한국정보전자통신기술학회논문지(jkiiect)'19-10 Vol.12 No.5
논문 19-12-05-467 http://dx.doi.org/10.17661/jkiiect.2019.12.5.467

NVIDIA Jetson TX1 기반의 사람 표정 판별을 위한 YOLO 모델 FPS 향상 방법

배승주*, 최현준*, 정구민**

YOLO Model FPS Enhancement Method for Determining Human Facial Expression based on NVIDIA Jetson TX1

Seung-Ju Bae*, Hyeon-Jun Choi*, Gu-Min Jeong**

요 약 본 이 논문에서는 NVIDIA Jetson TX1에서 YOLO v2 모델의 정확도를 유지하면서 FPS를 개선하는 방법을 제안한다. 일반적으로, 딥러닝 모델에서는 연산량을 줄여 처리 속도를 높이기 위해 파라미터들을 실수형에서 정수형으로 변환하여 정수 연산을 통해 속도를 높이거나 네트워크의 깊이를 감소시키는 방법을 사용한다. 그러나 이 방법들은 인식 정확도가 떨어질 수 있다. 이 논문에서는 YOLO v2 모델을 이용해 표정인식기를 개발하고 정확도 유지 시키기 위해 정수 연산이나 네트워크 깊이 감소를 사용하는 대신, 다음 세 가지 방법을 통해 연산량 및 메모리 소모를 줄인다. 첫 번째, 3x3 필터를 1x1 필터로 교체하여 각 Layer 당 매개 변수 수를 9 분의 1로 줄인다. 두 번째, TensorRT의 추론 가속 기능 중 CBR (Convolution-Add Bias-Relu)을 통해 연산량을 줄이고, 마지막으로 TensorRT를 사용하여 반복되는 동일한 연산구조를 가진 레이어를 통합하여 메모리 소비를 줄인다. 시뮬레이션 결과, 기존 YOLO v2 모델에 비해 정확도는 1 % 감소했지만 FPS는 기존 3.9 FPS에서 11 FPS로 282%의 속도 향상을 보였다.

Abstract In this paper, we propose a novel method to improve FPS while maintaining the accuracy of YOLO v2 model in NVIDIA Jetson TX1. In general, in order to reduce the amount of computation, a conversion to an integer operation or reducing the depth of a network have been used. However, the accuracy of recognition can be deteriorated. So, we use methods to reduce computation and memory consumption through adjustment of the filter size and integrated computation of the network The first method is to replace the 3x3 filter with a 1x1 filter, which reduces the number of parameters to one-ninth. The second method is to reduce the amount of computation through CBR (Convolution-Add Bias-Relu) among the inference acceleration functions of TensorRT, and the last method is to reduce memory consumption by integrating repeated layers using TensorRT. For the simulation results, although the accuracy is decreased by 1% compared to the existing YOLO v2 model, the FPS has been improved from the existing 3.9 FPS to 11 FPS.

Key Word : Deep Learning, Embedded system. Facial expression recognition, TensorRT, YOLO

1. 서론

최근 CNN 연구는 정확도를 높이는 것 외에 다양한 분야로 연구가 확대 되고 있다. 그 이유는 2015년 ImageNet Challenge의 결과를 보면 CNN 모델인 ResNet의 경우 사진 판별 오차율이 3.57%로 사람보다 적은 오차를 기록 하였

This paper is a research conducted by the government (Ministry of Trade, Industry and Energy) in 2019 with support of the core technology development project of the robotics industry (No. 10080615)
*Electronic Engineering, Kookmin University
**Corresponding Author : Electronic Engineering, Kookmin University (gm1004@kookmin.ac.kr)
Received October 03, 2019 Revised October 13, 2019 Accepted October 13, 2019

는데[1], 이미 사진 한 장에 대한 판별 정확도는 발전이 더딘 완성단계에 와 있음을 알 수 있다. 따라서 최근 연구는 이미지에서 객체들을 찾고 무엇인지 판별 하는 객체인식(Object Detection) 분야 등으로 확대 되고 있다. 대표적인 모델로는 Faster-RCNN, SSD(Single Shot Detector), YOLO(You Only Look Once)가 있다. Faster-RCNN 등의 알고리즘이 Region Proposal을 탐색 하고 탐색 된 Proposal 마다 Convolution을 진행 하기 때문에 여러 번의 판별 네트워크를 통과하는데[2] 비해 YOLO는 한 번의 네트워크를 통과하여 이미지에서 객체가 있는 Bounding Box를 찾고 Box안에 객체가 무엇인지 까지 판단 하기 때문에 타 모델에 비해 빠른 속도를 장점으로 가진다[3]. YOLO 모델은 현재 세 번째 버전인 YOLO v3까지 개발이 되었다. YOLO 모델 중 두 번째 버전인 YOLO v2는 첫 번째 모델의 7x7 필터를 3x3과 1x1로 변경하고 연산에서 상당한 시간을 차지하는 Fully-Connected를 Fully-Convolution으로 대체하고 Concat 연산 등을 적용함으로써 속도에 초점을 두며, 소폭의 정확도를 상승 시켰다[4]. 이 장점 때문에 컴퓨팅 파워가 PC에 비해 낮은 임베디드 시스템에서 객체인식 모델로 활용 하기에 적합하다.

임베디드 시스템에서는 딥러닝을 적용함에 있어 서버나 PC 기반의 환경 보다 낮은 성능이나 저전력, 낮은 메모리 소모량 등의 다양한 제약사항을 갖는다[5]. 이를 극복 하기 위해서는 기본적으로 비교적 낮은 연산량을 가지는 모델을 선정해야 하며, 메모리 소모량을 해결 하기 위해 네트워크 자체의 크기와 학습과 예측에 사용 되는 파라미터들을 최소화 해야 한다. 네트워크의 크기와 파라미터들을 최소화 하기 위해 다양한 연구들은 대표적으로 작은 Convolution Filter를 적용 하여 네트워크의 구조를 더욱 깊게 설계하고 파라미터 개수를 줄인 구글 Inception 모델이 있으며[6], 파라미터의 데이터 타입을 Integer 연산으로 바꾸어 수행하여 파라미터의 크기를 직접적으로 줄이며, 연산량을 감소 시키는 Quantization 방법과[7] 이미 학습이 완료된 네트워크의 파라미터 값들을 비교하여 값이 다른 파라미터 보다 월등히 작아 다음 Layer의 Activation에 영향을 미치지 않는 노드들을 제거하는 네트워크 프루닝 기법[8] 등이 있다. 하지만 위 방법들은 각각의 파라미터값에 민감한 반응을 보이는 YOLO 모델에서는 심각한 정확도의 하락을 보인다.

이 논문에서는 Jetson TX1에서 YOLO v2 모델의 FPS를 향상 시키기 위해 기존 네트워크의 3x3 Convolution Filter 중 일부를 1x1 Convolution Filter로 대체하여 모델의 학습 파라미터와 Feature Map의 개수를 줄여 모델의 크기와 연산량을 줄이는 방안과 NVIDIA의 추론 가속기인 TensorRT의 CBR 연산과 반복 되는 Layer의 통합 연산을 이용하여 메모리 소모량과 연산량을 감소시켜 FPS를 향상 시키는 방법을 제안 한다.

2. 관련 연구

2.1 객체인식 모델에 따른 성능비교 연구

Object Detection 모델을 비교 할 때는 두 가지의 중요한 지표가 있다. 첫 번째는 이미지를 처리 할 때의 계산 시간을 의미하는 FPS(Frame Per Second), 검출 정밀도(Precision)와 재현율(Recall)을 통해서 산출 하는 MAP(Mean Average Precision) 이다.

Geforce GTX Titan X를 이용한 위 세가지 모델에 대한 MAP와 FPS에 관한 실험을 보면 20개의 class를 가진 Pascal VOC Data set을 학습하여 비교한 결과 ResNet을 기반으로 한 Faster R-CNN의 경우 MAP는 76.4, FPS는 5가 나왔으며, SSD300은 74.3의 MAP와 46 FPS를 기록 하였다. YOLO v2의 경우 학습 하는 이미지의 크기 마다 차이를 보였는데 가장 큰 학습 이미지 크기인 544x544 모델의 경우 78.6의 MAP와 40 FPS를 보였으며, 본 논문에서 사

용한 YOLO v2 416x416 모델은 76.8의 MAP와 67 FPS를 기록 하였다[4].

YOLO는 입력 이미지를 그림 1과 같이 SxS의 Grid Cell로 나누며 각 Grid Cell에 포함된 Bounding Box의 정중앙에 객체가 위치하도록 조정 하며, 객체가 있을 것 같은 확률을 계산하여 Confidence Score를 붙인다. 때문에 YOLO 네트워크의 마지막 예측 Layer는 Class당 객체의 x, y 중심좌표와 Bounding Box의 넓이, 높이, Confidence Score까지 5개의 채널을 가지며, 이를 통해 단 한번의 연산을 통해 객체의 검출과 Class의 예측률을 표시 하기 때문에 빠른 속도를 갖는다[3].

그림 1. YOLO 모델의 그리드 셀 예시
Fig. 1. Example of YOLO Grid Cell

2.2 연산량 최적화를 통한 딥러닝 모델의 성능 향상 연구

연산량이 많은 딥러닝을 최적화 하기 위한 연구가 다양한 관점에서 진행 되고 있다. 구글의 Inception 모델은 이 연구에서는 작은 Convolution Filter의 이점에 대해 설명 한다. 기존 5x5 Filter를 두번의 3x3 Filter로 대체 하여 하나의 Layer를 두 개로 늘려 네트워크의 구조를 더욱 깊게 가져가는 반면 학습 시켜야 하는 파라미터는 5x5 = 25 개에서 9+9의 18개로 줄였다. 또한 1x1 convolution을 이용 하여 출력 Feature Map의 개수를 원하는 Depth에 맞춰 비선형적으로 줄이는 방식을 제안 하고 있다[6]. 하지만 구글 Inception의 방식에서는 네트워크의 정확도를 향상 시키기 위해 네트워크의 병렬화를 사용 했다. 이는 연산량을 늘리고 GPU의 병목 현상을 초래 할 수 있다.

YOLO 모델에서도 컴퓨팅 파워가 낮은 디바이스를 위한 YOLO-tiny 모델을 제공 하고 있다. YOLO-tiny는 30단 구조로 이루어진 YOLO v2 모델에서 특정 Layer 들을 제거 하여 15단 구조로 네트워크의 깊이를 줄였다.

YOLO v2는 네트워크가 깊은 만큼 앞단이 학습이 되지 않는 Gradient Vanishing 문제를 해결 하기 위해 26 번째 층에서 16 번째 층의 출력값을 다시 불러와 합쳐서 다시 Convolution을 진행 한다. YOLO-tiny 모델은 이러한 전파 Layer들을 없애고 총 20단의 Convolution Layer를 9단으로 줄였다. 이를 통해 약 145 FPS의 처리 속도를 얻었으나 정확도가 대폭 감소 하였다[3,4].

3. Jetson TX1 기반 YOLO v2 모델의 FPS 향상 방법 제안

이 논문에서는 Jetson TX1에서 YOLO v2 모델을 기반으로 사람의 표정 인식기를 구현하고 이 정확도를 최대한 유지 하면서 FPS를 높이기 위해 그림 2와 같이 3x3 Convolution Filter를 1x1 filter로 대체 하여 네트워크의 깊이를 원래 네트워크와 동일하게 유지하면서 파라미터를 줄여 FPS를 향상 시키는 방법과 TensorRT의 CBR 연산과 반복 Layer 통합 연산을 이용한 가속을 통해 FPS를 향상 시키는 방법을 제안한다. 또한 TensorRT를 적용하기 위해 YOLO의 네트워크 정의 파일과 Weight 파일을 Caffe 프레임으로 변환하는 방법을 제안 한다.

3.1 필터 크기 감소를 이용한 연산량과 모델 크기 축소

기존 YOLO v2 모델은 다음 표 1과 같은 네트워크 구조를 가지고 있다. Layer 항목의 Conv는 컨볼루션을 의미하며, Max는 Max

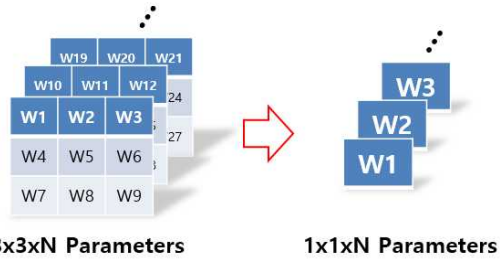

그림 2. 1x1 필터를 적용한 파리미터 수 감소
Fig. 2. Parameter Reduction by Applying 1x1 Filter

Pooling, Route는 이전 Layer의 결과값을 가져와 합치는 역할을 하는데, Filter 항목에 적힌 Layer의 값을 받는다. 27번 Layer의 경우 16번 Layer의 출력값과 25번 Layer의 출력을 합한다. 마지막으로 Reorg Layer는 입력의 크기를 줄이기 위해 입력 Feature를 Size 항목에 기재된 수로 나누어 줄여준다. Filter 항목에 경우, 컨볼루션 필터 개수를 의미한다. YOLO 모델의 정확도를 최대한 유지하면서, 속도를 향상 시키기 위해 1x1 Convolution Filter를 이용 한다. 이를 통해 네트워크의 정확도의 직접적으로 영향을 끼치는 Layer들의 깊이를 유지 하면서 기존 3x3 Filter가 가진 9 x Depth 개의 파라미터를 1 x Depth 개의 파라미터로 대체 하여 모델의 크기를 줄이며, 예측과 학습에서의 연산량을 줄인다. 표 2에서 나타내듯이 총 7개의 3x3 Filter를 1x1로 대체 하여 각 Layer 별로 파라미터의 개수를 9분의 1로 줄였으며, 총 파라미터 개수는 기존 69,965 개에서 41,805 개로 40.24%를 감소 시켰다.

표 1. YOLO v2 모델의 기본 구조
Table 1. Basic Network Structure Based On YOLO v2

Number	Layer	Filter	Size
1	Conv	32	3x3
2	Max		2x2
3	Conv	64	3x3
4	Max		2x2
5	Conv	128	3x3
6	Conv	64	1x1
7	Conv	128	3x3
8	Max		2x2
9	Conv	256	3x3
10	Conv	128	1x1
11	Conv	256	3x3
12	Max		2x2
13	Conv	512	3x3
14	Conv	256	1x1
15	Conv	512	3x3
16	Conv	256	1x1
17	Conv	512	3x3
18	Max		2x2
19	Conv	1024	3x3
20	Conv	512	1x1
21	Conv	1024	3x3
22	Conv	512	1x1
23	Conv	1024	3x3
24	Conv	512	3x3
25	Conv	1024	3x3
26	Route	16	
27	Reorg		/2
28	Conv	1024	3x3
29	Route	26, 24	
30	Conv	45	1x1

3.2 TensorRT의 CBR 연산을 이용한 네트워크 가속

NVIDIA 플랫폼에서는 Deepstream, cuDNN, TensorRT 등 다양한 추론 가속기를 지원 한다. 본 논문에서는 이중 TensorRT를 이용하였다.

TensorRT의 가속의 핵심은 3가지에 있다. 첫 번째는 CNN에서 일반적으로 사용하는 Convolution, Bias Add, Relu의 3단계의 연산을 1가지로 묶은 CBR연산이며, 두 번째는 동일

표 2. 1x1 필터로 대체한 레이어
Table 2. Layer Replaced With 1x1 Filter

Number	Layer	Filter	Size
3	Conv	64	1x1
7	Conv	128	1x1
11	Conv	256	1x1
15	Conv	64	1x1
21	Conv	1024	1x1

23	Conv	1024	1x1
25	Conv	1024	1x1

한 구조의 Layer가 연속 되면, Layer들의 중복 연산을 파악하여 텐서를 재활용 하는 것이고, 마지막은 파라미터 값을 INT8, FP16 등으로 Quantization 하는 것이다. TensorRT를 이용하기 위해서는 네트워크의 구조가 기입 되어 있는 모델 파일을 NVIDIA의 딥러닝 프레임워크인 Caffe의 Prototxt 형태로 변경해야 하며, 파라미터가 기입 되어 있는 Weight 파일 역시 Caffe의 Caffemodel 타입으로 변환 해야 한다. YOLO 모델은 Darknet이라는 자체 플랫폼에서 C언어를 기반으로 작성이 되어 있기 때문에 Caffe 타입으로 변환을 위해서는 각 Layer의 필터나 기능들을 Caffe 코드로 기입 해야 한다. Layer들 중 Activation Function으로 사용하는 Leaky Relu는 TensorRT에서 지원을 하지 않기 때문에 Relu와 Scale, Eltwise를 통해 근사하는 방안을 제안 한다.

각 Convolution Layer는 현재 층의 연산 결과를 다음층으로 전달하기 위해 Activation Function을 이용하기 때문에 각 층에서는 Convolution 연산을 진행 한 후 Bias를 더하고 활성함수를 거쳐 다음 층으로 결과를 전달 한다. TensorRT는 그림 3과 같이 Convolution과 Bias Add, Relu의 연산을 하나의 CBR 함수를 통해 처리하여 따로 하는 과정에서 발생 하는 불필요한 메모리의 읽기/쓰기를 줄여 메모리 소모를 최소화 하고 속도를 개선 한다.

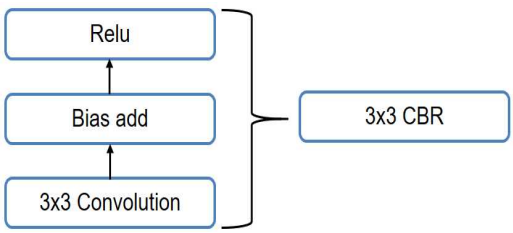

그림 3. CBR 연산 구조
Fig. 3. CBR Operation

4. Jetson TX1 기반의 YOLO v2 모델을 이용한 사람 표정 인식 구현

실험을 위해 Jetson TX1에서 YOLO v2 모델과 YOLO v2-tiny 모델, 최적화 모델을 각각 Happiness, Sadness, Surprise, Anger의 4 가지 표정을 판별 할 수 있게 구현하였으며, USB 카메라를 통해 실시간으로 표정판별을 진행하며 FPS를 측정하였다. 기본 YOLO v2 모델과 YOLO v2-tiny 모델을 이용한 표정인식기 구현은 YOLO 공식 github의 오픈소스를 이용하여 표정인식 데이터셋을 재학습 하여 구현하였고, 최적화 모델은 다음 세가지 과정을 통해 진행 하였다.

첫 번째, YOLO v2 모델의 네트워크 구조가 정의된 Cfg 파일에서 해당하는 3x3 필터를 1x1로 변경하였다.

두 번째, 수집한 표정인식 데이터셋을 총 15,000번 학습시키고, 각 1000번 당 파라미터 값를을 파일로 저장해두었다.

세 번째, TensorRT를 적용하기 위해 YOLO의 Cfg파일과 파라미터가 기록되어 있는 Weight 파일을 Caffe 구조로 변경 하였다. YOLO v2의 Layer 중 Leaky Relu와 이전 Layer의 결과값을 얻어와 합쳐서 연산 하는 route, reorg의 경우 TensorRT에서 지원을 하지 않기 때문에 Leaky Relu는 Relu+Scale+Eltwise의 조합으로 구성 하였으며, route와 reorg Layer는 Concat과 Reshape을 통해 구현 하였다.

FPS 향상 실험은 1x1 filter를 적용하여 변경 된 네트워크와 기본 네트워크를 동일하게 학습 시켜 실험을 진행 하였다. 또한 일반적인 연산량 개선 방법 중 네트워크의 깊이를 축소한 방법인 YOLO-tiny와 필터 변경과 TensorRT가 적용 된 최종 개선 모델과 정확도 및 FPS를 비교 하였다.

4.1 데이터셋과 데이터 전처리

표정 인식기를 구현 하기 위해 총 1870장의 이미지를 수집하였다. 이중 1430장을 학습 시켰

으며, 나머지 440장의 이미지를 가지고 테스트를 진행 하였다. 구현에 사용한 416x416 모델은 학습을 진행 하기 전에 사진을 416x416으로 resize 하므로 픽셀 정보 손상을 방지 하기 위해 416x416 이상의 이미지 만을 학습 시켰다.

YOLO 네트워크에서 예측을 위한 최종 Layer는 관련연구의 설명 대로 각 Class 별로 5개의 채널을 갖고 있다. 때문에 학습과 검출의 정확도 향상을 위해서 각 이미지의 전처리를 진행한다. 필수 요소는 각 이미지 별로 5개의 채널을 채우기 위해 객체의 x, y 중심좌표와 Bounding Box의 넓이와 높이를 0 ~ 1 사이의 값으로 매핑 했으며, 마지막에는 이미지를 Labeling 하여 이미지와 똑같은 이름으로 Text 파일을 만들어 기록하였다.

4.2 YOLO v2 모델과 YOLO v2-tiny 모델의 정확도 산출 실험 결과

테스트는 표정 마다 이미지 크기를 다르게 하여 각 110장씩, 총 440장의 데이터를 준비하였다. 테스트는 총 10번이 진행 되었으며, 각 정확도를 합하여 평균치를 구하였다. 표 3의 결과를 보면 tiny 모델의 경우 가장 높은 정확도가 85% 였으며, YOLO v2와 동일하게 5,000번을 학습 시킨 모델의 경우 15%의 정확도 차이를 보였다. 이를 통해 네트워크의 깊이를 줄이는 방법이 정확도에 큰 영향을 미친다는 것을 확인 할 수 있었다.

표 3. 학습 횟수별 정확도
Table 3. Accuracy By Number Of Learning Time

Model type	Number of learning	Accuracy
YOLO v2	10,000	80 %
YOLO v2	5,000	92.5 %
YOLO v2-tiny	15,000	85 %
YOLO v2-tiny	5,000	77.5 %

YOLO v2 모델의 경우 5,000번의 학습 보다 10,000번을 학습 시킨 모델의 정확도가 더 낮은 이유는 Overfitting이 일어난 것으로 추정 된다. 학습 데이터셋을 과도하게 학습하여 분류 곡선이 일반화 되지 않아 학습 시키지 않은 테스트셋에 대해 낮은 정확도를 보였다. 그림 4는 테스트 이미지에 대한 결과 이다.

그림 4. 표정인식 결과
Fig. 4. Facial Expression Recognition Result

4.3 Jetson TX1에서의 FPS 실험 결과

다음으로는 Jetson TX1 에서의 실시간성을 판별 하기 위해서 웹캠을 연결하여 FPS를 판별해 보았다. 웹캠을 통하여 실시간으로 이미지를 판별하게 했을 때 YOLO v2 모델은 평균 3.9 FPS를 기록 했다.

이 후 1x1 Filter와 TensorRT를 적용 했을 때에는 정확도가 1% 하락 하였으며, 2.1 FPS의 상승 효과를 얻어 6 FPS의 처리속도를 기록 했으며, 최종적으로 TensorRT를 적용 하였을 때는 정확도의 하락 없이 5 FPS가 상승 하여 11 FPS의 처리속도를 기록 하였다. 표 4는 기본 YOLO v2 모델과 개선된 모델의 FPS와 정확도 비교를 보여 준다.

표 4. 기존 네트워크 대비 FPS 개선 속도
Table 4. FPS Improvement Rate Compared To Basic Network

Model type	Average FPS	FPS increase rate compared to existing	Accuracy
Basic Model	3.9	0 %	92.5 %
Apply 1x1 Filter	6	153 %	91.5 %
Apply TensorRT	11	282 %	91.5 %

Tiny 모델의 경우 평균 FPS가 14.6을 기록하였다. 표 5는 제안 방안을 적용한 최적화 네트워크와 기본 네트워크, 일반적으로 네트워크의 깊이를 줄여 연산속도를 향상 시키는 YOLO v2-tiny의 정확도와 평균 FPS를 최종 비교한 모습 이다. 기본 네트워크는 30단의 모든 Layer의

표 5 . 네트워크 별 정확도와 FPS 비교
Table 5. Network-specific Accuracy, FPS Comparison

Model type	Average FPS	Accuracy
Basic Model	3.9	92.5 %
YOLO v2-tiny	14.6	85 %
Optimization Model	11	91.5 %

연산을 수행 하기 때문에 정확도는 가장 높지만 FPS는 상당히 낮은 모습을 보인다. YOLO v2-tiny 모델은 정확도는 최적화 네트워크 보다 높지만 Layer를 기존 30단에서 15단으로 줄여 정확도는 기본 네트워크에 비해 7.5 % 감소 하였다. 그에 반해 최적화 네트워크는 11 FPS를 기록하였지만 기본 네트워크의 정확도와 1% 밖에 차이가 나지 않는 모습을 보여준다.

5. 결론

본 논문에서는 NVIDIA Jetson TX1을 기반으로 YOLO v2, YOLO v2-tiny 모델을 이용해 4 가지 표정을 실시간으로 구분 하는 딥러닝 모델을 구현 하였고, 정확도를 유지하면서 FPS를 향상 시키는 방안을 제안하였다. 그 결과 1%의 정확도 하락과 7.1 FPS의 향상 효과를 얻을 수 있었다. 반면 네트워크의 깊이를 축소 시키는 YOLO v2-tiny 모델은 10.7 FPS를 향상 시켰지만 정확도 면에서는 7.5%의 큰 하락을 보였다. 이 실험을 통해 정확도를 유지하며 처리 속도를 향상 시키기 위해서는 네트워크의 깊이를 유지하는 방안이 효율적임을 알 수 있었다. 또한 임베디드 시스템의 연산 가속을 위해서는 네트워크 구조를 수정하는 것 외에도 연산 자체의 메모리 소모를 줄이는 방식이 조금 더 큰 처리 속도를 개선 할 수 있다는 것을 알 수 있었다.

향후 처리속도를 높이기 위한 연구 방향은 Quantization의 적용과 네트워크 프루닝을 들 수 있다. YOLO는 각 파라미터에 대해 민감한 편이기 때문에 Quantization을 이용하면 정확도의 큰 하락을 겪을 수도 있다. 이를 개선 하기 위해 학습 전 미리 데이터 타입을 변경하거나 정수 변환이 아닌 실수이되 변수의 크기를 일부 줄이는 방법을 적용해야한다.

REFERENCE

[1] Large Scale Visual Recognition Challenge(2018). http://www.image-net.org/challenges/LSVRC/ (accessed Oct., 15, 2018).

[2] Ren, S., He, K., Girshick, R., Sun. J, “Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in neural,” information processing systems, pp. 91-99, 2015

[3] Redmon, J., Divvala, S., Girshick, R., Farhadi, A, “You only look once: Unified, real-time object detection,” Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 779-788, 2016

[4] Redmon, J., Farhadi, A., “YOLO9000: better, faster, stronger,” arXiv preprint, 2017

[5] M.H Lee, W.C Kang, “Performance Enhancement and Evaluation of a Deep Learning Framework on Embedded Systems using Unified Memory,” KIISE Transactions on Computing Practices, Vol.23, No.7, 2017

[6] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Rabinovich, A., “Going deeper with convolutions,” Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1-9, 2015

[7] Gong, Y., Liu, L., Yang, M., Bourdev, L.,

"Compressing deep convolutional networks using vector quantization," arXiv preprint arXiv:1412.6115, 2014

[8] Han, S., Mao, H., & Dally, W. J., "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,"arXiv preprint arXiv:1510.00149, 2015

저자약력

배 승 주(Seung-Ju Bae) **[일반회원]**

- 현 재 국민대학교 전자공학과 석사과정
- 관심분야: 차량용 마이컴, 차량용 소프트웨어, 딥러닝, 자율주행, 영상처리

최 현 준(Hyeon-Jun Choi) **[일반회원]**

- 현 재 국민대학교 전자공학과 석사과정
- 관심분야: 차량용 마이컴, 차량용 소프트웨어, 자율주행, 안드로이드, IoT

정 구 민(Gu-Min Jeong) **[일반회원]**

- 1995년 서울대학교, 제어계측공학과 학사
- 1997년 서울대학교, 제어계측공학과 석사
- 2001년 서울대학교, 전기컴퓨터공학부 박사
- 2001년~2004년 ㈜ 네오엠텔 기반 기술팀, 팀장(co-founder)
- 2005년~현재 국민대학교 전자공학부 교수
- 2013년~현재 ㈜ 유비벨록스 사외이사
- 2015년~현재 국가기술표준원 자동차 전기전자 및 통신 전문위원회 위원장
- 2019년~현재 ㈜ 휴맥스 사외이사
- 관심분야 : 차량용 마이컴, 차량용 소프트웨어, 커넥티드카, 자율주행