Dumbshit guide to ZMO 🩻

Ustni zkouska

- kybic vylosoval dva tematicke celky (segmentace, registrace, deskriptory textur, detekce telisek, ...) , z kazdeho si clovek vybral jednu metodu o ktere mluvil

Classical segmentation methods

Active contours 🐍

- we place a shape $x(s) = [x(s), y(s)], s \in [0, 1]$ into an image and would like it to strech and latch onto notable features such as edges, additionally it should be smooth and continous
- the contour is described as $v(s, t)$, where s is arc length and t is time
  - **advantage** over simpler methods is that it can interpolate unclear edges
  - **disadvantage** is that this requires good initialization
- minimize the energy of the contour

$$E_{\text{total}} = E_{\text{img}} + E_{\text{contour}}$$

- where E_{img} is computed as smoothed gradient of the image
- **Contour energy**

$$E_{\text{contour}} = \alpha \left\| \frac{d\mathbf{v}}{ds} \right\|^2 + \beta \left\| \frac{d\mathbf{v}^2}{d^2 s} \right\|^2$$

  - where the first term is **elasticity**

$$\alpha \left\| \frac{d\mathbf{v}}{ds} \right\|^2 \approx ||(v_{i+1} - v_i)||^2 = (x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2$$

  - second term is **stiffness**

$$\begin{aligned} \beta \left\| \frac{d\mathbf{v}^2}{d^2 s} \right\|^2 &= ||(v_{i+1} - v_i) - (v_i - v_{i-1})||^2 \\ &= (x_{i+1} - 2x_i + x_{i-1})^2 + (y_{i+1} - 2y_i + y_{i-1})^2 \end{aligned}$$

- **Solving via gradient descent**

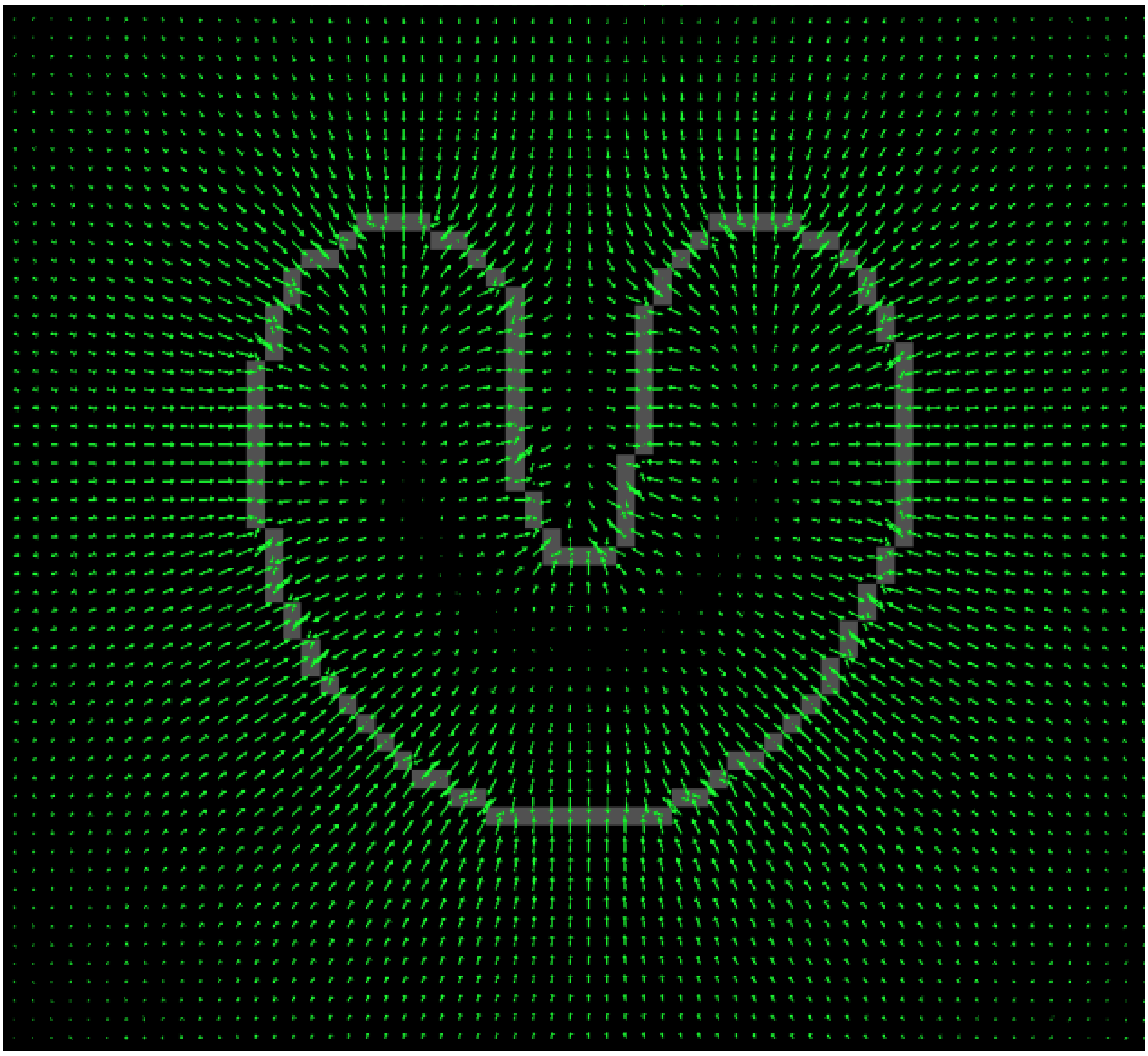
$$\frac{\partial v}{\partial t} = \alpha \frac{d\mathbf{v}^2}{ds^2} - \beta \frac{d\mathbf{v}^4}{d^4 s} + k f_E + \lambda f_B$$

  - f_E is an the external force
  - f_B is the ballon force
    - when we do not have information about whether the curve should grow or shrink we can add ballon force to the equation to control this
  - $\alpha, \beta, \kappa, \xi$ are hyper-parameters
- **Gradient vector flow (GVF)**
  - can be used as an alternative to ballon force
  - energy is high where there are edges, but the edge information is localized, with GVF we use smooth interpolation, obtaining a vector field, where each pixels points in the direction of higher gradients, using this we obtain this formulation

$$\mathbf{x}_t(s, t) = \alpha \mathbf{x}''(s, t) - \beta \mathbf{x}''''(s, t) + \mathbf{v}$$

    - gradient vector flow is defined as a vector field that minimizes the energy functional

$$\mathcal{E} = \int\int \mu(u_x^2 + u_y^2 + v_x^2 + v_y^2) + ||\nabla f||^2||\mathbf{v} - \nabla f||^2 \mathrm{dx\,dy}$$

    - again it is solved by gradient descent
    - 
- **Algorithm**
    - sample n contour points
    - for each contour point minimize the total energy by gradient descent
        - i.e. move it to a point in a small window around it in the direction given by the energy gradient
    - if the points haven't moved very much in the last iteration, we stop

Level sets

- the contour is represented as a mathematical function that evolves over time
- to evolve a curve C we compute a surface u such that C is level set of u and evolve u

Active shape models

- **Description**
    - have M images with a shape that is manually annotated with N landmarks in a vector $x^i = (x_1^1, y_1^1, \ldots, x_1^N, y_1^N)^T$, this forms a *PDM* (point distribution model) $x = \{x^1, \ldots, x^n\}$
    - in each image align the shape onto the reference shape
        - this gives us the mean shape and the variance for each points
    - use PCA to find a lower dimensional approximation for

$$x = \hat{x} + P_t b$$

    - where $\hat{x}$ is the mean shape, P_t is the matrix of eigenvectors from PCA and b are model parameters
    - each principal p_i component belongs to an eigenvalue λ_i and shows the direction in which the shapes vary the most in the dataset
- **Aligning the shapes**
  - assume that the coordinates for the moving shape are (x_i, y_i) and (x'_i, y'_i) for the reference shape
  - we already have the correspondences from the manual annotation, so we fit the shape onto the image using rotations R, translations t and scaling s
  - the weights can be used to prioritize important images or supress noisy images in the dataset or they don't have to be included at all
  - we minimize the sum of squared distances over all of the points analytically (set partial derivatives to zero, solve the linear equations)

$$\begin{aligned} E &= \sum_{i=1}^{M} w_i \|sR\mathbf{x} + t - \mathbf{x}'\|^2 \\ &= \sum_{i=1}^{M} w_i \left\| s \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} - \begin{bmatrix} x'_i \\ y'_i \end{bmatrix} \right\|^2 \end{aligned}$$

  - new shapes can be generated by varying the parameter b

Active appearance models

- ASM + linear appearance model

Superpixels

- small compat groups of similar pixels and of similar size defined as

$$C_k = [R_k, G_k, B_k, x_k, y_k]^T$$

- they can be used to replace the pixel grid, reduce image redundancy and speed up subsequent processing

Slic (simple linear iterative clustering)

- **Variables**
    - N - number of image pixels
    - K - num of superpixels
    - $\frac{N}{K}$ - average superpixel area
    - $S = \sqrt{\frac{N}{K}}$ - distance between clusters
- **Algorithm**
    - modified k-means
    - init cluster centers for each C_k
    - for each C_k
        - find similar pixels in $2S \times 2S$ neighbourhood
        - recompute centers
        - repeat until convergence
- distances between two superpixels i, j are computed as
    - $d_{RGB} = ||c_i - c_j||_2$, where $c = [R, G, B]$
    - $d_{xy} = ||v_i - v_j||_2$, where $v = [x, y]$
    - $D_s = \sqrt{d_{RGB}^2 + (\frac{m}{S} d_{xy})^2}$
    - where m is a parameter

Graph cuts

- used to refine a segmentation given an initial labeling
- we have a prediction y_p for each pixel p
  - this can be for example based on color clusters obtained via k-means
  - we connect each pixel to the foreground and background with a terminal edge that has a cost based on the function $R_p(y_p)$
- and we assign a cost to each edge between pixels with the smoothness term $B_{pq}(y_p, y_q)$, which should be low for similar pixels and large for dissimilar pixels
- then we solve the max flow / min cut problem to minimize the energy function

$$E(y) = \sum_{p \in P} R_p(L_p) + \sum_{(p,q) \in N} B_{pq}(y_p, y_q)$$

Normalized cuts

- compute cuts which are normalized by that area to avoid very small partitions
- weights are defined as a product between feature similarity and spatial proximity

$$w_{ij} = e^{\frac{-\|\boldsymbol{F}(i)-\boldsymbol{F}(j)\|_2^2}{\sigma_I}} \cdot \begin{cases} e^{\frac{-\|\boldsymbol{X}_{(i)}-\boldsymbol{X}_{(j)}\right|_2^2}{\sigma_X^2}} & \text{if } \|\boldsymbol{X}(i)-\boldsymbol{X}(j)\|_2 < r \\ 0 & \text{otherwise.} \end{cases}$$

- compute association as the sum of the edges of all vertices in V_a

$$assoc(V_A, V) = \sum_{u \in V_a, v \in V} w(u, v)$$

- then we compute a normalized cut as

$$\operatorname{NCut}(V_A, V_B) = \frac{\operatorname{cut}(V_A, V_B)}{\operatorname{assoc}(V_A, V)} + \frac{\operatorname{cut}(V_A, V_B)}{\operatorname{assoc}(V_B, V)}$$

- and minimize this instead, solving this exactly is NP-complete but appproximations are good enough
- the following problem is equivalent to minimizing the normalized cut

$$\min_x Ncut(x) = \min_y \frac{y^T(D-W)y}{y^T D y}$$

  - where D is the diagonal degree matrix (with degree being the sum of outgoing weights) and W is the adjacency weight matrix
- to which we obtain a solution by solving the eigenvalue system

$$(D-W)y = \lambda D y$$

  - the eigenvalues here can be reshaped to 2d matrices the size of the image to show the resulting segmentations

Random walker segmentation

- weights are defined based on pixel similarity for example (or as in normalized cuts)

$$w_{pq} = e^{-\beta|z_p - z_q|}$$

  - β is a parameter, which defines how easy / hard it is to move in the network
- given a mask for fg/bg pixels we want to compute the probability that a random walker from an unlabeld pixel ends up in labeled a node, this can be solved analytically by constructing the laplacian

matrix

$$L = D - W, \quad L_{ij} = \begin{cases} d_i & \text{if i=j, degree of node i} \\ -w_{ij} & \text{for adjacent vertices} \\ 0 & \text{else} \end{cases}$$

- in the random walker algorithm we want to solve the energy Q, where x is the vector of probabilities of the walker reaching foreground / background

$$Q(x) = x^T L x = \sum_{e_{ij}} w_{ij}(x_i - x_j)^2$$

- we divide the matrix L into four submatrices $L = \begin{bmatrix} L_{MM} & L_{MU} \\ L_{UM} & L_{UU} \end{bmatrix}$
  - L_{MM} - marked → marked nodes relationships
  - L_{MU} - releationship marked → unmarked
  - L_{UM} - releationship unmarked → marked
  - L_{UU} - unmarked → unmarked nodes relationships
- then this problem has a solution, which can be computed via the conjugated gradients method (a method for solving large sparse systems of equations)

$$L_{UU} x_U = -L_{UM} x_M$$

Texture description

Gabor filters

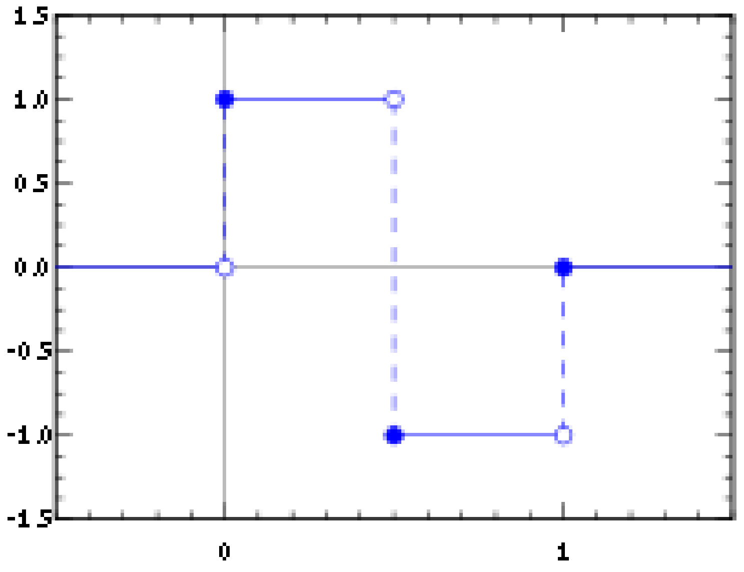
- gabor filters are composed of a sinusoidal and a gaussian filter

$$G_{\text{real}}[i,j] = \exp\left(-\frac{i^2+j^2}{2\sigma^2}\right)\cos(2\pi f(i\cos\theta + j\sin\theta)),$$
$$G_{\text{imag}}[i,j] = \exp\left(-\frac{i^2+j^2}{2\sigma^2}\right)\sin(2\pi f(i\cos\theta + j\sin\theta)),$$

  where σ is the standard deviation, f is the frequency, and θ is the rotation angle
- a collection of Gabor filters with different parameters form a filter bank
- which can then be used to compute the texture descriptors like

$$D = (I * G_{\text{real}})^2 + (I * G_{\text{imag}})^2$$

Haar Wavelets [o tomhle vim hovno]



- in higher dimensions these are binary filters composed of square functions (rectangles), to compute a descriptor we compute the correlation of the filter with an image, this corresponds to subtracting pixels from the region with -1 from the pixels in the region with 1

- **CWT (continous wavelet transform)**
  - two-scale relationship

$$h_{i+1}(k) = [h]_{\uparrow 2^i} * s_i(k)$$
$$g_{i+1}(k) = [g]_{\uparrow 2^i} * s_i(k)$$

- **Textons**
  - filter the image with filters from a filterbank (e.g. gabor filters, gaussian filters, DoG)
  - cluster the filter responses by k-means into a small set of prototype textons
    - the cluster centers are the textons we are searching for, we call these appearance vectors
  - textons can be converted back into image patches
    - for invertible filters this is easy
    - for non-orthogonal non-invertible filters set it up as a least squares problem
      - filter matrix F where each row is a flattened filter from the filter bank
      - obtain a local patch P_k as

$$P_k = F^+ c_k$$

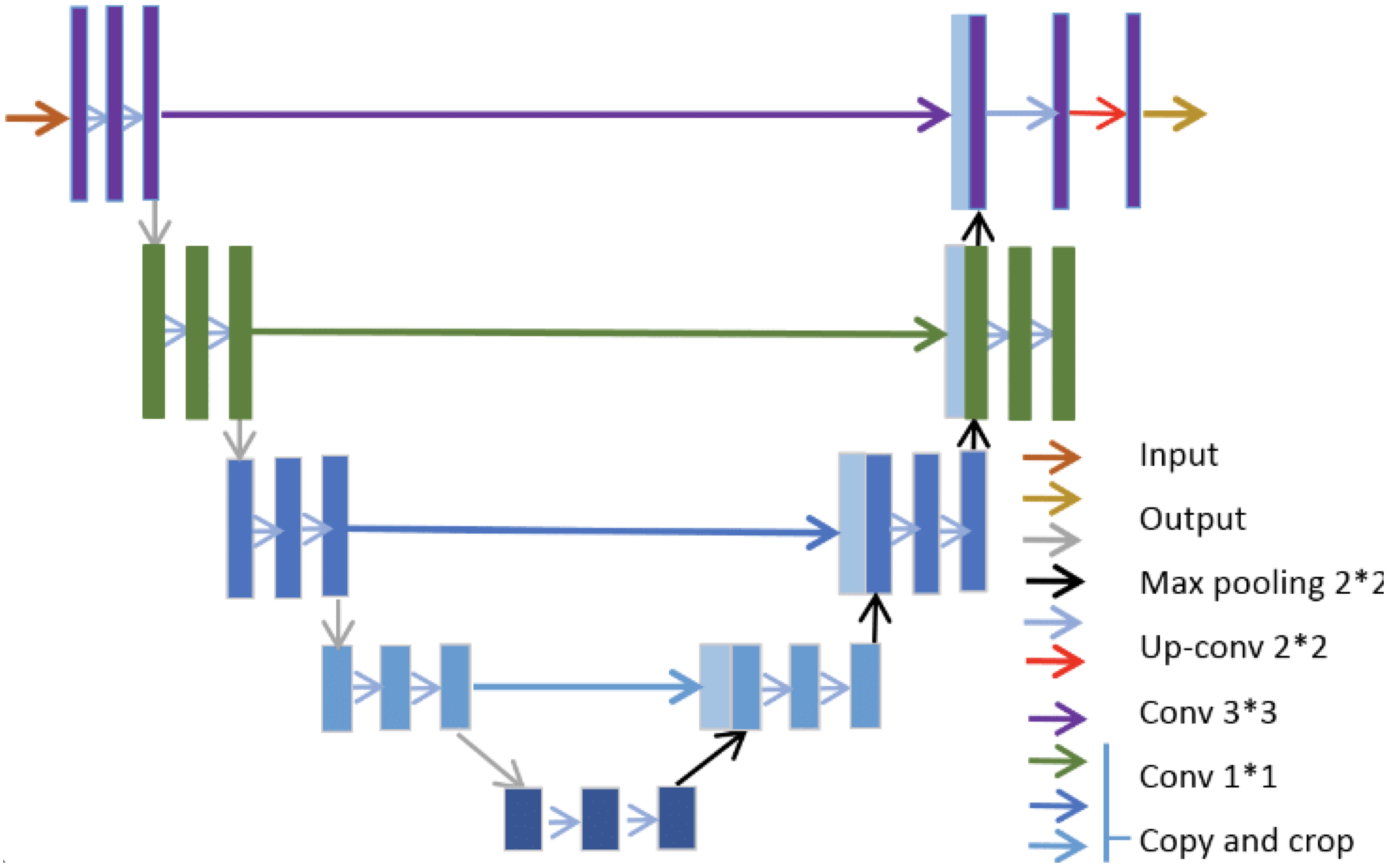
Deep learning methods

U-net

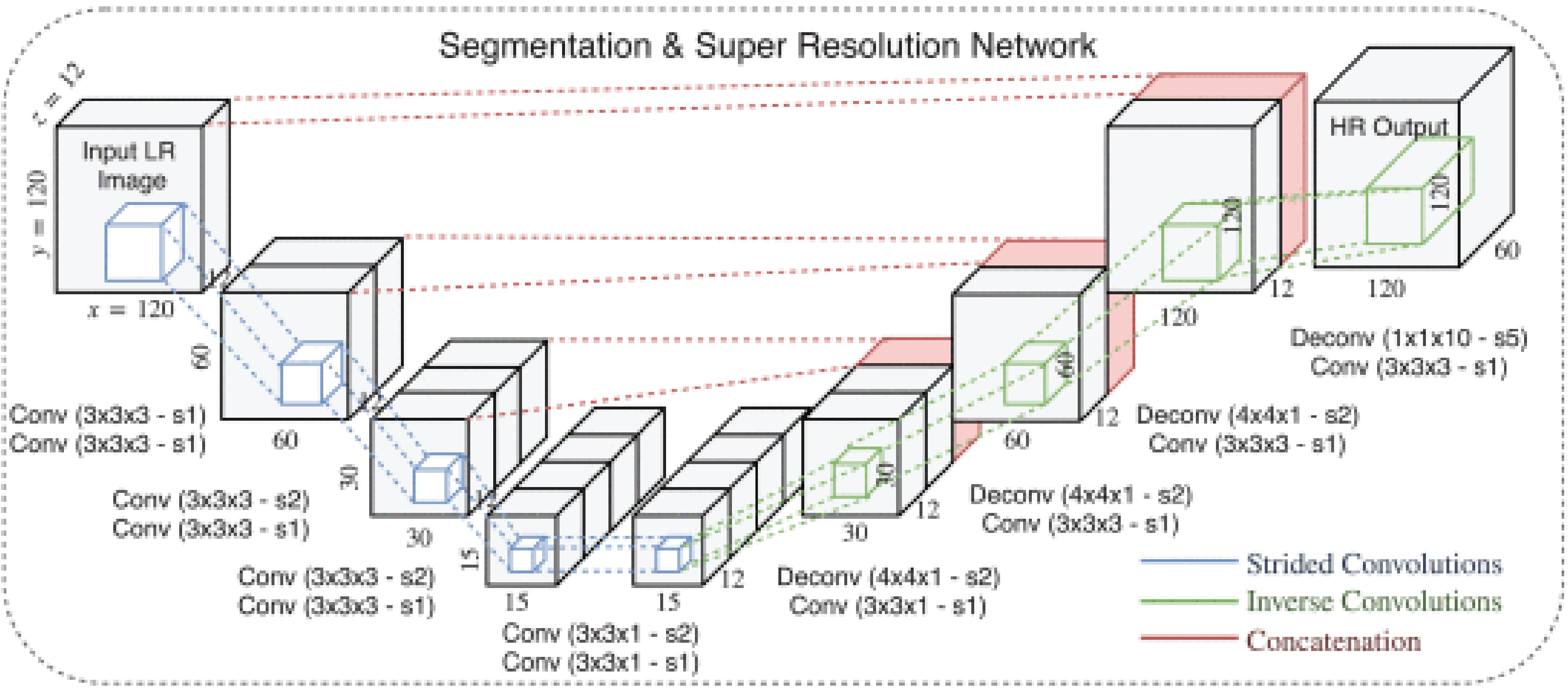
- uses and encoder and a decoder block with skip connections to compute the segmentation map without loosing information by propagating the inputs from the earlier layers
- its fully convolutional
- as a loss function we use cross entropy

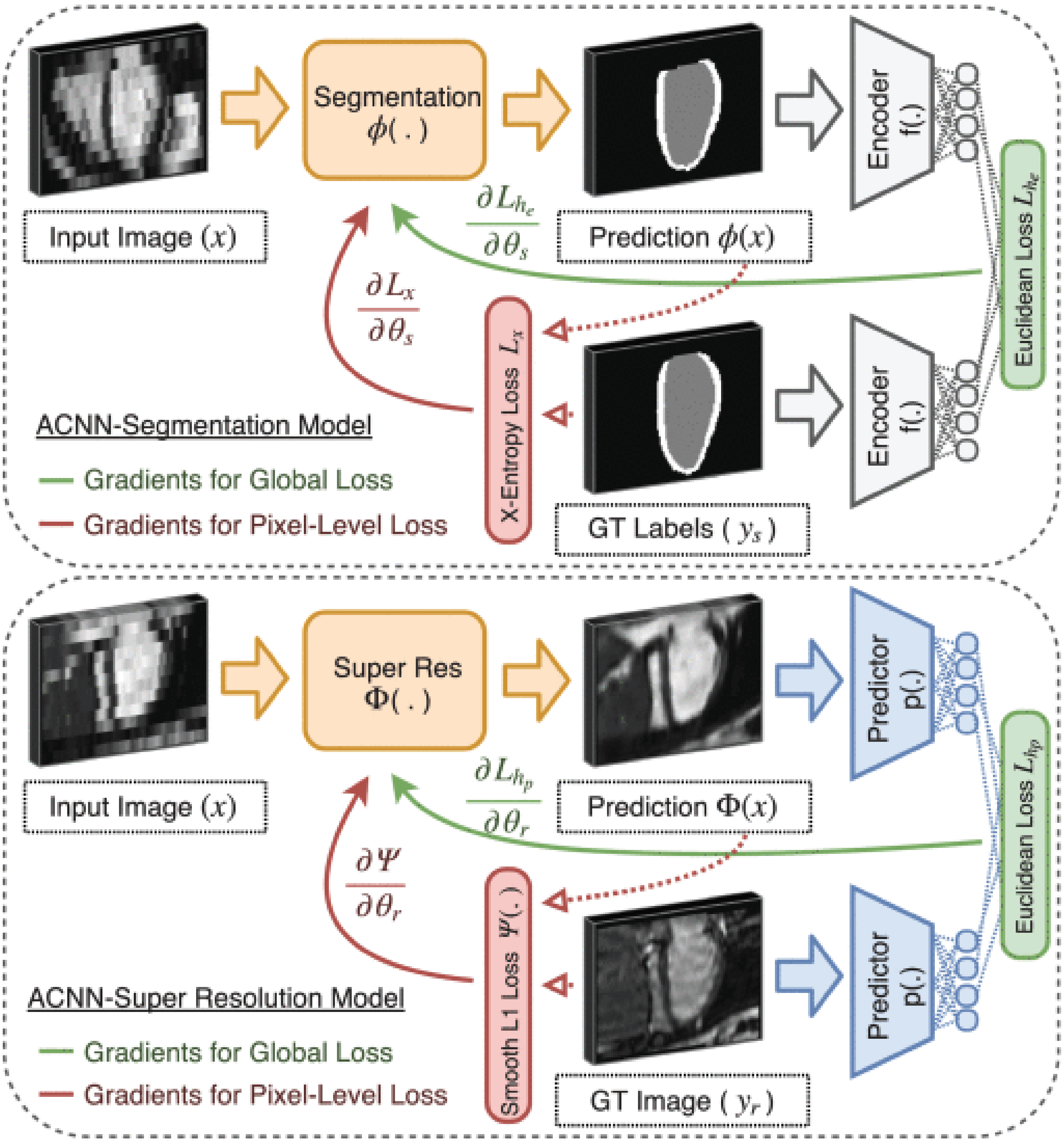
$$E = \sum_{x \in \Omega} w(x) \log(p_{l(x)}(x))$$

  - where w is a weight map to give some pixels more importance than other pixels

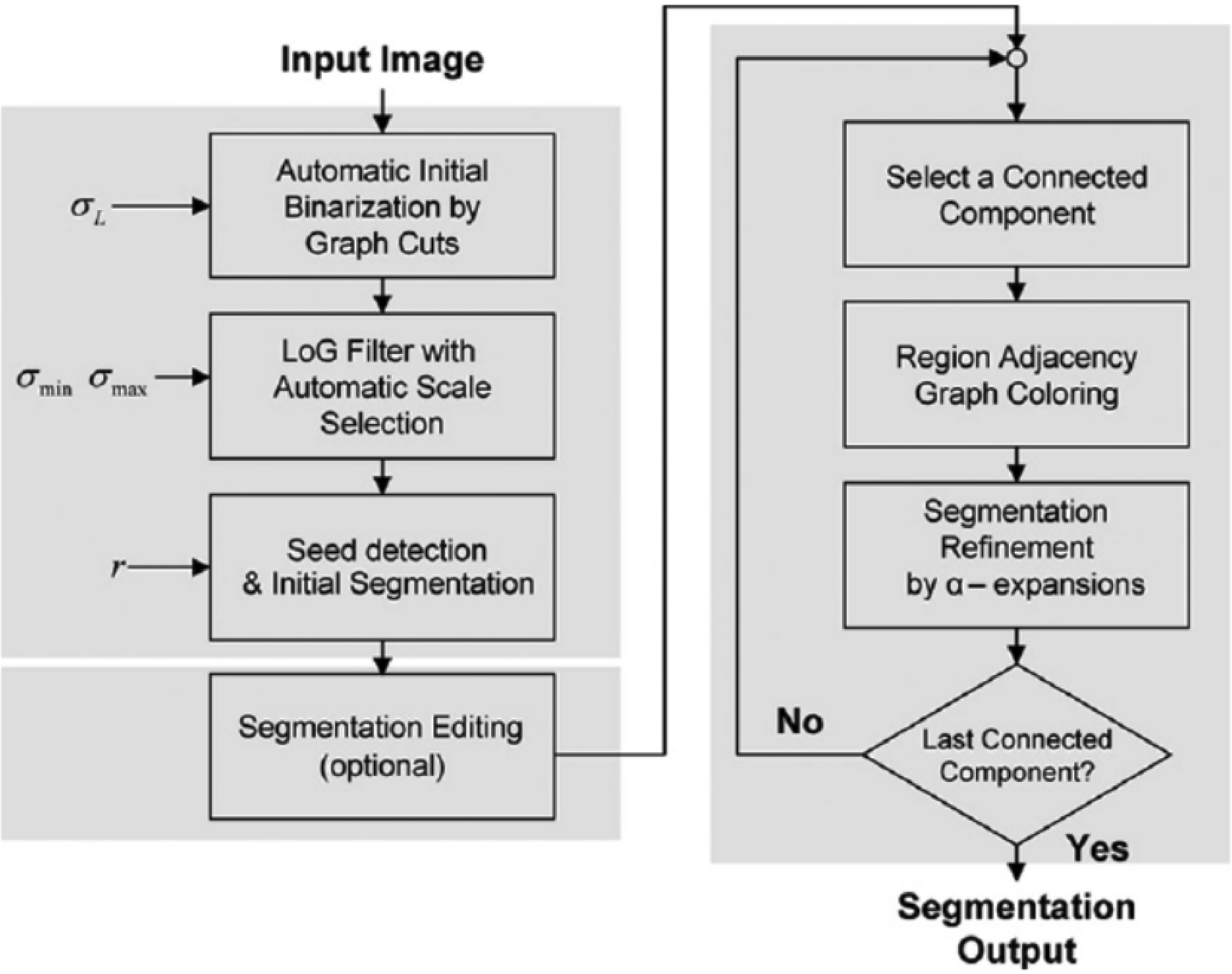
- 

Anatomically constrained neural network

- we add a prior to the network about the shape and size of the structures we are looking to segment
- two models are used
- **3D U-net** for segmentation / super resolution
  - 

- two losses are used
- **pixel-wise cross-entropy loss**
  - between ground truth labels and the resulting segmentation
- **global euclidian loss**
  - between the encoded feature vector from the predicted segmentation and the ground truth segmentation, these are extracted by an encoder block (?from an autoencoder pretrained on the labels?)
  - this is meant to increase the global accuracy of the structure
- 
Input Image (x)
Segmentation $\phi(\,.\,)$
Prediction $\phi(x)$
Encoder f(.)
Euclidean Loss L_{h_e}
$\frac{\partial L_{h_e}}{\partial \theta_s}$
$\frac{\partial L_x}{\partial \theta_s}$
X-Entropy Loss L_x
Encoder f(.)
GT Labels (y_s)
ACNN-Segmentation Model
Gradients for Global Loss
Gradients for Pixel-Level Loss
Input Image (x)
Super Res $\Phi(\,.\,)$
Prediction $\Phi(x)$
Predictor p(.)
Euclidean Loss L_{h_p}
$\frac{\partial L_{h_p}}{\partial \theta_r}$
$\frac{\partial \Psi}{\partial \theta_r}$
Smooth L1 Loss $\Psi(\,.\,)$
Predictor p(.)
GT Image (y_r)
ACNN-Super Resolution Model
Gradients for Global Loss
Gradients for Pixel-Level Loss

Improved automatic detection of cells and nuclei


Input Image
σ_L
Automatic Initial Binarization by Graph Cuts
$\sigma_{\min}$ $\sigma_{\max}$
LoG Filter with Automatic Scale Selection
r
Seed detection & Initial Segmentation
Segmentation Editing (optional)
Select a Connected Component
Region Adjacency Graph Coloring
Segmentation Refinement by α – expansions
No
Last Connected Component?
Yes
Segmentation Output

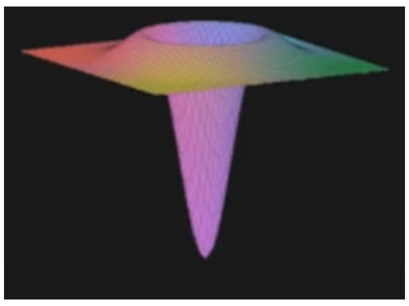
- **Binarization by graph cuts**
    - 128 bin histogram of the image
        - model the histogram with a mixture of poisson distribution, the histogram is expected to be bimodal (two maxima), e.g. one poisson distribution for the foreground and one for the background

$$h(i) = P_0 \cdot p(i|0) + P_1 \cdot p(i|1)$$

    - find threshold t which minimizes the KL divergence between the model and the true histogram by minimizing the criterion

$$\begin{aligned} t^* = \arg\min_t \{ & \\ & \mu \\ & - P_0(t)(\ln P_0(t) + \mu_0(t) \ln \mu_0(t)) \\ & - P_1(t)(\ln P_1(t) + \mu_1(t) \ln \mu_1(t)) \\ \} & \end{aligned}$$

    - we split the histogram by t^* and refine the result by graph cuts
- **Laplacian of Gaussian with a scalespace**
    - LoG filter is specialized to detect blobs (seeds)
    - by running it on the image with an increasing level of blur we can detect blobs of different size
    - this way we can accurately detect the seeds as well as the scale at which they are occuring

    - 
    - detect initial seeds
        - LoG response computed over a scale space $[\sigma_{min}, \ldots, \sigma_{max}]$
        - use euclidian distance map $D_N(x, y)$ to compute a single response surface as

$$R_N(x,y) = \underset{\sigma \in [\sigma_{\min}, \sigma_{\text{MAX}}]}{\arg\max} \left\{ \text{LoG}_{\text{norm}}(x, y; \sigma) * I_N(x, y) \right\}$$

            - where $\sigma_{\text{MAX}} = \max\{\sigma_{\min}, \min\{\sigma_{\max}, 2 \times D_N(x, y)\}\}$
            - Distance map $D(x, y)$ is the same thing as in the watershed algorithm, the euclidian distance of pixel at x, y from the nearest object
- **Segmentation refinement (α expansions and graph coloring)**
    - we want to find a multi-way cut, this is NP hard but we can solve it approximately via alpha expansions
    - iteratively, we simplify the problem to a two-class cut by taking one cell as α class and the rest as $\hat{\alpha}$ class and run this until convergence
    - for large number of cells this is impractical, so they use an approach based on graph coloring instead which assigns colors to the vertices of a graph in such a way that no two adjacent vertices share the same color
    - then alpha expansion is used and in each step the nodes with the same color are assigned class α and the others are $\hat{\alpha}$

Locality Sensitive Deep Learning for Detection and Classification of Nuclei in Routine Colon Cancer Histology Images

- **Nucleus detection**
    - a sliding window of size $H \times W$ is fed into a CNN, where the label map is defined as sharp peaks around the annotated nuclei centers

$$\hat{y}_j = \begin{cases} \frac{1}{1 + \left(\|\mathbf{z}_j - \mathbf{z}_m^0\|_2^2\right)/2}, \text{if within } d \text{ around nucleus center } z_m^0 \\ 0, \text{otherwise} \end{cases}$$

    - the output of the network is a parameter estimation layer $S1$, that estimates coordinates of the nuclei center $z_m = (u_m, v_m)$ and the probability of being nuclei h_m, predicted label map is then computed from the $S2$ layer as

$$\hat{y}_j = \begin{cases} \frac{h_m}{1 + \left(\|\mathbf{z}_j - \mathbf{z}_m^0\|_2^2\right)/2}, \text{if } z_m^0 \text{ is not within } d \text{ of another pred } z_{m'}^0 \\ 0, \text{otherwise} \end{cases}$$

    - the size of the $S1$ layer determines the range of nuclei centers predicted from one window as $[0, M]$
- **Nucleus classification**
    - patch based multiclass classifier (different nuclei types), take several windows with the predicted nucleus and consider the ensemble of those predictions, they found that this works better, than single window predictor

---

Detection of vessels and fibers

- **Frangi vesselness filter**
  - a filter to find tubular structures based on the second derivatives of the image, similar idea as in Harris corner detector but we are interested in different structure
  - approximate the image by a taylor expansion with dI_x, dI_y
  - for each pixel compute the hessian matrix as $H = \begin{bmatrix} dIx^2 & dIx \cdot dIy \\ dIy \cdot dIx & dIy^2 \end{bmatrix}$ and obtain its eigenvalues to get how the image changes in the x and y directions at that pixel from those we compute two metrics

$$R = \frac{|\lambda_1|}{|\lambda_2|}, \\ S = \lambda_1^2 + \lambda_2^2,$$

  - and from those compute the vesselness measure

$$V = \begin{cases} 0 & \text{if } \lambda_2 <= 0, \\ \exp(-\frac{R}{2B^2})(1 - \exp(-\frac{S}{2C^2})) & \text{otherwise} \end{cases}$$

  - for 2D, where B and C are parameters
  - for 3D we have three eigenvalues and a corresponding extended veselness measure

---

Detection of nodules using local image features and sequential k-nn classifiers

- subsample image 512x512 $\rightarrow$ 256x256
- initial seeds are calculated by thresholding SI and CV and expanded into clusters by hysteresis thresholding
  - **shape index**
    - $\mathrm{SI} = \frac{2}{\pi} \arctan\left(\frac{k_1 + k_2}{k_1 - k_2}\right)$
  - **curvedness**
    - $\mathrm{CV} = \sqrt{k_1^2 + k_2^2}$
    - where k_1 and k_2 are the principal curvatures calculated as the first and second order derivatives of the image blurred with a gaussian filter with scale $\sigma = 1$
- recursively merge clusters within
  - 3 voxels until no more can be merged
  - 7 voxels until no more can be merged
- adjust candidate locations so that they sit at the brightest local spot
- **false positive reduction with sequential knn classifiers**
- Sequential Forward Floating Selection (SFFS, leave one out) for feature selection
- first knn classifier
  - 18 hand designed features
  - merge neighbouring clusters
  - readjust clusters to the brightest local spots again
- second knn classifier
  - 135 hand designed features (with 18 coming from the first step)
  - posterior probability of a nodule being a structure as calculated by the first classifier
- the advantage of using two sequential classifiers is speed

Detection of nodules Using Multi-View Convolutional Networks

- in 2016 when this paper was released processing 3d scans directly would be too costly, so they opted to use a ensemble-like approach, feed different views into the network and combine the decisions for each one into the final prediction
- **method**

- detect suspicious lesions via some other algorithms
    - for solid nodules they used the previous method
- apply the same CNN network to each view
- obtain a 16-dim embedding
- fuse the embeddings via different **fusion operators** to obtain predictions
    - single view
        - concat all and pass trough FC+softmax
    - committee-fusion
        - pass each view trough FC+softmax and take majority vote
    - late-fusion
        - concat all and pass through FC+softmax
    - mixed fusion
        - concat 3 views, pass each through FC+softmax and take majority vote

Large scale detection of mamographic lesions with deep learning [TODO]

-

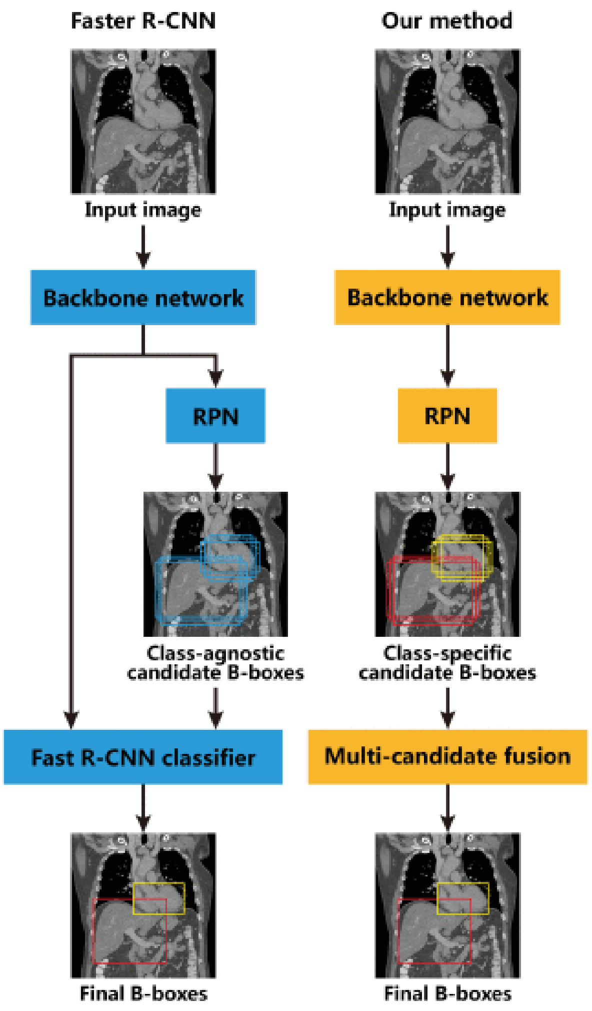
Localization of organs and structures

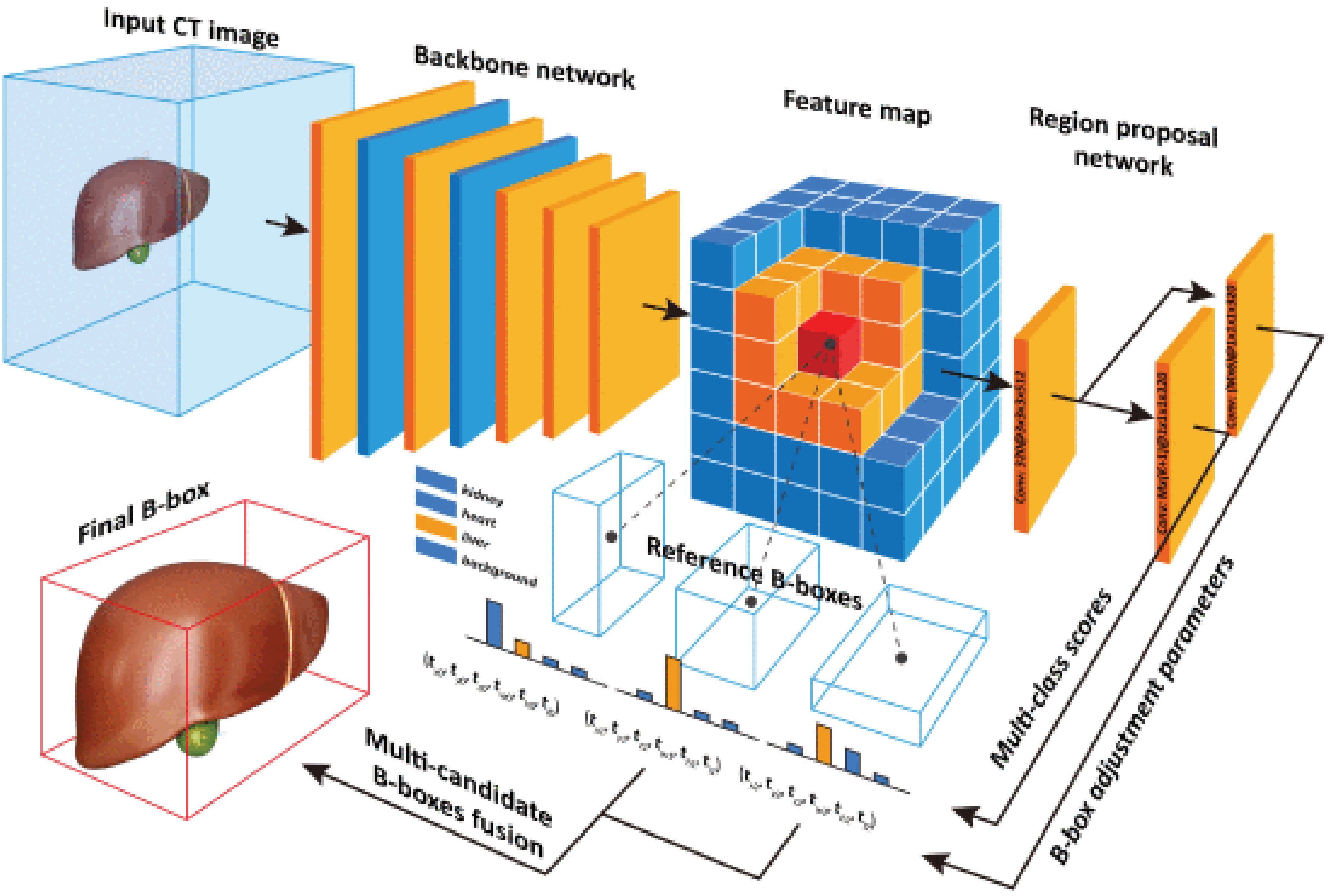
Sequential estimation and integrated detection network

- used for detecting and measuring structures in fetal brain
- inspired by sequential tracking
- input is a 3D ultrasound volume
- **Object detection by importance sampling**
- **Probabilistic boosting tree (PBT)**
    - is used to decide between a positive and a negative example of an object
    - PBT combines a binary decision tree with boosting, letting each tree node be an AdaBoost classifier
- **Integrated Detection Network (IDN)**
    - The IDN is used to combine the predictions from multiple observation and transition models, which helps to improve the detection performance

Region proposal network

- inspired by R-CNN to predict a bounding box around organs

- 

- backbone extracts a feature map, region proposal network (RPN) generates candidate B-boxes for what could be foreground objects and these are then classified by the fast R-CNN classifer and NMS (non-maximum suppression) is applied
- in this paper it is argued that each organ is present at most once in a CT scan and thus the network can be simpler and the region proposal network can generate class specific boxes directly (which are the NMSd) this method is able to run in 3D
- 
Input CT image
Backbone network
Feature map
Region proposal network
Final B-box
kidney
heart
liver
background
Reference B-boxes
$(t_x, t_y, t_z, t_w, t_h, t_d)$
$(t_x, t_y, t_z, t_w, t_h, t_d)$
$(t_x, t_y, t_z, t_w, t_h, t_d)$
Multi-candidate B-boxes fusion
Multi-class scores
B-box adjustment parameters

Registration

- **rigid registration** : only rotations, translations, scaling
- **non-rigid registration** : allows local deformations and warping

ICP (iterative closest point registration)

- an algorithm used for aligning point clouds (image stitching) with rigid registration
- given two sets of point clouds P_t and P_{t+1}
- while not converged or max iterations reached
  - obtain correspondence clusters $c_i = \{(x_j, y_k), \ldots\}$ by assigning each $y_j \in P_{t+1}$ to the closest $x_k \in P_t$
  - closest points can be found quickly via k-D trees, with that we gain a speedup of $O(N_p N_x) \rightarrow O(N_p \log N_x)$
  - translate P_{t+1} so that the centroids (centers of mass) for each correspondence cluster c_i match

$$\bar{x_i} = \bar{y}_i \quad \forall i$$

  - find a rotation R that rotates P_{t+1} such that the total distance for all the correspondences (x_j, y_k) is minimised
  - optimal rotation matrix can be found by solving the following least squares problem

$$\underset{\mathbf{R}^\top\mathbf{R}=\mathbf{I}, |\mathbf{R}|=1}{\arg\min} \sum_i \|\mathbf{R}\mathbf{x}_i - \mathbf{y}_i + \mathbf{t}\|^2$$

  - where the translation is simple and done in the first step
  - and the rotation is found via computing the point covariance matrix $\mathbf{M} = \mathbf{X}^\top\mathbf{Y}$
  - where $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{3\times n}$
  - and then transforming the first problem into

$$\underset{\mathbf{R}^\top\mathbf{R}=\mathbf{I}, |\mathbf{R}|=1}{\arg\max} \quad \mathrm{tr}(\mathbf{M}\mathbf{R})$$

  - and solving by taking the SVD of as $\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ and $\mathbf{R} = \mathbf{V}\mathbf{U}^\top$

  Coherent point drift

- consider the registration as probability density estimation problem where one set of points is a GMM (gaussian mixture model) and the other are data points, this is more robust and resistant to noice than ICP
- uses the **EM-algorithm** to align the datapoints to the GMM model
  - **E-step**: Compute the posterior probabilities (P) for each data point belonging to each Gaussian component in the GMM. Functions as a soft-correspondence assignment.
  - **M-step**: Update the GMM parameters by solving for the non-rigid transformation (W) that aligns the Gaussian centroids to the data points, based on the computed posterior probabilities. This involves solving a linear system of equations derived from the energy function that incorporates the motion coherence constraint
- **Annealing**: Reduce σ^2 according to a predetermined annealing schedule, allowing the algorithm to transition from a coarse to a fine match strategy
- **Velocity Field Computation**: After convergence, compute the velocity field (v(z)) that represents the underlying non-rigid transformation. The velocity field is defined in terms of the Gaussian kernel weights (W) and the Gaussian kernel function (G)

Splines

- a function $s(x)$ is a polynomial spline of degree n with knots if it satisfies these two properties
  - **piecewise polynomial**
    - $s(x)$ is a polynomial function of degree n in each of its segment $[x_k, x_{k+1}]$
  - **higher order continuity**
    - continous in all derivatives of degrees $[0, \ldots, n-1]$ at the knots
- **B-splines (basis splines)**
  - zero degree b-spline is a step function, b-splines of a degree n can be obtained by convolving $n+1$ zero degree b-splines
  - B-spline is a spline that has minimal support (w.r.t. a given degree and smoothness)

        - minimal support means that a minimum number of non-zero basis functions that contribute to a particular B-spline curve or surface at any given parameter value
    - a signal can than be represented as a linear combination of shifted cubic b-splines as

$$s(x) = \sum_{k \in \mathbb{Z}} c[k] \beta_+^n (x - k)$$

    - this can be extended to multiple dimensions

Optimization of mutual information for multiresolution image registation

- a mutual information criterion is used for aligning the images
- the images are modeled with parzen windows using cubic b-splines as the kernel function
- the mutual information is expressed with is taylor expansion and terms above second order are ingored
- marquardt-levenberg strategy is used together with a hessian approximated from first-order derivatives to minimise the criterion
- a multiresolution strategy is used to improve robustness and speed a different quantization of the gray levels occurs at each level of the pyramid
- **mutual information**
    - $H(X, Y) = H(X) + H(Y) - H(X, Y)$

Fast parametric image registration

- use b-splines to describe the image and the deformations a double multiresolution measure and a scalar pixel-based difference measure
- given two images f_r and f_t
- find a geometric model $g(x)$ that minimizes the sum of squared differences

$$E = \frac{1}{||I||} \sum_{i \in I} (f_t(g(i)) - f_r(i))^2$$

  between the images, the full geometric model is made with b-splines as

$$g(\mathbf{x}) = \mathbf{x} + \sum_{j \in I_c \subset \mathbb{Z}^N} c_j \beta_{n_m} (\mathbf{x}/\mathbf{h} - \mathbf{j})$$

- and the function can be optimized via gradient descent or the Marquardt–Levenberg algorithm

$$x_{k+1} = x_k - \frac{g'(x_k)^T g(x_k)}{g'(x_k)^T g'(x_k)^T + \mu_k I}$$

Image matching as diffusion (maxwell demons)

- non-rigid, no global deformation model
- inspired by the second law of thermodynamics -> the entropy of a closed system never decreases and is headed to its maximum
- **algorithm**
    - extract demons from the image and find normals (gradients)
    - compute demon forces and update the model
- can suffer from bad initialization (shapes dont overlap) and get trapped in local minima

Optical flow method

- **Optical flow**

    - for each pixel compute its movement vector, the direction and speed with which it moved between the two images
    - assumes
        - color constancy
        - small motion
- **Lucas-Kanade**
    - local method, constant flow
    - compute the gradints, build an overdetermined system of equations involving the gradients, the velocity vector and the time gradient and sole for the velocity vector with least squares, this is done for each pixel

$$[I_x \; I_y] v = -I_t$$

- **Horn-Schnuck**
    - global method, flow can vary for different pixels
    - minimize smoothness and brightness constancy
- **Lucas/Kanade Meets Horn/Schunck: Combining Local and Global Optic Flow Methods**
    - **Spatial smoothing**
        - This component is based on the Lucas-Kanade method, which uses a local energy-like expression and spatial constancy assumptions on the optic flow field
    - **Spatiotemporal smoothing**
        - This component is based on the Horn/Schunck method, which supplements the optic flow constraint with a regularizing smoothness term

Diffeomorphic registration

- **diffeomorphic** : invertible and differentiable with differentiable inverse

---

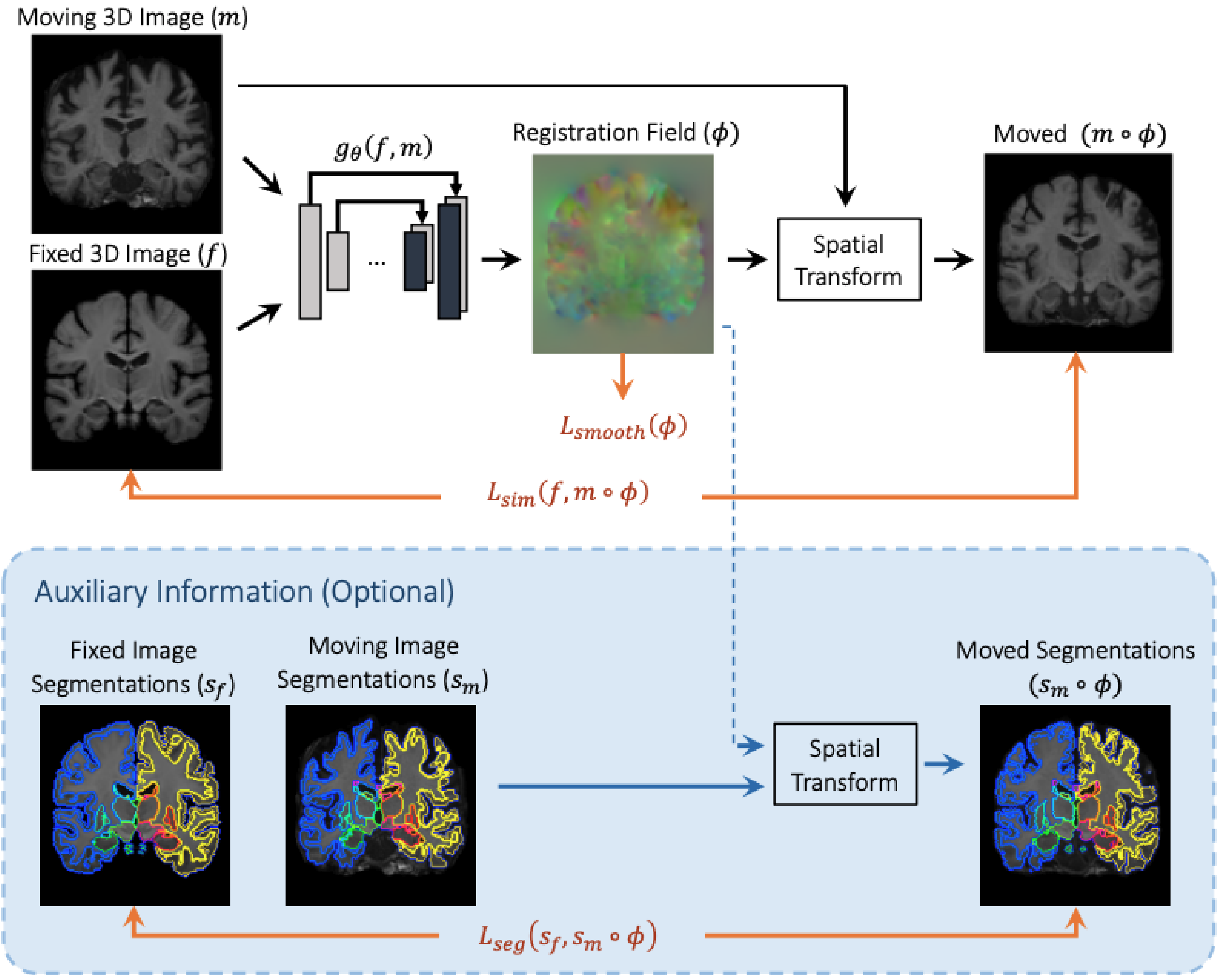
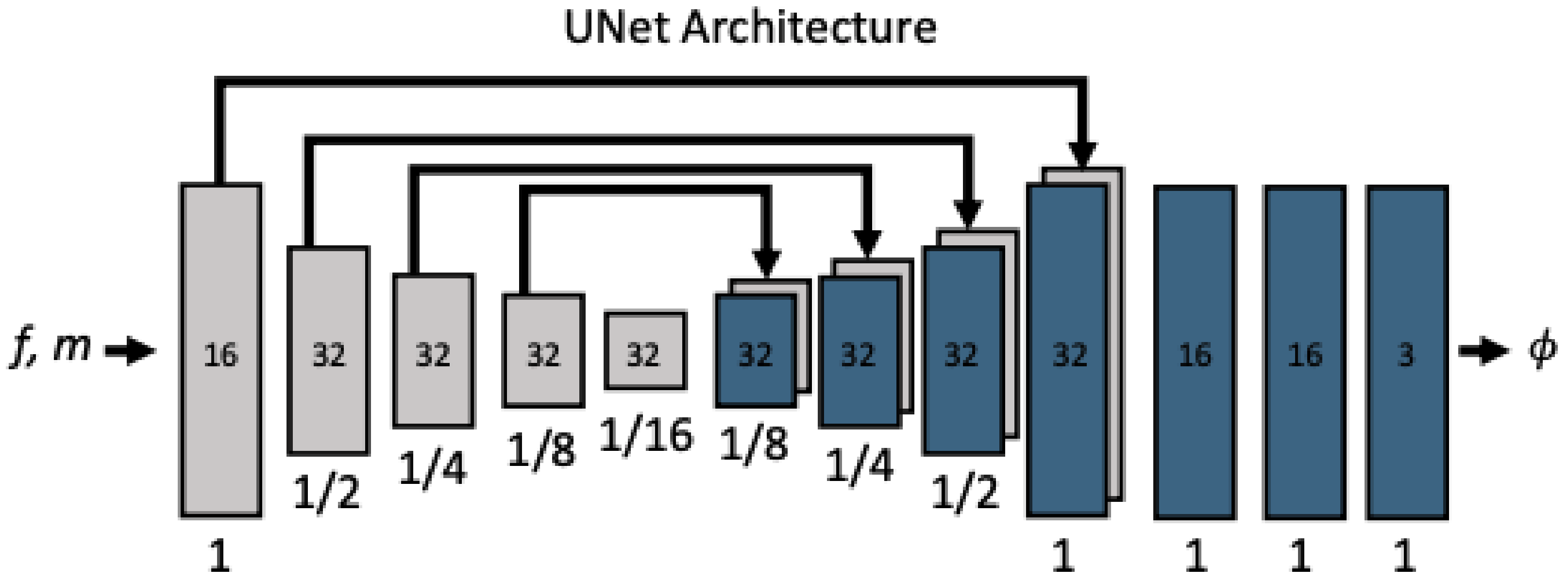
Deep learning for registration

VoxelMorph

- **Architecture**
    - use registred images for training, they are concatenated and fed into the network
    - the network predicts a registration field ϕ that transforms one image m onto another image f via $m \circ \phi$
    - transformations are computed with a sub-pixel accuracy via a displacement u as $p' = p + u(p)$ and linearly interpolated
- **Objective**

$$\hat{\phi} = \arg\min_{\phi} L_{\text{sim}}(f, m \circ \phi) + \lambda L_{\text{smooth}}(\phi)$$

    - where L_{sim} can be cross correlation or sum of squared differences
    - and $L_{\text{smooth}} = \sum_{p \in \Omega} ||\nabla u(p)^2||$, this should ensure a realistic smooth deformation fi

- 
Moving 3D Image (m)
Fixed 3D Image (f)
$g_\theta(f, m)$
Registration Field (ϕ)
Spatial Transform
Moved ($m \circ \phi$)
$L_{smooth}(\phi)$
$L_{sim}(f, m \circ \phi)$
Auxiliary Information (Optional)
Fixed Image Segmentations (s_f)
Moving Image Segmentations (s_m)
Spatial Transform
Moved Segmentations ($s_m \circ \phi$)
$L_{seg}(s_f, s_m \circ \phi)$
- 

- **Registration comparision**
  - **IOU (intersection over union, Jaccard index)**

    $$J = \frac{|X \cap Y|}{|X \cup Y|}$$

  - **Dice score**

    $$D = \frac{2|X \cap Y|}{|X| + |Y|}$$

    - related to Jaccard index via $J = \frac{D}{2-D}$

Quicksilver

-