Implementation of CNN-Based Parking Slot Type Classification using Around View Images

Hoseok Do¹,², Jihyun Kim², Kwon Lee², Deukhyeon Kim², Kyuyeol Chae², Jin Young Choi¹
¹ASRI, Dept. of Electrical and Computer Eng., Seoul National University, Seoul, Republic of Korea
²Advanced Camera Lab, LG Electronics, Seoul, Republic of Korea
{hoseok03, jychoi}@snu.ac.kr, {jihyun88.kim, k.lee, deukhyeon.kim, kyuyeol.chae}@lg.com

Abstract—This paper presents a commercial implementation of CNN-based classification of parking slot type using around view images. The existing automatic parking systems use ultrasonic sensors, but they often fail to classify the types of parking slots. Around view images can depict the types of parking slots distinguishably. However, due to the diverse lighting and ground conditions, it is difficult to classify the parking slot type using images. Moreover, it is hard to find the parking lines since the lines are often occluded by vehicle or erased. To overcome these problems, we have constructed an extensive dataset composed of 480,000 images acquired on various environments using around view monitoring (AVM) camera mounted on a commercial vehicle. For training our CNN-based classifier, we subdivided parking slot types into ten categories and finally derived three parking slot types for actual applications. To operate the classifier in real vehicles, we designed CNN model suitable to embedded systems and implemented it using GPU. In experimental evaluation, the implemented CNN-based classifier achieves an accuracy of 94.15%. And the processing time of our classifier achieves 3.67ms per frame (270 fps) on the NVIDIA Tegra CX embedded system.

I. INTRODUCTION

The automatic parking system is a useful equipment widely used in mass-produced vehicles. The automatic parking system consists of a step of finding a parking space and a step of controlling the vehicle to park in the parking space. In both steps, it is important to recognize surrounding obstacles. The methods for perceiving available parking space are roughly divided into two categories: rebound-sensor-based method and vision-based method [10]. In the rebound-sensor-based method, a vacant space is detected between adjacent vehicles using various range-finding sensors, such as ultrasonic sensors [1]-[4], laser scanners [5]-[6], short-range radars [7]-[8], structured light [9]. Most vehicle manufacturers use ultrasonic sensors for the automatic parking system [4]. Ultrasonic sensors have the advantage of low cost and are installed in many mass-produced vehicles in recent years.

However, there is a limitation in finding parking spaces using the ultrasonic sensor. Since the ultrasonic sensor can detect only the presence of an obstacle, all the empty spaces are recognized as a parking space regardless of whether it is actually a parking space or not. So, the automatic parking system based on the ultrasonic sensor can recognize only the space adjacent to the obstacle as the parking space. In addition, a vacant space can be misrecognized as a parking space by the ultrasonic sensors even if it is not a parking slot. Especially, as shown in Fig. 1, ultrasonic sensors cannot discriminate an empty space of one parallel parking slot and an empty space composed of two or three consecutive parallel parking spaces. The vision-based methods identify the parking slots defined by parking line painted on the ground. Although many vision-based methods are proposed in recent year, these methods rely on the parking line features such as edges and corners [10]-[13]. In real world, it is often hard to find the parking line features, since the lines are often occluded by vehicle or erased. So, the performance of the vision-based methods can be degraded in these cases.

To mitigate these problems, we developed a CNN-based classifier based on overall appearance of a parking space. To this end, we construct our own dataset composed of extensive real-world images acquired by around view monitoring (AVM) cameras already mounted in many mass-produced vehicles. In actual applications, three classes (perpendicular, parallel, others) are enough to be applied to automatic parking systems. However, in practice, each class of the three classes may include too versatile images hard to be classified by one class. This is because the images even for a same parking slot type are significantly different depending on whether the vehicle is parked in the parking slot. To resolve this issue, we categorize 10 fine-grained classes by subdividing each class into subtypes of a parking slot and adding certain kinds of obstacles such as vehicles and pillars. The fine-grained classification results are combined into the three classes in hierarchical manner for actual applications. This fine-graining/combining scheme has been evaluated as improving the accuracy of the classification in experimental evaluation. Our classifier is implemented on the NVIDIA Tegra CX system using CUDA and FP16.

In experiments, the accuracy and the computational complexity of CNN models are evaluated. And we also experimented with changing the angle of camera installation

Figure 1. Example cases difficult to be classified using ultrasonic sensors. (a) is similar to the empty space size of three perpendicular parking slots in (b) is similar to the empty space size of a parallel parking slot in (a).
because the extrinsic parameters of the camera may change slightly during driving or repair.

II. CNN-BASED CLASSIFIER IMPLEMENTATION

The parking lot image data have been directly acquired using vehicles equipped with AVM camera and modified to be used for training our classifier. Using these images, our CNN model has been trained to classify parking spaces into 10 classes. We then implemented this CNN model on the embedded system using CUDA.

A. Data acquisition

The images have been captured from wide angle cameras for AVM system installed in left and right side mirrors of both sedan and sport utility vehicle (SUV). The cameras have different intrinsic parameters such as Field-of-View (FoV) and extrinsic parameters such as install height and angle. The resolution of the camera installed in sedan is 1024 by 768 and the FoV of that is 192 degrees, the install height and angle are 0.988m and 30.3 degrees. The camera installed in SUV has the resolution of 1280 by 720 and the FoV of 190 degrees, and its install height and angle are 1.132m and 46.1 degrees.

We have applied the lens distortion correction method proposed by [14] to the images captured from each camera because the wide-angle camera has heavy lens distortion. Then, to create a CNN input image of the same resolution from different AVM camera images, we have calculated homography matrices which convert the coordinates on undistorted image domain to the coordinates on virtual image domain, i.e. bird-eye view image domain. In other words, the original images captured from each AVM camera in sedan and SUV are transformed into the bird-eye view images, where one pixel of the transformed images indicates 2cm by 2cm in vehicle domain. Since the image is transformed assuming that the vehicle surroundings are flat ground, standing objects such as pillars and vehicles look distorted during this transform process. And since each vehicle has different camera parameters, this distortion is different for each vehicle.

Images were acquired using different types of vehicles for various parking spaces in various places and situations. Using these images, we constructed a dataset of 480,000 labeled images. By training our classifier using images of various kinds of vehicles, the classifier can be easily applied to diversiform vehicles in commercialization. Fig. 2 shows the original image and bird-eye view image which is used as the input image of our network.

B. CNN model

Our CNN model has been simplified based on MobileNet [15] for reduction of computational complexity. The detailed structure is shown in Fig. 3. After normalizing the input image of 230 by 345, it is used as the input of the CNN model. Since NVIDIA GPUs execute warps of 32 parallel threads, it is efficient to execute operation in 32 units. Thus, the number of channels in each layer is set to a multiple of 32. For the same reason, the input image is resized to 64 by 96 of which height and width are multiples of 32.

The number of classes is 10 and defined as Table I. Parking space classes are labelled according to the separating line and parking status. We set the input image height to 345 which corresponds to 6.9m in actual space, and the average length of the parallel parking slot is about 5m, thus the length of the parallel parking slot is about 70 percent of the image area. Therefore, an image can have part of one or two parallel parking slots. If one parallel parking slot is fully present, it is classified as empty parallel or parked parallel according to parking status. If part of a parallel parking slot or part of two consecutive parallel parking slots exist in an image, it is

![Figure 3. The proposed CNN model structure.](image-url)
CUDA is a parallel computing platform and programming model for general computing on GPU. By running thousands of threads on GPU using CUDA, we speed up our algorithm. In CUDA, a block is organized as 3D array of threads, and a grid is organized as a 2D array of blocks. Fig. 4 shows an example of a grid, a small square represents a thread.

Most operations of our algorithm such as image transformation, image normalization, convolution, max pooling, fully connected layers, batch normalization, rectified linear units (ReLU) have been coded in the CUDA kernels. These operations should access data from multiple memory spaces, but the embedded systems have a limited memory bus. Hence, efficient memory access is crucial to speed up our algorithm. The GPU architecture supports several different memory spaces as global, texture, constant, shared and register memory. Threads in a block can share data with the other threads within the block through shared memory, and data stored in global memory is visible to all GPU threads and the host within the application [17]. To speed up the process, we reduce the number of accessing global memory since accessing global memory is slower than accessing other memory such as shared memory. However, the shared memory size of our embedded system, NVIDIA tegra CX, is limited to 48KB. Thus, we design the block and thread size of each kernel to

C. Implementation on GPU

Recently, deep-learning algorithms demand a lot of computational complexity and GPUs have been considered as a general method to accelerate the processing time. In [16], satisfactory performance in terms of speed was achieved by training neural networks using multi-core computation devices. There are some frameworks to exploit GPU such as Tensorflow, Compute Unified Device Architecture (CUDA). While Tensorflow based on Python, CUDA is compatible with C/C++, which is suitable to our embedded AVM system.
efficiently use the shared memory.

All threads in a grid execute the same kernel function, and it is efficient to execute operation in 32 units. Thus, the CNN model weight parameters trained on PC have been converted to be suitable to the embedded system. The weights have been converted to be arranged in order of output channel, depth, width, and height since the number of a channel is set to multiples of 32. And the weights and input feature map of each layer are stored in the shared memory for matrix multiplication operation. In this way, the threads can access consecutive memory locations in a warp.

A thread is assigned to each output feature and the block is arranged in two-dimensions(2D) where x-direction is D and y-direction is multiplication of W and H, where D, W and H mean the depth, width and height of an input image, respectively, as shown in Fig. 4. The stream, that is a concurrent operation in CUDA, is used to infer the left and right images simultaneously. CUDA streams are work-queues to express concurrency between different tasks.

Our embedded system is based on PASCAL which supports the 16-bit floating point (half precision, or FP16) arithmetic. Using FP16 can reduce the computational time for transferring data and usage of memory compared to typical floating point formats such as 32-bit single precision and 64-bit double precision. CUDA defines ‘half’ and ‘half2’ structures for FP16 and provides the intrinsic functions for arithmetic, comparison, conversion, and others. The float weights trained by the network and the normalized image data are converted to half2 type which enables to speed up our algorithm in an embedded system. Also, the operations of the algorithm are executed using FP16 intrinsic functions.

Fig. 5 shows an example of shared memory allocation using half2 structure. The size of the input and output feature map is 16 by 24 by 128 as shown in Fig. 5 (a). We set the thread size to 32 by 32 and set the block size to 2 by 12. 16 by 2 by 64 features are allocated to the thread block as shown in Fig. 5 (b), because half2 structure can process two features simultaneously on one thread. The input data are stored for computing output data in shared memory all together. The shared memory size required for input feature map is 18 by 4 by 64 including padding as shown in Fig. 5 (c). Then we get 2 by 16 by 64 output of depth-wise convolution and point-wise convolution. Using half2 structure, we can compute twice as much output as 32-bit floating point operations.

We do not use CUDA implementation libraries such as cuBLAS (Basic Linear Algebra Subprograms) which does not support half2 to implement our model. When implementing our CNN model, using the method proposed in this subsection reduces the computation time by 40%, rather than using the cuBLAS library. We do not use the cuDNN (CUDA Deep Neural Network) library either. Although cuDNN supports half2, it has a license cost issue and shows relatively slower speed than our method.

III. EXPERIMENTAL RESULTS

A. Classification accuracy

Fig. 6 shows the parking slot type images used in this paper. It includes various lighting and ground conditions. Some images have parking slots that have parking lines occluded or erased. A test set consisting of 24,000 images has been created and the performance of our implemented classifier has been evaluated based on this test set. Test set has been labeled by 3 types – parallel, perpendicular, others. And we have compared the performance of our CNN model with the other CNN models.

CNN models have been trained using 10 class labeled data set. And the inference results have been combined into 3 type class for testing. We trained each model for 40 epochs with batch size 256, and we used Adam optimizer. As shown in Table II, our model shows reasonable accuracy performance.
even with a small number of parameters, one tenth to one-hundredth that of the other models. By reducing the size of model parameters, the neural network execution time can be also reduced [18]. Therefore, our model that has small weight parameters is suitable to commercial embedded systems that require real-time processing.

To evaluate the effect of subdividing the class into 10 classes, we experimented with the training set of the combined version of the 3-class as the test set. 3-class version of our model has been trained using dataset labeled with 3 type class directly. 3-class version dataset is the merged version of original 10-class training set, in the same way in test set. Table III shows the accuracy for the entire test set and precision / recall value of each class. The accuracy of our model trained with 10-class dataset, is 1.5% higher than that trained with 3-class dataset. Especially, since parallel type is subdivided into four classes, the parallel precision shows the most significant improvement by 4% when training 10 classes dataset.

### B. Implementation using CUDA

Our goal is to make our algorithm working on GPU embedded in real vehicles. Thus we have implemented our algorithm in PC-based CUDA and applied the implementation to the embedded system. We first implemented our algorithm in C programming language based on Intel Core i7-6600 CPU at 3.60GHz with 16GB RAM. And it was further implemented in CUDA to accelerate on NVIDIA GeForce GTX 1070 using CUDA 9.0 version and the computing capability of 6.1. Table IV shows the processing time of each layer. When implemented in CUDA, it is 3600 times faster than the CPU version, and the computation time of the separable convolution layer is greatly reduced.

We then ported the PC version of the CUDA code to the NVIDIA Tegra CX system using CUDA 9.0 version and the computing capability of 6.2. Tegra CX has 790 MHz clock rate GPU, 1600 MHz clock rate and 128-bit bus width memory, and 256 CUDA cores. First, we have used the same 32-bit float as the PC code, then we have changed it to 16-bit float to speed up. By using 16-bit float, performance is maintained and the computation time is reduced by 45% as shown in Table IV. The computation time is greatly reduced in layers with large input size and high computational complexity, such as Separable Conv 2 and 4.

### C. Experiments on physical deformation

We have used AVM cameras mounted on the side mirrors to evaluate the effectiveness of our algorithm. The results show a significant improvement in accuracy and speed compared to previous methods.
of a vehicle. The side mirrors of the vehicle are parts that are relatively easily exposed to physical impact. Therefore, the extrinsic parameters of the camera may change slightly while driving. Or when the side mirror is replaced or repaired, the camera mounting angle could be slightly changed.

To see if our algorithm is robust against physical deformations, we experimented with changing the extrinsic parameter values pitch, roll, and yaw. These parameters are used to create the bird-eye view from the original wide-angle image. We have changed each angles from -5 degrees to 5 degrees and checked the accuracy change. We have experimented with several image sequences in the test set with parking spaces near the vehicle. When the parking slot is far away, even if the angle is only slightly deformed, it is classified as no parking space in the image ROI. As shown in Fig. 7, changing the pitch angle has a greater impact on performance than roll or yaw angle. We can see that our algorithm is robust to changes in roll and yaw angles. Changing the pitch value by -5 degrees causes the greatest decrease in accuracy, because the image distortion is highly dependent on the changes of the pitch angle. To mitigate this problem, ROI of the image should be further expanded, or additional learning is required.

IV. CONCLUSIONS

Most vehicle manufacturers use ultrasonic sensors for the automatic parking system due to its low cost. However, ultrasonic sensors cannot distinguish the parking slot type efficiently since the ultrasonic sensor can detect only the presence of an obstacle. Although many vision-based methods are proposed in recent years to overcome this problem, the existing vision-based methods rely on the parking line features such as edges and corners. Using the parking line features cannot classify the parking slot type since the parking lines are often occluded by vehicles or erased. To overcome these problems, we subdivided parking slot types into 10 classes to train our deep learning model and finally derive 3 parking slot types. Experimental results demonstrate that the proposed method could classify the parking slot types and shows the satisfactory accuracy of 94.15%. The proposed method is implemented on GPUs, and the processing time of our proposed method satisfies the real time requirement by achieving 3.67ms speed per frame on the NVIDIA Tegra CX embedded system.

REFERENCES