JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지(J. Korea Inst. Inf. Commun. Eng.) Vol. 21, No. 6 : 1143~1148 Jun. 2017

CUDA를 이용한 Convolutional Neural Network의 효율적인 구현

기철민[1*] · 조태훈[2]

Efficient Implementation of Convolutional Neural Network Using CUDA

Cheol-Min Ki[1*] · Tai-Hoon Cho[2]

[1]Department of Computer Engineering, Korea University of Technology and Education, Cheonan 31253, Korea
[2*]School of Computer Science and Engineering, Korea University of Technology and Education, Cheonan 31253, Korea

요 약

현재 인공지능과 딥 러닝이 사회적인 이슈로 떠오르고 있는 추세이며, 다양한 분야에 이 기술들을 응용하고 있다. 인공지능 분야의 여러 알고리즘들 중에서 각광받는 방법 중 하나는 Convolutional Neural Network이다. Convolutional Neural Network를 적은 양의 데이터에서 이용하거나, Layer의 구조가 복잡하지 않은 경우에는 학습시간이 길지 않아 속도에 크게 신경 쓰지 않아도 되지만, 학습 데이터의 크기가 크고, Layer의 구조가 복잡할수록 학습시간이 상당히 오래 걸린다. 이로 인해 GPU를 이용하여 병렬처리를 하는 방법을 많이 사용하는데, 본 논문에서는 CUDA를 이용한 Convolutional Neural Network를 구현하였으며, 비교에 사용한 Framework/Program들 보다 학습 속도가 빨라지고 큰 데이터를 학습 시키는데 더욱 효율적으로 진행하도록 한다.

ABSTRACT

Currently, Artificial Intelligence and Deep Learning are rising as hot social issues, and these technologies are applied to various fields. A good method among the various algorithms in Artificial Intelligence is Convolutional Neural Networks. Convolutional Neural Network is a form that adds Convolution Layers to Multi Layer Neural Network. If you use Convolutional Neural Networks for small amount of data, or if the structure of layers is not complicated, you don't have to pay attention to speed. But the learning should take long time when the size of the learning data is large and the structure of layers is complicated. In these cases, GPU-based parallel processing is frequently needed. In this paper, we developed Convolutional Neural Networks using CUDA, and show that its learning is faster and more efficient than learning using some other frameworks or programs.

키워드 : Convolutional Neural Network, 기계 학습, 병렬처리, CUDA, 고속 처리

Key word : Convolutional Neural Network, Machine Learning, Parallel Processing, CUDA, Fast Speed

Received 26 May 2017, **Revised** 28 May 2017, **Accepted** 01 June 2017

* **Corresponding Author** Cheol-Min Ki(E-mail:kigoon@koreatech.ac.kr, Tel:+82-31-761-5467)
Department of Computer Engineering, Korea University of Technology and Education, Cheonan, 31253, Korea

Open Access https://doi.org/10.6109/jkiice.2017.21.6.1143 print ISSN: 2234-4772 online ISSN: 2288-4165

ⓒThis is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License(http://creativecommons.org/li-censes/ by-nc/3.0/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.
Copyright Ⓒ The Korea Institute of Information and Communication Engineering.

Ⅰ. 서 론

최근 전 세계적으로 가장 활발하고 이슈가 되고 있는 분야는 인공지능(AI)과 딥 러닝(Deep Learning)이다. 이는 4차 산업혁명이라는 말과 함께 여러 분야에 다양하게 적용되며, 이 기술들을 응용하고 있다.

딥 러닝의 다양한 방법들 중 영상처리 분야에서 많이 사용되는 방법 중 하나는 Convolutional Neural Network(CNN)[1,2]이다. CNN은 일반적인 Multi-layer Neural Network에 Convolution Feature를 추출하는 Convolution Layer를 추가한 형태이다.

CNN은 CPU의 성능이 향상되면서 상당히 각광을 받기 시작했는데, 많은 양의 데이터를 학습시키거나 Neural Network의 Layer 구조가 복잡한 경우에는 학습시간이 상당히 오래 걸린다. 이로 인해 CNN의 연산을 GPGPU(General Purpose Graphic Processing)를 이용하여 학습하는 방법들이 나타났고, 학습시간을 상당히 줄일 수 있었다.

본 논문에서는 GPGPU 중 하나인 CUDA(Compute Unified Device Architecture)를 이용한 CNN의 구현을 진행하였으며, 실험 및 속도 비교는 기존에 존재하는 Deep Learning Framework / Program 과의 비교를 진행하였다.

Ⅱ. 선행 연구

본 논문에서는 CUDA를 이용한 CNN의 구현을 진행하였으므로, CUDA와 CNN에 대한 간략한 설명을 제시한다.

2.1. Convolutional Neural Network

CNN은 다양한 컴퓨터 비전 분야에서 많은 연구가 활발히 진행 중인 인공신경망의 한 종류이다. CNN은 일반적으로 몇 개의 층으로 이루어져 있으며, 기본적으로 Convolution Layer, Pooling Layer, Feedforward Layer(Fully-connected Layer)의 3가지의 다른 층을 가지고 있고, 경우에 따라 여러 Layer가 추가되기도 한다. Convolution Layer는 입력한 데이터에 Convolution 연산을 적용하여 Convolution Feature를 추출하는 층이다. 이는 유의미한 자질을 추출하는 층이라고 할 수 있다.

Pooling Layer는 Convolution Layer에서 추출한 Convolution Feature의 양을 줄이는 과정인데, 이는 이미지의 특성상 픽셀의 개수가 너무 많고, 이로 인해 연산 횟수가 상당히 많아지기 때문에 연산 속도의 향상을 위해 Feature의 개수를 줄여나가는 과정이며, 이는 다시 말해 Sub-sampling하는 과정이라고 할 수 있다.

Feedforward Layer는 일반적인 Multi-Layer Neural Network의 구조와 동일하며, Layer 간의 Neuron들의 연결은 모두 연결되어 Fully-connected Layer라고도 한다. 이는 Convolution Layer와 Pooling Layer의 반복 구조에서 나온 Feature들을 이용하여 분류를 진행할 때 사용된다.

CNN은 위에서 설명한 Convolution Layer와 Pooling Layer를 번갈아 사용하여 Convolution Feature를 추출하고, 최종적으로 추출된 Feature를 Feedforward Layer를 이용하여 분류를 진행한다. 학습 과정에서 모든 Layer에 대한 추측을 진행할 때는 Forward Propagation 과정을 각 Layer의 Weight와 Bias 등의 파라미터를 갱신할 때는 Backpropagation 과정이라고 한다. 아래 그림 1은 필기체 문자 데이터인 MNIST Dataset[3]을 인식하기 위한 CNN의 구조이다.

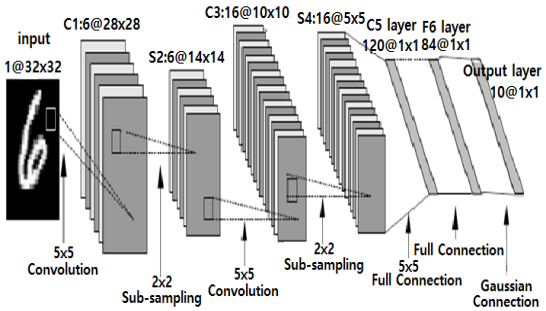

Fig. 1 Structure of CNN for MNIST Dataset test

그림에서 나타나는 과정은 위에서 설명한 것처럼 Convolution Layer와 Pooling Layer의 반복으로 이루어지며, 최종적으로 나타난 Feature들이 Feedforward Layer와 이어지는 구조로 나타난다. 위의 구조의 과정을 거쳐 나타난 Output값이 분류된 영상의 라벨 값이며, Ground Truth와 비교하여 정확도를 판별한다.

2.2. CUDA

GPGPU는 일반적으로 컴퓨터 그래픽스를 위한 계산만 맡았던 GPU(Graphic Processing Unit)를 일반 범용 목적의 병렬 계산 프로세서로 응용하면서 나타났다[4]. 초기 GPU의 기능은 매우 제한적이었고, 단지 그래픽스 파이프라인의 특정 부분을 가속시키기 위해서만 사용되었다. 이 후 프로그램 가능한, 보다 유연한 형태의 GPU로 넘어오면서 3D 그래픽 분야를 위해서만 사용되던 GPU에 대한 새로운 접근이 시도되었는데, 이는 GPU가 행렬과 벡터, 부동 소수점 형 연산에 높은 계산 성능을 보이는 것에서 기인한 것으로, 그래픽 연산 뿐 아니라 일반 컴퓨팅 영역에서도 GPU를 활용하고자 하는 체제를 말한다. GPGPU는 병렬 처리에 적합한 연산에서 뛰어난 연산 속도를 낼 수 있다는 이점이 있어서 상당히 좋지만, GPU 병렬 프로세싱을 위한 프로그래밍을 작성해야 하며, GPGPU 프로그램 작성을 위해 다양한 방법들이 존재한다. 본 논문에서는 GPGPU 프로그래밍 방법 중 하나인 CUDA를 이용한다.

CUDA는 GPU에서 수행하는 알고리즘을 C 프로그래밍 언어를 비롯한 산업 표준 언어를 사용하여 작성할 수 있도록 하는 GPGPU 기술이다. CUDA의 처리는 그림 2와 같으며, 메인 메모리를 GPU 메모리로 복사 한 후 CPU가 GPU에 프로세스를 지시한 후 GPU 각 코어에서 병렬로 프로세스를 수행하며, GPU 메모리로부터 결과물을 메인 메모리에 복사하는 흐름을 가진다.

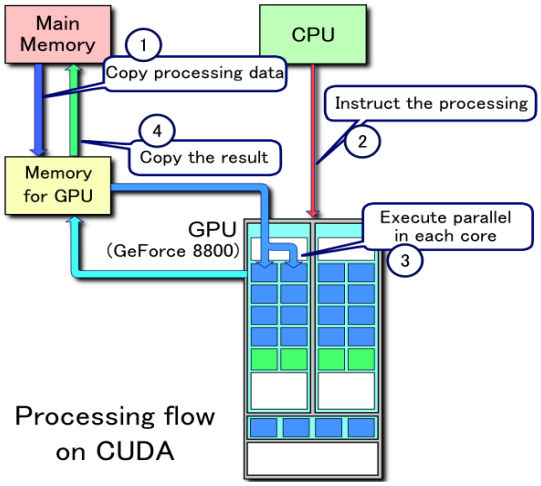

Fig. 2 Processing flow on CUDA [5]

Ⅲ. 구현 방법

3.1. Forward Propagation

본 논문에서는 CUDA를 이용하여 Convolutional Neural Network의 구현을 진행했다. 일반적인 CNN에서 사용되는 Layer들의 CUDA 프로그래밍을 진행하였으며[6], 세부적으로 Convolution Layer, Pooling Layer, Fully-connected Layer의 Backpropagation Algorithm을 구현하였다[7].

먼저 Forward Propagation은 이전 Layer의 Neuron과 Weight가 곱해진 값들의 합과 Bias와의 합이 Activation Function을 통과한 결과가 Neuron의 값으로 결정된다. 여기서 이런 Layer의 모든 Neuron이 다음 Layer의 Neuron과 연결되어 있다면 Fully-connected의 Forward Propagation이 되고, 일정한 Mask 영역에서 Convolution 연산을 한 값이 다음 Neuron과 연결된다면 Convolution Layer의 Forward Propagation이 된다. Fully-connected Layer에서는 모든 Neuron이 연결되어 있기 때문에 각각의 Neuron에 대응되는 Weight가 존재하지만 Convolution Layer에서는 Convolution Mask에 대응되는 Weight만 존재하면 되므로, Convolution Layer의 Weight의 개수는 확연히 줄어든다. 위에서 설명한 Forward Propagation의 식은 아래 식 (1)과 같다.

$$z_{x,y}^{l} = w^{l} * \sigma(z_{x,y}^{l-1}) + b_{x,y}^{l}, \quad a_{x,y}^{l} = \sigma(z_{x,y}^{l}) \tag{1}$$

위 식에서 z는 Layer l 에서의 Neuron, σ는 Activation Function, b는 bias, w는 weight이며, a는 Activation Function의 결과이다.

Fully-connected Layer의 Forward Propagation에서는 (이전 Layer의 Neuron 개수 + 1) × (현재 Layer의 Neuron 개수) 만큼의 반복 연산이 진행된다. 이를 CUDA 프로그래밍하면, (현재 Layer의 Neuron 개수)가 CUDA Block의 개수가 되고, (이전 Layer의 Neuron 개수 + 1)개는 Thread의 개수가 된다. 위에서 설명한 연산에 Activation Function 에 맞는 연산을 적용시켜 현재 Layer의 Neuron 값으로 확정하며, 본 논문에서는 Activation Function을 Hyperbolic Tangent와 Softmax 두 가지를 구현했다.

다음 Convolution Layer의 Forward Propagation은 총

6개의 반복문 (현재 Layer의 Feature Map 개수, 현재 Layer Feature Map의 Width, Height, 이전 Layer의 Feature Map 개수, Convolution Kernel의 Width, Height) 이 사용되며, CUDA에서는 6개의 반복문 모두를 Thread와 Block으로 병렬 구성할 수 없기 때문에, 병렬화 하는 부분은 현재 Layer의 Feature Map 개수와 Width, Height 총 3가지를 사용하고, Map 개수는 Block, Width와 Height는 각각 Thread x, y로 설정하며, 내부에 존재하는 3개의 반복문은 Thread 별로 동일하게 동작하도록 한다. 이 후 Activation Function을 적용하는데, Convolution Layer에서는 Hyperbolic Tangent 만을 구현했다.

마지막으로 Max-pooling Layer의 Forward Propagation은 총 5개의 반복문 (현재 Layer의 Feature Map 개수, Width, Height, Pooling Mask의 Width, Height) 이 사용되며, 위의 Convolution Layer와 동일하게 Feature Map 개수는 Block, Map의 Width와 Height는 각각 Thread x, y로 설정하며, Max 값을 추출할 마스크의 반복문은 각 Thread 마다 반복하도록 한다. 이 후 Backpropagation을 위해 Max-pooling 이전 Layer의 Max 값과 index를 저장해둔다.

3.2. Backpropagation

한 영상에 대해 Forward Propagation이 진행되면 마지막으로 출력되는 Output 값과 미리 설정한 Ground Truth에 대한 에러 값을 구하면서 Backpropagation을 진행한다. 최종적인 Output의 개수는 상황에 따라 다르고, 보통 개수가 많지 않으므로 CPU에서 연산하는 것과 GPU에서 연산하는 것의 차이는 크지 않지만, GPU와 CPU 사이의 데이터 전송시간이 상당하기 때문에, 이 부분도 CUDA로 구현하여 데이터 전송시간을 줄인다[8].

Backpropagation은 현재 Layer와 이전 Layer 사이의 변화량 Delta 값을 계산할 때, Weight와 Delta 값을 곱한 값의 누적 값을 Activation Function을 미분한 함수를 적용한 값과 Learning Rate의 곱을 Weight Update의 값으로 이용하게 된다. 이로 인해 Backpropagation의 경우 Delta를 계산하는 커널 함수와 Weight Update 함수 두 개가 동일한 구조의 반복문으로 같은 연산을 진행하게 된다. 그러므로 CUDA 프로그래밍을 할 때 Delta 계산 함수와 Weight Update 함수를 나눠서 구성해야하며, 두 함수가 하나의 커널 함수로 구성되어 있다면, 필요한 Delta가 존재하지 않을 수 있고 결과가 다르게 나올 수 있다. Backpropagation의 과정을 수식으로 표현하면 아래 식 (2)와 같다.

$$\delta^{l}_{x,y} = \delta^{l+1}_{x,y} * w^{l+1}_{-x,-y}\sigma'(z^{l}_{x,y}), \qquad (2)$$
$$\frac{\partial C}{\partial w^{l}_{x,y}} = \delta^{l}_{x,y} * \sigma(z^{l-1}_{-x,-y})$$

첫 번째 수식에서 나오는 δ는 변화량 Delta이며, 두 번째 수식의 결과를 이용하여 Weight Update를 진행한다.

먼저 Fully-connected Layer의 Backpropagation의 Delta 계산은 (현재 Layer의 Neuron 개수) X (다음 Layer의 Neuron 개수) 만큼의 반복 연산이 진행된다. 그러므로 2차원의 Block과 Thread를 두고, Index를 주어 두 반복문을 모두 병렬로 계산하도록 한다. 이후 Weight Update 또한 동일한 방식으로 반복문을 병렬로 계산하며, Delta와 Learning Rate를 곱한 값을 기존에 가지고 있던 Weight 값에서 빼주는 연산을 진행한다.

Convolution Layer의 Backpropagaiton의 Delta 계산은 Forward Propagation과 동일하게 6개의 반복문 (현재 Layer의 Feature Map 개수, Weight, Height, 이전 Layer의 Map 개수, Convolution Kernel의 Width, Height)로 구성되어 있으며, 6개 모두를 병렬화 할 수 없으므로, 현재 Layer의 Map 개수, Width, Height를 각각 Block, Thread x, y로 구성하여 Delta의 계산을 진행하고, Weight Update 또한 동일하게 병렬화를 진행한다.

Max-pooling Layer의 Backpropagation은 계산이나 Weight Update를 할 필요는 없고, Sub-sampling만 진행했던 Layer이기 때문에, Forward Propagation에서 연산을 하면서 저장했던 Mask 사이즈 내의 최댓값이 나타난 Index 자리에 Delta 값만 할당해주면 된다. 이를 다시 말하면 Up-sampling만 진행하면 된다고 볼 수 있다. 그러므로 현재 Layer의 Neuron 수만큼 반복이 진행되며, Delta 값을 할당해준다. 최종적으로 Input Layer와 연결 된 Layer까지 Delta를 계산하고, Weight Update가 끝나면 하나의 Input 영상에 대한 Backpropagation 과정이 끝난다[9,10].

Ⅳ. 실험 결과

본 논문에서는 위에서 설명한 방법으로 구현한 CNN과 기존에 존재하는 Deep Learning Framework/Program 들 간의 속도 비교를 진행하였으며, Input 영상을 모아서 Delta를 계산하고 Weight Update를 하는 방식인 Batch 방식을 사용하지 않고, SGD(Stochastic Gradient Descent) 방식을 사용하여 매 Input 영상 마다 Weight Update를 진행하는 방식으로 테스트를 진행하였다. 초기 Learning Rate는 0.001로 설정하고, 매 Epoch 마다 0.993씩 Learning Rate를 줄이는 방식을 사용하였다. 입력 영상은 CPU 버전의 Elastic Distortion을 이용하여 Image Processing을 진행했으며, Deep Learning Framework 중 하나인 Caffe Framework는 기본적으로 Elastic Distortion이 없기 때문에 Image Processing을 하지 않은 경우만 구현한 CNN과의 비교를 진행하였다.

테스트의 실험 환경은 Intel Xeon CPU E3-1225 V5 (3.30GHz, 4-Core), 16GB RAM, GPU는 NVIDIA Geforce GTX 750ti의 시스템에서 진행하였고, OS는 Microsoft Windows 10 64bit, IDE는 Microsoft Visual Studio 2013을 이용하였으며, 실험은 MNIST Dataset의 Training Set을 1 Epoch (60000장) 학습시키는데 걸리는 시간을 측정했다. 테스트에 사용된 CNN의 구조는 아래 그림 3과 같다.

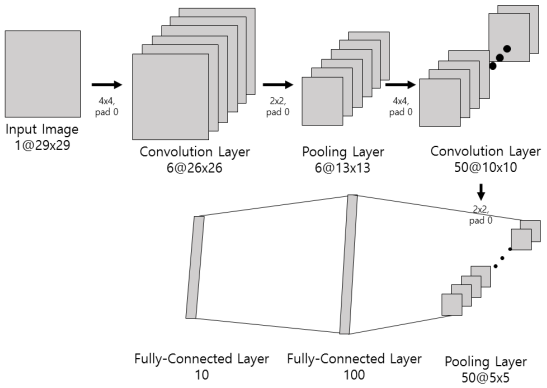

Fig. 3 Structure of CNN used in the test

위에서 나타난 그림 3의 구조로 각 프로그램들의 속도 비교 결과는 다음 표 1과 같다.

Table. 1 Training Time (1 Epoch)

Kinds	Time (s)	
	Distortion On	Distortion Off
Implemented CNN	67.9	50.7
Darknet[11]	72.8	56.7
Net (CPU-ver)[12]	180.6	128.7
CUDA-CNN[13]	130.6	115.9
Caffe[14,15]	-	78.6

표 1에서 사용된 Darknet은 C와 CUDA로 작성된 Open Source Neural Network Framework이며, 주로 Real-time Object Detection에 사용된다. Net program은 C로 작성된 CNN 코드이며, 본 논문에서는 Net program을 기반으로 GPU Version의 구현을 진행하였다. CUDA-CNN은 CUDA를 이용해 CNN의 속도 향상을 진행한 프로그램이며, Caffe는 Berkely AI Research에 의해 개발된 Deep Learning Framework로, 다른 Framework/Program에 비해 다양한 Layer와 Activation Function을 사용할 수 있다.

Ⅴ. 결 론

학습을 1 Epoch 진행하고, Training Set에 대한 정확도는 거의 비슷한 결과가 나오는데, 여기서 차이가 발생할 수 있는 이유는 자료 형에 따라 계산되는 값들의 정확도가 달리지기 때문이며, 위 실험 결과에 따르면 본 논문에서 구현한 방법은 다른 Program이나 Framework들과 비슷하거나, 빠른 속도를 보여주었다. 조금 더 속도를 올리기 위해서는 현재 반복문 안쪽에서 Index 접근 오류를 방지하기 위해 Atomic 연산을 진행하는데, 이를 Shared Memory를 이용하여 CUDA 커널 함수를 최적화 하는 방법이 있으며, 이는 Atomic 연산에서 사용되는 Synchronize가 없기 때문에 속도 향상에 도움이 될 것이다.

또한 더 좋은 하드웨어를 사용하거나, 앞에서 설명한 Input 영상을 모아서 계산하는 Batch 방식을 이용하고, CUDA에서 연산이 상당히 빠른 Matrix Multiplication을 이용한다면 더욱 좋은 성능의 CNN을 구현할 수 있을 것으로 보인다.

차후 개선할 문제는 위에서 설명한 속도적인 측면과, 여러 가지 Activation Function과 다용도로 사용할 수

있는 다양한 Layer들의 범용적인 구현이 필요할 것으로 보인다.

REFERENCES

[1] Y. LeCun, L. Bottou, Y. Bengio, and P.Haffner, “Gradient-Based Learning Applied to Document Recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.

[2] Patrice Y. Simard, Dave Steinkraus, John Platt, “Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis,” *Proceedings of the ICDAR*, pp. 958-962, 2003.

[3] Yann Lecun’s Homepage. The MNIST DATABASE of Handwritten Digits [Internet], Available: https://yann.lecun.com/exdb/mnist.

[4] Wikipedia. Description of GPGPU [Internet]. Available: https://ko.wikipedia. org/wiki/GPGPU.

[5] Wikipedia. CUDA Processing Flow [Internet]. Available: https://upload.wikimedia.org/wikipedia/commons/5/59/CUDA_processing_flow_%28En%29.PNG.

[6] J. Long, E. Shelhamer, T. Darrell, “Fully Convolutional Networks for Semantic Segmentation,” *Proceeding of the CVPR*, pp. 3431-3440, 2015.

[7] Y. Huang, K. Li, G. Wang, M. Cao, P. Li, Y. Zhang. (2015, May). Recognition of convolutional neural network based on CUDA technology. *arXiv preprint arXiv:1506.00074* [Online]. Available: https://arxiv.org/abs/1506.00074.

[8] Dan C. Ciresan, U. Meier, J. Masci, Luca M. gambardella, J, Schmidhuber. (2011, February). High-Performance Neural Networks for Visual Object Classification. *Arxiv preprint arXiv:1102.0183* [Online]. Available: https://arxiv.org/abs/1102.0183.

[9] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, “Overfeat: Integrated recognition, localization and detection using convolutional networks,“ *Proceedings of the ICLR*, 2014.

[10] Kumar Chellapila, Sidd Puri, and Patrice Simard, “High performance convolutional neural network for document processing”, *International Workshop on Frontiers in Handwriting Recognition*, 2006.

[11] Joseph Chet Redmon’s Homepage. Darknet Framework [Internet]. Available: https://pjreddie.com/darknet

[12] Dan Ciresan’s Homepage. Net CPU Version Program [Internet]. Available: https://people.idsia.ch/~ciresan/index.htm.

[13] University of Science and Technology of China. CUDA-CNN program [Internet]. Available: https://github.com/zhxfl/ CUDA-CNN.

[14] Berkeley AI Research. Caffe framework [Internet]. Available: http://caffe.berkeleyvision.org.

[15] Y.jia, E. Shelhamer, J. Donahue, S. Karayev, J.Long, R. Girshick, S. Guadarrama, and T. Darrell. (2014, June). Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093* [Online]. Available: https://arxiv.org/abs/1408.5093.

기철민(Cheol-Min Ki)

2016. 2 : 한국기술교육대학교 컴퓨터공학부 졸업
2016. 3 ~ 현재 : 한국기술교육대학교 컴퓨터공학과 재학
※관심분야 : 영상처리, 컴퓨터비젼

조태훈(Tai-Hoon Cho)

1991. 5 : Virginia Tech 전기공학과 공학박사
1992. 3 ~ 1998.2 : LG산전연구소 수석연구원
1998. 3 ~ 현재 : 한국기술교육대학교 컴퓨터공학부 교수
※관심분야 : 영상처리, 컴퓨터비젼