creative commons

COMMONS DEED

저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.

변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

공학석사 학위논문

멀티쓰레드 객체 탐지 응용의 종단 간 지연시간 단축

Reducing End-to-end Latency

in a Multithreaded Object Detection Application

2021 년 12 월

서울대학교 대학원

전기정보공학부

김 지 후

멀티쓰레드 객체 탐지 응용의 종단 간 지연시간 단축

지도 교수 홍 성 수

이 논문을 공학석사 학위논문으로 제출함

2021 년 12 월

서울대학교 대학원

전기정보공학부

김 지 후

김지후의 공학석사 학위논문을 인준함

2021 년 12 월

위 원 장 _____ 김 태 환 _____ (인)

부위원장 _____ 홍 성 수 _____ (인)

위 원 _____ 심 규 석 _____ (인)

초 록

객체 탐지는 이미지를 입력으로 받아 객체들의 분류(Classification)와 위치 탐지(Localization)를 수행하는 컴퓨터 비전 기술이다. 최근에 객체 탐지에 많이 사용되고 있는 딥러닝은 높은 계산량을 요구한다는 것이 알려져 있다. 따라서 객체 탐지를 가속하려는 연구들은 대부분 딥러닝 모델을 가볍게 최적화하여 딥러닝 추론(Inference) 시간을 단축시키려 한다. 이런 연구들 중 YOLO(You Look Only Once)는 적당한 정확도와 빠른 추론 속도로 객체 탐지를 위한 대표적인 딥러닝 모델 중 하나로 자리잡았다. 그러나 객체 탐지 응용의 종단 간 지연 시간(end-to-end latency) 관점에서 봤을 때, 추론뿐만 아니라 image fetch, 전처리, 후처리, 화면 출력 등 다른 여러 과정들도 상당한 실행 시간을 차지한다. 또한 멀티쓰레드 객체 탐지 응용에서는 객체 탐지의 각 과정에 소모되는 시간이 중첩되어 종단 간 지연 시간을 단축시키는 것이 어렵다. 본 논문에서는 YOLO 를 기반으로 한 멀티쓰레드 객체 탐지 응용인 Darknet-YOLO 의 종단 간 지연 시간을 단축시키기 위한 3 가지 기법을 소개한다. 우리는 응용에서 나타나는 blocking time 을 분석하고, 이를 제거할 수 있는 해결책을 구현한다. 또한 각 쓰레드의 수행시간을 예측하고 쓰레드의 시작 시간을 동적으로 조절한다. 마지막으로 상당한 계산량을 필요로 하는 preprocessing 함수들을 GPU 가 idle 할 때 GPU 에서 수행하도록 한다. 본 기법은 Nvidia Xavier AGX Jetson 에서 평가되었다.

Nsight System 과 Nvidia Tools Extension(NVTX)를 사용하여 프로파일링 한 결과, Darknet-YOLO 의 종단 간 지연 시간은 58.18% 단축되었다.

주요어 : Object detection, Multithreaded application, End-to-end latency, Darknet-YOLO

학 번 : 2020-25213

목 차

제 1 장 서 론 5

제 2 장 Darknet-YOLO 10

제 3 장 문제 정의 및 해결 기법 13

제 1 절 Blocking time elimination 14

제 2 절 Late fetch 18

제 3 절 Accelerated computing of preprocessing 20

제 4 장 실험 및 결과 22

제 1 절 실험 환경 22

제 2 절 구현 및 실험 평가 23

제 5 장 관련 연구 25

제 1 절 객체 탐지 응용의 딥러닝 추론 시간 단축 25

제 2 절 딥러닝 추론 이외 과정의 실행 시간 단축 26

제 6 장 결 론 29

참고문헌 30

Abstract 33

표 목차

[표 1] 24
[표 2] 24

그림 목차

[그림 1] 9
[그림 2] 10
[그림 3] 10
[그림 4] 12
[그림 5] 13
[그림 6] 14
[그림 7] 16
[그림 8] 17
[그림 9] 19
[그림 10] 21
[그림 11] 25
[그림 12] 26
[그림 13] 26
[그림 14] 27
[그림 15] 27

제 1 장 서 론

객체 탐지는 이미지를 입력으로 받아 객체들의 분류(Classification) 및 위치 탐지(Localization)를 수행하는 컴퓨터 비전 기술이다. 객체 탐지는 안면 인식(face detection)[1], 차량 감지(vehicle detection)[11], 감시 시스템(Surveillance)[3] 등 다양한 분야의 응용에 활용되고 있다. 특히 안면 인식 응용의 경우, 코로나 19 시대가 도래함에 따라 중요성이 더욱 대두되고 있다. 유동인구가 많아 일일이 발열 체크를 할 수 없는 다중 이용 시설의 입구에는 안면 인식을 이용한 발열 측정기가 많이 활용되고 있기 때문이다. 열화상 카메라로 체온을 측정할 때, 정확한 체온 측정을 위해서는 안각 내측의 온도를 측정해야 한다. 따라서 먼저 안면 인식을 바탕으로 안각 내측의 위치를 파악한 뒤, 열화상 카메라로 이 부위의 온도를 측정하여 발열 체크를 수행한다.

이런 일련의 과정에 소요되는 시간을 단축시킬 수 있다면, 많은 사용자들이 발열 체크를 위해 서서 기다려야 하는 불편함을 줄일 수 있을 것이다. 3 과정 (안면 인식, 온도 측정, 측정된 온도를 바탕으로 발열 체크) 중 객체 탐지를 이용한 안면 인식은 다른 과정들에 비해 상당한 계산량을 요구하고 소요되는 시간도 크다[7], [15]. 특히 최근에 객체 탐지에 딥러닝을 사용하면서, 요구되는 계산량은 더 늘어났다. 따라서 안면 인식 발열 측정에

소요되는 시간을 유의미하게 줄이기 위해서는 객체 탐지 수행시간을 단축해야 한다.

딥러닝을 이용한 객체 탐지를 가속하려는 기존의 연구들은 대부분 딥러닝 추론(Inference) 시간을 단축시키기 위해, 더 적은 계산량으로 더 높은 정확도를 낼 수 있도록 딥러닝 모델을 최적화해왔다. 그 중 특히 YOLO 모델은 1-stage detector 모델의 대표 주자로써 SSD 이후 모델들의 아이디어까지 채용하여 최근까지도 버전업을 계속하고 있다. YOLOv3 는 2-stage detector 인 R-CNN 보다 1000 배, Faster R-CNN 보다 8 배 빠르며[6], 같은 1-stage detector 모델인 SSD 보다도 2.2 배 빠르다[2]. 그러면서 mAP(mean Average Precision)는 Faster R-CNN 보다는 7.5% 낮지만, SSD 와 비슷한 수치를 가진다[20].

이처럼 딥러닝 모델은 계속 발전해왔지만, 객체 탐지는 딥러닝 추론만으로 이루어지지 않는다. 카메라로부터 이미지를 fetch 하여 전처리 과정을 거치고, 추론을 거친 뒤에는 후처리 과정을 거쳐 display 된다. 이런 과정들을 모두 고려한 종단 간 지연 시간(end-to-end latency) 관점에서 봤을 때, 딥러닝 추론이 차지하는 시간은 일부분일 뿐이다.

멀티쓰레드 객체 탐지 응용에서는 이런 객체 탐지의 각 과정에 소모되는 시간이 중첩되기 때문에 종단 간 지연 시간을 단축시키는 것이 난해하다. 한 thread 의 수행 시간을 단축하더라도 다른 과정을 수행하는 thread 를 기다려 종단 간 지연시간에는 변화가 없을 수 있기 때문이다. 따라서 멀티쓰레드 객체 탐지 응용의 종단 간 지연시간을 줄이기 위해서는, 응용의

총체적 분석을 바탕으로 critical path 를 식별하여 이를 줄여야 한다.

본 논문에서는 YOLO 모델을 Darknet 프레임워크를 사용하여 구현한 멀티쓰레드 객체 탐지 응용인 Darknet-YOLO 를 자세히 분석하고, 종단 간 지연 시간을 단축시킨다. Darknet-YOLO 의 객체 탐지 과정은 Image fetch 단계, Inference 단계, Result Display 단계로 구성되어 있다. Image fetch 단계에서는 이미지 프레임이 캡처 되고 전처리 되며, Inference 단계에서는 훈련된 딥러닝 모델을 사용하여 입력으로 받은 이미지 속 객체의 위치와 클래스가 예측되고, Result display 단계에서는 예측된 결과값이 화면에 출력된다.

Darknet-YOLO 는 Image fetch, Inference, Result display 단계가 그림 1 과 같이 각각 fetch thread, detect thread, display thread 에서 실행되는 3 단계 멀티쓰레드 파이프라인 구조를 가지고 있다. n 번째 image frame 의 Image fetch 단계와 n-1 번째 image frame 의 Inference 단계, n-2 번째 image frame 의 Result display 단계가 동시에 실행되어 모두 끝나야 다음 파이프라인 단계로 넘어간다.

우리는 이런 멀티스레드 파이프라인 구조의 응용에서 종단 간 지연 시간을 단축하기 위한 3 가지 기법을 소개한다.

Blocking time elimination

우리는 Image fetch 단계에서 발생하는 blocking time 의 원인을 분석하고, 이를 제거할 수 있는 해결책을 구현한다.

Late fetch

Image fetch 단계의 수행시간이 다른 단계보다 작은 경우, fetch thread 는 다음 파이프라인이 시작될 때까지 sleep 한다. 우리는 파이프라인 단계의 실행 시간을 동적으로 예측하고 fetch thread 의 실행 시작을 늦춘다.

Accelerated computing of preprocessing

Image fetch 단계의 전처리 함수들은 상당한 computation 을 필요로 한다. 우리는 Darknet 프레임워크에 전처리 함수들을 Detect thread 의 유휴 시간 동안 GPU 에서 수행하도록 한다.

우리는 Nvidia Xavier AGX Jetson 에서 본 기법을 평가하였다. Nsight System 과 Nvidia Tools Extension(NVTX)를 사용하여 프로파일링한 결과, 3 가지 기법을 모두 적용한 경우 Darknet-YOLO 의 종단 간 지연 시간은 58.18% 단축되었다.

본 기법은 시스템 자원을 더 적극적으로 사용하도록 만들기에 소비전력을 늘린다. 따라서 필요한 경우에만 본 기법을 적용하여 소비전력을 줄일 것을 제안한다. 안면 인식 발열 측정에서의 적용을 예로 들면, 유동인구가 많은 시간대에는 본 기법을 적용하여 객체 탐지를 빠르게 수행하고, 유동인구가 많지 않은 시간대에는 객체 탐지 수행 시간을 늦춰 소비 전력을 절감할 수 있을 것이다.

이 논문의 나머지 부분은 다음과 같이 구성된다. 2 장에서는 Darknet-YOLO 를 설명하며 3 장에서는 문제를 정의하고 해결책을

제시한다. 4 장에서는 실험 및 결과, 5 장에서는 관련 연구를 설명하고 6 장에서는 논문의 결론을 맺는다.

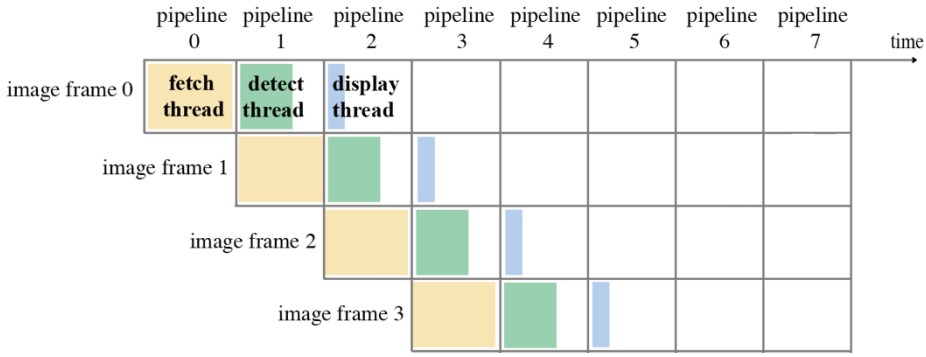

그림 1. Darknet-YOLO 의 3 단계 멀티쓰레드 파이프라인

제 2 장 Darknet - YOLO

Darknet 프레임워크는 딥러닝 모델을 학습, 추론할 수 있는 오픈 소스 신경망 프레임워크로, Darknet-YOLO 는 Darknet 프레임워크를 사용하여 YOLO 모델을 구현한 멀티쓰레드 객체 탐지 응용이다.

Darknet-YOLO 의 객체 탐지 실행 과정은 그림 2 와 같다. Darknet-YOLO 는 객체 탐지 실행 과정을 Fetch thread, Detect thread, Display thread 로 나누어 멀티쓰레드 파이프라인 구조로 동시에 실행한다. 그림 3 은 각 thread 에서 수행되는 기능들을 보인다.

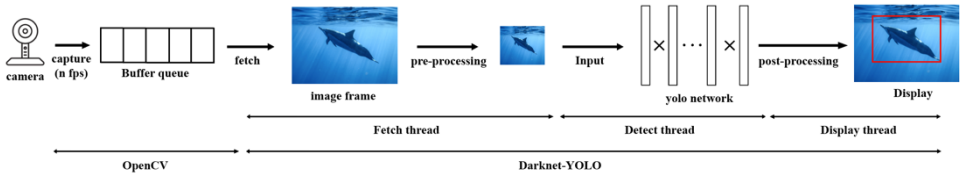

그림 2. Darknet-YOLO 의 객체 탐지 실행 과정

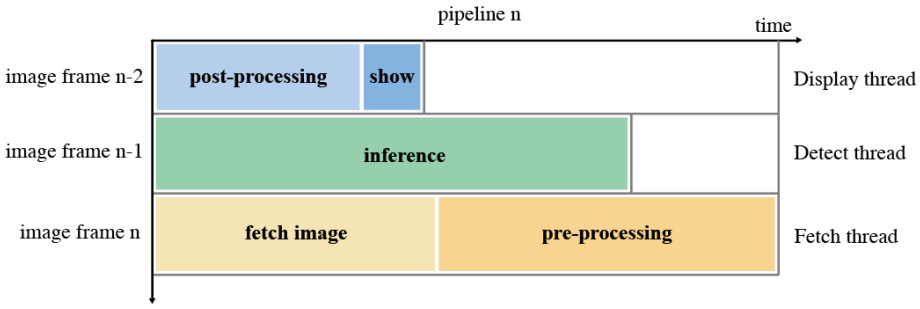

그림 3. Darknet-YOLO 의 각 thread 에서 수행되는 함수들

Darknet-YOLO 는 카메라에서 캡처 된 이미지 프레임을 가져오기 위해 OpenCV(Open Source Computer Vision Library)를 사용한다. 만약 이미지 프레임이 카메라로부터 m fps 의 속도로 캡처 되면 OpenCV 의 버퍼 큐에 $1/m$ (sec)의 속도로 enqueue 된다. Darknet-YOLO 는 Fetch thread 가 실행되면 캡처 된 이미지 프레임을 가져오기 위해 OpenCV 의 버퍼 큐에서 dequeue 한 뒤, 다음과 같은 전처리 과정을 거친다. 먼저, 카메라로부터 캡처 된 이미지 프레임은 OpenCV 의 구조체인 *iplImage* 로 저장된다. Darknet 프레임워크는 이미지 프레임을 저장하는 구조체로 *image* 를 사용하기 때문에 *iplImage* 구조체에서 *image* 구조체로의 데이터 복사가 실행된다. 다음으로, OpenCV 는 컬러 포맷으로 BGR 을 사용하기 때문에 Darknet 프레임워크가 사용하는 RGB 컬러 포맷으로 변환된다. 마지막으로, 다음 단계인 딥러닝 추론에서 YOLO 신경망의 입력에 넣기 위해 이미지 프레임의 크기를 조절한다.

Detect thread 가 실행되면 Fetch thread 에서 전처리 된 이미지 프레임을 입력으로 하여 딥러닝 추론 과정을 거친다. 그리고 그 결과로 탐지된 객체들의 클래스와 위치가 예측된다.

Display thread 가 실행되면 Detect thread 의 결과값인 예측된 객체들의 클래스와 위치가 이미지 프레임 안에 레이블과 경계 박스로 표시되고, OpenCV 의 *VideoCapture* 라이브러리를 통해 화면에 출력된다. 이를 위해 Darknet 프레임워크는 다시 *image* 구조체에서 *iplImage* 구조체로 데이터를 복사하는 후처리 과정을 거친다.

Darknet-YOLO 의 멀티쓰레드 파이프라인에서 thread 들은 항상 동시에 시작되고, 실행이 끝난 thread 는 그렇지 않은 thread 를 기다린다. 따라서 가장 긴 실행 시간을 가지는 thread 에 의해 멀티쓰레드 파이프라인의 실행 시간이 결정된다. Darknet-YOLO 의 멀티쓰레드 파이프라인 중 가장 오랜 실행 시간을 가지는 thread 는 Fetch thread 이고, 그 이유는 다음과 같다.

Darknet-YOLO 는 YOLO 신경망 계산에 소모되는 시간을 고려하여 이미지 프레임이 캡처되는 속도를 제어한다. 즉, OpenCV 의 버퍼 큐에 캡처 된 이미지 프레임이 계산되지 못한 채 쌓이는 것을 방지하기 위해 이미지 프레임 캡처 주기를 Detect thread 의 실행 시간보다 길게 조절한다.

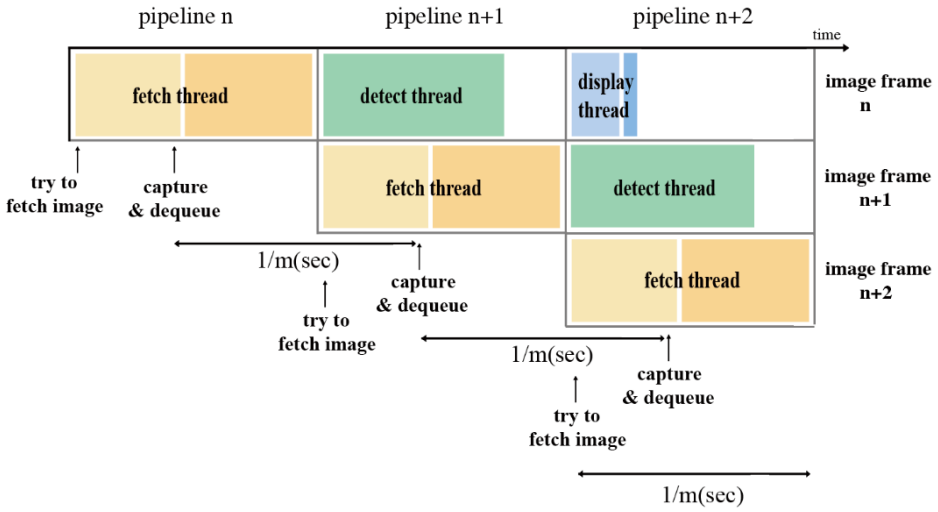

그림 4. 프레임 속도가 m fps 일 때 Fetch thread 의 실행 시간

제 3 장 문제 정의 및 해결 기법

본 논문의 목표는 Darknet-YOLO 의 종단 간 지연 시간을 단축시키는 것이다. Darknet-YOLO 의 종단 간 지연 시간은 이미지 프레임 하나를 fetch 하려는 순간부터 이미지 프레임 안에 객체 탐지 결과가 표시되어 화면에 display 될 때까지의 시간으로 정의되고, 그림 5 와 같다.

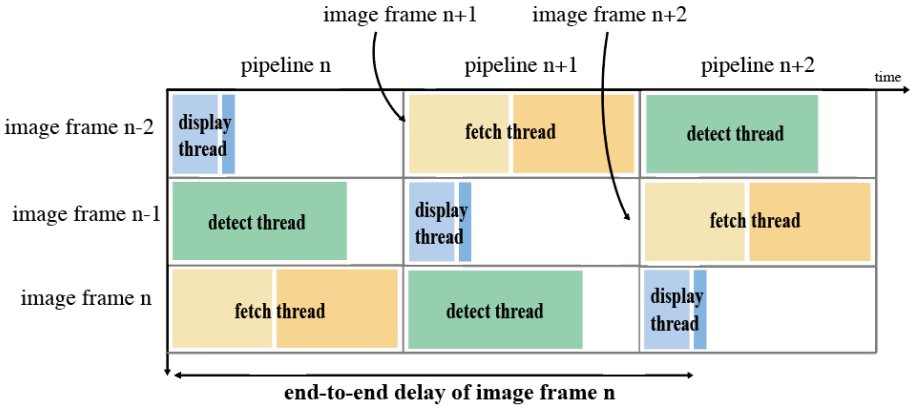

그림 5. Darknet-YOLO 의 종단 간 지연 시간

Darknet_YOLO 의 종단 간 지연 시간은 타겟 하드웨어인 Nvidia Xavier AGX Jetson 에서 Fetch thread 의 실행 시간과 Display thread 의 후처리 과정의 실행 시간에 의해 결정된다. 후처리 과정의 실행 시간은 비교적 짧기 때문에 종단 간 지연 시간은 Fetch thread 의 실행 시간에 가장 큰 영향을 받는다. 이 장에서는 종단 간 지연 시간을 단축시키기 위해 Fetch thread 의

실행 시간 중 불필요한 지연 시간을 제거할 수 있는 3 가지 기법을 제시한다.

3.1 Blocking time elimination

Darknet-YOLO 는 카메라에서 이미지 프레임을 캡처하는 주기, 즉 프레임 속도(framerate)가 Detect thread 의 실행 시간보다 길도록 조절한다. 이는 Fetch thread 에서 이미지 프레임을 가져오려 할 때, 느린 프레임 속도 때문에 blocking time 이 유발되는 것을 의미한다.

Blocking time 은 이미지 프레임을 가져오려 시도하는 순간부터 버퍼 큐에서 실제로 dequeue 될 때까지의 시간으로, 프레임 속도가 m fps 일 때 식 (2)로 표현된다.

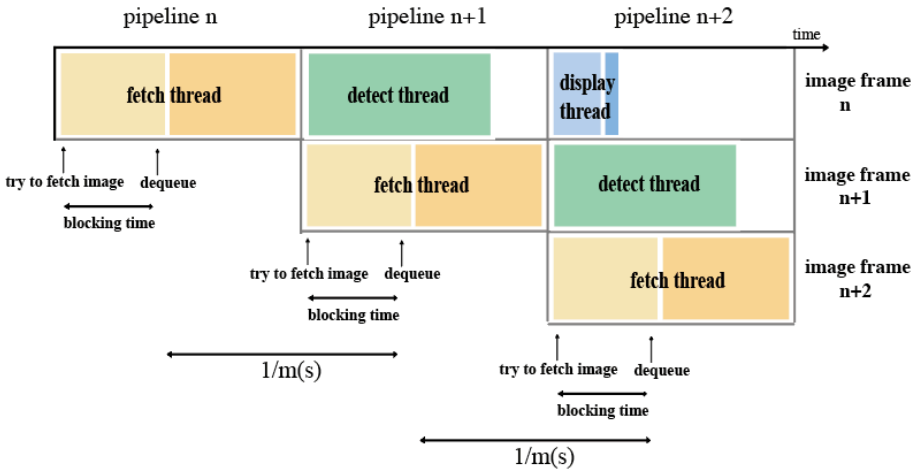

그림 6. Fetch thread 의 blocking time

$$Blocking_time = \frac{1}{m(\text{sec})} - ET_{fetch_thread_without_blocking_time} - ET_{pre_processing} \quad (2)$$

Blocking time 으로 인해 Fetch thread 의 실행 시간이 늘어나면서 Darknet-YOLO 의 종단 간 지연 시간은 더욱 증가하게 된다. 본 논문에서는 blocking time 을 없애기 위해 OpenCV 에서 프레임 속도를 조절하는 부분을 제거하고, 카메라의 최대 프레임 속도로 이미지 프레임을 캡처하였다. 그 결과 그림 7 과 같이 blocking time 이 제거되면서 Fetch thread 의 실행 시간은 Detect thread 의 실행 시간보다 짧아지고, 멀티쓰레드 파이프라인의 실행 시간은 Detect thread 의 실행 시간으로 결정된다. 또한 blocking time 동안 Fetch thread 가 polling 하면서 다른 thread 들에 가해졌던 오버헤드가 해소되면서 Detect thread 와 Display thread 의 실행 시간도 단축된다. Blocking time elimination 기법으로 단축된 종단 간 지연 시간은 다음 식 (3)으로 표현된다.

$$End_to_end\ delay = 2 \times ET_{detect_thread} + ET_{post_processing} \quad (3)$$

카메라의 최대 프레임 속도로 이미지 프레임을 캡처하게 되면 end-to-end latency 는 단축되지만 그림 8 과 같은 문제가 발생한다. Detect thread 에서 이미지 프레임이 YOLO 신경망에서 계산되는 속도가 느리면 버퍼 큐에는 계산되지 못한 이미지

프레임이 계속 쌓이게 된다. 그 결과 Detect thread 에서 계산되는 이미지 프레임은 가장 최신의 이미지 프레임이 아니고, Detect thread 에서는 오래된 이미지 프레임을 계산하게 된다.

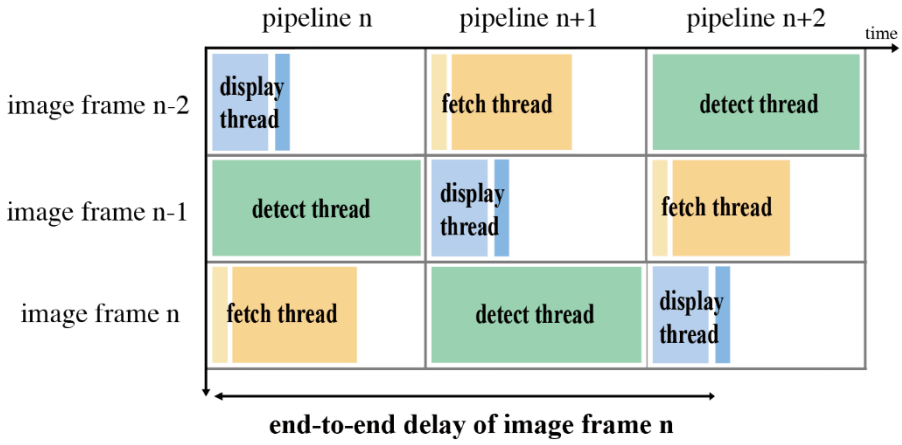

그림 7. Blocking time elimination 기법 적용 결과

이와 같은 문제를 해결하기 위해 본 연구에서는 Darknet 프레임워크에 프레임 속도마다 발생되는 타이머 인터럽트와 버퍼 스택을 생성하였다. 타이머 인터럽트의 ISR(Interrupt Service Routine)에서는 이미지 프레임이 캡처 되면 OpenCV 의 버퍼 큐에서 Darknet 프레임워크의 버퍼 스택으로 옮긴다. Fetch thread 가 실행되면서 이미지 프레임을 가져올 때 버퍼 스택에 이미지 프레임이 있다면 가져오고, 없다면 다시 OpenCV 를 사용해 버퍼 큐에서 가져온다. 어떤 방법으로도 프레임 속도와 관계없이 Detect thread 에서는 가장 최신의 이미지 프레임을 계산할 수 있도록 보장된다.

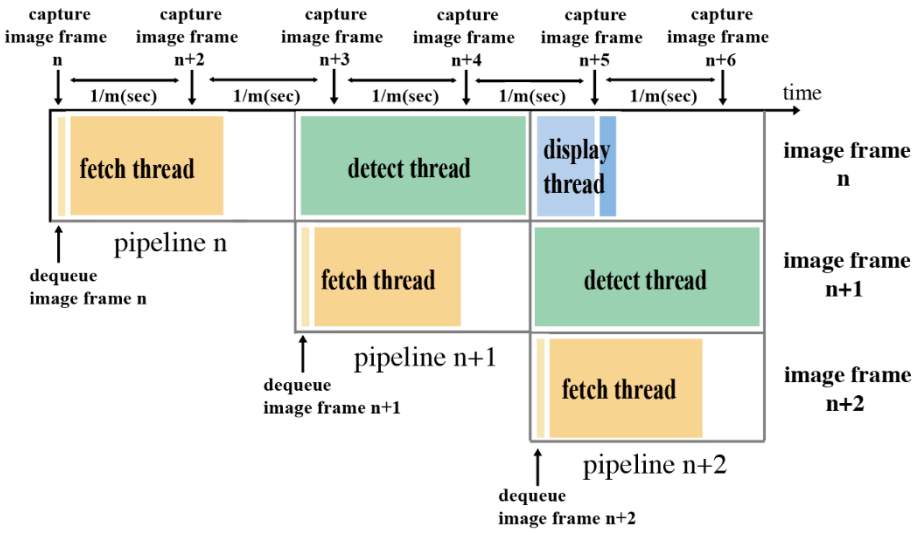

그림 8. 프레임 속도 조절 코드의 제거로 인한 문제 발생

3.2 Late fetch

Darknet-YOLO 는 Fetch thread 가 실행되면 이미지 프레임을 버퍼 스택이나 버퍼 큐에서 가져온다. 하지만 멀티쓰레드 파이프라인의 남은 실행 시간 동안에도 이미지 프레임은 계속해서 캡처 되어 축적된다. Fetch thread 가 실행 되자마자 이미지 프레임을 가져오면 Detect thread 이 실행될 때는 이미 오래된 이미지 프레임이 되고, 카메라에 실제로 인식되고 있는 이미지 프레임과 격차가 발생하게 된다. 본 연구에서는 가능한 YOLO 신경망에 입력되기 직전, 즉 Detect thread 가 실행 되기 직전에 캡처 된 이미지 프레임을 가져오기 위해 최대한 다음 파이프라인의 실행 시작 시간에 가깝게 Fetch thread 의 실행 시작을 늦춘다. Fetch thread 가 실행되면 식 (4)로 정의된 Sleep time 동안 잠들게 된다.

$$\text{Sleep_time}_n = ET_{detect_thread_{n-1}} - ET_{fetch_thread_{n-1}} \qquad (4)$$

Sleep time 은 현재 파이프라인의 실행 시간으로부터 다음 파이프라인의 실행 시간을 예측하여 정의된 값이다. Sleep time 은 Nsight System[9]과 Nvidia Tools Extension[10]으로 프로파일링 한 결과 대체로 일정하다. 그 이유는 3.1 절에서 설명했듯이, Detect thread 의 실행 시간을 고려하기 위해 Fetch thread 에서

이미지 프레임을 가져오는 주기가 조절되기 때문이다. 정의된 Sleep time 의 오차범위는 실험 결과 2.1% 미만이다.

Fetch thread 가 Sleep time 동안 잠들게 되면 그림 9 와 같이 Fetch thread 의 실행 시간은 Detect thread 의 실행 시간과 거의 동일해진다. Late fetch 기법으로 인해 단축된 end-to-end latency 는 다음 식 (5)와 같이 표현된다.

$$\text{End_to_end } delay = \left(ET_{fetch_thread} - ET_{sleep_time}\right) + ET_{detect_thead} + ET_{post_processing} \quad (5)$$

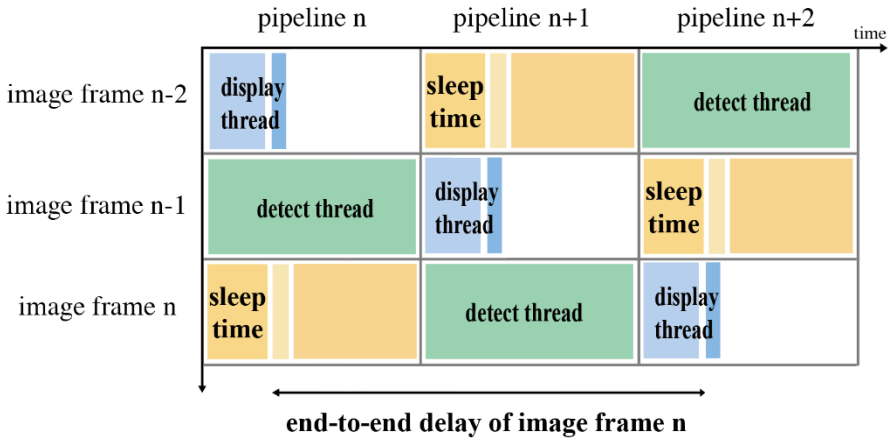

그림 9. Late fetch 기법의 적용 결과

3.3 Accelerated computing of preprocessing

Darknet 프레임워크에서 전처리 함수들은 낮은 GPU 활용률을 보인다. Fetch thread 의 전처리 과정에서 호출되는 함수들은 이미지 프레임의 데이터 복사, 이미지 컬러 포맷 변환, 신경망에 입력되기 위한 사이즈 조정이다. 이 전처리 함수들의 가장 큰 특징은 단순한 코드의 반복이다. 1280*800 해상도 이미지 프레임 기준 각 함수들의 코드 반복 횟수는 최소 99 만회에서 최대 307 만회이다. 이런 단순한 코드 반복 패턴을 보이는 전처리 함수들을 GPU 에서 실행되도록 병렬 프로그래밍한다면 단 1 회의 실행만으로 같은 기능을 수행할 수 있고, 전처리 과정의 실행 시간은 현저하게 단축될 수 있다. 하지만 Darknet 프레임워크는 YOLO 신경망 계산 외에는 거의 GPU 를 사용하지 않는다. 그 이유는 Nvidia GPU 는 Multi process service 를 사용하지 않는 이상 순차적으로 GPU 커널이 실행되기 때문에 전처리 과정에서 GPU 를 사용하게 되면 그만큼 YOLO 신경망 계산이 지연되기 때문이다.

하지만 본 연구에서는 3.2 절에서 설명한 기법인 Late fetch 기법으로 전처리 과정을 GPU 에서 실행시킨다면 Sleep time 은 더욱 늘어난다. 그에 따라 Fetch thread 의 실행은 더욱 지연되고, 이는 Detect thread 에서 YOLO 신경망 계산이 끝난 후 CPU 에서 객체 탐지 결과를 처리할 때 정확히 overlap 된다. 이는 YOLO

신경망 계산이 지연되지 않고 전처리 과정을 GPU 에서 실행시킴으로써 종단 간 지연 시간을 단축시켰음을 뜻한다.

또한 Darknet-YOLO 의 Makefile 에 전처리 과정에서 GPU 를 사용할 것인지의 옵션을 추가함으로써 전처리 함수들을 GPU 병렬 최적화 API 에서 실행할 것인지의 여부를 선택할 수 있다. 전처리 함수들을 병렬 최적화 API 에서 실행한다면, 전처리 과정의 실행 시간이 단축됨에 따라 sleep time 이 늘어나면서 그림 10 과 같이 end-to-end latency 는 더욱 단축된다.

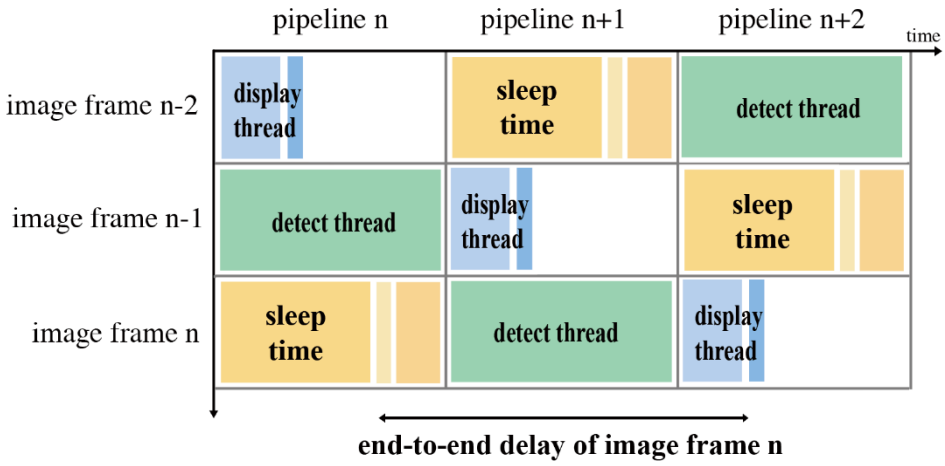

그림 10. Accelerated computing of pre-processing 기법의 적용 결과

제 4 장 실험 및 결과

4.1 실험 환경

본 장에서는 제안된 세 가지 기법을 적용하고, 실험을 통해 원래의 Darknet-YOLO 와 비교함으로써 결과를 보인다. 본 연구의 실험 환경 하드웨어 및 소프트웨어 구성은 다음 표 1, 2 와 같다. 본 연구는 실제 임베디드 시스템에 적용될 수 있도록 Nvidia Xavier AGX Jetson 에서 실험되었다. 또한 임베디드 환경에서 YOLO 모델을 사용하여 추론하기 위해 최적화된 가중치인 tiny-YOLO weights 가 사용되었다.

Hardware
Nvidia Xavier AGX Jetson
Microsoft Corp. LifeCam HD-3000

표 1. 하드웨어 구성

Software
Nvidia Jetpack 4.3.2
Darknet-YOLO with tiny-YOLO weights
OpenCV 4.3
CUDA 10.2

표 2. 소프트웨어 구성

4.2 구현 및 실험 평가

본 연구의 실험에서는 Darknet-YOLO 에서 호출되는 모든 함수의 실행 시간을 Nsight System 과 Nvidia Tools Extension(NVTX)를 사용하여 측정하였다. 실험 환경에서 측정한 원래의 Darknet-YOLO 에서 호출되는 각 thread 평균 실행 시간은 그림 11 과 같다.

Detect thread 의 실행 시간 평균은 75.559ms 이고, Detect thread 보다 긴 실행 시간을 가지도록 원래의 Darknet-YOLO 에서 이미지 프레임이 캡처 되는 주기는 100ms, 즉 10 fps 이다.

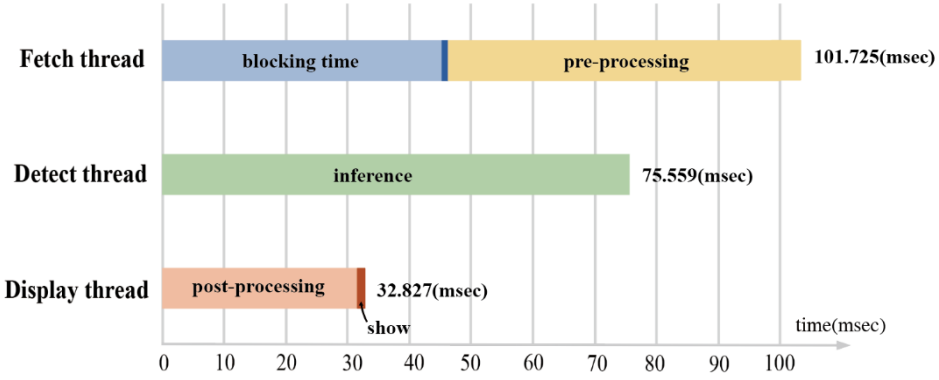

그림 11. Darknet-YOLO 의 각 thread 실행 시간 평균

본 논문에서 end-to-end latency 를 단축시키기 위해 제시한 세 가지 기법을 적용하여 실험해본 결과는 다음과 같다. 먼저 blocking time elimination 기법에서 blocking time 을 없애기 위해 카메라의 최고 프레임 속도인 30 fps 로 이미지 프레임 캡처

주기를 설정하였다. 그리고 가장 최신의 이미지 프레임을 가져오지 못해 일어나는 문제를 해결한 결과, 단축된 실행 시간은 그림 12 와 같다.

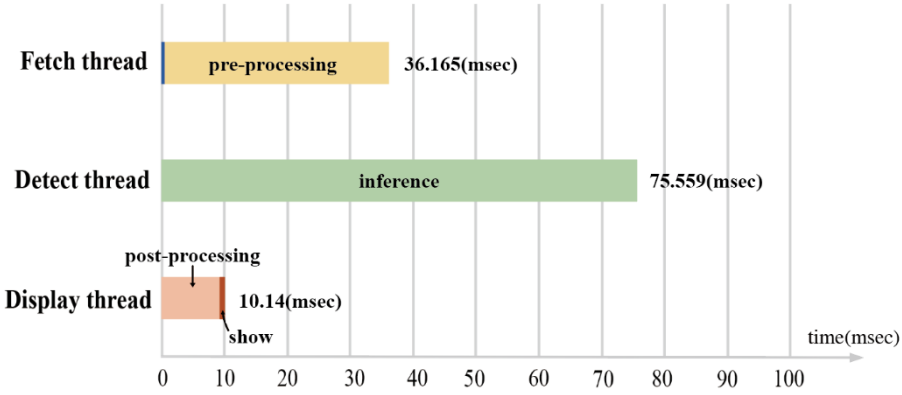

그림 12. Blocking time elimination 기법을 적용한 실험 결과

위의 기법에 Late fetch 를 적용한 실험 결과, sleep time 과 단축된 실행 시간은 그림 13 과 같다.

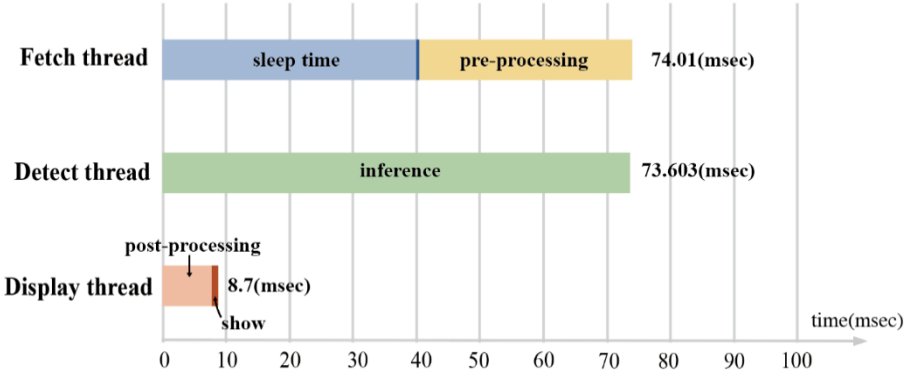

그림 13. Late fetch 기법을 적용한 실험 결과

그 다음으로 Accelerated computing of preprocessing 기법을 적용한 결과, sleep time 은 그림 14 와 같이 늘어나고 제시된 3 가지 기법 모두 적용한 Darknet-YOLO 의 end-to-end latency 는 그림 15 와 같이 58.18% 단축되었다.

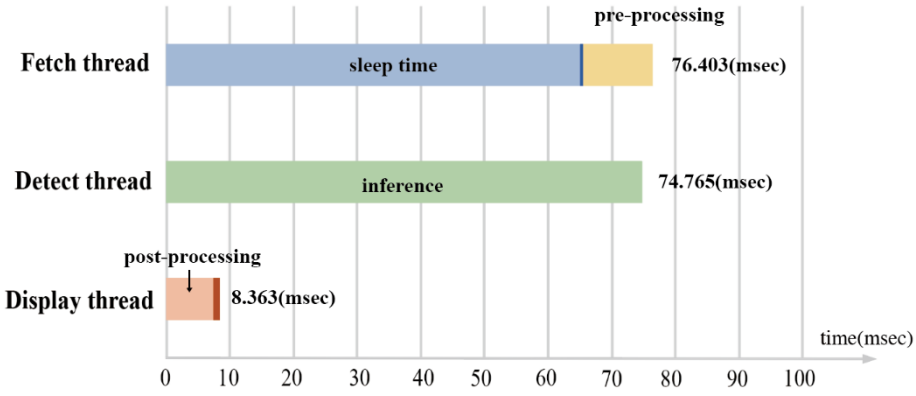

그림 14. Accelerated computing of preprocessing 기법을 적용한 실험 결과

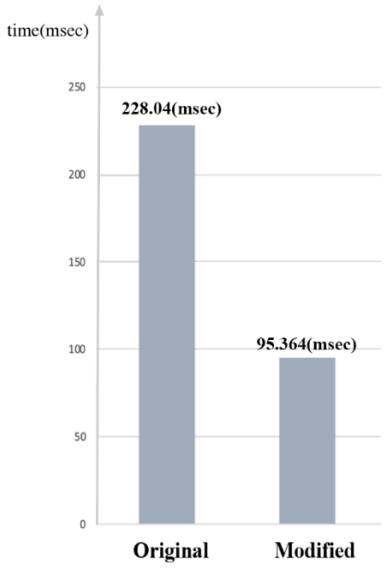

그림 15. Darknet-YOLO 에 3 가지 기법을 적용한 실험 결과

제 5 장 관련 연구

제 5 장에서는 본 연구와 관련된 연구들을 설명한다. 객체 탐지 응용의 종단 간 지연 시간을 단축시키기 위한 기존의 연구들은 두 가지로 나눌 수 있다. 첫번째는 딥러닝 추론에 소모되는 시간을 단축시키는 연구이며, 두번째는 딥러닝 추론 이외의 과정에서 소모되는 시간을 단축시키는 연구이다.

5.1 객체 탐지 응용의 딥러닝 추론 시간 단축

객체 탐지를 위한 딥러닝 모델은 객체의 위치 탐지와 분류를 순차적으로 수행하는 2-stage detector 모델과 이를 동시에 수행하는 1-stage detector 모델로 분류할 수 있다. 2-stage detector 모델의 대표격인 R-CNN[17]은 2014 년에 CNN(Convolutional Neural Network)을 객체 탐지 분야에 최초로 적용시킨 모델이며, CNN 을 이용한 검출 방식이 기존 방식에 비해 높은 정확도를 낼 수 있다는 것을 보여주었다. 이를 개량하여 등장한 Fast R-CNN[5]은 R-CNN 모델에 비해 9 배 빠른 추론 속도와 더 높은 정확도를 보여주었다. 하지만 여전히 임베디드 시스템 분야에서 사용하기에는 너무 무거웠고, 이를 충족하기 위해 1-stage detector 모델인 YOLO(You Look Only Once)[4]가 등장하였다. YOLOv1 모델은 R-CNN 과 Fast R-CNN 보다

정확도는 조금 떨어지지만 추론 속도는 각각 1000 배, 100 배나 더 빠르다[4]. 이후에 등장한 SSD(Single Shot Multibox Detector)[2]는 YOLO 모델보다는 느리지만 2-stage detector 와 비교할만한 정확도를 가졌다. YOLO 또한 이런 모델들의 아이디어를 차용하여 계속 개량되고 있다.

5.2 딥러닝 추론 이외 과정의 실행 시간 단축

객체 탐지 응용의 종단 간 지연 시간을 단축시키기 위해 딥러닝 추론 이외 과정의 실행 시간을 줄이는 연구는 드물다. 우리는 멀티쓰레드 객체 탐지 응용의 종단 간 지연 시간을 단축시키는 연구를 하나 발견하였다[8]. [8]에서 제안하는 3 가지 기법은 on-demand capture, zero-slack pipeline, contention-free pipeline 이다. 이 중 on-demand capture 와 zero-slack pipeline 은 우리가 제안하는 기법과 유사하다. On-demand capture 는 우리의 blocking time elimination 과 세부 구현만 다를 뿐 거의 동일하다. Zero-slack pipeline 은 late fetch 와 유사하지만 쓰레드의 수행시간을 runtime 에 동적으로 예측하지 않고 pre-runtime 에 예측한다는 차이가 있다. Contention-free pipeline 은 메모리 경합을 제거하기 위해 Detect thread 를 Fetch thread, Display thread 와 겹치지 않도록 sequentially 수행하도록 만든다. 본 논문에서는 메모리 경합 문제를 다루지는 않지만, 멀티쓰레드 파이프라인의 구조 변화 없이 전처리 함수들을 GPU 가 idle 할 때

수행하도록 하는 Accelerated computing of preprocessing 을 제안한다.

제 6 장 결 론

본 논문에서는 Darknet-YOLO 의 end-to-end latency 를 단축시키기 위해 3 단계 멀티쓰레드 파이프라인을 자세히 분석한다. 멀티쓰레드 파이프라인의 실행 시간을 결정하는 가장 긴 실행 시간을 가지는 Fetch thread 는 프레임 속도 조절로 야기되는 blocking time 때문에 불필요한 지연 시간이 유발된다. 본 논문에서는 이를 제거하고 그로 인해 나타나는 문제점을 Darknet 프레임워크에 버퍼 스택을 생성함으로써 해결하였다. 또한 Late fetch 기법에서는 다음 파이프라인의 실행 시간을 현재 파이프라인 실행 시간으로부터 미리 예측하여 Fetch thread 의 실행 시작을 동적으로 조절한다. Darknet-YOLO 를 프로파일링 한 결과, 멀티쓰레드 파이프라인의 실행 시간을 시간이 갈수록 일정해지기 때문에 Sleep time 의 오차범위는 2.1% 미만이 되는 것을 확인하였다. Accelerated computing of preprocessing 기법에서는 Darknet-YOLO 에서 실행되는 전처리 함수들은 낮은 GPU 활용률을 해결한다. 전처리 함수들의 GPU 병렬 최적화를 수행하는 API 를 Darknet 프레임워크 안에 생성함으로써 전처리 함수들의 실행 시간을 70% 단축하였다. 동시에 Sleep time 은 식 (4)에 의해 더욱 늘어나고, 종단 간 지연 시간은 더욱 단축된다. 3 가지 기법을 전부 적용해 실험한 결과, Darknet-YOLO 의 종단 간 지연 시간은 58.18% 단축되었다.

참고 문헌

[1] Yang, Wang, and Zheng Jiachun. "Real-time face detection based on YOLO." *2018 1st IEEE international conference on knowledge innovation and invention (ICKII)*. IEEE, 2018.

[2] Liu, Wei, et al. “Ssd: Single shot multibox detector” *European conference on computer vision*. Springer, Cham, 2016.

[3] Joshi, Kinjal A., and Darshak G. Thakore. "A survey on moving object detection and tracking in video surveillance system." *International Journal of Soft Computing and Engineering* 2.3 (2012): 44-48.

[4] Redmon, Joseph, et al. “You only look once: Unified, real-time object detection.”*Proceedings of the IEEE conference on computer vision and pattern recognition.* 2016.

[5] Girshick, Ross. “Fast r-cnn *InProceedings of the IEEE International Conference on Computer Vision* (pp. 1440-1448).” Piscateaway, NJ:IEEE.[Google Scolar](2015).

[6] 7 Ren, Shaoqing, et al. "Faster R-CNN: towards real-time object detection with region proposal networks." *IEEE transactions on pattern analysis and machine intelligence* 39.6 (2016): 1137-1149.

[7] Bieszczad, Grzegorz, et al. "Image processing module for high-speed thermal camera with cooled detector." *Infrared*

Technology and Applications XXXVII. Vol. 8012. International Society for Optics and Photonics, 2011.

[8] Jang, Wonseok, et al. "R-TOD: Real-Time Object Detector with Minimized End-to-End Latency for Autonomous Driving." *2020 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2020.

[9] Nvidia nsight systems. *https://developer.nvidia.com/nsight-systems.*

[10] Nvidia tools extension. *https://docs.nvidia.com/nsight-visual-studio-edition/2020.1/nvtx/index.html*.

[11] Michalopoulos, Panos G. "Vehicle detection video through image processing: the autoscope system." *IEEE Transactions on vehicular technology* 40.1 (1991): 21-29.

[12] Viraktamath, Dr SV, Pratiksha Navalgi, and Ambika Neelopant. "Comparison of YOLOv3 and SSD Algorithms." *International Journal of Engineering Research & Technology (IJERT)* 10.02 (2021).

[13] Long, Xiang, et al."PP-YOLO: An effective and efficient implementation of object detector." *arXiv preprint arXiv:2007.12099 (2020).*

[14] Kim, Jeong-ah, Ju-Yeong Sung, and Se-ho Park. "Comparison of Faster-RCNN, YOLO, and SSD for real-time vehicle type recognition." *2020 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*. IEEE, 2020.

[15] Cheong, Yuen Kiat, Vooi Voon Yap, and Humaira Nisar. "A novel face detection algorithm using thermal imaging." *2014 IEEE Symposium on Computer Applications and Industrial Electronics (ISCAIE)*. IEEE, 2014.

[16] https://pjreddie.com/darknet/yolo/

[17] Wang, Jin, et al. "Dimensional sentiment analysis using a regional CNN-LSTM model." *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. 2016.

Abstract

Reducing End-to-end Latency in a Multithreaded Object Detection Application

Jihu Kim

Electrical and Computer Engineering

The Graduate School

Seoul National University

Object detection is a computer vision technology that receives images as input and performs object classification and localization. Deep learning, which has been widely used for object detection recently, is known to require high computational demand. Therefore, most existing studies accelerating object detection have tried to optimize the deep learning model to require less computational demand, in order to reduce the time consumed in deep learning inference. Among these studies, YOLO (You Look Only Once) has been established as one of the representative deep learning models for object detection due to its fast inference speed

with modest accuracy. However, not only inference but also other processes such as image fetch, pre-processing, post-processing, and display occupy significant time in the end-to-end latency of an object detection application. Also, in a multithreaded object detection application, the time spent for each process can be overlapped and makes it challenging to reduce the end-to-end latency. In this paper, we present 3 techniques to reduce the end-to-end latency of a Darknet-YOLO which is a multithreaded object detection application based on YOLO. First, we thoroughly analyze the cause of blocking time and eliminate it. Second, we predict the execution time of each thread and dynamically control the starting time. Lastly, we run the pre-processing functions requiring high computational demand on the GPU when the GPU is idle. Proposed techniques are evaluated on Nvidia Xavier AGX Jetson. As a result of profiling with Nsight System and Nvidia Tools Extension(NVTX), the end-to-end latency of Darknet-YOLO is reduced by 58.18%.

Keywords : Object detection, Multithreaded application, End-to-end latency, Darknet-YOLO

Student Number : 2020-25213