Single Shot Multibox Detector 를 통한 윈도우즈용 실시간 문자 인식 Inference Program 개발

이다민*, 왕진영*, 신영진**, 남동윤**, 이상환***
*한양대학교 융합기계공학과
**새론오토모티브㈜ 생산기술팀
***한양대학교 기계공학부
e-mail : damin4973@gmail.com

Real Time Word Detecting Inference Program for Windows Through Single Shot Multibox Detector

Da-Min Lee*, Jin-Yeong Wang*, Yeong-Jin Shin**, Dong-Yun Nam**, Sang-Hwan Lee***
*Dept. of Mechanical Convergence Engineering, Han-yang University
**Production Technology Team, Saeron Automotive Corp.
***Dept. of Mechanical Engineering, Han-yang University

요 약

브레이크 패드 제작 공정에서 문자 인식은 사람이 직접 인식하거나 컴퓨터 비전 기술의 역할이었다. 하지만 사람의 인식 오류나 잉크가 번진 문자같은 새로운 형태의 문자를 인식하지 못하는 비전 기술의 단점 등 많은 한계가 존재했다. 본 논문에서는 C/CUDA 로 설계한 Single Shot Multibox Detector 기반 Inference Program 을 통해 더 정확한 문자인식 결과를 제시하고, CUDA 를 이용한 향상된 연산속도를 통해 실시간 문자 인식이 가능하도록 하였다. 문자 인식 정확도는 약 96.6%로 기존 비전 기술보다 더 뛰어난 성능을 보였다.

1. 서론

최근 인건비 상승과 공정의 효율성 등으로 인해 스마트 팩토리에 대한 관심이 높아지고 있다. 결함 검출, 문자 인식 등의 역할을 하는 비전 기술은 잉크가 번진 문자같은 예측하지 못한 문자가 들어오면 인식률이 떨어지는 단점이 있다.반면에 CNN(Convolutional Neural Network)기반의 딥러닝은 예측 못한 문자까지 인식 가능하다는 장점이 있다. CNN 은 위치 기반 데이터에 특화된 네트워크이며, 객체 분류, 객체 검출, 문자 인식 등에 많이 사용되고 있다.

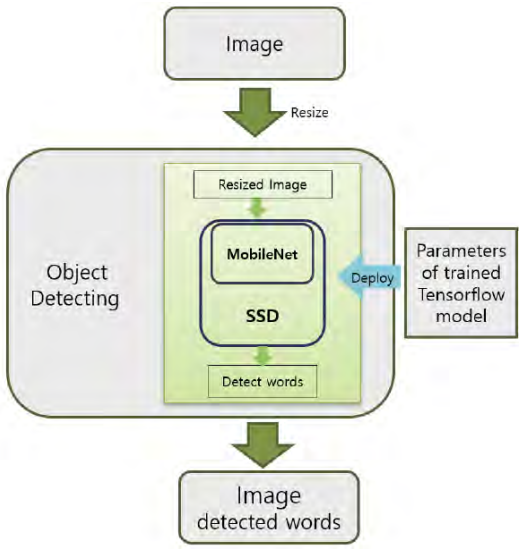
본 논문에서는 CNN 기반의 Single Shot Multibox Detector(SSD)를 사용해 브레이크 패드가 차종 별로 구분될 수 있도록 하는 마킹 문자를 인식해 올바른 분류가 이뤄질 수 있도록 한다. 리눅스용 Inference program 은 TensorRT 등 여러 라이브러리가 있지만, 그에 비해 윈도우용 프로그램은 많지 않다. 본 논문에선 Tensorflow Object Detection API 로 학습된 SSD 모델의 변수를 C 와 CUDA 로 설계한 SSD 모델에 배치하여 문자 인식 Inference program 을 개발하였다.

2. 연구방법

본 연구에서 문자 인식은 객체 검출 기술(Object Detecting)을 사용하고, 객체 검출에 사용되는 SSD 모델의 Backbone 모델은 MobileNet 을 사용한다.

윈도우즈용 Inference program 이 객체 검출을 하려면 학습된 모델의 변수가 필요하다. 본 연구에서는 Tensorflow Object Detection API 를 사용해 주어진 데이터를 토대로 SSD-MobileNet 을 학습 시킨 후, 모델의 변수를 추출해 C 와 CUDA 로 설계한 SSD-MobileNet 에 배치한다. 그 후 브레이크 패드 사진(그림 1)을 입력하면 객체 검출을 진행하여 실시간으로 문자를 인식하게 된다. 그림 2 는 문자 인식의 전체 흐름도이다.

(그림 1) 브레이크 패드



(그림 2) 문자 인식 흐름도

2.1 데이터

학습에 사용된 브레이크패드 이미지(그림 1)는 1000 장이다. 1000 장 이외에 예측 못한 이미지가 들어올 수 있으므로, data augmentation 을 통해 더욱 다양한 데이터셋을 만들었다. 그리고 문자 검출 모델을 학습시키기 위해선 각 문자에 레이블링이 된 데이터가 필요하다. 브레이크 패드 이미지에서 문자를 하나씩 박싱하고 각 문자대로 레이블링한 뒤, 모델 학습 시 레이블링 된 데이터를 사용한다.

2.2 모델 학습

Python 환경에서 Tensorflow Object Detection API 를 이용해 SSD-MobileNet 을 학습하였다. [1] 학습에는 COCO dataset 로 미리 학습된 SSD-MobileNet_v1_coco 를 적용하여 Transfer Learning 을 하였다. 학습 환경은 Window os 에서 Geforce GTX 1060 6GB 이다.

학습이 완료된 후 학습된 모델에서 변수를 이진 파일 형식으로 추출하여 Inference 모델에 적용한다. 변수 추출에는 Tensorflow 를 이용하였다.

2.3 Inference Program

윈도우즈에서 프로그램을 개발하기 위한 언어로 C 를 사용했고, 실시간으로 검출하기 위해 속도 향상을 목적으로 CUDA 를 사용하였다. 기본적인 Convolution, Activation 그리고 Batch Normalization 에는 CUDA 의 cuDNN 을 사용하였고, MobileNet 에서의 Depthwise Separable Convolution(DSC), SSD 의 상자를 예측하는 기술 등은 직접 CUDA 로 직접 설계하였다. [2][3] 그리고 시간적 손실이 큰 CPU 와 GPU 간 데이터 전송을 최소화하기 위해 MobileNet 에서 연산이 시작되기 전 Host 에서 Device 로 데이터를 전송하고, SSD 에서 Non-Maximum Suppression 이 끝난 후 Device 에서 Host 로 데이터를 전송하여 2 번의 전송만 하도록 했다.

2.3.1 SSD-MobileNet

SSD 의 특징 추출기로 사용된 MobileNet 구조는 1 개의 Convolution layer 와 13 개의 DSC layer 로 총 14 개 층으로 구성된다. [4] MobileNet 구조 중 DSC 는 일반 Convolution 연산을 depthwise 와 pointwise 로 나눠 연산하여 비교적 연산량이 적다. DSC 가 구조의 대부분을 차지하는 MobileNet 은 다른 모델보다 연산 속도가 빨라 실시간 검출 상황에 적합하다. 특히 DSC 를 CUDA 로 설계하여 연산을 더 빠르게 하였다.

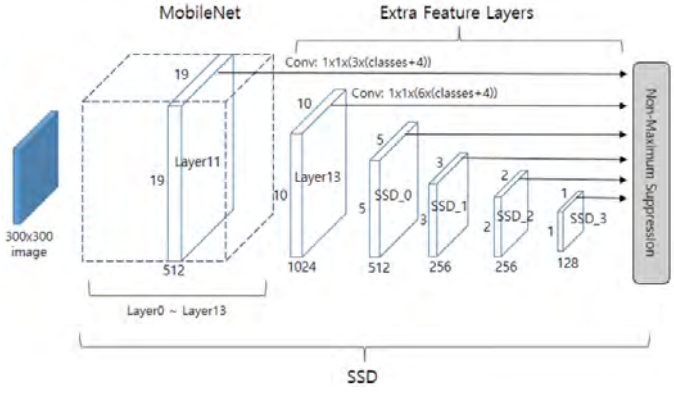
MobileNet 으로 특징을 추출한 후, 총 6 개의 추출 영역(feature map)에서 객체를 찾아내는 상자를 예측한다. 그림 3 을 보면 Layer11, Layer13, SSD_0, SSD_1, SSD_2, SSD_3 총 6 개의 영역에서 1x1 Convolution 을 통해 상자를 예측하는데, 채널 수의 의미는 c 개의 각 클래스에 대한 점수와 4 개의 상자 위치 정보(y_center, x_center, h, w)를 가지는 n 개의 상자이다. 따라서 각 특징 영역 당 곱해지는 Convolution 의 채널 수는 n(c+4)가 된다. Layer11 만 3 개의 박스가 생성되고 나머지 특징 영역에선 각각 6 개의 박스가 생성된다. [5]

객체 검출 성능을 높이기 위해 스케일[0.2, 0.35, 0.5, 0.65, 0.8, 0.95, 1.0]과 종횡비[1, 2, 3, 1/2, 1/3]의 조합으로 다양한 모양과 크기의 상자가 생성되게 하였다. 그리고 특징 영역에 균일하게 앵커 상자를 배치해 모든 영역에서 객체 검출이 가능하도록 하였다.

이 후 생성된 상자의 각 클래스에 대한 점수엔 배경 클래스도 포함되어 있어 배경에 대한 점수는 제거한 뒤 Softmax 를 적용하였다. 그리고 4 개의 상자의 크기/위치 데이터는 각 앵커 상자에 상대적인 값이므로 원본 이미지에 맞게 변환시켰다. [6]

위 과정을 거쳐 나온 모든 상자들을 나타내기 전 각 클래스 당 한계값(0.5)를 넘지 못하는 상자들은 제외하였고, 이 후 Non-Maximum Suppression 을 통해 최종적으로 각 클래스 별로 점수가 가장 높은 상자들만 나타나게 하였다.

생성된 상자는 문자의 순서와 무관하게 생성되기 때문에 바로 문자를 출력할 순 없다. 그림 4 와 같이 순서대로 문자가 출력되게 하기 위해 생성된 상자들의 중점을 이용했다. 상자의 중점을 기준으로 먼저 x_center 의 오름차순으로 정렬을 한 뒤, y_center 의 값이 다른 상자들의 값보다 일정 이상 큰 상자는 아랫줄 문자로 분류하여 문자 순서대로 출력되게 하였다.



(그림 3) SSD with MobileNet as backbone

3. 연구결과

문자 인식 정확도를 측정하기 위해 40 장의 브레이크 패드에서 모든 글자에 대한 정확도는 96.6%일 정도로, 템플릿 매칭 등으로 제한적인 문자 인식을 하는 비전 기술보다 정확도가 크게 향상되었음을 볼 수 있었다.

Tensorflow 와 속도를 비교해보면, Inference Program 의 연산 시간은 약 2.1 초가 소요되고 Tensorflow 는 약 5.5 초가 걸린다.(표 1) C/CUDA 로 만든 모델이 Python 에서의 Tensorflow 모델보다 2 배 이상의 빠른 성능을 보여주었다. 그림 4 는 Inference Program 이 실시간으로 문자를 인식하는 모습이다.

4. 결론

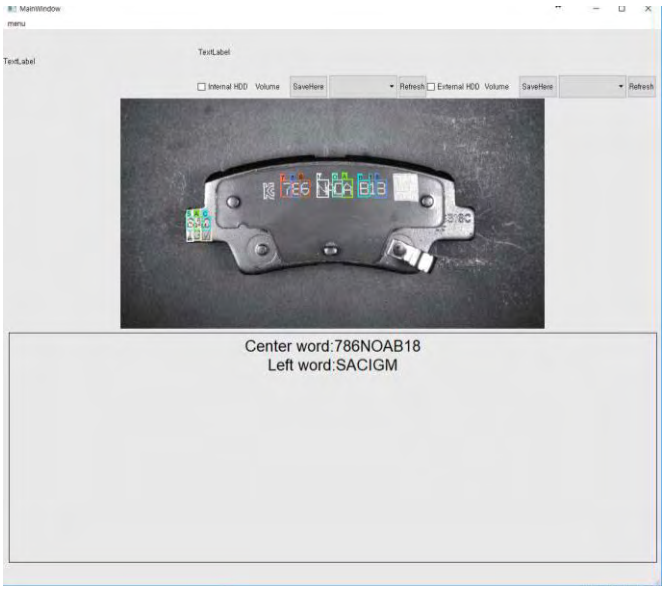
리눅스가 아닌 윈도우를 사용하는 산업 현장에서 본 논문에서 개발한 윈도우용 실시간 문자 인식 Inference Program 은 업무의 효율성을 높여줄 것이라 예상한다. 개선할 점은 산업에서 에러는 최소화해야 하기 때문에 더 많은 데이터를 통해 모델을 학습시켜 프로그램을 만든다면 정확도를 높일 수 있을 것이다.

감사의 글

이 논문은 새론오토모티브(주)의 지원을 받아 연구되었음

<표 1>Tensorflow 와 C/CUDA Inference Program 계산 속도 비교

	Computation time (second)
Tensorflow (Python)	5.5
Inference Program (C/CUDA)	2.1



(그림 4) 실시간 문자 인식 Inference Program

참고문헌

[1] Tensorflow Object Detection API, https://github.com/tensorflow/models/tree/master/research/object_detection

[2] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen and John Tran. “cuDNN: Efficient Primitives for Deep Learning”. arXiv: 1410.0759, 2014

[3] NVIDIA cuDNN, https://developer.nvidia.com/cudnn

[4] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam. “MobileNets : Efficient Convolutional Neural Networks for Mobile Vision Applications”. arXiv: 1704.04861, 2017

[5] W. Liu, D. Anguelov, D.Erhan, S.Christian, S.Reed, C.-Y.Fu, and A. C. Berg. “SSD: Single Shot Multibox Detector”. In ECCV, 2016.

[6] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. arXiv: 1506.01497, 2016