Fully connected CRF’s

D.A. Forsyth
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
    • $[\text{cost}(\text{obs}_1 l=\text{first}), \text{cost}(\text{obs}_1 l=\text{second}), \ldots]$
  • $A$ 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
In the fully connected pairwise CRF model, \( G \) is the complete graph on \( X \) and \( C_G \) is the set of all unary and pairwise cliques. The corresponding Gibbs energy is

\[
E(x) = \sum_i \psi_u(x_i) + \sum_{i<j} \psi_p(x_i, x_j),
\]

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?

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

\[
\psi_p(x_i, x_j) = \mu(x_i, x_j) \sum_{m=1}^{K} w^{(m)} k^{(m)}(f_i, f_j),
\]

where each \( k^{(m)} \) is a Gaussian kernel \( k^{(m)}(f_i, f_j) = \exp(-\frac{1}{2}(f_i - f_j)^T \Lambda^{(m)}(f_i - f_j)) \), the vectors \( f_i \) and \( f_j \) are feature vectors for pixels \( i \) and \( j \) in an arbitrary feature space, \( w^{(m)} \) are linear combination weights, and \( \mu \) 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 \( p_j \):

\[
k(f_i, f_j) = w^{(1)} \exp \left( -\frac{|p_i - p_j|^2}{2\theta_\alpha^2} - \frac{|I_i - I_j|^2}{2\theta_\beta^2} \right) + w^{(2)} \exp \left( -\frac{|p_i - p_j|^2}{2\theta_\gamma^2} \right).
\]

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 \( \theta_\alpha \) and \( \theta_\beta \). The smoothness kernel removes small isolated regions [19]. The parameters are learned from data, as described in Section 4.
Variational Inference for FCCRFs

Minimizing the KL-divergence, while constraining \( Q(X) \) 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:
This is non-local means or bilateral filter.
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 $s$

$$s_i = \sum_j w(f_i, f_j) v_j$$
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

\[
S_i = \sum_j w(f_i, f_j) v_j
\]

- Weight function
- Pixel value
- Smoothed estimate
- Feature vector at pixel location
Natural choices

- In
  \[ s_i = \sum_j w(f_i, f_j) v_j \]

- \( f \) is
  - color, position, perhaps a texture feature

- \( w \) is
  \[ w(f_i, f_j) = \exp - \frac{1}{2} \left[ (f_i - f_j)^T M (f_i - f_j) \right] \]

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

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

Filtered version of an image

Image

Weights that depend on pixel position

Weights that depend on image value

\[ k(x) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} c(\xi, x) s(f(\xi), f(x)) \, d\xi \]
Bilateral filter

\[ S_i = \frac{\sum_j w(f_i, f_j) g_{ij} v_j}{\sum_j w(f_i, f_j) g_{ij}} \]

- Weight function
- Pixel value
- Smoothed estimate
- Feature vector at pixel location
- Weight depending on pixel location
Bilateral filter

\[ s_i = \frac{\sum_j w(f_i, f_j) g_{ij} v_j}{\sum_j w(f_i, f_j) g_{ij}} \]

- **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
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

\[ s_i = \sum_j w(f_i, f_j) g_{ij} v_j \]

This is now a class probability at a pixel
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.
Bilateral filter

\[ s_i = \frac{\sum_j w(f_i, f_j) g_{ij} v_j}{\sum_j w(f_i, f_j) g_{ij}} \]

- **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
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.
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 $\bar{x} \in H_d$ is enclosed by a simplex uniquely identified by the nearest remainder-0 lattice point $\bar{l}_0$ (the zeroes highlighted in red) and the ordering of the coordinates of $\bar{x} - \bar{l}_0$. 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 ($O(d^2)$).
**Algorithm 1** Mean field in fully connected CRFs

- Initialize $Q$
- **while** not converged do
  - $\tilde{Q}_{i}^{(m)}(l) \leftarrow \sum_{j \neq i} k^{(m)}(f_i, f_j)Q_{j}(l)$ for all $m$
  - $\hat{Q}_{i}(x_i) \leftarrow \sum_{l \in \mathcal{L}} \mu^{(m)}(x_i, l) \sum_{m} w^{(m)} \tilde{Q}_{i}^{(m)}(l)$
  - $Q_i(x_i) \leftarrow \exp\{-\psi_u(x_i) - \hat{Q}_i(x_i)\}$
  - **normalize** $Q_i(x_i)$
- **end while**

- $Q_i(x_i) \leftarrow \frac{1}{Z_i} \exp\{-\phi_u(x_i)\}$
- **See Section 6 for convergence analysis**
- **Message passing** from all $X_j$ to all $X_i$
  - **Compatibility transform**
    - **Local update**

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

non-local means (splat blur slice)
Long range connections seem to help manipulate these

\[ k(f_i, f_j) = w^{(1)} \exp \left( -\frac{|p_i - p_j|^2}{2\theta^2_\alpha} - \frac{|I_i - I_j|^2}{2\theta^2_\beta} \right) + w^{(2)} \exp \left( -\frac{|p_i - p_j|^2}{2\theta^2_\gamma} \right). \]

appearance kernel  

smoothness kernel
Long-range connections. We have examined the value of long-range connections in our model by varying the spatial and color ranges $\theta_\alpha$ and $\theta_\beta$ 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.

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 $\theta_\alpha$ and $\theta_\beta$. (b) Results for one image across two slices in parameter space, shown as black lines in (a).
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