공학석사 학위논문

# Inception V4 Network의 FPGA 구현을 위한 데이터 재사용 최적화

Data reuse optimization for an FPGA implementation of Inception V4 Network

2021 년 02 월

서울대학교 대학원

전기 정보 공학부

송 병 기

# Inception V4 Network의 FPGA 구현을 위한 데이터 재사용 최적화

지도 교수 이 혁 재

이 논문을 공학석사 학위논문으로 제출함
2021 년 02 월

서울대학교 대학원
전기 정보 공학부
송 병 기

송병기의 공학석사 학위논문을 인준함
2021 년 02 월

위 원 장 김 태 환 (인)

부위원장 이 혁 재 (인)

위 원 류 수 정 (인)

# 초 록

최근 컴퓨터 비전 분야에서 Deep Convolution Neural Network가 높은 성능을 보이고 있고, FPGA를 이용하여 이미지 추론을 가속하는 연구가 활발히 진행되고 있다. Deep CNN은 깊은 네트워크 특성때문에 많은 양의 weight 파라미터와 중간 feature map 데이터를 생성한다. 이로 인해, FPGA 상에서 추론할 때 많은 off-chip 메모리 접근을 하게 되고 이는 가속기 추론 속도 성능과 에너지 효율의 bottleneck으로 작용한다.

위 문제를 해결하기 위해 한 번 off-chip 메모리에 접근하여 가져온 데이터를 on-chip에서 최대한 재사용하는 방법들이 소개되었다. 하지만 기존의 데이터 재사용 방법들은 이미지 분류에서 높은 성능을 보이는 Inception V4 네트워크에 최적의 결과를 내지 못하는 모습을 보인다.

본 논문에서는 Inception V4 네트워크의 branch 구조를 고려하여 데이터를 on-chip에서 최대한 많이 재사용하는 Mixed convolution 방법을 제안한다. Mixed convolution은 Inception 모듈의 입력 feature map 데이터를 재사용하는 Grouped convolution과 branch 내에서 생성되는 중간 feature map 데이터를 재사용하는 Fused convolution을 모두 사용하는 것으로 2가지 방법의 장점을 모두 이용한다. 그 결과, Inception 모듈에서 생성되는 feature map 데이터에 대해서 421KB의 추가 on-chip 버퍼 메모리를 사용하여 off-chip 메모리 데이터 전송량을 37MB에서 12MB로, baseline대비 66.4% 감소시켰다. 또한, on-chip 버퍼 메모리를 최적화하기 위해 Inception-C 모듈에 full weight 재사용 방법을 사용함으로써 218KB의 추가 on-chip 버퍼

메모리를 사용하여 off-chip 메모리 데이터 전송량을 11MB로 더욱 줄여 baseline대비 68.6% 감소시켰다.

**주요어 : 데이터 재사용, Off chip 메모리 접근 데이터 크기, CNN 가속기, Inception V4 네트워크**
**학 번 :** 2019-20004

# 목 차


제 1 장 서 론 ..... 1
1.1 연구의 배경 및 내용 ..... 1
1.2 논문 구성 ..... 4

제 2 장 관련 연구 ..... 5
2.1 Inception Network ..... 5
2.2 FPGA 가속기 연구 ..... 9
2.2.1 Off chip memory 접근 관련 연구 ..... 9

제 3 장 Inception v4 Network의 데이터 재사용 방법 ..... 14
3.1 Grouped Conovlution ..... 14
3.1.1 Grouped Conovlution의 동작 방식 ..... 16
3.1.2 Grouped Conovlution의 on chip 버퍼 ..... 22
3.2 Fused Conovlution ..... 23
3.2.1 Fused Conovlution의 동작 방식 ..... 24
3.2.2 Fused Conovlution의 fused 버퍼 크기 ..... 26
3.3 Mixed Conovlution ..... 27
3.3.1 Mixed Conovlution의 동작 방식 ..... 27
3.3.2 Mixed Conovlution의 on chip 버퍼 크기 ..... 30
3.4 Full weight 재사용을 적용한 Mixed Conovlution ..... 31

제 4 장 실험 결과 및 분석 ..... 33
4.1 FPGA 하드웨어 가속기 시스템 ..... 33
4.2 가속기 모듈 내부 구조 ..... 34
4.3 Mixed convolution을 지원하는 Data Controller 구현 ..... 37

제 5 장 실험 결과 및 분석 ..... 39
5.1 Off-chip 메모리 데이터 전송 크기 비교 ..... 39
5.2 On-chip 버퍼 크기 비교 ..... 42
5.3 Full weight 재사용과 on-chip 버퍼 크기 분석 ..... 45
5.4 FPGA 리소스 사용량 비교 분석 ..... 46

제 6 장 결론 ..... 48

참고문헌 ..... 49

Abstract ..... 51

# 표 목차

표 5.1 Baseline의 Inception 모듈 별 off-chip 메모리 데이터 전송량 .....40
표 5.2 On chip 버퍼 크기 비교 .....44
표 5.3 종류별 on chip 버퍼 크기 비교 .....44
표 5.4 Full weight 재사용 dataflow 적용 시점에 따른 on-chip 버퍼 크기 비교 .....45
표 5.5 FPGA 리소스의 사용량 비교 분석 .....47

# 그림 목차

그림 2.1 GoogleNet에 적용된 Inception 모듈 구조 ..... 5
그림 2.2 Inception V4 network 구조 ..... 7
그림 2.3 Fused Convolution의 dataflow .....11
그림 2.4 Inception V4 network의 모듈별 데이터 크기 .....13
그림 3.1 Inception V4 network의 Inception 모듈 .....15
그림 3.2 기존 Single layer processing 가속기의 layer 처리 시간별 분석 .....17
그림 3.3 Grouped convolution layer 처리 시간 별 분석 .....19
그림 3.4 Grouped convolution의 첫 번째 layer 처리 과정 ..19
그림 3.5 Grouped convolution의 두 번째 layer 처리 과정 ..20
그림 3.6 Grouped convolution의 세 번째 layer 처리 과정 ..21
그림 3.7 Plane 2에 대한 Grouped convolution 반복 과정 ...21
그림 3.8 Fused convolution의 연산 과정 .....25
그림 3.9 Inception 모듈별 Mixed convolution 적용 .....29
그림 4.1 FPGA 하드웨어 가속기 시스템 .....33
그림 4.2 가속기 모듈 내부 구조 .....34
그림 4.3 Data Controller를 이용한 on-chip 버퍼 관리 .....37
그림 5.1 Inception 모듈별 off chip 메모리 데이터 전송 크기 비교 .....40

# 제 1 장 서 론

## 1.1 연구의 배경 및 내용

지난 몇년간, 컴퓨터 비전 분야는 Deep Convolutional Neural Network(Deep CNN)의 발전으로 눈부신 정확도 향상을 이루어 냈고, 이를 실생활에 활용할 수 있도록 하는 연구가 활발히 진행되고 있다. Deep CNN은 모델 네트워크를 이루는 layer를 깊게 쌓아 만든 구조를 갖는데, 이러한 깊은 망 구조는 일정 수준의 정확도 성능을 얻기 위하여 필수적인 요소로 작용하고 있다. 실제로 이미지 분류기 네트워크인 Inception 네트워크[3],[4],[5] 혹은 ResNet 네트워크[6], 객체 검출기 네트워크인 YOLO 네트워크[7]는 모두 깊은 망 구조를 취하고 있다.

깊은 망 구조를 갖는 Deep CNN은 layer의 수가 증가함에 따라, weight 파라미터 크기와 중간 feature map 데이터 크기, 이미지 1장을 처리하는 데 필요한 연산량이 모두 증가하는 결과를 초래하였다. 이는 곧 이미지 추론 속도인 FPS(Frame per Second) 성능을 악화시키는 요인으로 작용하게 된다. 그리하여, 실시간으로 작동하는 Deep CNN을 사용할 수 있도록 Deep CNN 연산을 가속화하는 연구가 진행되었고, 에너지 효율이 높은 FPGA를 사용한 다양한 Deep CNN용 하드웨어 가속기 설계 방법들이 많이 제안되었다[9].

FPGA를 사용한 Deep CNN용 하드웨어 가속기는 주로 2가지에 초점을 맞추고 있다. 첫 번째는, Dataflow이다. FPGA 상에서 convolution 연산은 데이터 블록을 가져와 동일한 연산을 반복적으로 수행하는 과정으로 이루어지는데 어떤 data를 가져와서 얼만큼

사용하느냐에 따라 다양한 연산 방법이 존재하고, 이는 곧 convolution 연산의 병렬성에 큰 영향을 끼치게 된다. 또한 off-chip 메모리에 얼만큼 접근하는지에 대해서도 영향을 끼치게 되므로, off-chip 메모리에 접근하는 데에 전력 비용이 가장 많이 든다는 점을 고려하면 시스템 전체 에너지에도 큰 영향을 끼치게 된다. 실제로 SmartShuttle[8]은 최적화 문제를 정의하여 모델 네트워크의 각 layer마다 최적의 data 재사용 방식을 적용하여 off-chip 메모리 접근을 줄이고 최대한 on-chip에서 데이터를 재사용함으로써 off-chip 데이터 전송 크기를 줄였다. 또한 Fused Convolution[1]은 처음으로 single layer가 아닌 across layer dataflow를 제시하였으며, on-chip 버퍼 메모리를 추가로 사용하여 off-chip 메모리 접근을 획기적으로 감소시켰다.

두 번째로는, Quantization이다. Deep CNN은 데이터 형식이 floating point 32bit이 아닌 더 작은 precision의 bit data를 이용하여도 모델의 정확도 성능 손실이 적음을 많은 연구들에서 밝혀 냈다[11-13]. 이는 곧 FPGA 상의 리소스 절감으로 이어지고, 에너지 효율이 높은 하드웨어 가속기 설계로 이어질 수 있다. 실제로 Mixed precision을 제안한 논문[2]에서는 각 layer에 최적화된 bit width를 적용함으로써, 높은 연산 속도와 적은 FPGA 리소스 필요량을 보여주었다.

하지만, 위 방법들은 이미지 분류기 네트워크인 Inception network에 대해서는 잘 적용되지 못한다. Inception Network는 이미지 분류기 네트워크로서, ILSVRC 2012 Test 데이터 셋에 대해서 95% 이상의 정확도 성능을 갖는다[10]. Inception Network의 주된 특징은 Inception 모듈로, 1개의 feature map에 대해 여러 branch로 나뉘어 처리하는 특성을 갖는다. 이러한 branch 구조는 기존의 single layer를 target으로 하는 dataflow를 적용할 수 있어도 최적의 결과를 갖지

못한다. 예를 들면, 여러 branch로 나뉘어 처리하기 때문에 같은 input feature map 데이터를 여러 번 off-chip 메모리에서 load해야 하는 상황이 발생하게 되고 이는 곧 off-chip 메모리 접근을 증가시키는 결과를 초래하게 된다. 그리하여, 높은 성능의 Inception network를 FPGA 상에서 가속시키기 위해 Inception network를 target으로 한 특정 dataflow이 필요하다.

본 논문에서는 해당 문제점을 바탕으로 Inception network의 branch 구조를 고려하여 새로운 dataflow를 제안한다. 새로운 dataflow는 input feature map 데이터를 최대한 재사용하는 Grouped convolution과 각 branch 내의 중간 feature map 데이터를 최대한 재사용하는 Fused convolution을 모두 적용하여 feature map 데이터를 on-chip에서 최대한 재사용한다. 또한, on-chip 버퍼 메모리 크기를 줄이기 위해, weight 파라미터 재사용 방식을 Inception-C 모듈만 다르게 적용하여 처리한다. 이 Mixed convolution을 사용하면, 적은 양의 추가 FPGA 리소스를 사용하여 off-chip 메모리 접근량을 크게 줄일 수 있다.

## 1.2 논문 구성

본 논문의 구성은 다음과 같다. 2장에서는 여러 버전의 Inception network를 바탕으로 Inception network의 큰 특징인 branch 구조에 대해서 살펴본다. 또한, 기존의 Deep CNN FPGA 하드웨어에 적용된 기법들을 소개한다. 3장에서는 2장에서 소개한 기법들이 Inception network에 적용하여 최적의 결과를 갖지 못함을 밝히면서, Input feature map 데이터를 재사용하는 Grouped convolution과 branch 내의 intermediate feature map 데이터를 재사용하는 Fused convolution을 바탕으로 한 Mixed convolution을 소개하며 동작 과정을 상세히 설명한다. 또한 on-chip 버퍼 크기를 최적화하는 dataflow를 적용하여 FPGA 보드의 BRAM 리소스를 감소시키는 방안을 설명한다. 그리고 4장에서는 본 연구에서 제안한 Mixed convolution을 검증하기 위해 설계한 하드웨어 가속기 구조를 보며 설계 과정에 적용한 여러 기법들을 소개한다. 5장에서는 Mixed convolution을 적용하였을 때, off-chip 메모리 접근량 및 데이터 전송 크기를 얼만큼 줄일 수 있는지 보이고, 해당 하드웨어 가속기에서 필요로 하는 on-chip 버퍼 메모리를 비교함과 동시에 FPGA 보드 리소스를 분석함으로써 Mixed convolution의 효능을 보인다. 마지막으로 6장에서는 논문을 정리하면서 결론을 맺는다.

# 제 2 장 관련 연구

## 2.1 Inception Network

이미지 분류기 CNN은 AlexNet[15]을 기반으로 발전하였고, layer를 깊게 쌓아 만든 깊은 망 구조를 취하면서 정확도 성능을 향상시켰다. 이미지 분류기 중 GoogleNet은 Inception 모듈을 처음 사용한 네트워크로써, Inception-V1으로도 불리며 발표된 그 해에 ILSVRC ImageNet 대회에서 1등을 수상하였다. GoogleNet에 적용된 Inception 모듈은 그림 2.1과 같다. 그림과 같이, 같은 feature map에 대하여 총 4가지의 branch로 나뉘어 각각 연산을 수행한다. 이는 기존의 layer를 수직방향으로 쌓아 만든 구조와 대비되는 것으로, 동일한 feature map 혹은 image에 다른 kernel 크기를 적용하여 convolution을 함으로써 다양한 정보를 추출할 수 있었다.

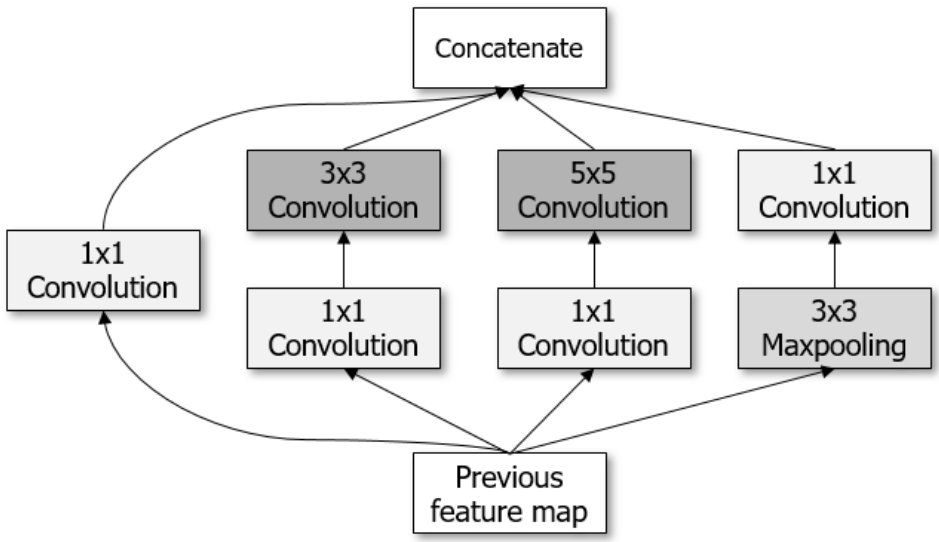



그림 2.1 GoogleNet에 적용된 Inception 모듈 구조

GoogleNet은 이러한 Inception 모듈을 9번 쌓아 만든 구조를 띄며, 총 22개의 layer로 이루어져 있다. 이러한 GoogleNet을 다시 연산량 비용과 weight 파라미터 크기를 고려하여 네트워크를 수정하면서 여러 버전의 Inception network를 발표하였으며, Inception V4는 그 마지막 버전으로 가장 높은 정확도 성능을 갖는 Inception network이다.

Inception V4 네트워크의 구조는 그림 2.2에 나타나 있다. Inception V4 네트워크는 처음에 Stem 모듈을 시작으로 Inception-A, Reduction-A, Inception-B, Reduction-B, Inception-C 모듈을 차례대로 거친다. 또한 Inception-A, Inception-B, Inception-C 모듈을 각각 4번, 7번, 3번을 반복적으로 거치는 구조를 띄고 있고 이에 따라 네트워크의 크기 또한 여타 Deep CNN과 같이 큰 크기를 보인다. 하지만, 각 모듈에 1x1 convolution layer를 두어 다양한 kernel 크기로 convolution을 진행하기 앞서 feature map의 channel 크기를 감소시키므로 weight 파라미터 크기는 상대적으로 낮은 모습을 보이고 있는데, ImageNet 데이터 셋에 대하여 Top 1 정확도가 80% 이상인 네트워크 중에서는 거의 가장 작은 크기의 weight 파라미터 크기를 갖는다[16].

Inception V4 네트워크의 각 Inception 모듈은 GoogleNet에 적용된 Inception 모듈을 변형시켜 사용하였다. Inception V4 네트워크의 Inception 모듈은 1x7, 7x1, 1x3, 3x1 등의 1D kernel 크기를 사용하였으며, 더 복잡한 branch 구조를 띄고 있기도 한다. 하지만, 본질적으로는 branch 구조를 띄고 있어 같은 input feature map 데이터에 대하여 다양한 branch로 분기되어 계산되며 이후에 concatenate를 통해 추출한 feature를 합친 후 다음 layer로 넘겨주게 된다.

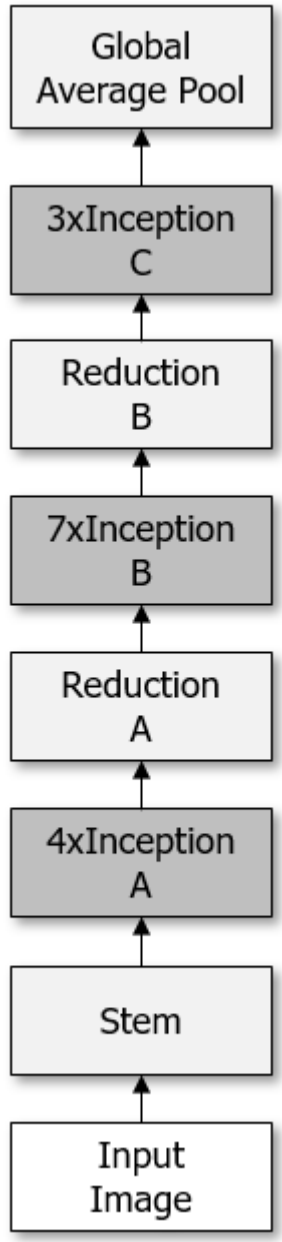



그림 2.2 Inception V4 network 구조[5]

이러한 Inception network를 사용하는 추론용 하드웨어 가속기를 설계하는 방법 중 각 branch를 혹은 layer를 이루는 kernel 크기 별로 서로 다른 연산기를 두어 parallel한 연산을 할 수 있도록 하는 방법이 제안되었다[17]. 하지만 이는, 각 연산 장치끼리의 복잡한 data control, 큰 리소스 요구량과 같은 한계가 있다. Layer를 1개씩 처리하여 나가는 Single layer 가속기는 연산의 병렬성이 떨어질 수 있지만 하드웨어 설계가 단순하다는 장점이 있어 본 논문에서는 해당 single layer 가속기 형태를 이용한다. 또한, 아직까지 Inception network의 branch 구조를 target으로 한 하드웨어 설계가 충분히 검토되지 않았기 때문에 본 논문에서는 이를 고려한 데이터 재사용 방법을 제안하고자 한다.

## 2.2 FPGA 가속기 연구

Deep CNN은 많은 수의 layer로 구성된 깊은 망 구조로 인해, 중간 feature map 데이터의 크기, weight 파라미터 크기, 1개의 이미지 데이터를 추론할 때 소요되는 시간이 모두 증가하게 되었다. 그런데 Deep CNN을 구성하는 convolution layer는 일정 데이터 블록에 대해서 반복적인 연산을 수행하기 때문에 데이터 연산의 병렬화가 진행될 수 있고, 이를 이용하여 Deep CNN을 빠르게 추론할 수 있는 Deep CNN 하드웨어 가속기 연구가 활발히 진행되었다. 그 중에서도 에너지 효율이 높다고 알려진 FPGA를 활용하여 하드웨어 가속기를 설계하는 것이 큰 관심을 불러일으키고 있다.

FPGA를 활용한 Deep CNN 하드웨어 가속기 연구의 가장 큰 주제는 off-chip memory 접근을 줄이는 것이다. Off-chip 메모리에 빈번하게 접근하게 되면, 전체 추론 속도가 감소하게 되고 메모리 접근에 드는 전력 소모 또한 커지게 된다. 그리하여, off-chip 메모리 접근을 줄이는 새로운 dataflow를 제안하거나 on-chip 버퍼 메모리를 추가하여 관리하는 방법을 제안한 다양한 연구가 진행되었다.

### 2.2.1 Off-chip 메모리 접근 관련 연구

SmartShuttle[8]은 Deep CNN 하드웨어 가속기에서 전력 소모가 가장 큰 부분이 DRAM 접근임을 보이고, 이를 줄이기 위한 방법으로 flexible한 dataflow를 제안하였다. 이는 CNN 모델을 이루는 각 layer마다 최적의 데이터 재사용 방법과 Tiling 크기가 다르다는 점에

기반한 것으로, layer마다 최대로 재사용하는 데이터 종류와 데이터 블록의 크기(Tiling factor)를 다르게 하여 off-chip 메모리 접근을 최소화할 수 있음을 보여주었다. SmartShuttle도 마찬가지로 1개의 layer씩 순차적으로 처리해 나가는 single layer processing 가속기 구조를 가정하고 있으나, AlexNet, VGG16 네트워크와 같이 수직방향으로 적층된 네트워크에 대해서만 검증하였으며 Inception 네트워크와 같은 branch 구조를 갖는 네트워크에 대해서는 적용시키지 않았다.

Fused Convolution[1]은 기존의 single layer 가속기 구조에서 고려되지 않았던 across-layer dataflow를 처음으로 제시하며 off-chip 메모리 접근 및 데이터 전송 크기를 획기적으로 감소시켰다. 그림 2.3은 across-layer의 dataflow를 보여주고 있다. Input feature map의 일부를 이용하여 연산을 진행한 후 생성된 중간 feature map 데이터를 on-chip에 저장하여 놓고, 이를 재사용하여 최종 output feature map 데이터를 만들어 off-chip 메모리에 저장하게 된다. 즉, Input feature map을 off-chip 메모리에 접근하여 load한 후에 convolution 연산 과정에서 생성되는 중간 feature map을 다시 off-chip 메모리에 접근하여 저장하지 않고 on-chip 버퍼 메모리에서 저장하고 이를 on-chip에서 재사용하는 방법이다. Fused convolution은 이와 같은 dataflow로 off-chip 메모리의 데이터 전송 크기를 줄였으나, 중간 feature map data를 on-chip 버퍼 메모리에 저장해야 하므로 BRAM 리소스 필요량이 매우 크다는 단점이 있다.

Mixed Dataflow를 제안한 논문[2]에서는 Deep CNN을 2가지 그룹으로 나누어 각각의 그룹에 알맞은 weight 파라미터 재사용 dataflow를 제안하였다. 그 dataflow는 각각 row-based weight 재사용 dataflow와 full weight 재사용 dataflow이다.

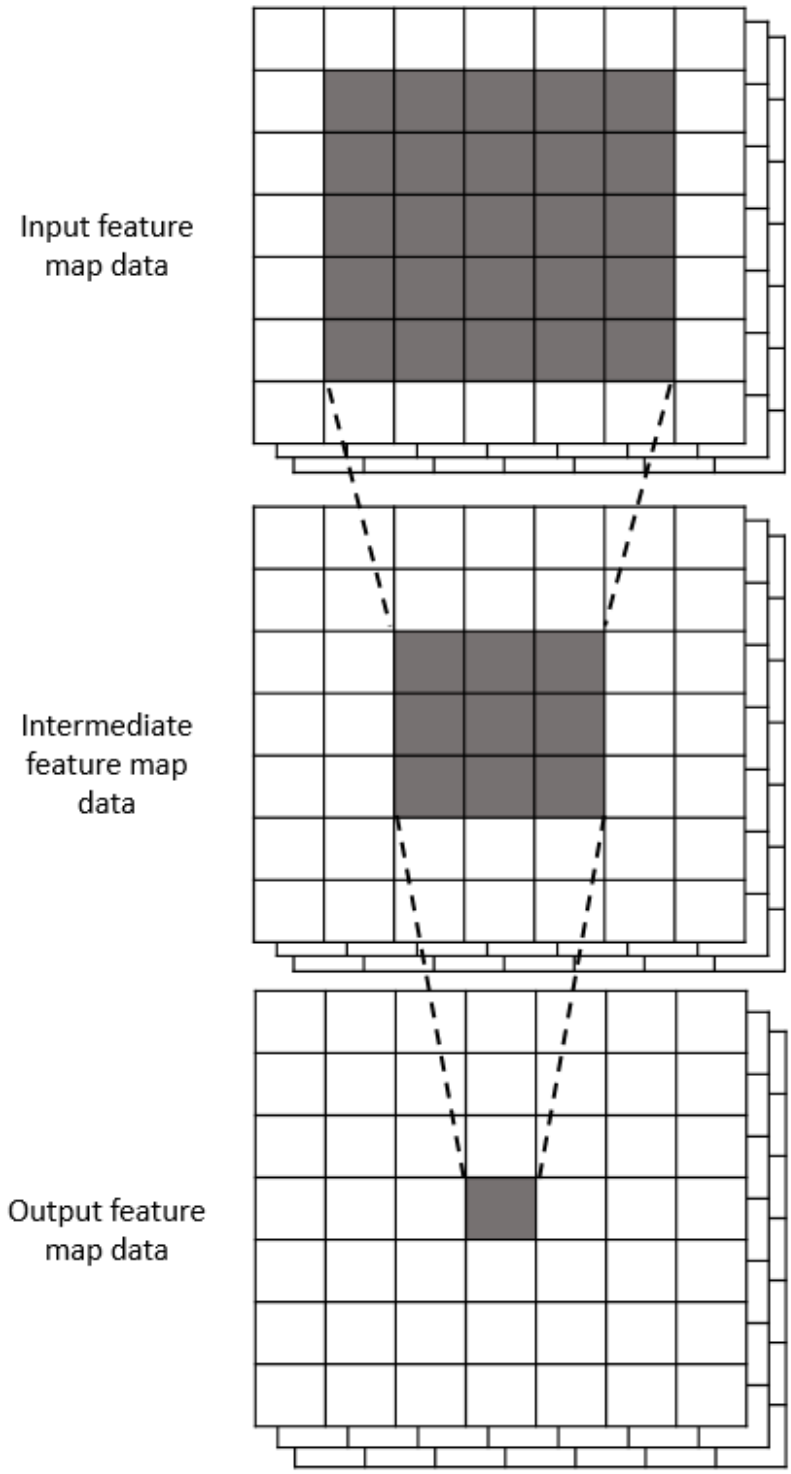



그림 2.3 Fused Convolution의 dataflow[1]

Row-based weight 재사용 방법은 weight 파라미터 데이터 블록(Tile block)을 한 번 load하여 Input feature map 데이터의 한 row에 대해서만 재사용하는 dataflow로, partial sum 데이터를 일시적으로 저장하기 위한 on-chip 버퍼 메모리의 크기가 상대적으로 작다. 하지만, 한 번 불러온 weight 데이터 블록을 row에 대해서만 재사용하기 때문에, 다음 번 height에 해당하는 input feature map 데이터에 convolution 연산을 하기 위해서는 같은 weight 데이터 블록을 다시 off-chip 메모리에 접근하여 불러와야 한다는 단점이 있다. 그에 반면, full weight 재사용 방법은 한 번 불러온 weight block 데이터에 대해서 최대로 재사용한다. 즉, 한 번 off-chip 메모리에 접근하여 weight 데이터 블록을 load하면 그 이후에는 동일한 weight 데이터 블록을 다시 접근하여 load하는 경우가 없다. 하지만, full weight 데이터 재사용 dataflow는 partial sum 데이터를 일시적으로 저장할 on-chip 버퍼 메모리의 크기가 상대적으로 크고, row-based weight재사용 dataflow와 달리 input feature map data 블록을 여러 번 load해야 하는 단점이 있다.

위와 같은 특성으로 인해, row-based weight 재사용 dataflow[14]는 output feature map 크기가 상대적으로 크고 weight 파라미터 데이터 크기가 상대적으로 작은 Deep CNN 네트워크의 초반부에 사용된다. 하지만, weight 파라미터 데이터 크기가 상대적으로 커지는 Deep CNN 네트워크의 후반부에서는 weight 파라미터 데이터를 다시 load하는 비용이 커지기 때문에 full weight 재사용 dataflow가 row-based weight 재사용 dataflow에 비해 더 적합하다. 또한, off-chip 메모리 접근 및 전송 크기를 더 줄이기 위해 row-based weight 재사용 dataflow를 적용시킬 때는 weight 파라미터 데이터를 on-chip에 모두 저장하고, full weight 재사용 dataflow를 적용시킬 때는

feature map 데이터를 on-chip에 모두 저장하는 방법이 이용되기도 한다[2].

Inception V4 네트워크 또한 일반적인 Deep CNN과 동일하게 후반부로 갈수록 feature map 데이터 크기는 감소하고, weight 파라미터 데이터 크기가 증가한다. 그림 2.4는 이를 보여주고 있다. 이전 연구 결과에 기반하여 본 논문에서도 row-based weight 재사용 dataflow를 Inception V4 네트워크에 적용한다. 하지만, row-based weight 재사용 dataflow만 적용한다면, 이전 연구들과 같이 최적의 off-chip 메모리 접근량을 얻을 수 없다. 이는 마찬가지로, Inception network의 branch구조를 고려하지 않았기 때문이다.

그리하여 3장에서는 기존의 연구들을 바탕으로 Inception network의 branch 구조에 알맞은 데이터 재사용 방법을 제안하여 off-chip 메모리 접근량 및 데이터 전송 크기를 더 줄일 수 있음을 보인다.

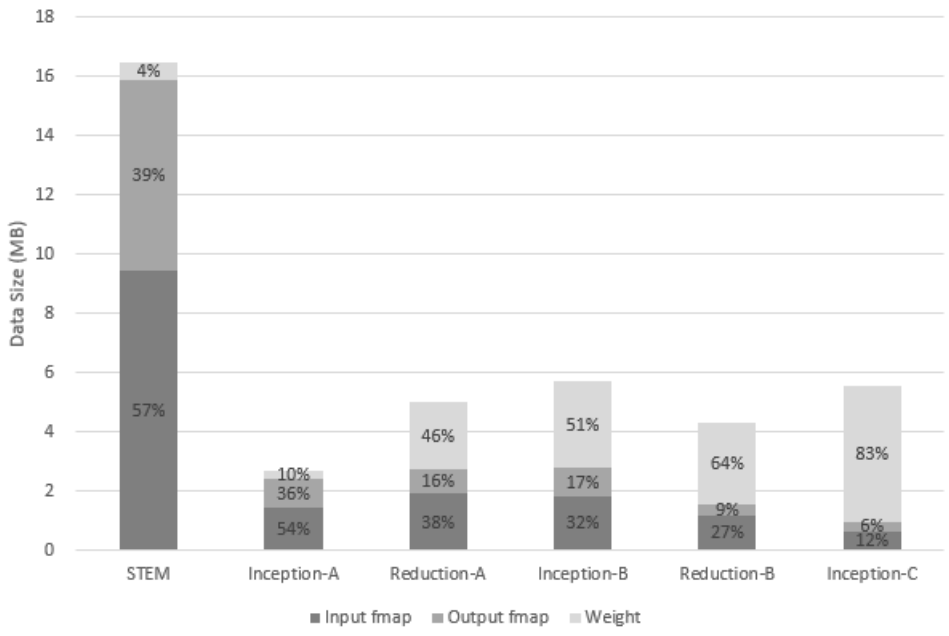



그림 2.4 Inception V4 network의 모듈별 데이터 크기

# 제 3 장 Inception V4 네트워크의 데이터 재사용 방법

## 3.1 Grouped Convolution

본 장에서는 Inception 모듈의 branch 구조를 기반으로 데이터를 on chip에서 재사용하여 off chip 메모리 접근 및 데이터 전송 크기를 줄이는 방법을 제안한다. 먼저 Grouped convolution을 소개한다. Inception V4 네트워크의 Inception-A, Inception-B, Inception-C 모듈은 그림 3.1과 같다. 각 모듈 모두 4개 이상의 branch로 구성되며, 각 branch는 같은 input feature map에 대해 서로 다른 filter를 이용하여 다른 feature를 추출한다. 여기서 주목할 점은 서로 다른 branch가 같은 input feature map을 공유한다는 것이다. 기존의 single layer processing 가속기는 이 점을 고려하지 않고, 1개의 layer씩 순서대로 처리한다. Inception 모듈의 각 layer를 어떤 순서로 처리하는지는 가속기의 종류에 따라 다양한 조합이 나올 수 있지만, 1개의 layer씩 순서대로 처리하는 방법으로 인해 같은 input feature map을 on chip으로 load하기 위해 off chip 메모리에 여러 번 접근하게 된다. Grouped convolution은 각 Inception 모듈의 input feature map을 최대한 재사용하여 off chip memory에 여러 번 접근하는 문제를 해결하는 convolution 방법이다.

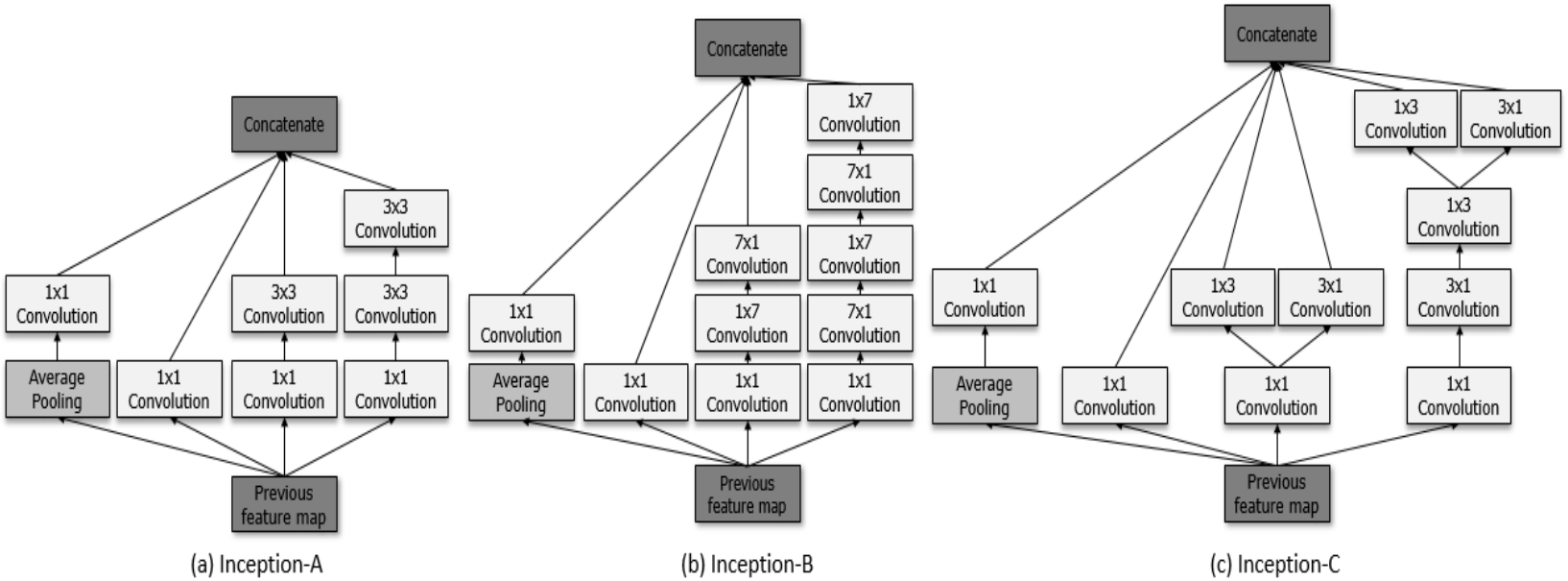


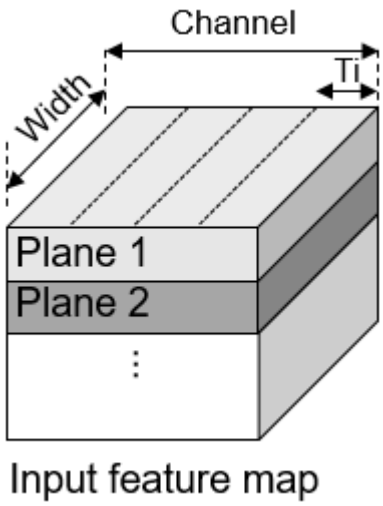

그림 3.1 Inception V4 Network의 Inception 모듈

### 3.1.1 Grouped Convolution의 동작 방식

그림 3.2는 기존 single layer processing 가속기가 Inception-A 모듈에서 앞 쪽에 위치한 3개의 1x1 convolution layer를 처리하는 예를 시간에 따라 간략하게 보여준다. 이 예시는 Baseline 하드웨어 가속기의 row based weight 재사용 dataflow를 따르며 3개의 1x1 convolution layer를 branch에 따라 1st layer, 2nd layer, 3rd layer로 구분하여 부른다. 먼저, row-based weight 재사용 dataflow에 따라 convolution 연산에 필요한 input feature map 데이터의 일부인 Plane 1, 1st layer의 weight 데이터를 각각 on chip 버퍼인 Input buffer와 Weight buffer에 저장한다. 그 후, on chip 버퍼에 저장된 데이터를 가져와 연산기에 넣어 convolution을 진행하고, 그와 동시에 다음 연산에 필요한 input 데이터인 Plane 2을 on chip 버퍼에 load한다. Convolution 연산 결과, input feature map인 Plane 1과 동일한 크기의 output feature map block이 생성되고 이를 off chip 메모리에 저장한다(그림 3.2의 Save 1). Off-chip 메모리에 데이터를 모두 저장한 후에는 미리 load한 Plane 2와 weight 데이터를 연산기에 넣어 convolution을 진행한다. 위 과정을 반복하여 1st layer를 전부 처리하면, 동일한 과정을 2nd layer와 3rd layer에 대해서 차례대로 동일하게 진행한다.

위 과정은 그림 3.2에서 볼 수 있듯이, 동일한 input feature map 데이터를 load하는 것에 대한 overhead가 존재한다. 기존의 single layer processing 가속기에서는 3개의 1x1 convolution layer를 처리하기 위해서 같은 input feature map 데이터를 총 3번 load해서 연산을 진행하게 된다. 이는 곧 off-chip 메모리 접근량 및 데이터 전송

크기 증가를 의미한다. 즉, Inception 모듈의 branch 구조는 기존의 single layer processing 가속기의 처리 방식에 잘 맞지 않음을 의미한다.



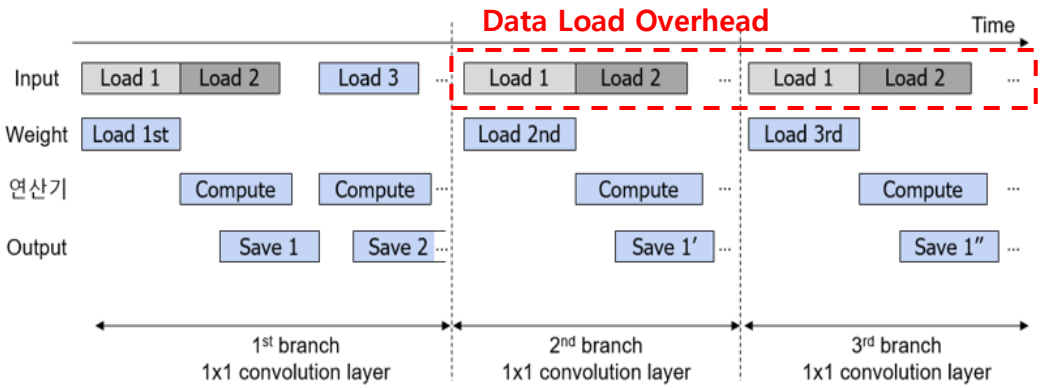




그림 3.2 기존 Single layer processing 가속기의 layer 처리 시간별 분석

반면, Grouped convolution은 3개의 1x1 convolution layer들을 묶어 1개의 layer로 취급한다. 즉, 3개의 1x1 covolution layer를, 더 큰 크기의 output channel dimension을 갖는 1x1 filter를 사용하는 layer로 바꾸어 처리하게 된다. 그 결과, input feature map 데이터를 load하는 횟수는 1번으로 감소하며, off chip 메모리 접근 횟수 및 데이터 전송 크기를 줄일 수 있다.

그림 3.3은 single layer processing 가속기에서 해당 Grouped convolution을 처리하는 연산 과정을 시간 축으로 나타낸 것이다. 기존의 row based weight 재사용 dataflow에서 동일한 input feature map을 2번째 branch와 3번째 branch의 1x1 convolution layer를 위해 다시 load하는 것과 달리, Grouped convolution에서는 input feature map 데이터를 한 번만 load한다. 또한, 기존의 dataflow에서는 weight data를 각 layer가 진행될 때 load했던 것과 달리, Grouped convolution의 경우 처음 시작할 때 3개의 layer 처리에 필요한 weight 파라미터 데이터를 모두 load한다. 이는 앞서 언급하였듯이, row-based weight 재사용 dataflow에서는 weight 데이터를 여러 번 load하는 것을 막기 위해 convolution 연산을 하기 전에 layer를 처리하기 위해 필요한 weight 데이터를 on-chip에 load한다.

Row based weight 재사용 dataflow를 적용한 Grouped convolution의 자세한 과정은 그림 3.4, 3.5, 3.6, 3.7에 나타나 있다. 먼저 Input feature map의 Plane 1에 해당하는 data를 on-chip 버퍼인 input line 버퍼 메모리에 저장한다. 그와 동시에, 3개의 1x1 convolution layer에 해당하는 weight 파라미터 데이터를 on-chip weight 버퍼 메모리에 저장한다. 그 후, on-chip 버퍼 메모리에 저장된 Plane 1 data와 첫 번째 layer의 weight block을 가져와 convolution을 진행하여 첫 번째 layer의 output feature map에 해당하는 데이터를

생성한다. 이 과정이 그림 3.4에 나타난 과정이다. 그 결과 output feature map의 To channel에 해당하는 블록 데이터가 생성되고, 다음 To에 해당하는 weight 파라미터 데이터 블록을 이용하여 다음 To에 output feature map 데이터를 만들어낸다.

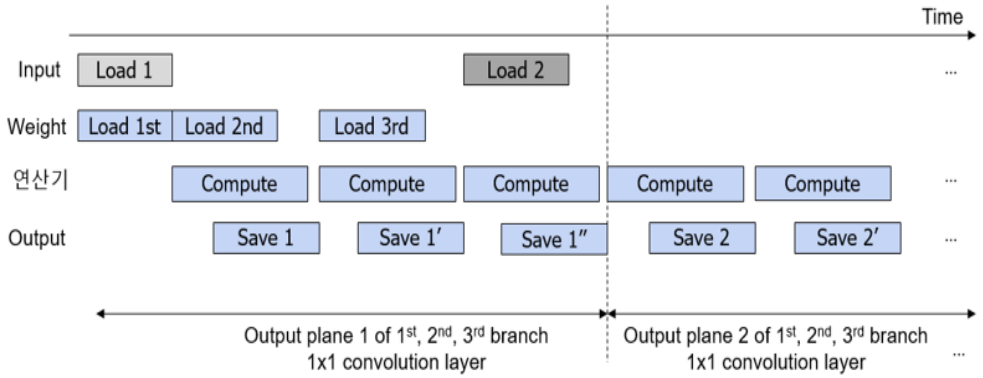



그림 3.3 Grouped convolution layer 처리 시간 별 분석

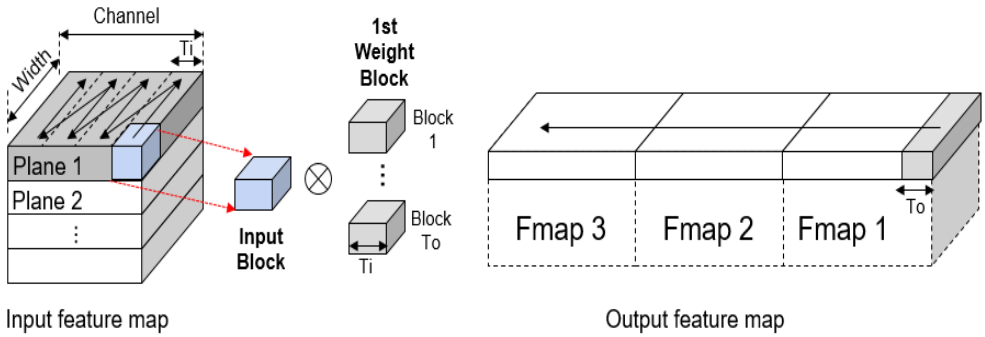



그림 3.4 Grouped convolution의 첫 번째 layer 처리 과정

첫 번째 layer의 output feature map(Fmap 1)이 모든 channel에 대해 데이터가 만들어지면, 두 번째 layer의 weight 데이터 블록을 동일하게 가져와 동일한 연산을 진행한다. 그 과정은 그림 3.5에 나타나 있다. 두 번째 layer에 대해서도 Plane 1에 대하여 convolution 연산을 진행하게 되고 To channel 블록만큼 데이터가 생성된다. 이 과정에서 Input feature map 데이터는 이미 Input 버퍼에 있으므로 데이터를 load하기 위해 off-chip 메모리에 접근하지 않는다.

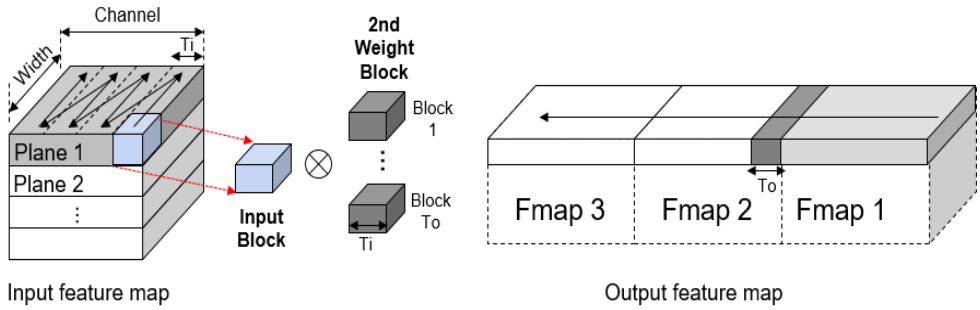



그림 3.5 Grouped convolution의 두 번째 layer 처리 과정

두 번째 layer의 output feature map(Fmap 2)이 모든 channel에 대해 데이터가 만들어지면, 마지막 layer의 weight 데이터 블록을 동일하게 가져와 동일한 연산을 진행한다. 그림 3.6은 그 과정을 보여준다. Grouped convolution으로 마지막 layer를 처리할 때, convolution 연산을 처리함과 동시에 다음 input feature map 데이터를 load하는 시간을 hiding하기 위하여 미리 input feature map 데이터를 on-chip으로 load한다.

이와 같이 Plane 1에 대하여 3개의 layer의 output feature map이 모두 만들어지면, 다시 미리 load한 input feature map의 Plane 2에 해당하는 데이터를 이용하여 위 과정을 반복(그림 3.7)하면서 output

feature map 데이터를 생성한다.

Grouped convolution은 Inception 모듈의 input feature map 데이터를 최대로 재사용하는 방법으로 on-chip 버퍼를 추가적으로 사용하여 off-chip 메모리 접근 및 데이터 전송 크기를 감소시킬 수 있다. 그 데이터 전송 크기 감소량은 input feature map의 크기에 비례하여 커진다.

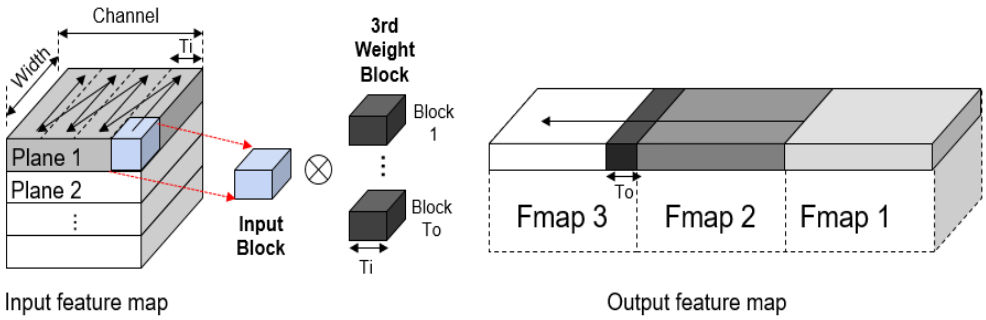



그림 3.6 Grouped convolution의 세 번째 layer 처리 과정

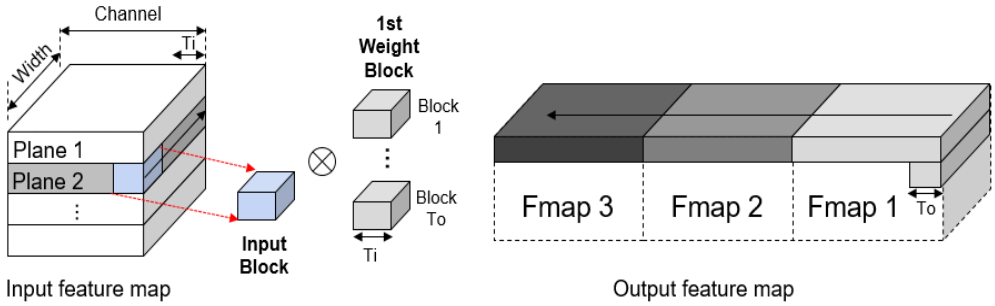



그림 3.7 Plane 2에 대한 Grouped convolution 반복 과정

### 3.1.2 Grouped convolution의 on chip 버퍼 크기

Grouped convolution은 off chip 메모리 접근 및 데이터 전송 크기를 감소시키는 대신에 on chip 버퍼 메모리 크기를 증가시킬 수 있다. 그 이유는 row-based weight 재사용 dataflow에서 1개의 layer를 처리할 때 필요한 weight 파라미터 데이터를 on-chip 버퍼 메모리에 모두 저장하기 때문이다. 그리하여, Grouped convolution은 1x1 convolution layer 3개를 묶어 1개의 layer로 처리하기 때문에 한 layer의 weight의 크기가 커지게 되며, 이는 weight 버퍼 메모리 크기를 증가시킬 수 있다.

실제로 Inception V4 네트워크에서 가장 큰 weight 파라미터 크기를 갖는 layer는 Reduction-A 모듈에 있는 3x3 convolution layer로서, 8bit fixed point 데이터 형식 기준 1.3MB 크기를 갖는다. 하지만 Grouped convolution을 적용한다면, Inception-A, Inception-B 모듈에서 Grouped convolution으로 처리되는 layer의 weight 파라미터 크기는 1.3MB보다 작지만, Inception-C 모듈의 Grouped convolution layer는 1.3MB보다 더 커지게 되며 그로 인해 필요한 weight 버퍼 크기를 증가시킨다.

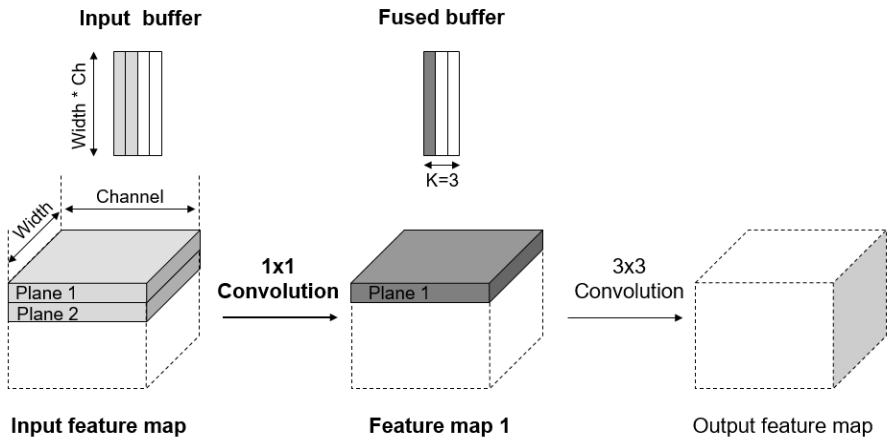
## 3.2 Fused Convolution

Fused convolution은 앞서 언급했던 것과 같이, 여러 convolution layer를 cascade하여 처리하는 방법으로, 중간 feature map을 off chip 메모리에 저장하지 않는다[1]. Pyramid 모양의 multi layer sliding window를 만들어 convolution을 처리하며 중간 feature map을 on chip 버퍼에 저장하여 off chip 메모리 접근 및 데이터 전송 크기를 줄인다. 이와 같은 방법을 inception 모듈에 적용한다면, 각 branch를 fuse하여 처리함으로써 중간 feature map 데이터를 최대한 재사용할 수 있다.

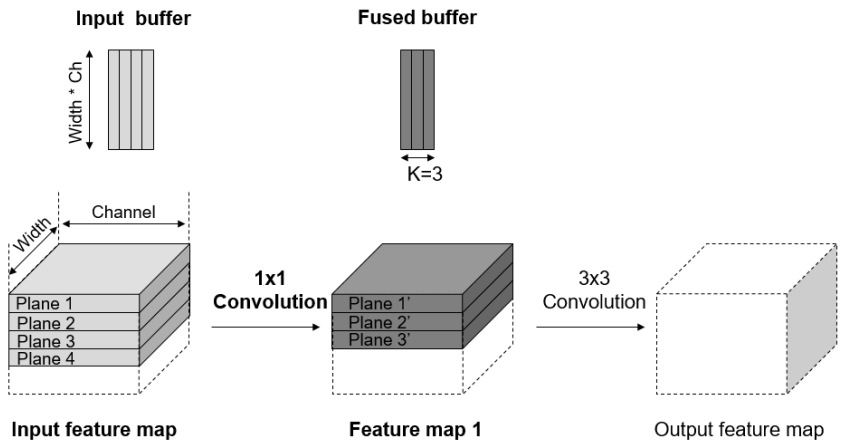
Fused convolution은 2가지 설계가 가능하다. 한 가지는 Recomputing이고, 다른 한 가지는 Storing이다. Storing은 on chip 버퍼를 충분히 두어 필요한 중간 feature map 및 input feature map 데이터가 on chip 버퍼에 모두 저장될 수 있게 하는 설계이다. 반대로 Recomputing은 on chip 버퍼를 최소한으로 사용하여 중간 feature map을 on chip 버퍼에 저장할 수 있도록 하지만, 한정된 버퍼의 크기로 인해 이미 convolution을 하여 얻은 데이터가 사라질 경우 다시 convolution하는 설계이다. 2 가지 설계 방법의 trade off가 존재하지만 본 연구에서는 Recomputing으로 인한 지연시간 비용이 없도록 하기 위해 on chip 버퍼를 더 사용하는 Storing 설계 방법을 이용한다. 또한, 연속된 layer의 2D feature map pyramid를 사용한 원래의 방법과는 다르게 row based weight 재사용 dataflow에 맞게 변형하여 중간 feature map을 fused line buffer에 저장한다.

### 3.2.1 Fused Convolution의 동작 방식

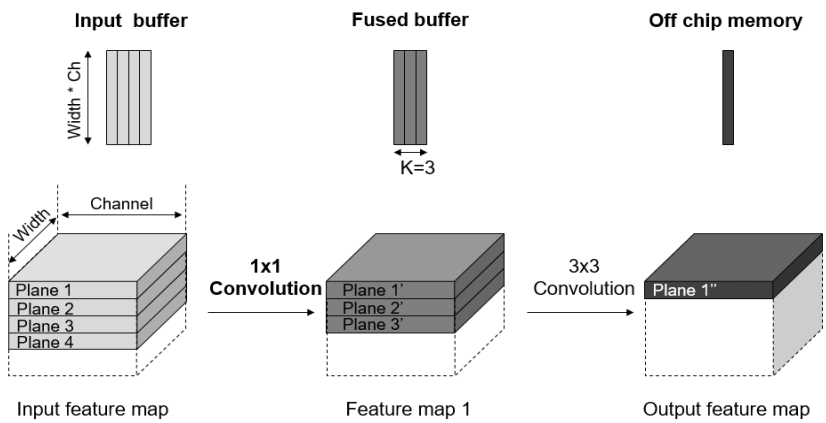
그림 3.8는 Fused convolution의 연산 과정의 한 예시를 보여준다. 예시는 Inception-A 모듈의 왼쪽에서 3번째 branch에 Fused convolution를 적용한 것으로 1개의 1x1 convolution layer와 1개의 3x3 convolution layer, 총 2개의 layer를 fuse 하여 처리한다. 먼저, input feature map의 Plane 1을 off-chip 메모리로부터 on-chip 버퍼인 input buffer로 load한다. 그 후 input Plane 1을 이용한 1x1 convolution 연산을 통해 feature map 1의 Plane 1’를 생성하고 이를 fused buffer에 저장한다. Convolution 연산을 하는 과정에서 input feature map의 Plane 2를 input buffer에 load한다(그림 3.8 a). 이 과정을 반복하여 Feature map 1의 Plane 3’까지 fused buffer에 저장한다(그림 3.8 b). Feature map 1의 첫 3개의 Plane 데이터가 fused buffer에 저장되었기 때문에 1x1 convolution layer의 연산을 멈추고 3x3 convolution layer의 연산을 시작하여 output feature map의 Plane 1” 데이터를 생성하고 이를 off chip 메모리에 저장한다(그림 3.8 c). 더 이상 convolution을 진행할 feature map 1 데이터가 없으면 다시 1x1 convolution layer로 돌아가 연산을 시작한다. 이와 같은 형식을 반복적으로 진행하면서 intermediate layer의 데이터를 fused buffer에 저장하여 off-chip 메모리에 접근하지 않고, 그 결과 off-chip 메모리 데이터 전송 크기를 많이 줄일 수 있다.




(a)




(b)




(c)

그림 3.8 Fused convolution의 연산 과정

### 3.2.2 Fused Convolution의 fused 버퍼 크기

Fused convolution으로 각 branch를 처리하게 되면 각 branch의 중간 feature map을 off chip 메모리에 저장하지 않는다. 이는 곧 off chip 메모리 접근 및 데이터 전송 크기를 감소함을 의미한다.

하지만 fused 버퍼와 같은 추가적인 on chip 버퍼 메모리가 필요하고 Storing 방법을 사용하기 때문에 fuse 되는 layer수에 따라 그 크기는 증가한다. Fused 버퍼로 인한 추가 on chip 버퍼의 크기는 식 (3.1)로 구할 수 있다.

$$\text{Extra fused buffer} = \sum_{l=1}^{k-1} K_C^l * (W^l * Ch^l) \qquad (3.1)$$

여기서 k는 fuse 하는 layer의 수이고, $K_c$는 해당 layer filter의 column 크기이다. W는 해당 layer의 input feature map의 가로 길이이며, Ch는 해당 layer input feature map의 channel 크기이다. 식 (3.1)를 이용하면 각 Inception 모듈의 branch 별로 필요한 fused 버퍼의 크기를 계산할 수 있다. 하드웨어 가속기에서는 각 branch를 모두 support할 수 있어야 하므로, 구해진 branch별 fused 버퍼 크기 중 가장 큰 값을 갖게 된다.

## 3.3 Mixed Convolution

Grouped convolution은 Inception 모듈의 input feature map 데이터 재사용을 극대화하는 방법이고, 각 branch 내의 중간 feature map 데이터 재사용을 고려하지 않는다. 반대로, Fused convolution은 각 branch 내의 중간 feature map 데이터 재사용을 극대화하는 방법이지만, 입력 feature map 데이터의 재사용을 고려하지 않는다. 이 절에서는 Grouped convolution과 Fused convolution의 장점을 모두 이용하는 Mixed convolution의 방법을 제안한다.

### 3.3.1 Mixed Convolution의 동작 방식

Mixed convolution은 Grouped convolution을 이용하여 입력 feature map 데이터를 최대로 재사용함과 동시에 Fused convolution을 이용하여 각 branch 내의 중간 feature map 데이터를 최대로 재사용한다.

그림 3.9는 Inception V4 네트워크의 Inception 모듈에 Mixed convolution 방법을 적용한 것을 보여준다. 먼저 Inception-A 모듈은 각 branch에 존재하는 1x1 convolution layer 3개를 묶어 Grouped convolution으로 처리한다. 연산 과정은 앞 장에서 설명한 과정과 동일하다. 하지만 모든 output feature map을 off chip 메모리에 저장한 것과 달리 Mixed convolution에서는 output feature map을 각각 다른 곳에 저장한다. Branch 내 추가 convolution layer가 없는 $1^{st}$ layer의 경우 생성된 output feature map을 off chip 메모리에 저장하지만, 추가 convolution layer가 존재하는 $2^{nd}$, $3^{rd}$ layer의 경우 on chip 버퍼에

저장한다. 이는 off chip 메모리 접근 및 데이터 전송 크기를 줄이기 위함으로, output feature map이 on chip 버퍼에 저장할 수 있을 정도로 충분히 작다는 점에 착안하였다. Grouped convolution이 끝나면 on chip 버퍼에 저장된 feature map 데이터를 이용하여 각 branch내의 남은 layer 처리를 진행한다. Branch 내의 남은 layer 개수에 따라 Single layer convolution과 Fused convolution을 선택하여 진행한다. 또한, average pooling과 1x1 convolution이 연속으로 존재하는 branch는 fused 방식으로 처리를 함으로써 중간 feature map을 재사용한다.

Inception-B 모듈의 경우, 1x1 convolution의 Grouped convolution이 끝나면 2개의 convolution layer와 4개의 convolution layer가 남게 된다. 이때, 4개의 convolution layer를 한 번에 Fused convolution으로 처리하게 되면 fused 버퍼가 많이 필요하게 된다. 하지만 Grouped convolution에서 사용한 on chip 버퍼를 재사용하여 중간 feature map을 저장하게 된다면, fused 버퍼의 크기를 줄일 수 있다. 그리하여 4개의 layer를 한 번에 Fused convolution으로 처리하는 것이 아닌, 2개의 layer로 나누어 2번 Fused convolution으로 처리하며 첫 번째 Fused convolution의 output feature map을 on chip 버퍼(Buffer 0)에 저장하는 방법을 이용한다. 그렇게 하기 위해서, 3번째 branch의 1x7 convolution layer와 7x1 convolution layer를 먼저 처리하여 Buffer 0의 공간을 만들어 준다.

Inception-C 모듈의 경우 branch에서 다시 branch로 나뉘는 구조를 띄는데 이와 같은 경우 일반적인 convolution으로 처리하며, Fused convolution은 4번째 branch에서만 적용된다. 그리고 나머지 layer는 다른 Inception 모듈과 동일한 처리 방식을 갖는다.

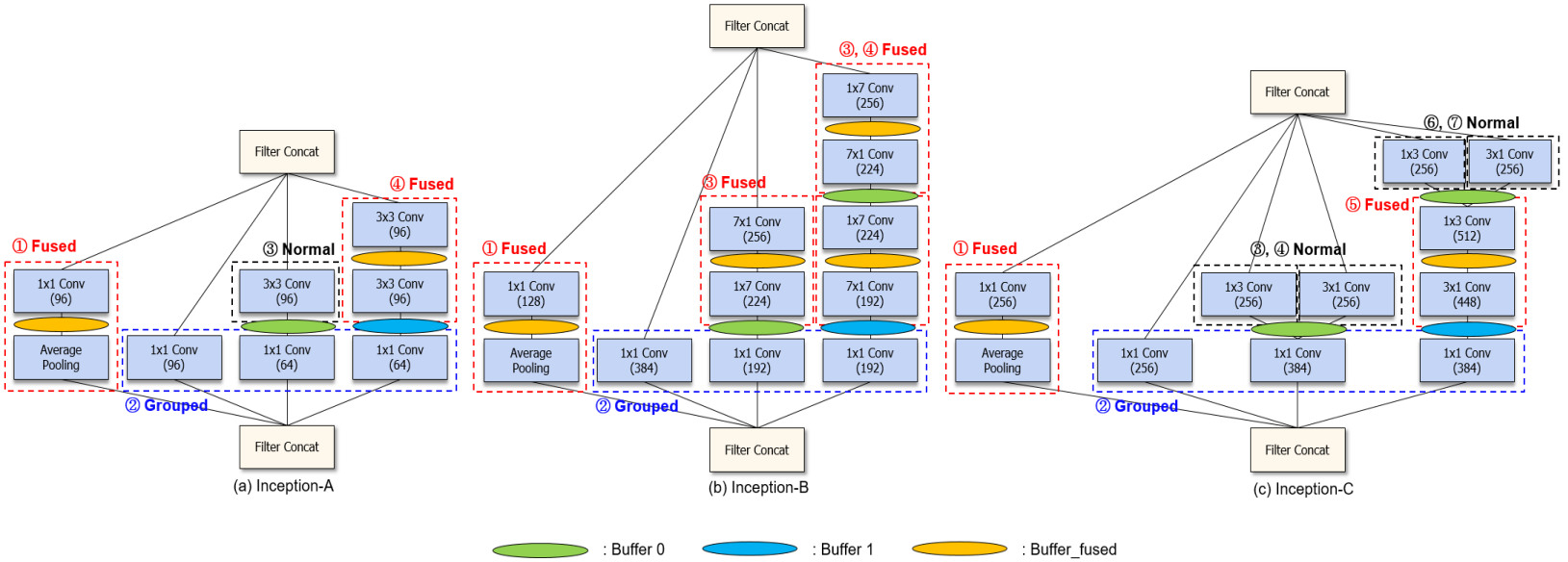



그림 3.9 Inception 모듈 별 Mixed convolution 적용

### 3.3.2 Mixed Convolution의 추가 on chip 버퍼 크기

Mixed convolution 방법의 on chip 버퍼의 크기를 결정하는 요인은 총 4가지다. Grouped convolution으로 인한 증가한 weight buffer, Grouped convolution의 output feature map과 Fused convolution의 output feature map을 저장할 buffer 0, buffer1, Fused convolution의 중간 feature map을 저장할 fused buffer가 그 4가지다. 이를 표현하면 식 (3.2)로 표현할 수 있다.

$$\text{Extra on chip buffer} = Buff_w + Buff_0 + Buff_1 + Buff_{fused} \quad (3.2)$$

Buffer 0, buffer 1, fused buffer를 구하는 식은 식 (3.3), (3.4), (3.5)을 통해 구할 수 있다. N은 Grouped convolution으로 처리하는 2nd layer의 집합이며, M은 Grouped convolution으로 처리하는 3rd layer의 집합이다. K는 Fused convolution으로 처리되는 layer 중 2번째로 처리되는 layer의 집합이다.

$$Buff_0 = \max_{l \in N} (output\_fmap_l) \quad (3.3)$$

$$Buff_1 = \max_{l \in M} (output\_fmap_l) \quad (3.4)$$

$$Buff_{fused} = \max_{l \in K} (K_l^c * W_l * Ch_l) \quad (3.5)$$

식 (3.5)에서 $K^c$ 는 해당 layer의 column filter size이고, W와 Ch는 해당 layer의 입력 feature map의 width와 channel 크기이다. 위 식들을 이용하여 Mixed convolution의 추가 on chip 버퍼 크기를 구하면 421KB 추가 버퍼가 필요함을 알 수 있다. 또한 비교 결과 4가지 종류의 버퍼 중에서 Grouped convolution으로 인한 weight buffer 증가가 전체 증가의 57.6%(242.5 KB)를 차지함을 알 수 있다.

## 3.4 Full weight 재사용을 적용한 Mixed Convolution

Mixed Convolution을 적용하면 Inception 모듈에서 생성되는 feature map 데이터에 대해서 off-chip 메모리 데이터 전송 크기를 많이 줄일 수 있다. 하지만 on-chip 버퍼 메모리가 추가로 필요하고 그 중에서도 242.5 KB 정도의 weight 버퍼가 추가로 필요하다. 이는 Inception-C 모듈에 Grouped convolution을 적용할 때 필요한 것으로 full weight 재사용 dataflow를 이용하여 줄일 수 있다.

Full weight 재사용 dataflow는 row-based weight 재사용 dataflow에 비해 더 작은 weight 버퍼 크기를 가질 수 있다. 하지만 row-based weight 재사용 dataflow에 비해 input, output 버퍼가 더 많이 필요하여 두 dataflow 사이에는 trade off가 존재한다. 그래서 상대적으로 weight 크기보다 feature map size가 큰 network 초기 layer에서는 row weight 재사용 dataflow를 적용하고, weight 크기가 feature map size보다 큰 network 후반부 layer에서는 full weight 재사용 dataflow를 사용하는 방법이 제안되었다[2].

Inception-C 모듈에서 full weight 재사용 dataflow를 사용한다면, Grouped convolution을 적용할 때 추가로 필요했던 weight 버퍼 크기를 줄일 수 있다. 하지만 기존에 적용하였던 Grouped convolution과 Fused convolution을 적용하는 것이 아닌 1개의 layer씩 처리하는 형태로 변하게 된다. 또한, 각 layer의 output은 off-chip 메모리로의 데이터 전송 크기를 줄이기 위해 on chip 버퍼에 저장한다. 이때 on chip 버퍼는 Grouped convolution에 사용하였던 버퍼를 재사용함으로써 on chip 버퍼 크기를 줄일 수 있다.

하지만, 각 layer의 output feature map을 on chip 버퍼에 저장하기

위해서는 기존의 on chip 버퍼보다 더 큰 크기의 버퍼를 필요로 한다. 기존의 on-chip 버퍼는 Grouped convolution과 Fused convolution의 최종 output feature map 데이터를 저장하기 위한 용도로 사용되었지만, full weight 재사용 dataflow를 Inception-C 모듈에 적용하게 된다면 Inception-C 모듈을 이루는 layer들의 output feature map 데이터까지 담을 수 있는 크기여야 하기 때문이다.

계산 결과, mixed convolution을 위해 필요한 buffer 0, buffer 1의 크기의 합이 153KB이었고, Inception-C 모듈을 full weight 재사용 dataflow로 처리하기 위해서 필요한 buffer0, buffer 1의 크기의 합이 193KB이다. 총 40KB의 on chip 버퍼가 추가적으로 필요하다. 하지만 이는 weight 버퍼 크기의 감소량이 더 크므로 전체적으로는 on chip 버퍼의 총량은 full weight 재사용 방법을 적용하기 전보다 감소하게 된다.

이렇듯, full weight 재사용 방법을 Inception-C 모듈에만 적용하여 on-chip 버퍼 메모리 크기는 줄일 수 있으며, input feature map 데이터 또한 on-chip 버퍼 메모리에 저장하기 때문에 row-based weight 재사용 dataflow만 적용한 mixed convolution 방법보다 더 적은 양의 off-chip 메모리 접근량 및 데이터 전송 크기를 감소시킨다.

# 제 4 장 Inception V4 가속기 구현

## 4.1 FPGA 하드웨어 가속기 시스템

본 연구에서 제안한 Mixed convolution 방법을 FPGA 상에서 검증하기 위하여 선행 연구의 Row-based weight 재사용 dataflow를 사용하는 하드웨어 가속기를 설계하였다. FPGA 보드를 이용한 하드웨어 가속기 전체 시스템은 그림 4.1과 같다. 먼저 FPGA 보드와 Host PC를 PCIe 인터페이스로 연결하고 이를 통해 데이터 통신을 한다. 또한 FPGA 보드 PL 영역에 가속기 동작을 제어하는 컨트롤러 모듈과 네트워크 추론을 가속하는 가속기 모듈을 설계하였다. 설계한 가속기 시스템은 single layer processing 가속기 구조를 띄고, 내부 컨트롤러 모듈은 Host PC로부터 전달받은 layer configuration 데이터를 가속기 모듈에 전달한다. 가속기 모듈은 이 데이터를 바탕으로 convolution, max pooling, average pooling 연산 등을 수행한다.

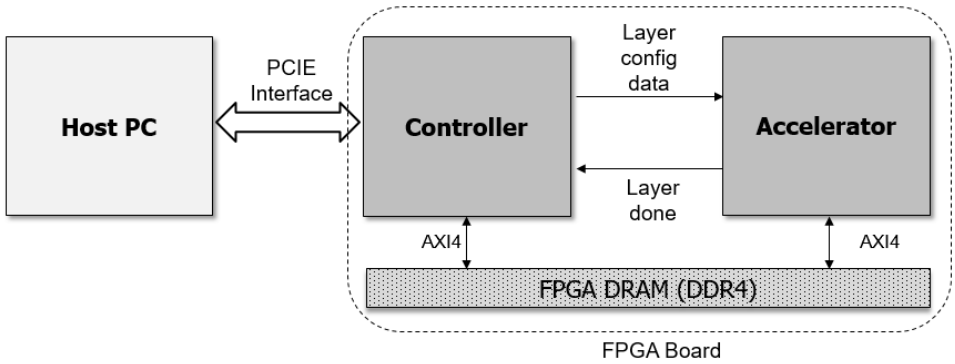



그림 4.1 FPGA 하드웨어 가속기 시스템

## 4.2 가속기 모듈 내부 구조

가속기 모듈은 1개의 layer를 처리하는 연산 모듈로서 convolution, max pooling, average pooling 등의 연산을 수행한다. 그 구조는 그림 4.2와 같다. 가속기 모듈은 크게 컨트롤러 모듈로부터 전달받은 layer configuration 데이터를 바탕으로 control signal을 생성해내는 Data Controller, 각 input 데이터, weight 파라미터 데이터, output를 off-chip 메모리에 저장 혹은 읽어오는 Loader와 Saver, MAC 연산을 담당하는 Processing Element(PE)로 이루어져 있다.

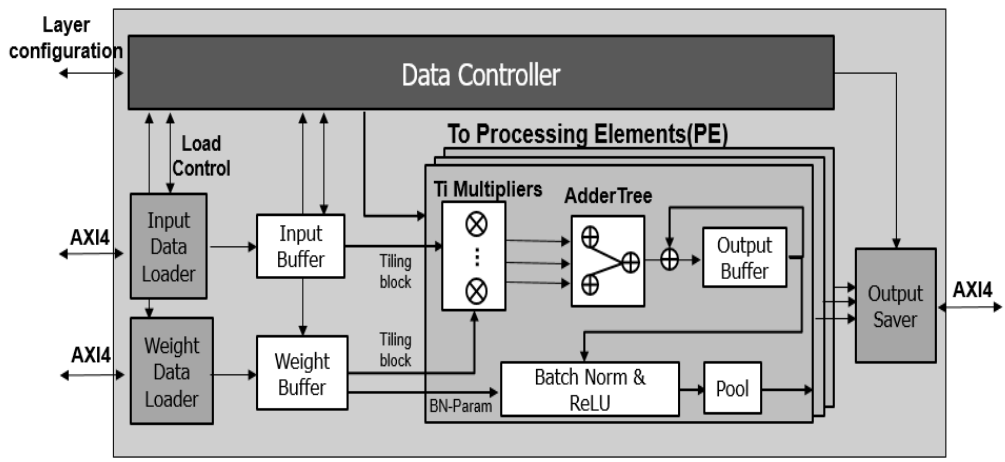



그림 4.2 가속기 모듈 내부 구조

Row-based weight 재사용 방식에서 가속기 모듈의 동작은 Input 데이터와 weight 파라미터 데이터를 off-chip 메모리로부터 load하면서 시작된다. Input Data Loader는 convolution 연산에 필요한 kernel의 크기 데이터를 바탕으로 필요한 크기만큼 input 데이터를 Input Buffer에 load하고 Weight Data Loader는 1개의 layer를 처리하기

위해 필요한 모든 weight 파라미터 데이터를 on-chip Weight Buffer에 load한다.

해당 데이터들의 on-chip 버퍼 메모로의 load가 완료되면, on-chip 버퍼 메모리로부터 tile 블록을 가져와 Processing Element에 넣어 convolution 연산을 시작한다. 해당 연산은 To 개의 Processing Element를 이용하여 To개의 tile block을 병렬적으로 처리한다. 하나의 Processing Element에는 Ti개의 kernel multiplier와 multiply 결과를 2개씩 더하는 Adder Tree로 이루어져 있어 MAC 연산을 수행한다. MAC 연산을 거쳐 나온 partial sum 데이터는 on-chip 버퍼에 저장되어 관리되며, 모든 input channel에 대하여 convolution이 끝나고 나온 최종 output 데이터는 Batch normalization과 quantization을 거쳐 8 bit activation 데이터로 나오게 된다. 그 후, Output Saver를 통해 off-chip 메모리로 저장되거나, 적절한 on-chip 버퍼 메모리로 저장된다.

Batch normalization은 convolution의 결과에 scale과 bias를 이용하여 해당 값의 분포를 변형시킨다[]. 이는 모델 네트워크 훈련 시에 성능에 많은 영향을 끼치는 layer로 현재 거의 모든 Deep CNN에 적용된다. 본 연구에서 설계한 가속기에서는 batch normalization 연산의 overhead를 줄이기 위하여 학습된 gamma 값과 beta 값, 훈련 시 사용한 이미지 데이터의 이동 평균, 이동 분산을 이용하여 새로운 scale과 bias를 만든다. 식(4.1)은 새로운 scale과 bias를 나타내고 있다.

$$\text{new scale} = \frac{\mu * \gamma}{\sqrt{\sigma^2 + \epsilon}}, \qquad new\ bias = -\frac{\mu * \gamma}{\sqrt{\sigma^2 + \epsilon}} + \beta \qquad (4.1)$$

$\gamma$와 $\beta$는 모델 네트워크 훈련 과정에서 얻어진 gamma와 beta 파라미터이며 $\mu$는 훈련용 데이터의 이동 평균, $\sigma$는 이동 표준편차이다. $\varepsilon$는 $10^{-5}$의 값을 갖는다. 새롭게 구한 scale과 bias를 이용하여 batch normalization을 1번의 곱셈과 1번의 덧셈으로 연산을 단순화하고 하드웨어 비용을 줄이며 연산 식은 식 (4.2)와 같다.

$$y = x * new\ scale + new\ bias \qquad (4.2)$$

x는 convolution 연산 결과 생성된 output 데이터를 의미한다. 새로운 batch normalization 파라미터인 scale과 bias는 조그마한 변화에도 성능에 큰 영향을 끼치므로 16 bit을 이용하여 표현한다. Scale 데이터는 모든 layer의 new scale 값을 토대로 integer bit과 fractional bit을 각각 4 bit과 12 bit으로 표현하였다. 또한, Bias 데이터는 integer bit과 fractional bit을 각각 8 bit으로 표현하였다.

Batch normalization 연산까지 모두 마친 data는 ReLU activation과 quantization을 거쳐 최종 output feature map 데이터로 나온다. ReLU activation은 데이터의 MSB를 이용하여 부호를 바탕으로 연산을 진행하고, 그 이후 bit shift를 이용하여 quantization을 진행한다. 최종 output으로 나오는 activation 데이터는 8 bit으로 표현되며, layer에 따라 integer bit과 fractional bit을 각각 4 bit/4 bit, 3 bit/5 bit으로 표현한다.

## 4.3 Mixed Convolution을 지원하는 Data Controller 구현

Mixed convolution을 수행하려면 on-chip 버퍼 메모리가 추가로 필요하다. Grouped convolution은 2 개의 branch의 1x1 convolution layer의 output으로 나온 feature map 데이터를 on-chip 버퍼 메모리에 저장하고, Fused convolution의 경우 fuse하여 처리하는 layer의 intermediate feature map 데이터를 on-chip 버퍼 메모리에 저장한다. 그리하여 그 용도에 따라 on-chip 버퍼 메모리 3종류(Buffer 0, Buffer 1, Buffer fused)를 추가하고 해당 버퍼 메모리를 관리하는 logic을 추가로 설계하여 Mixed convolution을 수행한다.

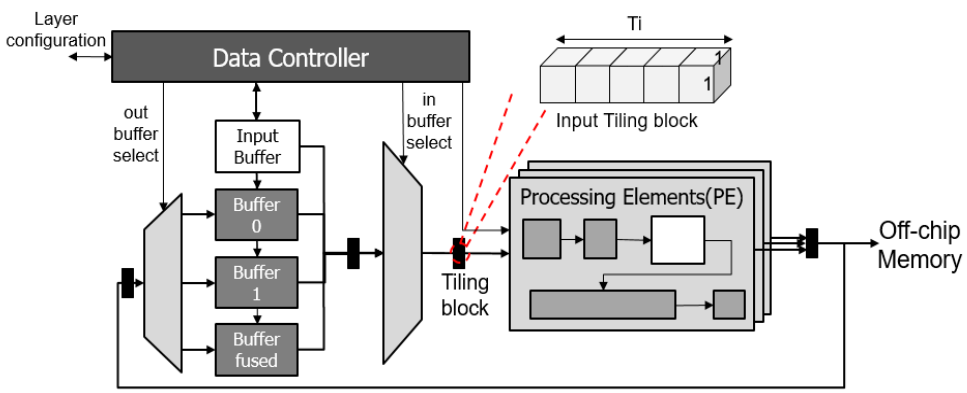



그림 4.3 Data Controller를 이용한 on-chip 버퍼 관리

그림 4.3은 추가한 on-chip 버퍼 메모리를 관리하는 모습을 보여준다. 기존의 Input Buffer에서 input feature map 데이터 블록을

load한 것과 달리, 처리하는 layer에 따라 input feature map 데이터를 적절한 on-chip 버퍼 메모리로부터 load한다. 예를 들면, Grouped convolution의 첫 번째 branch 1x1 convolution layer와 연결된 layer를 처리할 때 input feature map 데이터를 Buffer 0에서 가져온다. On-chip 버퍼 메모리로부터 load한 tile 블록을 이용하여 convolution 연산을 수행하고 최종 output 데이터는 마찬가지로 처리하는 layer에 따라 off-chip 메모리 혹은 on-chip 버퍼 메모리에 저장된다.

On-chip 버퍼 메모리를 관리하는 logic은 Data Controller 모듈에 구현되어 작동한다. Data Controller 모듈은 layer configuration 데이터를 바탕으로 처리하고자 하는 layer의 종류가 일반 Convolution, Grouped convolution, Fused convolution 중 어떤 것인지 파악한다. 그 이후, 해당 layer 종류에 따라 in buffer select 신호와 out buffer select 신호를 생성해낸다. 이 buffer select 신호를 바탕으로 on-chip 버퍼 메모리를 관리하고 각 연산에 필요한 데이터를 알맞게 Processing Element에 넣어 layer processing을 진행한다.

# 제 5 장 실험 결과 및 분석

본 장에서는 앞선 장에서 제안한 Mixed convolution을 적용한 하드웨어를 설계한 결과와 그에 대한 분석을 진행한다. Mixed convolution의 효과를 분석하기 위하여 baseline 하드웨어 가속기와의 on-chip 버퍼 메모리 크기 및 off-chip 메모리 데이터 전송 크기를 비교한다. Feature map 데이터와 weight 파라미터 데이터는 8 bit을 사용하며, partial sum 데이터는 32 bit, batch normalization 파라미터는 scale과 bias 모두 16 bit을 사용한다.

## 5.1 Off-chip 메모리 데이터 전송 크기 비교

Mixed convolution은 feature map 데이터에 대해 off-chip 메모리 접근량 및 전송 데이터 크기를 줄이기 위해 제안되었다. 그러므로, 본 절에서는 row-based weight 재사용 dataflow를 사용하는 single layer processing 가속기(Baseline 하드웨어 가속기)와 Mixed convolution을 지원하는 row-based weight 재사용 dataflow 기반의 하드웨어 가속기의 off-chip 메모리 데이터 전송 크기를 분석하였다.

그림 5.1은 데이터 재사용 방법에 따라 feature map 데이터에 대한 Inception 모듈별 off-chip 메모리 데이터 전송 크기 비율을 나타낸 것이다. Grouped는 Grouped convolution만 적용한 결과이고, Fused는 Fused convolution만 적용한 결과이며 Mixed와 Mixed+FWR은 각각 Mixed convolution을 적용한 것과 Mixed convolution과 full weight

재사용 dataflow를 적용한 것의 결과이다. 또한, Mixed+FWR은 full weight 재사용 dataflow를 Inception-C 모듈에만 적용한 것으로서, Inception-B 모듈과 Inception-A 모듈에 대해서는 row-based weight 재사용 dataflow를 적용하여 Mixed와 동일한 off-chip 메모리 데이터 전송 크기를 갖는다. Total은 Inception 모듈의 데이터 크기를 모두 합한 것으로 Inception V4 네트워크에 구성된 Inception 모듈 모두를 고려한 결과이다.

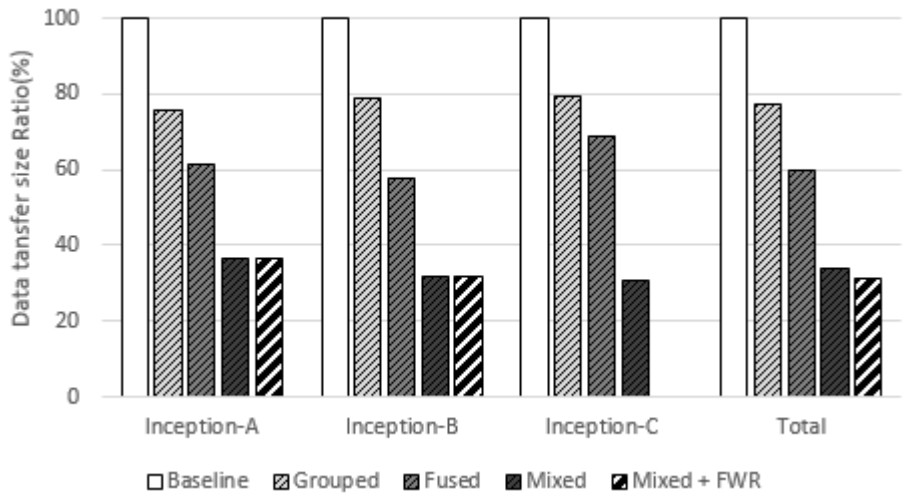



그림 5.1 Inception 모듈 별 off-chip 메모리 데이터 전송 크기 비율

| | Inception A | Inception B | Inception C | Total |
|---|---|---|---|---|
| Data size | 3,841 KB | 2,793 KB | 967 KB | - |
| 반복 횟수 | 4 | 7 | 3 | - |
| 총 Data size | 15,366 KB | 19,551 KB | 2,900 KB | 37,817 KB |

표 5.1 Baseline의 Inception 모듈 별 off-chip 메모리 데이터 전송량

그림 5.1을 보면, Mixed convolution을 적용한 방법과 full weight 재사용 dataflow를 적용한 방법 모두 모든 Inception 모듈에서 63% 이상의 off-chip 메모리 데이터 전송 크기 감소를 이루어 냈다. Inception V4 네트워크의 모든 Inception 모듈을 고려한 수치(그림 5.1의 Total)에서는 Mixed의 경우 66%, Mixed+FWR의 경우 소폭 증가한 69%의 감소를 이루어 냈다. Mixed+FWR의 경우 Inception-C 모듈에 대해서 off-chip 메모리 데이터 전송 크기가 0임을 알 수 있는데, 이는 full weight 재사용 dataflow를 적용할 때는 feature map 데이터를 on-chip 버퍼 메모리에 저장하기 때문에 off-chip 메모리 접근이 일어나지 않는다. 또한, 모든 Inception 모듈에 대해서 Grouped convolution과 Fused convolution만 사용하였을 때보다 2가지 convolution을 모두 적용한 Mixed convolution을 사용하였을 때 더 많은 off-chip 메모리 데이터 전송 크기 감소를 이루어 냈다. 이는 Inception 네트워크를 처리할 때, input feature map 데이터를 최대한 재사용하는 방법과 intermediate feature map 데이터를 최대한 재사용하는 방법 모두를 고려해야 함을 보여주는 결과다.

표 5.1은 Baseline 하드웨어 가속기의 Inception 모듈 별 off-chip 메모리 데이터 전송 크기를 나타낸 것이다. Inception V4 네트워크에서 feature map 데이터를 가장 많이 차지하고 있는 모듈은 Inception-B 모듈로서, Mixed convolution을 통해 가장 많은 off-chip 메모리 데이터 전송 크기를 감소시킬 수 있었다. 또한, full weight 재사용 dataflow를 Inception-C에 적용하면 해당 크기만큼 추가적으로 off-chip 메모리 데이터 전송 크기를 감소할 수 있다.

## 5.2 On chip 버퍼 크기 비교

제안된 데이터 재사용 방법은 추가 on chip 버퍼 메모리를 필요로 한다. 본 절에서는 Baseline 하드웨어 가속기와 제안된 데이터 재사용 방법을 FPGA 상에 구현하기 위해 필요한 총 on chip 버퍼 크기를 비교한다. 제안된 데이터 재사용 방법은 row-based weight 재사용 dataflow를 사용하는 Mixed convolution과 full weight 재사용 dataflow를 사용하는 Mixed convolution 2가지를 모두 분석한다.

먼저 Baseline 하드웨어 가속기를 FPGA 상에 구현하기 위해 필요한 on chip 버퍼의 크기는 식 (5.1)과 같다. 차례대로 weight 버퍼, input 버퍼, output 버퍼의 크기를 구하여 총합을 계산한다. L은 Inception V4 네트워크의 모든 layer 집합이다.

$$\text{On chip buffer} = \max_{l\in[1,L]}(param_l) + \max_{l\in[1,L]}(input\_buff_l) + \max_{l\in[1,L]}(output\_buff_l) \quad (5.1)$$

Mixed convolution은 Baseline과 비교하여 3 종류의 on-chip 버퍼가 추가된다. Buffer 0, Buffer 1은 Grouped convolution과 Fused convolution의 output feature map을 저장하기 위한 용도로 사용되며, Buffer fused은 Fused convolution의 intermediate feature map을 저장하기 위해 사용된다. 총 필요한 on chip 버퍼의 크기는 식 (5.2)와 같다. 차례대로 weight 버퍼, input 버퍼, output 버퍼, buffer 0, buffer 1, buffer fused의 크기를 구하여 총합을 계산한다. M은 새롭게 정의된 layer의 집합으로, Grouped convolution과 Fused convolution에 의해

Inception V4 네트워크를 처리하는 layer가 바뀐 것을 반영한 것이다. GR은 Grouped convolution으로 처리하게 되는 layer 집합을 의미하며, FU는 Fused convolution으로 처리하게 되는 layer 집합을 의미한다.

$$\text{on chip buffer} = \max_{l \in M}(param_l) + \max_{l \in M}(input\_buff_l) + \max_{l \in M}(output\_buff_l) + \max_{l \in [GR]}(2nd\_out\_fmap_l) + \max_{l \in [GR]}(3rd\_out\_fmap_l) + \max_{l \in [FU]}(fused\_input\_buff_l) \quad (5.2)$$

Inception-C 모듈에 Full weight 재사용 dataflow를 적용하여 처리하는 Mixed convolution의 경우는 식 (5.2)에서 buffer 0와 buffer 1의 크기를 구하는 부분이 변경된다. Full weight 재사용 dataflow에서는 feature map을 on chip 버퍼에 모두 저장하므로 buffer 0와 buffer 1의 크기가 Inception-C 모듈의 feature map 크기를 고려하여 설계되어야 한다. Inception-C 모듈의 가장 큰 feature map 크기만큼 buffer가 필요하므로 기존의 Mixed convolution에서 필요한 buffer 0, 1의 크기와 비교하여 max 값을 갖는다.

식 (5.1), (5.2)를 이용하여 각 방법의 on chip 버퍼 크기를 구하면 표 5.2과 같다. Mixed convolution을 위해서는 Baseline에 비해 421 KB의 추가 on chip 버퍼 메모리가 필요함을 알 수 있다. 또한, Inception-C 모듈에 full weight 재사용 dataflow를 적용하면 218 KB의 추가 on chip 버퍼 메모리가 필요하며 이는 Mixed convolution에 비해 감소한 결과를 보여준다. 표 5.3 각 방법에 따라 on chip buffer 종류별 크기를 나타냈다.

표 5.2 On chip 버퍼 크기 비교

| | Baseline | Mixed | Mixed + FWR |
|---|---|---|---|
| On chip 버퍼 크기(KB) | 1,384 | 1,805 | 1,602 |

표 5.3 종류별 on chip 버퍼 크기 비교

| | Baseline | Mixed | Mixed + FWR |
|---|---|---|---|
| Input 버퍼 (KB) | 68 | 68 | 68 |
| Output 버퍼 (KB) | 19 | 19 | 19 |
| Weight 버퍼 (KB) | 1,297 | 1,540 | 1,297 |
| Buffer 0(KB) | 0 | 76 | 96 |
| Buffer 1 (KB) | 0 | 76 | 96 |
| Buffer fused (KB) | 0 | 26 | 26 |

## 5.3 Full weight 재사용과 on-chip 버퍼 크기 분석

이 절에서는 full weight 재사용 dataflow과 on-chip 버퍼 크기를 분석한다. 3장에서 full weight 재사용 dataflow를 Inception-C 모듈에만 적용하여 on-chip 버퍼 크기를 줄일 수 있다고 하였다. 또한, off-chip 메모리 접근량 및 데이터 전송 크기 또한 감소할 수 있다고 하였다. 이 full weight 재사용 dataflow를 사용하는 시점을 Inception-B 모듈과 Reduction-B 모듈로 적용한 결과를 바탕으로 Inception-C 모듈에만 적용하는 방법의 타당성을 밝힌다.

우선 표 5.4는 full weight 재사용 dataflow를 적용하는 시점에 따른 on-chip 버퍼 메모리를 나타낸 것이다. Full weight 재사용 dataflow는 feature map 데이터를 여러 번 load하는 것을 막기 위해 feature map 데이터를 on-chip 버퍼에 저장한다. 이를 지원하기 위해 Buffer 0, Buffer 1의 크기가 커야 하며 full weight 재사용 dataflow를 적용하는 시점이 빠를수록 필요로 하는 on-chip 버퍼 크기는 커진다. 표 5.4는 Buffer 0, Buffer 1, Buffer fused, weight buffer, input buffer, output buffer를 모두 합친 수치이며, Inception-C 모듈부터 적용한 것이 on-chip 버퍼 메모리 크기면에서 가장 좋음을 알 수 있다.

**표 5.4 Full weight 재사용 dataflow 적용 시점에 따른 on-chip 버퍼 크기 비교**

| | Inception-B | Reduction-B | Inception-C |
|---|---|---|---|
| On chip 버퍼 크기(KB) | 1,988 | 1,795 | 1,602 |

## 5.4 FPGA 리소스 사용량 비교 분석

이 절에서는 Baseline 하드웨어 가속기와 Mixed convolution을 지원하는 하드웨어 가속기를 FPGA상에 구현하여 각 하드웨어의 FPGA 리소스 사용량을 비교 분석한다. FPGA 보드는 Xilinx KCU 1500 Kit를 사용하였다.

표 5.5는 Baseline 하드웨어 가속기와 Mixed convolution을 지원하는 하드웨어의 FPGA 상 리소스 및 특징을 비교한 것이다. 두 가속기 모두 299 x 299 크기의 입력 이미지를 사용하였고, 200MHz 동작 주파수에서 작동한다. Baseline 하드웨어 가속기와 비교하여 Mixed convolution 가속기는 LUT 리소스와 Flipflop 리소스를 더 많이 사용한다. 이는 Grouped convolution과 Fused convolution을 처리하기 위해 늘어난 control signal 및 여러 logic 때문이다. 하지만 리소스 증가량이 다소 작음을 확인할 수 있다. 또한, BRAM 리소스도 증가하였다. 이는 앞선 장에서 분석하였던 수치만큼 on chip 버퍼가 증가하여 나타난 것으로 필요한 추가 on chip 버퍼 크기만큼 BRAM 리소스가 증가한 것을 확인할 수 있다. 반면, 두 가속기의 DSP 리소스 사용량은 동일하다. 이는 구현한 두 가속기 모듈에서 Processing Element의 개수를 동일하게 하였기 때문이다. 즉, 두 가속기에서 사용하는 MAC 연산기 개수는 32x32로, tiling factor인 Ti, To로 결정되며 두 가속기는 같은 Ti, To 크기를 갖는다. 그러므로 DSP 리소스 사용량은 동일한 것은 확인할 수 있다.

Baseline 하드웨어 가속기와 Mixed convolution을 지원하는 가속기의 off chip 메모리 bandwidth는 각각 704MB/s, 533MB/s로 Mixed convolution 가속기가 Baseline 대비 24% 정도 감소한 것을

보인다. 이는 Mixed convolution을 적용하여 feature map 데이터에 대해 off chip 메모리 데이터 전송 크기를 줄였기 때문이며 감소한 bandwidth 크기만큼 off-chip 메모리 접근 전력 소모량도 줄 수 있음을 알 수 있다.

**표 5.5 FPGA 리소스의 사용량 비교 분석**

| | Baseline | Mixed |
|---|---|---|
| Platform | XCKU115 | XCKU115 |
| Frequency | 200 MHz | 200 MHz |
| CNN Size (GOP) | 87.3 | 87.3 |
| Image Size | 299 x 299 | 299 x 299 |
| LUTs (k) | 81.3 ((*)12.26%) | 86.7 ((*)12.59%) |
| FFs (k) | 117.0 ((*)8.82%) | 118.2 ((*)8.91%) |
| BRAMs (36Kb) | 520 ((*)24.10%) | 615((*)28.47%) |
| DSPs | 647 ((*)11.72%) | 647((*)11.72%) |
| Bandwidth (MB/s) | 704 | 533 |

(*): FPGA 보드의 전체 리소스 양 대비 사용량 비율

# 제 6 장 결론

본 연구에서는 Deep CNN 중 이미지 분류기 중 하나인 Inception V4 네트워크를 target으로 하는 하드웨어 가속기를 설계하였다. 그와 동시에 하드웨어 가속기 상에서의 off-chip 메모리 접근 및 데이터 전송 크기를 줄이기 위하여 Inception 모듈의 구조를 고려한 convolution 방법 및 데이터 재사용 방법을 제안하였다. 제안한 방법은 Inception 모듈의 input feature map 데이터와 각 branch의 intermediate feature map 데이터를 최대한 재사용하는 방법으로 그 두가지 방법을 모두 적용하여 처리하는 Mixed convolution을 사용하면 off-chip 메모리의 데이터 전송 크기를 감소시킬 수 있다. 제안된 방법의 효율성을 검증하기 위하여, Xilinx Kintex Ultrascale FPGA 보드 위에 제안된 방법을 이용하는 single layer processing 가속기를 구현하여 선행 연구와 비교하였다. 그 결과, 218 KB의 추가 on chip 버퍼를 이용하여 Mixed convolution을 적용하면, Inception V4 네트워크의 모든 Inception 모듈에서 feature map 데이터에 대해 63% 이상의 off chip 메모리 데이터 전송 크기를 감소시킨다. 또한, Inception V4 전체 네트워크 측면에서 feature map 데이터의 off chip 메모리 전송 크기를 58 MB에서 32 MB로 45% 감소시킨다.

# 참고 문헌


[1] M. Alwani, H.Chen, M.Ferdman, and P.Milder, "Fused-layer CNN accelerators," in *Proceeding of the IEEE/ACM International Symposium on Microarchitecture(MICRO)*, 2016, pp. 1-12.

[2] D. T. Nguyen, H. Kim, and H.-J. Lee, "Layer-specific Optimization for Mixed Data Flow with Mixed Precision in FPGA Design for CNN-based Object Detectors," *IEEE Trans. Circuits Syst. Video Technol.*, 2020.

[3] C.Szegedy, W.Liu, Y.Jia, P.Sermanet, S.Reed, D.Anguelov, D.Erhan, and A.Rabinovich, "Going deeper with convolutions," in *Proceedings of te IEEE Conference on Computer Vision and Pattern Recognition(CVPR), 2015, pp. 1-9.*

[4] C.Szegedy, V.Vanhouche, S.Ioffe, J.Shlens, and Z.Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition(CVPR), 2016, pp. 2818-2826.*

[5] C.Szegedy, S.Ioffe, V.Vanhouche, and A.Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in AAAI, *vol. 31, no. 1, Feb. 2017.*

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition(CVPR), 2016, pp. 770–778.*

[7] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, *pp. 7263–7271.*

[8] J.Li et al., "SmartShuttle: Optimizing off-chip memory accesses for deep learning accelerators," in *Proceedings of the IEEE conference on Design, Automation & Test in Europe Conference & Exhibition(DATE), 2018, pp. 343-348*.

[9] K. Guo, S. Zeng, J. Yu, Y. wang, and H. Yang, "[DL] A survey of FPGA-based neural network inference accelerators," ACM Trans. Reconfigurable Technol. Syst., vol. 12, no. 1, pp. 2:1-2:26, Mar. 2019.

[10] O. Russakovsky, et al., "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision, vol. 115, pp. 211-252, 2015*.

[11] D. D. Lin, S. S. Talathi, and V. S. Annapureddy, “Fixed Point Quantization of Deep Convolutional Networks,” in Proc. Int. Conf. Mach. Learn. (ICML), 2016, pp. 2849–2858.

[12] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding,” 2015. [Online]. Available: arXiv:1510.00149.

[13] A. Zhou, A. Yao, K. Wang, and Y. Chen, "Explicit Loss-Error-Aware Quantization for Low-Bit Deep Neural Networks," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition(CVPR), 2018, pp. 9426-9435.

[14] D. T. Nguyen, T. N. Nguyen, H. Kim, and H.-J. Lee, “A high-throughput and power-efficient FPGA implementation of YOLO CNN for object detection,” *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 27, no. 8, pp. 1861–1873, 2019.

[15] A. Krizhevsky, I. Sutskever, and G. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Proc. Neural Inf. Process. Syst., 2012, pp. 1097–1105*.

[16] S. Bianco, R. Cadene, L. Celona and P. Napoletano, "Benchmark Analysis of Representative Deep Neural Network Architectures." in *IEEE Access, vol 6, pp. 64270-64277*.

[17] X. Lin, S. Yin, F. Tu, L. Liu, X. Li, and S. Wei, "LCP:a Layer Clusters Paralleling mapping method for accelerating Inception and Residual networks on FPGA," in *Proceedings of the 55th Annual Design Automation Conference(DAC), 2018, pp. 1-6*.

## Abstract

# Data Reuse Optimization for an FPGA implementation of Inception V4 Networks

Byeongki Song
Electrical and Computer Engineering
The Graduate School
Seoul National University

Deep Convolutional neural networks (DCNN) has been widely used in computer vision and achieved high performance enhancement. In addition, a lot of accelerator designs has been proposed using FPGA for CNN inference. DCNNs generate huge amounts of weight parameters and intermediate feature map data which requires many off-chip memory accesses during inference on FPGA accelerator. This leads to performance degradation and poor energy efficiency.

To reduce off-chip memory accesses, various of data reuse methods have been proposed. However, previous data reuse methods show low reusability on Inception V4 network which has high performance on image classification.

Considering branch topology of inception module, proposed data reuse method named Mixed convolution reuse feature map data using on-chip memory. Mixed convolution takes advantages of both Grouped convolution and Fused convolution which reuse input feature map data of inception module and intermediate feature map data of a

branch respectively. As a result, Mixed convolution minimizes off chip feature map data transfer of inception modules, reducing by 66.4%, from 37MB to 12MB using extra 421KB on-chip buffer memory. In addition, to optimize on-chip buffer memory size required to minimize off-chip data transfer, Full weight reuse dataflow is applied to Inception-C module which results in reduction of off-chip feature map data transfer of inception module, reducing by 68.6%, from 37MB to 11MB using extra 218KB on-chip buffer memory.

Keywords : Data Reuse, Off chip memory data transfer size, CNN Accelerator, Inception V4 Network
Student Number : 2019-20004