뉴럴 네트워크의 뉴로모픽 하드웨어와 소프트웨어 공동설계

신태환
인천대학교

정재용*
인천대학교

Ⅰ. 서 론

1. 연구배경

최근 글로벌 IT 기업들이 다양한 분야에 딥 러닝[1] 기술을 적용하고 있다. 실제로 딥 러닝 기술은 음성 인식 기능을 갖춘 개인 비서 소프트웨어 애플(Apple)의 시리(Siri), 얼굴 인식 기능을 갖춘 페이스북(Facebook)의 딥페이스(DeepFace)에 적용되었고, 구글(Google)은 자율 주행 자동차와 음성 및 영상 인식 등 다양한 분야에 딥 러닝 기술을 적용하였다. 그 중 구글 딥마인드(DeepMind)가 개발한 인공 지능 바둑 프로그램 알파고(AlphaGo)는 인간과의 대결에서 승리하면서 딥 러닝에 대한 관심을 증폭시켰다.

딥 러닝(deep learning)이란 인공 지능의 한 분야로 인간의 두뇌에 뉴런의 정보 처리 과정을 모사하여 모델링 한 심층 신경망(deep neural network)을 기반으로 한 기술이다. 딥 러닝은 주어진 학습 데이터로부터 특정 목적을 달성하기 위해 스스로 프로그램을 생성하는 기계 학습 알고리즘의 하나로 컴퓨터 비전, 음성 인식, 자연어 처리 등의 다양한 분야에 접목시킬 수 있다[2-5].

딥 러닝은 1980년대에 처음 등장하였으나 학습 시간이 오래 걸리는 문제와 과적합(overfitting) 문제로 인해 실용화 되지 못 하였다[6]. 또한 딥 러닝을 구현하려면 다량의 학습 데이터와 복잡한 행렬 연산을 처리할 수 있는 능력이 필요하지만 그 당시 인터넷 기술이 발전하지 않아 학습 데이터를 수집하기가 어려웠으며 하드웨어의 연산 처리 능력이 낮아 발전이 정체되고 있었다.

하지만 최근 빅 데이터의 출현으로 딥 러닝 학습에 필요한 막대

*교신저자 (E-mail: jychung@inu.ac.kr)

한 양의 데이터 확보가 가능해졌고 GPGPU(General-Purpose computing on GPU)의 등장으로 하드웨어의 연산 능력이 크게 증가함에 따라 딥 러닝 구현에 필요한 수 많은 연산량을 빠르게 처리할 수 있게 되었다. 그리고 2012년 1000개의 이미지를 분류하는 ILSVRC (ImageNet Large Scale Visual Recognition Competition) 대회에서 컨볼루션 신경망 (convolutional neural network)을 이용한 AlexNet[2] 이 우수한 성적을 거두면서 딥 러닝을 다시금 각광받게 하였다.

딥 러닝 기술은 기존에 수십 년간 개발된 알고리즘보다 뛰어난 성능을 보여주고 있으나 아직 실용화 단계에서 해결해야 할 문제들이 존재한다. 기존 컴퓨팅 방식에서 거대한 심층 신경망의 방대한 연산량을 처리하기 위해 고성능의 프로세서를 이용한다. 그러나 기존 컴퓨팅 방식은 방대한 연산량을 처리하는 과정에서 프로세서의 연산 성능을 저하시키고 많은 에너지를 소비하는 구조를 갖는다. 이에 따라 대규모 심층 신경망을 모바일 장치에서 효율적으로 구현할 수 있는 설계 방안을 모색하기 위해 다음과 같은 연구를 진행하게 되었다.

2. 연구 목적

현재 대부분의 컴퓨팅 방식은 프로세서와 메모리가 분리되어 있는 폰 노이만 구조를 갖는다. 이러한 컴퓨팅 방식에서 심층 신경망을 구현하려면 방대한 연산량을 처리할 수 있는 고성능 프로세서와 다량의 가중치 파라미터(weight parameter)들을 저장할 수 있는 대용량 메모리가 요구된다. 일반적으로 방대한 연산량을 처리하기 위해 처리 요소(processing element)의 수를 늘리는 방법을 이용한다. 하지만 메모리의 낮은 대역폭(bandwidth)이 버스에 병목 현상(bottleneck)을 발생시켜 처리 요소의 수를 제한한다. 또한 연산에 필요한 가중치 파라미터를 불러오기 위해 빈번히 메모리에 접근하는데, 메모리에 접근하기 위한 입출력 연산은 간단한 수치 연산에 비해 상대적으로 높은 전력을 소모시킨다[7]. 대부분의 모바일 장치들은 이러한 컴퓨팅 방식을 기반으로 한다.

기존 연구에서 [8]의 경우, 텐서 분해로 계수(rank)를 선택하고 커널의 차원 분해를 통해 파라미터와 연산량을 감소시키는 압축 방식을 제안하였고, 모바일 GPU 를 사용하여 실험을 진행하였다. [9]의 경우, 컨볼루션 레이어와 완전 연결 레이어의 파라미터를 각 레이어의 특성에 맞게 양자화하여 연산의 가속과 파라미터의 압축을 동시에 수행하는 방식을 제안하였고, CPU 를 사용하여 실험을 진행하였다. 각 논문은 폰 노이만 컴퓨팅 방식에서 신경망을 동작시킬 때, 동작 속도를 향상시키고 전력 소모를 낮추기 위해 각 논문에서 제안하는 압축 방식을 적용하여 연산량과 파라미터의 수를 감소시키는 방안에 대한 연구가 진행되었다.

그러나 신경망의 규모가 커지면 연산량과 파라미터의 수도 같이 증가하게 되고, 근본적으로 폰 노이만 구조에서 신경망을 구현할 때 발생하는 구조적 문제점, 병목 현상으로 인한 연산 성능 저하와 빈번한 메모리 접근으로 발생하는 높은 전력 소모는 개선되지 않는다.

본 논문에서는 대규모 컨볼루션 뉴럴 네트워크를 모바일 장치에서 효율적으로 구현하기 위한 시스템을 제안한다. 컨볼루션 뉴럴 네트워크는 크게 컨볼루션 레이어와 완전 연결 레이어로 구성되어 있다. 컨볼루션 레이어는 가중치의 수가 적고 연산량이 많으나, 완전 연결 레이어는 가중치의 수가 많고 연산량이 적은 특징이 있다. 컨볼루션 뉴럴 네트워크는 뉴로모픽 아키텍쳐를 갖는 뉴로모픽 하드웨어로 연산을 가속시킬 수 있다. 그러나 완전 연결 레이어를 뉴로모픽 하드웨어로 구현할 때, 완전 연결 레이어의 특성상 노드와 노드를 연결하는 커넥션(connection)의 수가 많아 뉴로모픽 하드웨어로 구현하기에는 어려움이 따른다.

따라서, 연산량이 많은 컨볼루션 레이어는 뉴로모픽 아키텍쳐를 갖는 뉴로모픽 하드웨어로 연산을 가속시켜 컨볼루션 레이어의 연산을 처리하고, 파라미터의 수가 많은 완전 연결 레이어는 파라미터의 수를 감소시킬 수 있는 압축 기술을 적용하여 기존 폰 노이만 방식을 이용하는 시스템을 제안한다. 압축 기술을 적용하여 파라미터의 수를 감소시키면, 연산량과 메모리의 접근 횟수가 줄어들어 연산 속도는 증가시킬 수 있고, 전력 소모는 감소시킬 수 있다.

3. 논문의 구성

본 논문은 총 5장으로 구성되어 있으며 다음과 같은 순서로 진행된다.

2장에서는 이미지 분류에 뛰어난 성능을 가진 컨볼루션 뉴럴 네트워크의 구조와 각 레이어에 대해 간단히 살펴본다. 또한 인간의 뇌를 모방한 뉴로모픽 아키텍쳐와 뉴럴 네트워크의 가중치 파라미터의 수를 감소시킬 수 있는 압축 기술을 살펴본다.

3장에서는 컨볼루션 뉴럴 네트워크를 모바일 장치에서 구현하기 위한 시스템 설계 방법을 제안한다. 또한 인공 신경망 파라미터 수 최적화 알고리즘과 압축된 완전 연결 레이어를 빠르게 처리하기 위해 사용된 희소 행렬 벡터 곱에 대해 서술하였다.

4장에서는 본 논문에서 제안한 시스템 설계 방법을 검증하였으며 압축 기술이 적용된 완전 연결 레이어와 적용되지 않은 완전 연결 레이어의 연산 속도를 다양한 프로세서로 비교하는 실험을 진행하였다.

5장에서는 결론을 도출하였다.

Ⅱ. 컨볼루션 뉴럴 네트워크

1. 컨볼루션 뉴럴 네트워크의 구조

현재 AlexNet, ZFNet, GoogLeNet, VGGNet 등의 다양한 컨볼루션 뉴럴 네트워크 모델이 존재한다. 본 논문에서는 위와 같은 다양한 신경망 중 AlexNet 모델을 이용하여 이미지 분류를 구현하였기 때문에 이 모델에 대해 살펴보고자 한다.

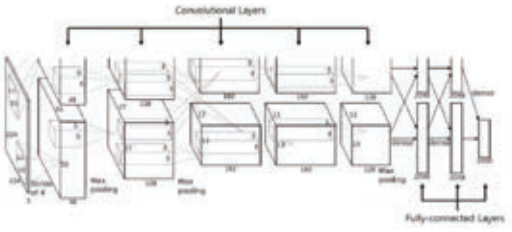
〈그림 1〉은 컨볼루션 뉴럴 네트워크를 기반으로 한 AlexNet 모델이다.

AlexNet 은 크게 5개의 컨볼루션 레이어(convolutional layers)와 3개의 완전 연결 레이어(fully-connected layers)로 구성되어 있으며 특정 컨볼루션 레이어 사이에 서브샘플링(sub-sampling)을 위한 맥스-풀링 레이어(max pooling layer)가 존재한다. AlexNet 에 224 x 224 사이즈의 이미지가 입력되면 컨볼루션 레이어, 맥스 풀링 레이어, 완전 연결 레이어를 거쳐 총 1000개의 결과가 확률 값으로 출력되는 구조를 갖는다.

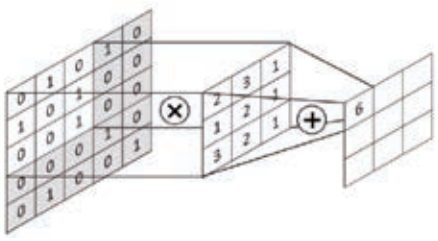
컨볼루션 레이어는 입력된 이미지로부터 컨볼루션(convolution) 연산을 수행하여 특징(feature) 맵을 추출해낸다. 컨볼루션 연산은 일정한 크기와 학습된 가중치 파라미터를 가진 커널이 입력된 이미지를 이동(stride)하면서 픽셀 값에 대응하는 파라미터들과 곱하고 더하는 과정을 반복하여 특징을 추출해낸다. 이 과정을 간단한 예로 〈그림 2〉에 나타내었다.

컨볼루션 레이어에는 여러 종류의 커널이 사용되고 다양한 특징들이 추출된다.

입력된 이미지에 대해 컨볼루션 연산이 수행되고 나면 활성 함수(activation function)의 입력으로 들어가게 된다. 활성 함수는 뉴런에 입력된 값에 따라 뉴런이 출력하는 신호의 세기를 결정하는 함수이다. AlexNet 은 학습 속도를 높이기 위해 ReLU(Rectified Linear Unit) 활성 함수를 사용하였다. ReLU 를 사용하기 전 활성 함수로 사용되던 시그모이드(sigmoid) 함수는 기울기 하강(gradient descent)을 기반으로 한 학습 도중 역전파(backpropagation) 과정에서 레이어가 깊어질수록 가중치가 0으로 수렴하는 문제(gradient vanishing problem)가 발생한다[10]. 이러한 문제를 해결하기 위해 ReLU 가 등장하였으며 식 (1) 과 같이 표현될 수 있다.



〈그림 1〉

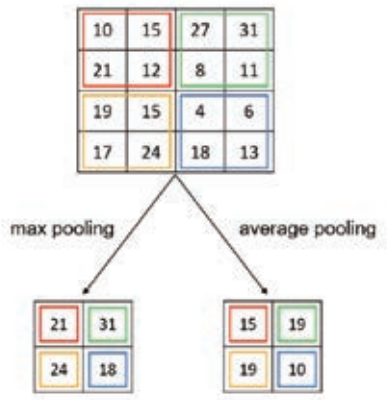


〈그림 2〉

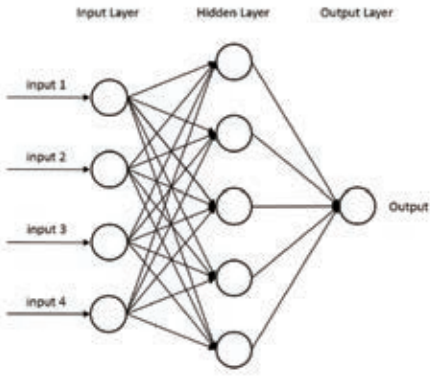
$$f=\begin{cases}(x<0)\ f(x)=0\\(x\geq 0)\ f(x)=x\end{cases} \quad (1)$$

ReLU 는 입력 x 가 0 미만일 경우 0 이 출력되고, 0 이상일 경우 입력 x 를 출력으로 취한다. 또한 ReLU 는 시그모이드 함수에 비해 연산이 간단하기 때문에 학습 속도가 빨라 현재 심층 신경망의 활성 함수로 많이 사용되는 추세이다.

입력된 이미지로부터 컨볼루션 연산이 수행된 후 활성 함수를 거치고 나면 특징 맵(feature map)이 추출되고, 특징 맵은 서브 샘플링을 위해 풀링 레이어를 거치게 된다. 서브 샘플링은 입력된 이미지의 차원을 축소시키는 기법으로, 주로 특징 맵의 이미지 크기를 줄여 연산량을 감소시키기 위한 용도로 사용된다. 일반적으로 풀링 레이어는 평균 풀링과(average pooling)와 맥스 풀링(max pooling)을 많이 사용한다. 평균 풀링은 커널에 대응되는 픽셀들의 평균을 취하여 이미지의 크기를 줄이는 방법이고, 맥스 풀링은 커널에 대응되는 픽셀들 중 가장 큰 값을 선택하여 이미지의 크기 줄이는 방법이다. 맥스 풀링과 평균 풀링의 간단한 예를 〈그림 3〉에 나타내었다.



〈그림 3〉



〈그림 4〉

풀링은 일반적으로 2 x 2 커널을 사용하여 겹치는 부분 없이 2 칸씩 이동(stride)하기 때문에 이미지의 가로와 세로의 크기가 반으로 줄어들게 된다. 하지만 AlexNet 의 경우 3 x 3 커널을 사용하여 2칸씩 이동하는 오버래핑 풀링(overlapping pooling)을 이용하였고 과적합(overfitting)을 감소시켜 예측 성능을 향상시켰다.

컨볼루션 레이어와 풀링 레이어를 거치고 나면 완전 연결 레이어의 연산을 수행하게 된다. 완전 연결 레이어는 입력 레이어의 각각의 노드들이 다음 레이어의 모든 노드들과 서로 연결 되어있는 구조로, 〈그림 4〉에 간단한 완전 연결 레이어의 구조를 나타내었다.

완전 연결 레이어는 컨볼루션 레이어에서 추출한 특징을 입력으로 받아 1000개 객체로 분류하는 연산을 수행하고, 30개의 완전 연결 레이어를 거치고 나면 최종적으로 1000 개의 객체들이 확률 값으로 출력된다. 이 중 가장 높은 확률 값을 갖는 객체는 컨볼루션 뉴럴 네트워크가 입력된 이미지에 대해 최종적으로 판단한 객체를 의미한다.

컨볼루션 레이어와 완전 연결 레이어의 파라미터 개수와 연산량은 서로 상반된 특성을 가지고 있다. 컨볼루션 레이어는 파라미터 개수가 적지만 연산량이 많고, 완전 연결 레이어는 파라미터 개수가 적지만 연산량이 많은 특성이 있다. 컨볼루션 레이어의 총 가중치 파라미터 개수는 약 2.3 M 개이고 연산량은 약 666 M FLOPs 이며, 완

〈표 1〉

Layer	Weight parameters	FLOPs
Convolutional layer 1	35K	105M
Convolutional layer 2	307K	224M
Convolutional layer 3	885K	150M
Convolutional layer 4	664K	112M
Convolutional layer 5	442K	75.0M
Fully-connected layer 6	37.7M	37.7M
Fully-connected layer 7	16.8M	16.8M
Fully-connected layer 8	4.1M	4.1M

전 연결 레이어의 총 가중치 파라미터 개수는 약 58.6 M 개이고 연산량은 약 58.6 M FLOPs 이다. 컨볼루션 뉴럴 네트워크의 각 레이어마다 요구되는 연산량과 가중치 파라미터의 수를 〈표 1〉에 나타내었다[8].

2. 뉴로모픽 아키텍쳐

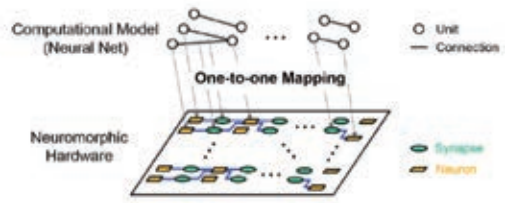
현재 일반적으로 널리 사용되는 폰 노이만 컴퓨팅 방식은 프로세서와 메모리가 분리되어 있고, 버스를 통해 데이터를 주고 받는 구조를 갖는다. 이러한 구조는 뉴럴 네트워크 모델을 구현하기에, 속도와 전력 소모 측면에서 비효율적인 구조를 가지고 있다. 폰 노이만 구조에서 신경망 모델을 구현하기 위해 주로 처리 요소(processing element)의 수를 늘려 수 많은 연산량을 처리하고, 가중치 파라미터들을 외부 메모리(off-chip memory)에 저장하는 방법을 이용한다.

하지만 외부 메모리의 느린 대역폭(bandwidth)이 버스에 병목 현상을 발생시켜 처리 요소의 수를 제한하고, 이로 인해 같은 시간 동안 처리할 수 있는 프로세서의 처리량도 제한되어 신경망을 구현하는 속도가 느려지게 된다. 또한 신경망 구현에 필요한 데이터들을 불러오기 위해 빈번히 외부 메모리에 접근해야 하지만 〈표 2〉에서 보는 것과 같이 외부 메모리에 접근하는 것은 상대적으로 높은 전력이 소모된다[7,11]. 이러한 폰 노이만 컴퓨팅 방식이 갖는 한계를 뉴로모픽(neuro-morphic) 아키텍쳐로 극복할 수 있다[12].

뉴로모픽 아키텍쳐는 저전력으로 동작하는 인간의 뇌 구조를 모방하여 개발된 구조로 인간의 생물학적 뉴런의 정보 처리 과정을 모델링하여 하드웨어로 구현한 기술이다. 이러한 뉴로모픽 아키텍쳐를 이용한다면 컨볼루션 레이어의 연산을 가속시킬 수 있다.

〈표 2〉

Operation	Energy [pJ]	Relative Cost
32 bit int ADD	0.1	1
32 bit float ADD	0.9	9
32 bit int MULT	3.1	31
32 bit float MULT	3.7	37
32 bit 32KB SRAM	5	50
32 bit DRAM	640	6400



〈그림 5〉

컨볼루션 뉴럴 네트워크는 컨볼루션 레이어와 완전 연결 레이어로 나누어 질 수 있다. 컨볼루션 레이어는 입력 데이터로부터 다양한 특징을 추출하기 위해 서로 다른 커널(kernel)들이 존재하고, 각각의 커널들은 컨볼루션 연산을 반복적으로 수행하기 때문에 완전 연결 레이어보다 상대적으로 가중치 파라미터의 수는 적지만 연산량이 많은 특징이 있다. 이와 같은 특징으로 본 논문에서는 컨볼루션 레이어의 가중치 파라미터들을 온 칩(on-chip)에 저장하여 연산을 가속시킬 수 있는 뉴로모픽 하드웨어를 이용하였다[13,14].

이 뉴로모픽 하드웨어는 뉴런(neuron)과 시냅스(synapse)로 구성되어 있으며 〈그림 5〉과 같이 뉴럴 네트워크의 유닛(unit)과 커넥션(connection)을 뉴로모픽 하드웨어의 뉴런과 시냅스로 일대일 대응시켜 실행하게 된다.

뉴런과 시냅스는 병렬로 구성되어 있으며 데이터 저장과 연산을 담당하고, 뉴로모픽 하드웨어에 입력된 데이터는 입력부터 출력까지 연속적으로 시냅스 연산을 거치게 된다.

뉴로모픽 아키텍쳐는 폰 노이만 구조와 달리 온 칩 혹은 오프 칩 메모리를 사용하지 않고 모든 가중치 파라미터를 레지스터에 저장하기 때문에 병목 현상이 발생하지 않으며 저전력으로 동작할 수 있다.

3. 뉴럴 네트워크 압축 기술

컨볼루션 뉴럴 네트워크 모델은 상당히 많은 가중치 파라미터를 가지고 있다. 하지만 대부분의 가중치 파라미터들이 필요 이상으로 많기 때문에 압축 기술을 이용하면 가중치 파라미터의 수를 감소시킬 수 있다[15]. 가중치 파라미터를 감소시키면 연산량을 현저히 줄일 수 있고, 메

모리 사용량도 감소시킬 수 있는 장점이 있다[16]. 본 논문에서는 가중치 파라미터의 수를 감소시키기 위해 행렬 분해(matrix factorization)[17]와 pruning[18]을 이용하였다.

가중치 행렬의 가중치들은 필요 이상으로 많아 선형적으로 압축될 수 있다. 행렬을 압축하기 위한 가장 일반적인 방법은 SVD(Singular Value Decomposition)를 통한 낮은 계수 근사법(low-rank approximation)이다. m x n 크기를 갖는 가중치 행렬 W 에 SVD를 적용하면 식 (2) 와 같이 표현될 수 있다.

$$W = USV^T \tag{2}$$

식 (2) 에서 U 는 $m \times m$ 크기를 갖는 직교행렬, S 는 $m \times n$ 크기를 갖는 대각행렬, V^T는 $n \times n$ 크기를 갖는 유니터리행렬이고, S 는 U 또는 V^T 와 결합될 수 있으며 이는 식 (3) 과 같이 표현될 수 있다.

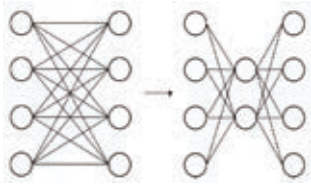
$$W = W_1 W_2 \tag{3}$$

기존 레이어는 두 개의 레이어로 표현될 수 있고 행렬 분해가 적용된 레이어는 기존 레이어와 같은 출력을 갖는다. 기존 레이어에 행렬 분해를 적용하면 〈그림 6〉과 같은 형태를 갖는다.

가중치 행렬은 행렬 분해를 통해 분해될 수 있으나, 가중치 행렬의 대부분이 Full rank 를 갖기 때문에 파라미터의 수는 감소시킬 수 없다. 따라서, 본 논문에서는 가장 큰 값 개를 특이값으로 취하고 나머지는 0으로 만드는 계수 근사법을 이용하였다[14].

가중치 파라미터를 감소시킬 수 있는 방법 중 또 다른 하나는 pruning 이다. pruning 은 작은 값을 갖는 파라미터를 0으로 취하는 방법이다. 일반적으로 작은 파라미터 값을 0으로 만들기 위해 임계값을 정하고 임계값 이하의 값들을 모두 0으로 만든다. 그러나 이러한 방법은 파라미터의 수나 예측 성능을 보장하지 않는다. 따라서, 본 논문에서는 목표 파라미터 개수를 입력으로 받아 파라미터를 정렬하고, 가장 큰 값부터 목표 파라미터 개수까지 파라미터 값들을 유지하면서 나머지 값들을 0으로 만드는 방법을 이용하였다[14].

위 두 가지의 압축 기술을 결합하여 학습된 네트워크에 적용한다면 파라미터의 감소율을 증가시킬 수 있다. 따라서, 본 논문에서는 [14] 에서 제안한 행렬 분해와 pruning 이 결합된 압축 기술을 완전 연결 레이어에 적용하여 파라미터의 수를 감소시켰다.



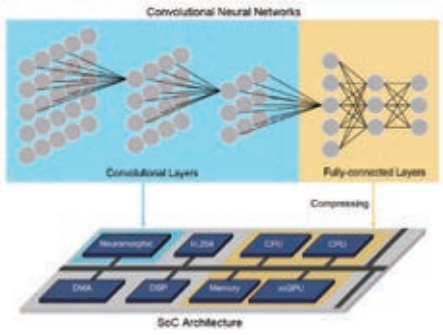
〈그림 6〉

Ⅲ. 뉴럴 네트워크 하드웨어/소프트웨어 공동 설계

컨볼루션 뉴럴 네트워크는 크게 컨볼루션 레이어와 완전 연결 레이어로 구분되며 두 레이어의 특성은 서로 상이하다. 컨볼루션 레이어는 연산량이 많으나 파라미터의 개수는 적고, 완전 연결 레이어는 연산량이 적으나 파라미터 개수는 많다. 이러한 특성을 활용하여 본 논문에서는 〈그림 7〉과 같은 시스템을 제안한다.

컨볼루션 레이어의 경우 파라미터의 수가 적기 때문에 뉴로모픽 하드웨어로 구현하기가 용이하고, 뉴로모픽 하드웨어로 구현 시 다량의 처리 요소(processing element)를 활용하여 저전력으로 높은 처리량을 제공 받아 연산을 가속시킬 수 있다.

스마트 폰의 어플리케이션 프로세서(Application Processor, AP)는 대부분 멀티 코어 CPU 와 모바일 GPU 를 이미 포함하고 있으므로, 이러한 기존의 인프라



〈그림 7〉

를 활용하여 완전 연결 레이어를 소프트웨어로 구현할 수 있다. 완전 연결 레이어의 경우 파라미터 수가 많기 때문에 고밀도의 DRAM 에 저장하는 것이 바람직하며, 요구되는 연산량은 적어 멀티 코어 CPU 또는 모바일 GPU 로도 처리할 수 있다.

본 논문의 기여는 다음과 같이 요약된다.

1) FPGA 를 활용하여 제안하는 시스템의 프로토타입을 실제로 구현하고 예상되는 문제점을 파악하여 해결책을 제시한다.
2) 최신 인공 신경망의 완전 연결 레이어를 다양한 방식의 소프트웨어로 구현하고 최적화하며, 다양한 임베디드 컴퓨팅 플랫폼에서 실행할 때의 성능을 평가한다.

1. 하드웨어 시스템 구현

〈그림 8〉 에 제안한 시스템을 실제로 구현한 구성도를 나타내었고, Xilinx 사의 FPGA VIRTEX-7 (XC7V2000T), SoC 플랫폼 Zynq-7000(XC7Z020)을 이용하였다.

Zynq-7000 은 PS(Processing System) 영역과 PL(Programmable Logic) 영역으로 나누어져 있고 PS 영역은 Dual-core ARM Cortex-A9, PL 영역은 Artix-7 과 동급의 FPGA로 구성되어 있다.

VIRTEX-7 은 뉴로모픽 아키텍쳐를 갖는 뉴로모픽 하드웨어가 구현되어 있고 컨볼루션 레이어의 연산을 수행한다. ZYNQ 의 PS 영역은 딥 러닝 프레임 워크 Caffe[19] 를 이용하여 압축 기술이 적용된 완전 연결 레이어의 연산을 수행한다. ZYNQ 의 PL 영역은 Xilinx 사에서 제공하는 IP를 이용하여 DMA(Direct Memory Access)와 FIFO(First In First Out) 모듈로 설계하였다. DMA 는 데이터를 고속으로 전송시키기 위해 사용되었으며, FIFO 는 모듈 간 동작 속도 차이로 인해 데이터를 임시로 저장

〈그림 8〉

시키기 위해 사용되었다. DMA 는 CPU의 개입 없이 메모리에 직접 접근하여 데이터를 주변 장치들로 전송하는 기능을 수행한다. DMA 가 CPU 대신 데이터 전송에 관여하는 동안 CPU는 다른 작업을 수행할 수 있어 효율성이 높아지는 장점이 있다.

〈그림 8〉 에 구현된 시스템의 동작 과정은 다음과 같다. Caffe 에서 테스트 이미지를 메모리에 저장하면 DMA 는 메모리에 접근하여 Scatter/Gather 모드로 이미지 데이터를 가져온다. Scatter/Gather 모드는 메모리 상에 흩어져있는 데이터들을 모아 하나의 연속적인 데이터처럼 사용할 수 있게 해주는 기능으로 DMA 는 메모리 상에 흩어진 데이터들을 모아 한번에 전송시킬 수 있다. DMA 는 컨볼루션 레이어의 연산을 수행하기 위해 이미지 데이터를 스트림(stream) 형식으로 FPGA 에 전송시킨다. 전송된 이미지 데이터는 모듈 간 동작 속도의 차이로 인해 FIFO 모듈에 저장되고 저장된 이미지 데이터는 RGB 정규화(normalization) 채널로 분리되어 정규화 과정을 거쳐 컨볼루션 레이어 모듈의 입력으로 들어가게 된다. 이미지 데이터는 컨볼루션 레이어의 연산을 거치고 두 개의 FIFO 모듈을 통해 DMA 로 전송되고 전송된 데이터는 다시 Scatter/Gather 모드로 메모리에 저장시킨다. Caffe 는 메모리에 저장된 데이터를 완전 연결 레이어의 입력 벡터로 받아 행렬 연산을 수행하고 최종적으로 테스트 이미지에 대해 컨볼루션 뉴럴 네트워크가 인지한 결과가 출력된다.

2. 완전 연결 레이어 소프트웨어 구현

완전 연결 레이어의 경우 많은 수의 가중치 파라미터가 존재하나 필요 이상으로 많기 때문에 선형적으로 압축이 가능하다. 많은 수의 파라미터는 신경망의 구조적 유연성을 높여 예측 성능을 높이기 위한 기술의 하나로, 훈련 단계에서는 반드시 요구되나 추론 단계에서는 불필요하다. 효율적인 추론을 위하여 파라미터 수를 감소시키는 연구가 많이 진행되고 있으며[20,21], 가중치 행렬의 낮은 계수 근사법(low-rank approximation), pruning 방법 또는 두 방법의 결합을 통해 정확도를 유지하면서 파라미터 수를 수 십배 이상 감소시키는 것이 가능하다. 기존의 파라

미터 수를 감소시키기 위한 연구는 주로 레이어 별로 수행되었으며, 레이어 별로 목표 파라미터 수가 주어 졌을 때 정확도 감소량을 최소화하는 방식으로 최적화가 진행되어 왔다. 그러나 전체 뉴럴 네트워크의 목표 파라미터 수가 정해졌을 때, 각 레이어 별로 목표 파라미터를 어떻게 설정해야 하는지에 대한 연구는 아직 진행되지 않았다. 본 논문에서는 그리디 알고리즘을 통해 레이어 별 최적 파라미터 수를 찾고자 한다. 또한 압축 기술이 적용된 완전 연결 레이어에 CSR 방식을 적용하여 연산 속도를 향상시키고자 한다.

가. 그리디 알고리즘 기반 레이어 별 파라미터 수 최적화

본 논문에서 해결하고자 하는 최적화 문제를 수식으로 나타내면 식 (4)와 같다.

$$\underset{\alpha_1,\ldots,\alpha_n}{\text{Maximize}}\ \mathrm{P}(\alpha_1,\ldots,\alpha_n)\ \textit{subject to}\ \alpha_1+\alpha_2+\cdots+\alpha_n=\beta \tag{4}$$

식 (4) 에서 α_i 는 i 번째 레이어의 파라미터 수, n 은 레이어의 개수, β 는 목표 파라미터 개수, P 는 네트워크의 예측 성능을 의미하며, 목표 파라미터 개수(β)가 정해졌을 때 예측 성능이 최대가 되는 각 레이어의 파라미터 개수($\alpha_1,\ldots,\alpha_n$)를 의미한다. 함수 P 는 각 최적화 변수에 대해 Convex 함수가 아니므로 최적화 패키지를 이용하여 최적해를 구하기가 어렵다. 따라서 본 논문에서는 이 문제를 위해 그리디 알고리즘을 사용하고자 한다.

그리디 알고리즘은 주로 최적의 값이나 경로를 구할 때 사용되는 알고리즘이다. 다양한 경우의 수가 주어졌을 때 최적이라고 판단되는 경우를 선택해나가는 방식으로 진행하여 최종적인 값에 도달하게 된다. 전체적인 경로에 대해 알 수 없어 상황에 따라 최적의 경우를 판단하기 때문에 알고리즘을 통해 얻은 값이 항상 최적의 값은 아니다. 하지만 전체적인 경로를 알 수 없어도 근사적인 값의 도달할 수 있고 속도가 빠르다는 장점이 있다.

2.3 절에서 서술한 압축 기술을 각각의 완전 연결 레이어에 적용하면 가중치 파라미터 개수에 따라 예측 성능이 달라진다. 이때 정해진 파라미터 개수에 따라 각각의 레이어 마다 최대의 예측 성능을 갖는 파라미터 개수를 얻기 위해 그리디 알고리즘을 사용하였다. 그리디 알고리즘을 이용한 최적의 파라미터 개수를 얻는 방법은 다음과 같으며 Algorithm1에 요약하였다.

〈표 3〉

Algorithm 1:

입력 : 목표 파라미터 개수

// 완전 연결 레이어의 기본 파라미터 개수를 기점으로 한다

// 감소시킬 파라미터의 수 (스텝 크기) 를 정한다

1: 각 레이어 별로 정해진 파라미터의 수를 감소시켜 예측 성능을 측정

2: 각 레이어의 예측 성능을 비교

3: 예측 성능의 변화량이 가장 낮은 레이어 별 파라미터의 개수를 합

4: 목표 파라미터 개수와 비교

5: 목표 파라미터보다 크면 변화량이 가장 낮은 레이어 별 파라미터의 개수를 기점으로 1번부터 위 과정을 반복, 그렇지 않다면 각 레이어 별 파라미터 개수와 예측 성능 반환

우선 목표 파라미터 개수를 정하고, 기존의 완전 연결 레이어 별 파라미터 개수를 기점으로 진행한다. 본 논문에서는 목표 파라미터의 개수를 1M 으로 정하였고, AlexNet 의 완전 연결 레이어 별 파라미터 개수(37M, 16M, 4M)를 기점으로 하였다. 목표 파라미터 개수가 입력으로 들어오면 첫번째 레이어부터 정해진 파라미터 개수(스텝 사이즈)만큼 감소시켜 예측 성능을 측정하고 두번째, 세번째 레이어도 이와 같은 방식으로 예측 성능을 측정한다 (line 1). 측정된 예측 성능 중 가장 작은 변화량을 갖는 파라미터의 개수를 모두 합하고 입력된 목표 파라미터 개수와 비교한다 (line 2, 3, 4). 목표 파라미터 개수와 비교했을 때 목표 파라미터보다 크다면 가장 작은 변화량을 갖는 레이어 별 파라미터 개수를 기점으로 다시 위 과정을 반복하고, 작거나 같다면 각 레이어 별 파라미터 개수와 예측 성능을 반환한다.

그리디 알고리즘을 사용하면 최적의 파라미터 개수와 근사한 값을 도출해 낼 수 있다. 하지만 모든 경우에 대해 항상 최적은 아니기 때문에 이 부분 대해 유념해야 한다.

나. 희소 행렬 벡터 곱 구현

pruning 방식을 이용하여 레이어 별 파라미터 수를 감소시킬 경우 각 레이어의 가중치 행렬은 밀집 행렬에서 희소 행렬로 변화한다. 일반적인 딥 러닝 프레임 워크의 경우에는 가중치 행렬이 밀집 행렬인 경우를 가정하고 구현

되어 있어, 희소 행렬의 이점을 제대로 활용하지 못한다. 따라서 본 논문에서는 희소 행렬 벡터 곱을 구현하여 파라미터 감소 기술이 실제 성능 향상으로 이어지도록 한다.

희소 행렬은 메모리 공간을 효율적으로 사용하기 위해 주로 COO(Coordinate) 방식과 CSR(Compressed Sparse Row) 방식을 사용한다. 이러한 방식들은 희소 행렬의 0이 많을수록 기존에 차지하던 메모리 사용량을 현저히 감소시킬 수 있으며 행렬의 연산 시간도 단축시킬 수 있는 장점이 있다.

COO 방식은 0이 아닌 값들의 행과 열의 인덱스(index)와 0이 아닌 값들을 3개의 배열로 저장하는 방식으로 행의 인덱스 배열, 열의 인덱스 배열, 00 이 아닌 값의 배열로 나타낼 수 있다.

CSR 방식도 0이 아닌 값들을 저장하는 방식은 COO 방식과 같으나 행의 인덱스를 저장하고 있는 배열을 압축한다는 점에서 차이가 있다. CSR 방식은 COO 방식에서 행의 인덱스 배열을 압축하였기 때문에 행의 인덱스에 접근이 빠르고 행렬-벡터 곱(matrix-vector multiplication) 연산을 수행하는데 이점이 있다.

본 논문에서는 밀집 행렬의 성질을 띠는 완전 연결 레이어의 가중치 행렬에 압축 기술을 적용하였고 희소 행렬의 형태로 변형된 가중치 행렬을 CSR 방식을 이용하여 완전 연결 레이어의 연산 속도를 향상시켰다.

Ⅲ. 실험 및 결과

본 장에서는 압축 기술이 연산 성능에 미치는 영향을 확인하기 위해, CPU와 GPU로 기존 완전 연결 레이어와 압축된 완전 연결 레이어의 연산 속도를 측정하여 비교하였다. 또한 제안한 시스템에서 압축 기술을 적용한 완전 연결 레이어가 뉴로모픽 하드웨어의 처리 속도에 맞추어 CPU 나 GPU 로 처리 가능한지 확인하기 위해 프레임 비율(frame rate)로 비교 분석하였다. 실험 환경은 다음과 같다.

완전 연결 레이어의 연산은 가중치 행렬과 입력 벡터의 행렬-벡터 곱 연산을 수행하므로 선형대수 라이브러리인 uBLAS[22]와 ViennaCL[23]을 이용하였으며, 언어는 C++

〈표 4〉

예측 성능	그리디 알고리즘 사용한 경우			알고리즘을 사용하지 않은 경우 (Manually optimized)		
	First Fully Connected Layer	Second Fully Connected Layer	Third Fully Connected Layer	First Fully Connected Layer	Second Fully Connected Layer	Third Fully Connected Layer
50 %	8 M	7 M	3 M	8 M	7 M	3 M
49 %	6 M	3 M	2 M	6 M	3 M	2 M
48 %	4 M	2 M	2 M	4 M	2 M	2 M
47 %	3 M	2 M	2 M	3 M	2 M	2 M

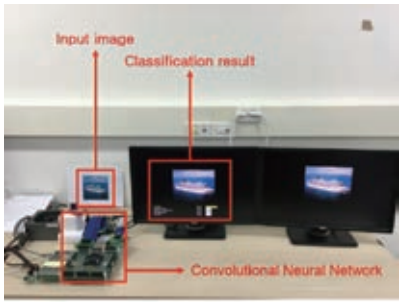
〈표 5〉

FC layer	Non-zero weight parameters	Non-zero weight parameter / Baseline FC layer weight parameter
Baseline FC layer	58,621,952	100 %
50 % FC layer	17,999,538	31 %
49 % FC layer	11,000,637	19 %
48 % FC layer	7,999,648	14 %
47 % FC layer	6,998,779	12 %

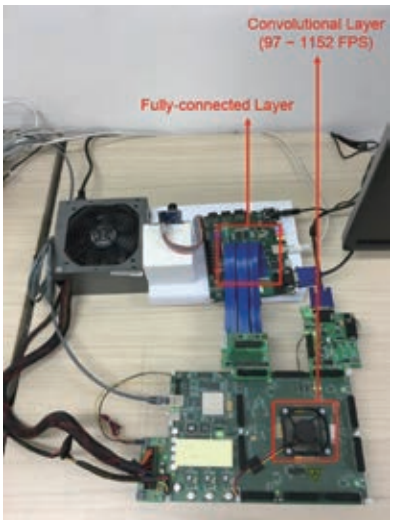
로 구현하였다. 실험에 사용된 임베디드 플랫폼 ODROID (ODROID-XU4)는 mobile CPU (Quad-core ARM Cortex-A15, Quad-core ARM Cortex-A7)와 mobile GPU(Mali-T628) 로 구성되어 있다.

실험에 사용된 완전 연결 레이어는 AlexNet 의 완전 연결 레이어이고, 압축 기술이 적용된 완전 연결 레이어는 3.2.1 절에 그리디 알고리즘을 통해 얻어낸 파라미터의 개수를 이용하였다. 〈표 4〉 에 완전 연결 레이어의 예측 성능이 50 ~ 47% 일 때, 레이어 별 최적의 가중치 파라미터 개수를 나타내었다. 〈표 4〉 에서 그리디 알고리즘을 사용한 경우와 사용하지 않은 경우를 비교하였을 때, 예측 성능에 따른 레이어 별 최적의 파라미터 개수가 같음을 확인하였다. 또한 완전 연결 레이어에 압축 기술을 적용했을 때, 가중치 파라미터의 감소량을 〈표 5〉 에서 확인할 수 있다.

〈그림 9〉 과 4.2 는 본 논문에서 제안한 시스템을 기반으로 컨볼루션 뉴럴 네트워크를 구현한 실험 환경이다. 〈그림 9〉 은 이미지가 카메라를 통해 실시간으로 컨볼루션 뉴럴 네트워크에 입력되면, 컨볼루션 뉴럴 네트워크는 입력된 이미지로부터 연산을 수행하여 결과를 출력하고, 이 중 가장 높은 확률 값을 갖는 5개의 객체가 모니터에 출력되는 실험이다. 〈그림 10〉 는 〈그림 9〉 에 컨볼



〈그림 9〉



〈그림 10〉

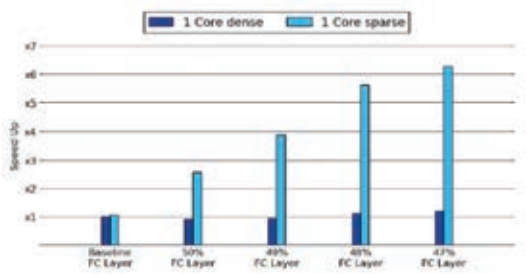
루션 뉴럴 네트워크 부분을 확대한 그림으로, ZedBoard (위)는 카메라에서 입력받은 이미지를 커넥터를 통해 VIRTEX-7(아래)로 전송한다. 뉴로모픽 하드웨어로 구현된 VIRTEX-7 은 이미지에 대해 컨볼루션 레이어의 연산을 수행하고, 연산된 결과를 다시 ZedBoard 로 전송한다. ZedBoard 는 입력 받은 결과에 대해 완전 연결 레이어의 연산을 수행하고, 이 중 가장 높은 값을 갖는 5개의 객체를 모니터에 출력한다.

〈표 6〉

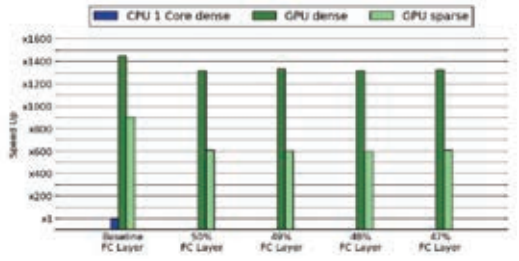
STEP	Uncompressed FC Layer	Compressed FC Layer
Step 1(데이터 변환)	138 ms	
Step 2(이미지 전송 및 수신)	2 ms	
Step 3(데이터 변환)	1 ms	
Step 4(데이터 저장)	5 ms	
Step 5(완전 연결 레이어 연산)	661 ms	271 ms

〈그림 10〉에서 컨볼루션 레이어를 구현한 뉴로모픽 하드웨어는 초 당 97 ~ 1152 장의 이미지를 처리할 수 있는 연산 성능을 가지고 있는데 압축된 완전 연결 레이어가 뉴로모픽 하드웨어의 연산 성능에 맞추어 처리할 수 있는지에 대해 아래의 실험을 통해 확인한다.

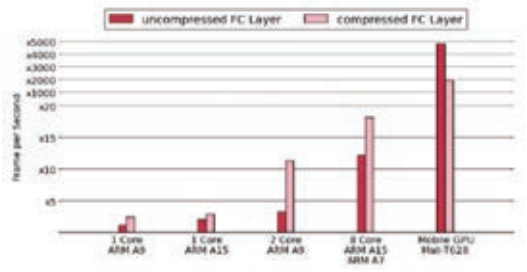
〈표 6〉 은 검증 시스템을 위해 구현된 코드로부터 각 단계별 소요 시간을 측정한 결과이다. 〈표 6〉 에서 Step 1 은 FPGA로 이미지 데이터를 전송하기 위해 프로토콜에 따라 데이터를 변환시키는 시간, Step 2 는 데이터를 FPGA로 전송하고 다시 수신하는 시간, Step 3 는 FPGA에서 수신한 데이터를 완전 연결 레이어에서 연산 가능하도록 데이터를 처리하는 시간, Step 4 는 완전 연결 레이어의 입력 노드에 데이터를 저장하는 시간, Step 5 는 완전 연결 레이어의 연산 시간을 의미한다.



〈그림 11〉



〈그림 12〉



〈그림 13〉

압축된 레이어의 연산 속도는 압축되지 않은 레이어의 연산 속도보다 약 2 ~ 3배 빠른 연산 속도를 가지며, Step 5 를 통해 확인할 수 있다. Step 1 은 다른 단계에 비해 높은 소요 시간을 갖는다. Step 1 은 1 Frame 을 처리할 때 소요되는 시간 중 15 ~ 30 % 비중을 차지하며 이는 전체적인 동작 속도를 저하시키는 원인이 된다. 따라서 입력 이미지의 포맷을 미리 변환시키면 Step 1 의 데이터 변환 과정을 수행할 필요가 없으므로 전체적인 동작 속도를 향상시킬 수 있다.

〈그림 11〉 ~ 〈그림 13〉에 Baseline FC layer 는 압축 기술이 적용되지 않은 완전 연결 레이어이고 50 ~ 47% FC layer 는 압축 기술이 적용된 완전 연결 레이어이다. Baseline FC layer 는 밀집 행렬, 50 ~ 47 % FC layer 는 희소 행렬의 성질을 띠고 있으며 Baseline FC layer 와 50 % FC layer 는 유사한 예측 성능을 갖는다. 또한 dense 연산은 일반적인 행렬 연산(2D-array) 방식을 의미하며, sparse 연산은 CSR(compressed sparse row) 방식을 의미한다.

〈그림 11〉 에 싱글 코어(1 Core)로 측정한 dense 와 sparse 의 연산 속도를 나타내었고, CPU 는 ARM-A15 를 사용하였다. dense 연산의 경우 Baseline FC layer 와 50 % FC layer 는 비슷한 연산 속도를 갖지만 sparse 연산의 경우 약 2배 빠른 연산 속도를 보였고, 압축률이 증가할수록 약 6배 빠른 연산 속도를 보였다.

〈그림 12〉 에 GPU 로 측정한 dense 와 sparse 의 연산 속도를 나타내었고, GPU 는 Mali T-628 사용하였다. CPU 1 Core 로 측정한 dense 연산의 시간을 기준으로 잡았을 때 〈그림 12〉 과 같은 연산 속도가 측정되었다. 50% FC layer의 sparse 연산을 보면 압축 기술이 적용되었음에도 불구하고 속도가 감소하였다. 그 이유는 GPU 에서 희소 행렬의 연산을 할 경우 스레드와 스레드 블록의 동기화 및 작업량 균등 분배, 메모리 접근, 데이터 재사용 등을 고려하여 희소 행렬 연산을 수행해야 하기 때문이다[24, 25].

〈표 7〉

FC Layer	ARM-A9		ARM-A15		ARM-A9		ARM-A15, A7		Mali-T628	
	1 Core		1 Core		2 Core		8 Core		GPU	
	dense	sparse	dense	sparse	dense	sparse	dense	sparse	dense	sparse
Baseline	1.1	0.7	3.2	3.8	2.0	1.2	12.2	7.9	4739.3	2958.5
50%	0.8	2.4	3.4	11.4	1.5	2.9	9.5	18.2	4310.3	1988.0
49%	0.9	3.9	4.7	18.4	1.7	1.7	10.4	25.8	4366.8	1972.3
48%	1.0	5.3	6.6	27.3	1.9	1.9	11.3	33.8	4310.3	1964.6
47%	1.1	6.1	7.9	29.5	2.1	2.1	12.4	38.0	4329.0	1984.1

〈그림 13〉 에 싱글 코어, 멀티 코어, GPU 로 측정한 연산 속도를 프레임 비율(frame rate)로 나타내었다. Uncompressed FC Layer 는 Baseline FC layer 가 사용되었고 dense 연산을 수행하였으며 compressed FC Layer 는 50 % FC layer 사용되었고 sparse 연산을 수행하였다. 〈표 7〉 에 실험에 사용된 프로세서들의 dense 와 sparse 의 연산 속도를 프레임 비율로 나타내었다.

전체적인 실험을 통해 다음과 같은 결과를 도출하였다. CPU 의 경우 압축 기술을 완전 연결 레이어에 적용하였을 때 dense 연산에 비해 약 2 ~ 5 배 정도 연산 속도가 향상되었다. 그러나 프레임 비율은 최대 18 FPS 를 갖기 때문에 뉴로모픽 하드웨어의 FPS(최소 97 FPS)에 못 미치므로 CPU로 완전 연결 레이어를 구현하기에는 부족한 성능을 보였다. GPU의 경우 CPU와 반대로 dense 연산에 비해 약 2배 정도 연산 속도가 감소하였으나 이는 희소 행렬에 최적화된 연산 기법을 사용하면 더욱 빠른 연산 속도를 얻을 수 있다. 그러나 CPU와 비교했을 때 약 600 배 빠른 연산 속도를 갖고, 프레임 비율은 약 2000 FPS 이므로 GPU로 완전 연결 레이어를 구현하기에는 충분한 성능을 보였다.

Ⅳ. 결 론

본 논문에서는 폰 노이만 컴퓨팅 방식을 기반으로 한 모바일 장치에서 대규모 심층 신경망을 빠르고 효율적으로 구현하기 위한 시스템을 제안하였다. 이에 따라 연산량이 많고 파라미터의 수가 적은 컨볼루션 레이어는 뉴로모픽 아키텍처를 갖는 뉴로모픽 하드웨어를 이용하고, 연산량이 적고 파라미터의 수가 많은 완전 연결 레이어는 압축 기술을 적용하여 소프트웨어를 이용하는 시스템을 구현하였다.

이 시스템을 검증하기 위해 FPGA와 Programmable SoC 플랫폼을 활용하여 실제로 구현하였다. 이러한 시스템에서 압축 기술이 적용된 완전 연결 레이어를 소프트웨어로 구현하였을 때 뉴로모픽 하드웨어의 동작 속도에 맞추어 처리 가능한지 확인하기 위해 CPU와 GPU로 성능을 평가하였다.

CPU 의 경우, 압축 기술이 적용되었을 때 연산 속도는 향상되었으나 뉴로모픽 하드웨어와 같이 동작하기에는 부족한 성능을 보였고 GPU 의 경우, 충분히 동작할 수 있는 성능을 보였다. 그러나 GPU 는 압축 기술이 적용되었을 때 연산 속도가 감소하였는데 이는 희소 행렬에 최적화된 연산 기법을 사용하면 더욱 빠른 연산 속도를 얻을 수 있다.

정해진 파라미터 개수에 따라 최대 예측 성능을 갖는 파라미터 개수를 구하기 위해 그리디 알고리즘을 이용하였고, 알고리즘을 사용한 결과와 사용하지 않는 결과를 비교하였을 때 같은 파라미터 개수를 갖는 것으로 확인하였다.

또한 압축 기술이 적용된 완전 연결 레이어를 구현할 때 CSR 방식을 이용하였고 실험을 통해 연산 속도가 향상되는 것을 확인할 수 있었다. 압축 되지 않은 완전 연결 레이어와 압축된 완전 연결 레이어를 다양한 방법으로 평가하였을 때 압축 기술이 연산 성능에 미치는 영향을 실험 결과를 통해 확인할 수 있었다.

본 논문에서 제안한 시스템은 컨볼루션 뉴럴 네트워크가 이용될 수 있는 모든 분야에 접목 가능하며, 신경망의 동작 효율성을 높일 수 있다. 다수의 이미지 중 특정 이미지만 분류하는 이미지 분류 작업, 이미지 내에 사람의 얼굴 혹은 특정 객체를 인식하는 객체 인식 작업, 자연어 혹은 영상 데이터를 인지하여 시각 장애인에게 음성으로 정보를 제공하는 웨어러블 디바이스 등 다양한 분야에 적용될 수 있다. 또한 이 시스템은 모바일 장치에만 국한되지 않고 운전자 보조시스템(ADAS) 혹은 자율 주행 자동차, 의료 영상을 분석하는 의료 분야, 무인 매장 시스템, 폐쇄회로 TV(CCTV) 기반의 지능형 감시 시스템 등의 분야에도 접목될 수 있다.

이 논문은 인천대학교 2015년도 자체연구비 지원에 의하여 연구되었음

참 고 문 헌

[1] LeCun, Y., Bengio, Y., & Hinton, G., (2015), “Deep learning”, Nature, 521(7553), pp. 436–444.

[2] Krizhevsky, A., Sutskever, I., and Hinton, G. E., 2012, “Imagenet classification with deep convolutional neural networks”, Advances in neural information processing systems, pp.1097–1105.

[3] Alsharif, O., and Pineau, J., 2013, “End-to-end text recognition with hybrid HMM maxout models”, arXiv preprint arXiv:1310.1811.

[4] Sharif R. A., Azizpour, H., Sullivan, J., and Carlsson, S., 2014, “CNN features off-the-shelf: an astounding baseline for recognition”, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 806–813.

[5] Taigman, Y., Yang, M., Ranzato, M. A., and Wolf, L., 2014, “Deepface: Closing the gap to human-level performance in face verification”, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp.1701–1708.

[6] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D, (1989), “Backpropagation applied to handwritten zip code recognition”, Neural computation, 1(4), pp. 541–551.

[7] Schemmel, J., Briiderle, D., Griibl, A., Hock, M., Meier, K., and Millner, S., 2010, “A wafer-scale neuromorphic hardware system for large-scale neural modeling”, Proceedings of 2010 IEEE International Symposium on Circuits and Systems, pp. 1947–1950.

[8] Kim, Y. D., Park, E. H., Yoo, S.J., Choi, T. L., Yang, L., and Shin, D. J., 2015, “Compression of deep convolutional neural networks for fast and low power mobile applications”, arXiv preprint arXiv:1511.06530.

[9] Wu, J., Leng, C., Wang, Y., Hu, Q., and Cheng, J., 2015, “Quantized Convolutional Neural Networks for Mobile Devices”,

arXiv preprint arXiv:1512.06473.

[10] Maas, A. L., Hannun, A. Y., and Ng, A. Y., 2013, "Rectifier nonlinearities improve neural network acoustic models", In Proc. ICML, 30(1).

[11] Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M. A., and Dally, W. J., 2016, "EIE: efficient inference engine on compressed deep neural network", arXiv preprint arXiv:1602.01528.

[12] Merolla, P. A. et al., (2014), "A million spiking-neuron integrated circuit with a scalable communication network and interface", Science, 345(6197), pp. 668-673.

[13] Chung, J. Y., and Shin, T. H., and Kang, Y. S., 2015, "INsight: A Neuromorphic Computing System for Evaluation of Large Neural Networks", arXiv preprint arXiv:1508.01008.

[14] Chung, J. Y., and Shin, T. H., 2016, "Simplifying deep neural networks for neuromorphic architectures", Proceedings of the 53rd Annual Design Automation Conference. ACM, pp.126.

[15] Misha, D., Babak, S., Laurent, D., Marcaurelio, R., and Nando, D. F., 2013, "Predicting parameters in deep learning", In Advances in Neural Information Processing Systems, pp. 2148-2156.

[16] Wenlin, C., James, T. W., Stephen, T., Kilian, Q. W., Yixin, C., 2015, "Compressing convolutional neural networks", arXiv preprint arXiv:1506.04449.

[17] Emily, L. D., Wojciech, Z., Joan, B., Yann, L., and Rob, F., 2014, "Exploiting linear structure within convolutional networks for ecient evaluation", In Advances in Neural Information Processing Systems, pp. 1269-1277.

[18] Han, S., Jepp, P., John, T., and William, D., 2015, "Learning both weights and connections for efficient neural network", Advances in Neural Information Processing Systems, pp. 1135-1143.

[19] http://caffe.berkeleyvision.org/

[20] Tara, N. S., Brian, K., Vikas, S., Ebru, A., and Bhuvana, R., 2013, "Low-rank matrix factorization for deep neural network training with high-dimensional output targets", In Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, pp. 6655-6659.

[21] Jian, X., Jinyu, L., and Yifan G., 2013, "Restructuring of deep neural network acoustic models with singular value decomposition", In INTERSPEECH, pp. 2365-2369.

[22] http://www.boost.org/

[23] http://viennacl.sourceforge.net/

[24] 조용연, 배덕호, 김상욱, (2013), "GPU 기반의 외적을 통한 희소행렬 곱셈방안", 정보과학회논문지, 19(10), pp. 524-528.

[25] Baskaran, M. M., and Bordawekar, R., (2008), "Optimizing Sparse Matrix-Vector Multiplication on GPUs", IBM Research Report, RC24704 (W0812-047).

신태환

• 2015년 인천대학교 전자공학과 학사
• 2017년 인천대학교 전자공학과 석사

〈관심분야〉
SoC, FPGA, Deep Neural Network

정재용

• 2006년 8월 연세대학교 전자공학과 학사 졸업
• 2008년 12월 텍사스오스틴대학교 전자공학과 석사
• 2011년 5월 텍사스오스틴대학교 전자공학과 박사
• 2011년 6월~2013년 8월 Synopsys Inc, USA
• 2013년 8월~현재 인천대학교 조교수

〈관심분야〉
뉴로모픽 시스템, 딥 러닝, VLSI CAD