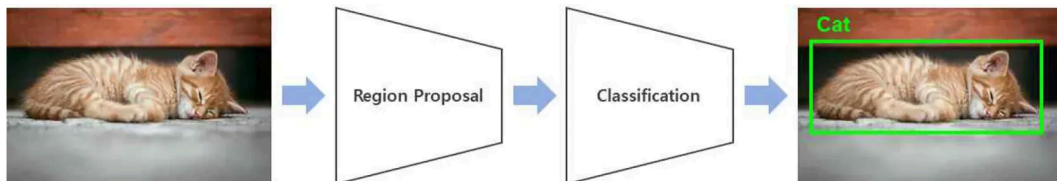


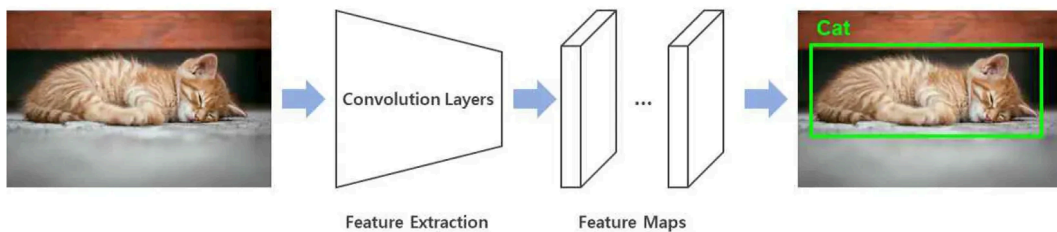
## 4.03 스터디

### 17차시. 객체검출I (Two stage detector)

#### 객체검출 모델 분류



(a) 2-Stage detector



(b) 1-Stage detector

#### Two-Stage Detector

객체 검출이 Region Proposal + Classification 2단계로 이뤄짐

- R-CNN (2014)
- Fast R-CNN
- Faster R-CNN
- Masked R-CNN (Instance segmentation)

#### One-Stage Detector

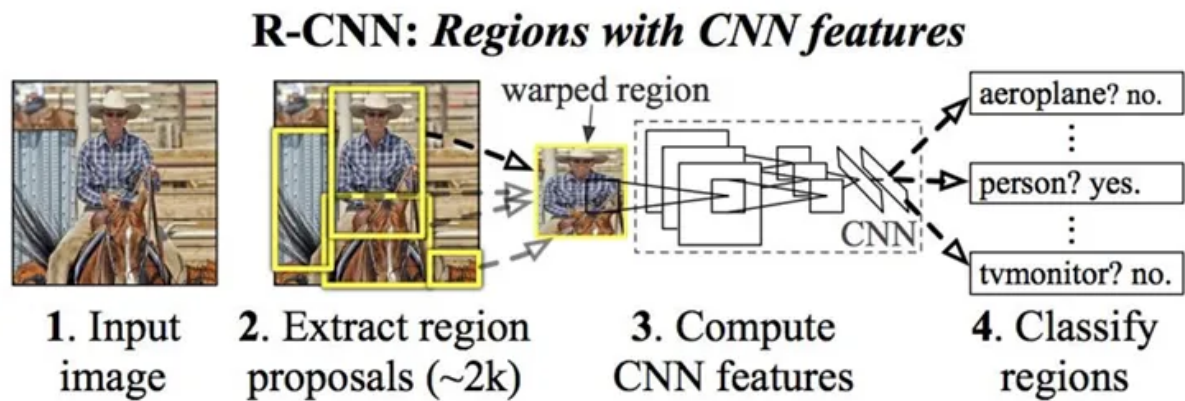
CNN 기반으로 객체 위치를 바로 예측

- Yolo series
  - Yolo1 ~Yolo11 (Object detection)
- SSD series

- SSD
- RetinaNet

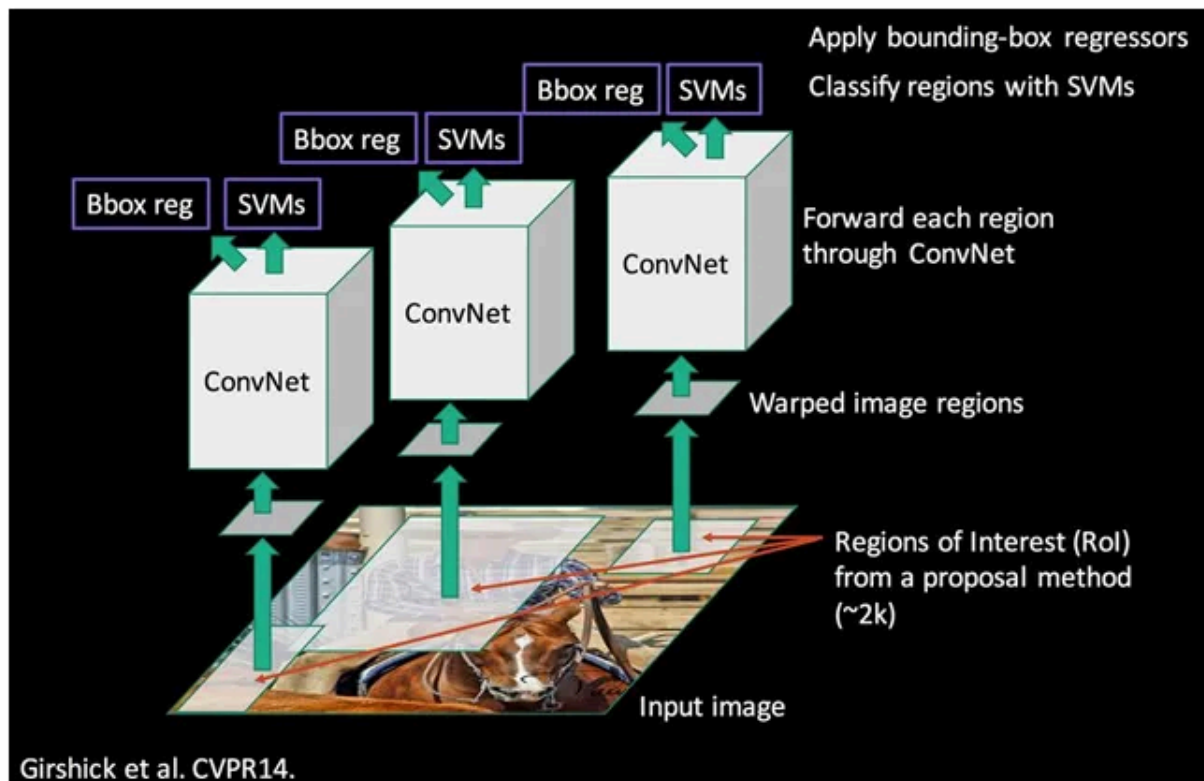
\*최근에는 One-Stage Detector 모델이 정확도와 속도 모두 우수함

## R-CNN




CNN을 이용한 최초의 객체 검출 모델

진행과정 요약:



1. 입력 이미지 (Input Image)
2. RoI 생성 (Regions of Interest from a proposal method ~2k)
3. RoI 크기 변환 (Warped Image Regions)
4. RoI를 ConvNet에 통과 (Forward each region through ConvNet)
5. SVM을 이용한 객체 분류 (Classify regions with SVMs)
6. Bounding Box Regressor를 적용하여 바운딩 박스 보정 (Apply bounding-box regressors)

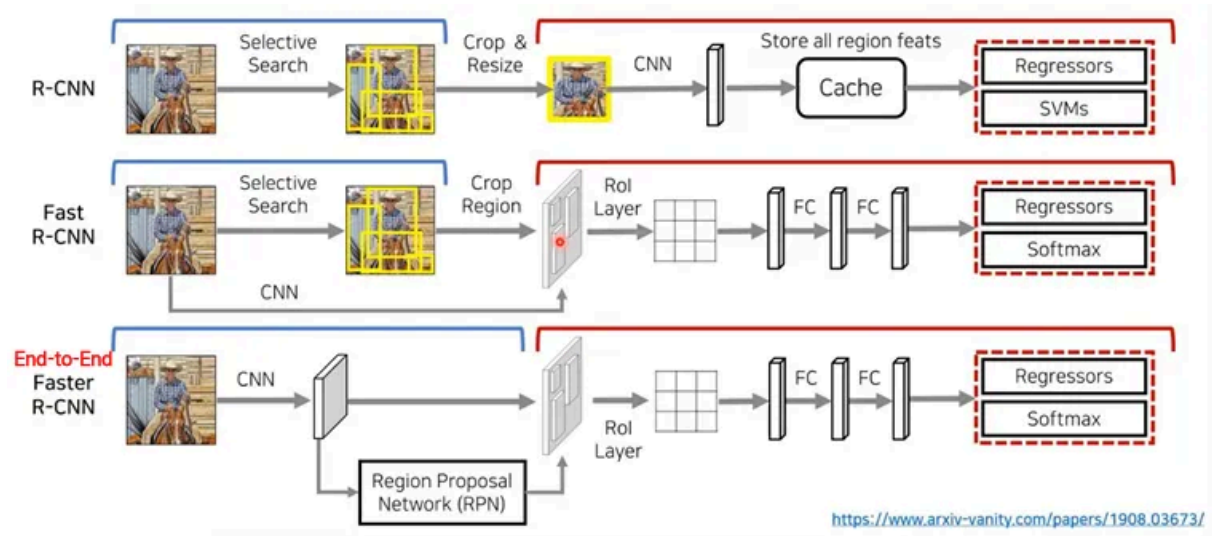
**\*IoU; Intersection over Union**

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$


$$\text{IoU} = \frac{\text{겹치는 영역의 넓이 (Intersection)}}{\text{전체 합집합 영역의 넓이 (Union)}}$$

- NMS 중복 제거, 객체 검출 모델에서 성능 평가를 하는 과정 등에서 사용

## Fast R-CNN & Faster R-CNN



**Fast R-CNN :** 전체 이미지에 대해 CNN을 한 번만 수행하고, ROI Pooling을 통해 각 객체 후보 영역을 공통 feature map에서 효율적으로 추출함 (CNN 연산을 공유한다)

**Faster R-CNN:** Selective Search를 대신하는 RPN

## ROI Pooling (Region of Interest Pooling)

ROI마다 CNN을 다시 수행하지 않고, 한 번 만든 feature map에서 Pooling으로 뽑자

⇒ 다양한 크기의 객체 후보 영역(Region Proposal)을 FC에 입력가능한 고정된 크기로 변환

## RPN

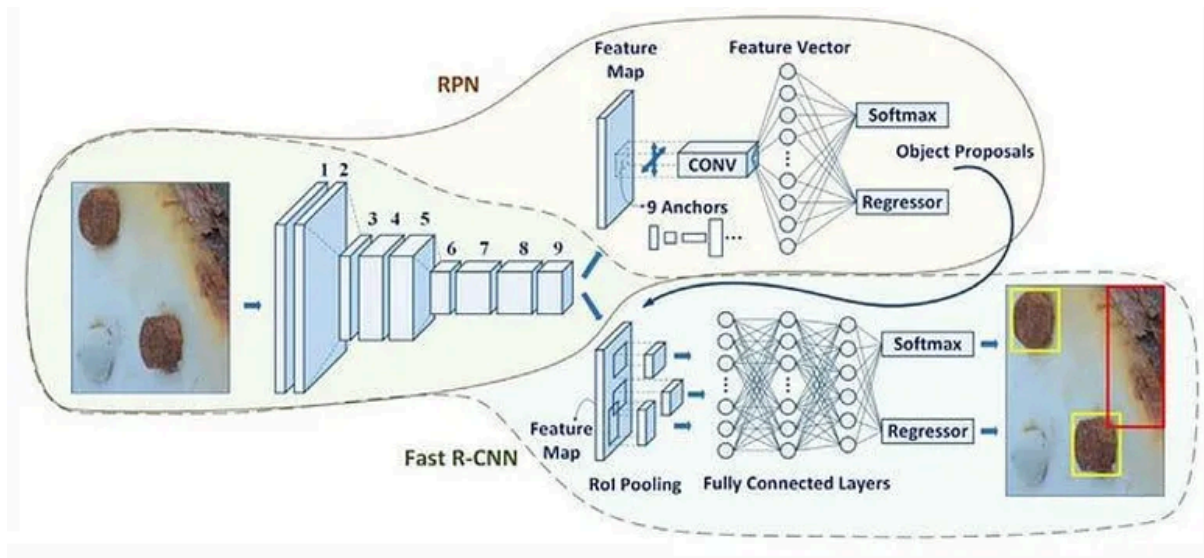
feature map 상에서 anchor들을 기반으로 객체 후보 영역(Region Proposal)을 예측하는 CNN 기반의 네트워크.

기존의 Selective Search에 비해 속도와 정확도 모두에서 크게 향상

### 동작과정

- **Anchor Target Generation**
  - 여러 크기의 Anchor Box를 생성하여 객체 위치 예측
- **GT(Ground Truth) 박스와 IOU(Intersection Over Union) 계산**
  - 실제 객체의 위치와 예측된 Anchor Box 간의 IOU 값을 계산하여 최적의 박스를 선택
- **Proposal 생성**
  - 최적화된 후보 영역을 선별하여 객체 검출 수행

## Faster R-CNN 동작 과정



- CNN을 통해 Feature Map 생성
  - 이미지를 CNN에 입력하여 Feature Map을 추출.
- RPN에서 Region Proposal(ROI) 생성
  - Feature Map을 기반으로 각 위치에서 9개의 Anchor Box를 생성.
  - 각 Anchor Box에 대해:
    - Softmax 분류 → 객체(Positive)인지 배경(Negative)인지 판별.
    - Bounding Box Regression → Anchor Box를 조정하여 더 정확한 위치로 보정.
  - 최종적으로 가장 가능성이 높은 Region Proposal(ROI)들을 선택.
- 선택된 ROI를 Fast R-CNN으로 전달
  - RPN이 선택한 Region Proposal을 RoI Pooling Layer에 전달.
  - RoI Pooling을 통해 고정된 크기의 Feature Map을 생성.
  - 이후 Fully Connected(FC) Layer를 통과하여:
    - Softmax 분류 → 최종 객체 클래스 예측.
    - Bounding Box Regression → 바운딩 박스의 최종 위치 보정.

CNN을 통과한 Feature Map을 공유하면서, 위쪽(RPN)에서 Region Proposal(객체 후보 박스, ROI)을 예측하고 이를 아래쪽 Fast R-CNN으로 전달하여 최종 객체 검출을 수행

## 모델 비교

모델	속도(1장 처리)	특징
R-CNN	~50s	<b>Selective Search</b> SVM으로 분류
Fast R-CNN	~2s	<b>ROI Pooling 도입</b> End-to-End 학습
Faster R-CNN	~0.198ms	<b>RPN 도입</b>

## Faster R-CNN

### 모델 불러오기, 이미지 불러와서 전처리

```
import os
from torchvision.io import read_image
from torchvision import models

# 이미지 로드 및 디바이스로 전송 (예: CUDA 또는 CPU)
data_dir = "./figure"
img_path = os.path.join(data_dir, "dog.jpg")
img = read_image(img_path).to(device)

# Faster R-CNN 가중치 로드
weights = models.detection.FasterRCNN_ResNet50_FPN_Weights.DEFAULT
fasterRCNN = models.detection.fasterrcnn_resnet50_fpn(weights=weights)
fasterRCNN = fasterRCNN.to(device) # 모델도 디바이스로 전송

# COCO 데이터셋 클래스 레이블 리스트
coco_labels_list = weights.meta["categories"]

# 전처리 함수 가져오기
preprocess = weights.transforms()
```

```
# 이미지에 전처리 적용
# 모델 앞단에 GeneralizedRCNNTransform 있어서 불필요
batch_img = preprocess(img)
batch_img = batch_img.unsqueeze(0) # 배치 차원 추가 (1, C, H, W)
```

## 모델 예측

```
import time
fasterRCNN.eval()

start = time.time()
pred = fasterRCNN(batch_img)
stop = time.time()
print(f"estimation time = {(stop - start)*1000:.3f}ms")

pred
```

### (참고) 모델 출력 pred

- pred : 원소 1개인 리스트
- pred[0] : 키-값이 3개 들어있는 딕셔너리
  - 'boxes' : 바운딩박스 좌표 - 좌상단, 우하단
  - 'labels', : 탐지된 객체의 클래스 ID
  - 'scores' : 탐지된 객체의 신뢰도

```
print(type(pred), len(pred))    # <class 'list'> 1
print(type(pred[0]), len(pred[0])) # <class 'dict'> 3
```

## 스코어가 임계값(threshold) 이상인 정보만 남기기



```

## 스코어가 0.7 이상인 박스만 남기기

# 신뢰도 0.7 이상인 객체를 선택하는 Boolean Mask 생성
threshold = 0.7
indices = pred_dict['scores'] >= threshold
print("indices = ", indices)

# 신뢰도 0.7 이상인 박스, 클래스, 신뢰도만 필터링하여 저장
pred_boxes = pred_dict['boxes'][indices]
pred_labels = pred_dict['labels'][indices]
pred_scores = pred_dict['scores'][indices]
print("pred_boxes = \n", pred_boxes)
print("pred_labels = \n", pred_labels)
print("pred_scores = \n", pred_scores)

```

## 박스 그리기

```

import random

image = img.permute(1, 2, 0).cpu().numpy()

# 클래스(91개) 별 바운딩 박스 색깔을 랜덤하게 만들기
color_array = [[random.randint(0, 255) for _ in range(3)] for _ in range(91)]

for i in range(len(pred_boxes)):
    #좌상단
    x_min = int(pred_boxes[i][0])    #rectangle함수에 넣기위해서 정수로 캐스팅
    y_min = int(pred_boxes[i][1])
    #우하단
    x_max = int(pred_boxes[i][2])
    y_max = int(pred_boxes[i][3])

    color = color_array[pred_labels[i]]

    cv2.rectangle(image,

```

```

        pt1=(x_min, y_min),
        pt2=(x_max, y_max),
        color=color,
        thickness=2)

cv2.putText(image,
            text=coco_labels_list[pred_labels[i]] + ' {:.2f}'.format(pred_scores
            org=(x_min + 10, y_min - 10), # 박스 선에 겹치니까 살짝 올려주기
            fontFace=0,
            fontScale=0.8,
            color=color, thickness = 2)

plt.figure(figsize=(8, 6))
plt.imshow(image)
plt.grid(None)
plt.axis("off")
plt.show()

```

## 바운딩 박스 그릴 때 주의점

### OpenCV와 Matplotlib의 색상 표현 방식

- OpenCV : BGR
- Matplotlib : RGB

⇒ plt를 써서 원하는 색깔을 출력하려면 OpenCV 이미지(BGR)를 RGB로 변환 필요

```

import cv2
import matplotlib.pyplot as plt

img_bgr = cv2.imread("dog.jpg")    # OpenCV: BGR
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB) # BGR → RGB

plt.imshow(img_rgb)    # 이제 제대로 된 색상으로 출력

```

```
plt.axis("off")
plt.show()
```

## 의미적 분할 (Semantic segmentation)

### 이미지 출력 함수

```
## draw함수 정의
def show(imgs : list):
    if not isinstance(imgs, list):
        imgs = [imgs]

    fig, axs = plt.subplots(ncols=len(imgs), figsize = (12, 6), squeeze=False) #

    for i, img in enumerate(imgs):
        img = F.to_pil_image(img) # permute dimension
        axs[0, i].imshow(img)
        axs[0, i].axis("off")
```

### (참고) F.to\_pil\_image( )

: Tensor 이미지를 PIL 이미지로 변환 → `matplotlib.pyplot.imshow()` 로 정상적으로 이미지를 출력

[C, H, W] → [H, W, C]

\* `img.detach().permute(1, 2, 0)`를 해도 동일한 결과를 얻지만 `permute( )` 함수에 경우 텐서 이미지가 객체로 유지됨.

즉, `to_pil_image()` 함수는 내부에서

`permute` , `clamp` , `to(uint8)` 등의 처리가 포함된 것

```
img = torch.randn(3, 224, 224) # [C, H, W]

img1 = img.permute(1, 2, 0) # [H, W, C] 텐서
```

```
img2 = to_pil_image(img)      # PIL.Image.Image
```

```
print(type(img1)) # <class 'torch.Tensor'>  
print(type(img2)) # <class 'PIL.Image.Image'>
```

## 이미지 리스트, 모델 로드

```
## 이미지 리스트만들기  
data_dir = "./figure"  
img1 = read_image(os.path.join(data_dir, "dog.jpg"))  
img2 = read_image(os.path.join(data_dir, "peoples.jpg"))  
img_list = [img1, img2]  
  
## Semantic Segmentation 모델 불러오기  
weights = models.segmentation.FCN_ResNet50_Weights.COCO_WITH_VOC_L  
model = models.segmentation.fcn_resnet50(weights=weights, progress=True  
transforms = weights.transforms(resize_size = None)
```

## 모델 추론

```
## 모델 추론  
model.eval()  
output = model(batch)['out']
```

### (참고) model output

```
output.shape # torch.Size([2, 21, 576, 768])
```

2개 이미지, 각 픽셀(576 \* 768)마다 21개(배경1 + 20) 클래스에 대한 예측값( logit )

```
[-1.1796e+00, -1.1796e+00, -1.1796e+00, ..., -9.5306e-01,  
-9.5306e-01, -9.5306e-01]
```

```
[-1.1796e+00, -1.1796e+00, -1.1796e+00, ..., -9.5306e-01,
```

```
-9.5306e-01, -9.5306e-01],  
...
```

## 모델 출력(logit 값) 을 softmax를 통해 확률 분포로 변환

```
normalized_masks = torch.softmax(output, dim=1)
```

## 사람&개 마스크 만들어 출력

```
# 클래스 축에서 가장 확률 높은 클래스 선택  
class_dim = 1  
boolean_dog_masks = (normalized_masks.argmax(dim = class_dim) == sem_c  
                    (normalized_masks.argmax(dim = class_dim) == sem_class_to_idx  
                    )  
  
dogs_with_masks = [  
    draw_segmentation_masks(img, masks=mask, colors= "red", alpha=0.6)  
    for img, mask in zip(img_list, boolean_dog_masks)  
]  
  
show(dogs_with_masks)
```

### \* **draw\_segmentation\_masks** (임포트 함수)

세그멘테이션 마스크를 이미지 위에 색깔로 덧씌워줌

```
from torchvision.utils import draw_segmentation_masks  
  
output_img = draw_segmentation_masks(  
    image: Tensor,          # [C, H, W] 형태의 이미지 텐서  
    masks: Tensor,         # [H, W] or [N, H, W] 형태의 boolean mask  
    colors: str or list,    # 덧씌울 색상 (예: "red", "blue", 또는 RGB 리스트)  
    alpha: float = 0.6     # 마스크의 불투명도 (0: 투명, 1: 완전불투명)  
)
```