KIPS Trans. Comp. and Comm. Sys.
Vol.12, No.12 pp.363~370
ISSN: 2287-5891 (Print), ISSN 2734-049X (Online)

https://doi.org/10.3745/KTCCS.2023.12.12.363

A Study on the Application of Task Offloading for Real-Time Object Detection in Resource-Constrained Devices

Jang Shin Won† · Yong-Geun Hong††

ABSTRACT

Object detection technology that accurately recognizes the road and surrounding conditions is a key technology in the field of autonomous driving. In the field of autonomous driving, object detection technology requires real-time performance as well as accuracy of inference services. Task offloading technology should be utilized to apply object detection technology for accuracy and real-time on resource-constrained devices rather than high-performance machines. In this paper, experiments such as performance comparison of task offloading, performance comparison according to input image resolution, and performance comparison according to camera object resolution were conducted and the results were analyzed in relation to the application of task offloading for real-time object detection of autonomous driving in resource-constrained devices. In this experiment, the low-resolution image could derive performance improvement through the application of the task offloading structure, which met the real-time requirements of autonomous driving. The high-resolution image did not meet the real-time requirements for autonomous driving due to the increase in communication time, although there was an improvement in performance. Through these experiments, it was confirmed that object recognition in autonomous driving affects various conditions such as input images and communication environments along with the object recognition model used.

Keywords : Autonomous Driving, Object Detection, Task Offloading, Resource-constrained Device, Real-time

자원 제약적 기기에서 자율주행의 실시간 객체탐지를 위한 태스크 오프로딩 적용에 관한 연구

장 신 원† · 홍 용 근††

요 약

도로와 주변의 상황을 정확히 인지하는 객체탐지 기술은 자율주행 분야에 핵심적인 기술이다. 자율주행 분야에 객체탐지 기술은 추론 서비스의 정확도와 함께 실시간성도 요구된다. 고성능 머신이 아닌 자원제약 기기에서 정확도와 함께 실시간성을 위한 객체탐지 기술을 적용하기 위해서는 태스크 오프로딩 기술을 활용해야 한다. 본 논문에서는 자원 제약적 기기에서 자율주행의 실시간 객체탐지를 위한 태스크 오프로딩 적용과 관련하여 태스크 오프로딩의 성능 비교, 입력 이미지 해상도에 따른 성능 비교, 카메라 객체 해상도에 따른 성능 비교 등의 실험을 수행하고 결과를 분석하였다. 본 실험에서 낮은 해상도의 이미지는 태스크 오프로딩 구조의 적용을 통하여 성능 개선을 도출할 수 있었고, 이는 자율주행의 실시간 기준을 충족하였다. 높은 해상도의 이미지는 성능 개선은 있었으나 통신 시간의 증가에 따른 이유로 자율 주행의 실시간 기준을 충족하지 못하였다. 이러한 실험을 통해 자율주행에서의 객체인식은 사용하는 객체인식 모델과 함께 입력 이미지, 통신 환경 등의 다양한 조건이 영향을 미친다는 것을 확인할 수 있었다.

키워드 : 자율주행, 객체인식, 태스크 오프로딩, 자원 제약적 기기, 실시간성

1. 서 론

사람의 개입 없이도 차량이 스스로 목적지까지 이동하는 자율주행 기술은 전체 5단계 중 현재 2단계를 거쳐서 3단계로 발전하고 있다. 이러한 자율주행 기술의 3단계, 4단계, 5단계로 발전하기 위해서 반드시 필요한 기술이 도로와 주변의 상황을 정확히 인지하는 기술이며, 이러한 기술에는 대표적으로 딥러닝 기반의 객체인식(Object Detection) 기술이 있다. YOLO (You Only Look Once), SSD(Single Shot MultiBox Detector), RFCN(Region-based Fully Convolutional Networks) 등이 대표적인 객체인식 기술이다. 이러한 객체인식 기술을 자율주행 분야에 사용하기 위해서는 단순히 추론의 정확도 뿐만 아니라 추론 시간도 매우 중요하다. 도로에 있는 객체를 인식하는데 실시간으로 탐지하지 못한다면 아무리 정확도가 높다고 하더

※ 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 : November 7, 2023
Accepted : November 17, 2023
* Corresponding Author : Yong-Geun Hong(yghong@dju.kr)

라도 사용할 수가 없기 때문이다. 높은 추론 정확도와 실시간 추론 조건을 달성하기 위해 가장 쉬운 방법은 성능이 높은 머신을 사용하는 것이다. 그러나 이는 비용 인상을 유발하므로 해당 서비스에 대한 시장성(접근성, 대중성 등)이 떨어질 우려가 있다.

이와 같이 보급성이 떨어지는 것을 방지하고 대중성을 높이기 위해서는, (고성능의 머신을 활용하여 객체인식 기술을 제공하기 보다)자원 제약적인 기기에서 태스크 오프로딩 방식을 적용하는 것이 합리적인 접근 방법일 수 있다.

태스크 오프로딩(Task Offloading)은 컴퓨팅 작업을 다른 처리장치 또는 시스템으로 전달하여 수행하도록 하는 기술 또는 구조를 의미한다. 태스크 오프로딩은 모바일 장치 등 연산 능력과 저장 공간, 전력 등의 자원이 제한되는 상황에서 컴퓨팅 자원이 풍부한 플랫폼으로 작업을 전달함으로써, 자원이 부족한 장치의 성능 한계를 넘어서는 작업을 빠르게 처리할 수 있다[1].

따라서 본 논문에서는 자원 제약적 기기에서 자율주행의 실시간 객체탐지를 위한 태스크 오프로딩 적용에 관한 연구 결과를 소개하고자 한다. 본 논문의 구성은 다음과 같다. 먼저 2장에서는 관련 연구로서 객체탐지 알고리즘 및 모델, 자율주행 자동차의 객체탐지, 태스크 오프로딩, AI 서비스 제공을 위한 프레임워크 등에 대하여 설명한다. 3장에서는 본 논문에서 실험한 자원 제약적 기기에서 실시간 객체탐지를 위한 태스크 오프로딩 실험환경과 실험 절차에 대해서 설명한다. 4장에서는 크게 3개의 실험 조건(태스크 오프로딩의 성능 향상 확인, 입력 이미지 해상도에 따른 성능 비교 확인, 카메라 객체 해상도에 따른 성능 비교 확인)으로 나누어서 실험한 결과를 설명하고 그러한 결과에 대한 고찰을 하였다. 마지막 5장에서는 결론 및 향후 연구 방향에 대하여 기술한다.

2. 관련 연구

2.1 객체탐지 알고리즘 및 모델

본 논문에서는 실시간 인공지능 서비스의 한 예로 자율주행을 들었다. 자율주행에 사용되는 대표적인 인공지능 기술은 객체탐지이다. 객체탐지는 주어진 이미지에서 물체를 분류(Classification)하고, 해당 물체의 위치를 특정 짓는(Localization) 기술이다. 딥러닝 기반 객체탐지 방식은 크게 두 가지로 분류되는데, Region Proposal을 통한 위치 특정 후 물체 분류가 순차적으로 이루어지는 방식을 two-stage, 위치 특정 및 물체 분류가 동시에 수행되는 방식을 one-stage 방식이라고 한다[2].

각 방식은 장단점이 존재한다. Two-stage의 경우 one-stage보다 더 높은 정확도를 가진다. 그러나 속도가 느려 실시간 AI 서비스 등에는 적용하기 어렵다. 반면 one-stage의 경우 정확도는 비교적 떨어지지만, 높은 추론 속도를 보인다. 따라서 one-stage 객체탐지 방식은 자율주행을 비롯한 실시간 AI 서비스 적용에 적합하다.

본 논문의 실험은 자율주행 상황을 가정한다. 따라서 one-stage 방식의 모델을 채택하는 것이 바람직하며, one-stage의 아키텍처 중 하나인 SSD-MobileNet을 채용하였다. 본 논문은 AI 서비스 제공 및 운영에 관해 연구를 진행하는 것이므로, 직접 학습은 하지 않았으며 Intel AI[3]에서 제공하는 coco dataset으로 사전 훈련된 SSD-MobileNet v1을 사용한다.

2.2 자율주행 자동차의 객체탐지

자율주행 자동차에는 다양한 센서들이 활용되며, 이 데이터들을 종합하여 최종 판단을 내린다. 이 중 카메라 데이터 기반 객체탐지는 고가 장비인 LiDAR, RADAR 등이 해결하지 못하는 부분을 해결한다. 카메라 기반 객체탐지는 다양한 업무를 수행하는데, 주행차로 유지 및 차선 변경을 위한 차선 인식 외에도 도로의 합류 지점 및 분기 지점에 대한 인식과 주행 지원, 보행자나 표지판, 도로변의 주정차 차량 등의 형상 정보를 구분하여 안전한 주행 및 주차 보조를 지원한다[4].

이러한 주행 보조 및 자율주행이 제대로 수행되기 위해서는 카메라 기반 객체탐지가 실시간으로 이루어져야 할 필요가 있다. 자율주행에서 객체탐지가 실시간으로 간주하기 위한 최소 FPS(Frame Per Seconds)는 30 FPS이다[5]. 객체를 탐지하는 정확도가 아무리 높더라도 해당 기준치를 만족시키지 못하면 자율주행에 사용할 수 없다.

2.3 AI 서비스 제공을 위한 프레임워크

Tensorflow serving[6]은 AI 기반 서비스 제공을 위한 시스템이다. Tensorflow serving의 가장 큰 장점은 동일한 서버 아키텍쳐와 API를 유지한 채로 다양한 사전 학습 모델을 쉽게 배포할 수 있다는 점이다[2]. Tensorflow serving을 통해 모델을 배포하기 위해선 해당 사전 학습 모델의 saved_model 포맷이 필요하다. 이를 해당 폴더에 넣고 Tensorflow serving container에 바인드 마운트(Bind mount)과정만 수행해주면, 추가적인 수행 없이 Tensorflow serving 서버가 생성된다. 엔드 포인트로는 gRPC와 REST를 제공한다. 이 중 gRPC는 프로토버프(Protobuf)를 통한 직렬화를 사용하여 적은 오버헤드 및 빠르고 효율적인 통신을 제공한다[7].

이러한 Tensorflow serving은 주로 가상화 도구인 Docker를 사용하여 수행된다. 따라서 높은 유연성 및 개발 배포의 편리성과 신속성을 보장받을 수 있다.

3. 자원 제약적 기기에서 실시간 객체탐지를 위한 태스크 오프로딩 실험

3.1 연구 환경

본 절은 태스크 오프로딩 구조를 적용하고 실험을 수행할 머신들의 성능과 사용된 소프트웨어 버전 정보를 기술한다.

1) 실험에 사용되는 하드웨어 사양

자원 제약적 기기로는 한백전자사의 AIoT AutoCar Ⅱ로

Table 1. Comparison Between HW Performance

	Machine A	Machine B
CPU	Quad-core ARM A57 @ 1.43 GHz	Intel Core i7-11700K
GPU	NVIDIA Maxwell architecture with 128 NVIDIA CUDA® cores	GeForce RTX 3060 Lite Hash Rate
RAM	4GB	32GB
Geekbench score	742	10,647

Nvidia Jetson Nano가 탑재되어 있는 머신으로 Geekbench Multi-Core점수는 742점이다(머신 A). 그리고 자원 제약적 기기와 대비하여 실험할 고성능 머신은 Intel Core i7-11700K와 GeForce RTX 3060이 탑재된 머신으로 Geekbench Multi-Core 점수는 10,647점이다(머신 B).

2) 실험에 사용되는 소프트웨어 사양

실험 수행에 사용된 주요 소프트웨어 및 패키지들의 버전 정보는 다음의 Table 2와 같다.

3) 실험에 사용되는 입력 이미지 정보

객체탐지 학습 모델의 추론 속도에 영향을 줄 수 있는 요소들에 대해 명확히 밝혀진 바는 없다. 따라서 측정하고자 하는 변수 외에 나머지 변수들을 통제하고자 이미지 한 장을 지정하여 모든 실험을 수행하였다. 사용한 이미지는 UC 버클리 인공지능 실험실(BAIR)에서 제공하는 BDD100K 데이터셋[8] 중 하나로 720p 해상도의 이미지이다.

Table 2. Comparison Between SW Specification

	Machine A	Machine B
OS	Ubuntu 18.04	Ubuntu 20.04
Docker version	-	23.0.1
Python version	3.6	3.8
Tensorflow version	gpu 1.31.1+nv19.5	2.9.1
Tensorflow serving image	-	2.12.1-gpu
tensorflow object detection api	0.1.1	0.1.1
gRPC version	1.34.1	1.37.0
JetPack version	4.5-b129	-
Pre-trained Object detection model	SSD-MobileNet v1 (coco)	

Fig. 1. Input Image to be Experimented

본 실험에서 연속적인 동영상이나 여러 장의 서로 다른 이미지를 사용하지 않고 한 장의 이미지를 사용하는 이유는 각 동영상 프레임 이미지별로 다른 특성을 갖고 있는 이미지 시퀀스보다는 동일한 특성을 갖고 있는 이미지 한 장을 사용하는 것이 실험의 결과에 영향을 미치는 다양한 변수들에 대한 통제를 하기 위해서이다. 실제로 동영상도 연속적인 이미지의 시퀀스이고, 이미지를 연속적으로 처리하면 동영상 처리와 같게 된다. 영상처리가 코드 상에서 처리되는 원리도 동영상에서 이미지를 한 장씩 불러온 뒤 이에 대한 객체탐지를 진행하고 그 결과를 끊임없이 보여주는 방식으로 이루어진다. 예를 들어 30 FPS의 카메라를 사용하면 초당 이미지가 30장씩 들어온다는 의미이다. 사전 훈련 모델이 이를 장당 약 33ms 내로 처리하면 30FPS를 유지하는 것이고, 처리하지 못한다면 그 이하의 성능을 보여주는 것이다.

3.2 실시간 객체탐지를 위한 태스크 오프로딩 실험

본 절에서는 태스크 오프로딩 기법이 어떻게 구현되었는지 설명하고, 세부 작동 및 시간 측정에 포함되는 작동에 대하여 기술한다.

1) 태스크 오프로딩 구조 설계

태스크 오프로딩 구조는 크게 두 요소로 구성된다. 컴퓨팅 작업 요청을 하는 클라이언트와 요청을 처리하는 서버이다. 본 논문에서는 머신 A를 클라이언트로, 머신 B를 서버로 설정하였다. 클라이언트는 내장된 카메라를 통해 이미지 데이터를 획득하며, 이를 Tesnorflow serving 및 사전 학습 모델에서 요구하는 형태로 포맷팅(formatting) 후 서버로 처리 요청한

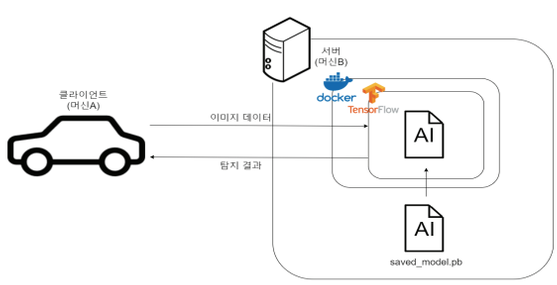

Fig. 2. Architecture of Proposed Task Offloading

다. 서버는 이 요청을 수신 후 사전 학습 모델을 통해 추론하고 그 결과물을 클라이언트로 반환한다. 클라이언트는 이 정보를 기반으로 자율주행을 수행한다. 데이터 획득 및 포맷팅, 추론 요청은 파이썬 환경에서 수행되었다. 이 논문에서는 이렇게 한백전자사의 자율주행 키트인 머신 A와 서버 역할을 수행하는 머신 B로 구성된 구조를 kit-server라고 지칭한다.

2) 클라이언트 및 서버의 상세 동작

태스크 오프로딩 구조를 사용한 경우, 클라이언트의 주요 작동 알고리즘은 [알고리즘 1]과 같다.

[알고리즘 1] 태스크 오프로딩 시 클라이언트의 작동

Data: 이미지
Result: 탐지 결과
1 이미지 파일을 BGR 순서의 3차원 숫자 배열로 읽어들인다.
2 이미지 배열을 RGB 순서로 재배열한다.
3 이미지 배열을 4차원으로 확장시킨다.
4 서버와의 gRPC 통신 준비를 한다.
5 서버 및 사전 학습 모델의 입력 형태에 맞추어 요청서를 작성한다.
6 추론 요청 후 결과를 반환받는다.

태스크 오프로딩 구조를 사용하지 않고 로컬에서 모두 수행 시, 주요 작동 알고리즘은 [알고리즘 2]와 같다.

[알고리즘 2] 로컬 수행 시 작동

Data: 이미지
Result: 탐지 결과
1 이미지 파일을 BGR 순서의 3차원 숫자 배열로 읽어들인다.
2 이미지 배열을 RGB 순서로 재배열한다.
3 이미지 배열을 4차원으로 확장시킨다.
4 이미지 배열을 텐서 프로토로 변환한다.
5 사전 학습 모델을 통해 추론한다.

3) 매개변수 정의

본 항에서는 태스크 오프로딩 구조의 성능 측정 및 실험 결과 파악을 위한 매개변수를 정의한다.

D_{px}은 입력 이미지의 픽셀 수로, 이미지의 너비와 이미지의 높이를 곱해 얻을 수 있다. [알고리즘 1]에서 t_{fmt}는 1~5줄의 수행에 걸리는 시간이며, [알고리즘 2]에서 t_{fmt}는 1~4 줄의 수행에 걸리는 시간이다. t_{docker}는 [알고리즘 1]의 6번째 줄의 수행에 걸리는 시간이다. 이는 서버와의 통신 속도가 포함된 시간으로 다음과 같은 표현이 가능하다.

Table 3. Definition of Parameters to Measure Performance

Symbol	Definition
D_{px}	Number of pixels in the input image [n x 10^5]
t_{fmt}	Time spent formatting an input image [ms]
t_{trans}	Time spent communicating with the server [ms]
t_{inf}	Time spent inference learning model [ms]
td_{ocker}	Time spent requesting inference from server and receiving results [ms]
tt_{otal}	Total Time Required [ms]

$$t_{docker} = t_{trans} + t_{inf}$$

t_{total}은 모든 과정 수행을 수행하는 동안 걸린 시간이다. t_{total}을 통해 FPS를 구할 수 있다.

$$FPS = 1{,}000 / t_{total}$$

4. 자원 제약적 기기에서 실시간 객체탐지를 위한 태스크 오프로딩 실험 결과 및 고찰

4.1 실험 1 (kit(local) / kit-server 비교)

실험 1은 태스크 오프로딩 구조의 적용이 얼마나 많은 성능 향상을 가져오는지 확인하기 위한 실험이다. D_{px} = 0.9 * 10^5일 때, 태스크 오프로딩을 적용하지 않고, 클라이언트에서 모든 업무를 수행하는 경우(kit(local))와 태스크 오프로딩을 적용하여 클라이언트와 서버가 업무를 분담하여 수행하는 경우(kit-server)를 비교한다.

Table 4에서 kit(local)의 경우 FPS는 4.6이 나온다. 이는 객체탐지 기반 자율주행을 위한 기준 FPS의 1/6 수준으로 자율주행을 수행할 수 없는 수치이다. FPS는 t_{total}로 결정되고, t_{total}은 t_{fmt}와 t_{inf}(또는 t_{docker})의 합이다. t_{total}은 215.7이며 t_{inf}는 211.3이다. t_{total} 중 t_{inf}의 비율은 0.979이며 이는 총 소요시간의 대부분이 추론에 의한 것임을 의미한다.

반면 Table 4에서 kit-server의 경우 속도는 41.8 FPS가 나온다. 이는 객체탐지 기반 자율주행을 위한 기준 FPS를 만족하는 수준이다. 태스크 오프로딩 구조가 적용된 경우 t_{total}은 t_{fmt}와 t_{docker}의 합이다. t_{total}은 23.8이며 t_{docker}는17.6이다. t_{total}에 대한 t_{docker}의 비율은 0.73이다. 이때, t_{docker}는 t_{trans}와 t_{inf}의 합인데 이와 관련한 내용은 실험 2에서 서술할 예정이다. 결론적으로 t_{trans}가 포함되어 있음에도 기준 FPS를 충족할 수 있었다.

4.2 실험 2 (입력 이미지 해상도에 따른 성능 비교)

객체탐지 기반 자율주행을 수행하는 데 전제 조건이 30 FPS임은 맞지만, 정확도 또한 동등하게 중요하다. 정확도에 가장 많은 영향을 주는 요소는 객체탐지 모델이다. 일반적으로 객체탐지 two-stage 모델이 객체탐지 one-stage 모델보다 더 높은 정확도를 보이며, 모델의 hidden layer(또는 파라미터)가 많을수록 정확도가 높다. 그러나 이러한 요소들은 객체탐지 모델의 추론 속도와 관련되어 있다. 객체탐지 two-stage 모델은 one-stage 모델보다 느리며, 객체탐지 모델의 hidden

Table 4. Comparison between kit(local) vs. kit-server

	kit(local)	kit-server
t_{fmt}	2.7	3.9
t_{inf} / t_{docker}	211.3	17.6
t_{total}	215.7	23.8
FPS	4.6	41.8

D_{px} = 0.9 x 10^5

Table 5. Comparison between kit(local) vs. kit-server vs. server(local) with D_{px} = 0.9 x 10^5

	kit(local)	kit-server	server(local)
t_{fmt}	2.7	3.9	0.2
t_{inf} / t_{docker}	211.3	17.6	7.2
t_{total}	215.7	23.8	7.5
FPS	4.6	41.8	132.9

D_{px} = 0.9 × 10^5

Table 6. Comparison between kit(local) vs. kit-server vs. server(local) with D_{px} = 9.216 x 10^5

	kit(local)	kit-server	server(local)
t_{fmt}	5.31	20.5	0.8
t_{inf} / t_{docker}	272.7	52.5	7.3
t_{total}	279.9	78.6	8.2
FPS	3.5	12.7	121

D_{px} = 9.216 × 10^5

layer(또는 파라미터)가 많을수록 속도는 느려진다. 즉, 정확도와 속도는 트레이드 오프(trade-off)관계이다.

결과적으로 전제 조건인 30 FPS를 충족시키기 위해서는 특정 객체탐지 모델이 강제될 수밖에 없으며, 객체탐지 모델 관련 부분에서 선택지가 거의 없다는 것을 의미한다.

객체탐지 추론 서비스의 정확도 개선을 위해 선택할 수 있는 두 번째 옵션은 입력 이미지의 해상도이다. 정확도는 이미지 해상도에 크게 의존한다[9]. 실험 1은 D_{px} = 0.9 * 10^5 (300×300)의 이미지를 사용해 수행된 것으로, 정확도를 보장하는 해상도라고 하기 힘들다. 따라서 실험 2에서는 입력 이미지의 D_{px}을 9.216 * 10^5(1280x720)와 20.736 * 10^5 (1920×1080)으로 증가시킨 후, 클라이언트의 로컬 수행 결과와(kit (local)) 태스크 오프로딩을 적용한 수행 결과(kit- server), 추가로 서버에서 로컬로 수행한 결과(server(local))를 사용한다. 로컬에서 객체탐지를 수행하는 경우, 네트워크 통신을 하지 않으므로 t_{inf}를 구할 수 있다. t_{docker}와 t_{trans} 및 t_{inf}의 관계를 이용하여 t_{trans}를 구한다.

Table 5는 D_{px} = 0.9 * 10^5일 때, kit(local)과 kit-server 및 server(local)의 비교 결과이다. 실험 1과 동일한 내용으로, 태스크 오프로딩 구조를 적용하여 자율주행 기준인 30 FPS 이상을 맞출 수 있었다. 이때, kit(local)은 총 소요시간의 대부분이 객체탐지 추론(t_{inf})에 의한 것이었다. kit-server의 경우도 비슷하게 t_{docker}가 총 소요시간의 많은 부분을 차지하지만, 이것이 추론에 의한 것으로 판단하기는 이르다. t_{docker}는 t_{trans}와 t_{inf} 의 합이기 때문이다. t_{trans}와 t_{inf}의 값을 구하기 위해 서버의 로컬에서 실험을 진행한 결과, server(local)의 t_{inf}는 7.3이 나왔다. 따라서 다음의 식을 통해 kit-server의 t_{trans}를 유추할 수 있다.

$$17.6(t_{docker}) - 7.3(t_{inf}) = 10.3(t_{trans})$$

t_{trans}는 10.3이 나온다. 이는 t_{inf}보다 더 높은 값이며, t_{total}에 대한 그 비율은 0.58으로 t_{total}에 대한 t_{inf}의 비율인 0.41보다 높다.

Table 6은 D_{px} = 9.216 × 10^5일 때 kit(local)과 kit-server 및 server(local)의 비교 결과이다. kit(local)의 경우 FPS는 3.5, kit-server의 경우 FPS는 12.7이 나왔다. 두 항목 모두 Table 5보다 FPS가 감소하였으며, 태스크 오프로딩을 적용하였음에도 객체탐지 기반 자율주행의 기준 FPS에 못 미치는 것을 확인할 수 있다. 반면 이미지에서 검출된 객체의 개수는 5개에서 6개로 정확도의 개선이 있었음을 확인할 수 있었다.

Table 6에서 이미지의 크기(D_{px})는 Table 5에 사용된 이미지에 비해 약 10배 증가하였다. [9]에 따르면 해상도와 추론 시간은 양의 상관관계에 있으므로 FPS 감소의 주 원인이 해상도 증가에 따른 t_{inf} 증가라고 예측할 수도 있다.

kit(local)의 경우 t_{inf}는 272.7로 D_{px} = 0.9 * 10^5일 때의 t_{inf}(211.3) 비해 약 29% 증가하였다. kit(local)은 객체탐지를 로컬에서 수행하므로, 통신 등의 변수 개입이 최소화된다. 또한 실험 1에서 확인한 대로, 추론 시간이 객체탐지 서비스 총 소요 시간의 대부분을 차지한다. 따라서 해상도의 증가가 추론 시간의 증가를 끌어냈고, 결과적으로 FPS가 감소하였다고 할 수 있다.

kit-server의 경우 t_{inf}를 직접적으로 확인할 수는 없지만, server(local)의 결과를 통해 D_{px} = 9.216 * 10^5일 때의 t_{trans}와 t_{inf}를 유추할 수 있다. t_{docker} = t_{trans} + t_{inf}이고, server(local)을 통해 t_{inf}를 알 수 있으므로, t_{trans}를 구할 수 있다.

$$78.6(t_{docker}) - 7.3(t_{inf}) = 71.3(t_{tans})$$

kit-server의 경우 [9]과는 다소 다른 양상을 보인다. Table 5에 사용된 이미지에 비해 D_{px}이 약 10배 증가하였음에도 t_{inf}는 7.3으로 D_{px} = 0.9 * 10^5일 때의 t_{inf}(7.2) 대비 1.3% 증가하였다. 대신 t_{trans}의 증가량이 많은데, t_{trans}는 71.3으로 D_{px} = 0.9 * 10^5일 때의 t_{trans}(10.3) 대비 713% 증가하였다.

이와 비슷한 현상은 D_{px} = 20.736 * 10^5인 경우에도 확인할 수 있다. Table 7은 D_{px} = 20.736 * 10^5일 때 kit(local), kit-server와 server(local)에 대한 수행 결과를 나타낸 표이다. D_{px} = 0.9 * 10^5일 때의 t_{inf}(211.3)일 때에 비해 kit(local)과 kit-server 둘 다 FPS가 감소하였으며, D_{px} = 9.216 * 10^5일 때

Table 7. Comparison between kit(local) vs. kit-server vs. server(local) with D_{px} = 20.736 x 10^5

	kit(local)	kit-server	server(local)
t_{fmt}	8.6	38.1	1.7
t_{inf} / t_{docker}	303.3	104.6	7.4
t_{total}	313.8	151.3	9.2
FPS	3.1	6.6	108.3

D_{px} = 20.736 × 10^5

에 비해서도 kit(local)과 kit-server 모두 FPS가 감소하였다.

kit(local)의 경우 t_{inf}는 303.3으로, D_{px} = 0.9 * 10^5일 때의 t_{inf}(211.3) 비해 약 43.5% 증가하였으며, D_{px} = 9.216 * 10^5일 때의 t_{inf}(272.7) 비해 11.2% 증가하였다. kit(local)은 통신 등의 변수 개입이 최소화되었고, 추론 시간이 총 소요시간의 대부분을 차지하므로 FPS 감소의 주요인은 객체탐지 추론시간이라 볼 수 있다. 즉, Table 5와 동일하게 해상도의 증가에 따른 추론 시간 증가로 FPS가 감소했다고 볼 수 있다.

반면 kit-server의 경우 t_{inf}는 7.4이다. 이는 D_{px} = 0.9 * 10^5일 때의 t_{inf}(7.2)에 비해 2.7% 증가한 수치이며, D_{px} = 9.216 * 10^5의 t_{inf}(7.3)에 비해 1.3% 증가한 수치이다. server(local)의 결과를 사용하여 D_{px} = 20.736 * 10^5일 경우의 kit-server의 t_{trans}는 다음의 식을 통해 유추할 수 있다.

$$104.6(t_{docker}) - 7.4(t_{inf}) = 97.2(t_{trans})$$

Table 6에서 확인한 결과와 비슷하게, t_{inf}의 증가보다는 t_{trans} 증가가 크다. t_{trans}는 97.2로 D_{px} = 0.9 * 10^5일 때의 t_{trans}(10.3)의 943% 수준이며, D_{px} = 9.216 * 10^5일 때의 t_{trans}(71.3)의 136% 수준이다. t_{trans}가 t_{total} 중 차지하는 비도 0.64로 높은 것을 확인할 수 있다.

4.3 실험 3 (카메라 객체 해상도에 따른 성능 비교)

실험을 수행할 때, 실제 자율주행 상황과 최대한 유사하게 설정하였는데 그 설정 중 하나는 카메라의 사용 여부이다. 카메라를 통한 이미지 데이터 획득은 객체탐지가 로컬에서 수행되든, 태스크 오프로딩 구조의 적용에 의해 서버에서 수행되든 상관없이 클라이언트에서 이루어져야 하는 작업이다. 비록 실험은 이미지의 특징이 결과에 영향을 미치는 것을 방지하고자 특정 이미지를 지정하여 수행하였으나, 입력 이미지의 해상도와 동일하게 카메라 객체를 생성하여 실제 자율주행과 유사하도록 실험을 수행하였다. 그러나 실험을 하면서, 이미지의 해상도가 아닌 카메라 객체의 해상도가 객체탐지 추론 시간에 영향을 주는 것처럼 보이는 현상을 확인하였고, 이 현상을 분석하기 위해 다음과 같이 실험을 진행하였다.

Table 8, Table 9, Table 10의 실험은 태스크 오프로딩 구조를 적용하지 않고 kit의 로컬에서 수행하였으며, 이미지 데이터의 해상도가 각각 D_{px} = 0.9 * 10^5, D_{px} = 9.216 * 10^5, D_{px} = 20.736 * 10^5일 때, 동일한 해상도의 카메라 객체를 사용한 kit(local, cam)과 카메라 객체를 사용하지 않은 kit(local, no-cam)의 t_{fmt}, t_{inf}, t_{total}, FPS를 비교한 결과이다. 추가로 객체탐지 서비스 수행 도중 확인한 python 프로세스의 CPU의 사용량을 비교한다.

Table 8은 D_{px} = 0.9 * 10^5일 때 kit(local, cam)과 kit(local, no-cam)의 수행 결과를 비교한 표이다. kit(local, cam)의 t_{inf}는 kit(local, no-cam)대비 43.9만큼 더 높다. CPU 사용량은 1%p 차이로 거의 비슷하다. Table 9는 D_{px} = 9.216 * 10^5일 때 kit(local, cam)과 kit(local, no-cam)을 비교한 표이다.

Table 8. Comparison between kit(local, cam) vs. kit(local, no-cam) with D_{px} = 0.9 x 10^5

	kit(local, cam)	server(local, no-cam)
t_{fmt}	2.7	1.9
t_{inf} / t_{docker}	211.3	167.4
t_{total}	215.7	170.7
FPS	4.6	5.8
CPU	172%	171%

D_{px} = 0.9 × 10^5

Table 9. Comparison between kit(local, cam) vs. kit(local, no-cam) with D_{px} = 9.216 x 10^5

	kit(local, cam)	server(local, no-cam)
t_{fmt}	5.3	3.4
t_{inf} / t_{docker}	272.7	169.9
t_{total}	279.9	174.1
FPS	3.5	5.74
CPU	202%	173%

D_{px} = 9.216 × 10^5

Table 10. Comparison between kit(local, cam) vs. kit(local, no-cam) with D_{px} = 20.736 x 10^5

	kit(local, cam)	server(local, no-cam)
t_{fmt}	8.6	6.2
t_{inf} / t_{docker}	303.3	174.9
t_{total}	313.8	181.9
FPS	3.1	5.49
CPU	246%	172%

D_{px} = 20.736 × 10^5

kit(local, cam)의 t_{inf}가 kit(local, no-cam)의 t_{inf}보다 102.8만큼 더 높다. CPU 사용량은 29%p 차이 난다. Table 10은 D_{px} = 20.736 * 10^5일 때 kit(local, cam)과 kit(local, no-cam)을 비교한 결과이다. kit(local, cam)의 t_{inf}가 kit(local, no-cam)의 t_{inf}보다 128.4만큼 더 높다. CPU 사용량은 약 74%p 차이 난다.

위 비교를 통해 카메라 객체의 사용 여부 및 그 해상도가 t_{inf}에 영향을 주는 것을 확인할 수 있었다.

반면 D_{px}로 표현되는 이미지의 해상도는 추론에 크게 영향을 주지 않음을 확인할 수 있다. kit(local, no-cam)에서 각 D_{px}의 t_{inf}는 167.3, 169.9, 174.9이며, D_{px} = 9.216 * 10^5의 t_{inf}는 D_{px} = 0.9 * 10^5 의 t_{inf}대비 1.5% 증가한 수치이고, D_{px} = 20.736 * 10^5의 t_{inf}는 D_{px} = 9.216 * 10^5의 t_{inf} 대비 2.9% 증가한 수치이다. 이는 server(local)의 D_{px} 별 t_{inf}의 증가 수치와 비슷한 결과이다.

이를 통해 실험 2에서 확인했던 kit(local)에서 해상도의 증가가 추론 시간의 증가를 유도했다는 점은 잘못된 표현임을 알 수 있었다. 입력 이미지의 해상도와 동일하게 카메라

객체를 생성하는 데, 카메라 객체의 해상도 증가는 더 많은 CPU를 사용하였고, 결과적으로 사전 학습 모델에 사용될 컴퓨팅 자원이 감소되어 추론 시간의 증가를 유발하였다고 추측할 수 있다.

4.4 각 실험 결과 고찰

본 절에서는 실험의 결과들을 간략하게 정리하고 고찰한다. 실험 1은 태스크 오프로딩 구조를 적용하고 그 성능을 측정 및 비교하기 위한 실험이었다. 태스크 오프로딩 없이 로컬에서 모든 작업을 수행할 경우 4.6 FPS가 나오며, 이는 객체탐지 기반 자율주행을 수행하기 어려운 FPS이다. 반면, 태스크 오프로딩를 적용하여 서버에서 객체탐지를 수행하고 로컬에서 나머지 업무를 수행할 경우 41.8 FPS가 나온다. 이는 객체탐지 기반 자율주행의 기준 FPS인 30 FPS를 충족하는 결과이다. 실험 1을 통해 자원 제약적인 디바이스에서 태스크 오프로딩 구조가 어느 정도 효용성을 보이는 것을 확인할 수 있었다.

실험 1은 $D_{px} = 0.9 * 10^5$ 이미지를 사용한 결과이다. [9]에 따르면 해상도와 정확도는 비례 관계에 있으며, 특정 구간에서 해상도 증가에 따른 정확도 증가가 급격히 일어나며 이후 완만해진다. 정확도 또한 자율주행에서 중요한 평가 지표이므로, 정확도를 높이기 위해 이미지 해상도를 증가시킨 경우 태스크 오프로딩 구조에 어떠한 영향이 있을지 확인하고자 실험 2를 진행하였다. 실험 2는 300×300, 1280×720, 1920×1080 해상도의 이미지를 사용하여 클라이언트의 로컬에서 업무를 수행한 경우, 서버에서 객체탐지를 수행하고 나머지 업무를 클라이언트에서 수행한 경우를 비교하였다. 두 경우 모두 해상도가 증가할수록 FPS가 감소하였으며, 1280×720 및 1920×1080 이미지의 경우 태스크 오프로딩을 적용하더라도 기준 FPS인 30 FPS에 도달하지 못하였다.

FPS 감소의 직접적인 원인이 무엇인지 파악하기 위해서 서버의 로컬 추론 시간을 구하고, 이를 네트워킹 통신 시간과 추론 시간이 합쳐진 결과에서 제함으로써 통신 시간과 추론 시간을 구분하였다. 이때 해상도 증가에 따른 추론 시간의 증가폭은 아주 낮은 반면, 해상도 증가에 따른 네트워킹 통신 지연시간의 증가폭은 아주 높았다. 또한 네트워킹 통신 지연 시간이 전체 서비스 시간에서 차지하는 비율을 확인하여 FPS 감소의 주 원인은 네트워킹 통신 지연시간임을 확인할 수 있었다.

결론적으로 실험 2를 통해 자율주행차량에서의 객체탐지 서비스 같은 실시간 인공지능 서비스 제공에 있어 사전 학습된 모델 자체의 추론 시간도 중요하지만, 통신 환경 또한 중요함을 확인할 수 있었다.

실험 3은 실험 2를 수행하던 도중 발생한 특정 현상에 관한 내용이다. 클라이언트로 사용한 머신은 Nvidia Jetson Nano 기반 자율주행 키트로 카메라가 탑재되어 있다. 실제 자율주행 상황과의 유사성을 위해 클라이언트의 로컬 수행 시마다 입력 이미지의 해상도와 동일한 크기로 카메라 객체를 생성하여 실험을 진행하였다. 실험 도중 이미지의 해상도가 추론 시간에 영향을 미치는 것이 아니라 이미지의 해상도를 따라 생성한 카메라가 추론 시간에 더 큰 영향을 미치는 것처럼 보이는 현상을 확인하였다. 이 현상을 확인하기 위해 카메라 객체를 해상도에 따라 생성 후 객체 탐지를 수행한 결과와 카메라 객체의 생성 없이 객체 탐지만 수행한 결과를 비교하였다. 결과적으로 사전 학습 모델의 추론 시간에 더 큰 영향을 준 것은 카메라의 사용 여부와 그 크기였다. 카메라를 사용하지 않을 경우, 이미지 해상도 증가에 따른 추론 시간의 증가가 있었으나 그 증가는 미미하였다.

또한 이러한 결과는 실험 2에서 확인한 이미지의 해상도가 추론 시간의 증가를 유도하지만, 그 증가폭은 미미하다는 결과와 일치하는 결과이다.

실험 2와 실험 3에서 해상도의 증가가 추론 시간의 증가를 유도하긴 하였으나 그 증가가 낮은 현상을 확인하였는데, 이는 [9]와 다소 일치하지 않는 결과이다. 본 논문은 하나의 사전 학습 모델을 사용하였고, 같은 모델이더라도 파라미터의 개수나 학습된 조건 등이 다를 수 있으며, 사전 학습 모델의 추론 시간은 하드웨어의 특징에 상당한 영향을 받는 점 등을 고려했을 때 [9]의 가설을 반박하는 내용이라고 하기에는 다소 무리가 있을 것으로 보인다.

5. 결론 및 향후 연구

본 논문은 실시간 인공지능 서비스 예시 중 하나로 자원 제약적인 기기에서 자율주행하는 상황을 가정하였다. 낮은 해상도의 이미지에서는 태스크 오프로딩 구조의 적용을 통해 유의미한 성능 개선을 도출할 수 있었으며 자율주행의 실시간 기준을 충족하는 수치를 기록하였다.

자율주행 분야에서 가장 먼저 고려되어야 하는 것이 실시간성은 맞으나, 정확도 또한 동등하게 중요하다. 해상도가 낮은 이미지의 경우, 이미지 내 모든 객체가 검출되지 못하거나 신뢰 점수(confidence score)가 떨어지는 등 정확도와 관련한 문제가 발생할 여지가 있다. 따라서 높은 해상도의 이미지에서도 태스크 오프로딩의 효용성을 확인할 필요가 있었다.

해상도가 높은 이미지의 경우 성능 개선은 있었으나, 자율주행의 실시간 기준을 충족하지 못하는 수치를 기록하였다. 해상도로 인한 시간적 손실이 어느 과정에서 일어나는지 파악하기 위해, 통신을 제외한 후 객체 탐지 성능을 측정하였다. 통신을 제외한 실험에서, 해상도 증가에 따른 추론 시간의 증가는 미미한 것으로 확인되었다. 즉, 해상도 증가에 따른 통신 시간의 증가가 시간적 손실의 주 요인임을 확인할 수 있었다. 또한 이러한 경향은 해상도가 높아질수록 더 크게 나타났다. 이러한 실험의 결과를 통해 실시간 인공지능 서비스에 있어 중요한 점이 통신 환경이라는 점을 증명할 수 있었다.

본 논문의 통신 환경은 이더넷과 이더넷 케이블을 이용한 환경이다. 실제 자율주행 상황은 무선 네트워크 환경이며 무선 네트워크 환경은 일반적으로 유선 네트워크 환경보다 느리다는 점에서 본 논문의 한계가 있을 수 있다. 다만 통신 환경

에 영향을 많이 받는 만큼, 차세대 핵심 기술 중 하나인 5G MEC(Mobile Edge Computing) 등의 환경에서 좋은 시너지 효과를 낼 수 있을 것으로 기대된다.

추가로 특정 조건 하에서 태스크 오프로딩이 높은 효용성을 발휘할 수 있다는 점도 확인하였다. 특정 조건 중 하나는 좋은 통신 환경이며, 이에 관한 추가 연구가 필요하다.

References

[1] S. H. Kwak, "A task offloading scheme for supporting autonomous platooning based on fog computing," Master degree dissertation, The Graduate School of Chung-Ang University, 2021.

[2] Y. Hong, "A study on the system for AI service production," *KIPS Transactions on Computer and Communication Systems*, Vol.11, No.10, pp.323-332, 2022. DOI: https://doi.org/10.3745/KTCCS.2022.11.10.323.

[3] Intel AI Object Detection [Internet], https://github.com/IntelAI/models/blob/master/docs/object_detection/tensorflow_serving/Tutorial.md.

[4] Generic technologies enabling autonomous driving [Internet], https://spri.kr/posts/view/21781?code=column.

[5] Lewis. "Object Detection for Autonomous Vehicles Gene," 2016.

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

[7] Compare gRPC services with HTTP APIs [Internet], https://learn.microsoft.com/ko-kr/aspnet/core/grpc/comparison?view=aspnetcore-7.0.

[8] BDD100K: A Large-scale Diverse Driving Video Database [Internet], https://bair.berkeley.edu/blog/2018/05/30/bdd/.

[9] Q. Liu, S. Huang, J. Opadere, and T. Han, "An edge network orchestrator for mobile augmented reality," *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, Honolulu, HI, USA, pp.756-764, 2018, doi: 10.1109/INFOCOM.2018.8486241.

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