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

This example thesis briefly shows the main features of our thesis style, and how to use it for your purposes.

Contents

Chapter 1

Introduction

1.1 Motivation

Modern image techniques are rabidly becoming, less noisy, less expensive and higher resolution especially in Biology and Medicine. This large array of different imaging techniques is producing an ever increasing volume of high dimensional data which is impossible for humans to analyze without computer assistance. Particularly for three dimensional data humans have difficulty using their normal spacial reasoning as the images can not truly be displayed in three dimensions because one needs to look through the tissues to see the relevant markers and hence humans must resort to analyzing the images in a series of two dimensional slices [5], [22], [12], [27]. To Service researchers working with three dimensional data we are creating an open-source distributed framework for 3d image segmentation that can be easily adopted to many data types.

1.2 Image Segmentation as Structured Prediction

Structured Prediction learning is a subset of supervised machine learning which solves problems where 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 modeled as massively single variable classification problem where each pixel is assigned a feature vector but the spatial relation between pixels is not considered. A more natural representation of the image segmentation problem would consider spacial relationship between the pixels and try to classify objects together. We model image segmentation as conditional random field of groups of pixels which are have edges to their spatially adjacent groups. These groups of pixels can also be called Super-Pixels.

The conditional random field more specifically has 1 node per super-pixel

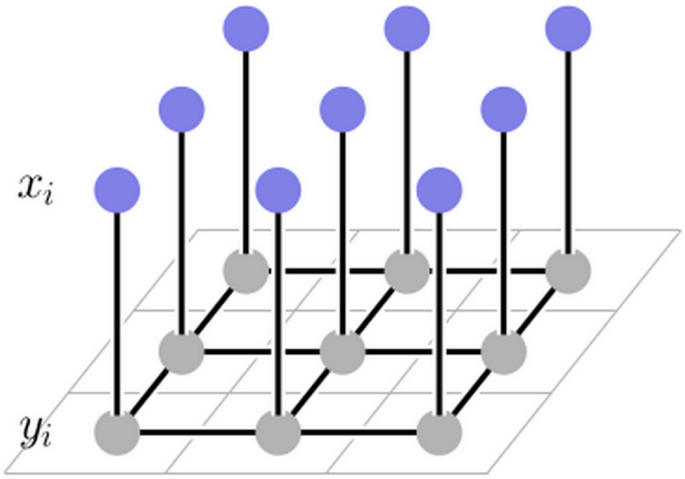


Figure 1.1: Visualization of a Conditional random field for image segmentation where y_i are the super-pixel labels, the edges between y_i are the pairwise potentials, x_i are the unary features of that super pixel and the edges between x_i and y_i are the unary potentials

representing its label which has a unary potential linked to its unary-feature node and a pairwise potential linked to each of its neighbors labeled nodes.

But when we model the problem jointly over all super-pixels the output domain experiences combinatorial growth with increasing number of super-pixels. Due to this large domain standard classification methods will not work, in this paper we take the Structured Support Vector Machine (SSVM) approach with a the Block Coordinate Frank-Wolfe solver (BCFW).

1.3 The Pipeline

Our pipeline starts with either color or greyscale images in 3D tif stacks. These images are read and 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 had. This training dataset is then used to learn an SSVM with BCFW. The training occures in equal paritions localy on different machines and at the end of every round the accumulated weightes are recombined by the master node. Each round each machine finds the most constraining possible next update direction by solving the max-oracle. We implemented Mean Field to perform the max-oracle step but also included Loopy Bliefe propagation from the

factorie framework[21]. The moving of data and managing computational power is done through the SPARK framework[35] with the CoCoA algorithm for lower network traffic. CoCoA and the BCFW solver is taken from the dissolve-struct framework [26]. Once training is completed the model is again synchronized on the executor node and prediction is performed on the test dataset.

Add image showing. Original Image, Super-pixel Boundaries,real ground truth, Ground truth interms of super pixels, a graph with labels and features, some image representaing and SVM, and then predicted outcome again interms of pixels

1.4 Compare To other 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 superpixels, but there are many other options which the user can easily integrate into the system to replace SLIC. For the SSVM solver we chose BCFW because it can be distributed over n machines if the dataset has n training examples. And the

1.4.1 Super-pixel Alternatives

One group of super-pixel generation algorithms are graph based. Where in 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 some feature criteria of the pixel nodes. 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 [30]. The global criteria, defined by normalized cut, captures information of total dissimilarity between node grouping and also total similarity within the groups. The segmentation algorithms based on this criteria can achieve a computational complexity of $\mathcal{O}(N^{\frac{3}{2}})$ [18]. While the algorithm has had some success [24] we prefer SLIC for 3D images due to its lower complexity of $\mathcal{O}(N)$.

Another graph approach was proposed by Fezenszwalb and Huttenlocher [7] 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 pixel nodes. 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 direciton of the nearest stationary point of an underlying density function which finds the mode of the distribution [4]. It has also been shown that this algorithm is equavalent to the Nadaray-Watson regression kernel. Again this algorithm 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 [34]. 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 quiet fast this algorithm also can not control the number or compactness of the resulting super-pixels.

1.4.2 SSVM Solver Alternatives

The most common way of solving SVMs are the stochastic gradient descent algorithms. This class of algorithms has a direct generalization for problems with objective functions which are not differentiable, the stochastic subgradient descent algorithm which which has found some applications in structured learning [28] [33] [29]. Stochastic subgradient descent has a good convergence rate of $\tilde{O}(\frac{1}{\varepsilon})$ and like BCFW only requires on max-oracle call. But applying line search to an objective that is not differentiable may result in convergence to a suboptimal point. Therefore the user must specify a good sequence of step sizes [28]. This additional tuning parameter of the stepsize sequence is not required for BCFW because we can calculate the optimal stepsize per iteration.

1.4.3 Alternative Software Packages

NiftySeg is a 3D Image segmentation framework targeting brain imagery. It uses EM based segmentation and has good features implemented for brain tissue and performs this fast [3]. While NiftySeg has gotten good results in its specialization it can not run on a distributed cluster.

Another specialized open-source framework was published in 2013 named NIRFAST [11]. It is specialized for MRI images and utilizes blood oxygen saturation, water content and lipid concentration to construct the image segments. But again 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 [20] and [17]. While we will reproduce the relevant information here [17] 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 w to correctly classify all data based on their features $\Phi(x)$

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

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

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

Considering that re-scaling w and b does not effect the distance of any point to the decision plane we can scale them such that $t_n(w^T \Phi(x_x) + b) \geq 1$ holds for all points. we can drop the min_n term from the optimization objective because that there will always be at least one point for which the above statement is an equality ,due to that point being the closest point to the margin. By this transformation we arrive at the canonical primal definition:

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

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

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 dataset was or what w was initialized. The SVM has one clearly optimal solution because maximizing the margin to the closest datapoints with the constrained on all labels being classified correctly. For a detailed comparison see [23]. It is intuitive to think that choosing the decision boundary which has 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].

Alternatively to solving the primal formulation directly we can solve it in the dual. The dual formulation allows us to use kernel function because it is a simple sum of scaler's; and later we will see for our large dimensional problem the dual parameters can be very sparse which provides computational advantages. To construct the Dual problem we differentiating the Lagrangian and substitute back into $L(w, b, a)$.

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

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

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

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 exactly equivalent to the primal solution as the Lagrangian is only equal in the limit. The difference between the primal and dual scores 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 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 labeled it as A. This method of classification is computationally inefficient and unfeasible 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 must encapsulate 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 but, in contrast to the SMV, predicting requires maximizing over the possible labels $\arg\max_{y \in \mathcal{Y}_i} L_i(\boldsymbol{y}) - \langle W, \psi_i(\boldsymbol{y}) \rangle$. With a training dataset $D = (x_i, y_i)_{i=1}^n$, we estimate w by minimizing :

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

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

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

This first formulation still has exponential number of constrains due to $\mathcal{Y}_i$, 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:

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

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

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

The calculation of $\widetilde{H_i}(w)$ is also referred to as the "max oracle" because it 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 sparce representation in the dual variable vector α which is advantageous for sub-gradient optimization. With the Karush-Kuhn-Tucker conditions we can map between the dual and primal forms $W = A\alpha = \sum_{i, \boldsymbol{y} \in \mathcal{Y}_i} \alpha_i(\boldsymbol{y}) \frac{\psi_i(\boldsymbol{y})}{\lambda n}$. Computing the gradient of 2.16 we find $\nabla f(\alpha) = \lambda A^T A\alpha - b = \lambda A^T W - b$. Aditionally note that the domain of $f(alpha)$ is the product of n proabability simplicies, $\mathcal{M} := \Delta_{|\mathcal{Y}_1|} \times ... \times \Delta_{|\mathcal{Y}_n|}$. It will show later that the sparsity in α and the ability to easily move back to the primal will be 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 [8], is a quadratic programming solver with wider applications than typical projected gradient

methods as it only requires optimizing 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 linearization of f (using the current α) constrained on being inside $\mathcal{M}$.

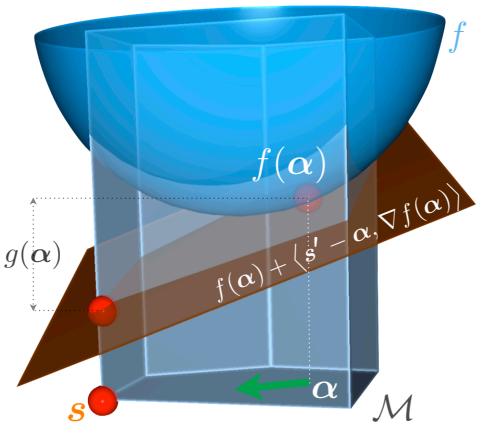


Figure 2.1

All following steps are a convex combination of a new $\boldsymbol{s}$ and the previous iteration with some step-size γ. This formulation is advantageous for problems when α is in high dimensions, as the current α can be expressed in terms of the the initial $\alpha^{(0)}$ and all $\boldsymbol{s}$ 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 α for image segmentation would never fit into memory. Additionally Frank-Wolfe has the advantage that we can compute the duality gap for free on every iteration because 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 sounds stopping criteria $f(\boldsymbol{\alpha}) - f(\boldsymbol{\alpha}^*) \leq \epsilon$. It has also been shown [6] that the algorithm converges to a ϵ-approximate solution within $\mathcal{O}(\frac{1}{\epsilon})$

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

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

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

Let $\gamma := \frac{2}{k+2}$ or optimize γ by line-search Update

$\boldsymbol{\alpha}^{(k+1)} := (1-\gamma)\boldsymbol{\alpha}^{(k)} + \gamma \boldsymbol{s}$

end

Algorithm 1: Frank-Wolfe on a Compact Domain

2.3.2 Block Coordinate Frank-Wolfe

A 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 [17] improves this to only requiring one call to the maixmization oracle in the context of SSVMs. The method described [17] and reproduced here in Algorithm 2 can be applied to any constraint optimization problem of the form

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

having a Cartesian product over $n \geq 1$ blocks as the domain $\mathcal{M} = \mathcal{M}^1 \times \ldots \times \mathcal{M}^n \subseteq \mathbb{R}^m$. The reduction in oracle calls comes from this structure in $\mathcal{M}$. This structure allows 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 $\boldsymbol{\alpha}_i \in \mathbb{R}^{m_i}$ for 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.

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

for $k = 0 ... K$ **do**

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

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

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

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

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 γ in closed form, leaving us one less tuning.

BCFW for the SSVM

For our image segmentation applications we used the BCFW algorithm 3 for SSVMs such that we only need to maintain the primal w. The equivalence between Algorithm 2 and Algorithm 3 rests on the relations : $W_s = A\boldsymbol{s}_{[i]}$ 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 $W^{(0)} := W_i^{(0)} := \bar{W}^{(0)} := 0, \quad \ell^{(0)} := \ell_i^{(0)} := 0;$

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

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

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

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

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

Let $\gamma := \frac{\lambda(W_i^{(k)} - W_s)TW^{(k)} - \ell_i^{(k)} + \ell_s}{\lambda ||W_i^{(k)} - W_s||^2}$ and clip to $[0, 1]$;

Update $W_i^{(k+1)} := (1-\gamma)W_i^{(k)} + \gamma W_s$ and
$\ell_i^{(k+1)} := (1-\gamma)\ell_i^{(k)} + \gamma \ell_s$;

Update $W^{(K+1)} := W^{(k)} + W_i^{(k+1)} - W_i^{(k)}$ and
$\ell^{(k+1)} := \ell(k) + \ell_i^{(k+1)} - \ell_i^{(k)}$;

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

end

Algorithm 3: Block-Coordinate Frank-Wolfe Algorithm for SSVM

where $H_i(\boldsymbol{y}, W^{(k)}) := L_i(\boldsymbol{y}) - \langle W, \psi_i(\boldsymbol{y}) \rangle$ is the hing 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 [17]. 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 SSVM is expressed as a quadradic 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 w. Our CRF energy is seperated into its unary and pariwise portion.

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

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

where $\kappa^D(y_i)$ and $\kappa^V_{ij}(y_i, y_j)$ 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 W^D is the column vector $[W_1^D, ..., W_K^D]^T$ then the unary term of the energy can be written as an inner product

$$D_i(y_i) = \langle W^D_{y_i}, x_i \rangle, \tag{2.22}$$

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

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

with parameters $W^V = (w_{ab})_{(a,b)\in\{1,...,K\}^2}$, 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 letting, $W = ((W^D)^T, (W^V)^T)^T$, $F^D(Y) = \sum_{i\in\nu} \kappa^D_i(y_i)$, $F^V(Y) = \sum_{(i,j)\in\zeta} \kappa^V_{ij}(y_i, y_j)$ and $F(Y) = [F^D(Y)^T, F^V(Y)^T]^T$ to define the total energy of a CRF as an inner product

$$E_W(Y) = \langle W, F(Y) \rangle \tag{2.25}$$

where ν is the set of vertecies of super-pixels extracted from the input image and ζ 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 simplest 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

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

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

$$E_W(Y) = \langle W, F(Y) \rangle \tag{2.27}$$

Data-dependent Pairwise CRF Model

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

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

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

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

The function $\varpi(x_i, x_j)$ is some kind of 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 [31] and [19].

Average Intensity Difference For all neighbouring nodes A and B we calculate the average intensity (for RGB its also the grey-scale intensity) and simply take their squared difference. Quantiles are computed for the entire dataset 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 its also the grey-scale intensity). The values are calculated for all neighbours in the training set and sorted to find quantil 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 its also the grey-scale intensity). The values are calculated for all neighbours in the training set and sorted to find quantil boundaries which result in equal mass being binned to each group. The number of bins depends on argument ”numDataDepGraidBins”.

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

Uniqueness in Opposit Neighbourhood The internal super-pixel mean intensity $AvgI()$ and the average mean in its one hope neighbourhood $AvgNeigh()$ is calcualted for all nodes. For every 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 quantil 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 as defined by $\arg\max_{\boldsymbol{y}\in\mathcal{Y}_i} L_i(\boldsymbol{y}) - \langle W, \psi_i(\boldsymbol{y})\rangle$ and, depending on the structure of $\boldsymbol{y}$, it can be intractable. When modeling the image segmentation problem with a pairwise CRF as in 2.28 then a naive maximization by calculating the energy for all possible $\boldsymbol{y} \in \mathcal{Y}_i$ would be intractable because $|\mathcal{Y}_i| = R^{|}\boldsymbol{y}|$ where R is the number of possible labels and $|\boldsymbol{y}|$ is equal 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 include 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 dataset. 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 where in 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 learned 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 baysian networks which can be expressed as trees [15]. We can transform a baysian network if it is a tree into a biparti factor graph where each variable is a class variable in the baysian network and each factor is some probabilistic relation between variables (and hence has an edge to all those variables). If label a biparti factor graph with factors $f_A, f_B, f_C ... = F$ and variables $x_1, x_2, x_3, x_4 ... = X$. Then we can quickly see that the marginal probability of a leaf is indipendent of the remaining graph variables given all factors it is connected too. The Sum-Product Algorithm then 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 descendant

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

More percisely 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 μ 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 a 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 [25]. 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

SuperPixel 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 consider it one datapoint with its features and with one label. If we were to use a simple grid of square super-pixels we will have a large chance of summerizing features on a super-pixel which includes significant portions of two or more labels but 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 dataset cleaner overall. But the smaller we choose a super pixel the larger the resulting graph gets and the graph size has an exponential effect on the time the max-oracle function requires to decode. Additionally if the super pixels are too small the unary features extracted from the 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 where to run simple k-means on this space with a euclidean distance function we would run into problems do to 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-mean. In terms of complexity

the key difference to K-means cluster is the assumption which allows SLIC to have a $\mathcal{O}(N)$ is 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 general object Datatype to be saved in each pixel location. If the user would like to cluster something other than RGB or greyscale images they must define distance functions on their Datatype. For greyscale $d_{data} = \sqrt{(I_k - I_i)^2}$ where I_k is the greyscale 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.

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 $3x3$ neighborhood, to the lowest gradient position. **while** $E \leq threshold$ **do**

  for *each cluster center* C_k **do**

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

  end

  Compute new cluster centers Compute residual error E L1 distance between previous centers and recomputed centers

end

Enforce Connectivity

Algorithm 4: SLIC Super-pixels

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. As the size of the super pixel is increased the time required for training decreases exponentially, see Figure 3.1.

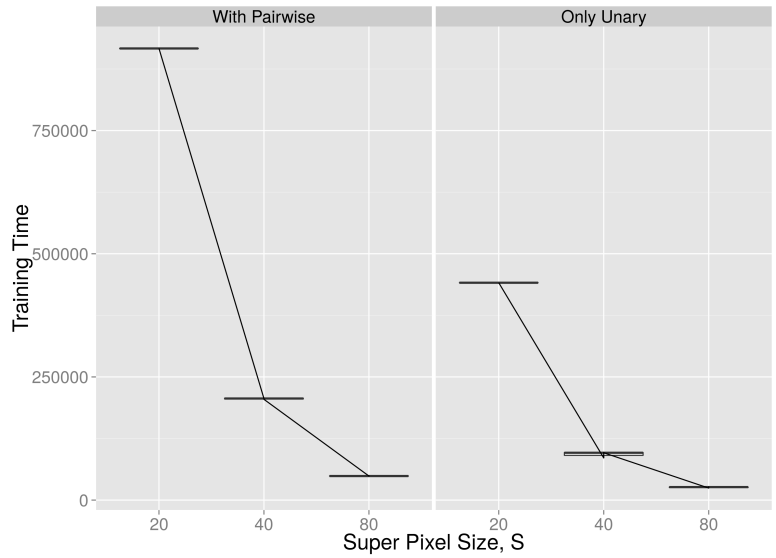


Figure 3.1: The primary time consuming function in our BCFW is the max-oracle. The difficulty of the problem depends on the number of possible $\boldsymbol{y} \in \mathcal{Y}_i$, by increasing the size of the super-pixels we exponentially decrease the set of possible $\boldsymbol{y}$ and hence also the max-oracle. As expected the effect is more prominent with pairwise models.

3.3 SLIC vs Naive Squares

When directly comparing the test scores of an 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 dataset, see Figure 3.3.

3.4 Running ScalaSLIC

ScalaSlic can be used as a standalone package on any kind of data which is saved on spatially 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 DataType can be an RGB triple, a single greyscale byte but it could also be for example a series of RGB values from a video sequence. While the DataType is technically free from the scala compilers perspective, practically one must be able to define the following functions: $distFn : (DataType, DataType) => Double$, which sim-

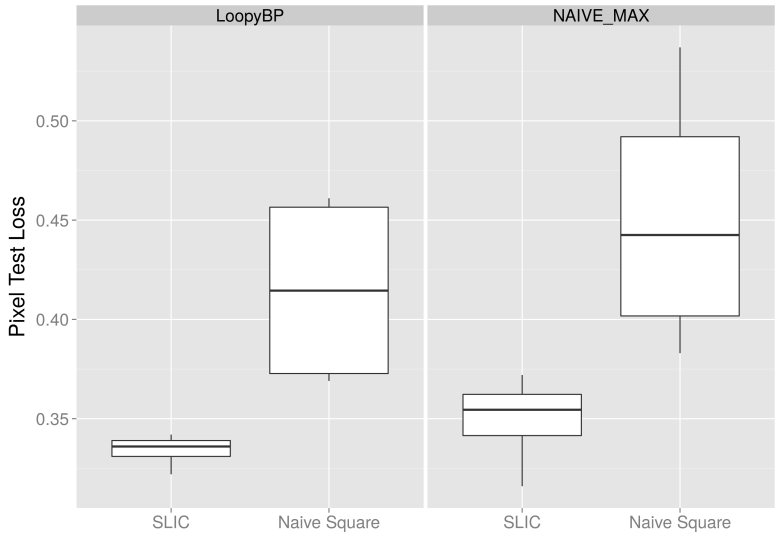


Figure 3.2: Comparing the per pixel test loss using either SLIC or Square super-pixels. The experiment was repeated for Unary model and a Pairwise model solved with Loopy Belief Propagation. Data (SynthData see A.5, WhiteNoise:0.40 SquarImgSize:30 OsilNoise:0.40 SupSqrColorShift:0.0 SuperPix, S:30, M:30,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

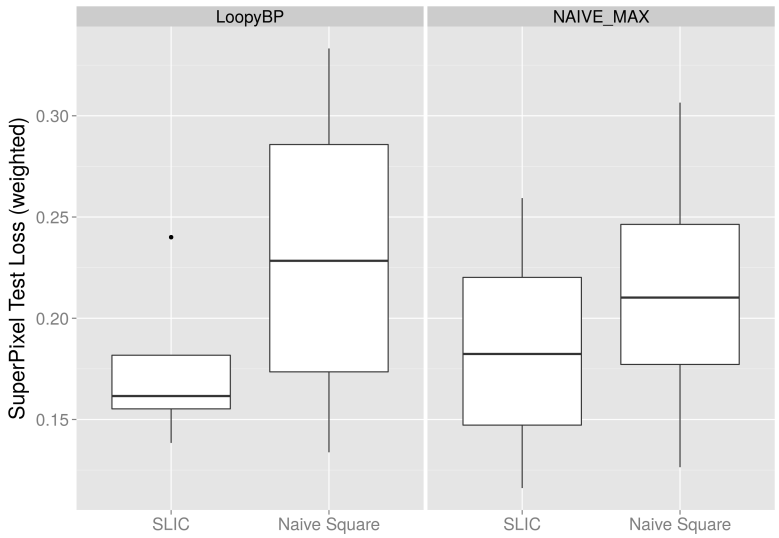


Figure 3.3: Comparing the per class frequency per super-pixel test loss using either SLIC or Square super-pixels. The experiment was repeated for Unary model and a Pairwise model solved with Loopy Belief Propagation. Data(Single Channel 3D Nuclei Images [9])

ply measure the distance between two pixels. The functions $rollingAvgFn : ((DataType, DataType, Int) => DataType)$ and $normFn : ((DataType, Int) => DataType)$ are used for the cluster update step to get an average point in this "DataType" space. For here are the functions we use for RGB $(a : (Int, Int, Int), b : (Int, Int, Int)) => sqrt(Math.pow(a._1 - b._1, 2) + Math.pow(a._2 - b._2, 2) + Math.pow(a._3 - b._3, 2))$, $distFnCol = (a : (Int, Int, Int), b : (Int, Int, Int)) => sqrt(Math.pow(a._1 - b._1, 2) + Math.pow(a._2 - b._2, 2) + Math.pow(a._3 - b._3, 2))$, $sumFnCol = (a : (Int, Int, Int), b : (Int, Int, Int)) => ((a._1 + b._1, a._2 + b._2, a._3 + a._3))$ and $normFnCol = (a : (Int, Int, Int), n : Int) => ((a._1/n, a._2/n, a._3/n))$. The final mandatory input for ScalaSLIC is "S" which determens the initial grid spacing of the superpixels and thereby also setting the initial number of cluster centers and partially how large the superpixels will be. How large the super pixels will be is also modulated by the parameter "M" which specifies the compactness of the superpixels. If M is not specified we go for the alternative approach of normalizing distances by the max distance in the each superpixel set.

3.4.1 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: $Distance(P, C) = \frac{M^2}{S^2}\sqrt[2]{(x_k - x_i)^2 + (y_k - y_i)^2 + (z_k - z_i)^2} + \sqrt{(r_k - r_i)^2 + (g_k - g_i)^2 + (b_k - b_i)^2}$. LABColor-space can be used optionally instead of RGB. For an illustration of the effect of paramater *M* on the segmentation see Figures 3.4, 3.5 and 3.6

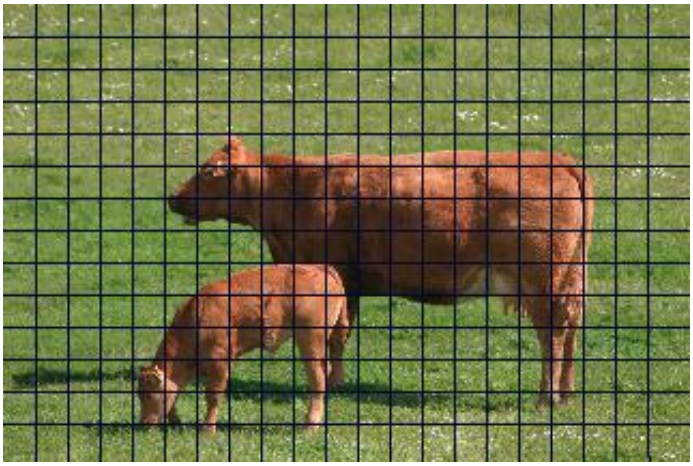


Figure 3.4: SLIC example segmentation on a single 2D MSRC image. With parameters: S=15, M=1000

Figure 3.5: SLIC example segmentation on a single 2D MSRC image. With parameters: S=15, M=45

Figure 3.6: SLIC example segmentation on a single 2D MSRC image. With parameters: S=15, M=14

Do to the inherint diffiuclty of seeing through a block of solid tissue we will visualize the effect of parameter M by only drawing the voxels labelled as forground, this 3D dataset is a binary classification problem and hence can be displayed like this.

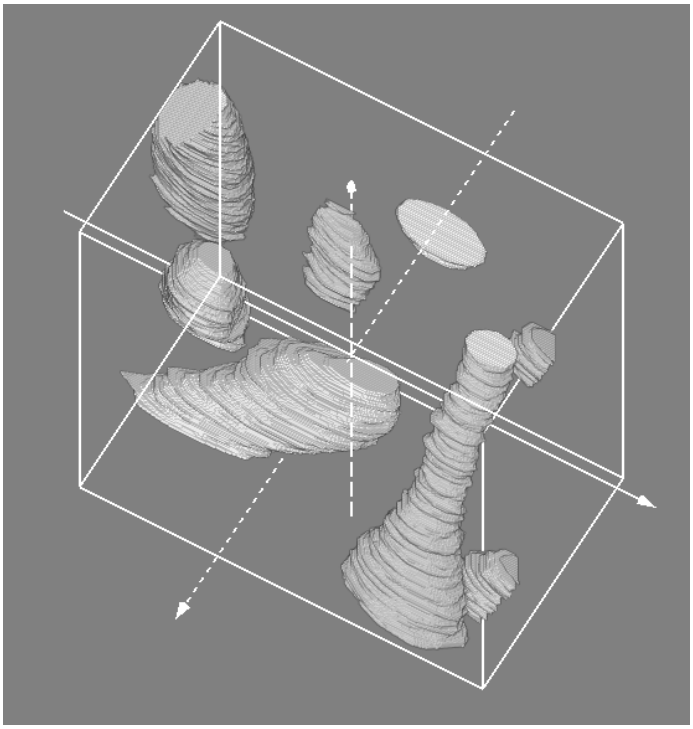


Figure 3.7: The original per pixel ground truth labeling with background transparent and foreground textured to have a surface. Data used in this image is a subset of the EM Mitochondria dataset [13]

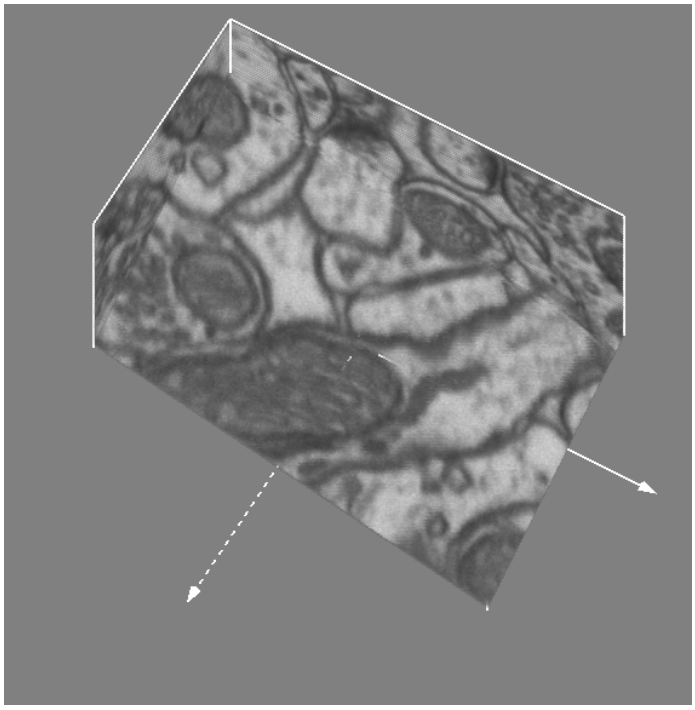


Figure 3.8: The figure shows a cross sectional cut of the raw data used to construct super-pixels in Figure 3.9 and 3.10. Data used in this image is a subset of the EM Mitochondria dataset [13]

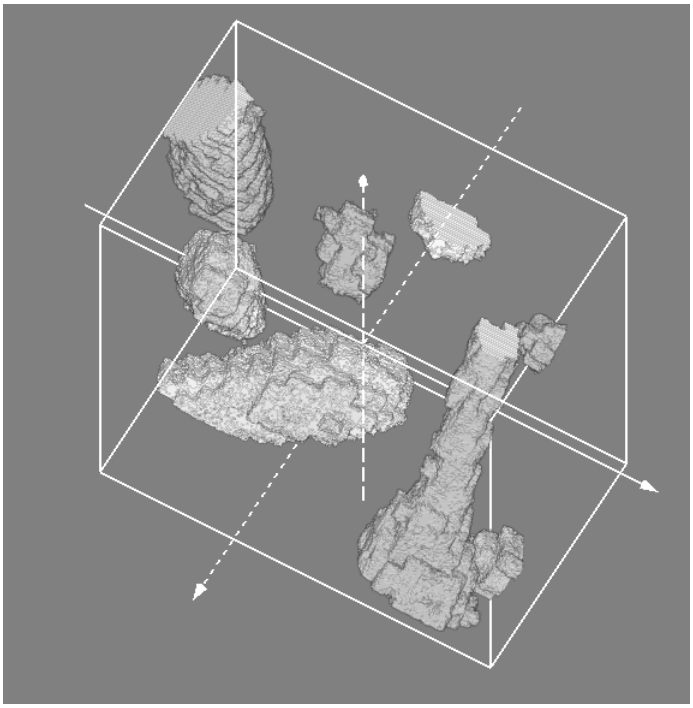


Figure 3.9: The figure shows the textured boundaries of the volumn encompassed by the super-pixels which where labled as being mitochondrial foreground in their graph representation. This image varies from Figure 3.10 in that it used a higher $M = 50$The original per pixel ground truth labeling with background transparent and foreground textured to have a surface and hence results in more compact super-pixels. Data used in this image is a subset of the EM Mitochondria dataset [13]

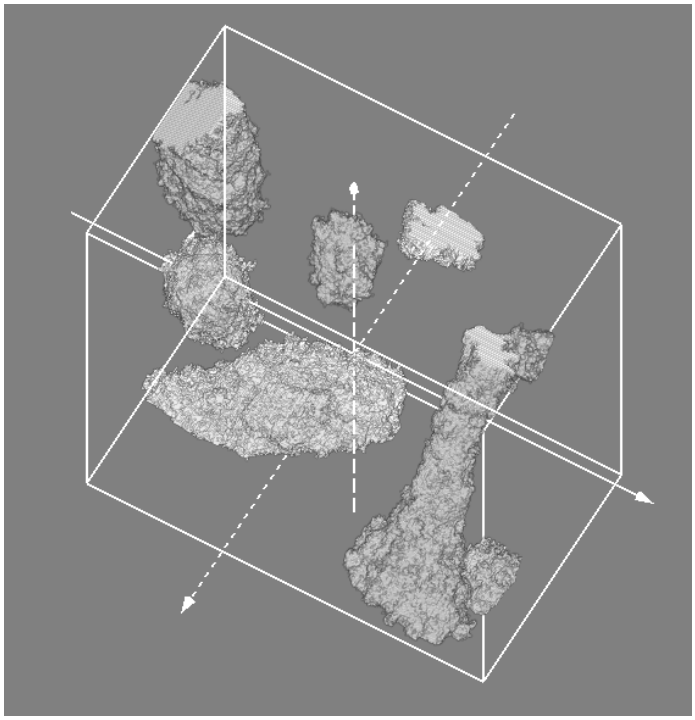


Figure 3.10: The figure shows the textured boundaries of the volumn encompassed by the super-pixels which where labled as being mitochondrial foreground in their graph representation. This image varies from Figure 3.10 in that it used a lower $M = 9$ and hence results in less uniformly shaped super-pixels. Data used in this image is a subset of the EM Mitochondria dataset [13]

@inproceedingsspark, title=Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, author=Zaharia, Matei and Chowdhury, Mosharaf and Das, Tathagata and Dave, Ankur and Ma, Justin and McCauley, Murphy and Franklin, Michael J and Shenker, Scott and Stoica, Ion, booktitle=Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, pages=2–2, year=2012, organization=USENIX Association

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.

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. 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 benificial to use a more complex data dependent model but the score variance is high in our data. Surprisingly we found that with a $\lambda = 100$ Mean Field performed close to LoopyBP. See Figure 4.1 for details.

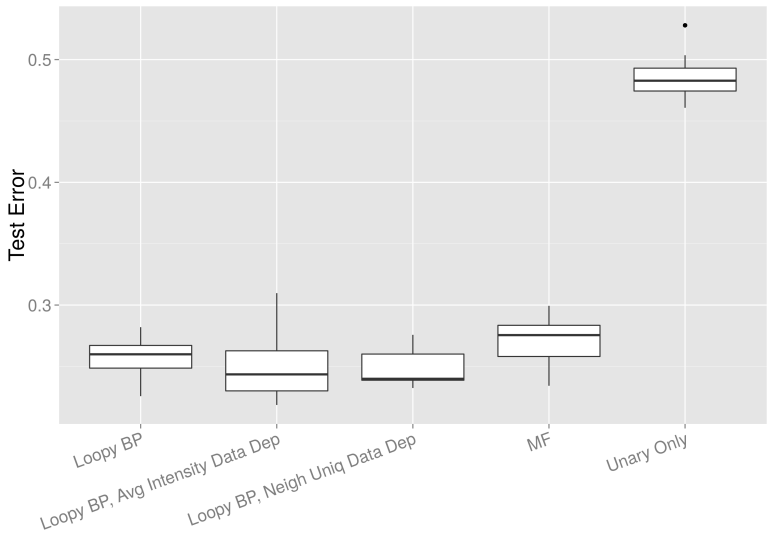


Figure 4.1: Over 30 rounds we trained the SSVM with different max oracle functions and using different models. LoopyBP and MF where trained on a simple pairwise model as in 2.4.1. Naive Max oracle was trained a Unary only model see 2.4.1. Additionally we trained two data dependent pariwise models (see 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” calcluates 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 Labeling split into 82^3 cubes with super-pixels of size $S = 10$ [13])

4.2 Transition probability table sparseness relation to Pairwise advantage

During our exploration of many different datasets we saw high variance on whether the pairwise models increases accuracy or decreases as compared to the baseline naive unary max oracle. To examin 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 label A for all 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 occure 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 use this information. Once below 0.3 the pairwise benefit starts decreasing again because the likelihood of the 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"

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

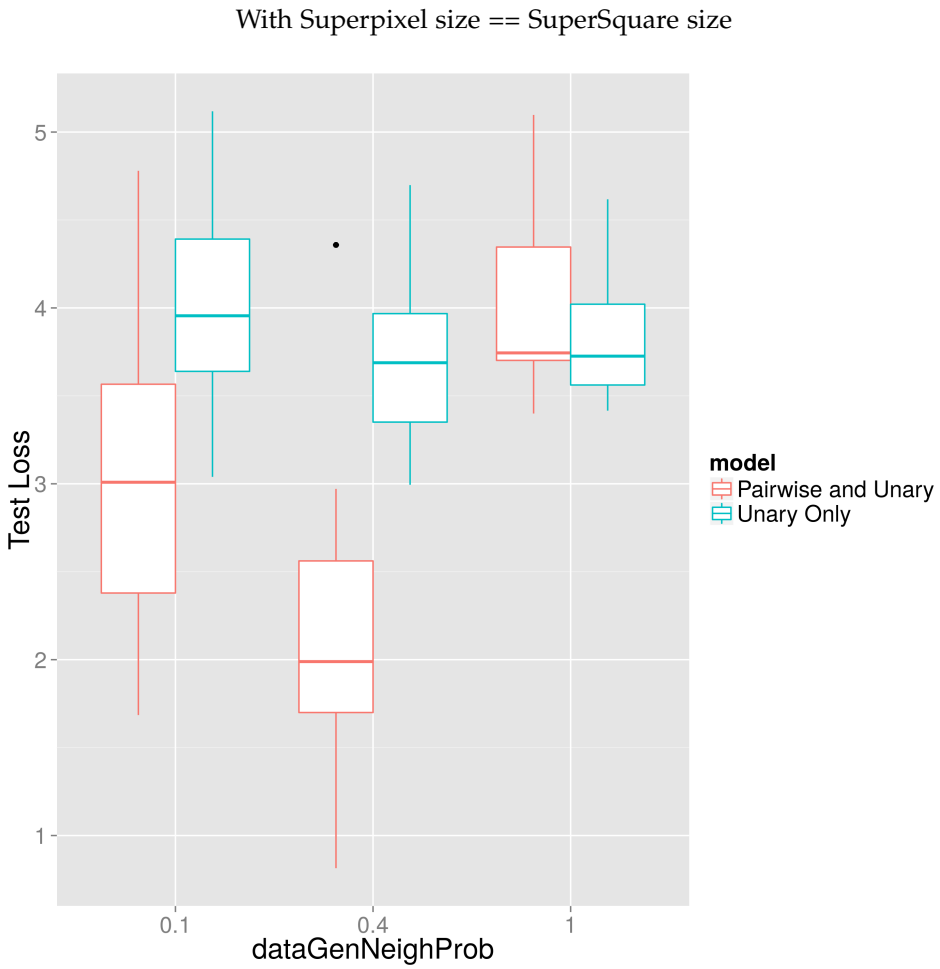
Learned Pairwise Weights

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

goes to 1 all classes can be adjacent to all others and as it approaches zero the image becomes more and more background because that label is reserved as a fallback when all other labels are not allowed. In Table 4.1 the reader can inspect columns of the transition probability above to the learned weights corresponding to the pairwise potentials of the same label pairs below, noticing that those label pairs which have the highest transition probability also have the highest weight in the corresponding learned pairwise potentials. In contrast table 4.2 does not have this correspondence, as the pairwise term is not particularly helpful with data that has such a uniformly distributed transition probability table.

With the sparse transition probability explanation for the pairwise advantage in mind we can construct a dataset which should be even better, by making the super-pixels exactly equal to the supersquares. This configuration should result in transition probabilities from the label to itself being as

Figure 4.3: Test Error vs Label Transition Sparsity



(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 superpixels 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)

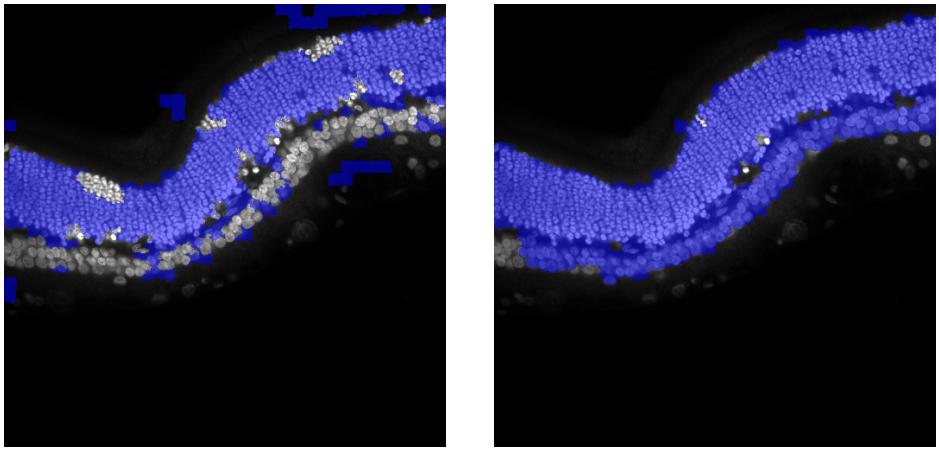
sparse as the rest of the matrix. In Figure 4.3a we see the expected result, that the distance between the pairwise and unary model increases as we farther increase the sparsity of the transition probability matrix.

4.3 Smoothness, as an visualization of the pairwise improvement

By visual inspection of the predicted output of our pairwise model we subjectively saw a trend of the pairwise models performing better when a smooth solution is more likely than disjoint singular labels.

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 superpixels internally. The pairwise model does not count the number of neighbors which are of the same label but instead simply adds a factor based on the transition probability between two adjacent classes. Hence we can still see a few superpixels standing alone but their number is significantly reduced.

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 cells which are only slightly disconnected from the main body. These nearby easily mistaken objects are all grouped into the main object in using the pairwise term which turns out to give a worse score than the few unary mistakes made on that band lower band. Hence we must be mindful of the way we are including the spacial relations into our model with some datasets we 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 is to set the compactness parameter for the SLIC preprocessing very low so that thin long superpixels could be placed between the two groups of classes.



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

Figure 4.5: Per-superpixel predicted labels are superimposed on the raw image. This Nuclei detection dataset only has foreground and background hence we made only colored the Nuclei label 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 [9])

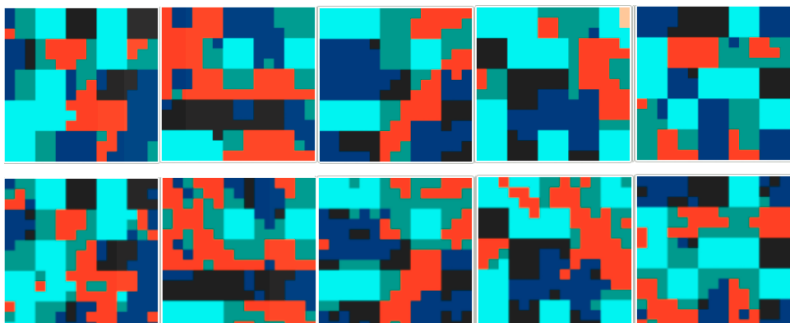


Figure 4.4: This figure shows the predicted labels per pixel for a pairwise model in the top row and a unary only model in the bottom row. By visual inspection the reader can see that the top pairwise model is more smooth in that the blobs of labels are more connected and generally in larger collections.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

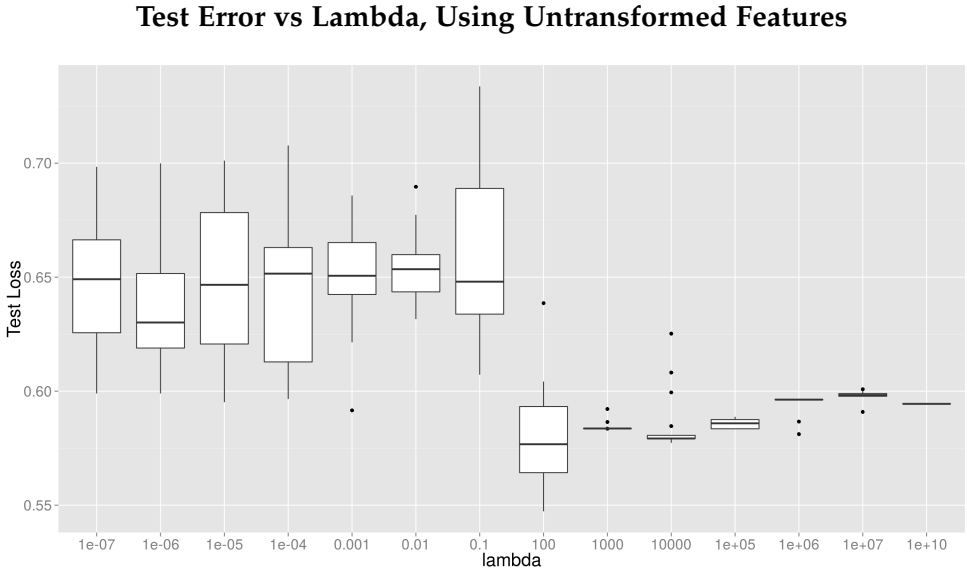


Figure 4.6: We can see a clear separation between lambdas which are not strict enough to result on good convergence between 0.1 and 100. Additionally we see that lambdas which are too strict result in lower accuracy again do to them not allowing full exploration of the relations between the features from 10000 to 1e+10 ;Axis (Test Error (lower is better) vs Lambda); Data(MSRC version 2 [32])

4.4 Impact of the Regularizer

Out of all the tuning parameters "lambda" needs to be adjusted most carefully, as it determines the freedom W has. In the optimization function λ 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 8). 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 over rounds, Figure 4.7. We can see that for low lambdas the Structured Hinge loss $\widetilde{H}_i(w)$ does not decrease over time so in these experiments they are failing to gain useful information in the W.

To gain farther understanding about the effect lambda has on convergence we can consider the effect it has on the $norm(w)$ over time. See Figure 4.9 which shows us how the norm of 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 lambdas. 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

Structured Hinge Loss over time, faceting on different lambdas

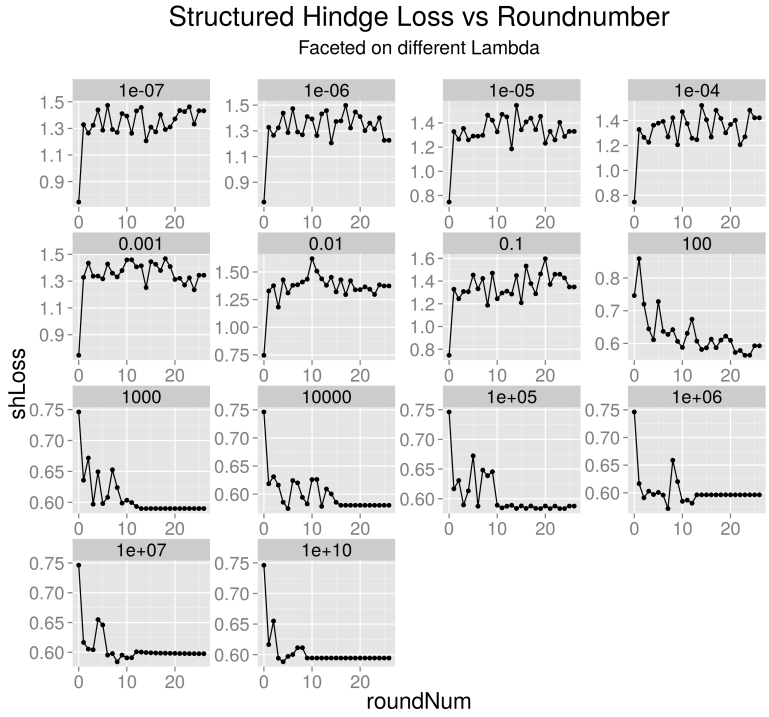


Figure 4.7: A clear difference can be seen between low lambdas up to 0.1 which do not have a negative trend and also do not seem to converge in terms of structured hinge loss, while lambdas 100 and above have a clear negative trend over time and as lambda gets larger they also converge faster. It should be noted that each experiment starts with the same w and hence also starts with the same structured hinge loss of 0.7462. Although the vertical axis is free in this graph we can see that with lower lambda each round results in a larger change in the model as expressed by the change in structured hinge loss.; Data(MSRC version 2 [32])

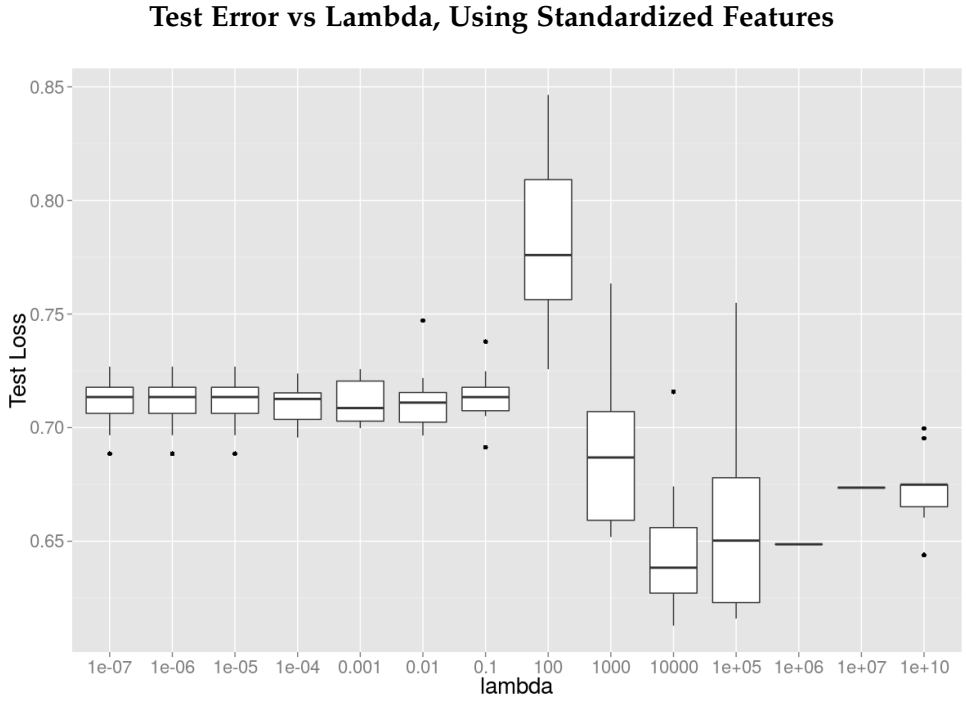


Figure 4.8: Similar to Figure 4.6 we can see that low lambdas do not converge properly and hence have significantly higher loss. In this experiment the lowest loss is at 100000 several orders of magnitude higher than in Figure 4.6 which ran the same experiment only changing the feature standardization setting.; Data(MSRC version 2 [32])

perception which still has the ability 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 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 $norm(w)$ can appear to be staying constant while the direction of w changes continuing to improve accuracy. Still since we change 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 Figure4.10 and we observe a decreasing trend as rounds increase.

Lambdas 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 Figure4.10 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 high lambdas the first term of the dual would drop out due to the squared A with a λ in the 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 lambdas 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 α 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 datapoint. If we where to consider this solution in the special case of the binary SVM w would a vector which if extended would go through the centroid of both classes. The figures used in this section where based on data which was infact binary and if we look at Figure 4.8 which used the MSRC dataset with 24 labels the high lambda solutions are much farther away from the optimal than with binary labeled data.

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 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 as compared to the single pairwise term in the simple model. To test this hypothesis we reran this experiment with larger suboptimal lambda selections. In Figure 4.12 we see the Structured Hingle 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; here we chose compare the Structured Hinge loss because, in contrast to the training error, this is the function which the SSVM is optimizing and hence also a more direct link to the $norm(w_{pairwise})$ curve mentioned above. Preliminary results with high number of rounds ≥ 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

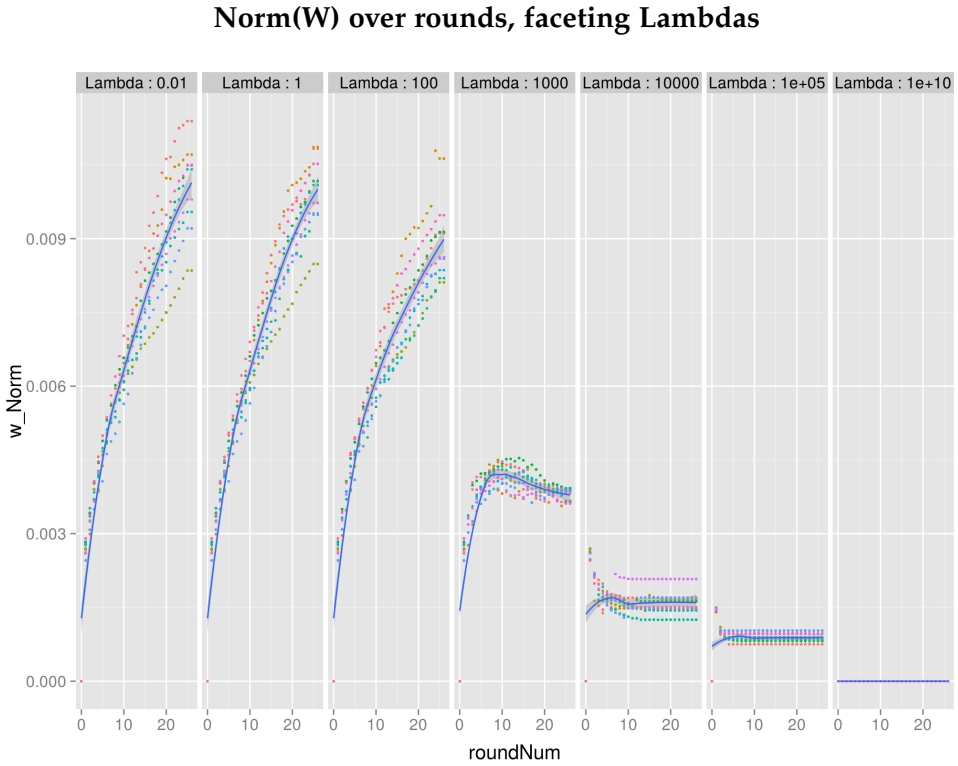


Figure 4.9: The change in the Norm of the weight vector W over time is a good indicator of the algorithms convergence depending on lambda. When Lambda is 1 we the augmented hinge loss will prefer W which have the smallest amount of lass but once the loss can not be improved significantly anymore it will start to choose W which with a larger margin which is the same as a lower norm. The smaller λ 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 W does not have an expected trend but one can still see if the algorithm converged or not by how much it is changing. Data(Single Channel 3D Nuclei [9])

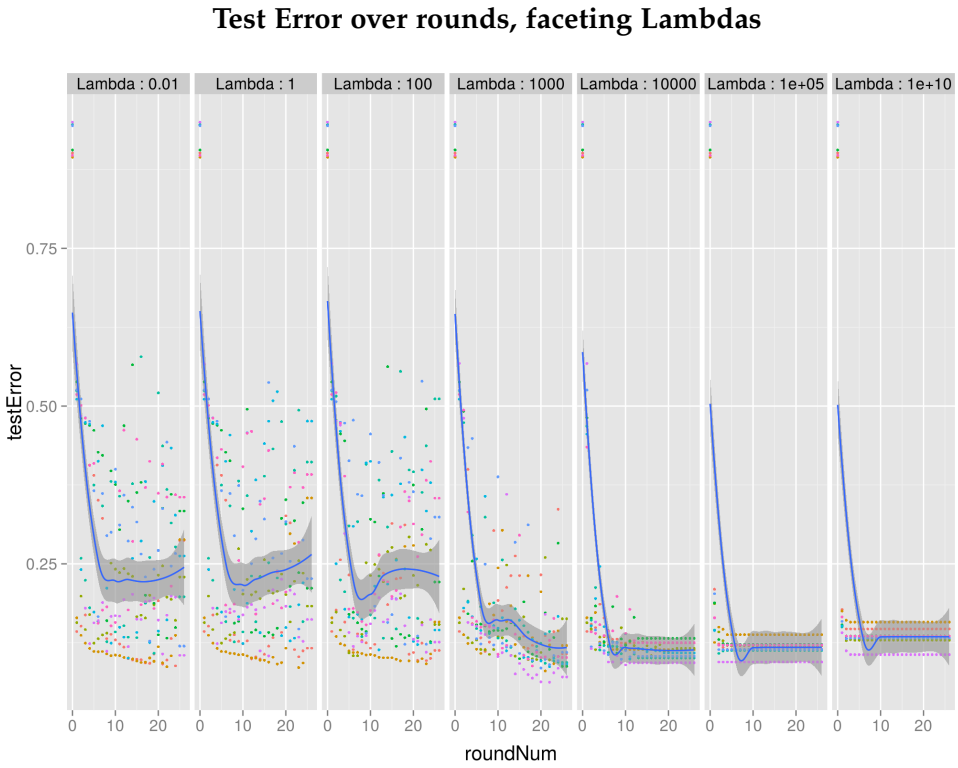


Figure 4.10: Looking at the curve of Test Error over time with different lambdas we can see that higher lambdas converge faster and if one sets lambda two low it does not seem to converge at all. Data(Single Channel 3D Nuclei [9])

system which would allow for different regularization of the pairwise term seperatly from the unary term.

4.6 Max-Oracle Methods

Amongst the three Max-Oracle methods for Pairwise CRF's, MPLP, Loopy Belief Propagation and Mean Field we see significant performance differences, see Figure 4.13. But only LoopyBP manages to outperform the unary model using the exact Naive_Max oracle. In this experiment we used rather high max iteration limit for MPLP which we expected to results in the best score because Loopy Belief Propagation only ran for 10 iterations and Mean Field makes assumptions which should result in loss of some information. But it turns out that LoopyBP had the best score and even had the lowest cpu time to achieve that score among the pairwise models, see Figure 4.14 for timing.

Change this section to correspond to the first paragraph of results

Change this to a series of experiments comparing LoopyBP and MF with different number of iterations !!

maybe Include statistics from OT's test sweet on synth data

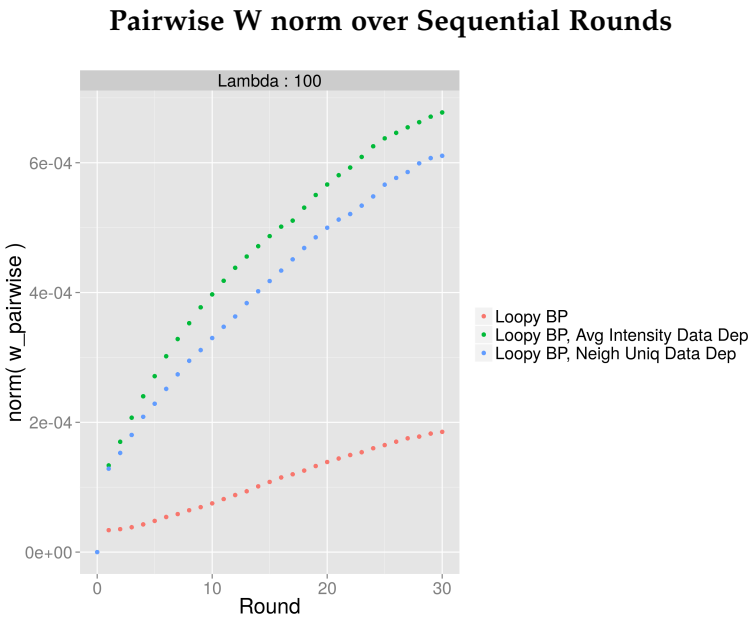


Figure 4.11: The above graph displays the norm of the pairwise portion of the weight vector. The weight vector is always initialized to zero at round zero. The curve of the norm of w can indicate convergence behavior. In this experiment it appears that Loopy BP (the Simple Pairwise model) is converging faster than both data dependent pairwise models. Data:(EM Mitochondira Labeling split into 82^3 cubes with super-pixels of size $S = 10$ [13])

Structured Hinge Loss over Time, including Linear Trend Line

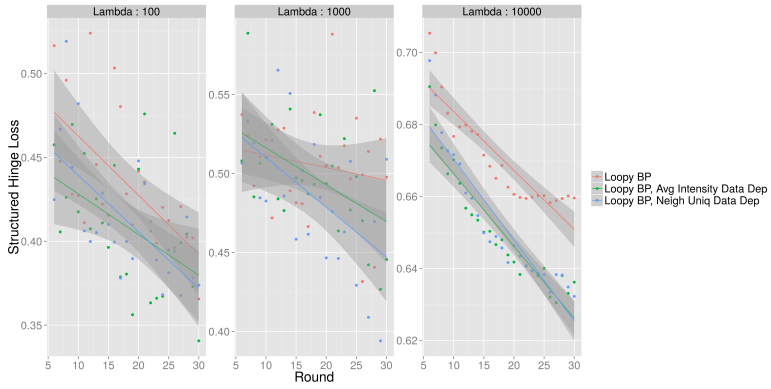


Figure 4.12: This Figure displays the Structured Hinge Loss ($\max_{y \in \mathcal{Y}_i} L_i(\boldsymbol{y}) - \langle W, \psi_i(\boldsymbol{y}) \rangle$) over time of the different pairwise models. With the added linear regression line we can see that as we increase Lambda the difference between the simple pairwise model to the data dependent models increases. Higher lambdas 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 $S = 10$ [13])

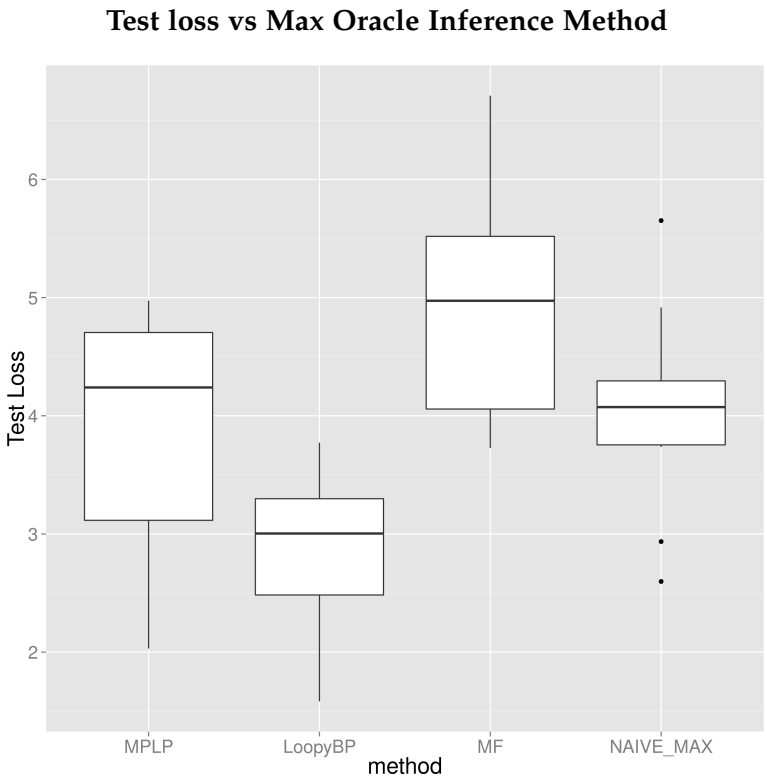


Figure 4.13: On this synthetic dataset we found that LoopyBP had significant better accuracy than all other methods. And also that MPLP occationaly got better results than the Naive-Max inference method but the variance was too high for this to be a sigificant difference. Hence we can conclude that only LoopyBP outperformed the baseline of Naive-Max max Oracle. Data (WhiteNoise:0.4, SuperSquareSize:60, OsilNoise:0.4, SuperSquareShift:0.0, S:5, M:5)

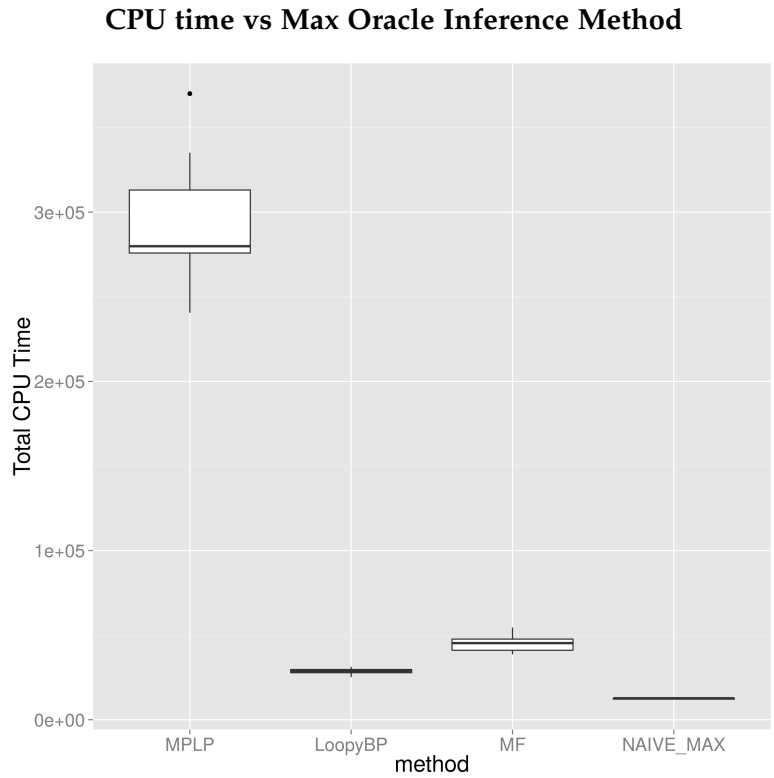


Figure 4.14: Clearly MPLP takes the longest time at its default iteration rate, but this cpu time is not warrented as we can see in Figure 4.13 loopyBP outperforms MPLP and now we can see it does this with lower cpu time. Infact the only method which has lower cpu time than LoopyBP is the Naive method but this one also gets worse accuracy. (WhiteNoise:0.4, SuperSquareSize:60, OsilNoise:0.4, SuperSquareShift:0.0, S:5, M:5)

Chapter 5

Distributing Workload

The dissolve framework which we used to solve the SSVM was written with distribution in mind by

5.1 Single Node, Multiple cores

Spark can be used to distribute the inference work over a cluster but also it allows for local parallelization without writing multi-threaded code. In some computing environments it may be cheaper to use one 36core 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 36core machine other than the ram needed for the executor it must have 18432mb available for all the drivers. When using *sbtrun − mainch.ethz.dalab.dissolve.examples.neighbourhood.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 config.

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. When running dissolve-struct locally and we found that the system scales well. 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 even though the node then needs to switch out cpu work from the background linux system and the spark framework. 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). At one executor there is not difference, at two executors

we have a jump because this is when merging of results first occurs then as the number of executors increases the 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 farter ensure that the oracleFn is being scaled aswell as we thing 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. 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) [10].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 made 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 [10].

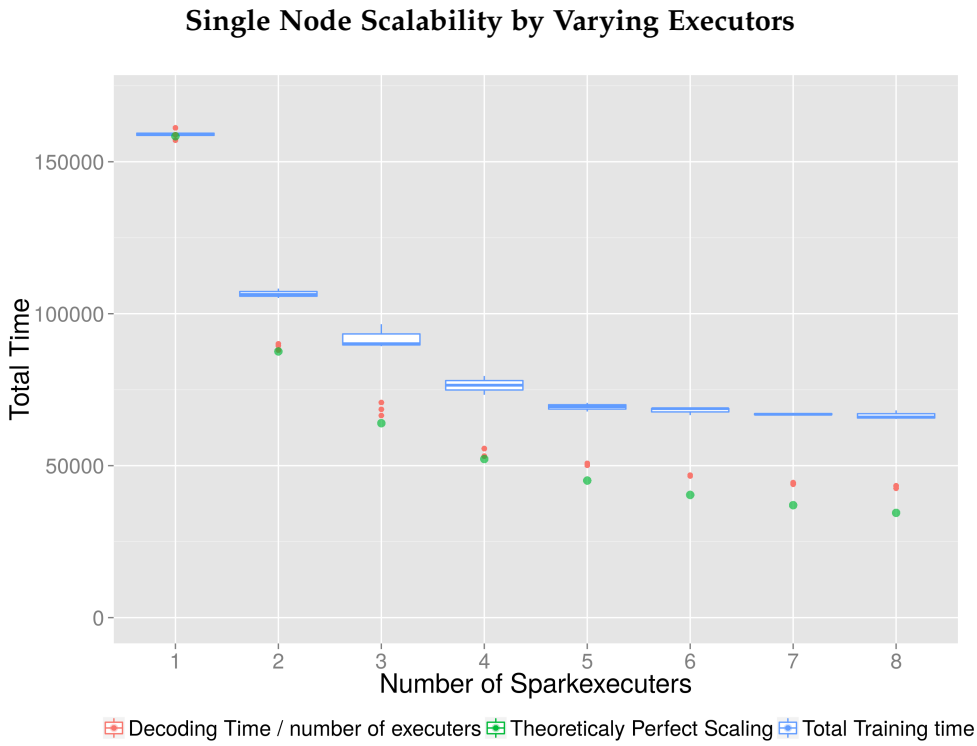


Figure 5.1: In blue we see the total time to train the model as measured on the drivers node, this time consists mostly of time spent in the oracleFn but also there is a constant overhead of 16000ms and for the experiments with greater than one executor we have a significant amount of work dont on a 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 experimetn but with more loopy belief iterations to get a more precise max oracle decoding. In contrast 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.

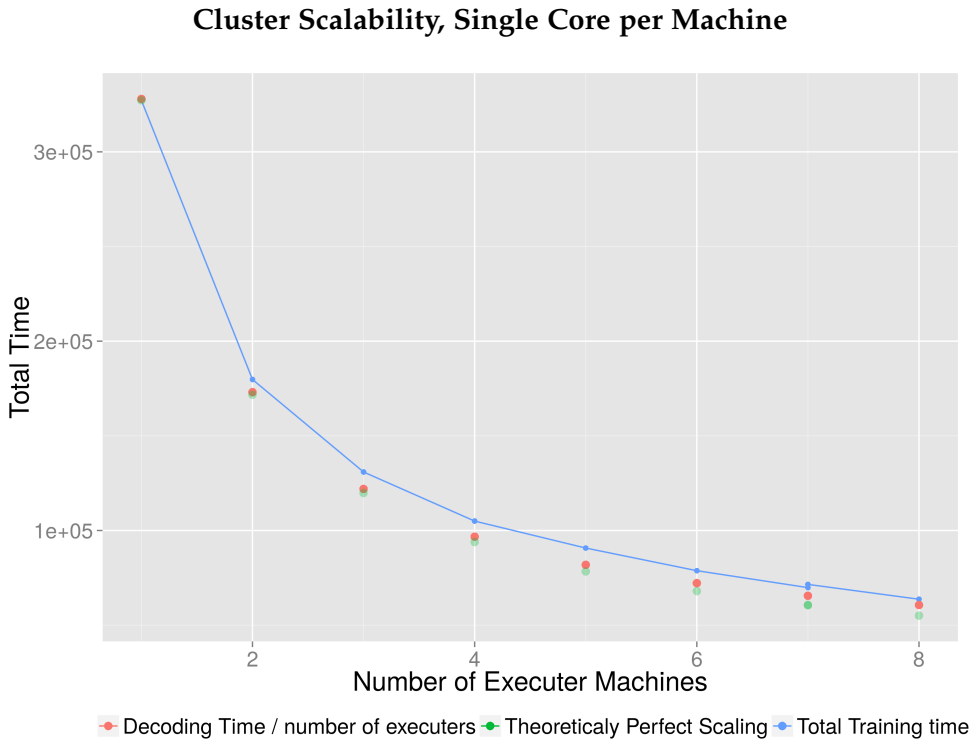


Figure 5.3: This Figure displays the total training time required for the entire MSRC data set on a varying number of m3.large AWS virtual machines each forced to use just one core. We can again see a good scaling curve as the true train time is close to the expected train time if there was no delay in context switching (Theoretically Perfect Scaling). As expected 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 we considered our segmetnation accuracy acceptable for the simplicity of our features we were expecting the pairwise models to singificantly improve performance over the unary models. As explained in the results section 4.3 we find that, as expected, the pairwise do produce much smoother results but this is not captured in the loss function and hence on some datsets the unary gets higher accuracy even if the resulting segmentation includes a single super-pixel labled as sheep inside a larger cluster of cow labels. We suggest for future work to develop a new loss function to use to evaluate the accuracy on the test set which captures the natural plausability of the output produced. In many applications it is more important for the segmentations to be biologically plausable than having the highest per pixel match to the ground truth . Still with the simple weighted loss function used for all our training we achieve satisfying results with very good scalability and preliminary reasearch has shown that the accuracy can be vastly improveed by using more sofisticated features.

citation needed

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 preinstalled features to include Ray projection features **??**. And recently deeplearning features have found a lot of success on natural image segmetnation problems [14]. These deeplearning 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 sparese 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 inorder to keep this package simple as it will be published open-source speratly from the image segmetnation. But in practice we found that the preprocessing takes a significant amount of time and could easily be destributed on a per image basis. Once this preprocessing step is distriubted one could also expand the distance measure to include more complex features than just the LAB space distance.

6.3 Expansion of the SSVM

One clear implrovement we have liked to test would be to transform the dual problem into a kernel space because it would allow us to find a non linear decision bounary. While it is clealry advantagious from a learning theory perspective it would come at additional computational complexity because to get W for synchronization between nodes one would have to sum up all the support vectors choosen thus far. But it is worth attempting empriically because the making the classes more easily speratble will make the max-oracle problem easier and hence one could possible reduce the number of iterations used for LoopyBP or Mean Field.

6.4 Improving Automation

As we described in section 4.4 the lambda paramater is critical for good convergence of BCFW. The user is advises to always rerun their experiment with lambda constants at several scales to find the best range. Lambda could easily be selected automatically by minimizing the crossvalidation error on the training set. Additionally we explaine the relevance of compactness paramater M for the super-pixel preprocessing described in section **??**. The paramater 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

When using our provided data importing functions we expect the data to be stored in two folders one for the raw image files and one folder for the ground truth mask. These folders must be labled ”Images” and ”GroundTruth” respectively. Both raw images and ground truth can be in .bmp, .tif or .png formats but the format has to be uniform throughout a dataset. Additionally it should be noted that the ground truth mask is mapped to its image only by file name so in the end the images folder and the groundtruth folder must have the same number of files with the same set of names. Ground truth masks labels are read in per pixel color or greysacle value, the ground-truth mask should be considered a space indexed label mapping so each label must have exactly one value here, which also means these files should 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 where the data resides. The caches are performed for each image individually and are named after the raw image of origin aswell as some information about what options where used when creating this cache. 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 portion of the pipeline. If the user wishes to rewrite the cache without checking if there are matching flags in the files one should specify the input argument ”recompFeat=true” or the user can change the ”runName=” input argument as it will a new set of caches while leaving the old ones as is. The files ending in ”.mask” in the Im-

ages folder contain the mapping between pixel index to superpixel id. Files ending in ".graph2" contain the completed graph structure of the associated image, I.E. nodes which only save their own features and the node id's of their neighbors. Files ending in ".classCount" , ".colorlabelmapping2" and ".transProb" contain the class frequency count, map between colors used in the groundtruth files and internal label id and the transition probabilities between labels respectively. The files ending in ".labels2" contain the cache for the true labeled graph of these training dataset.

A.1.2 Feature Standardization

Standardizing features is important especially when some features are degrees of magnitude larger than others, for example when including "featAddIntensityVariance" and "featHistSize".

A.2 Features

Since our data is split up into non uniformly shaped superpixels we could not use off the shelf feature functions. We implemented some basic features like color or intensity histograms, Co-occurance matrix, and some features based on the superpixel graph structure like the sum of neighbor histograms, in neighbourhood intensity uniqueness and some others. All these features are computed after the superpixel bounds have been determined. Most of the features are extracted by once running overall pixels in the image and adding to some moving average of the feature indexed by the superpixel id of that corresponding pixel. If using our graph construction script *genGraphFromImages*() the features are specified as arguments into either $featureFn$ or $afterFeatreFn$. As the name implies afterFeatureFn is run after the graph is constructed and provides the feature functions with the graph edge information, featureFn is for unary features only. If also using our start-up main *ch.ethz.dalab.dissolve.examples.neighbourhood.runReadTrainPredict* then the feature functions are constructed for the user who only has to specify which features are desired.

A.2.1 Predefined Feature List

All below features can be specified in the runtime arguments of *runReadTrainPredict* for color, greyscale , 2d and 3d image datasets.

"featIncludeMeanIntensity" Averages all pixels assigned to a superpixel into one mean intensity. For color this mean intensity is the corresponding converted greyscale intensity. This feature may also be added to the metadata of a node such that the datadependent pairwise models can use it.

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

"featUseStdHist" Will construct non histigrams with non uniform bin sizes. Rather it initially computes a global distribution of intensities and constructs bins which should contain approximately equal datapoints. "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 dataset and are ofcourse not adjusted for new data hence the training set must be sufficiently large for the distribution to be accurate.

"featCoOcurNumBins" Co-occurance matrices are constructed again binning all pixels into uniformly sized ranges of the intensity spectrum and then counting all occurrences of a pixel binned to bin A neighbouring a pixel in bin B. Again the number of bins must be divisible by three if used on color data. One can specify exactly which kind of neighborhood should be used by default we only consider one pixel hop and no diagonals, to change this one must alter the feature function call with different *directions* input.

"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 superpixel, for color images this is again the converted greyscale value $\frac{Red}{3} + \frac{Green}{3} + \frac{Blue}{3}$.

"featUniqueIntensity" Computes mean intensities and variances if not yet computed for all superpixels then measures the number of standard deviations away from the mean of a superpixels one hope neighbourhood its own intensity is.

"featUnique2Hop" Same as "featUniqueIntensity" but using a 2 edge hope neighbourhood.

"featNeighHist" Computes histograms per superpixels as in "featHistSize" and then sums all the histograms of neighboring superpixels not include the data from the superpixels own space and then normalizes. "featHistSize" again determines the size of the bins used here.

"featAddSupSize" The count of the number of voxels assigned to a particular superpixel.

A.3 Additional Runtime Arguments

Critical

"useNaiveUnaryMax" Setting this to true will result in dissolve 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 appoximation 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 superpixels features. This model was only implemented for Loopy Beliefe Propagation decoding. See Section 2.28

"mfTemp" Specify any Double value for use as the Temperature paramater in the Mean Field decoding. See Section 2.5.1

"useLoopyBP" If set to true, Uses the Loopy B

"useMPLP"

"loopyBPmaxIter"

"useRandomDecoding"

"stoppingCriterion"

"roundLimit"

"gapThreshold" mentino gapCheck

"onlyUnary" Used to specify if the max oracle should run decoding on a CRF with only unary potentials, see Section **??**

"lambda" The regularizing parameter see section 4.4

"doLineSearch"

"numClasses" The expected range of labels.

"isColor" Set to true if the data being read in is using RGB leave false if data is grey scale.

"useClassFreqWeighting" If set to true the lossFn will not be a zero-one loss but rather every missclassificaiton is counted as one inverse class frequency. This is import to set as true if using dataset which has large imbalance in the label counts.

"superPixelSize" Set to an Integer greater than zero this is the S parameter in our SLIC implementation. See Section 3.1

"slicCompactness" Set a double value greater than zero, this corresponds to the M paramater in SLIC. See Section 3.4.1

inputArgsdataFilesDir The path to your training raw images and ground truth. In this folder we expect a subfolder named GroundTruth and one named Images. The user can also specify these seperatly with "imageDataFilesDir" and "groundTruthDataFilesDir"

"numberOfCoresToUse"

A.4 Example Run

For viewing the predicted labels we recommend UCSB's bioView3D [16].

put code for some example runs and file structures

Optional

"trainTestEqual" If set to true, the preprocessor will not reserve any data for testing but rather train on all possible data.

"squareSLICoption" If set to true, the SLIC preprocessing will simply produce square superpixels. Run time for these superpixels is only $\mathcal{O}(n)$.

"LOSS_AUGMENTATION_OVERRIDE " If set to true optimization the objective without considering loss between graph labels. As in $\min_W \frac{\lambda}{2}||W||^2 \frac{1}{n} \sum_{i=1}^{n} \max_{y in \mathcal{Y}_i} \langle W, \psi_i(\boldsymbol{y}) \rangle$

"initWithEmpiricalTransProb" Will precalculate the class transition probability table and initialize the pairwise section of the w to these values.

"dbcfwSeed" A random seed used inside BCFW setting this to non -1 value will allow for deterministic output.

"runName" A simple string used in logging and in the cache files.

"checkpointFreq"

"dataGenSparsity" The desired percentage of the ground truth which is background when generating synthetic data. See section A.5

"dataNoiseOnlyTest" If set to true the data generated will only add the specified noise to the test set not the training dataset.

"dataGenCanvasSize" Number of pixel rows/columns per image being generated. Images are always square.

"dataRandSeed" A seed value for the random number generators used in the data synthesis functions. If set to a non -1 value then the data generation is deterministic. Allowing experiments to be run over different machines without having to transfer the data.

"dataGenSquareSize" The size of the super-squares used in synthetic data generation, see Section A.5

"dataGenSquareNoise" A decimal between 0.0 and 1.0 specifying how much SuperSquareShift noise should be added, see Section A.5

"dataGenHowMany" An Integer specifying how many images to generate.

"dataGenOsilNoise" A decimal between 0.0 and 1.0 specifying how much OscillationNoise noise should be added, see Section A.5

"dataGenGreyOnly" If Set to true, the synthetic data generated will only have features in gray scale values.

"dataGenEnforNeigh" Setting this value between 0.0 and 1.0 specifies the probability with which two labels are allowed to be neighbors. Addtionally one must set "dataGenEnforNeigh" to true for this setting to take effect. see Section A.5

"recompFeat"

"maxColorValue"

"dataDepUseIntensity"

"numDataDepGraidBins"

"dataDepUseIntensityByNeighSD"

"dataDepUseIntensityBy2NeighSD"

"dataDepUseUniqueness"

"dataDepUseUniquenessInOtherNeighbourhood"

"slicMinBlobSize"

"standardizeFeaturesByColumn"

"alsoWeighLossAugByFreq"

"splitImagesBy"

"optimizeWithSubGraid"

"pairwiseModelPruneSomeEdges"

"slicSimpleEdgeFinder"

"filterOutImagesWithOnlyOneLabel"

"leaveOneOutCrossVal"

"leaveOutCVmaxIter"

"logOracleTiming"

"enableOracleCache"

"oracleCacheSize"

"H"

"sampleFrac"

"sampleWithReplacement"

"enableManualPartitionSize"

"NUM_PART"

"debug"

"debugWeightUpdate"

"debugMultiplier"

"sparse"

"debugPrintSuperPixImg" If set to true, after training is complete *ch.ethz.dalab.dissolve.examples.neighb* will print a colored overlay ontop of the input image indicating the predicted labels per pixel. This function will also print out the correct label for comparison. It assumes that

"compPerPixLoss" When using a folder *../data/debug* exists. *ch.ethz.dalab.dissolve.examples.neighbourh* this parameter determines if after training the model will be tested on the reconstructed per pixel labels in addition to the standard per pixel error reporting.

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. [1. Randomly choose a true color per label. 2. Divide the image into a grid of "super-squares" and randomly assign a label to each super square 3. paint each real pixel with the true color plus all types of noise for this coordinate and project back into RGB or 8-Bit Greyscale space. The ground truth is generated in the same way with the same colors just without the noise added] [We have 3 types of noise 1. SuperSquareShift, where in we equally shift all pixels in this supersquare in a random color direction. 2. WhiteNoise, simply add a random color per pixel 3. OscillationNoise, this noise is base on the x,y coordinates, the S chosen for that experiment and a random phase shift chosen per image. osilatingNoise*(cos(x*/osilationWaveLength+rPhaseX)*cos(y*/osilationWaveLength+rPhaseY)+1)*255/2] [Finally we also add a restriction on which labels can be neighbours this is randomly chosen between possible pair of labels with the probability labled "dataGenNeighProb". Note that Label zero is considered the background and can be adjacent to any other label]. See Figures : (A.1, A.2 and A.3) for visual examples of these types of noise and Figure A.4 for the all together as a typical dataset. The 3 types of noise are needed to make the

TODO Pseudocode styling on the following

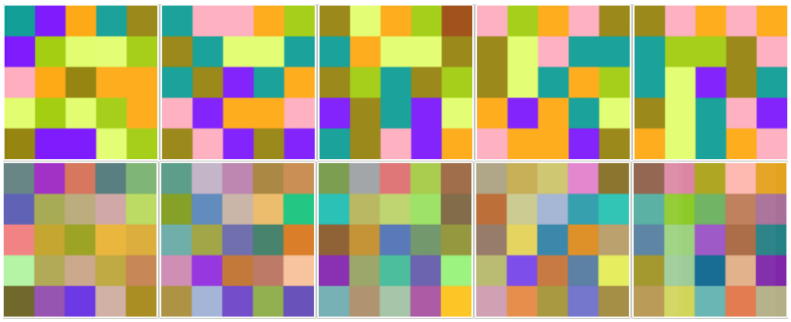


Figure A.1: An example of SuperSquareShift noise, the top row is the true ground truth and the bottom row is the image data used for features with SuperSquareShift=0.4 included.

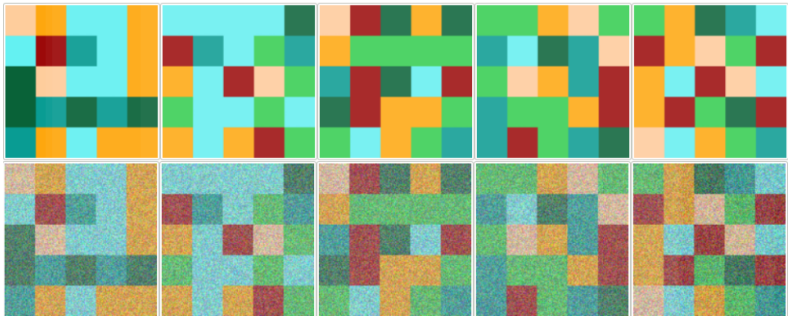


Figure A.2: An example of WhiteNoise, the top row displays the groundtruth and the bottom are the feature images with whtieNoise=0.4 added.

unary model based classifiers less powerfull 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 OscillationNoise 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 dataset which has ”dataGenNeighProb=0.3” in contrast to Figures A.1, A.2 and A.3 which have ”dataGenNeighProb=1.0”.

Figure A.3: And example of OsilationNoise, the top row displays the groundtruth and the bottom row are the feature images with osilationNoise=0.4 added.

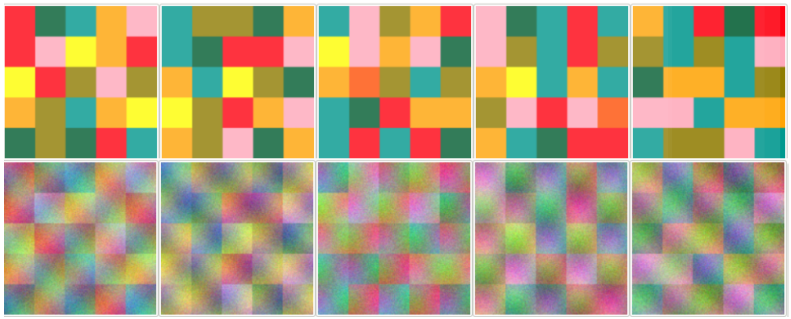


Figure A.4: An example of all types of noise. The groundtruth is in the top row an the feature images are in the bottom row with the following nois added: Super-Square-shift=0.05, WhiteNoise=0.35 and OsilationNoise=0.45.

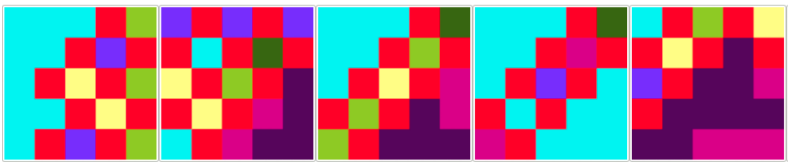


Figure A.5: An example dataset showing groudataGenNeighProb=0.3, it can be contrasted to A.4 where the ground truth was generated with groudataGenNeighProb=1.0. We can see that in the above image some colors never occur next to each other.

Bibliography

[1] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien 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] Manuel Jorge Cardoso, Matt Clarkson, and Sébastien Ourselin. Niftyseg. 2011.

[4] 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.

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

[6] 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.

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

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] Kristian Kvilekval, Dmitry Fedorov, Boguslaw Obara, Ambuj Singh, and B.S. Manjunath. Bisque: A platform for bioimage analysis and management. *Bioinformatics*, 26(4):544–552, Feb 2010.

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

[18] 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.

[19] Aurelien 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.

[20] 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.

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

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

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

[24] 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.

[25] 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.

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

[27] 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.

[28] 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.

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

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

[31] 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.

[32] 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.

[33] 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.

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

[35] 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.