KIPS Trans. Comp. and Comm. Sys.
Vol.12, No.11 pp.319~328
ISSN: 2287-5891 (Print), ISSN 2734-049X (Online)

https://doi.org/10.3745/KTCCS.2023.12.11.319

A Study on the Image/Video Data Processing Methods for Edge Computing-Based Object Detection Service

Jang Shin Won† · Yong-Geun Hong††

ABSTRACT

Unlike cloud computing, edge computing technology analyzes and judges data close to devices and users, providing advantages such as real-time service, sensitive data protection, and reduced network traffic. EdgeX Foundry, a representative open source of edge computing platforms, is an open source-based edge middleware platform that provides services between various devices and IT systems in the real world. EdgeX Foundry provides a service for handling camera devices, along with a service for handling existing sensed data, which only supports simple streaming and camera device management and does not store or process image data obtained from the device inside EdgeX. This paper presents a technique that can store and process image data inside EdgeX by applying some of the services provided by EdgeX Foundry. Based on the proposed technique, a service pipeline for object detection services used core in the field of autonomous driving was created for experiments and performance evaluation, and then compared and analyzed with existing methods.

Keywords : EdgeX Foundry, Edge Computing, Object Detection, Artificial Intelligence

에지 컴퓨팅 기반 객체탐지 서비스를 위한 이미지/동영상 데이터 처리 기법에 관한 연구

장 신 원† · 홍 용 근††

요 약

에지 컴퓨팅 기술은 클라우드 컴퓨팅과 달리 기기와 사용자와 가까운 곳에서 데이터를 분석하고 판단하여 실시간 서비스, 민감한 데이터 보호, 네트워크 트래픽 감소와 같은 장점을 제공한다. 에지 컴퓨팅 플랫폼의 대표적인 오픈소스인 EdgeX Foundry는 현실 세계의 다양한 장치와 IT 시스템 사이에서 서비스를 제공하는 오픈소스 기반 엣지 미들웨어 플랫폼이다. EdgeX Foundry는 기존의 센싱된 데이터를 다루기 위한 서비스와 함께 카메라 장치를 다루기 위한 서비스를 제공하는데, 이 서비스는 단순 스트리밍 및 카메라 장치 관리만 지원할 뿐 EdgeX 내부에 장치에서 얻은 이미지 데이터를 저장하거나 처리하지 않는다. 본 논문에서는 EdgeX Foundry에서 제공하는 서비스 일부를 응용하여 EdgeX 내부에 이미지 데이터를 저장하고 처리할 수 있는 기법을 제시한다. 제시한 기법을 기반으로 실험 및 성능 평가를 위해 자율주행 분야에서 핵심적으로 사용되는 객체탐지 서비스를 위한 서비스 파이프라인을 만든 후 기존 방법과 비교 분석하였다. 이 실험을 통해 에지 컴퓨팅 플랫폼에서 이미지/동영상 데이터를 저장하고 처리하는 과정 등이 추가되었음에도 기존 방법에 비해 지연시간이 거의 없는 것을 확인할 수 있었다.

키워드 : EdgeX Foundry, 에지 컴퓨팅, 객체탐지, 인공지능

1. 서 론

하드웨어 및 통신 기술의 발전에 따라 다방면에서 IoT 및 인공지능 기술의 접목 시도가 이루어지고 있다. 이러한 도전은 동영상 및 이미지 분야에도 적극적으로 이루어지고 있으며, 이와 함께 카메라 디바이스를 다루는 에지 컴퓨팅 플랫폼들도 등장하고 있다. EdgeX Foundry는 이러한 에지 컴퓨팅 플랫폼 중 하나로 ds-usb-camera 라는 마이크로 서비스와 함께 카메라 디바이스의 메타데이터를 관리하고 스트리밍 할 수 있는 기법을 제공한다. 그러나 카메라 디바이스를 통해 습득한 동영상 및 이미지 데이터를 저장하거나 처리하는 기법은 제공하지 않는다. 지금까지의 에지 컴퓨팅 플랫폼은 숫자, 문자 등의 간단하면서도 용량이 작은 데이터들만 주로 처리하였기에, 동영상 및 이미지 데이터 등과 같은 복잡하고 용량이 큰 데이터는 아직 에지 컴퓨팅 플랫폼 내부에서 처리하지 못하고 있다. ds-usb-camera에서 제공하는 아키텍쳐에서도 카메라 디바이스가 촬영한 이미지(영상)가 EdgeX Foundry 내부에 저장되지 않으며 단순히 스트리밍 주소만 외부 서버에 제공할 뿐이다. 용량이 큰 동영상이나 이미지 자체를 에지 컴퓨팅 플랫폼 내부에 저장하는 것은 지금까지는 이런 수요가 없었기에 필요 없었다. 하지만, 향후 많은 분야에서 적용될 것으로 예상되는 이미지 분류, 객체탐지 같은 대용량의 이미지/동영상 데

※ 이 논문은 2021년도 정부(과학기술 정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임(2021-0-00188, AI 기능 지원 프레임워크 기반의 이기종 IoT 플랫폼 연동 오픈소스 및 국제 표준 개발).
† 비 회 원 : 대전대학교 AI연구실 연구원
†† 정 회 원 : 대전대학교 AI융합학과 교수
Manuscript Received : September 1, 2023
Accepted : September 25, 2023
* Corresponding Author : Yong-Geun Hong(yghong@dju.kr)

이터를 다루는 인공지능 서비스를 에지 컴퓨팅 기반으로 제공하기 위해서는 이러한 데이터도 에지 컴퓨팅 내부에서 저장하고 처리하는 기술이 꼭 필요한 기능이다.

따라서 본 논문은 에지 컴퓨팅 환경에서 카메라 디바이스를 통해 습득한 동영상 및 이미지 데이터를 에지 컴퓨팅 플랫폼 내부적으로 저장하고 다루는 기법에 관해 기술하고자 한다. 설계한 기법을 EdgeX Foundry라는 오픈소스 기반으로 구현하여 정상적으로 동작 여부 확인 및 기존 기법과의 성능 평가도 수행하였다. 본 논문의 구성은 다음과 같다. 먼저 2장에서는 에지 컴퓨팅 플랫폼으로써 EdgeX Foundry에 관한 내용과 일반적으로 대역폭이 낮고 대기 시간이 긴 IoT 환경에서 사용하는 메시징 프로토콜인 Message Bus에 관한 내용을 설명한다. 이후 제시한 방안의 핵심인 컴퓨터의 이미지 인식에 대하여 설명한다. 3장에서는 에지 컴퓨팅 플랫폼 내부에 이미지를 저장하고 가공하는 Image to Array 기법에 관하여 기술하며, 그 동작과 결과에 관해 기술한다. 4장은 제시한 Image to Array 기법이 적용된 객체탐지 예시 파이프라인을 만든 후, 파이프라인을 통해 걸리는 시간을 측정하여 성능을 평가한다. 마지막 5장에서는 결론 및 향후 연구 방향에 관하여 기술한다.

2. 관련 연구

2.1 EdgeX Foundry

EdgeX Foundry는 현실 세계의 다양한 장치와 IT 시스템 사이에서 서비스를 제공하는 오픈소스 기반 엣지 미들웨어 플랫폼이다. Fig. 1에서 보는 바와 같이 EdgeX Foundry는 Device Services, Core Services, Supporting Services, Application Services의 총 4개의 계층으로 구성된다. Device Services는 센서 및 장치와 EdgeX를 연결하는 계층이다. Device Services에서 수집된 데이터는 EdgeX 데이터로 변환되며 일반적으로 Core Services 계층으로 전달된다. Supporting Services 계층은 다른 세 계층에 속하지 않는 나머지 마이크로 서비스를 의미하며 습득한 데이터를 가공하는 규칙, 알람 주기 등의 기능이 포함된다. Application Services 계층은 EdgeX의 최북단 계층으로, 데이터를 추출, 처리 및 변환하고 EdgeX 외부로 내보내는 계층이다. 일반적으로 데이터는 최남단인 Device Services 계층에서 최북단인 Application Services 계층으로 이동한다[1].

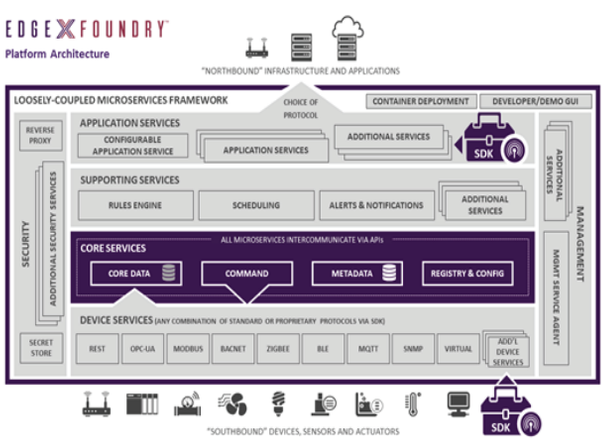


Fig. 1. EdgeX Layers[1]

2.2 Message Bus

1) Message Bus 개요

Message Bus는 서로 다른 시스템이 하나의 공통 메시징 플랫폼을 통해 표준화된 형식으로 통신하고 데이터를 교환할 수 있도록 하는 메시징 인프라이다. Message Bus를 사용함으로써 각 구성 요소 간 결합도를 낮추어 독립적이고 유연한 작동을 지원할 수 있다. 이 메시징 시스템은 Topic과 Publish, Subscribe를 통해 작동하며 그 예는 Fig. 2에서 보이는 바와 같다. 하나의 Message Bus로 연결된 온도 측정 디바이스와 스프링클러가 있다고 가정한다. 온도 측정 디바이스는 'Temperature'라는 Topic의 Publisher이며 스프링클러는 'Temperature'라는 Topic의 Subscriber이다. 온도 측정 디바이스는 특정 주기마다 'Temperature' Topic으로 온도를 Publish 한다. 'Temperature'의 Subscriber인 스프링클러는 'Temperature'라는 Topic을 통해 온도 값을 전달받으며, 온도 값에 따라 정해진 처리를 실행한다. 각 구성 요소는 Publisher이자 Subscriber일 수 있으며 동시에 여러 Topic에 대해 Subscribe가 가능하다[2].

2) EdgeX의 Message Bus와 REST API

EdgeX는 계층 간 통신으로 Message Bus를 사용한다. 기본적으로 Redis DB를 사용하며, ZMQ, MQTT, NATS Bus 프로토콜을 지원한다. EdgeX의 REST(REpresentational State Transfer) API는 장치 및 센서와 EdgeX 시스템이 상호 작용하는 데에 사용된다. 즉, 장치에서 수집된 이벤트는 HTTP endpoint를 통해 Core Services 계층의 Core Data 마이크로 서비스에 전달되며 선택적으로 DB에 저장한 후 Message Bus에서 지정한 Topic으로 이를 게시한다. 게시된 이벤트는 모든 계층에서 Topic을 통해 접근할 수 있다[3]. Fig. 3은 이러한 동작 과정을 표시한 것이다.

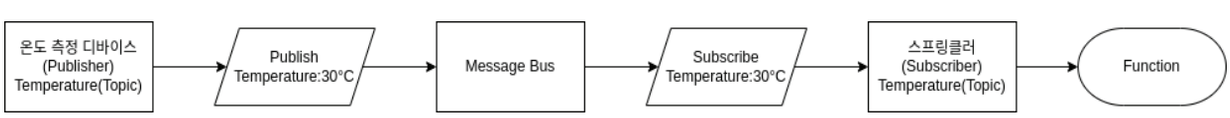


Fig. 2. Message Bus Pub/Sub

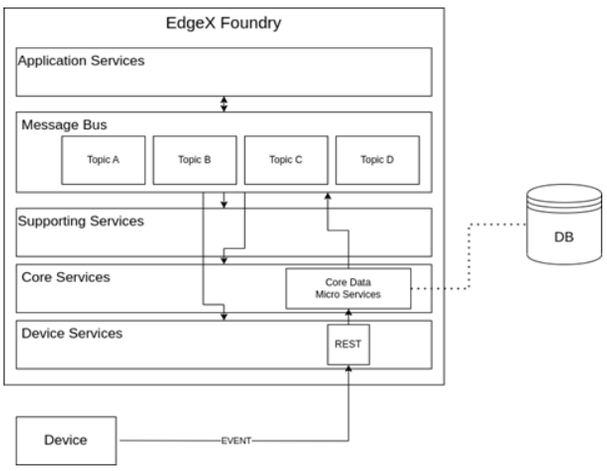


Fig. 3. Message delivery in EdgeX

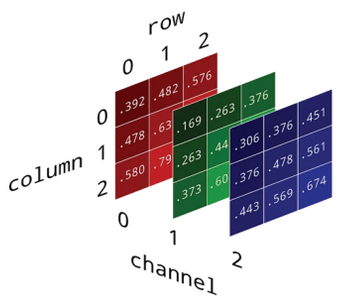


Fig. 4. Image Array[4]

2.3 컴퓨터의 이미지 인식

컴퓨터는 숫자 형식의 데이터만 이해할 수 있다. 따라서 이미지는 컴퓨터에 저장될 때 몇 가지 과정을 거쳐 숫자들로 변환된다. 가장 먼저 적색, 녹색, 청색의 세 채널로 나누어진다. 이후 이 채널들은 3차원 배열로 변환되며, 배열의 각 요소는 이미지의 픽셀과 매핑된다. 요소는 0에서 255의 값을 가질 수 있으며 이들은 픽셀에 대한 채널의 강도(Intensity)를 뜻한다. 값이 0에 가까워질수록 강도가 약함을, 255에 가까워질수록 강도가 강함을 의미한다. 즉, 컴퓨터는 Fig. 4에서 보는 바와 같이 이미지 데이터를 3차원 배열 형태로 저장 및 인식한다[4].

3. 에지 컴퓨팅 내부에서 이미지/동영상을 처리하고 저장하기 위한 Image to Array 기법

3.1 EdgeX에서 제공하는 카메라 서비스 한계

EdgeX에서 제공하는 카메라 서비스는 단순히 카메라의 메타데이터를 관리하고, 해당 카메라의 스트리밍 주소를 제공한다. 즉, 카메라를 통해 얻은 동영상 또는 이미지 데이터가 EdgeX 내부에 전달되지 않는다. Fig. 5는 EdgeX Foundry에서 제공하는 ds-usb-camera가 사용하는 아키텍쳐이다. 이 아키텍쳐에서 카메라 디바이스가 촬영한 이미지(영상)는 EdgeX Foundary 내부에 저장되지 않으며 단순히 영상 스트리밍 주소만 외부 서버에 제공할 뿐이다[5].

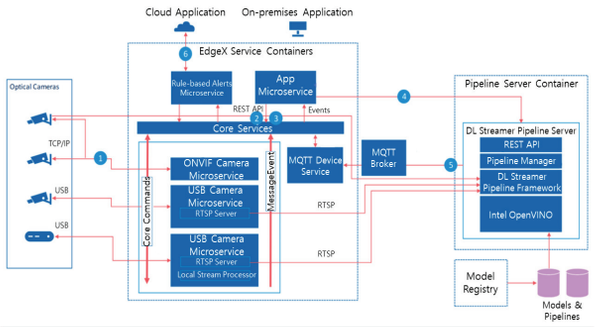


Fig. 5. Camera Service Architecture[5]

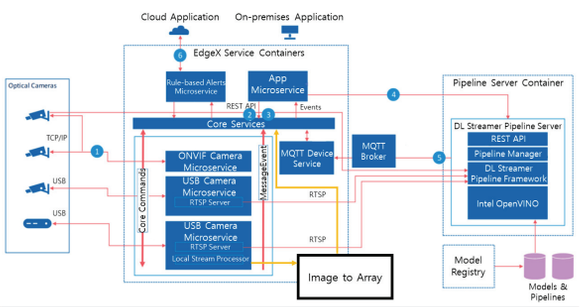


Fig. 6. Camera Service Architecture with Image to Array

그러므로 위 아키텍쳐에서, 이미지 데이터는 단순히 스트리밍될 뿐이며 이는 에지 컴퓨팅 플랫폼 위에서 작동하였다고 보기 어렵다. 또한 사용자는 이미지 데이터에 대한 정보를 확인하거나 저장 및 가공할 수 없다. 본 논문에서는 이러한 문제점을 해결하기 위하여 에지 컴퓨팅 플랫폼 내부에 이미지 데이터를 저장하고 다룰 수 있는 Image to Array 기법을 제안하고자 한다.

3.2 Image to Array 기법 제안

Image to Array 기법은 컴퓨터의 이미지 인식 방법에서 착안한 것으로, 2가지 핵심 함수로 구성된다. 먼저 카메라를 통해 얻은 이미지 데이터를 문자열 형태의 다차원 배열로 변환하는 변환 함수와 변환한 내용을 REST API를 활용하여 지정 endpoint에 POST하는 전송 함수로 구성된다.

해당 서비스는 다음의 환경 변수가 필요하다.

- IMAGES_DIRECTORY : 이미지 데이터가 있는 디렉토리
- POST_URL : 다차원 배열로 변환된 이미지 데이터를 POST하는 endpoint

제시한 Image to Array 기법이 적용된 아키텍쳐는 Fig. 6과 같다. 촬영된 이미지는 Image to Array 기법을 통해 문자열 형의 다차원 배열로 변환되며 REST API를 이용해 POST된다.

3.3 동작 결과 확인

POST_URL을 EdgeX의 endpoint로 지정한 후 제시한 방안을 수행하였다. 작동 결과는 EdgeX UI(localhost:4000)의 Events/Readings에서 확인할 수 있다.

"objectValue":"[[[81, 137, 62], [62, 117, 36], [51, 107, 16], [55, 119, 22], [69, 129, 31],

Fig. 7. Results of Image to Array

{
"Id": "4c9af6d1-0a18-49a7-b719-d6d25e27eca5",
"Origin": 1674022437845086516,
"DeviceName": "sample-json",
"ResourceName": "json",
"ProfileName": "sample-json",
"ValueType": "Object",
"Units": "",
"ObjectValue": "[[[81, 137, 62], [62, 117, 36], [51, 107, 16], [55, 119, 22], [69, 129, 3
1], [85, 134, 42], [92, 139, 47], [91, 142, 49], [75, 141, 33], [58, 118, 20], [49, 116, 1
], [53, 120, 15], [62, 117, 16], [34, 79, 0], [33, 69, 5], [95, 121, 47], [120, 141, 64], [
122, 139, 68], [85, 112, 45], [63, 108, 39], [52, 111, 31], [67, 126, 36], [78, 134, 45], [
70, 128, 44], [62, 114, 29], [63, 115, 40], [73, 126, 56], [62, 116, 41], [69, 121, 46], [6
3, 135, 60], [95, 150, 67], [57, 118, 25], [68, 125, 28], [96, 152, 53], [58, 112, 16], [62
, 114, 29], [93, 137, 62], [114, 151, 82], [143, 180, 110], [65, 111, 38], [26, 78, 4], [47
, 99, 24], [45, 98, 20], [65, 118, 40], [66, 120, 44], [60, 114, 39], [72, 124, 50], [97, 1
44, 72], [98, 150, 68], [84, 138, 54], [90, 146, 59], [80, 140, 52], [67, 129, 43], [76, 15
5], [128, 95, 60], [89, 60, 18], [146, 117, 75], [143, 116, 69], [104, 80, 32], [170, 148, 10
7], [165, 142, 101], [167, 144, 103], [155, 133, 92], [157, 138, 96], [150, 133, 90], [150, 1
28, 87], [144, 121, 79], [131, 105, 56], [141, 117, 69], [150, 128, 81], [164, 142, 95], [159
, 135, 89], [140, 114, 65], [146, 123, 71], [151, 128, 74], [161, 137, 89], [184, 161, 111],
[195, 169, 120], [205, 178, 125], [233, 206, 151], [232, 207, 150], [232, 210, 153], [230, 20
9, 152], [230, 209, 154], [223, 201, 141], [238, 213, 147], [243, 218, 154], [246, 222, 162],
[243, 222, 167], [246, 227, 171], [239, 220, 162], [236, 216, 165], [209, 188, 141], [186, 16
7, 125], [173, 155, 117], [149, 135, 98], [137, 124, 90], [160, 146, 119], [165, 149, 126], [
132, 125, 97], [86, 76, 51], [75, 63, 41], [58, 43, 22], [44, 29, 10], [64, 49, 30], [66, 50,
34], [30, 17, 0], [27, 14, 5], [30, 19, 13], [35, 25, 16], [66, 58, 39], [57, 44, 27], [50, 3
3, 23], [34, 16, 4], [41, 25, 2], [53, 32, 13], [49, 28, 7], [49, 31, 9], [62, 45, 27], [29,
15, 4], [36, 22, 13], [33, 17, 2], [55, 38, 18], [66, 50, 27], [43, 35, 12], [40, 33, 14], [3
7, 23, 10], [30, 12, 2], [28, 11, 1], [38, 22, 7], [43, 24, 9], [47, 21, 4], [60, 36, 10], [6
5, 40, 18], [41, 17, 7], [37, 15, 4], [52, 31, 10], [52, 30, 9], [38, 9, 1], [68, 40, 19], [5
7, 29, 15], [47, 20, 11], [46, 22, 12], [42, 20, 7], [45, 24, 7], [48, 27, 6], [49, 28, 7], [
44, 21, 3], [38, 15, 0]]"
}

Fig. 8. Stored data with Image to Array

Fig. 7에서 EdgeX 내부에 저장하려고 하는 이미지 데이터가 EdgeX Events/Readings 객체의 objectValue 키의 value에 다차원 배열 형태로 저장된 것을 확인할 수 있다. 이때, 배열이 너무 길어 온전히 표현되지 않는 것을 확인할 수 있다. 제시한 방안이 에지 컴퓨팅 플랫폼(디바이스) 수준에서 다루기 어려운 수준의 크기이기에 표현되지 못한 것인지, 아니면 단순히 페이지의 출력 한도에 의해 그러한 것인지 확인하기 위하여 EdgeX의 redis DB에 RESP.app를 통해 직접 접근하였고 EdgeX Events 객체의 objectValue의 value에 온전한 다차원 배열 형태로 저장되었음을 확인하였다.

제시한 방안을 사용해 EdgeX 에지 컴퓨팅 플랫폼 내부에 이미지 데이터를 저장할 수 있다. 이때, 저장되는 데이터는 바이너리 파일과는 달리 숫자의 다차원 배열 형태이므로 가공 및 변환이 용이하다.

4. 객체탐지 서비스에서 Image to Array 기법 성능 평가

4.1 전체 시스템 구성

본 논문에서는 Image to Array 기법의 활용과 그 확장성을 평가하고 성능을 측정하기 위하여 일련의 객체탐지 서비스 예시 파이프라인을 구축하였다. 측정하고자 하는 성능은 시간 복잡도 측면의 성능이다.

해당 실험의 실험군과 대조군은 다음과 같이 설정하였다. 제시한 파이프라인을 실험군으로, Intel AI github에서 제공하는 object detection benchmark.py 모듈을 대조군으로 사용할 수 있도록 수정 후[6] 제안하는 Image to Array 기법을 활용하여 동작하는 tensorflow serving에 요청하는 과정을 대조군으로 설정하였다.

해당 실험에서 측정한 내용은 다음과 같다.

Table 1. Machine Information

	Machine-1	Machine-2	Machine-3
CPU	Intel Core i7-11700K	AMD Ryzen 7 5800H	Intel Core i7-1165G7
GPU	GeForce RTX 3060 Lite Hash Rate	GeForece RTX 3060 Mobile Max-Q	Intel Iris Xe Graphics
Single-core score	1,804	1,542	1,652
Multi-core score	10,805	8,589	6,018

Fig. 9. Images Used in the Experiment

- Image to Array 기법에 의해 변환되기 전 이미지 사이즈와 Image to Array 기법을 통해 변환 후 사이즈
- Image to Array 기법에 의해 변환 후의 크기 배율
- 이미지 한 개의 객체 탐지에 소요된 시간(추론 소요 시간)
- 이미지 한 개의 변환 및 전송, 객체 탐지 등 모든 과정을 포함하여 소요된 시간(총 소요 시간).

해당 실험은 각기 다른 성능의 머신 3개에서 수행되었다. 머신-1은 Intel Core i7-11700K CPU 및 GeForce RTX 3060 Lite Hash Rate를 사용하는 머신(일반 PC)이다. Geekbench 수행 결과에서 Single-core 점수는 1,804점, Multi-core 점수는 10,805점이 나왔다. 머신-2는 AMD Ryzen 7 5800H CPU 및 GeForece RTX 3060 Mobile Max-Q를 사용하는 머신(노트북)이며 Geekbench 수행 결과 Single-core 점수는 1,532점, Multi-core 점수는 8,589점이 나왔다. 머신-3는 Intel Core i7-1165G7 CPU 및 Intel Iris Xe Graphics를 사용하는 머신(노트북)이며, Geekbench 수행 결과에서 Single-core 점수는 1,652점, Multi-core 점수는 6,018점이 나왔다.

실험에 사용된 이미지는 2017 validation COCO dataset이다. 이 중 10개의 이미지를 선정하여 각 군마다 성능을 평가하였다. Fig. 9는 사용된 이미지와 할당한 번호이다.

4.2 EdgeX Foundry 구성

에지 컴퓨팅 플랫폼 오픈소스로 카메라 장치를 지원하는 미들웨어 플랫폼인 EdgeX Foundry를 사용하였다. EdgeX

Compose를 이용하여 Docker container 환경에서 EdgeX를 구성하였고, EdgeX Foundry 2.2 Kamakura 버전을 사용하였다.

사용한 서비스는 ds-usb-camera, ds-rest, asc-http, mqtt-bus이다. ds-usb-camera는 카메라 장치의 메타데이터를 관리하고 스트리밍 주소를 제공하는 서비스이다. ds-rest는 EdgeX의 Core Data 마이크로 서비스에 데이터를 푸시할 수 있도록 데이터 프로필과 HTTP endpoint를 생성해주는 서비스이다. 본 논문에서는 ds-rest의 여러 프로필 중 sample-json 프로필을 사용하였다. asc-http는 지정 http endpoint로 EdgeX의 데이터를 내보내는 서비스이다. Trigger로 edgex-messagebus를 사용하였다. mqtt-bus는 EdgeX의 internal message bus로 mqtt를 사용하도록 만드는 서비스이다[1]. Fig. 10은 EdgeX에서 정의한 Makefile에 따라 EdgeX 및 나열한 옵션을 수행하는 코드이며, Fig. 11은 실행한 EdgeX와 마이크로 서비스들에서 데이터의 대략적인 흐름도이다.

```
$ make run ds-usb-camera no-secty ds-rest asc-http mqtt-bus
```

Fig. 10. Code for EdgeX Compose Execution

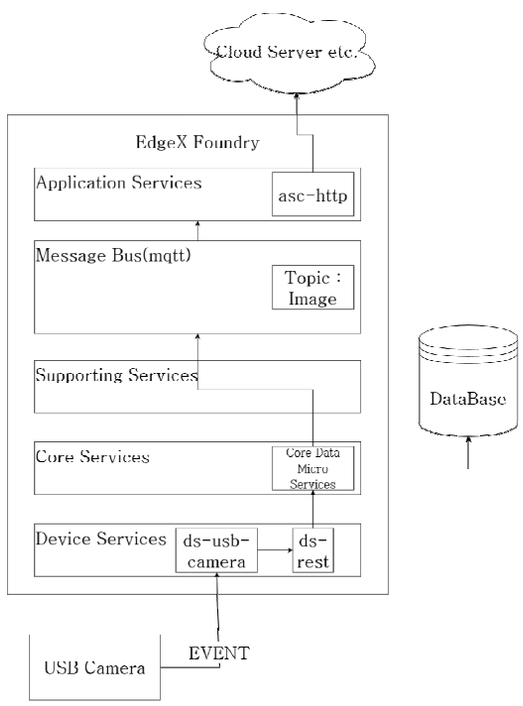


Fig. 11. Flowchart According to Configuration

```
docker run \
    --name=tfserving \
    -d \
    -p 8501:8501 \
    -v "$(pwd)/obj_detection_rfcn:/models/$model_name" \
    -e MODEL_NAME=$model_name \
    tensorflow/serving:latest
```

Fig. 12. Code for Execution of Tensorflow Serving


```
{
  "Id": "ee47d048-b4e9-4b0c-945c-1f66abbb9081",
  "Origin": 1674024041414281812,
  "DeviceName": "sample-json",
  "ResourceName": "json",
  "ProfileName": "sample-json",
  "ValueType": "Object",
  "Units": "",
  "ObjectValue": "{\"instances\" : [[[[81, 137, 62], [62, 117, 36], [51, 107, 16], [55, 119, 22], [69, 129, 31], [85, 134, 42], [92, 139, 47], [91, 142, 49], [75, 141, 33], [58, 118, 20]
```


Fig. 13. Image to Array for Tensorflow Serving

4.3 EdgeX Foundry 구성

Tensorflow serving은 고성능으로 유연하게 AI 서비스 운영을 지원하는 시스템이다. Docker container 환경에서 수행 가능하며, REST API를 지원하므로 endpoint와 적절한 request format으로 손쉬운 사용이 가능하다[7].

본 실험은 제시한 서비스의 시간 복잡도를 평가하는 것이므로 별도의 모델 학습 없이, Intel AI github에서 제공하는 사전 학습된 two-stage 모델 RFCN(Region-based Fully Convolutional Networks)과 one-stage 모델 SSD(Single Shot multibox Detector)-MobileNet을 사용하였다. 다음의 Fig. 12는 SSD-MobileNet을 사용한 docker tensorflow serving container를 실행하는 코드이다.

4.4 Image to Array for Tensorflow Serving

위에서 제시한 Image to Array 기법을 tensorflow serving의 REST Predict request format[7]에 맞추어 저장하도록 개량하였다. REST Predict request format은 중괄호 안에 instances 키와 다차원 배열의 value를 요구한다. Fig. 13은 tensorflow serving의 REST Predict request format에 맞춰 EdgeX에 저장된 데이터이다.

본 실험에서는 이러한 가공 및 변환 과정을 에지 컴퓨팅 플랫폼 내부에 저장되기 전에 수행하였다. Image to Array 기법은 이미지를 다차원 배열 형태로 저장하므로 에지 컴퓨팅 플랫폼 내부나 외부에서도 다양한 형태로 가공 및 사용할 수 있다.

4.5 Detection Pipeline Server with EdgeX

AI 기반 서비스를 제공하기 위해 사용된 사전 훈련된 모델(pre-trained model)과 서버 플랫폼에 따라, AI 서비스 사용을 위한 이미지의 입력 형태가 다소 상이할 수는 있겠지만, 이미지는 배열과 유사한 형태로 바뀌어야 한다는 것은 공통적이다.

가령 본 논문에서 사용한 SSD-MobileNet과 RFCN의 경우 입력으로 4차원 배열을 요구한다. 또한 논문에서 사용한 tensorflow serving은 클라이언트로부터 REST 통신을 통해 데이터를 전달받을 수 있는데, 직렬화 형식으로 JSON을 사용한다. 결과적으로, SSD-MobileNet(또는 RFCN)과 tensorflow serving을 활용하여 AI 서비스를 제공하는 경우, 클라이언트는 이미지 배열을 문자열 형태로 변환 후 특정 key의 value로 입력하여 서버로 전달해야 한다. 이때, Image to Array 기법을 사용하여 저장하였으므로, 별다른 변환 과정 없이 저장된 이미지 배열을 그대로 value로 입력하여 이미지 처리 서비스를 손쉽게 사용할 수 있다.

Detection pipeline server with EdgeX 마이크로 서비스는

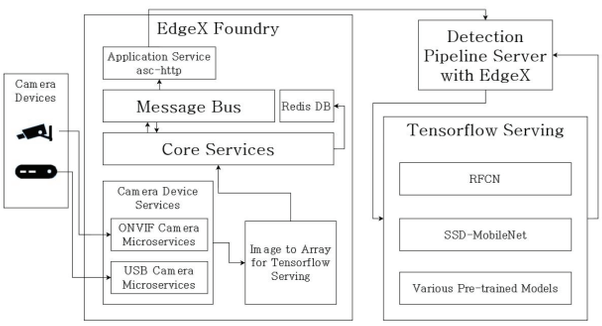


Fig. 14. Detection Pipeline with EdgeX & Image to Array

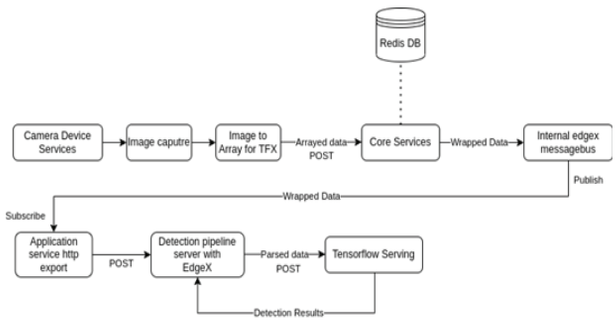


Fig. 15. Detection Pipeline Flow Chart

예시 파이프라인을 구축하기 위해 Flask[8] 및 REST API[9]를 활용하여 만든 간단한 서버다.

'application service http export' 마이크로 서비스를 통해 EdgeX에서 detection pipeline server with EdgeX의 end-point에 EdgeX events 객체를 POST하면, 이를 파싱하여 readings의 objectValue의 value를 추출한다. Image to Array for tensorflow serving에서 EdgeX에 저장할 때 tensorflow serving의 REST Predict request format에 맞추어 저장하였으므로 파싱 이외의 별다른 가공은 필요하지 않으며, 추출한 데이터를 그대로 tensorflow serving에 쿼리한 후 결과를 받는 것으로 서버의 역할은 끝이 난다.

Detection pipeline server with EdgeX 모듈 및 Image to Array 기법과 EdgeX(에지 컴퓨팅 플랫폼)를 활용한 일련의 예시 파이프라인의 최종 아키텍쳐는 Fig. 14와 같다.

먼저 'Camera Device service' 마이크로 서비스를 통해 동영상 스트리밍을 시작한다. 스트리밍 중 캡쳐된 이미지는 저장되며, Image to Array for tensorflow serving을 통해 tensorflow serving의 REST predict request format에 맞추어진 문자열형의 다차원 배열로 변환한 후 EdgeX ds-rest로 생성한 endpoint에 POST된다. 이 endpoint는 EdgeX의 Core Services 계층의 endpoint이다. Core Service 계층에 POST된 데이터는 Redis DB에 저장되며, EdgeX 이벤트 객체로 wrapping되어 internal Message Bus에 Publish 된다. Trigger로 edgex-messagebus를 사용하는 application service http export는 internal edgex message bus로부터 publish된 EdgeX 이벤트 객체를 수신한 후 detection pipeline server with EdgeX로 내보낸다. detection pipeline server with EdgeX는 EdgeX 이벤트 객체를 받아 파싱하여 objectValue의 value에 담긴 이미지 다차원 배열을 추출하여 tensorflow serving에 추론 요청을 한다. tensorflow serving은 객체 탐지를 수행 후 detection pipeline server with EdgeX로 그 결과를 반환한다. Fig. 15는 이를 시각화한 순서도이다.

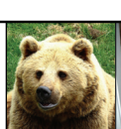
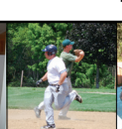
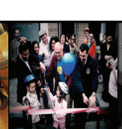
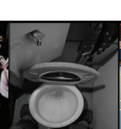
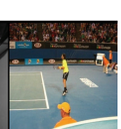
5. 실험 결과

이번 장에서는 앞 장에서 설명한 기법의 실험 측정 결과를 확인하고 관찰한 정보를 고찰하며 성능을 평가한다. '바이너리 이미지를 문자열형으로 변환'이라 하면 일반적으로 크기가 커질 것으로 예측할 수 있다. 그러므로 성능의 평가는 이미지의 크기가 얼마나 증가하였는지, 크기의 증가가 시간 복잡도 측면에서 얼마나 영향을 미칠지 평가함을 의미한다. 각 Fig의 범례 중 experimental은 실험군을 뜻하며, 제시한 방안이 적용된 예시 파이프라인을 가리킨다. comparison은 대조군을 뜻하며, 수정된 object detection benchmark.py 클라이언트 모듈에 의한 수행을 가리킨다. 이때, 화면 출력 및 기록 등은 측정시간에 포함되지 않는다.

5.1 Image to Array for Tensorflow Serving을 통해 변환 전후 사이즈 및 크기 배율

먼저 Image to Array to tensorflow serving을 거친 후의 이미지의 크기와 그 배율을 확인하였다. Table 2는 이미지별 원래 크기와 Image to Array for tensorflow serving을 통해

Table 2. Image Information

Image										
Name	Image-1	Image-2	Image-3	Image-4	Image-5	Image-6	Image-7	Image-8	Image-9	Image-10
Source image size(MB)	0.336	0.062	0.318	0.321	0.015	0.104	0.273	0.212	0.059	0.036
Size after conversion(MB)	5.760	4.121	6.189	4.754	1.216	2.311	5.477	4.143	2.502	1.213
Magnification	17.143	66.468	19.462	14.810	81.067	22.221	20.062	19.542	42.407	33.694

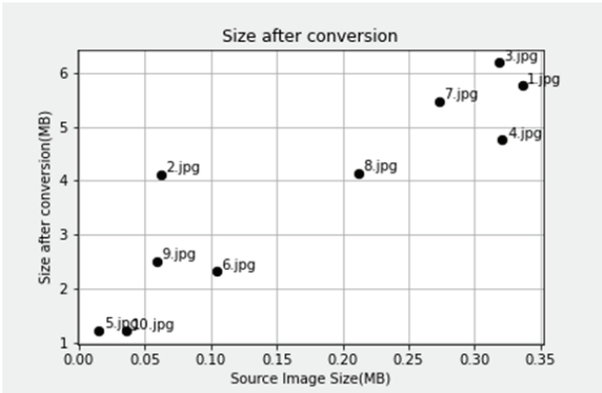


Fig. 16. Size after Conversion

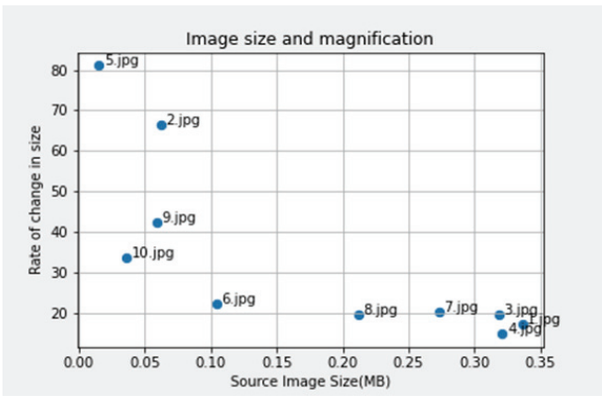


Fig. 17. Image Size and Magnification

변환 후 크기이다. 이를 산점도로 표현하면 Fig. 16과 같다. 상관분석 결과 r=0.901로 원본 크기와 변환 이후 크기는 강한 양의 상관관계가 있음을 확인할 수 있다. 즉, 원본 크기가 클수록 변환 이후 크기도 크다. 반면, 배율은 반대의 경향을 보인다. Fig. 17은 원본 크기와 변화 배율의 산점도이며 상관 분석 시 r=-0.767의 강한 음의 상관을 확인할 수 있다. 즉, 원본 크기가 작을 수록 변화 배율이 높다.

5.2 각 머신 별 추론 소요 시간 비교

이번에는 각 머신에서 모델 별 추론 소요 시간을 비교한다. 추론 소요 시간은 REST API를 통해 tensorflow serving에 추론 요청을 한 직후부터 객체탐지가 수행되고 결과가 반환될 때까지 걸린 시간이다. 추론 시간 복잡도 자체는 일반적으로 머신의 CPU(또는 GPU)성능에 의존한다. Fig. 18, Fig. 19, Fig. 20은 각 머신과 RFCN 모델을 사용했을 때의 이미지 별 추론 소요 시간을 시각화한 자료이며 Fig. 21, Fig. 22, Fig. 23은 각 머신과 SSD-MobileNet 모델을 사용했을 때의 이미지별 추론 소요 시간을 시각화한 자료이다. 3개 머신 및 모델 조합에서 모두 실험군과 대조군 간 차이가 거의 없음을 확인할 수 있다. 추론 소요 시간이 동일하므로, 추론에 소요된 시간을 제외하고 다른 과정에 의해 소요되는 시간들을 비교할 수 있다.

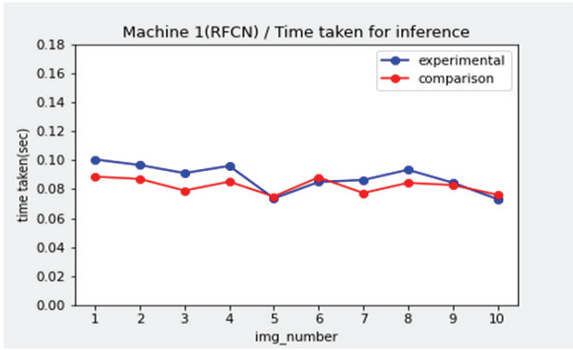


Fig. 18. Machine-1(RFCN) Time Taken for Inference Comparison

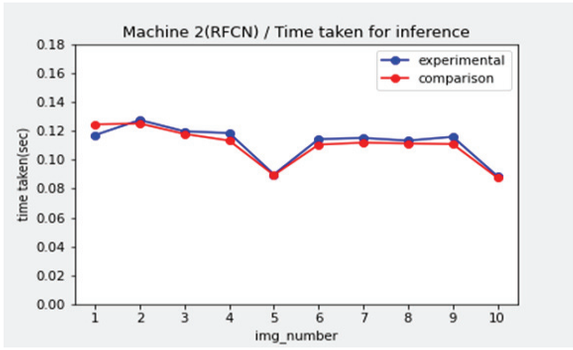


Fig. 19. Machine-2(RFCN) Time Taken for Inference Comparison

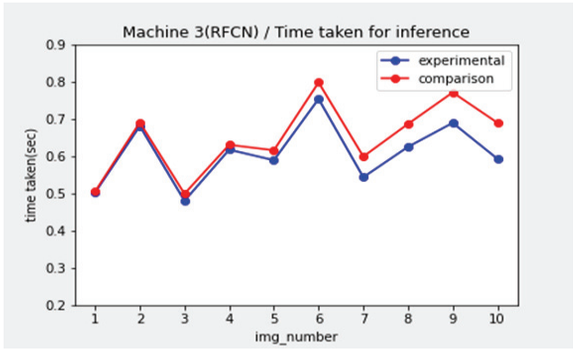


Fig. 20. Machine-3(RFCN) Time Taken for Inference Comparison

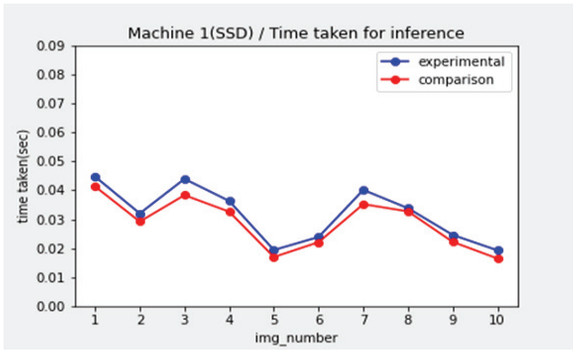


Fig. 21. Machine-1(SSD-Mobilenet) Time Taken for Inference Comparison

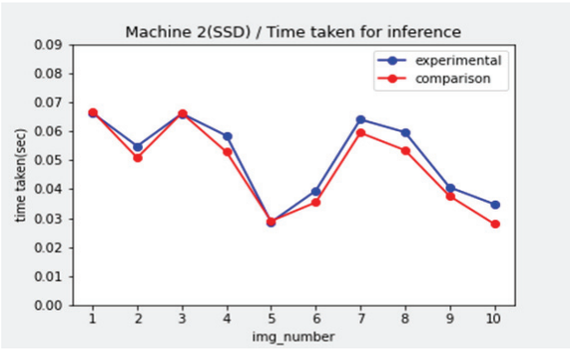


Fig. 22. Machine-2(SSD-Mobilenet) Time Taken for Inference Comparison

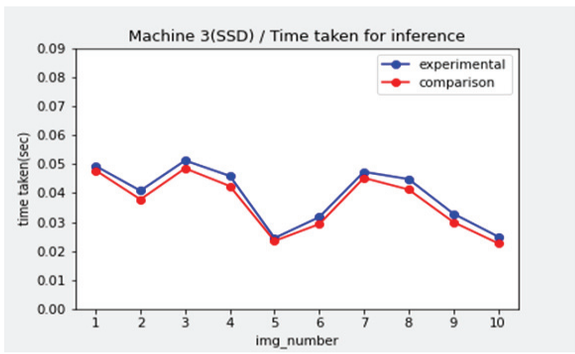


Fig. 23. Machine-3(SSD-Mobilenet) Time Taken for Inference Comparison

5.3 각 머신 별 총 소요 시간 비교

마지막으로 각 머신 별 총 소요 시간을 비교한다. 총 소요 시간은 이미지 변환 과정을 포함하여 모든 과정을 수행하는 동안 걸린 시간이다. 다음의 자료들은 각 머신 및 두 모델 조합에서 이미지 별 총 소요 시간을 시각화 한 자료이다. Fig. 24, Fig. 25, Fig. 26은 각 머신에서 RFCN 모델을 사용한 결과이며, Fig. 27, Fig. 28, Fig. 29는 각 머신에서 SSD-MobileNet 모델을 사용한 결과이다. 3개의 머신 및 두 모델 조합 모두 실험군과 대조군 간 차이가 거의 없음을 확인할 수 있다.

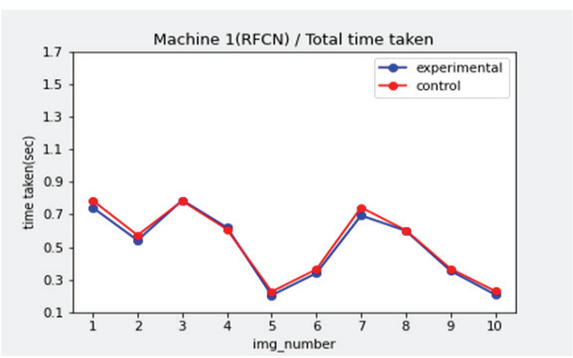


Fig. 24. Machine-1(RFCN) Total Time Taken Comparison

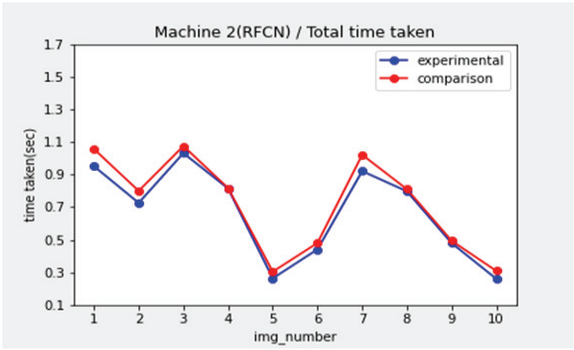


Fig. 25. Machine-2(RFCN) Total Time Taken Comparison

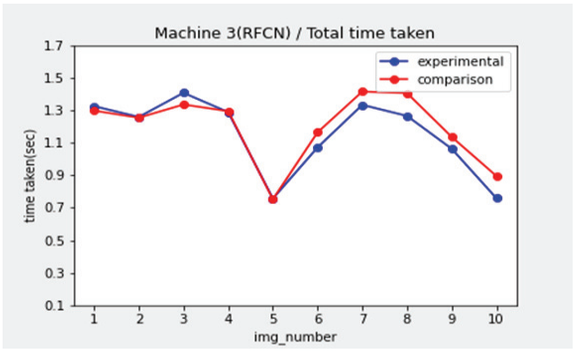


Fig. 26. Machine-3(RFCN) Total Time Taken Comparison

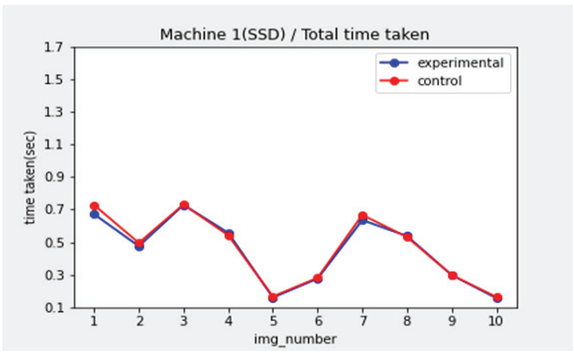


Fig. 27. Machine-1(SSD-Mobilenet) Total Time Taken Comparison

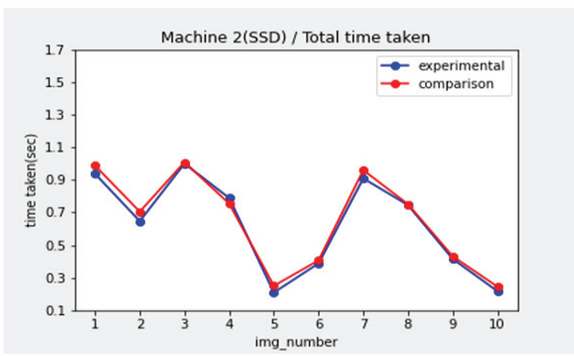


Fig. 28. Machine-2(SSD-Mobilenet) Total Time Taken Comparison

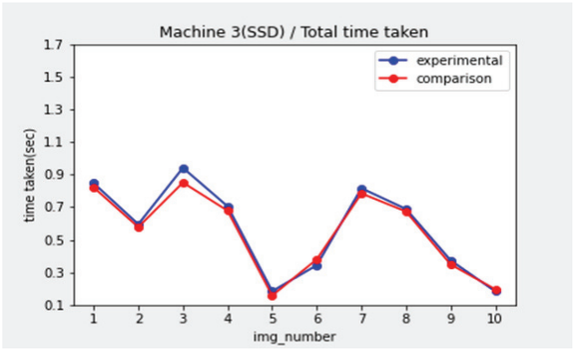


Fig. 29. Machine-3(SSD-Mobilenet) Total Time Taken Comparison

5.4 평가

위 실험을 통해 시간 복잡도 측면에서의 성능을 측정할 수 있었다. 대조군은 클라이언트 모듈을 사용한 단순 추론 요청 및 결과 반환 과정이다. 이에 반해 실험군은 원본 데이터가 에지 컴퓨팅 플랫폼 내부에 저장되는 과정, 에지 컴퓨팅 플랫폼 내부에서 외부 서버로 POST되는 과정, 외부 서버에서 추론 및 결과 반환의 과정을 포함한다. 즉, 더 많은 데이터 이동 과정이 존재하며, 이 과정에서 Image to Array 기법을 통해 증가된 데이터의 크기가 영향을 미칠 것으로 예측하였다. 그러나 위 실험에서 Image to Array 기법을 통해 이미지 데이터의 크기가 증가하더라도, 그 영향이 미비하여 전체 서비스 시간에 영향을 미치지 않음을 확인하였다.

결과의 원인으로 크게 두 가지를 추측할 수 있다. 첫 번째는 루프백 주소를 사용하여 실험이 이루어진 점이다. 루프백 주소란 호스트 자기 자신을 가리키는 주소로, 이 주소를 통해 전송된 패킷은 물리적 네트워크를 거치지 않고 오직 NIC(Network Interface Card)만 거친다. 따라서 통신 환경 등을 고려할 필요가 없으며 물리적 네트워크를 사용하는 방식보다 더 빠르다[10]. 일반적으로 에지 컴퓨팅 플랫폼과 디바이스가 같은 머신에서 동작하는 경우는 거의 없다. 따라서 데이터 수집 머신과 Image to Array 기법이 수행되는 머신, 에지 컴퓨팅 플랫폼이 동작하는 머신 등이 논리적으로 분할되어 있으므로 루프백 주소를 사용하지 못한다. 실험은 루프백 주소를 사용함으로써, 가장 이상적인 통신 환경과 논리적으로 모두 같은 머신에서 위 과정이 수행되는 형태이다.

두 번째는 데이터의 크기이다. 데이터의 크기는 변화 배율만 보면 큰 폭으로 증가하였다고 할 수 있다. 그러나 그 크기 자체는 일반적인 대역폭에 비하면 크다고 하기 어렵다. 즉 통신 과정에서, 몇백 kilobyte 단위의 데이터가 전송되는 것이나 몇 megabyte 단위의 데이터가 전송되는 것이나 동일한 부하의 작업이라는 것이다.

이러한 이유들로, Image to Array 기법을 통해 데이터 크기가 증가하더라도, 전체 서비스 시간에 큰 영향을 미치지 않은 것으로 추정된다.

6. 결론 및 향후 연구

본 논문에서는 이미지를 문자열 형태의 다차원 배열로 변환하여 에지 컴퓨팅 플랫폼에 저장하는 방안을 기술하였다. Image to Array 기법은 이미지를 문자열 형태의 3차원 배열 형식으로 에지 컴퓨팅 플랫폼 내부에 저장한다. 이 데이터는 다차원 배열 형식이기 때문에, 이미지 분석, 조작 등 다양한 용도로 쉽게 접근할 수 있고 원하는 목적에 따라 가공 및 처리할 수 있다. 또한 배열을 사용하는 다른 이미지 처리 도구나 라이브러리와의 통합에 용이하다. 그러나 원본 바이너리 이미지에 비해 크기가 증가하는데, 일반적으로 데이터 크기의 증가는 저장 및 전송 효율성 저하와 관련이 있기 때문에 실험을 통해 얼마나 치명적일지 확인하고자 하였다. 본 논문에서는 이러한 단점이 시간 복잡도 측면에서 얼마나 큰 영향을 미칠지, 3대의 서로 다른 머신에서 Ubuntu 시스템을 구축하고 10개의 2017 validation COCO dataset의 데이터를 이용하여 소요 시간을 측정하였다.

실험군은 제시한 방안이 적용된 예시 파이프라인이었으며, 대조군에 비해 더 많은 추가 절차가 존재한다. 데이터 크기의 증가 및 추가 절차로 인해 실험군이 시간 복잡도 측면에서 낮은 성능을 보일 것이라는 일반적인 예측과 달리, 시간 복잡도 측면에서 큰 손실을 보이지 않았다. 이러한 결과의 원인으로 다음의 실험 환경 및 한계를 지목하였다. 1. 루프백 주소를 활용하여 실험이 수행된 점, 2. 원본 데이터의 크기가 작은 점이다. 이러한 단점에도 불구하고, 이미지를 에지 컴퓨팅 플랫폼 내부에 다차원 배열 형태로 저장함으로써 높은 확장성을 제공한다는 점에서 제시한 Image to Array 기법의 활용성은 높을 것으로 사료된다.

본 논문에서는 통신 환경과 그 변수들을 고려하지 않았으며 모두 루프백 주소를 이용해 이루어졌다. 또한 본 논문에서는 통신 프로토콜로 REST API만 사용하였다. tensorflow serving은 gRPC를 지원하며, gRPC는 REST에 비해 대용량 전송에 유리하다. 본 논문의 실험은 한 개의 이미지에 대해 객체탐지가 수행됨을 가정하며, 모든 과정이 논리적으로 분할되지 않고 한 개의 머신에서 수행되었다. 또한, 본 논문에서는 실시간 객체탐지 상황을 가정하고 실험을 진행하였다. 실시간성을 충족하기 위해서는 일괄 처리를 사용하지 않으므로 일괄 처리 시의 성능은 고려하지 않았다. 다만 Image to Array 기법을 적용하면 이미지의 크기가 증가하는 것으로 미루어 볼 때, 일괄 처리 시 성능의 저하가 있을 것으로 예측할 수 있다. 차후 연구에서 batch size를 조정하여 일괄 처리 시에 실제로 성능 저하가 있는지, 있다면 그 정도가 얼마인지 등에 대한 분석을 할 예정이다. 또한 다양한 통신 프로토콜 및 통신 환경을 고려하며, 각 작동이 논리적으로 분할된 상태에서 Image to Array 기법의 실용성을 평가할 예정이다.

References

[1] EdgeX Foundry [Internet]. https://docs.edgexfoundry.org/2.2/.

[2] Message Queue vs Message Bus -- what are the differences? [Internet], https://stackoverflow.com/questions/7793927/message-queue-vs-message-bus-what-are-the-differences.

[3] EdgeX Fondry North-South Messaging [Internet], https://docs.edgexfoundry.org/2.2/design/adr/0023-North-South-Messaging/.

[4] Pixels, Arrays, and Images [Internet], https://levelup. gitconnected.com/pixels-arrays-and-images-ef3f03638fe7.

[5] Software Flow, in USB Camera Device Service [Internet], https://github.com/edgexfoundry/device-usb-camera.

[6] Intel AI Object Detection with Tensorflow Serving on CPU [Internet], https://github.com/IntelAI/models/blob/master/docs/ object_detection/tensorflow_serving/Tutorial.md

[7] Tensorflow serving [Internet], https://www.tensorflow. org/tfx/guide/serving

[8] Flask [Internet], https://flask.palletsprojects.com/en/2.2.x/.

[9] What Is Restful API? [Internet], https://aws.amazon.com/ko/what-is/restful-api/.

[10] Ray Hunt, "Transmission Control Protocol/Internet Protocol (TCP/IP)," in Encyclopedia of Information Systems, pp. 489-510, 2003.

장 신 원

https://orcid.org/0000-0001-5933-1730
e-mail : largestone23@gmail.com
2023년 총신대학교 (학사)
2023년 ~ 현 재 대전대학교 AI연구실 연구원
관심분야 : 객체인식, 자원제약 환경에서의 AI 서비스

홍 용 근

https://orcid.org/0000-0003-2974-3820
e-mail : yghong@dju.kr
1997년 경북대학교 컴퓨터공학과(학사)
1999년 경북대학교 컴퓨터공학과(석사)
2013년 경북대학교 컴퓨터공학과(박사)
2001년 ~ 2020년 한국전자통신연구원 실장
2021년 ~ 현 재 대전대학교 AI융합학과 교수
관심분야 : IoT, 지능형 에지 컴퓨팅, 추론 시스템