임베디드 시스템용 딥러닝 추론엔진 기술 동향

Trends in Deep Learning Inference Engines for Embedded Systems

유승목 (Seung-mok Yoo, yoos@etri.re.kr) 고성능디바이스SW연구실 책임연구원
이경희 (Kyung Hee Lee, kyunghee@etri.re.kr) 고성능디바이스SW연구실 책임연구원
박재복 (Jaebok Park, parkjb@etri.re.kr) 고성능디바이스SW연구실 선임연구원
윤석진 (Seok Jin Yoon, sjyoon@etri.re.kr) 고성능디바이스SW연구실 책임연구원
조창식 (Changsik Cho, cscho@etri.re.kr) 고성능디바이스SW연구실 책임연구원/실장
정영준 (Yung Joon Jung, jjing@etri.re.kr) 차세대시스템SW연구실 책임연구원
조일연 (Il Yeon Cho, iycho@etri.re.kr) 초성능컴퓨팅연구본부 책임연구원

ABSTRACT

Deep learning is a hot topic in both academic and industrial fields. Deep learning applications can be categorized into two areas. The first category involves applications such as Google Alpha Go using interfaces with human operators to run complicated inference engines in high-performance servers. The second category includes embedded applications for mobile Internet-of-Things devices, automotive vehicles, etc. Owing to the characteristics of the deployment environment, applications in the second category should be bounded by certain H/W and S/W restrictions depending on their running environment. For example, image recognition in an autonomous vehicle requires low latency, while that on a mobile device requires low power consumption. In this paper, we describe issues faced by embedded applications and review popular inference engines. We also introduce a project that is being development to satisfy the H/W and S/W requirements.

KEYWORDS deep learning, neural networks, embedded system

Ⅰ. 서론

2016년 Google의 Alpha Go가 이세돌과 바둑시합에서 승리함으로써 한국뿐 아니라 전 세계에 커다란 충격을 안겨 주었다. 이후 인공지능(혹은 AI)이란 전문가들뿐 아니라 일반인들에게도 일상적인 용어가 되었고, 이에 기반한 미래사회의 재편이 다가왔음이 끊임없이 언론을 통해 보도되고 있다.

2016년 이전에 공개된 Apple의 Siri나 Amazon Alexa는 AI를 이용한 장난감 정도로 취급되던 것

* DOI: https://doi.org/10.22648/ETRI.2019.J.340403
* 이 논문은 2019년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임(No. 2017-0-00068, 운전자 주행경험 모사기반 일반도로환경의 자율주행4단계(SAE)를 지원하는 주행판단엔진 개발).

들에서, 이 대국 이후에는 미래 컴퓨팅 플랫폼의 중심축으로 바라보는 인식변화가 이루어지고 있다. 이들의 동작방식은 단말기에서 음성 및 기본 정보를 수집하여 고성능 서버로 보내면, 입력값들을 기계학습 모델을 이용하여 추론 후 입력에 상응하는 결과를 단말기로 되돌려주는 방식을 취한다.

위와는 다른 영역으로 자율주행자동차와 같이 좀 더 복잡한 방식의 연구들이 상용화를 눈앞에 두고 있다. 이들은 Apple Siri와는 달리 운행 환경에 영향을 받으며 응답속도와 같이 특정 제한 조건이 전체 성능에 영향을 미친다. 이외 모바일 환경에서 적용 가능한 AI 서비스들도 다시금 부각되고 있는데, 이들은 서버기반 서비스에 비해서 크기나 전력 같은 다른 요구사항들을 가진다. 본 고에서는 AI 분야들 중 자율주행자동차와 같이 적용분야의 특수성으로 인해 AI 처리를 위한 서버를 고려하지 않고 단말단에서 AI 추론을 처리해야 하는 임베디드 시스템에 적용 가능한 AI 추론 프레임워크를 살펴보고자 한다.

임베디드 환경에서는 서버기반이나 PC기반의 개발/실행 환경과는 달리, 크기 및 동작에 물리적 제한이 존재한다. 연산을 위한 CPU가 서버나 PC 환경과 다를 수 있다. 이는 적용될 최종 서비스에서 물리적 크기의 제약성이나 넓은 동작온도와 같은 환경적인 요구사항을 우선적으로 만족해야 하기 때문이다. 이와 함께 모바일 장비와 같은 경우 저전력 운용에 대한 요구사항이 존재할 수 있다. 이는 H/W뿐 아니라 S/W 제약성으로도 작용한다. 자율주행자동차와 같은 경우 입력에 대한 빠른 응답속도에 대한 제한도 존재한다.

위 요구사항들을 만족하기 위해서 임베디드용 딥러닝 프레임워크는 추론과정에 최적화되어 있다. 개발 단계에서는 실행과는 다른 프레임워크가 사용될 수 있기 때문에, 개발 효율성과 개발된 모델의 범용성을 위해 프레임워크 간의 상호 연동이 필수적으로 고려되어야 한다.

임베디드 시스템의 특성상 다양한 프로세서와 GPU, FPGA와 같은 연산가속기를 사용할 수 있다. 딥러닝 모델 실행 시, H/W 독립적으로 동작하기 위해선 표준화 혹은 표준에 준하는 H/W 연산을 지원하는 플랫폼이 필요하다.

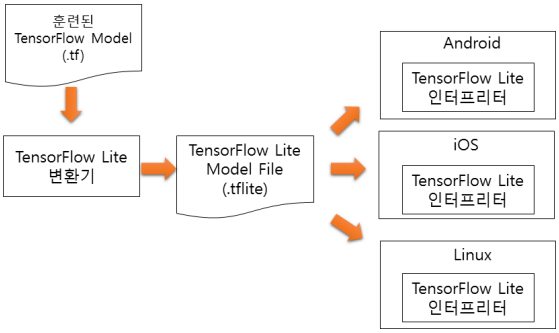
본 고에서는 임베디드 환경에서 동작 가능한 주요 딥러닝 추론엔진 프레임워크의 특징을 살펴보고자 한다.

Ⅱ. 임베디드용 딥러닝 추론 프레임워크

딥러닝 프로그램 개발과정은 수집된 데이터로부터 학습을 통해 신경망 모델을 만드는 과정과 이를 기반으로 실제 데이터를 입력하여 추론하는 과정으로 나뉜다. 학습과정은 방대한 양의 데이터를 이용해 장시간에 걸친 반복된 계산과정을 반복하기 때문에 빠른 프로세싱 파워와 큰 메모리를 요구한다. 반면, 실제 데이터를 이용한 동작환경에서는 일반적인 응용프로그램에 비해 많은 연산과 메모리를 사용하지만, 학습단계와 비교해서는 상대적으로 적은 정도의 연산량과 메모리를 요구한다. 실제 적용단계에서는 동작환경에 대한 물리적 크기와 전력의 제한이 요구되기 때문에, 개발단계에서 연산에 중점된 모델을 사용하기보다는 실행에 효율적인 프레임워크를 사용하는 추세로 변하고 있다.

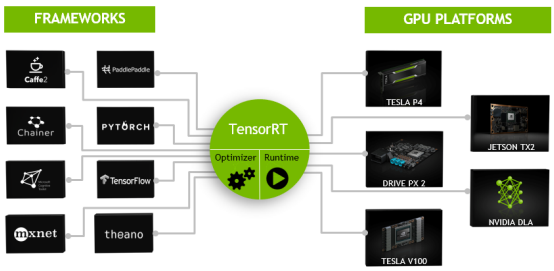
1. TensorFlow Lite

TensorFlow Lite는 Google이 2017년에 TensorFlow를 기반으로 하여 모바일과 임베디드 분야용



출처 TensorFlow, https://www.tensorflow.org/lite/guide?hl=ko. CC BY 4.0.

그림 1 TensorFlow Lite의 구조



출처 Nvidia TensorRT, https://developer.nvidia.com/tensorrt

그림 2 TensorRT 지원 프레임워크/플랫폼

으로 개발한 딥러닝 프레임워크다[1]. TensorFlow Lite는 TensorFlow로 개발된 모델을 대상 임베디드 시스템용으로 크기 및 용량을 축소 변환하여 동작시킨다. TensorFlow Lite는 학습된 TensorFlow 모델을 사용 환경에 맞도록 TensorFlow Lite 변환기(Converter)를 통해서 TensorFlow Lite용 모델로 변환하며, 이 모델을 디바이스에 내장된 TensorFlow Lite 인터프리터가 해석하여 추론을 수행한다. TensorFlow Lite 변환기는 모델의 크기와 성능 최적화 기능도 포함한다.

TensorFlow Lite 변환기는 기존에 개발된 다양한 TensorFlow 모델을 가져다 그대로 사용할 수 있도록 허용한다. 변환기는 양자화 과정을 포함하여 신경망 모델의 크기를 줄이면서 성능은 기존과 유사하게 유지해 준다. 생성된 신경망 모델은 TensorFlow Lite 인터프리터상에서 동작하기 때문에 이식성이 좋으며, 하위단의 OS나 H/W에 독립적이다.

TensorFlow Lite 인터프리터는 대상 임베디드 장치에 최적화되도록 개발되어 있으며, 시스템에 따라 GPU와 같은 하드웨어 가속 장치를 지원하여 실행 시간과 결과에 높은 성능을 제공한다.

현재는 추론 기능만 제공하고 있으나 향후 디바이스상에서 학습 기능도 제공할 계획이다.

2. TensorRT

TensorRT는 Nvidia가 CPU 혹은 자체 GPU기반 플랫폼을 위해 개발한 딥러닝 프레임워크로 학습된 신경망을 최적화하는 프로그램(optimizer)과 런타임 실행엔진을 포함하는 기계학습 추론엔진으로, 고성능과 짧은 응답 대기시간을 특징으로 하고 있다[2].

TensorRT는 Nvidia GPU를 사용하여 대규모 데이터를 효과적으로 처리할 수 있는 다중 연산 및 초고속 병렬 연산을 지원함으로써, CPU만을 사용했을 때보다 최대 40배 빠른 성능이 측정되었다. 컴퓨터 비전, 신경망 기반 기계 번역, 자동 음성 인식, 음성 합성 및 추천 시스템 등 다양한 인공지능 분야에 사용되고 있으며, 상대적으로 고성능이 요구되는 하이퍼스케일 데이터센터, 임베디드 및 자율주행 GPU 플랫폼에서 학습을 거친 신경망을 배포할 수 있는 플랫폼을 제공한다.

TensorRT는 int8과 fp16의 최적화를 통하여 추론을 매우 빠른 속도로 수행하며, 자율주행자동차에서의 물체 감지 및 경로 계획 등 응답 지연이 중요한 임베디드 응용에서 강점을 보인다. Nvidia는 다양한 GPU 아키텍처군을 발표함에 따라, 각 아키텍처 특성에 맞추어 최적화를 계속 진행하고 있다.

TensorRT는 ONNX(Open Neural Network Ex-

change) 파서 및 런타임을 포함하고 있어서, ONNX 상호 연동성을 제공하는 Caffe2, Microsoft Cognitive Toolkit, MXNet, PyTorch 신경망 프레임워크에서 학습된 딥러닝 모델도 TensorRT에서 동작 가능하다.

TensorRT는 기계학습에 가장 많이 사용되는 Google TensorFlow 프레임워크와 공학 계열에서 많이 사용하는 MathWorks Matlab의 신경망 프레임워크에 통합되어 있다. Nvidia GPU기반의 Jetson, Drive 및 Tesla 계열의 플랫폼을 지원한다.

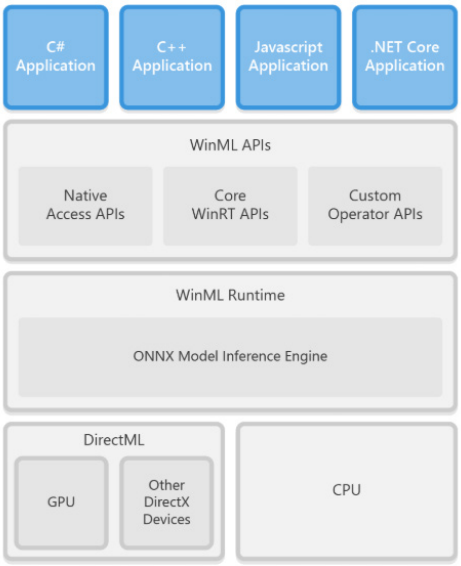
TensorRT는 x86에서만 Python, C++ 두 언어를 지원하고, 그 외의 H/W 플랫폼에서는 C++만을 지원한다. 사용자가 직접 정의한 기계학습 프레임워크에서 학습 및 추론을 수행할 경우 TensorRT C++ API를 사용해야 한다.

3. Windows ML

Windows ML(혹은 WinML)은 Microsoft에서 제공하는 신경망 엔진으로 MS-Windows Server 2019와 MS-Windows 10 운영체제에서 지원한다[3]. WinML은 C++, C#, JavaScript 등 다양한 프로그래밍 언어 인터페이스를 제공한다. WinML의 구조는 그림 3과 같다.

WinML API는 core API, custom operator, native API로 구성된다. Core API는 WinML의 주요 API로서 신경망 모델을 읽어 들이고 처리하는 것과 관련된 함수들이다. Custom operator는 기본적으로 제공되는 신경망 연산 함수외 사용자가 정의한 연산자를 사용할 수 있도록 지원하는 라이브러리이다. Native API는 Direct3D 등의 API를 통해 하드웨어와 인터페이스를 제공하는 함수를 제공하고 있다.

WinML은 ONNX를 기본 신경망 포맷으로 사용하고 있다. ONNX 규격으로 표현된 딥러닝 모델



출처 https://docs.microsoft.com/en-us/windows/ai/windows-ml

그림 3 WinML의 구조

은 WinML에서 실행 가능한 프로그램으로 최종 변환한 후, 타겟 디바이스를 통해 실행된다.

WinML은 기본적으로 CPU기반 코드를 생성하며, DirectML을 통해 GPU와 같은 가속 디바이스를 사용할 수 있도록 하고 있다. DirectML은 GPU 하드웨어 가속 인터페이스인 DirectX의 확장 버전으로 신경망 모델이 고속으로 실행될 수 있도록 지원하는 low-level 사용자 인터페이스다. DirectML은 신경망 모델이 대상 하드웨어에 최적화되도록 지원하며, 하드웨어 자원 관리를 통해 신경망 실행 태스크의 스케줄링과 같은 기능을 제공한다.

4. Core ML

Core ML은 Apple이 iOS 응용 개발자가 기계학습 기술을 그들의 응용 프로그램에 통합하기 위해서 제공하는 프레임워크로 개발환경은 Apple의 Mac OS X로 한정되며 개발된 응용프로그램의 동작 기기도 Apple 제품으로 한정된다[4]. Core ML은 앞

에서 소개한 다른 엔진들과는 달리 Support Vector Machine이나 Random Forest와 같은 다양한 종류의 기계학습 모델을 지원한다.

Core ML에서 추론은 단지 짧은 코드의 삽입으로 실행 가능하다. iOS11 이전에는 학습된 모델을 가져와 Accelerate[5]나 MPS(Metal Performance Shaders)와 같은 기존 프레임워크를 사용할 수 있도록 이식하는 작업이 요구되었다. 특히 메모리 사용을 많이 하는 CPU 작업에는 Accelerate를 사용하고, 연산량이 많이 요구되어 GPU를 사용해야 하는 작업에는 MPS를 사용할 수 있다. Core ML은 많은 세부 사항을 추상화하여 처리한다. iOS11은 얼굴감지, 객체추적, 언어번역 및 개체명 인식과 같이 이미지 및 텍스트 데이터로 동작할 때 사용할 수 있는 일반적인 작업들을 도메인특화 레이어로 확장되었다.

Core ML은 자체에 변환 툴을 제공하고 있어서 다른 플랫폼에서 개발된 신경망 모델을 MXNet converter나 TensorFlow converter를 이용해서 Core ML 모델 포맷으로 변환 가능하다.

Apple은 다른 프레임워크와 쉽게 사용할 수 있도록 오픈소스 및 모듈식 패키지 형태로 제공하고 있다.

III. 프레임워크 간 상호 연동을 위한 신경망 데이터 포맷

대부분의 신경망 관련 개발에서 신경망과 학습 데이터를 저장하거나 이 모델의 공유 시 자체포맷을 사용한다. 학습을 통해 신경망 모델을 개발하거나 개발된 학습 모델을 이용하는 응용 개발자 간에 상이한 개발 플랫폼을 사용하는 경우 개발에 따른 어려움이 발생한다. 이를 해결하기 위해 서로 다른 신경망 프레임워크 간에 학습 모델의 상호 호환을 위한 연구와 표준화가 진행되고 있다.

1. NNEF

Khronos 그룹에서는 서로 다른 신경망 프레임워크 간에 신경망 정보를 주고받기 위한 상호 호환성 제공을 위해 표준화를 진행해 왔고, 2018년 NNEF(Neural Network Exchange Format) 버전 1.0을 발표하였다[6].

NNEF는 네트워크 구조와 데이터파일을 분리한다. 특히, NNEF는 텐서 개념을 확장하여 프레그먼트와 신경망 연산에 필요한 함수 및 구문을 정의하고 있다. 이는 네트워크 구조 또는 개별 매개변수 데이터에 대한 독립적인 액세스를 지원하며, 파일은 tar 또는 zip과 같은 선택적 압축 및 암호화를 제공한다.

NNEF는 일정한 형식으로 네트워크를 구성하는 간편화된 방법을 제공하며, 이종 플랫폼 간에도 통일된 형태의 신경망 모델을 개발할 수 있는 방안을 제공한다.

2. ONNX

ONNX는 2017년 Facebook과 Microsoft 같은 AI 관련 회사가 신경망 프레임워크 간의 상호 연동을 위해 제안한 딥러닝 모델의 표현 방식이다[7]. ONNX를 지원하는 단체는 NNEF와 달리, S/W 회사뿐 아니라 Intel, Nvidia과 같은 H/W 회사도 참여하고 있다. 덕분에 ONNX는 WinML, PyTorch 등 다수의 딥러닝 프레임워크에서 지원되고 있고, 여러 플랫폼 상에서 최적화가 진행되어 있다. NNEF에 비해 다수의 툴들이 존재하며, 코드화 및 전체적인 활용도 면에서는 NNEF에 비해 우위를 점하고 있다.

딥러닝 모델은 계산 그래프로 표현되어 동작한다. 이러한 프레임워크의 그래프들은 특정장치(CPU, GPU, FPGA 등)에서 실행되도록 최적화 및

표 1 NNEF와 ONNX의 비교

	NNEF	ONNX
개방	비영리 단체의 안정적인 오픈소스 구현	오픈소스 프로젝트
형식	텍스트 기반의 절차형식	데이터구조 지향적인 Protobuf
오퍼레이션	Flat 형식으로 네트워크 표현. 복합 오퍼레이션 표현 가능	Flat 형식으로 네트워크 표현
흐름 구조	절차구문으로 동적그래프 표현	컨트롤플로우를 통한 동적그래프 표현
데이터	개념수준에서 양자화에 접근하여 추론 가속화를 위한 유연한 최적화 지원	Tensor의 구체적인 데이터 유형 사용

출처 https://www.khronos.org/blog/nnef-and-onnx-similarities-and-differences[9]

변환을 위해 중간코드인 IR(Intermediate Representation) 생성단계를 거친다[8]. IR 포맷은 연산 그래프를 공통적으로 표현함으로써, ONNX는 개발자가 자신의 작업에 적합한 프레임워크를 선택하고 사용자가 혁신적인 기능 향상에 집중할 수 있도록 지원하고 하드웨어 공급 업체는 플랫폼 최적화를 간소화할 수 있게 한다.

특히 ONNX는 신경망의 컴퓨팅 그래프에 대한 형식은 물론 그래프 안에 사용하는 광범위한 연산자 목록에 대한 포맷까지도 정의할 수 있다.이는 ONNX가 신경망뿐 아니라 ONNX-ML이라는 이름으로 전통적인 ML 모델도 표현할 수 있도록 지원한다.

IV. 국내 기술 동향

ETRI는 자율주행자동차에서 필수적인 딥러닝 영상인식을 효율적으로 처리하기 위한 개방형 병렬 컴퓨팅 프레임워크를 개발하고 있다. 개발 중인 개방형 병렬 컴퓨팅 프레임워크는 Intel의 PlaidML

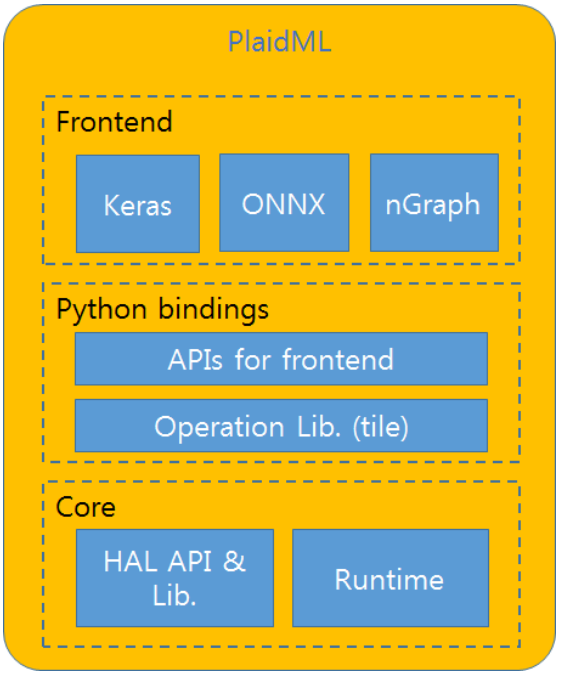


그림 4 PlaidML 프레임워크 구조도

을 기반으로 하여 설계되었다[10].

PlaidML은 Python 사용자 인터페이스를 제공하며, 프론트엔드에서 사용자 편의성을 위해 Keras와 같은 인터페이스를 제공한다. 하위 계층에서는 다양한 H/W를 지원하지만, 단일 프로세서에서만 동작 가능하다. PlaidML은 그림 4와 같이 프론트엔드, Python bindings, Core 계층으로 구성된다. 프론트엔드는 Keras와 같은 사용자 개발환경을 제공한다. Python bindings 계층에서는 프론트엔드에서 넘어온 API들과 신경망 관련 함수와 연산에 빈번히 사용되는 선형대수 연산 등을 H/W에서 빠르게 처리하기 위한 변환 및 최적화 과정을 거쳐 H/W 독립적인 중간코드가 생성된다. 이 코드는 마지막 Core 계층으로 전달되며 HAL(Hardware Abstraction Layer)에서 다시 한번 최적화 및 변환을 거쳐 최종 실행 코드가 생성된다.

ETRI가 개발하고 있는 개방형 병렬 컴퓨팅 프레

임워크는 타 딥러닝 프레임워크와의 상호 연동성과 사용자 편의성을 제공하기 위해 프론트엔드에서 NNEF 사용자 인터페이스를 제공하고, Core 계층에서는 H/W 플랫폼의 범용성과 이식성을 위해서 Khronos 그룹의 OpenCL에 기반하여 동작할 수 있도록 설계하였으며, 임베디드 환경 제약성에서도 고성능을 추구하기 위해 다중 프로세서를 지원하는 구조로 설계하여 개발하고 있다.

1. 프론트엔드

프론트엔드 계층은 사용자 프로그램에서 신경망 엔진을 실행하기 위해서 사용하는 API와 같은 사용자 접근 기능들을 포함한다. 대부분의 사용자 프로그램은 프론트엔드를 통해서 신경망 실행 엔진에 접근할 수 있다. 프론트엔드 계층은 다수의 기존 응용 프로그램과의 연결성 및 사용자 개발 편의성을 위해 Python 인터페이스를 제공하며, 이 계층 또한 대부분 Python으로 프로그램되어 있다.

Intel은 PlaidML의 확장을 위해 제3의 개발자가 프론트엔드를 개발하여 연결할 수 있도록 한다. 현재 개발되어 배포되고 있는 프론트엔드는 Keras, ONNX, nGraph가 있다. 이들은 사용자 편의성과 타 프레임워크와의 상호 연동을 위해 제공하는 것으로, 이들을 사용하지 않고도 직접 PlaidML API들을 사용하여 응용 프로그램을 개발할 수 있다.

개방형 병렬 컴퓨팅 프레임워크에서는 NNEF 포맷으로 신경망 표현 모델을 읽고 쓸 수 있는 인터페이스가 구현되어 있다[11]. NNEF는 ONNX에 비해 활용예가 부족한 단점이 있지만, Khronos 그룹이 주관하는 규격이므로 영상 처리, 방송 혹은 통신 서비스 관련 국제 표준과의 연계에서 장점을 발휘할 수 있다. 또한 ETRI를 비롯하여 Google, AMD, Intel 등의 글로벌 기업이 표준화에 참여하고 있으므로 NNEF의 활용도는 현재보다 증가가 확실시되고 있다.

2. Python bindings

Python bindings 계층은 프론트엔드에서 직접적으로 불리거나 다른 프론트엔드 개발에 사용되는 API와 실제 수행될 H/W 부분(CPU 혹은 GPU)을 연결하기 위한 인터페이스를 제공하는 부분과 응용프로그램에서 작성된 코드를 하위 계층에서 처리하기 편리하도록 하기 위한 코드 변환 및 관련 라이브러리 부분으로 구성된다. 이 계층은 사용자 프로그램과의 연결을 위해서 프론트엔드에서 사용되는 언어인 Python과 C/C++ 언어로 작성된다. 이와 함께 선형대수와 관련된 부분의 병렬 연산 및 가속화를 위해서 Tile이라는 중간 언어도 사용된다. 하드웨어 독립적인 방법으로 사용자 편의성으로 생성된 코드(예, Keras code)를 그래프 연산이 편리하도록 변환하여 계산에 최적화할 수 있는 구조로 변환하며, 각 Tile 함수들을 연결하여 복합 Tile 함수로 구성하여 연산한다.

Tile은 GPU kernel 생성을 위해 개발된 PlaidML에서 사용되는 중간 언어로 수학적 형식에 가깝게 설계되어 있다. 선형대수 및 신경망 구조에서 빈번히 사용되는 함수들은 내부적으로 Tile로 작성되어 있거나 변환된다. 이 언어의 주 기능은 다차원 텐서 구조를 평면화하고, 텐서들을 타겟에 넣을 수 있는 적합한 크기로 최적화된 블록을 할당한다. 병렬 연산을 하기 위한 스케줄링 및 레이아웃을 생성하고 이를 최종적으로 시맨틱 트리로 생성한다.

개발자가 Tile에 익숙한 경우 임베디드 코드 형식으로 프론트엔드 계층에서도 함수를 작성할 수 있다.

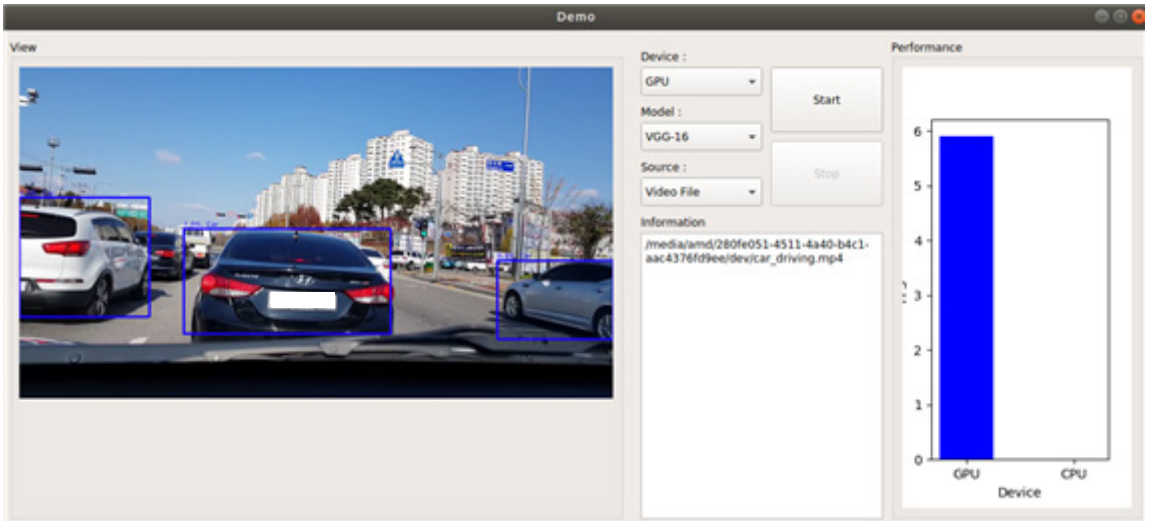


그림 5 개발한 개방형 추론엔진에서 차량 인식 화면 및 성능 예

3. Core

Core 계층은 신경망 연산을 위한 함수들이 최종적으로 수행되는 단으로 타겟 H/W 코드가 생성되는 부분과 런타임 시 해당 코드를 실행시킬 수 있는 부분으로 구성된다.

HAL까지 입력되는 값은 프론트엔드에서 획득된 타겟 디바이스에 대한 정보와 타겟 독립적인 동작 코드가 들어오기 때문에, 새로운 타겟을 위한 플랫폼 개발 시 타겟에 맞는 코드생성 부분만 개발하면 전체적으로 문제없이 동작할 수 있다.

이 계층은 주로 C/C++로 구현되고, 해당 타겟에 따라 OpenCL이나 LLVM 등에 의해 최종코드가 생성된다. 현재 지원하는 H/W로는 Intel x86 CPU, Nvidia CUDA, Apple Metal, Khronos Open-CL이 있다.

임베디드 시스템들은 동작환경에 대한 다양한 제약사항이 존재한다. IoT(Internet of Things)와 같이 가벼운 계산환경에서 자율주행자동차에서 영상인식과 같이 복잡한 계산환경까지 실제 적용에서는 넓은 스펙트럼이 존재한다. 자율주행자동차와 같이 복잡한 계산을 필요로 하는 분야에서는 멀티프로세서를 사용하는 것이 적합한 접근일 수 있다. 이를 위해 개방형 추론엔진은 다중 프로세싱을 지원하며 특정 H/W 제품에 국한되지 않기 위하여 OpenCL에 기반한 병렬 추론엔진을 개발하고 있다.

V. 결론

최근에 AI 및 딥러닝 분야에 대한 관심도가 날로 증가하고 있고, 관련 응용 프로그램 및 서비스가 지속적으로 나오고 있다. 본 고에서는 이들 중에서도 임베디드용 딥러닝 추론 프레임워크 중에서 빈번히 사용되고 있는 대표적인 4개의 추론 프레임워크에 대해서 각각이 가지는 특징과 그에 따른 장단점을 살펴보았다.

TensorFlow Lite의 경우 TensorFlow라는 프레임워크가 딥러닝에서 가지는 점유율을 장점으로 기존 개발 응용들을 큰 변화 없이 자동화된 툴들을 이용하여 변환함으로써 동작시킬 수 있다는 부분이 큰 장점으로 부각된다. TensorRT는 딥러닝 분야에서 Nvidia GPU가 가지는 우월적 점유율과 성능이라는

무기를 기반으로 임베디드 추론 프레임워크 시장에서 강점을 부각하고 있다. 이와 함께 ONNX를 지원함으로써 상호 연동성도 지원한다. Windows ML은 MS-Windows 계열의 OS에서 제공된다. PC 및 서버시장에서의 점유율을 바탕으로 자사가 주도하고 있는 신경망 포맷인 ONNX를 활용하여 신경망 모델을 표현하며, OS에 탑재되는 DirectML을 통해 신경망 실행의 가속을 지원하는 점들을 장점으로 내세우고 있다. 마지막으로 Core ML의 경우 모바일 시장에서의 Apple의 보급률과 사용 편의성들이 부각되는 제품적 특징을 보인다. 이와 함께 Core ML은 기존의 기계학습 모델들도 지원함으로써 폭넓은 가능성을 보여준다.

소개된 4개의 프레임워크는 각기 다른 특징과 장단점을 가지지만, 어느 한 프레임워크만으로 자율주행자동차와 같은 임베디드 환경에서 환경/자원 제약성을 만족하는 특성들과 성능 요구조건을 만족하는 특성들을 모두 지원할 수 있는 것은 없다. 본 고에서는 임베디드 환경에서의 요구조건을 만족시키기 위해 ETRI가 개발하고 있는 개방형 병렬 컴퓨팅 프레임워크를 소개하였다. 이 프레임워크는 자원 제약 및 고성능 요구조건을 위해 멀티 GPU기반으로 설계되었으며, 연산가속기 지원을 위해 이식성 및 범용성을 제공할 수 있는 OpenCL 상에서 동작하도록 되어 있다. 프레임워크 간의 학습과 추론을 분리하여 개발할 수 있도록, NNEF 인터페이스를 지원하여 프레임워크 간 상호 연동성을 제공한다.

향후 산업 및 생활 전반에 걸쳐 다양한 종류의 딥러닝기반 서비스가 나타날 것으로 기대하며, 이들을 위해 멀티프로세서뿐 아니라 이기종 프로세서 간 병렬 컴퓨팅 등 다양한 분야의 임베디드용 추론 프레임워크에 대한 성능향상과 최적화가 이루어질 것으로 예상한다.

약어 정리

AI	Artificial Intelligence
API	Application Program Interface
FPGA	Field-Programmable Gate Array
GPU	Graphics Processing Unit
ML	Machine Learning
NNEF	Neural Network Exchange Format
ONNX	Open Neural Network Exchange

참고문헌

[1] TensorFlow, "TensorFlow Lite guide," https://www.tensorflow.org/lite/guide?hl=ko

[2] Nvidia, "TensorRT," https://developer.nvidia.com/tensorrt

[3] Microsoft, "Windows Machine Learning," https://docs.microsoft.com/en-us/windows/ai/windows-ml

[4] Developer.apple, "Core ML," https://developer.apple.com/documentation/coreml

[5] Appcoda, "Introduction to the Accelerate Framework in Swift," https://www.appcoda.com/accelerate-framework/

[6] KHRONOS, "Neural Network Exchange Format (NNEF)," https://www.khronos.org/nnef

[7] ONNX, https://onnx.ai/

[8] ONNX.GitHub, "Open Neural Network Exchange- ONNX," https://github.com/onnx/onnx/blob/master/docs/IR.md

[9] KHRONOS, "NNEF and ONNX: Similarities and Differences," https://www.khronos.org/blog/nnef-and-onnx-similarities-and-differences

[10] Intel, AI, https://www.intel.ai/plaidml/

[11] K. H. Lee et al., "An Implementation of a Parser for Neural Network Exchange Format," *IEMEK Symp.Embedded Technol.*, Jeju island, Rep. of Korea, May 24-25, 2018, pp. 49-51.