THE UNIVERSITY OF MELBOURNE

Image Segmentation I

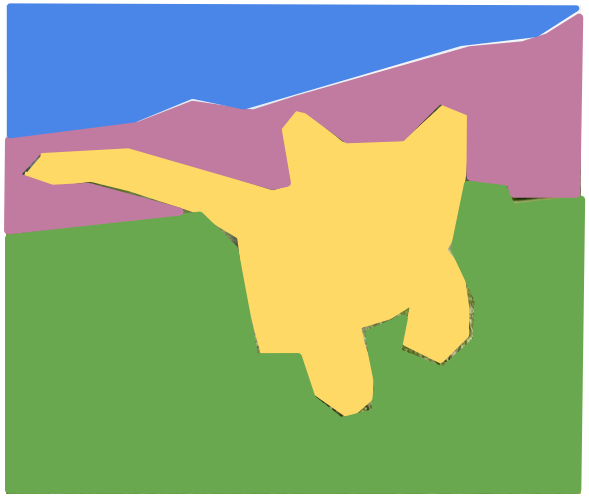
Semester 2, 2021

Kris Ehinger

Demo

- https://chocopoule.github.io/grabcutweb/

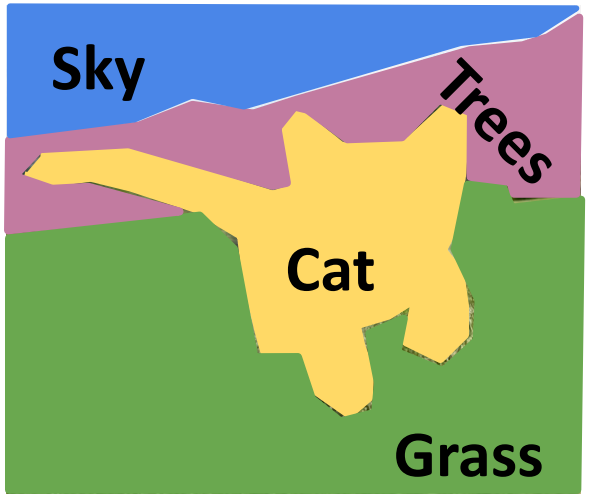
Segmentation

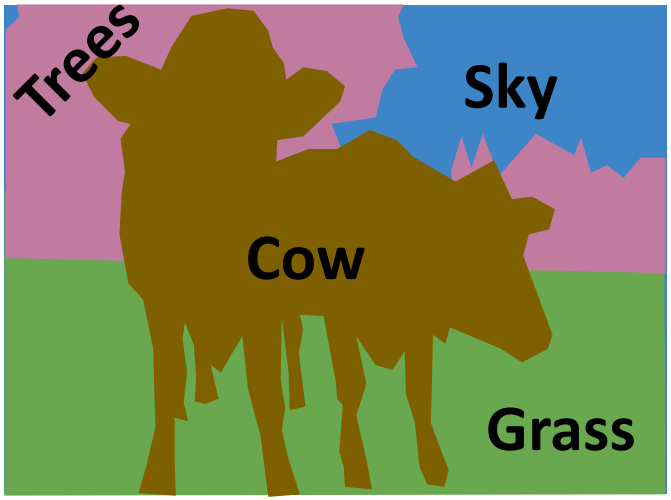


Separate image into different regions (objects, textures)

Image: J. Johnson

Semantic segmentation





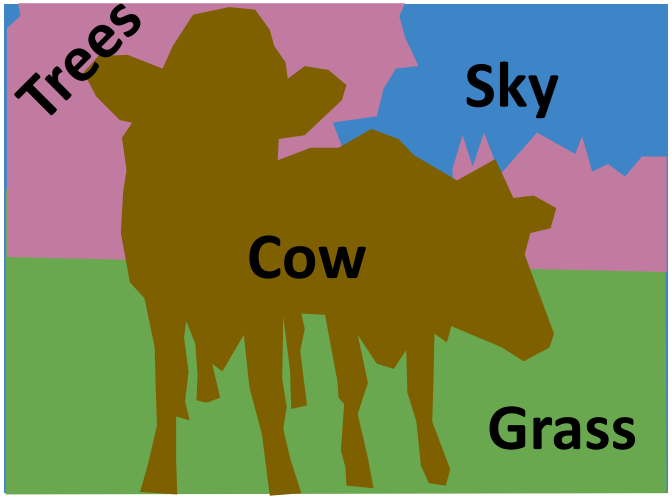
Separate image into different *labelled* regions

Image: J. Johnson

Instance segmentation

Semantic segmentation

Instance segmentation



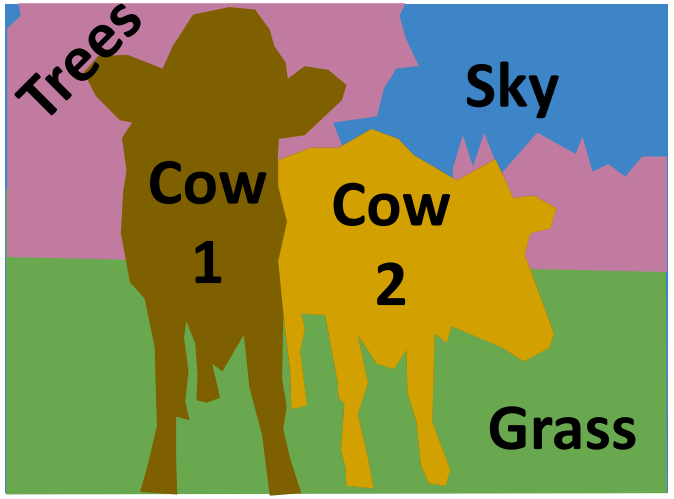
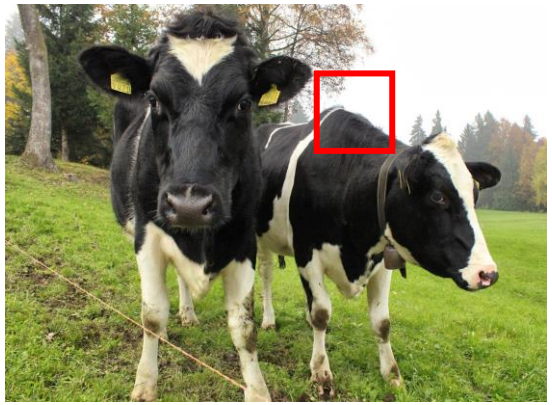
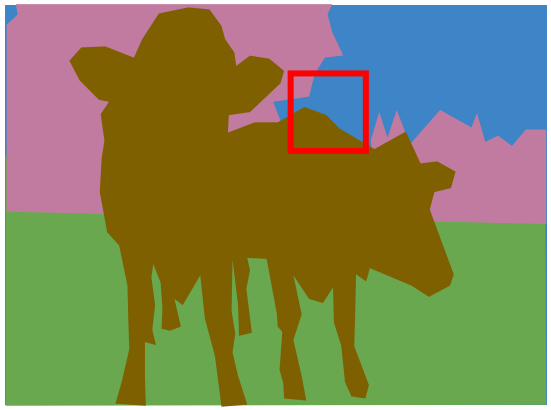


Image segmentation



Input: Image

Clustering?
Graph cuts?
Classification?



Output: Pixel classification
(and, optionally, labels)

Outline

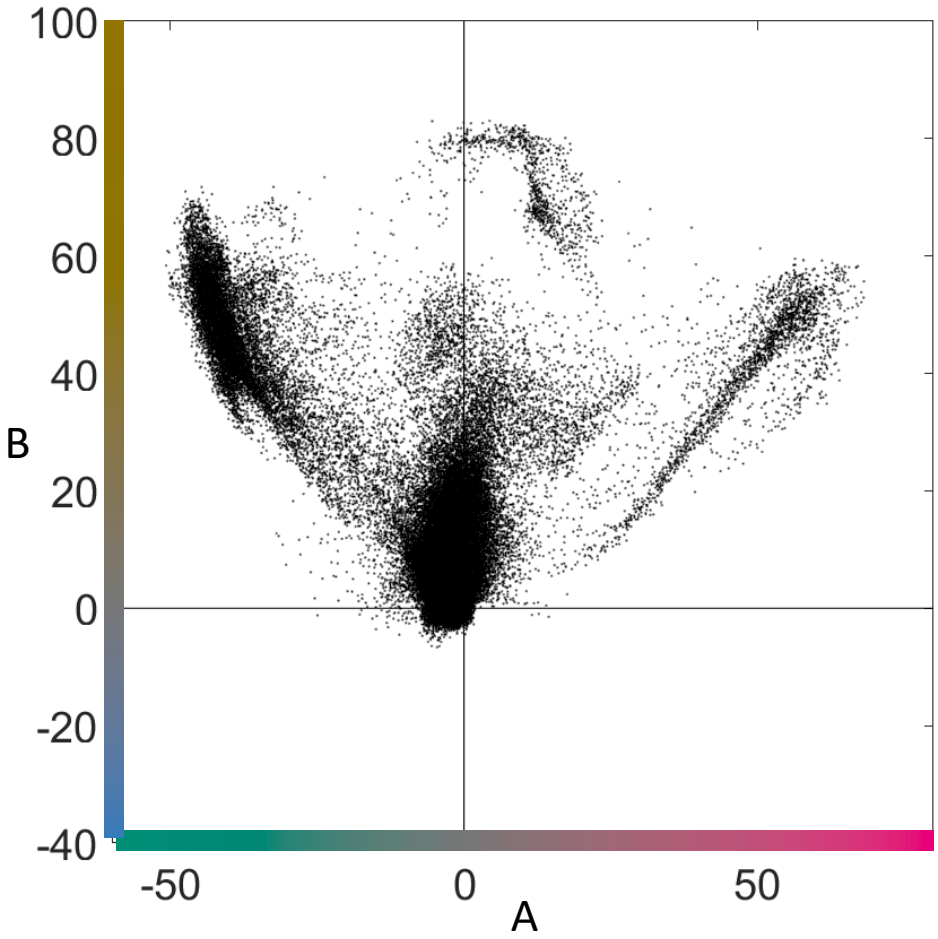
- Pixel clustering
- Superpixel segmentation
- Graph-based segmentation

Learning objectives

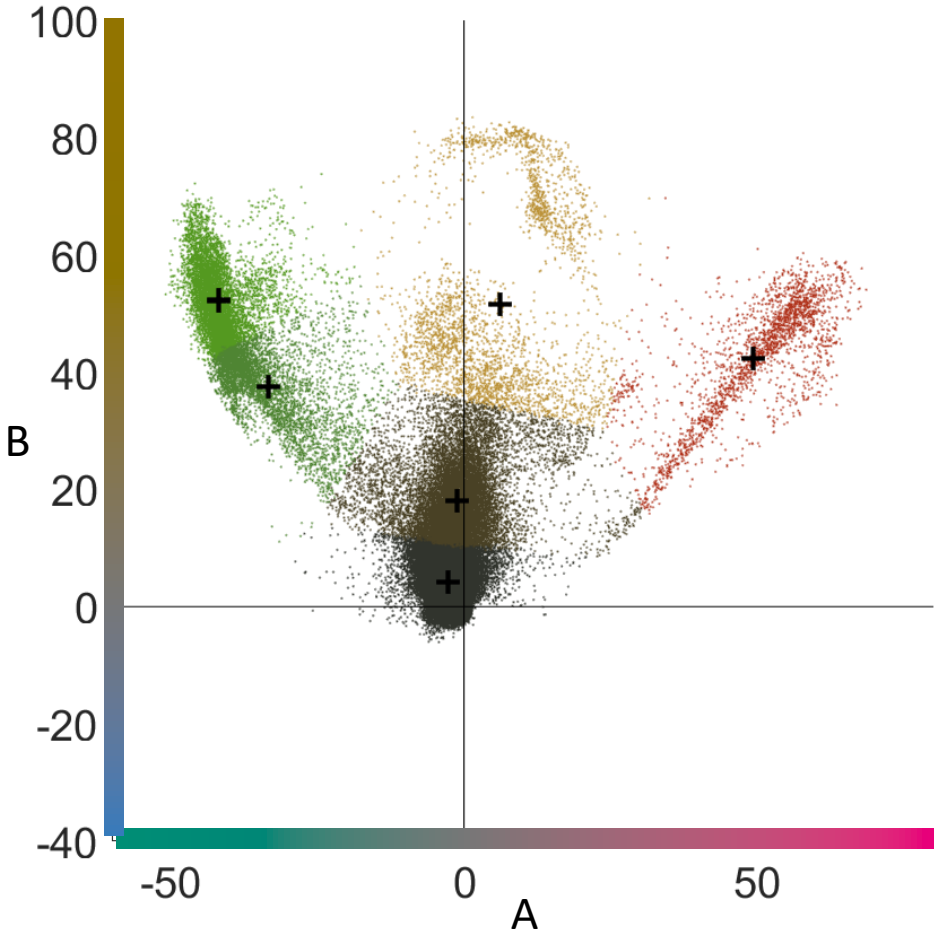
- Implement clustering algorithms for segmentation and compare/contrast clustering methods
- Implement an algorithm for computing superpixels and explain their common applications
- Explain graph-based methods for image segmentation

Pixel clustering

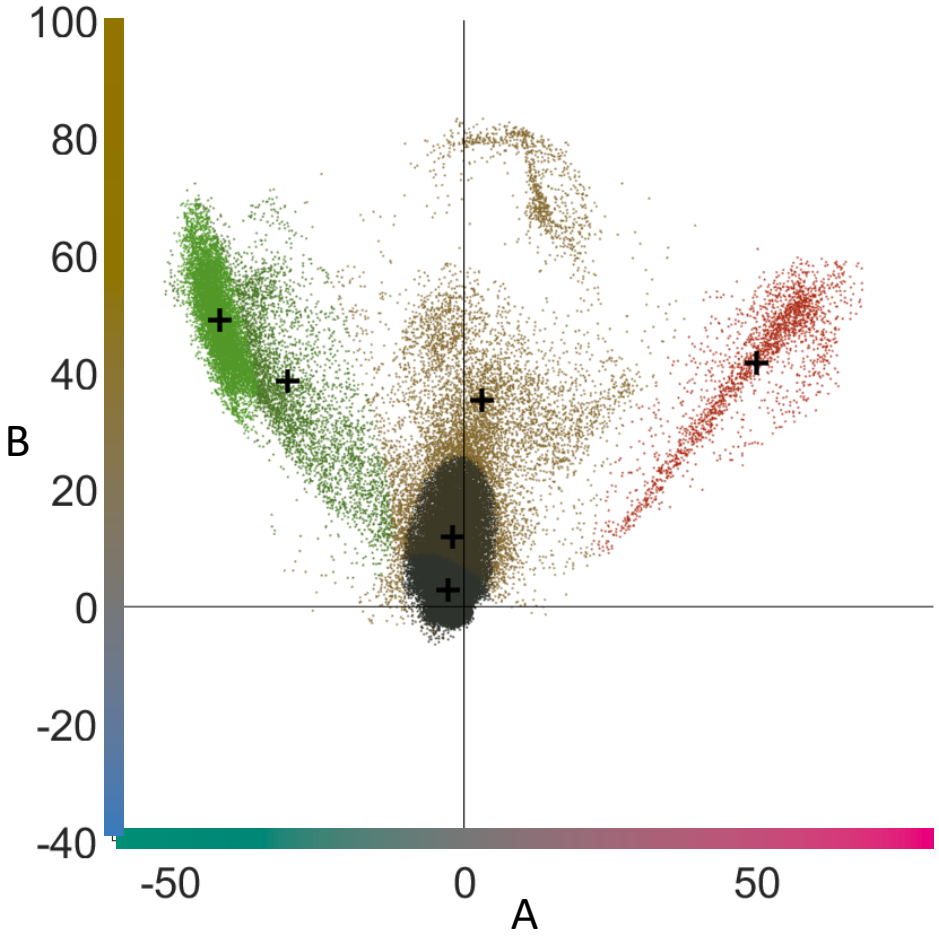
Colour clustering



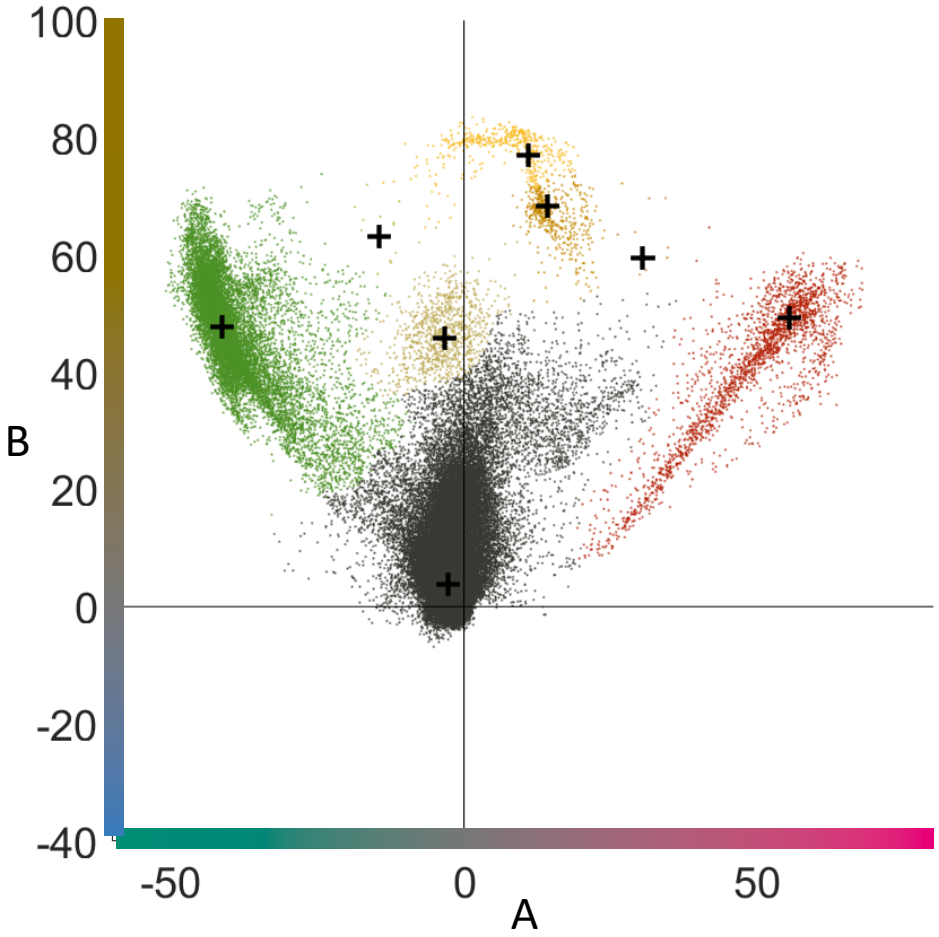
K-means (k = 6)



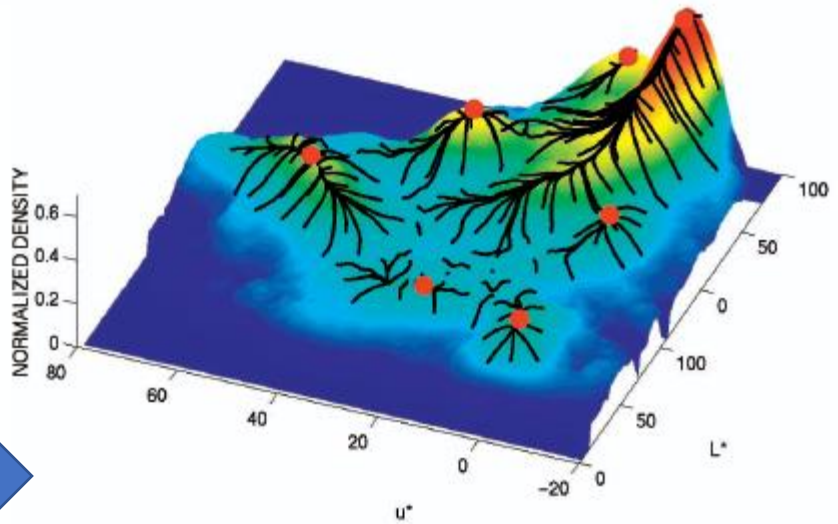
Gaussian mixture model (k = 6)

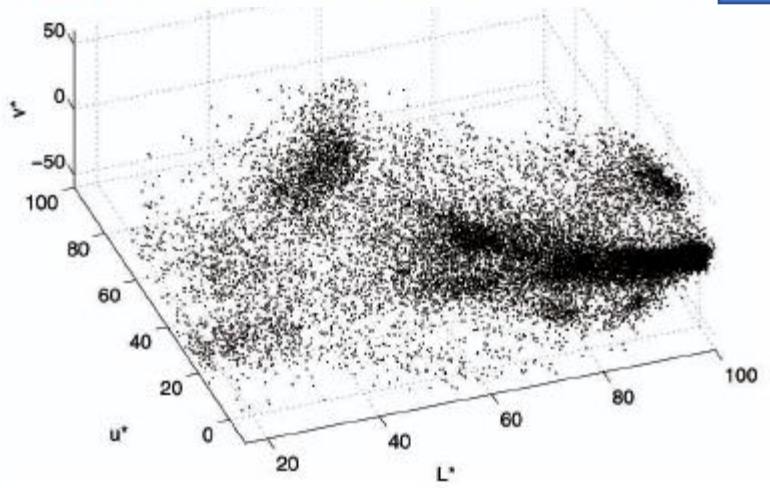


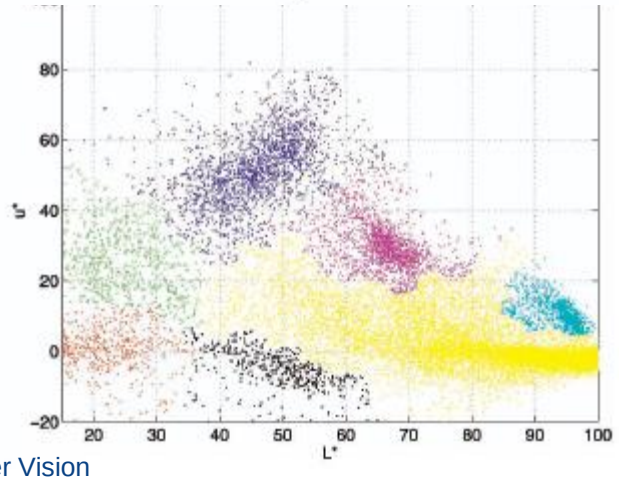
Mean shift (bandwidth = 7)



Mean shift clustering

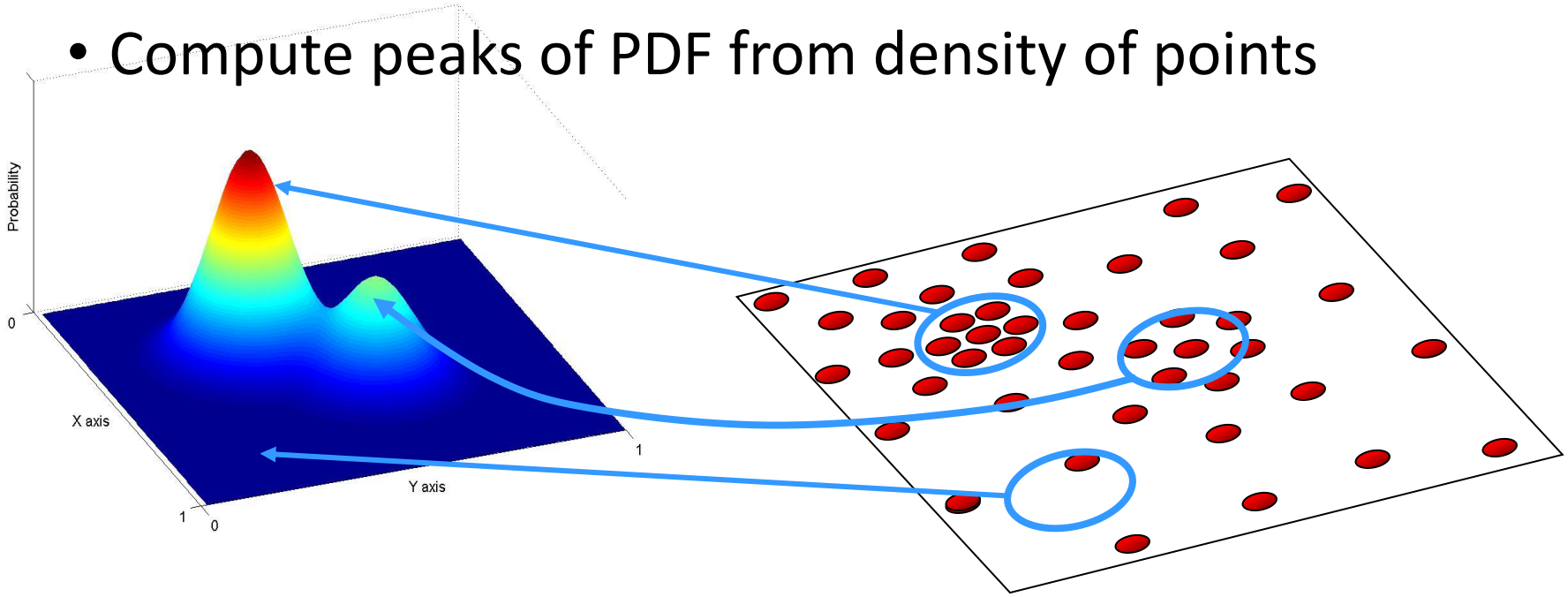






Mean shift clustering

- Assume points are samples from an underlying probability density function (PDF)
- Compute peaks of PDF from density of points



Assumed Underlying PDF

Real Data Samples

Image: Y. Ukrainitz & B. Sarel

Mean shift

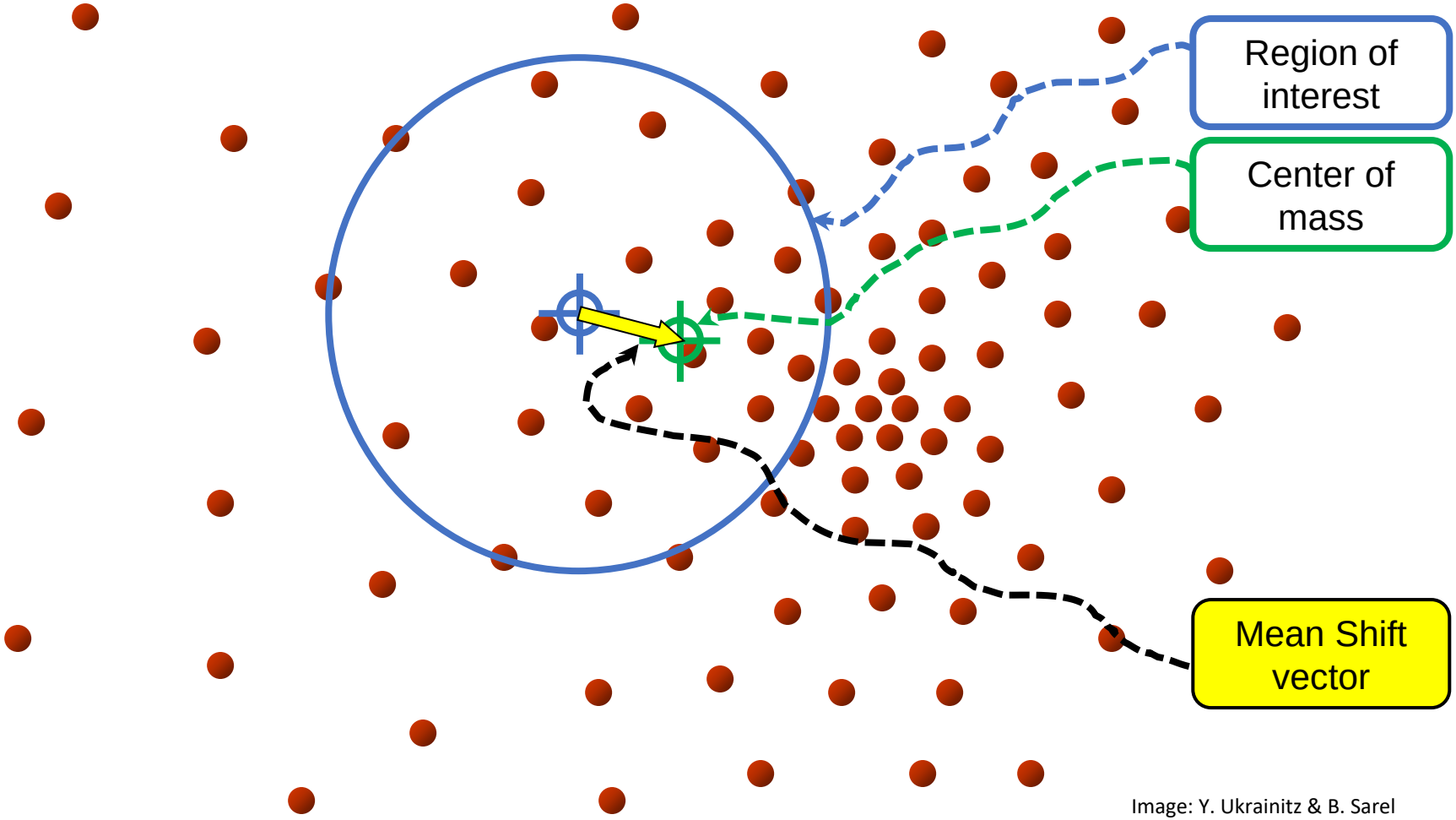
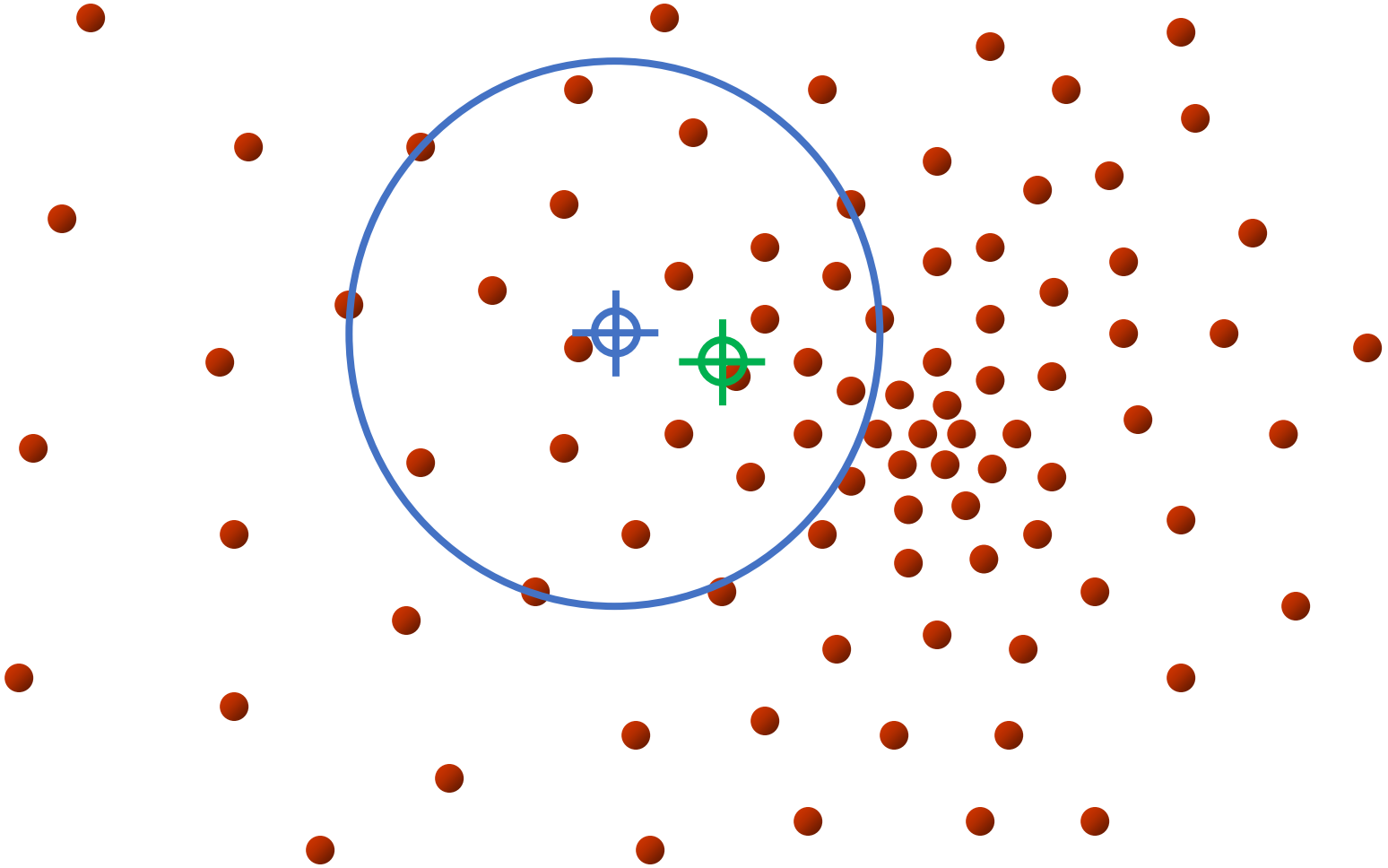
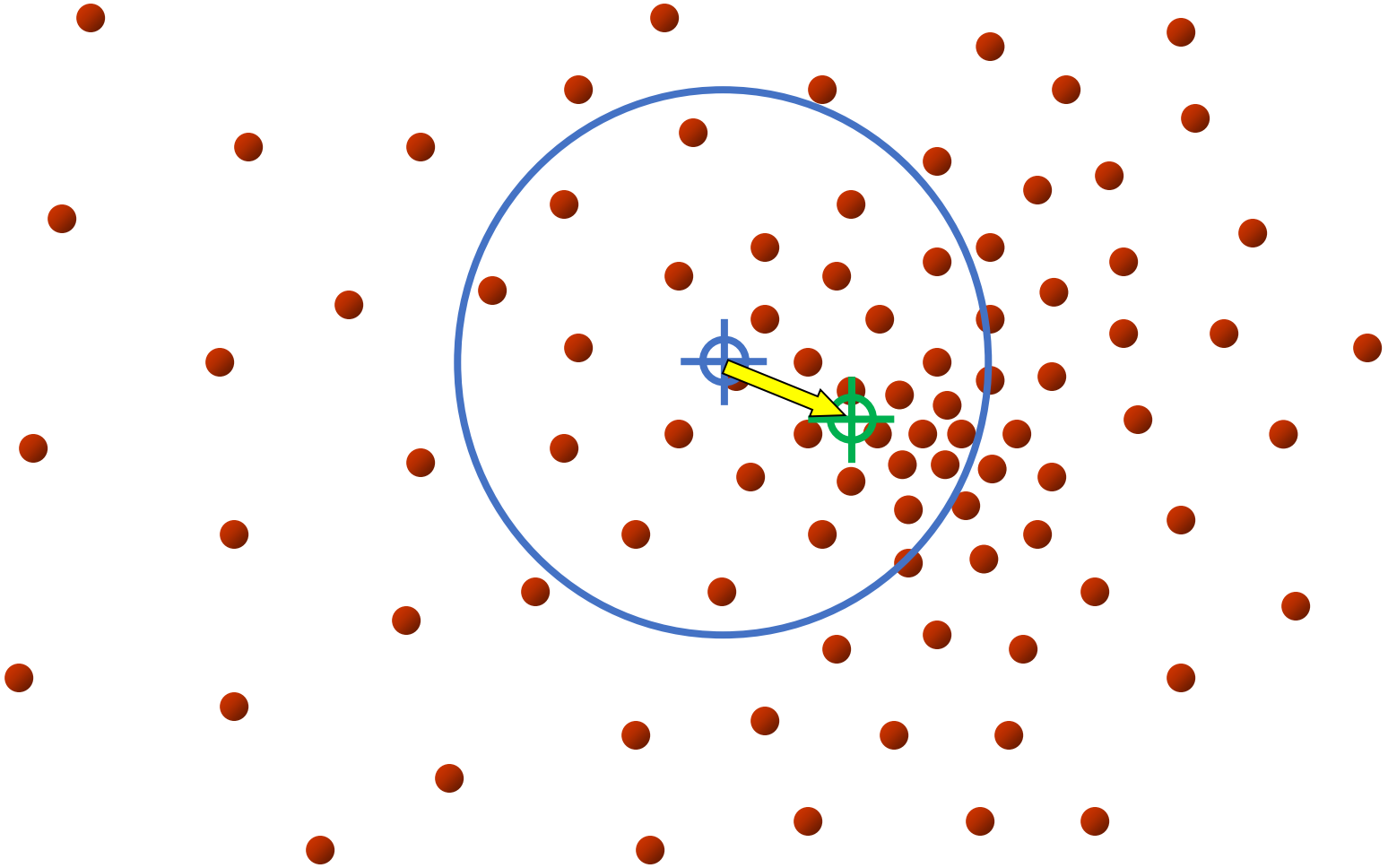


Image: Y. Ukrainitz & B. Sarel

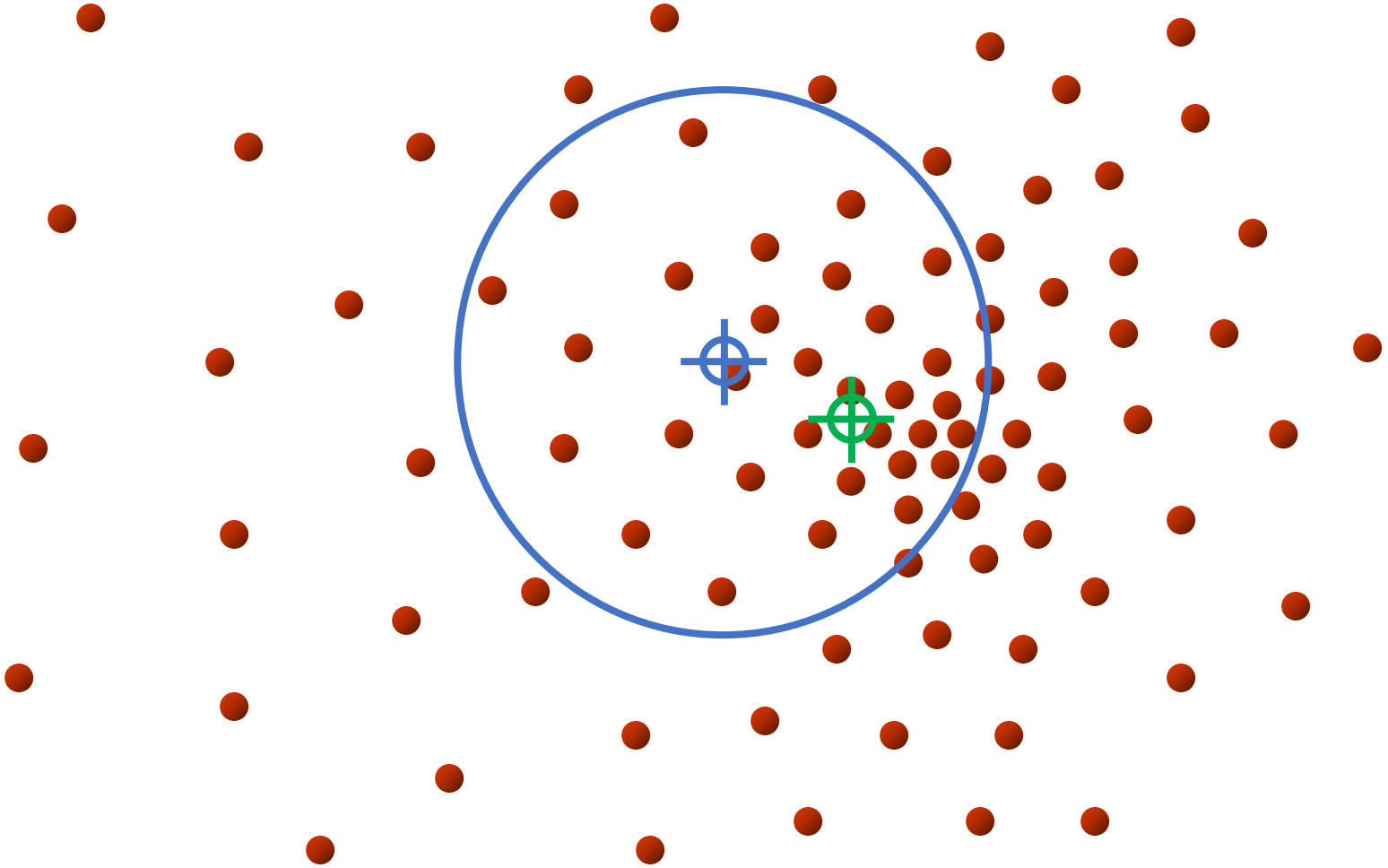
Mean shift



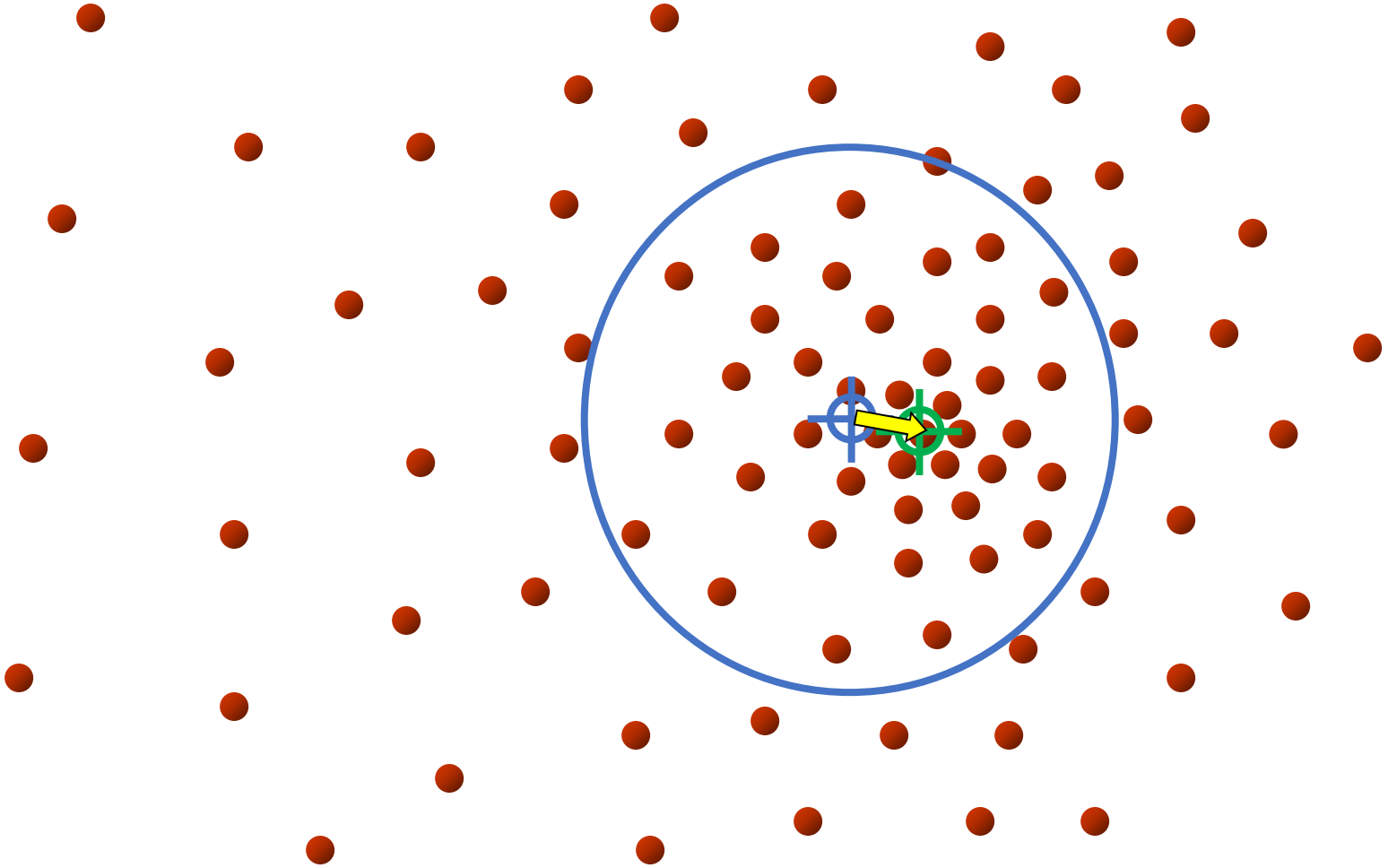
Mean shift



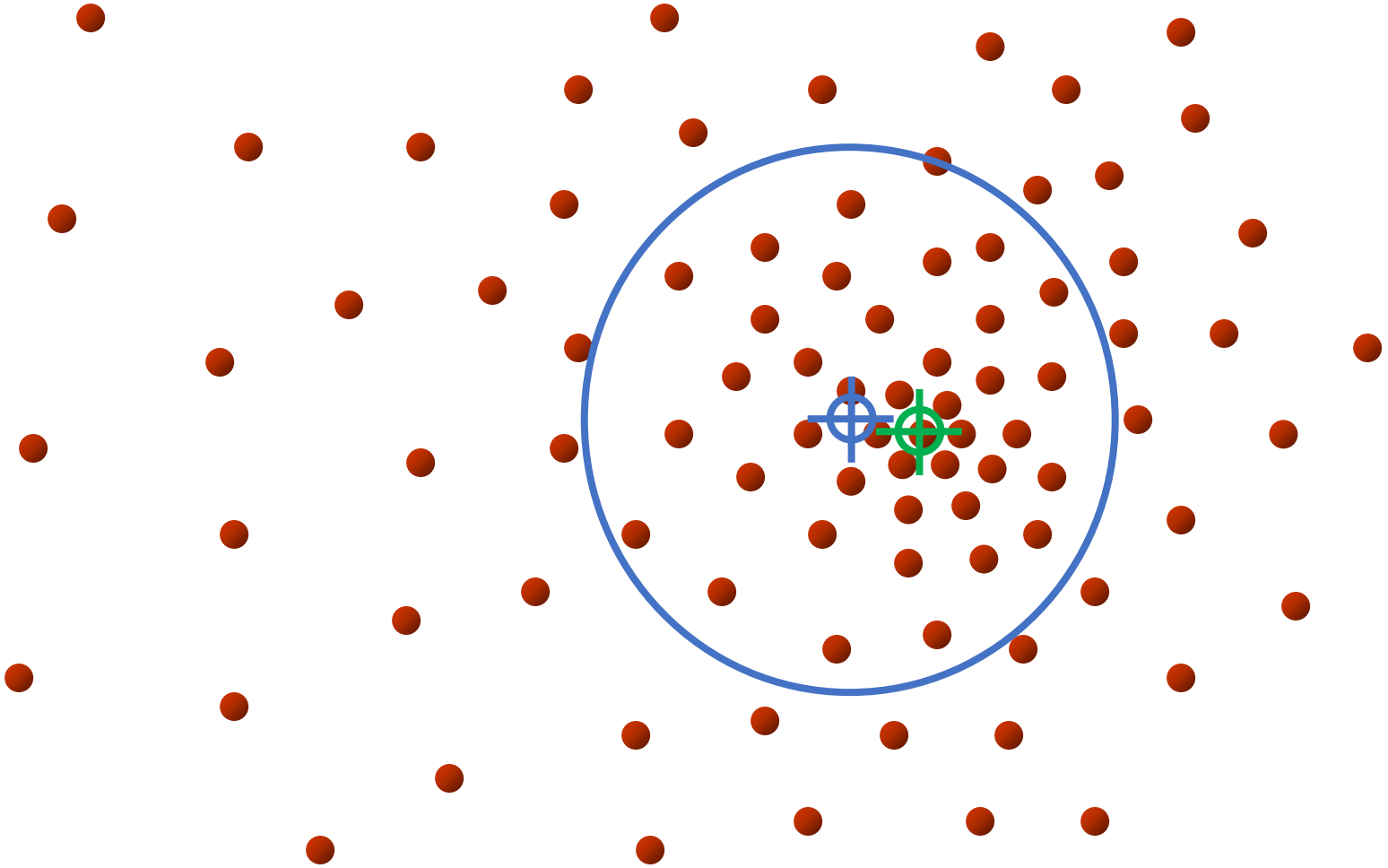
Mean shift



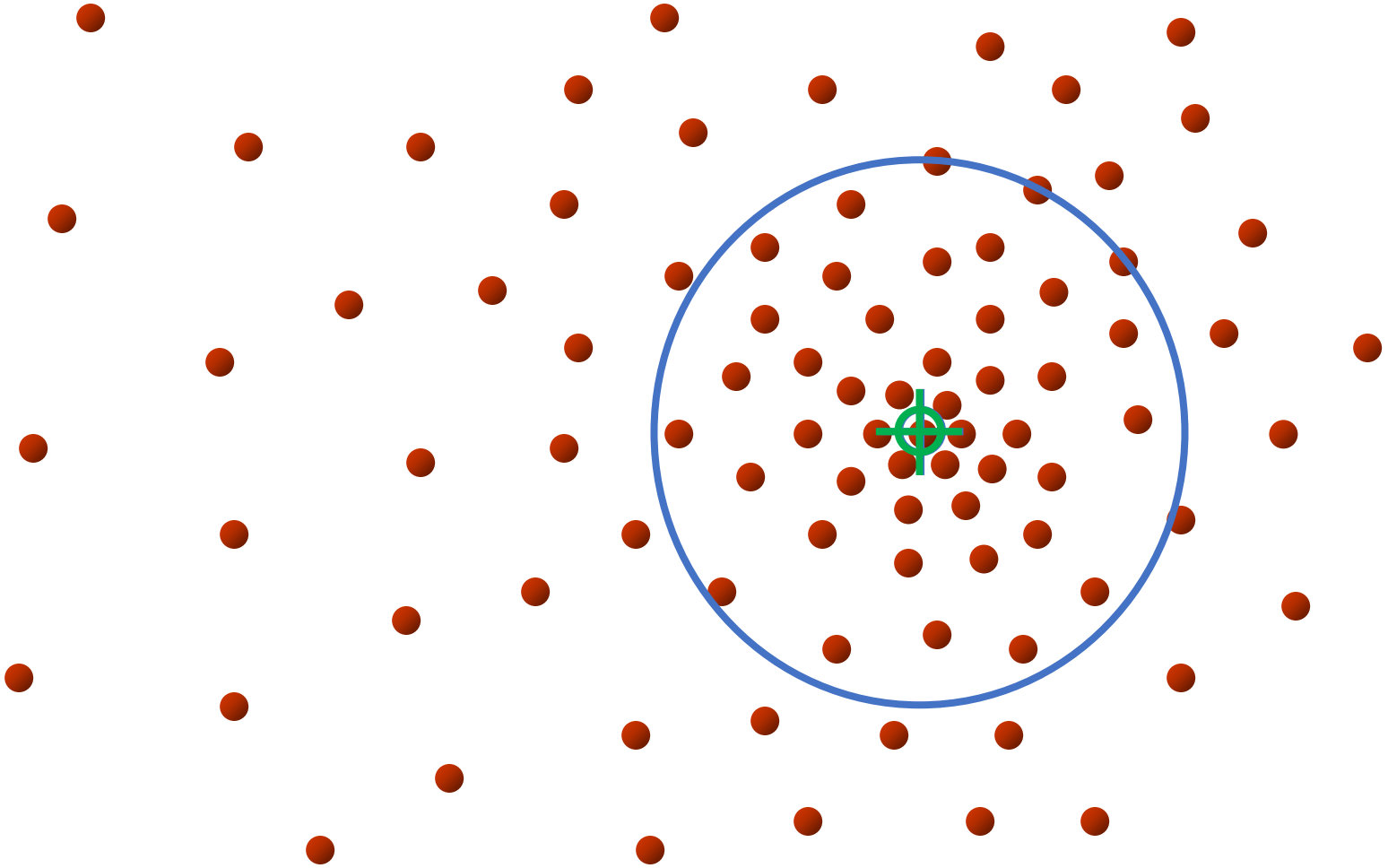
Mean shift



Mean shift



Mean shift



Mean shift algorithm

- Compute mean shift vector **m(x)**
- Translate kernel window by **m(x)**

Gaussian kernel:

$$g(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

$$\mathbf{m}(\mathbf{x}) = \left[\frac{\sum_{i=1}^{n} \mathbf{x}_i g\left(\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{h} \right)}{\sum_{i=1}^{n} g\left(\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{h} \right)} \right] - \mathbf{x}$$

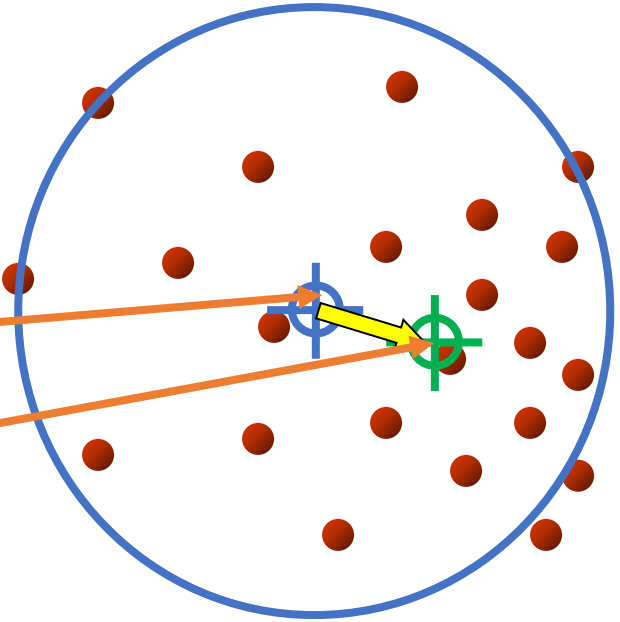


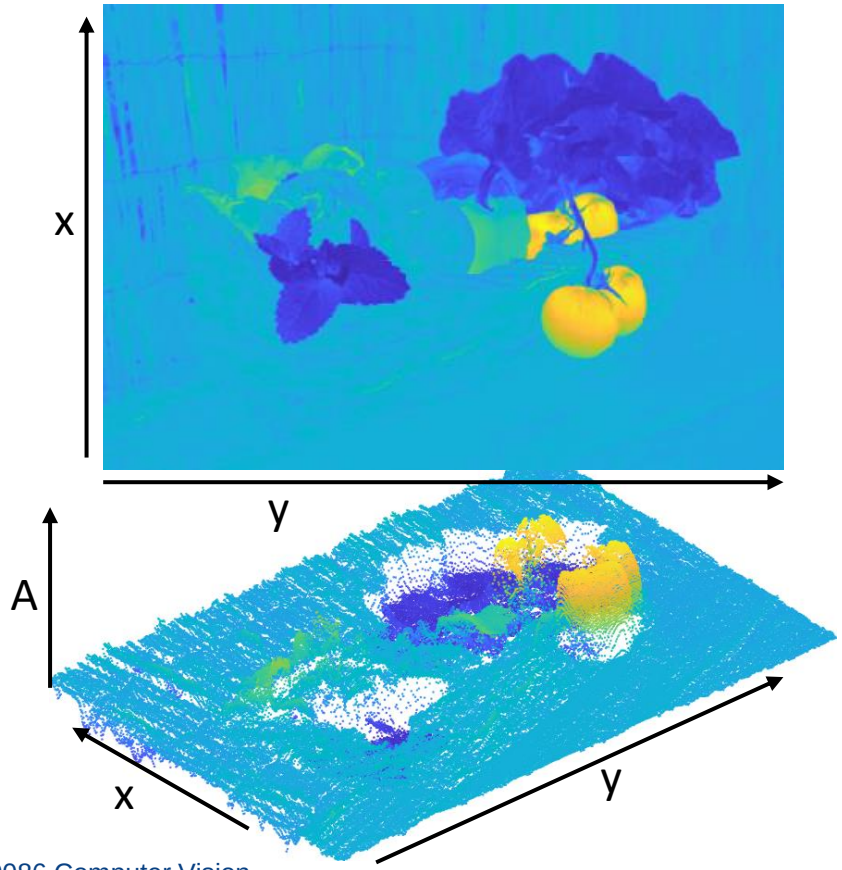
Image: Y. Ukrainitz & B. Sarel

Mean shift algorithm

- For each point:
  - Centre a window on that point
  - Compute the mean of the data in the search window
  - Centre the search window at the new mean location
  - Repeat (b,c) until convergence
- Assign points that lead to nearby modes to the same cluster
- Free parameters: kernel (commonly Gaussian), bandwidth

Mean shift segmentation

- Cluster in spatial+colour space; e.g.: (x,y,R,G,B) or (x,y,L,A,B) coordinates



Mean shift parameters

Increasing bandwidth →

Increasing spatial bandwidth relative to colour →

Summary

- Pixel clustering is a fast, simple approach to image segmentation
- Example: mean shift clustering in the colour+spatial domain
  - Automatically discover number of clusters; no need to choose k
  - But do need to choose bandwidth
- Pixel clustering separates colour regions – regions may not correspond to objects

Superpixels

Superpixels

- **Oversegmentation** methods segment image into regions that are smaller than objects
  - Objects are separated from background
  - But objects are also separated into many parts
- Superpixels = groups of adjacent pixels with similar characteristics (e.g., colour)

Superpixel segmentation

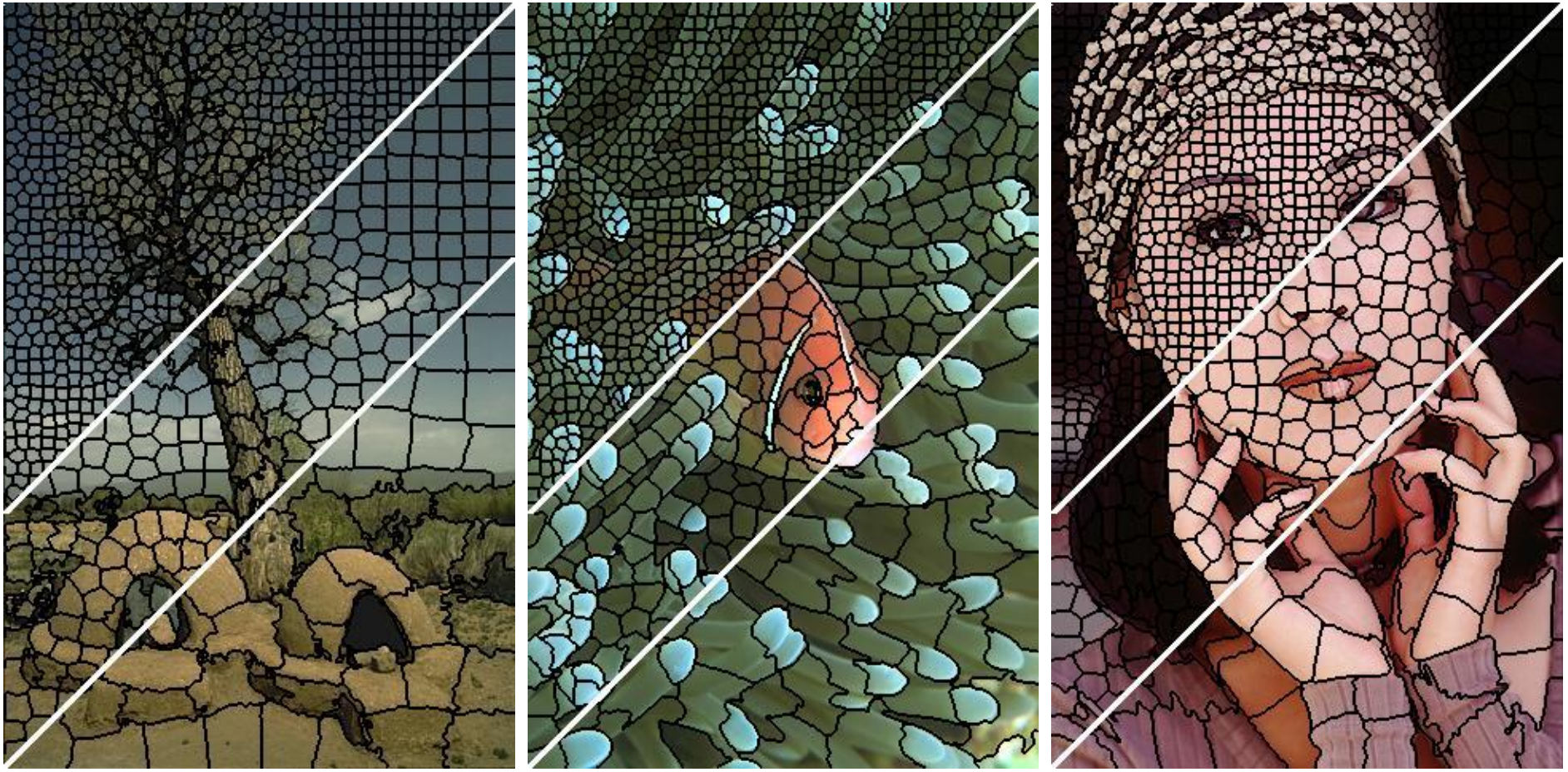


Image: https://www.epfl.ch/labs/ivrl/research/slic-superpixels/

SLIC superpixel algorithm

- Initialise cluster centres on non-edge pixels:
  - Initialise k cluster centres $c_k = [x_k, y_k, l_k, a_k, b_k]$ by sampling the image in a regular grid
  - For each centre c_k, check an N x N neighbourhood around c_k to find the pixel with lowest gradient. Set c_k to this pixel's [x, y, l, a, b].

SLIC superpixel algorithm

- For each cluster centre c_k:
  - In a 2M x 2M square neighbourhood around c_k, measure pixel similarity to c_k
  - Assign pixels with similarity < threshold to cluster k
  - Compute new cluster centre c_k
- Repeat until average change in cluster centres (L1 distance) falls below a threshold
- Similarity measure: $D = D_{lab} + \frac{\alpha}{M} D_{xy}$

$$D_{lab} = \sqrt{(l - l)^2 + (a - a_k)^2 + (b - b_k)^2}$$

$$D_{xy} = \sqrt{(x - x_k)^2 + (y - y_k)^2}$$

α = weighting parameter

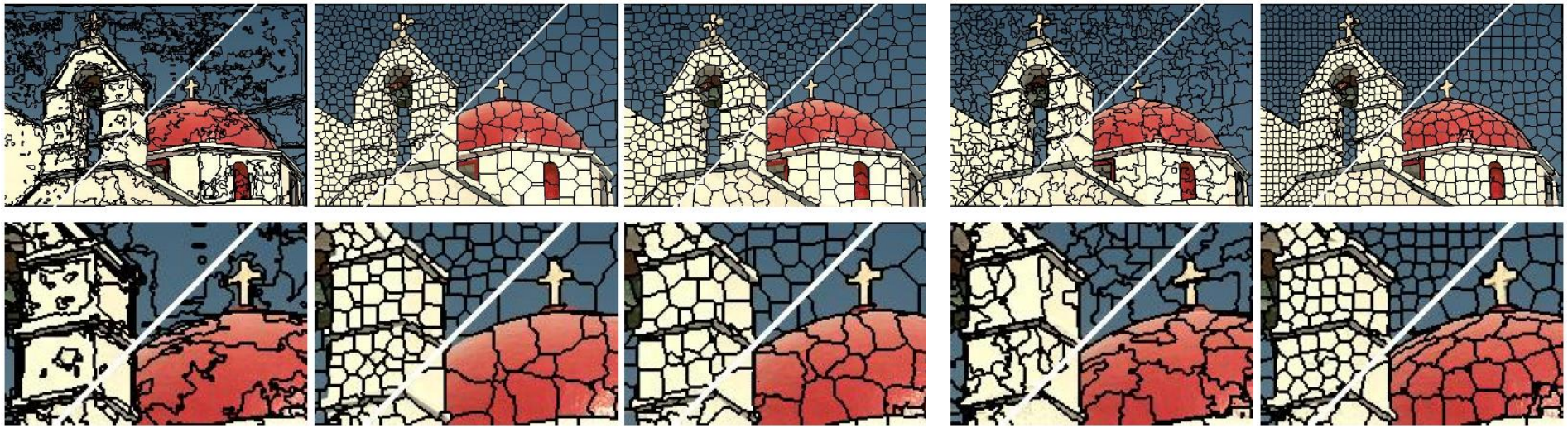
SLIC superpixel algorithm

- Similarity metric does not guarantee that clusters will be connected pixels
- To enforce connectivity, pixels not connected to main cluster are re-assigned to closest adjacent cluster

Superpixel methods

Graph-based methods

Gradient-descent-based



Felzenszwalb & Huttenlocher (2004)

Veksler, Boykov, & Mehrani (2010) – spatially compact

Veksler, Boykov, & Mehrani (2010) – constant colour

QuickShift (Vedaldi & Soatto, 2008)

SLIC

Image: Archana, et al. (2012)

Superpixel applications

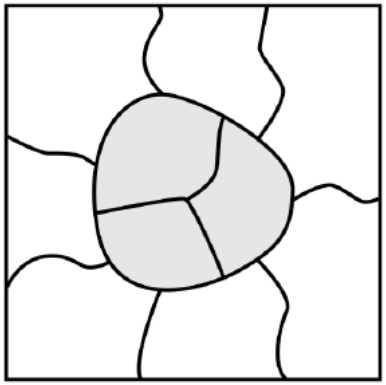
- Superpixels are a multipurpose intermediate image representation
- More compact representation for algorithms with high time complexity (600x800 pixels -> 200 superpixels)
- Common application: object segmentation
  - Oversegment image
  - Combine superpixels to find objects

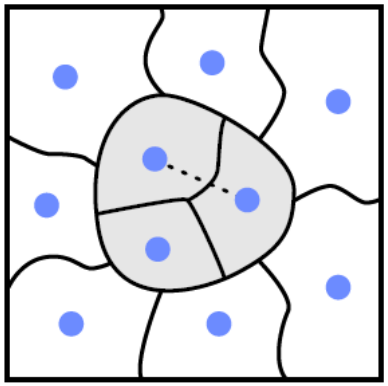
Superpixel merging

- Region Adjacency Graph (RAG)
  - Vertices = image regions (pixels or superpixels)
  - Edge weights = difference between regions
- To merge superpixels:
  - Identify edges below a threshold and re-label superpixels connected by these edges as one region
  - Or iteratively:
    - Find lowest-weight edge, relabel connected superpixels as one region
    - Recompute RAG, repeat until a criterion is met (e.g., all edges above a threshold)

Superpixel merging

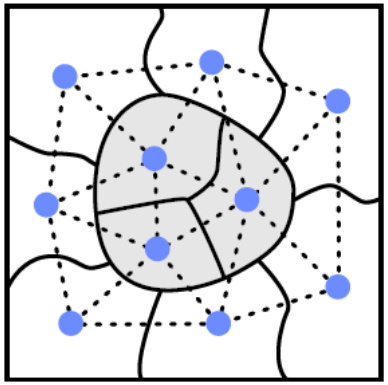
Superpixels

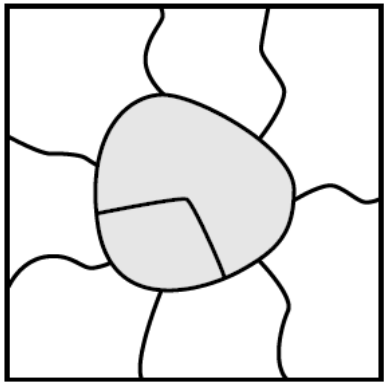




Find lowest-weight edge, merge superpixels

RAG





Recompute RAG, repeat

Image: Galvão, Guimarães, & Falcão (2020)

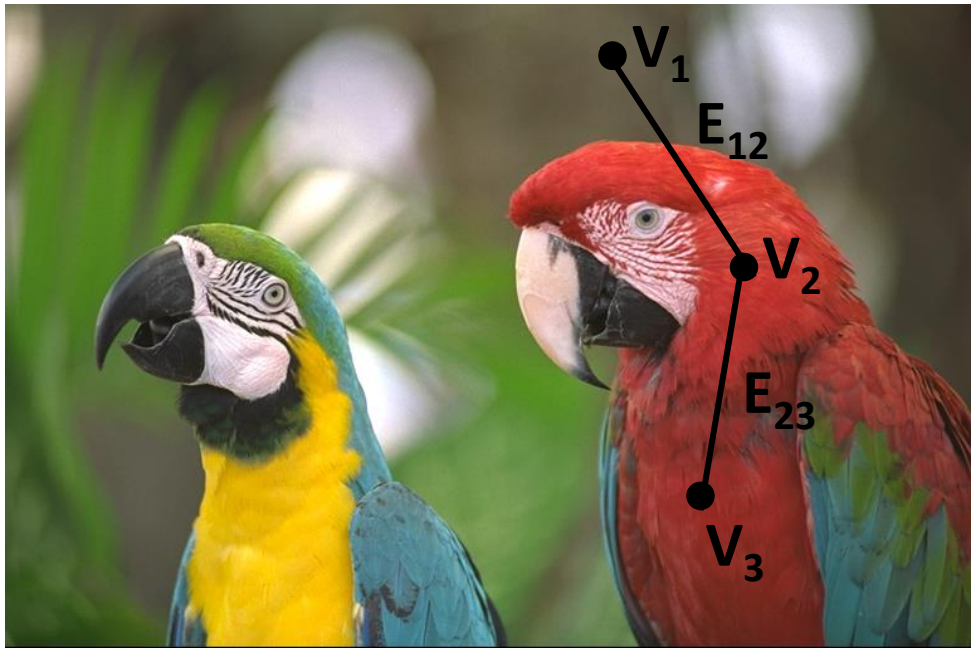
Summary

- Superpixels = regions of similar pixels, produced through oversegmentation
- Various algorithms for computing superpixels, SLIC is one common option
- Superpixels are a compact, intermediate representation used as a first step for:
  - Segmentation (especially graph-based methods)
  - Object detection/localisation
  - Video tracking

Graph-based segmentation

Images as graphs

- Represent image as a graph **G** = (**V**,**E**)
  - Vertices = image regions (pixels or superpixels)
  - Edge weights = similarity between regions



$E_{12} < E_{23}$

Graph cuts

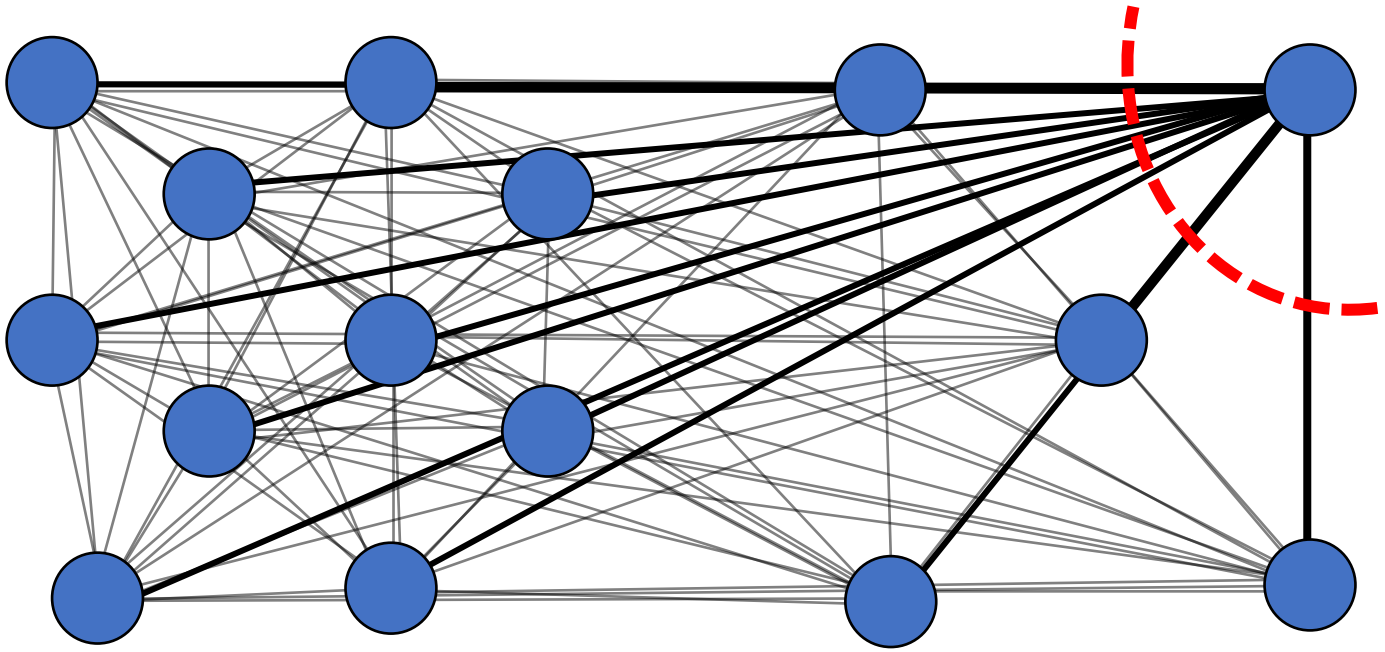
- Consider image as a fully-connected graph
- Partition graph into disjoint sets A,B to maximize total edge weight = remove low-weight edges between dissimilar regions
- Minimize value of cut:

$$cut(A, B) = \sum_{u \in A, v \in B} w(u, v)$$

Weight of edge connecting u and v

Graph cuts

- Not ideal for image segmentation – tends to create small, isolated sets



Edge weight = 1/distance

Normalised cuts

- Instead of minimizing cut value, minimize cut value as a fraction of total edge connections in entire graph (normalised cut)
- Normalised cut (Shi & Malik, 2000):

$$Ncut(A,B) = \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(B,V)}$$

$$= \frac{\sum_{u \in A, v \in B} w(u,v)}{\sum_{u \in A, t \in V} w(u,t)} + \frac{\sum_{u \in A, v \in B} w(u,v)}{\sum_{v \in B, t \in V} w(v,t)}$$

Normalized cuts results

Image: D. Hoiem

GrabCut

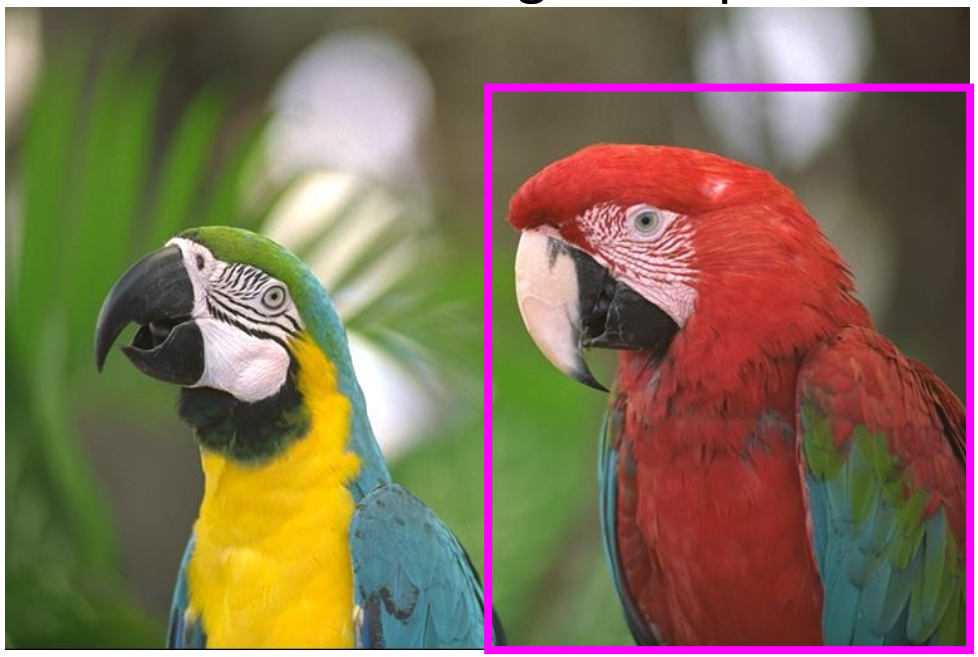
- Segments image pixels into just two classes: foreground (object) and background
- Uses colour clustering + graph cuts to find optimal classification of pixels into each class

Rother, Kolmogorov, & Blake (2004)

GrabCut algorithm

- Requires user to initialise algorithm with a bounding box

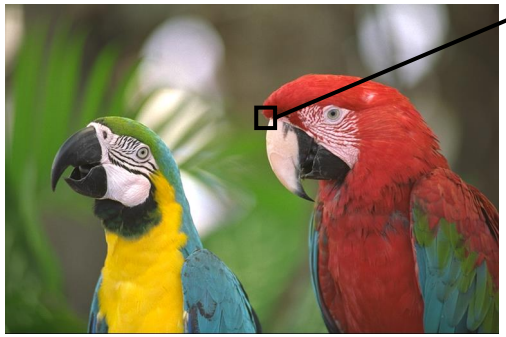
Outside box = background pixels

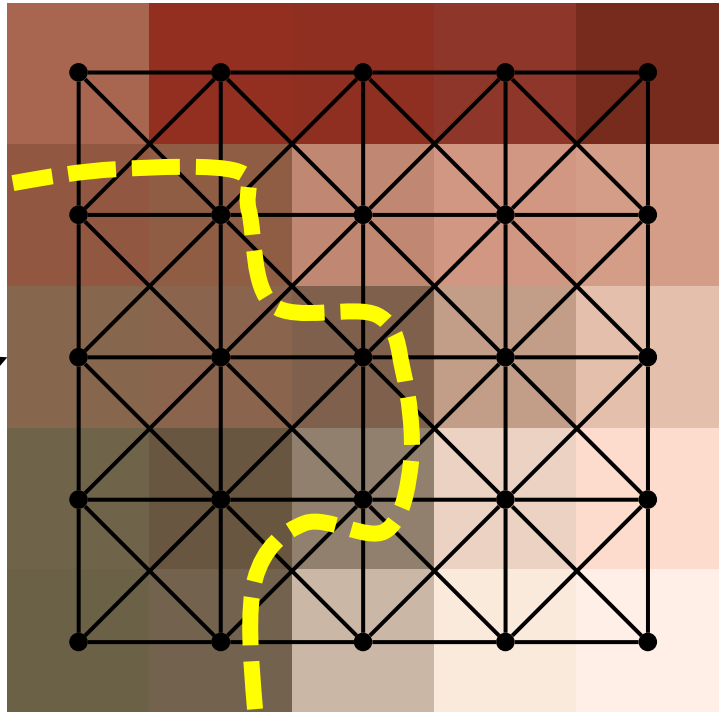


Inside box = treat as foreground pixels initially

GrabCut algorithm

- For each class (foreground, background), represent distribution of pixel colour as a Gaussian mixture model (GMM)
- Represent image pixels as a graph (8-way connectivity)





GrabCut algorithm

- Denote the pixel graph as **G** and the GMM as θ
- α indicates label of each pixel (foreground or background)
- Iterate until convergence:
  - Find graph cut (label assignment) to minimize

$$E(\alpha, \theta, \mathbf{G}) = U(\alpha, \theta, \mathbf{G}) + \gamma V(\alpha, \mathbf{G})$$

-log likelihood of cluster assignments in GMM

Weighting parameter

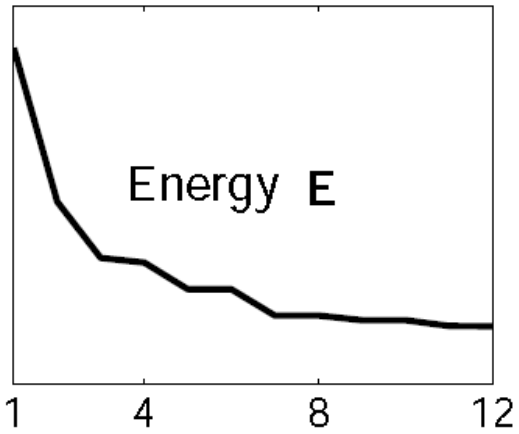
Smoothness penalty based on colour similarity, applied to neighbouring pixels with different labels in α

  - Recompute GMM for new label assignment

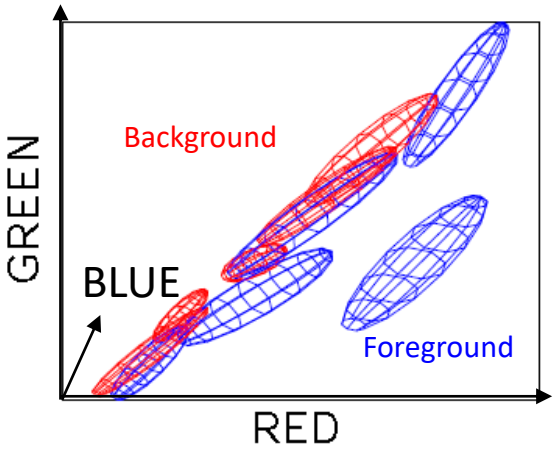
GrabCut example

Initialisation

E over iterations



Initial GMM



Final GMM

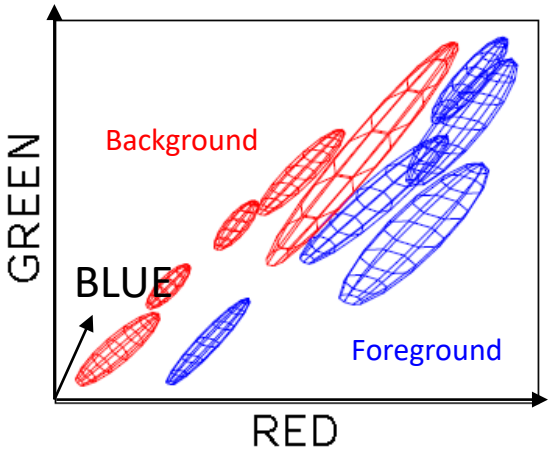
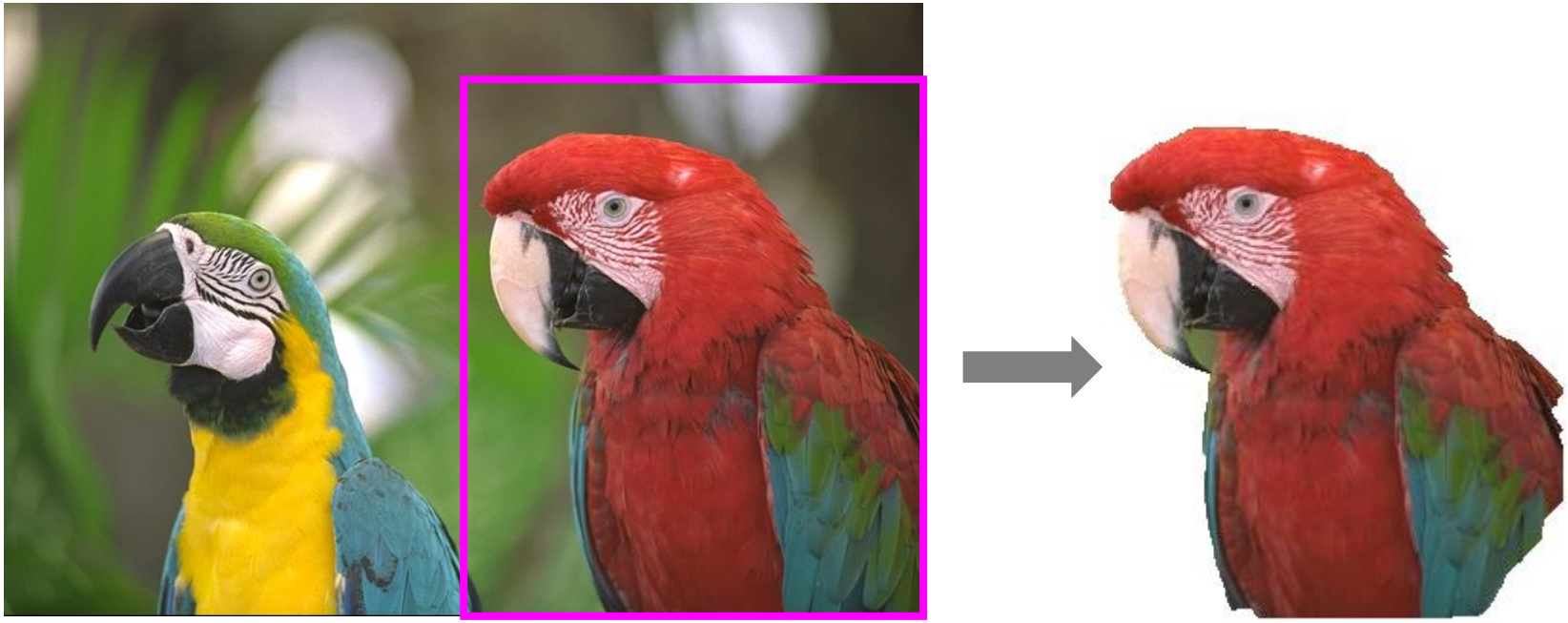


Image: Rother, Kolmogorov, & Blake (2004)

GrabCut result



https://chocopoule.github.io/grabcutweb/

Summary

- Graph-based methods represent an image as a graph (of pixels or superpixels)
- Segmentation removes edges to break graph into subgraphs, generally trying to optimize:
  - Similarity within connected region
  - Dissimilarity across disconnected regions
  - Smoothness/connectivity of connected regions
- Normalized cuts – segment into multiple regions
- GrabCut – segment into foreground/background

Summary

- Various ways to approach image segmentation, but many methods use some combination of pixel clustering and graph analysis
- The methods discussed so far do segmentation but not semantic segmentation (regions with no labels)
- How to get labels?
  - Unlabelled regions can be input to an object classification method
  - Or, segmentation and classification can be done simultaneously (next lecture)

THE UNIVERSITY OF MELBOURNE

Image Segmentation II

Semester 2, 2021

Kris Ehinger

Demo

- https://removal.ai

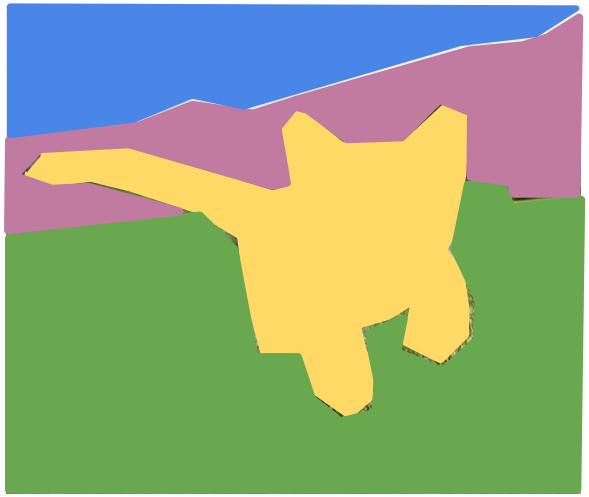
Outline

- Segmentation as classification
- U-Net
- Instance segmentation

Learning outcomes

- Explain how semantic segmentation is performed by fully-convolutional networks
- Implement max unpooling and transposed convolution
- Explain two common architectures for segmentation problems (U-Net and Mask R-CNN)

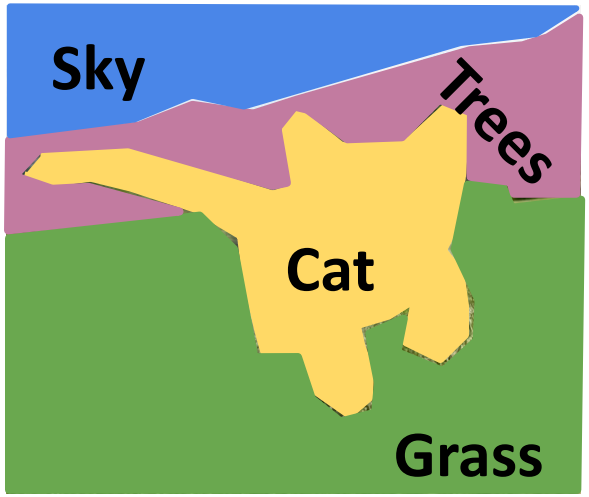
Segmentation

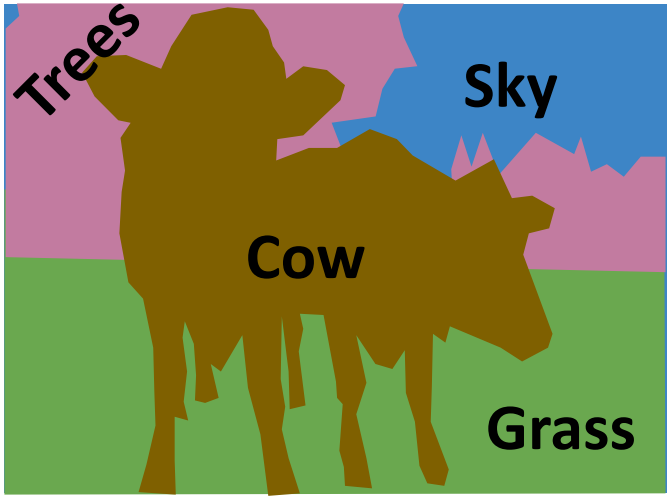


Separate image into different regions (objects, textures)

Image: J. Johnson

Semantic segmentation





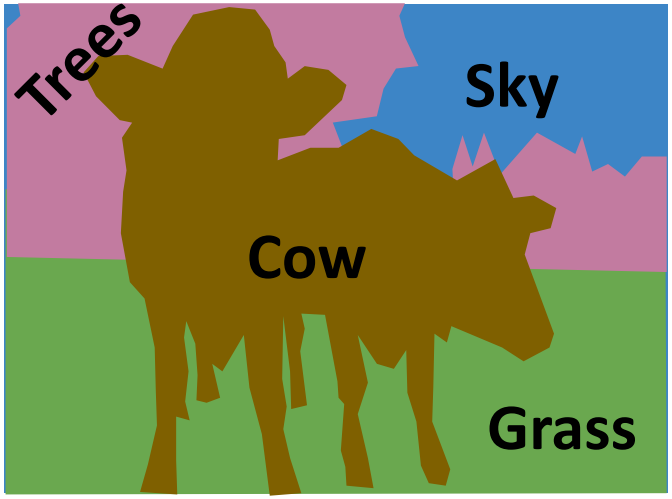
Separate image into different *labelled* regions

Image: J. Johnson

Instance segmentation

Semantic segmentation

Instance segmentation



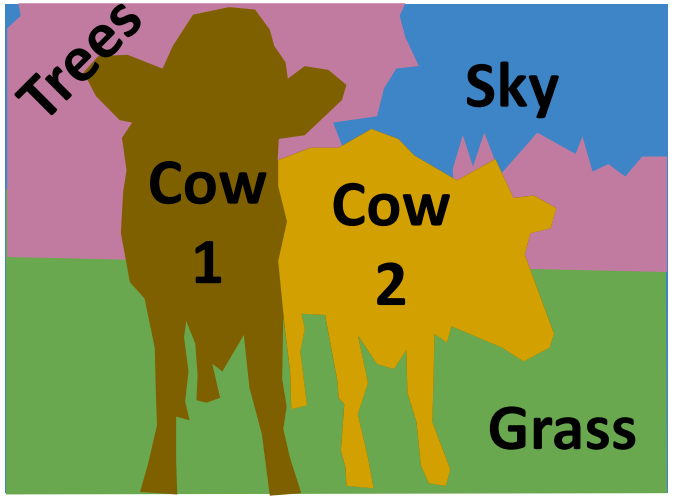
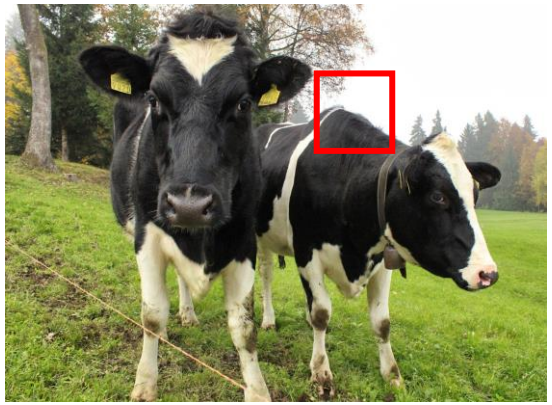
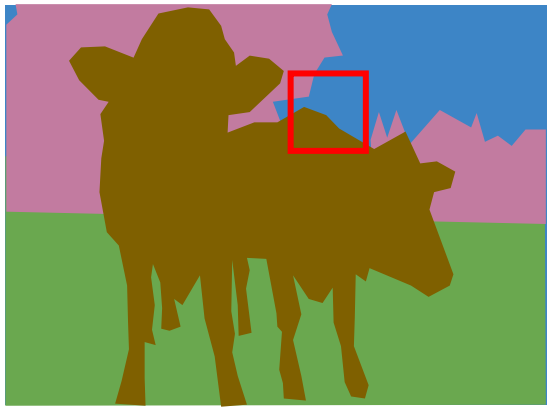


Image segmentation



Input: Image

Clustering?
Graph cuts?
Classification?

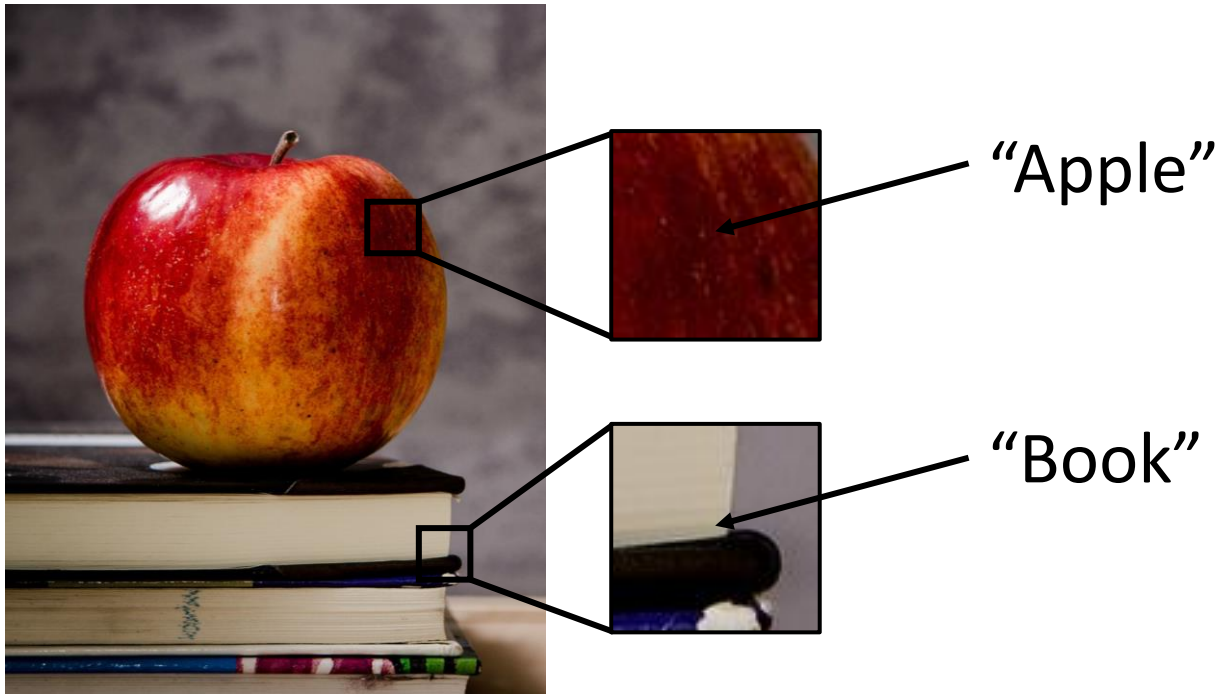


Output: Pixel classification
(and, optionally, labels)

Segmentation as classification

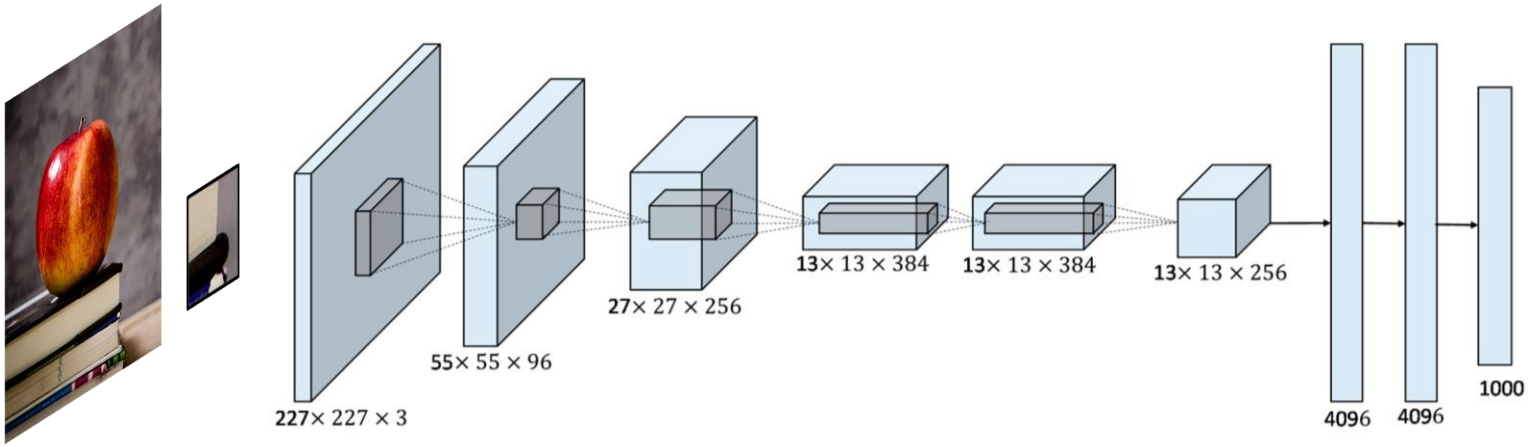
Pixel classification

- Image segmentation as a classification problem
- Given a window (NxN pixels), classify central pixel

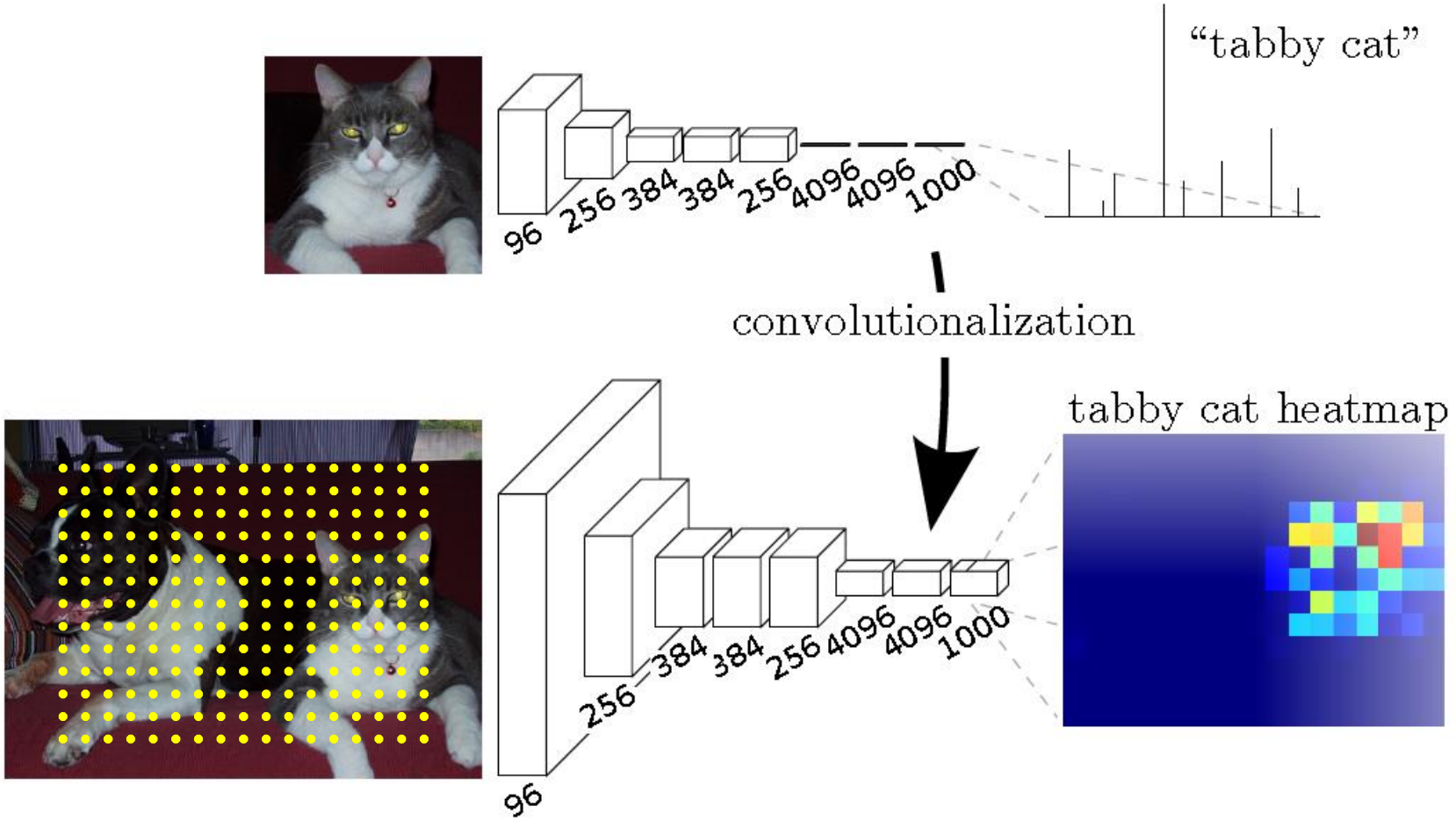


Pixel classification

- Classifying individual pixels is potentially very slow (e.g., a small image = 600x800 = 480,000 pixels)
- But a CNN can classify multiple pixels in parallel



Parallel patch classification



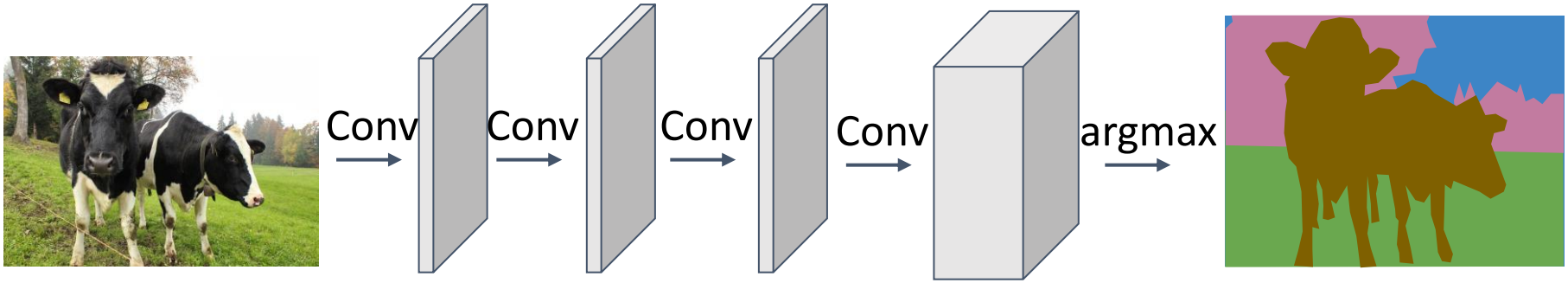
Long, Shelhamer, & Darrell (2015)

Fully-convolutional network

- Fully-connected network (FCN) has only convolutional layers, no fully-connected layers
- Last layer is a spatial map
- Can accept any size image as input, output map size depends on input size

Parallel patch classification

- Standard CNN architecture poses two problems:
  - Receptive field size is linear with the number of convolutional layers
  - Most methods downsample (e.g., with maxpooling) to reduce computation



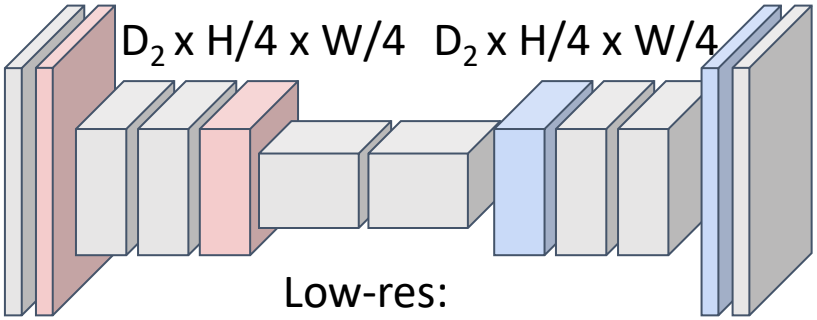
We could avoid maxpooling, but this will make the network slow and doesn't solve the first problem

Image: J. Johnson

Parallel patch classification

- Solution: use encoder-decoder structure
- Encoder downsamples image, decoder upsamples

Input:
3 x H x W



High-res:
D_1 x H/2 x W/2

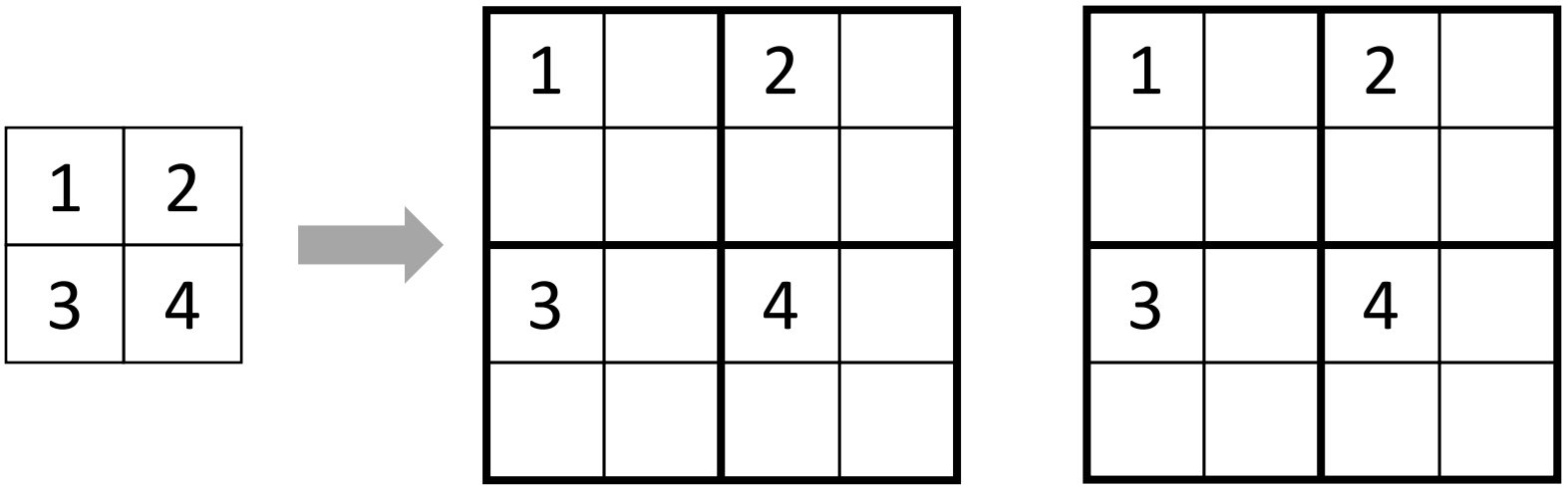
High-res:
D_1 x H/2 x W/2

Predictions:
H x W

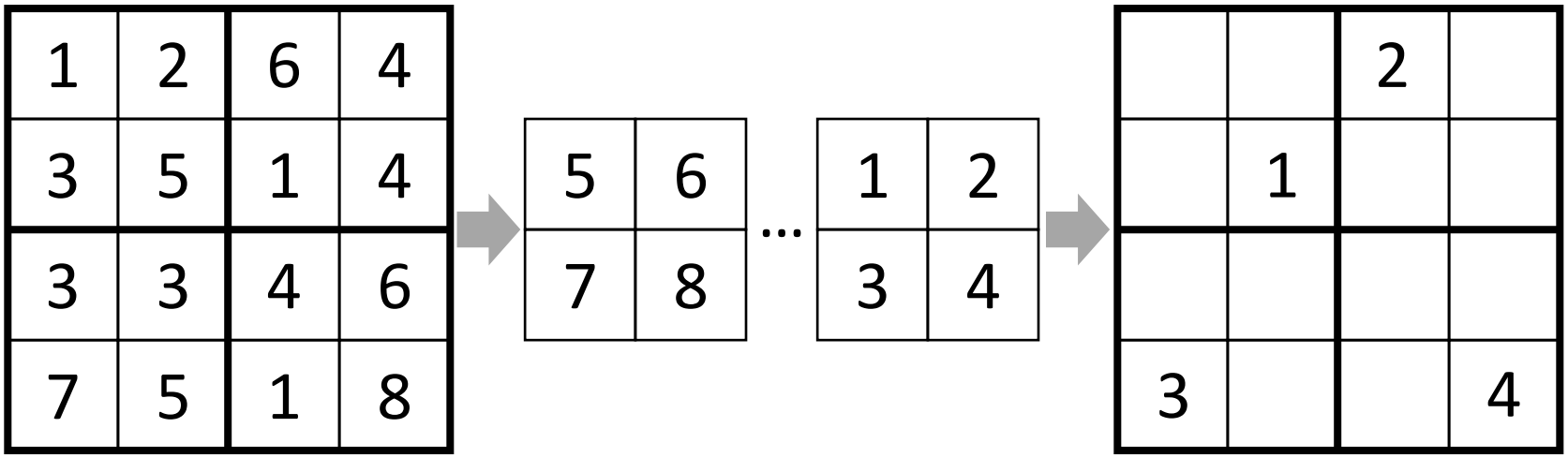
Image: J. Johnson

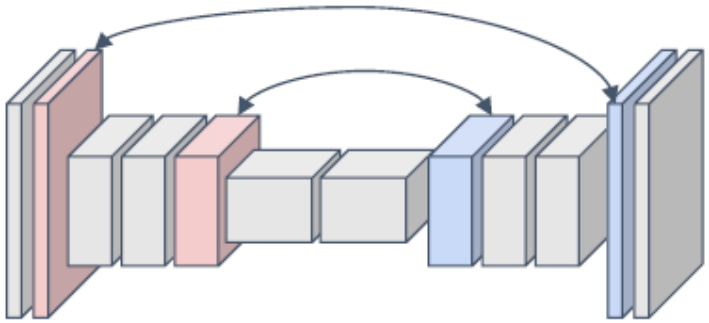
Upsampling

- How to upsample a feature layer in a CNN?



Max Unpooling



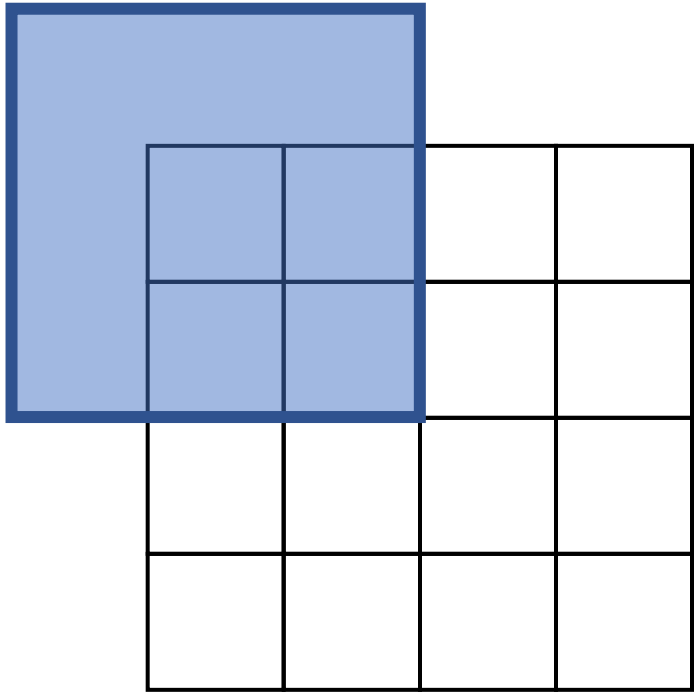


Each upsampling layer is paired with a downsampling layer
The locations of the max items are saved and passed to upsampler

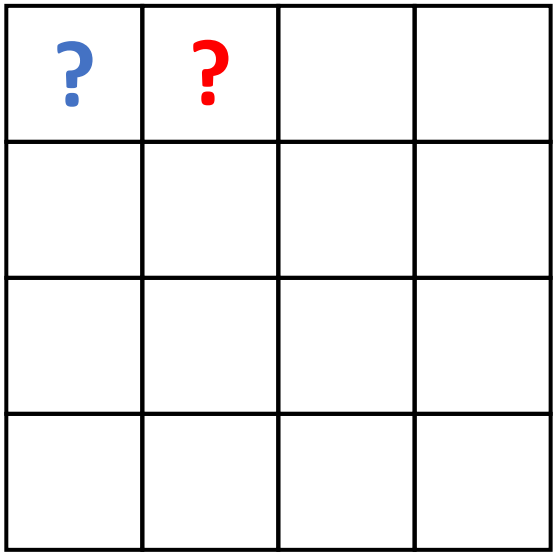
Image: J. Johnson

Convolution review

kernel = 3x3, stride = 1, pad = 1



Input: 4 x 4

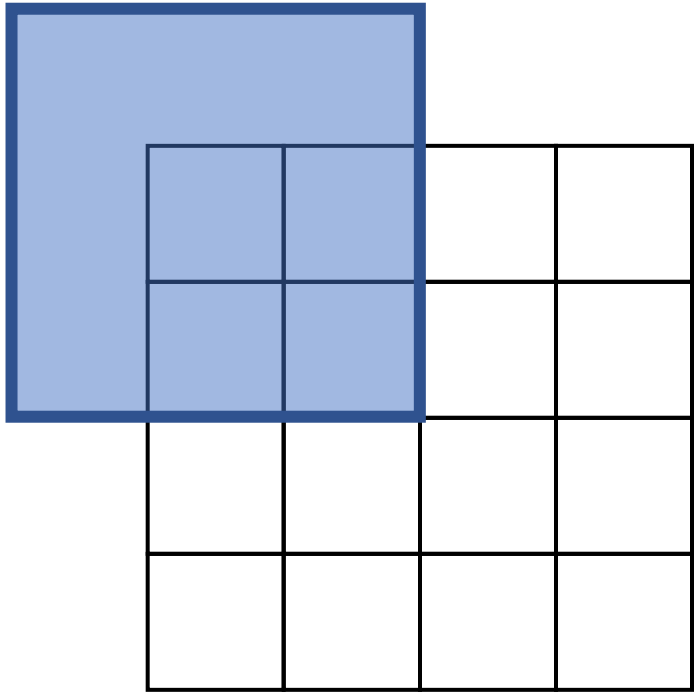


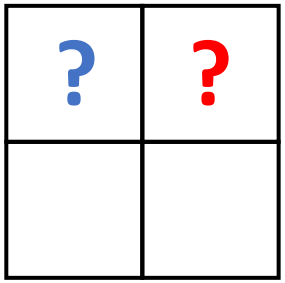
Output: 4 x 4

Based on slide by: J. Johnson

Convolution review

kernel = 3x3, stride = 2, pad = 1





Input: 4 x 4

Output: 2 x 2

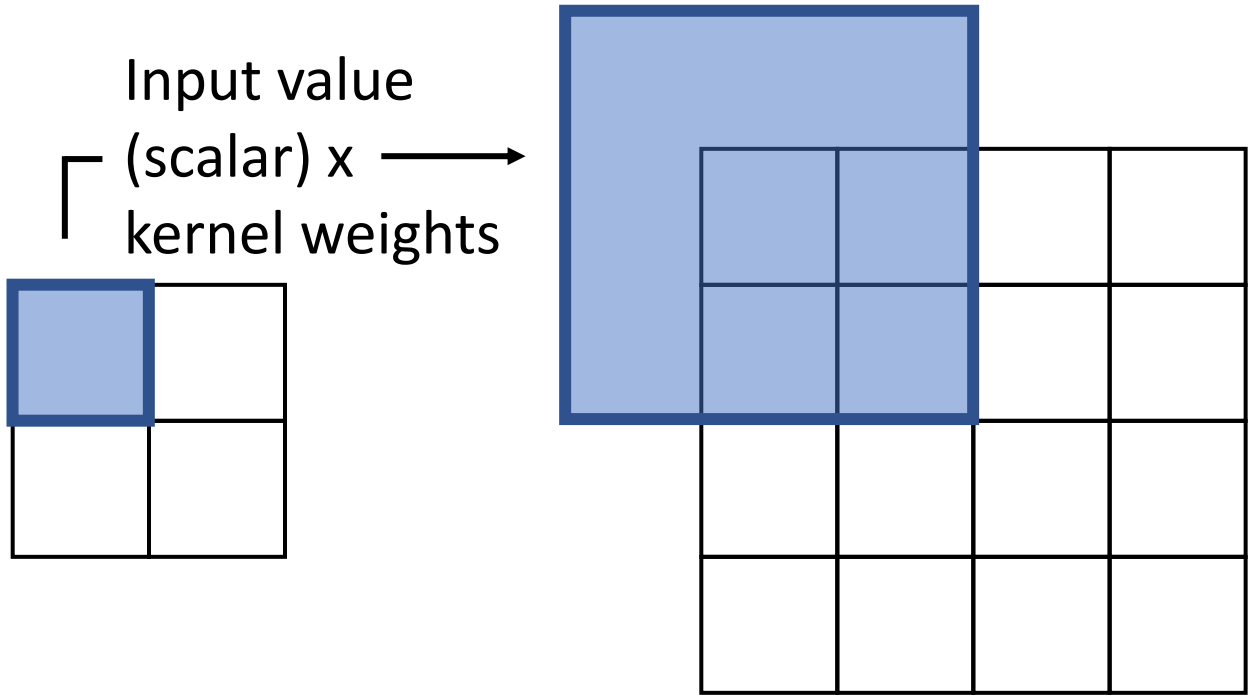
Based on slide by: J. Johnson

Transposed convolution

- Convolution with stride > 1 does a form of downsampling
  - E.g., stride = 2 means filter moves 2 pixels in input for every 1 pixel in output
  - “Learnable downsampling”
- Can we reverse this to do upsampling?
  - E.g., filter moves 2 pixels in *output* for every 1 pixel in *input*
- **Transposed convolution**: convolution with a stride < 1
  - In some papers, may also be called deconvolution, upconvolution, fractionally strided convolution

Transposed convolution

3x3 transposed convolution, stride = 2



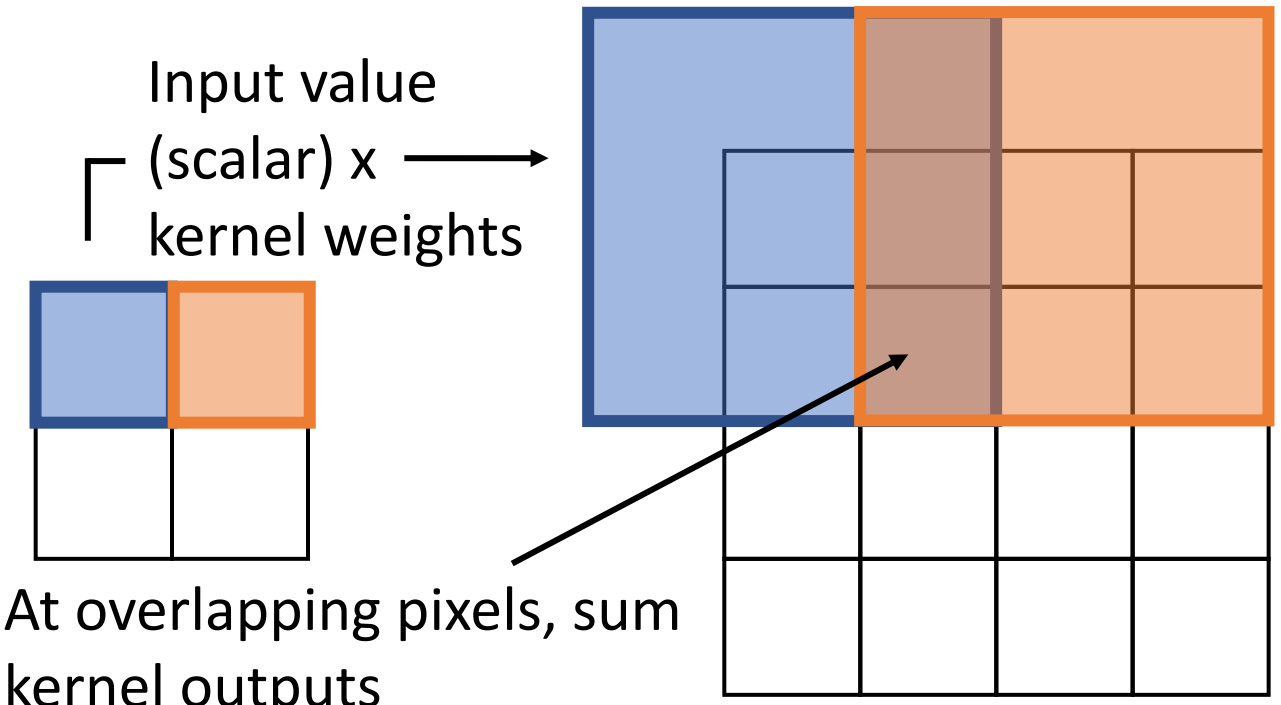
Input: 2 x 2

Output: 4 x 4

Based on slide by: J. Johnson

Transposed convolution

3x3 transposed convolution, stride = 2

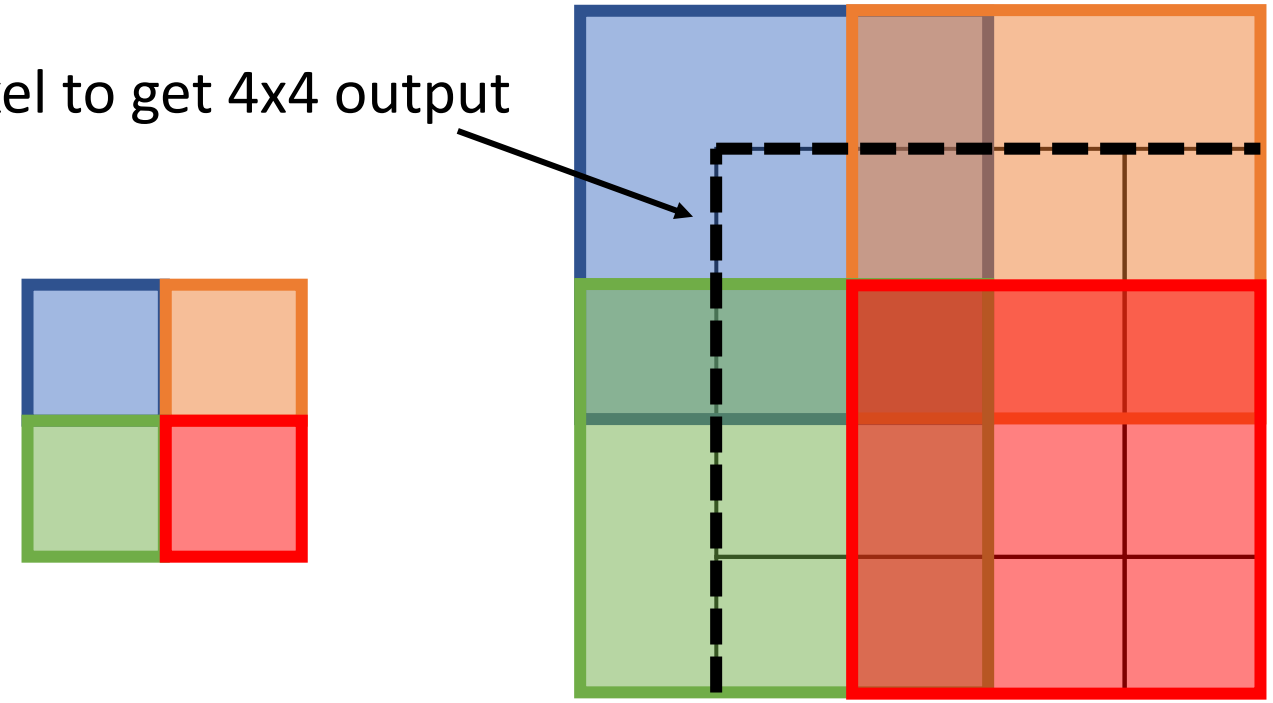


Based on slide by: J. Johnson

Transposed convolution

3x3 transposed convolution, stride = 2

Trim 1 pixel to get 4x4 output

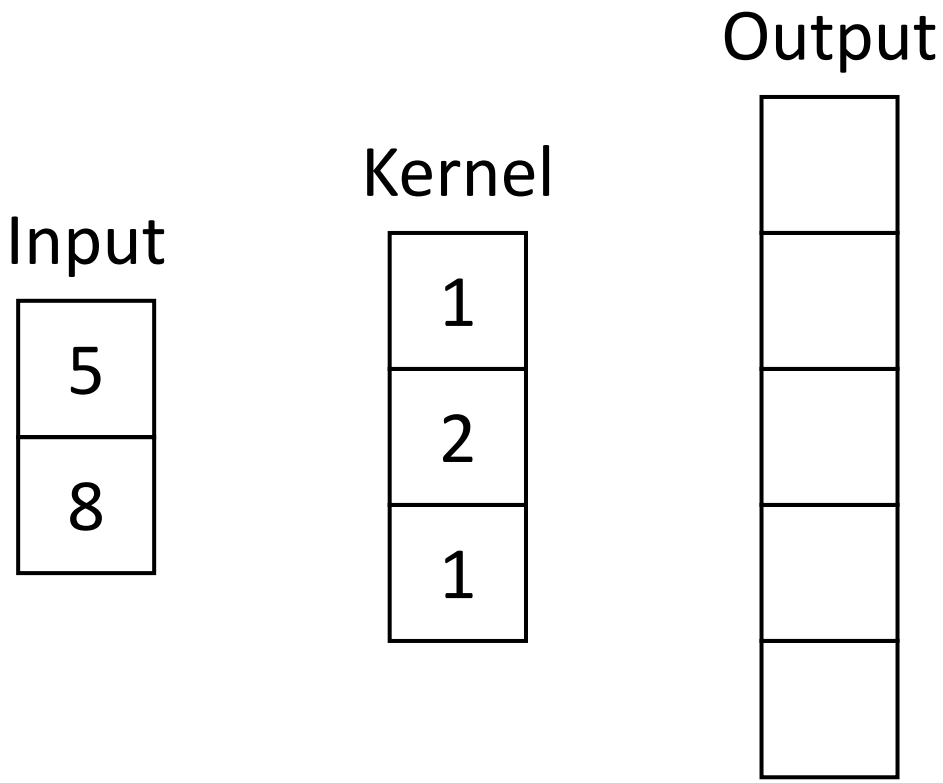


Input: 2 x 2

Output: 4 x 4

Based on slide by: J. Johnson

Practice: 1D example



PollEv.com/
krisehinger432

Drag the
correct values
into spaces 1-5

Why is it transposed convolution?

We can express convolution as a matrix multiplication:

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & x & y & z & 0 & 0 \\ 0 & 0 & x & y & z & 0 \\ 0 & 0 & 0 & x & y & z \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{bmatrix}$$

kernel=3, stride=1, pad=1

Transposed convolution multiplies by the transpose of the same matrix:

$$\vec{x} *^T \vec{a} = X^T\vec{a}$$

$$\begin{bmatrix} x & 0 & 0 & 0 \\ y & x & 0 & 0 \\ z & y & x & 0 \\ 0 & z & y & x \\ 0 & 0 & z & y \\ 0 & 0 & 0 & z \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} ax \\ ay + bx \\ az + by + cx \\ bz + cy + dx \\ cz + dy \\ dz \end{bmatrix}$$

kernel=3, stride=1, pad=1

Based on slide by: J. Johnson

Why is it transposed convolution?

We can express convolution as a matrix multiplication:

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & 0 & x & y & z & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix}$$

kernel=3, stride=2, pad=1

Transposed convolution multiplies by the transpose of the same matrix:

$$\vec{x} *^T \vec{a} = X^T \vec{a}$$

$$\begin{bmatrix} x & 0 \\ y & 0 \\ z & x \\ 0 & y \\ 0 & z \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az + bx \\ by \\ bz \\ 0 \end{bmatrix}$$

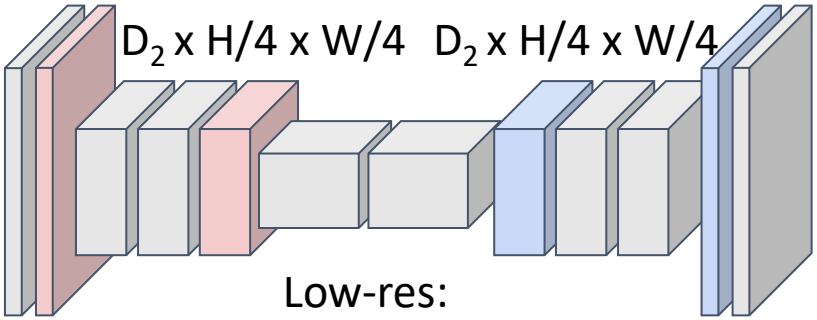
kernel=3, stride=2, pad=1

Based on slide by: J. Johnson

Parallel patch classification

- Solution: use encoder-decoder structure
- Encoder downsamples image, decoder upsamples

Input:
3 x H x W



High-res:
D_1 x H/2 x W/2

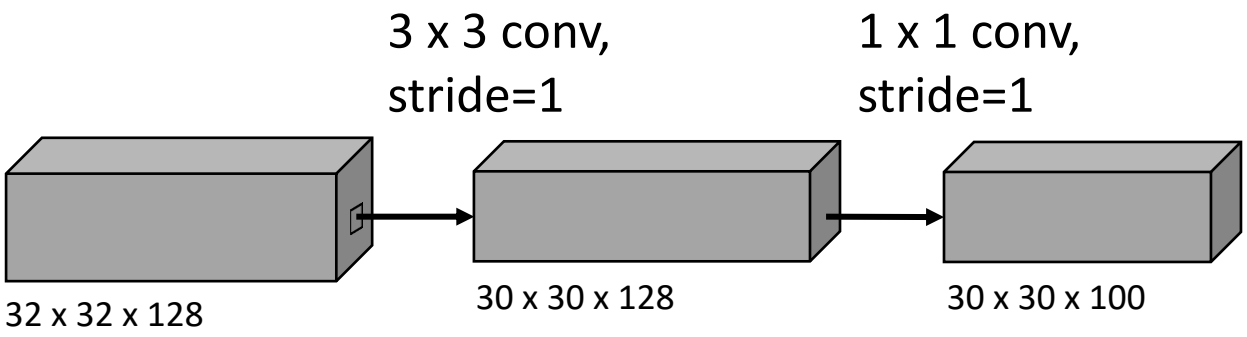
High-res:
D_1 x H/2 x W/2

Predictions:
H x W

Image: J. Johnson

1x1 convolution layer

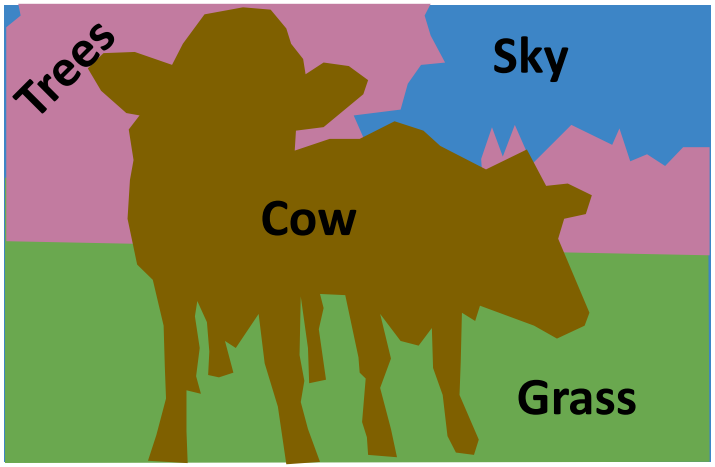
- Convolutional layer with a 1x1 kernel
- Commonly found as last layer(s) of a fully-connected network
- What do these kernels learn?



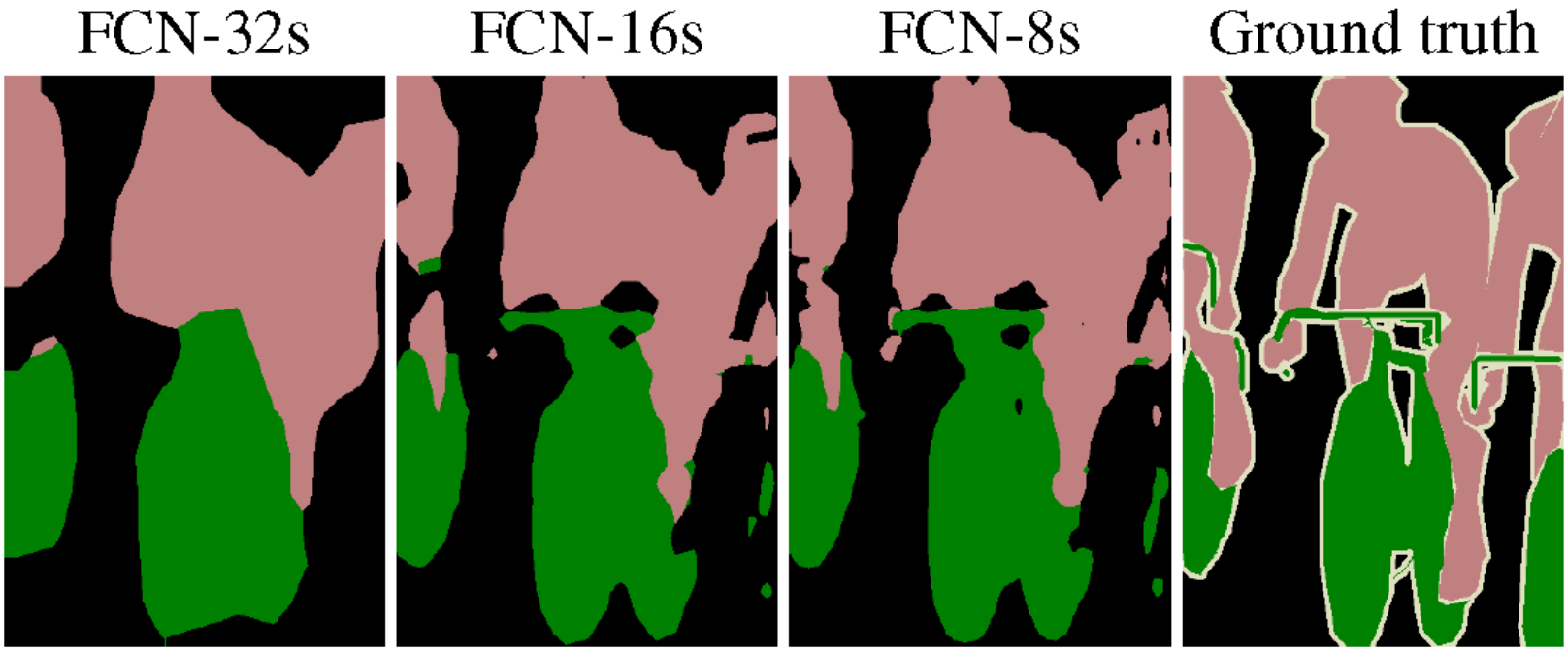
Loss function

- At each pixel, compute cross-entropy loss between predicted class and known classes

$$E = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$



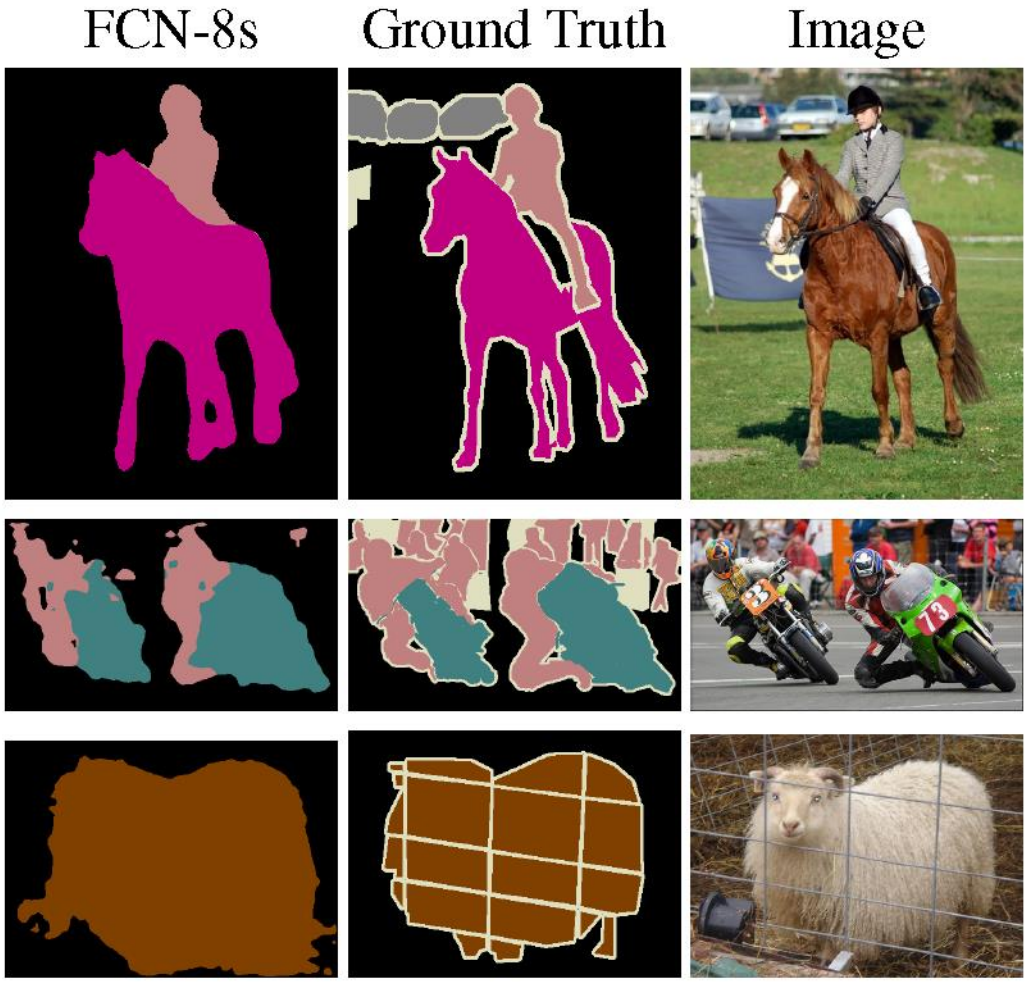
Segmentation with FCN results



Decreasing stride of convolution→

Long, Shelhamer, & Darrell (2015)

Segmentation with FCN results



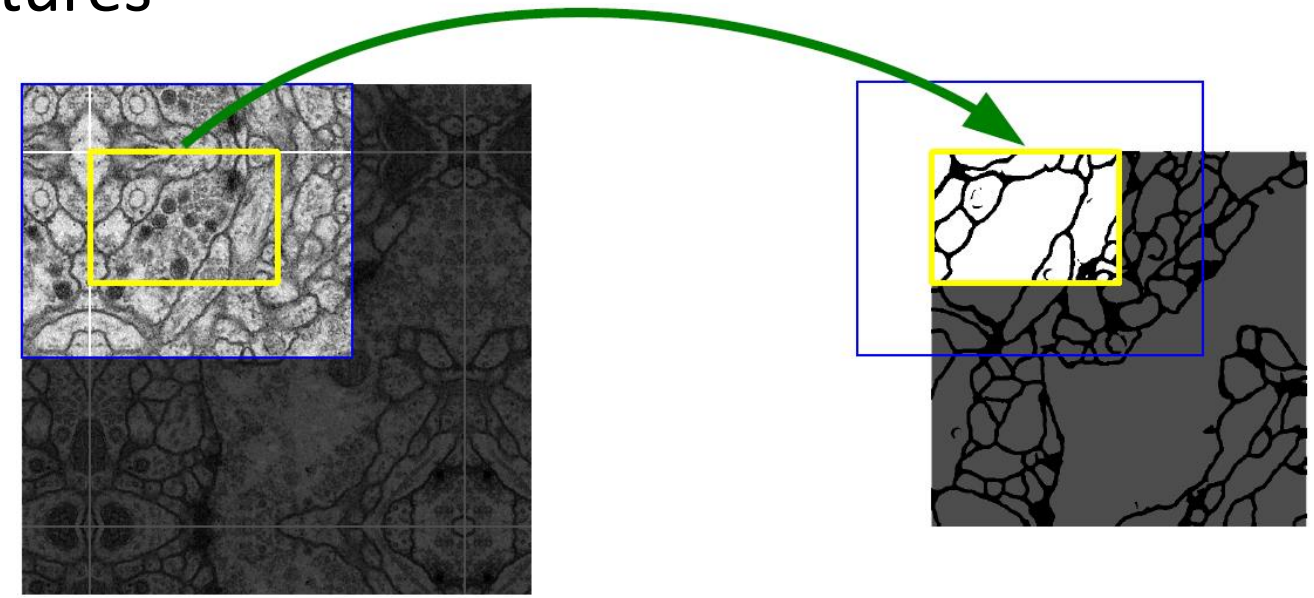
Summary

- Semantic segmentation can be treated as a pixel classification problem
- This can be done efficiently with a fully-convolutional network
- Encoder-decoder architecture:
  - Downsample with max pooling, strided convolution
  - Upsample with max unpooling, transposed convolution
- Output is a label for each pixel

U-Net

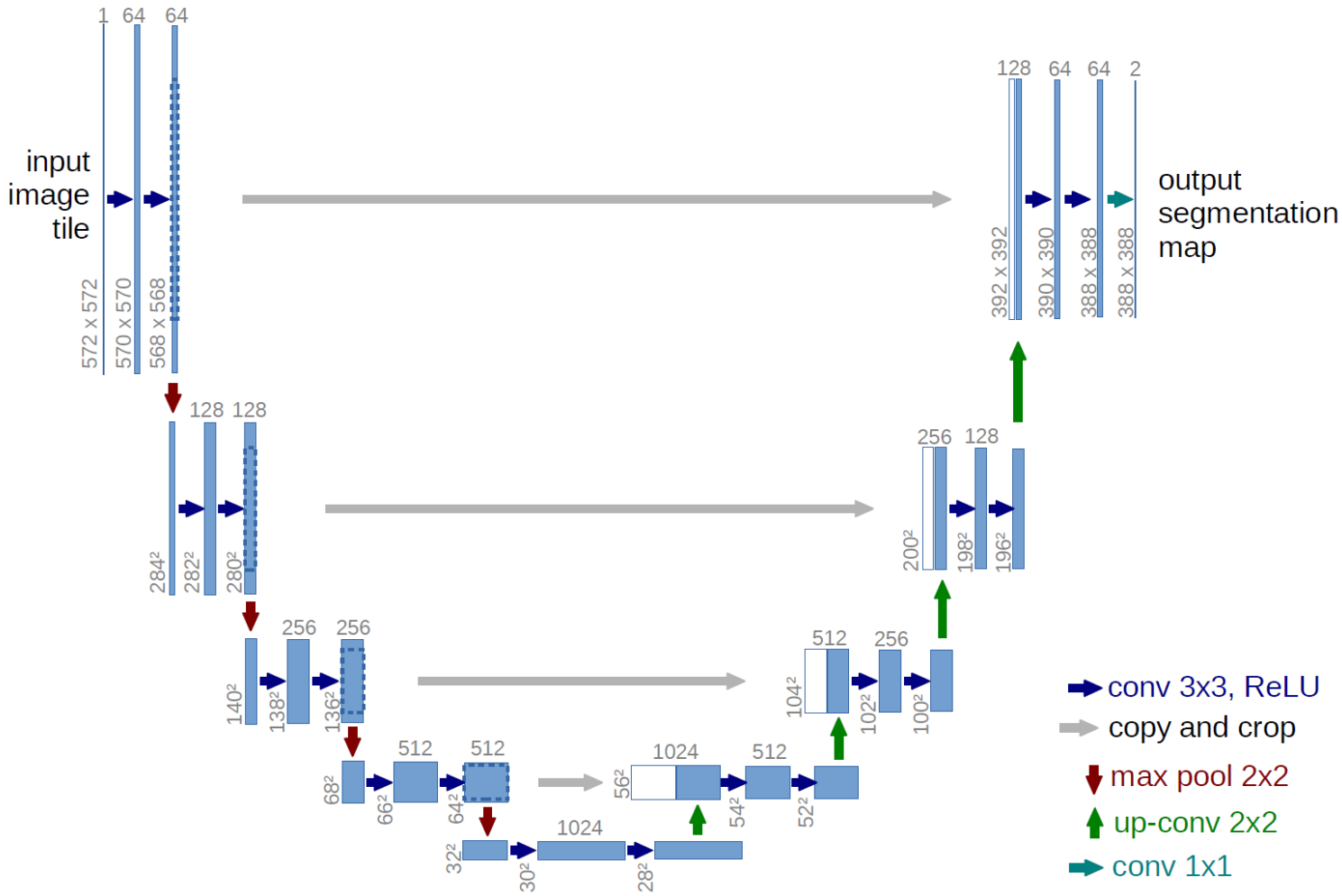
U-Net

- Originally proposed for medical image segmentation
- Encoder-decoder structure with some additional features



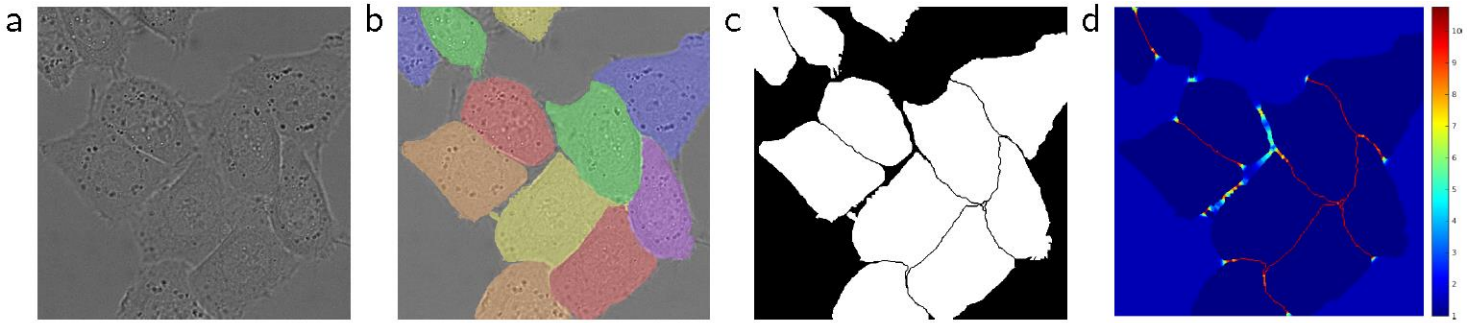
Ronneberger, Fischer, & Brox (2015)

U-Net architecture

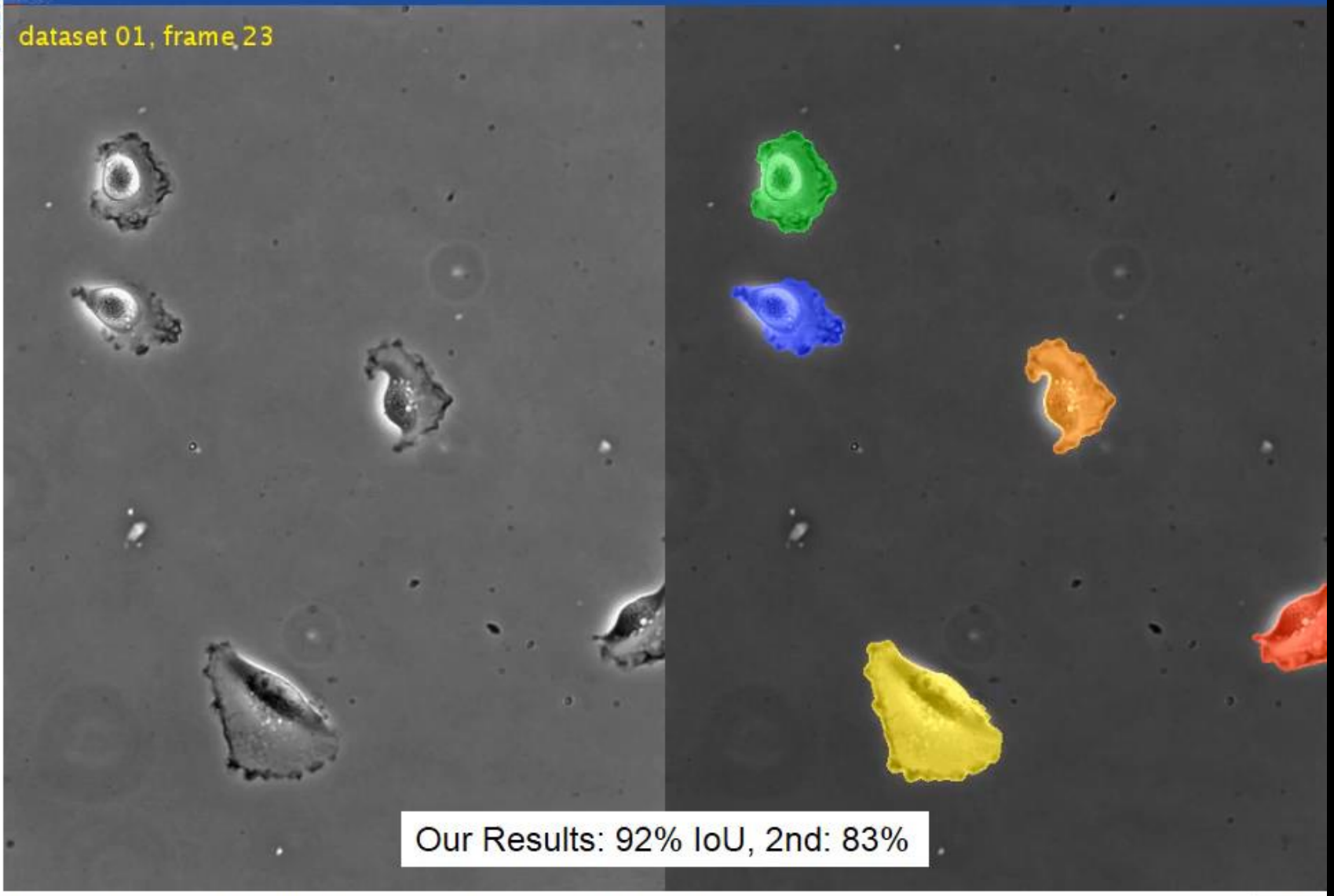


U-Net architecture

- In decoder, the upsampled feature map is concatenated with the original features from the corresponding encoder layer
- For cell segmentation, U-Net uses only 2 classes and weighted cross-entropy loss: edges between cells have higher weight



ISBI cell tracking challenge: **PhC-U373**



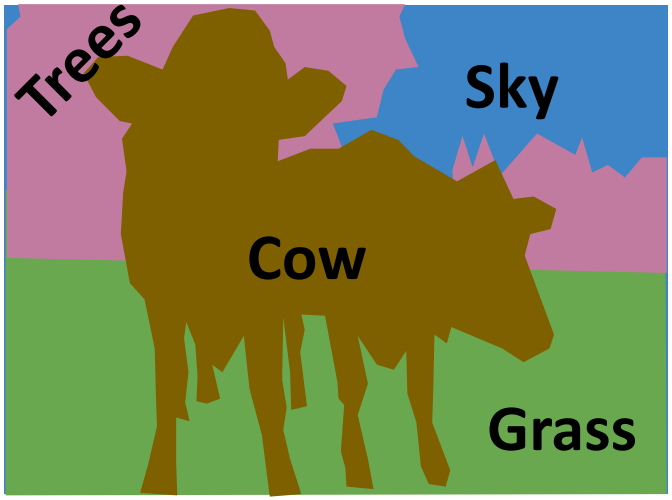
Summary

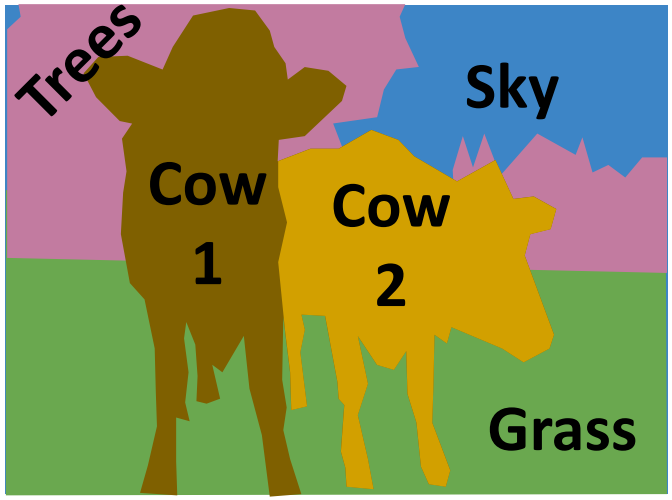
- U-Net: fully-convolutional network for segmentation, with some modifications
- Originally for medical image analysis, but commonly used for other segmentation tasks
- Also used as an encoding-decoding network for tasks like:
  - Image denoising
  - Image inpainting

Instance segmentation

Instance segmentation

- Semantic segmentation classifies pixels, doesn't distinguish between instances
- How to separate instances?





Side note: Stuff and things

- Visual scenes include both “things” and “stuff”
  - “Things” = objects, “countable nouns” (cow, person, car)
  - “Stuff” = regions/materials, “mass nouns” (road, grass)
  - Some classes can be either (trees)
- Instance segmentation (and computer vision in general) mainly focusses on “things”

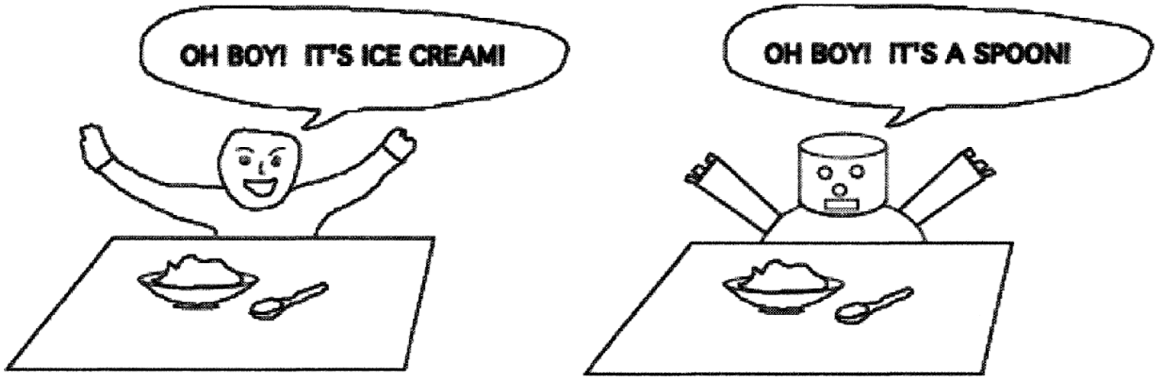


Image: Adelson (2001) “On seeing stuff: The perception of materials by humans and machines”

Instance segmentation

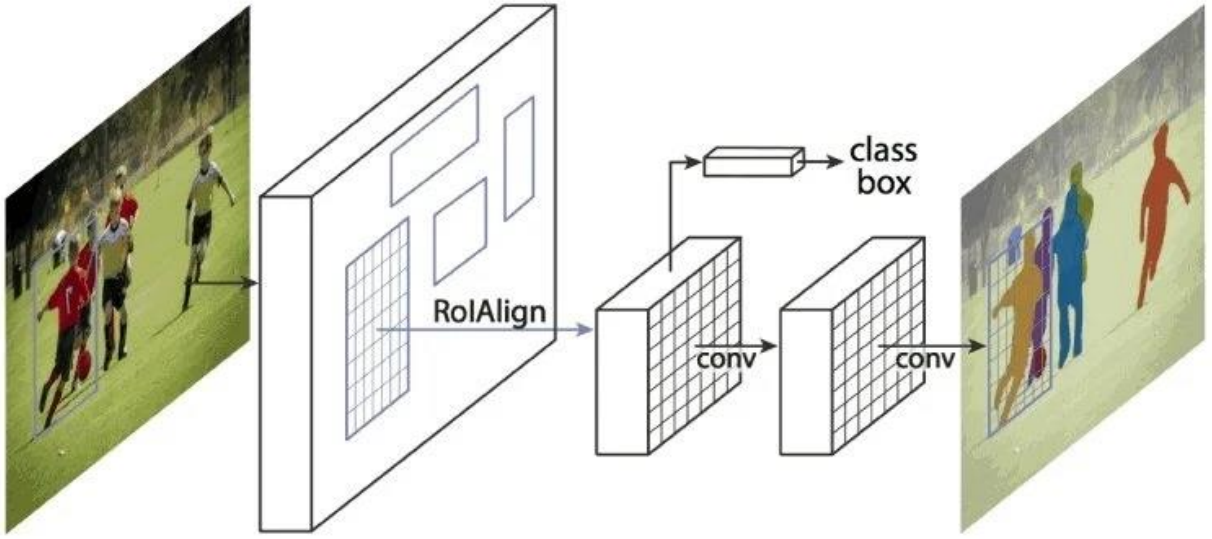
- Instead of giving each pixel an instance label, extract patches of image that are likely to be separate instances
- Do segmentation within each patch
- Commonly-used architecture: Mask R-CNN

R-CNN

- R-CNN = Region-based convolutional neural network
- Efficiently extracts “regions of interest” (image patches likely to contain objects) for further processing
- (We’ll cover these networks in more detail next week)

Mask R-CNN

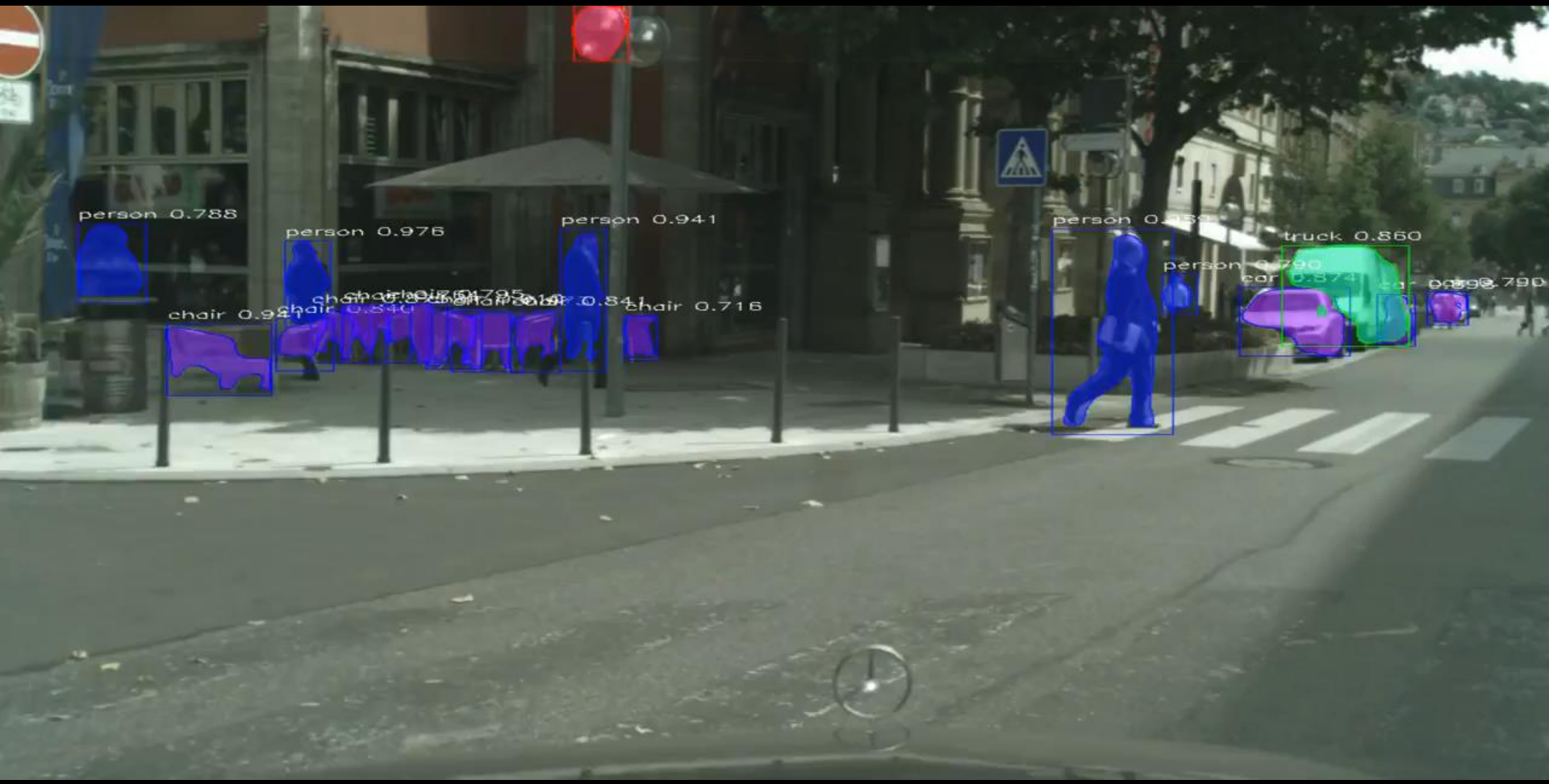
- Mask R-CNN takes patches extracted by R-CNN and runs them through a fully-convolutional network
- FCN predicts a binary segmentation mask ("object" or "background")



He, Gkioxari, Dollár, & Girshick (2017)

Mask R-CNN results

He, Gkioxari, Dollár, & Girshick (2017)



https://www.youtube.com/watch?v=EfaCtOEEmLA

Summary

- Instance segmentation gives separate labels to each instance of a class, not just class labels
- Generally only applies to countable “thing” classes
- Typical methods detect object patches, then do binary segmentation within the patch

Summary

- Segmentation can be approached in various ways – clustering/graph-based methods vs. pixel classification with FCNs
- Advantages of FCNs
  - Better able to handle very complex objects/backgrounds
  - Likely to give better results for the classes on which they are trained
- Disadvantages of FCNs
  - Tend to be worse at capturing precise boundary details
  - May not generalise to classes outside their training set