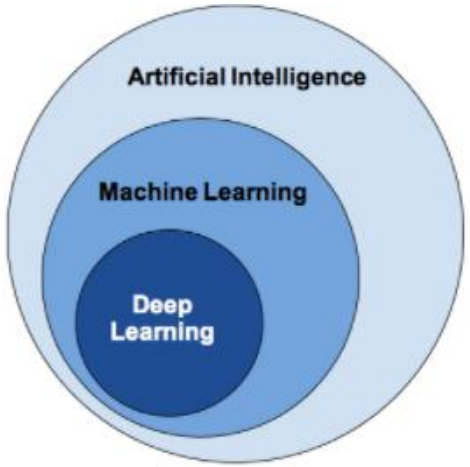
Image Processing

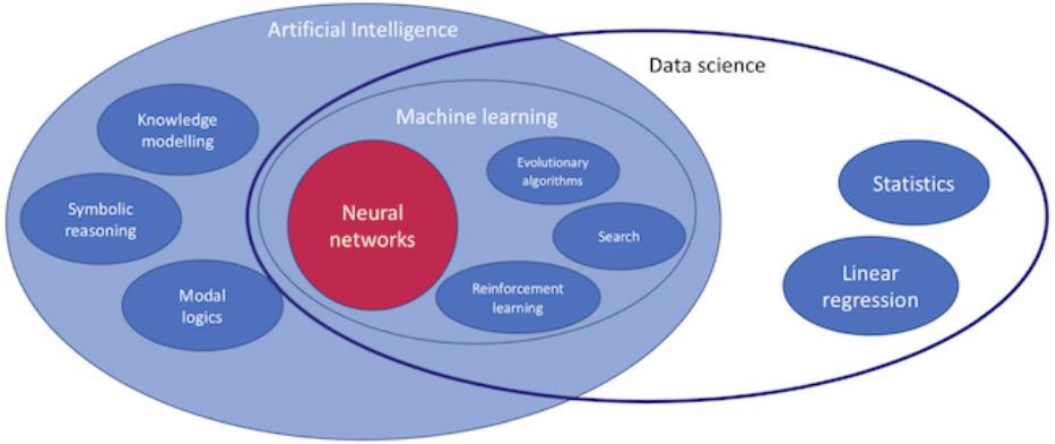
2025. 11

Yongjin Jeong, KwangWoon University

Machine Learning

- **What is ML**
  - the study of computer algorithms that improve automatically **<u>through experience and by the use of data</u>**. 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/

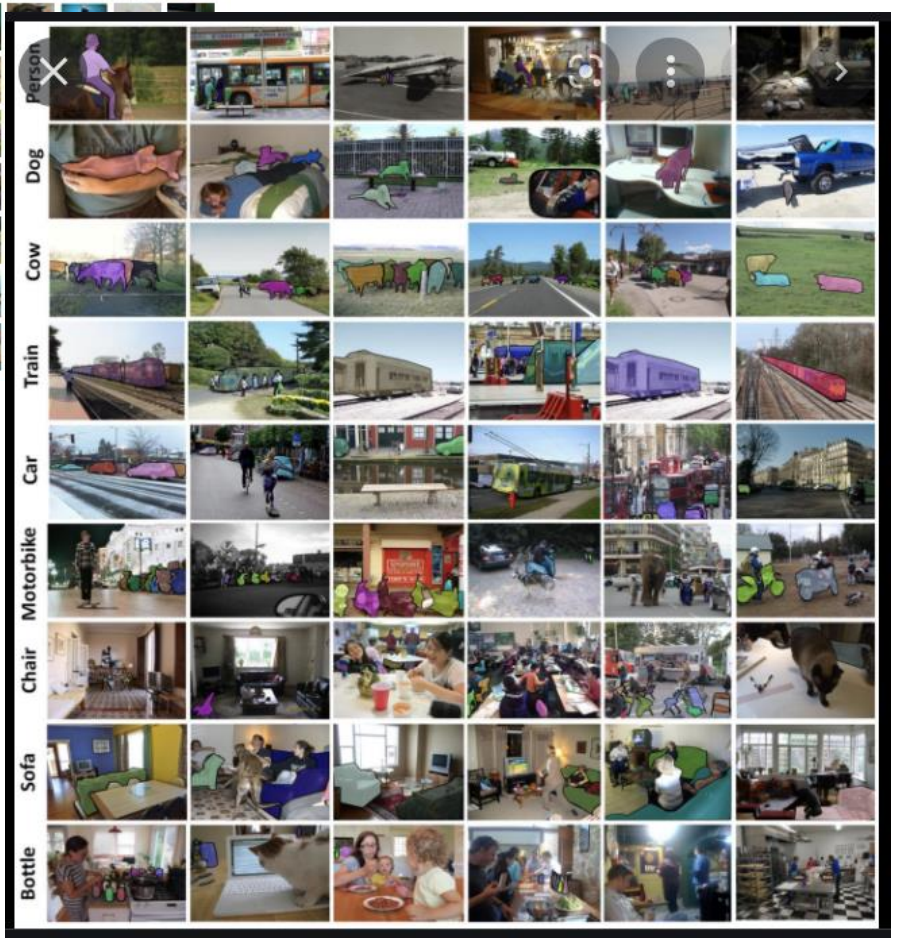
Datasets for Machine Learning

- **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

Fashion MNIST

Datasets for Machine Learning

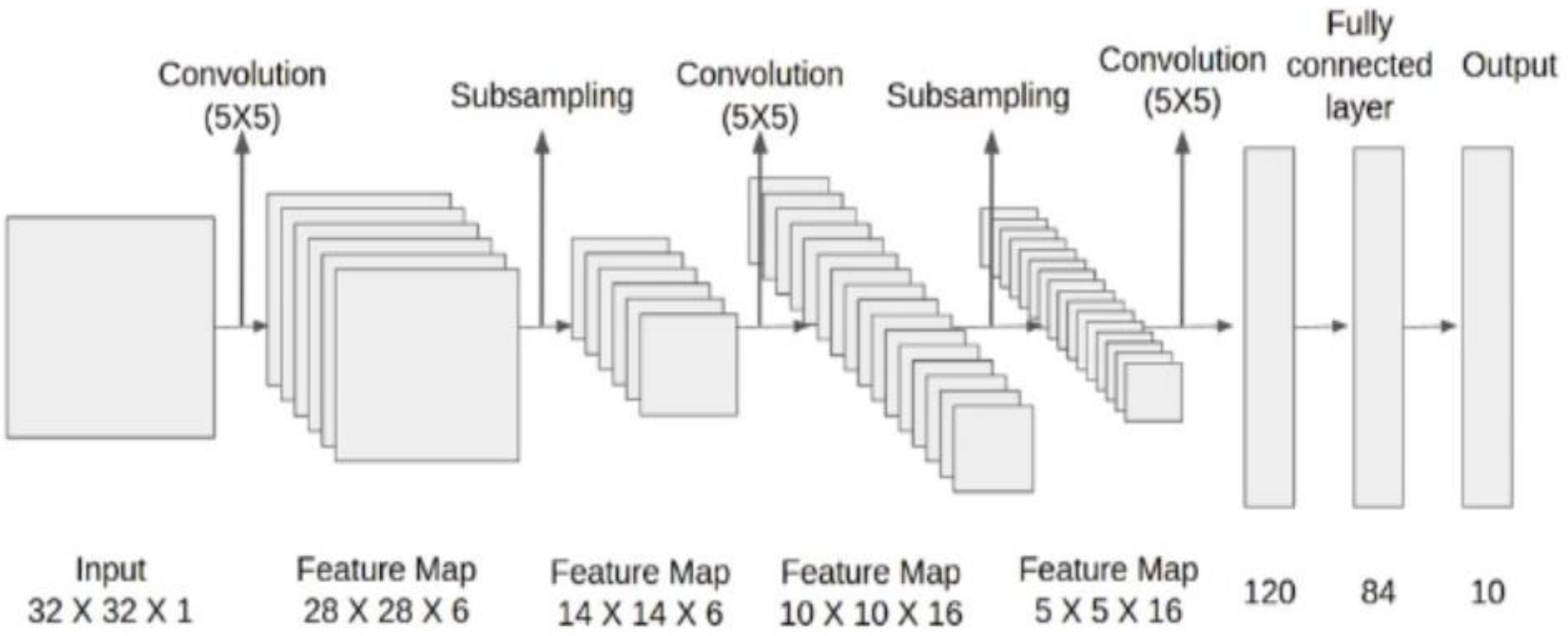
- **Cifar-10 (Canadian institute for advanced research)**
  - 32*32 color images in 10 classes, with 6,000 per class
  - 50,000 training, and 10,000 test images
  - Bigger dataset (Cifar-100)
- **Microsoft COCO** (common objects in Context)
  - Large image recognition/classification, object detection, 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

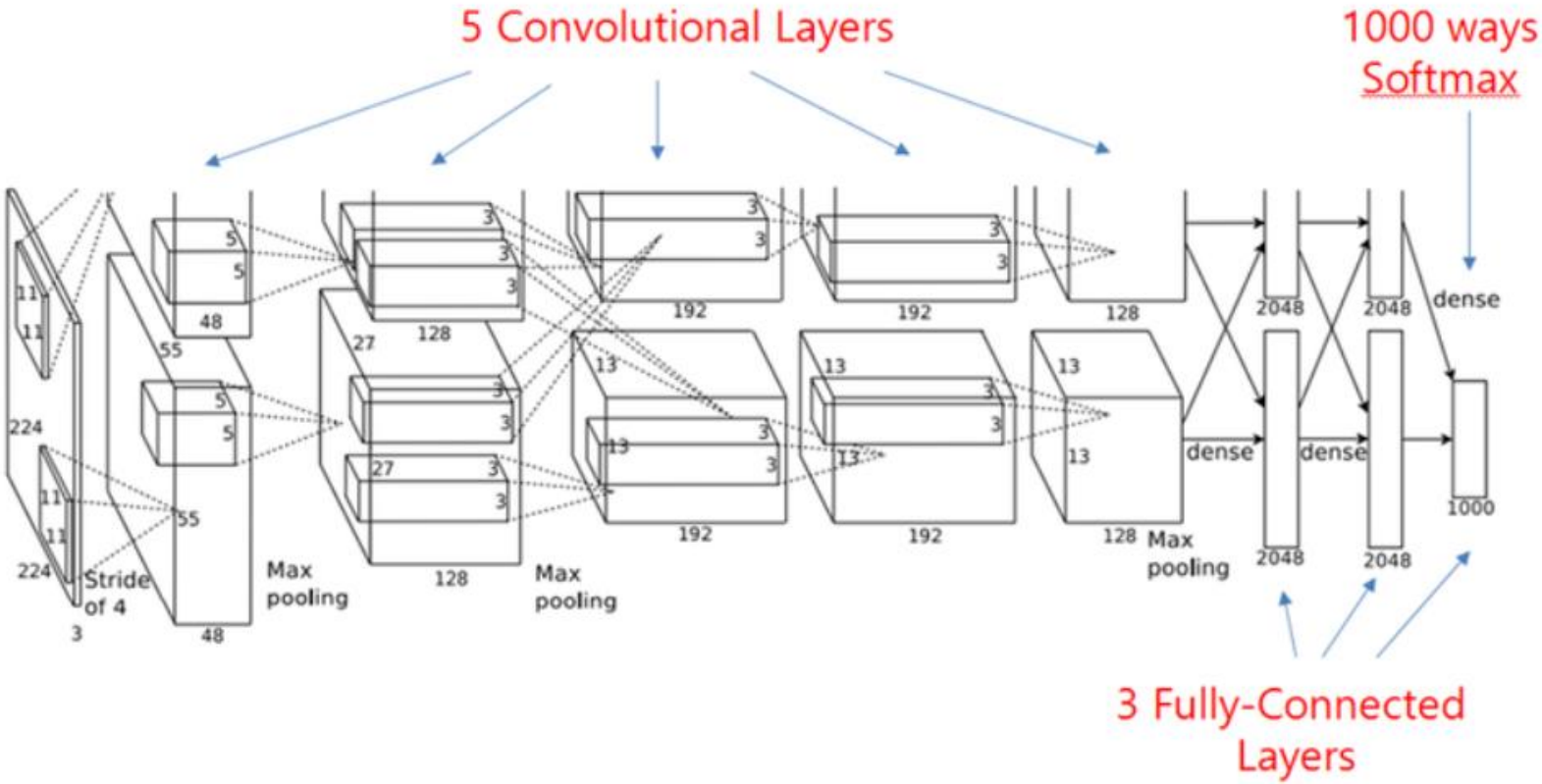
Lenet-5 (1998, LeCun et al.)



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

ILSVRC Winners

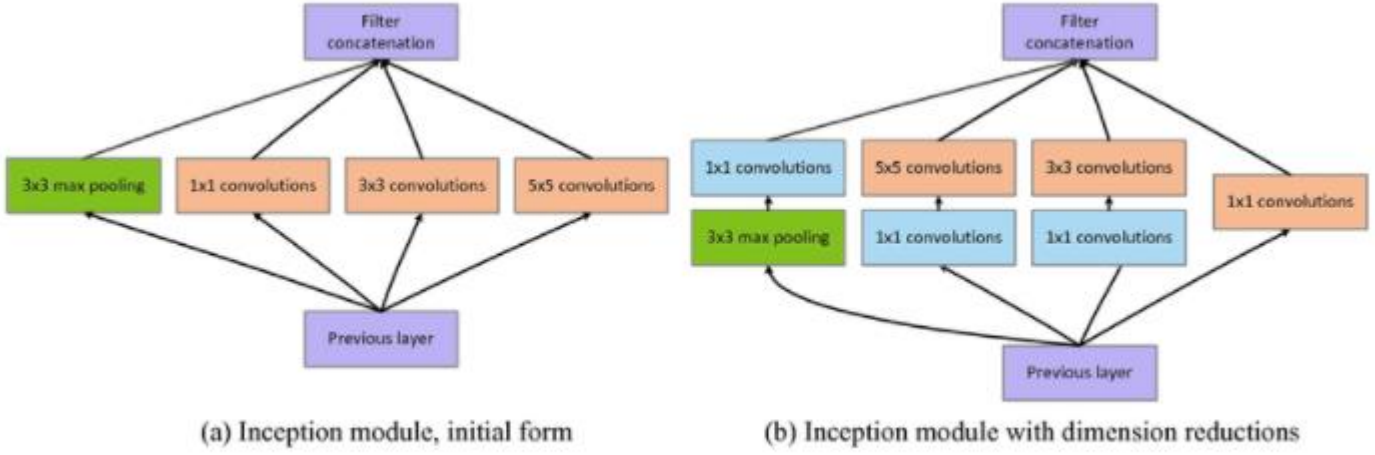
- **ALexNet – 2012 Winner (top-5 error rate 15.3%)**
  - Use CNN (prior to 2012, the classification model error was 25%)
  - Regarded as a Pioneer of CNN and starting point of the Deep learning
  - 60 million parameters
  - Used ReLU activation function, data augmentation, dropout, and overlapped pooling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

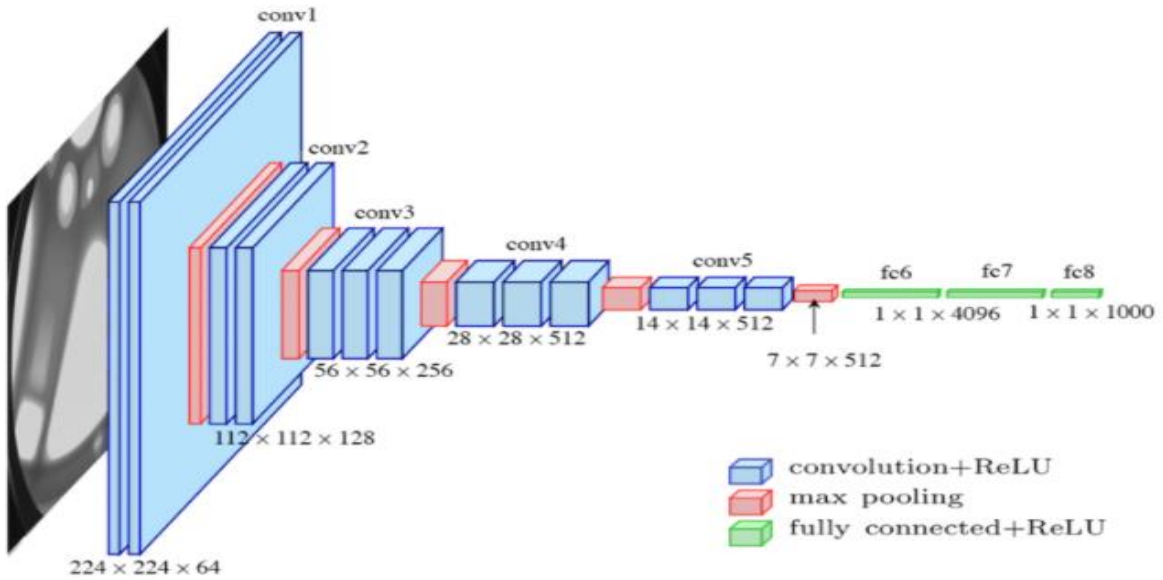
- **Inception V1 (GoogleNet) – 2014 Winner (top-5 error rate 6.67%)**
  - The v1 stands for 1st version and later there were further versions v2, v3, etc. It is also popularly known as GoogLeNet.
  - Deep with 22 layers.
  - Used multiple types of filter size, instead of being restricted to a single 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

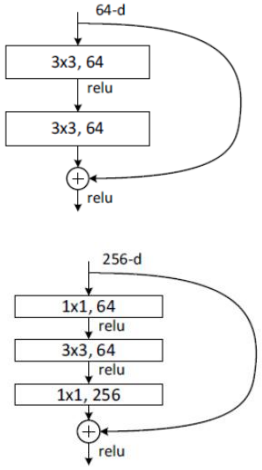
- **VGG-16 (University of Oxford) – 2014 Runners-Up (top-5 error rate 7.3%)**
  - Despite not winning the competition, VGG-16 architecture was appreciated and went on to become one of the most popular image classification models.
  - instead of using large-sized filters like AlexNet, it uses several 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.



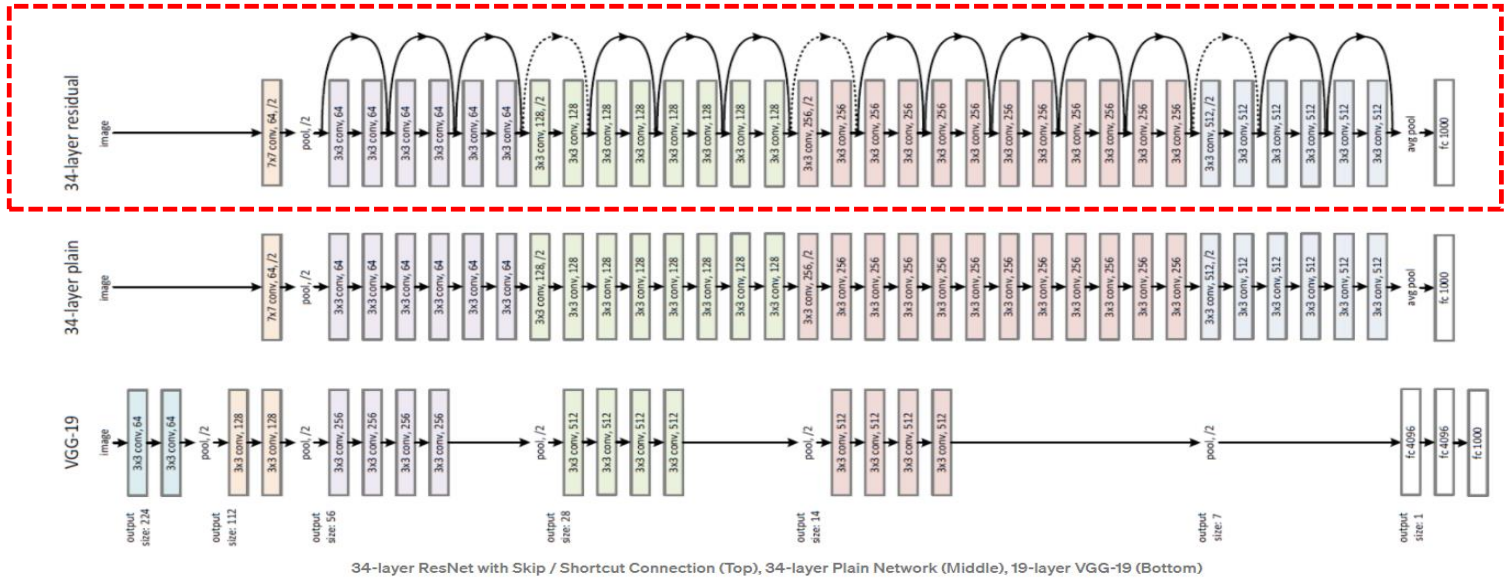
ImageNet Challenge (2014) – VGG-16 (Source)

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)

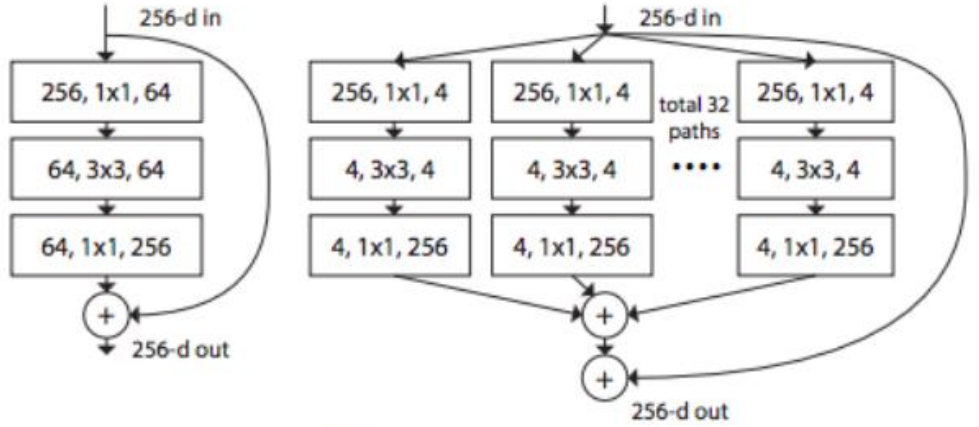


34-layer ResNet with Skip / Shortcut Connection (Top), 34-layer Plain Network (Middle), 19-layer VGG-19 (Bottom)

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

ILSVRC Winners

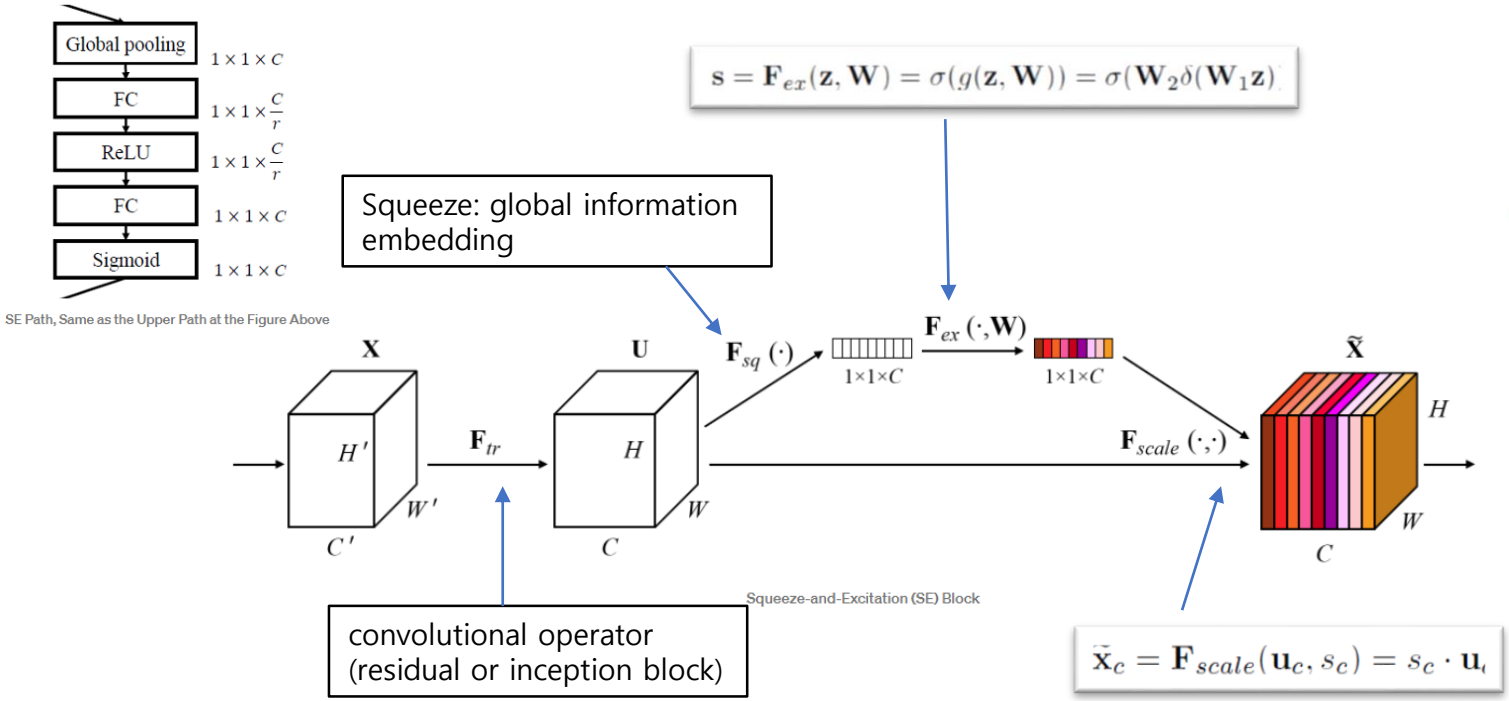
- **ResNeXt – 2016 Runners-Up (top-5 error rate 4.1%)**
  - Developed in the collaboration of the Researchers from UC San Diego and Facebook AI Research.
  - Inspired by (ResNet + VGG + Inception).
  - Still it became a popular model.
  - stacks the blocks and then use the ResNet approach of residual blocks. Here the hyper-parameters such as width and filters were also shared.

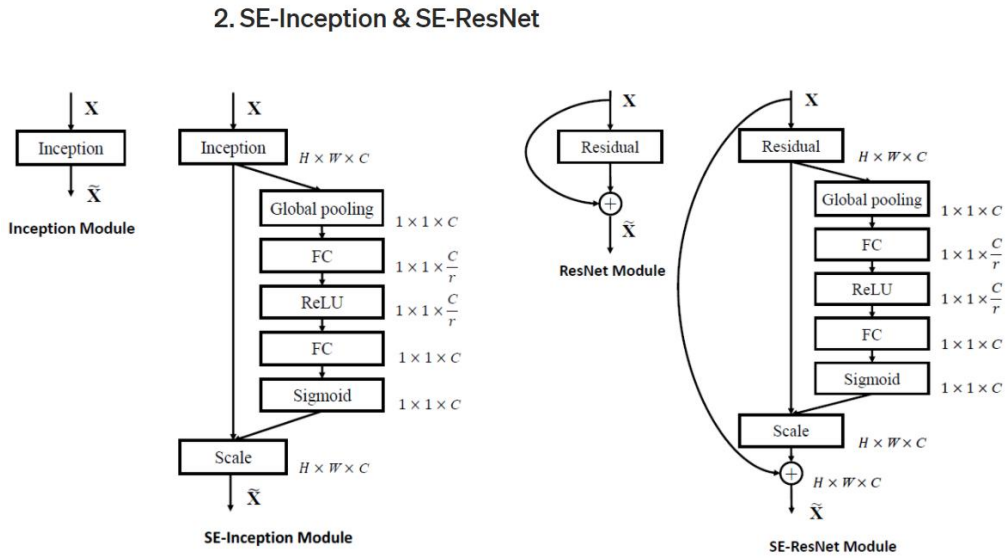


ImageNet Challenge (2016)- ResNeXt (Source)

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%).





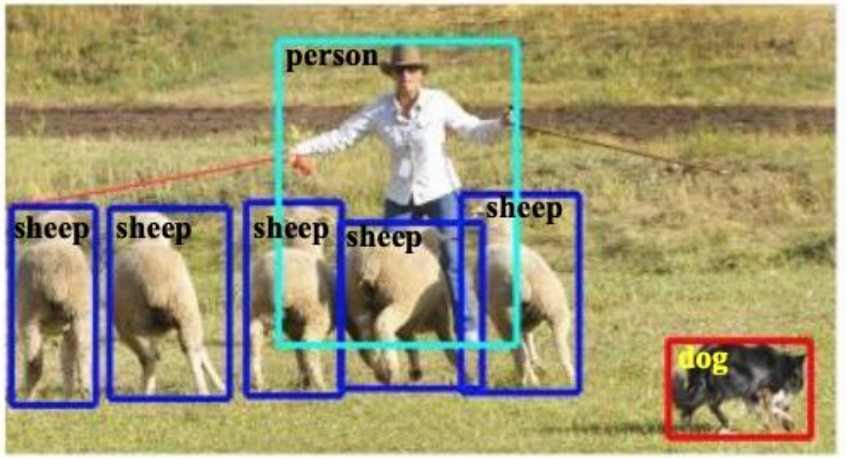
(*) Human Beings – Top-5 Error Rate – 5.1%

Computer Vision Tasks

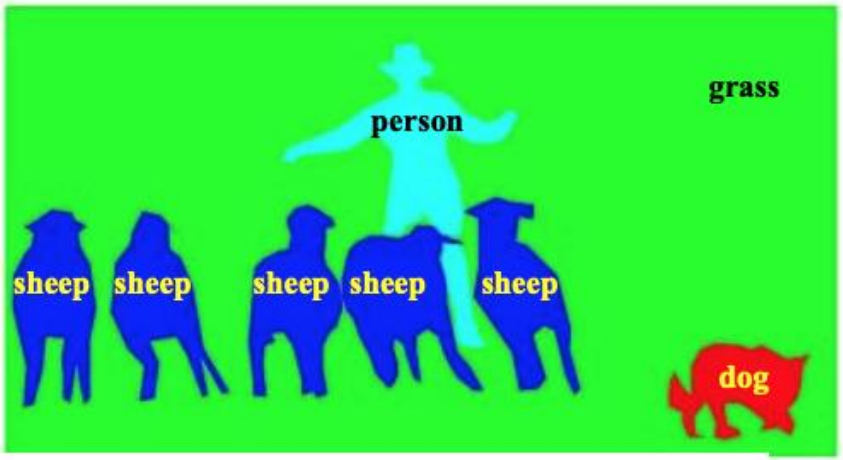
- Image Classification (이미지 분류)
  - 이미지에 어떤 객체들이 들어 있는지만 알아내는 것
  - 여기서는 이미지에 사람, 양, 개가 있다는 것을 찾아낸다
- Object Detection (객체 검출)
  - 찾아낸 객체의 위치까지 알아내는 것
  - 일반적으로는 객체가 있는 위치를 박스 형태 (Bounding Box)로 찾아낸다
- Segmentation (세그멘데이션 or 분할)
  - 객체의 위치를 박스형태가 아니라 비트 단위로 찾아낸다
  - 즉, 각 비트가 어떤 객체에 속하는지를 분류해내는 것
- Object 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

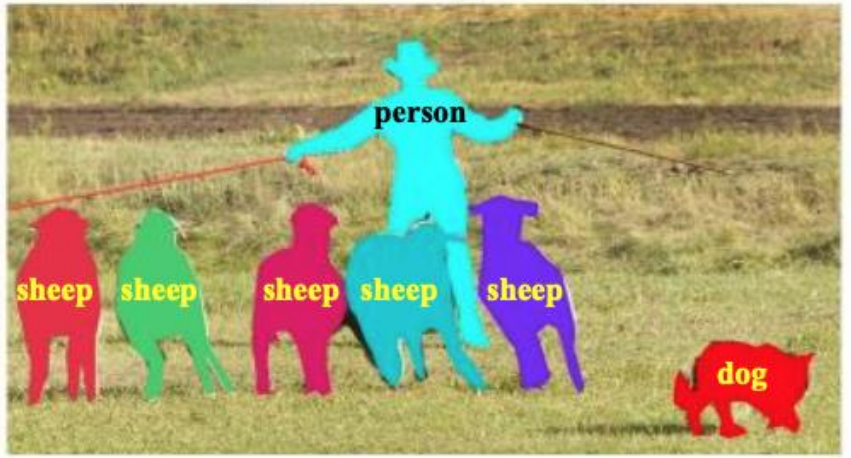
(a) Object Classification



(b) Generic Object Detection (Bounding Box)



(c) Semantic Segmentation



(d) Object Instance Segmetation

Computer Vision Tasks (Another View)

Classification

This image is CC0 public domain

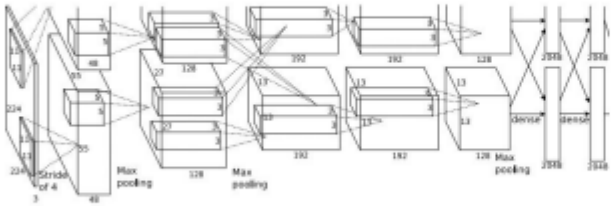


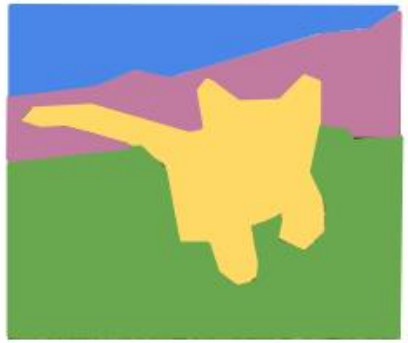
Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Vector: 4096

Fully-Connected: 4096 to 1000

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

Semantic Segmentation



GRASS, CAT, TREE, SKY

No objects, just pixels

Classification + Localization

CAT

Single Object

Object Detection

DOG, DOG, CAT

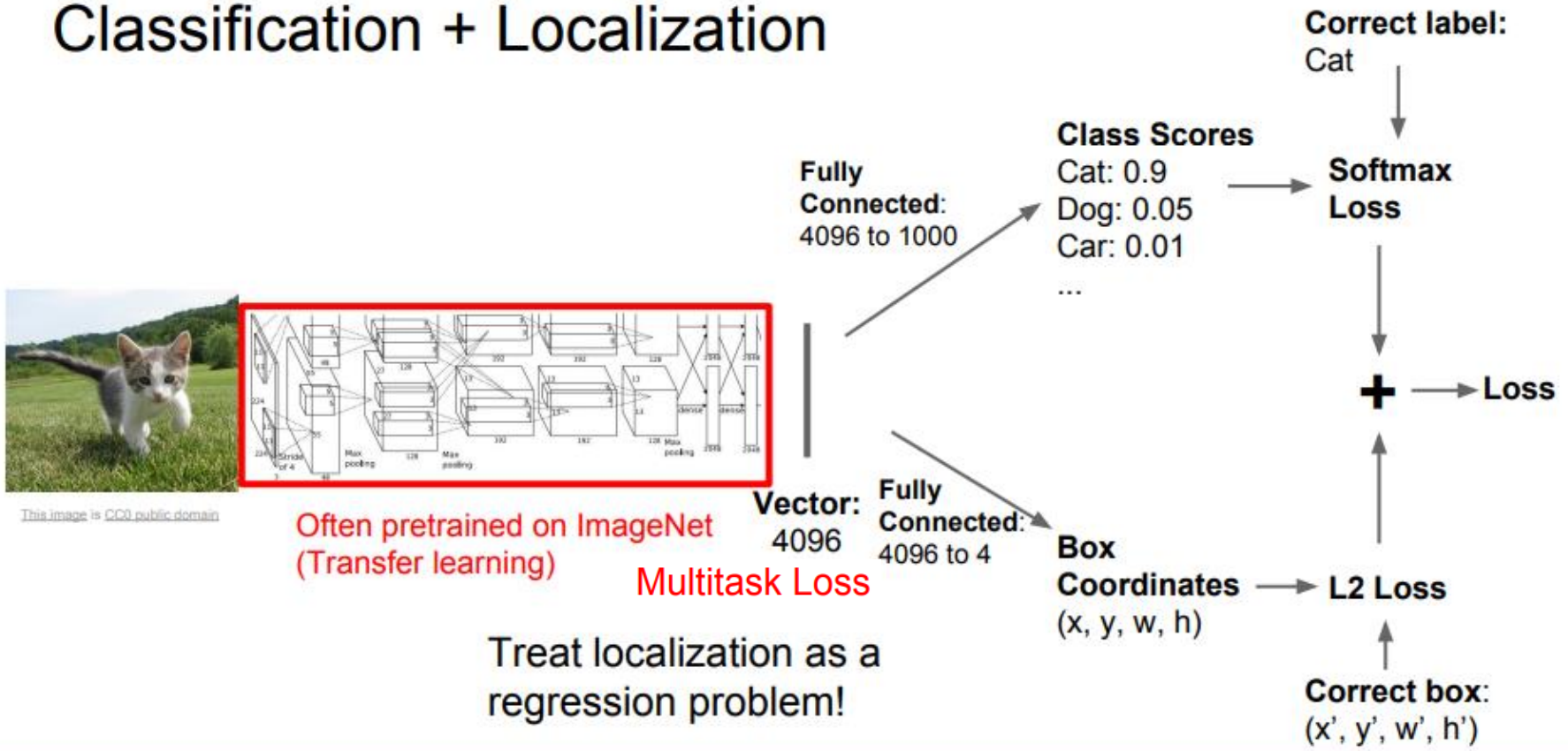
Instance Segmentation

DOG, DOG, CAT

Multiple Object

This image is CC0 public domain

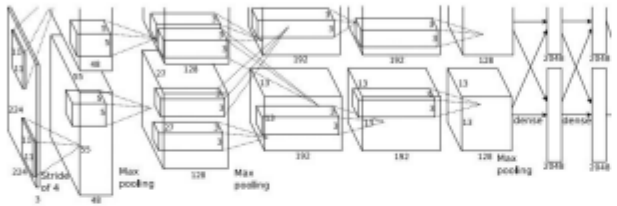
Classification + Localization

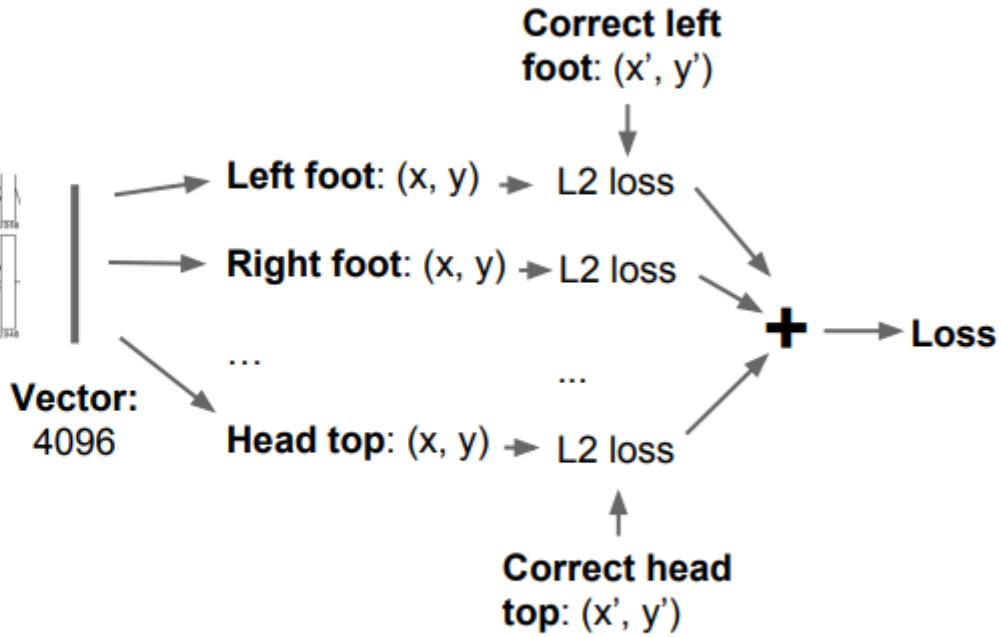


Aside: Human Pose Estimation

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



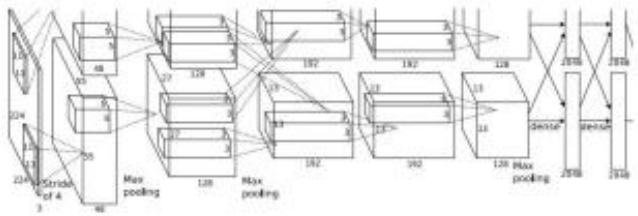


Toshev and Szegedy, "DeepPose: Human Pose Estimation via Deep Neural Networks", CVPR 2014

Object Detection (객체검출): Multi-objects

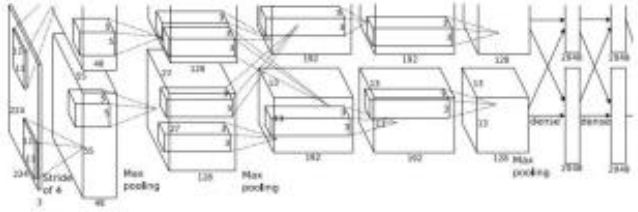
Object Detection as Regression?

Each image needs a different number of outputs!



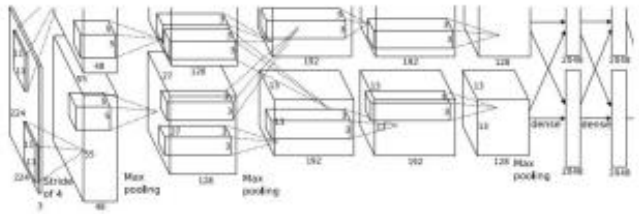
CAT: (x, y, w, h)

4 numbers



DOG: (x, y, w, h)
DOG: (x, y, w, h)
CAT: (x, y, w, h)

16 numbers



DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
....

Many numbers!

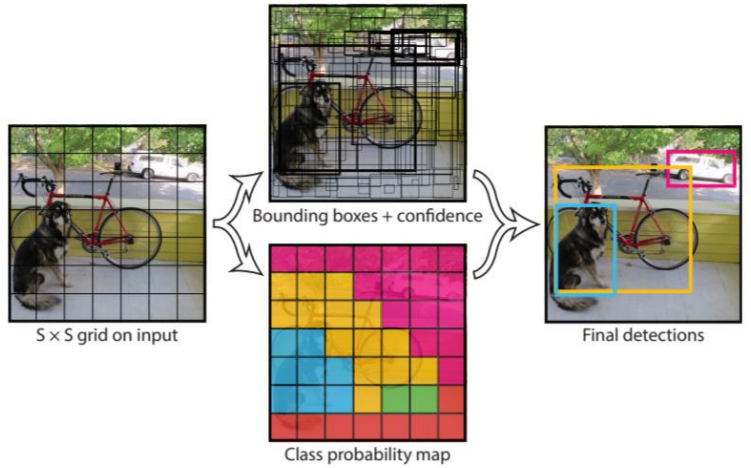
Object Detection

- 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

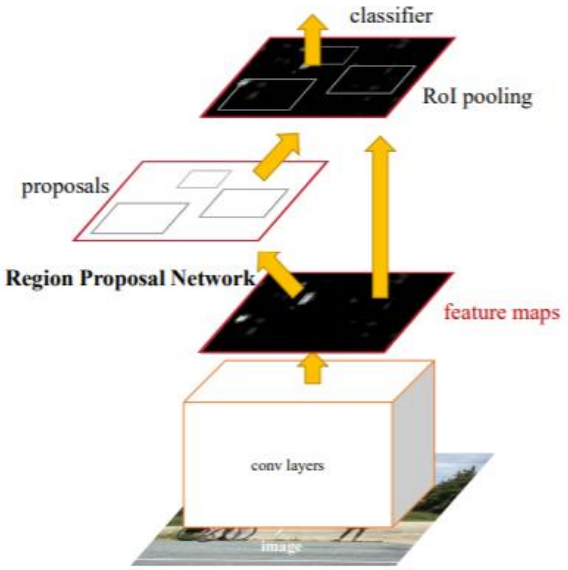
(자세 검출의 예)

Object Detection

- Object Detection:
  - Localization + Classification
- **One-stage Detector**
  - 위의 두 과정을 동시에 진행
  - YOLO (You Only Look Once), SSD, RetinaNet

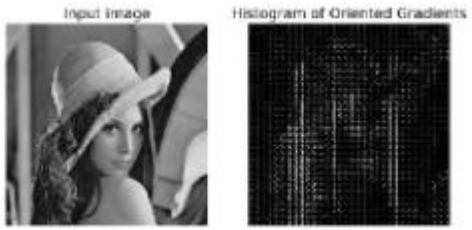


- **Two-stage Detector**
  - 위의 두 과정을 순차적으로 진행
  - R-CNN (Regions with CNN features), Faster R-CNN, DenseNet



Object Detection

- **Early approaches:**
  - Viola & Jones '01:
    - Haar Features + Adaboost + cascading classifier for fast rejectio
    - First competitive real-time object detection
    - Mainly used for face detection
  - Dalal & Triggs '05
    - Histogram of Oriented Gradients (HOG)
    - Support Vector Machines



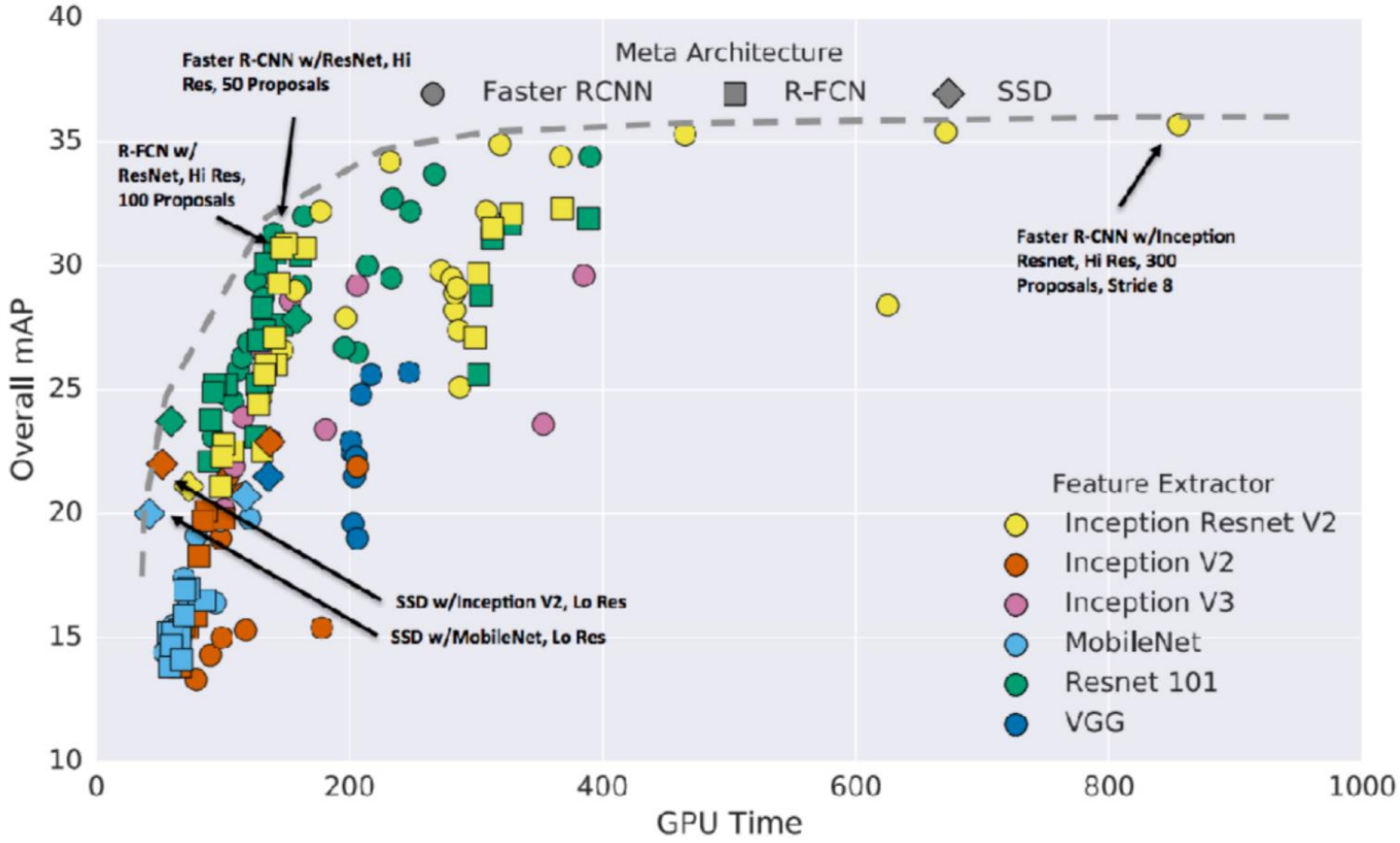
- **Modern approaches:**
  - Sliding Window: classify each possible window by CNN
  - Regional proposal CNN (R-CNN): find interesting image regions first, then classify by CNN
  - Single-Shot Multi-box Detectors: joint detection and classification
    - You Only Look Once (YOLO)
    - Single-Shot Multi-box 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-CNN 에 적용)
  - Sliding window 를 전체에 대해 하지 않고 물체가 있을 것 같은 영역에 대해서만 함. (스케일링 등이 필요 없어 빠름): Segmentation 이용.

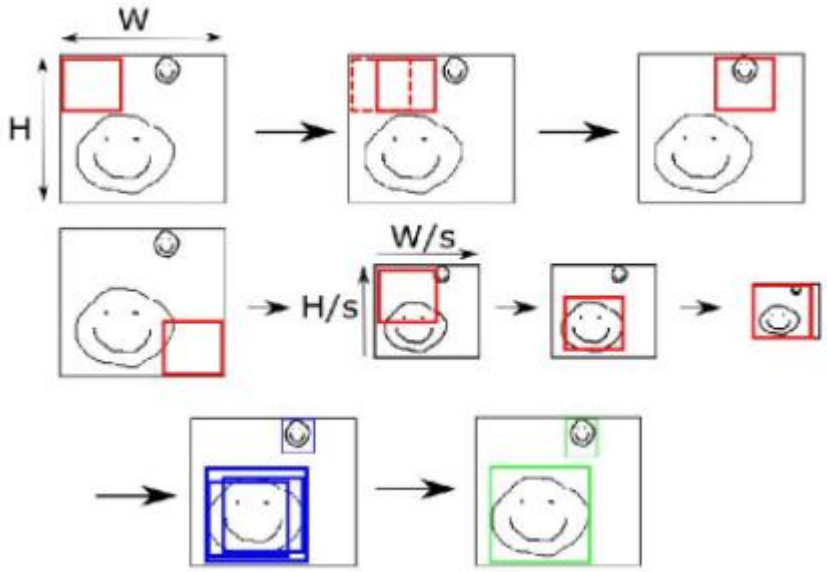
Object Detection - Algorithms

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



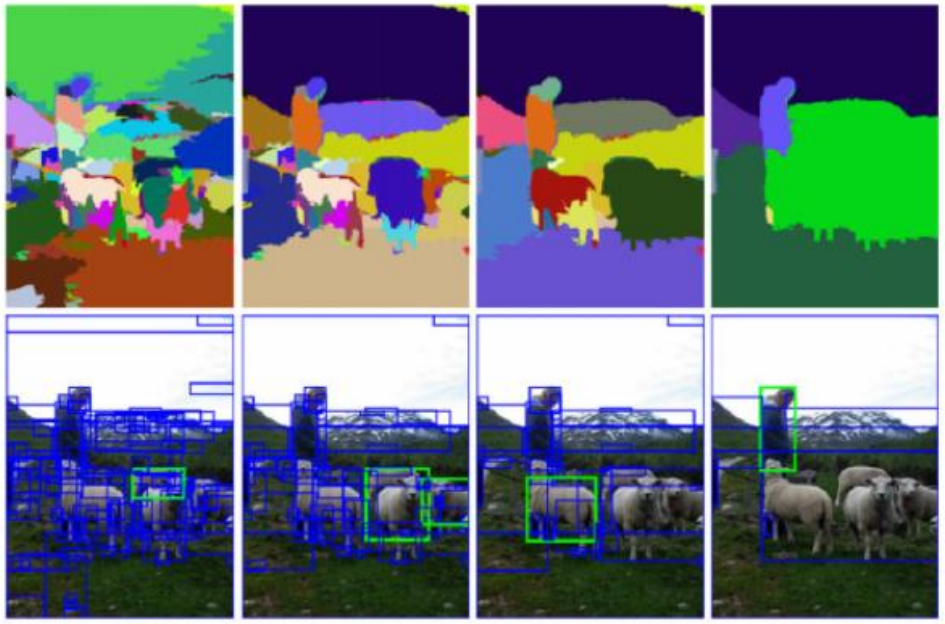
Object Detection (객체인식)

- **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



region size increased

Object Detection (객체인식)

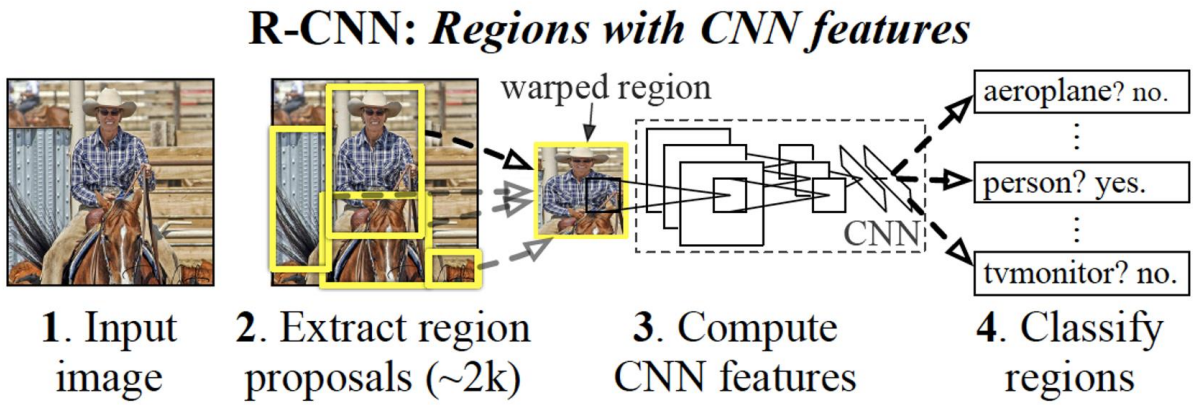
- **Sliding Window 방식**
  - 이미지를 일정한 크기의 사각형 윈도우로 잘라가며, 모든 위치와 크기에 대해 분류기를 적용
  - 예: 고양이를 찾는다면, 이미지 전체를 작은 격자처럼 훑으면서 각 윈도우가 고양이인지 아닌지 판별
  - 장점: 구현이 단순하고 직관적. 초기 객체 탐지 연구에서 널리 사용.
  - 단점:
    - 계산량이 매우 많음 → 이미지 크기가 크거나 객체 크기가 다양하면 윈도우 수가 폭발적으로 증가.
    - 다양한 크기와 비율의 객체를 커버하기 위해 여러 윈도우 크기를 시도해야 함.
    - 실제 응용에서는 속도가 느려 비효율적.
- **Region Proposal 방식**
  - Sliding Window처럼 모든 위치를 탐색하지 않고, 객체가 있을 가능성이 높은 영역만 후보로 제안.
  - 대표적인 방법: Selective Search (색상, 질감, 크기, 형태 등을 고려해 영역을 병합) (이후 R-CNN 계열에서는 Region Proposal Network (RPN)을 사용해 CNN 기반으로 후보 영역을 자동 생성)
  - 장점:
    - 탐색 공간을 크게 줄임 → 연산량 감소, 속도 향상,
    - 객체가 있을 법한 영역만 집중적으로 분석하므로 정확도가 높음.
  - 단점: Region Proposal을 생성하는 과정 자체가 추가 연산을 요구 (초기 Selective Search는 여전히 느렸음 → 이후 RPN으로 개선)

RPN(Region Proposal Network)

- **RPN의 핵심 개념**
  - 역할: 이미지 전체를 무작정 탐색하는 대신, CNN으로 추출한 feature map 위에서 객체 후보 영역을 제안
  - 입력: CNN이 만든 feature map
  - 출력: 객체가 있을 확률과 그 위치(경계 박스, bounding box).
- **작동 방식**
  - Feature Map 생성: 입력 이미지를 CNN에 통과시켜 특징 맵(feature map)을 얻는다.
  - Sliding Window 적용: 작은 네트워크가 feature map 위를 슬라이딩하며 각 위치를 검사
  - Anchor Boxes 생성: 각 위치마다 다양한 크기와 비율의 **Anchor Box**(기본 박스)를 미리 정의 (예: 3가지 크기 × 3가지 비율 = 9개의 anchor)
  - 분류(Classification): 각 anchor가 객체를 포함하는지 아닌지 확률을 예측
  - 회귀(Regression): anchor 박스를 실제 객체에 맞게 위치와 크기를 조정 (offset 값 학습)
  - Non-Maximum Suppression (NMS): 겹치는 후보 영역 중 가장 확률이 높은 것만 남겨 최종 Region Proposal을 생성.
- RPN은 Faster R-CNN의 핵심 모듈로, CNN feature map을 기반으로 객체 후보 영역을 자동으로 제안 (객체 탐지 속도와 정확도가 크게 향상)

R-CNN

- **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)

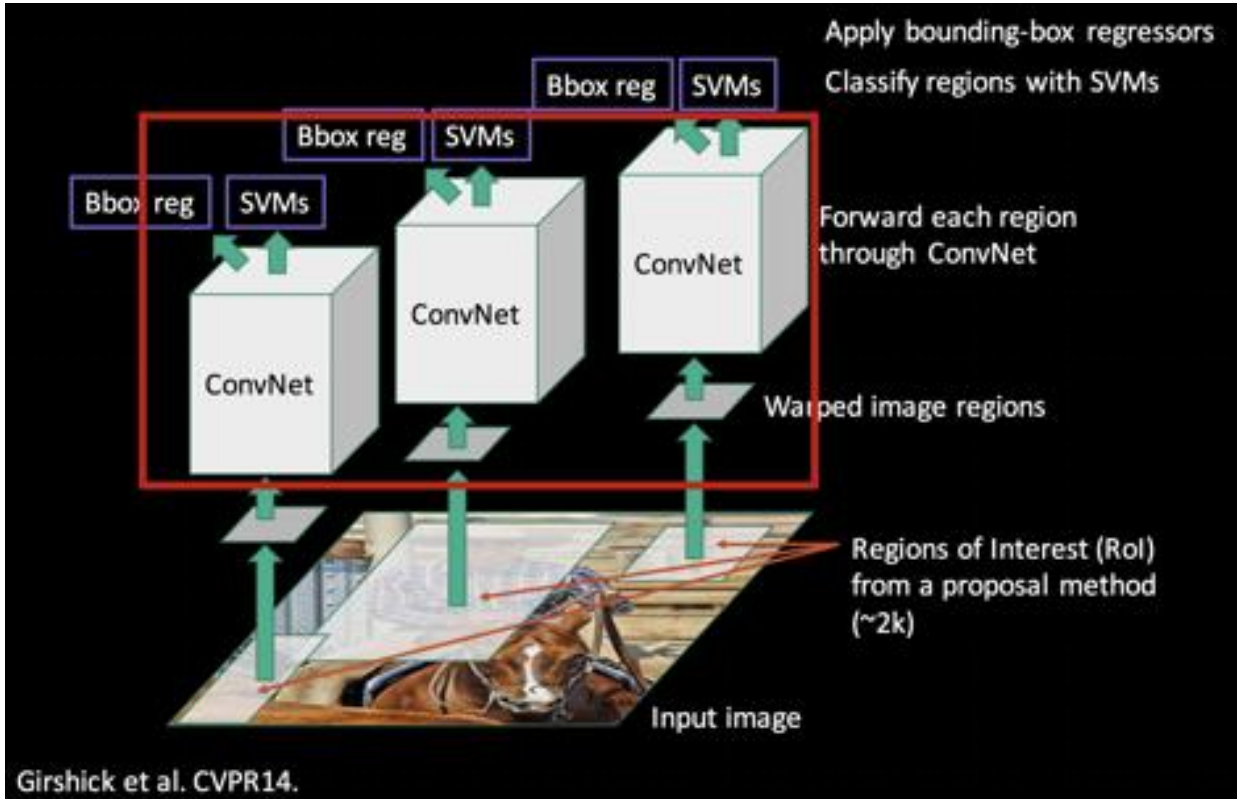


Task: → 1. Region Proposal → 2. Region Classification

R-CNN

- **R-CNN Modules**

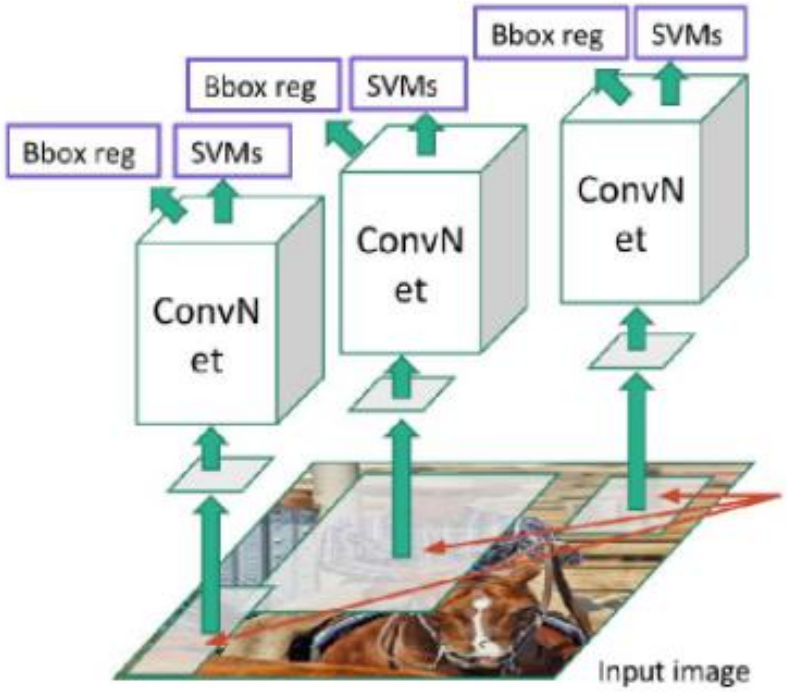
R-CNN 모델 구성 모듈	기능 및 설명
Region Proposal	카테고리와 무관하게 우선 객체가 있을 법한 영역(Region)을 찾는 모듈 – selective search
(사전 훈련된) CNN	찾은 영역(Region)에서 객체를 탐지하기 위해 특성 맵(Feature Map)을 추출하는 모듈
SVM	추출된 특성 맵을 분류하는 모듈 (객체를 식별하는 과정이다.)
Bounding Box Regression	회귀(Regression)를 활용해 해당 객체의 바운딩 박스를 표현하는 모듈



R-CNN (요약)

- **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



Linear Regression for bounding box offsets

Classify regions with SVMs

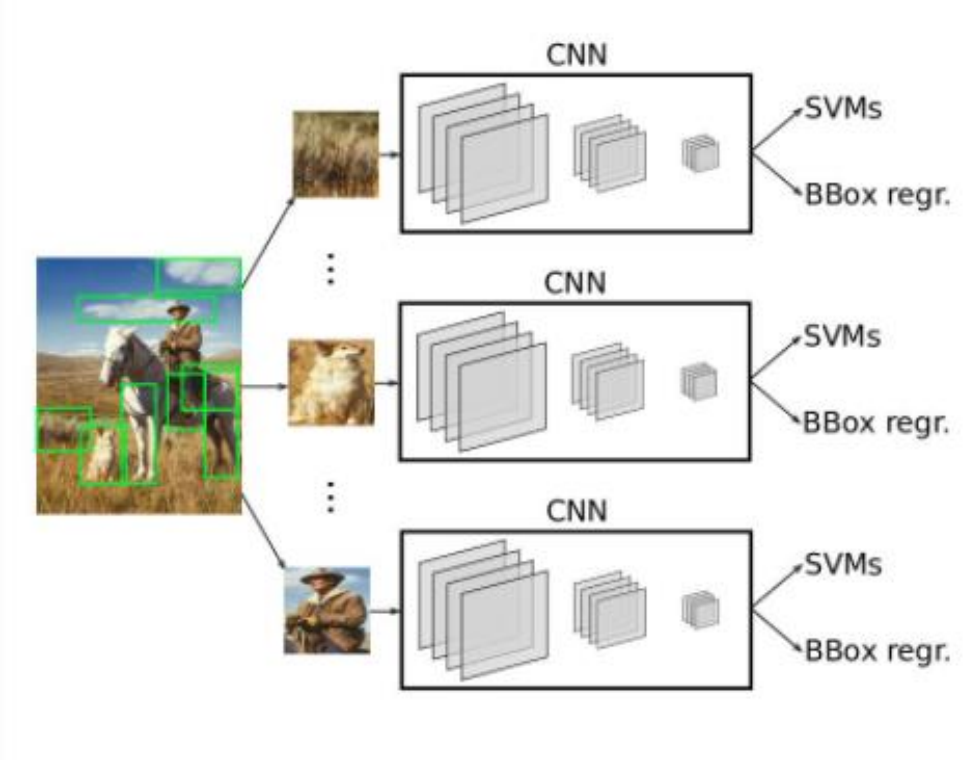
Forward each region through ConvNet

Warped image regions

Regions of Interest (RoI) from a proposal method (~2k)

Problems in R-CNN

- **Problems in R-CNN**

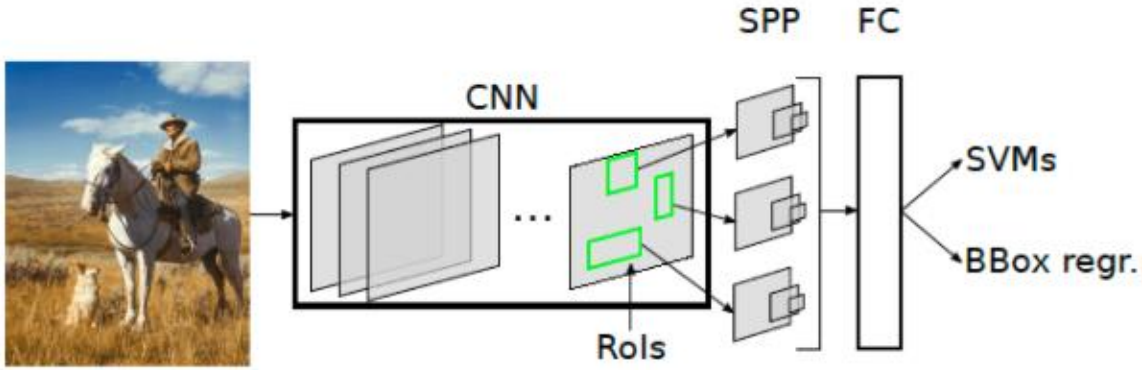


- R-CNN requires full forward pass for **each** region proposal
  - → Training is slow
  - → Inference is slow
- We can share computations when computing feature maps
- Why not use region proposals on feature maps?

- Slow
- Then, the key idea in order to improve the inference speed is that we **use the region proposals on the 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** 과정 (각 후보 영역을 feature map 위에서 잘라내고, 고정 크기로 변환)을 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)

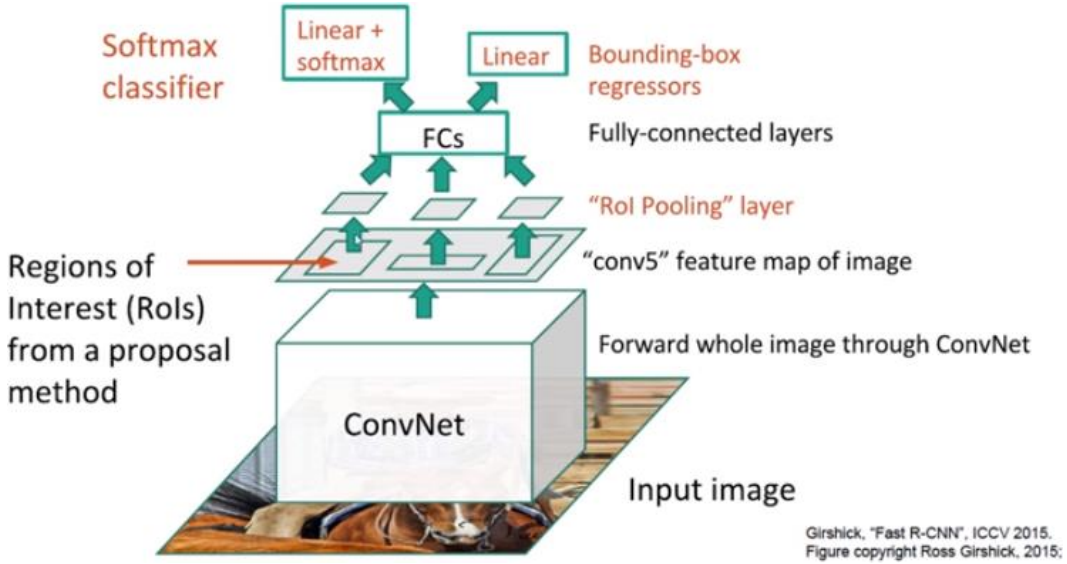
\+ Image-wise computation shared → Speed up by SPP during inference: ≈ 24–104×

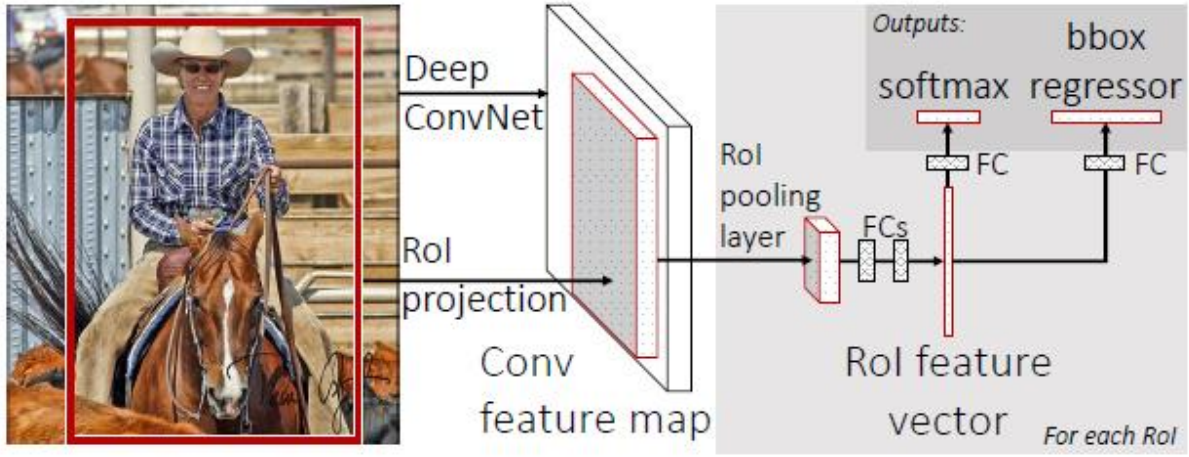
— R-CNN problems: **slow training** / not end-to-end

<Fast R-CNN의 구조>

1. 입력 이미지 → CNN → Feature Map 생성
   - 이미지 전체를 CNN에 통과시켜 하나의 feature map을 얻는다.
2. Region Proposal (외부 알고리즘)
   - Selective Search 같은 비학습적 방법으로 수천 개의 후보 영역(RoI)을 뽑는다.
3. RoI Pooling
   - 각 후보 영역을 feature map 위에서 잘라내고, 고정 크기로 변환한다.
4. 분류 + Bounding Box Regression
   - 각 RoI에 대해 어떤 객체인지 분류하고, 박스 좌표를 보정한다.

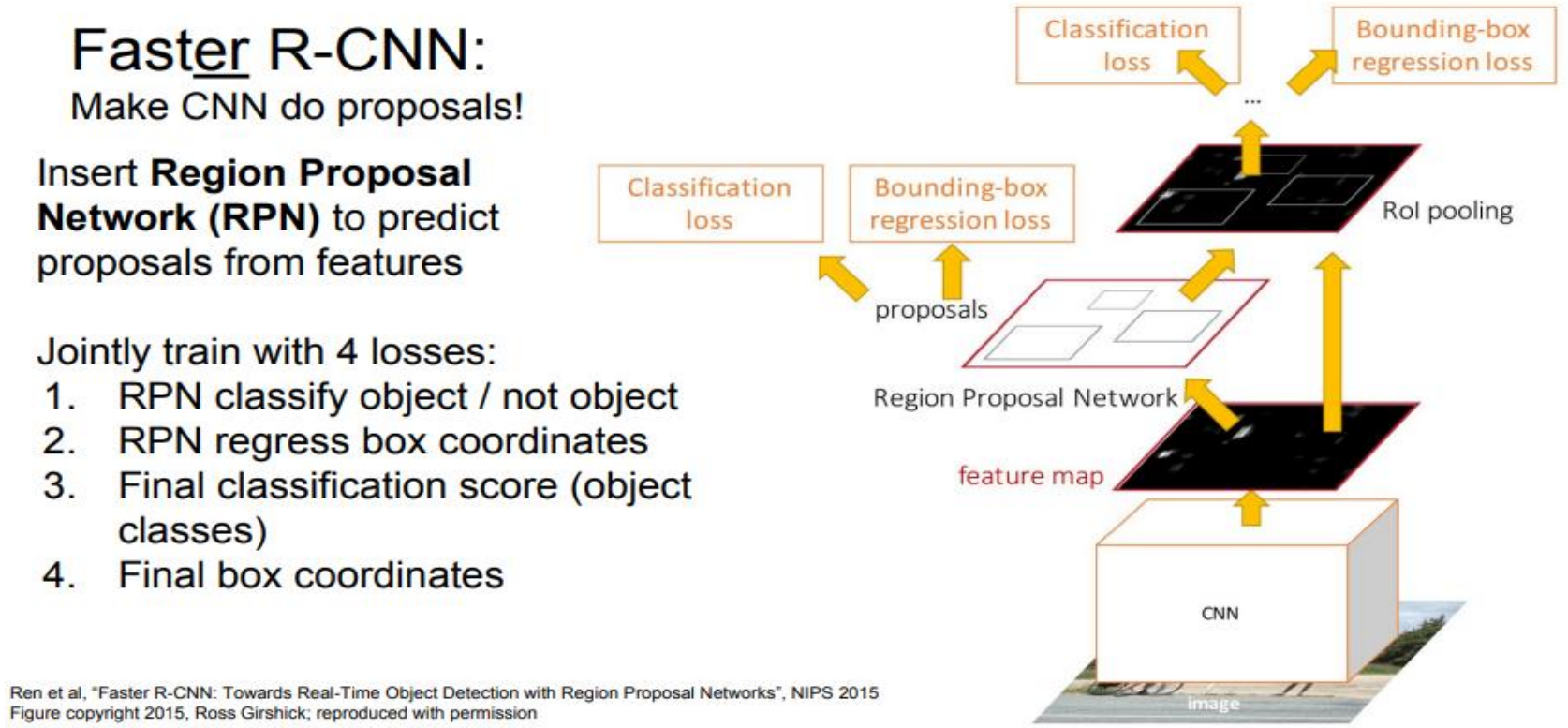
Fast R-CNN


Softmax classifier
Linear + softmax
Linear
Bounding-box regressors
FCs
Fully-connected layers
"RoI Pooling" layer
Regions of Interest (RoIs) from a proposal method
"conv5" feature map of image
Forward whole image through ConvNet
ConvNet
Input image
Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015;


Deep ConvNet
RoI projection
Conv feature map
RoI pooling layer
FCs
FC
FC
Outputs:
softmax
bbox regressor
RoI feature vector
For each RoI

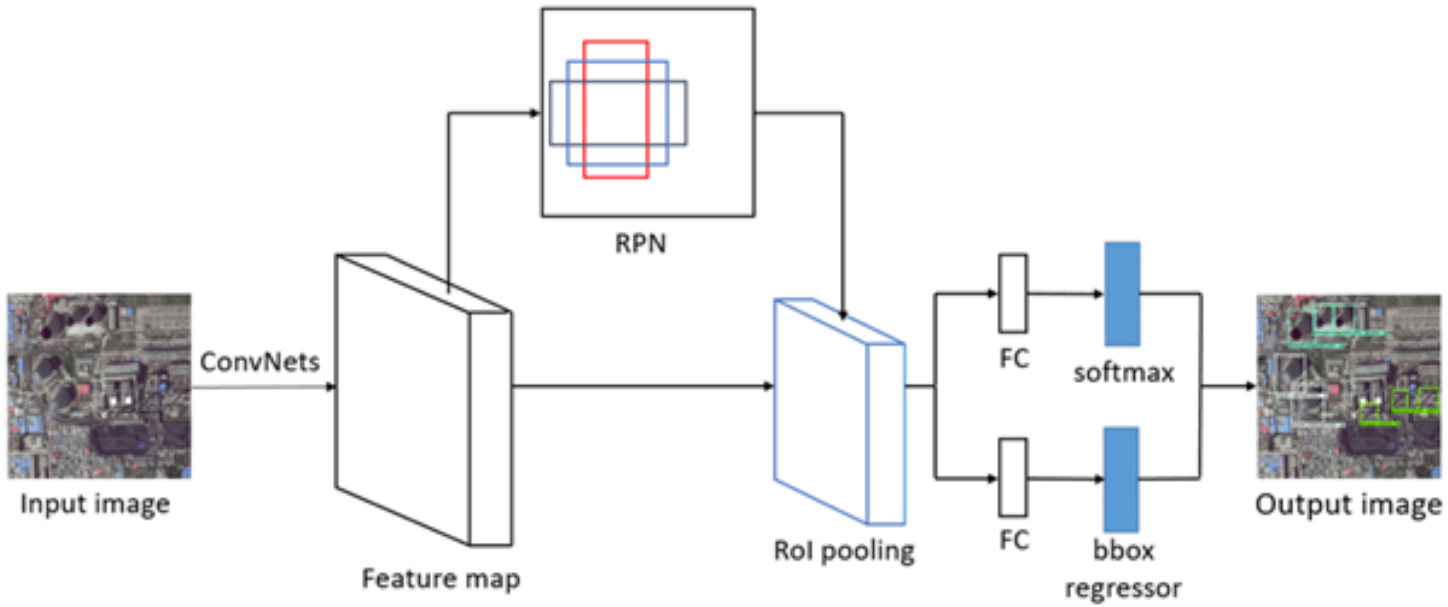
Faster R-CNN

- **Region Proposal Network(RPN)를 도입** → CNN feature map 위에서 객체 후보를 자동으로 제안.
  - 이때 RPN이 feature map 상에서 object가 있을 법한 영역을 학습으로 찾는다. (no need of Selective Search)



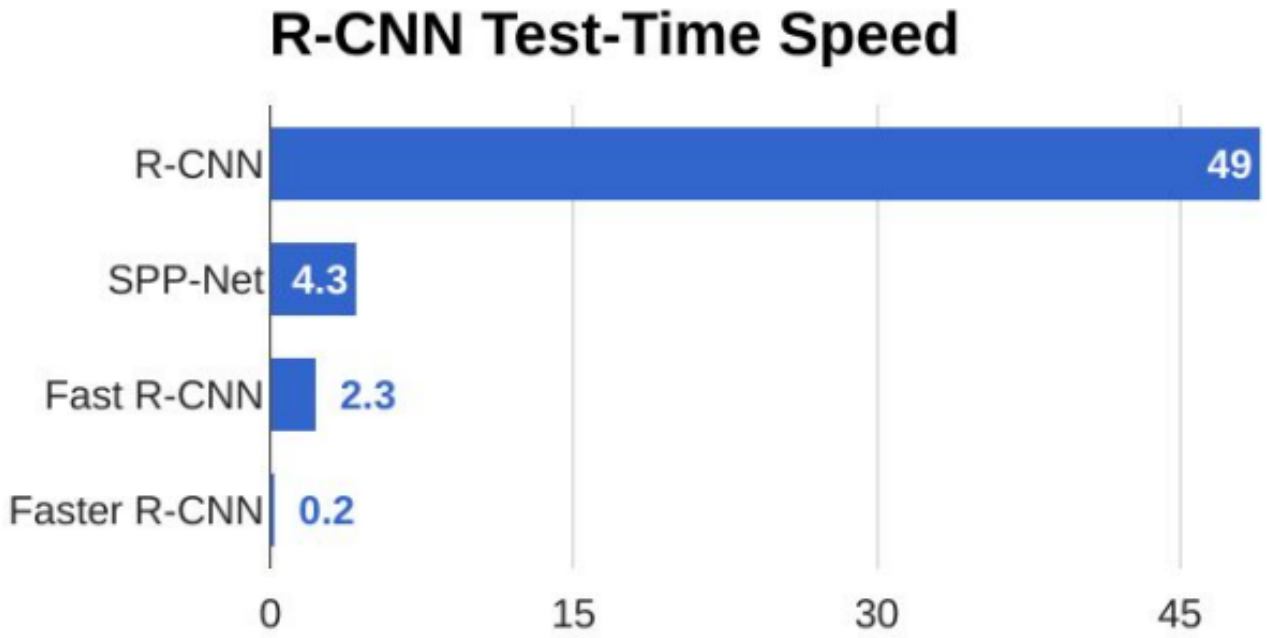
Faster R-CNN

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

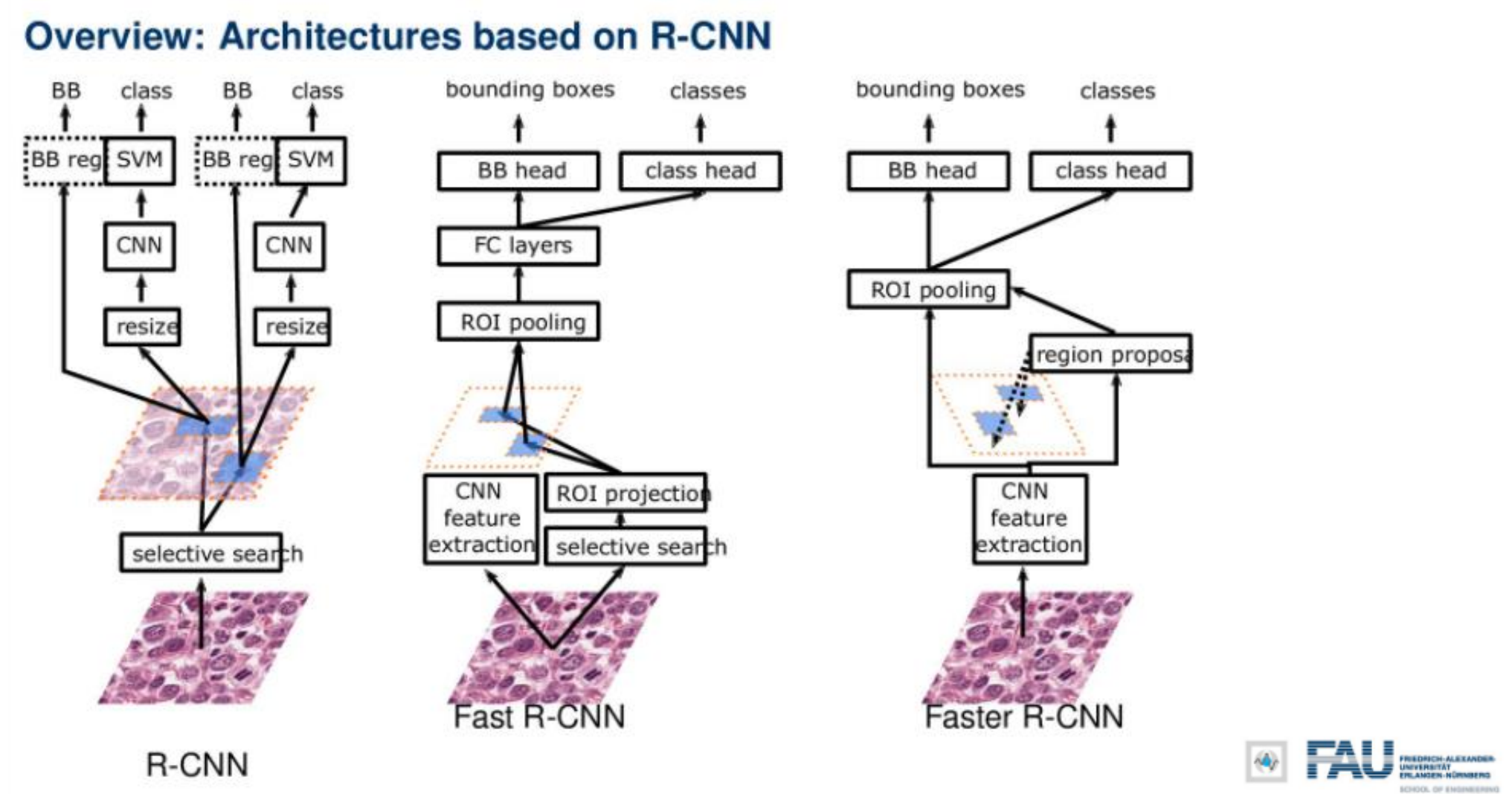


Faster R-CNN

- **Make CNN do proposals!**



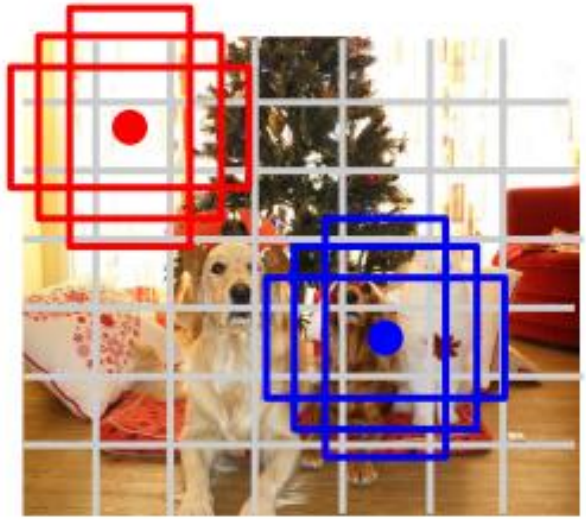
Architectures based on R-CNN



Detection without Proposals: YOLO / SSD

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

Input image
3 x H x W



Divide image into grid
7 x 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 x 7 x (5 * B + C)

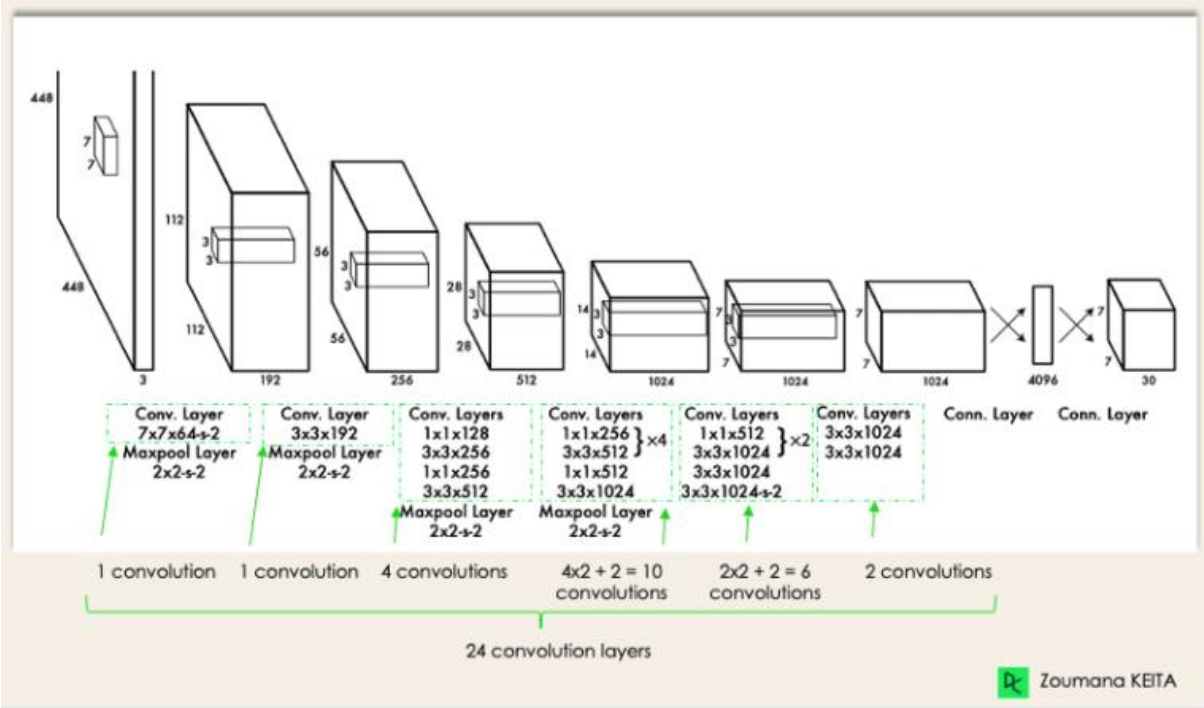
Redmon et al, "You Only Look Once: Unified, Real-Time Object Detection", CVPR 2016
Liu et al, "SSD: Single-Shot MultiBox Detector", ECCV 2016

YOLO(You Only Look Once)

- 이미지를 한 번만 CNN에 통과시켜 전체 객체의 위치와 종류를 동시에 예측하는 방식
- 기존 R-CNN 계열처럼 Region Proposal을 따로 생성하지 않고, 이미지를 격자로 나누어 각 격자에서 Bounding Box와 클래스 확률을 직접 예측하기 때문에 매우 빠른 속도 가능.
- YOLO의 핵심 아이디어
  - 단일 패스(single forward pass): 이름 그대로 "You Only Look Once". 이미지를 CNN에 한 번만 넣어 전체 객체를 탐지.
  - 그리드 분할(Grid-based prediction): 입력 이미지를 S×S 격자로 나누고, 각 격자 셀이 객체의 존재 여부, Bounding Box 좌표, 클래스 확률을 동시에 예측.
  - End-to-End 학습: Region Proposal 단계가 따로 없고, CNN이 바로 위치와 클래스까지 학습.
  - 속도: 실시간 탐지 가능(초당 수십 프레임). 자율주행, CCTV, 로봇 비전 등 실시간성이 중요한 분야에서 강점.
- YOLO의 구조
  - 입력 이미지 → CNN → Feature Map 생성
  - Feature Map을 격자 단위로 나눔.
  - 각 격자 셀에서 여러 개의 Bounding Box와 클래스 확률 예측 (tensor of S×S×(B×5+C)).
  - Non-Maximum Suppression(NMS)으로 겹치는 박스 중 가장 확률 높은 것만 남김.

YOLO(You Only Look Once)

- A good review site: A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS (https://arxiv.org/html/2304.00501v6)
- 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 SxS 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.
  - 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, $Pr(Class_i|Object)$

YOLO(You Only Look Once)

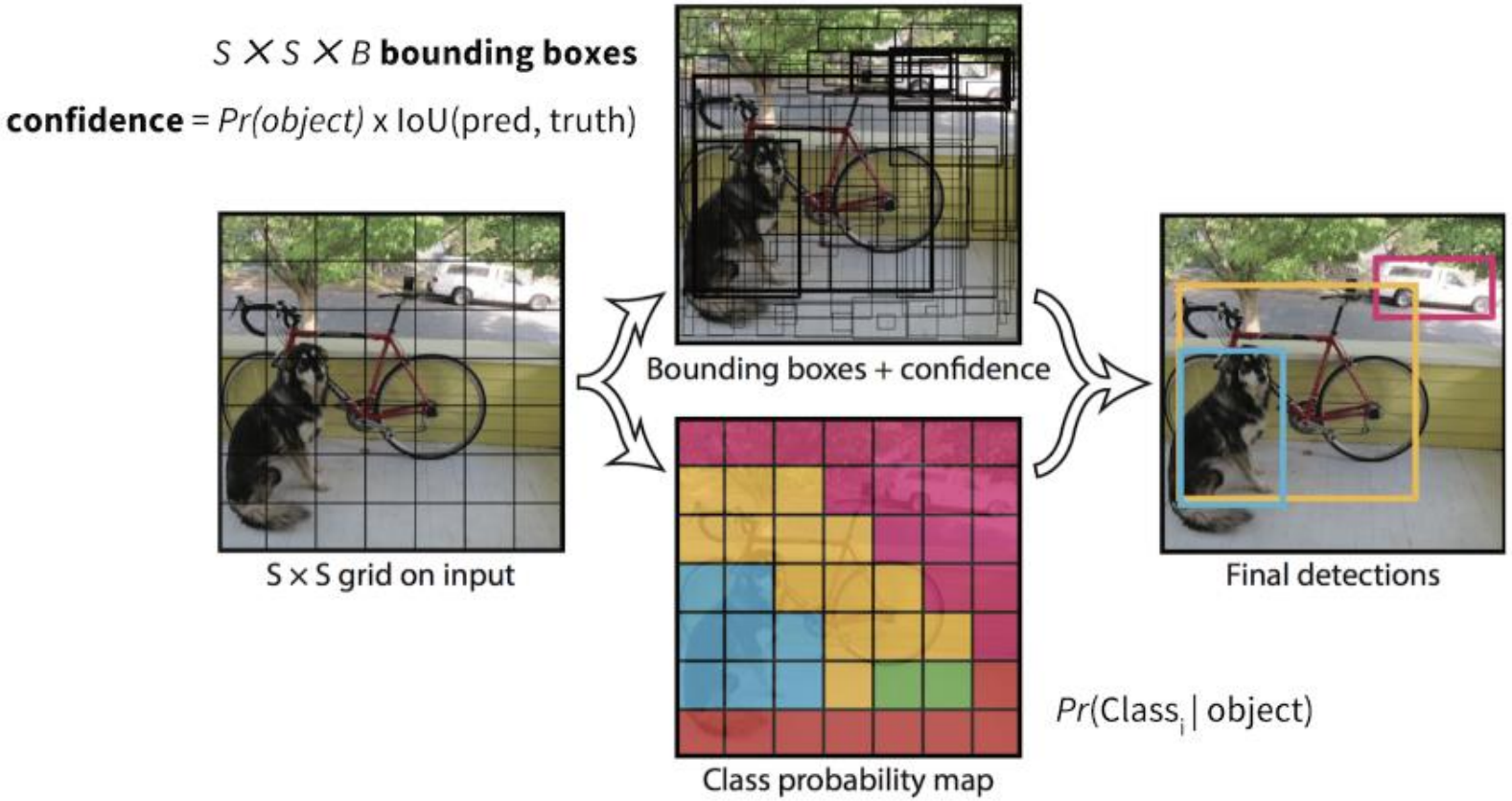


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

YOLO algorithm

단계	진행 내용	출력
1. 입력 이미지 전처리	이미지를 고정 크기(예: 416×416)로 변환함	CNN 입력용 텐서 데이터임
2. Pretrained CNN Backbone	Darknet, ResNet 등 사전학습된 CNN으로 특징 추출함	Feature map (예: 13×13×1024)임
3. 그리드 분할 및 박스 예측	Feature map을 S×S 그리드로 나눔, 각 셀이 바운딩 박스 좌표·객체 존재 확률·클래스 확률 예측함	바운딩 박스 + 확률 벡터임
4. 확률 결합	객체 존재 확률 × 클래스 확률을 곱해 최종 점수 계산함	특정 클래스일 확률 값임
5. 후처리 (NMS)	IoU 기준으로 겹치는 박스 중 가장 확률 높은 것만 남김	중복 제거된 최종 바운딩 박스 집합임
6. 최종 결과	각 객체의 클래스 라벨과 바운딩 박스 좌표 반환함	라벨 + 좌표 정보임

- **(2 단계): CNN Backbone**

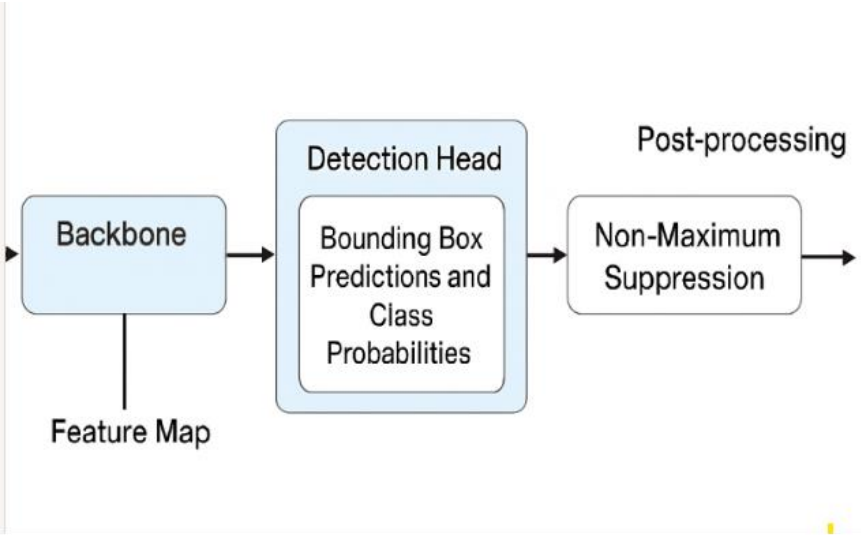
- **(3 단계): 추가 CNN**
- 각 그리드 셀은 B개의 anchor box(사전 정의된 박스 크기/비율)를 기반으로 예측을 수행
- 각 anchor box마다 다음을 출력 (바운딩 박스 좌표 (x,y,w,h), 객체 존재 확률(Objectness score), 클래스 확률 벡터 (예: [dog, cat, car])
- CNN의 마지막 레이어가 회귀(regression)와 분류(classification) 동시에 수행
- 출력 텐서 크기 예시: (S,S,B×(5+C)), 이때 C는 클래스 갯수, 5 = (x, y, w, h, objectness)

- **(4 단계): Post-processing**
- 각 anchor box의 **객체 존재 확률 × 클래스 확률**을 곱해 최종 점수를 계산 (예: objectness=0.9, class(dog)=0.8 → 최종 dog score=0.72)
- 모델 출력에서 각 박스별로 확률을 계산하고, 이 값들을 기반으로 **후처리 단계(NMS)**에 넘겨서 중복 박스를 제거.
- 결과적으로 "이 박스가 특정 클래스일 확률"이 산출됨

YOLO algorithm

YOLO 단계별 설명

단계	구성 요소	역할	출력
2단계 Backbone	Pretrained CNN (예: Darknet, ResNet)	입력 이미지를 특징 맵으로 변환함	Feature map (예: 13×13×1024)임
3단계 Detection Head	추가 CNN 레이어 (1×1 conv, 3×3 conv)	각 그리드 셀·anchor box마다 바운딩 박스 좌표, objectness, 클래스 확률 예측함	$(S, S, B \times (5 + C))$ 텐서임
4단계 Post-processing	수학적 연산 + NMS	objectness × 클래스 확률 결합, IoU 기반 중복 박스 제거함	최종 라벨 + 좌표 정보임



YOLO Loss 구성

구성 요소	설명	적용 대상
1. Localization Loss	예측한 바운딩 박스 좌표 (x, y, w, h)와 실제 박스 좌표의 차이	박스 예측값
2. Objectness Loss	해당 셀에 객체가 있는지 없는지에 대한 확률 예측 손실	객체 존재 확률
3. Classification Loss	객체가 있을 경우, 올바른 클래스인지에 대한 분류 손실	클래스 확률

(*) feature map 에서는 자체 loss를 직접 계산하지 않는다. 대신 Detection Head의 예측값에 대한 loss가 역전파되어 feature map을 생성한 Backbone까지 영향을 준다.

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 x 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, width, height):
        * (x, y) are relative to the grid cell, and width/height are relative to the image.
      - Predict the confidence score:
        * Confidence = Pr(Object) * IoU(predicted_box, ground_truth_box)
      - Predict the class probabilities for all C classes:
        * Class_Probabilities = [Pr(Class_1), Pr(Class_2), ..., Pr(Class_C)]

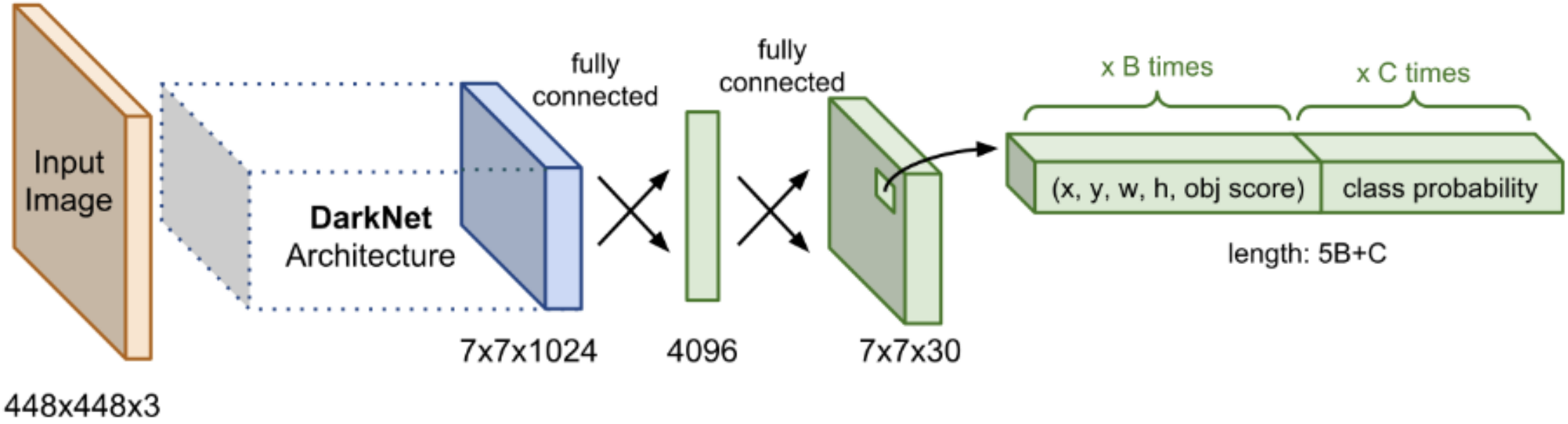
3. Adjust bounding box predictions:
   a. Use the anchor boxes as a starting point.
   b. During training, update the bounding box coordinates to better match the ground truth using backpropagation.

4. Post-processing:
   a. Compute the final detection scores for each anchor box:
      - Detection_Score = Confidence * 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 x S Grid: 이미지가 S x 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:

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

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

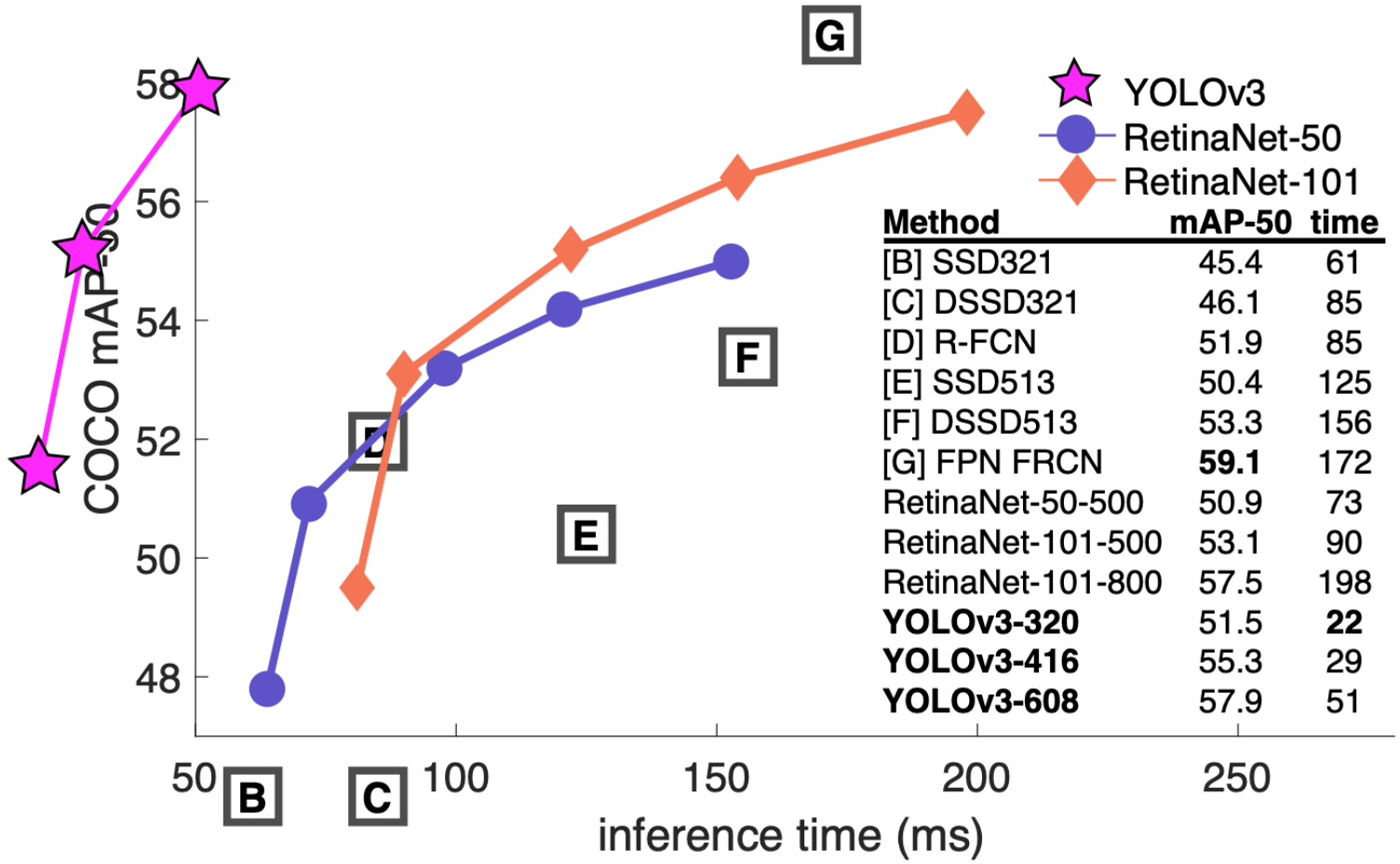
$$\mathcal{L} = \mathcal{L}_{\text{loc}} + \mathcal{L}_{\text{cls}}$$

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-v3 (Darknet-53): 53 convolutional layers

YOLO (You Only Look Once)

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



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

YOLO (You Only Look Once)

- 객체 인식(classification)과 위치 예측(localization)을 동시에 하나의 신경망으로 수행
  - 즉, 객체 예측과 경계 박스를 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 (7x7x30 = 1470 벡터 출력)

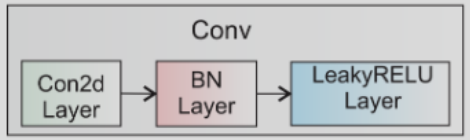
- C로 구현된 DarkNet을 framework로 사용

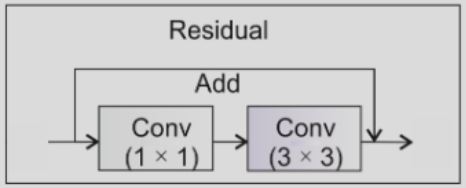
- 참고: https://github.com/qqwweee/keras-yolo3
- 동작 데모 : https://pjreddie.com/darknet/yolo/#demo

YOLOv3

- Backbone Network (Darknet-53)
  - 53 convolutional layers (each with batch normalization, Leaky Relu)
  - The following figure shows only the backbone; it does not include the detection head composed of multi-scale predictions.

Layer	Filters	size	Repeat		Output size
Image					416 × 416
Conv	32	3 × 3/1	1		416 × 416
Conv	64	3 × 3/2	1		208 × 208
Conv	32	1 × 1/1	Conv		208 × 208
Conv	64	3 × 3/1	Conv	×1	208 × 208
Residual			Residual		208 × 208
Conv	128	3 × 3/2	1		104 × 104
Conv	64	1 × 1/1	Conv		104 × 104
Conv	128	3 × 3/1	Conv	×2	104 × 104
Residual			Residual		104 × 104
Conv	256	3 × 3/2	1		52 × 52
Conv	128	1 × 1/1	Conv		52 × 52
Conv	256	3 × 3/1	Conv	×8	52 × 52
Residual			Residual		52 × 52
Conv	512	3 × 3/2	1		26 × 26
Conv	256	1 × 1/1	Conv		26 × 26
Conv	512	3 × 3/1	Conv	×8	26 × 26
Residual			Residual		26 × 26
Conv	1024	3 × 3/2	1		13 × 13
Conv	512	1 × 1/1	Conv		13 × 13
Conv	1024	3 × 3/1	Conv	×4	13 × 13
Residual			Residual		13 × 13

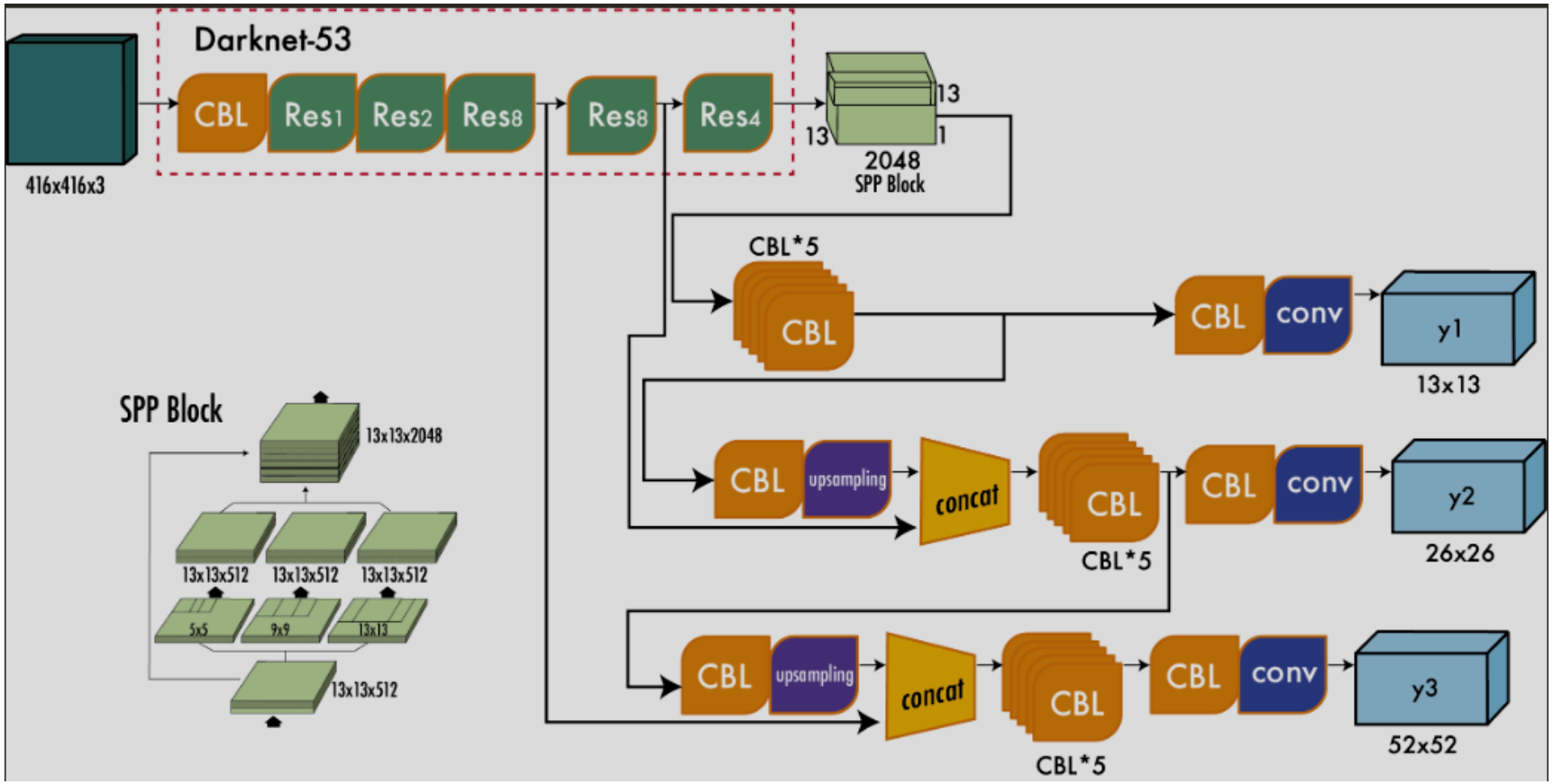




https://arxiv.org/html/2304.00501v6

YOLOv3

- YOLOv3 Multi-scale detection architecture
  - For the COCO dataset with 80 categories, each scale provides an output tensor with a shape of N×N×[3×(4+1+80)], where N×N is the size of the feature map (or grid cell), the 3 indicates the boxes per cell and the 4+1 include the four coordinates and the objectness score.

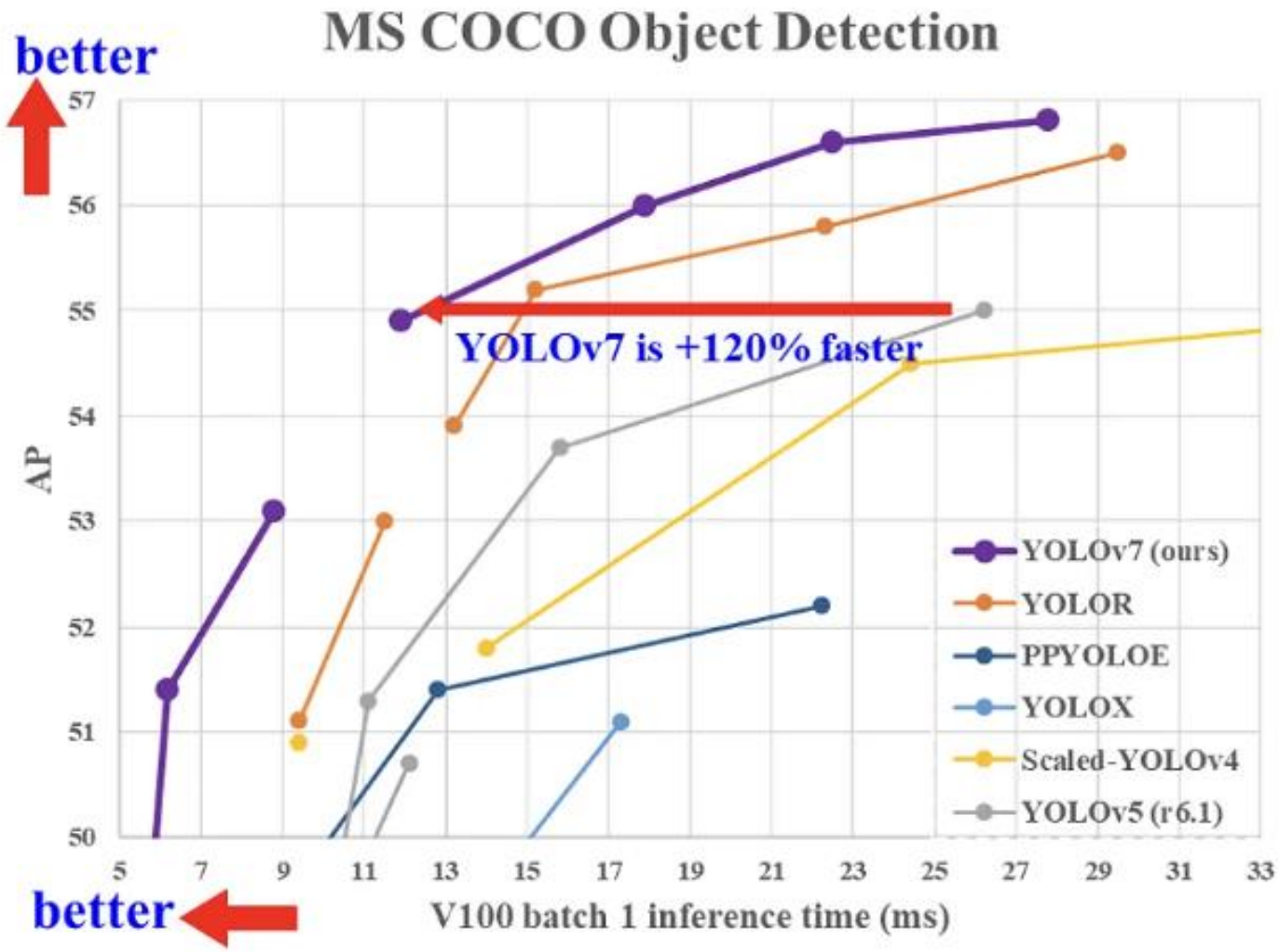


(*) CBL(Convolution-BatchNorm-Leaky ReLU)

3 different sized outputs

https://arxiv.org/html/2304.00501v6

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



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

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

Version	Date	Anchor	Framework	Backbone	AP (%)
YOLO	2015	No	Darknet	Darknet24	*63.4*
YOLOv2	2016	Yes	Darknet	Darknet24	*78.6*
YOLOv3	2018	Yes	Darknet	Darknet53	**33.0**
YOLOv4	2020	Yes	Darknet	CSPDarknet53	**43.5**
YOLOv5	2020	Yes	Pytorch	YOLOv5CSPDarknet	**55.8**
PP-YOLO	2020	Yes	PaddlePaddle	ResNet50-vd	**45.9**
Scaled-YOLOv4	2021	Yes	Pytorch	CSPDarknet	**56.0**
PP-YOLOv2	2021	Yes	PaddlePaddle	ResNet101-vd	**50.3**
YOLOR	2021	Yes	Pytorch	CSPDarknet	**55.4**
YOLOX	2021	No	Pytorch	YOLOXCSPDarknet	**51.2**
PP-YOLOE	2022	No	PaddlePaddle	CSPRepResNet	**54.7**
YOLOv6	2022	No	Pytorch	EfficientRep	**52.5**
YOLOv7	2022	No	Pytorch	YOLOv7Backbone	**56.8**
DAMO-YOLO	2022	No	Pytorch	MAE-NAS	**50.0**
YOLOv8	2023	No	Pytorch	YOLOv8CSPDarknet	**53.9**
YOLO-NAS	2023	No	Pytorch	NAS	**52.2**

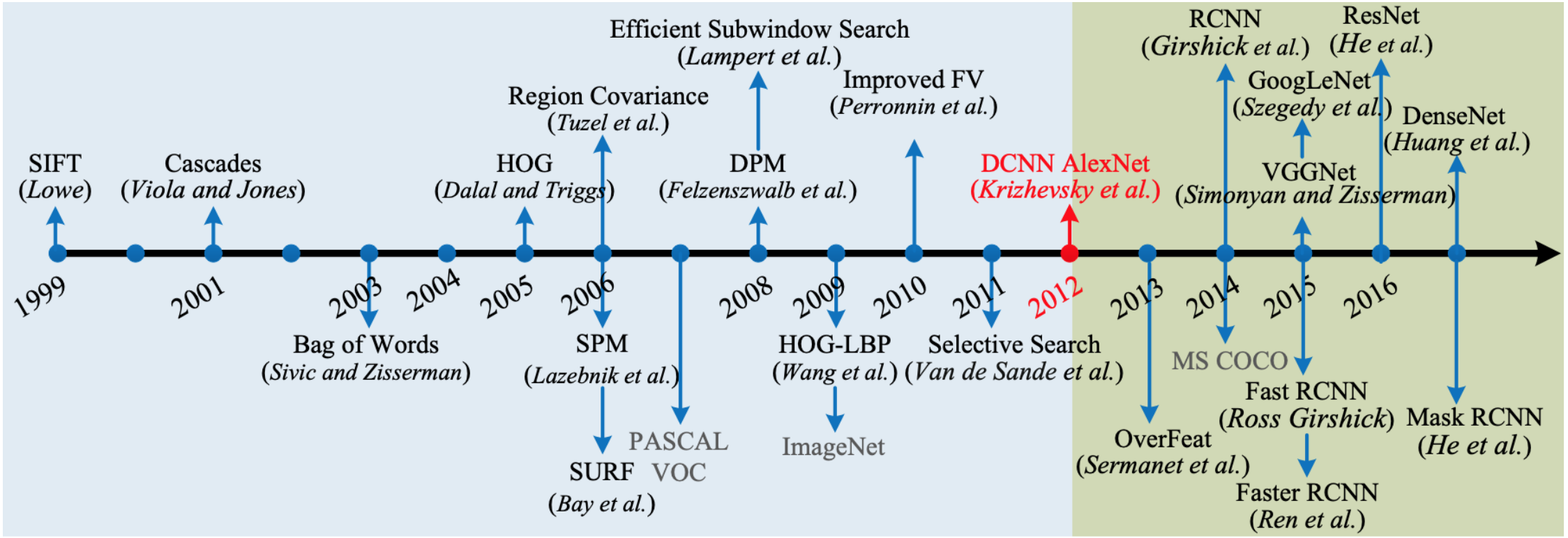
https://arxiv.org/html/2304.00501v6

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)

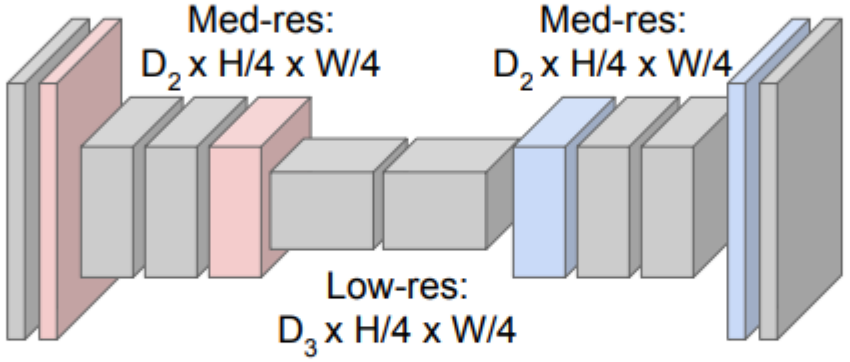
Semantic Segmentation Idea: Fully Convolutional

Downsampling:
Pooling, strided convolution

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

Upsampling:
Unpooling or strided transpose convolution

Input:
3 x H x W



High-res:
D_1 x H/2 x W/2

High-res:
D_1 x H/2 x W/2

Predictions:
H x W

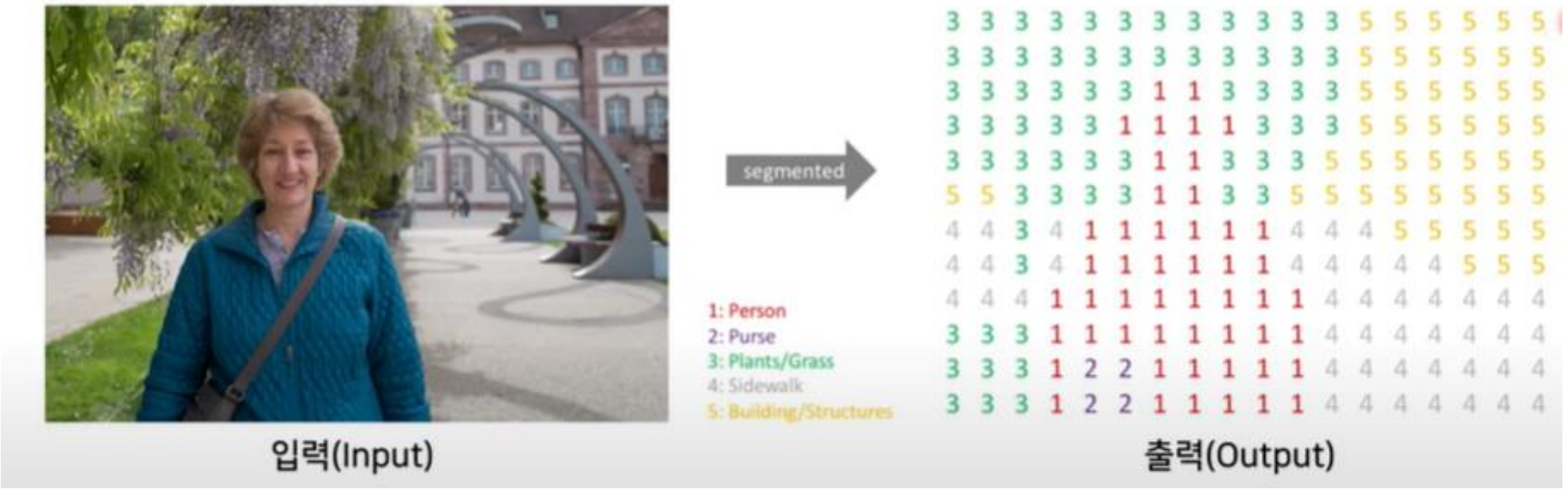
Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

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

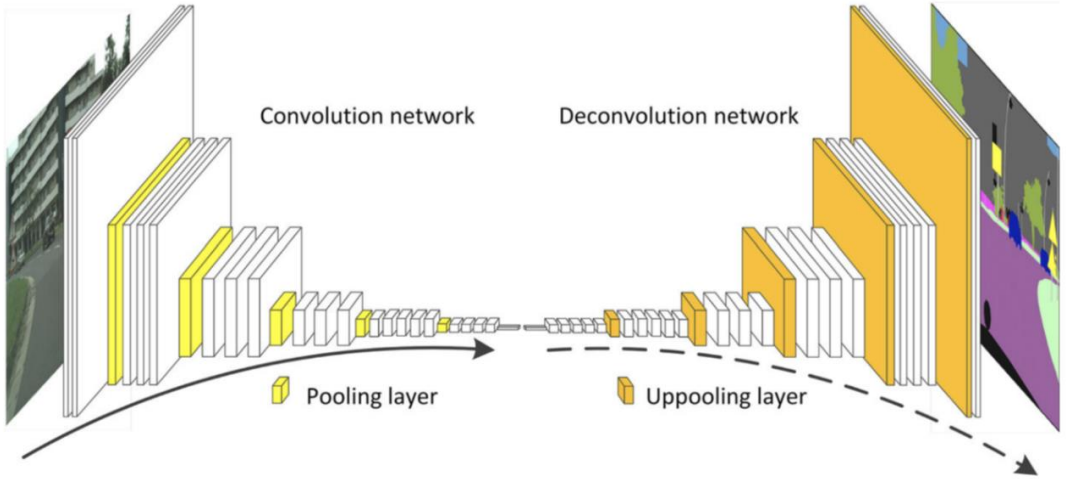
Semantic Segmentation (분할)

- 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)



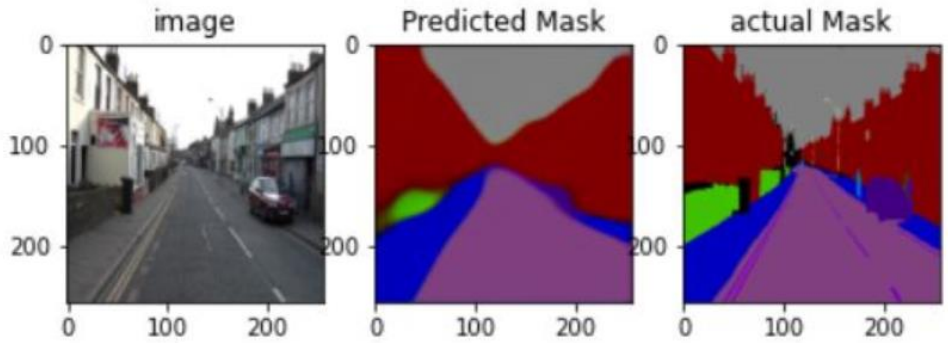


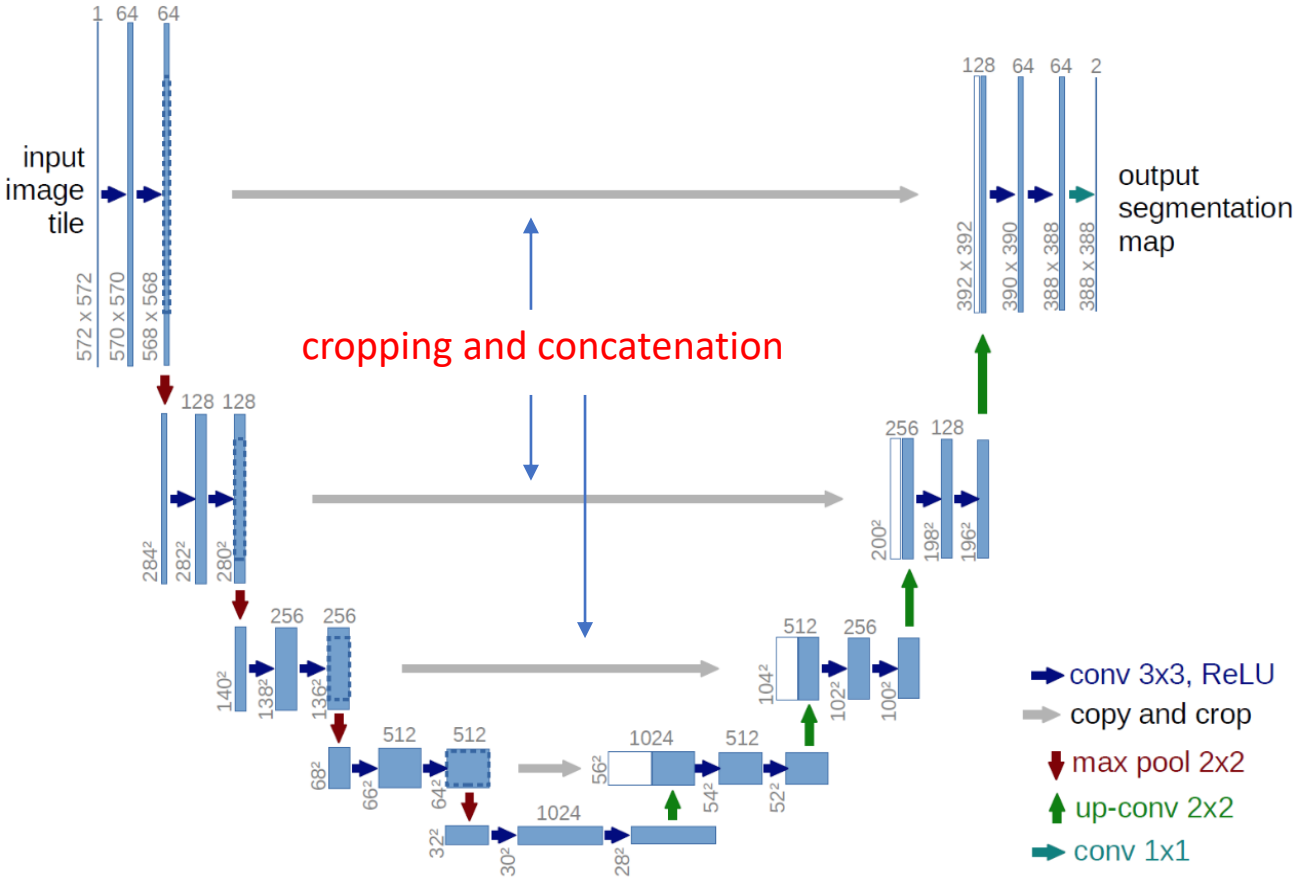
Figure 4: Auto-Encoder Results for Semantic Segmentation

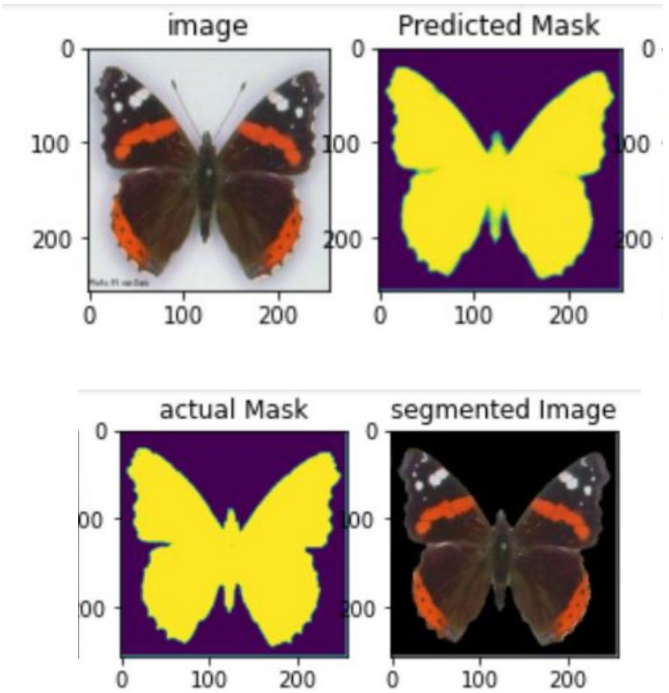
- Seems ok, but not good enough because of Information loss in feature maps
- Need connection between the encoder and the decoder -> U-Net

https://www.analyticsvidhya.com/blog/2021/01/auto-encoders-for-computer-vision-an-endless-world-of-possibilities/

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)

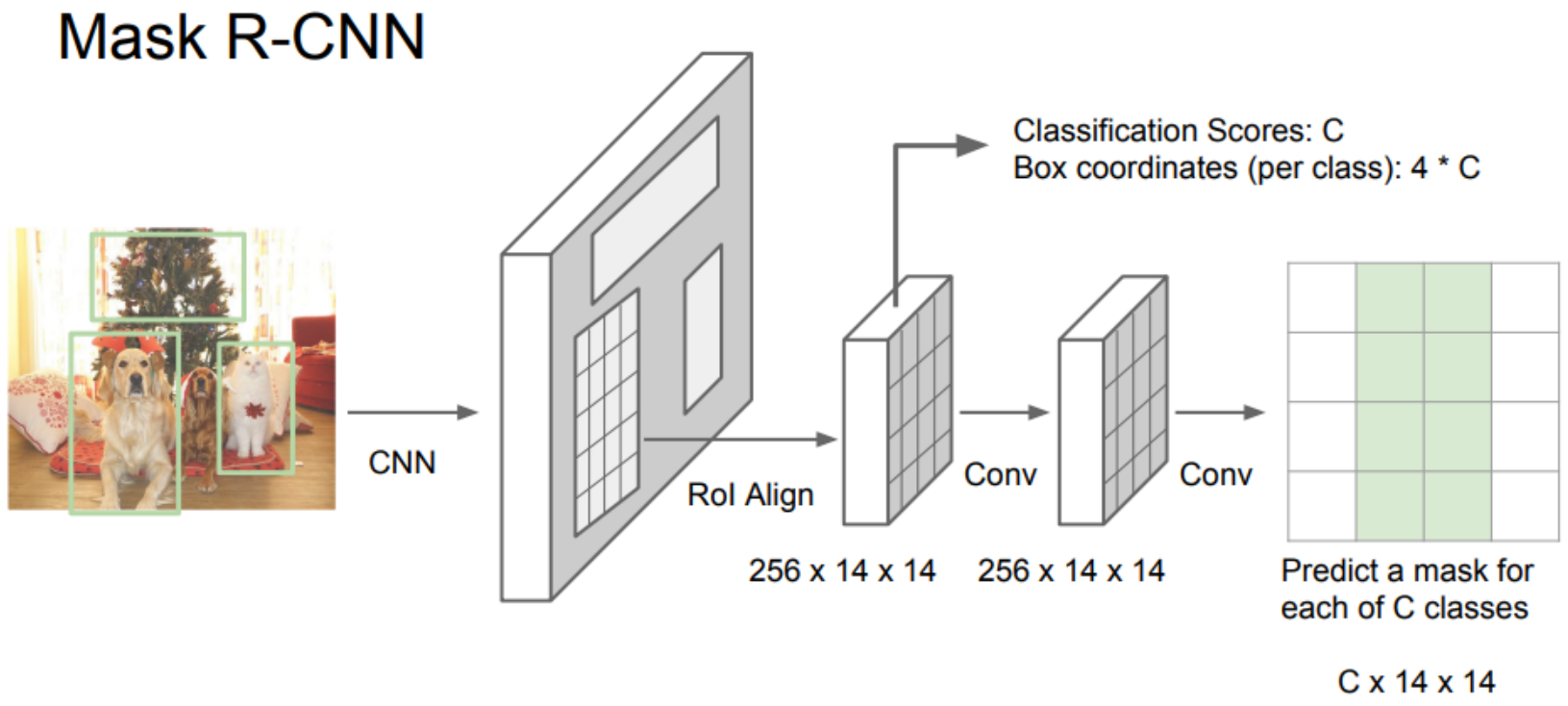




https://www.analyticsvidhya.com/blog/2021/01/auto-encoders-for-computer-vision-an-endless-world-of-possibilities/

Instance Segmentation (Mask R-CNN)

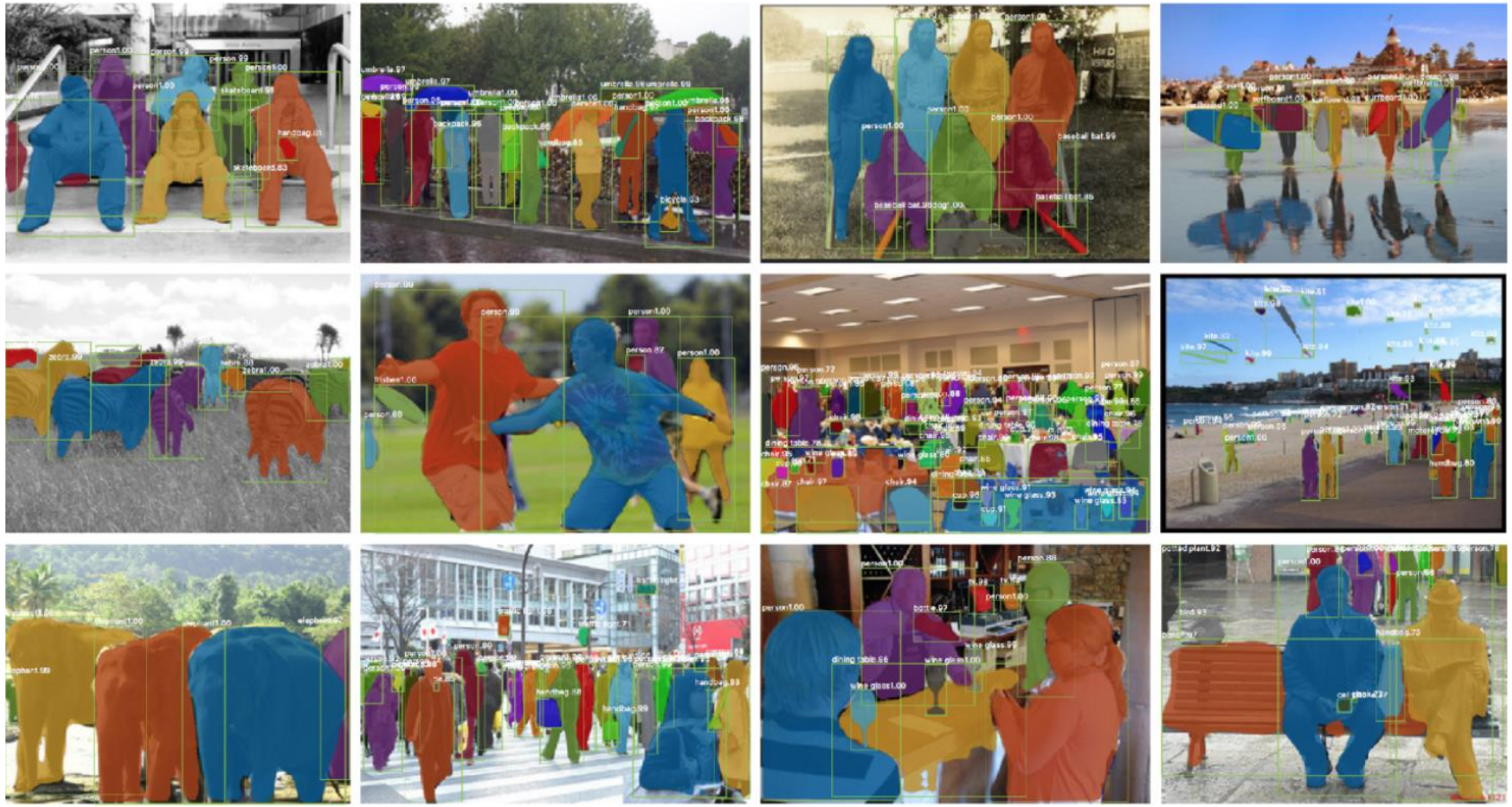
- Object Detection (Faster R-CNN) + Image Segmentation
- Use a pre-trained model which has ResNet-50-FPN backbone



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

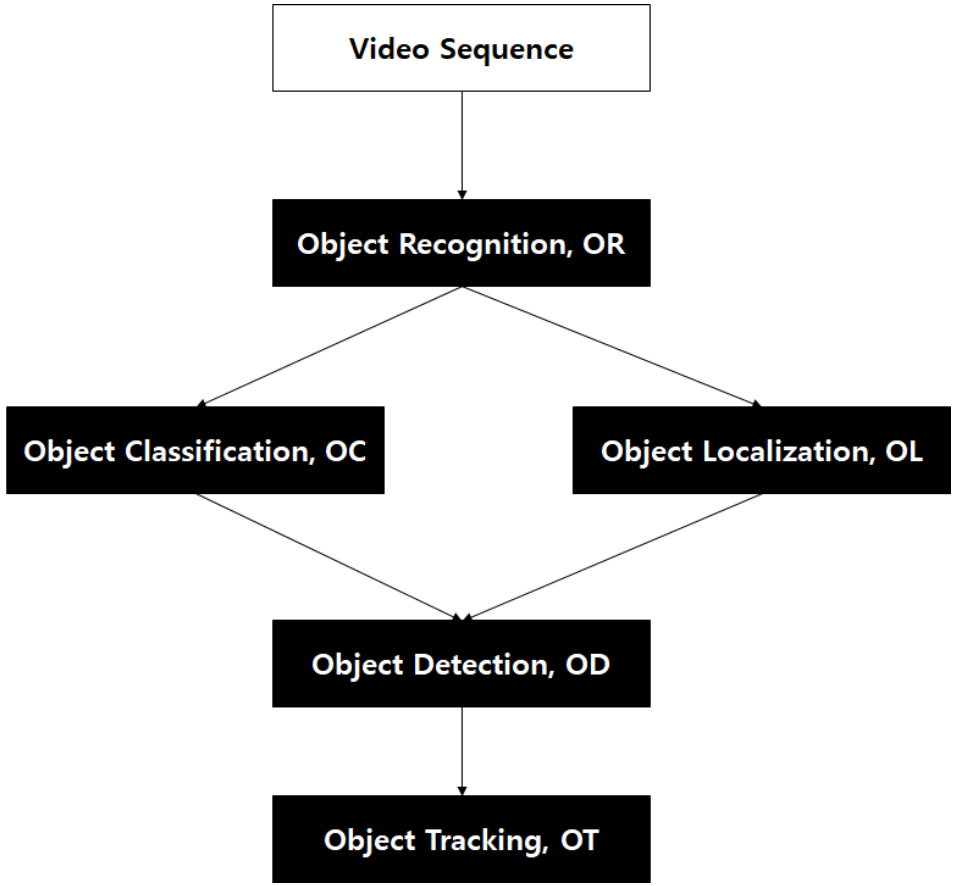
Mask R-CNN

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



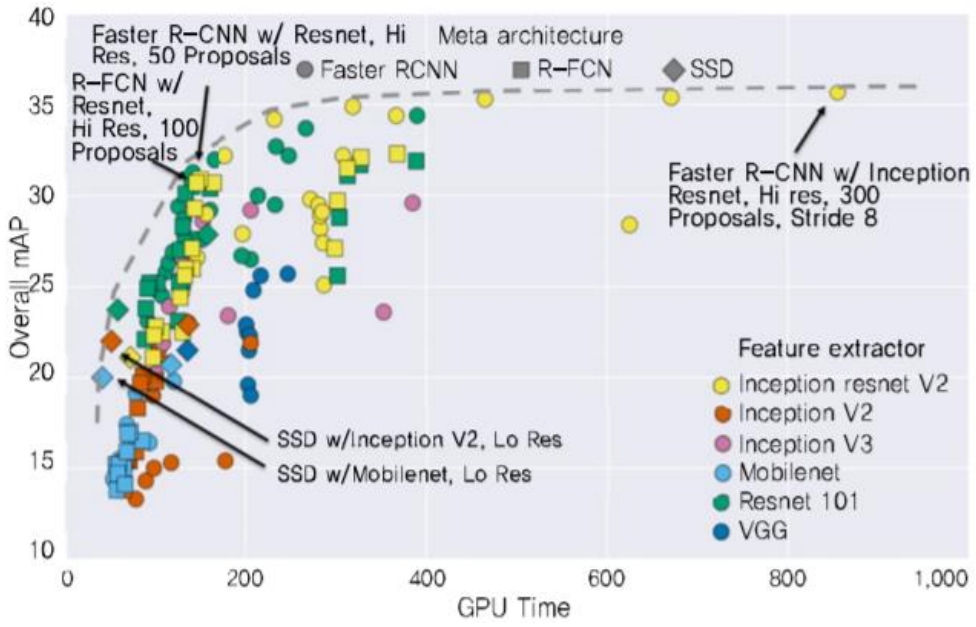
(Object) Recognition, Detection, Tracking

- 객체 인식, 탐지, 추적 개념

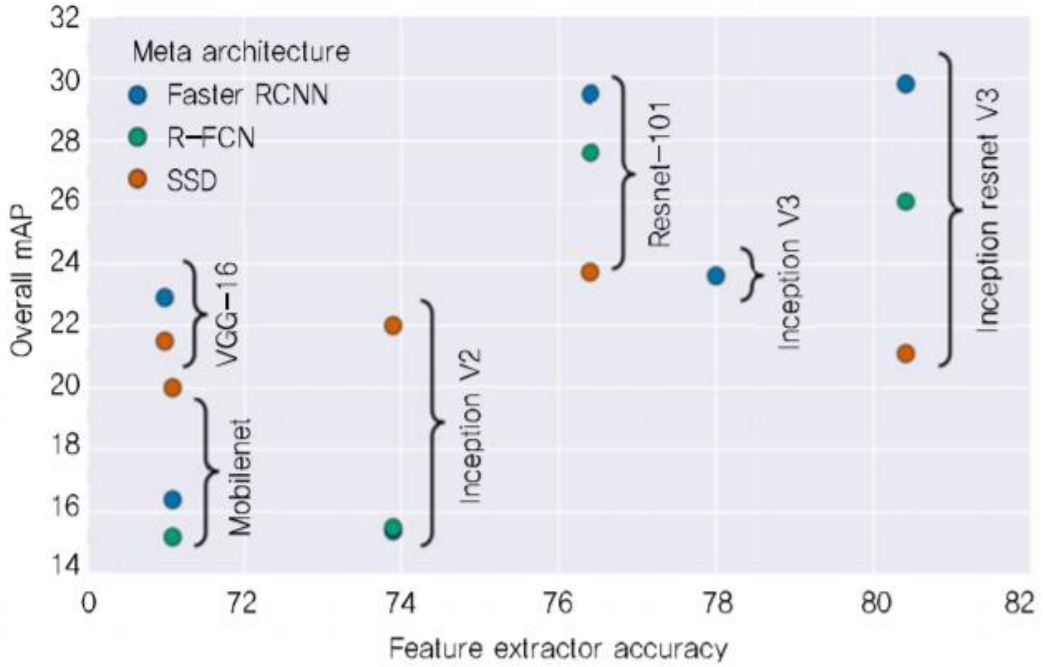


Object Recognition

- 객체 탐지 모델의 성능 비교,

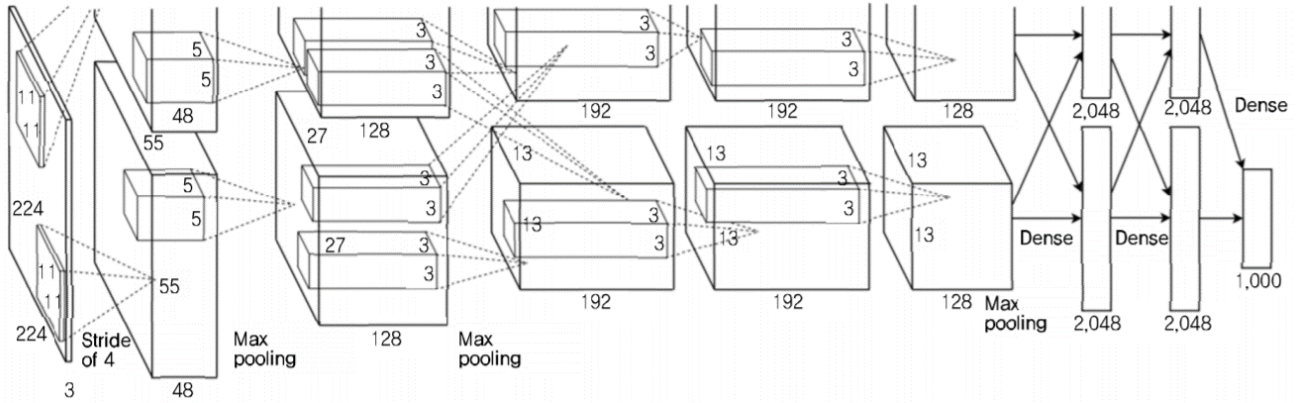


- CNN 모델 및 구조의 객체 인식 성능 비교

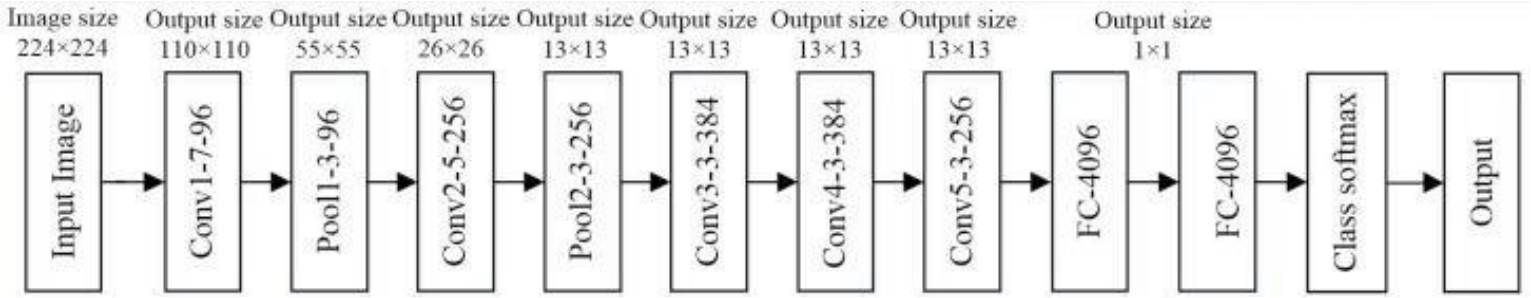


CNN Networks

- AlexNet: 8 개 층 CNN

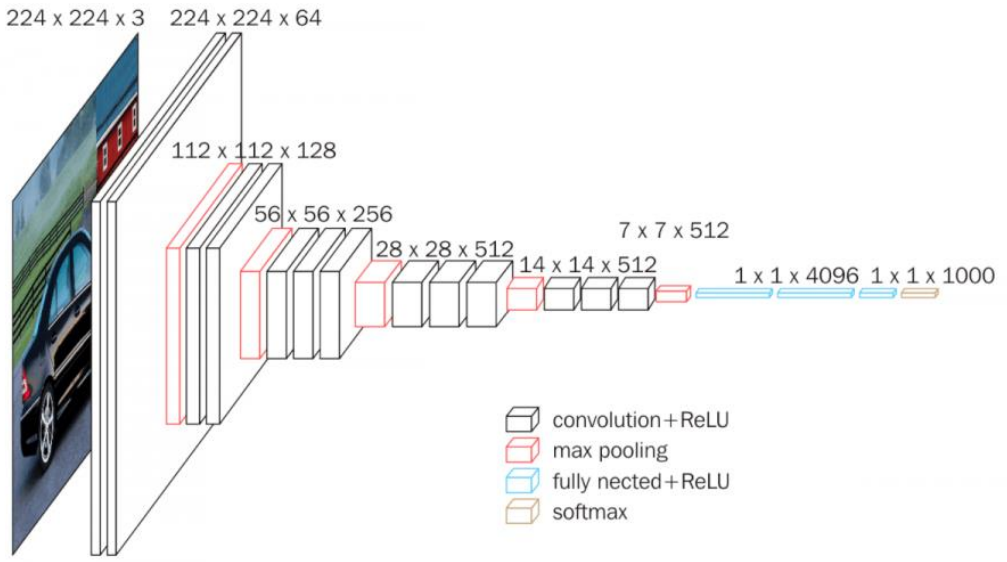


- ZFNet: AlexNet 의 구조 개선

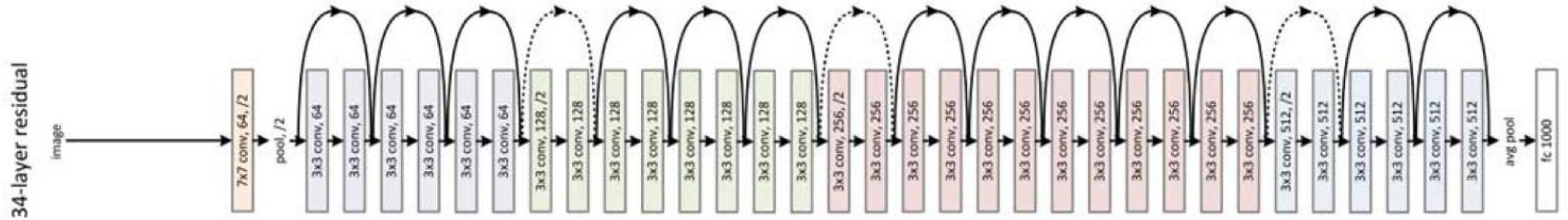


CNN Networks

- 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 × 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			

DarkNet-53 구조

Object Tracking

- **Object Tracking 기본과정**

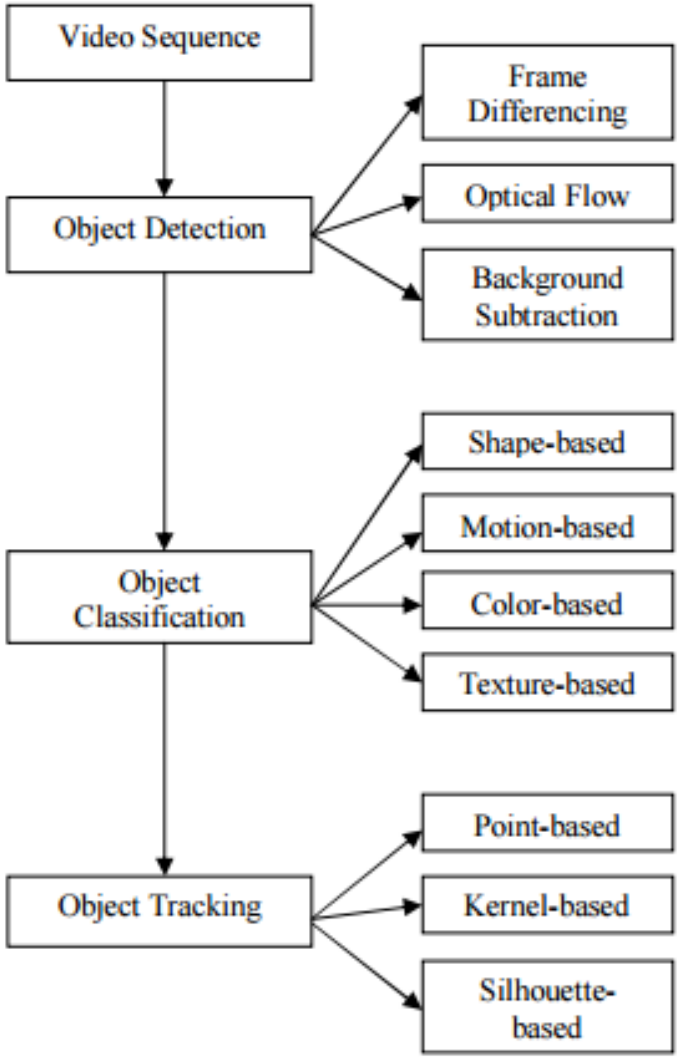
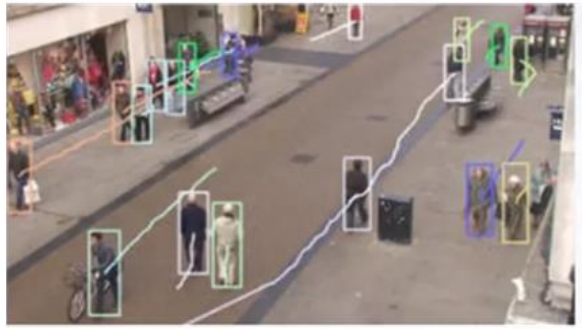


Fig 1: Basic steps for tracking an object [8]

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

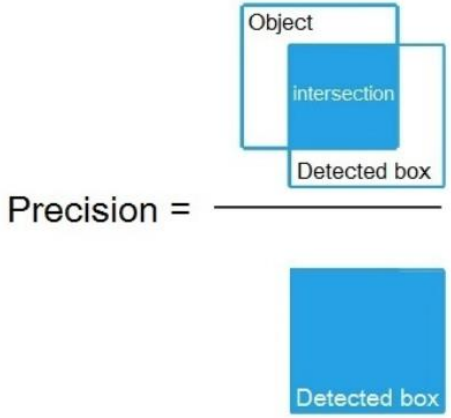
Object Detection Performance

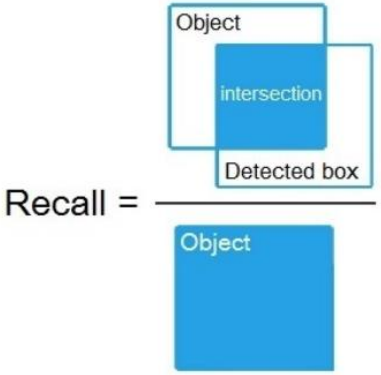
- 객체 검출에서는 단순히 어떤 객체가 있는지를 찾는 분류 문제가 아니므로 정확도 등 만으로 성능을 평가할 수 없고, 예측한 객체의 위치가 실제 객체의 위치와 얼마나 일치하는지를 평가해야 한다

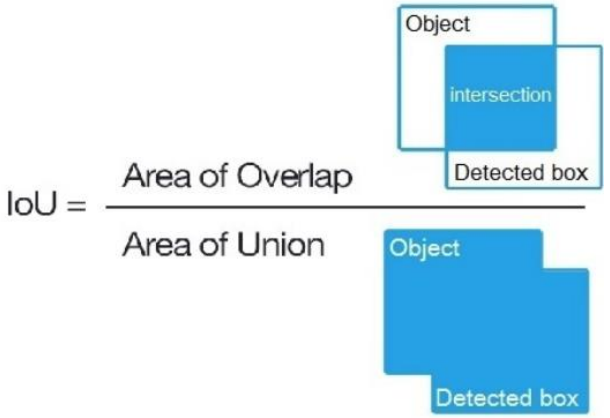
- 성능 평가척도
  - IoU (Intersection over Union)
  - mAP (mean Average Precision)
  - AR (Average Recall)

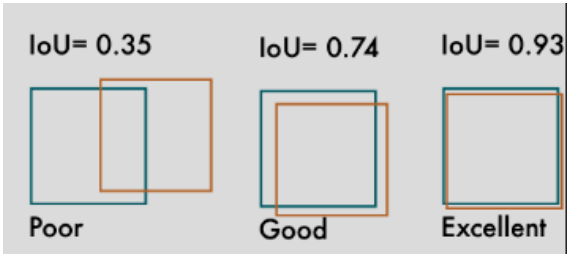
Object Detection Performance

- 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









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

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

Performance - IoU

- 실제 이미지 예

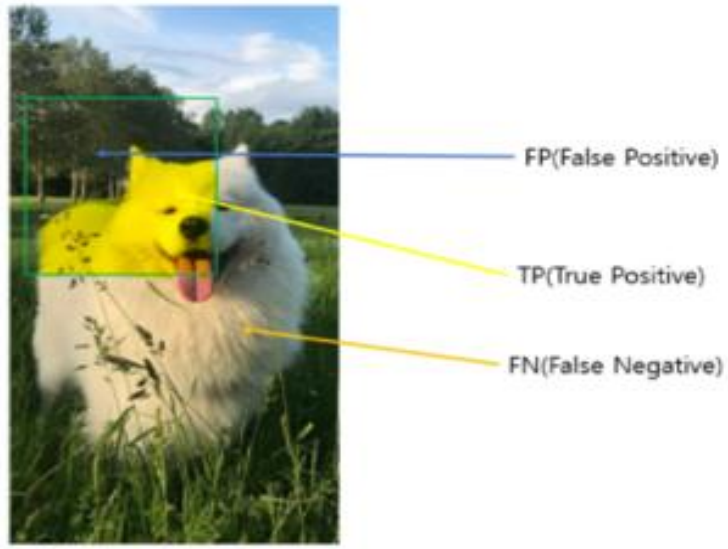
Performance – IoU and Dice coefficient

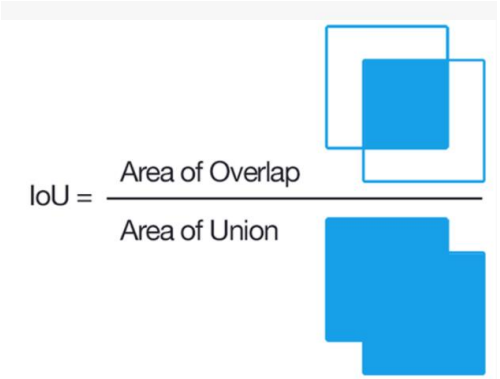
- Similarity between the prediction and the ground truth

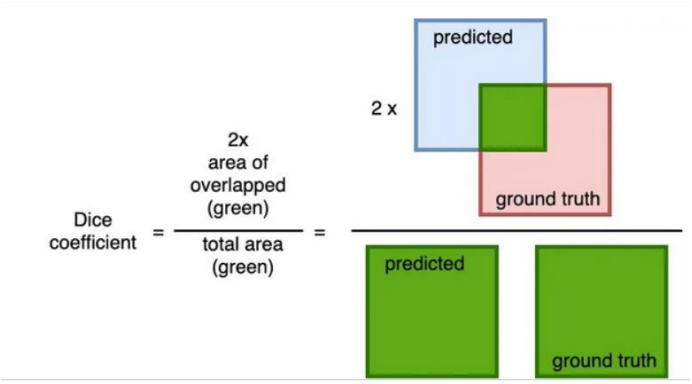
$$IoU = \frac{Inter\,sec\,tion}{Union} = \frac{TP}{FP + TP + FN} = \frac{\sum y\hat{y}}{\sum \hat{y} + y - y\hat{y}}$$

$$Dice\ coef\,f\,icient = \frac{2 \times Inter\,sec\,tion}{Union + Inter\,sec\,tion} = \frac{2TP}{2TP + FN + FP} = \frac{2\sum \hat{y}y}{\sum \hat{y} + y}$$

$$Dice = \frac{2 \times \text{Area of Intersection}}{\text{Area of Predicted} + \text{Area of Ground Truth}}$$







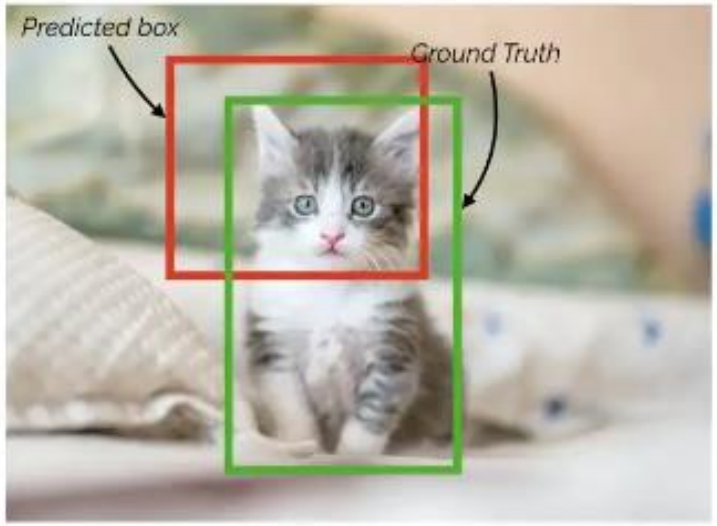
- Both IOU and Dice are in range (0,1)
- Dice is generally more sensitive to small differences between masks, making it suitable for tasks where precise segmentation is critical, such as medical imaging.

Object Detection Performance

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

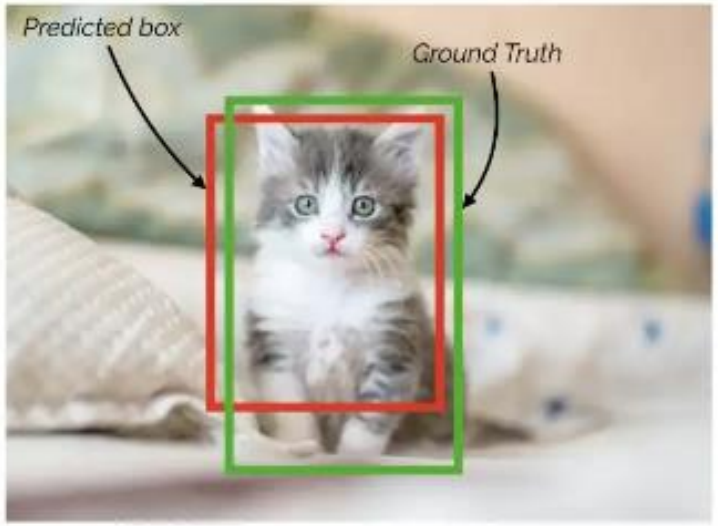
If IoU threshold = 0.5

False Positive (FP)



IoU = ~0.3

True Positive (TP)



IoU = ~0.7

Performance – mAP (mean Average Precision)

- 객체 탐지(Object Detection) 모델의 성능을 평가하는 대표적인 지표로, Precision-Recall 곡선의 면적(AP)을 클래스별로 계산한 뒤 평균을 낸 값.
- IoU (Intersection over Union): 예측 박스와 실제 박스가 얼마나 겹치는지 측정.
  - 보통 IoU ≥ 0.5 또는 0.75일 때 "탐지 성공"으로 간주
- AP (Average Precision)
  - Precision과 Recall은 trade-off 관계가 있다.
  - 다양한 threshold에서 Precision-Recall 곡선을 그린 뒤, 그 아래 면적을 계산한 값이 AP 이다.
  - 즉, AP는 한 클래스에 대한 탐지 성능 요약값.
- mAP (mean Average Precision)
  - 여러 클래스가 있을 때, 각 클래스별 AP를 구한 뒤 평균을 내는 게 mAP 임.
  - 예: COCO 데이터셋에서는 IoU 기준을 0.5~0.95까지 0.05 간격으로 계산해 평균
    - mAP@0.5: IoU ≥ 0.5일 때 평균 AP
    - mAP@0.5:0.95: IoU 0.5~0.95까지 평균 AP (더 엄격한 기준)

Performance – mAP (예제)