한국빅데이터학회지
제9권 제1호, 2024, pp. 89-100 https://doi.org/10.36498/kbigdt.2024.9.1.89

ONNX 기반 런타임 성능 분석: YOLO와 ResNet

ONNX-based Runtime Performance Analysis: YOLO and ResNet

김정현 · 이다은 · 최수빈 · 전경구†

인천대학교 임베디드시스템공학과

요 약

컴퓨터 비전 분야에서 You Look Only Once(YOLO)와 ResNet 등의 모델은 실시간 성능과 높은 정확도로 인해 널리 사용되고 있다. 그러나 실제 환경에 이러한 모델들을 적용하려면 런타임 호환성, 메모리 사용량, 컴퓨팅 리소스 및 실시간 조건 등의 요소를 고려해야 한다. 본 연구에서는 세 가지 심층 모델 런타임 ONNX Runtime, TensorRT 및 OpenCV DNN의 특성을 비교하고, 2가지 모델에 대한 성능을 분석한다. 이러한 분석을 통해 현장 적용을 위한 런타임 선택에 기준을 제공해 주는 것이 논문의 목표이다. 실험에서는 차량 번호판 인식 및 분류 업무에 대해 소요 시간, 메모리 사용량, 정확도 평가 지표를 기반으로 런타임들을 비교한다. 실험 결과, ONNX Runtime은 복잡한 객체 탐지 성능이 우수하며, OpenCV DNN은 제한된 메모리 환경에 적합하고, TensorRT는 복잡한 모델의 실행 속도가 우수하다는 것을 보여준다.

■ 중심어 : ONNX, 딥 모델, Inference Runtime, 객체 감지, 이미지 분류

Abstract

In the field of computer vision, models such as You Look Only Once (YOLO) and ResNet are widely used due to their real-time performance and high accuracy. However, to apply these models in real-world environments, factors such as runtime compatibility, memory usage, computing resources, and real-time conditions must be considered. This study compares the characteristics of three deep model runtimes: ONNX Runtime, TensorRT, and OpenCV DNN, and analyzes their performance on two models. The aim of this paper is to provide criteria for runtime selection for practical applications. The experiments compare runtimes based on the evaluation metrics of time, memory usage, and accuracy for vehicle license plate recognition and classification tasks. The experimental results show that ONNX Runtime excels in complex object detection performance, OpenCV DNN is suitable for environments with limited memory, and TensorRT offers superior execution speed for complex models.

■ Keyword : ONNX, Deep Model, Inference Runtime, Object Detection, Image Classification

2024년 05월 27일 접수; 2024년 06월 16일 수정본 접수; 2024년 06월 20일 게재 확정.
* 이 논문은 인천대학교 2024년도 자체연구비 지원(2024-0078)에 의하여 연구되었습니다.
†교신저자 (kjun@inu.ac.kr)

Ⅰ. 서론

최근 딥러닝 응용 산업은 온보드(on-board) 모델 추론으로 발전하고 있다. 데스크톱이나 서버에서는 충분한 컴퓨팅 파워로 추론 시간이 짧지만, 온보드 환경에서는 낮은 컴퓨팅 파워로 인해 추론 시간이 길어지는 문제가 있다. 또한, 각 보드마다 전용 운영체제를 사용함에 따라 학습에 사용한 런타임(runtime)이 달라져 추론이 불가능할 수도 있다.

딥러닝 모델을 다양한 런타임에서 호환하기 위해 Open Neural Network eXchange (ONNX) 형식이 개발되었다. ONNX는 연구에 많이 사용하는 PyTorch와 산업에 사용되는 Tensorflow뿐만 아니라 TensorRT, openCV 등의 다양한 런타임 사이에서 자유롭게 학습과 추론을 할 수 있도록 지원하는 표준이다. ONNX를 적용하는 방법은 학습된 가중치가 포함된 모델 파일을 ONNX 형식으로 변환한 후, 사용할 런타임에서 ONNX 파일을 로드하여 추론을 진행하는 것이다.

온보드 딥러닝이 대두되며 런타임 별 성능 비교 연구의 필요성[1]이 증가하고 있다. 런타임 간 모델 호환을 위해 ONNX 형식으로 변환할 때, 일부 파라미터들이 무시되어 정확도가 변동될 수 있으며 런타임마다 모델 로드 속도, 추론 속도, 메모리 사용량 등이 달라질 수 있다. 이는 런타임마다 각각의 최적화 및 실행 방식이 다르기 때문이다. 이러한 문제에 대한 연구는 런타임을 활용하는 연구[2-4]에 비해 덜 조명되고 있다. 따라서 본 논문에서는 런타임 별 성능 비교 실험을 진행하고자 한다.

본 논문에서는 미국과 캐나다 차량 번호판 데이터를 이용하여 런타임 별 성능을 비교한다. 차량 번호판은 번호판을 찍는 위치, 차량의 종류에 따라 다양한 각도와 크기로 나타날 수 있다. 또한, 차량 유형 및 환경 조건이 다양한 데이터를 포함하고 있기 때문에 차량 번호판 데이터를 채택하였다. 미국과 캐나다는 한국과 달리 주 및 지방에 따라 다양한 모양과 디자인의 차량 번호판을 가지고 있어, 추론 런타임 별 성능 비교에 적합하다고 판단하였다.

실험 결과를 통해 런타임 및 컴퓨팅 자원에 따른 새로운 통찰을 발견할 수 있었다. 예를 들어, GPU를 사용할 때 객체 탐지 작업에서 CPU에 비해 큰 모델과 경량 모델의 메모리 사용량 차이가 적다는 점과 이미지 분류 작업에서는 모델의 은닉층 개수가 달라져도 메모리 사용량 변화가 크지 않다는 점이 그 예시이다. 또한, 동일한 ONNX 모델로 테스트했을 때 런타임 별 정확도에 차이가 발생하는 점도 발견하여 그 이유를 추측하였다.

본 논문의 기여 사항은 다음과 같다.

- 런타임의 정량적 성능 평가: 모델 로드 시간, 추론 시간, 메모리 사용량 비교
- 런타임에 따른 모델 정확도 차이 비교 및 분석: 동일 ONNX 모델일지라도 런타임 별로 정확도 차이가 발생하는 현상 분석
- 활용도 높은 비전 인식 모델들에 대한 런타임 선택 가이드라인 제시

Ⅱ. 배경 지식

2.1 ONNX

ONNX는 기계학습 모델을 표현하는 오픈 포맷으로, 다양한 프레임워크 및 런타임 간 모델 호환성을 제공한다. 학습된 모델을 ONNX로 변환한 후, 호환되는 환경에 로드하여 추론할 수 있다. 본 논문에서는 차량 번호판 인식의 하위 작업들을 수행하도록 학습된 PyTorch 딥 모델들을 ONNX로 변환하여 사용한다.

2.2 추론 런타임

2.2.1 ONNX 런타임

ONNX 런타임은 ONNX 모델을 실행할 수 있는 엔진으로, CPU와 GPU에서 모두 추론이 가능하다. 크로스 플랫폼 기반인 ONNX 런타임은 리눅스, 윈도우, 맥 등 여러 운영체제에서 실행 가능하며, Python, C#, C++, Java 등의 언어들을 지원한다. ONNX 런타임은 DNN 및 기존 기계 학습 모델을 모두 지원하며, TensorRT, OpenVINO 및 DirectMl과 같은 하드웨어 별 가속기를 포함한다.

2.2.2 TensorRT 런타임

TensorRT는 딥 모델의 고성능 추론을 가능하게 하는 SDK로 추론 런타임을 포함한다. 이를 TensorRT 런타임으로 표기하며, ONNX 모델과 호환이 가능하다. TensorRT 런타임은 Nvidia GPU에서만 사용 가능하며, 모델 로드 시 낮은 응답 시간과 효율적인 전력 및 메모리 사용을 위해 자동 최적화를 진행한다.

2.2.3 OpenCV DNN 런타임

OpenCV의 Deep Neural Networks (DNN) 모듈을 통해 CPU와 GPU에서 ONNX 모델 추론이 가능하다. OpenCV는 여러 컴퓨터 비전 알고리즘을 포함하는 오픈 소스 BSD 라이선스 라이브러리로, 모듈식 구조를 가지며 패키지에 공유 및 정적 라이브러리가 포함된다. 본 논문에서는 OpenCV의 DNN 모듈을 OpenCV DNN 런타임으로 표기한다.

2.3 차량 번호판 인식

차량 번호판 인식의 하위 작업들은 객체 탐지와 이미지 분류 딥 모델로 구현할 수 있다. 객체 탐지 딥 모델을 통해 번호판의 위치를 찾는 번호판 탐지와 번호판에 적힌 알파벳과 숫자를 인식하는 문자 인식 작업을 수행할 수 있다. 이미지 분류 딥 모델을 사용하여 번호판 이미지로 주(state)를 판단하는 번호판 주 분류 작업도 가능하다.

YOLO는 객체 탐지, ResNet은 이미지 분류가 필요한 작업에 대중적으로 사용되는 딥 모델이다. YOLO 시리즈[5-9]는 번호판 탐지 및 문자 인식 작업에 사용 가능하며, 객체 탐지가 필요한 다른 작업들에도 많이 사용[10-12]되고 있다. ResNet[13]은 주 분류 작업에 적합한 이미지 분류 딥 모델로, 다른 작업들에도 사용[14-16]되고 있다.

2.4 기존 연구

런타임들의 성능을 비교 분석하는 연구는 진행되고 있으나, 다양한 모델들, 성능 척도들 그리고 CPU와 GPU를 모두 고려하여 비교 분석하는 경우는 적다. 다양한 모델들에서 런타임 별 성능을 비교하는 이유는 일반적인 산업적 활용에 앞서 모델들이 다양한 하드웨어에서의 실행 속도나 성능 등의 영향을 고려하여 최적의 하드웨어를 선정할 수 있도록 하기 위함이다.

PyTorch에서 학습된 평균 전력 계산 모델을 ONNX로 변환하여 ONNX 런타임과 TensorRT에서 지연시간을 비교한 연구[17]가 수행되었다. 해당 연구는 TensorRT의 뛰어난 성능을 강조하였다. 또한, PyTorch와 TensorRT에서 학습된 여러 모델들을 ONNX로 변환하고, 모델이 실패하는 시나리오를 분석하는 연구[18]도 진행되었다. 이 연구에서는 모델 연산자와 ONNX 변환기 사이의 오류 관계를 두 가지 가설을 통해 평가 및 검증하여, 딥러닝 소프트웨어들이 나아가야 할 방향을 제시하였다. 더불어, 여러 종류의 모델들을 ONNX로 변환하는 방법을 제시하고, ONNX로 변환된 모델들의 형식을 분석하여 사용된 기계학습 기술을 판단하는 알고리즘 및 추론 시스템을 제안하는 연구[19]도 수행되었다.

Ⅲ. 런타임 평가 환경 구성

3.1 평가에 사용한 딥 모델

3.1.1 객체 탐지 : YOLOv7

번호판 탐지와 문자 인식 작업을 위해, 객체 탐지 모델인 YOLOv7 standard 버전과 경량화 모델인 tiny 버전을 사용한다. Standard 버전은 더 많은 은닉층과 파라미터를 가지고 있어 일반 GPU에 적합하게 설계되었으며, 이후 논문에서는 YOLOv7로 표기한다. Tiny 버전은 저사양의 엣지(edge) GPU에 적합하며, YOLOv7-tiny로 표기한다. 번호판 탐지와 번호판 문자 인식 작업에 각각 두 모델을 별도로 적용하여 구현한다.

3.1.2 이미지 주 분류 : ResNet

번호판 주(state) 분류 작업을 위해, 이미지 분류 모델 ResNet을 사용한다. ResNet은 은닉층의 개수에 따라 여러 버전들 존재하며, 본 논문에서는 ResNet18, ResNet34, ResNet50을 사용한다.

3.2 데이터셋

연구에 활용한 미국과 캐나다 차량 번호판 데이터는 차량 번호판 데이터를 무료로 수집할 수 있는 인터넷 상의 사이트(https://platesmania.com/)를 이용하였다. 학습 및 테스트에 사용된 차량 및 번호판의 이미지 예시는 <그림 1>과 같다. 미국과 캐나다의 번호판은 주(state)마다 색상, 글꼴, 배치 등이 다르며, 동일한 주 내에서도 여러 가지 디자인이 존재한다. 또한 번호판은 차량의 서로 다른 위치에 부착되어 있다. 다양한 디자인을 갖춘 각 이미지들은 학습에 사용되기 위해 전처리 과정을 거친다.

3.2.1 데이터셋 구성

번호판 탐지: YOLO를 학습시키기 위해 미국과 캐나다의 주차 혹은 주행 중 차량 이미지들

(a)

(b)

〈그림 1〉 차량 및 번호판 원본 이미지의 예.
(a) 미국 (b) 캐나다

을 사용한다. 학습 데이터셋은 4,512장, 테스트 데이터셋은 1,177장으로 구성된다. 타겟 클래스는 번호판 1개이며, 이미지 내 번호판 위치 정보들은 별도로 존재한다.

번호판 문자 인식: 이 작업에는 <그림 1>의 우측과 같이 번호판만 표시된 이미지를 사용한다. 미국과 캐나다 번호판에서 알파벳과 숫자를 각각의 클래스로써 레이블링한다. 단, 알파벳 'O'와 숫자 '0'은 같은 클래스로 구분하는데, 둘 간의 구별을 위한 충분한 학습 데이터 확보가 어렵기 때문이다. 학습 데이터셋은 각 문자별 빈도수가 균일하도록 구성하여, 총 385장으로 구성한다. 한 이미지에 여러 개의 문자가 나타나기 때문에 학습 샘플의 개수는 이보다 많아진다. 테스트 데이터셋은 97장으로 동일한 기준으로 구성한다.

번호판 주 분류: ResNet 학습에도 <그림 1>의 우측과 같은 번호판 이미지를 사용한다. 학습 데이터셋은 5,768장, 테스트 데이터셋은 1,439장으로 구성된다. 미국의 행정구역인 50개의 주와 1개의 특별구를 별도의 클래스로 구분한다. 캐나다는 Alberta, British Columbia, Manitoba, Ontario, Quebec, Saskatchewan를 분류 클래스로 한다.

3.2.2 전처리

번호판 탐지와 문자 인식 데이터셋의 전처리 방식은 동일하다. 이미지를 그레이스케일(grayscale)로 변환하고, 원본 이미지의 비율을 유지한 채로 384 x 384로 크기를 조정한다. 이로 인해 생기는 빈 공간은 제로 패딩한다. 마지막으로 학습 데이터셋에 대해 평균과 표준편차를 이용하여 정규화를 진행하며, 이 값들은 테스트에서도 사용된다. YOLO 훈련과정에 augmentation이 포함되기 때문에 별도로 준비하지는 않는다.

주별 분류 데이터셋의 전처리는 이미지를 그레이스케일로 변환 후, 224 x 224 로 크기를 조정한다. 이후, 정규화를 진행하며, 일부 학습 데이터셋에 대해서는 좌우 반전과 회전의 augmentation을 적용한다.

3.3 학습 환경 및 하이퍼라미터

PyTorch로 구현된 모델들을 Ubuntu 서버에서 학습시킨다. 서버 사양은 Intel Xeon W2255와 Nvidia Quadro RTX 5000을 사용한다. YOLOv7과 YOLOv7-tiny는 기본 제공되는 하이퍼파라미터 설정을 사용하고, 에폭(epoch) 수는 150으로 한다. ResNet 시리즈는 학습률을 0.001로 하고, Adam 최적화 기법을 사용한다. 손실 함수로는 교차 엔트로피 오차 함수를 사용하며, Early Stopping으로 36 에폭에서 종료한다. 모든 모델들에 대해 가장 낮은 검증 손실을 기록한 에폭의 가중치를 테스트에서 사용한다.

3.4 모델 학습 결과 검증

번호판 탐지와 문자 인식 작업 모델의 학습 완료 후, 데이터셋 별 정확도를 평가한 결과는 <표 1>과 같다. 이를 통해 학습이 과적합 없이 수행되었음을 알 수 있다. 객체 탐지 모델의 정확도 평가 지표로는 Precision, Recall, mean Average Precision at IOU threshold 0.5 (mAP@.5)를 사용하였다.

〈표 1〉 번호판 탐지 및 문자 인식 작업 모델의 정확도

Task	Model	Dataset	Precision	Recall	mAP@.5
License Plate Detection	YOLOv7	Train	0.984	0.978	0.994
		Test	0.985	0.977	0.990
	YOLOv7 -tiny	Train	0.996	0.996	0.999
		Test	0.997	0.981	0.993
Character Recognition	YOLOv7	Train	0.999	0.999	0.999
		Test	0.997	0.999	0.998
	YOLOv7 -tiny	Train	0.998	0.999	0.999
		Test	0.996	0.999	0.996

〈표 2〉 데이터셋에 의한 주별 분류 작업 모델의 정확도

Task	Model	Dataset	Accuracy
Plate State Classification	ResNet18	Train	1.000
		Test	0.997
	ResNet34	Train	1.000
		Test	0.997
	ResNet50	Train	1.000
		Test	0.997

번호판 주 분류 작업 모델 역시 과적합 없이 학습되었다. 학습 종료 후 데이터셋 별 정확도를 평가한 결과는 <표 2>와 같다. Accuracy 결과를 보면, 모든 모델은 학습 데이터셋에 대해 정확도 1.0을, 테스트 데이터셋에 대해서는 0.997의 정확도를 보였다.

3.5 추론 런타임 성능 비교

추론 런타임 비교는 소요 시간, 사용 메모리 그리고 정확도를 성능 척도로 하며, 세 가지 런타임 활용을 위해 모델들을 ONNX로 변환한다. ONNX 런타임과 TensorRT 런타임의 추론 및 평가 지표 도출 코드는 Python으로 작성되었으며, OpenCV DNN 런타임은 C++로 작성되었다.

소요 시간 지표로 model load time과 Frames

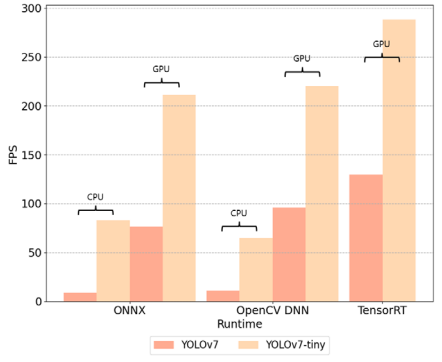
Per Second(FPS)를 사용한다. model load time은 모델을 런타임으로 로드하는 데 소요되는 시간이다. FPS는 1초 당 처리하는 이미지 개수다. 메모리 지표로는 Average Memory Usage를 사용하며, 테스트셋 이미지 추론 작업 동안 사용되는 메모리 평균 사용량이다. 정확도 지표는 앞서 설명한 mAP@.5와 Accuracy를 사용한다.

Ⅳ. 성능 비교

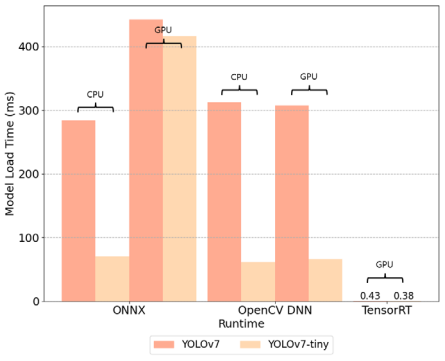
4.1 소요 시간 비교

4.1.1 번호판 탐지 작업

<그림 2>는 FPS와 모델 로드 시간을 보여준다. <그림 2>(a)에서 FPS의 경우 GPU가 CPU에 비해 큰 모델일수록 성능 향상 폭이 커진다. 전반적으로 성능 향상은 최소 2배 이상이며, 특히 OpenCV DNN의 YOLOv7은 9배 이상 우수하다. 이는 큰 모델의 네트워크 구조가 GPU의 병렬 처리를 통해 복잡한 계산을 처리하기에 적합하기 때문으로 추측된다. <그림 2>(b)의 모델 로드 시간은 런타임 별로 차이가 있으며, GPU가 CPU보다 항상 우수한 것은 아니다. TensorRT 런타임은 YOLOv7과 YOLOv7-tiny 모두 가장 빠른 모델 로드 시간을 보인다. GPU 사용 시, ONNX 런타임의 YOLOv7이 가장 긴 모델 로드 시간이 필요하며, CPU 사용 시에는 더 빠르다. OpenCV DNN 런타임은 CPU와 GPU의 차이가 거의 없으며, 이러한 결과는 모델을 GPU에 로드하는 과정에서 메모리 대역폭의 영향을 받기 때문으로 생각된다. 이를 통해 번호판 탐지 작업에서 TensorRT 런타임의 YOLOv7-tiny가 적합한 것으로 보인다.



(a)

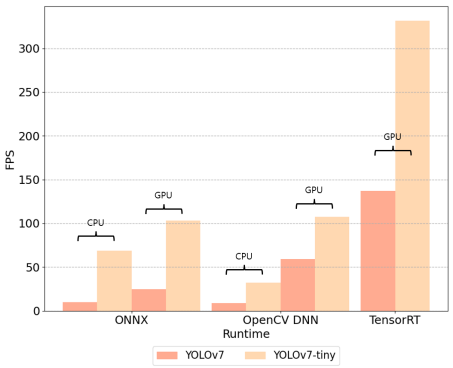


(b)

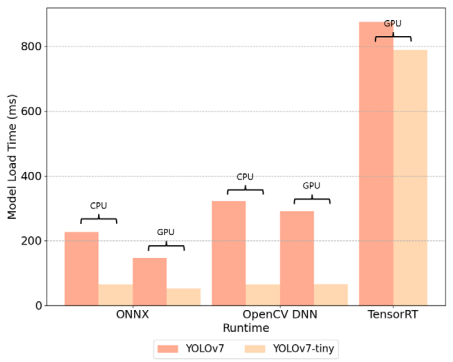
〈그림 2〉 추론 런타임 간 번호판 탐지 FPS 및 모델 로드 시간 비교. (a) FPS (b) 모델 로드 시간

4.1.2 번호판 문자 인식 작업

<그림 3>은 객체 탐지 작업의 복잡도가 런타임 성능에 미치는 영향을 보여준다. <그림 3>(a)는 FPS를 나타내며, 전반적으로 CPU에 비해 GPU의 성능 향상이 최소 1.5배 이상이며, OpenCV DNN 런타임의 YOLOv7은 7배 이상의 성능 향상을 보인다. 여전히 큰 모델일 때 성능 향상 폭이 크지만, 번호판 탐지 작업보다는 낮은 향상 폭을 보인다. 이는 문자 인식 작업에서 타겟 클래스 수의 증가로 인해 복잡도가 더 커졌기 때문으로 보인다. <그림 3>(b)는 모델 로드 시간을 나타내며, 번호판 탐지 작업과는 다른 결과를 보여준다. TensorRT 런타임의 모델 로드 시간이 가장 길며, 탐지 작업에 비해 2000배 이상 증가한 것을 알 수 있다. 반면에 ONNX 런타임은 가장 우수한 결과를 보여주며, 탐지 작업과 비교했을 때 YOLOv7과 YOLOv7-tiny 모두 모델 로드 시간이 최소 3배 이상 감소하였다. 이



(a)



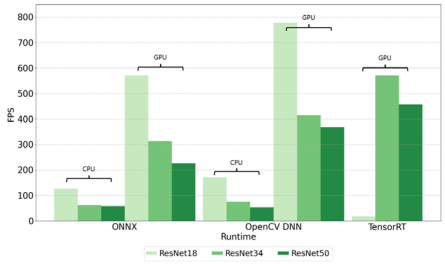
(b)

〈그림 3〉 추론 런타임 간 문자 인식 FPS 및 모델 로드 시간 비교. (a) FPS (b) 모델 로드 시간

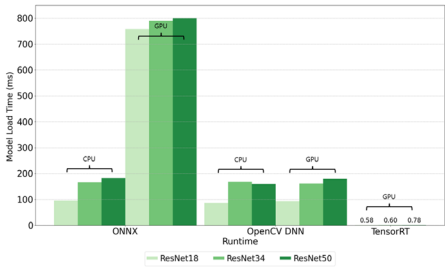
를 통해 번호판 탐지 작업 보다 복잡한 번호판 문자 인식 작업에서는 TensorRT 런타임보다 ONNX 런타임이 더 적합함을 의미한다.

4.1.3 번호판 주 분류 작업

<그림 4>는 딥 모델의 은닉층 개수가 런타임 성능에 미치는 영향을 보여준다. <그림 4>(a)는 FPS를 나타내며, 모델에 따라 적합한 런타임이 다름을 알 수 있다. TensorRT 런타임은 은닉층 개수가 많은 모델일수록 우수한 성능을 보이며, 다른 GPU 사용 런타임에 비해 ResNet34는 약 35%, ResNet50은 약 20% 이상 성능이 우수하다. 이는 TensorRT에서 제공하는 최적화가 레이어 수가 많은 모델에 더 적합하기 때문으로 추측된다. OpenCV DNN 런타임은 ResNet18에서



(a)



(b)

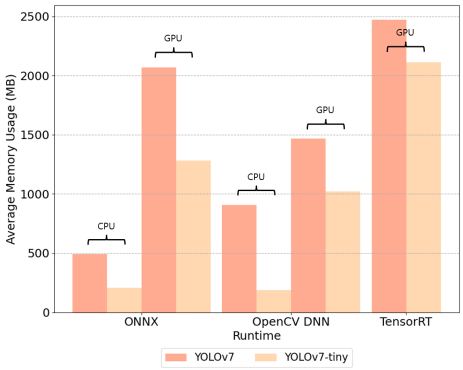
〈그림 4〉 추론 런타임 간 주별 분류 FPS 및 모델 로드 시간 비교. (a) FPS (b) 모델 로드 시간

가장 우수한 성능을 보이며, 다른 런타임에 비해 최소 35% 이상의 FPS를 보여준다. <그림 4>(b)는 모델 로드 시간을 나타내며, 런타임들의 GPU 최적화 수준 차이를 보여준다. TensorRT 런타임에서는 모든 모델이 1ms 이하로 매우 빠르며, GPU를 사용하는 ONNX 런타임의 경우 모든 모델이 다른 런타임에 비해 최소 4배 이상의 모델 로드 시간을 요구한다. 이를 통해 딥 모델의 복잡성이 높은 번호판 주 분류 작업 경우 TensorRT 런타임을, 그렇지 않을 경우에는 하드웨어 사양에 따른 런타임 선택이 필요함을 알 수 있다.

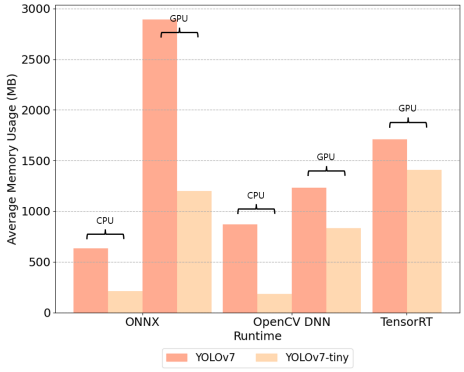
4.2 메모리 사용량 비교

4.2.1 번호판 탐지 작업

<그림 5>는 추론에 사용되는 자원의 평균 메모리 사용량을 보여준다. CPU 사용 시에는 RAM 사용량만을, GPU 사용 시에는 RAM과 VRAM을 합친 사용량을 표시한다. CPU에서 경량 모델과 일반 모델 간의 메모리 사용량 차이가 두



〈그림 5〉 추론 런타임 간 번호판 탐지 평균 메모리 사용량 비교



〈그림 6〉 추론 런타임 간 문자 인식 평균 메모리 사용량 비교

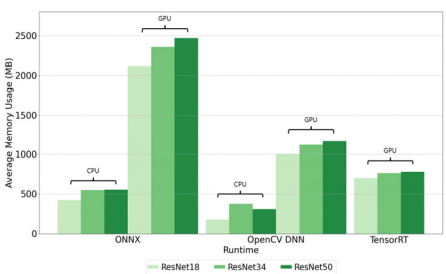
드러지는 것을 알 수 있다. YOLOv7에 비해 YOLOv7-tiny는 GPU에서 최소 15%, 최대 39%까지 감소하였지만, CPU의 ONNX 런타임은 약 58%, OpenCV DNN 런타임은 약 80%까지 크게 감소하였다. 이는 CPU에서만 모델을 로드하고 추론을 진행하여 메모리 사용 효율이 높기 때문으로 추측된다. 이를 통해 객체 탐지 작업인 번호판 탐지 작업에서 GPU 리소스가 충분하고 빠른 처리 속도를 필요로 한다면 TensorRT 런타임을, 메모리 사용량 최소화 또는 제한된 환경인 경우에는 ONNX 또는 OpenCV DNN 런타임을 고려하는 것이 좋아 보인다.

4.2.2 번호판 문자 인식 작업

<그림 6>은 번호판 문자 인식 작업에서의 평균 메모리 사용량을 보여준다. 앞서 언급한 경우와 유사하게 CPU에서 경량 모델의 메모리 사용량이 적었다. ONNX 런타임에서 GPU를 사용할 때 YOLOv7-tiny는 높은 메모리를 사용하지만, YOLOv7의 메모리 사용량은 약 59% 증가하였다. 반면, CPU에서는 ONNX 런타임에서 약 57%, OpenCV DNN 런타임에서 약 79% 감소하였다. 이를 통해 타겟 클래스 개수 증가와 상관없이 CPU의 메모리 사용량이 낮다는 것을 알 수 있다. 복잡해진 객체 탐지 작업인 번호판 문자 인식 작업에서 ONNX 및 OpenCV DNN 런타임은 CPU를 사용함으로써 적은 메모리 사용으로 경량 모델을 활용할 수 있음을 보여준다.

4.2.3 번호판 주 분류 작업

<그림 7>은 번호판 주 분류 작업에서의 메모리 사용량을 보여준다. 동일 런타임에서 GPU를 사용할 때, 이미지 분류 모델의 은닉층 개수와 상관없이 비슷한 메모리 사용량을 보이는 것을 알 수 있다. GPU 사용 런타임 간에는 모델이 달라도 최대 15%의 차이만 존재했다. 특히, TensorRT 런타임에서 ResNet18에 비해 ResNet50의 메모리 사용량은 약 11%만 증가하였다. 이러한 결과는 모든 런타임이 GPU를 사용할 때 모델 연산에 필요한 최적화를 충분히 수행했음을 의미한다. 또한, CPU를 사용할 때 모든 모델은 낮은 메모리 사용량을 보였다. 이는 객체 탐지 작업인 번호판 탐지 작업과 유사하게 ONNX 런타임



〈그림 7〉 추론 런타임 간 주별 분류 평균 메모리 사용량 비교

이 제한된 환경에서 효율적인 자원 사용이 가능함을 보여준다.

4.3 정확도 비교

<표 3>과 <표 4>는 각각 객체 탐지 작업과 이미지 분류 작업에서의 런타임 별 정확도를 PyTorch 모델과 비교하여 보여준다. PyTorch 모델과 ONNX 모델 간, 그리고 런타임 간 동일 모델의 경우에도 정확도 차이가 존재하지만, 그 정도는 매우 작다. 다만, ResNet50의 경우 2.06%p의 차이가 발생했는데, 이는 상대적으로 복잡한 모델을 최적화하는 과정에서 연산의 차이가 발생한 것으로 추측된다. 이러한 차이는 모델을 ONNX로 변환할 때 최적화 과정에서 근사치가 사용되고, TensorRT 런타임에서 모델 로드 시 사용하는 부동 소수점 연산의 한계 때문으로 생각된다.

또 다른 원인은 YOLOv7과 YOLOv7-tiny의 Non-maximum Suppression (NMS) 연산 방식의 차이다. OpenCV DNN 런타임에서는 NMS를 모델 외부에서 별도로 구현해야 하지만, 다른 두 런타임에서는 기본적으로 제공된다. 결론적으로, ONNX 변환 시의 최적화, 런타임들의 정밀도, 그리고 NMS 제공 여부 등의 차이로 인해 정확도 차이가 발생하는 것으로 추측된다. 그럼에도 불구하고, 정확도 차이가 근소하기 때문에 런타임 선택에 있어 큰 고려사항은 아니다.

〈표 3〉 추론 런타임 간 번호판 탐지 및 문자 인식 작업 모델의 정확도

Task	Model	Runtime	mAP@.5	mAP@.5 Delta from PyTorch [%p]
License Plate Detection	YOLOv7	PyTorch	0.9900	-
		ONNX	0.9862	-0.38
		TensorRT	0.9877	-0.23
		OpenCV DNN	0.9907	+0.07
	YOLOv7-tiny	PyTorch	0.9930	-
		ONNX	0.9872	-0.58
		TensorRT	0.9872	-0.58
		OpenCV DNN	0.9905	-0.25
Character Recognition	YOLOv7	PyTorch	0.9980	-
		ONNX	0.9980	0.00
		TensorRT	0.9980	0.00
		OpenCV DNN	0.9900	0.00
	YOLOv7-tiny	PyTorch	0.9960	-
		ONNX	0.9961	+0.01
		TensorRT	0.9961	+0.01
		OpenCV DNN	0.9961	+0.01

4.4 런타임 별 사용성 비교

사용 편의성과 개발 용이성 측면에서 정성적으로 세가지 런타임을 비교해 볼 수 있다. 사용 편의성 면에서, ONNX 런타임은 여러 환경에서 사용이 가능한 호환성과 문제 파악을 위한 세부적인 에러 메시지들을 제공한다는 장점을 가진

〈표 4〉 추론 런타임 간 주별 분류 모델의 정확도

Task	Model	Runtime	Accuracy	Accuracy Delta from PyTorch [%p]
Plate State Classification	ResNet 18	PyTorch	0.9970	-
		ONNX	0.9966	-0.04
		TensorRT	0.9974	+0.04
		OpenCV DNN	0.9974	+0.04
	ResNet 34	PyTorch	0.9970	-
		ONNX	0.9964	-0.06
		TensorRT	0.9974	+0.04
		OpenCV DNN	0.9974	+0.04
	ResNet 50	PyTorch	0.9970	-
		ONNX	0.9764	-2.06
		TensorRT	0.9840	-1.30
		OpenCV DNN	0.9882	-0.88

다. TensorRT는 모델 최적화 과정을 위한 파라미터 설정 및 조정의 난이도가 높아 사용 편이성이 상대적으로 낮다. OpenCV DNN은 GPU 없이 CPU만으로도 동작이 가능하다는 장점을 가지지만, C++ 언어 기반이라는 한계를 가진다.

개발 용이성 면에서, ONNX 런타임은 디버깅 도구를 제공하여 개발 중 문제점 해결이 쉽다는 장점을 가진다. TensorRT는 최적화 작업의 어려움으로 인해 상대적으로 개발 난이도가 높고, OpenCV DNN은 C++ 언어 장벽으로 인해 Python과 같은 언어에 익숙한 개발자들에게 어렵게 느껴질 수 있다.

Ⅴ. 결론

본 논문은 번호판 탐지 및 문자 인식, 그리고 분류 태스크들에 대해 YOLO와 ResNet 기반 모델들을 세 가지 런타임에서 수행하면서 자원 사용, 성능, 그리고 사용성 측면에서 런타임들을 비교하였다. 그 결과, 자원 사용에 속하는 모델 로딩 시간과 메모리 사용량에서는 런타임 별로 차이를 보였으나, 정확도 성능 면에서는 유사한 수준을 보여, 런타임을 선택함에 있어 플랫폼의 가용 자원 여부가 가장 중요한 조건이 됨을 알 수 있었다. 향후 연구에서는 런타임과 태스크를 추가하여 비교분석을 확장하고자 한다. 예를 들어, 런타임에서는 Apache MXNet, Apple Core ML을 추가하고 태스크에서는 이미지 세그멘테이션과 포인트 클라우드 완성 등을 포함할 예정이다.

참 고 문 헌

[1] J. Georgiou, M. Yarally, S. Oliveira, J. Khan, D. Xu, S. Avila, and M. Ding, “Green AI: A Preliminary Empirical Study on Energy Consumption in DL Models Across Different Runtime Infrastructures,” arXiv Preprint, arXiv:2402.13640, 2024.

[2] J. Baek, Y. Ro, J. Jang, S. Jang, and J. Kim, “Method of Real-Time Indoor Localization and Monitoring in Parking Lot using Image Deep Learning Analysis,” Journal of the Institute of Electronics and Information Engineers, Vol. 60, No. 11, pp. 86-94, 2023.

[3] Guo, He, He, Lausen, Li, Lin, Shi, Wang, Xie, Zha, Zhang, Zhang, Zhang, Zhang, Zheng, and Zhu et al., “GluonCV and GluonNLP: Deep Learning in Computer Vision and Natural Language Processing,” Journal of Machine Learning Research, Vol. 21, No. 23, pp. 1-7, 2020.

[4] S. Maan, A. Tyagi, S. Kumar, and A. Kumar, “Security and Surveillance using Computer Vision,” International Journal of Computer Applications, Vol. 176, No. 27, pp. 30-35, 2020.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 779-788, 2016.

[6] J. Redmon, and A. Farhadi, “YOLO9000: Better, Faster, Stronger,” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 6517-6525, 2017.

[7] J. Redmon, and A. Farhadi, “YOLOv3: An Incremental Improvement,” arXiv Preprint, arXiv:1804.02767, 2018.

[8] A. Bochkovskiy, C. Wang, and H.M Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” arXiv Preprint, arXiv:2004.10934, 2020.

[9] C. Wang, A. Bochkovskiy, and H.M Liao, “YOLOv7: Trainable Bag-of-Freebies Sets New

State-of-the-Art for Real-Time Object Detectors," Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 7464-7475, 2023.

[10] Sikora, Malina, Kiac, Martinasek, Riha, Prinosil, Jirik, and Srivastava et al., "Artificial Intelligence-Based Surveillance System for Railway Crossing Traffic," IEEE Sensors Journal, Vol. 21, No. 14, pp. 15515-15526, 2021.

[11] M.O. Lawal, "Tomato Detection Based on Modified YOLOv3 Framework," Scientific Reports, Vol. 11, No. 1, 2021.

[12] M. Bae, J. Park, S. Jung, and C. Sim, "Fish Species and Disease Detection System Using Deep Learning-Based Object Detection Model," Journal of Korea Multimedia Society, Vol. 26, No. 8, pp. 898-910, 2023.

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

[14] J.R. Del Mar-Raave, H. Bahsi, L. Mršić, and K. Hausknecht, "A Machine Learning-Based Forensic Tool for Image Classification - A Design Science Approach," Forensic Science International: Digital Investigation, Vol. 38, pp. 1-13, 2021.

[15] J. Xu, "A Deep Learning Approach to Building an Intelligent Video Surveillance System," Multimedia Tools and Applications, Vol. 80, pp. 5495-5515, 2020.

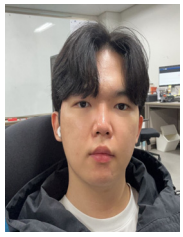
[16] S. Rhyou, H. Kim, and K. Cha, "Development of Access Management System based on Face Recognition using ResNet," Journal of Korea Multimedia Society, Vol. 22, No. 8, pp. 823-831, 2019.

[17] M. Sever, and S. Öğüt, "A Performance Study Depending on Execution Times of Various Frameworks in Machine Learning Inference," 2021 15th Turkish National Software Engineering Symposium, 2021.

[18] D. Choi, J. Lee, H. Hong, H. Kang, and H. Kim, "Performance Analysis of Deep Neural Networks through ONNX Conversion," The Institute of Electronics and Information Engineers, pp. 1085-1086, 2023.

[19] S. Kim, B. Han, and J. Heo, "Model Transformation and Inference of Machine Learning using Open Neural Network Format," The Journal of the Institute of Internet, Broadcasting and Communication, Vol. 21, No. 3, pp. 107-114, 2021.

저 자 소 개



김 정 현(Jeong-Hyeon Kim)

・2019년 3월～현재 : 인천대학교 임베디드시스템공학과 학사 과정

<관심분야> 컴퓨터 비전, 딥러닝, point cloud completion

이 다 은(Da-Eun Lee)

・2021년 3월～현재 : 인천대학교 임베디드시스템공학과 학사 과정

<관심분야> 컴퓨터 비전, 딥러닝, motion tracking

최 수 빈(Su-Been Choi)

・2022년 3월~현재 : 인천대학교 임베디드시스템공학과 학사 과정

<관심분야> 인공지능, 머신러닝, 딥러닝

전 경 구(Kyung-Koo Jun)

・1996년 2월 : 서강대학교(공학사)

・1998년 5월 : Purdue Univ (공학석사)

・2001년 5월 : Purdue Univ (공학박사)

・2004년 3월~현재 : 인천대학교 임베디드시스템공학과 교수

<관심분야> 딥러닝 기반 컴퓨터 비전