Detection

https://blog.lunit.io/2017/06/01/r-cnns-tutorial/

> **R-CNN : https://ganghee-lee.tistory.com/35**

Computer Vision

1. Classification
2. Object Detection

1-stage detector :

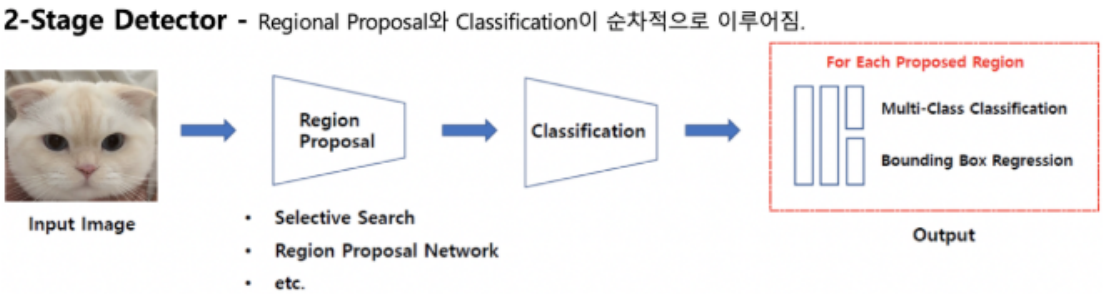
Regional proposal과 classification이 동시에 이루어진다.

→ classification과 localization 문제를 동시에 해결함

convolution network로 classification, box regression(localization) 수행

⇒ 2-stage detector 보다 정확도가 떨어지지만, 간단하고 빠르다.

2-stage detector :



Regional proposal과 classification이 순차적으로 이루어진다.

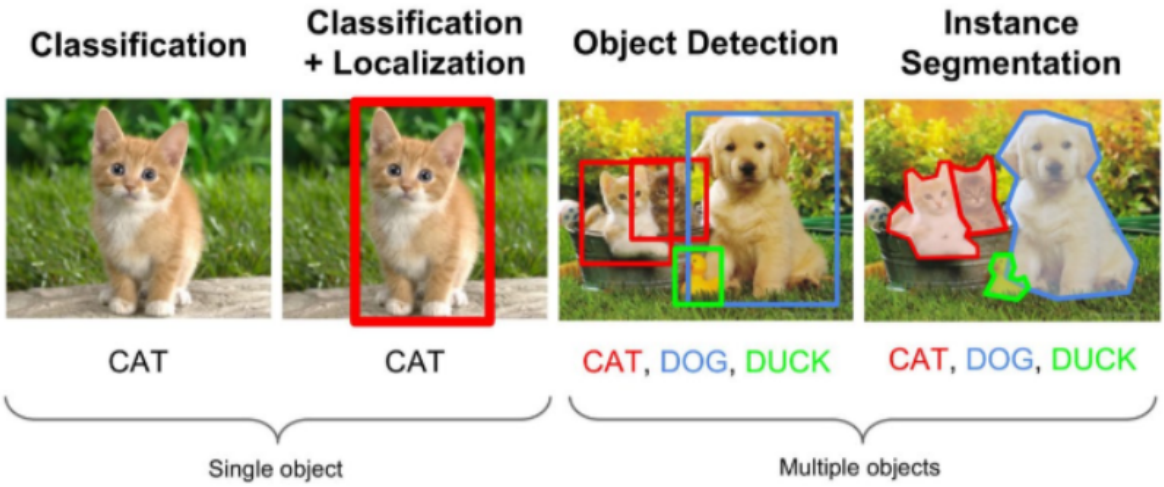
→ → classification과 localization 문제를 순차적으로 해결함

selective search, region proposal network 등의 알고리즘 및 네트워크를 통해 object가 있을만한 영역(ROI : Region of Interest) 추출 → convolution

network를 통해 classification, box regression(localization)수행

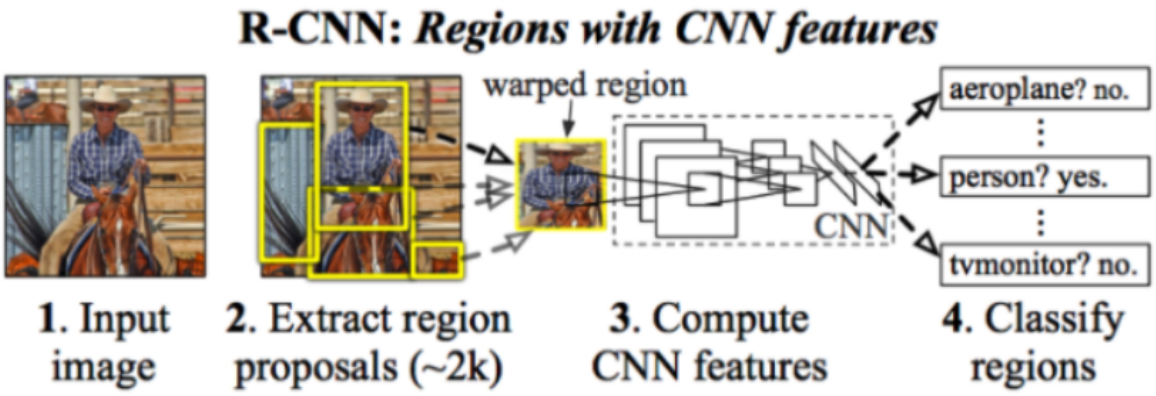
3. Image Segmentation

4. Visual relationship



1. Classification / 2. Object Detection / 3. Image segmentation

R-CNN



과정 :

> (1)region proposal 추출 → 각 region proposal별로 CNN 연산 → (2)classification, (3)bounding box regression

input image

→ regional proposal ouput 추출 (약 2000개, selective search 알고리즘 사용)

→ 이 output을 동일 input size로 warp

마지막 FC layer에서 input size는 고정이기 때문에 convolution layer에 대한 output size도 동일하게

→ 2000개의 warped image를 각각 CNN 모델에

→ 각각의 convolution 결과에 대해 classification 진행

학습 과정

1. ImageNet classification 데이터로 ConvNet을 pre-train 시킨다. → 모델 M을 얻는다.
2. M을 기반으로 → object detection 데이터로 ConvNet을 fine-tune(미세조정)시킨 모델 M'를 얻는다.
3. object detection 데이터 각각의 이미지에 존재하는 모든 region proposal들에 대해 모델 M'으로 feature vector F를 추출하여 저장한다.
4. 추출된 F를 기반으로 SVM, linear bounding-box regressor를 학습

region proposal : object가 있을 만한 영역을 찾는 모듈

sliding window의 비효율성을 극복하기 위한 방식 → 느림

selective search 알고리즘 사용

1. 색상, 질감, 영역크기 등.. 을 이용해 non-object-based segmentation을 수행한다.

이 작업을 통해 좌측 제일 하단 그림과 같이 많은 small segmented areas들을 얻을 수 있다.

2. Bottom-up 방식으로 small segmented areas들을 합쳐서 더 큰 segmented areas들을 만든다.

3. (2)작업을 반복하여 최종적으로 2000개의 region proposal을 생성한다.

CNN : 각각의 영역으로부터 고정된 크기의 feature vector 추출

최종적으로 CNN을 거쳐 각각의 region proposal로부터 4096-dimentional feature vector를 뽑아냄

SVM : SVM은 CNN으로부터 추출된 각각의 feature vector들의 점수를 class별로 매기고, 객체인지 아닌지, 객체라면 어떤 객체인지 등을 판별하는 역할을 하는 Classifier

CNN 모델로부터 feature가 추출되면 linear SVM을 통해 classification 진행함.

왜 softmax를 쓰지 않는가? 이 실험에서는 CNN fine-tunning을 위한 학습 데이터가 적어서 softmax를 사용하면 성능이 낮아짐

Fast R-CNN : https://ganghee-lee.tistory.com /36

R-CNN의 한계 극복

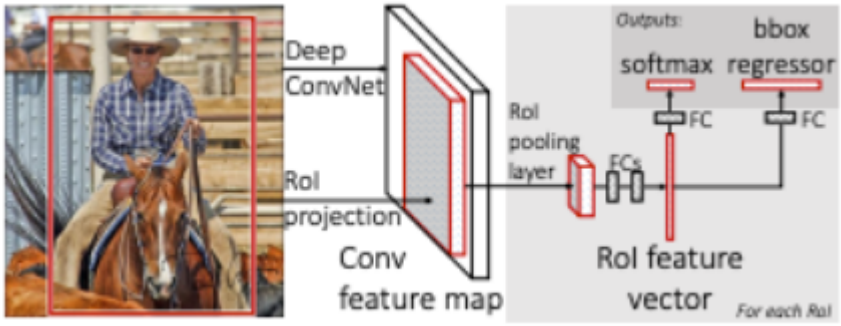
1. RoI pooling을 통해 속도 개선
2. CNN 특징 추출로부터 classification, bounding box regression까지 하나의 모델에서 학습

⇒ 학습 단계가 줄었다는 뜻이다.

- R-CNN에서는 Softmax classifier와 linear bounding-box regressor를 따로 학습했습니다.⇒ 반면, Fast R-CNN에서는 두 함수의 loss를 더한 multi-task loss를 기반으로 동시에 두 가지 task를 학습합니다.
- R-CNN : 여러장의 이미지에서 → 랜덤하게 N개의 영역을 샘플링한 mini-batch
- Fast R-CNN : 1~2장의 이미지에서 N개의 영역을 샘플링한 mini-batch

  ⇒ 연산량을 줄였다는 뜻이다

  오버피팅은 일어나지 않는 걸까.....?



Process

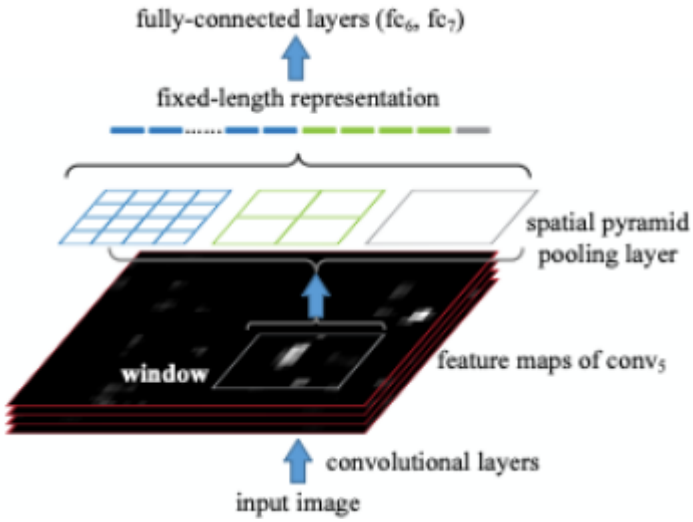
> 💡 (1)region proposal 추출 → 전체 image CNN 연산 → RoI projection, RoI Pooling → (2)classification, bounding box regression

region proposal을 CNN level로 통과시켜 classification, bounding box regression을 하나로 묶었다.

1. RoI를 찾는다. (selective search 사용)
2. 전체 이미지를 CNN에 통과시켜 feature map 추출
3. 1의 RoI를 feature map 크기에 맞춰서 projection
4. 3의 RoI를 Pooling → 고정된 크기의 **feature vector**를 얻는다.
5. 4의 feature vector를 FC layer에 통과, 구 브랜치로 나눔
6. 하나는 softmax를 통과하여 RoI에 대해 object classification 한다
7. bounding box regression을 통해 selective search로 찾은 box 위치 조정

Spatial Pyramid Pooling (SPP)

기존에 사용하던 Bag-of-words(BoW)은 이미지가 가진 feature들의 위치 정보를 잃어버린다.



1. CNN에 이미지 통과시킴 → feature map 추출
2. 4*4, 2*2, 1*1 피라미드로 feature map 나눠줌 (bin = 피라미드 1칸)
3. bin 안에서 max pooling(average를 해도 됨) 적용 → 각 bin마다 값 추출 → 피라미드 크기만큼 max 값 추출 → 3개의 피라미드 결과를 이어붙여 고정된 크기 vector를 만듦

→ 이 vector가 FC layer의 input으로 들어감

fast R-CNN에서는 SPP layer의 single level pyramid를 사용하고, 이것을 RoI Pooling layer라고 명칭.

RoI pooling

https://herbwood.tistory.com/8

end-to-end : Trainable

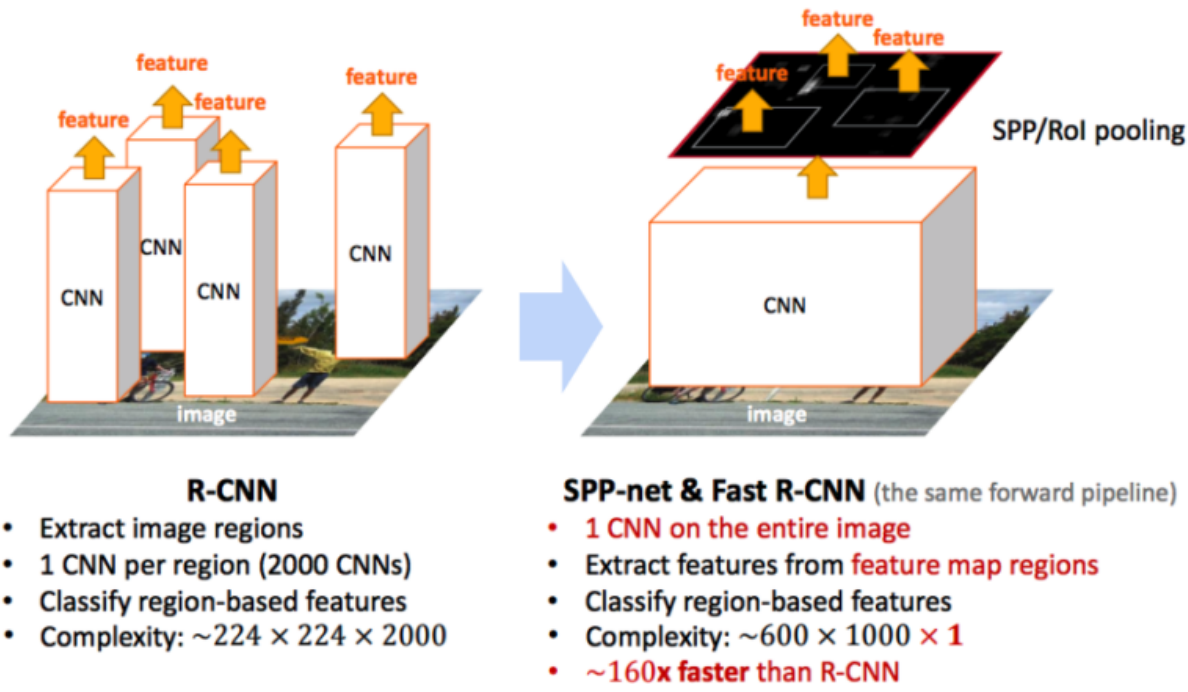
R-CNN에서 학습이 일어나는 부분

1. 이미지 넷으로 이미 학습된 모델을 가져와 fine tuning 하는 부분

2. SVM Classifier를 학습시키는 부분

3. Bounding Box Regression

R-CNN과 Fast R-CNN



> Faster R-CNN : https://ganghee-lee.tistory.com/37

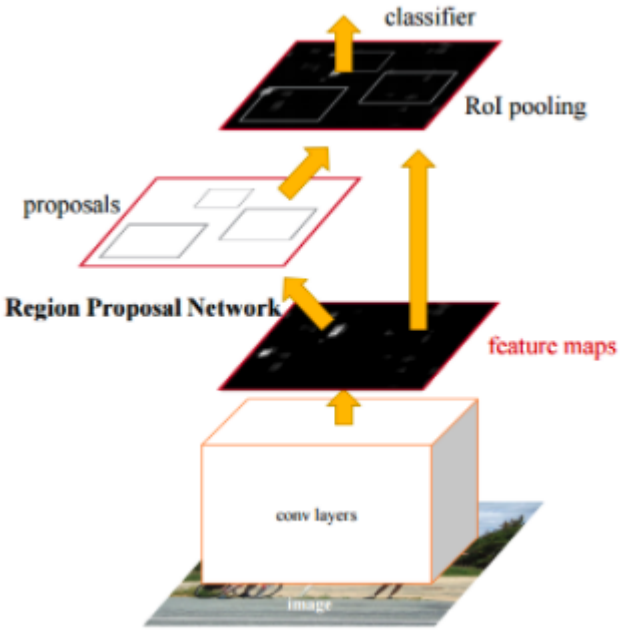
https://blog.lunit.io/2017/06/01/r-cnns-tutorial/

Fast R-CNN의 속도가 많이 개선됐지만, region proposal은 CPU를 사용하기 때문에 처리속도가 느리다. 이러한 region proposal 알고리즘을 개선하기 위해 Faster R-CNN이 등장했다.

→ 방법 : CNN 내부에 region proposal 생성하는 network를 설계했다!

가능한 이유

classification 학습 과정에서 학습되는 convolution filter들이 중요한 정보만 남기는 과정에서 위치 정보도 남기기 때문이다. 따라서 detection을 수행하는 CNN을 가진 feature map은 대략적인 위치 정보를 갖고 있다.



Region Proposal Network

위에서 설명한 지식을 사용해,

feature map이 갖고 있는 위치 정보를! 출력으로 가지는 네트워크인 region proposal network를 학습해보는 게 아이디어.

process

feature map 위의 N*N 크기의 window 영역을 입력받는다.

이 영역에 물체의 존재 유무에 대한 binary classification을 수행하는! classification network를 만든다. → sliding window 방식 사용

bounding-box regression이 위치 보정을 위해 사용됨

Fast R-CNN & RPN 학습 과정

1. M0 conv feature map을 기반으로 → RPN M1 학습
2. RPN M1을 사용하여 → 이미지에서 region proposal P1 추출
3. P1을 사용해 M0을 기반으로 Fast R-CNN 학습 → 모델 M2(==Fast R-CNN 모델) 얻음
4. M2의 conv feature를 고정시킨 상태에서 RPN을 학습 → M3 얻음 (== RPN 모델)
5. M3를 사용해 이미지에서 region proposal P2 추출
6. M3의 conv feature를 고정시키고 → M4 (==Fast R-CNN 모델) 학습

   ⇒ 왜?

backbone의 결과로 나온 feature map(1개, 최종 layer에서의 output) → RoI 생성 → classification, b-box regression 진행

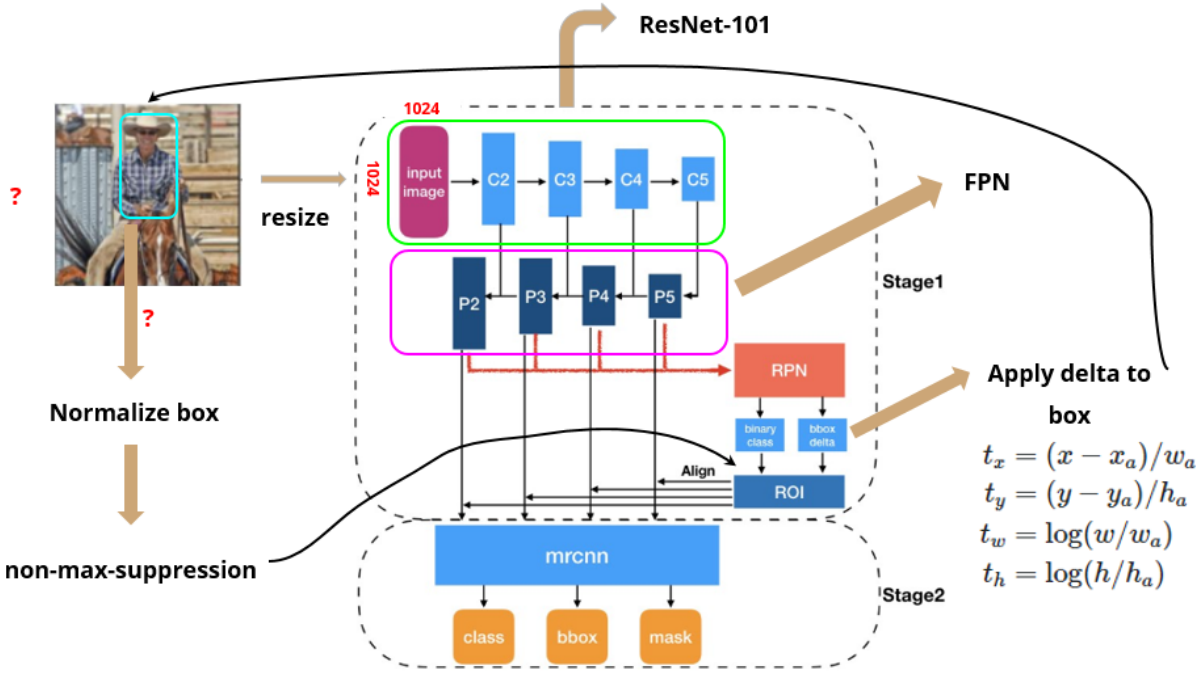
⇒ 문제점 : 중요한 feature 외의 feature는 잃어버리고, 최종 layer에서 다양한 크기의 object를 검출해야 하기 때문에 **여러 scale 값으로 anchor를 생성하므로 비효율적이다. → ???????????**

> Mask R-CNN : https://ganghee-lee.tistory.com/40

Image segmentation을 수행하기 위해 고안됨

달라진 점

1. Fast R-CNN의 classification, localization(bounding box regression) branch에 mask branch 추가.
2. RPN 전에 FPN (feature pyramid network) 추가
3. Image segmentation의 masking을 위해 RoI pooling 대신 RoI align 추가



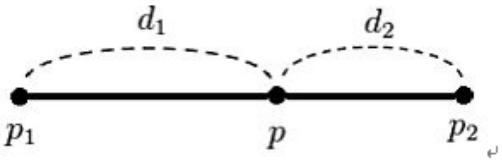
process

1. 이미지 resize (using bilinear interpolation)

   ▼ resize input image

   bilinear interpolation을 사용해 resize

   upsampling을 할 때 bilinear interpolation을 사용하고, 식은 아래와 같다.



일반적으로 두 지점 p1, p2 에서의 데이터 값이 각각 f(p1), f(p2)일 때, p1, p2 사이의 임의의 지점 p 에서의 데이터 값 f(p) 는 다음과 같이 계산할 수 있다.

$$f(p) = \frac{d_2}{d_1 + d_2} f(p_1) + \frac{d_1}{d_1 + d_2} f(p_2)$$

   나머지 값들은 zero padding

2. backbone network의 인풋으로 들어가기 위해 인풋 사이즈를 맞춘다 (using padding)
3. resNet-101을 통해 각 layer에서 feature map 생성

   ▼ ResNet

https://ganghee-lee.tistory.com/41

모델의 layer가 깊어질 수록 → 성능 하락

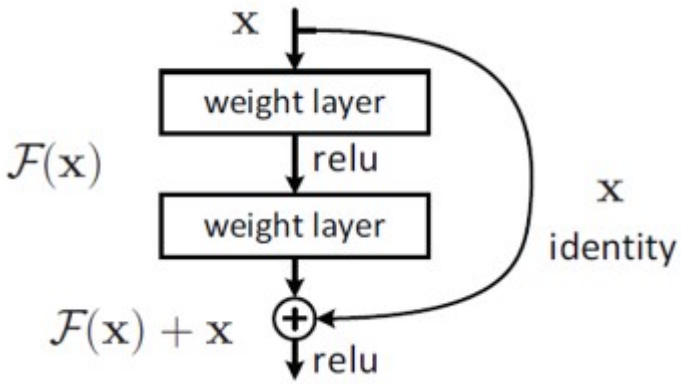
⇒ gradient vanishing/exploding 문제가 발생하기 때문!

> gradient vanishing(exploding) problem이란? 신경망의 weight가 계속 전파되면서 초기의 weight를 잊어버리거나 무한히 커지는 현상
>
> backpropagation을 해도 앞의 layer일수록 미분값이 작아져 그만큼 output에 영향을 끼치는 weight 정도가 작아지는 것
>
> == Degradation

⇒ 이 문제를 극복하기 위해 ResNet이 고안됨

- skip connection을 이용한 residual leraning

이미지 classification과 같은 문제에서는, input x에 대한 타겟값 y는 사실 x를 대변하는 것이기 때문에, y와 x의 의미가 같게 된다. 따라서 우리는 H(x)-x를 최소화 하는 방향으로 학습을 진행해야 한다. 이것(H(x)-x)을 잔차라고 한다. 이 잔차를 학습하는 것은 Residual learning이라고 한다.



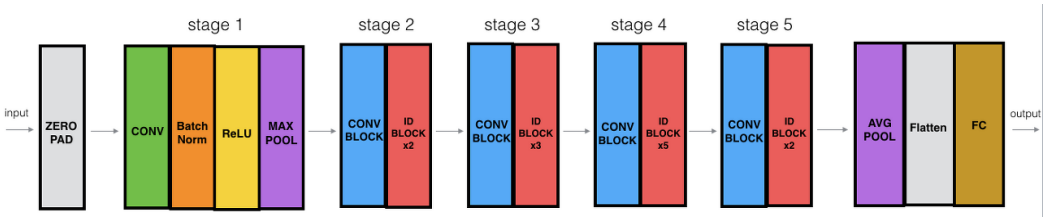
마지막 단계에 x를 더해주게 되면 미분하더라도 1을 갖기 때문에 최소 gradient로 1은 갖게 된다! → gradient vanishing 문제를 해결했다.

그럼 ResNet이 뭔데??

identity block은 네트워크의 output F(x)에 x를 더하는 것이고,

convolution block은 x에 1*1 convolution 연산을 거친 후 F(x)에 더해주는 것이다.

ResNet은 이 두 block을 쌓아서 만든 것이다.



ResNet-101의 경우, 각 stage마다 convolution block은 1개씩 존재한다.

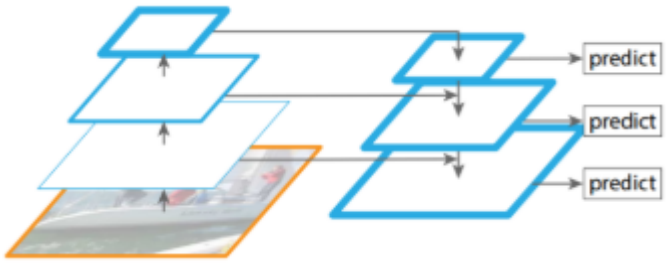
4. 3의 feature map에서 새로운 feature map 생성 (using FPN)

▼ FPN (Feature Pyramid Network)

anchor : window slide에 사용하는 window의 다양한 크기의 set

RPN에서는 layer를 통과하면 다양한 크기의 object가 검출되고, 여러 scale 값으로 anchor를 생성하기 때문에 비효율적이다. → 극복 방법으로 나왔다.

결론적으로, feature map들을 더하면서 이전 정보까지 유지할 수 있게 했고, 여러 anchor를 굳이 만들 필요가 없어진다.



작은 feature map에서 큰 anchor를 생성하여 detect 한다.

upsampling을 해서 이전 layer의 feature map을 fully convolution 연산 → filter 개수를 맞춰줌 → 새로운 feature map 형성

C1~5 feature map 생성

C5 → F2~5 생성

F5 → maxpooling → F6

결론적으로 F2~6가 생성됨. → 3*3 convolution → RPN

이때, F6은 F5에서 max pooling을 한 결과이기 때문에 convolution 연산 하지 않고 바로 RPN

5. 4의 feature map에 RPN 적용 → classification, b-box regression output 값 도출

   ▼ RPN

   pyramid feature map마다 scale 1개 * ratio 3개 = 3개의 anchor 생성

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

∴

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

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

   output : classification, b-box regression(=delta)

   delta 값에 anchor 정보 연산 → anchor bounding box 좌표값

6. b-box regression 값을 원래 이미지로 투영시켜 anchor box 생성
7. 생성된 anchor box 중 score가 가장 높은 걸 제외하고 모두 삭제 (using Non-max-suppression)

   ▼ Non-max-suppression

   이미지에 anchor 좌표를 대응시키면 → normalized coodinate로 대응시킨다 : FPN에서의 feature map의 크기가 다르기 때문에 크기가 통일되게 정규 좌표계로 이동 시키는 것

   anchor box 수천 개 생성

   NMS 알고리즘(score가 높은 b-box부터 정렬시킨다. IoU(겹치는 부분)를 계산해서 b-box와 0.7이 넘어가면 두 b-box는 동일 object를 가진 것이라고 판단하고 score 낮은 쪽을 제거해간다.) 사용하여 anchor 개수를 줄인다

8. anchor box의 size를 맞춘다 (using RoI align)

   ▼ RoI align

기존의 R-CNN에서 RoI fooling은 object detection을 위한 모델이었기 때문에 정확한 위치 정보가 중요하지 않았다. → input image의 위치 정보가 왜곡되기 때문에 segmentation에서 문제가 된다.

⇒ 그래서! bilinear interpolation을 이용해서! 위치정보를 담는! RoI align을 이용한다.

9. mask branch에 anchor box 값을 통과시킴

> CRAFT(Naver) : https://arxiv.org/abs/1904.01941