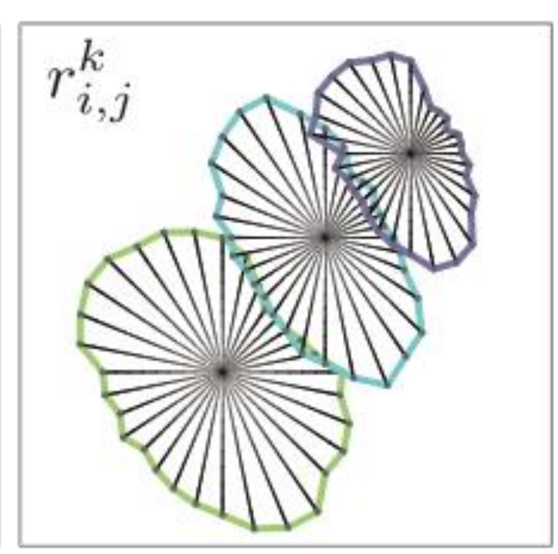
High τ: Objects tend to be considered as separate objects

Low τ: Objects tend to be considered as the same objects

Schmid et al., MICCAI (2018), https://github.com/stardist/stardist

Stardist: Nucleus segmentation





Non-maximum-suppression (NMS):

→ Object probabilities: Probability that pixel belongs to class "nucleus"

→ Multiple maxima lead to multiple possible polygons for the same nucleus

Algorithm:

→ Select polygon with highest object probability inside:

→ Look at other polygons: Is the overlap of with larger than threshold τ?

→ Yes: and are actually the same object, drop

→ No: and are separate nuclei

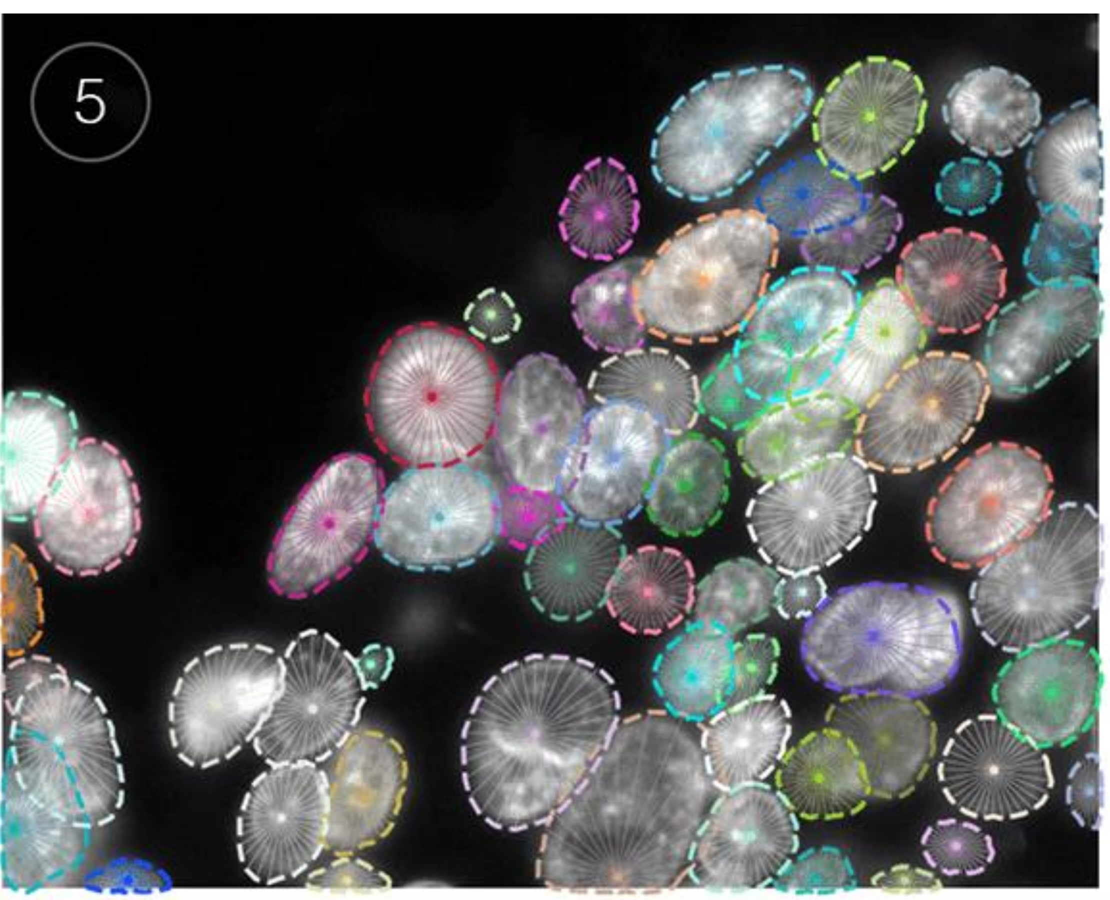
→ Setting τ very high leads to many false positives!

@jm_mightypirate

Schmid et al., MICCAI (2018), https://github.com/stardist/stardist

Stardist: Nucleus segmentation

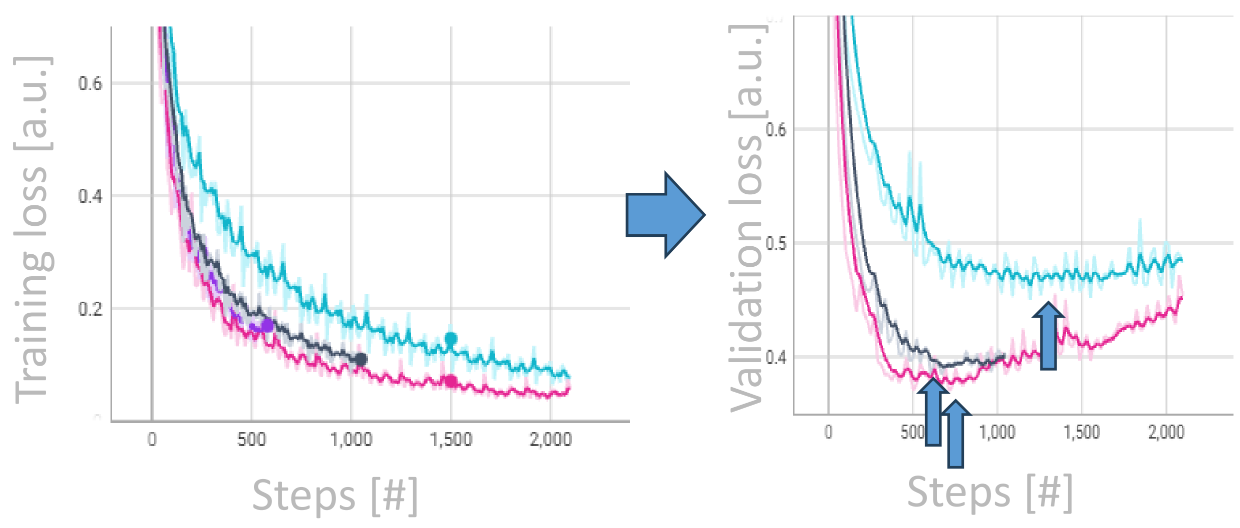
Non-maximum suppression



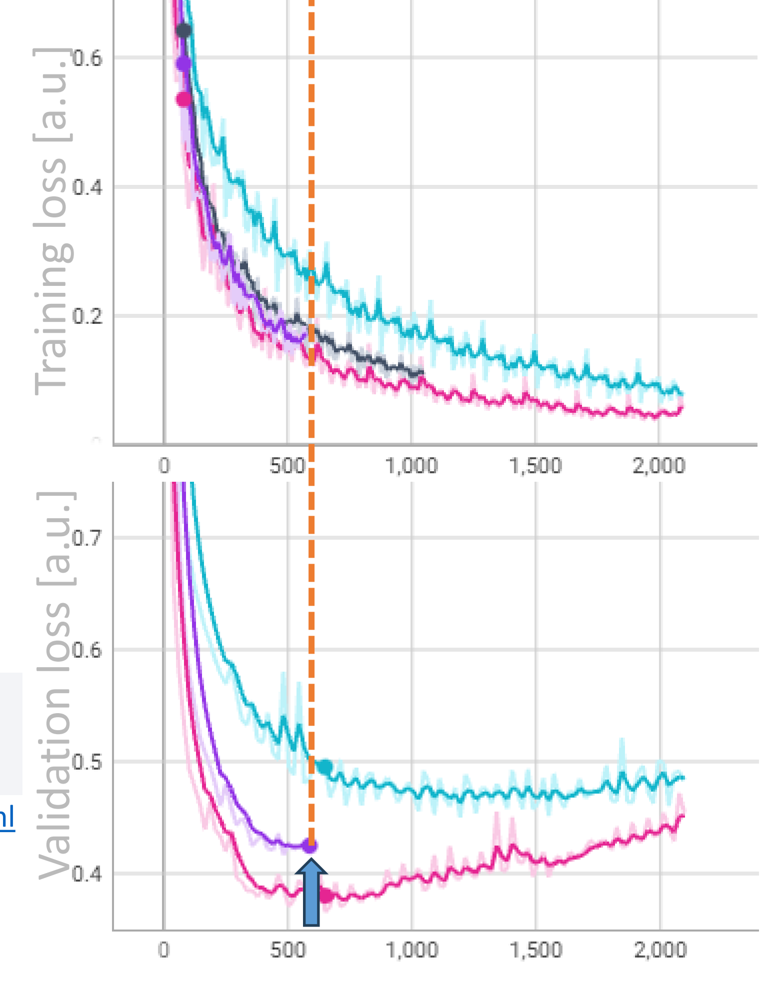
@jm_mightypirate

Schmid et al., MICCAI (2018), https://github.com/stardist/stardist

Overfitting



Overfitting:
→ Network is learning things „by heart"
→ Hint at this happening: Updated weights from training fail to perform well in test



Strategies:
→ Save two models: Last model and best model – may be susceptible to noise
→ Early stopping: End training loop preemptively if validation loss does not increase by ΔL within n episodes

```
early_stop_callback = EarlyStopping(monitor="val_accuracy", min_delta=0.00, patience=3, verbose=False,
mode="max")
trainer = Trainer(callbacks=[early_stop_callback])
```

https://lightning.ai/docs/pytorch/stable/common/early_stopping.html

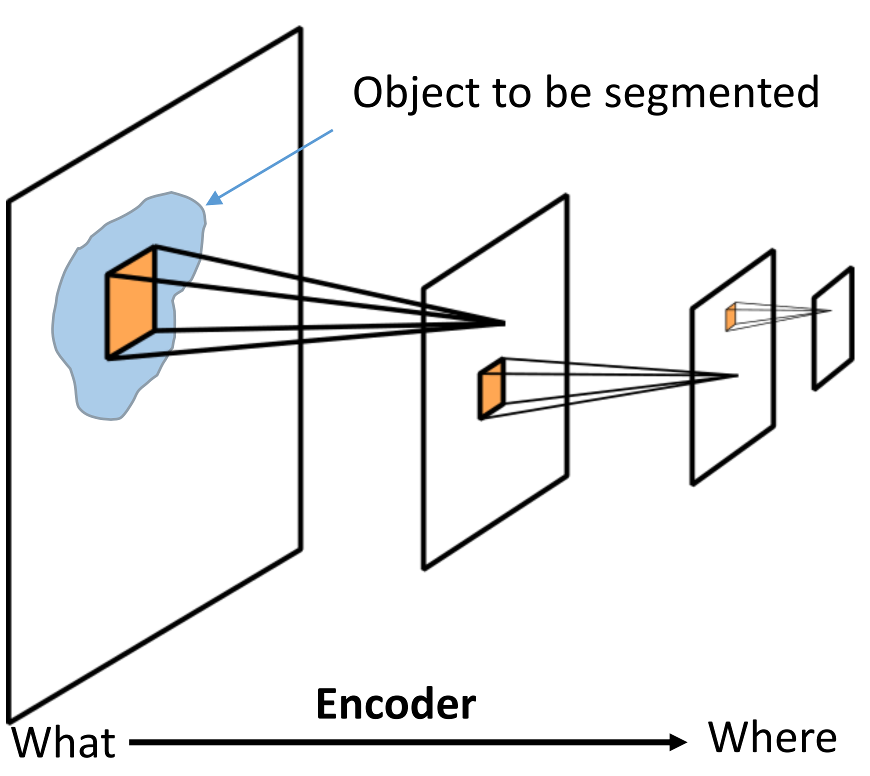
@jm_mightypirate

Receptive field

Many prediction frameworks use UNets – similar weak points

→Neurons in deeper layers can only "see" parts of the raw image

→Objects must be smaller than receptive field to be detectable/seen



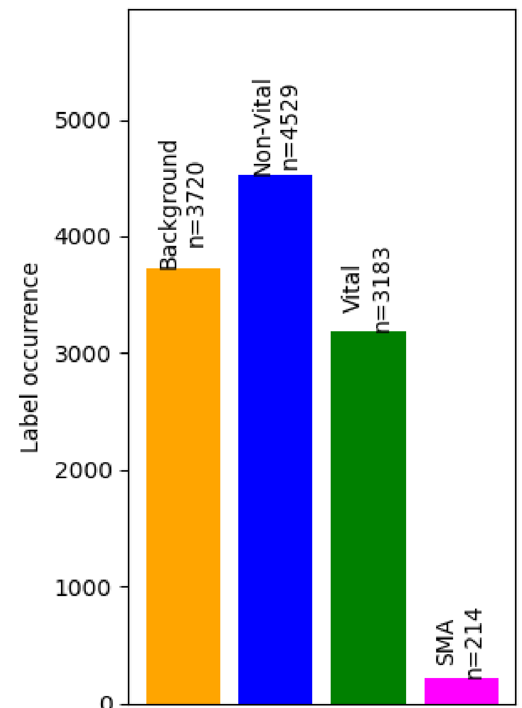
Strategies:

- Make network deeper (add more layers)
  - Increases receptive field ☺
  - Increases number of weights and hardware requirements ☹
- Resample input:

  Think before: Which level of detail is actually required to perform the task at hand?

Unbalanced training data

Common case Heterogeneous occurrence of labels in training data



→ Rare events will not be caught because they don't harm accuracy much:

Example: Two classes to be predicted: Necrotic tissue (0.1% of tissue area) & Vital tissue (99.9% of tissue area) Simply predicting *everything* as vital leads to a high accuracy!

→ Problematic in detecting rare events:

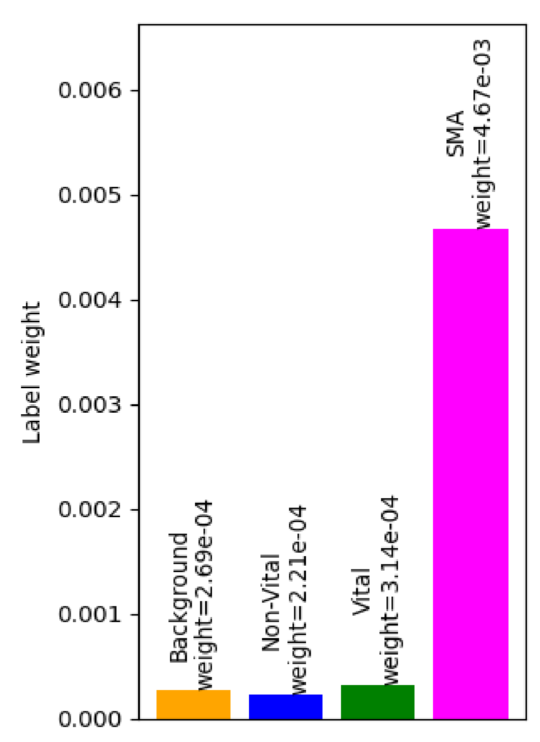
Rare pixel classes, few patients in study with specific mutation, rare disease

→Can happen intentionally or unintentionally

Is the iPhone racist? Chinese users claim iPhoneX face recognition can't tell them apart

APPLE has come under fire following numerous complaints from Chinese users who claim the iPhone X face recognition can't tell them apart.

https://www.news.com.au/



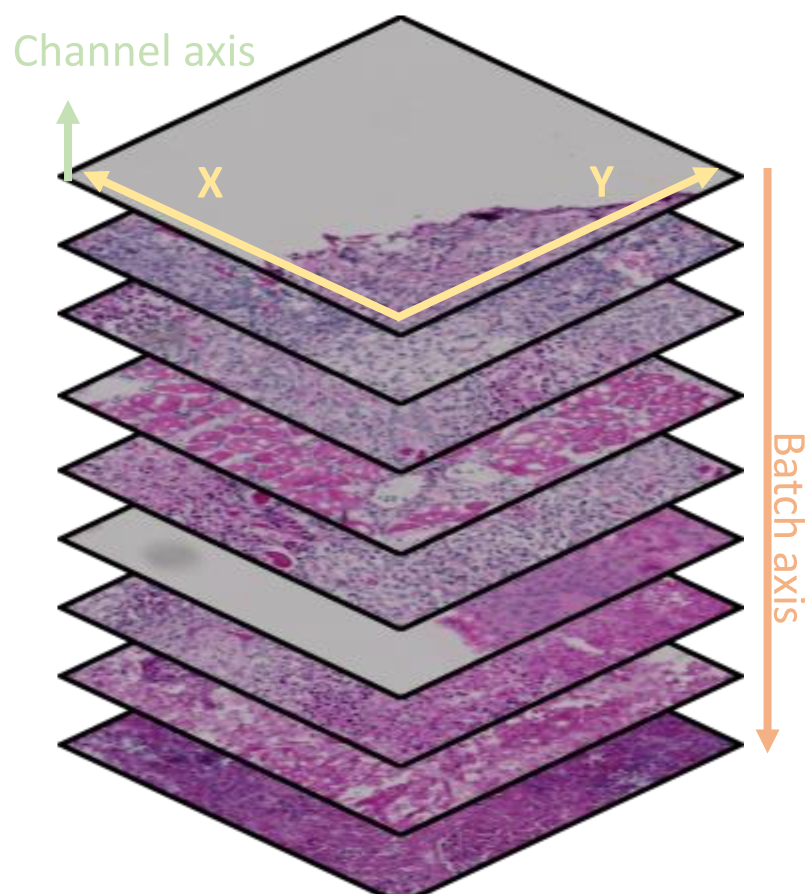
Strategy: Weighted sampling

→ During training, show rare samples more often to network than others

→ Still: More & better data > heavy weighting

Batch averaging

- During training and validation, images and labels are stacked into *batches* and processed in parallel:



Array dimensions: array.shape = [B, C, X, Y]

Batch averaging

- Images in batches are z-scaled before forward pass through network:

$$batch' = \frac{batch \ - \mu(batch)}{\sigma(batch)}$$

- Mean and standard deviation of batch:

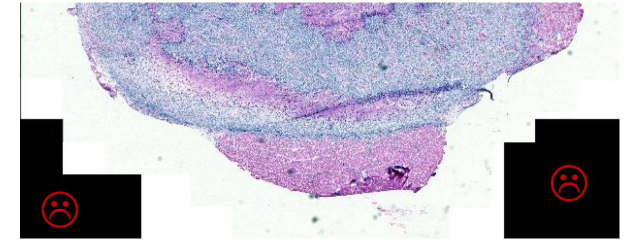
$$\mu(batch') = 0 \ \ \& \ \ \sigma(batch') = 1$$

Pitfalls:

- Different regions/timepoints in image data can have very different intensity characteristics (bleaching, diffraction in depth, etc)
- Shuffle training data

```
# Create dataloaders
num_workers = 80
train_dataloader = DataLoader(dataset=train_dataset, batch_size=batch_size, num_workers=num_workers, shuffle=True)
test_dataloader = DataLoader(dataset=test_dataset, batch_size=batch_size, num_workers=num_workers)
validation_dataloader = DataLoader(dataset=validation_dataset, batch_size=batch_size, num_workers=num_workers)
```

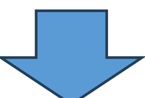
- Careful during inference! Microscopes sometimes automatically black out/avoid certain “uninteresting regions – including these in the batch-averaging will mess up the prediction!



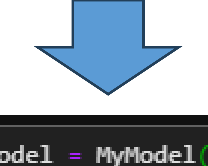
@jm_mightypirate

Information leakage

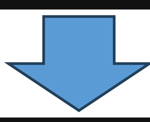
```
val_size = int(0.1 * len(MyDataset))
train_size = int(0.8 * (len(MyDataset) - val_size))
test_size = len(MyDataset) - val_size - train_size
train_dataset, test_dataset, validation_dataset = torch.utils.data.random_split(MyDataset, [train_size, test_size, val_size])

print('Samples in training set: ', len(train_dataset))
print('Samples in testing set: ', len(test_dataset))
print('Samples in validation set: ', len(validation_dataset))
```



```
# Create dataloaders
num_workers = 80
train_dataloader = DataLoader(dataset=train_dataset, batch_size=batch_size, num_workers=num_workers, shuffle=True)
test_dataloader = DataLoader(dataset=test_dataset, batch_size=batch_size, num_workers=num_workers)
validation_dataloader = DataLoader(dataset=validation_dataset, batch_size=batch_size, num_workers=num_workers, shuffle=True)
```



```
model = MyModel()
model.train(True)
```



```
trainer.fit(model, train_dataloaders=train_dataloader, val_dataloaders=validation_dataloader)
```

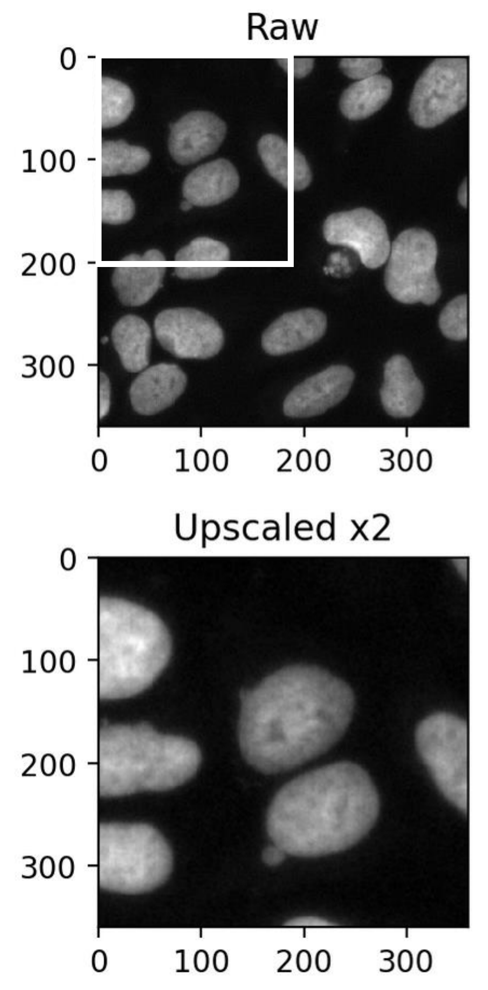
!!!Re-executing this cell multiple times mixes testing/training/validation data!!!

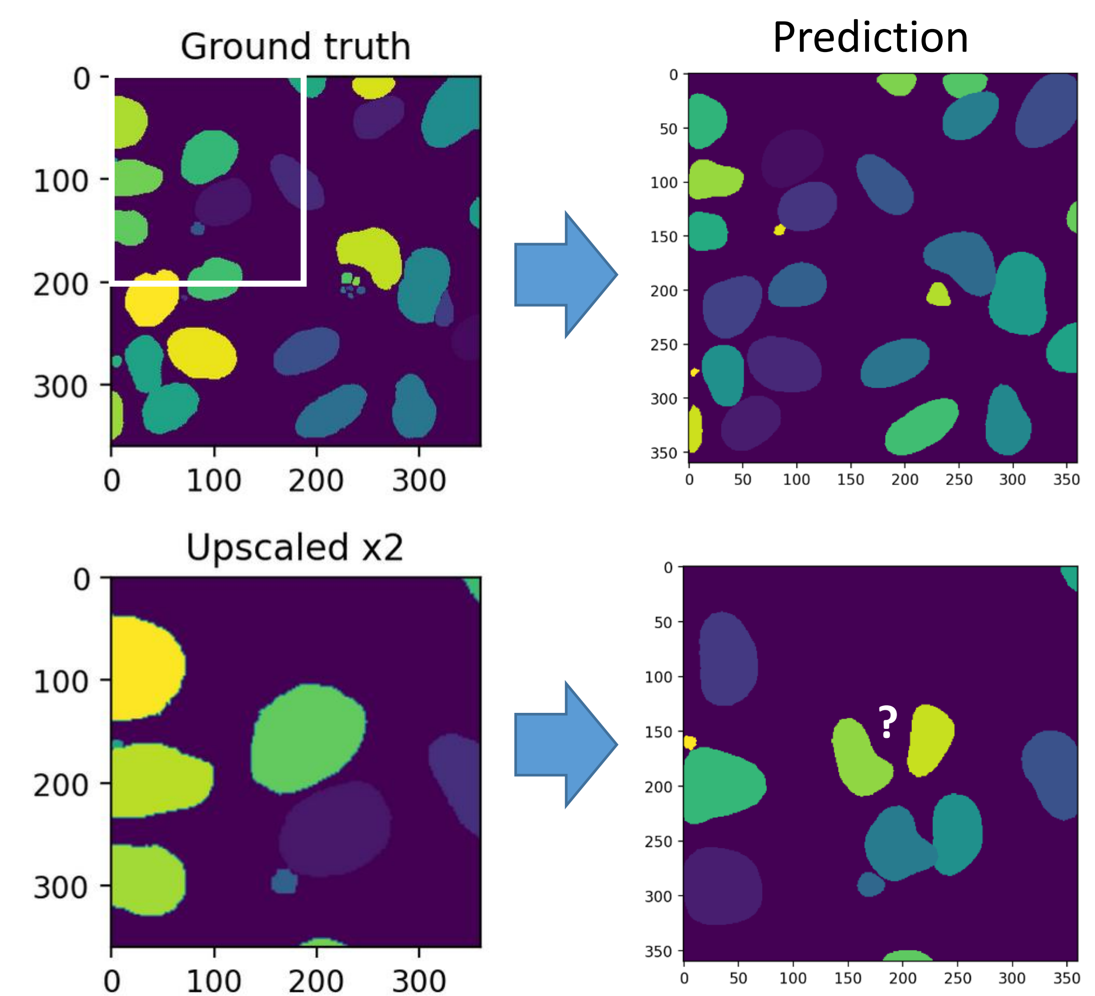
→ Bad practice, **don't do this at home**

→ Better: separate notebook to create separate folders for training/testing/validation data which is only executed once!

@jm_mightypirate

Example





What happened here?

Receptive field too small

I used a different resolution than during training

Overfitting

@jm_mightypirate

Takeaways

- With great power comes great responsibility: **Validate your models well!**
- Better data > better model
- Make model publicly available? → Bio image model zoo: https://bioimage.io/

@jm_mightypirate