Imperial College
London

Machine Learning for Imaging

Image Segmentation

Ben Glocker
b.glocker@imperial.ac.uk

Common image analysis tasks

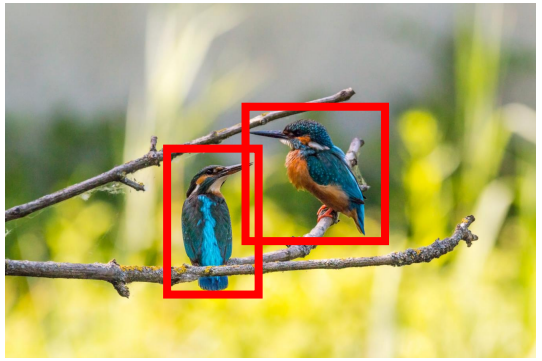
Image Classification

Output: Category (e.g., “bird”)

Object Detection

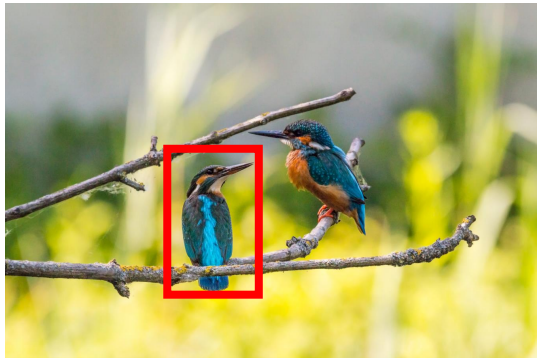
Output: Coordinates (e.g., centroid)

Object Localisation



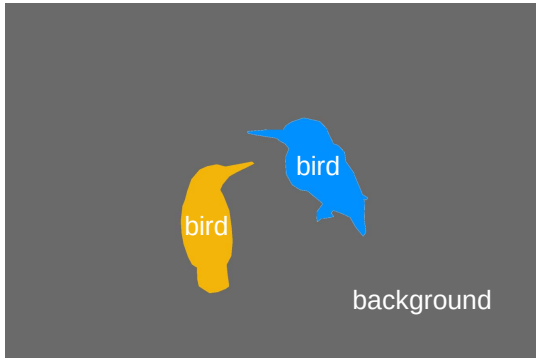
Output: Coordinates (e.g., bounding box)

Object Recognition



Output: Category (e.g., “kingfisher”)

Semantic Segmentation



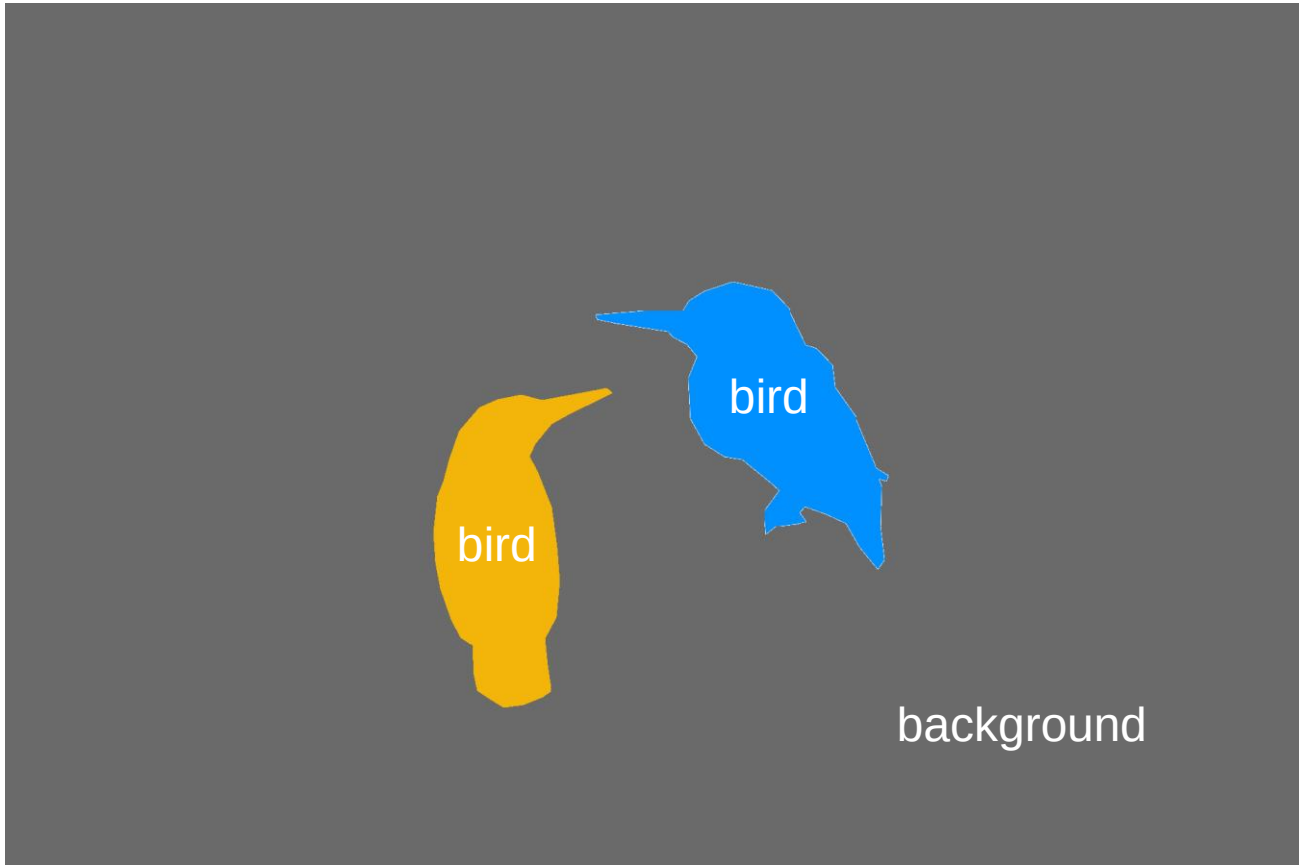
Output: Labelmap

Image Captioning

Output: Text
(e.g., “two birds sitting on a branch”)

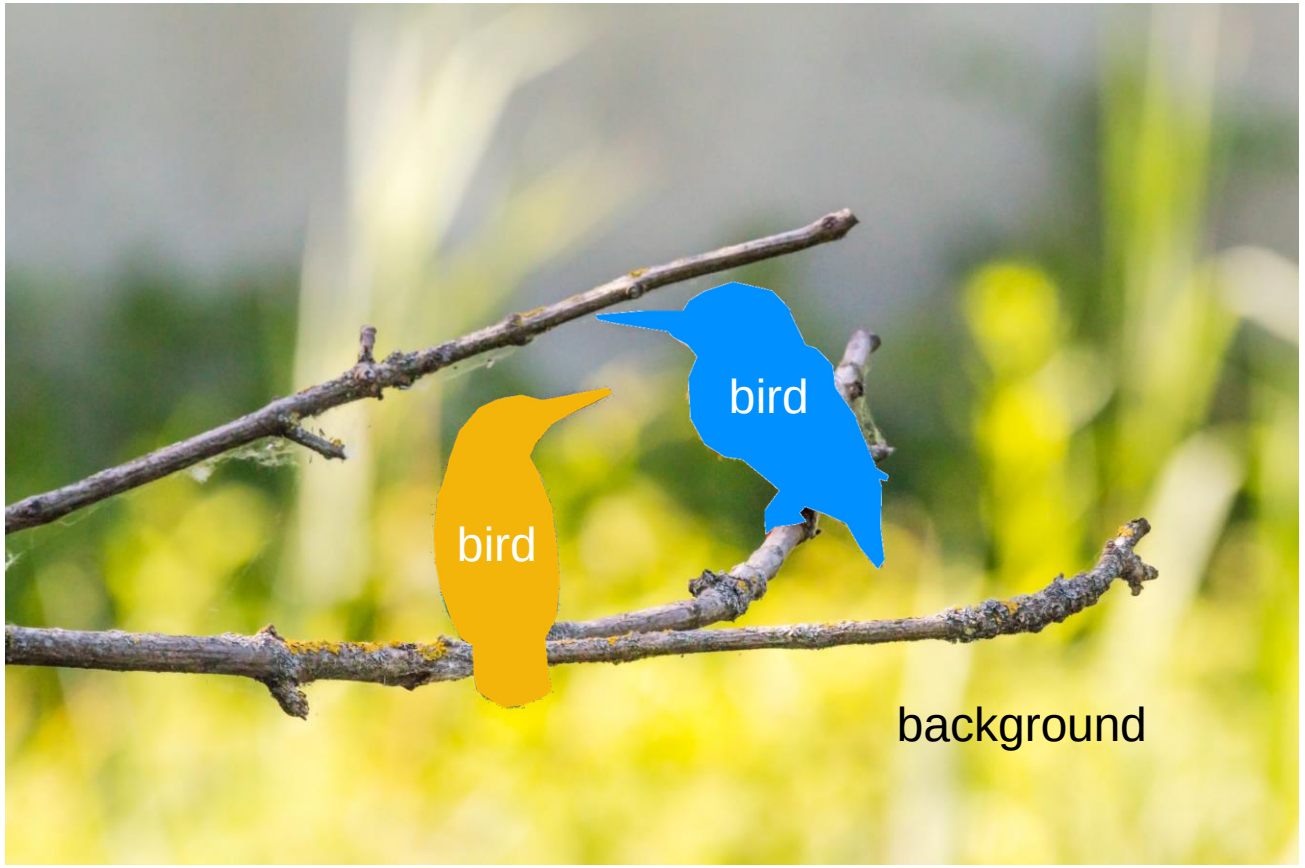
Semantic segmentation

Semantic segmentation



Output: Labelmap

Semantic segmentation



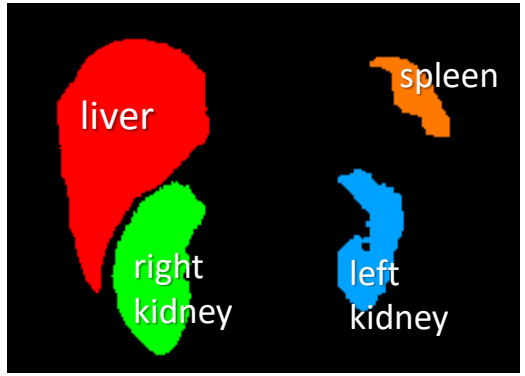
Output: Labelmap

Semantic segmentation

- In semantic segmentation, a segmented region is also assigned a **semantic meaning**
- This contrasts with segmentation based on ‘pure’ clustering of the image into coherent regions
- Here, we are mostly interested in semantic segmentation

raw image

segmentation/clustering

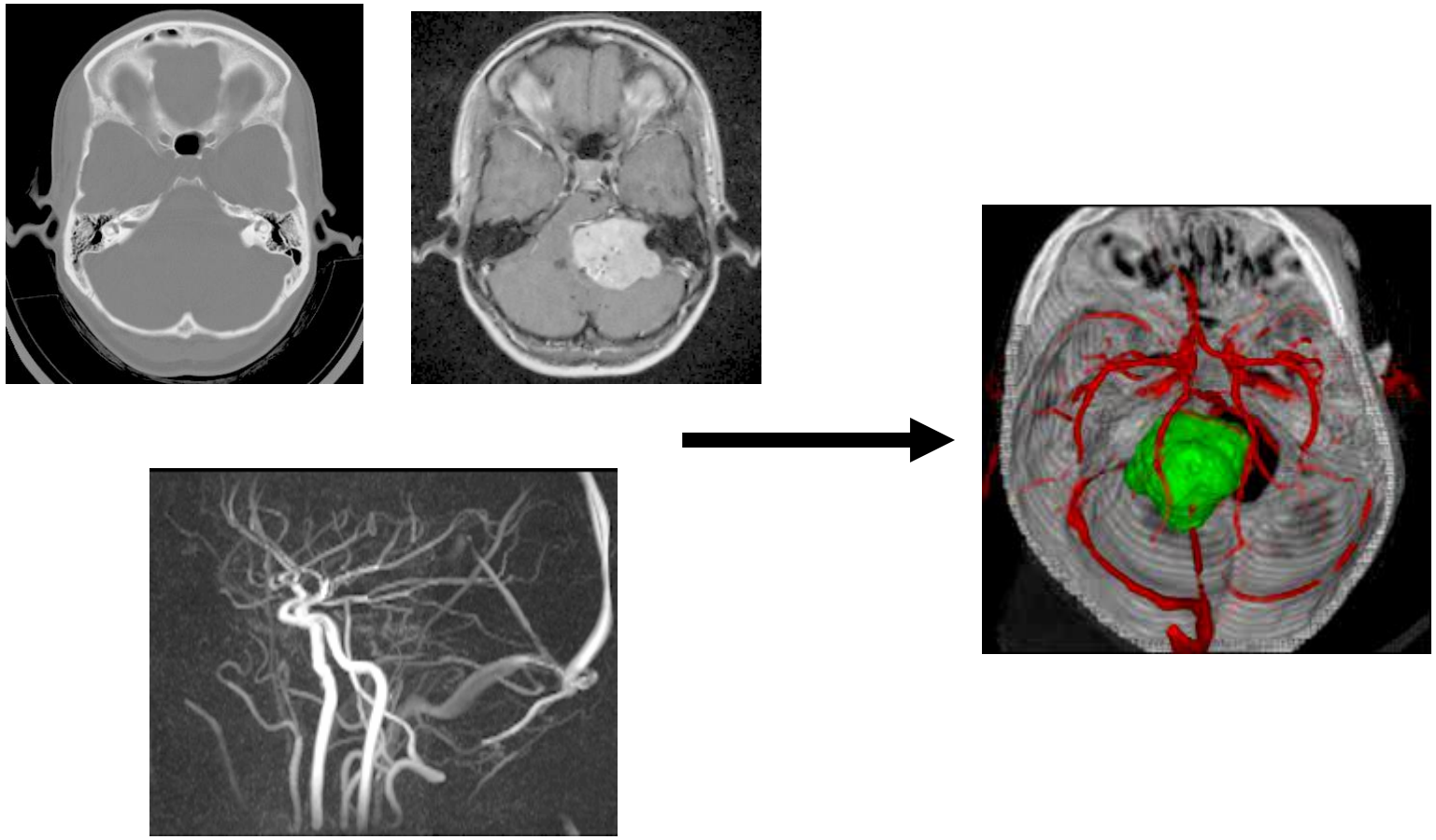


semantic segmentation

Segmentation in medical imaging

- **Image segmentation is useful for...**
  - conducting **quantitative analyses**, e.g. measuring the volume of the ventricular cavity.
  - determining the precise **location and extent** of an organ or a certain type of tissue, e.g. a tumour, for treatment such as radiation therapy.
  - creating **3D models** used for **simulation**, e.g. generating a model of an abdominal aortic aneurysm for simulating stress/strain distributions.

Example



C. Ruff, 1995

Challenges for image segmentation

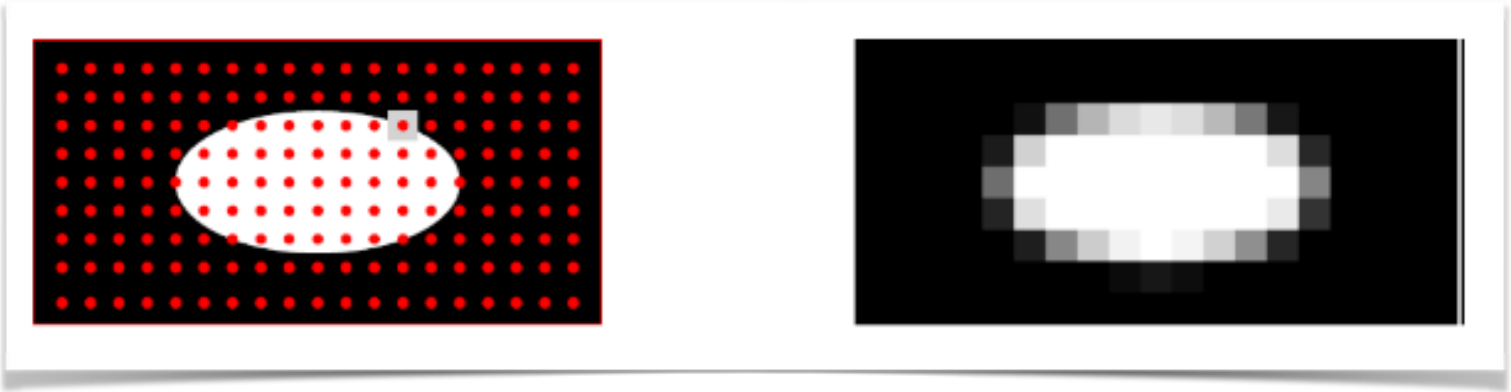
- There are several effects in imaging that may hamper the application of segmentation algorithms
  - noise,
  - partial volume effects,
  - intensity inhomogeneities, anisotropic resolution,
  - imaging artifacts,
  - limited contrast,
  - morphological variability,
  - and many more …

Noise

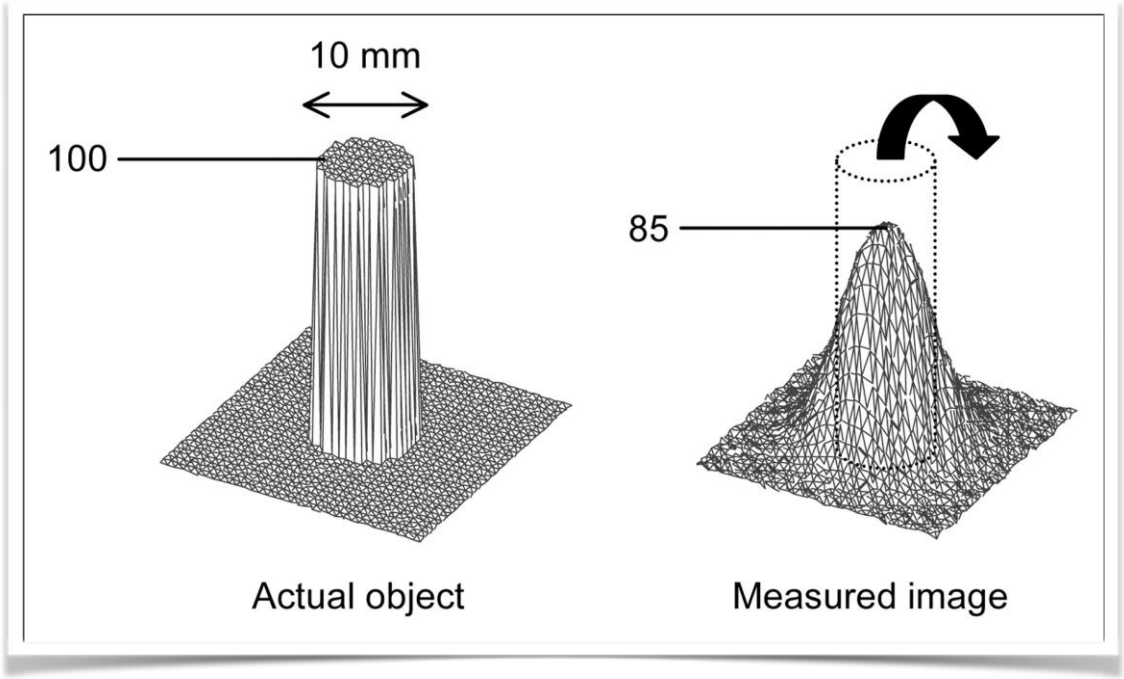
- Example: Transcranial 2D ultrasound image of the brain

Partial volume effects

- Due to the coarse sampling (red dots) the resulting image shows partial volume effects at the boundary of the white “organ”
- Both tissue types (black and white) contribute to the intensity value of the generated image (right) due to the relatively large influence area of each pixel (gray square in left image)

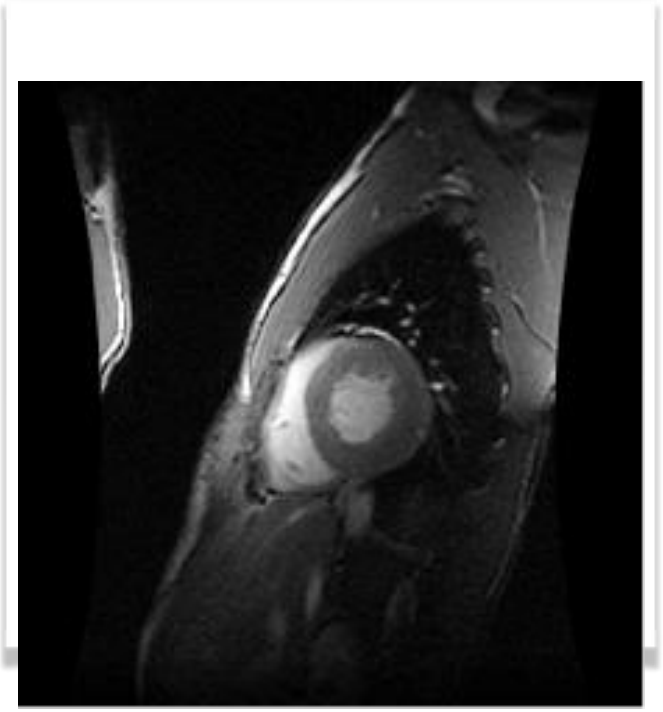


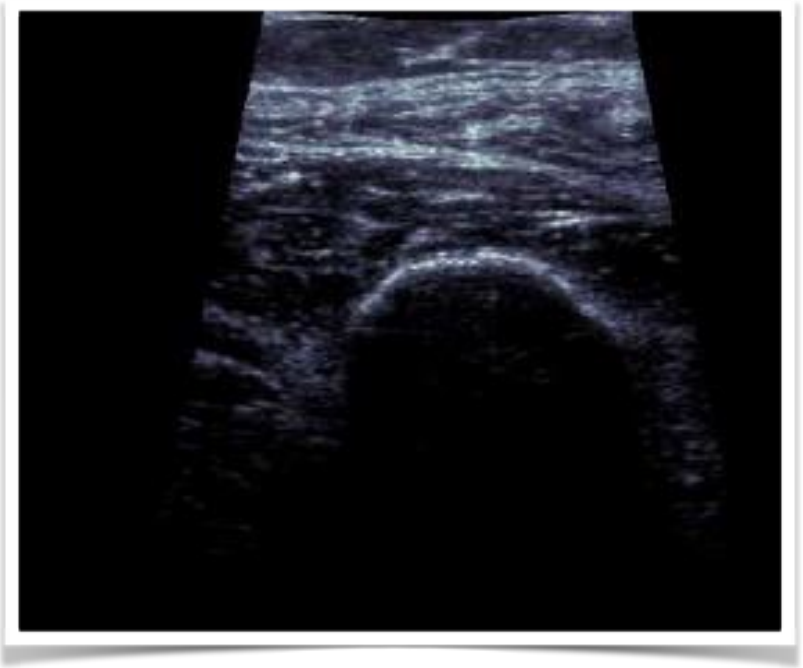
Partial volume effects



Intensity inhomogeneities

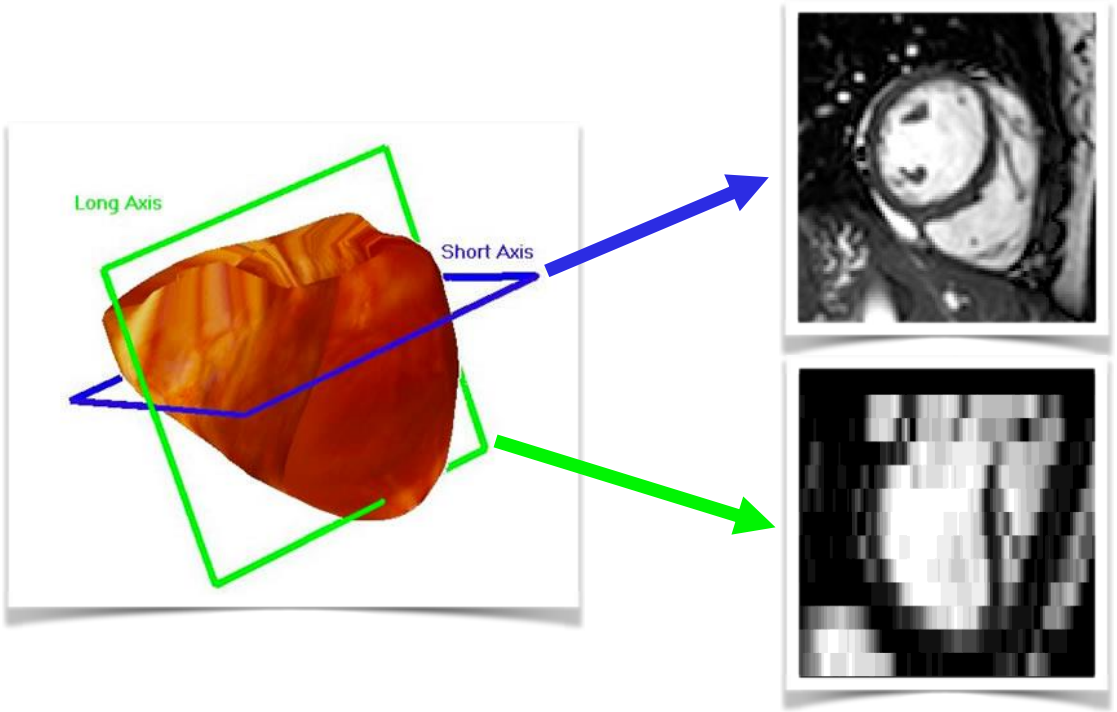
- Typical for MRI and ultrasound imaging data



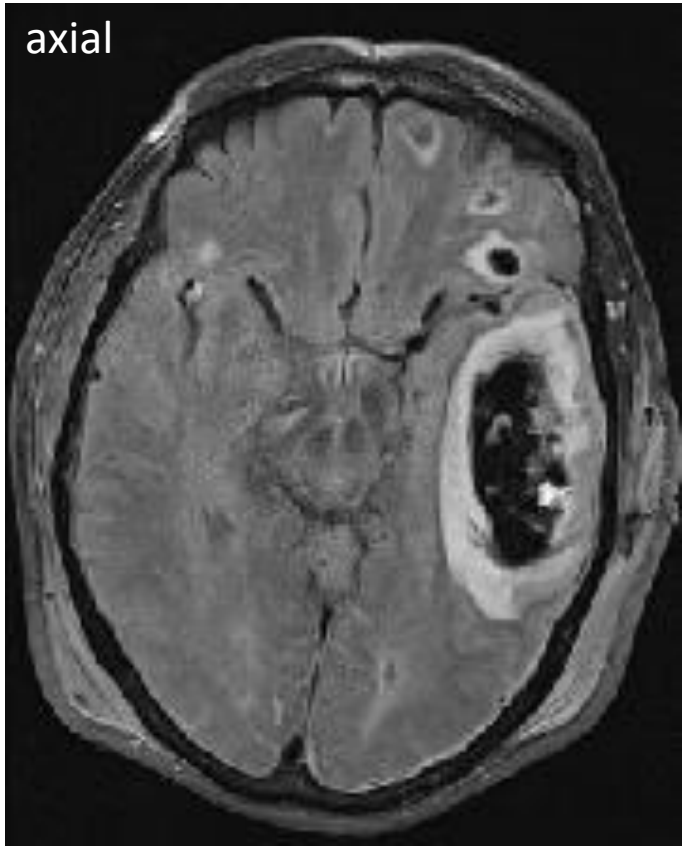


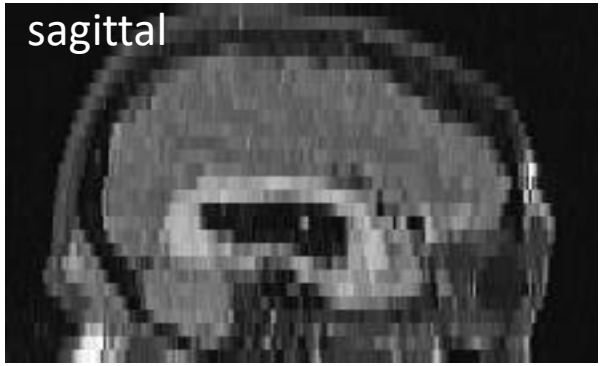
Anisotropic resolution

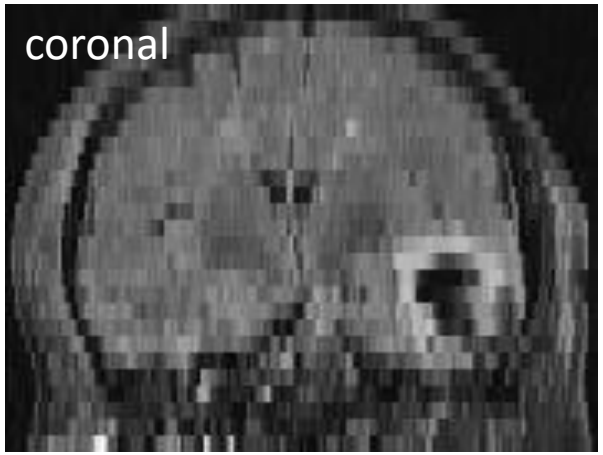
- Almost all 3D imaging modalities, such as CT or MRI suffer from this problem
  - For example, (dynamic) MRI acquisition of short axis slices of the left ventricle may have intra-slice resolution of 1.3mm and an inter-slice resolution of 8mm



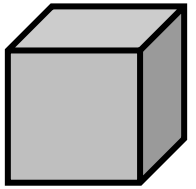
Anisotropic resolution



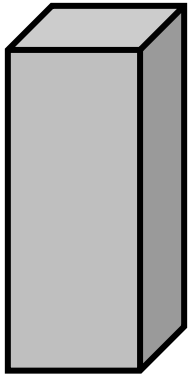




Voxel size: 0.7 x 0.7 x 5mm



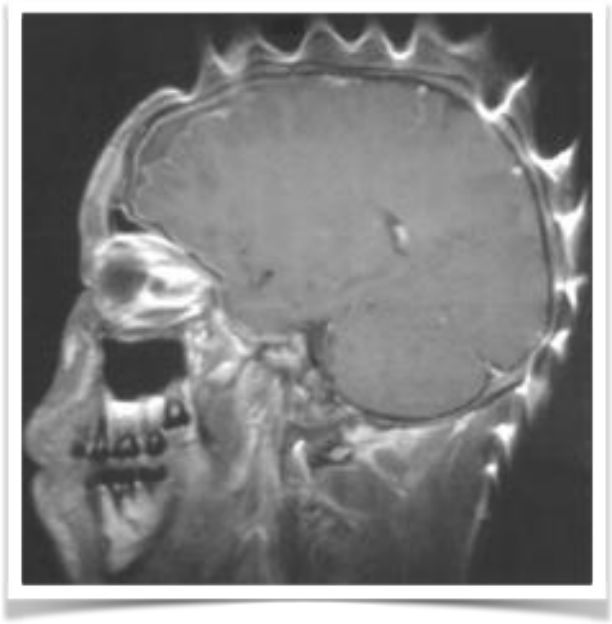
isotropic



anisotropic

Imaging artifacts

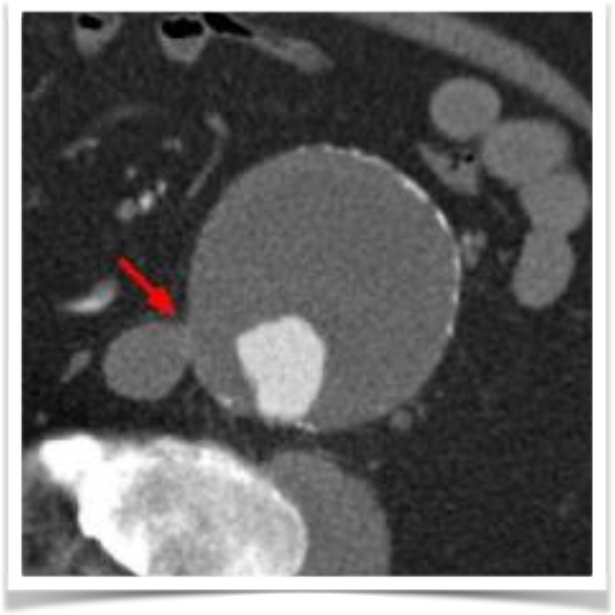
CT image showing streak artifacts caused by metal implants.

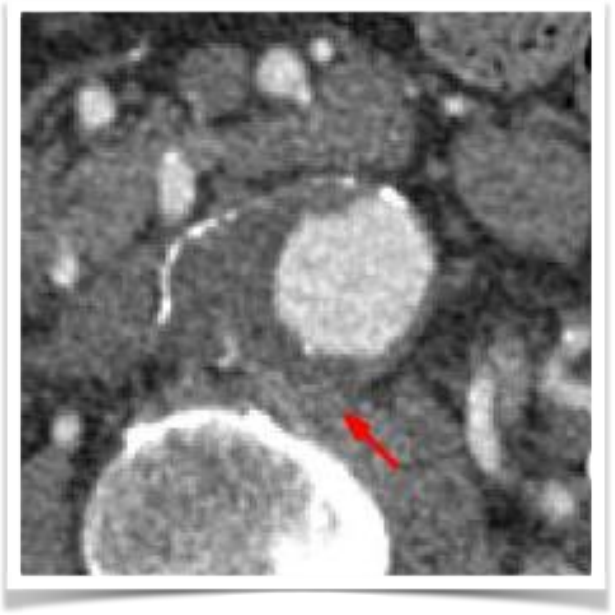


T1-weighted MRI image showing susceptibility artifacts caused by iron oxide particles suspended in the beeswax dressing in the hair of the patient.

Limited contrast

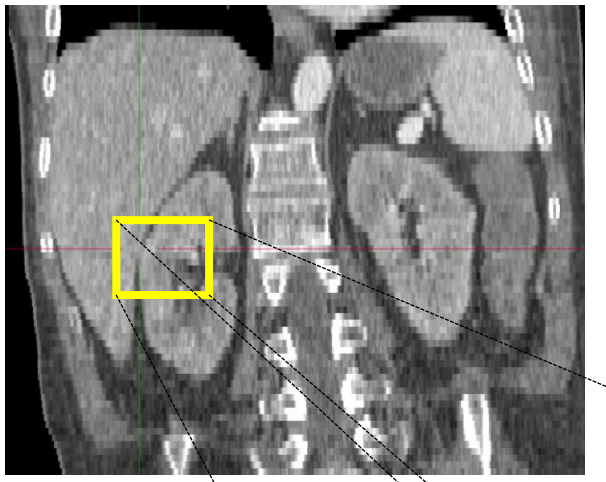
- Different tissues can have similar physical properties and thus similar intensity values
- Purely intensity-based algorithms are prone to fail or “leak” into adjacent tissue



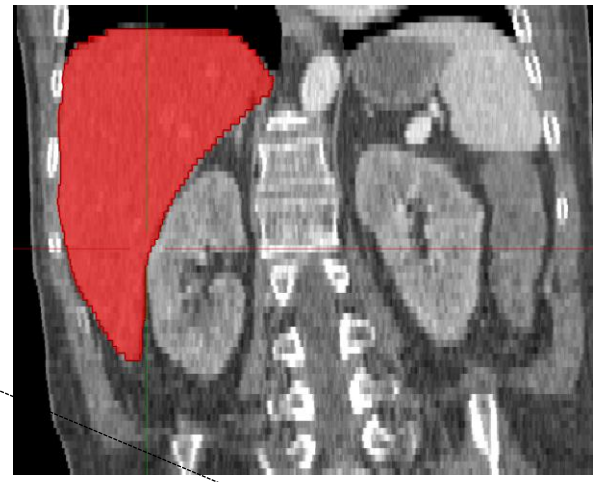


Example: Thrombus in the wall of abdominal aortic aneurysms is often hard to distinguish from the surrounding tissue (modality: CTA).

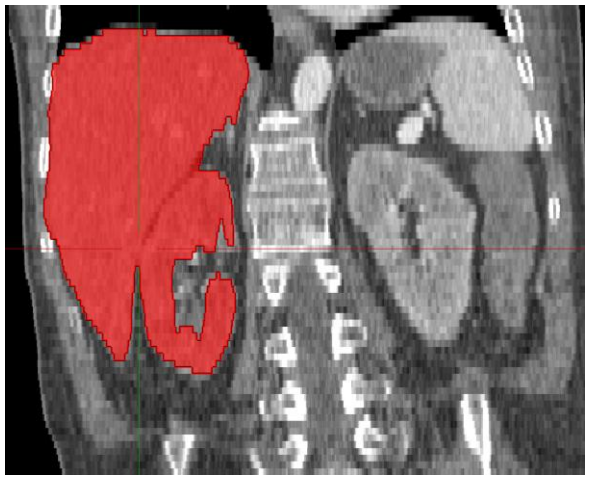
Limited contrast



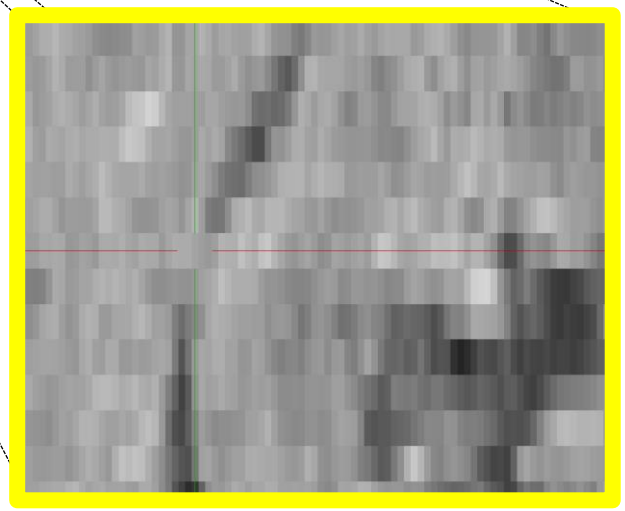
CT abdominal image



manual liver segmentation

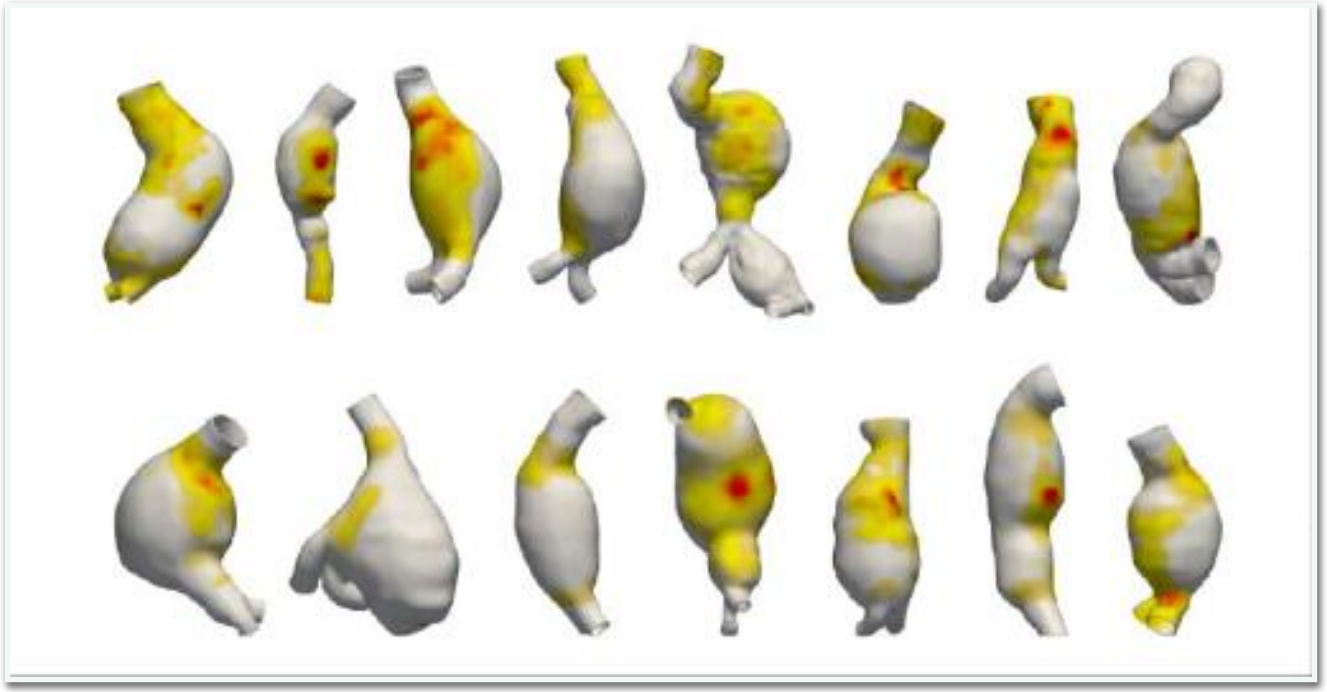


region growing with leakage



Morphological variability

- Morphological variability makes it hard to incorporate meaningful prior information or useful shape models



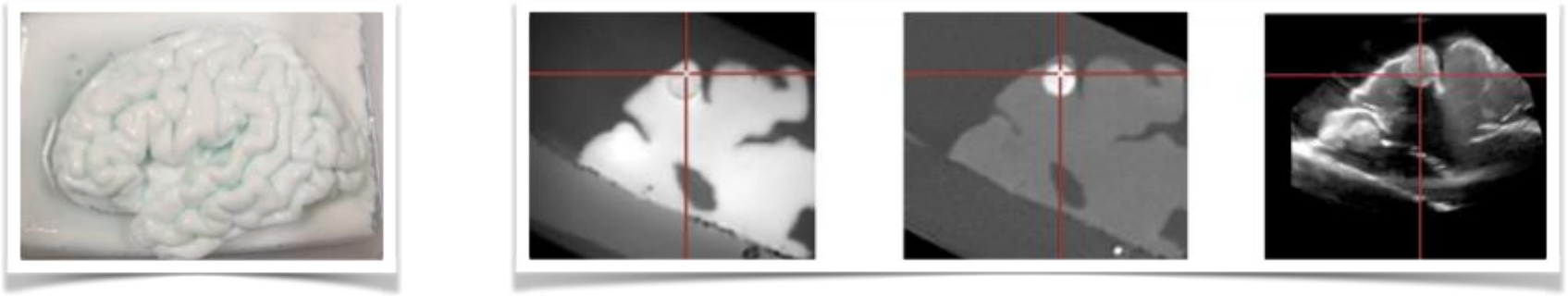
Example: A collection of abdominal aortic aneurysms acquired with PET-CT and colored by FDG-18 uptake values.

Segmentation evaluation

- Ground truth
- Gold standard
- Performance measures
  - precision/recall
  - sensitivity/specificity
  - accuracy
  - overlap
  - distance measures

Ground truth

- Reference or standard against a method can be compared, e.g. the optimal transformation, or a true segmentation boundary
- GT is usually only available for:
  - Synthetic or simulated phantoms (e.g., computer phantoms)
  - Physical phantom (e.g., gel phantoms)

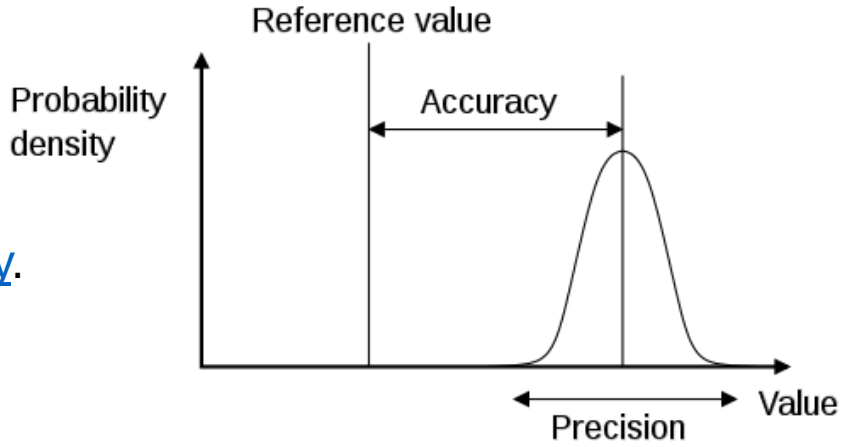


PVA mould of a brain model, scanned with MR,CT and US.

Gold standard

- Expert (often referred to as *gold* standard)
  - Manual segmentation by human observer (e.g. experienced clinician)
- Disadvantage
  - Requires training and is tedious and time-consuming
  - Intra-observer variability (disagreement between same observer on different occasions)
  - Inter-observer variability (disagreement between observers)
- Remedy
  - Human observer can perform segmentation repeatedly
  - Multiple experts can perform segmentations
  - Agreement or disagreement can be quantified

How to assess performance?



Precision

- is a description of random errors, a measure of statistical variability.
- the repeatability, or reproducibility of the measurement

Accuracy has two definitions

- More commonly, it is a description of systematic errors, a measure of statistical bias; as these cause a difference between a result and a "true" value, ISO calls this *trueness*.
- Alternatively, ISO defines accuracy as describing a combination of both types of observational error above (random and systematic), so high accuracy requires both high precision and high trueness

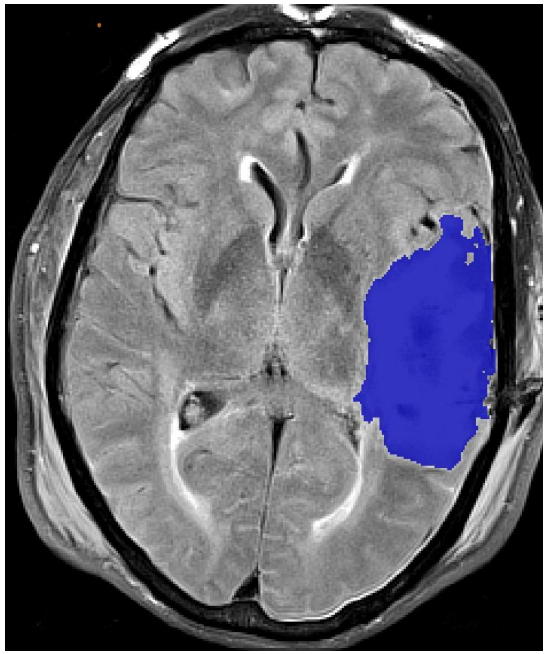
Robustness

- refers to the degradation in performance with respect to varying noise levels or other imaging artefacts

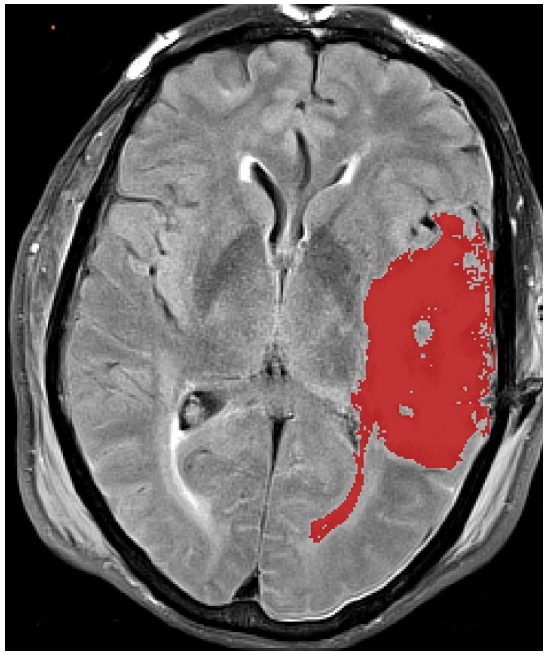
Performance measures

- Assume we have an automated segmentation algorithm: how do we assess ‘how good it is’?
- There are several ways of quantifying segmentation performance

MR image



Gold standard

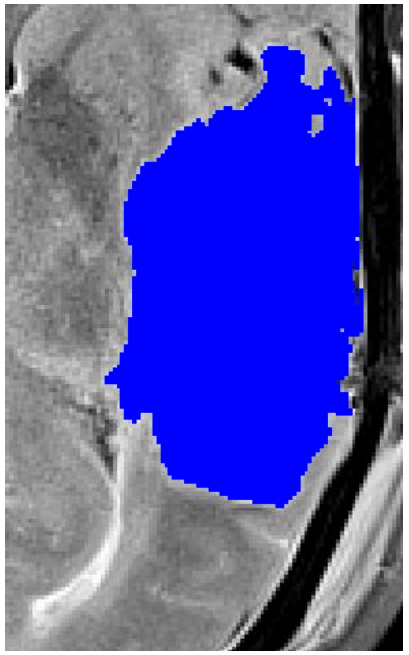


Auto Segmentation

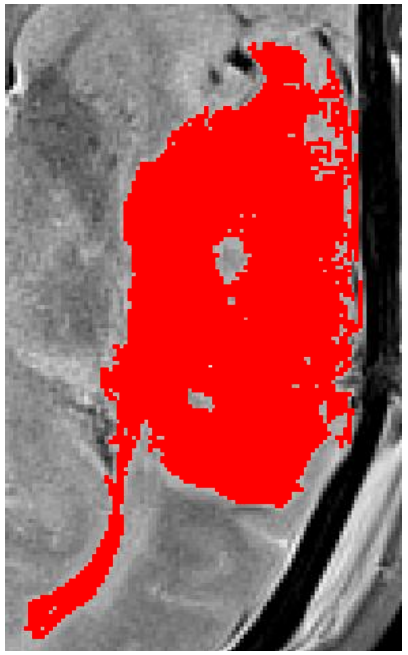
Performance measures

- Assume we have an automated segmentation algorithm, how do we assess ‘how good it is’?
- There are several ways of quantifying segmentation performance

MR image



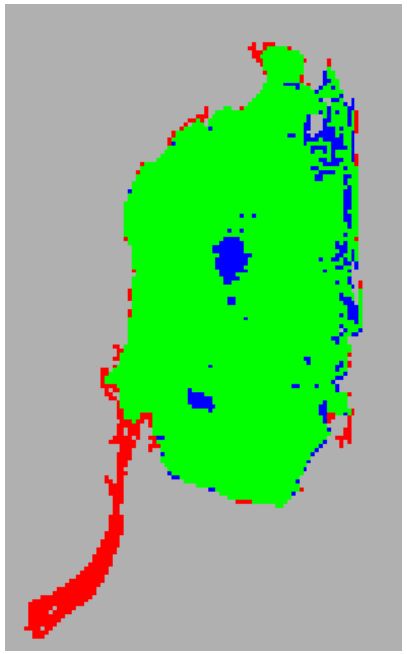
Gold standard



Auto Segmentation

Performance measures

- Measure performance of an automated method in terms of **agreement** of its result with a reference gold standard

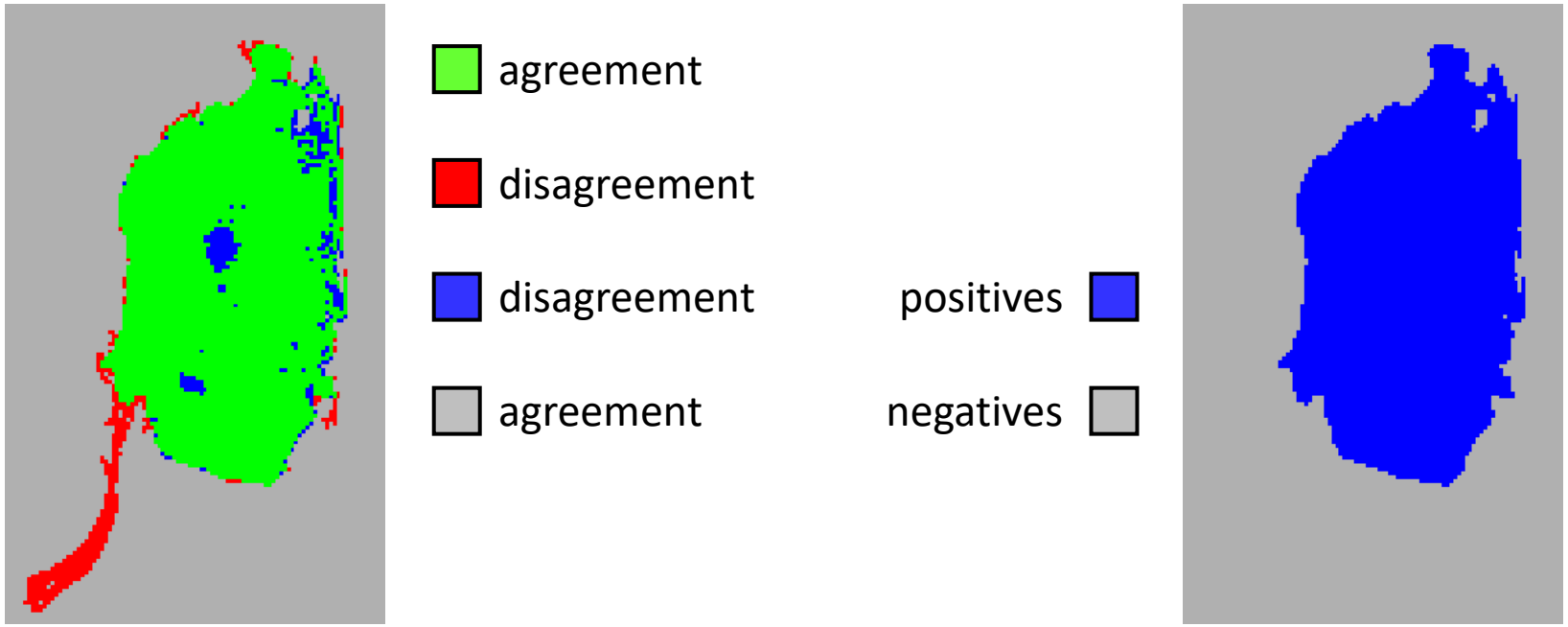


Gold standard

Auto Segmentation

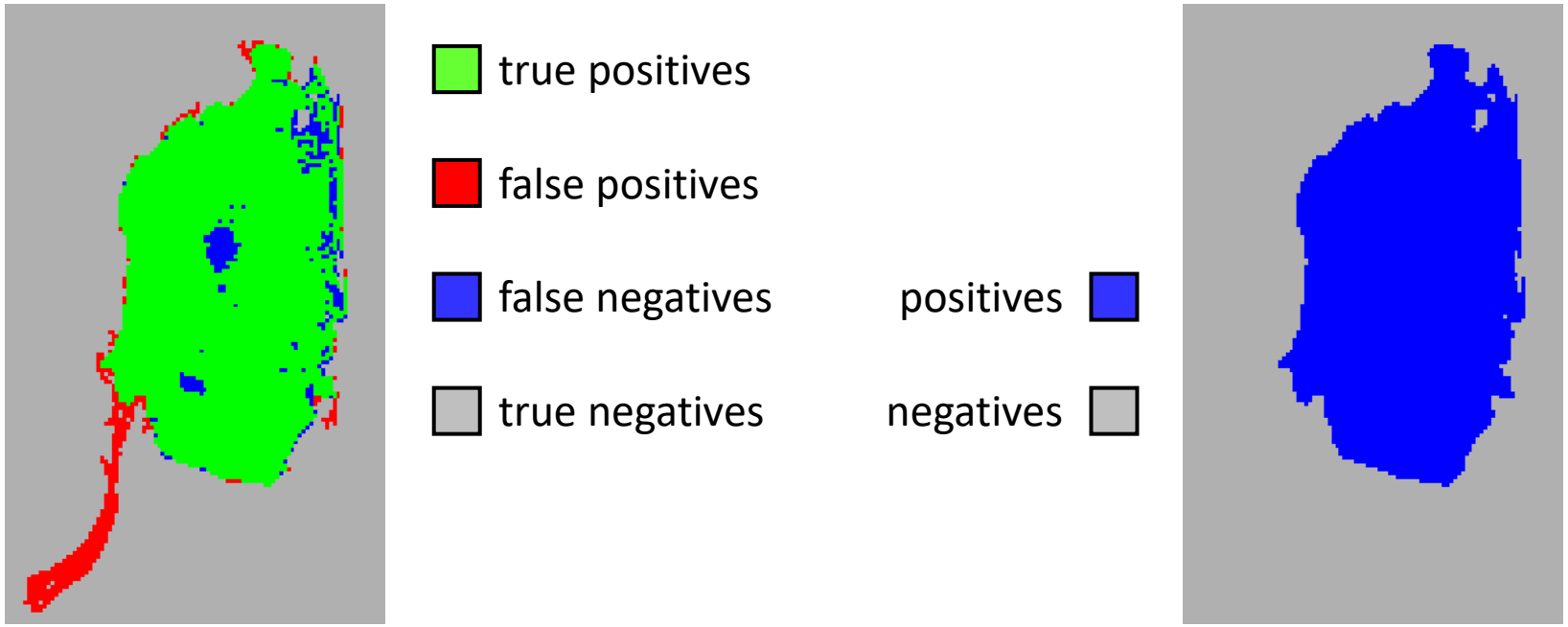
Performance measures

- Measure performance of an automated method in terms of **agreement** of its result with a reference gold standard



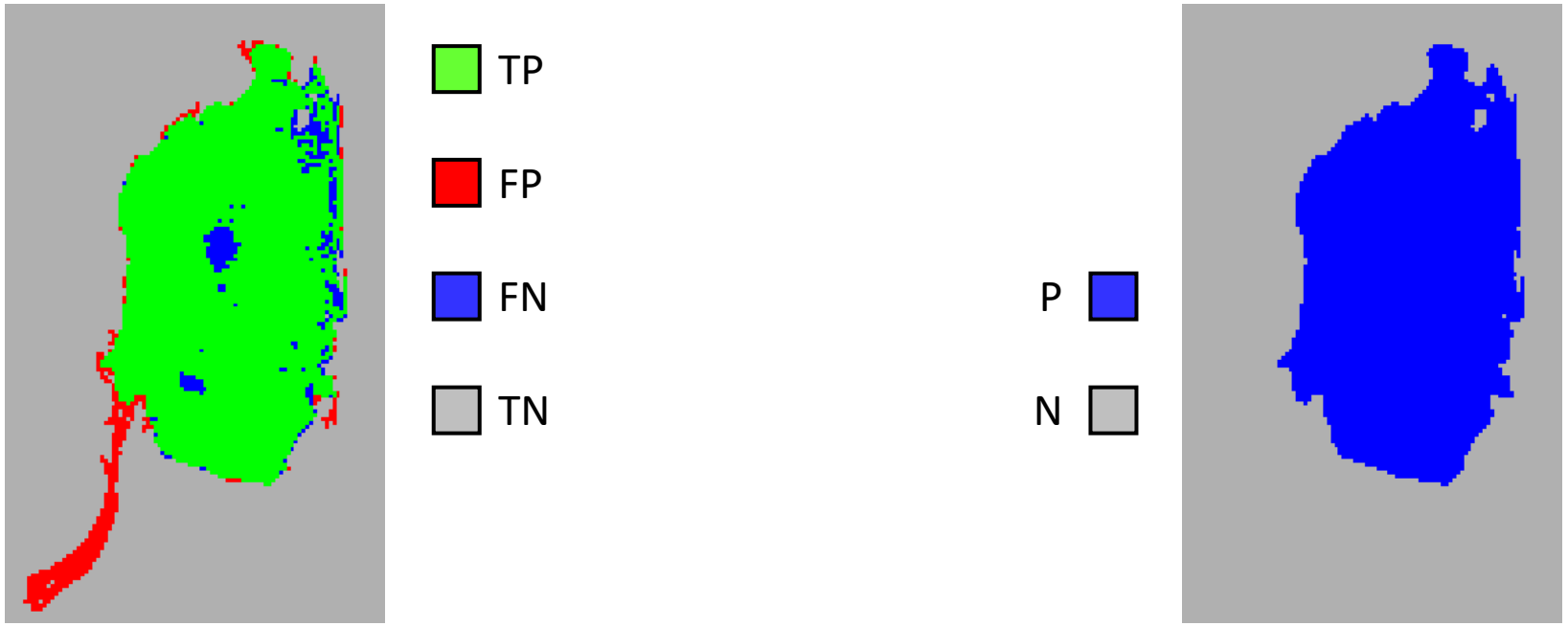
Performance measures

- Measure performance of an automated method in terms of **agreement** of its result with a reference gold standard



Performance measures

- Measure performance of an automated method in terms of **agreement** of its result with a reference gold standard



Confusion matrix

condition positive (P)
the number of real positive cases in the data
condition negatives (N)
the number of real negative cases in the data

true positive (TP)
eqv. with hit
true negative (TN)
eqv. with correct rejection
false positive (FP)
eqv. with false alarm, Type I error
false negative (FN)
eqv. with miss, Type II error

		True condition	
	Total population	Condition positive	Condition negative
Predicted condition	Predicted condition positive	**True positive**	**False positive**, Type I error
	Predicted condition negative	**False negative**, Type II error	**True negative**

Accuracy, precision, recall, ...

accuracy

$$ACC = \frac{TP+TN}{P+N},\ P = TP + FN,\ N = TN + FP$$

precision or **positive predictive value**

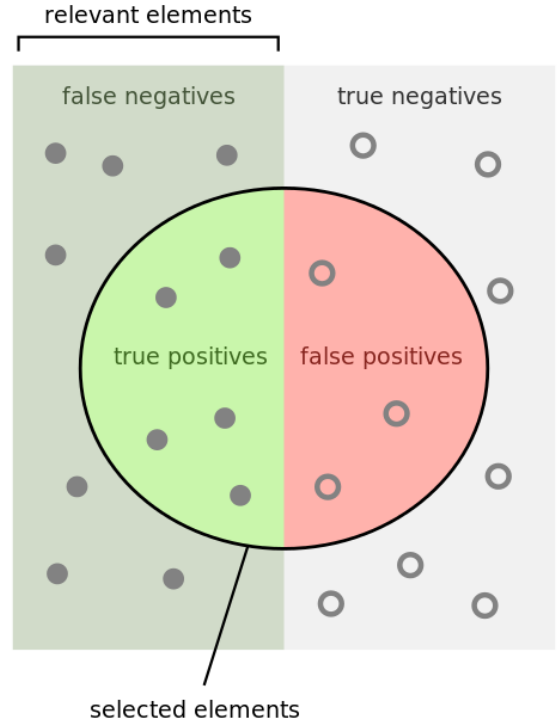
$$PPV = \frac{TP}{TP + FP}$$

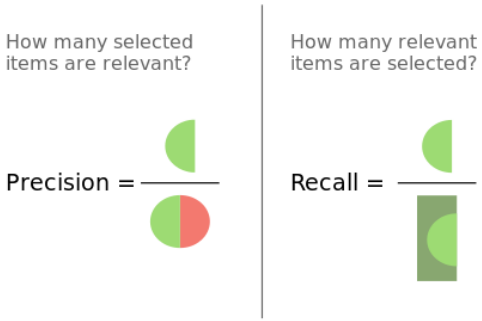
recall, sensitivity, **hit rate** or **true positive rate**

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN}$$

specificity or **true negative rate**

$$TNR = \frac{TN}{N} = \frac{TN}{TN + FP}$$



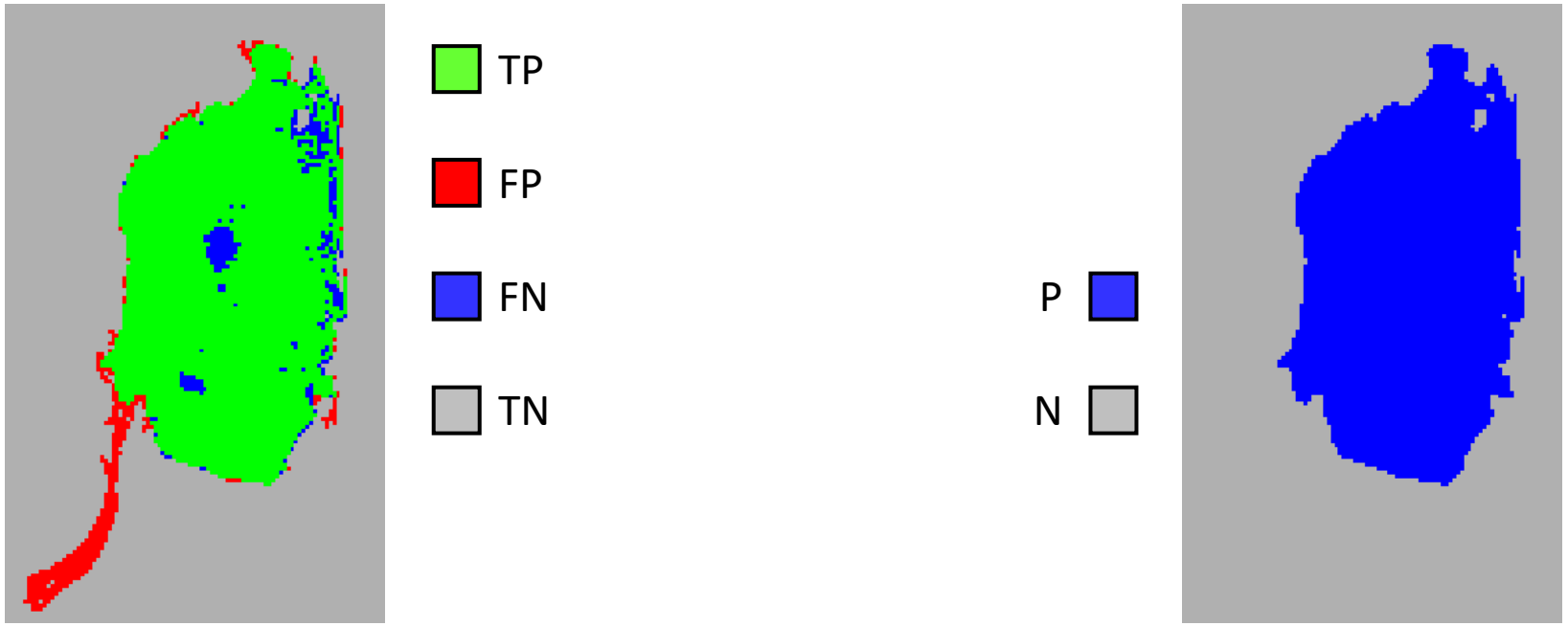


F1 score

precision $PPV = \frac{TP}{TP+FP}$ **recall** $TPR = \frac{TP}{TP+FN}$

F1 score
is the harmonic mean of precision and recall

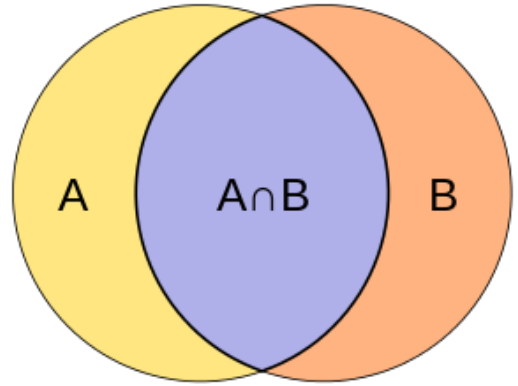
$$F_1 = 2\frac{PPV \cdot TPR}{PPV + TPR} = \frac{2TP}{2TP + FP + FN}$$



Overlap measures

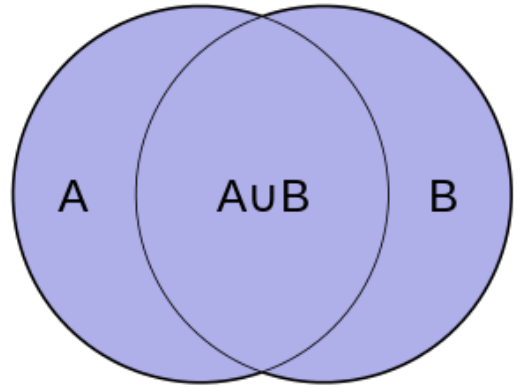
- Measure agreement by looking at overlap of reference and predicted segmentation (e.g., auto segmentation)
- **Jaccard Index**
  aka **Intersection over Union (IoU)**
  aka **Jaccard Similarity Coefficient**

$$JI = \frac{|A \cap B|}{|A \cup B|}$$



- **Dice's Coefficient**
  aka **Sørensen Index**
  aka **Dice Similarity Coefficient (DSC)**

$$DSC = \frac{2|A \cap B|}{|A| + |B|}$$



Dice similarity coefficient

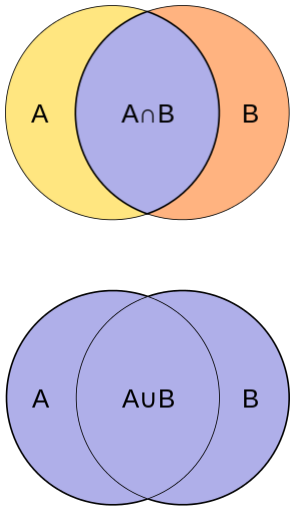
- Most widely used measure for evaluating segmentation
- Assume A is the reference, and B the prediction

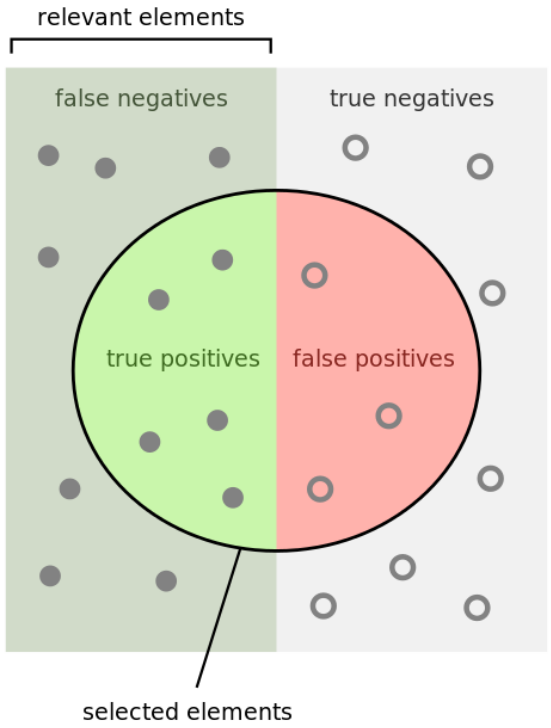
$$DSC = \frac{2|A \cap B|}{|A| + |B|}$$

- $|A| = TP + FN$
- $|B| = TP + FP$
- $|A \cap B| = TP$

$$DSC = \frac{2TP}{2TP + FP + FN} = F_1$$

DSC is equivalent to F1 score!





Other measures

volume similarity

$$VS = 1 - \frac{\big||A| - |B|\big|}{|A| + |B|} = 1 - \frac{|FN - FP|}{2TP + FP + FN}$$

surface distance measures

- **Hausdorff distance**

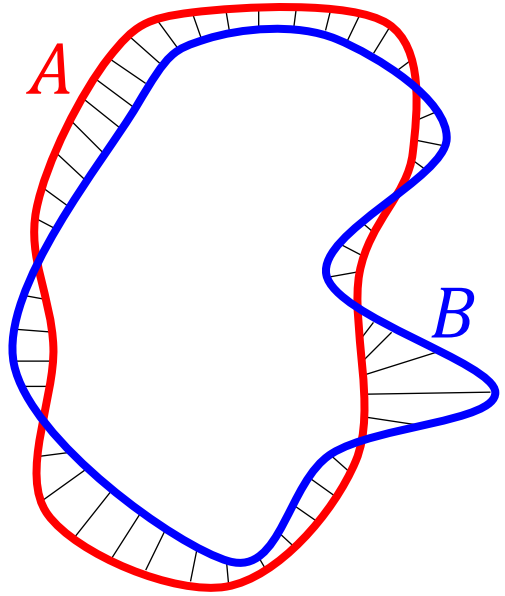
$$HD = \max\big(h(A,B), h(B,A)\big)$$

$$h(A,B) = \max_{a \in A} \min_{b \in B} \|a - b\|$$

- **(symmetric) average surface distance**

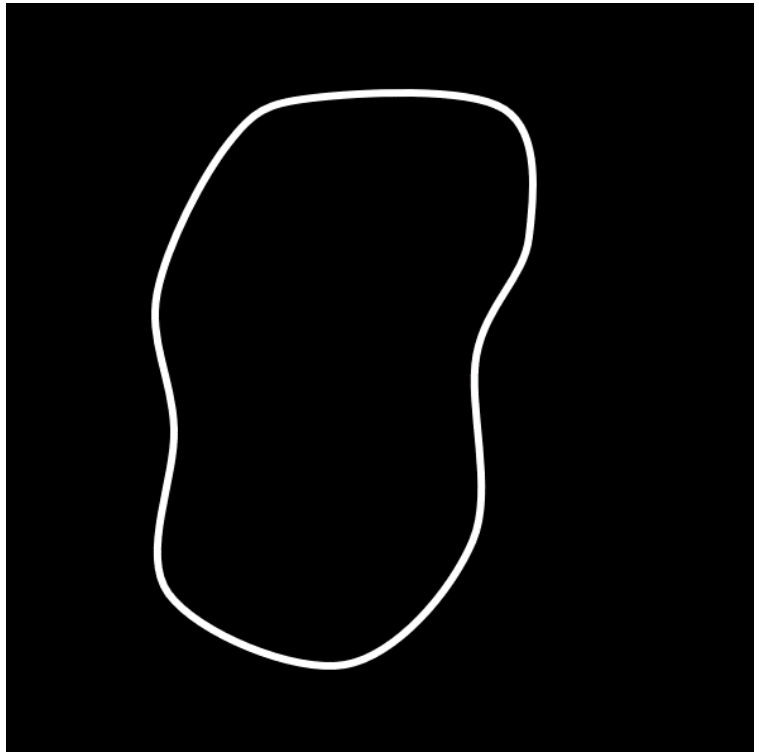
$$ASD = \frac{d(A,B) + d(B,A)}{2}$$

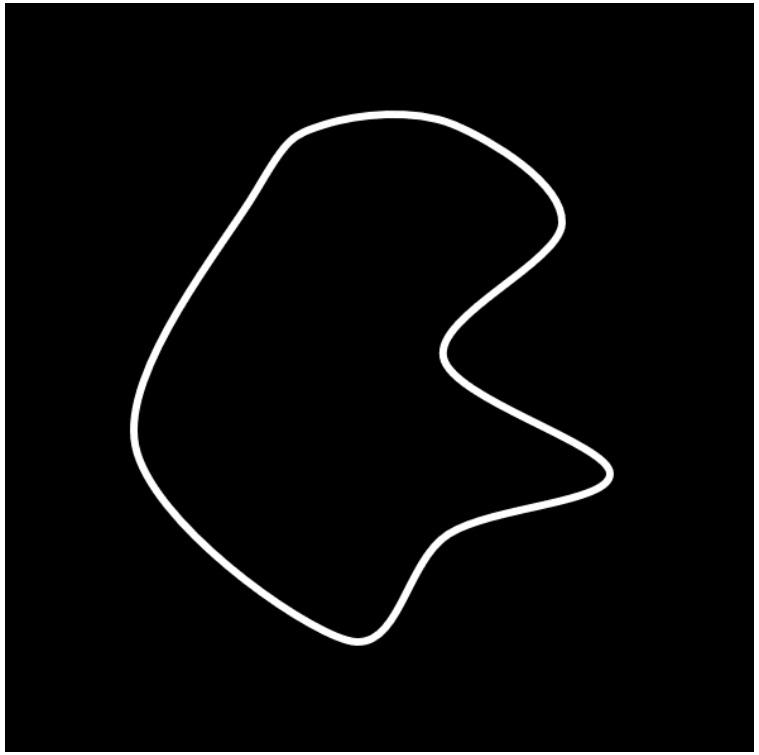
$$d(A,B) = \frac{1}{N} \sum_{a \in A} \min_{b \in B} \|a - b\|$$



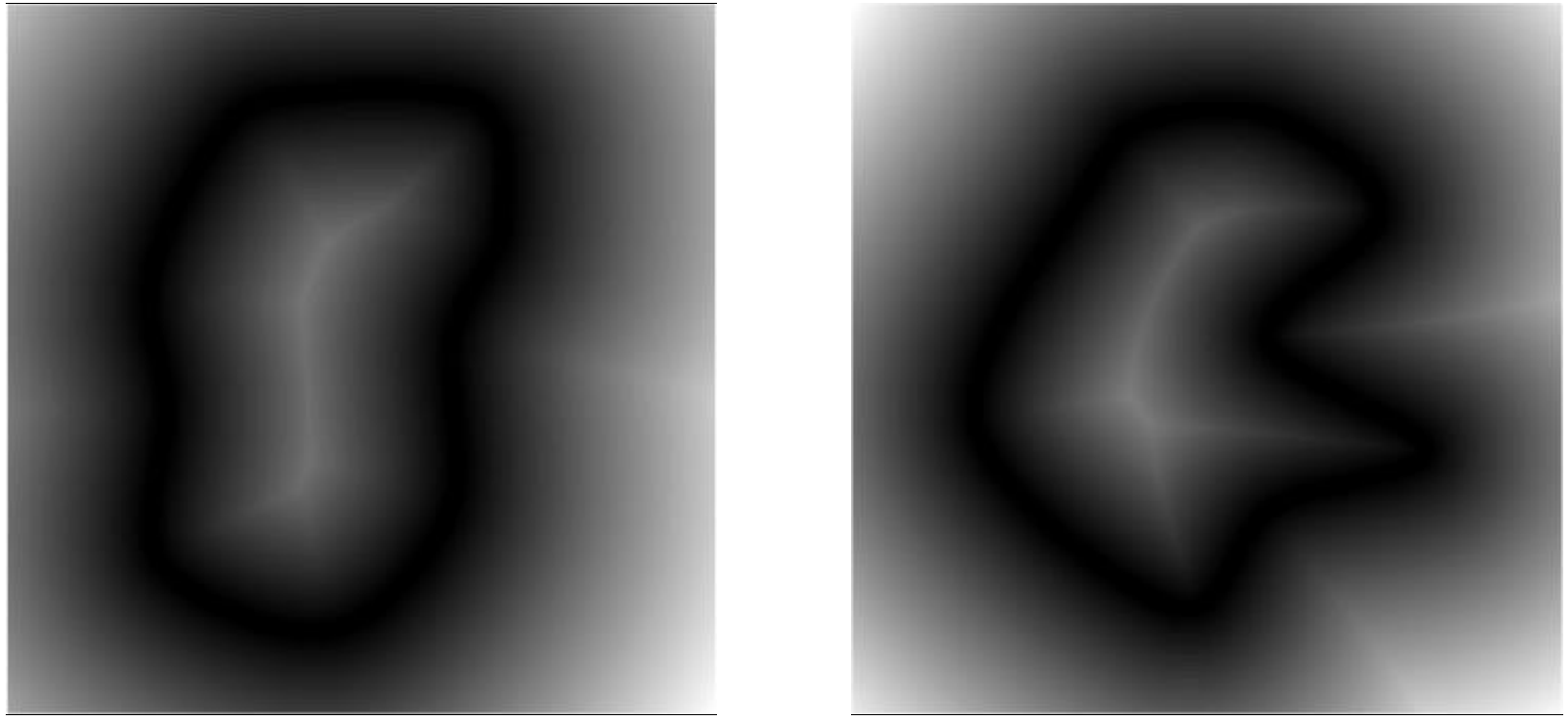
Maier-Hein et al. 2023. Metrics reloaded: Recommendations for image analysis validation

Surface distance



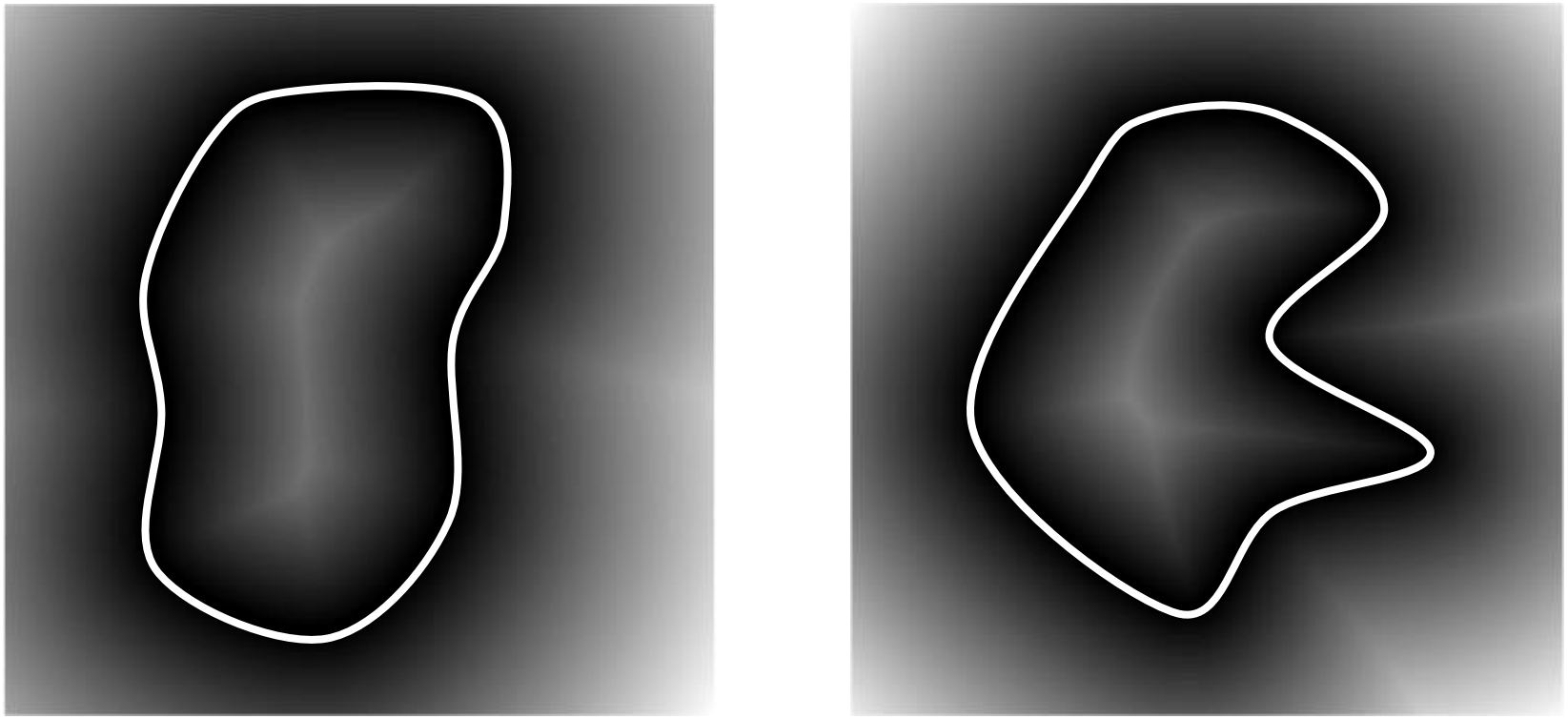


Surface distance



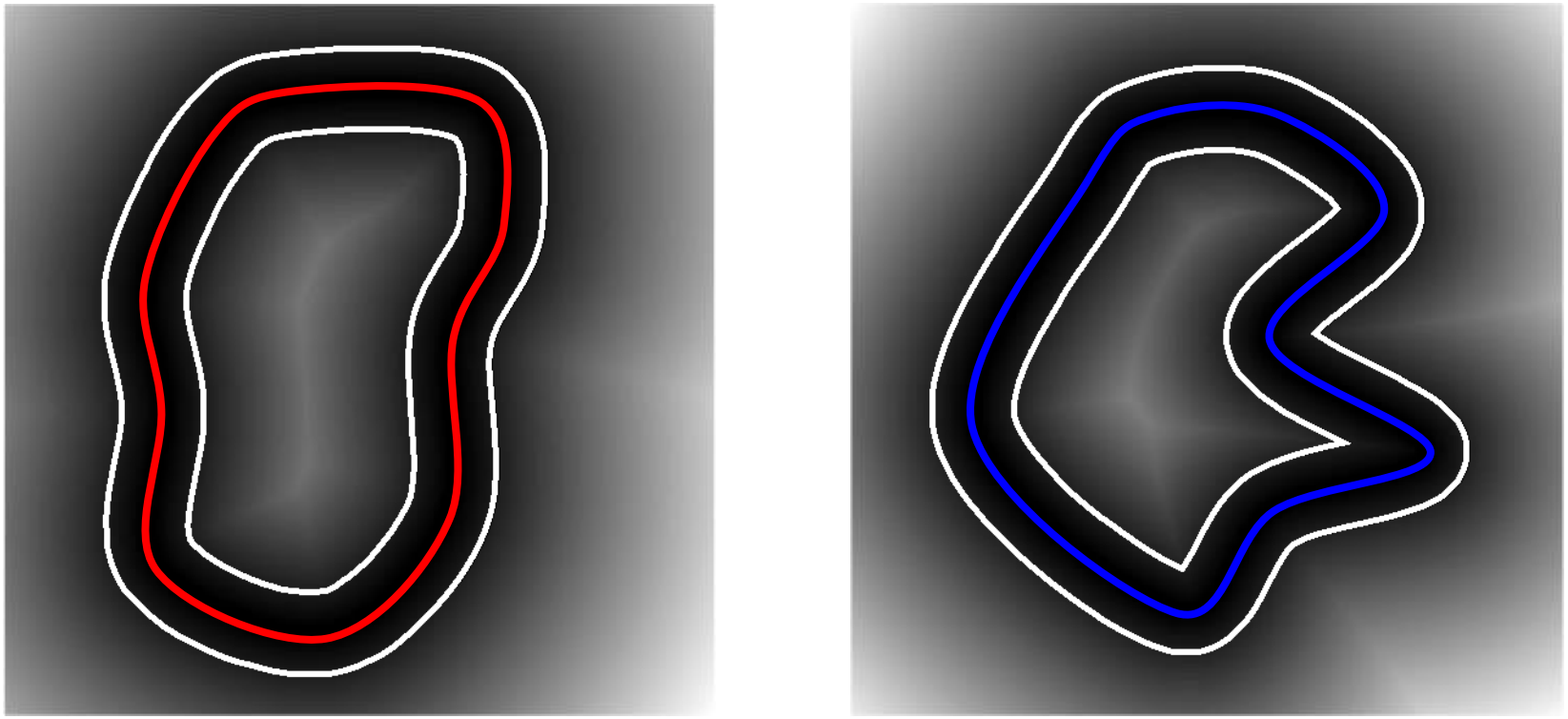
Euclidean Distance Maps

Surface distance



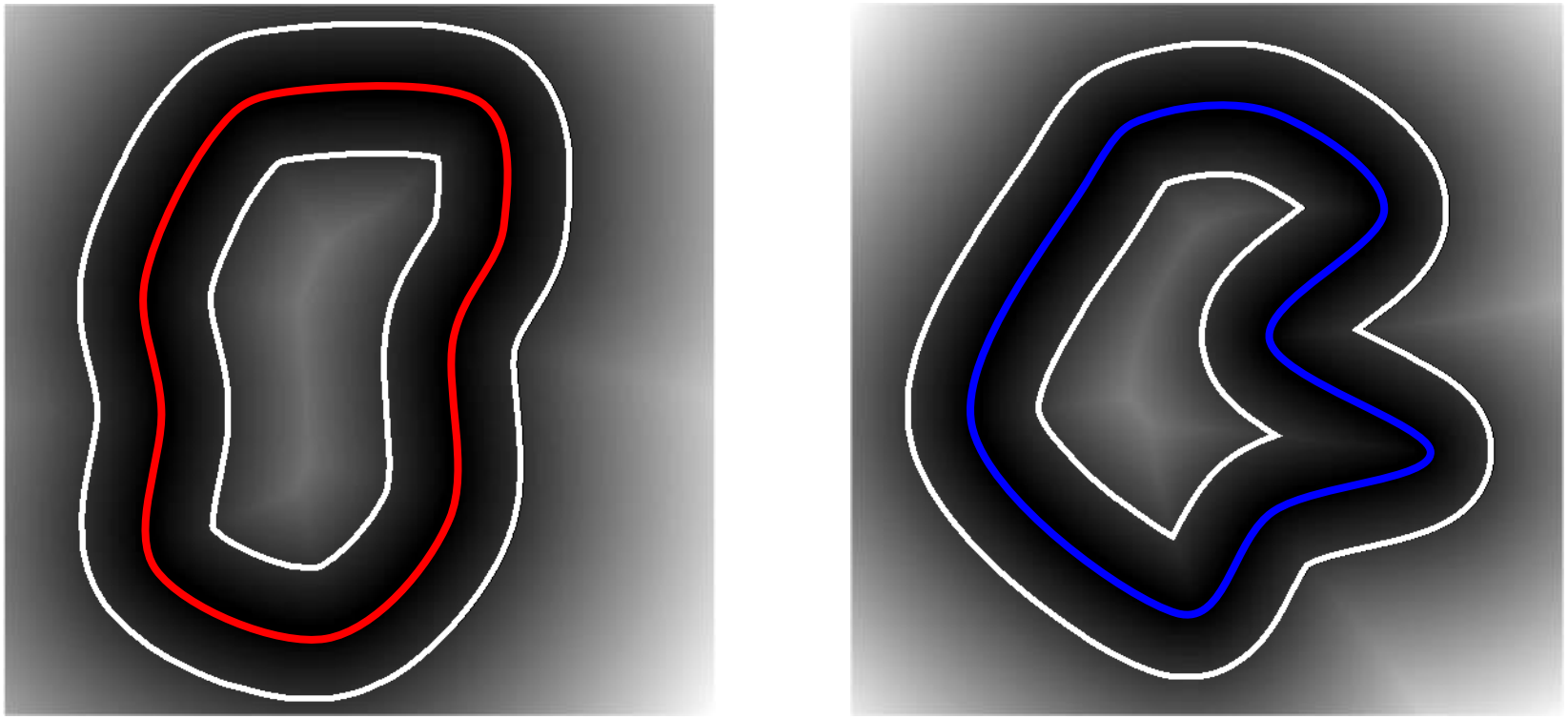
Zero-Level Surface

Surface distance



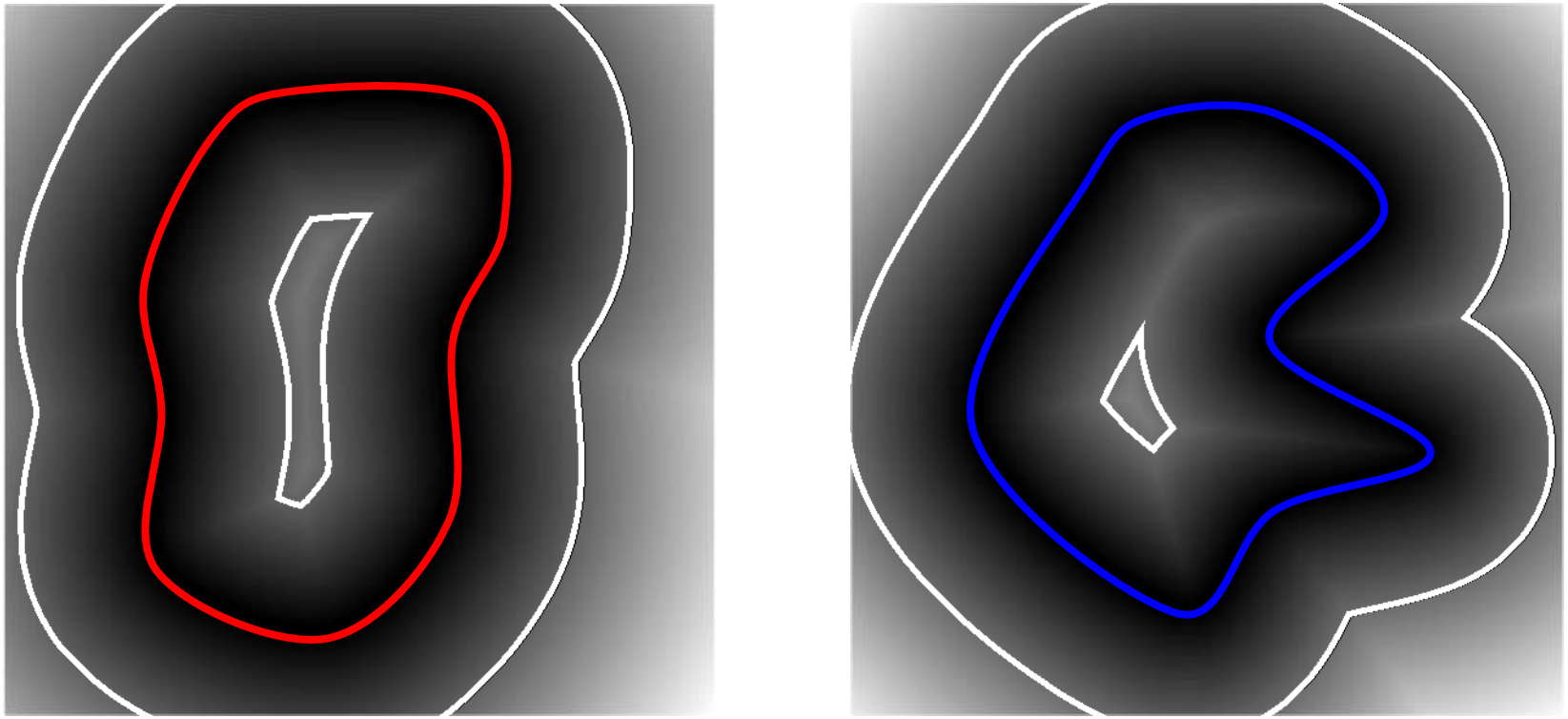
30-Pixels Distance Surface

Surface distance



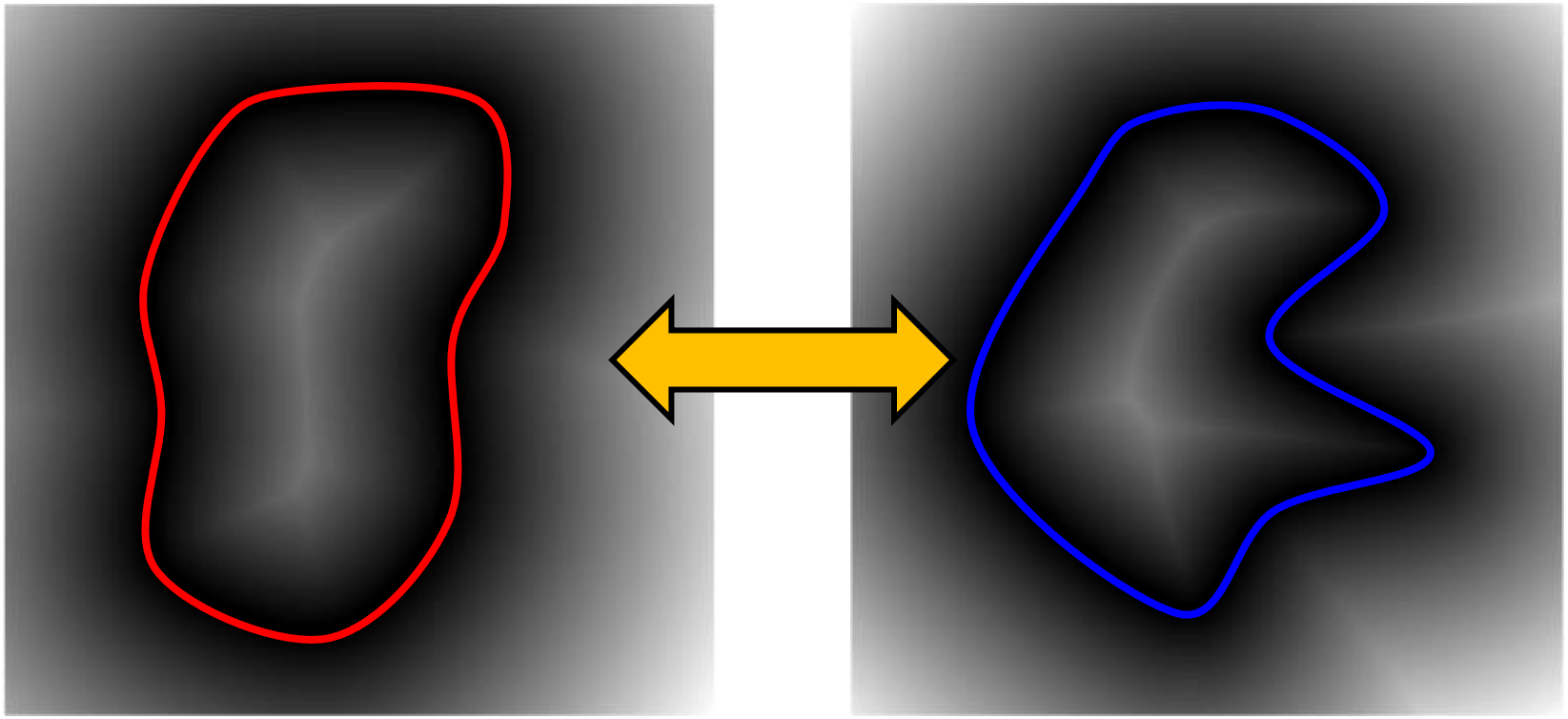
50-Pixels Distance Surface

Surface distance



100-Pixels Distance Surface

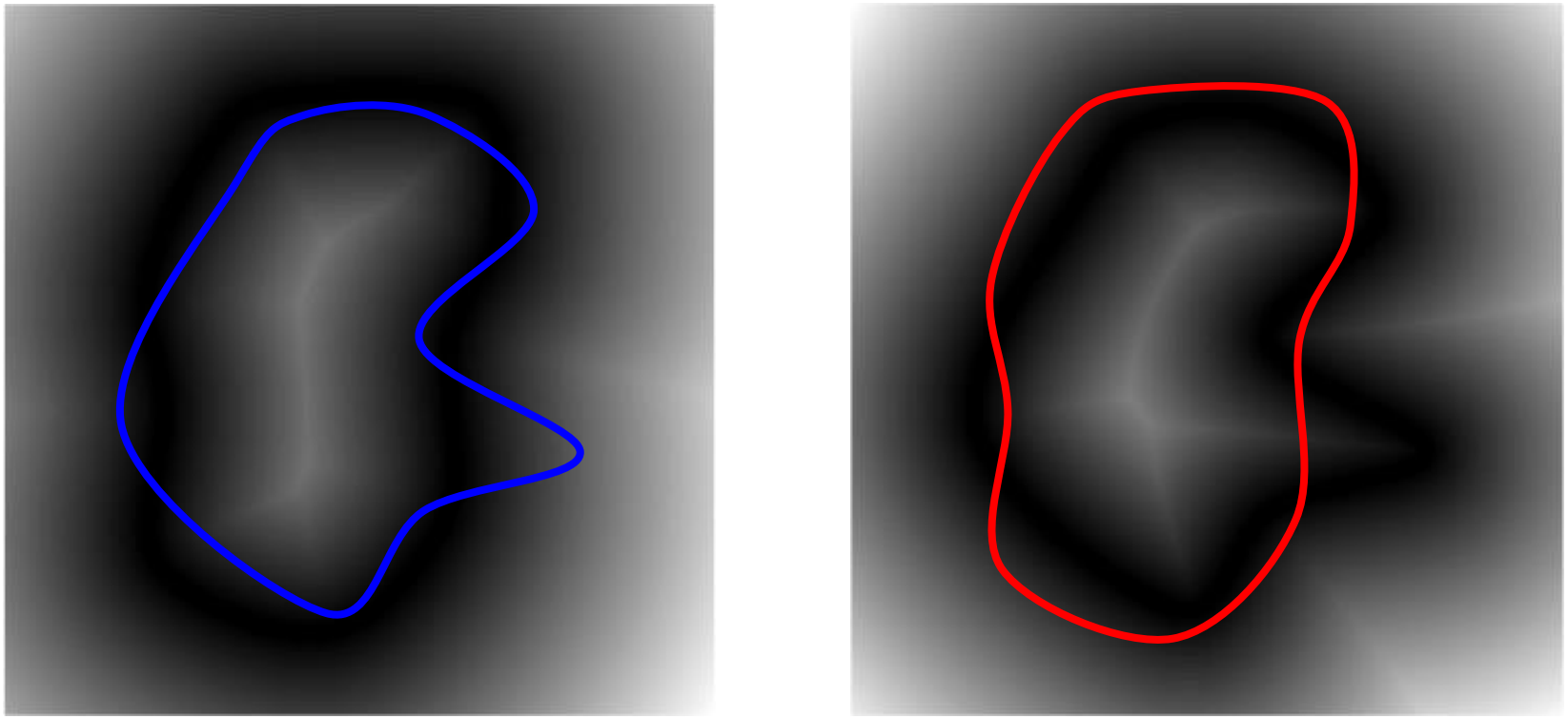
Surface distance



Euclidean Distance Maps

Surface distance

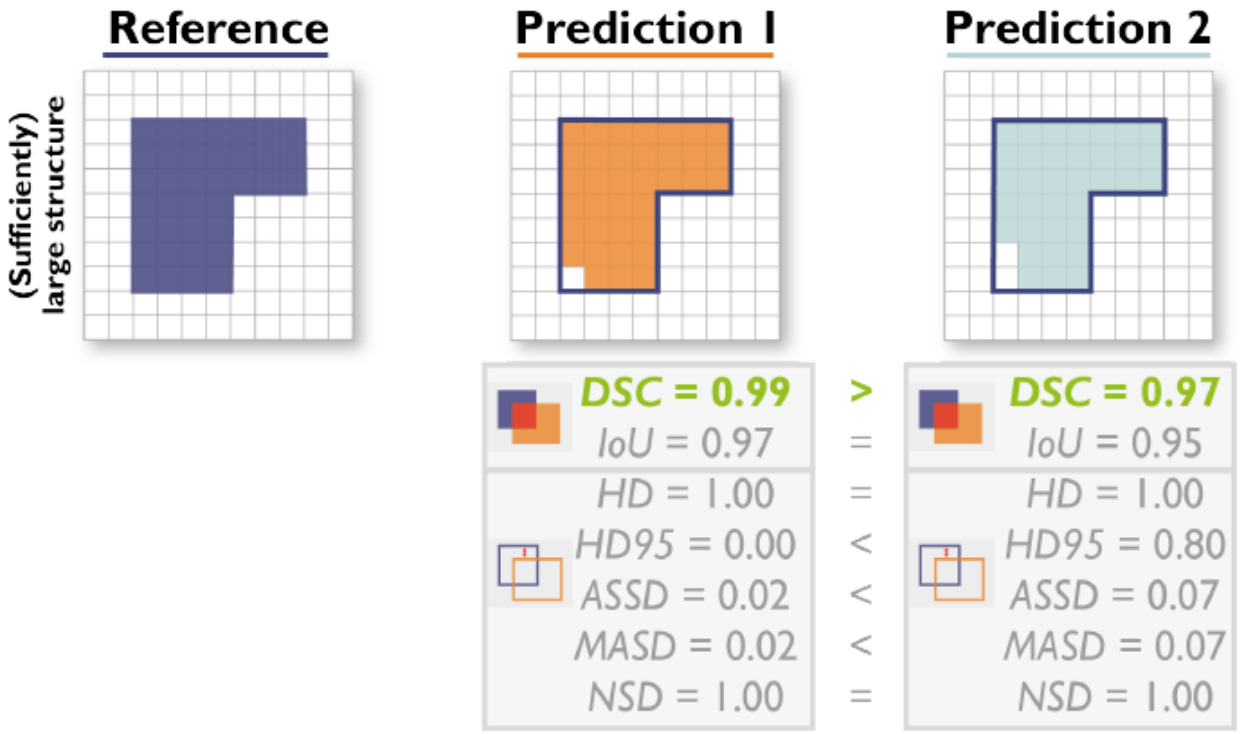
Sum up distances along pixels on the boundaries of one contour overlaid on the distance map of the other



Euclidean Distance Maps

Pitfalls in segmentation evaluation

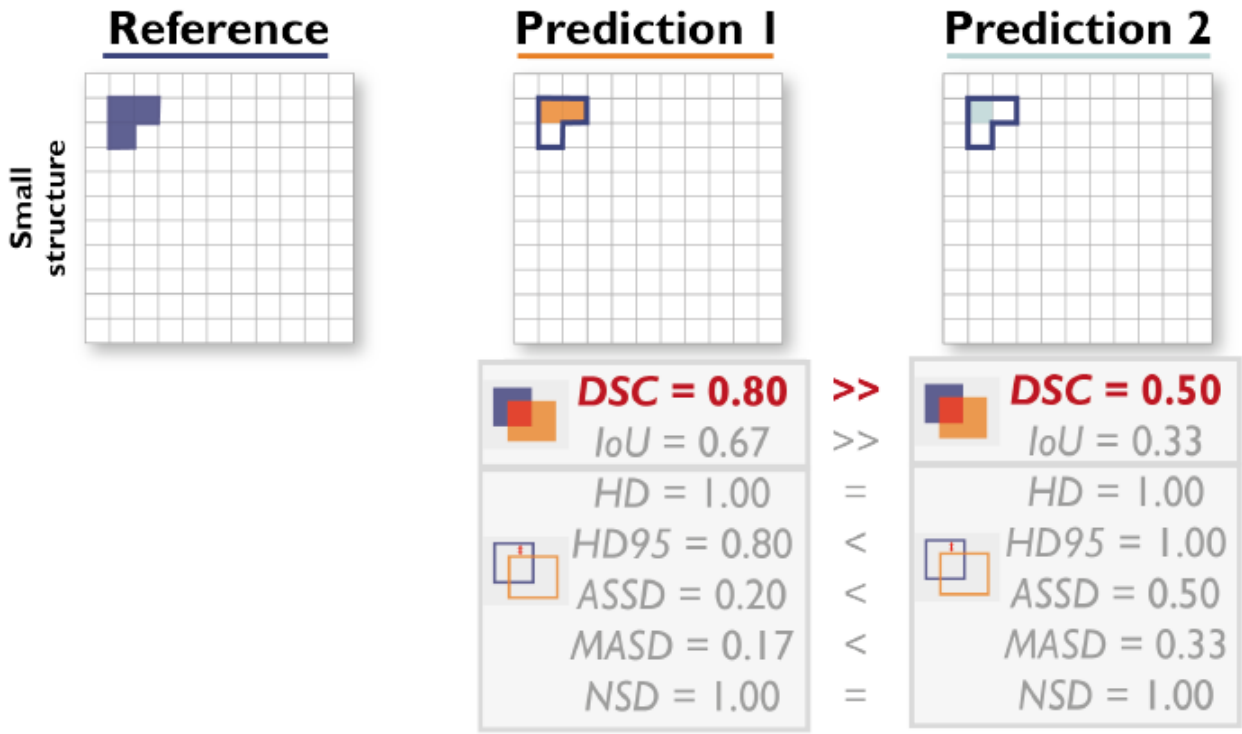
Effect of structure size



Reinke et al. 2023. [Understanding metric-related pitfalls in image analysis validation]

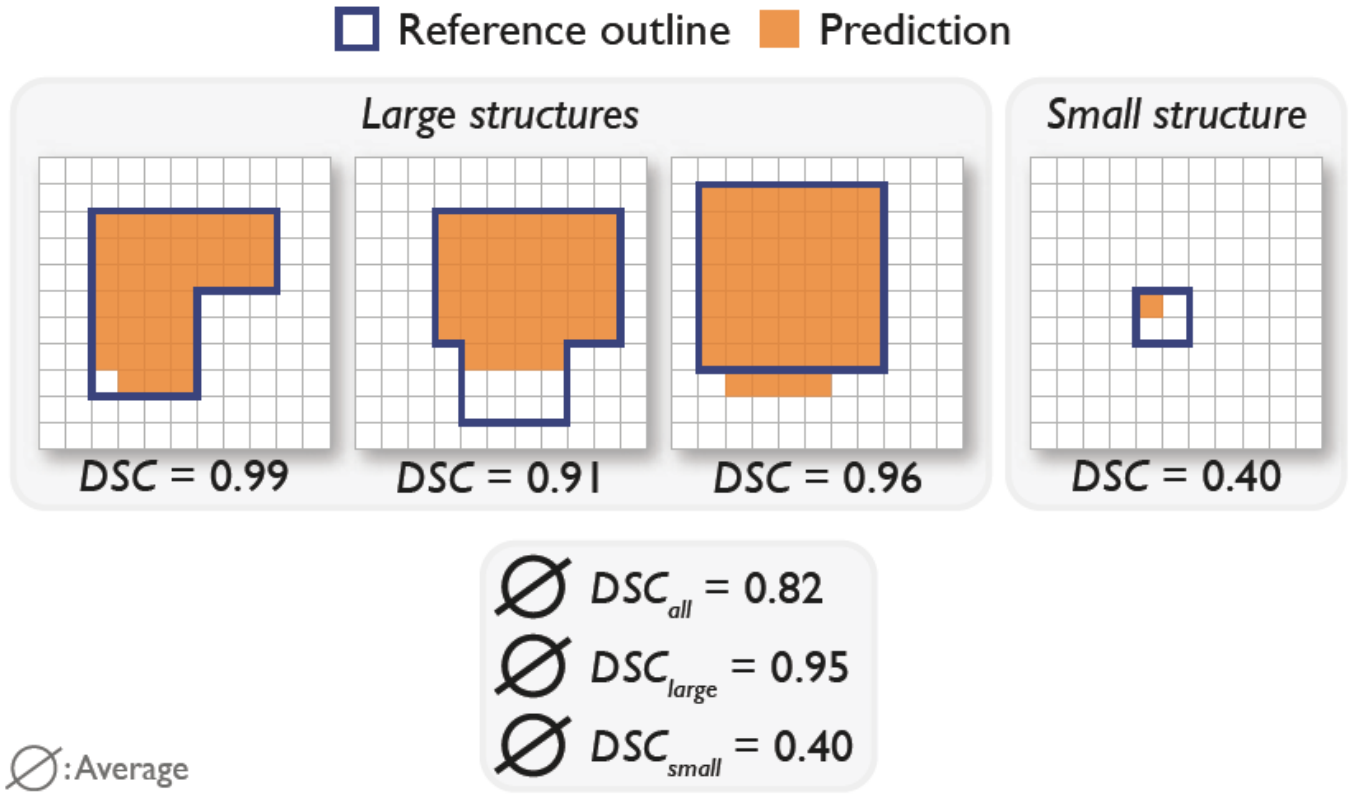
Pitfalls in segmentation evaluation

Effect of structure size



Pitfalls in segmentation evaluation

Effect of structure size



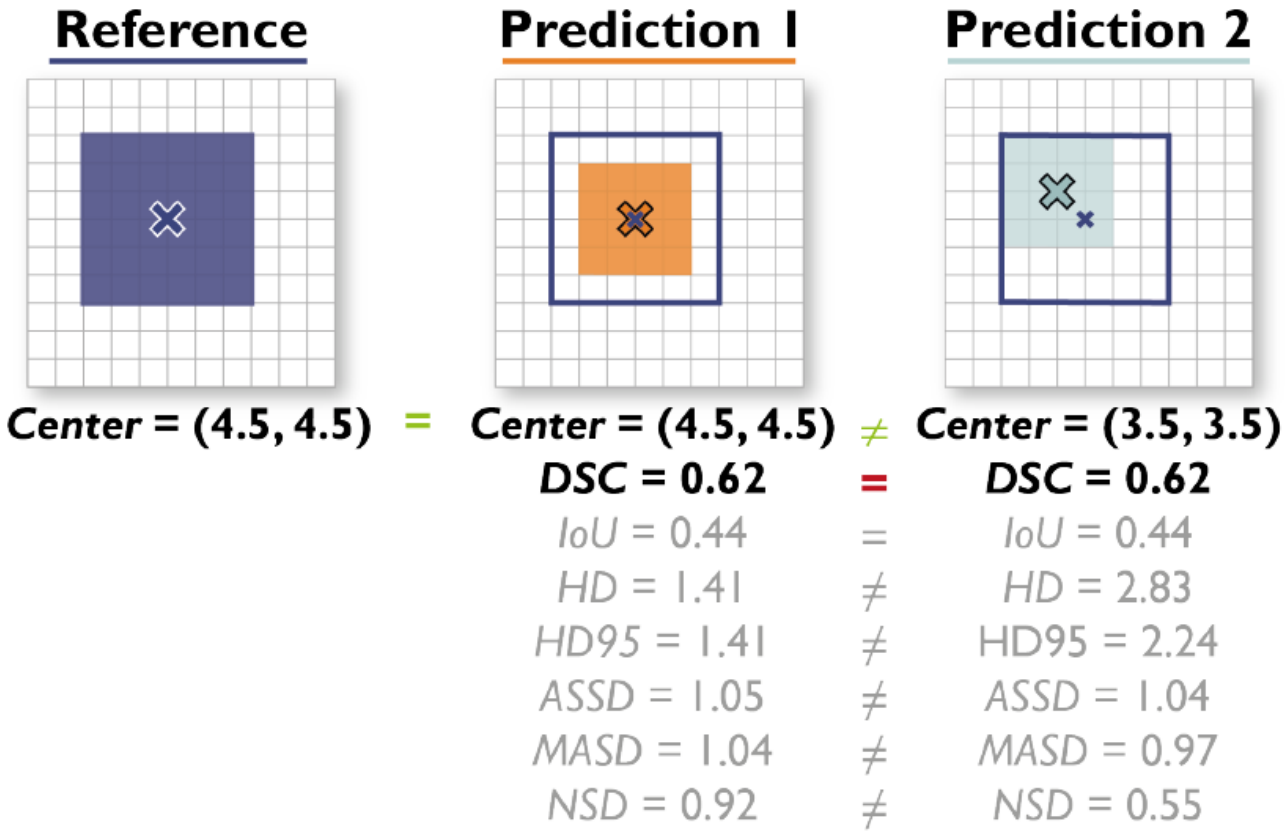
Pitfalls in segmenta

Effect of structure shape

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

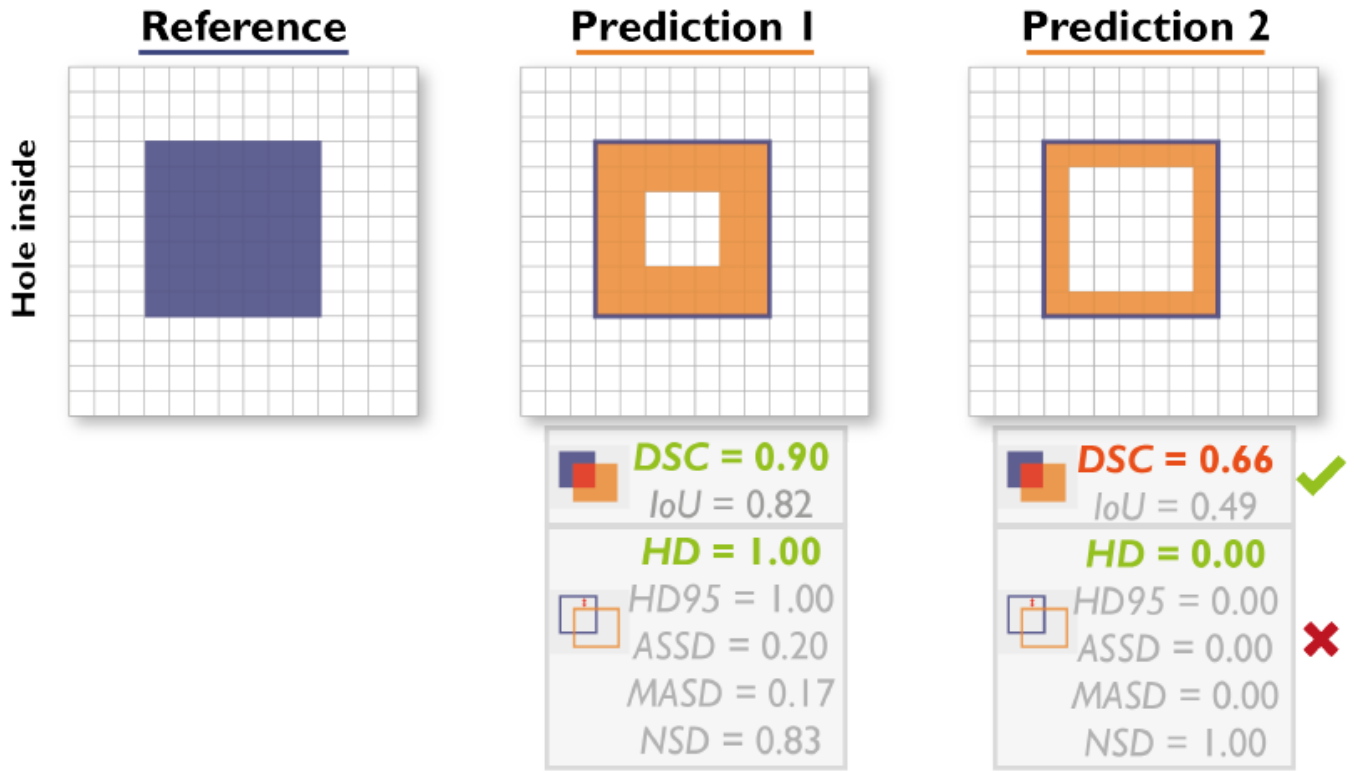
Pitfalls in segmentation evaluation

Effect of spatial alignment



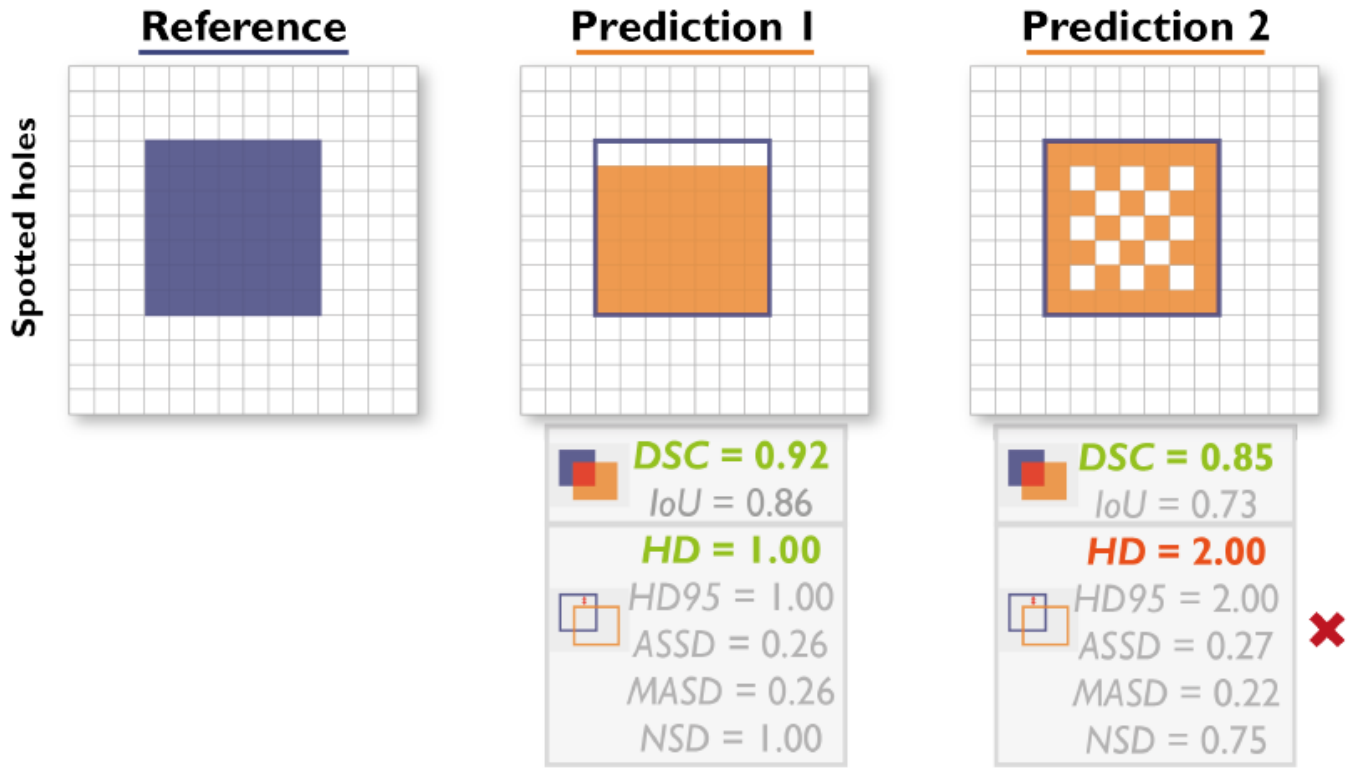
Pitfalls in segmentation evaluation

Effect of “holes”



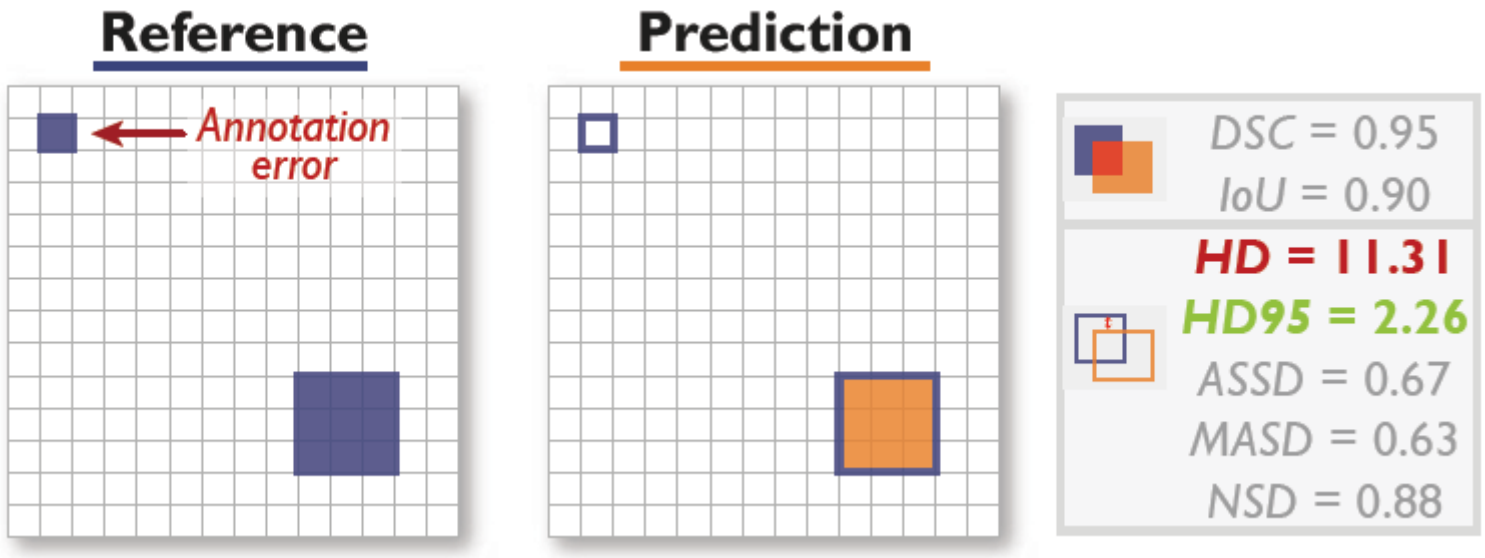
Pitfalls in segmentation evaluation

Effect of “holes”



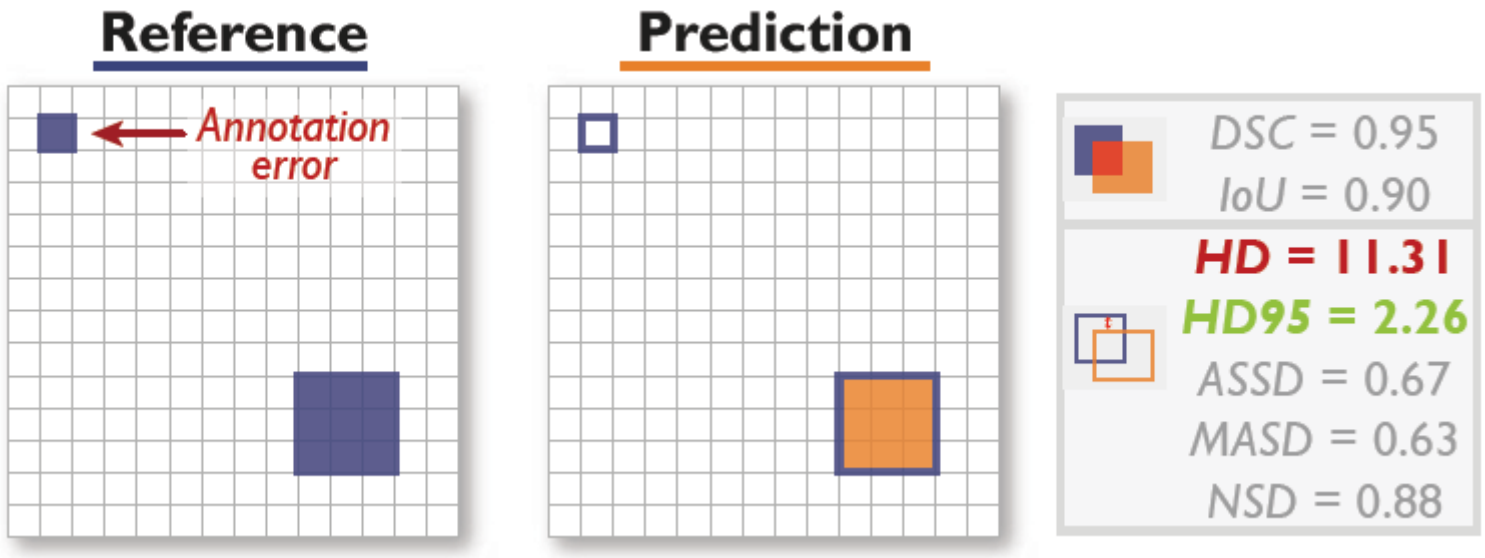
Pitfalls in segmentation evaluation

Effect of annotation noise



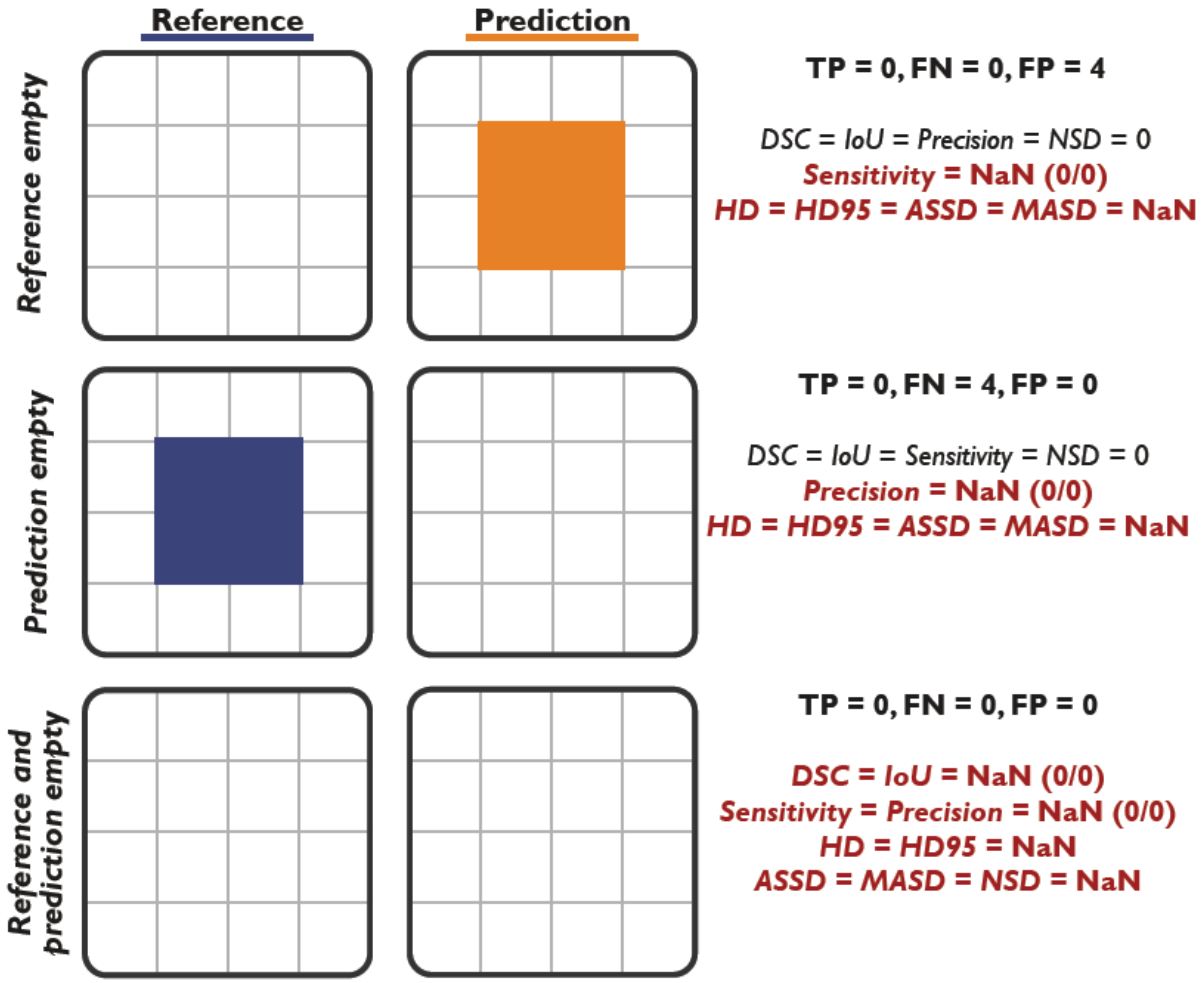
Pitfalls in segmentation evaluation

Effect of annotation noise



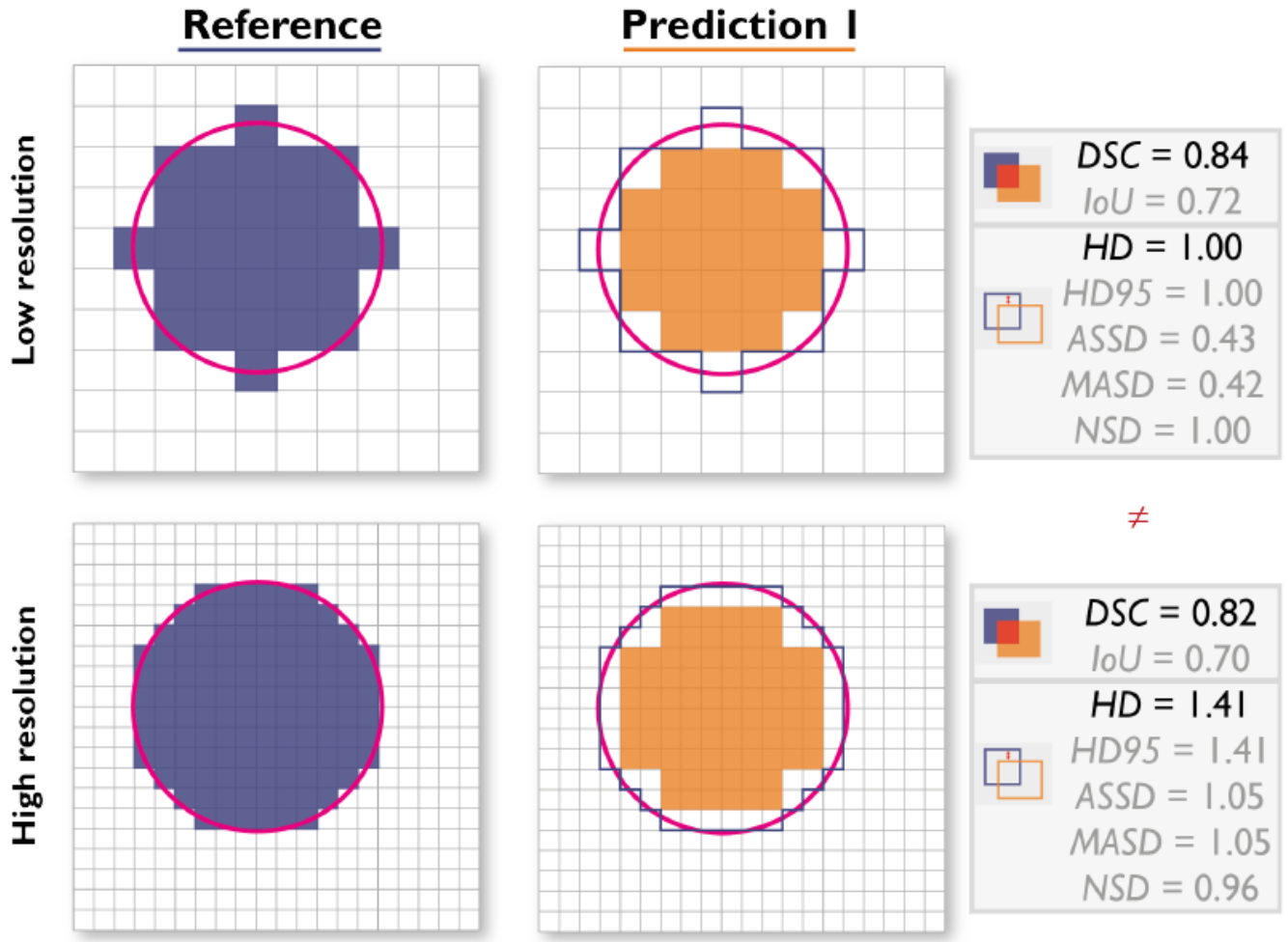
Pitfalls in segmentation evaluation

Effect of “empty” labelmaps



Pitfalls in segmentation evaluation

Effect of resolution

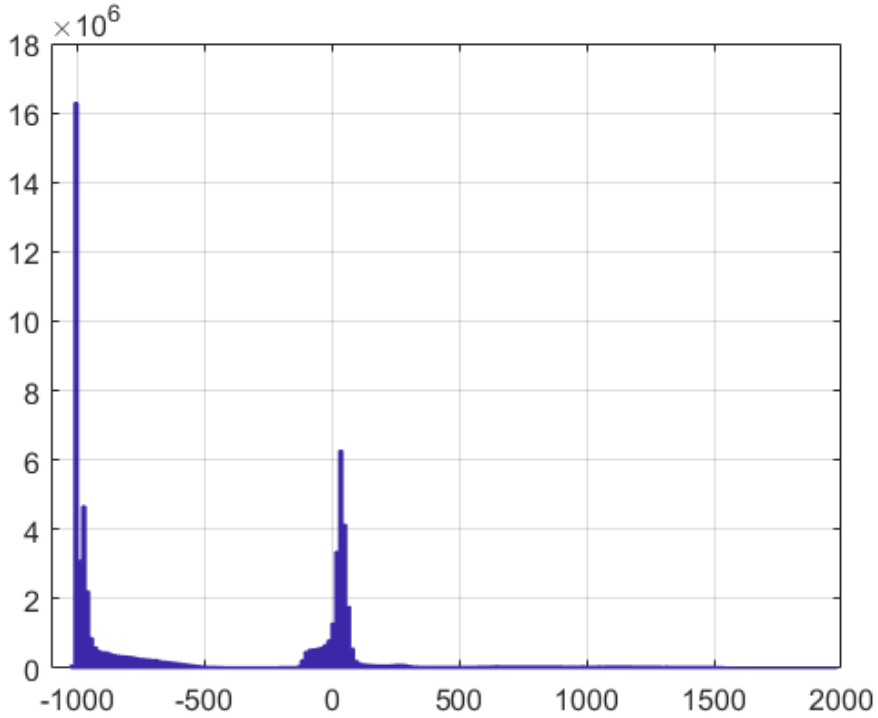


Segmentation methods

- Intensity-based segmentation
  - e.g., thresholding
- Region-based
  - e.g., region growing
- Graph-based segmentation
  - e.g., graph cuts
- Active contours
  - e.g., level sets
- Atlas-based segmentation
  - e.g., multi-atlas label propagation
- Learning-based segmentation
  - e.g., random forests, **convolutional neural networks**

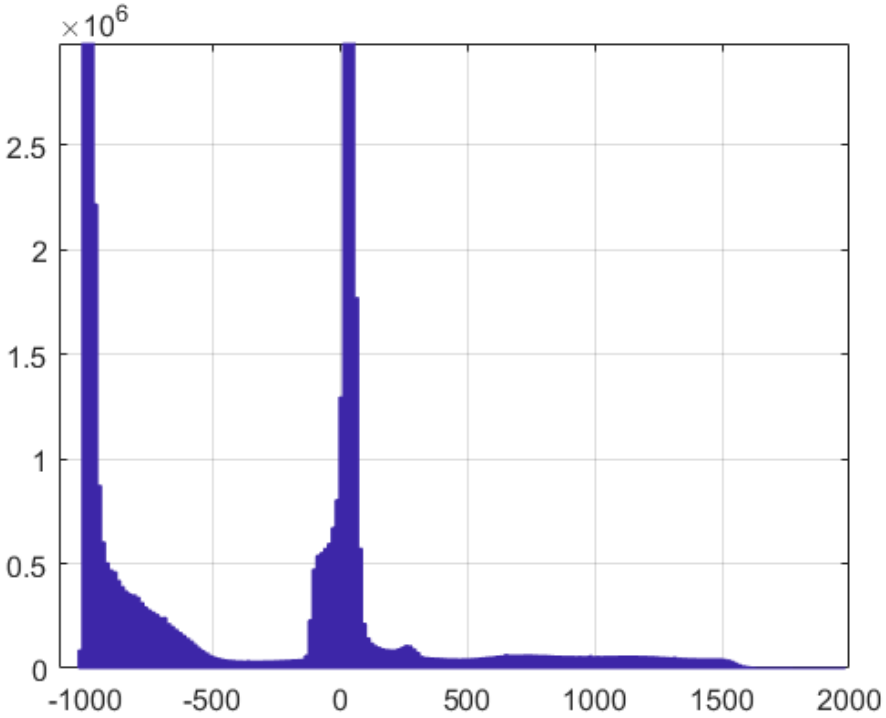
Simple thresholding

- Select a threshold on the intensity range



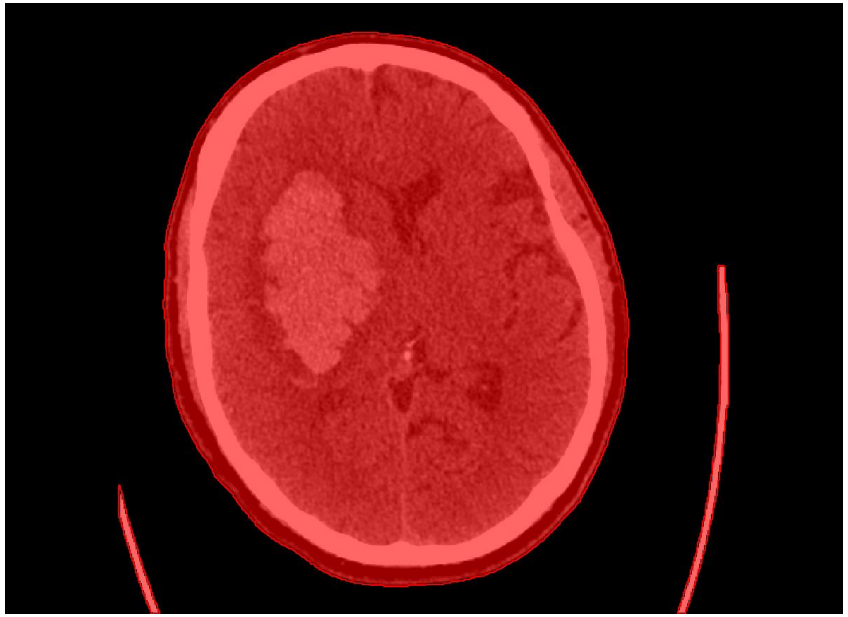
Simple thresholding

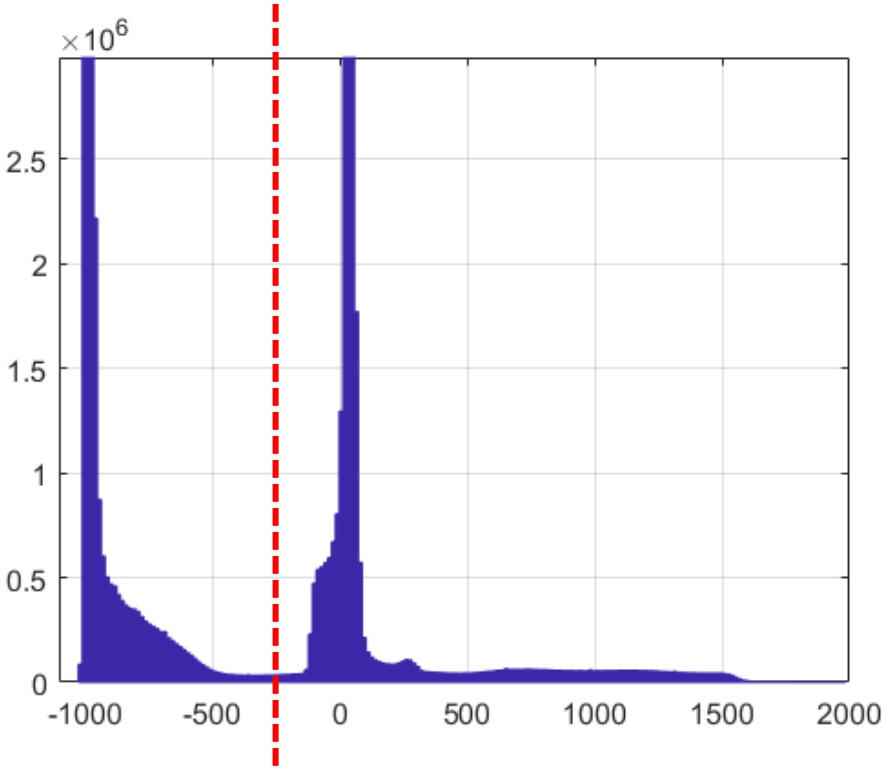
- Select a threshold on the intensity range



Simple thresholding

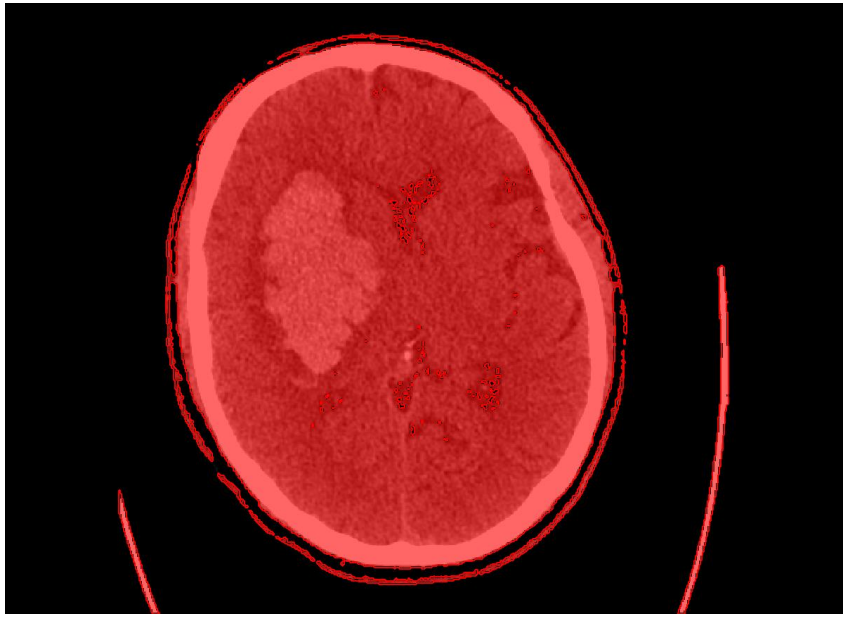
- Select a threshold on the intensity range

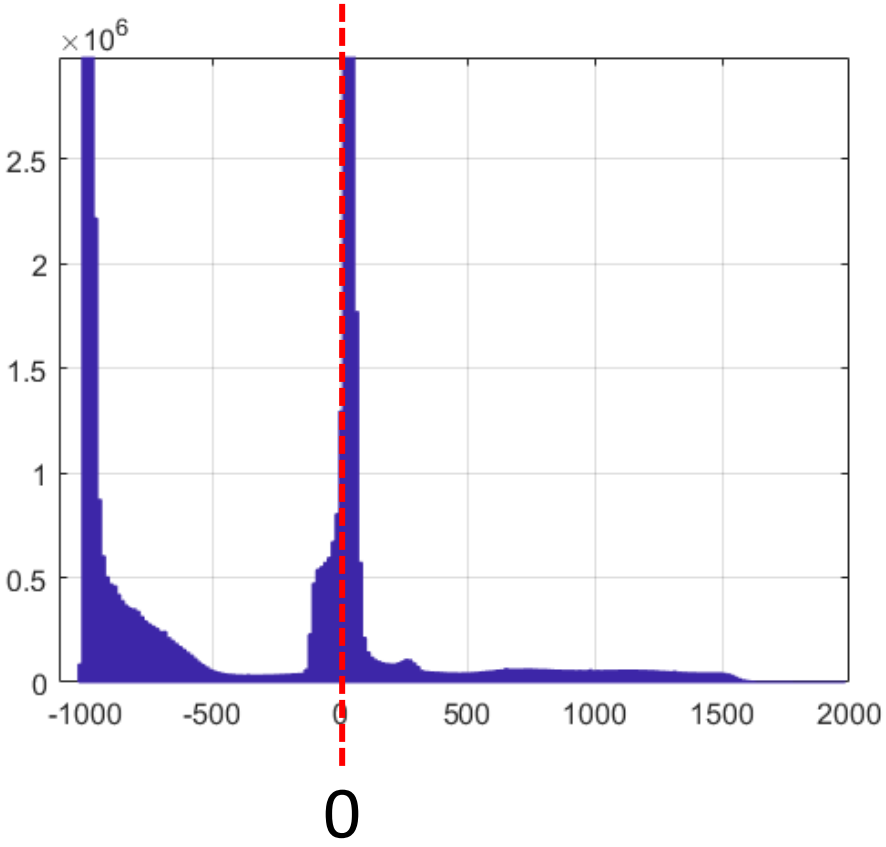




Simple thresholding

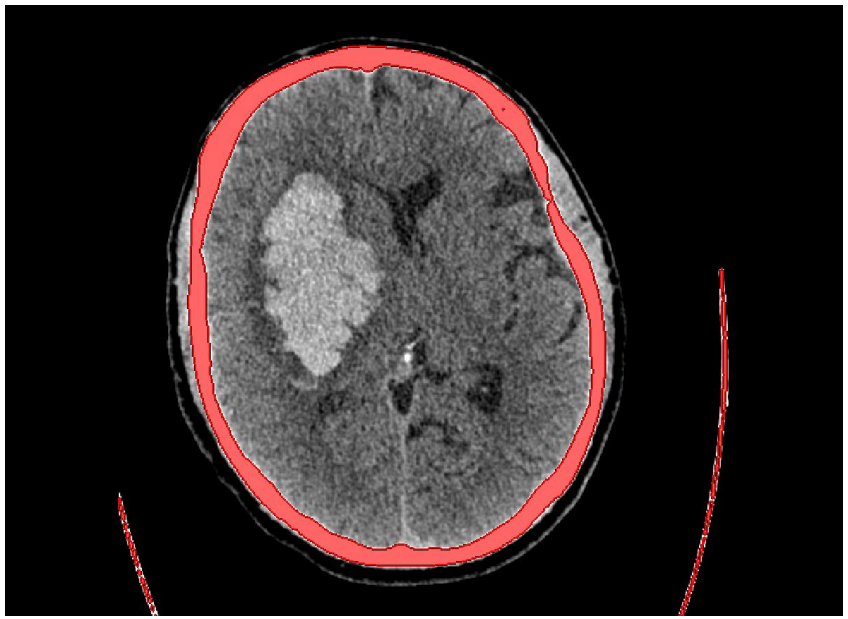
- Select a threshold on the intensity range

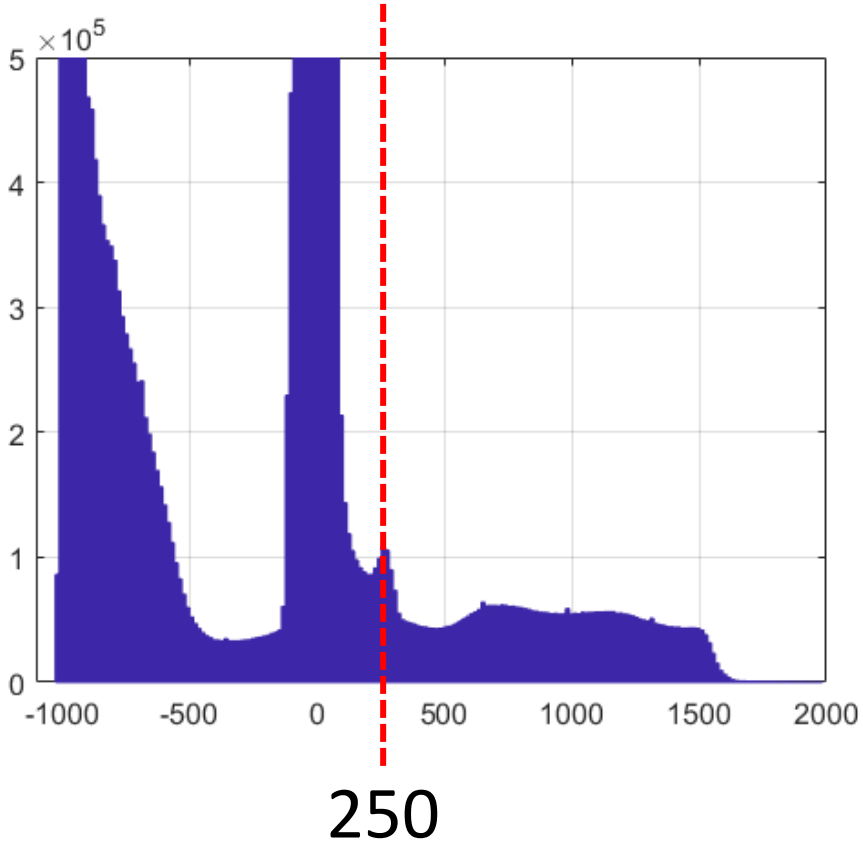




Simple thresholding

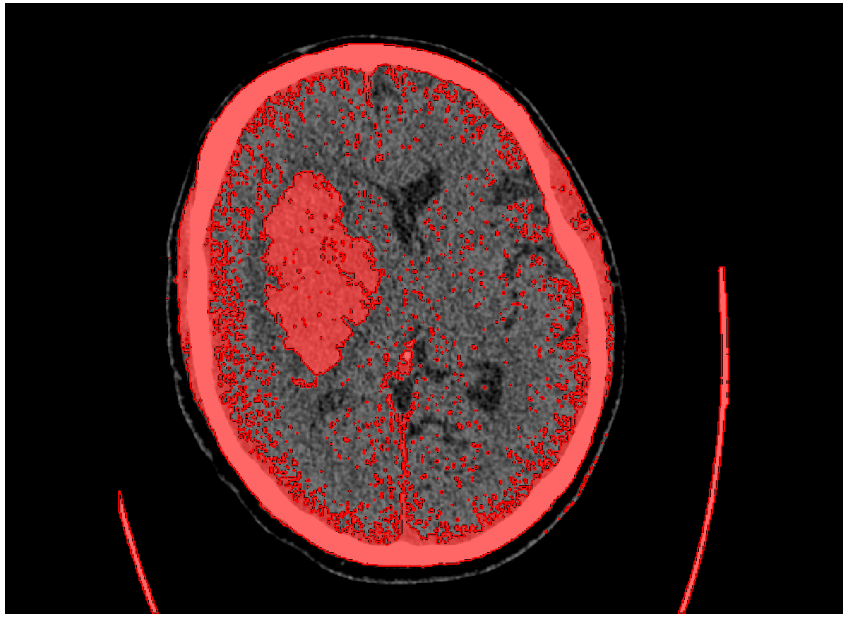
- Select a threshold on the intensity range

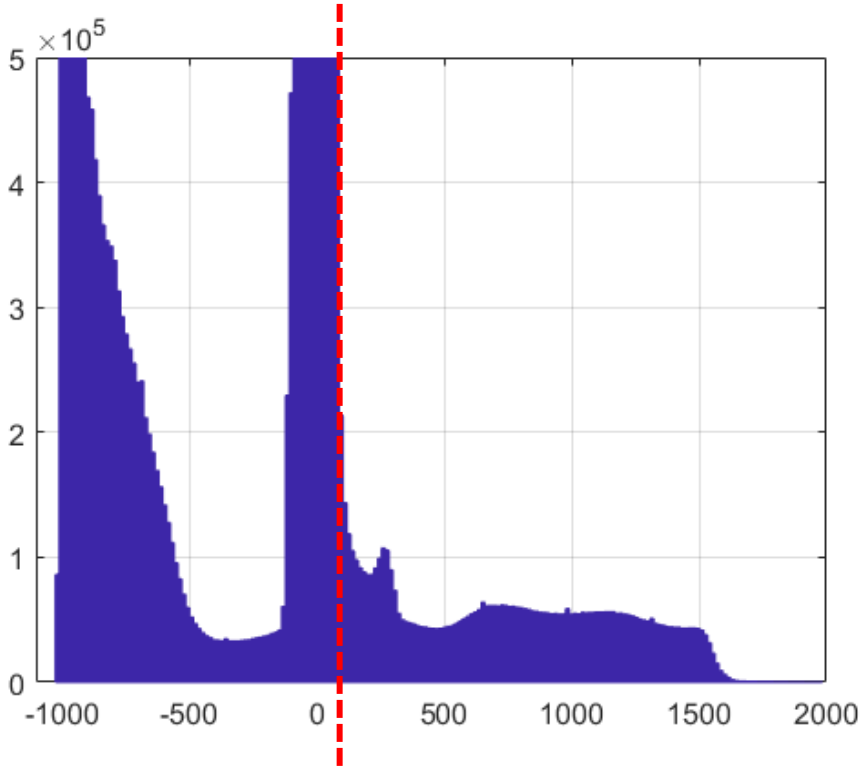




Simple thresholding

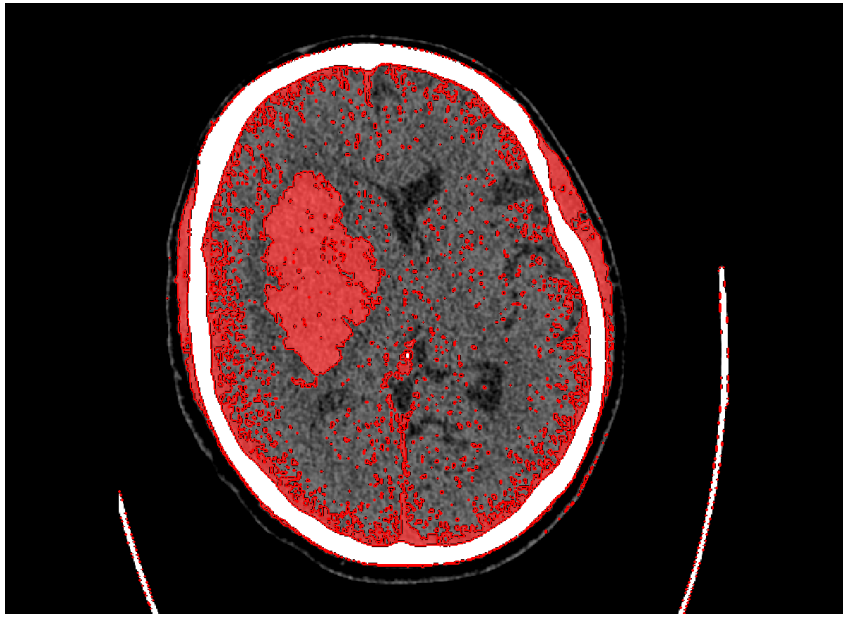
- Select a threshold on the intensity range

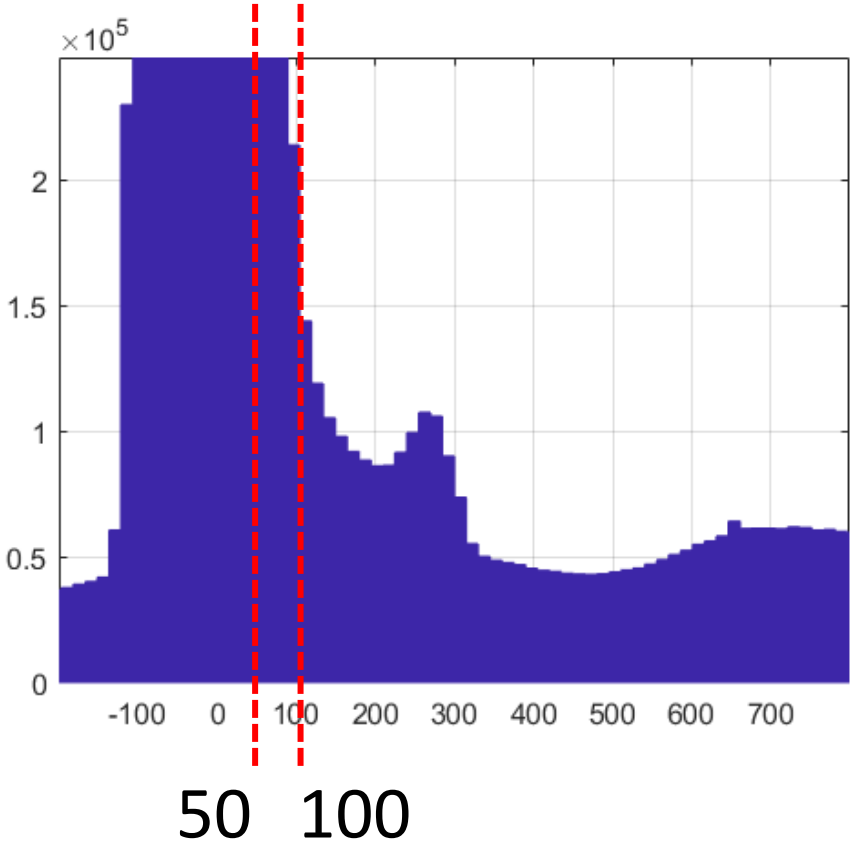




UL thresholding

- Select a lower and upper threshold





Thresholding

Advantages

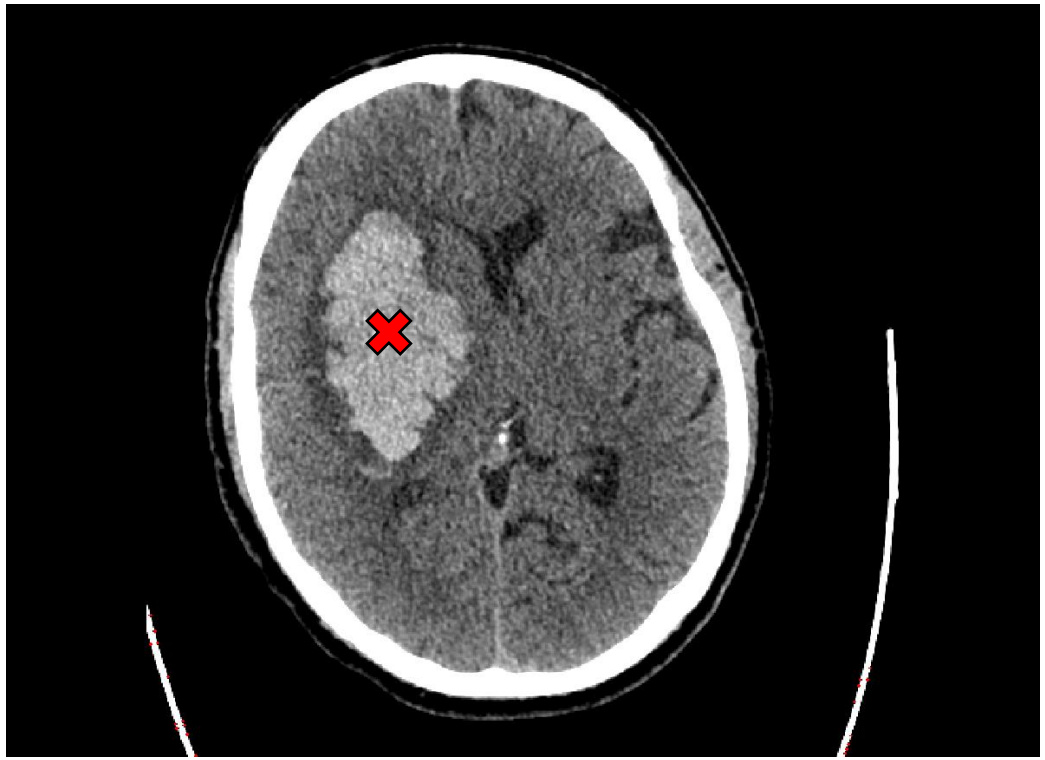
- simple
- fast

Disadvantages

- regions must be homogeneous and distinct
- difficulty in finding consistent thresholds across images
- leakages, isolated pixels and ‘rough’ boundaries likely

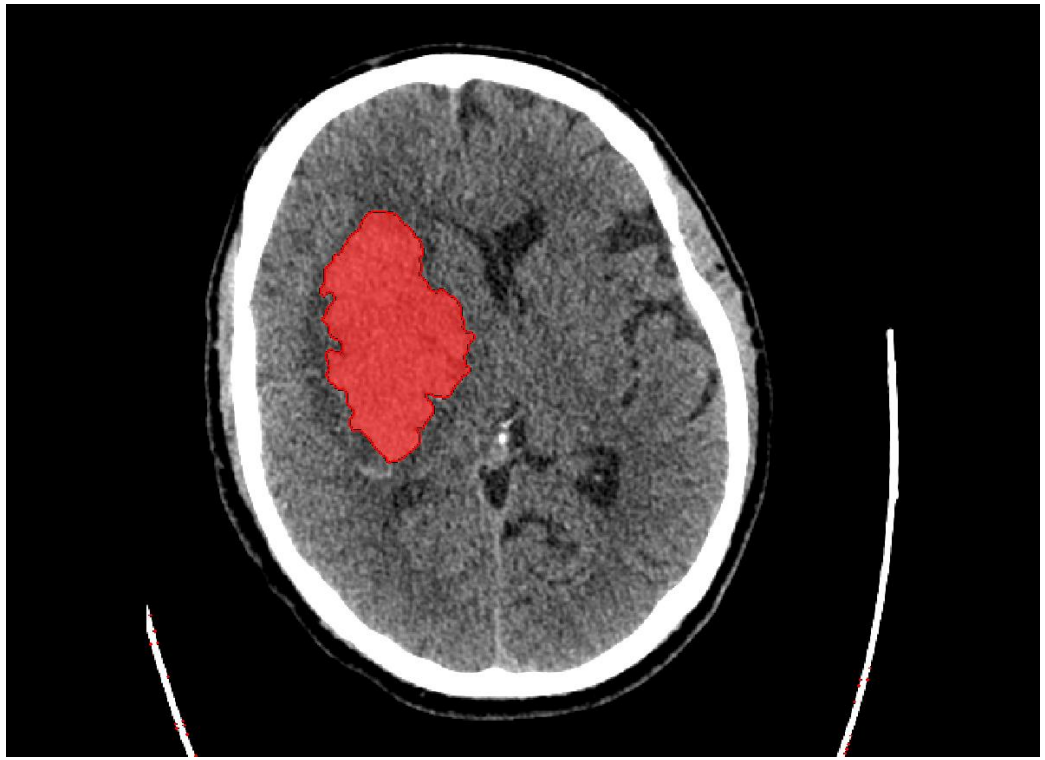
Region growing

- Start from (user selected) seed point(s), and grow a region according to an intensity threshold



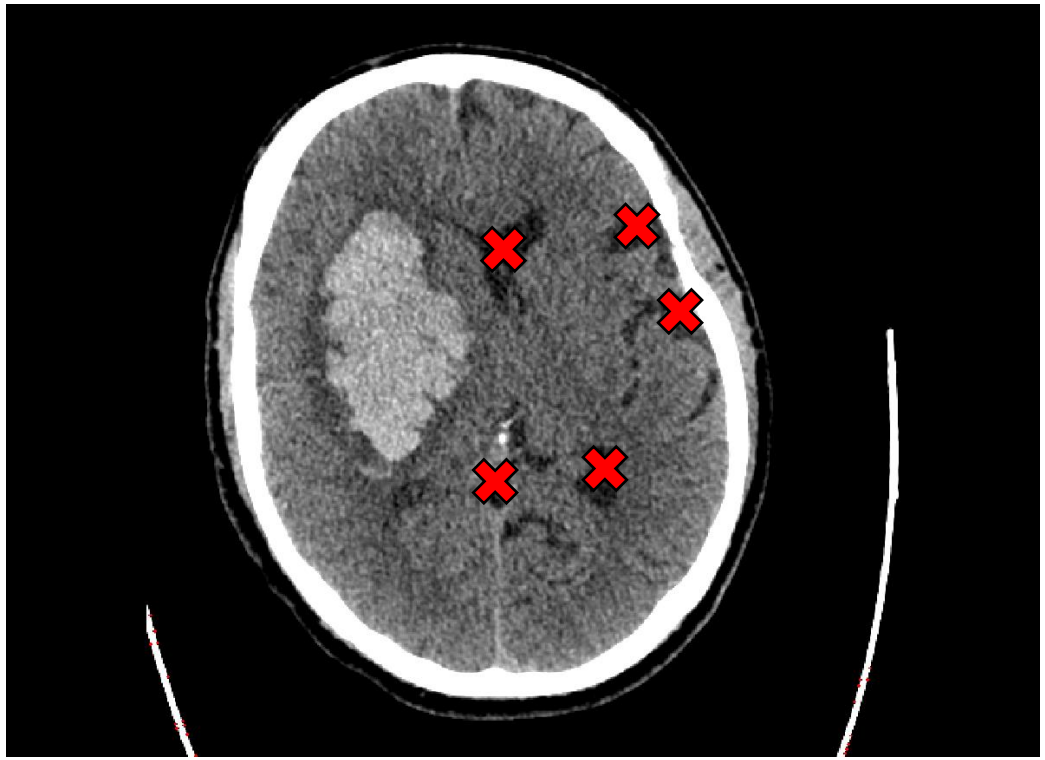
Region growing

- Start from (user selected) seed point(s), and grow a region according to an intensity threshold



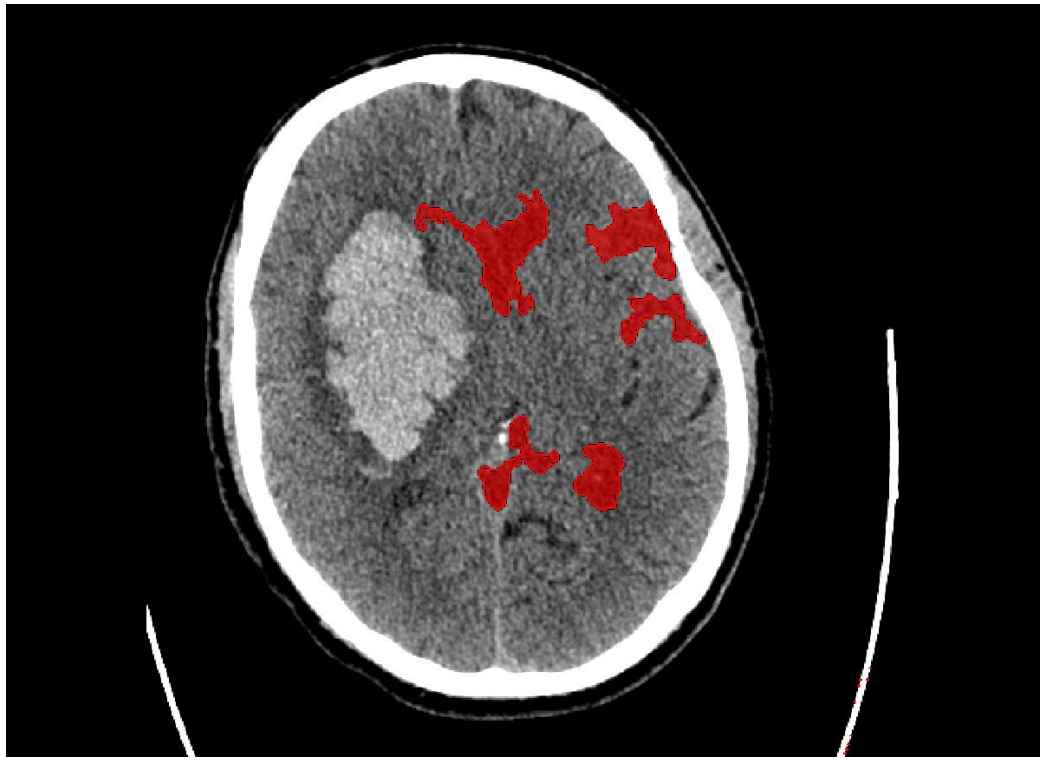
Region growing

- Start from (user selected) seed point(s), and grow a region according to an intensity threshold



Region growing

- Start from (user selected) seed point(s), and grow a region according to an intensity threshold

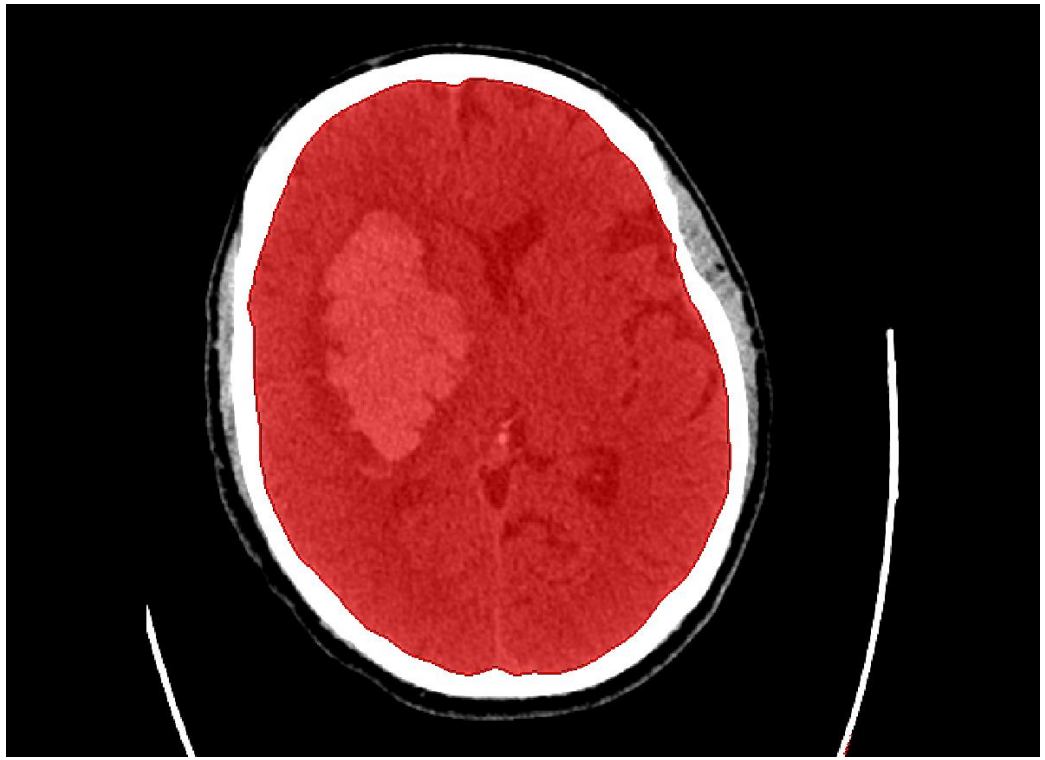


Region growing

- Start from (user selected) seed point(s), and grow a region according to an intensity threshold

Region growing

- Start from (user selected) seed point(s), and grow a region according to an intensity threshold



Region growing

Advantages

- relatively fast
- yields connected region (from a seed point)

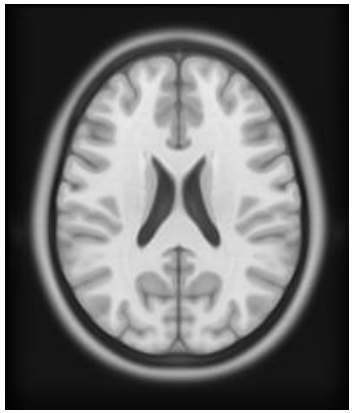
Disadvantages

- regions must be homogeneous
- leakages and ‘rough’ boundaries likely
- requires (user) input for seed points

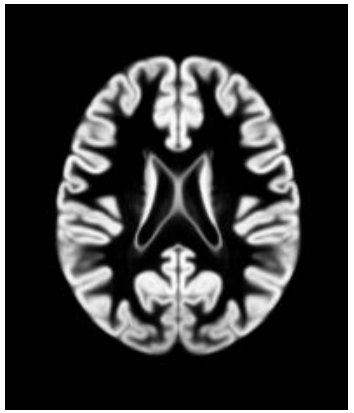
Atlas-based segmentation

What is an atlas?

MNI ICBM 152 Nonlinear atlases (2009)

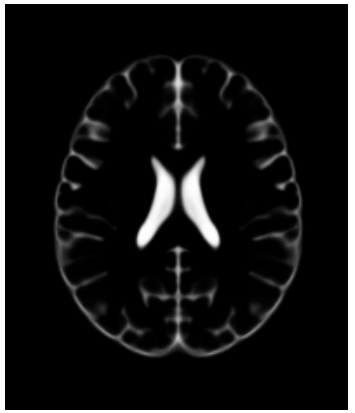


T1 MRI



grey matter

white matter

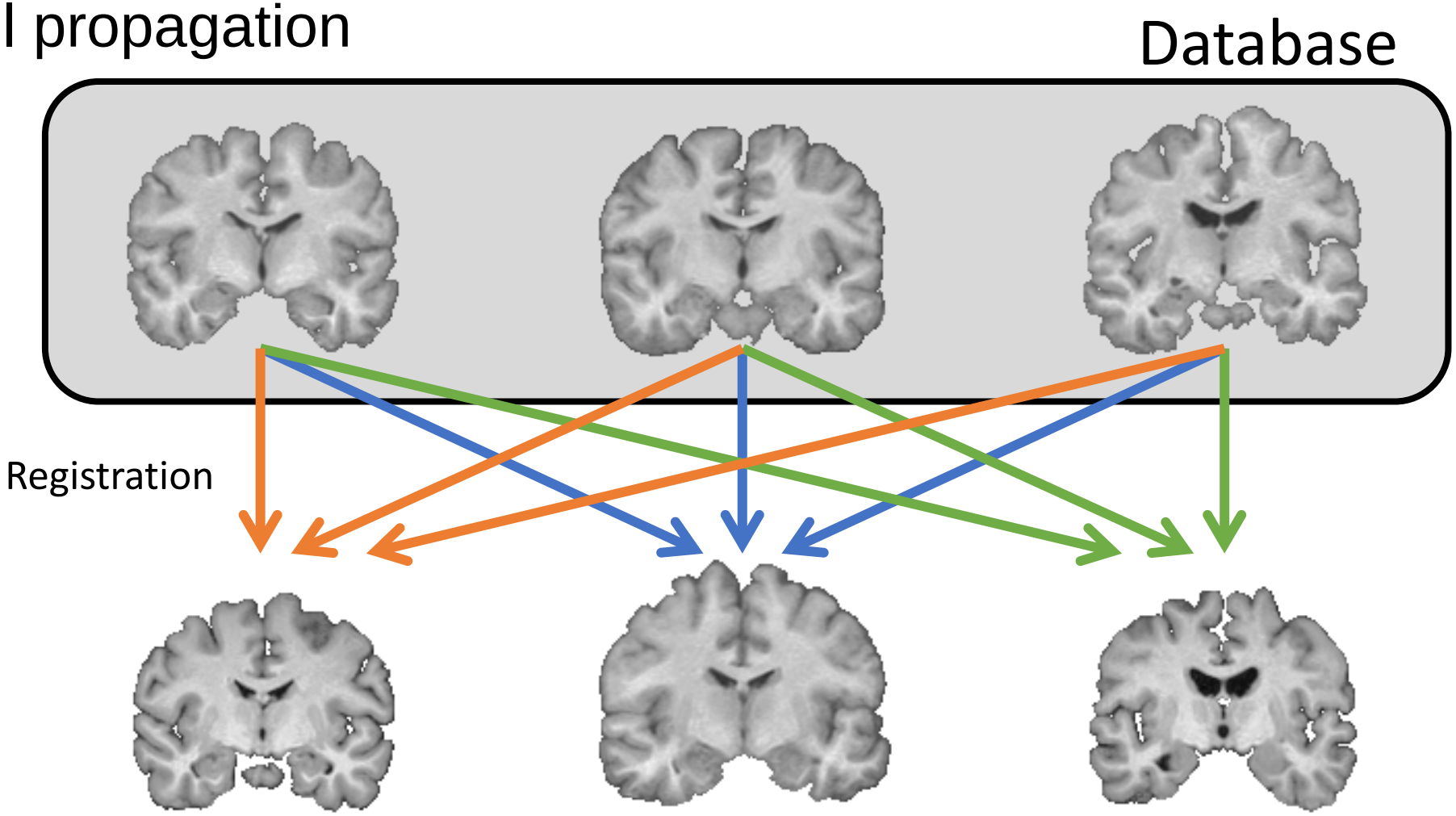


CSF

- A map or chart of the anatomy
- Atlases usually have
  - geometric information about points, curves or surfaces, or
  - label information about voxels (anatomical regions or function)
- Atlases are usually constructed from example data
  - single subjects
  - populations of subjects, e.g. by averaging to produce probabilistic atlases

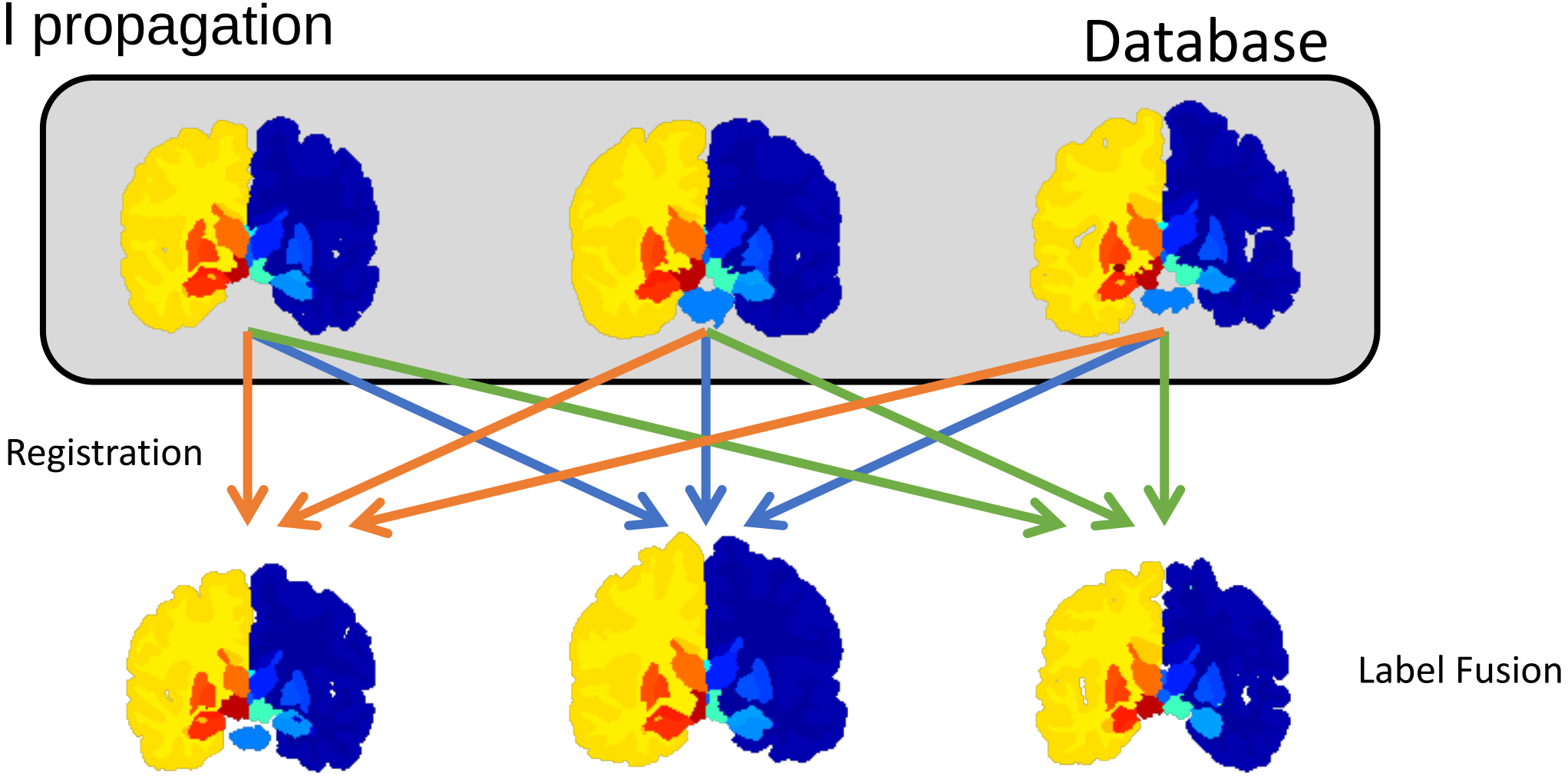
Segmentation via registration

Label propagation

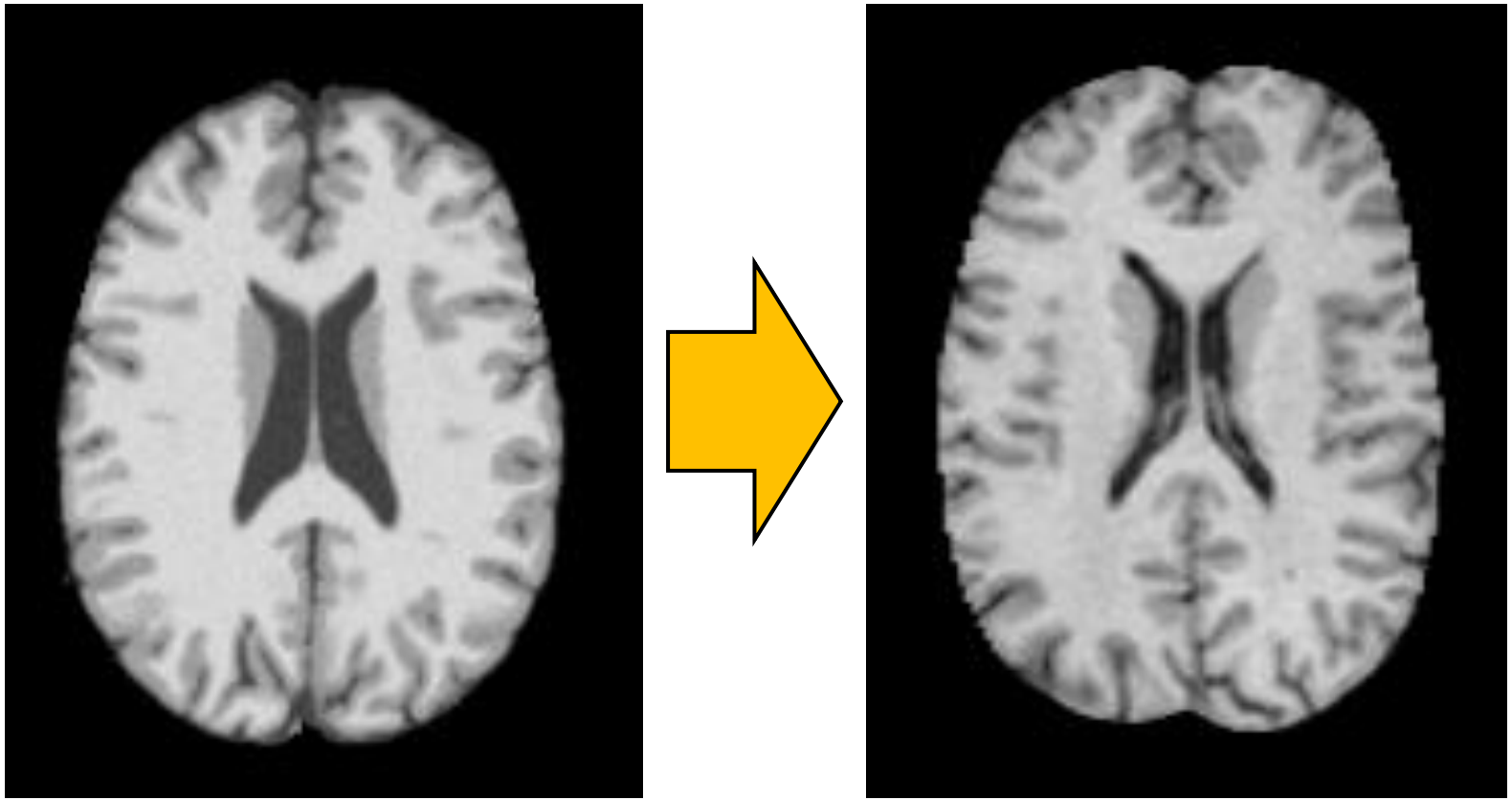


Segmentation via registration

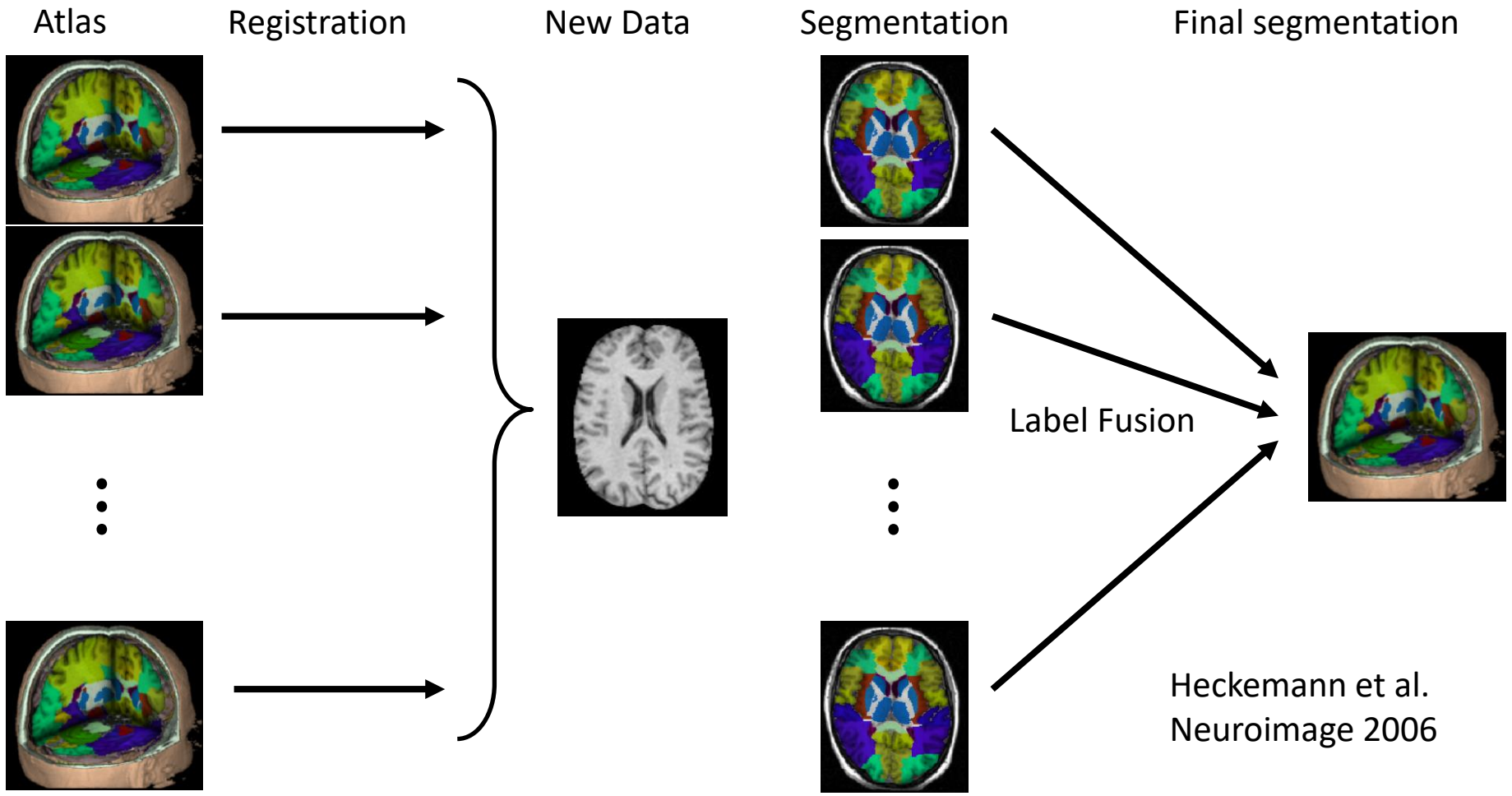
Label propagation



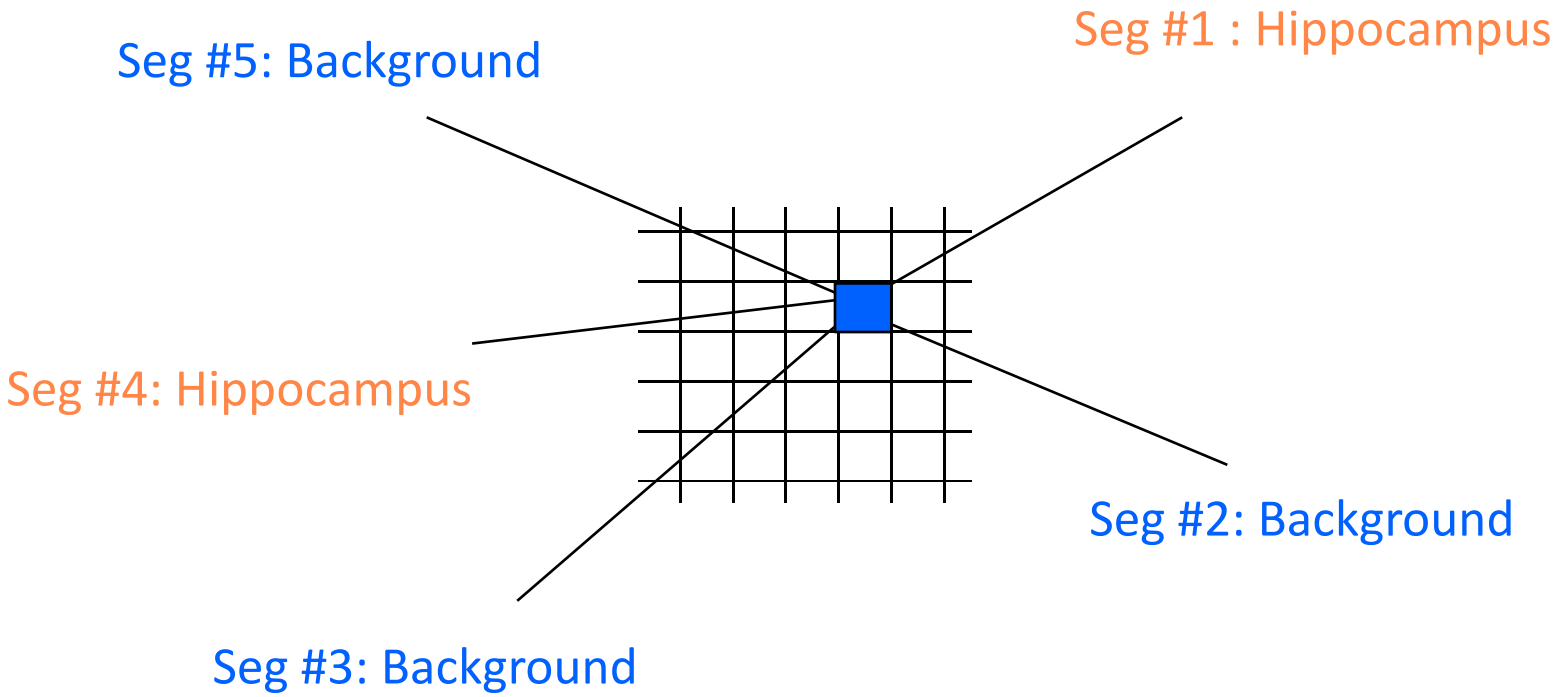
Segmentation via registration



Multi-atlas label propagation



Label fusion via majority voting



Multi-atlas label propagation

Advantages

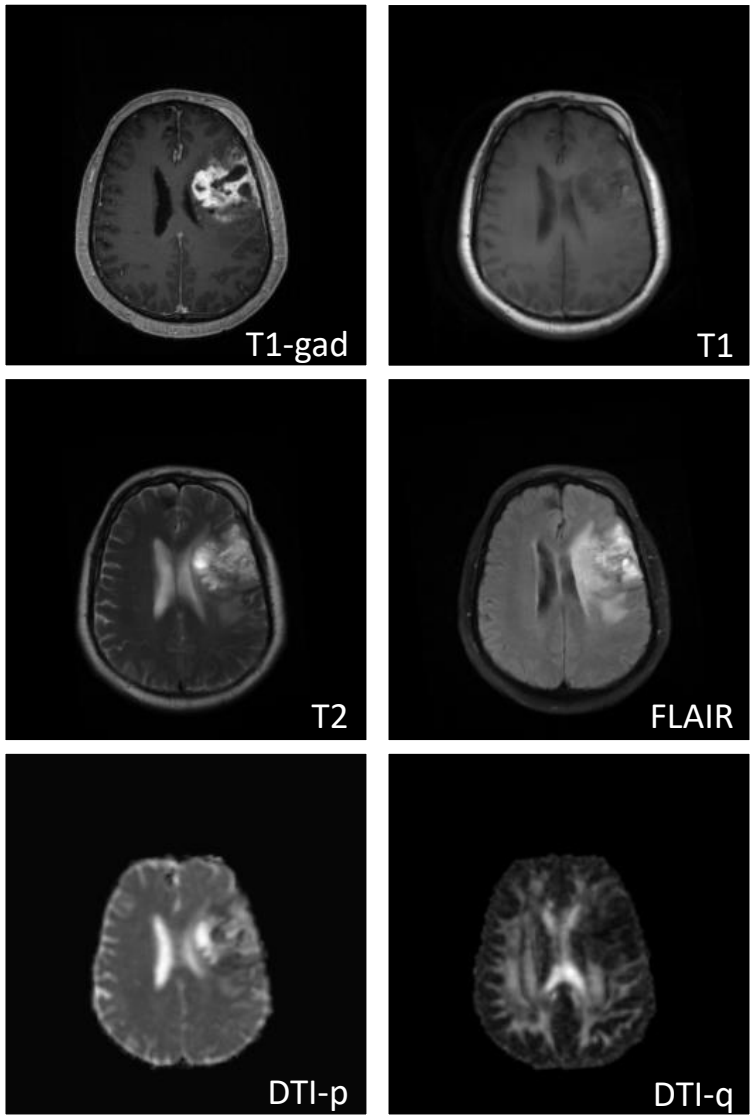
- robust and accurate (like ensembles)
- yields plausible segmentations
- fully automatic

Disadvantages

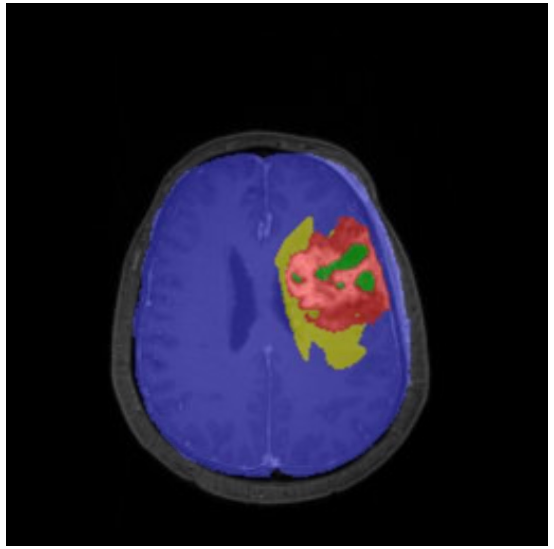
- computationally expensive
- cannot deal well with abnormalities
- not suitable for tumour segmentation

Random forests

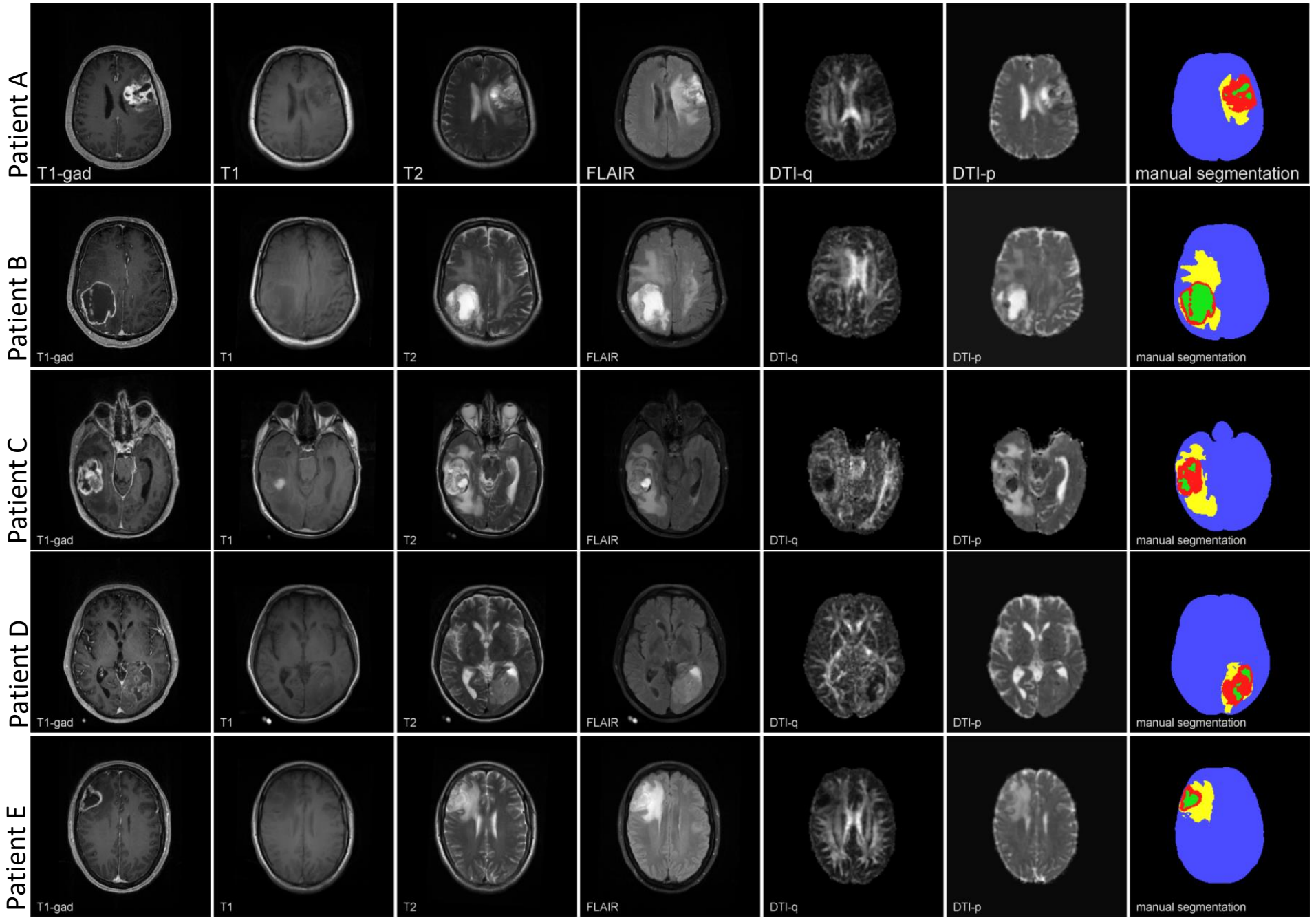
Multi-channel 3D MRI input data



Segmentation of tumorous tissues:

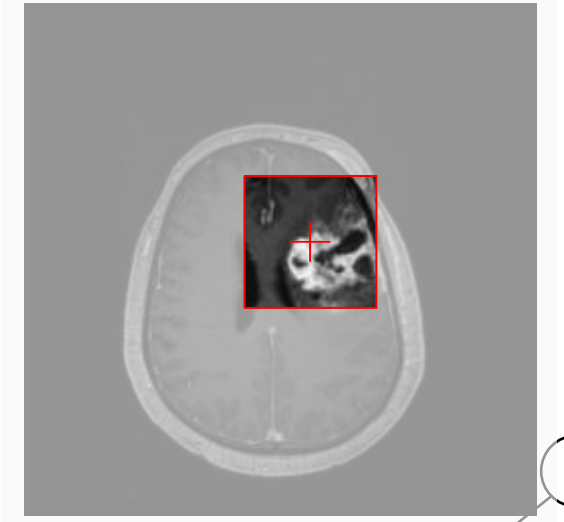


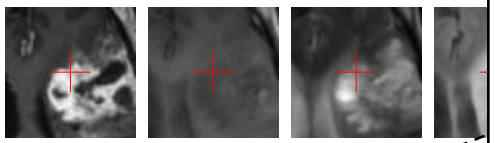
- Active cells
- Necrotic core
- Edema
- Background


Patient A
T1-gad
T1
T2
FLAIR
DTI-q
DTI-p
manual segmentation
Patient B
T1-gad
T1
T2
FLAIR
DTI-q
DTI-p
manual segmentation
Patient C
T1-gad
T1
T2
FLAIR
DTI-q
DTI-p
manual segmentation
Patient D
T1-gad
T1
T2
FLAIR
DTI-q
DTI-p
manual segmentation
Patient E
T1-gad
T1
T2
FLAIR
DTI-q
DTI-p
manual segmentation

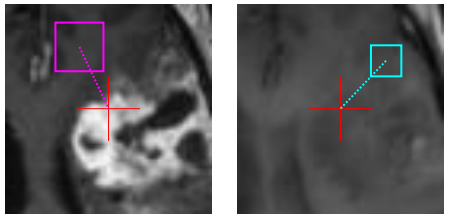
Training: Growing the trees

Determine tests to distinguish between tissue classes



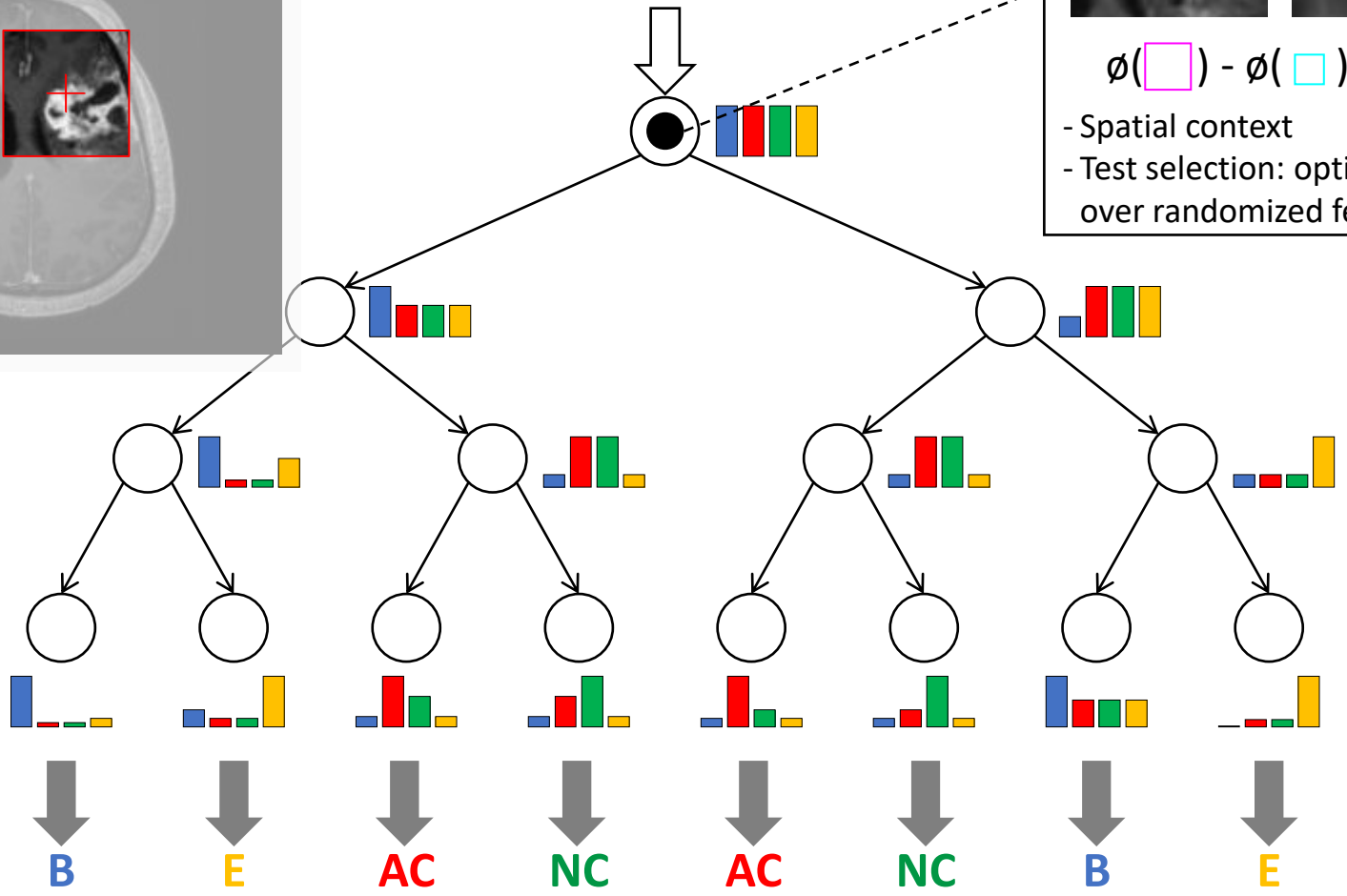


test example:



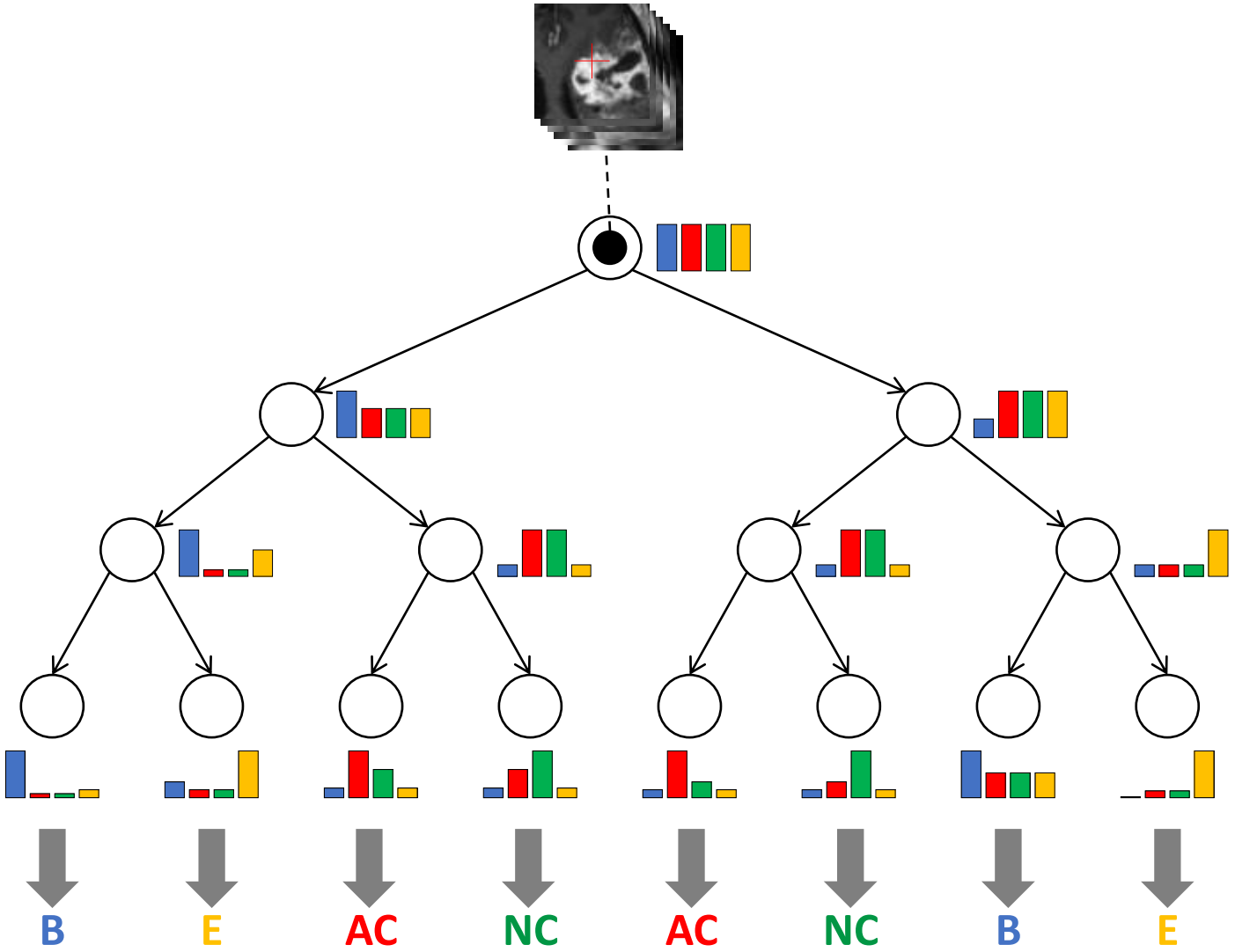
ø( ) - ø( ) > θ ?

- Spatial context
- Test selection: optimization over randomized features



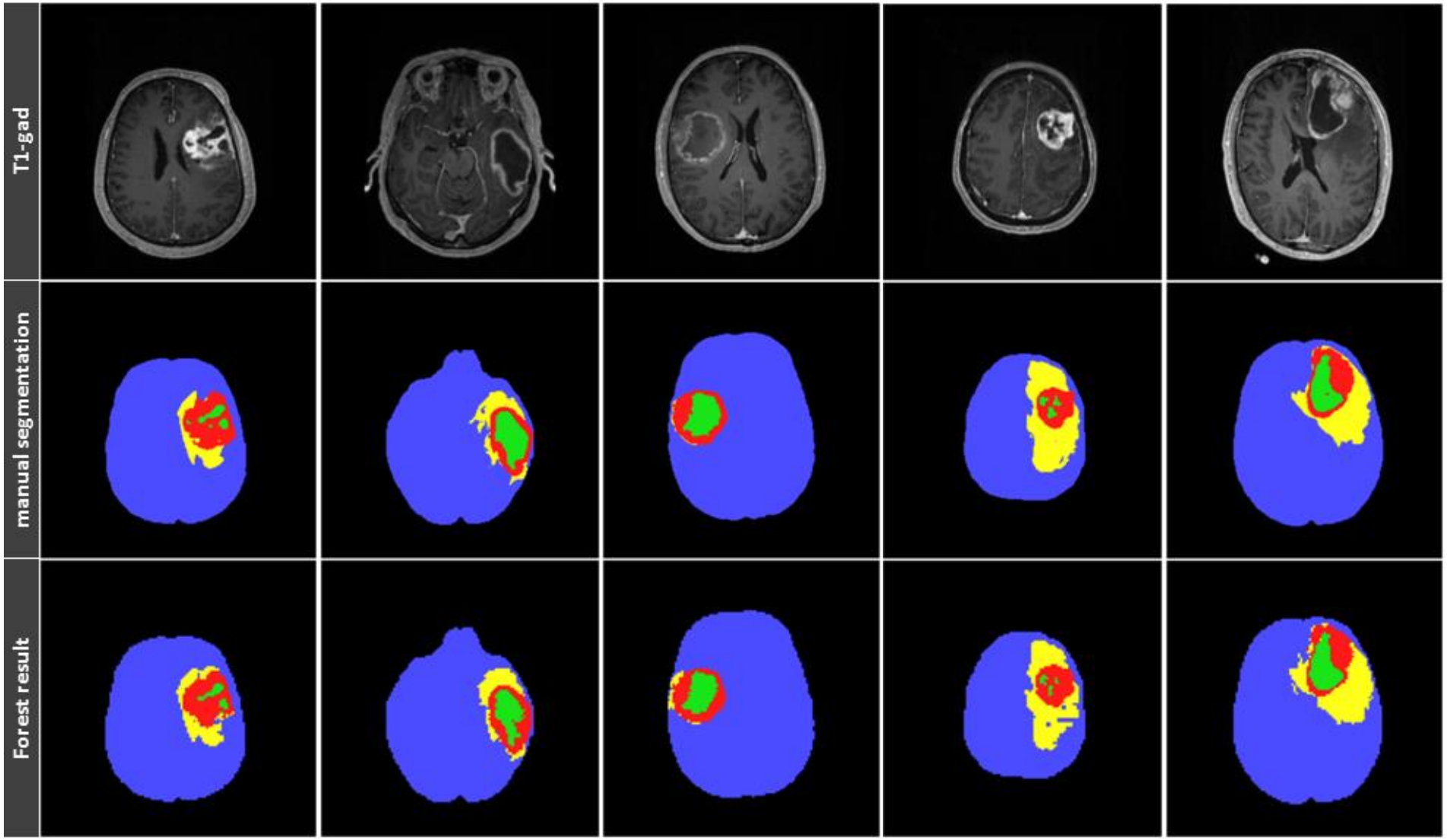
Testing: Traversing the trees

Apply the learned tests to classify tissue points



https://www.doc.ic.ac.uk/~bglocker/pdfs/zikic2012miccai.pdf
https://www.doc.ic.ac.uk/~bglocker/pdfs/zikic2012brats.pdf

Visual results



Random forests

Advantages

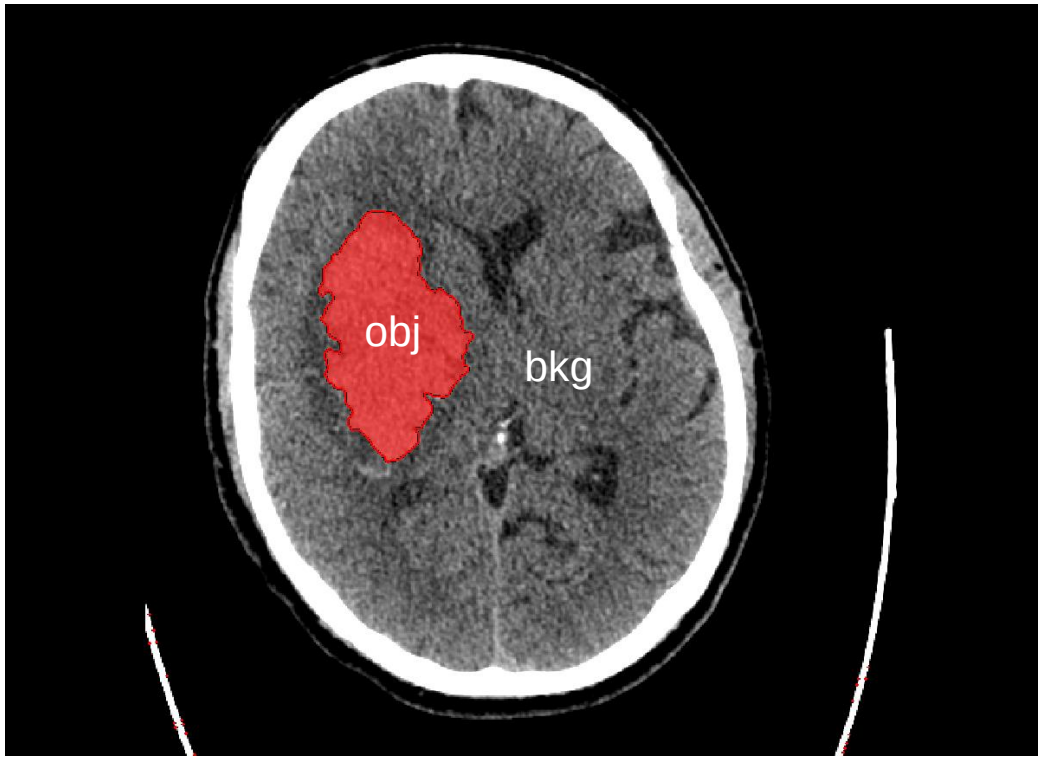
- ensemble classifiers are robust and accurate
- computationally efficient
- fully automatic

Disadvantages

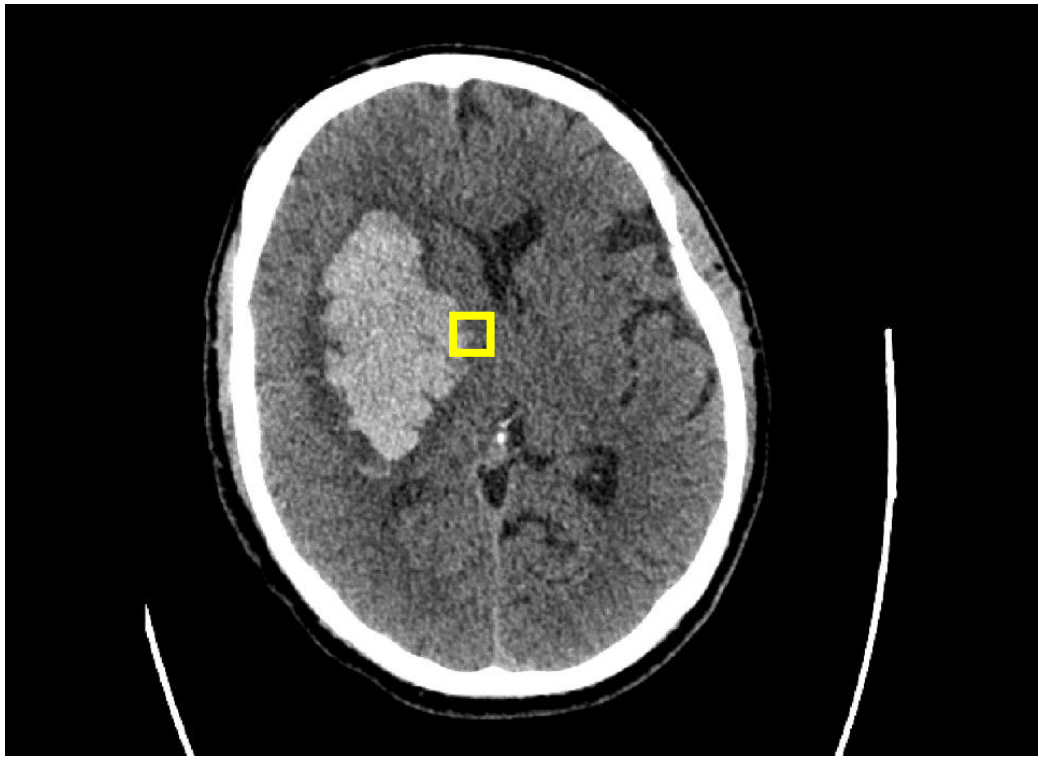
- shallow model, no hierarchical features
- no guarantees on connectedness

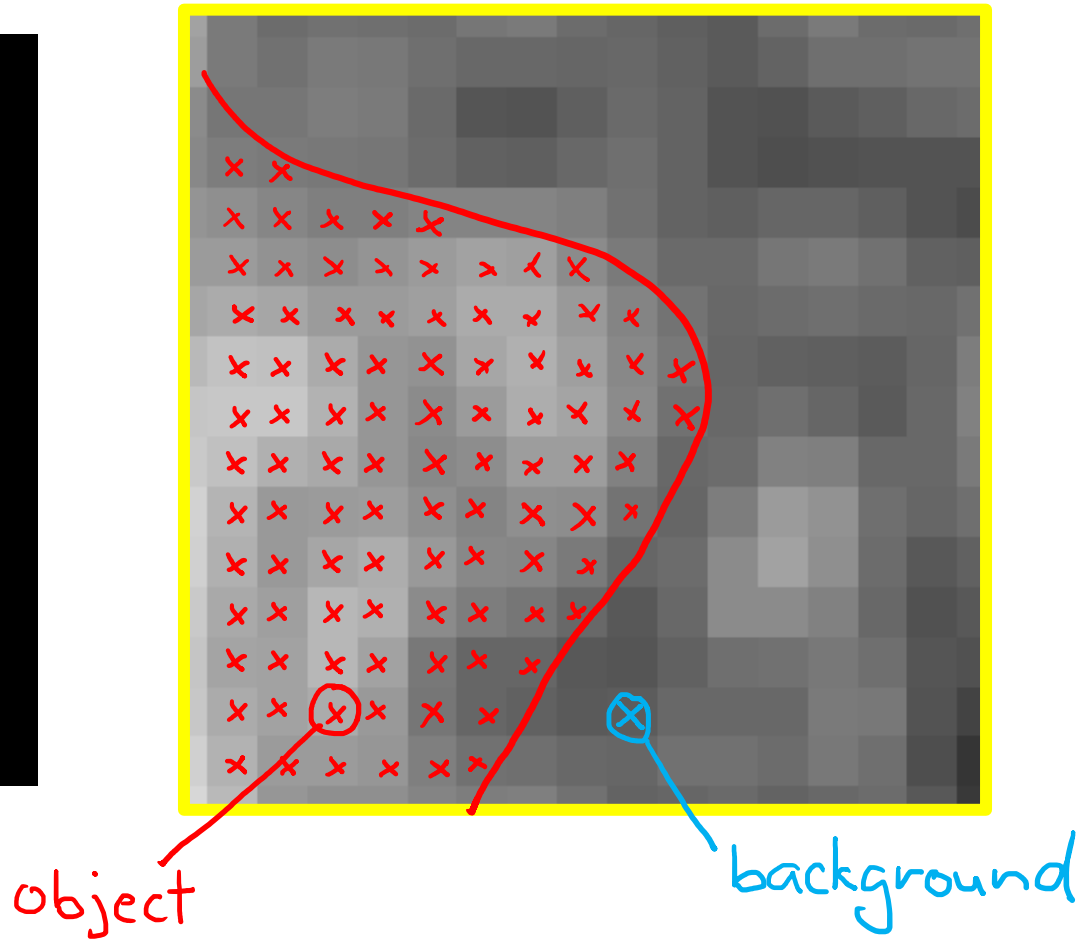
…and then…

Segmentation via dense classification

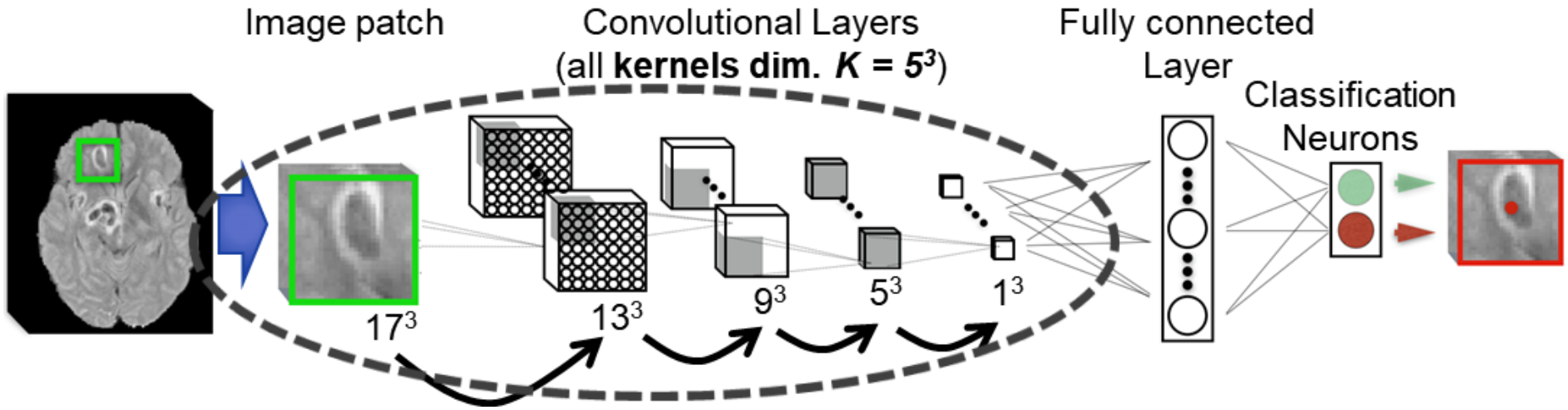


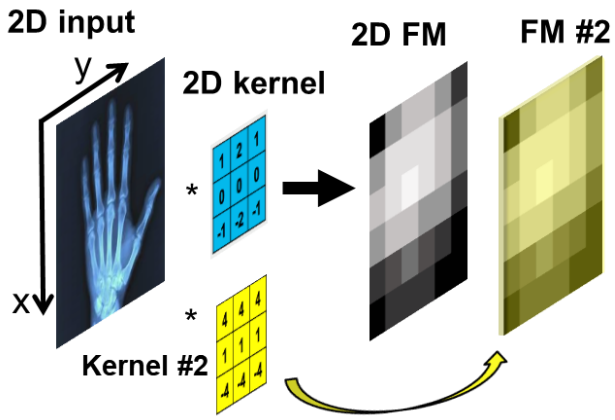
Segmentation via dense classification

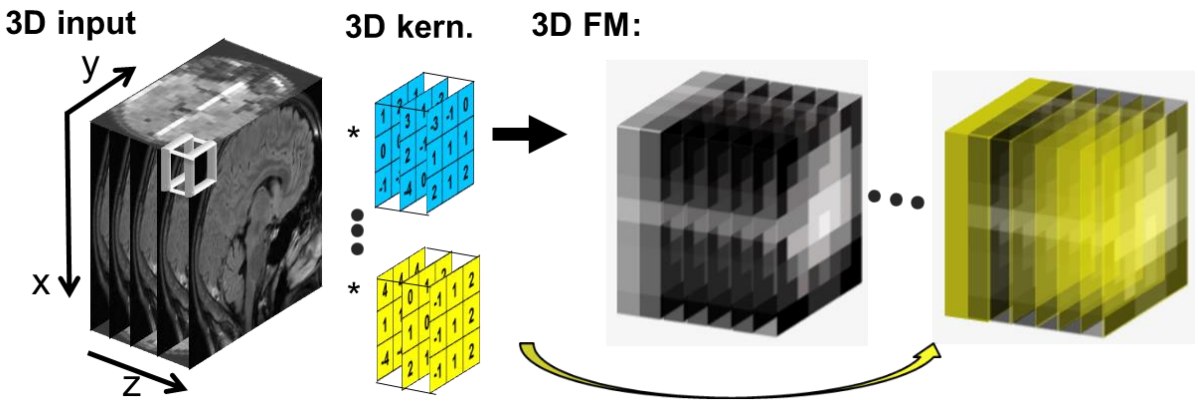




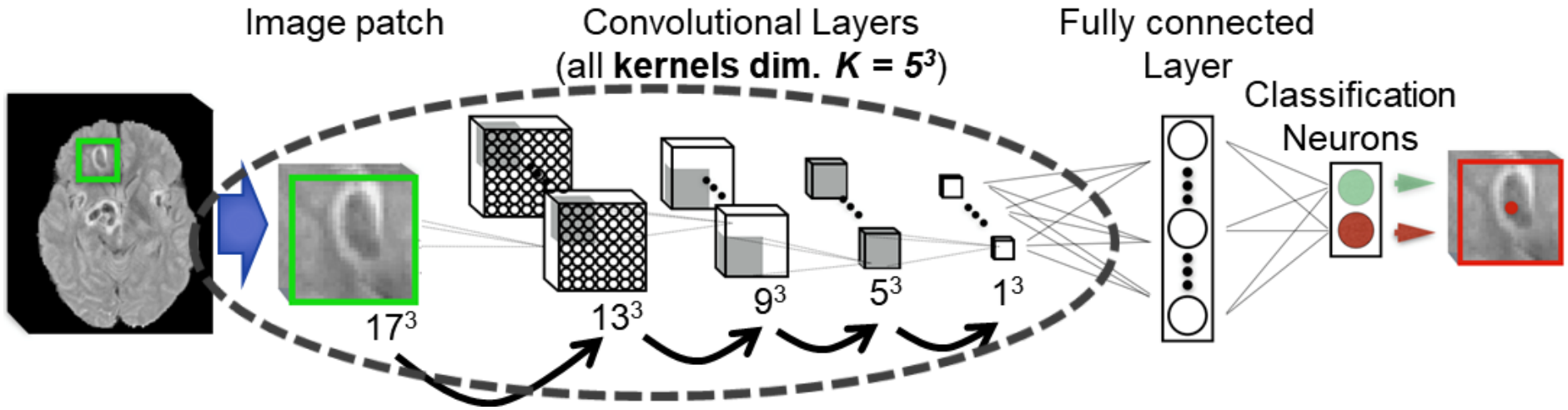
Segmentation via dense classification

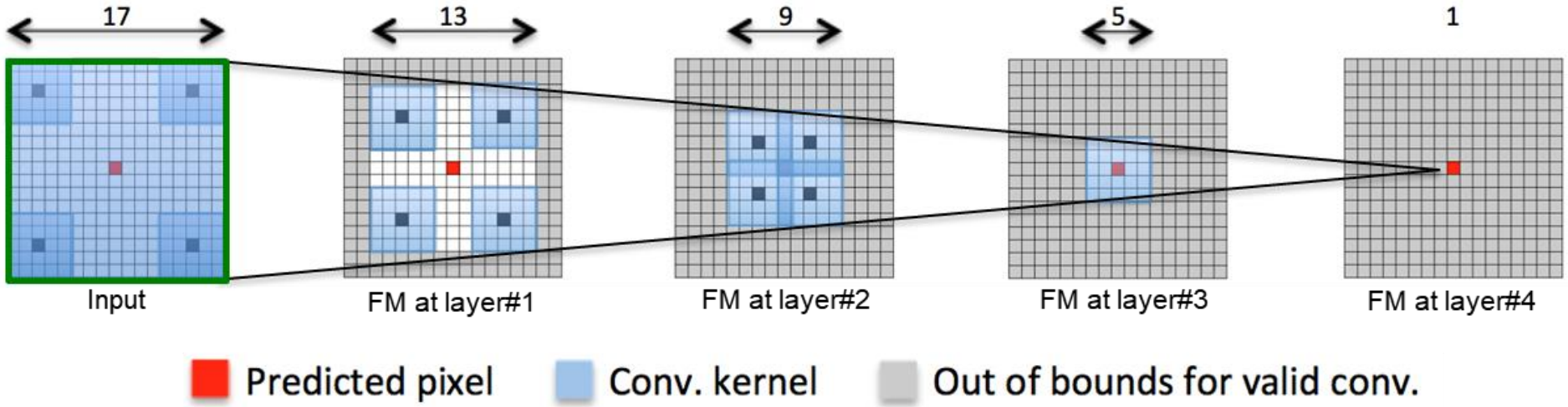




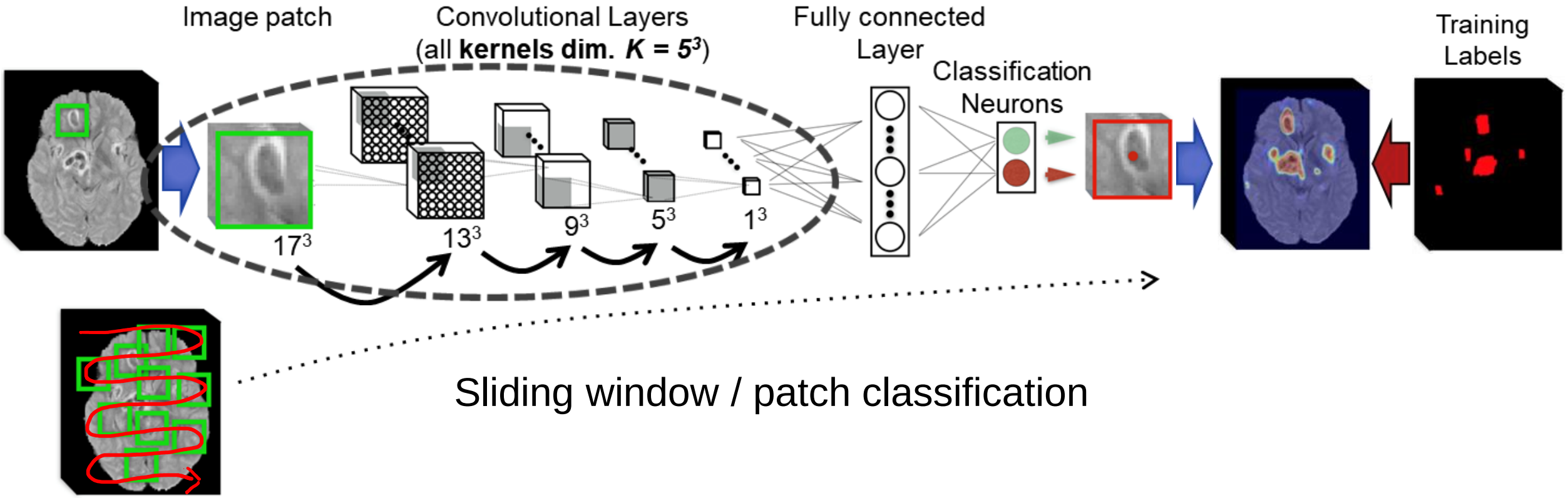


Segmentation via dense classification

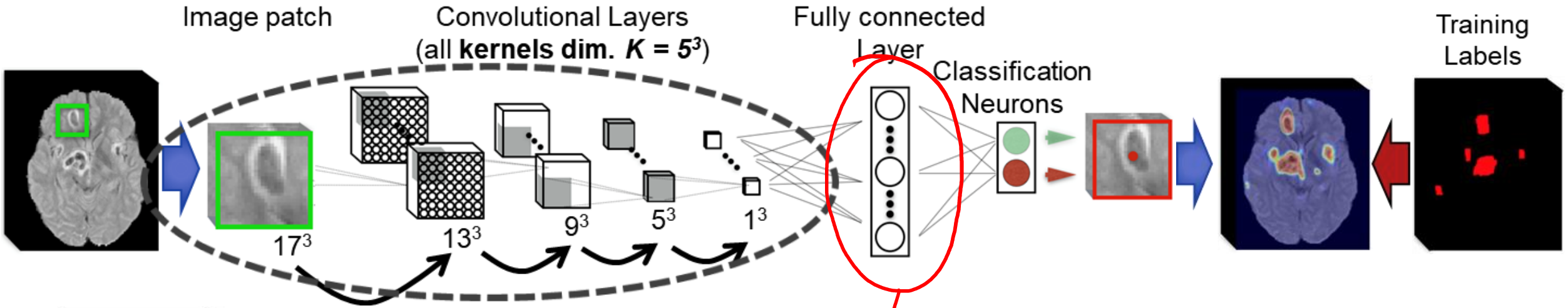




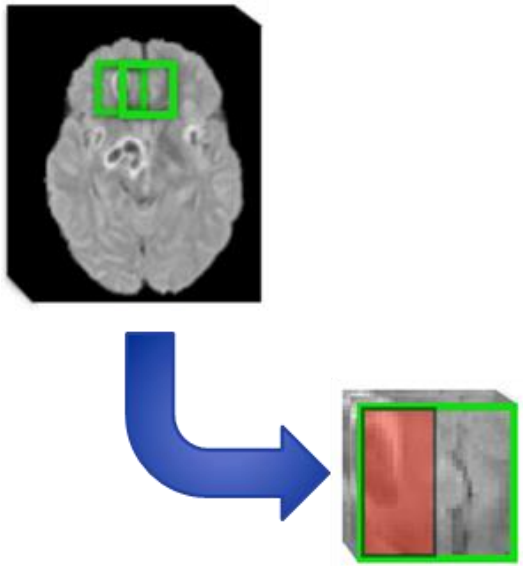
Segmentation via dense classification



Segmentation via dense classification

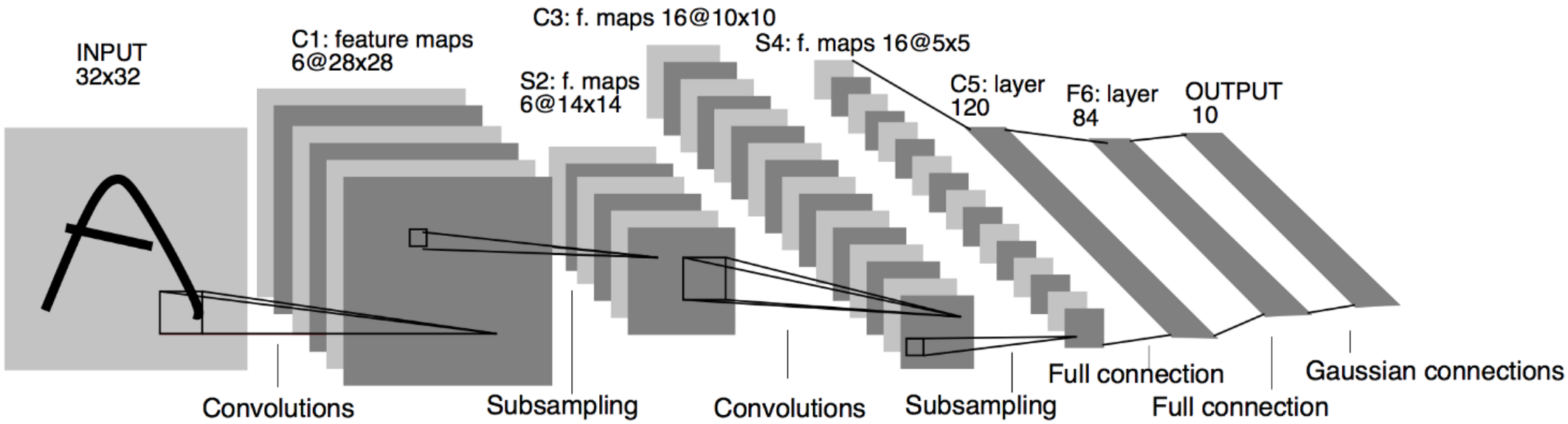


This is the crux



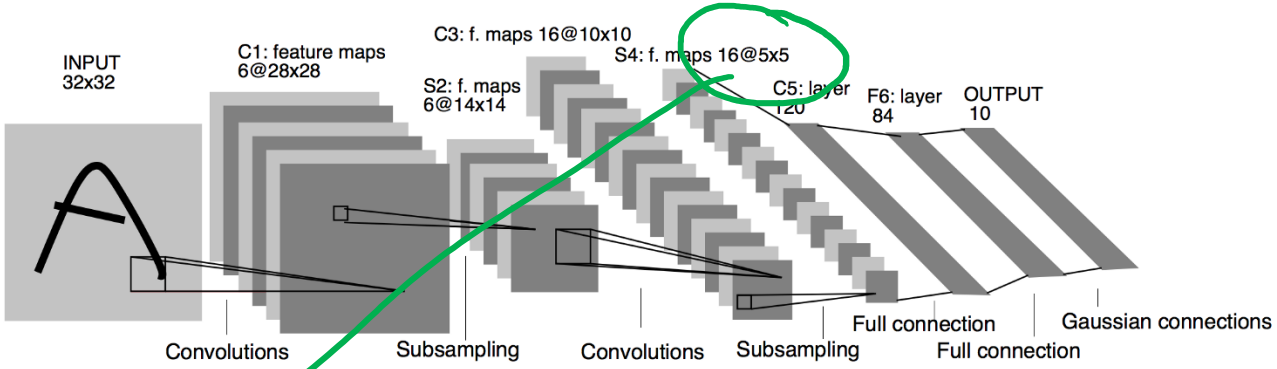
Very inefficient. Redundant computations!

Example: LeNet



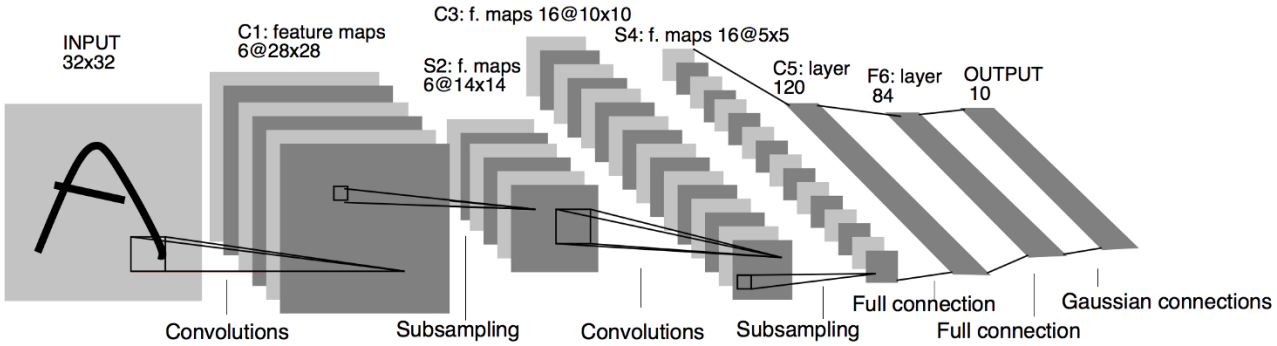
LeCun et al. 1998. Gradient-based learning applied to document recognition

LeNet in PyTorch




```python
class LeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.fc1 = nn.Linear(16 * ? * ?, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

LeNet in PyTorch



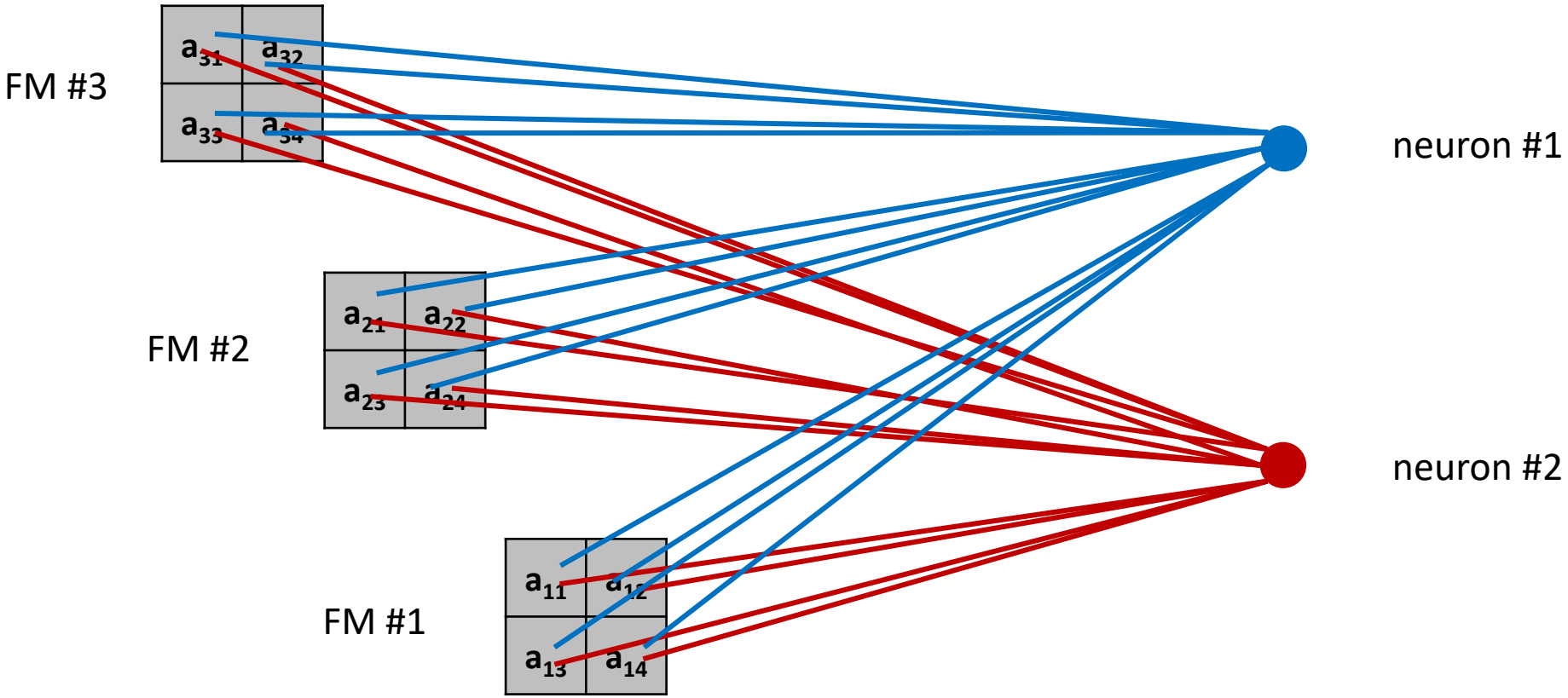

```python
class LeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

Question:
What happens when we feed in an image of size 64×64?

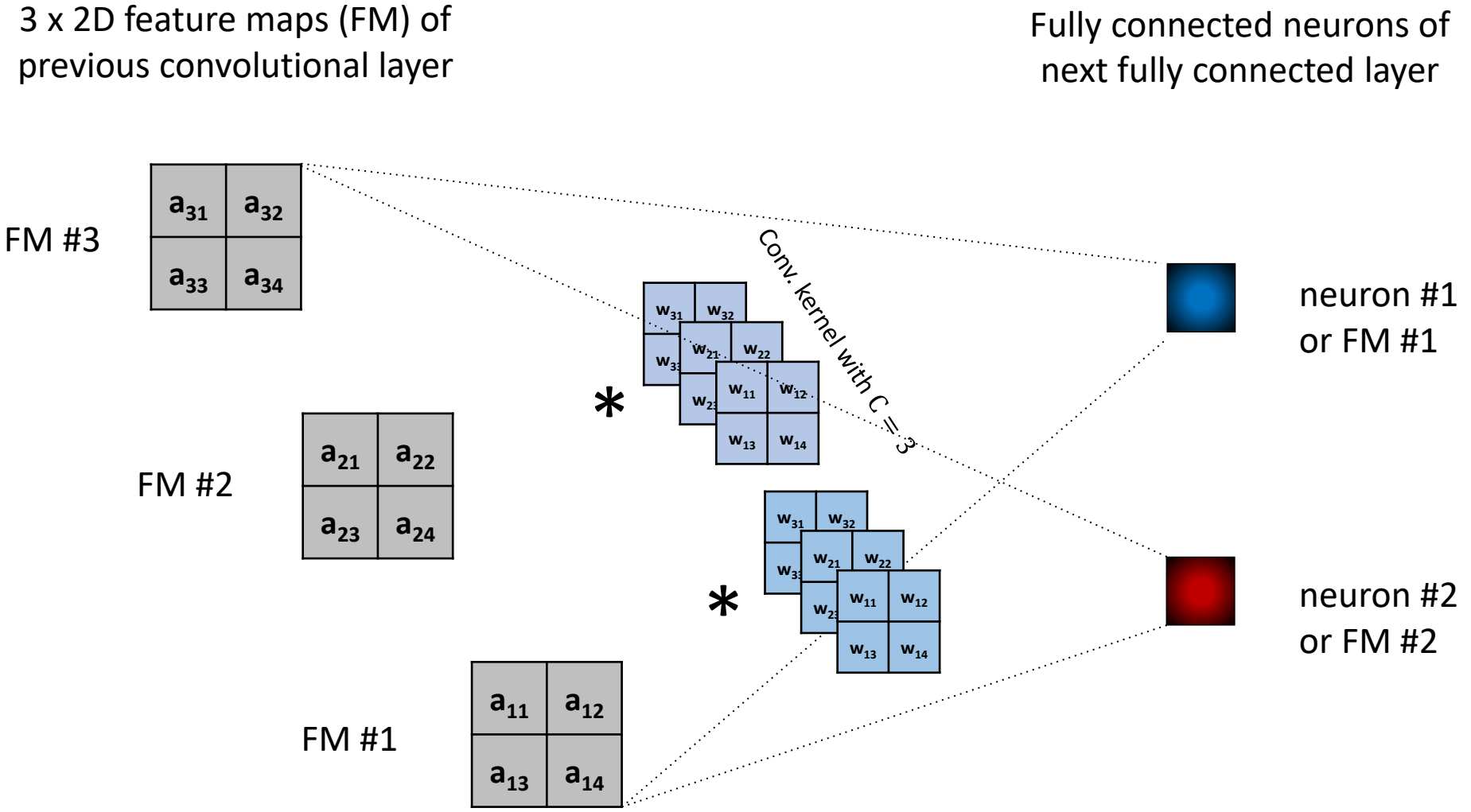
Fully connected to fully convolutional

3 x 2D feature maps (FM) of previous convolutional layer

Fully connected neurons of next fully connected layer



Fully connected to fully convolutional



Fully convolutional LeNet

```
class LeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

```
class FCLeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

Fully convolutional LeNet

```
class LeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

```
class FCLeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.conv3 = nn.Conv2d(??, ???, kernel_size=?)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

Fully convolutional LeNet

```
class LeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

```
class FCLeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.conv3 = nn.Conv2d(16, 120, kernel_size=5)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

Fully convolutional LeNet

```
class LeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

```
class FCLeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.conv3 = nn.Conv2d(16, 120, kernel_size=5)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

Fully convolutional LeNet

```python
class LeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

```python
class FCLeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.conv3 = nn.Conv2d(16, 120, kernel_size=5)
        self.conv4 = nn.Conv2d(120, ??, kernel_size=?)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

Fully convolutional LeNet

```python
class LeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

```python
class FCLeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.conv3 = nn.Conv2d(16, 120, kernel_size=5)
        self.conv4 = nn.Conv2d(120, 84, kernel_size=1)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

Fully convolutional LeNet

```
class LeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

```
class FCLeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.conv3 = nn.Conv2d(16, 120, kernel_size=5)
        self.conv4 = nn.Conv2d(120, 84, kernel_size=1)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

Fully convolutional LeNet

```
class LeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

```
class FCLeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.conv3 = nn.Conv2d(16, 120, kernel_size=5)
        self.conv4 = nn.Conv2d(120, 84, kernel_size=1)
        self.conv5 = nn.Conv2d(84, ??, kernel_size=?)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

Fully convolutional LeNet

```python
class LeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

```python
class FCLeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.conv3 = nn.Conv2d(16, 120, kernel_size=5)
        self.conv4 = nn.Conv2d(120, 84, kernel_size=1)
        self.conv5 = nn.Conv2d(84, 10, kernel_size=1)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

Fully convolutional LeNet

```python
class LeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

```python
class FCLeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.conv3 = nn.Conv2d(16, 120, kernel_size=5)
        self.conv4 = nn.Conv2d(120, 84, kernel_size=1)
        self.conv5 = nn.Conv2d(84, 10, kernel_size=1)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

Fully convolutional LeNet

```python
class LeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

```python
class FCLeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.conv3 = nn.Conv2d(16, 120, kernel_size=5)
        self.conv4 = nn.Conv2d(120, 84, kernel_size=1)
        self.conv5 = nn.Conv2d(84, 10, kernel_size=1)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.conv3(x))
        x = F.relu(self.conv4(x))
        x = self.conv5(x)
        return F.log_softmax(x, dim=1)
```

Fully convolutional LeNet

```python
class LeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

```python
class FCLeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.conv3 = nn.Conv2d(16, 120, kernel_size=5)
        self.conv4 = nn.Conv2d(120, 84, kernel_size=1)
        self.conv5 = nn.Conv2d(84, 10, kernel_size=1)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.conv3(x))
        x = F.relu(self.conv4(x))
        x = self.conv5(x)
        return F.log_softmax(x, dim=1)
```

Fully convolutional LeNet

```python
class LeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

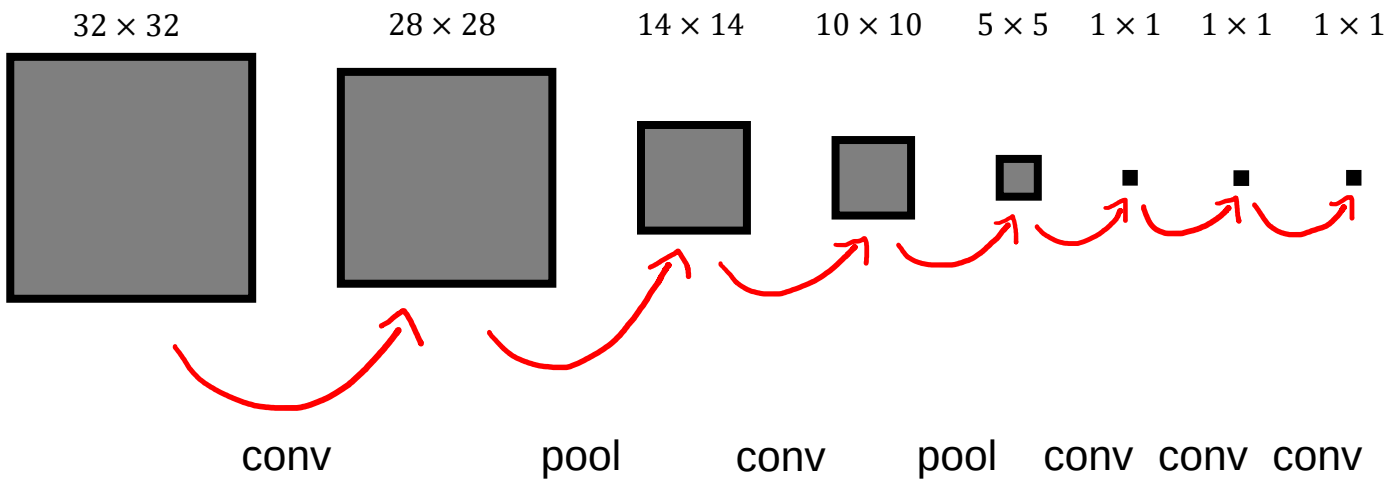
```python
class FCLeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.conv3 = nn.Conv2d(16, 120, kernel_size=5)
        self.conv4 = nn.Conv2d(120, 84, kernel_size=1)
        self.conv5 = nn.Conv2d(84, 10, kernel_size=1)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size(0), -1)
        x = F.relu(self.conv3(x))
        x = F.relu(self.conv4(x))
        x = self.conv5(x)
        return F.log_softmax(x, dim=1)
```

Fully convolutional LeNet

Question:
What happens when we feed in an image of size 64×64?

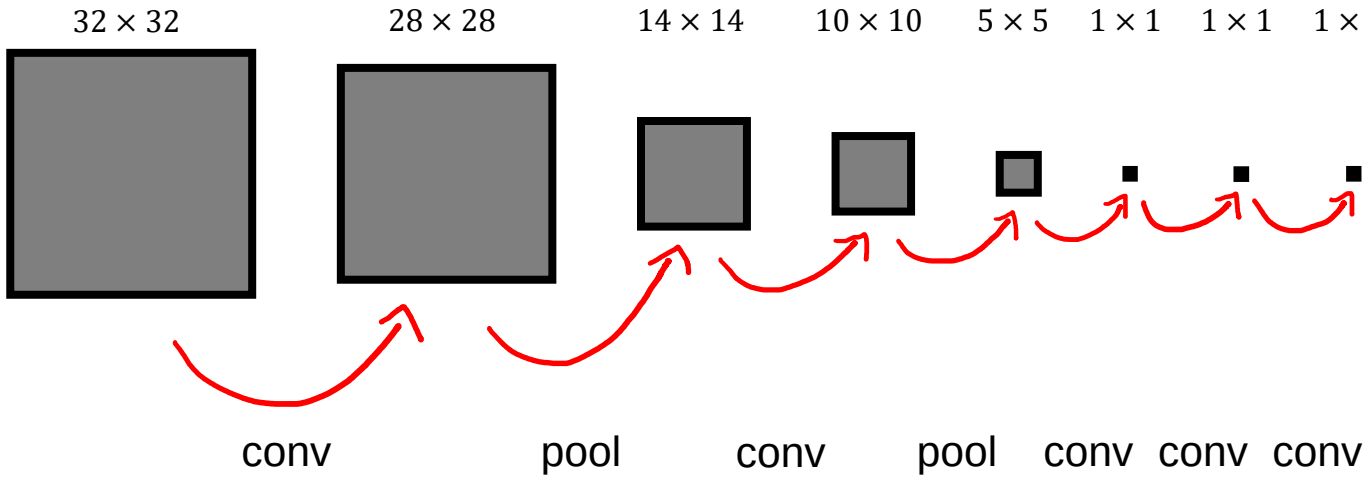
```
class FCLeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.conv3 = nn.Conv2d(16, 120, kernel_size=5)
        self.conv4 = nn.Conv2d(120, 84, kernel_size=1)
        self.conv5 = nn.Conv2d(84, 10, kernel_size=1)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = F.relu(self.conv3(x))
        x = F.relu(self.conv4(x))
        x = self.conv5(x)
        return F.log_softmax(x, dim=1)
```

Fully convolutional LeNet

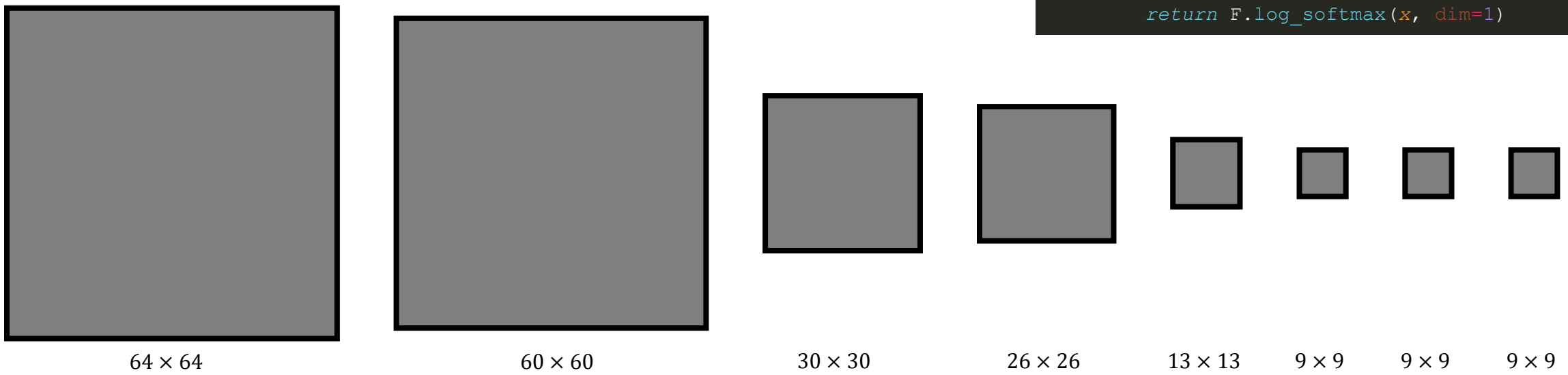



```python
class FCLeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.conv3 = nn.Conv2d(16, 120, kernel_size=5)
        self.conv4 = nn.Conv2d(120, 84, kernel_size=1)
        self.conv5 = nn.Conv2d(84, 10, kernel_size=1)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = F.relu(self.conv3(x))
        x = F.relu(self.conv4(x))
        x = self.conv5(x)
        return F.log_softmax(x, dim=1)
```

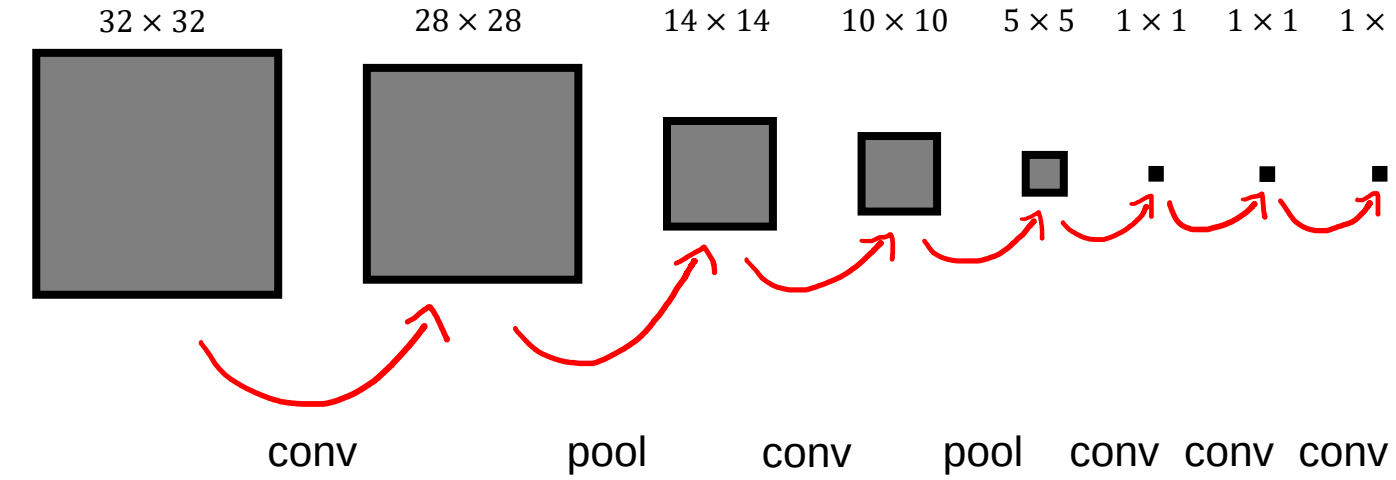
Fully convolutional LeNet



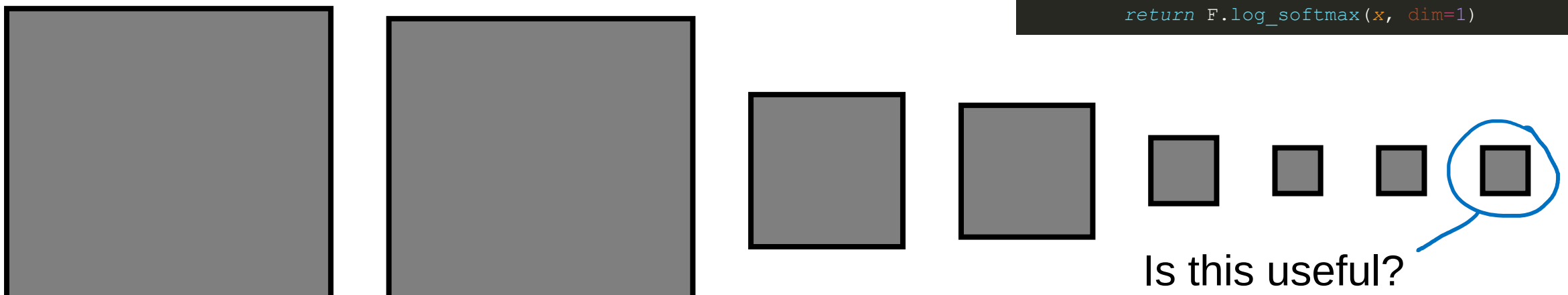

```
class FCLeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.conv3 = nn.Conv2d(16, 120, kernel_size=5)
        self.conv4 = nn.Conv2d(120, 84, kernel_size=1)
        self.conv5 = nn.Conv2d(84, 10, kernel_size=1)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = F.relu(self.conv3(x))
        x = F.relu(self.conv4(x))
        x = self.conv5(x)
        return F.log_softmax(x, dim=1)
```



Fully convolutional LeNet

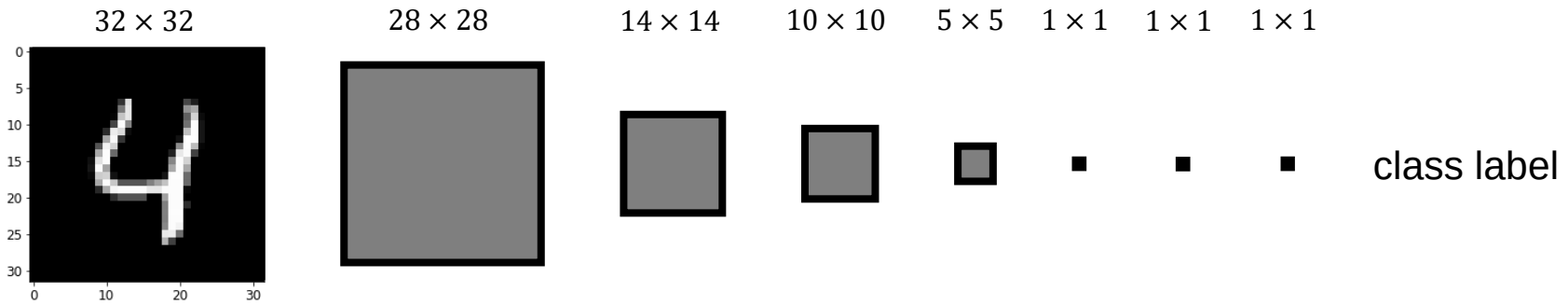


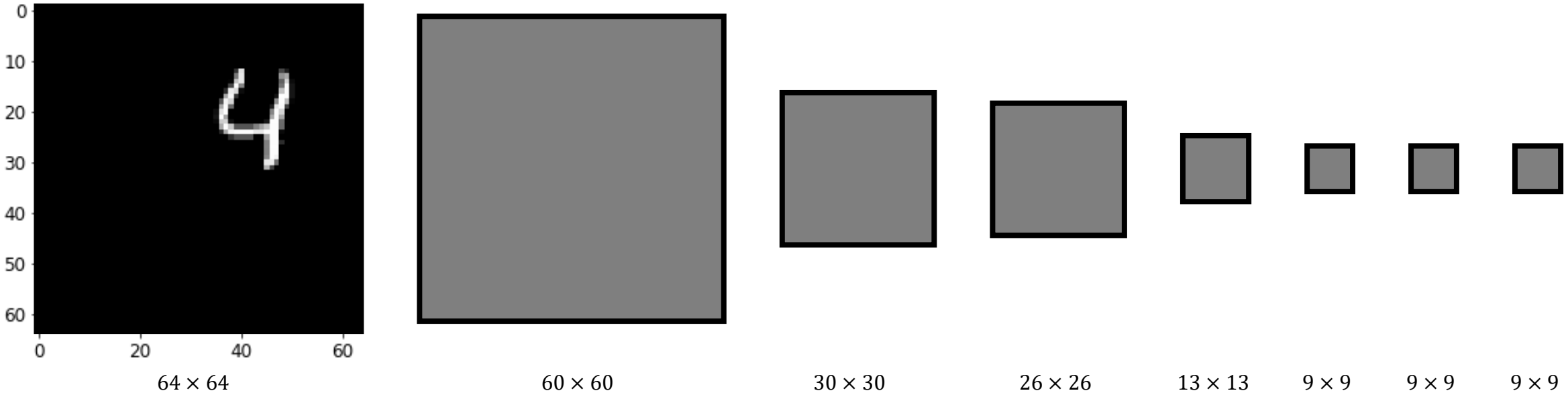

```
class FCLeNet(nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.conv3 = nn.Conv2d(16, 120, kernel_size=5)
        self.conv4 = nn.Conv2d(120, 84, kernel_size=1)
        self.conv5 = nn.Conv2d(84, 10, kernel_size=1)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = F.relu(self.conv3(x))
        x = F.relu(self.conv4(x))
        x = self.conv5(x)
        return F.log_softmax(x, dim=1)
```



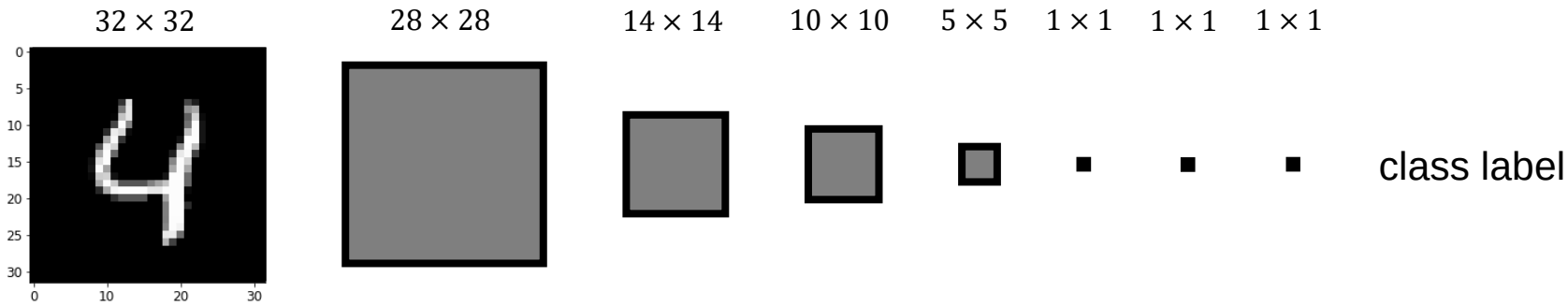
64 × 64 60 × 60 30 × 30 26 × 26 13 × 13 9 × 9 9 × 9 9 × 9

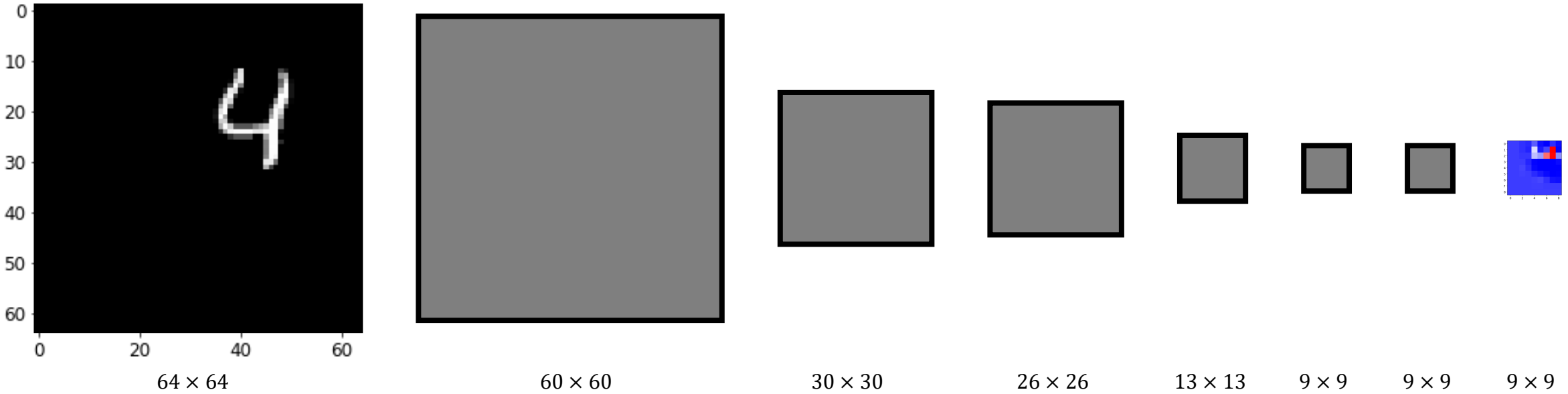
Beyond classification



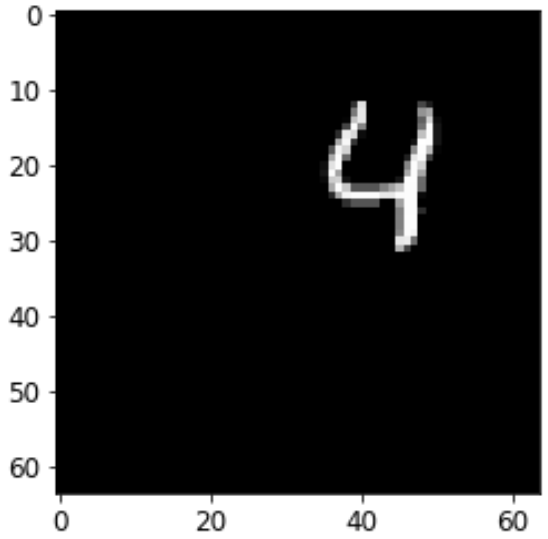


Beyond classification

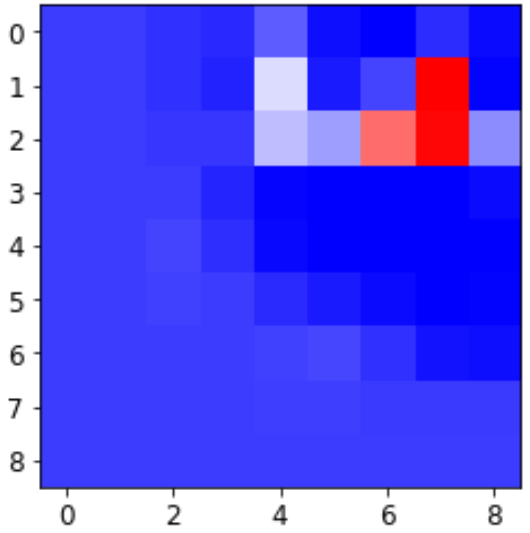




Beyond classification

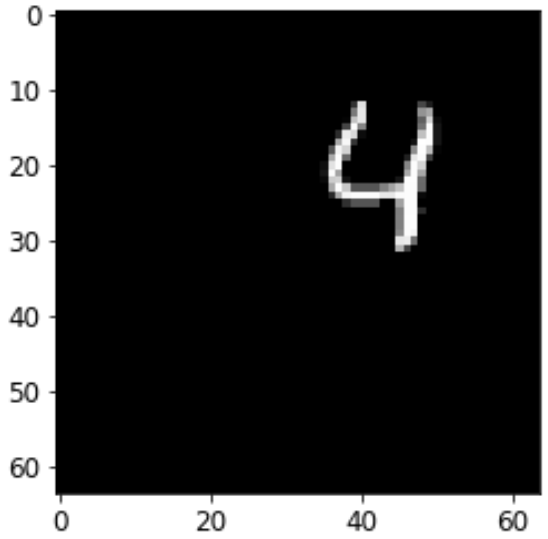


input image

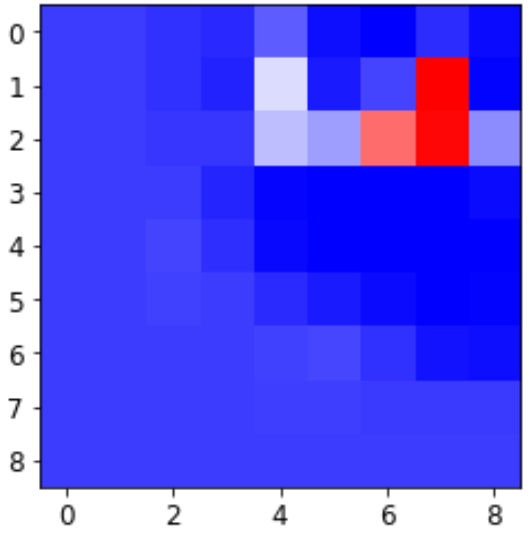


classification map
for label 4

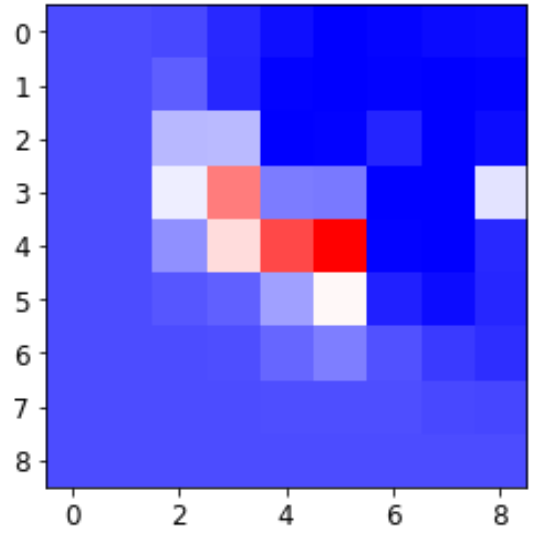
Beyond classification



input image

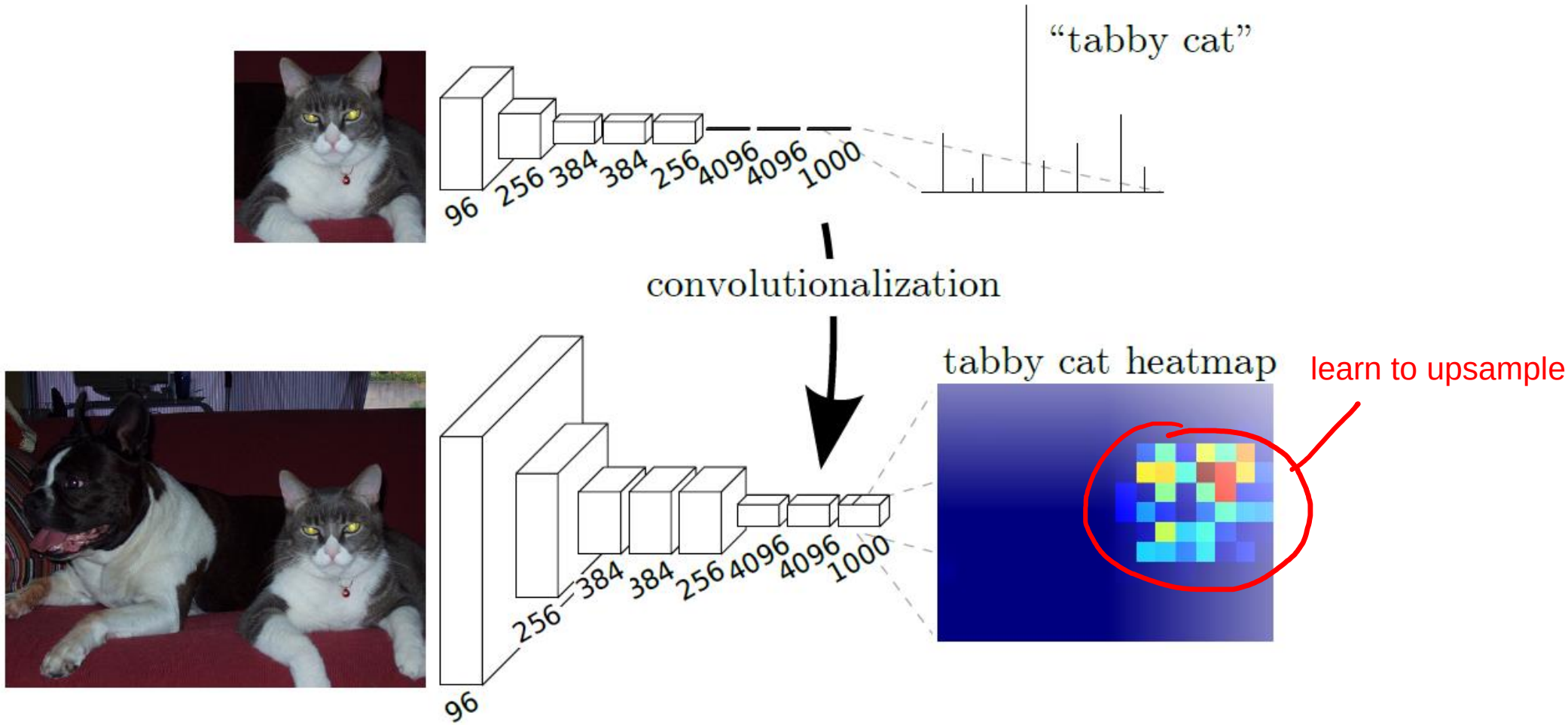


classification map
for label 4



classification map
for label 5

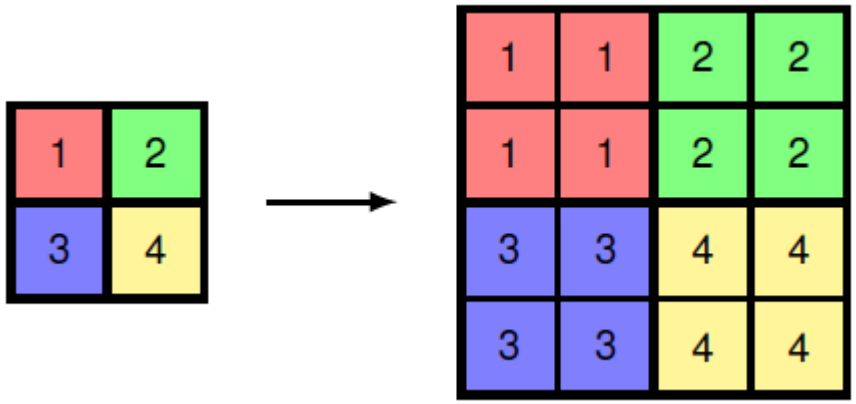
Beyond classification



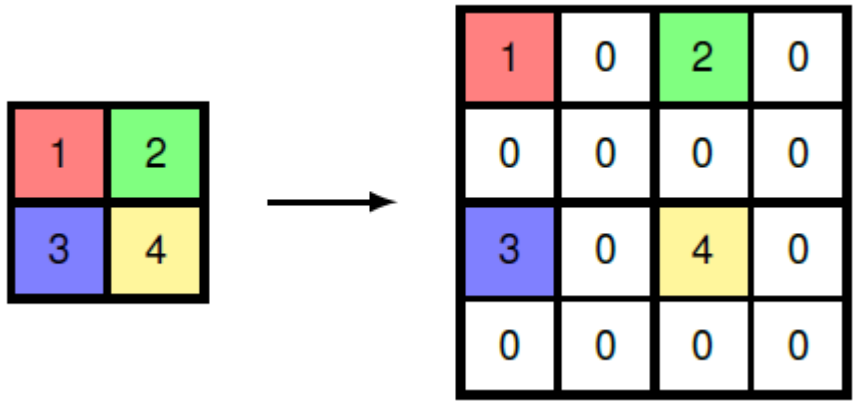
Long et al. 2015. [Fully Convolutional Networks for Semantic Segmentation]

Up-sampling

- Unpooling



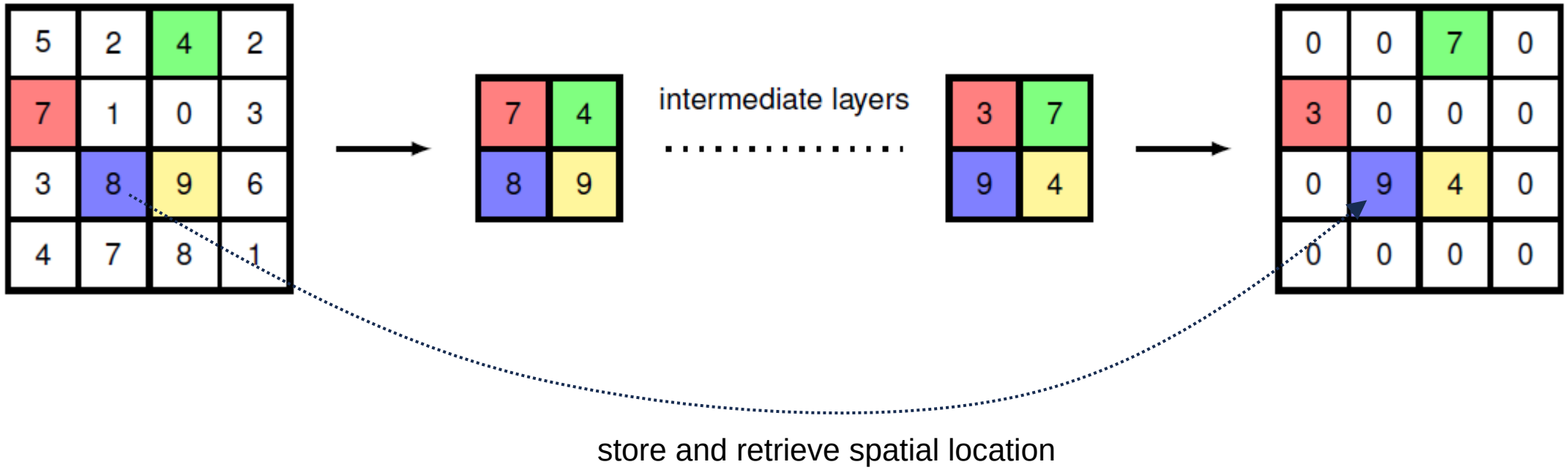
average (un)pooling



max (un)pooling

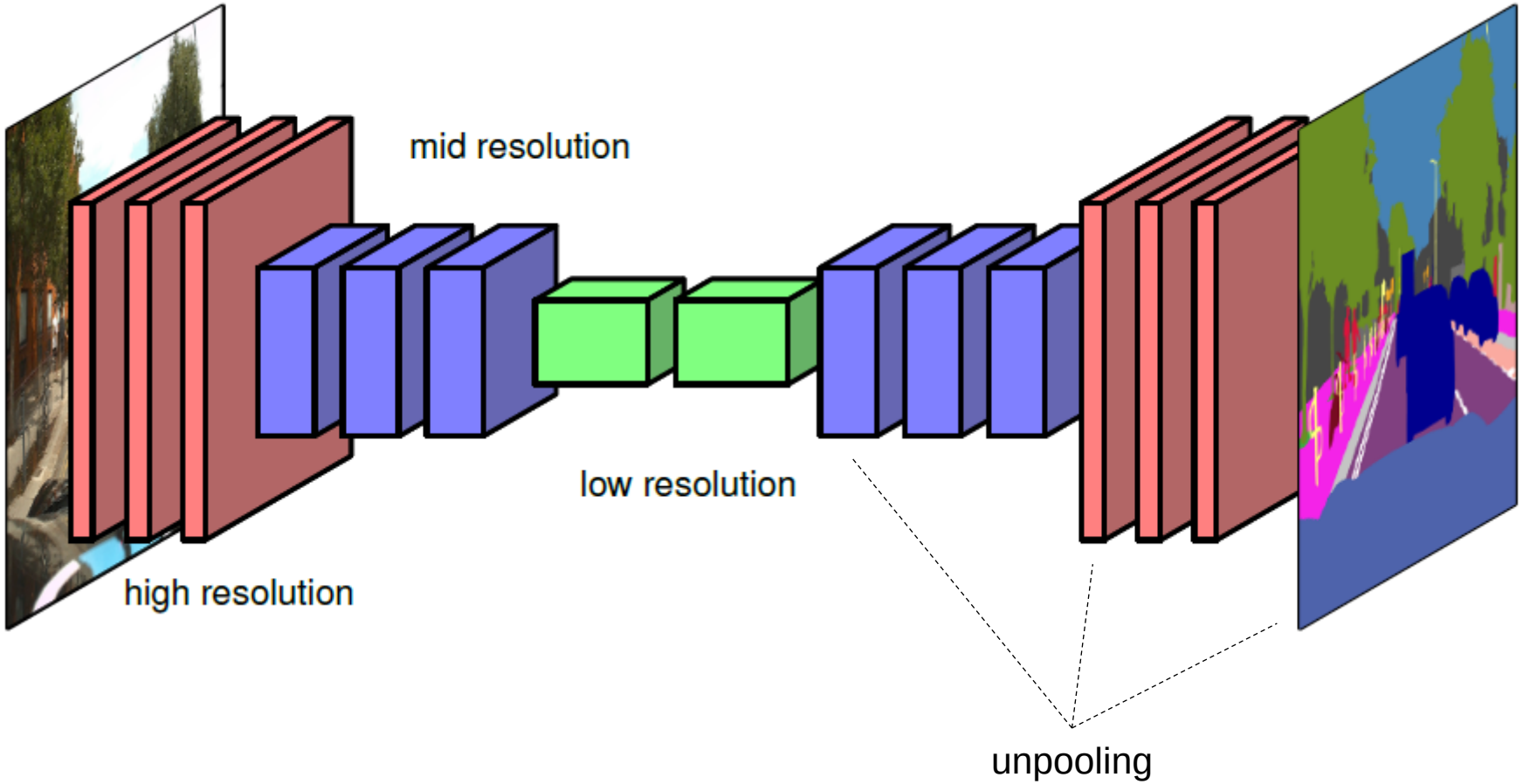
Up-sampling

- Unpooling with spatial information



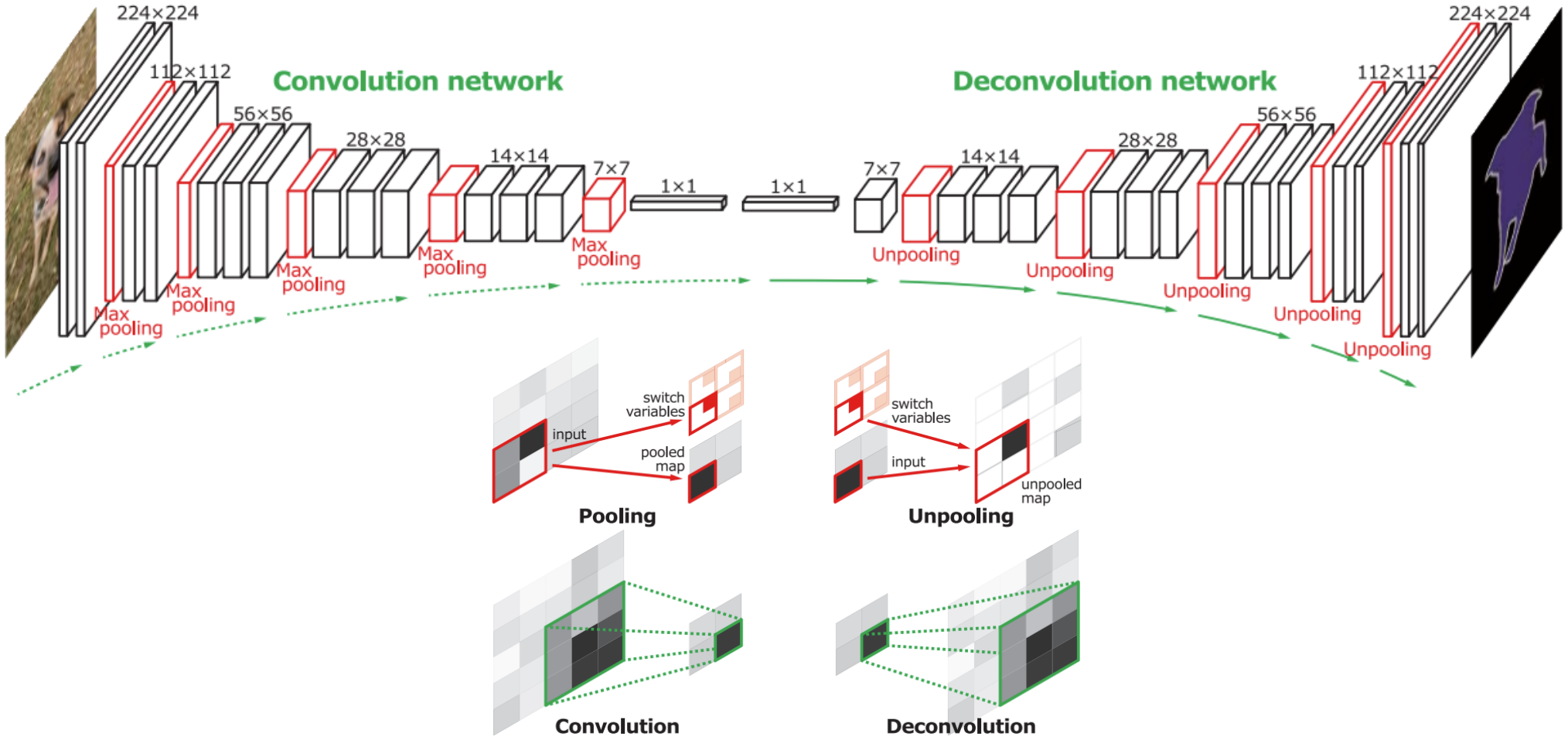
Bishop & Bishop. Deep Learning: Foundations and Concepts. 2023

Encoder-decoder network



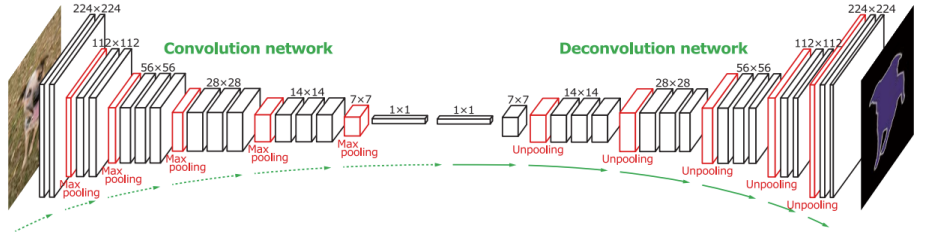
Encoder-decoder network

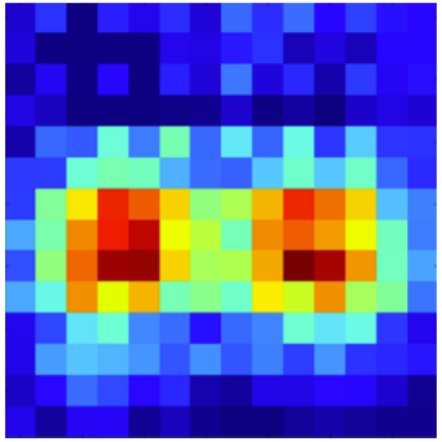
VGG-16



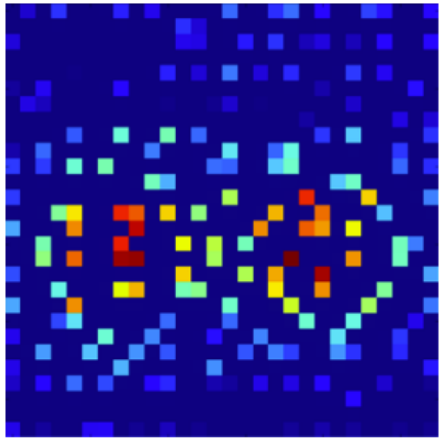
Noh et al. 2015. [Learning Deconvolution Network for Semantic Segmentation]

Encoder-decoder network

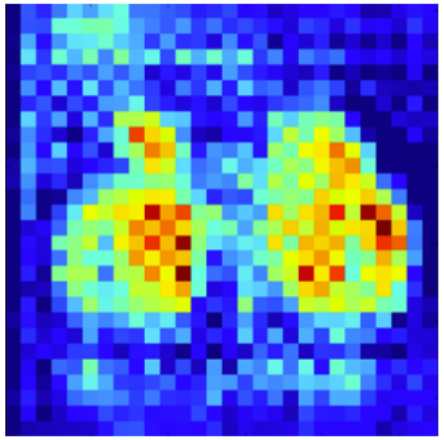




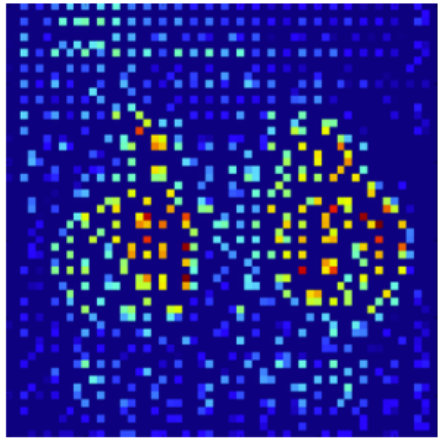
14 × 14 deconv



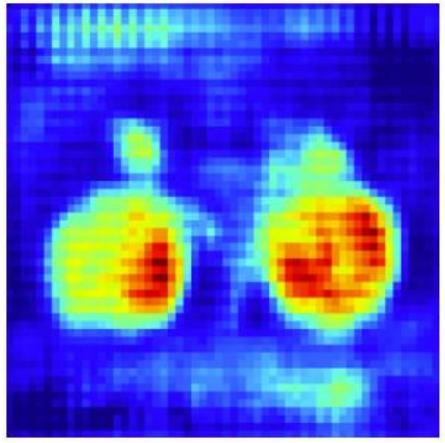
28 × 28 unpool



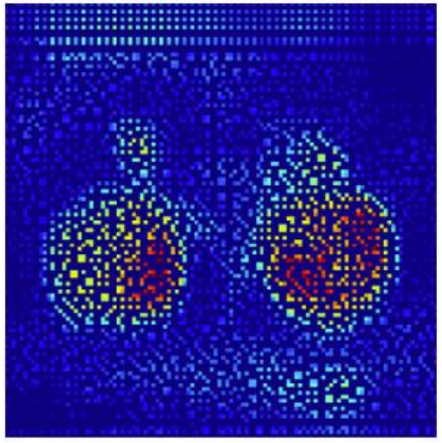
28 × 28 deconv



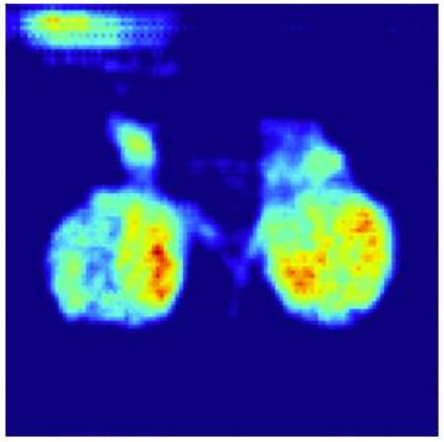
56 × 56 unpool



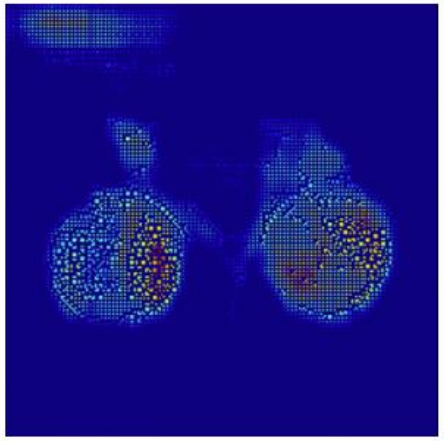
56 × 56 deconv



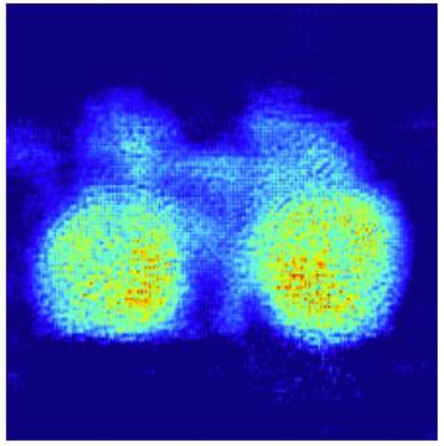
112 × 112 unpool



112 × 112 deconv



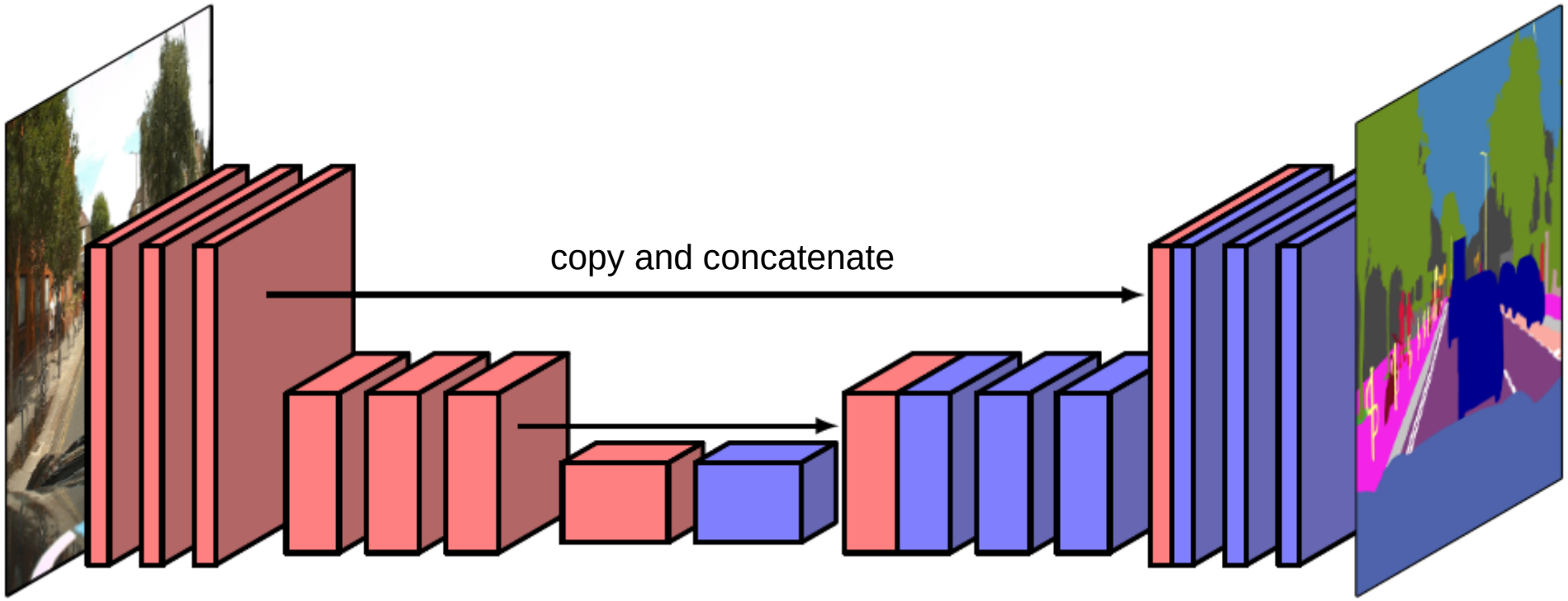
224 × 224 unpool



224 × 224 deconv

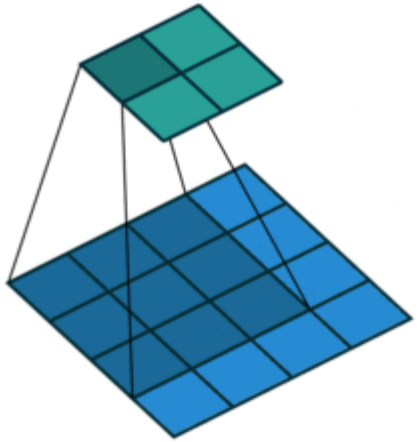
Noh et al. 2015. Learning Deconvolution Network for Semantic Segmentation

U-Net architecture

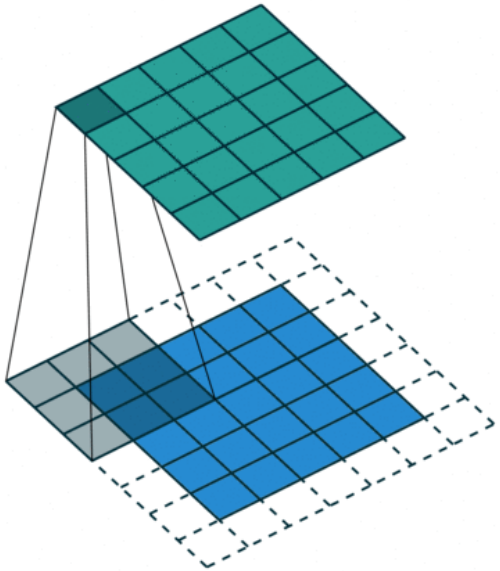


Ronneberger et al. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation

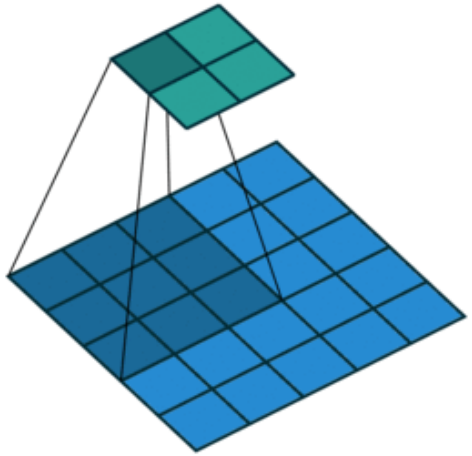
Convolutions



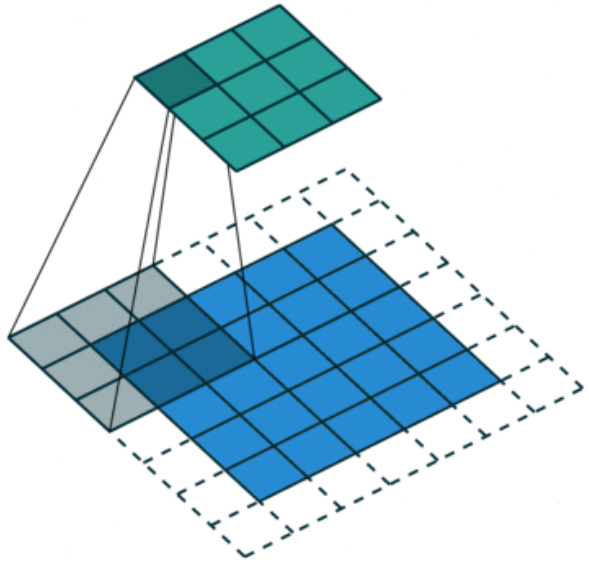
no padding, unit strides



padding



non-unit strides

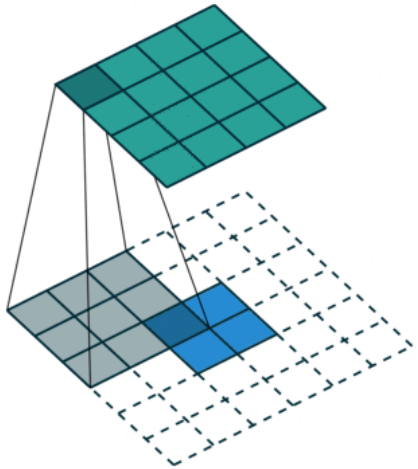


padding, non-unit strides

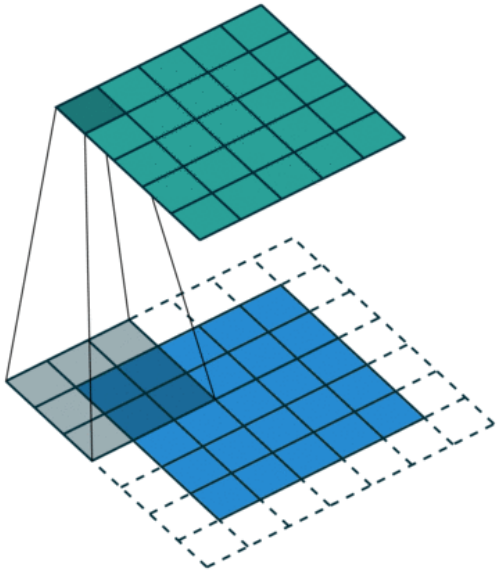
Convolutions can be used for ‘learned’ down-sampling

Source: https://github.com/vdumoulin/conv_arithmetic

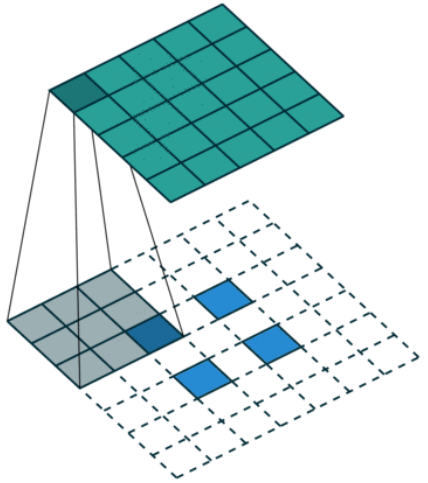
Transpose convolutions



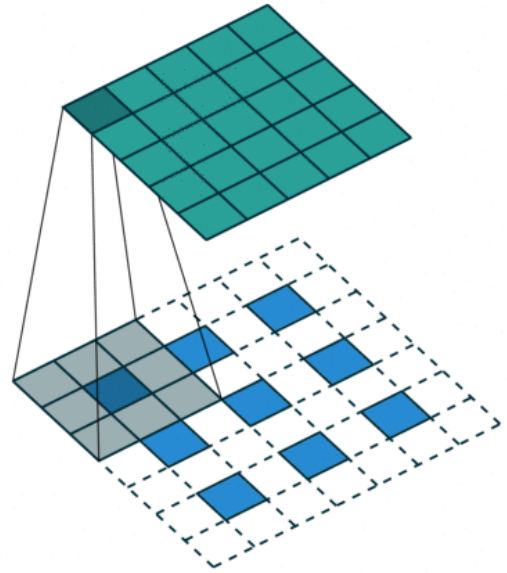
no padding, unit strides



padding



non-unit strides

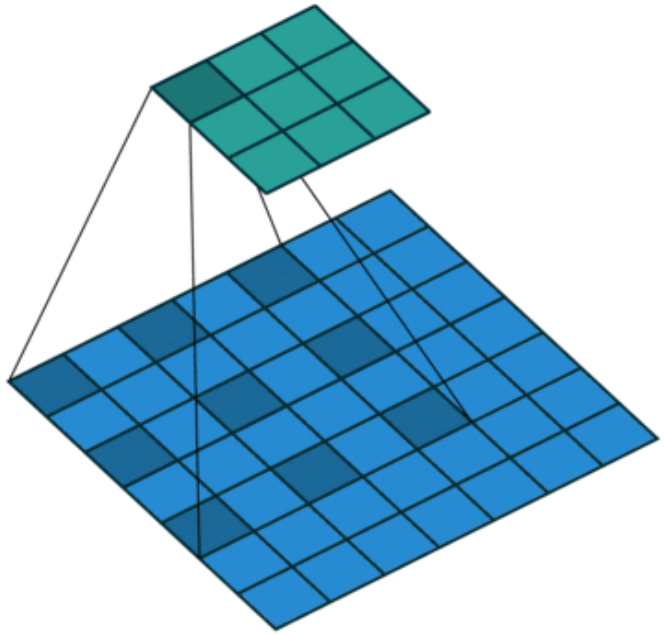


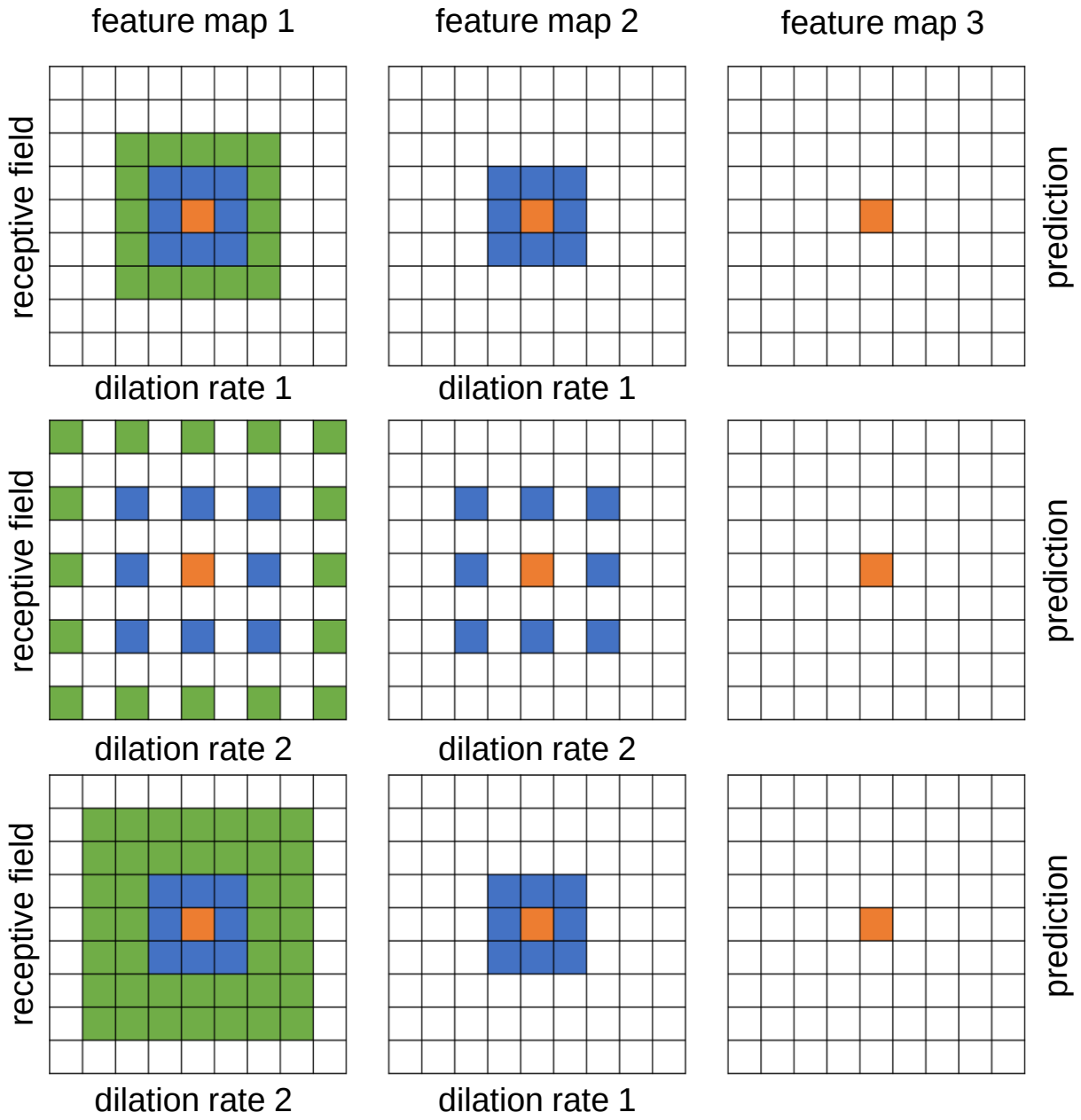
padding, non-unit strides

Transpose convolutions can be used for ‘learned’ up-sampling

Source: https://github.com/vdumoulin/conv_arithmetic

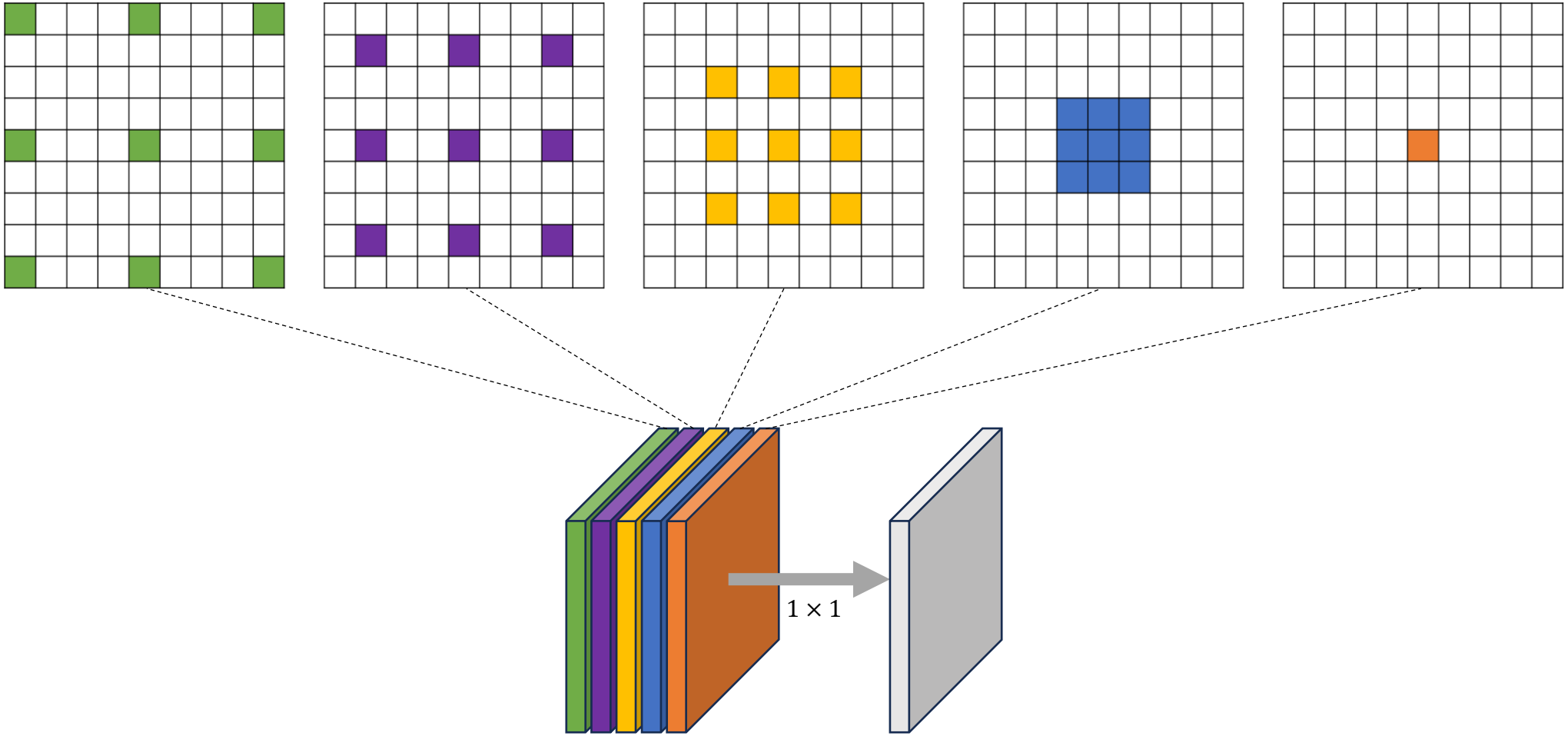
Dilated convolutions





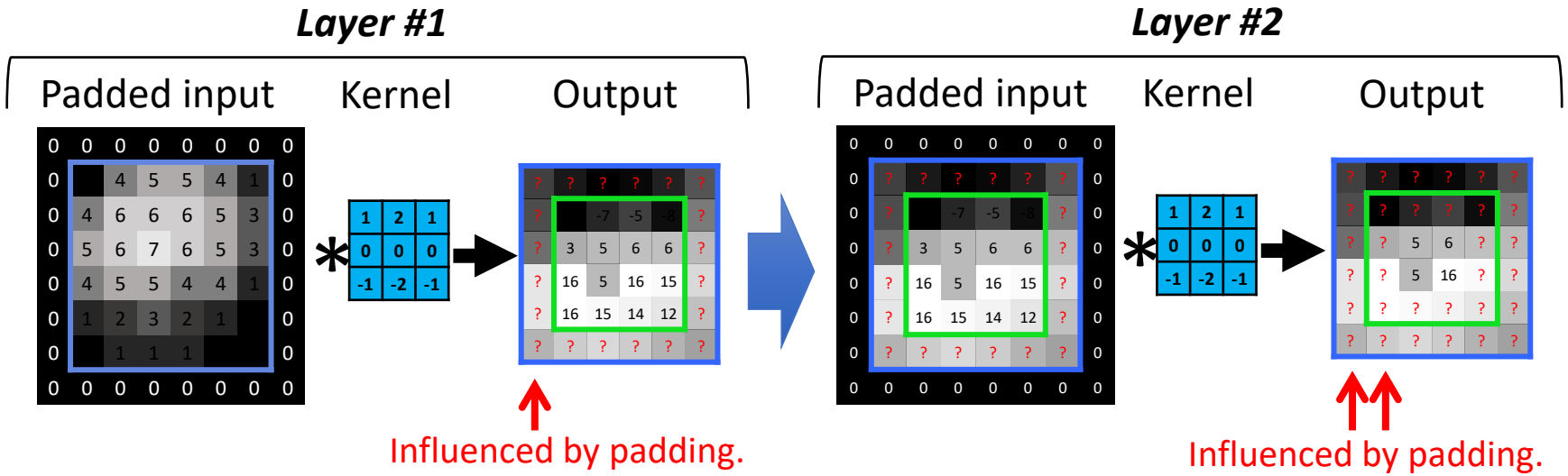
Yu and Koltun 2015. [Multi-Scale Context Aggregation by Dilated Convolutions]

Atrous spatial pyramid pooling



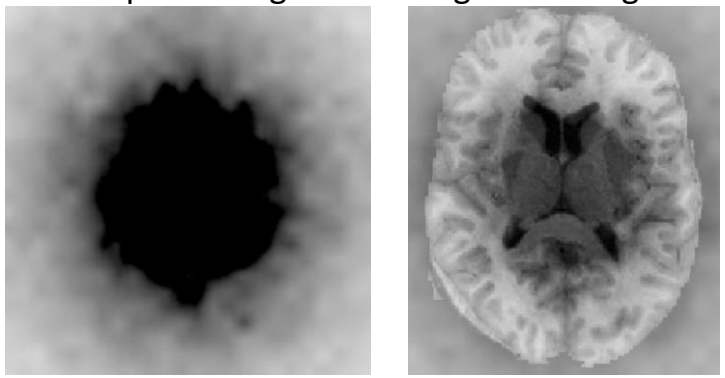
Chen et al. 2017. [DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs]

Padding effects

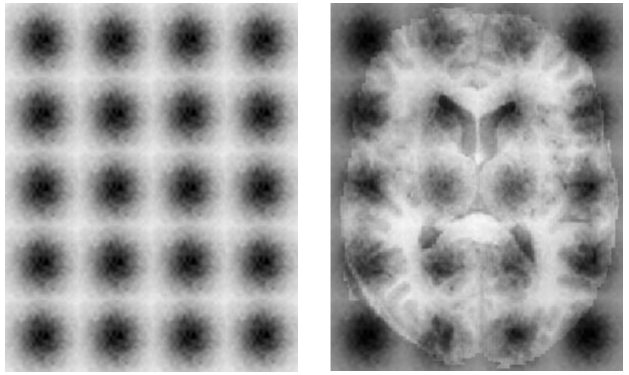


Artifacts that may show up due to padding (exaggerated for clarity):

If processing whole image at testing.

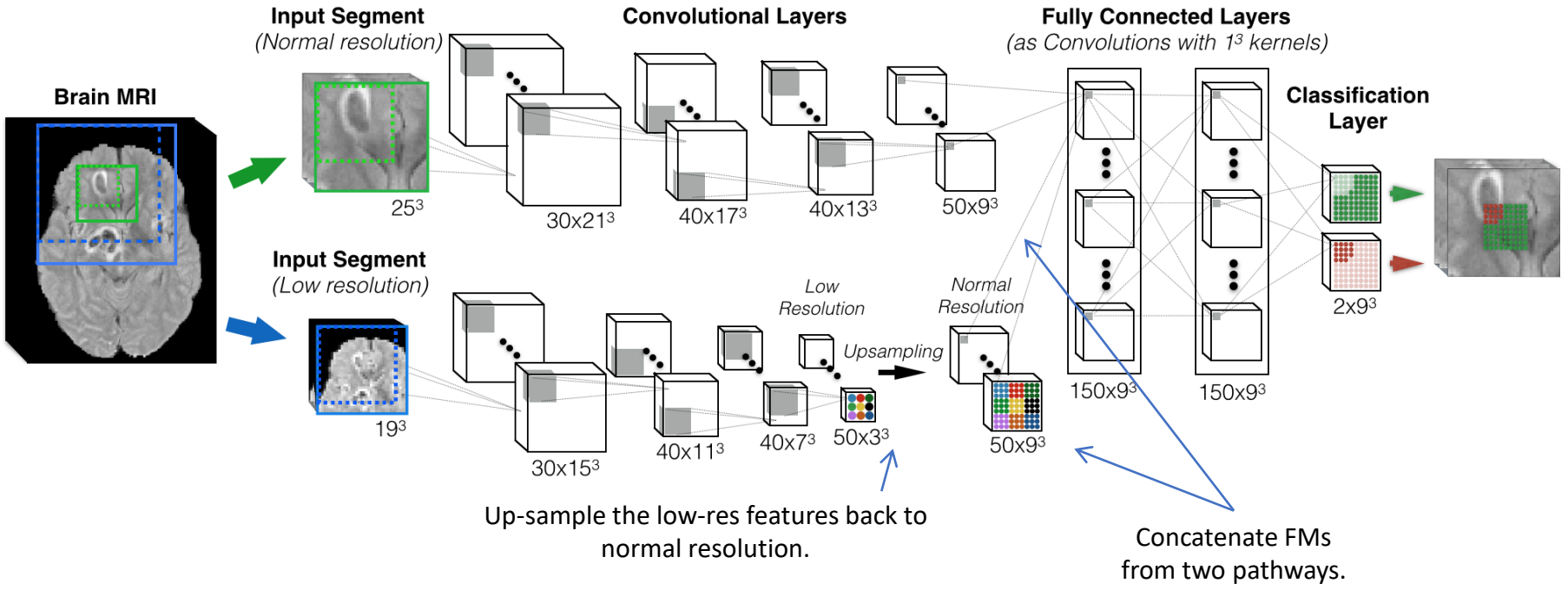


If tiling the image at testing.



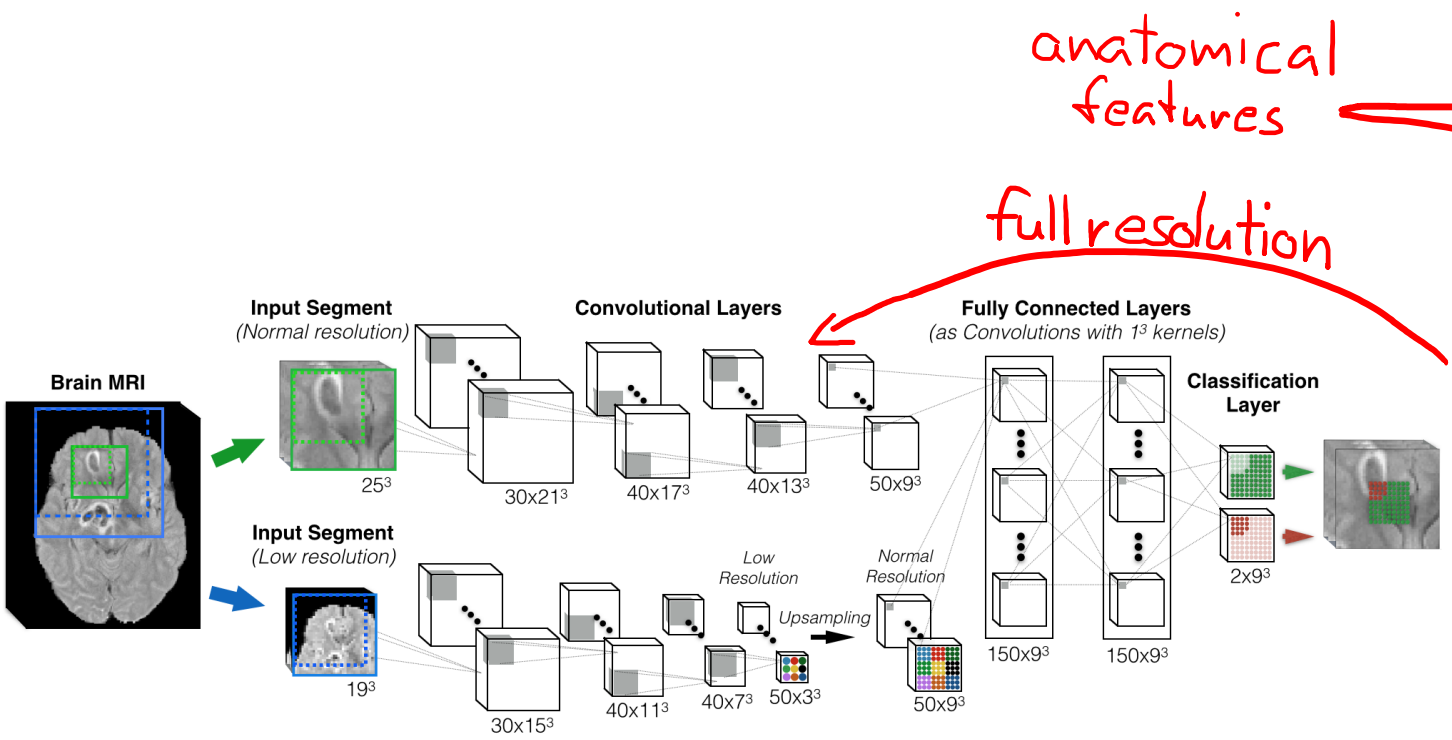
Multi-scale processing

- What else can we do to make the network “seeing” more context?
- **Idea:** Add pathways which process down-sampled images



Kamnitsas et al. 2017. Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation

Multi-scale feature maps

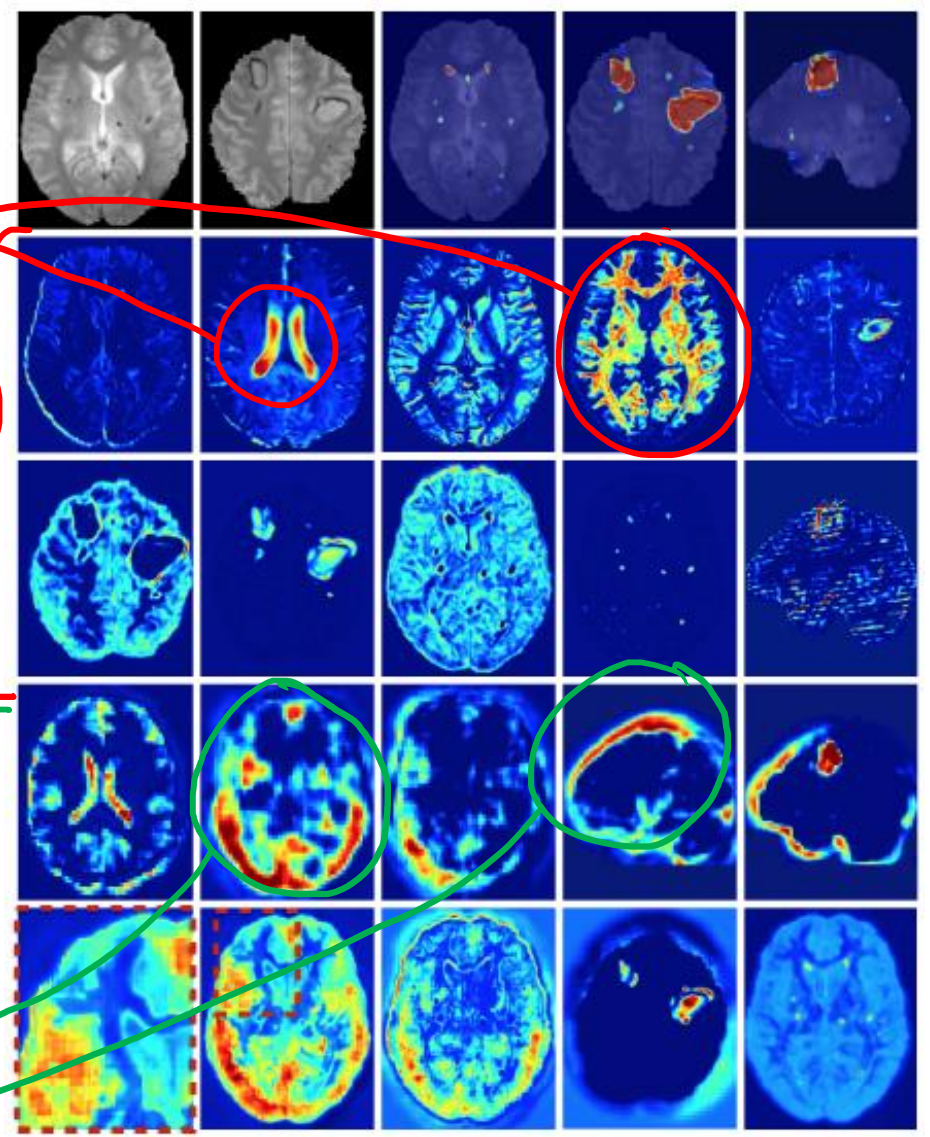


anatomical features

full resolution

low resolution

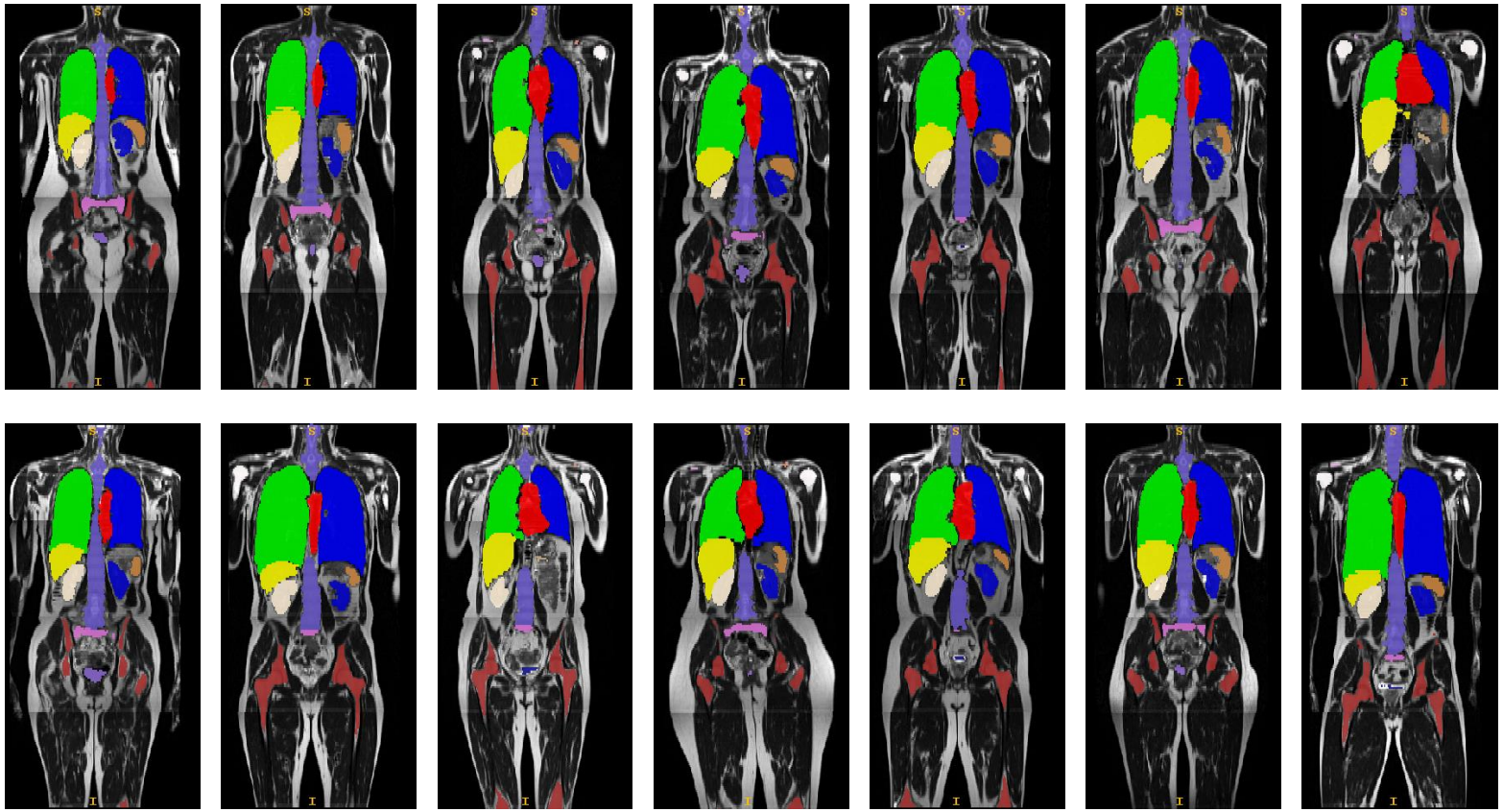
location features



Kamnitsas et al. 2017. [Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation]

Example: Multi-organ segmentation

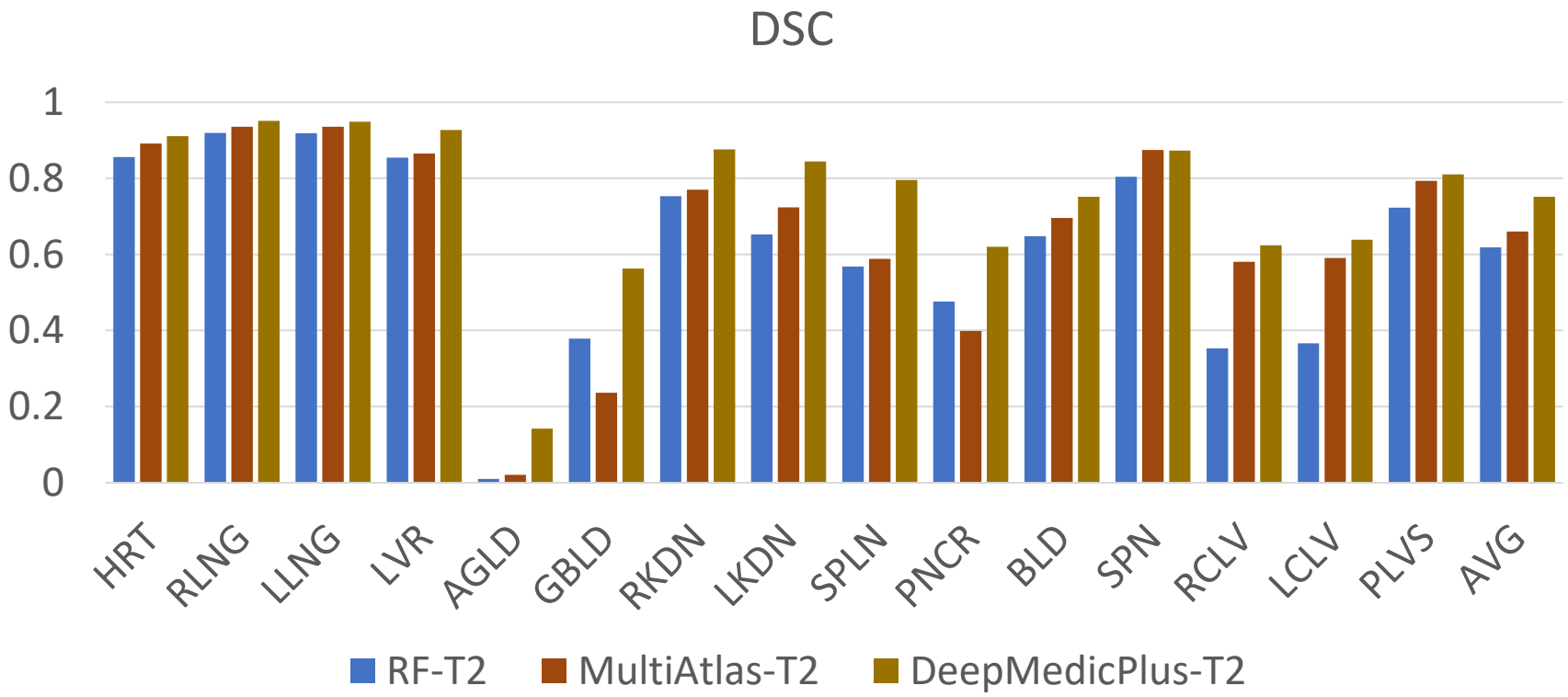
- Multi-atlas vs random forests vs CNNs

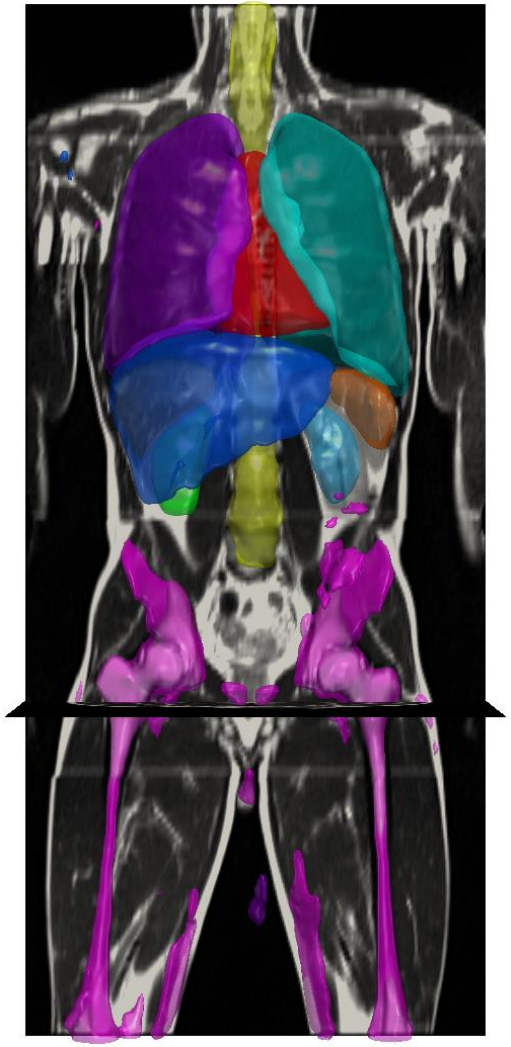


Lavdas et al. 2017. Fully automatic, multiorgan segmentation in normal whole body magnetic resonance imaging

Example: Multi-organ segmentation

- Multi-atlas vs random forests vs CNNs

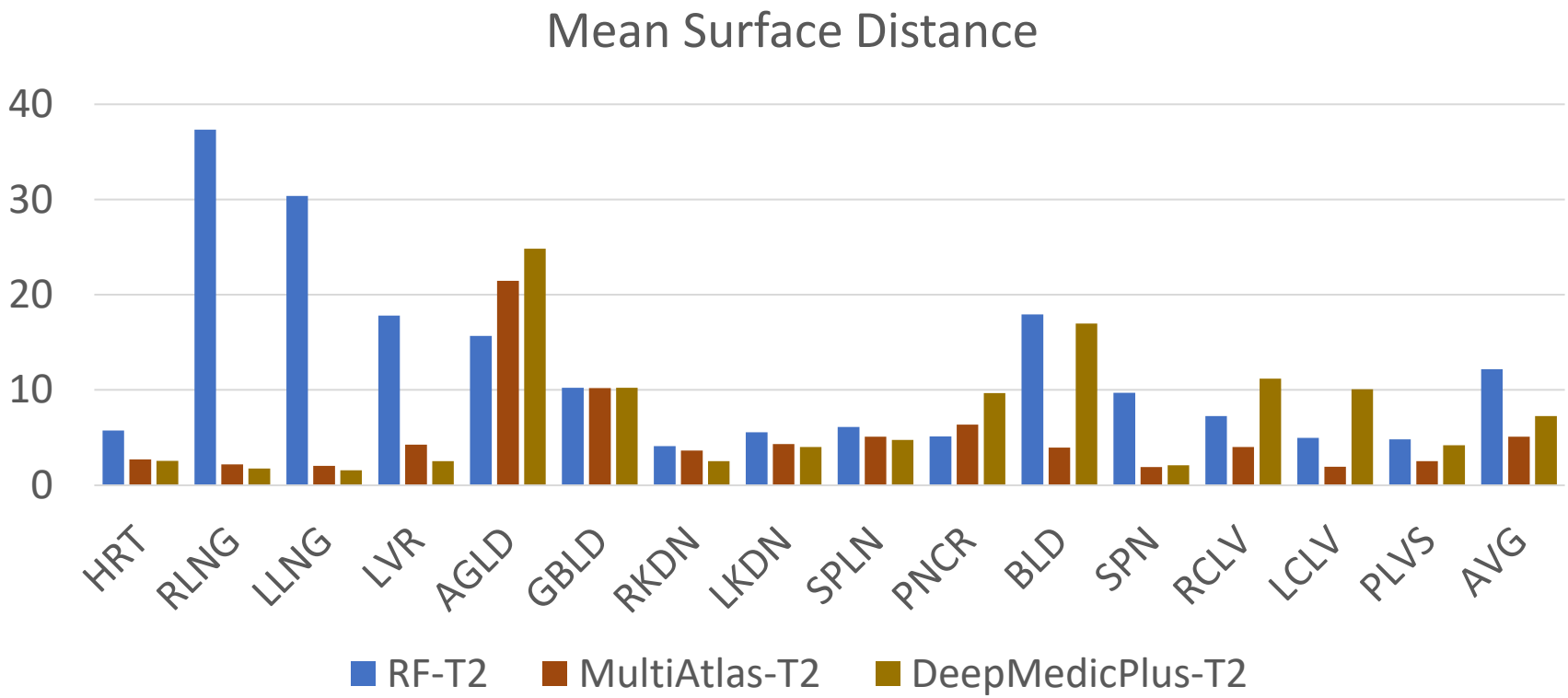


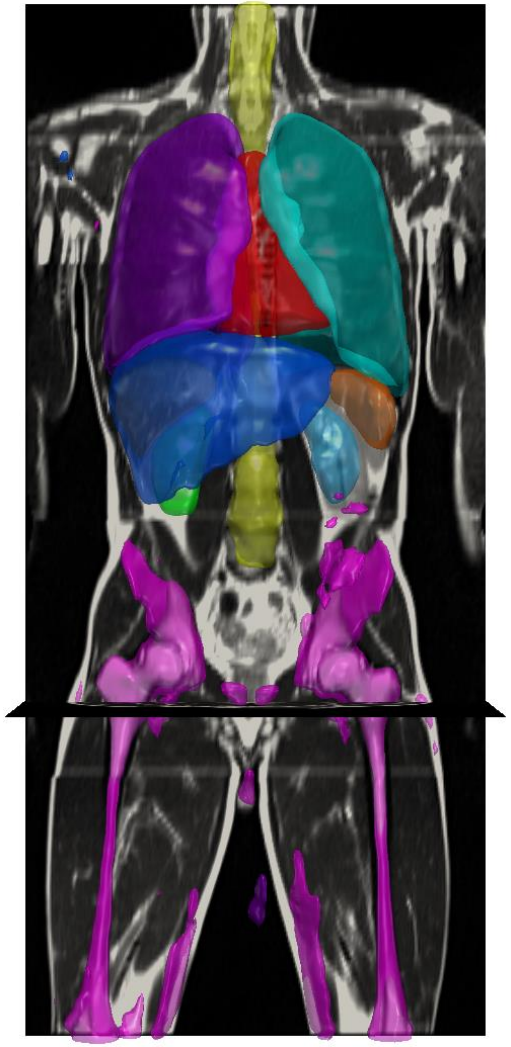


Lavdas et al. 2017. Fully automatic, multiorgan segmentation in normal whole body magnetic resonance imaging

Example: Multi-organ segmentation

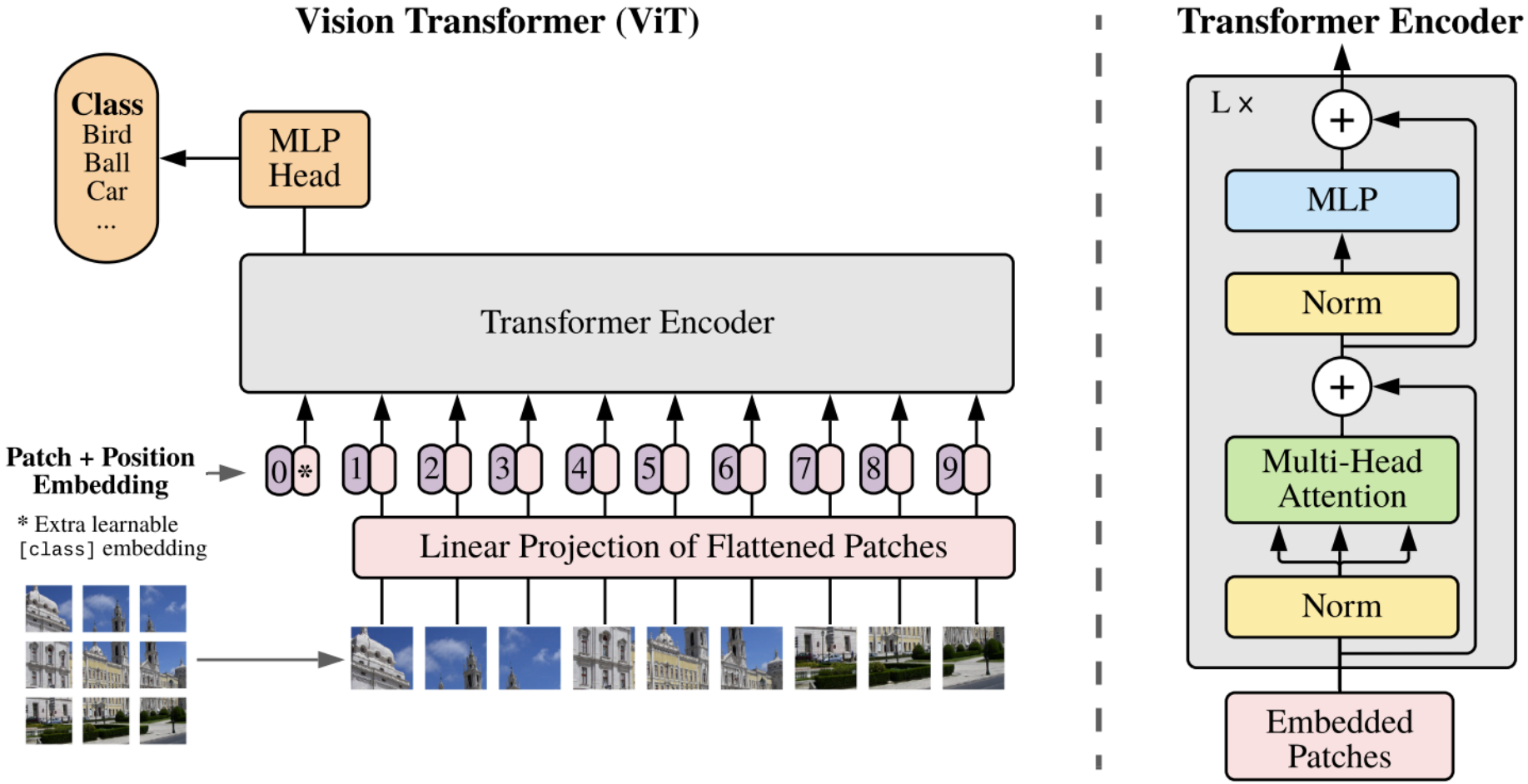
- Multi-atlas vs random forests vs CNNs





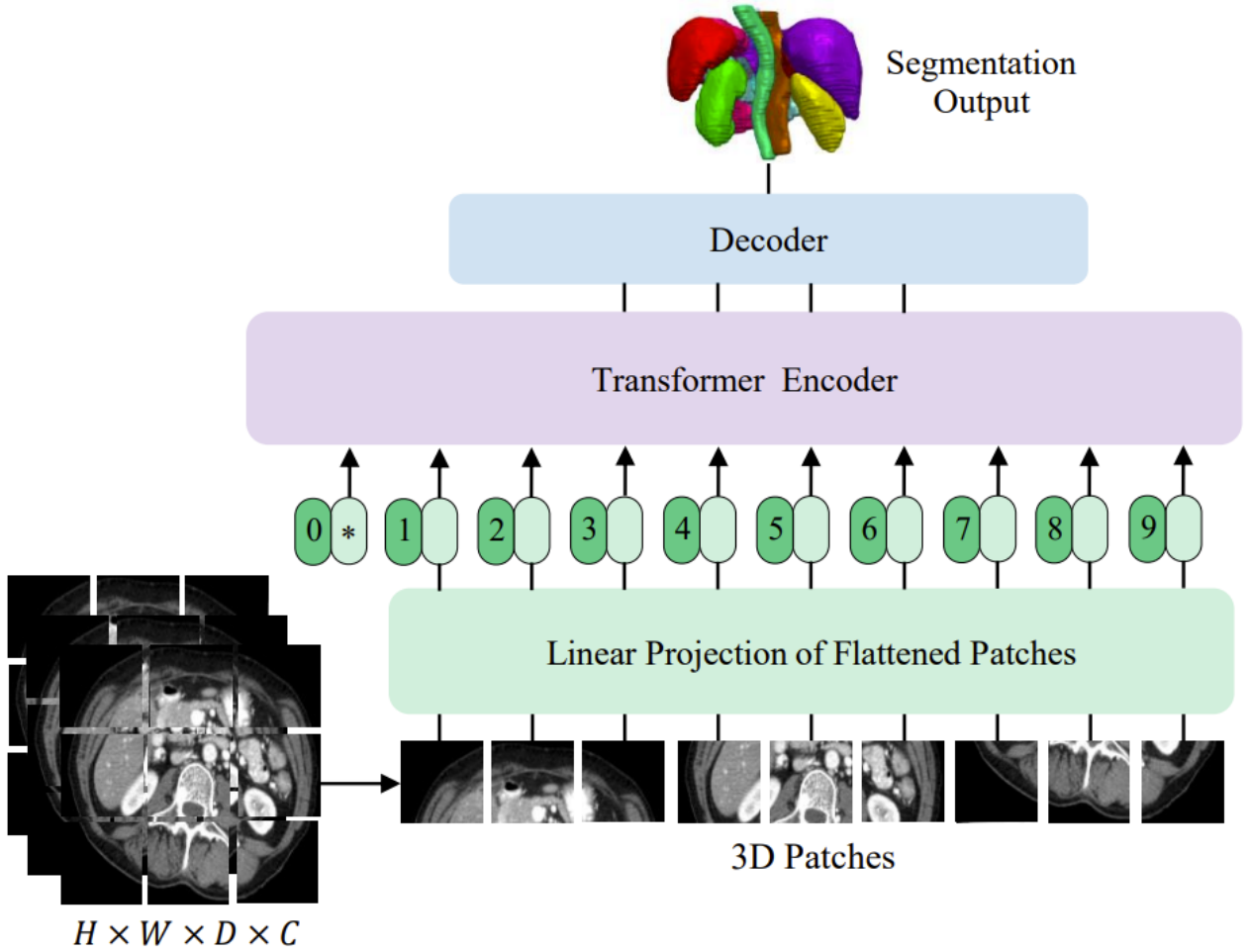
Lavdas et al. 2017. Fully automatic, multiorgan segmentation in normal whole body magnetic resonance imaging

Vision transformers



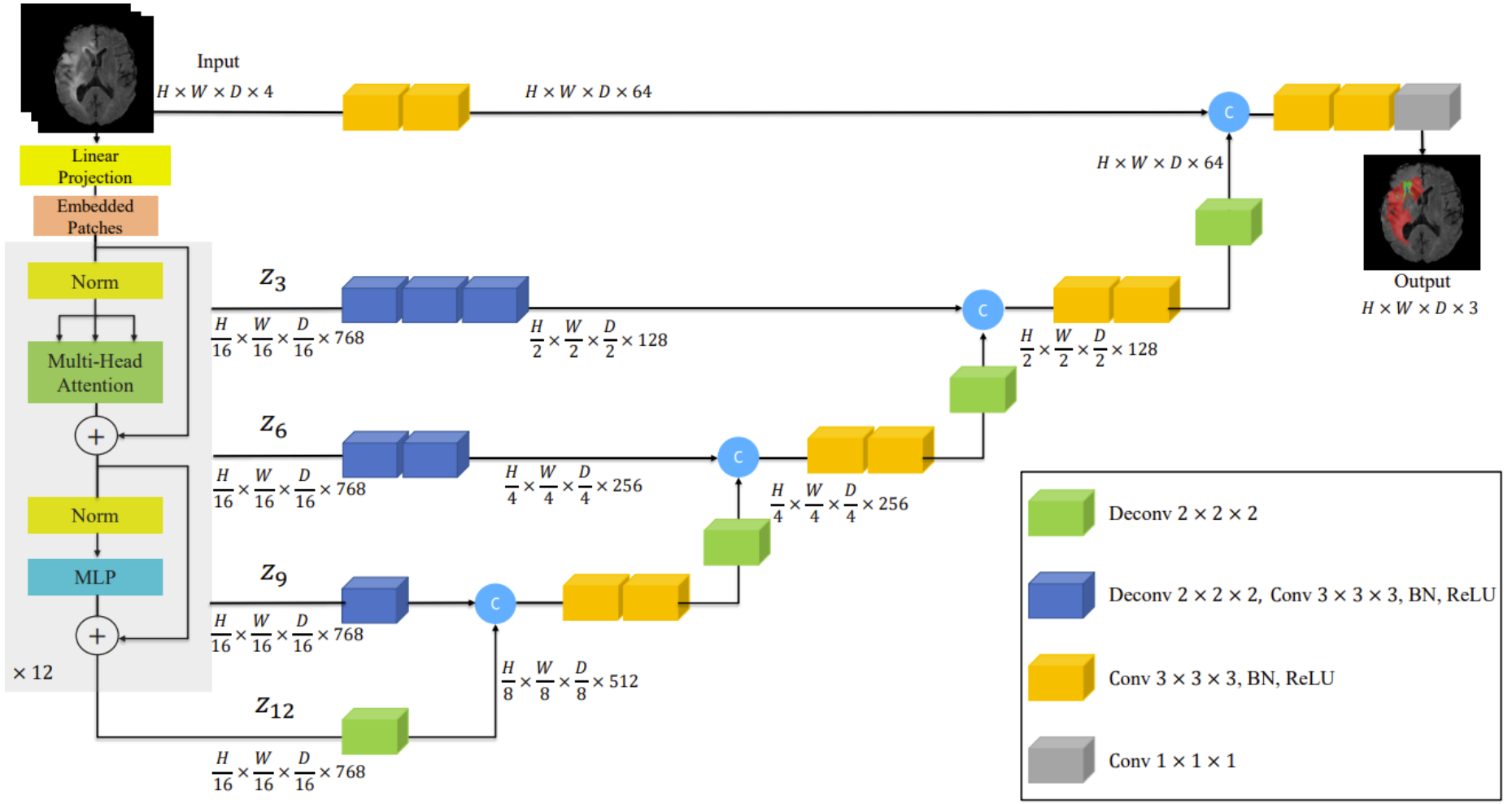
Dosovitskiy et al. 2021. [An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale]

Transformers for image segmentation



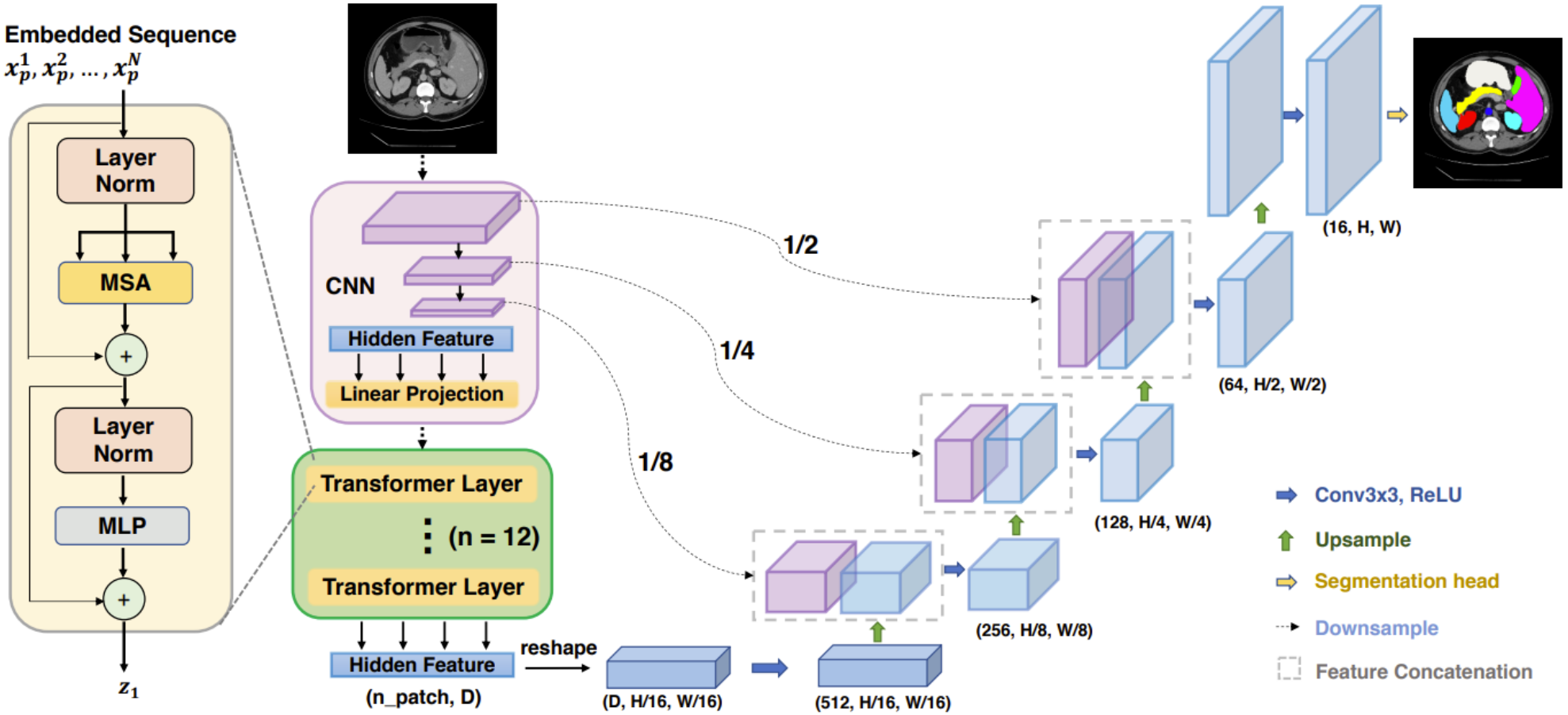
Hatamizadeh et al. 2021. [UNETR: Transformers for 3D Medical Image Segmentation]

Transformers for image segmentation



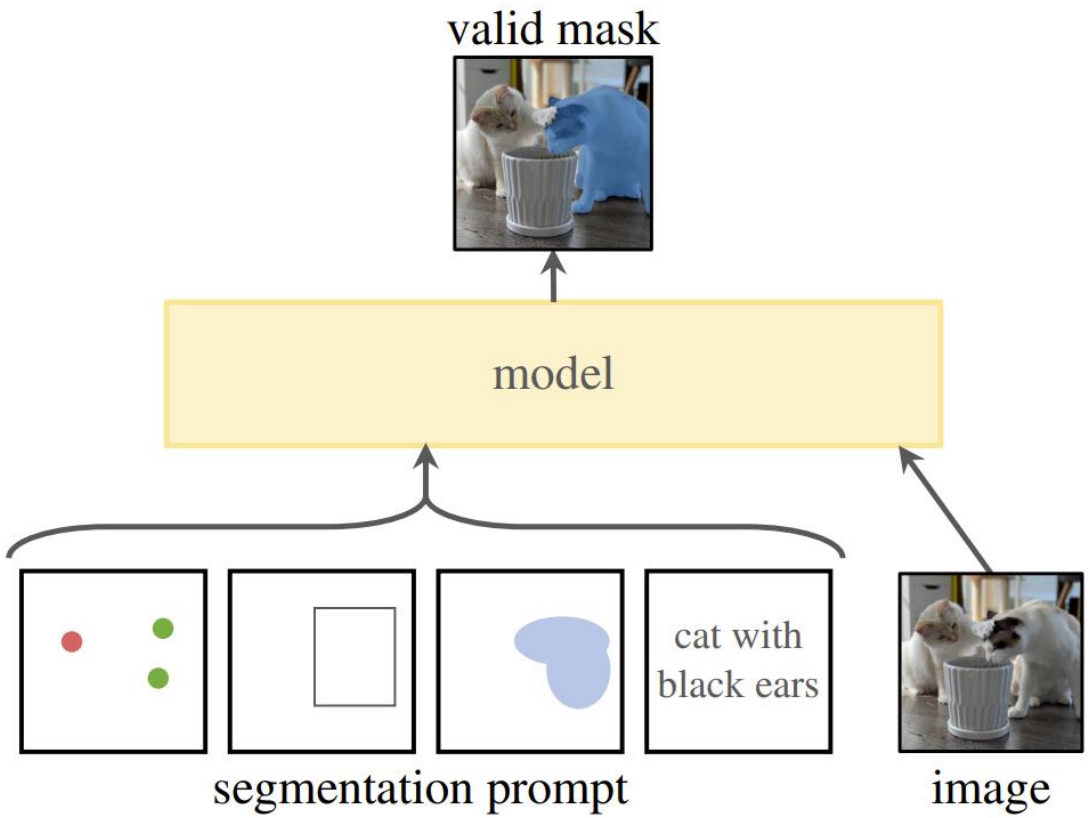
Hatamizadeh et al. 2021. [UNETR: Transformers for 3D Medical Image Segmentation]

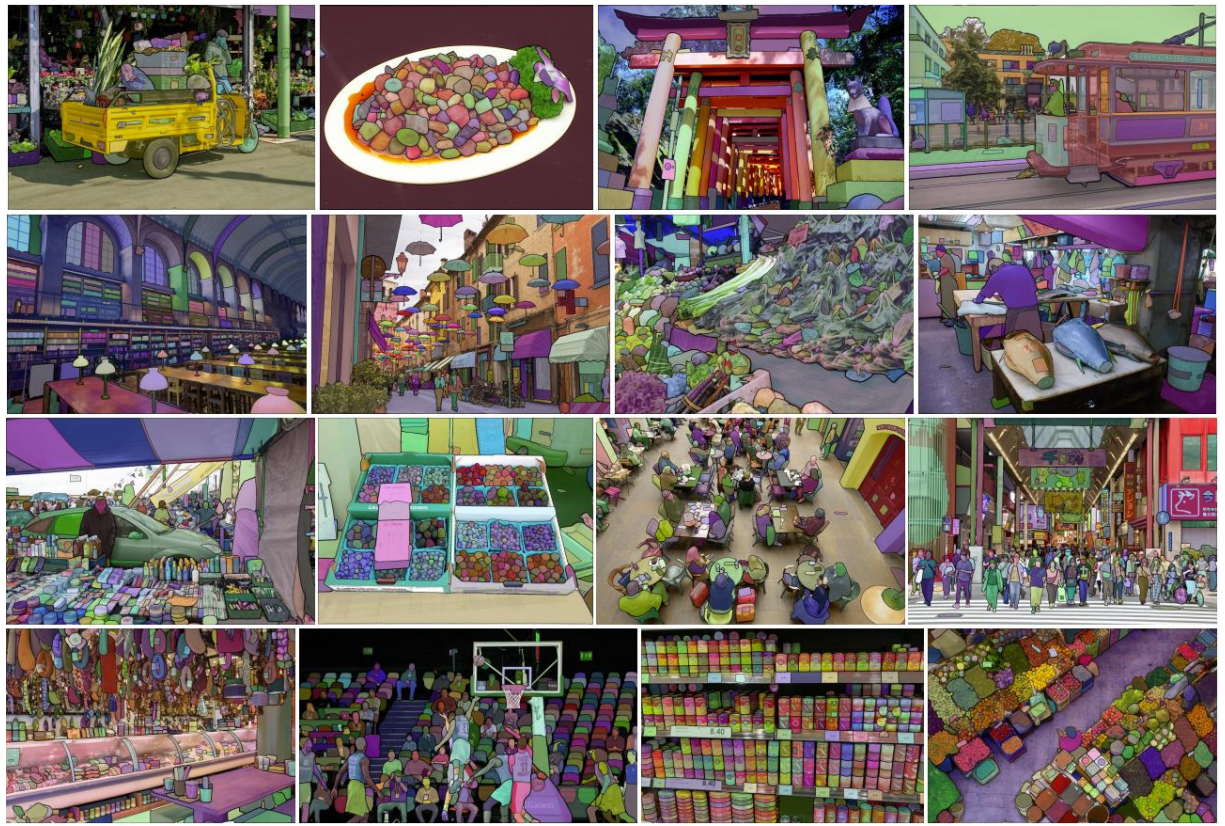
Transformers for image segmentation



Chen et al. 2021. [TransUNet: Transformers Make Strong Encoders for Medical Image Segmentation]

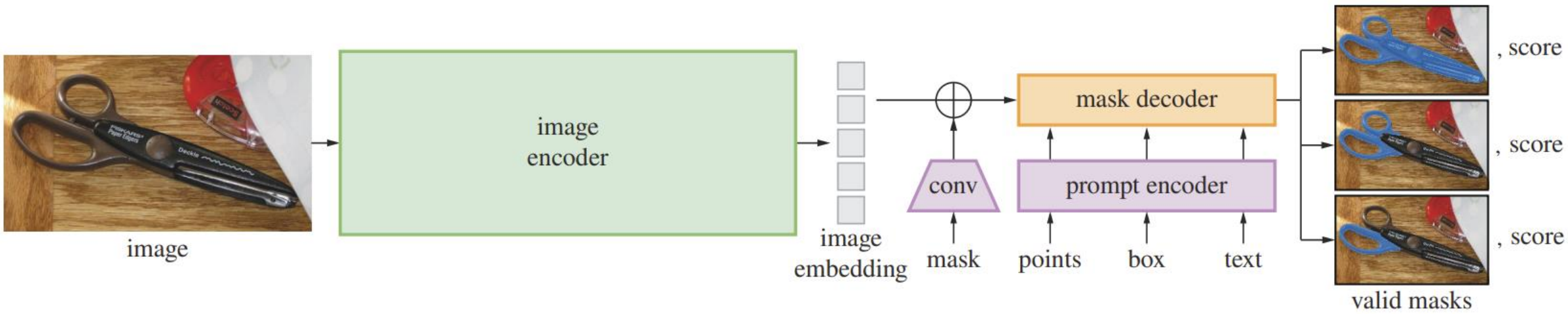
Transformers for image segmentation





Kirillov et al. 2023. Segment Anything

Segment anything model



Kirillov et al. 2023. Segment Anything

Tutorial 2 - Segmentation