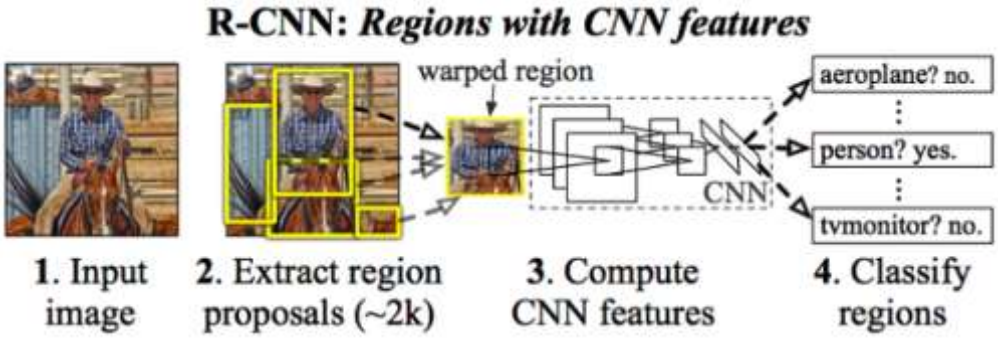
R-CNN (Region Proposal Convolutional Neural Network)

Deep-Learning Seminar

Posted by KBJ (/about) 2018-04-14 08:39:59

Faster R-CNN 논문을 읽고 그것에 대한 논문 리뷰를 포스팅 할 계획이었지만 R-CNN과 Fast R-CNN 내용을 필수적으로 알아야하기 때문에 R-CNN 부터 Fast R-CNN을 간략하게나마 우선 설명을 드리겠습니다..

R-CNN (Region Proposal Convolutional Neural Network)



이미지를 분류하는 CNN과 이미지가 있는 영역을 추천해주는 Region Proposal 알고리즘을 연결하는 Object Detection 기법 입니다. 다시 말하자면 말을 타는 카우보이의 이미지가 있을 때 Region Proposal 알고리즘으로 영역을 추천해주고 그 영역을 CNN 모델을 통해 "이것은 사람이다"를 알 수 있습니다.

영역을 추천해주지 않고 기존 Naive한 방식으로는 다양한 사이즈의 window를 좌측 상단부터 우측 하단까지 Sliding Window 방식으로 모든 영역을 탐색하는 방식이 있었습니다. 하지만 당연히 매우 느리고 비효율 적 입니다. 따라서 Region Proposal으로 물체가 있을 법 한 영역을 빠르게 찾아내는 방법을 고안하게 됩니다. R-CNN은 Region Proposal 기법이 성능을 좌지우지 합니다. 이때 Selective Search 알고리즘을 주로 사용합니다.

Selective Search(SS)

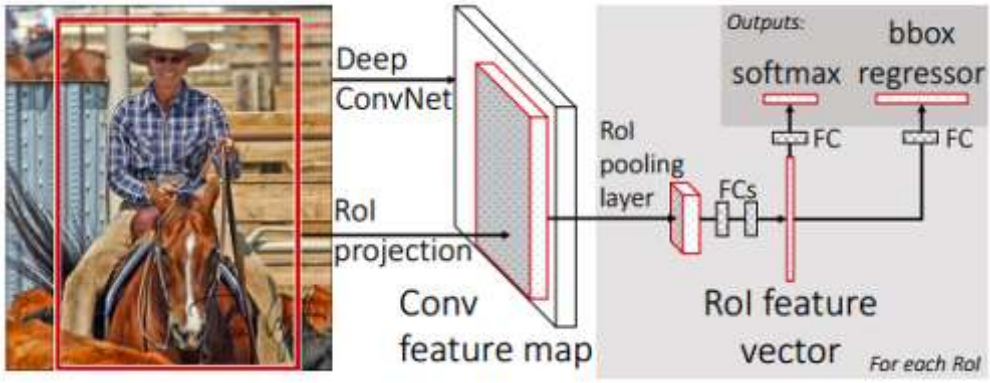
2013년 detection 분야에서 1위를 차지한 알고리즘 입니다. SS는 Exhaustive Search와 Segmentation을 결합한 방식 입니다. 우선 Segmentation 방식이란 이미지, 영상의 데이터 특성인 색(rgb, hsv..), 모양 등에 따라 영역을 나누어 주는 것 입니다. 하지만 모든 경우를 찾기는 힘듭니다. 이를 Exhaustive 하게 Search (후보가 될만한 모든 영역을 샅샅이 조사)하는 것이 Selective Search(SS) 알고리즘 입니다. SS는 작은 객체 부터 큰 객체 까지 모두 포함을 할 수 있도록 Bottom-Up 방법을 사용합니다.

R-CNN은 SS 알고리즘을 사용하여 이미지로부터 2000개 가량의 Region Proposal을 추출 합니다. 이 추출된 영역들을 동일한 크기로 만든 후, CNN을 통해 Feature을 추출 하여 Classification 합니다. 이때의 문제점은 Object Detection된 객체의 위치를 정확히 알기 어렵습니다.

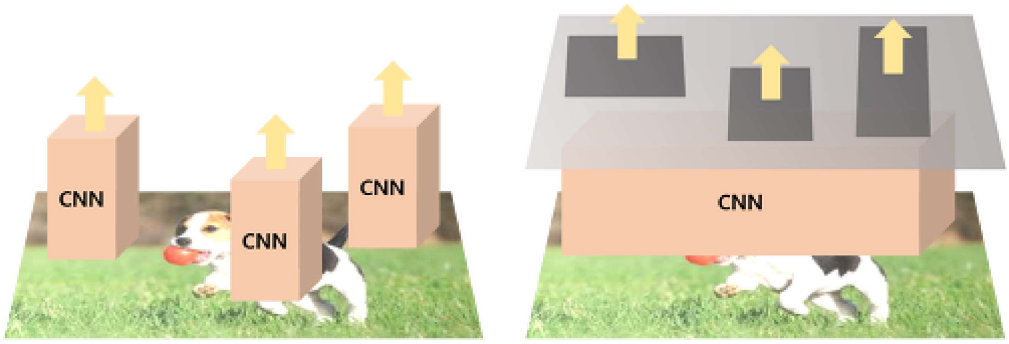
위의 그림과 같이 강아지가 중앙에 존재 하지 않아도(빨간 박스) CNN은 강아지로 Classification 결과를 낼 것 입니다. 이러한 Positional Invariance한 CNN의 특성은 Bounding-Box Regression로 해결이 가능합니다. 이 Bounding-Box

Regression을 통해 빨간 박스는 파란색 박스로 변환이 됩니다. R-CNN은 이미지 한 장당 몇 천개의 Region Proposal된 영역 마다 CNN을 통해 Feature을 추출해야 하기 때문에 학습과 테스트 모두 속도가 매우 느린 단점이 있습니다. 그리고 CNN(fine-tuning), SVM Fitting(Classifier), Bounding-Box Regressor 모두 단계적으로 학습을 해야한다는 단점 또한 있습니다.

FAST R-CNN



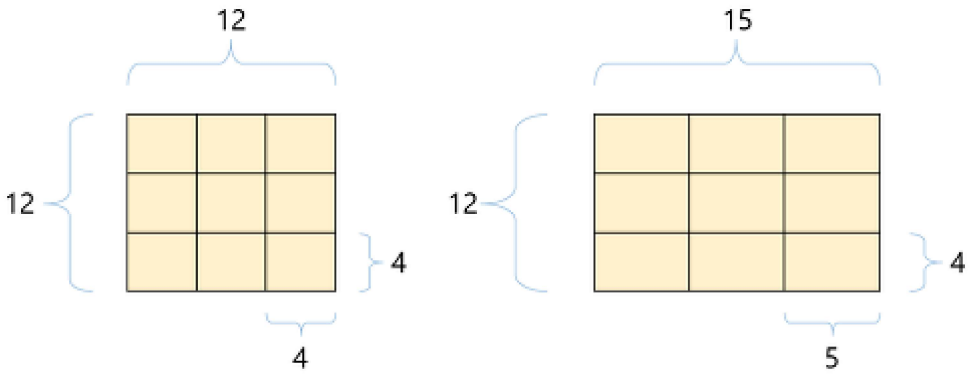
Fast R-CNN은 기존 R-CNN의 속도를 개선한 알고리즘 입니다. Fast R-CNN은 R-CNN에 비해 속도 뿐만 아니라 Single-Stage의 학습(Multi-task Loss 사용)이라는 점에서 차이가 있습니다. Fast R-CNN은 각 추천된 영역 마다 CNN을 적용하는 R-CNN과 달리 CNN 연산을 한 번만 수행합니다.



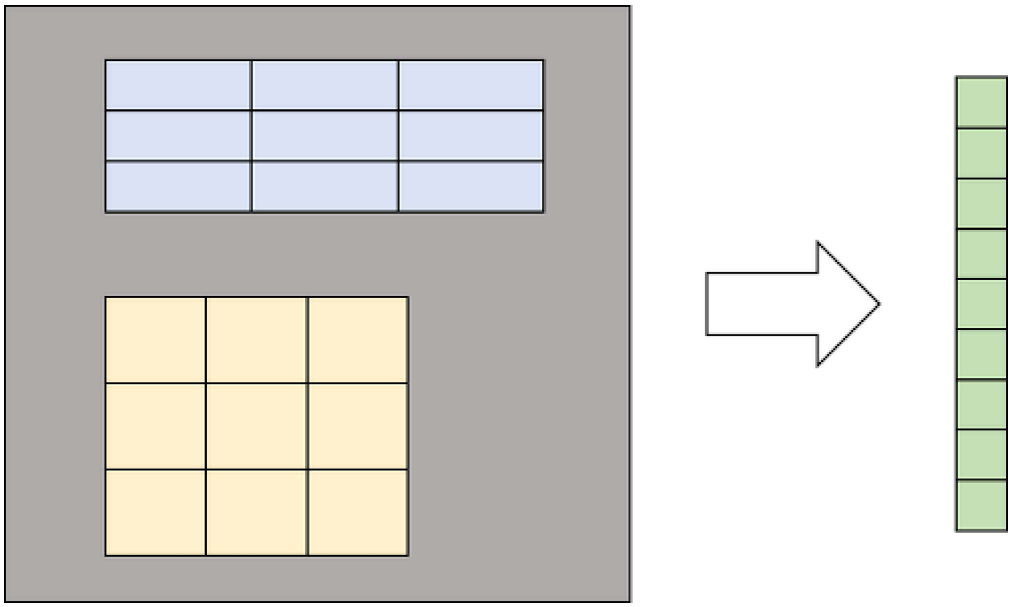
왼쪽이 기존 R-CNN, 오른쪽이 Fast R-CNN 입니다. Feature Map에서 다양한 크기의 Region로부터 일정한 크기의 Feature을 추출 합니다. 기존 R-CNN에서는 이미지의 각 Region을 CNN 모델에 넣기 위해 같은 크기로 crop해야만 했습니다. 이때 당연히 정보손실이 이루어 질 수 밖에 없었습니다. Fast R-CNN은 그러한 정보 손실 없이 Image가 아닌 Feature Map에서 일정 크기로 Feature을 추출하려고 합니다.

RoI Pooling Layer - Single-Level Spatial Pyramid Pooling(SPP)

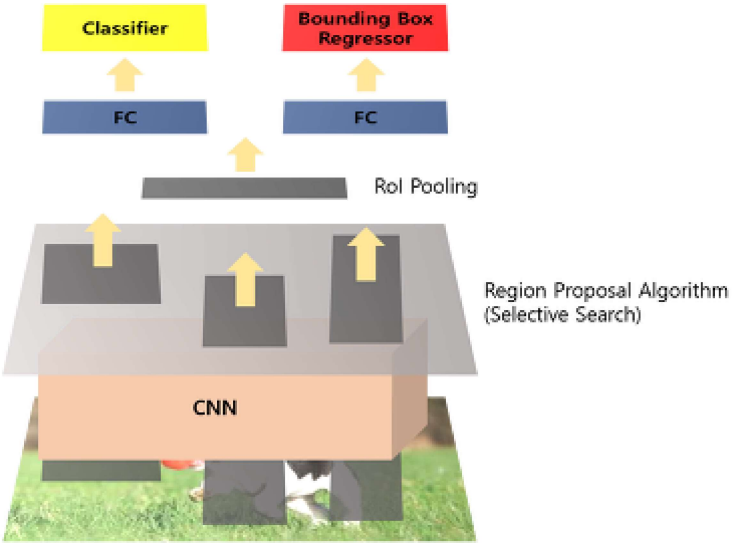
SPP에는 Bag of Words(Bow) 기법이 쓰입니다. Bow는 문서를 자동으로 분류하기 위한 방법중에 하나입니다. 이것을 이용하면 다양한 크기의 Input으로 부터 일정한 크기의 Output(Feature)을 추출 할수 있습니다. 이미지에서 히스토그램으로 R,G,B의 분포를 끄집어 내는 것을 생각하시면 이해가 쉬울 것 같습니다. 하지만 BoW를 사용하면 위치 정보의 손실이 일어납니다. 이때 사용하는 것이 Spatial Pyramid Pooling, SPP 입니다. SPP는 이미지를 일정 개수로 지역을 나누어 각 지역마다 BoW를 적용 하는 기법입니다.



예를들어 12*12 이미지가 있습니다. 이를 4*4의 window를 이용하여 BoW를 적용시키면 3*3 Output이 나옵니다. 마찬가지로 15*12 크기의 이미지가 있을 때 5*4의 window를 이용하면 3*3 Output을 얻을 수 있습니다. 이를 통해 Fast-RCNN에서는 일정한 길이의 Feature을 추출 할 수 있는 RoI Pooling Layer를 만들 수 있습니다.



Feature Map의 특정 영역에 일정 개수(그림에서는 9개)로 나눈 후 Max Pooling 또는 Average Pooling을 적용해 그림과 같이 일정한 Feature Vector를 얻을 수 있습니다.



Fast R-CNN의 전반적인 내용을 다시한번 설명 드리겠습니다. 이미지에 대해 CNN을 진행하여 Feature Map을 얻습니다. Selective Search로 추천받은 지역에 대해 그 부분의 Feature Map을 RoI Pooling을 통해 일정한 크기의 Feature Vector를 얻습니다. 이 Vector을 FC를 이용하여 Classification을 하고, 위치 조정을 위해 Bounding Box Regression을 합니다.

Training Fast R-CNN

Classifier와 Bounding Box Regressor를 함께 최적화하는 학습 방법을 사용합니다. 우선 객체들은 k+1개의 class로 Label 되어집니다. +1인 이유는 배경이라는 class를 포함 하기 위해서 입니다. Fast R-CNN의 Loss Function을 알려드리겠습니다.

$$L(p,\ u,\ t^u,\ v) = L_{cls}(p,\ u) + \lambda[u \geq 1]L_{loc}(t^u,\ v)$$

p는 예측한 class(Label), u는 실제 Label을 의미하고, tu는 실제 Box Coordinates를, v는 예측한 Box Coordinates를 의미합니다. 우선 Classification Loss(Lcls)은 Cross entropy로 구할 수 있습니다. Lloc는 Bounding Box Location에 대한 Loss이며, Smooth L1 Loss로 구합니다.

$$L_{\text{loc}}(t^u, v) = \sum_{i \in \{\text{x,y,w,h}\}} \text{smooth}_{L_1}(t_i^u - v_i),$$

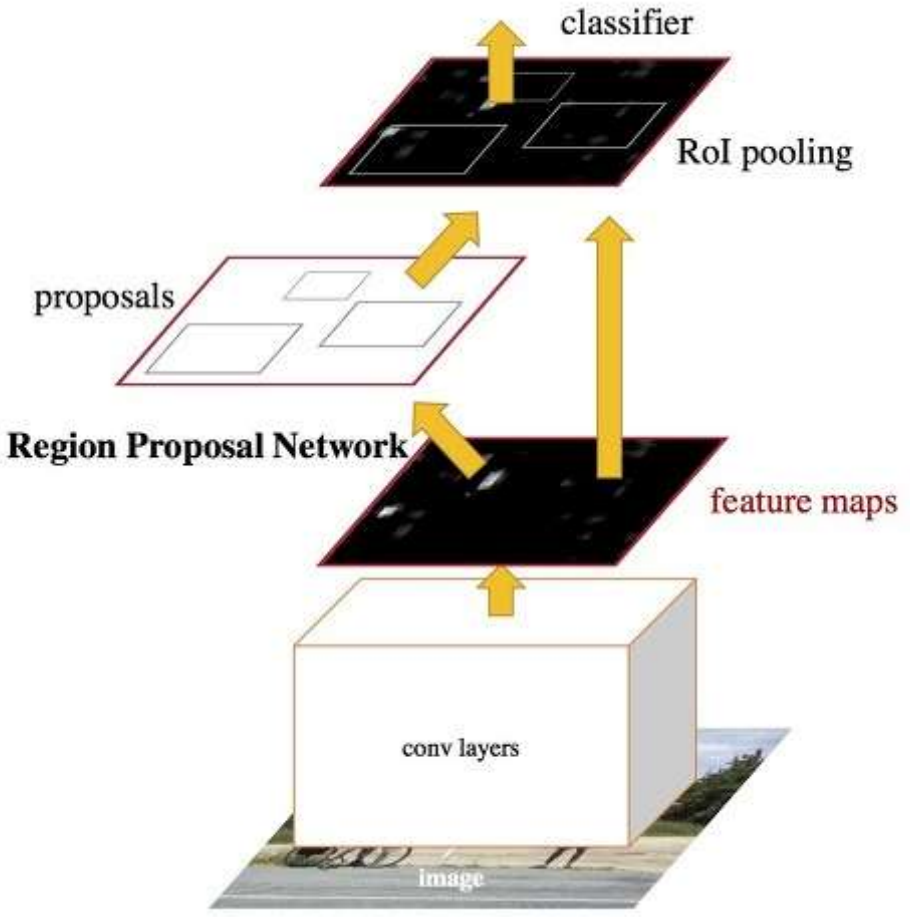
in which

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases}$$

Bounding Box Location은 u가 1보다 크거나 같을 때만 학습이 됩니다. u가 0이면 배경이기 때문에 학습이 이루어 지지 않습니다.

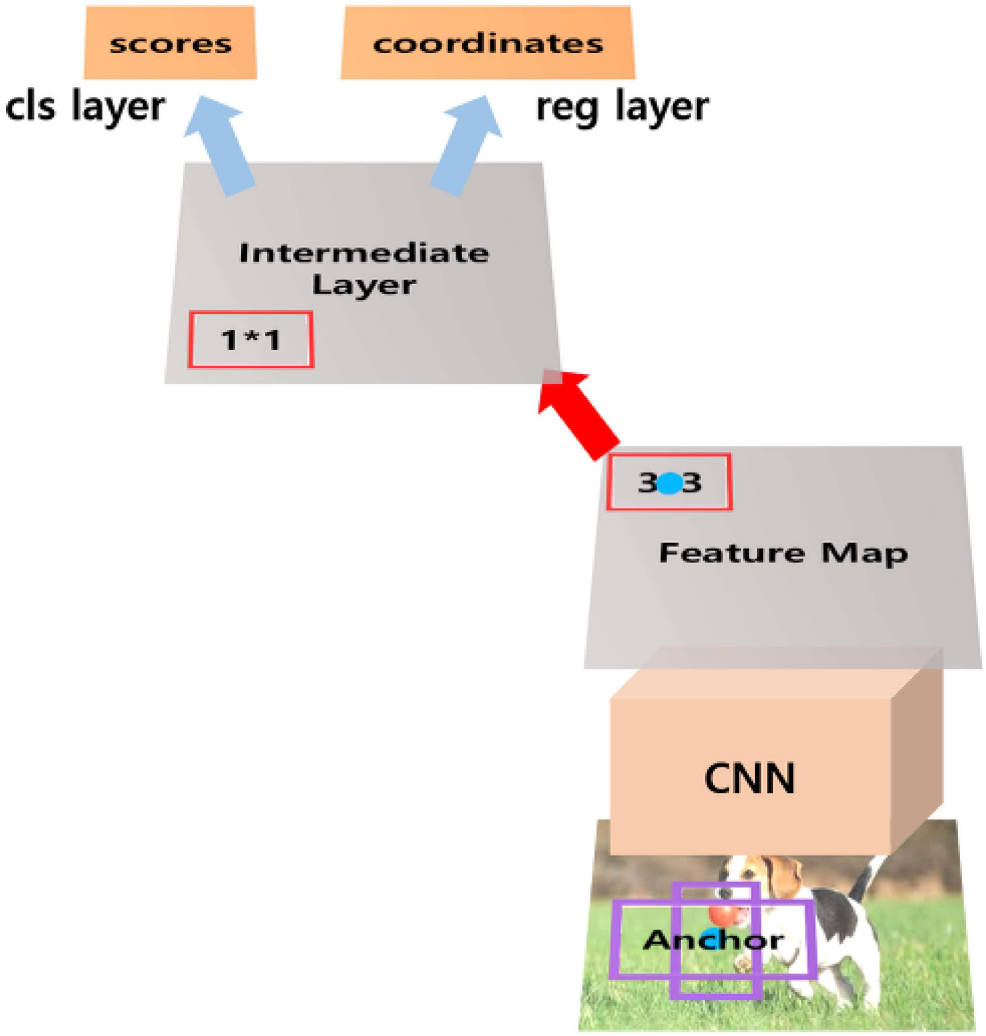
Fast R-CNN은 기존 R-CNN에 비해 20배 이상 빨라졌습니다. 하지만 아직 Real Time 문제를 해결하기에는 무리가 있습니다. 이에 CNN 외부에서 처리 되어지는, 그리고 CPU로 연산이 이루어지는 Region Proposal Algorithm(Selective Search)를 내부로 이동시켜서 함께 학습을 하는 Faster R-CNN 모델이 등장하게 됩니다.

FASTER R-CNN



위에서 말했듯이 Region Proposal을 CNN 내부에 새로 Network를 만들어서 더욱 더 빠르게 Object Detection을 진행 할 수 있습니다. 이 Network는 기존 CNN을 같이 공유 하는것이 목표입니다. 즉 Faster R-CNN은 Fast R-CNN + RPNs(Region Proposal Networks) - SS(Selective Search)로 생각 할 수 있습니다. 위 그림은 논문에 수록 된 Figure 2 입니다. 이를 더 설명하기 쉽게 새로 그림을 그려 설명 드리겠습니다.

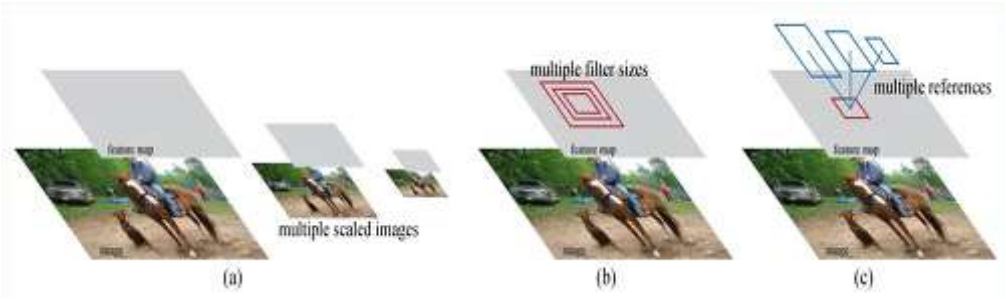
Region Proposal Networks(RPNs)



RPNs은 Image를 Input으로, Object Proposals(coordinates) 와 Objectness Score(scores)을 Output으로 하는 network 입니다. 즉 Feature Map 위 Layer에서 모든 일이 일어나는 것이 아니라, Image에도 Anchor라는 메카니즘을 사용합니다.

이를 자세하게 설명 드리겠습니다. 우선 Image에 CNN(Vgg16으로 예시)을 적용 시켜 나온 Feature Map은(Conv Layer 13) 14*14*512의 벡터 입니다. 이 Feature Map에 3*3 Filter를 Stirde 1, Padding 1을 주고 총 512 output channels이 생성되도록 Convolution 시킬 것입니다. 이렇게 생겨난 Intermediate Layer은 14*14*512가 됩니다.(Stride 1, Padding 1, 3*3 Filter) 이후 Intermediate Layer를 1*1 Filter로 Stride는 1, Padding은 0을 두번 적용시켜서 scores(cls layer)와 coordinates(reg layer)를 얻습니다. 이 때 cls layer은 Object가 있는지, 없는지를 판단하기 위해 2 벡터가 필요하므로 output channels가 2가 됩니다. 반면에 reg layer는 위치를 나타내는 x, y, w, h 총 4 벡터가 필요하므로 output channels가 4가 됩니다. 즉 cls layer은 14*14*2, reg layer은 14*14*4가 됩니다. 하지만 여기에 추가 해야할 것이 있습니다. 바로 Fast R-CNN에서 매우 중요한 Anchor 입니다.

Anchor



Multiple-Scale Detection을 위해 보통 (a) Image의 크기를 조절하는 Image Pyramid를 사용하거나 (b) Filter의 크기를 조절하는 Filter Pyramid기법을 사용합니다. 하지만 RPNs은 (c) Filter와 이미지의 크기를 고정한 후 다양한 Anchor을 사용하는 Anchor Pyramid 기법을 이용합니다. 즉 고정된 크기의 Sliding Window를 적용 하며 그 Window 중앙 위치를 중심으로 Pre-defined 한 다양한 비율, 크기의 Anchor Box 정보를 적용시키며 Feature을 추출합니다.

이 Anchor Box는 논문에서 9개로(3가지 크기당 3가지 비율) 사용 되었습니다. 정확히 말하자면 Feature Map에 3*3 Sliding Window를 적용 할 때 그 중심에 대응되는 기존 Image의 점에서 Anchor Box를 적용 시킵니다. 여기서 의문은 Feature Map의 점과 Image의 점을 어떻게 대응시키냐? 입니다. 이는 쉽게 해결할 수 있습니다. Image가 w*h의 크기를 가졌을 때 Feature map은 w/r*h/r로 표현을 할 수 있습니다. 많이 사용하는 VGG의 경우에는 r이 16입니다. 이를통해 Feature Map의 점과 Image의 점을 대응시킬 수 있습니다.

위의 그림은 기존 Image에 Anchor 중심을 표시한 것 입니다.(그림 출처 (https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/)) Anchor은 기존 MultiBox 메커니즘과 달리 Translation-Invariant 합니다. 한국어로 이동 불변성입니다. 즉 어떤 객체가 이미지 어디에있던지 Anchor은 그 객체를 제대로 판단 할 수 있습니다. 그리고 9개의 Anchor 만을 사용함으로써 파라미터의 개수가 헌저하게 떨어집니다. 즉 small dataset에 대해 Overfitting의 위험도가 헌저하게 낮아집니다.

이제 위의 내용을 조금 수정하겠습니다. Sliding Window를 할때 Anchor 9개에 대해 적용 하기 때문에 cls layer은 2*9의 output channel을 가지게 됩니다. 따라서 14*14*2*9가 됩니다. 그리고 reg layer은 4*9의 output channel을 가지기 때문에 14*14*4*9의 아웃풋을 얻게 됩니다. cls layer을 통해 해당 anchor에 object가 있는지 없는지를 학습하고, reg layer을 통해 Bounding Box 위치를 학습 합니다. 이제 RPNs의 Loss Function에 대해 알아보겠습니다.

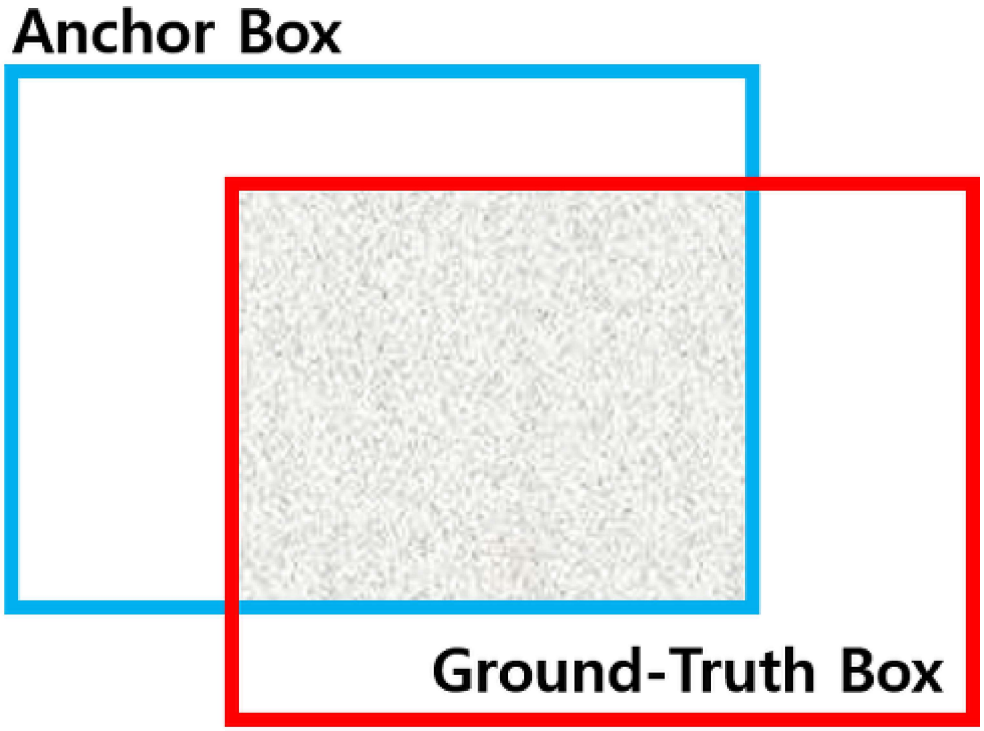
Loss Function

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*)$$

cls Layer에서의 Loss와 reg Layer에서의 Loss를 더해주면 RPNs의 Loss를 구할 수 있습니다.

$$\frac{1}{N_{cls}} \sum_{i} L_{cls}(p_i, p_i^*)$$

우선 cls Layer의 Loss에 대해 살펴 보겠습니다. Anchor에 객체가 있는지 예측 값이 Pi, 실제 값이 Pi*입니다. 예측 값은 cls Layer의 output이 될 것입니다. Pi*는 IoU를 통해 구할 수 있습니다. IoU란 Intersection-over-Union으로 Anchor가 Object인지 Background인지 Labeling하는 기법 입니다.



Anchor Box와 Ground-Truth Box의 교집합/합집합을 하면 IoU를 구할 수 있습니다. 즉 Anchor Box와 Ground-Truth Box가 얼마나 교차되어 있는지를 통해 객체인지 아닌지를 구하는 기법입니다. 이때 두가지 기준으로 Anchor에 Labeling을 합니다.

1. 가장 높은 IoU를 가지고 있는 Anchor

2. IoU > 0.7인 Anchor

입니다. 2번 기준만으로는 아주 드물게 Object인지 판단 못하는 경우가 생겨서 1번 기준을 추가했다고 합니다. 저 기준을 만족하면 Positive, 0.3 미만인 IoU를 가지면 Negative로 Labeling 합니다. 물론 Negative가 Positive에 비해 매우 많아질 것 입니다. 이를 방지 하기 위해 Positive와 Negative anchors를 50:50의 비율을 맞춰 줍니다. 이렇게 Pi*는 Positive Anchor이면 1, Negative Anchor이면 0의 값을 가지게 됩니다. Lcls 는 Log loss를 사용하였습니다. 이렇게 cls Layer의 Loss는 계산이 됩니다.

$$\lambda \frac{1}{N_{reg}} \sum_{i} p_i^* L_{reg}(t_i,\ t_i^*)$$

reg Layer의 Loss에는 Pi*가 곱해져 있습니다. 즉 이것의 IoU를 통해 객체가 맞으면 학습이 이루어 질 것이고, 객체가 아니면 0이 곱해져서 학습이 이루어 지지 않을 것입니다. ti는 reg Layer에서 나온 예측된 Bounding Box 값(x,y,w,h)이고 ti*는 실제 Ground-Truth Box의 값(x,y,w,h)입니다. 그리고 Lreg는 Fast R-CNN에서도 쓰는 Smooth L1 Loss Function을 사용합니다.

$$\begin{aligned}
t_{\mathrm{x}} &= (x - x_{\mathrm{a}})/w_{\mathrm{a}}, \quad t_{\mathrm{y}} = (y - y_{\mathrm{a}})/h_{\mathrm{a}}, \\
t_{\mathrm{w}} &= \log(w/w_{\mathrm{a}}), \quad t_{\mathrm{h}} = \log(h/h_{\mathrm{a}}), \\
t_{\mathrm{x}}^* &= (x^* - x_{\mathrm{a}})/w_{\mathrm{a}}, \quad t_{\mathrm{y}}^* = (y^* - y_{\mathrm{a}})/h_{\mathrm{a}}, \\
t_{\mathrm{w}}^* &= \log(w^*/w_{\mathrm{a}}), \quad t_{\mathrm{h}}^* = \log(h^*/h_{\mathrm{a}}),
\end{aligned}$$

우선 위와같은 형태로 구한 x,y,w,h와 실제 x*,y*,w*,h*를 통해 계산합니다. 이때 xa,ya,wa,ha는 Anchor를 통해 구할 수 있습니다.

$$L_{\mathrm{loc}}(t^u, v) = \sum_{i \in \{\mathrm{x,y,w,h}\}} \mathrm{smooth}_{L_1}(t_i^u - v_i),$$

in which

$$\mathrm{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases}$$

이것을 통해 Smooth L1 Loss Function에 넣습니다. vi는 예측한 t(x,y,w,h)이고, tiu는 실제 t(x*,y*,w*,h*)입니다. Smooth L1 Loss Function은 -1~1 구간은 L2, 나머지는 L1을 쓰는 방식입니다. L1과 L2 Loss Function 은 아래와 같은 차이가 있습니다.

L2 loss function	L1 loss function
Not very robust	Robust
Stable solution	Unstable solution
Always one solution	Possibly multiple solutions

경계값 밖에서는 L2 Loss보다 덜 민감한 L1 Loss를 적용했습니다. RPNs의 Loss Function에서 lambda는 Ncls와 Nreg의 차이 때문에 곱해준다고 합니다. 이는 논문의 EXPERIMNETS에서 큰 의미는 없다고 하였습니다. RPNs은 이런 Loss Function을 가지고 학습이 됩니다.

Training RPNs

RPNs의 Loss Function을 가지고 어떻게 학습이 이루어지는지 설명 드리겠습니다. 우선 end-to-end로 Back Propagtion이 이루어집니다. 이때 SGD(Stochastic Gradient Descent)방식을 사용합니다. Anchor들의 Sample은 위에서 설명한 것처럼 50대50 비율로 만들어 놓습니다. 그리고 처음 60k의 mini-batches에는 0.001의 Learning Rate로, 다음 20k의 mini-batches에는 0.0001의 Learning Rate로 학습 시킵니다. 이때 Momentum은 0.9를 주었고 Weight Decay는 0.0005를 주어서 학습을 했다고 합니다. 이제 RPNs과 Fast R-CNN으로 이루어진 Faster R-CNN에서 어떻게 CNN의 Feature Map을 공유해서 학습이 이루어 지는지 설명 드리겠습니다.

Sharing Features for RPNs and Fast R-CNN

논문에서는 4-Step Alternation Training방식을 사용하였습니다.

1. RPNs 학습

2. RPNs의 나온 Region Proposal로 Fast R-CNN 학습

3. RPNs 학습

4. RPNs의 나온 Region Proposal로 Fast R-CNN 학습

이것을 풀어서 설명해 드리겠습니다.

0. 미리 학습된 CNN인 ImageNet M0가 있습니다.

1. 이를 기반으로 RPNs을 학습시켜 M1을 얻습니다.

2-1. M1을 통해 RPNs에서 Region Proposal P1을 얻습니다.

2-2. P1과 M0을 이용하여 Fast R-CNN을 학습 시켜 M2를 얻습니다.

3. M2를 이용하여 RPNs을 학습시켜 M3을 얻습니다.

4-1. M3을 통해 RPNs에서 Region Proposal P2를 얻습니다.

4-2. P2와 M3을 통해 Fast R-CNN을 학습 시켜 M4를 얻습니다.

매우 복잡합니다. 이유로는 논문 마감일이 얼마 안남아서.. 급하게 구현하다보니.. 비효율적인 구조를 사용하였다고 합니다. 추후 Joint Training 방식으로 약 1.5배의 성능을 얻었다고 합니다.

Reference

Fast R-CNN, Microsoft Research, Ross Girshick, Fast R-CNN, arXiv:1504.08083. (url) (https://arxiv.org/pdf/1504.08083.pdf)

Faster R-CNN, Microsoft Research, Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, arXiv:1506.01497. (url) (https://arxiv.org/pdf/1506.01497.pdf)

CNN : (url) (http://pythonkim.tistory.com/52), Selective Search(SS) : (url) (https://m.blog.naver.com/PostView.nhn?blogId=laonple&logNo=220918802749&proxyReferer=https%3A%2F%2Fwww.google.co.kr%2F), BoW : (url) (http://darkpgmr.tistory.com/125), Fast R-CNN : (url) (http://m.blog.daum.net/_blog/_m/articleView.do?blogid=0YjD7&articleno=8)

Faster R-CNN : (한글 블로그1) (https://jamiekang.github.io/2017/05/28/faster-r-cnn/) (한글 블로그2) (https://curt-park.github.io/2017-03-17/faster-rcnn/) (한글 블로그3) (https://blog.lunit.io/2017/06/01/r-cnns-tutorial/) (PR-012) (https://www.youtube.com/watch?v=kcPAGIgBGRs) (CS231n) (https://www.youtube.com/watch?v=_GfPYLNQank&t=3126s) (영문 블로그1) (https://medium.com/@smallfishbigsea/faster-r-cnn-explained-864d4fb7e3f8) (영문 블로그2) (https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/)

#AI #DeepLearning #Seminar

댓글

kbj

댓글을 작성해주세요

SEND

 (mailto:bjbj8083@gmail.com)

 (https://github.com/byeongjokim)

 (https://www.instagram.com/byeongjo_)