

A solid blue vertical bar is positioned on the left side of the slide, extending from the top to the bottom.

Embedded Object Detection With Jetson TX2

1. Embedded Deep Learning Inference
 2. Object Detection with Deep Learning
 3. YOLO V1
 4. YOLO V2
 5. YOLO with Jetson TX2
 6. Demo
-



Embedded Deep Learning Inference

Embedded Deep Learning Workflow

Training



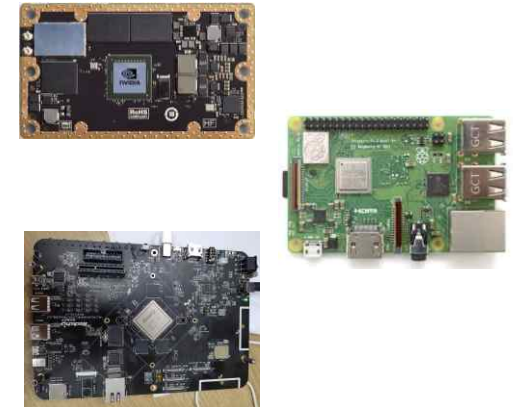
- 모델 설계/학습
- Tensorflow
- Caffe
- Darknet

Conversion



- 모델 변환
- TensorRT
- ARM NN SDK
- RKNN Toolkit

Deploy

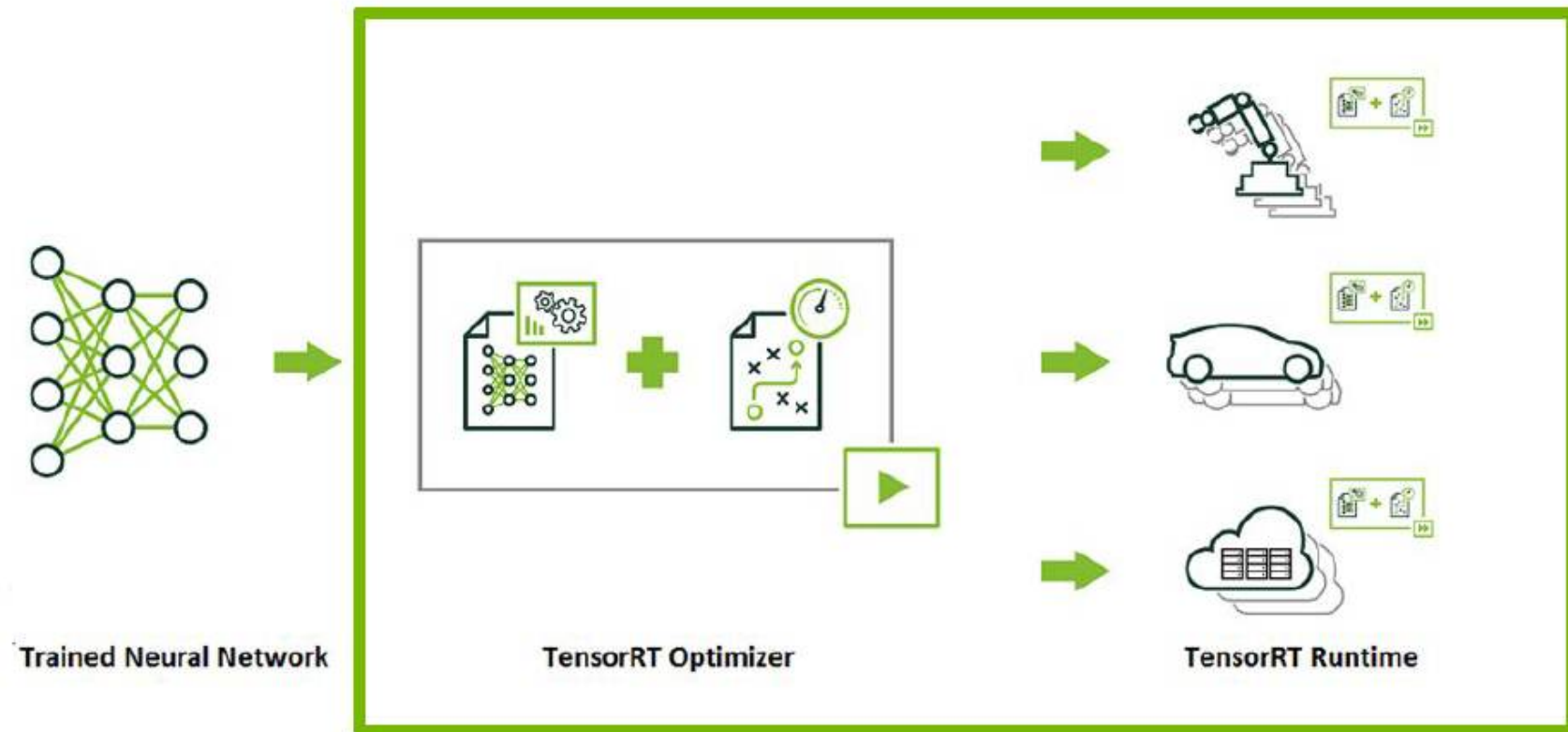


- 모델 실행
- TensorRT
- ARM NN API
- RKNN API
- Darknet

Embedded Deep Learning Frameworks

► NVIDIA TensorRT

- Inference engine for production deployment of deep learning applications

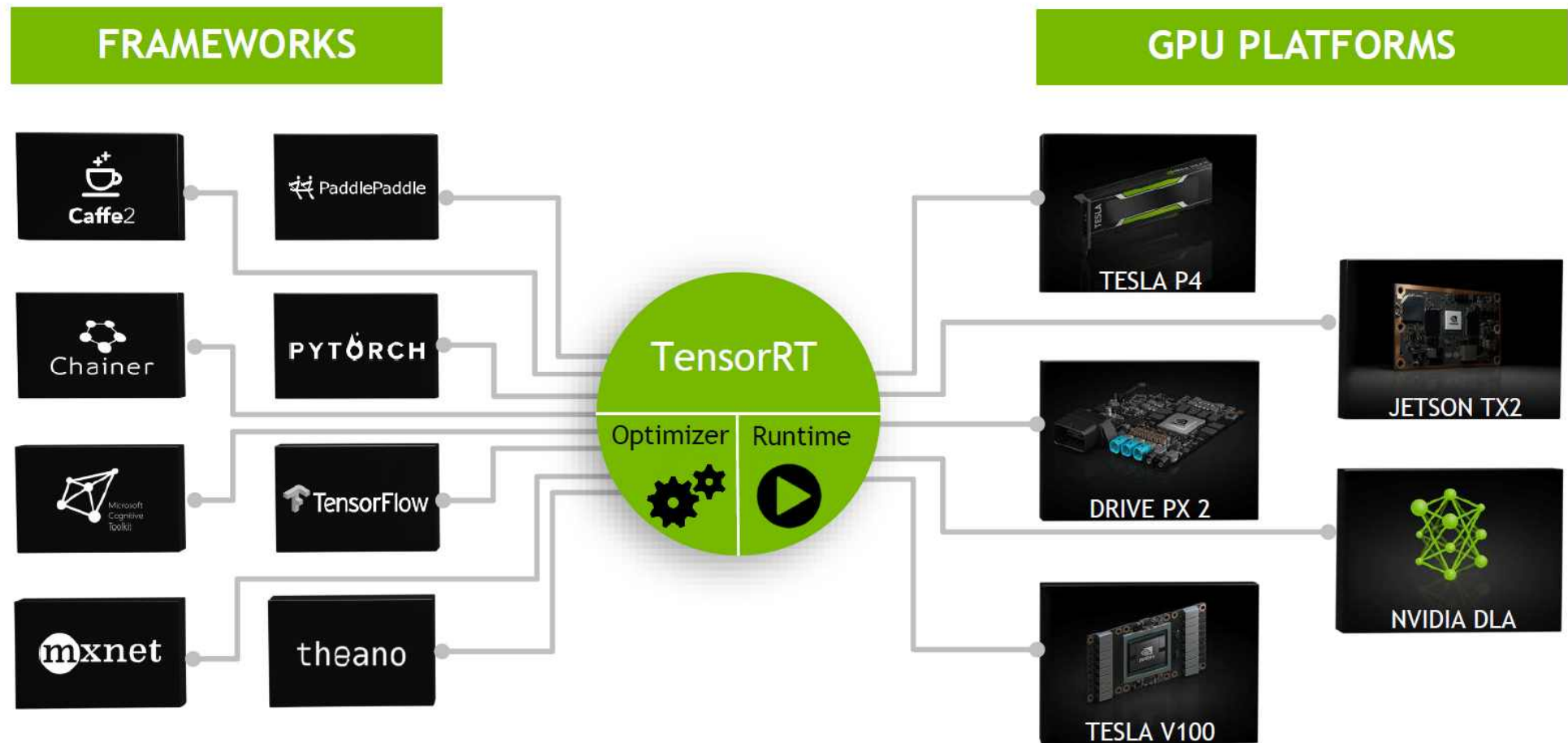


- Allows Developers to focus on developing AI powered applications
: TensorRT ensures optimal inference performance

Embedded Deep Learning Frameworks

► NVIDIA TensorRT

-. Programmable Inference Accelerator



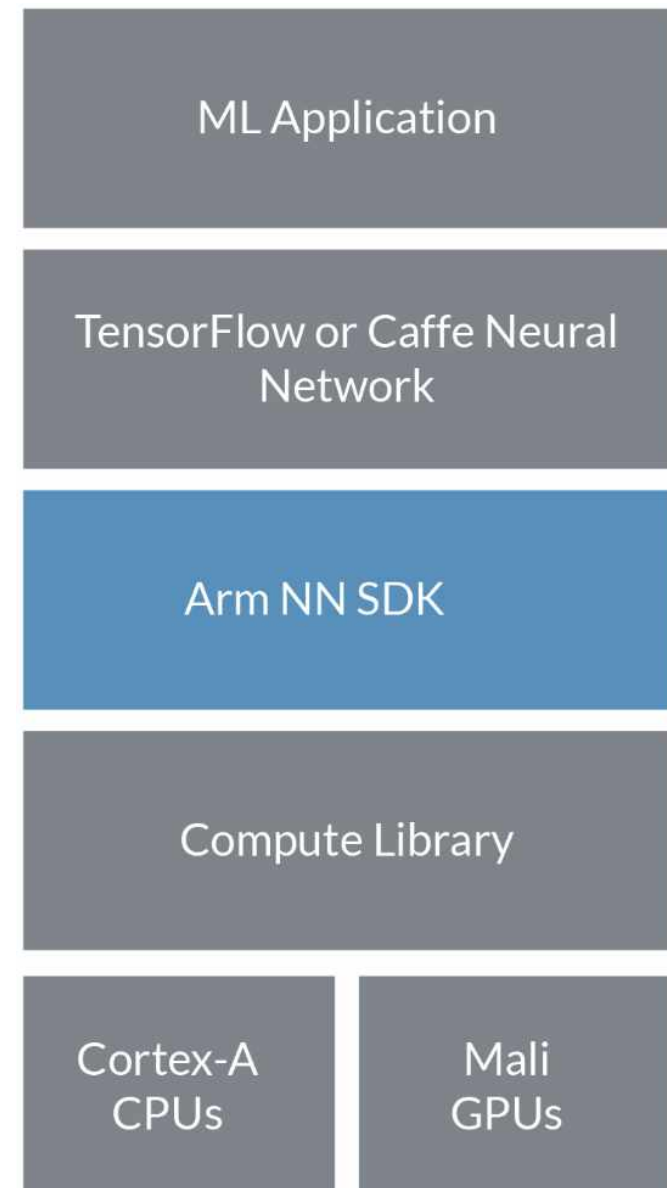
Embedded Deep Learning Frameworks

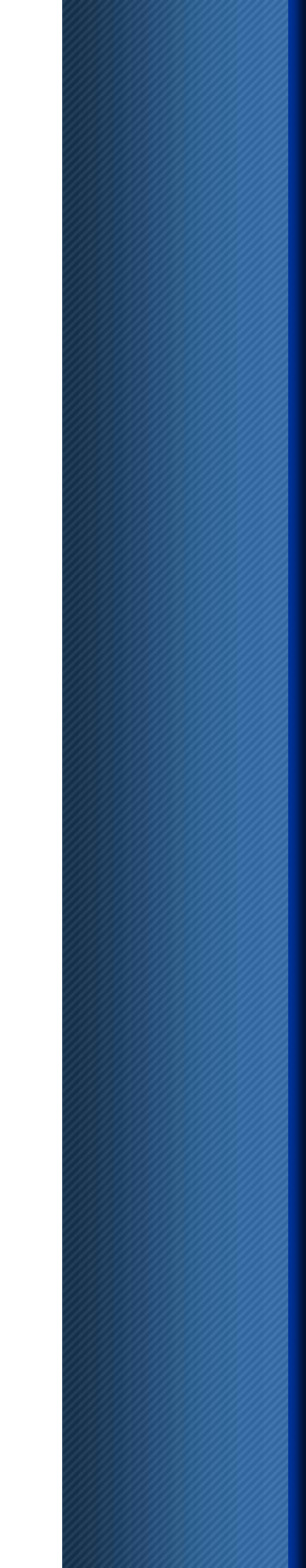
► ARM NN SDK

Arm NN is an inference engine for CPUs, GPUs and NPU.

Arm NN SDK is a set of open-source Linux software and tools that enables machine learning workloads on power-efficient devices. It provides a bridge between existing neural network frameworks and power-efficient [Arm Cortex CPUs](#), [Arm Mali GPUs](#) or the [Arm Machine Learning processor](#).

Arm NN SDK utilizes the Compute Library to target programmable cores, such as Cortex-A CPUs and Mali GPUs, as efficiently as possible. It includes support for the Arm Machine Learning processor and, via CMSIS-NN, support for Cortex-M CPUs. .



A solid blue vertical bar is located on the left side of the slide, extending from the top to the bottom.

Object Detection With Deep Learning

Object Detection With Deep Learning

► Generic Object Detection with DL

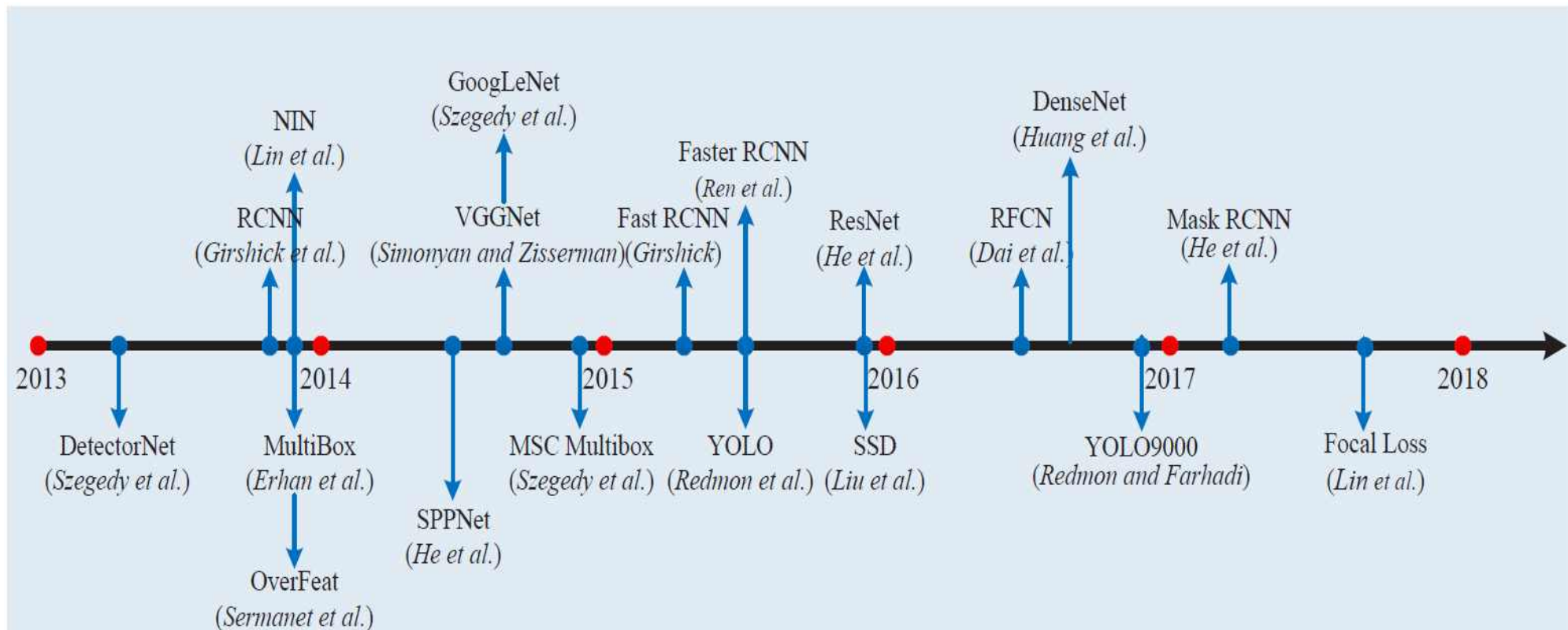


Fig. 6 Milestones in generic object detection

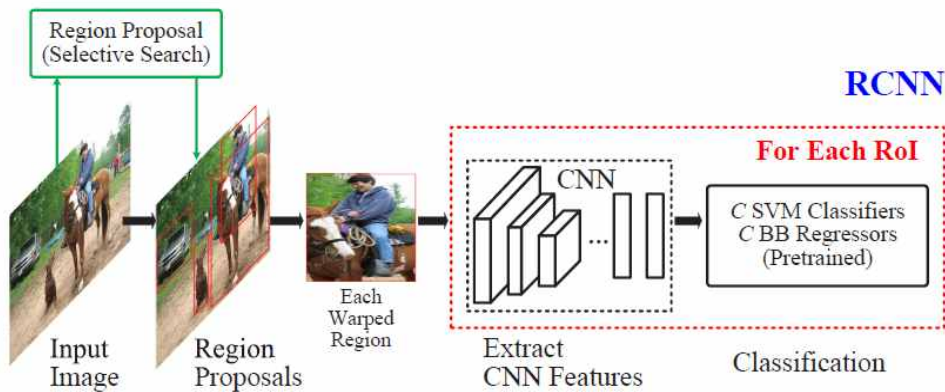
Deep Learning for Generic Object Detection: A Survey

<https://arxiv.org/abs/1809.02165>

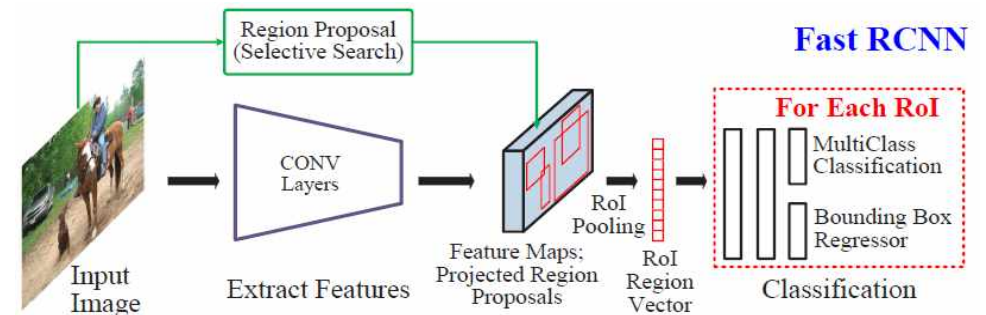
Object Detection With Deep Learning

► Region Based(Two Stage Framework)

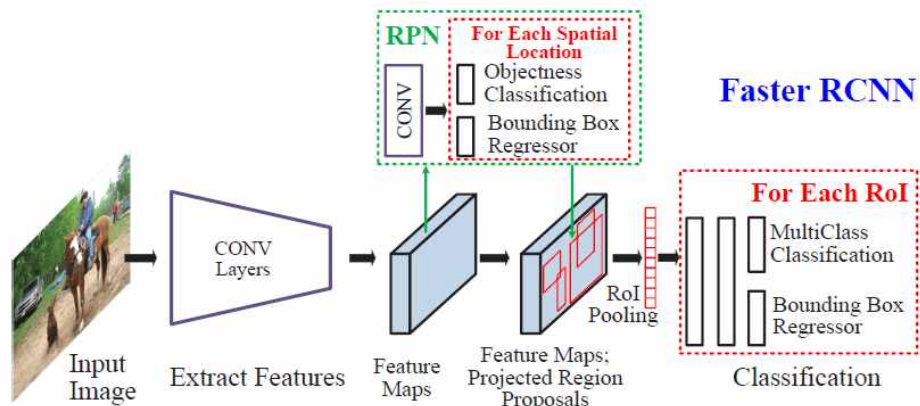
< RCNN >



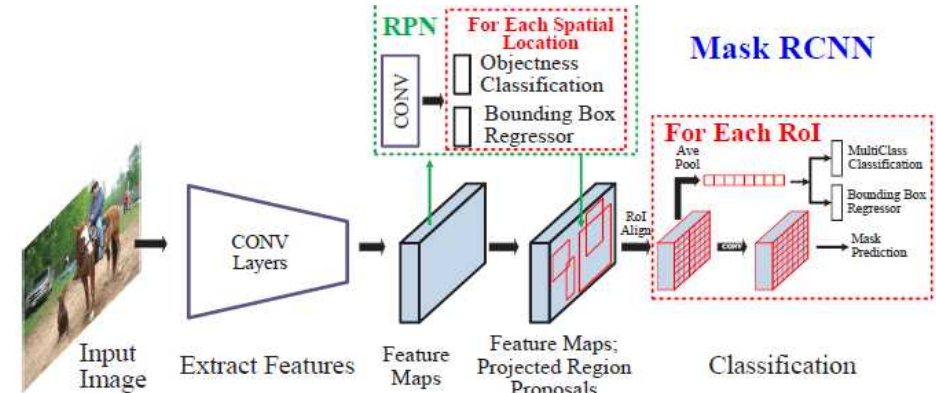
< Fast RCNN >



< Faster RCNN >

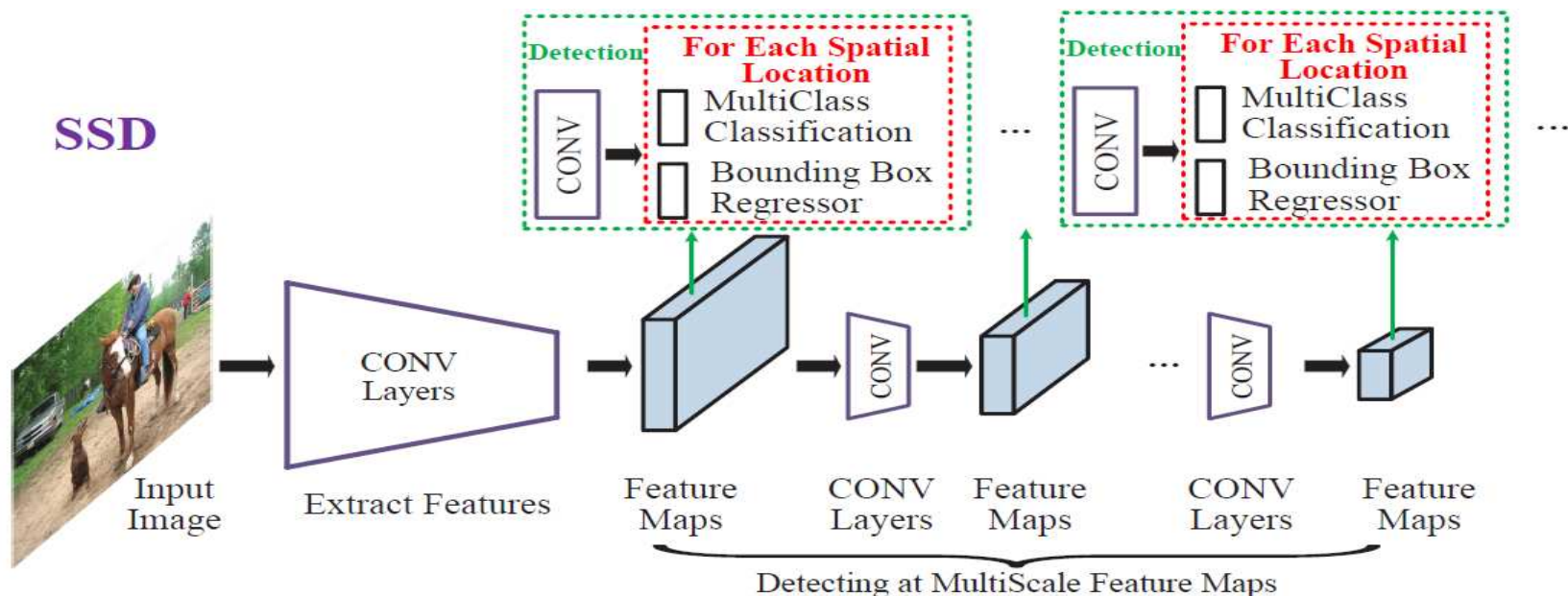
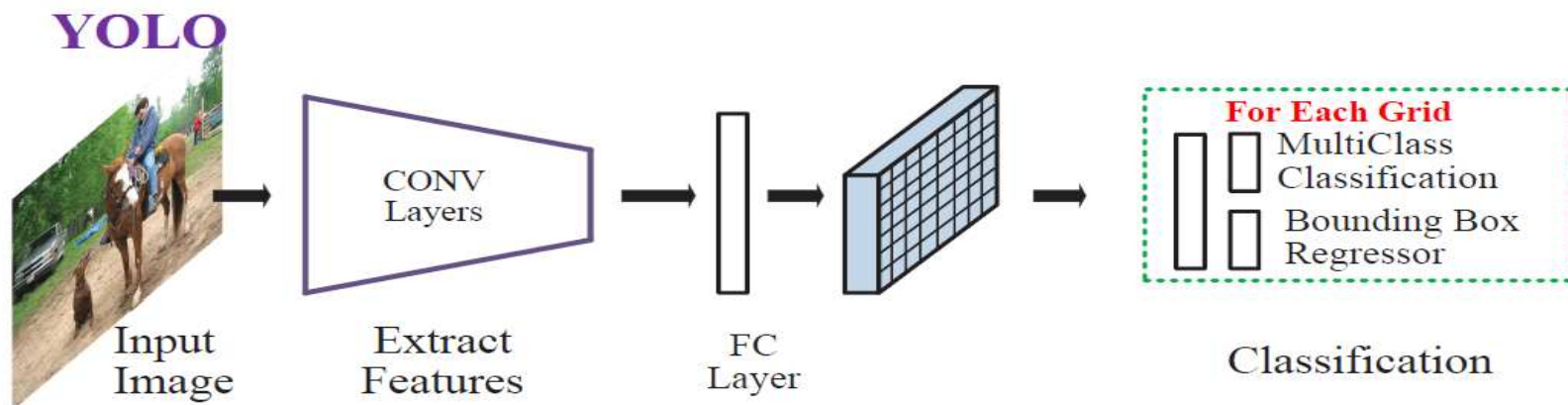


< Mask RCNN >



Object Detection With Deep Learning

► Unified Pipeline(One Stage Pipeline)



YOLO V1

Written by 박 철(e2g1234@naver.com)

YOLO(You Only Look Once)

You Only Look Once: Unified, Real-Time Object Detection

Joseph Redmon*, Santosh Divvala*[†], Ross Girshick[‡], Ali Farhadi*[†]

University of Washington*, Allen Institute for AI[†], Facebook AI Research[‡]

<http://pjreddie.com/yolo/>

Abstract

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

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

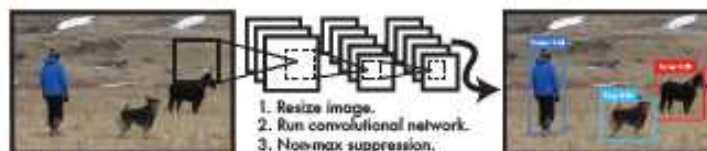


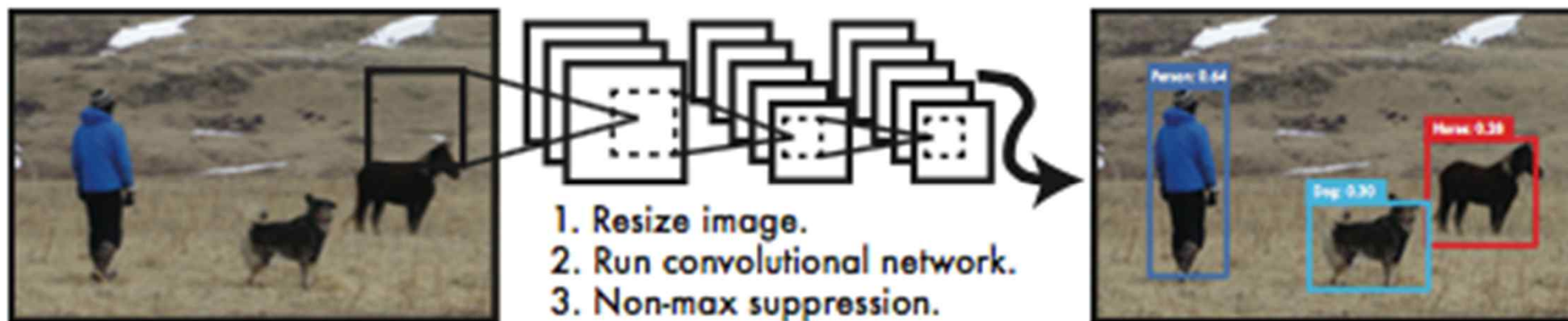
Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448×448 , (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

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

We reframe object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. Using our system, you only

- ✓ object detection을 공간적으로 분리된 bounding box들과 class(분류할 object 종류들)에 대한 확률과 연관된 regression 문제로 재정의
- ✓ Single neural network는 전체 이미지에서 한번의 평가를 통해서 직접적으로 bounding box들을 예측하고 class의 확률을 계산
- ✓ 전체 detection pipeline이 Single network이기 때문에, 탐지 성능이 최적화 됨
 - 통합된 아키텍처를 사용하므로 빠름
 - YOLO model은 45 FPS로 실시간 이미지 연산이 가능
 - 더 작은 버전의 네트워크인 Fast YOLO는 매우 놀랍게도 다른 실시간 탐지모델보다 2배의 mAP(Mean Accuracy Precision:: 평균 정확도)를 가지면서 155 FPS의 성능을 보여줌
 - 최신의 탐지 시스템들과 비교했을 때, YOLO는 localization error가 조금 더 높지만, 배경으로 인한 예측 실패율은 더 낮음
 - 마지막으로 YOLO는 object의 아주 일반적인 표현들을 학습합니다. 이 말은 DPM이나 R-CNN같은 다른 탐지 대비 결과가 더 좋다는 것을 의미함

The YOLO Detection System



YOLO를 이용하여 이미지를 처리하는 과정은 간단하고 직관적임.

(1) input image를 448 x 448의 이미지로 resize

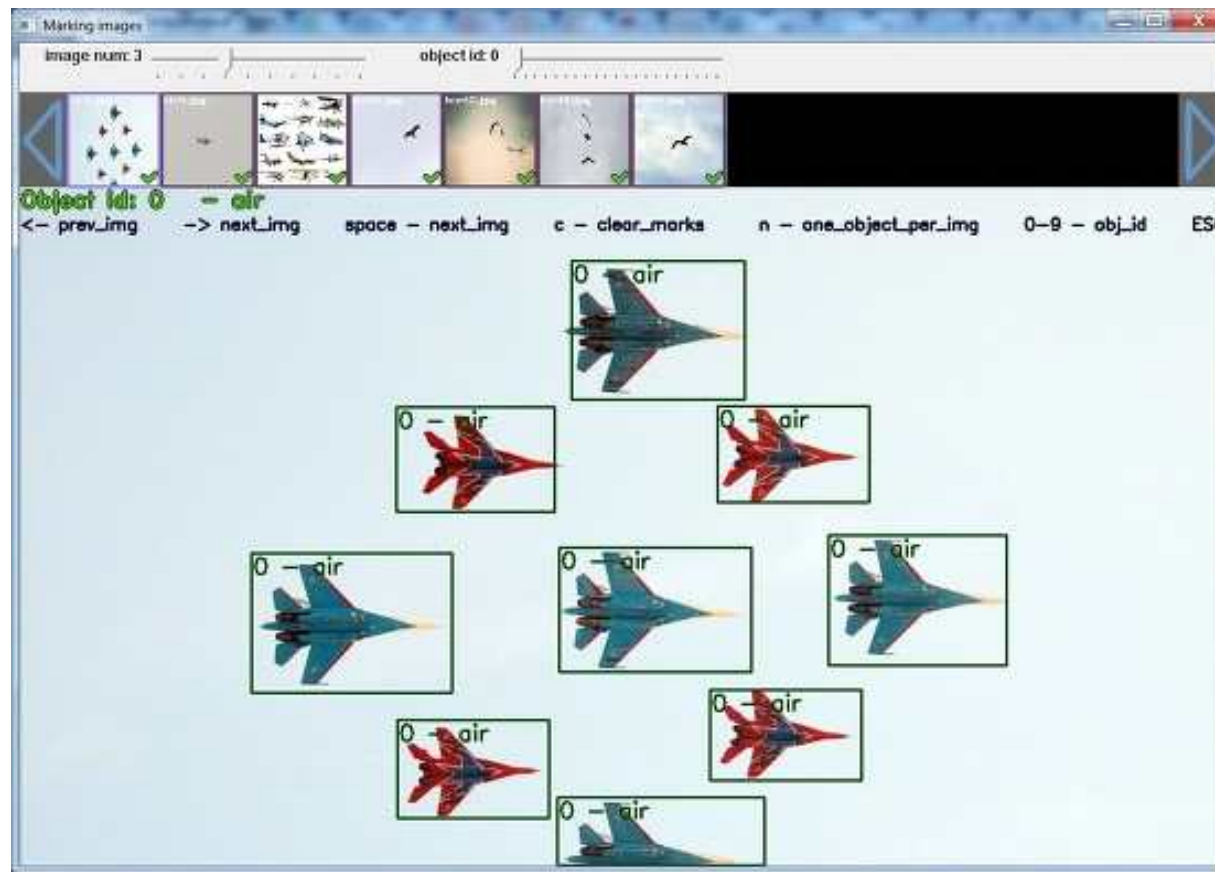
(2) image에서 작동하는 single convolution network를 실행

(3) 해당 모델을 통해서 나온 확률 값을 threshold로 잘라서 결과값을 보여줌

데이터셋 작업과 Ground truth

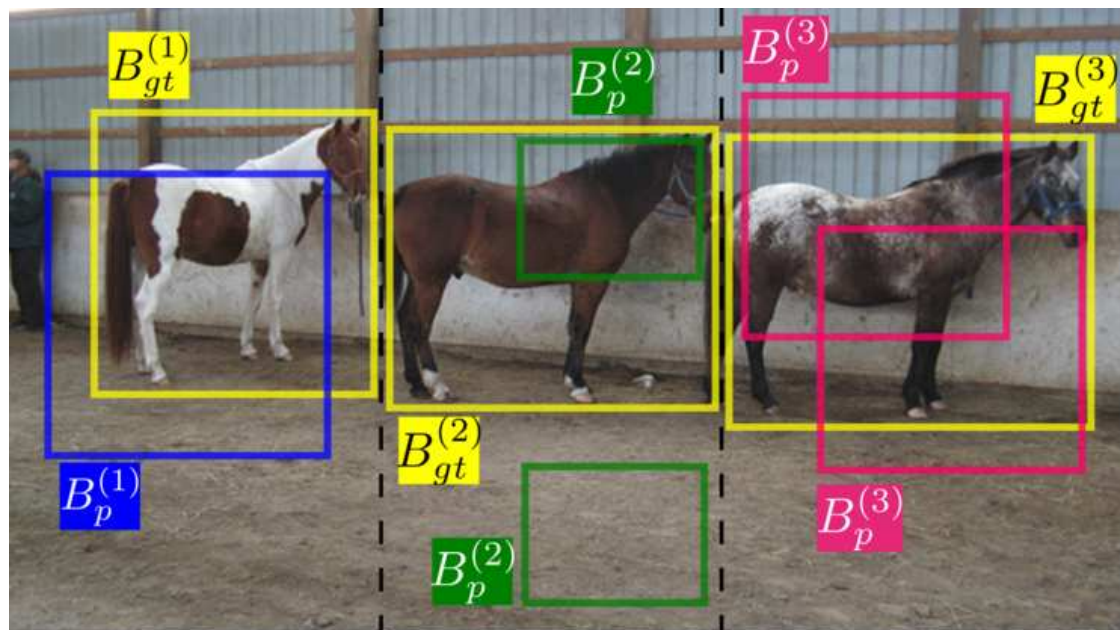
✓ ground truth

- 사물의 실제 위치를 나타내는 ‘실제(ground truth; 이하 GT)’ 바운딩 박스 정보가 이미지 레이블 상에 포함되어 있습니다.
- ✓ 학습하는 과정에서 실제 데이터 라벨(또는 클래스) - 이를 Ground Truth Label이라고 합니다



IOU(intersection over union)

- Detection 문제의 경우, 사물의 클래스 및 위치에 대한 예측 결과를 동시에 평가
- 각 예측 바운딩 박스 와 실제 바운딩 박스가 얼마나 '겹쳐지는지' 를 평가



(1) 매칭 성사

(2) 매칭 실패 -
IOU 문턱값 미만

(3) 매칭 실패 -
한 B_{gt} 에 복수 개 B_p 매칭

confidence score

해당 모델이 해당 box안에 object가 있을 확률이 얼마나 되는지,
그리고 해당 object가 자신이 예측한 object가 맞을 확률이 얼마나 되는지에 대한 확률에 대한 score

$$Pr(Class_i | Object) * Pr(object) * IOU_{pred}^{truth} = Pr(Class_i) * IOU_{pred}^{truth} \quad (1)$$

$Pr(Class_i | Object)$

각각의 cell은 C에 대해서 예측

C는 class의 조건부 확률

C는 객체를 포함하는 grid cell에 따라 달라짐.

하나의 grid cell은 예측된 상자수 B와는 상관없이 한가지 종류의 Class의 확률만 계산

테스트 시에는 class의 조건부 확률과 각각의 개별적인 box에 대해서 confidence 예측

$Pr(object)$

$Pr(Class_i)$

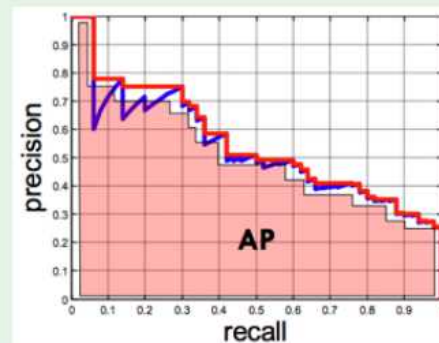
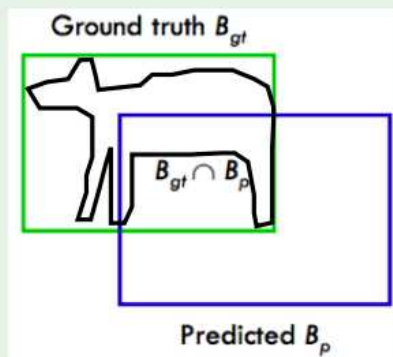
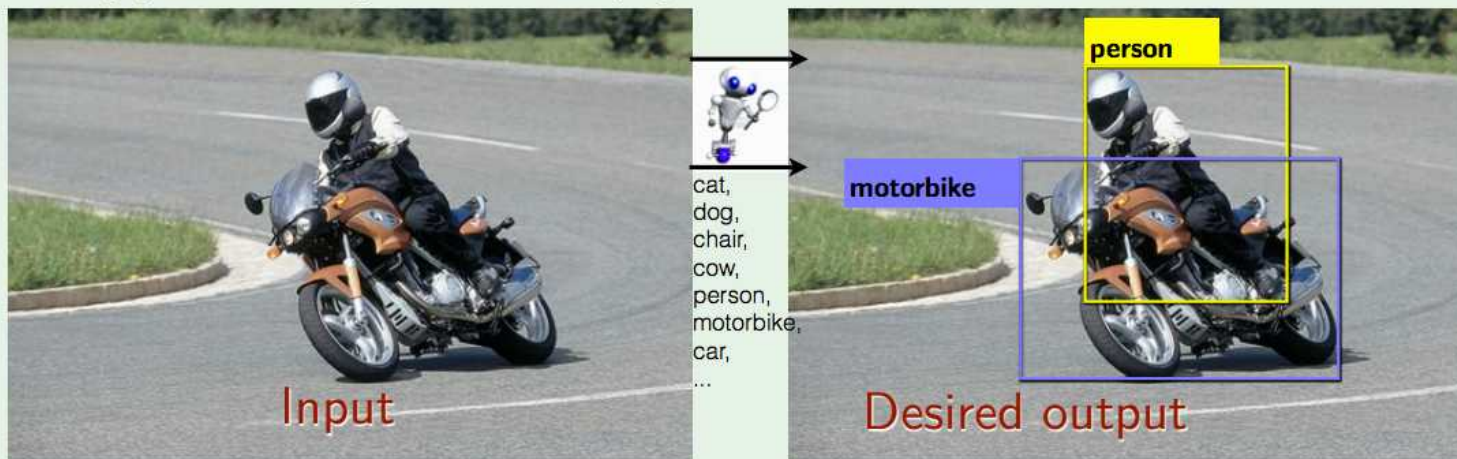
$P(A \cap B) = P(A, B)$: 교집합
 $P(A | B) = P(A, B) / P(B)$: 조건부 확률
 $P(A, B) = P(A | B) \times P(B)$: 교집합

$P(Class \cap Object) = P(A, Object)$: 교집합
 $P(Class | Object) = P(Class, Object) / P(Object)$: 조건부 확률
 $P(Class, Object) = P(Class | Object) \times P(Object)$: 교집합

mAP(mean Average Precision)

Formalizing the object detection task

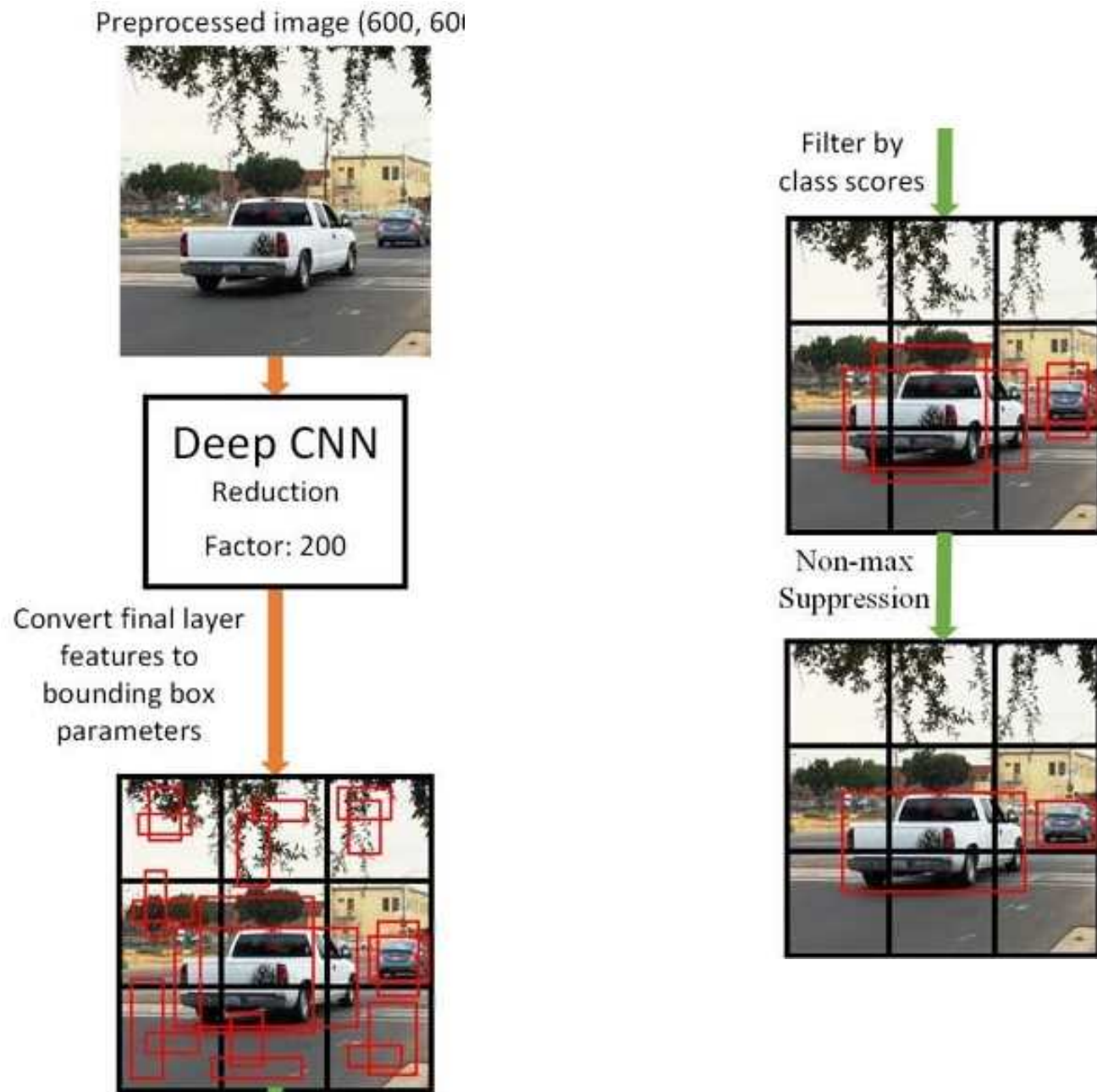
Many possible ways, this one is popular:



Performance summary:

Average Precision (AP)
0 is worst 1 is perfect

YOLO 처리과정의 개요



How unified detection works?

regression problem으로 object를 detection
image를 $S \times S$ 의 grid cell로 나눔
각각의 cell은 B개의 bounding box와 confidence score 예측
예측값은 $S \times S \times (B * 5 + C)$ 크기의 tensor로 구성됨

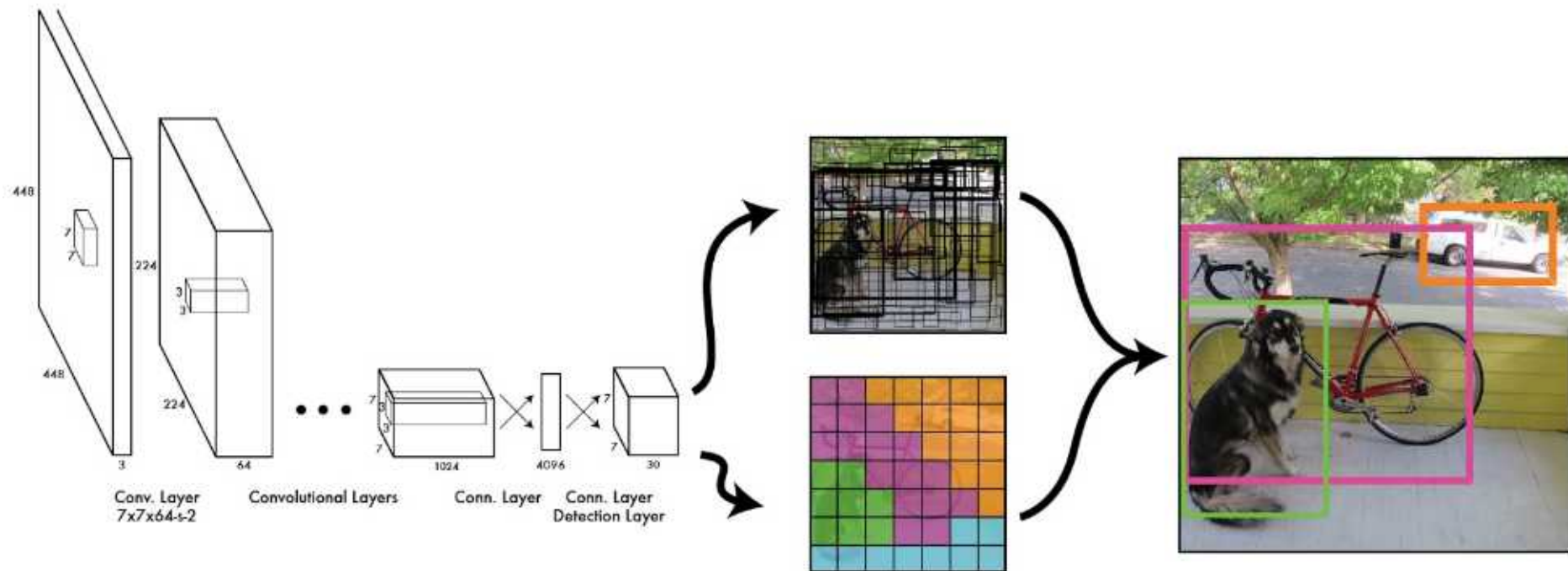
Pascal VOC 데이터를 이용해 평가

$S=7$, $B=2$ 사용

Pascal VOC data는 20개의 class로 label 됨

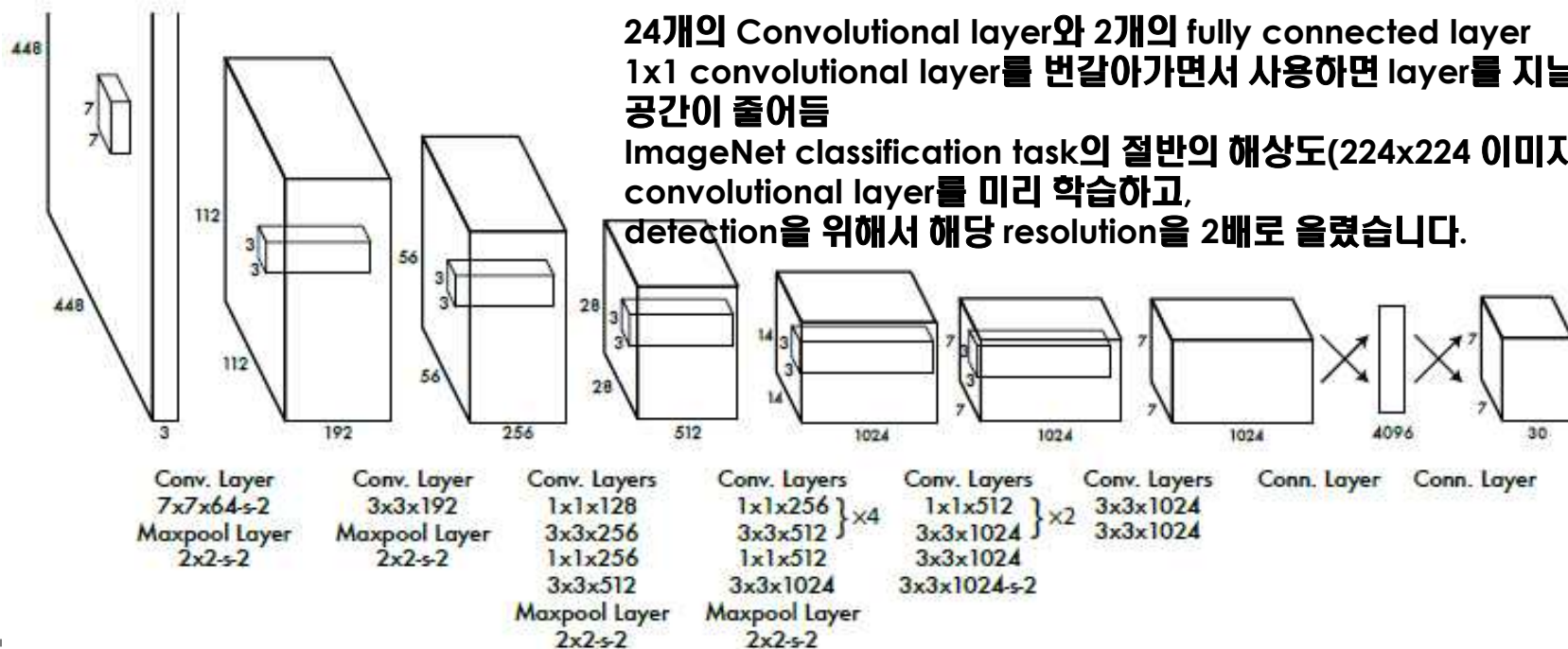
$C=20$

최종 예측 값 $7 \times 7 \times 30$ tensor가 됨



Network Design

- ✓ Convolutional neural network로 구현
- ✓ Pascal VOC detection dataset으로 평가
- ✓ 네트워크의 첫번째 Convolutional layer는 image로부터 특징을 추출
- ✓ fully connected layer는 출력확률과 좌표를 예측
- ✓ network 아키텍처는 이미지 분류를 위한 GoogLeNet 모델로부터 영감 받음
 - network는 24개의 Convolutional layer와 2개의 fully connected layer로 구성
 - GoogLeNet이 사용한 inception 모듈 대신 간단하게 1x1 reduction layer와 3x3 convolutional layer 사용



Network Design

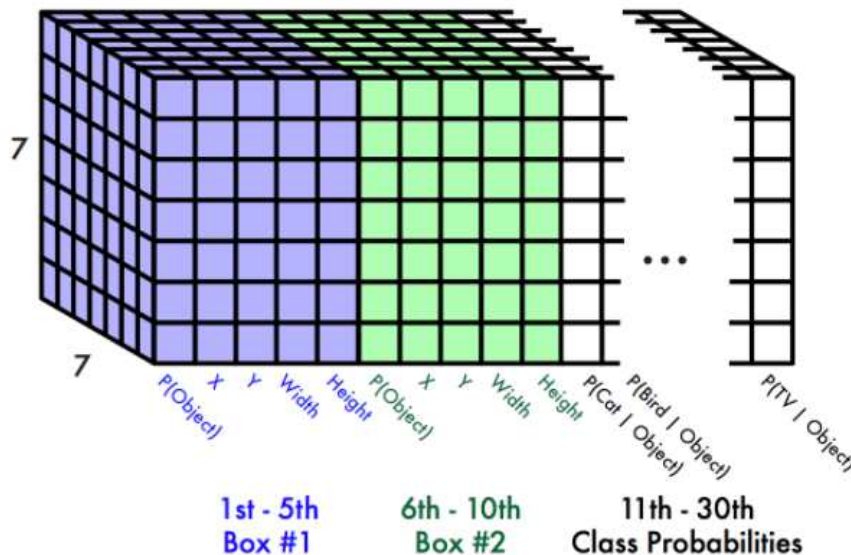
- ✓ 빠른 object detection의 한계를 넘게 설계된 YOLO의 빠른 버전을 학습
- ✓ Fast YOLO는 기존에 24개의 Convolution Layer를 사용하던 YOLO에 비교해서 9개로 더 적은 Convolution Layer를 사용
- ✓ 나머지 network의 크기나 학습, 테스트 파라미터들은 기존 YOLO와 Fast YOLO는 같음.
- ✓ network의 최종 output은 7x7x30 tensor로 출력

Each cell predicts:

- For each bounding box:
 - 4 coordinates (x, y, w, h)
 - 1 confidence value
- Some number of class probabilities

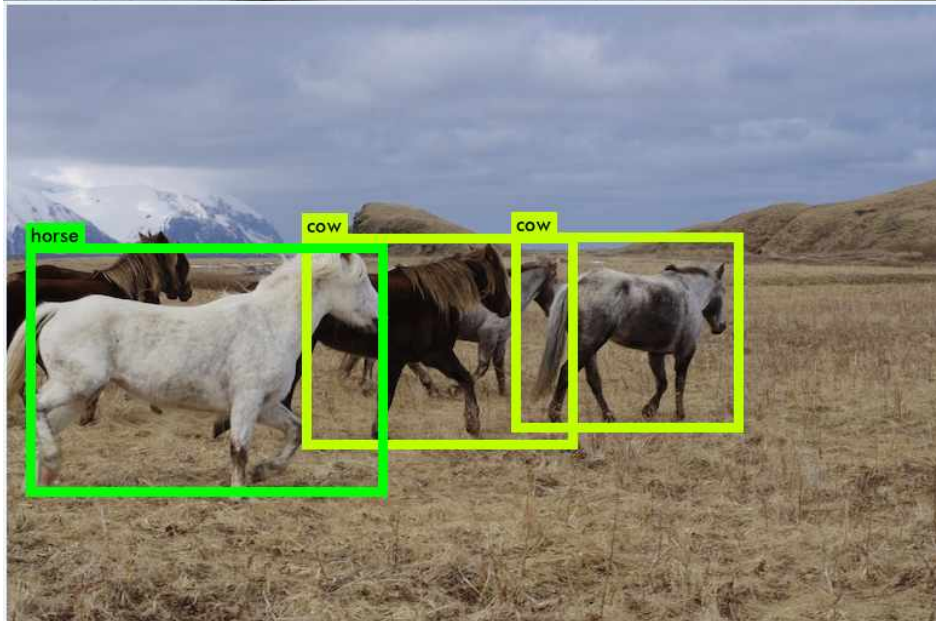
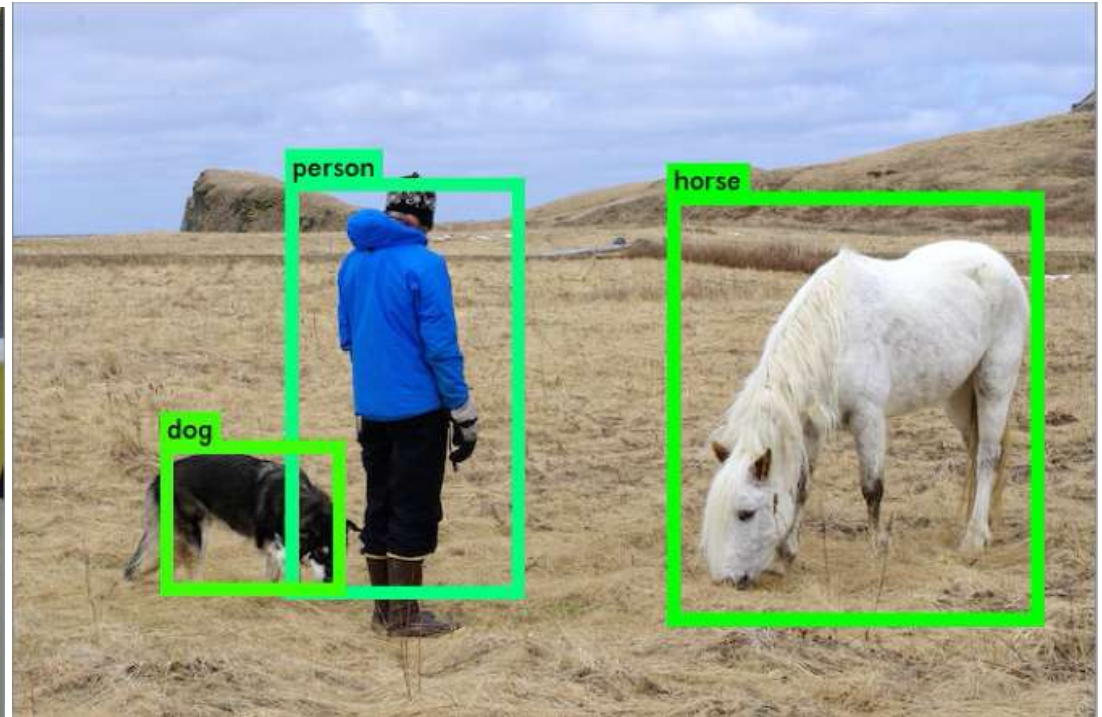
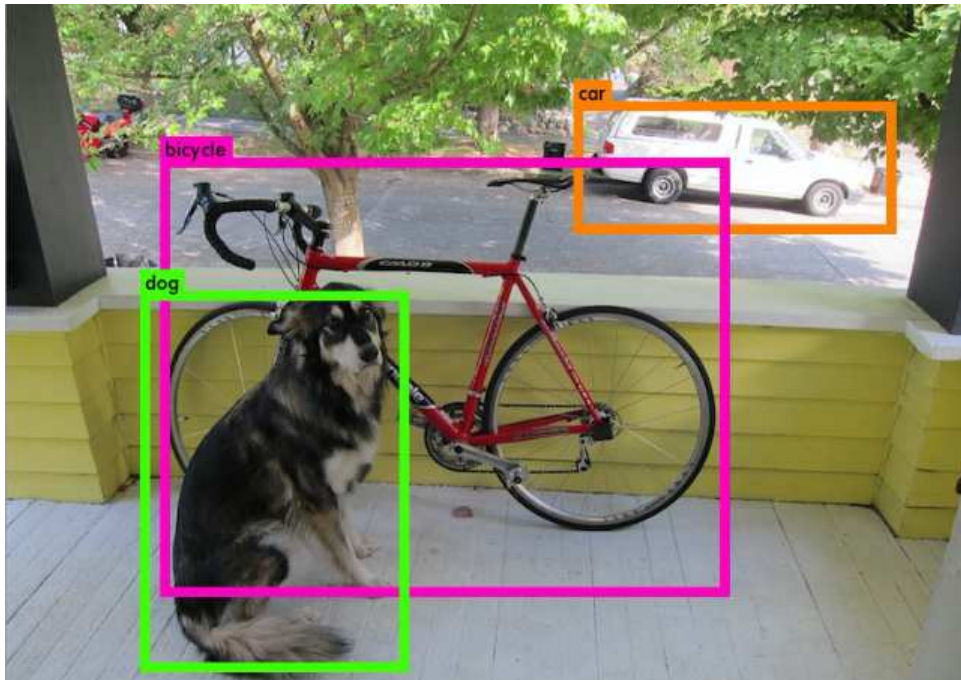
For Pascal VOC:

- 7x7 grid
- 2 bounding boxes / cell
- 20 classes



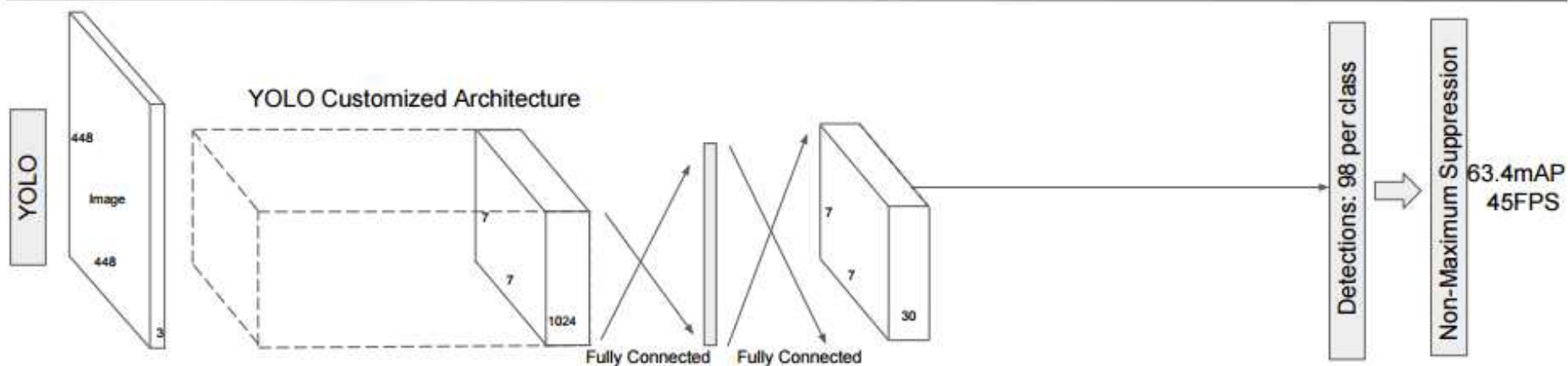
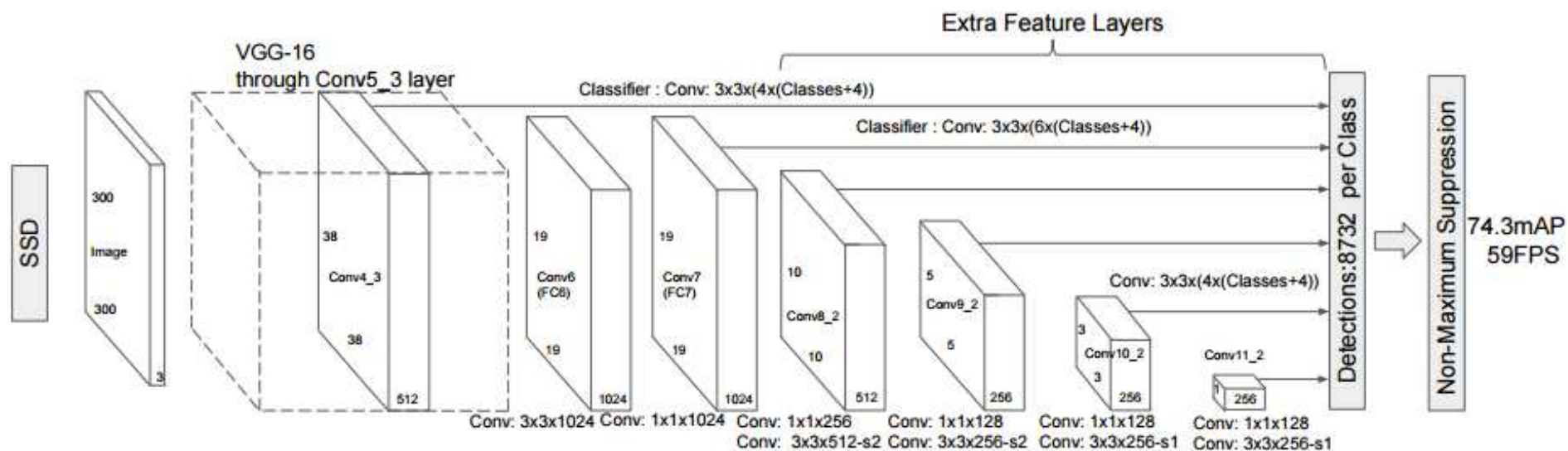
$$7 \times 7 \times (2 \times 5 + 20) = 7 \times 7 \times 30 \text{ tensor} = \mathbf{1470 \text{ outputs}}$$

Detection 예



SSD:

YOLO + default box shape + multi-scale



A solid blue vertical bar is positioned on the left side of the slide.

YOLO V2

Flexibility: 네트워크의 크기를 조절해서 FPS와 mAP를 조절 가능

성능 향상의 요인

1. Batch Normalization: 모든 CL에 배치 정규화
2. High Resolution Classifier: Classification network를 고해상도로 학습시킴
3. FCL를 CL로 대체
4. Anchor Boxes 사용 : 경계박스를 처음부터 직접 예측-> 앵커박스를 초기값으로 사용하여 예측
5. Darknet-19 (new network)
6. Dimension Clusters: 실제 경계박스들을 클러스터링 하여 최적의 앵커박스를 찾음
7. Direct location prediction: 경계 박스 위치는 직접 예측
8. Pass-through: 26x26 중간 특징 맵을 skip하여 13x13 레이어에 붙임
9. Multi-Scale Training: 학습데이터의 크기를 다양한 스케일로 학습
10. Fine-Grained Features: 최종 특징 맵의 크기를 7x7에서 13x13으로 키움

YOLO V2

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

Better

1. Batch normalization

모든 convolutional layer에 batch normalization

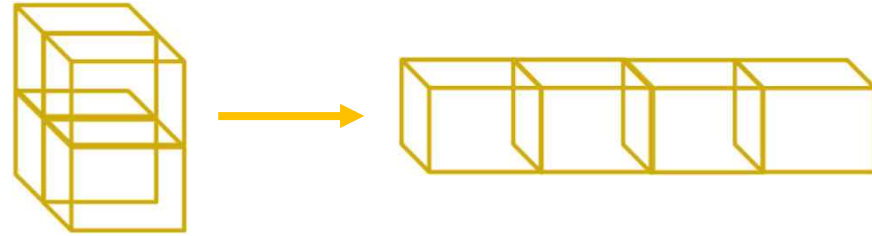
2. High-resolution classifier

Classifier network를 448 x 448 for 10 epochs에 대해 학습함

3. Convolutional with Anchor Boxes

- FC제거, anchor boxes 사용
- 448 input images 대신 416으로 shrink
from 416x416 to 13x13

Better



4. Fine-grained features

- Pass-through layer 추가
: features from an earlier layer at 26x26 resolutions
- 26x26x512 feature map \rightarrow 13x13x2048 feature map

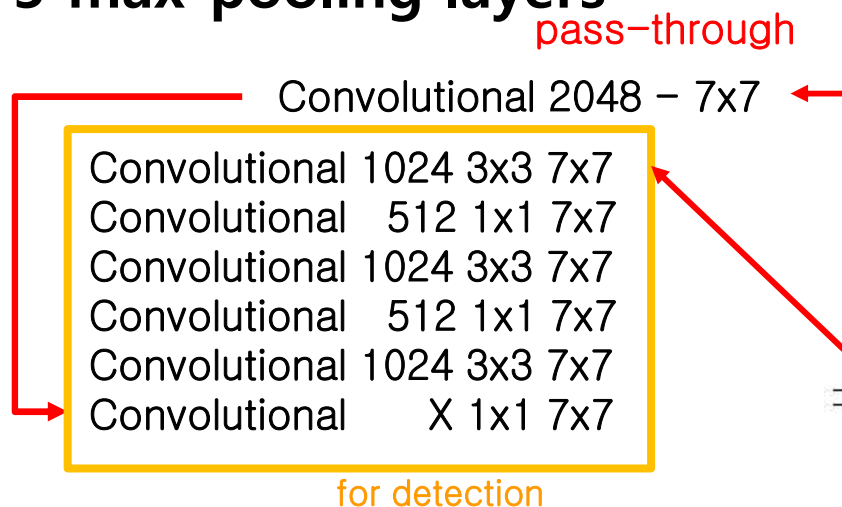
5. Multi-scale Training

- Every 10 batches,
randomly chooses a new image dimension size [320, 352, ..., 608]

YOLO V2

Faster – Darknet-19 (91%/93.3% for ImageNet)

- 3x3 conv. Filters 사용
- Pooling 후에 채널 수 두 배로
- 1x1 filters between 3x3 filters
- Global average pooling
- 19 convolutional layers
and 5 max-pooling layers



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

Faster – Darknet-19

Training for classification

- ImageNet 1000 class for 160 epochs
- Data augmentation
- Initial training 224x224 → 448x448 fine-tuning

Training for detection

- VOC: 5 boxes with 5 coordinates and 20 classes per box
 - 160 epochs with varying learning rate
-

YOLO V2

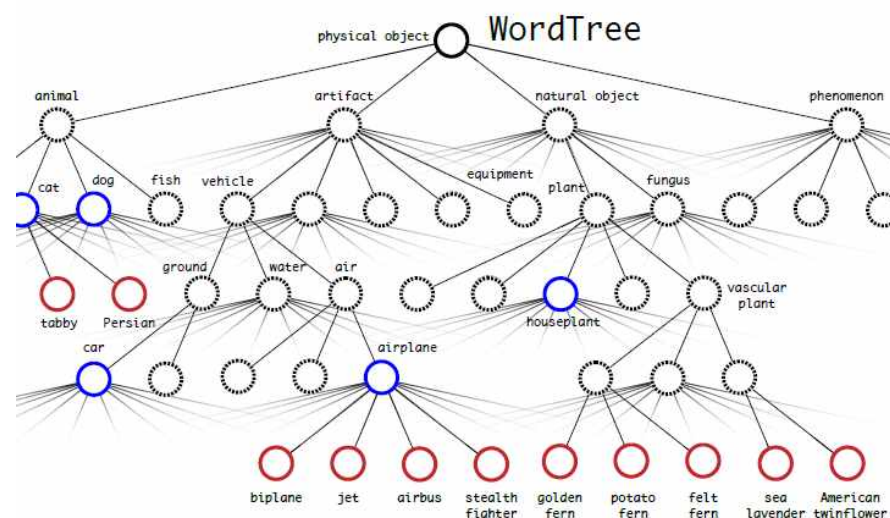
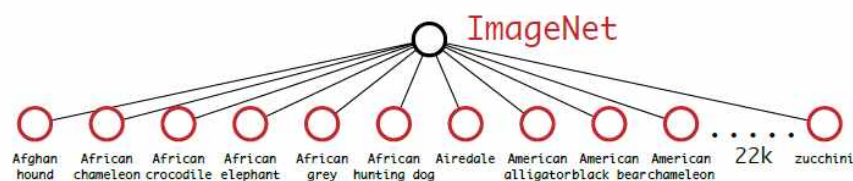
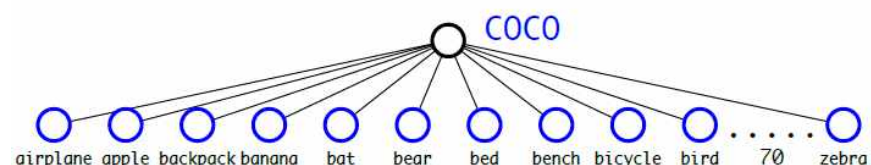
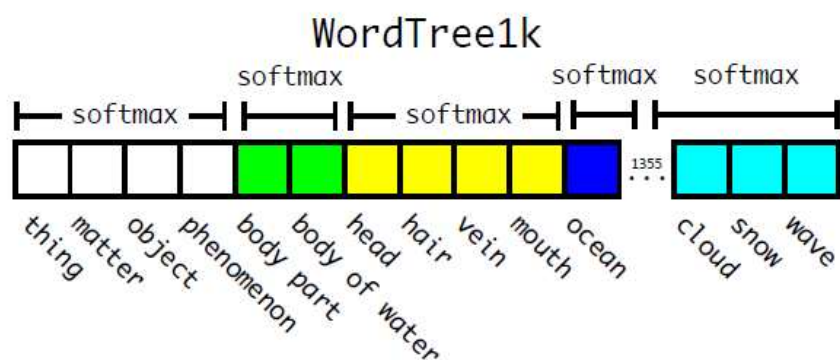
Stronger – YOLO9000

Jointly training on classification and detection data

- WordTree: a hierarchical model of visual concepts

- When seeing detection-data, full YOLOv2 loss function

- When seeing classification-data, loss from the classification-specific parts



Yolo With Jetson TX2

Jetson TX2 Developer Kit

► Jetson TX2 Developer Kit

JETSON TX2 MODULE

- NVIDIA Pascal™ Architecture GPU
- 2 Denver 64-bit CPUs + Quad-Core A57
- 8GB L128 bit DDR4 Memory
- 32GB eMMC 5.1 storage
- 802.11ac Wifi and bluetooth

JETSON Camera Module

- 5MP Fixed Focus MIPI CSI Camera

I/O

- USB 3.0 Type A
- USB 2.0 Micro AB(Support Recovery and Host Mode)
- HDMI
- Gigabit Ethernet
- PCI-E x4
- SATA data & Power
- GPIOs, I2C, SPI, CAN
- Camera Expansion Header

...



Jetson TX2 Developer Kit

► Jetpack (<https://developer.nvidia.com/embedded/jetpack>)

- . NVIDIA Embedded AI Solution
- . Include OS Images, Libraries and APIs, development tools, samples and documents

Jetpack 3.3

OS	Linux4Tegra 28.2.1 -. Ubuntu, linux kernel, bootloader, NVIDIA drivers, flashing Utilities
TensorRT	Version 4.0 High performance deep learning Inference runtime
cuDNN	Version 7.1.5 CUDA Deep Neural Network library support for convolutions, activation functions and tensor transformations
CUDA	provides a comprehensive development environment for C and C++ developers building GPU-accelerated applications.
Multimedia API	Camera applications API(libargus) : low-level frame-synchronous API Sensor driver API(V4L2 API) : enables video decode, encode, format conversion and scaling functionalities.
Computer Vision	VisionWorks Software development package for CV and image processing OpenCV
Dev Tools	CUDA Toolkit NVIDIA System Profiler NVIDIA Nsight Graphics

Jetson Inference

▶ Jetson Inference (<https://github.com/dusty-nv/jetson-inference>)

- NVIDIA DIGITS와 TensorRT를 이용한 임베디드 DL에 대한 가이드 제공
- NVIDIA GPU Cloud 를 이용한 training guide
- Ubuntu DIGITS를 이용한 training guide
- Classification(ImageNet), Detection(DetectNet), Segmentation(SegNet) 기능 제공
- TensorRT 사용

Live Camera Demo

```
$ ./detectnet-camera coco-bottle    # detect bottles/soda cans in the camera
$ ./detectnet-camera coco-dog       # detect dogs in the camera
$ ./detectnet-camera multyped       # run using multi-class pedestrian/luggage detector
$ ./detectnet-camera.pednet         # run using original single-class pedestrian detector
$ ./detectnet-camera facenet        # run using facial recognition network
$ ./detectnet-camera                # by default, program will run using multyped
```

▶ OpenCV3 for Jetson TX2

<https://jkjung-avt.github.io/opencv3-on-tx2/>

YOLO with Jetson TX2

► YOLO (<https://pjreddie.com/darknet/yolo/>)

Detection Using A Pre-trained Model

1) Download source

```
git clone https://github.com/pjreddie/darknet  
cd darknet
```

2) Modify Makefile

```
GPU=1  
CUDNN=1  
OPENCV=1
```

3) Make

4) Download Pre-trained weight file

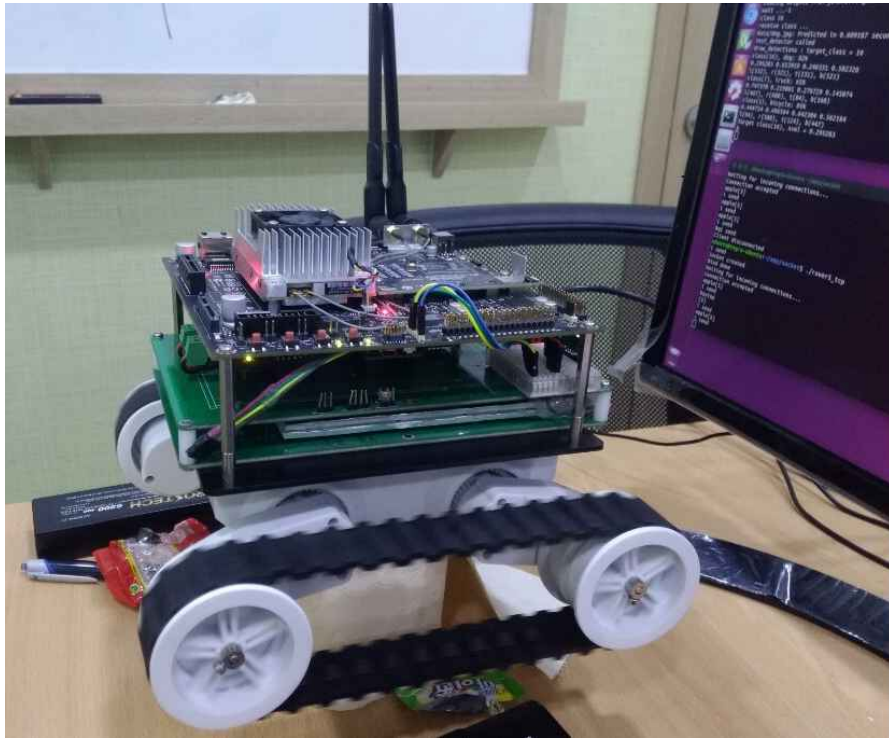
```
wget https://pjreddie.com/media/files/yolov3.weights
```

5) Execute Detection using live camera

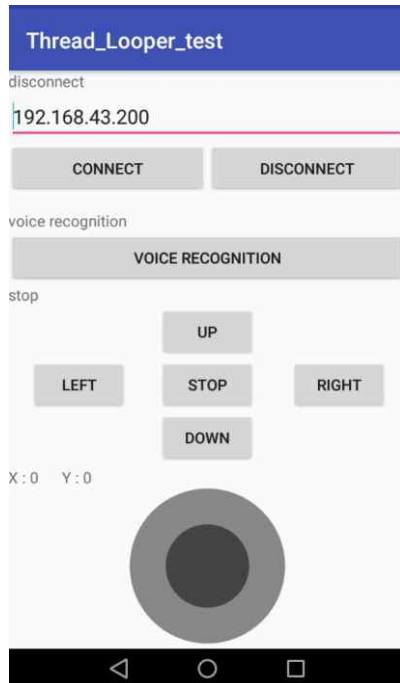
```
./darknet detector demo cfg/coco.data cfg/yolov3-tiny.cfg yolov3-tiny.weights "nvcamerasrc !  
video/x-raw(memory:NVMM),width=(int)1280, height=(int)720,format=(string)I420,  
framerate=(fraction)30/1 ! nvvidconv flip-method=0 ! video/x-raw,format=(string)BGRx !  
videoconvert ! video/x-raw, format=(string)BGR ! appsink"
```

Object Tracking 구현

사용되는 장비 사진



장비 구성



Smart Phone

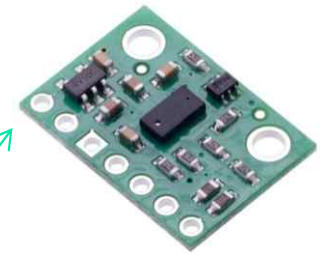


WIFI 통신

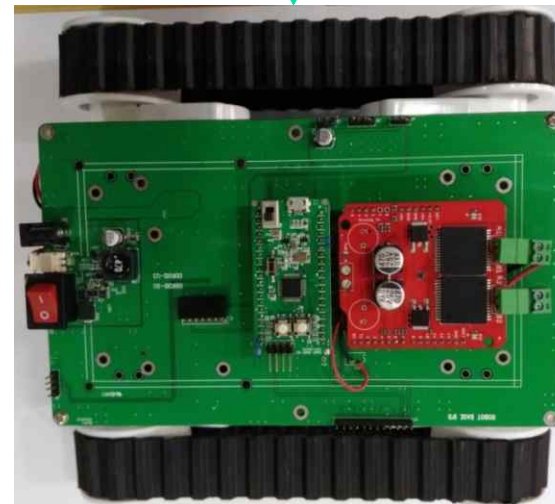


Serial 통신

JETSON TX2

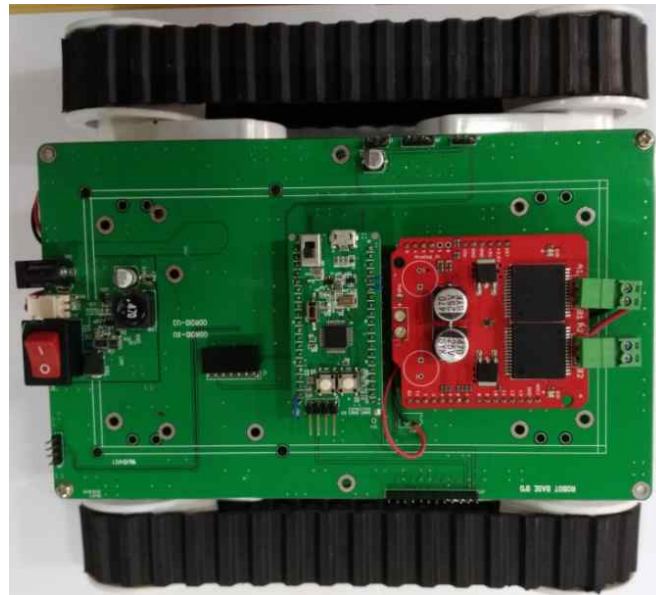
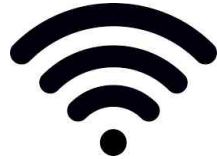
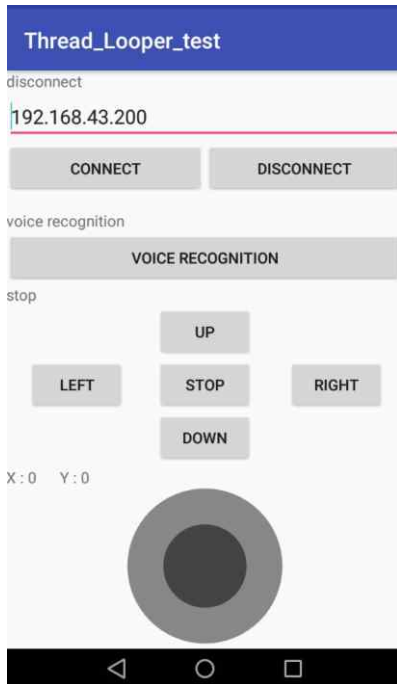


Laser 센서



구동체

최종 결과물



1. 물체 인식 후 레이저 센서에 의해서 일정한 거리를 유지 하면서 물체를 따라감
2. 스마트폰에서 오렌지 하면 이를 jetson에 전달하여 오렌지가 있는 곳으로 감

구현 세부 내용

1. Jetson TX2 설치

- ✓ Jetpack install
 - ▣ 4.2 이 최신이므로 4.2로 진행
- ✓ System Setup
- ✓ Building from Source on Jetson

JetPack

NVIDIA JetPack SDK는 AI 응용 프로그램을 작성하는 가장 포괄적인 솔루션입니다. JetPack 인스톨러를 사용하여 최신 OS 이미지로 Jetson Developer Kit을 플래시하고, 호스트 PC 및 개발자 키트용 개발자 도구를 설치하고 개발 환경을 시작하는 데 필요한 라이브러리 및 API, 샘플 및 문서를 설치하십시오.

JetPack 4.2

JetPack 4.2는 Jetson AGX Xavier, Jetson TX2 시리즈 모듈 및 Jetson Nano를 지원하는 최신 프로젝트 릴리스입니다. 주요 기능으로는 LTS Kernel 4.9 지원, 새로운 Jetson.GPIO Python 라이브러리, TRT Python API 지원 및 GStreamer 프레임 워크 용 새 가속 렌더러 플러그인이 있습니다.

이 릴리스에서 사용 가능한 새로운 기능에 대한 요약은 아래 하이라이트를 참조하고 향후 릴리스에서 계획된 추가 기능에 대한 정보를 포함하여 자세한 내용은 JetPack 릴리스 정보를 참조하십시오.

JetPack 설치 :

Jetson Nano 개발자 키트

아래의 SD 카드 이미지를 다운로드하십시오.

[SD 카드 이미지 다운로드](#)

그리고 [Jetson Nano Developer Kit 시작하기](#)의 단계를 따르십시오.

Jetson AGX Xavier, TX2 및 Nano 개발자 키트

NVIDIA SDK 관리자를 다운로드하여 JetPack을 설치하십시오.

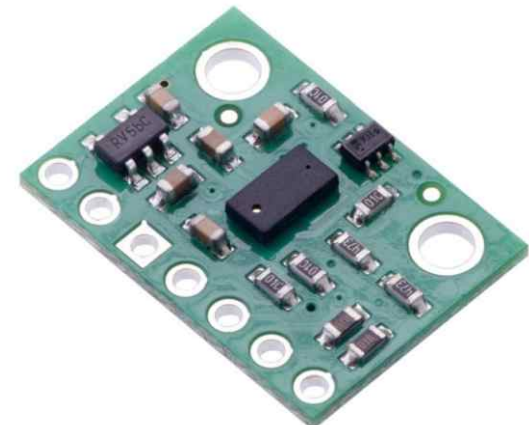
[NVIDIA SDK 관리자 다운로드](#)

2. JetsonTX2 디바이스 제어

- ✓ GPIO IN/OUT 제어
- ✓ IMU 센서 제어
- ✓ 레이저 센서 제어
- ✓ 시리얼 통신 구현
 - STM32 테스트 프로그램 구현
- ✓ WIFI 통신 구현
 - 간단한 소켓 구현
- ✓ Motor 제어



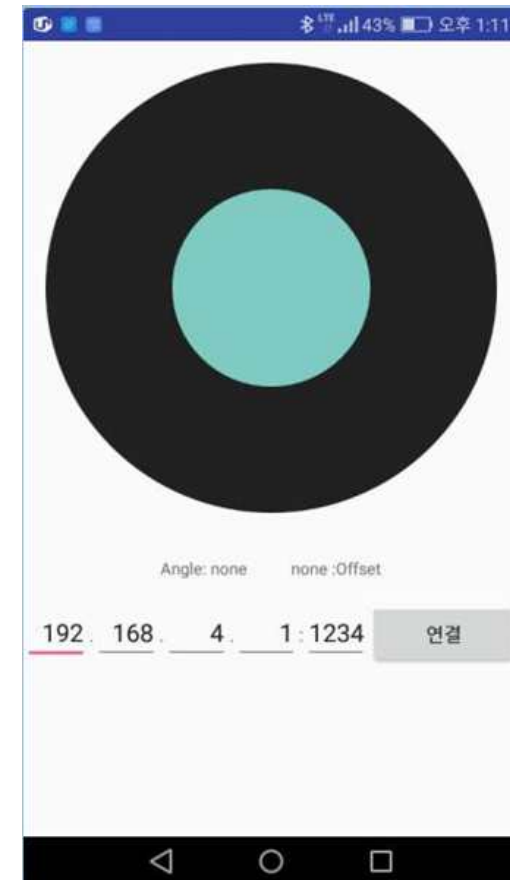
MPU6050 : 6-DOF IMU



VL53L0X Time-of-Flight Distance Sensor,
200cm Max

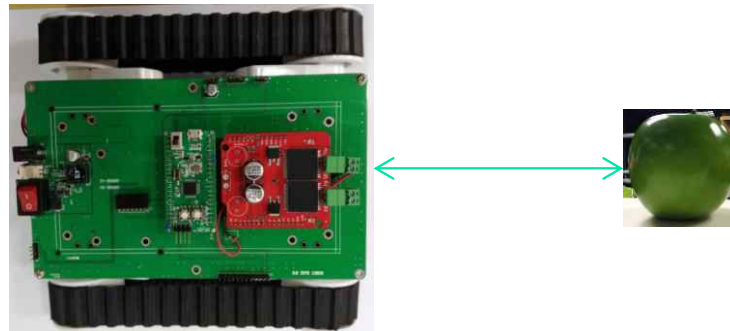
3. 안드로이드 어플(3시간)

- ✓ WiFi 통신
- ✓ 조이스틱
- ✓ 음성인식



4. YOLO 기반 물체 추적

- ✓ Yolo 활용 object detect
- ✓ 일정 거리를 두고 사물을 따라가는 기능



- ✓ 스마트폰을 사물의 이름을 전달하여 따라감



감사합니다