Segmentation-2
Classification, Segmentation, Labeling and Modeling: Modern Approaches

Jerry L. Prince
Image Analysis and Communications Laboratory
Johns Hopkins University

Slide deck structure and many slides taken from Russell H. Taylor “Segmentation and Modeling” slides.
Segmentation Variants

• Image segmentation: divide an image into regions
• Voxel classification: cluster voxel intensities by type
• Voxel labeling: identify voxel anatomy/type by name
Classes of Segmentation Approaches

• Voxel-based versus boundary-based:
  – Voxel-based: voxels are labeled (object vs. background)
  – Boundary-based: boundaries are identified

• Supervised versus unsupervised:
  – Supervised: examples are provided
  – Unsupervised: the data itself provides all information

• Automatic versus semi-automatic
  – Automatic: no user information provided
  – Semi-automatic: some user information is provided
Topics Today

• Support Vector Machines (supervised classification)
• Random walker algorithm (manual-assisted segmentation)
• Graph cuts and Markov random fields (segmentation)
• Multi-atlas segmentation (registration-based segmentation)
• Deep neural networks (segmentation)
SUPPORT VECTOR MACHINES
Linear classification means that decision boundaries are linear.
General Linear Boundaries

- Divide space into labeled regions
- Piecewise linear boundaries
- Linear in any monotone transformation
- Quadratic boundaries

Credit: Hastie, 2001
Linear Discriminant Analysis (LDA)

- Predictor $G(x)$ takes on values $1, \ldots, K$
- Find discriminant function $\delta_k(x)$ and
- $G(x) = \arg\max_k \delta_k(x)$

- Linear discriminant analysis
  - Each class is multivariate Gaussian with the same covariance matrix
  - Log ratio between probabilities yields linear boundaries
Quadratic Boundaries

- Use LDA in 5-dimensional space: $x_1, x_2, x_1x_2, x_1^2, x_2^2$
  - Linear in 5-D, quadratic in original space

- Assume covariance matrices are different
  - Leads naturally to quadratic boundaries
  - Quadratic discriminant analysis = QDA
Dimensionality Reduction

• With K centroids, the linear decision rule can be made in (K-1)-dimensional space

• A correct projection of the N-D data points into K-1 dimensions is needed

• Consideration of the prior probabilities can be made in the (K-1)-D space

Credit: Hastie, 2001
Separating Hyperplanes

- Previous focus modeled all the points in each class
- Focus now on the points near the decision boundary
- Principle: A large margin on the training data will lead to good separation of the training data
- Maximize distance $C$ from a line that separates the data
- Depends on fact that such a line exists

Credit: Hastie, 2001
Support Vector Classifier

- Permit some points—support points—to be on the “wrong side” of the margin
- Tuning parameter (tolerance) $\gamma$ chosen small yields a larger margin involving points far away
- Defined still as a linear classifier

Credit: Hastie, 2001
Support Vector Machine

- Move from linear classifiers to higher-order using transformations via basis functions
  \[ h(x_i) = (h_1(x_i), h_2(x_i), \ldots, h_M(x_i)) \]

- Replacing these in the support vector classifier yields functions that depend only on
  \[ K(x, x') = \langle h(x), h(x') \rangle \]

- Never need to compute \( h(x_i) \) at all
  - So-called “kernel trick”

- Popular kernels:
  - \( d \) degree polynomial
    \[ K(x, x') = (1 + \langle x, x' \rangle)^d \]
  - Radial basis function
    \[ K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{c}\right) \]
  - Neural network
    \[ K(x, x') = \tanh(\kappa_1 \langle x, x' \rangle + \kappa_2) \]
  - Large \( \gamma \) leads to squiggly boundary (and overfitting)
  - Estimate \( \gamma \) by cross-validation
SVM Example

SVM Boundary

Bayesian Boundary

Extend to multiclass problems by solving many two-class problems and choosing the winner

Credit: Hastie, 2001
RANDOM WALKER ALGORITHM
Random Walks

- Choose seed pixels with labels $\ell_1, \ldots, \ell_K$
Random Walks

• Choose seed pixels with labels \( \ell_1, \ldots, \ell_K \)
• Define edge probabilities as
  \[ w_{ij} = \exp\left( -\beta (f(p_i) - f(p_j))^2 \right) \]
  where \( f \) is image intensity
  – Note: \( 0 \leq w_{ij} \leq 1 \)

\( x_{ij} \) small on edge

\( \odot \) Jerry L. Prince

Grady, 2006
Random Walks

- Choose seed pixels with labels $\ell_1, \ldots, \ell_K$
- Define edge probabilities as
  \[ w_{ij} = \exp\left(-\beta(f(p_i) - f(p_j))^2\right) \]
  where $f$ is image intensity
  - Note: $0 \leq w_{ij} \leq 1$
- Compute for each nonseed pixel the probability that a random walk reaches seed $\ell_k$ first
- Segmentation is the label with the highest probability

Grady, 2006
The Dirichlet Problem

- Probabilities are found by finding \( x \) (all pixel probabilities) that minimizes

\[
D(x) = \frac{1}{2} \sum_{e_{ij} \in E} w_{ij} (x_i - x_j)^2 = \frac{1}{2} x L x
\]

- Where \( L \) is the combinatorial Laplacian matrix

\[
L_{ij} = \begin{cases} 
  d_i, & \text{if } i = j \\
  -w_{ij}, & \text{if } v_i \text{ and } v_j \text{ are adjacent nodes} \\
  0, & \text{otherwise}
\end{cases}
\]

- Where \( d_i \) is the sum of all edge weights incident on pixel \( i \)
Solving the Dirichlet Problem

• Seed pixels are in $V_M$. Unmarked pixels are in $V_U$.
• Seeds may have different labels

Then

$$D(x_U) = \frac{1}{2} \begin{bmatrix} x_T^M & x_T^U \end{bmatrix} \begin{bmatrix} L_M & B \\ B^T & L_U \end{bmatrix} \begin{bmatrix} x_M \\ x_U \end{bmatrix}$$

$$= \frac{1}{2} (x_T^M L_M x_M + 2x_T^U B_T x_M + x_T^U L_U x_U)$$

• Differentiate with respect to $x_U$ and set equal to zero, leading to

$$L_U x_U = -B^T x_M$$

• This amounts to solving a modified Laplace’s equation with boundary conditions (the seed potentials)
Multiple Labels

- The label $s$ of seed $v_j$ is given by $Q(v_j)$
- Define
  
  $$m_j^s = \begin{cases} 
  1 & \text{if } Q(v_j) = s, \\
  0 & \text{if } Q(v_j) \neq s. 
  \end{cases}$$
- The probabilities for label $s$ can then be found by solving
  
  $$L_U x^s = -B^T m^s$$

- Repeat for all labels
- Assigned label at a given pixel is
  
  $$\hat{l} = \arg \max_s x^s$$
GRAPH CUTS SEGMENTATION
Graph Cuts Framework

- Pixels are nodes in a graph
- Pixels are connected within image
  - Neighbor links (n-links)
- Pixels connect to source S and sink T
  - Terminal links (t-links)
- Segmentation is produced by a graph cut
- Separates s and t

Boykov et al., 2001
Principle of Max-Flow Min-Cut

- Consider a flow network from $S$ to $T$ with capacities on the directed edges
- What is the maximum flow out of $S$ (and into $T$)?
Principle of Max-Flow Min-Cut

• Consider a flow network from S to T with capacities on the directed edges
• What is the maximum flow out of S (and into T)?
• It is equal to the sum of the edge weights (capacities) in the minimum cut
• (A “cut” separates the source from the sink)
• This problem can be solved exactly and rapidly using linear programming
Capacities on t-links

- Water flows from $S$ to $T$
- Each pixel has:
  - $\text{Prob}(\text{foreground})$
  - $\text{Prob}(\text{background})$
- $S$ t-links get capacities:
  - $R(p, S) = -\ln \text{Prob}(f)$
- $T$ t-links get capacities:
  - $R(p, T) = -\ln \text{Prob}(b)$
- S to background flows easily
- Foreground to T flows easily

Water has difficulty flowing to the foreground

Water has difficulty flowing from the background
Foreground/Background Probabilities

- Could use memberships from fuzzy K-means posterior densities from EM method
- Could use fuzzy segmentation with manual assist

\[ \Gamma(p) = \frac{1}{1 + \beta|A - B|} \]
Capacities on n-links?

- We want to cut at boundaries \(\Rightarrow\) capacity should be small on boundaries.
- Let \(f(p)\) be a “feature” at pixel \(p\) (could be intensity).
- Suitable boundary capacities are

\[
B_{p,q} = \exp \left[ -\frac{\|f(p) - f(q)\|^2}{2\sigma^2} \right] \cdot \frac{1}{\|p - q\|}
\]
What does the min cut do?

- Find cut $C$ separating $S$ and $T$ with minimum edge cost

\[ E(C) = \lambda \left( \sum_{p \in S_c} R(p, S) + \sum_{p \in T_c} R(p, T) \right) + \sum_{(p,q) \in N_c} B_{p,q} \]

- $S_c$ are the edges in the $S$ t-links that are cut by $C$
- $T_c$ are the edges in the $T$ t-links that are cut by $C$
- $N_c$ are the edges in the $N$ n-links that are cut by $C$
Solving the Max Flow Problem

• Classical Floyd-Fulkerson algorithm
  – Find the minimal cost path from S to T
  – Increase flow until edge with min weight saturates
  – Reduce remaining edges on path by $w_{\text{min}}$
  – Remove zero capacity edges and repeat

• Boykov and Kolmogorov’s algorithm can be faster
Markov Random Fields

• Bayesian image restoration \( \hat{f} = \arg \max_f P(f|g) \)

• Posterior density is \( P(f|g) \propto P(g|f)P(f) \)

• Image corrupted by Gaussian noise

\[
P(g|f) = \frac{1}{Z_1} \exp \left[ -\frac{1}{2} \left( (g - f)^T \Sigma^{-1}(g - f) \right) \right]
\]

• Prior probability

\[
P(f) = \frac{1}{Z_2} \exp (-U(f))
\]
Gibbs Distributions

• A clique in a graph is a collection of nodes that is fully connected (every node is connected to every other node)

• Gibbs distribution has

\[ P(f) = \frac{1}{Z_2} \exp \left( -U(f) \right) \]

\[ U(f) = \sum_{c \in C} V_c(f) \]

• Simple smoothness:

\[ V_c(f(p), f(q)) = \| f(p) - f(q) \|^2 \]

Nearest neighbor is most common
4 neighbors in 2D
6 neighbors in 3D
Form of Posterior Density

• Posterior also has the form of a MRF
  – Conditional:
    \[ P(g|f) = \frac{1}{Z_1} \exp \left[ -\frac{1}{2} (g - f)^T \Sigma^{-1} (g - f) \right] \]
  – Prior:
    \[ P(f) = \frac{1}{Z_2} \exp \left( - \sum_{c \in C} V_c(f(p), f(q)) \right) \]
  – Posterior:
    \[ P(f|g) \propto P(g|f)P(f) = \frac{1}{Z_3} \exp \left[ -E(f) \right] \]

• To maximize the posterior w.r.t. \( f \) is to minimize \( E(f) \)
Adapt for Segmentation

- $f$ is a “label” or “cartoon” image
- $g$ is a noisy observation of $f$

\[
P(g|f) = \frac{1}{Z_1} \exp \left[ -\frac{1}{2} \left( (g - f)^T \Sigma^{-1} (g - f) \right) \right]
\]

\[
= \frac{1}{Z_1} \exp \left[ -\frac{1}{2\sigma^2} \sum_{i=1}^{J} (g_i - f_i)^2 \right]
\]

- Posterior energy to minimize

\[
E(f) = \sum_{c \in C} V_c(f(p), f(q)) + \frac{1}{2\sigma^2} \sum_{i=1}^{J} (g_i - f_i)^2
\]

\[
= \sum_{c \in C} V_c(f(p), f(q)) + \lambda \sum_{i=1}^{J} D(f(p))
\]
Iterative Approach

• Specify an image labeling (foreground/background)
• Optimize a graph cut based on current labeling
• Repeat
• Two approaches for specifying the graph to cut
  – $\alpha - \beta$ swap
  – $\alpha$ expansion
• These methods use different graphs
• They both can be used for multiple labels (components in a scene, e.g., CSF, GM, WM)
• But $\alpha$ expansion has provable convergence properties
\(\alpha - \beta\) Swap

- Choose all pixels having either \(\alpha\) or \(\beta\) label
  - E.g., could be foreground and background = all pixels
- Connect all neighbors with edges (n-links)
- All above nodes are connected to \(\alpha\) (source) and \(\beta\) (sink) nodes
- Find best label to reduce energy

From Boykov et al., 2001
Weights for $\alpha - \beta$ Swap

- Weights

<table>
<thead>
<tr>
<th>edge</th>
<th>weight</th>
<th>for</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_p^{\alpha}$</td>
<td>$D_p(\alpha) + \sum_{q \in \mathcal{N}<em>p, q \notin \mathcal{P}</em>{\alpha \beta}} V(\alpha, f_q)$</td>
<td>$p \in \mathcal{P}_{\alpha \beta}$</td>
</tr>
<tr>
<td>$t_p^{\beta}$</td>
<td>$D_p(\beta) + \sum_{q \in \mathcal{N}<em>p, q \notin \mathcal{P}</em>{\alpha \beta}} V(\beta, f_q)$</td>
<td>$p \in \mathcal{P}_{\alpha \beta}$</td>
</tr>
<tr>
<td>$e_{{p,q}}$</td>
<td>$V(\alpha, \beta)$</td>
<td>${p,q} \in \mathcal{E}$, $p,q \in \mathcal{P}_{\alpha \beta}$</td>
</tr>
</tbody>
</table>

- Apply max flow / min cut to this
- This will relabel the graph/image with optimal descent
- Repeat the process with different labels until convergence

From Boykov et al., 2001
\( \alpha \) Expansion

- Pick a single label \( \alpha \)
- Find an expansion of \( \alpha \) in all other labels that reduces the energy
- Introduce auxiliary labels separating current partition

From Boykov et al., 2001
\( \alpha \) Expansion Weights

<table>
<thead>
<tr>
<th>edge</th>
<th>weight</th>
<th>for</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_p^\alpha )</td>
<td>( \infty )</td>
<td>( p \in P_\alpha )</td>
</tr>
<tr>
<td>( t_p^\alpha )</td>
<td>( D_p(f_p) )</td>
<td>( p \notin P_\alpha )</td>
</tr>
<tr>
<td>( t_p^\alpha )</td>
<td>( D_p(\alpha) )</td>
<td>( p \in P )</td>
</tr>
<tr>
<td>( e_{{p,a}} )</td>
<td>( V(f_p, \alpha) )</td>
<td></td>
</tr>
<tr>
<td>( e_{{a,q}} )</td>
<td>( V(\alpha, f_q) )</td>
<td>{p, q} \in \mathcal{N}, , f_p \neq f_q</td>
</tr>
<tr>
<td>( t_q^\alpha )</td>
<td>( V(f_p, f_q) )</td>
<td></td>
</tr>
<tr>
<td>( e_{{p,q}} )</td>
<td>( V(f_p, \alpha) )</td>
<td>{p, q} \in \mathcal{N}, , f_p = f_q</td>
</tr>
</tbody>
</table>

From Boykov et al., 2001
Example 1

(a) Original image  (b) Initialization  (c) Segmentation

Credit: Boykov and Funka-Lea, 2006
Example 2

(a) A slice with seeds

(b) 3D object

Credit: Boykov and Funka-Lea, 2006
Example 3

(a) Graph-cut-based hippocampus segmentation
(b) Multi-atlas-based hippocampus segmentation
(c) Manual hippocampus segmentation

Credit: van der Lijn et al., 2008
Multi-Atlas Segmentation

Subject T1w Image

Atlas T1w Images

© Jerry L. Prince
Learn Deformable Registrations

Subject T1w Image

Atlas T1w Images

Nonlinear Transformations

\( \phi_1 \)

\( \phi_2 \)

\( \phi_3 \)
Apply Nonlinear Transformations

Subject T1w Image

These “look” like the subject—they are in registration with it...mostly
Apply Transformations to Labels

Subject T1w Image

The registered labels have the anatomy of the subject...mostly
Fuse the Labels Into One Label Map

Subject T1w Image

Multi-Atlas Segmentation

Majority vote is fastest way to fuse labels, but probabilistic fusion is better
Comments on Multi-atlas Segmentation

• Registration is not perfect
  – Single registration is flawed
  – Multiple atlases are needed
  – Atlas selection could be added
  – Method is dependent on quality of registration

• Label fusion is key
  – Majority vote is fast
  – Statistical fusion methods are better, but slower

• Deformable registration is (typically) slow
  – 30 atlases is common
  – Parallelization is perfect so not 30x slower

• Topology not guaranteed
  – Anatomical realism not expected
  – Post-processing often used
DEEP NEURAL NETWORKS
Deep Learning

NEURAL NETWORK BASICS
Feed-Forward Neural Networks

- **Perceptron**
  - Visible units (inputs $v_i$)
  - Trainable weights $w_i$
  - Bias $w_0$
  - Activation function $f(\cdot)$

- **Activation function**
  - Must be nonlinear
  - Logistic sigmoid for binary classification
    $$f(z) = \frac{1}{1 + \exp[-z]}$$

$$y(v; \Theta) = f\left(\sum_{i=1}^{D} v_i w_i + w_0\right) = f\left(w^\top v + w_0\right)$$

Credit: Deep Learning for Medical Image Analysis, Zhou, Greenspan, and Shen, 2017
Linear Decision Boundary

decision = \begin{cases} 
1 & \text{if } y(\mathbf{v}) > t \\
0 & \text{otherwise}
\end{cases}
Multi-Layer Neural Network

• Single layer is fundamentally limited by its linear separation function in a classification task

• Solution: Add more layers

\[
y_k (\mathbf{v}; \Theta) = f^{(2)} \left( \sum_{j=1}^{M} W_{kj}^{(2)} f^{(1)} \left( \sum_{i=1}^{D} W_{ji}^{(1)} v_i \right) \right)
\]

\[
\mathbf{v} \xrightarrow{W^{(1)}} \mathbf{z}^{(1)} \rightarrow f^{(1)} \left( \mathbf{z}^{(1)} \right) = \mathbf{a} \xrightarrow{W^{(2)}} \mathbf{z}^{(2)} \rightarrow f^{(2)} \left( \mathbf{z}^{(2)} \right) = \mathbf{y}
\]

• Can add even more layers

\[
y_k = f^{(L)} \left( \sum_l W_{kl}^{(L)} f^{(L-1)} \left( \sum_m W_{lm} f^{(L-2)} \left( \cdots \sum_{i}^{1} W_{ji}^{(1)} x_i \right) \right) \right)
\]
Fig. 12.18  (a) A single layer enables linear separation in feature space. (b) Adding nodes in an additional (hidden) layer, arbitrary convex decision boundaries can be represented. (c) With two hidden layers, convex figures can be combined to represent arbitrary concave decision boundaries.
A 3-layers fully connected neural network (DNN)


© Jerry L. Prince
Activation Functions $f(z)$

- Must be nonlinear
  - Otherwise the entire network is just a linear weight matrix
  - Ideally “squashes” real line into a small range of numbers
  - Should be differentiable (but this is violated all the time!)
- Logistic sigmoid
  \[ \sigma(z) = \frac{1}{1 + \exp(-z)} \]
  - Range is [0,1]
- Hyperbolic tangent
  \[ \tanh(z) = \frac{\exp[z] - \exp[-z]}{\exp[z] + \exp[-z]} \]
  - Range is [-1,1]
Learning in Feed-Forward NNs

• Fundamental principle is error minimization

• Training set
  \[ \{x_n, t_n\}, n = 1, ..., N \]

• Observation \( x \in \mathbb{R}^D \)

• Class indicator vector \( t \in \{0,1\}^K \)
  - This identifies which neuron *should* fire for a given training example

• For K-class classification use cross-entropy error function

\[
E(\Theta) = -\frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \ln y_{nk}
\]

• \( t_{nk} \) is the k-th element of the n-th indicator vector

• \( y_{nk} \) is the k-th element of the NN prediction vector for \( x_n \)
Gradient of Error Function

• Use the chain rule as follows

\[
\frac{\partial E}{\partial W^{(l)}} = \frac{\partial E}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial a^{(L-1)}} \cdots \frac{\partial a^{(l+2)}}{\partial a^{(l+1)}} \frac{\partial a^{(l+1)}}{\partial a^{(l)}} \frac{\partial a^{(l)}}{\partial z^{(l)}} \frac{\partial z^{(l)}}{\partial W^{(l)}}
\]

• Where

\[
\frac{\partial a^{(l+1)}}{\partial a^{(l)}} = \frac{\partial a^{(l+1)}}{\partial z^{(l+1)}} \frac{\partial z^{(l+1)}}{\partial a^{(l)}} = f'(z^{(l)})(W^{(l+1)})^T
\]

![Graphical representation of a backpropagation algorithm in an L-layer network.]

• Gradient is

\[
\nabla E(W) = \left[ \frac{\partial E}{\partial W^{(1)}} \cdots \frac{\partial E}{\partial W^{(l)}} \cdots \frac{\partial E}{\partial W^{(L)}} \right]
\]
Gradient Descent = Backpropagation

• Update coefficients using

\[ W^{(\tau+1)} = W^{(\tau)} - \eta \nabla E (W^{(\tau)}) \]

• \( \eta \) is called the “learning rate”
• Batch gradient descent
  – Evaluate and update using all training samples
• Stochastic gradient descent
  – Evaluate and update on individual training samples selected randomly
• Mini-batch gradient descent
  – Evaluate and update on small sets of samples
Problems with Deep Neural Networks

• Problems:
  – can have millions of parameters (weights)
  – Ignores structure of images
  – Training is slow

• Solutions
  – Use local connectivity
  – Share weights via translations
  – Use rectified linear unit (ReLU) as nonlinearity
  – Use a simple optimizer
  – Use unsupervised pre-training
  – Use larger data sets
  – Use GPUs

http://www.rsipvision.com/exploring-deep-learning/
Large Problem: Overfitting

Solution:
Divide available labeled data:
1. Training
2. Validation
3. Testing

Report final performance
Principle of Autoencoders

- How to use MLPs as “feature generators”
- Reduce number of nodes and try to reconstruct *itself*
- Unsupervised NNs

- No training outputs (classes) are needed
- Gets at the question: what are the features key to describe these images/data?
Deep Learning

CONVOLUTIONAL NEURAL NETWORKS
Convolutional Neural Networks (Key #1)

- Key idea #1: “convolution” (relates to data locality)
  - Localized connectivity
  - Use the same coefficients, but shifted, within the same layer

What is “stride”?

Convolutional Neural Networks (Key #2)

- Key idea #2: “max pooling”
  - Reduces dimension
  - Keeps largest “responses”
  - Reduces shift dependence

http://cs231n.github.io/convolutional-networks/
Convolutional Neural Networks (Key #3)

• Key idea #2: “ReLU activation function”
  – Trains orders of magnitudes faster

[Graphs showing various activation functions such as sigmoid, tanh, ReLU, and softplus]

Optimization in ConvNets

• Choose a loss (error, cost) function
• Gradient descent = standard backpropagation
  – Computes gradient against all the training data
• Mini-batch gradient descent
  – Choose 100 or 256 examples from the training data (of 1.2 million)
  – Compute gradient against mini-batch
• Stochastic gradient descent
  – Compute gradient against only one training example
  – Not common because it is not as efficient
• Dropout
  – Randomly drop neurons with probability p during training
  – At test time use all neurons and multiply each activation by p to account for the scaling
• Batch normalization (and instance normalization)
  – Solves “internal covariate shift” problem; accelerates training

© Jerry L. Prince
AlexNet (2012, started modern era)

FIGURE 2.1 An illustration of the weights in the AlexNet model. Note that after every layer, there is an implicit ReLU nonlinearity. The number inside curly braces represents the number of filters with dimensions mentioned above it.
Using Pre-Trained CNNs

- AlexNet was successful in part because it was easy to retrain
- Fine-tuning
  - Download pre-trained AlexNet
  - Fix weights in early layers if task is similar
  - Re-train using SGD for new categories
- CNN activations as features
  - Use FC7 activation of an image as a generic feature descriptor
  - Better than SIFT or HoG features (so-called “hand-crafted” features)
Improving AlexNet

- AlexNet won ILSVRC challenge in 2012
- Overfeat won ILSVRC-2013
- GoogleNet won ILSVRC-2014
  - Added 1x1 convolutional layer after a regular convolutional layer
  - 1x1 is useful because it combines R,G,B color channels
- VGG-19 is a deep CNN
  - Stacks of smaller sized filters
- Deep Residual Networks (residual “blocks” are important now)
  - 152-200 layers
  - Instead of learning mapping $h(x)$, learn $f(x)-x$, then later add $f(x)+x$
- Fully Convolutional Networks

![FIGURE 2.3 Fully Convolutional Net – AlexNet modified to be fully convolutional for performing semantic object segmentation on PASCAL VOC 2012 dataset with 21 classes.](image)
SegNet: Encoder-Decoder Architecture

- Features:
  - 13 convolutional layers = first 13 in VGG16
  - Discard fully connected layers; using only convolutional layers


Fig. 2. An illustration of the SegNet architecture. There are no fully connected layers and hence it is only convolutional. A decoder upsamples its input using the transferred pool indices from its encoder to produce a sparse feature map(s). It then performs convolution with a trainable filter bank to densify the feature map. The final decoder output feature maps are fed to a soft-max classifier for pixel-wise classification.
U-Net: CNNs for Biomedical Segmentation

- Features
  - Skip connections
  - Transfers entire feature map from encoder to decoder

Deep Learning

EXAMPLE: BRAIN MRI SEGMENTATION
Case Example: Brain MRI Segmentation

• Chapter 10 in Deep Learning for Medical Image Analysis
  – Authors: Akshay Pai, Yuan-Ching Teng, Joseph Blair, Michiel Kallenberg, Erik B. Dam, Stefan Sommer, Christian Igel, and Mads Nielsen

• Application of CNN to Brain MRI Segmentation

• Major hurdles to overcome
  – 3D data are too large for CNNs of appreciable depth
  – Large variability in data
  – Too few training samples
CNN Architecture

- Use tri-planar input images (two scales)
- All feature maps and convolutions are 2D
- ReLU is used for activation
- 135 classes are to be segmented
Datasets

• Input images are T1-weighted MRI
• Labels from the MICCAI 2012 multi-atlas challenge
  – 40 cortical labels
  – 94 non-cortical labels
CNN used in Experiment

**Figure 10.2** The convolutional neural network used in the experiment. The dotted box represents the architecture of CNN without centroids and the solid box (outermost) represents the architecture of the CNN with centroids as an input feature. There are 135 neurons in the softmax layer (instead of 134) since the background class is also included.
Training

• Trained using 10 images
• Used remaining 5 for validation
• Split training into 4 parts (mini-batches) to fit in GPU
• Trained for 60 epochs (one epoch trains on n mini-batches)
• Randomly sampled 4,000,000 voxels from training set and 200,000 voxels from validation for each epoch.
  – Validation set was used to detect overfitting; none was found
• Data from training and validation set combined and the network trained from 15 images for 10 epochs
• After training, classify images in test set
• Evaluate using Dice coefficient

\[
\text{Dice}(A, B) = 2 \times \frac{|A \cap B|}{|A| + |B|},
\]
• In imaging, each pixel/voxel has a potential object that we are trying to detect.

\[
\text{Dice}(A, B) = \frac{|A \cap B|}{FN + 2TP + FP}
\]

If A is "truth".
Results

- Compare to multi-atlas method with majority vote (Reg)

Table 10.3

<table>
<thead>
<tr>
<th>Method</th>
<th>Mean Dice score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registration</td>
<td>0.724</td>
</tr>
<tr>
<td>CNN without centroids</td>
<td>0.720</td>
</tr>
<tr>
<td>CNN with centroids</td>
<td>0.736</td>
</tr>
</tbody>
</table>

FIGURE 10.4 The Dice score of different methods. This box plot illustrates the variance in the Dice measure for each method.
Conclusion of Example

• Inferior to other methods in MICCAI 2012
• Could improve
  – Use convolution kernels at different scales
  – Use more training data
  – Reduce number of classes
  – Combine registration and CNN results
  – Choose a different architecture—e.g., deeper network
EXAMPLE: SUPER-RESOLUTION
Super-resolution in MRI

- **Super-resolution**: improve resolution by multiple observations of the same object
  - Multi-resolution
  - Multi-spectral
  - Multi-orientation

- Very long history in MRI
  - (Park et al, 2003)
  - (Greenspan, 2009)
  - (Van Reeth et al., 2012)

- Multi-orientation SR is most successful to date

Example: Good in-plane resolution, poor through-plane resolution
Single Image Super-resolution

- Single-image is highly desirable
  - Saves imaging time
  - Reduces impact of patient motion

- Additional desirable traits of super-resolution:
  - Do not require an atlas
  - No PSF inversion—i.e., no deconvolution

- Our approach:
  - Use self-training
  - Use Fourier burst accumulation (FBA)
  - Jog et al., MICCAI 2016
  - Zhao et al., ISBI 2018, MICCAI 2018

FBA: Delbracio and Sapiro, TCI, 2015
Self Super-Resolution (EDSSR)

- Rotate observed image and apply 1-D blur
- Train EDSR (CNN) regression on 32x32 patches to restore original image patches
- Rotate image and apply regression multiple times
- Apply Fourier burst accumulation

EDSR: B. Lim et al., CVPR, 2017.

Synthetic images (using EDSR)
EDSSR Result: Axial View

- Original 1x1x1mm image
- Blurred 1x1x3mm image
- Super-resolution 1x1x1mm image
- Difference image

Credit: Zhao et al., ISBI 2018
EDSSR Result: Sagittal View

Original 1x1x1mm image

Blurred 1x1x3mm image

Super-resolution 1x1x1mm image

Difference image

Credit: Zhao et al., ISBI 2018
Example on Real LR Image

B-Spline
Original: 0.83x0.83x2.20 mm

SSR (Jog et al.)

EDSSR (Zhao et al.)
Super-res: 0.83x0.83x0.83 mm

Credit: Zhao et al., ISBI 2018
Aliasing and Anti-aliasing

• Spatial undersampling causes aliasing
• Blurring signal before sampling is anti-aliasing
• Consider 2D MRI
  – Contiguous slices with slice thickness = slice separation
  – Approx 3/4 of all MRI
  – Slice thickness = anti-aliasing?
• Slice profile is not effective anti-aliasing
  – 2D MRI suffers from aliasing
Evidence of Aliasing

Mamoset brain data courtesy of Dzung Pham, HJF
Synthetic Multi-Orientation Resolution Enhancement (SMORE)

Input MRI: voxel size $a \times a \times b$ [mm] ($b > a, k = b/a$)
- $a \times a$ HR in axial plane
- aliased $a \times b$ LR in sagittal and coronal planes

(a) BSP Interpolation to make voxel isotropic

(b) Blur axial slice in one axis

- Gaussian filter: width=k, FWHM=k
- Downsample by k, Upsample by k

HR Axial slice
LR slice
Aliased LR slice

(c) Extract 32x32 paired patches from aliased LR, LR, and HR axial slices. Feed paired aliased LR and LR patches to train SAA model. Feed paired aliased LR and HR patches to train SSR model.

Self Anti-aliasing (SAA) EDSR [3] model
Self super-resolution (SSR) EDSR model

(d) Extract 32x32 patches from LR sagittal and coronal slices

Aliased LR sagittal and coronal slice
Aliased LR sagittal and coronal slice

Trained SAA network

(e) Estimate SAA sagittal slices.
(e) Estimate SAA coronal slices.

SAA on LR sagittal slice
SAA on LR coronal slice

Trained SSR network

(f) Estimate SSR coronal slices.
(f) Estimate SSR sagittal slices.

SSR on LR coronal slice
SSR on LR sagittal slice

(g) FBA reconstruction [2]

© Jerry L. Prince  Credit: Zhao et al., MICCAI 2018
Marmoset Brain

- 0.15x0.15x1.0mm
- Proton density weighted
- $k = 6.6667$
- Aliasing is prominent

B-spline Interpolation  EDSSR Super-resolution  SMORE Super-resolution

Credit: Zhao et al., MICCAI 2018
Marmoset Close Up

B-spline Interpolation  EDSSR Super-resolution  SMORE Super-resolution

Credit: Zhao et al., MICCAI 2018
Marmoset Zoomed Even More

B-spline Interpolation  EDSSR Super-resolution  SMORE Super-resolution
Comments on EDSSR and SMORE

• Advantages:
  – No external atlas required
  – No contrast matching is necessary
  – Single stack of images is the only requirement
  – Save imaging time; avoid complications from motion

• Disadvantages:
  – Requires on-line training ➔ time consuming
    • Can be made faster with pre-training, transfer learning
  – Achieves in-plane resolution, no higher
  – Requires intensity inhomogeneity correction (N4)
EXAMPLE: CEREBELLUM
PARCELLATION
NEW METHOD USING DEEP NETWORKS
Block Diagram

Pre-process → Locate

Locating Network: A Deep Neural Network

Extend bounding box → Crop

Segment

Segmenting Network: Deep Fully Convolutional Neural Network

Uncrop → Post-process
Locating Network

Input block

Contracting block

Convolution kernel 3
stride 1

Instance normalization

ReLU

Convolution kernel 3
stride 2

Element-wise sum

© Jerry L. Prince
Segmenting Network

128x96x96x48 → 64x48x48x48 → 32x24x24x96 → 16x12x12x192 → 8x6x6x384 → 4x3x3x768

Input block
Contracting block
Convolution kernel 1 stride 1
Upsampling (nearest neighbor)

Expanding block

Instance normalization
ReLU
Convolution kernel 3 stride 1
Dropout
Feature concatenate

© Jerry L. Prince
Example Results

Control

SCA2

SCA6

SCA8
Comparison to Leading Method

![Graph showing comparison between CGCUTS and Proposed algorithms for different brain structures]

Algorithm:
- CGCUTS
- Proposed

Dice scores range from 0.0 to 1.0.
Deep Learning

OTHER NOTIONS AND METHODS
Problem with Deep Models

• A two-layer NN can approximate any continuous function
  – Why seek a deep network?
  – Ans: approximate complex functions to the same accuracy with fewer neurons/weights \( \Rightarrow \) train with a smaller training set

• Deep networks experience the “vanishing gradient problem”
  – Propagated errors “messages” get smaller and smaller due to repeated multiplications

\[
\frac{\partial E}{\partial W(l)} = \frac{\partial E}{\partial a(L)} \frac{\partial a(L)}{\partial a(L-1)} \ldots \frac{\partial a(l+2)}{\partial a(l+1)} \frac{\partial a(l+1)}{\partial a(l)} \frac{\partial a(l)}{\partial z(l)} \frac{\partial z(l)}{\partial W(l)}
\]

• Solutions?
  – Alternative activation functions (e.g., hyperbolic tangent, soft sign)
  – Greedy layer-wise pretraining: pretrain using unsupervised approach each layer and then fine-tune the joint network
Auto-Encoder

- Unsupervised way to train a neural network to reconstruct itself!
- Basic idea is to encourage $y$ to resemble $x \rightarrow$ no label data at all
- Use the loss function: $E(X,Y) = \frac{1}{2} \sum_{i=1}^{N} ||x_i - y_i||^2$
- # neurons in hidden layer can be smaller than # inputs
  - Dimensionality reduction
- # neurons in hidden layer can be larger than # inputs
  - Encourage sparseness
  - Discover hidden structure
Stacked-Auto Encoder

- Greatly improves the representational power
- Lower layers learn simple patterns in the image
- Higher layers learn abstract patterns/relationships

Training
- Initialize weights randomly
- Use backpropagation

Greedy training
- Train lowest layer
- Fix outputs; train next layer, and so on

Classification task
- Stack an output layer on top
- Train it with SAE fixed
- Fine tune entire network
Restricted Boltzmann Machine

- Two layer, undirected graphical model
  - Visible layer with bias term
  - Hidden layer with bias term
  - Symmetric connectivity between layers; no connectivity within layers
- Symmetry of connections
  - Learn hidden layer from observations, and
  - Reconstruct observations from hidden layer
  - Therefore, RBM is a type of auto-encoder
- Joint probability between $\mathbf{v}$ and $\mathbf{h}$

$$P(\mathbf{v}, \mathbf{h}, \Theta) = \frac{1}{Z(\Theta)} \exp[-E(\mathbf{v}, \mathbf{h}, \Theta)]$$

$$E(\mathbf{v}, \mathbf{h}, \Theta) = - \sum_{i=1}^{D} \sum_{j=1}^{F} v_i W_{ij} h_j - \sum_{i=1}^{D} a_i v_i - \sum_{j=1}^{F} b_j h_j$$
Deep Belief Network

• Stack multiple hidden layers above a visible layer
  – Top two layers are bi-directional
  – Remaining layers are directed “generative” edges

• Remains an auto-encoder
• Greedy layer-wise training can be used
• Use trained weights in DBN for a deep neural network for classification
Deep Boltzmann Machine

• Stack hidden layers on a visible layer
  – All layers are bidirectional
• Still thought of as auto-encoder
• Training
  – Generative trains on minimization of negative log likelihood
  – Classification trains on minimization of negative posterior density
Deep Learning

RESOURCES
## Software for Deep Networks

### Table 2.1

A curated list of software for Convolutional Neural Networks

<table>
<thead>
<tr>
<th>Software</th>
<th>Interfaces provided</th>
<th>AutoDiff</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caffe [53]</td>
<td>Python, MATLAB, C++, command-line</td>
<td>No</td>
</tr>
<tr>
<td>Theano [91]</td>
<td>Python</td>
<td>Yes</td>
</tr>
<tr>
<td>Tensorflow [5]</td>
<td>C++, Python</td>
<td>Yes</td>
</tr>
<tr>
<td>CNTK [6]</td>
<td>C++, command-line</td>
<td>No</td>
</tr>
<tr>
<td>MatConvNet [94]</td>
<td>MATLAB</td>
<td>No</td>
</tr>
<tr>
<td>Chainer [1]</td>
<td>Python</td>
<td>No</td>
</tr>
</tbody>
</table>
Data for Training

- [https://grand-challenge.org](https://grand-challenge.org)