

SSD: Single Shot MultiBox Detector

Abstract

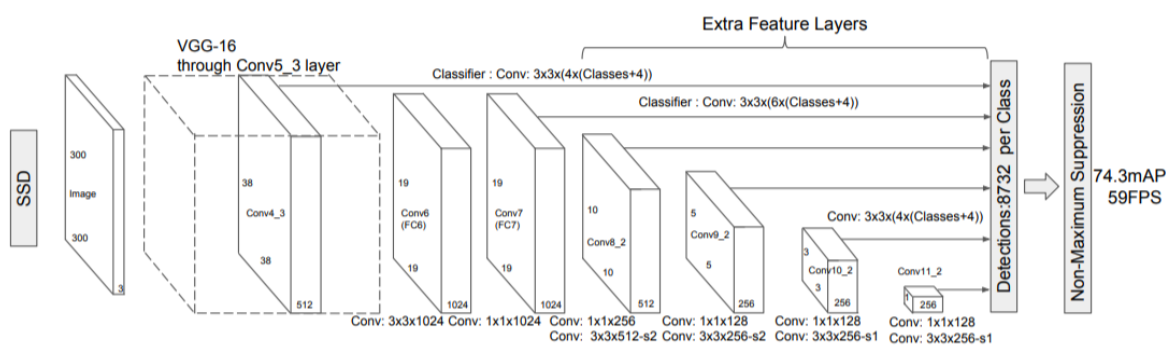
RCNN 계열의 2-stage detector는 region proposals와 같은 다양한 view를 모델에 제공하여 높은 정확도를 보여주고 있음. 하지만 region proposals를 추출하고 이를 처리하는 과정에서 많은 시간이 걸려 detection 속도가 느리다는 단점

반면 YOLO v1는 원본 이미지 전체를 통합된 네트워크로 처리하기 때문에 detection속도가 매우 빠름. 하지만 grid cell별로 2개의 bounding box만을 선택하여 상대적으로 적은 view를 모델에 제공하여 정확도가 떨어짐

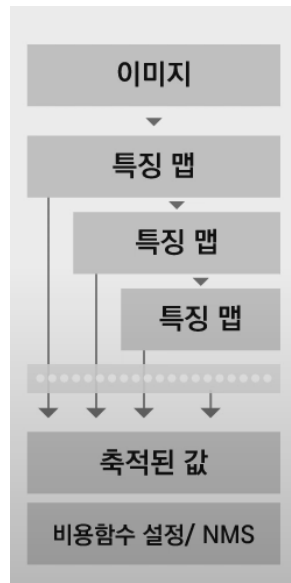
이처럼 일반적으로 정확도와 detection 속도는 trade-off관계에 있음

하지만 SSD는 다양한 view를 활용하면서 통합된 network구조를 가진 1-stage-detector로서 높은 정확도와 빠른 속도를 가짐

Model Preview



- VGG16을 base network로 사용하고 보조 network(auxiliary network)를 추가한 구조
- fc layer를 conv layer로 대체하면서 detection속도가 향상
- 총 6개의 서로 다른 scale의 feature map을 예측에 사용
- default box를 사용하여 객체의 위치를 추정함



Main ideas

1. Multi-scale feature maps

SSD모델의 YOLO v1모델과 마찬가지로 **하나의 통합된 네트워크로 detection**을 수행하는 **1-stage detector**

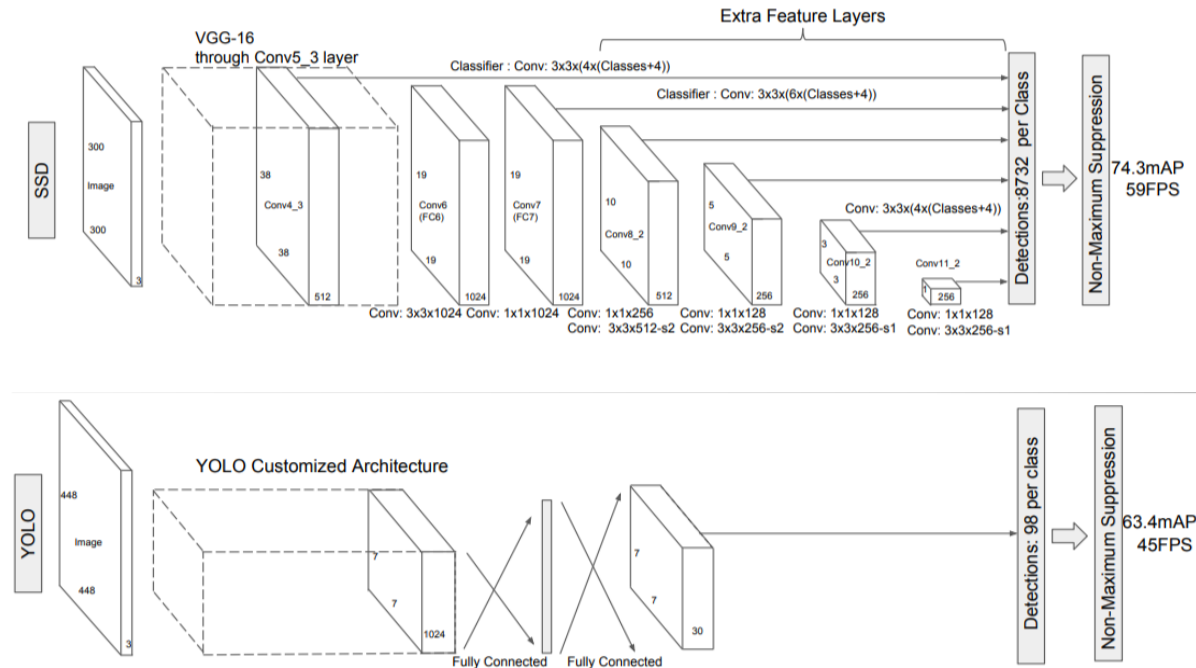
전체 network는 **pre-trained된 VGG16을 base network**로 사용하고, 이후 보조 **network(auxiliary network)**를 추가한 구조

보조 network는 일반적인 **conv layer**로 구성

base network 후반부에 등장하는 **fc layer**를 **conv layer**로 바꿔줘 보조 network와 **연결**함

→ 이 과정에서 **fc layer가 제거되면서 detection 속도가 향상되는 이점**

전체 network는 일반 convolutional network와 크게 다르지 않음

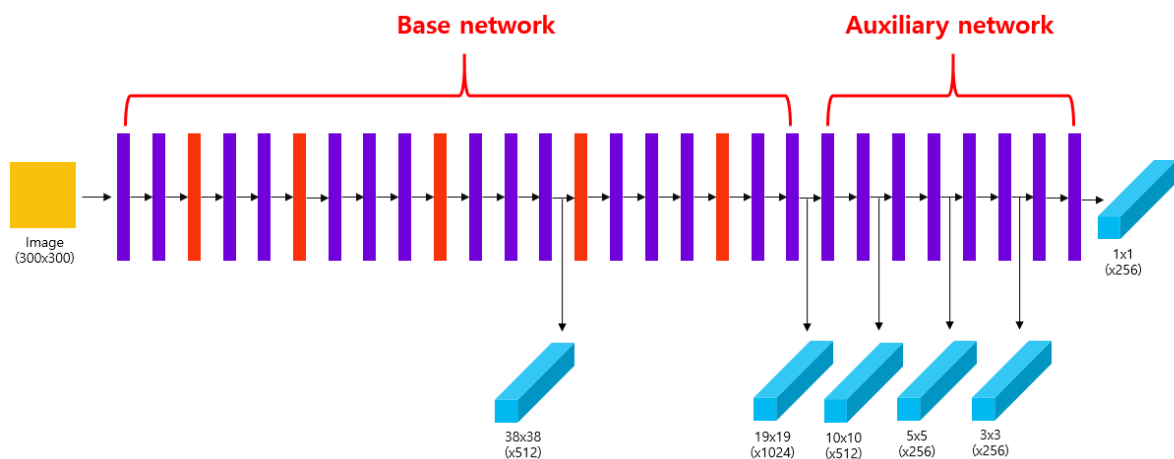


SSD 모델의 핵심적인 아이디어는 다양한 scale의 feature map

기존의 모델은 convolutional network를 거친 단일한 scale을 가진 feature map을 detection을 위해 사용 - YOLO v1모델의 경우 7x7 크기의 feature map만 사용

하지만 이처럼 단일한 scale의 feature map을 사용할 경우, 다양한 크기의 객체를 포착하는 것이 어렵다는 단점

→ 이러한 문제를 해결하기 위해 SSD 중간에 conv layer의 feature map들을 추출하여 detection시 사용하는 방법을 제안



먼저 base network conv4_1 layer에서 **38x38x(512) 크기의 feature map**을 추출

(→ Convolutional Network 중간의 conv layer에서 얻은 feature map)

그 다음 base network의 conv7 layer에서 **19x19x(1024) 크기의 feature map**을 추출

다음으로 auxiliary(보조) network의 conv8_1, conv9_2, conv10_2, conv11_2 layer에서 각각 **10x10x(512), 5x5x(256), 3x3x(256), 1x1x(256) 크기의 feature map**을 추출

이러한 과정을 통해 **총 6개의 scale**을 가진 feature map을 얻을 수 있음

→ **다양한 scale의 feature map**을 사용하여 보다 **다양한 크기의 객체를 탐지하는 것이 가능함**

2. Default boxes

원본 이미지에서 보다 **다양한 크기의 객체를 탐지**하기 위해 feature map의 각 cell마다 **서로 다른 scale과 aspect ratio**를 가진 **Default box**를 생성

Default box는 Faster R-CNN모델에서 사용하는 **anchor box**와 개념적으로 유사하지만 서로 다른 크기의 feature map에 적용한다는 점에서 차이가 있음

SSD 모델은 **38x38, 19x19, 10x10, 5x5, 3x3, 1x1** 총 6개의 scale의 feature map의 각 cell마다 **default box**를 생성

Default box의 scale = s_k

여기서 s_k 는 **원본 이미지에 대한 비율**

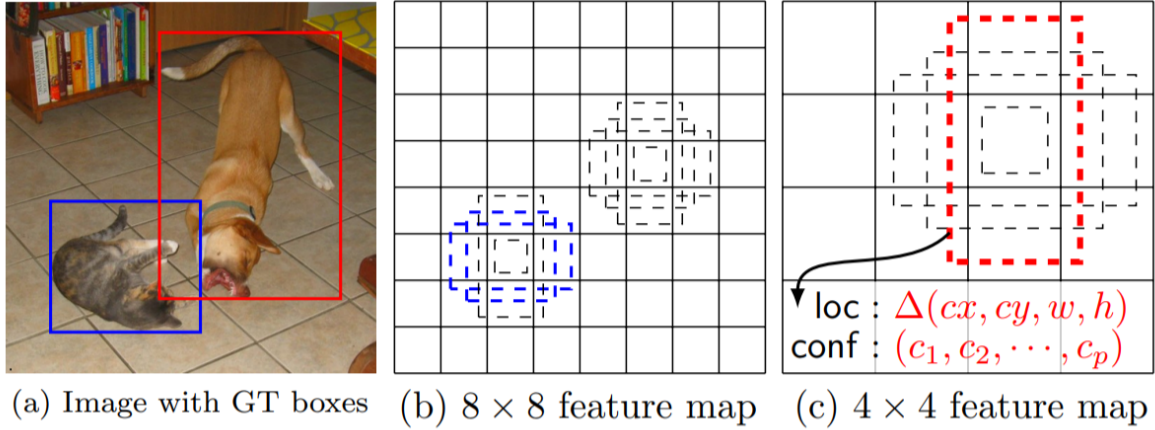
ex) 원본 이미지의 크기가 300x300이며, $s=0.1$ 이며 **aspect ratio가 1:1**일 때, default box의 크기는 **30x30**(=300x0.1 x 300x0.1)

각 feature map별로 적용할 default box의 scale을 구하는 공식은 다음과 같음

$$s_k = s_{min} + \frac{s_{max} - s_{min}}{m - 1} (k - 1), k \in [1, m]$$

- $s_{min} = 0.2$
- $s_{max} = 0.9$
- m : 예측에 사용할 feature map의 수, SSD 모델일 경우 $m = 6$

k 는 feature map의 순서(1번째 feature map → $k=1$)



첫 번째 feature map(크기가 38×38)의 $s_k = 0.2$ 이며, 두 번째 feature map은 $k = 2$ 이기 때문에 $s_k = 0.34$

마지막 feature map(크기가 1×1)의 경우 $s_k = 0.9$

즉 feature map의 scale이 작아질수록 default box의 scale은 커짐

이는 위에 그림에서 볼 수 있듯이 feature map의 크기가 작아질수록 더 큰 객체를 탐지할 수 있음을 의미합니다.

3. Predictions

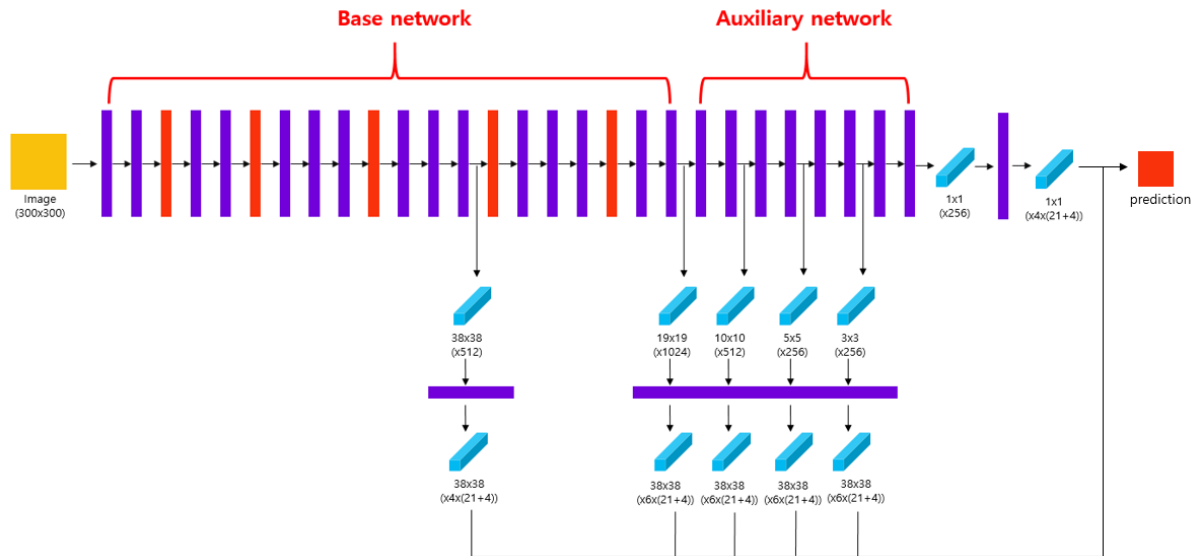
각각의 feature map은 서로 다른 수의 default box를 적용

첫 번째(38×38)와 마지막(1×1) feature map은 aspect ratio가 1:1, 1:2, 1:1/2인 box와 aspect ratio가 1:1일 때 추가적으로 사용하는 box까지 총 4개의 default box를 적용

이는 feature map의 각 cell마다 4개의 default box가 생성됨을 의미

나머지 4개의 feature map은 위에서 언급한 6개의 default box를 모두 적용

총 서로 다른 scale의 feature map에 맞는 default box를 적용한 경우 **총 default box의 수는 8732(=38x38x4 + 19x19x6 + 10x10x6 + 5x5x6 + 3x3x6 + 1x1x4)개**



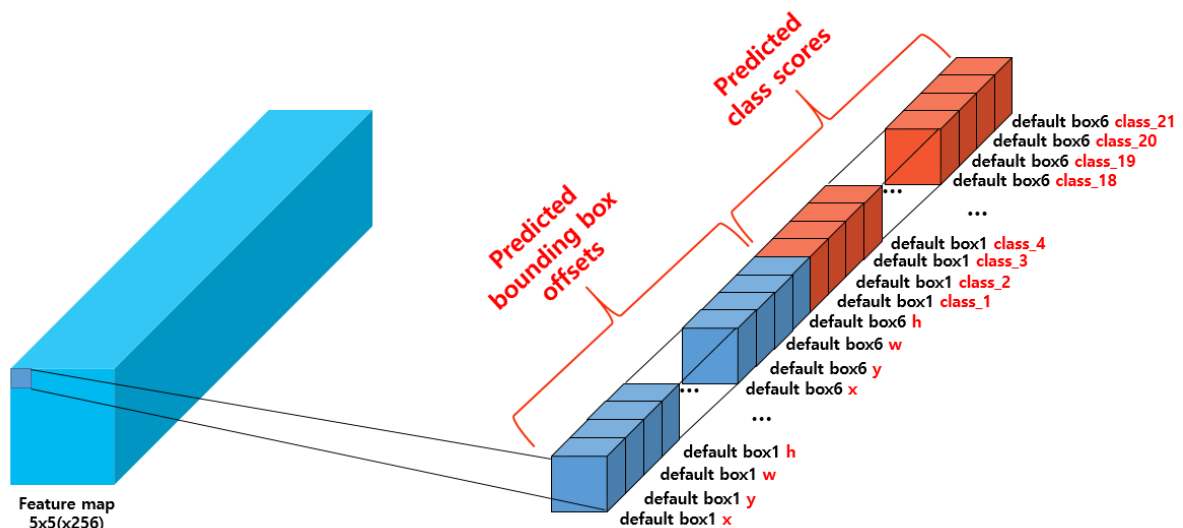
최종 예측을 위해 서로 다른 scale의 feature map을 추출한 후 3x3(stride=1, padding=1) conv 연산을 적용

이 때 default box의 수를 k , 예측하려는 class의 수를 c 라고 할 때,

output feature map의 channel 수는 $k(4+c)$ 가 되도록 설계

→ 4는 bbox의 $x, y, w, h == \text{offset}$

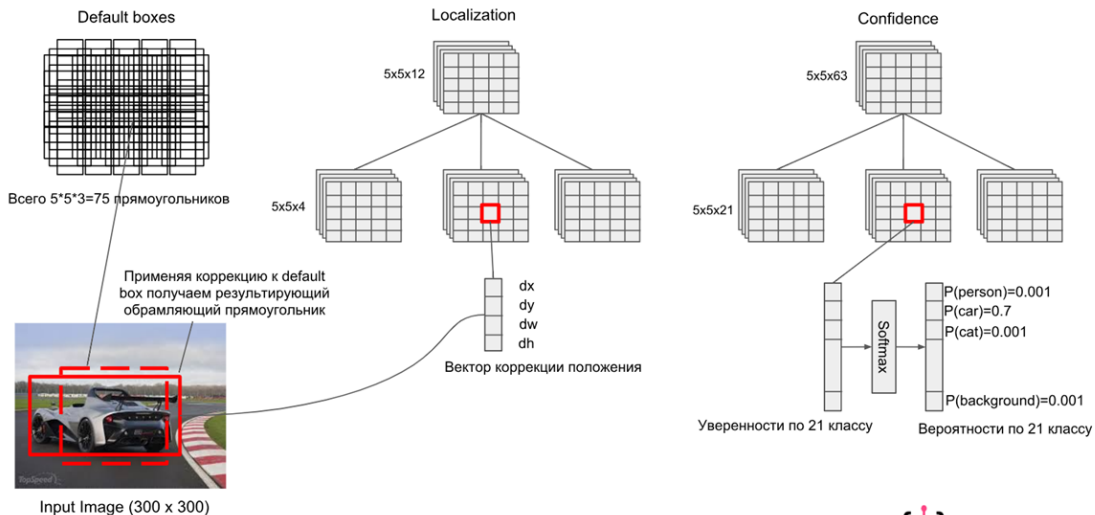
이는 각 feature map의 cell이 k 개의 default box를 생성하고 각 box마다 4개의 offset과 class score를 예측한다는 것을 의미



##SSD 모델은 예측하려는 class의 수가 20개인 PASCAL VOC 데이터셋을 사용해 학습을 진행

따라서 class의 수는 배경(no object)을 포함(+1)하여 $c = 21$ 입니다. 예를 들어 $5 \times 5 \times (256)$ 크기의 feature map을 추출할 경우 $k = 6$, $c = 21$ 이므로 conv 연산을 적용한 output feature map의 크기는 $5 \times 5 \times 6 \times (4+21)$

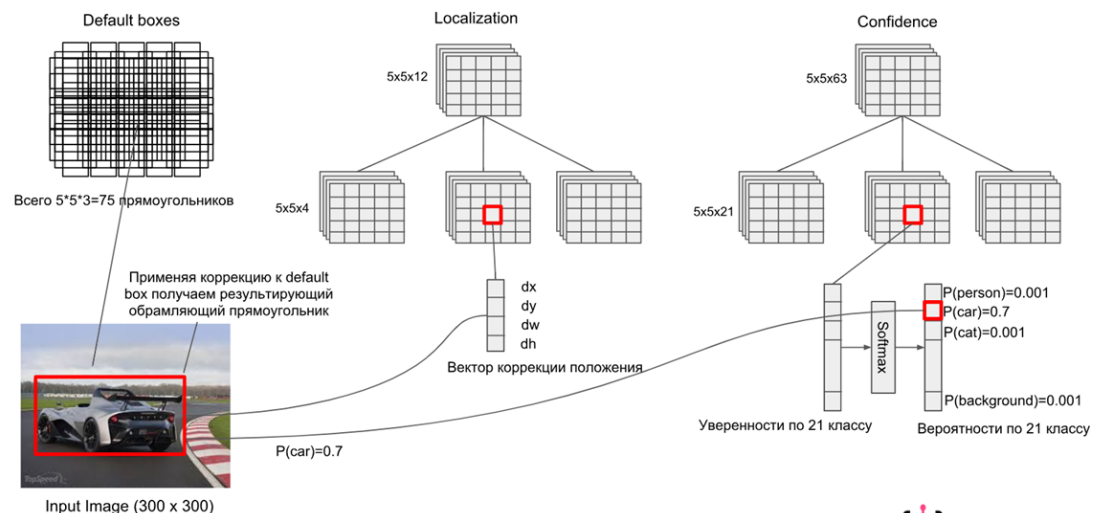
Коррекция границ, классификация и фильтрация (1)



deepsystems.io

빨간색 점선이 matching 된 default box라고 한다면, 거기에 해당하는 cell의 같은 순서의 predicted bounding box의 offset만 update 되고 최종적으로는 아래와 같이 predict 된다.

Коррекция границ, классификация и фильтрация (1)



deepsystems.io

4. Matching strategy

학습을 진행할 때 default box의 학습 대상을 지정하기 위해 어떤 default box가 어떤 ground truth box와 대응하는지 결정해야 한다.

이를 위해 **default box와 ground truth box를 매칭하는 작업**이 필요

- 두 영역의 IOU가 0.5 이상인 것들을 match

하지만 일반적으로 이미지 내에서 배경(background)에 해당하는 box가 많기 때문에 **negative sample의 수가 positive sample의 수보다 훨씬 많음**

- 클래스 불균형(class imbalance) 문제가 발생

이를 해결하기 위해 높은 confidence loss를 가진 sample을 추가하는 **hard negative mining**을 수행

hard negative - 실제로 negative인데 positive라고 잘못 예측하기 쉬운 데이터


hard negative mining 는 hard negative 데이터를 (학습데이터로 사용하기 위해) 모으는 (mining) 것이다.

hard negative mining 으로 얻은 데이터를 원래의 데이터에 추가해서 재학습하면 false positive 오류에 강해진다.

→ 이 때 positive/negative sample의 비율: 1:3

R-CNN 논문 리뷰와 OHEM 논문 리뷰를 참고

5. Loss function

$$L(x, c, l, g) = \frac{1}{N} (L_{conf}(x, c) + \alpha L_{loc}(x, l, g))$$


Class Score Bounding box Offset

전체적으로 로스는 Yolo의 로스 평션과 매우 유사

각 클래스 별로 예측한 값과 실제 값 사이의 차인 L_{conf} 와 바운딩 박스 리그레션 예측 값과 실제 값 사이의 차인 L_{loc} 를 더한 값입니다. L_{conf} 먼저 더 자세히 들여다 보겠습니다.

$$L_{conf}(x, c) = - \sum_{i \in Pos} x_{ij}^p \log(\hat{c}_i^p) - \sum_{i \in Neg} \log(\hat{c}_i^0) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$$

(기본적으로 Cross Entropy Loss라고 생각하시면 됨)

여기서 x_{ij}^p 라는 값이 등장합니다. 이는 즉 특정 grid의 i번째 default box가 p클래스의 j번째 ground truth box와 match가 된다 (IoU가 0.5 이상)라는 의미입니다. 즉, 모델이 물체가 있다고 판별한 default box들 가운데서 해당 박스의 ground truth 박스하고만 cross entropy loss를 구하겠다는 의미입니다. 뒷 부분은 물체가 없다고 판별한 default box들 중에 물체가 있을 경우의 loss를 계산해줍니다. 다음으로 L_{loc} 을 보겠습니다.

$$L_{loc}(x, l, g) = \sum_{i \in Pos} \sum_{m \in \{cx, cy, w, h\}} x_{ij}^k \text{smooth}_{L1}(l_i^m - \hat{g}_j^m)$$

$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \quad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h$$

$$\hat{g}_j^w = \log\left(\frac{g_j^w}{d_i^w}\right) \quad \hat{g}_j^h = \log\left(\frac{g_j^h}{d_i^h}\right)$$

Faster R-CNN 모델과 마찬가지로 **default box의 중심 좌표 (cx, cy)와 너비와 높이 (w, h)를 사용하여 smooth L1 loss**를 통해 구함. smoothL1은 Fast RCNN에서 제시된 Robust bounding box regression loss이다.

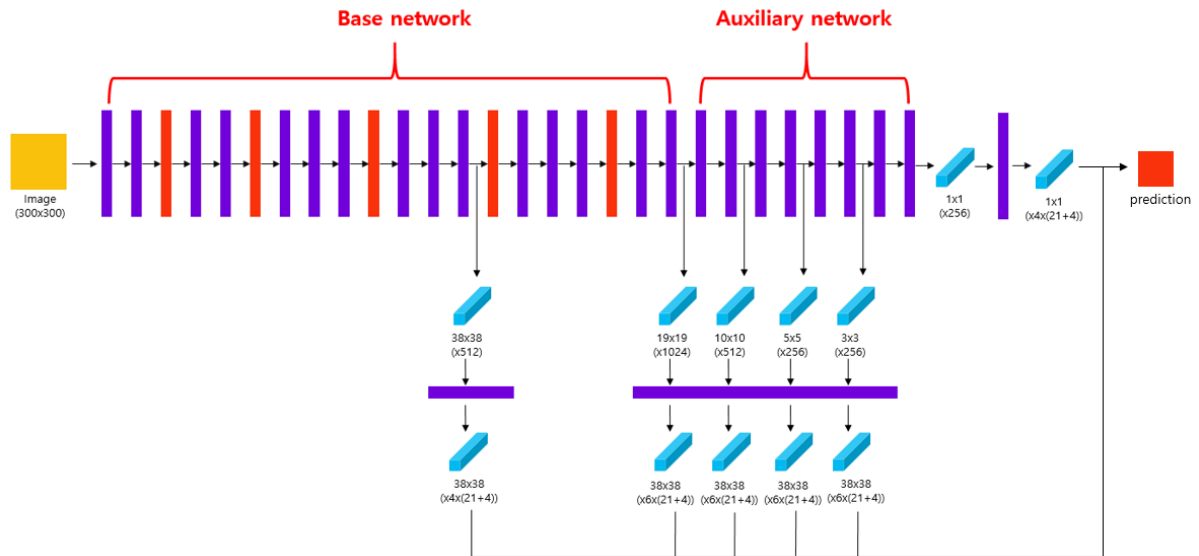
그 아래는 bounding box regression 시에 사용하는 예측 값들을 말한다.

x, y 좌표 값은 **절대 값**이기 때문에 **예측값과 실제 값 사이의 차를 default 박스의 너비 혹은 높이로 나누었다** → 값이 0에서 1 사이로 정규화되는 효과

너비와 높이의 경우엔 **로그를 씌워서 정규화** 시킨 것

l 은 **예측한 box의 파라미터**(여기선 좌표), g 는 **ground truth box의 파라미터**(여기선 좌표)를 의미

Training SSD



1) 전체 Network 구성하기(base network + auxiliary network)

학습을 위해 base network와 auxiliary network를 합쳐 전체 네트워크를 구성합니다. pre-trained된 VGG16 모델을 불러와 마지막 2개의 fc layer를 conv layer로 대체합니다. 이후 최종 output feature map의 크기가 1x1이 되도록 auxiliary network를 설계합니다.

2) 이미지 입력 및 서로 다른 scale의 feature map 얻기

SSD network에 300x300 크기의 이미지를 입력합니다. 이후 전체 network 구간 중 conv4_1, conv7, conv8_2, conv9_2, conv10_2, conv11_2 layer에서 각각 feature map을 추출합니다.

- **Input** : 300x300 sized image
- **Process** : feature map extraction
- **Output**
 - 38x38(x512) sized feature map
 - 19x19(x1024) sized feature map
 - 10x10(x512) sized feature map
 - 5x5(x256) sized feature map
 - 3x3(x256) sized feature map
 - 1x1(x256) sized feature map

3) 서로 다른 scale의 feature map에 conv 연산 적용하기

앞의 과정에서 얻은 서로 다른 scale의 feature map에 3x3 conv(stride=1, padding=1) 연산을 적용합니다. 이 때 각 feature map마다 서로 다른 수의 default box를 사용함에 주의해야 합니다.

- **Input** : 6 feature maps
- **Process** : 3x3 conv(stride=1, padding=1)
- **Output**
 - 38x38(x4x(21+4)) sized feature map
 - 19x19(x6x(21+4)) sized feature map
 - 10x10(x6x(21+4)) sized feature map
 - 5x5(x6x(21+4)) sized feature map
 - 3x3(x6x(21+4)) sized feature map
 - 1x1(x4x(21+4)) sized feature map

4) 전체 feature map 병합하기

3)번 과정에서 얻은 모든 feature map을 8732 x (21+4) 크기로 병합합니다. 이를 통해 default box별로 bounding box offset 값과 class score를 파악할 수 있습니다.

- **Input** : 6 feature maps
- **Process** : merge feature maps
- **Output** : 8732 x (21+4) sized feature map

5) loss function을 통해 SSD network 학습시키기

위에서 얻은 feature map과 ground truth 정보를 활용하여 localization loss를 계산합니다. 이후 negative sample에 대하여 Cross entropy loss를 구한 후 loss에 따라 내림차순으로 정렬합니다. 그 다음 negative sample에서 loss가 높은 순으로 positive sample의 3배만큼의 수를 추출합니다. 이러한 **hard negative mining** 과정을 통해 얻은 hard negative sample과 positive sample을 사용하여 confidence loss를 계산합니다. 앞서 얻은 localization loss와 confidence loss를 더해 최종 loss를 구한 후 backward pass를 수행하여 network를 학습시킵니다.

Results

| Method | VOC2007 test | | VOC2012 test | | COCO test-dev2015 | | |
|---------|--------------|-------------|--------------|-------------|-------------------|-------------|-------------|
| | 07+12 | 07+12+COCO | 07++12 | 07++12+COCO | trainval35k | | |
| | 0.5 | 0.5 | 0.5 | 0.5 | 0.5:0.95 | 0.5 | 0.75 |
| SSD300 | 74.3 | 79.6 | 72.4 | 77.5 | 23.2 | 41.2 | 23.4 |
| SSD512 | 76.8 | 81.6 | 74.9 | 80.0 | 26.8 | 46.5 | 27.8 |
| SSD300* | 77.2 | 81.2 | 75.8 | 79.3 | 25.1 | 43.1 | 25.8 |
| SSD512* | 79.8 | 83.2 | 78.5 | 82.2 | 28.8 | 48.5 | 30.3 |

PASCAL VOC 데이터셋으로 시험했을 때, Faster R-CNN 모델은 7FPS, mAP=73.2%, YOLO v1 모델은 45FPS, mAP=63.4%의 성능을 보인 반면 SSD 모델은 59FPS, mAP=74.3%라는 **놀라운 속도와 detection 정확도**를 보였습니다.

단점으로는 작은 물체에 대해 성능이 떨어진다.

왜그러냐면 작은 물체는 첫번째 feature map에서 추출이 될 것인데 이는 아직 abstract level이 높지 않은 feature map이기 때문에 성능이 좋지 않다라고 말한다.