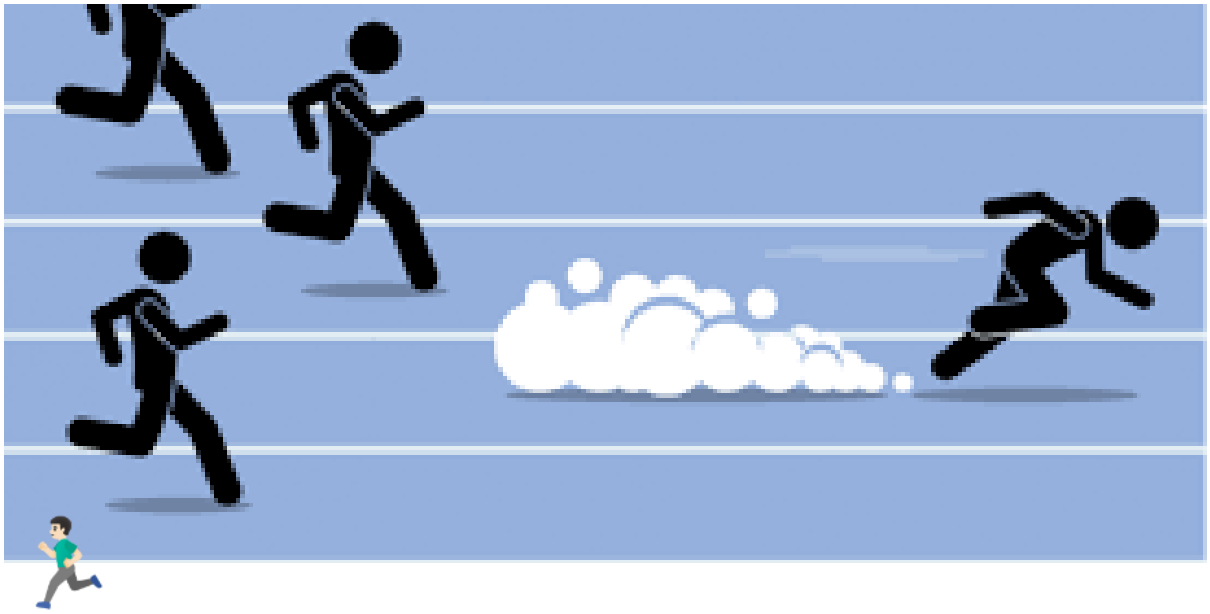
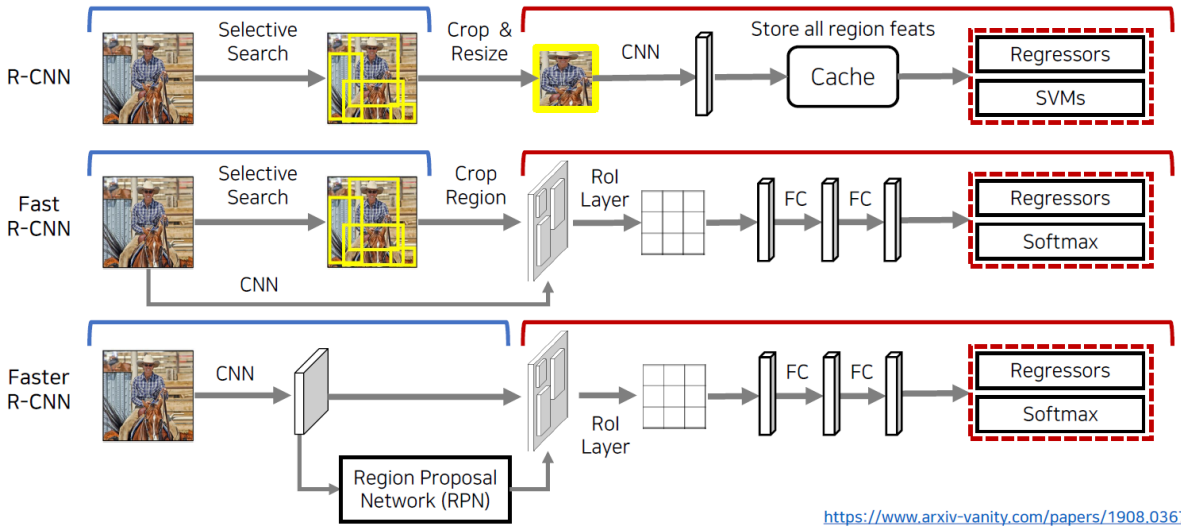


Faster R-CNN Review

R-CNN / Fast R-CNN / Faster R-CNN 비교



https://www.arxiv-vanity.com/papers/1908.03673/

발전 방향

R-CNN (CVPR 2014)	장점	CNN을 이용해 각 Region의 클래스를 분류할 수 있습니다.
	단점	전체 프레임워크를 End-to-End 방식으로 학습할 수 없습니다. 따라서 Global Optimal Solution을 찾기 어렵습니다.
Fast R-CNN (ICCV 2015)	장점	Feature Extraction, RoI Pooling, Region Classification, Bounding Box Regression 단계(step)를 모두 End-to-End로 묶어서 학습할 수 있습니다.
	단점	여전히 첫 번째 Selective Search는 CPU에서 수행되므로 속도가 느립니다.
Faster R-CNN (NIPS 2015)	장점	RPN을 제안하여, 전체 프레임워크를 End-to-End로 학습할 수 있습니다.
	단점	여전히 많은 컴포넌트로 구성되며, Region Classification 단계에서 각 특징 벡터(feature vector)는 개별적으로 FC layer로 Forward 됩니다.

- **R-CNN**

  1) 이미지에 대하여 CPU상에서 **Selective search**를 진행하여 물체가 존재할 법한 2000개의 위치를 찾음 (**Region proposal**)

  2) 2000개의 각 물체를 개별적으로 CNN 네트워크에 넣어서 **Feature vector**를 추출

  3) Feature vector들에 대하여 SVM을 이용하여 Classification 진행하고, Regressor를 이용하여 정확한 물체의 위치가 어딘지 Bounding Box를 조절하여 예측할 수 있게함

- **Fast R-CNN**

  1) R-CNN과 마찬가지로 **Region proposal**을 찾음

  2) 다만, R-CNN과 다르게 Feature map을 추출하기 위해 CNN을 **한번만 거침**

  3) **RoI Pooling**을 통해 각각의 Region들에 대해서 Feature에 대한 정보를 추출

  cf) CNN 구조를 보면, Feature map은 input image에 대해서 각각의 위치에 대한 정보를 어느정도 보존하고 있기 때문에 가능

  4) 기본적인 CNN 네트워크를 이용해서 Softmax layer를 거쳐서 각각에 클래스에 대한 probability를 구하게 됨

- **Faster R-CNN**
  - R-CNN과 Fast R-CNN은 모두 **CPU**에서 **Region proposal**을 진행하여 속도가 매우 느림
  - Region proposal을 위한 모든 연산을 **GPU** 상에서 수행할 수 있도록 하는 **RPN 네트워크** 제안
  - 이후, Fast R-CNN에서 사용하는 Detection 네트워크의 아키텍처를 사용해서 실제로 각각의 물체가 존재할 법한 위치에 대하여 Classification과 Regressor 진행

- 기존의 Fast R-CNN에서 Region proposal을 진행하는 Selective search를 RPN 네트워크로

  바꾸어서 모든 과정 자체를 **end-to-end** 방식으로 학습될 수 있도록 아키텍처를 바꾼 구조
- 한마디로 RPN + Fast R-CNN
- Fast R-CNN 구조에서 conv feature map과 RoI Pooling사이에 RoI를 생성하는

  Region Proposal Network가 추가된 구조

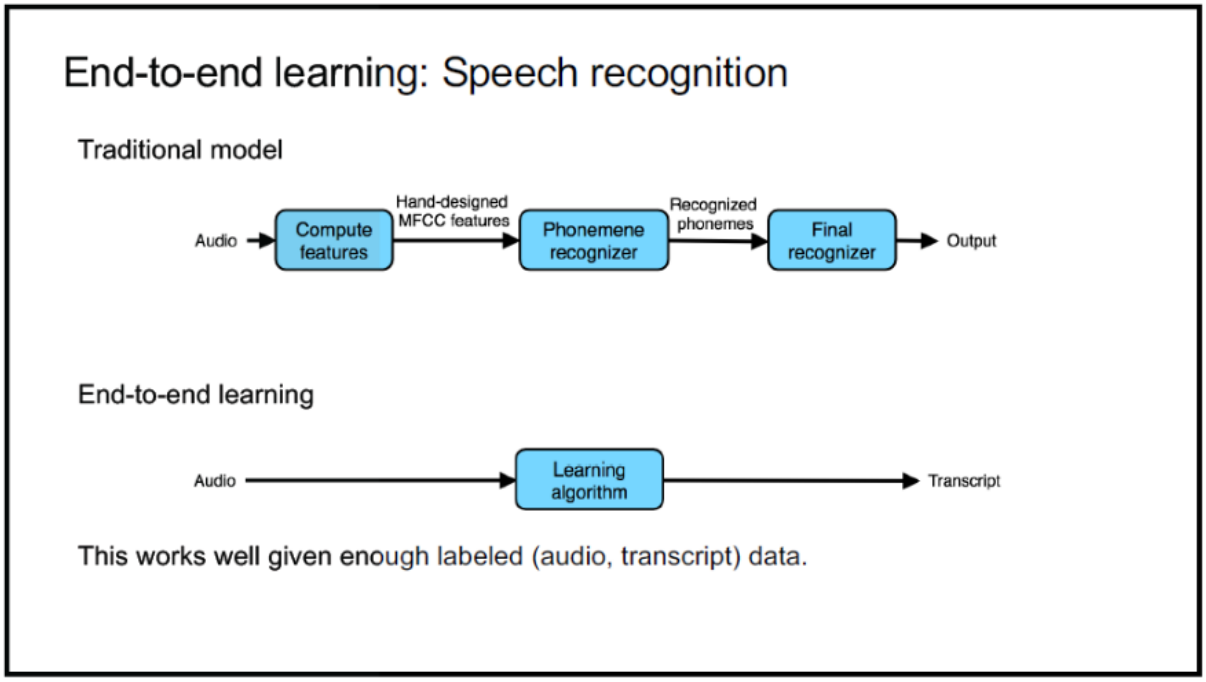
- **RoI Pooling** 을 하나 추가함으로써

  (1) CNN후에 region proposal 연산 - 2000xCNN연산 → **1번의 CNN연산**

  (2) 변경된 feature vector가 결국 기존의 region proposal을 projection시킨 후 연산한 것이므로 해당 output으로 classification과 bbox regression도 학습 가능
- 그러나 여전히 Fast R-CNN에서도 R-CNN에서와 마찬가지로 RoI를 생성하는 Selective search 알고리즘은 CNN외부에서 진행되므로 이 부분이 속도의 **bottleneck**
- 따라서 이 RoI 생성마저 CNN내부에서 함으로써 더욱 빠르면서 정확한 region proposal을 생성한 Faster R-CNN 등장

End-to-end

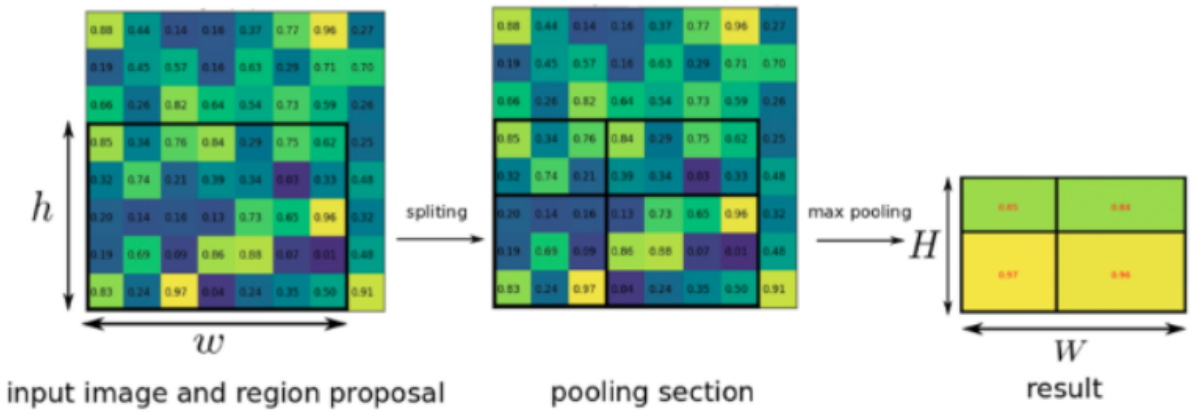
- 입력에서 출력까지 파이프라인 네트워크 없이 신경망으로 한 번에 처리한다는 의미



- 예를 들어, 기존의 Speech recognition system은 MFCC로 음성 파일의 특징 추출 → ML 알고리즘으로 음소를 알아냄 → 음소들로 단어를 만듦 → words 출력 같은 복잡한 과정을 거쳤음
- end-to-end learning은 음성 파일에서 바로 출력을 구할 수 있음

RoI Pooling layer

- RoI 영역에 해당하는 부분만 **max-pooling**을 통해 feature map으로부터 고정된 길이의 저차원 벡터로 축소하는 단계



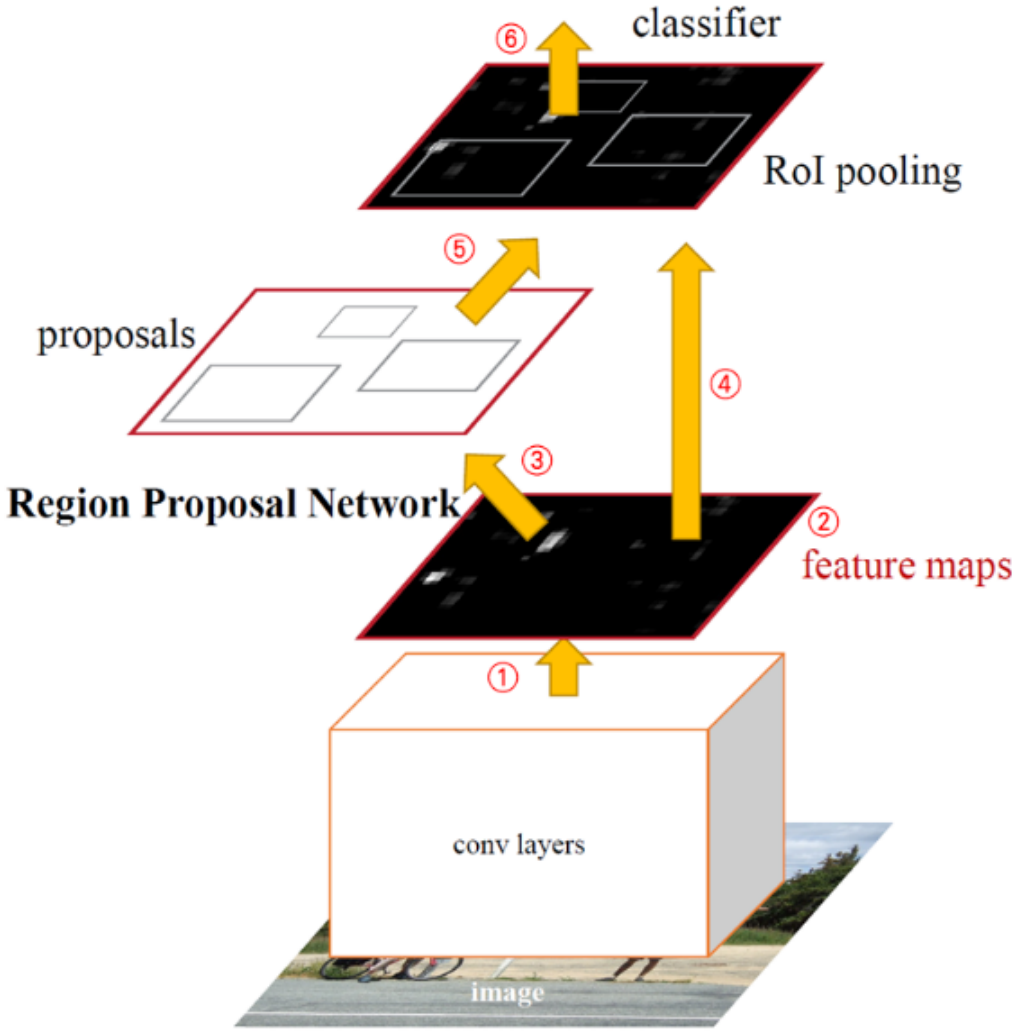
(1) 미리 설정한 HxW크기로 만들어주기 위해서 (h/H) * (w/W) 크기만큼 grid를 RoI위에 만듬

(2) RoI를 grid크기로 split시킨 뒤 max pooling을 적용시켜 결국 각 grid 칸마다 하나의 값을 추출

- feature map에 투영했던 hxw크기의 RoI는 HxW크기의 고정된 feature vector로 변환됨
- 원래 이미지를 CNN에 통과시킨 후 나온 feature map에 이전에 생성한 RoI를 projection시키고 이 RoI를 FC layer input 크기에 맞게 고정된 크기로 변형할 수 있음
- 더이상 2000번의 CNN연산이 필요하지 않고 **1번의 CNN연산**으로 속도를 대폭 높일 수 있음

Faster R-CNN

- 영역 추정을 위한 깊은 합성곱 네트워크(RPN) + 객체 탐지 모듈
- Attention 메커니즘과 유사하게 RPN은 탐지기(detector)가 어디에 주목할지 알려줌



1) ① 합성곱 계층을 활용해 전체 이미지에 합성곱 연산을 함

2) ② 그 결과 **Feature map**을 구하고, Feature map은 ③ RPN과 ④ 분류기(classifier)에 전달됨

- 분류기는 Fast R-CNN에서 사용한 분류기와 같음
- 피처 맵이 RPN과 분류기에 **동시 전달**된다 = RPN과 분류기가 피처 맵을 공유해서 사용한다

3) ③ RPN은 Feature map을 기반으로 객체가 있을 만한 곳을 찾아주는 **영역 추정**을 함

4) ⑤ 영역 추정 결과를 **RoI Pooling** 함

5) ⑥ 피처 맵과 영역 추정 경계 박스를 활용해 객체 탐지 수행

- 여기서 RPN은 객체가 어디에 있을지 즉, 탐지기가 어디에 주목을 해야 하는지 알려줌.
- Attention 메커니즘과 유사

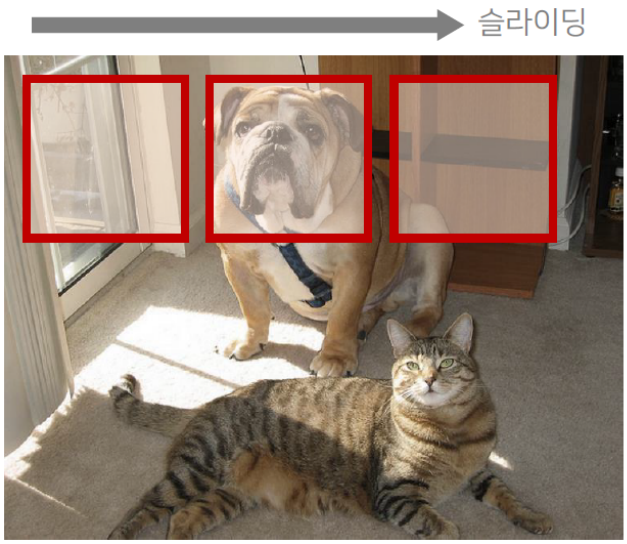
Faster R-CNN은 **영역 추정**과 **이미지 분류 모두**를 하나의 통합 네트워크에서 수행

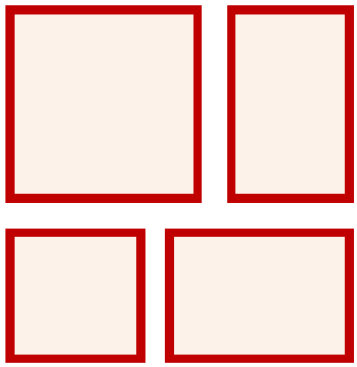
영역 추정을 독립된 모듈 (CPU)로 수행하는 Fast R-CNN과 다름

Region Proposal

① Sliding Window

- 이미지에서 다양한 형태의 윈도우(window)를 슬라이딩(sliding)하며 물체가 존재하는지 확인
- 너무 많은 영역에 대하여 확인해야 한다는 단점 존재
- 특히나 Feature map이 아닌 input image에 대해서 CPU 장치를 이용해 sliding window를 진행하게 되면, 넓은 input space 상에서 많은 영역에 대하여 확인해야하므로 느림





다양한 형태의 윈도우(window)

② Selective Search

- **인접한 영역(region)**끼리 **유사성**을 측정해 큰 영역으로 차례대로 통합해 나감
- R-CNN / Fast R-CNN에서 사용하는 **Region Proposal** 방식
- CPU 기반에서 1장의 이미지에 대해 2초 가량 시간이 소요될 수 있음

Algorithm 1: Hierarchical Grouping Algorithm

Input: (colour) image
Output: Set of object location hypotheses L

Obtain initial regions $R = \{r_1, \cdots, r_n\}$ using [13]
Initialise similarity set $S = \emptyset$
foreach *Neighbouring region pair* (r_i, r_j) **do**
  Calculate similarity $s(r_i, r_j)$
  $S = S \cup s(r_i, r_j)$

while $S \neq \emptyset$ **do**
  Get highest similarity $s(r_i, r_j) = \max(S)$
  Merge corresponding regions $r_t = r_i \cup r_j$
  Remove similarities regarding r_i : $S = S \setminus s(r_i, r_*)$
  Remove similarities regarding r_j : $S = S \setminus s(r_*, r_j)$
  Calculate similarity set S_t between r_t and its neighbours
  $S = S \cup S_t$
  $R = R \cup r_t$

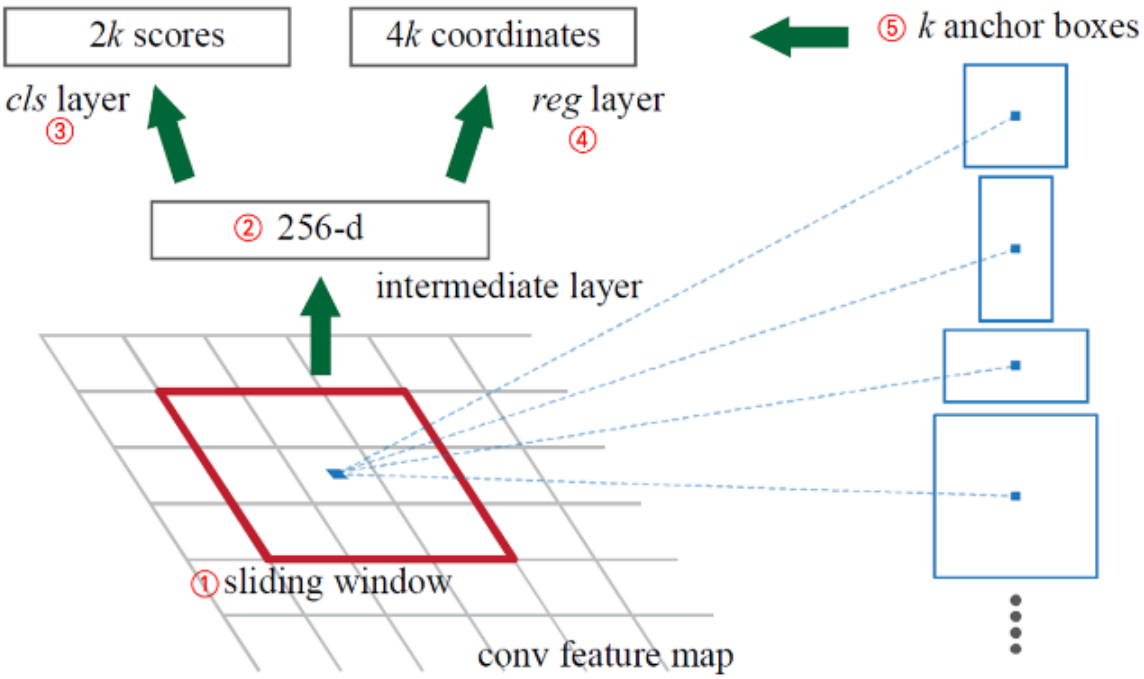
Extract object location boxes L from all regions in R

Selective Search 알고리즘

(처리 과정)

Region Proposal Networks

- **크기에 상관없이 이미지 전체를 입력받아** 영역 추정 경계 박스를 반환함
- 각 경계 박스는 객체 존재 여부를 점수로 나타냄
- 영역 추정 경계 박스를 만들기 위해, Faster R-CNN에서는 **Sliding window 방식**을 적용



- RPN의 input 값은 이전 CNN 모델에서 뽑아낸 feature map
- Region proposal을 생성하기 위해 feature map위에 nxn window를 **sliding window** 함
  - 각 슬라이딩 윈도우는 작은 차원의 피처로 매핑됨 (ZF에서는 256차원, VGG에서는 512차원)
- 이때 object의 크기와 비율이 어떻게 될지 알 수 없으므로, **k개의 anchor box**를 미리 정의
- anchor boxs는 bounding box가 될 수 있는 것들이므로, 가능할만한 box모양 k개 정의해놓는 것
- 가로세로길이 3종류 x 비율 3종류 = 9개의 anchor box를 이용한다.
- 9개의 anchor box를 이용하여 classification과 box regression을 구함

1) CNN에서 뽑아낸 feature map에 대해 3x3 conv filter 256개를 연산하여 **depth를 256**으로 만듬

2) 그 후 **1x1 convolution** 2개를 이용하여 각각 classification과 bounding box regression을 계산함

3) 네트워크를 가볍게 만들기 위해 binary classification으로 **bounding box에 물체가 있나 없나 판단**

- RPN단계에서 classification과 bounding box regression을 하는 이유는 결국 학습을 위함
- 위 단계로부터 positive / negative examples들을 뽑아내는데 다음 기준에 따름

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

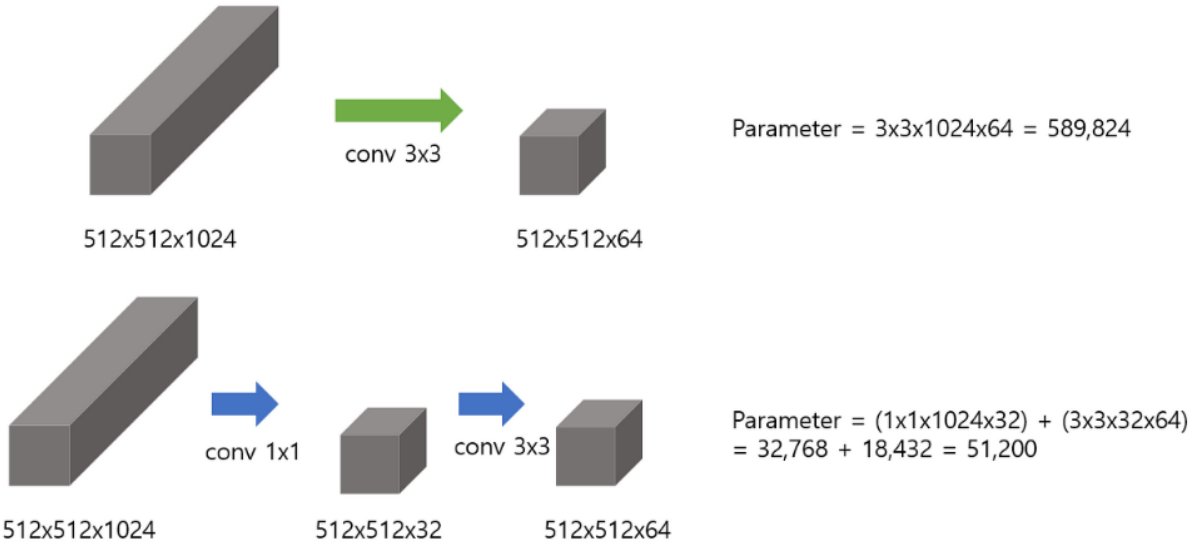
  - IoU가 0.7보다 크거나, 한 지점에서 모든 anchor box중

    가장 IoU가 큰 anchor box는 **positive example**로 만듬
  - IoU가 0.3보다 작으면 object가 아닌 background를 뜻하므로

    negative example로 만듬
  - 사이에 있는 IoU에 대해서는 애매한 값이므로 학습 데이터로 이용하지 않음

1x1 Convolution layer

- **Channel 수 조절, 연산량 감소**
  - 1x1 conv layer를 사용하면 output size는 변함이 없지만 channel(filter)의 수 가능
  - channel 수를 적절히 조절하면 파라미터를 효과적으로 줄일 수 있음
  - 파라미터가 감소됨에 따라 같은 결과라도 1x1 conv layer를 사용하는 편이 연산량 감소 효과
  - 파라미터 수 = kernel size * kernel size * input channel * output channel



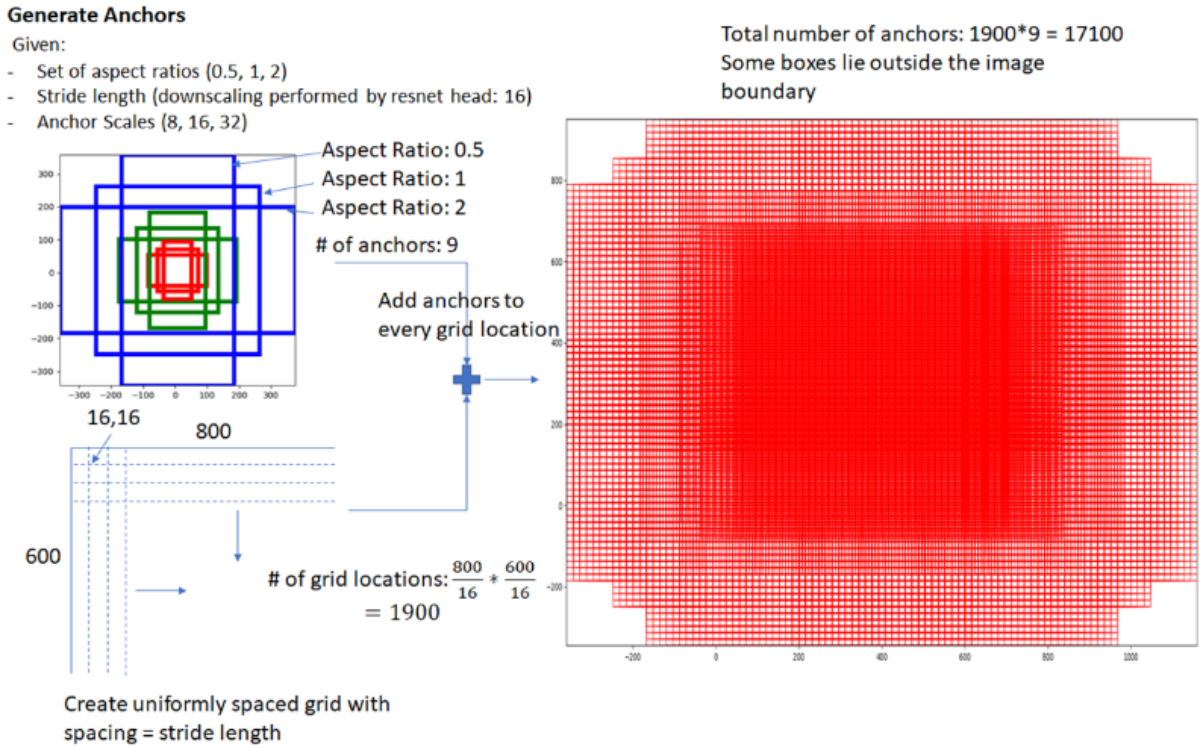
- **비선형성**
  - 1x1 conv layer를 사용함에 따라 그만큼 Activation function을 더 사용할 수 있게되는데, 이 때문에 모델의 비선형성이 좋아질 수 있음

Anchors

- 각 sliding window의 **중심 위치**마다 여러 bounding box region을 예측함
- 각 sliding window 위치마다 최대로 예측할 수 있는 bounding box region 개수는 k개

  → 그렇기 때문에 **regressor layor**는 좌표값을 4k개 갖음.

  bounding box 하나에는 총 4가지 좌표값이 있으므로, k개 경계 박스에는 좌표값이 4k개
- **classification layer**는 bounding box 하나에 객체일 확률과 객체가 아닐 확률로 두 가지 확률값이 있기 때문에 2k개 점수값을 갖음



- **feature map**에 촘촘하게 점(그리드)을 찍음 (실제로 점을 찍는다는 말이 아니라 위치만 표시)
- 모든 점(그리드)마다 앵커 박스 9개를 구하면서 앵커 박스가 서로 굉장히 많이 겹치게 되고,
  이렇게 구한 많은 앵커 박스를 바탕으로 **Region Proposal** 추정을 위해 훈련함

1) (800, 600) 크기를 갖는 이미지에 합성곱 연산을 해서 (800/16, 600/16) 크기의 feature map을 만듦

2) feature map 위에 **1,900개(= 800/16 * 600/16)의 그리드**를 표시

3) 1,900개의 각 그리드마다 앵커 박스를 9개씩 그리면서 앵커 박스가 총 17,100개가 됨

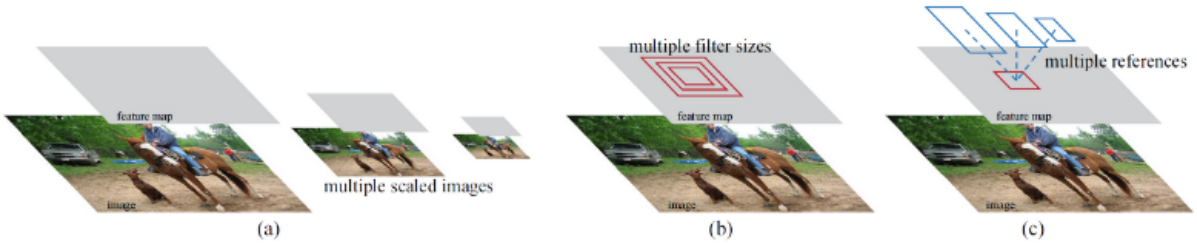
- 앵커 박스가 이렇게나 많고 다양하니 어떤 객체든 포함할 것
- RPN은 이런 앵커 박스를 바탕으로 훈련해서 **Region Proposal**을 추정

Translation-Invariant Anchors

- **위치 불변성** (Translation-Invariance)
- 이미지 안에서 객체 위치가 변하더라도 같은 객체로 인식하는 특성
  (**이미지 안에서 객체 위치가 변한다고 서로 다른 객체라고 인식하면 안됨!!!**)
- Faster R-CNN의 RPN은 **Sliding window** 방식으로 이미지의 전체 영역을 훑기 때문에,
  위치가 변해도 같은 객체로 잘 인식

Multi-Scale Anchors as Regression

- 멀티 스케일 예측에는 3가지 방법이 있음



(a) 이미지 피라미드 방식

- OverFeat, SPP-net에서 이미지 피라미드 방식을 사용
- 입력 이미지를 다양한 크기(멀티 스케일)로 조정해서 각 스케일마다 feature map을 구함
- 효율적인 방법이지만 시간이 오래 걸림

(b) 필터(sliding window)를 다양한 크기(멀티 스케일)로 사용하는 방법

- feature map은 하나지만 다양한 필터를 사용해 풀링하는 방식

- 필터 피라미드 방식

(c) 앵커 피라미드(pyramids of anchors) 방식

- 이미지 피라미드, 필터 피라미드보다 더 빠르고 효율적
- **다양한 크기의 앵커 박스**를 활용해 객체 분류, 경계 박스 회귀를 수행
- 앵커 피라미드 방식을 적용하면 **추가 연산 없이 피처를 공유할 수 있음**

Loss

- **multi-task loss**를 사용하여 feature vector를 Classifier와 Bounding box regressior을 동시에 학습
- 두 모델을 한번에 학습시키기 때문에, R-CNN 모델처럼 **각 모델을 독립적으로 학습시켜야 하는 번거로움이 없다는 장점**

분류 손실 회귀 손실

$$L(\{p_i\},\{t_i\}) = \frac{1}{\underset{⑦}{N_{cls}}} \sum_i \underset{⑤}{L_{cls}}(\underset{①}{p_i}, \underset{②}{p_i^*}) + \underset{⑨}{\lambda} \frac{1}{\underset{⑧}{N_{reg}}} \sum_i p_i^* \underset{⑥}{L_{reg}}(\underset{③}{t_i}, \underset{④}{t_i^*})$$

- i : mini-batch 내 anchor의 index
- pi : anchor에 객체가 포함되어 있을 확률
- pi* : anchor가 positive면 1, negative면 0 → negative: 객체가 아닌 배경
- ti : 예측된 bounding box의 좌표 벡터(4가지 좌표값)
- ti* : ground truth 좌표 벡터(실제 bounding box의 4가지 좌표값)
- Lcls : Classification loss → 객체 vs 객체x 에 대한 로그 손실
- Lreg : L1 Smooth loss → positive anchor box일 때만(객체일 때만) 활성화
  → negative일 때는 bounding box를 구할 필요가 없으므로
- Ncls : mini-batch의 크기 (논문에서는 256)
- Nreg : anchor 위치 개수
- λ : balancing parameter (default=10): 분류 손실과 회귀 손실 간 균형을 맞추기 위해서

Training RPNs

- 확률적 경사 하강법(SGD)과 역전파로 end-to-end 훈련을 할 수 있음

- 앞서 여러 앵커 박스를 만들어 영역 추정을 한다고 했는데,

  모든 앵커 박스마다 손실 함수를 적용해 최적화하면 **negative label에 치우친 결과가 나옴**

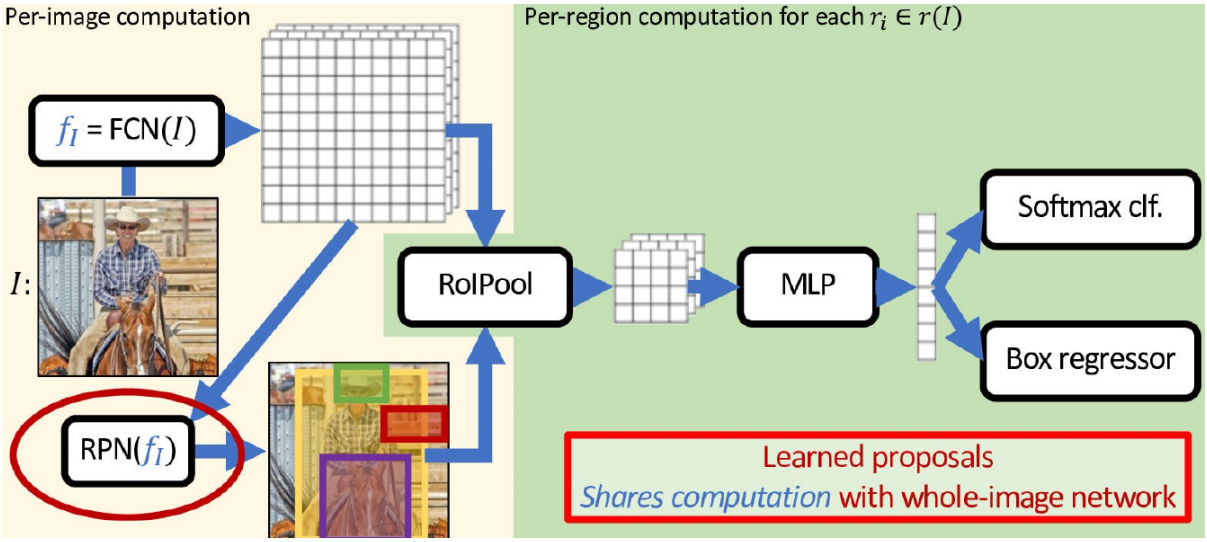
  - positive label보다 negative label이 압도적으로 많기 때문!
  - 웬만하면 이미지 안에서 객체가 차지하는 공간 비율보다 **배경이 차지하는 비율이 더 크니까!**
- 손실 함수를 계산하기 위해 하나의 이미지에서 앵커 박스 256개를 무작위로 샘플링
  - positive 앵커와 negative 앵커를 **1 (128개) : 1 (128개) 비율**로 뽑음
- 훈련할 때 새로운 계층은 '평균 0, 분산 0.01인 가우시안 분포에서 뽑은 가중치'로 초기화
- 나머지 모든 계층은 이미지넷 분류 영역에서 사전 훈련된 모델의 가중치로 초기화

Sharing Features for RPN and Fast R-CNN

- Faster R-CNN은 탐지기(detector)로 Fast R-CNN을 사용하고, RPN 적용
  - RPN과 Fast R-CNN이 **합성곱 피처를 공유**해 사용하기 때문에 효율적
- 그런데 RPN과 Fast R-CNN은 합성곱 피처를 서로 다른 방법으로 독립적으로 훈련
  - 그렇기 때문에 **RPN과 Fast R-CNN이 합성곱 피처를 공유하면서도 서로 다른 방법으로 분리**해서 사용할 수 있도록 해야 함
  - **(i) Alternating training**, (ii) Approximate joint training, (iii) Non-approximate joint training
  - (ii)와 (iii)은 사용하기에 어려워서 본 논문에서는 (i) 방식을 채택

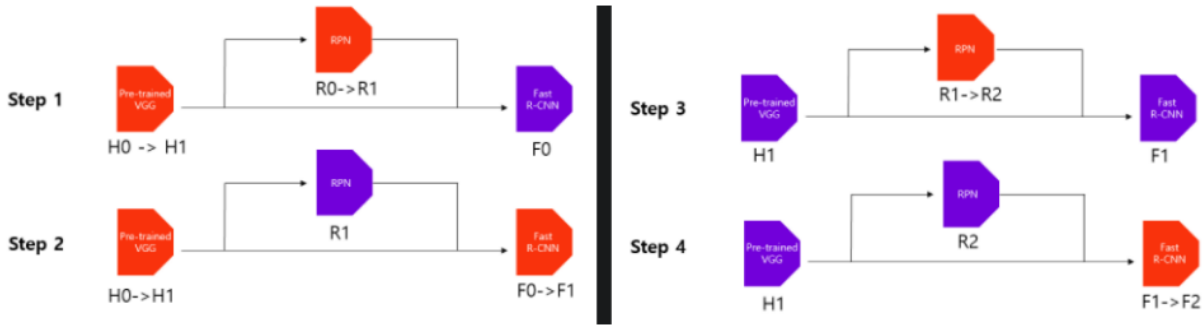
Alternating Training

- Alternating training: 번갈아가며 훈련한다



- 먼저 RPN을 훈련하고, 이어서 영역 추정 경계 박스(**Region estimation bounding box**)를 사용해 Fast R-CNN을 훈련
- Fast R-CNN으로 튜닝된 네트워크는 다시 RPN을 초기화하는 데 사용하고, 이런 절차를 반복
- 그러면 RPN과 Faster R-CNN이 **피처를 공유하면서도 독립적인 방법으로 훈련 가능**

4-Step Alternating Training



- RPN과 Fast R-CNN에서 공유되는 feature map을 학습시키기 위해 4단계로 학습 진행

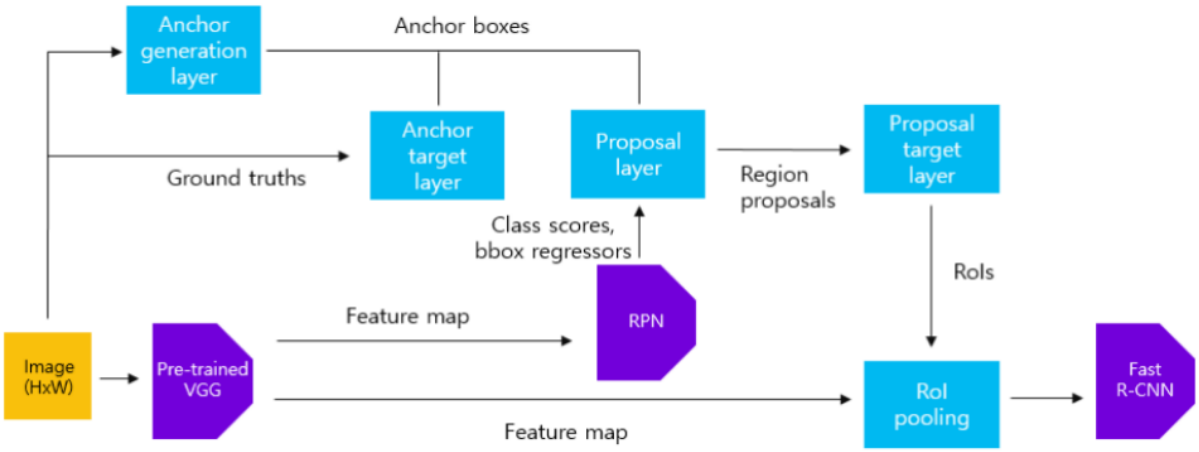
1. RPN training 방법을 통해 학습. 이때, pre-trained VGG도 같이 학습
2. (1)로 학습된 RPN을 이용해 RoI를 추출하여 Fast R-CNN 부분을 학습시킴.

   이때, pre-trained VGG도 같이 학습
3. (1)과 (2)로 학습된 pre-trained VGG를 통해 추출한 feature map을 이용해 RPN을 학습시킴.

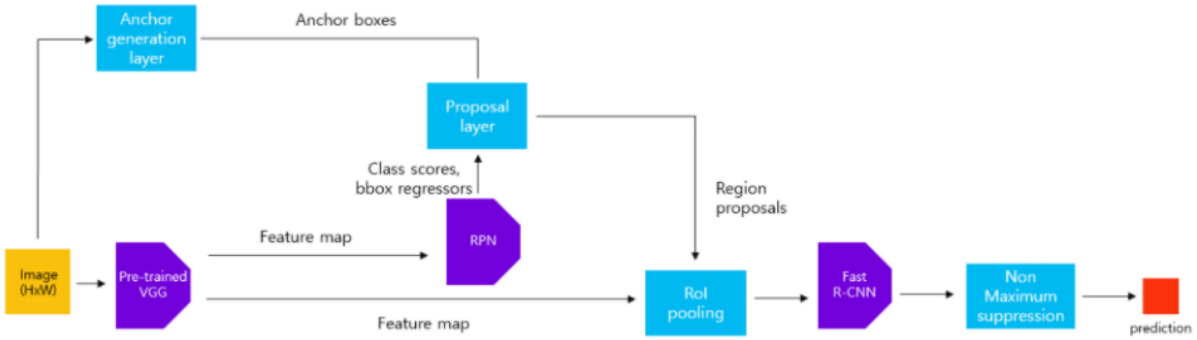
   이때, pre-trained VGG는 학습시키지 않음
4. (1),(2)를 통해 학습된 pre-trained VGG와 (3)을 통해 학습된 RPN을 이용해 Fast R-CNN을 학습

   이때, pre-trained VGG와 RPN은 학습시키지 않음

- Training



- Detection



- training은 detection과 다르게 'Anchor target layer'와 'Proposal target layer'가 존재하는데, 이는 학습을 위해 필요한 ground truth를 만들어 주는 layer들임