2013 Second IAPR Asian Conference on Pattern Recognition

Vehicle Detection in Satellite Images by Parallel Deep convolutional Neural Networks

Xueyun Chen, Shiming Xiang, Cheng-Lin Liu, and Chun-Hong Pan.
National Laboratory of Pattern Recognition
Institute of Automation, Chinese academy of Sciences
Beijing 100190, China
Email: {xueyun.chen, smxiang, liucl, chpang}@nlpr.ia.ac.cn

***Abstract*—Deep convolutional Neural Networks (DNN) is the state-of-the-art machine learning method. It has been used in many recognition tasks including handwritten digits, Chinese words and traffic signs, etc. However, training and test DNN are time-consuming tasks. In practical vehicle detection application, both speed and accuracy are required. So increasing the speeds of DNN while keeping its high accuracy has significant meaning for many recognition and detection applications. We introduce parallel branches into the DNN. The maps of the layers of DNN are divided into several parallel branches, each branch has the same number of maps. There are not direct connections between different branches. Our parallel DNN (PNN) keeps the same structure and dimensions of the DNN, reducing the total number of connections between maps. The more number of branches we divide, the more swift the speed of the PNN is, the conventional DNN becomes a special form of PNN which has only one branch. Experiments on large vehicle database showed that the detection accuracy of PNN dropped slightly with the speed increasing. Even the fastest PNN (10 times faster than DNN), whose branch has only two maps, fully outperformed the traditional methods based on features (such as HOG, LBP). In fact, PNN provides a good solution way for compromising the speed and accuracy requirements in many applications.**

Keywords*—Remote Sensing; Object detection; Deep convolutional Neural Networks;

I. Introduction

Detecting vehicle in high-resolution satellite images is a highly challenging task. Hidden in a tree, sheltered by a building or jammed together in the streets or parks in very close distances, vehicle detection always arouse human interests. Many works have been done, various features and methods have been used [1-8].

Hinz [3] built a hierarchical 3D-model to describe the prominent geometric features of the cars, Chen et al. [7] segmented road by its straight line contours and used SVM to detect vehicle in road. Liang et al. [8] combined the HOG descriptors and the selected Haar features and used Multiple Kernel Learning method to detect vehicle in wide area motion imagery. Zhao et al. [1] showed the boundary of the car body, the boundary of the front windshield, and the shadow were the key features for vehicle detection. Ali et al. [2] detected vehicle by adaboost method based on pose-indexed features and pose estimators. Grabner et al. [6] used boosting method based on Haar wavelets, HOG and LBP. They segmented the image into streets, buildings, trees, etc, discarding vehicle detections that are not present on the streets. Kembhavi et al.[4] detected vehicles in San Francisco using the multi-scales HOG features computed on the color probability maps. They showed HOG outperformed SIFT. SIFT, HOG and LBP are the most popular features in object detection or image classification. SIFT is very similar to HOG, LBP is more suitable for texture feature such as face recognition [20].

Recently, machine learning intelligence has been reported enable to match human performance on recognition task of handwritten digits and traffic signs [9]. The machine is based on deep convolutional neural networks (DNN). Convolutional neural networks originates from the study on cats striate cortex by Hubel and Wiesel [12]. They first proposed the concept of receptive field. Fukushi [13] proposed Neocognitron, a hierarchy model of the multi-layer neural networks. He realized the concept of receptive field. LeCun [14-15] gave the normal form of convolutional neural networks (CNN)(LeNet-5, LeNet7), which has been used in many recognition tasks, include digit recognition, face detection. Rowley [17] detected face using a simple CNN which only had three layers but with three types receptive fields in the first layer. Garcia [16] realized a LeNet-5 structure CNN for face detection. They showed CNN outperformed Adaboost method [21] in CMU and MIT test sets obviously. Compared with CNN, DNN is more deep (6-10 layers) and wide (40-250 maps per layer). Training and test DNN are time-consuming works, the former often needs 1-2 days even with the help of a GPU card. For practical application like object detection in satellite images, speed and accuracy are the same important, we seek to simply DNN, increasing its speed while keep its accuracy in a high level.

Reminded by the LeNet-5 structure, where the 6 maps of the fist max-pooling layer can supply $2^6 - 1 = 127$ different map-connection manners to the higher layer theoretically. LeCun only used 15 manners of them. This implies that selection of the the map-connection manners are variable. In DNN, all the maps in the above convolutional layer has the full map-connections to the lower layer. This produced the maximal number of map-connections, and this is why that DNN is much slow than the traditional CNN. We simplify DNN by introducing the concept of branches, we divided all maps into several parallel branches, each branch has the same number of maps and the map-connections only occur in the internal maps of the branch.

Ciresan et al. [9-11] trained different DNNs by different preprocessed images, using the average of the outputs of all

978-1-4799-2190-4/13 $26.00 © 2013 IEEE
DOI 10.1109/ACPR.2013.33

DNNs. In most situations, such average only increases the accuracy by $0.5 - 1.0\%$ [9], while adding the expense of time-consuming of a lot of times. We used four preprocessed images: gray, gradient, gradient in the image thresholding at 120, gradient in the negative image thresholding at 60. Instead of training four DNNs, we divide the maps of the first convolutional layer into four parts, allocating the same number of maps to the four images respectively, training the same PNN on the four images synchronously. Experiments show that our method increased the accuracy obviously, without more expense of time-consuming. PNN also outperformed the vehicle detection rate (about 70%) of [4] in the same challenging environments of San Francisco city.

II. Architecture of PNN

The layers of PNN can be denoted as: ($Input$, C^1, M^1, $\cdots$, C^{n_l}, M^{n_l}, H^1, $\cdots$, H^{n_h}, $Ouput$). Where C^i and M^i ($i = 1, \cdots, n_l$) are the convolutional and max-pooling layers respectively, they act as the feature extractor. H^l ($l = 1, \cdots, n_h$) are the hidden layers, they and the output layer compose the final Multi-Layer Perceptron (MLP) classifier. For convenience, suppose all convolutional and max-pooling layers have the same number of maps, denote $C^i = (C^i_1, \cdots, C^i_{n_m})$, $M^i = (M^i_1, \cdots, M^i_{n_m})$, where n_m is the number of the mapping layer. Now divide all the maps of the convolutional and max-pooling layers into n_b branches as the following:

$$\begin{cases} Branch = (B_1, \cdots, B_{n_b}) \\ B_j = \left\{ \begin{array}{l} (C^1_{jk}, M^1_{jk}, \cdots, C^{n_l}_{jk}, M^{n_l}_{jk}) : \\ k = 1, \cdots, b_w. \end{array} \right\} \\ j = 1, \cdots, n_b \end{cases} \quad (1)$$

$b_w = \frac{n_m}{n_b}$ is the branch width. $C^l_{jk} = C^l_{(j-1)b_w+k}$, $M^l_{jk} = M^l_{(j-1)b_w+k}$, suppose n_m can be divided by n_b without remainder.

Denote I, D^l, O^l as the definition domains of input, convolutional and max-pooling layers respectively. $R^1 = [0, 255]$ is the gray range, $R = [-1, 1]$ is the kernel function range. Then we have:

$$\begin{cases} Input : I -> R^1 \\ C^l_{jk} : D^l -> R \\ M^l_{jk} : O^l -> R \end{cases} \quad (2)$$

Denote flt^1_{jk} as the filter which connects C^1_{jk} with $input$, flt^l_{jpk} as the filter which connects C^l_{jp} with $M^{(l-1)}_{jk}$. Use $tanh$ as the kernel function, b^1_{jk},b^l_{jp} are their biases:

$$\begin{cases} C^1_{jk}(x,y) = tanh(b^1_{jk} + \sum\limits_{(u,v)\in A^1} (flt^1_{jk}(u,v) \\ \qquad \times Input(x+u, y+v)) \\ C^l_{jp}(x,y) = tanh(b^l_{jp} + \sum\limits_{k=1}^{b_w} \sum\limits_{(u,v)\in A^{l-1}} \\ \qquad (flt^l_{jpk}(u,v) \times M^{l-1}_{jk}(x+u, y+v)) \end{cases} \quad (3)$$

where $(x, y) \in D^l$, $j = 1, \cdots, n_b$, $k = 1, \cdots, b_w$, A^1 and A^l are the definition domains of the filters. $l = 2, \cdots, n_l$.

From (3) we can see that connections only occur between maps of the same branch. There is no connection between branches. The max-pooling layer can be expressed as the following:

$$\begin{cases} T^l_{jk} : O^l -> D^l \\ T^l_{jk}(x,y) = \underset{(x',y')}{argmax} \left\{ \begin{array}{l} C^l_{jk}(x',y') : \\ mx \le x' < mx + m, \\ my \le y' < my + m \end{array} \right\} \\ M^l_{jk}(x,y) = C^l_{jk}(T^l_{jk}(x,y)) \end{cases} \quad (4)$$

T is a map from O^l to D^l, m is a constant positive integer which determine the size of O^l, in this paper, m=2.

We denote $\mathrm{Dim}(*)$ is the number of all nodes in the layer $*$, $Node(*, i)$ as the value of the i-th node of the layer $*$, n_c is the number of classes. Then we define $H^l = (h^l_1, \cdots, h^l_{Dim(H^l)})$,$1 \le l \le n_h$, $Ouput = (out_1, \cdots, out_{n_c})$, w^l_{jk}, w^o_{jk} denote the weights of the hidden layers and output layer respectively, bia^l_j, bia^o_j denote the biases of the hidden layers and output layer respectively. We have:

$$\begin{cases} h^1_j = tanh(bia^1_j + \sum\limits_{k=1}^{Dim(M^{n_l})} w^1_{jk} Node(M^{n_l}, k)) \\ h^l_j = tanh(bia^l_j + \sum\limits_{k=1}^{Dim(H^{l-1})} w^l_{jk} h^{l-1}_k) \\ out_j = tanh(bia^o_j + \sum\limits_{k=1}^{Dim(H^{n_h})} w^o_{jk} h^{n_h}_k) \end{cases} \quad (5)$$

III. Training PNN

We define the training set as: $\{(Input^{(q)}, Label^{(q)}) : q = 1, \cdots, n_{sp}\}$, where q is the sample-index, n_{sp} is the number of all samples in training set. In addition, $Label^{(q)} = (lab^{(q)}_1, \cdots, lab^{(q)}_{n_c})$, $lab^{(q)}_i = 1$, if $Input^{(q)}$ belong to i-th class, otherwise equals to -1. We denote $Output^{(q)} = PNN(W, Input^{(q)})$, where W is the set of all filters, biases and weights in PNN. Then we have:

$$\begin{cases} E = \frac{1}{2} \sum\limits_{q=1}^{n_{sp}} \sum\limits_{k=1}^{n_c} (out^{(q)}_k - lab^{(q)}_k)^2 \\ W = \arg\min\limits_{W}(E) \end{cases} \quad (6)$$

By the steepest descent Method, we have:

$$\Delta W(t+1) = -\varepsilon \frac{\partial E}{\partial W} + \gamma \Delta W(t) - \beta W(t) \quad (7)$$

Where ε= LearnRate, γ= Momentum, β=WeightDecay, LearnRate is a very small value, such as 0.001. Sometimes, Momentum is used to speed convergence, WeightDecay is used to limit the norms of the Weights.

A. Back Propagation

We compute the error of every layer from output layer to the first convolutional layer by the back propagation algorithm, because $\frac{d}{dx}(\tanh(x)) = 1 - (\tanh(x))^2$, the error of the output and hidden layers are:

$$\begin{cases} \delta out^{(q)}_j = (out^{(q)}_j - lab^{(q)}_j)(1 - (out^{(q)}_j)^2) \\ \delta h^{n_h(q)}_j = (\sum\limits_{l=1}^{n_c} w^{out}_{lj} \delta out^{(q)}_l)(1 - (h^{n_h(q)}_j)^2) \\ \delta h^{l(q)}_j = (\sum\limits_{k=1}^{n_c} w^l_{kj} \delta h^{(l+1)(q)}_k)(1 - (h^{l(q)}_j)^2) \end{cases} \quad (8)$$

where $1 \leq l < n_h$. We denote the set $Set^l(x,y)$= $\{(x',y',u,v) : x'+u = x, y'+v = y, (x',y') \in C^{l+1}, (u,v) \in A^{l+1}\}$, then the error of the Max-Pooling layers are:

$$\begin{cases} \delta Node^{(q)}(M^{n_l}, i) = \sum\limits_{k=1}^{Dim(H^1)} w^1_{kji} \times \delta h^{1(q)}_k \\ \delta M^{l(q)}_{jk}(x,y) = \sum\limits_{p=1}^{b_w} \sum\limits_{(x',y',u,v) \in Set^l(x,y)} \\ \delta c^{(l+1)(q)}_{jp}(x',y') flt^{l+1}_{jpk}(u,v)(1-(M^{l(q)}_{jk}(x,y))^2) \\ 1 \leq l < n_l \end{cases} \quad (9)$$

We define the map $F^{l(q)}_{jk} : D^l -> O^l$ as:

$$F^{l(q)}_{jk}(x',y') = \begin{cases} (x,y), if\ T^{l(q)}_{jk}(x,y) = (x',y') \\ (-1,-1), \quad otherwise \end{cases} \quad (10)$$

The error of the convolutional layer is:

$$\delta c^{l(q)}_{jk}(x,y) = \begin{cases} 0, \quad if\ F^{l(q)}_{jk}(x,y) = (-1,-1) \\ \delta M^{l(q)}_{jk}(F^{l(q)}_{jk}(x,y)), \ otherwise \end{cases} \quad (11)$$

where $1 \leq l \leq n_l$, $1 \leq j \leq n_b$, $1 \leq k \leq b_w$.

B. Weights Updating

Suppose momentum and WeightDecay are zero, the weights and biases of the output and hidden layers are updated as:

$$\begin{cases} w^o_{lj}(t+1) = w^o_{lj}(t) - \varepsilon \delta out^{(q)}_l h^{n_h(q)}_j \\ bia^o_l(t+1) = bia^o_l(t) - \varepsilon \delta out^{(q)}_l \\ w^l_{kj}(t+1) = w^l_{kj}(t) - \varepsilon \delta h^{l(q)}_k h^{(l-1)(q)}_j \\ bia^l_k(t+1) = bia^l_k(t) - \varepsilon \delta h^{l(q)}_k \\ w^1_{kj}(t+1) = w^1_{kj}(t) - \varepsilon \delta h^{1(q)}_k Node^{(q)}(M^{n_l}, j) \\ bia^1_k(t+1) = bia^1_k(t) - \varepsilon \delta h^{1(q)}_k \end{cases} \quad (12)$$

where $2 \leq l \leq n_h$, the filters and biases of the convolutional layers are updated as:

$$\begin{cases} flt^l_{jpk}(u,v)(t+1) = flt^l_{jpk}(u,v)(t) - \\ \varepsilon \sum\limits_{(x,y) \in D^l} M^{(l-1)(q)}_{jk}(x+u, y+v) \delta c^{l(q)}_{jp}(x,y) \\ b^l_{jp}(t+1) = b^l_{jp}(t) - \sum\limits_{(x,y) \in D^l} \varepsilon \delta c^{l(q)}_{jp}(x,y) \\ flt^1_{jp}(u,v)(t+1) = flt^1_{jp}(u,v)(t) - \\ \varepsilon \sum\limits_{(x,y) \in D^1} Input^{(q)}(x+u, y+v) \delta c^{1(q)}_{jp}(x,y) \\ b^1_{jp}(t+1) = b^1_p(t) - \varepsilon \sum\limits_{(x,y) \in D^1} \delta c^{1(q)}_{jp}(x,y) \end{cases} \quad (13)$$

where $2 \leq l \leq n_l$, $1 \leq q \leq n_{sp}$. In practical training, the samples are often input in batch, the batch size is in the range [10,100], and the weights update once after a batch input.

IV. Implementation Detail

Vehicles can parked in any spatial place, they have colorful appearances, method based on gray is not suitable for vehicle locating. We rely on the gradients to locate vehicle. The normal gradient is the maximal norm of the gradients in RGB channel as computed in [18], In order to enhance the borders of the black and white vehicles, we computing gradients on the two thresholding images as shown in Figure 1.

A. Object Locating

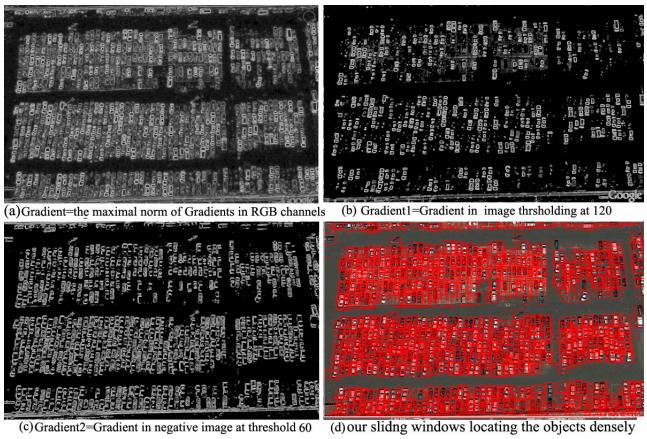


Fig. 1. Examples of locating vehicles in a large park. We compute three type gradients as (a), (b) and (c) respectively, locating the objects in the three images respectively, (d) the set of the location windows, there are 2918 location widows covering 544 vehicles, 99.2% vehicles are located correctly.

In Figure 1 (a), the border of the black vehicles are too dim to be locate correctly. In (c), all the borders of the dim black vehicles are enhanced. In (b), the borders of white vehicles are also enhanced, the backgrounds are flted, this helps to detect white vehicles under the trees. The final location windows is shown in (d). We locate the objects by Algorithm 1:

Algorithm 1 *Object Locating*

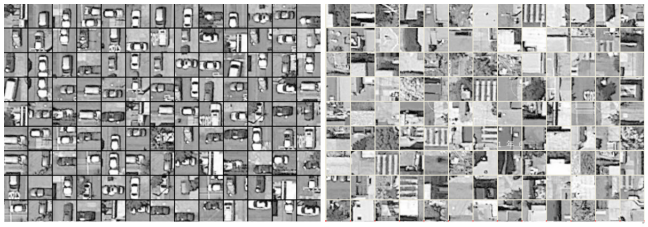
Input: The three gradient images, sliding window size, sliding step.
Output: All location windows.

1: On each gradient image, generate the sliding window grid to cover the whole image.
2: For each sliding window W_p at position $p = (x_0, y_0)$, compute the geometric center $p1 = (x_1, y_1)$ on W_p, center the W_p on (x_1, y_1), denote it as W_{p1}.
3: Enlarge the size of W_{p1} twice, compute the new geometric center $p2$ on the enlarged window. Center W_{p1} on the new center $p2$.
4: Output all location windows on the three gradient images.

The sliding window size is 32×32, the sliding step is 16. Some repetitive windows are filtered by a small distance limit (5 pixel). Our database has 63 images, 1368×972 size, 6887 vehicles at all. Our method generates 197513 location windows, 3135 windows per image, 99.7% vehicles are located correctly. To achieve the same locating precision, the normal sliding window method needs 10400 sliding windows per image, our methods is more efficient in searching.

In order to get rotation-invariant and scale-invariant neural networks, we rotate every location window 11 times by: 0^0, 4.5^0, 9^0, $\cdots$, 45^0, then shrink or enlarge the non-rotating images into multi-scalings: 0.8,0.9,1.0,1.1,1.2,1.3. For each location window, we get four preprocessed images: Gray, Gradient, Gradient1 and Gradient2 (see the (a), (b), (c) of Fig. 1), these preprocessed images are normalized into 48×48 size and [0,255] gray range. We store all these rotated, shrunk or enlarged preprocessed images in our database.

Fig. 2 shows some samples in our training database.



(a) positive samples (b) negative samples

Fig. 2. Partial samples (rotation angle=0) from one image in training set

B. Implementation of PNN

Fig. 3 shows the structure of PNN when using only Gray input. It has 9-layer: 48x48-80@C42x42,7-80@M21x21,2-80@C18x18,4-80@M9x9,2-80@C6x6,4-80@M3x3,2-300N-2N. It means: 48x48 input, a convolutional layer with 80 maps, 42x42 size and 7x7 filters, a max-pooling layer with 80 maps, 21x21 size and 2x2 fields, $\cdots$, $\cdots$, a max-pooling layer with 80 maps, 3x3 size and 2x2 fields, a full connected hidden layer with 300 nodes, an output layer with 2 nodes.

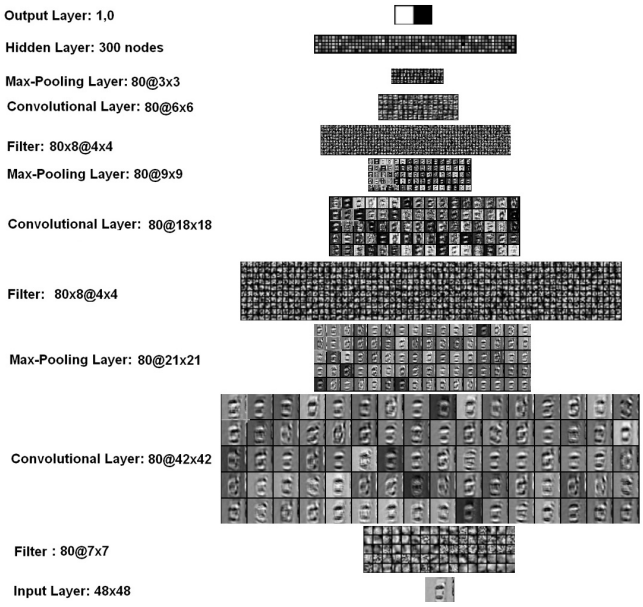


Fig. 3. The Structure of PNN (b_w=8), input only Gray image

We wrote the PNN codes according to the formulas (3-13), we trained PNN on the GPU card, initial weights were set by a uniform random distribution in the range [-0.05, 0.05], all initial biases were set to zero. LearnRate=0.001, Momentum=0,WeightDecay=0, batch size=50. Training ended when the validation error was near to zero. The samples with the same scaling and rotation angle were trained in one epoch, we changed scalings or rotation angles in the store sequence when a new epoch began.

V. Experiment

Our database include 63 images, 6887 vehicles, 197513 samples from google earth at San Francisco city. 31 images, 3874 vehicles, 94858 samples are used as training set, other 32 images and 102655 samples are used as test set. We denote False Alarm Rate (FAR) and Detection Rate (DR) as:

$$\begin{cases} \text{FAR} = \dfrac{\text{number of false alarms}}{\text{number of vehicles}} \times 100\% \\ \text{DR} = \dfrac{\text{number of detected vehicles}}{\text{number of vehicles}} \times 100\% \end{cases} \tag{14}$$

To be fair and objective, some overlapped False Alarms are fused into one alarm.

TABLE I. FAR of PNN (Input only Gray Image)

b_w	test(s)	train(h)	Detection Rate				
			95%	90%	85%	80%	75%
2	35.21	23.10	55.7%	36.4%	23.5%	17.2%	12.9%
4	49.05	28.52	54.0%	34.6%	22.4%	16.5%	12.1%
8	56.43	34.87	53.4%	32.5%	21.3%	15.9%	11.4%
10	75.20	37.58	52.8%	30.4%	20.8%	14.9%	10.7%
16	110.7	46.80	50.9%	28.5%	18.6%	13.6%	9.35%
20	144.6	54.30	47.3%	26.7%	17.3%	12.3%	8.75%
40	237.3	75.23	44.6%	25.6%	16.0%	11.2%	8.25%
80	364.5	102.9	41.2%	23.0%	14.4%	10.5%	7.83%

Table 1 shows the influence of b_w. If b_w=1, the PNN training will not converge. In 85% detection rate, from b_w=2 to b_w=80, the fastest speed is about 10 times of the slowest speed, with FAR increases no more than 10%. The test and training time unit is second and hour on GPU cards.

TABLE II. FAR of PNN (b_w=80)

Input Data	Detection Rate				
	95%	90%	85%	80%	75%
Gray	41.2%	23.0%	14.4%	10.5%	7.83%
Gradient	43.5%	24.8%	15.7%	11.3%	8.20%
Gradient1	48.0%	26.5%	17.5%	12.8%	9.45%
Gradient2	44.6%	24.7%	15.9%	11.4%	8.26%

Table 2 shows the results of different inputs. In contrast to our expectation, the gradient input does not perform better than gray input, this is because the gradient image lost many details information of the object texture. Gradient2 performs better than Gradient1. Fig. 1 shows that Gradient2 contains more information than Gradient1.

TABLE III. FAR of Four methods (Input multi-images)

Method	Detection Rate				
	95%	90%	85%	80%	75%
PNN(b_w=80)	39.63%	21.54%	13.84 %	10.08%	7.52%
PNN(b_w=40)	43.01%	24.03%	14.82%	10.67%	7.94%
HOG+SVM	65.21%	40.21%	28.71%	21.81%	15.42%
LBP+SVM	74.35%	46.82%	32.20%	24.72%	17.37%

In Table 3, we input multi-images: Gray, Gradient, Gradient1 and Gradient2. We divided the 80 maps of the first convolutional layer of PNN into four equal parts, allocate 20 maps for each image. Comparing Table 2 and Table 3, we can see the result of multi-images is much better than any single image. It shows that the multi-images are complementary to each other, they have produced good resonance effect.

HOG feature is computed here as [18], where Gaussian smoother parameter $\sigma = 2$, derivative mask is [-1,0,1], spacial orientation bins is 9, cell size is 8x8, each overlapped block include four cells. LBP feature is computed as [19], where P=8, R=2, using 58 uniform patterns and 1 nonuniform pattern. The detection window is divided into $1 \times 1 + 2 \times 2 + 3 \times 3 + 4 \times 4 + 5 \times 5 = 55$ blocks. We used rbf kernel in SVM, all other parameters are optimized.

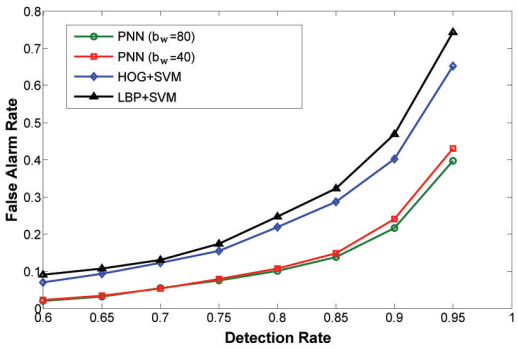


Fig. 4. False Alarm Rates of four methods on our vehicle test set.

Fig. 4 shows the difference between b_w=80 and b_w=40 are very subtle. Fig. 5 shows cars in different orientations

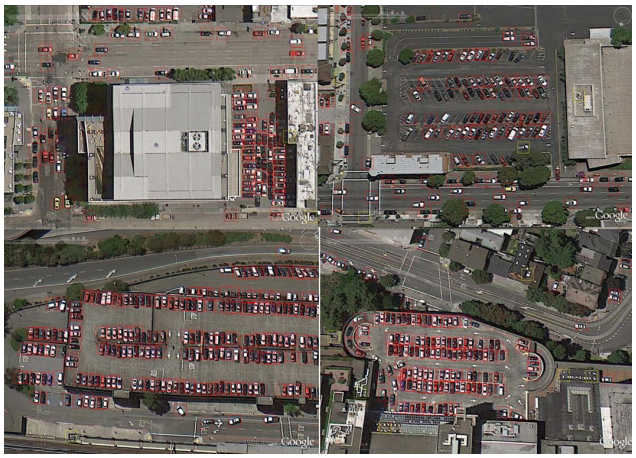


Fig. 5. Some detection results by PNN (b_w=40) in San Francisco city. Red frames are the right detection, the yellow frames are the false alarm.

and places are detected correctly, it shows that our PNN has got scale-invariant and rotation-invariant power by training on samples with different scaling and rogation angles. However some black cars are missed, it implies that illumination and color play important roles in vehicle detection.

VI. Conclusion

Practical applications like vehicle detection in satellite images often have high requirements in speed and accuracy. We proposed parallel deep convolutional neural network (PNN), which divides all maps of DNN into different branches, no direct map-connections between branches. The conventional DNN becomes a special form of PNN which has only one branch. Experiments on vehicle detection showed PNN can be 10 times faster than DNN, while keeping a good accuracy. All PNNs outperformed the traditional methods based on features. They also outperformed the vehicle detection result of [4] in the same challenging environments of San Francisco city. Via adjusting the branch width, PNN provides a good solution for compromising both speed and accuracy requirements in practical applications.

Acknowledgment

This work was supported in part by the National Basic Research Program of China (973 Program) Grant 2012CB316300 and the Strategic Priority Research Program of the Chinese Academy of Sciences (Grant XDA06030300).

References

[1] T. Zhao R. Nevatia, Car Detection in Low Resolution Aerial Images, Proc. ICCV , vol.1, pp. 710-717, 2001.

[2] K. Ali, F. Fleuret, D. Hasler, and P. Fua, A Real-Time Deformable Detector, IEEE Trans. PAMI , 34(2):225-239, February 2012.

[3] S. Hinz, Detection and Counting of Cars in Aerial Images, Proc. ICIP, 2003.

[4] A. Kembhavi, D. Harwood, L. S. Davis, Vehicle Detection Using Partial Least Squares, IEEE Trans. PAMI , 33(6):1250-1265 June 2011.

[5] L. Eikvil, L.Aurdal and H. Koren, Classification-based Vehicle Detection in High-resolution Satellite Images, Journal of Photogrammetry and Remote Sensing, 64(1):65-72, January 2009.

[6] H. Grabner, T. Nguyen, B. Gruber, and H. Bischof, On-Line Boosting-Based Car Detection from Aerial Images, ISPRS J. Photogrammetry and Remote Sensing, 63(3):382-396, 2008.

[7] L. Chen , Z. Jiang, J. Yang, Y. Ma, A Coarse-to-fine Approach for Vehicles Detection from Aerial Images, International Conference on Computer Vision in Remote Sensing (CVRS), pp. 221-225, 2012.

[8] P. Liang, G. Teodoro, H. Ling, E. Blasch, G. Chen, L. Bai, Multiple Kernel Learning for vehicle detection in wide area motion imagery, 15th International Conference on Information Fusion (FUSION), pp. 1629-1636, 2012.

[9] D. C. Ciresan, U. Meier and J. Schmidhuber, Multi-column Deep Neural Networks for Image Classification , Proc. CVPR 2012.

[10] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber, Convolutional Neural Network Committees for Hand-written Character classification, In International Conference on Document Analysis and Recognition, pp. 1250-1254, 2011.

[11] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella,and J. Schmidhuber, Multi-Column Deep Neural Network for Traffic Sign Classification, Neural Networks, January 23, 2012.

[12] D. H. Wiesel and T. N. Hubel, Receptive fields of single neurones in the cats striate cortex , J. Physiol., 148:574-591, 1959. 2

[13] K. Fukushima, Neocognitron: A self-organizing neural network for a mechanism of pattern recognition unaffected by shift in position , Biological Cybernetics, 36(4):193-202, 1980.

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition , Proceedings of the IEEE, 86(11):2278-2324, November 1998.

[15] Y. LeCun, F.-J. Huang, and L. Bottou, Learning methods for generic object recognition with invariance to pose and lighting, Proc. CVPR, 2004.

[16] C. Garcia and M. Delakis, Convolutional Face Finder:A Neural Architecture for Fast and Robust Face Detection, IEEE Trans. PAMI, 26(11):1408-1423, November 2004.

[17] H. A. Rowley, S. Baluja, and T. Kanade, Neural Network-Based Face Detection, IEEE Trans. PAMI, 20(1):23-38, January 1998

[18] N. Dalal and B. Triggs, Histograms of oriented gradients for human detection Proc. CVPR ,vol. 1, pp. 886-893, 2005.

[19] T. Ojala, M. Pietikainen, T. Maenpaa, Multiresolution Gray Scale and Rotation Invariant Texture Classification with Local Binary Patterns, IEEE Trans. PAMI, 24(7):971-987, July 2002.

[20] C. Huang, S. Zhu, K. Yu, Large Scale Strongly Supervised Ensemble Metric Learning,with Applications to Face Verification and Retrieval http://arxiv.org/abs/1212.6094, 2011.

[21] P. Viola and M. Jones, Rapid Object Detection Using a Boosted Cascade of Simple Features, Proc. Intl Conf. Computer Vision and Pattern Recognition, vol. 1, pp. 511-518, 2001.