공학석사학위논문

딥러닝 기반 객체 검출 기술을 사용한 실시간 레고 블록 인식 및 분류 기법 제안

Proposal of Real-Time Lego Block Recognition and Classification Technique Using Deep Learning-Based Object Detection Technology

2022년 1월

창원대학교 대학원
문화융합기술협동과정
김 형 진

공학석사학위논문

딥러닝 기반 객체 검출 기술을 사용한 실시간 레고 블록 인식 및 분류 기법 제안

Proposal of Real-Time Lego Block Recognition and Classification Technique Using Deep Learning-Based Object Detection Technology

지도교수 유 선 진

이 논문을 공학석사학위논문으로 제출함

2022년 1월

창원대학교 대학원
문화융합기술협동과정
김 형 진

김형진의 공학석사학위논문을 인준함

심사위원장 남 상 훈 (인)

심사위원 유 선 진 (인)

심사위원 오 범 석 (인)

2022년 1월

창원대학교 대학원

목 차

그 림 목 차

표 목 차

국문 초록

객체 검출 기술은 컴퓨터 비전 분야에서 원하는 객체에 대해 위치 정보와 종류를 분류하여주는 기술이다. 객체 검출은 자율주행이나 차량 번호판 인식 등 다양한 분야에서 응용되고 있다. 하지만 이러한 객체 검출 기술은 다중 객체 검출이나 추적 같은 경우에서 무궁무진한 활용도를 가짐에도 실제로는 부족한 성능을 보였다. 하지만 CNN의 등장으로 부족한 성능을 개선하는 연구가 진행됐고, 이에 따라 다중 객체 검출에도 유용한 YOLO가 등장하였다. 기존의 CNN이 속도가 느리다는 단점이 있었지만, YOLO가 이를 보완하였다. 이를 통해 본 논문에서는 YOLO를 이용하여 다중 객체 검출에 유용한 시스템에 적용하여 사용할 수 있도록 레고 블록에 대한 인식과 분류 기법을 조립 가능한 모델을 제안 및 설명해주는 시스템을 제안한다. 해당 시스템에서는 바닥에 흩뿌려진 레고를 검출함으로써 현재 가지고 있는 레고의 개수와 종류를 파악할 수 있도록 하였고, 이를 통해 조립 가능한 모델에 대한 리스트를 선정하여 보여준다. 그리고 조립 과정 중 필요한 레고를 검출해, 해당 레고의 정보를 제공한다. 검출 알고리즘에는 YOLO 중에서도 YOLOv4와 비교를 위해 CNN의 검출 알고리즘 중 하나인 Faster R-CNN을 사용하였다. 각각 다른 조건에서 YOLOv4를 학습시키고, 이 중에서 결과 비교를 통해 가장 나은 결과값을 도출해내는 조건의 YOLOv4를 선별하여 해당 조건과 Faster R-CNN과 비교를 통해 YOLOv4의 상대적 검출 성능을 확인하였고, YOLOv4의 부족한 검출 부분은 Dropout 기법을 적용하여 검출 정확도를 향상시켰다. YOLOv4는 동일한 환경에서 Faster R-CNN보다 더 뛰어난 블록 검출률을 보였지만 블록이 겹쳐져 있거나, 붙어있을 시에는 검출에 어려움을 겪었다. 이를 위해 Dropout 기법을 적용하여 새로운 데이터를 인식하지 못하는 과적합 문제를 다소 해소하였고, 겹쳐있는 블록에 대해서도 기존의 YOLOv4와 비교하였을 때 검출이 우수한 것을 확인할 수 있었다.

키워드

Deep Learning, Object Detection, YOLOv4, Unity, Faster R-CNN

Ⅰ. 서 론

객체 검출이란 영상이나 이미지에서 찾고자 하는 객체의 위치를 예측하고 이를 분류하는 과정을 뜻한다. 객체 검출은 얼굴 인식, 자율 주행, 차량 번호판 인식 등 다양한 분야의 어플리케이션에서 필요로 하는 수요가 높은 기술이기 때문에 이와 관련하여 꾸준히 연구가 진행되고 있다[1-4]. 최근에는 CNN(Convolutional Neural Networks) 기술을 중심으로, 컴퓨터 비전 분야에서 다양한 연구를 통해 두각을 드러내고 있으며, 객체 검출과 트레킹 등 다양한 작업이 이에 해당한다. 특히, CNN을 기반으로 한 객체 검출 알고리즘들이 대표적인 벤치마크 데이터 세트인 PASCAL VOC(Visual Object Classes)와 MS COCO에서 기존에 제안된 검출 알고리즘의 성능을 압도함에 따라 CNN을 기반으로 한 검출 알고리즘들이 활발히 연구되고 있다[5-6]. 대표적인 CNN 기반의 검출 알고리즘으로는 R-CNN, Fast R-CNN, Faster R-CNN, YOLOv1, SSD, YOLOv2, YOLOv3 등이 제안되었다[7-13]. 그리고 각종 검출 알고리즘들이 등장함에 따라 응용되는 분야도 더욱 넓어지고 있다. 자율 주행이나, 바이오 이미지, 의료 분야, 산업적 분야 등 여러 방면에서 쓰이고 있는 객체 검출 기술은 점점 발전하고 있는 기술 중 하나다[14]. 다중 객체 추적(Multi Object Track-ing; MOT) 또는 다중 대상 추적(Multi Target Tracking; MTT) 기술은 활용할 수 있는 분야가 폭넓음에도 불구하고 실제에 적용하기에는 다소 부족한 모습을 보이고 있다. 하지만 최근에는 추적 모듈에도 CNN 기반의 딥러닝 모델을 사용하여 객체 추적 문제를 개선하는 연구가 진행되고 있다[15]. 최근 딥러닝 기반 다중 객체 추적 알고리즘의 CNN 기반 객체 특징 추출 과정을 적용하여 다중 객체 검출이 이루어졌지만, 정확도와 비교하였을 때 속도가 느리다는 단점이 있었다. 하지만 이러한 단점을 보완하고 나타난 것이 바로 YOLO이다[16]. 본

논문에서는 CNN 기반 네트워크인 YOLO와 Faster R-CNN으로 실시간으로 다중 객체 검출을 실제로 적용하고자 한다. 이를 통해 어떤 알고리즘이 더 우수한지 판별하고, 이 과정에서 발생한 객체 검출 성능의 문제를 Dropout으로 해소한 YOLOv4와 비교하여 유의미한 결과의 여부를 확인하고자 한다.

Ⅱ. 기술 현황 분석

2.1 객체 인식(Object Detection) 종류

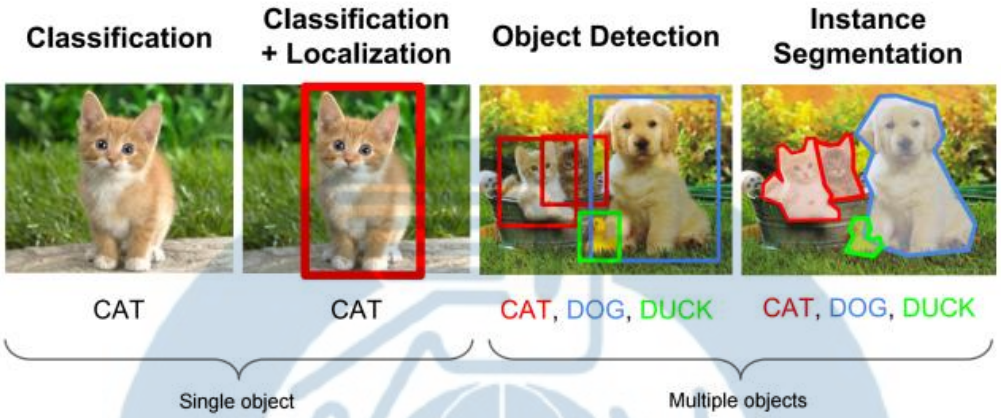


그림 1. 이미지 분류 및 검출, 분할의 특징

분류(Classification)

이미지를 다루는 분야의 딥러닝에서 분류란, 입력으로 받은 이미지에서 객체가 어떤 종류인지 구분하는 과정이다[17]. 이는 클래스(Class), 라벨(Label) 혹은 Class label이라고 불린다. 클래스는 MNIST 데이터 세트를 예시로 들면, 0에서 9까지 총 10개의 수를 모두 다른 클래스로 분류하고 하나의 숫자를 이미지로 입력했을 시, 이것을 0에서 9 사이의 숫자 중 어떤 수인지 클래스를 구분하여 출력하는 것이다. 그림 1의 첫 번째 이미지의 경우엔 입력된 이미지가 고양이인데, 해당 이미지의 클래스를 하단의 'CAT'로 분류하여 출력한 것이다.

지역화(Localization)

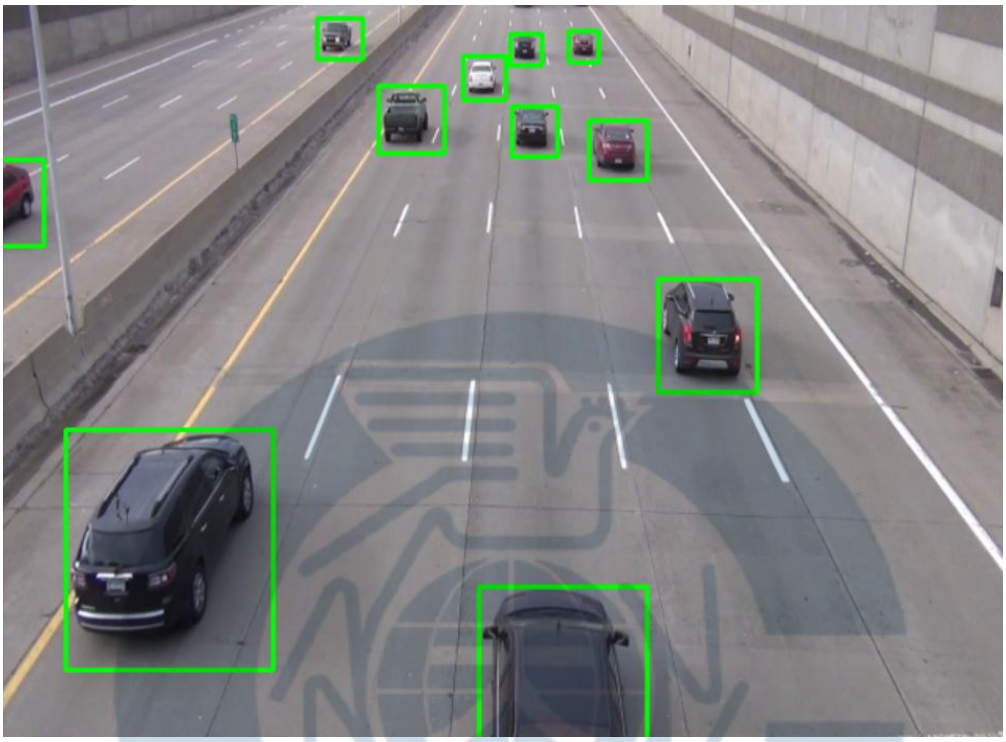


그림 2. Localization 예시

지역화란 앞의 그림 1의 두 번째 경우처럼 모델이 제시된 이미지 내의 특정 객체가 어느 위치에 존재하고 있는지 정보를 알려주며, 보통 바운딩 박스로 사용을 하여 바운딩 박스의 네 꼭짓점 픽셀(Pixel) 좌표가 출력되는 것이 아닌 좌상단(Left top), 혹은 좌하단(Right bottom) 좌표를 출력한다. 그림 2를 보면 실제 고속도로의 CCTV에서 지나가는 자동차들을 바운딩 박스로 표기한 것을 볼 수 있다.

객체 검출(Object Detection)

객체 검출은 일반적으로 지역화와 분류가 함께 진행되는 것을 뜻한다. 객체를 검출하는 방법에는 모델이 필요한 목적에 따라 원하는 특정한 객체만 검출하는 방법과 화면에 있는 객체를 모두 검출하는 다중 객체 검출 모델이 있다. 종종 객체 검출은 지역화의 의미로만 사용되는 경우도 있다. 이 경우는 이미지 위에 모델이 학습한 객체의 위치만 바운딩 박스로 표현되고 클래스 종류는 구분하지 않는 경우이다[18].

또한, 영상 및 이미지를 입력값으로 넣게 되면 유저가 따로 찾고 싶은 사람이나 객체뿐만 아니라 다양한 요소가 특징으로 잡아 찾을 수 있게 된다. 객체 검출의 목표는 마치 사람의 눈을 통해 보는 것처럼 판단할 수 있게 되어, 자연스러우며 속도와 정확도를 중요시하여 결과를 도출해내는 것이 목표이다.

객체 검출은 자율주행, 바이오 이미지, 산업적 분야, 로봇 비전 분야에서도 다양하게 쓰인다. 또한, 분류와도 비슷한 면이 존재한다. 객체 검출 시스템은 크게 두 가지의 구조로 나뉜다. 객체를 어떤 구조를 통해 학습하고 판별해내는지에 따라 One-stage 기법과 Two-stage 기법으로 나뉘게 된다. One-stage 기법에는 대표적인 예시 모델로 YOLO와 SSD, RetinaNet 등이 사용되고, Two-stage 기법으로는 R-CNN 구조에서 Fast R-CNN과 Faster R-CNN이 사용된다. 두 가지의 차이점은 구조가 다른 것도 있지만 목표가 명확하게 다르다는 것이다. 가장 큰 차이점은 One-stage 기법은 속도를 중시함에 있고, Two-stage 기법은 정확도가 연구에 집중되어있다는 것이다.

2.2 객체 검출 모델 분류

2.2.1 객체 인식을 위한 데이터 세트 및 성능 지표

객체 검출을 하는 데에 있어서 방법을 찾기 위해 여러 데이터 세트가 출시되고 있다. 데이터 세트는 크게 세 가지로 나뉘는데 Pascal VOC(PASCAL Visual Object Classification), ImageNet, COCO(Common Objects in COntext) 이다[18]. 그 중 첫 번째로, PASCAL VOC 데이터 세트는 객체 검출, 객체 분류, 객체 분할 같은 용도로 주로 쓰이고 있다. 객체가 있는 바운딩 박스가 포함된 학습 및 테스트를 하기 위해 10,000개 정도의 이미지가 존재한다. PASCAL VOC 데이터 세트는 개수로는 20개의 카테고리를 포함하여 다른 데이터 세트보다 종류는 적으나, 객체 검출에 있어서 대표적인 참조 데이터 세트로 사용한다.

두 번째로 ImageNet은 2013년부터 바운딩 박스가 있는 객체 검출 데이터 세트를 발표했다. 방대한 훈련 데이터를 담고 있는데, 무려 훈련을 위한 이미지 500,000개와 200개의 클래스로 구성되어 있다. 데이터 세트의 크기에 따라 훈련 시 계산 능력을 필요로 하므로, 많은 수의 클래스를 가진 ImageNet은 물체 인식 작업을 복잡하게 만들어 거의 사용되지 않는다.

세 번째는 COCO 데이터 세트로, 이는 마이크로소프트사에서 제작하였다[20]. 해당 데이터 세트는 그 용도가 객체 검출, 키 포인트 감지, 객체 분할 등과 같은 여러 문제 해결에 있다. 그리고 보통 그림 3과 같이 COCO 물체 감지 과제에서는 바운딩 박스 위에 물체가 존재하는 이미지를 80개의 클래스를 각각 분류하는 것으로 구성되어 있다. 데이터 세트는 매년 변경되지만 훈련하고 테스트하기 위한 이미지를 제공하는데, 120,000개 이상의 이미지가 보편적으로 제공된다.

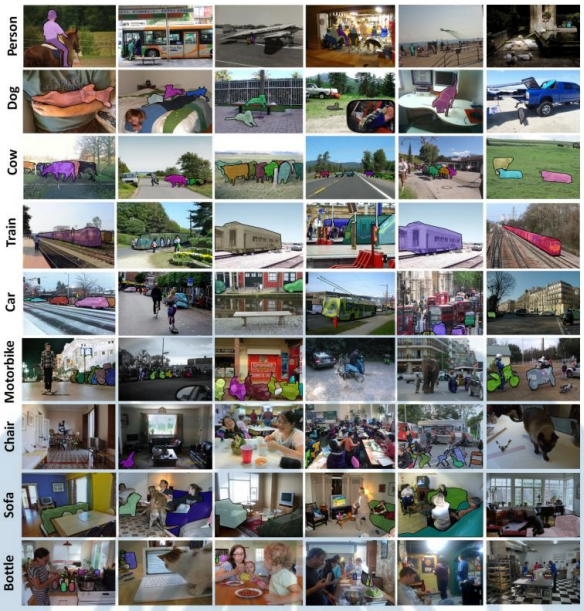


그림 3. COCO 데이터 세트의 분할된 개체 예시

객체 검출에 대한 정밀도를 평가하려면 우선 낮은 신뢰도로 인해 예측되지 않은 영역을 제거해야 한다. 그 후 IoU(Intersection over Union) 영역을 0과 1의 사이 값으로 정하여 사용한다. 이는 예측 박스와 실제 영역의 박스의 사이에 겹치는 범위에 해당하는데, 그림 4를 참고하면 어떤 영역이 무엇인지 알 수 있다. 이 IoU 값이 높을수록 일반적으로 찾고자 하는 객체에 대한 박스의 예측값이 더 뛰어남을 의미한다. 그림 5를 보면 IoU의 값에 따른 박스의 상태를 알 수 있다. 또한, IoU가 지정한 임곗값보다 높은 경계 상자 후보를 유지한다.

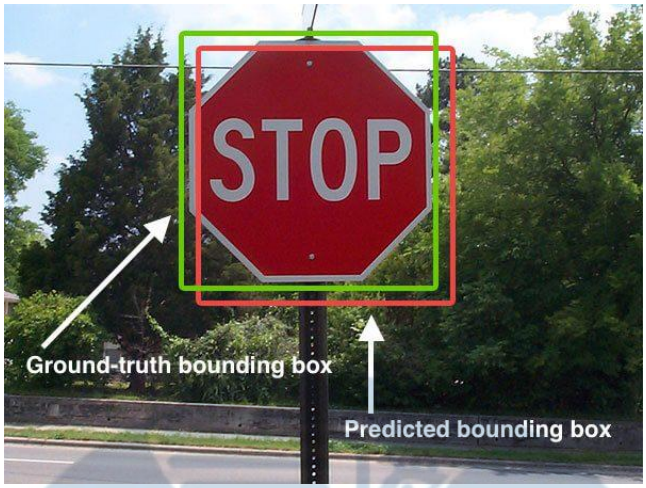


그림 4. STOP 표지판을 검출하는 예시, 예측 바운딩 박스를 의미하는 붉은 박스와 실제 바운딩 박스를 의미하는 녹색 박스
(Adrian Rosebrock, 'IoU for object detection', 2016년 11월 07일, 2021년 11월 22일 접속. https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/)

그림 5. IoU 값에 따른 박스 상태
(Adrian Rosebrock, 'Intersection over Union (IoU) for object detection', 2016년 11월 07일, 2021년 11월 22일 접속. https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/)

물체 감지 문제에 일반적으로 사용되는 측정 항목을 평균 정밀도(mAP)라고 하는데, mAP 메트릭은 소수의 클래스에서 극도의 전문성을 가지지 않으므로 다른 클래스에서는 성능이 약한 특징이 있다.

mAP의 점수는 IoU에 의해 고정적으로 계산되는 것이 보편적이지만, 경계 상자의 수가 늘어남에 따라 바뀔 수 있다. 하지만 COCO에서 상자가 과하게 생성되는 것을 막기 위해서, 가변 IoU의 값에 대한 평균 정밀도 점수의 평균을 구하여 잘못된 분류에 패널티를 주도록 하였다.

2.2.2 객체 검출 모델 YOLOv4

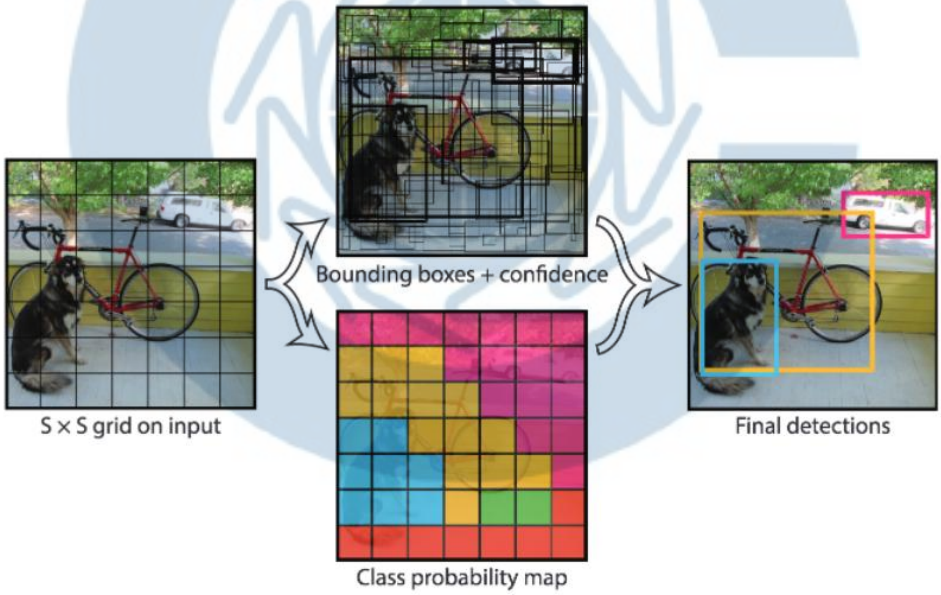


그림 6. YOLO의 이미지 인식 기법 구조

그림 6은 YOLO의 이미지 인식 기법의 구조를 보여준다. YOLO 같은 One-stage 기법은 R-CNN 계열의 Region proposal 기법이 생략됨으로써,

객체 인식 속도가 향상되었다. 입력 이미지를 S×S 크기의 그리드 영역으로 나누고, 해당 물체가 있을 것 같다고 예측하는 곳에 바운딩 박스를 친다. 그리고 바운딩 박스의 크기와 중심에 해당하는 좌표를 구해서 정확도를 의미하는 Confidence를 구한다. 실제 물체의 영역을 의미하는 Ground truth 박스와 예측값의 겹치는 비율을 나타내는 IoU와 곱하여서 Confidence 값을 구한다[21].

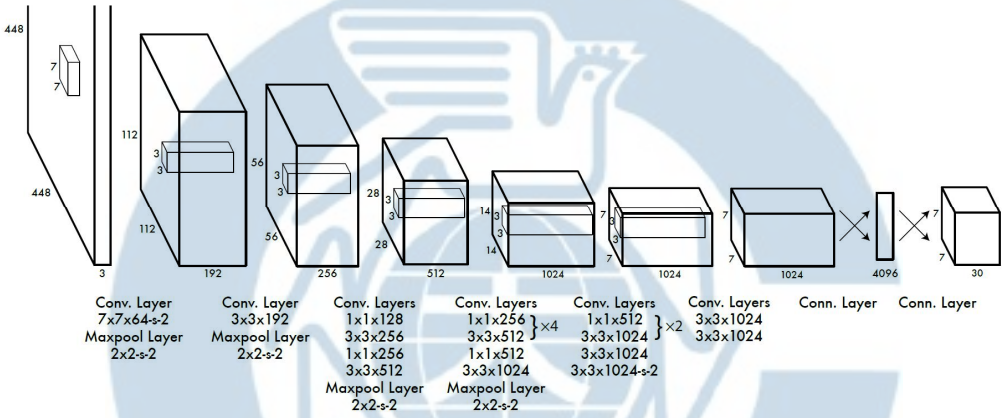


그림 7. YOLO의 컨볼루션 네트워크 구조

그림 7은 YOLO에서 사용되는 컨볼루션 네트워크 구조를 나타내고 있다. YOLO의 신경망 구조는 이미지 분류에 쓰이는 GoogLeNet을 사용했다[22]. YOLO는 총 24개의 컨볼루션 레이어(Convolutional layers)와 2개의 전결합 계층(Fully connected layers)으로 구성되어 있다. GoogLeNet의 인셉션 구조 대신 YOLO는 1×1 축소 계층(Reduction layer)과 3×3 컨볼루션 레이어의 결합을 사용했다. 1×1 축소 계층과 3×3 컨볼루션 레이어의 결합이 인셉션 구조를 대신한다. YOLO 모델의 전체 구조는 다음과 같다. 이 모델의 최종 아웃풋은 7×7×30의 예측 텐서(Prediction tensors)이다.

YOLO의 CNN 네트워크 모델은 CNN으로 구현되었고, PASCAL VOC 데이터 세

트를 이용하여 평가되었다. 네트워크의 초기 컨볼루션 레이어는 이미지의 특징들을 추출하고 전결합 계층은 출력될 확률과 좌표를 예측한다. YOLO 네트워크 아키텍처에서 이미지를 분류하는 방법은 24개의 컨볼루션 레이어와 전결합 계층 2개가 존재하는 네트워크로, GoogLeNet 모델에서 고안해 낸 것이다. YOLO는 GoogLeNet에서 사용하는 초기 계층 대신 단순히 1×1 축소 레이어와 3×3 컨볼루션 레이어를 사용한다[23].

위의 방식이 YOLO의 시작이고, 여기에 Darknet-19 모델을 이용하여 전이 학습을 통해 성능을 향상시킨 것이 YOLOv2 이다[24]. 이후 YOLOv3에서는 멀티라벨 분류기법과 모델을 Darknet-19에서 Darknet-53으로 더 깊게 설정하면서 정확도와 속도를 올렸다[25]. YOLOv4에서는 이전과 달리 BoF 데이터 증강기법을 이용하여 입력받은 이미지에 기하학적 변형과 광도를 바꿔주어, 학습모델의 학습에 있어서 이미지의 다양한 환경의 변화에도 견고하도록 하였다. 또한, BoS 기법처럼 다른 네트워크와 함수를 변경하였고, 백본 단계에선 기존의 레이어에 PANet을 추가함으로써 이전의 YOLO보다 성능을 향상했다.

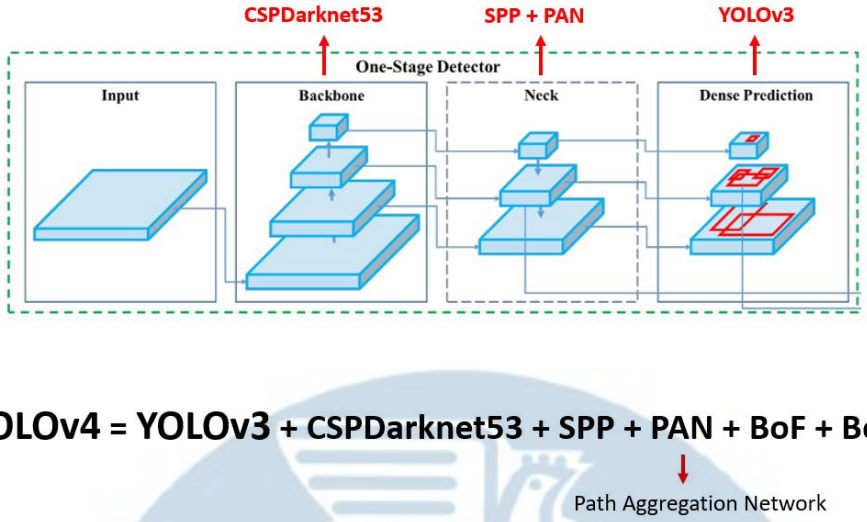


그림 8. YOLOv4 전체적인 구조

다음 그림 8은 YOLOv4의 전체적인 구조를 나타낸다. 입력 이미지가 들어왔을 때 백본 구조에서는 그림과 같이 CSP Darknet53을 사용하였고, 중복 기울기를 제외함으로써 정확도 및 속도를 높였다. 또한, 넥 부분에서는 SPP와 PAN을 사용하여, 입력받은 이미지에 대해 이전 버전처럼 크기 크기를 224×224로 고정하지 않아도 된다는 것이다. 그리하여 이미지가 크더라도 학습에 이용할 수 있고, 더욱이 정확성을 높일 수 있다. 마지막 부분에서는 기존의 YOLOv3과 같이 네트워크 구조를 구성하였다.

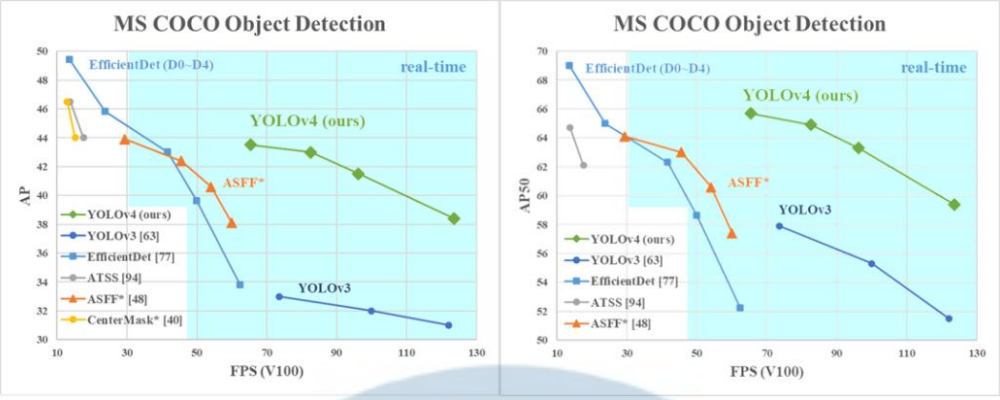


그림 9. 제안된 YOLOv4 및 기타 객체 검출기 비교

위의 그림 9를 보면 알 수 있듯이 YOLOv4에서는 COCO 데이터 세트에 대한 객체 검출에서 평균 정밀도를 뜻하는 AP의 수치 43%, AP50은 65%를 기록하였고 동일한 조건인 Tesla V100에서 65FPS의 속도를 기록하며 속도와 정확도의 측면에서 다른 검출기보다 뛰어난 검출기임을 증명했다[26]. 또한, YOLOv4는 이전 YOLOv3와 비교하여 평균 정밀도 AP가 10%, FPS는 12%가 증가하였음을 보여준다.

2.2.3 객체 검출 모델 Faster R-CNN

직전에 개발된 Fast R-CNN의 경우 RP에서 CNN으로 가는 길목에서 RoI 풀링을 통해 아주 많은 시간을 절약했지만, 여전히 RP를 생성해내는 것 자체는 많은 시간이 소요된다. Fast R-CNN의 단점은 Selective search를 수행하는 지역 제안(Region proposal) 부분이 외부에 존재하기 때문에 추론(Inference)에서 병목현상(Bottleneck)을 일으키는 것이다[27]. 그러므로 만들어진 Faster R-CNN에서는 RPN(Region Proposal Network) 자체를 학습한다. 즉, Faster R-CNN의 목표는 Selective search 없이 RPN을 학습하는

구조로 모델을 만드는 것이다. RPN은 피쳐 맵(Feature Map)을 입력값(Input)으로, RP를 출력값(Output)으로 하는 네트워크라고 할 수 있고, selective search의 역할을 온전히 대체한다[28]. Faster R-CNN의 전체 구조는 다음의 그림 10과 같다.

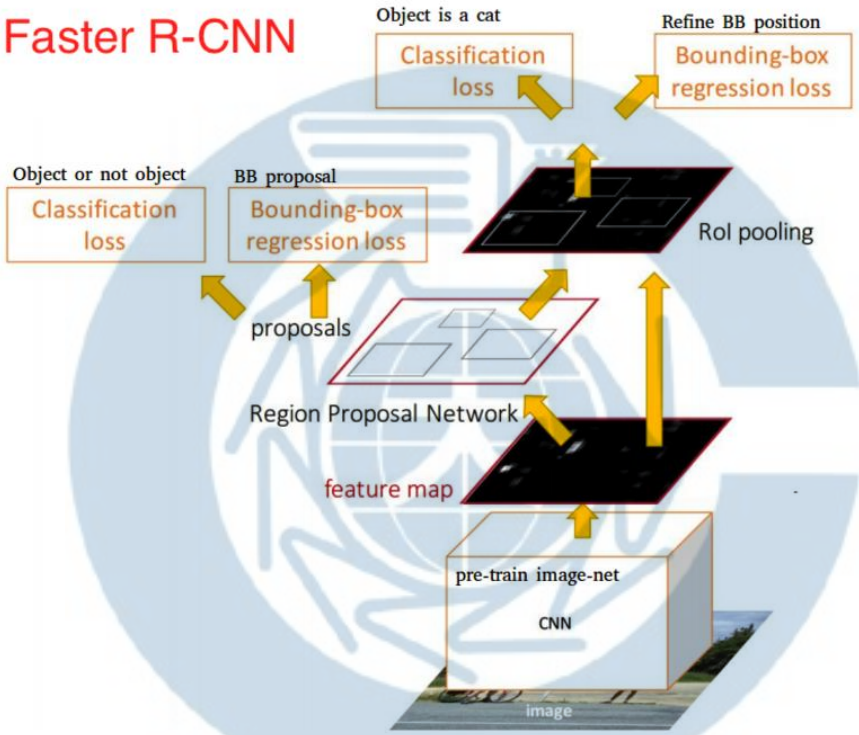


그림 10. Faster R-CNN 전체적인 구조

1×1 컨볼루션이란 Pointwise 컨볼루션이라고도 하는데, 채널을 여러 개 가진 피쳐 맵에 대한 채널 축소의 성격을 가지고 있다. 가령 32×32×512라는 피쳐 맵이 있다고 가정하면, 여기에 1×1 컨볼루션을 거쳐 갈 때 32×32×128, 32×32×1024와 같이 채널 부분의 파라미터 크기를 변경하는 것이 가능하다[29]. 일반적인 컨볼루션의 경우에는 채널의 크기를 맞

춰서 계산하므로 이미지 피쳐 맵의 크기(F×F×N)×필터 맵의 크기(F`×F`×N)만큼 연산을 한다. 반면 1×1 컨볼루션은 각 픽셀에 대해 입력 이미지 파라미터와 필터 파라미터 간에 전결합 계층(Fully connected layer)와 같은 형태가 된다. 즉, CNN 레이어에서 채널 단위의 축소나 확장의 기능을 넣어주는 것과 같다[30].

RPN의 입력값은 Shared 네트워크인 CNN의 출력값, 즉 피쳐 맵을 입력값으로 한다. 그리고 입력값 피쳐 맵에 3×3×256-d(혹은 512. Shared CNN Output 피쳐 맵에 따라 다르게 설정된다.) 필터로 한 번 더 컨볼루션을 거친다. 이 결과, 원래의 피쳐 맵과 동일한 크기의 피쳐 맵이 된다. Sliding window 과정에서 각 앵커(Anchor)마다 RP를 구한다. 그리고 해당 피쳐 맵에 1×1 Conv를 수행하여 2개의 출력값을 도출한다. 2개의 Output은 각각 10×10×(9×2), 10×10×(9×4)의 크기이다. 9×2는 Anchor×Class, 9×4는 Anchor×4개의 좌표를 의미한다. 두 레이어는 Adjustment 레이어 역할로, 출력값 벡터들은 앞서 생성한 RP와 클래스의 Delta 값들을 의미한다.

이 단계에서 생성된 2개의 Output은 각각 물체인지 아닌지 판별하거나, 바운딩 박스를 예측하는 용도로 사용되고, RPN의 유틸리티성에 집중하기 위해 모델을 가볍게 할 목적으로 설계되었다. 위 과정에서 1x1 컨볼루션의 효과로 파라미터 개수가 급감하였고, 이에 따라 RP에 대한 과적합(Overfitting)을 어느 정도 방지한다. 하지만 바운딩 박스 회귀(Bounding box regression)에서는 Initial 한 바운딩 박스를 필요로 하고, Classification 레이어 역시 바운딩 박스가 필요하다. 기존의 Fast R-CNN의 경우, Selective search로부터 바운딩 박스의 좌표를 이용해서 학습했지만, Faster R-CNN은 그렇지 않기 때문에 앵커를 사용한다[31].

앵커는 미리 정의된 Rreference 바운딩 박스이다. 다양한 크기와 비율로

N개의 앵커를 미리 정의하고 3×3 필터로 Sliding Window를 할 때, Sliding마다 N개의 바운딩 박스 후보를 생성하는 것이다. 다음 그림으로 직관적으로 이해할 수 있다.

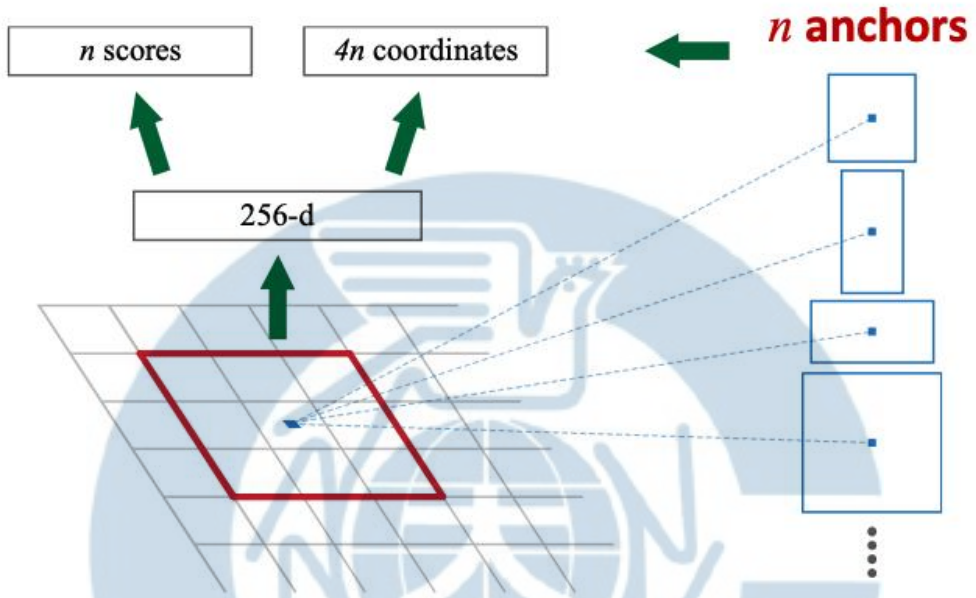


그림 11. Faster R-CNN의 앵커

그림 11을 참고 시, Window sliding마다 물체인 앵커마다 2개 또는 4개의 값을 가지게 되고, 각 앵커마다 Positive나 Negative 라벨을 지정하여 이를 트레이닝 세트로 선정하여 Classifier와 Regressor를 학습한다. 예시에서는 128x128, 256x256, 512x512 3개의 크기와 2:1, 1:1, 1:2 3개의 비율을 이용하여 9개의 Pre-defined 앵커를 사용하였다.

앵커마다 Positive 라벨을 달아주는 기준은 2가지이다. 첫 번째는 가장 IoU가 높은 앵커, 그리고 두 번째 기준은 IoU가 0.7 이상인 앵커이다. Negative 라벨의 경우엔 Positive 라벨을 결정하는 것과 정반대의 기준을

쓰고 임곗값은 0.3에 해당한다. 중간에 속한 IoU들은 라벨을 달지 않고 학습에 포함 시키지 않는다. 주의할 점은 대부분 앵커는 Negative 샘플일 것이기 때문에 샘플링도 다시 비율을 맞춰서 학습을 진행해야 한다. 그리고 Positive 샘플의 경우 전경이라고 판단하고 반대의 경우는 배경이라고 판단한다[32].

2.2.4 Dropout

Dropout 기법은 딥러닝 학습 시 과적합 문제가 발생했을 때, 해결하는 방법의 하나다. 그림 12의 좌측과 같이 2개의 은닉층을 보유한 신경망의 경우, Dropout이 적용되면 그림 13처럼 노드 중 일부를 무작위로 선택하여 사용하게 되는 것이다. 즉, 신경망을 학습할 때 은닉층이 보유한 모든 노드가 사용되는 것이 아닌, 이 중 일부만 무작위로 선택되어 사용하는 것을 의미한다. 하지만 생략된 노드는 학습 시, 별다른 영향을 주지 않으며 보통 절반에서 80%가량의 노드를 사용한다.

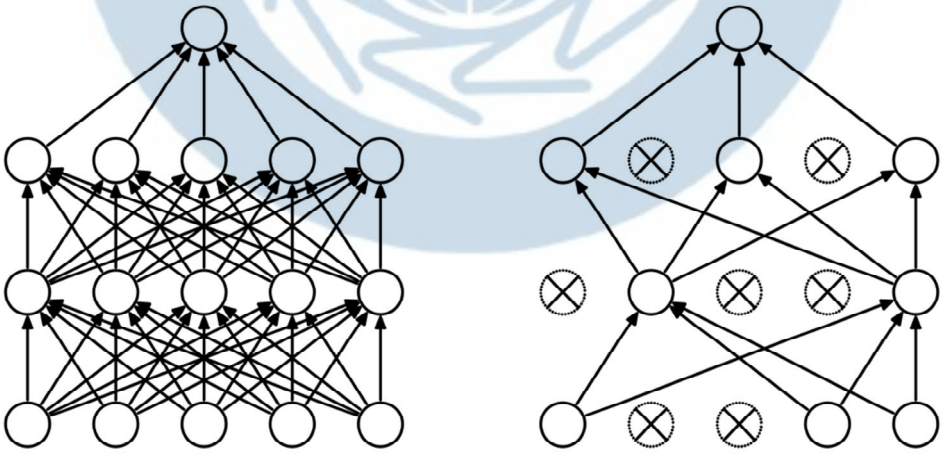


그림 12. 좌: Dropout 사용 전 2개의 은닉층을 포함한 신경망, 우: Dropout 사용 후의 신경망 형태

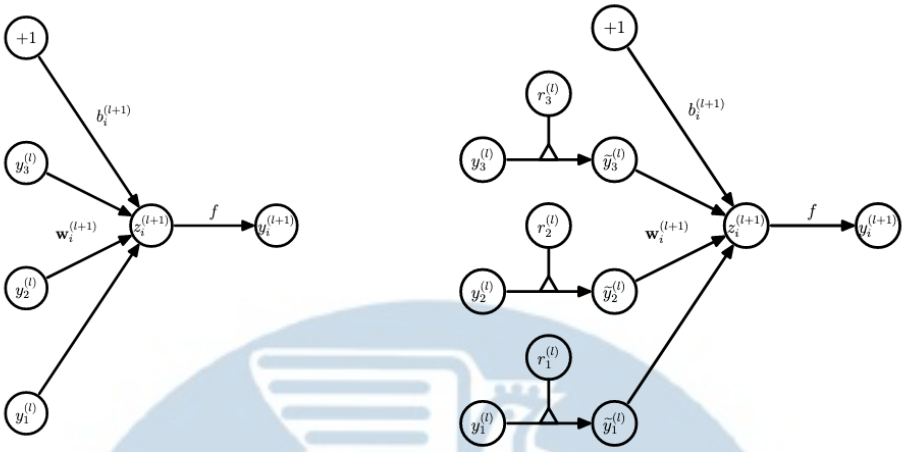


그림 13. 좌: Dropout 적용 전 일반 신경망, 우: Dropout 적용 후의 신경망

그림 13은 일반 신경망과 일반 신경망에 Dropout을 적용한 후를 보여주는데, 우측의 그림을 보면 Dropout의 적용은 신경망에 베르누이의 분포에 해당하는 변수를 무작위로 곱셈 적용하는 것이다. 여기서 베르누이 랜덤 변수는 노드가 존재할 확률이 p, 평균이 p, 일 때 분산이 $p(1-p)$인 변수이다. 입력 노드에 랜덤 변수를 기입하여 곱하게 되면, 최종적으로 랜덤 변수의 수에 따라 노드가 줄어드는 것이다.

2.3 연구 목표

2020년에 들어서 COVID-19로 인해 언택트(Untact) 시대로 도입하여, 이에 따라 집에 있는 시간이 늘어나면서 어린이를 뜻하는 '키드(Kid)'와 어른을 의미하는 '어덜트'(Adult)의 합성어로 아이 같은 감성과 취향을 지닌 어른을 뜻하는 신조어인 '키덜트'가 늘어나고 있고, 이들의 문화 대표주자인 레고가 해를 거듭하여 매출량을 기록하며 성장세를 이어나가고 있다

[33-35]. 이에 따라 레고와 관련된 콘텐츠도 생겨나며, 최근에는 조립에 도움을 주는 어플리케이션까지 등장했다. 하지만 이는 정적인 이미지에 대한 객체 검출 기술을 사용하여, 유저가 현재 어디를 보고 있는지 찾은 블록은 어디 있는지 한 번만 한 눈을 팔아도 알기가 힘들다. 이를 위해 본 논문의 조립 가이드 시스템을 활용하여 레고 블록에 대한 다중 검출과 검출된 블록을 중심으로 조립 가능한 모델을 추천해주는 시스템을 제안한다. 우선 해당 시스템에서 흩뿌려진 레고를 객체 검출을 통해 어떤 블록이 몇 개 있는지에 대해 표기해주고, 이를 종합하여 조립 가능한 모델을 리스트로 보여주는 것이 첫 단계이다. 이후 선택한 모델에 대해 조립 과정을 이미지로 알려주고, 현재 조립 과정에 필요한 블록에 대해서 따로 검출을 통해 실시간으로 위치를 알려주도록 가이드를 제작하였다. 본 논문에서는 레고 블록에 대한 검출기의 속도와 정확도 향상을 목표로 하고 있으므로, 객체 검출 알고리즘 YOLOv4와 Faster R-CNN을 비교해 더 나은 알고리즘 선별 후 Dropout을 적용한 YOLOv4와 비교하여 적용하고자 한다.

Ⅲ. 시스템 구현

3.1 학습용 데이터 구축

본 연구에서는 올바른 레고 블록의 검출을 위해서 학습용 데이터를 4단계의 공정을 통해 구축한다. 첫 번째 단계에서는 데이터를 수집하는 것으로, 데이터 수집이 용이하도록 설치된 플랫폼에서 스마트폰을 사용하여 직접 필요한 레고 블록을 촬영하여 데이터를 수집하였다. 그리고 나머지 부족한 데이터는 가상에서 제작된 레고 합성 이미지 데이터를 통해 보강하였다. 두 번째 단계는 전용 프로그램을 통해 이미지마다 각각의 레고 블록에 일치하도록 라벨링 하는 과정을 거쳤다. 세 번째 데이터 가공 단계에서는 라벨링 작업이 끝난 데이터에 대해서 현재 수집한 데이터로 부족한 면을 보완하기 위해 데이터를 증강했다. 네 번째 변환 단계에서는 실제 데이터와 합성 이미지 데이터들을 학습하고자 하는 알고리즘에 맞게 데이터 세트를 변환하는 작업으로, 실제 데이터는 Roboflow 사이트에서 변환하였고, 합성 이미지 데이터는 변환에 필요한 정보를 파싱하여 변환을 진행해서 학습용 데이터를 구축하였다.

3.1.1 데이터 수집

데이터 수집은 실제 데이터의 경우 그림 15에서 볼 수 있듯이 호환성이 좋고 일반화된 사각형 브릭을 중심으로 흰 블록 2×4×2, 2×2×2와 검은 블록 2×4×2, 2×2×2와 빨간 블록 2×4×2, 2×2×2에 대해 수집을 진행하였다. 그리고 다음 사진과 같이 검출 진행에 있어서 정확한 검출을

위해 잡음이 없는 배경을 따로 만들어 진행하였다. 해당 배경은 데이터 학습에 진행될 레고 블록 검은색, 흰색, 빨간색 중 무채색인 검은색과 흰색을 제외한 빨간색이 눈에 잘 띄는 보색인 청록색을 배경색으로 지정하였다. 그리고 데이터 수집 시 일정한 높이와 각도를 위해서 거치대를 통해 카메라를 고정하고 바닥을 회전시킴으로써 촬영을 그림 14와 같은 환경에서 진행하였다. 또한, 데이터의 중복을 피하고자 데이터 세트에 해당하는 레고 블록이 직사각형임을 고려하여, 360도 회전의 촬영이 아닌 180도 회전하는 촬영 방법으로 시도하였다. 그리고 조밀한 데이터 수집을 위해 일반 사진 촬영이 아닌 동영상 촬영으로 진행하여 이미지로 변환하였다. 그리고 데이터의 절대적인 양을 늘리기 위해 합성 이미지 데이터를 사용하였다. 합성 이미지 데이터는 레고 블록을 가상에서 제작하거나 조립할 때 쓰이는 LDraw 사이트에 의해 제작되었다. 합성 데이터는 그림 16처럼 다양한 환경에서 블록을 다각도로 찍어낼 수 있어 데이터의 양을 늘리는 것이 가능하다. 합성 이미지 데이터로는 레고 블록 30,000개에 해당하는 이미지에 대해 20개 종류의 블록을 사용하였다.

그림 14. 데이터 수집 환경

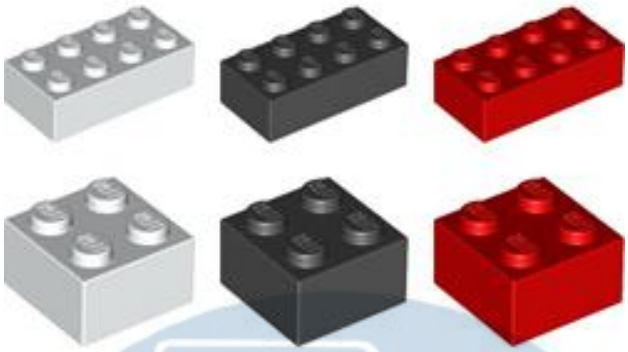


그림 15. 수집한 실제 데이터에 쓰인 블록

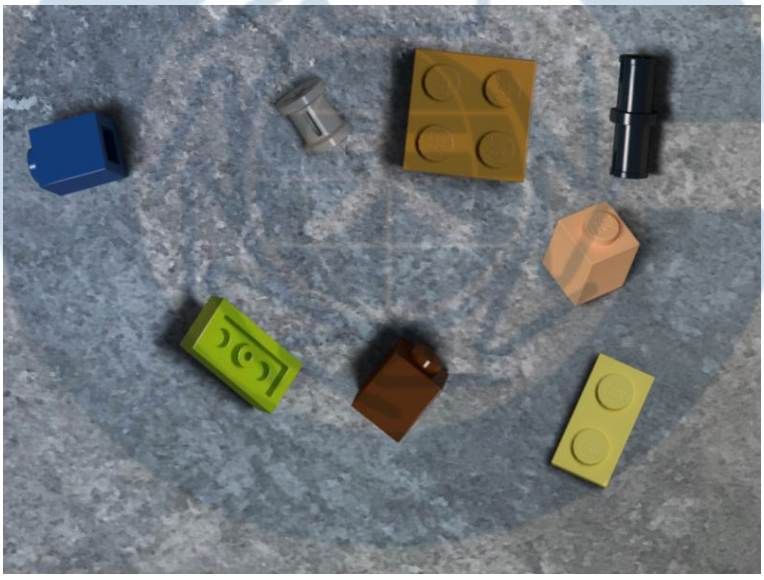


그림 16. 합성 이미지 데이터의 예시

표 1. 데이터 수집 정보

데이터 수집 정보	설명
데이터 수집 장치	갤럭시 Z 플립
데이터 수집 형식	jpg
데이터 수집 방법	거치대를 통해 휴대폰을 고정하여, 일정한 각도마다 레고 블록에 대한 자료 수집
해상도	1440×1440
수평해상도	96DPI
수직 해상도	96DPI
비트 수준	24

3.1.2 데이터 가공 및 증강

3.1.1에서 데이터 학습에 사용할 데이터를 수집하는 환경에 관해 설명하였다. 이를 토대로 진행하여 수집한 데이터들을 영상을 이미지로 변환한 뒤 데이터 학습에 사용할 수 있도록 이미지 데이터를 가공하는 단계를 진행한다.

레고 블록 한 개에 약 25초가량으로 촬영을 시작하였다. 그 후 스크립트 작성을 통해 촬영한 영상을 1초에 25장의 이미지가 생성되도록 나누어 600장의 이미지가 생성되도록 했다. 이미지 간의 차이가 적은 무의미한 데이터로 인한 손실을 피하고자 600장 중 4N 번째마다 이미지를 따로 저장하여 150장을 선별하였으며, 이미지에 포함되지 않은 각도의 데이터를 보완하기

위해 Roboflow 사이트를 통해 데이터 증강 옵션을 활용하여 이미지 크롭 및 좌우 반전, 각도 조절을 사용하였다. 그리하여 여섯 블록에 대해 4,800여 장을 이미지 데이터로 사용하여 라벨링 작업을 진행하였다.

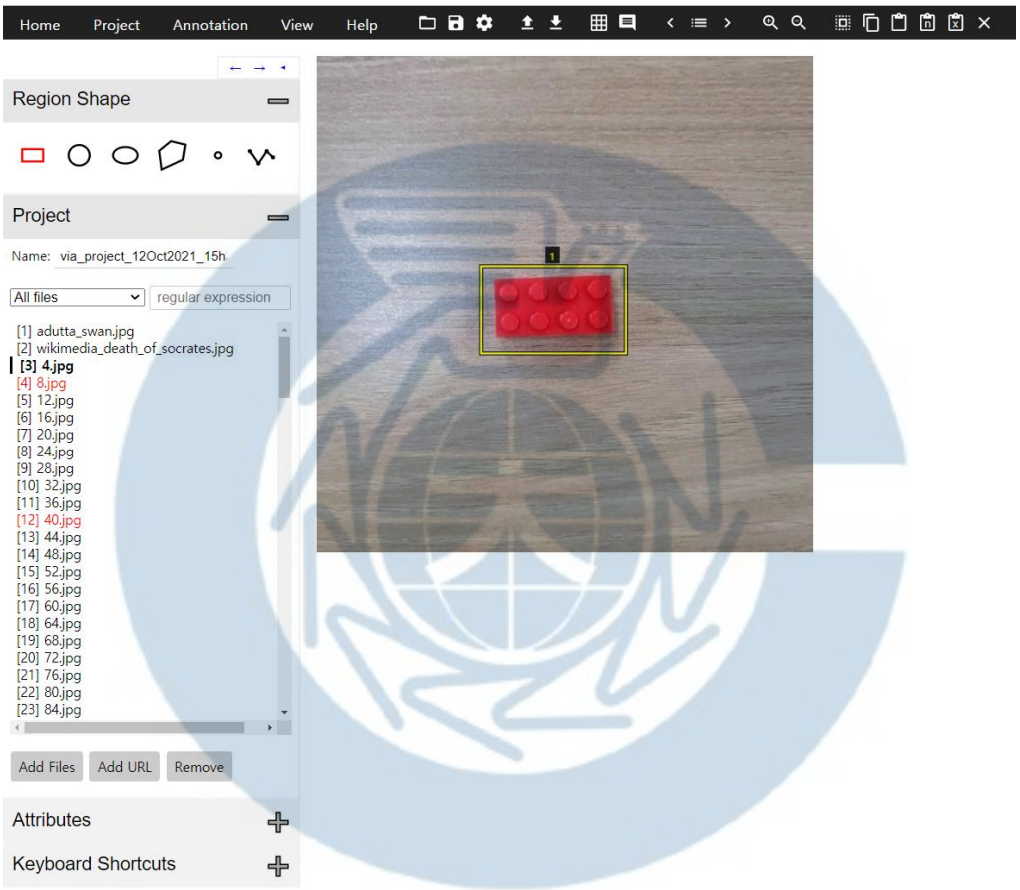


그림 17. VGG Image Annotator를 통한 Bounding Box 영역 표기 예시

라벨링 작업은 전용 프로그램인 VIA(VGG Image Annotator)을 통해 수행하였다. 이 작업은 3단계를 통해 진행한다. 첫 번째로 작업자가 해당 이미지에서 객체를 인식하여 위치를 파악하고, 두 번째는 인식한 객체를 마우스를 통해 영역 선택을 진행한다. 해당 작업은 객체의 영역에 대해 바운딩

박스를 지정하는 단계로 객체가 박스의 영역 안에 들어가도록 마우스를 통해 Annotation 한다. 그 후 세 번째는 영역 내의 객체가 의미하는 것을 선택하여 라벨링을 하는 단계이다. 객체에 대하여 바운딩 박스를 지정하고 라벨링을 한 예시는 그림 17과 같다.

위에서 사용한 라벨링 작업 프로그램인 VIA는 이미지의 영역을 정의하고 해당 영역에 대한 텍스트 설명을 만드는 데 사용할 수 있는 이미지 주석 도구이고, Visual Geometry Group에서 개발되어 BSD-2 조항 라이선스에 따라 출시된 오픈 소스 프로젝트이다. 특징으로는 HTML, CSS 및 Javascript에 기반하고 있고, 오프라인으로도 사용이 가능한 점이 있다. 또한, 지원 가능한 경계 영역 형태로는 직사각형, 원, 타원, 다각형, 점이 있고, cvs와 json 파일 형식의 지역 데이터를 불러오거나 내보내는 것이 가능하다.

3.1.3 데이터 파싱

합성 이미지 데이터의 경우 VOC 파일 형식인데, 파일 형식마다 바운딩 박스의 좌표와 클래스 표기 방식이 다르기 때문에 YOLO에서 사용 가능한 Darknet 파일의 형식으로 변환이 필요하다. 변환할 시, VOC 파일 형식에는 클래스의 목록을 정리한 names 파일이 없어서 Class를 따로 찾아 파일을 만들어 변경해야 한다. 그러나 30,000장이나 되는 데이터 세트에서 일일이 찾아가며 클래스를 맞춰보는 것은 불가능에 가깝다. 그렇기 때문에 데이터 파싱을 통해 names 파일을 일괄 처리하여 변환을 진행하였다. 진행에는 xml 파일의 데이터를 분석하고 다루기 쉬운 파이썬의 모듈 중 하나인 xml.etree.ElementTree를 이용하였다. 이를 통해 xml의 수식에서도 원하는 부분인 names에 대한 데이터를 읽어 들여, 해당 정보만 따로 txt파일 형식으로 저장하였다. 그림 18은 xml 파일 형식의 예시로 해당 과정에서는

중단에 있는 name에 관한 값만 카피하여 따로 저장하였다.

```
<?xml version="1.0" ?>
<annotation>
  <items>None</items>
  <filename>9</filename>
  <path>None</path>
  <source>
    <database>Unspecified</database>
  </source>
  <size>
    <width>2048</width>
    <height>2048</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <lighting>[1005, (52.0, 104.0)]
</lighting>
  <object>
    <name>10247</name>
    <pose>m39</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>146</xmin>
      <ymin>960</ymin>
      <xmax>195</xmax>
      <ymax>1017</ymax>
    </bndbox>
  </object>
  <object>
```

그림 18. xml 파일 형식 예시

3.2 데이터 학습

3.2.1 개발 환경 1 : Colaboratory

본 연구는 Google에서 제공하는 Colab(Colaboratory)에서 진행되었다. Colab을 사용하면 웹 브라우저에서 Python을 작성하고 실행할 수 있다[36]. Colab은 환경을 따로 설정하지 않아도 되고, 심지어 Colab에서 제공하는 GPU를 무료로 사용할 수 있다는 장점이 있다. 또한, Google에서 제공하는 클라우드에서 코드를 실행하기 때문에 현재 컴퓨터의 사양 및 성능과 관계없이 사용할 수 있다. Colab은 현재 텐서플로우, 신경망 개발 및 학습, AI 개발과 같은 머신러닝 분야에서 널리 사용되고 있다.

3.2.2 데이터 학습 1: YOLOv4

데이터 학습 전 전처리 작업이 필요하다. VIA에서 생성한 데이터를 YOLOv4에서 인식 가능한 형태로 데이터를 변환해야 YOLOv4에서 학습데이터로 사용하는 것이 가능하기 때문이다. 본 논문에서는 Roboflow 사이트에서 표 2와 같이 데이터 변환을 진행하였다.

표 2. 이미지 데이터 전처리 : VGG Image Annotator > YOLOv4

변환 전 이미지의 data


```
"batches": [
        "Nfj1JqDomgoPguLhrYn9/IM3KIWZoy1LyNd8OTShi",
        "Nfj1JqDomgoPguLhrYn9/j6lPgvUVCbPRSSDMVRay"
    ],
    "name": "100.jpg",
    "owner": "XbyQ4Vz1pTZl6dNSQGJP",
    "height": 480,
    "r": 0,
    "uploader": "FdUsisgOy1YAlR9fmsljquO7OZt2",
    "width": 480,
    "extension": "jpg",
    "datasets": [
        "Nfj1JqDomgoPguLhrYn9"
    ],
    "split.Nfj1JqDomgoPguLhrYn9": "train",
    "annotations": {
        "yolov4": {
            "used": true,
            "converted": "{₩"key₩":₩"100.jpg₩",₩"boxes₩":[{₩"label₩":₩"2-4-2,red,bric₩",
₩"x₩":225,₩"y₩":239,₩"width₩":136,₩"height₩":126}],₩"width₩":480,₩"height₩":480}",
            "key": [
                "100.jpg"
            ],
            "original": {
                "source": "via",
                "format": "json",
                "annotation": "{₩"fileref₩":₩"₩",₩"size₩":67902,₩"filename₩":₩"100.jpg₩",
₩"base64_img_data₩":₩"₩",₩"file_attributes₩":{},₩"regions₩":{₩"0₩":{₩"shape_attributes₩":
{₩"name₩":₩"rect₩",₩"x₩":157,₩"y₩":176,₩"width₩":136,₩"height₩":126},
₩"region_attributes₩":{₩"display_name₩":₩"2*4*2,red,bric₩"}}}}"
            }
        }
```


변환 후 이미지의 Annotation data


```
100_jpg.rf.121ab9a86f34a28372f390e46941a15b.jpg 136,153,254,262,1
```

YOLOv4 학습은 YOLOv4 검출기 내부에 있는 train.py를 통해 작성된 스크립트에서 진행되었다. 표 3과 같이 train.py 에서는 학습을 진행하는 이미지의 크기와 배치, 그리고 파일의 위치와 구조정보와 같은 파라미터를 제어할 수 있다. 본 연구에서는 학습 과정에서 이미지의 크기를 416×416으로 줄였으며, 이를 통해 학습 시 고해상도 이미지로부터 발생하는 부하를 약간 저하했다. 배치 크기는 일반적으로 진행하는 10과 16을 기준으로 학습을 진행하였고, 에포크는 50회, 100회, 150회로 3회를 학습하였다. 모델은 YOLOv4로 진행하였으며, 가중치 파일도 YOLOv4에 적용 가능한 YOLOv4.py를 사용하였다. 에포크의 경우 학습의 횟수를 의미하고, 배치는 한 번의 학습에 이루어질 데이터의 양을 의미한다. 학습은 알고리즘을 선별하기 위해, 동일한 조건에서 에포크의 수에 변화를 주는 것을 3회, 그리고 배치 크기에 변화를 주는 것을 2회로 진행하였다.

표 3. YOLOv4 학습 파라미터 해석

파라미터	설명
--img	이미지 크기
--epochs	학습 횟수
--batch	배치 크기
--data	학습 데이터
--cfg	모델의 구조정보
--weights	사전 학습된 가중치
--name	결과에 대한 이름 지정

3.2.3 데이터 학습 2: Faster R-CNN

아래의 표 4 또한 데이터의 변환을 보여주는 것으로, Faster R-CNN도 VIA에서 만들어낸 데이터를 Faster R-CNN에서 사용할 수 있도록 COCO 데이터 세트로 변환을 해야 사용할 수 있다. 본 실험에서는 YOLO와 마찬가지로 Roboflow를 통해 데이터 변환을 진행하였다. 변환할 시, 다음의 표 4와 같이 변환이 이루어진다.

이 데이터들을 토대로 파라미터에 적절한 값을 부여하여 학습을 진행하였다. 이 과정에서 수정이 가능한 파라미터의 종류로는 가중치, 클래스의 개수 그리고 카테고리의 수 등이 있다. 해당 모델은 YOLOv4와 테스트 결과를 비교할 수 있도록 동일한 데이터로 학습을 진행하였다. 파라미터는 밑의 표5와 같이 해석할 수 있다.

표 4. 이미지 데이터 전처리 : VGG Image Annotator > Faster R-CNN

변환 전 이미지의 data


```
"batches": [
        "Nfj1JqDomgoPguLhrYn9/lM3KlWZoy1LyNd8OTShi",
        "Nfj1JqDomgoPguLhrYn9/j6lPgvUVCbPRSSDMVRay"
    ],
    "name": "100.jpg",
    "owner": "XbyQ4Vz1pTZI6dNSQGJP",
    "height": 480,
    "r": 0,
    "uploader": "FdUsisgOy1YAlR9fmsljquO7OZt2",
    "width": 480,
    "extension": "jpg",
    "datasets": [
        "Nfj1JqDomgoPguLhrYn9"
    ],
    "split.Nfj1JqDomgoPguLhrYn9": "train",
    "annotations": {
        "yolov4": {
            "used": true,
            "converted": "{₩"key₩":₩"100.jpg₩",₩"boxes₩":[{₩"label₩":₩"2-4-2,red,bric₩",
₩"x₩":225,₩"y₩":239,₩"width₩":136,₩"height₩":126}],₩"width₩":480,₩"height₩":480}",
            "key": [
                "100.jpg"
            ],
            "original": {
                "source": "via",
                "format": "json",
                "annotation": "{₩"fileref₩":₩"₩",₩"size₩":67902,₩"filename₩":₩"100.jpg₩",
₩"base64_img_data₩":₩"₩",₩"file_attributes₩":{},₩"regions₩":{₩"0₩":{₩"shape_attributes₩":
{₩"name₩":₩"rect₩",₩"x₩":157,₩"y₩":176,₩"width₩":136,₩"height₩":126},
₩"region_attributes₩":{₩"display_name₩":₩"2*4*2,red,bric₩"}}}}"
            }
        }
```


변환 후 이미지의 Annotation data


```
    "images": [
        {
            "id": 0,
            "license": 1,
            "file_name": "100_jpg.rf.121ab9a86f34a28372f390e46941a15b.jpg",
            "height": 416,
            "width": 416,
            "date_captured": "2021-10-22T11:07:38+00:00"
        }
```

표 5. Faster R-CNN 학습 파라미터 해석

파라미터	설명
cfg.MODEL.WEIGHTS	모델에 적용 가능한 가중치 선택
cfg.merge_from_fle	학습하고자 하는 모델 선택
cfg.MODEL_HEADS.NUM_CLASSES	클래스의 개수
cfg.SOLVER.MAX_ITER	인터랙션의 수

3.3 시스템 제작

3.3.1 시스템 설계

Unity는 게임 개발을 위해 상용화되어있는 엔진으로 멀티플랫폼을 통해 다양하게 응용할 수 있으며, 무료로 이용할 수 있는 특징이 있다. 현재에는 게임뿐만 아니라 영화나 애니메이션, 건축 등 다양한 분야에서 개발할 수 있도록 이용되고 있다. 또한, 객체 지향적인 프로그래밍이기 때문에 모든 물체를 객체로 표현할 수 있고, 전문적인 프로그래머가 아닌 초보 프로그래머라도 간단한 문법을 통해서 쉽게 구현할 수 있다는 장점이 있다. 이러한 Unity 플랫폼에 객체 검출 알고리즘을 적용하여, 레고 블록 인식 및 분류 기법을 사용한 시스템을 설계하였다. 다음의 그림 19은 해당 시스템의 시나리오를 보여준다.

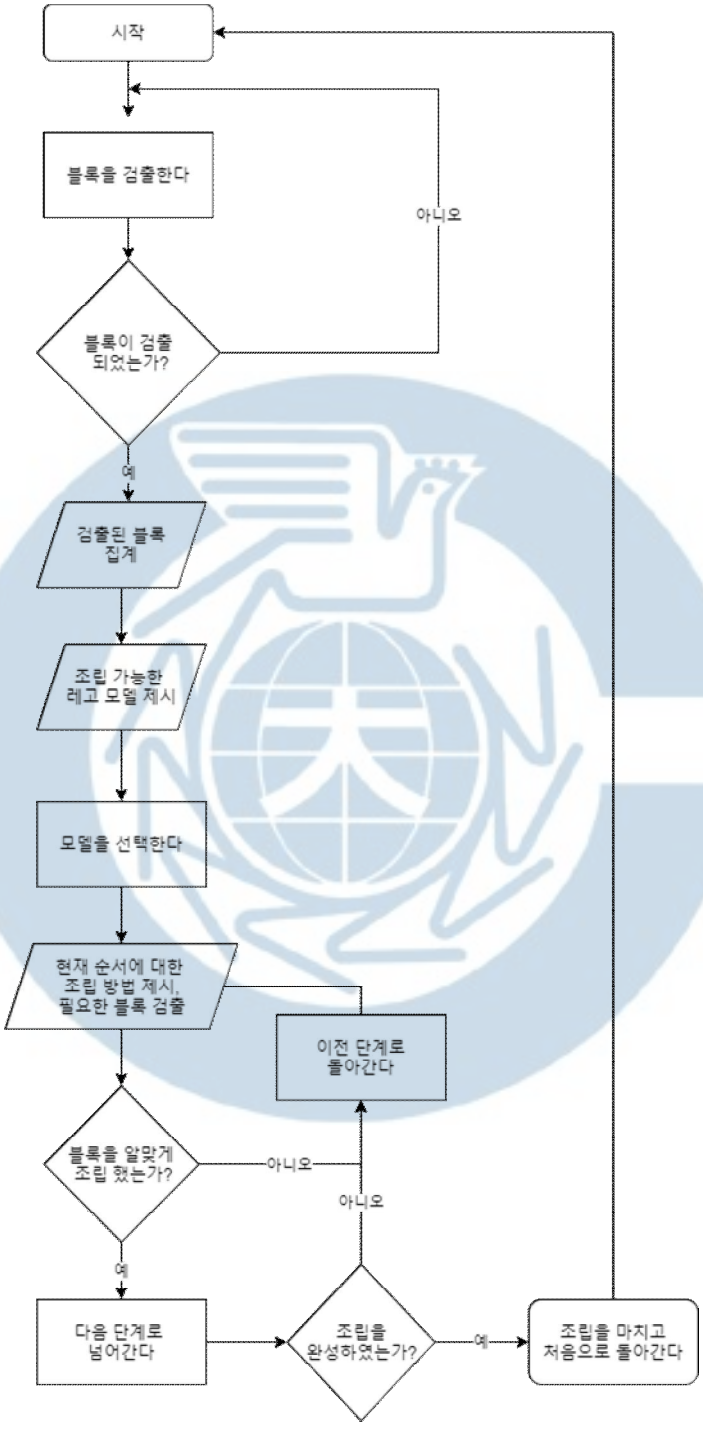


그림 19. 시스템 시나리오

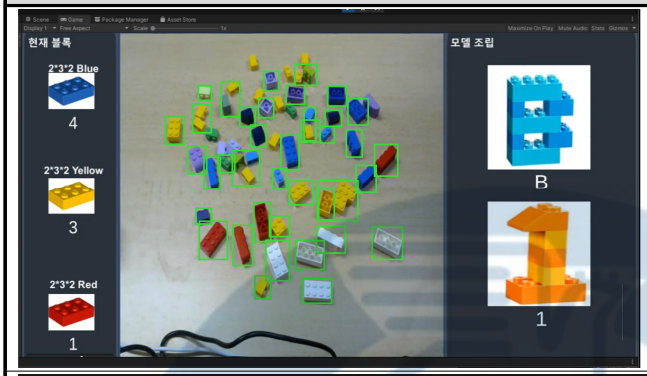
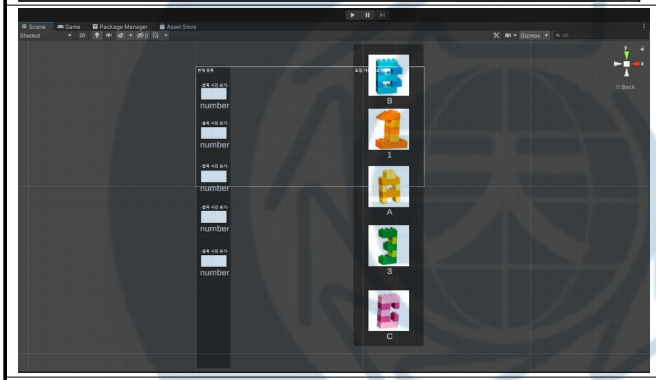
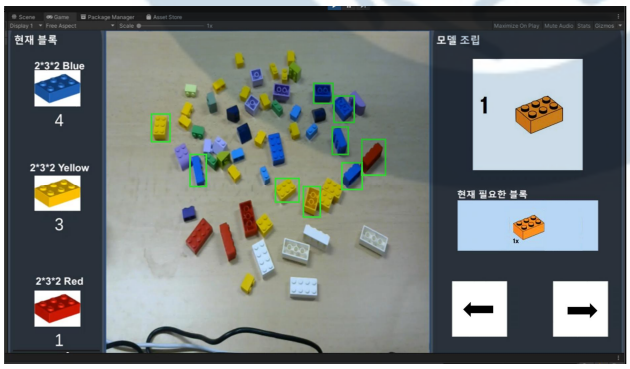
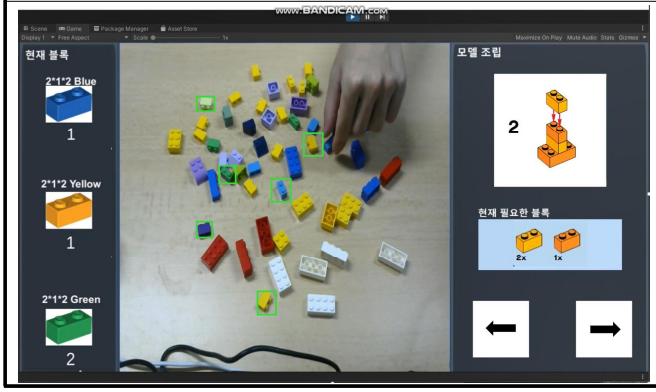
해당 시스템을 실행하게 되면 웹캠에 의해 현재 영상이 시스템으로 전송되어 이미 학습되어있는 객체 검출 알고리즘에 의해 블록을 검출하게 된다. 그리고 검출된 블록을 집계하여 조립 가능한 모델을 제시하여 준다. 조립 가능한 모델 중 하나를 선택하게 되면 조립하는 방법이 나타나게 된다. 조립 방법 단계에서는 조립 순서에 따른 조립 설명과 해당 순서에 필요한 블록을 검출해주어 조립 시 도움을 준다. 현재 자신의 조립 단계에 따라 조립했을 경우 화살표 버튼을 통해 다음 단계로 넘어갈 수 있고, 혹여나 이전 단계로 되돌아가야 할 때도 화살표를 통해 돌아갈 수 있다. 그리고 제시된 시스템대로 조립을 진행하여 완성하였을 경우 End 버튼을 통해 현재 조립을 끝마치고 다시 처음의 전체 블록을 검출하는 화면으로 되돌아가게 된다. 그리고 다음 표는 해당 시나리오가 작동하는 시스템 UI에 대한 구성을 보여준다.

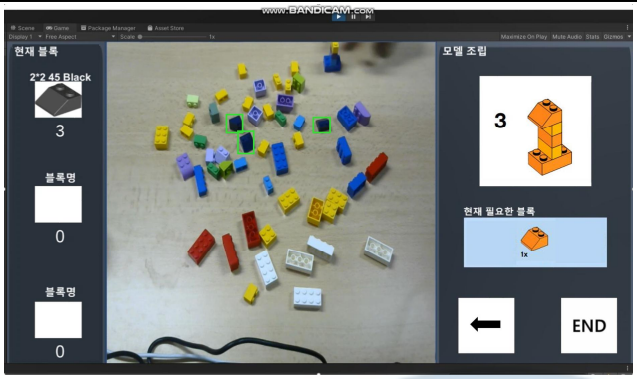
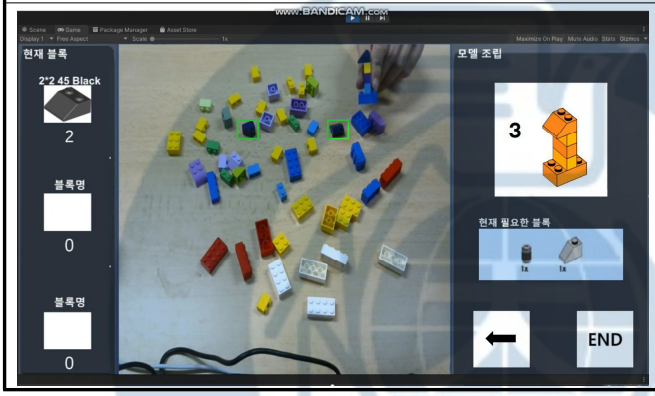
3.3.2 시스템 수행

그림 20. 시스템 수행 환경

본 논문에서는 그림 20과 같은 환경에서 레고 조립 시스템을 진행하였다. 모니터에 웹캠을 단 상태에서 웹캠이 책상 위 블록을 검출할 수 있도록 각도를 조정한 뒤, 블록을 검출하여 시스템을 통해 조립을 진행하였다. 본 논문에서 제안한 시스템을 통해 레고 모델을 조립하는 과정은 다음의 표와 같다.

표 6. 시스템 수행 과정

시스템 수행	설명
	중앙의 캠 화면을 통해 검출된 블록의 종류와 수를 왼쪽의 칸에 표기하고, 이를 중심으로 조립 가능한 모델을 오른쪽에 표기
	칸이 부족하여 보이지 않을 경우 드래그를 통해 스크롤을 이동하여 추가로 볼 수 있음
	조립 가능한 모델 선택 시, 오른쪽 칸에 순서대로 조립 장면과 현재 필요한 블록을 표기 필요한 블록만을 캠에서 검출해주고, 왼쪽의 칸에 이를 표기 화살표를 누를 시 다음 순서로 이동
	다음 단계의 조립 장면, 필요한 블록 표기 및 검출

	마지막 단계의 조립 장면, 필요한 블록 표기 및 검출
	끝까지 조립을 완성한 모습

우선 시스템의 UI는 Cam에서 받은 영상 이미지를 출력하여 객체 검출이 실행된 화면을 중앙에서 보여주는 것으로 시작한다. 검출된 블록을 종합하여 왼쪽의 칸에 표기해주고, 이를 종합하여 조립 가능한 모델들을 오른쪽 칸에 표기하여준다. 조립하고자 하는 모델을 선택하면 조립 순서별로 조립 설명 이미지와 현재 필요한 블록이 표기된다. 그리고 캠에서 현재 필요한 블록을 중심으로 해당 블록만 따로 검출하여 보여준다. 조립했을 시, 이전 단계나 다음 단계로 화살표 버튼을 통해 이동할 수 있도록 제작하였다. 조립을 끝까지 마쳤을 시에는 End 버튼을 통해 다시 조립 가능한 모델을 표기해주는 화면으로 이동할 수 있도록 하였다.

이를 통해 위와 같은 환경에서 시스템 수행을 한 결과, 현재 가진 레고

블록을 통해 실시간으로 조립 가능한 모델을 추천받고, 알려주는 블록을 따라 조립을 진행하여 정상적으로 모델을 완성할 수 있었다.

Ⅳ. 실험 결과

4.1 성능 지표

첫 지표는 정밀도(Precision)과 재현율(Recall)이다. 정밀도는 검출한 모든 결과 중에서 옳게 검출한 비율을 말한다. 각각은 다음의 정밀도와 재현율에 관한 식처럼 표현할 수 있는데, TP(True Positive)는 옳게 검출한다는 뜻이며, FP(False Positive)는 틀린 검출을 의미한다. 정밀도는 즉 객체 검출이 진행된 것 중에서 제대로 검출이 된 비율을 의미하는 바이다. 재현율은 검출이 되어야 하는 물체 중 검출이 제대로 실행된 비율을 말한다. FN은 False Negative의 약자로 검출이 되어야 하는 물체이지만, 정작 검출되지 않은 것을 의미한다[39].

$$\mathrm{Pr}ecision = \frac{TP}{TP+FP} \quad Recall = \frac{TP}{TP+FN}$$

정밀도와 재현율에 관한 식 (1)

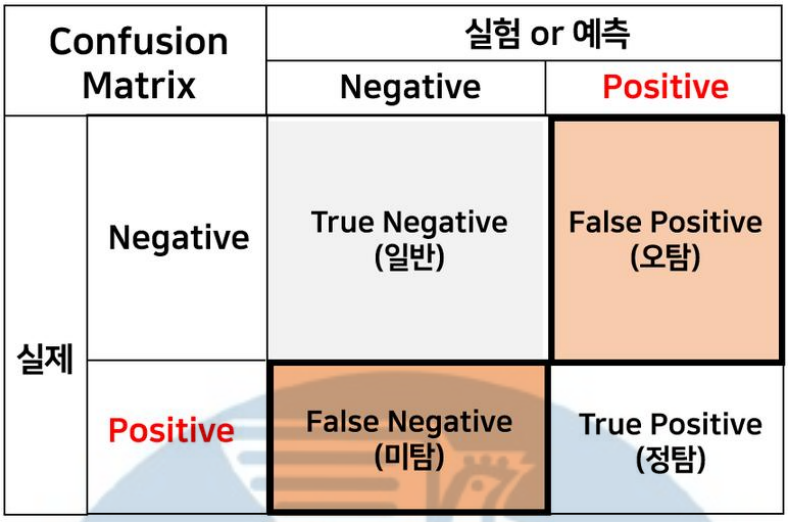


그림 21. TP FP FN TN 표

정밀도과 재현율의 값은 결과가 항상 0에서 1 사이에서 정해진다. 이 둘의 관계를 보면 정밀도가 높으면 재현율이 낮고, 반대로 재현율이 높으면 정밀도가 낮은 편이다. 그래서 이 중 하나의 값으로만 성능을 평가하기에는 무리가 있고, 두 값을 모두 활용해야 할 필요가 있다. 그래서 필요한 것이 PR(정밀도-재현율) 곡선 및 AP이다.

PR 곡선은 Confidence 수준에 따른 임곗값에 해당하는 값의 변화에 대해 객체 검출 알고리즘의 성능을 평가하는 방법이다. 우선 Confidence는 검출기가 검출한 결과에 대해 얼마나 확신을 가지고 검출을 했느냐에 대한 것을 값으로 나타낸 것이다. 최댓값이 1이고 최솟값이 0으로, 어떤 물체를 검출했을 시 Confidence 값이 0.999를 측정했다면 그것은 아주 큰 확신을 가지고 검출을 했다는 것을 의미한다. Confidence 수준이 높다는 것이 높은 검출 정확도와 직결되는 것은 아니다. 알고리즘이 자체적으로 판단하여 추측하여 판단하는 것이기 때문이다. 그래서 Confidence가 낮다는 것은 검출한 것에 확신이 없다는 것을 뜻한다. 그래서 유저들은 대부분

Confidence 수준에 임곗값 값을 지정하여 검출 시 특정 값이 나와야 올바른 검출을 했다고 인정하도록 하였다. 따라서 Confidence 수준에 따른 임곗값의 변화는 정밀도과 재현율 변화로 이어질 것이고, 이것이 그래프가 된 것이 PR 곡선이다. 이 PR 곡선에서 x와 y 각 축은 재현율과 정밀도을 의미하며, 이를 통해 각 축의 변화에 따른 상관관계를 파악할 수 있다.

4.2 YOLOv4 결과 비교

YOLOv4 학습결과를 Tensorboard를 통해 모니터링 하였다. 각각 동일한 데이터 세트에서 동일한 Train, Vaild, Test 데이터 파일 하에 학습되어 졌다. 하늘색 그래프의 경우, Img 크기 416*416, 에포크 100회, 배치 크기 16으로 지정하였고, 주황색의 경우 Img 크기 416*416, 에포크 150회, 배치 크기 16으로 지정하였으며, 주황색은 Img 크기 416*416, 에포크 50회, 배치 크기 10으로 진행하였다.

에포크 횟수가 100회일 때까지는 mAP가 높아지는 반면에 150회까지 늘렸을 땐 오히려 mAP가 줄어들고, 정확도가 낮아졌기 때문에 150회까지의 학습은 바르지 않은 것으로 판단이 된다. 더군다나 낮아진 수치로 인해 오히려 과적합(Overfitting)이 의심되고, 배치 크기가 낮을 땐 오히려 수치가 작게나마 올라갔다.

다음의 그림 22에 정밀도의 경우에는 16의 배치 크기와 에포크는 100인 모델이 0.8751로 가장 높은 성능을 보인다. 또한, 배치 크기를 줄인 경우에는 학습 성능이 낮은 모습을 보였다.

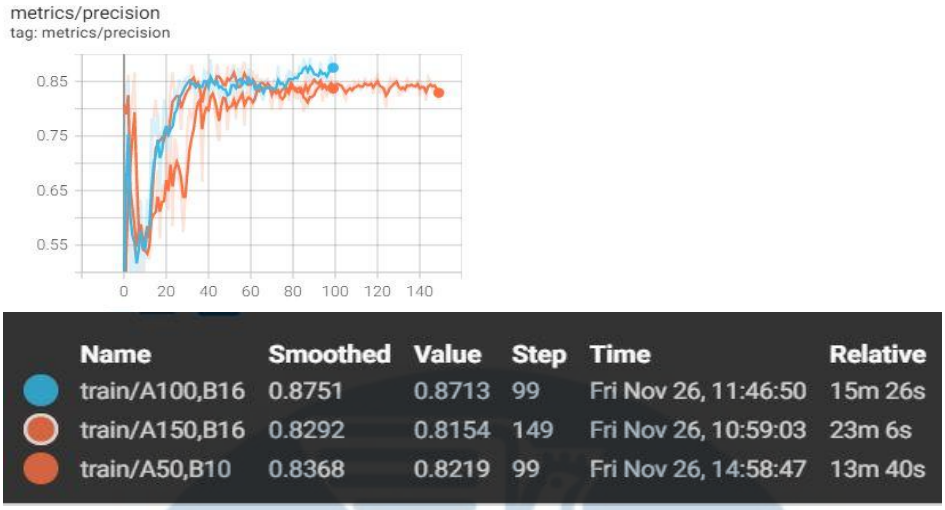


그림 22. YOLOv4 학습 정밀도 그래프

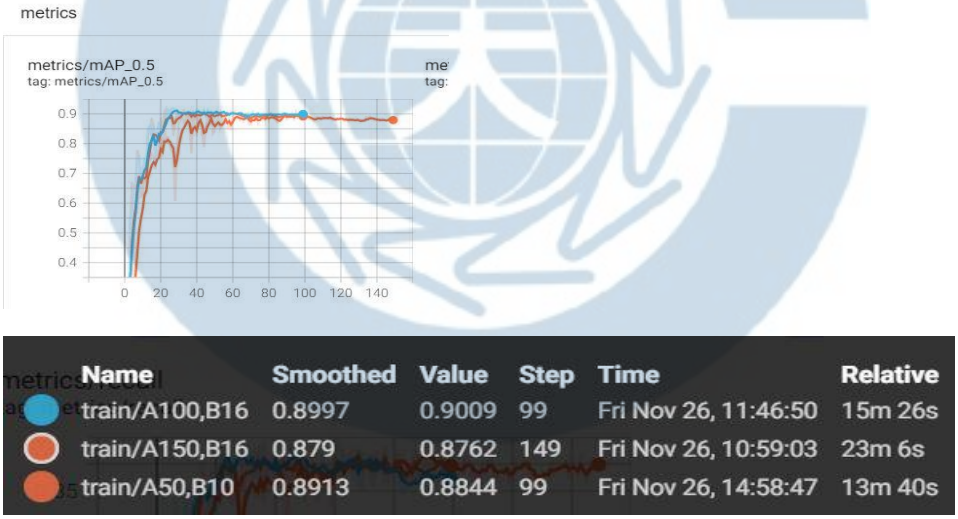


그림 23. YOLOv4 학습 mAP_0.5 그래프

그림 23의 mAP_0.5의 경우, 16의 배치 크기와 에포크가 100인 모델이 높은 성능을 보여주었다. 0.9009로 가장 높은 성능을 보였다.

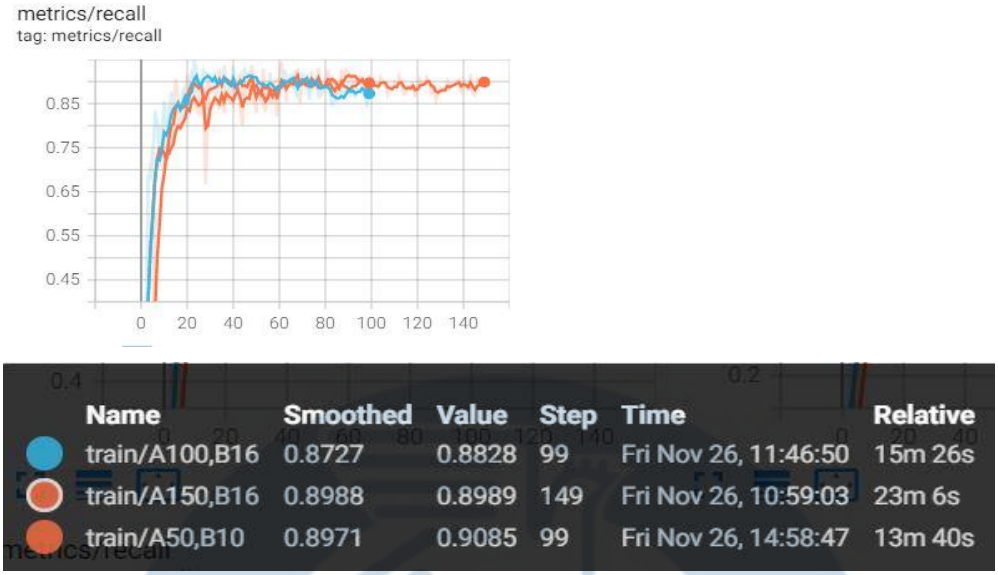


그림 24. YOLOv4 학습 재현율 그래프

그림 24의 재현율 지표의 경우 앞의 그림 22, 23의 그래프와 다르게 대체로 모든 모델이 유사한 성능을 보여주고 있어, 재현율의 경우 별다른 차이가 없는 것을 알 수 있다.

4.3 Faster R-CNN 결과 비교

YOLOv4 모델의 성능 비교를 위해 객체 검출이 가능한 Faster R-CNN을 비교 대상으로 삼아, 동일한 공간에서 동일한 개수의 블록을 촬영한 사진에 대해 객체 검출의 결과를 비교하였다. 결과적으로 YOLOv4 모델이 더 정확한 검출을 하고 있다는 나타내고 있다.

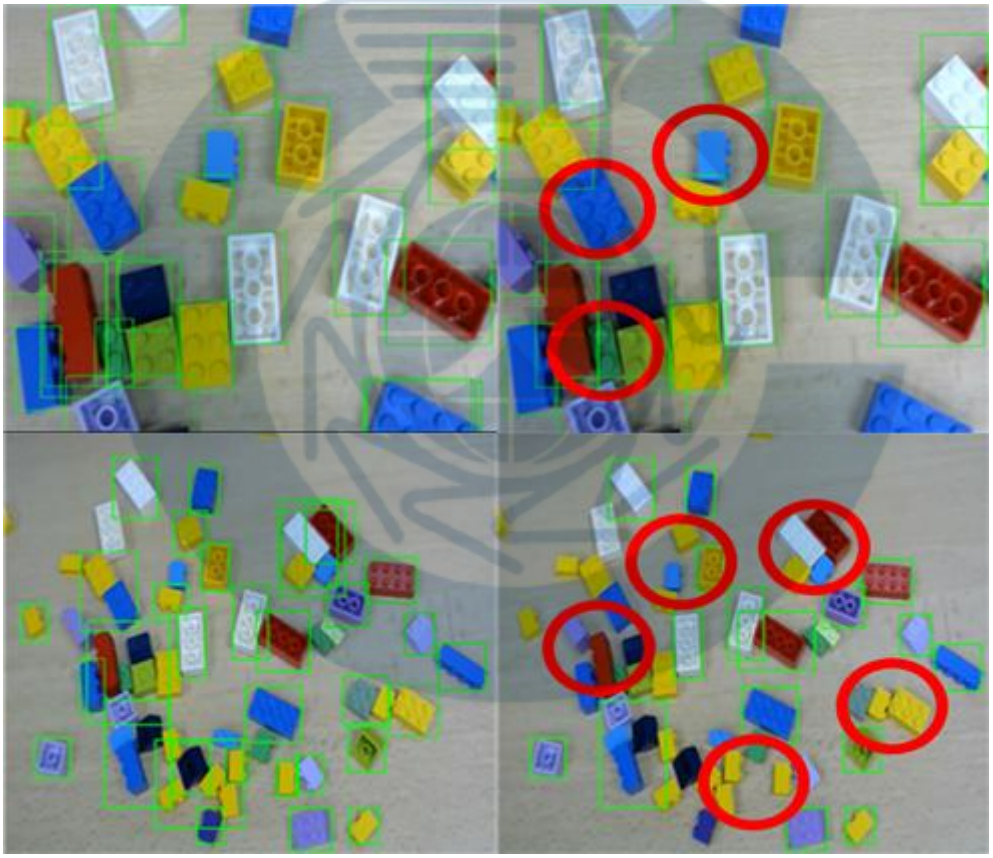


그림 25. 같은 사진의 YOLOv4(좌)와 Faster R-CNN(우) 모델 결과 비교

YOLOv4 모델의 성능을 비교하기 위해 Faster R-CNN 모델과 비교해보면, 그림 25와 그림 26의 레고 검출률의 차이에서 눈에 보일 확연한 차이를 보

이고 있다. YOLOv4가 검출률이 낮은 블록인 2×1×2블록을 검출할 수 있을 정도의 거리임에도 Faster R-CNN은 검출률이 낮은 모습을 보인 반면 YOLOv4의 경우 시스템 실행 시 보인 2×1×2 크기의 블록에 대해 검출률이 낮은 모습을 보인 것을 제외하고는 대부분을 검출해내었다. 다만 비교 실험 도중 YOLOv4와 Faster R-CNN 두 알고리즘이 모두 위에서 진행한 검출에 비해 45도 측면에서의 검출에서 약한 검출률을 보였다. 측면에서는 겹쳐져 있거나 블록을 가리는 경우가 있는데 실제로는 약간의 거리가 있는 블록도 겹쳐 보이기 때문에 위에서 아래로 찍은 사진을 비교할 때에 비해 낮은 검출률을 보였다.

4.4 Dropout을 적용한 YOLOv4 결과 비교

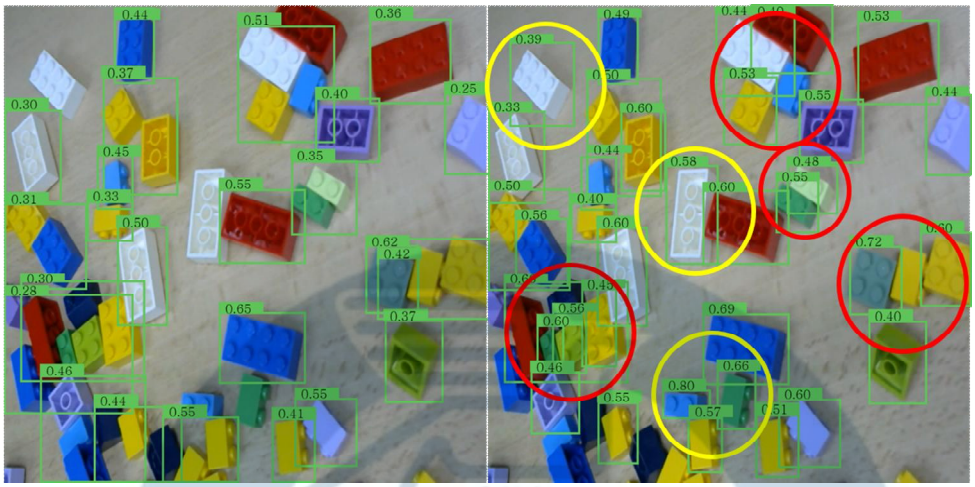


그림 26. 객체 검출 모델에 대한 레고 블록 검출 정확도 비교

4.3 파트의 비교에서 나온 YOLOv4의 결과를 Dropout을 진행한 YOLOv4와 비교하였을 경우, 그 차이가 확연하게 드러나는 것을 볼 수 있었다. 그림 27에서 좌측의 그림에서 나타나는 기존의 YOLOv4는 앞의 그림 25, 26에서 Faster R-CNN보다 검출률이 높아 밀집된 블록도 검출하였지만, 밀집된 블록을 하나로 인식하여 개별로 구분하는 것에선 낮은 성능을 보였다. 반면 그림 27의 우측에 표기되어있는 본 논문에서 사용할 YOLOv4는 붉은색 원을 확인할 시, 밀집되어있는 레고 블록도 기존의 YOLOv4와 다르게 대부분을 개별 블록으로 인식하는 결과를 볼 수 있었다. 또한, 노란색 원에서는 2×1×2 같은 작은 블록과 이전에 검출하지 못했던 블록 또한 검출하는 것을 보여주었다. 그리고 밀집된 부분의 레고 블록의 정확도 수치를 비교할 시, 둘 다 올바르게 검출이 된 블록의 정확도 수치에서 적게는 3%에서 크게는 10%가 넘도록 차이가 났고, 평균적으로는 8%가 차이가 발생했기 때문에, 유의미한 차이가 있는 것을 확인할 수 있었다.

Ⅴ. 결론

본 연구에서는 객체 검출 기술을 Unity에서 적용하여, 레고 블록을 실시간 객체 검출로 인식하고 분류할 수 있는 기법을 이용한 조립 시스템을 구현하고 수행하였다. 해당 시스템에서는 현재 가지고 있는 블록을 검출함으로써 이를 집계하여 어떤 모델을 만들 수 있는지 알려주며, 모델 조립 시 필요한 레고 블록의 위치 정보를 알려주도록 제작했다. 시스템에 사용될 알고리즘의 선별을 위해 YOLOv4와 Faster R-CNN 객체 검출 알고리즘이 본 논문에 사용되었다. YOLOv4의 하이퍼 파라미터를 변경하여 각각의 데이터를 비교를 통해 더 나은 경우를 선별하였고, 이를 Faster R-CNN과 비교하여 더 나은 알고리즘을 다시 한번 선별하였다. 그리고 여기서 더 나아가 알고리즘의 정확도를 개선 시키도록 하였다.

시스템의 수행에는 YOLOv4를 사용하였다. 검출된 블록을 통해 다음 단계로 넘어가는데 시스템이 원활하게 작동하였다. 레고 블록들을 평평하게 펼치고 가이드 시스템에 따라 하나의 모델을 조립하는 것까지 수행하였다. 우선 YOLOv4와 Faster R-CNN을 동일한 환경에서 같은 수의 블록에 대해 검출 성능을 비교한 결과 검출률과 정확도 두 부분에서 모두 YOLOv4가 우수한 것이 결과로 나왔다. 하지만 YOLOv4는 높은 검출률에 비해 둘 이상이 조밀하게 붙어있는 블록의 경우 하나의 블록으로 보는 경향이 강했고, YOLO의 단점인 작은 물체 감지에 힘들다는 점으로 인해 2×1×2 정도의 작은 블록은 2×4×2나 2×2×2 크기의 블록에 비해 검출률이 낮았다. 이와 같은 문제를 해결하기 위해 Dropout 기법을 적용하여 정확도를 향상시켰다. 이를 통해 레고 블록의 과적합에 따른 새로운 데이터를 인식하지 못하는 문제를 해결하였다. 가령 밀집된 레고를 하나로 보는 문제를 개별로 인식할 수 있었고, 같은 장면에서의 정확도를 기존의 YOLOv4와 비교하였을 때 수치상으로 평균 8%가 오른 것을 확인할 수 있었다.

ABSTRACT

Proposal of Real-Time Lego Block Recognition and Classification Technique Using Deep Learning-Based Object Detection Technology

Hyeongjin Kim
Dept. of Culture and Technology Convergence
Graduate School of
Changwon National University

Object detection technology is a technology for classifying location information and types of desired objects in the field of computer vision. Object detection is being applied in various fields such as autonomous driving and license plate recognition. However, this object detection technology showed insufficient performance in practice even though it has infinite applications in the case of multi-object detection or tracking. However, with the advent of CNN, research to improve the insufficient performance was conducted, and accordingly, YOLO, which is useful for multi-object detection, appeared. Existing CNNs had the disadvantage of being slow, but YOLO compensated for them. Through this, in this paper, we propose a

system that recommends multiple detection of Lego blocks and a model that can be assembled around the detected blocks so that a system useful for multi-object detection can be produced and used using YOLO. The system detects the Legos scattered on the floor so that the number and types of Legos they currently have can be identified, and a list of models that can be assembled is selected and displayed. It detects the required Lego during the assembly process and provides information on the Lego. Among YOLOs, YOLOv4 and Faster R-CNN, one of the detection algorithms of CNN, were used for the detection algorithm. By learning YOLOv4 under different conditions, and applying YOLOv4 under the condition tha.

t derives the best result by comparing the results, it was confirmed whether the condition showed excellent performance even under the condition with Faster R-CNN. YOLOv4 showed a better block detection rate than Faster-RCNN in the same environment, but it was difficult to detect when blocks overlapped during system progress.

KEYWORDS

Deep Learning, Object Detection, YOLOv4, Unity, Faster R-CNN

참고문헌

[1] Masi, I., Wu, Y., Hassner, T., & Natarajan, P. (2018, October). Deep face recognition: A survey. In 2018 31st SIBGRAPI conference on graphics, patterns and images (SIBGRAPI) (pp. 471-478). IEEE.

[2] Arnold, E., Al-Jarrah, O. Y., Dianati, M., Fallah, S., Oxtoby, D., & Mouzakitis, A. (2019). A survey on 3d object detection methods for autonomous driving applications. IEEE Transactions on Intelligent Transportation Systems, 20(10), 3782-3795.

[3] Wu, X., Sahoo, D., & Hoi, S. C. (2020). Recent advances in deep learning for object detection. Neurocomputing, 396, 39-64.

[4] Everingham, M., Van Gool, L., Williams, C. K., Winn, J., & Zisserman, A. (2010). The pascal visual object classes (voc) challenge. International journal of computer vision, 88(2), 303-338.

[5] Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... & Zitnick, C. L. (2014, September). Microsoft coco: Common objects in context. In European conference on computer vision (pp. 740-755). Springer, Cham.

[6] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 580-587).

[7] Girshick, R. (2015). Fast r-cnn. In Proceedings of the IEEE

international conference on computer vision (pp. 1440-1448).

[8] Ren, S., He, K., Girshick, R., & Sun, J. (2016). Faster R-CNN: towards real-time object detection with region proposal networks. IEEE transactions on pattern analysis and machine intelligence, 39(6), 1137-1149.

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

[10] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016, October). Ssd: Single shot multibox detector. In European conference on computer vision (pp. 21-37). Springer, Cham.

[11] Redmon, J., & Farhadi, A. (2017). YOLO9000: better, faster, stronger. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7263-7271).

[12] Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767.

[13] Laroca, R., Severo, E., Zanlorensi, L. A., Oliveira, L. S., Gonçalves, G. R., Schwartz, W. R., & Menotti, D. (2018, July). A robust real-time automatic license plate recognition based on the YOLO detector. In 2018 International Joint Conference on Neural Networks (IJCNN) (pp. 1-10). IEEE.

[14] 임헌국. (2021). 자율주행 차량 영상 기반 객체 인식 인공지능 기술 현황. 한국정보통신학회논문지, 25(8), 1117-1123.

[15] Wojke, N., Bewley, A., & Paulus, D. (2017, September). Simple online and realtime tracking with a deep association metric. In 2017 IEEE international conference on image processing (ICIP) (pp. 3645-3649). IEEE.

[16] 김경훈, 허준호, & 강석주. (2020). CPU 환경에서의 실시간 동작을 위한 딥러닝 기반 다중 객체 추적 시스템. 방송공학회논문지, 25(2), 192-199.

[17] light-tree.tistory, “딥러닝 객체 검출 용어 정리”, Sep 21. 2018, accessed Nov 15. 2021, https://light-tree.tistory.com/75.

[18] Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., & Pietikäinen, M. (2020). Deep learning for generic object detection: A survey. International journal of computer vision, 128(2), 261-318.

[19] Arthur Ouaknine Zyl Stroy, “Review of Deep Learning Algorithms for Object Detection”, Feb 18. 2018, accessed Nov 22. 2021, https://medium.com/zylapp/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852.

[20] Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... & Zitnick, C. L. (2014, September). Microsoft coco: Common objects in context. In European conference on computer vision (pp. 740-755). Springer, Cham.

[21] Adarsh, P., Rathi, P., & Kumar, M. (2020, March). YOLO v3-Tiny: Object Detection and Recognition using one stage improved model. In 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS) (pp. 687-694). IEEE.W

[22] Du, J. (2018, April). Understanding of object detection based on CNN family and YOLO. In Journal of Physics: Conference Series (Vol. 1004, No. 1, p. 012029). IOP Publishing.

[23] Chen, J., Liu, Z., Wang, H., Núñez, A., & Han, Z. (2017). Automatic defect detection of fasteners on the catenary support device using deep convolutional neural network. IEEE Transactions on Instrumentation and Measurement, 67(2), 257-269.

[24] Dong, E., Zhu, Y., Ji, Y., & Du, S. (2018, August). An improved convolution neural network for object detection using YOLOv2. In 2018 IEEE International Conference on Mechatronics and Automation (ICMA) (pp. 1184-1188). IEEE.

[25] Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767.

[26] “YOLO Is Back! Version 4 Boasts Improved Speed and Accuracy”, Synced, Mar 27. 2020, accessed Fab 3. 2022, https://syncedreview.com/2020/04/27/yolo-is-back-version-4-boasts-improved-speed-and-accuracy/

[27] Qian, R., Liu, Q., Yue, Y., Coenen, F., & Zhang, B. (2016, August). Road surface traffic sign detection with hybrid region proposal and fast R-CNN. In 2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD) (pp. 555-559). IEEE.

[28] Eggert, C., Brehm, S., Winschel, A., Zecha, D., & Lienhart, R. (2017, July). A closer look: Small object detection in faster R-CNN. In 2017 IEEE international conference on multimedia and

expo (ICME) (pp. 421-426). IEEE.

[29] Liu, B., Zhao, W., & Sun, Q. (2017, October). Study of object detection based on Faster R-CNN. In 2017 Chinese Automation Congress (CAC) (pp. 6233-6236). IEEE.

[30] Basri, H., Syarif, I., & Sukaridhoto, S. (2018, October). Faster R-CNN implementation method for multi-fruit detection using tensorflow platform. In 2018 international electronics symposium on knowledge creation and intelligent computing (IES-KCIC) (pp. 337-340). IEEE.

[31] Yanagisawa, H., Yamashita, T., & Watanabe, H. (2018, January). A study on object detection method from manga images using CNN. In 2018 International Workshop on Advanced Image Technology (IWAIT) (pp. 1-4). IEEE.

[32] Ren, S., He, K., Girshick, R., & Sun, J. (2016). Faster R-CNN: towards real-time object detection with region proposal networks. IEEE transactions on pattern analysis and machine intelligence, 39(6), 1137-1149.

[33] Lego sales and profits surge to record highs, Richard Milne, last modified Sep 28. 2021, accessed Act 10. 2021, https://www.ft.com/content/d2e845f1-1484-4745-8e3e-4995ee85f772

[34] 김경희, 코로나 확산에 레고 매출도 3년만에 증가세, Joongang, Sep 3. 2020, accessed Act 11. 2021, https://www.joongang.co.kr/article/23864119#home

[35] 이승민, 레고, 코로나19 반사이익, Yonhap, Nov 29. 2020, accessed Act 11. 2021, https://www.yna.co.kr/view/AKR20201129058400009

[36] Googke Colaboratory, “Colaboratory에 오신 것을 환영합니다.”, accessed Nov 20. 2021, https://colab.research.google.com/notebooks/intro.ipynb#scrollTo=OwuxHmxllTwN.

[37] Unity, , Unity User Manual 2020.3 (LTS), last modified May 28. 2021, accessed May 29,https://docs.unity3d.com/Manual/index.html

[38] An efficient use method for unity 3D engine

[39] Davis, J., & Goadrich, M. (2006, June). The relationship between 정밀도-재현율 and ROC curves. In Proceedings of the 23rd international conference on Machine learning (pp. 233-240).

감사의 글

이 논문을 작성할 때까지 많은 분들의 도움을 받았습니다. 이 도움들이 없었다면 논문이 나올 수 없었을 것으로 생각하여 이 글을 통해 감사를 전하고자 합니다.

첫 번째로 연구에 있어서 언제나 아낌없이 지원해주셨던 유선진 교수님께 깊은 감사의 뜻을 표하고자 합니다. 대학원에 입학하기 전부터 상담을 해주시며 학부 졸업을 앞두고 길을 방황하던 저에게 인생에 새로운 길을 열어 주셨습니다. 또한, 아무것도 모르던 저에게 하나부터 세세하게 연구 방법과 다양한 지혜들을 가르쳐 주셔서 차근차근 배워나갈 수 있었습니다. 혹시 실수하더라도 항상 부드럽게 타일러 주시고, 어떻게 고쳐나가야 하는지 알려주신 덕분에 즐겁게 연구를 할 수 있었습니다. 이를 통해 별 탈 없이 논문을 쓸 수 있었고, 석사 졸업까지 도달할 수 있었습니다. 이에 진심으로 감사드립니다.

연구실 사람들에게도 많은 도움을 받을 수 있었습니다. 랩장인 변공규 선생님과 박사 우동현 선생님께도 많은 것을 배울 수 있었습니다. 저보다 먼저 석사 과정을 거치며 쌓아온 삶의 경험을 통해, 연구나 평소 생활함에 있어서 막히는 부분이 있을 때는 조언을 해주시며 이를 통해 부족한 부분을 고쳐나갈 수 있었습니다. 또한, 논문을 피드백 받을 때도 날카로운 지적을 통해 먼저 확인을 해주셔서 논문 수정을 할 때도 세세한 부분까지 고칠 수 있었습니다. 그리고 같은 연구실 생활을 한 하은총, 이현주, 최현빈과 함께 하면서도 많은 부분을 배울 수 있었습니다. 각자 연구한 분야는 다르지만, 연구를 진행하면서 세밀하게 준비하는 모습이나, 다른 분야에서 새로운 영감을 얻을 때 많은 도움이 되어 연구 활동에 많은 자극을 받을 수 있었습니다. 그리고 비록 다른 연구실 소속이지만 함께 석사 과정을 보낸 최하람, 강동현과도 함께 지내며 즐거운 연구실 생활을 할 수 있었고, 평소에도 서로 도움을 주고받아, 어떤 때는 같은 연구실 같기도 한 친구들이었습니다. 모두 감사드립니다.

이에 더하여 논문작성에 있어서 많은 도움을 주신 이재호 박사님께 감사드립니다.

비록 늦게 회의를 시작하였지만, 논문 토픽을 정함에 있어서부터 논문을 출판할 때까지, 바쁘시더라도 먼 거리에서도 피드백을 주셨고, 제가 모르거나 막히는 부분이 있을 때면 어떻게든 제게 필요한 정보를 제공해주셔서 현재의 논문을 완성할 수 있었습니다. 정말 감사드립니다.

마지막으로 언제나 저를 믿고 지원해주신 부모님께 감사드립니다. 연구실 생활로 인해 자주 찾아뵙지 못하였지만, 항상 전화로 위로와 걱정을 해주셔서 늘 저에게 힘이 되어 주셨습니다. 이러한 부모님의 믿음과 격려로 인해 부족한 부분이 있을 때나 힘든 상황이 있을 때 기죽지 않고 이겨낼 수 있었습니다. 이에 정말 감사드립니다.

저의 연구실 생활은 위의 모든 분들이 있어 많은 도움을 받았고, 이로 인해 무사히 석사 졸업을 할 수 있었습니다. 앞으로도 이 도움을 잊지 않고 최선을 다하도록 하겠습니다. 감사합니다.

2022년 1월

김형진