SPFCN: Select and Prune the Fully Convolutional Networks for Real-time Parking Slot Detection

Zhuoping Yu[1,2], Zhong Gao[1,2], Hansheng Chen[1,2], Yuyao Huang[1,2,*]

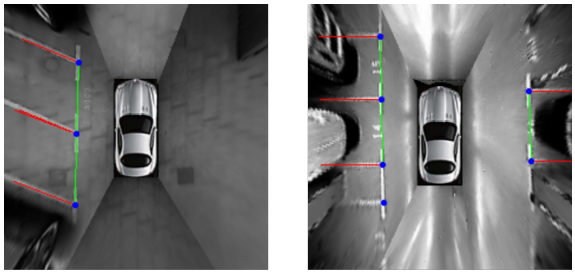
***Abstract*— For passenger cars equipped with automatic parking function, convolutional neural networks are employed to detect parking slots on the panoramic surround view, which is an overhead image synthesized by four calibrated fish-eye images, [1] [2]. The accuracy is obtained at the price of low speed or expensive computation equipments, which are sensitive for many car manufacturers. In this paper, the same accuracy is challenged by the proposed parking slot detector, which leverages deep convolutional networks for the faster speed and smaller model while keep the accuracy by simultaneously training and pruning it. To achieve the optimal trade-off, we developed a strategy to select the best receptive fields and prune the redundant channels automatically during training. The proposed model is capable of jointly detecting corners and line features of parking slots while running efficiently in real time on average CPU. Even without any specific computing devices, the model outperforms existing counterparts, at a frame rate of about 30 FPS on a 2.3 GHz CPU core, getting parking slot corner localization error of 1.51±2.14 cm (std. err.) and slot detection accuracy of 98%, generally satisfying the requirements in both speed and accuracy on on-board mobile terminals. An example of the detection result is shown in Fig. 1.**

I. INTRODUCTION

With the development of ADAS and autonomous vehicles, automatic parking assist system has become an intensive research topic. A typical automatic parking assist system works by first specifying a target location, namely finding an available parking slot for the vehicle. There is a large literature on various solutions to this problem. The proposed solutions can be roughly divided into two categories: one is infrastructure based approach and the other is on-board sensor based approach. Infrastructure based approaches typically use pre-built maps and sensors which adds to the cost and cannot be fully implemented in a short time. On-board sensor based approaches use only sensors mounted on vehicles to detect available parking slots, which is more flexible and can be further divided into free space based and parking slot marking based methods[3].

The free space based method specifies the target location by identifying the free space between two adjacent vehicles. This is a widely used method that can be implemented using various sensors such as different kinds of radars and cameras, etc. However, the free space based method has the inherent disadvantage[1] that it can not find free space when there are no adjacent vehicles and its accuracy depends on the positions and poses of the vehicles in the adjacent parking slots.

[1] School of Automotive Studies, Tongji University, Shanghai, China
[2] Institute of Intelligent Vehicles, Tongji University, Shanghai, China
* Corresponding author. E-mail: huangyuyao@tongji.edu.cn



(a) non-vertical (b) multi-slots

Fig. 1. An example of parking slot detection result, with corners marked as blue points, entry lines marked green and seperating lines marked red. The synthetic overhead view is composed by images from four calibrated, undistorted and homographic transformated fish-eye cameras mounted on the vehicle, according to [3]. The artifacts and misalignments caused by calibration error can be further reduced by additional transformation [4] and stitching algorithms [5], but we focus on the detection pipeline.

The parking slot is a group of line segments drawn on the ground to indicate the effective parking area. Unlike the free space based method, the parking slot marking based method acquires an image through a camera mounted on a vehicle to find a slot mark based on the visual identification, so its performance does not depend on the presence of vehicles in the adjacent slots.

The detection methods based on location markers can be divided into two categories: line-based and angle-based[1]. In this paper, a pure visual method is designed to find the parking slots, in which both line features and point features will be detected to improve accuracy. Our method differs from other parking slot detectors in three major contributions as concluded below:

- First, we introduce the hourglass structure[6] into parking slot detection and discuss the network with/without stacked hourglass and intermediate supervisions.
- Second, we design a basic module named SP Module (select and prune module), which can select the most suitable receptive field of the convolution, and at the same time, prune the redundant channels automatically.
- Third, we use both point features and line features to jointly infer the parking slots location, which improves the accuracy while compressing the image size and speeding up the processing.

II. RELATED WORKS

Parking Slot Detection: One of the first works to use traditional machine learning method for parking slot detection was presented by Xu et al. [7]. Later implemen-

tations of traditional methods include using probabilistic Hough transform[8], Radon transform[9] to detect slot lines or using Harris detector[10], [11], Ada-Boost[3] to detect slot corners. The results of these methods largely depend on whether the feature design and subsequent logical inference are reasonable, and they are also sensitive to the size or direction of the input images. Also the complications of shadows and ground marking poses difficulty in solving the parking slot detection problems with traditional methods.

As for the deep-learning-based detection methods, for example, Zhang et al.[1] uses YOLO V2[12] to locate the parking slot corners. YOLO is an end-to-end, fast object detection network designed not exclusively for point feature detection but corners are only represented by only a few pixels in the images. Thus it is not a perfect option for pixel-level classification. The parking slot corner, however, is a small feature represented by a few pixels in a small image. Besides, unlike GPU implementation, running YOLO in real time on CPU can be a serious problem due to its massive number of parameters. Today's production cars are often fitted with only CPUs or even MCUs. So there still lies potential in the performance of the backbone itself.

Hourglass-like Network Structure: In this paper, hourglass structure[6] is selected as our backbone. This architecture is specifically designed for key-point detection and has been widely used in many key-point detection networks, e.g. human joint detection.

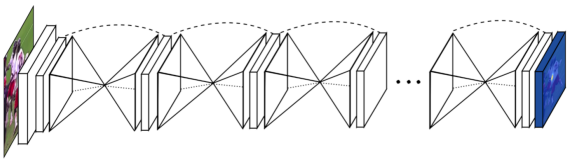


Fig. 2. Stacked hourglass network[6]

The architecture is shown in Fig. 2. It has the advantages of flexibility in stacking the hourglasses to adjust the number of the parameters and the speed of network. The input image is first down-sampled, the possible regions of the key-points and lines are found on the smaller feature map, and the feature size is restored by up-sampling. The key areas detected by the previous layer are superimposed on the features of skip, thereby it can increase the weight of the key areas.

Efficient Convolution Module Design: Regarding the basic convolution modules, there is an increasing need in efficient model design, which involves network acceleration and lightweight. SqueezeNet[13] implemented AlexNet[14]-level accuracy on ImageNet with a 50-fold reduction in parameters. MobileNet[15], [16], [17] and ShuffleNet[18], [19] use depth-wise separable convolution[20] to compress the parameters and FLOPs of the network while ensuring accuracy. Based on ResNet[21] and DenseNet[22], PeleeNet[23] uses a two-way dense layer and a stem block to optimize its performance.

Li et al.[24] realized that the convolution kernel size of visual neurons is rarely considered when constructing the CNN. They propose a dynamic selection mechanism to adaptively adjust the weights of kernels of different sizes when fusing multiple branches containing information of different spatial scales.

Inspired by that, we design a new convolution module which can adaptively select the convolution kernel size and prune the redundant channels. We call it the SP Module and substitute it for the residual modules[21] in the original stacked hourglass network.

III. METHOD

We developed an efficient and practical solution for simultaneous detection of corners and marking lines of the parking slots. The general idea is using FCN (Fully Convolutional Networks) to accomplish all the jobs, without the help of FC (Fully Connected) layers. The classification, regression, detection, and segmentation tasks generally can be modeled as a softmax heatmap generator followed by a certain type of proposal (e.g. Region Proposal Network[25], etc.) and pooling (e.g. ROI Pooling[25], Line Pooling[26], Average Pooling, etc.), and finally thresholding. We then develop a scalable and tailorable strategies for training and pruning simple convolution channels by selecting from the candidate kernels. The trained networks with simple post-processing logic can show good efficiency and flexibility to be generalized to other problems by following the similar solution paradigm, the solution for parking slot detection in this paper is an example of such idea.

A. Network Structure

Among all the FCNs, we choose the Stacked Hourglass Network[6] as the basic architecture, for its scalability to different layers of stacks. The preliminary network we designed, as shown in Fig. 3, stacked two sets of hourglass structures.

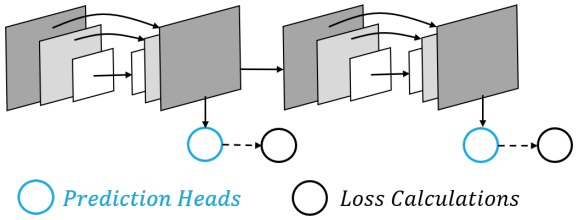
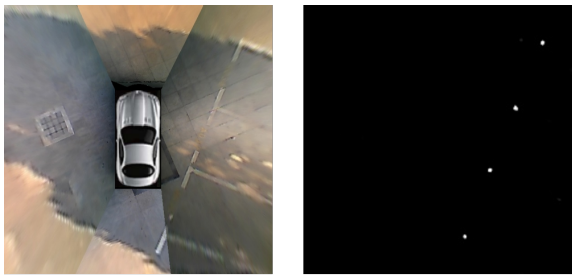


Fig. 3. Our architecture with two stacked hourglasses

The prediction heads of the parking slots detection task produces heatmaps of slots' corners, entrance lines and seperating lines, as shown in Fig. 4. The final layers, as well as the intermediate supervision layers, supervise the layers by softmax losses through those heads. The existence of the intermediate supervision layers can assist the back propagation of the loss.

B. SP Module

We propose the Select-Prune Module, which is able to automatically SELECT the best convolution kernel candidates,



(a) input image (b) prediction of the mask

(c) prediction of the entry (d) prediction of the side

Fig. 4. Predicted heatmaps of the Heads

and PRUNE the least contributional channels in the selected kernel. It works as a substitute for residual block[21] in the original stacked hourglass architecture.

Select Module: As we know, only kernels with "correct" receptive field can respond to the "correct" feature, making the receptive field a critical hyper-parameter to tune. Thanks to dilated convolution[27], using multiple receptive fields in one convolutional layer can be done by composing convolutional kernels with different dilation value. Furthermore, there should be a most efficient combination of dilation values in each layer, which means the strategy to assemble kernels of all the different dilations (like in GoogLeNet[28]) is wasteful. So we designed the Select Module to select the convolution kernel with the best receptive fields in a convolutional layer.

As shown in the Fig. 5, we set the same input for a group of kernel candidates of different dilation values during select mode, and then evaluate the contribution of each candidate using two layers of full connection (these FC layers are only used during training), the contribution is applied as a normalized weights on the results of the output tensors, and trained in an end-to-end manner. Even the weights are soft and both kernels contribute to the final results during training phase, the weaker kernel may be optimized out during pruning phase and finally we left only the best one for further training. Although in this example there are only two dilation values, the Select Module can be easily applied to the case with multiple dilation values.

For an input feature $X \in \mathbb{R}^{N\times C'\times H'\times W'}$, we denote the two convolution kernels as $F_1 : X \to U_1 \in \mathbb{R}^{N\times C\times H\times W}$ and $F_2 : X \to U_2 \in \mathbb{R}^{N\times C\times H\times W}$, with a dilation value of 1 and 2 respectively, which means they are just like a basic convolutional layer with a kernel size of 3 and 5 in terms of receptive fields. The input of the fully connected *Contribution Evaluation Networks (CEN)* of the l^{th} convolutional layer is the flattened parameters of the i^{th} candidate convolution kernels in the same layer, denoted as W_i^l. We have:

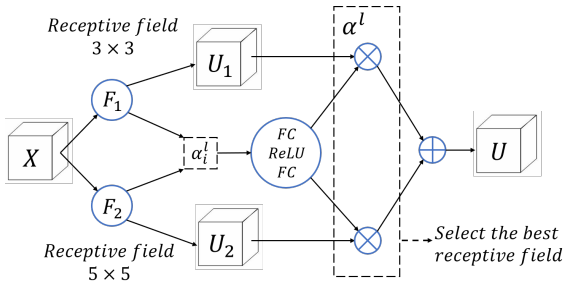


Fig. 5. Diagram of Select Module

$$\begin{aligned}\alpha_i^l &= \mathrm{CEN}(W_i^l) \\ &= Softmax(FC(ReLU(FC(W_i^l))))\end{aligned} \tag{1}$$

and,

$$U_{output}^l = \sum_i \alpha_i^l U_i^l \tag{2}$$

in which α_i is the summing weights of each kernel's output tensor.

In order to highlight the optimal kernel, we need to obtain sparse weights for different kernels. Considering that $\sum_i \alpha_i = 1$, we designed the following regularization term to do that:

$$L_{\mathrm{CEN}}^l = 1 \Big/ \sqrt{\sum_i \alpha_i^{l2}} \tag{3}$$

It can be proved (according to the triangle inequality) that, when one of the $\alpha_i = 1$ and others become zero, L_{CEN}^l obtain its minimum value 1. With such regularization, vector $[\alpha_i]$ tends to be sparser as driven by the loss function.

Prune Module: The network is being trained and pruned iteratively[29], splitting the learning procedure into interleaved training phases and pruning phases. The network may go through several epochs during one training phase, and during the following pruning phase, the candidate convolutional channels are traversed, and those that contribute less than a certain level are pruned. The contribution score is calculated by the norm of the weights, as in [30]. In case of that all the channels play reasonable roles and the network stops pruning too early, we decide to constantly prune channels of the "global" minimum contribution among all layers in the network each time, thus the network will continue pruning to achieve every configuration of the trade-offs. To make contributions in different layers comparable, we use softmax to normalize the contribution of different channels in the same layer before compared with channels from other layers.

In short, there are two types of channel that should be pruned:

- The channel's contribution is too small in its layer, for example, less than 1%;
- The channel's contribution is the least in the whole model.

For a simple convolutional network, using the above pruning strategy is sufficient. But since the hourglass architecture used in this paper has skip connection, special considerations are needed to keep the manifold consistent. The approach is shown in Fig. 6.

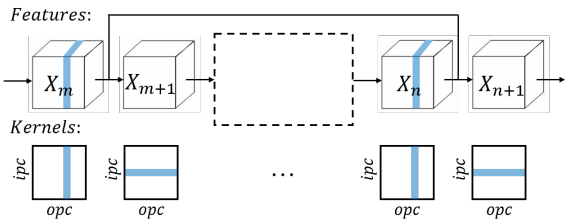


Fig. 6. Pruning in hourglass, blue parts represent pruned channels

Pruning the n-th convolution output affects not only the input of the (n+1)-th layer, but also the input and output channels of the layer from skip connection. Like the method mentioned in [31], the convolution kernels connected by skip connections are grouped into the same group, and the score is the sum of the pruning scores of each convolution kernel in the group.

C. Slots Inference

The network's input is a bird eye view image, and the output is the estimated slot corners and marking lines (including entry lines and seperating lines).

The surround-view image sequence could be synthesized in real-time during inference from outputs of multiple wide-angle cameras mounted on the vehicle. Four fish-eye cameras are used, and the surround-view is the composite view of them: the front, left, rear and right view. A lookup table is used to calculated the relations between the fish-eye image and the bird's-eye-view. The transformation matrix from the bird's eye view image coordinate system to the world coordinate system originating at vehicle center is calibrated beforehand. Once a parking slot is detected in the bird's eye view, its world coordinates can be calculated. Further details about bird's eye view image synthesis can be found in [3].

We first detect the corners of slots by finding local maximums on the corners' heatmap. Then, pairs of corners within reasonable distance are selected as the proposal for entrance lines. Next, we filter out the false positives by averaging out the scores along the corners' connection on the heat map of entrance lines, thus semantically determine which pairs of corners do belong to a same slot. Pairs that exceeds the threshold will further be used to solve the direction of the parking space. Since the standard parking slots generally possess entrance line and seperating lines that are at 90°, 45° or 135°, again the average scores along proposed seperating lines on those direction can be verified for testing the existence of the lines. If the seperating lines are not detected, it is considered that this set of corner points does not generate parking spaces. The filtered lines and corners are finally assembled together as the detected parking slots.

IV. EXPERIMENTS

A. Dataset

We train and test on the challenging DeepPS[3] dataset, which includes various scenarios&weather (indoor, outdoor, rainy, shadow), time (day, night) and parking slot types (perpendicular, parallel and slanted). There are 9527 training images and 2138 validating images, all in bird's-eye view.Coordinates of the corners, types and the direction of the slots are labelled by hand. The dataset can be found at https://cslinzhang.github.io/deepps/.

B. Metrics

Here we follow the definition of Zhang et al.[1]: for a ground-truth marking-point g_i, if there is a detected marking point d_i satisfying $|g_i - d_i| \leq \delta$, where δ is a predefined threshold, we deem that g_i is correctly detected and d_i is a true positive. In this paper, we evaluate the performance of the model by *accuracy* which refers to the accuracy of slot corner detection with the tolerance of 6 cm by default (approximately 1.5 pixels in the image), instead of 16 cm which was set by Zhang et al.[3] [1].

C. Training Details

The input images are 224×224 bird's eye view gray-scale images, with corner points and slot line labels. The Adam[32] optimization algorithm is adopted and the learning rate is set to 1e-3.

The performance of an object detection problem is often affected by extreme class imbalance. For the two-class problem in this paper, specifically, the problem is significant since most pixels are not the desired corners or lines, contributing to most of the loss. Therefore, Focal Loss[33] is used instead of the standard Cross Entropy Loss such that the imbalance problem is solved to some extent.

$$L_{heatmaps} = \begin{cases} -(1-p)^2 log(p), & \text{if around}(y == 1) \\ -(1-y)^4 p^2 log(1-p), & \text{otherwise} \end{cases} \tag{4}$$

The total loss is listed as below:

$$\begin{aligned} L_{total} = & L_{heatmaps}^{corners} + L_{heatmaps}^{entry lines} + \lambda_{sl} L_{heatmaps}^{side lines} \\ & + \lambda_{\text{CEN}} L_{\text{CEN}} + \lambda_{L1} L_1 \end{aligned}, \tag{5}$$

where a suppression factor for losses of the less sensitive seperating lines is set to of $\lambda_{sl} = 0.1$, the regularization term for Contribution Evaluation Networks is scaled with $\lambda_{\text{CEN}} = 10$ since its small upper bound, and the L1 regularization for getting sparser kernel weights is set to $\lambda_{L1} = 0.05$.

D. Structure Comparison

To determine the number of hourglasses to be stacked, we first tried stacking two hourglasses, each composed of five successive pooling layers, and then five successive upsampling layers, with the weight of intermediate supervision set to 0.1. The result is shown in Fig. 7.

Fig. 7 illustrates the accuracy of corner detection, where *Inter* refers to the intermediate supervision, which is the

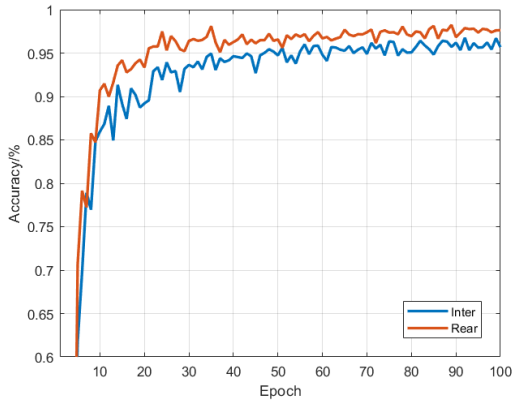


Fig. 7. Accuracy of intermediate and final supervision during training

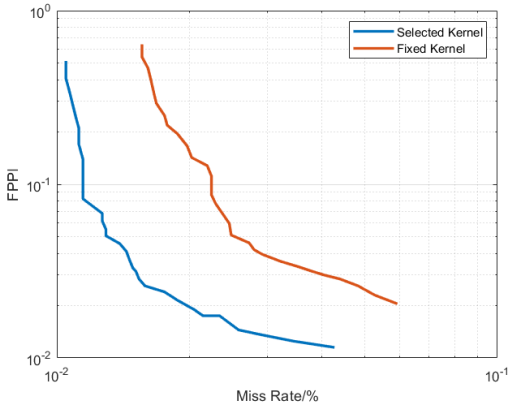


Fig. 8. Comparison of selected and fixed kernel networks

output of the first hourglass structure, and *Rear* is the final result after two hourglasses. So we arrived at the following conclusions:

- Single-hourglass network can well achieve fast detection of parking slot corners and lines in the images;
- Multi-hourglass network can improve the accuracy and converge faster.

A single-hourglass network is sufficient for the task in this paper. Considering that we want to perform real-time detection on CPU, we finally opt for the single-hourglass network.

E. Selecting Kernels

Since the network needs to be pruned during training, the initial number of channels is increased to 64, thereby providing a larger search space.

Different convolution kernel sizes collect information in different receptive fields. We have chosen different convolution operations for different feature sizes. The same receptive field of 5×5 can be achieved by either using standard convolution with kernel size of 5 or using dilated convolution with kernel size of 3 and dilation value of 2. When the feature size is less than 28×28, dilated convolution is used for better efficiency. When the feature size is greater than 28×28, standard convolution is used because the torch implementation (version 1.2.0)[34] of standard convolution is optimized, which makes it faster at larger feature sizes.

When introducing the dilated convolution as a substitute for conventional convolution, we fix the kernel size to 3 and change the dilation value to achieve different sizes of receptive fields. BatchNorm and ReLU are also added after the convolution except for the final layer.

We have retrained two networks under the same conditions, one using the learned best kernel type and the other using fixed 3×3 standard convolution. The accuracy comparison is shown in Fig. 8.

Obviously, selected receptive fields achieve better performance than fixed convolution. However, it should be pointed out that due to the existence of regularization, the select module requires more training time to converge.

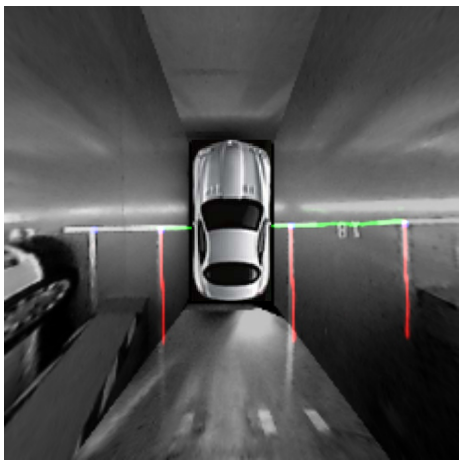


Fig. 9. Detection result using the selected kernels, the images' misalignment are caused by stitching errors

Fig. 9 shows the detection result of the model with selected kernels, where the green line is the parking slot entry line and the red line is the seperating line with fixed length. It can be seen that the model can detect the parking slots accurately, including the missing line covered by the car body.

F. Pruning Channels

Fig. 10 shows the model accuracy during the learning (training & pruning) process. Accuracy decreases more and more rapidly when prune ratio increases.

To find a balance between network speed and accuracy, we chose three different pruning configurations for fine-tuning, whose processing time is approximately 30 ms, 22 ms and 17 ms respectively. The accuracy of the model without pruning is added as a baseline (with process time of 52 ms). As is shown in Fig. 11, after 50 epochs of fine-tuning, accuracy of the three pruned networks is generally restored to the level before pruning. It can be noticed that the accuracy of the lighter network goes slightly down in the later training epochs and we have the best parameters at about the 34th epoch.

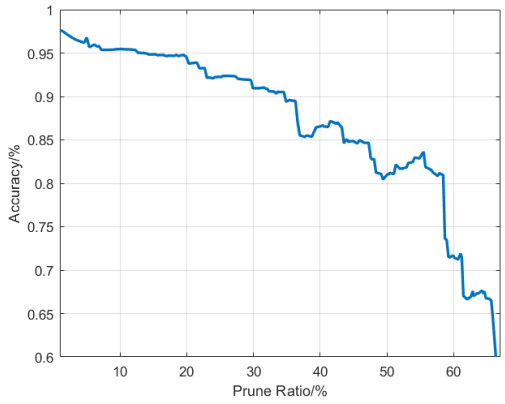


Fig. 10. Location Accuracy of Key Points in the pruning process

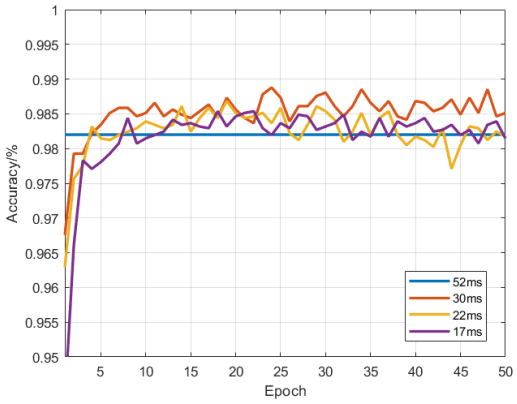


Fig. 11. Accuracy in fine-tuning process

G. Evaluation and Comparison with Other Methods

To show that modeling the slot corner detection problem as a key-point detection problem is better than modeling it as an object detection task in [3] [1], we calculated the localization error of the parking slot markings on the validate set and compared it with the results of several object detection networks (other results come from [1]), all error is converted to centimeters for comparison and the result is shown in Table I.

TABLE I
LOCALIZATION ERROR OF DIFFERENT METHODS (OTHER RESULTS COME FROM [1])

Method	Localization Error (Unit: cm)
Faster R-CNN[25]	6.12±3.87
SSD[35]	2.52±1.95
DeepPS (YOLO V2-based)[1]	2.58±1.75
Our Method	1.51±2.14

And for the overall slot detection, the comparison of the results on the same validation set can be seen in Table II. Compared with the previous methods using AdaBoost or YOLO v2, the method used in this paper has gained a lot of speed improvements while sacrificing a bit of accuracy, which makes it easier to arrange into storage space and computing power In weaker environments, such as on-board equipment.

TABLE II
ACCURACY OF DIFFERENT METHODS

Method	Parameter size/MB	Precision	Recall
PSD_L[3] (16cm)	8.38 MB	98.55%	84.64%
DeepPS[1] (16cm)	255 MB	**99.54%**	**98.89%**
Our Method (6cm)	**2.39 MB**	98.01%	97.31%
Our Method (16cm)	**2.39 MB**	98.26%	97.56%

The frame rate of the model with an input image size of 224×224 is tested on both 2.3 GHz CPU and GTX 1080 Ti GPU, as shown in Table III.

TABLE III
SPEED ON DIFFERENT DEVICES (UNIT: FRAMES PER SECOND)

	Network Only	Full Model (w/ post-processing)
CPU@2.3GHz	31.7	30.9
GTX 1081Ti@batch=1	320.1	145.7
GTX 1081Ti@batch=256	998.5	276.4

Here the model includes the network and the post-processing module for slots inference, which generates a list of slots based on the detected corners and lines. The post-processing module works on CPU only, which limits the overall speed, especially when multiple batches are processed in parallel. As a result, the model frame rate is significantly lower than the network frame rate on GPU.

Fig. 12 visualizes four examples of the detection results, in which the corners are marked with blue points. Also the pruned network can recognize parking slots with non-right angled seperating lines and entry lines. The model can also identify corners that doesn't belong to any slot, which is shown in the figure but excluded from the slot list during post-processing.

In the above examples, it suggests that by leveraging both point and line detection, the result can be improved under challenging situations. Yet when line detection fails or yields poor results, the parking slot is likely to be excluded from the final result. However, from a practical point of view, here the missed slot detection (false negative) is more acceptable than the false detection (false positive).

V. CONCLUSIONS

In this paper, we propose a deep-learning-based model to detect the parking corners and lines for slots inference. The hourglass architecture is used to generate point and line features, and the features are leveraged jointly for parking slot detection. The result is quite promising in accelerating fully convolutional networks without complicate architecture searching methods.

At the same time, the proposed effective network design solution, using the SP module to automatically outline the

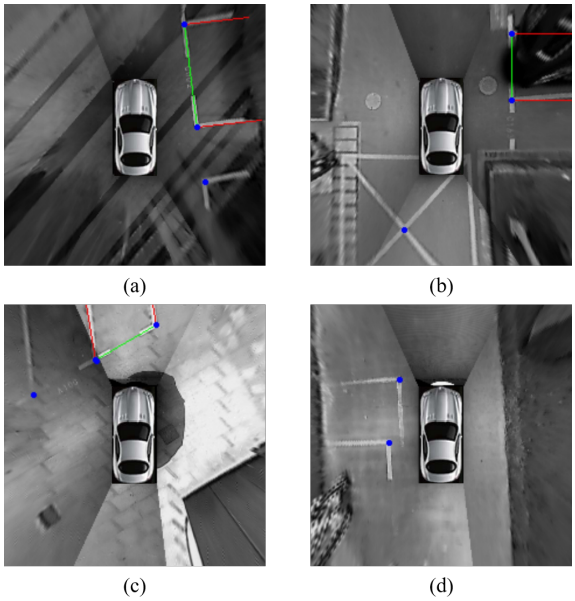


Fig. 12. Different results under shadows and interference, (a) yields correct corners and slots, (b) encounters a misleading ground marking and gives a wrong corner, but excludes it during slot inference, (c) yields the correct corners yet misses one slot as a result of the ground marking degradation, (d) gets completely wrong corners because of the misleading ground marking,yet successfully excludes the corners as no proper slot can be formed

shape of the network and choose the optimal convolutional kernel configurations during training, can be borrowed for many other applications. The SP module can directly replace the convolution blocks in any network, and get free performance improvement by sacrificing some training speed but without additional inference overhead.

ACKNOWLEDGMENT

The research leading to these results has partially received funding from the 2018 Shanghai AI Innovative Development Project.

REFERENCES

[1] Zhang Lin, Huang Junhao, Li Xiyuan, and Xiong Lu. Vision-Based Parking-Slot Detection: A DCNN-Based Approach and a Large-Scale Benchmark Dataset. *IEEE Transactions on Image Processing*, 27:5350–5364, November 2018.

[2] Wu Yan, Yang Tao, Zhao Junqiao, Guan Linting, and Jiang Wei. VH-HFCN based Parking Slot and Lane Markings Segmentation on Panoramic Surround View. *arXiv:1804.07027 [cs]*, April 2018. arXiv: 1804.07027.

[3] Zhang Lin, Li Xiyuan, Huang Junhao, Shen Ying, and Wang Dongqing. Vision-Based Parking-Slot Detection: A Benchmark and A Learning-Based Approach. *Symmetry*, 10:64, March 2018.

[4] Bruls Tom, Porav Horia, Kunze Lars, and Newman Paul. The right (angled) perspective: Improving the understanding of road scenes using boosted inverse perspective mapping. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, page 302–309, Jun 2019.

[5] Xiao Liu, Lin Zhang, Ying Shen, Shaoming Zhang, and Shengjie Zhao. Online camera pose optimization for the surround-view system. *the 27th ACM International Conference on Multimedia*, page 383–391, 2019.

[6] Newell Alejandro, Yang Kaiyu, and Deng Jia. Stacked Hourglass Networks for Human Pose Estimation. *arXiv:1603.06937 [cs]*, March 2016.

[7] Jin Xu, Guang Chen, and Ming Xie. Vision-guided automatic parking for smart car. In *Proceedings of the IEEE Intelligent Vehicles Symposium 2000 (Cat. No.00TH8511)*, pages 725–730, Dearborn, MI, USA, 2000. IEEE.

[8] Ho Gi Jung, Dong Suk Kim, Pal Joo Yoon, and Jaihie Kim. Parking Slot Markings Recognition for Automatic Parking Assist System. In *2006 IEEE Intelligent Vehicles Symposium*, pages 106–113, Meguro-Ku, Japan, 2006. IEEE.

[9] Wang Chunxiang, Zhang Hengrun, Yang Ming, Wang Xudong, Ye Lei, and Guo Chunzhao. Automatic Parking Based on a Bird's Eye View Vision System. *Advances in Mechanical Engineering*, 6:847406, January 2014.

[10] Suhr, Jae Kyu, and Jung Ho Gi. Full-automatic recognition of various parking slot markings using a hierarchical tree structure. *Optical Engineering*, 52:037203, March 2013.

[11] Suhr, Jae Kyu, and Jung Ho Gi. Sensor Fusion-Based Vacant Parking Slot Detection and Tracking. *IEEE Transactions on Intelligent Transportation Systems*, 15:21–36, February 2014.

[12] Redmon Joseph and Farhadi Ali. YOLO9000: Better, Faster, Stronger. *arXiv:1612.08242 [cs]*, December 2016. arXiv: 1612.08242.

[13] Iandola Forrest N., Han Song, Moskewicz Matthew W., Ashraf Khalid, Dally William J., and Keutzer Kurt. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5mb model size. *arXiv:1602.07360 [cs]*, February 2016. arXiv: 1602.07360.

[14] Krizhevsky Alex, Sutskever Ilya, and Hinton Geoffrey E. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60:84–90, May 2017.

[15] Howard Andrew G., Zhu Menglong, Chen Bo, Kalenichenko Dmitry, Wang Weijun, Weyand Tobias, Andreetto Marco, and Adam Hartwig. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv:1704.04861 [cs]*, April 2017. arXiv: 1704.04861.

[16] Sandler Mark, Howard Andrew, Zhu Menglong, Zhmoginov Andrey, and Chen Liang-Chieh. MobileNetV2: Inverted Residuals and Linear Bottlenecks. *arXiv:1801.04381 [cs]*, January 2018. arXiv: 1801.04381.

[17] Howard Andrew, Sandler Mark, Chu Grace, Chen Liang-Chieh, Chen Bo, Tan Mingxing, Wang Weijun, Zhu Yukun, Pang Ruoming, Vasudevan Vijay, Le Quoc V., and Adam Hartwig. Searching for MobileNetV3. *arXiv:1905.02244 [cs]*, May 2019. arXiv: 1905.02244.

[18] Zhang Xiangyu, Zhou Xinyu, Lin Mengxiao, and Sun Jian. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. *arXiv:1707.01083 [cs]*, July 2017. arXiv: 1707.01083.

[19] Ma Ningning, Zhang Xiangyu, Zheng Hai-Tao, and Sun Jian. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. *arXiv:1807.11164 [cs]*, July 2018. arXiv: 1807.11164.

[20] Chollet François. Xception: Deep Learning with Depthwise Separable Convolutions. *arXiv:1610.02357 [cs]*, October 2016. arXiv: 1610.02357.

[21] He Kaiming, Zhang Xiangyu, Ren Shaoqing, and Sun Jian. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015. arXiv: 1512.03385.

[22] Huang Gao, Liu Zhuang, van der Maaten Laurens, and Weinberger Kilian Q. Densely Connected Convolutional Networks. *arXiv:1608.06993 [cs]*, August 2016. arXiv: 1608.06993.

[23] Wang Robert J., Li Xiang, and Ling Charles X. Pelee: A Real-Time Object Detection System on Mobile Devices. *arXiv:1804.06882 [cs]*, April 2018. arXiv: 1804.06882.

[24] Li Xiang, Wang Wenhai, Hu Xiaolin, and Yang Jian. Selective Kernel Networks. *arXiv:1903.06586 [cs]*, March 2019.

[25] Ren Shaoqing, He Kaiming, Girshick Ross, and Sun Jian. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *arXiv:1506.01497 [cs]*, June 2015. arXiv: 1506.01497.

[26] Lee Jun-Tae, Kim Han-Ul, Lee Chul, and Kim Chang-Su. Semantic line detection and its applications. *ICCV*, page 9, 2017.

[27] Yu Fisher and Koltun Vladlen. Multi-Scale Context Aggregation by Dilated Convolutions. *arXiv:1511.07122 [cs]*, November 2015. arXiv: 1511.07122.

[28] Christian Szegedy Wei Liu Yangqing Jia Pierre Sermanet Scott E. Reed Dragomir Anguelov Dumitru Erhan Vincent Vanhoucke and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[29] Molchanov Pavlo, Tyree Stephen, Karras Tero, Aila Timo, and Jan Kautz. Pruning Convolutional Neural Networks for Resource Efficient

Inference. *arXiv:1611.06440 [cs, stat]*, November 2016. arXiv: 1611.06440.

[30] Li Hao, Kadav Asim, Durdanovic Igor, Samet Hanan, and Graf Hans Peter. Pruning Filters for Efficient ConvNets. *arXiv:1608.08710 [cs]*, August 2016. arXiv: 1608.08710.

[31] You Zhonghui, Yan Kun, Ye Jinmian, Ma Meng, and Wang Ping. Gate Decorator: Global Filter Pruning Method for Accelerating Deep Convolutional Neural Networks. *arXiv:1909.08174 [cs, eess]*, September 2019. arXiv: 1909.08174.

[32] Kingma Diederik P. and Ba Jimmy. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, December 2014. arXiv: 1412.6980.

[33] Lin Tsung-Yi, Goyal Priya, Girshick Ross, He Kaiming, and Dollár Piotr. Focal Loss for Dense Object Detection. *arXiv:1708.02002 [cs]*, August 2017. arXiv: 1708.02002.

[34] Paszke Adam, Gross Sam, Chintala Soumith, Chanan Gregory, Yang Edward, DeVito Zachary, Lin Zeming, Desmaison Alban, Antiga Luca, and Lerer Adam. Automatic differentiation in pytorch. http://https://pytorch.org/, 2017.

[35] Liu Wei, Anguelov Dragomir, Erhan Dumitru, Szegedy Christian, Reed Scott, Fu Cheng-Yang, and Berg Alexander C. SSD: Single Shot MultiBox Detector. *arXiv:1512.02325 [cs]*, 9905:21–37, 2016. arXiv: 1512.02325.