Jetson Nano 기반 Docker 환경에서 YOLOv8의 분석 및 구현

김태경, 오명훈
호남대학교, 호남대학교
rlaxorud2702@gmail.com, mhoh@honam.ac.kr

Analysis and Implementation of YOLOv8 in Jetson Nano-Based Docker Environment

Tae-Kyung Kim and Myeong-Hoon Oh
Dept. of Computer Engineering Honam University

요 약

본 논문은 NVIDIA Jetson Nano 환경에서 YOLOv8을 활용한 객체 인식 기반 실시간 모니터링 시스템 구현 방법을 제시한다. YOLOv8은 Ultralytics에서 개발한 최신 객체 감지 모델로, 이전 버전 대비 속도, 정확도, 연산 효율성 측면에서 균형 잡힌 성능을 제공하며,저전력 · 저비용 하드웨어 환경에서도 실시간 추론이 가능하다. 또한, 단순한 객체 감지뿐 아니라 분할, 자세 추정 등 다양한 컴퓨터 비전 작업을 지원하여 활용 범위가 넓다. 본 연구에서는 Jetson Nano의 내장 GPU를 활용하여 YOLOv8의 경량화 모델을 Docker 컨테이너 기반으로 실행함으로써 실시간 감시, 자동화 시스템, 로봇 비전 등 임베디드 시스템을 활용하는 환경에 적합한 구조를 구현하였다.

I. 서 론

최근 실시간 감시 시스템, 스마트 팜, 자율주행 차량 등 다양한 분야에서 실시간 객체 인식 기술의 사용이 증가하고 있다. 이러한 기술은 고속 데이터 처리와 낮은 지연 시간을 요구하며, 대용량 영상의 중앙 서버 전송 방식은 네트워크 부하와 응답 속도 저하라는 한계가 있다. 이에 따라 데이터를 현장에서 직접 처리하는 엣지 컴퓨팅과 Jetson Nano와 같은 저전력 임베디드 GPU 보드의 활용이 확대되고 있다.

Jetson Nano는 소형 폼 팩터와 낮은 소비 전력으로 실시간 AI 응용에 적합하지만, 최신 대규모 모델이나 고해상도 비전 작업에는 성능 한계가 있다. 이러한 한계로 인해 소프트웨어 호환성과 복잡한 환경 구성을 효율적으로 관리하기 위해 YOLOv8을 Docker 컨테이너 기반으로 Jetson Nano에서 운용하는 방식을 제안한다[1].

YOLOv8은 최신 딥러닝 객체 탐지 모델로, 향상된 백본과 넥(Neck) 아키텍처, 앵커 프리 구조, 독립 헤드 설계를 통해 복잡한 환경과 작은 객체 탐지에서도 뛰어난 성능을 보인다. 실시간 객체 탐지, 교통 관리, 영상 감시 등 다양한 산업 분야에서 활용되고 있으며, 다양한 크기의 경량화 모델(YOLO8n, YOLO8s 등)도 제공되어 저사양 엣지 디바이스에서도 실시간 인식이 가능하다.

Docker 컨테이너는 애플리케이션과 모든 종속성을 일관된 환경으로 패키징하여, 운영체제 및 패키지 환경에 관계없이 손쉽게 배포와 관리를 할 수 있다. 본 논문은 Jetson Nano 환경에서 YOLOv8을 컨테이너 기반으로 구현 시 얻을 수 있는 환경 재현성, 배포 용이성, 성능 안정성 등 다양한 이점을 제시한다.

II. 시스템 작동 설계

Jetson Nano 환경에 맞는 JetPack 및 CUDA 버전에 최신화된 베이스 이미지를 사용하고, 패키지 충돌을 방지하기 위해 Kitware 저장소를 삭제했다. 이후 Ubuntu 18.04 버전 저장소의 GPG 서명 오류를 우회하기 위해 apt 서명 검증을 비활성화하였다. 아래 표 1은 위 내용에 대한 도커 환경 설정이다.

표 1 . 베이스 이미지 및 패키지 관리 설정

```
FROM nvcr.io/nvidia/l4t-pytorch:r32.7.1-pth1.10-py3
RUN find /etc/apt/ -type f -exec sed -i '/kitware/d' {} ₩;
RUN echo 'Acquire::AllowInsecureRepositories "true";' >
/etc/apt/apt.conf.d/99insecure ₩
 && echo 'Acquire::AllowDowngradeToInsecureRepositories "true";'
>> /etc/apt/apt.conf.d/99insecure ₩
 && echo 'APT::Get::AllowUnauthenticated "true";' >>
/etc/apt/apt.conf.d/99insecure
```

Jetson Nano 환경에서 Python 3.8, OpenCv, Git등 AI 및 비전 개발에 필요한 주요 패키지와 시스템 도구들을 설치하고 작업 디렉터리를 /workspace로 설정하였다. 아래 표 2는 위 내용에 대한 도커 환경 설정이다.

표 2. 주요 패키지 설치 및 작업 디렉터리 설정

```
RUN apt-get update && apt-get install -y --allow-unauthenticated ₩
    locales ₩
    wget ₩
    git ₩
    python3-opencv ₩
    python3.8 ₩
    python3.8-distutils ₩
    python3.8-dev ₩
WORKDIR /workspace
```

Python 3.8 환경에서 pip를 설치하고, 프로젝트 의존성 패키지와 Ultralytics, YOLOv8 라이브러리를 설치한다. 이후 YOLOv8 사전학습 모델 파일을 다운로드하고, 호스트의 test.py 스크립트를 컨테이너로 복사한 후, 컨테이너 실행 시 자동으로 스크립트가 실행될 수 있도록 설정했다. 아래 표3은 위 내용에 대한 도커 환경 설정이다[1].

표 3. YOLOv8 실행을 위한 패키지 및 모델 설치

```
RUN wget https://bootstrap.pypa.io/pip/3.8/get-pip.py && python3.8
get-pip.py
COPY requirements.txt .
RUN python3.8 -m pip install --upgrade pip && python3.8 -m pip
install --no-cache-dir -r requirements.txt
RUN git clone https://github.com/ultralytics/ultralytics.git && ₩
    cd ultralytics && ₩
    python3.8 -m pip install .
RUN wget
https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov
8n.pt
RUN wget https://bootstrap.pypa.io/pip/3.8/get-pip.py && python3.8
get-pip.py
COPY test.py .
CMD ["python3.8", "test.py"]
```

표 1, 2, 3에 나온 코드를 하나의 도커 파일로 실행하면 아래 그림 1과 같이 실행된다. 왼쪽에는 객체 인식 탐지 결과가 뜨며 신뢰도, 각 프레임별 탐지된 객체 수, 클래스, 추론 시간, 후처리 시간이 출력되고 있으며, 오른쪽에는 캠을 통해 객체 인식이 되며 사전 학습된 모델과 실시간으로 인식 중인 객체의 정확도를 보여주고 있다.

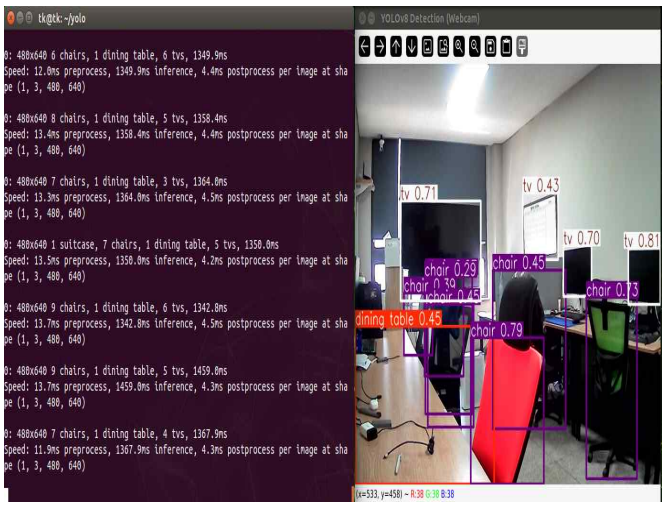

그림 1. 객체 인식 화면

Jetson Nano 보드에서 해당 이미지를 Docker 컨테이너를 통해 실행시켰을 때 네이티브(직접 설치) 환경에 비해서 정확도 및 속도에서의 차이가 없지만, Docker 환경에서 사용 시 복잡한 의존성 문제, 설치 오류를 크게 줄일 수 있으며, 별다른 패키지 설치 필요없이 모델을 사용할 수 있다. 또한, 환경 재현성과 이식성이 높아 다른장치로 이전하거나 재설치하는 경우에도 동일한 환경을 쉽게 구축할 수 있다는 장점이 존재한다[2]. 다음 표 4는 Docker를 사용한 환경과 Docker를 사용하지 않은 환경에서의 차이점을 비교한 표이다.

표 4. Docker 사용 여부에 다른 차이

	Docker 환경	네이티브 환경
정확도 및 추론 속도	동일(차이 없음)	
환경 구축	의존성 문제 및 설치 오류 최소화로 안정적	의존성 충돌 및 오류 발생 가능성 높음
패키지 설치	설치 없이 바로 사용	개별 패키지 수동 설치
환경 재현성	동일 환경 복제 가능	장치마다 재설정 필요
이식성	동일 환경 구축 가능	환경 재구축 필요

추가적으로 Jetson Nano 보드에서 YOLO 버전별로 성능 차이를 비교하는 실험을 진행하였다. Jetson Nano는 GPU 추론 시 CPU 추론과 다르게 GPU 전용 최적화 도구를 활용한 행렬 연산과 텐서 처리를 통해 CPU 기반 시스템을 능가하는 성능을 보여주었으며, AI 워크로드에 필요한 하드웨어 가속성이 높다는 것을 알았다[3],[4]. 이를 통해 YOLOv8n, YOLOv8s 모델이 기존 YOLOv5 모델에 비해서 추론 속도 및 정확도 면에서 향상된 결과를 나타내는 것을 알 수 있었다[5],[6]. 다음 표 5에서는 GPU를 활용한 YOLO 모델에 따른 성능 차이를 정리한 표다.

표 5. YOLO 모델별 성능 차이

모델	평균 FPS	GPU 사용률	mAP(%)	메모리 사용량
YOLOv5s	9.1 FPS	81%	84.3%	2.3 GB
YOLOv8n	13.2 FPS	78%	87.1%	2.4 GB
YOLOv8s	10.6 FPS	85%	88.4%	2.7 GB

Ⅳ. 결 론

본 연구에서는 Jetson Nano 환경에서 YOLOv8 객체 탐지 모델을 안정적으로 실행하기 위해, 도커 파일을 활용한 컨테이너 기반 시스템을 개발하였다. Docker 컨테이너 환경과 네이티브 환경의 추론 속도와 정확도는 거의 동일하지만 환경 구축, 이식성, 환경 재현성, 운영 및 개발 편의성에서 우수함을 알 수 있었다.

참 고 문 헌

[1] Ultralytics, "물체 감지: YOLOv8 사전 학습 모델," Ultralytics 공식 블로그, 2024 https://www.ultralytics.com/ko/blog/object-detection-with-a-pre-trained-ultralytics-yolov8-model

[2] 에지 디바이스에서의 가상화 환경에 따른 YOLOv8 추론 성능 비교, 2024 https://www.dbpia.co.kr/journal/articleDetail?nodeId=NODE11861818#a

[3] Unlocking the Power of NVIDIA Jetson Nano Developer Kit for AI and Robotics https://www.fibermall.com/blog/nvidia-ai-board.htm

[4] Implementing NVIDIA Jetson for AI-powered automation https://industrialautomation.nl/en/software-it-ot-cybersecurity/implementing-nvidia-jetson-for-ai-powered-automation/

[5] 딥러닝 알고리즘 기반 보행자 감지 시스템 Pedestrian Detection System Based on Deep Learning Algorithm https://koreascience.or.kr/article/JAKO202407261208187.page?lang=ko

[6] YOLOv8 YOLOv5: 상세 비교 https://docs.ultralytics.com/ko/compare/yolov8-vs-yolov5/#performance-comparison