

:

TorchVision 객체 검출 모델 성능비교

이름 김준호

국민대학교 전자공학부

jhk00@kookmin.ac.kr

요 약

본 보고서는 TorchVision 라이브러리에 포함된 딥러닝 기반 객체 검출 모델들을 비교 분석하여 그 성능을 평가한다. 보고서에 사용되는 모델들은 fastercnn_resnet50_fpn, fcos_resnet50_fpn, retinanet_resnet50_fpn, ssdlite320_mobilenet_v3_large 모델이다. 또한, 객체 검출 결과인 바운딩 박스와 신뢰도를 사진 위에 오버랩하여 각 모델의 객체 검출 결과를 직관적으로 볼 수 있도록 구현했다. 각 모델간 같은 이미지의 객체를 검출하는 것을 비교함으로써 정확도, 실행시간 등을 비교할 수 있었다.

1. 서론

Faster R-CNN 모델은 객체를 식별하는데 도움을 주는 특징을 추출하고, 사물이 있을 것으로 예상되는 영역을 찾아내어 정확하게 사물을 인식하는 두 단계로 나뉜다. Faster R-CNN의 핵심 아이디어는 Region Proposal Network (RPN)이다. 기존 Fast RCNN 구조를 그대로 계승하면서 selective search를 제거하고 RPN을 통해서 RoI를 계산한다. 이를 통해 GPU를 사용한 RoI 계산이 가능해졌으며, RoI 계산 역시도 학습시켜 정확도를 높일 수 있었다. 아래는 Faster R-CNN의 전반적인 구조이다.

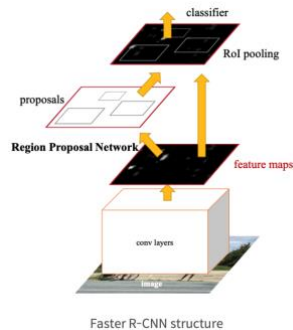


그림 1. Faster R-CNN

FCOS(Fully Convolutional One-Stage Object Detection) 모델은 fully convolutional 네트워크를 사용하여 객체 검출 작업을 수행하는 딥러닝 모델이다. 이 모델은 객체의 위치를 예측하기 위해 특징맵을 사용한다. 또한 한 번의 순전파로 객체 검출을 수행하는 단일 단계 검출 방법이기 때문에 모델이 간단하고 빠르게 실행될 수 있도록 도와준다. FCOS은 anchor-box free한 detector를 제안한다. Anchor box를 제거한

detector에서는 low recall rate, ambiguous sample 문제가 발생할 수 있는데 이런 문제를 해결할 수 있는 구조를 제안하고 해당 구조에 대한 효율성을 실험을 통해 입증했다.

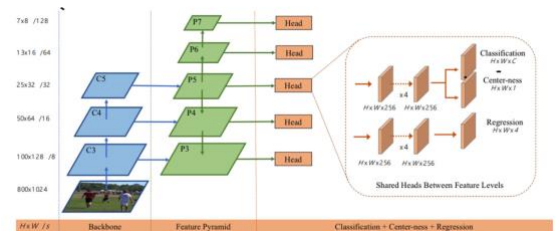


그림 2. FCOS

RetinaNet은 피라미드 형태의 다양한 크기의 특징 맵을 생성하기 위해 Feature Pyramid Network (FPN)을 사용한다. 이는 다양한 객체 크기를 처리할 수 있는 효과적이다. 또한 객체의 위치를 예측하기 위해 여러 개의 앵커 박스를 사용합니다. RetinaNet은 두 개의 별도의 네트워크 브랜치를 가지고 있습니다. 하나는 객체의 클래스를 분류하기 위한 Classification Head이고, 다른 하나는 객체의 경계 상자를 조정하기 위한 Regression Head이다. 이 두 브랜치를 통해 객체의 위치와 클래스를 예측한다. RetinaNet은 FPN과 앵커 박스를 사용하여 다양한 크기의 객체를 처리할 수 있습니다. 다중 스케일 특징과 앵커 박스를 활용하여 객체를 정확하게 검출할 수 있습니다. 이는 높은 정밀도와 작은 객체에 대한 좋은 검출 성능을 제공한다. 하지만 단점으로는 객체의 위치를 예측하기 위해 여러 개의 앵커 박스를 사용하므로 계산 비용이 높을 수 있다.

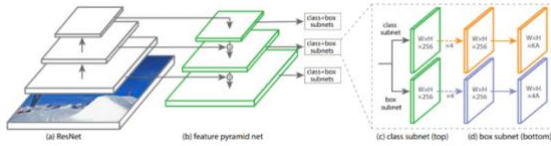


그림 3. RetinaNet

SSD (Single Shot Multibox Detector)는 기존 R-CNN, YOLO의 구조를 활용하여 구성되어 있습니다. 앞서 지금까지 사용한 모델들은 전부 2-stage model 인데 SSD는 1-stage model 이기 때문에 속도가 빠른 대신 정확도가 낮다. SSD는 객체의 위치와 클래스를 예측하기 위해 앵커 박스라는 사전 정의된 박스를 사용합니다. 이 앵커 박스는 다양한 크기와 모양을 가지고 있어서 이미지의 다양한 위치에 적용될 수 있습니다. 이를 통해 다양한 크기와 모양의 객체를 검출할 수 있습니다. SSD의 장점은 한 번의 실행으로 객체를 검출할 수 있어 속도가 빠르다. 하지만 작은 객체의 검출에는 다소 어려움을 겪을 수 있다. 작은 객체는 하위 수준의 단계에서 검출되기 때문에 정확성이 낮을 수 있다.

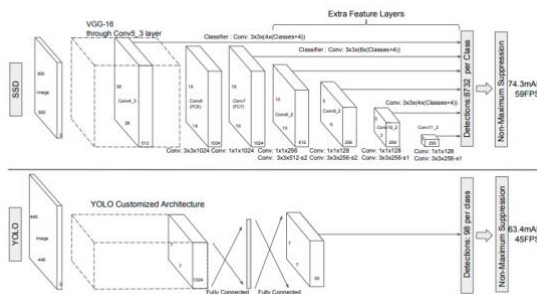


그림 4. SSD

2. 과제 수행 내용

2.1 PIL 라이브러리를 사용하여 사진 가져오기

Image.open()함수를 사용하여 각 모델마다 사용할 이미지를 지정해준다. 딥러닝 모델은 일반적으로 텐서 형식의 데이터를 입력으로 받기 때문에, PIL 이미지를 텐서로 변환해야 한다. ToTensor() 변환은 이미지를 0에서 1 사이의 값을 가지는 픽셀 값으로 변환하고, 이미지 채널 순서를 RGB에서 C x H x W로 변경한다. 이렇게 변환된 텐서 형식의 이미지는 딥러닝 모델에 입력으로 사용될 수 있다. 이렇게 전처리된 이미지(텐서)들을 각 딥러닝 모델마다 생성해준다.

2.2 PyTorch의 TorchVision 모듈에 있는 다양한

객체 인식 모델 사용

```
from torchvision.models.detection
fasterrcnn_resnet50_fpn, fcos_resnet50_fpn,
```

retinanet_resnet50_fpn, ssdlite320_mobilenet_v3_large 을 import 해줌으로써 딥러닝 모델들을 가져와 사용할 수 있다. fasterrcnn_resnet50_fpn(pretrained=True) 같이 인자로 pretrained=True 를 줌으로써 이미 학습된 모델을 사용한다. 또한 .eval() 메소드를 사용함으로써 사용하는 딥러닝 모델을 추론 모드로 전환한다. 추론 모드에서는 모델의 가중치 업데이트 및 그래디언트 계산이 비활성화된다. 이를 통해 주어진 입력에 대해 예측만 생성하도록 한다.

2.2 학습된 모델을 이용해 이미지의 객체 검출

이미지를 검출하는 단계에서 각 모델들의 속도 차이를 확인하기 위해 start_time=time.time(), end_time=time.time()으로 추론을 하기전과 한 후의 시간을 측정해준다. 모델을 이용해 추론을 진행할 때 with torch.no_grad():을 사용하는데 모델이 가중치 업데이트를 수행하지 않고, 그래디언트 계산을 비활성화하여 메모리를 절약할 수 있도록 한다. 과제 수행 내용 2.2 에서 만든 딥러닝 모델에 model_name([image_tensor]) 전처리된 이미지 텐서를 입력으로 준다. 이후 추론 결과들을 model_prediction 에 넣어준다.

2.4 객체 검출 결과를 이미지에 표시

먼저 신뢰도에 따라 객체를 검출하는 기준인 임계값을 지정해준다. ImageDraw.Draw()을 사용해 이미지 위에 바운딩 박스와 신뢰도를 표시할 수 있는 객체를 만든다. 이후 임계값을 넘는 신뢰도를 포함하는 정보들인 검출된 객체의 바운딩 박스 정보와 검출 신뢰도를 이미지 위에 그린다. 이후 표시된 검출 결과를 .show() 메소드를 통해 직접 볼 수 있도록 한다.

3. 실험 결과 및 분석

신뢰도 임계값을 0.7 로 두고 Faster Rcn, FCOS, Retinanet, SSD 딥러닝 모델로 객체 검출한 결과는 아래 그림 5, 6, 7, 8 과 같다.



그림 5. Faster Rcnnc

그림 6. FCOS



그림 7. Retinanet

그림 8. SSD

결련 시간 (초)	FasterRcnn	FCOS	Retinanet	SSD
	1.2789	0.8721	0.9356	0.1113
	1.1974	0.9325	0.9408	0.1125
	1.1129	0.8714	0.9330	0.1039
	1.1615	0.9052	1.1049	0.1101
	1.2486	0.9687	0.9512	0.1053
평균 시간	1.19986	0.90998	0.9731	0.10862

1.

그림 5 를 보면 Faster Rcnnc 만이 bowl 클래스를 검출했고 cat 의 신뢰도가 1.0 인 것을 보면 Faster Rcnnc 이 정확도 측면에서는 꽤 성능이 좋은 것을 알 수 있다. 또한 이 것을 통해 FCOS, Retinanet, SSD 는 가려진 객체 또는 작은 객체를 잘 구분하지 못하는 성질을 알 수 있다. 하지만 Faster Rcnnc 의 그림 5 에서 바운딩 박스를 보면 고양이의 오른쪽 귀부분이 살짝 넘어간 것을 볼 수 있다. 그림 6 을 보면 FCOS 는 cat 의 신뢰도가 0.73 인 것에 반해 바운딩 박스는 잘 잡힌 것을 알 수 있다. Retinanet 과 SSD 를 사용한 그림 7 과 8 을 보면 신뢰도도 높고 바운딩 박스도 잘 잡힌 것을 알 수 있다.

표 1 을 보면 **Faster Rcn**n 이 가장 느린데 정확도가 높은 대신 속도가 다른 딥러닝 모델에 비해 상대적으로 느린 것을 알 수 있다. 이 표 중에 주목되는 것은 **SSD** 의 평균 속도인데 다른 딥러닝 모델들보다 약 9 배에서 11 배 정도 빠른 것을 알 수 있다. **Cat** 의 신뢰도도 0.98 으로서 큰 객체를 검출 할 때 꽤 정확한 신뢰도와 빠른 속도를 보장해준다는 것을 알 수 있다. **FCOS** 와 **Retinanet** 은 속도가 서로 비슷하지만 이번 이미지에서는 **Retinanet** 의 cat 의 신뢰도가 더 높다는 것을 보아 정확도가 더 높다는 것을 알 수 있다.

신뢰도 임계값을 0.5 로 두고 Faster Rcn, FCOS, Retinanet, SSD 딥러닝 모델로 객체 검출한 결과는 아래 그림 9, 10, 11, 12 와 같다.

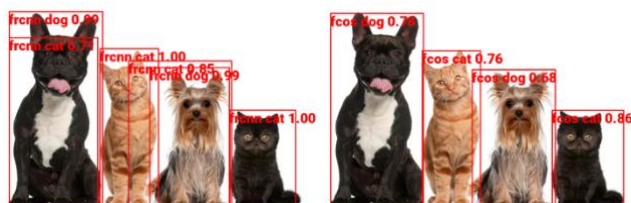


그림 9. Faster Rcnnc

그림 10. FCOS

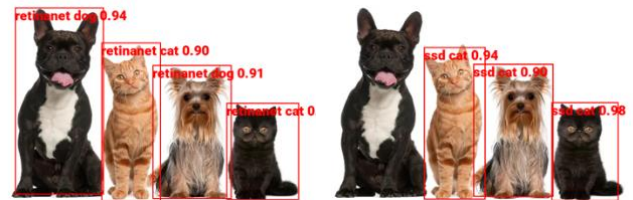


그림 11. Retinanet

그림 12. SSD

결린 시간 (초)	FasterRcnn	FCOS	Retinanet	SSD
	1.2817	1.0976	1.1662	0.1029
	1.2339	1.1037	1.1631	0.1005
	1.2182	1.0906	1.1553	0.0993
	1.2959	1.0843	1.1664	0.0963
	1.3031	1.1449	1.1971	0.1011
평균 시간	1.26656	1.10422	1.16962	0.10002

丑 2.

그림 9 를 보면 **Faster Rcn** 이 검출하는 정확도가 높아 4 개의 **cat** 과 **dog** 클래스의 객체를 잘 검출했다. 하지만 이 때문에 오히려 **dog** 클래스의 객체를 구할 때 **cat** 클래스도 겹쳐서 구해지는 오류를 볼 수 있다. 표 2 를 보면 **Faster Rcn** 이 평균 검출 시간이 제일 높은 것을 알 수 있다. 그림 10 을 보면 **FCOS** 는 비록 신뢰도는 낮게 검출했지만 **cat** 과 **dog** 클래스의 객체를 잘 검출한 것을 알 수 있다. 그림 11 은 **Retinanet** 으로 추론한 영상인데 객체 전부 비교적 높은 신뢰도로 잘 검출 된 것을 볼 수 있다. 맨 오른쪽 가려진 신뢰도는 0.90 이다. 앞서 그림 7 로 확인했듯이 크기가 큰 객체 검출을 잘하는 것을 알 수 있다. 그림 12 를 보면 **SSD** 모델이 왼쪽 검은색 강아지를 검출하지 못한 것을 볼 수 있는데 1-stage detect 특성상 속도는 빠르지만 정확도가 떨어지는 것을 알 수 있다. 그렇듯 표 2 를 보면 **SSD** 모델 검출시간이 다른 딥러닝 모델보다 약 11 배에서 12 배정도 빠른 것을 알 수 있다.

신뢰도 임계값을 0.7 로 두고 Faster Rcn, FCOS, Retinanet, SSD 딥러닝 모델로 객체 검출한 결과는 아래 그림 13, 14, 15, 16 와 같다.

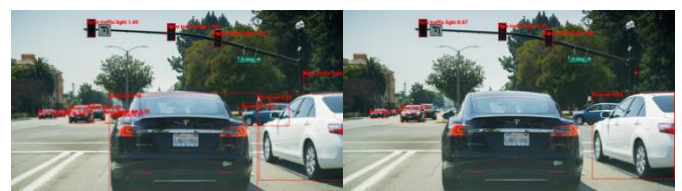


그림 13. Faster Rcnnc

그림 14. FCOS

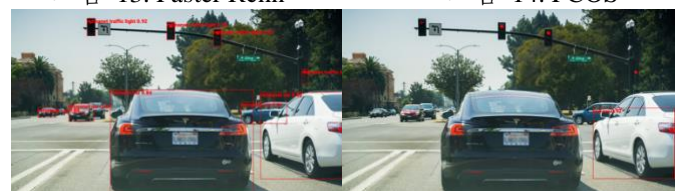


그림 15. Retinanet.

그림 16. SSD

:

결린 시간 (초)	FasterRcnn	FCOS	Retinanet	SSD
	1.3337	1.0958	1.2002	0.1107
	1.3566	1.0952	1.2081	0.1048
	1.3398	1.1216	1.2308	0.1084
	1.3640	1.1228	1.2227	0.1162
	1.3295	1.1921	1.2726	0.1104
평균 시간	1.34472	1.1255	1.22688	0.1101

표 3.

4. 결론

본 보고서에서 TorchVision 라이브러리의 네 가지 딥러닝 객체 검출 모델 frcnn_resnet50_fpn, fcos_resnet50_fpn, retinanet_resnet50_fpn, ssdlite320_mobilenet_v3_large 을 비교하였다. 첫째로, 속도 측면에서는 실험 결과, ssdlite320_mobilenet_v3_large 모델이 가장 빠른 속도를 보였으며, frcnn_resnet50_fpn 모델이 상대적으로 느린 속도를 가지고 있었다. 둘째로, 정확성 측면에서는 실험 결과 frcnn_resnet50_fpn 과 retinanet_resnet50_fpn 모델이 높은 정확도를 달성하였다. 실험 결과를 종합하면, retinanet_resnet50_fpn 모델이 TorchVision 라이브러리의 다른 객체 검출 모델들보다 우수한 성능을 보였다. 객체 오검출도 적었고 알맞게 검출한 객체들의 신뢰도도 높게 나왔다. 이를 통해 실시간 객체 검출이 아니라면 retinanet_resnet50_fpn 다양한 응용 분야에 적용할 수 있는 강력한 모델임을 확인할 수 있었다. 또한 ssdlite320_mobilenet_v3_large 은 객체 검출이 정확하진 않지만 속도가 매우 빨라 실시간으로 객체 검출하는 분야에 적용할 수 있는 모델임을 확인할 수 있었다. 결론으로는 특정 응용 분야에 따라 다른 모델들도 선택할 수 있는 장점이 있기에 실제 사용 시에는 응용 분야와 요구 사항을 고려하여 적합한 모델을 선택하는 것이 중요하다.

참고문헌

- [1] 염창동형준킴 블로그, 갈아먹는 Object Detection [4] Faster R-CNN, 2020.01.07 게시, 2023.06.06 조회, <https://yeomko.tistory.com/17>
- [2] Pajamacoder 블로그, [ObjectDetection] FCOS(Fully Convolution One-Stage Detection) 논문 리뷰, 2021.05.31 게시, 2023.06.06 조회, <https://pajamacoder.tistory.com/7>
- [3] Talktato 블로그, [논문 리뷰] (RetinaNet) Focal Loss for Dense Object Detection, 2022.07.1 게시, 2023.06.06 조회, <https://talktato.tistory.com/13>
- [4] rcchun 블로그, [딥러닝] SSD : Single Shot Multibox Detector, 2021.11.26 게시, 2023.06.06 조회, <https://velog.io/@rcchun/딥러닝-SSD-Single-Shot-Multibox-Detector>

:

ChatGPT

- 프롬프트 1

Write code that does the following:

1. loads an image using PIL
2. loads pre-trained object detection model using the TorchVision module of PyTorch
3. runs object detection using the pre-trained model on the loaded image
4. displays the object detection results (bounding box, object label, confidence value) overlayed on the loaded image
5. if there are too many detection boxes in the outcome, apply thresholding on the detection confidence to show only high-confidence boxes

- 결과 1

```
import torch
import torchvision.transforms as transforms
from PIL import Image, ImageDraw
from torchvision.models.detection import fasterrcnn_resnet50_fpn

# Load image using PIL
image_path = "path/to/your/image.jpg"
image = Image.open(image_path)

# Load pre-trained object detection model
model = fasterrcnn_resnet50_fpn(pretrained=True)
model.eval()

# Apply transformations to the image
transform = transforms.Compose([transforms.ToTensor()])
image_tensor = transform(image)

# Run object detection on the image
with torch.no_grad():
    predictions = model([image_tensor])

# Display object detection results
threshold = 0.5 # Confidence threshold for showing detection boxes
draw = ImageDraw.Draw(image)
for i in range(len(predictions[0]['scores'])):
    if predictions[0]['scores'][i] > threshold:
        bbox = predictions[0]['boxes'][i]
        label = predictions[0]['labels'][i]
        confidence = predictions[0]['scores'][i]

        draw.rectangle([bbox[0], bbox[1], bbox[2],
                        bbox[3]], outline="red")
        draw.text((bbox[0], bbox[1], f"{label}
{confidence}"), fill="red")

image.show()
```

- 프롬프트 2

let me know all models from object detection model using the TorchVision module of PyTorch

- 결과 2

Here are some commonly used object detection models available in the TorchVision module of PyTorch:

1. Faster R-CNN:

- `fasterrcnn_resnet50_fpn`: Faster R-CNN with a ResNet-50 backbone and Feature Pyramid Network (FPN).
- `fasterrcnn_mobilenet_v3_large_fpn`: Faster R-CNN with a MobileNetV3-Large backbone and FPN.
- `fasterrcnn_mobilenet_v3_large_320_fpn`: Faster R-CNN with a MobileNetV3-Large (width 0.8) backbone and FPN.

2. Mask R-CNN:

- `maskrcnn_resnet50_fpn`: Mask R-CNN with a ResNet-50 backbone and FPN.
- `maskrcnn_mobilenet_v3_large_fpn`: Mask R-CNN with a MobileNetV3-Large backbone and FPN.

3. RetinaNet:

- `retinanet_resnet50_fpn`: RetinaNet with a ResNet-50 backbone and FPN.
- `retinanet_mobilenet_v3_large_fpn`: RetinaNet with a MobileNetV3-Large backbone and FPN.

4. SSD (Single Shot MultiBox Detector):

- `ssdlite320_mobilenet_v3_large`: SSD with a MobileNetV3-Large backbone.
- `ssdlite320_mobilenet_v2`: SSD with a MobileNetV2 backbone.
- `ssd300_vgg16`: SSD with a VGG16 backbone.

5. YOLO (You Only Look Once):

- `yolov5s`: YOLOv5 with a small model size.
- `yolov5m`: YOLOv5 with a medium model size.
- `yolov5l`: YOLOv5 with a large model size.
- `yolov5x`: YOLOv5 with an extra-large model size.

These models differ in terms of their backbone architecture, feature extraction methods, and model size. Each model has its own strengths and trade-offs in terms of speed, accuracy, and resource requirements. You can choose the model that best suits your specific use case based on these factors.

Remember to check the TorchVision documentation for the most up-to-date list of available object detection models and their specific configurations.