Supplementary Material – Can We Learn Heuristics For Graphical Model Inference Using Reinforcement Learning?

Safa Messaoud, Maghav Kumar, Alexander G. Schwing
University of Illinois at Urbana-Champaign
{messaou2, mkumar10, aschwing}@illinois.edu

Recall, given an image x, we are interested in predicting the semantic segmentation $y = (y_1, \dots, y_N) \in \mathcal{Y}$ by solving the inference task defined by a Conditional Random Field (CRF) with nodes corresponding to superpixels. Hereby, N denotes the total number of superpixels. The semantic segmentation of a superpixel $i \in \{1, \dots, N\}$ is referred to via $y_i \in \mathcal{L} = \{1, \dots, |\mathcal{L}|\}$, which can be assigned one out of $|\mathcal{L}|$ possible discrete labels from the set of possible labels $\mathcal{L}$. We formulate the inference task as a Markov Decision Process that we study using two reinforcement learning algorithms: DQN and MCTS. Specifically, an agent operates in $t \in \{1, \dots, N\}$ time-steps according to a policy $\pi(a_t|s_t)$ which encodes a probability distribution over actions $a_t \in \mathcal{A}_t$ given the current state s_t. The current state subsumes the indices of all currently labeled variables $I_t \subseteq \{1, \dots, N\}$ as well as their labels $y_{I_t} = (y_i)_{i \in I_t}$, *i.e.*, $s_t \in \{(I_t, y_{I_t}) : I_t \subseteq \{1, \dots, N\}, y_{I_t} \in \mathcal{L}^{|I_t|}\}$. The policy selects one superpixel and its corresponding label at every time-step.
In this supplementary material we present:

1. **Appendix A**: Further training and implementation details
2. **Appendix B**: Further details on the policy network
   - **B 1**: DQN
   - **B 2**: MCTS
3. **Appendix C**: Additional Results
   - **C 1**: Comparison of the reward schemes
   - **C 2**: Visualization of the learned embeddings
   - **C 3**: Learned policies
   - **C 4**: Qualitative results

A: Further Training and Implementation Details

PSPNet, TrackR-CNN and and the hypercolumns from VGGNet are not fine-tuned. Only the graph policy net is trained. For MOTS, we apply our model to every frames of the video. For MCTS, we set the number of simulations during exploration to 50 and the simulation depth to 4. At test time, we run 20 simulations with a depth of 4. The models are trained for 10 epochs, equivalent to around $375,000$ training iterations for Pascal VOC and $188,000$ for MOTS. The parameters of the energy function, α_p, β_p, w_b, c_b, λ_b and C, are obtained via a grid search on a subset of 500 nodes from the training data. The number of iterations K of the graph neural network is set to 3. The dimension F of the node features b_i equals 85 for Pascal VOC and 30 for MOTS, consisting of the unary distribution, the unary distribution entropy, and features of the bounding box. The bounding box features are the confidence and label of the bounding box, its unary composition at the pixel level, percentage of overlap with other bounding boxes and their associated labels and confidence. The embedding dimension p is 32 for Pascal VOC and 16 for MOTS. As an optimizer, we use Adam with a learning rate of 0.001.

B: Further details on the policy network (DQN, MCTS)

B1: DQN

Both for DQN and MCTS, three different sets of actions are encouraged at training iteration t:

- $\mathcal{M}_1^{(t)}$: Selecting nodes adjacent to the already chosen ones in the graph, at iteration t. Otherwise, the reward will only be based on the unary terms as the pairwise term is only evaluated if the neighbors are labeled ($t = 2$ in Tab. 1 in the main paper).

  We assign a score $M_1(s_t, a_t)$ to every available action $a_t = (i_t, y_{i_t}) \in \mathcal{A}_t$ to encourage the exploration of the set $\mathcal{M}_1^{(t)}$:

$$M_1(s_t, a_t) = \frac{|\{j : (j \notin I_t) \text{ and } (i_t, j) \in \mathcal{E}\}|}{|\{j : (i_t, j) \in \mathcal{E}\}|}. \tag{1}$$

- $\mathcal{M}_2^{(t)}$: Selecting nodes with the lowest unary distribution entropy, at iteration t. A low entropy indicates a high confidence of the unary deep net. Hence, the labels of the corresponding nodes are more likely to be correct and would provide useful information to neighbors with higher entropy in the upcoming iterations. We assign a score $M_2(s_t, a_t)$ to every available action $a_t = (i_t, y_{i_t}) \in \mathcal{A}_t$ to encourage the exploration of the set $\mathcal{M}_2^{(t)}$:

$$M_2(s_t, a_t) = \frac{\exp\left(-S_{i_t}\right)}{\sum_{j \in \{1,\dots,N\} \setminus I_t} \exp\left(-S_j\right)}. \tag{2}$$

  Here, S_{i_t} denotes the entropy of the unary distribution evaluated at node i_t.

- $\mathcal{M}_3^{(t)}$: Assigning the same label to nodes forming the same higher order potential at iteration t, *i.e.*,

$$M_3(s_t, a_t) = \begin{cases} 1 & \text{if } y_{i_t} = \underset{k \in \mathcal{L}}{\operatorname{argmax}} \sum\limits_{\{j : j \in I_t \text{ and } (i_t, j) \in \mathcal{C}\}} \mathbb{1}_{\{y_j = k\}} \\ 0 & \text{otherwise} \end{cases}. \tag{3}$$

For **DQN**, at train time, the next action a_t is selected as follows:

$$a_t^* = \begin{cases} \underset{a_t \in \mathcal{A}_t}{\operatorname{argmax}}\, Q(s_t, a_t; \theta) & \text{with probability } \epsilon \\ \underset{a_t \in \mathcal{A}_t}{\operatorname{argmax}}\, M_1(s_t, a_t) & \text{with probability } (1-\epsilon)/4 \\ \underset{a_t \in \mathcal{A}_t}{\operatorname{argmax}}\, M_2(s_t, a_t) & \text{with probability } (1-\epsilon)/4 \\ \underset{a_t \in \mathcal{A}_t}{\operatorname{argmax}}\, M_3(s_t, a_t) & \text{with probability } (1-\epsilon)/4 \\ \text{Random} & \text{with probability } (1-\epsilon)/4. \end{cases} \tag{4}$$

Here ϵ is a fixed probability modeling the exploration-exploitation tradeoff. At test time, $a_t^* = \underset{a_t \in \mathcal{A}_t}{\operatorname{argmax}}\, Q(s_t, a_t; \theta)$.

B2: MCTS

As described in Sec 3.5 of the main paper, for a given graph $G(V, \mathcal{E}, w)$, MCTS operates by constructing a tree, where every node corresponds to a state s and every edge corresponds to an action a. The root node is initialized to $s_1 = \emptyset$. Every node stores three statistics: 1) $N(s)$, the number of times state s has been reached, 2) $N(a|s)$, the number of times action a has been chosen in node s in all previous simulations, and 3) $\tilde{r}(s, a)$, the averaged reward across all simulations starting at state s and taking action a. A simulation involves three steps : 1) selection, 2) expansion and 3) value backup. After running n_{sim} simulations, an empirical distribution $\pi^{\text{MCTS}}(a|s) = \frac{N(a|s)}{N(s)}$ is computed for every node. The next action is then chosen according to π^{MCTS}. A policy network $\pi_\theta(a|s)$ is trained to match a distribution π^{MCTS} constructed through these simulations. In the following, we provide more details about each of these steps.

Selection corresponds to choosing the next action given the current node s_t, based on four factors : 1) a variant of the probabilistic upper confidence bound (PUCB) [2] given by $U(s_t, a_t; \theta) = \frac{\tilde{r}(s_t, a_t)}{N(a_t|s_t)} + \pi_\theta(a_t|s_t)\frac{\sqrt{N(s_t)}}{1+N(a_t|s_t)}$, 2) $M_1(s_t, a_t)$ 3) $M_2(s_t, a_t)$ and 4) $M_3(s_t, a_t)$ similarly to DQN in **Appendix B1**. Formally,

$$a_t^* = \underset{a_t \in \mathcal{A}_t}{\text{argmax}} \begin{cases} U(s_t, a_t; \theta) + M_1(s_t, a_t) & \text{with probability } \frac{1}{3} \\ U(s_t, a_t; \theta) + M_2(s_t, a_t) & \text{with probability } \frac{1}{3} \\ U(s_t, a_t; \theta) + M_3(s_t, a_t) & \text{with probability } \frac{1}{3} \end{cases}. \tag{5}$$

Expansion consists of constructing a child node for every possible action from the parent node s_t. The possible actions include the nodes which have not been labeled. The child nodes' cumulative rewards and counts are initialized to 0. Note that selection and expansion are limited to a depth d_{sim} starting from the root node in a simulation.

Value backup refers to back-propagating the reward from the current node on the path to the root of the sub-tree. The visit counts of all the nodes in the path are incremented as well.

Final Labeling: Once n_{sim} simulations are completed, we compute π^{MCTS} for every node. The next action a_t from the root node is decided according to $\pi^{\text{MCTS}}(a_t|s_t)$: $a_t \sim \pi^{\text{MCTS}}(a_t|s_t)$ at train time and $a_t = \text{argmax}_{a_t \in \mathcal{A}_t}\, \pi^{\text{MCTS}}(a_t|s_t)$ at inference. The next node becomes the root of the sub-tree. The experience (s_t, π^{MCTS}) is stored in the replay buffer. The whole process is repeated until all N nodes in the graph G are labeled. We summarize the MCTS training algorithm below in Alg. 1. Note that we run 10 episodes per graph during training, but for simplicity we present the training for a single episode per graph.

Algorithm 1: Monte Carlo Tree Search Training

input : Head node: s_1, n_{sim}: number of simulations, d_{sim} : depth of simulations
output: A labeling $y \in \mathcal{Y}$ for all the nodes V

// Looping over the graphs from the dataset
for ***all*** $G(V, \mathcal{E}, w)$ **do**
  // Initialization
  $s_1 = \emptyset$
  $\tilde{r}(s_i, a) = 0, \forall s_i, i \in \{1, \dots, N\}, \forall a$
  // Looping over graph nodes V
  for $t = 1$ **to** N **do**
    // Running simulations
    for $n = 1$ **to** n_{sim} **do**
      // Create and expand a sub-tree
      for $j = t$ **to** $t + d_{sim}$ **do**
        Select a_j according to Eq. (5) and advance temporary state in sub-tree
      end
      Backup rewards along the visited nodes in the simulations
      Update node visit counts
    end
    Compute tree policy π^{MCTS} with visit counts
    Select the next action $a_t \sim \pi^{\text{MCTS}}(a_t|s_t)$
    Update the root node $s_{t+1} \leftarrow s_t \oplus a_t$
    Store (s_t, π^{MCTS}) in Replay Buffer
  end
  Sample M examples from Replay Buffer to update neural network parameters using **Eq. (4)** in the main paper
end

C: Additional Results

C1: Comparing Reward Schemes

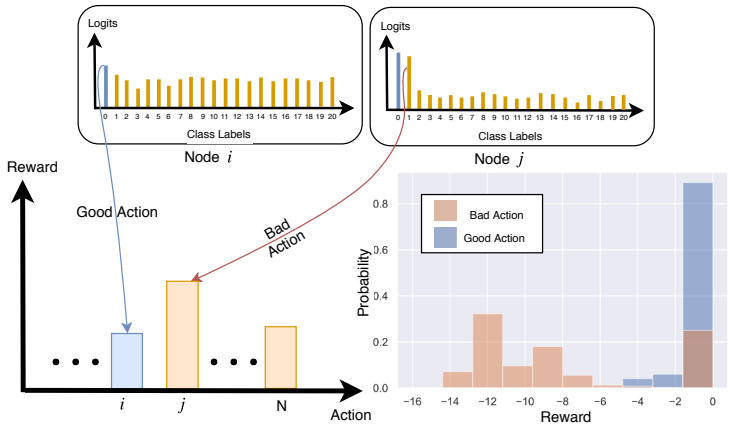

Figure 1. Explanation of the low performance of the first reward scheme ($r_t = -E_t + E_{t-1}$).

In Tab. 2 of the main paper, we observe that the second reward scheme ($r_t = \pm 1$) generally outperforms the first one ($r_t = -E_t + E_{t-1}$). This is due to the fact that, the rewards for wrong actions, in this scheme, can be higher than the ones for good actions. Specifically, in Fig. 1 we plot the distribution of the rewards of good actions (in blue) and the one of wrong actions (in orange) for $50,000$ randomly chosen actions from the replay memory. To better illustrate the cause, we consider

the unary energy case and visualize the class distribution of two nodes i and j. If we label node i to be of class 0 (good action), and node j to be of class 1 (wrong action), the resulting rewards are $f_i(0)$ for node i and $f_j(1)$ for node j. Note that $f_i(0) < f_j(1)$, since the distribution of the labels in case of node i is almost uniform, whereas the mass for node j is put on the first two labels.

C2: Visualization of the learned embeddings

In Tab. 2 in the main paper, we observe that our model can produce better segmentations than the ones obtained by just optimizing energies. Guided by the reward and due to network regularization, the policy net captures contextualized embeddings of classes beyond energy minimization. Intuitively, a well calibrated energy function yields rewards that are well correlated with F1-scores for segmentation. When TSNE-projecting the policy nets node embeddings for Pascal VOC data into a 2D space, we observe that they cluster in 21 groups, as illustrated in Fig. 2.

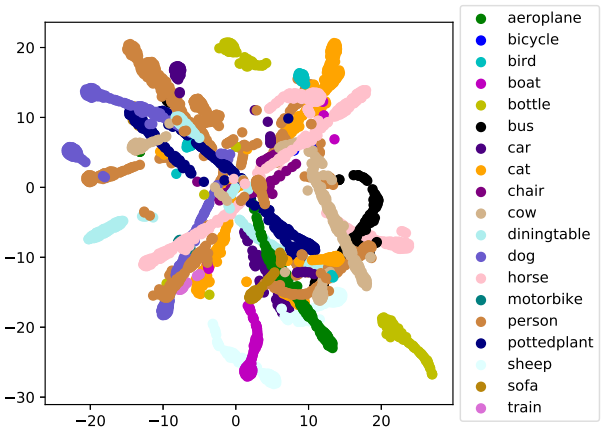

Figure 2. Visualization of the learned embeddings.

C3: Learned Policies

In Fig. 3, we visualize the learned greedy policy. Specifically, we show the probability map across consecutive time steps. The probability maps are obtained by first computing an N dimensional score vector $\phi((i,\cdot)|s_t) = \sum_{y \in \mathcal{L}} \pi((i,y)|s_t)$ $\forall i \in \{1,\ldots,N\}$ by summing over all the label scores per node and then normalizing $\phi((i,\cdot)|s_t)$ to a probability distribution over the non selected superpixels $i \in \{1,\ldots,N\} \setminus I_t$. The selected nodes are colored in white. The darker the superpixel, the smaller the probability of selecting it next. We found that the heuristic learns a notion of smoothness, selecting nodes that are in close proximity and of the same label as the selected ones. Also, the policy learns to start labeling the nodes with low unary distribution entropy, then decides on the ones with higher entropy.

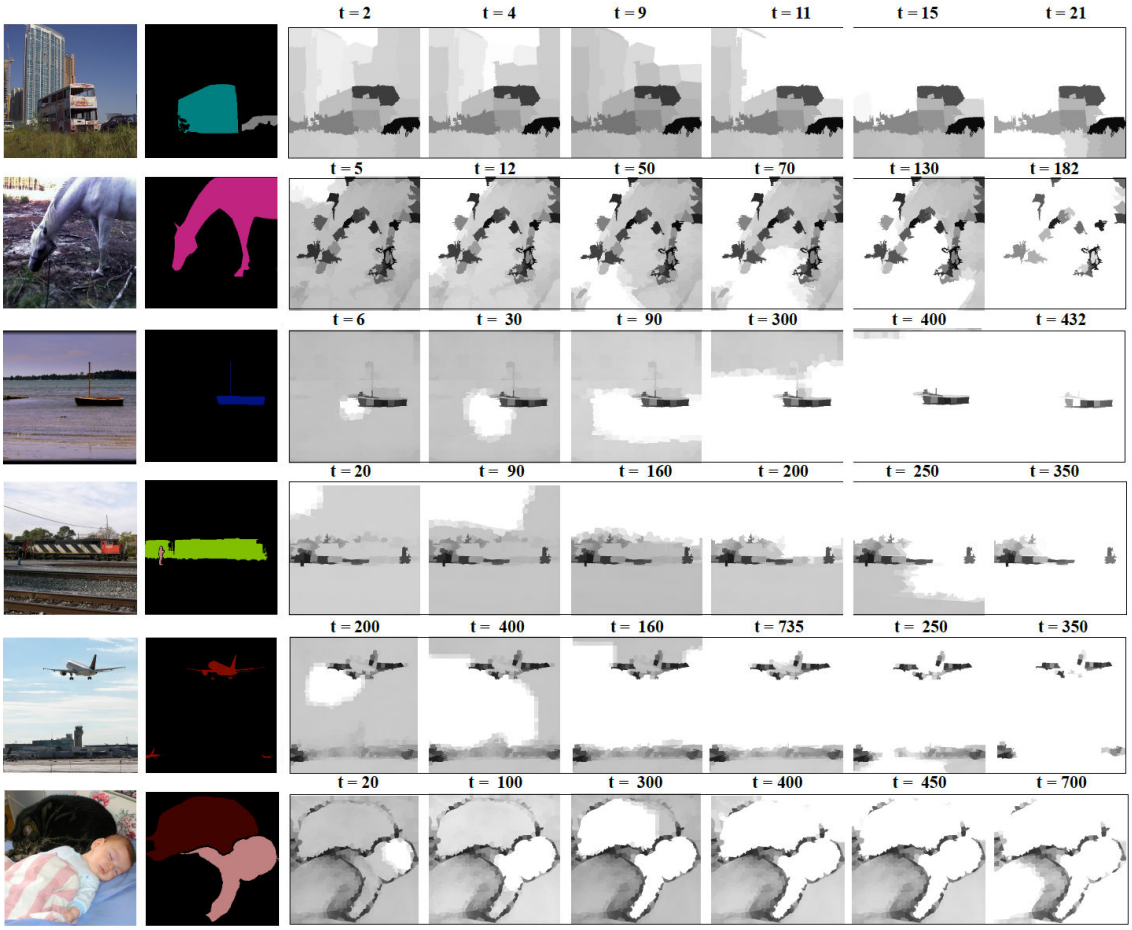

Figure 3. Visualization of our learned policy.

C4: Additional qualitative results

In the following, we present additional qualitative results. In Fig. 4 and Fig. 5, we present the segmentation results of our policy on examples from the Pascal VOC and the MOTS datasets respectively. The pairwise potential, together with the superpixel segmentation helped reduce inconsistencies in the unaries obtained from PSPNet/TrackR-CNN across all the examples. HOP1 resulted in better learning the boundaries of the objects. The energy which includes the HOP2 potential provides the best results across all energies as it helped better segment overlapping objects.

We include additional results comparing PSPNet/TrackR-CNN, DQN and MCTS outputs for the energy function with unary, pairwise, HOP1 and HOP2 potentials in Fig. 6 and Fig. 7. The policies trained with DQN/MCTS improve over the PSPNet/TrackR-CNN results across almost all our experiments. Additional failure cases are presented in Fig. 8 and Fig. 9.

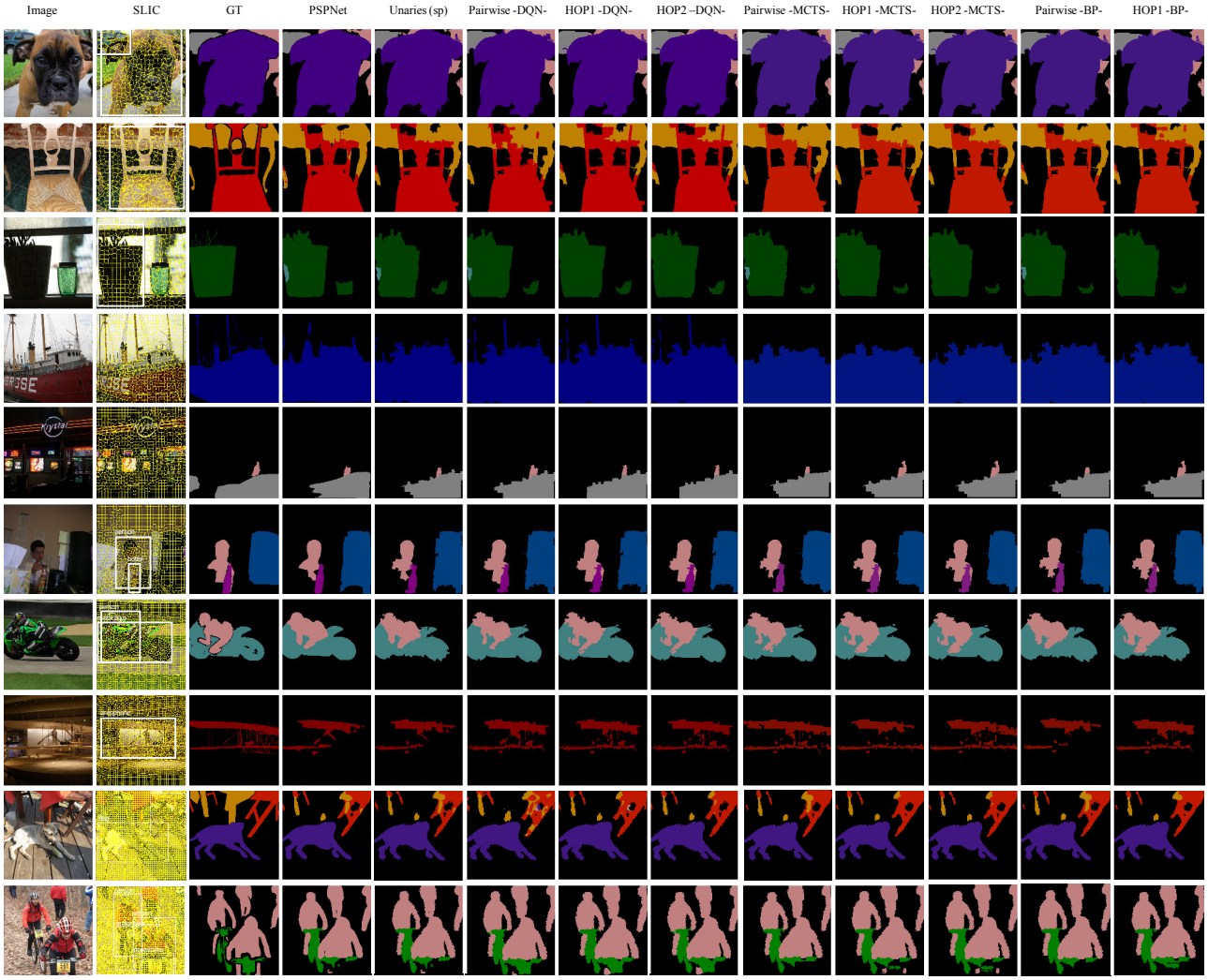

Figure 4. Output of our method for different potentials for Pascal VOC.

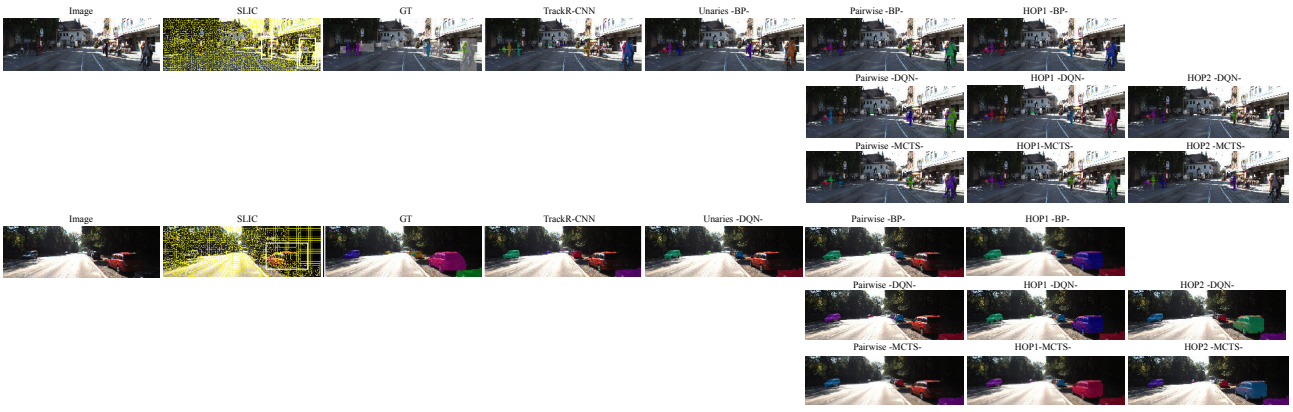

Figure 5. Output of our method for different potentials for MOTS.

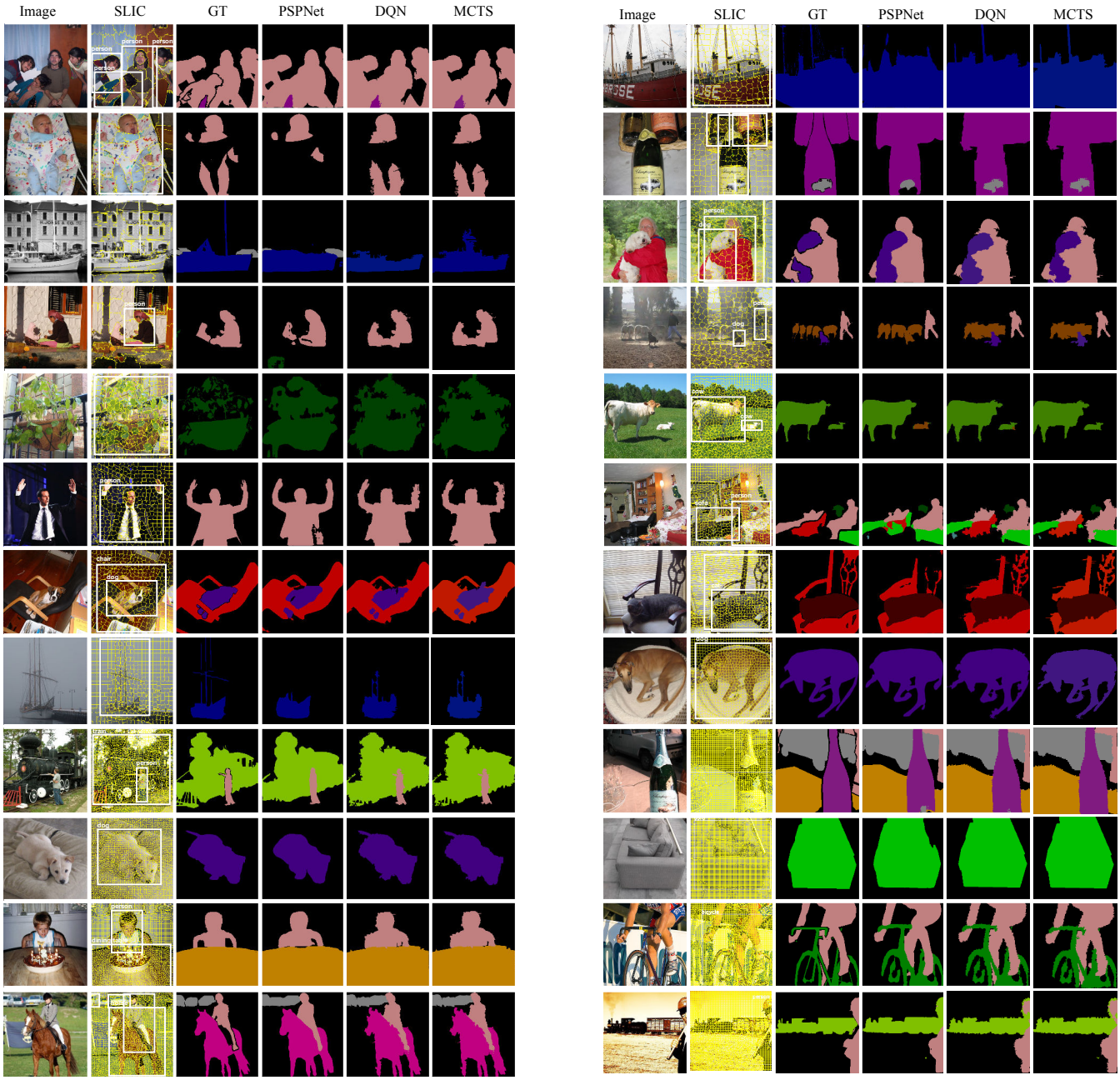

Figure 6. Additional success cases on Pascal VOC.

Figure 7. Additional success cases on MOTS.

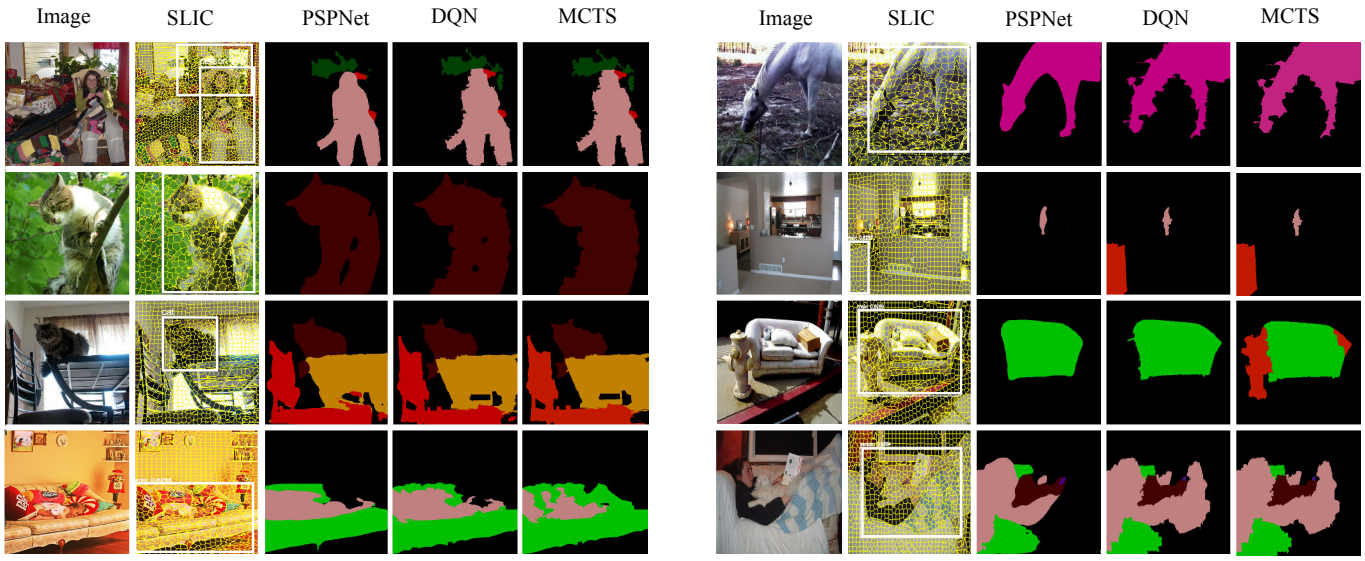

Figure 8. Additional failure cases on Pascal VOC.

Figure 9. Additional failure cases on MOTS.

References

[1] C. D. Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230, 2011.

[2] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017. 3

[3] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.