Detectron2 를 활용한 Object detection 기술 연구

요 약

최근 AI 의 발전과 함께 머신러닝, 딥러닝 등의 학습 방식이 발전하면서 객체 인식 분야도 함께 진보하고 있다. 객체 인식 기술은 우리의 삶 모든 부분에 영향을 미치게 될 것이며 그 예로는 바이오이미징에서의 질병 식별, 산업 검사 및 로봇 비전과 같은 여러 분야가 있을 것이며 그 중에서도 자율 주행 무인 자동차 분야는 객체 인식 기술이 중요한 역할을 하게 될 것이다. 이번 논문에서는 FAIR 에서 개발한 Pytorch 기반의 Object Detection, Segmentation 라이브러리인 Detectron2 에 대하여 알아보고 활용하여 실생활에 바로 적용하여 삶을 유용하게 할 수 있는 것을 다뤄볼 것이다.

1. 서론

1.1. 연구배경

오늘날 정보통신기술의 발전으로 인해 하루에도 수많은 양의 영상들이 발생하고 이를 가공 처리하는 과정이 발생한다. 하지만 이 과정을 사람이 하기에는 시간과 비용이 많이 소모된다. 그러므로 사람이 아닌 인공지능이 하기 위하여 우리는 수많은 영상과 사진 등의 데이터셋(dataset)을 가지고 지도학습을 통하여 DNN(Deep Neural Network)이 객체 인식을 할 수 있도록 한다. 객체 인식 기술이 적극적으로 활용되는 분야에는 여러 가지가 있는데, 자율 주행 자동차 분야에 가장 적극적으로 활용되고 있고 그 외에도 사건, 사고 현장에서 사람을 인식하는 것, 사람이 쉽게 불이 난 것을 확인할 수 없는 산불 현장에서의 산불 검출 등에서도 쓰이고 있다. 수많은 종류의 대용량의 영상들이 있는 현재에 다양한 객체 인식 기술들이 개발되어 있고 앞으로 발전되고 새로운 기술들이 개발될 것이다. 따라서 이번 논문에서는 객체 인식 기술에 관하여 알아보고 그 중에서 FAIR 에서 개발한 Pytorch 기반의 Object Detection, Segmentation 라이브러리인 Detectron2 에 대하여 알아보고 활용하여 실생활에 바로 적용하여 삶을 유용하게 할 수 있는 것을 다뤄볼 것이다.

1.2. 연구목표

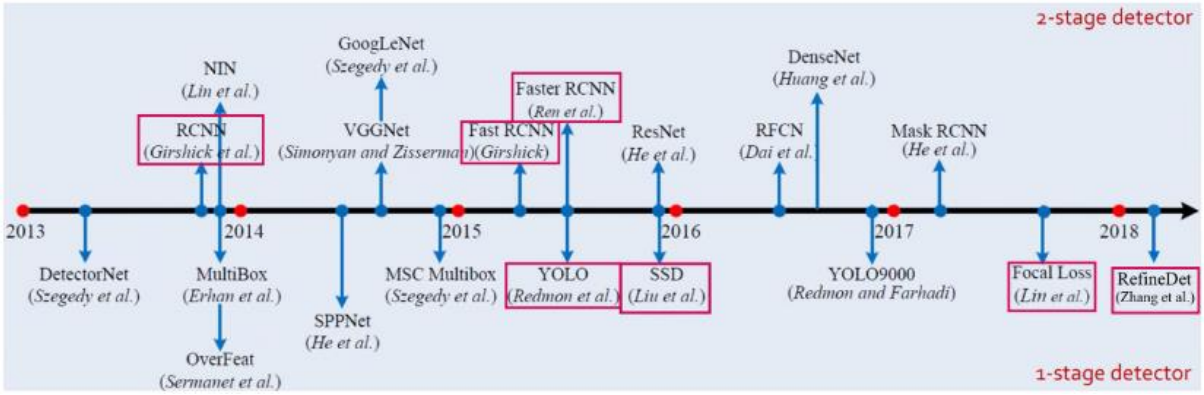
본 프로젝트의 연구 목표는 객체인식기술의 원리에 대하여 알아보고 detectron2 가 이미 인식할 수 있는 객체들에 대하여 알아본 후, 이를 활용하여 실생활에 접목하여 보는 것이다. 그 이후 이미 학습된 detectron2 에 대하여 Dataset(산불 발생 현장, 일상 현장, 문자 검출 등)을 가지고 학습시켜 detectron2 를 활용하여 이미지, 동영상에서 다양한 Object 를 검출할 수 있게 하는 것이 최종 목표이다.

2. 관련연구

2.1. OpenPose & DensePose

OpenPose 는 인간 자세 예측 (Human Pose Estimation)의 한 분야로 오로지 카메라 한대로만 가지고 사람의 몸, 얼굴, 손가락마디를 예측 하는 것이었다면 DensePose 는 2d 좌표만을 추정하였던 OpenPose 에 비해, RGB 이미지 상의 인간의 모든 픽셀을 3D 표면에 매핑하는 것이라고 할 수 있다. 각 인간 영역 내의 신체 부위 좌표를 여러 프레임 마다 회귀 시키며, 지역 기반 모델(Region-based model) 과 fully convolutional networks 로 이루어져있으며 Cascading 을 통해서 정확도를 향상한다. DensePose 는 자세 추정 (Pose Estimation), 객체 탐지 (Object Detection) 분야와 관련되어 있고, 여기서 Part 와 Instance 를 분할하는 문제를 해결하는 것이 목적이다.

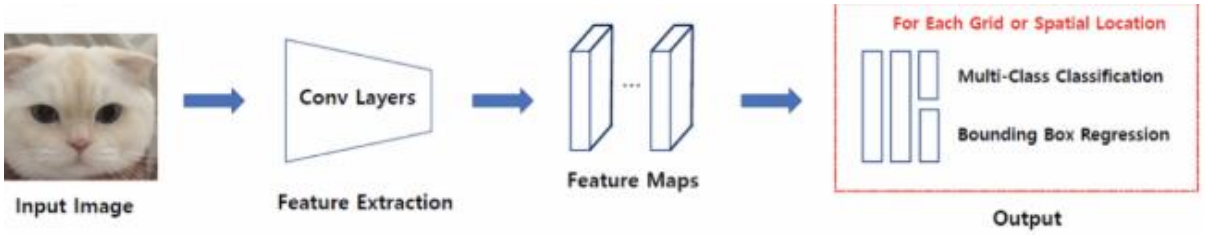
2.2. 1-Stage detector & 2-Stage dector



[그림 1] 1-Stage detector & 2-Stage detector

1- Stage detector

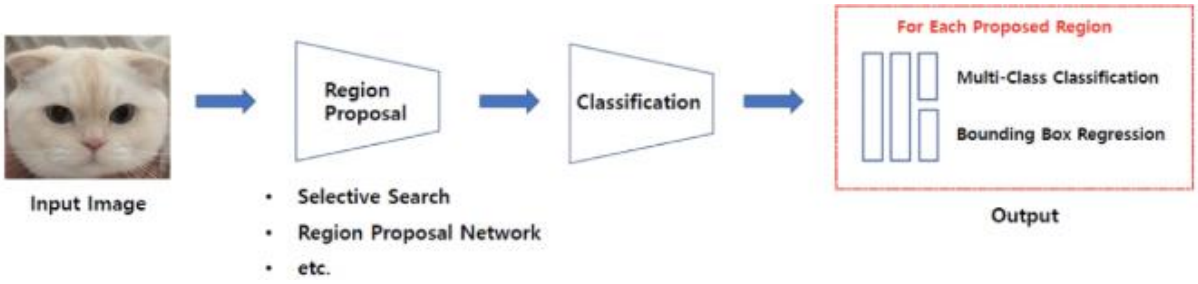
Regional proposal 과 classification 이 동시에 이루어진다.



[그림 2] 1-Stage detector process

2- Stage detector

Regional proposal 과 classification 이 순차적으로 이루어진다.



[그림 3] 2-Stage detector process

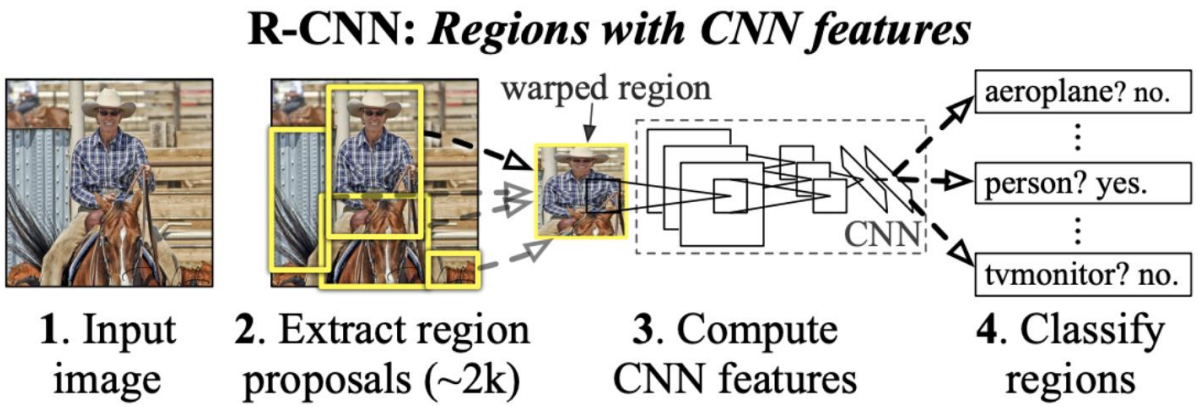
2.3. RCNN

R-CNN은 Image classification을 수행하는 CNN과 localization을 위한 regional proposal 알고리즘을 연결한 모델이다.

RCNN 의 과정은

1. Image를 입력 받는다.
2. Selective search 알고리즘에 의해 regional proposal output 약 2000개를 추출한다.
추출한 regional proposal output을 모두 동일 input size로 만들어주기 위해 warp해준다.
(이전의 Sliding window 방식은 너무 비효율적이어서 Selective search 알고리즘을 사용)
3. 2000개의 warped image를 각각 CNN 모델에 넣는다. 그리고 CNN모델에 들어가 feature vector를 뽑고 각각의 class마다 SVM로 classification을 수행한다.
4. 각각의 Convolution 결과에 대해 classification을 진행하여 결과를 얻는다.

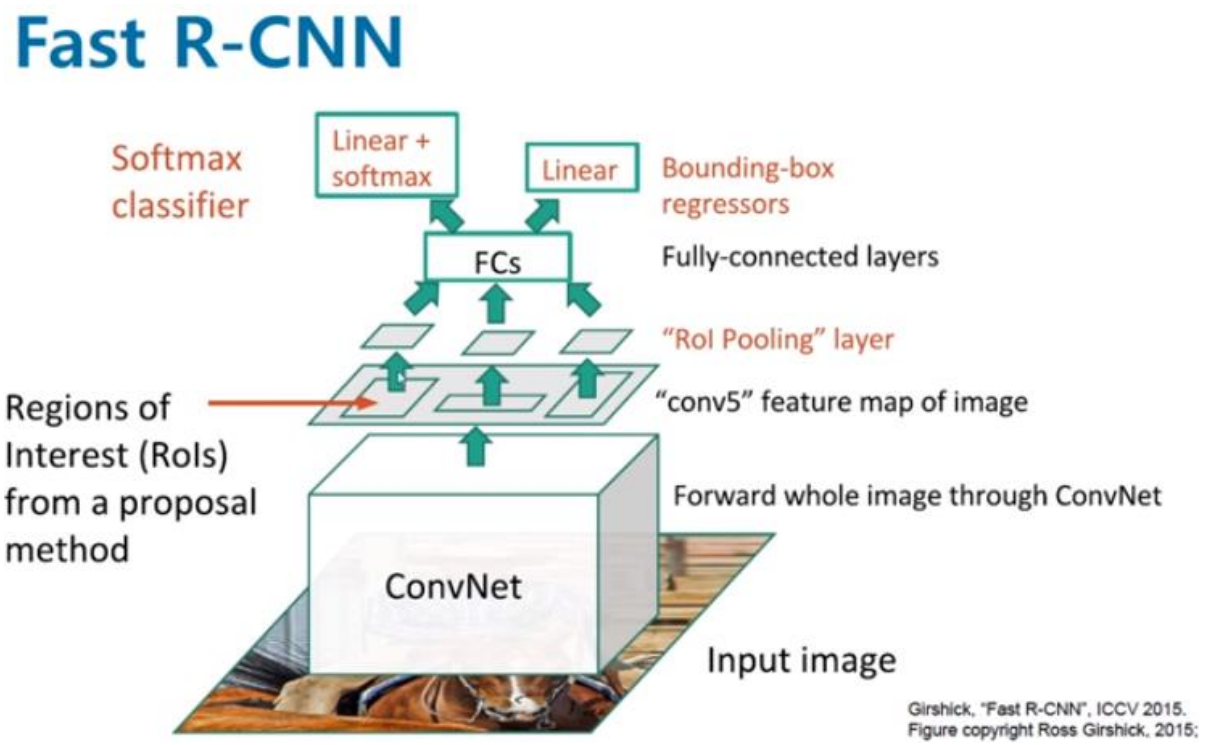
[1]



[그림 4] R-CNN Object detection system overview

RCNN은 selective search로 2000개의 region proposal을 뽑고 각 영역마다 CNN을 수행하기 때문에 CNN연산 * 2000 만큼의 시간이 걸려 수행시간이 매우 느리다.
이것을 Fast RCNN에서 RoI Pooling으로 보완하게 된다.

2.4. Fast RCNN

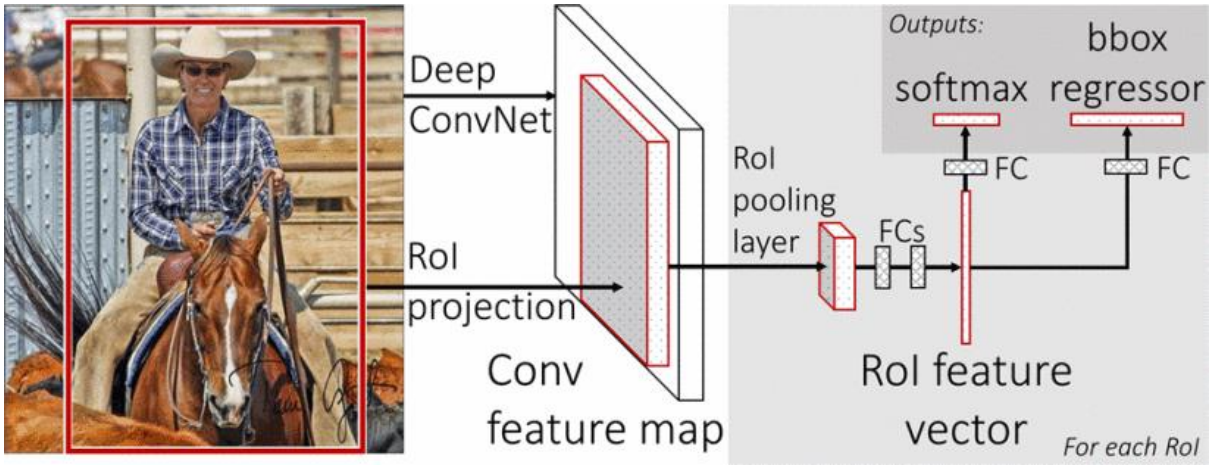


[그림 5] Fast R-CNN system overview

Fast RCNN 의 과정은

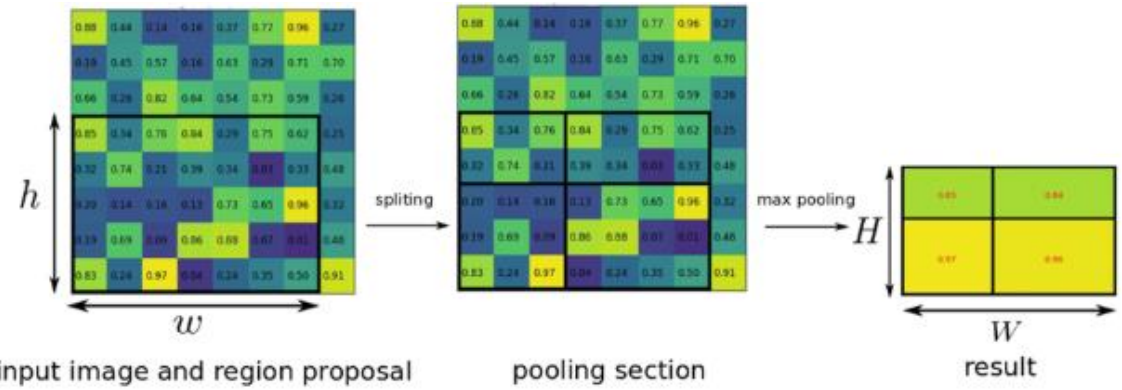
1. R-CNN에서와 마찬가지로 Selective Search를 통해 RoI를 찾는다. 그리고 전체 이미지를 CNN에 통과시켜 feature map을 추출한다.
2. Selective Search로 찾았던 RoI를 feature map크기에 맞춰서 projection시킨다.
3. projection시킨 RoI에 대해 RoI Pooling을 진행하여 고정된 크기의 feature vector를 얻는다.
4. feature vector는 FC layer를 통과한 뒤, 구 브랜치로 나뉘게 된다.
5. 하나는 softmax를 통과하여 RoI에 대해 object classification을 한다. 그리고 bounding box regression을 통해 selective search로 찾은 box의 위치를 조정한다.

[2]



[그림 6] Fast R-CNN architecture

RoI Pooling



[그림 7] RoI overview

Fast R-CNN에서 먼저 입력 이미지를 CNN에 통과시켜 feature map을 추출한다.
그 후 이전에 미리 Selective search로 만들어놨던 RoI(=region proposal)을 feature map에 projection시킨다.
위 그림의 가장 좌측 그림이 feature map이고 그 안에 검은색 box가 투영된 RoI이다.

미리 설정한 HxW크기로 만들어주기 위해서 (h/H) * (w/H) 크기만큼 grid를 RoI위에 만든다.
RoI를 grid크기로 split시킨 뒤 max pooling을 적용시켜 결국 각 grid 칸마다 하나의 값을 추출한다.
위 작업을 통해 feature map에 투영했던 검은색 박스크기의 RoI는 result 크기의 고정된 feature vector로 변환된다.
이렇게 RoI pooling을 이용함으로써
원래 이미지를 CNN에 통과시킨 후 나온 feature map에 이전에 생성한 RoI를 projection시키고
이 RoI를 FC layer input 크기에 맞게 고정된 크기로 변형할 수가 있다
따라서 더이상 2000번의 CNN연산이 필요하지 않고 1번의 CNN연산으로 속도를 대폭 높일 수 있다.

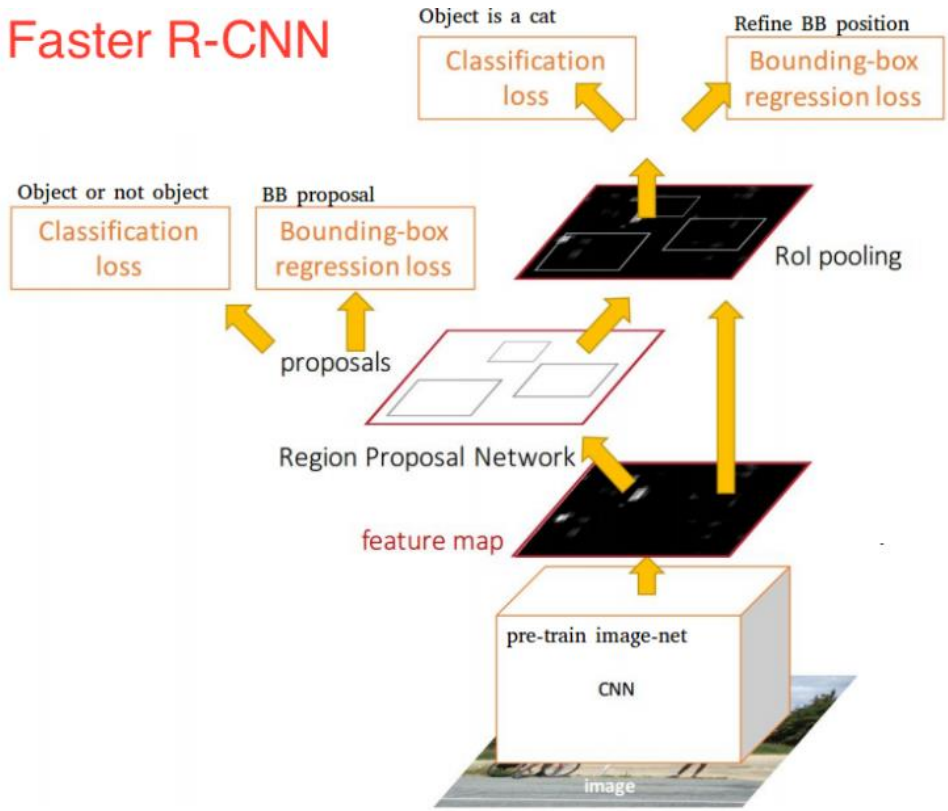
Fast RCNN의 경우도 RCNN처럼 RoI를 만드는 Selective search 알고리즘이 CNN외부에서 진행되는데 이것을 내부에서 하는 Faster RCNN이 나오게 된다.

2.5. Faster RCNN

Faster R-CNN은 Fast R-CNN에 RPN 이 결합되었다고 할 수 있다.
Faster R-CNN은 Fast R-CNN구조에서 conv feature map과 RoI Pooling사이에 RoI를 생성하는 Region Proposal Network가 추가된 구조이다.

그리고 Faster R-CNN에서는 RPN 네트워크에서 사용할 CNN과 Fast R-CNN에서 classification, bounding box regression을 위해 사용한 CNN 네트워크를 공유하자는 개념에서 나왔다.

[3]

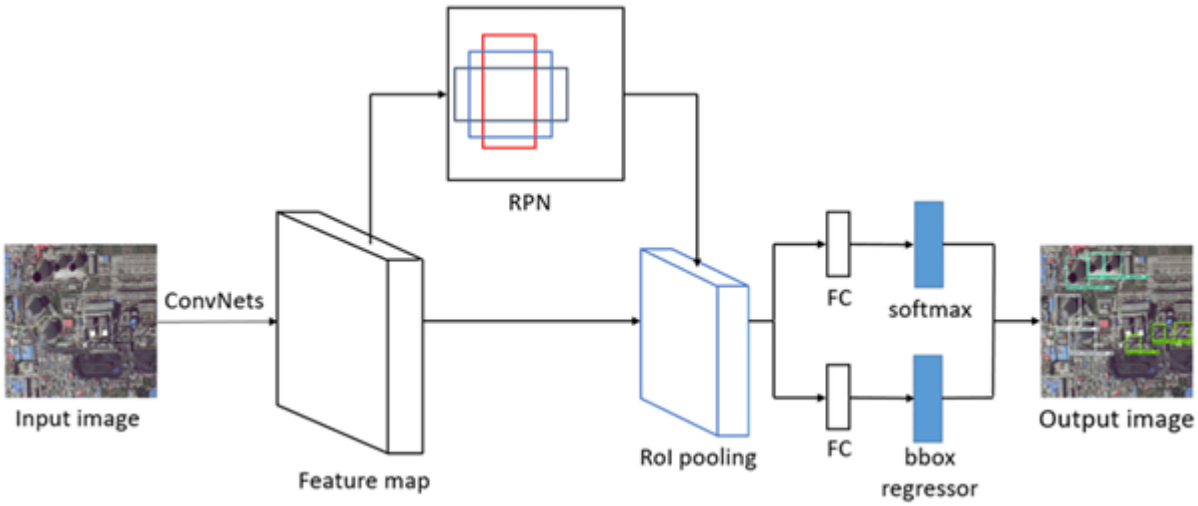


[그림 8] Faster R-CNN is a single, unified network for object detection

위 그림에서와 같이 CNN을 통과하여 생성된 conv feature map이 RPN에 의해 RoI를 생성한다. 주의해야할 것이 생성된 RoI는 feature map에서의 RoI가 아닌 original image에서의 RoI이다.

따라서 original image위에서 생성된 RoI는 conv feature map의 크기에 맞게 rescaling된다.

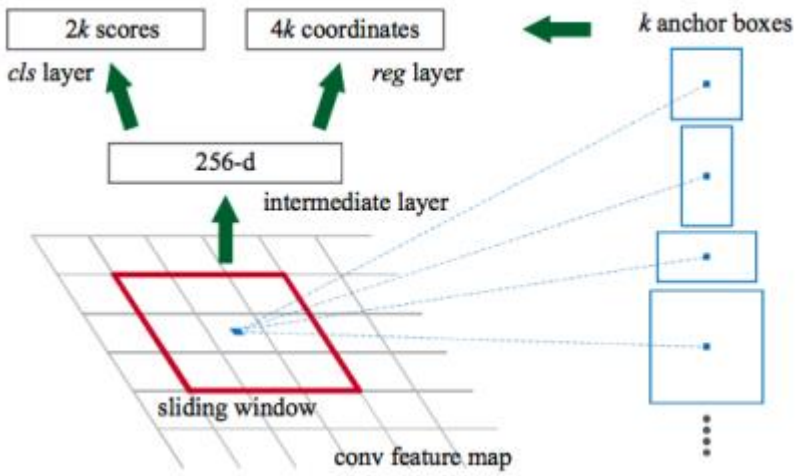
feature map 에 RoI 가 투영되고 나면 FC layer 에 의해 classification 과 bounding box regression 이 수행된다.



[그림 9] RoI & Fc layer

위 그림에서 보다시피 마지막에 FC layer를 사용하기에 input size를 맞춰주기 위해 RoI pooling을 사용한다. RoI pooling을 사용하니까 RoI들의 size가 달라도 되는 것처럼 original image의 input size도 달라도 된다.

RPN



[그림 10] RPN

RPN의 input 값은 이전 CNN 모델에서 뽑아낸 feature map이다.

Region proposal을 생성하기 위해 feature map위에 nxn window를 sliding window시킨다.

이때, object의 크기와 비율이 어떻게 될지모르므로 k개의 anchor box를 미리 정의해놓는다.

이 anchor box가 bounding box가 될 수 있는 것이고 미리 가능할만한 box모양 k개를 정의해놓는

것이다.
여기서는 가로세로길이 3종류 x 비율 3종류 = 9개의 anchor box를 이용한다.

이 단계에서 9개의 anchor box를 이용하여 classification과 bbox regression을 먼저 구한다.
CNN에서 뽑아낸 feature map에 대해 3x3 conv filter 256개를 연산하여 depth를 256으로 만든다.
그 후 1x1 conv 두개를 이용하여 각각 classification과 bbox regression을 계산한다.
RPN에서 이렇게 1x1 convolution을 이용하여 classification과 bbox regression을 계산하는데
이때 네트워크를 가볍게 만들기 위해 binary classification으로 bbox에 물체가 있나 없나만 판단한다. 무슨 물체인지 classification하는 것은 마지막 classification 단계에서 한다.
RPN단계에서 classification과 bbox regression을 하는 이유는 결국 학습을 위함이다.
위 단계로부터 positive / negative examples들을 뽑아내는데 다음 기준에 따른다.

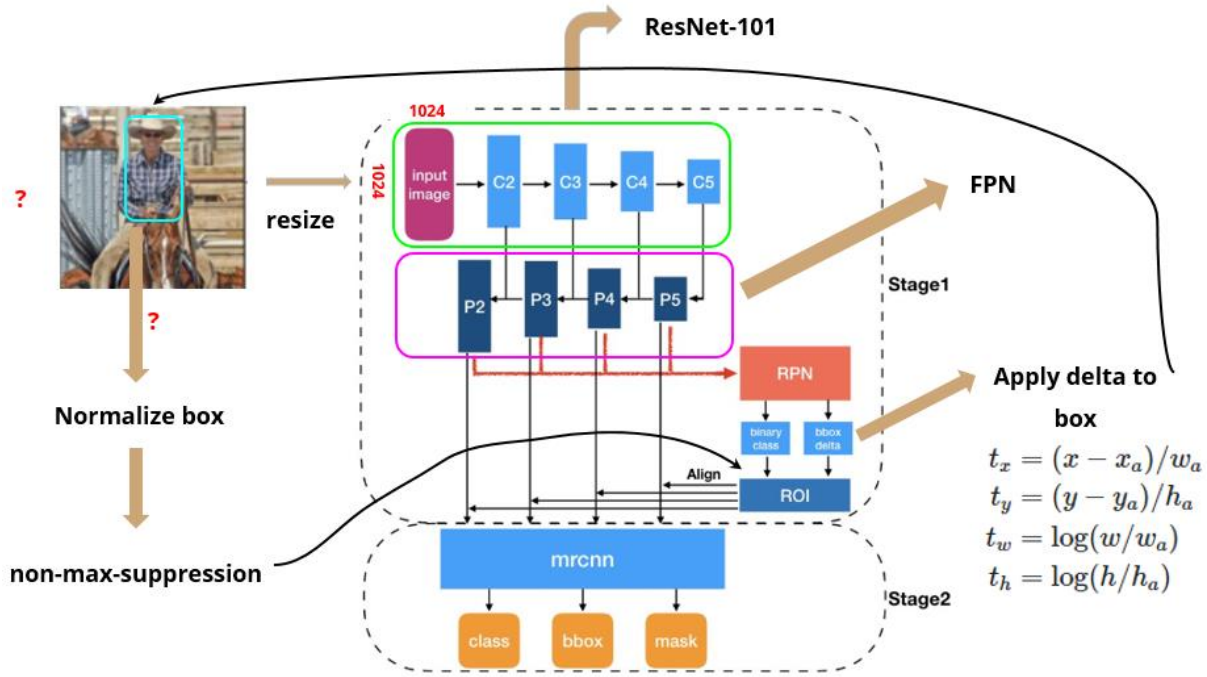
$$p^* = \begin{cases} 1 & \text{if } IoU > 0.7 \\ -1 & \text{if } IoU < 0.3 \\ 0 & \text{if otherwise} \end{cases}$$

IoU가 0.7보다 크거나, 한 지점에서 모든 anchor box중 가장 IoU가 큰 anchor box는 positive example로 만든다.
IoU가 0.3보다 작으면 object가 아닌 background를 뜻하므로 negative example로 만들고
이 사이에 있는 IoU에 대해서는 애매한 값이므로 학습 데이터로 이용하지 않는다.

Faster R-CNN에 대한 학습이 완료된 후 RPN모델을 예측시키며 하마 한 객체당 여러 proposal값이 나올 것이다.

이 문제를 해결하기 위해 NMS알고리즘을 사용하여 proposal의 개수를 줄인다.
NMS알고리즘은 다음과 같다.
1. box들의 score(confidence)를 기준으로 정렬한다.
2. score가 가장 높은 box부터 시작해서 다른 모든 box들과 IoU를 계산해서 0.7이상이면 같은 객체를 detect한 box라고 생각할 수 있기 때문에 해당 box는 지운다.
3. 최종적으로 각 object별로 score가 가장 높은 box 하나씩만 남게 된다.

2.6. Mask RCNN



[그림 11] Mask RCNN architecture

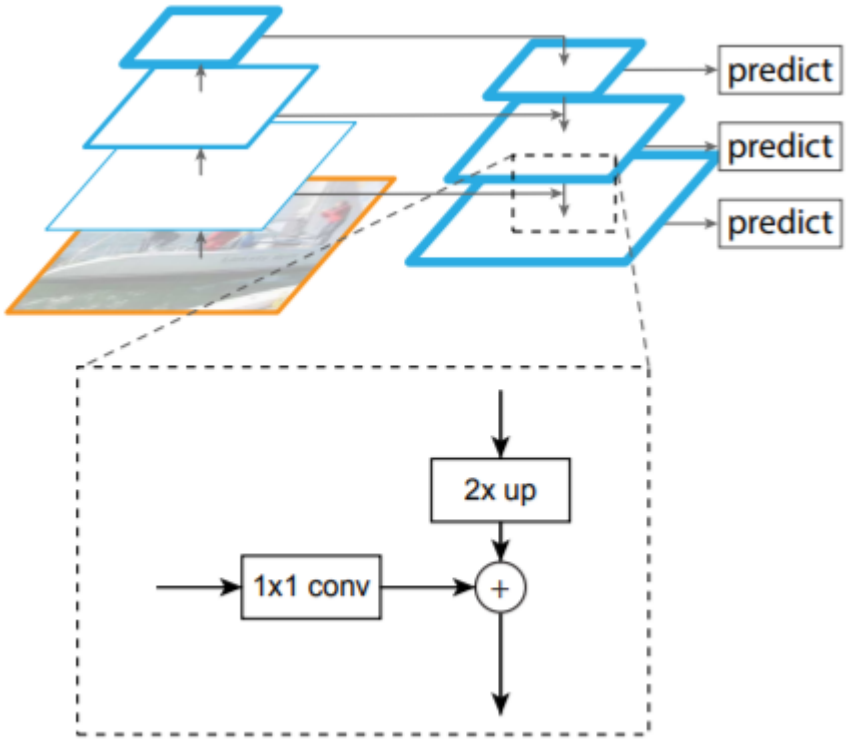
Input image resize

Mask R-CNN에서는 backbone으로 ResNet-101을 사용하는데 ResNet 네트워크에서는 이미지 input size가 800~1024일때 성능이 좋으므로 input image를 이 사이즈로 맞춰준다.
이렇게 resize를 한 후 네트워크의 input size인 1024 x 1024로 맞춰주기 위해
나머지 값들은 zero padding으로 값을 채워준다.
그리고 [4]Mask R-CNN에서는 Backbone으로 ResNet모델을 사용한다.

FPN

FPN에서는 위 그림과 같이 마지막 layer의 feature map에서 점점 이전의 중간 feature map들을 더하면서 이전 정보까지 유지할 수 있도록 한다. 이렇게 함으로써 더이상 여러 scale값으로 anchor를 생성할 필요가 없게 되고 모두 동일한 scale의 anchor를 생성한다. 따라서 작은 feature map에서는 큰 anchor를 생성하여 큰 object를, 큰 feature map에서는 다소 작은 anchor를 생성하여 작은 object를 detect할 수 있도록 설계되었다.

마지막 layer에서의 feature map에서 이전 feature map을 더하는 것은 Upsampling을 통해 이루어진다.



[그림 12] Feature map 관련

먼저 2배로 upsampling을 한 후 이전 layer의 feature map을 1x1 Fully convolution 연산을 통해 filter개수를 똑같이 맞춰준후 더함으로써 새로운 feature map을 생성한다.

RPN

위의 과정을 통해 생성된 내용들을 각각 RPN 모델에 전달하는데 Faster R-CNN 와 달리 이제 각 feature map 에서 1 개 scale 의 anchor 를 생성하므로 결국 각 pyramid feature map 마다 scale 1 개 x ratio 3 개 = 3 개의 anchor 를 생성한다.

RPN 을 통해 output 으로 classification 값, bbox regression 값이 나오는데 이때 bbox regression 값은 delta 값이다.

$$
\begin{aligned}
t_x &= (x - x_a)/w_a \\
t_y &= (y - y_a)/h_a \\
t_w &= \log(w/w_a) \\
t_h &= \log(h/h_a)
\end{aligned}
$$

.

$$
\begin{aligned}
t_x^* &= (x^* - x_a)/w_a \\
t_y^* &= (y^* - y_a)/h_a \\
t_w^* &= \log(w^*/w_a) \\
t_h^* &= \log(h^*/h_a)
\end{aligned}
$$

t_x, t_y : 박스의 center coordinates
t_w, t_h : 박스의 width, height
x, y, w, h : predicted box
x_a, y_a, w_a, h_a : anchor box
x^*, y^*, w^*, h^* : ground-truth box

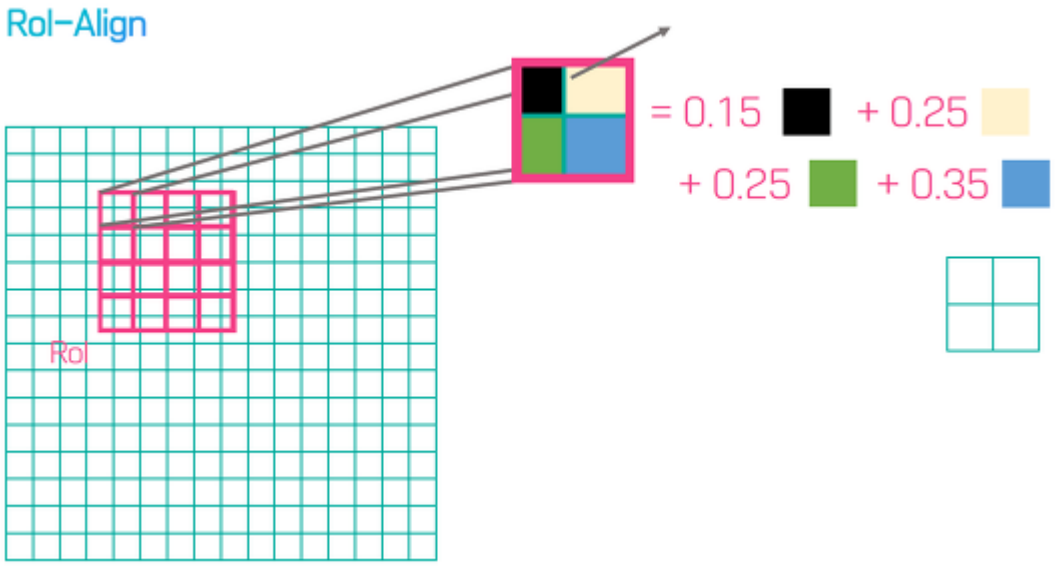
t값들, 즉 delta값을 output 으로 받게 된다. 따라서 이 delta값에 anchor정보를 연산해서 원래 이미지에 대응되는 anchor bounding box 좌표값으로 바꿔주게 된다.

NMS

원래 이미지에 anchor 좌표를 대응시킨 후에는 각각 normalized coordinate로 대응시킨다.
이는 FPN에서 이미 각기 다른 feature map크기를 갖고있기에 모두 통일되게 정규좌표계로 이동시키는 것이다. 이렇게 수천 개의 anchor box가 생성되면 NMS알고리즘을 통해 anchor의 개수를 줄인다. 각 object마다 대응되는 anchor가 수십개 존재하는데 이때 가장 classification score가 높은 anchor를 제외하고 주위에 다른 anchor들은 모두 지우는 것이다.
NMS알고리즘은 anchor bbox들을 score순으로 정렬시킨 후 score가 높은 bbox부터 다른 bbox와 IoU를 계산한다. 이때 **IoU가 해당 bbox와 0.7이 넘어가면 두 bbox는 동일 object를 detect한 것이라 간주**하여 score가 더 낮은 bbox는 지우는 식으로 동작한다. 최종적으로 각 객체마다 score가 가장 큰 box만 남게되고 나머지 box는 제거한다.

RoI align

bilinear interpolation 을 이용해서 위치정보를 담는 RoI align 을 이용한다.



[그림 13] RoI Align

3. 프로젝트 내용

3.1. 접근방식

Detectron2 가 인식할 수 있는 객체들이 어떤 것이 있는지를 알아보고 이를 바탕으로 실생활에 적용하여 할 수 있는 것이 어떤 것이 있는지 아이디어를 구상하여 프로젝트를 진행한다.
그 이후에는 이미 학습된 detectron2 를 문자 검출 관련 Dataset 를 가지고 지도학습을 통해 detectron2 로 문자에 대한 이미지를 검출할 수 있게 한다.

3.2. 요구사항

기존에 학습된 모델인 Detectron2 를 이용하여 프로젝트를 진행하므로 아이디어와 연관된 Dataset 을 찾아본다. 그리고 문자를 임의적으로 넣은 이미지 관련 Dataset 을 찾아봐야 하고 detectron2 를 어떻게 학습시킬지에 대한 내용을 공부한다.

4. 향후 일정 및 역할 분담

앞으로의 일정으로는 detectron2 가 인식할 수 있는 객체의 종류와 범위에 대해 이해하고 이를 활용하여 할 수 있는 아이디어를 구상해보고 이를 결과물로 작성 후, detectron2 가 문자에 대한 이미지를 검출할 수 있도록 학습을 시킬 예정이다.

5. 결론 및 기대효과

이번 프로젝트의 기대효과는 detectron2 를 활용해봄으로써 Object detection 기술에 대해 알아봄과 동시에 detectron2 의 학습을 통해 문자검출이 가능하게 하는 것이다.

6. 참고문헌

[1] "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation", Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik; Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014, pp. 580-587

[2] "Fast R-CNN", Ross Girshick; 2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp.1440-1448

[3] "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun; in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137-1149; Part of Advances in Neural Information Processing Systems 28 (NIPS 2015)

[4] "Mask R-CNN", Kaiming He, Georgia Gkioxari, Piotr Dollar, Ross Girshick; Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2961-2969