ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Distributed Structured Prediction for 3D Image Segmentation

Master Thesis

Ruben Wolff

September 5, 2015

Advisors: Prof. Dr. T. Hofmann, Dr. A. Lucchi, Dr. M. Jaggi
Department of Computer Science, ETH Zürich

**Abstract**

Humans have evolved to have great pattern recognition abilities in retrieving information from data which resembles that of natural vision i.e. a three dimensional scene with different surfaces reflecting light through a translucent medium. But when it comes to recognizing objects in a scene which is solid throughout we have to resort to viewing a series of 2D slices. In practice it is time-consuming for humans to analyse three dimensional imagery; for some tasks two dimensional slices are not informative enough for good accuracy. This natural deficiency and the ever-increasing frequency in use of 3D imaging techniques is our motivation for developing flexible open-source pipeline for performing distributed 3D image segmentation.

We take a structured learning approach and construct a conditional random field (CRF) from super-pixels extracted with a 3D implementation of the Simple Linear Iterative Cluster (SLIC) algorithm. The CRF includes features of each super-pixel's local information as well as its spacial relation to the neighbouring super-pixels. A Structured Support Vector machine is trained on this CRF representation with a distributed implementation of the Block-Coordinate Frank-Wolfe algorithm. Label prediction is performed by Loopy Belief Propagation.

The viability of this pipeline is tested by measuring accuracy and scalability in a variety of configurations on real-world data sets as well as with synthetic experiments designed to illustrate particular strengths of the system. Our experiments indicate that with the right feature functions this approach can be successfully applied to a large variety of image segmentation tasks with a high or low number of training examples.

# Contents

Chapter 1

# Introduction

## 1.1 Motivation

Modern imaging techniques are rapidly becoming less noisy, less expensive, and higher resolution in the fields of biology and medicine in particular. Their (can't really use 'their' here) wide array of imaging techniques has given rise to an ever-increasing volume of high dimensional data which in the absence of computer assistance would be near impossible for humans to analyse [6], [25]. In the processing of three dimensional data especially, humans have difficulty in using their normal spacial reasoning as the images can not truly be displayed in three dimensions - it is necessary for one to look through the tissues to see the relevant markers and consequently, humans must resort to analysing the images in a series of two dimensional slices [15], [30]. We are creating an open-source distributed framework for 3D image segmentation for service researchers working with three dimensional data that can be easily adapted to many data types.

## 1.2 Image Segmentation as Structured Prediction

Structured Prediction is a subset of supervised machine learning which solves problems in the case that the output variable can not be represented as a simple label or scalar value. In our applications, we are attempting image segmentation which in a naive implementation could be modelled as a massively single nominal variable classification problem where each pixel is assigned a feature vector but the spatial relation between pixels is not considered. While the classification ground truth is per pixel, we are interested in the true objects which originally produced these pixels and their relation to other objects in the visible field. Hence the problem is more accurately modelled as a joint classification problem over all of the pixels in the image. A conditional random field (CRF) is a type of undirected proba-

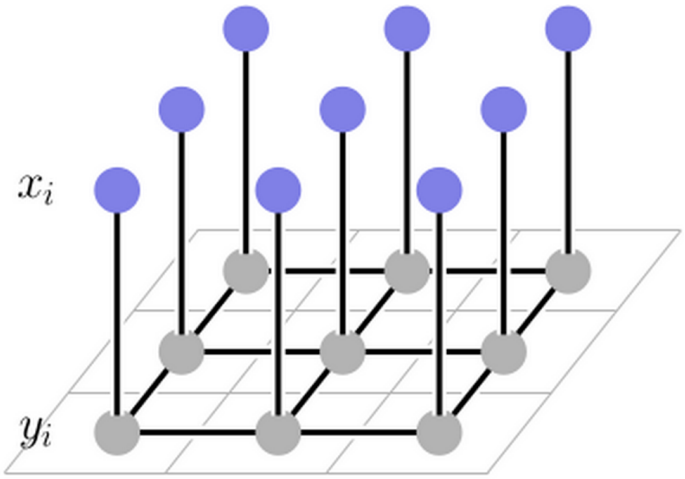



Figure 1.1: Visualization of a Conditional random field for image segmentation where $y_i$ are the super-pixel labels, the edges between $y_i$ are the pairwise potentials, $x_i$ are the unary features of that super pixel and the edges between $x_i$ and $y_i$ are the unary potentials

bilistic graphical model that can encode known information between states of nodes and that can be used to jointly calculate the marginal probability of each node state based on the possible relative states of all other nodes. We structure image segmentation as a CRF of pixel groupings which have edges to their spatially adjacent groups. These groups of pixels can also be called super-pixels.

More specifically, the conditional random field has 1 node per super-pixel representing its label which has a unary potential linked to its unary-feature node, and a pairwise potential linked to each of its neighbours. Considering each pixel to be part of a joint output domain results in an output space of $K^n$, where $K$ is the number of possible labels and $n$ is the number of super-pixels. In this paper we take the Structured Support Vector Machine (SSVM) approach to perform joint prediction over all variables and their relations simultaneously.

## 1.3 The Pipeline

For color or grayscale 3D tif stacks, image data super-pixels are constructed with the Simple Linear Iterative Clustering (SLIC) algorithm. The super-pixel to pixel assignment mask is then used to construct a graph where each node receives a feature vector calculated from only the pixels assigned to this super-pixel. The ground truth is transformed into a graph with the same mask by assigning each node the label which the majority of its pixels have.

The super-pixel generation and all other data set preprocessing is currently performed on the driver node without distributing. This training data set is then used to learn a SSVM with Block Coordinate Frank-Wolfe (BCFW). The training occurs in equal partitions locally on different machines; at the end of every round the accumulated weights are recombined by the master node. Each of the executor machines find the most constraining, possible next update direction; this sub-problem is called the max-oracle. We implemented Mean Field to perform the max-oracle step but also included Loopy Belief propagation from the factorie package[24]. The transfer of data and management of computational resources is done through the SPARK framework[40] with the CoCoA algorithm which is designed to reduce lower network traffic. CoCoA and the BCFW solver is taken from the dissolve-struct framework [29]. Once training is completed the model is again synchronized on the executor node and prediction is performed on the test data set.

## 1.4 Comparison To Related Work

For the super-pixel preprocessing step of our pipeline we choose to use SLIC super-pixels because of its time complexity and ability to control the number and shape regularity of the resulting super-pixels. Regularity of super-pixels in space is important to our model because it does not retain any information about the original spacial relations between pixels except for the case where super-pixels were neighbours during preprocessing. For the SSVM solver we chose BCFW because it can be distributed over $n$ machines if the data set has $n$ training examples and has good convergence properties even with approximate max-oracle solvers [19].

### 1.4.1 Super-pixel Alternatives

One group of alternative super-pixel generation algorithms is graph-based wherein each pixel is initially considered to be a single node in a graph with undirected edges to its adjacent pixels. The edges are assigned weights based on a feature map including information from both neighbours' member pixels. Then standard graph processing algorithms are used to minimize a global energy function over this graph resulting in a disconnected graph of super-pixels. One such algorithm is the Normalized cuts algorithm [34]. The global criteria, defined by normalized cut, captures information regarding total dissimilarity between node groupings, and also total similarity within the groups. The segmentation algorithms based on this criteria can achieve a computational complexity of $\mathcal{O}(N^{\frac{3}{2}})$ [21]. While the algorithm has had some success [27] we prefer SLIC for 3D images due to its lower complexity of $\mathcal{O}(N)$.

Another graph approach was proposed by Fezenszwalb and Huttenlocher

[9] which performs well with images that include both high variance and low variance regions. The algorithm performs global clustering where each super-pixel is the smallest spanning tree to cover its pixels. This approach has a better time complexity of $\mathcal{O}(N \log N)$ but in contrast to SLIC, it does not have the ability to control the number of super-pixels or the regularity of their shapes.

A non-parametric approach was introduced by Comaniciu and Meer which performs a recursive mean shift in the direction of the nearest stationary point of an underlying density function to find the mode of the distribution [5]. It has been shown that this algorithm is equivalent to the Nadaray-Watson regression kernel. It should be noted that this algorithm also does not have the ability to control the number or shape of the resulting super-pixels.

An empirically faster mode-seeking algorithm, Quick Shift, was introduced in 2008 [39]. The algorithm, for each pixel, finds the closest pixel in terms of increasing the Parzen density estimator and then moves them together. While it is quite fast, this algorithm can not also control the number, or compactness of the resulting super-pixels.

### 1.4.2 SSVM Solver Alternatives

The most common way of solving SVMs is through the implementation of the stochastic gradient descent algorithms (SGD). The direct generalization of SGD for problems with objective functions which are not differentiable - the stochastic sub-gradient descent algorithm (SSGD) - has found some applications in structured learning [32] [38] [33]. Stochastic sub-gradient descent has a good convergence rate of $\tilde{O}(\frac{1}{\epsilon})$ and like BCFW only requires one max-oracle call. But for SSGD to actually achieve the stated convergence rate it is assumed the user has specified an appropriate sequence of step sizes for the given problem [32]. This additional tuning parameter of the step-size sequence is not required for BCFW because we can calculate the optimal step-size per iteration.

### 1.4.3 Alternative Software Packages

NiftySeg is a 3D Image segmentation framework targeting brain imagery. It uses EM based segmentation with good performance on its data. Another valuable portion of the framework are the features specially designed for MRI images[4]. NiftySeg is a very useful tool in its specific field of application but as it is not a distributed system the size of the data sets it can reasonably process is limited.

Another specialized open-source framework was published in 2013 and named NIRFAST [14]. It is optimized for MRI images and utilizes blood oxygen

saturation, water content and lipid concentration to construct the image segments. Once again, however, this framework was not designed to run on a distributed system.

Chapter 2

# Theory

The notation we use here is a modified version of the notation used in [23] and [19]. While we will reproduce the relevant information here, [19] is the original publication of the BCFW solver and [1] is the original publication of the SLIC super pixel algorithm.

## 2.1 Recall Support Vector Machines

The traditional binary Support Vector Machine problem can be stated as finding the best weight vector $\boldsymbol{w}$ to correctly classify all multidimensional data points $x \in \mathcal{X}$ based on their features $\Phi(x)$

$$y(x) = w^T \Phi(x) \tag{2.1}$$

If $t_n \in \{-1, 1\}$ is the n'th true label for $x_n$ we can formulate the maximum margin optimization problem as :

$$\arg\max_w \{\frac{1}{||w||} min_n [t_n (w^T \Phi(x_n))]\} \tag{2.2}$$

Considering that re-scaling $\boldsymbol{w}$ does not affect the distance of any point to the decision plane we can scale them such that $t_n(w^T\Phi(x_x)) \geq 1$ holds for all points. We can drop the $min_n$ term from the optimization objective by adding the $t_n(w^T\Phi(x_x)) \geq 1, \forall x_x$ constraint because due to the $\boldsymbol{w}$ rescaling there will always be at least one point for which the above statement is an equality. By this transformation we arrive at the canonical primal definition:

$$\arg\min_w \frac{1}{2}||w||^2 \quad S.T. \tag{2.3}$$

$$t_n(w^T\Phi(x_x)) \geq 1 \tag{2.4}$$

When contrasting SVM methods with the basic Perceptron one can recognize that a Perceptron can find many different decision boundaries between

the classes depending on what the ordering of that data set was or what $w$ was initialized. The SVM has less freedom on the decision plane because it jointly optimizes the misclassification error and the margin of the support vectors. For a detailed comparison see [26]. It is intuitive to think that choosing the decision boundary with the largest margin would minimize the generalization error and it has indeed been shown theoretically that maximizing the margin minimizes bounds on the generalization error [2].

As an alternative to solving the primal formulation directly we can otherwise solve it in the dual. The dual formulation allows us to use kernel function; and later we will see for our large dimensional problem that the dual parameters can be very sparse, providing computational advantages. To construct the Dual problem we differentiate the Lagrangian and substitute back into $L(w, a)$.

$$L(w,a) = \frac{1}{2}||w||^2 - \sum_{n=1}^{N} a_n t_n (w^T \Phi(x_n)) - 1 \quad S.T. \tag{2.5}$$

$$w = \sum_{n=1}^{N} a_n t_n \Phi(x_n) \tag{2.6}$$

$$0 = \sum_{n=1}^{N} a_n t_n \tag{2.7}$$

Which leads us to the canonical dual representation:

$$\arg\max_{a} \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m \Phi(x_n)^T \Phi(x_m) \quad S.T. \tag{2.8}$$

$$a_n \geq 0, \forall_n \tag{2.9}$$

$$\sum_{n=1}^{N} a_n t_n = 0 \tag{2.10}$$

### 2.1.1 Duality Gap

The dual solution is not necessarily equal to the primal solution as the Lagrangian is only exact in the limit. The difference between the primal and dual is called the duality gap. It is possible for the gap to be zero - referred to as strong duality - but in most cases it is sufficient to compute the duality gap every couple of rounds to see if it is below a certain threshold in order to decide whether or not to stop iterating the solver.

### 2.1.2 SVM with High Class count

When using an svm for a problem with more than 2 classes one must train several classifiers with one pivot class. If we have classes [A,B,C] then we

train two SVMs [ $SVM_{AvsB}(x_n)$, $SVM_{AvsC}(x_n)$ ] predict labels by choosing the label which got the best score vs A or we predict A if both SVMs labelled it as A. This method of classification is computationally inefficient especially for problems with large numbers of classes. Image segmentation is a problem with a massive class space because we do not simply have the $R$ number of labels which a pixel could take but we have the combination of labels over the entire image, hence the number of possible labels for a single image is $R^n$ where n is the number of pixels.

## 2.2 Structured Support Vector Machine

Structured Prediction is the category of machine learning techniques designed to work with data that is more complex than a real valued outcome variable. The data will be expressed in a model which describes the relations between subsets of the data. In our image segmentation problem the output domain $y \in \mathcal{Y}(x)$ is the set of all possible label configurations for a given image $x$. And $x \in \mathcal{X}$ is the set of all possible images. Structured Support Vector machines generalize the maximum margin approach to this kind of data. The standard approach for constructing a SSVM is to define a joint feature mapping $\Phi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$ which can include a measure of distance between two $y$ and also between different images $x$. On this joint feature space we again learn a linear function to the output, in contrast to the SVM; predicting requires maximizing over the possible labels $\arg\max_{y \in \mathcal{Y}_i} L_i(\boldsymbol{y}) - \langle \boldsymbol{w}, \psi_i(\boldsymbol{y}) \rangle$. With a training data set $D = (x_i, y_i)_{i=1}^n$, we estimate $\boldsymbol{w}$ by minimizing :

$$\min_{\boldsymbol{w},\xi} \frac{\lambda}{2}||\boldsymbol{w}||^2 + \frac{1}{2}\sum_{i=1}^{n} \xi_i \quad \text{S.T.} \tag{2.11}$$

$$\langle \boldsymbol{w}, \psi_i(\boldsymbol{y}) \rangle \geq L(\boldsymbol{y}_i, \boldsymbol{y}) - \xi_i \quad \forall i, \forall \boldsymbol{y} \in \mathcal{Y}(x_i) \tag{2.12}$$

where $\psi_i(\boldsymbol{y}) := \theta(x_i, y_i) - \theta(x_i, y), \quad L_i(y) := L(y_i, y)$ and $L(y_i, y)$ is the distance between the true label configuration and another possible label configuration - typically we use the hamming distance here. The slack variable $\xi_i$ holds the accepted error in this soft-margin formulation and $\lambda$ is the regularization parameter.

This first formulation still has (an) exponential number of constraints due to $\mathcal{Y}(x_i)$, but we can alleviate this issue by replacing the $\sum_i |\mathcal{Y}_i|$ linear constraints with $n$ piecewise-linear constraints as defined by this hinge-loss:

$$\widetilde{H_i}(w) := \max_{y \in \mathcal{Y}_i} L_i(\boldsymbol{y}) - \langle \boldsymbol{w}, \psi_i(\boldsymbol{y}) \rangle \tag{2.13}$$

$$\min_{w} \frac{\lambda}{2}||\boldsymbol{w}||^2 + \frac{1}{n}\sum_{i=1}^{n}\widetilde{H_i}(w) \quad \text{S.T.} \tag{2.14}$$

$$\xi_i \geq \widetilde{H_i}(w) \tag{2.15}$$

The calculation of $\widetilde{H_i}(w)$ is also referred to as the "max-oracle" and is typically implemented with computational tricks to avoid a lot of the complexity of performing the max directly. In the following sections we assume an efficient solver for $\widetilde{H_i}(w)$ exists.

### 2.2.1 Dual Formulation SSVM

There are $m := \sum_i |\mathcal{Y}_i|$ variables which could become support vectors. In this formulation we use $\alpha_i(\boldsymbol{y})$ as the dual variable associated with the training example $i$ and $\boldsymbol{y} \in \mathcal{Y}_i$ as the potential output. Solving the Lagrangian for the SSVM similar to 2.8 results in the following dual form:

$$\min_{\alpha \in \mathbb{R}, \alpha \geq 0} f(\alpha) := \frac{\lambda}{2}||A\alpha||^2 - b^T\alpha \quad \text{S.T.} \tag{2.16}$$

$$\sum_{\boldsymbol{y} \in \mathcal{Y}_i} \alpha_i(\boldsymbol{y}) = 1 \quad \forall i \in [n] \tag{2.17}$$

where $A \in \mathbb{R}^{d \times m}$ is the column matrix $A := \{\frac{1}{\lambda n}\psi_i(\boldsymbol{y}) \in \mathbb{R}^d | i \in [n], \boldsymbol{y} \in \mathcal{Y}_i\}$ and $b := (\frac{1}{n}L_i(\boldsymbol{y}))_{i \in [n], \boldsymbol{y} \in \mathcal{Y}_i}$. In this formulation we will have a very sparse representation in the dual variable vector $\alpha$ which is advantageous for sub-gradient optimization. With the Karush-Kuhn-Tucker conditions we can map between the dual and primal forms $\boldsymbol{w} = A\alpha = \sum_{i, \boldsymbol{y} \in \mathcal{Y}_i} \alpha_i(\boldsymbol{y}) \frac{\psi_i(\boldsymbol{y})}{\lambda n}$. Computing the gradient of 2.16 we find $\nabla f(\alpha) = \lambda A^T A\alpha - b = \lambda A^T \boldsymbol{w} - b$. Additionally, note that the domain of $f(\alpha)$ is the product of $n$ probability simplicities, $\mathcal{M} := \Delta_{|\mathcal{Y}_1|} \times ... \times \Delta_{|\mathcal{Y}_n|}$. The sparsity in $\alpha$ and the ability to easily move back to the primal is crucial for solving high dimensional problems.

## 2.3 Quadratic Programming Solvers

### 2.3.1 Frank-Wolfe optimization

The Frank-Wolfe algorithm, originally published in 1956 [10], is a quadratic programming solver with the same wide applications as typical sub-gradient methods due to its only requiring optimization of linear functions over the feasible set $\mathcal{M}$. It is applicable to any convex optimization problem where the feasible set $\mathcal{M}$ is compact and the objective $f$ is convex and continuously differentiable. The basics steps of the algorithm start with choosing a feasible search corner $\boldsymbol{s}$ by minimizing the linearisation of $f$ (using the current $\alpha$) constrained on being inside $\mathcal{M}$.

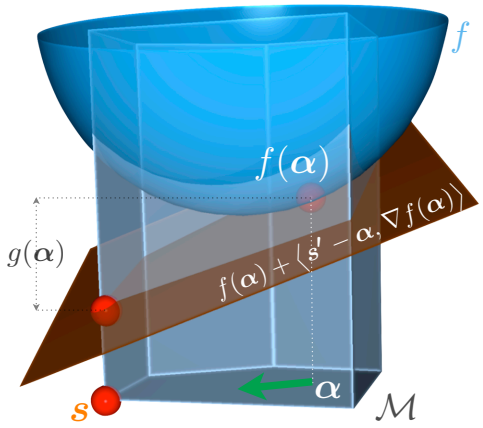



Figure 2.1

All of the following steps are a convex combination of a new $s$ and the previous iteration with some step-size $\gamma$. This formulation is advantageous for problems when $\alpha$ is in high dimensions, as the current $\alpha$ can be expressed in terms of the initial $\alpha^{(0)}$ and all $s$ corners selected in the iterations leading up to the current round. An $\alpha$ that can be written in this form can be stored in memory as a sparse object and hence never needs to allocate memory. Without this property an $\alpha$ for image segmentation would never fit into memory. Additionally, Frank-Wolfe has the advantage that we can compute the duality gap for free, because when $f$ is convex its minimization over $\mathcal{M}$ is a lower bound on the value of the globally optimal $f(\boldsymbol{\alpha}^*)$. Being able to compute the duality gap allows us to monitor the progress of the algorithm over time and also allows us to choose a theoretically-sound stopping criteria $f(\boldsymbol{\alpha}) - f(\boldsymbol{\alpha}^*) \leq \epsilon$. It has also been shown [7] that the algorithm converges to a $\epsilon$-approximate solution within $\mathcal{O}(\frac{1}{\epsilon})$

---

1 Let $\alpha^{(0)} \in \mathcal{M}$;

2 **for** $k = 0 \ldots K$ **do**

3 Compute $\boldsymbol{s} := \arg\min_{s' \in \mathcal{M}} \langle \boldsymbol{s}', \nabla f(\boldsymbol{\alpha}^{(k)}) \rangle$ ;

4 Let $\gamma := \frac{2}{k+2}$ or optimize $\gamma$ by line-search update
$\boldsymbol{\alpha}^{(k+1)} := (1 - \gamma)\boldsymbol{\alpha}^{(k)} + \gamma \boldsymbol{s}$

5 **end**

**Algorithm 1:** Frank-Wolfe on a Compact Domain

---

### 2.3.2 Block Coordinate Frank-Wolfe

One disadvantage of the Frank-Wolfe algorithm described above is that it requires $n$ calls to the maximization oracle for SSVMs. The Block-Coordinate Frank-Wolfe algorithm [19] improves this by only requiring one call to the maximization oracle in the context of SSVMs. The method described in [19] and reproduced here in Algorithm 2 can be applied to any constraint optimization problem of the form

$$\min_{\alpha \in \mathcal{M}^1 \times ... \times \mathcal{M}^n} f(\alpha) \tag{2.18}$$

having a Cartesian product over $n \geq 1$ blocks as the domain $\mathcal{M} = \mathcal{M}^1 \times ... \times \mathcal{M}^n \subseteq \mathbb{R}^m$. The reduction in oracle calls comes from this structure in $\mathcal{M}$. This structure allows us to perform cheap update steps affecting only one variable block $\mathcal{M}^{(i)}$ instead of optimizing the entire problem simultaneously. We assume that each factor is convex and compact $\mathcal{M}^{(i)} \subseteq \mathbb{R}^{m_i}$, with $m = \sum_{i=1}^{n} m_i$. In the following $\boldsymbol{\alpha}_i \in \mathbb{R}^{m_i}$ is the $i$-th block of coordinates of a vector $\boldsymbol{\alpha} \in \mathbb{R}^m$. The iterative root of Algorithm 2 chooses one block uniformly at random and leaves the other blocks unchanged. If we only had one partitioning i.e. $n = 1$ then Algorithm 2 is equivalent to the original Frank-Wolfe algorithm.

---

1 Let $\alpha^{(0)} \in \mathcal{M}^{(1)} \times ... \times \mathcal{M}^{(n)}$;

2 **for** $k = 0 \; ... \; K$ **do**

3 Pick $i$ at random in {1, ... ,n };

4 Find $\boldsymbol{s}_{(i)} := \arg\min_{\boldsymbol{s}'_{(i)} \in \mathcal{M}} \langle \boldsymbol{s}'_{(i)} \nabla_{(i)} f(\boldsymbol{\alpha}^{(k)} \rangle$ ;

5 Let $\gamma := \frac{2n}{k+2n}$ or optimize $\gamma$ by line-search;

6 Update $\boldsymbol{\alpha}_{(i)}^{k+1} := \boldsymbol{\alpha}_{(i)}^{(k)} + \gamma(\boldsymbol{s}_{(i)} - \boldsymbol{\alpha}_{(i)}^{(k)})$;

7 **end**

**Algorithm 2:** Block-Coordinate Frank-Wolfe Algorithm on product Domain

---

Another advantage of BCFW over other methods like Stochastic Gradient Descent is that we can compute our optimal step-size $\gamma$ in closed form.

#### BCFW For SSVM

For our image segmentation applications we used the BCFW algorithm 3 for SSVMs such that we only need to maintain the primal $\boldsymbol{w}$. The equivalence between Algorithm 2 and Algorithm 3 rests on the relations : $\boldsymbol{w}_s = A\boldsymbol{s}_{[i]}$

and $\ell_s = b^T \boldsymbol{s}_{[i]}$ in the primal update. Where $\boldsymbol{s}_{[i]}$ is a zero-padded version of $\boldsymbol{s}_{(i)} := \boldsymbol{e}^{\boldsymbol{y}_i^*} \in \mathcal{M}$ such that $\boldsymbol{s}_{[i]} \in \mathcal{M}$.

---

Let $\boldsymbol{w}^{(0)} := \boldsymbol{w}_i^{(0)} := \bar{\boldsymbol{w}}^{(0)} := 0, \quad \ell^{(0)} := \ell_i^{(0)} := 0;$

Using $H_i(\boldsymbol{y}, \boldsymbol{w}^{(k)}) := L_i(\boldsymbol{y}) - \langle \boldsymbol{w}, \psi_i(\boldsymbol{y}) \rangle$ ;

**for** $k = 0 \ldots K$ **do**

  Pick $i$ at random in {1, ... ,n };

  Solve $\boldsymbol{y}_i^* := \arg\max_{\boldsymbol{y} \in \mathcal{Y}_i} H_i(\boldsymbol{y}, \boldsymbol{w}^{(k)})$ ;

  Let $\boldsymbol{w}_s := \frac{1}{\lambda n} \psi_i(\boldsymbol{y}_i^*)$ and $\ell_s := \frac{1}{n} L_i(\boldsymbol{y}_i^*)$;

  Let $\gamma := \frac{\lambda(\boldsymbol{w}_i^{(k)} - \boldsymbol{w}_s) T \boldsymbol{w}^{(k)} - \ell_i^{(k)} + \ell_s}{\lambda ||\boldsymbol{w}_i^{(k)} - \boldsymbol{w}_s||^2}$ and clip to $[0, 1]$;

  Update $\boldsymbol{w}_i^{(k+1)} := (1 - \gamma) \boldsymbol{w}_i^{(k)} + \gamma \boldsymbol{w}_s$ and
  $\ell_i^{(k+1)} := (1 - \gamma) \ell_i^{(k)} + \gamma \ell_s$ ;

  Update $\boldsymbol{w}^{(K+1)} := \boldsymbol{w}^{(k)} + \boldsymbol{w}_i^{(k+1)} - \boldsymbol{w}_i^{(k)}$ and
  $\ell^{(k+1)} := \ell(k) + \ell_i^{(k+1)} - \ell_i^{(k)}$ ;

  (Optionally: Update $\bar{\boldsymbol{w}}^{(k+1)} := \frac{k}{k+2} \bar{\boldsymbol{w}}^{(k)} + \frac{2}{k+2} \boldsymbol{w}^{(k+1)}$ )

**end**

**Algorithm 3:** Block-Coordinate Frank-Wolfe Algorithm For SSVM

---

where $H_i(\boldsymbol{y}, \boldsymbol{w}^{(k)}) := L_i(\boldsymbol{y}) - \langle \boldsymbol{w}, \psi_i(\boldsymbol{y}) \rangle$ is the hinge loss as used in 2.13.

Important for our image segmentation application is that these BCFW convergence properties also hold if the max-oracle is solved approximately. We simply require that the oracle gives a candidate direction $s_{(i)}$ with multiplicative accuracy $\nu \in (0, 1]$ in terms of the duality gap on the current block [19]. But with approximate oracles there is a slow down in convergence inversely proportional to the accuracy at $\frac{1}{\nu^2}$. We will utilize this property in order to quickly find an approximate maximal energy with variational methods.

## 2.4 CRF Models

The following model describes the basis of how we construct the joint feature function $\psi$ used in the SSVM formulation 2.13. The SSVM is expressed as a quadratic programming problem hence the constraints must be linear. We express the energy function $E_w$ as an inner product of the features and the weight vector $\boldsymbol{w}$. Our CRF energy is separated into its unary and pairwise

portion.

$$D_i(y_i) = \langle \boldsymbol{w}^D, \kappa_i^D(y_i) \rangle \tag{2.19}$$

$$V_{ij}(y_i, y_j) = \langle \boldsymbol{w}^V, \kappa_{ij}^V(y_i, y_j) \rangle \tag{2.20}$$

where $\kappa^D(y_i)$ and $\kappa_{ij}^V(y_i, y_j)$ are feature maps dependent on both the observed data and the labels.

$$\kappa^D(y_i) = (I(y_i = 1)x_i^T, ..., I(y_i = K)x_i^T)^T, \tag{2.21}$$

where $x_i$ is the feature vector associated with super pixel node $i$, $K$ is the max label and $y_i \in \{1, ..., K\}$. If $\boldsymbol{w}^D$ is the column vector $[\boldsymbol{w}_1^D, ..., \boldsymbol{w}_K^D]^T$ then the unary term of $E_w$ can be written as an inner product

$$D_i(y_i) = \langle \boldsymbol{w}_{y_i}^D, x_i \rangle, \tag{2.22}$$

Which gives the unary factor of node $i$ if it were labelled with $y_i$. The pairwise feature mapping can be defined similarly

$$\kappa_{ij}^D(y_i, y_j) = (I(y_i = a, y_j = b))_{(a,b)\in\{1,...,K\}^2} \tag{2.23}$$

with parameters $\boldsymbol{w}^V = (w_{ab})_{(a,b)\in\{1,...,K\}^2}$, then we can write the pairwise term as a simple indexing

$$V_{ij}(y_i, y_j) = w_{y_i y_j} \tag{2.24}$$

This pairwise term defines the edge energy between adjacent node $i$ and $j$ as the learned cost in $w$ of the transition between label $y_i$ and $y_j$. We now combine the unary and pairwise term by allowing, $\boldsymbol{w} = ((\boldsymbol{w}^D)^T, (\boldsymbol{w}^V)^T)^T$, $\mathcal{F}^D(Y) = \sum_{i\in\nu} \kappa_i^D(y_i)$, $\mathcal{F}^V(Y) = \sum_{(i,j)\in\zeta} \kappa_{ij}^V(y_i, y_j)$ and
$\mathcal{F}(Y) = [\mathcal{F}^D(Y)^T, \mathcal{F}^V(Y)^T]^T$ to define the total energy of a CRF as an inner product

$$E_{\boldsymbol{w}}(Y) = \langle \boldsymbol{w}, \mathcal{F}(Y) \rangle \tag{2.25}$$

where $\nu$ is the set of vertices of super-pixels extracted from the input image and $\zeta$ is the set of edges. For a visualization see Figure 1.1.

### 2.4.1 CRF model variations

Of the three CRF models we used to describe our image segmentation data, the unary model is the most simple and also the only one where we can, in a reasonable amount of time, compute an exact solution to the max-oracle problem.

### Unary CRF Model

In what we call the unary model each super-pixel label node is simply connected to its unary potential which simplifies the energy function too

$$E_{\boldsymbol{w}^D}(Y) = \langle \boldsymbol{w}^D, \mathcal{F}^D(Y) \rangle \tag{2.26}$$

### Pairwise CRF Model

The pairwise model contains the same factors represented in the unary model but also includes the edge potentials as described in 2.25

$$E_{\boldsymbol{w}}(Y) = \langle \boldsymbol{w}, \mathcal{F}(Y) \rangle \tag{2.27}$$

### Data-dependent Pairwise CRF Model

The data-dependent pairwise model is an expansion of the pairwise model by redefining the pairwise energy as in

$$\mathcal{F}^V(Y) = \sum_{(i,j)\in\zeta} \kappa_{ij}^{DV}(y_i, yj) \tag{2.28}$$

$$\kappa_{ij}^{DV}(y_i, y_j) = (I(y_i = a, y_j = b) * \varpi_{ij}^I)_{(a,b)\in\{1,\ldots,K\}^2} \tag{2.29}$$

$$\varpi_{ij}^I = (I(\varpi(x_i, x_j) = c))_{c\in range(\varpi)} \tag{2.30}$$

The function $\varpi(x_i, x_j)$ is a binning function, with a small discrete range in $\mathbb{Z}$. It is used to give the pairwise term more flexibility to learn transition probabilities between labels under certain contexts. For examples of such functions and an analysis of the advantages of using a data dependent pairwise term see [35] and [22].

**Average Intensity Difference** For all neighbouring nodes $A$ and $B$ we calculate the average intensity (for RGB we also use grayscale intensity) and simply take their squared difference. Quantiles are computed for the entire data set and used to calculate fixed boundaries for binning the pairwise edges based on argument numDataDepGraidBins.

**Average Intensity Difference scaled by one hop neighbourhood Standard Deviation** For all neighbouring nodes $A$ and $B$ we calculate the difference in average intensity divided by the sum of variances of the one hope neighbourhoods of both nodes (for RGB it's also the grayscale intensity). The values are calculated for all neighbours in the training set and sorted to find quantile boundaries which result in equal mass being binned to each group. The number of bins depends on argument numDataDepGraidBins.

**Average Intensity Difference scaled by two hop neighbourhood Standard Deviation** For all neighbouring nodes $A$ and $B$ we calculate the difference in average intensity divided by the sum of variances of the two hope neighbourhoods of both nodes (for RGB it's also the grayscale intensity). The values are calculated for all neighbours in the training set and sorted to find quantile boundaries which result in equal mass being binned to each group. The number of bins depends on argument numDataDepGraidBins.

**Uniqueness difference** For all neighbouring nodes we calculate the uniqueness in its one hop neighbourhood by the number of neighbourhood standard deviations away from the neighbourhood mean the superpixel average intensity is. For any two neighbouring nodes $A$ and $B$ we then compute the difference in this uniqueness measure. The values are calculated for all neighbours in the training set and sorted to find quantile boundaries which result in equal mass being binned to each group. The number of bins depends on argument numDataDepGraidBins.

**Uniqueness in Opposite Neighbourhood** The internal super-pixel mean intensity $AvgI()$ and the average mean in its one hope neighbourhood $AvgNeigh()$ is calculated for all nodes. For all neighbouring nodes $A$ and $B$ we compute $(AvgI(A) - AvgNeigh(B))^2 + (AvgI(B) - AvgNeigh(A))^2$ as the total uniqueness of each node in the others' neighbourhood. The values are calculated for all neighbours in the training set and sorted to find quantile boundaries which result in equal mass being binned to each group. The number of bins depends on argument numDataDepGraidBins.

## 2.5 Max Oracles

The max-oracle problem, $\arg\max_{\boldsymbol{y}\in\mathcal{Y}_i} L_i(\boldsymbol{y}) - \langle \boldsymbol{w}, \psi_i(\boldsymbol{y})\rangle$, depending on the structure of $\boldsymbol{y}$ can be intractable. When modelling the image segmentation problem with a pairwise CRF as in 2.28, then a naive maximization by calculating the energy for all $\boldsymbol{y} \in \mathcal{Y}_i$ would be intractable because $|\mathcal{Y}_i| = R^{|\boldsymbol{y}|}$ where $R$ is the number of possible labels and $|\boldsymbol{y}|$ is equal to the number of super-pixels. Hence we must intelligently approximate this max-oracle or use a simpler model.

### 2.5.1 Naive Unary Max

When solving a model with unary potentials only, as in 2.26, we can find an exact solution with $\mathcal{O}(K|\boldsymbol{y})i|)$ by simply evaluating the loss-augmented potential for each super-pixel node and every possible label. This method can actually get rather good results if the unary term includes features which

have information about the surrounding space or the classes have repeating patterns in each super-pixel significantly distinct from the other classes.

#### Mean Field

All variational methods approximate a difficult to compute true distribution $P$ with a distribution with lower computational complexity $Q$ which is close to the true distribution, in terms of the Kullback–Leibler divergence. The distance between $P$ and $Q$ is defined as $D_{KL}(Q||P) = \sum_{\boldsymbol{Z}} Q(\boldsymbol{Z}) \log \frac{Q(\boldsymbol{Z})}{P(\boldsymbol{Z}|\boldsymbol{X})}$, where $\boldsymbol{Z}$ are some unobserved variables and $\boldsymbol{X}$ is a data set. In the case of our image segmentation problem $\boldsymbol{Z}$ are the hidden labels $\boldsymbol{y}$ and the data is $\mathcal{X}$. One of these variational methods is the Mean Field inference wherein a computationally feasible $Q$ is constructed as a product of individual distributions for a subset of the unobserved variables i.e. $Q(\boldsymbol{Z}) = \prod_{i=1}^{M} q_i(\boldsymbol{Z}_i|X)$. In our image segmentation problem we could model the pairwise CRF as a series of exponential distributions. The parameters of $Q$ can be quickly inferred from the data with variational calculus. When considering the CRF visualization of Figure 1.1, Mean Field would prune all edges between the labels, but then find the distributions of the individual $q_i$ iteratively as a function of all other $q_i$.

### 2.5.2 Loopy Belief Propagation

The Sum-Product algorithm is a message-passing algorithm used to find marginals on Bayesian networks which can be expressed as trees [18]. We can transform a Bayesian network that is a tree into a bipartite factor graph where each variable is a class variable in the Bayesian network and each factor is some probabilistic relation between variables (and hence has an edge to all those variables). Then we can quickly see that the marginal probability of a leaf is independent of the remaining graph variables given all factors it is connected to. Below we name the factors and variables of the bipartite graph as $f_A, f_B, f_C ... = F$ and $x_1, x_2, x_3, x_4 ... = X$ respectively. The Sum-Product Algorithm computes the marginal probabilities by applying the following rules to each node in the factor graph starting with the root.

**Product Rule** : At each variable node take the product of all its descendants.

**Sum-product Rule** : At each factor node $f_i$ take the product of all its descendants and then sum over all variables except for the uncompleted parent node.

More precisely we can define the messages:

Variable to factor message:

$$\mu_{x \to f}(f_i) = \prod_{h \in n(x) \setminus f_i} \mu_{h \to x} \tag{2.31}$$

factor to variable message:

$$\mu_{f \to x}(x_i) = \sum_{x_j \in X \setminus x_i} (f(x_j) \prod_{y \in n(f) \setminus x_i} \mu_{y \to f(y)}) \tag{2.32}$$

Where $\mu$ are types of messages, $\setminus\{a\}$ define the set operator of excluding $a$, $n(a)$ is the set of neighbours of $a$. Using the Sum-Product algorithm one can compute all marginals in $\mathcal{O}(n^2)$ and with some additional rules recomputing factors can be avoided.

Loopy Belief Propagation is an extension on the Sum-Product algorithm which simply applies the same rules to a loopy graph in each iteration ignoring that it is actually not a tree. But in practice it has shown good results [28]. We expect good results on our super-pixel graph because the direct connections are by far the most important and do not depend on much more than their adjacent nodes.

Chapter 3

# Super-Pixel Preprocessing

The computationally most expensive step during the preprocessing stage of our pipeline is the construction of super-pixels. Once the super-pixels are constructed it can be considered a single data point with its features and with one label. If we were to use a simple grid of square super-pixels we will have a good chance of summarizing features on a super-pixel which includes significant portions of two or more labels however, the resulting training set will attribute the features coming form this area to just the label which had the highest pixel count even if that is only 51% of the super-pixel. One way to minimize these overlapping features is to make the super-pixels small. As the number of super-pixels which are on the edge between two objects is smaller than the area inside each object this will make the data set cleaner overall. But the smaller we choose a super-pixel, the larger the resulting graph gets and for pairwise models the graph size has an exponential effect on the number of possible decodings for the max-oracle. Additionally, if the super-pixels are too small the unary features extracted from them may not be able to capture the signature patterns present in each object. We would like to choose a super-pixel size which optimizes these two criteria. With a fixed $S$ we are left with the specific boundaries of the super-pixel to combat the problem of overlapping features.

There are many super-pixel generation algorithms which attempt to align with edges of objects; since we are dealing with large images we choose to use the SLIC algorithm [1].

The SLIC approach is to perform a kind of localized k-means cluster on a unified space including the pixel value and its spacial coordinates. For RGB color images this space could be the the 6D space of [r,g,b,x,y,z]. If we were to run simple k-means on this space with a euclidean distance function we would run into the problem of the RGB space being bounded while the xyz space is not. Normalizing the spacial distance proportional to the selected super-pixel size is one key difference to K-means. In terms of complexity

the key difference to K-means cluster is the assumption which allows SLIC to have a $\mathcal{O}(N)$ , that assignments to a super-pixel cluster will never be farther than $2S \times 2S \times 2S$ away from the center, resulting in $K(8S^3)$ or $8N$ assignments, since $S = \sqrt[3]{\frac{N}{K}}$, $K = \frac{N}{S^3}$. While the complexity of K-means is $\mathcal{O}(NK)$, SLIC has $\mathcal{O}(N)$.

## 3.1 SLIC Algorithm

Our ScalaSLIC implementation allows for a general object data type to be saved in each pixel location. If the user would like to cluster something other than RGB or grayscale images they must define distance functions on their data type. For grayscale $d_{data} = \sqrt{(I_k - I_i)^2}$ where $I_k$ is the grayscale intensity of the cluster center and $I_i$ that of the to-be-compared pixel. For RGB $d_{data} = \sqrt{(l_k - l_i)^2 + (a_k - a_i)^2 + (b_k - b_i)^2}$ where [$l$,$a$,$b$] are the dimensions of the CIELAB color space.

---

1 Initialize cluster centers $C_k = [DataType_k, x_k, y_k, z_k]^T$ by sampling pixels at regular grid steps $S$. Perturb cluster centers in a $3x3$ neighbourhood, to the lowest gradient position.

2 **while** $E \leq threshold$ **do**

3 | **for** *each cluster center* $C_k$ **do**

4 | | Assign the best matching pixels from a $2S \times 2S \times 2S$ cube neighbourhood around the cluster center according to the distance measure $D_s = d_{data} + \frac{m}{S}\sqrt[2]{(x_k - x_i)^2 + (y_k - y_i)^2 + (z_k - z_i)^2}$

5 | **end**

6 | Compute new cluster centers

7 | Compute residual error $E$ (L1 distance between previous centers and recomputed centers)

8 **end**

9 Enforce Connectivity

**Algorithm 4:** SLIC Super-pixels

---

Our current implementation of ScalaSLIC is not distributed but the cluster assignment loop is parallelized by threading.

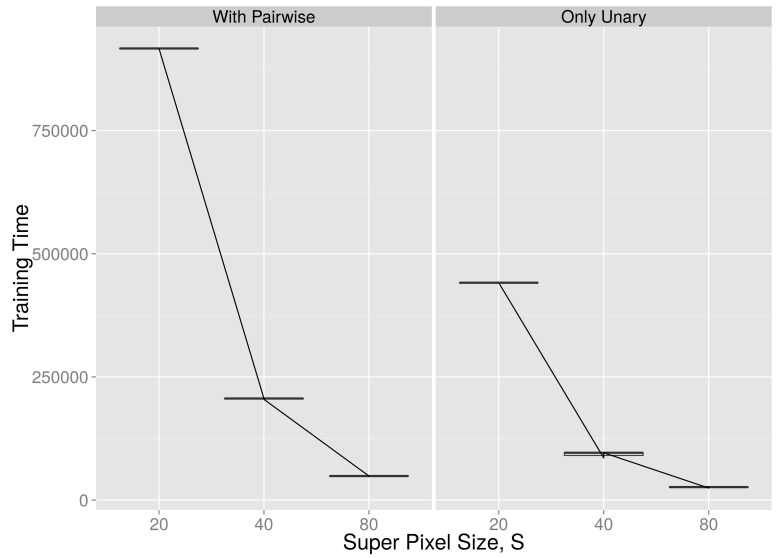



Figure 3.1: Here we plot the total training time varying the size of super-pixels (and therefore the number of nodes in the graph). Larger graphs result in larger data volumes moving across the network but the primary time-consuming function in our BCFW is the max-oracle. We repeat the same super-pixel resizing experiment with LoopyBP max-oracle and the Naive max for a unary model.

## 3.2 Super-pixel Size Effect On Training Time

One of the key factors in dealing with our high dimensional data is to not consider each pixel individually but rather to create super-pixels which then make each image significantly lower dimensional. The difficulty of the problem depends on the number of possible $\boldsymbol{y} \in \mathcal{Y}_i$, by decreasing the size of the super-pixels the number of super-pixels goes up with $\boldsymbol{y} = (\frac{L}{S})^3$, where $C$ is the length of one edge of a cubic volumetric image. The work required for the naive max-oracle on the unary model is $K(\frac{L}{S})^3$, where $K$ is the number of possible labels. For the pairwise CRF the naive approach of computing the energy for all possible $\boldsymbol{y}$ would be $K^{(\frac{L}{S})^3}$. Although LoopyBP does not try all possible $\boldsymbol{y}$ we expect the size of the graph to increase computation time more than for any unary based max-oracle. See Figure 3.1 for empirical results on total training time for both LoopyBP on a pairwise model and the Naive method on a unary model.

## 3.3 SLIC vs Naive Squares

When directly comparing the test scores of a SSVM trained on SLIC versus Square super-pixels we found that SLIC does in fact increase performance in both Unary and Pairwise models. In contrast to most figures here we are looking at the per pixel loss(see Figure 3.2. The UCSB Nuclei data set is largely skewed towards one class and hence we train on a loss that is weighted by the inverse class frequency. With loss weighting we also find a significant improvement in test loss on this Nuclei data set, see Figure 3.3.

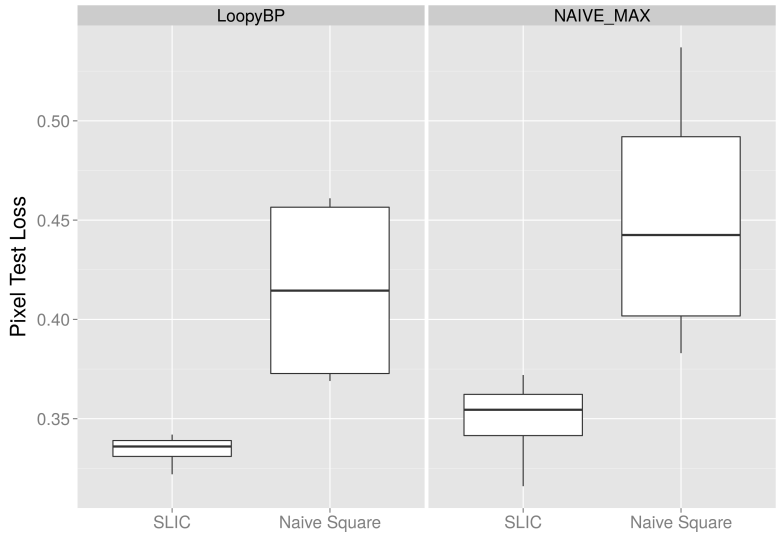



Figure 3.2: Comparing the per pixel test loss using either SLIC or Square super-pixels. The experiment was repeated for Unary model and a Pairwise model solved with Loopy Belief Propagation. Data ( SynthData see A.5, WhiteNoise:0.40 SquarImgSize:30 OsilNoise:0.40 SupSqrColorShift:0.0 SuperPix, S:30, M:30, Max Decoding:LoopyBP/NiveMax )

## 3.4 Visualization SLIC compactness parameters

In our implementation we slightly alter the use of $M$ in contrast to the above distance function. Let $x$,$y$ & $z$ be vectors containing both pixel coordinates and the spacial centres of the super-pixel clusters index by their subscript. And let $r$, $g$, $b$ be the vectors containing the RGB color information for pixels and super-pixel centres. Here we use subscript $_k$ for some cluster center and subscript $_i$ for some pixel id: $Distance(P,C) = \frac{M^2}{S^2}\sqrt[2]{(x_k - x_i)^2 + (y_k - y_i)^2 + (z_k - z_i)^2} + \sqrt{(r_k - r_i)^2 + (g_k - g_i)^2 + (b_k - b_i)^2}$.

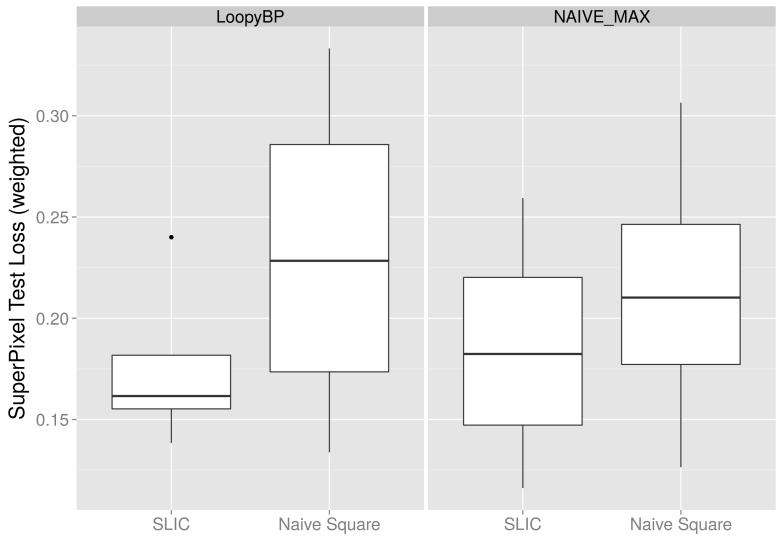


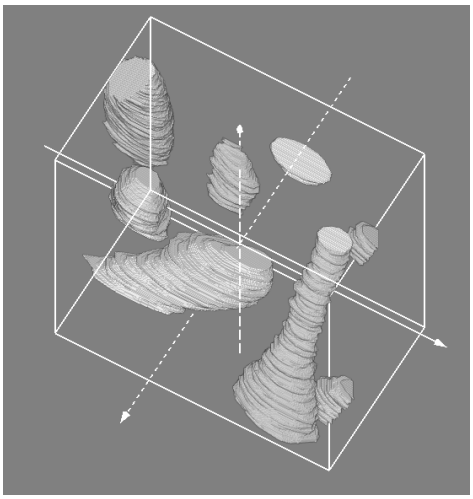

Figure 3.3: Comparing the per class frequency per super-pixel test loss using either SLIC or Square super-pixels. The experiment was repeated for Unary model and a Pairwise model solved with Loopy Belief Propagation. Data( Single Channel 3D Nuclei Images [11])

(a) S=15, M=1000 (b) S=15, M=45 (c) S=15, M=14

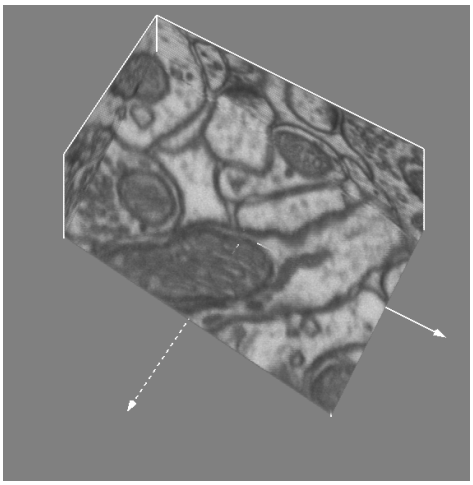
Figure 3.4: SLIC example segmentation on a single 2D MSRC image, varying the compactness paramater $M$

LABColor-space can be used optionally instead of RGB. For an illustration of the effect of parameter $M$ on the segmentation see Figure 3.4 and Figure 3.5

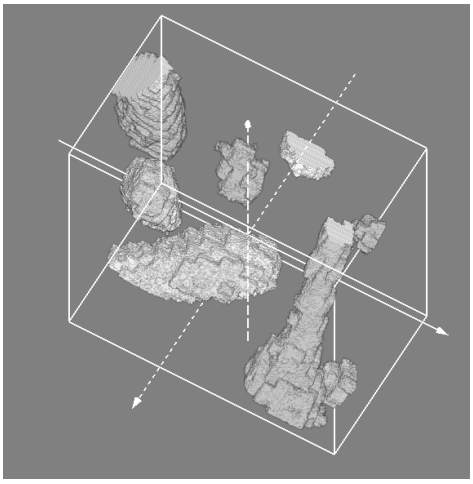
Due to the inherent difficulty of seeing through a block of solid tissue we will visualize the effect of parameter $M$ by only drawing the voxels labelled as foreground; this 3D data set is a binary classification problem and hence can be displayed like this.



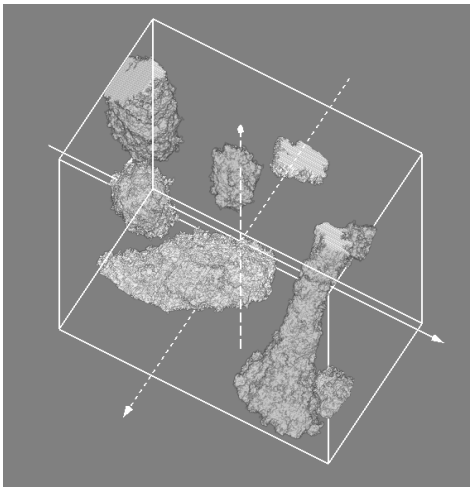
(a) Original per pixel groundtruth



(b) Original raw voxels



(c) SLIC super-pixels $M = 50$



(d) SLIC super-pixels $M = 9$

Figure 3.5: Where we visualize the effect of the SLIC compactness parameter $M$ on a 3D data set. The EM mitocondria [16] data has only two labels allow us to display one volumentrically while the other is transparent. In this case we are displaying those voxels which are labeled as mitochondria foreground. Image a) is the original ground truth, Image b) is a cross-sectional cut of the raw voxels c) are the boundaries of the super-pixels labelled as foreground with M=50 and d) Are the boundaries of the super-pixels labelled as foreground with M=9. We can see that as $M$ is lowered the boundaries of the super-pixels become less cubic and fit the true shape more.

Chapter 4

# Results

## 4.1 3D Dataset, EM Mitochondria Labelling

To demonstrate the image segmentation pipeline in all its parts we choose to use a EM Mitochondria dataset because it is a natural dataset with objects which are not easily distinguishable, and still has good labeled data [16]. The single training image of size $1024 \times 768 \times 165$ was split into 82 by 82 by 82 cubes and super-pixels where constructed with $S = 10$ and $M = 9$ producing a graph with 921 nodes on average (recall $S$ is the super-pixel size and $M$ is its compactness, see section 3.4 for details). We then trained the Naive Max classifier on a unary model, Mean Field and LoopyBP on a pairwise model and LoopyBP on two data dependent pairwise models. The results show a clear superiority of pairwise models over unary models. Amongst the pairwise models it can be beneficial to use a more complex data dependent model but the score variance is too high for a statistically significant difference. Surprisingly we found that with a $\lambda = 100$ Mean Field performed close to LoopyBP with lower training time (recall $\lambda$ is the regularization parameter). See Figure 4.1 for details.

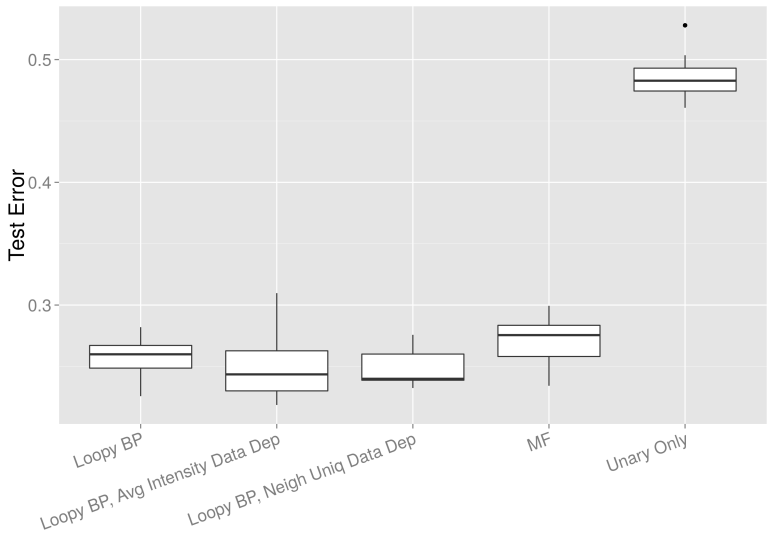



Figure 4.1: Over 30 rounds we trained the SSVM using different models and different max oracle functions. LoopyBP and MF where trained on a simple pairwise model as in 2.4.1. Naive Max oracle was trained a Unary only model see 2.4.1. Additionally we trained two data dependent pairwise models see section 2.28 with LoopyBP max-oracle. The data dependent pairwise transition binning function "Avg Intensity" groups neighbouring nodes by the difference in the mean intensity inside each super-pixel, the function "Neigh Uniq" calculates how many standard deviations away the mean of node A is from the distribution of intensities in the one hope neighbourhood of node B. (Data: EM Mitochondira Labeled Image split into $82^3$ cubes with super-pixels of size $S = 10$ [16] )

## 4.2 Sparse Transition Probability Tables

During our exploration of different datasets we saw high variance on whether the pairwise models increases accuracy as compared to the baseline naive unary max oracle. To examine which situations the pairwise term is particularly helpful we designed as set of synthetic experiments. For details on the data generation functions see Appendix A.5. In the following experiments we generated data with a significant amount of noise and set rules for how many labels are allowed to be adjacent to any label A for all other labels. We call the probability that two labels are allowed to be neighbours dataGenNeighProb. If we were to allow any label to be next to any other label then the only thing the pairwise term can learn is that super pixels generally occur in groups. So they are generally more likely to occur next to one of their own label versus any other. But by restricting only few labels to be allowed as neighbours we give the pairwise term a target which

contains many zeros and hence will have a greater impact on the decoding energy. In the following graph we show the accuracies of pairwise versus unary model while changing the probability that any 2 classes are allowed to be neighbours.


Figure 4.2: As the datagenNeighProb decreases it creates a more sparse transition probability matrix which makes it easier for the pairwise models to identify this information. Once below 0.3 the pairwise benefit starts decreasing again because the likelihood of most of the image being the background label increases. Data ( SynthData, WhiteNoise:0.40 SquarImgSize:30 OsilNoise:0.40 SupSqrColorShift:0.0 SuperPix, S:5, M:5, Max Decoding:LoopyBP )

As the reader can see the largest difference between unary and pairwise occurs in the middle of the graph. This is because as dataGenNeighProb goes to 1.0 all classes can be adjacent to all others. As it approaches 0.0 the image becomes mostly background because that label is reserved as a

Table 4.1: Transitions tables using dataGenNeighProb=0.4

True Transition probability between labels

| | | | |
|---|---|---|---|
| 0.000e+00 | 4.245e-01 | 3.271e-02 | 0.000e+00 |
| 4.245e-01 | 0.000e+00 | 2.861e-02 | 0.000e+00 |
| 3.271e-02 | 2.861e-02 | 1.667e-03 | 1.333e-02 |
| 0.000e+00 | 0.000e+00 | 1.333e-02 | 0.000e+00 |

Learned Pairwise Weights

| | | | |
|---|---|---|---|
| -7.349e-04 | 4.557e-04 | 3.112e-04 | -2.030e-05 |
| 4.557e-04 | -1.305e-04 | -3.408e-04 | -1.313e-04 |
| 3.112e-04 | -3.408e-04 | -1.149e-04 | 1.250e-04 |
| -2.030e-05 | -1.313e-04 | 1.250e-04 | -5.570e-05 |

Table 4.2: Transition tables using dataGenNeighProb=1.0:

True Transition probability between labels

| | | | |
|---|---|---|---|
| 1.046e-01 | 1.106e-01 | 1.120e-01 | 2.292e-03 |
| 1.106e-01 | 1.061e-01 | 1.099e-01 | 2.500e-03 |
| 1.120e-01 | 1.099e-01 | 1.092e-01 | 2.778e-03 |
| 2.292e-03 | 2.500e-03 | 2.778e-03 | 1.389e-04 |

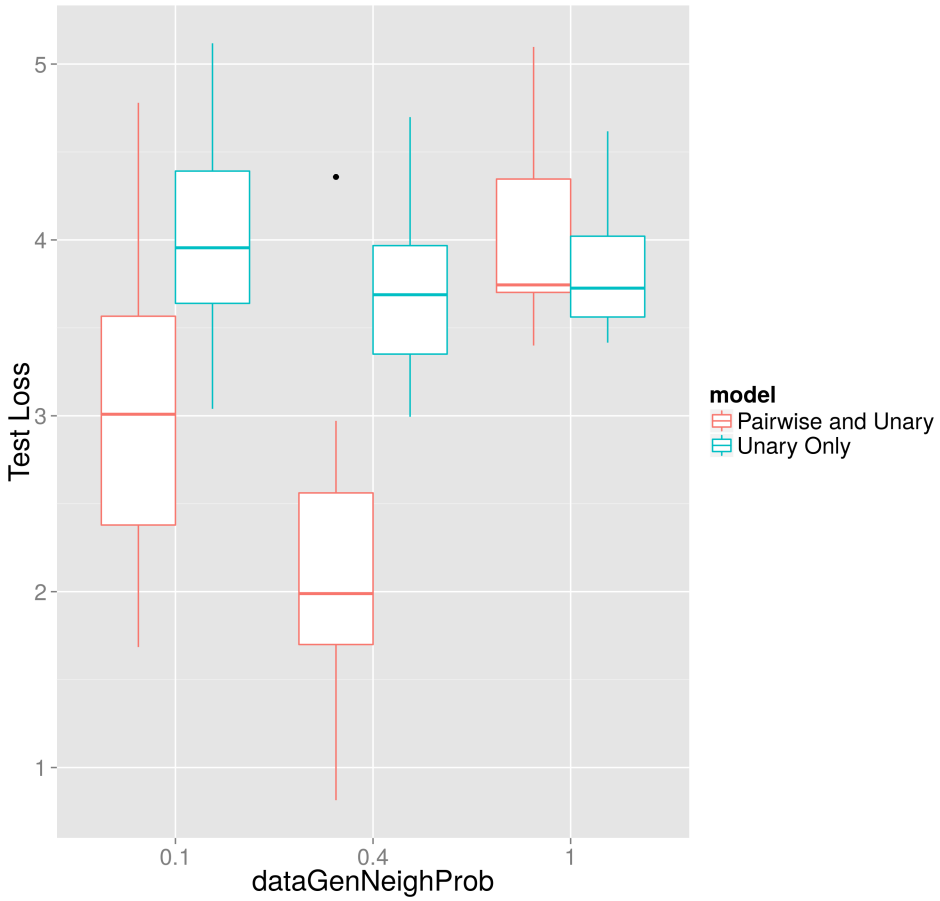
Learned Pairwise Weights

| | | | |
|---|---|---|---|
| 7.650e-05 | 3.878e-04 | 2.306e-04 | 4.022e-05 |
| 3.878e-04 | 2.015e-04 | 3.430e-04 | 4.590e-05 |
| 2.306e-04 | 3.430e-04 | -7.471e-05 | 3.638e-05 |
| 4.022e-05 | 4.590e-05 | 3.638e-05 | -4.405e-05 |

fallback when all other labels are not allowed due to the neighbouring rules. In Table 4.1 the reader can inspect columns of the transition probability and the weights learned in the pairwise model for data using this transition probability table. Notice, those label pairs which have the highest transition probability also have the highest weight in the corresponding learned pairwise potential column. In contrast table 4.2 does not have this correspondence. BCFW does not have an objective of learning the correct transition probabilities but rather to decrease the hinge loss. For this objective it only alters the direction of $\boldsymbol{w}$ if that portion of the feature vector is useful in distinguishing data. Hence when the transition probability is not sufficiently distinguishable between images it will not be learned. With the sparse transition probability explanation for the pairwise advantage in mind we can construct a dataset which should give more advantage to the pairwise model by making the super-pixels exactly equal to the supersquares. This

Figure 4.3: Test Error vs Label Transition Sparsity

With Super-pixel size == SuperSquare size




(a) The figure shows a substantial improvement in the test Error for the pairwise model. These results can be compared to 4.2 which is the same experiment but with smaller super pixels. Since in this experiment we set the super-pixels equal to the supersquare size the diagonal of the transition probability matrix is just as sparse as the rest. Data ( SynthData, WhiteNoise:0.40 SquarImgSize:30 OsilNoise:0.40 SupSqrColorShift:0.0 SuperPix, S:30, M:30, Max Decoding:LoopyBP )

configuration should result in transition probabilities from the label to itself (the diagonal) being as sparse as the rest of the matrix. In Figure 4.3a we see the expected result, that the distance between the pairwise and unary model increases as we farther increase the sparsity of the transition probability matrix.

## 4.3 Prediction Smoothing

By visual inspection of label predictions we subjectively observed a trend of the pairwise models performing better when disconnected single labels in clusters of others labels is unlikely. In figure 4.4 we can see the upper row of predicted labels by the pairwise model is much smoother than the lower row of unary model predictions. As described above the supersquares are always of equal size and in this configuration contain several super-pixels internally. The image is not perfectly smoothed into squares because we jointly optimize pairwise and unary factors. Figure 4.5 gives evidence for the same understanding of smoothness as figure 4.4 but with real world data. Again we see that the pairwise model has less standalone super pixel labels indicating that the pairwise term is working as expected. But in this particular experiment the the pairwise term also had a negative effect as on the lower side of the nuclei cluster is another set of high contrast objects which are only slightly disconnected from the main body. These nearby easily mistaken objects are all grouped into the main nuclei cluster in using the pairwise term result in a worse score than the few mislabeling made by the unary model. Hence we must be mindful of the way in which we are including the spacial relations into our model, as in this datasets where one must consider that there may be a separation between groups of classes which is less than one super pixel wide. One way to combat this could be to set the compactness parameter for the SLIC preprocessing very low so that thin long super-pixels could be placed between the two groups of classes.

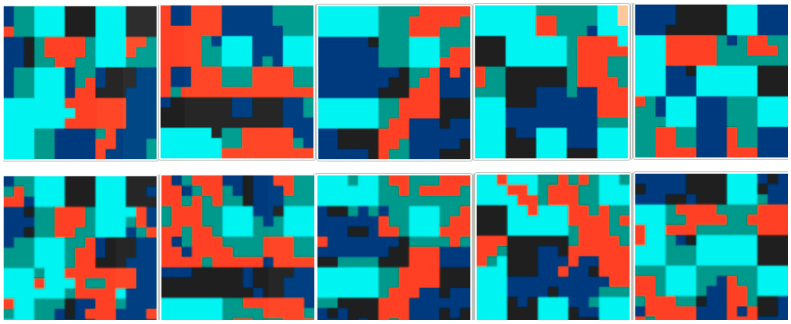
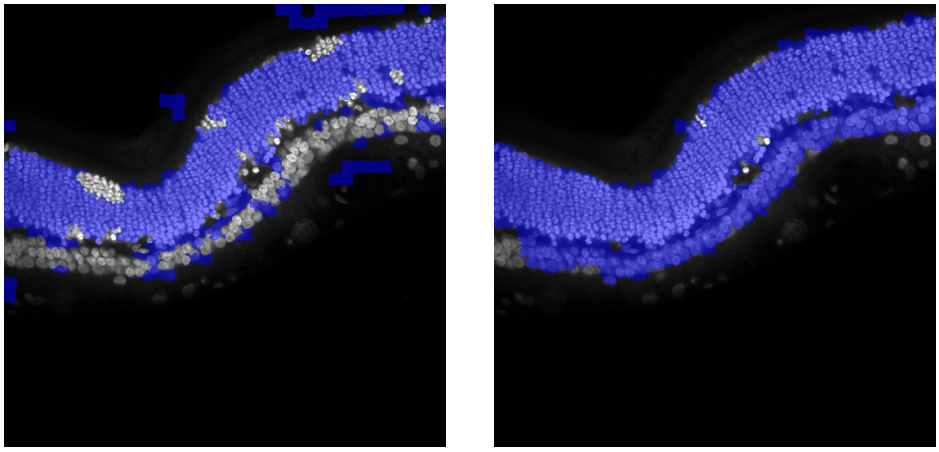


Figure 4.4: This figure shows the predicted labels per pixel for a pairwise model in the top row and a unary only model in the bottom row. By visual inspection the reader can see that the top pairwise model is more smooth in that the blobs of labels are more connected and generally in larger collections. While the bottom unary model output frequently has a single super pixel with a label surrounded by larger groups of distinct labels. Data ( SynthData, WhiteNoise:0.40 SquarImgSize:30 OsilNoise:0.40 SupSqrColorShift:0.0 SuperPix, S:30, M:30, Max Decoding:LoopyBP/NiveMax )



(a) Unary Model prediction (b) Pairwise Model prediction

Figure 4.5: Per super-pixel predicted labels are superimposed on the raw image. This Nuclei detection dataset only has foreground and background hence we only colored the Nuclei label foreground predictions. The reader can observe that the Unary model (a) has more disconnected components than the Pairwise model (b). In this particular dataset the nuclei are always connected in one large object hence giving pairwise model an advantage to not make this kind of error. Data( Single Channel 3D Nuclei [11])

## 4.4 Impact of the Regularizer

Out of all the tuning parameters lambda needs to be adjusted most carefully, as it determines the freedom $\boldsymbol{w}$ has. In the optimization function $\lambda$ can be interpreted as weighing the size of the margin and the actual loss defined in the slack variable (see equation 2.14 on page 10). As we can see in figure 4.6 choosing the right $\lambda$ for your problem is crucial for the final outcome. This pattern in test error can be further explained when looking at the structured hinge loss (2.13) over rounds, Figure 4.7. We can see that for low $\lambda$ the Structured Hinge loss $\widetilde{H}_i(w)$ does not decrease over meaning that the $\lambda$ was so restrictive on $||\boldsymbol{w}||$ that BCFW could not find a piece of information which was useful enough to warrant changing $||\boldsymbol{w}||$.

To gain farther understanding about the effect $\lambda$ has on convergence we can consider the effect it has on the $||\boldsymbol{w}||$ over time. See Figure 4.9 which shows us how $||\boldsymbol{w}||$ does not converge with a $\lambda$ of 100 and below but as $\lambda$ is increased we can see the curvature also increases indicating it would converge faster with higher $\lambda$. The test scores with really low $\lambda$ are not good but they are still significantly better than pure chance labeling, this can be explained when considering that as $\lambda$ goes to zero the optimization problem becomes a simple structured perception which still has the ability

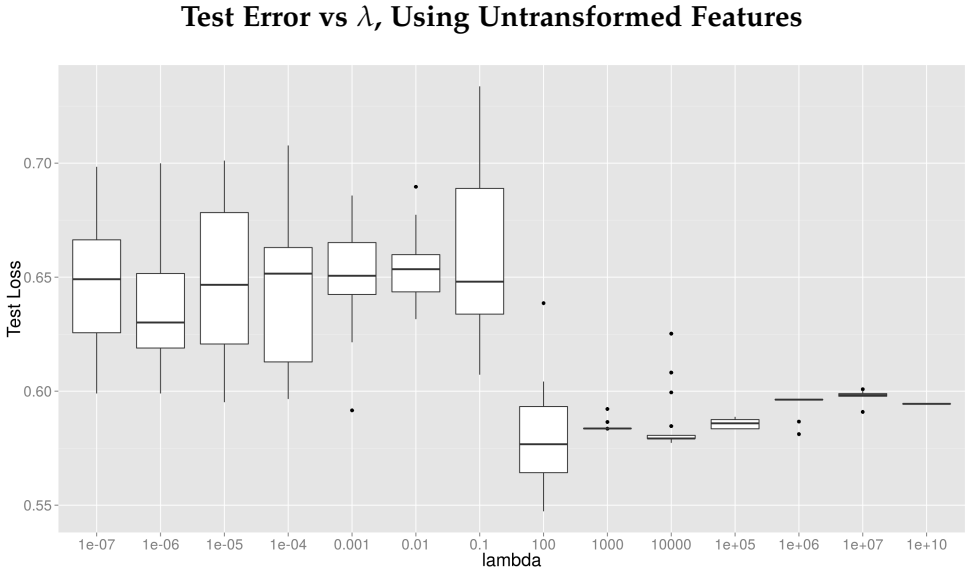



Figure 4.6: Displayed are the test scores of the same model run with different regularization $\lambda$. The distribution displayed was produced by in sample cross validation. We can see a clear separation between $\lambda$ which are not strict enough to result in good convergence ($\geq 100, \leq 10000$) and those which do not converge ($\leq 0.1$). Additionally $\lambda$ which are too strict result in lower accuracy again do to them not allowing full exploration of the relations between the features ($\geq 100000$). Data( MSRC version 2 [36])

to learn a decision boundary but it of course has very poor generalizability as can be seen by the high variance in the score between rounds in Figure 4.10.

$\lambda$ 1000 has a nice shape showing in early rounds $\boldsymbol{w}$ is changing significantly from its initialization of zero and then the norm increase plateauing indication the model has reached some kind of a stable point. Although the $||\boldsymbol{w}||$ can appear to be staying constant while the direction of $\boldsymbol{w}$ changes continuing to improve accuracy. Still since we change $\boldsymbol{w}$ in incremental steps such a curve as with $\lambda$=1000 is indication of a good choice. This choice of $\lambda$ can be confirmed if one calculates the test error every round as in Figure 4.10 and we observe a decreasing trend over successive rounds.

$\lambda$ greater than or equal to 10000 make their initial move away from zero in the first round but then steadily decrease until plateauing. One would expect that this strange kind of behavior would not result in any reasonable solution but it turns out if we look at Figure 4.10 that the test error for the highest $\lambda$ is actually very close to the best solution found at $\lambda = 1000$. To explain this behavior consider the dual objective $f(\alpha) := \min_{\alpha \in \mathbb{R}^m, \alpha \geq 0} \frac{\lambda}{2}||A\alpha||^2 - b^T\alpha$ where $A{:=}\frac{1}{\lambda n}\psi_i(\boldsymbol{y}) \in \mathbb{R}^d | i \in [n], y \in \mathcal{Y}_i$ and $b{:=}(\frac{1}{2}L_i(y))_{i \in [n], y \in \mathcal{Y}_i}$, with very

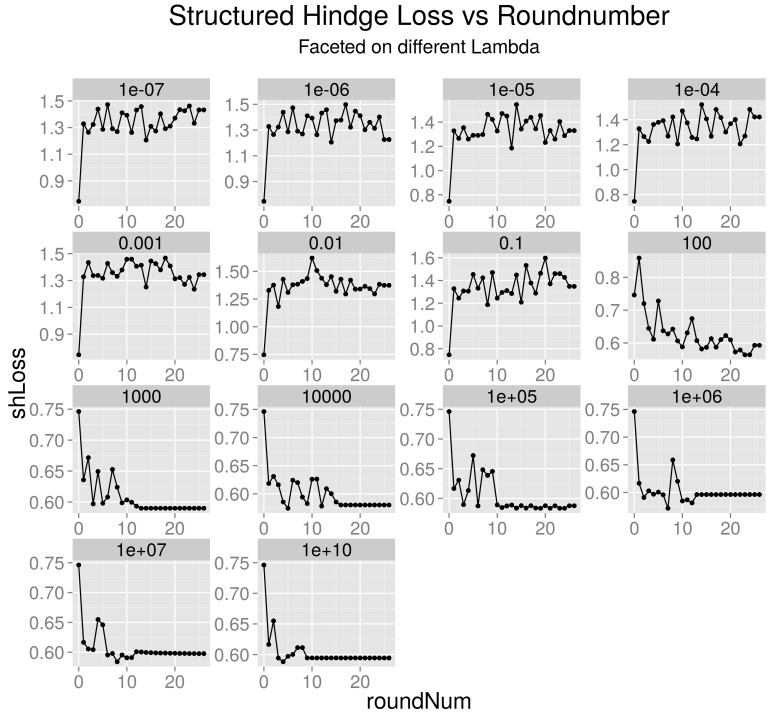



Figure 4.7: This Figure shows the Structural Hinge Loss $\max_{y \in \mathcal{Y}_i} L_i(\boldsymbol{y}) - \langle \boldsymbol{w}, \psi_i(\boldsymbol{y}) \rangle$ over time varying the regularization paramater $\lambda$. A clear difference can be seen between low $\lambda$ up to 0.1 which do not have a negative trend and also do not seem to converge in terms of structured hinge loss, while $\lambda$ 100 and above have a clear negative trend over time and as $\lambda$ gets larger they also converge faster. It should be noted that each experiment starts with the same $\boldsymbol{w}$ and hence also starts with the same structured hinge loss of 0.7462. Data( MSRC version 2 [36])

high $\lambda$ the first term of the dual would drop out due to the squared $A$ with a $\lambda$ in the following denominator.

$$\lim_{\lambda \to +\infty} (f(\alpha) := \min_{\alpha \in \mathbb{R}^m, \alpha \geq 0} \frac{\lambda}{2} ||A\alpha||^2 - b^T \alpha) = \min_{\alpha \in \mathbb{R}^m, \alpha \geq 0} -b^T \alpha$$

Now we can see that with high $\lambda$ solving the dual would end up optimizing for the point which would have the highest loss for each $\mathcal{Y}_i$ individually such that $\alpha_i(y^*) = 1$ where $argmax_{y^*} L_i(y^*)$ and $\alpha_i(y') = 0, \forall y' \neq y^*$ since $\alpha$ is constraint with $\sum_{y \in \mathcal{Y}_i} \alpha_i(y) = 1 \forall i \in [n]$. With KKT conditions $w = A\alpha$ is simply the sum of the joint features maps of the most violating label configurations for each data point. If we where to consider this solution in the special case of the binary SVM $\boldsymbol{w}$ would be the vector which if extended

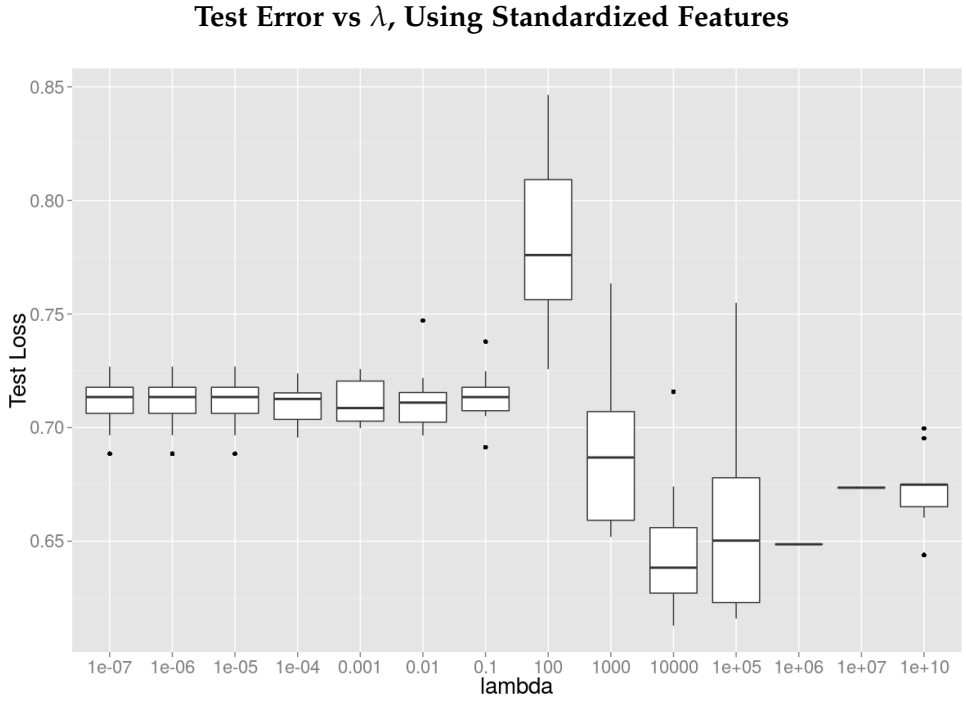



Figure 4.8: Similar to Figure 4.6 we can see that low $\lambda$ do not converge properly and hence have significantly higher loss. In this experiment the lowest loss is at 100000 several orders of magnitude higher than in Figure 4.6 which ran the same experiment only changing the feature standardization setting.; Data( MSRC version 2 [36] )

would go through the centroid of both classes. The figures used in this section where based on data which was in fact binary [11].

## 4.5 Data Dependent Pairwise Models

To examine farther why the data dependent models did not outperform the standard pairwise model we plot the convergence rate of the pairwise indexes of $\boldsymbol{w}$. As seen in Figure 4.11 the curve of the norm of $w_{pairwise}$ over sequential rounds is much steeper for the standard pairwise model as compared to the two data dependent models presented. This indicates that the data-dependent pairwise terms may have needed more rounds to converge than simple pairwise model. A rational for the difference in convergence rate could be that the data dependent term can only gain information from the portion of the transitions which are binned into that group of the data dependent pairwise model hence it probabilistically requires more data to gain the same amount of information in each bin. To test this hypothesis we reran this experiment with larger suboptimal $\lambda$ selections. In Figure 4.12 we

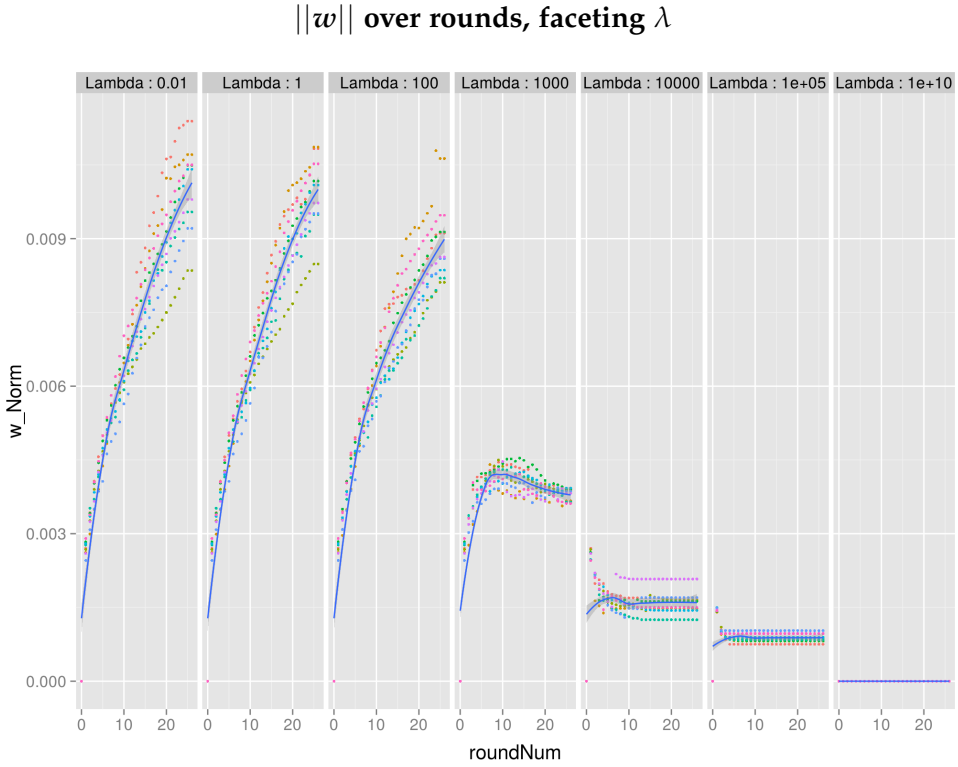



Figure 4.9: The change in the Norm of the weight vector $\boldsymbol{w}$ over time is a good indicator of the algorithms convergence depending on $\lambda$. When $\lambda$ is 1 the augmented hinge loss will prefer the $\boldsymbol{w}$ which has the smallest amount of loss, once the loss can not be improved significantly anymore it will start to choose $\boldsymbol{w}$ with a larger margin (Large Margin and Low Norm are equivalent in the SVM). The smaller $\lambda$ gets the longer the algorithm can optimize the loss without worrying about the margin. If $\lambda$ where zero then we would have a simple Perceptron. When $\lambda$ becomes too large the norm curve of $\boldsymbol{w}$ does not have the expected trend but one can still see if the algorithm converged. Data( Single Channel 3D Nuclei [11])

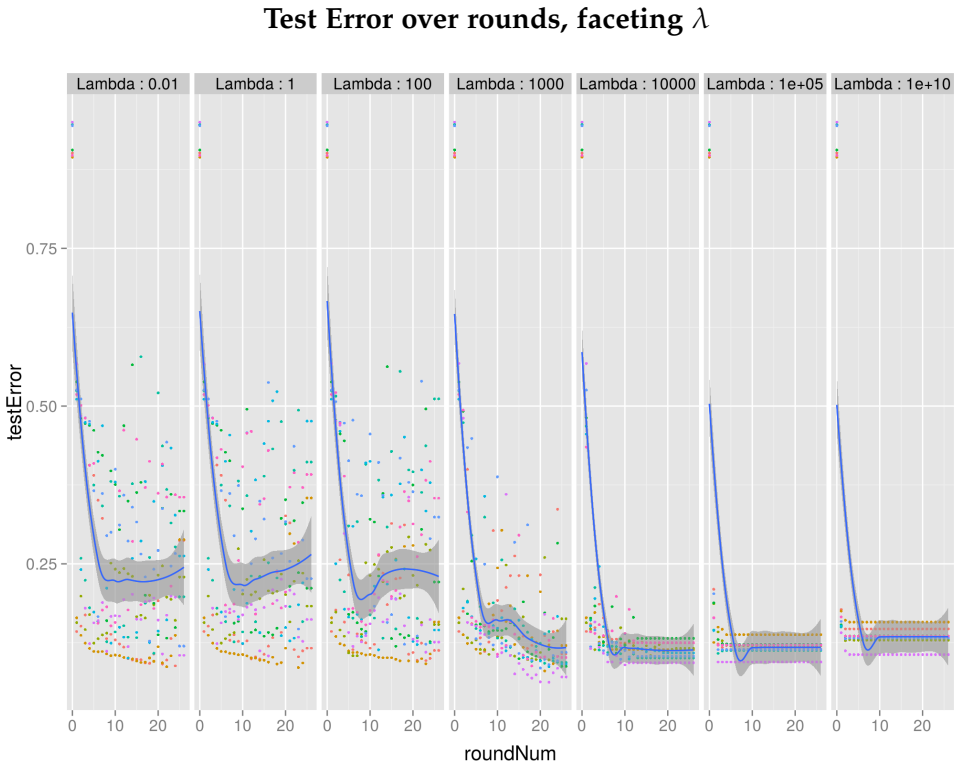



Figure 4.10: Looking at the curve of Test Error over time with different $\lambda$ we can see that higher $\lambda$ converge faster and if one sets $\lambda$ two low it does not seem to converge at all. Data( Single Channel 3D Nuclei [11])

see the Structured Hinge Loss over time and as the $\lambda$ gets higher there is a trend of the data dependent models improving their gain over the simple pairwise model. indicating that the data dependent pairwise term could converge better with a higher $\lambda$ value but such high $\lambda$ result in lower Test Error due to the unary term as described above. Preliminary results with high number of rounds $\geq 110$ show the data dependent models starting to converge and improving the test error over simple pairwise models. Put since the unary term does not require this many rounds to converge we purpose an addition to the current system which would allow for different regularization of the pairwise term separately from the unary term.

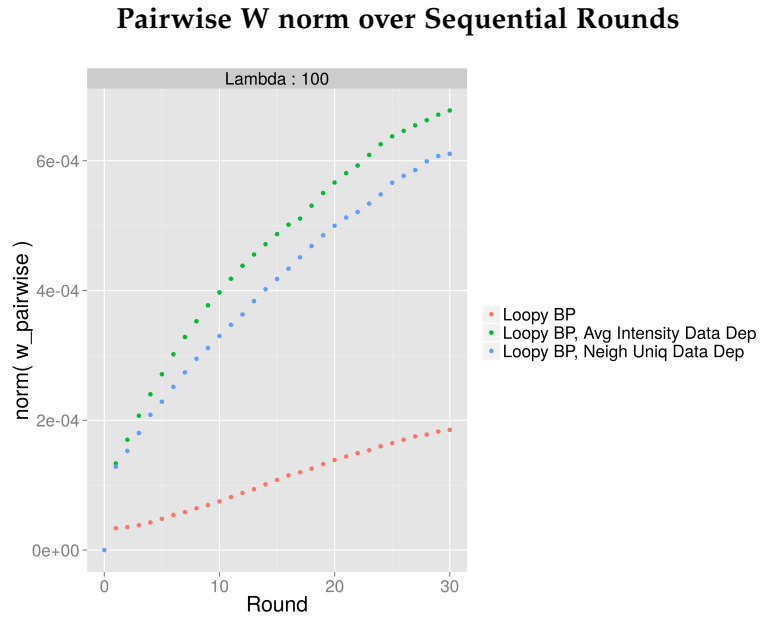



Figure 4.11: The above graph displays the norm of the pairwise portion of the weight vector. The weight vector is always initialized to zero at round zero. The curve of $||w||$ can indicate convergence behavior. In this experiment it appears that Loopy BP (the Simple Pairwise model) is converging faster than both data dependent pairwise models. Data ( EM Mitochondira Labeling split into $82^3$ cubes with super-pixels of size $S = 10$ [16] )


Figure 4.12: This Figure displays the Structured Hinge Loss ($\max_{y \in \mathcal{Y}_i} L_i(\boldsymbol{y}) - \langle \boldsymbol{w}, \psi_i(\boldsymbol{y}) \rangle$) over time of the different pairwise models. With the added linear regression line we can see that as we increase $\lambda$ the difference between the simple pairwise model to the data dependent models increases. Higher $\lambda$ cause faster convergence hence we conclude that data dependent models require more rounds to converge. Data:( EM Mitochondira Labeling split into $82^3$ cubes with super-pixels of size $S = 10$ [16] )

Chapter 5

# Distributing Workload

## 5.1 Single Node, Multiple cores

Spark can be used to distribute the inference work over a cluster but also it utlizes local parallelization without writing multi-threaded code. In some computing environments it may be cheaper to use one 36-core machine versus nine 4-core machines, or it could be advantageous for the user to avoid time lost in network synchronization. When distributing on one machine one must consider memory limitations, by default spark assigns 512mb per driver hence if one is running a 36-core machine other than the ram needed for the driver it must have 18432mb available for all the executors. When using `sbt run-main runReadTrainPredict` to start the local job one can specify the internal driver memory used with the spark_driver_memory which should be specified as a string just as in the spark configuration.

As is typical when distributing we do not get perfect scaling as adding more executors requires more synchronization work. Figure 5.1 shows total training time required for the MSRC dataset on a 8 core machine varying the number of local-spark-executors. Even when setting the number of executors equal to the total number of cores available we did not observe any thrashing but rather still saw a slight improvement from 7 to 8. Scalling well even up to the max number of cores can be partially attributed to the design where the driver has little work until a round ends and the executor results needto be combined, during this time when the driver is active the executors are idling. Hence they do not compete for computational resources even when both are on a single machine. The increase in the time needed for the Spark Driver to coordinate the different spark-executors is visualized by the difference between blue and green points (Total Training Time - Theoretically perfect Scaling). From one to two executors there is a jump in this difference because this is when merging of results first occurs. As the number of executors increases the distribution coordination time increases

but with negative second derivative. Additionally we plotted what training time we would expect if the entire system only needed to run the oracleFn and was being distributed perfectly in Red (Decoding Time / number executors). As expected this curve is slightly above the perfect theoretical scaling because of course the cores of the CPU and the work we do inside the oracleFn are not completely independent. To farther ensure that the oracleFn is being scaled as well as we think we reran the above experiment with significantly higher loopy belief propagation iterations such that the individual oracle calls are significantly longer. In Figure 5.2 we see that under these conditions we get scaling which is even closer to theoretically perfect hence affirming that the oracle calls themselves are very well distributed.

## 5.2 Multiple Nodes, Single Core

As we expected on tasks with a high max-oracle decoding time the additional time required for spark to recombine results over the network does not significantly impact scalability when contrasting to parallelizing on a single machine without network time, see Figure 5.3 for details. Additionally the data that needs to be transferred over the network is very minimal because of how dissolve-struct is designed. Dissolve-struct when configure to run Distributed-BCFW utilizes a method optimized for low network traffic called CoCoA (Communication-efficient distributed dual Coordinate Ascent) [13].CoCoA can be applied to a large class of linear regularized loss minimization objectives, which includes the SSVM image segmentation objective 2.14. The primal-dual structure of these problems was used to aggregate partial results from local computations in such a manner that reduced network traffic and avoided conflicts with updates made by different machines. It has also been shown that the CoCoA communication and weight update scheme has little effect on the total amount of computation needed to achieve the same accuracy as globally communicating methods [13].

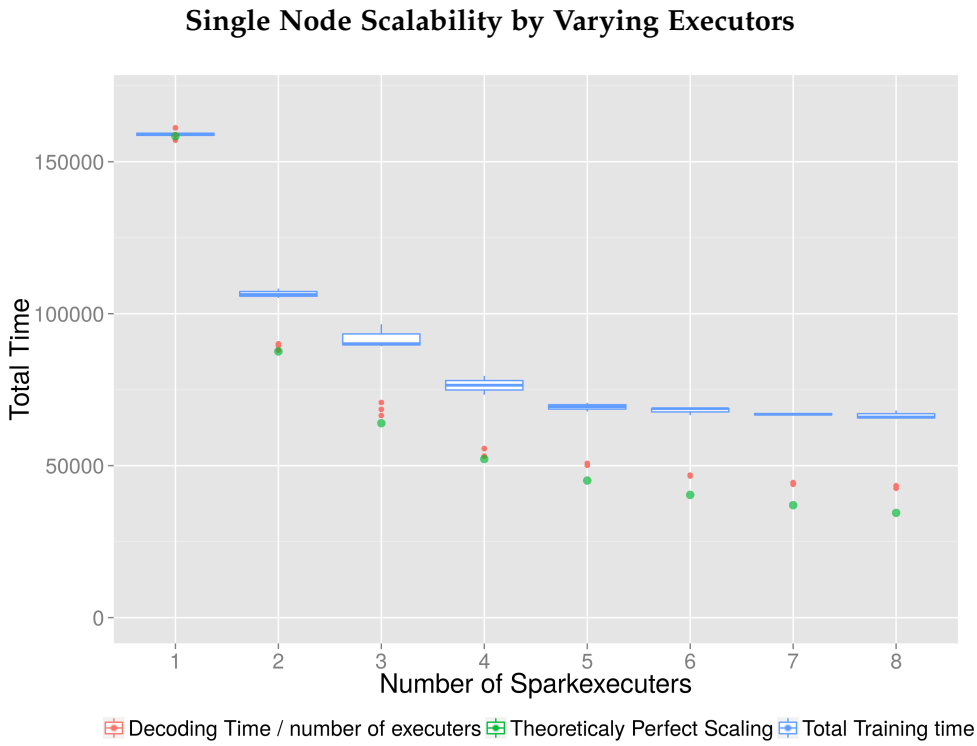



Figure 5.1: In blue we see the total time to train the model as measured on the driver node, this time consists mostly of time spent in the oracleFn but also there is a constant overhead of 16000ms and for the experiments with greater than one executor we have a significant amount of work don't on a single thread when combining and coordinating the executors. The red points are the recorded total amount of time spent inside the oracleFn divided by the number of cores available to spark shifted to match the constant overhead of the single executor experiment. The Green points are the theoretically best scaling extrapolating from the pure decoding time of the experiment with only one executor by simply dividing by the number of cores and also shifting the constant overhead.


Figure 5.2: See description for Figure 5.1 we reran the same experiment but with more loopy belief iterations to get a more precise max oracle decoding. Comparing to Figure 5.1 we see an even better scaling with very little increase in context switching time as seen by the difference to the perfect scaling curve in green. This was expected as we modulated on parameter which only effects the max-oracle timing.

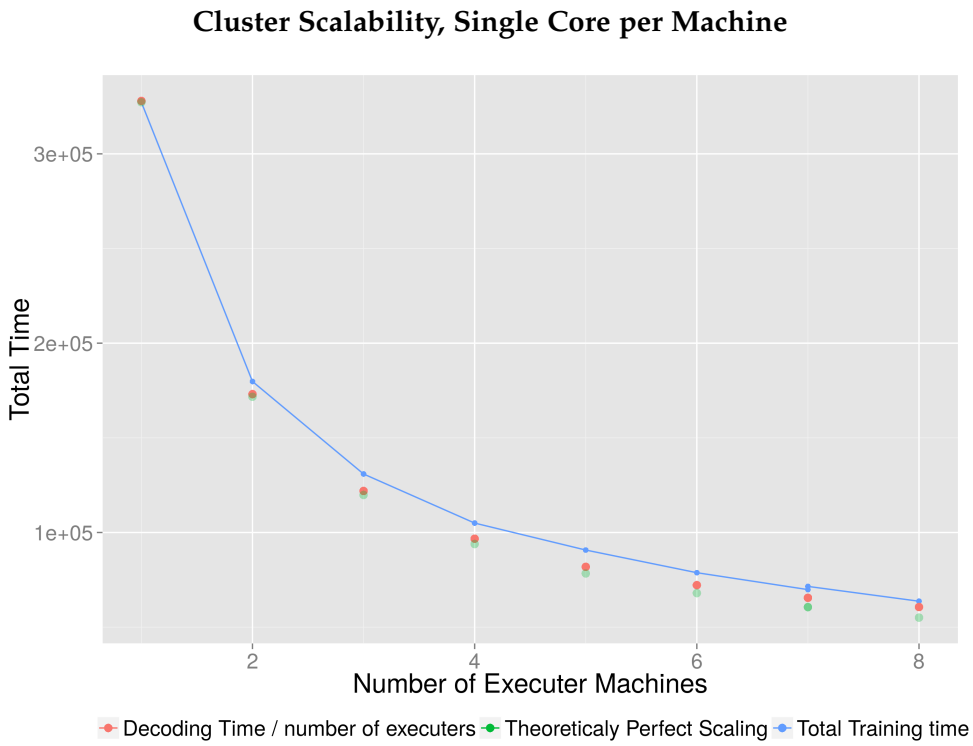



Figure 5.3: This Figure displays the total training time required for the entire MSRC data set on a varying number of m3.large AWS virtual machines each configured to use just one core. We can again see a good scaling curve as the true train time is close to the expected train time if there was no delay in context switching (Theoretically Perfect Scaling). Bu the scaling is not perfect and as more machines are added the difference in training time produced by additional time used on the driver node for recombination and coordination increases the distance to perfect scaling.

Chapter 6

# Discussion

While the EM Mitochondria data set resulted in good performance of the pairwise model there were others like the MSRC data set where the unary model was not significantly overtaken. As explained in the results section 4.3 we find that the pairwise models do produce much smoother (and therefore more natural) predictions. But this is not captured in the Hamming distance loss function and hence on some data sets the unary gets higher accuracy even if the prediction has single labels in a neighbourhood which we know a-priori would be impossible. We suggest for future work to develop a new loss function which evaluates the accuracy on the test set by also capturing the natural plausibility of the output produced. In many applications it is more important for the segmentations to be biologically plausible than having the highest per pixel match to the ground truth. Some replacements to the Hamming distance have been purposed. The PASCAL natural image segmentation challenge was measured using the per class Jaccard distance [8]. A measure which encourages labelling with the correct size was proposed by Pletscher and Kohli [31]. Balancing false positives versus false negatives with class priors was attempted by Lempitsky et. al. [20]. But all the above loss functions do not consider the topology of the labelled objects as a whole. A possible candidate for a more biologically realistic comparison could be the Rand index. It provides a segmentation accuracy in terms of pair-wise pixel connectivity [3] and may reward the pairwise models for their prediction smoothing.

Still, with the simple weighted loss function used for all our training we achieve satisfying results with very good scalability, and preliminary research has shown that the accuracy can be vastly improved by using more sophisticated features. The code used in this report will be released open source as part of the dissolve-struct package at:
`http://dalab.github.io/dissolve-struct/`

## 6.1 Expansion of Features

We did not focus too much attention on the features as their choice is very dependent on the data at hand. Still for biological data we would like to expand the set of pre-installed features to include Ray projection features **??**. And recently deep-learning features have found a lot of success on natural image segmentation problems [17]. These Deep-Learning features actually result in a much higher dimensional problem than what we are solving but since we solve it in the dual and representing alpha as sum of sparse vectors the system is robust to the feature vector becoming very large.

## 6.2 Expansion of ScalaSLIC

We chose to perform the preprocessing on single machine because it only needs to be done once before training and in order to keep this package simple as it will be published open-source separately. But in practice we found that the preprocessing takes a significant amount of time and could easily be distributed on a per image basis. Once this preprocessing step is distributed one could also expand the distance measure to include more complex features than just the LAB space distance.

## 6.3 Improving Inference

While we were rather satisfied with the approximate decoding of LoopyBP many interesting problems only require binary labeling. For this subset of problems efficient Graph-Cut algorithms exist which can perform exact inference even with loopy CRF [37]. With exact inference convergence will be faster and the variance of the estimator will be lower allowing for easier fine tuning. Hence we hope to add graph-cut as an option for solving the max-oracle problem in the future.

## 6.4 Improving Automation

As we described in section 4.4 the $\lambda$ parameter is critical for good convergence of BCFW. The user is advised to always rerun their experiment with $\lambda$ constants at several scales to find the best range. \$ could easily be selected automatically by minimizing the cross validation error on the training set. Additionally we explain the relevance of compactness parameter $M$ for the super-pixel preprocessing described in section **??**. The parameter $M$ is independent of the choice of $\lambda$ and could be optimized by running slick several times and optimizing for the most uniformity of ground truth labels within each super-pixel.

Appendix A

# Implementation Details

## A.1 Preparing and Reading in data

Regarding the data-importing functions that we provide, we expect the data to be stored in two folders - one for the raw image files and the second a folder for the ground truth mask. It is necessary to give these folders their respective names - ”Images” and ”GroundTruth”. Both raw images and ground truth can be in .bmp, .tif or .png formats but the format must be uniform throughout a given data set. Furthermore, it should be noted that due to the ground truth mask's being mapped to its image by file name only, the images folder and the ground truth folder must therefore have the same number of files, and the same set of names. Ground truth masks' labels are read in per pixel. The ground truth mask itself should be considered a space-indexed label mapping, therefore each label must have exactly one value here. This also demands that (a qualifier for 'these ' would be nice given the brevity of the rest of sentence) these files be much smaller than the image files. The data location is specified by the input argument ”dataDir=”.

### A.1.1 Caching

If the user chooses to run our preprocessing functions for creating superpixels, extracting features, and constructing the graph, then all of these will be cached on disk in the same folders in which the data reside. The caches are performed for each image individually and are named after the raw image file name. With this very simple caching system we prevent the user from having to recompute every image in the event of a crash or when changing a few parameters for later portions of the pipeline. If the user wishes to rewrite the cache without first checking whether or not there are matching flags in the files, one should in that event, specify the input argument ”recompFeat=true” or, by preference, the user can change the ”runName=” input argument to generate a new set of caches while leaving the old ones as

they are. The files ending in ".mask" in the "Images" folder contain the mapping between the pixel index and super-pixel id. Files ending in ".graph2" contain the completed graph structure of the associated image, I.E. nodes which save their own features only and the node ids of their neighbours'. Files ending in ".classCount" , ".colorlabelmapping2" and ".transProb" contain the class frequency count, the map between colors used in the ground truth files and internal label id, and the transition probabilities between labels respectively. The files ending in ".labels2" contain the cache for the true labelled graph of these training data sets.

## A.2 Running ScalaSLIC

ScalaSlic can be used as a standalone package on any kind of data which is located on a uniform grid like color voxels. When constructing a new SLIC() object one needs to input the data in a "Array[Array[Array[DataType]]]" format, where data type can be a RGB triple, a single grayscale byte but it could also be, for example, a series of RGB values from a video sequence. While the data type is technically free from the scala compiler's perspective, practically one must be able to define the following functions: `distFn:(DataType, DataType) => Double` , which simply measure the distance between two pixels. The functions:

```
rollingAvgFn = ((DataType, DataType, Int) => DataType);
normFn = ((DataType, Int) => DataType);
```

are used for the cluster update step to get an average point in this "DataType" space. As an exmaple the following are the functions we use for RGB

```
val distFnCol = (a: (Int, Int, Int), b: (Int, Int, Int)) =>
    sqrt(Math.pow(a._1 - b._1, 2) + Math.pow(a._2 - b._2, 2) +
    Math.pow(a._3 - b._3, 2));
val sumFnCol = (a: (Int, Int, Int),b: (Int, Int, Int)) => ((a._1 +
    b._1, a._2 + b._2, a._3 + a._3));
val normFnCol = (a: (Int, Int, Int), n: Int) => ((a._1 / n, a._2 / n,
    a._3 / n));
```

The final mandatory input for ScalaSLIC is S which determines the initial grid spacing of the super-pixels and thereby also setting the initial number of cluster centers, and how large the super-pixels will be. Additionally super-pixels will be modulated by the parameter M which specifies the compactness of the super-pixels. If M is not specified we go for the alternative approach of normalizing distances by the max distance in the each super-pixel set.

## A.3 Features

We implemented a host of basic features such as color and intensity histograms, a co-occurrence matrix, and other features based on the super-pixel graph structure - the sum of neighbour histograms and the neighbourhood intensity uniqueness to name just two, and it is only after the super-pixel bounds have been determined that all of these features may be computed. Most of the features are extracted by running overall pixels in the image once and adding to some the moving average of the feature indexed by the super-pixel id of that corresponding pixel. Our graph construction script `genGraphFromImages` is designed in such a way that the specified feature functions are divided into input argument `featureFn` or `afterFeatreFn`. As the name implies, afterFeatureFn is run after the graph is constructed and provides those feature functions with the graph edge information; featureFn is for unary features only. Alternatively, when using our start-up main method `runReadTrainPredict` the already implemented feature functions can be enabled or disabled with runtime arguments.

### A.3.1 Predefined Feature List

All of the below features can be specified in the run time arguments of `runReadTrainPredict` for color, grayscale, 2D and 3D image data sets.

**featIncludeMeanIntensity** Averages all pixels assigned to a super-pixel into one mean intensity. For color, this mean intensity is the corresponding converted grayscale intensity. This feature may also be added to the meta data of a node so that the data-dependent pairwise models can use it.

**featHistSize** If set above zero, we construct a normalized histogram of colors or grayscale intensities with equally sized bins. Bin sizes are always $\frac{255}{featHistSize}$. The color version of this histogram constructs bins per color dimension and keeps the same number of bins specified, hence for color featHistSize must be divisible by three.

**featUseStdHist** Will construct a discrete distribution like histograms but with non uniform bin sizes. It initially computes a global distribution of intensities and constructs bins which should contain approximately equal data points. featHistSize is still used to specify the number of bins but only the standardized bins are used in the features. The bin sizes are computed once for the whole data set and are of course not adjusted for new data hence the training set must be sufficiently large for the distribution to be accurate.

**featCoOcurNumBins** Co-occurrence matrices are constructed by again binning all pixels into uniformly sized ranges and then counting all occur-

rences each pair of bins appearing as neighbours. Again the number of bins must be divisible by three if used on color data. One can specify exactly which kind of neighbourhood should be used; by default we only consider one pixel hop and no diagonals - in order to change this one must alter the feature function call with different *directions* input. For a detailed description see [12].

**featAddOffsetColumn** This will simply add a column of ones into the feature vector giving the SSVM one more degree of freedom for its decision boundary.

**featAddIntensityVariance** Calculates pixel intensity variance per super-pixel; for color images this is again the converted grayscale value $\frac{Red}{3} + \frac{Green}{3} + \frac{Blue}{3}$.

**featUniqueIntensity** Computes mean intensities and variances for all super-pixels, and then measures the number of standard deviations away from the mean of its one hope neighbourhood that its own intensity is.

**featUnique2Hop** Equivalent to featUniqueIntensity save for using a 2 edge hope neighbourhood.

**featNeighHist** Computes histograms per super-pixels as in featHistSize and then sums all of the histograms of neighbouring super-pixels not including the data from the super-pixels' own space and finally normalizes the vector. featHistSize Again determines the size of the bins used.

**featAddSupSize** The count of the number of voxels assigned to a particular super-pixel.

## A.4 Additional Runtime Arguments

Critical

**useNaiveUnaryMax** Setting this to true will result in Dissolve-Struct using the simple per node max decoding inside the oracle function; see section 2.5.1.

**useMF** If set to true, dissolve will use the Mean Field approximation to solve the max-oracle problem; see Section 2.5.1

**modelPairwiseDataDependent** If set to true, the max-oracle will be performed on a CRF where the pairwise potentials are dependent on the label but also a function of the two super-pixels' features. This model was only implemented for Loopy Belief Propagation decoding; see Section 2.28

**mfTemp** Specify any double value for use as the temperature parameter in the mean field decoding; see Section 2.5.1

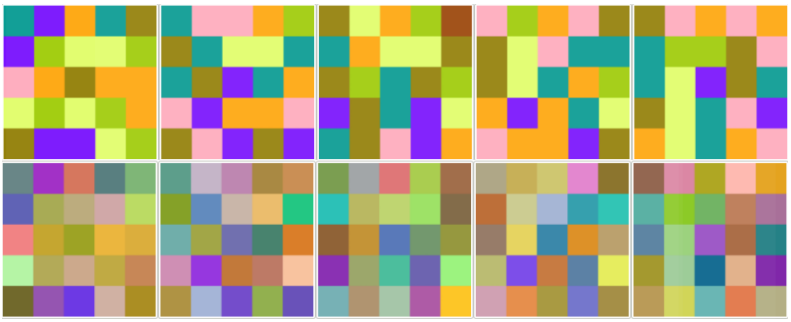


Figure A.1: An example of SuperSquareShift noise, the top row is the true ground truth and the bottom row is the image data used for features with SuperSquareShift=0.4 included.

**useLoopyBP** If set to true, Loopy Belief Propagation will be used for the max-oracle function; see

## A.5 Synthetic Data Generation

In order to intuitively show where the pairwise models have an advantage over a non structured approach we constructed a series of synthetic data generation functions.

---

See Figures : (A.1, A.2 and A.3) for visual examples of these types of noise and figure A.4 for them all together as a typical data set. The three types of noise are needed to make the unary model based classifiers less powerful because if they are too accurate themselves it is unlikely that the SSVM would add any norm to the weight vector to use the unnecessary pairwise term. The oscillatory noise is set at a wavelength proportional to S such that adjacent super-pixels inside one super-square will always have different amount of noise added. The dataGenNeighProb is used to make the transition probability more sparse and hence easier to learn making the difference between unary and pairwise models more evident. Figure A.5 is an example of a data set which has dataGenNeighProb=0.3 in contrast to figures A.1, A.2 and A.3 which have dataGenNeighProb=1.0.

```
1  instantiate new random number generated with seed dataRandSeed;
2  Randomly choose a true color per label;
3  choose a phase shift randomly for each color and spacial dimension =:
   PhaseX, PhaseY;
4  Divide the image into a grid of supersquares of size
   dataGenSquareSize;
5  With probability dataGenNeighProb choose which labels can not be
   neighbours;
6  osilationWaveLength := 1/S, where S is the superPixelSize;
7  for every supersquare Q in Grid do
8      squareNoise := choose a color uniformly at random;
9      Choose a true label at random from the set of allowed neighbours ;
10     for every pixel P in Q do
11         x := P(1);
12         y := P(2);
13         truePixel := retrieve the true color for the label at this pixel;
14         write true pixel color to groundTruth file;
15         whiteNoise := choose a color uniformly at random;
16         for each color dimension c in (RGB) do
17             OscillationNoise(c) :=
               (cos(x*/osilationWaveLength+PhaseX(c))*
               cos(y*/osilationWaveLength+PhaseY(c))+1)*255/2;
18         end
19         noisyOutput = ( truePixel + (dataAddedNoise)whiteNoise +
           (dataGenOsilNoise)OscillationNoise +
           (dataGenSquareNoise)squareNoise )/(1+ dataAddedNoise +
           dataGenOsilNoise + dataGenSquareNoise ) ;
20         write noisyOutput to image file ;
21     end
22 end
```

**Algorithm 5:** Synthetic Data Generation Algorithm

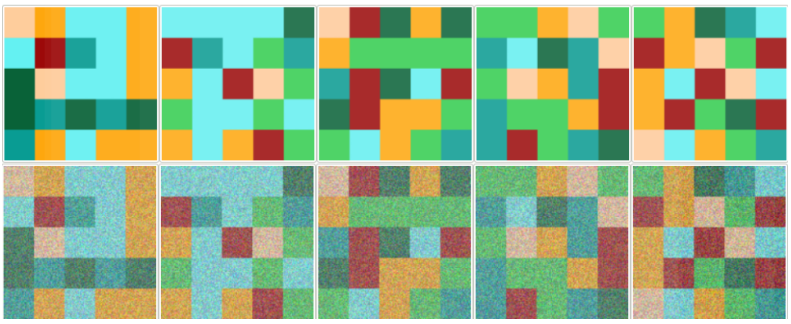


Figure A.2: An example of White Noise, the top row displays the ground truth and the bottom are the feature images with WhiteNoise=0.4 added.

Figure A.3: An example of OscillationNoise, the top row displays the ground truth and (in) the bottom row are the feature images with OscillationNoise=0.4 added.

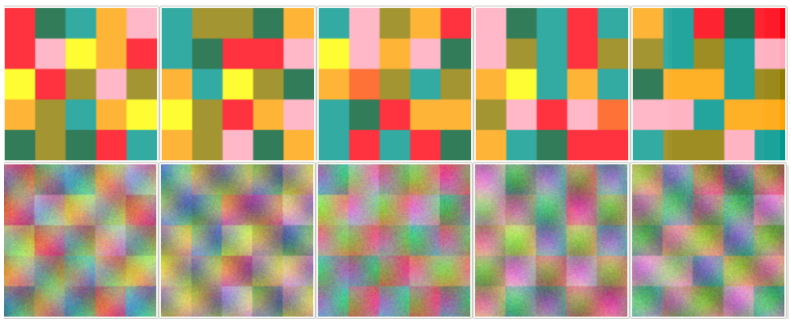


Figure A.4: An example of all types of noise. The ground truth is in the top row and the feature images are in the bottom row with the following noise added: Super-Square-shift=0.05, WhiteNoise=0.35 and dataGenOsilNoise=0.45.

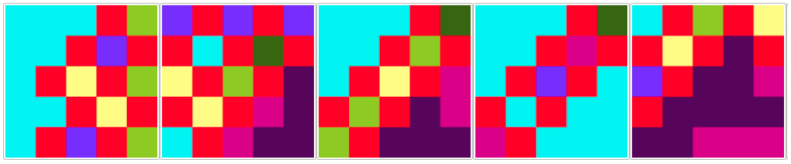


Figure A.5: An example data set showing groudataGenNeighProb=0.3, it can be contrasted to A.4 where the ground truth was generated with groudataGenNeighProb=1.0. We can see that in the above image some colors never occur next to each other.

# Bibliography


[1] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurélien Lucchi, Pascal Fua, and Sabine Susstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(11):2274–2282, November 2012.

[2] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[3] Kevin Briggman, Winfried Denk, Sebastian Seung, Moritz N Helmstaedter, and Srinivas C Turaga. Maximin affinity learning of image segmentation. In *Advances in Neural Information Processing Systems*, pages 1865–1873, 2009.

[4] Manuel Jorge Cardoso, Matt Clarkson, and Sébastien Ourselin. Niftyseg. 2011.

[5] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(5):603–619, 2002.

[6] Winfried Denk and Heinz Horstmann. Serial block-face scanning electron microscopy to reconstruct three-dimensional tissue nanostructure. *PLoS Biol*, 2(11):e329, 2004.

[7] Joseph C Dunn and S Harshbarger. Conditional gradient algorithms with open loop step size rules. *Journal of Mathematical Analysis and Applications*, 62(2):432–444, 1978.

[8] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes

(voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[9] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.

[10] Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110, 1956.

[11] Elisa Drelie Gelasca, Jiyun Byun, Boguslaw Obara, and B.S. Manjunath. Evaluation and benchmark for biological image segmentation. In *IEEE International Conference on Image Processing*, Oct 2008.

[12] Robert M Haralick, Karthikeyan Shanmugam, and Its' Hak Dinstein. Textural features for image classification. *Systems, Man and Cybernetics, IEEE Transactions on*, (6):610–621, 1973.

[13] Martin Jaggi, Virginia Smith, Martin Takac, Jonathan Terhorst, Sanjay Krishnan, Thomas Hofmann, and Michael I Jordan. Communication-efficient distributed dual coordinate ascent. In Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3068–3076. Curran Associates, Inc., 2014.

[14] Michael Jermyn, Hamid Ghadyani, Michael A Mastanduno, Wes Turner, Scott C Davis, Hamid Dehghani, and Brian W Pogue. Fast segmentation and high-quality three-dimensional volume mesh creation from medical images for diffuse optical tomography. *Journal of biomedical optics*, 18(8):086007–086007, 2013.

[15] Philipp J Keller, Annette D Schmidt, Joachim Wittbrodt, and Ernst HK Stelzer. Reconstruction of zebrafish early embryonic development by scanned light sheet microscopy. *science*, 322(5904):1065–1069, 2008.

[16] Graham Knott, Herschel Marchman, David Wall, and Ben Lich. Serial section scanning electron microscopy of adult brain tissue using focused ion beam milling. *The Journal of Neuroscience*, 28(12):2959–2964, 2008.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In

*Advances in neural information processing systems*, pages 1097–1105, 2012.

[18] Frank R Kschischang, Brendan J Frey, and Hans-Andrea Loeliger. Factor graphs and the sum-product algorithm. *Information Theory, IEEE Transactions on*, 47(2):498–519, 2001.

[19] Simon Lacoste-Julien, Martin Jaggi, Mark Schmidt, and Patrick Pletscher. Block-coordinate frank-wolfe optimization for structural svms. *arXiv preprint arXiv:1207.4747*, 2012.

[20] Victor Lempitsky, Andrea Vedaldi, and Andrew Zisserman. Pylon model for semantic segmentation. In *Advances in neural information processing systems*, pages 1485–1493, 2011.

[21] Alex Levinshtein, Adrian Stere, Kiriakos N Kutulakos, David J Fleet, Sven J Dickinson, and Kaleem Siddiqi. Turbopixels: Fast superpixels using geometric flows. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(12):2290–2297, 2009.

[22] Aurélien Lucchi, Yunpeng Li, Xavier Boix, Kevin Smith, and Pascal Fua. Are spatial and global constraints really necessary for segmentation? In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 9–16. IEEE, 2011.

[23] Aurélien Lucchi, Yunpeng Li, Kevin Smith, and Pascal Fua. Structured image segmentation using kernelized features. In *Computer Vision–ECCV 2012*, pages 400–413. Springer, 2012.

[24] Andrew McCallum, Karl Schultz, and Sameer Singh. FACTORIE: Probabilistic programming via imperatively defined factor graphs. In *Neural Information Processing Systems (NIPS)*, 2009.

[25] Sean G Megason and Scott E Fraser. Imaging in systems biology. *Cell*, 130(5):784–795, 2007.

[26] David Meyer, Friedrich Leisch, and Kurt Hornik. The support vector machine under test. *Neurocomputing*, 55(1):169–186, 2003.

[27] Greg Mori. Guiding model search using segmentation. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1417–1423. IEEE, 2005.

[28] Kevin P Murphy, Yair Weiss, and Michael I Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 467–475. Morgan Kaufmann Publishers Inc., 1999.

[29] Tribhuvanesh Orekondy, Aurélien Lucchi, and Martin Jaggi. dissolve-struct - distributed solver library for structured output prediction. 2014.

[30] Hanchuan Peng, Phuong Chung, Fuhui Long, Lei Qu, Arnim Jenett, Andrew M Seeds, Eugene W Myers, and Julie H Simpson. Brainaligner: 3d registration atlases of drosophila brains. *nAture methods*, 8(6):493–498, 2011.

[31] Patrick Pletscher and Pushmeet Kohli. Learning low-order models for enforcing high-order statistics. In *International Conference on Artificial Intelligence and Statistics*, pages 886–894, 2012.

[32] Nathan D Ratliff, J Andrew Bagnell, and Martin Zinkevich. (approximate) subgradient methods for structured prediction. In *International Conference on Artificial Intelligence and Statistics*, pages 380–387, 2007.

[33] Ben Taskar Carlos Guestrin Daphne Roller. Max-margin markov networks. *Advances in neural information processing systems*, 16:25, 2004.

[34] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000.

[35] Jamie Shotton, John Winn, Carsten Rother, and Antonio Criminisi. Textonboost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context. *International Journal of Computer Vision*, 81(1):2–23, 2009.

[36] Jamie Shotton, John Winn, Carsten Rother, and Antonio Criminisi. Textonboost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context. *International Journal of Computer Vision*, 81(1):2–23, 2009.

[37] Martin Szummer, Pushmeet Kohli, and Derek Hoiem. Learning crfs using graph cuts. In *Computer Vision–ECCV 2008*, pages 582–595. Springer, 2008.

[38] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. In *Journal of Machine Learning Research*, pages 1453–1484, 2005.

[39] Andrea Vedaldi and Stefano Soatto. Quick shift and kernel methods for mode seeking. In *Computer Vision–ECCV 2008*, pages 705–718. Springer, 2008.

[40] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.