# Advanced Computer Vision

## Annotation/Object detection/Segmentation

**220929**

**고 우 영**

https://github.com/airobotlab/lecture_5_yolov5

# Requirement

- **Requirement**
  - 카메라에서 실시간으로 영상정보 획득 from 천장 (실내, 조명/위치 고정)
  - **4fps**(1/4초) 객체인식
  - 객체인식 위치정보 전송 to PLC
- **개발 전 고려사항**
  - 입력 이미지 사이즈 (3000x4000?, 2000x1000?, 640x640?, 224x224?)
  - 어떤 machine?
    - 노트북, 미니컴(Rpi, JetsonNano, Jetson orin), 서버, Cloud
    - GPU 사용 가능 여부
    - 재학습 여부(Continual learning or re-training)
  - Machine 사양에 따른 모델 선정
    - 크고 성능이 좋은 모델, 작고 빠른 모델, 모델 최적화 필요 여부 등
  - Python? C?

# 입력크기/속도

4 fps -> 250 ms/image 이하

Nano

YOLOv5n

4 $MB_{FP16}$
6.3 $ms_{V100}$
28.4 $mAP_{COCO}$

Small

YOLOv5s

14 $MB_{FP16}$
6.4 $ms_{V100}$
37.2 $mAP_{COCO}$

Medium

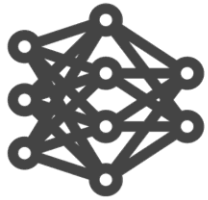
YOLOv5m

41 $MB_{FP16}$
8.2 $ms_{V100}$
45.2 $mAP_{COCO}$

Large

YOLOv5l

89 $MB_{FP16}$
10.1 $ms_{V100}$
48.8 $mAP_{COCO}$



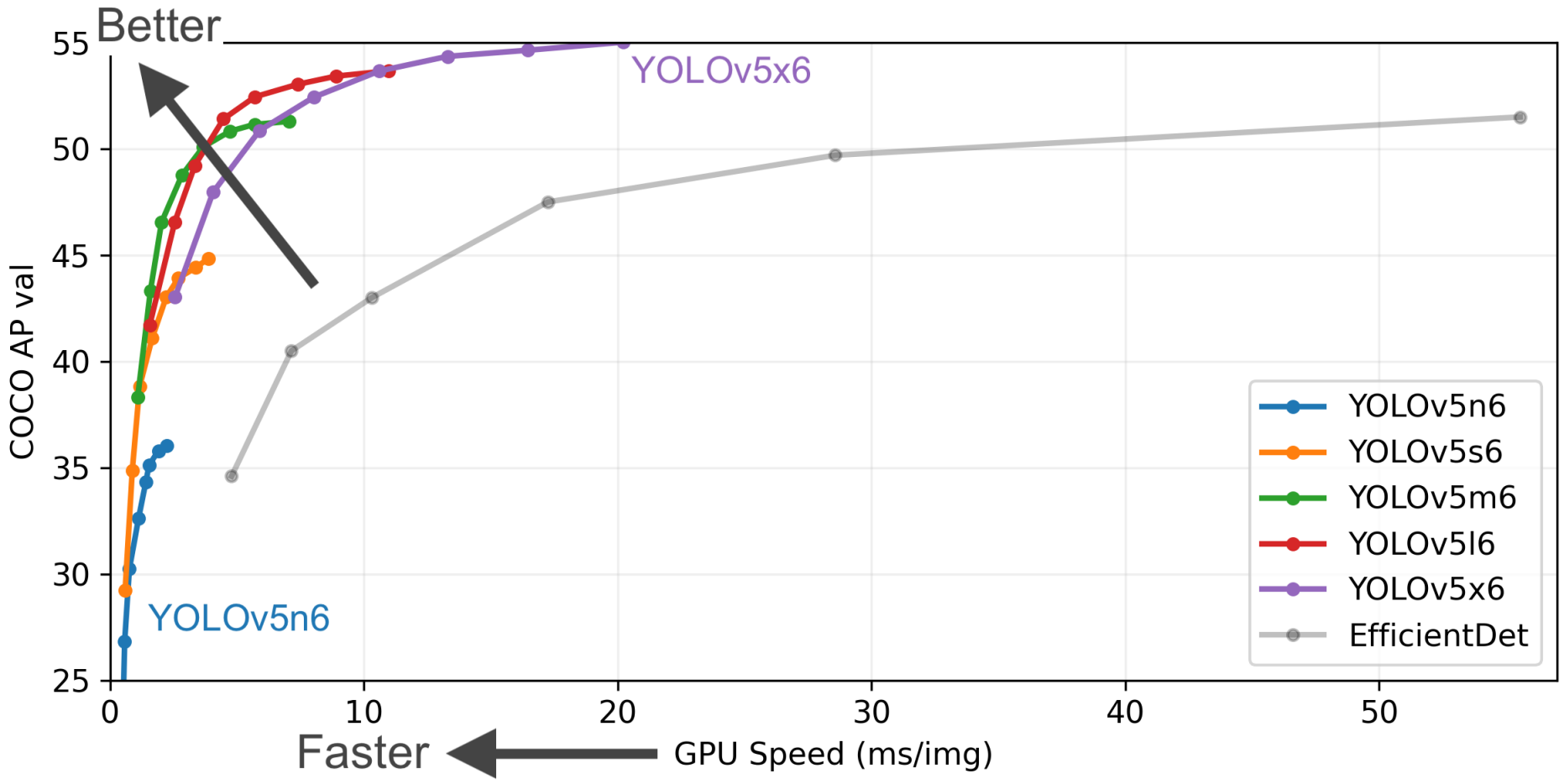
XLarge

YOLOv5x

166 $MB_{FP16}$
12.1 $ms_{V100}$
50.7 $mAP_{COCO}$

# 입력크기/속도

4 fps -> 250 ms/image 이하



# 입력크기/속도

4 fps -> 250 ms/image 이하

| Model | size (pixels) | mAP$^{val}$ 0.5:0.95 | mAP$^{val}$ 0.5 | Speed CPU b1 (ms) | Speed V100 b1 (ms) | Speed V100 b32 (ms) | params (M) | FLOPs @640 (B) |
|---|---|---|---|---|---|---|---|---|
| YOLOv5n | 640 | 28.0 | 45.7 | **45** | **6.3** | **0.6** | **1.9** | **4.5** |
| YOLOv5s | 640 | 37.4 | 56.8 | 98 | 6.4 | 0.9 | 7.2 | 16.5 |
| YOLOv5m | 640 | 45.4 | 64.1 | 224 | 8.2 | 1.7 | 21.2 | 49.0 |
| YOLOv5l | 640 | 49.0 | 67.3 | 430 | 10.1 | 2.7 | 46.5 | 109.1 |
| YOLOv5x | 640 | 50.7 | 68.9 | 766 | 12.1 | 4.8 | 86.7 | 205.7 |
| | | | | | | | | |
| YOLOv5n6 | 1280 | 36.0 | 54.4 | 153 | 8.1 | 2.1 | 3.2 | 4.6 |
| YOLOv5s6 | 1280 | 44.8 | 63.7 | 385 | 8.2 | 3.6 | 12.6 | 16.8 |
| YOLOv5m6 | 1280 | 51.3 | 69.3 | 887 | 11.1 | 6.8 | 35.7 | 50.0 |
| YOLOv5l6 | 1280 | 53.7 | 71.3 | 1784 | 15.8 | 10.5 | 76.8 | 111.4 |
| YOLOv5x6<br>+ TTA | 1280<br>1536 | 55.0<br>**55.8** | 72.7<br>**72.7** | 3136<br>- | 26.2<br>- | 19.4<br>- | 140.7<br>- | 209.8<br>- |

# NVIDIA TRT Optimization

4 fps -> 250 ms/image 이하

- **만약 더 빠르게 하고 싶으면?**
  - 그래프/Floating point 최적화
- **만약 python을 못쓰는 환경이면?**
  - 언어 변환 필요
  - Embedded board: C언어
  - 웹: Java or C#
- **속도+언어 둘다 해결하고 싶다?**
  - ONNX 포팅
  - TRT에서 구동

# NVIDIA TRT Optimization

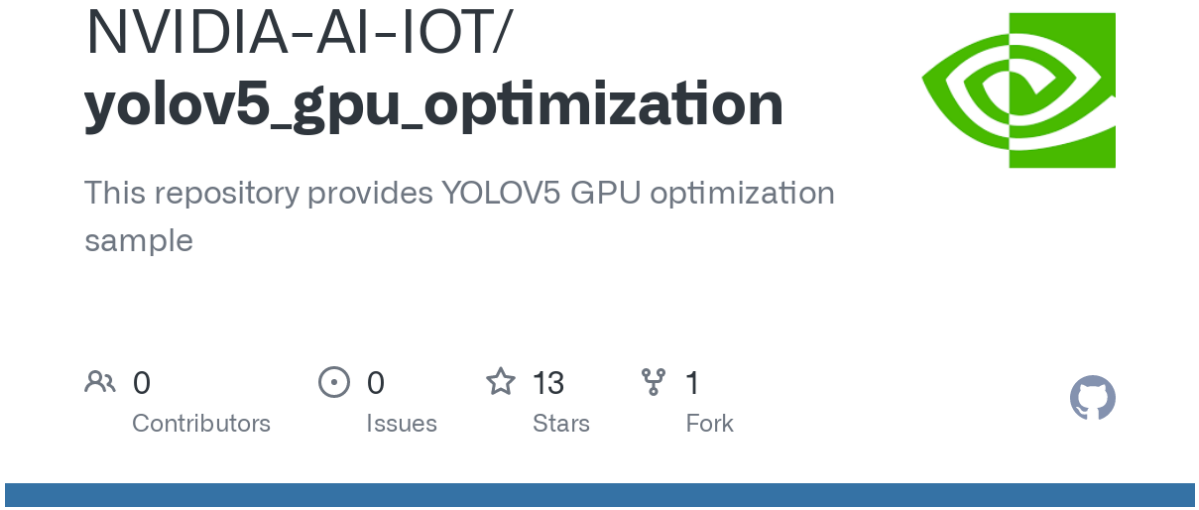
4 fps -> 250 ms/image 이하

```
# 모델에 대한 입력값
x = torch.randn(batch_size, 1, 224, 224, requires_grad=True)
torch_out = torch_model(x)

# 모델 변환
torch.onnx.export(torch_model,               # 실행될 모델
                  x,                         # 모델 입력값 (튜플 또는 여러 입력값들도 가능)
                  "super_resolution.onnx",   # 모델 저장 경로 (파일 또는 파일과 유사한 객체 모두 가능)
                  export_params=True,        # 모델 파일 안에 학습된 모델 가중치를 저장할지의 여부
                  opset_version=10,          # 모델을 변환할 때 사용할 ONNX 버전
                  do_constant_folding=True,  # 최적화시 상수폴딩을 사용할지의 여부
                  input_names = ['input'],   # 모델의 입력값을 가리키는 이름
                  output_names = ['output'], # 모델의 출력값을 가리키는 이름
                  dynamic_axes={'input' : {0 : 'batch_size'},    # 가변적인 길이를 가진 차원
                                'output' : {0 : 'batch_size'}})
```

- **속도+언어 둘다 해결하고 싶다?**
  - ONNX 포팅
  - TRT에서 구동

# NVIDIA TRT Optimization

4 fps -> 250 ms/image 이하




https://github.com/NVIDIA-AI-IOT/yolov5_gpu_optimization

# NVIDIA TRT Optimization

4 fps -> 250 ms/image 이하

NVIDIA TRT Optimization 후 속도 **2**배 증가, 정확도 큰 차이 없음

| Model | size (pixels) | mAP$^{val}$ 0.5:0.95 | mAP$^{val}$ 0.5 | Speed CPU b1 (ms) | Speed V100 b1 (ms) | Speed | params | FLOPs |
|---|---|---|---|---|---|---|---|---|
| YOLOv5n | 640 | 28.0 | 45.7 | **45** | **6.3** | | | |
| YOLOv5s | 640 | 37.4 | 56.8 | 98 | 6.4 | | | |
| YOLOv5m | 640 | 45.4 | 64.1 | 224 | 8.2 | | | |
| YOLOv5l | 640 | 49.0 | 67.3 | 430 | 10.1 | | | |
| YOLOv5x | 640 | 50.7 | 68.9 | 766 | 12.1 | | | |
| | | | | | | | | |
| YOLOv5n6 | 1280 | 36.0 | 54.4 | 153 | 8.1 | | | |
| YOLOv5s6 | 1280 | 44.8 | 63.7 | 385 | 8.2 | | | |
| YOLOv5m6 | 1280 | 51.3 | 69.3 | 887 | 11.1 | | | |
| YOLOv5l6 | 1280 | 53.7 | 71.3 | 1784 | 15.8 | | | |
| YOLOv5x6<br>+ TTA | 1280<br>1536 | 55.0<br>**55.8** | 72.7<br>**72.7** | 3136<br>- | 26.2<br>- | | | |

| Model | Input Size | precision | FPS bs=32 | FPS bs= 1 | mAP@0.5 |
|---|---|---|---|---|---|
| yolov5n | 640 | FP16 | 1295 | 448 | 45.9% |
| yolov5s | 640 | FP16 | 917 | 378 | 57.1% |
| yolov5m | 640 | FP16 | 614 | 282 | 64% |
| yolov5l | 640 | FP16 | 416 | 202 | 67.3% |
| yolov5x | 640 | FP16 | 231 | 135 | 68.5% |
| yolov5n6 | 1280 | FP16 | 341 | 160 | 54.2% |
| yolov5s6 | 1280 | FP16 | 261 | 139 | 63.2% |
| yolov5m6 | 1280 | FP16 | 155 | 99 | 68.8% |
| yolov5l6 | 1280 | FP16 | 106 | 68 | 70.7% |
| yolov5x6 | 1280 | FP16 | 60 | 45 | 71.9% |

# 개발 순서

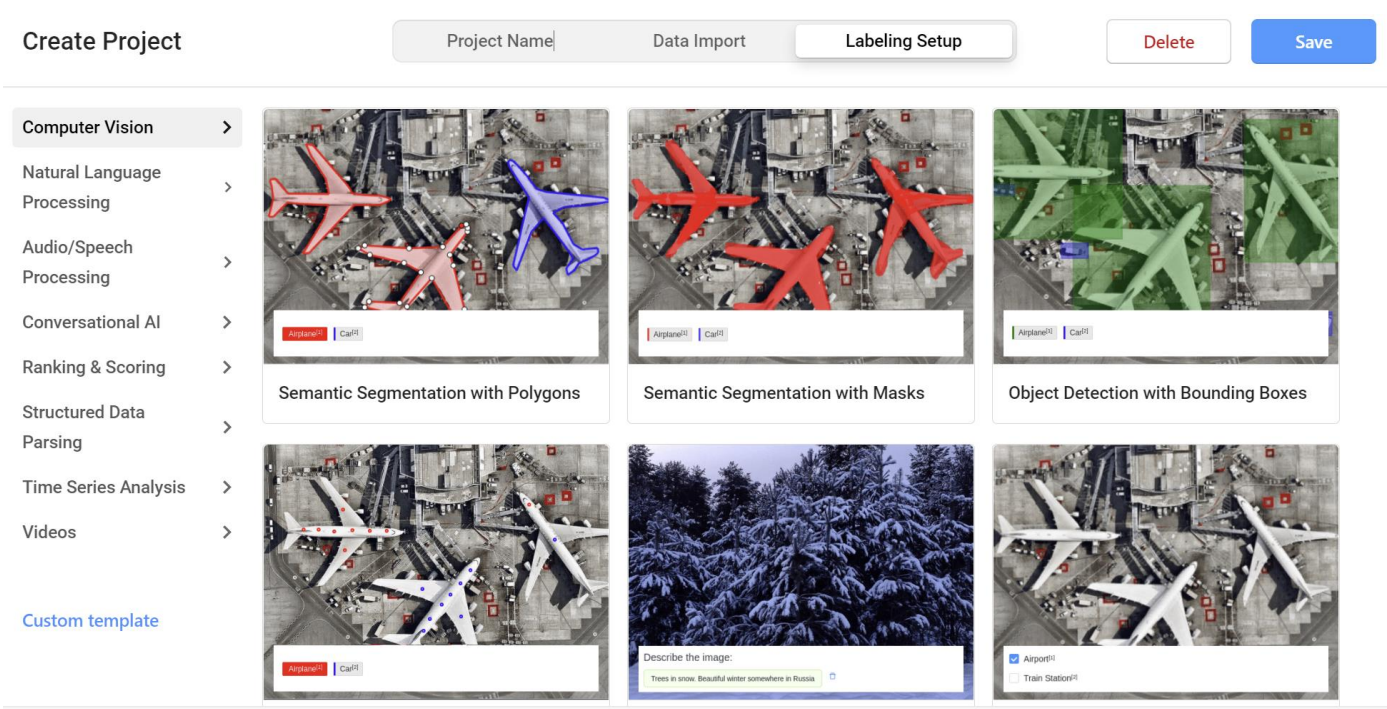
- **1) 데이터 Annotation**
  - Label-studio
- **2) Yolov5 학습**
  - https://github.com/ultralytics/yolov5
- **3) 학습모델을 이용한 추론**

# Annotation

**LabelStudio**

# Label Studio

- https://labelstud.io/
- **# Install the package**
  - **pip install -U label-studio**
- **# Launch it!**
  - **label-studio**



# Label Studio

- https://labelstud.io/
- **# Install the package**
  - **pip install -U label-studio**
- **# Launch it!**
  - **label-studio**


Output preview

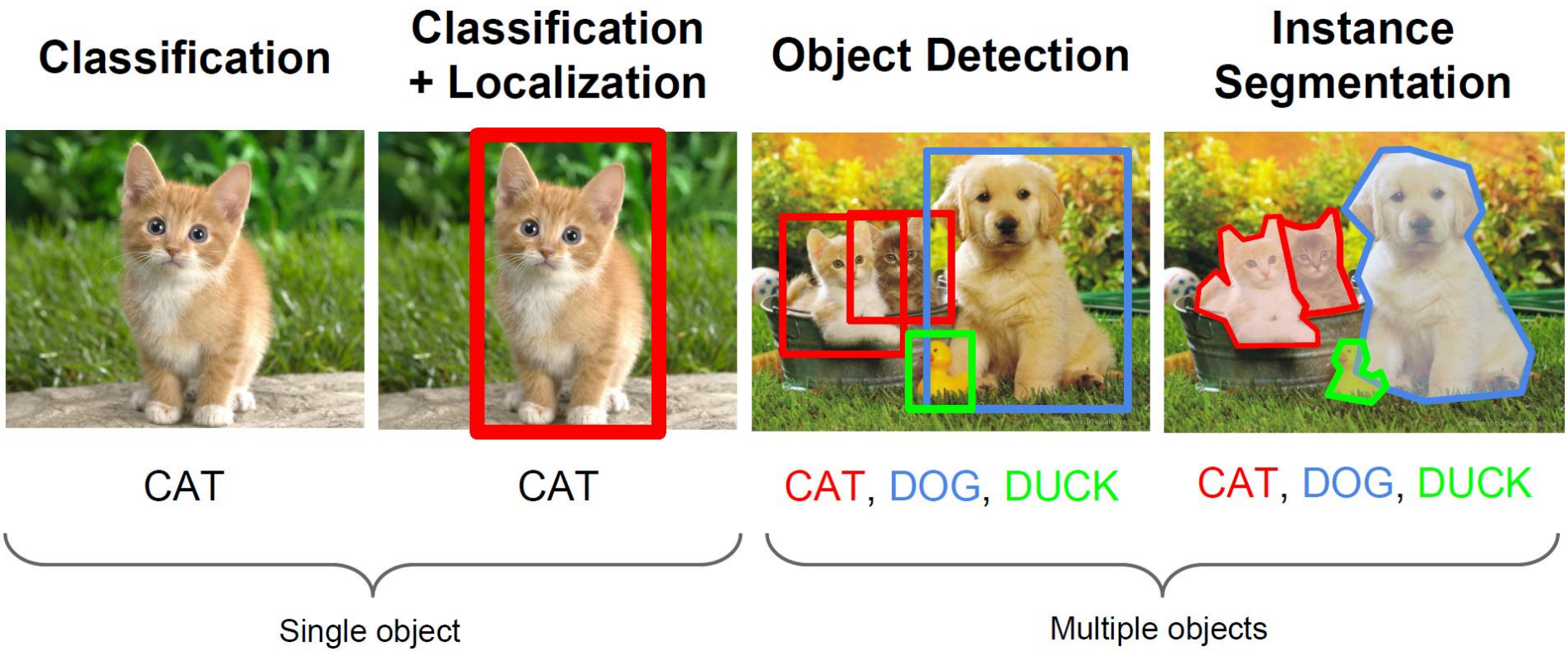

```
[
    {
        "original_width": 600,
        "original_height": 403,
        "image_rotation": 0,
        "value": {
            "x": 24.666666666666668,
            "y": 49.62779156327544,
            "width": 31.666666666666668,
            "height": 39.702233250620345,
            "rotation": 0,
            "rectanglelabels": [
                "Airplane"
            ]
        },
        "id": "yytSY7KW36",
        "from_name": "label",
        "to_name": "image",
        "type": "rectanglelabels"
    },
    {
        "original_width": 600,
        "original_height": 403,
        "image_rotation": 0,
        "value": {
            "x": 62.666666666666664,
            "y": 65.50868486352357,
            "width": 8.833333333333334,
            "height": 24.317617866004962,
            "rotation": 0,
            "rectanglelabels": [
                "Car"
            ]
        },
        "id": "ICNn84tJme",
        "from_name": "label",
        "to_name": "image",
        "type": "rectanglelabels"
    }
]
```

# Object Detection

## History/yolov5/DETR

# Image Understanding

- 이미지에서 물체(object)를 탐지한 후, 해당 물체의 클래스와 위치를 예측하는 task



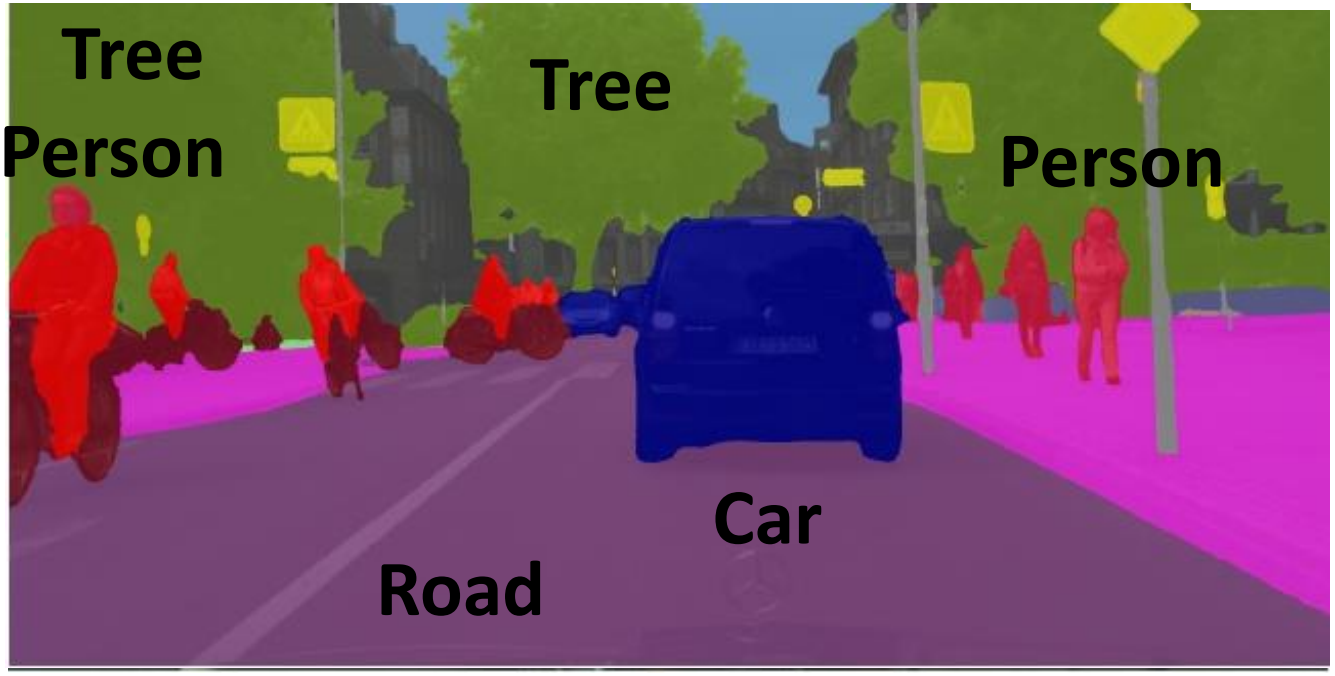



https://hoya012.github.io/blog/Tutorials-of-Object-Detection-Using-Deep-Learning-the-application-of-object-detection/


person.95
person.72
person.97
person.86
person.81
person.88
person.94
person.89
person.87
person.99
person.77
person.96
person.91
chair.98
chair.96
dining table.96
dining table.75
cup.75
cup.71
chair.99
chair.98
chair.95
dining table.78
chair.92
wine glass.80
chair.85
cup.71
chair.87
chair.97
chair.94
wine glass.91
wine glass.93
cup.96
cup.91
wine glass.83

# Image Understanding

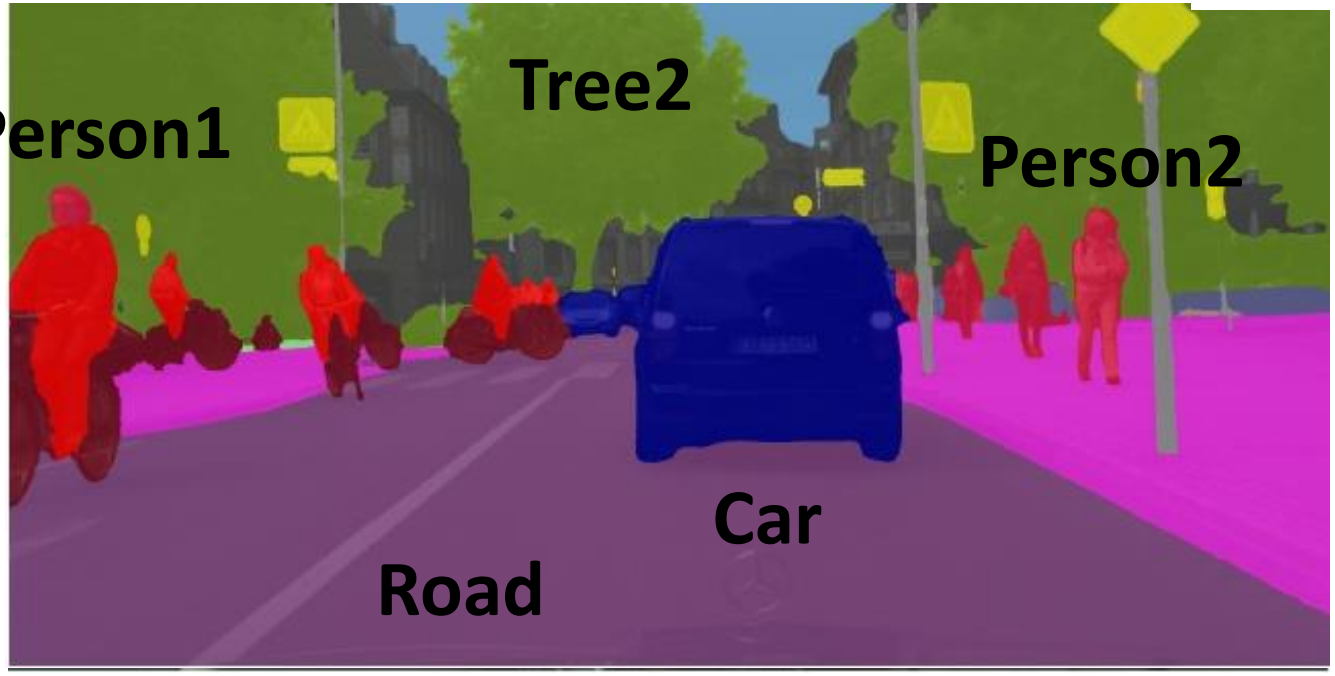
# Semantic Segmentation





TUM, Computer Vision 3: Detection, Segmentation and Tracking
https://kr.123rf.com/%EC%9D%BC%EB%9F%AC%EC%8A%A4%ED%8A%B8/%EC%B9%B4%EB%A9%94%EB%9D%BC.html

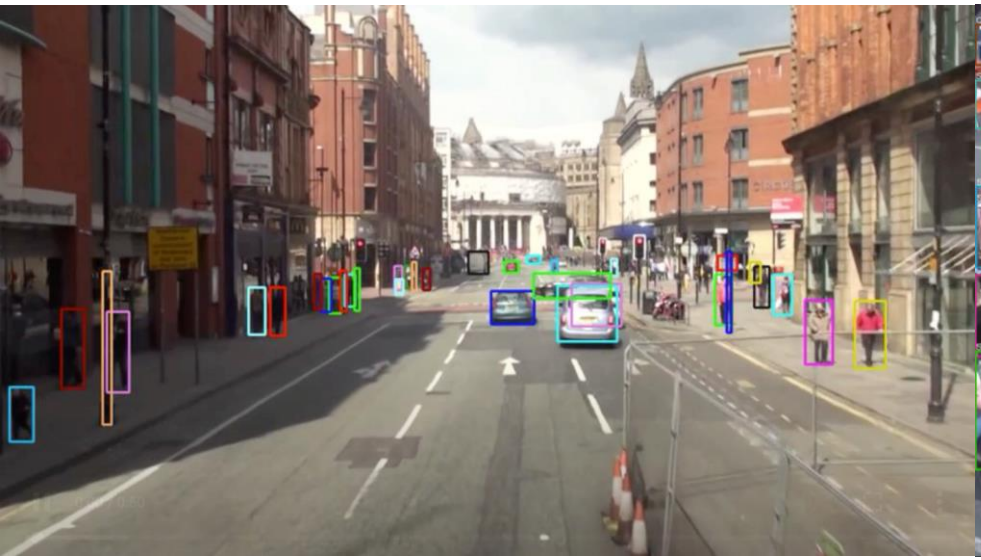
# Instance Segmentation

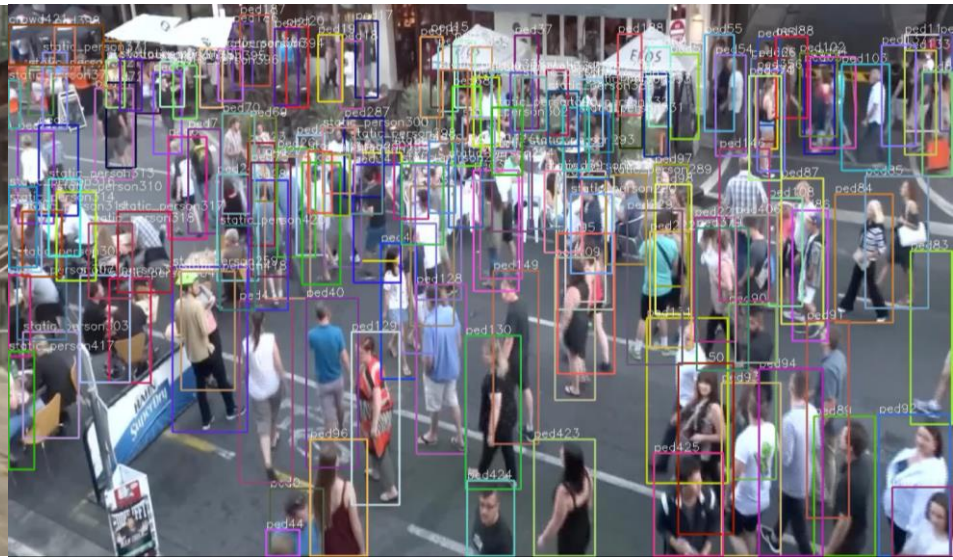




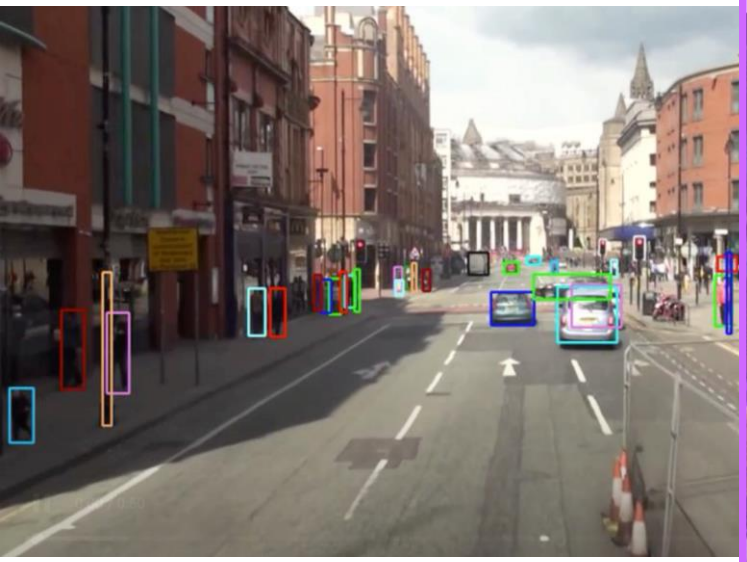
TUM, Computer Vision 3: Detection, Segmentation and Tracking
https://kr.123rf.com/%EC%9D%BC%EB%9F%AC%EC%8A%A4%ED%8A%B8/%EC%B9%B4%EB%A9%94%EB%9D%BC.html

# Video Understanding

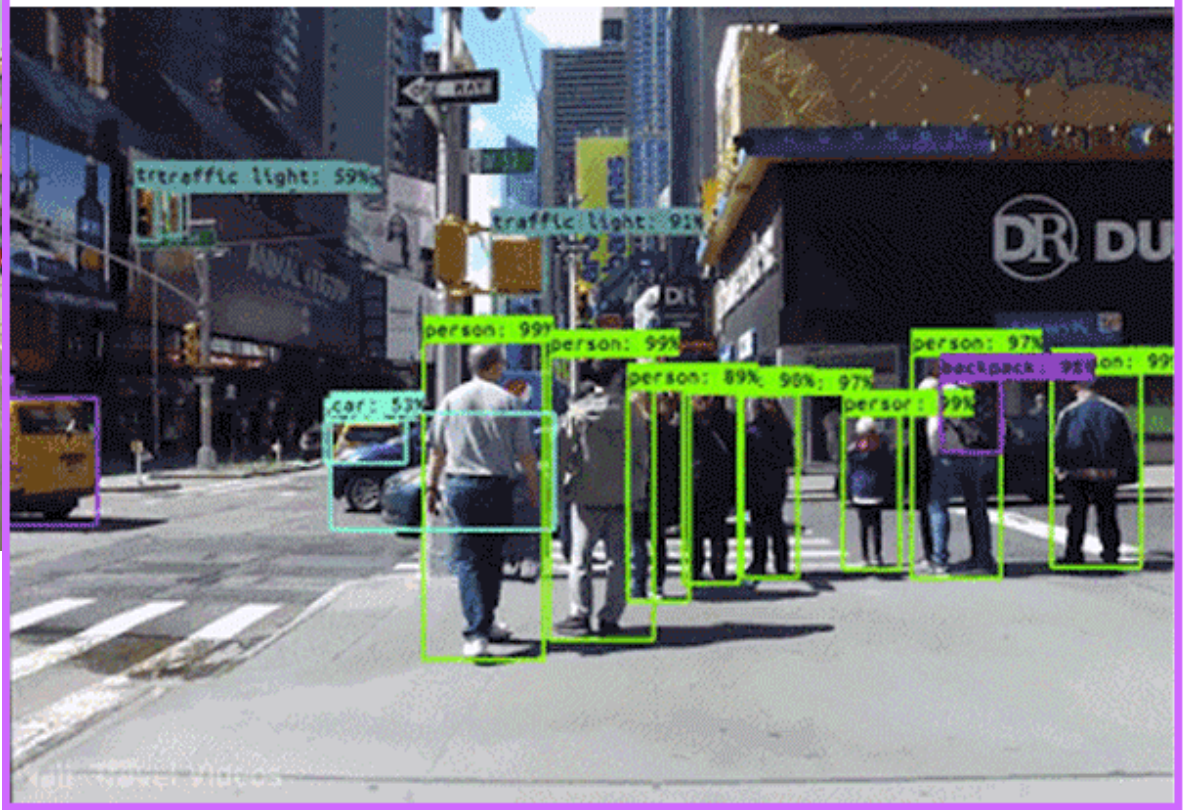




# Video Understanding



## 자율주행



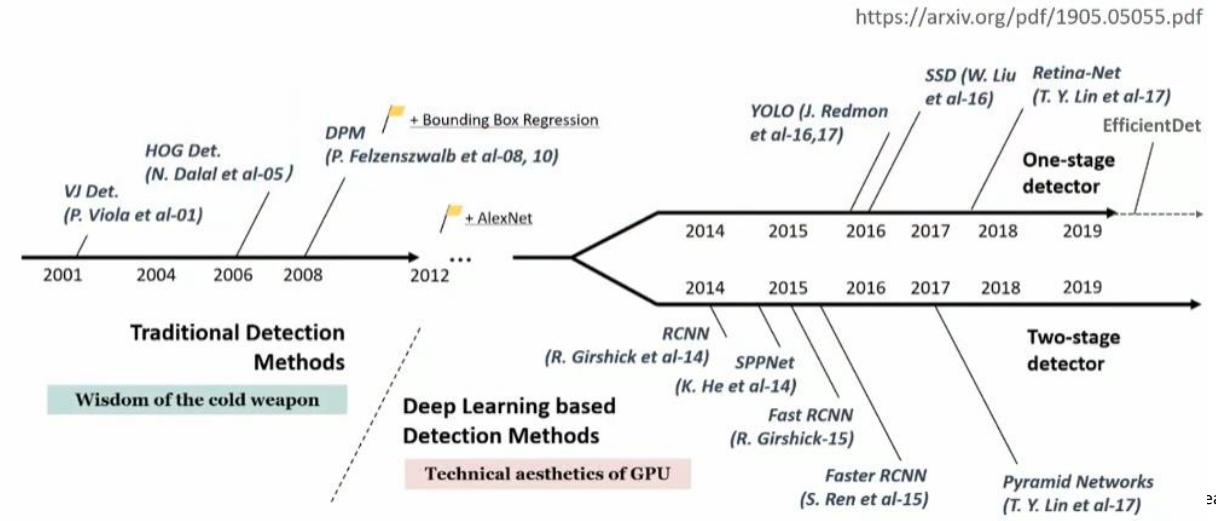

출처 : MEDIUM

# Image Understanding

1) CNN
2) 이미지 분류 고급
3) Object Detection
4) Object Tracking
5) Segmentation

# Object Detection

- 이미지에서 물체(object)를 탐지한 후, 해당 물체의 클래스와 위치를 예측하는 task
- Type
  - detector 앞단에 후보 지역을 추천하는 RPN(Region Proposal Network) 존재 여부
  - 1-stage detector : YOLO
    - 추론 속도가 빨라 real-time task
  - 2-stage detector: Faster R-CNN
    - 성능이 좋지만 느림

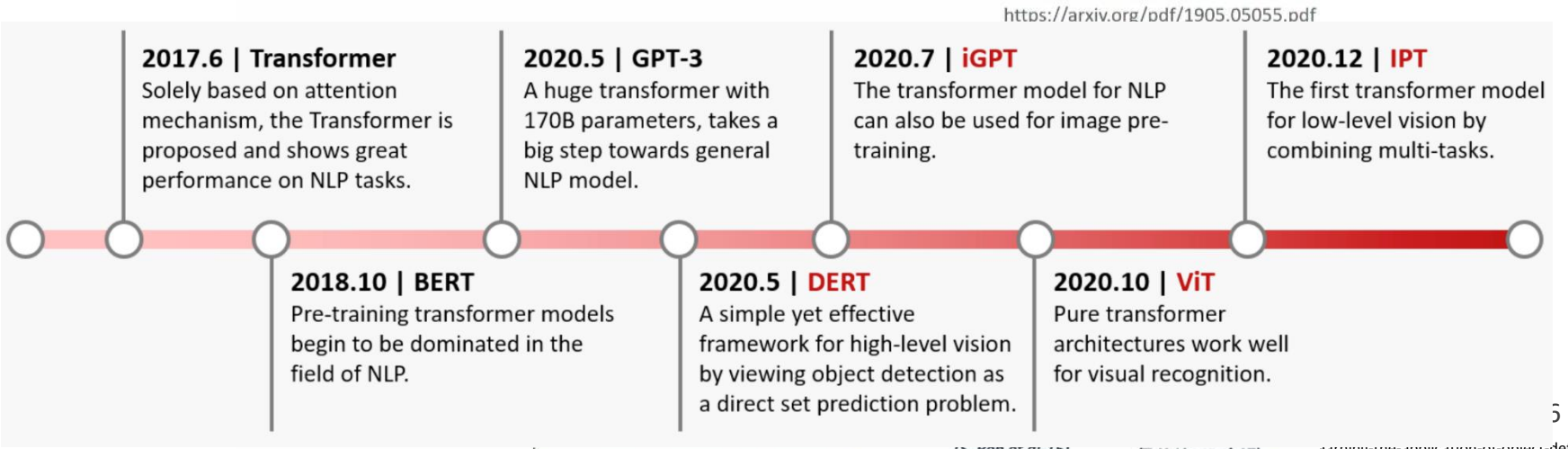





earning-the-application-of-object-detection/

# Object Detection

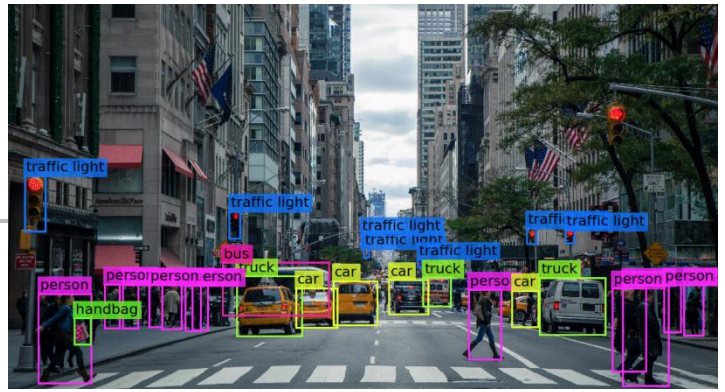
- 이미지에서 물체(object)를 탐지한 후, 해당 물체의 클래스와 위치를 예측하는 task
- Type
  - detector 앞단에 후보 지역을 추천하는 RPN(Region Proposal Network) 존재 여부
  - 1-stage detector : YOLO
    - 추론 속도가 빨라 real-time task
  - 2-stage detector: Faster R-CNN
    - 성능이 좋지만 느림

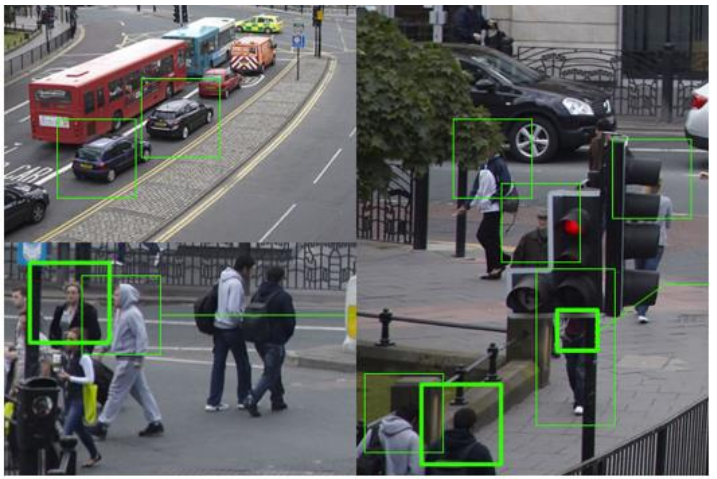



(T. Y. Lin et al-17)

# OD application

- 자율주행 자동차
- CCTV Surveillance
- OCR
- Aerial Image 분석
- 신체인식
- 제조업
- 스포츠 경기 분석
- 무인점포




Text json

1종대형 1종보통 1종소형 특수(대형견인 소형견인,구난)2종보통 2종소형 원동기

면허번호

11-95-123456-72

성명

클로바

주민등록번호

990602-1234567

http://mrkim.cloudy.so/board_DJpE79/174208
https://hoya012.github.io/blog/Tutorials-of-Object-Detection-Using-Deep-Learning-the-application-of-object-detection/
http://www.itdaily.kr/news/articleView.html?idxno=202118

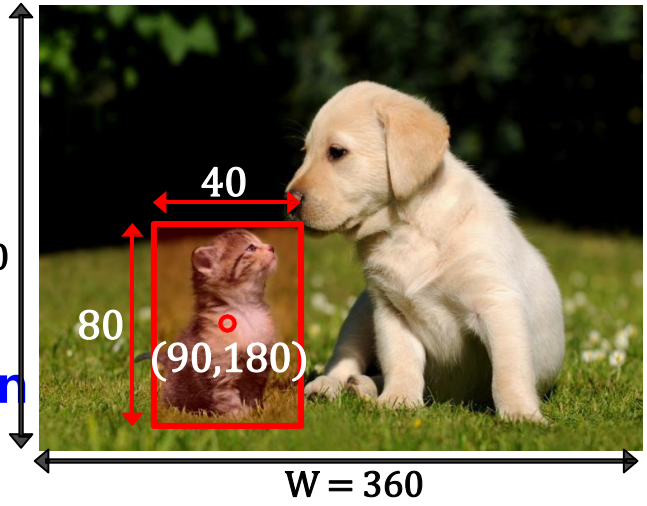
# Object Detection

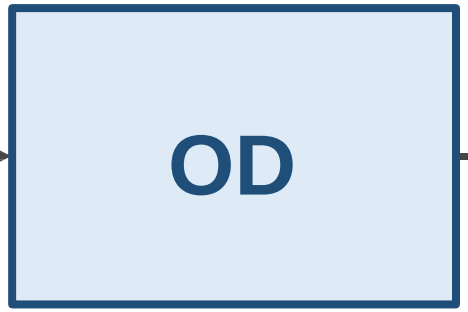
- **Task**
  - Find Bounding Box: **Regression**
  - Classify the Bounding Box: **Classification**

# Object Detection

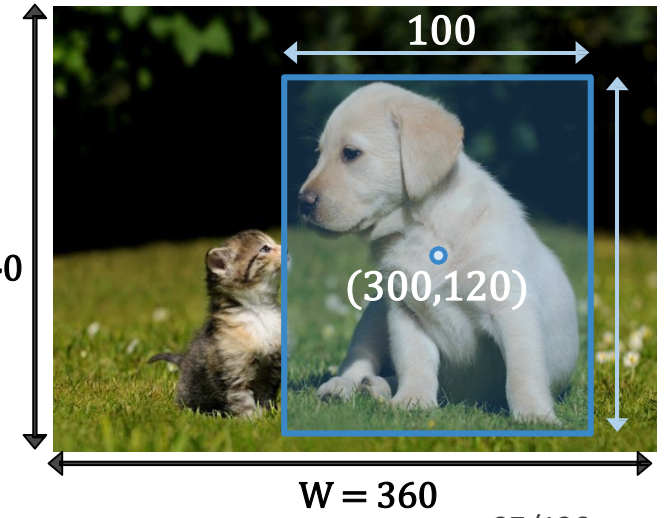

```
{
    "original_width": 600,
    "original_height": 403,
    "image_rotation": 0,
    "value": {
        "x": 24.666666666666668,
        "y": 49.62779156327544,
        "width": 31.666666666666668,
        "height": 39.702233250620345,
        "rotation": 0,
        "rectanglelabels": [
            "Airplane"
        ]
    },
    "id": "yytSY7KW36",
    "from_name": "label",
    "to_name": "image",
    "type": "rectanglelabels"
},
```


- **Task**
  - Find Bounding Box: **Regression**
    - (X, Y, W, H)
  - Classify the Bounding Box: **Classification**



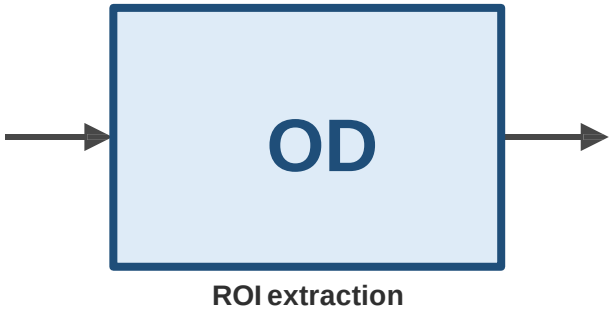




ROI extraction



# Object Detection

- **Task**
  - Find Bounding Box: **Regression**
    - (X, Y, W, H)
    - Each image nees a different number of outputs




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

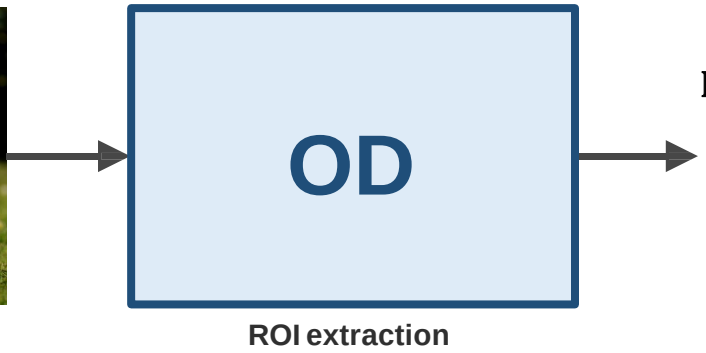
DOG: (x, y, w, h)
DOG: (x, y, w, h) **16**
CAT: (x, y, w, h)

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

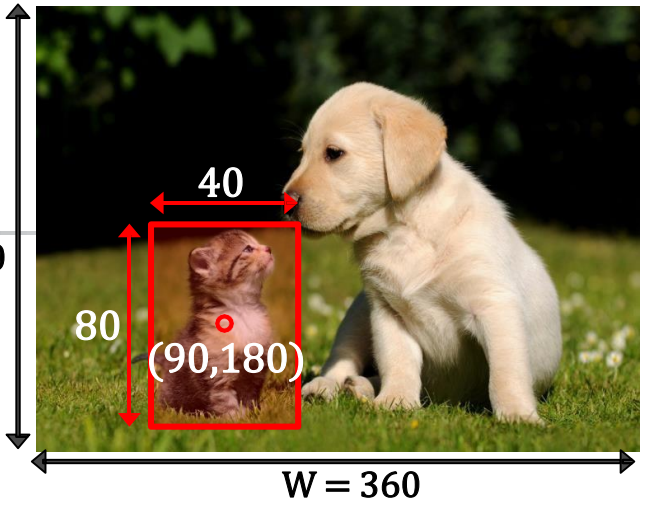
**Many!**

# Object Detection

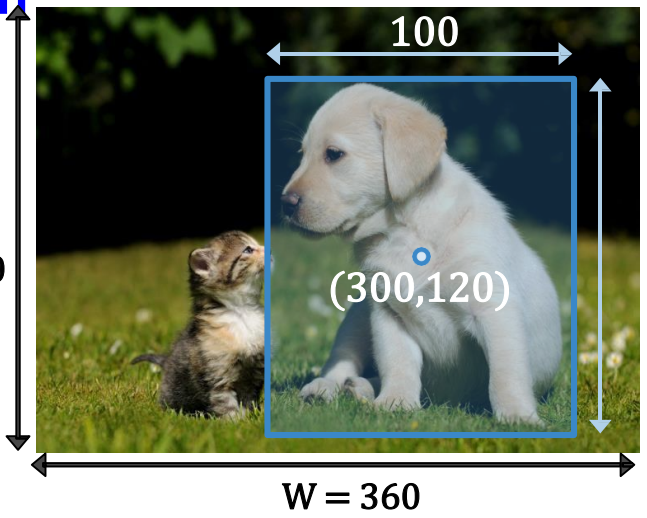
- **Task**
  - Find Bounding Box: **Regression**
    - (X, Y, W, H)
  - Classify the Bounding Box: **Classification**
    - Dog or Cat




ROI extraction



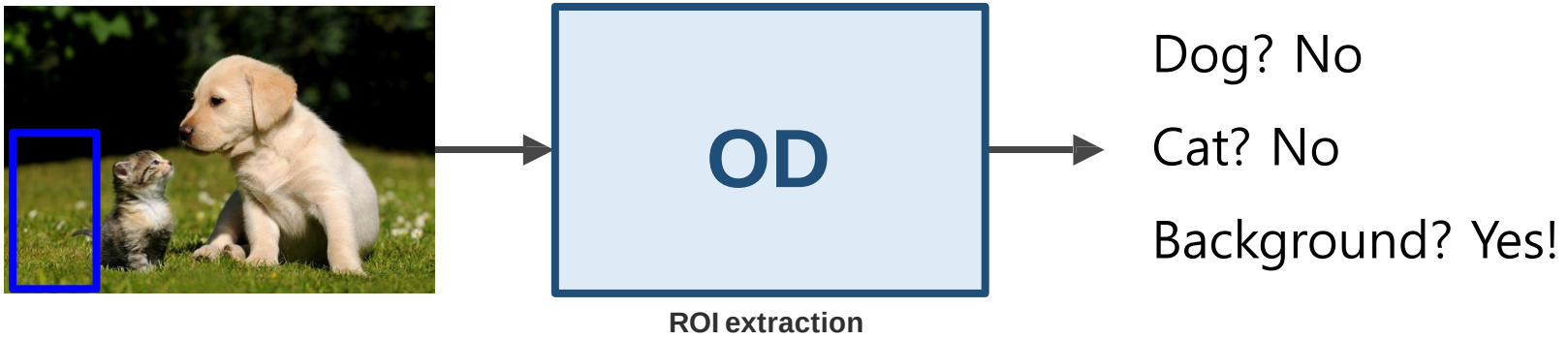

- Cat
  - Label: [1, 0]
  - Box: $[\frac{90}{360}, \frac{180}{240}, \frac{40}{360}, \frac{80}{240}]$




- Dog
  - Label: [0, 1]
  - Box: $[\frac{300}{360}, \frac{120}{240}, \frac{100}{360}, \frac{200}{240}]$

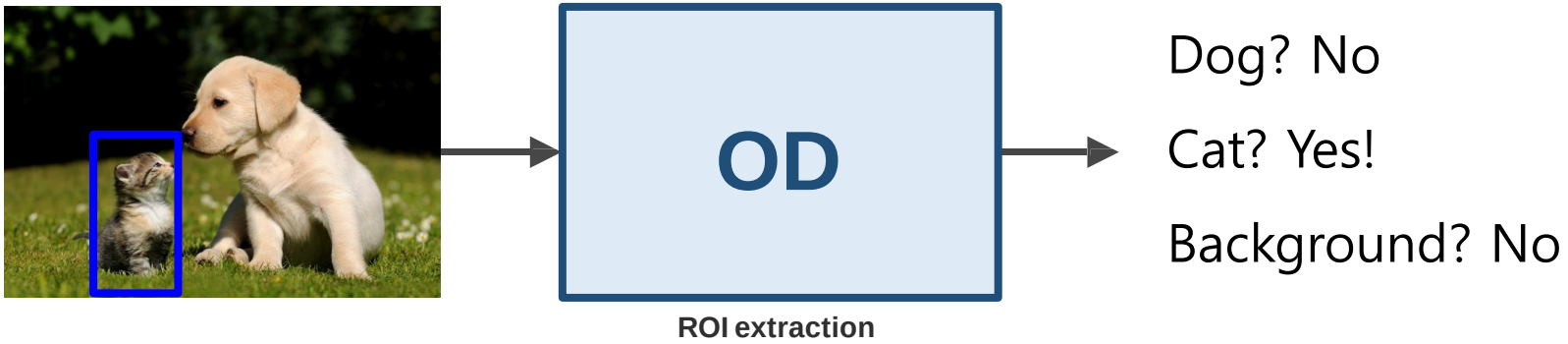
# Object Detection

- **Task**
  - Find Bounding Box: **Regression**
    - (X, Y, W, H)
  - Classify the Bounding Box: **Classification**
    - Dog or Cat
    - Apply CNN to image



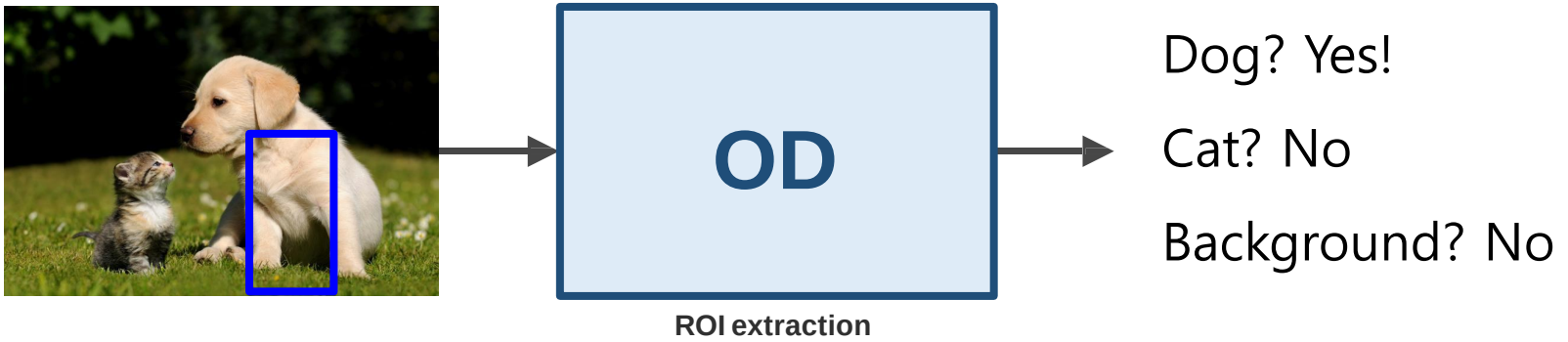
# Object Detection

- **Task**
  - Find Bounding Box: **Regression**
    - (X, Y, W, H)
  - Classify the Bounding Box: **Classification**
    - Dog or Cat
    - Apply CNN to image



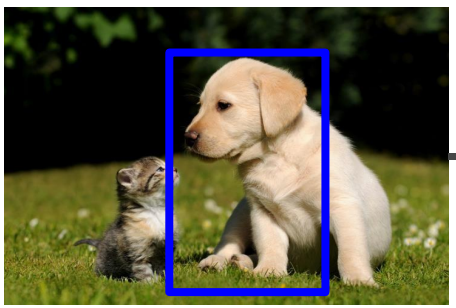
# Object Detection

- **Task**
  - Find Bounding Box: **Regression**
    - (X, Y, W, H)
  - Classify the Bounding Box: **Classification**
    - Dog or Cat
    - Apply CNN to image



# Object Detection

- Sliding window + Box regression + Classification

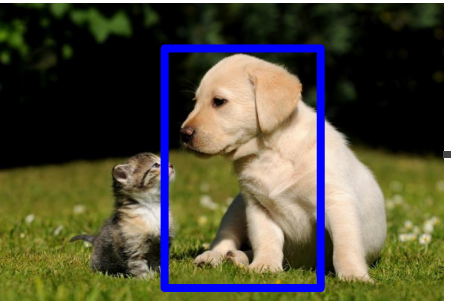


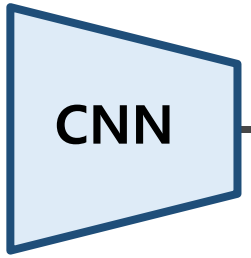

Dog? Yes!

Cat? No

Background? No

ROI extraction






Feature map
(5x5x1024)

# Traditional Object Detection

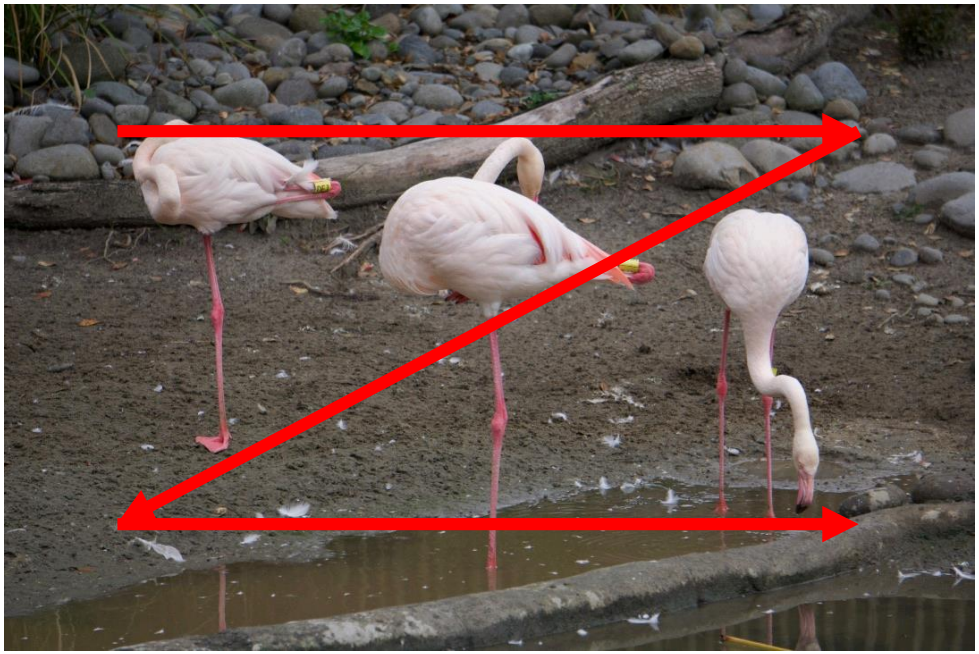
- **1. Templete matching + Sliding window**

Image

Template

# Traditional Object Detection

- **1. Templete matching + Sliding window**

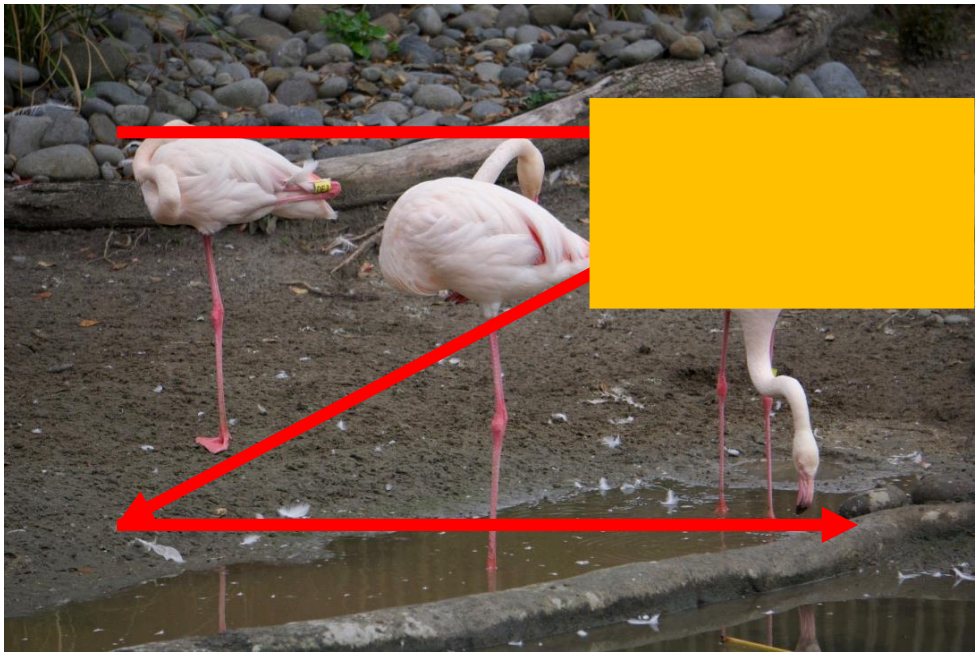


Image

Template

# Traditional Object Detection

- **1. Templete matching + Sliding window**
  - Occlusion: need whole object
  - Detect given instance. Not a object class



Image

Template

# Traditional Object Detection

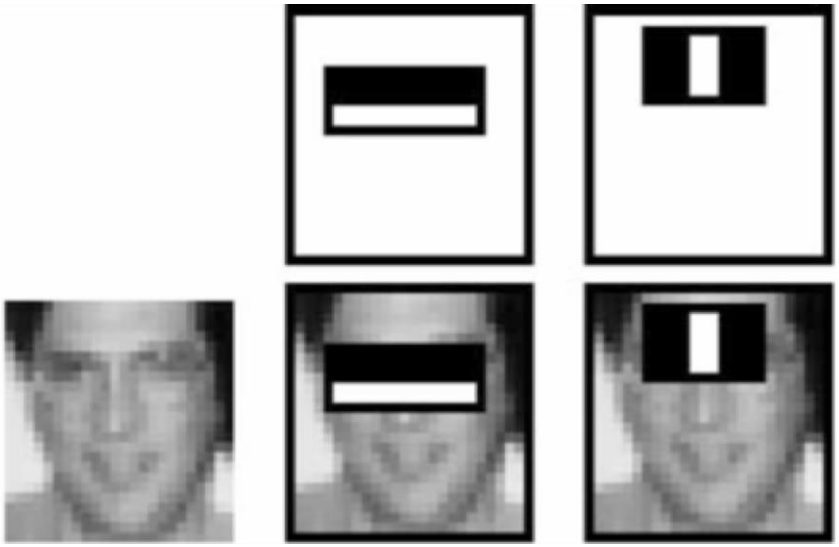
- **1. Templete matching + Sliding window**
  - Occlusion: need whole object
  - Detect given instance. Not a object class

Appearance
& shape changes

Pose changes

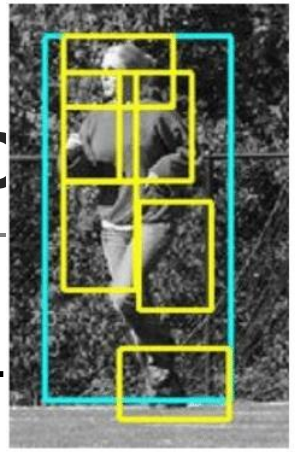
# Traditional Object Detection

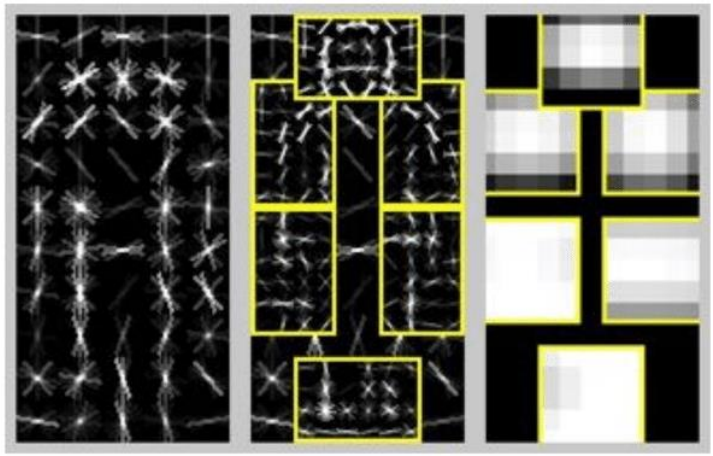
- **2. Feature extraction + classification**
  - Learning multiple weak learners to build a strong classifier
  - Make many small decisions and combine them for a stronger final decision



Haar features

# Traditional Object D

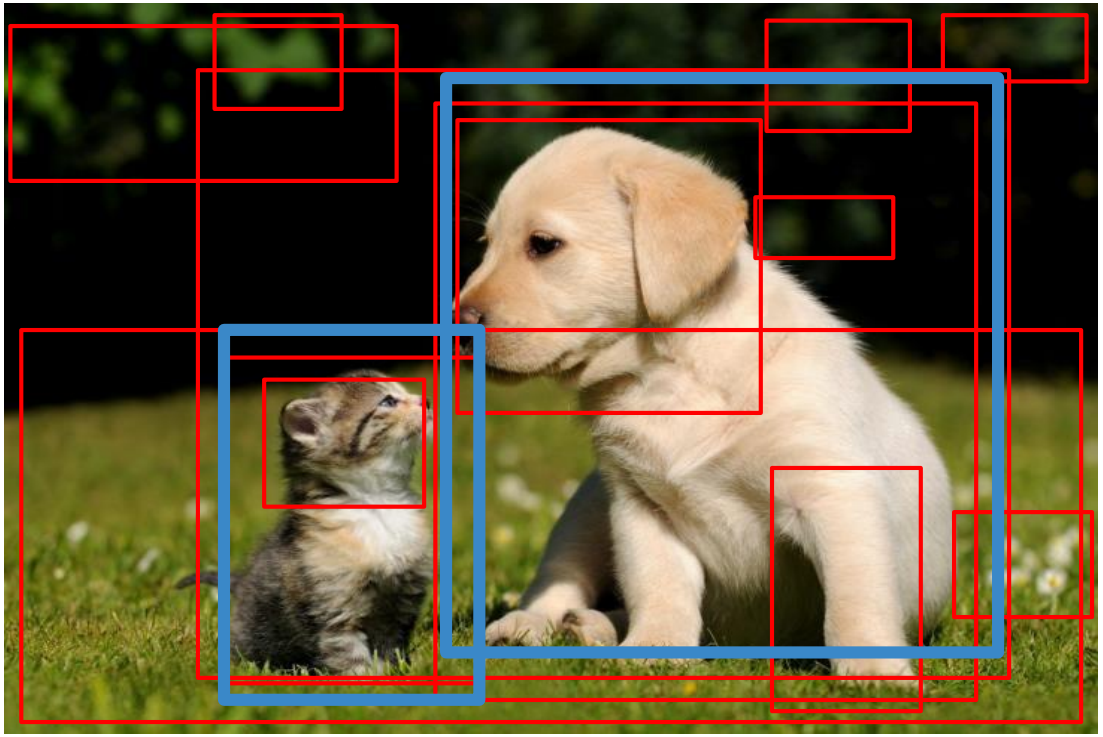


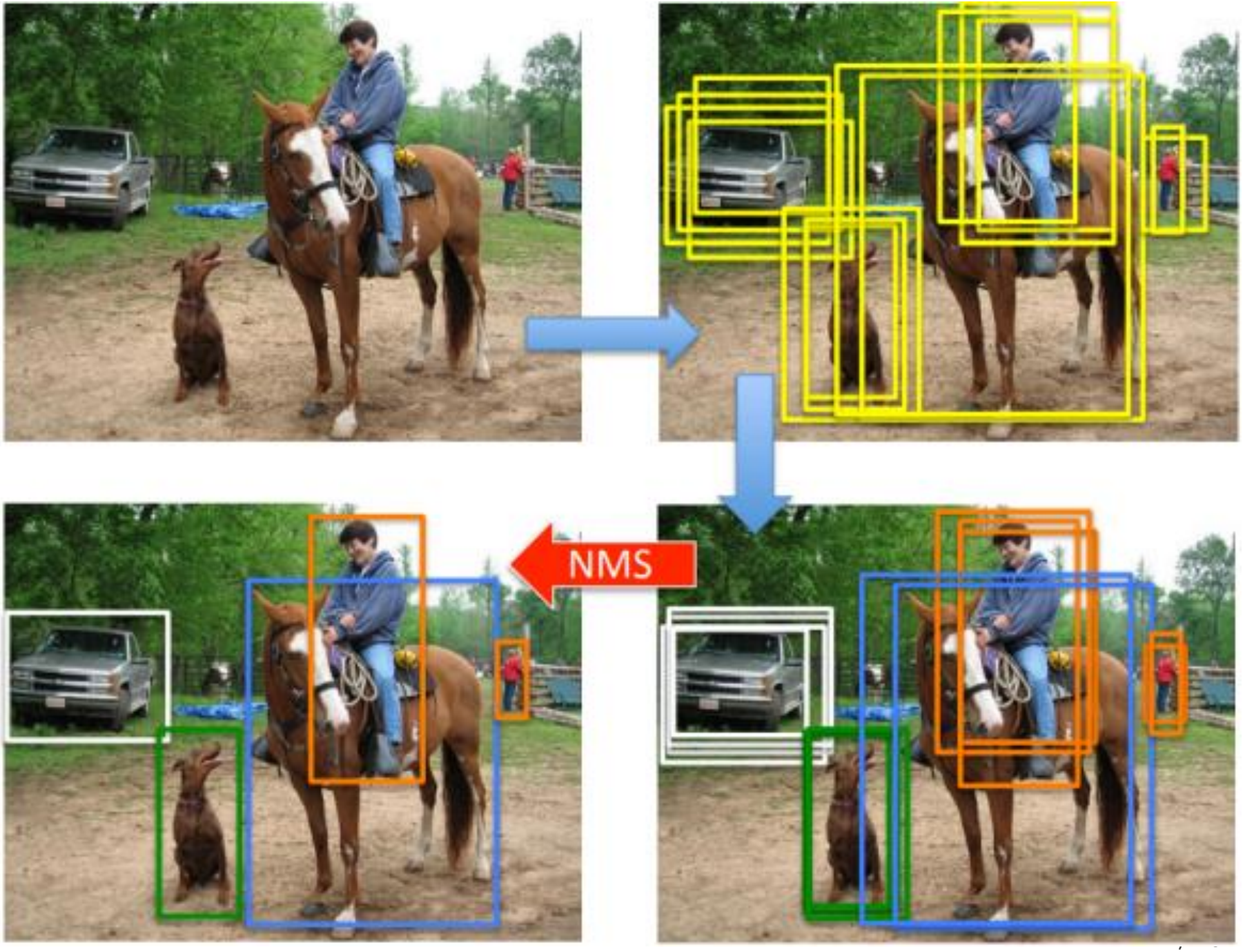


- **2. Feature extraction +**
  - Step 1: Select your Haar-like features
  - Step 2: Integral image for fast feature evaluation
    - I can evaluate which parts of the image have highest cross-correlation with my feature (template)
  - Step 3: AdaBoost for to find weak learner
    - I cannot possibly evaluate all features at test time for all image locations
    - Learn the best set of weak learners
    - Our final classifier is the linear combination of all weak learners

# Object Proposal

- **NMS (Non-Maximum Suppression)**
  - We need a generic, class-agnostic objectness measure
  - Many candidate object proposals (ROI, Region of Interest)

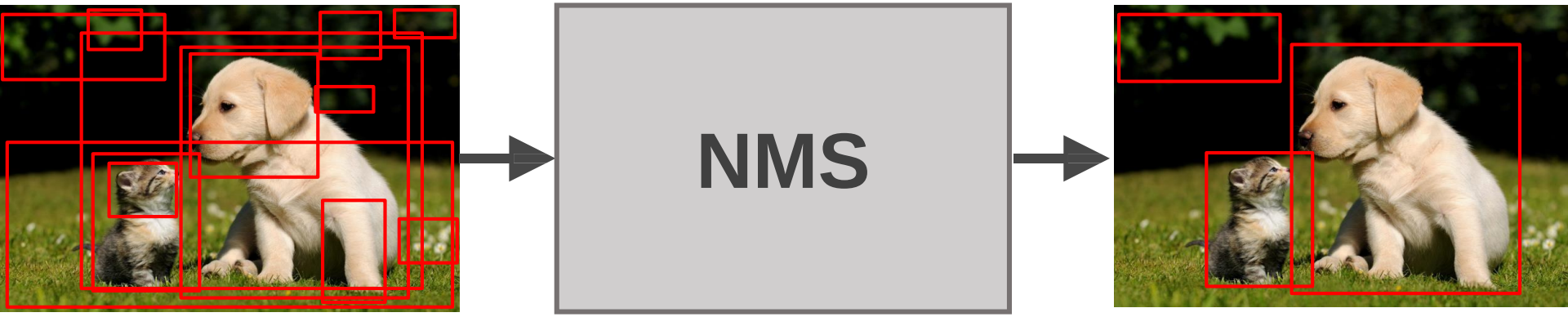





https://medium.com/@pedroazevedo6/object-detection-state-of-the-art-2022-ad750e0f6003

# Object Proposal

- **NMS (Non-Maximum Suppression)**
  - We need a generic, class-agnostic objectness measure
  - Many candidate object proposals (ROI, Region of Interest)
  - Many boxes trying to explain one object
  - We need a method to keep only the **"best" boxes**



# Object Proposal

- **NMS**


**Algorithm 1** Non-Max Suppression

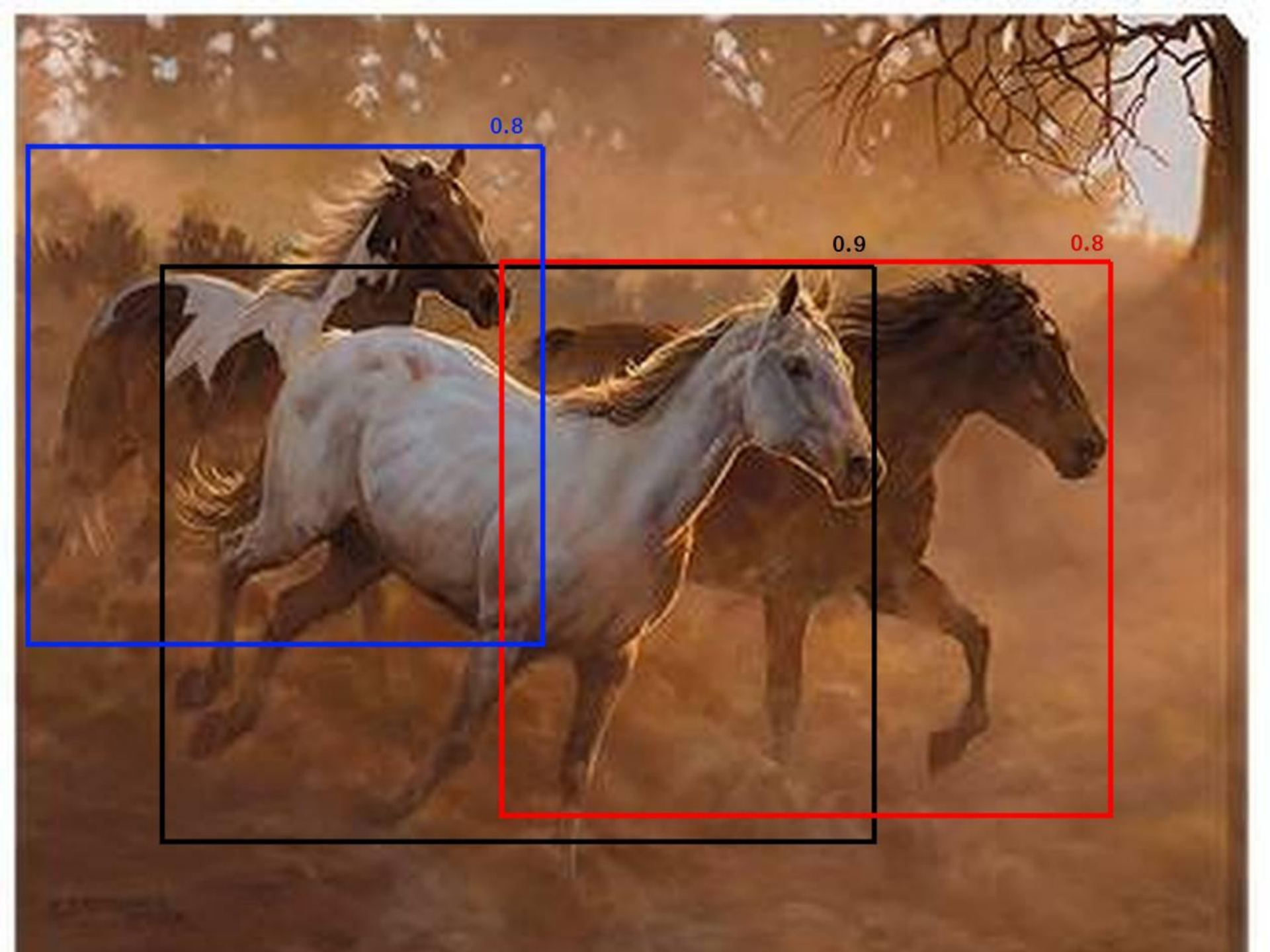
```
procedure NMS(B, c)
    B_nms ← ∅                                   Initialize empty set
    for b_i ∈ B do                              => Iterate over all the boxes
        discard ← False                         Take boolean variable and set it as false. This variable indicates whether b(i) should be kept or discarded
        for b_j ∈ B do                          Start another loop to compare with b(i)
            if same(b_i, b_j) > λ_nms then      If both boxes having same IOU
                if score(c, b_j) > score(c, b_i) then
                    discard ← True              Compare the scores. If score of b(i) is less than that of b(j), b(i) should be discarded, so set the flag to True.
        if not discard then                     Once b(i) is compared with all other boxes and still the discarded flag is False, then b(i) should be considered. So add it to the final list.
            B_nms ← B_nms ∪ b_i
    return B_nms                                Do the same procedure for remaining boxes and return the final list
```

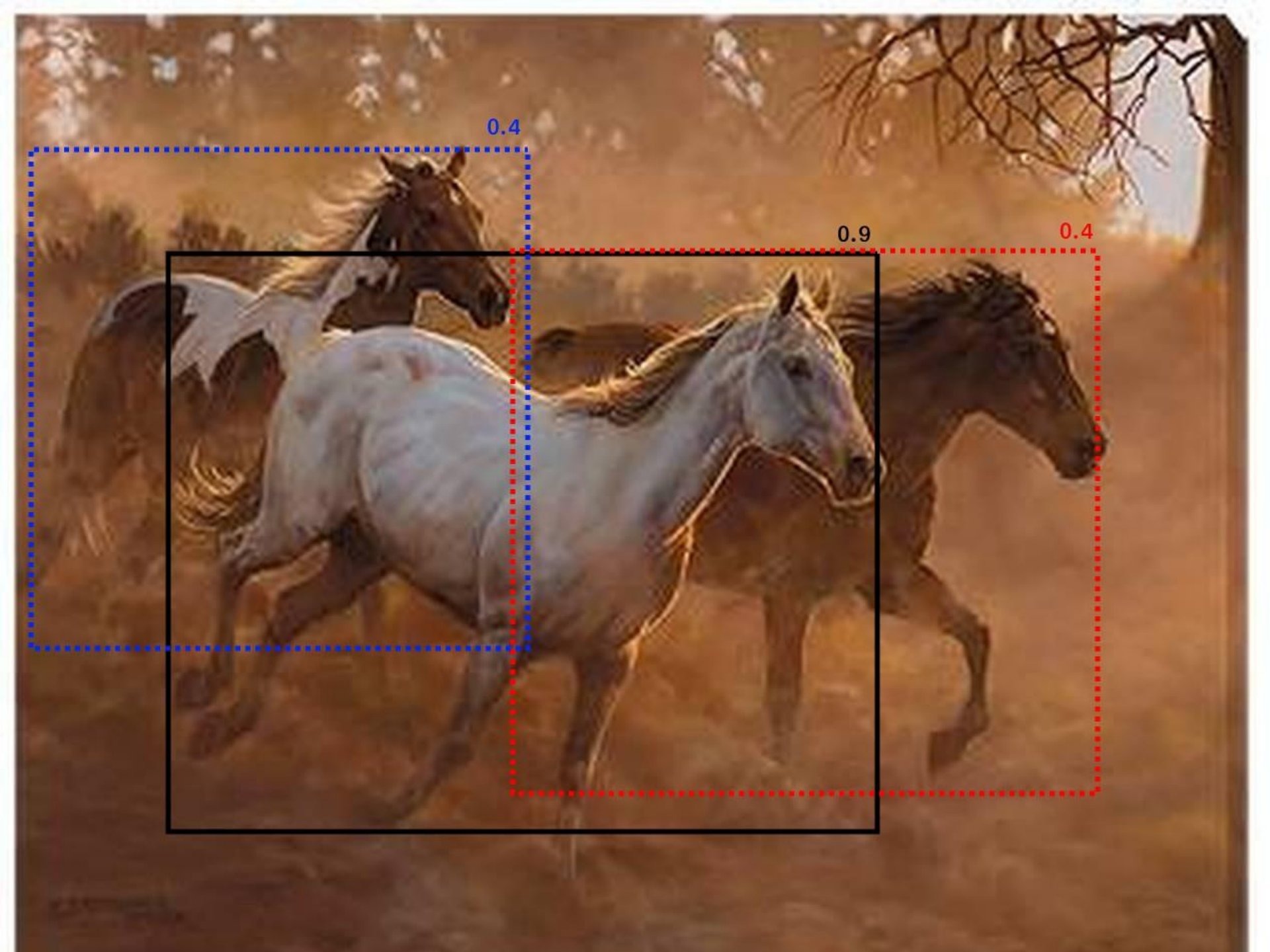
Start with anchor box i

For another box j

If they overlap

Discard box i if the score is lower than the score of j


0.8
0.9
0.8


0.4
0.9
0.4

# Region Overlap

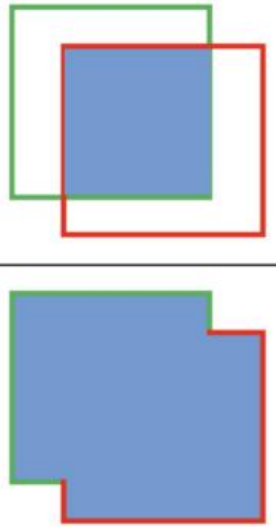
- IoU (Intersection Over Union)
- Precision
- Recall
- Average Precision
- FPS

$B_p$ = 실제 (*Gound Truth*)

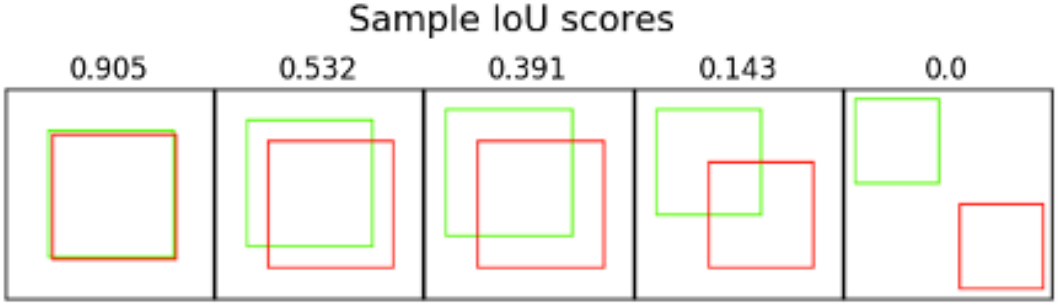
$B_{gt}$ = 예측 (*Prediction*)

$$IOU = \frac{area\ of\ overlap}{area\ of\ union} = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})}$$

$$= \frac{\text{실제} \cap \text{예측 중복 영역}}{\text{실제} \cup \text{예측 전체 영역}}$$



waytoliah.com






https://hoya012.github.io/blog/Tutorials-of-Object-Detection-Using-Deep-Learning-how-to-measure-performance-of-object-detection/

# Object Proposal

- **NMS**


**Algorithm 1** Non-Max Suppression

```
procedure NMS(B, c)
    B_nms ← ∅                                   Initialize empty set
    for b_i ∈ B do                              => Iterate over all the boxes
        discard ← False                         Take boolean variable and set it as false. This variable indicates whether b(i) should be kept or discarded
        for b_j ∈ B do                          Start another loop to compare with b(i)
            if same(b_i, b_j) > λ_nms then      If both boxes having same IOU
                if score(c, b_j) > score(c, b_i) then
                    discard ← True              Compare the scores. If score of b(i) is less than that of b(j), b(i) should be discarded, so set the flag to True.
        if not discard then                     Once b(i) is compared with all other boxes and still the discarded flag is False, then b(i) should be considered. So add it to the final list.
            B_nms ← B_nms ∪ b_i
    return B_nms                                Do the same procedure for remaining boxes and return the final list
```

Start with anchor box i

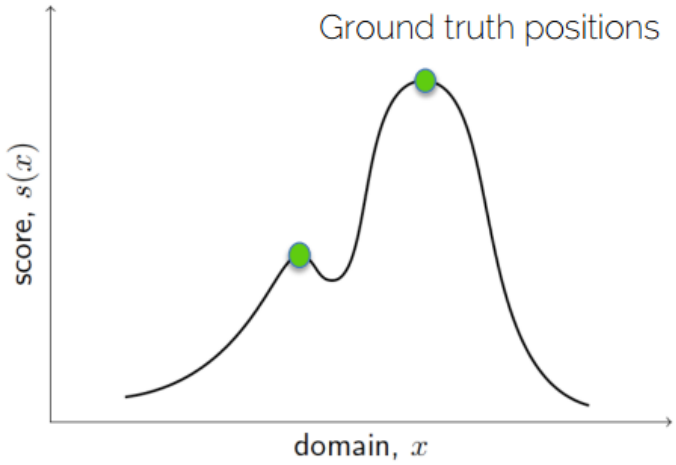
For another box j

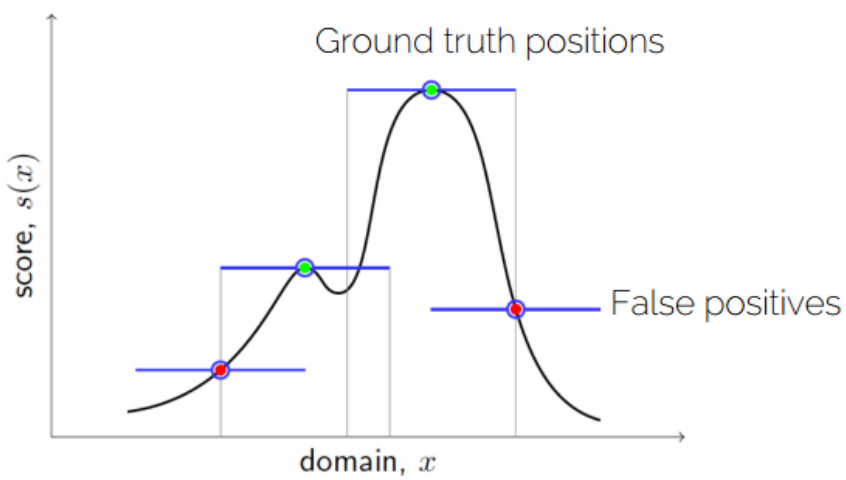
If they overlap

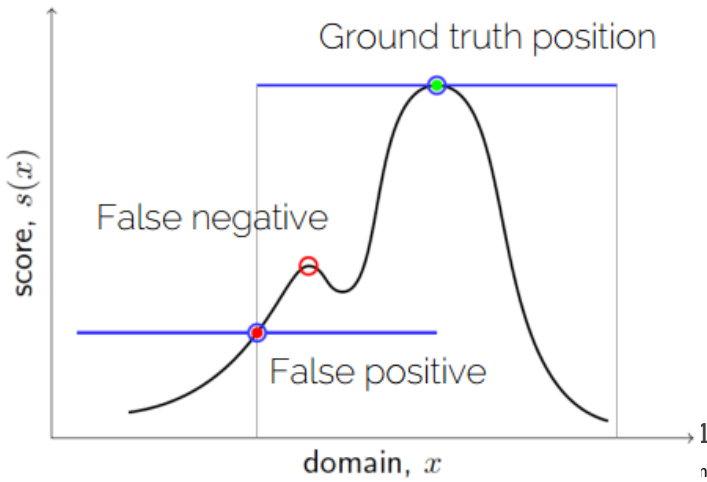
Discard box i if the score is lower than the score of j

# Object Proposal

- **NMS**

# OD performance

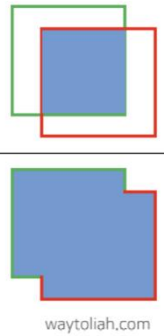
- Confidence: 검출한 것에 대해 얼마나 확신이 있는지
- IoU (Intersection Over Union)
- Precision
- Recall
- AP(Average Precision): Precision-Recall 그래프에서 아래 면적
- mAP(mean Average Precision): 각 클래스 당 AP의 평균
- mAP@0.5: mAP의 평균을 IoU>0.5로 구한 값
- mAP@0.5:0.95: 0.5~0.95 IoU threshold 값을 0.05
- 간격으로 측정한 mAP의 평균값
- FPS: 초당 몇 frame 연산을 할수 있는지

$B_p$ = 실제 (*Gound Truth*)

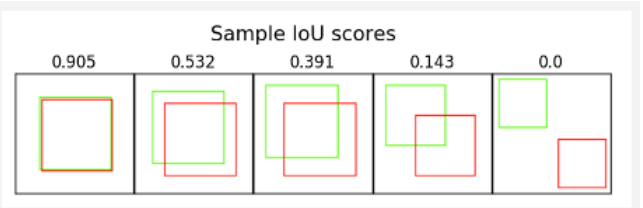
$B_{gt}$ = 예측 (*Prediction*)

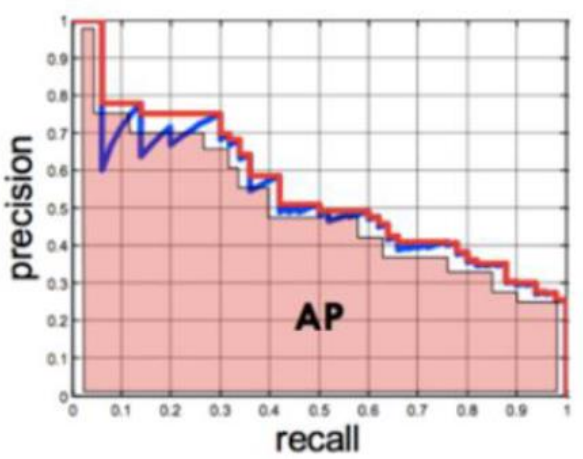
$$IOU = \frac{area\ of\ overlap}{area\ of\ union} = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})}$$

$$= \frac{\text{실제} \cap \text{예측 중복 영역}}{\text{실제} \cup \text{예측 전체 영역}}$$



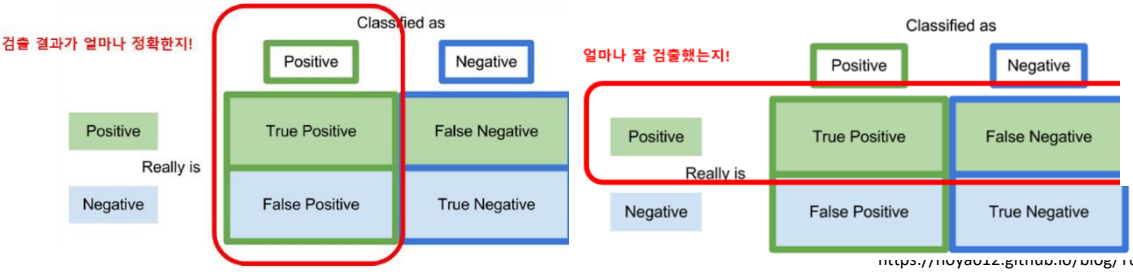
waytoliah.com




Precision = TP / (TP + FP)

Recall = TP / (TP + FN)

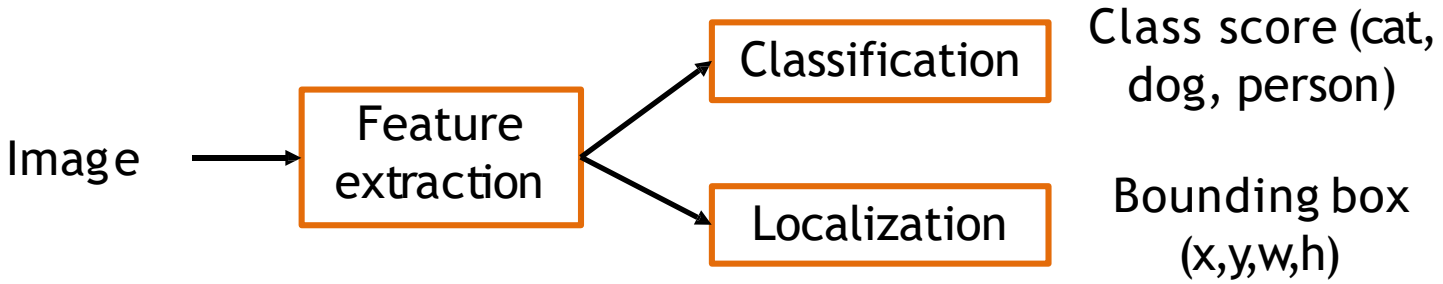




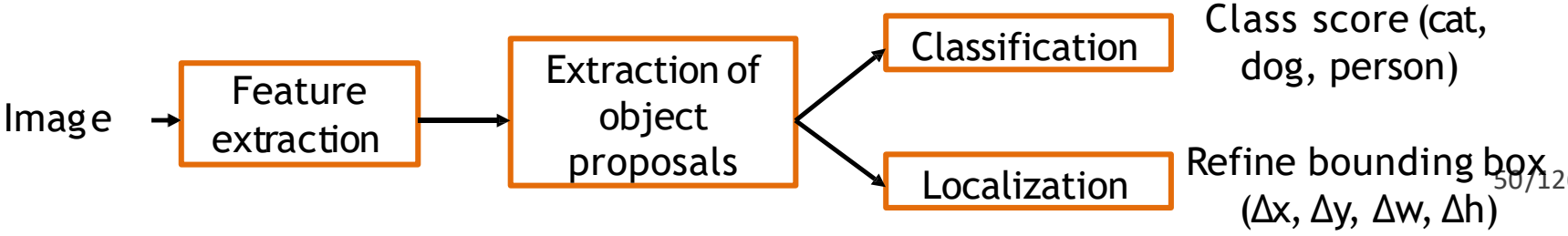

https://hoya012.github.io/blog/Tutorials-of-Object-Detection-Using-Deep-Learning-how-to-measure-performance-of-object-detection/

# Object Detection Type

- **1-stage detector**
  - 추론 속도가 빨라 real-time task
  - YOLO, SSD, RetinaNet, CenterNet, CornerNet, ExtremeNet




- **2-stage detector: Faster R-CNN**
  - 성능이 좋지만 느림
  - R-CNN, Fast R-CNN, Faster R-CNN, SPP-Net, R-FCN, FPN



# 2-stage detector

- **1) R-CNN**



Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

# 2-stage detector

- **1) R-CNN**

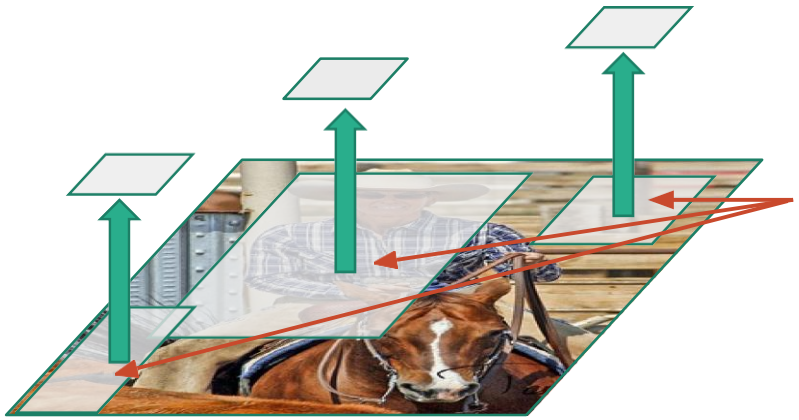
**RoI (Regions of Interest) ~2k**



Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

# 2-stage detector

- **1) R-CNN**



**Warped image region**

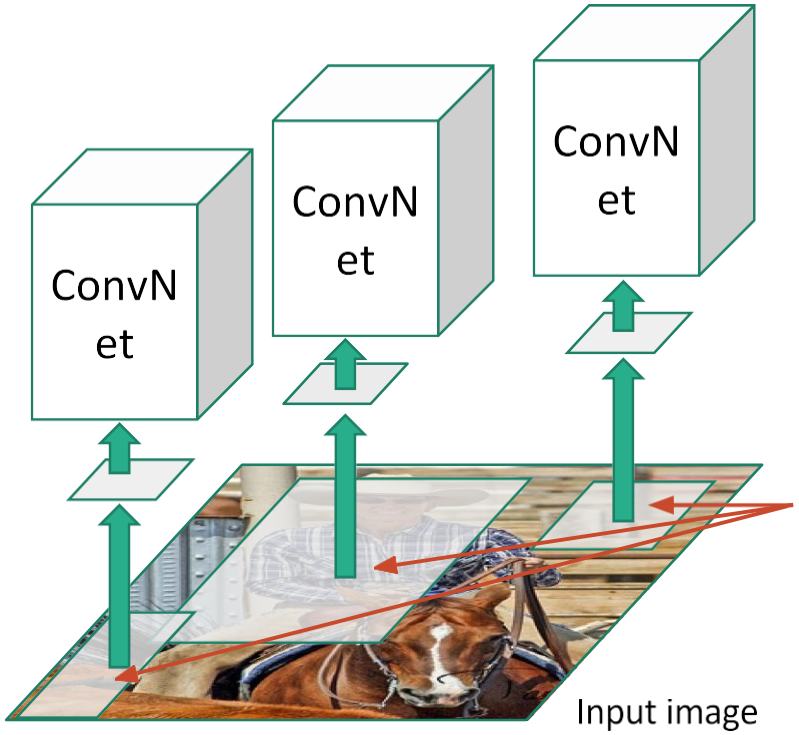
**RoI (Regions of Interest) ~2k**



Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

# 2-stage detector

- **1) R-CNN**




**Forward each region through ConvNet**

**Warped image region**

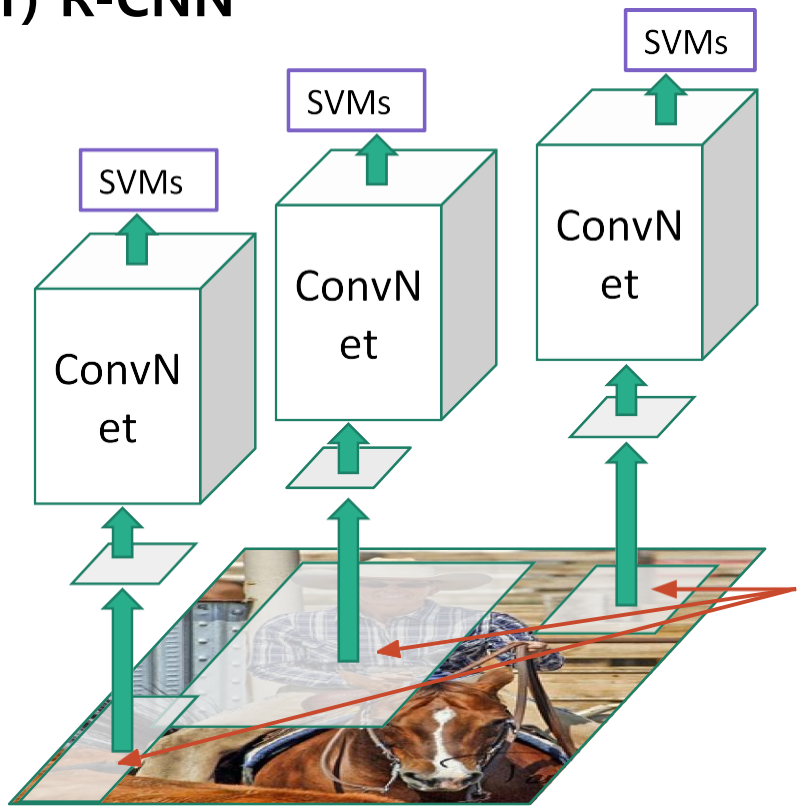
**RoI (Regions of Interest) ~2k**



Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

# 2-stage detector

- **1) R-CNN**




**Classify regions**

**Forward each region through ConvNet**

**Warped image region**

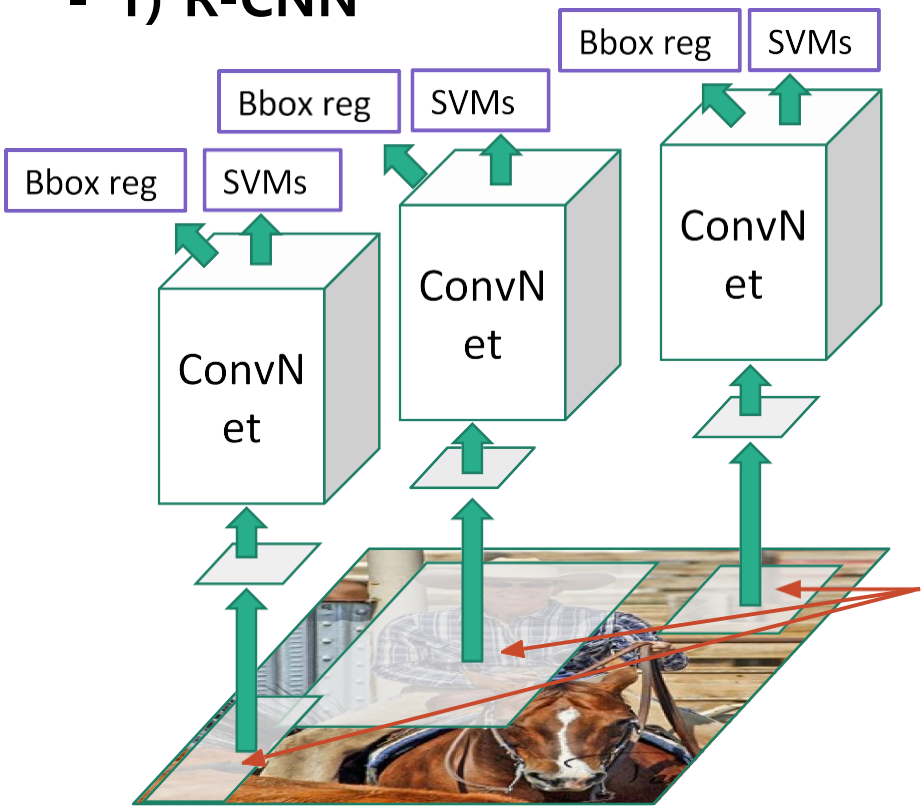
**RoI (Regions of Interest) ~2k**



Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

# 2-stage detector

- **1) R-CNN**




**Classify regions & Regression for Bbox**

**Forward each region through ConvNet**

**Warped image region**

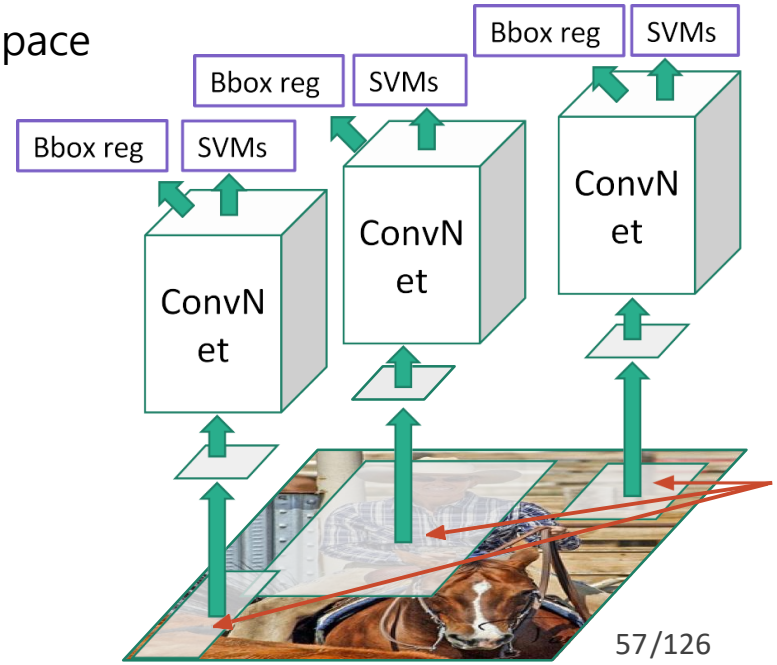
**RoI (Regions of Interest) ~2k**



Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

# 2-stage detector

- **1) R-CNN**
  - Training objectives
    - Fine-tune network with softmax classifier (log loss)
    - Train post-hoc linear SVMs (hinge loss)
    - Train post-hoc bounding-box regressions (least squares)
  - Training is slow (84h), takes a lot of disk space
  - Inference (detection) is **slow**
    - 47s / image with VGG16

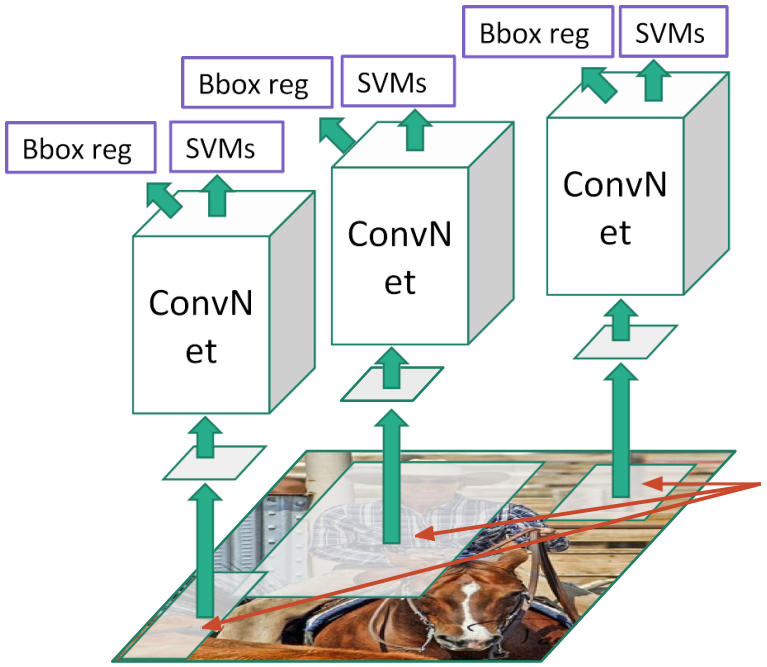




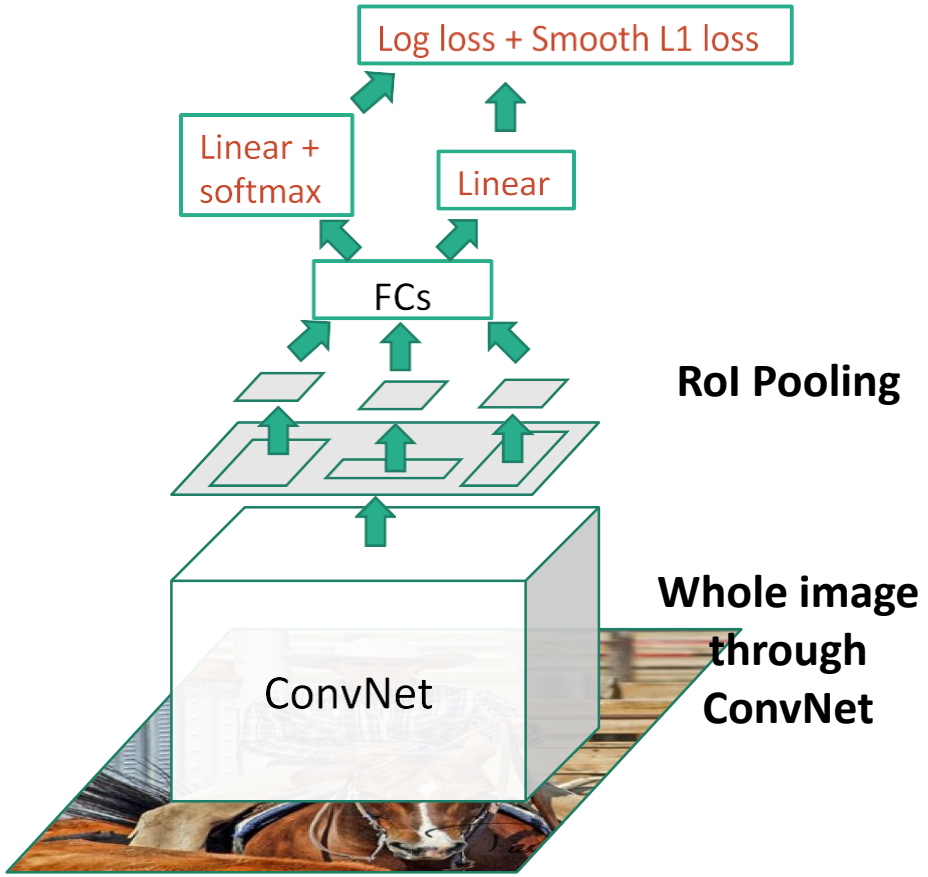

Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

# 2-stage detector

- 2) Faster R-CNN




R-CNN



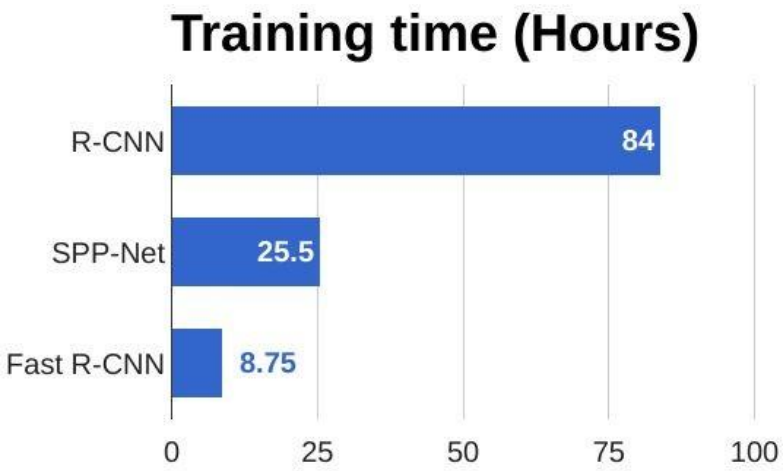

Faster R-CNN

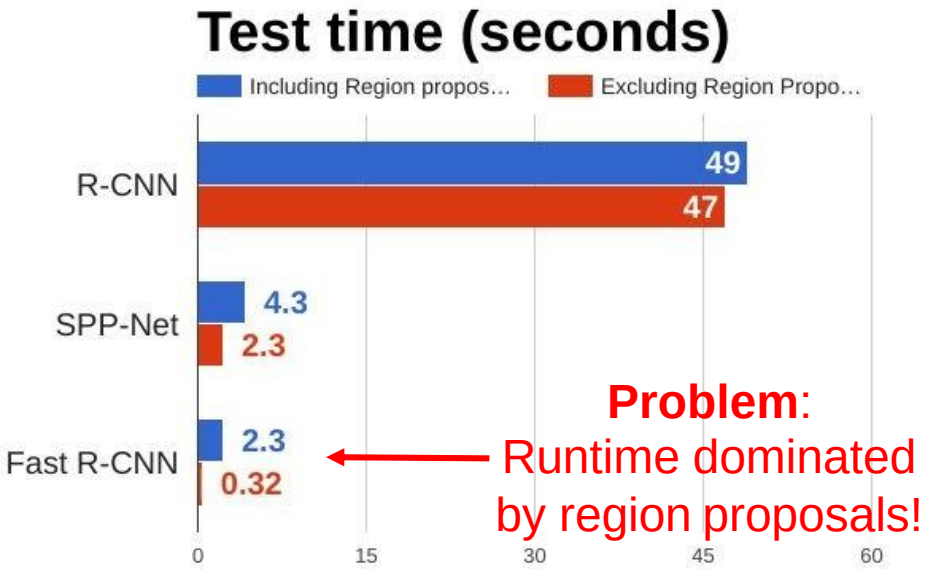


Girshick, "Fast R-CNN", ICCV 2015.

# 2-stage detector

- **2) Faster R-CNN**



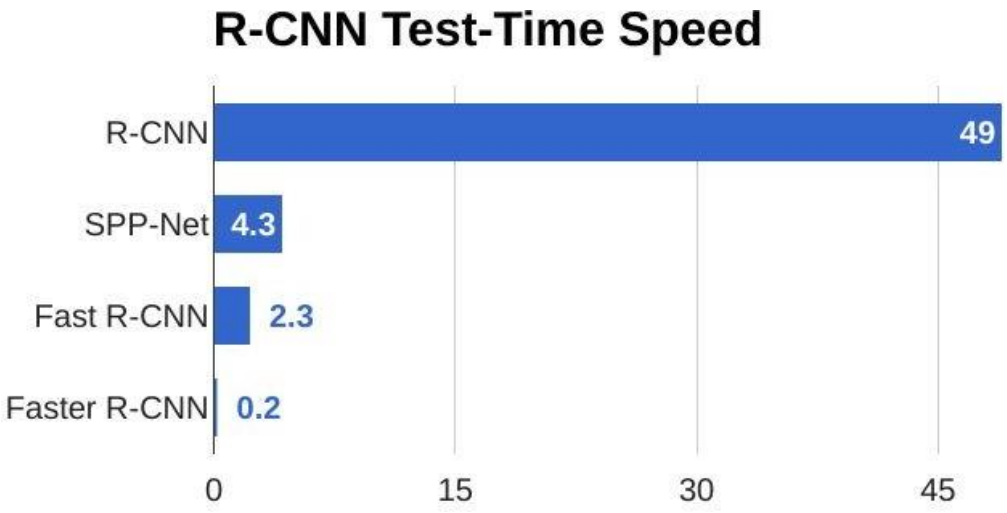



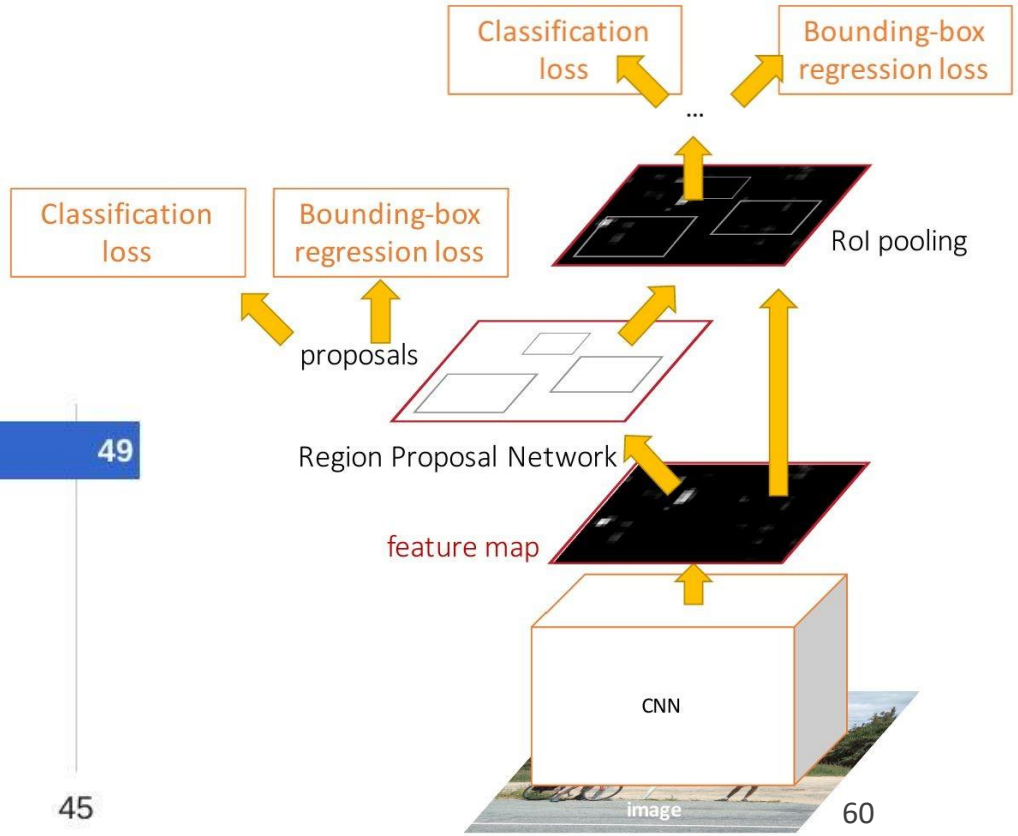



Girshick, "Fast R-CNN", ICCV 2015.

# 2-stage detector

- **3) Faster R-CNN**
  - Make CNN do proposals!
  - Insert Region Proposal Network (RPN) to predict proposals from features
    - RPN classification (object/non-object)
    - RPN regression (anchor -> proposal)
    - Fast R-CNN classification (type of object)
    - Fast R-CNN regression (proposal -> box)



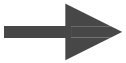



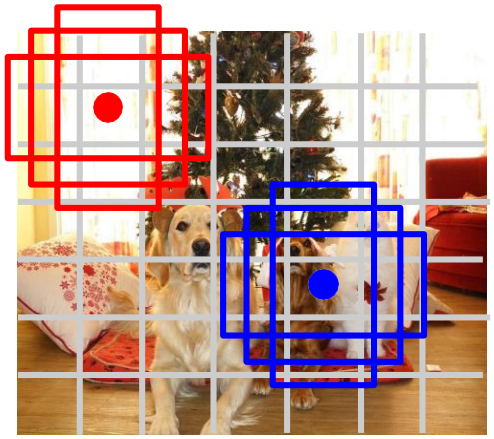

Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015

# 1-stage detector

- **1) YOLO, SSD in reach grid cell**
  - Regress from each 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: 7x7x(5xB+C)

Input image 3 x H x W





Divide image into grid 7x7
base boxes centered at
each grid cell, B=3



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 and SSD

- **PROS:**
  - **Very fast**
  - **End-to-end trainable and fully convolutional**
  - **SSD detects more objects than YOLO**
- **CONS:**
  - **Performance is not as good as two-stage detectors**
  - **Difficulty with small objects**



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

# OD variables

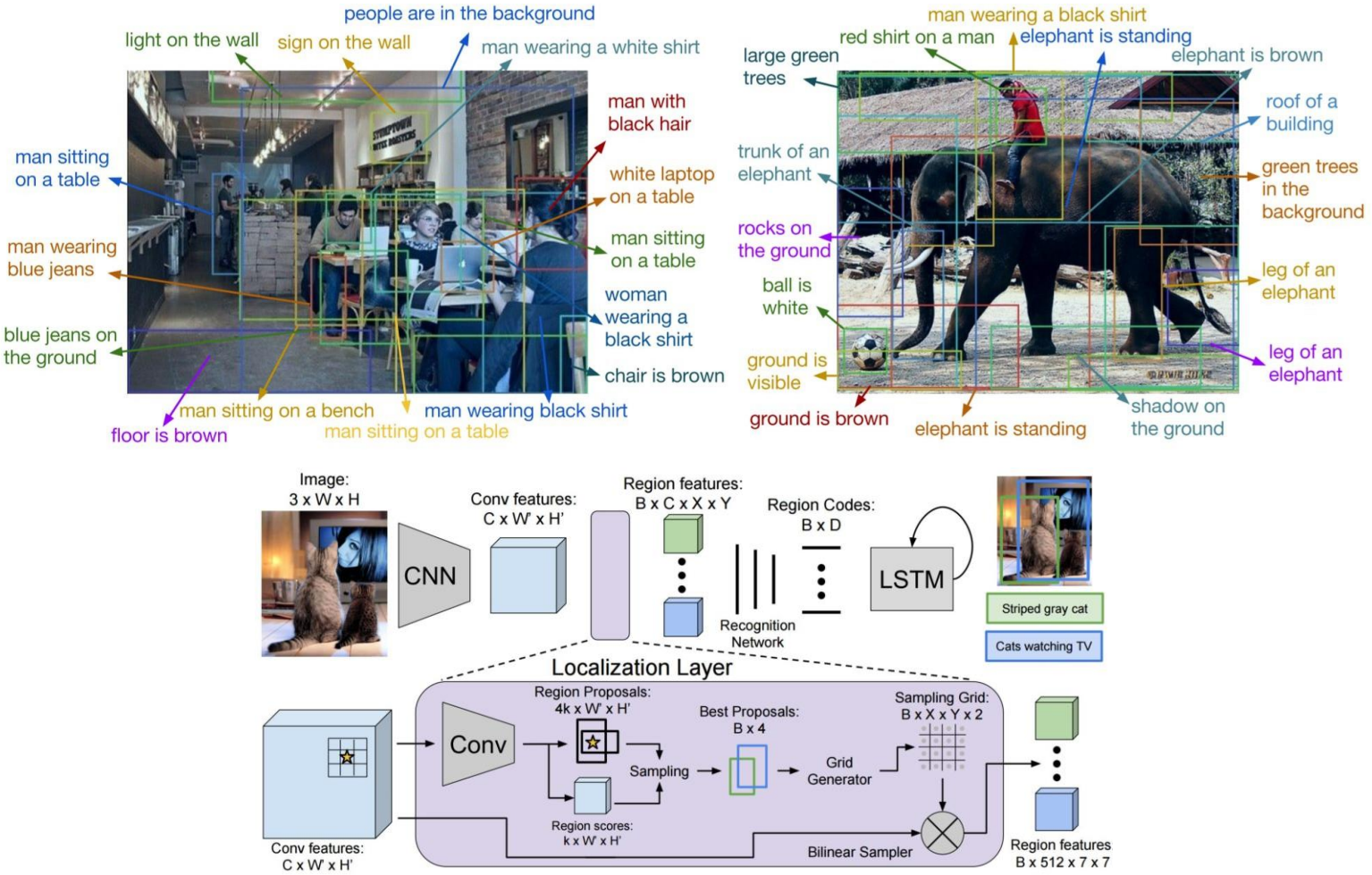
- **Base Network**
  - VGG, ResNet, Inception v1/2/3, MobileNet
- **Architecture**
  - YOLO, SSD, FaSTER R-CNN
- **Size**
  - Input image size, Region proposal
- **Takeaways**
  - Faster R-CNN is slower but more accurate
  - YOLO/SSD is faster but not as accurate



Girshick, "Fast R-CNN", ICCV 2015.

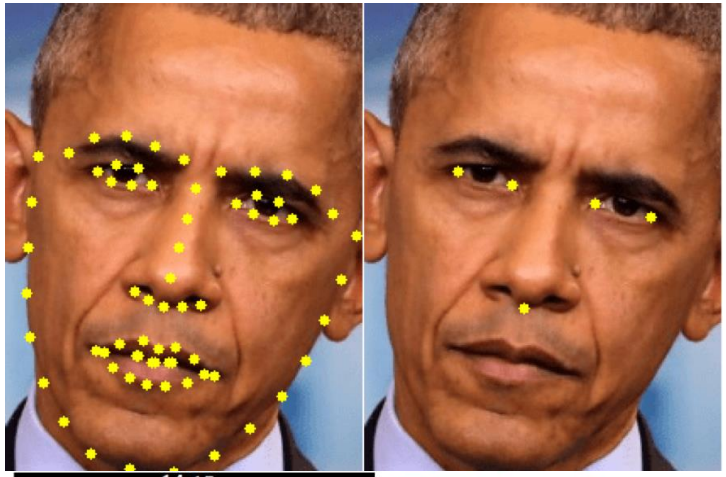
# OD Application

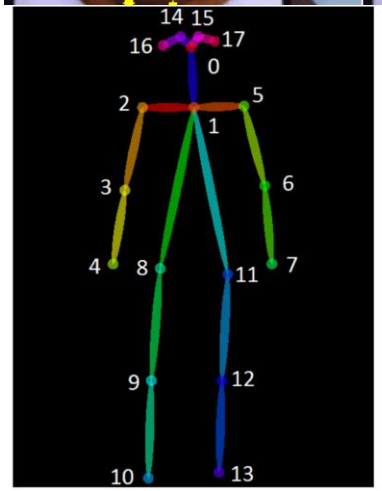
- **Object Detection + Captioning**

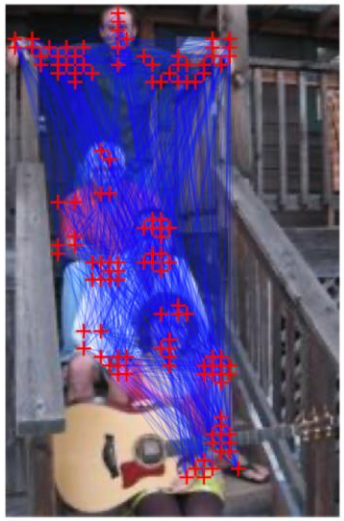


# OD Application

- **Detection beyond 2D boxes**

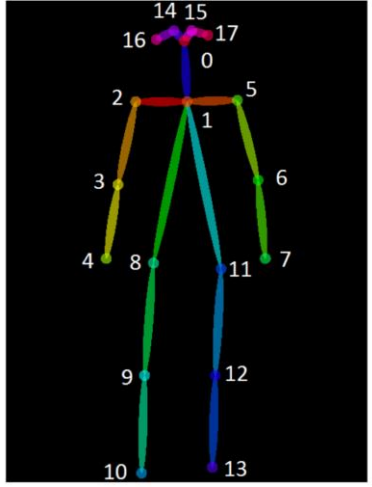




# OD Application

- **Pose Estimation data**
  - **COCO dataset**
    - 12만(19GB)
    - 17 key points
    - 여러 사람에 대한 데이터셋
  - **MPII dataset**
    - 2.5만
    - 16 key points
    - 한명에 대한 데이터셋

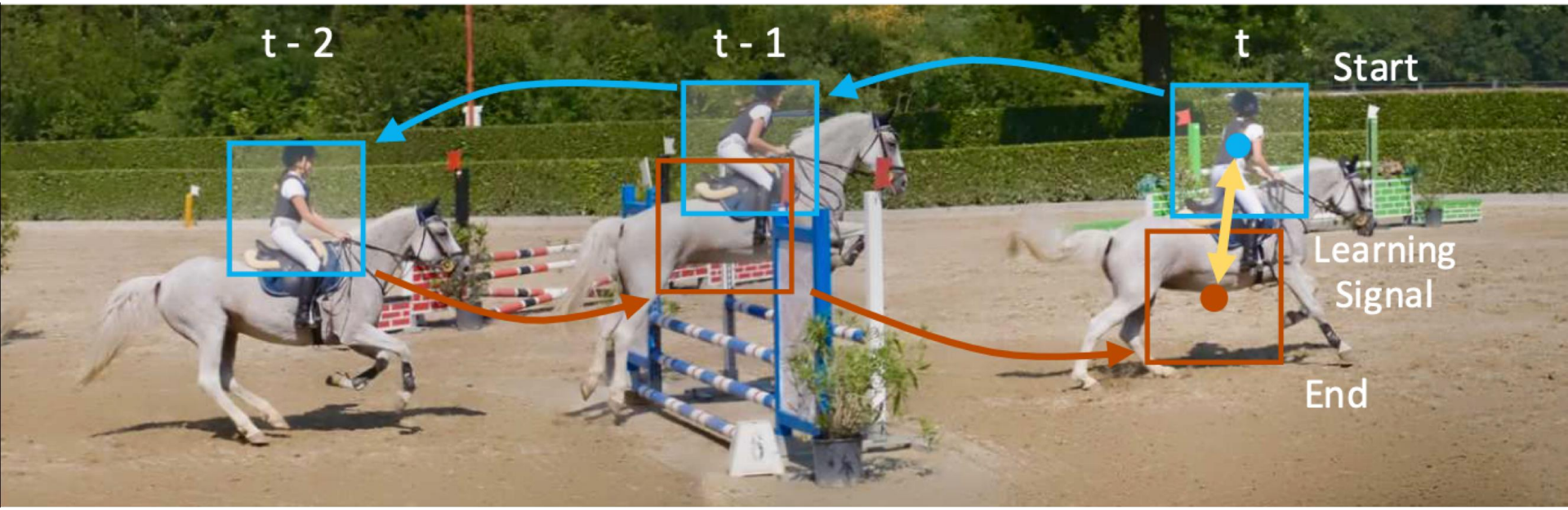



COCO KeyPoints

MPII KeyPoints

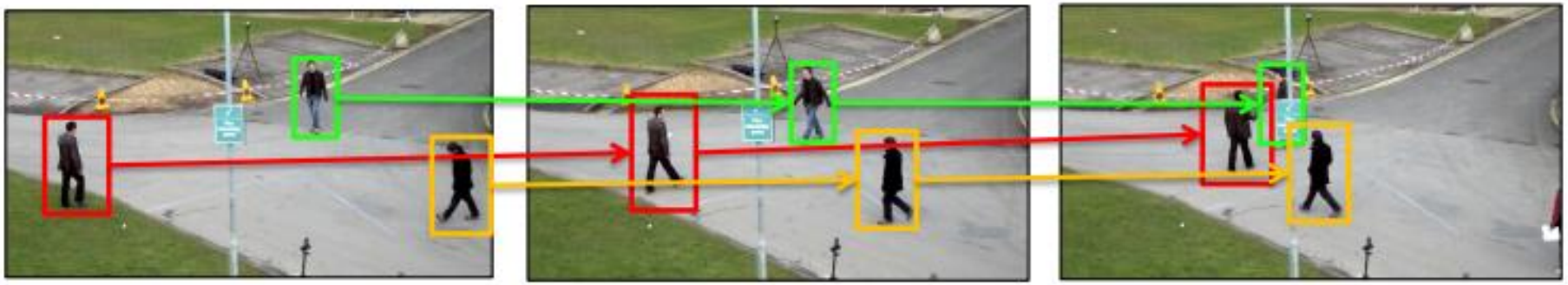
# OD Application

- **Object Tracking(Single)**



# OD Application

- **Object Tracking(Multi)**
  - **Graph based method**



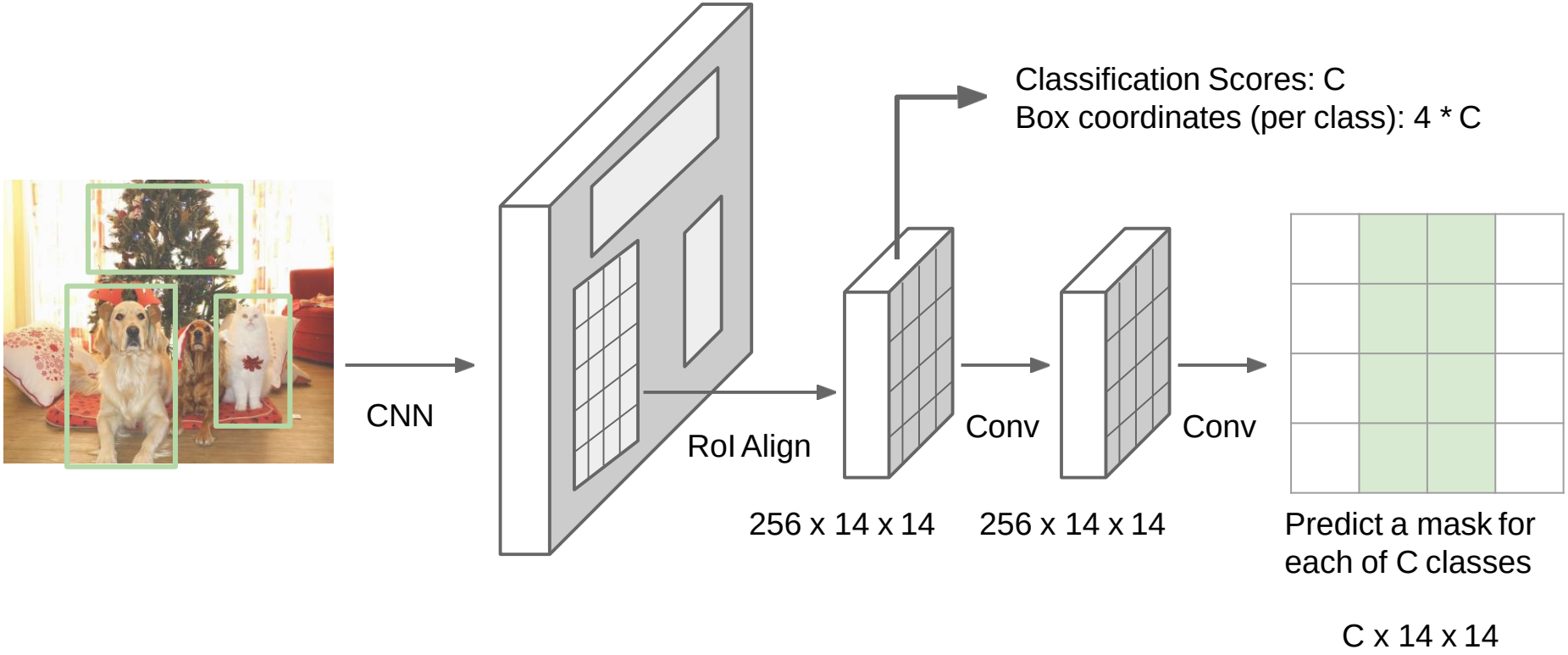
# OD Application

- **Object Tracking(Multi)**
  - **DeepSort**

**DeepSort Demo**

# OD Application

- **Segmentation**






He et al, “Mask R-CNN”, arXiv 2017

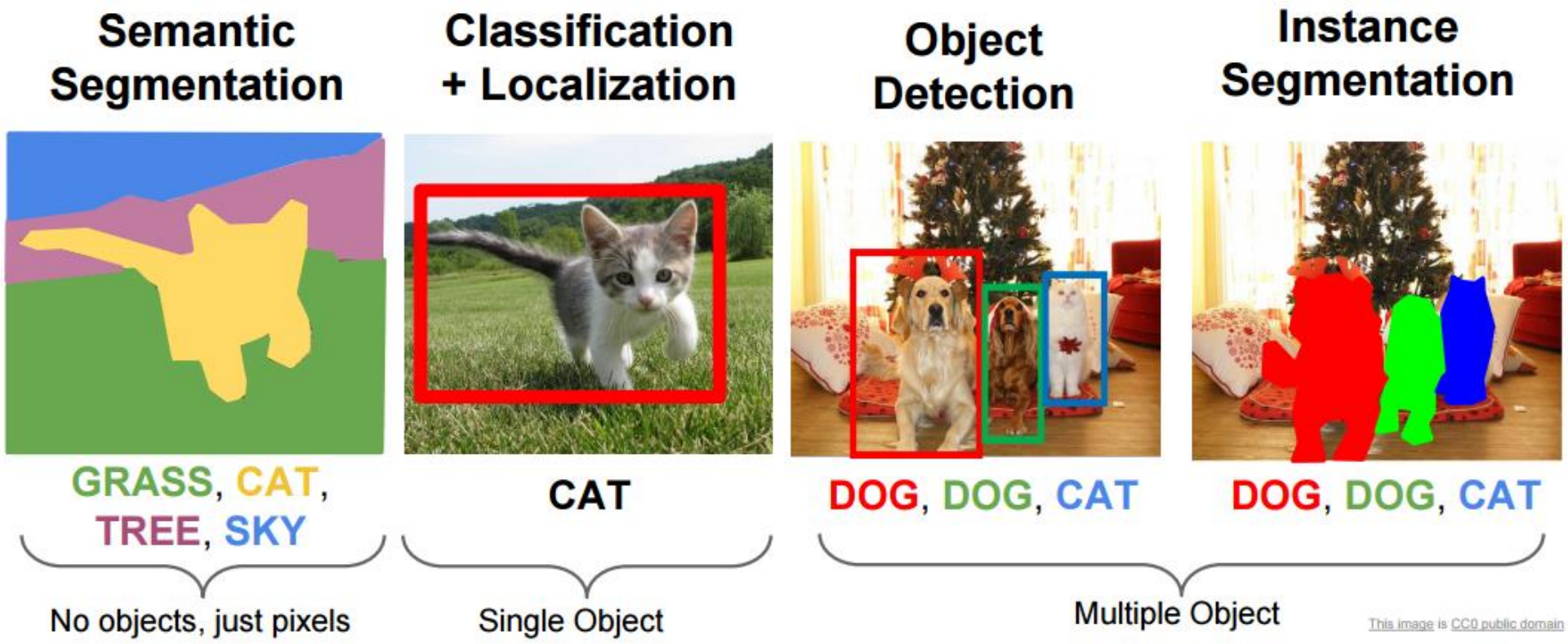
# OD Application

- **Segmentation**



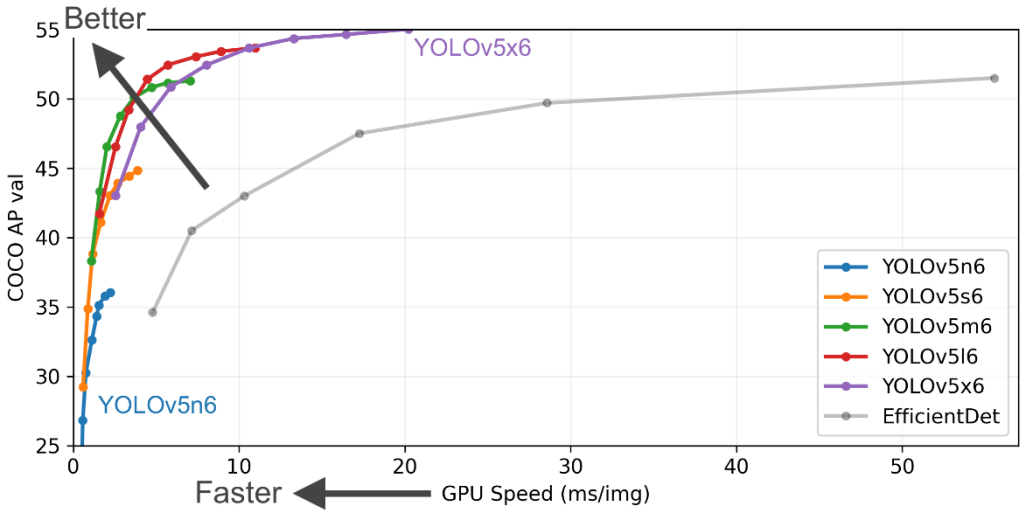

He et al, “Mask R-CNN”, arXiv 2017

# Recap


Semantic Segmentation
Classification + Localization
Object Detection
Instance Segmentation
GRASS, CAT, TREE, SKY
CAT
DOG, DOG, CAT
DOG, DOG, CAT
No objects, just pixels
Single Object
Multiple Object
This image is CC0 public domain

# OD Practice

- **Yolov5[URL]**
  - https://github.com/ultralytics/yolov5

| Model | size (pixels) | mAP$^{val}$ 0.5:0.95 | mAP$^{val}$ 0.5 | Speed CPU b1 (ms) | Speed V100 b1 (ms) | Speed V100 b32 (ms) | params (M) | FLOPs @640 (B) |
|---|---|---|---|---|---|---|---|---|
| YOLOv5n | 640 | 28.0 | 45.7 | **45** | **6.3** | **0.6** | **1.9** | **4.5** |
| YOLOv5s | 640 | 37.4 | 56.8 | 98 | 6.4 | 0.9 | 7.2 | 16.5 |
| YOLOv5m | 640 | 45.4 | 64.1 | 224 | 8.2 | 1.7 | 21.2 | 49.0 |
| YOLOv5l | 640 | 49.0 | 67.3 | 430 | 10.1 | 2.7 | 46.5 | 109.1 |
| YOLOv5x | 640 | 50.7 | 68.9 | 766 | 12.1 | 4.8 | 86.7 | 205.7 |
| | | | | | | | | |
| YOLOv5n6 | 1280 | 36.0 | 54.4 | 153 | 8.1 | 2.1 | 3.2 | 4.6 |
| YOLOv5s6 | 1280 | 44.8 | 63.7 | 385 | 8.2 | 3.6 | 12.6 | 16.8 |
| YOLOv5m6 | 1280 | 51.3 | 69.3 | 887 | 11.1 | 6.8 | 35.7 | 50.0 |
| YOLOv5l6 | 1280 | 53.7 | 71.3 | 1784 | 15.8 | 10.5 | 76.8 | 111.4 |
| YOLOv5x6<br>+ TTA | 1280<br>1536 | 55.0<br>**55.8** | 72.7<br>**72.7** | 3136<br>- | 26.2<br>- | 19.4<br>- | 140.7<br>- | 209.8<br>- |

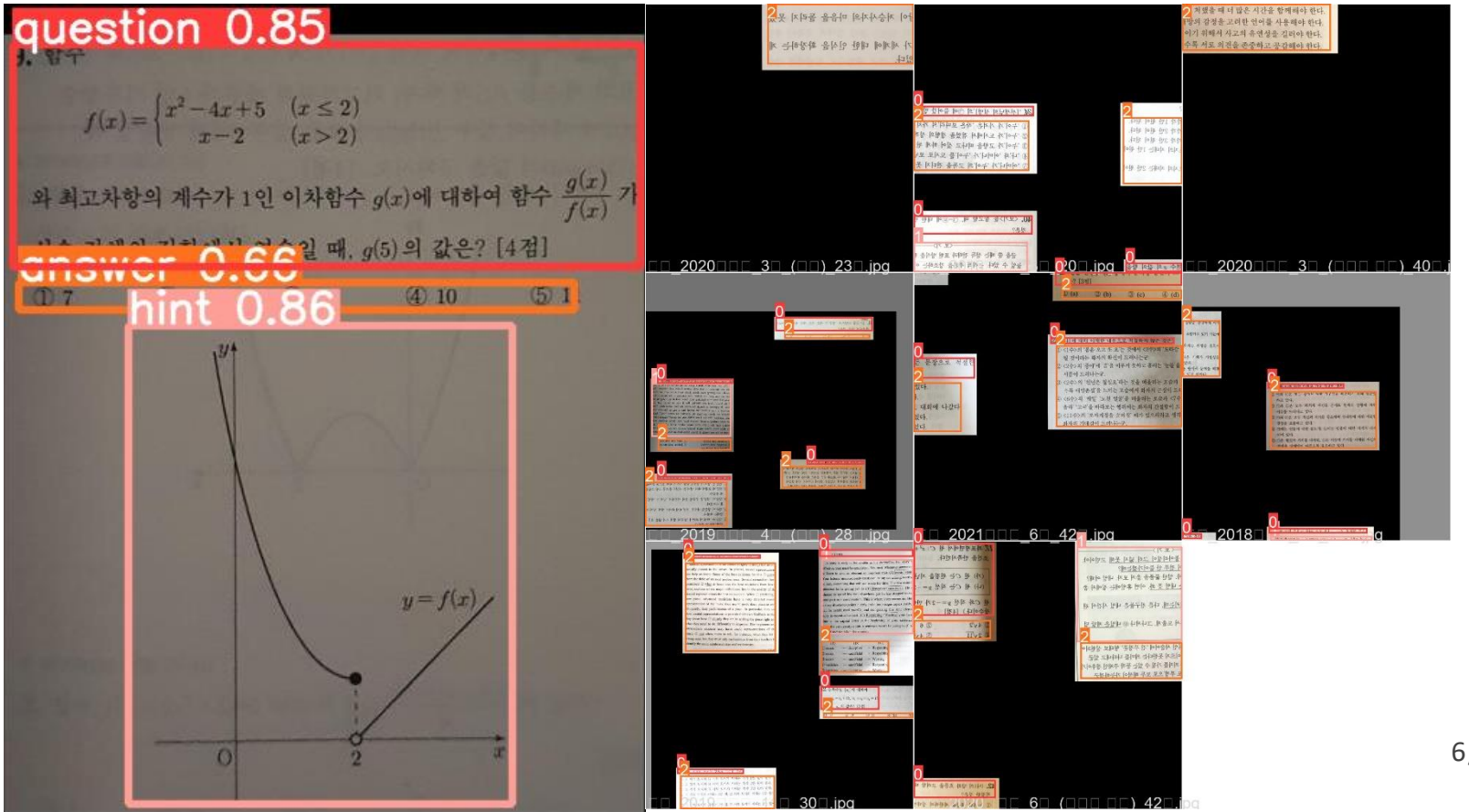
# OD Practice

- **Yolov5[URL]**
  - https://github.com/airobotlab/lecture_5_yolov5
  - 1_train_yolo_colab_220509.ipynb
    - mydaya 폴더 안에 데이터셋 등록(image&label)

```
# 데이터 경로와 class 지정, 중요!!
import yaml

data_yaml = dict(
    train = './mydata/images/test/',
    val = './mydata/images/small/',
    nc = 3,
    names = ['question', 'hint', 'answer']
)

# Note that I am creating the file in the yolov5/data/ directory.
with open('data/mydata.yaml', 'w') as outfile:
    yaml.dump(data_yaml, outfile, default_flow_style=True)
```

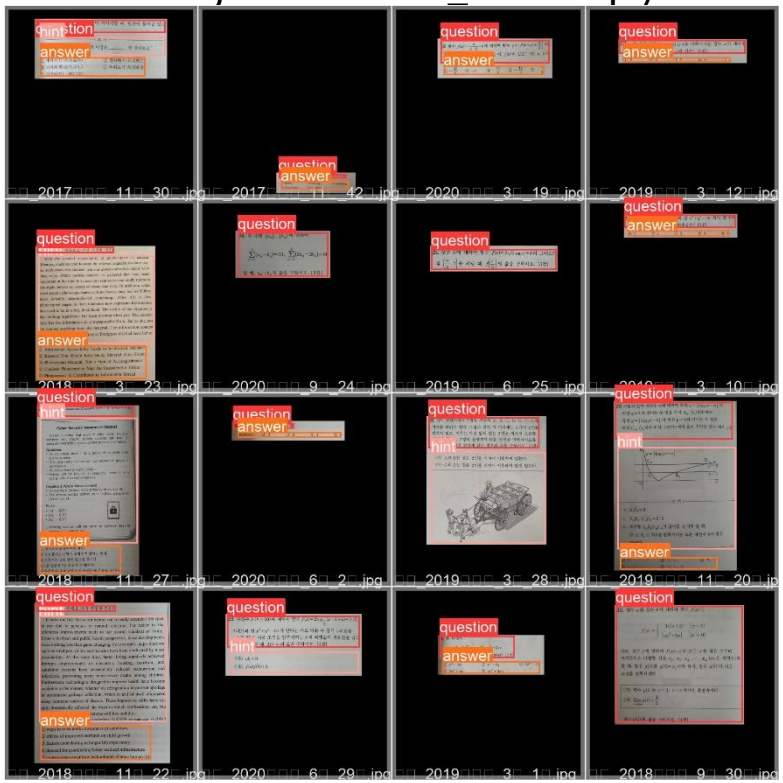
# OD Practice

- **Yolov5[URL]**
  - https://github.com/airobotlab/lecture_5_yolov5
  - 1_train_yolo_colab_220509.ipynb



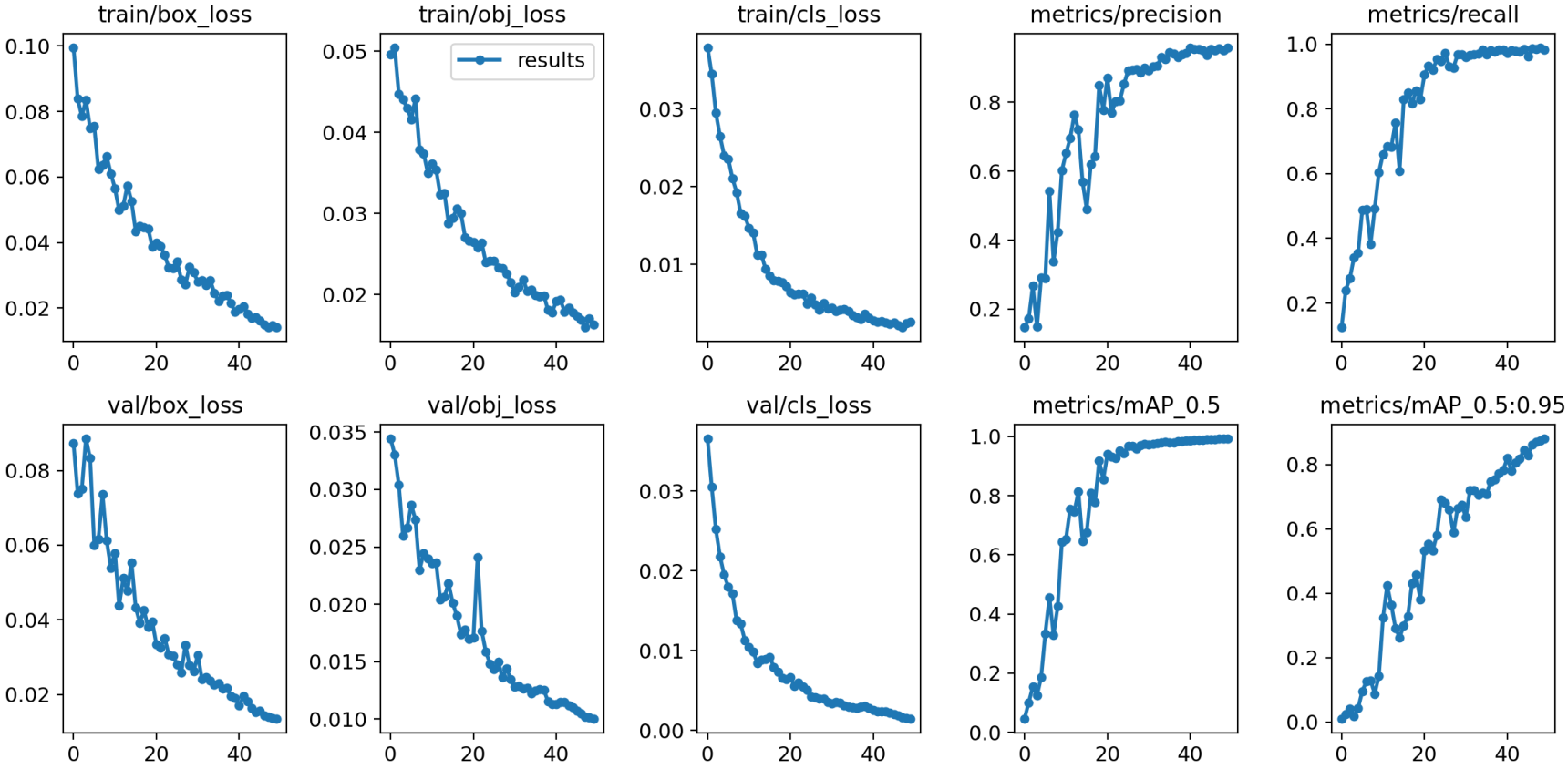
# OD Practice

- **Yolov5[URL]**
  - https://github.com/airobotlab/lecture_5_yolov5
  - 1_train_yolo_colab_220509.ipynb
  - segmentation_colab.ipynb



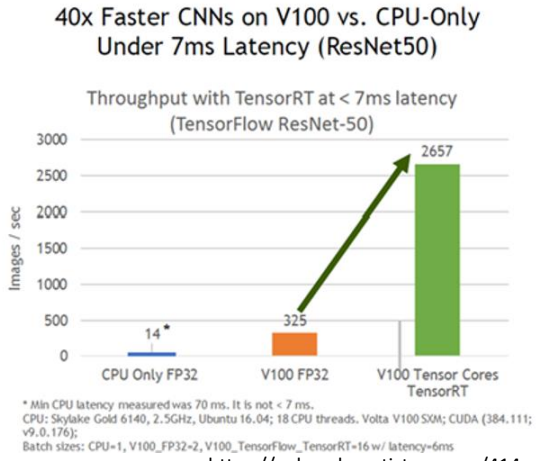
# OD Practice

- **Yolov5[URL]**
  - https://github.com/airobotlab/lecture_5_yolov5
  - 1_train_yolo_colab_220509.ipynb

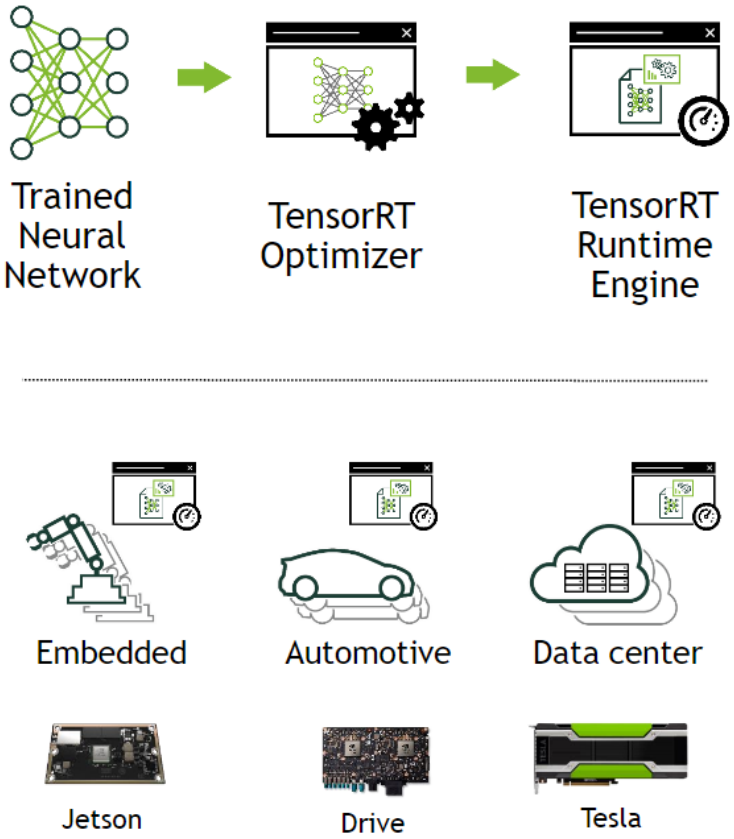


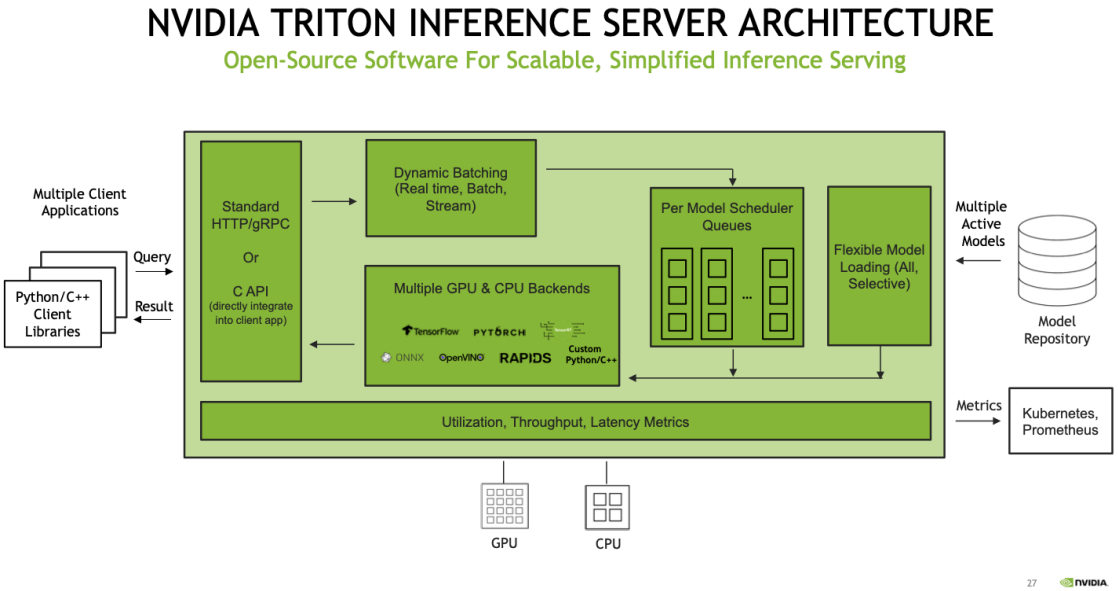
# TensorRT

- **TensorRT**
  - 학습된 딥러닝 모델을 최적화
  - NVIDIA GPU 상에서의 추론 속도를 수배 ~ 수십배 까지 향상
  - 딥러닝 서비스를 개선하는데 도움을 줄 수 있는 모델 최적화 엔진
  - Caffe, Pytorch, TensorFlow 모델을 NVIDIA GPU 플랫폼(TESLA T4 , JETSON TX2, TESLA V100)에 아름답게 싣는 것
  - NVIDIA GPU 연산에 적합한 최적화 기법들을 이용하여 모델을 최적화하는 Optimizer 와 다양한 GPU에서 모델 연산을 수행하는 Runtime Engine 을 포함
    - 양자화 및 정밀도 캘리브레이션
    - 그래프 최적화
    - 커널 자동 튜닝
    - 동적 텐서 메모리 및 멀티 스트림 실행



# TensorRT






https://dataintegration.info/deploy-fast-and-scalable-ai-with-nvidia-triton-inference-server-in-amazon-

# Pose estimation

**Skeleton detection with yolov7**

# Pose estimation 실습

- **Code: https://github.com/airobotlab/colab_pose_detection.git**

- **run 220922_pose_detection_yolov7_colab.ipynb on colab**

# Vision Transformer for OD

## DETR: End-to-End Object Detection with Transformers

Facebook, (ECCV2020)

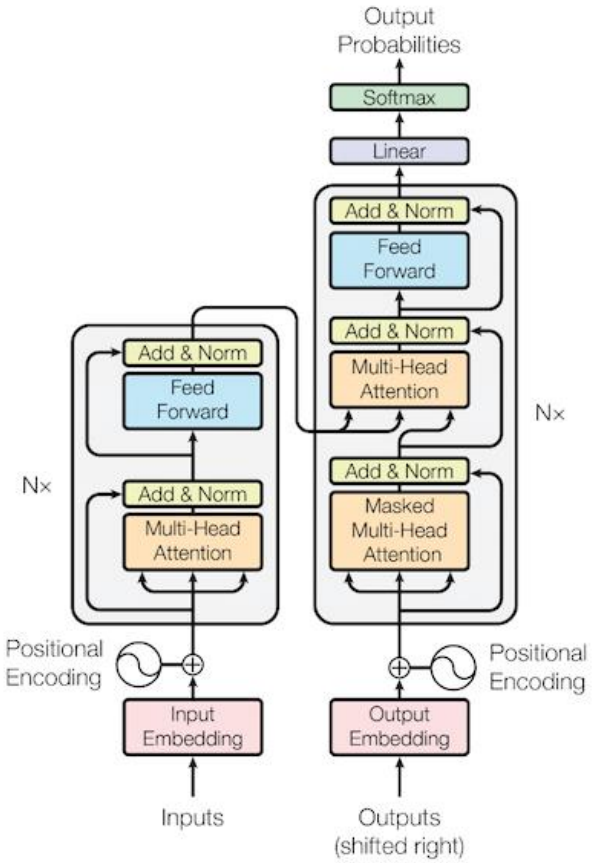
220412

고 우 영

# Object Detection

- **Problems**
  - Object detection task가 "복잡한 라이브러리를 최소화 하고, Classification 처럼 간단하게 행해져야 한다"
  - 기존의 Faster R-CNN, YOLO와 같은 전통적인 object detection 모델은 NMS 등 수 많은 proposals을 추려내는 과정을 거쳐야함
  - DETR은 Transformer encoder-decoder와 Bipartite matching을 이용해 유니크한 예측들을 행하는, 아주 간단한 구조

# Transformer

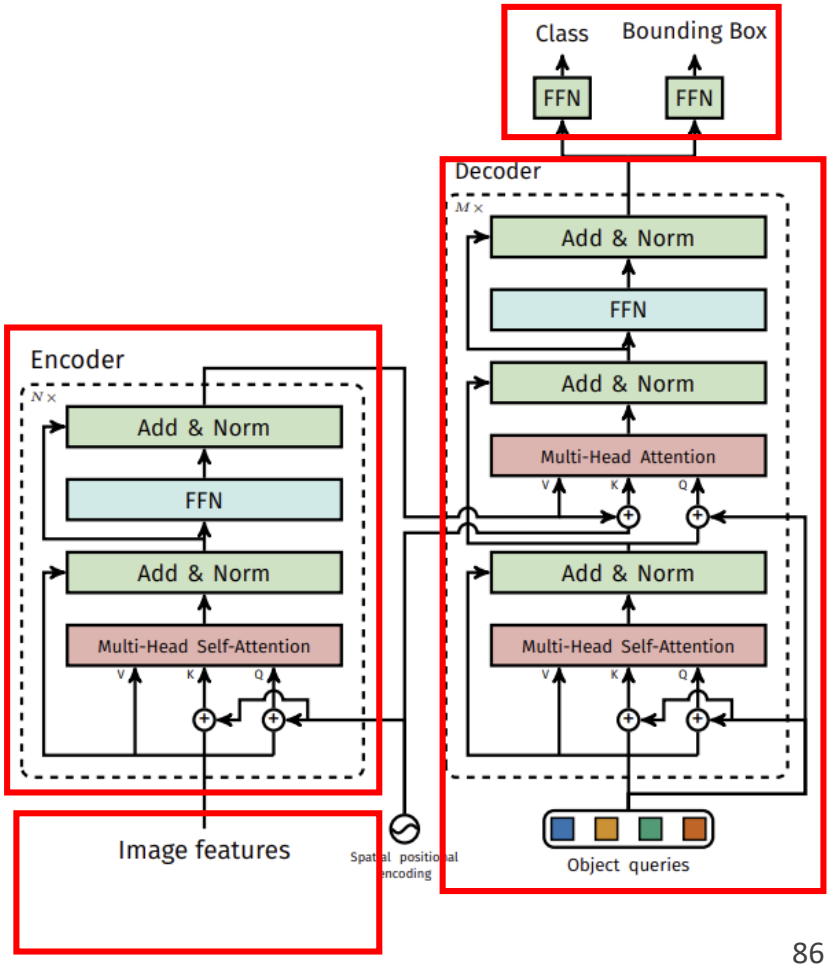


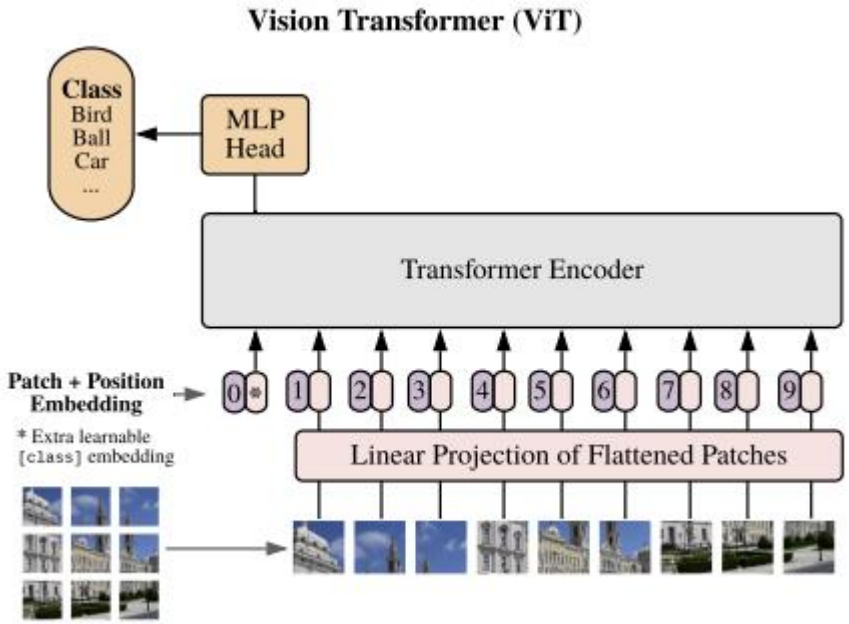



https://hoya012.github.io/blog/Tutorials-of-Object-Detection-Using-Deep-Learning-the-application-of-object-detection/

# Transformer

- 이미지에서 물체(object)를 탐지한 후, 해당 물체의 클래스와 위치를 예측하는 task



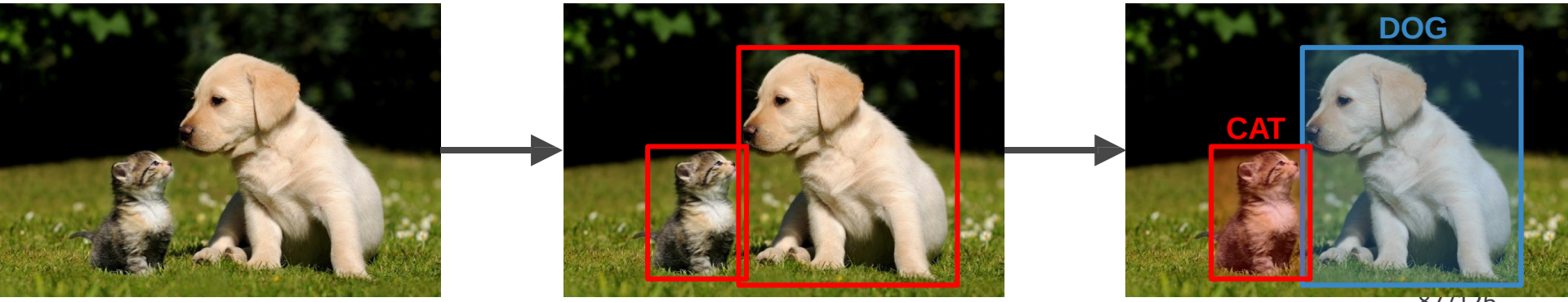






https://hoya012.github.io/blog/Tutorials-of-Object-Detection-Using-Deep-Learning-the-application-of-object-detection/

# DETR: End-to-End Object Detection with Transformers

- **Infomation**
  - paper: https://arxiv.org/pdf/2005.12872.pdf
  - code: https://github.com/facebookresearch/detr
- **Contributions**
  - DETR (DEtection TRansformer)
  - End-to-End Object Detection (ROI extraction + classification)

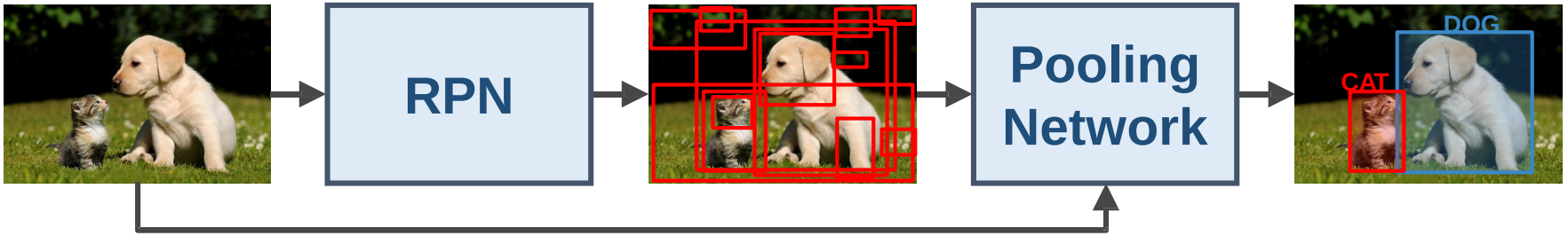


# DETR: End-to-End Object Detection with Transformers

- **Contributions**
  - DETR (DEtection TRansformer)
  - End-to-End Object Detection (ROI extraction + classification)

# DETR: End-to-End Object Detection with Transformers

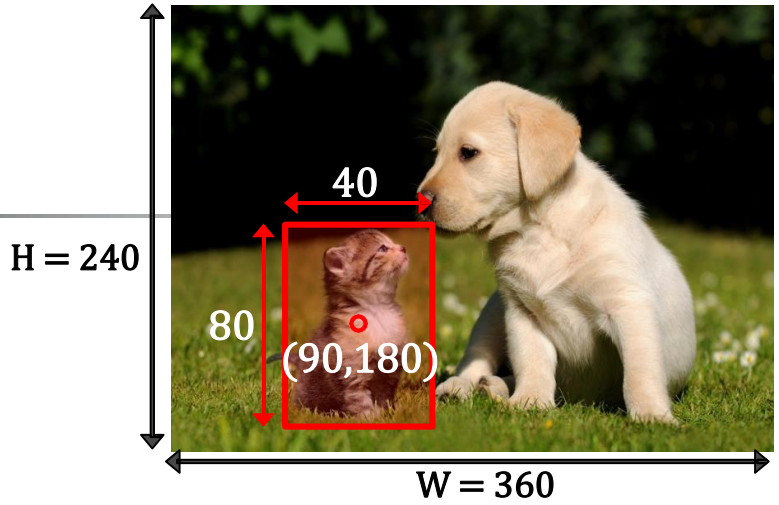
- **Faster R-CNN Model**
  - RPN (Region Proposal Network) + Pooling Network



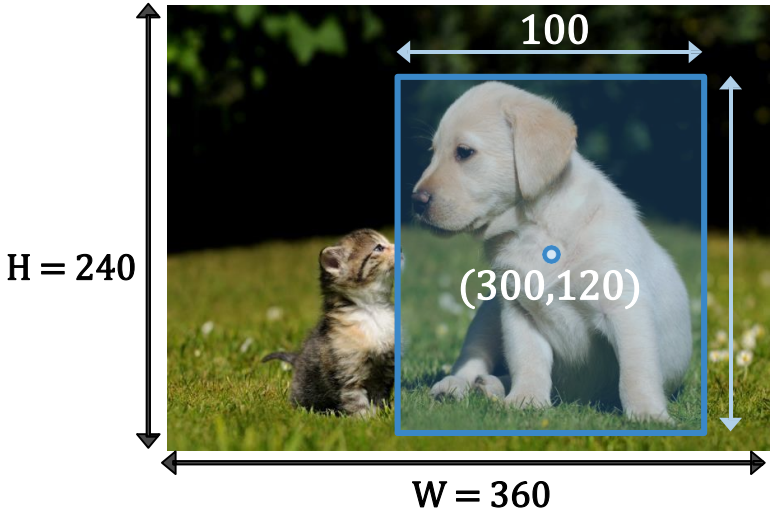

  - Disadvantage: NMS (Non-Maximum Suppression)
    - RPN generates a bunch of bounding boxes
    - It takes long time to remove the redundant bounding boxes
    - NMS is a heuristic → Not end-to-end network → Requires labor-intensive 'heuristic' process

# DETR Input / Target

- **Input**
  - Image $\in \mathbb{R}^{C \times H \times W}$
- **Output**
  - Label: $[class_1, \dots, class_n]$
    - $where\ n = \#class$
  - Box: $[\frac{x_c}{W}, \frac{y_c}{H}, \frac{w}{W}, \frac{h}{H}]$
    - $(x_c, y_c)$: center position
    - $(w, h)$: center position
    - $(W, H)$: image size




- Cat
  - Label: [1, 0]
  - Box: $[\frac{90}{360}, \frac{180}{240}, \frac{40}{360}, \frac{80}{240}]$

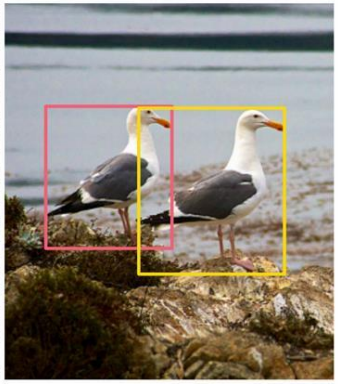



- Dog
  - Label: [0, 1]
  - Box: $[\frac{300}{360}, \frac{120}{240}, \frac{100}{360}, \frac{200}{240}]$

# DETR Input / Target

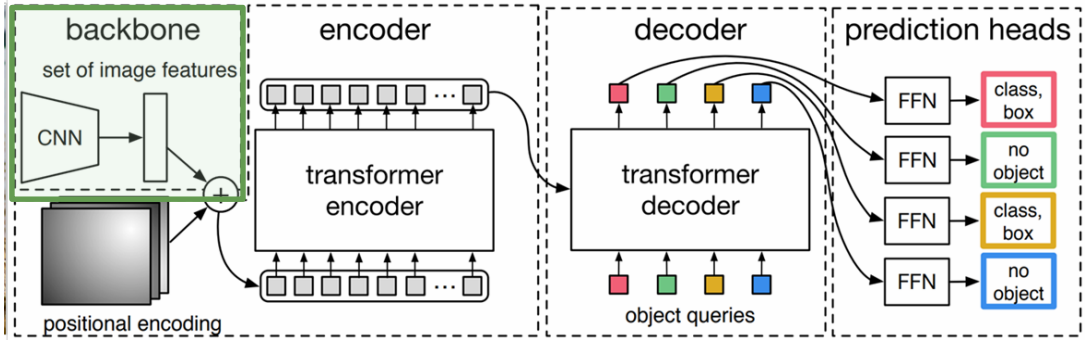
- **DETR consists of 3 parts**
  - 1) Backbone: CNN
  - 2) Encoder: Positional encoding + Transformer
  - 3) Decoder: Transformer

DETR

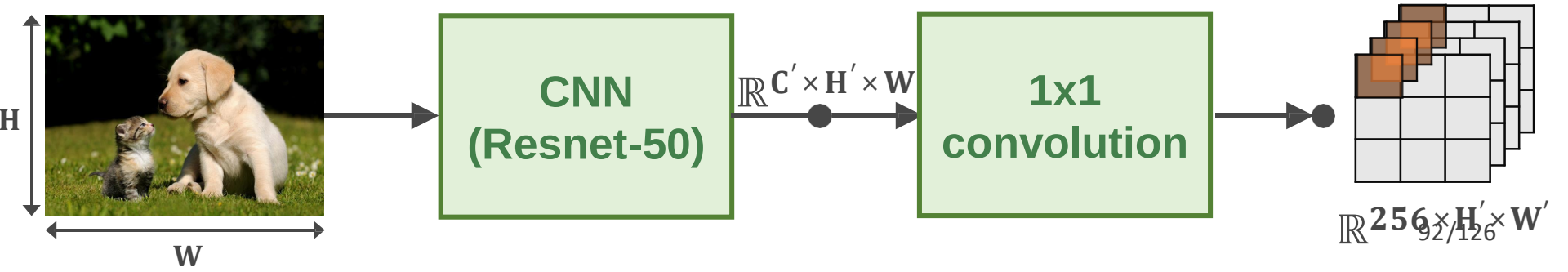


# DETR ARCHITECTURE

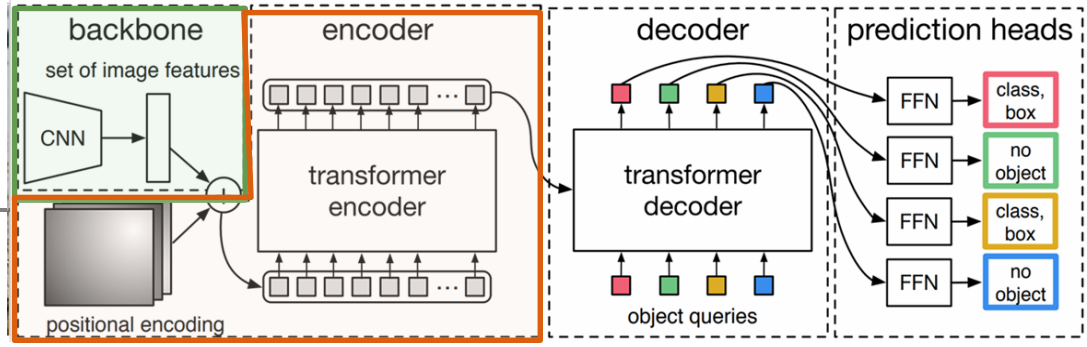
- **1) Backbone**
  - CNN (Resnet-50)
    - $Output \in \mathbb{R}^{C' \times H' \times W'}$
    - $where\ C' = 2048,\ H' = \frac{H}{32},\ W' = \frac{W}{32}$



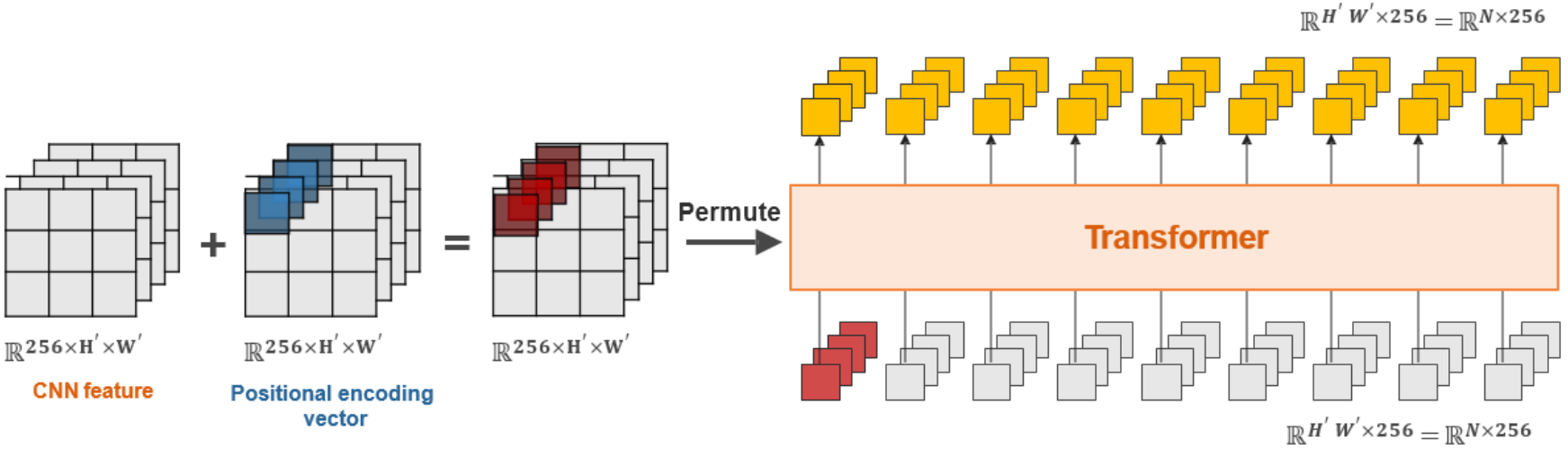

  - 1X1 Convolution
    - $Output \in \mathbb{R}^{256 \times H' \times W'}$
    - Reduce # of channel to 256



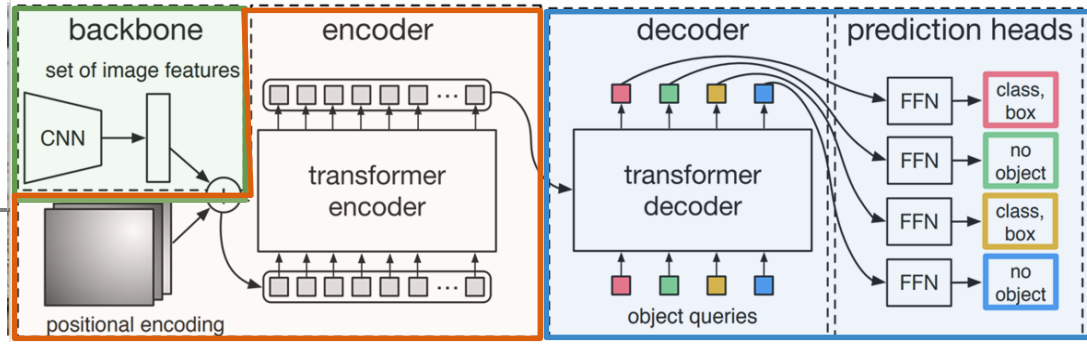
# DETR ARCHITECTURE



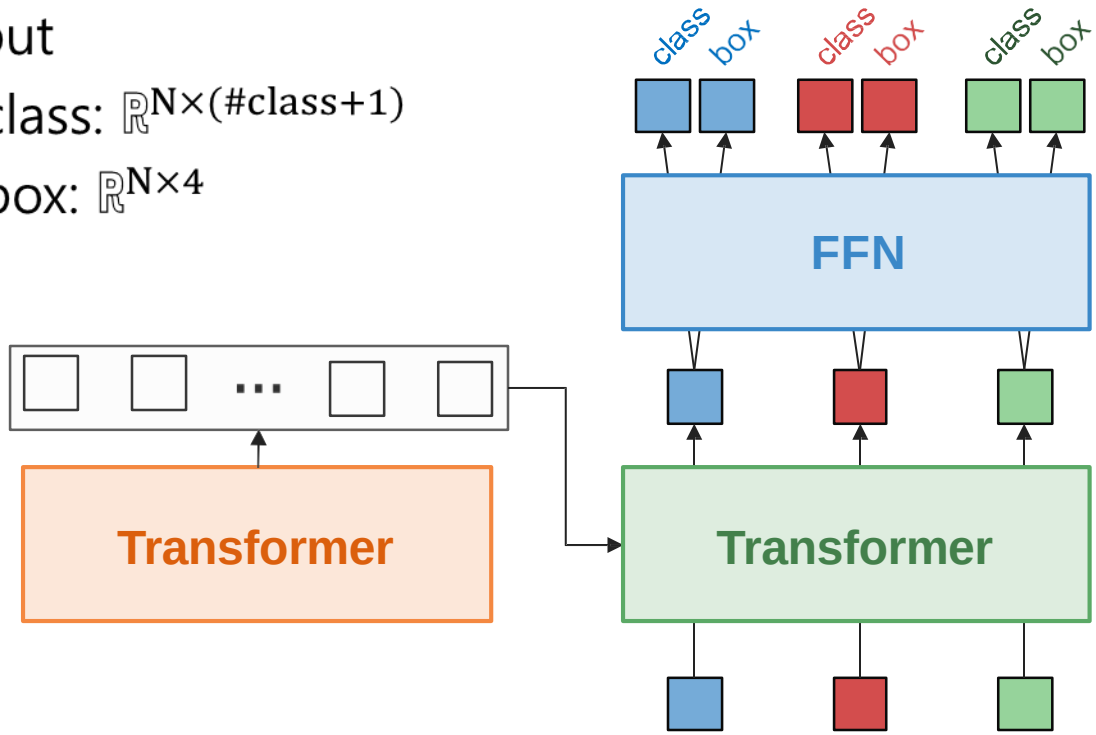

- **2) Encoder**
  - CNN (Resnet-50)
    - $Output \in \mathbb{R}^{C' \times H' \times W'}$
    - $where\ C' = 2048,\ H' = \frac{H}{32},\ W' = \frac{W}{32}$
  - Input: CNN feature + Positional encoding vector
  - Output: $\mathbb{R}^{N \times 256}$
    - Permute tensor's dimension from $\mathbb{R}^{256 \times H' \times W'}$ to $\mathbb{R}^{N \times 256}$ $where\ N = H'W'$



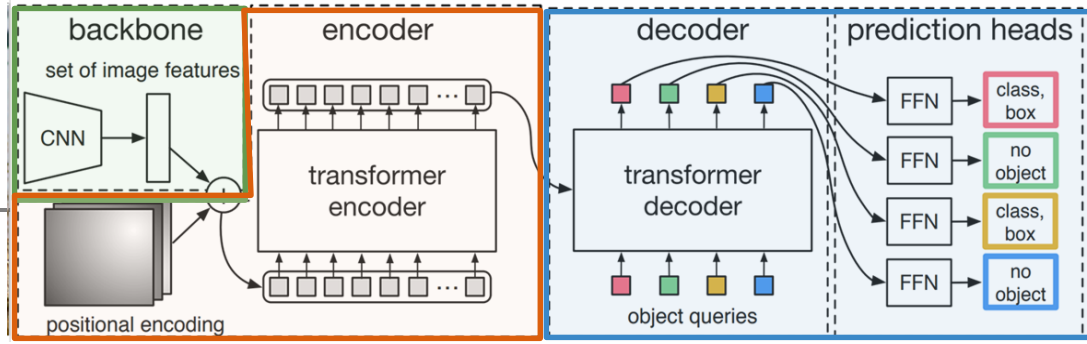
# DETR ARCHITECTURE




- **3) Decoder**
  - Input
    - Encoder's output
    - Object queries $\in \mathbb{R}^{N\times 256}$
    - *where* $N = maximum$ # of objects (predefined number), learnable parameters
  - Output
    - class: $\mathbb{R}^{N\times(\#class+1)}$
    - box: $\mathbb{R}^{N\times 4}$



# DETR ARCHITECTURE




- **3) Decoder**
  - 이미지의 어느 부분을 위주로 봐야할 지 (물체가 어느 위치에 있
    확률이 높을 지)

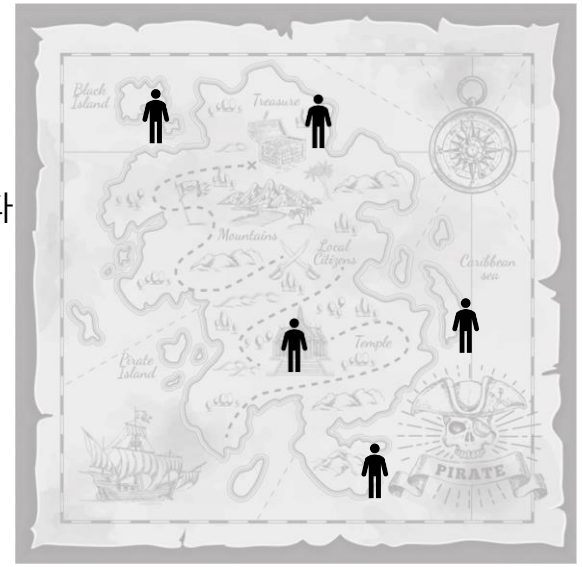
보물지도 (이미지)

보물이 여기 있을 거 같은데?

**Enc-dec attention**
보물이 이쯤에 있을 것이다

**Self attention**
내가 이미 찾았으니
너는 다른 곳을 찾아봐라

보물 사냥꾼 (Object Queries)

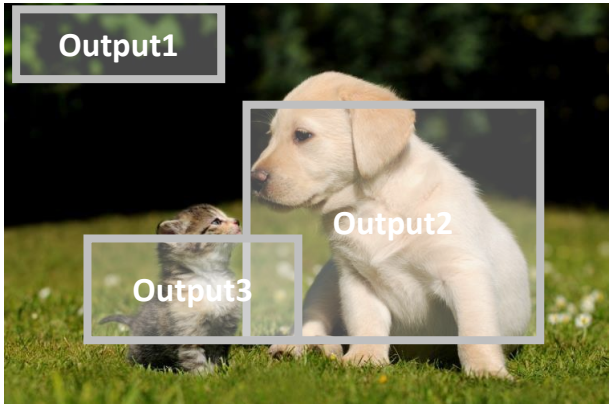


X 나머지

# DETR Loss

- **Bipartite matching**
  - Assignment problem
    - → Hungarian matching algorithm

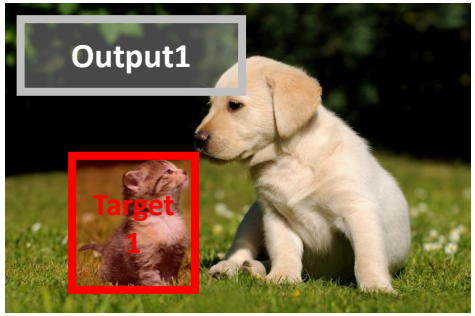
$$\hat{\sigma} = \arg\min_{\sigma \in \mathfrak{S}_N} \sum_{i}^{N} \mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)})$$





- **Loss combination: Case 1**

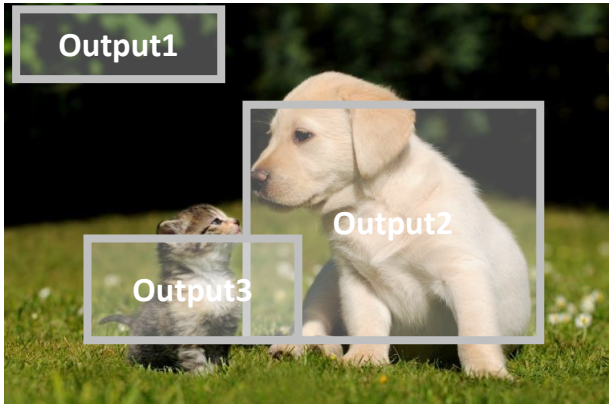
| | Target 1 | Target 2 | $\phi$ |
|---|---|---|---|
| Output 1 | $L(y_1, \hat{y}_1)$ | $L(y_1, \hat{y}_2)$ | 0 |
| Output 2 | $L(y_2, \hat{y}_1)$ | $L(y_2, \hat{y}_2)$ | **0** |
| Output 3 | $L(y_3, \hat{y}_1)$ | $L(y_3, \hat{y}_2)$ | 0 |




# DETR Loss

- **Bipartite matching**
  - Assignment problem
    - → Hungarian matching algorithm

$$\hat{\sigma} = \arg\min_{\sigma \in \mathfrak{S}_N} \sum_{i}^{N} \mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)})$$





- **Loss combination: Case 1**

| | Target 1 | Target 2 | ϕ |
|---|---|---|---|
| Output 1 | $L(y_1, \hat{y}_1)$ | $L(y_1, \hat{y}_2)$ | **0** |
| Output 2 | $L(y_2, \hat{y}_1)$ | $L(y_2, \hat{y}_2)$ | 0 |
| Output 3 | $L(y_3, \hat{y}_1)$ | $L(y_3, \hat{y}_2)$ | 0 |


# DETR Loss

- **Hungarian Loss**

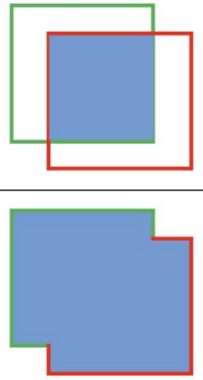
$$\mathcal{L}_{\text{Hungarian}}(y, \hat{y}) = \sum_{i=1}^{N} \left[ -\log \hat{p}_{\hat{\sigma}(i)}(c_i) + \mathbb{1}_{\{c_i \neq \varnothing\}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\hat{\sigma}}(i)) \right]$$

class 예측 bbox 예측

$B_p$ = 실제 (*Gound Truth*)

$B_{gt}$ = 예측 (*Prediction*)

$$IOU = \frac{area\ of\ overlap}{area\ of\ union} = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})} = \frac{\text{실제} \cap \text{예측 중복 영역}}{\text{실제} \cup \text{예측 전체 영역}}$$



waytoliah.com



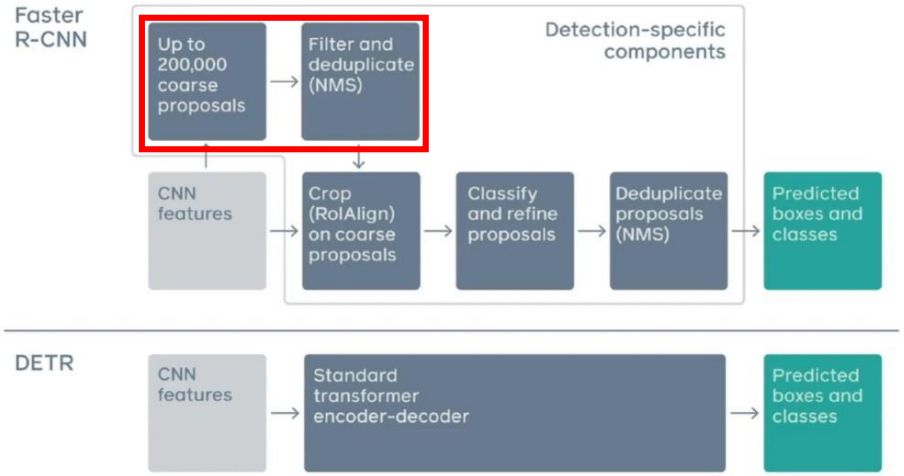
| Sample IOU Score | IOU Threshold |
|---|---|
| IOU=0.50 | >=0.50 |
| IOU=0.75 | >=0.75 |
| IOU=0.95 | >=0.95 |

waytoliah.com

https://ballentain.tistory.com/12
https://www.waytoliah.com/1491

# Experiment

- **Comparison with Faster R-CNN**
  - R-CNN
    - RPN(Region Proposal Network)과 NMS(Non-Maximum Suppression)을 거쳐 최종적인 후보 위치를 예측
    - 해당 지역들에 대해 다시 한번 detection을 수행하는 복잡한 pipeline
  - DETR
    - 이미지에 관하여 직접적이고 절대적인(absolute) 방식으로 box set을 예측
    - NMS와 같은 hand-crafted 과정을  사용하지 않아 end-to-end 구조를 구축



# Experiment

- **Comparison with Faster R-CNN**
  - Dataset: COCO 2017 detection dataset
  - 118k training images / 5k validation images
  - 7 instances per image on average
    - (up to 63 instances in a single image)
  - Optimizer: AdamW
  - Backbone: pretrained Resnet50, ResNet101
  - # GPU: 16 (V100)
  - Epoch: 300
  - Training time: 72 hours

# Experiment

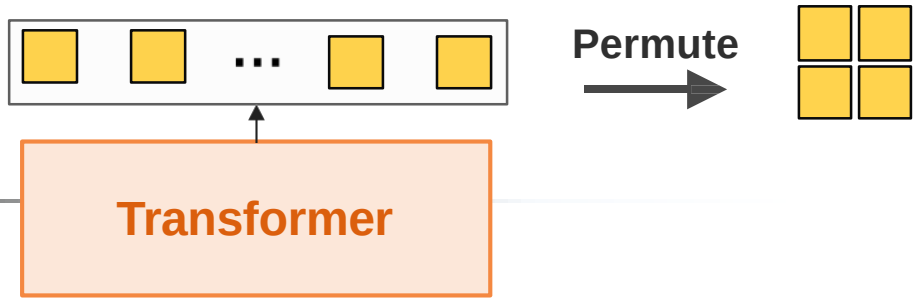
- **Comparison with Faster R-CNN**
  - Use less parameters, low computation cost
  - Performance is comparable to the Faster RCNN
    - Better performance for large objects ($AP_L$)
    - worse performance for small objects ($AP_S$)

| Model | GFLOPS/FPS | #params | AP | $AP_{50}$ | $AP_{75}$ | $AP_S$ | $AP_M$ | $AP_L$ |
|---|---|---|---|---|---|---|---|---|
| Faster RCNN-DC5 | 320/16 | 166M | 39.0 | 60.5 | 42.3 | 21.4 | 43.5 | 52.5 |
| Faster RCNN-FPN | 180/26 | 42M | 40.2 | 61.0 | 43.8 | 24.2 | 43.5 | 52.0 |
| Faster RCNN-R101-FPN | 246/20 | 60M | 42.0 | 62.5 | 45.9 | 25.2 | 45.6 | 54.6 |
| Faster RCNN-DC5+ | 320/16 | 166M | 41.1 | 61.4 | 44.3 | 22.9 | 45.9 | 55.0 |
| Faster RCNN-FPN+ | 180/26 | 42M | 42.0 | 62.1 | 45.5 | 26.6 | 45.4 | 53.4 |
| Faster RCNN-R101-FPN+ | 246/20 | 60M | 44.0 | 63.9 | **47.8** | **27.2** | 48.1 | 56.0 |
| DETR | 86/28 | 41M | 42.0 | 62.4 | 44.2 | 20.5 | 45.8 | 61.1 |
| DETR-DC5 | 187/12 | 41M | 43.3 | 63.1 | 45.9 | 22.5 | 47.3 | 61.1 |
| DETR-R101 | 152/20 | 60M | 43.5 | 63.8 | 46.4 | 21.9 | 48.0 | 61.8 |
| DETR-DC5-R101 | 253/10 | 60M | **44.9** | **64.7** | 47.7 | 23.7 | **49.5** | **62.3** |

**Faster RCNN+**

**DETR**

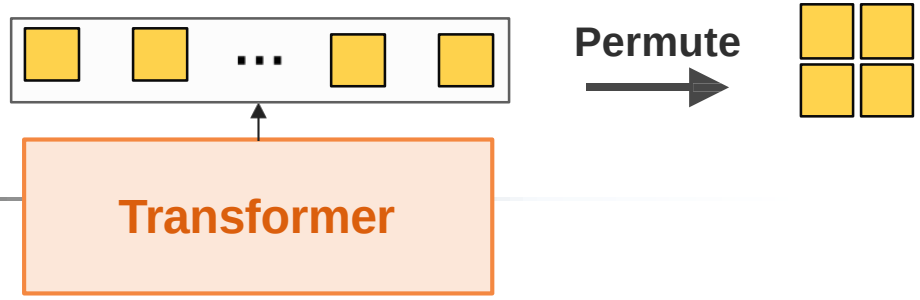
# Experiment



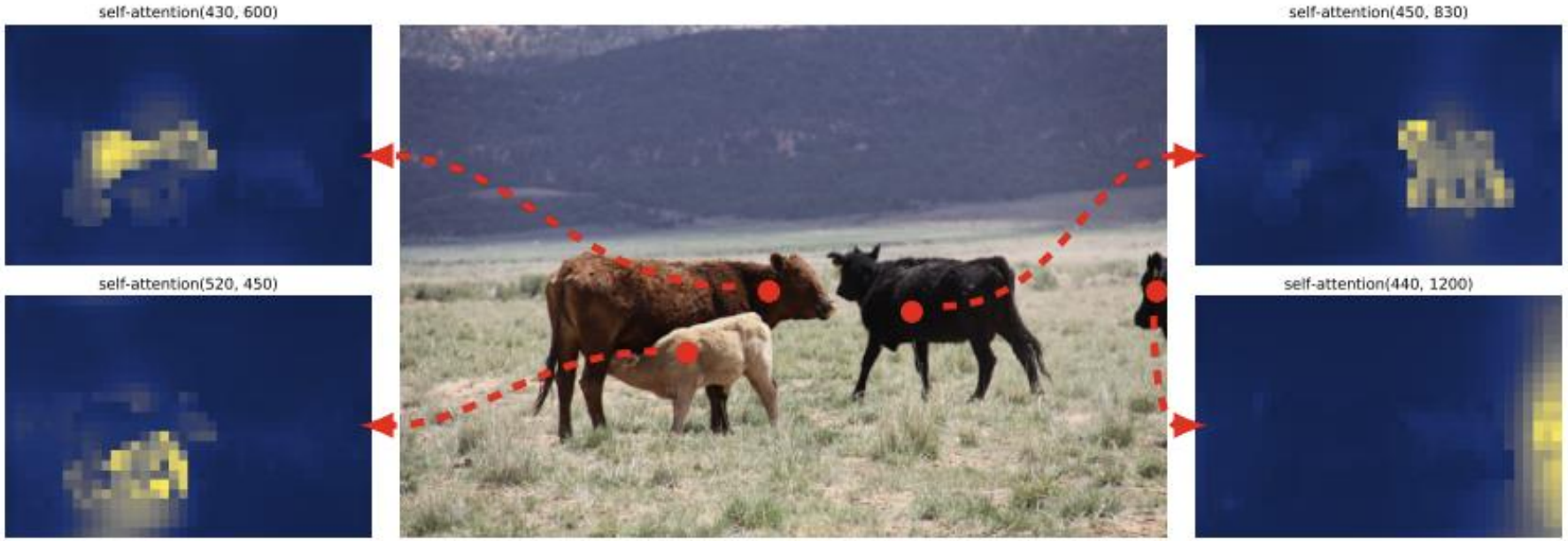

- **Ablations**
  - The effect of self-attention in Encoder
    - Attention map을 통해 Encoder 에서부터 이미 물체를 어느 정도 구분
    - 학습된 임베딩을 Key와 Value로 사용하는 Decoder가 detection 하는 과정을 좀 더 쉽게 만들어 줌
    - Encoder를 사용하지 않는 경우 약 4~5 AP 감소
      - 큰 물체(APL)에 대한 detection 성능이 크게 저하됨
    - DETR의 attention mechanism이 이미지 내 물체를 분리(disentangle)하는 데에 굉장히 중요함을 의미

| #layers | GFLOPS/FPS | #params | AP | $AP_{50}$ | $AP_S$ | $AP_M$ | $AP_L$ |
|---|---|---|---|---|---|---|---|
| 0 | 76/28 | 33.4M | 36.7 | 57.4 | 16.8 | 39.6 | 54.2 |
| 3 | 81/25 | 37.4M | 40.1 | 60.6 | 18.5 | 43.8 | 58.6 |
| 6 | 86/23 | 41.3M | 40.6 | 61.6 | 19.9 | 44.3 | 60.2 |
| 12 | 95/20 | 49.2M | 41.6 | 62.1 | 19.8 | 44.9 | 61.9 |

# Experiment

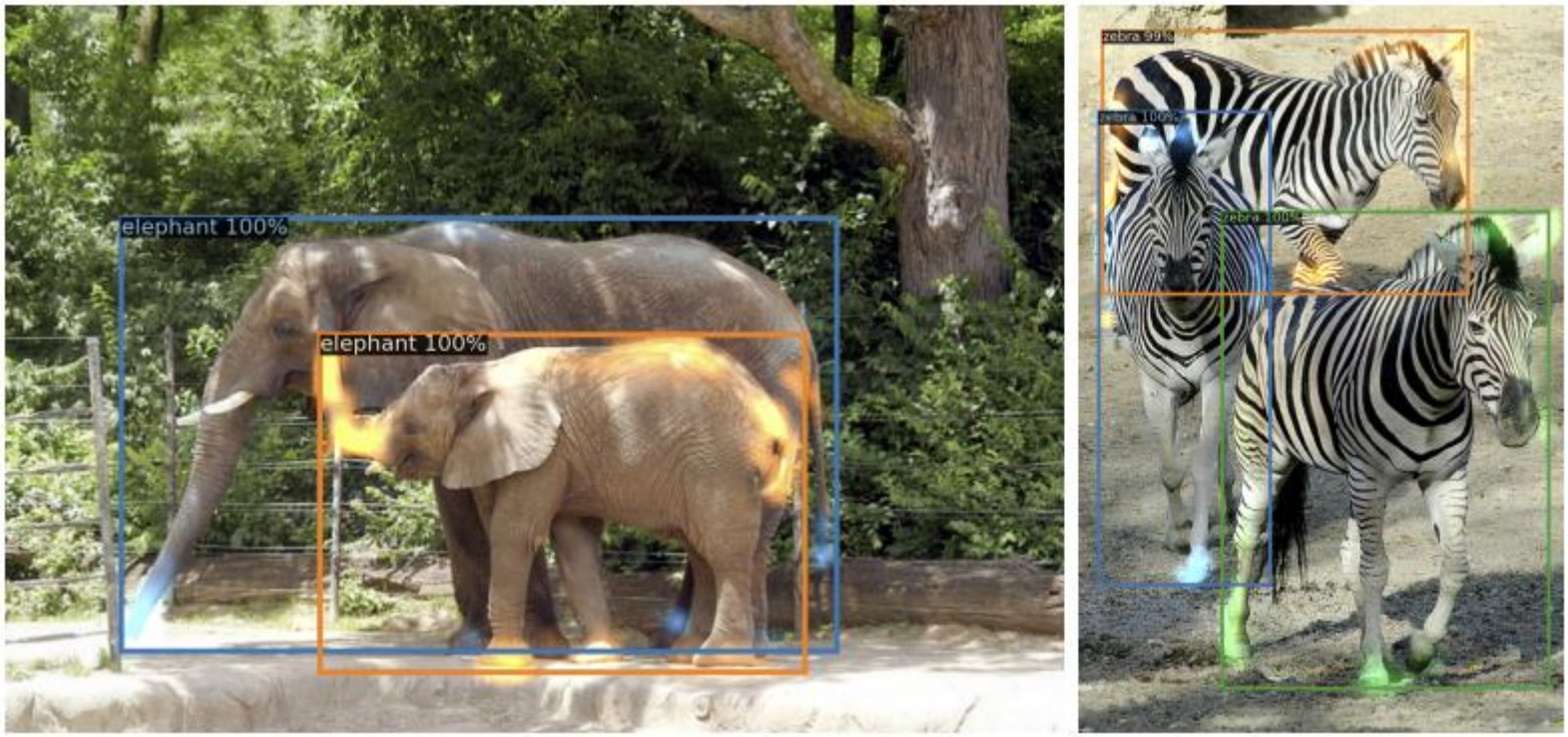



- **Ablations**
  - The effect of self-attention in Encoder
    - Encoder self-attention for a set of reference points
    - Encoder is able to separate individual instances

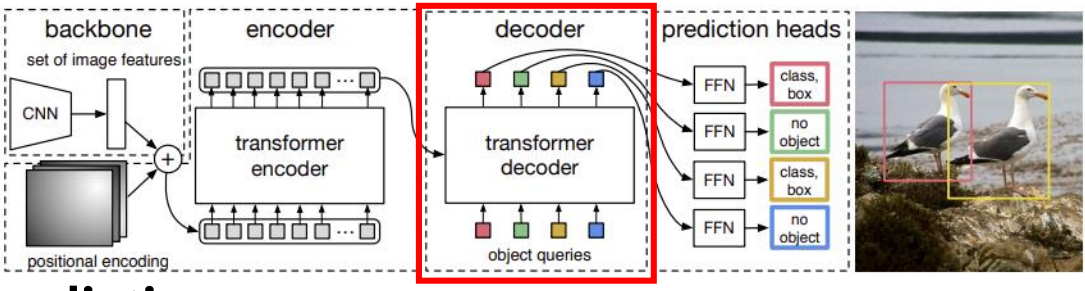


# Experiment

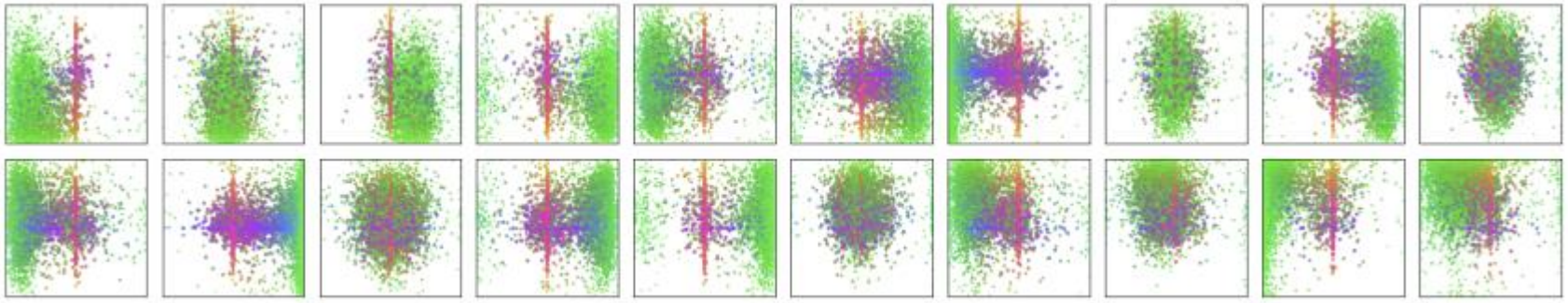
- **Ablations**
  - The effect of self-attention in Decoder
    - Visualizing decoder attention for every predicted object



# Experiment

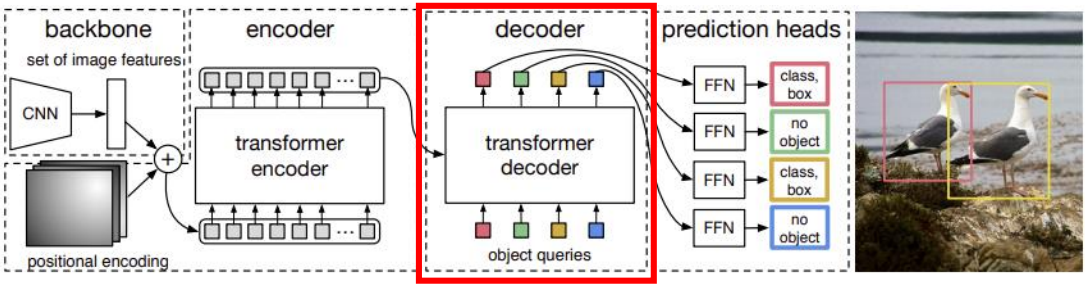



- **Visualization of all box predictions**
  - Visualization of all box predictions on all images from COCO 2017 val set for 20 out of total N = 100 prediction slots



  - Each query produces bounding box [$xc$, $yc$, $w$, $h$]
  - Plot center positions $xc$, $yc$ of all images
  - The points are color-coded
    - green: 작은 물체
    - red: 수평으로 큰 물체
    - blue: 수직으로 큰 물체

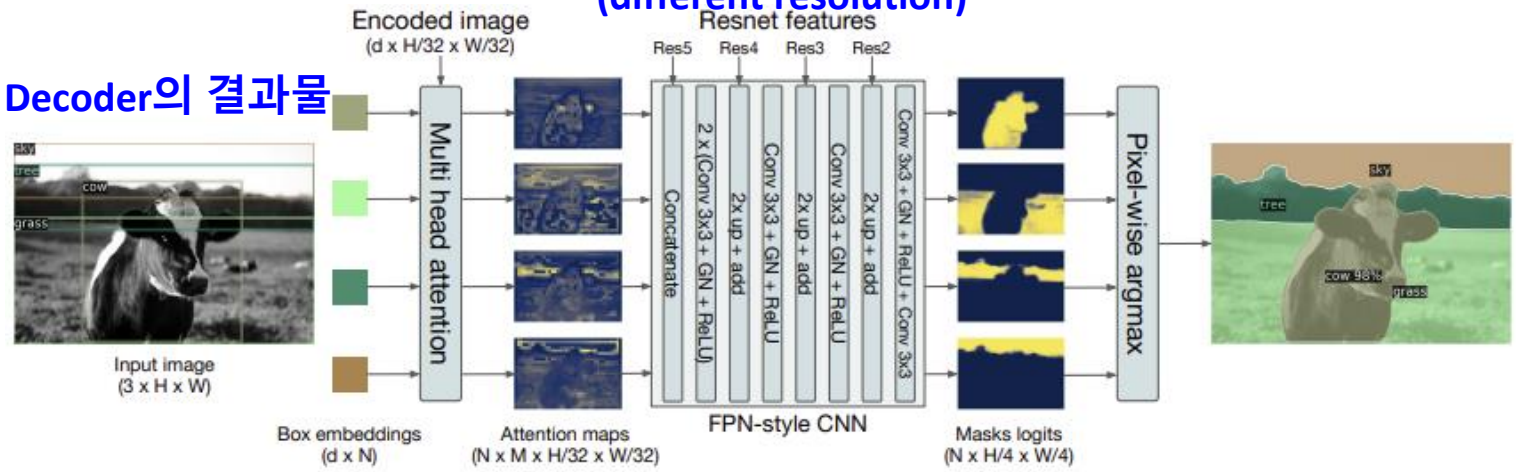
# Experiment



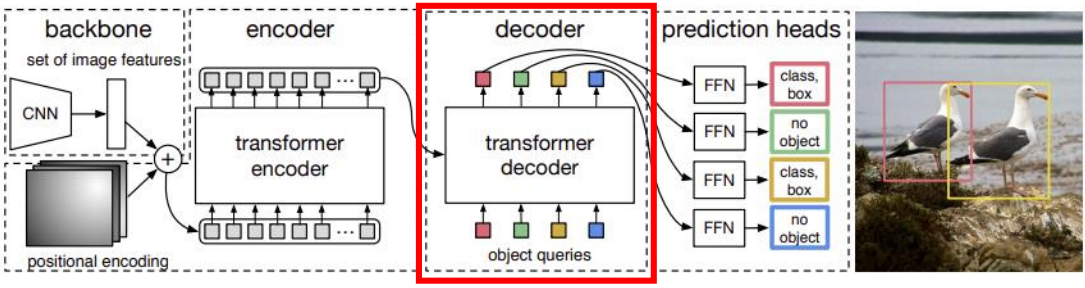

- **Panoptic segmentation**
  - DETR can be naturally extended by adding a mask head on top of the decoder outputs
  - DETR의 decoder가 출력한 결과물(object queries)을 encoder에 의해 encoding 된 결과물과 attention 수행
  - attention map을 기반으로 여러 resolution의 feature map과의 연산을 통해 masked image를 얻음
  - 해당 masked image의 픽셀마다 argmax를 적용하여 특정 object로의 분류 수행

**따로 저장해 둔 feature maps (different resolution)**

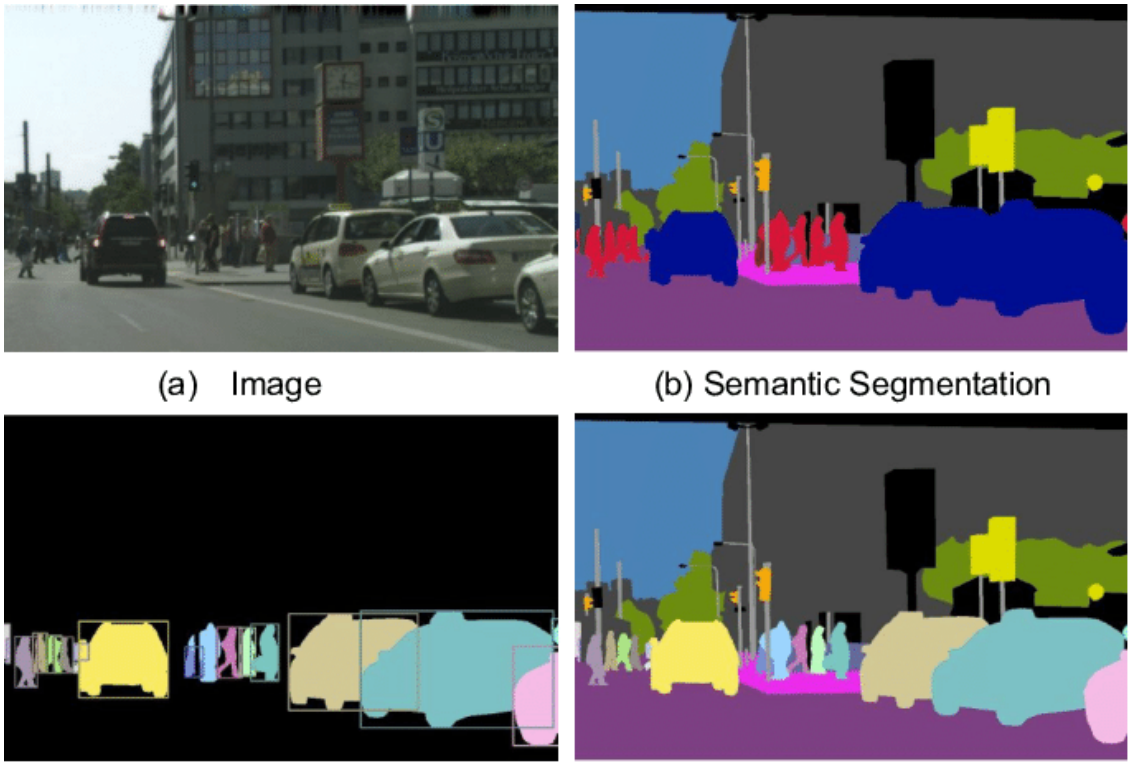
**Decoder의 결과물**



# Experiment



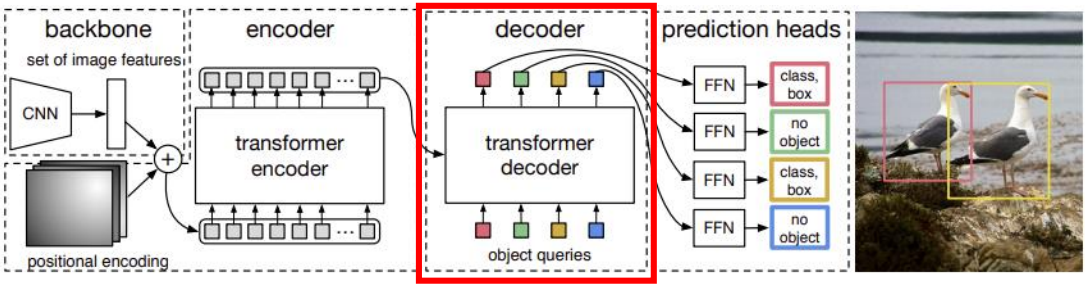

- **Panoptic segmentation**
  - Semantic+Instance
  - 모든 픽셀을 사전 정해진 class로 분류 + 동일 class 내에서도 서로 다른 객체를 구분
  - DETR can be naturally extended by adding a mask head on top of the decoder outputs



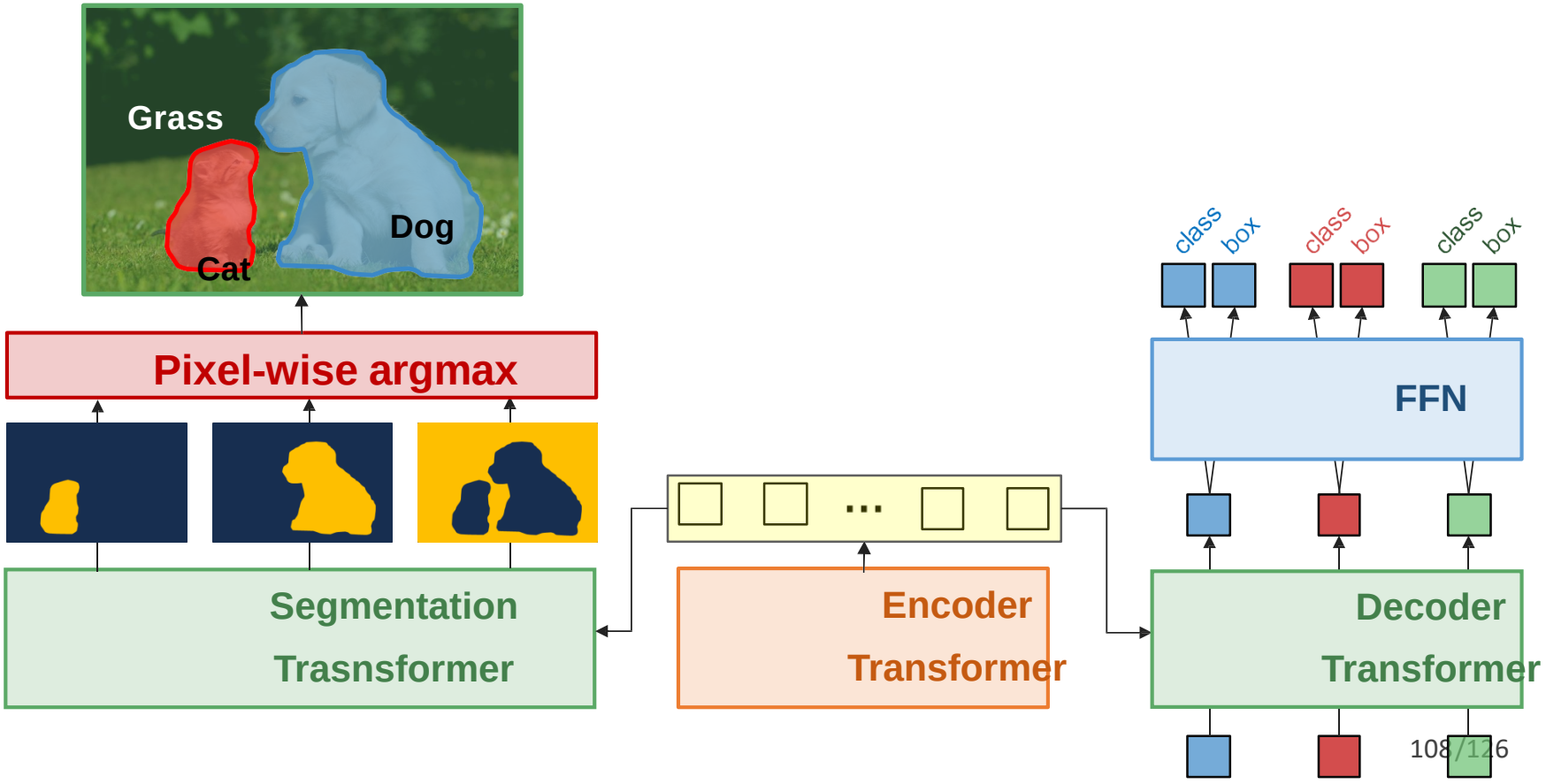



https://www.researchgate.net/figure/b-semantic-segmentation-c-instance-segmentation-and-d-panoptic-segmentation-for_fig5_342409316

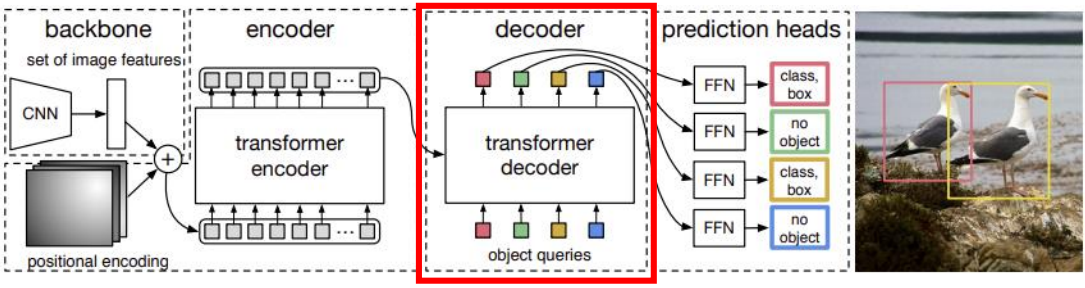
# Experiment




- **Panoptic segmentation**
  - Visualization of all box predictions on all images from COCO 2017 val set for 20 out of total N = 100 prediction slots



# Experiment




- **Panoptic segmentation**
  - Visualization of all box predictions on all images from COCO 2017 val set for 20 out of total N = 100 prediction slots



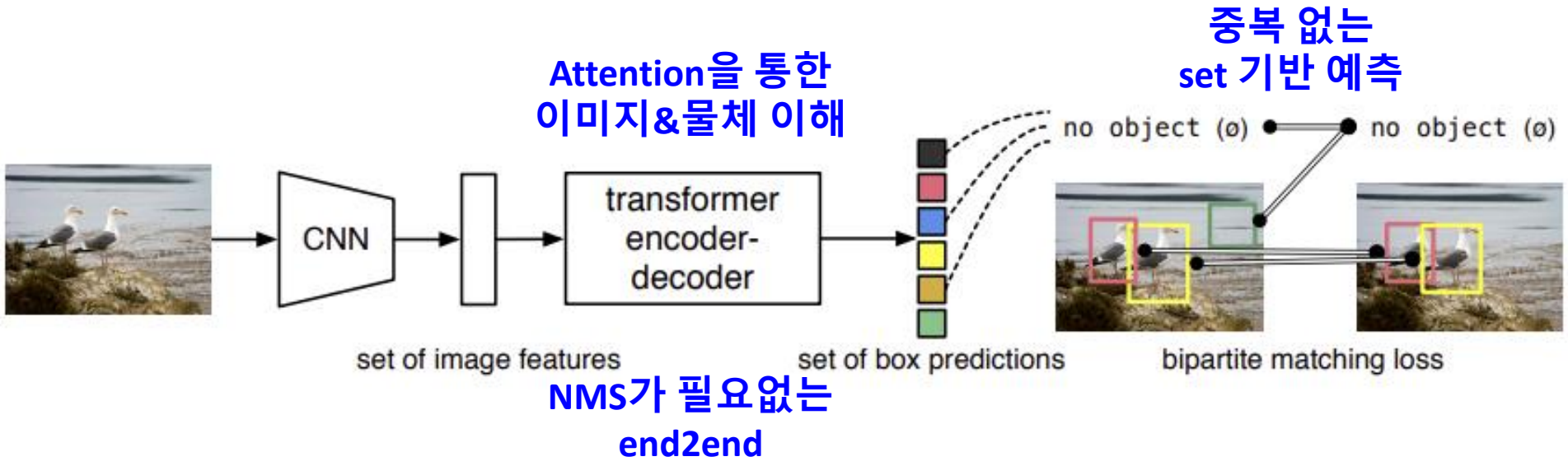
# Source CODE

- **Written by Pytorch**

```python
import torch
from torch import nn
from torchvision.models import resnet50

class DETR(nn.Module):

    def __init__(self, num_classes, hidden_dim, nheads,
                 num_encoder_layers, num_decoder_layers):
        super().__init__()
        # We take only convolutional layers from ResNet-50 model
        self.backbone = nn.Sequential(*list(resnet50(pretrained=True).children())[:-2])
        self.conv = nn.Conv2d(2048, hidden_dim, 1)
        self.transformer = nn.Transformer(hidden_dim, nheads,
                                          num_encoder_layers, num_decoder_layers)
        self.linear_class = nn.Linear(hidden_dim, num_classes + 1)
        self.linear_bbox = nn.Linear(hidden_dim, 4)
        self.query_pos = nn.Parameter(torch.rand(100, hidden_dim))
        self.row_embed = nn.Parameter(torch.rand(50, hidden_dim // 2))
        self.col_embed = nn.Parameter(torch.rand(50, hidden_dim // 2))

    def forward(self, inputs):
        x = self.backbone(inputs)
        h = self.conv(x)
        H, W = h.shape[-2:]
        pos = torch.cat([
            self.col_embed[:W].unsqueeze(0).repeat(H, 1, 1),
            self.row_embed[:H].unsqueeze(1).repeat(1, W, 1),
        ], dim=-1).flatten(0, 1).unsqueeze(1)
        h = self.transformer(pos + h.flatten(2).permute(2, 0, 1),
                             self.query_pos.unsqueeze(1))
        return self.linear_class(h), self.linear_bbox(h).sigmoid()

detr = DETR(num_classes=91, hidden_dim=256, nheads=8, num_encoder_layers=6, num_decoder_layers=6)
detr.eval()
inputs = torch.randn(1, 3, 800, 1200)
logits, bboxes = detr(inputs)
```

# CONCLUSION

- **DETR (DEtection TRansformer)**
  - OD를 direct set-prediction 관점에서 새롭게 접근
  - 트랜스포머 구조를 활용하는 것과 더불어 geometric Prior(RPN, NMS)를 없앰으로써 굉장히 간결한 pipeline을 구축함
  - 확장성이 뛰어나고 충분히 competitive한 성능을 얻어내어 object detection 및 vision 분야에서 transformer의 높은 잠재성을 확인



# CONCLUSION

- **DETR (DEtection TRansformer)**
  - **Advantages**
    - Achieves comparable results to Faster R-CNN baseline on the COCO dataset
    - Very straightforward to implement and flexible architecture
    - Achieves better performance on large objects than Faster R-CNN
  - **Disadvantages**
    - long training time
    - worse performance on small objects
    - ※These drawbacks were solved by the follow-up paper,

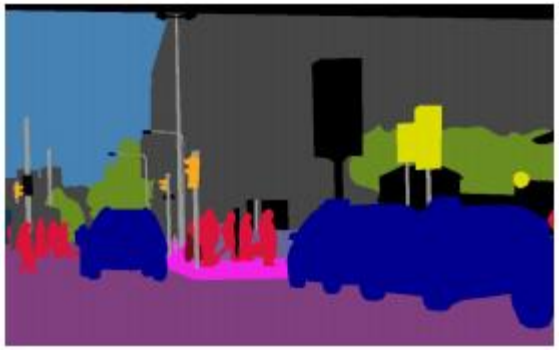
      "Deformable Transformers for End-to-End Object Detection,", 2020
  - **Applications**
    - OCR, Smart Review, etc.

# Reference


- http://dsba.korea.ac.kr/seminar/?mod=document&uid=1784
- https://jihyeonryu.github.io/2021-04-02-survey-paper1/
- https://arxiv.org/pdf/2010.11929.pdf

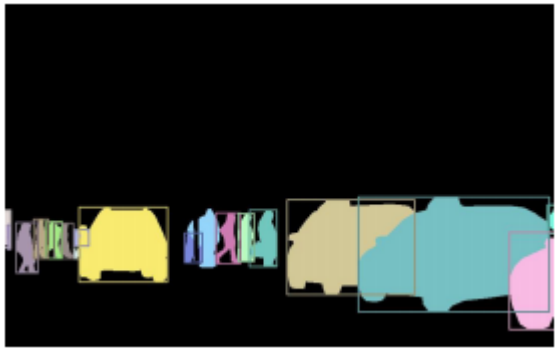
# Segmentation

**Semantic/Instance/Panoptic segmentation**



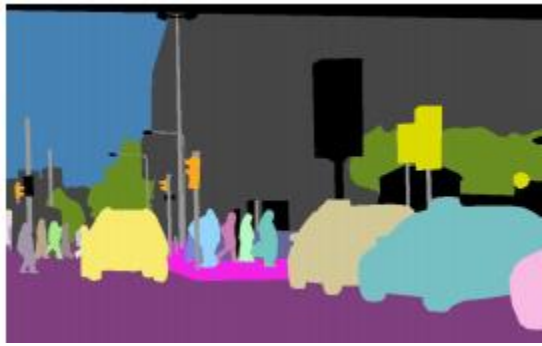
**Semantic Segmentation**

+



**Instance Segmentation**

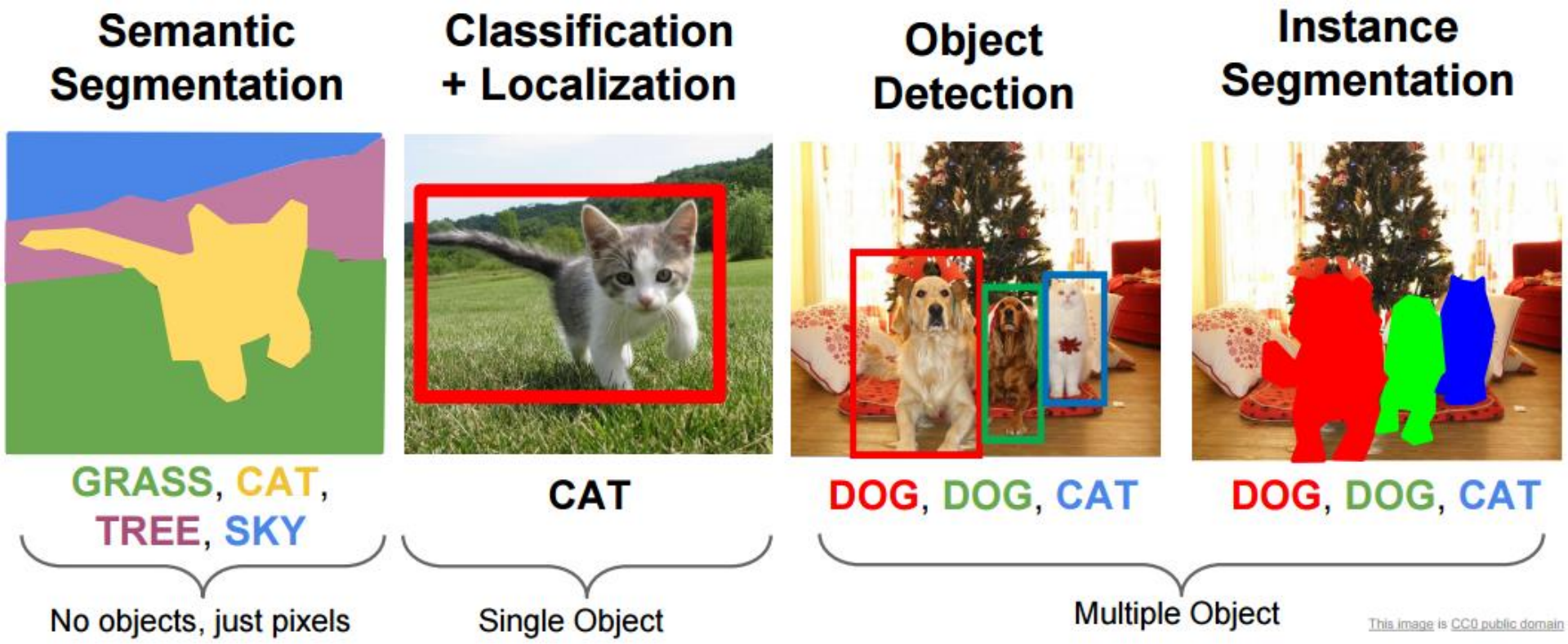
=



**Panoptic Segmentation**

https://github.com/airobotlab/lecture_5_yolov5

# Segmentation

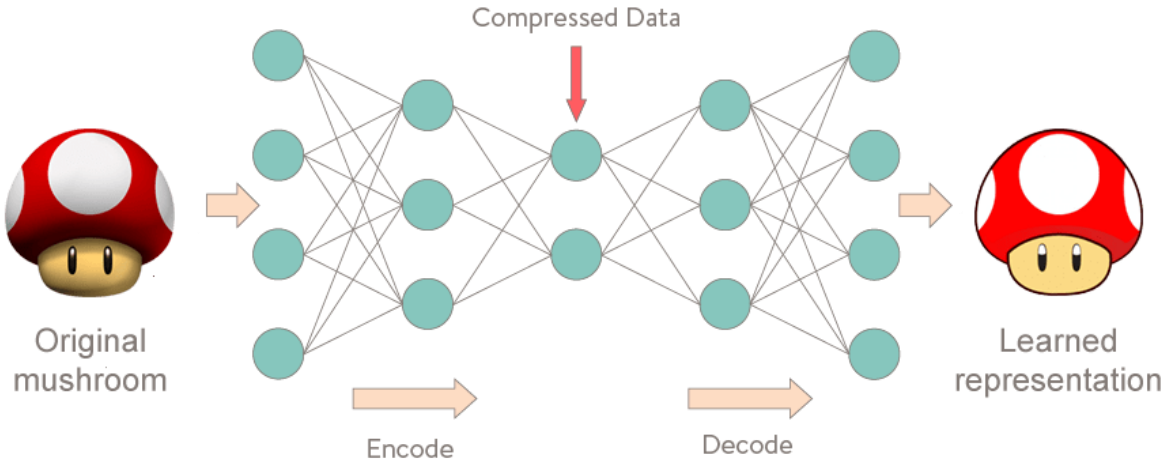


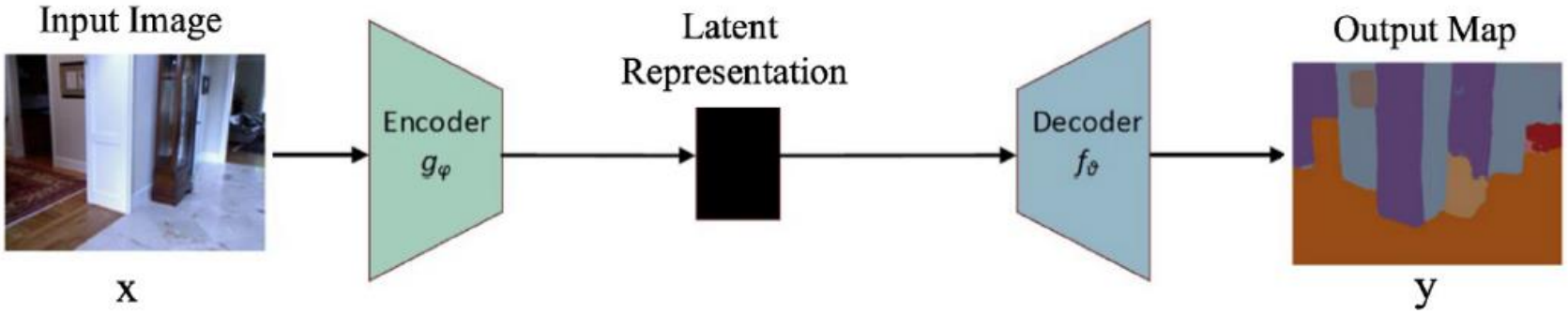



https://medium.com/@ashika.pobbathi/image-segmentation-using-deep-learning-models-and-datasets-98609c02a2b

# Encoder-Decoder & Auto-Encoder Model

- **Encoder to extract input embedding vector**
- **Decoder make a output from embedding vector**




https://medium.com/@ashika.pobbathi/image-segmentation-using-deep-learning-models-and-datasets-98609c02a2b

# Segmentation

- **2D Image Dataset**
  - **PASCAL Visual Object Classes (VOC), 21 classes**
  - **PASCAL Context, 59 classes**
  - **Microsoft Common Objects in Context, 91 classes, 328,000 images**
  - **Cityscapes, 30 classes grouped into 8 categories**

*gtFine_trainvaltest.zip (241MB) [md5]*

fine annotations for train and val sets (3475 annotated images) and dummy annotations (ignore regions) for the test set (1525 images)

*gtCoarse.zip (1.3GB) [md5]*

coarse annotations for train and val set (3475 annotated images) and train_extra (19998 annotated images)

*leftImg8bit_trainvaltest.zip (11GB) [md5]*

left 8-bit images - train, val, and test sets (5000 images)

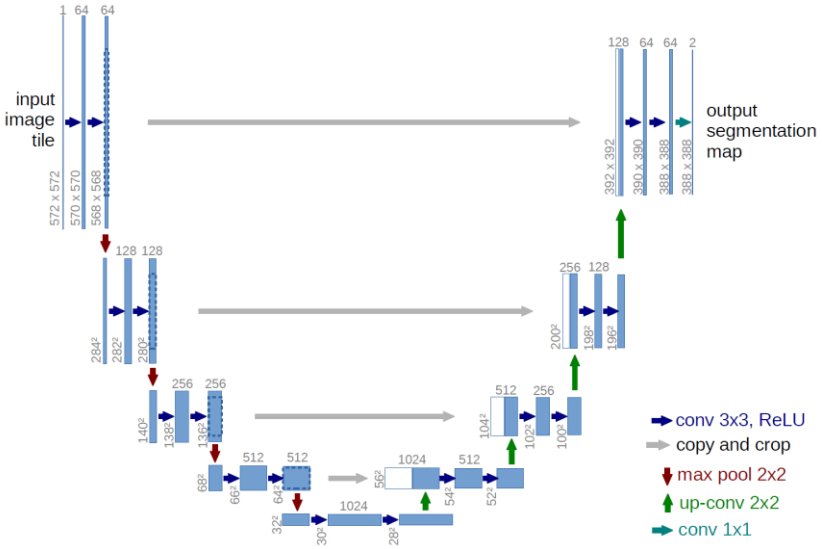
*leftImg8bit_trainextra.zip (44GB) [md5]*

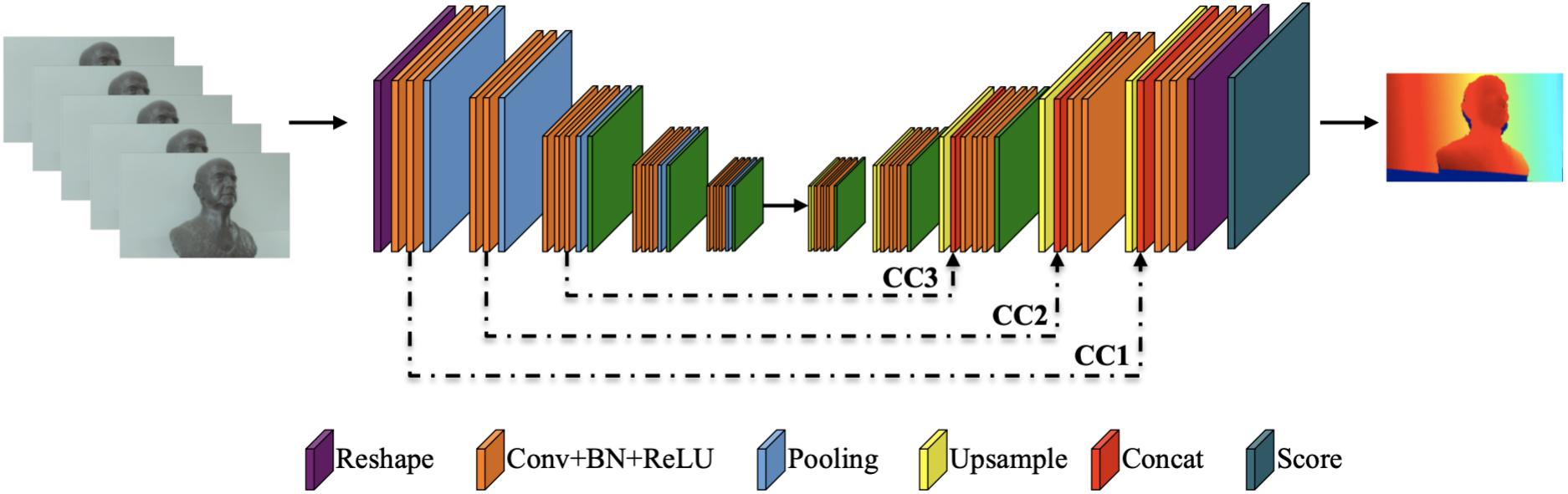
left 8-bit images - trainextra set (19998 images, note that the image "troisdorf_000000_000073_leftImg8bit.png" is corrupt/black)



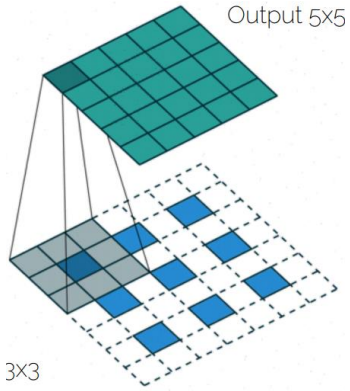
https://www.cityscapes-dataset.com/download

# Unet

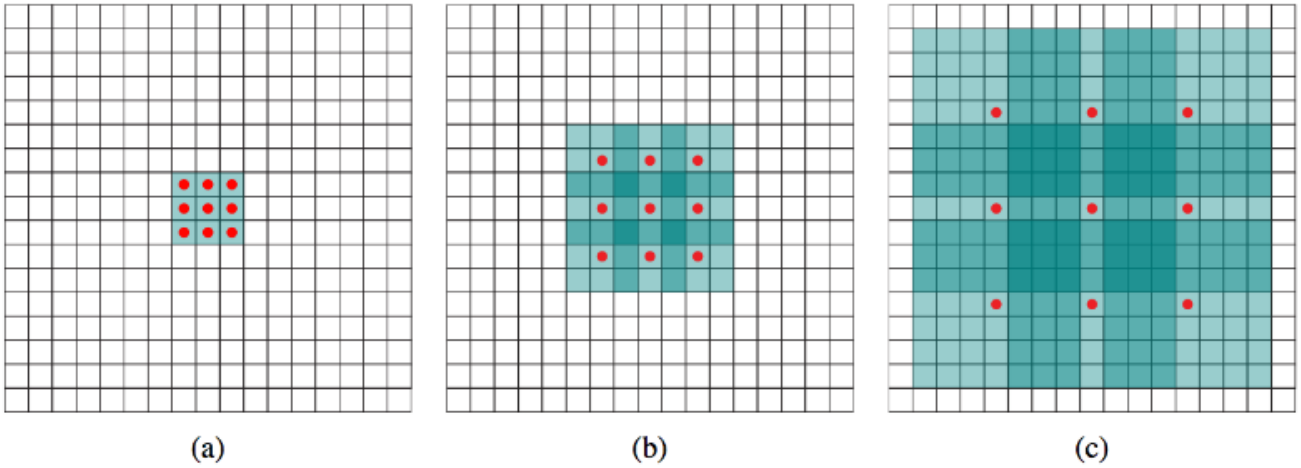

input image tile
output segmentation map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1


CC3
CC2
CC1
Reshape
Conv+BN+ReLU
Pooling
Upsample
Concat
Score

# Upsampling




Transposed Convolution




Dilated Convolutions 2D

# Segmentation dataset format

- Same size, input image – mask with label
- Multi class classification

Input

segmented

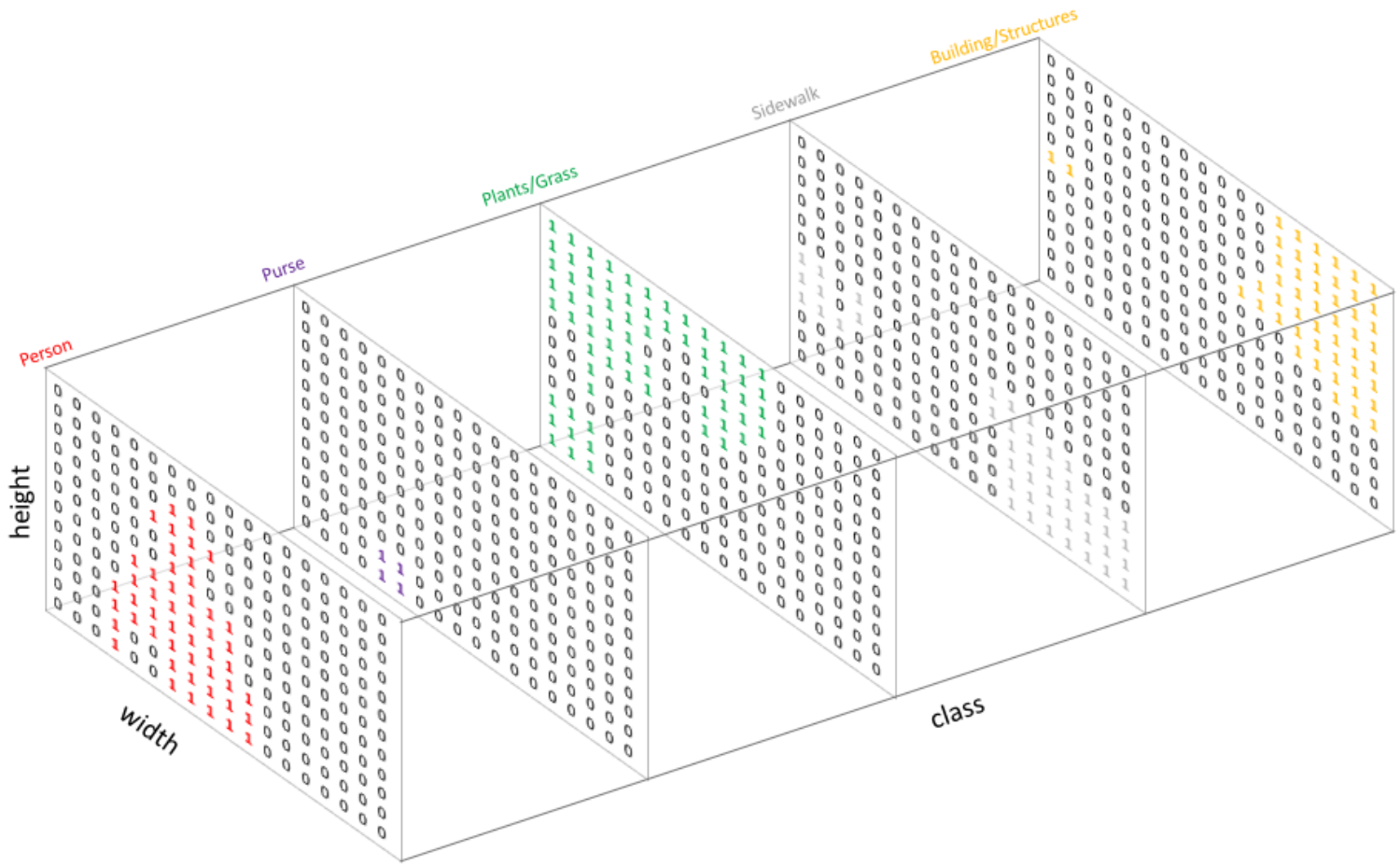
1: Person
2: Purse
3: Plants/Grass
4: Sidewalk
5: Building/Structures

```
3 3 3 3 3 3 3 3 3 3 3 3 5 5 5 5 5 5
3 3 3 3 3 3 3 3 3 3 3 3 5 5 5 5 5 5
3 3 3 3 3 3 1 1 3 3 3 3 5 5 5 5 5 5
3 3 3 3 3 1 1 1 1 3 3 3 5 5 5 5 5 5
3 3 3 3 3 3 1 1 3 3 3 5 5 5 5 5 5 5
5 5 3 3 3 3 1 1 3 3 5 5 5 5 5 5 5 5
4 4 3 4 1 1 1 1 1 1 4 4 4 5 5 5 5 5
4 4 3 4 1 1 1 1 1 1 4 4 4 4 4 5 5 5
4 4 4 1 1 1 1 1 1 1 1 4 4 4 4 4 4 4
3 3 3 1 1 1 1 1 1 1 1 4 4 4 4 4 4 4
3 3 3 1 2 2 1 1 1 1 1 4 4 4 4 4 4 4
3 3 3 1 2 2 1 1 1 1 1 4 4 4 4 4 4 4
```

Semantic Labels

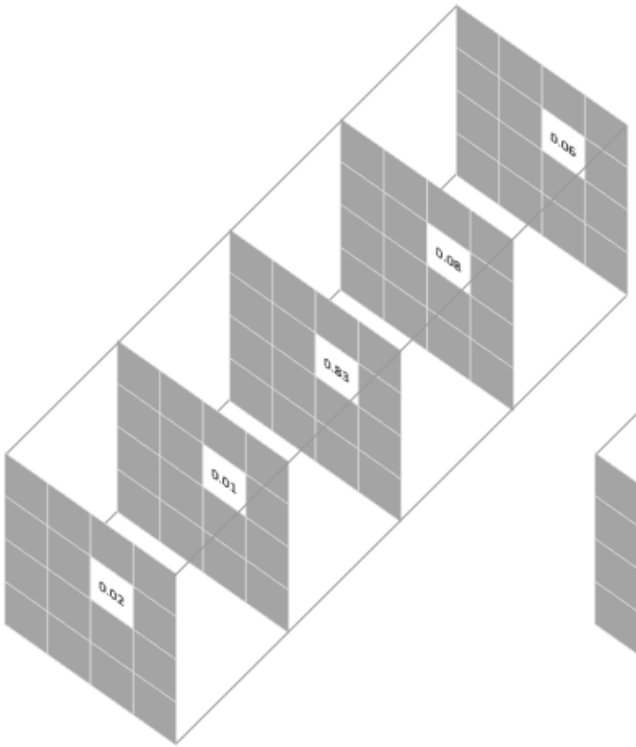
# Object Detection

- **Multi-label classification**

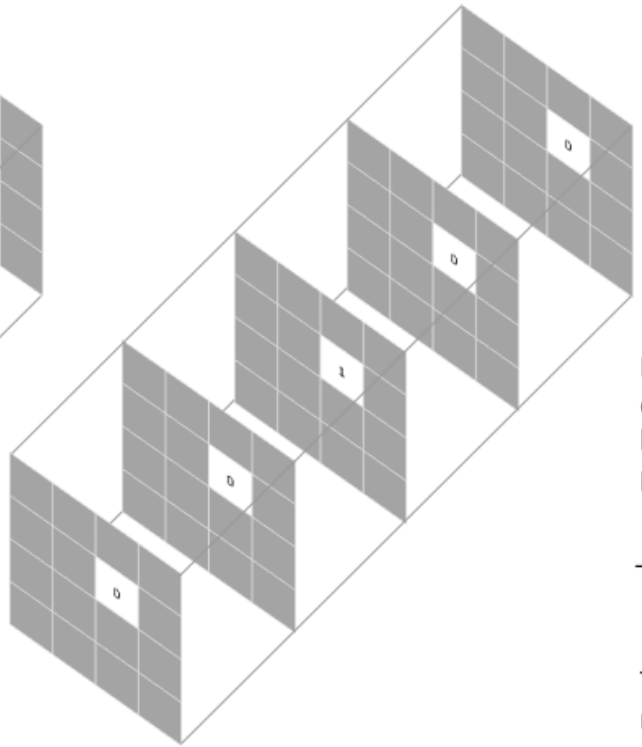


# Object Detection

- **Multi-label classification**




Prediction for a selected pixel




Target for the corresponding pixel

Pixel-wise loss is calculated as the log loss, summed over all possible classes

$$-\sum_{classes} y_{true} \log\left(y_{pred}\right)$$

This scoring is repeated over all **pixels** and averaged

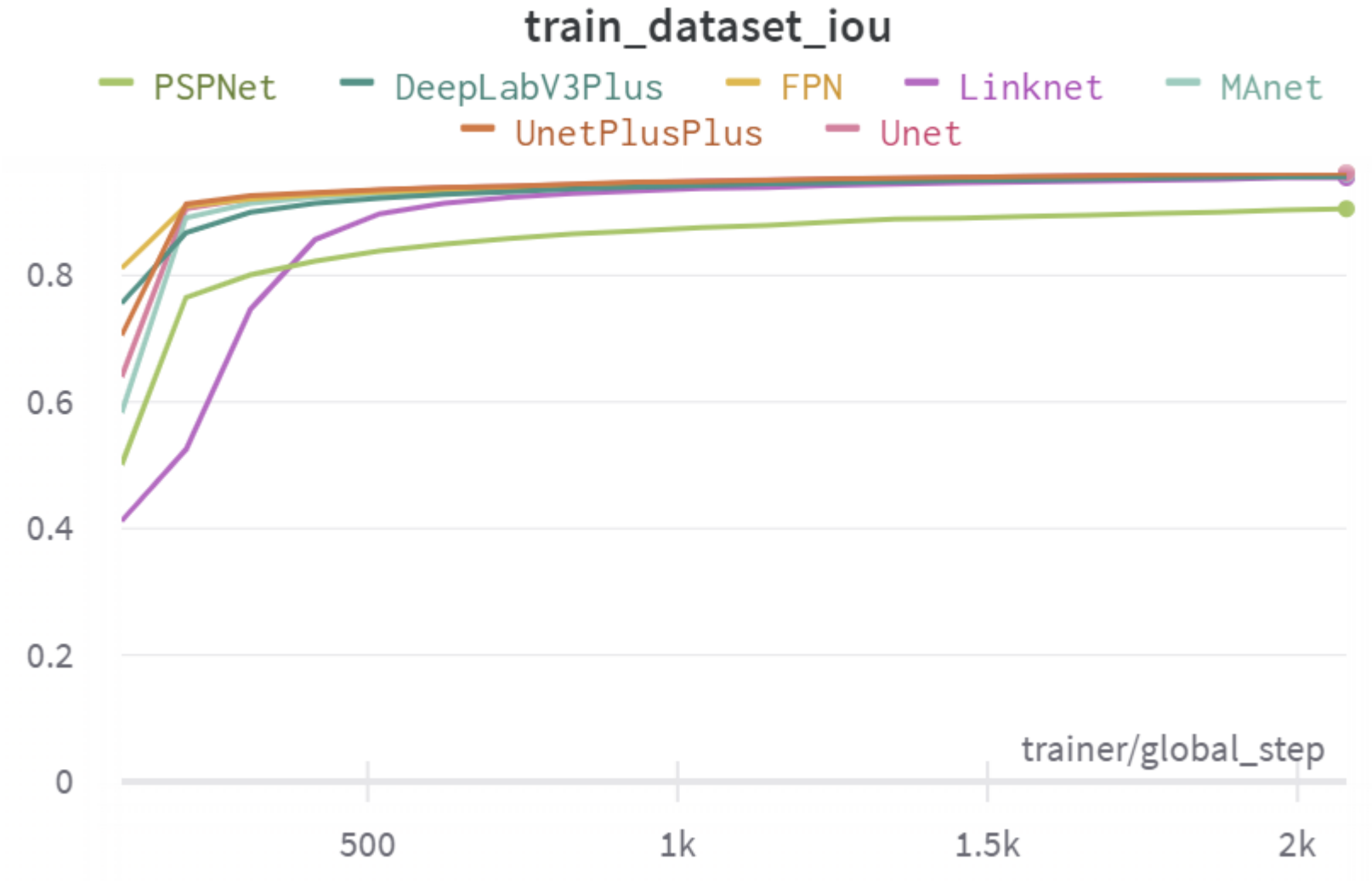
# Experiment

- **Simple Oxford Pet Dataset(3x256x256)**

| | Test | Valid | Traintime (min) /batch | Iteration /s | Parameters(M) | CPU 100 image inference (s) | Batch |
|---|---|---|---|---|---|---|---|
| PSPNet | 87.83 | 87.57 | 1:30 | 4.67 | 28.4 | **10.3** | 32 |
| DeepLabV3 | 91.46 | 91.12 | **23:49** | 2.88 | 33.1 | 40.8 | **16** |
| DeepLabV3+ | 91.34 | 91.19 | **5:07** | 3.43 | 29.5 | 30.2 | 32 |
| FPN | 91.71 | 91.64 | 1:22 | 4.63 | 30.2 | 34.1 | 32 |
| Linknet | 91.65 | 91.42 | 1:22 | 4.74 | 28.7 | 35.0 | 32 |
| MAnet | 92.02 | 91.35 | 1:32 | 4.10 | 38.2 | 36.4 | 32 |
| Unet | 92.05 | **91.88** | 1:35 | 4.27 | 31.2 | 34.8 | 32 |
| Unet++ | 91.75 | 91.78 | 1:48 | 3.71 | 31.9 | 40.0 | 32 |

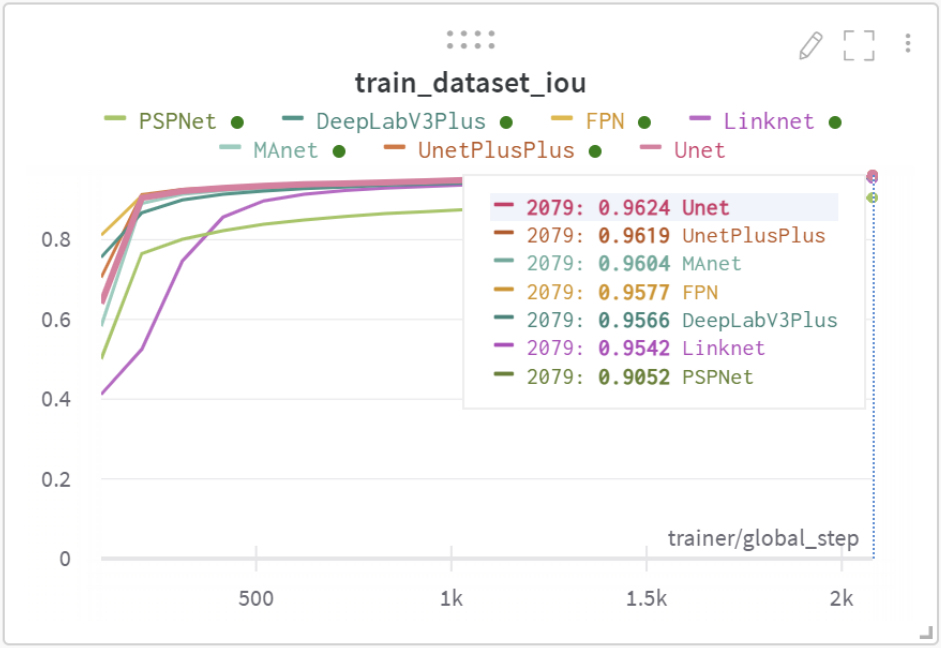
# Experiment

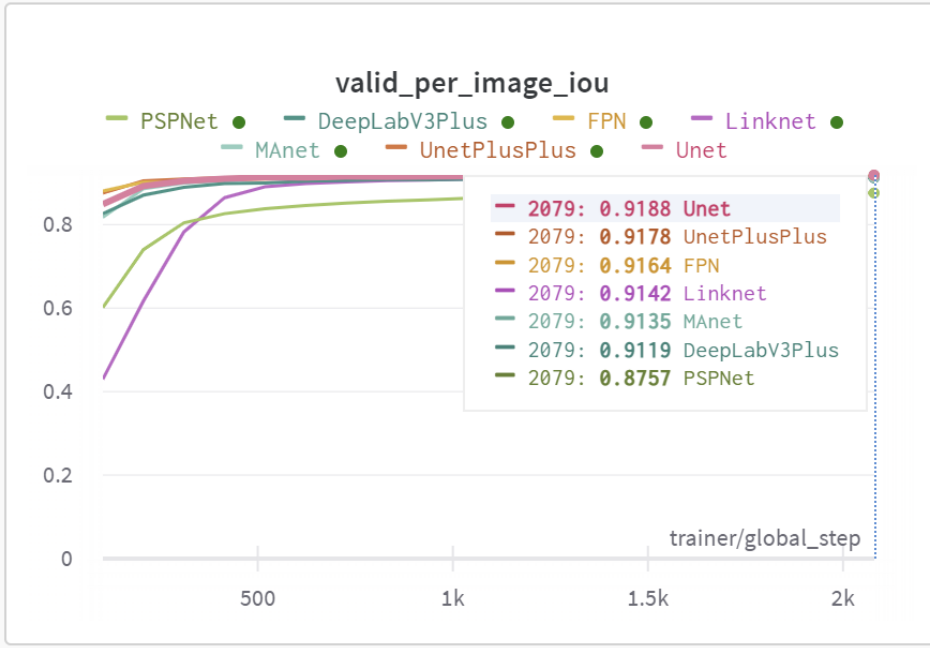
- **Simple Oxford Pet Dataset(3x256x256)**



# Experiment

- **Simple Oxford Pet Dataset(3x256x256)**



# 감사합니다