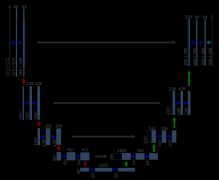


UNIVERSITY OF TWENTE.

Deep Learning for 3D Medical Image Analysis

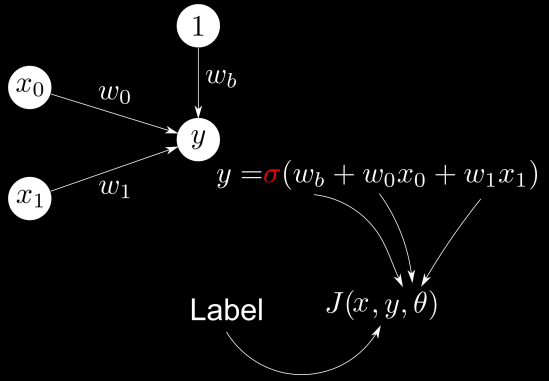
Lecture 3: Segmentation

February 25, 2025

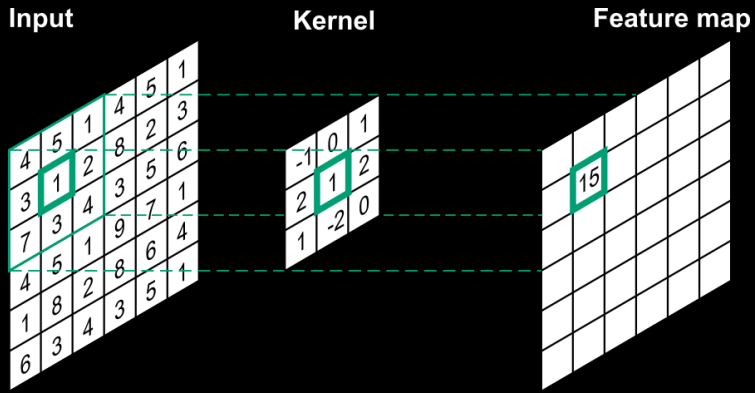
Jelmer Wolterink

Mathematics of Imaging & AI group

Recap: Networks

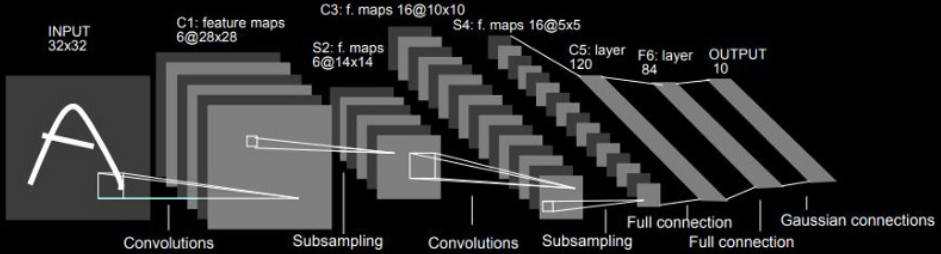


Recap: Filters



$$g(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h(u,v) f(i-u, j-v)$$

Recap: CNNs



Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324

This week

- Image segmentation

This week

- Image segmentation
- Segmentation networks

This week

- Image segmentation
- Segmentation networks
- Segmentation losses

This week

- Image segmentation
- Segmentation networks
- Segmentation losses
- Evaluation metrics

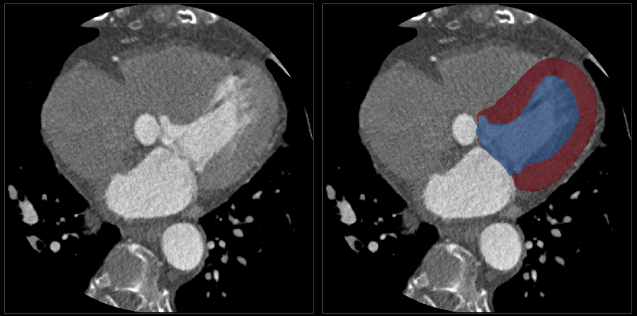
Image segmentation

The problem: image segmentation

Semantic segmentation

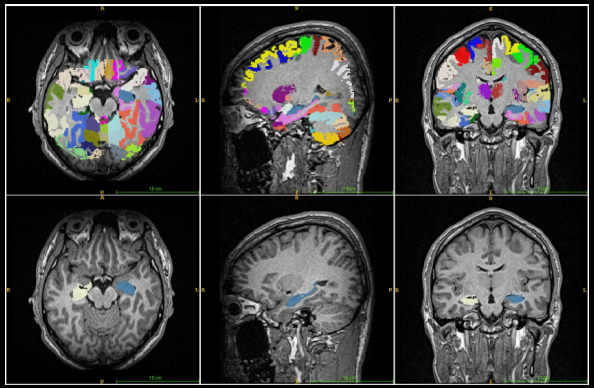
The goal of *semantic* segmentation is to label each pixel (or voxel) in an image with one of n classes.

For example, left ventricle (blue), myocardium (red), and background (no color).



The problem: image segmentation

The number of classes n can be very high! E.g., brain parcellation $\rightarrow$ dozens of labels



From classification to segmentation

In the previous lecture, we have seen image regression or classification

- Input: 2D or 3D image
- Output: **scalar** or **label**

But many problems in imaging are image-to-image

- Input: 2D or 3D image
- Output: 2D or 3D image

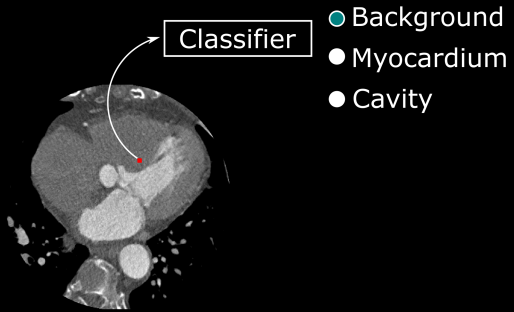
This requires different

- Architectures
- Loss functions
- Training strategies

Voxel classification

Voxel classification

- Input: 2D or 3D image
- Output: 2D or 3D image
- **Strategy**: classify each pixel/voxel



Voxel classification before DL

Context

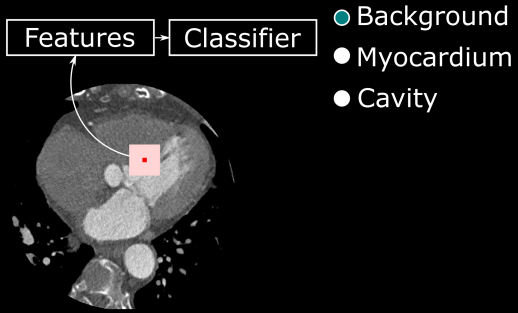
- Describe local neighborhood by features
- Train a classifier to distinguish classes

Example features

- Gaussian filters
- Edge detectors
- Radiomics

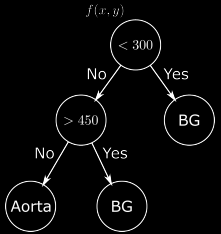
Example classifiers

- kNN
- SVM
- Random Forests

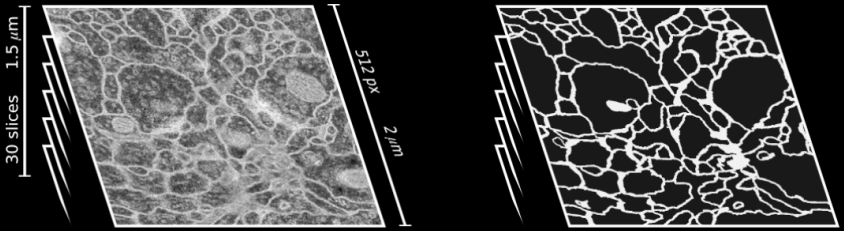


Example: thresholding

- Task: aorta segmentation
- Classifier: decision tree
- Input feature: image intensity $f(x, y)$ (in HU)

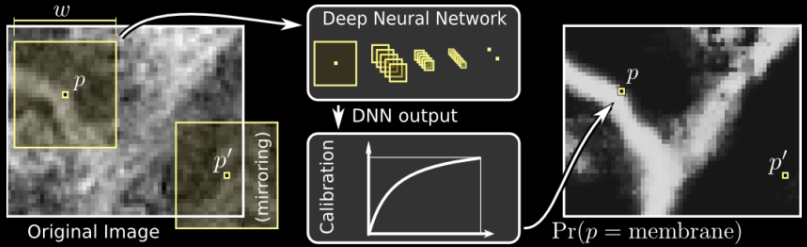


CNNs as classifier



Dan Ciresan et al. “Deep neural networks segment neuronal membranes in electron microscopy images”. In: *NIPS* 25 (2012)

CNNs as classifier



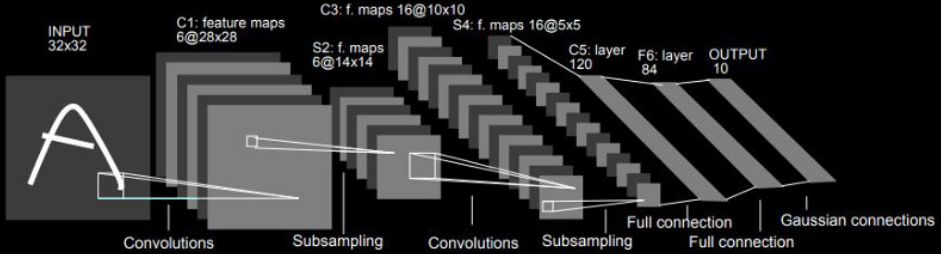
Dan Ciresan et al. “Deep neural networks segment neuronal membranes in electron microscopy images”. In: *NIPS* 25 (2012)

CNNs as classifier

Layer	Type	Maps and neurons	Kernel size
0	input	1 map of 95x95 neurons	
1	convolutional	48 maps of 92x92 neurons	4x4
2	max pooling	48 maps of 46x46 neurons	2x2
3	convolutional	48 maps of 42x42 neurons	5x5
4	max pooling	48 maps of 21x21 neurons	2x2
5	convolutional	48 maps of 18x18 neurons	4x4
6	max pooling	48 maps of 9x9 neurons	2x2
7	convolutional	48 maps of 6x6 neurons	4x4
8	max pooling	48 maps of 3x3 neurons	2x2
9	fully connected	200 neurons	1x1
10	fully connected	2 neurons	1x1

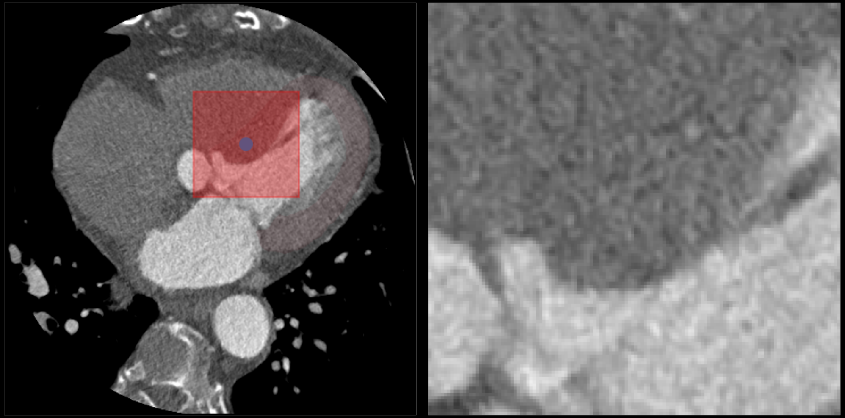
Dan Ciresan et al. “Deep neural networks segment neuronal membranes in electron microscopy images”. In: *NIPS* 25 (2012)

Looks familiar?



Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324

Sliding window CNN

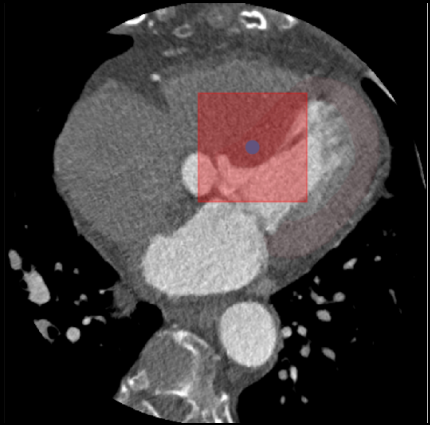


Sliding window CNN

Relatively parameter efficient: same CNN for each pixel

Major drawbacks

- *Training* Large input samples for single pixels
- *Training* Sampling schemes to balance classes
- *Inference* Extremely slow
- *Inference* Many redundant operations



All-convolutional networks

Solution: only use convolutions

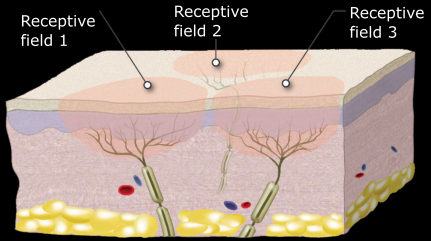
- Convolution is translation equivariant
- Operations are shared between neighboring windows
- This drastically reduces the number of operations

Receptive fields

Receptive field: biological basis

In *biological* neurons

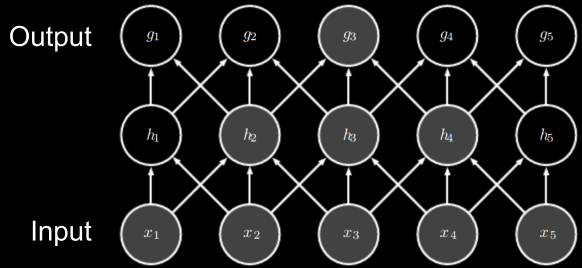
The receptive field of an individual sensory neuron is the particular region of the sensory space (e.g., the body surface, or the visual field) in which a stimulus will trigger the firing of that neuron.



Receptive field: deep learning

In *artificial* neural networks

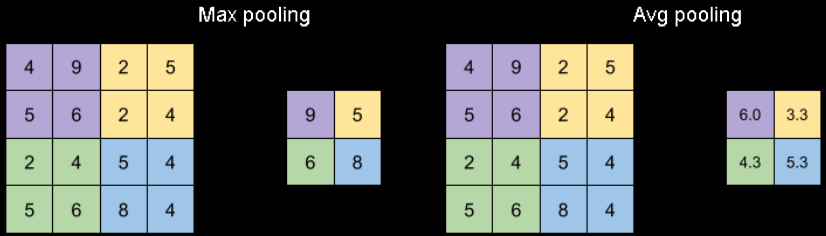
The input units that affect the output of a hidden unit are known as the receptive field of that unit.



How to increase the receptive field?

- Downsampling (pooling)
- **Strided** convolution
- **Dilated** convolution

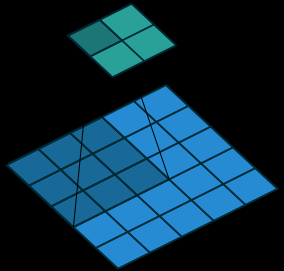
Pooling (last lecture)



Strided convolution

Make use of a stride s

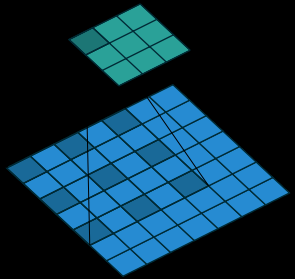
$$g(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h(u,v) f(si-u, sj-v)$$



Dilated or a trous convolution

Make use of a dilation rate d

$$g(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h(u,v) f(i - du, j - dv)$$



Strided vs. dilated convolutions

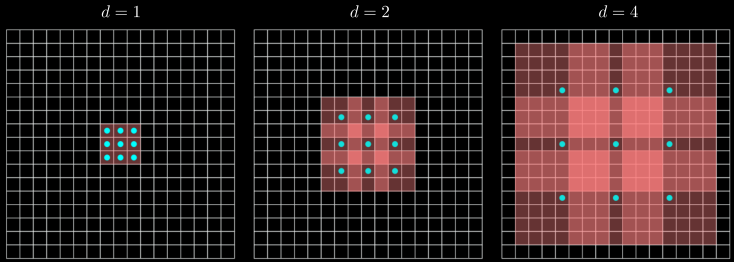
Differences

- *Strided* convolution skips elements in the input image
- *Dilated* convolution skips elements in the kernel
- *Strided* convolution reduces the resolution of the image
- *Dilated* convolution preserves the resolution of the image
- *Strided* convolution is equivariant to translations of s pixels
- *Dilated* convolution is always translation equivariant

$$g(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h(u,v) f(si - u, sj - v) \qquad g(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h(u,v) f(i - du, j - dv)$$

Dilated CNN

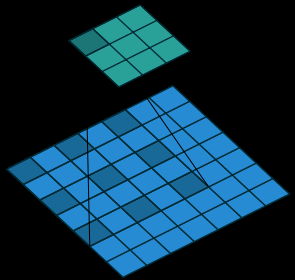
- Dilated convolutions can be stacked
- By doubling d in each layer, the receptive field grows *exponentially*
- The number of parameters grows *linearly*
- In this example, the receptive field is is 15×15 pixels with only 27 weights
- The receptive field is *contiguous* (no holes)



Fisher Yu and Vladlen Koltun. “Multi-scale context aggregation by dilated convolutions”. In: *ICLR 2016* (2015)

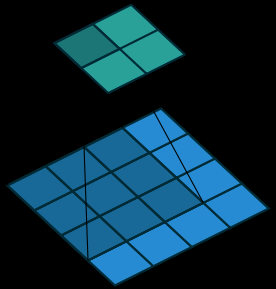
Convolution and padding

- Convolution preserves the **resolution** of the image
- But the output image is smaller than the input image in case of a *valid* convolution
- We can pad before (*same* convolution)
- We can pad afterwards (not preferred)



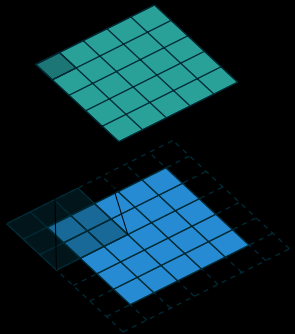
Convolution and padding

- Convolution preserves the **resolution** of the image
- But we lose pixels around the edges if we do any *valid* convolution
- We can pad before (*same* convolution)
- We can pad afterwards (not preferred)



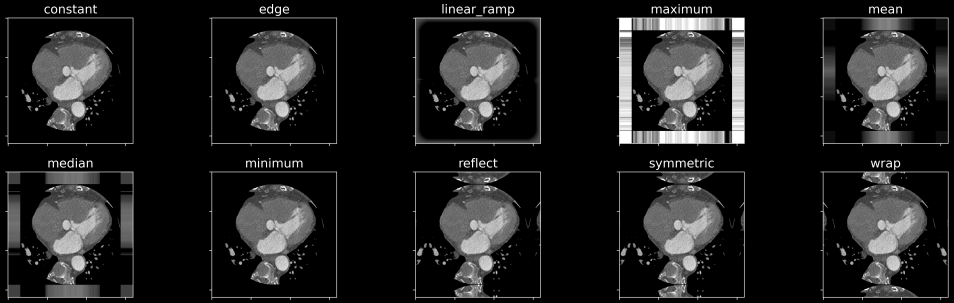
Convolution and padding

- Convolution preserves the **resolution** of the image
- But we lose pixels around the edges if we do any *valid* convolution
- We can pad before (*same* convolution)
- We can pad afterwards (not preferred)



Convolution and padding

- There are many ways to pad
- Consider what would make most sense (here, e.g., constant, edge or minimum)
- Padding shouldn't interfere with learning
- When constant padding, don't default to 0 in images



Receptive fields: demo

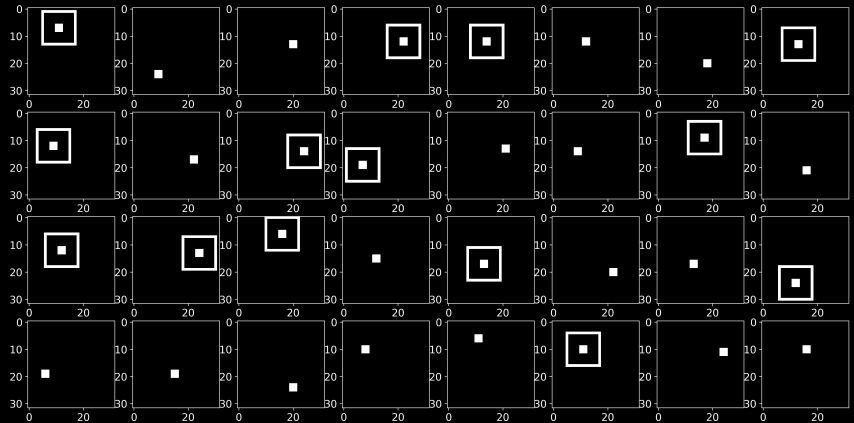
Toy problem

- *Image* $n \times n$ pixel image
- *Lesion* $m \times m$ lesion
- *Types*
  - *Benign* No ring around lesion
  - *Malign* Ring around lesion at distance d

Goal Segment the malign lesions, but not the benign ones

Receptive fields: demo

Samples made with $n = 32, m = 3, d = 5$



Network

Question: What is the receptive field of this network?

```
class ConvolutionalNetwork(nn.Module):

    def __init__(self, n_filter=8):
        super().__init__()
        layers = [nn.ZeroPad2d(3),
             nn.Conv2d(in_channels=1, out_channels=n_filter, kernel_size=3),
             nn.ReLU(),
             nn.Conv2d(in_channels=n_filter, out_channels=n_filter, kernel_size=3),
             nn.ReLU(),
             nn.Conv2d(in_channels=n_filter, out_channels=n_filter, kernel_size=3),
             nn.ReLU(),
             nn.Conv2d(in_channels=n_filter, out_channels=1, kernel_size=1),
             nn.Sigmoid()]
        self.layers = nn.Sequential(*layers)

    def forward(self, x):
        return self.layers(x)
```

Network

Question: What is the receptive field of this network? 7

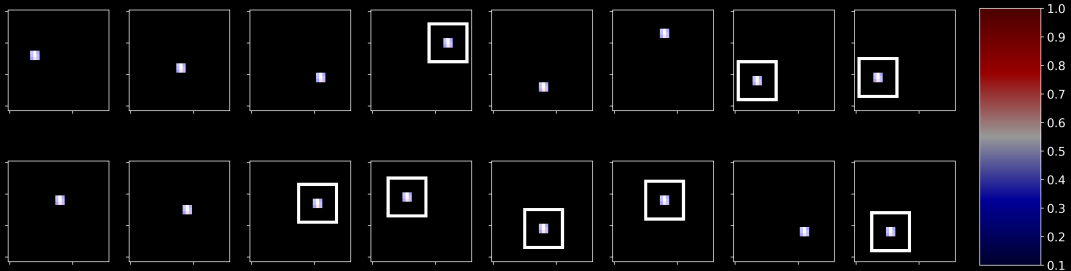
```
class ConvolutionalNetwork(nn.Module):

    def __init__(self, n_filter=8):
        super().__init__()
        layers = [nn.ZeroPad2d(3),
             nn.Conv2d(in_channels=1, out_channels=n_filter, kernel_size=3),
             nn.ReLU(),
             nn.Conv2d(in_channels=n_filter, out_channels=n_filter, kernel_size=3),
             nn.ReLU(),
             nn.Conv2d(in_channels=n_filter, out_channels=n_filter, kernel_size=3),
             nn.ReLU(),
             nn.Conv2d(in_channels=n_filter, out_channels=1, kernel_size=1),
             nn.Sigmoid()]
        self.layers = nn.Sequential(*layers)

    def forward(self, x):
        return self.layers(x)
```

Result

- The receptive field is too small to take the context information into account!
- For each pixel, the context can be at most three pixels away (half the receptive field)
- Here, the border is five pixels away



Dilated network

Question: What is the receptive field of this network?

```
class DilatedNetwork(nn.Module):

    def __init__(self, n_filter=8):
        super().__init__()
        layers = [nn.ZeroPad2d(7),
             nn.Conv2d(in_channels=1, out_channels=n_filter, kernel_size=3, dilation=1),
             nn.ReLU(),
             nn.Conv2d(in_channels=n_filter, out_channels=n_filter, kernel_size=3, dilation=2),
             nn.ReLU(),
             nn.Conv2d(in_channels=n_filter, out_channels=n_filter, kernel_size=3, dilation=4),
             nn.ReLU(),
             nn.Conv2d(in_channels=n_filter, out_channels=1, kernel_size=1),
             nn.Sigmoid()]
        self.layers = nn.Sequential(*layers)

    def forward(self, x):
        return self.layers(x)
```

Dilated network

Question: What is the receptive field of this network? 15

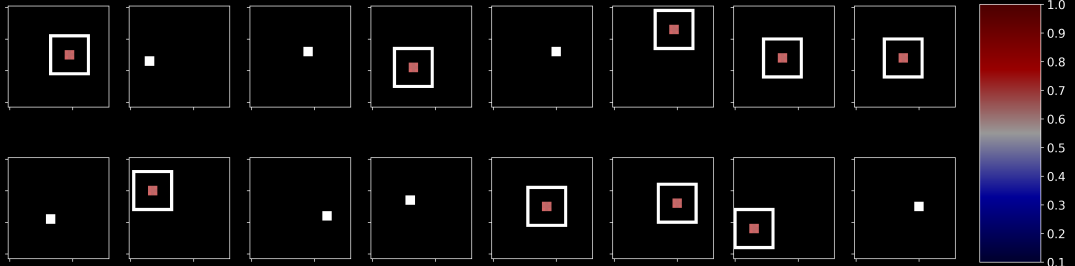
```
class DilatedNetwork(nn.Module):

    def __init__(self, n_filter=8):
        super().__init__()
        layers = [nn.ZeroPad2d(7),
             nn.Conv2d(in_channels=1, out_channels=n_filter, kernel_size=3, dilation=1),
             nn.ReLU(),
             nn.Conv2d(in_channels=n_filter, out_channels=n_filter, kernel_size=3, dilation=2),
             nn.ReLU(),
             nn.Conv2d(in_channels=n_filter, out_channels=n_filter, kernel_size=3, dilation=4),
             nn.ReLU(),
             nn.Conv2d(in_channels=n_filter, out_channels=1, kernel_size=1),
             nn.Sigmoid()]
        self.layers = nn.Sequential(*layers)

    def forward(self, x):
        return self.layers(x)
```

Result

The receptive field is *theoretically large enough* to take the context information into account!



The number of parameters is **exactly** the same in both networks.

Pros and cons of all-convolutional (dilated) networks

Strengths

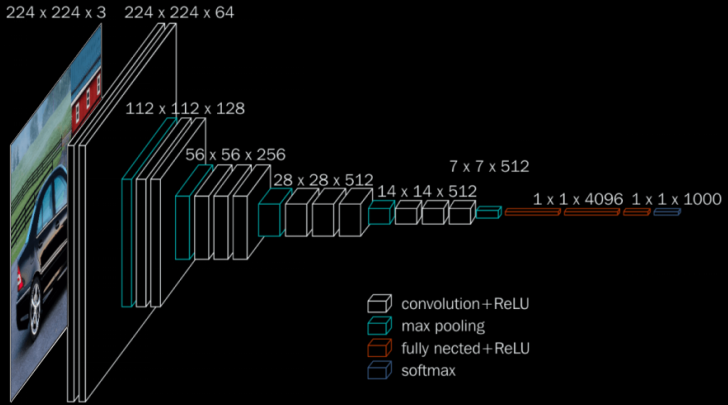
- No loss of resolution, very precise
- Translation equivariant
- The network is independent of image size! (As long as the image is large enough)

Weaknesses

- For large receptive fields, large padding needed
- Memory intensive

Encoder-decoder networks

VGG-16

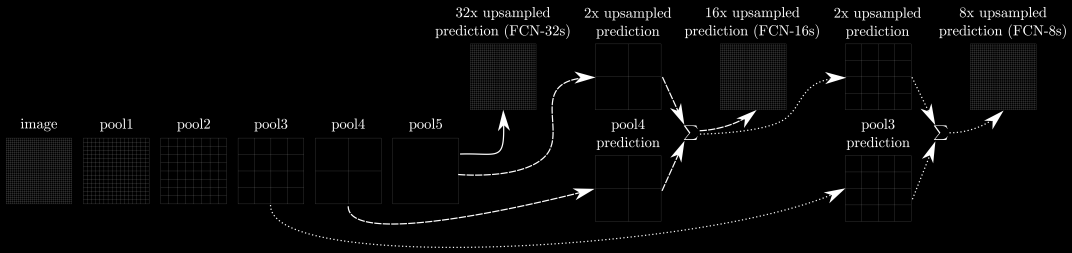


Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014)

Fully convolutional network (a FCN)

Main idea: Modify an image *classification* CNN to perform segmentation

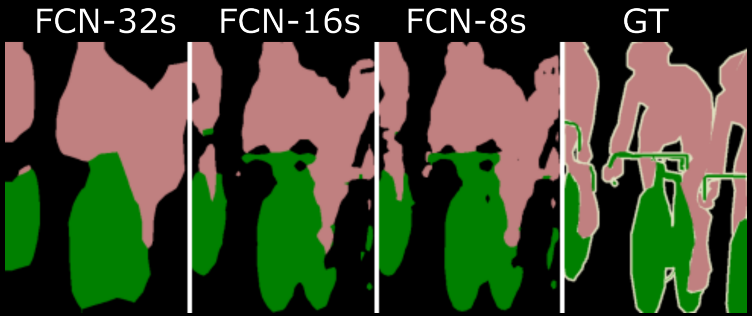
- The last layers are good at detecting *what* is in the image
- The first layers of the model define the *where*



Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation". In: *CVPR*. 2015, pp. 3431–3440

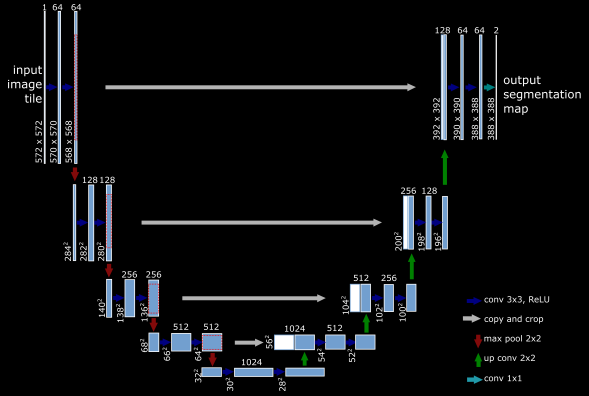
FCN

Combining predictions at multiple resolutions improves segmentation!



U-Net

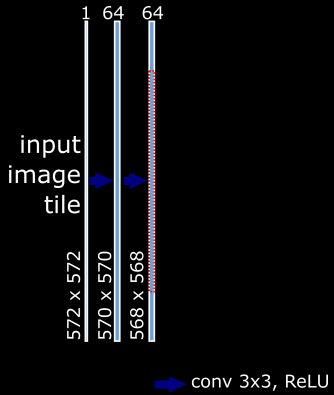
Main idea: An FCN with convolution layers on the upsampled images



Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *MICCAI*. Springer. 2015, pp. 234–241

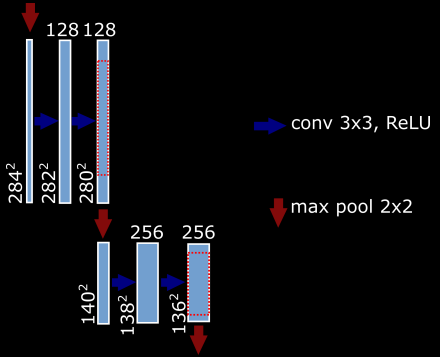
Dissecting a U-Net

- The input is a 2D (could be 3D) image tile
- The first layer convolves from 1 to 64 feature maps
- The second layer convolves from 64 to 64 feature maps
- 3×3 *valid* convolution is performed



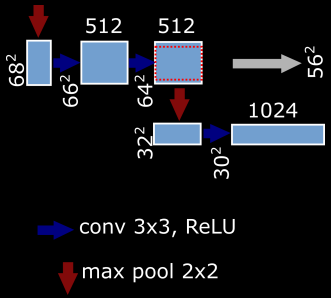
Dissecting a U-Net

- Max-pooling reduces the image size by a factor two
- 3×3 convolution is performed in each layer
- The number of feature maps doubles each *level*



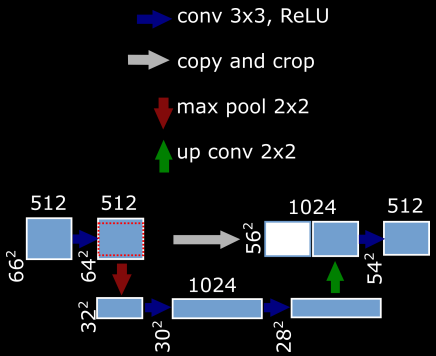
Dissecting a U-Net

- The lowest level has small feature maps (30×30 pixels)
- These feature maps have many channels (1024)



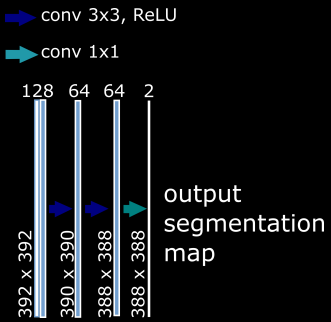
Dissecting a U-Net

- The *expanding* path starts
- Small feature maps are upsampled
- The number of feature maps decreases
- Maps from the *contracting path* are concatenated
- Concatenation combines the *what* and the *where*

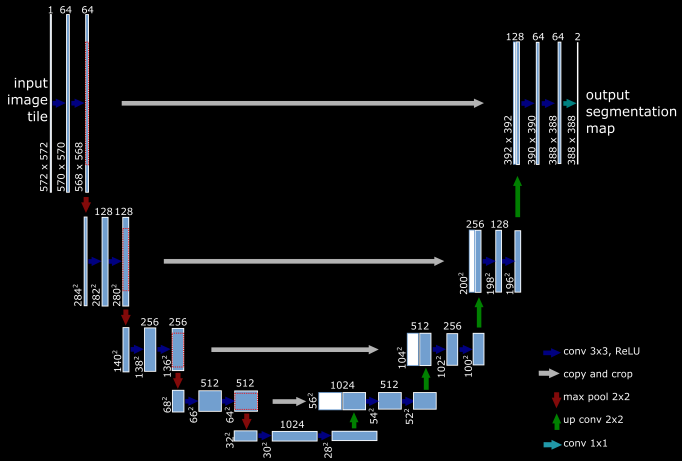


Dissecting a U-Net

- The feature maps are back at original resolution
- A final 1×1 *bottleneck* layer is applied
- The model has 2 output feature maps/labels



U-Net


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

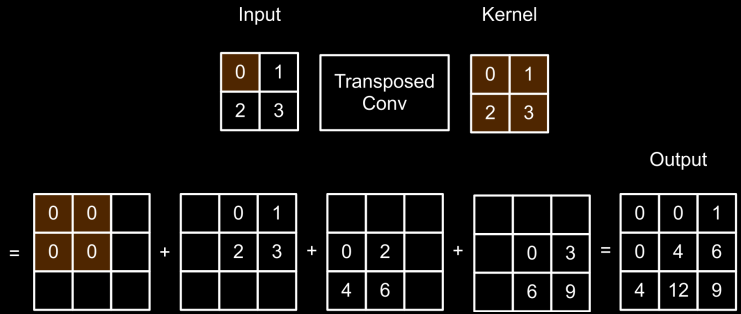
Parameters

Question How many trainable parameters do you think this U-Net has?

- 3000
- 30,000
- 300,000
- 3,000,000
- 30,000,000

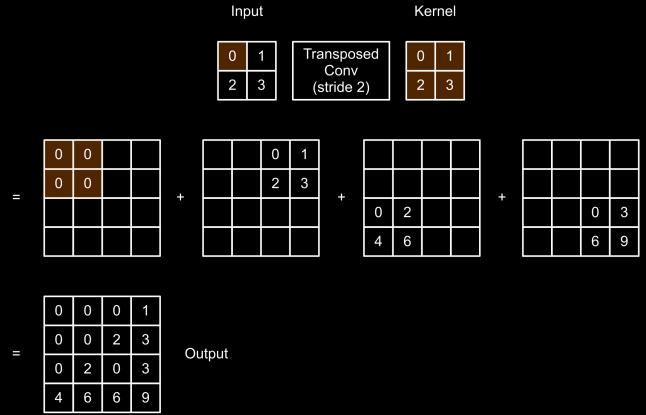
Up conv?

- How to make an image larger?
- Also called transposed convolution, fractionally strided or deconvolution



Up conv?

With stride 2, the stride is in the output, not the input



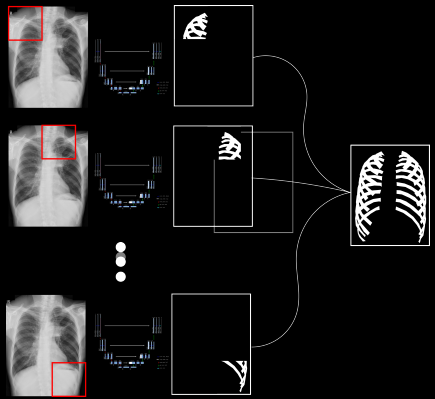
https://d2l.ai/chapter_computer-vision/transposed-conv.html

Variants of U-Nets

- There are many different variants of U-Nets
  - Swap out the layers for blocks
  - Add residual connections
  - More layers, fewer layers
  - Dilation
  - Dense blocks
  - Different activation functions
  - Deeper, shallower
  - Etc., etc.
- Important: most 'modern' U-Nets don't crop
- More levels $\rightarrow$ larger inputs, larger receptive fields

Practical use

- FCN/U-Net requires fixed size inputs
- Images may have different shapes
- Train with cropped images
- Test with shift-and-stitch/sliding window
- There can be severe boundary effects
- Combine multiple predictions
- The more overlap, the better (and slower)
- In the limit, back to sliding window



Pros and cons of encoder-decoder networks

Pros

- Intuitive
- Rapidly gives decent segmentations
- Widely available/easy to implement (in PyTorch/Tensorflow/Keras)

Weaknesses

- An encoder-decoder is not translationally equivariant
- Can be computationally demanding (especially in 3D)
- Demands a fixed size input image
- Boundary effects

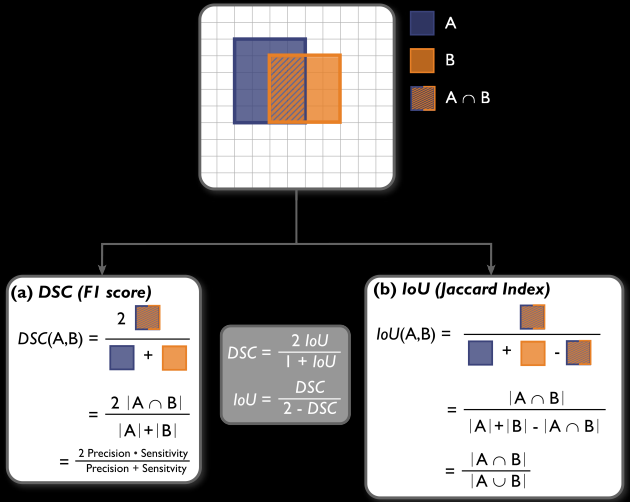
Metrics

Metrics

What is a good segmentation?

- Ideally, **qualitative** metrics by experts
  - Is the segmentation anatomically correct?
  - Is the segmentation topologically correct?
  - How useful is the segmentation in the clinic?
  - How valuable is the segmentation for downstream tasks
  - Smoothness, quality in specific locations, etc..
- In practice, **quantitative** metrics
  - **Overlap** How well does the *predicted* foreground mask correspond to the *target* background?
  - **Distance** How far from the *target* boundaries are the *predicted* boundaries
  - **Problem-specific** metrics, e.g., volume

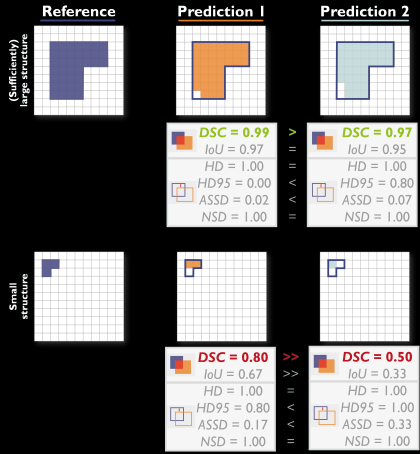
Overlap metrics



Reinke et al. "Common limitations of image processing metrics: A picture story". In: *arXiv preprint arXiv:2104.05642* (2021)

Pitfalls

- Not choosing the right metric for your problem
- E.g., Dice similarity coefficient for very small lesions → detection problem

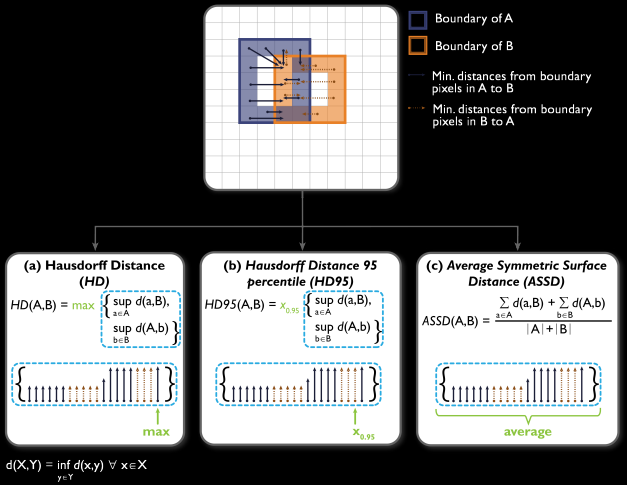


Pitfalls

Dice scores do not take shape into account

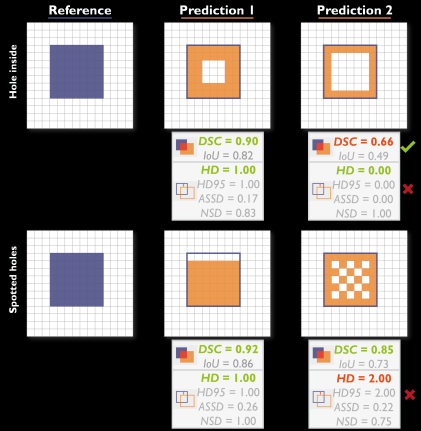
	DSC	IoU	HD	HD95	ASSD	NSD
Reference						
Prediction 1	DSC = 0.6	IoU = 0.4	HD = 1.4	HD95 = 1.3	ASSD = 0.9	NSD = 1.0
Prediction 2	DSC = 0.6	IoU = 0.4	HD = 3.6	HD95 = 3.1	ASSD = 1.0	NSD = 0.7
Prediction 3	DSC = 0.6	IoU = 0.4	HD = 3.0	HD95 = 2.0	ASSD = 0.7	NSD = 0.8
Prediction 4	DSC = 0.6	IoU = 0.4	HD = 2.2	HD95 = 2.0	ASSD = 0.7	NSD = 0.8
Prediction 5	DSC = 0.6	IoU = 0.4	HD = 2.0	HD95 = 1.2	ASSD = 0.8	NSD = 0.9

Distance metrics



Pitfalls

Distance metrics don't see holes. Metric can be perfect, but segmentation anatomically implausible



Message

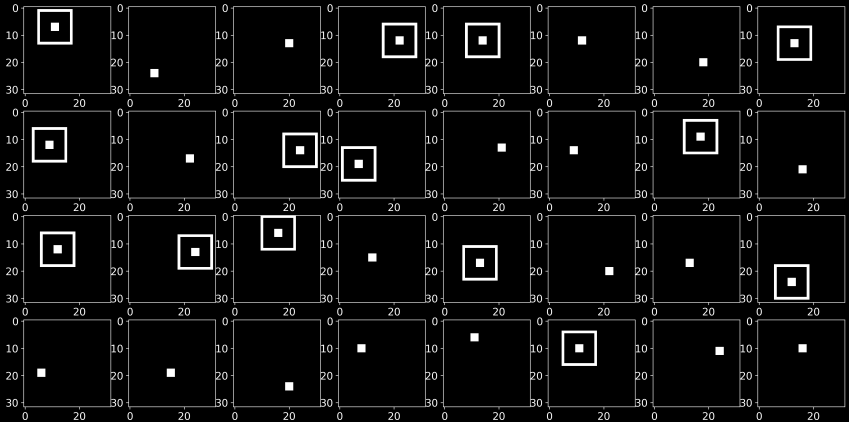
- Be aware of pitfalls of metrics (read Reinke paper)
- Always use a combination of metrics
- Try to think of problem-specific losses (e.g, volumes, number of components, no holes, ..)
- **Don’t trust metrics, look at your results**

Losses

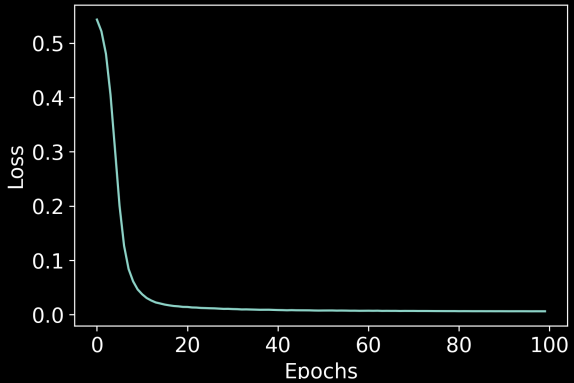
How to optimize a segmentation network?

- Optimizing a classification problem with thousands/millions of samples/pixels at the same time
- Recall binary cross-entropy from last lecture? $J(y, \hat{y}) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$
- We usually compute the loss per pixel/voxel, then *aggregate* over images and mini-batches
- Works for two classes, can be extended to multiple classes

Binary cross-entropy: demo

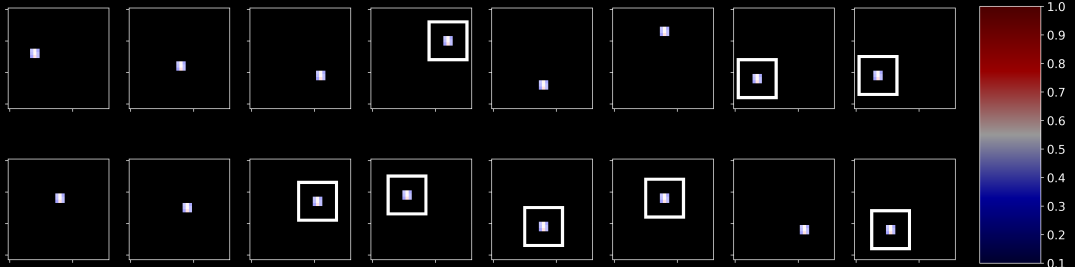


Binary cross-entropy



Binary cross-entropy

Binary cross-entropy loss is **close to zero**, while segmentation performance is very poor



Class imbalance

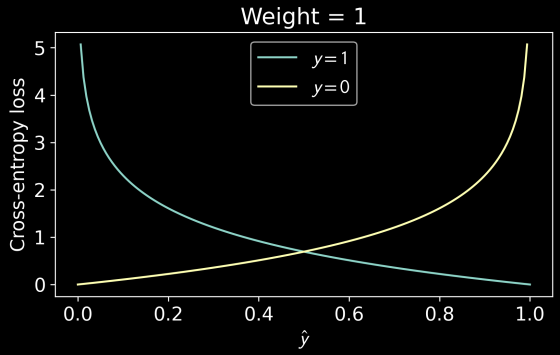
This problem has strong class imbalance:

- One in two images contains a ‘malign lesion‘
- In that image, 9 out of 32×32 pixels are positive
- This means that 0.4% of pixels is positive, and 99.6% negative
- Just setting all pixels to negative would result in a loss close to zero

Class weights

One solution: add class weights γ_i that are inversely proportional to class occurence

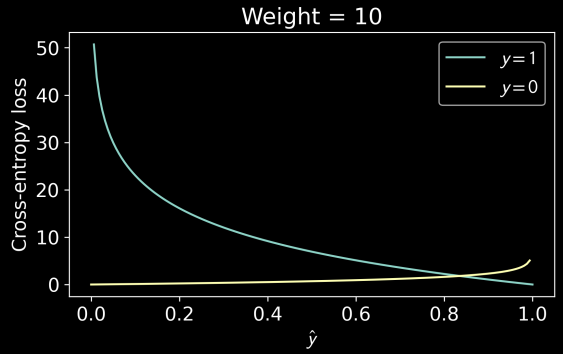
$$J(\mathbf{y}, \widehat{\mathbf{y}}) = -\sum_i \gamma_i y_i \log(\hat{y}_i)$$



Class weights

One solution: add class weights γ_i that are inversely proportional to class occurence

$$J(\mathbf{y}, \widehat{\mathbf{y}}) = -\sum_i \gamma_i y_i \log(\hat{y}_i)$$

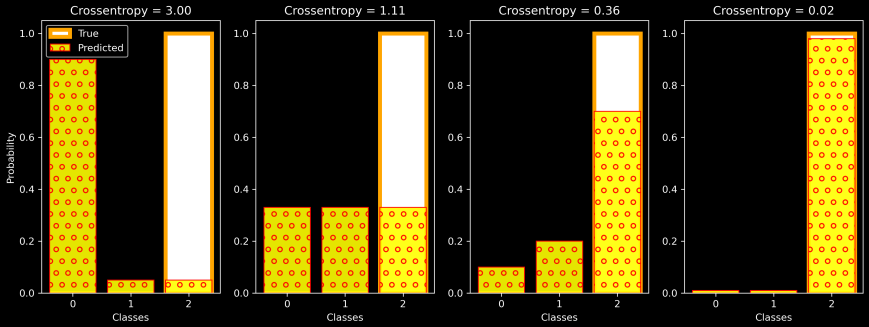


Binary vs. multiclass segmentation

Two things change if there are more than two classes

- *Output layer* Softmax instead of sigmoid
  - Sigmoid: $\sigma(z) = \frac{1}{1+\exp^{-z}}$
  - Softmax: $\sigma(\mathbf{z}) = \frac{\exp^{z_i}}{\sum_{j=1}^{K} \exp^{z_j}}$
  - Both sum outputs to 1, with values between 0 and 1
  - In multiclass classification, apply the softmax to *each* pixel/voxel individually
- *Loss function* Categorical cross-entropy instead of binary cross-entropy
  - Binary cross-entropy: $J(y, \hat{y}) = -y \log(\hat{y}) - (1-y) \log(1-\hat{y})$
  - Categorical cross-entropy: $J(\mathbf{y}, \widehat{\mathbf{y}}) = -\sum_i y_i \log(\hat{y}_i)$
  - Generalization to more than two classes

Categorical cross-entropy



Overlap loss

Alternative solution: directly optimize overlap

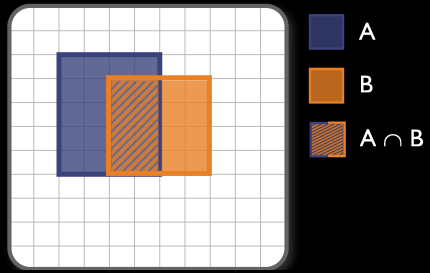
- High overlap $\rightarrow$ Low loss
- Less sensitive to imbalance
- But, less friendly gradients $\rightarrow$ combine CE and overlap loss
- There are many other loss functions for image segmentation*
- Dice loss

* Shruti Jadon. "A survey of loss functions for semantic segmentation". In: *CIBCB*. IEEE. 2020, pp. 1–7

Dice loss

Dice metric: $D = \frac{2|A \cap B|}{|A|+|B|} = \frac{2\,TP}{2\,TP+FN+FP}$

Dice loss: $D = 1 - \frac{2\sum_i A_i B_i + 1}{\sum_i A_i + \sum_i B_i + 1}$



Multiclass vs. multilabel segmentation

Distinction

- **Multiclass** classification is a classification task with more than two classes. Each sample can only be labeled as one class.
- **Multilabel** classification is a classification task labeling each sample with 0 to n labels from n possible classes. This can be thought of as predicting properties of a sample that are not mutually exclusive.

Examples of multilabel problems

- Overlapping structures in a 2D projection image (x-ray)
- Nested structures like brain tumors
- Lesions in an organ

Question: What would be the output function in a multilabel task?

Summary

We have discussed

- Convolution variants
- Segmentation architectures
- Metrics, losses, and pitfalls

Don’t forget

- Tutorial on Thursday: segmentation

Materials on Canvas