FPN: Feature Pyramid Networks for Object Detection

FPN(Feature Pyramid Net)

Abstract

스케일 불변성(scale-invariance)을 얻기 위해 Feature Pyramids를 사용하는 것은 필수적.

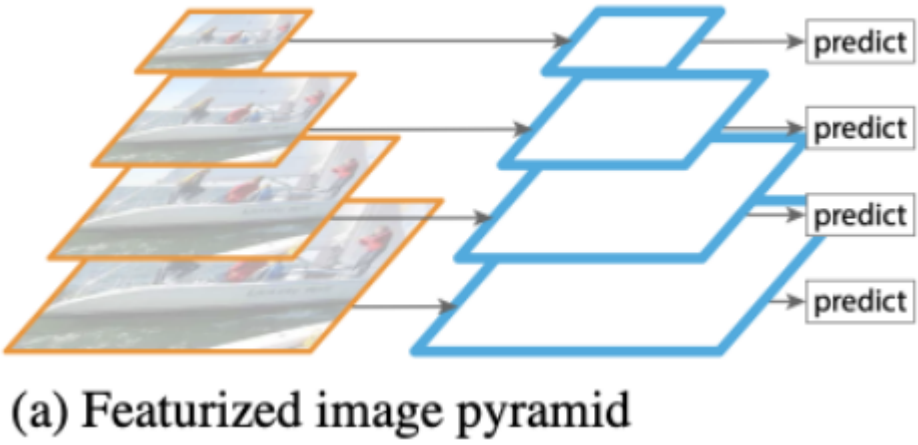
하지만 Feature Pyramids는 많은 연산량과 메모리가 필요하여, detection 속도 느려짐.

이를 개선하기 위해 제안된 방법이 FPN.

컴퓨팅 자원을 적게 차지하면서 다양한 크기의 객체 인식 방법.

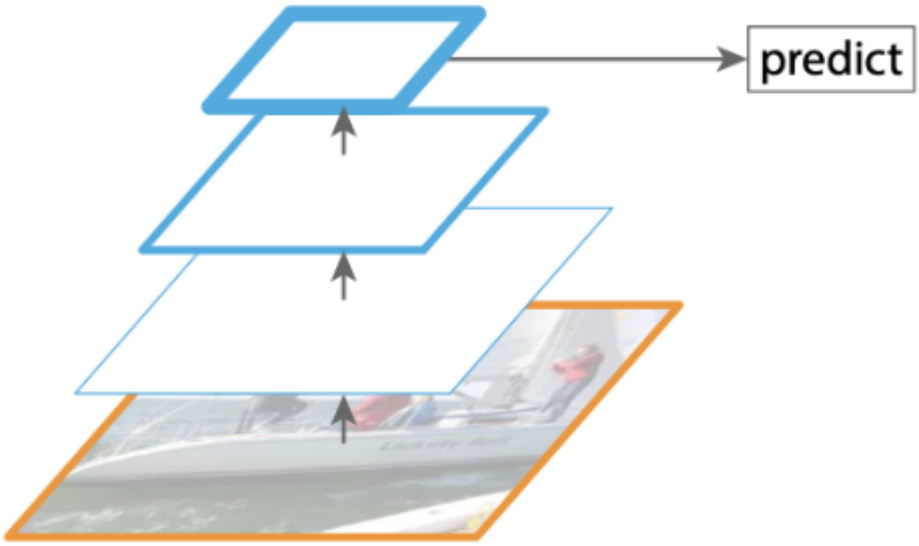
FPN을 Faster R-CNN에 사용하여 최고 성능 얻음.

Feature Pyramid



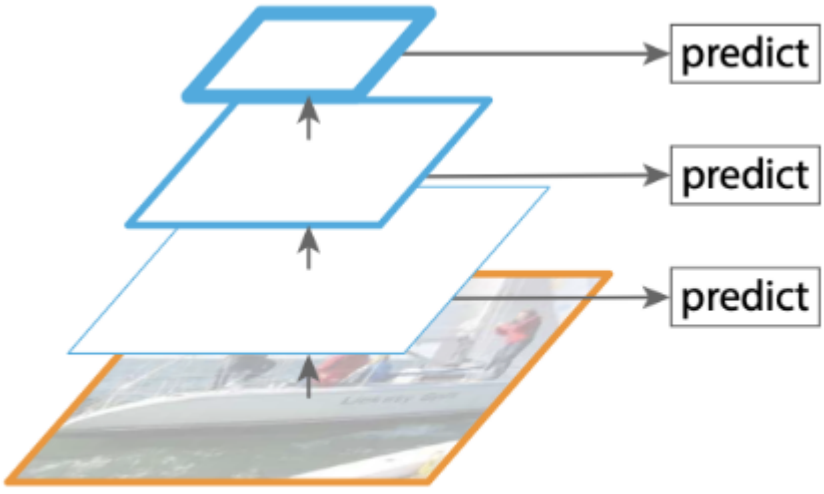
(a) Featurized image pyramid

(a) Input image의 크기를 다양하게 resize하고 이미지 네트워크에 입력하는 방법. 다양한 크기의 객체를 포착하는데 좋은 결과를 보여줌. 하지만 연산량이 많아 추론 속도 매우 느림, 메모리를 굉장히 많이 사용함. → 현실에서 사용 어려움.



(b) Single feature map

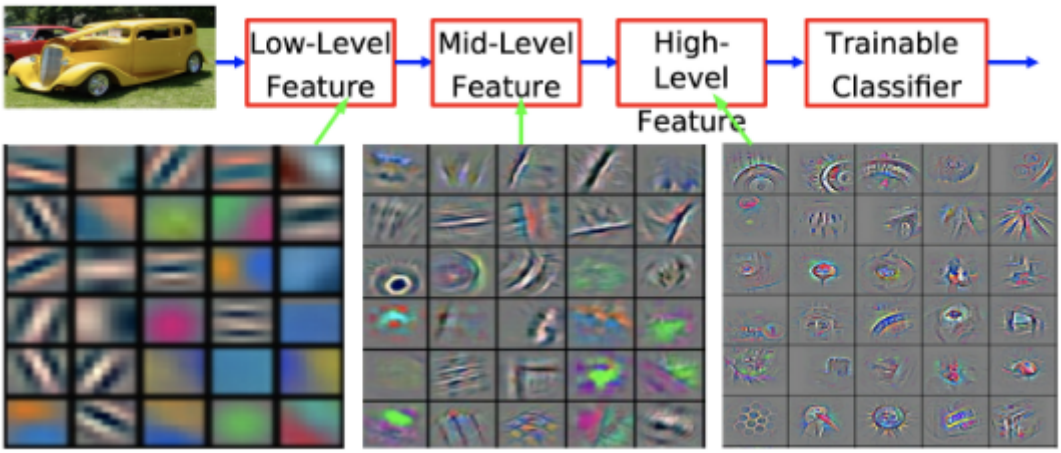
(b) Input image를 네트워크에 입력하여 최종 단계에 feature map에서 object detection 수행. YOLO v1이 이러한 기법 사용. Multi scale을 사용하지 않고 한번에 특징을 압축. 마지막에 압축된 특징 사용 → (a)보다 연산량은 작지만 성능 떨어짐.



(c) Pyramidal feature hierarchy

(c) CNN 신경망을 통과하는 중간 과정에 생성되는 feature map 각각에 object detection 수행. SSD가 이러한 기법 사용. 작은 물체에 대한 정보 살리면서 object detection을 수행할 수 있음. But!!!! 상위 레이어에서 얻게 되는 추상화 된 정보 활용 못함(semantic gap).

▼ 여기서 Pyramid란?



- CNN에서 얻을 수 있는 서로 다른 해상도의 feature map을 쌓아올린 형태
- 입력층에 가까울수록 high resolution, low-level feature(ex. 가장자리 곡선)
- 입력층에서 멀수록 low resolution, high-level feature(class 추론가능한 특징)

→ Object Detection 모델은 Pyramid의 각 level의 feature map을 일부 혹은 전부 사용하여 예측 수행

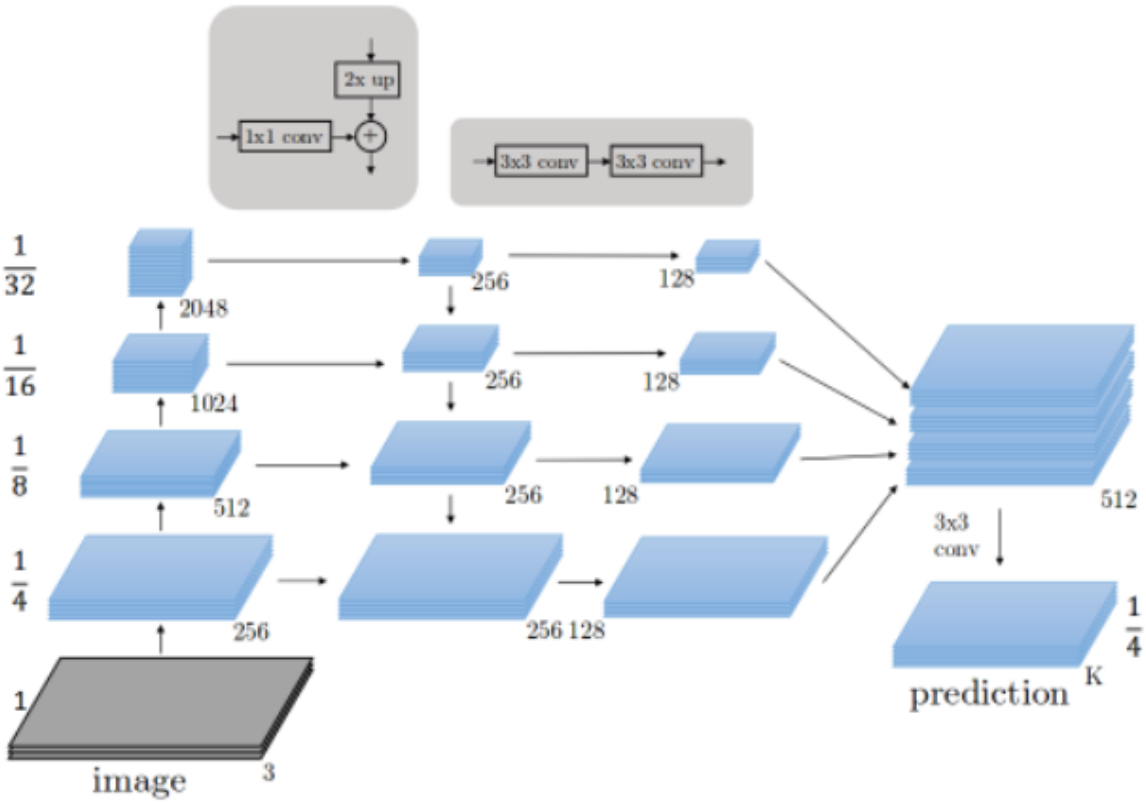
Feature Pyramid Network

FPN은 완전 새롭게 설계된 모델이 아니라 기존 convolutional network에서 지정한 layer별로 feature map을 추출하여 수정하는 네트워크라고 이해하면 편할듯.

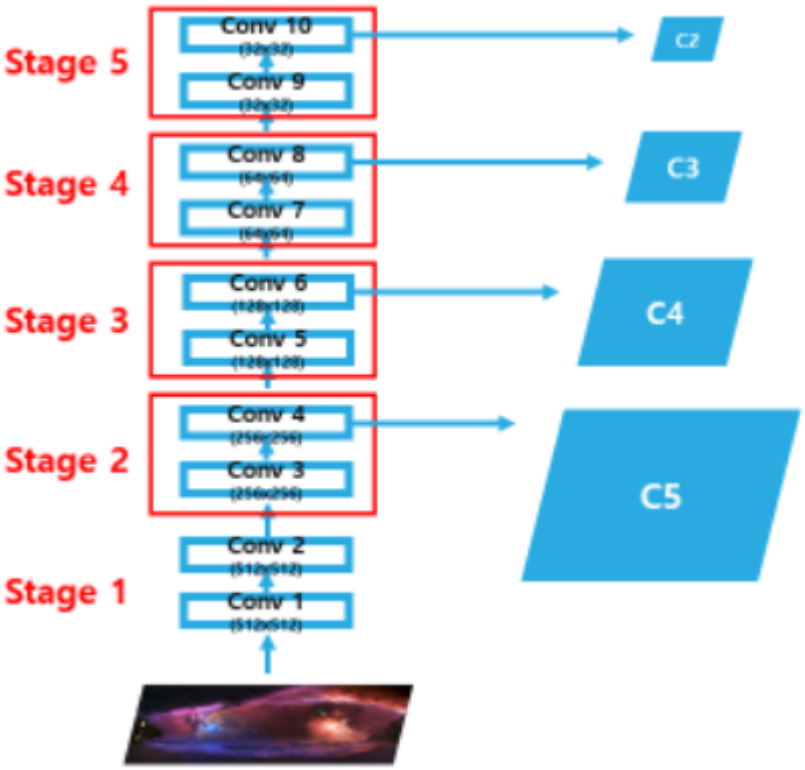
Single-scale 이미지를 Convolutional network에 입력 → 다양한 scale의 feature map 출력(논문에서는 ResNet 사용)

Bottom-up : resolution을 줄여가며 feature를 얻는 forward 과정

Top-down & Lateral connection : low resolution이지만 feature를 다시 원래 resolution으로 맞춰주면서 기존의 feature와 결합



Bottom-up



Image를 convolution network에 입력하여 2배씩 작아지는 feature map 추출하는 과정.

각 stage의 마지막 layer의 output feature map 추출.

논문에서는 같은 크기의 feature map을 출력하는 layer를 같은 stage에 속해있다고 정의.

각 stage별로 마지막 layer를 pyramid level로 지정 → 더 깊은 layer일수록 더 강력한 feature 보유하기 때문

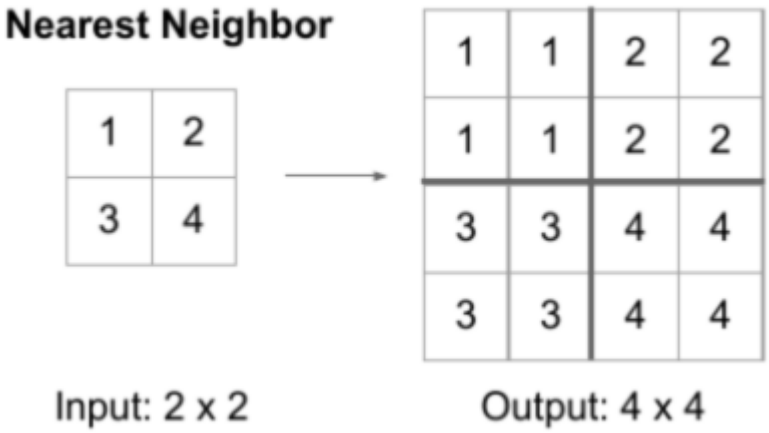
(그림설명)

ResNet의 경우 각 stage의 마지막 residual block의 output feature map을 활용하여 feature pyramid를 구성하며 각 output을 **{c5, c4, c3, c2}** 라고 지정. 이는 conv2, conv3, conv4, conv5의 output feature map임을 의미하며, 각각 **{4, 8, 16, 32} stride**를 가지고 있음. 여기서 {c5, c4, c3, c2}은 각각 원본 이미지의 1/4, 1/8, 1/16, 1/32 크기를 가진 feature map. conv1의 output feature map은 너무 많은 메모리를 차지하기 때문에 피라미드에서 제외.

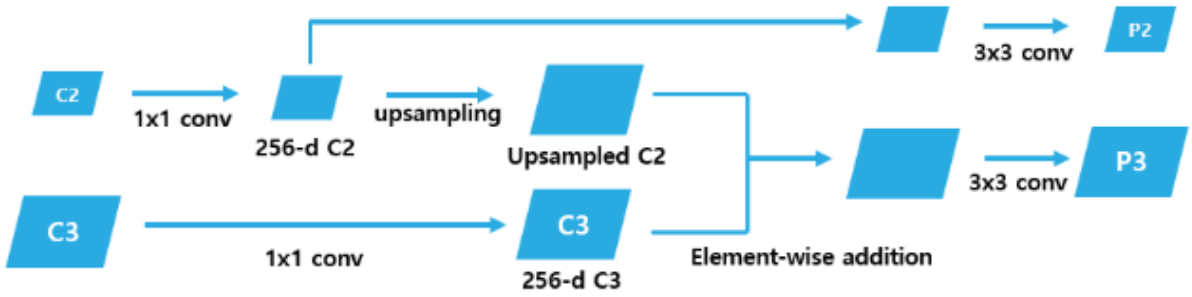
Top-down and Lateral connections

Pyramid level에 있는 Feature map을 2배 Upsampling, Channel 수를 동일하게 맞춰주는 과정.

Bottom-up 단계에서 stage가 진행될수록 사이즈가 1/2배되므로, upsampling을 2배로 해주면 사이즈가 맞아짐.



Upsampling 단계에서는 계산의 편의성을 위해 **nearest neighbor upsampling** 방식 사용.



Upsampling된 feature map과 바로 아래 level의 feature map과 element-wise addition 연산 진행 → **Lateral connections**

이후 각각의 feature map에 3x3 conv 연산 적용 → feature map 얻음

이 과정을 통해 4개의 서로 다른 Scale의 feature map 생성

(그림설명)

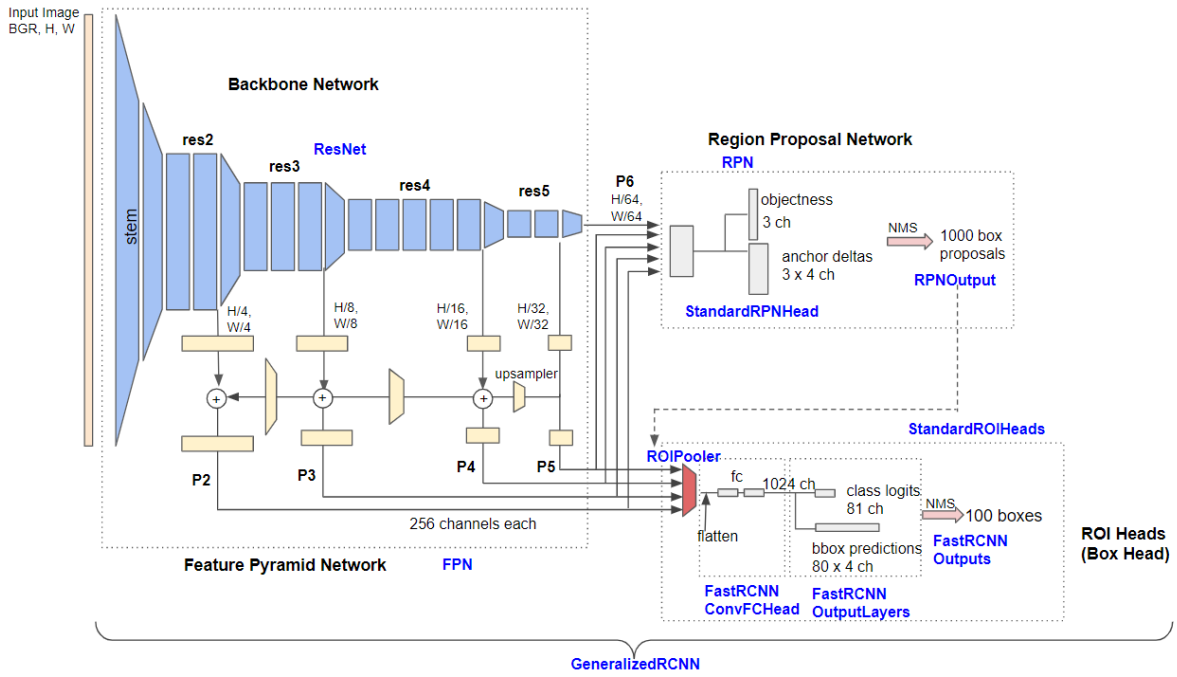
Lateral connections 과정을 수행. 이후 각각의 feature map에 3x3 conv 연산을 적용하여 얻은 feature map을 각각 **{p2, p3, p4, p5}**. 이는 각각 {c2, c3, c4, c5} feature map의 크기와 같음.

가장 높은 level에 있는 feature map c2의 경우 1x1 conv 연산 후 그대로 출력하여 p2를 얻음.

Application

저자들은 Feature Pyramid Network 기법을 기존 Object Detection 모델들에 적용하여 실제로 잘 작동하는 지를 확인해보고자 했음. 논문에서는 Faster R-CNN의 RPN과 Classifier에 적용하여 실험 진행.

ResNet(backbone) + Faster R-CNN with FPN 과정 설명



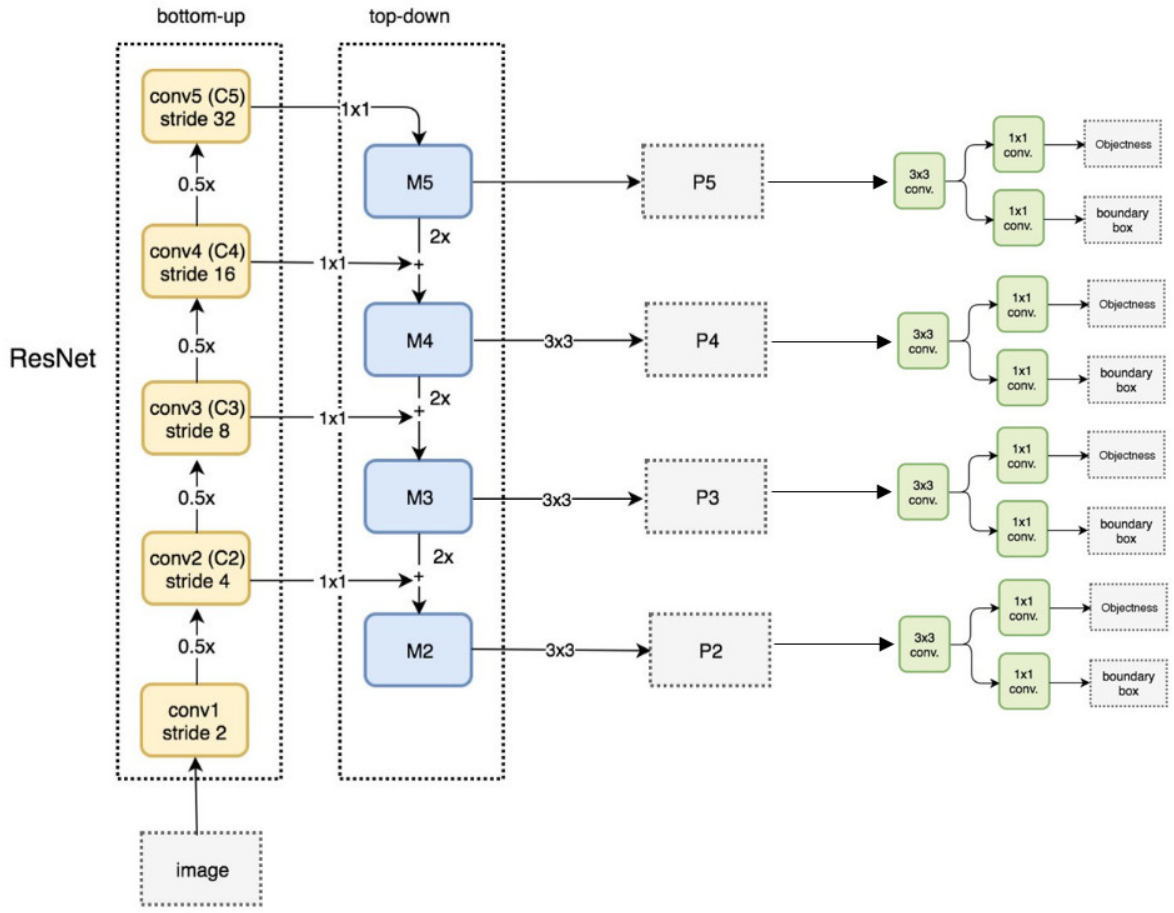
1) Build Feature Pyramid by FPN

먼저 ResNet 기반의 FPN에 이미지를 입력한 후 **Bottom-up pathway**을 거쳐 원본 이미지의 1/4, 1/8, 1/16, 1/32 크기에 해당하는 feature map {c5, c4, c3, c2}을 출력. 이후 **Top-down pathway** 과정을 통해 1x1 conv 연산을 적용하여 모든 feature map의 channel 수를 256으로 맞춰주고 크기를 2배로 upsampling. 마지막으로 **Lateral connections**을 통해 각 feature map을 바로 아래 pyramid level에 존재하는 feature map과 element-wise addtion 연산 수행. 이후 3x3 conv 연산을 수행하여 {p5, p4, p3, p2} feature map을 출력.

앞선 과정을 통해 얻은 feature pyramid {p5, p4, p3, p2}는 RPN와 RoI pooling 시 사용.

- **Input** : single-scale image
- **Process** : build feature pyramid
- **Output** : multi-scale feature map {p5, p4, p3, p2}

2) Class score and Bounding box by RPN



앞선 과정에서 얻은 feature map {p2, p3, p4, p5}를 **RPN(Region Proposal Network)**에 입력. 이 때 각 feature map을 위의 그림과 같이 개별적으로 RPN에 입력하여 각각의 class score과 bounding box regressor를 출력. 이후 Non maximum suppression 알고리즘을 적용하여 class score가 높은 상위 1000개의 region proposal만을 출력.

- **Input** : multi-scale feature map {p2, p3, p4, p5}
- **Prcoess** : region proposal and Non maximum suppression
- **Output** : 1000 region proposals

3) Max pooling by RoI pooling

1)번 과정에서 얻은 **multi-scale feature map {p2, p3, p4, p5}**와 2)번 과정을 통해 얻은 **1000개의 region proposals**를 사용하여 **RoI pooling**을 수행. Fast R-CNN은 single-scale feature map만을 사용한 반면, FPN을 적용한 Faster R-CNN은 multi-scale feature map을 사용하기 때문에 **region proposals를 어떤 scale의 feature map과 매칭**시킬지 결정해야함.

$$k = \lfloor k_0 + \log_2(\sqrt{wh}/224) \rfloor$$

논문에서는 위와 같은 공식에 따라 region proposal을 k번째 feature map과 매칭. w, h는 RoI(=region proposal)의 width, height에 해당하며, k는 pyramid level의 index, k0은 target level을 의미. 논문에서는 k0=4로 지정.

ex) 만약 112 x 112 크기의 RoI가 입력되면 k=3, P3에 매핑.

만약 512 x 512 크기의 RoI가 입력되면 4+log2(2.28) = 5.11로 P5에 매핑.

직관적으로 봤을 때 RoI의 scale이 작아질수록 낮은 pyramid level, 즉 해상도가 높은 feature map에 할당하고 있음을 알 수 있음.

위와 같은 공식을 사용하여 region proposal과 index가 k인 feature map을 통해 RoI pooling을 수행. 이를 통해 고정된 크기의 feature map을 얻을 수 있음.

- **Input** : multi-scale feature map {p2, p3, p4, p5} and 1000 region proposals
- **Process** : RoI pooling
- **Output** : fixed sized feature maps

4) Train Faster R-CNN

RoI pooling을 통해 얻은 고정된 크기의 feature map을 Fast R-CNN에 입력한 후 전체 네트워크를 multi-task loss function을 통해 학습

Experiment

method	backbone	competition	image pyramid	test-dev $AP_{@.5}$	test-dev AP	test-dev AP_s	test-dev AP_m	test-dev AP_l	test-std $AP_{@.5}$	test-std AP	test-std AP_s	test-std AP_m	test-std AP_l
ours, Faster R-CNN on **FPN**	ResNet-101	-		**59.1**	**36.2**	**18.2**	**39.0**	48.2	**58.5**	**35.8**	**17.5**	**38.7**	47.8
*Competition-winning **single-model** results follow:*													
G-RMI†	Inception-ResNet	2016		-	34.7	-	-	-	-	-	-	-	-
AttractioNet‡ [10]	VGG16 + Wide ResNet§	2016	✓	53.4	35.7	15.6	38.0	**52.7**	52.9	35.3	14.7	37.6	**51.9**
Faster R-CNN +++ [16]	ResNet-101	2015	✓	55.7	34.9	15.6	38.7	50.9	-	-	-	-	-
Multipath [40] (on minival)	VGG-16	2015		49.6	31.5	-	-	-	-	-	-	-	-
ION‡ [2]	VGG-16	2015		53.4	31.2	12.8	32.9	45.2	52.9	30.7	11.8	32.8	44.8

좋더라~

시간이 오래 지났기 때문에 해당 실험파트는 설명 생략. 의미가 없다~

Feature Pyramid 기법은 범용적으로 적용이 가능하여 이후에 많은 모델들에게 채택받았음. 논문에서는 Faster RCNN에 적용한 사례를 보여주었지만, 이 후 1 Step 모델들에도 접목되었다고함. 이 후 양방향으로 피쳐맵을 합쳐주던가, Skip Connection을 복잡하게 꼬아버리는 등의 변형이 등장하기도 함. 이렇듯 범용적으로 적용할 수 있는 기법의 발견은 큰 영향력을 가질 수 있는 것 같음.