Non-local means, FCCRFs and variational inference

D.A. Forsyth

Non-local means

- Smoothing
  - Estimate the value of a pixel using pixels that are nearby
    - eg gaussian filter, etc.
  - problem: some pixels might be on the other side of an edge
- Non-local means
  - Estimate the value of a pixel using pixels that are "similar"
    - eg write pixel value v; feature vector at pixel f; smoothed s

$$s_i = \sum_j w(\mathbf{f}_i, \mathbf{f}_j) v_j$$

Non-local means

- Non-local means
  - Estimate the value of a pixel using pixels that are “similar”
    - average all pixels, weighting by similarity
  - easy questions:
    - what is f? what is w?
  - harder:
    - how to get the sum quickly

$$\underset{\substack{\uparrow\\\text{smoothed}\\\text{estimate}}}{s_i} = \sum_j \overset{\substack{\text{Weight function}\\\downarrow}}{w}(\underbrace{\mathbf{f}_i, \mathbf{f}_j}_{\substack{\text{feature vector at}\\\text{pixel location}}})\overset{\substack{\text{Pixel value}\\\downarrow}}{v_j}$$

Natural choices

- In

$$s_i = \sum_j w(\mathbf{f}_i, \mathbf{f}_j) v_j$$

- f is
  - color, position, perhaps a texture feature
- w is

$$w(\mathbf{f}_i, \mathbf{f}_j) = \exp -\frac{1}{2}\left[(\mathbf{f}_i - \mathbf{f}_j)^T \mathcal{M} (\mathbf{f}_i - \mathbf{f}_j)\right]$$

- Notice this should simplify computing the sum
  - only “similar” pixels make reasonable contributions
  - but we must find them

Bilateral Filter

$$\mathbf{h}(\mathbf{x}) = k^{-1}(\mathbf{x}) \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \mathbf{f}(\xi) c(\xi, \mathbf{x}) s(\mathbf{f}(\xi), \mathbf{f}(\mathbf{x}))\, d\xi$$

Filtered version of an image

Image

Weights that depend on image value

Weights that depend on pixel position

$$k(\mathbf{x}) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} c(\xi, \mathbf{x}) s(\mathbf{f}(\xi), \mathbf{f}(\mathbf{x}))\, d\xi$$

Bilateral filter

Weight function

Pixel value

$$s_i = \frac{\sum_j w(\mathbf{f}_i, \mathbf{f}_j)\, g_{ij}\, v_j}{\sum_j w(\mathbf{f}_i, \mathbf{f}_j)\, g_{ij}}$$

smoothed
estimate

feature vector at
pixel location

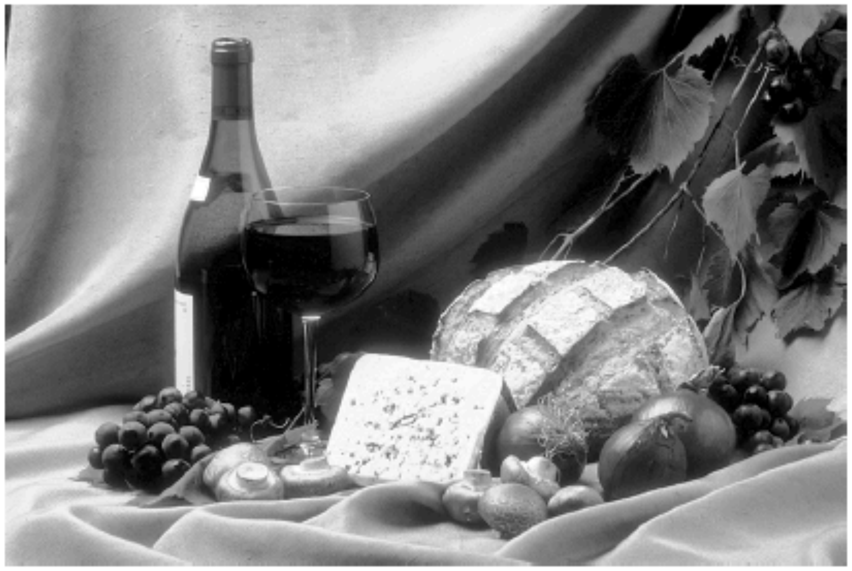
Weight depending
on pixel location

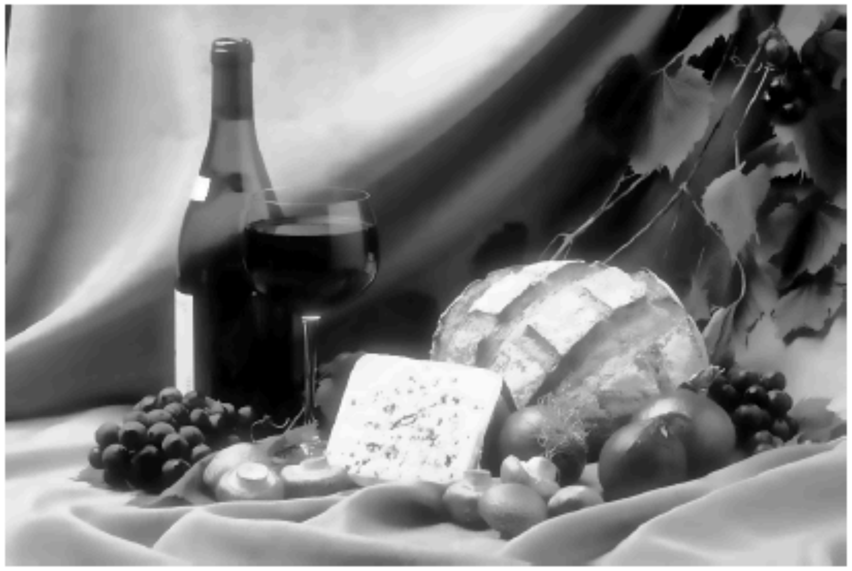
Bilateral filter

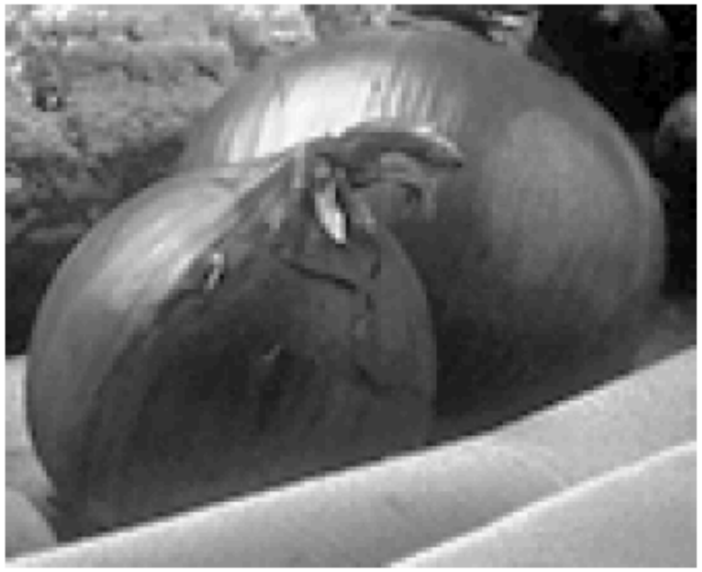
$$s_i = \frac{\sum_j w(\mathbf{f}_i, \mathbf{f}_j) g_{ij} v_j}{\sum_j w(\mathbf{f}_i, \mathbf{f}_j) g_{ij}}$$

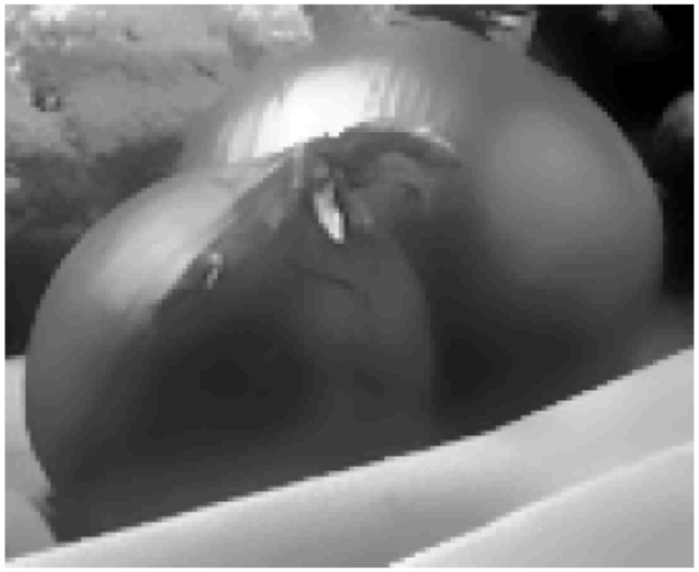
- Issue:
  - how to evaluate this
    - sum over all pixels? really?
- Notice:
  - we expect f's to cluster in some space
  - w falls off quite quickly for distance between f's
    - so clusters are what matters

Bilateral filter

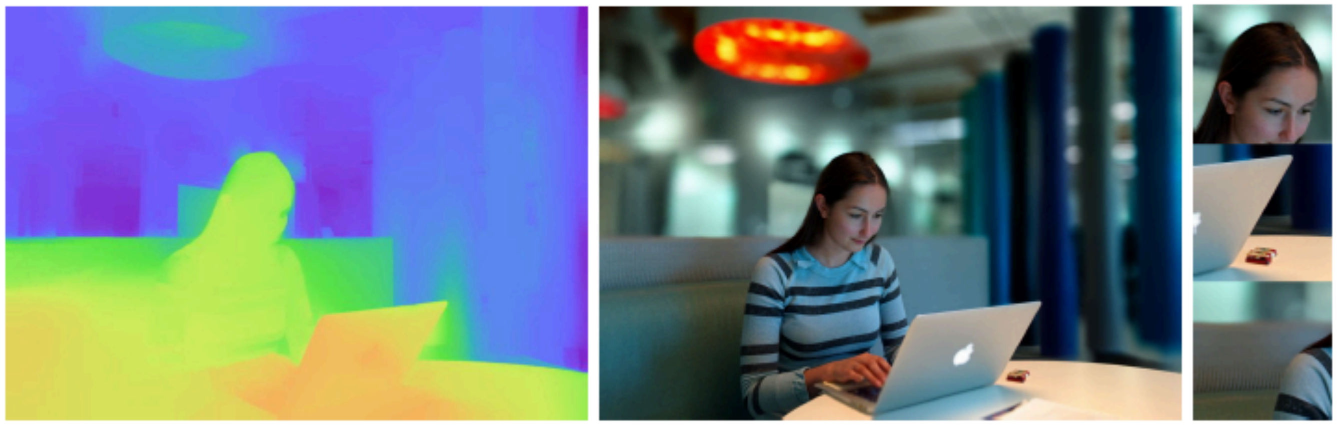

(a)

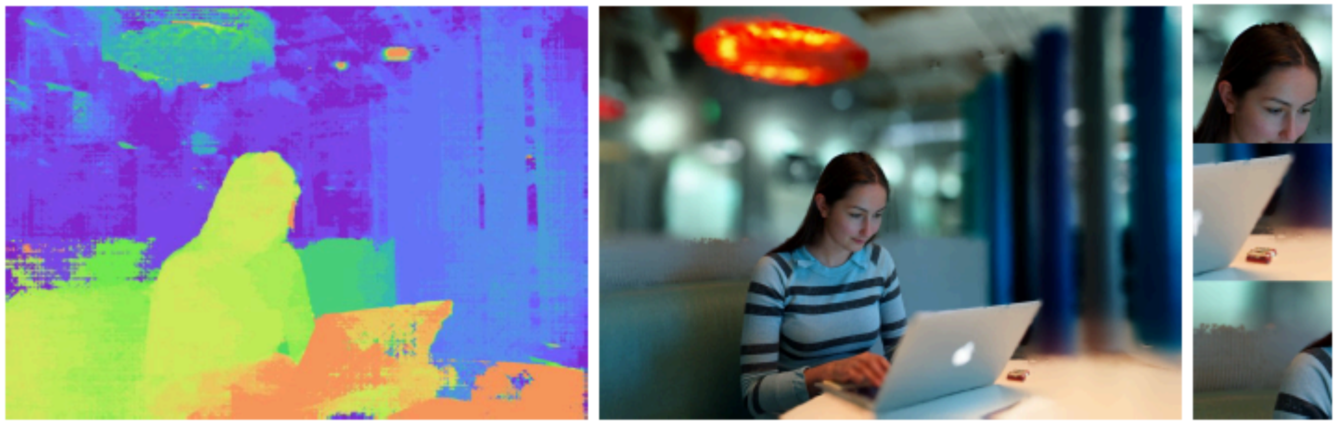

(b)


(c)


(d)Tomasi Manduci 98

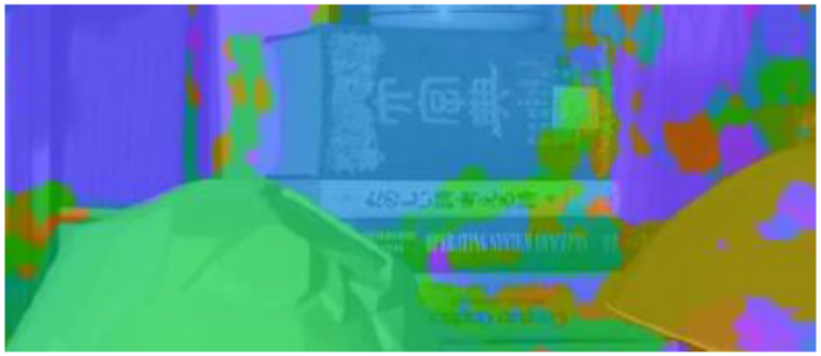
(a) Input stereo pair with cropped subregions of the right image

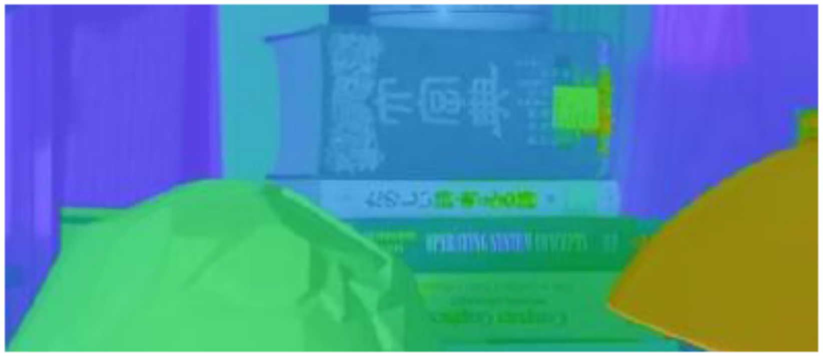

(b) Our algorithm's disparity and defocused image / subregions


(c) SGM's [14] disparity and defocused image / subregions

Barron et al 15

Depth maps


(a) Input (MAE = 6.00, RMSE = 38.8)

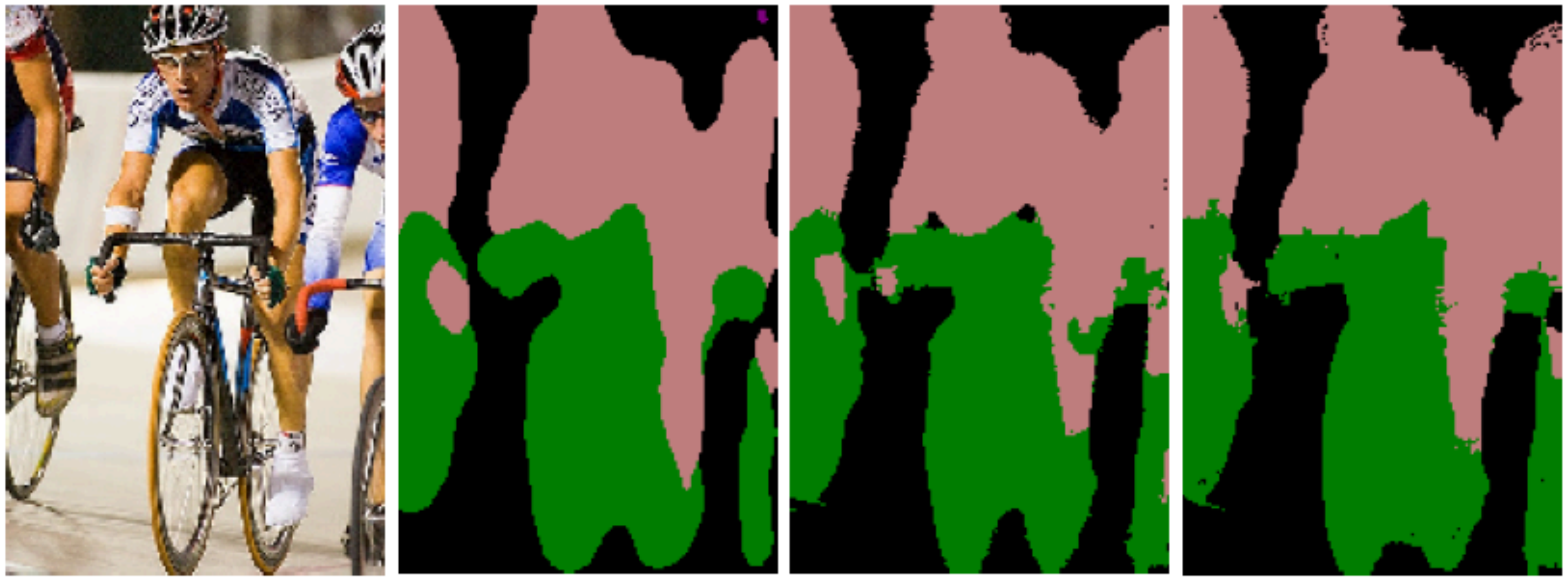

(b) Output (MAE = 3.02, RMSE = 17.9)

Barron et al 16

Apply to class probability maps

This is now a class probability at a pixel

$$s_i = \frac{\sum_j w(\mathbf{f}_i, \mathbf{f}_j) g_{ij} v_j}{\sum_j w(\mathbf{f}_i, \mathbf{f}_j) g_{ij}}$$



(a) Image (b) DeepLab (c) DenseCRF (d) BS (Ours)

Fig. 5: Using the DeepLab CNN-based semantic segmentation algorithm [6] (5b) as input our bilateral solver can produce comparable edge-aware output (5d) to the DenseCRF [22] used in [6] (5c), while being 8× faster.

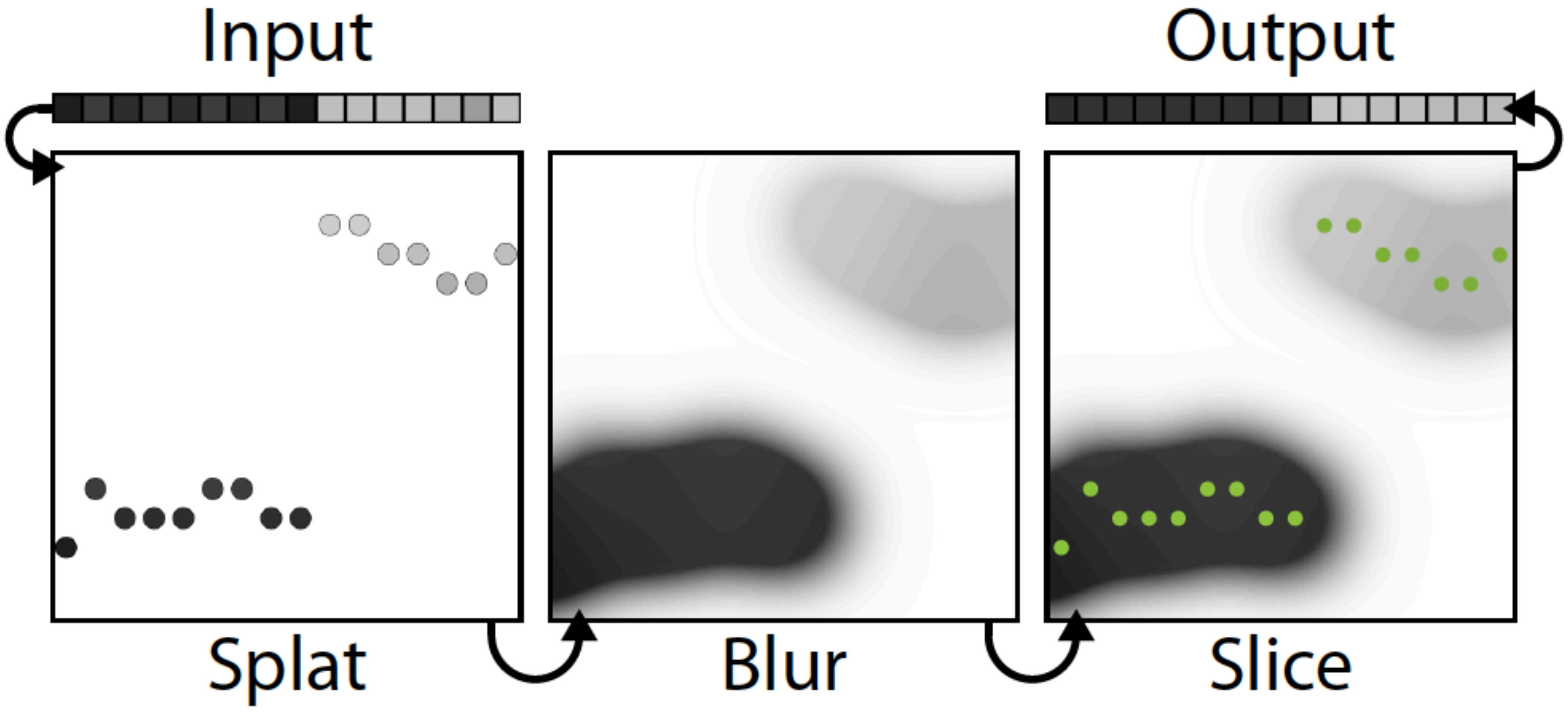
Barron et al FBS

Bilateral filter

$$s_i = \frac{\sum_j w(\mathbf{f}_i, \mathbf{f}_j) g_{ij} v_j}{\sum_j w(\mathbf{f}_i, \mathbf{f}_j) g_{ij}}$$

- Issue:
  - how to evaluate this
    - sum over all pixels? really?
- Notice:
  - we expect f's to cluster in some space
  - w falls off quite quickly for distance between f's
    - so clusters are what matters

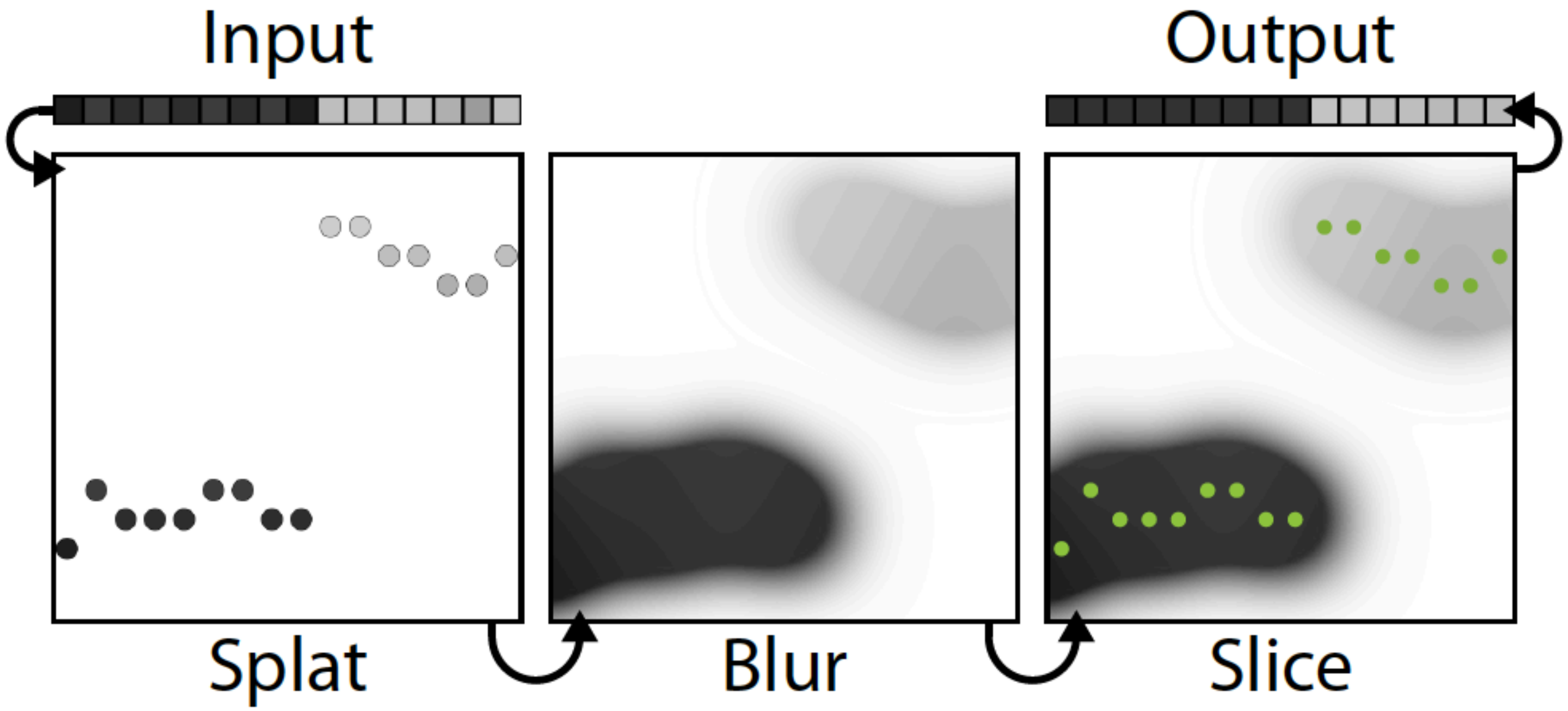
Splat, smooth, slice



from Adams, Baek, Davis

Need

- Some form of grid in high-D for f to splat onto
  - smoothing on this grid should be easy
  - it should be easy for pixels to find the closest point(s) on grid



Splat requires finding grid points enclosing splatted point

Blur requires finding neighboring grid points to a grid point

Slice requires interpolation from grid points

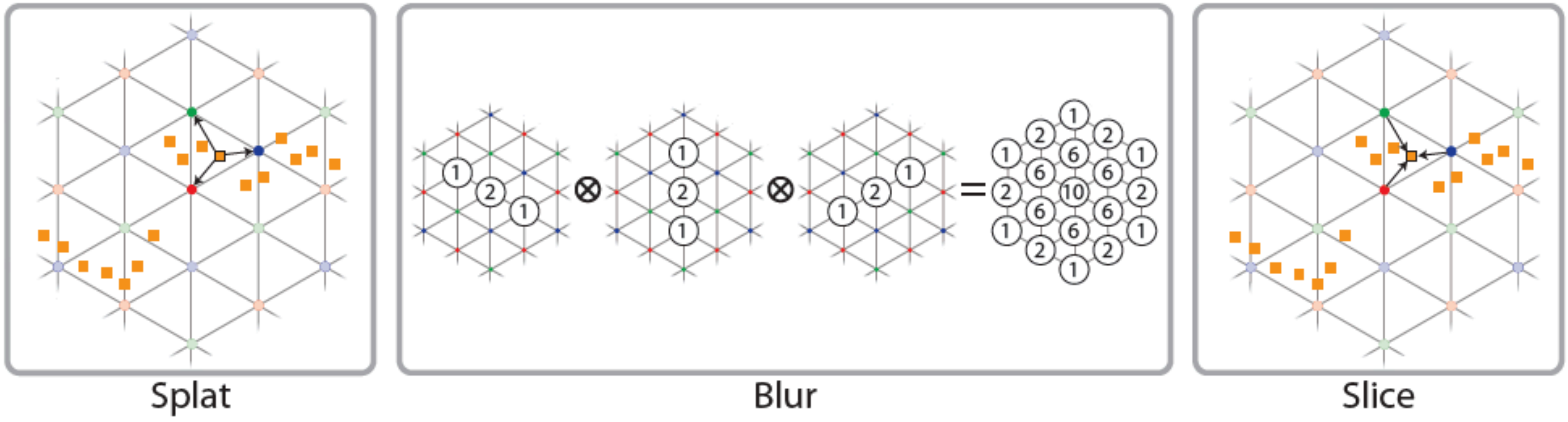


Figure 4: *To perform a high-dimensional Gaussian filter using the permutohedral lattice, first the position vectors $\vec{p}_i \in \mathbb{R}^d$ are embedded in the hyperplane H_d using an orthogonal basis for H_d (not pictured). Then, each input value* ***splats*** *onto the vertices of its enclosing simplex using barycentric weights. Next, lattice points* ***blur*** *their values with nearby lattice points using a separable filter. Finally, the space is* ***sliced*** *at each input position using the same barycentric weights to interpolate output values.*

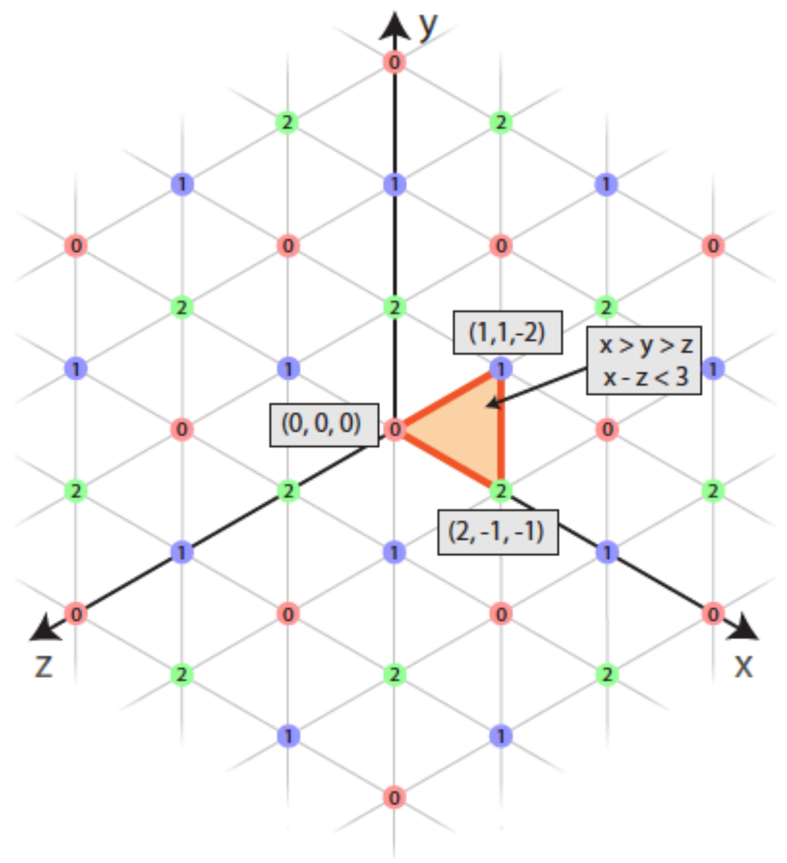


Figure 2: *The d-dimensional permutohedral lattice is formed by projecting the scaled grid* $(d+1)\mathbb{Z}^{d+1}$ *onto the plane* $\vec{x} \cdot \vec{1} = 0$. *This forms the lattice* $(d+1)A_d^*$, *which we term the permutohedral lattice, as it describes how to tile space with permutohedra. Lattice points have integer coordinates with a consistent remainder modulo* $d+1$. *In the diagram above, which illustrates the case* $d = 2$, *points are labeled and colored according to their remainder. The lattice tessellates the plane with uniform simplices, each simplex having one vertex of each remainder. The simplices are all translations and permutations of the canonical simplex (highlighted), which is defined by the inequalities* $x_0 > x_1 > \ldots > x_d$ *and* $x_0 - x_d < d+1$.

In "high" dimension

- Permutohedral lattice

The vertices of the simplex containing any point in H_d can be computed in $O(d^2)$ time. This property will be useful for the splat and slice stages of filtering.

The nearest neighbors of a lattice point can be computed in $O(d^2)$ time. This property will be useful during the blur stage of filtering.

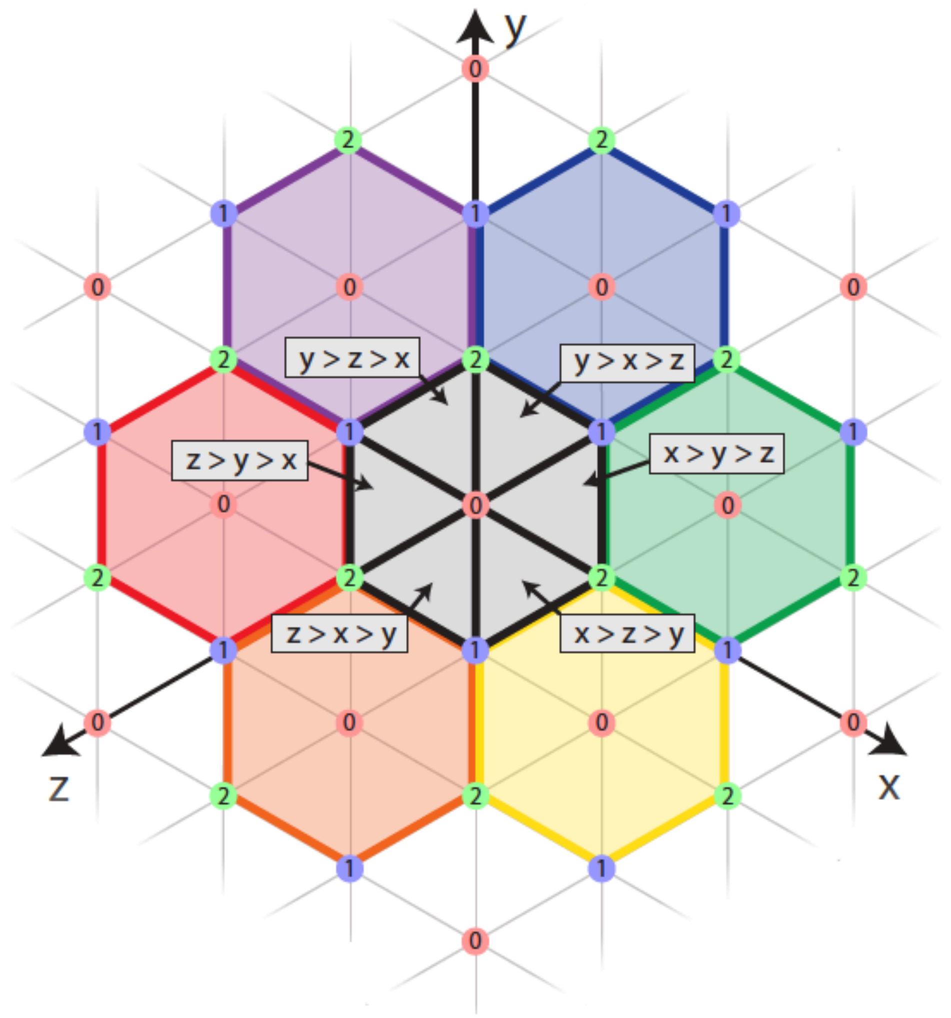
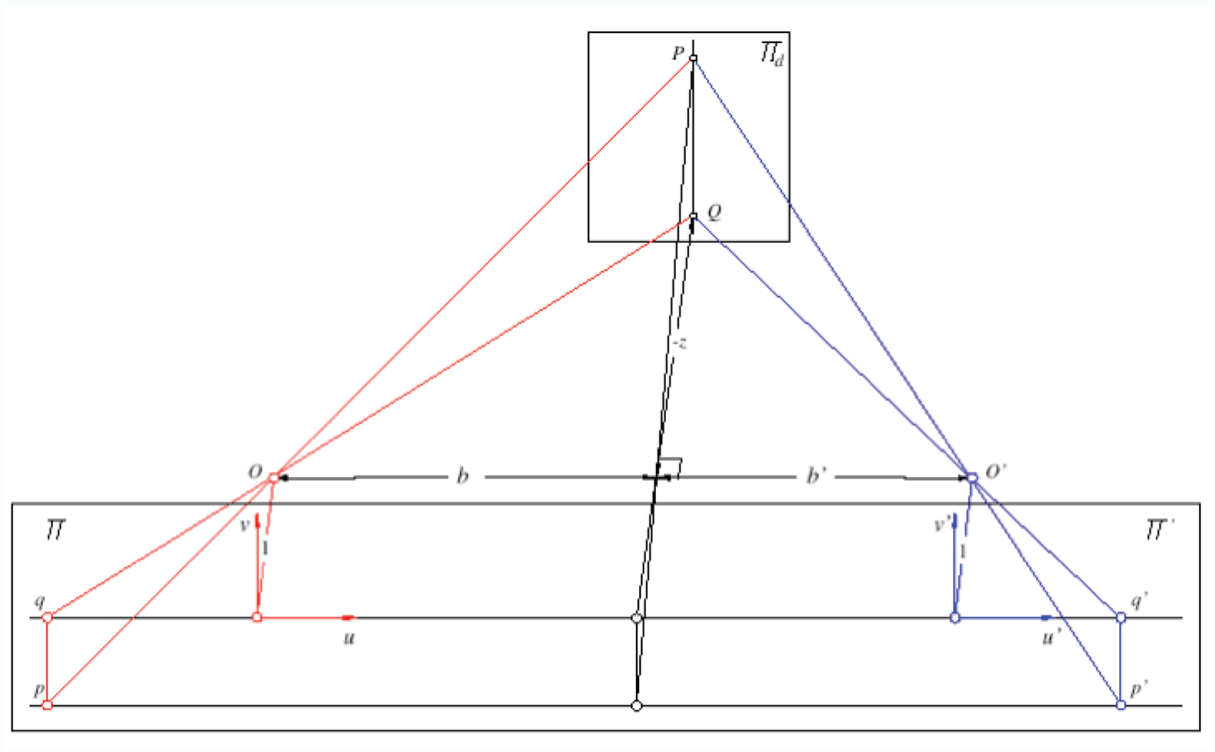


Figure 3: *When using the permutohedral lattice to tessellate the subspace H_d, any point $\vec{x} \in H_d$ is enclosed by a simplex uniquely identified by the nearest remainder-0 lattice point $\vec{l}_0$ (the zeroes highlighted in red) and the ordering of the coordinates of $\vec{x} - \vec{l}_0$. The nearest remainder-0 lattice point can be computed with a simple rounding algorithm, and so identifying the enclosing simplex of any point and enumerating its vertices is computationally cheap ($O(d^2)$).*

More general conditional random fields

- More than two labels
  - we’ve seen this case, very briefly, under stereo
- But now we have it for segmentation as well
  - one label per segment
  - costs:
    - per pixel:
      - how well does this label/pixel value go together
      - as in grabcut above
    - per pair:
      - how well does this label/pixel pair work together
      - usually, a form of smoothness
        - agree with your neighbors

Stereo as an optimization problem



- Original:
  - find q, q’ that match, and infer depth
- Now:
  - choose value of depth at q; then quality of match at q’ is cost
  - optimize this

Stereo as an optimization problem

- Typically:
  - quantize depth to a fixed number of levels
  - unary cost is color match
    - (photometric consistency constraint)
    - it can be helpful to match intensity gradients, too
  - pairwise cost from smoothness constraint on recovered depths
    - eg depth gradient not too big, etc.
  - massive discrete quadratic program

Discrete Quadratic Programs

- Minimize:
  - x^T A x + b^t x
  - subject to: x is a vector of discrete values
- Summary:
  - turn up rather often in early vision
    - from Markov random fields; conditional random fields; etc.
  - variety of cases:
    - some instances are polynomial
    - most are NP hard
      - but have extremely efficient, fast approximation algorithms
      - typically based on graph cuts, qv

More general conditional random fields

- x^t A x + b^t x
    - with x discrete, n labels
- Setting this up for segmentation
    - know a likelihood model for each label and pixel
        - cost(observation at pixel | label for that pixel)
    - easy way: x is a vector of one-hot vectors
        - one one-hot vector for each pixel
            - (eeew!) BIG
        - b is a vector
            - [cost(obs_1|l1=first), cost(obs_1|l1=second), ….]
    - A requires that nearby labels agree with one another
- Q: how to solve?

How to solve?

- Immense, very active literature
  - settled down a bit over the last 10 years, but…
- Key points:
  - Assume it is better to agree than disagree (this is in A)
    - Strong approximations available - they reduce to 0-1 case
      - a-expansion:
        - iterate over label values:
          - any label can either stay (0) or become a (1)
      - a-b swap:
        - iterate over pairs:
          - a, b pixels can stick (0) or swap (1)
    - Relatively fast, BUT iterative

Special case

- Every pixel is connected to every other pixel
    - with weights
- Yields a fast variational algorithm
    - based in non-local means

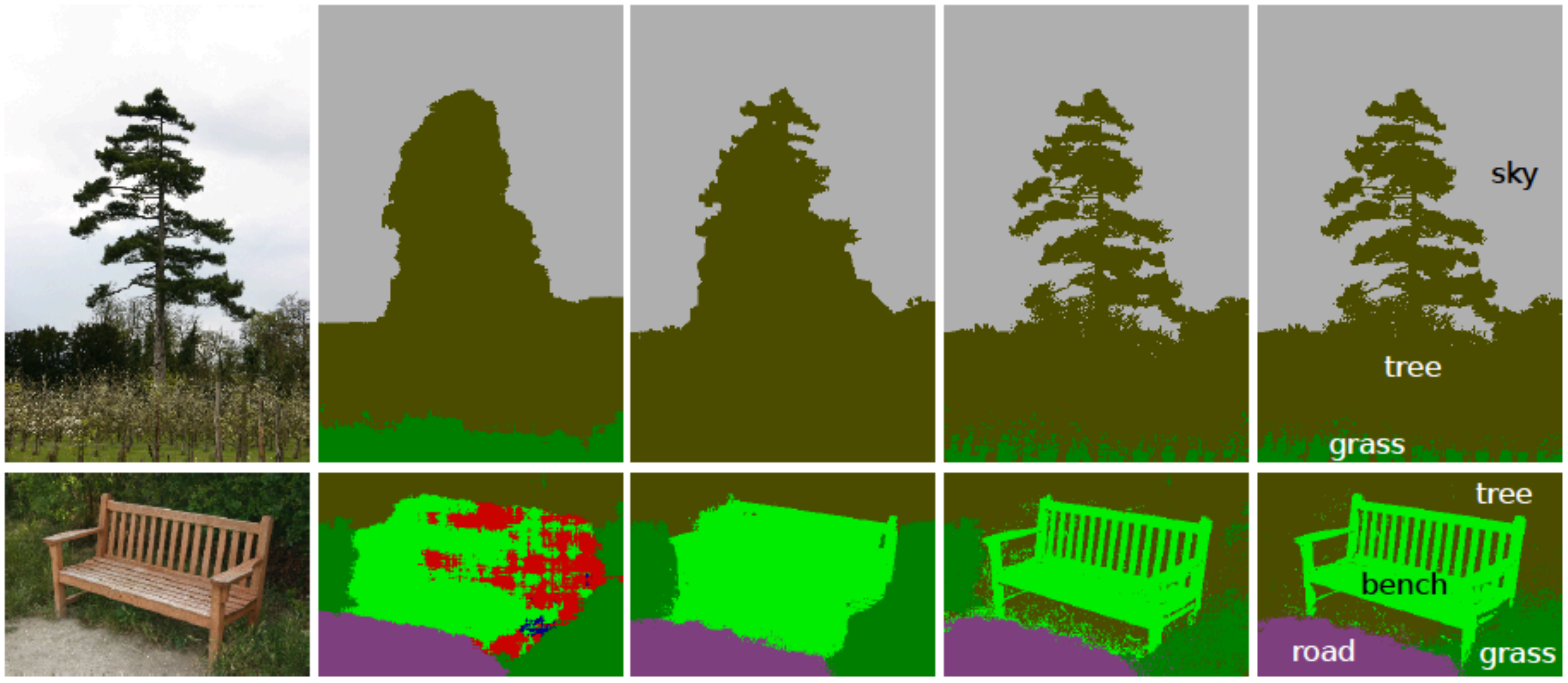
Fully connected CRF

In the fully connected pairwise CRF model, $\mathcal{G}$ is the complete graph on $\mathbf{X}$ and $\mathcal{C}_{\mathcal{G}}$ is the set of all unary and pairwise cliques. The corresponding Gibbs energy is

$$E(\mathbf{x}) = \sum_i \psi_u(x_i) + \sum_{i<j} \psi_p(x_i, x_j), \tag{1}$$

where i and j range from 1 to N. The unary potential $\psi_u(x_i)$ is computed independently for each pixel by a classifier that produces a distribution over the label assignment x_i given image features. The unary potential used in our implementation incorporates shape, texture, location, and color descriptors and is described in Section 5. Since the output of the unary classifier for each pixel is produced independently from the outputs of the classifiers for other pixels, the MAP labeling produced by the unary classifiers alone is generally noisy and inconsistent, as shown in Figure 1(b).

Why bother?



(a) Image (b) Unary classifiers (c) Robust P^n CRF (d) Fully connected CRF, MCMC inference, 36 hrs (e) Fully connected CRF, our approach, 0.2 seconds

Figure 1: Pixel-level classification with a fully connected CRF. (a) Input image from the MSRC-21 dataset. (b) The response of unary classifiers used by our models. (c) Classification produced by the Robust P^n CRF [9]. (d) Classification produced by MCMC inference [17] in a fully connected pixel-level CRF model; the algorithm was run for 36 hours and only partially converged for the bottom image. (e) Classification produced by our inference algorithm in the fully connected model in 0.2 seconds.

where i and j range from 1 to N. The unary potential $\psi_u(x_i)$ is computed independently for each pixel by a classifier that produces a distribution over the label assignment x_i given image features. The unary potential used in our implementation incorporates shape, texture, location, and color descriptors and is described in Section 5. Since the output of the unary classifier for each pixel is produced independently from the outputs of the classifiers for other pixels, the MAP labeling produced by the unary classifiers alone is generally noisy and inconsistent, as shown in Figure 1(b).

The pairwise potentials in our model have the form

$$\psi_p(x_i, x_j) = \mu(x_i, x_j) \underbrace{\sum_{m=1}^{K} w^{(m)} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j)}_{k(\mathbf{f}_i, \mathbf{f}_j)}, \qquad (2)$$

where each $k^{(m)}$ is a Gaussian kernel $k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) = \exp(-\frac{1}{2}(\mathbf{f}_i - \mathbf{f}_j)^{\mathrm{T}} \Lambda^{(m)} (\mathbf{f}_i - \mathbf{f}_j))$, the vectors $\mathbf{f}_i$ and $\mathbf{f}_j$ are feature vectors for pixels i and j in an arbitrary feature space, $w^{(m)}$ are linear combination weights, and μ is a label compatibility function. Each kernel $k^{(m)}$ is characterized by a symmetric, positive-definite precision matrix $\Lambda^{(m)}$, which defines its shape.

For multi-class image segmentation and labeling we use contrast-sensitive two-kernel potentials, defined in terms of the color vectors I_i and I_j and positions p_i and p_j:

$$k(\mathbf{f}_i, \mathbf{f}_j) = \underbrace{w^{(1)} \exp\left(-\frac{|p_i - p_j|^2}{2\theta_\alpha^2} - \frac{|I_i - I_j|^2}{2\theta_\beta^2}\right)}_{\text{appearance kernel}} + \underbrace{w^{(2)} \exp\left(-\frac{|p_i - p_j|^2}{2\theta_\gamma^2}\right)}_{\text{smoothness kernel}}. \qquad (3)$$

The *appearance kernel* is inspired by the observation that nearby pixels with similar color are likely to be in the same class. The degrees of nearness and similarity are controlled by parameters θ_α and θ_β. The *smoothness kernel* removes small isolated regions [19]. The parameters are learned from data, as described in Section 4.

An alternative strategy

- Variational inference
  - High level:
    - come up with simpler model that is “most like” intractable model
    - extract information from that
- Currently:
  - chose x_i (each 0 or 1) to maximize expression below

$$\log p(x|y) = \sum_{i=1}^{n} \lambda_i x_i + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \beta_{ij} \{x_i x_j + (1 - x_i)(1 - x_j)\}$$

$$+K$$

New setup

- H - hidden variables, 1 or -1
  - used to write x
- X - observations

$$\log p(H|X) = \sum_i \overset{\overset{\text{known}}{\downarrow}}{w_i} H_i + \sum_{ij} \overset{\overset{\text{known}}{\downarrow}}{\theta_{ij}} H_i H_j - \underset{\underset{\substack{\text{Unknowable,} \\ \text{but not depending} \\ \text{on H}}}{\uparrow}}{\log Z}$$

- We want to maximize this by choice of H
  - notice the -1, 1 trick

15.2 VARIATIONAL INFERENCE

We could just ignore intractable models, and stick to tractable models. This isn't a good idea, because intractable models are often quite natural. The discrete Markov random field model of an image is a fairly natural model. Image labels *should* depend on pixel values, and on neighboring labels. It is better to try and deal with the intractable model. One really successful strategy for doing so is to choose a tractable parametric family of probability models $Q(H;\theta)$, then adjust θ to find parameter values $\hat{\theta}$ that represent a distribution that is "close" in the right sense to $P(H|X)$. One then extracts information from $Q(H;\hat{\theta})$. This process is known as **variational inference**. What is remarkable is that (a) it is possible to find a $Q(H;\hat{\theta})$ without too much fuss and (b) information extracted from this distribution is often accurate and useful.

15.2.1 The KL Divergence

Assume we have two probability distributions $P(X)$ and $Q(X)$. A measure of their similarity is the **KL-divergence** (or sometimes **Kullback-Leibler divergence**) written

$$\mathbb{D}(P \,\|\, Q) = \int P(X) \log \frac{P(X)}{Q(X)} dX$$

(you've clearly got to be careful about zeros in P and Q here). This likely strikes you as an odd measure of similarity, because it isn't symmetric. It is not the case that $\mathbb{D}(P \,\|\, Q)$ is the same as $\mathbb{D}(Q \,\|\, P)$, which means you have to watch your P's and Q's. Furthermore, some work will demonstrate that it does not satisfy the triangle inequality, so KL divergence lacks two of the three important properties of a metric.

KL divergence has some nice properties, however. First, we have

$$\mathbb{D}(P \,\|\, Q) \geq 0$$

with equality only if P and Q are equal almost everywhere (i.e. except on a set of measure zero).

Remember this: *The KL divergence measures the similarity of two probability distributions. It is always non-negative, and is only zero if the two distributions are the same. However, it is not symmetric.*

KL divergence and Maximum likelihood

Second, there is a suggestive relationship between KL divergence and maximum likelihood. Assume that X_i are IID samples from some *unknown* $P(X)$, and we wish to fit a parametric model $Q(X|\theta)$ to these samples. This is the usual situation we deal with when we fit a model. Now write $H(P)$ for the entropy of $P(X)$, defined by

$$H(P) = -\int P(X) \log P(X) dx = -\mathbb{E}_P[\log P].$$

The distribution P is unknown, and so is its entropy, but it is a constant. Now we can write

$$\mathbb{D}(P \| Q) = \mathbb{E}_P[\log P] - \mathbb{E}_P[\log Q]$$

KL divergence and Maximum Likelihood

Then

$$\begin{aligned}\mathcal{L}(\theta) = \sum_i \log Q(X_i|\theta) \approx \int P(X) \log Q(X|\theta) dX &= \mathbb{E}_{P(X)}[\log Q(X|\theta)] \\ &= -H(P) - \mathbb{D}(P \| Q)(\theta).\end{aligned}$$

Equivalently, we can write

$$\mathcal{L}(\theta) + \mathbb{D}(P \| Q)(\theta) = -H(P).$$

Recall P doesn't change (though it's unknown), so $H(P)$ is also constant (though unknown). This means that when $\mathcal{L}(\theta)$ goes up, $\mathbb{D}(P \| Q)(\theta)$ must go down. When $\mathcal{L}(\theta)$ is at a maximum, $\mathbb{D}(P \| Q)(\theta)$ must be at a minimum. All this means that, when you choose θ to maximize the likelihood of some dataset given θ for a parametric family of models, you are choosing the model in that family with smallest KL divergence from the (unknown) $P(X)$.

15.2.2 The Variational Free Energy

We have a $P(H|X)$ that is hard to work with (usually because we can't evaluate $P(X)$) and we want to obtain a $Q(H)$ that is "close to" $P(H|X)$. A good choice of "close to" is to require that

$$\mathbb{D}(Q(H) \,\|\, P(H|X))$$

is small. Expand the expression for KL divergence, to get

$$\begin{aligned}\mathbb{D}(Q(H) \,\|\, P(H|X)) &= \mathbb{E}_Q[\log Q] - \mathbb{E}_Q[\log P(H|X)] \\ &= \mathbb{E}_Q[\log Q] - \mathbb{E}_Q[\log P(H,X)] + \mathbb{E}_Q[\log P(X)] \\ &= \mathbb{E}_Q[\log Q] - \mathbb{E}_Q[\log P(H,X)] + \log P(X)\end{aligned}$$

which at first glance may look unpromising, because we can't evaluate $P(X)$. But $\log P(X)$ is fixed (although unknown). Now rearrange to get

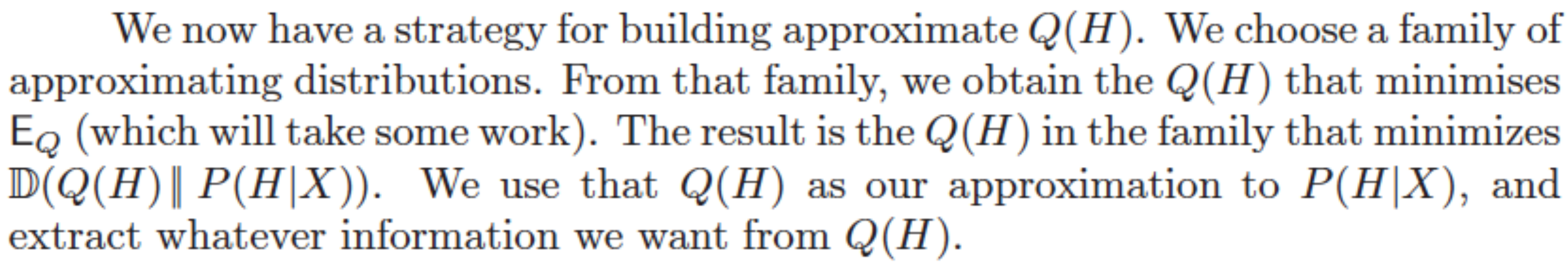
$$\begin{aligned}\log P(X) &= \mathbb{D}(Q(H) \,\|\, P(H|X)) - (\mathbb{E}_Q[\log Q] - \mathbb{E}_Q[\log P(H,X)]) \\ &= \mathbb{D}(Q(H) \,\|\, P(H|X)) - \mathsf{E}_Q.\end{aligned}$$

Here

$$\mathsf{E}_Q = (\mathbb{E}_Q[\log Q] - \mathbb{E}_Q[\log P(H,X)])$$

is referred to as the **variational free energy**. We can't evaluate $\mathbb{D}(Q(H) \,\|\, P(H|X))$. But, because $\log P(X)$ is fixed, when E_Q goes down, $\mathbb{D}(Q(H) \,\|\, P(H|X))$ must go down too. Furthermore, a minimum of E_Q will correspond to a minimum of $\mathbb{D}(Q(H) \,\|\, P(H|X))$. And we can evaluate E_Q.

Variational Inference



We now have a strategy for building approximate $Q(H)$. We choose a family of approximating distributions. From that family, we obtain the $Q(H)$ that minimises E_Q (which will take some work). The result is the $Q(H)$ in the family that minimizes $\mathbb{D}(Q(H) \| P(H|X))$. We use that $Q(H)$ as our approximation to $P(H|X)$, and extract whatever information we want from $Q(H)$.

- Questions:
  - what Q(H)?
    - hard, case by case basis; essentially, so that calculations go through
  - How to minimize?
    - straightforward (long, dull) calculation
      - (AML Ch15 for easiest example)

Variational Inference

We want to construct a $Q(H)$ that approximates the posterior for a Boltzmann machine. We will choose $Q(H)$ to have one factor for each hidden variable, so $Q(H) = q_1(H_1)q_2(H_2)\ldots q_N(H_N)$. We will then assume that all but one of the terms in Q are known, and adjust the remaining term. We will sweep through the terms doing this until nothing changes.

The i'th factor in Q is a probability distribution over the two possible values of H_i, which are 1 and -1. There is only one possible choice of distribution. Each q_i has one parameter $\pi_i = P(\{H_i = 1\})$. We have

$$q_i(H_i) = (\pi_i)^{\frac{(1+H_i)}{2}} (1-\pi_i)^{\frac{(1-H_i)}{2}} .$$

Notice the trick; the power each term is raised to is either 1 or 0, and I have used this trick as a switch to turn on or off each term, depending on whether H_i is 1 or -1. So $q_i(1) = \pi_i$ and $q_i(-1) = (1-\pi_i)$. This is a standard, and quite useful, trick. We wish to minimize the variational free energy, which is

$$\mathsf{E}_Q = (\mathbb{E}_Q[\log Q] - \mathbb{E}_Q[\log P(H, X)]).$$

Variational Inference for FCCRFs

Minimizing the KL-divergence, while constraining $Q(\mathbf{X})$ and $Q_i(X_i)$ to be valid distributions, yields the following iterative update equation:

$$Q_i(x_i = l) = \frac{1}{Z_i} \exp\left\{ -\psi_u(x_i) - \sum_{l' \in \mathcal{L}} \mu(l, l') \sum_{m=1}^{K} w^{(m)} \sum_{j \neq i} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) Q_j(l') \right\}. \tag{4}$$

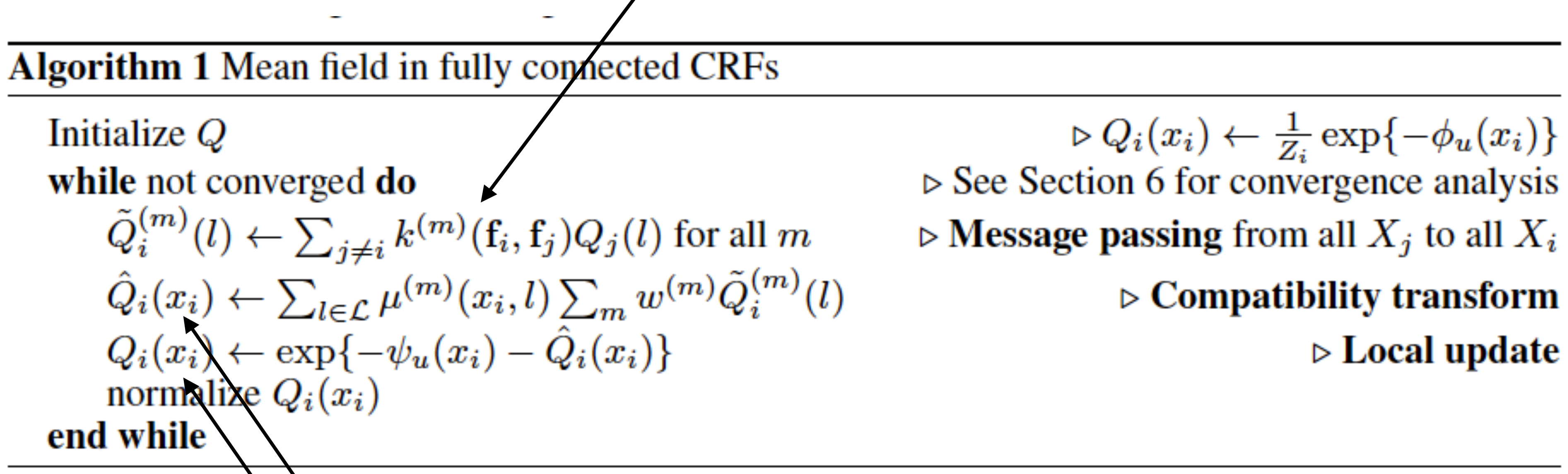
A detailed derivation of Equation 4 is given in the supplementary material. This update equation leads to the following inference algorithm:

$$Q_i(x_i = l) = \frac{1}{Z_i}\exp\left\{-\psi_u(x_i) - \sum_{l'\in\mathcal{L}}\mu(l,l')\sum_{m=1}^{K} w^{(m)} \underline{\sum_{j\neq i} k^{(m)}(\mathbf{f}_i,\mathbf{f}_j)Q_j}(l')\right\}$$

This is non-local means
or bilateral filter

Alg.

non-local means (splat blur slice)



Algorithm 1 Mean field in fully connected CRFs

```
Initialize $Q$                                                        ▷ $Q_i(x_i) \leftarrow \frac{1}{Z_i}\exp\{-\phi_u(x_i)\}$
while not converged do                                                ▷ See Section 6 for convergence analysis
    $\tilde{Q}_i^{(m)}(l) \leftarrow \sum_{j \neq i} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) Q_j(l)$ for all $m$          ▷ Message passing from all $X_j$ to all $X_i$
    $\hat{Q}_i(x_i) \leftarrow \sum_{l \in \mathcal{L}} \mu^{(m)}(x_i, l) \sum_m w^{(m)} \tilde{Q}_i^{(m)}(l)$    ▷ Compatibility transform
    $Q_i(x_i) \leftarrow \exp\{-\psi_u(x_i) - \hat{Q}_i(x_i)\}$                        ▷ Local update
    normalize $Q_i(x_i)$
end while
```

This is a vector of values, one per label l', hence the notation issue

This is a vector of values, one per label l', hence the notation issue

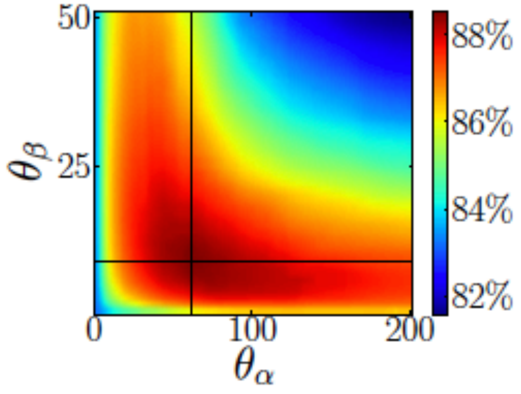
Long range connections seem to help

$$k(\mathbf{f}_i,\mathbf{f}_j) = w^{(1)} \underbrace{\exp\left(-\frac{|p_i-p_j|^2}{2\theta_\alpha^2} - \frac{|I_i-I_j|^2}{2\theta_\beta^2}\right)}_{\text{appearance kernel}} + w^{(2)} \underbrace{\exp\left(-\frac{|p_i-p_j|^2}{2\theta_\gamma^2}\right)}_{\text{smoothness kernel}}.$$

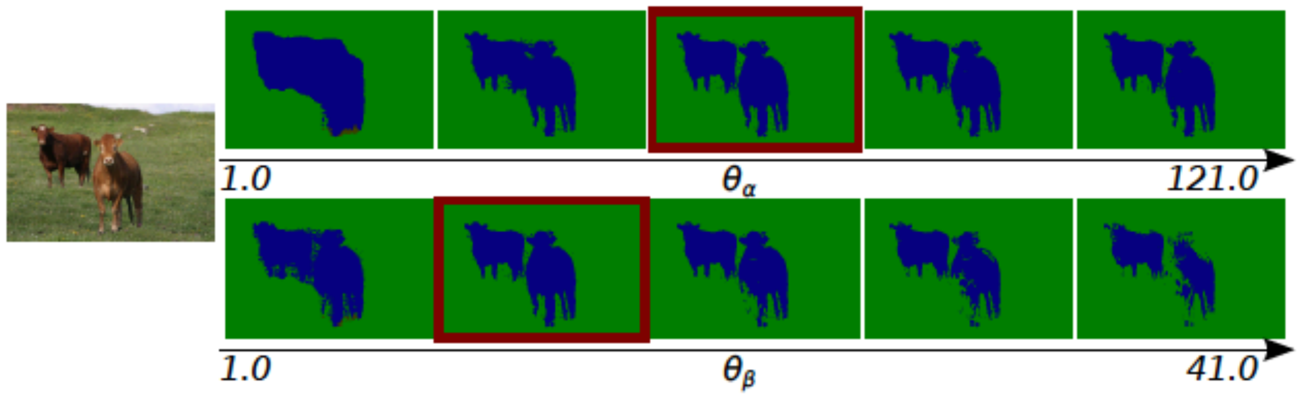
Manipulate these

Long range connections seem to help

Long-range connections. We have examined the value of long-range connections in our model by varying the spatial and color ranges θ_α and θ_β of the appearance kernel and analyzing the resulting classification accuracy. For this experiment, $w^{(1)}$ was held constant and $w^{(2)}$ was set to 0. The results are shown in Figure 6. Accuracy steadily increases as longer-range connections are added, peaking at spatial standard deviation of $\theta_\alpha = 61$ pixels and color standard deviation $\theta_\beta = 11$. At this setting, more than 50% of the pairwise potential energy in the model was assigned to edges of length 35 pixels or higher. However, long-range connections can also propagate misleading information, as shown in Figure 7.

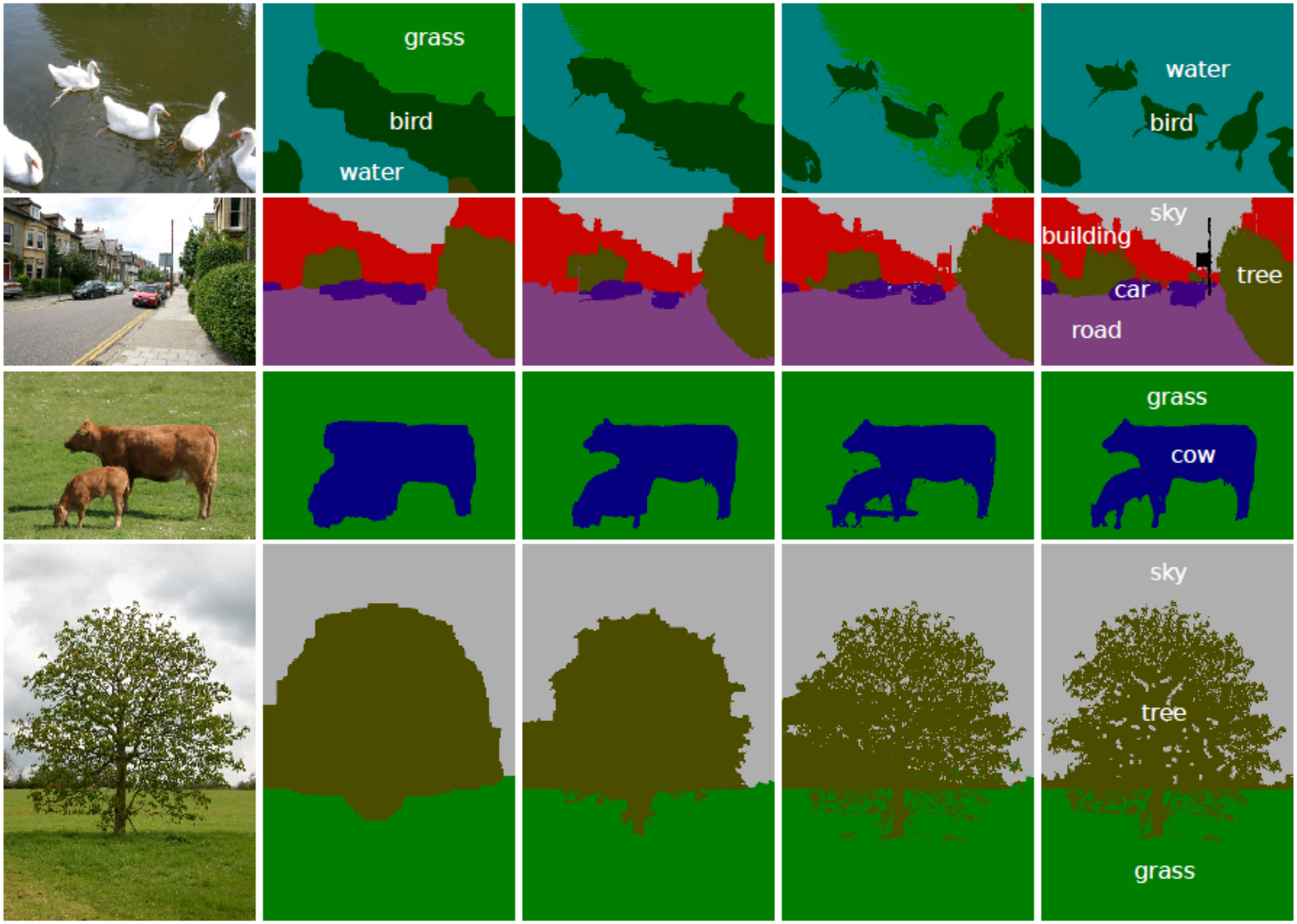


(a) Quantitative



(b) Qualitative

Figure 6: Influence of long-range connections on classification accuracy. (a) Global classification accuracy on the 94 MSRC images with accurate ground truth, as a function of kernel parameters θ_α and θ_β. (b) Results for one image across two slices in parameter space, shown as black lines in (a).

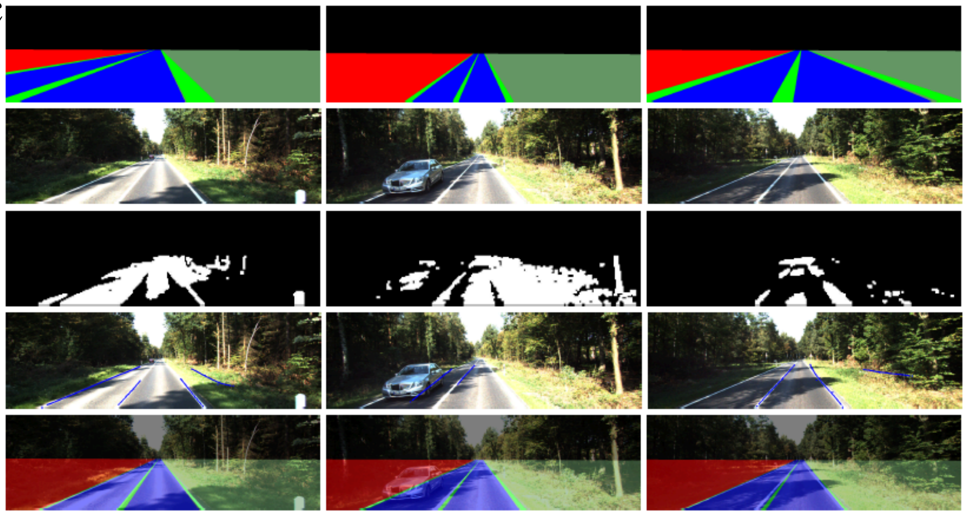

grass
bird
water
water
bird
sky
building
tree
car
road
grass
cow
sky
tree
grass
Image
Grid CRF
Robust P^n CRF
Our approach
Accurate ground truth

General summary

- Complicated but fast and efficient method
  - imposes spatial priors
  - results are pre deep learning
    - no end-to-end training
- For a while, widely used on semantic segmenters
    - train segmenter end-to-end
    - then bolt this on to smooth labels
  - now somewhat less common
    - why? not sure
- Weight training method exists
  - essentially, search
- Good evidence that applying fast bilateral solver
  - is faster, only slightly worse

General summary, II

- This will impose coherence constraints
  - things with the same label should look similar
- And spatial constraints
  - neighbors should mostly agree
- But it won't
  - make straight lines
  - make stylized layouts



Mansinghka et al 13