IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received December 13, 2019, accepted January 27, 2020, date of publication February 4, 2020, date of current version February 13, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2971566

iSEC: An Optimized Deep Learning Model for Image Classification on Edge Computing

ENDAH KRISTIANI[1,2], CHAO-TUNG YANG[3], (Member, IEEE), AND CHIN-YIN HUANG[1]
[1]Department of Industrial Engineering and Enterprise Information, Tunghai University, Taichung City 40704, Taiwan
[2]Department of Informatics, Faculty of Engineering and Computer Science, Krida Wacana Christian University, Jakarta 11470, Indonesia
[3]Department of Computer Science, Tunghai University, Taichung City 40704, Taiwan

Corresponding author: Chao-Tung Yang (ctyang@thu.edu.tw)

This work was supported by the Ministry of Science and Technology (MOST), Taiwan, under Grant 108-2745-8-029-007, Grant 108-2745-8-029-007, and Grant 108-2622-E-029-007-CC3.

ABSTRACT Optimization strategies in deep learning models require different techniques for different use cases. Besides, various phases of the model deployment life-cycle specify possible and particular optimization strategies. In this paper, an optimized deep learning model on the edge computing environment is proposed for image classification cases. For preparing the dataset, the image preprocessing and data augmentation methods are utilized to prepare the data for the training process. To accelerate the deep learning training process, this system implemented CPU optimization and hyperparameter tuning. Tensorflow is applied as a framework for the training model. InceptionV3, VGG16, and MobileNet are applied as topology implemented in the deep learning training comparison. In this case, InceptionV3 was used for modeling the deep learning applications on edge. To optimize the trained model, a Model Optimizer is used on the edge device. It can be seen in the experiments, MobileNet was the least accurate model (85%) and the longest time to load the model (71s). VGG16 was the most reliable (91%) and the shortest time to load the model (50s). InceptionV3 has median accuracy (87%) and the average time to load the model (52s).

INDEX TERMS Data augmentation, CPU optimization, hyperparameter tuning, deep learning, inference optimization, InceptionV3, VGG16, mobilenet, cloud computing, edge computing.

I. INTRODUCTION

In 2018, there are an estimated 62% of organizations using artificial intelligence (AI) that can create significant benefits for early adopters. A research [31] by McKinsey in 2017 found that after using automation, the healthcare, financial services, and professional services industries posted a 3-15% higher profit margin. This phenomenon is an opportunity for the machine learning developer to provide a better AI modelling [36]. The first step towards the successful implementation of the artificial intelligence modeling system is to design a stable, repeatable, and sustainable development process [32]. These are only strengthened in value for the modeling of deep learning (DL) and machine learning (ML).

Optimization allows developers to transform this step-by-step process into a streamlined process of prototyping. Advanced approaches to automatic adjustment enable developers to optimize search steps for data transformation, design, and software hyperparameter configuration [38], [39]. Therefore, it can significantly impact the entire process of model development. The optimization strategy can be different based on problem identification. It empowers developers to test their modeling efforts rapidly quickly and is designed to customize each system for best performance [33], [34].

Edge computing is inherently a decentralized system with intelligence in independent entities. Along with the emerging of the Internet of Things (IoT), they create opportunities for more decentralized and distributed computing infrastructures. This phenomenon embodied a cut-through of edge computing to complement cloud computing [17], [18]. Edge computing offers essential services to support this critical concept of Industry 4.0 [22], [23]. The outcomes of the edge computing environment can support the need in the case of inter-connectivity, higher reliability, real-time predictive analysis, and low latency [16], [24], [25]. Specifically, the form of Artificial Intelligence (AI) in edge computing is almost unavoidable. The use of edge computing in the industry mostly related to visualization and automation

The associate editor coordinating the review of this manuscript and approving it for publication was Giovanni Pau.

[27], [28]. The need to improve and optimize the algorithms is fundamental [29], [30].

There are three main phases of machine learning, dataset preparation, training, and the inference process. Cleaning, structuring complex dataset, and data wrangling in dataset preparation will allow a quick training process. Also, it enhances the model accuracy in the process of training data. In the training phase, it is essential to optimize the process by selecting a framework network, fine-tuning the model, and do hyperparameter tuning for better model performance. During the inference process, it is essential to optimize the model for ensuring the speed and the accuracy [37].

Object recognition technology is one of the essential applications of Computer Vision. This technology can empower computers the ability to see things like humans, find and locate specific objects within an image [1], [2]. Recently applications such as video surveillance or face recognition have been created in industries. The use of computer vision in the sector has been a good view of the future [3]. It can be inferred that new advances in AI will help to accelerate this trend towards manufacturing. In this case, the essential part of implementing a model of AI is data preparation, training, and inference [4], [5]. Therefore, the main focus of this paper is performance modeling and optimization for dataset preparation, training model, and inference on the edge-cloud computing [21].

In this paper, an optimized model of the edge computing environment for deep learning is proposed. First, for preparing the dataset, the image preprocessing and data augmentation methods are utilized to prepare the data for the training process. Second, a Jupyter notebook server with an IPython kernel is deployed as a machine learning server in the cloud with security support [8]–[10]. Third, on the edge side, a Raspberry Pi with the Intel® Neural Compute Stick 2 (Intel® NCS 2) is developed to provide decentralized service applications. The use case is based on the most stolen vehicle reported in the US in 2017. The use case dataset is generated from Vehicle Make and Model Recognition Dataset (VMMRdb). The images consist of 6877 files of 10 folders/classes. This system implemented CPU optimization and hyperparameter tuning to accelerate the deep learning training model. Tensorflow is used as a framework for the training model. InceptionV3, VGG16, and MobileNet are applied as topology used in the deep learning process. A Model Optimizer (mo) is used on the edge device to optimize the trained model. The specific objectives of this paper are listed as follows:

- Data preprocessing and augmentation to prepare the better dataset for machine learning training.
- Accelerating the deep learning training model with CPU optimization and hyperparameter tuning.
- Performance comparison testing of the chosen networks, InceptionV3, VGG16, and MobileNet.
- Deep learning inference optimization using Model Optimizer on Raspberry 4.

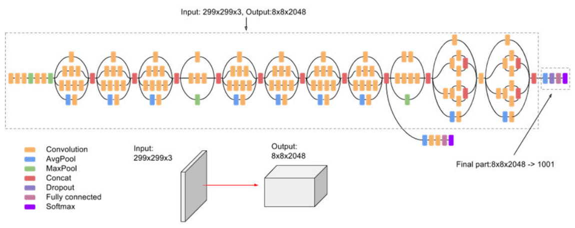


FIGURE 1. The diagram of Inception V3, image source Google [35].

II. BACKGROUND REVIEW AND RELATED WORKS

In this section, several components are used as the approaching methods of this work. The next subsections discuss each element in more detail.

A. TENSORFLOW FRAMEWORK

The TensorFlow framework is based on an opensource which supports more features, also has the supported package for creating the models. TensorFlow with CPU optimizations can give up to 14x Speedup in Training and 3.2x Speedup in Inference. TensorFlow is flexible enough to support experimentation with new deep learning models/topologies and system-level optimizations. Moreover, TensorFlow can be scaled or deployed on different types of devices ranging from CPUs, GPUs, and inference on small devices like mobile phones. TensorFlow has seamless integration with CPU, GPU, and TPU with no need for any configuration.

B. TENSORFLOW TOPOLOGY

There are several things to be considered when selecting the topology or network, that is time to train, size, accuracy, and inference speed. InceptionV3, VGG16, and MobileNet are the three networks which currently supported on the edge devices (CPU, Integrated GPU, Intel® Movidius ™ Neural Compute Stick). In this paper, the InceptionV3 was used as the machine learning model application.

C. INCEPTIONV3

Google's Inception V3 is the third version of the Deep Learning Architectures series [14]. Inception V3 was trained using 1000 classes from the first ImageNet Datasets and trained with over 1 million training images while Tensorflow has 1001 classes that are not used in the original ImageNet as a result of an optional background package. Figure 1 describes the Inception V3 model diagram as shown below:

D. CPU OPTIMIZATION

With all the instructions provided by the target CPU [26], CPUs, like Intel® Xeon processors, will achieve optimal performance when TensorFlow is built from the source. In addition to using the latest instruction sets, the Intel® Math Kernel Library for Deep Neural Networks (Intel® MKL-DNN) has been added to TensorFlow by Intel. These optimizations are often referred to as MKL or TensorFlow with MKL. TensorFlow with Intel MKL-DNN provides information on the optimization of Intel® MKL. By adjusting the thread pools [7], the two configurations listed below are utilized to optimize CPU performance.

- intra-op-parallelism-threads: nodes that can parallel their execution with multiple threads will schedule the individual pieces in this pool.
- inter-op-parallelism-threads: all ready nodes in this pool are planned.

These configurations are set in the config attribute through the tf. ConfigProto and passed to tf.session, as shown in the snippet below. For both configuration options, the number of logical CPU cores will be set by default if they are not set or set to zero. Testing showed that the default is effective for systems ranging from a single4-core CPU to multiple 70+ combined logical core CPUs. Setting the number of threads in both pools equal to the number of physical cores rather than logical cores is a common alternative optimization. To tune performance, Intel MKL uses the following environment variables:

- KMP-BLOCKTIME - Set the time for a thread to wait, in milliseconds, before sleeping after completing the execution of a parallel region.
- KMP-AFFINITY - Threads can be bound to physical processing units by the runtime library.
- KMP-SETTINGS - Enable (true) or disable (false) for the printing during program execution of OpenMP* run-time library environment variables.
- OMP-NUM-THREADS - Specifies the thread number to use.

E. HYPERPARAMETER OPTIMIZATION

The learning rate is a configurable hyperparameter applied in the training of neural networks that has a small positive value between 0.0 and 1.0. Generally, a high learning rate helps the model to learn more quickly at the cost of achieving a sub-optimal final set of weights. A lower learning rate may allow the model to learn a more optimal or even globally optimal weight range, but it may take a considerably longer time to train. If the learning rate is too large, it will result in weight updates that are too high, and the model's performance (such as its loss on the training dataset) will fluctuate over the training epochs. Therefore, it should not use too large or too low a learning rate. However, the model must configure in such a way that on average a good enough set of weights to approximate the mapping problem as the training dataset represents. The performance of the learning rate does not depend on the size of the model. The same standards that performed best for 1x size performed best for 10x size.

F. MODEL ANALYSIS

Next, the text-based version of different main classification metrics is listed as follows.

- Precision: The equation of precision is:

$$P = \frac{TP}{TP + FP} \tag{1}$$

where P is Precision, TP is the number of true positives and FP the number of false positives. The precision is intuitively the ability of the classifier not to label as positive a sample that is negative. The best value is 1, and the worst value is 0.

- Recall: The equation of recall is:

$$R = \frac{TP}{TP + FN} \tag{2}$$

where R is Recall, TP is the number of true positives and FN the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples. The best value is 1, and the worst value is 0.

- F1-Score: Compute the F1 score, also known as balanced F-score or F-measure. The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at one and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:

$$F1 = \frac{2(precisionrecall)}{(precision + recall)} \tag{3}$$

- ROC: ROC is a contrast to a true positive value (Y-axis) of the false positive rate(x-axis) for a range of thresholds varying from 0.0 to 1.0. The true positive rate (TPR) is calculated as the number of true positives (TP), split by the sum of true and false-negative (FN). It summarizes the model's performance when the result is positive in forecasting the positive category [8]. The equation of TPR is described as follows:

$$TPR = \frac{TP}{(TP + FN)} \tag{4}$$

The average F1 score of each class with weighting depending on the average parameter in the multi-class and multi-label case.

- Support: Support is the total number of classes being used for the evaluation.

G. RELATED WORKS

Jia X. et al. [11] suggested optimizing the CNN training method with AlexNet and ResNet-50 algorithms on the ImageNet dataset. They have used 1024 Tesla P40 GPUs, 1024 Tesla P100, and 2048 Tesla P40 GPUs in their experiments. However, they get less than 76 percent of the accuracy, although the learning template time is between 4 and 20 minutes.

Zhang Q. *et al.* [12] proposed an efficient deep learning model based on canonical polyadic decomposition to predict the cloud workload for the information technology industry. By converting the weight matrices to the canonical polyadic format, the parameters are much condensed in the proposed model. Also, they designed an efficient learning algorithm to train the parameters. They show in their results that the proposed model achieves a higher accuracy of training efficiency

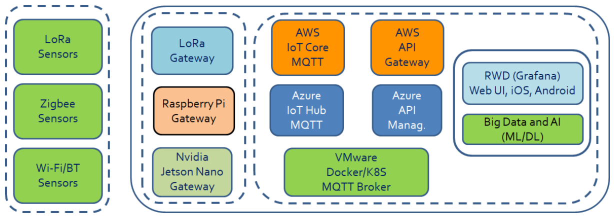


FIGURE 2. Sensors, Edge, and Cloud (iSEC) Framework.

and prediction of the workload than state-of-the-art machine-learning approaches.

A simple training strategy to improve the classification performance of a DNN was presented by Caliskan *et al.* [13]. They used an optimization algorithm for L-BFGS. Compared to the state-of-the-art classifiers, their classification experiments show that the proposed method substantially improves the DNN classifier training process and results in significant improvements in the accuracy of the results of the classification.

Li *et al.* [15] recommended a novel strategy for offloading to optimize the performancez with edge computing of IoT deep learning applications. They test the performance of executing multiple deep learning tasks with their approach in an edge computing environment in the performance evaluation. The results of the assessment show that their method outperforms other IoT deep learning optimization solutions.

Liang *et al.* [14] demonstrated the screens of smart devices tap locations that can be defined based on sensory data. The results of the experiment show that the prediction accuracy of tap location inference by using convolutional neural networks can be at least 90 percent. Also, the user's application patterns and passwords can be inferred with high accuracy based on the information found in the tap location.

III. SISTEM ARCHITECTURE AND IMPLEMENTATION

This system presents system architecture and implementation of the proposed cloud edge computing environment for deep learning.

A. iSEC

This project is part of Sensors, Edge, and Cloud (iSEC) development. Figure 2 describes the entire of the development framework.

B. SYSTEM ARCHITECTURE

The cloud architecture of this system is shown in Fig. 3. In this architecture, Intel Xeon Phi Processor 7210 is used as a hardware system. CentOS 7.4 (64 bit) is utilized as an operating system. Jupyter Notebook is installed with IPython Kernel to improve the performance. TensorFlow is applied as a framework for deep learning.

On the Edge side, the Raspberry Pi is used as the system device. Raspbian OS is applied as operating system. Python, TensorFlow, and OpenCV are installed in Raspberry as a tool for inference. OpenVino package is implemented as a library

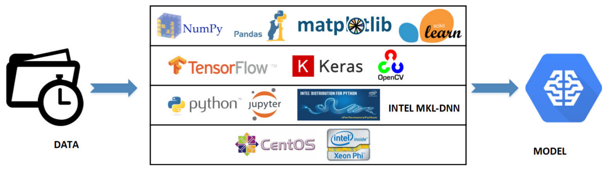


FIGURE 3. Cloud System Architecture.

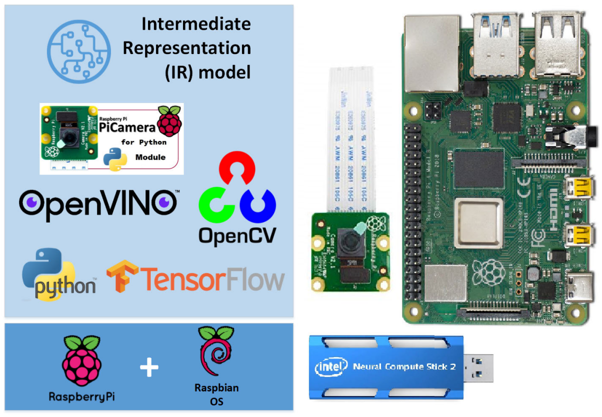

Intermediate Representation (IR) model
PiCamera for Python Module
OpenVINO
OpenCV
python
TensorFlow
RaspberryPi
+
Raspbian OS
intel Neural Compute Stick 2

FIGURE 4. Edge System Architecture and Raspberry 4 device.

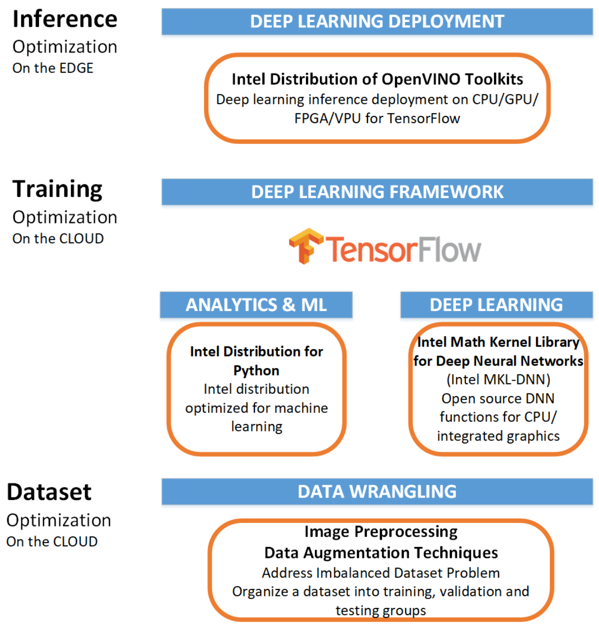

Inference
Optimization
On the EDGE
DEEP LEARNING DEPLOYMENT
Intel Distribution of OpenVINO Toolkits
Deep learning inference deployment on CPU/GPU/ FPGA/VPU for TensorFlow
Training
Optimization
On the CLOUD
DEEP LEARNING FRAMEWORK
TensorFlow
ANALYTICS & ML
DEEP LEARNING
Intel Distribution for Python
Intel distribution optimized for machine learning
Intel Math Kernel Library for Deep Neural Networks (Intel MKL-DNN)
Open source DNN functions for CPU/ integrated graphics
Dataset
Optimization
On the CLOUD
DATA WRANGLING
Image Preprocessing
Data Augmentation Techniques
Address Imbalanced Dataset Problem
Organize a dataset into training, validation and testing groups

FIGURE 5. Optimization Development.

for optimizing the inference process. Picamera is applied to connect with the camera module. Figure 4 describes the edge system architecture for inference process.

C. OPTIMIZATION DEVELOPMENT

Figure 5 describes the optimization development model used in this system. In dataset optimization, this system applied data wrangling to handle imbalanced datasets. This system utilized open AI software to accelerate the performance of

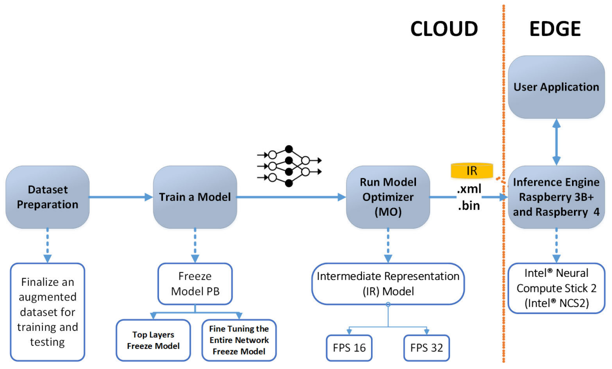

CLOUD
EDGE
User Application
Dataset Preparation
Train a Model
Run Model Optimizer (MO)
IR
.xml
.bin
Inference Engine Raspberry 3B+ and Raspberry 4
Finalize an augmented dataset for training and testing
Freeze Model PB
Top Layers Freeze Model
Fine Tuning the Entire Network Freeze Model
Intermediate Representation (IR) Model
FPS 16
FPS 32
Intel® Neural Compute Stick 2 (Intel® NCS2)

FIGURE 6. Workflow Diagram.

training and inference. The Intel Distribution of OpenVINO Toolkit facilitates model deployment for inference processing by converting and optimizing trained models for whichever hardware target is downstream [19], [20]. It offers support for models trained in TensorFlow, Caffe, and MXNet on CPU, integrated GPU, VPU (Movidius Myriad 2/Neural Compute Stick), and FPGA. The system used the TensorFlow framework to optimize deep learning libraries. The Data Analytics Acceleration Library and Intel Python distribution are essential building blocks for machine learning. The DNN (deep neural network) open-source libraries contain CPU optimized functions.

D. DATA PREPARATION, TRAINING, AND INFERENCE WORKFLOW

Figure 6 depicts the workflow diagram of this paper. There are two steps on the cloud edge system orchestration. First, in the training phase, the workflow starts from input data, creates a deep learning network, and produces the output classification. In this case, the system implemented three networks, InceptionV3, VGG16, and MobileNet. From this process, a trained model is produced. Second, in the inference phase, the workflow starts from new input from the camera and sensor, trained neural network model, and output the classification result.

E. USE CASE DATASET

The identification of dataset is based on the hottest wheels most stolen cars in US by year 2017 [40]. Therefore, we choose the top ten classes in this problem to shorten training time, as follows:

1) Honda Civic (1998): 45,062
2) Honda Accord (1997): 43,764
3) Ford F 150 (2006): 35,105
4) Chevrolet Silverado (2004): 30.056
5) Toyota Camry (2017): 17,276
6) Nissan Altima (2016): 13,358
7) Toyota Corolla (2016): 12,337
8) Dodge/Ram Pickup (2001): 12,004
9) GMC Sierra (2017): 10,865
10) Chevrolet Impala (2008): 9,487

FIGURE 7. VMMRdb dataset.

The number indicates number of stolen cars in each model in 2017.

The cars pictures are extracted from Vehicle Make and Model Recognition Dataset (VMMRdb) [6]. Then, we map multiple year vehicles to the stolen car category (based on exterior similarity) to provide more samples to work with. In this case, ten vehicle classification was selected based on the above problem, as follows:

1) Honda Civic (1997 – 1998)
2) Honda Accord (1996 – 1997)
3) Ford F150 (2005 – 2007)
4) Chevrolet Silverado (2003 – 2004)
5) Toyota Camry (2012 – 2014)
6) Nissan Altima (2013 – 2015)
7) Toyota Corolla (2011 – 2013)
8) Dodge Ram 1500 (1995 – 2001)
9) GMC Sierra 1500 (2007 – 2013)
10) Chevrolet Impala (2007 – 2009)

The images consist of 6877 files of 10 folders/classes based on the selected categories. For the training purpose, the dataset is divided into three categories of training, validation, and test using 0.7, 0.1, and 0.2 ratios, respectively. Training Data Set consists of 5098 images belonging to 10 classes. Validation Data Set consists of 586 images belonging to 10 classes. Test Data Set consists of 1193 images belonging to 10 classes. Figure 7 shows the sample of car dataset.

F. DATA PREPROCESSING AND AUGMENTATION

Based on the class distribution, it can be seen that certain classes were significantly lower than the others. There is a need to augment some of the dataset so that the dataset is more closely distributed. By using the augmentation techniques, the dataset can oversample minority classes in training set. This process would not apply invalidation or test in order not to create any bias on the data. From the data augmentation process, it can be seen there are four classes need to augment, toyota-camry, nissan-altima, toyota-corolla, and gmc-sierra.

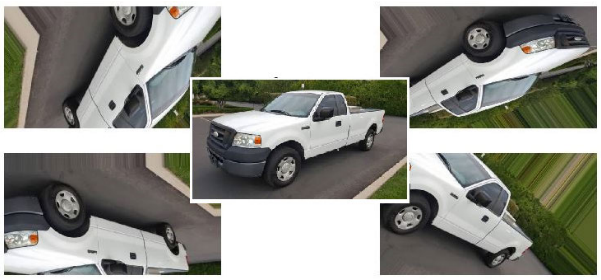


FIGURE 8. Image augmentation example.

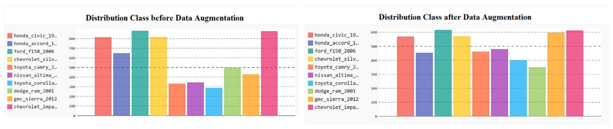


FIGURE 9. The comparison of class distribution before and after augmentation.

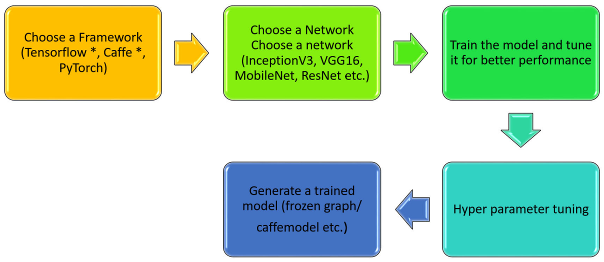

Choose a Framework (Tensorflow *, Caffe *, PyTorch)
Choose a Network Choose a network (InceptionV3, VGG16, MobileNet, ResNet etc.)
Train the model and tune it for better performance
Hyper parameter tuning
Generate a trained model (frozen graph/ caffemodel etc.)
FIGURE 10. Training Steps.

Figure 8 describes an example image and showing the effects of augmentation given a certain threshold of modification. The next step is to apply these random augmentations to the data.

Figure 9 shows the distribution in each class before and after data augmentation. It can be seen from the graph that the class distribution has been changed compared to before the augmentation process.

G. TRAINING STEPS

Figure 10 shows the steps of the training model. The design system chooses a TensorFlow framework in three topologies used in this system for model comparison, InceptionV3, VGG16, and MobileNet. The training model runs on the IPython Kernel CPU environment for better performance. Algorithm 1 and Figure 10 show the steps of training model.

H. INFERENCE STEPS

Algorithm 2 and Figure 11 shows the step of inference model.

I. MODEL OPTIMIZER

The redesigned Model Optimizer (MO) software is implemented as a Python code to convert the TensorFlow frozen graph into Intermediate Representation (IR) model. By using MO, it will improve performance and output. Also, using standard layers will get faster performance without the overhead of frameworks. In this project, there is two types of IR model, FPS16 and FPS32. Figure 12 shows the Model Optimizer process.

Algorithm 1 Training Process
```
1: $KMP - BLOCKTIME = 1$
2: $KMP - AFFINITY = granularity = fine, verbose, compact, 1, 0$
3: $KMP - SETTINGS = 1$
4: $OMP - NUM - THREADS = 8$
5: $optimizer = optimizers$
6: $Adam(lr = 0.001)$
7: $\leftarrow$ generate a trained model (frozengraph)
```

Algorithm 2 Inference Process
```
1: $plugin = IEPlugin(device - option) \leftarrow Load\ Plugin$
2: $net = IENetwork(model - xml, weightsmodel - bin) \leftarrow Read\ IR/Load\ Network$
3: $input - blob, out - blob = iternet.inputs \leftarrow Configure\ Input\ and\ Output$
4: $next(net.outputs)$
5: $n, c, h, w = net.inputs[input - blob].shape \leftarrow Load\ Model$
6: $exec - net = plugin.load(network = net)$
7: $inputs = input - blob : [cv2.resize(frame-, (w, h)).transpose((2, 0, 1))] \leftarrow Prepare\ input$
8: $res = exec - net.infer(inputs) \leftarrow Infer$
9: $res = res[out - blob]$
10: $top = res[0].argsort1 :][:: 1] \leftarrow Process\ Output$
11: $pred - label = labels[top[0]]$
```

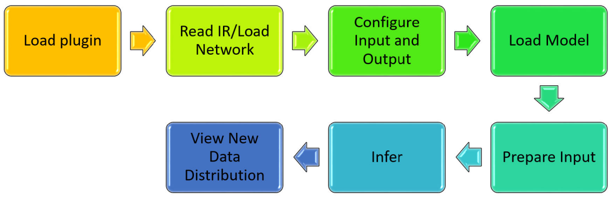


FIGURE 11. Inference Steps.

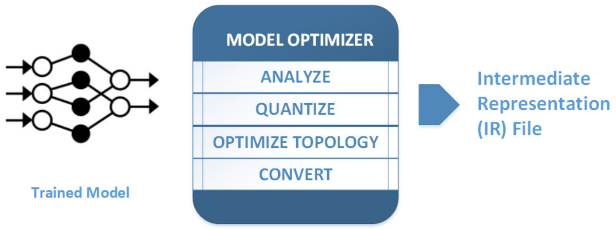


FIGURE 12. Model Optimizer.

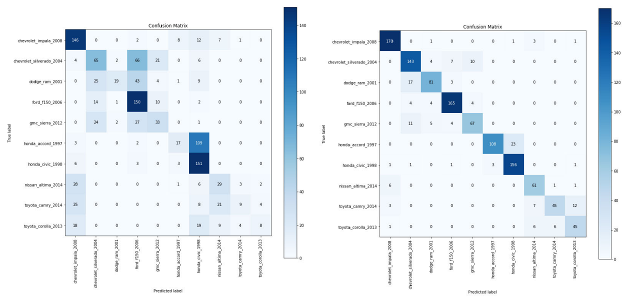
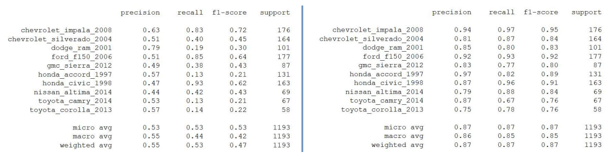


FIGURE 13. The comparison of Confusion Matrix before and after fine-tuning.



```
                          precision    recall  f1-score   support

     chevrolet_impala_2008       0.63      0.83      0.72       176
  chevrolet_silverado_2004       0.51      0.40      0.45       164
            dodge_ram_2001       0.79      0.19      0.30       101
           ford_f150_2006       0.51      0.85      0.64       177
          gmc_sierra_2012       0.49      0.38      0.43        87
        honda_accord_1997       0.57      0.13      0.21       131
         honda_civic_1998       0.47      0.93      0.62       163
       nissan_altima_2014       0.44      0.42      0.43        69
        toyota_camry_2014       0.53      0.13      0.21        67
      toyota_corolla_2013       0.57      0.14      0.22        58

                micro avg       0.53      0.53      0.53      1193
                macro avg       0.55      0.44      0.42      1193
             weighted avg       0.55      0.53      0.47      1193
```

```
                          precision    recall  f1-score   support

     chevrolet_impala_2008       0.94      0.97      0.95       176
  chevrolet_silverado_2004       0.81      0.87      0.84       164
            dodge_ram_2001       0.85      0.80      0.83       101
           ford_f150_2006       0.92      0.93      0.92       177
          gmc_sierra_2012       0.83      0.77      0.80        87
        honda_accord_1997       0.97      0.82      0.89       131
         honda_civic_1998       0.87      0.96      0.91       163
       nissan_altima_2014       0.79      0.88      0.84        69
        toyota_camry_2014       0.87      0.67      0.76        67
      toyota_corolla_2013       0.75      0.78      0.76        58

                micro avg       0.87      0.87      0.87      1193
                macro avg       0.86      0.85      0.85      1193
             weighted avg       0.87      0.87      0.87      1193
```

FIGURE 14. Classification Metrics before and after fine-tuning.

IV. EXPERIMENTAL RESULTS

This section discusses the experimental results. A confusion matrix, a classification report, a precision, f1, recall, and ROC graph are presented. The comparison of training and inference performance are compared among InceptionV3, VGG16, and MobileNet.

A. CONFUSION MATRIX

A standard graph to plot for analysis is a confusion matrix. It will play out the valid label and the predicted label on a diagram and color code the result accordingly. The ideal confusion matrix will have a diagonal line from the top left to the bottom right and no other color. A good pattern of diagonal line means that each predicted value matched the true value. The normal preview is that each class might lean toward one or two other categories that might look similar to the right class. Figure 13 presents the comparison of the confusion matrix between on top layer and fine-tuning. From the confusion matrix of the InceptionV3 classifier output, it can be seen that before fine-tuning, out of 1170 testing images, 627 are correctly classified, and 543 are misclassified. While after fine-tuning training, out of 1170 testing images, 1041 are correctly classified, and 129 are misclassified.

B. CLASSIFICATION REPORT

There are different main classification metrics, such as precision, recall, F1-Score, and support, to measure the classification model. Figure 14 shows the comparison of classification metrics before and after fine-tuning. It can be seen that there is an improvement in the classification metrics value before and after fine-tune.

C. PRECISION RECALL AND ROC

The Precision-Recall of the classification model is described in Figure 15. The micro average of Precision-Recall

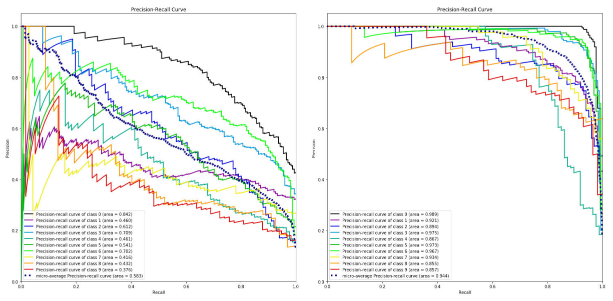


FIGURE 15. Precision-recall Curve before and after fine-tuning.

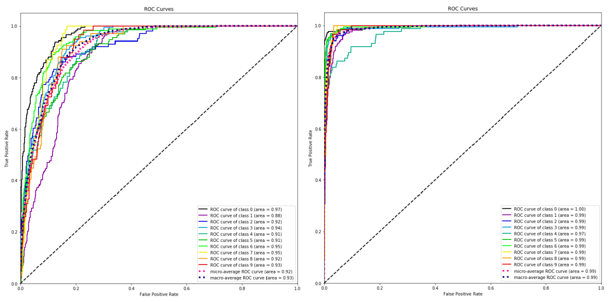
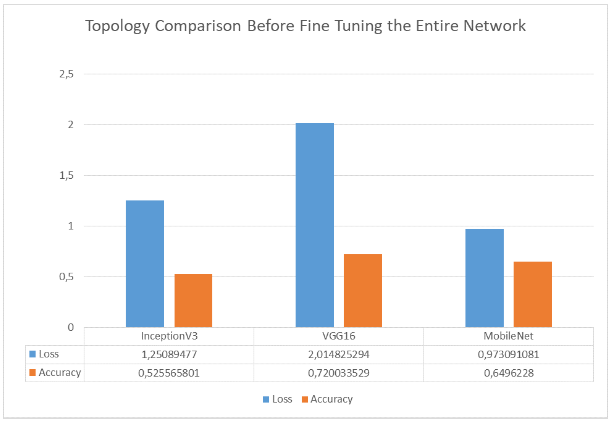


FIGURE 16. ROC Curve before and after fine tuning.



	InceptionV3	VGG16	MobileNet
Loss	1,25089477	2,014825294	0,973091081
Accuracy	0,525565801	0,720033529	0,6496228

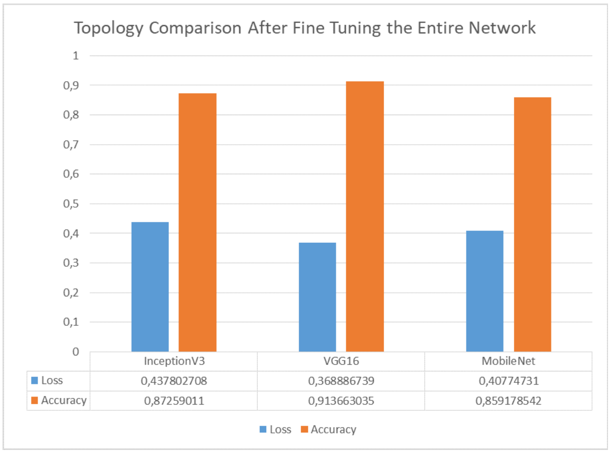
FIGURE 17. The Comparison of Loss and Accuracy before fine-tuning.

Curve before fine-tuning is 0.583, while after fine-tuning is 0.944.

The performance of a binary classifier system presented as ROC. Figure 16 demonstrates ROC curve per class. The micro average of the ROC curve before fine-tuning is 0.92, and the macro average of the ROC curve is 0.93. While after fine-tuning, both of the micro and macro averages of the ROC curve are 0.99.

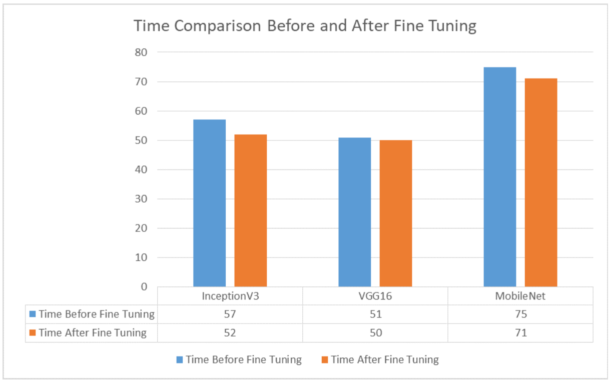
D. COMPARISON OF INCEPTIONV3, VGG16, AND MOBILENET

Figure 17 and 18 describes the comparison of InceptionV3, VGG16, and MobileNet topology before and after fine-tuning of the entire network. From the graphs, it can be seen that there is a significant improvement in the term of loss and accuracy.



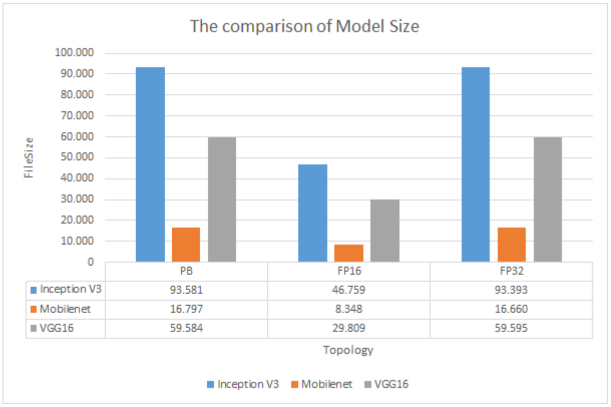
	InceptionV3	VGG16	MobileNet
Loss	0,437802708	0,368886739	0,40774731
Accuracy	0,87259011	0,913663035	0,859178542

FIGURE 18. **The Comparison of Loss and Accuracy after fine-tuning.**



	InceptionV3	VGG16	MobileNet
Time Before Fine Tuning	57	51	75
Time After Fine Tuning	52	50	71

FIGURE 19. **The Comparison of Time before and after Fine Tuning.**

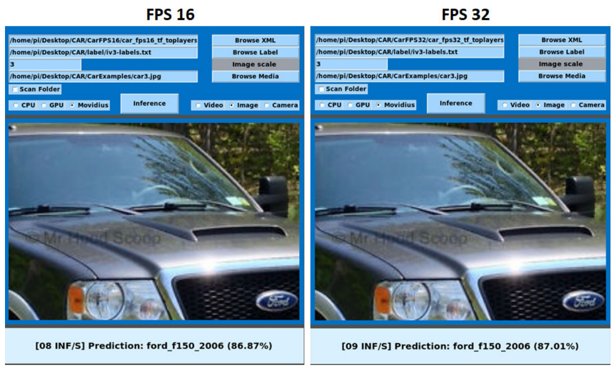


	PB	FP16	FP32
Inception V3	93.581	46.759	93.393
Mobilenet	16.797	8.348	16.660
VGG16	59.584	29.809	59.595

FIGURE 20. **The Comparison of Model Size.**

Figure 19 describes the comparison of the time needed to load the model. It can be seen from the graph that the MobileNet model consumes the highest time comparing to InceptionV3 and VGG16. Generally, the model loading time before and after fine-tuning has improved slightly.

Figure 20 shows the comparison of model size of InceptionV3, VGG16, and MobileNet topology. The InceptionV3 has the biggest model size, VGG16 is moderate, and MobileNet is the smallest model size. Based on the experiments, the model size before and after fine-tuning remains the same.



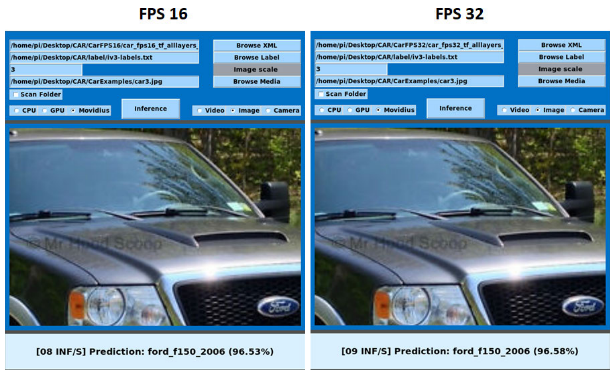
FPS 16:

/home/pi/Desktop/CAR/CarFPS16/car_fps16_tf_toplayers
/home/pi/Desktop/CAR/label/iv3-labels.txt
3
/home/pi/Desktop/CAR/CarExamples/car3.jpg
[08 INF/S] Prediction: ford_f150_2006 (86.87%)

FPS 32:

/home/pi/Desktop/CAR/CarFPS32/car_fps32_tf_toplayers
/home/pi/Desktop/CAR/label/iv3-labels.txt
3
/home/pi/Desktop/CAR/CarExamples/car3.jpg
[09 INF/S] Prediction: ford_f150_2006 (87.01%)

FIGURE 21. **The Inference of FPS16 and FPS32 before fine-tuning.**



FPS 16:

/home/pi/Desktop/CAR/CarFPS16/car_fps16_tf_alllayers_
/home/pi/Desktop/CAR/label/iv3-labels.txt
3
/home/pi/Desktop/CAR/CarExamples/car3.jpg
[08 INF/S] Prediction: ford_f150_2006 (96.53%)

FPS 32:

/home/pi/Desktop/CAR/CarFPS32/car_fps32_tf_alllayers_
/home/pi/Desktop/CAR/label/iv3-labels.txt
3
/home/pi/Desktop/CAR/CarExamples/car3.jpg
[09 INF/S] Prediction: ford_f150_2006 (96.58%)

FIGURE 22. **The Inference of FPS16 and FPS32 after fine-tuning.**

E. INFERENCE RESULTS

This study uses Neural Compute Stick 2 (NCS2) to enhance the performance of the Raspberry Pi 4 in processing computer vision. In this case, this project compared the inference result on Raspberry 4 using two kinds of IR model, FPS16 and FPS 32. Figure 21 shows the inference of FPS16 and FPS32 before fine-tuning. It can be seen that Frame per Second (FPS) value is eight on the FP16 model and nine on the FP32 model. The accuracy is 86.87% on the FP16 model and 87.01% on the FP32 model.

Figure 22 shows the inference of FPS16 and FPS32 after fine-tuning. It can be seen that Frame per Second (FPS) value is eight on the FP16 model and nine on the FP32 model. The accuracy is 96.53% on the FP16 model and 96.58% on the FP32 model.

V. CONCLUSION AND FUTURE WORKS

This paper demonstrates an optimized model of the cloud and edge computing environment for deep learning training and inference. The experiments conducted a deep learning training process on the cloud and inference process on edge. The comparisons result before and after fine-tuning were presented to examine the improvement in the training and inference phase. From the experiments, all model analysis value has been improved based on the implementation of fine-tuning. In comparing InceptionV3, VGG16, and MobileNet, there is a significant improvement before and after fine-tuning in the term of loss and accuracy. In terms of time, it can be seen from the result that the MobileNet model consumes the highest time comparing to InceptionV3 and VGG16. However, generally, the model loading time before and after fine-tuning has improved slightly. In the training phase, it can be seen that the confusion matrix, the classification report, the precision, f1, recall, and ROC graph are improved. In the inference phase, it can be seen that the accuracy and FPS have been enhanced after applied the fine-tuning model on both FP16 and FP32.

To sum up, MobileNet was the least accurate model (85%) but had the smallest model size (17,2MB) and the longest time to load the model (71s). VGG16 was the most accurate (91%) with the moderate model size (61MB) and the shortest time to load the model (50s). InceptionV3 has median accuracy (87%) with the most significant model size (95,8MB) and the average time to load the model (52s).

In the future, this model performance can be applied in different image classification cases. The training performance can be compared with the GPU environment. The security issue on edge computing might be a challenge research problem. Also, find another solution for the edge side, such as comparing with the Jetson NANO environment.

REFERENCES

[1] C. Zhang, F. Liu, and Y. He, "Identification of coffee bean varieties using hyperspectral imaging: Influence of preprocessing methods and pixel-wise spectra analysis," *Sci. Rep.*, vol. 8, no. 1, p. 2166, 2018.

[2] M. Poostchi, K. Silamut, R. J. Maude, S. Jaeger, and G. Thoma, "Image analysis and machine learning for detecting malaria," *Transl. Res.*, vol. 194, pp. 36–55, Apr. 2018.

[3] J. Chaki and N. Dey, *A Beginner's Guide to Image Preprocessing Techniques*. Boca Raton, FL, USA: CRC Press, 2018.

[4] M. Frid-Adar, I. Diamant, E. Klang, M. Amitai, J. Goldberger, and H. Greenspan, "GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification," *Neurocomputing*, vol. 321, pp. 321–331, Dec. 2018.

[5] H. Salehinejad, S. Valaee, T. Dowdell, and J. Barfett, "Image augmentation using radial transform for training deep neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2018, pp. 3016–3020.

[6] F. Tafazzoli, H. Frigui, and K. Nishiyama, "A large and diverse dataset for improved vehicle make and model recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jul. 2017, pp. 1–8.

[7] *Intelligent Devices, Smarter Decisions*. Accessed: Mar. 20, 2019. [Online]. Available: https://www.intel.com/content/www/us/en/homepage.html

[8] *Conda Documentation*. Accessed: Mar. 15, 2019. [Online]. Available: https://docs.conda.io/en/latest/

[9] *The Jupyter Notebook*. Accessed: Mar. 15, 2019. [Online]. Available: https://jupyter.org/

[10] *IPython Documentation*. Accessed: Mar. 15, 2019. [Online]. Available: https://ipython.readthedocs.io/en/stable/install/kernel-install.html

[11] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, and T. Chen, "Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes," Jul. 2018, *arXiv:1807.11205*. [Online]. Available: https://arxiv.org/abs/1807.11205

[12] Q. Zhang, L. T. Yang, Z. Yan, Z. Chen, and P. Li, "An efficient deep learning model to predict cloud workload for industry informatics," *IEEE Trans. Ind. Informat.*, vol. 14, no. 7, pp. 3170–3178, Jul. 2018.

[13] A. Caliskan, M. Yuksel, H. Badem, and A. Basturk, "Performance improvement of deep neural network classifiers by a simple training strategy," *Eng. Appl. Artif. Intell.*, vol. 67, pp. 14–23, Jan. 2018.

[14] Y. Liang, Z. Cai, J. Yu, Q. Han, and Y. Li, "Deep learning based inference of private information using embedded sensors in smart devices," *IEEE Netw.*, vol. 32, no. 4, pp. 8–14, Jul. 2018.

[15] H. Li, K. Ota, and M. Dong, "Learning IoT in edge: Deep learning for the Internet of Things with edge computing," *IEEE Netw.*, vol. 32, no. 1, pp. 96–101, Jan. 2018.

[16] C.-T. Yang, S.-T. Chen, W. Den, Y.-T. Wang, and E. Kristiani, "Implementation of an intelligent indoor environmental monitoring and management system in cloud," *Future Gener. Comput. Syst.*, vol. 96, pp. 731–749, Jul. 2019.

[17] E. Kristiani, C. T. Yang, Y. T. Wang, and C. Y. Huang, "Implementation of an edge computing architecture using OpenStack and Kubernetes," in *Proc. Int. Conf. Inf. Sci. Appl.* Singapore: Springer, Jun. 2018, pp. 675–685.

[18] E. Kristiani, C. T. Yang, C. Y. Huang, Y. T. Wang, and P. C. Ko, "The implementation of a cloud-edge computing architecture using OpenStack and Kubernetes for air quality monitoring application," *J. Mobile Netw. Appl.*, to be published.

[19] *Intel Distribution of OpenVINO Toolkit*. Accessed: Apr. 24, 2019. [Online]. Available: https://software.intel.com/enus/articles/OpenVINO-Using-TensorFlow

[20] *Intel Neural Compute Stick*. Accessed: Apr. 24, 2019. [Online]. Available: https://software.intel.com/en-us/movidius-ncs

[21] M. A. Zamora-Izquierdo, J. Santa, J. A. Martínez, V. Martínez, and A. F. Skarmeta, "Smart farming IoT platform based on edge and cloud computing," *Biosyst. Eng.*, vol. 177, pp. 4–17, Jan. 2019.

[22] S. Wang, Y. Zhao, J. Xu, J. Yuan, and C.-H. Hsu, "Edge server placement in mobile edge computing," *J. Parallel Distrib. Comput.*, vol. 127, pp. 160–168, May 2019.

[23] C. Li, J. Bai, and J. Tang, "Joint optimization of data placement and scheduling for improving user experience in edge computing," *J. Parallel Distrib. Comput.*, vol. 125, pp. 93–105, Mar. 2019.

[24] L. Greco, P. Ritrovato, and F. Xhafa, "An edge-stream computing infrastructure for real-time analysis of wearable sensors data," *Future Gener. Comput. Syst.*, vol. 93, pp. 515–528, Apr. 2019.

[25] R. Morabito, R. Petrolo, V. Loscrí, and N. Mitton, "Reprint of LEGIoT: A lightweight edge gateway for the Internet of Things," *Future Gener. Comput. Syst.*, vol. 92, pp. 1157–1171, Mar. 2019.

[26] C.-T. Yang, C.-W. Huang, and S.-T. Chen, "Improvement of workload balancing using parallel loop self-scheduling on Intel Xeon Phi," *J. Supercomput.*, vol. 73, no. 11, pp. 4981–5005, Nov. 2017.

[27] C.-T. Yang, C.-J. Chen, Y.-T. Tsan, P.-Y. Liu, Y.-W. Chan, and W.-C. Chan, "An implementation of real-time air quality and influenza-like illness data storage and processing platform," *Comput. Hum. Behav.*, vol. 100, pp. 266–274, Nov. 2019.

[28] C.-T. Yang, S.-T. Chen, J.-C. Liu, R.-H. Liu, and C.-L. Chang, "On construction of an energy monitoring service using big data technology for the smart campus," *Cluster Comput.*, vol. 23, no. 1, pp. 265–288, Mar. 2020.

[29] C.-T. Yang, Y.-W. Chan, J.-C. Liu, and B.-S. Lou, "An implementation of cloud-based platform with R packages for spatiotemporal analysis of air pollution," *J. Supercomput.*, pp. 1–22, Nov. 2017, doi: 10.1007/s11227-017-2189-1.

[30] C.-T. Yang, J.-C. Liu, S.-T. Chen, and H.-W. Lu, "Implementation of a big data accessing and processing platform for medical records in cloud," *J. Med. Syst.*, vol. 41, no. 10, p. 149, 2017.

[31] M. Chui, "Artificial intelligence the next digital frontier?" McKinsey Global Inst., New York, NY, USA, Tech. Rep., Feb. 2019, vol. 47. [Online]. Available: https://www.calpers.ca.gov/docs/board-agendas/201801/full/day1/06-technology-background.pdf

[32] C.-S. Shih, J.-J. Chou, and K.-J. Lin, "WuKong: Secure Run-Time environment and data-driven IoT applications for smart cities and smart buildings," *J. Internet Serv. Inf. Secur.*, vol. 2, no. 8, pp. 1–17, 2018.

[33] T. Robles, R. Alcarria, D. M. de Andrés, M. N. de la Cruz, R. Calero, S. Iglesias, and M. López, "An IoT based reference architecture for smart water management processes," *JoWUA*, vol. 1, no. 6, pp. 4–23, 2015.

[34] N. K. Giang, J. Im, D. Kim, M. Jung, and W. Kastner, "Integrating the EPCIS and building automation system into the Internet of Things: A lightweight and interoperable approach," *JoWUA*, vol. 1, no. 6, pp. 56–73, 2015.

[35] *Advanced Guide to Inception V3 on Cloud TPU*. Accessed: Jan. 10, 2019. [Online]. Available: https://cloud.google.com/tpu/docs/inception-v3-advanced

[36] P. Zhang, Q. Zhao, J. Gao, W. Li, and J. Lu, ''Urban street cleanliness assessment using mobile edge computing and deep learning,'' *IEEE Access*, vol. 7, pp. 63550–63563, 2019.
[37] M. Z. Khan, S. Harous, S. U. Hassan, M. U. Ghani Khan, R. Iqbal, and S. Mumtaz, ''Deep unified model for face recognition based on convolution neural network and edge computing,'' *IEEE Access*, vol. 7, pp. 72622–72633, 2019.
[38] C.-K. Tsung, H.-Y. Hsieh, and C.-T. Yang, ''An implementation of scalable high throughput data platform for logging semiconductor testing results,'' *IEEE Access*, vol. 7, pp. 26497–26506, 2019.
[39] C.-T. Yang, S.-T. Chen, W.-H. Cheng, Y.-W. Chan, and E. Kristiani, ''A heterogeneous cloud storage platform with uniform data distribution by software-defined storage technologies,'' *IEEE Access*, vol. 7, pp. 147672–147682, 2019.
[40] *2017 Hot Wheels Report*. Accessed: Jul. 15, 2018. [Online]. Available: https://www.nicb.org/news/news-releases/2017-hot-wheels-report

ENDAH KRISTIANI received the M.S. degree in electrical engineering (information technology) from Universitas Gadjah Mada, Yogyakarta, Indonesia, in 2007. She is currently pursuing the Ph.D. degree with the Department of Industrial Engineering and Enterprise Information and join the High Performance Computing Laboratory, Tunghai University, Taichung, Taiwan. In August 2007, she joined the Faculty of Engineering and Computer Science, Department of Informatics Engineering, Krida Wacana Christian University (UKRIDA), Jakarta.

CHAO-TUNG YANG (Member, IEEE) received the Ph.D. degree in computer science from National Chiao Tung University, in July 1996. In August 2001, he joined the Faculty of the Department of Computer Science, Tunghai University. He is currently a Distinguished Professor of computer science, Tunghai University, Taiwan. He is serving in a number of journal editorial boards, including *Future Generation Computer Systems*, *International Journal of Communication Systems*, *KSII Transactions on Internet and Information Systems*, and *Journal of Cloud Computing*. He has published more than 300 articles in journals, book chapters, and conference proceedings. His present research interests are in cloud computing, big data, parallel computing, and deep learning. He is a member of the IEEE Computer Society and ACM.

CHIN-YIN HUANG received the Ph.D. degree from Purdue University, USA. He is currently a Professor and the Department Chairman of Industrial Engineering and Enterprise Information with Tunghai University, Taiwan. His publications appear in *International Journal of Production Research*, *International Journal of Production Economics*, *Computers in Industry*, *Computers and Industrial Engineering*, *Robotics and Computer-Integrated Manufacturing*, *Epilepsy Research, Production Engineering*, *Engineering Computations*, and so on. He also coauthored chapters for *Handbooks of Industrial Engineering, Handbook of Industrial Robotics* and *Handbook of Automation*. He has coauthored two books in industrial engineering and management published in Taiwan. His research interests include healthcare management, clinical data analysis, distributed manufacturing systems, manufacturing process optimization, and industry 4.0.