World Wide Web (2021) 24:1299–1319
https://doi.org/10.1007/s11280-020-00778-y

# OpenCL-Darknet: implementation and optimization of OpenCL-based deep learning object detection framework

Yongbon Koo, et al. *[full author details at the end of the article]*

Received: 11 May 2019 / Revised: 19 October 2019 / Accepted: 2 January 2020 /
Published online: 7 February 2020

**Abstract**
Object detection is a technology that deals with recognizing classes of objects and their location. It is used in many different areas, such as in face-detecting systems [16, 34, 37], surveillance tools [9], human-machine interfaces [17], and self-driving cars [18, 23, 25, 26, 30]. These days, deep learning object detection approaches have achieved significantly better performance than the classical feature-based algorithms. Darknet [31] is a deep learning object detection framework, which is well known for its fast speed and simple structure. Unfortunately, Darknet can only work with Nvidia CUDA [6] for accelerating its deep learning calculations. For this reason, users have only limited options of selecting appropriate graphic cards. Open computing language (OpenCL) [35], an open standard for cross-platform, parallel programming of heterogeneous systems, is available for the general hardware accelerators. However, many deep learning frameworks including Darknet have no support for OpenCL.
In our previous paper, we presented OpenCL-Darknet [19], which transformed the CUDA-based Darknet into an open standard OpenCL backend. The original OpenCL-Darknet successfully showed its ability for the general graphics processing unit (GPU) hardware. However, it could not achieve competitive performance compared with the CUDA version, and it only supported a limited platform. In this study, we improved the performance of OpenCL-Darknet with several optimization techniques and added support for various architectures. We also evaluated OpenCL-Darknet not only in AMD R7 accelerated processing unit (APU) with OpenCL 2.0, but also in Nvidia GPU and ARM Mali embedded GPU with OpenCL 1.2 Profile. The evaluation using the standard object detection datasets showed that our advanced OpenCL-Darknet reduced the processing time by at most 50% on average for various deep learning object detection networks compared with our original implementation. We also showed that our OpenCL deep learning framework has competitiveness compared with the CUDA-based one.

**Keywords** deep learning · image processing · object detection · parallel programming · OpenCL

This article belongs to the Topical Collection: *Special Issue on Artificial Intelligence and Big Data Computing*
Guest Editors: Wookey Lee and Hiroyuki Kitagawa

## 1 Introduction

Humans can understand the world with their eyes and their brains, and computers can emulate human vision with cameras and computing components. The research about this emulation is computer vision. One of the major fields of computer vision is an object detection, which is a technology that deals with recognizing classes of objects and their location. With the rise of face detection [16, 34, 37], surveillance [9], human-machine interfaces [17], and self-driving cars [18, 23, 25, 26, 30], demands for fast and accurate object detection systems have also increased. Classic object detection systems use the scheme of feature-based methods, such as Haar feature extractions [29, 37] or histogram of oriented gradients (HOG) and linear support vector machine (SVM) algorithms [8].

These days, deep learning is applied to the object detection and has archived significant precision improvement. There are various deep learning frameworks that execute deep learning algorithms. Darknet [31] is one of the most widely used frameworks for object detection. It is fast, does not require installation, and supports CPU and graphics processing unit (GPU) computation. Unfortunately, like many other frameworks, Darknet only supports Nvidia CUDA [6], which is only available on the Nvidia graphic devices for accelerating its deep learning calculations. For this reason, users can have only limited options of selecting appropriate graphic cards, and they are forced to having restricted system configurations.

In our previous paper, we presented OpenCL-Darknet [19], which transformed a CUDA-based Darknet into an open standard open computing language (OpenCL) backend to resolve this situation. OpenCL [35] is a low-level open standard application programming interface (API) for cross-platform, parallel programming of heterogeneous systems. It is available not only for CPUs and GPUs, but also for digital signal processors (DSP), field-programmable gate arrays (FPGA), and other hardware accelerators. Our goal was to implement a deep learning-based object detection framework available for general hardware. The original OpenCL-Darknet successfully showed its ability with AMD GPU hardware. However, it could not achieve competitive performance compared to the CUDA version. Furthermore, the original OpenCL-Darknet was able to support only an AMD-based platform and had little generality.

In this study, we improved the performance of OpenCL-Darknet with several optimization techniques and added support for various architectures. We will explain the details of OpenCL-Darknet's implementation including the unit test methods and optimization techniques, such as adaptive kernel selection, kernel merging, and asynchronous kernel execution. We evaluated OpenCL-Darknet optimization in AMD R7 accelerated processing unit (APU) environment and added supports for ARM Mali embedded boards and Nvidia graphic cards. Finally, we compared the performance of OpenCL-Darknet with the CUDA-based Darknet. Our optimizations allowed us to achieve a higher performance than the previous one. Evaluations showed that our advanced OpenCL-Darknet reduced the processing time by at most 50% on average for various deep learning object detection networks compared with our original implementation. Currently, machines with AMD or ARM had no methods to execute CUDA codes. For this reason, those machines had to rely on CPU calculations only. OpenCL-Darknet can help developers to work with the general accelerators.

The rest of this paper is structured as follows. Section 2 introduces the backgrounds of deep learning object detection, Darknet framework, and OpenCL. Section 3 explains about our porting and optimization techniques, and Section 4 shows the evaluation environments and results. Related work is described in Section 5. Finally, we will talk about conclusion and our future work in Section 6.

## 2 Backgrounds

Deep learning [12] is one of the machine learning methods based on data representations, as opposed to task-specific algorithms. Deep learning learns the filters from data, which were hand-crafted by human prior knowledge, and this method leads to significantly better performance. Among various deep learning algorithms, a convolutional neural network (CNN) has been proved as a successful image analysis method. CNN consists of a sequence of various layers, such as convolutional layers, pooling layers, normalization layers, and fully connected layers.

You only look once (YOLO) [32] is a state-of-the-art, real-time deep learning object detection algorithm based on CNN. Prior detection algorithms repurpose classifiers or localizers to perform detection. They apply the model to an image at multiple locations and scales. Then high scoring regions of the image are considered detections. YOLO apply a single neural network to the full image. This network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities. It performs predictions with a single network evaluation unlike systems like R-CNN [33] which require thousand networks for a single image. This makes YOLO extremely fast.

Darknet is an open source neural network framework written in C and CUDA. It was developed for YOLO deep learning algorithm variants. It offers building blocks for designing, training and validating deep neural networks, and is well known for its fast processing speed and simple architecture.

Finally, OpenCL helps developers to incorporate advanced numerical and data analytics features, perform cutting-edge image and media processing. OpenCL consists of one host and one or more OpenCL devices. Each OpenCL device is composed of one or more compute units. Memory structure is also divided into host memory and device memory. The basic idea of OpenCL is data parallelism, which replaces loops with many kernels executing same instructions only on the part of large data. The OpenCL program runs with the following sequences: 1) setup the environment for the OpenCL program, 2) define the platform and build programs and kernels, 3) setup memory objects, 4) define the kernel arguments, and 5) transfer memory objects and execute kernels.

## 3 Porting and optimization strategies

### 3.1 Algorithm overview

General deep learning algorithms are processed with the sequence as described in Figure 1. Initialization loads a network structure and its weight on the main memory. Then, it establishes memory structures for the network calculations. In addition, OpenCL devices are initialized, and the loaded contents are transferred into the device memory from the host memory. The initialization is just a one-time step required only once at the beginning to deal with the multiple images or the image sequences.

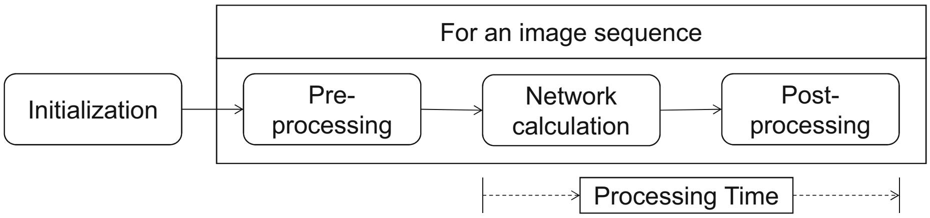


**Figure 1** General deep learning process

The deep learning process for every single image is composed of three steps: pre-processing, network calculation, and post-processing. First, pre-processing loads an image from a storage device and resizes it in accordance with the network setting. Its performance depends on the I/O throughput and CPU speed of the machine, not on the accelerator's efficiency. Second, network calculation performs layer-by-layer processing with an input image and weight matrices. The most important thing we consider is that the output of the previous layer can be treated as the input of the next layer. Such layer-by-layer iterations are the essential parts of the deep learning network. Finally, post-processing calculates the probabilities and locations of candidate classes. Non maximum suppression (NMS) is utilized to ignore redundant, overlapping bounding boxes, i.e., it merges all detections that belong to the same object [24].

The network calculation looks like a sequential processing, but the calculation of each layer can be accelerated by parallel hardware due to its own data parallel characteristic. For example, a convolutional layer can apply filters to an input image using general matrix multiplication (GEMM). Each row and column are independent and undergo the same operations: multiplication and addition. OpenCL devices can increase performance using this characteristic.

### 3.2 Unit test framework

As we described in the previous section, the Darknet framework was originally written in C and CUDA. CUDA and OpenCL have different hardware abstractions, memory management schemes, and data transfer mechanisms. Therefore, we need a careful approach to developing the OpenCL deep learning framework. We targeted layer-wise porting using the unit test framework. The unit test [3] is a software testing method in which the smallest testable parts of an application, called units, are individually and independently scrutinized for efficient operation. The purpose of the unit test is to validate that each unit of the software performs as designed. In our OpenCL-Darknet, a unit is an OpenCL kernel implementation. In detail, each layer in Darknet has both CPU and GPU implementation, and the user can select one under compiler options. Almost every deep learning layers in Darknet are deterministic layers, which means that different implementations of layers should make the same results on the same inputs. We collected representative data of every OpenCL kernel's input and output by running CPU implementations with several images. Then, we built a test framework to verify the OpenCL implementation of each kernel with this data. This procedure not only proved the right results for the implementations of various layers, but also provided a way to measure and compare the performance of each layer's implementations. Figure 2 shows the structure of our unit test for OpenCL-Darknet. Basic arithmetic functions were also ported to OpenCL with the same way. They included filling, scaling, and normalizing matrices. We needed to transform simple layers, such as activation to OpenCL to reduce unnecessary data transfers between a host and devices. Not only deep learning layers but also post-processing functions such as NMS functions needed to be implemented for the device.

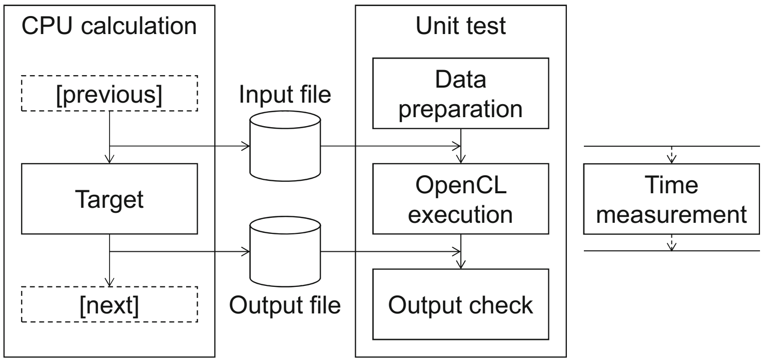


**Figure 2** Unit test procedure for OpenCL-Darknet

### 3.3 Convolutional layer calculation

A convolutional layer is the most important one for CNN because it is proved as the most time-consuming part for deep learning networks. There are many approaches for the efficient convolution. Among them, Darknet utilizes the GEMM method, and depends on Basic linear algebra subprograms (BLAS) libraries for the GEMM implementation.

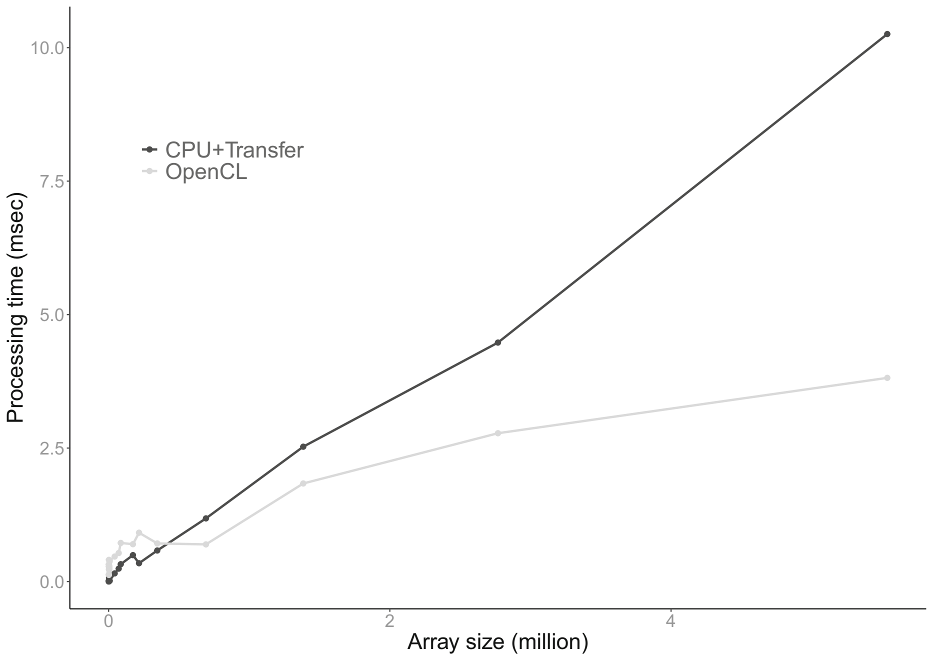
In OpenCL-Darknet, we utilized a GPU-accelerated BLAS library, clBLAS [4] and CLBlast [27]. clBLAS was developed by AMD and is well optimized for AMD graphic hardware. CLBlast was an open source BLAS library that designed to leverage the full performance potential of a wide variety of OpenCL devices from different vendors. We also used clRNG [5] to generate a random stream array in parallel on an OpenCL device.

### 3.4 Adaptive kernel selection

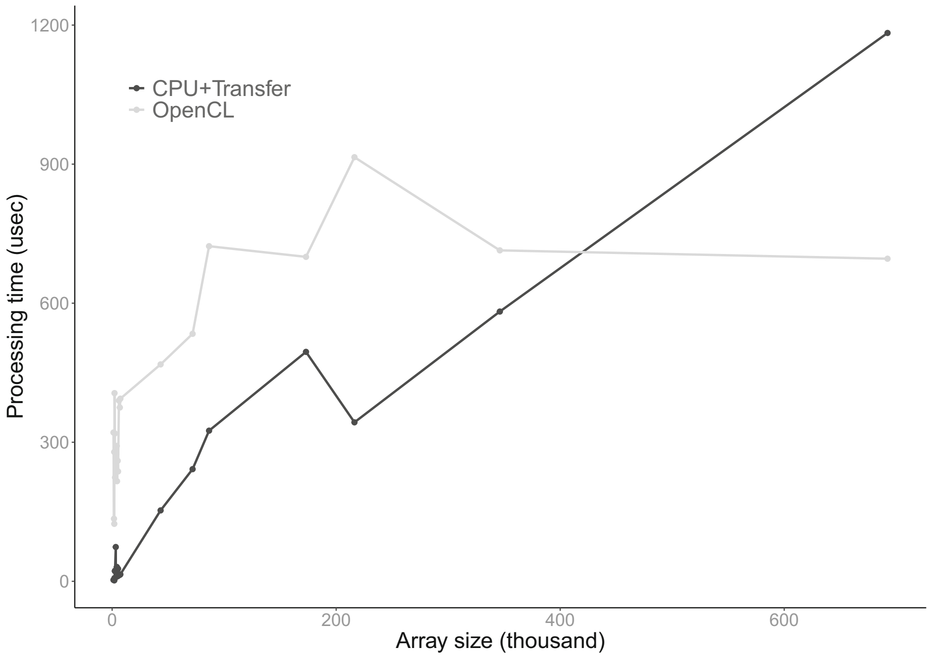
As described above, OpenCL-Darknet carried out all operations on the device side to reduce the data transfer overhead. However, some kernels did not require device memory input during calculations. They also involved quite a small data array and simple arithmetic. In this case, it was efficient to perform calculations in the host side and transfer the results to the device. We were able to determine such cases using the unit test framework. For example, Figure 3(a) shows that FILL kernel's CPU-side calculations with memory transfer are faster than GPU-side calculations when they deal with memory smaller than a given size. Moreover, we can see in Figure 3(b) that such small-sized memory fillings were much more frequent than large-sized memory handlings. Therefore, a strategy of making the calculations on the host and delivering the result to the device could be a better solution in this case.

### 3.5 Merging same-level kernels

There exists a sequence of kernels that can work on the same memory address with difference operations. Pseudo codes below show three kernels,`normalize()`,`add_bias()`, and`scale_bias()`, which are dealing the same memory objects. So, they can be merged



(a) FILL kernel performance comparison according to the input data array size



(b) FILL kernel performance for small size input data
(Each point means kernel calls for one image processing)

**Figure 3** FILL kernel performance comparison between CPU and GPU

into one kernel;`normalize_scale_add()`. Such kernel merging is able reduce the OpenCL execution overhead.

```
__kernel void normalize()
{
   output[index] = (output[index] - mean[filter]) /
(sqrt(variance[filter] + .00001f));
}

__kernel void add_bias()
{
   if (offset < size)
       output[(batch*n  +  filter)*size  +  offset]  +=
biases[filter];
}

__kernel void scale_bias()
{
   if (offset < size)
       output[(batch*n  +  filter)*size  +  offset]  *=
scales[filter];
}

/* Merged Kernel */
__kernel void normalize_scale_add()
{
   output[index] = (output[index] - mean[filter]) /
(sqrt(variance[filter] + .00001f));

   if (offset < size)
       output[index] = output[index] * scales[filter]
+ biases[filter];
}
```

### 3.6 Asynchronous kernel execution

When a host program executes OpenCL kernels, there are two different modes for waiting for the completion of the kernel execution. In a synchronous mode, a host code stops and waits for the device codes to be finished. In many cases, such standby operations are essential because of the dependency of host and device programs. In an asynchronous mode, a host code does not wait for the completion of the device execution. OpenCL-Darknet could improve its performance with an asynchronous kernel execution. Because OpenCL-Darknet performed all the operations, even the simple arithmetic, in the device side, we did not need host-device memory data transfer except for the first and last layers. In addition, deep learning layers have their own dependency for the sequence of layers. These situations allow many kernels to be executed along with host codes and do not destroy data. Such an optimization technique promoted the parallel execution of the kernel and host code, so improves the performance. In the case of asynchronous OpenCL kernel execution mode, the time measurement of the host side codes did not show precisely accurate results because the host did not wait for the completion of the device's kernel execution. For this reason, the layer-by-layer processing time should be measured in the synchronous mode.

### 3.7 Removing unnecessary operations

A CUDA-based Darknet can perform inference and training operations. However, in this study, OpenCL-Darknet only supported calculations for a generic inference situation. Almost all the embedded devices only run inference processes. Large-scale clusters were responsible for the training. Training procedures perform backward calculations after forward calculations. It means that one forward implementation is not only for the inference but also for training. And it also implies a forward operation contains some codes that are unnecessary during inference procedures. We carefully investigated every operation needed for the inference. As a result, we were able to remove unnecessary memory operations from OpenCL-Darknet and improve its performance.

## 4 Evaluations

### 4.1 Test environment

We evaluated OpenCL-Darknet in various environments and compared the performances with the previous OpenCL and CUDA-based implementations. AMD R7 GPU is integrated within an AMD Embedded RX-421BD APU chip. In an APU environment, CPU cores and GPU cores are connected internally and share the main memory. AMD R7 runs at 800 MHz and can perform half-precision floating point operations at the rate of 819 G floating point operations per second (GFLOPS). Odroid-XU4 is an ARM-based computing device, which contains a Mali-T628 MP6 graphic chip and an OpenCL 1.2 profile. Mali-T628 MP6 has a 533 MHz clock speed and 102.4 GFLOPS performance theoretically. Nvidia GTX1050Ti is a discrete GPU card that can be attached on the AMD RX-421BD board via a PCI-Express interface, and it offers a much faster performance than an APU or an ARM Mali device. Its 768 CUDA cores runs at 1366Mhz and has the processing speed of 3962 GFLOPS. We utilized GTX1050Ti to compare the performance of CUDA-based systems and OpenCL-Darknet. For accurate measurements, we turned off an integrated GPU when we were using a discrete GPU. More detailed descriptions of each hardware and software configuration are showed in Tables 1, 2, and 3.

The evaluations were performed with publicly available PASCAL visual object classes challenge (VOC) 2007 [10], Microsoft common objects in context (COCO) 2017 [22] data sets, and KITTI vision benchmark suite [11]. PASCAL VOC 2007 contains 4952 color images, each image has about 190,000 pixels. It is also composed of 20 classes, including person, car, and bicycle. Microsoft COCO 2017 contains 40,670 color images and 80 object categories.

**Table 1** Configuration for AMD R7 APU

| | |
|---|---|
| CPU | AMD Embedded RX-421BD 4 cores / 2100 MHz / 28 nm |
| GPU | R7 / 8 compute units 512 shading units / 800 MHz / 819 GFLOPS(FP16) |
| Memory | 16GB DDR4 |
| Storage | 256GB SSD / SATA III interface |
| Operating Systems | Ubuntu 14.04 / Kernel 3.16.0–77 |
| OpenCL | 2.0 Profile |
| clBLAS | 2.12 |
| CLBlast | 1.3.0 |

**Table 2** Configuration for ARM Mali

| | |
|---|---|
| CPU | Samsung Exynos5422 CortexTM-A15 2Ghz and CortexTM-A7 Octa core CPUs |
| GPU | Mali-T628 MP6 / 533 MHz / 102.4GFLOPS(FP16) |
| Memory | 2GB LPDDR3 |
| Storage | 64GB microSD |
| Operating Systems | Ubuntu 16.04 / Kernel 4.14.28 |
| OpenCL | 1.2 Profile |
| CLBlast | 1.3.0 |

The size of each image is about 300,000. KITTI vision benchmark suite contains 7481 images for validation specialized in the driving situation. They are the standard data sets for object classification and detection evaluations. Their main purposes are to recognize objects in realistic images from various object classes.

The detection models we tested were YOLOv2 and TinyYOLO which were described in Section II. YOLOv2 contains 32 layers, among which 23 layers are convolutional layers. TinyYOLO is a smaller one with 16 layers, containing 9 convolutional layers. Figure 4 shows examples of detection results using two YOLO algorithms with OpenCL-Darknet. The weights of YOLOv2 and TinyYOLO models for VOC were trained with VOC 2007 and VOC 2012 training data sets. And these models had a mean average precision (mAP) on the VOC 2007 test data of 78.6% and 57.1% respectively. Weights for COCO were trained with COCO trainval data and the trained model had a mAP on the COCO test-dev data of 48.1%. TinyYOLO on COCO does not reveal its accuracy, but we expect it has lower precision than YOLOv2. We trained the weights for KITTI dataset with a training set by ourselves. In this paper, using the unit test implementation, we demonstrated that our OpenCL-Darknet did not change the calculation results of any layer. Therefore, the precision of the detection algorithms did remain the same.

## 4.2 Layer-by-layer performance analysis

As described in Section III, a deep learning object detection executes three steps in every iteration: pre-processing, network calculation, and post-processing. Figure 5(a) shows the processing time consumed by each step. Because the pre-processing contains only file system operations and CPU operations, its performance is not determined by OpenCL devices, but by dataset configurations. Network calculation is the essential and the most time-consuming part of deep learning algorithms; it consumes almost 70%–90% of whole processing time. The performance is varied considerably with network and dataset configurations. To be specific, the influential elements are the depth of each network,

**Table 3** Configuration for NVIDIA GTX1050TI

| | |
|---|---|
| CPU | AMD Embedded RX-421BD 4 cores / 2100 MHz / 28 nm |
| GPU | Nvidia GTX1050Ti / 768 CUDA cores 1366 MHz / 3962 GFLOPS(FP16) / 4GB GDDR5 |
| Memory | 16GB DDR4 |
| Storage | 256GB SSD / SATA III interface |
| Operating Systems | Ubuntu 14.04 / Kernel 4.4.0–97 |
| CUDA | 8.0 |
| OpenCL | 1.2 Profile |
| CLBlast | 1.3.0 |

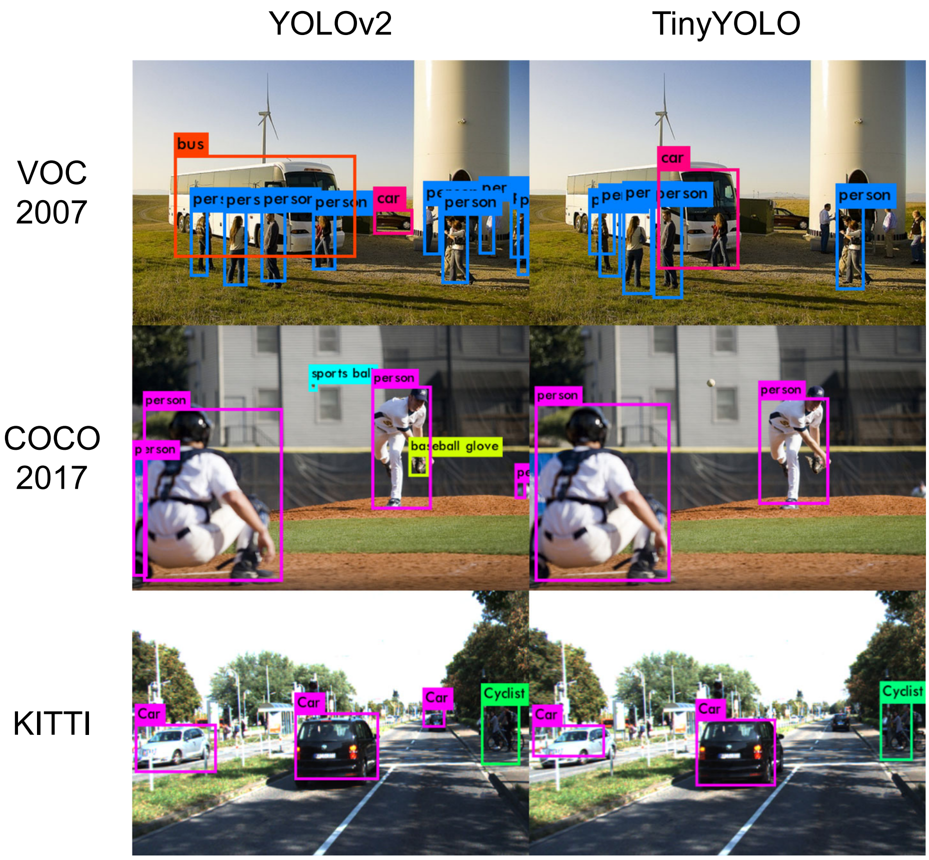


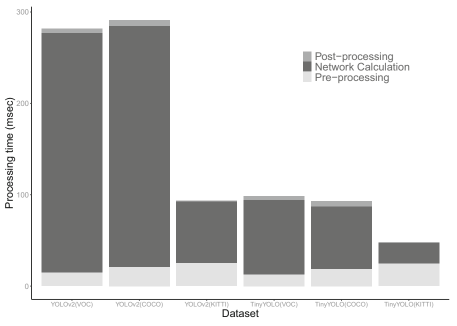
**Figure 4** Detection results using YOLOv2 and TinyYOLO for datasets

especially for the number of convolutional layers, and the size of input images. Post-processing involves NMS and drawing the bounding boxes. This is a part of object detection algorithms that can be accelerated by OpenCL. However, post-processing is not a big part of deep learning processing.

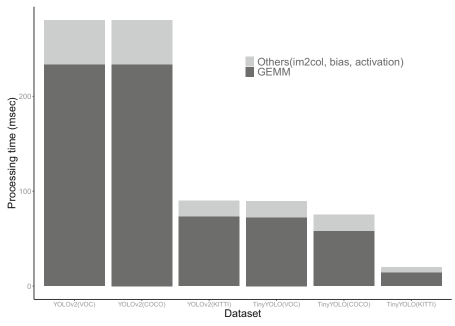
We analyzed the layer-by-layer processing time of network calculation step. As you can see in Figure 5(b), processing time is dominated by convolutional layers, and other layers seem no impact on performance. And Figure 5(c) shows that the major time consumers in the convolutional layers are the GEMM calculations.

CPU and GPU load in Figure 5(d) presents that there is a correlation between processing time and hardware utilization. In general, high utilization means efficiency. However, in Darknet's case, CPU and GPU utilization depends on both datasets and algorithms. We think dataset configurations are more dominant than algorithms because the size of the images is very important parameter for processing images. It can be an important hint when a system manager considers the consolidation of the system.

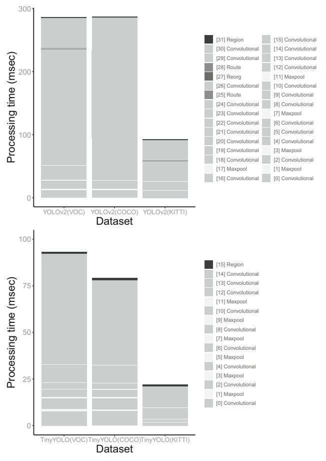
Next, we investigated the elements that could affect the performance of GEMM. Figure 6 shows that time consumed by GEMM is determined by their $K$ parameters. The $K$ parameter is the number of columns and rows of each input matrix and is proportional to the square of the



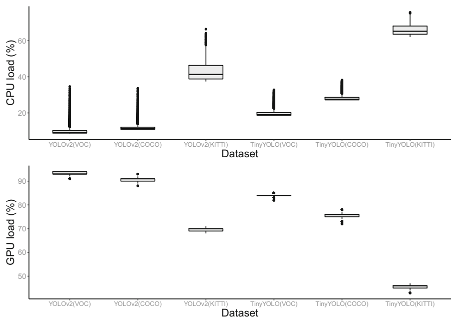
(a) Per-image deep learning step processing time



(c) Time consumption of operations in convolutional layers



(b) Per-image layer processing time



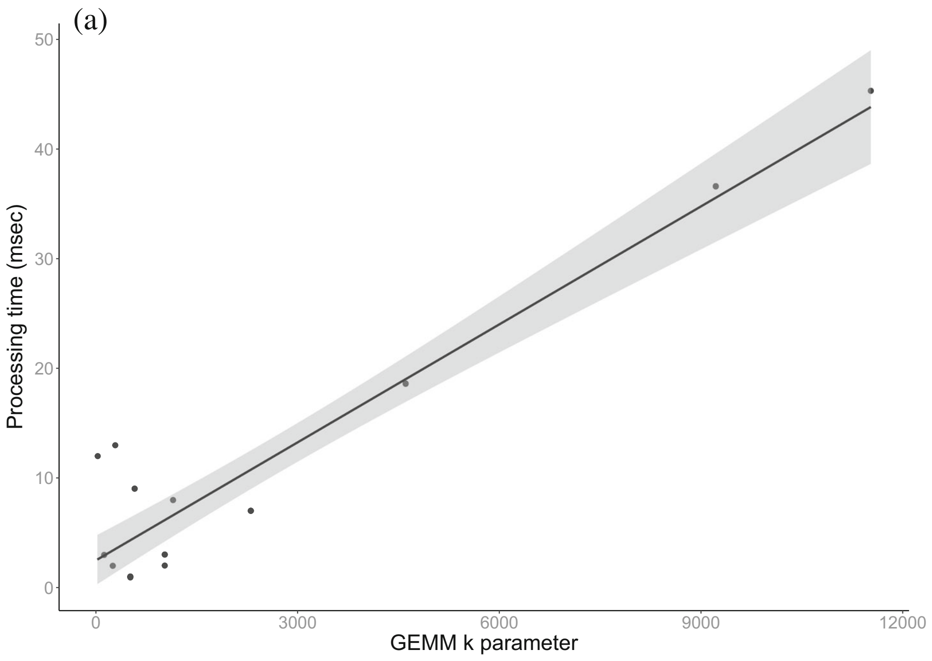
(d) CPU and GPU load

**Figure 5** Deep learning object detection processing time for AMD R7 APU

size of a convolutional layer. Therefore, we concluded that high-speed GEMM algorithms or convolution methods for the large-sized input matrices are essential to increase the overall performance. Currently, convolution methods based on the fast Fourier transform (FFT) are known as suitable algorithms for large-sized data arrays. This issue will remain as a future investigation.

## 4.3 Performance comparison with the original OpenCL-Darknet

We successfully adopted several optimization techniques and improved the original OpenCL-Darknet. Figure 7(a) presents the results tested on the AMD R7 APU, on which previous one is available. Compared with our original version, newly optimized OpenCL-Darknet achieved 32%–44% reduced average processing time for YOLOv2, and 16%–47% for TinyYOLO. With AMD R7 GPU, VOC and COCO images needed about 280 ms for YOLOv2, and 100 ms for TinyYOLO, including pre-processing and processing time. In other words, we were able to process 3–4 images per second with YOLOv2, and 10 images per second with TinyYOLO for these datasets without any external graphic card.



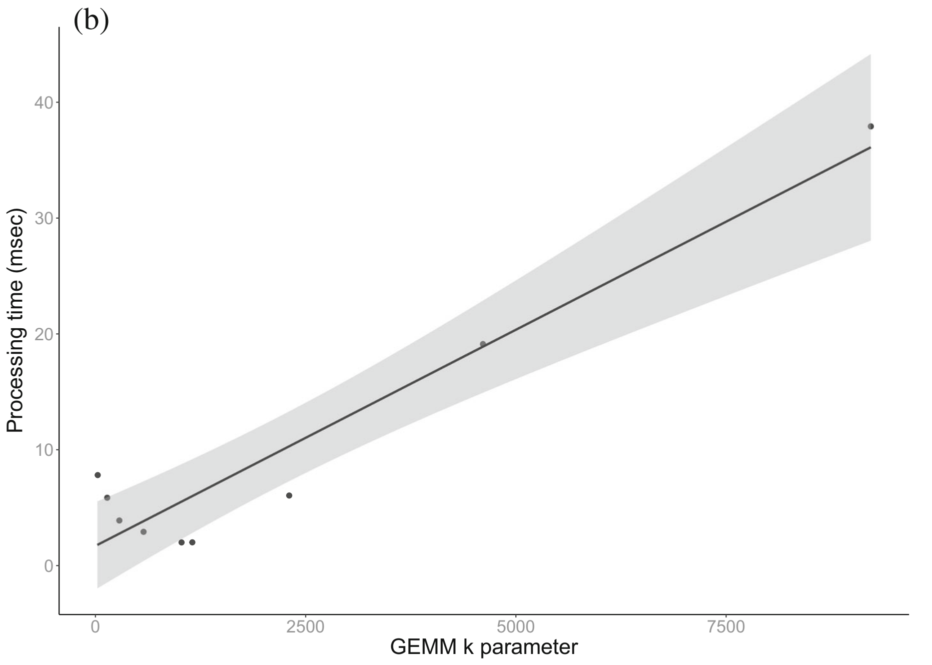
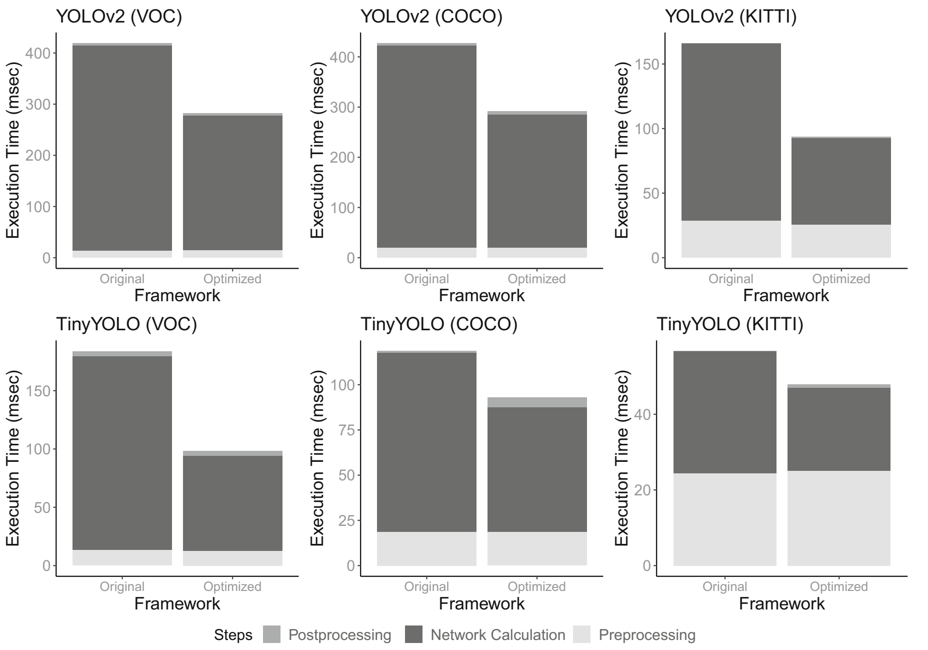
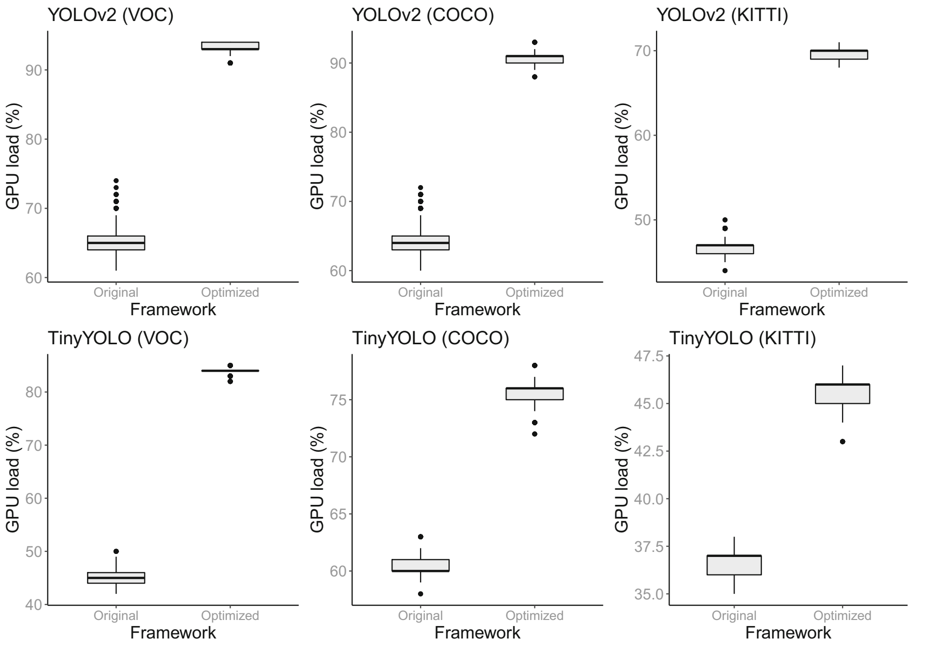


**Figure 6** clBLAS GEMM performance according to matrix size (AMD R7 APU)



(a) Deep learning step processing time comparison with the original OpenCL-Darknet



(b) GPU load comparison with the original OpenCL-Darknet

**Figure 7**

Figure 7(b) indicates that the optimizations increase GPU utilization and lead to the performance improvement. As stated in the previous section, high GPU utilization does not always lead to the fast processing every time. However, in this case, we can estimate that our optimization techniques utilize hardware resources efficiently.

### 4.4 Performance comparison between BLAS libraries

As described in Section III, we utilized clBLAS [4] and CLBlast [27] libraries for the GEMM calculation of convolutional layers. To compare the performance of two BLAS libraries, we evaluated OpenCL-Darknet with clBLAS and CLBlast in an AMD R7 GPU board. As you can see in Figure 8(a), in AMD R7 GPU environment, clBLAS is superior to CLBlast in almost every situation. GPU load in Figure 8(b) also proves clBLAS is efficient than its open source counterpart. clBLAS is implemented by a hardware supplier, AMD, so it is optimized to AMD environment. However, CLBlast is a much extensible library that works on many different hardware.

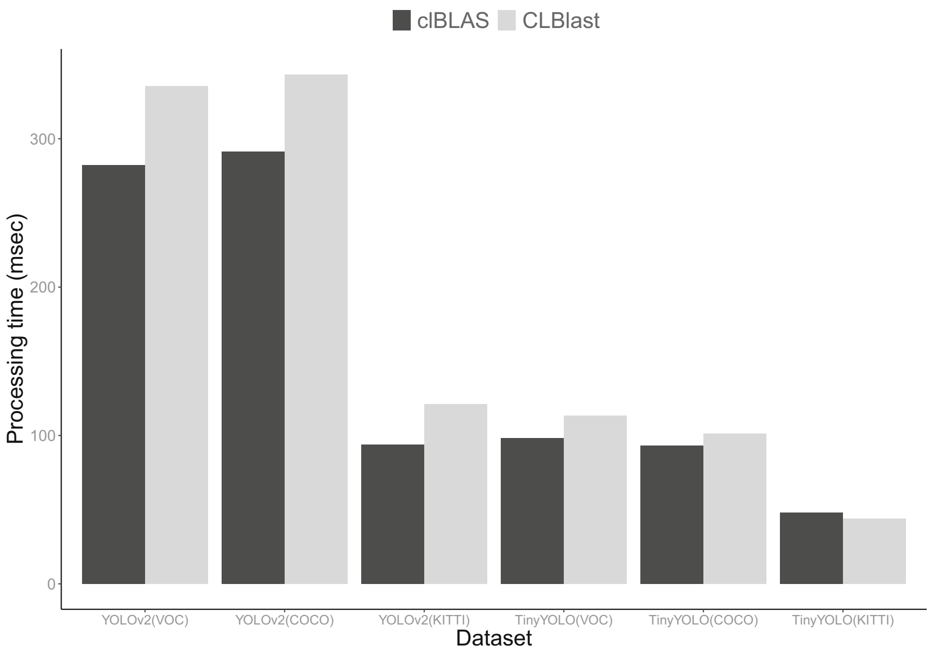
### 4.5 Performance comparison with CPU-based deep learning

Currently, ARM and AMD machines do not have decent deep learning object detection frameworks. For this reason, developers must rely on CPU-based frameworks or old-style image processing algorithms. Some recent devices are following a trend towards additional chips dedicated to neural net calculations. However, such a trend is not allowed for everyone. Cheap and low-powered boards cannot adapt it because of their high cost in hardware and software development. OpenCL-Darknet can be a good candidate to solve such problems. Figure 9(a) and (b) show that OpenCL-Darknet improves the performance compared with CPU-based Darknet. We got 10x–30x performance improvement for AMD R7 and 3x–13x improvement for ARM Mali. Furthermore, CPU-based deep learning needs at most 49 seconds to process an image. That means it cannot be tolerated by real applications. OpenCL-darknet shows that it is suitable for practical real-time image processing. Deep learning is a general-purpose algorithm, that is, developers will have great advantages in easy customization for the requirements of their programs.

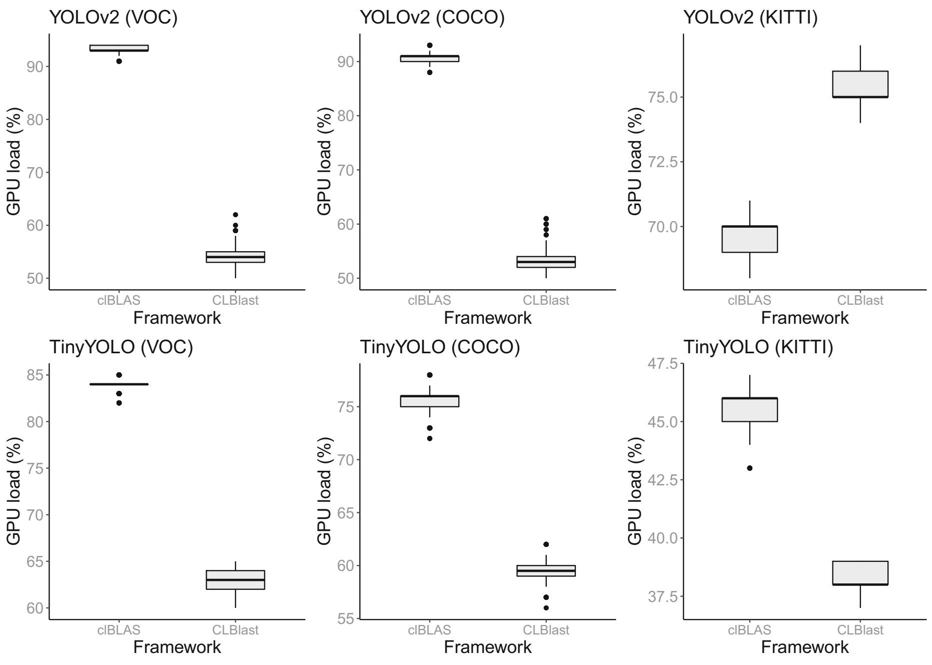
### 4.6 Performance comparison with CUDA version

We compared the performance of our OpenCL-Darknet with the CUDA-based Darknet. We ported OpenCL-Darknet to Nvidia GTX-1050Ti with CLBlast library. As you can see in Figure 10(a), evaluation shows that OpenCL-Darknet has 4%–28% less performance than the CUDA version. To be more specific, Figure 10(b) shows that OpenCL-Darknet has the equivalent performance compared with CUDA-based Darknet except GEMM operations. As we described in section III, GEMM operations are solely relied on BLAS libraries; CLBlast for OpenCL-Darknet, and cuBLAS [7] for CUDA-based Darknet. Author of CLBlast also admitted that they were not able to match cuBLAS due to lack of assembly-level optimizations [28].

GPU load in Figure 10(c) presents that CUDA-based Darknet consumes less GPU resource than OpenCL-Darknet. That implies GPU utilization is not a direct reason for less performance of the OpenCL BLAS libraries. Nugteren [28] stated that register pressure reduction and less

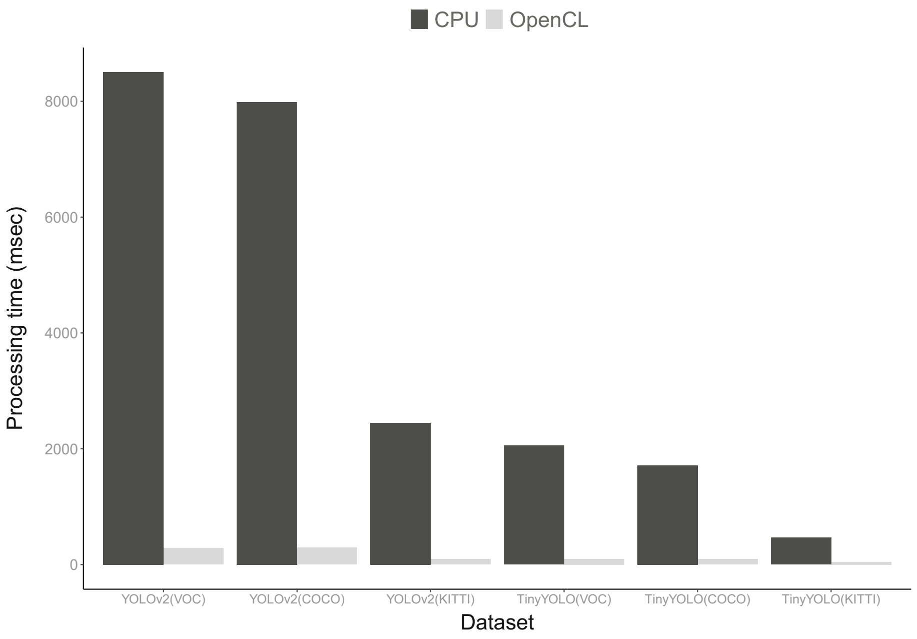


(a) Processing time comparison between BLAS libraries (AMD R7 APU)

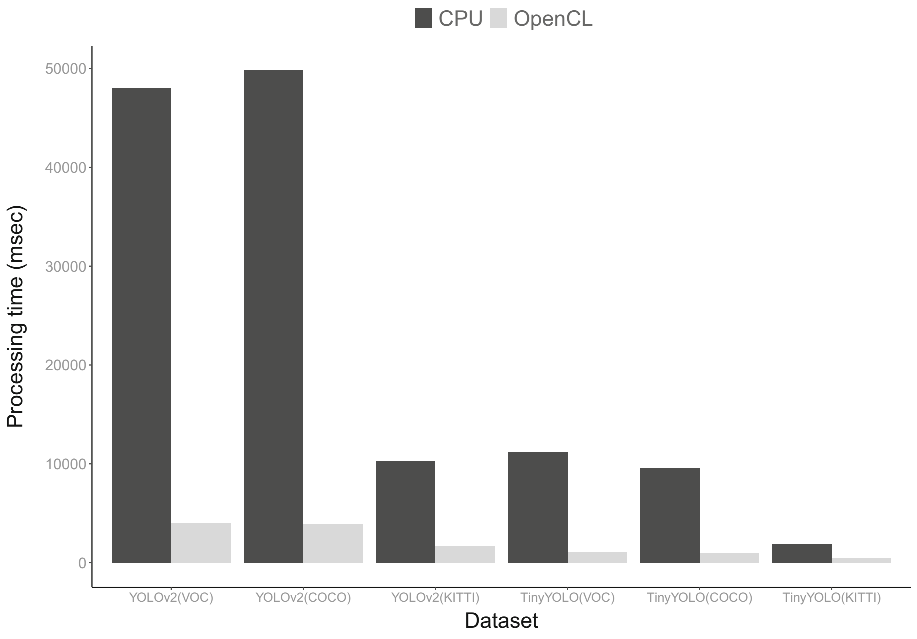


(b) GPU loadcomparison between BLAS libraries (AMD R7 APU)

**Figure 8** Comparison between BLAS libraries (AMD R7)

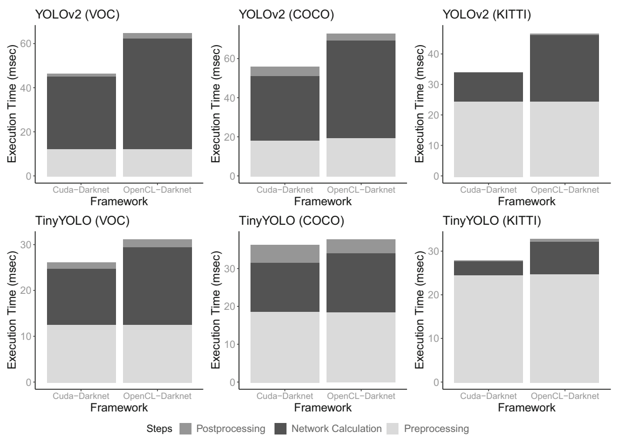


(a) Processing time comparison between CPU and OpenCL (AMD RX-421BD CPU vs. AMD R7 GPU)

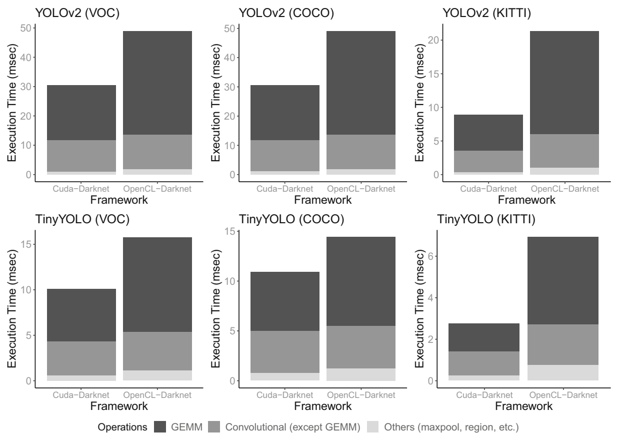


(b) Processing time comparison between CPU and OpenCL (ARM Exynos5422 CPU vs. ARM Mali-T628 MP6)

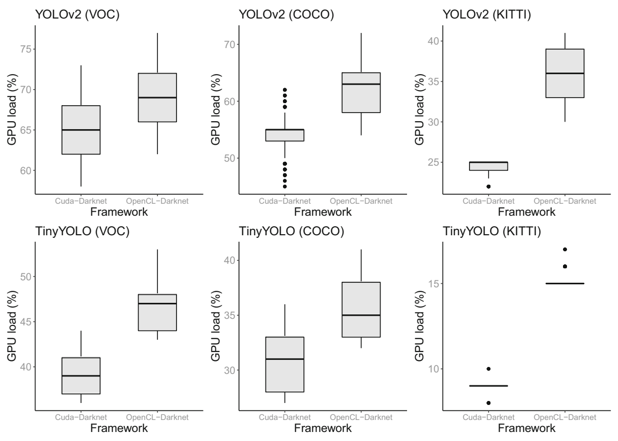
**Figure 9** Processing time comparison between CPU and OpenCL



(a) Deep learning step processing time comparison with CUDA-based Darknet



(b) Layer processing time comparison with CUDA-based Darknet



(c) GPU load comparison with CUDA-based Darknet

**Figure 10** Comparison with CUDA-based Darknet

register bank conflicts are the main reason of performance differential between BLAS libraries. For this reason, we concluded that the BLAS libraries for OpenCL are not optimized compared with those for CUDA especially for in an aspect of register level memory efficiency. Because BLAS optimization is highly related with hardware suppliers, it will remain as future work.

### 4.7 Summary

OpenCL-Darknet enables deep learning acceleration by ARM and AMD GPU, which have used only slow CPU-based methods. Even though there is some overhead because of the BLAS optimization issue, we expect that it can be solved if hardware vendors release BLAS algorithms optimized for their GPU hardware. We reduced the processing time by almost half compared with the original version presented in our previous paper, and we achieved competitive performance compared with CUDA-version as removed almost all the overhead except for that from the BLAS algorithm.

## 5 Related work

Gu et al. [13] transformed the CUDA-based Caffe deep learning framework into OpenCL-based one, OpenCL Caffe. They described OpenCL porting strategies that guaranteed algorithm convergence and examined the performance bottlenecks. They also proposed three key optimization techniques, kernel caching to avoid OpenCL online compilation overheads, a batched manner data layout scheme to boost data parallelism and multiple command queues to boost task parallelism. However, OpenCL caffe did not cover object detection situation. Moreover, its optimization could be adapted to only batches of images, not sequences of images.

Liao et al. [21] designed a unified and efficient OpenCL platform model for multi−/many-core clusters. Their UHCL-Darknet presented OpenCL-based DNN framework for heterogeneous clusters. However, they their implementation is only restricted to high-end CPU and GPU environment.

There is some research that utilized object detection algorithm. Hendry et al. [15] figured out the problem of car license plate detection using a YOLO-darknet deep learning framework. They propose a sliding-window single class detector via tiny YOLO CNN classifiers. Barry et al. [2] presented a xYOLO model for humanoid soccer robots on low-end hardware.

OpenCL is a widely used parallel acceleration library, and many researchers have taken advantage of it. Badía et al. [1] accelerated a sound-source localization algorithm, Steered Response Power with Phase Transform (SRP-PHAT), using OpenCL. SPR-PHAT implementations require to handle a high number of signals coming from a microphone array and a huge search grid that influences the localization accuracy of the system. They insisted that OpenCL achieved close-to CUDA performance in GPU. Lee et al. [20] utilized GPU to improve performance of an algorithm for forming groups or clusters of similar objects in the dataset. Haseljic et al. [14] implemented image segmentation algorithms known as Simple Linear Iterative Clustering based on OpenCL. However, it is only limited multi-core CPU.

clSpMV [36] optimized the sparse matrix vector multiplication (SpMV) kernel, which is a key computation in linear algebra. They proposed a new sparse matrix format, the Cocktail Format, to take advantage of the strengths of many different sparse matrix formats. They found that clSpMV provided the best representations of the input sparse matrices on both Nvidia and AMD platforms. OpenCL-Darknet does not take any advantage with the sparse matrix operations. Moreover, an analysis on YOLO weight matrices show that weights are not sparse. There are many near-zero values in matrices, but not many zero values.

## 6 Conclusions and future work

In our previous paper, we presented OpenCL-Darknet [35]. OpenCL-Darknet transformed the CUDA-based Darknet – a deep learning-based object detection framework – into an open standard OpenCL backend. In this study, we improved the performance of OpenCL-Darknet with several optimization techniques and added support for various architectures. We also evaluated OpenCL-Darknet with various hardware platforms, including AMD R7 APU with OpenCL 2.0, Nvidia GPU and ARM Mali embedded GPU with OpenCL 1.2. The evaluation using the standard object detection datasets showed that our optimized OpenCL-Darknet reduced overhead existed in the previous OpenCL-Darknet and achieved competitive performance compared with CUDA-based one.

As future work, we plan to make a more cost-effective deep learning framework by facilitating FPGA to reduce the cost to run object detection systems. OpenCL supports various types of hardware and we can expand OpenCL-Darknet with the minimum efforts. We also have plan to reduce BLAS overhead using other kinds of convolution implementation. FFT-based convolution can be a good candidate to achieve it.

**Funding** This work was supported by the Technology Innovation Program (20000946, Development of artificial intelligent computing platform technology for service robots capable of real-time processing of large-capacity, high-performance sensor fusion processing and deep learning) funded by the Ministry of Trade, Industry & Energy (MOTIE, Korea).

## References

1. Badía, J., Belloch, J., Cobos, M., Igual, F., Quintana-Ortí, E.: Accelerating the SRP-PHAT algorithm on multi and many-core platforms using OpenCL. J. Supercomput. **75**(3), 1284–1297 (2019)
2. D. Barry, M. Shah, M. Keijsers, H. Khan, and B. Hopman, "xYOLO: A Model For Real-Time Object Detection In Humanoid Soccer On Low-End Hardware," *arXiv preprint*, 2019
3. Beck, K.: Test Driven Development: by Example. Addison-Wesley Longman Publishing Co., Inc., Boston, MA (2002)
4. *clBLAS*, Advanced Micro Devices, Inc., Phoenix (n.d.), [Online]. Available: https://github.com/clMathLibraries/clBLAS
5. *clRNG*, Advanced Micro Devices, Inc., Phoenix (n.d.), [Online]. Available: https://github.com/clMathLibraries/clRNG
6. Cook, S.: CUDA Programming: a Developer's Guide to Parallel Computing with GPUs. Morgan Kaufmann Publishers Inc., San Francisco (2013)
7. *cuBLAS*, Nvidia Corporation, Santa Clara (n.d.), [Online]. Available: https://developer.nvidia.com/cublas
8. N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. of 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, San Diego, 2005, pp. 886–893
9. N. Dalal, B. Triggs, and C. Schmid, "Human Detection Using Oriented Histograms of Flow and Appearance," in *Computer Vision (ECCV* 2006*)*, Springer Berlin Heidelberg, 2006, pp. 428–441
10. M. Everingham, L. Van-Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results," [Online]. Available: http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html

11. A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? The KITTI vision benchmark suite,” in *Proc. of Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012
12. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. The MIT Press (2016)
13. J. Gu, Y. Liu, Y. Gao, and M. Zhu, “OpenCL Caffe: Accelerating and Enabling a Cross Platform Machine Learning Framework,” in *Proc. The 4th International Workshop on OpenCL*, New York, 2016, pp 8:1–8:5
14. H. Haseljic, E. Cogo, I. Prazina, R. Turcinhodzic, E. Buza, and A. Akagic, “OpenCL Superpixel Implementation on a General Purpose Multi-core CPU,” in *Proc. of 2018 IEEE International Conference on Imaging Systems and Techniques (IST),* Krakow, Poland, 2018
15. Hendry, Chern, R.: Automatic License Plate Recognition via sliding-window darknet-YOLO deep learning. Image Vis. Comput. **87**, 47–56 (2019)
16. Ji, Y., Kim, S., Kim, Y., Lee, K.: Human-like sign-language learning method using deep learning. ETRI J. **40**, 435–445 (2018)
17. Kim, J., Ryu, J.H., Han, T.M.: Multimodal Interface based on novel HMI UI/UX for in-vehicle infotainment system. ETRI J. **37**(4), 793–803 (2015)
18. Y. Koo, J. Kim, and W. Han, “A method for driving control authority transition for cooperative autonomous vehicle,” in *Proc. 2015 IEEE Intelligent Vehicles Symposium*, Seoul, 2015, pp. 394–399
19. Y. Koo, C. You, and S. Kim, “OpenCL-Darknet: An OpenCL Implementation for Object Detection,” in *Proc. The 1st International Workshop on Driving Computing Platform for Autonomous Vehicles*, Shanghai, 2018
20. W. Lee, and W. Loh, “G-OPTICS: fast ordering density-based cluster objects using graphics processing units,” in Int. J. Web Grid Serv., vol. 14(3), 2018
21. L. Liao, K. Li, K. Li, C. Yang, and Q. Tian, “UHCL-Darknet: An OpenCL-based Deep Neural Network Framework for Heterogeneous Multi−/Many-core Clusters,” in *Proc. of the 47th International Conference on Parallel Processing*, Eugene, OR, USA, 2018
22. T. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft COCO: Common objects in context,” in *European conference on computer vision*, pp. 740–755, 2014
23. Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., Haehnel, D., Hilden, T., Hoffmann, G., Huhnke, B., Johnston, D., Klumpp, S., Langer, D., Levandowski, A., Levinson, J., Marcil, J., Orenstein, D., Paefgen, J., Penny, I., Petrovskaya, A., Pflueger, M., Stanek, G., Stavens, D., Vogt, A., Thrun, S.: Junior: the Stanford entry in the urban challenge. J. Field Rob. **25**(9), 569–597 (2008)
24. A. Neubeck and L. Van Gool, “Efficient Non-Maximum Suppression,” in *Proc. The 18th International Conference on Pattern Recognition*, Washington, 2006, pp. 850–855
25. Noh, S., An, K.: Decision-making framework for automated driving in highway environments. IEEE Trans. Intell. Transp. Syst. **19**(1), 58–71 (2018)
26. Noh, S., Park, B., An, K., Koo, Y., Han, W.: Co-pilot agent for vehicle/driver cooperative and autonomous driving. ETRI J. **37**(5), 1032–1043 (2015)
27. C. Nugteren, “CLBlast: A Tuned OpenCL BLAS Library,” *arXiv preprint*, 2017
28. C. Nugteren, “CLTune: A Generic Auto-Tuner for OpenCL Kernels,” *arXiv preprint*, 2017
29. C. P. Papageorgiou, M. Oren, and T. Poggio, “A general framework for object detection,” in *Proc. 6th International Conference on Computer Vision*, Bombay, 1998, pp. 555–562
30. Park, M., Lee, S., Han, W.: Development of steering control system for autonomous vehicle using geometry-based path tracking algorithm. ETRI J. **37**(3), 617–625 (2015)
31. J. Redmon, “Darknet: Open Source Neural Networks in C (n.d.),” [Online]. Available: http://pjreddie.com/darknet
32. J. Redmon and A. Farhadi, “YOLO9000: Better, Faster, Stronger,” *arXiv preprint*, 2016
33. S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” in *Proc.* Advances in Neural Information Processing Systems, Montréal, 2015, pp. 91–99
34. Rowley, H.A., Baluja, S., Kanade, T.: Neural network-based face detection. IEEE Trans. Pattern Anal. Mach. Intell. **20**, 23–38 (1998)
35. Stone, J.E., Gohara, D., Shi, G.: OpenCL: a parallel programming standard for heterogeneous computing systems. Comput. Sci. Eng. **12**(3), 66–73 (2010)
36. B. Su and K. Keutzer, “clSpMV: A Cross-Platform OpenCL SpMV Framework on GPUs,” in *Proc. The 26th ACM International Conference on Supercomputing*, New York, 2012, pp 353–364
37. Viola, P., Jones, M.J.: Robust real-time face detection. Int. J. Comput. Vis. **57**, 137–154 (2004)

## Affiliations

**Yongbon Koo**[1] • **Sunghoon Kim**[1] • **Young-guk Ha**[2]

✉ Young-guk Ha
ygha@konkuk.ac.kr

Yongbon Koo
ybkoo@etri.re.kr

Sunghoon Kim
saint@etri.re.kr

[1] Electronics and Telecommunications Research Institute, 218 Gajeong-ro, Yuseong-gu, Daejeon, Republic of Korea

[2] Konkuk University, 120 Neungdong-ro, Gwangjin-gu, Seoul, Republic of Korea