Neural Networks 92 (2017) 39–46

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

2017 Special Issue

Application of structured support vector machine backpropagation to a convolutional neural network for human pose estimation

Peerajak Witoonchart *, Prabhas Chongstitvatana *

Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University, 17th floor, Engineering 4 Building (Charoenvidsavakham), Phayathai Road, Wang Mai, Pathumwan, Bangkok 10330, Thailand

ARTICLE INFO

Article history:
Available online 16 February 2017

Keywords:
Back propagation
Convolutional neural network
Deformable part model
Human pose estimation
Structured support vector machine

ABSTRACT

In this study, for the first time, we show how to formulate a structured support vector machine (SSVM) as two layers in a convolutional neural network, where the top layer is a loss augmented inference layer and the bottom layer is the normal convolutional layer. We show that a deformable part model can be learned with the proposed structured SVM neural network by backpropagating the error of the deformable part model to the convolutional neural network. The forward propagation calculates the loss augmented inference and the backpropagation calculates the gradient from the loss augmented inference layer to the convolutional layer. Thus, we obtain a new type of convolutional neural network called an Structured SVM convolutional neural network, which we applied to the human pose estimation problem. This new neural network can be used as the final layers in deep learning. Our method jointly learns the structural model parameters and the appearance model parameters. We implemented our method as a new layer in the existing Caffe library.

1. Introduction

Generic structural prediction for object parts is similar to the human pose estimation (HPE) problem, e.g., labeling and bounding car parts with boxes, labeling and bounding face parts with boxes, and labeling and bounding bus parts with boxes. The pose estimation problem based on a still two-dimensional (2D) image is defined as finding the human joints or parts in an image containing one human. This problem is difficult due to variations in the color of clothes and because some parts are partially or totally occluded. Previous state-of-the-art pose estimation solutions are based mainly on the success of the pictorial structure programming (Felzenszwalb & Huttenlocher, 2005), which was developed by Felzenszwalb (Felzenszwalb & Huttenlocher, 2005). This method rapidly became the standard approach for object localization. Ramanan (2007) adopted this method for HPE and it is now the standard method for HPE. Ramanan's method for solving the HPE involves clustering the subparts, before extracting the histogram of oriented gradients (HOG) features for each subpart and learning support vector machine (SVM) filters for each of the subparts. Pictorial structures are then created and the parts are populated, before the subparts are filtered with these SVM filters to relearn them structurally. This method has been improved in various ways, where one way involves finding a better structure, e.g., Tian, Zitnick, and Narasimhan (2012) proposed a pictorial structure tree model with added latent variables, Tian and Sclaroff (2010) carefully designed leaf node variations and latent nodes, which control the variations of the leaf nodes, while Pishchulin, Andriluka, Gehler, and Schiele (2013) added a loopy model for inference. Dantone, Gall, Leistner, and Van Gool (2014) focused on clustering parts into multimodal decomposable models. Cherian, Mairal, Alahari, and Schmid (2014) tried to improve the pictorial structure by obtaining a better prior model by parameterizing the geometric variables. However, if the model is improved, all of these methods must learn the structural model parameters. Latent SVM (Yang & Ramanan, 2011) is the standard method for learning these model parameters.

Currently, deep learning and feature learning are popular methods for finding features for classification, detection, and segmentation, including a deep convolutional network for facial point detection (Sun, Wang, & Tang, 2013), a deep network for pedestrian detection (Sermanet, Kavukcuoglu, Chintala, & Lecun, 2013), pose estimation with a deep network (Ouyang, Chu, & Wang, 2014), facial feature tracking with a restricted Boltzmann machine (Wu, Wang, & Ji, 2013), shape prior detection using deep learning for object segmentation (Chen, Yu, Hu, & Xunxun, 2013), and object detection with a deep network (Szegedy, Toshev, &

* Corresponding authors. Fax: +66 0 2218 6955.
E-mail address: peerajak@gmail.com (P. Witoonchart).

http://dx.doi.org/10.1016/j.neunet.2017.02.005
0893-6080/

Erhan, 2013). The most similar method to that proposed in our study is that described by Yang, Ouyang, Li, and Wang (2016), which resulted in a great improvement in the percentage of correct parts (PCP), where the PCP accuracy was high as 81%. They formulated the problem as the end-to-end backpropagation of SVM hinge loss, whereas we formulate the problem as a structured SVM (SSVM) (Tsochantaridis, Hofmann, Joachims, & Altun, 2004).

Our approach starts from Girshick's claim that a convolutional neural network (CNN) (LeCun, Bottou, Bengio, & Haffner, 1998) is a deformable part model (DPM) (Girshick, Iandola, Darrell, & Malik, 2015). However, they did not backpropagate the error from the DPM model to the CNN. If a DPM is a CNN, then the error must be backpropagated to the lower layer. Thus, we backpropagate the error of the DPM back to the CNN using the SSVM loss function.

To apply latent SVM (Yang & Ramanan, 2011) with deep learning, the normal method involves extracting features with a deep neural network and performing latent SVM learning as two distinct stages using the same pipeline, i.e., feature extraction, caching the result from the first stage, and then submitting it to the latent SVM learning algorithm in the second stage. For example Girshick (Girshick et al., 2015) extracted a pyramid of features from a CNN in the feature extraction stage, before caching the extracted features, and then learning with a latent SVM during the second stage. In the second stage, the latent SVM then learned all of the model parameters by switching between SVM optimization and inference combinatorial optimization. However, this type of method has inherent problems because it cannot learn the parameters extracted by the deep learning feature from the inference optimization error because they are conducted in two distinct stages. The learnable feature extraction parameters cannot be updated based on the error of the latent SVM. Thus, we propose SSVM CNN to address this problem.

In future research, we plan to use deep CNN as a feature extractor because it is known to perform well (Yang et al., 2016). However, in the present study, we determined the feasibility of this method by showing that minimizing the loss with the SSVM CNN can be employed for part-based detection. There may be cases where the losses are minimized well but part-based detection is not achieved. If this method is feasible, we aim to employ deep CNN as the front end feature extractor instead of HOG and the SSVM neural network as the back end for HPE. In this study, we demonstrated that it is feasible to use our SSVM CNN method as the back end in the planned system.

2. Our method

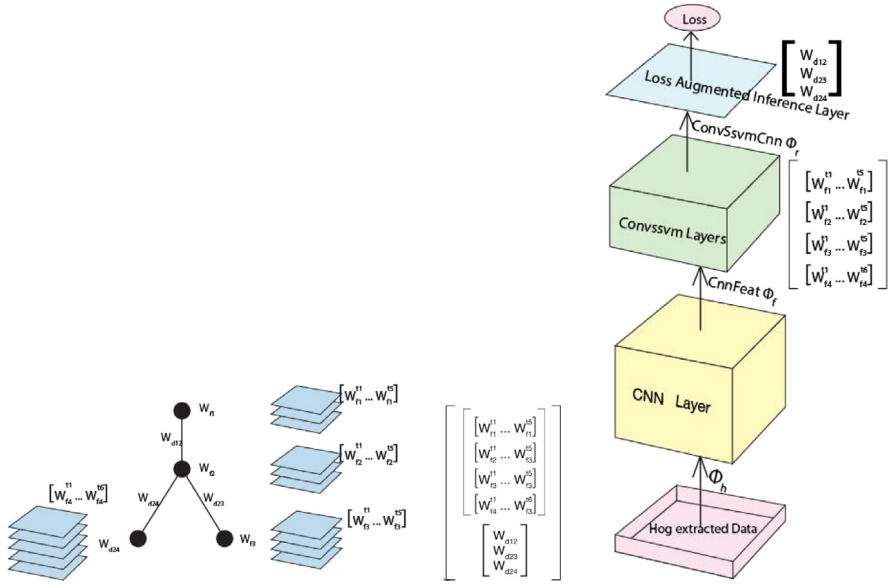
HPE is the problem of estimating human joint positions. The joints are considered to have been estimated correctly if the predicted error in the difference among pixels is less than a certain threshold. The HPE problem involves the following challenges: (1) the size of a human can be small or large depending on how close the camera is to the human when obtaining an image; (2) body parts can differ in their appearance, e.g., the appearance of a hand can be a fist or palm, while a limb can be vertical or horizontal; (3) human pictures are taken in three-dimensional (3D) contexts, but there may be many 2D pose appearances for the same 3D pose, and there are many possible 3D poses. To address these challenges, the following methods have been implemented in previous studies. (1) Multiscale human detection, which is sometime called an image pyramid or feature pyramid. An image of a feature $\mathbf{x}$ is scaled to all the layer values $l \in \mathrm{L}$, where $\mathrm{L} = \{1, \ldots, l_n\}$. (2) Mixture of part types. To address the different possible appearances of each human part, each human part model is designed so it comprises multiple different part types. The body parts obtained from training images are clustered into part types based on their relative joint positions in the image coordinates with respect to neighboring joints. The underlying assumption of this clustering method is that the same group of relative joint positions will appear similar. (3) Co-occurrence model. This model consider how two neighboring parts co-occur according to a system of biases. Each type of mixture of neighboring nodes has an associated bias. These measures are incorporated into a well-known pictorial structure model where their edges are quantified under the assumption that the energy required for placing parts only varies quadratically based on the relative distance, such as the energy required for stretching or compressing springs from their anchor positions with respect to their parent nodes. These models were developed directly from that proposed by Yang and Ramanan (2011), where they combined these three models into a single large model, i.e., the coocurrence model, deformable model, and appearance model. The first two models are called structural models in our study.

We propose a new method for learning the models mentioned above. We propose to jointly learn all the models, all their parameters, in the way the neural network learns their parameters by learning all of them using stochastic subgradient descent. In contrast to Yang and Ramanan (2011), who proposed learning the appearance model prior to structural training, our method can learn all the parameters by random initialization. We note that the DPM unary filters method is exactly equal to a convolutional operation during DPM inference. During DPM inference, each unary filter determines the "sliding dot product" between feature matrices and weight matrices. This operation is exactly equal to the weights of the convolutional layer in the CNN acting on their input matrices. Thus, we design the DPM unary filters, which vary according to the human parts and human part types, as the convolution layer in the CNN. This DPM filters defines the appearance model's weights because they give the feature similarity scores. We design the cooccurrence model's weights as parameters on the loss augmented inference layer. We design the DPM pairwise weights, deformable weights, or simply the spring term's weights as other types of parameters on the loss augmented inference layer. These designs are shown in Fig. 1. In this section, we show how an SSVM based on DPM can be implemented as a two-layer neural network, where the first layer is the convolutional layer (Convssvm layer in Fig. 1) and the other is the loss augmented inference layer (loss augmented inference layer in Fig. 1). By transforming the structured form of the model into a neural network in the model, our proposed method can jointly learn the structural model and appearance model, and then backpropagate the error to learn the underlying learnable parameters that can be extracted from the appearance model features (CNN layer in Fig. 1). Thus, our proposed method translates the SSVM model into a neural network model, and thus it inherits the neural network's innate ability to backpropagate the error to a lower layer, as well as calculating the exact SSVM loss and learning the original SSVM with a subgradient-based method.

2.1. DPM problem formulation

Our approach starts by formulating the DPM as an instance of SSVM learning. As shown by Ratliff, Bagnell, and Zinkevich (2007), the SSVM can be learned by subgradient descent. Thus, we start by formulating the detection problem.

Let $\mathbf{x_i}$ represent a matrix of RGB values for the ith training data and $\mathbf{y_i}$ represents the ith training bounding boxes label, where (top left column, top left row, bottom right column, bottom right row) are denoted by $[x_1, y_1, x_2, y_2]$. Each row of $\mathbf{y_i}$ represents each bounding box for each part in a similar manner to the method proposed by Yang and Ramanan (2011). Therefore, $\mathbf{y_i}$ is a matrix of $numparts \times 4$. The bold characters indicate that they are either vectors or matrices. First, we scale $\mathbf{x_i}$ to multiple scales. To define this scaling, let $\mathrm{L} = \{1, \ldots, l_n\}$ denote the set of all scaling levels.



(a) Structured form of the model weights.

(b) Vectorized form of the model weights.

(c) Neural network form of the model weights based on the two topmost layers.

Fig. 1. Proposed method based on the formulation of an SSVM two layer neural network, which are the two topmost (blue and green) layers in (c). The appearance model weights are the bottom Convssvm layer (green layer), which is the normal convolutional layer. Loss augmented inference based on the top layer (blue layer) has pairwise weights, i.e., the deformable model's weights, and the cooccurrence model's weights, which can be treated as structural weights. The deformable weights, $\mathbf{w}_{\mathbf{ef_d}}$, are shown as weights between the edges in (a), as subvectors in a concatenated vector of weights in (b), and as loss augmented inference weights in (c). The appearance model weights, $\mathbf{w}_{\mathbf{f}}^{\mathbf{t}}$, are shown as the weights for nodes in (a), as subvectors in a concatenated vector of weights in (b), and as the Convssvm layer in (c).

Let $l \in \mathrm{L}$ denote the lth layer. By scaling image x_i to multiple scales, we obtain an image pyramid of all $l \in \mathrm{L}$. The image pyramids are simply multiple RGB images, but each differs in terms of its scale. Each of these different sized images are pasted into a zero image matrix, thereby obtaining a four-dimensional matrix of size width $\times$ height $\times$ color channels $\times$ number of levels. The image pyramid denoted as $\mathbf{x_{iL}}$ is then used as the input for HOG feature extraction, which is denoted as $\mathbf{\Phi_h}(\mathbf{x_{iL}})$. Next, the HOG-extracted feature is used as an input for the CNN as deep pyramid features, where we refer to this CNN as *CnnFeat*. The result obtained by *CnnFeat* for $\mathbf{x_{iL}}$ is denoted as $\mathbf{\Phi_f}(\mathbf{x_{iL}})$.

2.2. Model and detection inference

Our model based on Yang and Ramanan (2011) has three major submodels and these submodules are defined as follows.

Appearance model

The appearance model comprises the individual filters for the individual type of part that the model designer aims to model, e.g., head filters and body filters. Our image representation usually has many channels, so each of these models is represented by a matrix comprising the filter size times the number of channels. Typical values are $5 \times 5 \times 32$ or $5 \times 5 \times 64$ for a filter of 5×5 with 32 and 64 channels, respectively. By determining the dot product of the filter and a feature of the same size, we obtain a particular score for a feature. These filters are in the domain of $\mathbf{R}^{S \times C}$, where S is the filter size and C is the number of channels. For each part and each type of part, there is an associated appearance model filter. The appearance model is the appearance filter. The similarity score created by a filter $\mathbf{w}_{\mathbf{f}}^{\mathbf{t}}$ is

$$score_{appearance}(y) = \mathbf{w}_{\mathbf{f}}^{\mathbf{t}} \cdot \mathbf{\Phi_f}(\mathbf{x_L}, y). \quad (1)$$

Co-occurrence model

If we suppose that there are m mixture types for a part and n mixture types for a neighboring part, then the total bias among these two parts is $m \times n$. This model gives the sum of the local and pairwise scores. For a parent node i and a child node j, the co-occurrence score ij is

$$score_{cooccurrence}(t_i, t_j) = b_{ij}^{t_i t_j}. \quad (2)$$

This can be treated as a bias to favor some particular local types, as well as the pairwise relationship among parent and child types. For example, if $b_{34}^{t_1 t_2}$ has a high value, this means that the parent part number 3, type 1 is likely to connect to child part number 4, type 2.

Deformable model

From each parent type t_i to each child part t_j, we have anchor positions for the the child wrt the parent, where there are a total of $t_i \times t_j$ anchors from parents to children. The anchor positions are trained in order to model them simply by taking the average of each of all the possible types of connections. The anchor positions must be available before SSVM training, so the part types are calculated by simple K-mean clustering for each appearance type to create different types of articulations. We employ mixtures of components to solve the many types of possible part appearances in the same manner as Yang and Ramanan (2011). Let $p \in \mathrm{P}$ be the pth body part, where $\mathrm{P} = \{1, \ldots, p_n\}$ is the set of all parts. Let $k \in \mathrm{K}$ be the kth type of a particular part, where $\mathrm{K} = \{1, \ldots, k_n\}$ is the set of all types of a particular part. Let K_p denote the total number of types K for a particular part p. We start by clustering the training image parts $p \in \mathrm{P}$ into K_p clusters. Now, we define the ith sample's SSVM feature function. The SSVM has a DPM tree $\mathbf{G} = \{\mathbf{V}, \mathbf{E}\}$ of features. First, we define the unary feature $\mathbf{\Phi_f}(\mathbf{x_{iL}}, \mathbf{y_i})$ as $\mathbf{\Phi_f}(\mathbf{x_{iL}})$ for evaluating at position $\mathbf{y_i}$. We then define the pairwise features

$$\Psi_{ij} = -\left[dx_{ij} \quad dy_{ij} \quad dx_{ij}^2 \quad dy_{ij}^2, \right]$$

where $dx_{ij} = x_{pi} - x_{pj} + anchor_x$ and $dy_{ij} = y_{pi} - y_{pj} + anchor_y$. We denote $\mathbf{\Psi}$ as $\left[\Psi_{ij}\right], \forall ij \in E$. Integrating the unary and pairwise features of the tree G gives $\mathbf{\Phi_a} = [\mathbf{\Phi_f} \quad \mathbf{\Psi},]$ where the subscript a denotes the word *all*. The score obtained by the deformable model of an edge ij is

$$score_{defrom}(i,j) = \mathbf{w_{ij_d}} \cdot \Psi_{ij}. \tag{3}$$

Assembling the submodels

For each node and edge in the pictorial structure, we concatenate all the bias weights, deformable weights, and appearance filter weights into two types of data structures. The first is the struct type based on the components and the second is the vector type. Using the vector type data structure, we create a large vector $\mathbf{w}$ comprising the learnable HPE parameters for SSVM learning:

$$score(t,y) = \sum_{i \in V} \mathbf{w_f^t} \cdot \mathbf{\Phi_f}(\mathbf{x_L}, \mathbf{y}) + \sum_{ij \in E} b_{ij}^{t_i t_j} + \mathbf{w_{ij_d}} \cdot \mathbf{\Psi_{ij}}, \tag{4}$$

or in matrix form,

$$score(t,y) = \mathbf{w} \cdot \mathbf{\Phi_a}(\mathbf{x_L}, \mathbf{y}). \tag{5}$$

To find the location $\mathbf{y}$ where the value of the score is maximizes, the equation above becomes:

$$\hat{\mathbf{y}} = \arg\max_{\hat{y} \in Y} \mathbf{w} \cdot \mathbf{\Phi_a}(\mathbf{x_L}, \mathbf{y}), \tag{6}$$

which is the prediction function. The value of $\hat{\mathbf{y}}$ is our prediction of the part locations for an unseen test image.

2.3. Subgradient optimization of SSVM

Next, we introduce the SSVM and a solver for obtaining the SSVM objective function. We employ the framework proposed by Ratliff et al. (2007) and Tsochantaridis et al. (2004). We apply this framework for generic structured learning to the HPE problem.

The aim of the SSVM is to produce a structural prediction by learning the maximum margin classifier for each of the training data. Probabilistic graphical models such as Markov random fields (MRFs) or conditional random fields can use an SSVM in the learning phase to learn the weight parameters. The SSVM is not an algorithm where data can be simply plugged in for learning or classification, such as the SVM, but instead it is a framework where inference, loss, and feature modules are need to be specified before it can be used. For example, if an SSVM is applied to an MRF, we must specify the MRF structure that the SSVM will learn and the MRF inference algorithm, as well as the MRF feature function, loss function, and loss augmented inference algorithm. The loss augmented inference algorithm is the inference algorithm with a loss function. Next, the SSVM learns the weights that maximize the prediction based on the training data. The SSVM learns a structural prediction function in the form of (6).

The SSVM objective function has the form described by Ratliff et al. (2007). Next, we describe how the method described by Ratliff et al. (2007) can be applied in our study. The application of their method to our problem leads to the formulation in the next subsection. To learn the model parameters in (6), we define the objective function for the SSVM in a similar manner to Ratliff et al. (2007):

$$\begin{aligned} &\min_{w,\xi} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{l} \xi_i \\ &s.t.\ \forall i \quad \mathbf{w} \cdot \Phi_{ai}(\mathbf{x_i}, \mathbf{y_i}) + \xi_i \\ &\quad \geq \max_{y \in Y} \mathbf{w} \cdot \Phi_{ai}(\mathbf{x_i}, \mathbf{y}) + \Delta(y, y_i). \end{aligned} \tag{7}$$

By minimizing the objective function above, we can learn the parameters $\mathbf{w}$ that maximize the training accuracy for the prediction function equation (6). The SSVM objective (7) can be solved by the subgradient method, as shown by Ratliff et al. (2007). In our context, the subgradient of the objective loss function is defined by:

$$\frac{\partial Obj_i}{\partial \mathbf{w}} = \mathbf{\Phi_{ai}}(\mathbf{x_i}, \hat{\mathbf{y}_i}) - \mathbf{\Phi_{ai}}(\mathbf{x_i}, \mathbf{y_i},) \tag{8}$$

where Obj_i is the minimization objective function (7). For the bth batch of many training data, the gradient is

$$\frac{\partial Obj_b}{\partial \mathbf{w}} = \frac{1}{b} \sum_{b} \mathbf{\Phi_{ab}}(\mathbf{x_b}, \hat{\mathbf{y}_b}) - \mathbf{\Phi_{ab}}(\mathbf{x_b}, \mathbf{y_b}), \tag{9}$$

where $\hat{\mathbf{y}_b}$ is the most violated constraint according to loss augmented inference.

In our context, the loss augmented inference is defined as

$$\hat{\mathbf{y}} = \arg\max_{y \in Y} \Delta(\mathbf{y}, \mathbf{y_i}) + \mathbf{w} \cdot \mathbf{\Phi_a}(\mathbf{x_i}, \mathbf{y}) - \mathbf{w} \cdot \mathbf{\Phi_a}(\mathbf{x_i}, \mathbf{y_i}), \tag{10}$$

where $\Delta(\mathbf{y}, \mathbf{y_i}) = 1 - \frac{Area(\mathbf{y} \cap \mathbf{y_i})}{Area(\mathbf{y} \cup \mathbf{y_i})}$ is the standard one minus bounding box intersection over the union loss. We then apply subgradient updating as normal stochastic gradient descent.

2.4. SSVM as a two-layer neural network

We extend the previously defined subgradient optimization of the SSVM by backpropagating further down to the lower layer of the neural network because solving the SSVM defined in the previous section can be implemented as a two-layer neural network. The top layer is the loss augmented inference layer and the lower layer is the normal linear layer in the neural network. In our special case of DPM as a CNN, the lower layer is the standard convolution layer.

To solve the SSVM by subgradient optimization, we must calculate the loss augmented inference $\hat{\mathbf{y}_i}$ such that the feature of the most violated constraint $\mathbf{\Phi_{ai}}(\mathbf{x_i}, \hat{\mathbf{y}_i})$ of (8) can be calculated. We note that the sliding dot product of $\mathbf{w_f^t}$ over all $\mathbf{\Phi_f}(\mathbf{x_{iL}})$ possible locations of $\hat{y}$ is actually a convolution operation in a exactly equals to convolution layer in CNN. If we design this convolution operation as a convolutional layer in the CNN, then the DPM filters are convolutional filters of the CNN. The *CnnFeat* topped with DPM filters is now called *ConvSsvmCnn* and the DPM filters are called the *ConvSsvm* layer. Their corresponding feedforward is

$$\mathbf{\Phi_{rf}^t}(\mathbf{x}, \mathbf{y}) = \sum_{\bar{\mathbf{y}}} \mathbf{w_f^t}(\bar{\mathbf{y}})\, \mathbf{\Phi_a}(\mathbf{x}, \mathbf{y} + \bar{\mathbf{y}}). \tag{11}$$

Thus, (11) defines a lower layer of the SSVM as a two-layer neural network. The quantity is actually the response map of $\mathbf{\Phi_f}(\mathbf{x_{iL}})$ convoluted by DPM unary filters $\mathbf{w_f^t}$. Let us denote $\mathbf{\Phi_r}$ as the concatenation of all $\mathbf{\Phi_{rf}^t}$. Using the constructed *ConvSsvm* layer, we now define the loss augmented inference layer in this two-layer neural network context. Clearly, the loss augmented inference layer performs loss augmented inference and finds the slack loss of the objective function, and $\hat{\mathbf{y}}$ is the most violated constraint as the argument that maximizes the loss augmented inference objective. The loss augmented inference objective function (10) can be rewritten using (4), (6), and (11) as follows

$$\begin{aligned} L_i = &\max_{y \in Y} \Delta(y, y_i) + \sum_{v \in V} \{\mathbf{\Phi_r}(\mathbf{x_{iL}}, y_v) - \mathbf{\Phi_r}(\mathbf{x_{iL}}, y_{vi})\} \\ &+ \sum_{ef \in E} \left\{\mathbf{w_{ef_d}} \cdot \mathbf{\Psi_{ef}}(\mathbf{x_i}, \mathbf{y}) - \mathbf{w_{ef_d}} \cdot \mathbf{\Psi_{ef}}(\mathbf{x_i}, \mathbf{y_i})\right\} \\ &+ \sum_{ef \in E} \left\{\left(b_{ef}^{t_e t_f}\right)_{\mathbf{y}} - \left(b_{ef}^{t_e t_f}\right)_{\mathbf{y_i}}\right\}. \end{aligned} \tag{12}$$

The term $\mathbf{\Phi_r}(\mathbf{x_{iL}}, \mathbf{y}) - \mathbf{\Phi_r}(\mathbf{x_{iL}}, \mathbf{y_i})$ can be treated in the neural network sense as picking two scalars from the matrix of the lower layer and performing subtraction. Thus, (12) defines the upper layer of the SSVM as a two-layer neural network and we can see that $\mathbf{w}$ in (6) is now divided into two different layers. The weights of the appearance model equation (1) are at the bottom of the convolutional layer. The weights of the co-occurrence and deformable models (2), (3) are at the top of the loss augmented inference layer. Fig. 1 shows the outputs of these two equations: (11), and (12).

Next, we define our back-propagation rules for the two-layer neural network. The gradient of the top layer, i.e., the loss augmented inference layer, is

$$\frac{\partial L_b}{\partial \mathbf{w}} = \frac{1}{b}\sum_b \Big\{ \mathbf{\Psi_b}\left(\mathbf{x_b}, \hat{\mathbf{y}_b}\right) - \mathbf{\Psi_b}\left(\mathbf{x_b}, \mathbf{y_b}\right) + \left[\delta\left(\hat{y}^{t_i t_j}\right)\right]_b - \left[\delta\left(y_i^{t_i t_j}\right)\right]_b \Big\}, \tag{13}$$

where $[\delta(a)]$ is a vector with elements that are all zero except for the element position at a. The gradient of the objective function wrt the response map layer, $\mathbf{\Phi_r}(\mathbf{x_{iL}})$, is

$$\frac{\partial L_i}{\partial \mathbf{\Phi_r}} = \delta\left(y_v, \hat{y}_v\right) - \delta\left(y_v, y_{iv}\right), \quad \forall y \in \mathrm{Y}, \forall v \in \mathrm{V}, \tag{14}$$

where $\delta(a, b) = 1$, if $a = b$, and $\delta(a, b) = 0$ otherwise. This can be obtained by creating a blank matrix of size $\mathbf{\Phi_r}(\mathbf{x}, \mathbf{y})$, and then setting $+1$ at position $\bar{y}_v$ and -1 at position y_{iv}, but if some $\bar{y}_v = y_{iv}$, then set 0 at that position.

The two gradients specified above define the loss augmented inference layer. In terms of the the backpropagation rule for *ConvSsvm* layers, which define the appearance layer of the pictorial structure, we can use a normal backpropagation rule for the convolution layer of the CNN.

2.5. Solving inferences with the max-sum algorithm

To solve for $\hat{\mathbf{y}}$ with the configuration that maximizes (12), we can use the standard max-sum algorithm. The max-sum algorithm aims to find the solution of the combinatorial optimization problem with the form,

$$\hat{L} = \arg\max_L \sum_{i \in V} m_i(l_i) + \sum_{ij \in E} g\left(l_i, l_j\right), \tag{15}$$

under Graph $G = \{V, E\}$. The objective of the combinatorial optimization problem, (12), is

$$\hat{\mathbf{y}} = \arg\max_{y \in \mathrm{Y}} \sum_{v \in V} \{\mathbf{\Phi_r}(\mathbf{x_{iL}}, y_v) + \Delta(y_v, y_{iv})\} + \sum_{ef \in E} \left\{ \mathbf{w_{ef_d}} \cdot \mathbf{\Psi_{ef}}(\mathbf{x_i}, \mathbf{y}) + \left(b_{ef}^{t_e t_f}\right)_{\mathbf{y}} \right\}. \tag{16}$$

This combinatorial objective problem is solved by the max-sum algorithm. Thus, the max-sum algorithm is performed on the topmost layer of our neural network.

3. Experimental

We trained our SSVM in the same manner as training a neural network using the stochastic gradient descent. Our network architecture can be the normal CNN connected with the newly formulated SSVM neural network as the last two layers. We can also simply connect the data layer with the newly formulated SSVM neural network as the last two layers. We implemented the loss augmented inference as a layer in the Caffe library (Jia et al., 2014).

3.1. Data preparation

The PARSE dataset comprises 100 positive training samples and 205 positive testing samples. Every image in this dataset show the full body of a human, usually in a sporting context. For every sample, the corresponding human joint positions are labeled. Some of the human parts are occluded in the images, but the estimated human joint positions are provided. In total, 14 joints are labeled for each sample, including the head, torso, left arm, right arm, left leg, and right leg. The sizes of the images range from 110–450 × 110–450. In some images, one or two full human bodies are present. The human body size also varies in the images. The background usually contains no humans, except for small heads in the audience on a sports screen. The human poses in the dataset vary from sitting to standing, and the legs and arms may be engaged in sports such as kung fu or gymnastics.

The dataset was not suitable for our algorithm so preprocessing was performed, as follows. The training data were mirror-flipped with the training labels and then added to the original training data. Thus, we had a double-sized set of training data. We then found the mid-point of each two joints, thereby obtaining a total of 26 joint and mid-joint points. We changed these points into boxes, where the box size was calculated by averaging the lengths between the joints in the training data. At this stage, our labels for the training algorithm were defined as (x_1, y_1, x_2, y_2) for each box. There were 26 boxes for each training label, which comprised our $\mathbf{y}$ image space.

The training image was then passed through the HOG pyramid feature. In this process, a training image was resized to obtain multiple images with different sizes based on the same picture, which is called a pyramid of images. The image pyramid was extracted by HOG to obtain a pyramid of features, which were then patched into a zero matrix to obtain $\mathbf{\Phi_h}(\mathbf{x_{iL}})$. Thus, the matrix outside the image boundary had values of 0, as shown in Fig. 3. These layers represent multiscale features of the image, which ranged from small to large. The brown rectangle inside the larger rectangle shows the real data obtained from the HOG features. The blue area in the larger rectangle contains all zeros. Note that the brown rectangle differed in size in each layer because of the multiscale features. The upper left of Fig. 3 shows a batch of pyramids for features before they were fed into the neural network. Typically, the largest size of a feature was about 150 × 150 × 32, which means that the feature size was 150 × 150 with 32 channels. The smallest feature size was typically about 30 × 30 × 32. Typically, there were 5–20 levels in a pyramid. A batch usually comprised five training images. The feature pyramids at all levels and batch sizes were then patched into a zero matrix of size 150 × 150 × 32 × (number of levels × number of image per batch) to obtain a four-dimensional matrix.

The label was added to the feature pyramid parameters to produce a practical label. This practical label and the batch data were then converted into a Lightning Memory-mapped Database so the Caffe GPU library could handle the data more efficiently.

3.2. Neural network structure

HOG-Conv-SSVM

The neural network structure was as follows: data layer — convolution layer *CnnFeat* — convolution layer *ConvSsvm* — loss augmented inference layer. The convolutional layer was placed in the middle in order to show that our method could learn the deep learning feature extraction parameters in the middle. Our weights were initialized randomly. Our mixture model for each node was as follows: $M_{ixture} = \{5, 5, 5, 6, 6, 6, 6, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 5, 5, 5, 5, 5, 5\}$.

Table 1
Strict percentage of correct points (PCP) (Yang et al., 2016) comparison.

Class	Our result	Yang and Ramanan (2011)
Head	59.0	**77.6**
Torso	75.1	**82.9**
L. arm	13.9	**35.4**
U. arm	34.6	**55.1**
L. leg	46.1	**63.9**
U. leg	55.1	**69.0**
total PCP	43.4	**60.7**

This means that the first node (head node) had five different mixture type, the second node had five different mixture type, and so on. In total, there were 138 $\sum_i M_{ixture}$ mixtures. The size of the *ConvSsvm* layer was $5 \times 5 \times 256 \times 138 = 883{,}200$, where 138 is $\sum_i M_{ixture}$. The loss augmented inference layer had $1 + \sum_i M_{ixture}(i) \times M_{ixture}(i-1) = 702$ weight elements of $\forall vq \in E$, $b_{vq}^{t_v t_q}$, and $133 \times 4 = 532$ weight elements of $\mathbf{w_{deform}}$. Therefore, the loss augmented inference layer had a total of 1234 weight elements. The *ConvSsvm* layer had a total of $5 \times 5 \times 256 \times 138 = 883{,}200$ weight elements. We set the size of the kernel for *CnnFeat* as $2 \times 2 \times 32$, so *CnnFeat* had a total of $2 \times 2 \times 32 \times 256 = 32{,}768$ weight elements. Therefore, our system had a total of $883{,}200 + 1234 + 32{,}768 = 917{,}202$ weight elements. The batch size was set to 50 and the feature size $\mathbf{\Phi_h}$ was set to 140×140. The data layer had $50 \times 32 \times 140 \times 140 = 31{,}360{,}000$ elements. The *CnnFeat* layer had $50 \times 256 \times 139 \times 139 = 247{,}308{,}800$ elements. The *ConvSsvm* layer had $50 \times 138 \times 135 \times 135 = 125{,}752{,}500$ elements. In total, we needed 404,421,300 elements to store our multilayer neural network data. The total GPU memory requirement for our system was 1,284,714,404 bytes. Our backpropagating elements were all in the neural network layers except for the data layer plus all the weights. There were 373,978,502 backpropagating elements.

We trained over 3000 iterations at a learning rate of 0.005. We used the L2 regularization terms with a coefficient of 0.1 and no momentum parameter.

3.3. Implementation as a Caffe layer

After $\mathbf{\Phi_h}(\mathbf{x_{iL}})$ in the data layer passed through the *CnnFeat* layer, $\mathbf{\Phi_f}(\mathbf{x_{iL}})$ was obtained. A further forward pass through the *ConvSsvm* layer produced $\mathbf{\Phi_r}(\mathbf{x_{iL}})$ as the response map, or heat map, as shown in Fig. 3. The loss augmented inference layer was implemented using the python layer in the Caffe library (Jia et al., 2014). The loss augmented inference layer used the max-sum algorithm to calculate the objective function's value. The max-sum algorithm's output comprised the optimal level and the most violated constraint $\hat{\mathbf{y}}$, where their output feature values were $\mathbf{\Phi_a}(\mathbf{x_{iL}}, \hat{\mathbf{y}})$, and $\mathbf{\Phi_a}(\mathbf{x_{iL}}, \mathbf{y_i})$, respectively. The max-marginal value maximized equation (16) and the loss function $\Delta(\hat{\mathbf{y}}, \mathbf{y_i})$, which was obtained by searching for each pyramid level and then finding the best max-marginal score. A higher score in the new pyramid level replaced the previous result.

After calculating the most violated constraint, we calculated the gradients in (13) and (14) exactly as defined in these equations. The average gradient and cost function for each batch size was calculated by normal mini-batch training with the neural network.

3.4. Results

We trained and tested our method based on the PARSE dataset (Ramanan, 2007).

The gradient descent results obtained according to the loss augmented inference loss, L_i, using (12) as the number of training

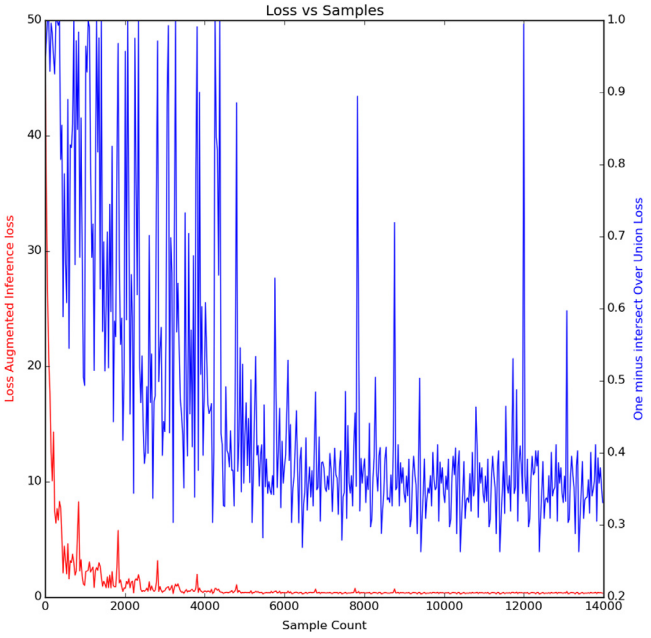


Fig. 2. Slack loss, one minus intersect over union loss, $\Delta(\hat{\mathbf{y}}, \mathbf{y_i})$, versus samples.

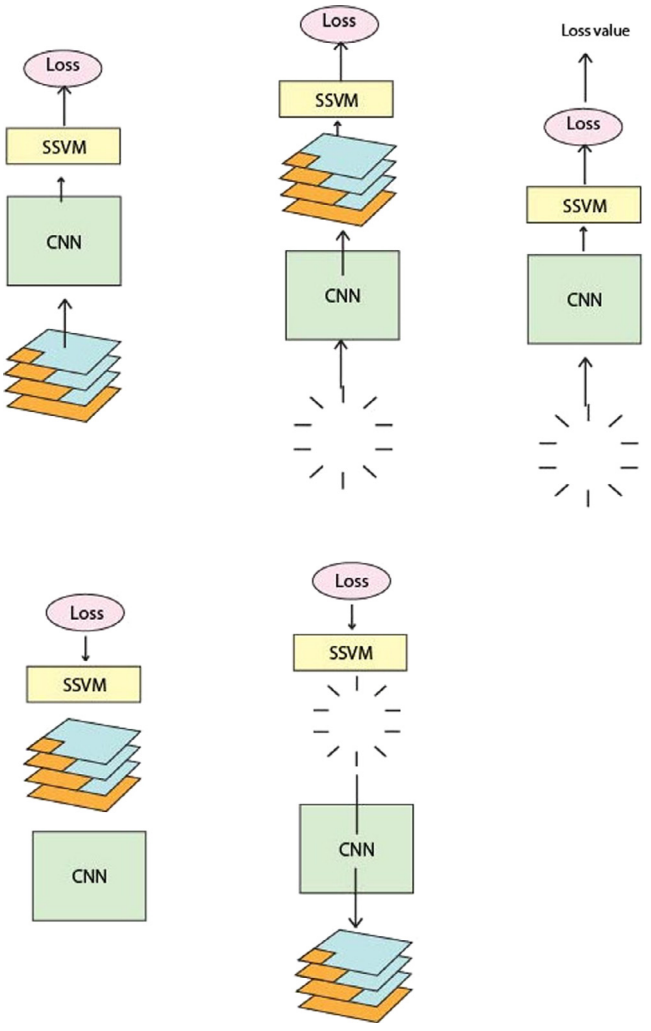


Fig. 3. Feature pyramid fed into the SSVM neural network as many layers. The blue color on layers of data are filled with zeros. The brown color on layers of data are filled with feature values.

Fig. 4. Visualization of our HPE results based on the PARSE test dataset. The green bounding boxes are a head. The yellow bounding boxes are a torso. The cyan bounding boxes are a left arm. The blue bounding boxes are a right arm. The red bounding boxes are a left limb. The deep blue bounding boxes are a right limb.

samples decreased are shown in Fig. 2. The one-minus-intersect-over-union, $\Delta\left(\hat{y}, y_i\right)$, loss was reduced to around 30%–40%. The value of L_i decreased from a value of 50 to 0.5, which shows that our gradient-based method was implemented correctly.

We compare our results with those obtained by Yang and Ramanan (2011) in Table 1, where the total PCP (as defined by Yang et al., 2016) is used as the metric. Our results are not competitive with their results, but Fig. 2 shows that our method can learn effectively.

Fig. 4 shows the HPE results based on a sample from the PARSE test set, where our results appear very promising.

4. Conclusion

Currently, many part-based detection methods rely on a CNN as the front end. Many studies have shown that classification and feature extraction by backpropagation of the classifier into a deep learning feature extractor gives better performance. We plan to apply our novel SSVM neural network layer with deep CNN to solve part-based image detection problems. In this study, we determined the feasibility of this approach by showing that reducing the loss of an SSVM neural network can be applied to part-based detection. In future research, we will add this new layer to deep CNN to create a full end-to-end neural network for solving the HPE problem, as well as other part-based detection problems.

In this study, we proposed a new learning algorithm called SSVM CNN for learning HPE models. We defined the SSVM as a two-layer neural network that can backpropagate the inference error to a deep learning feature extractor. Our SSVM within the SSVM CNN learns its weights in exactly the same way as it learns weights without backpropagation. Our method operates as a generic learning algorithm for broad deep learning computer vision problems. Our results are not yet as good as those obtained by state-of-the-art approaches, but our method can learn the model parameters for HPE without requiring negative samples. We consider that our method can perform much better if we use deep CNN as the front end. Due to the time-consuming nature of our training process, we have not tested the use of deep CNN as a front end, but we will implement our method as a full deep learning machine in the future.

References

Chen, F., Yu, H., Hu, R., & Xunxun, Z. (2013). Deep learning shape priors for object segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1870–1877). http://dx.doi.org/10.1109/CVPR.2013.244, URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6619088.

Cherian, A., Mairal, J., Alahari, K., & Schmid, C. (2014). Mixing body-part sequences for human pose estimation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2353–2360).

Dantone, M., Gall, J., Leistner, C., & Van Gool, L. (2014). Body parts dependent joint regressors for human pose estimation in still images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *36*(11), 2131–2143. http://dx.doi.org/10.1109/TPAMI.2014.2318702, URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6802375.

Felzenszwalb, P. F., & Huttenlocher, D. P. (2005). Pictorial structures for object recognition. *International Journal of Computer Vision*, *61*(1), 55–79.

Girshick, R., Iandola, F., Darrell, T., & Malik, J. (2015). Deformable part models are convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 437–446).

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., & Girshick, R. et al. (2014). Caffe: Convolutional architecture for fast feature embedding, arXiv preprint arXiv:1408.5093.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2323.

Ouyang, W., Chu, X., & Wang, X. (2014). Multi-source deep learning for human pose estimation. In *2014 IEEE conference on computer vision and pattern recognition* (pp. 2337–2344). http://dx.doi.org/10.1109/CVPR.2014.299, URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6909696.

Pishchulin, L., Andriluka, M., Gehler, P., & Schiele, B. (2013). Poselet conditioned pictorial structures. In *Proceedings of the IEEE computer society conference on computer vision and pattern recognition* (pp. 588–595).

Ramanan, D. (2007). Learning to parse images of articulated bodies. *Advances in Neural Information Processing Systems*, *19*, 1129–1136. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.91.6416&rep=rep1&type=pdf.

Ratliff, N. D., Bagnell, J. A., & Zinkevich, M. a. (2007). Subgradient methods for structured prediction. *Artificial Intelligence and Statistics*, (Online) URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.73.2438.

Sermanet, P., Kavukcuoglu, K., Chintala, S., & Lecun, Y. (2013). Pedestrian detection with unsupervised multi-stage feature learning. In *Proceedings of the IEEE computer society conference on computer vision and pattern recognition* (pp. 3626–3633). arXiv:arXiv:1212.0142v2 http://dx.doi.org/10.1109/CVPR.2013.465.

Sun, Y., Wang, X., & Tang, X. (2013). Deep convolutional network cascade for facial point detection. In *Proceedings of the IEEE computer society conference on computer vision and pattern recognition* (pp. 3476–3483). http://dx.doi.org/10.1109/CVPR.2013.446.

Szegedy, C., Toshev, A., & Erhan, D. (2013). Deep neural networks for object detection. In *Advances in neural information* (pp. 1–9). URL http://papers.nips.cc/paper/5207-deep-neural-networks-for-object-detection.

Tian, T.P., & Sclaroff, S. (2010). Fast globally optimal 2D human detection with loopy graph models. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (pp. 81–88).

Tian, Y., Zitnick, C. L., & Narasimhan, S. G. (2012). Exploring the spatial hierarchy of mixture models for human pose estimation. In A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, & C. Schmid (Eds.), *Lecture Notes in Computer Science*: *Vol. 7576. Computer vision — ECCV 2012* (pp. 256–269). Springer.

Tsochantaridis, I., Hofmann, T., Joachims, T., & Altun, Y. (2004). Support vector machine learning for interdependent and structured output spaces. In *ICML* (p. 104). ACM Press, URL http://portal.acm.org/citation.cfm?doid=1015330.1015341.

Wu, Y., Wang, Z., & Ji, Q. (2013). Facial feature tracking under varying facial expressions and face poses based on restricted Boltzmann machines. In *Proceedings of the IEEE computer society conference on computer vision and pattern recognition* (pp. 3452–3459). http://dx.doi.org/10.1109/CVPR.2013.443.

Yang, W., Ouyang, W., Li, H., & Wang, X. (2016). End-to-end learning of deformable mixture of parts and deep convolutional neural networks for human pose estimation. In *CVPR*.

Yang, Y., & Ramanan, D. (2011). Articulated pose estimation with flexible mixtures-of-parts. In *CVPR 2011* (pp. 1385–1392). IEEE, http://dx.doi.org/10.1109/CVPR.2011.5995741, URL http://www.mendeley.com/catalog/articulated-pose-estimation-flexible-mixturesofparts/.