Pusan National University

Computer Science and Engineering

Technical Report 2020-34

FPGA 를 이용한 High speed processing 지능형 CCTV

저자 1: 박종욱

저자 2: 박성찬

저자 3: 김문석

지도교수: 김호원 교수님

목 차

1. 서론 1

1.1. 연구 배경 1

1.1.1 국내 CCTV 상황 1

1.1.2 A.I 가속기 배경 2

1.2. 기존 문제점 3

1.3. 연구 목표 5

2. 연구 배경 6

2.1. 연구 개발 환경 6

2.2. CNN(Convolutional Neural Network) 6

2.3. CUDA(Compute Unified Device Architecture) 7

2.4. YOLOv3 7

2.5. FPGA(ZCU104) 7

2.6. Vitis A.I 8

3. 연구 내용 8

3.1. 하드웨어적 구현 8

3.2. Convolution Layer 9

3.3. Activation Function 12

3.4. Max Pooling 12

3.5. Fixed Point 14

3.6. IP Integrator 15

3.7. FPGA 보드 실행 모습 18

3.8. YOLOv3를 이용한 학습 19

3.9. FPGA 25
4. 연구 결과 분석 및 평가 27
4.1. 설계 변경 내역 27
4.2. 연구 결과 분석 27
5. 결론 및 향후 연구 방향 30
6. 개발 일정 및 역할 분담 30
6.1. 개발 일정 30
6.2. 역할 분담 31
7. 참고 문헌 33
8. 구현 코드 34

1. 서론

1.1. 연구 배경

(1)국내 CCTV 상황

CCTV는 우범지역, 불법주정차 만연지역 등 범죄가 일어날 만한 장소나 주민들이 피해가 있을 장소에 설치를 한다. 전통적인 CCTV 방식은 불법주정차용 CCTV, 과속 감지용 CCTV등 모두 개별적인 기능으로 존재했었다. 그렇게 되니 CCTV를 감시할 사람의 수도 부족하고 효율적이지 못해 다목적 CCTV를 많이 설치하고 성능개선을 추진하고 있ㄴㄴㄴㄴㄴㄴㄴㄴㄴㄴㄴㄴㄴㄴㄴㄴㄴㄴㄴㄴㄴㄴㄴㄴㄴㄴㄴㄴㄴㄴㄴㄴ다.

다목적 CCTV 신규설치

- 민원이 접수된 개소 등을 토대로 CCTV 설치 우선순위를 선정
- 편성된 예산범위 내에서 순차적으로 설치추진

다목적 CCTV 성능개선

- 노후CCTV 화질 개선: 41만 화소 → 200만 화소 CCTV 교체
- 메인카메라만 설치된 지역에 보조카메라 추가설치(개소별 방범강화)

추진일정

구 분	2018년				2019년				2020년				2021년				2022년			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
다목적 CCTV 신규 설치																				
다목적 CCTV 성능 개선																				

소요예산액

(단위: 백만원)

세부사업	재원	계	2018년	2019년	2020년	2021년	2022.6.
계	계	11,332	2,579	2,908	2,245	1,800	1,800
	구비	2,922	269	708	945	500	500
	시비	200	-	200	-		
	국비	8,210	2,310	2,000	1,300	1,300	1,300
다목적 CCTV 신규 설치	-	10,032	2,079	2,708	2,045	1,600	1,600
다목적 CCTV 성능 개선	-	1,300	500	200	200	200	200

그림 1. 다목적 CCTV 추진계획

그림(1)은 2018년 서울시 다목적 CCTV의 추진계획이다.

하지만 문제는 이러한 다목적 CCTV도 사람이 눈으로 확인을 해야 하는 것이다. 그렇기 때문에 현재 서울시에는 지능형 CCTV에 많은 관심이 있고 추진을 하고 있는 중이다. 1년 만에 다목적에서 지능형으로 관심을 둔 점은 한계를 느끼고 A.I를 통해 더욱 효과적인 방법을 찾은 것이다. 아래 그림(2)를 보면 서울시에서 2019년에 지능형 CCTV에 예산을 얼마나 썼는지 확인할 수가 있다.

사업 예산 및 집행 현황

전체 예산	6,962,500,000원
지출액	6,950,500,000원
집행잔액	12,000,000원
집행율	99.83%

예산 수입 내역

회계연도			2019	분야	일반공공행정
회계구분			일반회계	사업구분	보조
예산현액	계	소계			6,962,500,000원
		국비			0원
		시비			6,962,500,000원
	편성액				6,962,500,000원
	이월액				0원
	예산변경				0원
	수입대체경비				0원
지출액					6,950,500,000원
집행잔액					12,000,000원

그림 2. 다목적 CCTV 예산안

서울시만이 아니라 다른 지역에서도 원래의 CCTV한계는 똑같이 느끼고 있다. 이렇게 지능형 CCTV를 통해 사람의 눈을 통하지 않고 즉각 처리할 수 있는 CCTV를 개발하고자 한다.

(2) A.I 가속기 배경

A.I는 CPU와 GPU로 계산 및 동작을 진행하는데, GPU가 가지는 크기, 전력소비, 비용 등으로 인해서 IoT기기에 사용되기에는 불가능하다. 그래서 A.I 가속기라는 것이 나오기 시작했고 이는 A.I 모델의 추론 및 학습을 하는데 있어서 고속화, 저전력화, 크기 등에서 이점을 가지는 것이다. A.I 가속기는 주로 ASIC으로 개발하거나, FPGA를 이용하여 개발을 한다.

A.I가 각광받고 있는 지금 시대에 해외의 많은 기업들(Intel, NVIDIA, Google, Xilinx, Amazon…)과 국내 기업들(Naver, SK…)도 A.I 가속기시장에 투자를 함으로써 매우 큰 시장이 될 것으로 관찰되지만 우리나라에서의 연구는 아직 뚜렷한 성과를 내고 있지 않고 시장이 넓지가 않다. 그렇기 때문에 우리는 FPGA를 이용한 연구를 통해 A.I 가속기를 개발한다.

FPGA를 사용하는 이유는 A.I(Artificial Intelligence)를 컴퓨터로 사용하는 것보다 FPGA를 이용해 저전력화 및 크기와 속도에 이점을 가지고자 한다. Arduino나 Raspberry Pi 같은 임베디드 보드에는 ASIC(주문형 반도체)가 올라가기 때문에 소프트웨어적으로만

사용을 할 수 있지만, FPGA를 이용하면 하드웨어적, 소프트웨어적으로 둘 다 사용이 가능하기 때문에 FPGA를 사용한다. 아래 그림3 은 이번 개발에 사용하는 ZCU104 보드이다.

그림 3. ZCU-104

ZCU104 보드는 고속 DDR4 메모리 인터페이스, FMC 확장 포트, 멀티 기가 비트급 직렬 송수신기, 다양한 주변 장치 인터페이스, 맞춤 설계용 FPGA 패브릭을 갖춘 유연한 시제품 제작 플랫폼이다. OpenCV 라이브러리, 머신 러닝 프레임워크, 라이브 센서 지원을 갖춘 바로 활용 가능한 SDSoC 개발 환경 소프트웨어 플로우를 제공한다.

1.2. 기존 문제점

길거리를 지나가다 보면 CCTV가 있는 것을 확인할 수 있다. CCTV로 인해 범죄율이 낮아 졌다고는 하나 실제로 CCTV는 카메라를 통해 얻은 영상을 전송망을 통하여 전달받아 사람이 모니터를 이용하여 보안 상황을 실시간으로 직접 감시되어 진다. 또한, DVR등 저장 장치에 저장한 뒤에 사후에 저장된 영상을 검색해 대응하는 것 등의 수동적인 시스템이다. 즉 사람이 사람을 통해 대응이 가능하고 분석이 가능하다. 그렇기 때문에 좀 더 빠르고 안전하게 대비를 할 수 있는 지능형 CCTV를 개발하는 이유이다. 지능형 CCTV가 전통적인 CCTV보다 안전한 이유가 있다. 현재 지능형 CCTV는 아래 그림(3), 그림(4)와 같이 기능 개념은 사진으로 나타낸다.

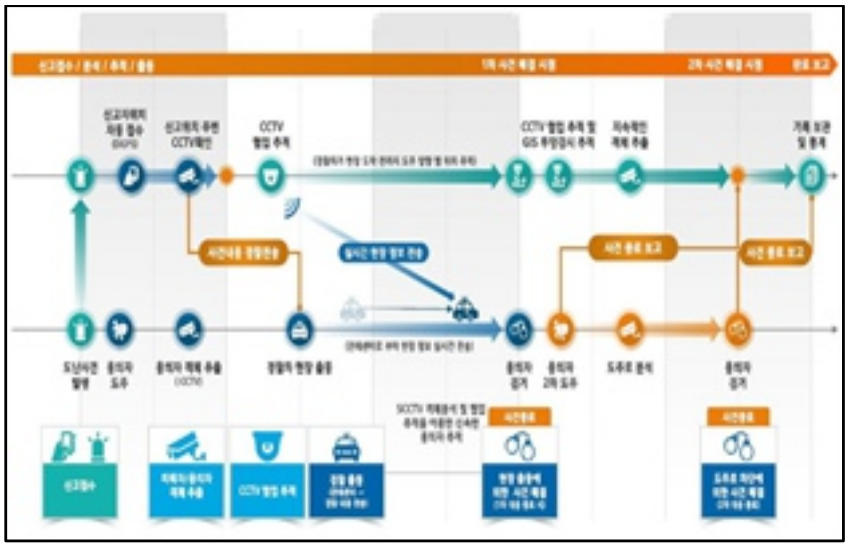


그림 4. 범죄 지능형 CCTV시나리오

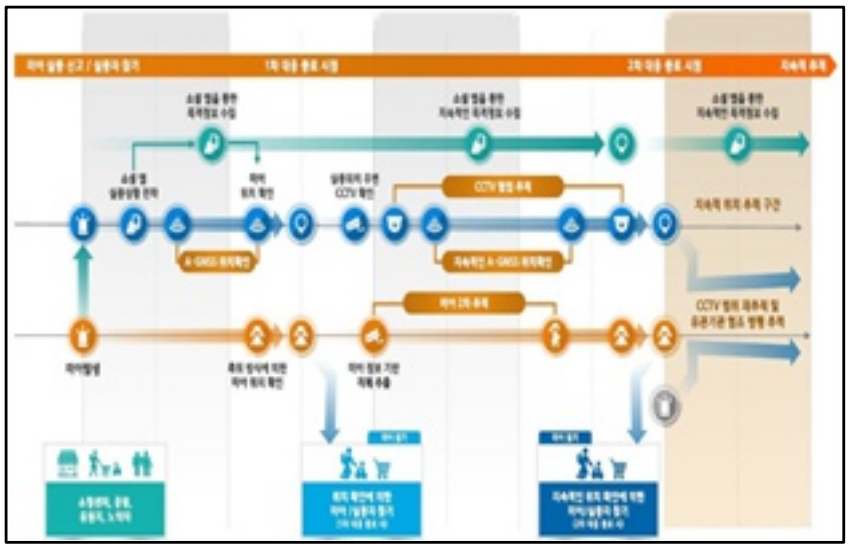


그림 5. 실종자 찾기 지능형 CCTV시나리오

이렇게 CCTV가 직접 경찰서, 학교 보안관에게 직접 상황 전파를 하면 유관기관 은 즉시 현장 대응을 할 수 있기에 기존의 것보다 더욱 빠른 조치를 취할 수 있다. 현재 개발되고 있는 지능형 CCTV는 A.I를 통해 영상을 데이터센터로 보내 분석을 하는 것처럼 데이터들을 중앙 데이터센터에 보내 영상을 분석한다. 즉 A.I는 컴퓨터를 거쳐서 영상을 분석해야 한다는 것이다. 우리는 이러한 단계에서도 발생할 사고 위험률을 조금 더 줄이기 위해 FPGA를 통한 지능형 CCTV를 개발한다.

1.3. 연구 목표

CCTV는 범죄예방, 사고예방 등의 이유로 점차 많은 수가 설치되고 있다. 최근에는 기존의 CCTV가 사람이 직접 모니터링을 해야 하는 문제 등으로 지능형 CCTV의 개발 및 사용이 활발히 이뤄지고 있다. 본 팀은 FPGA를 이용한 High speed processing 지능형 CCTV를 만들고자 한다. 소프트웨어적으로만 사용 가능한 다른 반도체 소자들과는 달리 하드웨어와 소프트웨어 둘 다 사용 가능한 FPGA를 사용함으로써 AI추론 파트를 병렬화 및 최적화 수행으로 빠르게 하여 빠른 감지를 할 수 있게 한다. 또한 FPGA에 OS를 사용함으로써 여러 대의 CCTV를 1개의 FPGA로 처리할 수 있어서 좀 더 효율적인 지능형 CCTV를 만들 수 있도록 한다.

또한 아래의 사진을 보면 미국의 시장조사기관 트랙티카가 예상한 AI Accelerator시장의 예측 모습이다. 매년 꾸준히 증가하다가 2025년쯤에는 663억 달러에 달하는 거대 시장으로 발전할 것으로 예측하지만 앞서 말했듯이 우리 나라에서의 연구는 아직 뚜렷한 성과를 내고 있지 않으며, 위에서 말한 기업들 대부분이 ASIC을 이용하여 수정이 불가능한 Accelerator를 연구하고 있다. ASIC은 초기 개발 과정이 매우 힘들고, 비용 또한 많이 들게 된다. 따라서 우리가 할 수 있는 연구는 ASIC과 비슷한 수정이 가능한 FPGA를 이용하여 AI Accelerator 시장에 도전할 수 있게 된다. 우리는 이를 이용한 서비스에도 직접 적용함으로써 얻는 이익을 보여주고자 한다.

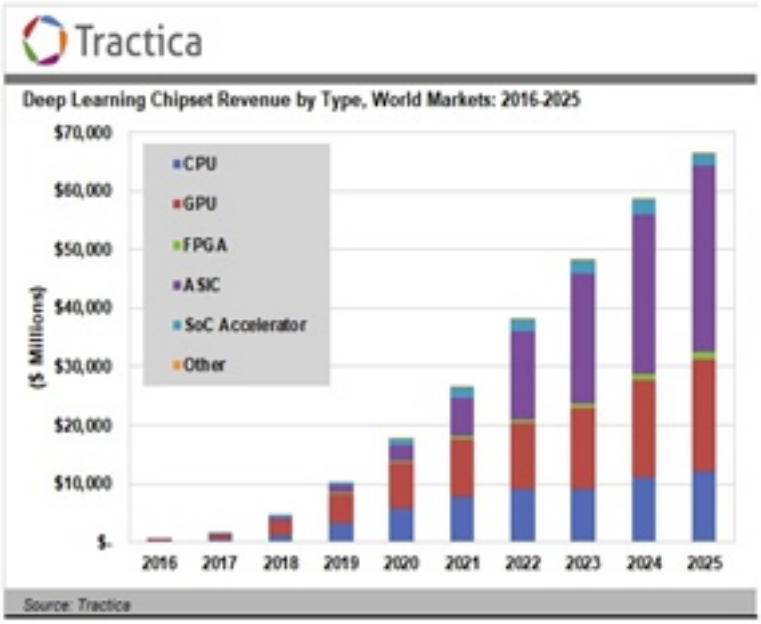


그림 6. AI chip 시장 예측도

2. 연구 배경 지식

2.1. 연구 개발 환경

그림 7. 개발환경

개발환경은 OS MAC과 Centos 7에서 사용하였다.

2.2. CNN (Convolutional Neural Network)

사진 데이터로 FC(Fully Connected)신경망을 학습시켜야 할 경우에, 3차원 사진 데이터를 1차원으로 평면화 시켜야 한다. 사진 데이터를 평면화 시키는 과정에서 공간 정보가 손실될 수밖에 없는데 결과적으로 이미지 공간 정보 유실로 인한 정보 부족으로 인공 신경망이 특징을 추출 및 학습이 비효율적이고 정확도를 높이는데 한계가 있다. 이미지 공간의 정보를 유지한 상태로 학습이 가능한 모델이 바로 CNN이다.

CNN은 기존 FNC(Fully Connected Neural Network)와 비교했을 때 아래와 같은 차별성을 갖는다.

1. 각 Layer 의 입출력 데이터의 형상 유지
2. 이미지의 공간 정보를 유지하면서 인접 이미지와의 특징을 효과적으로 인식
3. 복수의 필터로 이미지의 특징 추출 및 학습
4. 추출한 이미지의 특징을 모으고 강화하는 Pooling Layer
5. 필터를 공유 파라미터로 사용하기 때문에, 일반 인공 신경망과 비교하여 학습 파라미터가 매우 적음

2.3. CUDA(Compute Unified Device Architecture)

CUDA 는 NVIDIA 에서 개발한 GPU 개발 툴로써, 그래픽 처리 장치(GPU)에서 수행하는 (병렬 처리)알고리즘을 C 프로그래밍 언어를 비롯한 산업 표준 언어를 사용하여 작성할 수 있도록 하는 GPGPU(General-Purpose computing on Graphics Processing Units, GPU)기술이다. 즉, 많은 양의 연산을 동시에 처리하는 것이 가능하게 하기 때문에 딥러닝, 채굴과 같은 수학적 계산에 많이 쓰인다.

2.4. YOLOv3

YOLO는 You Only Look Once의 줄인 말로써 이 Object detector는 deep CNN을 통해 학습한 feature들로 object들을 detect하는데 사용된다. 이전 탐지 시스템은 classifier나 localizer를 사용해 탐지를 수행해왔지만, YOLO는 하나의 신경망을 전체 이미지에 적용한다. 이 신경망은 이미지를 영역으로 분할하고 각 영역의 Bounding Box와 확률을 예측한다. 이런 Bounding Box는 예측된 확률에 의해 가중치가 적용된다.

2.5. FPGA (ZCU104)

FPGA를 사용하는 이유는 A.I(Artificial Intelligence)를 컴퓨터로 사용하는 것보다 FPGA를 이용해 좀 더 빠르게 감지를 하고자 한다. Arduino나 Raspberry Pi 같은 임베디드 보드에는 ASIC(주문형 반도체)가 올라가기 때문에 소프트웨어적으로만 사용을 할 수 있지만, FPGA를 이용하면 하드웨어적, 소프트웨어적으로 둘 다 사용이 가능하기 때문에 FPGA를 사용한다. 아래 그림은 이번 개발에 사용하는 ZCU104 보드이다.

그림 8. ZCU104

ZCU104 보드는 고속 DDR4 메모리 인터페이스, FMC 확장 포트, 멀티 기가 비트급 직렬 송수신기, 다양한 주변 장치 인터페이스, 맞춤 설계용 FPGA 패브릭을 갖춘 유연한 시제품 제작 플랫폼이다. OpenCV 라이브러리, 머신 러닝 프레임워크, 라이브 센서 지

원을 갖춘 바로 활용 가능한 SDSoC 개발 환경 소프트웨어 플로우를 제공한다.

2.6. Vitis A.I

Xilinx의 'Vitis AI'는 통합 소프트웨어 플랫폼인 Vitis와 결합하여 소프트웨어 개발자들이 딥러닝 가속을 구현할 수 있도록 소프트웨어 코드 형태로 제공된다. 또한, 우리가 사용하는 Xilinx의 디바이스 상에서 실행되는 트레이닝 AI 모델을 최적화 및 압축, 컴파일을 할 수 있는 툴을 제공한다. Vitis플랫폼은 오픈소스 표준 개발 시스템 및 구현 환경과 완벽하게 연결되는 스택 기반의 아키텍처로 구현되어 있으며, 풍부한 표준 라이브러리를 갖추고 있다.

3. 연구 내용

3.1. 하드웨어적 구현

추론 단계는 학습 단계의 다음인 머신 러닝의 두 번째 단계이다. 학습된 파라미터에 기반해서 새로운 데이터에 대해서 예측이나 판단을 한다. 추론 단계에는 프로세싱 성능을 학습 단계보다 덜 필요로 하고 전력도 덜 소모한다. 학습이나 추론에 요구되는 높은 수준의 프로세싱 성능을 해결할 수 있는 것이 AI 가속기이다.

하드웨어 및 소프트웨어 기반 AI 가속기를 사용함으로써 머신 러닝을 빠르게 실행할 수 있다. 하드웨어 가속화는 학습 또는 추론에 둘 다에 사용할 수 있다. 예를 들어 전력 요구량을 줄이거나, 프로세싱 성능을 높일 수 있다. 이러한 AI 가속화기에는 다양한 칩이나 프로세싱 유닛들이 나와 있다. 병렬화 Processing 이 가능한 것으로 GPU, FPGA 가 있다.

CPU와 GPU는 상당한 프로세싱 성능을 활용할 수 있고, 학습과 추론을 가속화하기에 효과적이지만 메모리와 프로세싱 사이에 데이터를 이동시키기 위해서 많은 시간과 에너지를 소모한다. 또한, CPU는 성능을 중시하는 프로세서라서 소비 전력을 줄이는데 한계가 존재하고, GPU는 병렬 연산을 위해 수천 개의 코어를 사용하기 때문에 다른 프로세서들보다 전력 소모량이 많은 편이다.

Criteria	CPUs	GPUs	FPGAs
Processing Peak Power	Moderate	High	Very High
Power Consumption	High	Very High	Very Low
Flexibility	Highest	Medium	Very High
Training	Poor at training	The only production-ready training hardware	Not efficient
Inference	Poor at inference but sometimes free	Average for inference	Best for inference

그림 9. CPU vs GPU vs FPGA

또한, CPU/GPU 와 비교하여 FPGA 에서는 AI inference에서 Low Latency의 이득을 볼 수 있다. CPU/GPU는 High-batch size를 사용하고, processing 되기 이전에 모든 input 값이 준비가 될 때까지 기다려야 하는 단점이 존재한다. 따라서 Input을 처리하는데 있어서 Latency 가 길게 된다. FPGA 에서는. Low-batch size를 사용하여 processing 과정을 빠르게 진행함으로써 Latency를 줄일 수 있게 한다.

그림 10. CPU/GPU와 FPGA/ACAP비교 (ACAP: Adaptive Compute Acceleration Platform - 동적으로 작업 요구사항에 맞춰진 멀티코어 컴퓨팅 플랫폼을 의미함)

3.2. Convolution Layer

CONV(Convolution Layer)은 입력 데이터로부터 특징을 추출하는 역할을 한다. CONV에서 로컬 연결성(Local Connectivity)이 있다. 이미지와 같은 고차원 입력을 다룰 때에는 현재 레이어의 한 뉴런을 이전 볼륨의 모든 뉴런들과 연결하는 것은 비 실용적이다. 때문에 CONV에서는 각 뉴런을 입력 볼륨의 로컬한 영역에만 연결한다.

하드웨어상에서 Convolution은 단순 연산의 반복이므로 파이프라인 구조를 이용해서 효율적인 연산이 가능해진다.

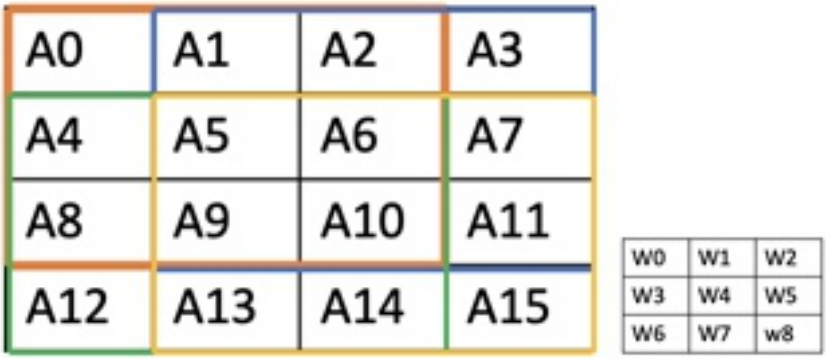


그림 11. 4*4이미지 크기에 3*3커널을 이용한 연산

위와 같이 4*4 이미지 크기에 3*3 커널을 이용한 Convolution 연산 시에 커널 1개를 이용하여 값을 1개씩 구하는 구조가 소프트웨어이다. 하드웨어에서는 파이프 라인 구조를 이용하여 값을 구할 때까지 기다리지 않고 매 클럭 주기마다 입력이 들어와 전체 작업의 일부를 진행하는 형식이다. 이는 큰 계산 Processing 과정을 작은 단계로 나누어 전체 회로가 동작을 하게 할 수 있다.

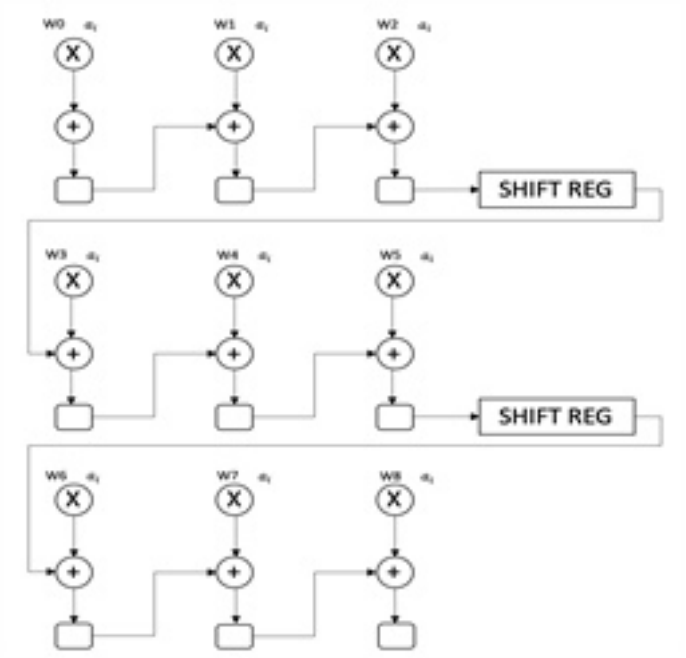


그림 12. Multiplication & Adder: 3*3커널

커널의 크기만큼 Multiplication과 Adder가 필요하게 될 것이다. 3*3 커널의 크기라면 총 9개씩의 Multiplication과 Adder가 필요한 예시이다. 또한 Shift Register를 통하여 불필요한 연산은 넘어가서 다음 연산에 쌓이는 구조가 된다. 이를 MAC(Multiply and Accumulate)라고 하겠다. 일반화를 한다면 N*N 이미지 크기와 K*K 커널 크기가 주어지면 K<N 이 만족해야 한다. Zero padding이 없다는 전제하에 출력의 크기는 (N-K+1)/s *

(N-K+1)/s 가 된다. 이와 같은 구조를 통해서 입력의 크기는 1번만 메모리에서 가져오기 때문에, 메모리 접근 시간과 Storage 요구사항이 크게 줄어 든다. 또한 입력을 끊거나 일시적으로 값을 다른 곳에 저장하지 않아도 되므로 효율적이다. 또한 일반적으로 구현을 했기 때문에 Stride도 다르게 변경하여도 된다.

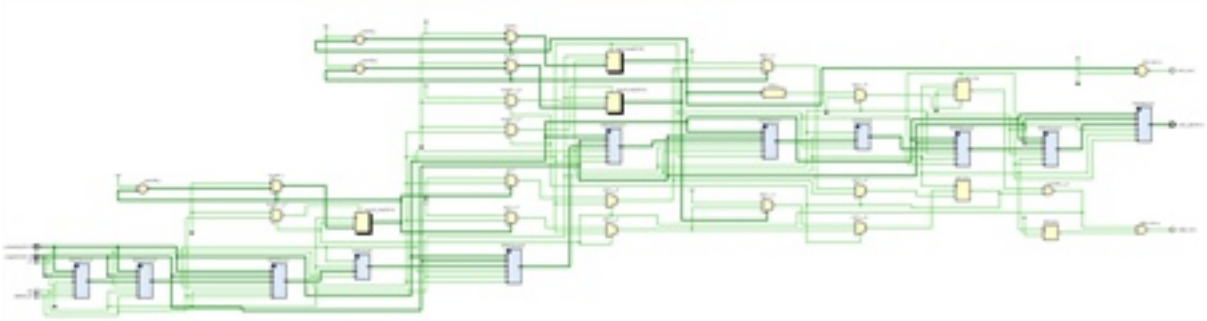


그림 13. Schematic 결과

Vivado 툴을 이용한 Schematic 결과이다. 총 9개의 MAC 과 2개의 Shift 모듈로 구성이 되어 있다. 그 외에는 간단한 Register, MUX, 와 Count를 위한 덧셈 연산으로 구성이 되어 있다.

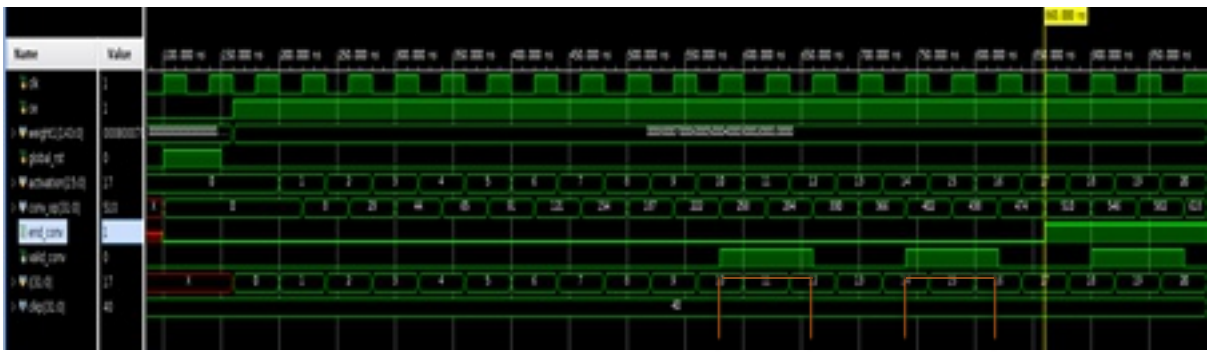


그림 14. Testbench작성 simulation결과

Testbench를 작성하여 Simulation을 한 결과이다.

kernel

[[0 1 2]
[3 4 5]
[6 7 8]]

activation map

[[0 1 2 3]
[4 5 6 7]

[8 9 10 11]
[12 13 14 15]]

convolution output

[[258 294]
[402 438]]

위의 예시를 사용하여 Testbench를 작성하였고, Valid_conv가 1이 된 순간에 Clock이 Rising 일 때, 값을 확인하면 258,294가 연이어 구해지며, Shift 후에 다음 값인 402와 438 값을 얻은 결과를 볼 수 있다. 그리고 end_conv가 1이 된다면 연산은 끝이 난 것으로 확인이 가능하다.

3.3. Activation Function

Activation Function은 쉽게 말해 비선형성을 주기 위한 함수이다. 하드웨어 상에서는 가장 많이 사용이 되는 Relu만이 효율적이다. 다른 Activation Function들은 소프트웨어에서 계산하는 것이 훨씬 효율적이다.

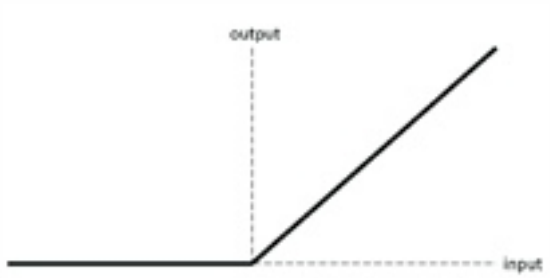


그림 15. Activation Function

3.4. Max Pooling

CONV들 중간중간에 주기적으로 Pooling layer을 넣어주게 되는데 그 이유는 네트워크의 피라미터의 개수나 연상양을 줄이기 위해 representation의 spatial한 사이즈를 줄이는 것이다. 우리가 사용할 Max-Pooling은 Activation Map을 M x N크기로 잘라 낸 후 그 안에서 가장 큰 값을 뽑아내는 방법이다. 아래 그림은 4x4 Activation Map에서 2x2 Max-pooling필터를 stride 2를 줘서 Max-pooling을 한 예이다.

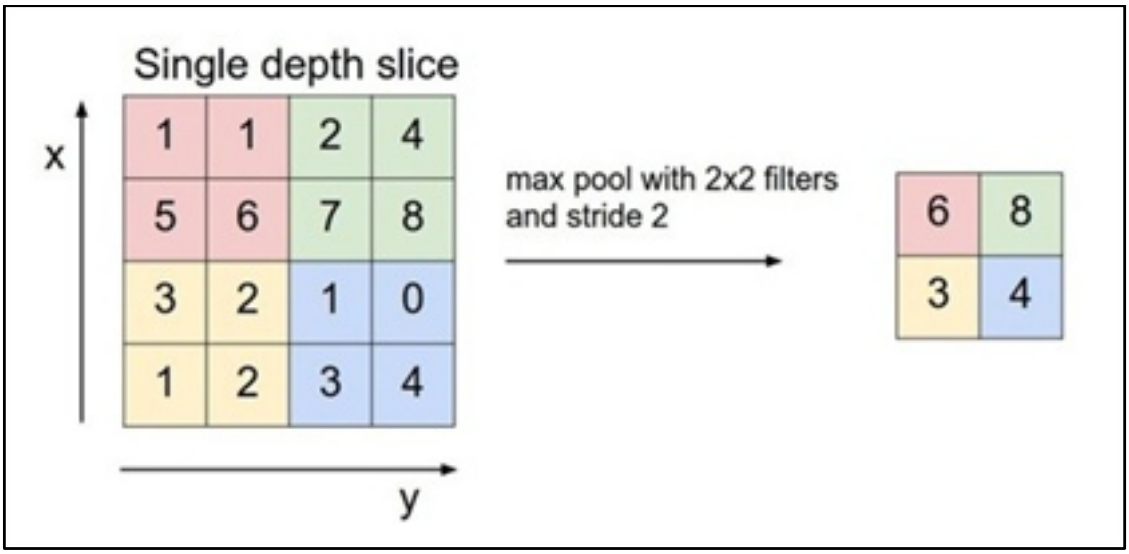


그림 16. Max Pooling

일반적으로 사용되는 대부분의 신경망에는 크기가 2*2보다 큰 Pooling Filter는 사용하지 않는다. 이는 이보다 큰 크기의 Filter를 사용한다면 결과가 손실이 많이 되기 때문이다.

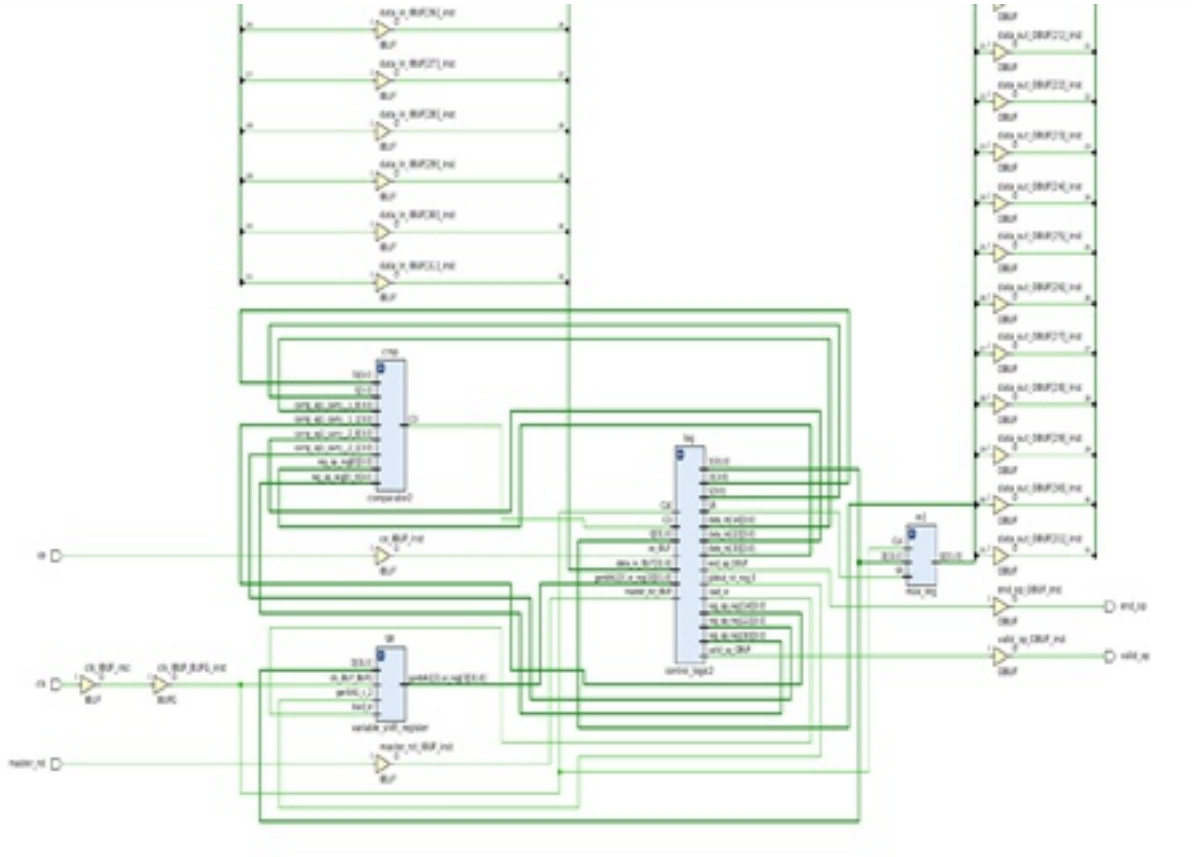


그림 17. Max Pooling 세부 블록도

Convolution이 파이프라인으로 동작하기 때문에, Pooling 또한 파이프라인으로 설계가 되어야 빠르게 동작이 가능하다. 즉 convolution에서 나오는 데이터를 매 클럭 사이클마다 Pooling에 input으로 들어가야 하고, 특정 사이클 후에 출력이 나타나야 한다. 이는 Max Pooling을 구현한 것으로 초기 조건으로 input matrix/feature map의 크기로 m*m, pooling filter의 크기 p*p를 초기에 설정해서 계산이 가능하게 개발을 진행하였다. 여기서부터 Control Unit이 필요하게 된다. Convolution과 Pooling 간에 연결이 필요하기 때문이다. Control Unit은 모든 제어 신호를 생성하고, 이 신호를 사용하여 Pooling의 동작 및 데이터를 알려준다. 내부에는 CU 이외에, max_reg(현재 가장 큰 value를 저장하고 있음), comparator(2개의 입력을 비교하여 큰 값을 선택), variable shift register로 구성된다.

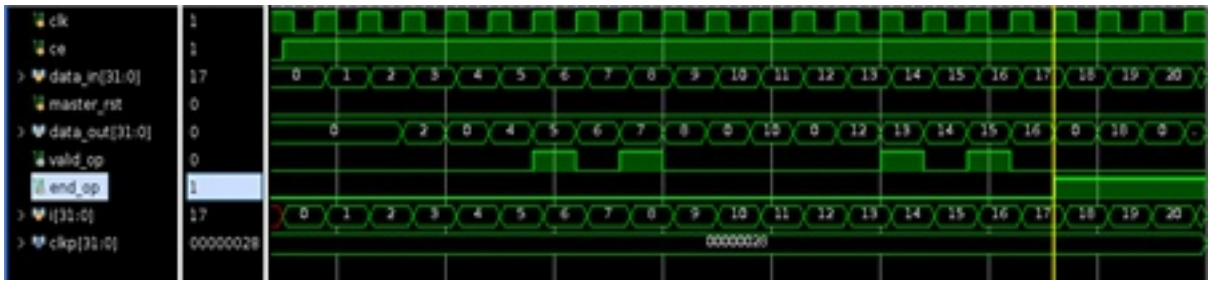


그림 18. Max Pooling Testbench

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

위와 같은 Input을 2*2 Kernel로 Pooling을 하게 되면 다음과 같다.

5	7
13	15

위의 Simulation 결과를 보면 Valid_op가 High가 되었을 때 5,7,13,15 결과를 확인할 수 있다. 이도 End_op가 1이 되었을 때부터는 계산하지 않아야 한다.

3.5. Fixed Point

실제로 데이터는 양수, 음수, 분수 등등 다양한 값들로 존재한다. 하드웨어는 단순한 Binary구조로 Register, Wire로 RTL Level의 디자인이 가능하다. 우리가 관심 있게 봐야

할 점 보통 binary를 보면 Decimal 값으로 생각하는 편이다. 하지만 하드웨어는 양수, 음수, 분수 등을 분류해서 계산하지 않는다. 정의한 대로 동작을 하게 될 것이다. 따라서 CNN에서 Weight 등을 계산하고 할 때 보통 Float형태를 사용한다. 하지만 하드웨어에는 이러한 개념이 없고 특정 bit와 그 크기만 인식을 한다. 따라서 하드웨어 연산을 용이하게 하기 위해 고정 소수점을 추가한 개념이 필요하게 된다. 제일 많이 사용되는 고정 소수점이 8bit 또는 16bit 환경이다.

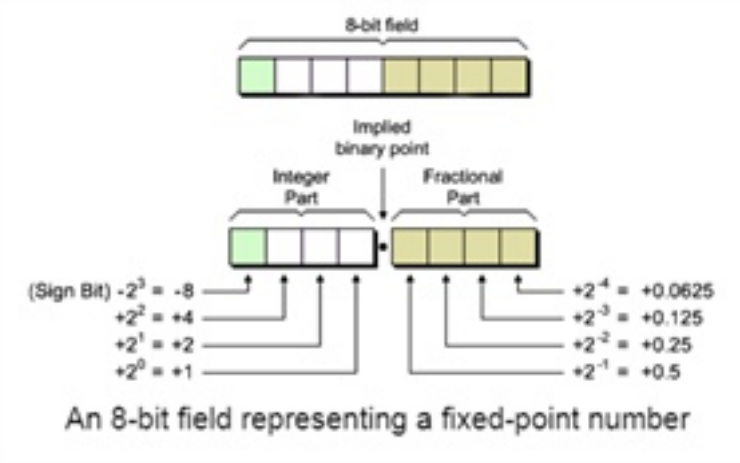


그림 19. 8-bit field

8-bit Field에서는 4비트를 정수형을 위해, 4비트는 소수점을 위해 사용한다. 위와 같이 정수형에서 1비트는 음수와 양수를 구분하는데 사용한다.

```
1 sign-bit + 3 integer-bits + 12 fractional-bits = 16 total-bits
```

1비트는 부호를 위해, 3비트는 정수 값, 나머지 12비트를 소수점을 나타낸다고 정의를 하여 총 16bit를 사용하는 고정 소수점이 포함된 Adder, Multiplication을 구현하였다.

3.6. IP Integrator

FPGA 보드에 올리기 위하여 ZCU104 보드에서 사용되는 Zynq UltraScale + MpSoC IP 내부에는 AXI bus와 RAM 등 보드 내부 Processor로 구성되어 있고 CNN 추론 연산을 진행할 DPU(Deep Learning Processing Unit) IP, 1번의 DPU동작이 끝나면 Interrupt를 발생시키고 처리해주는 Concat IP, 전체적인 System Reset을 하는 REST IP로 구성된다

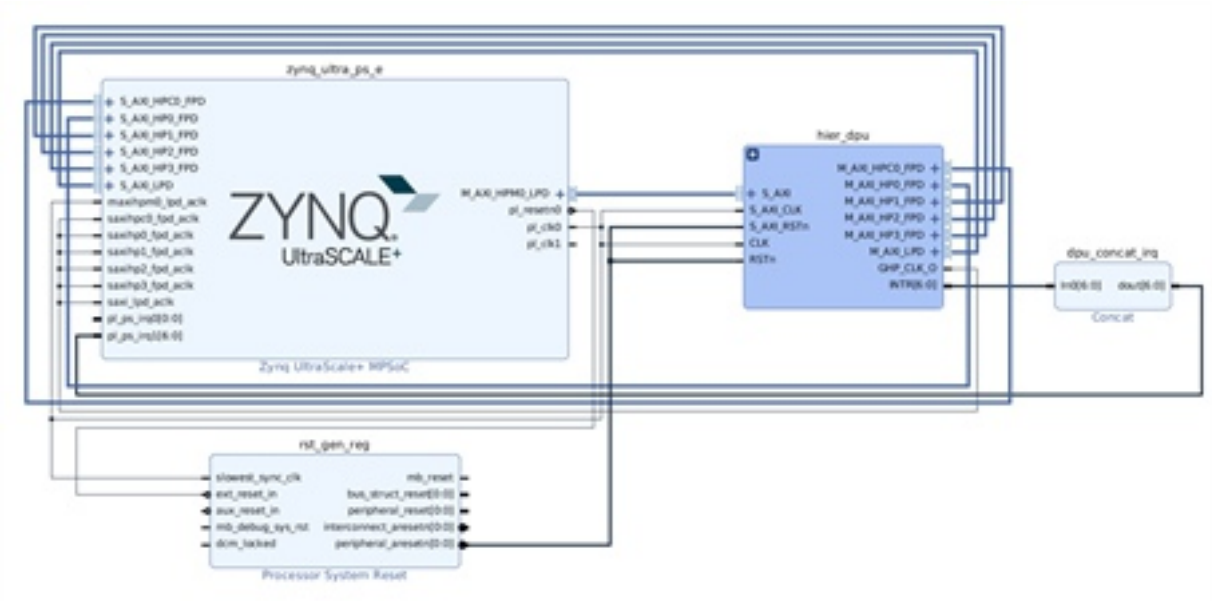


그림 20. ZCU104 보드에서 구현한 모습: FPD, LPD(DMA), Irq(Interrupt), S(slave), M(master)

Zynq_ultra_ps_e의 입력 시그널들을 보면FPD, LPD가 포함된 시그널은 DMA를 제어하기 위한 clk과 데이터로 구성된다. IRQ가 포함된 시그널은 Interrupt Request로 보드에 인터럽트를 발생시키거나 입력으로 받는 시그널이다. Hier_dpu의 입력 시그널은 데이터와 AXI BUS Slave 제어를 위한 clk,rstn, Slave 자체를 제어하기 위한 clk,rstn 시그널로 구성되며 출력에는 Master로 갈 데이터 들과 그 제어 clk, Interrupt를 발생시킬 시그널로 구성되어 있다. 그 dpu_concat_irq는 DPU로부터 발생한 Interrupt를 Master에 전달하는 용도로 사용되며, rst_gen_reg는 보드의 Reset을 하는데 사용된다.

DPU IP의 블럭도를 보면 다음과 같다.

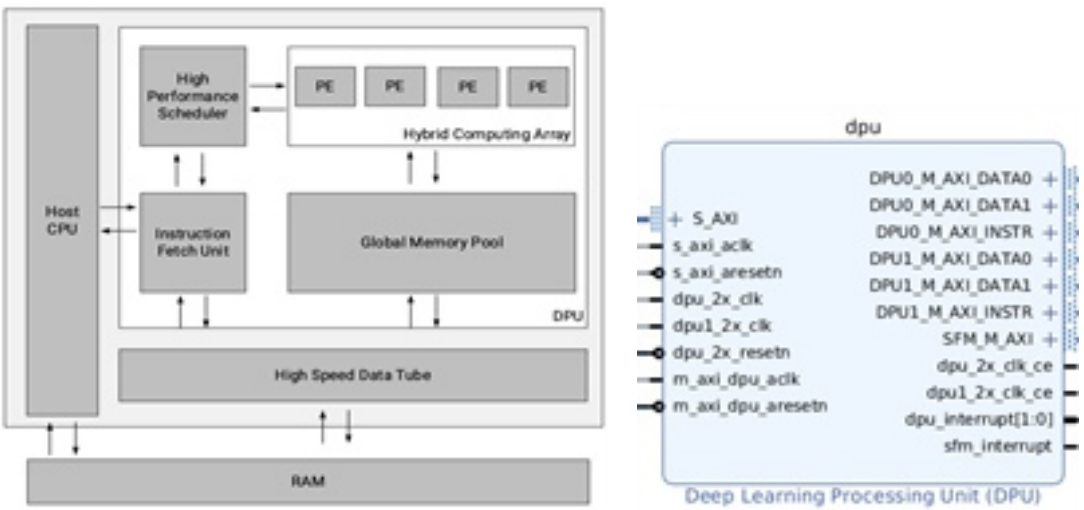


그림 21. DPU 블럭도 및 IP:PE(Processing element)

DPU는 CNN 연산을 하기 위한 Convolution, Pooling, fixed point, relu 모듈이 PE에 포함되며 이를 사용하기 위해서는 동작 시킬 명령어와 Global Memory Pool에 있는 특정 주소에 이미지 데이터를 줘야 한다. 결과는 Data Tube인 AXI Bus를 통하여 외부 메모리로 전달한다.

Signal Name	Interface Type	Width	I/O (In/Out)	Description
S_AXI	AXI slave interface	32	I/O	레지스터용 32비트 메모리 매핑
S_axi_aclk	Clock	1	I	S_AXI 제어용 clock
S_axi_arsetn	Reset	1	I	S_AXI 제어용 reset
Dpu_2x_clk	Clock	1	I	DPU안의 DSP 제어용 clock
Dpu_2x_resetn	Reset	1	I	DPU 안의 DSP 제어용 restet
M_axi_dpu_aclk	Clock	1	I	DPU 제어용 clock
M_axi_dpu_aresetn	Reset	1	I	DPU 제어용 reset
Dpux_m_axi_instr	AXI master interface	32	I/O	DPU에서 사용될 명령어
Dpux_M_AXI_Data(0,1)	AXI master interface	128	I/O	DPU에서 계산될 데이터DPU
Dpu_interrupt	Interrupt	1~3	O	DPU 계산 완료를 알려주는 Interrupt

다음으로는 상위 DPU의 1번의 작업이 완료되면 보낼 데이터를 Interrupt를 발생시켜 Zynq Ultrascale + MpSoC 보드에 알려주게 된다. 이러한 전체 IP를 Vivado에서 Generate Bitstream으로 컴파일하여 FPGA 보드에서 사용할 수 있게 한다. 이 이후 FPGA 보드에 들어가는 petalinux 및 사용될 OpenCV, tensorflow 등의 라이브러리가 포함된 Vitis AI Library를 이용하여 boot.BIN 파일을 만든 후 SD 카드에 넣는다.

3.7. FPGA 보드 실행 모습

보드에 전원, SSH를 실행하기 위해 내부 Network 연결용 Ethernet, petalinux 환경을 직접 사용하기 위한 DP, 키보드 및 카메라를 연결할 USB를 연결한 FPGA 보드 모습은 다음과 같다.

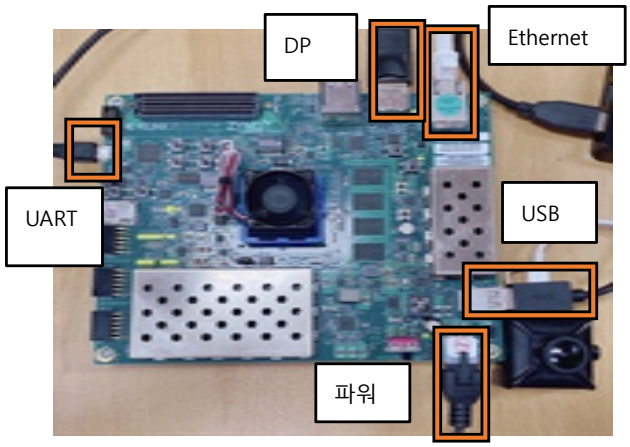


그림 22. ZCU104 구동 모습

SSH를 위해 보드에 연결된 Ethernet 주소를 설정하여 접속을 하면 다음과 같이 접속이 가능하다. 다만 UI 환경을 지원하려면 다른 Tool을 이용하여야 한다.

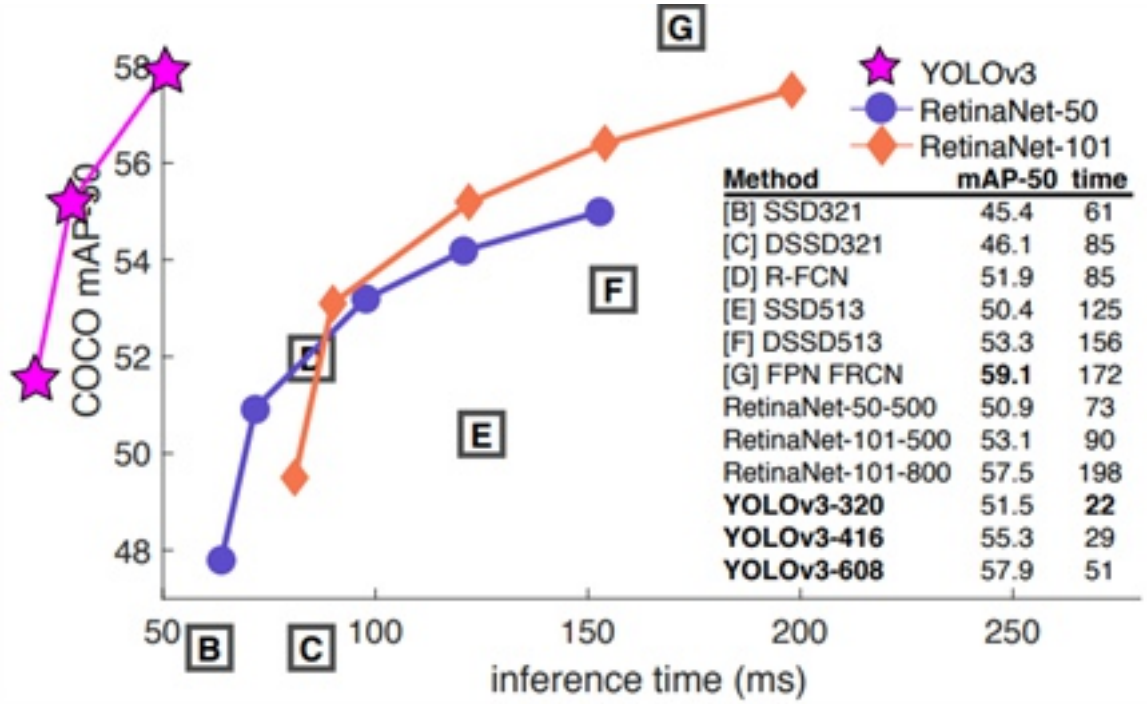
```
jonguk ~
ssh root@10.0.4.137
root@10.0.4.137's password:
Last login: Thu Sep 17 01:02:28 2020
mount: /mnt: /dev/mmcblk0p1 already mounted on /run/media/mmcblk0p1.
attempting to run /mnt/init.sh
/mnt
/etc/resolvconf/update.d/libc: Warning: /etc/resolv.conf is not a symbolic link to /etc/resol
vconf/run/resolv.conf
root@xilinx-zcu104-2019_2:~# ls
```

그림 23. ZCU104 SSH연결

보드에 리눅스 환경이 올라가 있기에 USB를 통해 마우스와 키보드를 연결, 모니터를 DP나 HDMI로 연결한다면 다음과 같이 사용할 수 있다. Linux 계열이기에 터미널 환경을 사용하여 이용한다.

그림 24. ZCU104 직접 연결

3.8. YOLOv3를 이용한 학습



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

그림 25. YOLOv3 성능비교 그래프

위 그래프는 YOLOv3(You Only Look Once)가 다른 모델들보다 더 빠르고 정확하다는 걸 보여주는 표이다. 우리는 현제 검출 정확도가 크게 저하되지 않으면서 최고 성능(즉, FPS가 높음)을 달성하는데 있어서 가장 뛰어나다고 평가되는 YOLO 프레임워크를 채택했다. 또한 우리가 사용하는 Slow fast net이 모델 두개를 사용함에 따라 YOLO를 사용하게 되었다.

YOLOv3를 이용해 학습시키기 위한 데이터들이 필요하기 때문에 YOLO-MARK를 이용해 데이터를 Labeling 하였다.

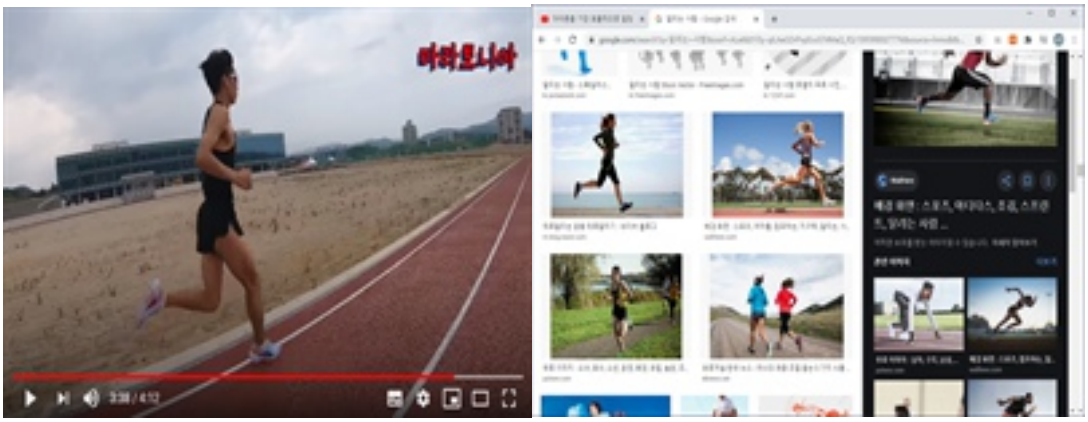
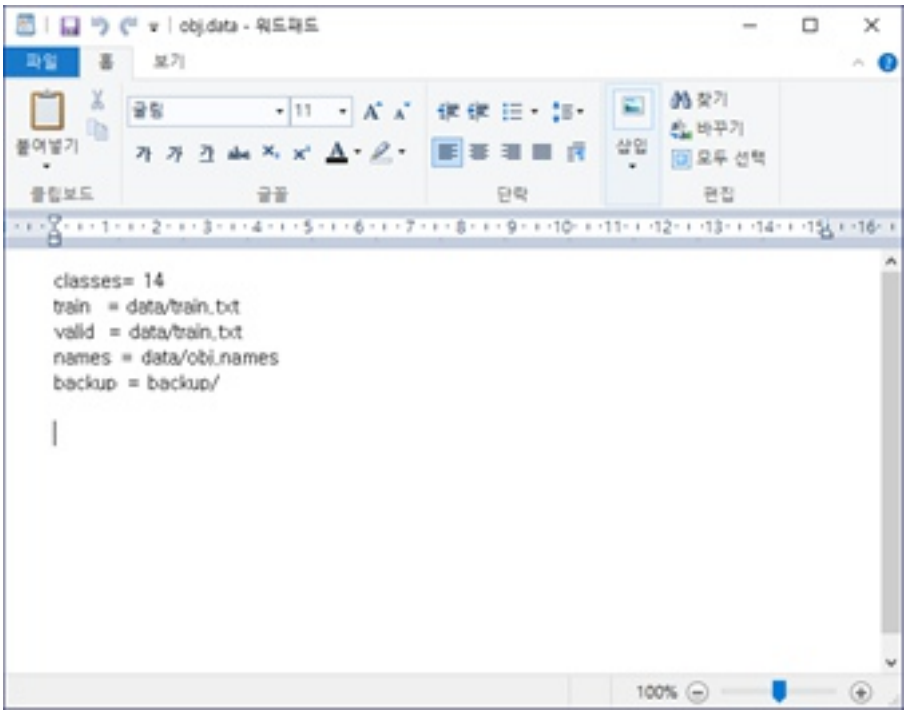


그림 26. 유튜브 & 구글 이미지 검색

학습에 사용하기 위한 액션들은 유튜브와 구글 이미지 검색을 통해 확보했다.



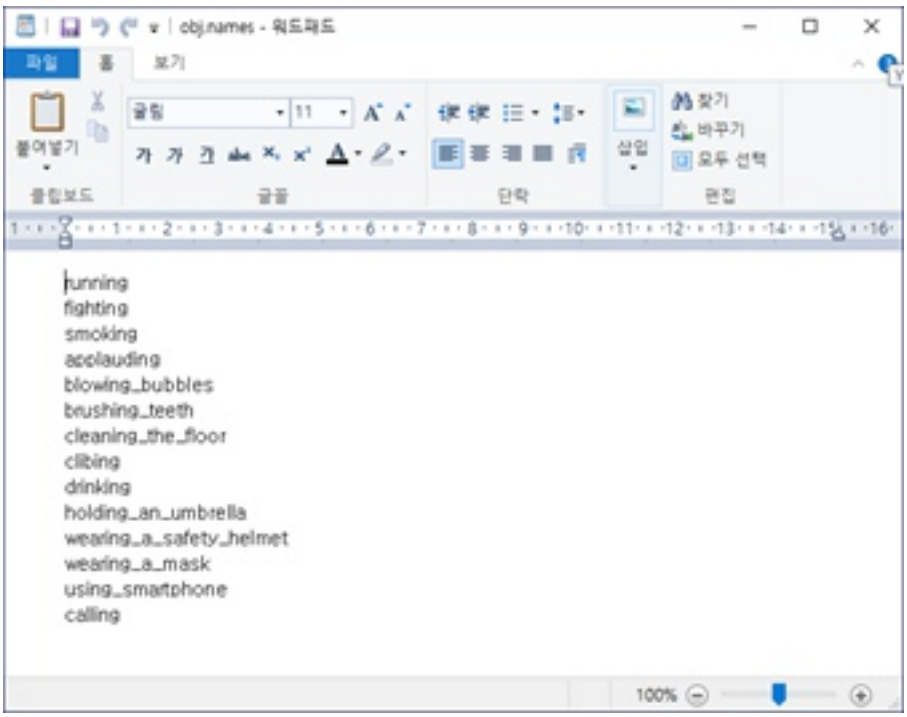


그림 27. obj.data & obj.names

학습시키고자 하는 데이터의 숫자, train 파일 위치를 obj.data 파일에 설정해 준 후, 학습시키고자 하는 데이터의 종류를 obj.names 파일에 입력한다.

그림 28. YOLO-MARK

유튜브와 구글 이미지로 확보한 데이터들을 img 폴더에 모은 후 YOLO-MARK를 실행하여 그림과 같이 이미지마다 물체의 이름이나 행동이 설정된 class 숫자를 설정해

준 후 Bounding Box를 그려서 Labeling을 진행했다.

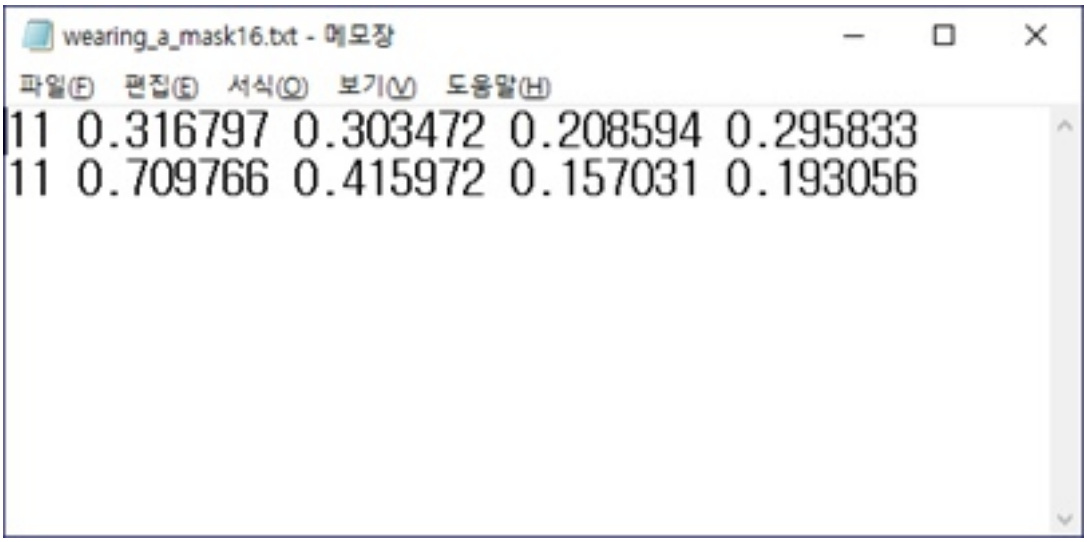


그림 29. labeling 결과 파일

Labeling이 끝난 이미지들은 txt 파일이 생성되어 mark 된 class 숫자와 해당하는 좌표 값이 출력된다.

Yolo mark를 통해 labeling 된 데이터들은 darknet yolo를 통해 학습하게 된다. Darknet yolo는 Nvidia가 만든 병렬 컴퓨팅 플랫폼 및 API 모델인 CUDA(Compute Unified Device Architecture)를 이용해 빠르게 학습을 할 수 있다. 학습에 필요한 Convolution layer는 darknet에서 제공하는 darknet53.conv.74를 이용했다.

학습에 앞서 cfg파일을 설정해 주었다. batch값은 한 번에 얼마나 많은 데이터를 배치할지를 정하는 값이다. Subdivisions는 batch의 값을 얼마나 쪼개서 학습할 건 지에 관한 설정이다. Batch 값은 높을수록, Subdivisions는 낮을수록 좋은 GPU성능을 요구하는 대신 빠른 학습을 할 수 있게 한다. Max_batches는 언제까지 iteration을 돌 건지 설정하는 값이다. Class * (원하는 횟수) 만큼의 값을 지정해 준다. 우리는 4000번의 횟수를 설정해 주었다. Filters는 필터의 숫자이다. (class의 수 + 5) * 3의 값을 설정해 주어야 한다. Anchor box는 탐지할 물체의 형태를 미리 지정해 주는 box이다. 예를 들어 자동차는 보통 세로보다 가로가 길고 사람은 가로보다 세로가 길다는 특징을 가지고 있다. 이렇게 설정이 끝난 cfg 파일은 데이터와 darknet54.conv.74를 이용해 학습하게 된다.

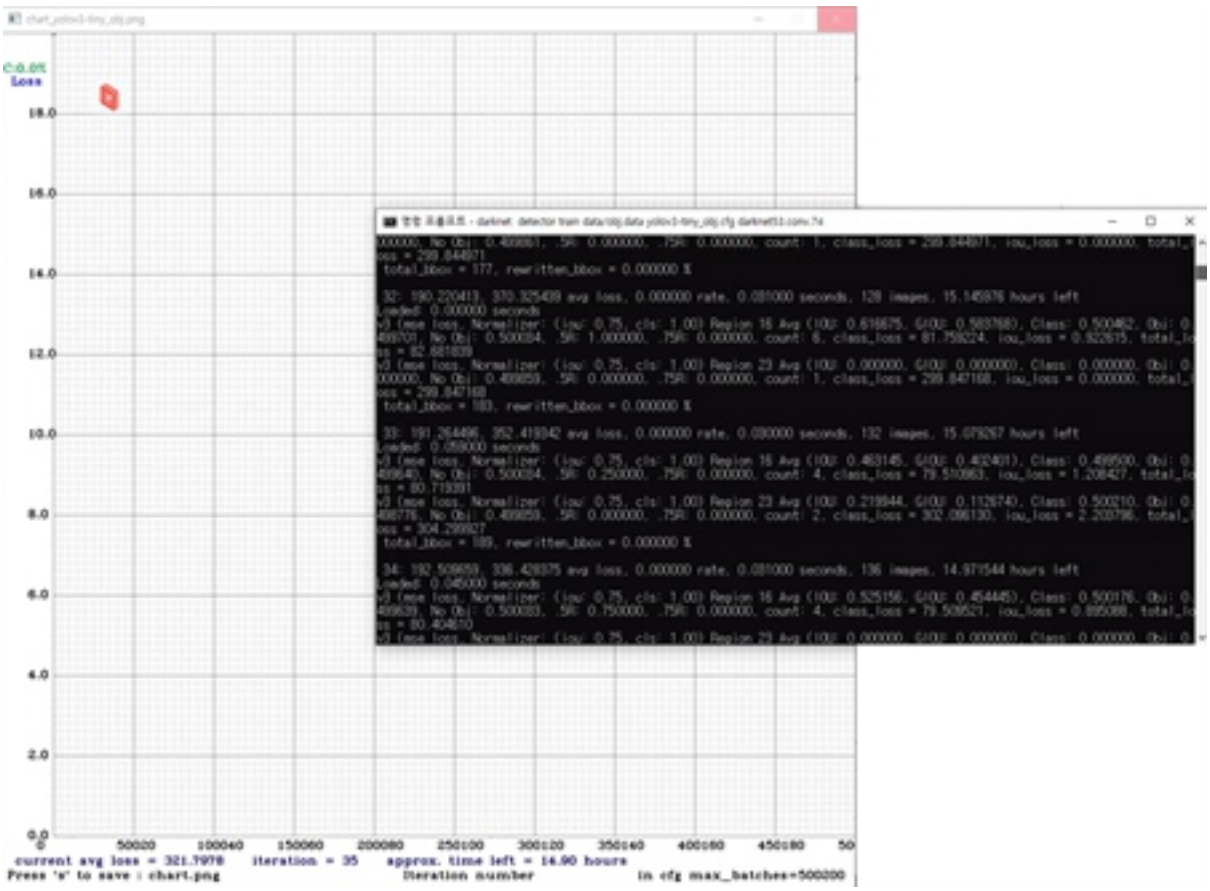


그림 30. Loss값을 나타내는 그래프

위 그림과 같이 학습을 시작하면 Loss값을 나타내는 그래프와 함께 학습이 시작된다. loss값은 iteration이 증가할수록 loss avg값이 줄어드는 것을 볼 수 있다. 학습 중에 볼 수 있는 Avg IOU는 현제 subdivision에서 이미지의 평균 IOU를 나타낸다. 실제 GT와 예측된 bbox의 교차율을 의미한다. 1에 가까울수록 학습이 잘되어가고 있다는 뜻이다. Region 16은 가장 큰 mask이다. Prediction scale을 이용하는 layer지만 작은 객체를 예측할 수 있다. Region 23은 가장 작은 mask이다. 마스크가 작을수록 큰 객체를 예측할 수 있다.

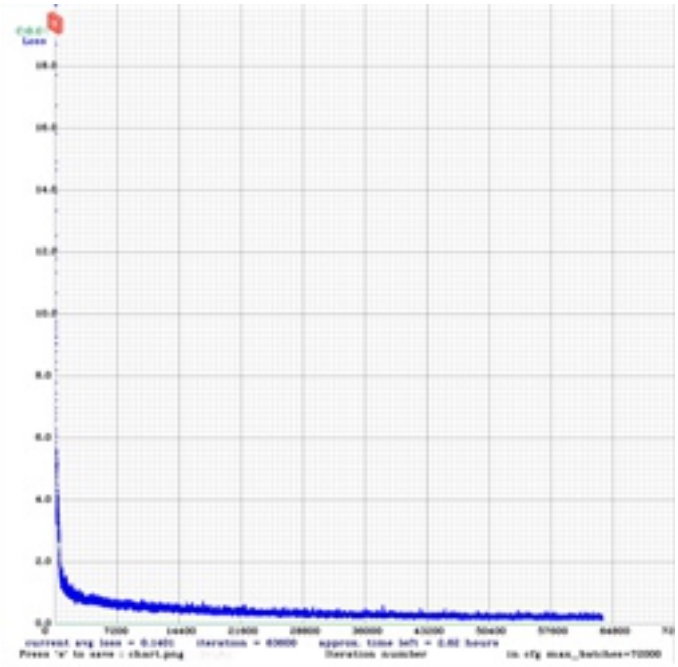


그림 31. 학습이 완료된 그래프

학습이 완료되면 완료된 Loss 그래프와 함께 weights 파일이 저장된다.

Darknet yolo로 학습된 가중치인 weights파일은 tensorflow를 이용해 동작 시키기 위해 파일을 변환해 주어야 한다. 우리는 DW2TF 툴을 이용해 먼저 weights파일과 cfg파일을 pb파일과 ckpt파일로 변환시켰다.

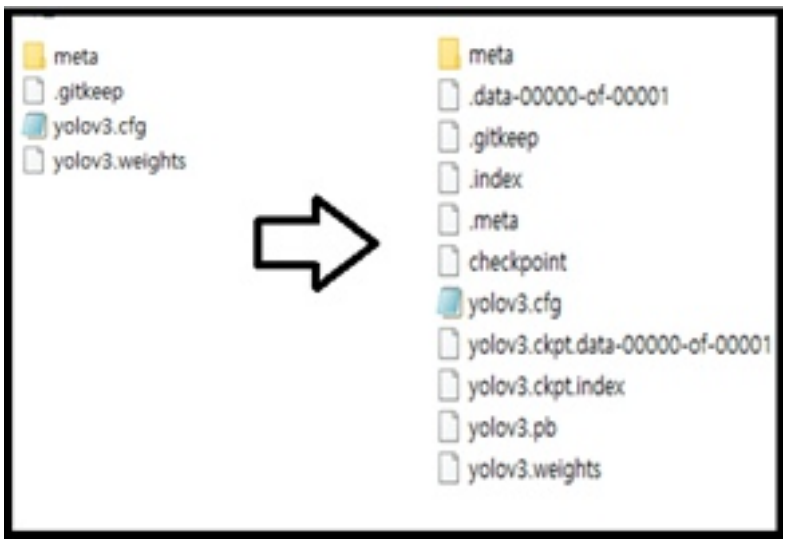


그림 32. pb, ckpt파일 생성

변환에 성공하면 위 그림과 같이 pb파일과 ckpt파일이 생성된다. 변환 과정에서 가중치의 그래프는 아래 그림과 같다.

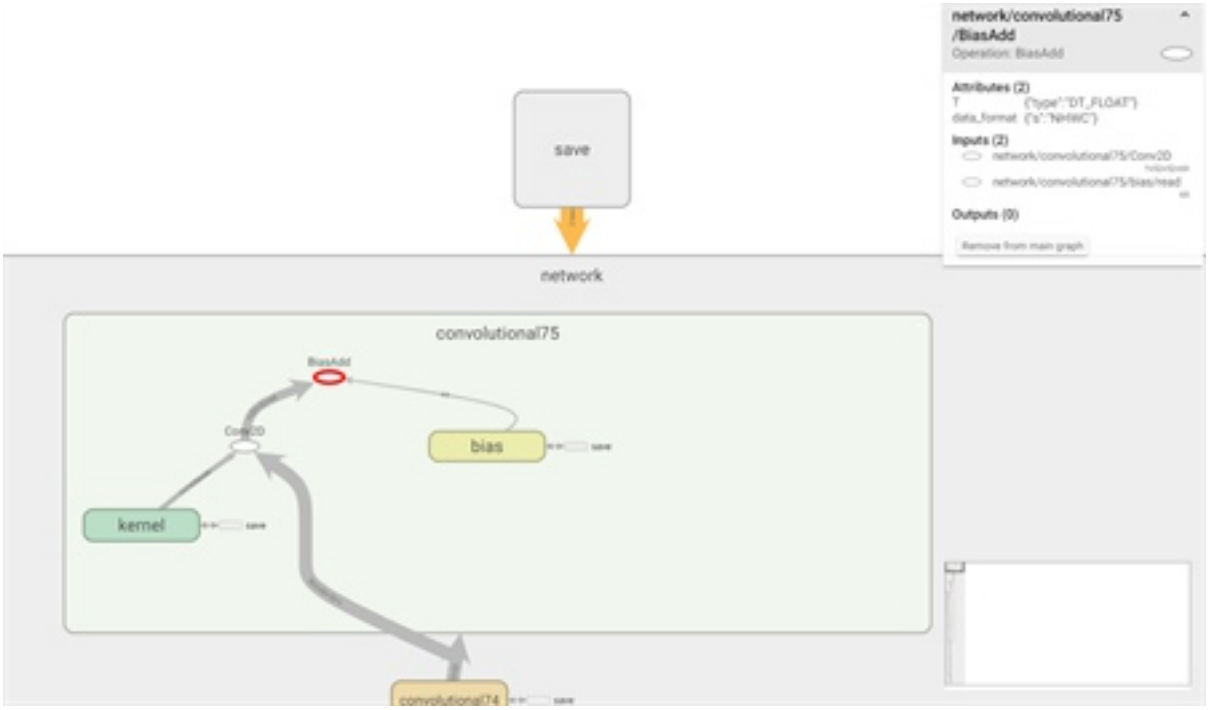


그림 33. 학습 결과

확인한 후 필요한 노드는 Tensorflow 내부에 있는 freeze_graph.py를 이용해 출력한다.

```
C:₩Users₩ParkSeongchan>python freeze_graph.py --input_graph yolov3.pb
 --input_checkpoint yolv3.ckpt --output_graph frozen_graph.pb --outpu
t_node_names network/convolutional75/BisaAdd --input_binary True
```

그림 34. freeze된 pb파일 출력

위 그림과 같이 input graph 파일과 input checkpoint파일 그리고 output graph 파일 이름과 출력할 노드의 이름을 입력하면 freeze된 pb파일이 나오게 된다.

3.9. FPGA

만들어진 AI 모델을 보드에서 동작하게 하기 위해서는 보드에 올린 라이브러리의 버전을 맞추어 둔 도커 환경에서 Quantization, Compiling 과정을 진행해야 한다. 이 환경은 보드와 똑같은 환경이므로 미리 테스트도 진행할 수 있다. 도커에서 AI 모델의 yolov3.elf, yolov3.prototxt, data_list.txt, meta.json 파일을 만들어야 한다. 여기서 필요한 yolov3.elf 파일은 양자화와 컴파일 과정이 끝난 고정된 weight를 가진 Yolov3 그래프를 나타낸다.

양자화란 연산량을 줄이면서 전력 효율성을 향상시키는 방법 중 하나이다. 양자화된 신경망은 메모리 액세스를 줄이고 연산 효율성도 높이기 때문에 전력 효율성이 향

상된다. 우리는 Vitis AI로 32bit 부동 소수 점을 8bit 고정점으로 전환하였는데, 이것은 모델 정확도와 효율적인 하드웨어 구현 사이의 좋은 타협점이다. 신경망을 하드웨어에서 진행하면 곱셈 및 덧셈 연산을 수백만 회 실행한다. 만약 양자화된 매개변수로 lower-bit 의 수학연산을 수행하고, 신경망의 중간 계산 값도 함께 양자화 한다면, 연산 속도는 빨라지고 성능도 향상된다.

양자화가 끝난 CNN모델을 구문 분석하여 데이터 흐름과 제어 흐름으로 구성된 연산 그래프를 생성한다. 그런 다음 배치 표준화된 레이어 통합 및 데이터 재사용 등의 프로세스를 통해 데이터를 최적화하고 흐름을 제어한다. 마지막으로 실행할 코드를 생성하고 모델은 커널로 분할되며, 일부는 DPU에서 실행되고 다른 일부는 CPU에서 실행된다. 출력이 된 .gv를 .png파일로 변환하여 커널 그래프를 볼 수 있다.

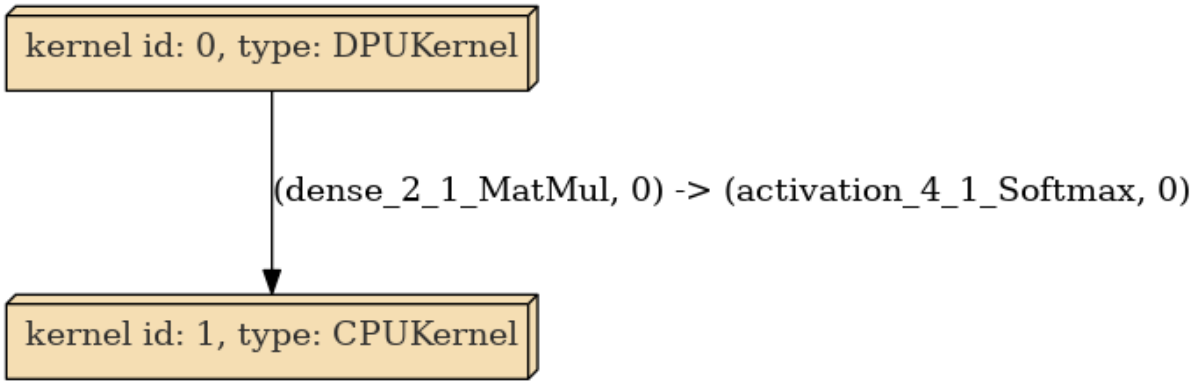


그림 35. 분리된 커널

이렇게 양자화와 컴파일 과정이 끝나면 Weight가 고정된 추론용 모델 elf 파일이 생성되고, prototxt가 생성되는데 이 파일은 파라미터에 관한정보 즉 클래스가 몇 개인지, 출력 Layer의 정보, 임계값 정보와 bias등의 정보를 얻을 수 있다. 또한 meta.json이라는 파일은 타겟이 어떤 것인지 어떤 라이브러리 인지를 확인할 수 있다.

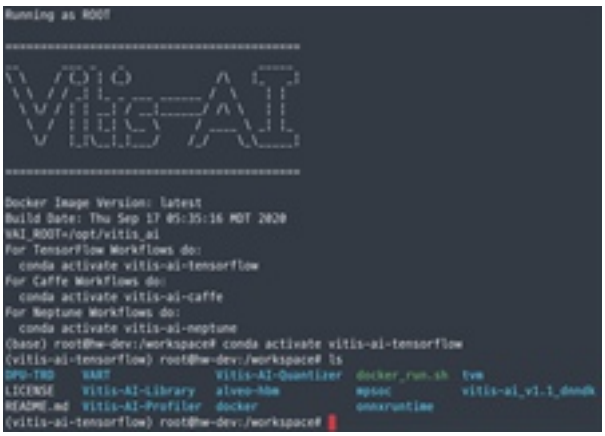


그림 36. Docker 실행 화면

4. 연구 결과 분석 및 평가

4.1. 설계 변경 내역

기존 조사한 바로는 FFT(Fast Fourier Transform)를 이용한 Convolution 연산을 구현 시에 장점이 많을 줄 알았지만, FFT를 사용시에 Convolution 에서는 효율적일지는 몰라도 Activation, Pooling 등을 사용 시에는 다시 FFT 역변환을 통하여 변환된 상태에서 계산을 진행해야 한다. 이를 효율적으로 바꾼 논문들도 존재하였지만, 대부분 면적에서의 최적화를 위해 변형된 FFT를 사용하는 편이였고, 우리는 FPGA를 사용하기 때문에 FFT 구현하여 직접 사용하기 보다는 DSP(Digital Signal Processing)사용으로 파이프라인 과 같은 병렬성을 더 높이는 방향으로 진행을 하기로 하였다. 또한 AXI 버스 및 인터페이스 부분도 구현을 하고자 하였지만, 고려할 사항 및 에러가 많아 IP를 사용하기로 하였다.

CCTV를 위한 AI 모델을 설계 시에는 기존에 사용하고자 했던 Slow Fast Net은 하드웨어상에서 구현에 문제가 있을 것으로 예상이 되어 모델링 시에 2개의 모델이 필요하게 되어 효율적인 YOLOv3로 바꾸어 설계를 진행하였다.

Xilinx 사의 Vitis AI Library를 분석하는데, 여기서도 Kernel 단계에서의 코드는 공개하지 않았다. 그래서 Framework 단계에서 모델을 설계하고 모델에서 Hardware 로직을 사용하기 위해서는 Xillix사에서 제공하는 IP를 사용해야한다. Convolution 및 Relu, Pooling 기능을 verilog로 구현 및 Processor와 연동 결과 IP로 제공되는 것보다 성능이 떨어져 Xilinx IP를 사용하였다.

4.2. 연구 결과

1. 아래 그림 37은 실행을 위한 필요 디바이스와 실행시키기 전의 사진이다. 셋팃이 끝난 후 실행을 시키면 아래 그림 38과 39처럼 마스크 착용, 흡연, 안전모 착용 등 정상적으로 빠르게 실행이 되는 것을 확인할 수가 있다. 테스트를 진행하기 위한 필요한 물품은 다음과 같다. 키보드, 마우스, 모니터, 보드 및 전원, 카메라 이다.

그림 37. 실행 전 필요 디바이스

아래 그림은 보드에서 수행결과를 모니터로 보여준 것이다. 마스크의 색깔에 상관없이 인식하며 안전모를 썼을 시에는 wearing a helmet 이라는 단어가 Box 위에 나타나며 마스크를 사용했을 시에는 wearing a mask라고 뜬다. 마스크를 벗으면 인식하지 못한다.

그림 38. ZCU14 FPGA Yolov3 실행 결과

아래 그림은 물을 마실 때 drinking 이라는 것과 담배를 필 시에 smking 이라는 것이 Box 위에 label로 나타난다. FPS는 평균 13.5 정도로 처리속도가 빠르게 나왔다.

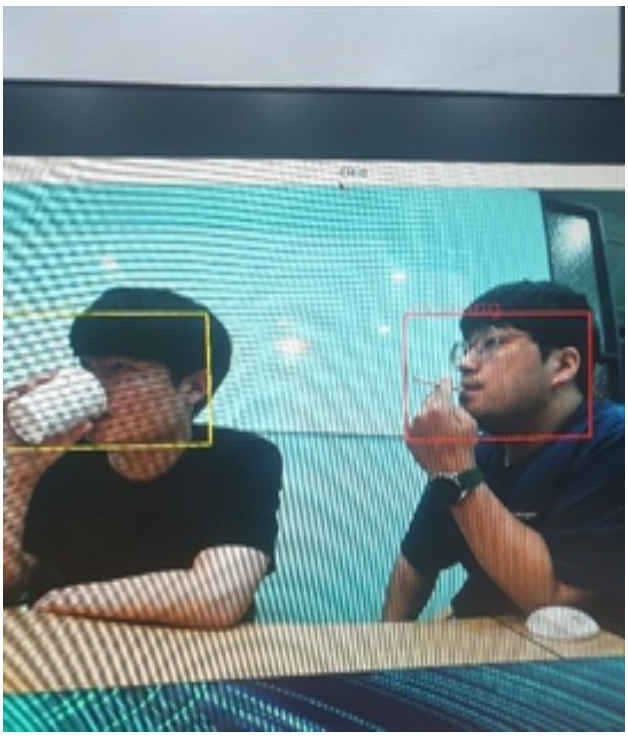


그림 39. ZCU104 FPGA Yolov3 실행결과

아래 그림은 최근에 나온 라즈베리 파이4에서 Yolov3-tiny를 수행해본 결과로 Yolov3에 비해 tiny는 Layer 수가 절반이 적음에도 불구하고 FPS가 0.3 정도로 처리를 거의 하지 못하는 정도였다.

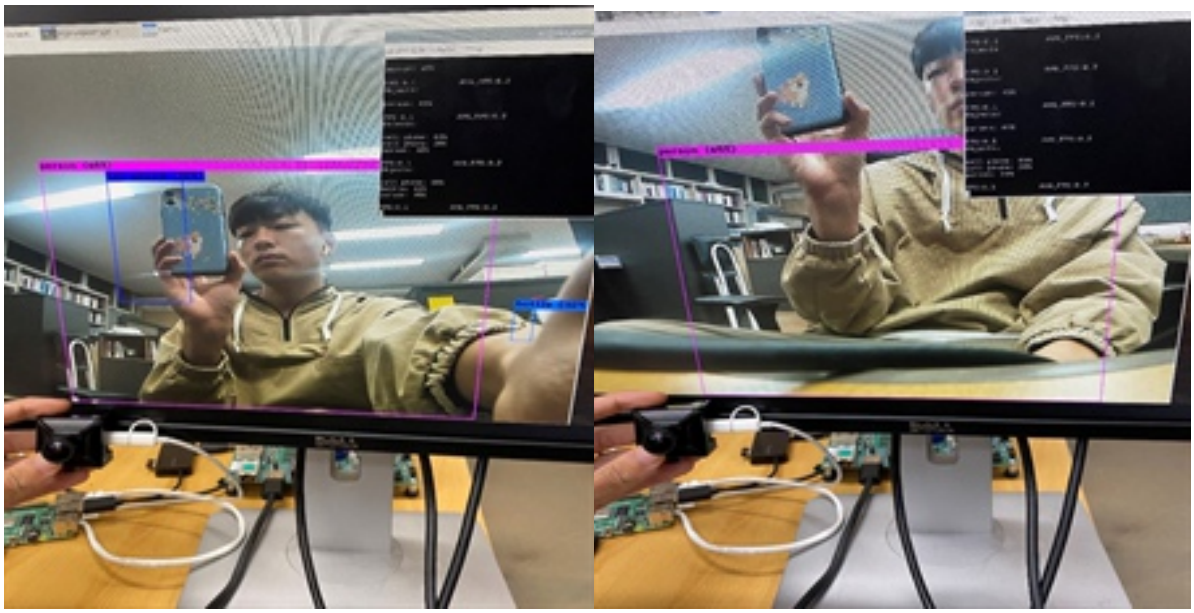


그림 40. 라즈베리 파이4 Yolov3-tiny 수행 결과

GPU가 없는 4.2GHz 쿼드 코어 i7을 이용하여 Yolov3를 돌려보았지만 FPS가 2.75 정도로 처리 속도가 빠른 편은 아니였다.

그림 41. PC Yolov3 수행 결과

그림 41. PC Yolov3 수행 결과

이렇게 다른 모듈로 실행했을 때와의 결과를 비교했을 때, 영상처리 속도가 확연히 눈에 보였다. 연구 결과 A.I 가속기를 개발한 모듈이 다른 모듈보다 영상처리 속도와 정확성이 높은 것으로 확인했다.

표 1. Yolov3 수행 결과 비교표

이름	FPS	CPU
ZCU104	13.5	Coretex-A53, Coretex-R5
라즈베리 파이4	0.3	ARM Coretex-A72 1.5GHz
아이맥 PC	2.75	쿼드 코어 Intel i7 4.2GHz

5. 결론 및 향후 연구 방향

현재 상품으로 판매되고 있는 IoT 제품에는 얼굴 인식 정도의 기능과 다른 몇가지의 센서로부터 받는 데이터를 분석하는 제품이 팔리고 있다. 이러한 제품과의 차이점은 그러한 제품은 한가지 용도로 밖에 쓰지 못한다는 것이다. FPGA 보드에 일반적인 추론 연산을 하는 하드웨어 로직을 추가하였기 때문에, 커스텀한 AI 모델을 보드 형식에 맞게 올려 놓는다면 1가지의 AI 모델이 아닌 다른 AI 모델을 사용할 수 있다는 점이 장점이다. GPU 비슷한 속도의 FPS를 측정할 수 있었고, GPU 대비 면적, 가격, 전력 소모량에서 확실한 이득을 볼 수 있었다.

향후 연구를 한다면 CNN 연산 기반이 아니라 DNN, RNN, ANN과 같은 다양한

머신러닝 분야에 대해서 연산하는 하드웨어 로직을 개발할 수 있을 것이다. 또한 RISC-V ISA(Instruction Set Architecture)에 적용하여 명령어 형태로 개발한다면 Fetch, Decode, Execute 단계에서 1개의 convolution layer 연산에 필요한 명령어가 1개로 줄어들 수 있으므로 성능 및 속도가 굉장히 증가할 것이다. 또한 Thread 개념을 추가하여 동작할 수 있게 한다면 분산형 처리가 가능해서 이 또한 속도도 올라가게 될 것이다.

또한, 아직까지는 사생활침해 이슈에 대해 해결책이 없는 상황이지만, 향후 인지능 CCTV 시장은 본연의 기능 외에도 다양한 활용성을 살려 지속적으로 성장할 것으로 보아 기술개발을 지속적으로 하여 글로벌 시장선점에 주력해야 한다. CCTV시스템의 사용 가이드라인을 조속히 제도화하고, 국내 중견, 대기업의 기술시장 참여를 진작시키는 것이 중요하다. 이를 통해 사생활침해 논란을 해소하고, 해외 제품의 국내시장 침투를 막아 국내 기술경쟁력을 향상시킬 수 있는 노력이 필요하다.

6. 개발 일정 및 역할 분담

6.1. 개발 일정

6월				7월					8월					9월				
2주	3주	4주	5주	1주	2주	3주	4주	5주	1주	2주	3주	4주	5주	1주	2주	3주	4주	5주
AI Inference 스터디																		
			AI Accelerator 및 AI Library 코드 분석															
						AI 서비스 모델 개발												
							중간 보고서 작성											
								Hardware Design & FPGA 설정										
												FPGA에서 모델 평가						
													설계 문서 작성					

														안정성 및 성능 평가			
															오류 수정 및 문제점 파악		
															최종 보고서 작성 및 발표 심사 준비		

6.2. 역할 분담

이름	역할 분담
박종욱	- FPGA 상에서 Convolution 연산 설계 및 Bus, interface 연동
김문석	- 이상탐지 AI 모델 개발 및 모델 FPGA 연동
박성찬	- 이상탐지 AI 모델 개발 및 모델 FPGA 연동
공동	- AI Accelerator 및 library 분석 - 성능 평가 - 설계 문서 작성 - 발표 심사 및 시연 준비

7. 참고 문헌

[1] KAIYUAN GUO. A Survey of FPGA-Based Neural Network Inference Accelerator, New York, 2018

[2] Kamran Chitsaz, Acceleration of Convolution Neural Network Using FFT-Based Split Convolutions, .2020

[3] Paolo meloni, NEURAghe:Exploiting CPU-FPGA Synnergies for Efficient and Flexible CNN Inference Acceleration on Zynq SoCs, ACM journal, 2018

[4] Kaiming He, Deep Residual learning for Image Recognition, CVPR, 2016

[5] Christoph Feichtenhofer, SlowFast Networks for Video Recognition, arXiv, 2019

[6] Joseph Redmon, YOLOv3: An Incremental Improvement, arXiv, 2018

[7] Abdul Montaqim. Available: https://roboticsandautomationnews.com/2019/05/24/top-25-ai-chip-companies-a-macro-step-change-on-the-micro-scale/22704/ (downloaded 2019, May, 24)

[8] Xilinx, guideline, Available: https://www.xilinx.com/support/documentation/boards_and_kits/zcu104/ug1267-zcu104-eval-bd.pdf

[9] Xilinx, guideline, Available: https://www.xilinx.com/applications/megatrends/machine-learning.html

[10] Github, qianglin-xlnx, Available: https://github.com/Xilinx/Vitis-AI