YOLO (You only look once)

Abstract

We present YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

Our unified architecture is extremely fast. Our base YOLO model processes images in real-time at 45 frames per second. A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second while still achieving double the mAP of other real-time detectors. Compared to state-of-the-art detection systems, YOLO makes more localization errors but is less likely to predict false positives on background. Finally, YOLO learns very general representations of objects. It outperforms other detection methods, including DPM and R-CNN, when generalizing from natural images to other domains like artwork.

기존의 multi-task 문제를 하나의 regression problem으로 재정의함

YOLO는 이미지 전체에 대해서 Single Neural Network이 한번만의 계산으로 bounding box와 class probability를 예측함.

파이프라인이 하나의 신경망 → end to end 형식

YOLO 초기모델은 초당 45 frame Fast YOLO는 초당 155 frame

→ 정말 빨라요!

natural image ~ 에서 일반화 할때 DPM, R-CNN을 능가하는 성능을 보임.

1. Introduction

Humans glance at an image and instantly know what objects are in the image, where they are, and how they interact. The human visual system is fast and accurate, allowing us to perform complex tasks like driving with little conscious thought. Fast, accurate algorithms for object detection would allow computers to drive cars without specialized sensors, enable assistive devices to convey real-time scene information to human users, and unlock the potential for general purpose, responsive robotic systems.

Current detection systems repurpose classifiers to perform detection. To detect an object, these systems take a classifier for that object and evaluate it at various locations and scales in a test image. Systems like deformable parts models (DPM) use a sliding window approach where the classifier is run at evenly spaced locations over the entire image [10].

More recent approaches like R-CNN use region proposal methods to first generate potential bounding boxes in an image and then run a classifier on these proposed boxes. After classification, post-processing is used to refine the bounding boxes, eliminate duplicate detections, and rescore the boxes based on other objects in the scene [13]. These complex pipelines are slow and hard to optimize because each individual component must be trained separately.

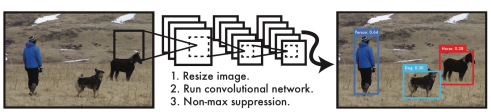


Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448 × 448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

We reframe object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. Using our system, you only look once (YOLO) at an image to predict what objects are present and where they are.

YOLO is refreshingly simple: see Figure 1. A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance. This unified model has several benefits over traditional methods of object detection.

기존의 detection 모델은 classifier를 재정의 하여 detector로 사용.

하지만 object detection은 하나의 이미지 내에서 개는 어디 있고 고양이는 어디 있는지 판단.

따라서 object detection은 분류 뿐만 아니라 위치 정보도 판단해야 함!

1

Deformable parts models (DPM)은 이미지 전체에 sliding window 방식으로 객체를 검출하는 모델

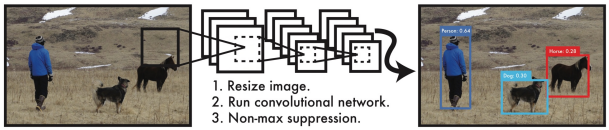
R-CNN은 이미지 안에서 bounding box를 생성하기 위해 region proposal 방법 사용!

→ 이러한 모델들은 복잡하기에 느림!

각 절차를 독립적으로 훈련해야 하기에 optimization 어려움.

이미지의 픽셀로 부터 bounding box의 위치(coordinates) 클래스 확률을 구하는 과정을 regressor problem으로 재정의

하나의 파이프라인으로 이미지 내 어떤 물체가 있고 어디에 있는지!



YOLO is refreshingly simple: see Figure 1. A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance. This unified model has several benefits over traditional methods of object detection.

First, YOLO is extremely fast. Since we frame detection as a regression problem we don't need a complex pipeline. We simply run our neural network on a new image at test time to predict detections. Our base network runs at 45 frames per second with no batch processing on a Titan X GPU and a fast version runs at more than 150 fps. This means we can process streaming video in real-time with less than 25 milliseconds of latency. Furthermore, YOLO achieves more than twice the mean average precision of other real-time systems. For a demo of our system running in real-time on a webcam please see our project webpage: http://pjreddie.com/yolo/.

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. Fast R-CNN, a top detection method [14], mistakes background patches in an image for objects because it can't see the larger context. YOLO makes less than half the number of background errors compared to Fast R-CNN.

하나의 convolution Network이 여러 bounding box와 class probabilites를 동시에 계산해줌.

YOLO는 이미지 전체를 학습하여 곧바로 detection performance를 최적화함.

YOLO는 짱 빠름! → 기존의 복잡한 프로세스를 하나의 회귀 문제로 바꿔서!

다른 real-time object detection 보다 2배 이상의 mAP를 갖음.

sliding window 나 region proposal 방식과 달리 YOLO 는 훈련과 테스트에서 이미지 전체를 봄.
그리하여 클래스의 모양에 대한 정보 뿐만 아니라 주변 정보까지 학습하여 처리함.
→ Fast R-CNN은 주변정보 처리 X!
∵ background 나 noise가 존재하면 물체로 인식! ⇒ background Error!

YOLO는 Fast R-CNN에 비해 background Error 가 절반 정도 줄어듦!

Third, YOLO learns generalizable representations of objects. When trained on natural images and tested on artwork, YOLO outperforms top detection methods like DPM and R-CNN by a wide margin. Since YOLO is highly generalizable it is less likely to break down when applied to new domains or unexpected inputs.

YOLO still lags behind state-of-the-art detection systems in accuracy. While it can quickly identify objects in images it struggles to precisely localize some objects, especially small ones. We examine these tradeoffs further in our experiments.

All of our training and testing code is open source. A variety of pretrained models are also available to download.

YOLO는 물체의 일반적인 부분을 학습

Natural image를 학습하여 Art-image로

test 한때 DPM이나 R-CNN보다 뛰어남!

따라서 새로운 이미지에 대해 robust하다!

→ 검출 정확도가 높다!

but Sota model 보다 정확도↓

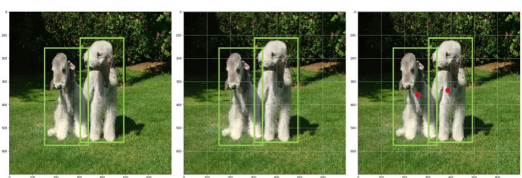
→ 작은 물체에 대한 검출 정확도↓

2. Unified Detection

We unify the separate components of object detection into a single neural network. Our network uses features from the entire image to predict each bounding box. It also predicts all bounding boxes across all classes for an image simultaneously. This means our network reasons globally about the full image and all the objects in the image. The YOLO design enables end-to-end training and real-time speeds while maintaining high average precision.

Our system divides the input image into an $S \times S$ grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object.

Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts. Formally we define confidence as $\Pr(\text{Object}) * \text{IOU}^{\text{truth}}_{\text{pred}}$. If no object exists in that cell, the confidence scores should be zero. Otherwise we want the confidence score to equal the intersection over union (IOU) between the predicted box and the ground truth.



input image를 $S \times S$ grid로 나누고, 만약 어떤 객체의 중심이 특정 grid cell 안에 위치한다면 그 그리드 셀이 해당 객체를 검출해야 함. (?)

각각의 grid cell은 B개의 bounding box와 bounding box에 대한 confidence score를 예측함.

Confidence score? → bounding box가 객체를 포함한다는 것을 얼마나 믿을만 한지, 그리고 예측한 bounding box가 얼마나 정확한지 나타냄!

$$\Pr(Object) * IOU^{truth}_{pred}$$

$$IOU = \frac{\text{real bbox} \cap \text{pred bbox}}{\text{real bbox} \cup \text{pred bbox}}$$

만약 grid cell에 객체가 없다면 $Pr(Object) = 0$ ∵ confidence score = 0

→ 그리드 셀에서 어떤 객체가 있다고 확신했다 $Pr(Object) = 1$

Each bounding box consists of 5 predictions: x, y, w, h, and confidence. The (x, y) coordinates represent the center of the box relative to the bounds of the grid cell. The width and height are predicted relative to the whole image. Finally the confidence prediction represents the IOU between the predicted box and any ground truth box.

Each grid cell also predicts C conditional class probabilities, $\Pr(\text{Class}_i|\text{Object})$. These probabilities are conditioned on the grid cell containing an object. We only predict one set of class probabilities per grid cell, regardless of the number of boxes B.

At test time we multiply the conditional class probabilities and the individual box confidence predictions,

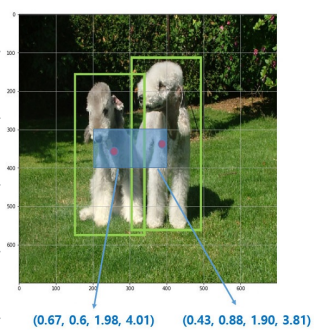
$$\Pr(\text{Class}_i|\text{Object}) * \Pr(\text{Object}) * \text{IOU}^{\text{truth}}_{\text{pred}} = \Pr(\text{Class}_i) * \text{IOU}^{\text{truth}}_{\text{pred}} \quad (1)$$

which gives us class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object.

각각의 bounding box는 x, y, w, h, confidence로 구성됨!

(x, y) 좌표 쌍은 bounding box 중심의 grid cell 내 상대 위치를 뜻함. (ex: (x, y) = (0.5, 0.5) → 중앙)

(w, h) 쌍은 bbox의 상대 너비와 상대 높이!



각각의 grid cell은 C를 예측함.

$C(\text{conditional class probabilities}) = \Pr(Class_i|Object)$

grid cell 안에 객체가 있다는 조건 하에 그 객체가 어떤 class인지에 대한 조건부 확률.

grid cell에 몇개의 bounding box가 있는지와는 무관하게 하나의 grid cell 에는 오직 하나의 class에 대한 확률 값만 구함.

$$\begin{aligned} &class\ specific\ confidence\ score \\ &= \Pr(Class_i|Object) * \Pr(Object) * IOU^{truth}_{pred} \\ &= \Pr(Class_i) * IOU^{truth}_{pred} \end{aligned}$$

→ conditional class probability * confidence score = bbox에 특정 class가 나타날 확률과 예측된 bbox가 class 객체에 얼마나 잘 맞는지!

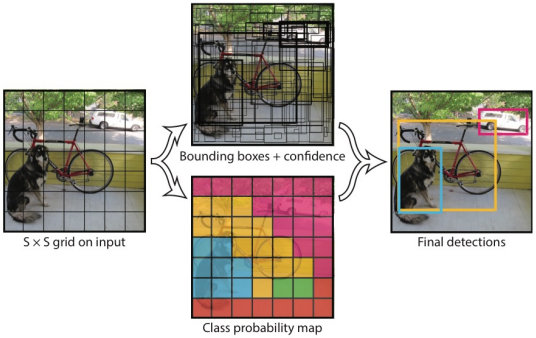


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

For evaluating YOLO on PASCAL VOC, we use $S = 7$, $B = 2$. PASCAL VOC has 20 labelled classes so $C = 20$. Our final prediction is a $7 \times 7 \times 30$ tensor.

이 이미지 셋에서는 $S=7$ $B=2$ $C=20$ (클래스)

$S=7$ 이면 input image는 7×7 grid로 나눔.

$B=2$ 면 하나의 grid cell에서 2개의 bbox를 예측

$\therefore S \times S \times (B \cdot 5 + C) = 7 \times 7 \times (2 \times 5 + 20)$

→ 최종 예측 tensor dimension = $(7 \times 7 \times 30)$

$$pred_{cell} = [c_1, c_2, \ldots, c_{20}, p_{c_1}, x, y, w, h, p_{c_2}, x, y, w, h]$$

Class probabilities | Box 1 | Box 2

Box1 confidence score | Box2 confidence score

2.1. Network Design

We implement this model as a convolutional neural network and evaluate it on the PASCAL VOC detection dataset [9]. The initial convolutional layers of the network extract features from the image while the fully connected layers predict the output probabilities and coordinates.

Our network architecture is inspired by the GoogLeNet model for image classification [34]. Our network has 24 convolutional layers followed by 2 fully connected layers. Instead of the inception modules used by GoogLeNet, we simply use 1×1 reduction layers followed by 3×3 convolutional layers, similar to Lin et al [22]. The full network is shown in Figure 3.

We also train a fast version of YOLO designed to push the boundaries of fast object detection. Fast YOLO uses a neural network with fewer convolutional layers (9 instead of 24) and fewer filters in those layers. Other than the size of the network, all training and testing parameters are the

GoogLeNet에서 가져온 신경망 구조.

24개의 conv layer와 2개의 FC layer로 구성

GoogLeNet의 인셉션 구조 대신 1×1 reduction layer

3×3 conv layer의 결합을 사용!

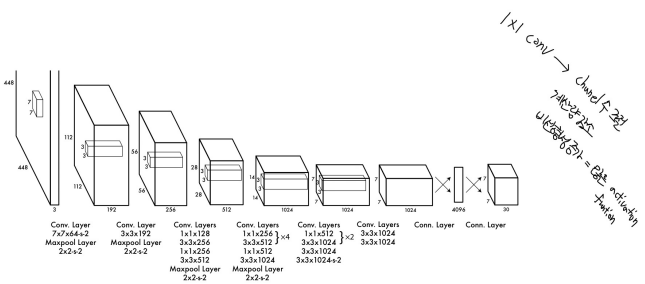


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

2.2. Training

We pretrain our convolutional layers on the ImageNet 1000-class competition dataset [30]. For pretraining we use the first 20 convolutional layers from Figure 3 followed by a average-pooling layer and a fully connected layer. We train this network for approximately a week and achieve a single crop top-5 accuracy of 88% on the ImageNet 2012 validation set, comparable to the GoogLeNet models in Caffe's Model Zoo [24]. We use the Darknet framework for all training and inference [26].

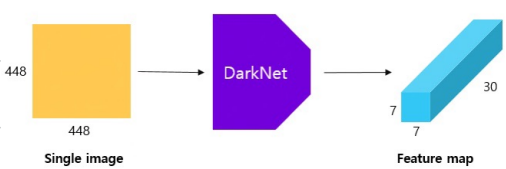
We then convert the model to perform detection. Ren et al. show that adding both convolutional and connected layers to pretrained networks can improve performance [29]. Following their example, we add four convolutional layers and two fully connected layers with randomly initialized weights. Detection often requires fine-grained visual information so we increase the input resolution of the network from 224×224 to 448×448.

Our final layer predicts both class probabilities and bounding box coordinates. We normalize the bounding box width and height by the image width and height so that they fall between 0 and 1. We parametrize the bounding box x and y coordinates to be offsets of a particular grid cell location so they are also bounded between 0 and 1.

We use a linear activation function for the final layer and all other layers use the following leaky rectified linear activation:

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.1x, & \text{otherwise} \end{cases} \quad (2)$$

We optimize for sum-squared error in the output of our

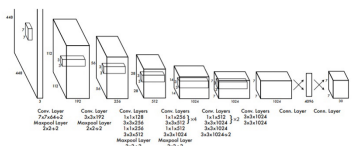


ImageNet 데이터로 분류 학습(?)

Train과 inference를 위해 DarkNet 사용!

DarkNet이 뭐지?

C 언어로 작성된 물체 인식 오픈소스 신경망 프레임워크 — Joseph Redmon



ImageNet은 분류를 위한 데이터셋 → object detection으로 바꿔야 함. 20개의 conv 뒤에 4개의 conv 와 2개의 FC 추가 하여 성능 향상!

→ 뒤에 6개의 추가된 layer의 weight는 임의로 초기화

또한 object detection을 위해서는 이미지의 해상도가 높아야 함. ∴ 224 × 224 → 448 × 448

YOLO 신경망의 마지막 계층에는 Linear activation function를 적용.

나머지 모든 계층에는 Leaky ReLU 적용!

YOLO의 Loss는 SSE (sum-squared Error)를 기반함 → SSE는 최적화 해야 함.

SSE를 사용한 이유? → 최적화하기 쉬워서!

but 최적화 한다고 mAP가 높아지는 것은 아니다?

We optimize for sum-squared error in the output of our model. We use sum-squared error because it is easy to optimize, however it does not perfectly align with our goal of maximizing average precision. It weights localization error equally with classification error which may not be ideal. Also, in every image many grid cells do not contain any object. This pushes the "confidence" scores of those cells towards zero, often overpowering the gradient from cells that do contain objects. This can lead to model instability, causing training to diverge early on.

또 다른 문제가 있는데 이미지 내 대부분의 grid cell에는 객체가 없음. 왜냐? 배경영역이 전경영역(:) 보다 크기 때문.
따라서 대부분의 grid cell의 confidence score = 0 이 되도록 학습함. 이는 모델에 악영향! (`^´)

To remedy this, we increase the loss from bounding box coordinate predictions and decrease the loss from confidence predictions for boxes that don't contain objects. We use two parameters, λ_{coord} and λ_{noobj} to accomplish this. We set $\lambda_{\text{coord}} = 5$ and $\lambda_{\text{noobj}} = .5$.

이를 개선하기 위해 객체 O coordinate의 loss weight 증가
객체 X coordinate의 loss weight 감소

Sum-squared error also equally weights errors in large boxes and small boxes. Our error metric should reflect that small deviations in large boxes matter less than in small boxes. To partially address this we predict the square root of the bounding box width and height instead of the width and height directly.

SSE는 큰 bounding box와 작은 bounding box에 대해 모두 동일한 가중치로 loss를 계산함.

큰 객체를 둘러싼 bbox는 조금 움직여도 여전히 객체를 잘 감싸지만 작은 객체를 둘러싼 bbox는 조금만 움직여도 작은 객체를 벗어남. → 개선책으로 bbox의 w와 h에 square root를 취해주어 너비와 높이가 커짐에 따라 그 증가율이 감소!
→ loss에 대한 가중치 감소!

YOLO predicts multiple bounding boxes per grid cell. At training time we only want one bounding box predictor to be responsible for each object. We assign one predictor to be "responsible" for predicting an object based on which prediction has the highest current IOU with the ground truth. This leads to specialization between the bounding box predictors. Each predictor gets better at predicting certain sizes, aspect ratios, or classes of object, improving overall recall.

YOLO는 하나의 grid cell → 여러개의 bbox 예측

training step에서 하나의 bbox predictor가 하나의 object에 대한 responsible이 있어야 함. → 객체 하나당 bbox 하나와 매칭시켜야 함.

이를 위해 예측된 여러 bbox 중 실제 객체를 감싸는 ground-truth bounding box와의 IOU가 가장 큰 것을 선택함. 이렇게 훈련된 bbox predictor는 특정크기(size) 종횡비(aspect ratio), 객체의 클래스(object class)를 잘 예측하게 됨.

Loss function

localization Loss

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

confidence Loss

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

classification Loss

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2$$

(1) Object가 존재하는 그리드 셀 i의 bounding box predictor j에 대해, x와 y의 loss를 계산.

(2) Object가 존재하는 그리드 셀 i의 bounding box predictor j에 대해, w와 h의 loss를 계산. 큰 box에 대해서는 작은 분산(small deviation)을 반영하기 위해 제곱근을 취한 후, sum-squared error를 구합니다. (같은 error라도 큰 box의 경우 상대적으로 IOU에 영향을 적게 줍니다.)

(3) Object가 존재하는 그리드 셀 i의 bounding box predictor j에 대해, confidence score의 loss를 계산. (Ci = 1)

(4) Object가 존재하지 않는 그리드 셀 i의 bounding box predictor j에 대해, confidence score의 loss를 계산. (Ci = 0)

(5) Object가 존재하는 그리드 셀 i에 대해, conditional class probability의 loss를 계산. (p_i(c)=1 if class c is correct, otherwise: p_i(c)=0)

λ_coord: coordinates(x, y, w, h)에 대한 loss와 다른 loss들과의 균형을 위한 balancing parameter.

λ_noobj: 객체가 있는 box와 없는 box 간에 균형을 위한 balancing parameter. (일반적으로 image내에는 객체가 있는 그리드 셀보다는 없는 셀이 훨씬 많으므로)

overfitting을 막기 위해 dropout과 data augmentation 적용.

(0.5)

2.3. Inference

Just like in training, predicting detections for a test image only requires one network evaluation. On PASCAL VOC the network predicts 98 bounding boxes per image and class probabilities for each box. YOLO is extremely fast at test time since it only requires a single network evaluation, unlike classifier-based methods.

The grid design enforces spatial diversity in the bounding box predictions. Often it is clear which grid cell an object falls in to and the network only predicts one box for each object. However, some large objects or objects near the border of multiple cells can be well localized by multiple cells. Non-maximal suppression can be used to fix these multiple detections. While not critical to performance as it is for R-CNN or DPM, non-maximal suppression adds 2-3% in mAP.

PASCAL VOC dataset에 대해서 YOLO는 한 image 당 98개의 bbox를 예측해주고 그 bbox 마다 클래스 확률을 구해줌!

YOLO의 grid design에는 단점이 있다.

→ 하나의 객체에 대해 여러 그리드 셀이 동시에 검출되는경우

객체의 크기가 크거나! 그리드 셀 경계에 인접해 있는 경우!

→ bbox 여러개 생성가능!

하나의객체가 정확히 하나의 그리드 셀에만 존재하는 경우에는 이런 문제가 없지만 객체의 크기나 위치에 따라 발생 가능함 → 다중 검출 (multiple detection) 문제!

↓

해결책: 비 최대 억제 (non-maximal suppression)

YOLO는 NMS로 mAP를 2-3% 향상!

2.4. Limitations of YOLO

YOLO imposes strong spatial constraints on bounding box predictions since each grid cell only predicts two boxes and can only have one class. This spatial constraint limits the number of nearby objects that our model can predict. Our model struggles with small objects that appear in groups, such as flocks of birds.

Since our model learns to predict bounding boxes from data, it struggles to generalize to objects in new or unusual aspect ratios or configurations. Our model also uses relatively coarse features for predicting bounding boxes since our architecture has multiple downsampling layers from the input image.

Finally, while we train on a loss function that approximates detection performance, our loss function treats errors the same in small bounding boxes versus large bounding boxes. A small error in a large box is generally benign but a small error in a small box has a much greater effect on IOU. Our main source of error is incorrect localizations.

YOLO는 하나의 그리드 셀 마다 두 개의 bbox를 예측. 그리고 하나의 그리드 셀 마다 오직 하나의 객체 검출.

→ 공간적 제약을 발생시킴!

공간적 제약 (spatial constraints) 이란 하나의 그리드 셀은 오직 하나의 객체만 검출하므로 하나의 그리드 셀에 두 개 이상의 객체가 붙어있다면 검출하지 못하는 문제.

YOLO 모델은 데이터로부터 bbox를 예측하는 것을 학습하기 때문에 훈련 단계에서 학습하지 못한 새로운 종횡비를 마주하면 힘들어함(?)

마지막으로 큰 bbox와 작은 bbox의 loss에 대해 동일 가중치를 준다는 점!

→ 부정확한 localization problem