Hidden Soldier Detection

양지현 김근호 김미라
권형근 오석준 우유정

KU-BIG

Index

1. Project Overview

Image Segmentation → Object Detection

KU-BIG

1. Project Overview

EfficientDet

RCNN

Model

YOLO

Nas-FPN

Accuracy

&

Speed

2. EfficientDet

2019.5.28

"EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks"

Mingxing Tan, Quoc V. Le

https://arxiv.org/abs/1905.11946

2019.11.20

"EfficientDet: Scalable and Efficient Object Detection"

Mingxing Tan, Ruoming Pang, Quoc V. Le

https://arxiv.org/abs/1911.09070

2. EfficientDet

핵심 Point

1. **Efficient multi-scale feature fusion**

   - BiFPN 구조 고안

   - 각 Input Feature에 가중치 부여

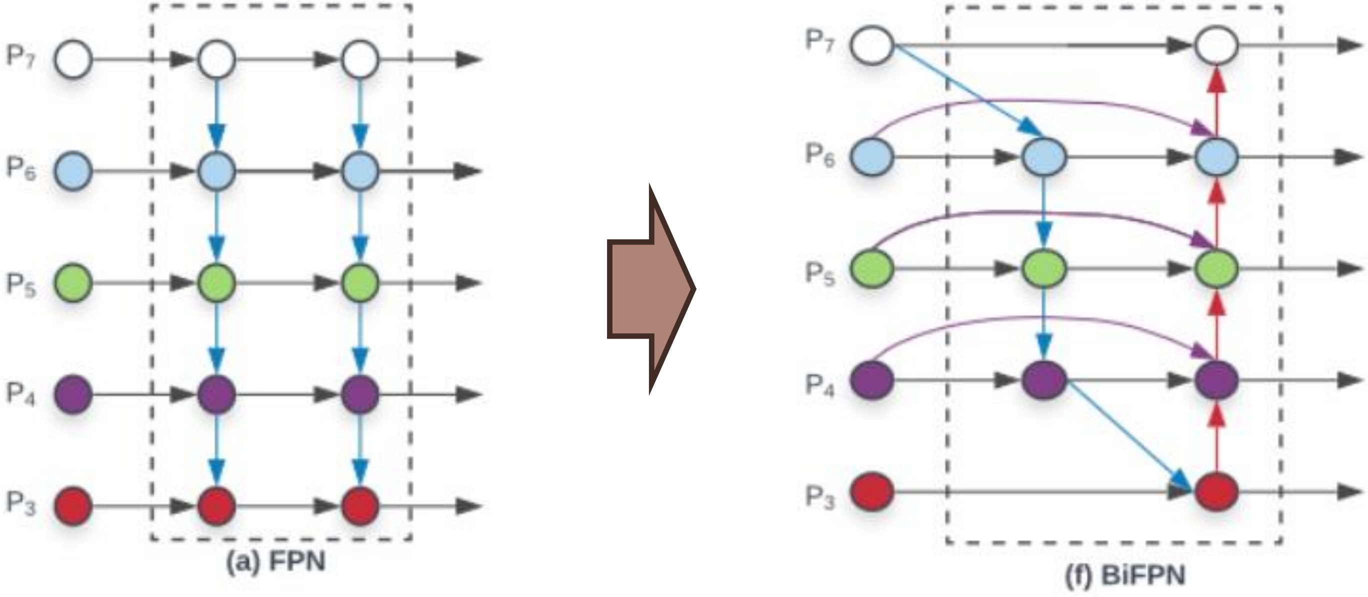
2. **Model Scaling**

   - EfficientNet 기법(Compound Scaling) 활용

   - Resolution, Depth, Width를 동시에 고려한 Scaling 기법

2. EfficientDet

➢ **Efficient multi-scale feature fusion - “BiFPN”**



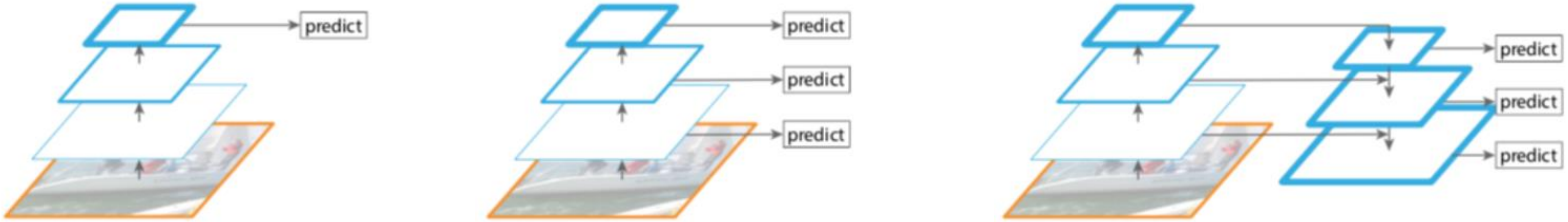
KU-BIG

2. EfficientDet

➢ Efficient multi-scale feature fusion - “BiFPN”

- **FPN(Feature Pyramid Network)란?**

  → 이미지 내에서 구하고자 하는 특징을 추출하는 구조 중 하나



(b) Single feature map (c) Pyramidal feature hierarchy (d) Feature Pyramid Network

2. EfficientDet

➢ **Efficient multi-scale feature fusion - “BiFPN”**

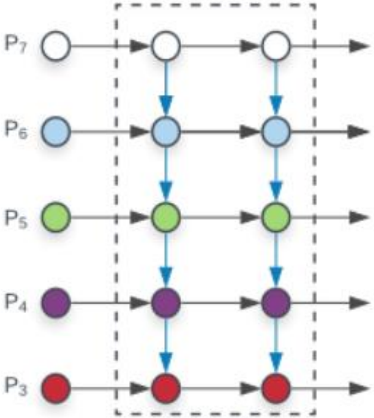
- **FPN(Feature Pyramid Network)**

Bottom-up을 진행하면서 semantic 정보를 응축

Top-down을 진행하면서 local 정보를 다시 포함

→ Semantic적인 측면과 local 정보를 적절히 고려

(d) Feature Pyramid Network

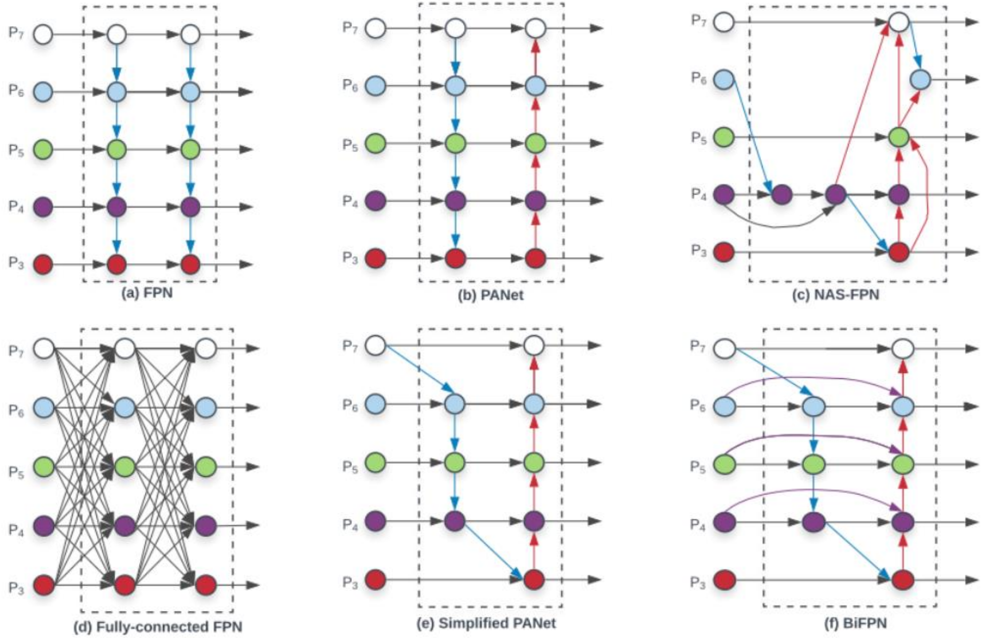


(a) FPN

KU-BIG

2. EfficientDet

➢ **Efficient multi-scale feature fusion - "BiFPN"**



- ✓ Top-down과 bottom-up 조합 고려
- ✓ 불필요한 edge 제거
- ✓ 최대한 많은 정보 포함

2. EfficientDet

➢ **Efficient multi-scale feature fusion - "BiFPN"**

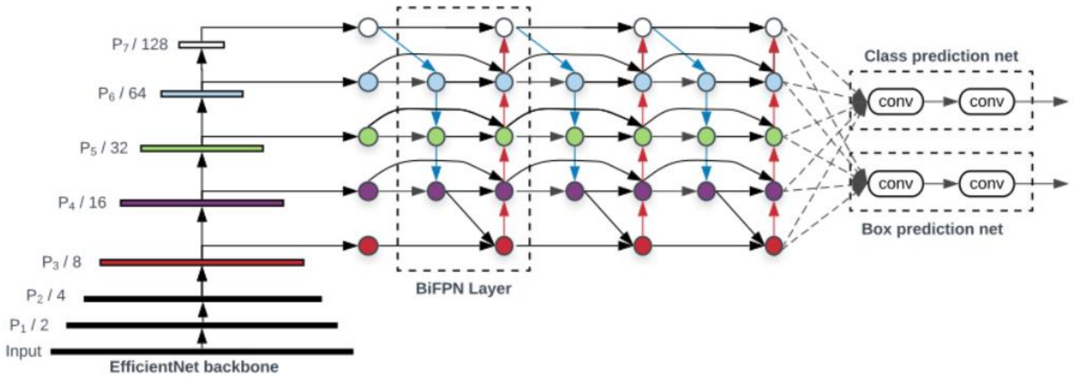


Figure 3: **EfficientDet architecture** – It employs EfficientNet [31] as the backbone network, BiFPN as the feature network, and shared class/box prediction network. Both BiFPN layers and class/box net layers are repeated multiple times based on different resource constraints as shown in Table 1.

	mAP	#Params ratio	#FLOPS ratio
Top-Down FPN [16]	42.29	1.0x	1.0x
Repeated PANet [19]	44.08	1.0x	1.0x
NAS-FPN [5]	43.16	0.71x	0.72x
Fully-Connected FPN	43.06	1.24x	1.21x
BiFPN (w/o weighted)	**43.94**	**0.88x**	**0.67x**
BiFPN (w/ weighted)	**44.39**	**0.88x**	**0.68x**

Table 4: **Comparison of different feature networks** – Our weighted BiFPN achieves the best accuracy with fewer parameters and FLOPS.

2. EfficientDet

➢ Efficient multi-scale feature fusion – "Weighted Feature Fusion"

FPN을 통해 추출한 여러 개의 input feature들을 처리하는 방법

기존 model: resize를 통해 크기를 맞춘 뒤 단순하게 합치는 방식

EfficientDet: Input feature 간에 가중치를 주고 학습을 통해 적절한 가중치를 찾는 방식

$$O = \sum_i I_i.$$

Conventional Feature Fusion

$$O = \sum_i w_i \cdot I_i,$$

Unbounded Feature Fusion

$$O = \sum_i \frac{e^{w_i}}{\sum_j e^{w_j}} \cdot I_i.$$

SoftMax-based Feature Fusion

$$O = \sum_i \frac{w_i}{\epsilon + \sum_j w_j} \cdot I_i$$

Fast normalized Feature Fusion

Model	Softmax Fusion mAP	Fast Fusion mAP (delta)	Speedup
Model1	33.96	33.85 (-0.11)	1.28x
Model2	43.78	43.77 (-0.01)	1.26x
Model3	48.79	48.74 (-0.05)	1.31x

2. EfficientDet

➢ Compound Scaling

EfficientNet

기존 모델을 바탕으로 complexity를 높임으로써 정확도를 높이는 방법 존재

Complexity를 높이는 방법: width scaling, depth scaling, resolution scaling

3가지 방법을 적절하게 늘림으로써 정확도를 높임

2. EfficientDet

➢ Compound Scaling

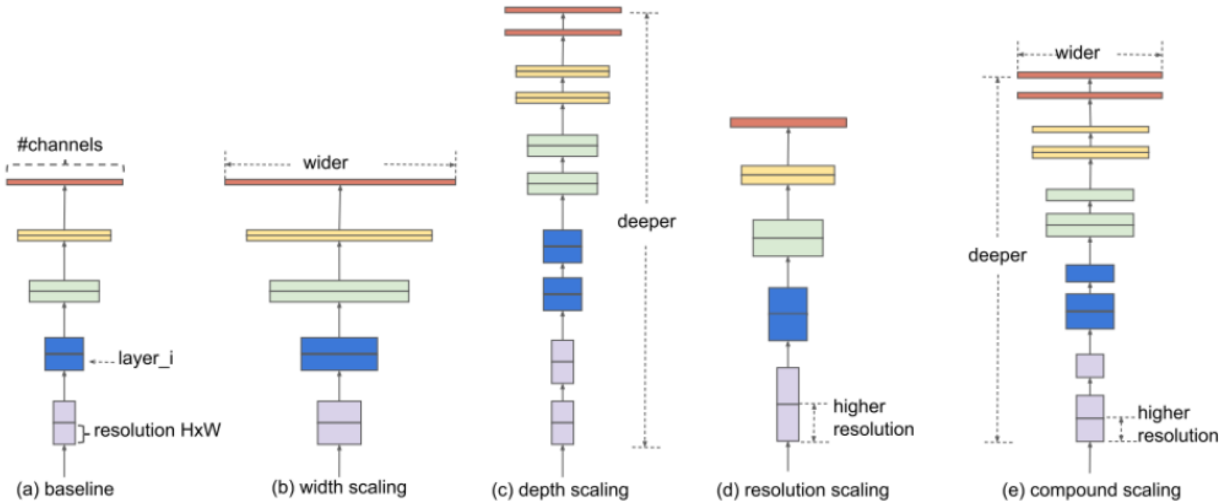
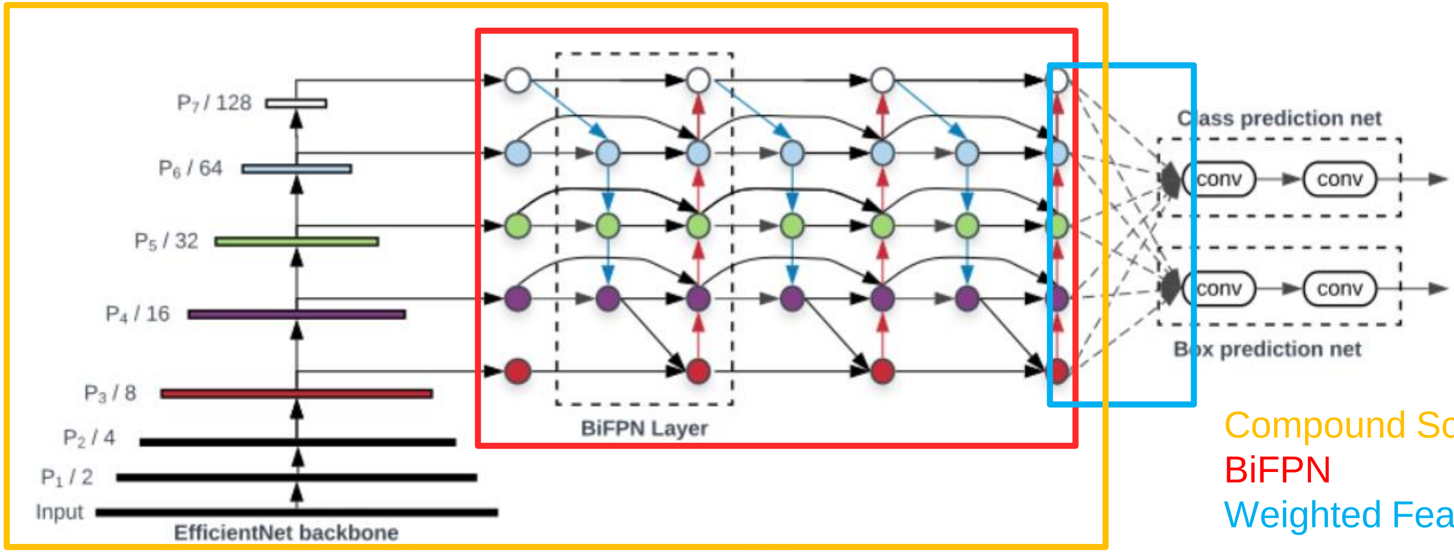


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

2. EfficientDet

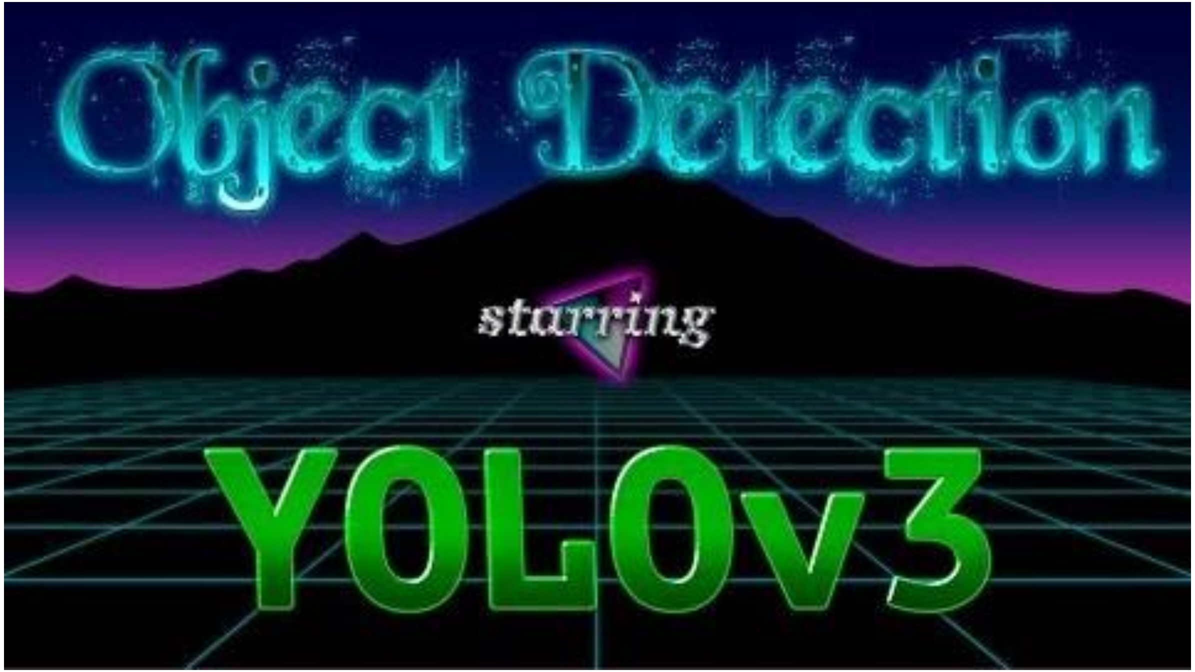
➢ **Final EfficientDet**



Compound Scaling
BiFPN
Weighted Feature Fusion

Figure 3: **EfficientDet architecture** – It employs EfficientNet [31] as the backbone network, BiFPN as the feature network, and shared class/box prediction network. Both BiFPN layers and class/box net layers are repeated multiple times based on different resource constraints as shown in Table 1.

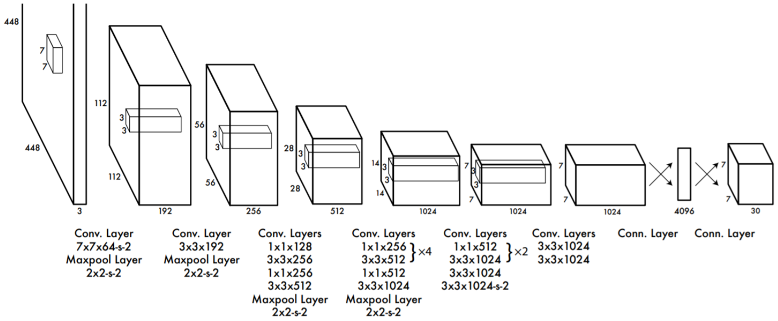
YOLO(You Only Look Once) - Real Time Object Detector

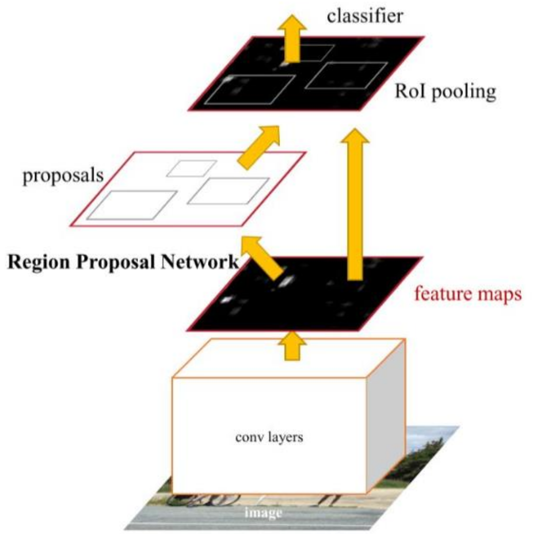


YOLO(You Only Look Once)

Advantages

1) 매우 빠른 속도 (Regression problem)





KU-BIG

YOLO(You Only Look Once)

Advantages

1) 매우 빠른 속도 (Regression problem)

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [38]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

YOLO(You Only Look Once)

Advantages
1) 매우 빠른 속도 (Regression problem)
2) Image를 전역적으로 파악 → Contextual information

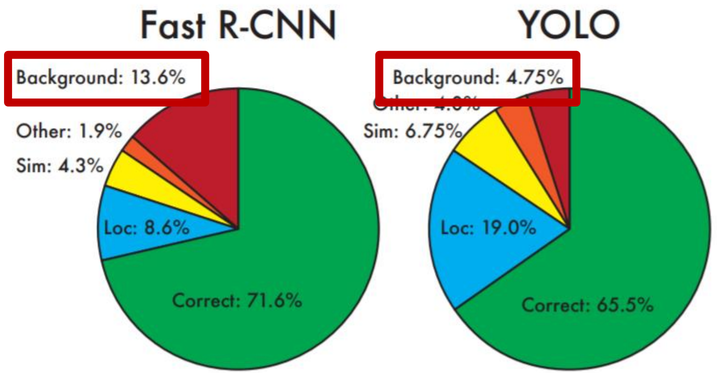
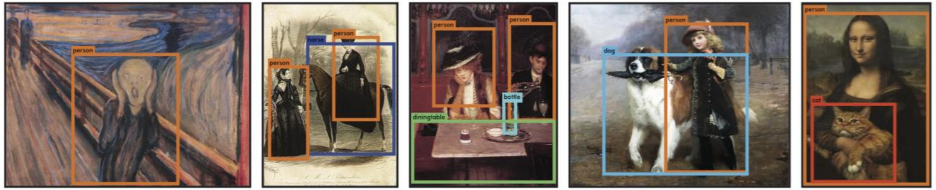


Figure 4: Error Analysis: Fast R-CNN vs. YOLO These charts show the percentage of localization and background errors in the top N detections for various categories (N = # objects in that category).

YOLO(You Only Look Once)

Advantages
1) 매우 빠른 속도 (Regression problem)
2) Image를 전역적으로 파악 → Contextual information
3) Generalizable representation 학습



	VOC 2007	Picasso		People-Art
	AP	AP	Best F_1	AP
YOLO	**59.2**	**53.3**	**0.590**	**45**
R-CNN	54.2	10.4	0.226	26
DPM	43.2	37.8	0.458	32
Poselets [2]	36.5	17.8	0.271	
D&T [4]	-	1.9	0.051	

YOLO(You Only Look Once)

Advantages
1) 매우 빠른 속도 (Regression problem)
2) Image를 전역적으로 파악 → Contextual information
3) Generalizable representation 학습

Limitations
1) 상대적으로 높은 localization error

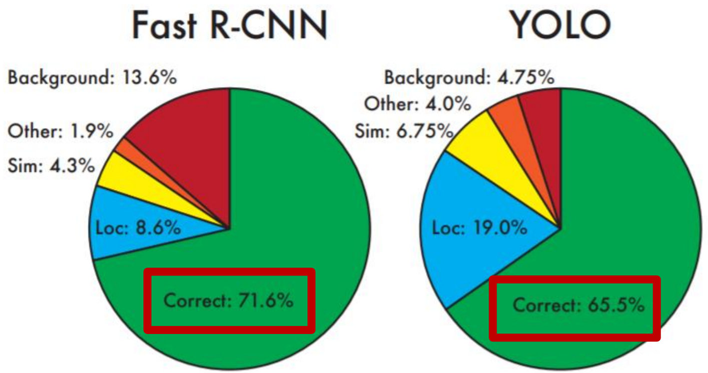


Figure 4: Error Analysis: Fast R-CNN vs. YOLO These charts show the percentage of localization and background errors in the top N detections for various categories (N = # objects in that category).

YOLO(You Only Look Once)

Advantages
1) 매우 빠른 속도 (Regression problem)
2) Image를 전역적으로 파악 → Contextual information
3) Generalizable representation 학습

Limitations
1) 상대적으로 높은 localization error

→ 속도와 정확도 간의 Trade-Off

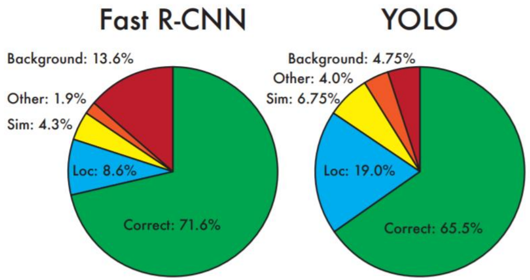


Figure 4: Error Analysis: Fast R-CNN vs. YOLO These charts show the percentage of localization and background errors in the top N detections for various categories (N = # objects in that category).

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [38]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

KU-BIG

YOLO v1 Architecture

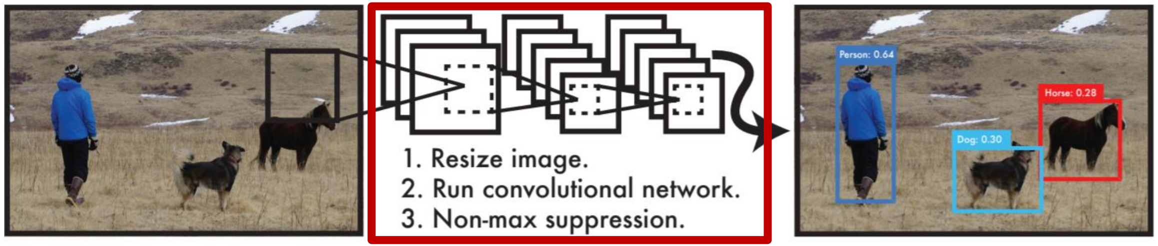
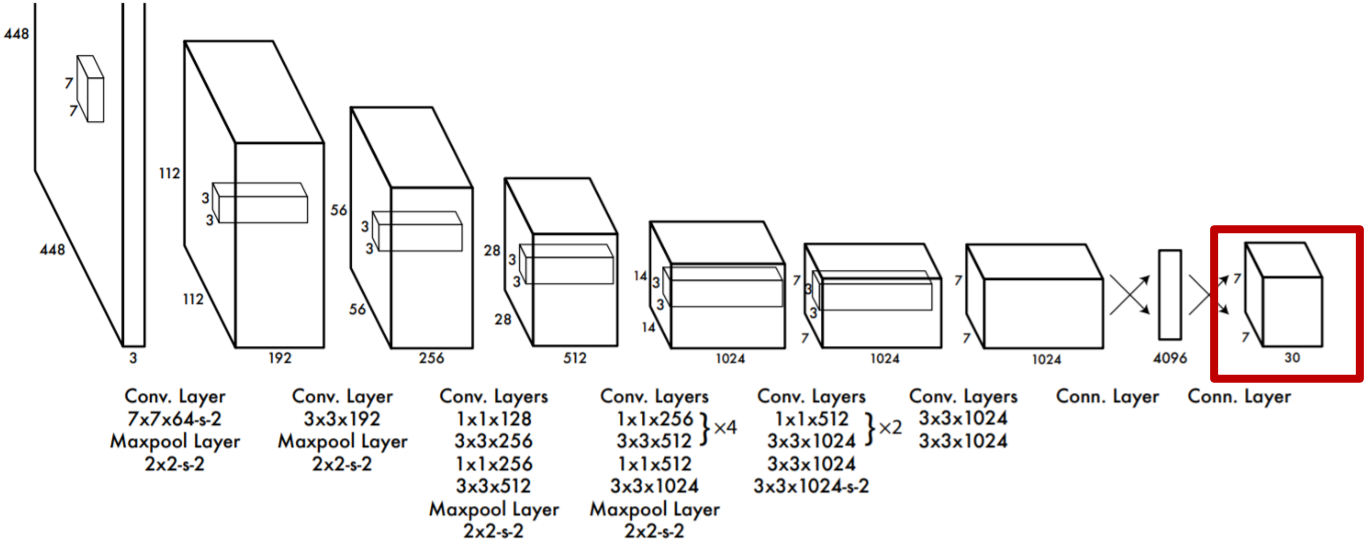
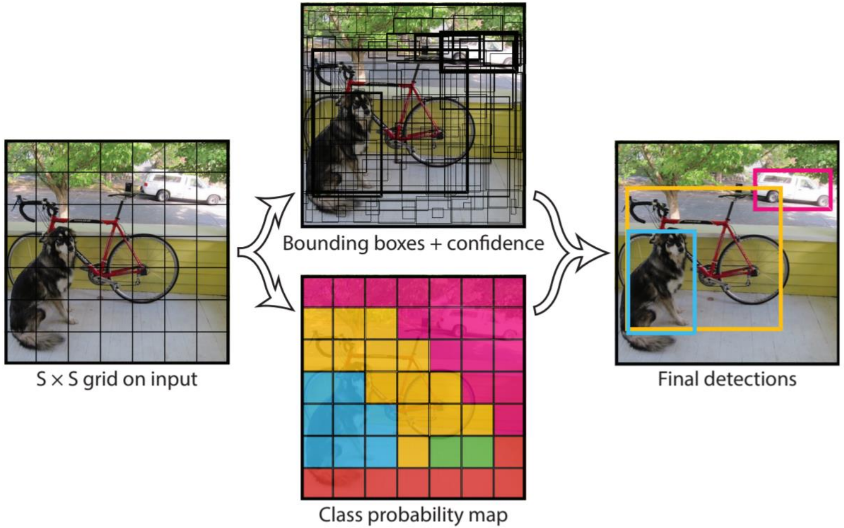


Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448×448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

YOLO v1 Architecture



YOLO v1 Architecture



S * S * (5 * B + C)

S = 7
B = 2
C = 20

$$Confidence = \Pr(Object) * IOU_{pred}^{truth}$$

(x, y, w, h, confidence) * B + C

➔ $\{objectness, x_1, y_1, w_1, h_1, x_2, \ldots h_2, C1, \ldots Cn\ \}$

YOLO v1 Loss functions

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

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

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

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

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

x_i, y_i, w_i, h_i

$= normalized\ x_{center}, y_{center}, w, h\ of\ i^{th} bbox$

$C_i, p_i(c) = \ Confidence, class\ probabilities$

$\lambda_{coord} = 5$

$\lambda_{noobj} = .5$

$Confidence = \Pr(Object) * IOU_{pred}^{truth}$

YOLO v2(YOLO9000: Better, Faster, Stronger)

Object Detection dataset(100k w/ 0.1k classes)은

Classification dataset(1m w/ 100k classes)에 비하면 제한적

➔ Classification data를 이용해서 현재 detect system에 적용,

➔ 이를 학습시키는 joint training algorithm

KU-BIG

YOLO v2(YOLO9000: Better, Faster, Stronger)

	YOLO								YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓				
new network?					✓	✓	✓	✓	✓
dimension priors?						✓	✓	✓	✓
location prediction?						✓	✓	✓	✓
passthrough?							✓	✓	✓
multi-scale?								✓	✓
hi-res detector?									✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

YOLO v2(YOLO9000: Better, Faster, Stronger)

1. Batch Normalization : Dropout 대신 Batch Normalization ➔ Regularization 효과

2. High Resolution Classifier : input image size 224 x 224 ➔ 416 x 416 , mAP 4% 증가

3. Convolutional With Anchor Boxes :

1) bounding box들을 직접 regression ➔ Anchor box offset 예측

2) grid 7x7 ➔ 13x13, downsample factor = 32 : 큰 object의 center는 가운데 cell에 위치

3) 한 cell에 objectness score 예측 ➔ 각 bbox마다 objectness score 예측

4. Dimension Clusters : k-means clustering을 통해서 training data에서 제일 적합한 anchor box 후보군 찾기

5. Direct location prediction : 1) offset 사용 2) grid cell의 위치에 상대적으로 anchor box를 predict → (x, y) in [0, 1]

6. Fine-Grained Features : concatenate(final layer 이전 layer의 feat map, final layer feat map) → 1% 성능 향상

7. Multi-Scale Training : iteration마다 model의 input 크기 다르게 ➔ {320, 352, …, 608} 임의의 image size 선택 후 학습

YOLO v2(YOLO9000: Better, Faster, Stronger)

Darknet-19

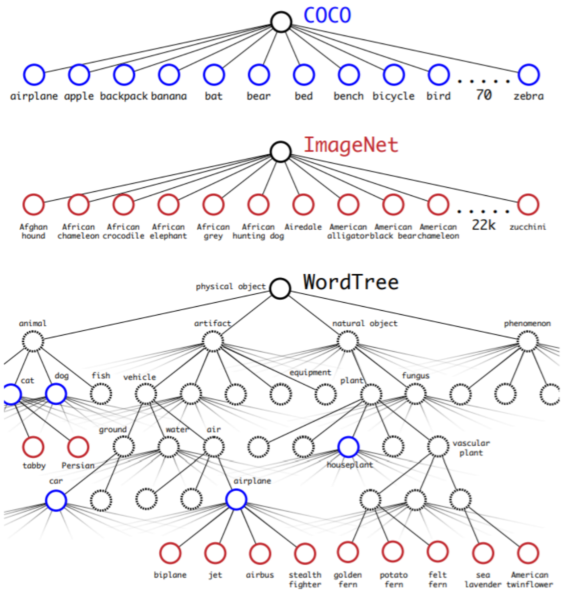
Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

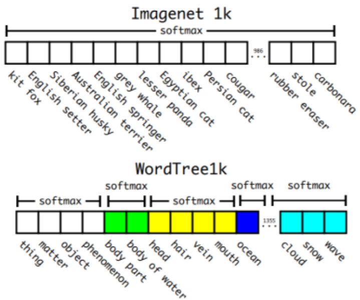
정확도가 높으면서 빠른 속도의 network

- ➔ 3x3 filter 사용
- ➔ 19개의 Conv layer
- ➔ 5개의 Pooling layer
- ➔ FC Layer 대신 1x1 Conv Layer

YOLO v2(YOLO9000: Better, Faster, Stronger)

Hierarchical Classification





ImageNet dataset : WordNet이라는 language dataset으로부터 파생 (계층적 data)

➔ WordNet 기반으로 계층적 tree를 구조화

➔ multi label 학습 진행

➔ Can detect **Wide Variety of Object Classes** in **Real-Time**

$$
\begin{aligned}
Pr(\text{Norfolk terrier}) = \; & Pr(\text{Norfolk terrier}|\text{terrier}) \\
& * Pr(\text{terrier}|\text{hunting dog}) \\
& * \ldots * \\
& * Pr(\text{mammal}|Pr(\text{animal}) \\
& * Pr(\text{animal}|\text{physical object})
\end{aligned}
$$

YOLO v3: An Incremental Improvement

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Table 1. **Darknet-53.**

FPN(feature pyramid network)와 같이
3개의 서로 다른 scale을 적용한 box를 predict

Output = {3d tensor encoding bounding box,
objectness,
class predictions}

➔ N x N x [3 * (4 + 1 + 80)]
4 : bounding box offsets
1 : objectness prediction
80 : class predictions

\+ 2 Layers previous ➔ Concatenation
→ Meaningful semantic information, finer-grained info.

Batch norm, offsets 등은 그대로 사용

Conclusion

- EfficientDet, YOLO v3
- 현재 YOLO v3를 기준으로 잡고 있으며
- 다른 모델들과의 비교 후 Hidden Soldiers Detector 모델 확정 예정

Efficientdet

YOLO v3

THANK YOU ☺