Supplementary Material:
Learning Dynamic Hierarchical Models for
Anytime Scene Labeling

Buyu Liu\textsuperscript{1,2} and Xuming He\textsuperscript{2,1}

\textsuperscript{1}The Australian National University, \textsuperscript{2}Data61, CSIRO
\{buyu.liu,xuming.he\}@anu.edu.au

1 Dataset Details

**CamVid** [1] CamVid consists of 701 images captured during the daytime and dusk and composed of 11 semantic classes. We follow the data split in [1] for training and test. We use the same resolution (240 × 320) and data preparation as [2, 3] to have a fair comparison. We note that there is a large gap between performances obtained from models that learned in high-resolution images and that of our setting. Specifically, we can obtain 9% and 10.8% performance improvement with high-resolution images of CamVid in terms of pixel-level accuracy and per-class criteria, respectively.

**Standford Background** [4] SBG has 715 images from urban and natural scene composed of 8 semantic classes. We follow the 472/143 split for training and test and results are reported under 5-fold cross validation as in [4].

**Siftflow** [5] Siftflow consists of 2688 images from 8 typical natural scenes. Every image has 256 × 256 pixels and they belong to one of 33 semantic classes. We use the 2488/200 split for training and test provided by [2].

2 Implementation Details

We will describe feature extraction, cost computation, policy learning meta feature design (cf. Sec. 5.1) and subset grouping method (cf. Sec. 4.1) in the following.

2.1 Feature Extraction

We consider the following four feature sets in our experiments. The cost of each feature type measures the computation time for an entire image. We note that this cost can be further reduced by efficient implementation of local features, and our framework can easily accommodate different cost evaluation methods.

- Darwin features are the probabilistic scores of the unary predictions trained using the Darwin [6] system.
- Geometric, Color, Texture and Position are extracted by the STAIR package [7]. We use average pooling to obtain the region descriptor.
- HoG, SIFT, LBP features are obtained from VLFeat [8]. We use average pooling to obtain the region descriptor.
Table 1. Average timing (ms) for computing all features for an image in CamVid.

<table>
<thead>
<tr>
<th>Darwin</th>
<th>Color+Loc.</th>
<th>Tex.</th>
<th>LBP</th>
<th>HOG</th>
<th>SIFT</th>
<th>Hyper.</th>
<th>Seg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2091</td>
<td>183</td>
<td>204</td>
<td>89</td>
<td>171</td>
<td>515</td>
<td>978</td>
<td>219</td>
</tr>
</tbody>
</table>

• Hyper-column features [9] are obtained from the fully convolutional network, fcn-8s [10]. Specifically, we apply pre-trained fcn model to images and extract the hyper-column feature on pool1, pool2, pool4 and fc7. Then we use maximal-pooling to extract superpixel features and apply PCA to reduce feature dimensions. Features are reduced to 900 and 500 for CamVid and SBG respectively. For CamVid and SBG, we use fcn-8s model pre-trained on PASCAL VOC. As for Siftflow, we use fcn-16s pre-trained on ImageNet and fine-tuned on the training set of Siftflow.

2.2 Cost Computation

We use feature computation and inference time as a surrogate for the cost in our experiments. For each action \( a_t \), we define the action cost \( c(a_t) = c_{ht} + c_{ft} \), or \( c_r \) (cf. Sec 3.3), where the individual costs are computed as follows:

- \( c_{ft} \) is the extraction time of any new group of features that have not previously been computed. We use GPU\(^1\) to extract hyper-column feature and it takes 0.067s on average. To have a fair comparison with other CPU-computed features, time cost for hyper-column feature is calibrated according to the power consumption ratio between CPU\(^2\) vs. GPU\(^3\). Time cost of applying PCA is also included.
- \( c_{ht} \) is the cost of applying weak learners, which starts from 0.0002s for a single weak learner and increases linearly with the number of weak learners \( m \).
- \( c_r \) is the time of generating finer segmentation and pooling features for newly split regions.

Table 1 shows the average timing for computing all features for an image in CamVid. We also include the segmentation tree generation time in the last column of Table 1. It takes about 0.17s to generation a 8-th layer hierarchical segmentation tree for an image in CamVid on average. We take it into account as an initial cost \( (t = 0) \) in our evaluation.

2.3 Details about Policy Meta-Features

We design three sets of model meta-features in \( \phi(s_t, a_t) \) for the approximate policy learning. As described in the paper, they are marginal distribution meta-features, active region meta-features and layer meta-features. The details of these meta-features are summarized as follows:

---

1. Using CPU implementation takes 28.41s on average, which is significantly higher than other features. As such, it would not be selected in our test-time budget range.

2. Intel i7 4930K [http://ark.intel.com/products/77780/Intel-Core-i7-4930K-Processor-12M-Cache-up-to-3_90-GHz](http://ark.intel.com/products/77780/Intel-Core-i7-4930K-Processor-12M-Cache-up-to-3_90-GHz)

• Marginal distribution meta-features are computed from the current label marginal distributions and image features on all regions: (M1) the average entropy of the label marginals; (M2) the average entropy gap between the previous marginals at \( t-1 \) and current marginals; (M3) a binary indicator vector to show what image feature sets have been used. We set its \( M \)-th value to 1 if the \( M \)-th feature set has been extracted in previous actions. \( M \in \{1, ..., 9\} \); (M4) a binary indicator vector to show which new image feature set is used. We set its \( M \)-th value to 1 if the \( M \)-th feature set is used in the current action but has not been extracted before; (M5) The mean, min and max of the difference between the highest and the second highest probability scores of each label marginal distributions.

• Active region meta-features are computed on active regions, including (A1) the normalized area of active regions in current image; (A2) the average entropy of label marginals on active regions; (A3) the gap between the average entropy of the previous and current label marginals in active regions.

• Layer meta-features consist of two parts: (L1) the layer-wise region counts in hierarchies normalized by the total number of regions; (L2) the layer-wise active region counts in hierarchies normalized by the total number of active regions.

We note that the meta-features depend on both state \( s_t \) and action \( a_t \). In particular, M1, M2, M3 and L1 depend on state \( s_t \) only while the rest are functions of \( a_t \) and \( s_t \). For instance,

• M4: If \( a_t = \alpha_t h_t \), we set its \( M \)-th value to 1 if the \( M \)-th feature set is included in \( h_t \) but has not been extracted before. If \( a_t \neq \alpha_t h_t \), we set all its values to zero.

• A1–A3, L2: If \( a_t = \theta_t \), the active regions are the newly generated \( Z^{t+1} \) after splitting. We use the \( Z^{t+1} \) to compute the meta features A1–A3 and L2. If \( a_t \neq \theta_t \), these meta-features are computed based on \( Z^t \).

2.4 Details about Subset Generation

To increase the diversity of discrete action candidates, we apply the action proposal generation procedure in Sec 4.1 on different subsets of images. The image subsets are formed by K-means [11] clustering w.r.t. following criteria.

• Image-based features. For each image in the training dataset, we use Gist [12] computed from 256 \( \times \) 256 rescaled images at 2 scales with 8 and 4 orientations. We then use K-mean method to group these features into 3, 5 and 7 groups.

• Reward-based features. Given the fixed sequence of actions \( \{a^0_t\}_{t=1}^T \) by maximizing immediate rewards on \( M \) images in training set, we can obtain their accumulated reward for each image \( m \) (as in Sec 4.1). We introduce a vector \( r_m \) to describe reward-based feature of the \( m \)-th image, in which the \( t \)-th value is the accumulated reward up to step \( t \). We then use K-mean method to group \( \{r_m\}_{m=1}^M \) features into 3, 5 and 7 groups.

We then generate one sequence of greedy actions (cf. Sec. 4.1) for each of the group. We then combine the generated sequences together to form \( \mathcal{A}^d \).
Data: Test image $I$, Policy $\eta$; Action space $A^d$

Result: Sequence of $s_t = \{B^t, Q^t, Z^t\}$

Initialize with $s_0$; $t = 0$

while Test-time budget is not reached do

Compute $\phi(s_t, a_t)$ for all $a_t \in A^d$

if $\max_{a_t} \eta^T \phi(s_t, a_t) \leq 0$ then

break;

else

Select $a_t = \arg \max_{a_t \in A^d} \eta^T \phi(s_t, a_t)$

Update $s_{t+1} = T(s_{t+1}|s_t, a_t)$

$t = t+1$

end if

end while

▷ Root of segmentation tree

▷ No more value improvement

▷ Select the best atom operator

▷ Update the hierarchical model

▷ Stop DHM and predict labels by $Q^t$

Fig. 1. Algorithm during test-time. Scene labeling is predicted by maximizing the marginal probabilities of all leaf nodes in $B^t$ at time $t$, $Q^t$.

2.5 Test procedure and Reward Function

We summarize the test algorithm for an individual test image after the policy is learned in Figure 1, which essentially generates a sequence of hierarchical models for label prediction.

On the left panel of Figure 2, we illustrate the intuition of our reward function. More specifically, we define the reward of individual action as the labeling loss improvement per unit cost. By maximizing the weighted average reward function, we can achieve good performance at any test-time budget.

In this work, we set discount factor $\gamma$ to 0.9 empirically. We also visualize the effectiveness of $\gamma$ on CamVid in the right panel of Figure 2, which corresponds to $\gamma = 0, 0.5, 0.9$. We can see that $\gamma = 0.9$ provides better overall anytime performance.
Fig. 3. Loss function trend (Left) and average IOU score (Right) as a function of cost on CamVid. Our D-NM achieves better trade-offs between efficiency and accuracy/labeling loss.

Table 2. Area under average accuracy v.s. time cost in CamVid. D-NM outperforms all methods consistently.

3 More Experimental Results

3.1 Additional Results on CamVid

We report the averaged IOU scores and labeling loss as a function of time on CamVid in Figure 3. The left panel of Figure 3 shows labeling loss trends with increasing cost. We also visualize some examples of our anytime output with specific actions in Figure 4.

We can see that D-NM has the lowest loss curve during most of time and its labeling loss decreases faster than static myopic settings in general. RS has the highest loss curve, which shows the importance and effectiveness of our policy learning. We note that F-SM achieves lower values at the beginning stage, as it works on the finest layer where the second term of our loss function is always at the minimum. However, F-SM gets saturated quickly and ends up with a similar lower score as the S-M.

To have a quantitative measurement, we compare the area under average accuracy v.s. time cost as proposed in [13] in the Table 2. We use the pixel-level accuracy as the measured accuracy. As shown in the table, D-NM outperforms all methods consistently.
3.2 Additional Algorithm Analysis

Ablation Study on Features. The image features we adopted in this work are commonly-used in scene labeling literature and have publicly available code. We have done an ablation study on CamVid by incrementally adding more feature types, and the results on 9 feature types used in this work are shown in Fig 5 (Left). We can see they are complementary to some degree and all contribute to the final performance.

Convergence of Q-learning. We design the reward using ground-truth guidance in each step (Eq.6), which is noise-free and reliable. Our Q learning is based on convergent Least-square policy iteration [26] and our stopping criterion is based on the accuracy on a validation set. Fig.5 (Right) shows the convergence of Q learning on Camvid.

Discretization of Split Threshold. We have explored finer quantizations on \( \Theta \) (e.g., \{0, 0.2, 0.4, 0.6, 0.8, 1\}) but empirically they do not improve the overall performance in our experiments.

References


