Latent Hierarchical Structural Learning for Object Detection

Long (Leo) Zhu[1] Yuanhao Chen[2] Alan Yuille [2] William Freeman[1]
[1]CSAIL, MIT [2]Department of Statistics, UCLA
{leozhu, billf }@csail.mit.edu {yhchen,yuille}@stat.ucla.edu

Abstract

We present a latent hierarchical structural learning method for object detection. An object is represented by a mixture of hierarchical tree models where the nodes represent object parts. The nodes can move spatially to allow both local and global shape deformations. The models can be trained discriminatively using latent structural SVM learning, where the latent variables are the node positions and the mixture component. But current learning methods are slow, due to the large number of parameters and latent variables, and have been restricted to hierarchies with two layers. In this paper we describe an incremental concave-convex procedure (iCCCP) which allows us to learn both two and three layer models efficiently. We show that iCCCP leads to a simple training algorithm which avoids complex multi-stage layer-wise training, careful part selection, and achieves good performance without requiring elaborate initialization. We perform object detection using our learnt models and obtain performance comparable with state-of-the-art methods when evaluated on challenging public PASCAL datasets. We demonstrate the advantages of three layer hierarchies – outperforming Felzenszwalb et al.'s two layer models on all 20 classes. SHOULD WE MENTION OUR INFERENCE TIME?? 8 SECONDS IS GOOD!!

1. Introduction

Object detection is an important task in computer vision which has made great use of learning [**?**]. Recent progress includes: (i) learning part-based models [**?**, **?**, **?**, **?**], (ii) learning appearance features [**?**], and (iii) learning context [**?**]. Other recent representative work includes [**?**].

This paper focusses on learning hierarchical models with deep structure. The success of 'shallow structures' with two layers [**?**] suggests that we can make further progress by extending to 'deep structures' which should give richer descriptions of shape and appearance. But this extension is not straightforward for the following two issues:

1. What are good part structures? Felzenszwalb *et al.*[**?**] describe a layer-wise training procedure which requires careful part selections and model initializations. This multi-stage training method is effective for shallow (2 layer) structures, but it seems hard to scale up to more layers without tuning the parameters, such as the number of parts at the deeper layers.

2. How to learn a deep structure efficiently? A deep structure has many more features and latent variables than a shallow structure. This means that we have to learn more parameters which requires more training data. The training time also increases due to the amount of training data and the greater amount of computations required (e.g., we need to compute inner products for more features). Therefore, learning a deep structure becomes more computationally challenging.

In this paper we show that simple part structures are sufficient to obtain state of the art performance. We will ar-

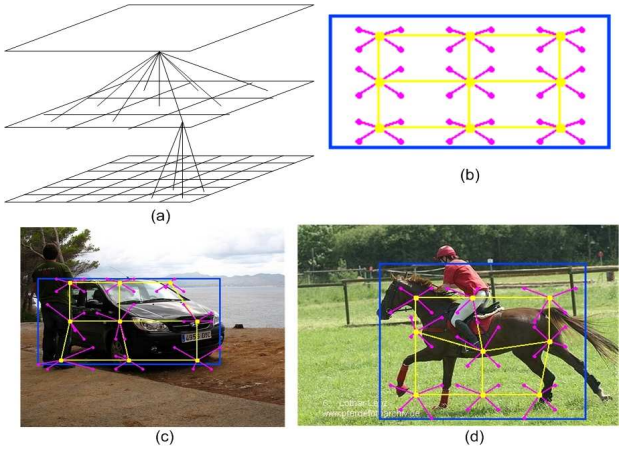

Figure 1. (a) A 3-layer tree model. The structure has three layers with nodes in simple grid layouts, i.e. 1×1, 3×3 and 6×6. (b) A reference template without part displacement (no deformation). Blue rectangle is the bounding box of the root node. Yellow dots indicate the center positions of 9 parts at the 2nd layer. Purple dots are the centers of 36 parts at the bottom layer. (c,d) examples of part displacement. Yellow grids connecting the 9 intermediate nodes show the deformation at the 2nd layer. Purple lines which connects the four child nodes to their parent node at the 2nd layer show the local deformation at the bottom layer.

gue, respectfully disagreeing with [**?**], that the choice of part structure does not significantly affect the performance although it may affect the convergence rate of learning. In fact our model is a simple extension of the 2-layer model used in [**?**]. In this paper, an object class consists of several prototype templates from different viewpoints each of which is represented by a 3-layer tree-structure model. The structure of the model is shown in figure 1. The first layer has one root node which represent the entire object. The root node has 9 child nodes at the second layer in a 3 by 3 grid layout. Each node at the second layer has 4 child nodes at the third layer which contains 36 nodes in a 6 by 6 grid layout. The numbers of layers and nodes are the same for different object classes and views. We will show that this simple structure leads to a meaningful model with good performance.

We can train deep structure models efficiently by modifying a recent approach to latent structural SVM learning [**?**]. This requires minimizing an objective function which is non-convex, but which can be expressed as the difference of two convex functions. Yu and Joachims apply the the concave-convex procedure (CCCP) [**?**] to this objective function to derive an algorithm that is guaranteed to converge to a local minima. The algorithm proceeds in two alternating steps, analogous to the EM algorithm): (1) estimate the latent variables using the current estimates of the model parameters. (2) estimate the model parameters using standard structural SVM learning (treating the estimated latent variables as groundtruth). But this method is not efficient enough for deep structures because of the large number of parameters and training data. Hence we developed an incremental concave-convex procedure (iCCCP) which sequentially adds training data at each iteration. This greatly reduces the training cost and enables us to learn deep structures efficiently. Overall, iCCCP is a simple training algorithm which learns multi-layer parameters simultaneously, avoiding complicated multi-stage layer-wise training, and does not require elaborate initialization.

In summary, this paper draws three conclusions: (1) Deep structures are better than shallow structures (3-layers outperform 2-layers on all 20 object classes), (2) Simple hierarchical structure are able to achieve good performance. (3) iCCCP learning is simple and efficient and enables us to learn multi-layer parameters simultaneously.

2. Background: Structural SVM and Latent Variables

Suppose we are given a set of training samples (x_1, y_1, h_1),...,$(x_N, y_N, h_N) \in \mathcal{X} \times \mathcal{Y} \times \mathcal{H}$ where x is an image patch, y is a label of object class. $h = (V, \vec{p})$ where V is a label of viewpoint, and $\vec{p}$ is the position of object parts. The task of structural SVM learning is to learn a discriminative function of the form:

$$F_w(x) = \underset{y,h}{\operatorname{argmax}}[w \cdot \Phi(x, y, h)] \tag{1}$$

where Φ is a joint feature vector that describes the relationship between input x and structured output (y, h), with w being the parameter vector. Φ have two forms: (i) Appearance features $\Phi_A(x, y, h)$ connect image features x to object classes y, viewpoints V and object parts $\vec{p}$. (ii) Shape features $\Phi_S(y, h)$, which are not related to x, capture the shape deformation of object parts. The optimization problem of computing this argmax is typically referred to as the "inference" problem.

The standard structural SVM problem assumes that the structure of h is given and fixed, i.e. the number of parts are known. To train structural SVMs we solve the following convex optimization problem [**?**]:

$$\min_w \frac{1}{2}||w||^2 + C\sum_{i=1}^{N}\left[\max_{y,h}[w \cdot \Phi_{i,y,h} + L_{i,y,h}] - w \cdot \Phi_{i,y_i,h_i}\right] \tag{2}$$

where C is a fixed penalty parameter, $\Phi_{i,y,h} = \Phi(x_i, y, h)$ and $L_{i,y,h} = L(y_i, y, h)$ is the loss function. For object detection problem, $L(y_i, y, h) = 1$ if $y_i = y$, 0 if $y_i \neq y$. This optimization problem can be solved efficiently using cutting-plane method [**?**]. A recent successful application to computer vision is human body parsing [**?**].

If h is not labeled in the training set, then we need to solve a latent structural SVM problem:

$$\min_w \frac{1}{2}||w||^2 + C\sum_{i=1}^{N}\left[\max_{y,h}[w \cdot \Phi_{i,y,h} + L_{i,y,h}] - \max_{h}[w \cdot \Phi_{i,y_i,h}]\right] \tag{3}$$

This optimization problem is non-convex. Yu and Joachims [**?**] offered a general solution to finding a local optimum using the CCCP concave-convex Procedure [**?**]. We note that [**?**] explored an alternative approach to this problem by transforming the structural learning problem into a standard binary SVM learning problem. In our implementation, we follow Yu and Joachims's strategy but modify CCCP to iCCCP.

3. Latent Hierarchical Structural Learning

3.1. Hierarchical Structure, Latent Variables and Features

An object class consists of two prototype templates from two different views each of which is represented by a 3-layer tree-structure model. The structure of the model is

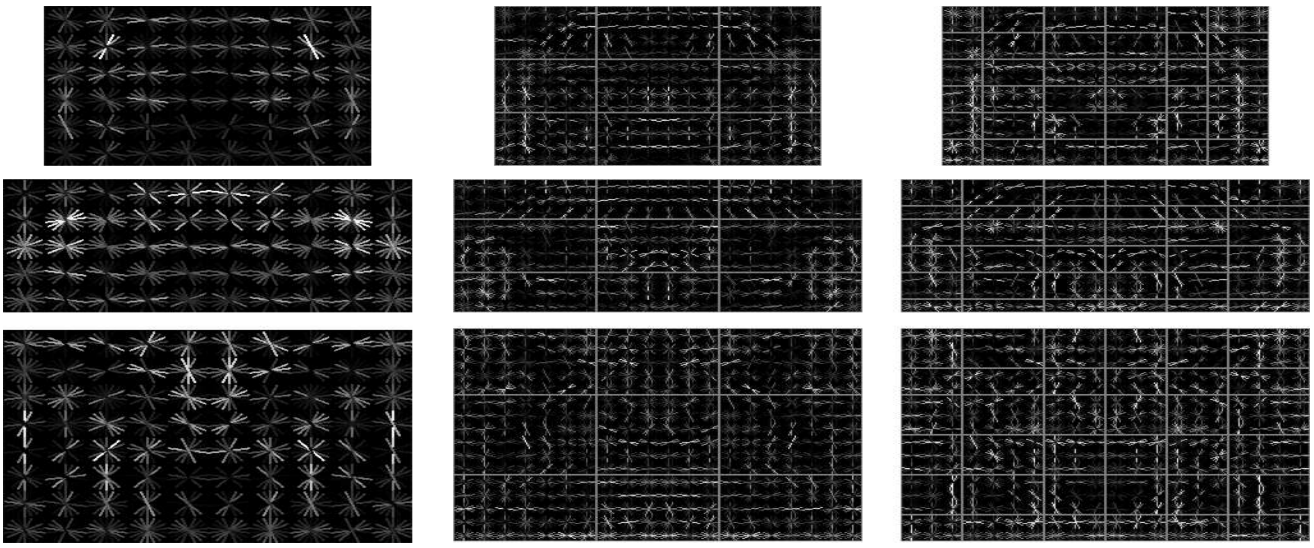

Figure 2. Some models (appearance only) learnt from the PASCAL 2007 dataset. The first 2 rows are the car models from two views and the last row is a horse model. Three columns show the weights in each orientation of the HOG cells at the 1st (1×1 grid), 2nd (3×3 grid) and 3rd (6×6 grid) layers, respectively. Each cell consists of 8×8 pixels. The models look semantically meaningful. The weights along the object boundary are high. The features at different layers capture object appearance in a coarse-to-fine way. The features at lower levels capture more detailed appearance (e.g. the horse legs at the 3rd layer look brighter.)

shown in figure 1. The first layer has one root node which represent the entire object. The root node has 9 child nodes at the second layer in a 3 by 3 grid layout each of which represents one ninth of an object. Each node at the second layer has 4 child nodes at the third layer which contains 36 nodes in a 6 by 6 grid layout. There are 46 $(1+3\times3+6\times6)$ nodes in total. The numbers of layers and nodes are the same for different object classes and views. But their aspect ratios may be different. Each tree model is associated with latent variables $h = (V, \vec{p})$. V is the index of viewpoint and $\vec{p} = ((u_1, v_1), (u_2, v_2), ..., (u_{46}, v_{46}))$ encode the positions of all nodes. For an object class, let $y = +1$ denote object and $y = -1$ denote non-object. Let $a = 1..46$ be the index of nodes. $b \in Ch(a)$ indexes the child nodes of node a. The feature vector is defined as follows:

$$\Phi(x, y, h) = \begin{cases} (\Phi_A(x, h), \Phi_S(h)) & \text{if } y = +1 \\ 0 & \text{if } y = -1 \end{cases} \quad (4)$$

The $\Phi_A(x, h)$ are the appearance features which contain HOG [**?**] descriptors $\Phi_A(x, \vec{p}_a)$ for all nodes. We followed the implementations of [**?**] to calculate HOG descriptors. Figure 2 shows the weights of HOG descriptors at different layers. The image patch corresponding to the entire object is represented by $W \times H$ cells each of which are 8×8 pixels. The HOG features for each cell represent the local gradient information which consists of 31 features including 9 contrast sensitive features, 18 insensitive features and 4 sums over the 9 contrast insensitive orientations. The length of the features $\Phi_A(x, \vec{p}_1)$ for the root node is $W \times H \times 31$. The 2nd layer has $2W \times 2H$ cells with two times resolution. The length of $\Phi_A(x, \vec{p}_a)$ for each node at this layer is $\frac{2}{3}W \times \frac{2}{3}H \times 31$. The bottom layer has the same resolution as the 2nd layer. The $\Phi_A(x, \vec{p}_a)$ for the 3rd layer is of length $\frac{2}{6}W \times \frac{2}{6}H \times 31$. There are $9 \times W \times H \times 31$ features for all 46 nodes. For a typical size with $W = 10, H = 5$ (see row 2 in figure 2), our 3-layer model has 13,950 HOG features in total.

The $\Phi_S(h)$ are shape features $\Phi_S(\vec{p}_a, \vec{p}_b), \forall a, b \in Ch(a)$, which encode the parent-child pairwise spacial relationship. More precisely, the shape features for a parent-child pair (a, b) are defined as $\Phi_S(\vec{p}_a, \vec{p}_b) = (\Delta u, \Delta v, \Delta u^2, \Delta v^2)$ where $(\Delta u, \Delta v)$ is the displacement of node b relative to its reference position which is determined by the position of the parent node a. Our 3-layer model has 180 $(4 \times 9 + 4 \times 36)$ shape features in total.

Obviously, our 3-layer structure is deeper than the shallow structure used in [**?**] while the HOG and shape features are the same as [**?**]. Unlike [**?**], the topology (part organization) of our model is predefined by hand to be identical for all object classes and views. We will show that this arbitrary (trivial) design does not affect the performance of the models while its simplicity avoids the need to carefully initialize the model structures.

3.2. Detection: Dynamic Programming

Suppose the parameters w are given. Given an image patch (subwindow), the detection task is to find a class label and part locations (y^*, h^*) with the best score: $(y^*, h^*) = \text{argmax}_{y,h}[w \cdot \Phi(x, y, h)]$. The models for two views $V = 1, 2$ are independently evaluated. To find a best

$\vec{p}$, the position $\vec{p}_1 = (u_1, v_1)$ of the root is first located by scanning all subwindows at different scales of the image pyramid. Given a location (u_1, v_1) of the root node, the best configuration $\vec{p}$ of the remaining 45 parts is obtained by dynamic programming which is a recursive procedure:

$$F(x, \vec{p}_a) = \sum_{b \in Ch(a)} \max_{\vec{p}_b} \{F(x, \vec{p}_b) + w \cdot \Phi_S(\vec{p}_a, \vec{p}_b)\} + w \cdot \Phi_A(x, \vec{p}_a) \quad (5)$$

where $F(x, \vec{p}_a)$ is the max score of a subtree with node a being the root. The boundary condition is $F(x, \vec{p}_a) = \Phi_A(x, \vec{p}_a)$ if a is a leaf node. This recursive form is equivalent to the discriminative function in equation 1, i.e. given a view V and $y = +1$, the score of the root $F(x, \vec{p}_1) = \max_{\vec{p}} w \cdot \Phi(x, \vec{p})$.

The bounding box determined at position (u_1, v_1) of the root node and the corresponding level of the image pyramid is output as an object detection if the score $F(x, \vec{p}_1) > 0$ (note $F(x, y) = 0$ if $y = -1$).

3.3. Optimization: standard CCCP

Learning the latent hierarchical model can be formulated as a learning problem defined in equation (3). This learning problem can be written equivalently as [**?**]:

$$\min_w \left[\frac{1}{2}||w||^2 + C \sum_{i=1}^{N} \max_{y,h} [w \cdot \Phi_{i,y,h} + L_{i,y,h}] \right] \quad (6)$$

$$- \left[C \sum_{i=1}^{N} \max_{h} [w \cdot \Phi_{i,y_i,h}] \right] \quad (7)$$

$$= \min_w \{f(w) - g(w)\} \quad (8)$$

where $f(w)$ are the first two terms (**??**) and $g(w)$ is minus the last term (7). Note f and g are both convex, but $f(w) - g(w)$ is not. The Concave-Convex Procedure (CCCP) gives an algorithm which alternates the following two steps (see the pseudo code in figure 3):

Step (1): find hyperplane q_t such that $-g(w) \leq -g(w_t) + (w - w_t) \cdot q_t, \forall w$

Step (2): Solve $w_{t+1} = \text{argmax}_w [f(w) + w \cdot q_t]$

The CCCP algorithm is guaranteed to converge to a local minimum. Step 1 constructs a hyperplane that upper bounds the concave part of the objective $-g$, so that the optimization problem solved at step 2 is convex.

Step (1) is performed by finding the best h^*: $h_i^* = \text{argmin}_h \, w_t \cdot \Phi_{i,y_i,h}$. Then the hyperplane constructed is $q_t = -C \sum_{i=1}^{N} \Phi_{i,y_i,h_i^*}$. Step (2) is to solve $\min_w [f(w) - C \sum_{i=1}^{N} w \cdot \Phi_{i,y_i,h_i^*}]$. Substituting $f(w)$ with the first two terms in equation (7) enables us to rewrite this as:

$$\min_w \frac{1}{2}||w||^2 + C \sum_{i=1}^{N} \left[\max_{y,h} [w \cdot \Phi_{i,y,h} + L_{i,y,h}] - w \cdot \Phi_{i,y_i,h_i^*} \right] \quad (9)$$

This is a standard structural svm problem without latent variables. The solution to this minimization can be found by differentiation and expressed in form:

$$w^* = C \sum_{i,y,h} \alpha_{i,y,h}^* \Delta\Phi_{i,y,h} \quad (10)$$

where $\Delta\Phi_{i,y,h} = \Phi_{i,y_i,h_i^*} - \Phi_{i,y,h}$ and the α^* are obtained by maximizing the corresponding dual function:

$$\max_\alpha \sum_{i,y,h} \alpha_{i,y,h} L_{i,y,h} - \frac{1}{2} C \sum_{i,j} \sum_{y,h,y',h'} \alpha_{i,y,h} \alpha_{j,y',h'} \Delta\Phi_{i,y,h} \cdot \Delta\Phi_{j,y',h'} \quad (11)$$

This is a standard structural SVM dual problem. We use the cutting plane method [**?**, **?**] to optimize the objective function in equation (11). The method creates a working set sequentially and then estimate the parameter α on the working set. More precisely, it seeks to create a working set for the first round training. We check all training examples (x_i, y_i, h_i). If $w \cdot \Phi_{i,y^*,h^*} + L_{i,y^*,h^*} - max_{y',h'}[w \cdot \Phi_{i,y',h'} + L_{i,y',h'}] > \delta$ where $(y^*, h^*) = \text{argmax}_{y,h}[w \cdot \Phi_{i,y,h} + L_{i,y,h}]$, (x_i, y', h') are examples already in the working set, and δ is a tolerance parameter, the working set is constructed by adding examples (x_i, y^*, h^*). The working set procedure is similar to the hard examples data mining used in [**?**]. It is easy to scan all positive training examples with bounding box groundtruth which limits the search space of $\vec{p}$ in dynamic programming. But scanning all subwindows in a large set of negative training images is very expensive. In the next subsection, we will introduce a new algorithm to make large-scale training more efficient.

When the working set has enough training samples (when we reach a a fixed memory limit), we estimate the parameters α on the working set. The optimization over the working set is performed by Sequential Minimal Optimization [**?**]. Then the algorithm proceeds to the next round, expands the working set and performs optimization over the new working set. The optimal solution of w_t at iteration t is obtained after several rounds of working set construction and training.

The solution w_t is used as a starting point in a new CCCP iteration, without having to reconstruct all the working set from scratch. After the CCCP algorithm converges i.e. $[f(w_t) - g(w_t)] - [f(w_{t-1}) - g(w_{t-1})] < \epsilon$ where ϵ

Instantiate $t = 0$.
Repeat $t = t + 1$;

1. Fill in latent variables:
   $h_i^* = \text{argmin}_h\, w_t \cdot \Phi_{i,y_i,h}$
2. Solve the structural SVM problem (given h, estimate w): $w_{t+1} = \text{argmin}_w\, f(w) - C\sum_i w \cdot \Phi_{i,y_i,h_i^*}$

Until $[f(w_t) - g(w_t)] - [f(w_{t-1}) - g(w_{t-1})] < \epsilon$.

Figure 3. Standard CCCP algorithm.

is a threshold set by hand, we get the local minimum at w^*, given in equation 10.

3.4. Incremental CCCP for Large Scale Training

Instantiate: $t = 0$; $S = \{x_i, y_i\}^+ \cup \{x_j, y_j\}^-, i = 1..N^+, j = 1..n$
Repeat $t = t + 1$

1. Fill in latent variables $(x_i, y_i) \in S$:
   $h_i^* = \text{argmin}_h\, w_t \cdot \Phi_{i,y_i,h}$
2. Solve the structural SVM problem over S (given h, estimate w):
   $w_{t+1} = \text{argmin}_w\, f(w) - C\sum_i w \cdot \Phi_{i,y_i,h_i^*}$
3. $S = S \cup \{x_j, y_j\}^-, j = nK^{t-1} + 1, nK^{t-1} + 2, ..., n \times K^t$

Until $[f(w_t) - g(w_t)] - [f(w_{t-1}) - g(w_{t-1})] < \epsilon$.

Figure 4. Incremental CCCP (iCCCP) algorithm. $\{x_i, y_i\}^+$ and $\{x_i, y_i\}^+$ refer to the positive and negative training set, respectively. N^+ is the size of the positive set.

In standard CCCP for latent structural SVM, a lot of computation cost is spent at step 2 essentially finding hard – WHAT IS "HARD"?? IS THIS DETERMINED BY AUTOMATICALLY BY THE ALGORITHMS??

training examples by scanning all subwindows from a large set of negative training images (e.g., 100 images which take 14 minutes to scan all subwindows in one round at 8 seconds per image). We propose an incremental CCCP (iCCCP) algorithm to handle this issue. See the pseudo code in figure 4. It is motivated by realizing that it is not necessary to get optimal solutions at the early stage of CCCP learning. iCCCP starts from a small number $n = 30$ of negative images, learns w given the hard negative examples selected from 30 images WHO SELECTS THESE EXAMPLES?? IT SOUNDS LIKE THIS IS DONE BY YOU AND CHEN!! IS THIS RIGHT?? and proceeds to update w by incrementally adding more negative images into the training set. The scaling factor K of new negative images is 1.15: iCCCP examines 30 negative images at the 1st iteration, 30×1.15 at the 2nd iteration, ..., and so on, until it converges. iCCCP is able to achieve similar accuracy with greatly reduced computational cost. For example, assuming that standard CCCP needs 10 iterations in which CCCP scans 100 negative images 10 times using 270 (10×27) minutes, iCCCP reduces the cost by a factor of 6.54 (6.54=$21.8 \times 30/100$; $21.8 = 1 + 1.15 + 1.15^2 + ... + 1.15^9$). We will empirically compare the performance of iCCCP with standard CCCP in section 4.4.

Note that for a 3-layer model, learning parameters w with a length of 14,130 (13950+180) takes around two times greater cost than learning a 2-layer model to calculate the inner product HOW MANY PARAMETERS DOES THE 2-LAYER MODEL HAVE??. More parameters need more training data (and time) to converge. The advantage of iCCCP over standard CCCP appears to be more critical while learning deep structures.

3.5. Implementation Details

The training-irrelevant implementations are identical to [**?**]. The HOG features and shape features are the same. All object classes are represented by two predefined templates. The aspect ratios of the two templates are determined by selecting two representative aspect ratios and sizes by counting their histograms in the groundtruth. The weights w are ensured to be symmetry – WHAT!! DO YOU MEAN SYMMETRIC?? WHY?? . Since we focus on the comparisons of models with different part structures, we do not use any post-processing, such as bounding box prediction, rescoring the classifiers using contextual information, etc. Our framework differs from [**?**] in the learning algorithm:

Simple Initialization and Simultaneous Multi-Layer Learning: [**?**] has a very complicated 3-stage layer-wise training procedure which initializes appearance weights in a coarse-to-fine way. They implemented a gradient decent algorithm to learn the weights w, which requires careful initialization. Instead, in this paper, learning a hierarchical model is performed by the iCCCP algorithm which does not need a multi-stage training procedure and does not require weights initializations. iCCCP initializes the positions $\vec{p}$ of all nodes at three layers by setting them in a regular grid layout without displacements $\Delta u = 0, \Delta v = 0$, and then learn the weights w of all nodes at different layers simultaneously. This simple design does not require careful initializations and pre-training of structures. In section 4.4, we will empirically show that the part structures and the initiations do not prevent iCCCP from learning a good model.

Bias Terms in Structural Learning: The discriminative function $F_w(x) = \text{argmax}_{y,h}[w \cdot \Phi(x, y, h)]$ does not include the bias terms which appear in binary svm learning. We introduce bias terms $b_{y,v}$ to obtain $w \cdot \Phi(x, y, h) + b_{y,v}$ by attaching extra constant terms $\Phi_b(y, v) = 10$ for all possible labels of (y, v), following [**?**]. The corresponding weights w_b are then $b_{y,v}/10$. These adjustments help improve the performance. For more details, see [**?**]. ([**?**] avoids this issue by transforming the structural learning

problem into a binary svm learning problem.)

Cutting Plane Method: We use the cutting plane method to learn w. At each round, this method proceeds by sequentially adding positive training examples together with 500 negative examples, which violate the KKT conditions, into a working set. The tolerance parameter is $\delta = 10^{-6}$. After several rounds, the number of hard negative examples decreases exponentially.

Updating Latent Variables: At step (1) of the iCCCP iterations, the states of latent variables (i.e. the positions of all nodes) are restricted to ensure that the box of the root overlaps with the groundtruth bounding box by at least 70%. The positions of the child nodes are restricted to ensure the child node overlaps with the corresponding reference box.

4. Experimental Evaluations

The PASCAL VOC 2007 dataset [**?**] was used for evaluations. It is the latest version that test annotations are available. There are 20 object classes which consist of 10000 images for training and testing. We follow the experimental protocols and evaluation criteria used in the PASCAL Visual Object Category detection contest 2007. A detection is considered correct if the intersection of its bounding box with the groundtruth bounding box is greater than 50% of their union. We compute Precision-Recall (PR) curves and score the average precision (AP) across a test set. All experiments are performed on a standard computer with a 3Ghz CPU. C is set to 0.005 for all classes. The detection time per image is 8 seconds. The starting number of negative images used in the iCCCP training is $n = 30$. The increasing rate K is 1.15. It takes 25 hours (about 25 iCCCP iterations) to train an object class with two mixture templates.

4.1. The detection results on the PASCAL dataset

Some detection results with a bounding box and the positions of 45 nodes are shown in figure 5. The models learnt on the car and horse datasets are shown in figure 2. We compared our approach with other representative methods reported in the PASCAL VOC detection contest 2007 [**?**]. The comparisons in table 1 show that our 3-layer model outperforms other methods in 13 categories and [**?**] on all 20 classes. The average APs per category are 0.296(us), 0.268 (UoCTTI[**?**]), 0.271 (UCI[**?**]), 0.275([**?**]) and 0.321 [**?**]. [**?**] and [**?**] are not listed in table 1 because both methods rely on training multiple models. Our method is better than [**?**] which seeks to rescore the detection hypotheses output by [**?**]. They make use of more features and more complex training algorithms. [**?**] is the only one that outperforms our method by using more feature kernels. However, [**?**] runs much slower (it takes 67 seconds to calculate complex image feature kernels).

Datasets	UoCTTI	L=2,Parts=9	L=2, Parts=36	L=3
Car	.470	.501	.491	.513
Horse	.436	.443	.443	.504

Table 2. Comparisons of models with 2-layer and 3-layer structures on the cat and horse datasets. "UoCTTI" reports the results from [**?**]. "L=2,L=3" are our models with top 2 layers (9 parts and 36 parts at the second layer) and complete 3 layers which are trained by iCCCP algorithm.

4.2. Deep is better than shallow

In table 1, we have shown that our 3-layer structure is better than the 2-layer model (UoCTTI) used in [**?**] on all object classes. In order to study how much gain is obtained by deep structures, besides the 3-layer model (L=3) introduced before, we also implemented three other 2-layer models: our model with 9 parts (3×3) and 36 parts (6×6) at the second layer, respectively, and the other 2-layer model (UoCTTI) with 6 selected parts [**?**]. The comparisons of these four models with different layers and parts are performed on two object classes, cars and horses. Table 2 shows the average precisions of these four models. On the car dataset, our 2-layer models (L=2) outperforms UoCTTI by 0.031 and 0.021, respectively. The 3-layer model (L=3) further improves the performance slightly by 0.012. On the horse dataset, the 3-layer model easily outperforms other three 2-layer models which perform similarly to each other. By comparing the two models with 9 parts and 36 parts, it is clear that simply adding more parameters/parts gives little improvement in performance. But better performance is obtained by adding the third level to the hierarchy. In conclusion, deep structures are better than shallow structures.

4.3. Part structures and Initializations

We also investigated the effect of different part structures and initializations. Three models are compared: 1) the UoCTTI model (M=UoCTTI) using their initializations ($\vec{p}$=UoCTTI); 2) the UoCTTI model using our trivial initializations ($\vec{p}$=0 means the displacements of all nodes are zeros); 3) our 2-layer model (9 parts, M=O) with trivial initializations ($\vec{p}$=0). Two datasets (car and horse) are evaluated. All models are learnt by the iCCCP algorithm. The APs at different iterations of iCCCP are plotted in figure 6. The model (M=UoCTTI, $\vec{p}$=UoCTTI) achieve better performance than the other two models in the early iterations, but they all converge eventually to similar performance. The two models (M=UoCTTI and M=O) with same trivial initializations ($\vec{p}$=0) start from same position. This shows that different structures do not affect the performance when iCCCP has converged, but they may have different performance if iCCCP has not performed enough iterations. The results also suggest that the convergence of the iCCCP algorithm is not very sensitive to the initialization.

class	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
us	**.294**	.558	.094	**.143**	.286	**.440**	**.513**	.213	**.200**	**.193**	**.252**	.125	**.504**	.384	**.366**	.151	**.197**	**.251**	**.368**	**.393**
UoCTTI	.290	.546	.006	.134	.262	.394	.464	.161	.163	.165	.245	.050	.436	.378	.350	.088	.173	.216	.340	.390
UCI	.288	**.562**	.032	.142	**.294**	.387	.487	.124	.160	.177	.240	.117	.450	**.394**	.355	**.152**	.161	.201	.342	.354
V07	.262	.409	**.098**	.094	.214	.393	.432	**.240**	.128	.140	.098	**.162**	.335	.375	.221	.120	.175	.147	.334	.289

Table 1. Performance Comparisons on the 20 PASCAL VOC 2007 challenge categories [**?**]. (us) refers to our 3-layer model. (UoCTTI) reports the results from [**?**] without special post-processing. (UCI) [**?**] is a method using multi-object relationship. (V07) is the best result for each category among all methods submitted to the VOC 2007 challenge. Our method outperforms the other methods in 13 categories. Our 3-layer model is better than UoCTTI's 2-layer model in all 20 categories. The average APs per category are 0.296(us), 0.268 (UoCTTI) and 0.271 (UCI).

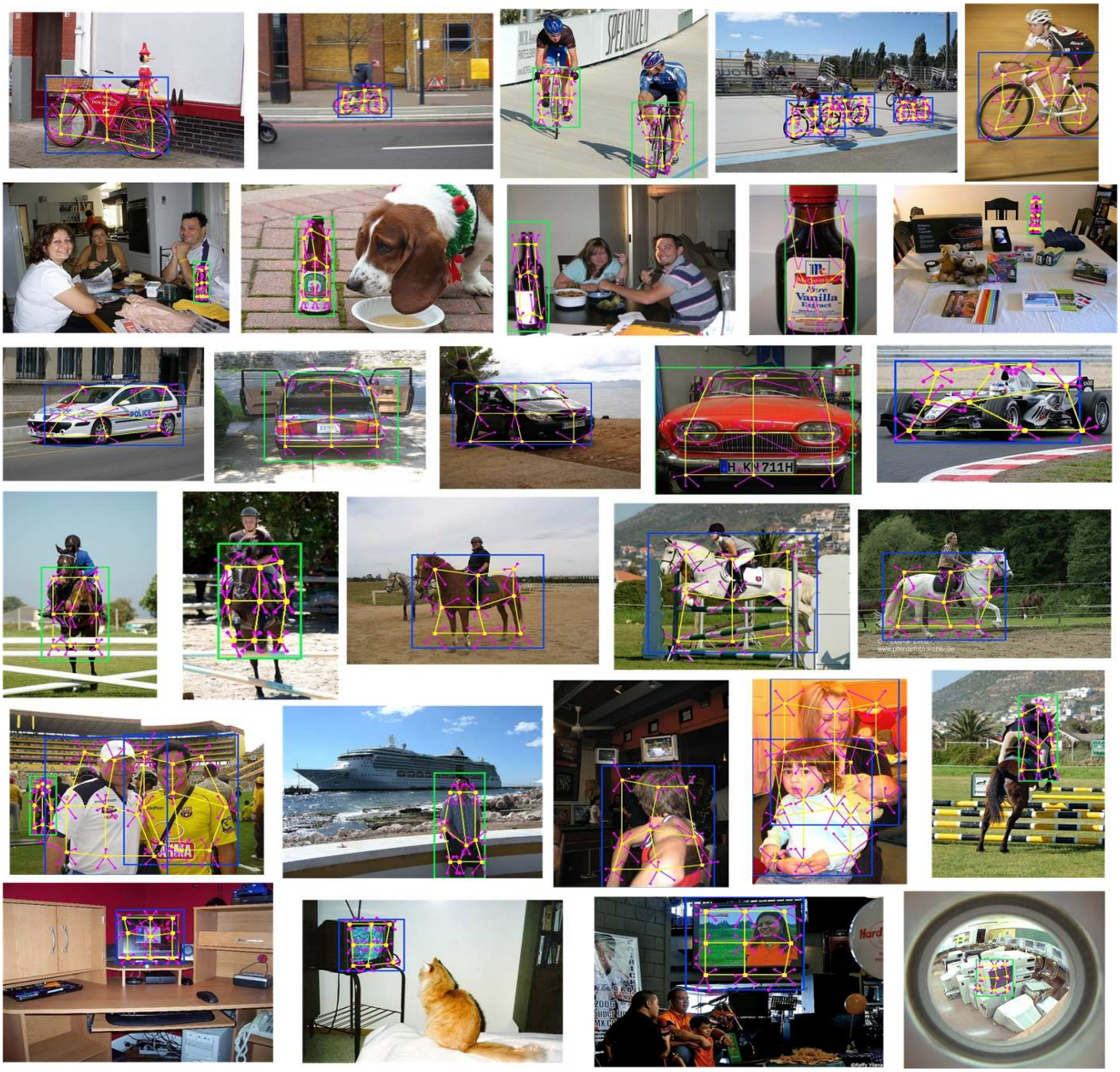

Figure 5. Some detection results from the PASCAL 2007 dataset. Each row contains several results of one class. Big rectangles are the bounding boxes of the root nodes. Blue and green indicate two different views. Nine yellow dots are the centers of nodes at the 2nd layer. Purple lines connect the parent-child pairs of nodes at the 2nd and 3rd layers. The yellow grids and purple lines show the deformations.

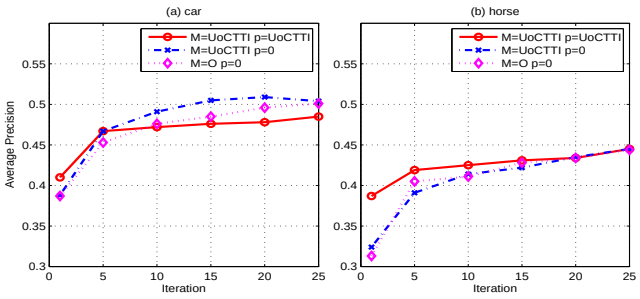

Figure 6. Comparisons of different part structures and initializations. "M=UoCTTI" refers to [**?**]. "M=O" is our model with top two layers. "p=UoCTTI"is using the initializations provided by the pre-trained one-layer model in [**?**]. "p=0" means that the displacements of all nodes are zeros. All models are learnt by iCCCP algorithms.

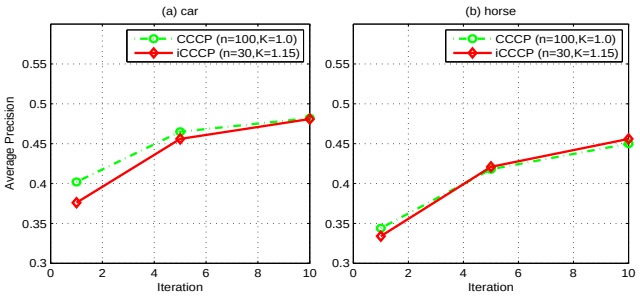

Figure 7. Standard CCCP vs. iCCCP. Standard CCCP uses 100 negative images at each iteration. iCCCP starts from using 30 images and sequentially increases the set with a scaling factor of 1.15.

4.4. Standard CCCP vs. iCCCP

Figure 7 compares the behaviors of standard CCCP and iCCCP. Standard CCCP uses $n = 100$ negative images at each iteration while iCCCP starts from using $n = 30$ negative images, incrementally adds extra 15% images at each iteration and uses 105 ($= 30 \times 1.15^9$) images at the 10th iteration. CCCP achieves better APs at the beginning because of accessing larger dataset, but CCCP and iCCCP converge to similar positions while iCCCP greatly reduces the computational cost as we discussed in section 3.4.

5. Conclusion

This paper describes a latent hierarchical structural learning method for object detection. We represent objects by mixture of hierarchical models with two or three layers. We developed a simple incremental convex-concave procedure (iCCCP) which is capable of learning deep three-layer models efficiently without multi-stage layer-wise training or elaborate initializations. The resulting object models are comparable with state-of-the-art methods on the PASCAL datasets. In particular, we show that models with deep structure outperform shallow structures and that simpler part structures are sufficient to obtain strong results.

Acknowledgments. Funding for this work was provided by NGA NEGI-1582-04-0004, MURI Grant N00014-06-1-0734, AFOSR FA9550-08-1-0489, NSF IIS-0917141 and gifts from Microsoft, Google and Adobe. Thanks to the anonymous reviewers for helpful feedback.