Real-Time Rain Detection and Wiper Control Employing Embedded Deep Learning

Chih-Hung G. Li, *Member, IEEE*, Kuei-Wen Chen, Chi-Cheng Lai, and Yu-Tang Hwang

***Abstract*—A state-of-the-art real-time rain detection and wiper control method is proposed in this article. Currently, commercial models adopt electronic sensors that can only sample the humidity of a small region of the windshield. The existing computer vision methods primarily focus on the detection and counting of raindrops and provide a recall rate of less than 70%. Here we adopted a holistic-view deep learning approach to build a visual classifier that is robust to large varieties of background scenes, illumination, and water forms. Specifically, Deep Residual Network (ResNet) was adopted as the visual classifier that distinguishes between rainy and fair street scenes and controls the wipers accordingly. To verify the practicality of the proposed deep learning framework, we tested the network on various embedded computing systems, including an embedded computing cluster. The results show that the deep learning rain detector outperforms previous state-of-the-art methods with higher rain recall and precision. It was also found that with the help of some graphic computation-enhancing components, commonly available embedded systems in the market can provide comparable performance to personal computers. While using the enhanced embedded systems to build a cluster, a performance superior to PC was witnessed. As the embedded system is cost-effective, small, and lighter, normalized performances for various aspects clearly show the competitive edge of the embedded systems and confirm the practicality of the proposed system. We also release the dataset of 160 k images used for training the visual rain detector.**

***Index Terms*—Computer performance, computer vision, machine learning, neural networks, vehicular automation.**

I. Introduction

IN THE CONTEXT of the Advanced Driver Assistance System (ADAS), advanced sensing assistance is of great importance and practicality, as vehicle driving by a human driver is an act of complex judgment and control involving numerous types and levels of sensing. Among those, it is not exaggerating that the visual information of the road condition ahead received by the driver is the most critical one that should be obtained in good quality at all times. Should the visual quality be degraded by rain, snow, fog, dust, or insufficient lighting, safety would be greatly compromised and the driver should slow down or stop immediately.

Since the invention in the early 1900s, windshield wiper has been the primary device for maintaining the visibility of the road condition perceived by the driver. Over a century, the windshield wiper is controlled by the driver. The timing for activating the wiper is based on the driver's judgment, mainly when the driver feels that it is not comfortable to operate the vehicle and the current visibility needs to be enhanced. However, as the driver should fully concentrate on operating the driving wheel and the pedals, the assistance on the wiper operation is of significant ADAS importance. It was also found that rain on the windshield deteriorates the object detection performance of the deep learning-based visual detectors [1]. In recent years, some rain-sensing wiper systems emerged in some marketed automobile models [2], [3], in which infrared emitters and receivers are adopted to detect the humidity of the windshield. However, the electronic sensor can only sample a small region of the windshield and the detection reflects the humidity instead of the visibility. Some visual methods based on computer vision were proposed to detect and count the numbers of raindrops. In addition to the demand for increasing the detection accuracy, a method that is robust at detecting all kinds of water forms such as streak and downpour other than raindrop is also needed.

The authors proposed a holistic-view deep learning approach for windshield rain detection and wiper control in [4]. Specifically, the proposed convolutional neural network (ConvNet) learns from human driver judgment and can perform robust visual classification subject to large varieties of background scenes, illumination, and water forms. To further verify the feasibility of the proposed framework, in this paper, we attempt to devise embedded systems to meet the demand for mobile applications. Detailed test results regarding the performance of each embedded system are also reported. We addressed the need for the detection speed and accuracy simultaneously to propose a method that can be practically realized in common vehicles. The proposed ConvNet rain detector is equipped with sufficient capacity to capture various visual rain effects, yet light enough to operate on embedded computing systems with acceptable speed and accuracy.

The contributions of the work are four-fold. We introduce a deep learning-based holistic-view rain detection scheme that can be executed on embedded computing systems. We present experimental evidence that the proposed embedded systems

Manuscript received August 21, 2020; revised December 1, 2020 and February 14, 2021; accepted March 8, 2021. Date of publication March 17, 2021; date of current version May 5, 2021. This work was supported by the Ministry of Science and Technology of the Republic of China, Taiwan under Contract MOST109-2637-E-027-008. The review of this article was coordinated by Prof. Z. Ma. *(Corresponding author: Chih-Hung G. Li.)*

Chih-Hung G. Li and Yu-Tang Hwang are with the Graduate Institute of Manufacturing Technology, National Taipei University of Technology, Taipei 10608, Taiwan (e-mail: cl4e@ ntut.edu.tw; projectreven927@gmail.com).

Kuei-Wen Chen is with the Graduate Institute of Mechatronics Engineering, National Taipei University of Technology, Taipei 10608, Taiwan (e-mail: 1294wade@gmail.com).

Chi-Cheng Lai is with Compal Electronics Inc., Taipei 114, Taiwan (e-mail: bg784533@gmail.com).

Digital Object Identifier 10.1109/TVT.2021.3066677

provide state-of-the-art performance in terms of computing speed and detection accuracy. We also demonstrate an embedded computing cluster that further enhances the performance of rain detection and wiper control. Finally, we release the dataset of 160k images used for training the ConvNet detector here and in [4].

II. Related Work

A. General Methods of Windshield Rain Detection

Using electronic components to indirectly detect the existence of water on the windshield is the most common method and the one currently adopted in the automobile industry, e.g., detection of the reflection of infrared by a device mounted inside the windshield glass near the rearview mirror [2], [3], [5]. Other similar approaches include detection of capacity change due to rain by a pair of copper plates [6], rain intensity by a pressure sensor [7], measurement of electrical conductivity [8], etc. A common problem with this type of method is that the component only detects local conditions regarded as a representation of the entire windshield. If the spot where the sensor detects experiences differently from the majority of the windshield, the signal may be inaccurate and the reaction of the wiper system may not be satisfactory. In [9], a camera was proposed to collect the light signals and the sampling area was somewhat enlarged; however, it is still relatively small compared to the entire windshield.

B. Computer Vision Rain Detection

The technology of computer vision has delivered progress and breakthrough in rain detection. Kurihata *et al.* [10] investigated the video images of the windshield. They devised a learning algorithm for raindrop patterns; visual detection was performed by template matching with the so-called eigendrops. Roser and Geiger [11] approximated the adherent raindrops as partial spheres and constructed the raindrop pattern on the windshield based on Snell's law of refraction, Fesnel's reflectivity, and the blurring effect due to out-of-focus. Raindrop detection was then performed by comparing the region of interest with the artificial raindrop pattern. Their comparison indicated a superior performance than the SURF baseline. You *et al.* [12] tackled the problem by analyzing raindrops' local spatio-temporal derivatives [13] based on their findings that the motion of raindrop pixels is slower than that of non-raindrop pixels, and the temporal change of intensity of raindrop pixels is smaller than that of non-raindrop pixels. However, their methods cannot be directly applied to a single image. Ishizuka and Onoguchi [14] proposed a method that can detect raindrops regardless of the shape. In the daytime, they examine the degree of the blur between a raindrop area and its surrounding area; at night, bright areas in which the intensity does not change so much are detected as raindrops. The above methods primarily depicted the advancement in raindrop detection and counting; however, none of them was tested with abundant raining data reflecting a large variety of rain patterns. For example, in a downpour, the water pattern on the windshield can be quite different from that of the raindrops (see Fig. 1); there was no evidence that the above methods can function effectively.

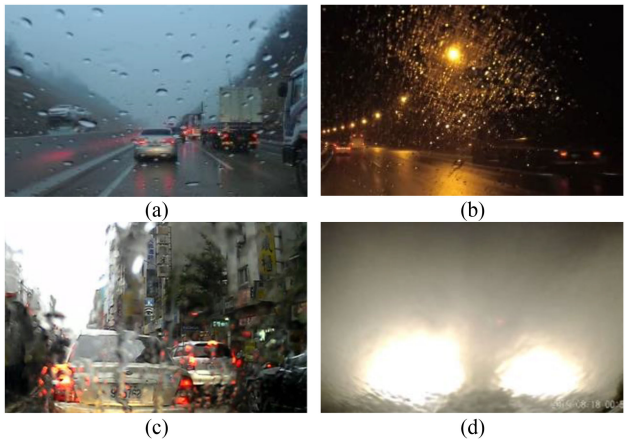

Fig. 1. Dashcam images showing the rain effects on the windshield: (a) day raindrops, (b) night raindrops, (c) water streaks; (d) downpours.

Also, as will be clearer in Section VI, a recall rate of less than 70% of the existing state-of-the-arts still needs to be improved.

C. Convolutional Neural Network

To propose a visual method that is effective for all water forms, the outstanding image recognition capabilities exhibited by Deep ConvNet attracted our attention [15]. It is capable of providing a high level of abstraction from the features it extracts. Although the detection of rain patterns on the windshield is different from ordinary object detection [16], [17], which focuses on fixed objects with a definite appearance, the visual effects of water forms do appear to follow some rules. In the past, ConvNet was adopted to detect the existence and locations of rain and dust; e.g., Eigen *et al.* [18] collected a dataset of clean/corrupted image pairs to train a specialized form of ConvNet for direct mapping of corrupted (rain or dust) image patches to clean ones. Qian *et al.* [19] trained an attentive Generative Adversarial Network for the task of transforming a raindrop degraded image into a clean one. Essentially these techniques restore the image and can enhance visibility without the help of a wiper. However, the computation intensity involved in these operations makes it questionable to operate the scheme on the vehicles in real-time on real roads; it is also unclear whether the proposals can work on water forms other than raindrops. Nevertheless, the above examples provided us sufficient confidence in ConvNet regarding high adaptability to large varieties of water forms and backgrounds. What we focused on in this study is to design an embedded system that achieves a good balance between detection accuracy and speed and can be operated for mobile applications.

III. Deep Learning Framework of Rain Detection

The proposed rain detection and windshield wiper control scheme is displayed in Fig. 2. We utilized the videos captured by ordinary front-facing dashcams that monitor the front view of the vehicle through the windshield for the training of the visual detector. The dashcam not only captures the changing street scenes in front of the vehicle; when there is rain, the holistic visual effects due to the rain are also truthfully captured

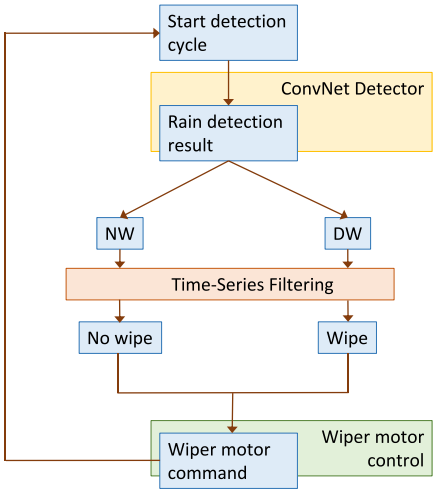

Fig. 2. Proposed windshield wiper control scheme. NW and DW are ConvNet output scores recommending "not to wipe" and "detected to wipe", respectively.

by the dashcam (see Fig. 1). We then build a deep ConvNet for the classification of rain and no-rain based on the holistic dashcam images. As will be clearer later, we train the ConvNet detector to output signals on wiper activation directly, instead of analyzing the topology of the adherent water. To suppress occasional detection outliers, we adopted a temporal treatment similar to [10] to remove impulsive detection errors that appear unreasonable. The filtered results are then used to command the wiper motor; experimental results were reported in [4].

A. Holistic Rain Detection

Although most of the previous visual rain detection works focused on the detection and counting of raindrops, water forms on the windshield and the resulted visual effects presented to the driver can be quite diverse. Roughly, the visual effects of water on the windshield can be categorized as raindrops, streaks, and downpours, besides the difference between daytime and nighttime. Raindrops, as pointed out by [11], work as small convex lenses and refract the light rays of the outside views according to Snell's law (see Fig. 1(a)). During the daytime, raindrops can create multiple inverted and blurred tiny images on the windshield. During the nighttime, the visual effects due to those convex lenses are more dominated by the point light sources on the streets; thus, the raindrops magnify the intensity of the light sources and exhibit small white dots like the ones in Fig. 1(b). When the rain becomes heavier, water may flow on the windshield and form streaks like the photo in Fig. 1(c). In this scenario, the irregular shape of the streak can highly distort the outside view; the visual effect is quite different from single raindrops and the method based on raindrop counting may fail [14]. When a downpour happens, the windshield can be covered by water puddles as shown in Fig. 1(d), in which the outside views become hardly distinguishable and the windshield view is dominated by the water ripples.

Here we adopt a holistic-view approach for detection of the rain condition on the windshield. Instead of counting the raindrops found on the windshield, deep ConvNet is trained to directly output a recommendation score for wiper activation regardless of the local details of the image. Given dashcam images of the holistic view X, the ConvNet detector classifies them into two categories DW and NW, which represent "detected to wipe" and "not to wipe", respectively. For the same period of X, the driver's visual experiences through the windshield are V, and the driver also divide V into DW' (wipe) and NW' (not wipe). The ConvNet is trained to reflect the driver's will on when to activate the wiper and when not. Thus, the task is to minimize a cost function defined as the difference between the recommendation of the detector and the driver's will on wiper activation,

$$DW^* = \underset{dw}{\arg\min} \|p\left(DW|X\right) - p\left(DW'|V\right)\|_2. \quad (1)$$

Note that the activation of the windshield wiper DW' can be quite subjective; e.g., some drivers tend to hesitantly activate the wiper and others simply cannot tolerate any water staying on the windshield. For a learning-based model and if there is only one single trainer, (1) will be obtained to reflect that specific trainer's will. If there are multiple trainers, (1) can only reflect some best average as $p(DW'|V)$ varies among different driver judgments.

B. Training Data

To tailor for a specific driver, the best solution to (1) is to collect the reactions of the driver and use them as the training set for the model. However, collecting data for each driver can be extremely tedious and impractical. A more feasible approach would be to obtain an average model based on data of multiple drivers. Assuming the drivers' visual experiences on dashcam videos are very close to the real scenes, at least regarding the judgment on wiper activation, V can be approximated by X, and (1) becomes,

$$DW^* = \underset{dw}{\arg min} \|p\left(DW|X\right) - p\left(DW'|X\right)\|_2. \quad (2)$$

Thus, X can be obtained from the internet where dashcam videos of fair and raining conditions are abundant. We collected videos of a total length of approximately 44 hours and sliced the videos into images at an interval of 1 frame per second (FPS). A total of 160k images were obtained, in which the numbers of fair and raining scenes are roughly equal; the ratio between the daytime and the nighttime images is roughly 9:7.

Three experienced drivers were hired to review these images and make a judgment for each image on whether the windshield wiper should be activated or not. The reviewers examined the visibility in the images just like what a driver normally does when driving. Thus, the DW' category may contain images of raindrops, streaks, or downpours; the NW' category contains primarily fair weather images. However, if the windshield has just been wiped and the water condition on the windshield shows that it needs not be wiped immediately, that image would very likely be classified as NW'. In sum, the classification of DW vs. NW conducted by ConvNet is not based on any topological model but directly trained from human judgment. We argue that this is the method closest to the reflection of the behavior of a human driver regarding the reaction to rain-degraded visibility. The images were then annotated accordingly and assembled to

TABLE I
CONTENTS OF THE TRAINING AND TESTING IMAGE DATA

		No-rain	Downpour	Streak	Raindrop
Train	Day	42,815	2,160	630	40,025
	Night	35,472	1,580	460	33,432

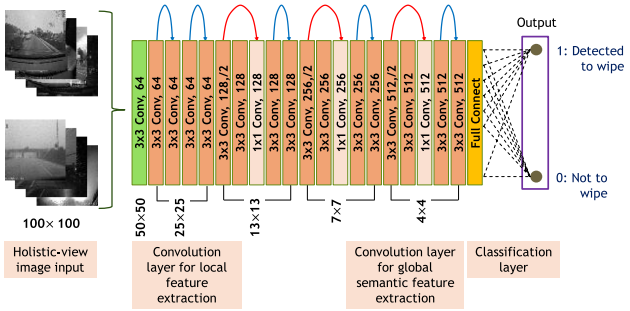

Fig. 3. Architecture of the 18-layer ResNet for visual rain detection.

form the training set for the visual detector. The composition of the training image data is exhibited in Table I; the backgrounds of the images have reflected the fundamental scenery commonly encountered by automobiles such as city roads, countryside, and highways in more than seven countries. It is worth noting that in the performance testing reported in Section V, the tested videos were never included in the training set. Thus, the obtained accuracy results may prove the generalization capability of the ConvNet detector.

C. Deep ConvNet

Deep ConvNet was adopted as the visual classifier of the deep learning framework; specifically, the 18-layer Deep Residual Network (ResNet-18) [20] was chosen for its superior performance in resolving the problems of vanishing gradients and degradation through learning residual functions. Instead of learning a direct mapping of $\mathcal{H}(x)$, the network is formulated to learn a residual function $\mathcal{F}(x) := \mathcal{H}(x) - x$. The shortcut connections of x realize the formulation of the residual function, yet introduce neither extra parameter nor computation complexity. In our earlier study, ResNet outperformed plain networks at converging accuracy; thus, we exclude the usage of plain networks in this report.

To realize real-time computation on embedded systems, we adopted the 18-layer ResNet instead of deeper networks. As shown in Fig. 3, the first few convolution layers extract the local visual features pertinent to the rainy effects. When the visual data propagate into deeper layers, a more abstract correlation is formed. Finally, the last few convolution layers extract the global semantic features for the final classification. During training, the training images collected above are resized to 100×100 pixels and transformed to greyscale for the input of the network. During detection, the same resizing and grey-scaling procedures are performed in real-time on the images captured by the camera. The final layer of the network is a binary output of 0 and 1 representing NW and DW, respectively.

D. Control Policy involving Time-Series Filtering

To devise a windshield wiper control system based on visual rain detection, the control policy π depends on the output ω of the visual detector Γ. Given that the rain detector examines the windshield condition at discrete times, in any instance i, the dashcam generates observation o_i, which forms the input $\mathrm{I}_i(o_i)$ to the visual detector. For a simple design, wiper control solely depends on the observation at the current instance as,

$$\pi\ (u_i|I_i) = \breve{\pi}\ (u_i|\omega_i)\,\Gamma\,(\omega_i|I_i)\,, \tag{3}$$

where $\breve{\pi}$ is a control policy function that only depends on the current ConvNet result in any instance. To remove some outlier visual detection, a time-series filtering scheme was proposed by [10] and adopted by [4]. The scheme works as a low-pass filter and screens out single detections that are not in line with a series of stable results. The time-series filtering scheme examines a series of consecutive detection results, instead of single detections as,

$$\pi\ (u_i|I_i) = \sum_{i-(n-1)}^{i} \tilde{\pi}\,(u_i|\omega_j)\,\Gamma\,(\omega_j|I_j)\ , \tag{4}$$

where $\tilde{\pi}$ is a control policy function that depends on a series of consecutive ConvNet results. Thus, the control command depends on the summation of the past few detection results, instead of the single result of the current instance. As reported in our previous work of [4], effective enhancement in precision and recall was achieved by checking every four consecutive ConvNet outputs ($n = 4$) and setting the wiper activation rule as,

$$\mathrm{S}\ (n) = \sum_{i-(n-1)}^{i} D\,(j) \geq \tau\,,$$
$$\mathrm{D}\ (j) = 1 \text{ for DW}; \mathrm{D}\ (j) = 0 \text{ for NW}, \tag{5}$$

where $\mathrm{S}(n)$ is the wiper activation function. When $\mathrm{S}(n)$ is greater or equal to the threshold τ, the wiper will be activated. In [4], precision and recall received some improvements when $\tau = 2$ or 3.

IV. COMPUTING SYSTEMS

A few embedded computing components commonly available in the market are acquired for this study, such as Jetson Nano, Raspberry Pi 4 (Pi4), and Neural Compute Stick 2 (NCS2). A mini personal computer (PC) was also included for comparison due to its relatively small size (see Fig. 4). The primary goal was to search for the embedded systems that have better speed and accuracy performance at executing deep neural networks in real-time. As our result later showed that Pi4 with NCS2 generates higher performance for our deep neural network, we also constructed a computing cluster using Pi4+NCS2 as the basic computing node.

A. System Specifications

The embedded computing components adopted in this study are introduced below:

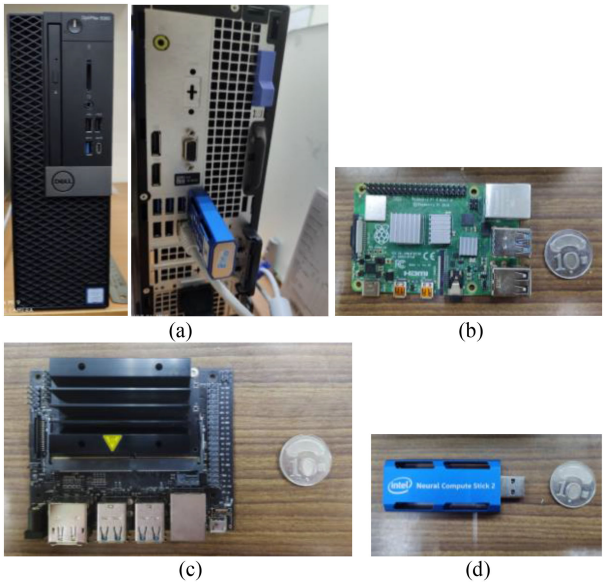

Fig. 4. Photos of the computing systems adopted in this study: (a) personal computer, (b) Raspberry Pi 4, (c) Jetson Nano; (d) Neural Compute Stick 2.

1) Personal Computer/Virtual Machine: A mini PC - Dell OptiPlex 5060 with dimensions of $29\,\mathrm{cm} \times 9\mathrm{cm} \times 29\mathrm{cm}$ was included in the study. It is equipped with a CPU of Intel i7-8700, 6 / 12, 3.20-4.60 GHz, a RAM of 24G, and a hard disk memory of 1T. In addition to Window10, virtual box 6.0.4 was installed and Ubuntu 16.04 was set up as the operating system of the virtual machine (VM), which acquires one virtual core, 6G memory, and 40G hard disk memory. In the VM, we installed Keras2.2.4, Tensorflow1.13.1, OpenCV4, and OpenVINO Toolkit for the required operations of the visual detector.

2) Jetson Nano: Jetson Nano is an embedded computing system developed for high-performance image processing. It is equipped with a CPU of Quad-Core ARM Cortex-A57 MPCore, 128-core NVIDIA Maxwell GPU, a memory of 4GB 64-bit LPDDR4 at 25.6GB/s, a storage of 16GB eMMC 5.1, and 12 lanes MIPI CSI-2 up to 4 cameras [21]. The dimensions of Jetson Nano are $10\mathrm{cm} \times 8\mathrm{cm} \times 3\mathrm{cm}$ and it consumes power between 5W and 10W.

3) Raspberry Pi 4: Raspberry Pi was initialized in 2015 for low-cost embedded computing. Now the fourth version of Raspberry Pi has become a widely adopted developer kit for many automation projects. It is equipped with a CPU of Quad-Core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5 GHz and up to 8GB LPDDR4-3200 SDRAM. The operating system is Raspbian; to execute our visual detector, we installed Keras2.2.4, TensorFlow1.13.1, OpenCV4, and OpenVINO Toolkit. To utilize NCS2, the Intel open-source codes on OpenVINO must be installed and operated on Raspbian.

4) Neural Compute Stick 2: Intel Neural Compute Stick has been developed based on the Vision Processing Unit (VPU) to accelerate vision-related AI algorithms. NCS2 is the 2nd generation of Neural Compute Stick equipped with Myriad 2 VPU, which contains 12 parallel vector processors called Streaming Hybrid Architecture Vector Engine (SHAVE). Myriad 2 VPU provides a computation speed of 2TFLOPS with a power consumption of less than 1W; it can be controlled by software to satisfy the needs of various applications and workloads.

B. Implementation of NCS2

To utilize NCS2 for the acceleration of image processing on PC and Pi4, two Python functions from the OpenVINO Toolkit – IEPlugin and IENetwork were utilized. IEPlugin is the main plugin interface and serves to initialize and configure the plugins such as CPU, GPU, or VPU. IENetwork allows us to reconstruct the deep neural network in NCS2 with the network file and the weight file and complete the setup of the input and output layers. It is worthy to note that the format of the input layer must contain four parameters as shown in Fig. 5. The reason is that NCS2 will perform the shunting action through four channels; if it is less or more than four, a format error will result.

C. Embedded Computing Cluster

Computing clusters are built upon communication and work-sharing among the programs, in which the storage space relies on a concept called "pool" [22]. This concept often appears in cloud technology, where the nodes of the cluster share and exchange information. In addition to this storage concept, it is also necessary to construct the computer's master-and-slave relationship. Through this mechanism, the resources needed by different nodes can be properly allocated to enhance the overall performance of the system. To utilize resources more effectively, we divide the visual detector into two blocks called the "client" and the "server." The communication protocol of these two nodes is achieved through the setting of Socket which controls the state of the Transmission Control Protocol (TCP). Under such a structure, one can specify the range of programs used by nodes, without overloading a single node. An illustration of the structure of the computing cluster is shown in Fig. 6.

Network File System (NFS) utilizes the network interface to allow different computers and operating systems to share individual files. The function of NFS is achieved through Remote Procedure Call (RPC). When NFS is activated, it will randomly grab some ports and then register the ports that need to be used with RPC. Then RPC will record the location and number of these ports, and open port 111, waiting for the client to request RPC. If RPC receives the request, it will send the RPC record of the server to the client, and the client can follow these records for data transmission. By the Linux commands, one or more Pi4s can be designated as NFS servers to store the program files required by the entire system. By adopting a network switch, the stored data can be shared among different nodes; thus, one can use it as a simplified pool to respond to the needs of each program. As shown in Fig. 6, pictures taken by the camera and the returned values must be sent to NFS. Any program that needs to use the data or objects will extract them from the same place; the nodes can concentrate on computation without wasting resources on data transmission.

A Python library called Fabric is adopted for constructing the core program of the computing cluster (see Fig. 7), as it is suitable for handling repetitive and huge amount of work. Its

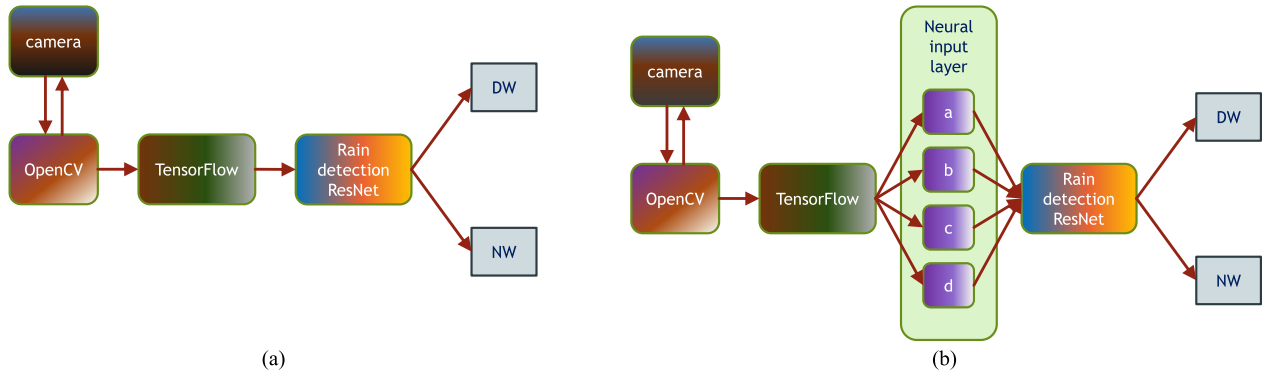

Fig. 5. Frameworks of the rain detection ResNet with and without NCS2: (a) the framework without NCS2; (b) the framework with NCS2.

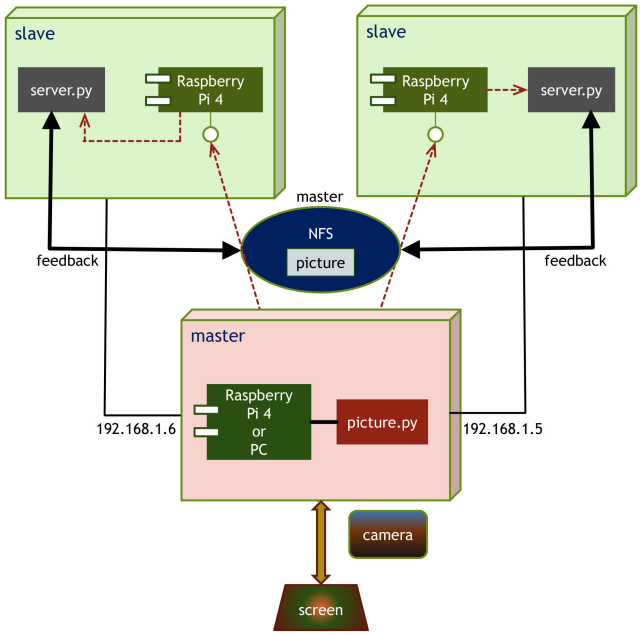

Fig. 6. Illustration of the structure of the computing cluster.

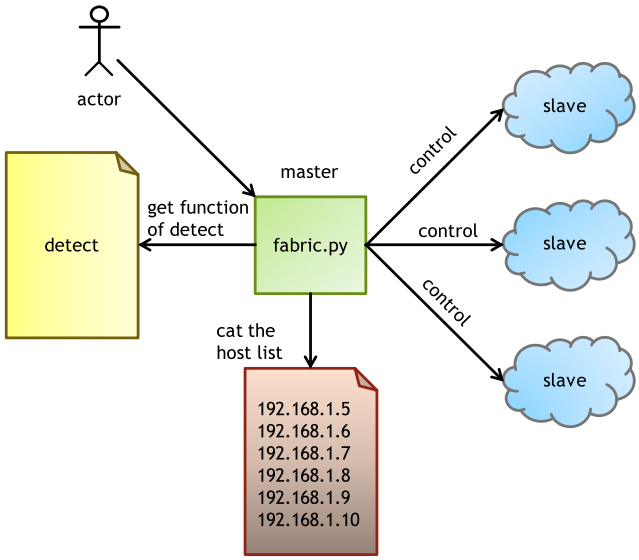

Fig. 7. Illustration of the Fabric structure for managing the data distribution of the nodes.

main function is to manage the data distribution of each node and perform backup and data transmission. Also, with this structure, one can easily and precisely control the activating sequence of the programs on the host and nodes to avoid sequence mistakes which may lead to systematic crashes. As shown in Fig. 7, when the user (actor) commands fabric.py, the program will look for two kinds of information from the internal parameters, one being the network location of all nodes, and the other being the function that needs to be activated in the command. With these two pieces of information, all nodes on the network can be controlled to perform the required actions. If one program has a problem, it will not affect the other programs; thus, it can be ensured that the entire system can remain functioning.

Communication between the programs is achieved by Network Socket, which serves as an endpoint for sending and receiving data across the network. TCP/IP network protocol is adopted to mark the application process, in which the three elements of IP address, protocol, and port number are used to identify where the program comes from, and where the reply needs to be sent. If two programs are to communicate with each other, it is necessary to establish a relationship between server and client. The server is responsible for completing the client's request and reporting a completion message after completion. The client initiates a control signal to call the server's service. The architecture chosen in this article is one-to-many, that is, one client (master) to multiple servers (detection program), to satisfy our need for distributed computing.

Physically, the embedded computing cluster is constructed with a maximum of 6 pieces of Pi4, each connected to an NCS2 on a USB port. An 8-port network exchanger connects all the Pi4s; USB chargers were adopted for the power supply. The overall dimensions of the 6-Pi cluster are approximately 19 cm $\times$ 18 cm $\times$ 12 cm. The photo of the embedded computing cluster is shown in Fig. 8.

V. Detection Performance

Experiments of real-time visual rain detection were carried out in the laboratory with a setup shown in Fig. 9. A webcam of 640 $\times$ 480 pixels is installed to capture the dashcam videos played on the computer monitor. The images captured are sent to the computing system being tested, where the ResNet detector is executed for real-time rain detection. The detection scores

Fig. 8. Photos of the embedded computing cluster constructed with 6 sets of Pi4+NCS2.

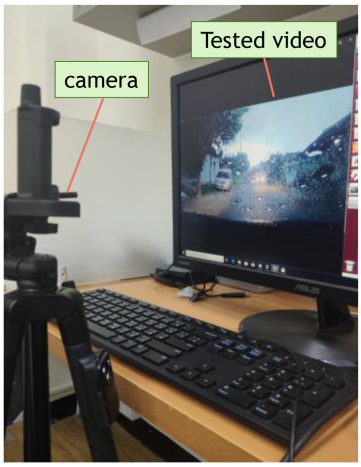

Fig. 9. Experiment setup of real-time visual rain detection.

are recorded and later compared with the ground truths for calculation of detection accuracy. Videos not included in the training set were solicited and utilized to test the detection speed and accuracy of the ConvNet detector. Five computational platforms were tested such as VM, VM+NCS2, Jetson Nano, Pi4, and Pi4+NCS2. To test the five computational platforms, 10 different videos reflecting various raining conditions were utilized, where films 1, 2, 3, 8, and 9 contain rainy scenes and the others contain no-rain scenes. To test the computing clusters, we focused on their rain recall performance and thus only used rainy films. The processing speed of each computational platform is evaluated by comparing the averages of the detection FPS, which reflects the duration from capturing the image to the output of the detection score. To test the detection accuracy of the program on each computational platform, the accuracy, precision, and recall rates are defined as the following,

$$\text{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn}, \tag{6}$$

$$\text{Precision} = \frac{tp}{tp + fp}, \tag{7}$$

$$\text{Recall} = \frac{tp}{tp + fn}, \tag{8}$$

where *tp*, *tn*, *fp*, and *fn* are defined in Table II. Note that the time series treatment described in (5) was not adopted in the experiments. The results are discussed in the following.

TABLE II
DEFINITIONS OF THE FOUR SITUATIONS FOR THE EVALUATION OF ACCURACY, PRECISION, AND RECALL

Detection \ Ground truth	Relevant/ Rain	Irrelevant/ No rain
Detected to wipe (DW)	True positive (tp) Detected to wipe on relevant images	False positive (fp) Detected to wipe on irrelevant images
Detected not to wipe (NW)	False negative (fn) Detected not to wipe on relevant images	True negative (tn) Detected not to wipe on irrelevant images

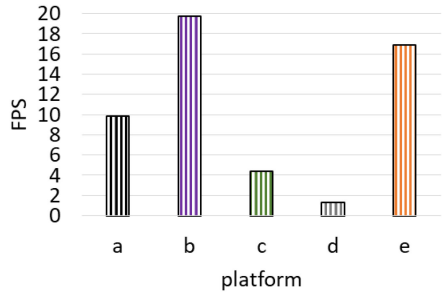

Fig. 10. Average FPS results of the five computation platforms on steadfast images: a. VM, b. VM+NCS2, c. Jetson Nano, d. Pi4; e. Pi4+NCS2.

A. *Processing Speed*

1) Various Computational Platforms: Prior to testing the dashcam videos, experiments were carried out on steadfast images, from which the maximum FPS of each platform could be achieved as the computation loading is lower than the dashcam videos that contain rapidly changing visual contents. The results are shown in Fig. 10, which indicates that the fastest system is VM+NCS2, with Pi4+NCS2 trailing with an FPS of 17. The results reveal the high effectiveness of acceleration by NCS2, particularly exhibited on Pi4, where the FPS increases by 16 folds.

The FPS results of each platform testing dynamic scenes are shown in Fig. 11, where the averages of FPS are also provided. One may find that the order of average speed at detecting videos is similar to that of detecting steadfast images, except that VM is slightly faster than Pi4+NCS2 at dynamic scenes. Most of the platforms exhibited a relatively steady detection speed, whereas VM+NCS2 showed more dependency on the video content; the difference can be as much as 8 FPS. A deeper investigation reveals that for those cases of relatively low FPS (Case 2, 3, 4, and 9), the videos are either nighttime recordings or the very dark daytime (see Fig. 12). It appears that dark images were less accelerated by VM+NCS2, likely due to the larger numbers associated with darkness increasing the loading of computation. The most intriguing result is still the enhancement of Pi4 by NCS2. With the help of acceleration by NCS2, Pi4 was able to provide a speed of 10 FPS similar to a PC. The discrepancy between light and dark images witnessed on VM+NCS2 was not seen on Pi4+NCS2. If a steady FPS is critical, Pi4+NCS2 is recommended; however, VM+NCS2 can further propel the brighter images.

2) Embedded Computing Cluster: We constructed computing clusters with one to six sets of Pi4+NCS2, as Pi4+NCS2

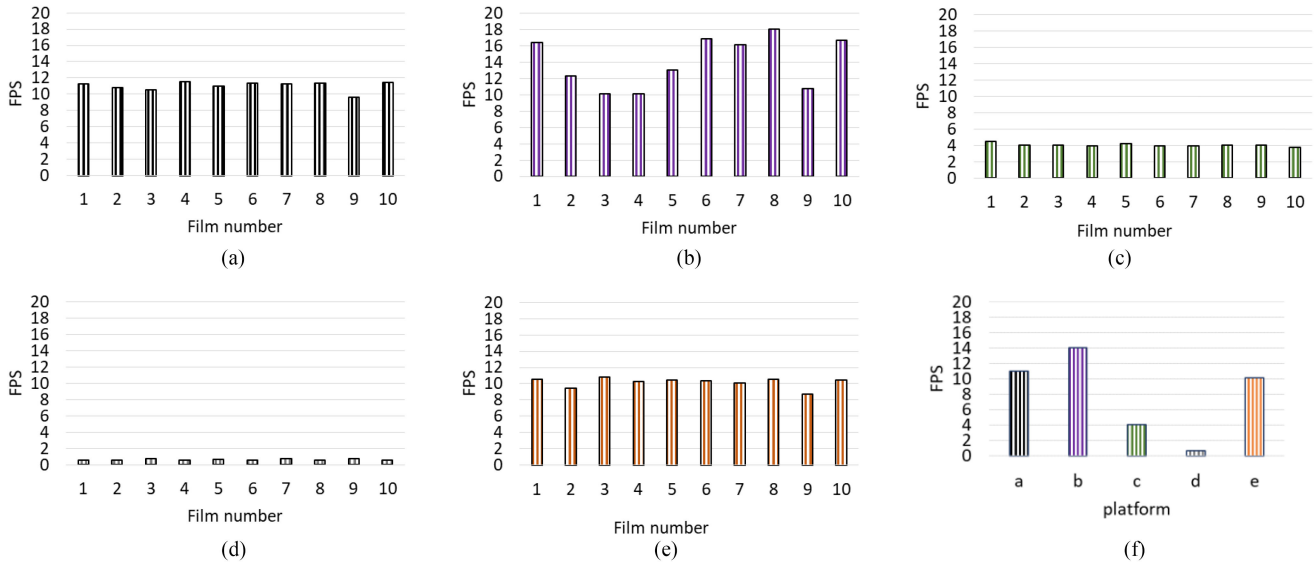

Fig. 11. FPS results of ten dashcam videos by five different computation platforms: (a) VM, (b) VM+NCS2, (c) Jetson Nano, (d) Pi4, (e) Pi4+NCS2; (f) averages.

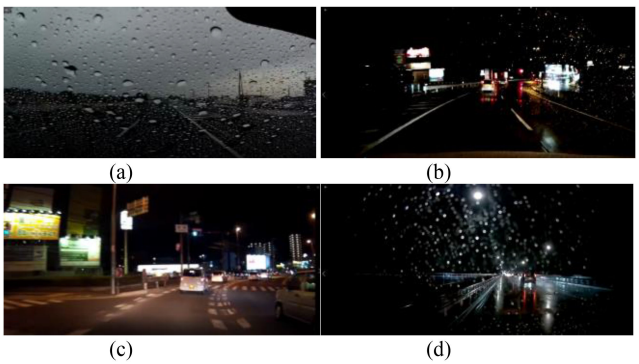

Fig. 12. Sample images of the test cases. Case 3, 4, and 9 are nighttime videos; Case 2 is daytime but very dark. All of them resulted in a relatively low FPS on VM+NCS2. (a) Case 2. (b) Case 3. (c) Case 4. (d) Case 9.

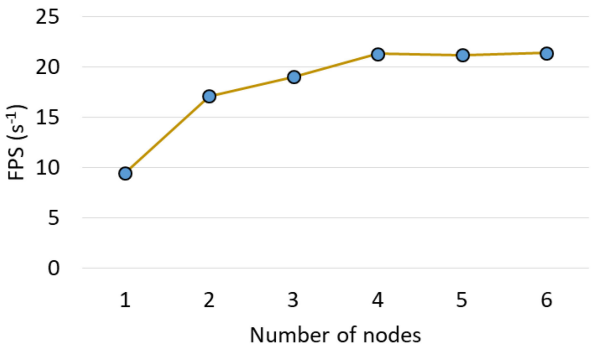

Fig. 13. FPS results of the computing clusters constructed with one to six sets of Pi4+NCS2.

generates a frame rate similar to a PC and its small size satisfies the need for an embedded system. The same real-time rain detection setup was adopted for testing the computing clusters. The average FPS of the cluster with different numbers of nodes was measured and discussed; the results are shown in Fig. 13. It was found that the computation speed increases with the number of nodes and most significantly when the computing cluster expands from one node to two nodes, while FPS almost doubles. However, a further increase in the number of nodes does not linearly increase FPS. With four Pi4s, FPS reaches its maximum of 22; further increase in the total nodes does not enhance FPS at all. With such speed performance, the cluster of Pi4+NCS2 clearly surpasses the PC in our test.

B. Rain Detection Accuracy

1) Various Computational Platforms: Fig. 14 shows the accuracy results of various computational platforms testing different dashcam videos. Some level of difference in accuracy exists among different films, indicating the differences in the nature of the videos, in which some are more difficult to be detected correctly than the others. It is interesting to note that NCS2 appears to enhance the average accuracy as well; the accuracy of Pi4 increases from 68% to 82% and VM increases from 75% to 78% with the help of NCS2. It appears that the inclusion of NCS2 increases the computation resources and not only enhances the processing speed but also the overall accuracy. Similar results are also found in precision and recall as shown in Fig. 15.

2) Embedded Computing Cluster: Two films containing rainy scenes were used for testing the recall performance of the computing clusters. One contains the footage that results in an average recall rate of 95% on a single Pi4; the other results in much lower recalls of 40% on a single Pi4. We intend to understand how the number of computing nodes would affect the recall rates on the scenes that present different levels of detection difficulty. The test results show that the number of computing nodes undeniably affects the rain recall rates as well as FPS (see Fig. 16). For the data with higher initial recalls of around 96%, recall increases to around 100% with four or more nodes; for the data with lower initial recalls, recall increases from 43% of one node to 61% of six nodes.

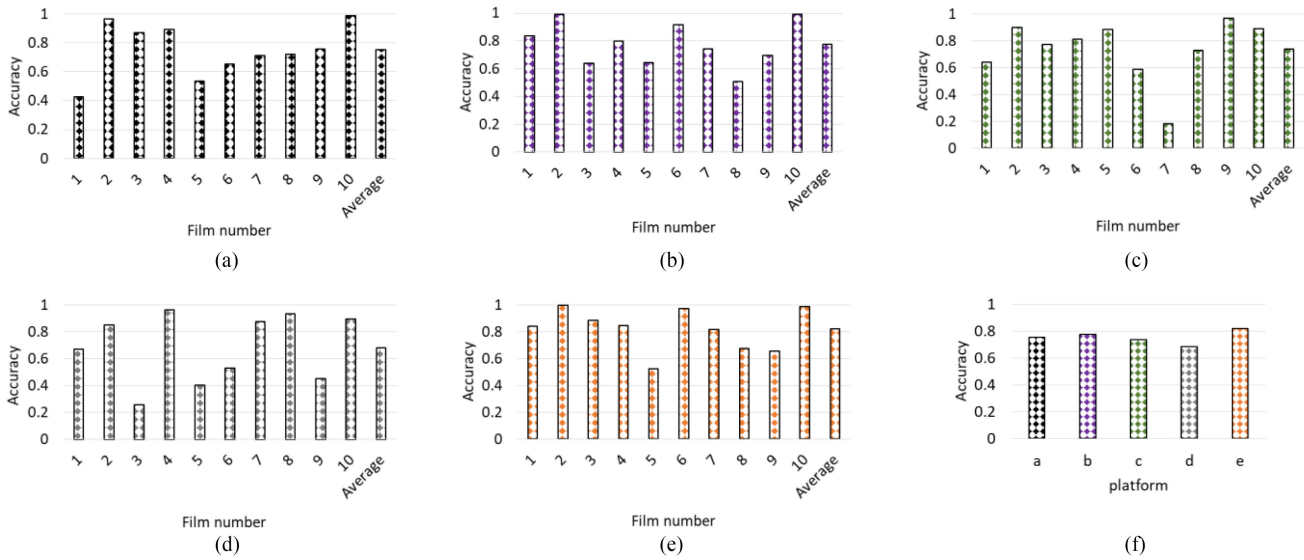

Fig. 14. Accuracy results of ten dashcam videos by five different computation platforms: (a) VM, (b) VM+NCS2, (c) Jetson Nano, (d) Pi4, (e) Pi4+NCS2; (f) average of each platform.

TABLE III
COMPARISON OF THE PRECISION AND RECALL RATES BETWEEN THE STATE-OF-THE-ARTS AND OUR RESULTS

	Kurihata1 [10] (sky only)	Kurihata2 [10] (all background)	Kurihata3 [10] (time-series)	Roser & Geiger [11] (raindrops)	Lai & Li [4] PC	Our results VM+NCS2	Our results Pi4+NCS2 (single set)
Precision	0.97	0.54	0.97	0.80	0.87	0.89	0.92
Recall	0.59	0.59	0.51	0.67	0.82	0.72	0.76

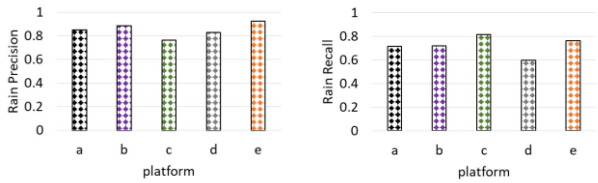

Fig. 15. Average precision and recall of the tested computation platforms: a. VM, b. VM+NCS2, c. Jetson Nano, d. Pi4; e. Pi4+NCS2.

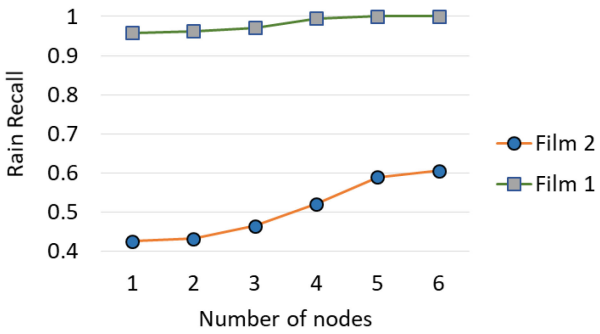

Fig. 16. Recall results of the computing cluster made of various numbers of Pi4+NCS2: film 1 contains high-recall rain scenes; film 2 contains low-recall rain scenes.

VI. DISCUSSION

We compare state-of-the-art methods for rain detection and wiper control in Table III. One may find that a single set of Pi4+NCS2 can generate precision and recall results similar to using a PC in [4] and better than the previous state-of-the-art methods. It confirms that our deep learning rain detection scheme can be satisfactorily executed on some commonly available embedded systems. Note that the reason why our recalls in Table III are somewhat lower than [4] maybe because we captured images in real-time from the monitor for detection and lost some visual quality, while the images for detection in [4] were directly obtained from the videos.

TABLE IV
SPECIFICATIONS OF THE EMBEDDED SYSTEMS AND THE PC IN THE STUDY

Item	Price (USD)	Dimensions (cm×cm×cm)	Mass (g)	Power (W)
PC	830	29×9×29	5260	260
PC+NCS2	912	35×9×29	5310	260
Pi4	54	4.5×8.5×2.5	46	2.1~6.4
Pi4+NCS2	136	4.5 × 14 ×2.5	96	~6.4
Jetson Nano	126	10×8×3	130	5~10
Cluster(4 Pi4+NCS2)	584	19×18×9	1450	~28

The embedded systems require less space and power and are often lighter and more cost-effective. In Table IV, we list the specifications of various embedded systems and the PC as a benchmark in terms of price, dimensions, mass, and power consumption, respectively. We then compare the FPS and accuracy of these systems normalized by each specific aspect; the results are shown in Fig. 17. One may find that Pi4+NCS2 presents

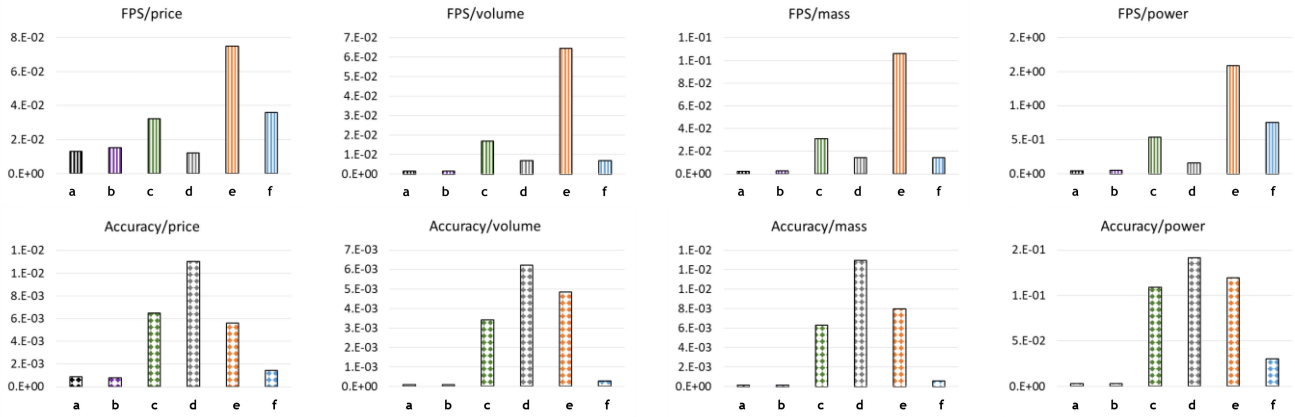

Fig. 17. Normalized FPS and accuracy where each pillar represents the result of a system denoted by a letter of the alphabet: a. VM, b. VM+NCS2, c. Jetson Nano, d. Pi4, e. Pi4+NCS2, and f. computing cluster with four sets of Pi4+NCS2.

Fig. 18. Samples of the detection results. The proposed visual detector is capable of detecting various rainy and fair conditions. (a) DW samples. (b) NW sample.

the highest normalized FPS in all four categories, while single Pi4 presents the highest normalized accuracy. In all categories, PC and PC+NCS2 did not show sufficient competitiveness over the other embedded systems. In terms of the performance-price ratio, Pi4+NCS2 is 4.9 times higher than PC+NCS2 regarding FPS and 7.1 times regarding accuracy. Although a single Pi4 presents high competitiveness in accuracy, its FPS performance is not sufficient; with an average of 1 FPS, it cannot execute the deep neural network in real-time. Pi4+NCS2 appears to be the best choice comparing all embedded systems and the PC-based system. Jetson Nano is also a quite competitive system; however, its FPS performance is substantially lower than Pi4+NCS2. The computing cluster is effective at pursuing higher FPS and accuracy performance (see Fig. 13 and Fig. 16); it also surpasses a PC in regards to price, volume, mass, and power consumption. However, while evaluating the performance-aspect ratios, the cluster is not as good as a single Pi4+NCS2.

VII. Conclusion

A global-feature visual approach was adopted for rain detection on the windshield; the deep learning framework can detect various types of rain conditions and output detection scores for wiper control (see Fig. 18). Some embedded computation platforms were tested for their performance on the execution of deep ConvNet. It was found that NCS2 effectively accelerates the visual detection program; particularly when it is used to accelerate Pi4. With the help of NCS2, Pi4 can generate an average detection speed of 10 FPS, compared to the 11 FPS of a PC. Experimental evidence also shows that the detection accuracy can be enhanced by NCS2 by up to 20%. The detection performances are similar to our previous results obtained from an ordinary PC and better than the other state-of-the-art methods. We also constructed computing clusters based on Pi4+NCS2. Test results showed that a cluster of four sets of Pi4+NCS2 further increased the computation speed to 20 FPS; rain recall was also substantially increased, particularly on the footage that originally resulted in low recalls. We release the dataset of the 160 k images used for the ConvNet training; the dataset can be reached at https://github.com/ntutindustry40/raindetection.

Acknowledgment

The authors would like to thank the Ministry of Science and Technology of the Republic of China, Taiwan, for supporting this research under Contract No.: MOST109-2637-E-027-008.

REFERENCES

[1] S. Hasirlioglu, F. Reway, T. Klingenberg, A. Riener, and W. Huber, "Raindrops on the windshield: Performance assessment of camera-based object detection," in *Proc. IEEE Int. Conf. Veh. Electron. Saf.*, 2019, pp. 1–7.

[2] Admin, Mercedes-Benz of Arrowhead, "How do rain-sensing wipers work?," Accessed: Aug. 13, 2020. [Online] Available: https://www.arrowheadmb.com/blog/how-do-rain-sensing-wipers-work/

[3] Technology, Toyota, "How rain-sensing wipers detect a downpour," Accessed: Aug. 13, 2020. [Online]. Available: https://www.toyota.ca/toyota/en/connect/1893/rain-sensing-wipers-rav4-highlander-sienna

[4] C. Lai and C. G. Li, "Video-based windshield rain detection and wiper control using holistic-view deep learning," in *Proc. IEEE Int. Conf. Automat. Sci. Eng.*, 2019, pp. 1060–1065.

[5] OpenRoad Auto Group, Audi Boundary, "How rain-sensing wipers work," Accessed: March, 2021. [Online]. Available: https://openroadaudi.com/blog/how-rain-sensing-wipers-work

[6] M. Jarajreh, A. L. Nortcliffe, and R. Green, "Fuzzy logic and equivalent circuit approach to rain measurement," *Electron. Lett.*, vol. 40, no. 24, pp. 1533–1534, 2004.

[7] L. Alazzawi and A. Chakravarty, "Design and implementation of a reconfigurable automatic rain sensitive windshield wiper," *Int. J. Adv. Eng. Technol.*, vol. 8, no. 2, pp. 73–82, 2015.

[8] A. H. M. F. Elahi and M. S. Rahman, "Intelligent windshield for automotive vehicles," in *Proc. 17th Int. Conf. Comput. Inf. Technol.*, 2014, pp. 392–396.

[9] S. Gormer, A. Kummert, S.-B. Park, and P. Egbert, "Vision-based rain sensing with an in-vehicle camera," in *Proc. IEEE Intell. Veh. Symp.*, 2009, pp. 279–284.

[10] H. Kurihata, T. Takahashi, Y. Mekada, H. Murase, Y. Tamatsu, and T. Miyahara, "Detection of raindrops on windshield from an in-vehicle video camera," *Int. J. Innov. Comput. Inf. Control*, vol. 3, no. 6, pp. 1583–1591, 2007.

[11] M. Roser and A. Geiger, "Video-based raindrop detection for improved image registration," in *Proc. IEEE 12th Int. Conf. Comput. Vis. Workshops*, 2009, pp. 570–577.

[12] S. You, R. T. Tan, R. Kawakami, and K. Ikeuchi, "Adherent raindrop detection and removal in video," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2013, pp. 1035–1042.

[13] A. Yamashita, I. Fukuchi, and T. Kaneko, "Noises removal from image sequences acquired with moving camera by estimating camera motion from spatio-temporal information," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2009, pp. 3794–3801.

[14] J. Ishizuka and K. Onoguchi, "Raindrop detection on a windshield based on edge ratio," in *Int. Conf. Pattern Recognit. Appl. Methods*, pp. 212–229, 2017. [Online]. Available: https://doi.org/10.1007/978-3-319-53375-9_12

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[16] C. G. Li and Y. Chang, "Automated visual positioning and precision placement of a workpiece using deep learning," *Int. J. Adv. Manuf. Technol.*, vol. 104, pp. 4527–4538, 2019.

[17] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 779–788.

[18] D. Eigen, D. Krishnan, and R. Fergus, "Restoring an image taken through a window covered with dirt or rain," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2013, pp. 633–640.

[19] R. Qian, R. T. Tan, W. Yang, J. Su, and J. Liu, "Attentive Generative Adversarial Network for raindrop removal from a single image," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 2482–2491.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[21] Nvidia Developer, "Jason Nano Developer Kit," Accessed: Aug. 18, 2020. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit

[22] P. Mell and T. Grance, NIST, Department of Commerce, "The NIST Definition of Cloud Computing," *Recommendations Nat. Inst. Standards Technol.*, pp. 1–7, Sep. 2011. Accessed: Aug. 4, 2020. [Online]. Available: http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf

Chih-Hung G. Li (Member, IEEE) received the B.S. degree in power mechanical engineering from National Tsing Hua University, Hsinchu, Taiwan, in 1990, and the M.S. and Ph.D. degrees in mechanical engineering from Carnegie Mellon University, Pittsburgh, PA, USA, in 1994 and 1998, respectively.

He is currently an Associate Professor with the Graduate Institute of Manufacturing Technology, National Taipei University of Technology, Taipei, Taiwan and was the Associate Dean with the College of Mechanical and Electrical Engineering. From 2014 to 2017, he was the Director of Automated Vehicle and Equipment Development Center in charge of the development of an innovative personal rapid transit system. He was also involved in multitudes of development projects including novel actuators, intelligent mobile robots, precision object localization and manipulation, and visual detection. His research interests include the areas of intelligent robotic and transit systems, innovative mechanisms, optimization of mechanical systems, and applied mechanics.

Kuei-Wen Chen received the B.S. degree from Mechanical Engineering Department, Taipei City University of Science and Technology, Taipei, Taiwan. He is currently working toward the M.S. degree with the Graduate Institute of Mechatronics Engineering, National Taipei University of Technology, Taipei, Taiwan. His primary research interests include the development of mechatronic systems with embedded systems and the application of artificial intelligence. He has much hands-on experience in the development of Linux-based clusters and software design for automation using Python and Shell Scripts.

Chi-Cheng Lai received the B.S. degree from Mechanical Engineering Department, Chung Yuan Christian University, Zhongli, Taiwan and the M.S. degree from the Graduate Institute of Mechatronics Engineering, National Taipei University of Technology, Taipei, Taiwan. He is currently with Compal Electronics, Inc. as a Software Engineer in charge of the development of AI applications. His research interests include AI application development and engineering analysis.

Yu-Tang Huang received the B.S. degree in mechanical engineering from Tatung University, Taipei, Taiwan, in 2015. He is currently working toward the M.S. degree with the Graduate Institute of Manufacturing Technology, National Taipei University of Technology, Taipei, Taiwan. His primary research interests include the application of artificial intelligence, digital image processing, and the design of innovative mechatronic systems. He is also involved in the Industry 4.0 development project with the WONDERLAND GROUP, Taipei, Taiwan.