**YOLO v1 –> YOLO v2(9000) -> YOLO v3**

**YOLO v1**

* 실시간 다중 객체 인식 모델
* 객체인식을 회귀문제로 인식하여, bounding box predict와 class 분류를 한번에 수행

*(We reframe object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities.)*

* + 기존 R-CNN등의 방식은 region proposal -> classification 의 two stage detector였으나, YOLO의 경우 Single Neural Network로 one stage로 구현
* 이러한 unified architecture는 complex pipeline의 제거로 **검출 속도의 향상**을 가져왔고, 실시간 객체인식이 가능한 수준의 fps에 도달 ( YOLO 45fps, Fast YOLO 155fps.. )
* Prediction 단계에서 **global하게 예측함**. Training 또는 test단계에서 전체 이미지를 보고 문맥적 인 정보를 인코드하여, **background error가 낮은 편**

*(Second, YOLO reasons globally about the image when making predictions. Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance.)*

* **객체를 일반화하는 성능**이 뛰어남. 새로운 domain이나 unexpected input에 대하여 break될 확률이 적다.
* **정확도 면에서 기존의 방식들 보다는 떨어질 수 있음. Trade-off**

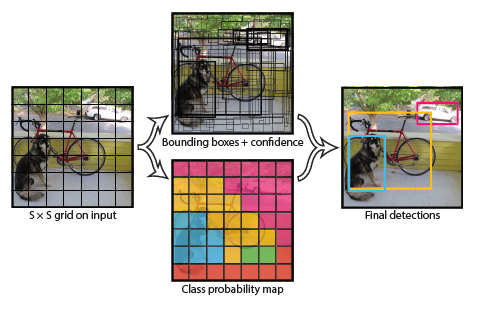
**Unified Detection**

* 이미지를 S x S grid로 나눈 뒤, 객체의 중심이 grid cell에 포함되면 그 grid cell이 객체 detection을 맡게 된다.
* 각 **grid cell은 Bounding box정보(x,y,w,h)와 confidence score**(박스안에 객체가 있을 확률이 어느정도 인지(box) / 해당 object가 예측한 object일 확률이 얼마나 되는가(objectness)) **+ 각 class에 대한 확률(softmax로 구현)**

*(Confidence prediction represents the IOU between the predicted box and any ground truth box.)*

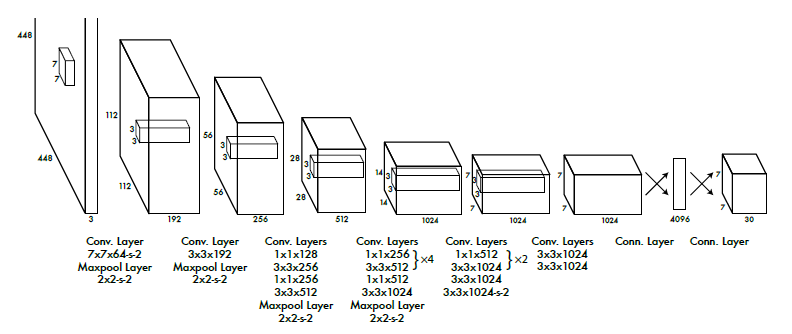


* 이러한 구조는 예측된 박스에 상관없이 각 grid cell당 하나의 class 를 가지게 된다.
* 최종적으로는 confidence score과 class probability의 곱을 통해, 해당 박스의 class를 한번에 예측하는 방식



(<https://docs.google.com/presentation/d/1kAa7NOamBt4calBU9iHgT8a86RRHz9Yz2oh4-GTdX6M/present?ueb=true&slide=id.g151008b386_0_44>)

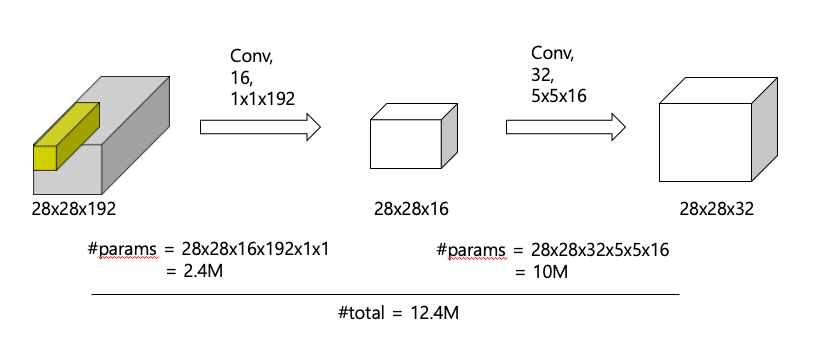
**Network Design**



* Convolutional neural net으로 구현 (YOLO v1에서는 24 convolutional layer과 마지막단에 2개의 fully connected layer로 구성되어 있으나, 마지막단의 fully connected layer는 yolo v2에서는 convolution network로 대체)
* 1x1 reduction layer과 3x3 convolutional layer사용

(**1x1 reduction layer는 dimension reduction**을 위해 사용하는 것. 즉 **filter 개수를 줄이고 -> 계산량 감소 -> 더욱 deep한 network 구성 가능**. feature pooling..)

#### https://t1.daumcdn.net/cfile/tistory/99FCEC335C0DF8351D

****

**YOLO v2**

* YOLO의 한계점. 낮은 Recall / localization error (다른 object detection에 비해서 region proposal과 같이 많은 box를 검출해낸 뒤 검사하는 방식이 아니라 단지 7x7x2개의 box를 검출하고 class를 분류하는 방식이었기 때문)
* Recall과 localization을 개선하는 데 집중한 것이 YOLO v2

성능을 높이기 위해 더욱 큰 network를 사용하거나, 여러 개의 model을 혼합하는 방식을 사용할 수도 있으나, 빠른 속도인 YOLO의 장점을 유지하고자 network를 더욱 간소화하고 여러 개선점을 적용.

**Batch Normalization(BN)**

Vanishing Gradient를 방지하기 위한 아이디어의 하나로, 기존에는 Activation function으로 ReLU를 사용하거나, small learning rate로 학습시키는 방법 등으로 해결하였으나, 이러한 간접적인 방식이 아닌 training 과정 자체를 안정화하기 위한 방법이 BN이다.

이러한 불안정화의 원인은 Internal Covariance Shift이고, 아래의 설명과 같다.

Internal Covariance Shift라는 현상은 Network의 각 층이나 Activation 마다 input의 distribution이 달라지는 현상을 의미한다. 이 현상을 막기 위해서 간단하게 각 층의 input의 distribution을 평균 0, 표준편차 1인 input으로 normalize 시키는 방법을 생각해볼 수 있고, 이는 [whitening](https://en.wikipedia.org/wiki/Whitening_transformation)이라는 방법으로 해결할 수 있다. Whitening은 기본적으로 들어오는 input의 feature들을 uncorrelated 하게 만들어주고, 각각의 variance를 1로 만들어주는 작업이다.

(즉, 학습 단계에서 mini-batch 단위로 학습할 때, vanishing gradient 현상을 방지하고, 자체적인 regularization 효과를 얻기 위해 YOLO v2에 적용한 것으로 이해..)

**High Resolution Classifier**

YOLO의 경우 Pre-trained된 classifier로 ImageNet data를 활용할때 224x224이미지로 학습을 진행했으나, YOLO v2의 경우 448x448로 이미지 해상도를 키워서 pre-train 했으므로, 큰 이미지에 대하여 filter를 adjust할 때에도 유리하다.

**Convolutional With Anchor Box**

기존 YOLO의 경우 bounding box를 예측할 때 fully connected layer를 사용했으나, YOLO v2의 경우 직접적인 좌표 예측대신 **anchor box**를 사용하여 offset과 confidence를 예측한다.

1. 네트워크의 output을 높은 해상도로 만들어주는 pooling layer(fully connected)제거
2. 객체 인식을 448 x 448 image 대신 416 x 416 image에서 수행하도록 바꿈.

* 기존 448x448의 경우 layer들을 통과하여 최종적으로 14x14 resolution feature map이 생성되게 되는데, 이 경우에는 짝수이므로 grid의 정확한 center가 1개가 아닌 4개로 설정. 그러나, 416x416의 경우 13x13개가 형성되므로, image의 중앙에 위치한 object를 담당하는 grid가 1개이므로 더욱 효율적.

(큰 객체의 경우 보통 이미지의 중앙에 위치하는 경향이 있으므로 이러한 변화 적용)

또한, 기존 YOLO의 경우 각 grid cell마다 bounding box를 2개씩 예측하여 다음과 같은 형태를 가지게 된다. {x1,y1,w1,h1,confidence,x2,y2,w2,h2,C1,C2…Cn}

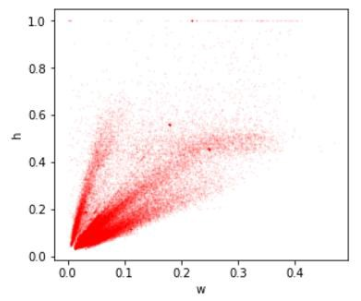
하지만 YOLO v2에서는 각 grid cell마다 일정한 개수의 anchor box를 사용하고, 각 anchor box마다 이러한 objectness와 class probability를 가지게 된다.

기존의 방식보다 더욱 더 많은 box를 예측하게 되므로, recall을 상승시키는 효과

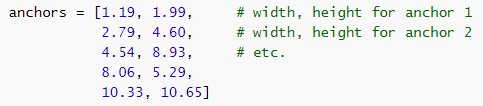
**Dimension Clusters**

**Anchor box**를 선정할 때, Faster R-CNN등과 같이 미리 선정된 box를 사용하는 것이 아닌, 데이터에 근거해 box를 선정(Training set에서 k-means clustering을 통해 자동적으로 adjust)

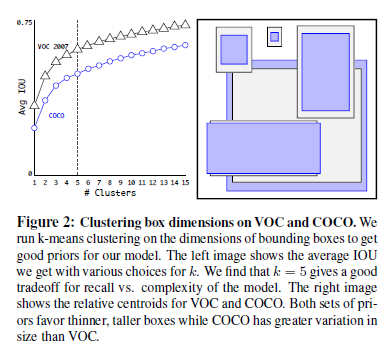
단, k-means clustering을 기존 방식과 똑같이 Euclidean distance를 사용해서 구현하면, 크기가 큰 박스와 작은 박스를 비교했을 때 error가 커질 수 있기 때문에, ground-truth box의 **center를 일치시킨 뒤**(box의 x,y값을 한점에 모아놓고 w,h값만을 고려한다는 개념으로 이해) IOU(Intersection Over Union)을 이용하여 거리를 구한다.



(해당 이미지처럼 center를 고정한 뒤 w,h 값만을 고려하고, 이러한 상태에서 위의 식을 이용해 clustering을 하고, 적절한 k를 선정하는 것)



(따라서 최종적으로 anchor box는 width, height값만을 가지는 형태가 된다)



(VOC와 COCO 데이터를 이용해서, k를 결정하는 그래프. k가 커지면 그만큼 계산량도 많아지고 모델의 복잡도가 상승하므로 적절한 k를 선정하는 것이 중요하다.)

**Ditection location prediction**

역시 **Anchor box**와 관련된 이슈로, 초기 학습 단계에서 박스의 x,y 좌표를 예측할 때 모델의 불안정성이 나타나게 된다.

기존의 region proposal network는 다음과 같은 식으로 x,y 좌표를 계산하게 된다.

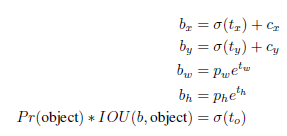
https://t1.daumcdn.net/cfile/tistory/99117F3E5B2747C228

예를들어, tx=1이라면 box를 width만큼 오른쪽으로 이동하고, tx=-1이라면 box를 왼쪽으로 이동하는 방식이다.

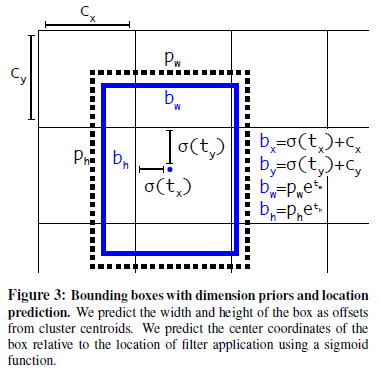
(이러한 방식이 왜 문제인가? YOLO의 경우 grid를 나누고 각 grid cell마다 일정한 개수의 anchor box를 이용하여 예측을 하는 방식인데, 위의 식을 적용할 경우 anchor box가 해당 grid cell을 벗어나서 옆의 grid cell에서 객체 인식을 하고 있는 경우가 생길 수 있다는 것)

따라서, YOLO v2에서는 이런 방식 대신 box가 grid cell을 벗어나지 않도록 offset 값을 [0,1] 사이의 값으로 제한을 둔다.(logistic activation을 활용하여)

위에서 설명했듯, 각 grid cell은 5개의 anchor box를 가지고 예측을 하고, 각 box는  https://t1.daumcdn.net/cfile/tistory/99B7F4435B274A6014(to는 objectness 값) 값을 가지게 되고, 식은 다음과 같다.



(Cx, Cy는 grid cell의 왼쪽 상단 좌표. Pw, Ph는 anchor box의 처음 width, height %5Csigma%20%20 = sigmoid)



**Fine-Grained Features**

YOLO v2는 13x13 feature map(grid cell)에서 예측한다. 이 경우 크기가 큰 object에는 충분하지만, 크기가 작은 object에는 불충분할 수 있다.

기존의 Faster R-CNN과 SSD의 경우 모델 내의 Network를 거치면서 생성된 다양한 해상도의 feature map을 이용하는데, YOLO v2는 이와 다르게 이전 레이어의 feature map을 26x26의 크기로 가져오는 passthrough layer방식을 사용한다. 가져온 26x26 크기의 feature map을 13x13 layer에 붙이는 것인데, (이때 단순히 같은 이미지를 더 많은 cell로 나누는 개념이 아닌 26x26x512 feature map을 13x13x(512x4)로 만드는 방식이다. 즉, 더욱 많은 kernel을 표현하는 것으로 이해하고, 다양한 feature 정보를 지니게 된다고 이해하면 될 듯..)

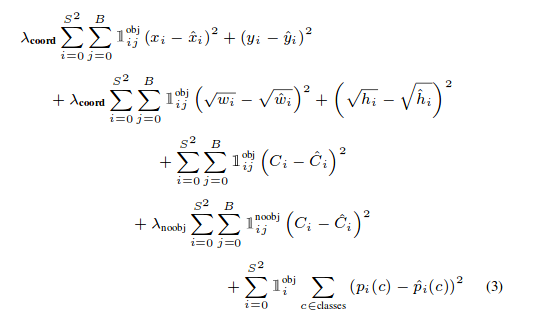
**Multi-Scale Training**

YOLO v2는 fully connected layer를 제거하여 convolution layer로 대체하였기 때문에, 어떠한 해상도의 image도 학습에 사용할 수 있다. (320x320 ~ 608x608)

따라서, 다양한 해상도에 대해 다양하게 학습하기 위해서 10번의 배치마다 학습 데이터를 {320,352,…,608}으로 32의 배수로 늘어나는 해상도 중 임의로 선택해서resize해서 학습하는 것이다.

* 다양한 해상도의 input에 대응할 수 있다.

**loss**



1. Object가 존재하는 grid cell i에서 예측한 bounding box j에 대해 x,y loss 계산
2. 위와 같으며, w,h loss 계산(큰 박스와 작은 박스와의 차이를 줄이기 위해 제곱근 취함)
3. Object가 존재하는 경우 confidence score의 loss 계산 (Ci=1)
4. Object가 존재하지 않는 경우 confidence score의 loss 계산 (Ci=0)
5. Object가 존재하는 경우, class probability의 loss 계산(Pi(c)=맞으면 1, 틀리면 0)

**YOLO v3**

**Bounding Box Prediction**

기존 YOLO 9000의 방식과 동일하지만, object score를 logistic regression으로 구한다. 예측된 박스가 다른 모든 박스들보다 IOU가 높다면 score는 1이 되게 되고, 어떠한 threshold를 넘지만 가장 높은 score가 아니라면 무시하게 된다. (각 ground truth object에 대하여 단 하나의 매칭만 하게 되고, 만약 매칭이 되지 않는다면 좌표 및 클래스 예측에는 loss가 적용되지 않고, Objectness에 대해서만 loss 적용)

**Class Prediction**

YOLO v3는 class probability를 구할 때 softmax를 사용하지 않고, 그냥 logistic classifier를 사용한다. 그 이유는, softmax의 경우 만약 person과 woman이라는 class가 있다면 둘 다 겹치는 요소이기 때문에 오히려 확률이 떨어질 가능성이 있기 때문이다

**Predictions Across Scales**,

YOLO v3는 스케일을 3번 다르게 해서 박스를 예측하는데, feature pyramid networks와 유사하다. 각 스케일에서 3개의 box를 예측하고, 그 다음 스케일에서는 이전의 feature map의 정보도 이용하여 upsampling을 하는 방식으로 진행(이전에 지나온 feature map의 정보도 이용한다고 이해). 즉, 단 한번의 예측이 아닌 최종 scale에서는 여러 번의 예측을 한 뒤에 나온 정보라는 것 (anchor box가 9개이니까 3개씩 나눠서 진행)

(<https://medium.com/@lsrock125/feature-pyramid-networks-for-object-detection-%EB%85%BC%EB%AC%B8%EC%9D%BD%EA%B8%B0-e4e577c4b423>)

**Feature Extractor**

Darknet 53(Darknet 19와 newfangled residual network의 혼합). 3x3과 1x1 convolutional layer를 더욱 추가했고, Shortcut connection(이전 네트워크의 output을 전달하는..?)

Network 구조가 GPU 활용에 최적화된 모델을 구성.