논문 2020-15-07

Development of a Low-cost Industrial OCR System with an End-to-end Deep Learning Technology

Bharat Subedi, Jahongir Yunusov, Abdulaziz Gaybulayev, Tae-Hyong Kim*

Abstract : Optical character recognition (OCR) has been studied for decades because it is very useful in a variety of places. Nowadays, OCR's performance has improved significantly due to outstanding deep learning technology. Thus, there is an increasing demand for commercial-grade but affordable OCR systems. We have developed a low-cost, high-performance OCR system for the industry with the cheapest embedded developer kit that supports GPU acceleration. To achieve high accuracy for industrial use on limited computing resources, we chose a state-of-the-art text recognition algorithm that uses an end-to-end deep learning network as a baseline model. The model was then improved by replacing the feature extraction network with the best one suited to our conditions. Among the various candidate networks, EfficientNet-B3 has shown the best performance: excellent recognition accuracy with relatively low memory consumption. Besides, we have optimized the model written in TensorFlow's Python API using TensorFlow-TensorRT integration and TensorFlow's C++ API, respectively.

Keywords : Embedded systems, Optical character recognition, Deep learning, End-to-end approach, Low-cost implementation

Ⅰ. Introduction

These days deep learning demonstrates outstanding performance in various applications such as computer vision, speech recognition, natural language processing, and games. Deep learning enables automatic processing of data with highly nonlinear and complex feature abstraction using a great number of sequentially connected layers, instead of manual extracting the optimum feature representation of data with professional domain knowledge. With automatic and high-level feature learning from a large amount of data, Deep learning is now considered as a state-of-the-art analysis tool for industry in the so-called Industry 4.0 era.

Deep learning has also revived interests on the optical character recognition (OCR) task and has achieved good successes [1-3]. The OCR has been a complicated problem due to various characters in different fonts, complex rules of languages, off-focus images, distortion by a camera, and so on. OCR based systems apply various techniques to overcome these difficulties. The traditional approach to the OCR is to divide into two different stages, text detection and text recognition, and handle them separately [4, 5]. Nowadays, deep learning-based methods show superior performance in both two areas. For text detection or spotting, convolutional neural networks (CNNs) are commonly used for

*Corresponding Author (taehyong@kumoh.ac. kr)
Received: Mar. 1, 2020, Revised: Mar. 23, 2020, Accepted: Mar. 30, 2020.
B. Subedi, J. Yunusov: Kumoh National Institute of Technology (M.E. Student)
A. Gaybulayev: Kumoh National Institute of Technology (Ph.D. Student)
T.-H. Kim: Kumoh National Institute of Technology (Prof.)
※ This paper was supported by Research Fund, Kumoh National Institute of Technology.

 2020 Apr. 15(2) 51-60
ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2020.15.2.51

feature extraction of target images including text [6, 7]. For text recognition, recurrent neural networks (RNNs) are often used to predict text sequentially in the text regions spotted by text detection. Text recognition, therefore, costs much more time than text detection, especially if target images have many text regions. Lately, there have been a few trials to combine those two stages where some trainable parameters are shared [8-11]. This end-to-end deep learning approach is getting more attention as it is good for convenient manipulation and better optimization. Integrating those two stages, however, requires sophisticated designing for optimum performance as they are quite different in various aspects.

For small-sized enterprises, there are several challenges in the development of deep learning-based OCR systems: design complexity, hardware requirements, data preparation, and so on. Especially, they usually have difficulties in hiring experts required to develop deep learning systems. Fortunately, there are currently a lot of open sources of OCR especially using deep learning technology, and deep learning frameworks and hardware platforms supporting embedded systems are rapidly emerging. The development of low-cost embedded systems for deep learning-based OCR is thus getting to have reality.

In this study, we designed a low-cost industrial OCR system with end-to-end deep learning techniques suitable for embedded systems. Our goal is to develop a real-time OCR system with commercial-grade performance. We tested many DL-based OCR techniques especially end-to-end based ones by aiming at implementing a low-cost embedded system. As the reference model, the unified end-to-end trainable fast oriented text spotting (FOTS) network was selected [11]. We tried to improve the model according to three metrics: recognition accuracy for better performance, time and memory spent by recognition in the target embedded system for lightweight implementation. Our experiences obtained from this study are presented and discussed in this paper.

In section 2, the architecture and design methodology of the target OCR system is explained. Implementation details of the OCR system are given in section 3, including data manipulation, training process, and lightweight optimization. Performance evaluation of the system is presented with some discussion in section 4. Finally, section 5 concludes this study.

Ⅱ. Architecture and Methodology

In order to develop a low-cost lightweight OCR system which is operating under the strict requirements of embedded systems in terms of memory consumption and computational load, each part of the whole system needs to be carefully designed. This section presents our design methods for three main parts of the OCR system, deep learning framework, embedded system hardware, and deep learning network.

1. Deep Learning Framework

The first step of developing a deep learning application may be to decide a deep learning framework in which neural networks are designed and trained. Currently, there are about 20 open-source or proprietary deep learning frameworks [12]. They are usually written in C++ or Python but also provide interfaces for other languages such as Java and Scala. The most popular framework, TensorFlow is written in Nvidia's compute unified device architecture (CUDA), C++, and Python, and also supports various languages including Java for Andriod OS and Javascript for Web interface. While C and C++ are standard languages for embedded systems, there are little open sources in those languages for deep learning applications because they are not convenient for matrix

manipulations used frequently in training and handling neural networks. The same situation applies to PyTorch, the second most popular framework.

In order to reduce the development time of the OCR system, we decided to make use of an open-source application as a baseline. So, C++ based framework cannot be the first option. Even if Python is used for fast implementation, we needed to consider later optimization for fast and stable operation in embedded systems. We, therefore, decided to use TensorFlow which supports both Python and C++. Furthermore, it provides several techniques to build lightweight implementation such as TensorFlow Lite and TensorFlow/TensorRT integration for increasing inference speed. Note that TensorRT is a high-performance deep learning interface for embedded systems, developed by Nvidia. Both the techniques speed up deep learning inference by calculating at a lower precision.

2. Embedded System Hardware

Most of the deep learning frameworks support GPU acceleration for fast training. TensorFlow natively supports CUDA, a parallel computing platform and application programming interface (API) model created by Nvidia. Since we aim at the development of a real-time OCR application based on an embedded system, a specific embedded system hardware platform supporting GPU acceleration needs to be decided for implementation.

The development board provided by Nvidia is naturally the first choice in our situation. There were three options: Jetson Nano, TX2, and AGX Xavier, in order of cost, from the cheapest one. Jetson Nano developer kit, which is about at $100, has a 128-core GPU and 4GB memory shared by CPU and GPU. The number of GPU cores and memory size is doubled at the next expensive model. Google's edge TPU devices such as the Coral development board or Raspberry Pi with Coral USB accelerator may be another choice. They are faster than Jetson Nano at a similar price, but they support TensorFlow Lite only. We, therefore, decided to use Nvidia's Jetson Nano. The development of deep learning-based OCR system under such very limited hardware resources is challenging.

3. Deep Learning Network

In order to rapidly develop a lightweight OCR system with a compact structure but having very good performance, we decided to use an end-to-end deep learning approach. We examined the existing deep learning-based text recognition algorithms with an end-to-end approach to choose a baseline model for our development.

M. Liao et al. proposed an end-to-end text recognition approach by combining their own TextBoxes++ [13] for text detection and the existing convolutional RNN based word recognition [14]. TextBoxes++ provides fast detection of arbitrary-oriented scene text with a very simple detection pipeline. It uses VGG-16 [15] as a backbone network for feature extraction, ten subsequent convolutional layers, and 6 text-box layers connected to 6 intermediate convolutional layers. Even if it shows a remarkable text localization accuracy, it is not appropriate for our target as its detection network is too heavy, and there is no tight integration between text detection and recognition parts.

STN-OCR [8] and Deep TextSpotter [9] are also single neural networks for text detection and recognition which show outstanding performance. As both of them use CNN for text recognition, they are remarkably lightweight and have a more integrated structure. FOTS [11] is a latest simultaneous detection and recognition network which shows state-of-the-art performance. Furthermore, it tightly integrates detection and recognition by sharing computation and visual information between them. It uses ResNet-50 [16] as a backbone feature extraction network, which shows a satisfactory performance with a very

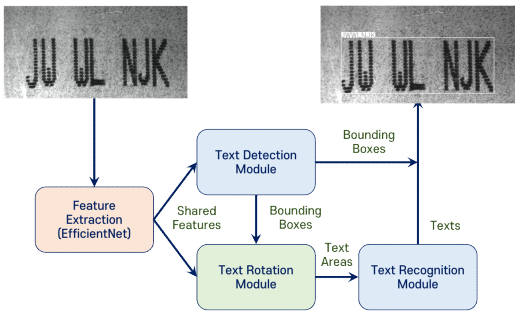

Fig. 1 The architecture of test recognition networks

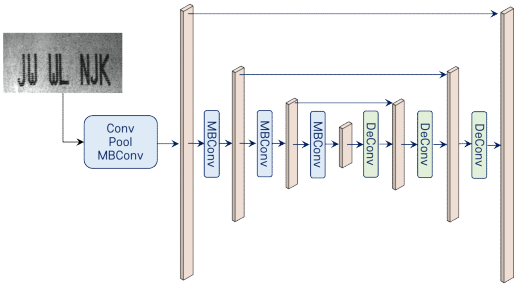

Fig. 2 The architecture of shared feature extraction network

deep structure but has modest complexity. For increasing recognition accuracy, it uses sequential convolutions, bi-directional RNN, and CTC decoder together for text recognition. We decided to choose the FOTS as our baseline model to obtain the best recognition quality and to customize and optimize it for our purpose.

The main optimization target of FOTS for our OCR system is the feature extraction module in the text detection part. FOTS already provides a real-time version named “FOTS RT” to accelerate the inferencing speed by replacing ResNet-50 by ResNet-34. But when we tested FOTS with such replacement, it has shown a considerable amount of performance degradation while inferencing speed is much faster. We tested many lightweight backbone networks for feature extraction including PVAnet [17] and various versions of MobileNets [18-20], which are known for satisfying performance, but the results are similar to FOTS RT. Lately, we found an interesting study to increase the performance by carefully balancing network depth, width, and resolution with a simple scaling method [21]. This study also provides a set of networks designed to efficiently use the scaling method called EfficientNets, which shows great performance improvements with controlled numbers of model parameters. As our evaluation of EfficientNets has verified such improvements, we decided to use those networks as the backbone feature extraction model of our OCR system. Fig. 1 and Fig. 2 show the architecture of our text recognition network based on FOTS and the shared backbone network both for text detection and recognition, respectively.

As shown in Fig. 2, our feature extraction network uses several mobile inverted bottleneck convolution (MBConv) blocks [22, 23] with squeeze-and-excitation optimization [24] instead of ResNet blocks.

Ⅲ. Implementation

The OCR system is necessary for automation of the manufacturing processes in the industry, especially for product inspection. In order to identify a product, a specific code is printed on its surface. In order to make sure that correct products are used in assembling, it is necessary to identify products, especially in automatic methods. Deep learning-based OCR systems can enhance recognition accuracy and robustness in different conditions caused by luminance, vibration, and light reflection if a large amount of data considering such conditions is prepared for training.

We tried to implement a commercial-grade OCR system that satisfies those industrial-level requirements. The target OCR system is composed of four parts: a public embedded system development board with GPU support, a vision camera and its interface, management software with a graphical user interface (GUI),

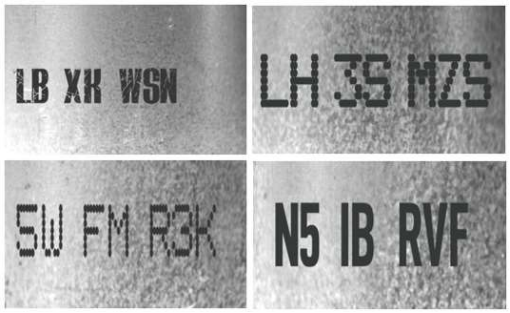

Fig. 3 Samples of the dataset with a different font, color, and background

and a deep learning-based recognition engine. The steps to implement the whole system are as follows: preparation of training data, implementation of text recognition network and training, completion of OCR software and hardware, and additional system optimization.

1. Data Collection and Augmentation

Qualified training and validation datasets are essential to developing a high-accuracy OCR system. Generally, a large number of labeled images are required to train a deep neural network from random initial weights. Capturing enough number of images from real products and labeling them requires a lot of manual work, which is actually impossible in small-sized enterprises. Therefore, we made a simple program to automatically generate artificial synthetic images of industrial products with arbitrary texts, by which the label information can be directly obtained. The program changes fonts, sizes, locations, character gaps, aspect ratios, and colors of texts. It also changes background product images and their brightness. By the combination of all possible changes, it can generate a huge number of training images. Fig. 3 shows samples of synthetic images generated by the data augmentation program.

In order to obtain original product images for validation, about 100 real products were used. Those products were also used to get background images required for the data augmentation process. We used 30K (1600×1200) pixel-sized synthetic grayscale images for training.

In order to obtain more realistic images, we did some post-processing on the generated images. For simulating partial light reflection on product surfaces, partial off-focusing, and camera noise, we performed highlighting, blurring, and noise adding on some random areas of randomly selected images.

2. Training and Testing

Training and testing of the implemented deep learning network with prepared data should be performed in two phases. In the first phase, the powerful server with multiple high-performance GPUs is used in rapid training and validation. We used an AMD server with 1920X CPU and 2 Nvidia's RTX 2080ti GPUs. As the deep learning framework, the GPU version of TensorFlow 1.14 was used. We customized the network for the target OCR application area. As target text is a product code, not a word, we limited the vocabulary of recognition text as characters only. Additionally, we simplified the shape of the bounding box, from arbitrary quadrangles to rectangles as text areas are usually rectangles in conditions of text recognition in the industry. These customizations could reduce training time efficiently. The Inferencing speed of our text recognition network is about 15FPS in that server.

The second stage is to copy the text recognition network and trained weights to the target embedded system. We tested Nvidia's Jetson TX2 and Nano. In TX2 with 8GB shared memory, text recognition takes less than 0.5 seconds without any memory shortage problem. However, at first, Nano's 4GB memory was not enough for the stable running of our recognition process, even if virtual memory is used. By configuring the options of TensorFlow's session to minimize memory usage such as allow_growth and allow_soft_placement, we managed to control

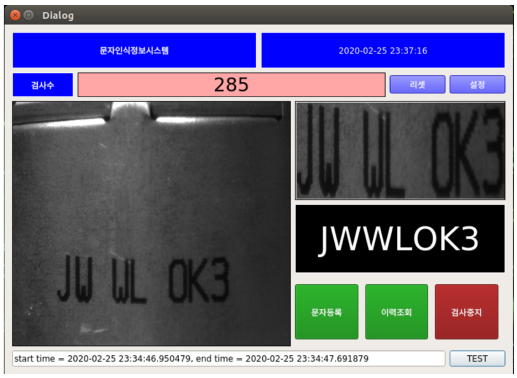

Fig. 4 A screenshot of OCR software

the memory usages within Nano's native memory. The exact memory usage and elapsed recognition time at Nano will be presented in the next section with some comparative analysis.

3. OCR Software and Hardware

An OCR system has to be equipped with software that provides users with control and management of the text recognition work. The GUI interface is necessary for the software to display product images and predicted results. Especially, OCR software for the industry may need to support history data management for text recognition and product inspection. The software of our OCR system has a window displaying product image taken by a camera, another two windows for showing detected text area and recognized text. It also has several buttons to control and manage recognition work. Fig. 4 shows a screenshot of the OCR software in our system which has a PyQT-based GUI.

While OCR software including the text recognition module is running in an embedded system board, the OCR system for industry needs its own housing to cover that processing board. The complete OCR system with housing also includes a power supply board, vision camera, and its interface, cooling equipment, and other modules for interface with external systems. We used a Basler's vision camera

Fig. 5 The complete OCR system

1600-60gm having an Ethernet interface that captures 1600×1200 sized images. Fig. 5 shows the inside of the complete OCR system with housing.

4. Further Optimization

The inference network we used for text recognition is developed using Python API of TensorFlow 1.14 Python API. Fortunately, it is running fine in Nvidia's Jetson Nano. In order to accelerate inferencing speed and to reduce memory consumption for better performance and stability, we tried to apply several feasible optimization techniques.

First, we attempted conversion to TensorRT from TensorFlow as the Nvidia's Jetson platform natively supports TensorRT. TensorRT's Python API, however, is totally different from TensorFlow's Python API. We need to parse the model into TensorRT using TensorFlow/UFF parser, but some TensorFlow's APIs are not fully supported. Due to the considerable burden of manual conversion, we moved to TensorFlow-TensorRT (TF-TRT) integration supported by TensorFlow. With this integration, TensorFlow code unsupported by TensorRT is automatically run by TensorFlow instead. We froze the inference graph in TensorFlow and created and ran the TensorRT-optimized inference graph. In order for speed-up, TF-TRT integration supports lower precision modes: FP16 and INT8. We succeeded in conversion and running the converted models. The comparison results of TF-TRT integration of our OCR system using EfficientNet-B3 backbone in the aspects

Table 1. Performance of TF-TRT integration

model	inferencing time (s)	recognition accuracy(%)	model size (MB)
Frozen model	0.073	99.18	51.4
TF-TRT (FP32)	0.073	99.14	102.0
TF-TRT (FP16)	0.070	99.04	85.9
TF-TRT (INT8)	3.230	96.80	51.4

of the inferencing time, recognition accuracy, and model sizes are summarized in Table 1.

This experiment was done in an x86 CPU server with Nvidia's RTX 2080ti GPUs for convenience. The TF-TRT integration model at the precision of FP16 inferences slightly faster than the frozen model while it consumes more memory and shows a slightly lower accuracy. TF-TRT integration at INT8 precision fails to optimize the frozen model, so it has a very long inference time. Actually, Jetson Nano also does not support the INT8 precision model in TensorRT. Meanwhile, conversion to TensorFlow Lite model has been skipped as it also supports a limited subset of TensorFlow APIs.

Finally, we tried to convert the programming language used in the OCR software from Python to C++ to obtain two goals. First, a program written in C++ is usually faster than the corresponding program written in Python. Second, C++ is a compiled programming language that generates binary code, while Python interpreters reveal the source code. Commercial software, therefore, is usually developed using a compilation language. Language conversion to C++ in the TensorFlow framework also requires much manual work, unfortunately. The conversion process is similar to previous optimization methods. We need to prepare a frozen graph of the model and the trained weights. Then a new graph model to use that frozen graph needs to be defined in C++. The significant conversion difficulties lie in OpenCV code for some computer vision tasks and Numpy code

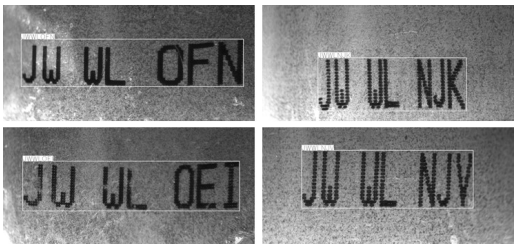

Fig. 6 Samples of real product images with recognized characters

for array manipulations. There are no simple conversion methods from OpenCV's matrices or Numpy's arrays to TensorFow's Tensors. Furthermore, The code written in Numpy's concise syntax for rapid and efficient array handling needs to be converted in completely manual ways. There are some libraries to provide APIs for Numpy code conversions such as xTensor [25] and NumCpp [26]. Although xTensor supports extensive Numpy syntax, its Numpy support is still limited, and its own array type, Xarray, makes conversion complicated sometimes. In a long time of manual conversion work, we finally generated the OCR software written in pure C++. While the second goal of our conversion has been realized, achieving the first goal is not satisfactory. During manual conversion, we had to use many for-loops since we did not find better solutions. Actually, TensorFlow C++ models should be optimized very carefully to show the best performance.

IV. Evaluation

We evaluated the implemented OCR system based on Nvidia's Jetson Nano. 1400 real images were used to measure the performance of various implementations with different configurations. Fig. 6 shows samples of real product images with recognized characters.

In order to decide the best backbone feature extraction network used for both text detection and recognition in our low-cost OCR system, we tested famous lightweight CNNs

Table 2. Performance Comparison

	memory (GB)	time (sec)	string accuracy	character accuracy
RN50	3.12	0.90	93.42%	98.79%
MN1	3.01	0.66	47.28%	87.59%
MN3	2.92	0.55	48.72%	91.14%
EN0	2.93	0.67	93.28%	98.77%
EN1	2.95	0.75	93.72%	99.01%
EN2	2.98	0.81	94.12%	99.09%
EN3	3.03	0.87	94.86%	99.19%

which are known to have excellent performance.

Candidates for evaluations are ResNet-50 (RN50) with 25M parameters, MobileNet-V1 (MN1) with 5.8M parameters, MobileNet-V3 (MN3) with 5.9M parameters, EfficientNet-B0 (EN0) with 6M parameters, EfficientNet-B1 (EN1) with 7.8M parameters, EfficientNet-B2 (EN2) with 9M parameters, and EfficientNet-B3 (EN3) with 12M parameters. Note that we have not included heavier versions of EfficientNet, from B4 to B7, according to the performance curve the authors provided. Those versions require much more memory to get higher accuracy. Performance metrics are the used memory size, inferencing time, and recognition accuracies with respect to the whole code and each character. Table 2 shows the evaluation results. Note that we resized input images to 800x600 pixel size in order to reduce inferencing time for real-time uses.

Table 2 shows that all candidate networks consume the memory under 3.2GB, which indicates that they can be used as the backbone networks of our Nano-based OCR system. The Inferencing times of all the candidates are less than 1 second. This inferencing time may not be regarded as a real-time speed, but 1 second may be acceptable considering the product inspection environment of the industry using conveyor-belt systems. Among the candidates, MobileNet-V1 is the lightest, and ResNet-50 is the heaviest. Regarding the accuracy, MobileNets do not show usable levels of accuracy. EfficientNets outperform ResNet-50 from the version B1 (EN1). Among EfficientNets from B0 to B3, the accuracy and the memory requirement seem to be in a linear relationship. Considering the ratio of the character accuracy to the consumed memory size, four versions of EfficientNets are almost the same, even if the lighter versions are slightly higher.

Ⅴ. Concluding Remark

We implemented a commercial-grade prototype of the low-cost industrial OCR system using an end-to-end deep learning approach. We used the cheapest embedded developer kit with GPU support for running deep neural networks. To achieve a simple implementation for limited computing resources and high accuracy for industrial use, we used the latest, end-to-end deep learning network and then optimized the network for our purpose. Several convolutional networks were tested to find the best feature extraction networks for both text detection and recognition. Finally, EffinientNet-B3 has been chosen as it shows the best recognition accuracy but consumes less memory than ResNet-50.

For further optimization, we tried to lighten the system by model conversion with TF-TRT integration and Tensorflow C++ API, respectively, but the results are not so satisfactory. For noticeable weight lightening, re-programming the whole network may be required to make it consistent with Tensorflow Lite or TensorRT.

References

[1] M. Jaderberg, K. Simonyan, A. Vedaldi, A. Zisserman, “Synthetic Data and Artificial Neural Networks for Natural Scene Text Recognition,” pp. 1-10, 2014.

[2] A. Poznanski, L. Wolf, “CNN-N-gram for

Handwriting Word Recognition," Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 2305-2314, 2016.

[3] Shangbang Long, Xin He, Cong Yao, "Scene Text Detection and Recognition: The Deep Learning Era," pp. 1-20, 2018.

[4] M. Jaderberg, K. Simonyan, A. Vedaldi, A. Zisserman, "Reading Text in the Wild with Convolutional Neural Networks," International Journal of Computer Vision, Vol. 116, No. 1, pp. 1-20, 2016.

[5] M. Liao, B. Shi, X. Bai, X. Wang, W. Liu, "Textboxes: A Fast Text Detector with a Single Deep Neural Network," Proceedings of Advancement of Artificial Intelligence, pp. 4161 - 4167, 2017.

[6] Z. Tian, W. Huang, T. He, P. He, Y. Qiao, "Detecting Text in Natural Image with Connectionist Text Proposal Network," Proceedings of European Conference on Computer Vision, pp. 56 - 72, 2016.

[7] X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He, J. Liang, "East: An Efficient and Accurate Scene Text Detector," Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 5551-5560, 2017.

[8] Christian Bartz, Haojin Yang, Christoph Meinel, "STN-OCR: A Single Neural Network for Text Detection and Text Recognition," pp. 1-9, 2017.

[9] M. Busta, L. Neumann, J. Matas, "Deep Textspotter: An End-to-end Trainable Scene Text Localization and Recognition Framework," Proceedings of IEEE International Conference on Computer Vision, pp. 2204-2212, 2017.

[10] T. He, Z. Tian, W. Huang, C. Shen, Y. Qiao, C.Sun, "An End-to-end Textspotter with Explicit Alignment and Attention," Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 5020 - 5029, 2018.

[11] X. Liu, D. Liang, S. Yan, D. Chen, Y. Qiao, J. Yan, "Fots: Fast Oriented Text Spotting with a Unified Network," Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 5676 - 5685, 2018.

[12] Wikipedia, Comparison of Deep-learning Software, Avaliable on : https://en.wikipedia.org/wiki/ Comparison_of_deep-learning_software

[13] M. Liao, B. Shi, X. Bai, "TextBoxes++: A Single-shot Oriented Scene Text Detector," Journal of IEEE Transactions on Image Process, Vol. 27, No. 8, pp. 3676-3690, 2018.

[14] B. Shi, X. Bai, C. Yao, "An End-to-end Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition," Journal of IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 39, No. 11, pp. 2298-2304, 2017.

[15] K. Simonyanm, A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," Proceedings of International Conference on Learning Representations, pp. 7-9, 2015.

[16] K. He, X. Zhang, S. Ren, J. Sun, "Deep Residual Learning for Image Recognition," Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 770-778, 2016.

[17] K. Kim, S. Hong, B. Roh, Y. Cheon, M. Park, "PVANET: Deep but Lightweight Neural Networks for Real-time Object Detection," pp. 1-7, 2016.

[18] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," pp. 1-9, 2017.

[19] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 4510-4520, 2018.

[20] A. Howard, M. Sandler, G. Chu, L. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, V. Q. Le, H. Adam, "Searching for MobileNetV3," Proceedings of IEEE International Conference on Computer Vision, pp. 1314-1324, 2019.

[21] M. Tan, Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolution Neural Networks," pp. 1-10, 2019.
[22] Sandler, M. Howard, A. Zhu, M. Zhmoginov, L. Chen, "Mobilenetv2: Inverted Residuals and Linear Bottlenecks," Proceedings of IEEE Conference on Computer Vision and Pattern Recognition , pp. 4510-4520, 2018.
[23] Tan, M. Chen, B. Pang, R. Vasudevan, V. Sandler, M. Howard, Q. V. Le, "MnasNet: Platform-aware Neural Architecture Search for Mobile," Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 2820-2828, 2019.
[24] Hu, J. Shen, G. Sun, "Squeeze-and-excitation Networks," Proceedings of IEEE Conferenc on Computer Vision and Pattern Recognition, pp. 7132-7141, 2018.
[25] xTensor, Multi-dimensional Arrays with Broadcasting and Lazy Computing, Avaliable on : https://xtensor.readthedocs. io/
[26] Tobias Knopp, NumCpp, Avaliable on : https://numcpp. readthedocs.io/

Bharat Subedi

He received a B.E. degree in Information Technology from Pokhara University, Nepal in 2014. He is currently an M.S. candidate in the Depart- ment of Computer Engi- neering at Kumoh National Institute of Technology. His research interests include computer vision tasks with various deep learning technologies.
Email: bharat@kumoh.ac.kr

Jahongir Yunusov

He received a B.E. degree in Information and Information Technology from Tashkent University of Information Techno- logies, Uzbekistan in 2016. He is currently an M.S. candidate in the Department of Computer Engineering at Kumoh National Institute of Technology. His research interests include various deep learning technologies.
Email: jahongir7174@kumoh.ac.kr

Abdulaziz Gaybulayev

He received an M.E. degree in Computer Engineering from Tashkent University of Information Technologies, Uzbekistan in 2015. He is currently a Ph.D. candidate in the Department of Computer Engineering at Kumoh National Institute of Technology. His research interests include various deep learning technologies.
Email: g.abdulaziz@kumoh.ac.kr

Tae-Hyong Kim (김 태 형)

He received a Ph.D. degree in Electrical and Electronic Engineering from Yonsei University in 2001. He is currently a professor in the Department of Computer Engineering at Kumoh National Institute of Technology. His current research interests include deep learning, big data, and IoT technologies.
Email: taehyong@kumoh.ac.kr