석사학위청구논문
2017학년도

실시간 얼굴 검출을 위한 CPU-FPGA 구조 연구

Cascade CNN with CPU-FPGA Architecture for Real-time Face Detection

광운대학교 대학원

전자통신과학과

남 광 민

실시간 얼굴 검출을 위한 CPU-FPGA 구조 연구

Cascade CNN with CPU-FPGA Architecture
for Real-time Face Detection

광운대학교 대학원

전자통신과학과

남 광 민

실시간 얼굴 검출을 위한
CPU-FPGA 구조 연구

Cascade CNN with CPU-FPGA Architecture
for Real-time Face Detection

지도교수 정 용 진

이 논문을 공학 석사학위 청구논문으로 제출함

2017년 12월 일

광운대학교 대학원

전자통신과학과

남 광 민

남광민의 공학 석사 학위논문을 인준함

심사위원장 _____ 인

심사위원 _____ 인

심사위원 _____ 인

광운대학교 대학원

2017년 12월 일

감사의 글

광운대학교 참빛관 812호에서 캡스톤과 KWIX 작품을 준비하던 인연으로 RTA 연구실에 몸 담게 되고, 다시 참빛관 810호에서의 2년까지 흘러 어느덧 졸업을 바라보는 시기가 되었습니다. RTA에서 공부하며 다른 곳에서는 겪을 수 없는 수많은 경험들을 얻었습니다. 교수님의 열정적인 지도, 많은 선후배님들의 교류와 도움, 수많은 교내외 사람들과의 인연들은 2년 동안 지식뿐만 아니라 마음까지 자라게 해주는 소중한 기회였습니다.

특히, 매일 많은 시간을 할애하시어 크고 작은 일에 모두 신경 써서 지도해주시고, 보이지 않는 곳에서도 도움주신 교수님께 가장 감사드립니다. 평소 해주신 많은 조언들은 여러 상황에서 제가 올바르게 방향을 잡을 수 있는 길잡이가 될 것입니다. 연구실을 떠나서도 이곳에서 얻었던 배움 잊지 않겠습니다.

그리고 힘든 상황에도 항상 미소를 잃지 않고 상냥하게 후배들을 이끌어 주었던 상진이형, 연구실의 중심을 지키며 있어준 원섭이형, 말하지 않아도 후배들을 잘 챙겨준 정환이형, 묵묵히 주변을 챙겨준 지원이, 궂은일에도 솔선수범하며 따뜻하게 대해준 성우에게 감사의 말씀을 드립니다.

걱정 없이 편안하게 공부할 수 있도록 언제나 변함없이 물심양면 지원해주시고, 진심어린 조언으로 인생의 빛을 밝혀주신 부모님께, 무뚝뚝한 대답에도 언제나 웃어주는 누나에게도 이 기회를 빌려 감사의 말씀을 전합니다.

2017년 12월 RTA 연구실
남 광 민 드림

국 문 요 약

얼굴 검출에는 다양한 포즈, 빛의 세기, 얼굴이 가려지는 현상 등의 많은 변수가 존재하므로, 높은 성능의 검출 시스템이 요구된다. 이에 영상 분류에 뛰어난 Convolutional Neural Network (CNN)이 적절하나, CNN의 많은 연산은 고성능 하드웨어 자원을 필요로한다. 그러나 얼굴 검출을 위한 소형, 모바일 시스템의 개발에는 저가의 저전력 환경이 필수적이고, 이를 위해 본 논문에서는 소형의 FPGA를 타겟으로, 얼굴 검출에 적절한 3-Stage Cascade CNN 구조를 기반으로하는 CPU-FPGA 통합 시스템을 설계 구현한다. 가속을 위해 알고리즘 단계에서 Adaptive Region of Interest (ROI)를 적용했으며, Adaptive ROI는 이전 프레임에 검출된 얼굴 영역 정보를 활용하여 CNN이 동작해야할 횟수를 줄인다. CNN 연산 자체를 가속하기 위해서는 FPGA Accelerator를 이용한다. 가속기는 Bottleneck에 해당하는 Convolution 연산의 가속을 위해 FPGA 상에 다수의 FeatureMap을 한번에 읽어오고, Multiply-Accumulate (MAC) 연산을 병렬로 수행한다. 본 시스템은 Terasic사의 DE1-SoC 보드에서 ARM Cortex A-9와 Cyclone V FPGA를 이용하여 구현되었으며, HD (1280x720)급 입력영상에 대해 30FPS로 실시간 동작하였다. CPU-FPGA 통합 시스템은 CPU만을 이용한 시스템 대비 8.5배의 전력 효율성을 보였다.

ABSTRACT

Since there are many variables such as various poses, illuminations and occlusions in a face detection problem, a high performance detection system is required. Although CNN is excellent in image classification, CNN operatioin requires high-performance hardware resources. But low cost low power environments are essential for small and mobile systems. So in this paper, the CPU-FPGA integrated system is designed based on 3-stage cascade CNN architecture using small size FPGA. Adaptive Region of Interest (ROI) is applied to reduce the number of CNN operations using face information of the previous frame. We use a Field Programmable Gate Array(FPGA) to accelerate the CNN computations. The accelerator reads multiple featuremap at once on the FPGA and performs a Multiply-Accumulate (MAC) operation in parallel for convolution operation. The system is implemented on Altera Cyclone V FPGA in which ARM Cortex A-9 and on-chip SRAM are embedded. The system runs at 30FPS with HD resolution input images. The CPU-FPGA integrated system showed 8.5 times of the power efficiency compared to systems using CPU only.

차 례

그 림 차 례

표 차 례

제 1 장 서 론

카메라, 영상 산업이 발전함에 따라 영상 내부의 정보를 이용한 연구와 어플리케이션이 증가하고 있으며, 특히 인간을 대상으로 하는 경우 더욱 활발하다. 그 중 얼굴 인식의 경우 활용도가 매우 높아 다양한 연구 방향이 제시되었는데, 현재 Deep Learning 기법, 특히 CNN 기법이 우수한 성능을 발휘하고 있다. 그림 1은 딥러닝 사용 이전의 다양한 얼굴 검출 알고리즘과 CNN을 포함한 많은 종류의 딥러닝 알고리즘을 이용한 얼굴 검출의 검출률을 나타낸다. True Positive Rate 최상위 성적은 모두 딥러닝 기법을 활용한 사례이다.

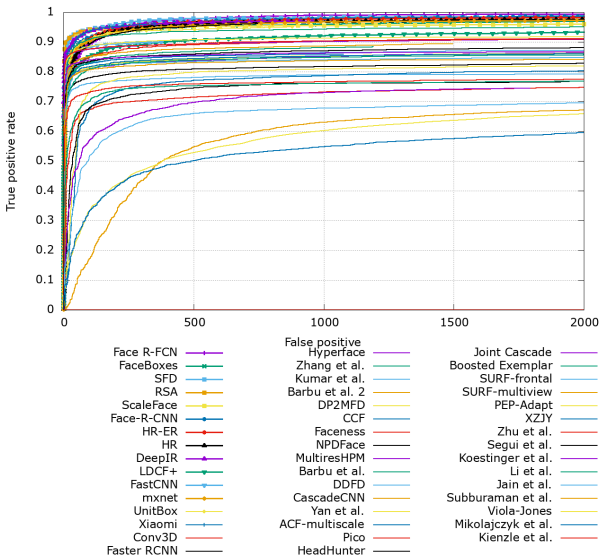


그림 1. Face Detection Dataset and Benchmark에 적용한 얼굴 검출 성능 결과 그래프

Fig. 1. Performance Cureves for Face Detection Dataset and Benchmark (FDDB)

이렇듯 얼굴 검출에 높은 성능을 기대할 수 있으므로 금융 본인 인증 시스템, CCTV를 이용한 용의자 탐색, 스마트 디바이스의 본인 인증, 다양한 엔터테인먼트 어플리케이션에의 활용 등 많은 분야에 접목시킬 수 있다. 다만 CNN 자체가 요구하는 연산량과 메모리 크기다는 단점이 있다. 그림 2는 CNN 연산의 핵심인 Convolutional Layer의 연산 방법을 나타낸다. 입출력 FeatureMap, 영상의 Width, Height, Weight의 크기가 모두 개별적인 인덱스가 되므로 연산량이 기하급수적으로 늘어난다. 많은 연산량의 빠른 처리를 위해 대부분의 경우 PC 환경에서 CNN 구조 연구[3]-[10]가 개발되는데, CCTV, 차량, 모바일 제품 등 실제품은 고가의 연산기를 장착하기에 어려움이 있고, 저가의 저전력 시스템이 요구된다. 본 연구는 위 조건에 입각하여, 임베디드용으로 개발된 소형 FPGA인 Intel FPGA사의 Cyclone V를 타겟으로 개발되었다. 임베디드 환경에서 많은 연산량을 고속 처리하기 위한 첫 번째 방법으로 CNN의 구조 설계를 3장 1절에서, 두 번째 방법으로 소프트웨어 알고리즘의 측면에서 이용되는 Adaptive ROI의 설명은 3장 2절에서, 마지막으로 하드웨어를 이용한 가속기에 대한 내용은 본론의 3장 3절에서 설명한다.

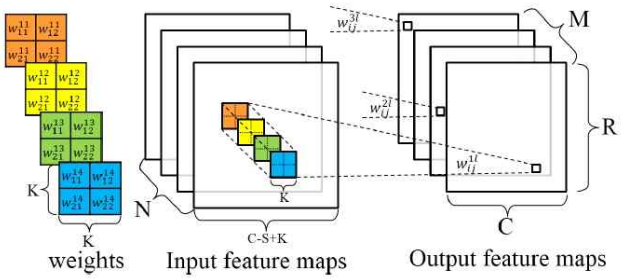


그림 2. CNN의 Convolutional Layer 구조 예시

Fig. 2. Example of Convolutional Layer in CNN

제 2 장 관련 연구

본 논문에 관련된 연구는 크게 Cascade CNN 구조에 관한 연구, CNN의 FPGA 가속에 관한 연구로 나뉜다.

2.1. Cascade CNN 구조

먼저 Cascade CNN 구조에 관한 연구로, Cascade 얼굴 검출기는 Viola and Jones [1]가 Haar-Like features와 AdaBoost 분류기를 사용하면서 제안되었다. 이후로 Mathias *et al.* [2]-[4]는 변형가능한 얼굴 검출기 모델을 제안하여 높은 성능을 보였다. 그러나 과도한 연산량과 학습량을 필요로하는 단점이 있었다. 최근 Convolutional Neural Network (CNN)의 적용으로 영상 분류[5], 얼굴 인식[6]에서 큰 발전을 보였다. Li *et al.*[7]은 Cascade CNN 구조를 얼굴 검출에 적용하는 시도를 했지만, Bounding Box Calibration으로 인한 과도한 연산이 필요하다는 단점이 있었다. 개선을 위해 Zhang *et al.*[8]은 Multitask CNN을 이용하여 Multiview 얼굴 검출의 정확도를 높였지만, 초기 검출 window의 한계로 인해 성능에 한계가 있었다. 본 논문에서는 얼굴 분류와 Calibration을 통합한 형태의 CNN, Adaptive ROI를 이용하여 성능과 속도 문제를 개선한다.

2.2. CNN 구조의 FPGA 설계

CNN의 발전과 더불어 CNN의 가속에 관한 연구도 왕성하게 진행되고 있다. 최근 FPGA, GPU, ASIC 등 다양한 형태의 가속기 설계[9]-[11]가 제안되어 성능을 높이고 있다. 이러한 연구 중 특히 FPGA를 이용한 가속기는

성능의 우수성, 높은 에너지 효율, 빠른 개발 기간, 수정의 용이성 때문에 더욱 주목받고 있다[9], [12]-[16]. 본 논문에서는 Multiple FeatureMap Load, Local Memory 재사용을 통해 Memory Bandwidth를 최적화하고, 공통적으로 사용되는 Kernel 크기를 기준으로 Computation Engine을 설계하여 Layer 종류에 관계없이 연산을 수행토록 한다.

2.3. CNN의 세부 Layer 구조

최근 CNN이 높은 성능과 함께 주목 받는 이유는 첫 째, Rectified Linear Unit(ReLU)의 등장으로 활성화 함수에서 나타나는 Gradient Vanishing 문제가 상당부분 해결되었다. 둘 째, 빅데이터의 발전으로 신경망 학습에 사용할 데이터가 풍부해져서 과적합(Overfitting)을 피하기 용이해졌다. 마지막으로, Drop out을 활용한 Regularization으로 대용량의 입력 데이터를 효과적으로 분류, 검출하는 것이 가능해졌기 때문이다. CNN을 이루는 Layer 종류는 크게 Convolution Layer, Pooling Layer, Fully Connected Layer, ReLU로 구분되며 구현 방식에 따라 조금씩의 차이는 존재한다.

특징 추출에 이용되는 Convolutional Layer는 입력 FeatureMap과 Weight를 Convolution 연산하여 출력 FeatureMap을 생성하는 역할을 한다. Convolutional Layer는 FeatureMap, Weight, Bias의 크기와 값에 따라 특징을 추출하며, 입출력 채널의 수에 따라 Weight의 개수가 달라진다. 연산은 아래 Figure 5와 같이 입력 FeatureMap과 Weight의 곱셈과 덧셈 누적을 반복한다. Convolution 연산 후에는 활성화 함수를 적용한다. 본 디자인은 더욱 높은 성능을 위해 ReLU에서 발전된 형태인 PReLU를 활성화 함수로 사용했다. PReLU는 Figure 6에서 확인할 수 있다. ReLU와는 달리 함수의 입력 값이 0보다 작거나 같더라도 0으로 변환시키지 않고, 학습으로 얻은

특정 Parameter를 곱한다.

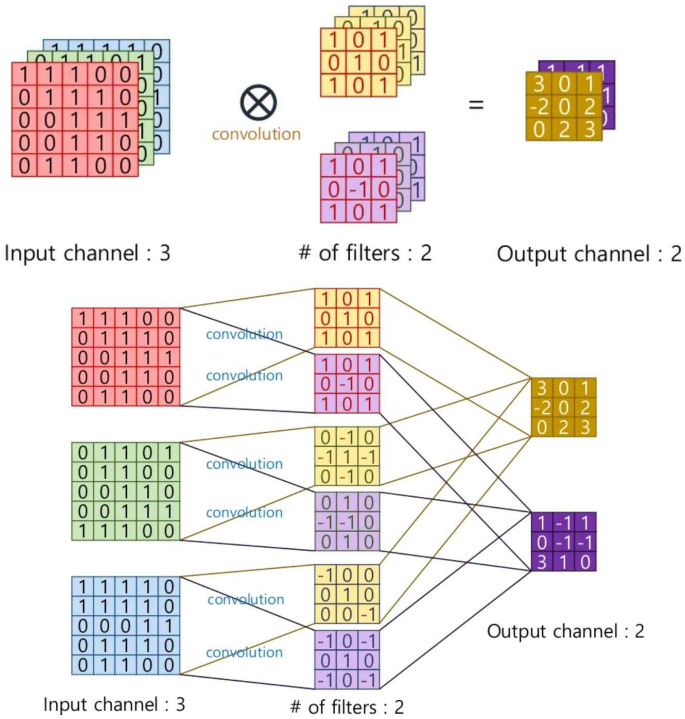


그림 3. Convolutional Layer 동작 과정

Fig. 3. Convolutional Layer Process

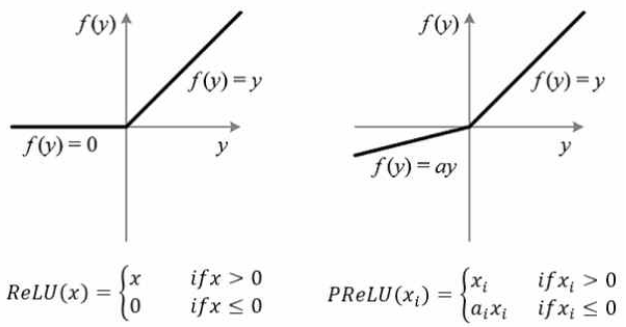


그림 4. ReLU와 PReLU

Fig. 4. ReLU and PReLU

Pooling Layer는 Max Pooling 기법을 이용한다. Max Pooling은 Figure 7과 같이 마스크에 해당하는 위치의 숫자끼리 비교하여 가장 큰 숫자를 남기는 기법이다. 이로써 주변 픽셀 중 특징을 가장 잘 보전하는 큰 숫자를 잃지 않으며 처리해야 할 데이터의 크기를 대폭 감소시킬 수 있다. 필터 크기와 Stride 옵션값을 조정하여 Pooling 처리의 단계를 선택한다.

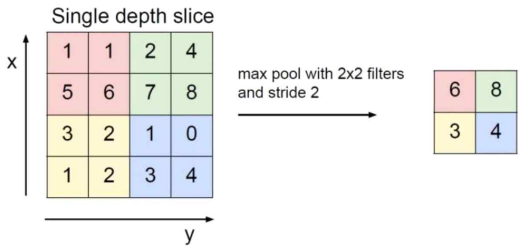


그림 5. Pooling Layer의 동작 과정

Fig. 5. Pooling Layer Process

Fully Connected Layer는 이전 Layer에서 추출된 특징을 이용하여 최종 분류의 역할을 수행한다. 계산 가능한 모든 노드들을 이용해 연산을 수행한다. 분류될 클래스의 개수에 따라 Confidence를 가지게 되는데 편차를 줄이기 위해 Softmax 기법을 이용해 전체 클래스의 확률을 더했을 때, 1이 나오도록 값을 조절한다.

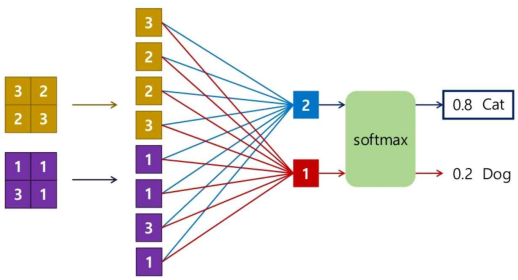


그림 6. Fully Connected Layer와 Softmax의 동작 과정

Fig. 6. Fully Connected Layer and Softmax Process

2.4. OpenCL

Open Computing Language (OpenCL)은 개방형 범용 병렬 컴퓨팅 프레임워크이다. CPU, GPU, DSP, FPGA 등 연산장치가 혼재되어있는 이종플랫폼을 사용자가 쉽게 사용할 수 있게 도와주는 역할을 한다. 딥러닝, 물리 시뮬레이션, 주가분석 등 다양한 분야에서 사용되고 있는데 작업 시간과 실행 시간을 대폭 줄여주기 때문이다. 흔히 비교되는 CUDA와 성능상의 차이는 크지 않지만, CUDA가 NVIDIA 플랫폼에서만 사용할 수 있다면, OpenCL은 Intel, AMD 등 각종 플랫폼에서 사용 가능하다는 장점이 있다. 본 디자인에서 활용되는 OpenCL의 용도는 소프트웨어 작업에 익숙한 사용자가 하드웨어 코딩 작업을 쉽게 처리할 수 있게 도와주는 것이다. HDL을 이용하여 하드웨어를 디자인, 구현할 경우 최적화와 성능에서 아직까지는 OpenCL의 사용을 앞서지만, 작업 속도 면에서는 OpenCL이 월등히 앞선다. 다양한 API가 계발되고 OpenCL의 전체적인 성능이 발전하면서, HDL에 미숙하거나 빠른 시간에 하드웨어 구현 작업을 마치고 싶은 사용자에게 OpenCL은 매우 유용하다. 또한 개발보드, FPGA에 대한 이해도가 떨어지더라도 API를 이용하여 작업을 완료하므로 빠른 업무 적응이 가능하다.

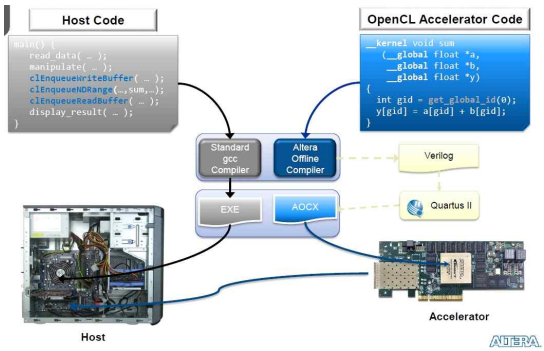


그림. 7. OpenCL 이용 모델

Fig. 7. OpenCL Model

제 3 장 Face Detection System

본 논문의 시스템은 그림 8과 같이 간략화된다. 입력 영상으로 저장된 동영상 혹은 웹캠을 통해 얻어지는 영상 프레임을 사용한다. 시스템 구현에 있어 입력 영상 크기는 HD (1280x720)급 해상도로 지정한다. 입력된 영상은 기본적으로 DE1-SoC의 ARM Core를 사용하는 소프트웨어 알고리즘을 거친다. 영상의 입출력, 다양한 영상의 전, 후처리는 ARM Core를 이용하여 실행되며, Bottleneck에 해당되는 CNN 부분은 FPGA 내에 설계, 구현된 하드웨어 가속기를 거쳐서 빠르게 실행된다. 얼굴 검출이 완료된 영상은 얼굴 검출 영역을 표시한 상태로 특정 메모리 주소에 저장되어 VGA 포트를 통해 모니터로 출력된다.

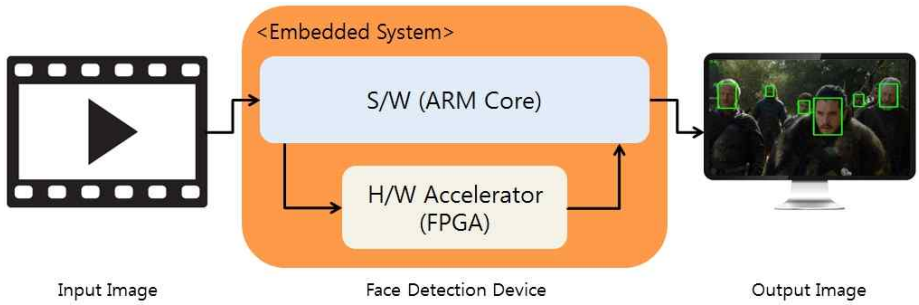


그림. 8. 제안하는 시스템의 개요

Fig. 8. Overview of Proposed System

3.1 Cascade CNN

전체적인 Pipeline은 그림 9와 같다. 먼저 입력된 영상은 다양한 크기로 변형되어 Image Pyramid를 생성한다. Image Pyramid는 총 세 단계로 구성되는 Cascade CNN에 최초 입력되는 영상이다.

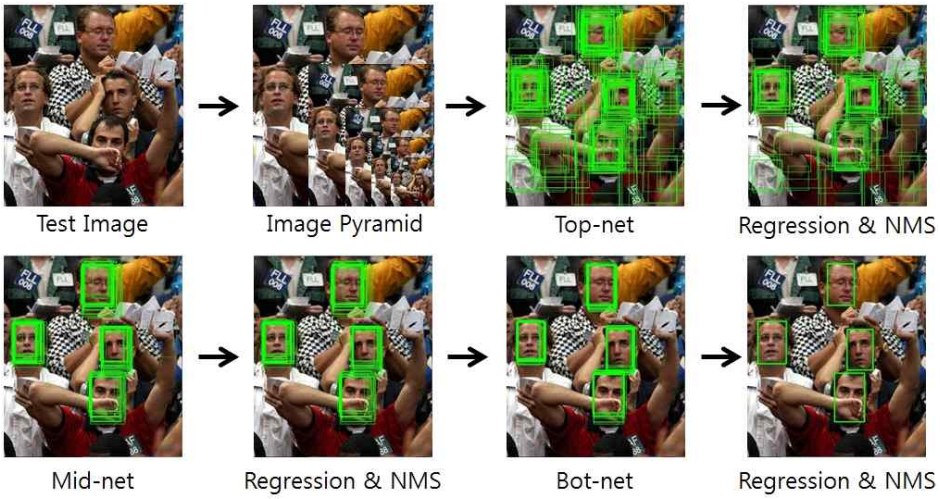


그림. 9. 제안하는 Cascade CNN의 파이프라인

Fig. 9. Pipeline of Proposed Cascade CNN

첫 번째 Stage인 Top-net은 가장 적은 연산량을 가지는 CNN으로, 개략적이지만 빠르게 입력 영상을 훑어 얼굴의 후보군을 검출한다. Top-net을 거치면서 얼굴 후보군의 대다수를 걸러낼 수 있어 연산량이 많은 Mid-net, Bot-net의 동작 횟수를 줄인다. 이는 실시간 동작에 필수적이다. Top-net을 거치면서 얼굴 후보군의 Bounding Box와 해당 Box의 Confidence, Regression 정보를 얻는다. 얻어진 Bounding Box 중 겹치는 영역이 많을 경우 Non-Maximum Suppression (NMS)을 통해 하나의 Bounding Box에

흡수되고, Regression으로 Box 좌표 위치가 보정된다.

두 번째 Stage인 Mid-net은 Top-net의 결과로 얻어진 얼굴 후보 영역을 입력으로 받아, 다시 한번 더욱 정교하게 얼굴을 검출한다. Top-net보다 많은 연산량으로 실제 얼굴이 위치하는 영역에 근접한 결과의 Bounding Box를 계산한다. 마찬가지로 NMS를 거쳐 Box의 수를 줄이고, Regression으로 Box 좌표 위치가 보정된다.

마지막 Stage인 Bot-net은 Mid-net보다 더욱 커진 크기의 CNN으로 가장 정교하게 최종 얼굴 검출을 마친다. 다만 수행 시간이 가장 길기 때문에 Bot-net의 동작을 줄이는 것이 실시간 동작에 큰 영향을 끼친다.

마지막 Stage인 Bot-net은 Mid-net보다 더욱 커진 크기의 CNN으로 가장 정교하게 최종 얼굴 검출을 마친다. 다만 수행 시간이 가장 길기 때문에 Bot-net의 동작을 줄이는 것이 실시간 동작에 큰 영향을 끼친다.

CNN의 전체 구조는 그림 10과 같다. Kernel의 크기는 빠른 연산을 위해 3x3, 2x2로 설계되었다.

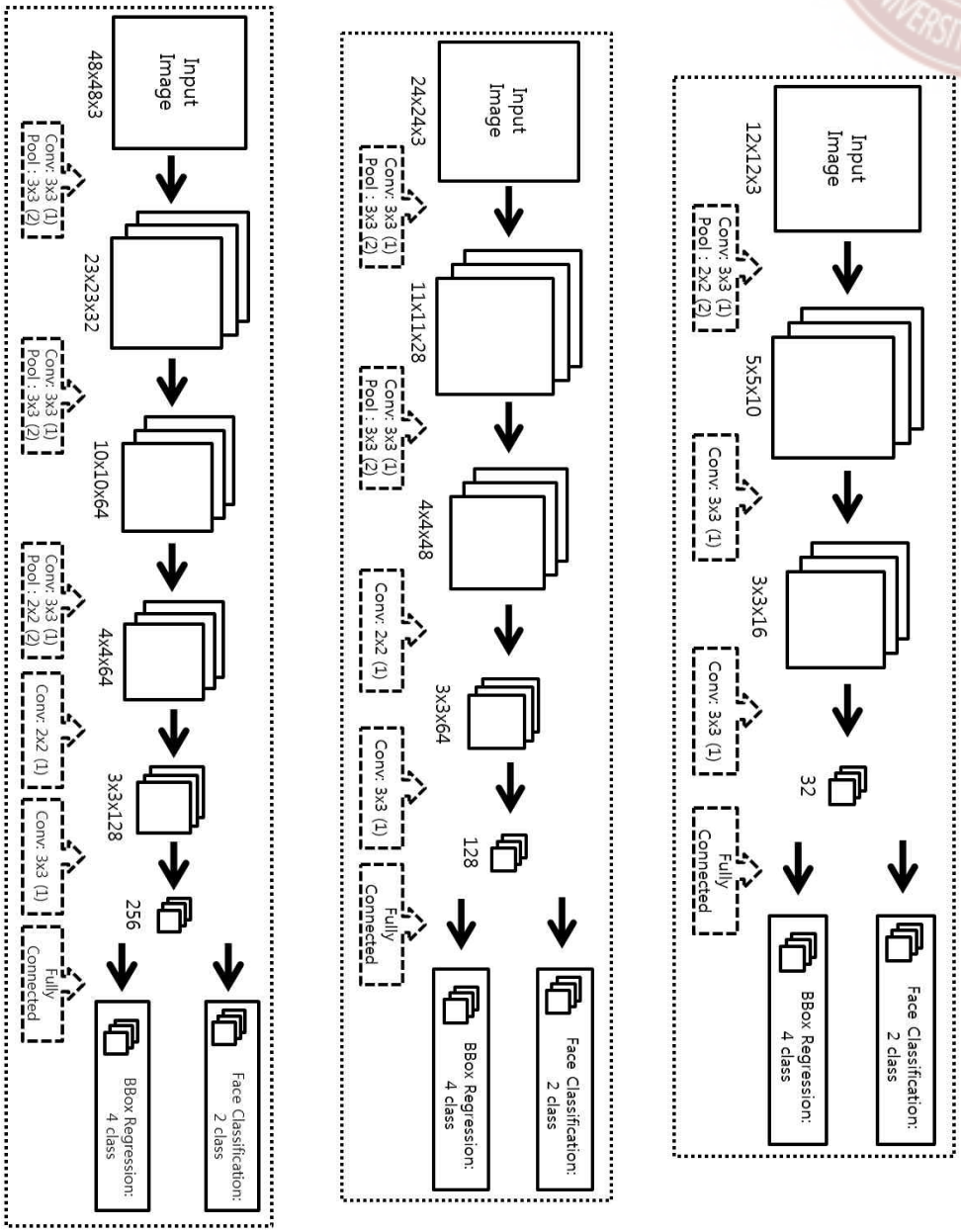


그림. 10. Cascade CNN의 구조 :
(a) Top-net, (b) Mid-net, (c) Bot-net
Fig. 10. Cascade CNN Architecture

Convolutional Layer를 거친 후, ReLU 보다 정교한 PReLU[20]를 활성 함수로 사용한다. PReLU는 식 (1)의 형태로 작동한다.

속도를 계산한다. 계산한 속도와 거리를 이용해 식 (3)과 같이 Time-To-Collision(TTC)를 계산하고 TTC가 3초 이내일 경우, 운전자에게 추돌 경보를 알려준다.

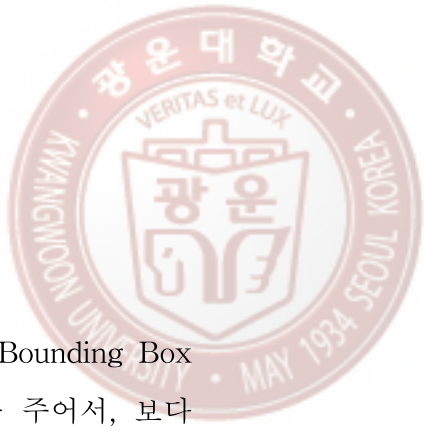
$$ReLU(x) = \begin{cases} x & if\, x > 0 \\ 0 & if\, x \leq 0 \end{cases} \qquad PReLU(x_i) = \begin{cases} x_i & if\, x_i > 0 \\ a_i x_i & if\, x_i \leq 0 \end{cases} \quad (1)$$

CNN의 학습을 통해 얼굴 이진 분류, Bounding Box Regression의 성능을 향상시킨다.

얼굴 분류는 얼굴인지 아닌지에 해당하는 두 개의 클래스로 객체를 분류하는 문제로 정의된다. 입력되는 각 영상에 대해 식(2)로 표현되는 Cross-Entropy Loss 방식으로 학습한다.

$$L_i^{\det} = -\left(y_i^{\det} \log\left(p_i\right) + \left(1 - y_i^{\det}\right)\left(1 - \log(p_i)\right)\right) \quad (2)$$

p_i는 입력 영상 내에 얼굴이 존재할 확률로, 네트워크로 계산된 결과이다. $y_i^{\det}$는 영상이 실제로 얼굴인지 아닌지 그 결과를 0 또는 1로 가지는 참값으로, 학습의 정답표 역할을 한다.



Bounding Box Regression은 얼굴 검출의 후보군에 속하는 Bounding Box에 Offset (Bounding Box의 Left, Top, Width, Height 값)을 주어서, 보다 정확한 얼굴 검출 위치를 찾기위해 사용된다. Offset을 찾는 과정은 Regression 문제로 정의되고, 식 (3)의 Euclidean Loss 방식이 학습에 사용된다.

$$L_i^{\text{box}} = \left\| \hat{y}_i^{\text{box}} - y_i^{\text{box}} \right\|_2^2 \tag{3}$$

$\widehat{y_i^{box}}$는 네트워크의 결과로 얻어진 Bounding Box의 정보이고, y_iBox는 Bounding Box가 위치해야할 위치의 좌표이다.

3.2 Adaptive ROI

Adaptive ROI는 이전 프레임에서 검출된 얼굴 위치를 참조하여, 현재 프레임에서 CNN에 입력할 영상 ROI의 수를 감소시키기위해 적용되는 기법이다. 이전 프레임의 얼굴 검출 여부와 현재 상황에 따라 각 경우를 나누고, ROI 영역을 설정하는데 그 규칙은 표 1과 같다.

표 1. Adaptive ROI 규칙

Table 1. Adaptive ROI Rule

	First Frame	Object Detected		Object Not Detected	
Case	0	1	2	3	4
Full-ROI	O	X	X	O	X
Half-ROI	X	X	O	X	X
Obj-ROI	X	O	O	X	X
Period	-	1-4	5	1	2-3

표 2. Adaptive ROI 적용 예시

Table 2. Adaptive ROI Example

	0	1	2	3	4	5	6	7	8	9	10	11
Object Detected	-	O	O	O	O	O	X	X	X	X	O	O
ROI	Full	Obj	Obj	Obj	Obj	Half	Full	No	No	Full	Obj	Obj
Case0	0	1	1	1	1	2	3	4	4	3	1	1

이전 프레임에서 얼굴이 검출됐을 경우, 얼굴 주변 영역만을 검출하고, 새로운 얼굴 객체의 유입이나 오검출에 대비하여 5번의 프레임 입력을 기준으로 입력영상의 절반에 해당하는 Half-ROI로 설정하여 검출한다. Half-ROI의 위치는 x축을 기준으로 3분할된 모양으로 나누어 변경된다. 이전 프레임에서 얼굴이 검출되지 않았을 경우, 입력 영상 전체를 대상으로 얼굴을 검출하지만, 매 프레임마다 얼굴을 검출하는 것은 계산시간을 많이 요구하므로 3번의 프레임 입력을 기준으로 검출을 수행한다. 표 2의 Adaptive ROI 동작 예시는 이전 프레임에서의 얼굴 검출여부와 프레임 입력 기준으로 Adaptive ROI가 어떻게 작동하는지 나타낸다.

Adaptive ROI를 사용할 경우 그림 11의 흰색 네모 박스 내부의 영역 만큼만을 CNN의 입력으로 사용한다. 영상 전체를 입력으로 사용하는 경우보다 매우 적은 양임을 확인할 수 있다. 그 효율성은 그림 12의 그래프에서 비교된다.

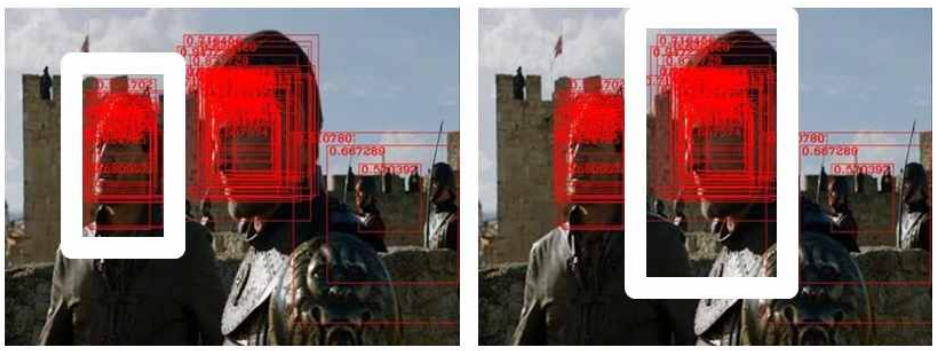


그림 11. Adaptive ROI가 적용된 영상 예시

Fig. 11. Sample Image for Applying Adaptive ROI

그림 12. Adaptive ROI 인자 변화에 따른 검출률과 연산량 변화

Fig. 12. Detection Rate & Operation Ratio depends on Adaptive ROI's Parameter

3.3 FPGA Accelerator

CNN 자체의 연산 속도를 높이기 위해서는 FPGA 상에 하드웨어 가속기가 이용된다. 그림 13은 CNN 하드웨어 가속기의 Block Diagram으로 가속기의 작동 방식를 보여준다. 가속기는 연산을 위해 Global Memory로부터 입력 FeatureMap과 Parameter를 Local Memory로 읽어들인다. 현재 사용되는 CNN, Layer 종류에 알맞은 연산 (Convolution, Pooling, Fully Connected)을 하도록 컨트롤러를 통해 제어하며, 가속기 내부의 주소 연산기를 통해서 Global Memory의 어느 주소에서 데이터를 읽어오고, 쓸지 판단한다. 현재 입력 FeatureMap이 모두 연산될 때까지 가속기는 같은 동작을 반복하며, Global Memory에 출력 FeatureMap을 저장한다.

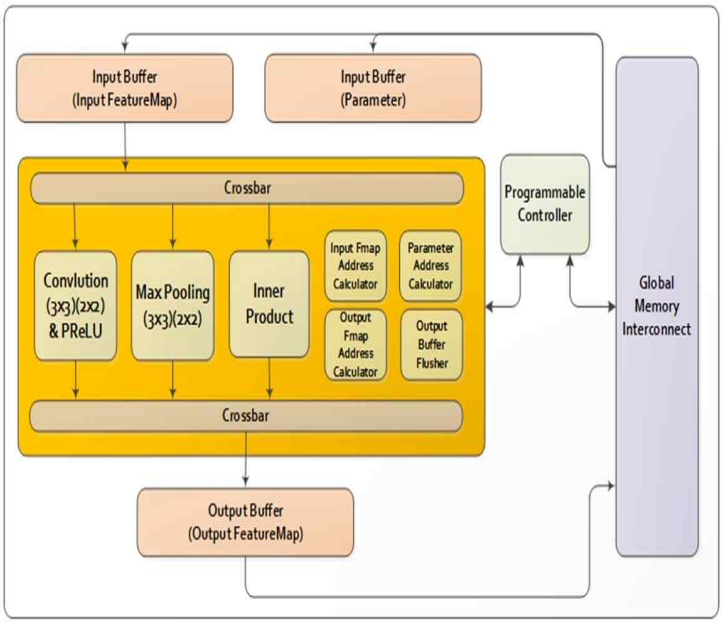


그림 13. CNN 가속기의 블록 다이어그램

Fig. 13. CNN Accelerator Block Diagram

3.3.1 Computation & Memory Optimization

하드웨어 가속기의 연산 성능을 증가시키기위해 모든 종류의 CNN 입력 FeatureMap들을 공통적으로 저장할 수 있는 버퍼 크기를 찾는다. 그림 14는 Top, Mid, Bot-net의 입력 FeatureMap이 최대로 입력되도록 3, 6, 12, 24, 48 크기를 기준으로 저장 공간을 나눈 버퍼이다. 3 이하 크기의 입력 FeatureMap은 3x3 버퍼에, 12 이상 24 이사 크기의 입력 FeatureMap은 24x24 버퍼에 저장된다.

그림 14. 입력 FeatureMap의 다중 읽기를 위한 Buffer

Fig. 14. Buffer for Multiple Input FeatureMap Loading

Top-net의 경우 그림 15 형태로 입력 FeatureMap을 읽어 저장한다. 하드웨어 자원에 적절한 버퍼의 크기를 정했다면,FeatureMap을 읽어 저장한다. 하드웨어 자원에 적절한 버퍼의 크기를 정했다면, 그 버퍼 크기 내에서 최대 개수의 입력 FeatureMap을 읽어와 동시에 연산할 수 있으므로 Memory Bandwidth를 낮춤과 동시에, 오버헤드 감소로 연산 속도 또한 높일 수 있다. CNN, Layer에 관계없이 공통적으로 사용가능한 커널의 병렬연산 지점을 Pseudo 코드를 활용하여 파악한다.

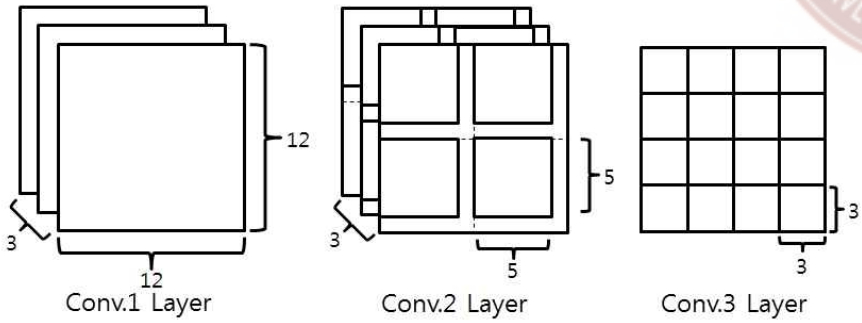


그림 15 Top-net에서의 다중 Input FeatureMap 로드

Fig. 15 Input FeatureMap Loading in Top-net

그림 16은 기본적인 CNN의 Convolution 반복 연산을 나타내는 코드이다. 그림 2를 참고하여 입출력 FeatureMap의 Depth, Width, Height, 커널 크기를 확인한다.

```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(out_d=0; out_d<M; out_d++) {
   for(in_d=0; in_d<N; in_d++) {
    for(m_row=0; m_row<K; m_row++) {
     for(m_col=0; m_col<K; m_col++) {
        output_fmap[out_d][row][col] +=
          weight[out_d][in_d][m_row][m_col] *
          input_fmap[in_d][S*row+m_row][S*col+m_col];
}}}}}}
```

그림 16. 기본적인 Convolution 연산의 슈도 코드

Fig. 16. Pseudo Code of Basic Convolution Operation

그림 17은 MAC 연산의 병렬처리 구간을 결정한 코드이다. Multiple 입력 FeatureMap을 처리하므로 반복문(in_fm)이 한 단계 추가되었다. 버퍼에 입력된 입력 FeatureMap의 수만큼 동작을 반복한다.

```
for(in_d=0; in_d<N; in_d++) {
 for(in_fm=0; in_fm<D; in_fm++) {
  for(out_d=0; out_d<M; out_d++) {
   for(row=0; row<R; row++) {
    for(col=0; col<C; col++) {
     #pragma unroll
     for(m_row=0; m_row<K; m_row++) {
      #pragma unroll
      for(m_col=0; m_col<K; m_col++) {
          output_fmap[out_d][row][col] +=
            weight[in_d][in_fm][out_d][m_row][m_col] *
            input_fmap[in_d][in_fm][m_row][m_col];
}}}}}}}
```

그림 17. 제안하는 Convolution 가속기의 구조

Fig. 17. Proposed Convolution Accelerator Structure

입력 FeatureMap을 기준으로 연산을 수행하므로 반복문의 차례 또한 입력 FeatureMap - 출력 FeatureMap이 우선되도록 변경한다. 커널의 크기가 3x3 혹은 2x2로 선택 가능하도록 병렬 처리한다. Local Memory 접근을 줄이기 위해 병렬 연산된 결과 값은 매 경우 Local Memory에 접근하지 않고, 하나의 레지스터에 저장되어 병렬 연산이 끝난뒤, Local Memory에 한번만 접근한다. 레지스터는 그림 18의 conv_result_temp에 해당하고, Local Memory는 output_fmap에 해당된다.

```
for(in_d=0; in_d<N; in_d++) {
 for(in_fm=0; in_fm<D; in_fm++) {
  for(out_d=0; out_d<M; out_d++) {
   for(row=0; row<R; row++) {
    for(col=0; col<C; col++) {
     #pragma unroll
     for(m_row=0; m_row<K; m_row++) {
      #pragma unroll
      for(m_col=0; m_col<K; m_col++) {
         conv_result_temp +=
           weight[in_d][in_fm][out_d][m_row][m_col] *
           input_fmap[in_d][in_fm][m_row][m_col];
     }}
     output_fmap[out_d][row][col] += conv_result_temp
}}}}}
```

그림 18. Loacl Memory 접근 횟수를 줄이는 Convolution 가속기 구조

Fig. 18. Local Memory Access Reduction Convolution Accelerator Structure

3.3.2 Implementation Details

임베디드 환경에서 구현된 전체 시스템 구조는 그림 19와 같다. On-Chip 영역에 CNN 가속기가 구현되었고, Off-Chip 영역에서 CNN을 제외한 나머지 소프트웨어 알고리즘이 동작하며 VGA 출력 또한 이루어진다. 얼굴 검출 시스템 작동에 앞서 FPGA, ARM Core에서 모두 접근할 수 있는 DDR 영역에 CNN 동작을 위한 다양한 Paramet-ers(Weight, Bias, PReLU)가 저장된다. 또한 얼굴 검출에 사용될 동영상을 불러온다. 얼굴 검출이 시작되면 ARM Core가 영상을 입력받고, 영상 전처리 과정을 진행하며 CNN 하드웨어 가속기에 데이터가 입력되도록 ROI를 결정한다. Global Memory의 특정 주소에 입력된 영상은 AXI 버스를 거쳐 순차적으로 CNN 가속기 내부의 Local Memory에 저장된다. CNN, Layer의 종류에 맞게 그림 19의 Compute Engine이 결정되고, Convolution, Pooling, Fully Connected 과정이 선택적으로 가속기 내부에서 동작한다. 구현된 가속기 내부 변수는 ARM Core에 의해 제어된다.

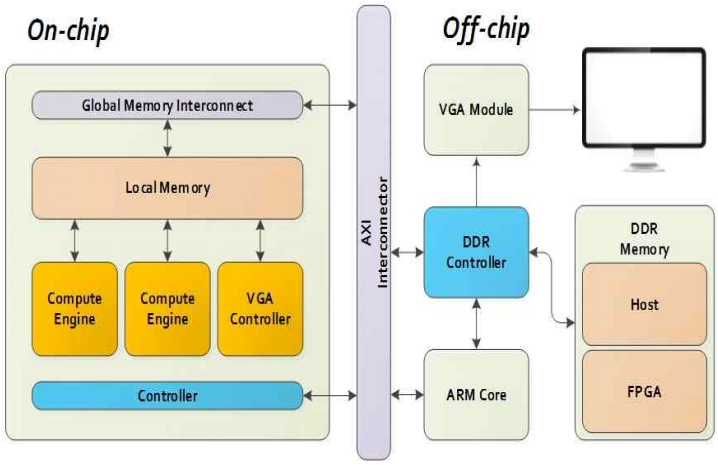


그림 19. 시스템 구현 개요

Fig. 19. Implementation Overview

가속기에 입출력될 FeatureMap과 Parameter는 가속기 내부의 메모리 주소 모듈이 계산한다. 연산이 완료된 데이터는 출력을 위한 Global Memory의 특정 주소에 저장된다. 출력할 영상을 준비하는 과정에서 FPGA에 구현된 VGA 컨트롤러가 VGA 출력 모듈에 영상의 동기 신호를 전달해준다. 데이터가 이동되는 과정은 시간 순으로 그림 20에 나타난다. 입력, 출력 버퍼는 각각 두 개씩 존재하며, 입출력 버퍼는 번갈아가며 Global Memory, Compute Engine에 접근한다. 예를 들어, 입력 버퍼 0과 출력 버퍼 1이 연산을 위해 Compute Engine과 데이터를 주고 받는 동안, 입력 버퍼 1과 출력 버퍼 0은 Global Memory와 데이터를 주고 받는다. Global Memory에 지속적으로 접근하므로, 데이터 전송을 위해 대기하는 시간을 줄일 수 있다.

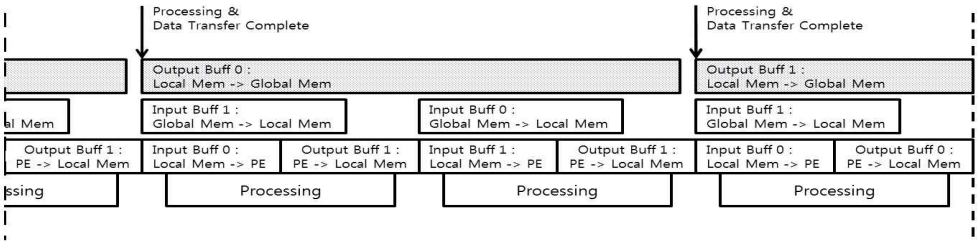


그림 20. 데이터 전송 순서 그래프

Fig. 20. Data Transfer Timing Graph

제 4 장 실험 및 분석

제시한 얼굴 검출 시스템의 검증을 위해 DE1-SoC 임베디드 보드(Dual-Core ARM Cortex A9, 1GB DDR3 SDRAM, Cyclone V 5CSEMA5F31)가 이용되었다. Intel FPGA for OpenCL 툴을 이용해 C/C++ 언어로 하드웨어를 개발하였다.

CNN의 학습을 위해 Face Detection Dataset and Benchmark (FDDB)[1] (2,845장-5,171얼굴), WIDER FACE[21] (12,203장-140,357얼굴) 데이터셋이 이용되었으며, 검증을 위해 FDDB, HD급 해상도의 동영상이 이용되었다.

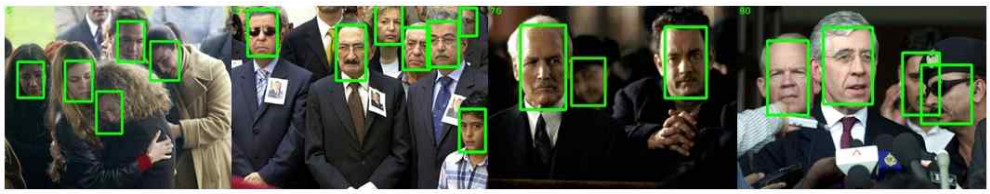


그림 21. FDDB 데이터를 이용한 얼굴 검출 결과 영상

Fig. 21. Our Face Detection Result Images using FDDB Dataset

표 3은 Cascade CNN 구조를 가지는 타 논문과 본 논문에서 설명한 얼굴 검출 시스템의 검출률을 나타낸 표이다. Test 데이터셋은 FDDB로 동일하며, 1, 2, 3 - Stage는 차례대로 Top, Mid, Bot-net과 같은 단계이다. 후반 CNN 단계로 넘어갈수록 많은 연산량이 요구됨에도, PC-GPU(Nvidia Titan Black) 환경에서 구현된 비교 시스템 [9], [19]과 임베디드 환경에서 구현된 본 시스템은 검출률에서 대등한 성능을 보인다.

표 3. 타 Cascade CNN과의 성능 비교

Table 3. Comparison with Previous CNNs

	Stage	[9]	[19]	Our System
CPU	-	Intel Xeon CPU	Unknown	ARM Cortex A9
Accelerator	-	Nvidia Titan Black GPU	Nvidia Titan Black GPU	CycloneV FPGA
Validation Accuracy	1	94.4%	94.6%	95.0%
	2	95.1%	95.4%	95.2%
	3	93.2%	95.4%	93.2%

표 4는 얼굴 검출을 위해 사용된 CPU, 가속기 종류 별 CNN의 동작 시간, FPS, 전0력 소모량을 나타낸다. 임베디드 환경에서 구현된 CPU-FPGA 통합 시스템은 30 FPS를 나타내며 실시간 얼굴 검출을 수행한다. 같은 임베디드 환경에서 CPU만을 사용한 경우보다 9배 이상 가속되었다.

전력 소모량은 임베디드 환경에 구현할 경우 PC에 비해 10여 배 이하로 전력 소모량이 줄어든다. 임베디드 환경에서 CPU만을 사용하면 전력 사용량은 적지만, 낮은 동작 속도로 인해 에너지 효율은 높지 못하다. 그러나 CPU-FPGA 통합 시스템은 낮은 전력 소모량과 빠른 동작 속도를 보장하여, 전력 대비 연산 효율성이 PC의 10.5배, CPU만을 이용한 임배디드 시스템의 8.3배로 가장 높다.

표 4. 동작 시간과 전력 소모

Table 4. Process Time and Power Consumption

	CPU	Accelerator	Top-net (12x12)	Mid-net (24x24)	Bot-net (48x48)	FPS
Embedded (DE1-SoC)	ARM Cortex A-9	-	0.3 ms	10 ms	54 ms	3.11
PC	Intel-i5 4460	Nvidia GTX 750 GPU	0.02 ms	0.76 ms	5.71 ms	34.81
Embedded (DE1-SoC)	ARM Cortex A-9	Cyclone V FPGA	0.027 ms	0.85 ms	7.04 ms	28.56

	Power	Power/FPS	Efficiency (W/FPS)
Embedded (DE1-SoC)	7.08 W	2.28 W	x1
PC	98.66 W	2.83 W	x0.8
Embedded (DE1-SoC)	7.79 W	0.27 W	x8.3

FPGA 가속기를 구성하는데 사용된 자원의 양을 표 5에서 볼 수 있다. 최대한 많은 양의 데이터를 최소한의 Global Memory 접근으로 읽고, 쓰기 위해 BRAM(Local Memory)의 사용량이 높게 나타난다. Convolution, Pooling, Fully Connected 모두 가속기에 설계, 구현되어서 LUT 사용량이 다소 높은 편이다.

표 5. FPGA 자원 사용량

Table 5. FPGA Resource Usage

Resource	LUT	FF	BRAM	DSP
Usage	26,939	53,886	417	28
Available	32,070	128,300	496	87
Percentage	84%	42%	84%	32%

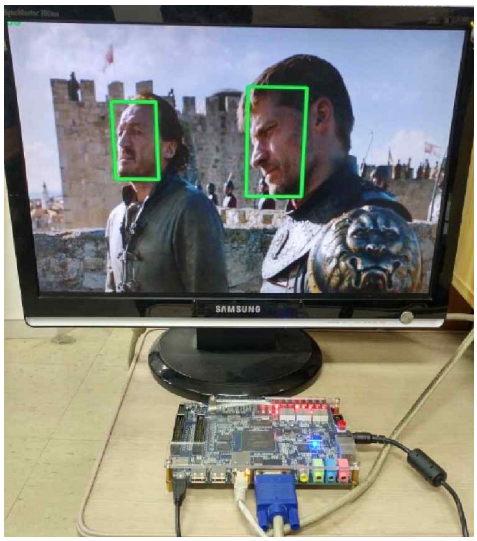


그림 22. 제안하는 시스템의 프로토타입

Fig. 22. Prototype of Proposed System

제 5 장 결론

본 논문에서는 저전력 임베디드 환경에서 실시간 동작하는 얼굴 검출 용 Cascade CNN의 CPU-FPGA 구조에 대해 분석하였다. 검증 결과 FPGA를 이용한 가속을 통해 HD (1280x720)급 영상을 입력으로 받아 실시간 동작하였으며, GPU를 활용한 PC 대비 1/10 수준의 전력을 소모하며 같은 동작을 수행했다. 따라서 CCTV, 모바일 환경 등 풍부한 전력, 하드웨어 자원을 쓸 수 없는 상황에서 CPU-FPGA 통합 시스템의 사용이 얼굴 검출에 적합함을 확인했다. 다만 영상 내부의 다양한 환경 변수로 인해, 영상의 종류에 따라 동작 시간과 검출률에 변화가 있는데, 검출 대상의 크기와 개수에 변화에 강인한 얼굴 검출 시스템의 추가적인 연구가 필요하다.

참 고 문 헌

[1] V. Jain and E. G. Learned-Miller, “FDDB: A benchmark for face detection in unconstrained settings”, Univ. Massachusetts, Amherst, MA, USA, Tech. Rep. MCS-2010-009, 2010.

[2] C. Zhang, “Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks”, Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp. 161-170

[3] P. Viola and M. J. Jones, “Robust real-time face detection”, Int. J. Comput. Vis., vol. 57, no. 2, pp. 137-154, 2004.

[4] M.Mathias, R. Benenson, M. Pedersoli and L. Van Gool, “Face detection without bells and whistles”, in Eur. Conf. Comput Vis., 2014, pp. 720-735.

[5] J. Yan, Z. Lei, L. Wen, and S. Li, “The fastest deformable part model for object detection”, in IEEE Conf. Comput. Vis. Pattern Recognit., 2014, pp. 2497-2504.

[6] X. Zhu and D. Ramanan, “Face detection, pose estimation, and landmark localization in the wild”, in IEEE Conf. Comput. Vis. Pattern Recognit., 2012, pp. 2879-2886.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks”, in Adv. Neural Inf. Process. Syst., 2012, pp. 1097-1105.

[8] Y. Sun, Y. Chen, X.Wang, and X. Tang, “Deep learning face representation by joint identification-verification”, in Adv. Neural Inf. Process. Syst., 2014, pp. 1988-1996.
[9] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua, “A convolutional neural network cascade for face detection”, in IEEE Conf. Comput. Vis. Pattern Recognit., 2015, pp. 5325-5334.
[10] C. Zhang and Z. Zhang, “Improving multiview face detection with multitask deep convolutional neural networks”, in IEEE Winter Conf. Appl. Comput. Vis., 2014, pp. 1036-1041.
[11] S. Chakradhar, M. Sankaradas, V. Jakkula and S. Cadambi, “A dynamically congurable coprocessor for convolutional neural networks”, In ACM SIGARCH Computer Architecture News, volume 38, pages 247-257. ACM, 2010.

[12] T. Chen and O. Temam. Diannao, “A small-footprint high-throughput accelerator for ubiquitous machine-learning”, SIGPLAN Not., 49(4): 269-284, Feb. 2014.
[13] A. Krizhevsky, I. Sutskever and G. E. Hinton, “Imagenet classication with deep convolutional neural networks”, Advances in Neural Information Processing Systems, 25 pages 1097-1105. Curran Associates, Inc., 2012.
[14] D. Aysegul, J. Jonghoon, G. Vinayak, K. Bharadwaj, C. Alfredo, M. Berin and C. Eugenio, “Accelerating deep neural networks on mobile processor with embedded programmable logic”, In NIPS 2013. IEEE, 2013.
[15] S. Cadambi, A. Majumdar, M. Becchi, S. Chakradhar and H. P. Graf,

“A programmable parallel accelerator for learning and classication”, In Proceedings of the 19th international conference on Parallel architectures and compilation techniques, pages 273-284. ACM, 2010.
[16] C. Farabet, C. Poulet, J. Y. Han and Y. LeCun, “Anfpga-based processor for onvolutional networks”, In Field Programmable Logic and Applications FPL 2009. International Conference, pages 32-37. IEEE, 2009.
[17] M. Peemen, A. A. Setio, B. Mesman and H. Corporaal, “Memory-centric accelerator design for convolutional neural networks”, In Computer Design (ICCD), 2013 IEEE 31st International Conference, pages 13-19. IEEE, 2013.
[18] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto and H. P. Graf, “A massively parallel coprocessor for convolutional neural networks”, In Application-specic Systems, Architectures and Processors 20th IEEE International Conference, pages 53-60. IEEE, 2009.
[19] Kaipeng Zhang, “Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks”, IEEE Signal Processing Letters, 2016, Volume:23, Issue: 10, pp. 1499 - 1503 IEEE Journals & Magazines
[20] K. He, X. Zhang, S. Ren and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on ImageNet lassification”, in IEEE Int. Conf. Comput. Vis., 2015, pp. 1026-1034.
[21] S.Yang, P. Luo, C. C. Loy, and X. Tang, “WIDER FACE:AFace detection benchmark”, arXiv:1511.06523.