applied sciences

Article

# EmbeddedPigDet—Fast and Accurate Pig Detection for Embedded Board Implementations

**Jihyun Seo [1], Hanse Ahn [1], Daewon Kim [1], Sungju Lee [2,*], Yongwha Chung [1,*] and Daihee Park [1]**

[1] Department of Computer Convergence Software, Korea University, Sejong 30019, Korea; goyangi100@korea.ac.kr (J.S.); hansahn@korea.ac.kr (H.A.); noin1996@korea.ac.kr (D.K.); dhpark@korea.ac.kr (D.P.)

[2] Department of Software, Sangmyung University, Cheonan 31066, Korea

* Correspondence: peacfeel@smu.ac.kr (S.L.); ychungy@korea.ac.kr (Y.C.)

Received: 16 March 2020; Accepted: 11 April 2020; Published: 22 April 2020

**Abstract:** Automated pig monitoring is an important issue in the surveillance environment of a pig farm. For a large-scale pig farm in particular, practical issues such as monitoring cost should be considered but such consideration based on low-cost embedded boards has not yet been reported. Since low-cost embedded boards have more limited computing power than typical PCs and have tradeoffs between execution speed and accuracy, achieving fast and accurate detection of individual pigs for "on-device" pig monitoring applications is very challenging. Therefore, in this paper, we propose a method for the fast detection of individual pigs by reducing the computational workload of $3 \times 3$ convolution in widely-used, deep learning-based object detectors. Then, in order to recover the accuracy of the "light-weight" deep learning-based object detector, we generate a three-channel composite image as its input image, through "simple" image preprocessing techniques. Our experimental results on an NVIDIA Jetson Nano embedded board show that the proposed method can improve the integrated performance of both execution speed and accuracy of widely-used, deep learning-based object detectors, by a factor of up to 8.7.

**Keywords:** agriculture IT; computer vision; pig detection; embedded board; image preprocessing; light-weight deep learning; YOLO; TinyYOLO

## 1. Introduction

The early detection and management of problems related to the health and welfare of individual livestock are important issues [1–5]. Pigs, in particular, are highly susceptible to many diseases and stresses due to many pigs in a "closed" pig room. In Korea, for example, five million pigs out of 20 million pigs die every typical year, according to statistics published by the Korean government [6]. Therefore, it is essential to minimize the potential problems (i.e., infectious diseases, hygiene deterioration, etc.) with individual pigs. However, farms generally have few farm workers relative to the number of pigs in a pig farm. For example, the pig farm from which we have obtained video monitoring data in Korea in the past had more than 2000 pigs per farm worker. It is almost impossible to take care of many individual pigs with a small number of farm workers. Therefore, the basic purpose of this solution is to identify the number of pigs in a room and to prevent deaths of individual pigs caused by potential problems (i.e., infectious diseases, deterioration of hygiene, etc.) using early detection of abnormalities.

Since the early 1990s, many studies have reported the use of surveillance techniques to solve the health and welfare problems in a pig room [7–9]. By using these camera-based surveillance techniques, for example, we can recognize the biter and the victim piglets of tail-biting in order to reduce the damage of the aggressive behavior. For analyzing this type of motion behavior, it is essential to detect individual pigs in each video frame because object detection is the first step for various types of vision-based high-level analysis. Although many researchers have reported the detection of pigs with typical learning and image processing techniques, the detection accuracy for highly occluded images may not be acceptable. Recently, end-to-end deep learning techniques have been proposed for object detection and thus some deep learning-based pig detection results have been reported very recently, in addition to results using typical learning and image processing techniques (see Table 1) [10–48]. However, "end-to-end" deep learning techniques that accept the input images directly without any image processing steps require a large number of parameters and heavy computational workload. For continuous video monitoring, such as various high-level analysis of pig monitoring, processing each video frame without delay is required. In the process of a large-scale installation, the problem of increased network bandwidth and heavy analyzation workload of the server arises due to the server requiring large-scale image data (i.e., video stream). In order to provide a practical method for pig monitoring system, it is efficient to analyze the data in a built-in device system that supports small-sized data processing after the data has been gathered [49]. Furthermore, we should consider practical issues, such as the monitoring cost, for large-scale pig farms. In Korea, for example, there is a large-scale pig farm having about 1000 pig rooms and the farm owner emphasized to us of the "installation/maintenance cost" in applying any vision-based monitoring solution to his farm. Additionally, due to the severe "ammonia gas" in a closed pig room, many pigs die from wasting disease and any PCB board will be corroded faster than normal monitoring environments. Therefore, a low-cost solution (rather than a typical PC-based solution) is required for "practical" monitoring of a pig room. Considering the effect of both the reduction in installation and the management costs through edge computing such as that shown in Reference [49], we can extend the solution for many high-level vision-based analyses, such as aggressive behavior analysis, in order to reduce damage to a pig farm by using a single embedded board.

In this study, we focus on detecting individual pigs with a low-cost embedded board to analyze individual pigs cost effectively, with the ultimate goal of 24 h monitoring in a large-scale pig farm. We propose first a "light-weight" version of a deep learning technique for detecting pigs with a low-cost embedded board. However, such light-weight object detector may not satisfy the accuracy requirement. To improve the accuracy of the light-weight object detector, we "preprocess" an input image in order to generate a three-channel composite image for the light-weight object detector, using simple image processing techniques. By effectively combining light-weight image processing and deep learning techniques, we can simultaneously satisfy both execution speed and accuracy requirements with a low-cost embedded board. The contribution of the proposed method can be summarized as follows:

- Individual pigs are detected with a low-cost embedded board, such as an NVIDIA Jetson Nano [50]. Although many pig detection methods have been proposed with typical PCs, an embedded board-based pig detection method is proposed here, to the best of our knowledge for the first time. Since low-cost embedded boards have more limited computing power than typical PCs, fast and accurate detection of individual pigs for low-cost pig monitoring applications is very challenging. Because this research direction for a light-weight pig detector is a kind of "on-device" AI [51–55], it can also contribute to the on-device AI community.
- For satisfying both execution speed and accuracy requirements with a low-cost embedded board, we first reduce the computational workload of $3 \times 3$ convolution of a deep learning-based object detector, in order to get a light-weight version of it. Then, with simple image preprocessing steps, we generate a three-channel composite image as an input image for the light-weight object detector, in order to improve the accuracy of the light-weight object detector. With this fast and accurate pig detector, we can integrate additional high-level vision tasks for continuous monitoring of individual pigs, in order to reduce the damage of a pig farm.

This paper is organized as follows: Section 2 summarizes previous pig detection methods. Section 3 describes the proposed method to detect pigs efficiently by using light-weight image processing and deep learning techniques. Section 4 explains the details of the experimental results, while Section 5 concludes the paper.

## 2. Background

The final goal of this study is to analyze the behavior of individual pigs automatically over 24 h in a cost-effective manner. In general, a low-cost camera, such as the Intel RealSense camera [56], can be used for the individual pig detection with color, infrared and depth input images. However, we cannot guarantee to get the color input image at night because many pig farms turn off the light at night. Therefore, we exclude the color image in this study for 24-h monitoring. Furthermore, the accuracy of each input image obtained from the low-cost camera may not be satisfactory for accurate pig detection. Figure 1b shows the detection results by adaptive thresholding technique [57] with the infrared and depth input images. Note that depth images produce inferior results than those with infrared images. Therefore, we use infrared input image to detect the individual pigs at daytime and nighttime. Note that, a typical color image such as RGB consists of three-channels (i.e., 24 bits) information while an infrared image consists of one-channel (i.e., 8 bits) information. From the one-channel infrared input image, we can carefully generate a three-channel "composite" image which can contain more useful information to detect individual pigs. The image preprocessing steps to generate a three-channel composite image will be described in Section 3.1.

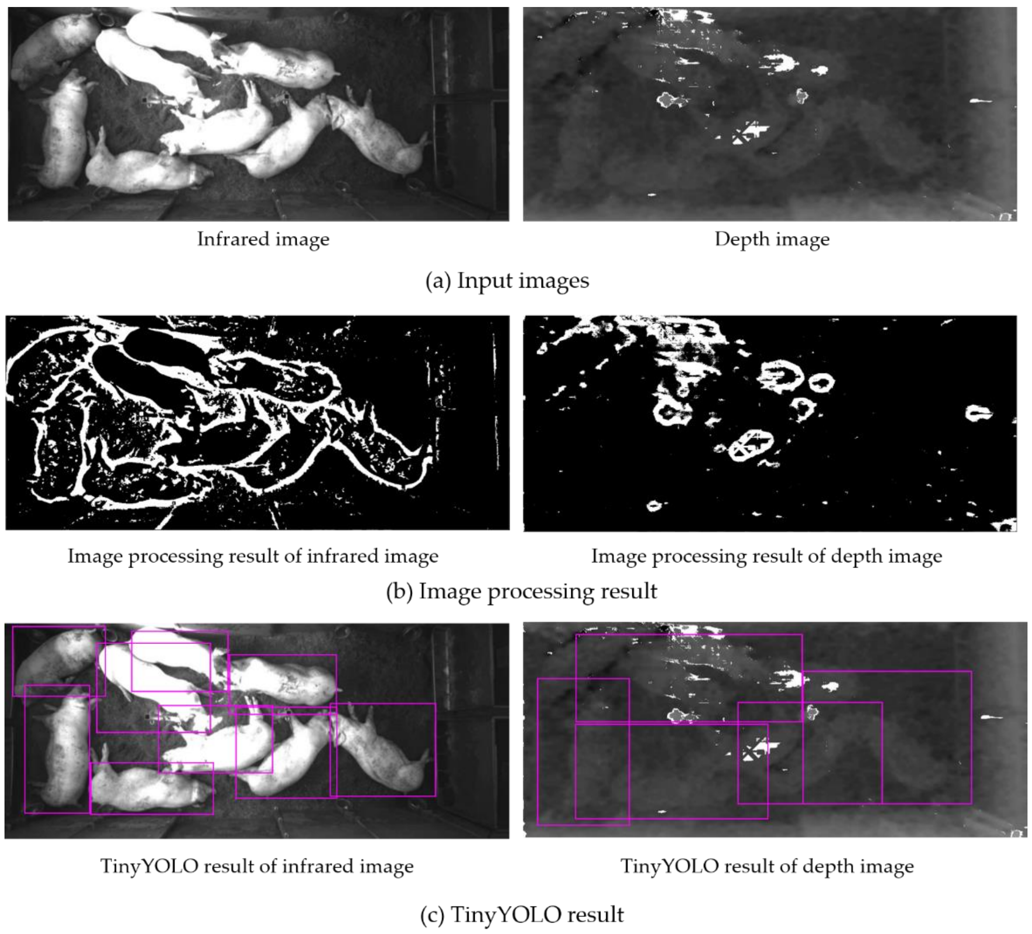

Infrared image
Depth image
(a) Input images
Image processing result of infrared image
Image processing result of depth image
(b) Image processing result
TinyYOLO result of infrared image
TinyYOLO result of depth image
(c) TinyYOLO result

**Figure 1.** The foreground detection results of the infrared and depth input images with adaptive thresholding [57] and TinyYOLO [58] techniques. Some pig parts are missed, while some background parts are included. Furthermore, it is difficult to detect each individual pig among the pigs in contact with each other. (**a**) Input images (**b**) Image processing results (**c**) TinyYOLO results.

Recently, end-to-end deep learning techniques have been widely applied to many image processing problems. For example, CNN-based methods, such as R-CNN [59], Fast R-CNN [60] and Faster R-CNN [61], have been proposed for end-to-end object detectors. In particular, You Only Look Once

(YOLO) [58] has been proposed as a single-shot object detector, in order to improve the execution times of those many-shot object detectors, such as R-CNN, Fast R-CNN and Faster R-CNN. TinyYOLO [58] is a tiny version of YOLO that has been developed for embedded board implementations. Figure 1c shows the detection results of individual pigs with TinyYOLO. In the results, many pigs are detected correctly but some pigs are missed with both the color and infrared input images. Therefore, we should consider the complementary information to improve the detection accuracy of the individual pigs.

Table 1 summarizes the previous methods for pig detection [10–48]. Due to individual pigs having high mortality rate, the owners face difficulty at identifying exact number of pigs which lead to management problems. Considering how this may result in various potential problems (i.e., contagious infection and hygiene deterioration), therefore, removing it in the early stage is ideal. Hence, the primary purpose of solving this issue would be identification of the number of pigs and mortality prevention using early detection of abnormalities. For previous research on identification, References [11,16–18,20–22,24–26,29,31,38,44–48] for detection and References [10,19,37,39,45] for tracking exist. In addition, previous studies for early detection of abnormalities exist as various topics, including research on the movement of pigs [17,62], research on aggressive behavior of pigs [63,64], research on attitude change [16,22,23,31,32,34,35,40,46], research on mounting behavior [21], research on low-growth pig's behavior [49], research on pig weight [29,33,38] and research on the density of pigs [9,11].

It is also essential to meet the execution speed requirement, in order to process the successive video frames without delay. However, many previous methods did not report the execution times. Among the previous methods of reporting the execution times, only the deep learning-based method [44] can satisfy both requirements of individual pig detection. However, none of the previous methods reported the pig detection using embedded boards. In a large-scale pig farm, a "cost effective" solution is definitely required. As explained, a low-cost solution (rather than a typical PC-based solution) is additionally required due to the severe ammonia gas in a closed pig room. However, since low-cost embedded boards have more limited computing power than typical PCs, the fast and accurate detection of individual pigs for "on-device" pig monitoring applications is very challenging.

To the best of our knowledge, this paper is the first report on how to improve the accuracy of individual pig detection with an embedded board, while satisfying the execution speed requirement, by using the complementary techniques of light-weight image processing and deep learning. That is, we first remove less-important 3 × 3 convolutional filters of a deep learning–based object detector, in order to obtain a light-weight version of it. Then, we generate a three-channel composite image as an input image for the light-weight object detector, in order to improve its accuracy.

**Table 1.** Some of the recent pig detection results (published during 2010–2019).

| Target Platform | Data Size | No. of Pigs in a Pen | Detection Technique | Individual Detection of Pigs | Execution Time (ms) | Reference |
|---|---|---|---|---|---|---|
| PC | Not Specified | Not Specified | Image Processing | No | Not Specified | [10] |
| | 720 × 540 | 12 | Image Processing | Yes | 220 (PC) | [11] |
| | 768 × 576 | Not Specified | Image Processing | No | 1000 (PC) | [12] |
| | 768 × 576 | Not Specified | Image Processing | No | 500 (PC) | [13] |
| | 150 × 113 | Not Specified | Image Processing | No | 250 (PC) | [14] |
| | 640 × 480 | 9 | Learning | No | Not Specified | [15] |
| | 720 × 576 | 9 | Image Processing | Yes | Not Specified | [16] |
| | 1280 × 720 | 7–13 | Image Processing | Yes | Not Specified | [17] |
| | Not Specified | 3 | Image Processing | Yes | Not Specified | [18] |
| | 352 × 288 | Not Specified | Learning | No | 236 (PC) | [19] |
| | 640 × 480 | 22–23 | Image Processing | Yes | Not Specified | [20] |
| | 640 × 480 | 22 | Image Processing | Yes | Not Specified | [21] |
| | Not Specified | 17–20 | Image Processing | Yes | Not Specified | [22] |
| | 256 × 256 | Not Specified | Image Processing | No | Not Specified | [23] |

**Table 1.** *Cont.*

| Target Platform | Data Size | No. of Pigs in a Pen | Detection Technique | Individual Detection of Pigs | Execution Time (ms) | Reference |
|---|---|---|---|---|---|---|
| | 1760 × 1840 | Not Specified | Image Processing | Yes | Not Specified | [24] |
| | 1280 × 720 | 23 | Image Processing | Yes | 971 (PC) | [25] |
| | Not Specified | 2–12 | Image Processing | Yes | Not Specified | [26] |
| | 320 × 240 | Not Specified | Image Processing | No | Not Specified | [27] |
| | 512 × 424 | Not Specified | Image Processing | No | Not Specified | [28] |
| | 1440 × 1440 | Not Specified | Image Processing | Yes | 1606 (PC) | [29] |
| | 960 × 540 | 1 | Deep Learning | No | Not Specified | [30] |
| | 2560 × 1440 | 4 | Deep Learning | Yes | Not Specified | [31] |
| | Not Specified | 1 | Image Processing | No | Not Specified | [32] |
| | 640 × 480 | Not Specified | Image Processing | No | Not Specified | [33] |
| | 512 × 424 | 1 | Image Processing | No | Not Specified | [34] |
| | 512 × 424 | Not Specified | Image Processing | No | Not Specified | [35] |
| | 512 × 424 | 1 | Image Processing | No | Not Specified | [36] |
| | 1294 × 964 | 1 | Image Processing | No | Not Specified | [37] |
| | 512 × 424 | 19 | Image Processing | Yes | 142 (PC) | [38] |
| | 512 × 424 | 1 | Deep Learning | No | 50 (PC) | [39] |
| | 512 × 424 | 22 | Image Processing | No | 56 (PC) | [40] |
| | 512 × 424 | 13 | Image Processing | No | 2 (PC) | [41] |
| | 640 × 480 | 22, 24 | Image Processing | No | 60 (PC) | [42] |
| | 1280 × 720 | 9 | Image Processing | No | 8 (PC) | [43] |
| | 1920 × 1080 | 9 | Deep Learning | Yes | 42 (PC) | [44] |
| | 960 × 720 | ~30 | Image Processing | Yes | Not Specified | [45] |
| | 1920 × 1080 | Not Specified | Deep Learning | Yes | 250 (PC) | [46] |
| | 1024 × 768 | 4 | Image Processing | Yes | 921 (PC) | [47] |
| | Not Specified | Not Specified | Deep Learning | Yes | 500 (PC) | [48] |
| Embedded Board | 1280 × 720 | 9 | Image Processing Deep Learning | Yes | **29.08 * (Nano)** | Proposed Method |

* computed with a pipelined implementation.

## 3. Proposed Method

In this study, we selected TinyYOLO [58] as our base model and improved both of its execution speed and accuracy. As explained in Section 2, TinyYOLO is a tiny version of YOLO [58] which is a widely used single-shot object detector. Although TinyYOLO is targeted for embedded board implementations, its computational workload cannot satisfy the execution speed requirement for our target embedded board such as Jetson Nano [50]. Furthermore, it is well known that reducing the computational workload of tiny networks without degrading accuracy is much more difficult than that of larger networks [55].

In order to reduce the computational workload of TinyYOLO, we first apply the filter clustering for each 3 × 3 convolutional layer of TinyYOLO and group them into a cluster having the maximum convolution value. In fact, the proposed filter clustering (FC) algorithm is a kind of filter pruning techniques [65] widely used for compressing deep networks. Unlike previous filter pruning techniques [65–71], however, the proposed FC algorithm can determine a pruning rate for each 3 × 3 convolutional layer separately and systematically. Then, we apply the bottleneck structuring (BS) algorithm to the result of the FC algorithm to obtain EmbeddedPigYOLO (i.e., light-weight version of TinyYOLO). For example, we apply the bottleneck structure [72] by replacing each 3 × 3 × $i$ convolutional filter with 1 × 1 × $i$/4, 3 × 3 × $i$/4 and 1 × 1 × $i$ convolutional filters, in order to reduce the computational workload of each 3 × 3 × $i$ convolutional filter further.

After reducing the computational workload of TinyYOLO, we need to improve the accuracy of EmbeddedPigYOLO. Our idea is to use the otherwise-idle CPU to allow EmbeddedPigYOLO to focus on individual pigs with the image preprocessing (IP) module. That is, in order to recover the accuracy of EmbeddedPigYOLO which is executed on GPU, we use CPU to generate a three-channel composite image as an input image for EmbeddedPigYOLO. In this study, the composite image is generated for focusing on the possible pig regions in a pig room. That is, the three-channel composite image can give complementary information to EmbeddedPigDet (i.e., IP + EmbeddedPigYOLO) and thus let

EmbeddedPigDet focus on individual pigs. Note that the computing power of an embedded CPU in a Nano is lower than that of a CPU in a typical PC. Based on our previous studies, we carefully evaluate each image preprocessing step, in order to understand the execution speed-accuracy tradeoff in the IP module.

Figure 2 shows the whole process of detecting individual pigs in embedded board environments. Note that the FC and BS algorithms are executed once during the training phase in order to determine the number of convolutional filters in EmbeddedPigYOLO and thus the IP module and EmbeddedPigYOLO are executed during the test phase.

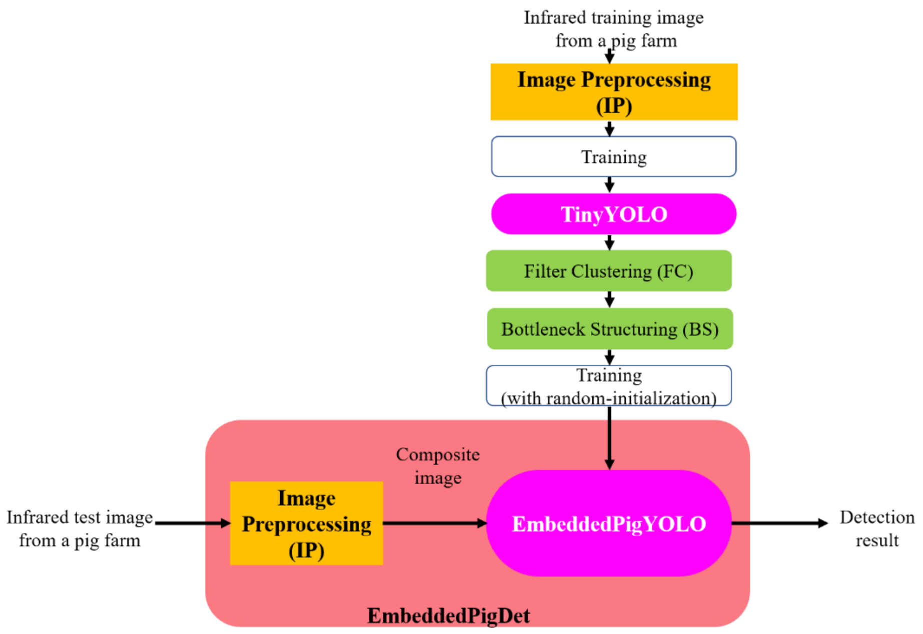


**Figure 2.** Overview of the proposed method EmbeddedPigDet consisting of image preprocessing (IP), filter clustering (FC) and bottleneck structuring (BS).

## *3.1. Image Preprocessing (IP) Module*

The objective of this module is to generate attention information to allow EmbeddedPigYOLO to focus on individual pigs and reduce the effect of illumination variation. Figure 3 shows the infrared input images at different illumination conditions (i.e., at 2 AM and 8 AM). Even with the infrared input images, the gray values at 2 AM are generally darker than those at 8 AM. Furthermore, the gray values of a pig at 2 AM are too dark to detect foreground (i.e., pig) from background, whereas strong sunlight through a window at 8 AM generates many illumination noises and deletes the texture information of some pigs. By reducing the effect of this illumination variation, we want to separate pigs from background (i.e., foreground detection) and separate individual pigs from a pig group (i.e., outline detection).

In this study, we consider two basic image preprocessing steps to help EmbeddedPigYOLO detect individual pigs. That is, we apply the contrast-limited adaptive histogram equalization (CLAHE) technique [73] in order to focus on the possible pig regions in a pig room. From the infrared input images having illumination variation, the objective of this technique is to maximize the "inter-class" variation (i.e., the gray values between pigs and background should be different) and minimize the "intra-class" variation (i.e., the gray values of pigs should be similar, regardless of its observed location and time) simultaneously. Therefore, we apply CLAHE twice with different parameter values in order to maximize the inter-class variation (block size = $2 \times 2$, clip limit = 160 and denoted as CLAHE1) and minimize the intra-class variation (block size = $16 \times 16$, clip limit = 12 and denoted as CLAHE2).

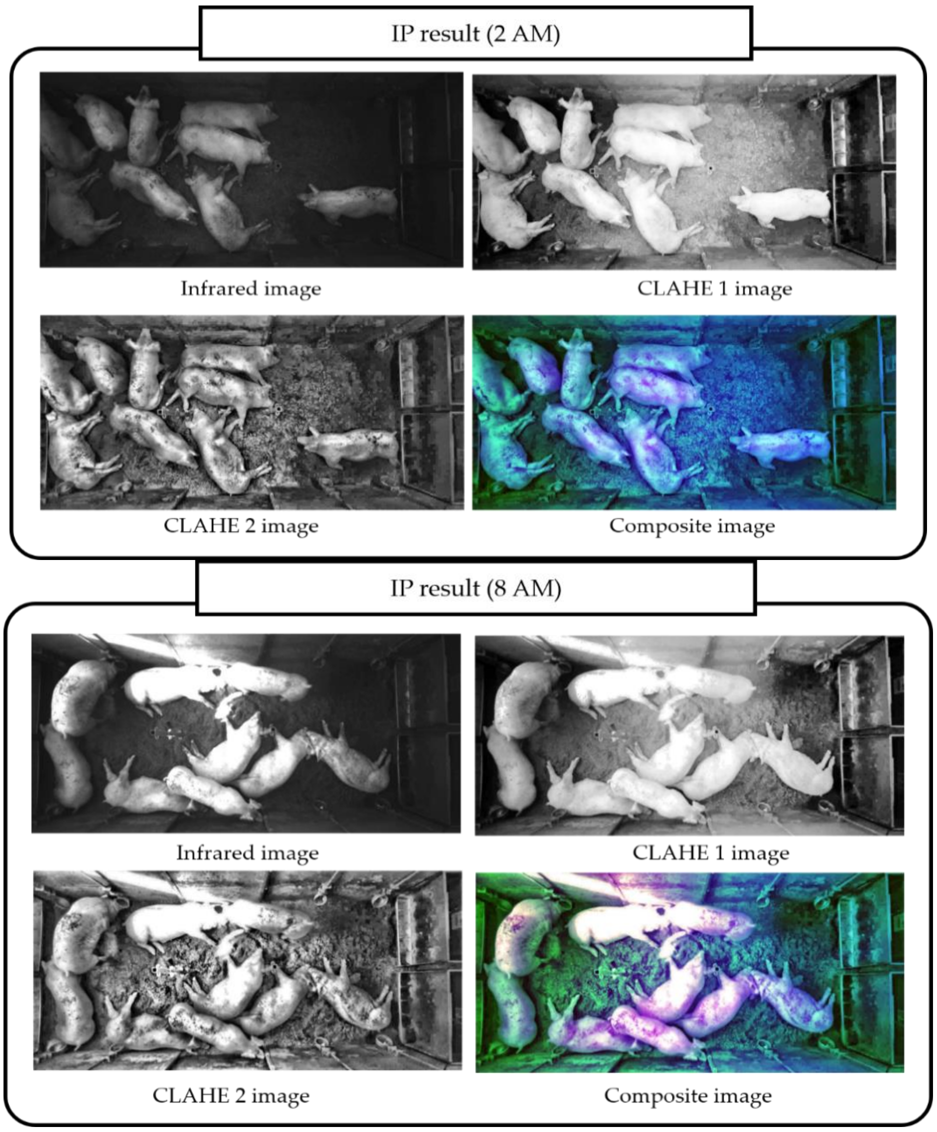


**Figure 3.** Infrared, CLAHE1, CLAHE2 and composite images at different illumination conditions.

Finally, these preprocessed images are concatenated with the infrared input image in order to generate a three-channel composite image, which is used as an input image for EmbeddedPigYOLO. As shown in Figure 3, the composite image is less affected by the illumination conditions, compared to the infrared input image. Although the qualities of these image preprocessing steps are not ideal, we can utilize this complementary and attention information to improve the accuracy of EmbeddedPigYOLO. Furthermore, with a pipeline execution between CPU and GPU, the additional CPU time for this module can be hidden by the GPU time for EmbeddedPigYOLO (see Section 4.3).

### *3.2. Filter Clustering (FC) Module*

As previously explained, we focus on pruning $3 \times 3$ convolutional filters. As each filter in a $3 \times 3$ convolutional layer plays the role of a feature extractor, multiple filters extracting a similar feature can be grouped into the same cluster. For this clustering, we first prepare 511 features, which can be made with a $3 \times 3$ binary pattern. Then, each filter in a $3 \times 3$ convolutional layer is convolved with 511 features and is grouped into a cluster having the maximum convolution value. At the end of the clustering, some clusters may contain multiple filters. We simply select the filter having the maximum convolution value in each cluster containing multiple filters.

As shown in Figure 4, for example, there are 32 filters in a 3 × 3 convolutional layer. Then, #1 filter goes to #2 cluster (i.e., #1 filter has the maximum convolution value with #2 feature and thus goes to #2 cluster), whereas #2 filter and #32 filter go to #511 cluster. Between #2 filter and #32 filter contained in #511 cluster, we simply select #32 filter as its convolution value is larger than that of #2 filter. Through this FC step, we can reduce the number of filters in each convolutional layer.

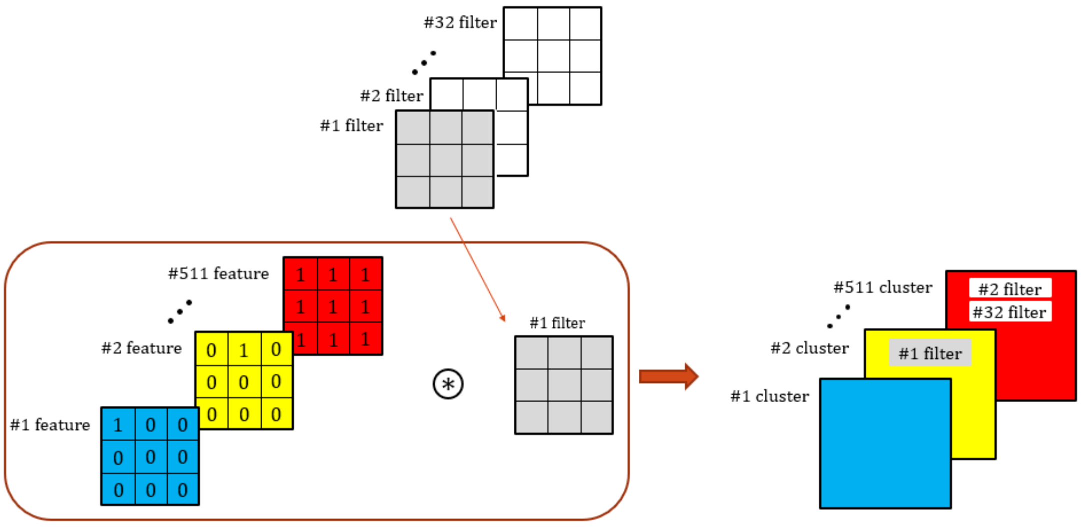


**Figure 4.** Illustration of the filter clustering algorithm.

EmbeddedPigYOLO (with FC) shown in Tables 2 and 3 represents the result of the FC module from TinyYOLOv2 [74] and TinyYOLOv3 [75] with pig training set, respectively. Note that, for the purpose of explanation, we denote the result from TinyYOLOv2 and TinyYOLOv3 as EmbeddedPigYOLO(v2) and EmbeddedPigYOLO(v3), respectively. The previous filter pruning techniques [65–71] removed half of the filters (i.e., 50% pruning rate) from each convolutional layer, regardless of the training set. However, each convolutional layer can have different importance and thus, different numbers of filters may need to be pruned from each convolutional layer. Since the FC module determines a pruning rate for each 3 × 3 convolutional layer separately depending on the training set, the number of filters determined by the FC module can be "odd" numbers (e.g., 61 in Conv3, 197 in Conv5, 349 in Conv7 and 353 in Conv8 shown in Table 2).

## *3.3. Bottleneck Structuring (BS) module*

As previously explained, we apply the bottleneck structure [72] to reduce the computational workload of EmbeddedPigYOLO (with FC) further. In this study, we use the bottleneck structure (by a factor of four). For example, we apply the bottleneck structure by replacing each 3 × 3 × $i$ convolutional filters with 1 × 1 × $i$/4, 3 × 3 × $i$/4 and 1 × 1 × $i$ convolutional filters. Since the number of filters determined by the FC module can be even numbers, we derive the minimum number that can satisfy the bottleneck structure (by a factor of four). For example, for 3 × 3 × 61 convolutional filters in Conv3 shown in Table 2, the BS module generates the result as 1 × 1 × 16, 3 × 3 × 16 and 1 × 1 × 64 convolutional filters.

**Table 2.** Comparison between TinyYOLOv2 [74] and EmbeddedPigYOLO(v2) with pig data set.

| | **TinyYOLOv2 [70]** | **EmbeddedPigYOLO(v2)** | |
|---|---|---|---|
| | **Filter** | **Filter with FC** | **Filter with FC+BS** |
| Conv1 | 3 × 3 (16) | 3 × 3 (16) | 3 × 3 (16) |
| Conv2 | 3 ×3 (32) | 3 × 3 (32) | 1 × 1 (8)<br>3 × 3 (8)<br>1 × 1 (32) |
| Conv3 | 3 × 3 (64) | 3 × 3 (61) | 1 × 1 (16)<br>3 × 3 (16)<br>1 × 1 (64) |
| Conv4 | 3 × 3 (128) | 3 × 3 (114) | 1 × 1 (29)<br>3 × 3 (29)<br>1 × 1 (116) |
| Conv5 | 3 × 3 (256) | 3 × 3 (197) | 1 × 1 (50)<br>3 × 3 (50)<br>1 × 1 (200) |
| Conv6 | 3 × 3 (512) | 3 × 3 (256) | 1 × 1 (64)<br>3 × 3 (64)<br>1×1 (256) |
| Conv7 | 3 × 3 (1024) | 3 × 3 (349) | 1 × 1 (88)<br>3 × 3 (88)<br>1 × 1 (352) |
| Conv8 | 3 × 3 (1024) | 3 × 3 (353) | 1 × 1 (89)<br>3 × 3 (89)<br>1 × 1 (356) |

**Table 3.** Comparison between TinyYOLOv3 [75] and EmbeddedPigYOLO(v3) with pig data set.

| | **TinyYOLOv3 [71]** | **EmbeddedPigYOLO(v3)** | |
|---|---|---|---|
| | **Filter** | **Filter with FC** | **Filter with FC+BS** |
| Conv1 | 3 × 3 (16) | 3 × 3 (16) | 3 × 3 (16) |
| Conv2 | 3 × 3 (32) | 3 × 3 (31) | 1 × 1 (8)<br>3 × 3 (8)<br>1 × 1 (32) |
| Conv3 | 3 × 3 (64) | 3 × 3 (60) | 1 × 1 (15)<br>3 × 3 (15)<br>1 × 1 (60) |
| Conv4 | 3 × 3 (128) | 3 × 3 (115) | 1 × 1 (29)<br>3 × 3 (29)<br>1 × 1 (116) |
| Conv5 | 3 × 3 (256) | 3 × 3 (204) | 1 × 1 (51)<br>3 × 3 (51)<br>1 × 1 (204) |
| Conv6 | 3 × 3 (512) | 3 × 3 (261) | 1 × 1 (66)<br>3 × 3 (66)<br>1 × 1 (264) |
| Conv7 | 3 × 3 (1024) | 3 × 3 (335) | 1 × 1 (84)<br>3 × 3 (84)<br>1 × 1 (336) |
| Conv8 | 1 × 1 (256) | 1 × 1 (256) | 1 × 1 (256) |
| Conv9 | 3 × 3 (512) | 3 × 3 (234) | 1 × 1 (59)<br>3 × 3 (59)<br>1 × 1 (236) |
| Conv10 | 1 × 1 (128) | 1 × 1 (128) | 1 × 1 (128) |
| Conv11 | 3 × 3 (256) | 3 × 3 (148) | 1 × 1 (37)<br>3 × 3 (37)<br>1 × 1 (148) |

EmbeddedPigYOLO (with FC+BS) shown in Tables 2 and 3 represents the result of the BS module from EmbeddedPigYOLO (with FC). For Conv1 of TinyYOLO, the actual execution time of applying the bottleneck structure was longer than that of Conv1 and thus, we did not apply the bottleneck structure to Conv1. With this EmbeddedPigYOLO (i.e., light-weight version of TinyYOLO) and the three-channel composite image (through the IP module), the proposed EmbeddedPigDet can improve both execution speed and accuracy of TinyYOLO simultaneously.

## 4. Experimental Results

### *4.1. Experimental Setup and Resources for the Experiment*

For the purpose of comparison, our individual pig detection experiments were conducted in the following PC as well as low-cost NVIDIA Jetson (NVIDIA, Santa Clara, CA, USA) environments: Ubuntu 16.04.2 LTS (Canonical Ltd., London, UK), OpenCV 4.1 for image processing [76], and

- PC: Intel Core i5-9400F 2.90 GHz (Intel, Santa Clara, CA, USA), NVIDIA GeForce RTX2080 Ti (NVIDIA, Santa Clara, CA, USA), 32 GB RAM.
- Jetson TX-2 [77]: dual-core Denver 2 64-bit CPU and quad-core ARM A57 complex, NVIDIA Pascal™ architecture with 256 NVIDIA CUDA cores, 8 GB 128-bit LPDDR4.
- Jetson Nano [50]: quad-core ARM A57 complex, NVIDIA Maxwell™ architecture with 128 NVIDIA CUDA cores, 4 GB 64-bit LPDDR4.

We conducted the experiment in a 3.2 m tall and 2.0 m wide × 4.9 m long pigsty at Chungbuk National University and installed a low-cost Intel RealSense camera (D435 model, Intel, Santa Clara, CA, USA) [56] on the ceiling to obtain the images. A total of nine pigs (Duroc × Landrace × Yorkshire) were raised in a pig room and the average initial body weight of each pig was (92.5 ± 5.9) kg. We acquired color, infrared and depth images through the low-cost camera installed on the ceiling and each image had a resolution of 1280 × 720, at 30 frames per second (fps). Figure 5 shows a pig room with a camera installed on the ceiling. To exclude the unnecessary region of the pig room, we set Region of Interest (RoI) as 608 × 288.

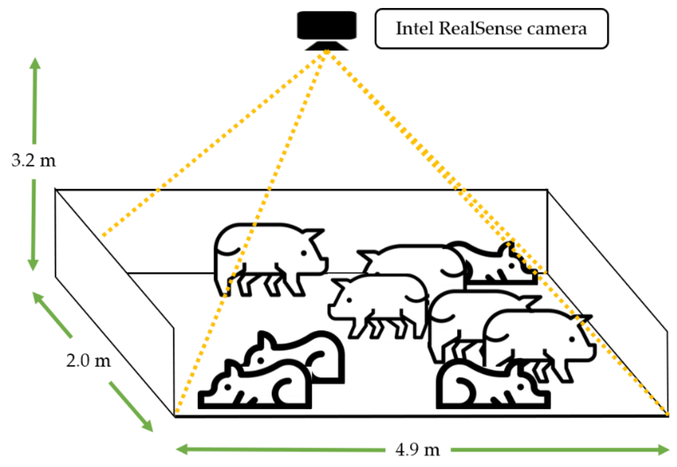


**Figure 5.** Experimental setup with Intel RealSense low-cost camera.

From the camera, we acquired 2904 training images and then trained EmbeddedPigYOLO (0.0001 for learning rate, 0.0005 for decay, 0.9 for momentum, leaky ReLU as the activation function, default anchor parameter and 20,000 for the iterations). Then, we obtained 1000 test images and conducted the test with light-weight image preprocessing and deep learning modules. The reported accuracy was the average of five-fold cross validation. Also, we implemented the proposed methods with YOLOv2 [74] and YOLOv3 [75], respectively (i.e., EmbeddedPigYOLO(v2) and EmbeddedPigDet(v2), EmbeddedPigYOLO(v3) and EmbeddedPigDet(v3)). With COCO data set [78], YOLOv3 could improve meaningfully the accuracy of YOLOv2 with additional computational workload [79]. In pig detection, however, the accuracy of YOLOv3 was similar to that of YOLOv2 but YOLOv3 was much (i.e., by a factor of 2) slower than YOLOv2. In the following, therefore, we reported the performance of YOLOv2 related methods only.

## *4.2. Evaluation of Detection Performance*

The main steps of the proposed method are to create a composite image from infrared input images using the image preprocessing (IP) module and then run EmbeddedPigYOLO with the composite image. Figure 6 shows the results of detecting pigs through EmbeddedPigDet (i.e., IP + EmbeddedPigYOLO). To evaluate the effect of the IP module of EmbeddedPigDet qualitatively, the results of TinyYOLO with the infrared images were also shown in Figure 6. The 2 AM and 8 PM images were night-time images and there were many pigs lying on the floor. Therefore, in these images, pixel values of pigs and background had similar values. On the other hand, the 8 AM and 2 PM images were daytime images. There was sunlight in the room and pigs moved relatively a lot in the daytime images. As we confirm in Figure 6, EmbeddedPigDet performed well in difficult detection situations, such as similar pixel values with background and sunlight.

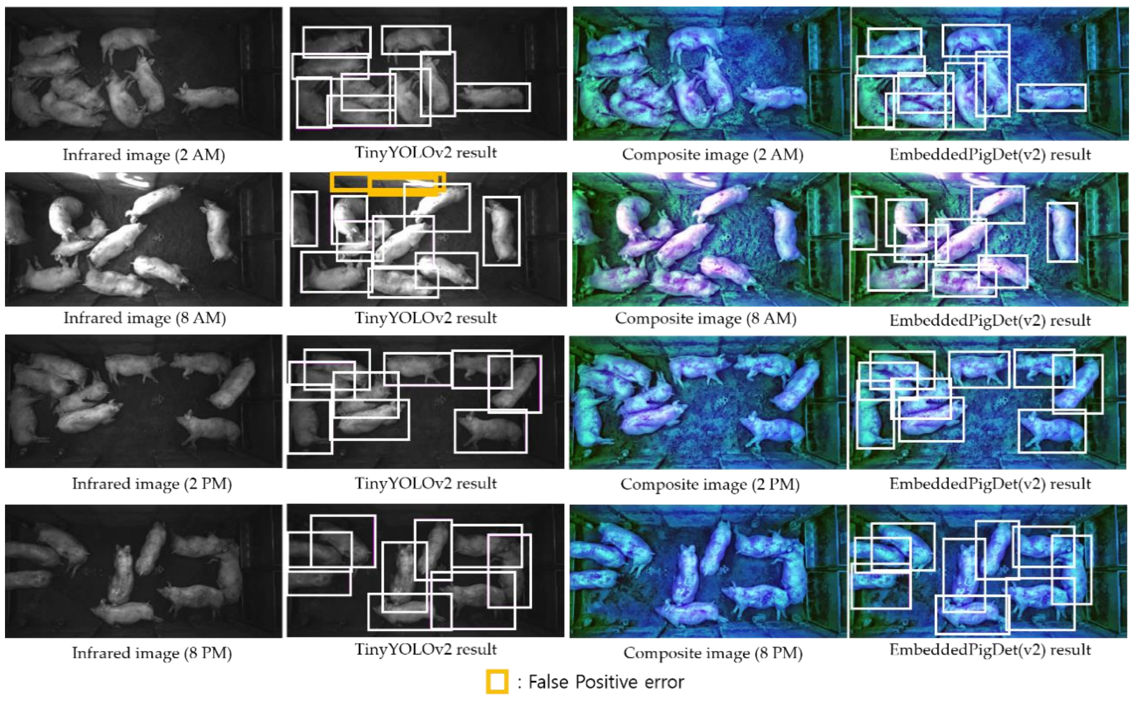


**Figure 6.** Detection results of TinyYOLOv2 with infrared images and EmbeddedPigDet(v2) with composite images.

In fact, this detection accuracy of EmbeddedPigDet was largely due to the IP module. In order to evaluate the effect of the IP module quantitatively, we compare the quality of the infrared input images and the three-channel composite images by using the following metrics:

$$mean = \frac{1}{MN}\sum_{i=0}^{N-1}\sum_{j=0}^{M-1} I_{ij} \tag{1}$$

where $I_{ij}$ is the image value at $(i, j)$ of $N \times M$ image.

$$contrast = \sqrt{\frac{1}{MN}\sum_{i=0}^{N-1}\sum_{j=0}^{M-1}(I_{ij} - mean)^2} \tag{2}$$

$$entropy = -\sum_{k=0}^{255} P_k log_2 P_k \tag{3}$$

where $P_k$ is the proportion of the pixel value $k$.

**Table 4.** Comparison of image quality between the input and composite images.

| | Mean | | Contrast | | Entropy | |
|---|---|---|---|---|---|---|
| | **Input** | **Composite** | **Input** | **Composite** | **Input** | **Composite** |
| 2 AM | 52.76 | 92.43 | 0.27 | 0.41 | 4.44 | 5.23 |
| 8 AM | 74.94 | 103.82 | 0.28 | 0.40 | 4.53 | 5.30 |
| 2 PM | 54.25 | 95.12 | 0.22 | 0.40 | 4.40 | 5.25 |
| 8 PM | 41.19 | 87.75 | 0.18 | 0.38 | 4.06 | 5.16 |

Table 4 compares the values of mean, contrast and entropy of the input image with those of the composite image. At 8 AM, note that, there was strong sunlight in the input image through a window in the pig room and thus the values of mean, contrast and entropy of the input image at that time period were relatively larger than those at other time periods. Based on the difference between mean values of the input and composite images, we could confirm that, generally, the darker input image became brighter in the composite image after the IP module (see Figure 6). Also, larger values of contrast and entropy generally indicate better image quality. Because the values of contrast and entropy were increased after the IP module, better image quality of the composite image could help to improve the detection accuracy.

## *4.3. Comparison of Detection Performance*

In order to evaluate the execution speed and the accuracy of the proposed method, we compared it with YOLOv2 [74] and TinyYOLOv2 [74]. YOLO is one of the most widely used "single-shot" object detectors, because it is faster than "multi-shot" object detectors. As explained, TinyYOLO is a tiny version of YOLO and thus YOLO can detect individual pigs more accurately than TinyYOLO but more slowly. Note that most previous studies of "end-to-end" object detectors used color images. As explained in Section 2, however, we could not get the color input image at night because of the turned-off light at night. Therefore, we reported only the accuracies of those object detectors with infrared input images in order to compare the proposed method at daytime and nighttime.

Figure 7 shows the failure cases of pig detection under two different illumination conditions (i.e., night-time and daytime images) by YOLOv2, TinyYOLOv2 and EmbeddedPigDet(v2). Since each method could detect most of the individual pigs, we show only the false positive (i.e., false pigs) and the false negative (i.e., missed pigs) cases. Regardless of the detection methods, detecting heavily occluded pigs were difficult. Motion information used for video object detection applications may need to be considered for detecting heavily occluded pigs more accurately. Furthermore, false detections occurred on the daytime image, which was caused by sunlight. For handling such pig-like sunlight, we may need more advanced training techniques to reduce the false positive errors. We will consider these issues to correct infrequent errors as our future work.

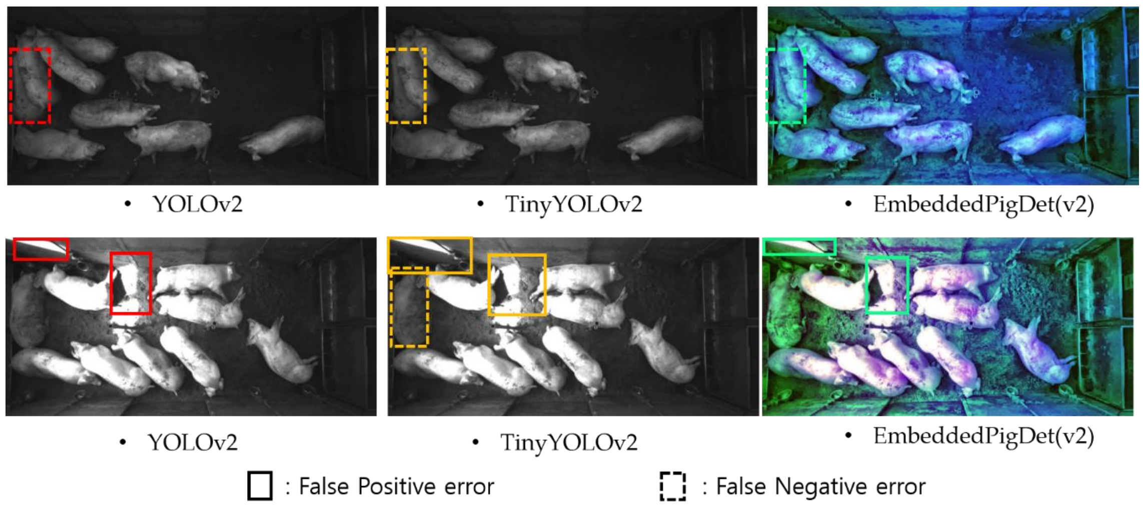


**Figure 7.** Failure cases for YOLOv2 [74], TinyYOLOv2 [74] and EmbeddedPigDet(v2).

To compare the quantitative accuracy of the proposed method with YOLO and TinyYOLO, we used Average Precision (denoted as AP), computed as the area under the precision-recall curve. Note that the precision was computed as the ratio of actual pigs to detected pigs as true by each model, while the recall was computed as the ratio of detected pigs as true by each model to actual pigs. In fact, mean AP (denoted as mAP), computed as the mean of AP for each class, is a detection metric widely used in object detection challenges, such as PASCAL VOC [79]. However, instead of mAP for multi-classes detection, we used AP for single-class (i.e., pig) detection. Based on Ref. [79], we considered the overlap (between bounding boxes of GT and each method) with an Intersection over Union larger than 0.5 as true detection. Tables 5 and 6 summarize the accuracy (i.e., AP) of YOLO, TinyYOLO and the proposed method. To evaluate the effect of the IP module, we also compare the accuracy of EmbeddedPigDet with that of EmbeddedPigYOLO. The accuracy of TinyYOLO was worse than that of YOLO and the accuracy of EmbeddedPigYOLO was worse than that of TinyYOLO. By using the IP module, however, the accuracy of EmbeddedPigDet could be improved and even better than that of TinyYOLO.

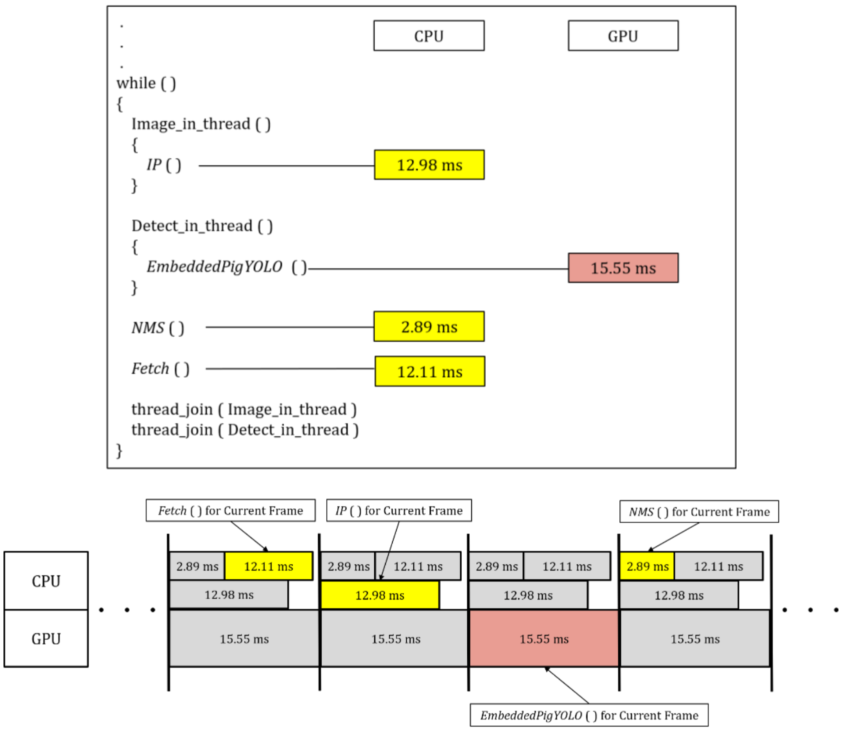
**Table 5.** Comparison of average performance on a TX-2.

| **Method** | | **Accuracy (AP) ↑** | **Speed (fps) ↑** | **Integrated Performance (AP × fps) ↑** |
|---|---|---|---|---|
| YOLOv2 [75] | End-to-end deep learning for object detection | 98.41 | 6.86 | 675 |
| TinyYOLOv2 [75] | | 96.96 | 24.94 | 2418 |
| EmbeddedPigYOLO(v2) (Ours) | | 95.60 | 56.91 | 5440 |
| EmbeddedPigDet(v2) (Ours) | Combination of image processing and deep learning | 97.66 | 32.73 | 3196 |
| **EmbeddedPigDet$_{\mathbf{pipe}}$(v2)** (Ours) | | **97.66** | **64.30** | **6279** (×9.3 than YOLO) (×2.5 than TinyYOLO) |

**Table 6.** Comparison of average performance on a Nano.

| Method | | Accuracy (AP) ↑ | Speed (fps) ↑ | Integrated Performance (AP × fps) ↑ |
|---|---|---|---|---|
| YOLOv2 [75] | End-to-end deep learning for object detection | 98.41 | 3.91 | 384 |
| TinyYOLOv2 [75] | | 96.96 | 12.78 | 1239 |
| EmbeddedPigYOLO(v2) (Ours) | | 95.60 | 21.19 | 2025 |
| EmbeddedPigDet(v2) (Ours) | Combination of image processing and deep learning | 97.66 | 15.83 | 1545 |
| **EmbeddedPigDet$_{\mathbf{pipe}}$(v2)** (Ours) | | **97.66** | **34.38** | **3357** (×8.7 than YOLO) (×2.7 than TinyYOLO) |

Because the execution speed requirement is another important factor in continuous monitoring applications, the processing throughput of each method was measured as frames per second (fps). Like YOLO and TinyYOLO, EmbeddedPigYOLO is an end-to-end deep network. However, EmbeddedPigDet requires additional overhead of the IP module and thus the CPU time for the IP module should be included in computing its execution speed. To reduce the effect of additional CPU time for the IP module of EmbeddedPigDet, we implemented the pipelined version of EmbeddedPigDet (denoted as EmbeddedPigDet$_{\mathbf{pipe}}$). With a pipelined execution, the additional CPU time for the IP module (e.g., 12.98 ms on a TX-2) can be hidden by the GPU time for EmbeddedPigYOLO (e.g., 15.55 ms on a TX-2) in processing the continuous video stream. In Figure 8, for the purpose of explanation between the CPU and GPU computation of EmbeddedPigDet$_{\mathbf{pipe}}$, we separately represented the image fetch step by CPU (denoted as *Fetch*), the image preprocessing module by CPU (denoted as *IP*), EmbeddedPigYOLO by GPU (denoted as *EmbeddedPigYOLO*) and the postprocessing step for Non-Maximum Suppression by CPU (denoted as *NMS*).


```
.
.
.
while ( )
{
   Image_in_thread ( )
   {
      IP ( )
   }

   Detect_in_thread ( )
   {
      EmbeddedPigYOLO ( )
   }

   NMS ( )

   Fetch ( )

   thread_join ( Image_in_thread )
   thread_join ( Detect_in_thread )
}
```

**Figure 8.** Illustration of the code structure for a pipelined execution and execution times of each step on a TX-2.

As shown in Table 6, YOLO and TinyYOLO could not satisfy the execution speed requirement on a Nano embedded board, although YOLO is a single-shot detector and TinyYOLO is a tiny version of YOLO. On the other hand, EmbeddedPigYOLO could improve the execution speed of TinyYOLO significantly. Although we minimized the additional overhead of the IP module with simple image processing techniques, the execution speed of EmbeddedPigDet was degraded. However, the CPU time for the IP module was less than the GPU time for EmbeddedPigYOLO. Therefore, with a pipelined execution, the additional CPU time for the IP module could be totally hidden by the GPU time and EmbeddedPigDet$_{\mathbf{pipe}}$ could recover its execution speed.

In general, there is a tradeoff between execution speed and accuracy. In order to represent this tradeoff with a single performance metric, we define the "integrated" performance as a product of execution speed and accuracy. Compared to the end-to-end deep learning-based methods (i.e., YOLO and TinyYOLO), the proposed method EmbeddedPigDet$_{\mathbf{pipe}}$ could improve the integrated performance by a factor of up to 9.3 and 2.7, respectively. As explained, the first goal of this study was to improve the execution speed of a well-known tiny object detector (i.e., TinyYOLO) for low-cost embedded board implementations. By generating the composite image and applying the pipelining technique, however, we could improve the integrated performance of both YOLO and TinyYOLO, regardless of the platform used. Since the proposed method could be applied to any 3 × 3 convolutional layer, the proposed method can also be applied to other tiny versions of CNN-based object detectors having 3 × 3 convolutional layers. Finally, we compared the cost effectiveness of each method by computing "per-cost" integrated performance. Compared to the end-to-end deep learning-based methods (i.e., YOLO and TinyYOLO), the proposed method could improve the per-cost integrated performance (see Table 7). For example, the proposed method could improve the per-cost integrated performance of YOLO and TinyYOLO by a factor of 1.6 and 1.1 on a typical PC, respectively. On a Nano board, however, the proposed method could improve the per-cost integrated performance of YOLO and TinyYOLO by a factor of 8.7 and 2.6, respectively. Across the platforms, furthermore, the proposed method on a Nano could provide better per-cost integrated performance than that of a PC by a factor of 2.4. TinyYOLOv2 could also provide slightly better per-cost integrated performance on a Nano than on a PC, whereas YOLOv2 could provide better per-cost integrated performance on a PC than on a Nano. That is, the lighter the method, the higher the per-cost integrated performance. Even with low-cost embedded boards, the accuracy of the proposed method was not degraded and thus the proposed method can be a practical solution for large-scale pig farms.

**Table 7.** Comparison of per-cost integrated performance on PC and Nano board platforms.

| **Method** | **Per-Cost Integrated Performance (AP × fps÷cost) ↑** | |
|---|---|---|
| | **PC ($2000)** | **Nano ($100)** |
| YOLOv2 [75] | 8.47 | 3.84 |
| TinyYOLOv2 [75] | 11.94 | 12.39 |
| **EmbeddedPigDet$_{\mathbf{pipe}}$ (v2)** (Ours) | **13.79** | **33.57** (×2.4 than PC) |

In fact, this analysis of cost effectiveness is closely related with the "on-device" AI issue (i.e., processing deep networks directly on embedded devices instead of cloud server platforms) [51–55]. For continuous monitoring of individual pigs with a cloud server, we should transmit the video stream of each pig room into the cloud server. However, the cost of a transmitter is not lower than the cost of a Nano board. Once we transmit the video stream, then we should consider the additional cost to detect individual pigs on the cloud server. As shown in Table 7, the higher the platform cost, the lower the per-cost integrated performance with the proposed method. This situation is very similar to *automated driving* and thus the on-device AI community is developing light-weight versions of deep networks for low-cost embedded boards. To the best of our knowledge, the main idea of this study (i.e., applying *filter clustering* to 3 × 3 convolutional layers in order to obtain a light-weight version of deep learning-based object detector, then applying *image preprocessing* for generating a three-channel composite image in order to improve its accuracy) was not reported by the on-device AI community. We believe the proposed idea can be one of the possible solutions for developing light-weight versions of deep networks for low-cost embedded boards. Furthermore, the proposed method can monitor individual pigs in a pig room with $200 total cost (including a RealSense camera and a Nano board). Since any owner of a large-scale farm does not want to pay a large monitoring cost, the proposed method can be one of the possible "practical" solutions for developing deep learning-based smart farm applications.

### *4.4. Discussion*

The necessity for the pig monitoring is present due to the difficulty in farm management as identification of exact number of pigs, which have high mortality rate, is impossible for the short-staffed farms. Our proposed approach expands the Infrared channel into three channels through IP process and expects the accuracy enhancement. Therefore, the main focus is using fast deep learning one-stage detector YOLO for the detection, furthermore, lightweight deep network and parallel processing technique has been applied to satisfy real-time processing in embedded-board. In general, it is challenge issue to improve both accuracy and speed, because there are tradeoff between them. To solve the problem, some studies can be considered.

The research can be approached with different machine learning methods which led us to examine various methods that can be incorporated into the research. The methods can be broadly divided into Dimensionality Reduction and Texture, Video and 3D, Other confidence methods.

### *4.5. Dimensionality Reduction and Texture*

Among the existing studies, there were methods (i.e., PCA, LLE) for reducing the input dimension and effectively performing feature extraction to solve the pig monitoring problem. In the case of Reference [80], the studies proposed a two-stage method combining PCA and SVM to pig detection problem and that method shows a performance of 2 fps on PC. However, this method has a limitation that it takes a long time to operate on the embedded board. [81] suggests that the performance of the classification problem is improved by applying PCA dimension reduction. Therefore, we will consider a quick detection method that combines our proposed model with PCA as an interesting future research topic. [82,83] was present as a method for detecting pigs using texture and we will consider texture fusion to 3-channel composite image or audio.

### *4.6. Video and 3D*

Previous studies that proposed detection using video stream include [84–87] and we will carry out future studies to improve detection accuracy by improving the accuracy of detection by simultaneously detecting and tracking using video stream or by fusing detector and LSTM. In addition, as show in Reference [40], we can consider the study of detecting the estrus through the detection of posture change of sow by including pose and action information through 3D Video. In case of the detection of aggressive behavior of pigs, Consideration of applying LSTM by adding motion information like [64] is at hand.

### *4.7. Other confidence methods*

In the future, we consider the subject of attack behavior and estrus detection using multimodal method that utilizes both voice information and image information by referring to References [88–91]. We will also compare and review the technology that can improve data by using the generation model in the proposed method by referring to References [92,93]. Restrictive Boltzmann Machine (RBM) is known to be an unsupervised learning and to be able to effectively perform machine learning. We will also conduct research on efficient preprocessing by introducing RBM method to our future research by referring to Reference [94], a research that borrowed RBM method.

## 5. Conclusions

The automatic detection of individual pigs in a surveillance camera environment is an important issue for the efficient management of pig farms. Especially for large-scale pig farms, practical issues, such as monitoring cost, should be considered. However, satisfying both execution speed and accuracy requirements with a low-cost embedded board is very challenging. For example, a deep learning-based object detector (i.e., YOLO) may not satisfy the execution speed requirement, whereas a tiny version of it (i.e., TinyYOLO) may not satisfy the accuracy requirement.

In this study, we focused on detecting individual pigs with a low-cost embedded board to analyze individual pigs cost effectively, with the ultimate goal of 24 h monitoring in a large-scale pig farm. The main idea of this study was first to apply the filter clustering to 3 × 3 convolutional layers and group into a cluster having the maximum convolution value in order to get EmbeddedPigYOLO (i.e., light-weight version of TinyYOLO). Then, in order to recover its accuracy, we generated a three-channel composite image as an input image for EmbeddedPigYOLO. The composite image was generated for focusing on the possible pig regions in a pig room by maximizing the inter-class variation through CLAHE1 while by minimizing the intra-class variation through CLAHE2. That is, the three-channel composite image could give the complementary information to EmbeddedPigYOLO and let it focus on individual pigs.

Based on the experimental result with more than 1000 test images, we confirmed that the proposed method can detect individual pigs more accurately than TinyYOLO and faster than YOLO and TinyYOLO, regardless of the platform. In terms of the integrated performance representing both execution speed and accuracy simultaneously, the proposed method can improve the integrated performance of both YOLO and TinyYOLO, by a factor of up to 9.3 and 2.7, respectively. If we consider the platform cost, the proposed method on a Nano board can improve the per-cost integrated performance of it on a typical PC by a factor of 2.4. Although we implemented the proposed method with TinyYOLO, the proposed method can also be applied to other tiny versions of object detectors having 3 × 3 convolutional layers.

We believe that the proposed method for low-cost embedded boards can be applied to large-scale pig farms in a cost-effective manner. Furthermore, by expanding this study, we will develop a tracking module to achieve our final goal, which is 24 h individual pig monitoring working on a low-cost embedded board. Once we obtain the tracking module for 24 h individual pig monitoring, we can extend the solution for many high-level vision-based analyses such as aggressive behavior analysis, in order to reduce the damage of a pig farm by using a single embedded board.

**Author Contributions:** Y.C. and D.P. conceptualized and designed the experiments; J.S., H.A., D.K. and S.L. designed and implemented the detection system; Y.C. and D.P. validated the proposed method; J.S., S.L. and Y.C. wrote the paper. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was supported by the Basic Science Research Program through the NRF funded by the MOE (2018R1D1A1A09081924).

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Banhazi, T.; Lehr, H.; Black, J.; Crabtree, H.; Schofield, P.; Tscharke, M.; Berckmans, D. Precision Livestock Farming: An International Review of Scientific and Commercial Aspects. *Int. J. Agric. Biol.* **2012**, *5*, 1–9.
2. Neethirajan, S. Recent Advances in Wearable Sensors for Animal Health Management. *Sens. Bio-Sens. Res.* **2017**, *12*, 15–29. [CrossRef]
3. Tullo, E.; Fontana, I.; Guarino, M. Precision livestock farming: An overview of image and sound labelling. In Proceedings of the 6th European Conference on Precision Livestock Farming, Leuven, Belgium, 10–12 September 2013; pp. 30–38.

4. Matthews, S.; Miller, A.; Clapp, J.; Plötz, T.; Kyriazakis, I. Early Detection of Health and Welfare Compromises through Automated Detection of Behavioural Changes in Pigs. *Vet. J.* **2016**, *217*, 43–51. [CrossRef] [PubMed]
5. Tscharke, M.; Banhazi, T. A Brief Review of the Application of Machine Vision in Livestock Behaviour Analysis. *J. Agric. Inform.* **2016**, *7*, 23–42.
6. Korean Government. *4th Industrial Revolution and Agriculture*; Korean Government: Seoul, Korea, 2016. (In Korean)
7. Han, S.; Zhang, J.; Zhu, M.; Wu, J.; Kong, F. Review of automatic detection of pig behaviours by using Image Analysis. In Proceedings of the International Conference on AEECE, Chengdu, China, 26–28 May 2017; pp. 1–6.
8. Schofield, C. Evaluation of Image Analysis as A Means of Estimating the Weight of Pigs. *J. Agric. Eng. Res.* **1990**, *47*, 287–296. [CrossRef]
9. Wouters, P.; Geers, R.; Parduyns, G.; Goossens, K.; Truyen, B.; Goedseels, V.; Van der Stuyft, E. Image-Analysis Parameters as Inputs for Automatic Environmental Temperature Control in Piglet Houses. *Comput. Electron. Agric.* **1990**, *5*, 233–246. [CrossRef]
10. McFarlane, N.; Schofield, C. Segmentation and Tracking of Piglets in Images. *Mach. Vis. Appl.* **1995**, *8*, 187–193. [CrossRef]
11. Cook, N.; Bench, C.; Liu, T.; Chabot, B.; Schaefer, A. The Automated Analysis of Clustering Behaviour of Piglets from Thermal Images in response to Immune Challenge by Vaccination. *Animal* **2018**, *12*, 122–133. [CrossRef]
12. Brunger, J.; Traulsen, I.; Koch, R. Model-based Detection of Pigs in Images under Sub-Optimal Conditions. *Comput. Electron. Agric.* **2018**, *152*, 59–63. [CrossRef]
13. Tu, G.; Karstoft, H.; Pedersen, L.; Jorgensen, E. Illumination and Reflectance Estimation with its Application in Foreground. *Sensors* **2015**, *15*, 12407–12426. [CrossRef]
14. Tu, G.; Karstoft, H.; Pedersen, L.; Jorgensen, E. Segmentation of Sows in Farrowing Pens. *IET Image Process.* **2014**, *8*, 56–68. [CrossRef]
15. Tu, G.; Karstoft, H.; Pedersen, L.; Jorgensen, E. Foreground Detection using Loopy Belief Propagation. *Biosyst. Eng.* **2013**, *116*, 88–96. [CrossRef]
16. Nilsson, M.; Herlin, A.; Ardo, H.; Guzhva, O.; Astrom, K.; Bergsten, C. Development of Automatic Surveillance of Animal Behaviour and Welfare using Image Analysis and Machine Learned Segmentation Techniques. *Animal* **2015**, *9*, 1859–1865. [CrossRef]
17. Kashiha, M.; Bahr, C.; Ott, S.; Moons, C.; Niewold, T.; Tuyttens, F.; Berckmans, D. Automatic Monitoring of Pig Locomotion using Image Analysis. *Livest. Sci.* **2014**, *159*, 141–148. [CrossRef]
18. Oczak, M.; Maschat, K.; Berckmans, D.; Vranken, E.; Baumgartner, J. Automatic Estimation of Number of Piglets in a Pen during Farrowing, using Image Analysis. *Biosyst. Eng.* **2016**, *151*, 81–89. [CrossRef]
19. Ahrendt, P.; Gregersen, T.; Karstoft, H. Development of a Real-Time Computer Vision System for Tracking Loose-Housed Pigs. *Comput. Electron. Agric.* **2011**, *76*, 169–174. [CrossRef]
20. Khoramshahi, E.; Hietaoja, J.; Valros, A.; Yun, J.; Pastell, M. Real-Time Recognition of Sows in Video: A Supervised Approach. *Inf. Process. Agric.* **2014**, *1*, 73–82. [CrossRef]
21. Nasirahmadi, A.; Hensel, O.; Edwards, S.; Sturm, B. Automatic Detection of Mounting Behaviours among Pigs using Image Analysis. *Comput. Electron. Agric.* **2016**, *124*, 295–302. [CrossRef]
22. Nasirahmadi, A.; Hensel, O.; Edwards, S.; Sturm, B. A New Approach for Categorizing Pig Lying Behaviour based on a Delaunay Triangulation Method. *Animal* **2017**, *11*, 131–139. [CrossRef]
23. Nasirahmadi, A.; Edwards, S.; Matheson, S.; Sturm, B. Using Automated Image Analysis in Pig Behavioural Research: Assessment of the Influence of Enrichment Subtrate Provision on Lying Behaviour. *Appl. Anim. Behav. Sci.* **2017**, *196*, 30–35. [CrossRef]
24. Guo, Y.; Zhu, W.; Jiao, P.; Chen, J. Foreground Detection of Group-Housed Pigs based on the Combination of Mixture of Gaussians using Prediction Mechanism and Threshold Segmentation. *Biosyst. Eng.* **2014**, *125*, 98–104. [CrossRef]
25. Guo, Y.; Zhu, W.; Jiao, P.; Ma, C.; Yang, J. Multi-Object Extraction from Topview Group-Housed Pig Images based on Adaptive Partitioning and Multilevel Thresholding Segmentation. *Biosyst. Eng.* **2015**, *135*, 54–60. [CrossRef]

26. Buayai, P.; Kantanukul, T.; Leung, C.; Saikaew, K. Boundary Detection of Pigs in Pens based on Adaptive Thresholding using an Integral Image and Adaptive Partitioning. *CMU J. Nat. Sci.* **2017**, *16*, 145–155. [CrossRef]
27. Lu, M.; Xiong, Y.; Li, K.; Liu, L.; Yan, L.; Ding, Y.; Lin, X.; Yang, X.; Shen, M. An Automatic Splitting Method for the Adhesive Piglets Gray Scale Image based on the Ellipse Shape Feature. *Comput. Electron. Agric.* **2016**, *120*, 53–62. [CrossRef]
28. Lu, M.; He, J.; Chen, C.; Okinda, C.; Shen, M.; Liu, L.; Yao, W.; Norton, T.; Berckmans, D. An Automatic Ear Base Temperature Extraction Method for Top View Piglet Thermal Image. *Comput. Electron. Agric.* **2018**, *155*, 339–347. [CrossRef]
29. Jun, K.; Kim, S.; Ji, H. Estimating Pig Weights from Images without Constraint on Posture and Illumination. *Comput. Electron. Agric.* **2018**, *153*, 169–176. [CrossRef]
30. Kang, F.; Wang, C.; Li, J.; Zong, Z. A Multiobjective Piglet Image Segmentation Method based on an Improved Noninteractive GrabCut Algorithm. *Adv. Multimed.* **2018**, 108876. [CrossRef]
31. Yang, A.; Huang, H.; Zhu, X.; Yang, X.; Chen, P.; Li, S.; Xue, Y. Automatic Recognition of Sow Nursing Behavious using Deep Learning-based Segmentation and Spatial and Temporal Features. *Biosyst. Eng.* **2018**, *175*, 133–145. [CrossRef]
32. Yang, Q.; Xiao, D.; Lin, S. Feeding Behavior Recognition for Group-Housed Pigs with the Faster R-CNN. *Comput. Electron. Agric.* **2018**, *155*, 453–460. [CrossRef]
33. Kongsro, J. Estimation of Pig Weight using a Microsoft Kinect Prototype Imaging System. *Comput. Electron. Agric.* **2014**, *109*, 32–35. [CrossRef]
34. Lao, F.; Brown-Brandl, T.; Stinn, J.; Liu, K.; Teng, G.; Xin, H. Automatic Recognition of Lactating Sow Behaviors through Depth Image Processing. *Comput. Electron. Agric.* **2016**, *125*, 56–62. [CrossRef]
35. Stavrakakis, S.; Li, W.; Guy, J.; Morgan, G.; Ushaw, G.; Johnson, G.; Edwards, S. Validity of the Microsoft Kinect Sensor for Assessment of Normal Walking Patterns in Pigs. *Comput. Electron. Agric.* **2015**, *117*, 1–7. [CrossRef]
36. Zhu, Q.; Ren, J.; Barclay, D.; McCormack, S.; Thomson, W. Automatic animal detection from kinect sensed images for livestock monitoring and assessment. In Proceedings of the International Conference on Computer and Information Technology, ICCCIT, Dhaka, Bangladesh, 21–23 December 2015; pp. 1154–1157.
37. Kulikov, V.; Khotskin, N.; Nikitin, S.; Lankin, V.; Kulikov, A.; Trapezov, O. Application of 3D Imaging Sensor for Tracking Minipigs in the Open Field Test. *J. Neurosci. Methods* **2014**, *235*, 219–225. [CrossRef] [PubMed]
38. Shi, C.; Teng, G.; Li, Z. An Approach of Pig Weight Estimation using Binocular Stereo System based on LabVIEW. *Comput. Electron. Agric.* **2016**, *129*, 37–43. [CrossRef]
39. Matthews, S.; Miller, A.; Plötz, T.; Kyriazakis, I. Automated Tracking to Measure Behavioural Changes in Pigs for Health and Welfare Monitoring. *Sci. Rep.* **2017**, *7*, 17582. [CrossRef]
40. Zheng, C.; Zhu, X.; Yang, X.; Wang, L.; Tu, S.; Xue, Y. Automatic Recognition of Lactating Sow Postures from Depth Images by Deep Learning Detector. *Comput. Electron. Agric.* **2018**, *147*, 51–63. [CrossRef]
41. Lee, J.; Jin, L.; Park, D.; Chung, Y. Automatic Recognition of Aggressive Pig Behaviors using Kinect Depth Sensor. *Sensors* **2016**, *16*, 631. [CrossRef]
42. Kim, J.; Chung, Y.; Choi, Y.; Sa, J.; Kim, H.; Chung, Y.; Park, D.; Kim, H. Depth-based Detection of Standing-Pigs in Moving Noise Environments. *Sensors* **2017**, *17*, 2757. [CrossRef]
43. Chung, Y.; Kim, H.; Lee, H.; Park, D.; Jeon, T.; Chang, H. A Cost-Effective Pigsty Monitoring System based on a Video Sensor. *KSII Trans. Internet Inf. Syst.* **2014**, *8*, 1481–1498.
44. Sa, J.; Choi, Y.; Lee, H.; Chung, Y.; Park, D.; Cho, J. Fast Pig Detection with a Topview Camera under Various Illumination Conditions. *Symmetry* **2019**, *11*, 266. [CrossRef]
45. Zhang, L.; Gray, H.; Ye, X.; Collins, L.; Allinson, N. Automatic Individual Pig Detection and Tracking in Pig Farms. *Sensors* **2019**, *19*, 1188. [CrossRef] [PubMed]
46. Nasirahmadi, A.; Sturm, B.; Olsson, A.; Jeppsson, K.; Muller, S.; Edwards, S.; Hensel, O. Automatic Scoring of Lateral and Sternal Lying Posture in Grouped Pigs Using Image Processing and Support Vector Machine. *Comput. Electron. Agric.* **2019**, *156*, 475–481. [CrossRef]
47. Psota, E.; Mittek, M.; Perez, L.; Schmidt, T.; Mote, B. Multi-Pig Part Detection and Association with a Fully-Convolutional Network. *Sensors* **2019**, *19*, 852. [CrossRef] [PubMed]

48. Li, B.; Liu, L.; Shen, M.; Sun, Y.; Lu, M. Group-Housed Pig Detection in Video Surveillance of Overhead Views using Multi-Feature Template Matching. *Biosyst. Eng.* **2019**, *181*, 28–39. [CrossRef]
49. Lee, S.; Ahn, H.; Seo, J.; Chung, Y.; Park, D.; Pan, S. Practical Monitoring of Undergrown Pigs for IoT-Based Large-Scale Smart Farm. *IEEE Access* **2019**, *7*, 173796–173810. [CrossRef]
50. NVIDIA. NVIDIA Jetson Nano. Available online: http://www.nvidia.com/object/embedded-systems-dev-kits-modules.html (accessed on 10 November 2019).
51. Mahdavinejad, M.; Rezvan, M.; Barekatain, M.; Adibi, P.; Barnaghi, P.; Sheth, A. Machine Learning for Internet of Things Data Analysis: A Survey. *Digit. Commun. Netw.* **2018**, *4*, 161–175. [CrossRef]
52. Ham, M.; Moon, J.; Lim, G.; Song, W.; Jung, J.; Ahn, H.; Woo, S.; Cho, Y.; Park, J.; Oh, S.; et al. NNStreamer: Stream Processing Paradigm for Neural Networks, Toward Efficient Development and Execution of On-Device AI Applications. *arXiv* **2019**, arXiv:1901.04985.
53. Nguyen, P.; Arsalan, M.; Koo, J.; Naqvi, R.; Truong, N.; Park, K. LightDenseYOLO: A Fast and Accurate Marker Tracker for Autonomous UAV Landing by Visible Light Camera Sensor on Drone. *Sensors* **2018**, *18*, 1703. [CrossRef]
54. Xiao, J.; Wu, H.; Li, X. Internet of Things Meets Vehicles: Sheltering In-Vehicle Network through Lightweight Machine Learning. *Symmetry* **2019**, *11*, 1388. [CrossRef]
55. Yang, T.; Howard, A.; Chen, B.; Zhang, X.; Go, A.; Sandler, M.; Sze, V.; Adam, H. Netadapt: Platform-aware neural network adaptation for mobile applications. In Proceedings of the European Conference on Computer Vision, Munich, Germany, 8–14 September 2018; pp. 285–300.
56. Intel. Intel RealSense D435. Available online: https://click.intel.com/intelr-realsensetm-depth-camera-d435.html (accessed on 28 February 2018).
57. Bradley, D.; Roth, G. Adaptive Thresholding using the Integral Image. *J. Graph. Tools* **2007**, *12*, 13–21. [CrossRef]
58. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vega, NV, USA, 26 June–1 July 2016; pp. 779–788.
59. Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 24–27 June 2014; pp. 580–587.
60. Girshick, R. Fast R-CNN. In Proceedings of the IEEE International Conference on Computer Vision, Las Condes, Chile, 11–18 December 2015; pp. 1440–1448.
61. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster R-CNN: Towards real-time object detection with region proposal networks. In Proceedings of the Advances in Neural Information Processing Systems, Montréal, QC, Canada, 7–12 December 2015; pp. 91–99.
62. Ott, S.; Moons, C.; Kashiha, M.; Bahr, C.; Tuyttens, F.; Berckmans, D.; Niewold, T. Automated video analysis of pig activity at pen level highly correlates to human observations of behavioural activities. *Livest. Sci.* **2014**, *160*, 132–137. [CrossRef]
63. Chen, C.; Zhu, W.; Steibel, J.; Siegford, J.; Wurtz, K.; Han, J.; Norton, T. Recognition of aggressive episodes of pigs based on convolutional neural network and long short-term memory. *Comput. Electron. Agric.* **2020**, *169*, 105166. [CrossRef]
64. Chen, C.; Zhu, W.; Liu, D.; Steibel, J.; Siegford, J.; Wurtz, K.; Norton, T. Detection of aggressive behaviours in pigs using a RealSence depth sensor. *Comput. Electron. Agric.* **2019**, *166*, 105003. [CrossRef]
65. Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; Graf, H.P. Pruning filters for efficient convnets. *arXiv* **2016**, arXiv:1608.08710.
66. He, Y.; Kang, G.; Dong, X.; Fu, Y.; Yang, Y. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv* **2018**, arXiv:1808.06866.
67. Yu, R.; Li, A.; Chen, C.; Lai, J.; Morariu, V.; Han, X.; Davis, L. Nisp: Pruning networks using neuron importance score propagation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 9194–9203.
68. Ding, X.; Ding, G.; Han, J.; Tang, S. Auto-balanced filter pruning for efficient convolutional neural networks. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018.

69. Lin, S.; Ji, R.; Li, Y.; Wu, Y.; Huang, F.; Zhang, B. Accelerating convolutional networks via global & dynamic filter pruning. In Proceedings of the International Joint Conferences on Artificial Intelligence, Stockholm, Sweden, 13–19 July 2018; pp. 2425–2432.
70. Peng, B.; Tan, W.; Li, Z.; Zhang, S.; Xie, D.; Pu, S. Extreme network compression via filter group approximation. In Proceedings of the European Conference on Computer Vision, Munich, Germany, 8–14 September 2018; pp. 300–316.
71. Zhuang, Z.; Tan, M.; Zhuang, B.; Liu, J.; Guo, Y.; Wu, Q.; Zhu, J. Discrimination-aware channel pruning for deep neural networks. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 3–8 December 2018; pp. 875–886.
72. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
73. Zuiderveld, K. *Contrast Limited Adaptive Histogram Equalization*; Academic Press Inc.: Cambridge, MA, USA, 1994.
74. Redmon, J.; Farhadi, A. YOLO9000: Better, faster, stronger. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 7263–7271.
75. Redmon, J.; Farhadi, A. Yolov3: An incremental improvement. *arXiv* **2018**, arXiv:1804.02767.
76. OpenCV. Open Source Computer Vision. Available online: http://opencv.org (accessed on 30 April 2019).
77. NVIDIA. NVIDIA Jetson TX2. Available online: http://www.nvidia.com/object/embedded-systems-dev-kits-modules.html (accessed on 30 April 2019).
78. Lin, T.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollar, P.; Zitnick, C. Microsoft COCO: Common Objects in Context. In Proceedings of the European Conference on Computer Vision, Zurich, Switzerland, 6–12 September 2014; pp. 740–755.
79. Everingham, M.; Van Gool, L.; Williams, C.; Winn, J.; Zisserman, A. The Pascal Visual Object Classes (VOC) Challenge. *Int. J. Comput. Vis.* **2010**, *88*, 303–338. [CrossRef]
80. Kwak, T.; Song, A.; Kim, Y. The Impact of the PCA Dimensionality Reduction for CNN based Hyperspectral Image Classification. *Korean J. Remote Sens.* **2019**, *35*, 959–971.
81. Sun, L.; Liu, Y.; Chen, S.; Luo, B.; Li, Y.; Liu, C. Pig Detection Algorithm Based on Sliding Windows and PCA Convolution. *IEEE Access* **2019**, *7*, 44229–44238. [CrossRef]
82. Kim, J.; Choi, Y.; Sa, J.; Ju, M.; Chung, Y.; Park, D.; Kim, H. *Pig Detection Using Texture Information*; The Institute of Electronics and Information Engineers: Seoul, Korea, 2016; pp. 403–406. (In Korean)
83. Choi, Y.; Lee, J.; Park, D.; Chung, Y. Noise-Robust Porcine Respiratory Diseases Classification Using Texture Analysis and CNN. *KIPS Trans. Softw. Data Eng.* **2018**, *7*, 91–98.
84. Mousas, C.; Anagnostopoulos, C. Learning motion features for example-based finger motion estimation for virtual characters. *3D Res.* **2017**, *8*, 25. [CrossRef]
85. Yuan, P.; Zhong, Y.; Yuan, Y. Faster r-cnn with region proposal refinement. *Tech. Rep.* **2017**.
86. Han, W.; Khorrami, P.; Paine, T.; Ramachandran, P.; Babaeizadeh, M.; Shi, H.; Huang, T. Seq-nms for video object detection. *arXiv* **2016**, arXiv:1602.08465.
87. Zhou, Y.; Li, Z.; Xiao, S.; He, C.; Huang, Z.; Li, H. Auto-conditioned recurrent networks for extended complex human motion synthesis. *arXiv* **2018**, arXiv:1707.05363.
88. Ngiam, J.; Khosla, A.; Kim, M.; Nam, J.; Lee, H.; Ng, A. Multimodal deep learning. In Proceedings of the International Conference on Machine Learning, Washington, DC, USA, 28 June–2 July 2011.
89. Chung, Y.; Oh, S.; Lee, J.; Park, D.; Chang, H.; Kim, S. Automatic detection and recognition of pig wasting diseases using sound data in audio surveillance systems. *Sensors* **2013**, *13*, 12929–12942. [CrossRef]
90. Han, S.; Lee, S.; Sa, J.; Ju, M.; Kim, H.; Chung, Y.; Park, D. Pigs boundary detection using both color and depth information. *Korean Inst. Smart Media* **2015**, *5*, 168–170.
91. Kim, H. Automatic identification of a coughing animal using audio and video data. In Proceedings of the Fourth International Conference on Information Science and Cloud Computing, Guangzhou, China, 18–19 December 2015; p. 264.
92. Bai, J.; Zhang, H.; Li, Z. The generalized detection method for the dim small targets by faster R-CNN integrated with GAN. In Proceedings of the IEEE 3rd International Conference on Communication and Information Systems (ICCIS), Singapore, 28–30 December 2018; IEEE: Piscataway, NJ, USA, 2018.

93. Choi, Y.; Lee, J.; Park, D.; Chung, Y. Enhanced Sound Signal Based Sound-Event Classification. *Korea Inf. Process. Soc.* **2019**, *8*, 193–204. (In Korean)
94. Sailor, H.; Patil, H. Novel unsupervised auditory filterbank learning using convolutional RBM for speech recognition. *IEEE/ACM Trans. Audio Speech Lang. Process.* **2016**, *24*, 2341–2353. [CrossRef]