**Faster R-CNN Real-Time Object Detection with Region Proposal Network**

**Script**

**Page [1]**

**Page [2]**

**Page [3]**

논문의 내용을 소개하기에 앞서, 우선 이 논문이 쓰여지게 된 배경부터 살펴보면 좋겠다 생각이 들어서 이 페이지를 추가하였습니다.

저 같은 경우에는, 이 논문을 선정하고 제일 먼저 들었던 생각이,

Faster R-CNN은 어떤 필요에 의해 고안이 되었고, 그렇다면, Faster R-CNN의 역사는 어떨지에 대해, 그리고 왜 하필 이름이 R-CNN인지 이런게 궁금했었습니다.

그리고 이 내용을 공부하는 과정에서 faster R-CNN에 대해 하나씩 이해할 수 있었기에 이와 같이 Introduction Part를 추가하였습니다.

그래서, 이 page의 내용에 대해 소개를 해보자면,

Image Classification을 연구하는 과정에서 딥러닝 개념이 도입된 Convolutional Neural Network, CNN이 고안되었습니다.

이후에는, 단순히 해당 이미지를 Classification을 하는 것을 넘어서, 그렇다면 그 물체가 이미지에서 어디에 있는지 파악을 하고 싶었고,

그래서 고안된 것이 여러 물체에 대한 Classification과 Localization을 모두 할 수 있는 Object Detecting입니다.

화면에서 보실 수 있듯이 Object Detecting은 크게 두 가지로 나뉠 수 있는데,

저는 여기서 two-stage detector에 초점을 맞추었고, two-stage detector의 발달 과정을 보면 RCNN에서 Fast RCNN, Faster RCNN으로 발달한 것을 볼 수 있습니다.

**Page [4]**

따라서, Faster R-CNN에 대해 설명하기에 앞서, 우선은 R-CNN에 대해 간단하게 살펴보자면,

R-CNN의 R은 Region으로,

Classification 및 Localization을 모두 진행하는 Algorithm임을 이름을 통해 엿볼 수 있습니다.

또, R-CNN의 경우에는

* 1. CPU상에서 **Selective Search** Algorithm으로 약 2000개의 Region Proposal, 즉 객체가 있을 법한 예상 범위에 대한 제안이 이루어 집니다.
  2. 이후에는 제안된 영역을 모두 CNN에 넣어서 Classification을 진행하는데,

이는 2,000개에 달하는 후보 이미지 각각에 대해서 convolution 연산을 수행하게 되어 비효율적이고 느린 문제가 있습니다.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

따라서, 보다 효율적인 Object Detecting연구가 진행되었고 그 결과로 Convolution연산을 한 번만 진행해도 되면서,

Classification Layer에 입력으로, 전체 이미지를 CNN하여 얻은 Feature Map과 Region Proposal을 한 번에 주는

새로운 Architecture를 고안하게 되었는데, 그것이 바로 Fast RNN입니다.

**이를 통해** 연산량을 크게 줄일 수 있었다고 합니다.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

이와 같이, Fast R-CNN에서는 연산량 단축을 통해 running time 단축엔 성공하였습니다.

하지만, Region proposal을 위해 사용한 Selective Search(SS)와 같은 알고리즘은 CPU기반으로 수행되기에 Region Proposal이 굉장히 느린 문제가 있습니다.

하나의 이미지당 기본 2초가 소요된다고 하니, Real-Time에 적용하기에는 턱없이 부족한 수치라고 생각할 수 있습니다.

따라서, Region Proposal을 위한 모든 연산을 GPU상에서 수행할 수 있도록 **Region Proposal Network, RPN이라는 개념을 도입**해서  
이를 통해 Region Proposal에 걸리는 시간을 획기적으로 단축하였는데, 이것이 **Faster R-CNN** 입니다.

**Page [5]**

서론이 다소 길었지만, 지금부터 본격적으로 Faster R-CNN에 대해 살펴보겠습니다.

쉽게 이야기하면, 화면에서 보실 수 있듯이,

**Faster R-CNN =** RPN + Fast R-CNN 이라고 볼 수 있겠습니다.

그만큼, Faster R-CNN에서는 RPN이 핵심 Idea라고 할 수 있는데,

RPN을 통해 앞서 말씀드렸듯이, 전체 Framework가 GPU상에서 한 번에 수행될 수 있게 되었고,

이를 통해 한 이미지에 대해 Detecting을 하는 데에 걸리는 시간이 0.2초, 즉 5fps가 될 정도로 크게 단축되었다고 합니다.

그럼, RPN이 어떤 것인지 좀 더 살펴보도록 하겠습니다.

이미지에 CNN을 적용하여 Feature Map을 뽑아내면, Feature Map만을 보고 어떤 위치에 Object가 있는지 알아낼 수 있기에

RPN에서는 전체 이미지의 Feature Map을 받고는,

‘어떠 어떠한 위치에 Object가 있을 것 같은데, 이 위치에 대해 Classification 진행해 볼래?’

이런 식으로 영역을 제안해주는 역할을 RPN이 해줍니다.

즉, RPN을 통해서 사물의 위치가 감지가 되면, 그 위치에 있는 사물이 어떤 사물인지 맞추는 방식으로 진행이 되기 때문에

RPN이 Attention Mechanism과 유사하다고 이해하실 수 있으며,

이 단계에서는 그 물체가 어떤 물체인지 Classification하는 것은 나중의 일이고,

그저 일단 Object가 위치하고 있을 법한 후보 영역들을 Feature Map을 통해 얻어내는 것 까지가 RPN의 역할입니다.

그리고, 그 후보 영역에 대한 Classification은 화면에서 볼 수 있듯이 Fast-RCNN구조를 그대로 가져와서 사용하였습니다.

**Page [6]**

그렇다면, 이러한 RPN은 어떠한 방식으로 동작할까요?

다양한 형태의 사물을 Detecting할 수 있도록 크기와 비율이 서로 다른 k개의 Anchor Box를 정의하여,

각각의 Anchor Box를 Sliding Window 방식으로 이동시키며 Feature Map으로부터 intermediate layer를 뽑고

각각의 intermediate layer에 대해 regression과 Classification을 진행합니다.

여기서의 Classification은,

물체의 종류를 맞추지는 않고, 그저 물체가 있는지 혹은 없는 지의 정보만을 파악한다는 것이 특징입니다.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

또, 본 논문에서는 정확한 Detecting을 위해 3가지의 Scale과 3가지의 Ratio로 총 9개의 **Anchor Box**를 정의하였습니다.

한편, 논문에서는 anchor box의 비율과 크기를 이렇게 한정 지었지만,

이 크기들을 바탕으로 하여 regression을 진행하기에 가지각색의 bounding box를 그리는 것이 가능해졌고, 그 결과 우측 그림에서 처럼 다양한 사이즈로 Bounding Box를 그리는 것이 가능해졌습니다.

**Page [7]**

참고로, Bounding Box Regression은

실제 4개의 Ground Truth 좌표 값을 학습해서

Ground Truth값과 예측된 4개의 좌표 값을 유사하게 맞추는 과정을 말합니다.

**Page [8]**

RPN을 통해 Region을 제안 받았다면, 그 Region과 Ground Truth Region을 비교하면서 성능을 평가할 수 있습니다.

이 과정은 IoU 계산을 통해 진행되며

말 그대로, Intersection over Union. 합집합 분에 교집합을 계산하는 것으로, 두 바운딩 박스가 겹치는 비율을 파악하는 것인데,

본 논문에서는

IoU가 70% 이상이면 모두 1로 Labeling, 혹은 70%가 넘지 않더라도 어떠한 값이 Ground Truth Box와 최대 IoU를 가진다면 그 값을 1로 Labeling

IoU가 30% 보다 작으면 모두 0으로 Labeling

IoU가 30%와 70% 사이, 또 그 값이 최대 IoU값이 아니라면 모두 -1로 Labeling (Default Label)

#       1 = object(Positive)

#       0 = background(Ignore)

#      -1 = Negative

**Page [9] [Loss Function Explanation]**

한편, 본 논문에서의 Loss Function은 IoU에 기반하고 있습니다.

*각각의 Loss는 Crossentropy를 이용하였다.*

*여기서,*

*은 정답 레이블이고,*

은 tx ty tw th 로 Ground Truth Bounding Box의 정보를 가지는 tuple data 입니다.

앞서 말씀드렸듯이 해당 Anchor가 positive면 1을 부여하는 식인데, 이렇게 함으로써

IoU가 negative도 positive도 아닌 의미가 없는 경우라면, 0을 부여해서, 식의 하단부가 아예 식에 반영되지 않도록 하였습니다.

이렇게 함으로써, 해당 Anchor에 물체가 존재하거나 존재하지 않을 가능성이 큰 경우에만 그 물체의 Bounding Box 좌표를 찾도록 하여 더욱 효율적인 학습이 되도록 하였다고 주장하였습니다.

한 가지 특이했던 점은,

RPN에서는 모든 앵커에 대해 전부 Loss를 계산하지 않았다는 점입니다.

Positive Anchor와 Negative Anchor의 비율이 1 : 1 이 되도록 랜덤하게 256개의 앵커만 샘플링하여 Loss를 계산하고 학습을 진행시켰다고 합니다.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**[Bounding Box Regression Equation – Explanation]**

한편,

Bounding Box Regression을 위해서 R-CNN논문에서 제안된 Parameterization을 그대로 이용했다고 합니다.

따라서, 이와 같은 방법으로 예측된 Bounding Box의 x, y, w, h의 좌표를 ground truth coordinate에 가깝게 맞추었다고 주장하고 있었습니다.

**Page [10]**

또한, IoU 계산을 통해서, 중복되는 박스를 제거함으로써 정답과 가장 근접한 Box만을 남기는 작업을 진행할 수 있습니다.

**즉, Class가 동일한 Bounding Box는 가지치기하는 것이며 이를 NMS(Non-Maximum Suppression) 기법이라 합니다.**

**본 논문에서는**

NMS를 위한 IoU의 Threshold값으로는 0.7을 선정하여

여러 바운딩 박스가 같은 클래스로 분류되면서 70%이상이 겹쳐 있다면 confidence 값이 낮은 박스를 제거하였고, 그 결과 최종적으로 한 **이미지당 약 2000개 정도의 Region Proposal만 나오도록 했습니다.**

**Page [11]**

지금까지 말한 것들이 모두 Region Proposal Network에서 이루어지며, 그렇다면 지금부터는

RPN과 Fast R-CNN이 어떻게 Feature Map을 공유하는가? 에 대해 말해보겠습니다.

논문에서는 Alternating Training 방식을 사용하여, 전체 학습 과정을 4단계로 세분화하여 설명하고 있었습니다.

[1] 우선, RPN Network를 먼저 Training 시키고

[2] 그 다음에 RPN Network를 통한 Propose된 Region으로 Fast R-CNN을 학습시킵니다.

즉, 두 번째 단계까지는, 앞쪽의 convolution Layer가 온전히 공유되지 못하는 한계가 있습니다.

[3] 따라서, 세번째 단계에서는 PRN Network에서 포함되지 않았던 앞쪽의 Convolution Layer까지 완전히 고정시킨 채로,

RPN Network안에 있는 추가적인 Convolution Layer에 대해서만 Fine Tuning을 진행합니다.

[4] 마찬가지로, 마지막 단계에서는, 앞쪽에 있는 Convolution Layer는 완전히 고정한 채로,

Fast R-CNN 에서만 포함되어 있는 Layer에 대해서만 Fine Tuning을 진행합니다.

이렇게 함으로써, 앞쪽의 Convolution Layer가 공유되면서 학습을 진행하였다고 합니다.

**Page [12]**

이러한 Faster R-CNN의 성능 평가를 위해 재미있는 실험을 많이 하고 있었습니다.

PASCAL VOC 2007 DATASET으로 테스트한 결과,

Selective Search Method를 사용했을 때 보다,

본 논문에서 채택하고 있는, RPN+ZF net을 사용하고, Feature Map을 공유할 때 성능이 가장 좋음을 확인할 수 있다고 합니다.

또, Proposal Region을 최종적으로는 2000개를 선정하지만, Test 할 때에서는 300개의 Region만 추출하더라도 충분히 좋은 성능이 나왔다고 밝히고 있었습니다.

**Page [13]**

또, VGGnet을 이용하고, 학습 데이터를 다양하게 가져갈수록 정확도가 많이 좋아짐을 확인할 수 있습니다.

한편, 아래쪽을 보시면 보실 수 있듯이,

VGGnet이 앞서 말씀드렸던 것 처럼 정확도는 좋지만, 기본적으로 ZF net 보다 더 깊은 Network이기 때문에, 속도 측면에서는 ZF보다 느린것을 확인할 수 있었고,

**Page [14]**

이거는, 본 논문에서 제시하고 있는 RPN을 사용할 때, 정확도가 전반적으로 상승하는 것을 확인할 수 있는 실험 결과이고,

**Page [15]**

이 외에도, Region Proposal을 몇 개로 하는게 가장 좋은지,

Normalization Parameter인 Lamda의 값을 얼마로 하는 것이 가장 좋은지,

심지어, Anchor Box의 사이즈 및 크기를 어떻게 구성할 때 정확도가 가장 좋은지 등등

다양하게 실험을 진행하였고, 본 논문에서 제시하고 있는 바로 진행할 때 모두 정확도가 가장 좋았다는 것을 수치적으로 확인할 수 있었습니다.

**Page [16]**

**Page [17]**

**Page [18]**

**Page [19]**