RISC-V 가상플랫폼 기반 Yolov3-tiny 물체 탐지 딥러닝 모델 구현

김도영, 설희관, 임승호
한국외국어대학교 컴퓨터공학부
eh4536@naver.com, a01054149478@gmail.com, slim@hufs.ac.kr

Implementation of Yolov3-tiny Object Detection Deep Learning Model over RISC-V Virtual Platform

DoYoung Kim, Seol Hui-Gwan, Seung-Ho Lim
Division of Computer Engineering, Hankuk University of Foreign Studies

요 약

딥러닝 기술의 발전으로 객체 인색, 영상 분석에 관한 성능이 비약적으로 발전하였다. 하지만 고성능 GPU 를 사용하는 컴퓨팅 환경이 아닌 제한적인 엣지 디바이스 환경에서의 영상 처리 및 딥러닝 모델의 적용을 위해서는 엣지 디바이스에서 딥러닝 모델 실행 환경 과 이에 대한 분석이 필요하다. 본 논문에서는 RISC-V ISA 를 구현한 RISC-V 가상 플랫폼에 yolov3-tiny 모델 기반 객체 인식 시스템을 소프트웨어 레벨에서 포팅하여 구현하고, 샘플 이미지에 대한 네트워크 딥러닝 연산 및 객체 인식 알고리즘을 적용하여 그 결과를 도출하여 보았다. 본 적용을 바탕으로 RISC-V 기반 임베디드 엣지 디바이스 플랫폼에서 딥러닝 네트워크 연산과 객체 인식 알고리즘의 수행에 대한 분석과 딥러닝 연산 최적화를 위한 알고리즘 연구에 활용할 수 있다.

1. 서론

딥러닝 기술의 발전으로 객체 인식, 영상 분석에 관한 성능이 비약적으로 증가되었다. 이런 딥러닝 기술을 제대로 활용하기 위해서는 고성능의 GPU 가 활용되는 환경이 제공되야 한다. 근래에 들어 GPU 가 필수적으로 있어야 하는 환경이 아닌 모바일이나 엣지 디바이스에서의 물체탐지 및 영상처리에 관한 수요가 늘어나고 있다. 저전력 기기, IOT 기기와 같은 환경에서의 영상처리 분석 기술이 요구되는 것이다.

엣지 디바이스에서의 딥러닝 기술의 요구에 따라 모바일이나 임베디드 환경에 적합한 딥러닝 모델들이 개발되고 있다. 더불어 컴퓨팅 환경에 제약이 있는 임베디드 시스템을 위한 네트워크 압축, Pruning, Quantization 과 같은 딥러닝 네트워크 최적화 기법들에 대한 연구가 이루어지고 있다. 제한적인 엣지 디바이스 환경에서의 네트워크 최적화를 위해서 딥러닝 연산 및 알고리즘에 대한 디바이스 플랫폼 레벨에서의 적용과 분석이 필요할 수 있다.

본 연구는 RISC-V ISA 를 기반으로 메모리, 저장장치, 인터럽트, 시스템 버스 등으로 구축된 RISC-V 가상 플랫폼에서 물체탐지 및 객체 인식 모델로 널리 사용되는 Yolo 모델 중에서 YOLOv3-tiny 모델을 포팅하여 구현하여 보고, RISC-V 가상환경에서 샘플 이미지에 대한 객체 인식을 위한 네트워크 연산 및 알고리즘을 적용하여 보았다. 이를 통해서 임베디드 프로세서 기반의 엣지 디바이스 시스템에서 딥러닝 기반 객체 인식 알고리즘의 수행에 따른 프로세서, 메모리, 저장장치 및 시스템 버스의 리소스 사용량을 분석할 수 있는 방법을 구축하였다. 본 적용을 바탕으로 RISC-V 기반 임베디드 엣지 디바이스 플랫폼에서 딥러닝 네트워크 연산과 객체 인식 알고리즘의 수행에 대한 분석과 딥러닝 연산 최적화를 위한 알고리즘 연구에 활용할 수 있다.

2. 본론

2.1 RISC-V 가상 환경 및 요구사항

RISC-V[1]는 최근 개발된 임베디드용 프로세서로써 저전력 및 확장성 있는 구조로 임베디드 디바이스에서의 활용도를 높여가고 있다. 이러한 RISC-V 프로세서의 활용과 분석을 용이하게 하기 위해서 RISC-V 프로세서 기반의 가상 플랫폼을 구축하여 시뮬레이션 할 수 있는 환경이

제공되었다[2]. RISC-V 가상 플랫폼은 SystemC[3]로 구현되어 RISC-V ISA 프로세서가 구현되었고 TLM 2.0 Bus 가 적용되어, 메모리, 저장장치, 인터럽트 컨트롤러등을 구성하여 가상 환경 플랫폼을 구성할 수 있게 제공된다.

RISCV-VP 는 가상을 환경을 통해서 RISC-V 개발환경을 제공하고 이는 개발환경 컴파일러와의 차이가 존재한다[4]. 그렇기 때문에 RISCV-VP 개발환경에서 yolo3[5]사용을 위한 요구사항이 필요하고 요구사항은 다음과 같다.

첫째, yolov3 를 기반으로 한 yolov3-tiny 모델을 정수형 변수만을 사용하지 않고 float 변수를 사용하는 방식으로 설계되어 있다. 이를 위해서 riscv-vp 상에 soft float 을 지원하는지 확인해야 한다. RISCV Virtual Prototype github 를 통해서 build 하는 과정에서 vp/dependencies 디렉토리 내부의 ./build_softfloat.sh 파일을 빌드 하는 것을 확인할 수 있다. 빌드 이후 그림 1 과 같이 softfloat-dist 디렉토리가 생성되며 디렉토리 내부를 살펴보면 그림 2 와 같이 소스 파일이 생성된다. 해당 소스파일들이 생성되어야 RISCV-VP 에서 softfloat 를 사용할 수 있다.

둘째, DLA 가속기를 활용한 CNN 시뮬레이터 모델의 RISC-V 컴파일러는 riscv32-unknown-elf-gcc 를 사용하며 컴파일 옵션은 -march=rv32ima -mabi=ilp32 을 사용한다. -march 인수는 프로그램이 실행될 구현 세트를 결정한다. rv32i 는 32 개의 32 비트 범용 정수 레지스터가 있는 로드 저장 ISA 를 사용하는 것이고 m 은 정수 곱셈과 나눗셈, a 는 원자 명령어를 사용한다는 것이다. m 옵션을 통해서 float 루틴에 곱셈과 나눗셈 명령어가 포함된다. -mabi=ilp32 를 통해서 multilib 를 적용시켜 컴파일 옵션을 사용한다. multilib 옵션은 그림 3 과 같이 다양하게 존재하며 riscv-gnu-toolchain 디렉토리 내부의 t-elf-multilib 파일 내부의 옵션을 선택하거나 추가해서 사용 가능하다.

```
dodo@ubuntu:~/RISCV/riscv-vp/vp/dependencies$ ls
build_debug_systemc_233.sh   spike-softfloat-20190310.tar.gz
build_softfloat.sh           systemc-2.3.3
build_systemc_233.sh         systemc-2.3.3.tar.gz
debug-patches-systemc-2.3.3  systemc-dist
softfloat-dist
```

(그림 1) vp/dependencies 디렉토리 내부

```
dodo@ubuntu:~/RISCV/riscv-vp/vp/dependencies/softfloat-dist/include/softfloat$ ls
internals.h  primitives.h      softfloat.h    softfloat_types.h
platform.h   primitiveTypes.h  softfloat.hpp  specialize.h
```

(그림 2) softfloat-dist 의 include 디렉토리 내부

```
19 MULTILIB_REQUIRED = march=rv32i/mabi=ilp32 \
20 march=rv32im/mabi=ilp32 \
21 march=rv32iac/mabi=ilp32 \
22 march=rv32imac/mabi=ilp32 \
23 march=rv32imafc/mabi=ilp32f \
24 march=rv64imac/mabi=lp64 \
25 march=rv64imafdc/mabi=lp64d
```

(그림 3) riscv-gnu-toolchain 옵션

2.2 RISC-V VP 기반 YOLOV3-tiny 구현 및 결과

YOLOv3-tiny 는 총 24 개의 레이어로 구성된 네트워크 구조를 가진다. Convolution, Maxpooling, Route, Upsample, Yolo 5 개의 레이어를 사용하는 반복구조를 가진 형태로 float 화 된 이미지 데이터를 연산한다.

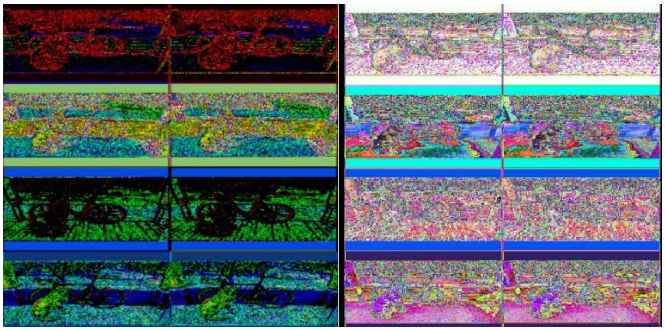
yolov3-tiny 는 네트워크 구조체를 선언하고 연산에 필요한 변수들을 cfg 파일에 read 하여 적용하거나 동적 할당한다. 이후 이미지를 load 하여 float 데이터화 시킨 이후 load 된 network 구조체에 이미지 파일을 적용시켜 forwarding 을 진행한다. 여기서 darknet 과 동일한 방식으로 forwarding 진행 시 network 구조체의 input 데이터 즉 연산에 사용하는 이미지 데이터가 소실되는 문제가 생긴다. forwarding 호출 소스코드를 darknet.c 소스파일에서 호출하고 전역변수를 통해서 각 호출 이전과 이후에 input, output 데이터를 저장하는 역할을 수행한다.

5 종류의 레이어에 대한 연산 결과를 확인하여 각 레이어가 올바른 연산을 수행하는지 확인할 수 있다. 네트워크 구조에서 feature map 을 추출하는 Convolution 레이어와 Maxpooling 레이어는 기존 darknet 과 흡사한 결과를 보이는 것이 중요하다. 올바른 feature map 을 얻기 위해서 기존에 사용하는 weights 파일을 올바르게 read 하여 사용해야 한다. RISCV-VP darknet 컴퍼일러 환경과 달리 size_t 변수의 크기가 4byte 로 되어있기 때문에 해당 부분을 수정하여 사용하였다. 이를 통해서 darknet 에서와 동일한 weights 파일을 적용하여 사용 가능 하다.

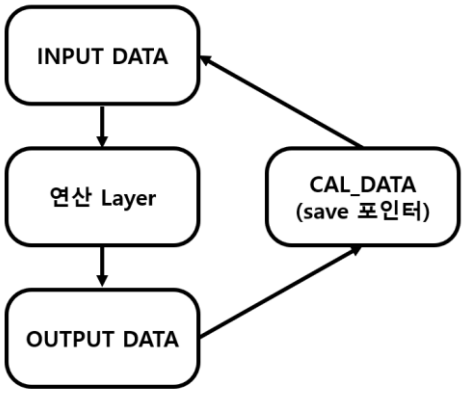
2 번의 convolution 연산을 수행한 결과 각각 darknet 과 비교한 결과인 그림 4 를 통해서 확인할 수 있다. Convolution 수행 결과를 통해서 feature map 추출이 올바르게 이루어진다는 것을 확인할 수 있으며 각 레이어에서 연산에 사용하는 input 데이터의 소실 또한 없는 것이 확인 가능하다. 24 개의 레이어를 통한 연산을 진행한 이후 get_network_boxes 함수를 통해서 예측한 바운딩박스의 오브젝트를 판별한다. 오브젝트가 있을 것으로 예상되는 박스를 draw_detection 함수를 통해서 이미지 위에 박스를 그려준다.

그림 5 는 input 데이터를 전역변수를 통해 전달하는

방식을 보여주며 그림 6 은 RISCV-VP 를 통한 DOG 이미지의 물체탐지를 보여주는 결과 이미지이다. 결과 화면을 통해서 총 5 개의 예측이 존재하는 이미지에서 트럭의 이미지만 올바르게 물체탐지를 한 결과가 존재한다. 강아지는 고양이라고 인지한 것으로 보아 물체 탐지가 가능하지만 Convolution 연산을 통한 feature map 추출 시 특정 부분 feature 를 올바르게 추출하지 못하는 것으로 확인되며 weights 파일의 변화 또는 연산 과정의 파라미터 정제가 필요할 것으로 확인된다.



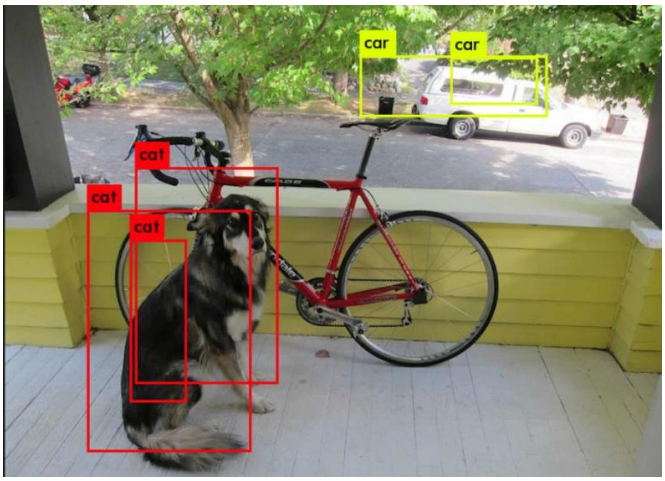
(그림 4) RISCV-VP/Darknet 에서의 convolution 연산 결과



(그림 5) cal_data 를 활용한 input 데이터 전달 아키텍쳐

3. 결론

본 논문에서는 RISC-V 가상 환경에서 Yolo3-tiny 와 같은 딥러닝 기반 객체 인식 알고리즘을 구현 적용하여 보았다. 해당 결과를 통해서 RISC-V 와 같은 임베디드 프로세서를 통한 물체탐지 딥러닝 모델의 적용과 분석 환경을 구성할 수 있었다. 딥러닝 모델을 GPU 없이 임베디드 프로세서로 실행할 경우 리소스에 제한적인 임베디드 프로세서의 과부화 문제점과 속도 측면에서 부족한 부분이 존재한다. 그렇기 때문에 하드웨어 측면으로 가속기를 부착하거나 딥러닝 기법의 경량화를 통한 네트워크의 전반적 경량화 적용이 필요할 것이며, 경량 딥러닝 네트워크 모델을 실제 RISC-V 에 적용할 시에 임베디드 시스템에서의 효율적인 물체탐지가 가능할 것으로 보인다. 향후 RISC-V 가상 플랫폼 기반의 객체 인식을 위한 네트워크 최적화에 대한 연구를 진행할 것이다.



(그림 6) RISC-V VP 에서 Object detection 을 수행한 결과

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean Government (MSIT) (NRF-2021R1F1A1048026).

참고문헌

[1] A. Waterman, Y.S. Lee, and R. Avizienis, D. Patterson, and K. Asanovi´c “The RISC-V Instruction Set Manual”, Volume II: Privileged Architecture, July, 2016, ttps://people.eecs.berkeley.edu/~krste/papers/riscv-privileged-v1.9.pdf

[2] Github, RISCV-V Virtual Prototype https://github.com/agra-uni-bremen/riscv-vp

[3] D.C. Black, J. Donovan, B. Bunton, and A. Keist, “SystemC : From The Ground up”, Eklectic Ally, Inc.

[4] Github, RISC-V GNU Compiler Toolchain https://github.com/riscv-collab/riscv-gnu-toolchain

[5] Joseph Redmon, Ali Farhadi, "YOLOv3: An Incremental Improvement", Technical Report, 2018.

[6] “RISC-V command Options”, https://gcc.gnu.org/onlinedocs/gcc/RISC-V-Options.html