Appendix — Structured Output Learning with High Order Loss Functions

We include the following:

- Detailed derivation of PASCAL factor messages.
- Detailed derivation of not-monotonic factor messages negative example LBC messages.
- Detailed derivation of modified convexity factor messages for positive example LBC messages.
- Detailed description of the synthetic data experiment.
- Additional results showing test outputs on different classes of object.
- Illustration of typical failure modes of PASCAL-trained model.

7 PASCAL Loss Factor

Our goal is to compute outgoing messages from a factor representing the PASCAL loss,

$$\Delta_{y^*}^{\text{PASCAL}}(y) = \frac{\sum_i y_i^*(1-y_i) - \sum_i y_i^*}{\sum_i y_i^* + \sum_i y_i(1-y_i^*)}. \tag{9}$$

As in the main text, let $N^+ = \sum_i y_i^*$, $N_0 = \sum_{i:y_i^*=1}(1-y_i)$, and $N_1 = \sum_{i:y_i^*=0} y_i$ be the number of ground truth pixels, false negatives, and false positives, respectively. We can rewrite the loss as $\Delta_{y^*}^{\text{PASCAL}}(y) = \frac{N_0 - N^+}{N^+ + N_1}$.

Consider computing a single message,

$$m_{\Delta\to i}(y_i) = \max_{y_{-i}} \left[\Delta(y_i, y_{-i}) + \sum_{i'\neq i} m_{i'\to\Delta}(y_{i'}) \right] \tag{10}$$

$$= \max_{y_{-i}} \left[\frac{N_0 - N^+}{N^+ + N_1} + \sum_{i'\neq i} m_{i'\to\Delta}(y_{i'}) \right]. \tag{11}$$

We can transform the sum of incoming messages to similarly be functions of N_0 and N_1. First, split the incoming messages into those coming from pixels that are labeled 0 in the ground truth and those that are labeled 1 in the ground truth:

$$\sum_{i'\neq i} m_{i'\to\Delta}(y_{i'}) = \sum_{i'\neq i: y_{i'}^*=0} m_{i'\to\Delta}(y_{i'}) + \sum_{i'\neq i: y_{i'}^*=1} m_{i'\to\Delta}(y_{i'}) \tag{12}$$

$$= s_1^{-i}(N_1) + s_0^{-i}(N_0) + \kappa, \tag{13}$$

where $s_0^{-i}(N_0)$ is constructed by sorting the incoming message differences

$$m_{i'\to\Delta}(0) - m_{i'\to\Delta}(1) \tag{14}$$

for $\{i' | i' \neq i, y_{i'}^* = 0\}$ in descending order, then taking a cumulative sum of the sorted values, and $s_1^{-i}(N_1)$ is constructed by sorting the incoming message differences

$$m_{i'\to\Delta}(1) - m_{i'\to\Delta}(0) \tag{15}$$

for $\{i' | i' \neq i, y_{i'}^* = 1\}$ in descending order, then taking a cumulative sum of the sorted values. By taking message differences, we have added a constant κ, but we have not changed the optimum location or relative values of assignments.

Finally, we must account for the contribution of y_i to N_0 and N_1. There are three possibilities:

- When computing messages for $m_{\Delta\to i}(0)$ and $y_i^* = 1$, optimize

$$f(N_0, N_1) = \frac{N_0 - N^+ + 1}{N^+ + N_1} + s_0^{-i}(N_0) + s_1^{-i}(N_1). \tag{16}$$

- When computing messages for $m_{\Delta\to i}(1)$ and $y_i^* = 0$, optimize

$$f(N_0, N_1) = \frac{N_0 - N^+}{N^+ + N_1 + 1} + s_0^{-i}(N_0) + s_1^{-i}(N_1). \tag{17}$$

- Otherwise, optimize

$$f(N_0, N_1) = \frac{N_0 - N^+}{N^+ + N_1} + s_0^{-i}(N_0) + s_1^{-i}(N_1). \tag{18}$$

Finally, note that the all s_1^{-i} function values needed can be computed using a single sort of all incoming messages from variables where $y_i^* = 0$. That is, a separate sort is not needed for each i. Let the cumulative sum of a sorted array of message differences be s_1. When evaluating $s_1^{-i}(c)$, check whether $m_{\Delta\to i}(1) - m_{\Delta\to i}(0)$ is greater than the cth largest message difference. If it is, compute $s_1^{-i}(c) = s_1(c) - (m_{\Delta\to i}(1) - m_{\Delta\to i}(0)) + (m_{\Delta\to r(c+1)}(1) - m_{\Delta\to r(c+1)}(0))$ where $r(c)$ gives the index of the cth largest message difference. s_0^{-i} can be computed analogously.

As stated in the main body, the empirical runtime for computing all outgoing messages from this factor is $O(N \log N)$ or $O(\log N)$ amortized per message. Runtimes for computing all outgoing messages from the factor are as follows: 10k pixels: .03s, 100k pixels: .32s, 1M pixels: 3.3s, 10M pixels: 34.5s.

8 Local Border Convexity Loss Factor

Recall from the main text that the local border convexity (LBC) loss is defined as,

$$\Delta_{y^*}^{LBC}(y) = \sum_{i \in \mathcal{F} \cup \mathcal{B}} 1\{y_i \neq y_i^*\} + \sum_{(q,\ldots,p) \in \mathcal{Q}} g(y_q, \ldots, y_p), \quad (19)$$

where $g(y_1, \ldots, y_m) = 0$ if $y_i \geq y_j$ for all $i < j$ and α otherwise.

To use this loss within a loss-augmented MAP routine, we just need to show how to compute outgoing messages from factors representing the g functions. The other terms are low order and can be added to singleton potentials as is standard. Thus, we would like to compute

$$m_{g \to i}(y_i) = \max_{y_{-i}} \left[g(y_i, y_{-i}) + \sum_{i' \neq i} m_{i' \to g}(y_{i'}) \right]. \quad (20)$$

We can compute all messages at once with a linear time dynamic programming algorithm. The variables in the scope of g form an ordered set, so we use the convention that the "start" (or "left-most") variable is the one that neighbors a foreground pixel, and the "end" or ("right-most") variable is the one that neighbors a background pixel.

Begin by constructing six arrays to cache the following values. There are three from the "left" and three from the "right":

- $L_1(i)$, which stores the maximum value of an assignment to variables 1 to i where all variables in the range are on.
- $L_2(i)$, which stores the maximum value of an assignment to variables 1 to i where at least one variable in the range has been off.
- $L_3(i)$, which stores the maximum value of an assignment to variables 1 to i where at least one variable in the range has been off, and at least one variable after the off variable(s) has been on.
- $R_1(i)$, which stores the value of an assignment to variables i to m where all variables in the range are off.
- $R_2(i)$, which stores the value of an assignment to variables i to m where at least one variable in the range has been on.
- $R_3(i)$, which stores the value of an assignment to variables i to m where at least one variable in the range has been on, and at least one variable before (i.e., with smaller index) the on variable has been off.

These arrays can be populated in linear passes like is standard in many dynamic programming algorithms. For notational convenience, also define maximums over the arrays in each direction:

$$L^*(i) = \max(L_1(i), L_2(i)) \quad (21)$$
$$R^*(i) = \max(R_1(i), R_2(i)) \quad (22)$$

From these arrays, we can easily compute each outgoing message. For a message i with value 0 $m_{g \to i}(0)$, there are three options:

$$A = L^*(i-1) + R_2(i+1) \quad (23)$$
$$B = L_3(i-1) + R^*(i+1) \quad (24)$$
$$C = L^*(i-1) + R^*(i+1) - \alpha, \quad (25)$$

then the final message is

$$m_{g \to i}(0) = \max(A, B, C). \quad (26)$$

For a message i with value 1 $m_{g \to i}(1)$, there are also three options:

$$D = L^*(i-1) + R_3(i+1) \quad (27)$$
$$E = L_2(i-1) + R^*(i+1) \quad (28)$$
$$F = L^*(i-1) + R^*(i+1) - \alpha, \quad (29)$$

then the final message is

$$m_{g \to i}(1) = \max(D, E, F). \quad (30)$$

The runtime for computing all outgoing messages is $O(N)$, or $O(1)$ amortized time per message.

9 One-sided Convexity Factor

In the 0-loss constrained MAP inference, we need to enforce the local border convexity constraint—that labelings along paths extending outward from the foreground region are monotonic and decreasing. Here we show how to compute outgoing messages from a factor that enforces this hard constraint. This is straightforward, because there are only $m+1$ legal joint settings of variables of the form $1^k 0^{m-k}$.

Using the same "left" and "right" convention of the previous section, compute six arrays in a linear pass:

- $L_1(i)$, which stores the value of setting the first i variables on.
- $L_2(i)$, which stores the maximum value of an assignment to the first i variables where the joint assignment follows the pattern $1^k 0^{i-k}$.
- $R_1(i)$, which stores the value of an assignment to variables i to m where all variables in the range are off.

- $R_2(i)$, which stores the value of an assignment to variables i to m where the joint assignment follows the pattern $1^k 0^{m-i-k}$.

The messages can then be computed:

$$m_{g\to i}(0) = L_1(i-1) + \max\left(R_1(i+1), R_2(i+1)\right) \tag{31}$$

$$m_{g\to i}(1) = \max\left(L_1(i-1), L_2(i-1)\right) + R_1(i+1). \tag{32}$$

The total computation time to compute all outgoing messages from a factor is $O(N)$ i.e., $O(1)$ amortized per message.

10 Synthetic Experiment

We created data on a 20 x 20 grid of locations. For each of 200 training cases, we assigned three special points, $A = (i_A, j_A)$, $B = (i_B, j_B)$, and $C = (i_C, j_C)$ to a random location on the grid. With each location (i, j), we associated a six dimensional feature vector, $\phi(i,j) = (f_A(i,j), f_B(i,j), f_C(i,j), g_A(i,j), g_B(i,j), g_C(i,j))$. Each feature gives a noisy measurement of the location of the special point it is associated with e.g., f_A gives a noisy detection of A's location. Features take value 0 at all except for two locations– a true location and a distractor location–which are determined as follows: for $X \in \{A, B, C\}$, $f_X(X) \sim \text{Uniform}(0, \frac{1}{\alpha})$ gives a response at the true location, $f_X(Y) \sim \text{Uniform}(0, 1)$ gives a response at distractor location Y which is drawn uniformly from the set of all locations. $g_X(X) \sim \text{Uniform}(0, \frac{.5}{\alpha})$ gives an on-average weaker response at the true location, and $g_X(Z) \sim \text{Uniform}(0, 1)$ is a *local* distractor: Z is chosen uniformly at random from the 5x5 grid centered at the true location.

Given weights w_A, w_B, w_C, the model predicts the location of special point X as $\arg\max_{ij} w_X^T \phi(i,j)$. We experimented with training according to two loss functions. First, the low order per-pixel loss gives loss of $\frac{1}{3}$ for each incorrect prediction. Second, we trained the model to optimize a high-order order-based loss, which cares only about the relative locations of the three points along the two dimensions. A loss of $\frac{1}{6}$ is incurred for each error in relative orderings. To find violated constraints under the high order loss, we use a modified version of the order-based potentials described in [4]. We used 200 images for training, 200 for validation (for choosing regularization constant C), and 500 for test.

Figure Fig. 6 shows the results. The optimal strategy to optimize the high order loss is to rely more heavily on the weaker but more local feature, while the optimal strategy to optimize the low order loss is to rely on the stronger but non-local noisy feature. The model learns appropriately under both loss functions. Interestingly, when the noise becomes very high ($\alpha = 2$), the pixel-loss-trained model is worse on both measures. In this case, for all settings of C we tried, the low-order loss uses all of its slack and pushes weights to zero, learning nothing. In this case, since the order-based loss is easier, it learns, and is able to outperform the low order loss according to both metrics.

Train \ Evaluate	Pixel Error	Order Error
$\alpha = .5$		
Pixel-loss	**7.1%**	3.2%
Order-loss	10.1%	**1.4%**
$\alpha = 1$		
Pixel-loss	**26.9%**	9.7%
Order-loss	28.2%	**5.2%**
$\alpha = 2$		
Pixel-loss	84.3%	44.9%
Order-loss	**67.5%**	**15.8%**

Figure 6: Synthetic results. Error on Pixel-based error or Order-based error for different values of α and train time loss functions.

11 Additional Qualitative Results

See Figures 2-5 for more results on Cow and for results on Aeroplane, Car, and Dog, respectively. Figure 6 shows failure modes for the PASCAL loss using examples from all of the object classes.

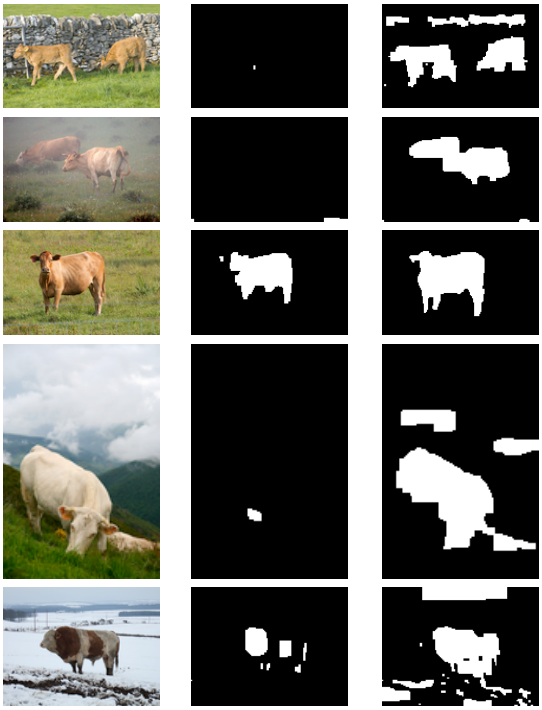

Image Pixel Loss PASCAL Loss

Figure 7: More test results on Cow dataset. Methods from left to right: (Left) Raw image. (Middle) Pixel Loss. (Right) PASCAL Loss.

Image Pixel Loss PASCAL Loss

Figure 8: Test results on Aeroplane dataset. Methods from left to right: (Left) Raw image. (Middle) Pixel Loss. (Right) PASCAL Loss.

Figure 9: Test results on Car dataset. Methods from left to right: (Left) Raw image. (Middle) Pixel Loss. (Right) PASCAL Loss.

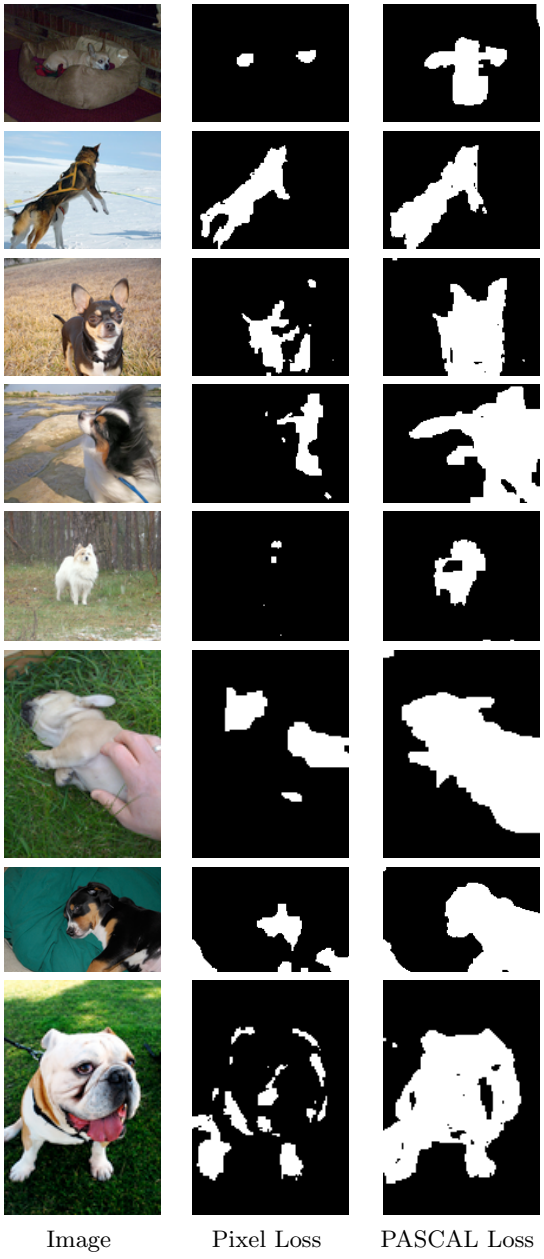

Figure 10: Test results on Dog dataset. Methods from left to right: (Left) Raw image. (Middle) Pixel Loss. (Right) PASCAL Loss.

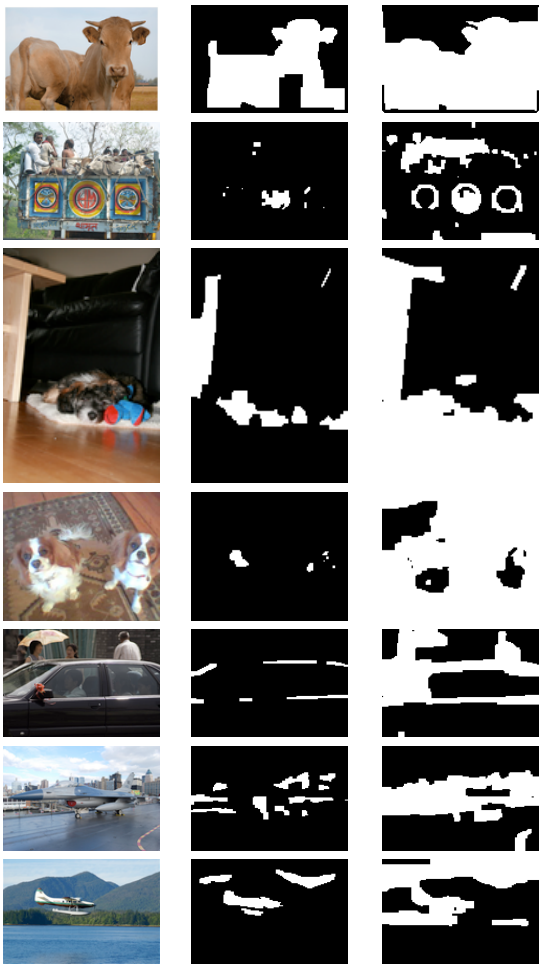

Figure 11: Typical failures of PASCAL loss-trained models. On difficult images, the PASCAL loss-trained model tends to label too many pixels as foreground. Errors are sometimes exaggerated when edge information in the image is weak, as in the second dog example. Methods from left to right: (Left) Raw image. (Middle) Pixel Loss. (Right) PASCAL Loss.