Scene Parsing with Object Instances and Occlusion Ordering

Joseph Tighe Marc Niethammer
University of North Carolina at Chapel Hill
{jtighe,mn}@cs.unc.edu

Svetlana Lazebnik
University of Illinois at Urbana-Champaign
slazebni@illinois.edu

Abstract

This work proposes a method to interpret a scene by assigning a semantic label at every pixel and inferring the spatial extent of individual object instances together with their occlusion relationships. Starting with an initial pixel labeling and a set of candidate object masks for a given test image, we select a subset of objects that explain the image well and have valid overlap relationships and occlusion ordering. This is done by minimizing an integer quadratic program either using a greedy method or a standard solver. Then we alternate between using the object predictions to refine the pixel labels and vice versa. The proposed system obtains promising results on two challenging subsets of the LabelMe and SUN datasets, the largest of which contains 45,676 images and 232 classes.

1. Introduction

Many state-of-the-art image parsing or semantic segmentation methods attempt to compute a labeling of every pixel or segmentation region in an image [2, 4, 7, 14, 15, 19, 20]. Despite their rapidly increasing accuracy, these methods have several limitations. First, they have no notion of object *instances* – given an image with multiple nearby or overlapping cars, these methods are likely to produce a blob of "car" labels instead of separately delineated instances (Figure 1(a)). In addition, pixel labeling methods tend to be more accurate for "stuff" classes that are characterized by local appearance rather than overall shape – classes such as road, sky, tree, and building. To do better on "thing" classes such as car, cat, person, and vase – as well as to gain the ability to represent object instances – it becomes necessary to incorporate detectors that model the overall object shape.

A growing number of scene interpretation methods combine pixel labeling and object detection [7, 8, 14, 11, 20, 22]. Ladický et al. [14] use the output of detectors to increase parsing accuracy for "thing" classes. However, they do not explicitly infer object instances. Kim et al. [11] and Yao et al. [22] jointly predict object bounding boxes and pixel labels. By doing so they improve the performance of both tasks. However, they rely on rather complex conditional random field (CRF) inference and apply their methods only to small datasets [6, 19] that contain hundreds of images and tens of classes. By contrast, we want to scale parsing to datasets consisting of tens of thousands of images and hundreds of classes.

In this work we interpret a scene in terms of both dense pixel labels and object instances defined by segmentation masks rather than just bounding boxes. Additionally, we order objects according to their predicted occlusion relationships. We start with our earlier region- and detector-based parsing system [20] to produce an initial pixel labeling and a set of candidate object instances for hundreds of object classes. Then we select a subset of instances that explain the image well and respect overlap and occlusion ordering constraints learned from the training data. For example, we may learn that headlights occur in front of cars with 100% overlap,[1] while cars usually overlap other cars by at most 50%. Afterwards, we alternate between using the instance predictions to refine the pixel labels and vice versa. Figure 1 illustrates the steps of our method.

Our method is related to that of Guo and Hoiem [7], who infer background classes (e.g., building, road) at every pixel, even in regions where they are not visible. They learn the relationships between occluders and background classes (e.g., cars are found on the road and in front of buildings) to boost the accuracy of background prediction. We go further, inferring not only the occluded background, but all the classes and their relationships, and use a much larger number of labels. The recent approach of Isola and Liu [10] is even closer to ours both in terms of its task and its output representation. This approach parses the scene by finding a configuration or "collage" of ground truth objects from the training set that match the visual appearance of the query image. The transferred objects can be translated and scaled to match the scene and have an inferred depth order. However, as the experiments of Section 3.2 demonstrate, our system considerably outperforms that of [10] in terms of pixel-level accuracy on the LMO dataset [15].

[1] Technically, headlights are attached to cars, but we do not make a distinction between attachment and occlusion in this work.

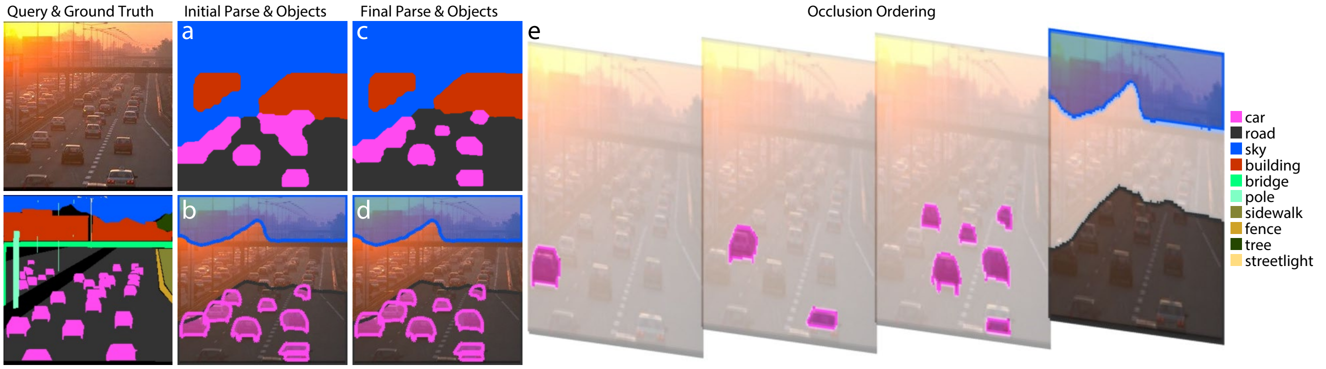

Figure 1. Overview of the proposed approach. We start with our parsing system from [20] to produce semantic labels for each pixel (a) and a set of candidate object masks (not shown). Next, we select a subset of these masks to cover the image (b). We alternate between refining the pixel labels and the object predictions until we obtain the final pixel labeling (c) and object predictions (d). On this image, our initial pixel labeling contains two "car" blobs, each representing three cars, but the object predictions separate these blobs into individual car instances. We also infer an occlusion ordering (e), which places the road behind the cars, and puts the three nearly overlapping cars on the left side in the correct depth order. Note that our instance-level inference formulation does not require the image to be completely covered. Thus, while our pixel labeling erroneously infers two large "building" areas in the mid-section of the image, these labels do not have enough confidence, so no corresponding "building" object instances get selected.

2. Inference Formulation

Given a test image, we wish to infer a semantic label at each pixel, a set of object instance masks to cover the image (possibly incompletely), as well as the occlusion ordering of these masks. We begin in Section 2.1 by describing our pixel label inference, which is largely based on our earlier work [20]. As a by-product of this inference, we generate a set of candidate instance masks (Section 2.2). Each candidate receives a score indicating its "quality" or degree of agreement with the pixel labeling, and we also define overlap constraints between pairs of candidates based on training set statistics (Section 2.3). We then solve a quadratic integer program to select a subset of instances that produce the highest total score while maintaining valid overlap relationships (Section 2.4). Next, we use a simple graph-based algorithm to recover an occlusion ordering for the selected instances. Finally, we define an object potential that can be used to recompute a pixel labeling that better agrees with the selected instances (Section 2.5).

2.1. Pixel-level Inference

We obtain an initial pixel-level labeling using our previously published parsing system [20]. Given a query or test image, this system first finds a retrieval set of globally similar training images. Then it computes two pixel-level potentials: a *region-based data term*, based on a nonparametric voting score for similar regions in the retrieval set; and a *detector-based data term*, obtained from responses of per-exemplar detectors [16] corresponding to instances in the retrieval set. The two data terms are combined using a one-vs-all SVM (as in [20], we use a nonlinear feature embedding to approximate the RBF kernel for higher accuracy). The output of this SVM for a pixel p_i and class c_i, denoted $E_{\text{SVM}}(p_i, c_i)$, gives us a unary CRF potential:

$$\psi_u(p_i, c_i) = -\log \sigma(E_{\text{SVM}}(p_i, c_i)) \,, \qquad (1)$$

where $\sigma(t) = 1/(1 + e^{-t})$ is the sigmoid function turning the raw SVM output into a probability-like score.

We infer a field of pixel labels $\mathbf{c}$ by minimizing the following CRF objective function:

$$E(\mathbf{c}) = \sum_i \underbrace{\psi_u(p_i, c_i)}_{\text{unary (eq.1)}} + \sum_i \underbrace{\psi_o(p_i, c_i, \mathbf{x})}_{\text{object potential}} + \sum_{i<j} \underbrace{\psi_{sm}(c_i, c_j)}_{\text{smoothing}} \,. \qquad (2)$$

The object potential ψ_o is not part of [20] and will be discussed in Section 2.5. It is based on the currently selected object masks, as encoded by the indicator vector $\mathbf{x}$ each of whose entries x_m is set to 1 if the corresponding candidate mask is selected and 0 otherwise. For the initial pixel label prediction, $\mathbf{x}$ is set to all zeros and ψ_o has no effect. The pairwise spatial smoothing term ψ_{sm} is the same one used in our previous work (eq. (4) in [20]). The minimization of eq. (2) is done using α-expansion [1, 13].

2.2. Candidate Instance Generation

This section explains how we generate object instance hypotheses. First, it is worth pointing out the different nature of hypotheses for "thing" and "stuff" classes. For "things" like cars, people, and fire hydrants, instances are discrete and well-defined. It is highly desirable to correctly separate multiple "thing" instances even when (or *especially* when) they are close together or overlapping. On the other hand, for "stuff" classes such as building, roads, trees, and sky, the notion of instances is a lot looser. In some cases, it may be possible to identify separate instances of buildings and trees, but most of the time we are content to treat areas occupied by these classes as undifferentiated masses. Accordingly, we manually separate all classes in our datasets into "things" and "stuff" and use different procedures to generate candidates for each.

For "things," we get candidates from per-exemplar masks transferred during the computation of the detector-

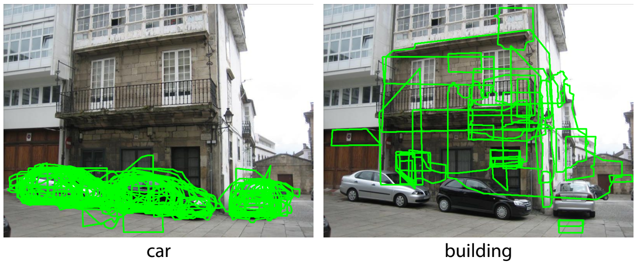

Figure 2. Candidate instance masks obtained from per-exemplar detections. There are many good "car" masks but no good "building" ones. Accordingly, we take the per-exemplar masks as our candidate "thing" instances but use a different procedure for "stuff" (see text).

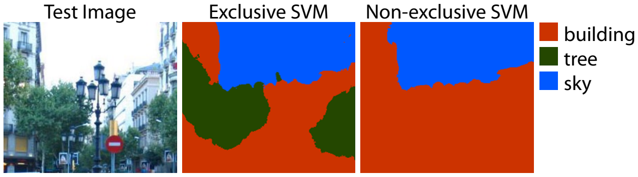

Figure 3. "Stuff" parsing using both exclusive and non-exclusive SVMs. The exclusive SVM generates good "tree" candidates, while the non-exclusive one generates a "building" candidate without holes. The connected components of both parses give us our "stuff" hypotheses (see text).

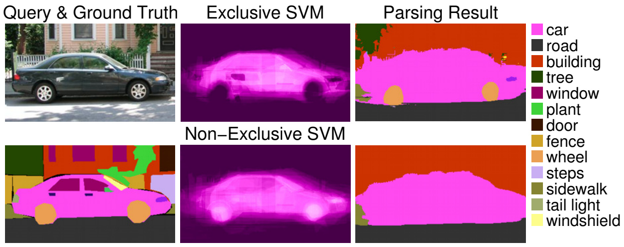

Figure 4. Exclusive vs. non-exclusive SVM output. The exclusive SVM has a very low response for areas of the car that have attached objects, like the wheels. This would penalize candidate object masks that include the wheel pixels. By contrast, the non-exclusive SVM has a high response for all parts of the car, and the resulting pixel labeling favors classes with larger area, which typically correspond to background or occluded classes.

based data term for pixel-level labeling. In that stage, the system of [20] scans the test image with per-exemplar detectors associated with every instance from the retrieval set, and for each positive detection, it transfers the corresponding exemplar's ground-truth segmentation mask into the detected location. As shown in Figure 2, these masks make fairly good candidates for "things" such as cars, but for "stuff" such as buildings, they tend to be overly fragmented and poorly localized.

To get "stuff" hypotheses that better fit the boundaries in the image, one might think of simply taking the connected components of the corresponding class labels from the initial pixel-level parse. However, the resulting masks may have artifacts due to occlusions. For example, a scene may have a building with trees in front of it (Figure 3), and if we simply pick out the "building" pixels from the initial parse, we will end up with a hypothesis that has tree-shaped holes. Instead, we would like to compute a layered representation of the image reflecting that both "building" and "tree" coexist in the same location, with the latter being in front. Fortunately, our datasets come from LabelMe where the ground truth consists of overlapping object polygons, and it allows us to learn such relationships [17].

The initial pixel-level parse is determined primarily by the combination SVM unary potentials (eq. 1). By default, the combination SVM is trained to assign to each pixel only the visible or front-most object label. For example, if a training pixel is contained within "headlight," "car," and "building" ground-truth polygons, it is used as training data only for the front-most "headlight" label. To generate better "stuff" hypotheses, we want a high response to background (occluded) classes as well. So we also train a second *non-exclusive* combination SVM where each pixel is considered a positive example of *every* ground-truth polygon that contains it. We denote the output of this combination SVM as $E_{\text{NXSVM}}(p_i, c_i)$ and the output of the original exclusive one as $E_{\text{XSVM}}(p_i, c_i)$. Figure 4 shows a comparison of the two.

We produce two pixel-level parses using $E_{\text{XSVM}}(p_i, c_i)$ and $E_{\text{NXSVM}}(p_i, c_i)$ in the unary of eq. (1), respectively, and restricting the labels to "stuff" (taking out the "things" ensures that they do not occlude the "stuff" hypotheses). The connected components of these parses give us our candidate "stuff" masks. The exclusive parse favors the foreground "stuff" objects, while the non-exclusive one favors the larger background "stuff" objects as the smaller occluding or attached objects tend to have equal or lower SVM responses and hence get smoothed away.

2.3. Instance Scores and Overlap Constraints

In order to perform instance-level inference, we need to assign a score to each candidate instance indicating its "quality" or degree of agreement with the initial pixel labeling (with a higher score indicating better quality), as well as to define overlap constraints between pairs of instances. These components will be defined in the current section.

We have found a unified scoring scheme that works equally well for for both "things" and "stuff." First, given instance m with class label c_m and segmentation mask O_m, we define an *unweighted* score $\hat{s}_m$ as the sum of non-exclusive SVM scores for its class label and every pixel p_i inside its segmentation mask:

$$\hat{s}_m = \sum_{p_i \in O_m} (1 + E_{\text{NXSVM}}(p_i, c_m)) \,. \tag{3}$$

Since any object with a negative score will be rejected by the optimization framework of Section 2.4, we move the effective decision boundary of the SVM to the negative margin so that as many reasonable candidate objects are considered as possible. It is important to use the *non-exclusive* SVM because we do not want to penalize an instance due to other classes that may occlude it. In the example of Figure 4, if we were to use the exclusive SVM to score a candidate car mask, the wheel regions would be penalized and a worse mask without the wheels would likely be preferred. Note, however, that we will still need the exclusive SVM to perform pixel-level inference, since the non-exclusive SVM tends to prefer the larger and more common classes.

So far, the scores defined by eq. (3) depend only on the SVM data term, which is computed once in the beginning and never modified. To allow the selected instances to iteratively modify the pixel labeling and vice versa, the score also needs to depend on the current labeling. To this end, we weight each instance score $\hat{s}_m$ by the percentage of pixels in the respective mask O_m that have agreement with the current pixel labels $\mathbf{c}$. At each pixel p_i inside the object mask, this agreement is given by $V_{\mathrm{p}}(c_m, c_i)$, a flag whose value is 1 if the instance label c_m can appear *behind* the pixel label c_i and 0 otherwise. For example, a candidate "car" instance is in agreement with "headlight," "window," and "license plate" pixels because all these are often seen in front of "car." The matrix of V_{p} values is learned by tabulating label co-occurrences in the training dataset, normalizing each count by the total count for the less common class, and then thresholding with a very low value (0.001 in the implementation). Our final weighted instance scores are given by

$$s_m = w_m \hat{s}_m \,, \quad \text{where} \;\; w_m = \frac{1}{|O_m|} \sum_{p_i \in O_m} V_{\mathrm{p}}(c_m, c_i)\,. \tag{4}$$

Our instance-level inference (Section 2.4) will attempt to select a combination of instance hypotheses that maximize the total score while requiring that every pair of selected objects have a valid overlap relationship. Now we explain how this relationship is defined. For any two instances with classes c_m and c_n and masks O_m and O_n, we first compute an overlap score

$$os_{mn} = \frac{|O_m \cap O_n|}{\min(|O_m|, |O_n|)}\,. \tag{5}$$

Note that we define our overlap score this way instead of the more standard intersection-over-union (I/U) in order to have a consistent score for attached objects. If we use I/U, a large window hypothesis partially overlapping with a building can have the same score as a small window hypothesis with full overlap.

Next, we encode the validity of the overlap relationship between m and n using the flag $V_{\mathrm{o}}(m, n)$, which is defined to be 1 if and only if it is possible for c_m to appear either behind or in front of c_n with an overlap score similar to os_{mn}, as determined by dataset statistics. It is important to make V_{o} depend on the overlap score, in addition to the pair of labels, because we want to allow only certain kinds of overlap for certain label pairs. For example, cars must overlap other cars by 50% or less, while headlights must overlap cars by 90% or more. For all pairs of labels (c, c'), we use the training set to build a histogram $H(c, c', os)$ giving the probability for c to be behind c' with overlap score of os (quantized into ten bins). Given instances m and n from the test image, we then determine whether their relationship is valid by thresholding the maximum of the histogram entries corresponding to both orderings:

$$V_{\mathrm{o}}(m, n) = 1_{\{\max(H(c_m, c_n, os_{mn}), H(c_n, c_m, os_{mn})) > t\}}\,. \tag{6}$$

Again we set a fairly conservative threshold of $t = 0.001$.

2.4. Instance-level Inference

This section introduces our method for selecting a subset of candidate instances that has the highest total score while maintaining valid overlap relationships. Let $\mathbf{x}$ denote a binary vector each of whose entries x_m indicates whether the mth candidate should be selected as part of our scene interpretation. We infer $\mathbf{x}$ by maximizing

$$\begin{gathered} \sum_m s_m x_m - \sum_{n \neq m} s_{mn} x_m x_n 1_{\{c_m = c_n\}} \\ \text{s.t.} \;\; \forall (x_m = 1, x_n = 1, n \neq m) \;\; V_{\mathrm{o}}(m, n) = 1\,, \end{gathered} \tag{7}$$

where c_m and c_n are the class labels of m and n, s_m is the instance score from eq. (4), and s_{mn} is is defined for any two overlapping objects of the same class in the same manner as s_m, but over the intersection of the respective object masks. By subtracting the s_{mn} term for any two selected objects of the same class, we avoid double-counting scores and ensure that any selected instance has sufficient evidence outside of regions overlapping with other instances. Without this term, we tend to get additional incorrect instances selected around the borders of correct ones. Eq. (7) can be rearranged into an integer quadratic program:

$$\text{minimize} \;\; M(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T Q \mathbf{x} - \mathbf{s}\mathbf{x} \quad \text{s.t.} \;\; A\mathbf{x} < \mathbf{b}\,, \tag{8}$$

where $\mathbf{s}$ is a vector of all object scores s_m and $Q(m, n) = s_{mn} 1_{\{c_m = c_n\}}$. The constraint matrix A is constructed by adding a row for each zero entry $V_{\mathrm{o}}(m, n)$ (that is, each pair of objects with invalid overlap). This row consists of all zeros except for elements m and n which are set to 1. Finally, $\mathbf{b}$ is a vector of ones with the same number of rows as A. With this encoding, we cannot select two candidate objects with invalid overlap, as it would result in a row of $A\mathbf{x}$ being larger than 1.

To infer the labels $\mathbf{x}$ we adopt two methods. The first is greedy inference that works by adding one candidate object at a time. In each round, it searches for the object whose addition will produce the lowest energy $M(\mathbf{x})$ (eq. 8) while still respecting all overlap constraints. This continues until each remaining object either increases the energy or violates a constraint. This method is very fast and works well. The second method is to use the integer quadratic programming solver CPlex [9]. Compared to the greedy method,

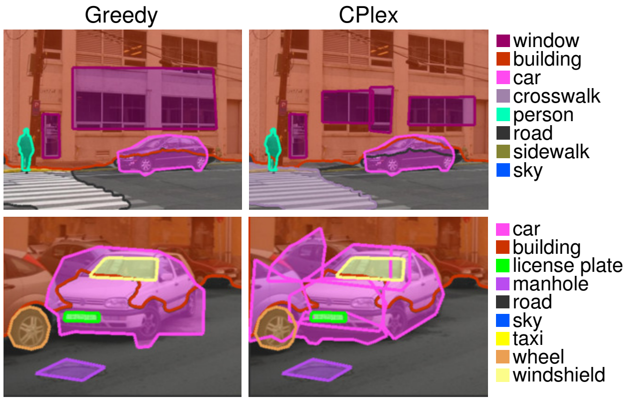

Figure 5. A comparison of our two instance-level inference methods, greedy and CPlex. In the first row, greedy inference combines six windows into one. In the second row, CPlex over-segments the foreground car, but greedy misses the two occluded cars in the background.

CPlex tends to find solutions composed of a larger number of smaller instances. These solutions have lower energies and give a slightly higher labeling accuracy overall. On the other hand, CPlex is much slower than greedy and cannot handle as many candidate objects. Figure 5 compares the two solutions on fragments of two test images; all the subsequent figures will show only the output of CPlex. Finally, note that prior to running the instance-level inference, we reject any instance less than 10% of whose pixels have the same label as the current pixel parse **c**.

2.5. Occlusion Ordering

The instances selected by the above process are guaranteed to have valid overlaps, but for each overlapping pair we do not yet know which one occludes the other. To determine this ordering, we build a graph with a node for each selected mask O_m. For each pair of overlapping masks O_m and O_n we add a pair of edges: one from m to n with edge weight equal to $H(c_m, c_n, os_{mn})$ and one from n to m with weight $H(c_n, c_m, os_{mn})$. These edges represent the situation that object m is behind object n and vice versa. For objects from the same class we weight the edges by the object size to favor larger objects to be in front of smaller ones.

To infer the occlusion order we now need to remove one edge from each pair to generate a directed acyclic graph with the highest edge weights possible. To do so, we remove the edge with the smaller weight for each pair, and check whether there are cycles remaining. If there are, we pick a random cycle and swap the edge pair with the smallest difference in weight, and continue until there are no more cycles. Finally, we perform a topological sort of the resulting graph to assign a depth plane to each object, which is useful for visualization, like in Figure 1(e).

The ordered object masks also help us to close the loop between instance-level and pixel-level inference. Specifically, they are needed to define the object potentials in eq. (2) as follows. We set $\psi_o(p_i, c_i)$ to 0 if no selected instance mask contains pixel p_i, to $-\log(0.5+\beta)$ if the front-most instance containing p_i is of class c_i, and to $-\log(0.5-\beta)$ otherwise. The constant β determines the amount by which the object potentials modulate the unaries, and it is set to 0.1 in our implementation. The form of our object potentials is inspired by [12].

We alternate between inferring objects instances (eq. 7) and pixel labels (eq. 2) until neither one changes. This tends to converge in three rounds and after the first round the results are already close to final. Figure 6 shows three rounds of alternating inference for a fairly complex scene.

3. Evaluation

3.1. Datasets and Performance Measures

We perform our experiments on two subsets of LabelMe [18] and one of its offshoots, SUN [21]. The first subset, LabelMe Outdoor or **LMO** [15], consists exclusively of outdoor scenes and has 2,488 training images, 200 test images, and 33 labels. The second one, **LM+SUN** [20], consists of both outdoor and indoor scenes and has 45,176 training images, 500 test images, and 232 labels. Both of these datasets have ground truth in the form of overlapping object polygons, which are needed both to train our system and to evaluate its performance for object inference.

For our pixel labeling, consistent with [20], we report the number of pixels correctly classified (per-pixel rate) and the average of per-class rates. The per-pixel rate indicates how well the system is doing on the large common classes, while the average per-class rate is more affected by the smaller, less common classes.

We evaluate the performance of our object instance predictions in two ways. The first, referred to as *Object P/R*, measures precision and recall of predicted instances. We define a correctly predicted instance to be one that has an I/U score greater than 0.5 with at least one ground truth polygon of the same class. If two predicted objects both have an I/U score over 0.5 with the same ground truth polygon, only one is considered correct. This is similar to the definition used in PASCAL [3] but relies on object masks rather than bounding boxes. Then precision is computed as the number of correctly predicted objects divided by the total number of predicted objects, while recall is the number of correctly predicted objects divided by the number of ground truth objects.

Our second measure, *Pixel P/R*, evaluates the pixel labeling generated by compositing the predicted instances back to front; any pixels that are not contained within a predicted object are considered unlabeled. Precision for this measure is the number of correct pixel labels divided by the total number of predicted pixels, and recall is the number of correct pixel labels divided by the total number of pixels in the test images. Note that Pixel P/R is affected by occlusion order inference, while Object P/R is not. Also, Pixel P/R tells

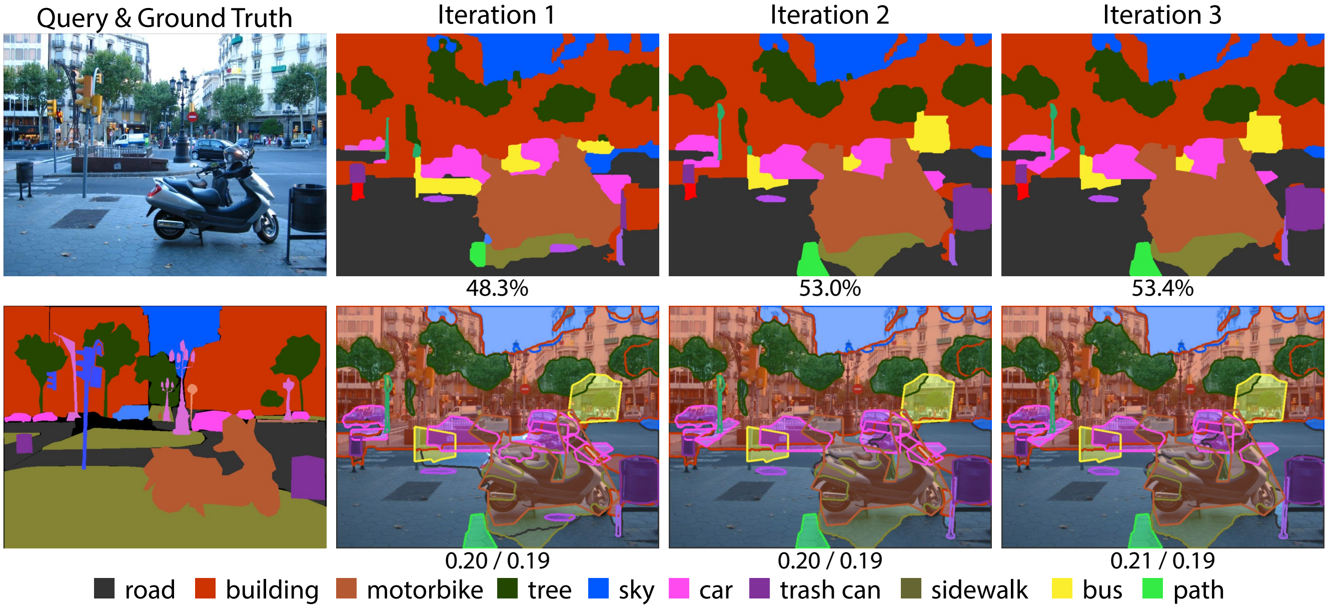

Figure 6. Alternating pixel-level and instance-level inference. For pixel labeling (top row) we show the per-pixel rate underneath each image and for instance predictions (bottom) we show Object P/R (Section 3.1). From iteration 1 to 2, we correct the pixel labeling of the trashcan on the right; from iteration 2 to 3 minor details in the background get refined. Based on the initial pixel labeling, two bus instances get predicted, and these unfortunately stay.

	LMO				LM+SUN			
	Instances	Object P/R	Pixel P/R	Pixel Parse	Instances	Object P/R	Pixel P/R	Pixel Parse
Initial Pixel Parse				**78.6 (39.3)**				61.8 (15.5)
NMS Detector	13734	3.1 / 21.4	58.0 / 50.5	77.8 (39.1)	146101	0.8 / 12.2	27.8 / 25.1	60.9 (15.1)
NMS SVM	3236	11.8 / 18.4	75.7 / 62.0	78.1 (38.8)	12839	9.3 / 12.9	53.3 / 52.8	61.8 (15.9)
Greedy	918	**44.3** / 20.0	**75.4 / 71.8**	78.4 (38.5)	4909	**24.5** / 13.1	**60.9** / 59.8	**62.1 (16.2)**
CPlex QP	993	42.8 / **21.0**	**75.4 / 71.8**	78.4 (38.6)	5435	22.3 / **13.3**	**60.9 / 59.9**	**62.1 (16.2)**

Table 1. Comparison of our instance-level inference baselines (NMS Detector, NMS SVM) and proposed methods (Greedy, CPlex QP). The top row shows our initial pixel labeling accuracy using the system of [20] with an approximate RBF kernel. The overall per-pixel rate is listed first and the average per-class rate is in parentheses. As expected, these numbers are almost identical to those in [20]. Below, the results for the four instance-level inference methods are reported after three iterations of alternating pixel and object inference. The Instances column gives the total number of instances selected in the test set by each method. The Object P/R and Pixel P/R measures are defined in Section 3.1.

us how well our instance predictions can produce a pixel parse without relying on dense CRF unaries or smoothing.

3.2. Experiments

The primary goal of our experiments is to validate our chief contribution, the instance-level inference framework. Since there are no existing methods capable of inferring object masks and occlusion ordering for hundreds of classes, we have implemented two baselines to compare against. Both are used to replace the instance-level inference of Section 2.4; all the other components of our system stay the same, including the estimation of occlusion ordering for the selected instances (Section 2.5), which is needed to compute the Pixel P/R measures.

Our first baseline, named *NMS Detector*, minimizes the reliance on the pixel parsing when inferring objects. It takes per-exemplar detection masks as candidate instances for both “things” and “stuff” and scores each candidate with the detector responses. To prevent the output from being overrun by false positives, we restrict the candidate instances to classes that appear in the pixel labeling, but this is the only way in which the pixel labeling is used. Simple greedy non-maximal suppression is performed on each class individually with a threshold of 50% overlap in the same manner as [5] to infer the final objects. Table 1 shows that this setup produces far more object instance predictions and thus has a far lower object precision. The object recall is fairly high but if we look at the Pixel P/R scores we can see that it fails to produce accurate pixel predictions.

For our second baseline, named *NMS SVM*, we use the same candidate objects as in Section 2.2 and score them as in Section 2.3. However, instead of the inter-class overlap constraints used in Section 2.3 we again use greedy non-maximal suppression. Thus, when compared to our proposed inference method, this baseline shows the effectiveness of overlap constraints. As can be seen in Table 1, NMS SVM produces two to three times the number of instance predictions over our proposed method, and has lower Object P/R and Pixel P/R performance.

In addition to the two baselines, Table 1 also shows results using both greedy inference and the CPlex solver [9].

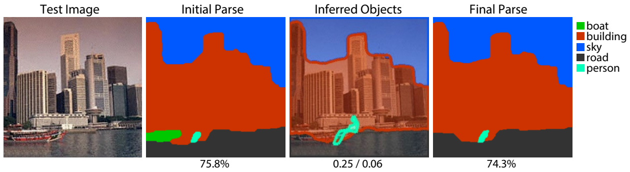

Figure 7. LMO test example where instance-level inference ends up hurting the pixel-level labeling. The initial pixel labeling for the boat is fairly good, but the water gets mislabeled as "road." Since "boat" cannot overlap "road," a boat instance is never turned on, which in turn decreases the pixel labeling performance.

The greedy inference tends to infer fewer object instances than CPlex and has a higher object precision, but it does miss some objects and thus has a lower object recall.

Somewhat disappointingly, instance-level inference does not give us any significant gains in pixel accuracy over our own previous system [20]. By comparing the initial pixel parse numbers in the top row of Table 1 to the final ones for the greedy and CPlex inference, we can see that our overall and average per-class rates drop slightly for the LMO dataset and increase slightly for LM+SUN. As we have learned from our experiments, alternating between pixel-level and instance-level inference only tends to improve performance in complex scenes where overlap relationships can help correct errors (e.g., in Figure 6, some wrong "building" labels get corrected to "trashcan"). Because most images in the LMO dataset are simplistic, with few overlapping instances, our iterative method does not produce any improvement. What is worse, in some cases, enforcing sensible overlap constraints can actually hurt the pixel-level accuracy. For example, in the LMO test image shown in Figure 7, the water on the bottom initially gets incorrectly but confidently labeled as "road." Then, though there is a decent initial "boat" labeling as well as good candidate "boat" masks, these masks cannot be selected as they have an inconsistent overlap with "road." Because there are so few boats in the test images, this brings down the "boat" class rate for the entire test set from 14.8% to 2.3%, and is almost single-handedly responsible for the drop in average per-class rate from the initial to the final pixel labeling accuracy on LMO. As for LM+SUN, it has more complex and varied scenes, and we do see small gains from initial to final pixel labeling accuracy on that dataset.

Finally, it is interesting to compare our results to those of Isola et al. [10], who also predict object instances by transferring ground truth masks from the training set. They report a per-pixel performance of 70.0% on the LMO dataset. If we just use our predicted objects to produce a pixel labeling we achieve a pixel accuracy of 71.8% (the recall from our Pixel P/R measure in Table 1) and our pixel parsing result has an accuracy of 78.4%, clearly outperforming [10].

4. Discussion

We have demonstrated a system capable of predicting object instances and their occlusion ordering in complex scenes, thus expanding the representational capability of existing pixel-level scene parsing methods. To improve our system, one key direction is generating better candidate instances. Currently, we manually divide our label set into "stuff" and "thing" classes and use different methods to generate candidate objects for each. To an extent, this division is arbitrary, as many classes, such as trees, can sometimes appear as "things" with well-defined boundaries, and sometimes as diffuse "stuff." We would like to explore ways of generating candidate object masks that would not rely on a hard things-vs-stuff split. We also plan to further investigate the problem of using the inferred object instances to update the pixel-level labeling so as to achieve more significant gains from alternating inference.

Acknowledgments. This research was supported in part by NSF grants IIS 1228082 and CIF 1302588, DARPA Computer Science Study Group (D12AP00305), Microsoft Research Faculty Fellowship, Sloan Foundation, and Xerox University Affairs Committee.

References

[1] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *PAMI*, 26(9):1124–37, Sept. 2004. 2

[2] D. Eigen and R. Fergus. Nonparametric image parsing using adaptive neighbor sets. In *CVPR*, 2012. 1

[3] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2008 (VOC2008) Results. http://www.pascal-network.org/challenges/VOC/voc2008/workshop/index.html. 5

[4] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Scene parsing with multiscale feature learning, purity trees, and optimal covers. In *ICML*, 2012. 1

[5] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *PAMI*, 32(9):1627–1645, 2010. 6

[6] S. Gould, R. Fulton, and D. Koller. Decomposing a scene into geometric and semantically consistent regions. In *ICC*, 2009. 1

[7] R. Guo and D. Hoiem. Beyond the line of sight: labeling the underlying surfaces. In *ECCV*, 2012. 1

[8] D. Hoiem, A. A. Efros, and M. Hebert. Putting objects in perspective. *IJCV*, 80(1):3–15, 2008. 1

[9] IBM. Cplex optimizer. http://www.ibm.com/software/commerce/optimization/cplex-optimizer/, Oct. 2013. 4, 6

[10] P. Isola and C. Liu. Scene collaging: Analysis and synthesis of natural images with semantic layers. In *ICCV*, Dec 2013. 1, 7

[11] B. Kim, M. Sun, P. Kohli, and S. Savarese. Relating things and stuff by high-order potential modeling. In *ECCV Workshop on Higher-Order Models and Global Constraints in Computer Vision*, 2012. 1

[12] J. Kim and K. Grauman. Shape sharing for object segmentation. In *ECCV*, 2012. 5

[13] V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? *PAMI*, 26(2):147–59, Feb. 2004. 2

[14] L. Ladický, P. Sturgess, K. Alahari, C. Russell, and P. H. Torr. What, where & how many? Combining object detectors and CRFs. In *ECCV*, 2010. 1

[15] C. Liu, J. Yuen, and A. Torralba. Nonparametric scene parsing via label transfer. *PAMI*, 33(12):2368–2382, 2011. 1, 5

[16] T. Malisiewicz, A. Gupta, and A. A. Efros. Ensemble of exemplar-SVMs for object detection and beyond. In *ICCV*, 2011. 2

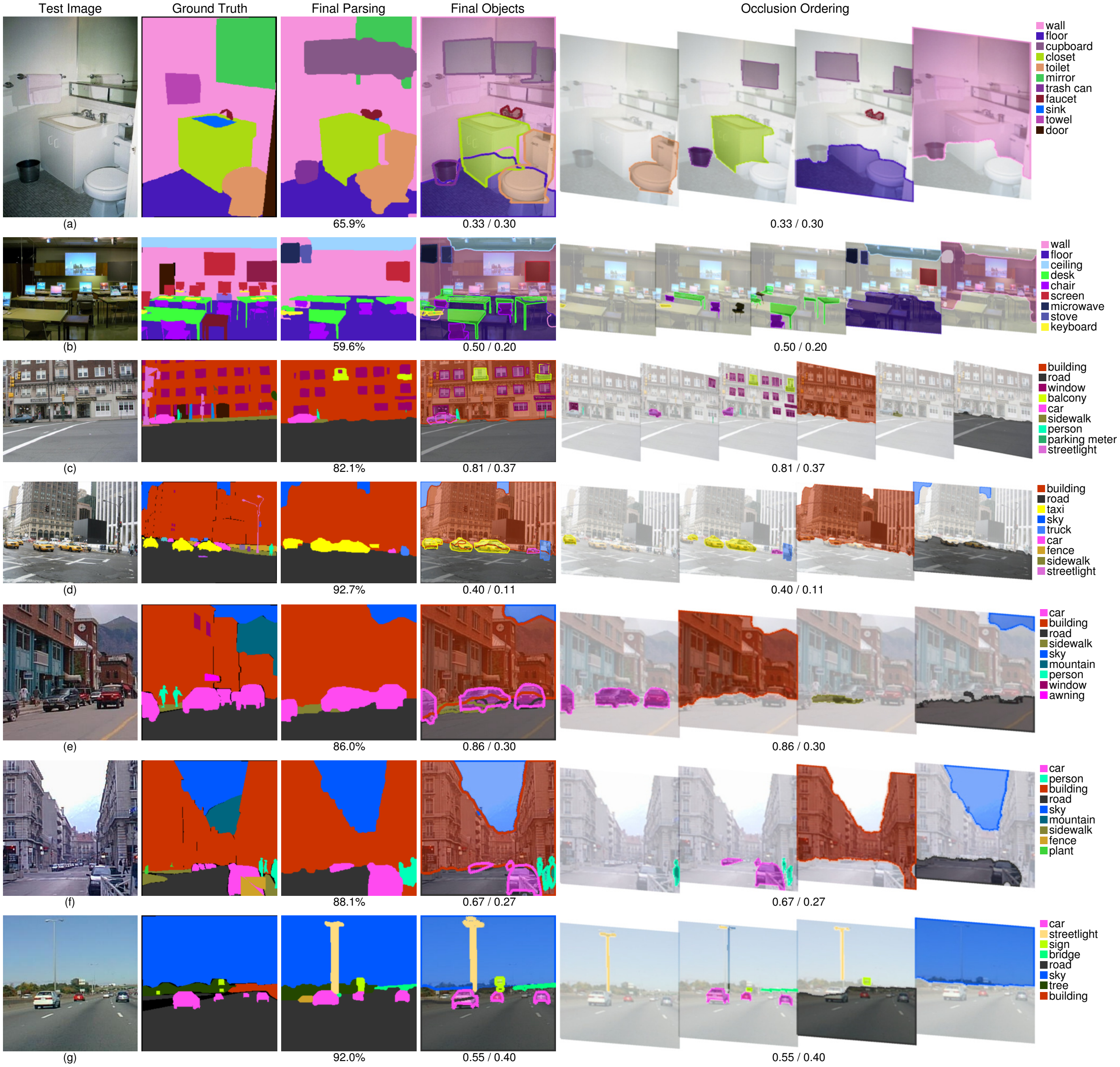

Figure 8. Example results on LMO (a-c) and LM+SUN (d-g). For pixel labeling we report the per-pixel rate and for instance prediction we report Object P/R under each image. In (a), we find the toilet, cabinet, a faucet, and even a trashcan that does not appear in the ground truth. We also incorrectly infer a number of cabinets above the sink. In (b), we find four desks, three of the chairs, and a keyboard. In (c), we find a lot of windows, a person, a car, a parking meter and even two balconies. There is a small piece of the sidewalk in the second occlusion slice, which is otherwise empty. In (d), the blob of taxis is split up into individual instances, and even a small partially occluded fourth taxi is correctly identified. In (e), a blob of cars is likewise split up into two instances. In (f), three people are correctly identified on the right side. In (g), the streetlight, missing from ground truth, is found, though three instances are predicted because there turns out to be very little overlap between the three thin masks and thus they form a valid configuration.

[17] B. C. Russell and A. Torralba. Building a database of 3d scenes from user annotations. In *CVPR*, 2009. 3

[18] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. LabelMe: A database and web-based tool for image annotation. *IJCV*, 77(1-3):157–173, 2008. 5

[19] J. Shotton, J. M. Winn, C. Rother, and A. Criminisi. TextonBoost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context. *IJCV*, 81(1):2–23, 2009. 1

[20] J. Tighe and S. Lazebnik. Finding things: Image parsing with regions and per-exemplar detectors. In *CVPR*, Jun 2013. 1, 2, 3, 5, 6, 7

[21] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *CVPR*, 2010. 5

[22] J. Yao, S. Fidler, and R. Urtasun. Describing the scene as a whole: Joint object detection, scene classification and semantic segmentation. In *CVPR*, 2012. 1