임베디드 장치에서 인공 신경망을 사용한 차선 탐지 최적화 성능 분석

이광배°, 신동군
성균관대학교 전자전기컴퓨터공학과
Kblee1022@gmail.com, dongkun@skku.edu

Lane Detection Optimization Performance Analysis Using Artificial Neural Network in Embedded Device

Kwangbae Lee°, Dongkun Shin
Department of Electrical and Computer Engineering, Sungkyunkwan University

요 약

최근 영상 데이터, 딥 러닝 등을 처리할 수 있는 임베디드 장치가 많아지고 있다. 자율주행 자동차는 실시간 탐지 및 프로세싱이 필요하지만, 임베디드 장치에서 기존의 방식을 사용한 차선 탐지는 실시간으로 수행되기 어렵다. 또한, 딥 러닝은 연산이 많기 때문에 저사양 장치에서 수행하기 위해서는 최적화해야 한다. 본 논문에서는 Odroid-XU3에서 ACL(Arm Compute Library)를 사용해 딥 러닝 기반의 차선 탐색 기능의 최적화를 제안한다. 실험 결과, 임베디드 장치에서 최적화를 통해 76%의 정확도로 5 FPS의 inference가 가능한 것을 확인했다.

1. 서 론

ADAS(Advanced Driver Assistance System)는 차선 사이에 차량을 배치하는 “차선 탐지”, 앞차와의 간격을 유지하는 “차량 간격 유지” 등 차량 운전 보조 기능의 집합이다. 기존 ADAS는 그림 1과 같이 컴퓨터 비전 기반이 대부분이었으나, 정확도 향상에 한계가 있고 복수의 카메라를 활용한 데이터가 필요하다는 문제점이 있다. 따라서, 최근 ADAS 기능을 구현하는데는 단일 카메라 데이터만으로도 높은 정확도를 달성할 수 있는 딥 러닝이 많이 사용되고 있다[1].

최근에는 고성능 모바일 AP와 GPU가 개발되면서 엣지 디바이스의 성능이 높아졌지만, 여전히 영상 데이터를 활용하는 딥 러닝은 많은 수의 연산이 필요하기 때문에, 실시간으로 딥 러닝을 처리하려면 클라우드 서버를 활용해야 한다. 그러나 클라우드 서버와의 통신은 지연 시간을 예측할 수 없으며 연결도 불안정하기 때문에, 실시간 전달과 정확성이 상당히 중요한 자율주행 자동차에는 치명적이다.

본 논문에서는 영상 데이터로부터 차선을 인식하고 탐지하는 과정을 딥 러닝을 통해 수행한다. 이러한 딥 러닝 모델을 Arm Compute Library가 제공하는 멀티 쓰레드와 NEON 명령어 기반 딥 러닝 가속 기능을 사용해 추론 속도를 높이는 것을 제안한다. 실험 결과 Odroid-XU3 장치에서 가속을 하지 않은 딥 러닝 기반의 차선 탐지는 초당 3 프레임 정도의 스피드로 inference가 가능하다. 가속 기법을 적용한 딥 러닝을 통한 차선 탐지 과정은 초당 6 프레임 정도로 향상되는 것을 확인했다.

"본 연구는 미래창조과학부 및 정보통신기술진흥센터의 SW 컴퓨팅산업원천기술개발사업(SW스타랩)의 연구결과로 수행되었음" (IITP-2017-0-00914)

2. 배 경

2.1 Lane Detection process

컴퓨터 비전 기반의 차선 탐색은 그림 1과 같은 프로세스로 진행된다[2]. 카메라 센서로부터 획득한 Raw Data를 입력받아, 각 프로세스는 컴퓨터 비전 기술을 기반으로 진행된다. Camera Calibration 단계부터 Binarize 단계까지는 입력 이미지에 대한 Pre-processing이며, Lane Line Extraction과 Road Left and Right Curvature Calculation 과정은 실제로 차선과 곡률을 추론하는 과정이다..

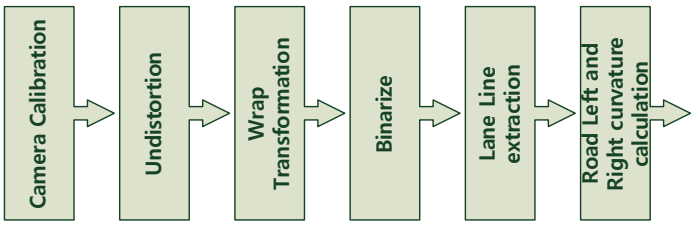

그림 1. 컴퓨터 비전 기반 차선 탐지 프로세스

6개 단계의 차선 탐지 프로세스 중에서 일부 단계만 딥 러닝으로 대체 가능하다[3]. 카메라 센서로부터 획득한 데이터는 Raw Data이기 때문에, 전반부 4개의 컴퓨터 비전 기반 Pre-processing

과정이 진행되어야 추론이 가능하다. 마지막 2개 단계에서 차선과 곡률을 추론하는 과정은 딥 러닝으로 대체 가능하다.

2.2 Arm Compute Library

Arm Compute Library[4]는 ARM 디바이스에서 딥 러닝 연산 최적화를 위해 제공하는 라이브러리다. 이 라이브러리는 OpenGL 기반 GPU 가속과 SIMD 명령어인 NEON을 지원한다[5].

OpenCL을 활용한 딥 러닝 연산으로는 GEMM(General Matrix Multiplication)과 Direct라는 두 가지 방법을 제공한다. GEMM 방법은 컨볼루션 레이어를 연산을 위해 im2col을 하고, 가중치 matrix를 reshape 한 후 연산이 진행된다. Direct 방법은 im2col을 사용하지 않고, 컨볼루션 레이어의 변수와 가중치 matrix의 변수를 하나씩 연산한다.

ACL을 사용하여 직접 신경망 구조를 만들고, 제공되어 있는 커널을 빌드해 실행 가능하다.

2.3 신경망 구조

차선 탐지에 필요한 상관계수(coefficient)를 구하기 위한 신경망의 구조는 그림 2 와 같다. 영상 정보를 720*1280 차원의 프레임 단위로 분할하여 입력으로 사용한다. 입력 데이터가 처음 지나는 배치 표준화 레이어(Batch normalization layer)는 각 층의 입력을 표준화하고, 데이터의 분포를 일정하게 만든다. 표준화를 거친 데이터는 128 개의 필터, 스트라이드 1, 패딩 없이 이루어진 첫 번째 컨볼루션 레이어(Convolution layer)를 통과한다. 이후 최대 풀링 레이어(Max pooling layer)를 통해서 컨볼루션 층의 결과로 나온 이미지의 사이즈를 줄인다. 컨볼루션 레이어나 최대 풀링 레이어를 반복적으로 거치고 나면 이미지의 주요 특징만 추출되며, 결과값으로 나오는 feature map 은 2 차원이다. 2 차원 데이터를 완전 연결 레이어(Fully connected layer)의 입력으로 전달하기 위해서는 1 차원의 데이터로 변환해야 한다. 이 과정을 위해서 영상을 1 차원의 데이터로 변환해주는 Flatten layer 를 사용한다. Flatten layer 를 통해 변환된 1 차원의 데이터가 4 개의 완전 연결 레이어를 차례로 통과해서 최종 결과를 도출해낸다. 과적합을 방지하기 위해 첫 번째 컨볼루션 레이어와 2 개의 완전 연결 레이어를 지나면서 0.5 비율의 드롭아웃을 사용한다. 드롭아웃을 통해 정규화 효과를 얻을 수 있다. 모든 레이어의 활성함수로는 ReLU 함수를 사용한다. 활성화 함수로 ReLU 를 사용하는 이유는 다른 활성화 함수를 사용할 경우보다 빠르고 더 효과적이기 때문이다.[6]

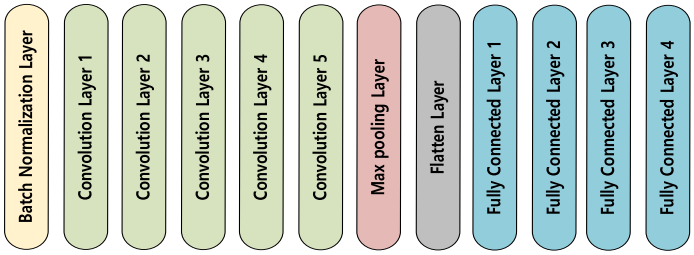

그림 2. 신경망 구조

3. 실 험

3.1 실험 환경

본 논문에서 임베디드 장치로는 Odroid-XU3[7]을 사용한다. Odroid-XU3은 Cortex™-A15 2Ghz 와 Cortex™-A7 Octa core CPU, Mali-T628 MP6 GPU를 탑재한 모바일 AP인 Samsung Exynos 5422를 사용한다.

Nvidia Titan V 기반 서버에서 딥 러닝 모델을 훈련하였고, 딥 러닝 프레임워크로는 Keras를, 백엔드로는 Tensorflow를 사용하였다. 임베디드 장치에서 딥 러닝 모델을 동작하기 위해 Arm Compute Library 18.03을 사용하였다.

신경망을 학습시키기 위한 데이터 셋으로는 Udacity에서 제공하는 실제 도로 주행 영상[8]을 사용하였다.

3.2 실험 결과

그림 3는 CPU만을 사용하는 전제 하에 Odroid-XU3에서 Arm Compute Library 사용 유무와 쓰레드 개수 조절에 따른 성능을 나타내는 그래프다. 쓰레드 4개를 사용할 때 가장 높은 성능을 보이는 이유는 Odroid-XU3의 CPU 코어가 4개의 Big 코어와 4개의 Little로 구분되어 있기 때문이다. 쓰레드의 수가 5개를 넘어가면, 4개의 Big 코어와 1개의 Little 코어가 사용되기 때문에 Big core가 Little 코어의 연산이 끝나기를 기다린다. 또한 NEON 방법을 사용하여 성능 향상이 나타난 것을 보인다.

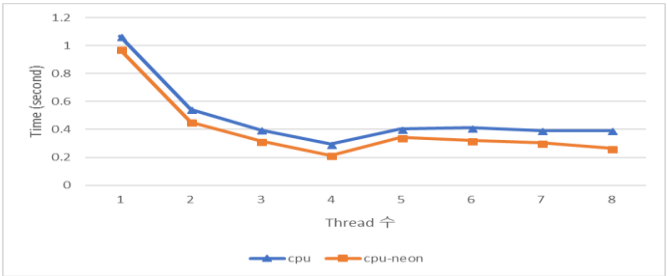

그림 3. Thread 개수 별 ACL 기반 차선 탐지 모델의 구동 시간

그림 4은 ACL을 사용했을 경우에 전력 소모량을 나타낸다. NEON을 사용했을 때는 CPU의 사용량이 거의 전부를 차지한다. OpenCL을 사용한 GPU 가속의 경우에는 CPU를 주로 사용하는 연산

방법보다 전체적인 전력이 절반 정도로 줄었다. NEON, OpenCL 등 ACL을 사용한 방식의 전력이 줄어든 이유는 기존보다 inference 시간이 줄었기 때문이다.

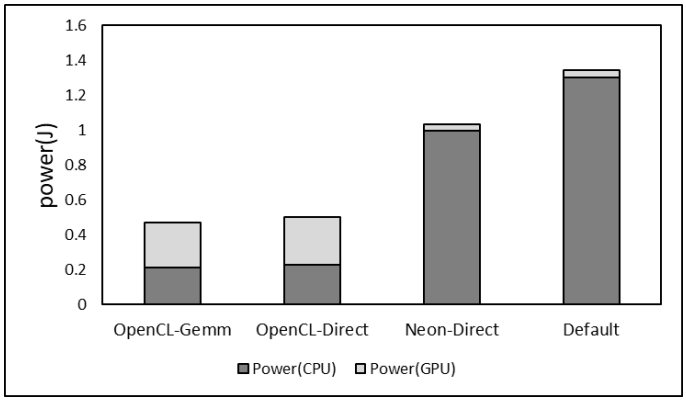

그림 4. ACL 방법에 따른 전력(J) 소모량

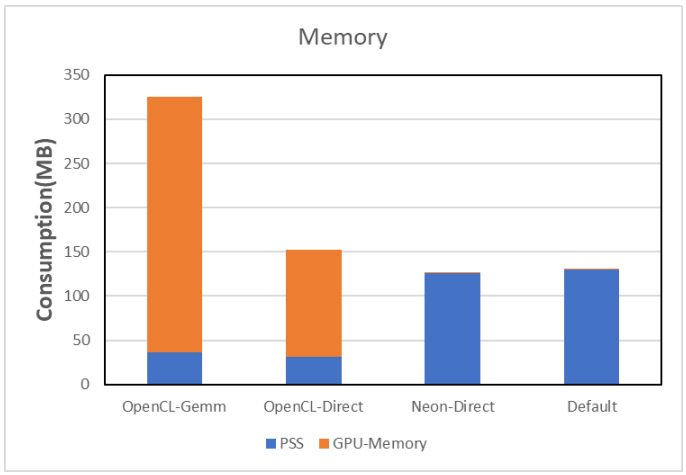

그림 5. ACL 방법에 따른 메모리 사용량

그림 5은 ACL을 사용했을 경우에 메모리 사용량을 나타낸다. GPU를 사용하는 OpenCL의 경우 가중치등의 연산에 필요한 데이터가 GPU-Driver가 할당받은 메모리를 사용한다. OpenCL GEMM 방법(325MB)을 이용하면 연산에 필요한 데이터를 모두 메모리에 올려놓기 때문에 OpenCL Direct 방법(153MB)보다 메모리 사용량이 높다. NEON을 사용했을 경우에는 가속 기법을 적용하지 않았을 때(130MB)와 비교했을 때 비슷한 메모리 사용량(125MB)를 보인다. 그 이유는 NEON 연산을 할 때 필요한 데이터들은 모두 기존과 같은 방식으로 가져와야 하기 때문이다.

그림 6은 Arm Compute Library 에서 제공하는 딥 러닝 연산 방법 별 성능을 측정한 그래프다. Arm에서의 딥 러닝 연산 가속이 없는 Keras를 사용할 때는 3 FPS 정도의 추론이 가능하다. OpenCL을 사용한 GPU 가속의 경우 6 FPS, SIMD 연산을 사용한 NEON을 사용한 경우 5 FPS 성능 향상이 가능하다.

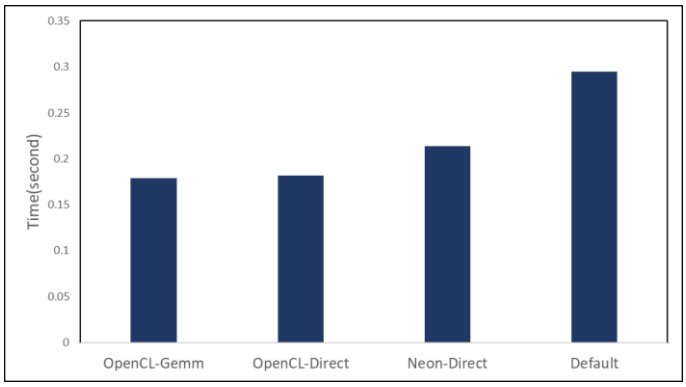

그림 6. ACL의 딥 러닝 연산 방법에 따른 Inference 시간

5. 결 론

이 논문에서는 실시간 프로세싱을 목표로한 딥 러닝 기반의 차선 탐지를 Odroid-XU3 에서 가속하는 방법에 대하여 분석하고 탐색하였다.

실시간 도출이 가능하게 하기 위해서 차선 탐색 프로세스 중 다른 부분도 딥 러닝을 사용할 수 있고[9], 더 빠른 성능을 달성하기 위해 tensor decomposition 등의 방법을 활용할 수도 있다. 차선 탐지를 실시간으로 도출 할 수 있게 최적화 하는 방향으로 앞으로의 연구를 진행할 예정이다.

참 고 문 헌

[1] Huval, Brody, et al., "An empirical evaluation of deep learning on highway driving," arXiv preprint, 2015.
[2] Zhou, Yong, et al., "A robust lane detection and tracking method based on computer vision," Measurement science and technology 17.4, 2006.
[3] https://github.com/jsistla/adv-lane-lines
[4] Arm Compute Library - https://github.com/ARM-software/ComputeLibrary
[5] Arm Community - https://community.arm.com/
[6] Deep Learning model - https://github.com/mvirgo/MLND-Capstone
[7] Odorid - www.hardkernel.com/
[8] Udacity - https://github.com/udacity/self-driving-car/tree/master/datasets
[9] Neven, Davy, et al., "Towards End-to-End Lane Detection: an Instance Segmentation Approach," arXiv preprint, 2018.