

Image Processing

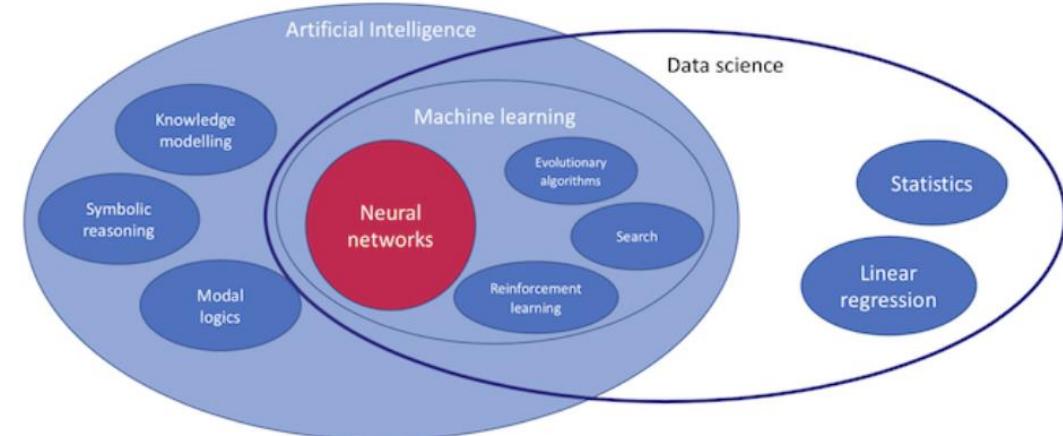
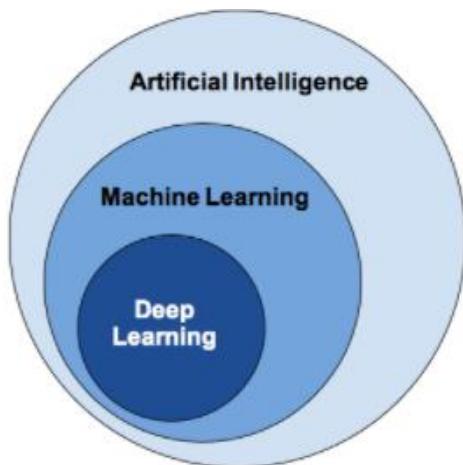
2024. 10

Yongjin Jeong, KwangWoon University

[참고] 본 자료에는 인터넷에서 다운받아 사용한 그림이나 수식들이 일부 있으니 다른 용도로 사용하거나 외부로 유출을 금해 주시기 바랍니다.

Machine Learning

- **What is ML**
 - the study of computer algorithms that improve automatically through experience and by the use of data. It is seen as a part of artificial intelligence. [wikipedia]
 - ML algorithms build a model based on sample data (training data) in order to make **predictions** or **decisions** without being explicitly programmed to do so.



[ref] <https://ictinstitute.nl/ai-machine-learning-and-neural-networks-explained/>

Datasets for Machine Learning

- **Iris** dataset
 - 4-column features, 150 samples
 - Target: 3 classes
- **MNIST**(Modified National Institute of Standards and Technology dataset)
 - 28*28 grayscale pixels, 60,000 train, 10,000 test set
 - Target: single digits 0 ~ 9
- **Fashion MNIST**
 - 28*28 pixels, 60,000 train set, 10,000 test set
 - Target:10 categories
- Many more in Kaggle.com



Iris Versicolor

Iris Setosa

Iris Virginica

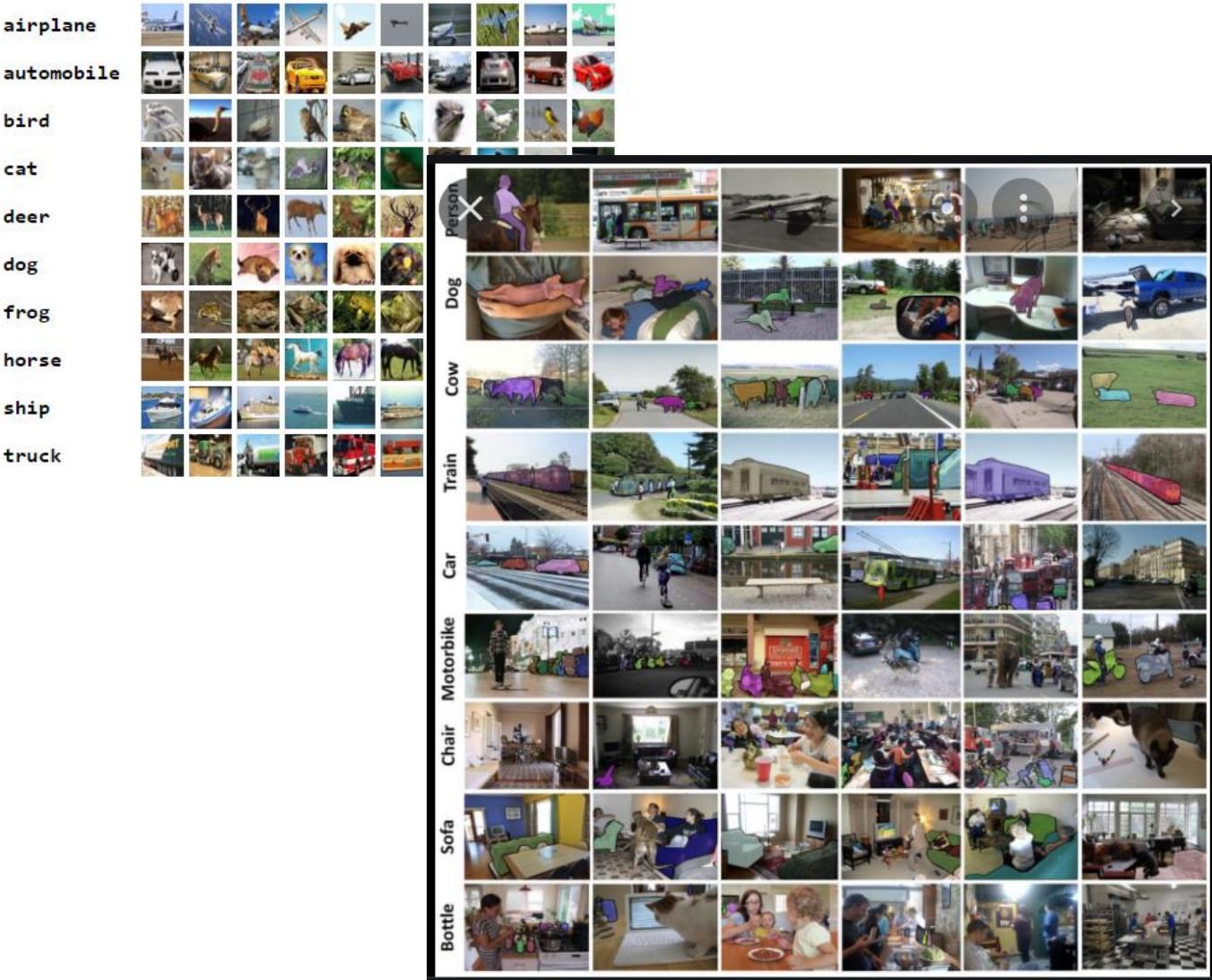
5	0	4	1	9	2	1	3	1	4
3	5	3	6	1	7	2	8	6	9
4	0	9	1	1	2	4	3	2	7
3	8	6	9	0	5	6	0	7	6
1	8	1	9	3	9	8	5	9	3
3	0	7	4	9	8	0	9	4	1
4	4	6	0	4	5	6	1	0	0
1	7	1	6	3	0	2	1	1	7
9	0	2	6	7	8	3	9	0	4
6	7	4	6	8	0	7	8	3	1



Fashion MNIST

Datasets for Machine Learning

- **Cifar-10 (Canadian institute for advanced research)**
 - 32*32 color images in 10 classes, with 6,000 per class
 - 50,000 training, and 10,000 test images
 - Bigger dataset (Cifar-100)
- **Microsoft COCO** (common objects in Context)
 - Large image recognition/classification, object detection, **segmentation**, and captioning dataset
 - 330K images (200K+ annotated), more than 2M instances in 80 object categories
 - 5 captions per image, and 250,000 people with key points



Datasets for Machine Learning

- **ImageNet**

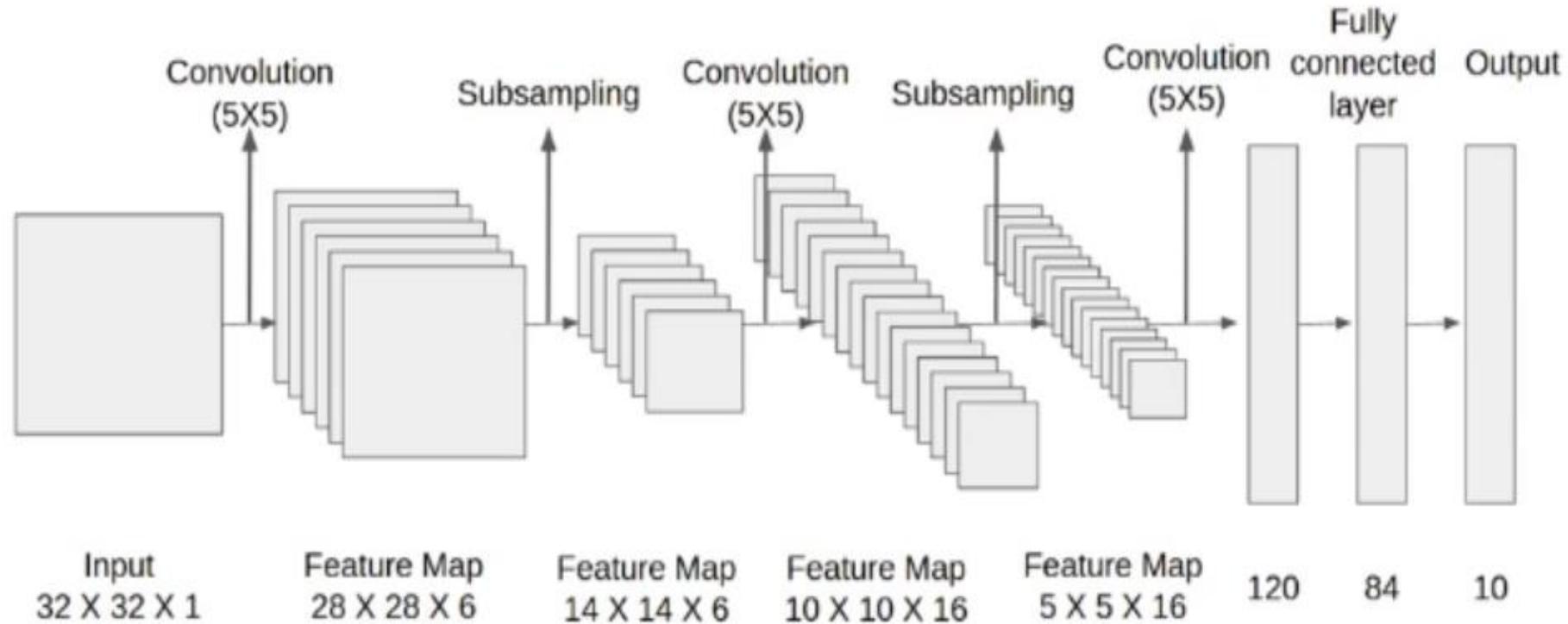
- largest image dataset for computer vision, used in [ILSVRC](#)(ImageNet Large Scale Visual Recognition Challenge) for image classification and object detection (150 GB)
- 469*387 resolution in average (usually cropped to 256*256 or 224*224 before usage)
- More than 14 million images with more than 21,000 groups(classes)
- More than 1 million images have bounding box annotations

- **ILSVRC**

- To evaluate algorithms for object detection and image classification (and localization) at large scale
- To measure the progress of computer vision for large scale image indexing for retrieval and annotation
- Uses smaller portion of the ImageNet (**1,000 categories with 1.3 million train images**, 50,000 validation images, and 100,000 test images)
- Available in Kaggle
- 2010 ~ 2017



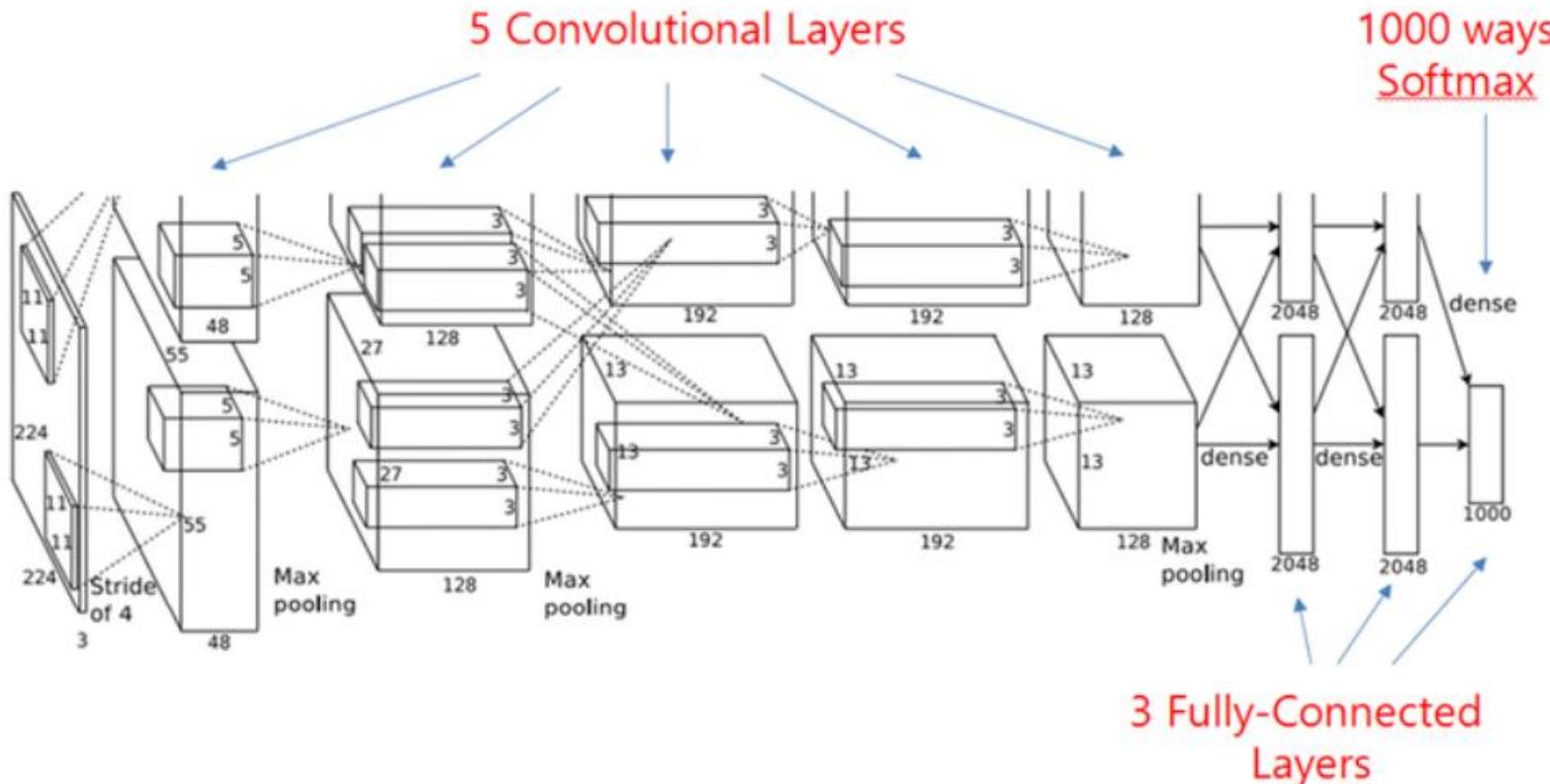
Lenet-5 (1998, LeCun et al.)



- First architecture for CNN (excellent on MNIST dataset)
- Small and easy to understand
- Uses **tanh** activation function

ILSVRC Winners

- **ALexNet – 2012 Winner (top-5 error rate 15.3%)**
 - Use CNN (prior to 2012, the classification model error was 25%)
 - Regarded as a Pioneer of CNN and starting point of the Deep learning
 - 60 million parameters
 - Used ReLU activation function, data augmentation, dropout, and overlapped pooling layers
 - Similar to LeNet-5, but Uses 2 GPUs in Parallel

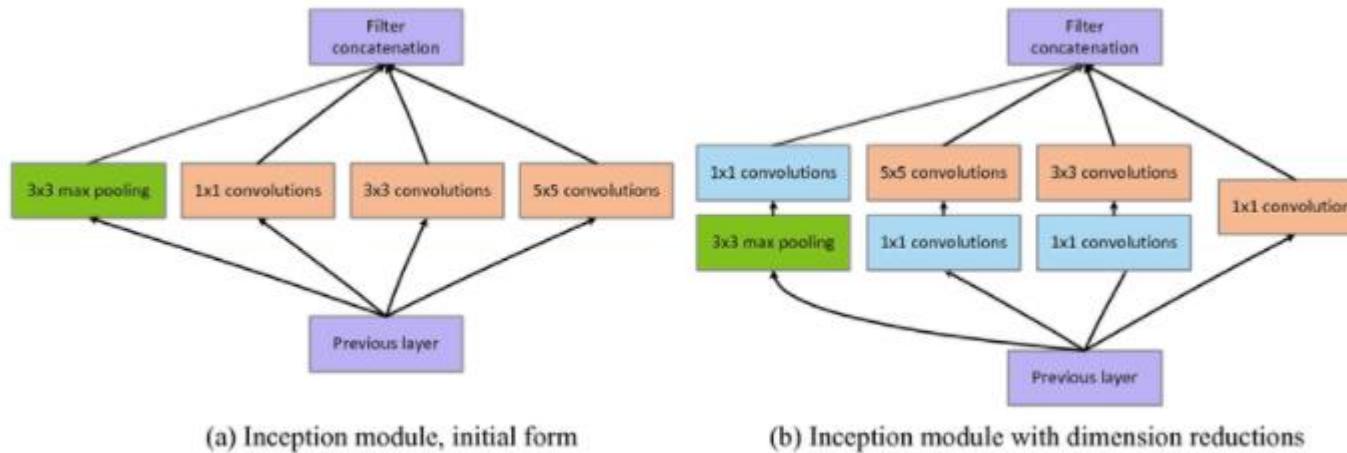


(*) top-5 error: The model is considered to have classified a given image correctly if the target label is one of the model's top 5 predictions. (image classification)

ILSVRC Winners

- **Inception V1 (GoogleNet) – 2014 Winner (top-5 error rate 6.67%)**

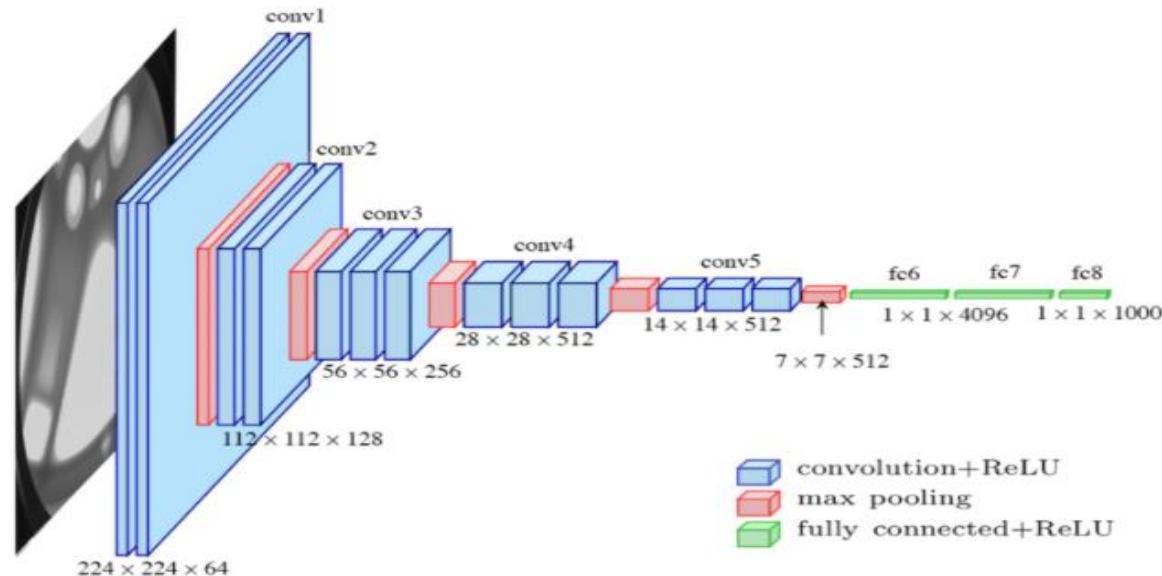
- The v1 stands for 1st version and later there were further versions v2, v3, etc. It is also popularly known as GoogLeNet.
- Deep with 22 layers.
- Used multiple types of filter size, instead of being restricted to a single filter size, in a single image block, which we then concatenate and pass onto the next layer.
- Used 1x1 convolutional with ReLU to reduce dimensions and number of operations.



ImageNet Challenge (2014)- Inception-V1 (GoogLeNet) [Source](#)

ILSVRC Winners

- **VGG-16 (University of Oxford) – 2014 Runners-Up (top-5 error rate 7.3%)**
 - Despite not winning the competition, VGG-16 architecture was appreciated and went on to become one of the most popular image classification models.
 - instead of using large-sized filters like AlexNet, it uses several 3×3 kernel-sized filters consecutively. The hidden layers of the network leverage ReLU activation functions.
 - VGG-16 is however very **slow to train** and the network weights, when saved on disk, occupy a **large** space.

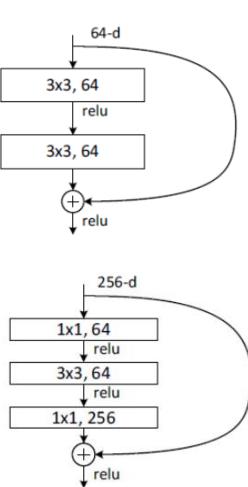


ImageNet Challenge (2014) – VGG-16 ([Source](#))

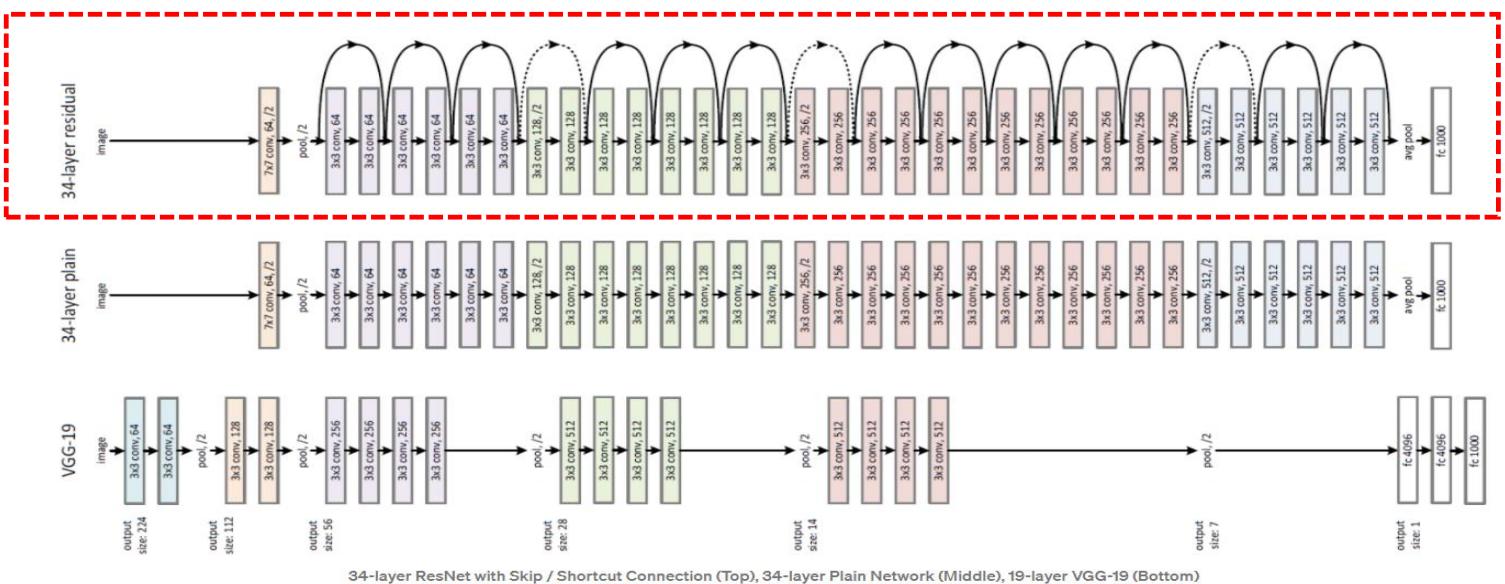
ILSVRC Winners

- **ResNet – 2015 Winner (top-5 error rate 3.57%)**

- ResNet ([Residual Network](#)) was created by the Microsoft Research team.
- To solve the problem of vanishing/exploding gradients, a [skip/shortcut connection](#) is added to add the input x to the output after few weight layers as below: (입력과 출력의 차이를 학습하여 작은 변화에 대해 민감하게 반응할 수 있도록 훈련)
- 1×1 Conv can reduce the number of connections (parameters) while not degrading the performance of the network so much. (as in Inception-V1)
- ResNet-18/34/50/101/152 has 1.8/3.6/3.8/7.6/11.3 GFLOPs (lower than VGG-16/19 with 15.3/19.6 GFLOPS)



The Basic block (top) and the Proposed Bottleneck design (bottom)

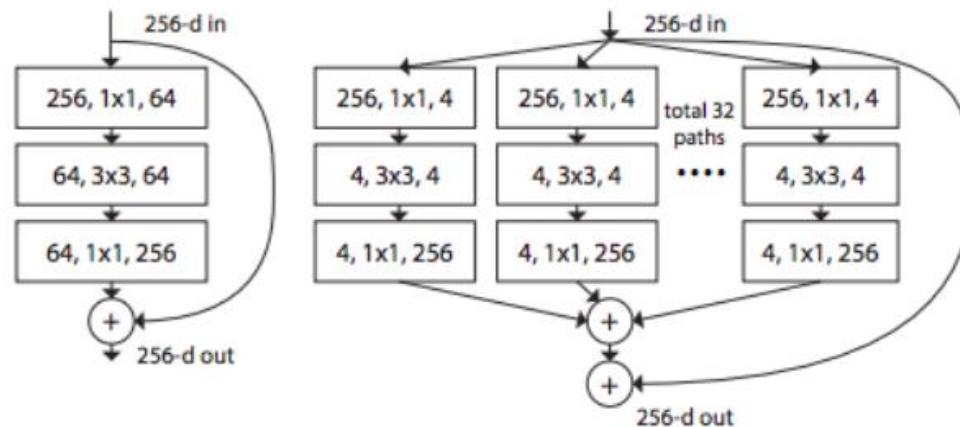


(*) VGG-19 (bottom): state-of-the-art approach in 2014
middle: deeper network of VGG-19 (i.e. more Conv layers)

ILSVRC Winners

- **ResNeXt – 2016 Runners-Up (top-5 error rate 4.1%)**

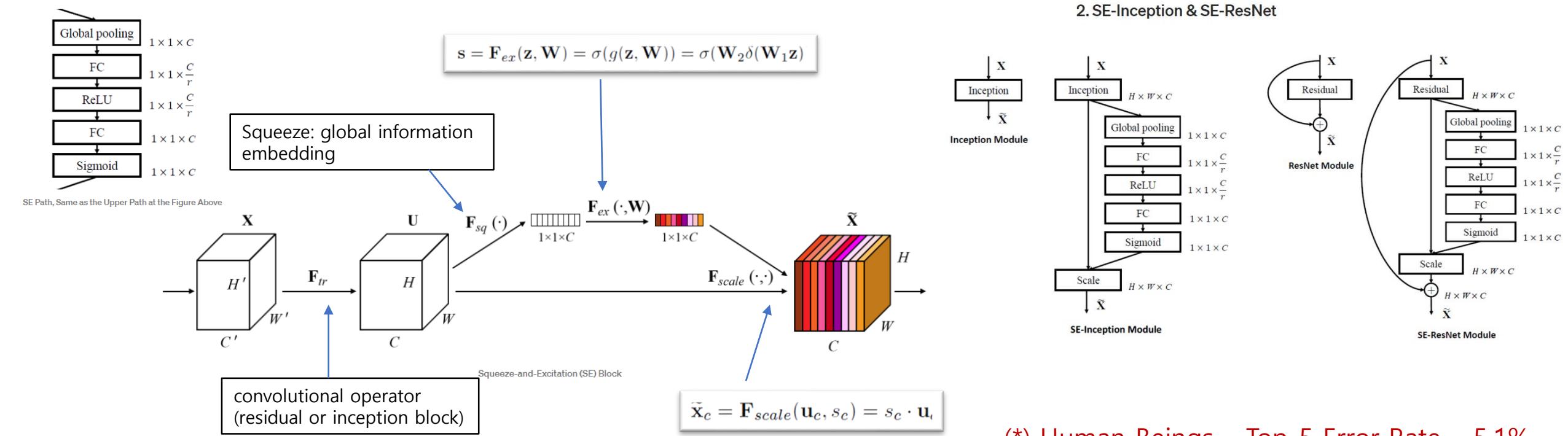
- Developed in the collaboration of the Researchers from UC San Diego and Facebook [AI](#) Research.
- Inspired by (ResNet + VGG + Inception).
- Still it became a popular model.
- stacks the blocks and then use the ResNet approach of residual blocks. Here the hyper-parameters such as width and filters were also shared.



ImageNet Challenge (2016)- ResNeXt ([Source](#))

ILSVRC Winners

- **SENet(Squeeze-and-Excitation Network) – 2017 Winner (top-5 error rate 2.251%)**
 - Developed in University of Oxford.
 - With “Squeeze-and-Excitation” (SE) block that **adaptively recalibrates channel-wise feature responses by explicitly modelling interdependencies between channels**, SENet is constructed.
 - SE block can be added to both Inception and [ResNet](#) block easily as **SE-Inception** and **SE-ResNet**. (achieved the best 2.25%).

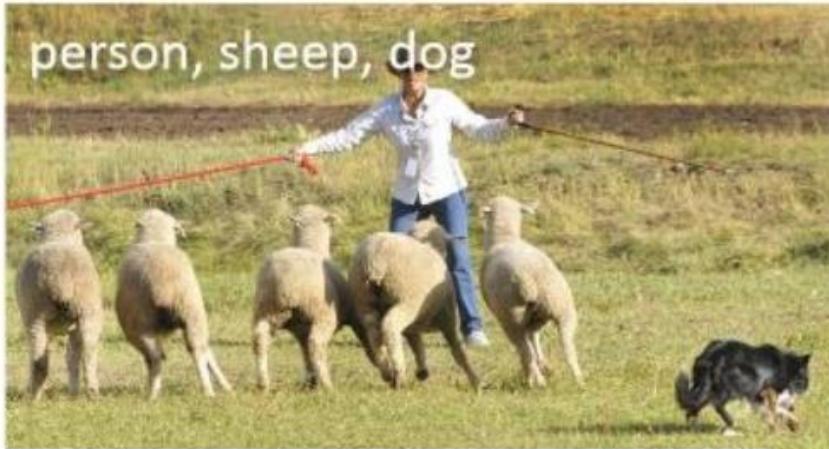


(*) Human Beings – Top-5 Error Rate – 5.1%

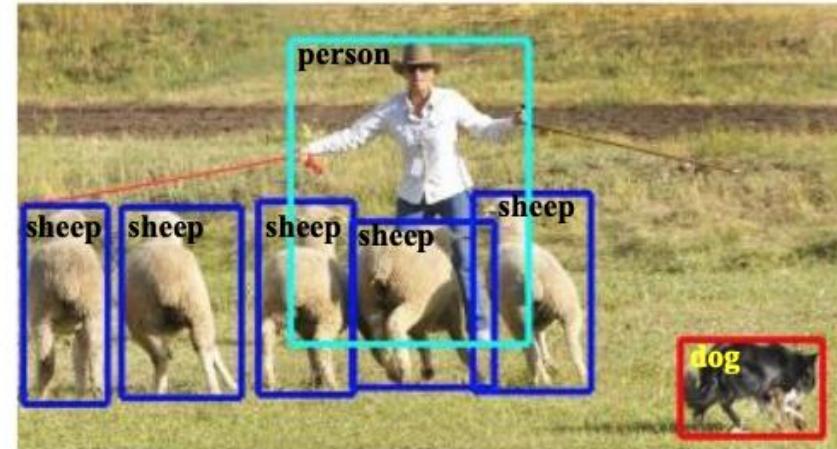
Computer Vision Tasks

- **Image Classification** (이미지 분류)
 - 이미지에 어떤 객체들이 들어 있는지만 알아내는 것
 - 여기서는 이미지에 사람, 양, 개가 있다는 것을 찾아낸다
- **Object Detection** (객체 검출)
 - 찾아낸 객체의 위치까지 알아내는 것
 - 일반적으로는 객체가 있는 위치를 박스 형태로 찾아낸다
- **Segmentation** (세그멘데이션 or 분할)
 - 객체의 위치를 박스형태가 아니라 비트 단위로 찾아낸다
 - 즉, 각 비트가 어떤 객체에 속하는지를 분류해내는 것
- **Object Instance Segmentation** (객체 인스턴스 세그멘테이션)
 - 이미지를 비트 단위로 어느 객체에 속하는지를 찾아낼 뿐 아니라 각 객체를 서로 다른 객체로 구분하는 기능까지 수행

Computer Vision Tasks



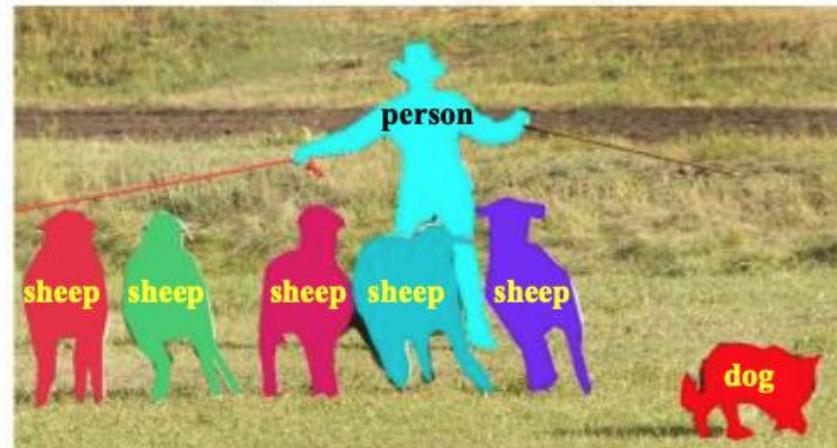
(a) Object Classification



(b) Generic Object Detection
(Bounding Box)



(c) Semantic Segmentation



(d) Object Instance Segmentation

Computer Vision Tasks (another view)

Classification



[This image is CC0 public domain](#)

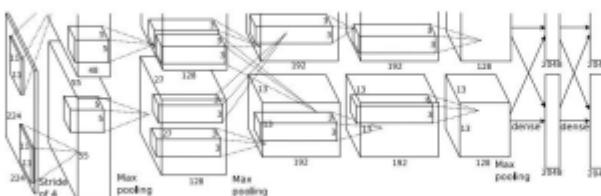


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Class Scores

Cat: 0.9
Dog: 0.05
Car: 0.01
...

Fully-Connected:
4096 to 1000

Vector:
4096

Semantic Segmentation



GRASS, CAT,
TREE, SKY

No objects, just pixels

Classification + Localization



CAT

Single Object

Object Detection



DOG, DOG, CAT

Instance Segmentation



DOG, DOG, CAT

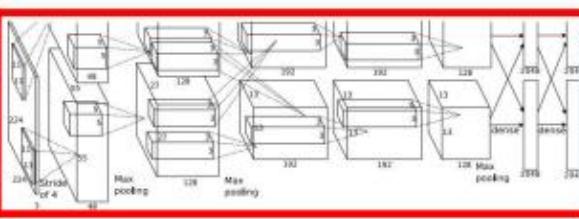
Multiple Object

Classification + Localization

Classification + Localization



This image is CC0 public domain



Often pretrained on ImageNet
(Transfer learning)

Multitask Loss

Treat localization as a
regression problem!

Fully
Connected:
4096 to 1000

Vector:
4096

Fully
Connected:
4096 to 4

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Box
Coordinates
 (x, y, w, h)

Correct label:
Cat

Softmax
Loss

+ → Loss

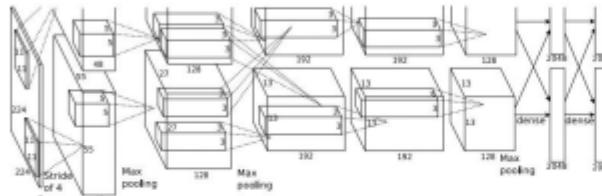
Correct box:
 (x', y', w', h')

Aside: Human Pose Estimation

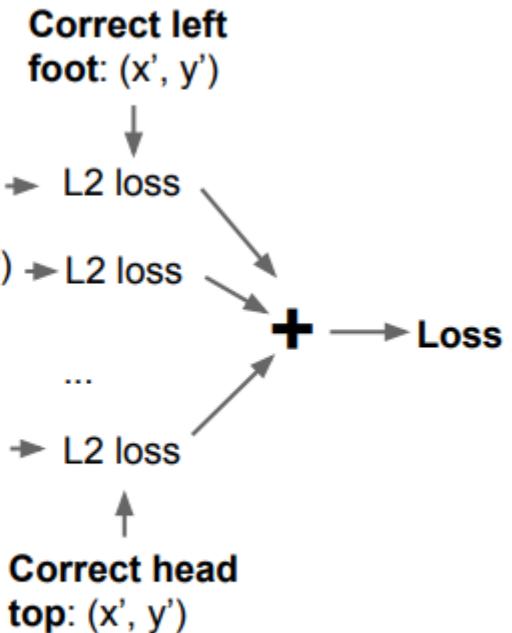


Represent pose as a set of 14 joint positions:

- Left / right foot
- Left / right knee
- Left / right hip
- Left / right shoulder
- Left / right elbow
- Left / right hand
- Neck
- Head top



Vector:
4096

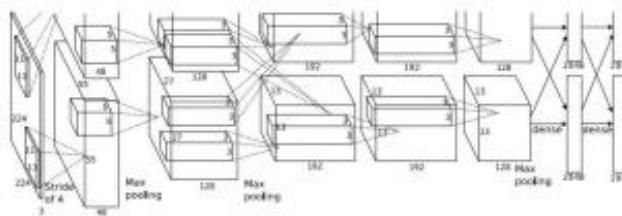


Toshev and Szegedy, "DeepPose: Human Pose Estimation via Deep Neural Networks", CVPR 2014

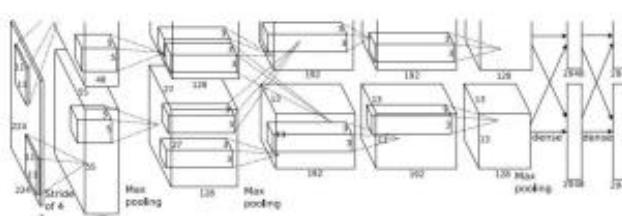
Object Detection (객체검출):Multi-objects

Object Detection as Regression?

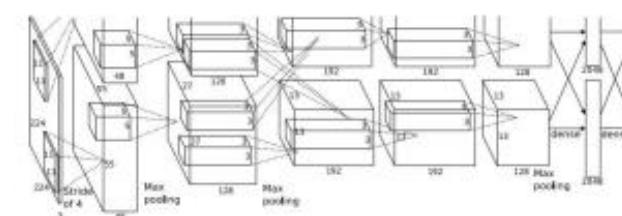
Each image needs a
different number of outputs!



CAT: (x, y, w, h) **4 numbers**



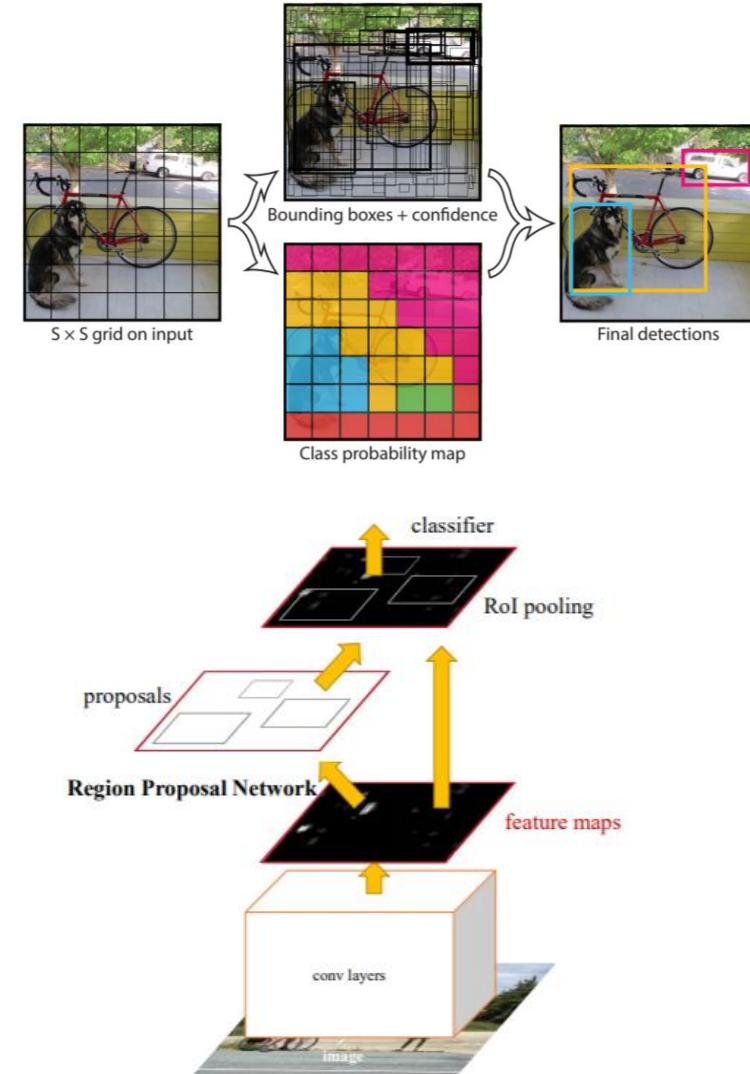
DOG: (x, y, w, h)
DOG: (x, y, w, h) **16 numbers**
CAT: (x, y, w, h)



DUCK: (x, y, w, h) **Many**
DUCK: (x, y, w, h) **numbers!**
....

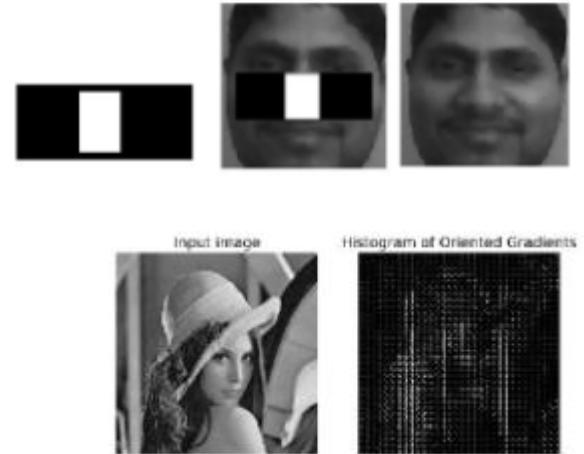
Object Detection

- **Object Detection:**
 - Localization + Classification
- **One-stage Detector**
 - 위의 두 과정을 동시에 진행
 - YOLO (You Only Look Once), SSD, RetinaNet
- **Two-stage Detector**
 - 위의 두 과정을 순차적으로 진행
 - R-CNN (Regions with CNN features), Faster R-CNN, DenseNet



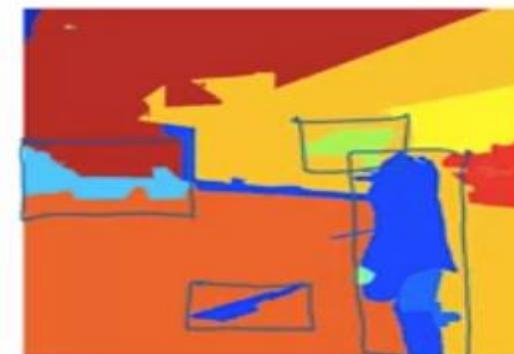
Object Detection

- **Early approaches:**
 - Viola & Jones '01:
 - Haar Features + Adaboost + cascading classifier for fast rejection
 - First competitive real-time object detection
 - Mainly used for face detection
 - Dalal & Triggs '05
 - Histogram of Oriented Gradients (HOG)
 - Support Vector Machines
- **Modern approaches:**
 - Sliding Window: classify each possible window by CNN
 - [Regional proposal](#) CNN (R-CNN): find interesting image regions first, then classify by CNN
 - Single-Shot Multi-box Detectors: joint detection and classification
 - You Only Look Once (YOLO)
 - Single-Shot Multi-box Detector (SSD)



Object Detection

- **슬라이딩 윈도우 방식 (초기)**
 - 객체 검출을 위해서 전체 이미지를 작은 크기로 나누고 각 부분을 대상으로 객체를 찾는 작업을 수행
 - 이 방식은 시간이 오래 걸리고 더욱이 윈도우 크기와 객체의 크기가 다를 때 찾지 못한다
- **Region Proposal (R-CNN에 적용)**
 - Sliding window 를 전체에 대해 하지 않고 물체가 있을 것 같은 영역에 대해서만 함. (스케일링 등이 필요 없어 빠름): segmentation 이용



Segmentation algorithm
≈ 2,000

Object Detection

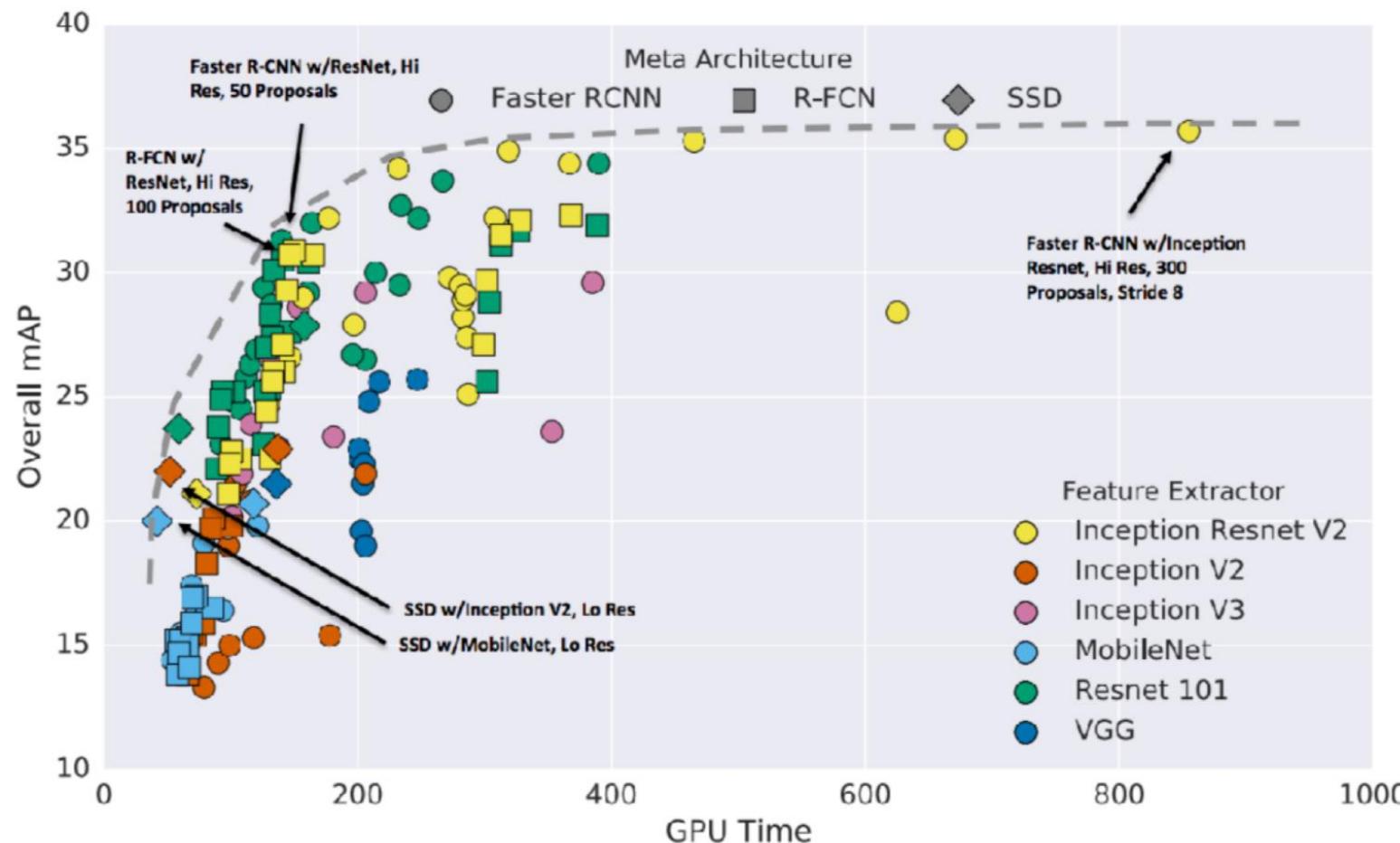
- 한편 객체의 위치를 찾는 문제
 - 분류가 아니라 회귀 문제
 - 즉, 객체의 경계 박스의 좌표 값은 회귀 문제로 예측
 - 이 방법은 사람의 자세를 예측하는 포즈 검출, 기준점 검출에서도 사용



(자세 검출의 예)

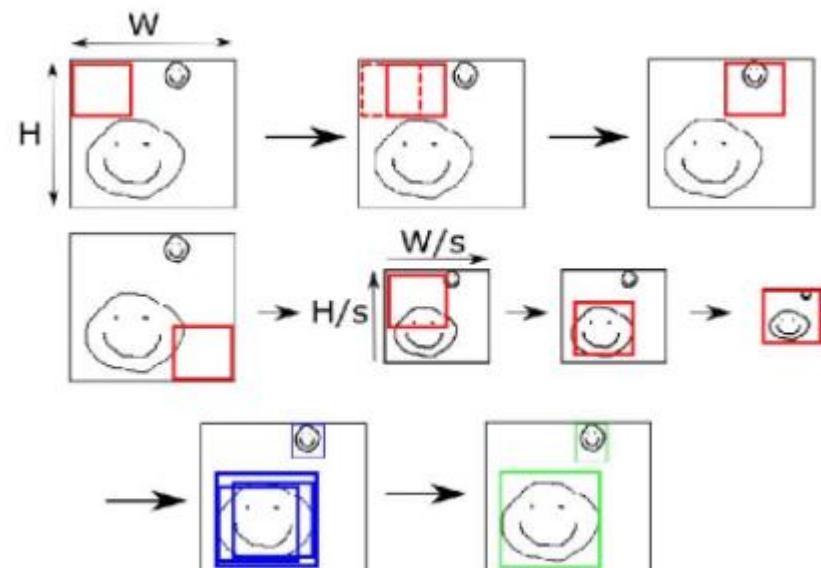
Object Detection - Algorithms

- Resnet을 사용한 Faster R-CNN등이 우수한 성능을 나타냈으나 최근 Mask R-CNN과 Yolo등 성능이 더 개선된 방식이 소개됨



Object Detection

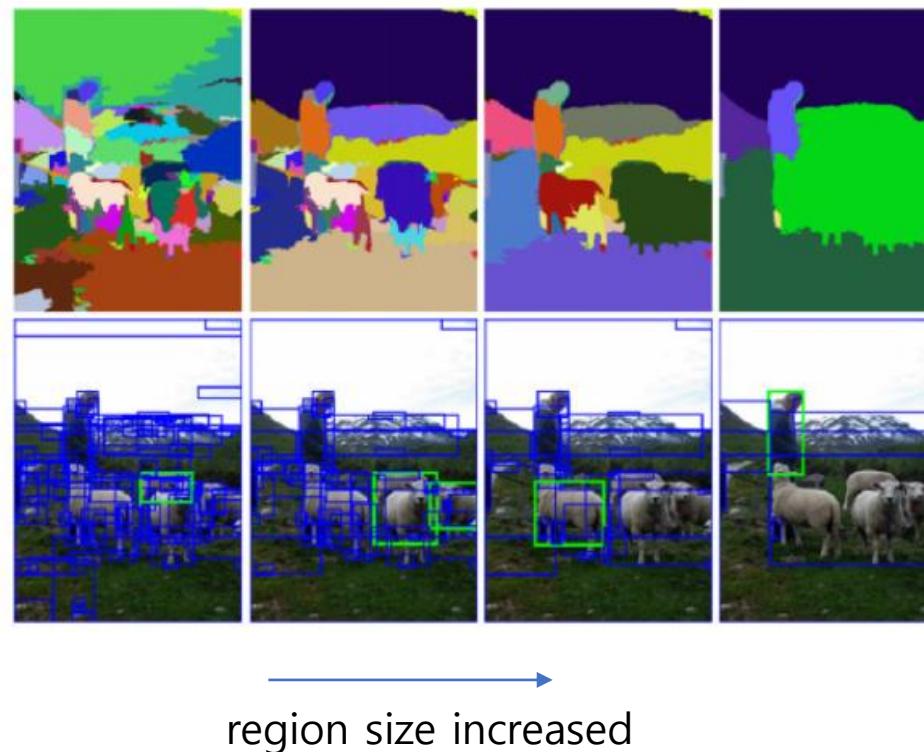
- **Single Window Approach (*Sliding Window*)**
 - Simply take your pre-trained CNN and just move it across the image
 - When you find an area of high confidence, you find it.
 - Repeat this process on multiple resolutions.
 - Multiple patches need be grouped to a single patch.
 - Disadvantage: Large number of patches -> computationally inefficient



Object Detection

- **Region Proposal: Selective Search (선택적 탐색) algorithm**

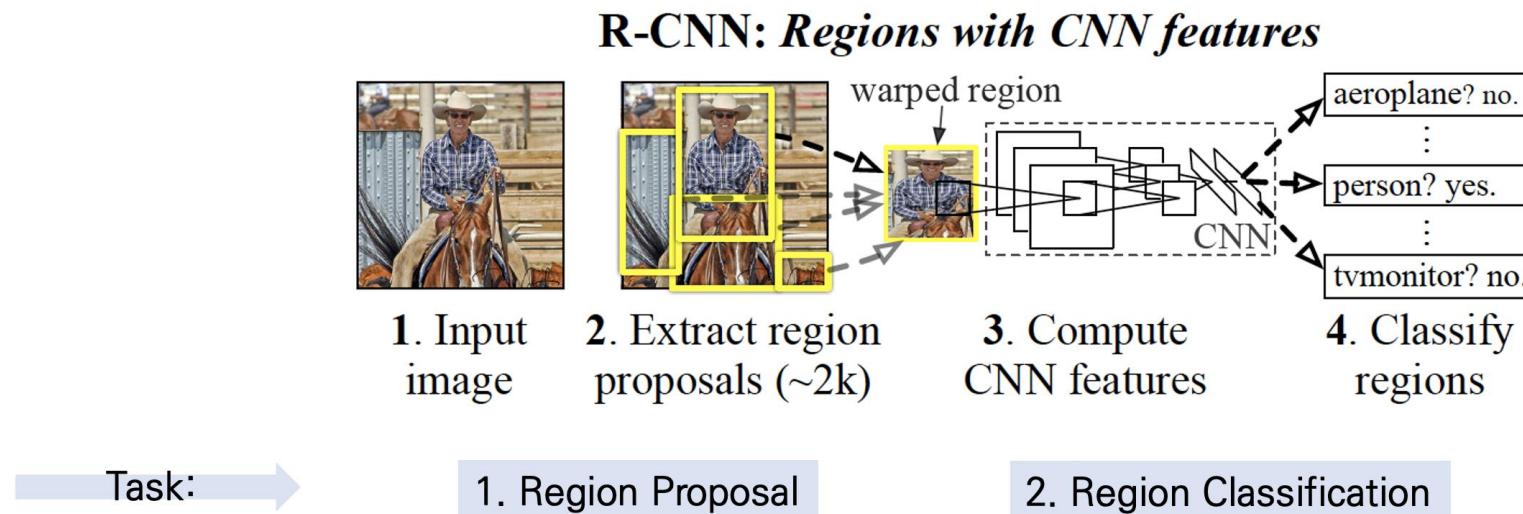
- Candidate objects by grouping pixels of similar texture or color
- Apply for different sized windows -> produce few thousand (~ 2k) object proposals per image (<< number possible windows)
- Essentially a form of coarse segmentation



Object Detection

- **Regional CNN (R-CNN)**

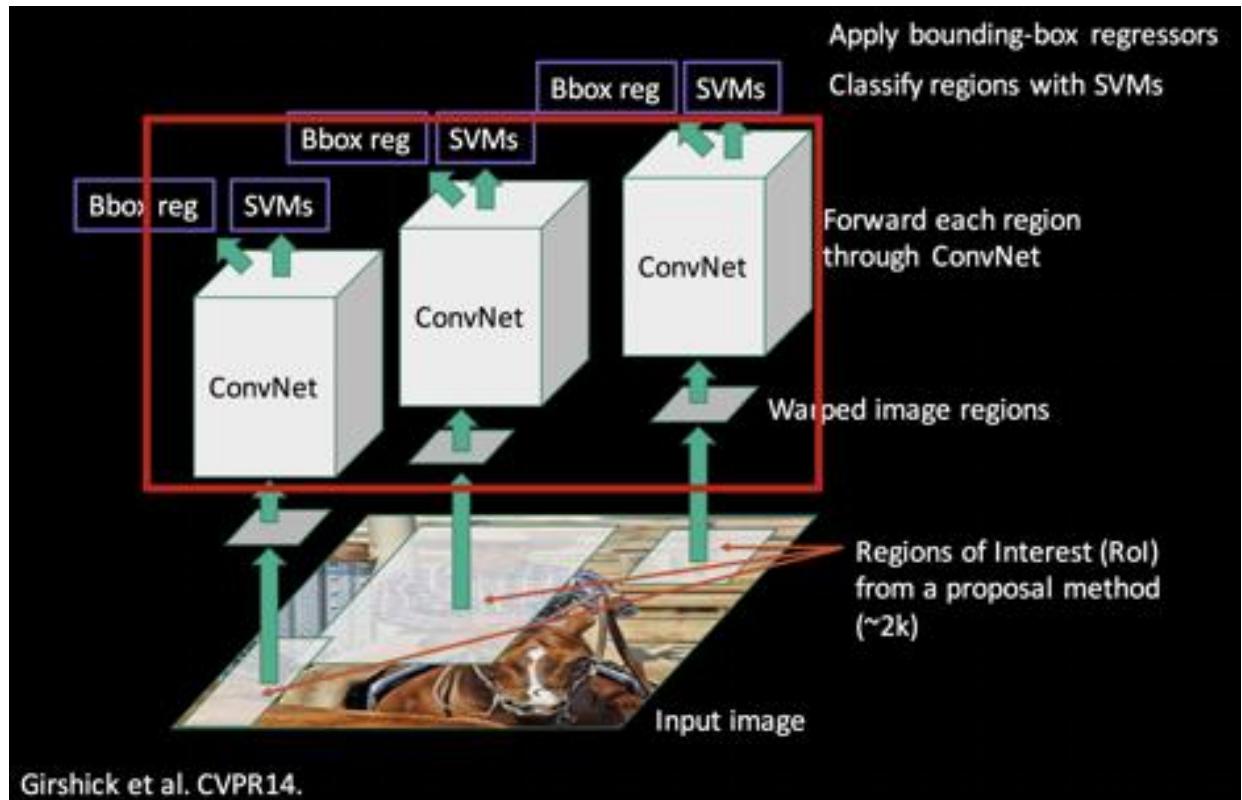
- CNN 모델을 Object Detection 에 최초로 적용시킨 Deep Learning 모델
- R-CNN (Regions with CNN features)
- 이미지 및 영상 내에 특정 객체가 있을 만한 영역에서 특징 추출(Feature Extraction) 성능이 뛰어난 CNN(합성곱신경망)을 적용시켜 객체를 탐지
- 즉, 이미지 내 특정 영역을 먼저 기준으로 잡고(=Localization), CNN 모델을 활용하여 객체를 분류 (=Classification)



Object Detection

- R-CNN modules

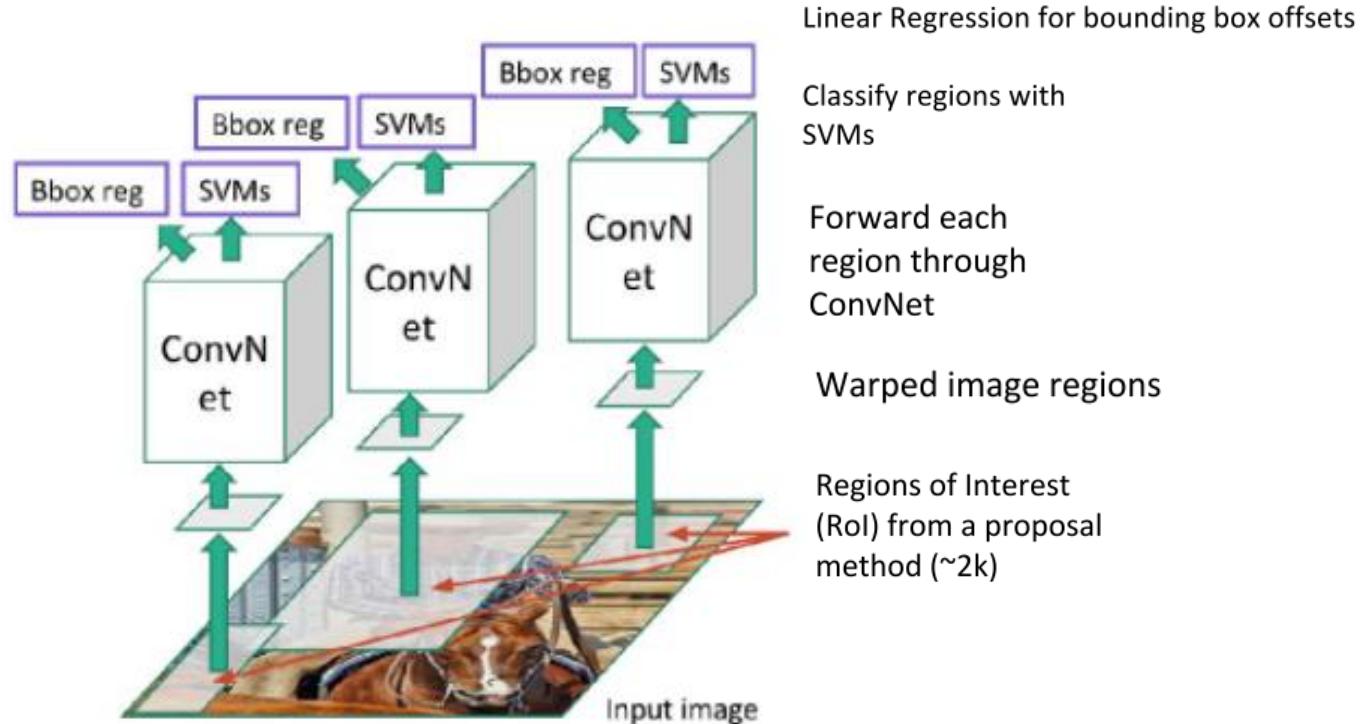
R-CNN 모델 구성 모듈	기능 및 설명
Region Proposal	카테고리와 무관하게 우선 객체가 있을 법한 영역(Region)을 찾는 모듈 – selective search
(사전 훈련된) CNN	찾은 영역(Region)에서 객체를 탐지하기 위해 특성 맵(Feature Map)을 추출하는 모듈
SVM	추출된 특성 맵을 분류하는 모듈 (객체를 식별하는 과정이다.)
Bounding Box Regression	회귀(Regression)를 활용해 해당 객체의 바운딩 박스를 표현하는 모듈



R-CNN

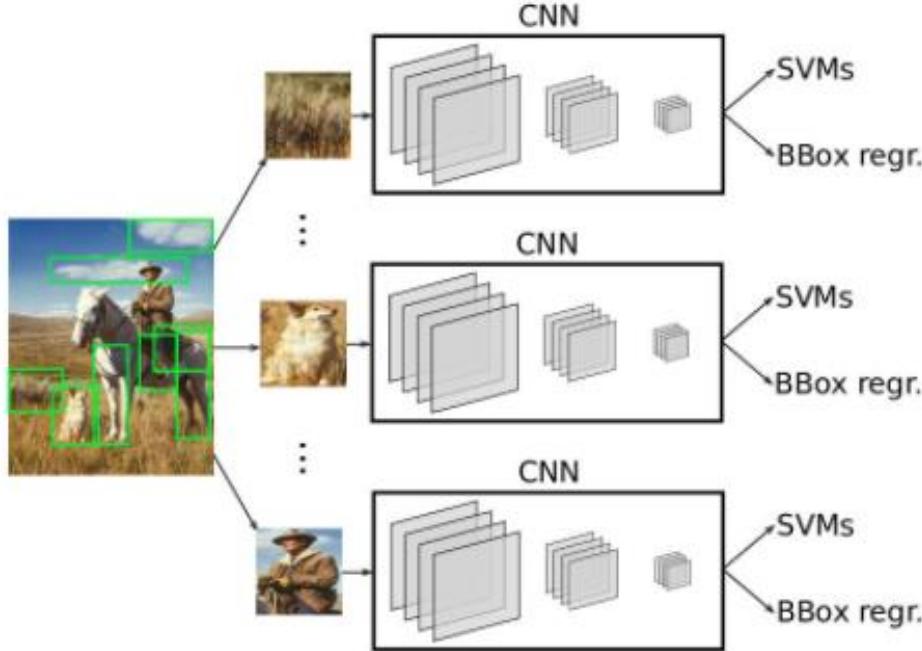
- **Regional CNN (R-CNN)**

- For each region proposal window
 - Warp to standard window size
 - Pre-trained CNN for feature extraction
 - Linear SVM for object classification
 - Linear regression for bounding box refinement
- + Improved retrieval rate at that time (2013) by more than 30%
- Much faster... but still slow
- Not end-to-end



Object Detection

- Problems in R-CNN

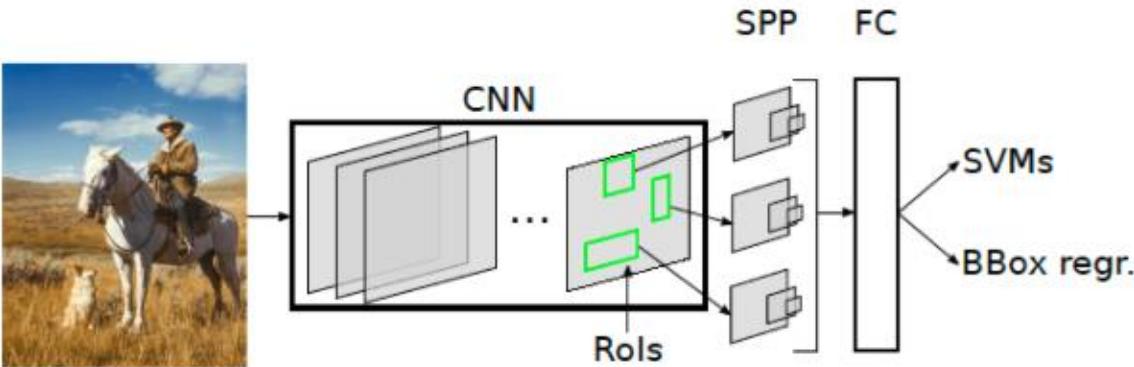


- Slow
- Then, the key idea in order to improve the inference speed is that we **use the region proposals on the feature maps**.

- R-CNN requires full forward pass for **each** region proposal
 - Training is slow
 - Inference is slow
- We can share computations when computing feature maps
- Why not use region proposals on feature maps?

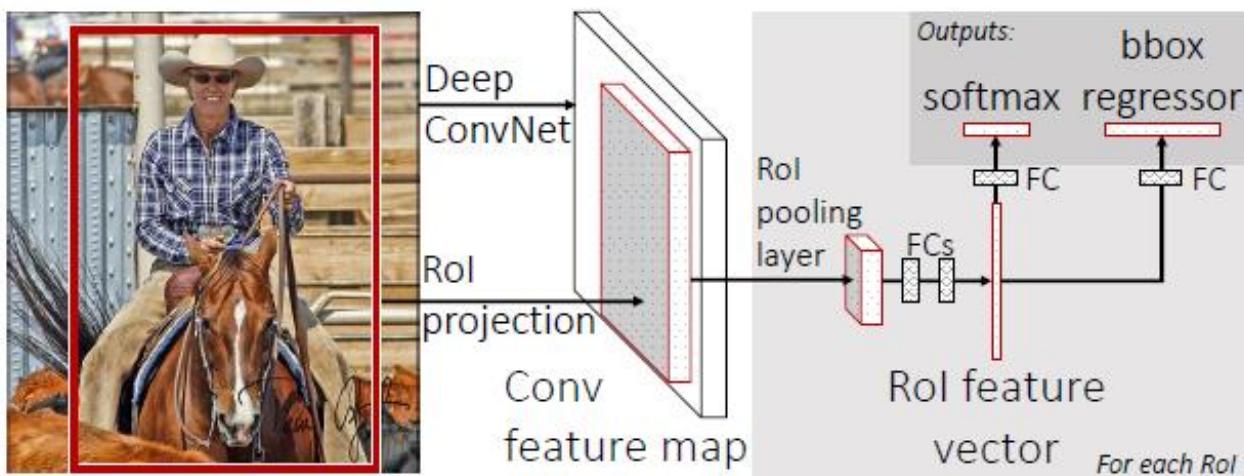
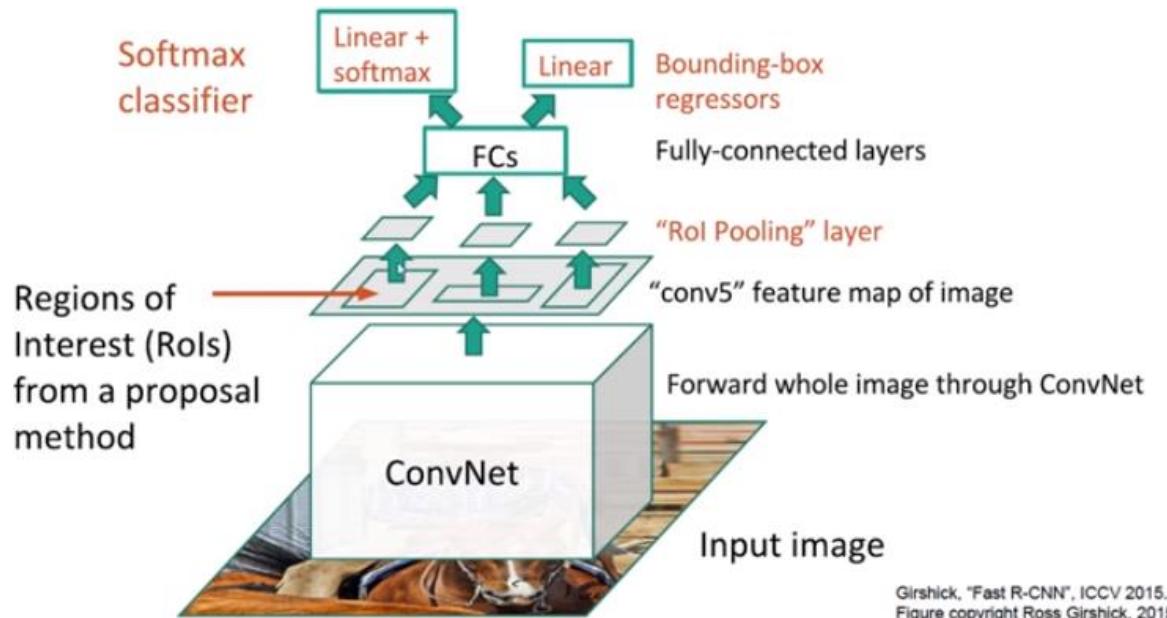
Fast R-CNN

- R-CNN 에서는 모든 Region Proposal (~2,000) 에 대해 CNN 수행하기 때문에 느림
- SPP(Spatial Pyramid Pooling)을 적용한 RoI Pooling 과정을 기존의 R-CNN 모델에 적용



- Pass full image through the network: Feature maps
- Apply region proposals to **last conv layer**
- Classification CNN has **fixed input size**
- **Spatial pyramid pooling** layer pools to fixed size using max-pooling
(orig. 3x w. different window size & stride)
- + Image-wise computation shared → Speed up by SPP during inference: $\approx 24\text{--}104\times$
- R-CNN problems: **slow training** / not end-to-end

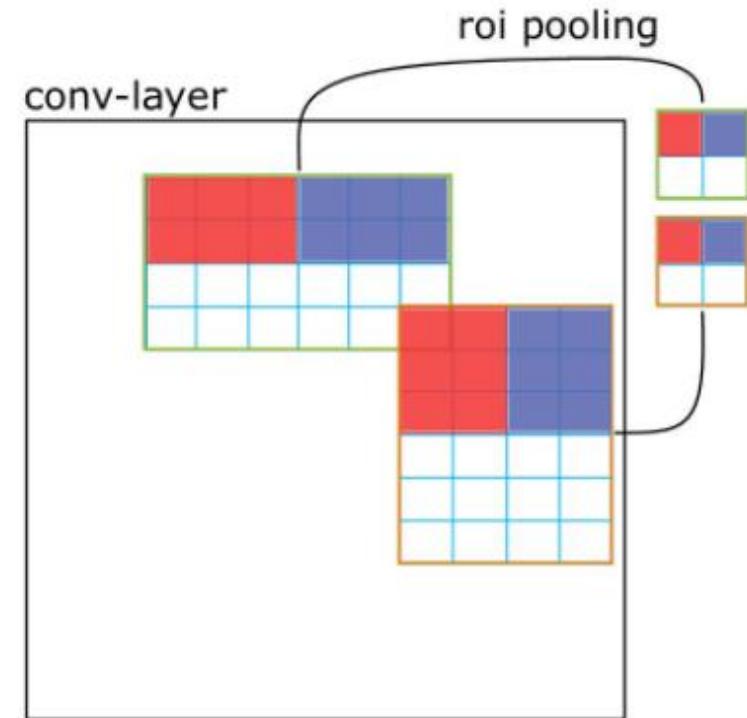
Fast R-CNN



Fast R-CNN

- **Roi Pooling**

- 마지막 부분의 Fully Connected Layer에 고정 입력 크기로 전달하기 위해 사용
- 미리 설정한 $H \times W$ 크기로 만들어주기 위해서 $(h/H) * (w/W)$ 과 같은 크기만큼 경계를 Roi 위에 생성
- 즉, 특성 맵은 결과적으로 $h \times w$ 크기의 Roi가 $H \times W$ 크기의 고정된 특성 벡터(Feature Vector)로 변환
- 이는 수많이 생성된 Region Proposal에 각각 CNN 연산이 필요한 것이 아닌, 단 1번의 CNN 연산만으로 R-CNN 동작을 가능하게 함 -> 스피드 향상



Faster R-CNN

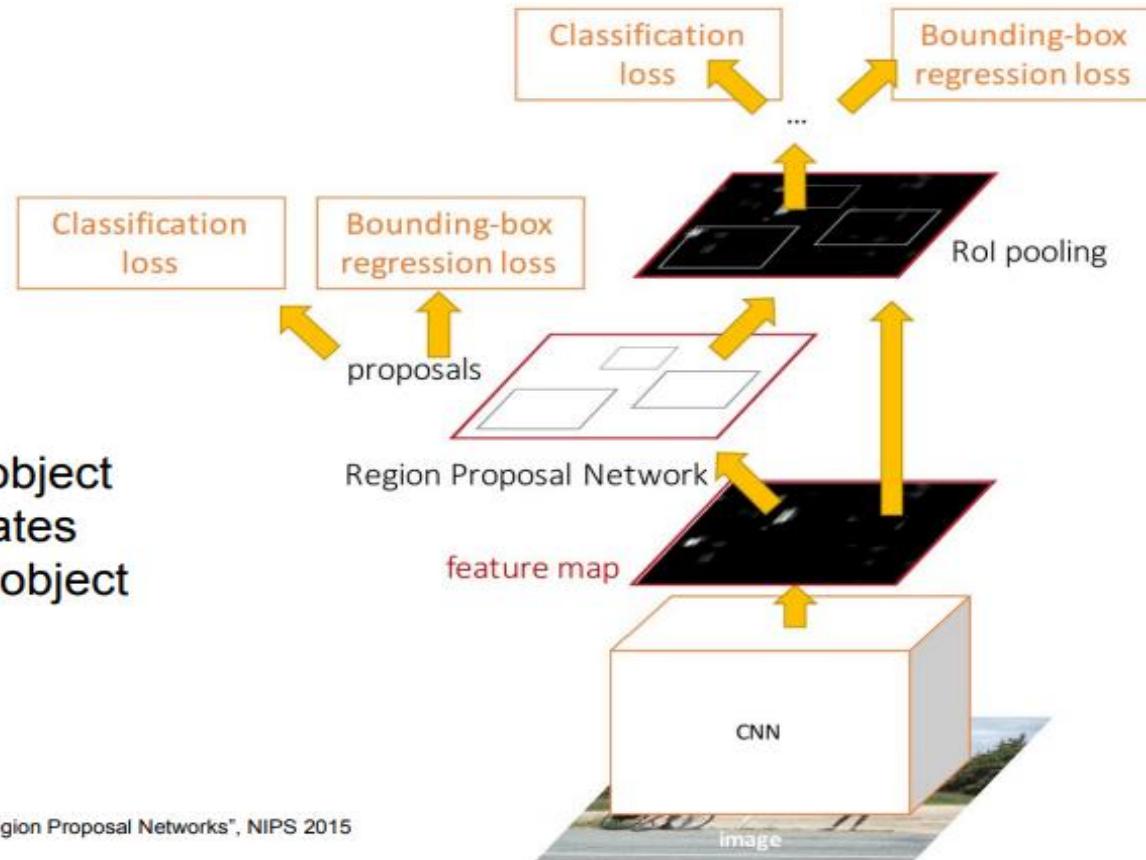
- Introduce RPN (region proposal network)
 - Perform region proposal directly into the architecture
 - Alleviating the need for Selective Search

Faster R-CNN:
Make CNN do proposals!

Insert **Region Proposal Network (RPN)** to predict proposals from features

Jointly train with 4 losses:

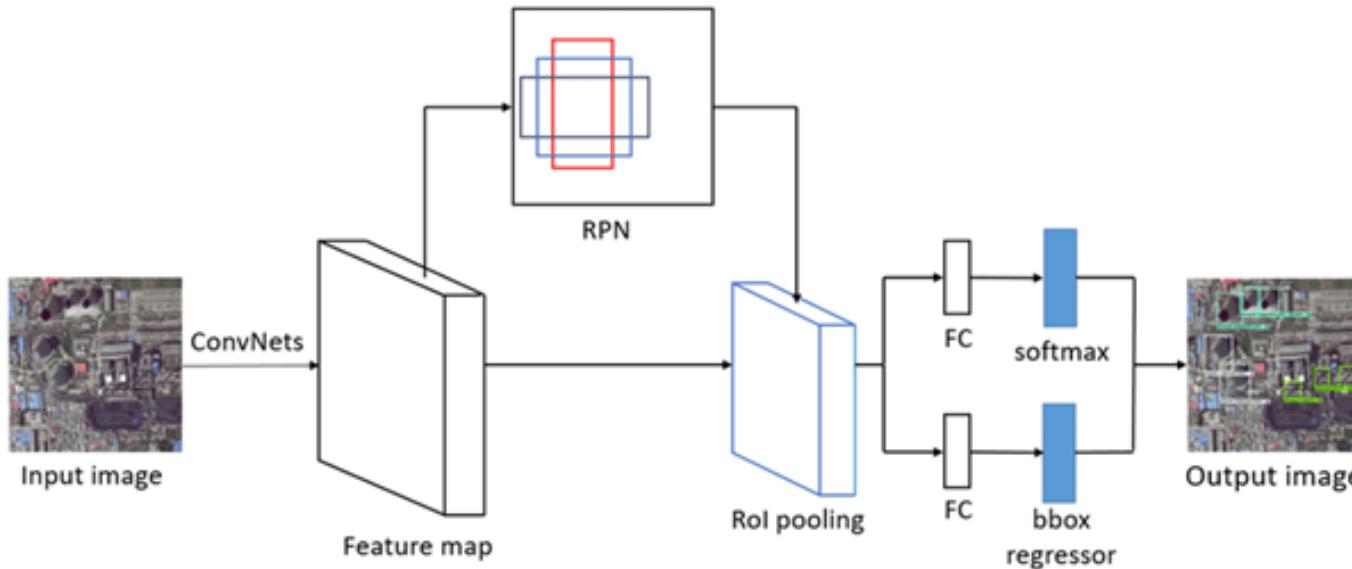
1. RPN classify object / not object
2. RPN regress box coordinates
3. Final classification score (object classes)
4. Final box coordinates



Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015
Figure copyright 2015, Ross Girshick; reproduced with permission

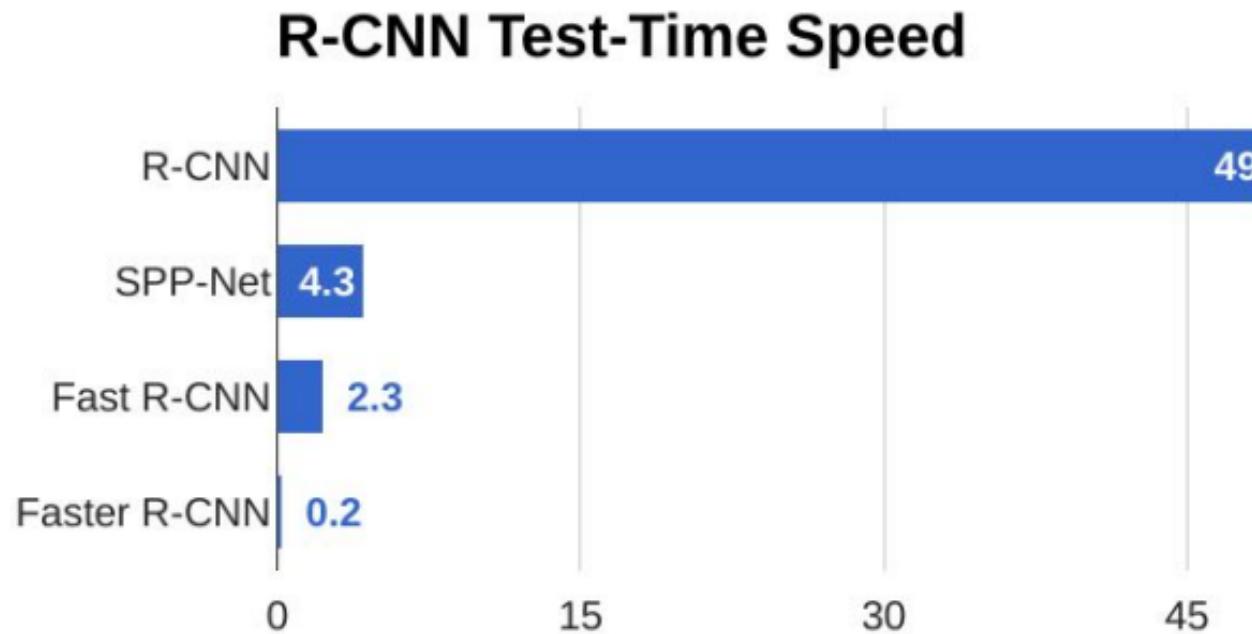
Faster R-CNN

- Faster R-CNN = RPN + Fast R-CNN 구조 (Faster R-CNN)
- Selective Search 대신 RPN 사용
- 기존의 선택적 탐색 알고리즘은 모델의 외부에서 동작하지만 RPN 은 모델 내부에서 함께 동작 -> 성능 향상



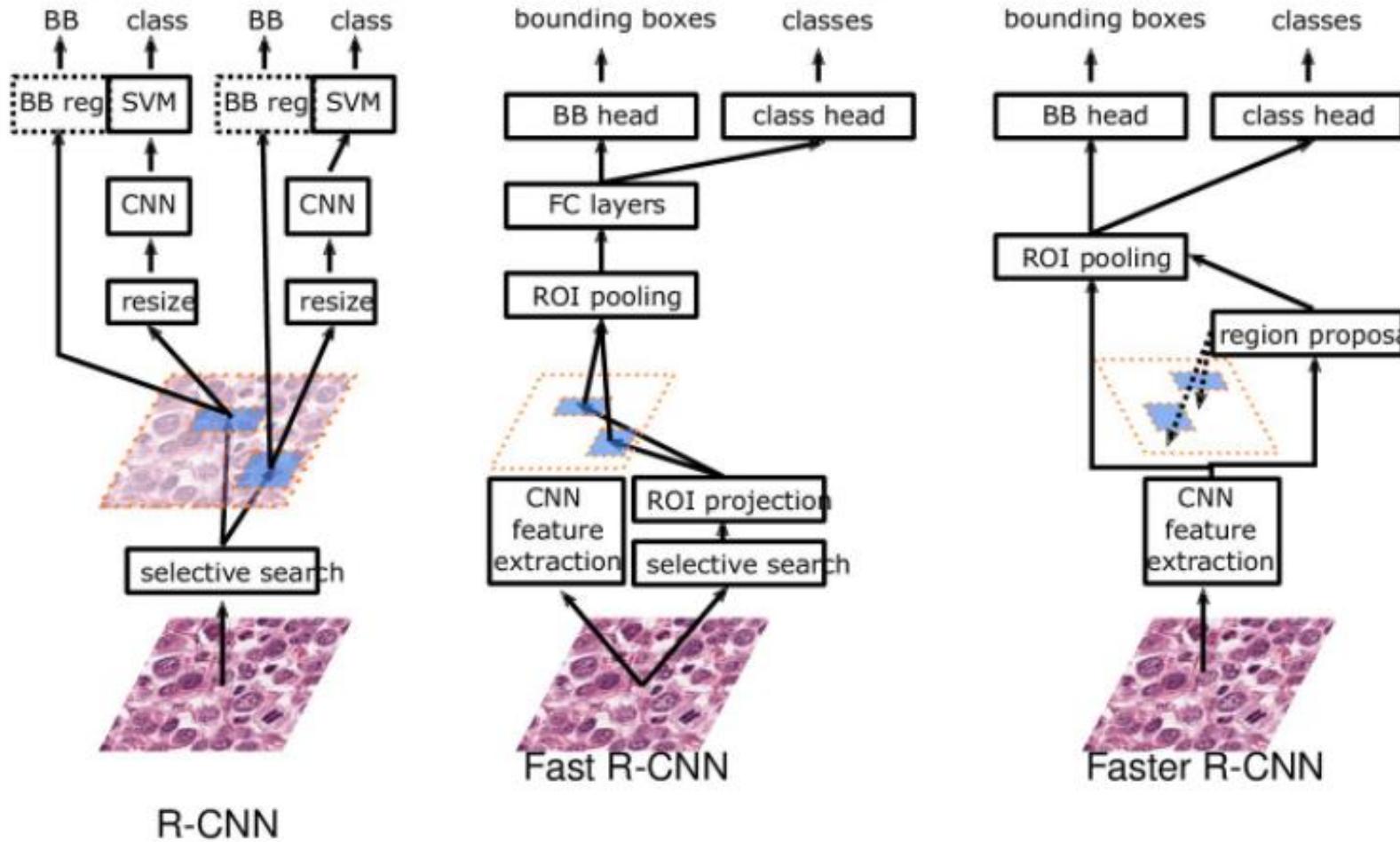
Faster R-CNN

- Make CNN do proposals!



Architectures based on R-CNN

Overview: Architectures based on R-CNN



Architectures based on R-CNN

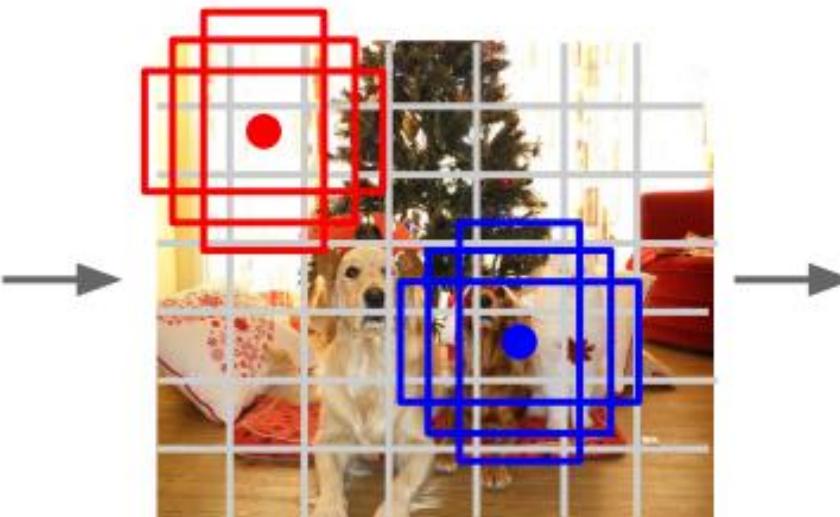
- **R-CNN 원리**
 - (1) 이미지 내에 객체가 존재하는 적절한 위치를 제안 (Selective Search)
 - (2) 제안된 위치의 이미지를 잘라냄
 - (3) 잘라낸 이미지의 특징맵 (feature map)를 추출 – CNN
 - (4) 특징 지도를 분류기에 입력하여 분류 - CNN
- **Fast R-CNN**
 - (3)(4) 의 과정 개선
 - 개별 ROI 대신 전체에 CNN 적용 (1 번의 CNN 연산)
 - ROI Pooling 레이어 사용 (fixed size ROI): Classifier 모델이 간단해짐
- **Faster R-CNN**
 - (1) 번 과정 개선
 - RPN(Region Proposal Network) 사용하여 Region Proposal 추출
 - 개별 ROI 대신 전체에 CNN 적용 (1 번의 CNN 연산)
 - ROI Pooling 레이어 사용 (fixed size ROI): Classifier 모델이 간단

Detection without Proposals: YOLO / SSD

Go from input image to tensor of scores with one big convolutional network!



Input image
 $3 \times H \times W$



Divide image into grid
 7×7

Image a set of **base boxes**
centered at each grid cell
Here $B = 3$

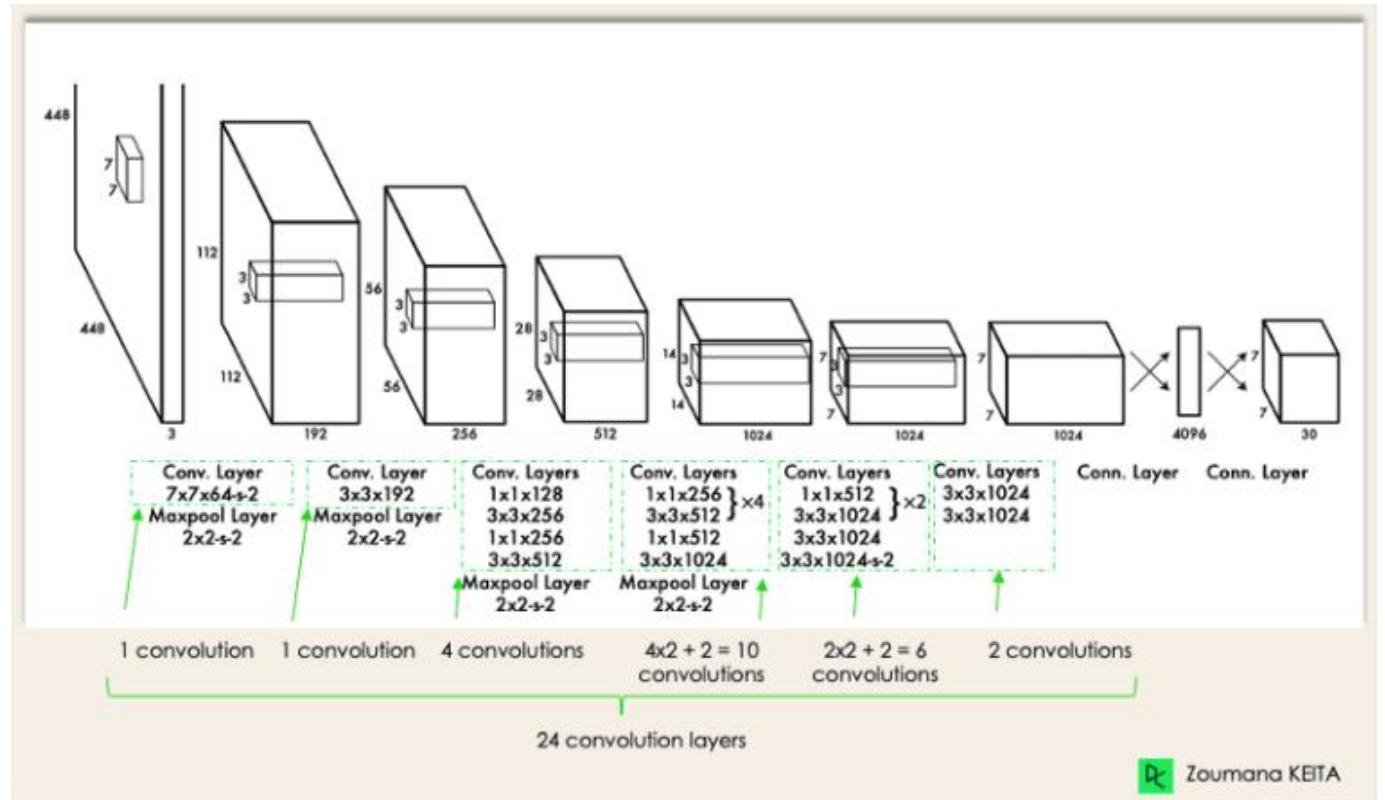
- Within each grid cell:
- Regress from each of the B base boxes to a final box with 5 numbers:
(dx , dy , dh , dw , confidence)
 - Predict scores for each of C classes (including background as a class)

Output:
 $7 \times 7 \times (5 * B + C)$

Redmon et al, "You Only Look Once:
Unified, Real-Time Object Detection", CVPR 2016
Jia et al, "SSD: Single-Shot MultiBox Detector", ECCV 2016

YOLO(You Only Look Once)

- It frames the object detection problem as a **regression problem** instead of a classification task:
 - by spatially separating bounding boxes and associating probabilities to each of the detected images using a single CNN network.
- Why popular?
 - Speed
 - Detection accuracy
 - Good generalization
 - Open-source
- YOLO architecture
 - Similar to GoogleNet with inception module replace by 1x1 and 3x3 conv layers.



YOLO(You Only Look Once)

- Unified Detection:
 - A single neural network
 - The 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.
- How it works?
 - Divide the original image into $S \times S$ grid cells of equal shape. (If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object.)
 - Each cell predicts B **bounding boxes** and **confidence scores** for those boxes. $\Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}}$
 - Determines the attributes of each bounding box using a single regression module, with a final vector (Y) for each bounding box
 - $Y = [x, y, w, h, c]$
 - x, y, h, w (center, width, height of bounding box)
 - c (confidence prediction): the IOU between the predicted box and any ground truth box.
 - Each grid cell also predicts C conditional class probability, $\Pr(\text{Class}_i | \text{Object})$

YOLO(You Only Look Once)

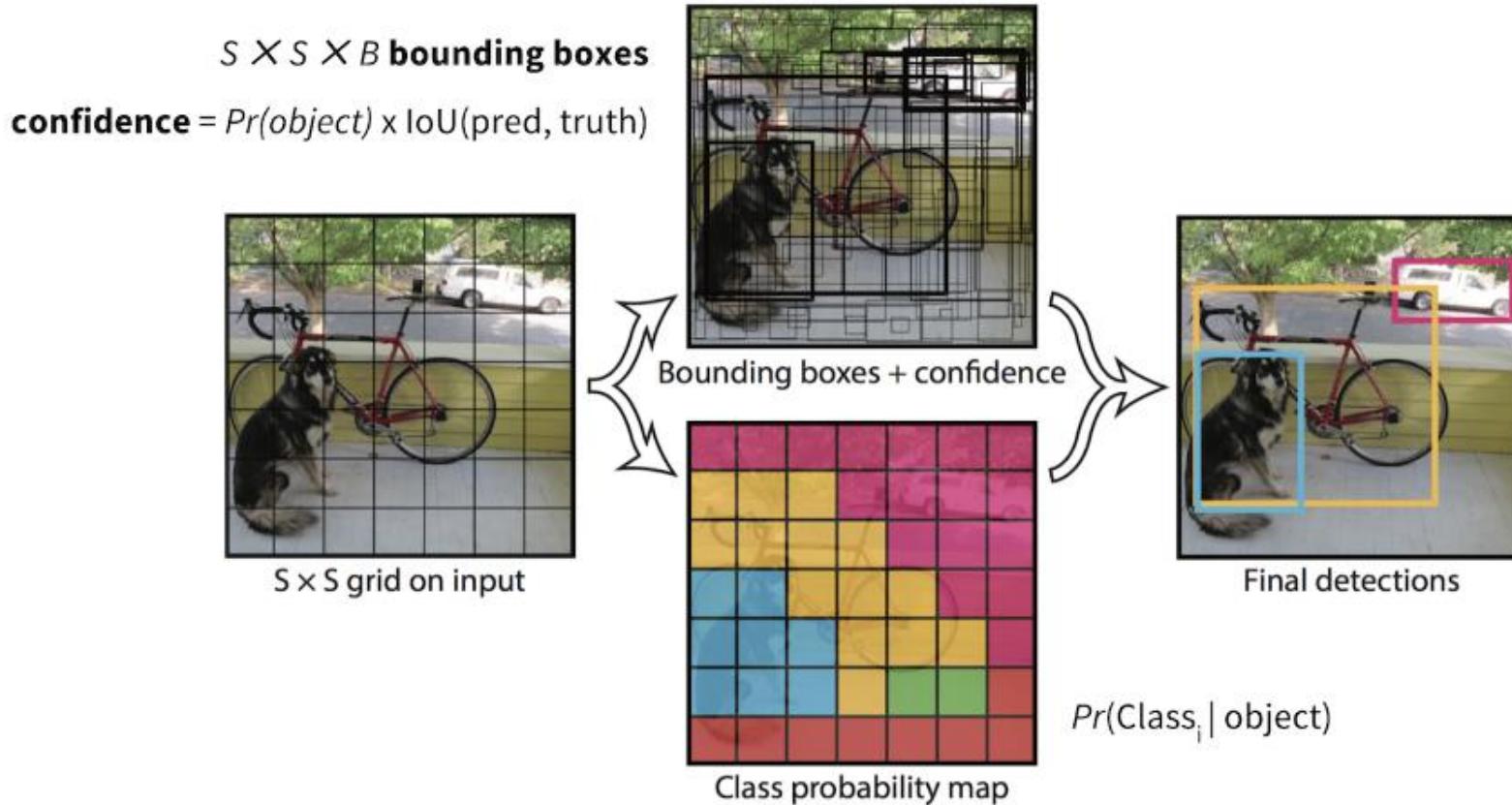


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

YOLO(You Only Look Once)

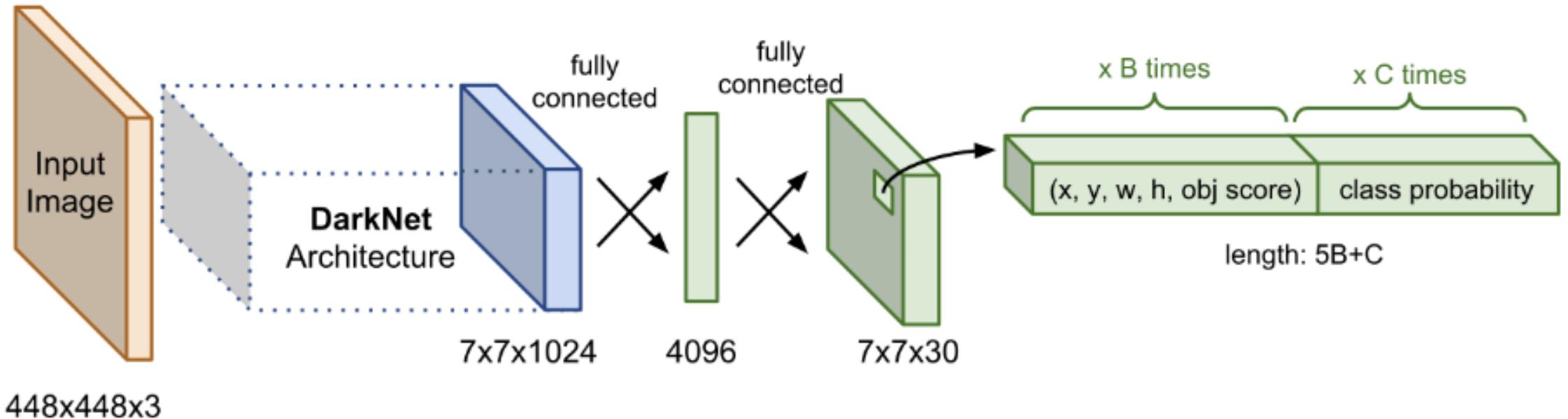
Input: Image I

Output: Detected objects with bounding boxes and class probabilities

1. Divide the input image I into an $S \times S$ grid.
2. For each grid cell (i, j) :
 - a. **Initialize multiple anchor boxes** with predefined sizes and aspect ratios.
 - b. For each anchor box:
 - Predict the bounding box coordinates ($x, y, \text{width}, \text{height}$):
 - * (x, y) are relative to the grid cell, and width/height are relative to the image.
 - Predict the confidence score:
 - * $\text{Confidence} = \text{Pr}(\text{Object}) * \text{IoU}(\text{predicted_box}, \text{ground_truth_box})$
 - Predict the class probabilities for all C classes:
 - * $\text{Class_Probabilities} = [\text{Pr}(\text{Class_1}), \text{Pr}(\text{Class_2}), \dots, \text{Pr}(\text{Class_C})]$
3. Adjust bounding box predictions:
 - a. Use the anchor boxes as a starting point.
 - b. During training, update the bounding box coordinates to better match the ground truth using backpropagation.
4. Post-processing:
 - a. Compute the final detection scores for each anchor box:
 - $\text{Detection_Score} = \text{Confidence} * \text{Class_Probability}$
 - b. Apply Non-Maximum Suppression (NMS) to remove overlapping boxes:
 - For each class:
 - * Sort all anchor boxes by their detection scores.
 - * Remove boxes that have a high IoU (overlap) with a higher-scoring box.
5. Return the final set of bounding boxes and class labels.

- $S \times S$ Grid: 이미지가 $S \times S$ 크기의 격자로 나누어진다. 각 격자 셀은 이미지의 한 부분을 담당한다.
- Anchor Boxes: 각 격자 셀은 다양한 크기와 비율의 앵커 박스를 사용해 객체를 탐지한다. 앵커 박스는 학습 과정에서 객체에 맞게 조정된다.
- Bounding Box Prediction: 각 앵커 박스는 위치, 크기, 신뢰도 점수, 그리고 클래스 확률을 예측한다.
- Non-Maximum Suppression (NMS): 중복된 바운딩 박스를 제거해 최종 탐지 결과를 생성한다.

YOLO Network



Loss Function:

$$\mathcal{L}_{\text{loc}} = \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$

$$\mathcal{L}_{\text{cls}} = \sum_{i=0}^{S^2} \sum_{j=0}^B (\mathbb{1}_{ij}^{\text{obj}} + \lambda_{\text{noobj}}(1 - \mathbb{1}_{ij}^{\text{obj}})) (C_{ij} - \hat{C}_{ij})^2 + \sum_{i=0}^{S^2} \sum_{c \in \mathcal{C}} \mathbb{1}_i^{\text{obj}} (p_i(c) - \hat{p}_i(c))^2$$

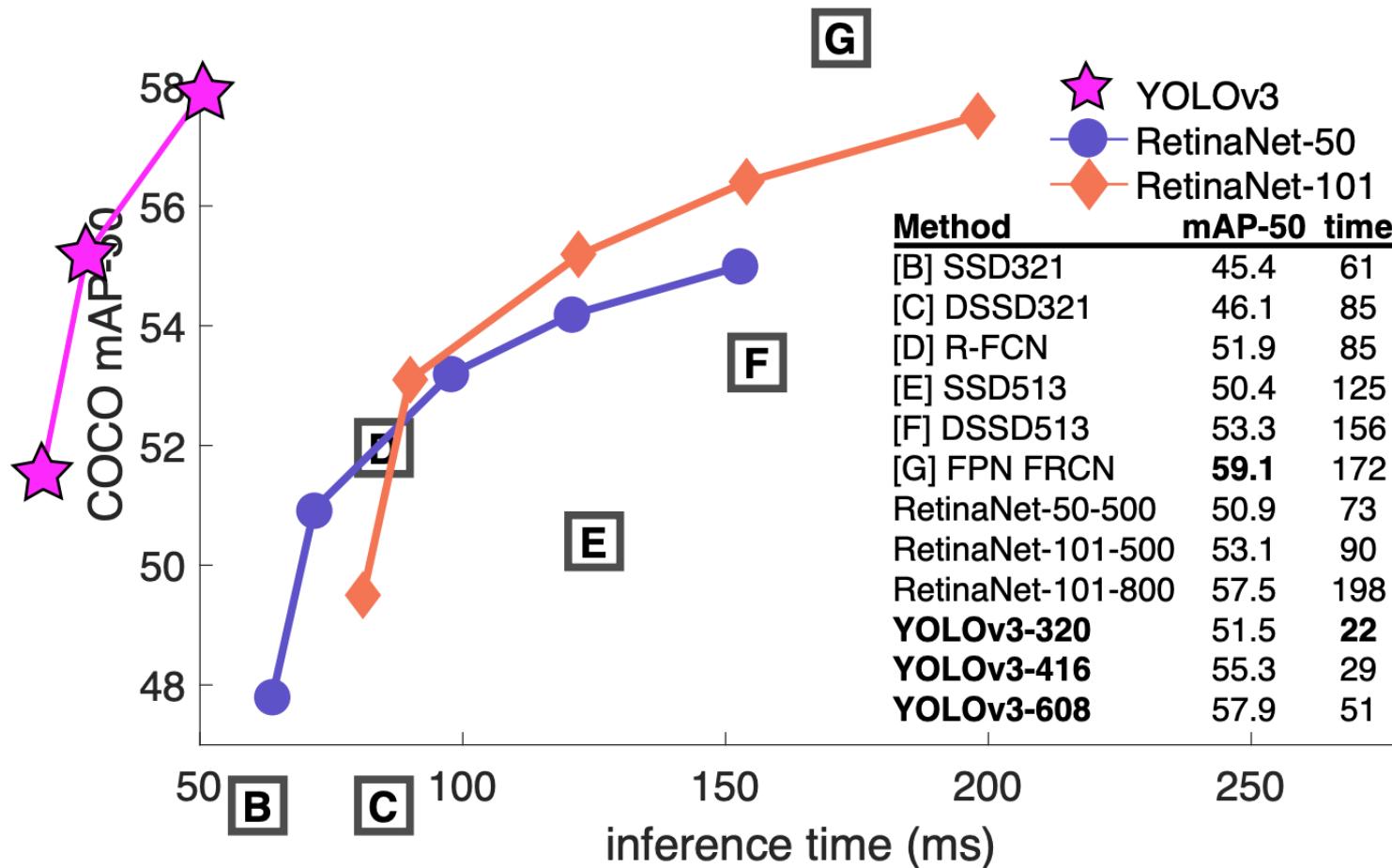
$$\mathcal{L} = \mathcal{L}_{\text{loc}} + \mathcal{L}_{\text{cls}}$$

YOLO(You Only Look Once)

- Training
 - Pre-train the convolutional layers on the ImageNet 1000-class competition dataset .
 - Then, convert the model to perform detection
 - add 4 CNN layers and 3 FC layers
 - increase the input resolution of the network from 224×224 to 448×448 .
 - The final layer predicts both class probabilities and bounding box coordinates
- Inferencing
 - Still requires one network
 - On PASCAL VOC, the network predicts 98 bounding boxes per image and class probabilities for each box.
- Darknet (backbone network for feature extraction):
 - Backbone network to extract features from the input images (followed by additional Conv layers to perform object detection)
 - an independent neural network framework developed by the creators of YOLO. It's written in C and CUDA (not based on TensorFlow or PyTorch).
 - primarily used for research and development

YOLO (You Only Look Once)

- 객체 검출 방식 중에 가장 속도가 빠르고 성능도 우수한 방법으로 널리 사용



YOLO (You Only Look Once)

- 객체 인식(classification)과 위치예측(localization)을 동시에 하나의 신경망으로 수행
 - 즉, 객체 예측과 경계박스를 찾는 작업을 동시에 수행하는 Single Shot Detector(SSD)을 수행
 - 기존의 다른 객체 검출 알고리즘들은 별도로 수행
- 입력 이미지 전체를 여러 지역(region)으로 나누고 경계 박스와 각 지역에 대한 객체 존재 여부를 확률로 예측
 - 이 경계 박스는 예측 확률의 가중합으로 계산
- 98개의 검출 결과를 제공 ($7 \times 7 \times 30 = 1470$ 벡터 출력)
- C로 구현된 DarkNet을 framework로 사용
- 참고: <https://github.com/qzwweee/keras-yolo3>
- 동작 데모 : <https://pjreddie.com/darknet/yolo/#demo>

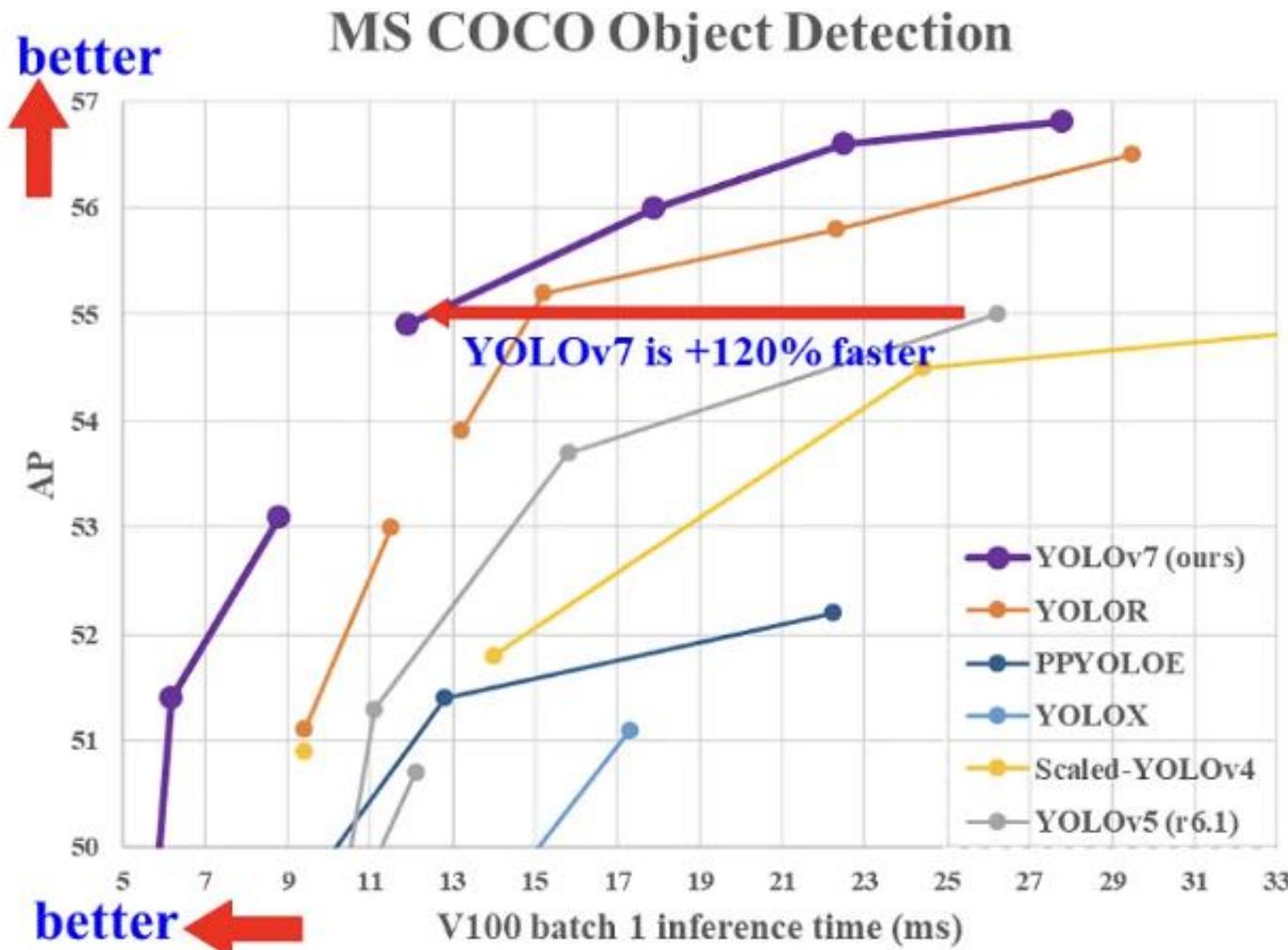
YOLOv3

- Backbone Network
 - Darknet-53 사용
 - 클래스 예측 시에 소프트맥스(Softmax) 대신에 개별 클래스 별로 로지스틱 회귀(Logistic Regression)를 사용
 - Refer to google for the details

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

YOLOv3 모델에서 채택한 DarkNet-53 구조

More YOLO's (v2, v3, v4, v5, v7,...)



<https://viso.ai/deep-learning/yolov7-guide/>

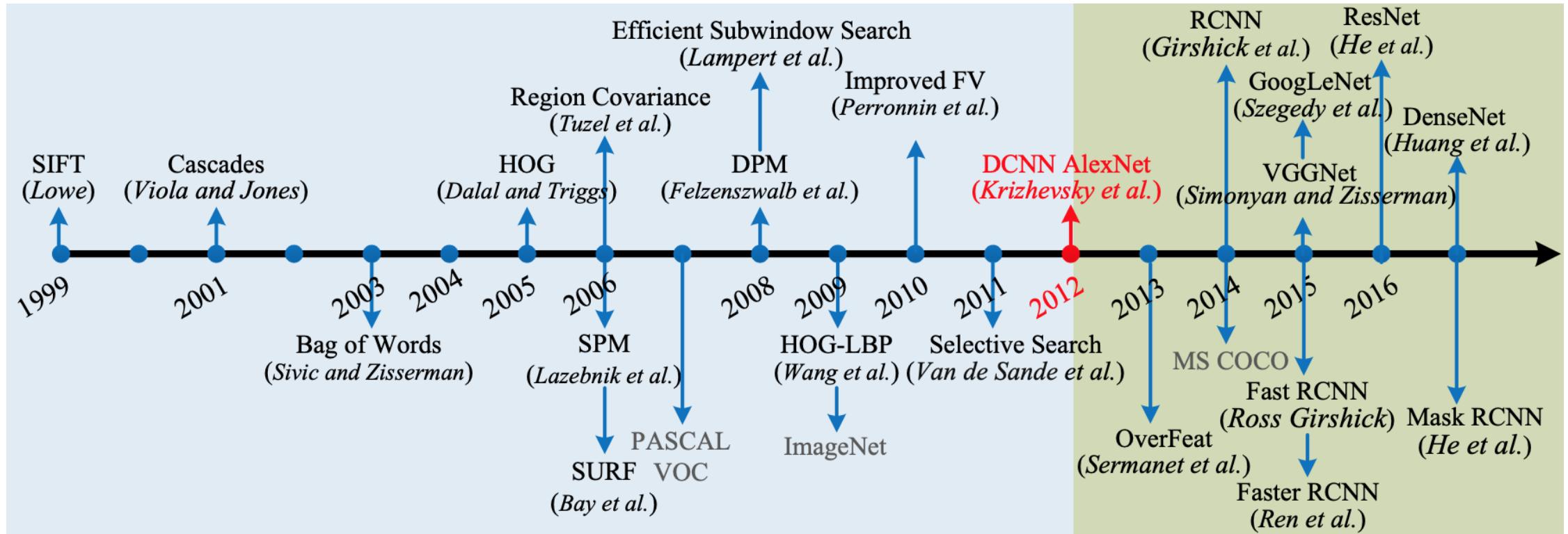
YOLO – Tiny YOLO3

- 정식 YOLO보다 간단하면서 속도가 빠른 tiny YOLO 버전
- 이를 사용하려면 아래와 같이 해당 버전을 설치
 - !wget <https://pjreddie.com/media/files/yolov3-tiny.weights>

Image Dataset for Object Detection – VOC, COCO

- VOC
 - 20개 클래스, 11,530 이미지, 27,450 annotation을 제공
 - image당 2.4개의 객체를 포함
- COCO
 - MS 사가 주최하는 COCO challenge 대회가 2015년부터 운영 중이며 여기서 제공되는 데이터가 COCO(common object context)
 - 91 카테고리, 20만개의 이미지, 50만개의 annotation, 32만개 영상을 제공
 - image당 평균 7.3 개의 객체를 포함

Image Dataset and Algorithms for Object Detection



Semantic Segmentation (분할)

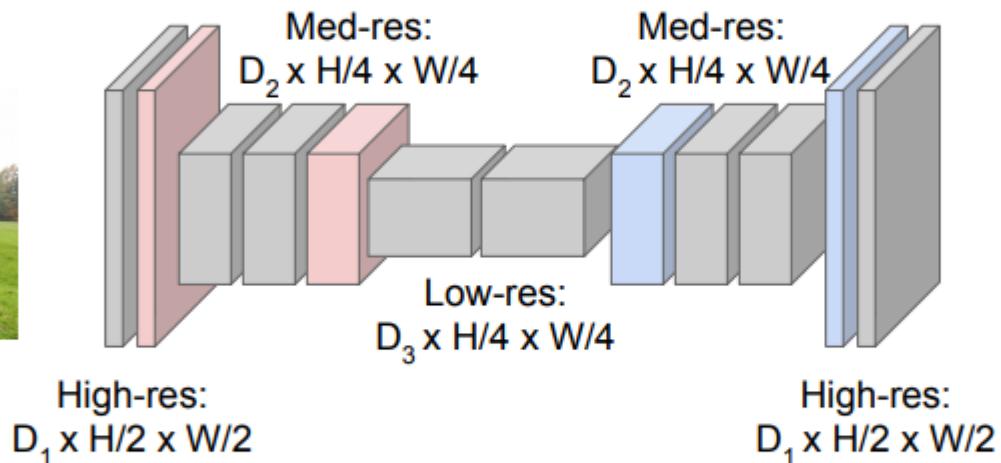
Semantic Segmentation Idea: Fully Convolutional

Downsampling:
Pooling, strided convolution



Input:
 $3 \times H \times W$

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!



Upsampling:
Unpooling or strided transpose convolution



Predictions:
 $H \times W$

Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

Semantic Segmentation (분할)

- 픽셀 단위로 객체의 클래스를 표시하는 작업
- 일반 머신러닝 모델 (분류, 회귀)
 - 정보를 계속 줄여나가는 작업을 주로 수행 (feature extraction)
 - 분류와 회귀는 모두 정보를 줄인 결과 클래스 이름이나 수치를 얻는다.
- 객체 분할
 - 원래 이미지 크기의 이미지를 다시 얻어야 한다
 - 따라서 정보를 축소한 후에 다시 정보를 늘리는 작업을 수행해야 한다.
 - Deconvolution, Up-Pooling 수행

Semantic Segmentation (분할)

- Goal:
 - To predict class labels for each pixel in the image
 - Construct a Segmentation map (size: Height x Width x 1) from the input image



Segmentation (분할)

- Encoder-Decoder architecture for Image Segmentation (convolutional auto-encoder)

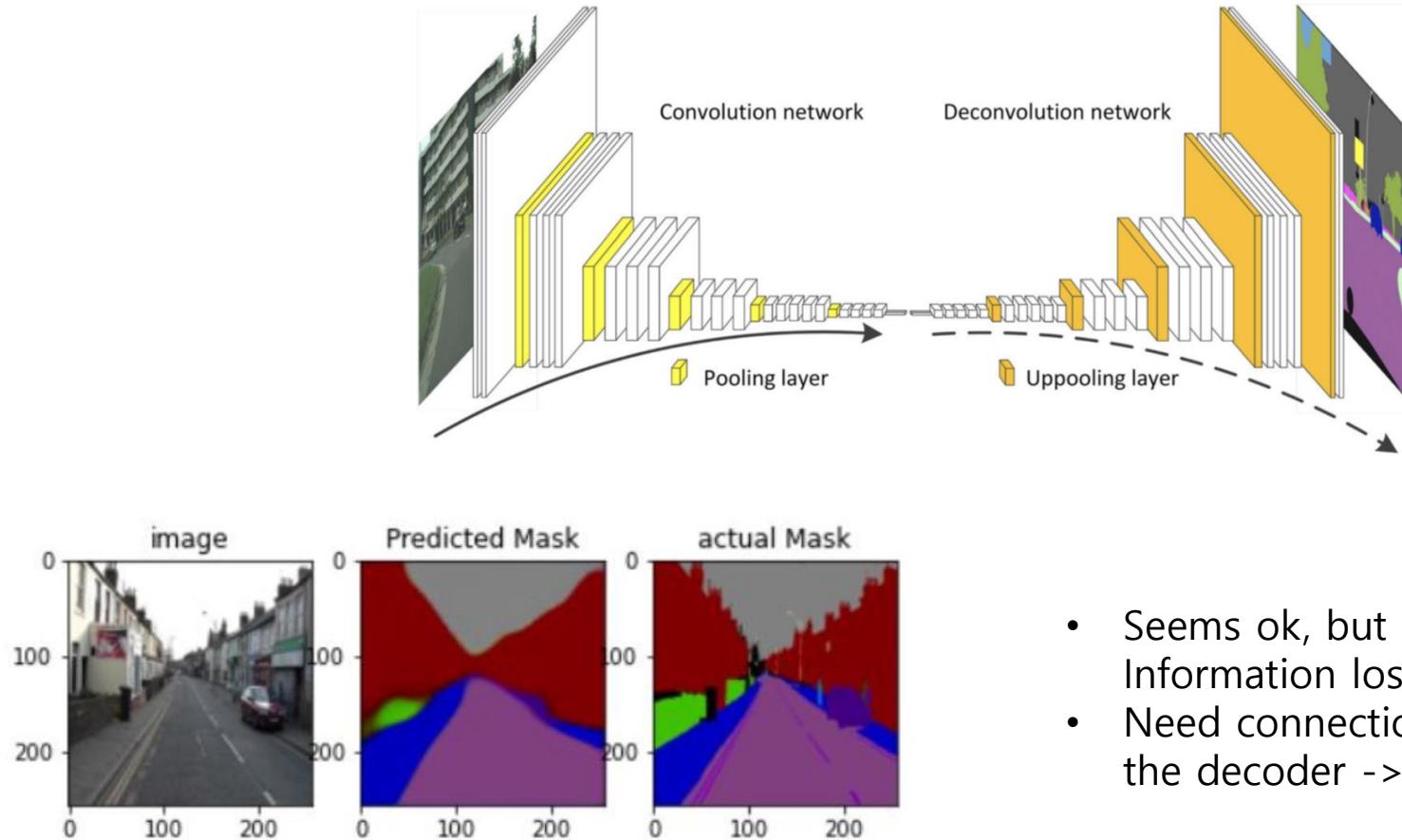
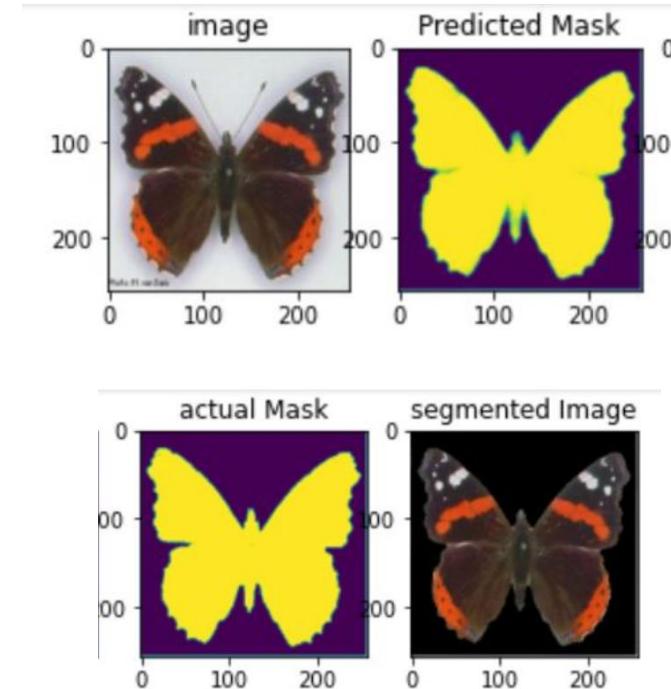
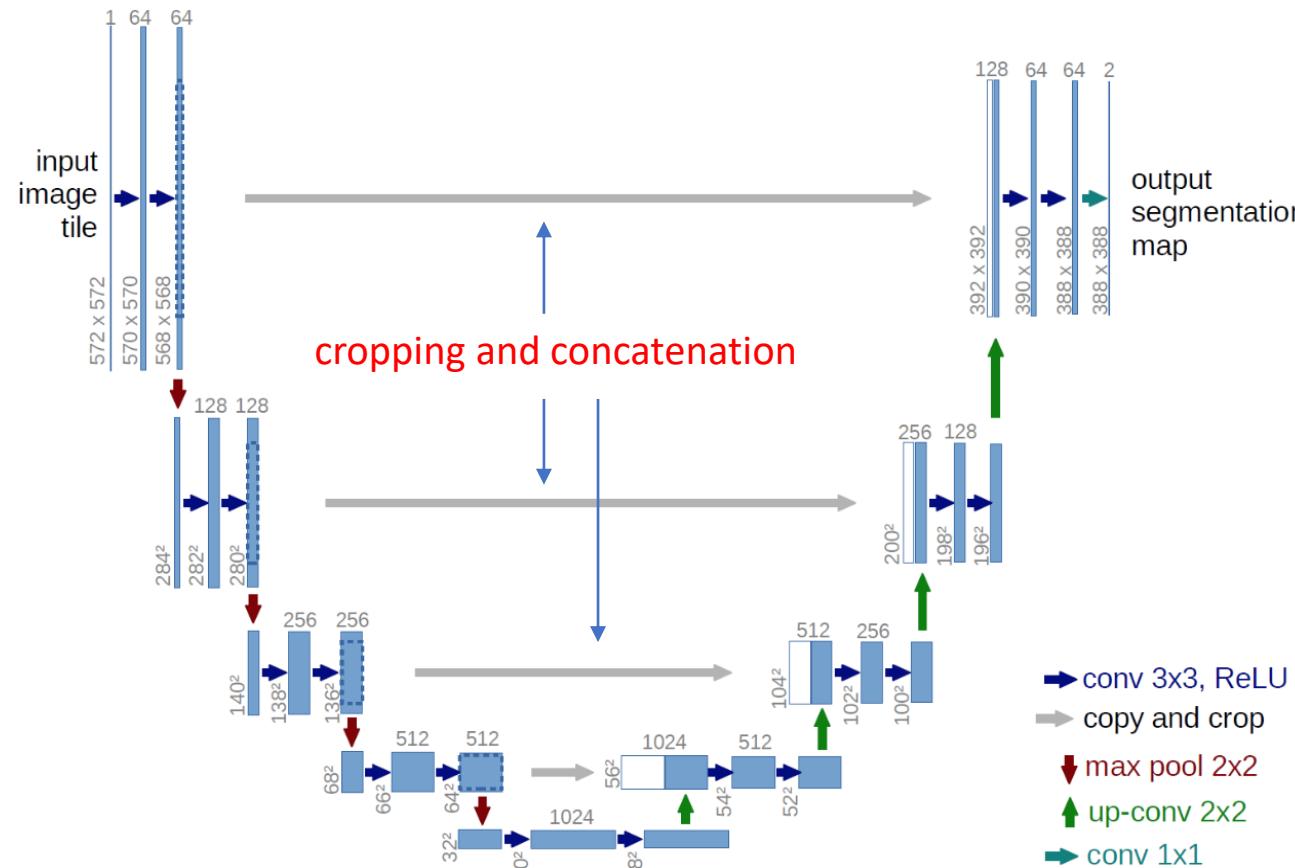


Figure 4: Auto-Encoder Results for Semantic Segmentation

- Seems ok, but not good enough because of Information loss in feature maps
- Need connection between the encoder and the decoder -> U-Net

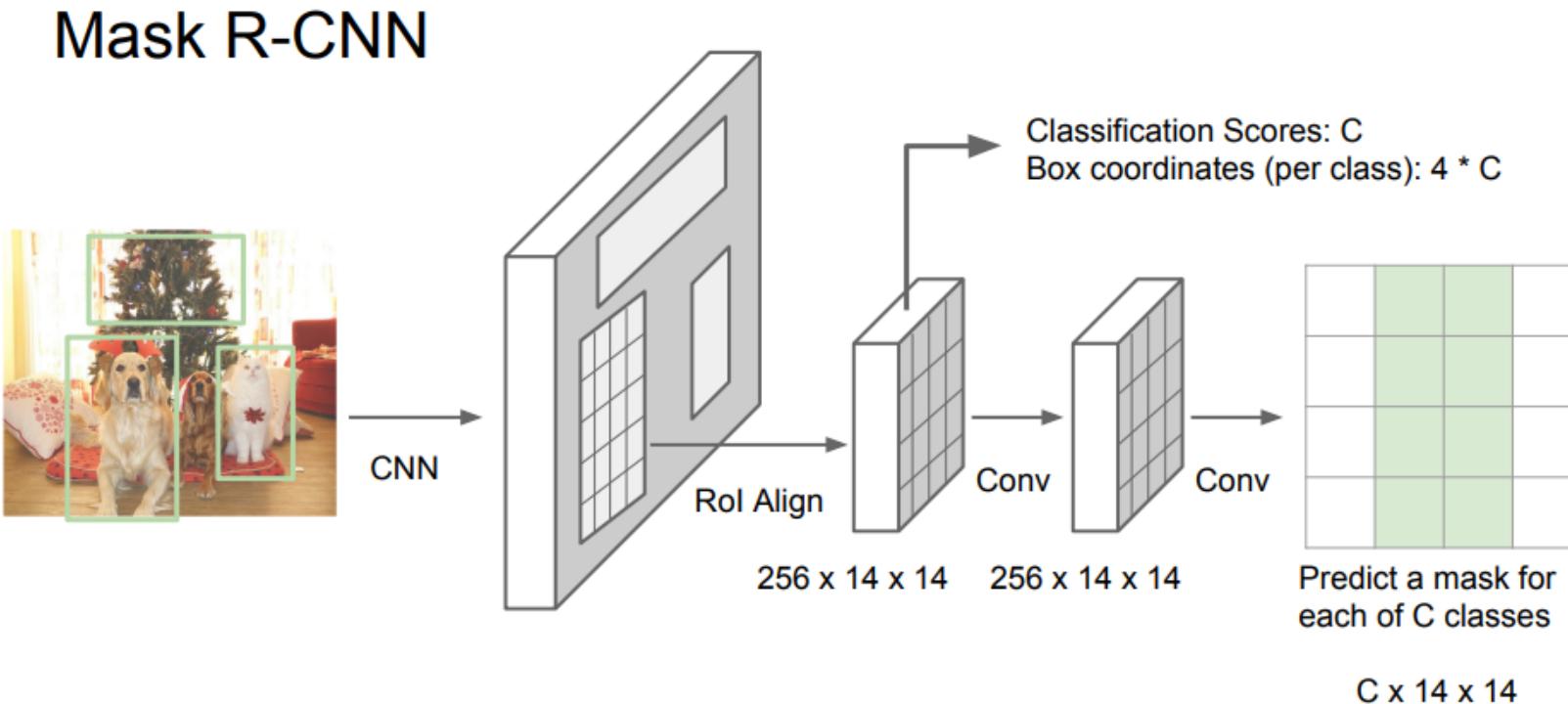
U-Net for image segmentation

- U-Net (2015): Convolutional Network for Biomedical Image Segmentation
 - a convolutional auto-encoder with additional connections between the encoder and the decoder parts (use **skip connection**: [Input Space + Latent Space] -> Output Space)
 - Contracting Path (extract context) and Expanding Path (up-sampling for dense resolution)



Instance Segmentation (Mask R-CNN)

- object detection (Faster R-CNN) + image segmentation
- use a pre-trained model which has ResNet-50-FPN backbone



He et al., "Mask R-CNN", arXiv 2017

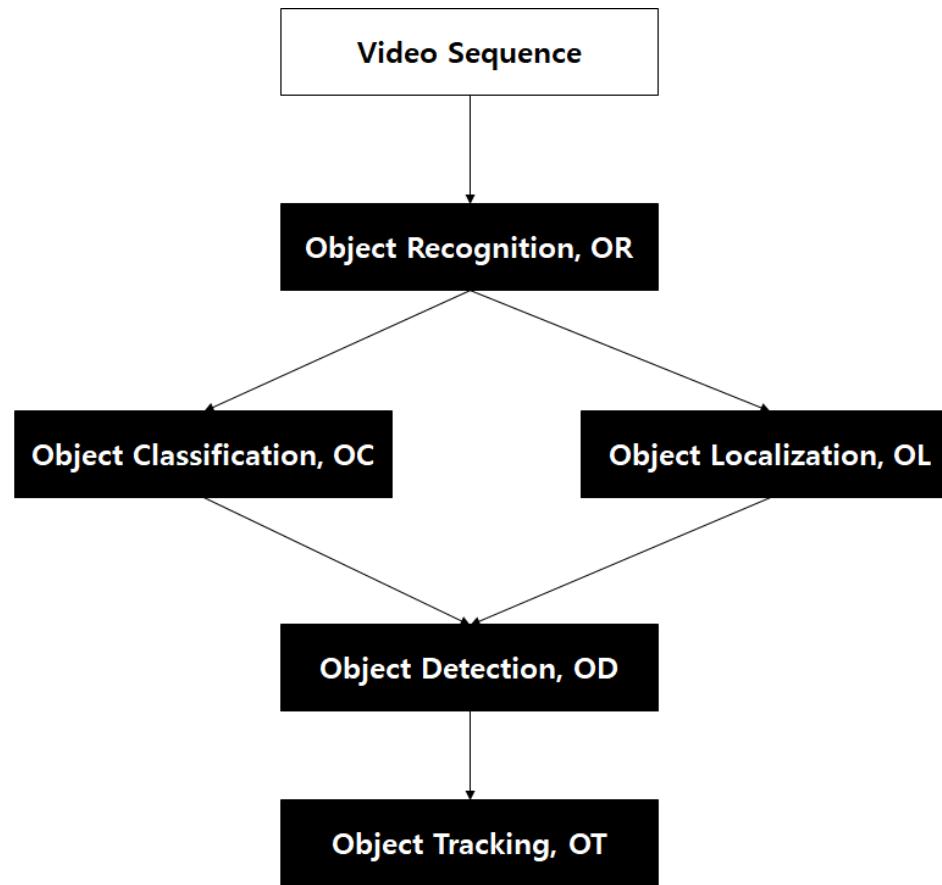
Mask R-CNN

- Detectron이라는 기술로 Facebook에서 공개
- 상당히 정교하게 객체 인식과 세그멘테이션을 수행



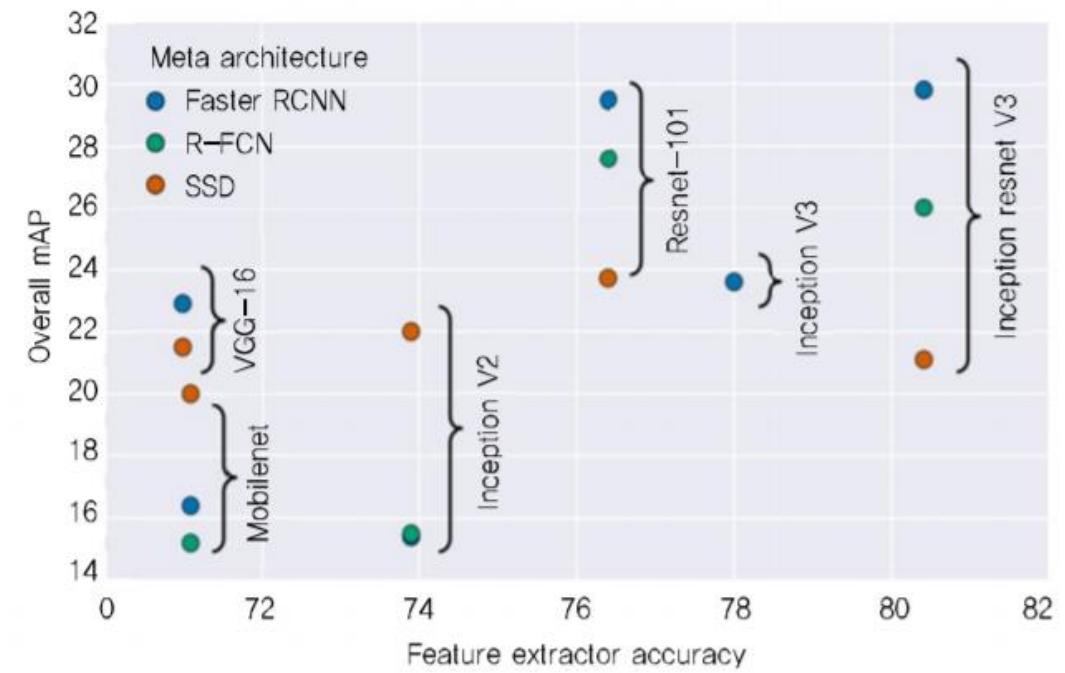
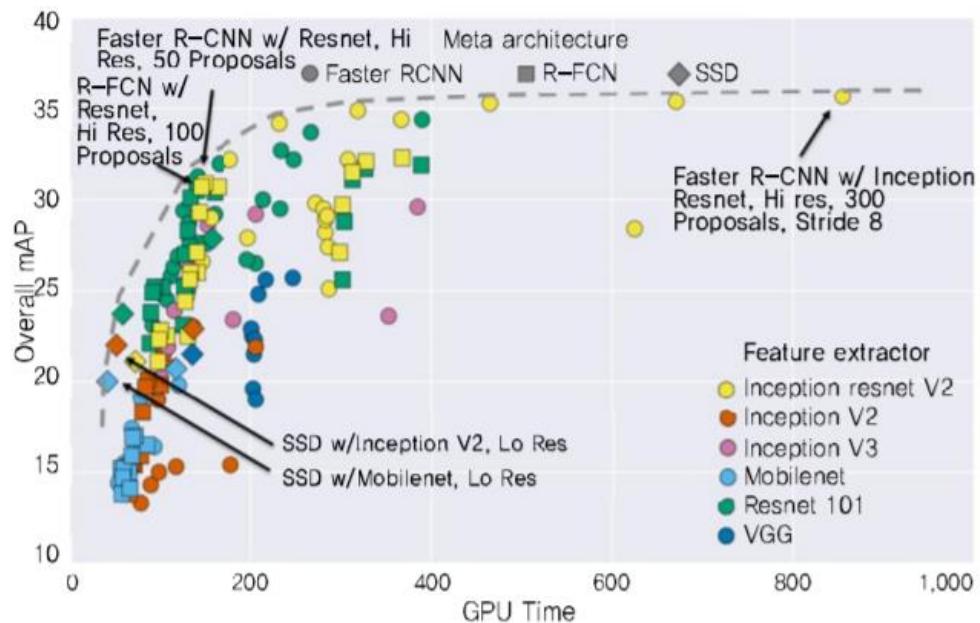
(Object) Recognition, Detection, Tracking

- 객체 인식, 탐지, 추적 개념



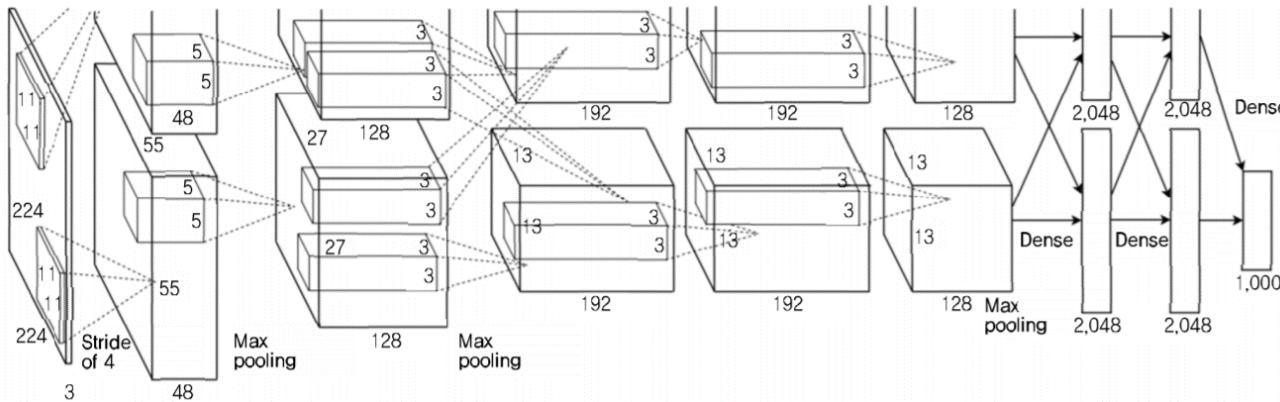
Object Recognition

- 객체 탐지 모델의 성능 비교,
- CNN 모델 및 구조의 객체 인식 성능 비교

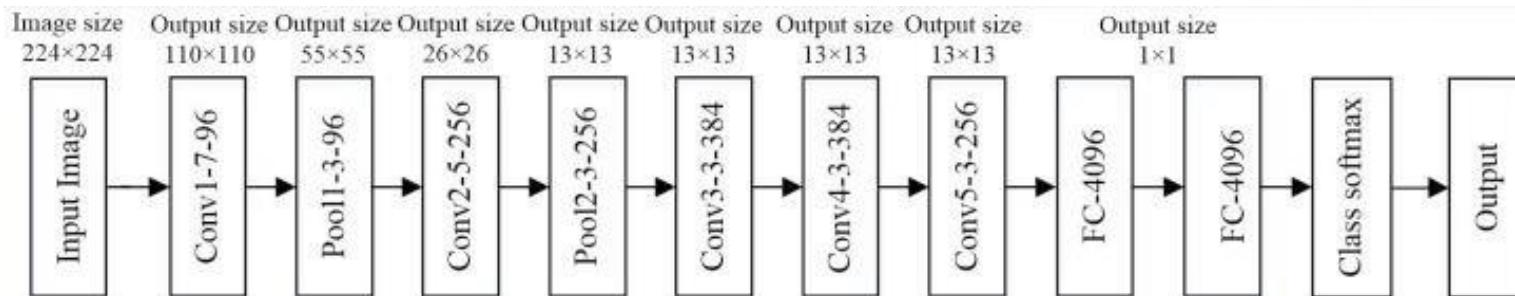


CNN Network

- AlexNet: 8 개 층 CNN



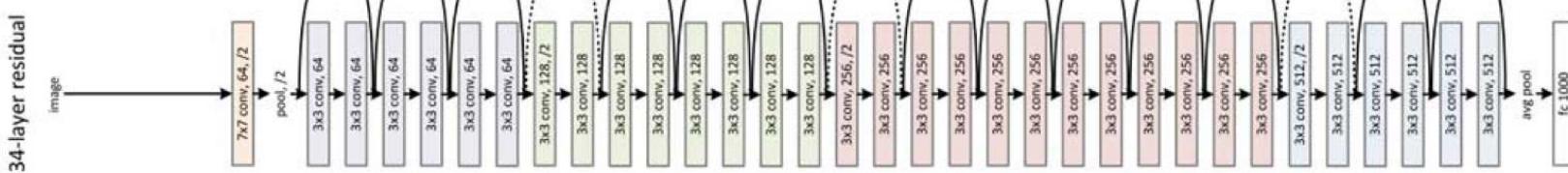
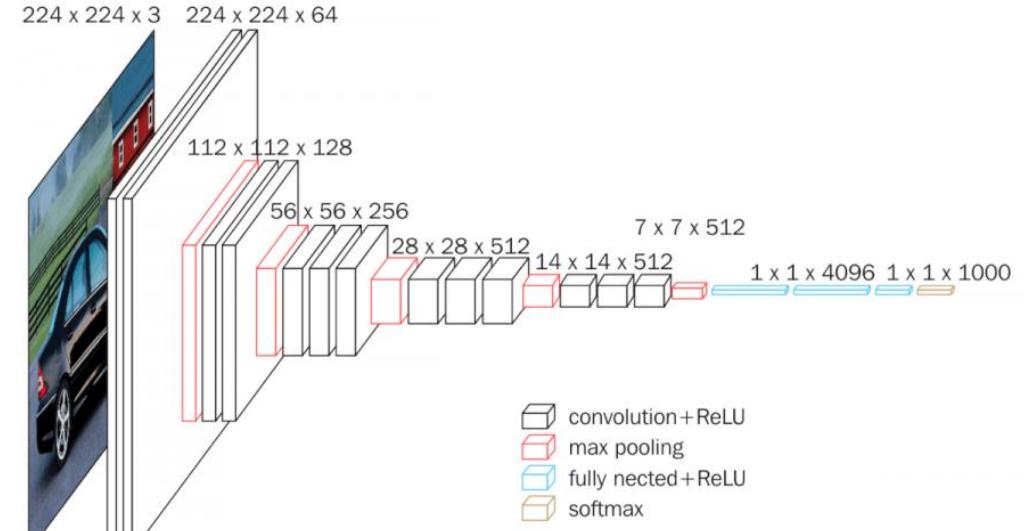
- ZFNet: AlexNet 의 구조 개선



CNN Network

- VGG: 전체 filter 사이즈를 3 으로 통일,
간단하면서도 우수한 성능

- ResNet:
 - VGG 구조를 34개 층으로
구성하고 Residual Learning
적용한 형태
 - Deeper Network 이면서도
Residual Learning(잔차학습)
통해 성능 및 계산 효율성 유지



Object Detection

- **YOLO (You Only Look Once) Model**
 - 빠른 성능으로 동영상에도 객체 인식 적용 가능하게 됨

객체 인식 및 탐지 모델	처리 속도
R-CNN	5초
Fast R-CNN	(영상) 0.5 프레임
Faster R-CNN	(영상) 7.0 프레임
YOLO	(영상) 45.0 프레임 [높은 버전의 YOLO의 경우 155프레임]

- **YOLO-V3**
 - Backbone 네트워크로 ResNet-101 대신에 DarkNet-53 구조를 채택

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

DarkNet-53 구조

Object Tracking

- Object Tracking 기본과정

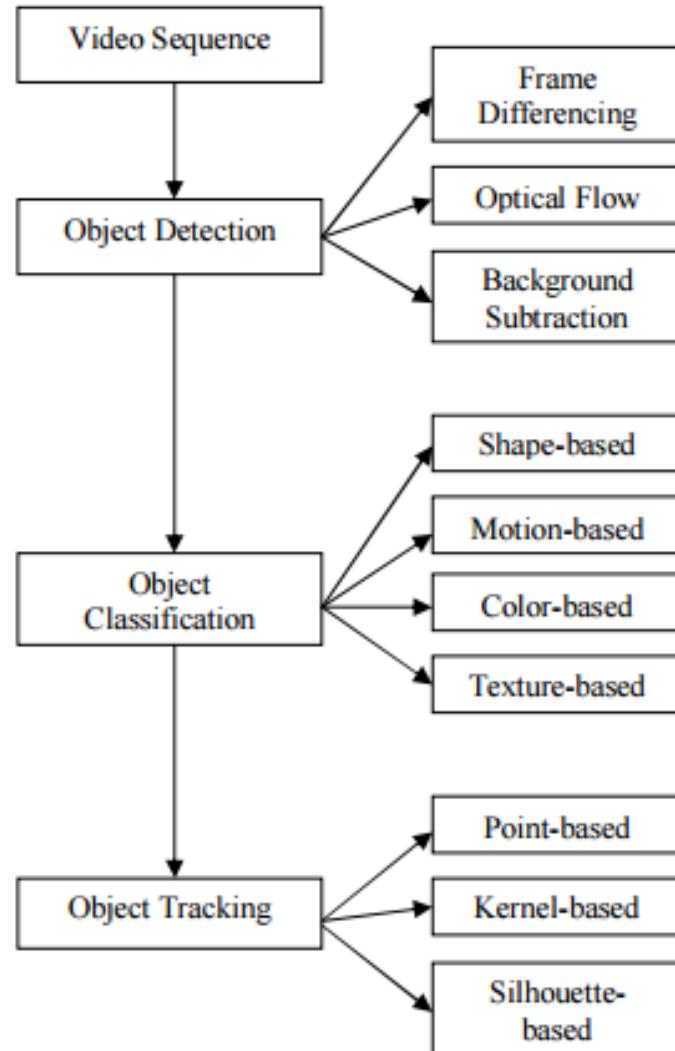


Fig 1: Basic steps for tracking an object [8]

Object Tracking

- **Object Tracking 방식**

- VOT (Visual Object Tracking): 단일 물체 (Single object) Tracking
- MOT (Multiple Object Tracking): 다중 물체 Tracking
- On-line Tracking and Off-line Tracking

- **Occlusion**

- 추적(Tracking)하는 객체에 부여한 고유의 트래킹 아이디(Traking ID)가 부정확해지는 문제
- (예) 추적하던 두 개 이상의 객체의 위치가 겹치는 경우에 트래킹 아이디가 서로 바뀌거나, 새로운 물체가 아닌데도 불구하고 아이디가 새로 부여되는 등 고유 트래킹 아이디가 부정확해지는 문제가 발생



Loss Function for Object Detection

- 학습을 시키려면
 - 입력 이미지에 대해서 객체의 종류와 위치를 찾아야 한다
- 레이블 - 2가지 제공해야
 - 이미지 클래스
 - 이미지가 위치한 박스의 좌표
- 출력이 두 가지 이상인 멀티 출력 모델을 만들어야 하며 손실함수는 분류에 관한 손실과 회귀에 관한 손실의 합으로 구성

$$Loss = \alpha * Softmax_Loss + (1-\alpha) * L2_Loss$$

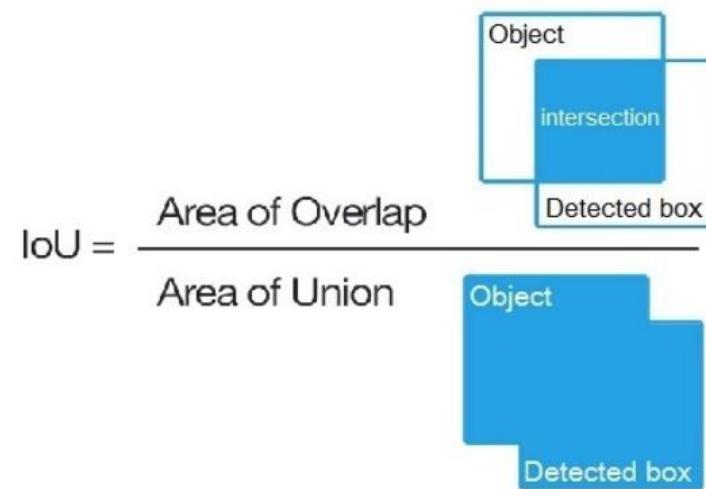
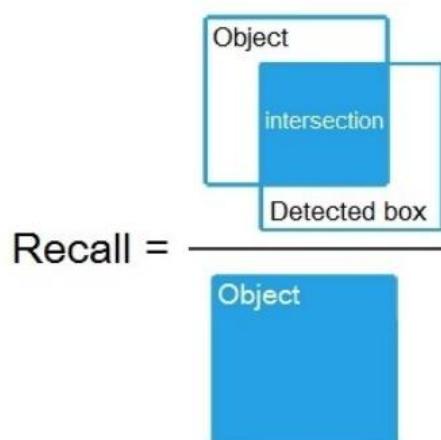
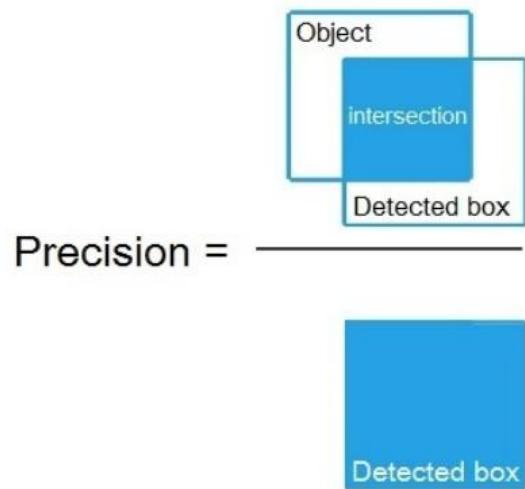
- 두 가지 성분의 손실의 상대적인 비중은 α 값으로 조정

Performance

- 객체 검출에서는 단순히 어떤 객체가 있는지를 찾는 분류 문제가 아니므로 정확도 등 만으로 성능을 평가할 수 없고, 예측한 객체의 위치가 실제 객체의 위치와 얼마나 일치하는지를 평가해야 한다
- 성능 평가척도
 - IoU (Intersection over Union)
 - mAP (mean Average Precision)
 - AR (Average Recall)

Object Detection Performance

- 객체검출 성능 평가: 객체의 위치를 얼마나 실제 위치와 겹치게 잘 찾았는지 평가
 - 정밀도 (precision) : 예측한 면적분에 겹치는 면적
 - 리콜 (recall) : 실제 이미지 면적분에 겹치는 면적
 - IoU (Intersection over Union) : 두 가지 면적의 전체집합 부분에 비하여 겹치는 부분의 면적의 비율



$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

Performance - IoU

- 실제 이미지 예

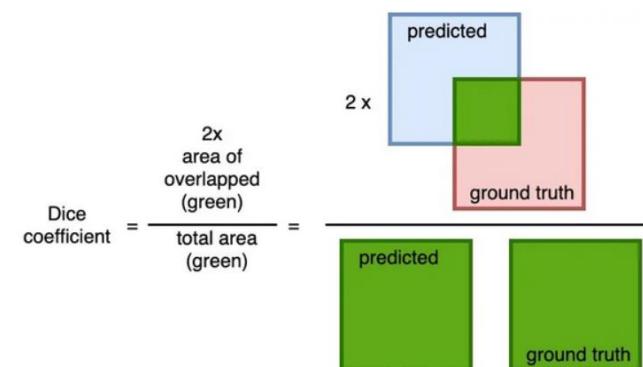
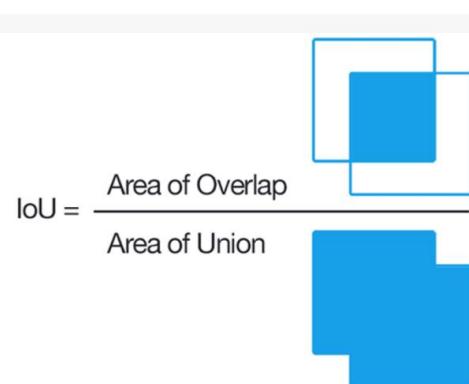
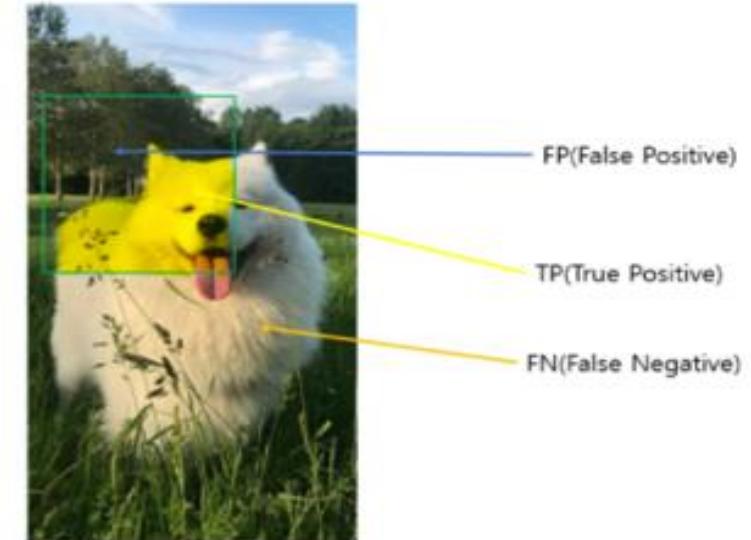


Performance – IoU and Dice coefficient

- Similarity between the prediction and the ground truth

$$IoU = \frac{\text{Intersection}}{\text{Union}} = \frac{TP}{FP + TP + FN} = \frac{\sum \hat{y}y}{\sum \hat{y} + y - \hat{y}y}$$

$$\text{Dice coefficient} = \frac{2 \times \text{Intersection}}{\text{Union} + \text{Intersection}} = \frac{2TP}{2TP + FN + FP} = \frac{2 \sum \hat{y}y}{\sum \hat{y} + y}$$
$$Dice = \frac{2 \times \text{Area of Intersection}}{\text{Area of Predicted} + \text{Area of Ground Truth}}$$



- Both IOU and Dice are in range (0,1)
- Dice is generally more sensitive to small differences between masks, making it suitable for tasks where precise segmentation is critical, such as medical imaging.

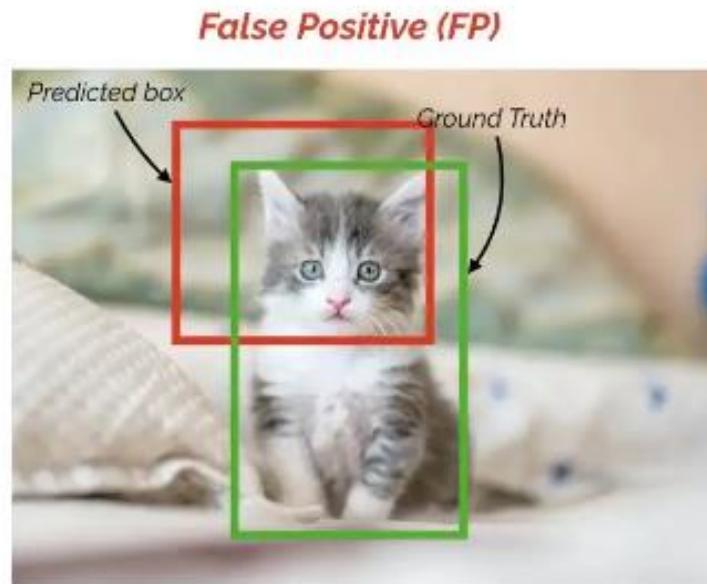
Performance – mAP

- 객체 검출에서 검출할 객체가 여러 개가 있을 때 이들의 평균 예측 성능을 표현하는데 mAP가 많이 사용
- AP (average Precision)
 - the area under the precision-recall curve
- mAP (mean Average Precision)
 - 검출 작업의 평균 정확도
 - 여러 객체에 대해 각각 AP를 구하고 이의 평균치를 구한 것
 - 일반적으로 mAP가 0.5 이상이면 true positive라고 판단

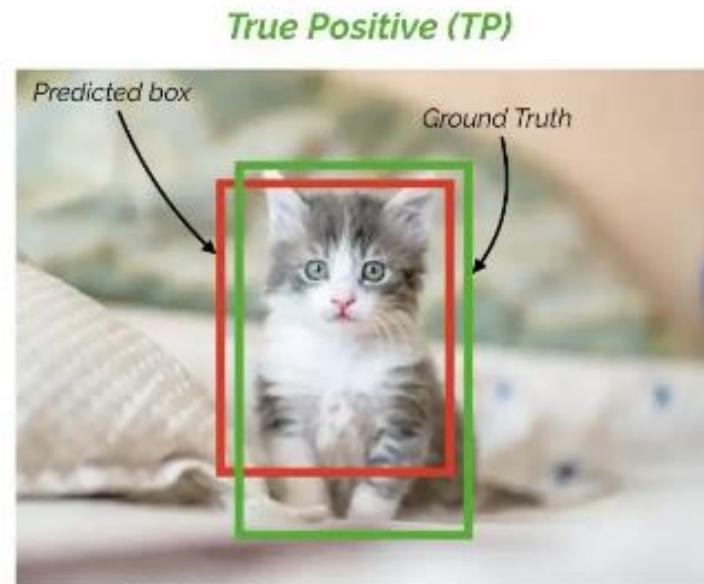
Object Detection Performance

- We calculate Precision and Recall using IoU value for a given IoU threshold.
- For example,

If IoU threshold = 0.5



$IoU = \sim 0.3$



$IoU = \sim 0.7$

Performance – mAP

- The mAP is calculated by taking the mean AP over all classes and/or overall IoU thresholds, depending on different detection challenges that exist. (usually, **IoU threshold of 0.5** is used)
- When we calculate **AP** by taking the area under the PR curve, this is done by segmenting the recalls evenly to 11 parts: { 0, 0.1, 0.2, … , 0.9, 1 }

$$\begin{aligned} AP &= \frac{1}{11} \sum_{r \in \{0.0, \dots, 1.0\}} AP_r \\ &= \frac{1}{11} \sum_{r \in \{0.0, \dots, 1.0\}} p_{interp}(r) \end{aligned}$$

, where $p_{interp}(r) = \max_{\tilde{r} \geq r} p(\tilde{r})$

