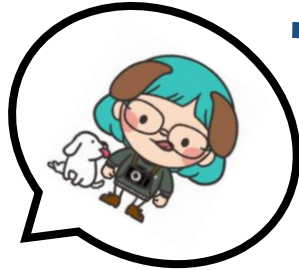




Jo Min Jung

Sophie



 **Github:** <https://github.com/happyBeagle>

 **E-mail:** hack2kong@gmail.com

 **Phone:** +82)10-6830-6667

Contents



```
graph LR; Contents[Contents] --- Introduce[Introduce Myself]; Contents --- Project1[Project: Model Compression]; Contents --- Project2[Project: Neural Architecture Search]; Contents --- Project3[Project: Small Object Detection]; Project1 --- Pytorch[- Pytorch to TensorRT]; Project1 --- Quantization[- Quantization];
```

Introduce Myself

Project: Model Compression

- Pytorch to TensorRT

- Quantization

Project: Neural Architecture Search


Project: Small Object Detection


Introduce Myself

Jo Min Jung (Sophie)



 **Github:** <https://github.com/happyBeagle>

 **E-mail:** hack2kong@gmail.com

 **Phone:** +82)10-6830-6667

Hobbies

Playing Game

Embroidery

Cooking

Interests

DL/ML

Model Compression

Computer Vision

Skills

Python

C++/C

JAVA

PyTorch

ROS

CUDA C/C++

Linux

Windows

GIT

Introduce Myself

Jo Min Jung (Sophie)



Aducation: Pusan National University

Bachelor *2014.03 ~ 2019.02*

Major Computer Science & Engineering



Careers: N Tech Service

Intern *2019.07 ~ 2019.08*

SW Engineer *web front-end & back-end*

Upstage

Intern *2021.08 ~ 2021.11*

SW Engineer & AI Researcher

OCR Model Lightweight

- *Pytorch to TensorRT*

- *Quantization*

Develop OCR Serializer

Introduce **Myself**

Jo Min Jung (Sophie)



Experience: **Boost Camp AI Tech 1st**

2021.01 ~ 2021.06

Study basic AI knowledge

Develop with Google 2nd

2018.01 ~ 2018.02

Learn basic SW develop knowledge

Best of the Best Vulnerability Analysis Track 5th

2016.07 ~ 2017.03

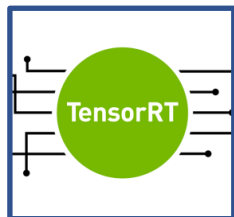
Learn how to analyze vulnerabilities

#HITCON 2018

Project: Model Compression

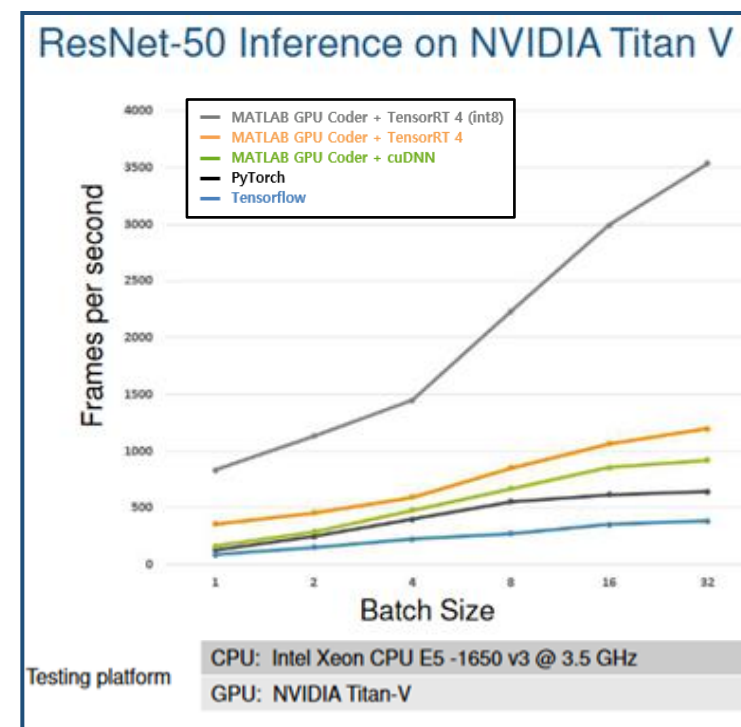
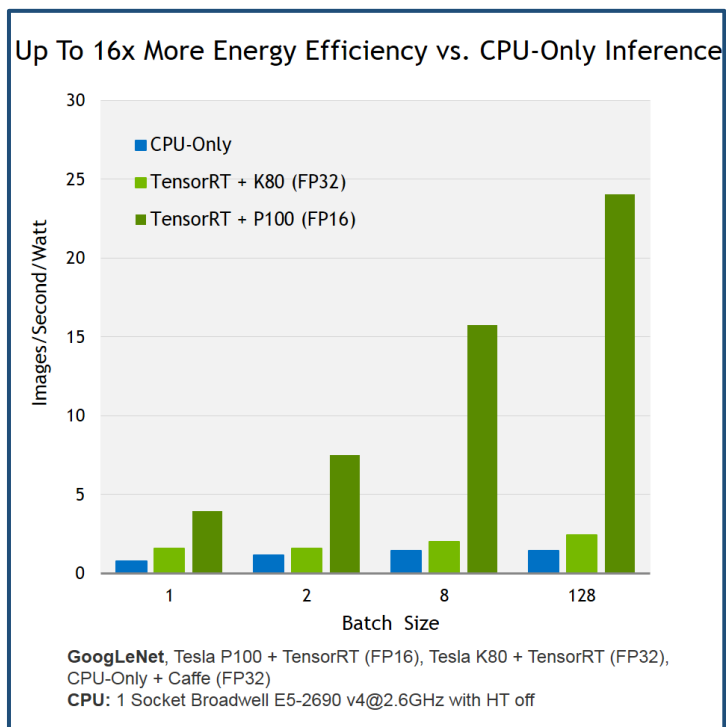
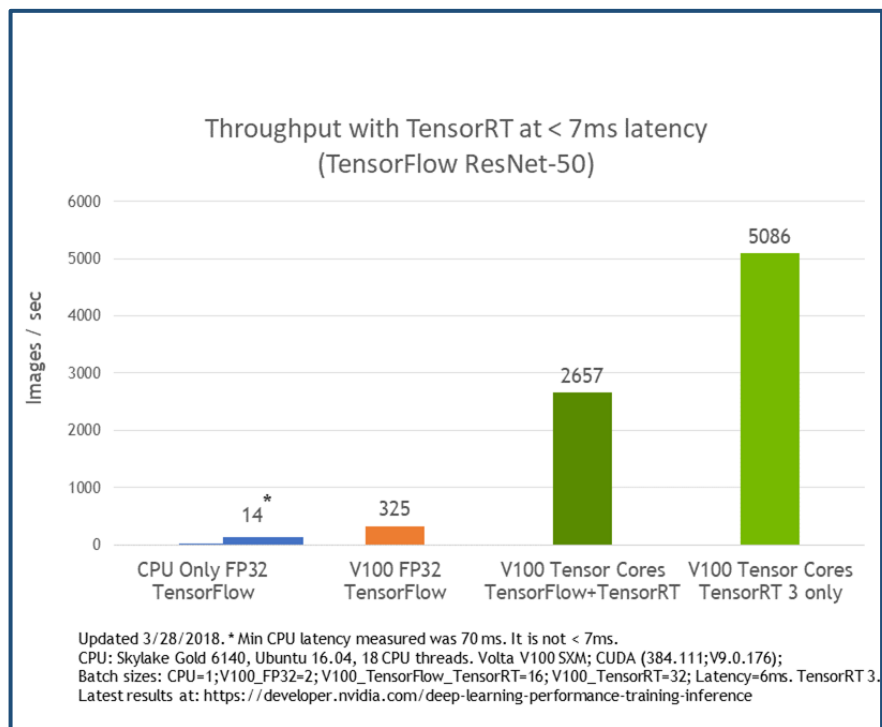
Pytorch to tensorRT

Why we use TensorRT?



What is TensorRT?

학습된 딥러닝 모델을 최적화하여 Nvidia GPU 상에서 **추론 속도를 수배에서 수십배까지 향상시켜서** 딥러닝 서비스를 하는데 도움을 줄 수 있는 **모델 최적화 엔진**



Project: Model Compression

Pytorch to tensorRT

How to convert PyTorch model to TensorRT?

Method 1

 PyTorch



ONNX



Method 2

 PyTorch



[torch2trt]



Pytorch에서 TensorRT로 변환할 때, 주로 **ONNX**를 거쳐서 변환을 한다.
해당 방법의 경우 아래의 제약 사항을 가진다.

- 1) PyTorch, ONNX, tensorRT 등 여러 라이브러리의 버전에 매우 의존적
- 2) ONNX에서 지원하지 않는 Layer의 경우, TensorRT에서 지원을 하더라도 변환 불가

torch2trt를 이용하여 변환할 때, 이점은 아래와 같다.

- 1) torch2trt에서 지원하지 모듈이더라도 커스텀으로 개발이 가능
- 2) torch2trt 모듈이 오픈소스로 코드가 오픈되어 있어 수정이 가능하다.

Project: Model Compression

Pytorch to tensorRT

Problem at converting from PyTorch to TensorRT

PyTorch에서 TensorRT로 변환할때 발생하는 문제점

TensorRT의 Kernel size에 제한이 존재함

AdaptiveAvgPooling Layer

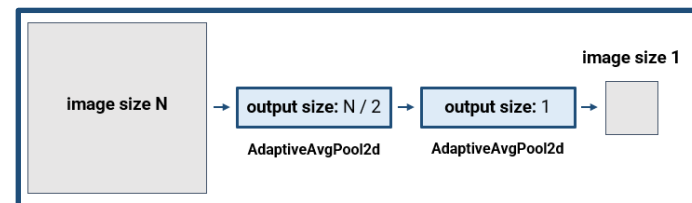
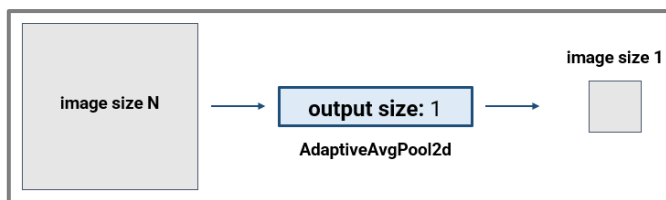
Input Image의 size에 따라 Kernel Size가 변한다.

∴ Input Image size가 큰 경우, TensorRT engine을 생성할 때 AdaptiveAvgPooling Layer가 생성되지 않음

문제점 해결 방안

AdaptiveAvgPooling Layer 쪼개기

AdaptiveAvgPooling Layer의 Kernel size의 크기가 일정 이상 된다면,
AdaptiveAvgPooling Layer를 두개로 나누어 생성할 수 있도록 torch2trt를 개선해줌



Project: Model Compression

Pytorch to tensorRT

Result of converting from PyTorch to TensorRT ※ Target Model: CRAFT

Image size 768 * 768

Image size 768*768 models performance (시간 단위 ms)

model name	library	inference time	DetEval
efficientUnet b5	pytorch	39	0.8224
efficientUnet b5	tensorRT	23	0.8224
efficientUnet b3	pytorch	29	0.8311
efficientUnet b3	tensorRT	17	0.8311
vgg16bnUnet	pytorch	26	0.9104
vgg16bnUnet	tensorRT	19	0.9099

27% ~ 41%
Inference Time 감소

Image size 1400 * 1400

Image size 1400*1400 models performance (시간 단위 ms)

model name	library	inference time	DetEval
efficientUnet b5	pytorch	115	0.8232
efficientUnet b5	tensorRT	68	0.8232
efficientUnet b3	pytorch	80	0.8200
efficientUnet b3	tensorRT	49	0.8203
vgg16bnUnet	pytorch	79	0.8687
vgg16bnUnet	tensorRT	61	0.8687

23% ~ 41%
Inference Time 감소

Layer 별 Inference Time 비교

Layer 별 inference time CRAFT + EfficientUnet b5 (image_size 768)

Layer 명	Pytorch (ms)	tensorRT (ms)	inference time 감소율
Convolution	19.465	17.2028	11.6%
sigmoid + elementwise	5.280	4.2611	19.29%
adaptiveavgpool2d	0.697	1.697	-143%

대부분의 주된 Inference Time을 가지는 Layer들의 경우
11% ~ 19% Inference Time 감소

Project: Model Compression

Quantization

Quantization Survey ※ Used Library: Pytorch

Dynamic Quantization

```
quantized_lstm = torch.quantization.quantize_dynamic(  
    float_lstm, {nn.LSTM, nn.Linear}, dtype=torch.qint8  
)
```

model: fp32 Size (KB): 3.743

model: int8 Size (KB): 2.655

29% 감소

Floating point FP32
638 μ s \pm 2.02 μ s per loop

Quantized INT8
473 μ s \pm 1.11 μ s per loop

26% 감소

Static Quantization * image_size=768

CRAFT (ICDAR13)

Aa 이름	backbone	DetEval	Inference Latency	total Latency	model memory(MB)	fuse_model
Base	VGGBN16Unet	0.9135	0.8424	0.9068	83.1718	
Trial 1	VGGBN16Unet	0.9055	0.5866	0.6690	20.9299	
Trial 2	VGGBN16Unet	0.9050	0.5339	0.6044	20.8527	convBN convBNReLU

Inference Latency: 30%~36.6% 감소

Model Memory: 약 75%감소

Model Evaluation: 0.8%~0.9% 감소

Project: Model Compression

What I learned

Pytorch to TensorRT

- 서비스를 위한 Model에 관해서 생각할 수 있는 계기가 됨
 - 성능뿐만아니라 Inference 서버, 혹은 장비의 자원의 고려
- Open Source 활용 방법에 대해 배움
 - 수정 및 적용하는 방법을 공부함
- TensorRT 공식문서를 보며 TensorRT를 이해하고 활용할 수 있는 힘을 기름

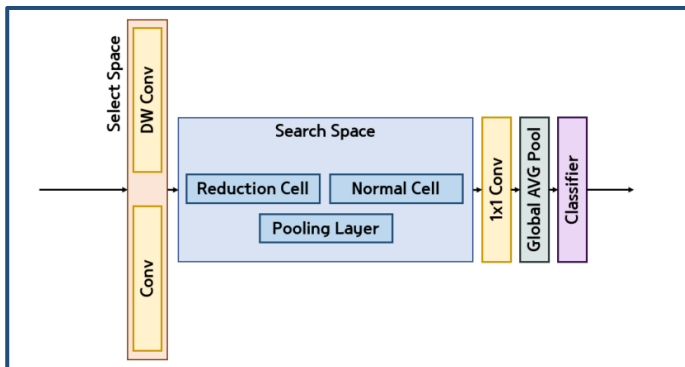
Quantization Survey

- Quantization에 대한 이해를 할 수 있게 됨
 - Pytorch 공식 문서 참고 (pytorch.org)
 - 관련 논문을 보며 학습함 ([Quantizing deep convolutional networks for efficient inference: A whitepaper](#))
- Pytorch에서 제공하는 Quantization 기법들을 실험하며 이해를 할 수 있게됨

Project: Neural Architecture Search

Search Lightweight Models

Search Space



Normal Cell	Reduction Cell	Pooling Layer	ECA
Inverted Residual V2	Inverted Residual V2	MaxPooling	Inverted Residual V2
Inverted Residual V3	Inverted Residual V3		Inverted Residual V3
Ghostbottleneck	Ghostbottleneck		Bottleneck
MBConv	MBConv		
Fire			
Bottleneck			

탐색하려는 모델의 Architecture 구조
→ First Conv, Middle Space, Last Conv

Last Convolution 고정 → 다양한 Task 활용

- Search Space 축소
- 충분한 feature 보장

Model Search Pruning

Model Flops Pruning

- Model Search의 목적이 경량 모델 탐색이므로 일정 Flops이상의 모델은 Training 과정을 거치지 않음

Training Pruning

- Early Stopping에 착안하여, 이전에 가장 좋은 성능을 낸 모델의 Training log를 기준으로 판단
- Epoch별 Model의 성능이 연속적으로 기준 미달시 Pruning

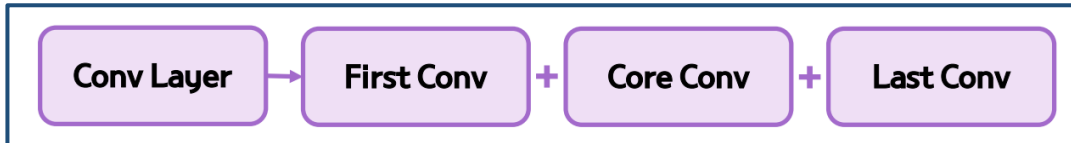
Architecture Similar Pruning

- Training Pruning에서 필터링된 Architecture를 저장
- 저장된 Architecture들과 탐색된 Architecture를 비교하여 일정 유사도 이상을 가진다면 Pruning

Project: Neural Architecture Search

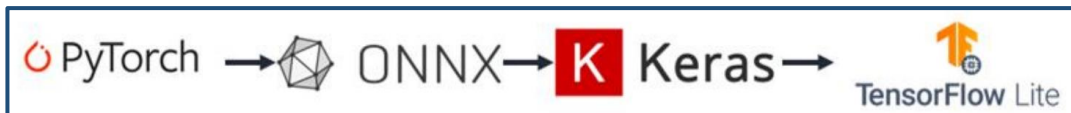
Other Functions For Service

Tensor Decomposition



- NAS를 통해 탐색된 최종 Model을 경량화 시켜주는 후처리 단계
- Tucker Decomposition으로 Model의 Convolution Layer들을 경량화
- Target Layer에 EVBMF함수를 사용하여 Rank를 추출하여 Tensor 분해
- Target Layer를 일반적인 Convolution Layer만 추가로 경량화
 - 모델의 성능 하락의 위험성을 줄이기 위해

Pytorch to TFLite Converter



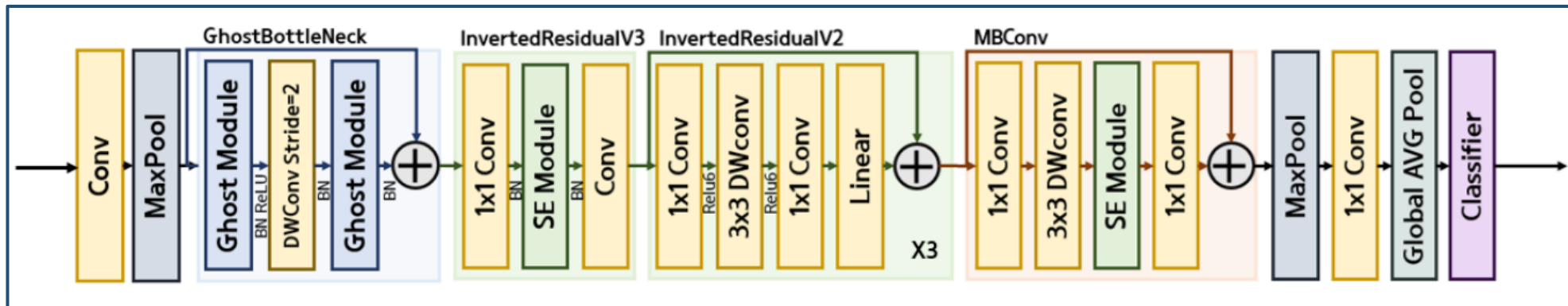
- On-Device를 위해 TFLite로 Model을 Converting함

Project: Neural Architecture Search

Result Model from NAS Project

GTSNet

Architecture of GTSNet



MACs Compare

연산량 (MACs)

Image Size	GTSNet_x1.0(Ours)	ShuffleNetV2_x0.5	MobileNetV2	GhostNet_x1.0
224x224	33,965,857	41,810,537	314,131,496	145,892,401
128x128	11,104,513	13,658,633	103,436,264	48,656,449
64x64	2,791,297	3,421,577	26,819,816	13,297,921
32x32	712,993	862,313	7,665,704	4,458,289

※ 기존의 Model들 보다 현저히 MACs가 적음

Performance Compare

CIFAR 100

모델	ACC.(%)	Total Params
GTSNet_x1.0(Ours)	0.6999	0.27M
ShuffleNetV2_x0.5	0.6795	0.44M
MobileNetV2	0.6834	3.6M
GhostNet_x1.0	0.6692	2.7M

CIFAR 10

이름	ACC.(%)	Total Params
GTSNet_x1.0(Ours)	0.9052	0.17M
ShuffleNetV2_x0.5	0.8941	0.35M
MobileNetV2	0.9049	2.2M
GhostNet_x1.0	0.8968	2.7M

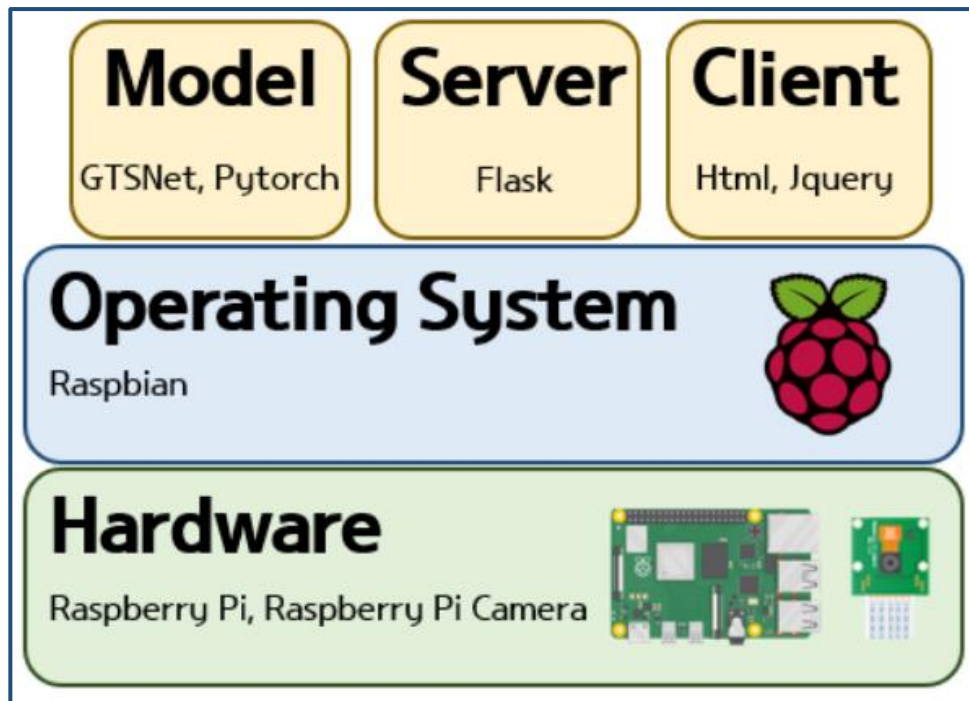
※ 기존의 Model들 보다 더 높은 성능을 보여줌

Project: Neural Architecture Search

Use of GTSNet

Garbage Classification with RPI

- Raspberry Pi에 GTSNet을 이용한 쓰레기 분류 Robot을 제작



※ 시연 영상: [\[YouTube\]](#)

Wearing a Face Mask Classification with Android

- GTSNet을 TFLite Model로 converting하여 Android 서비스 제작



- 마스크 체크인: 공공 장소에서 마스크 착용 여부 식별을 위해 개발
- On-Device AI 기술 이용: 인터넷 없이 사용가능

※ 어플리케이션 마켓 URL: [\[Google Play\]](#)

Project: Neural Architecture Search

What I learned

NAS Project

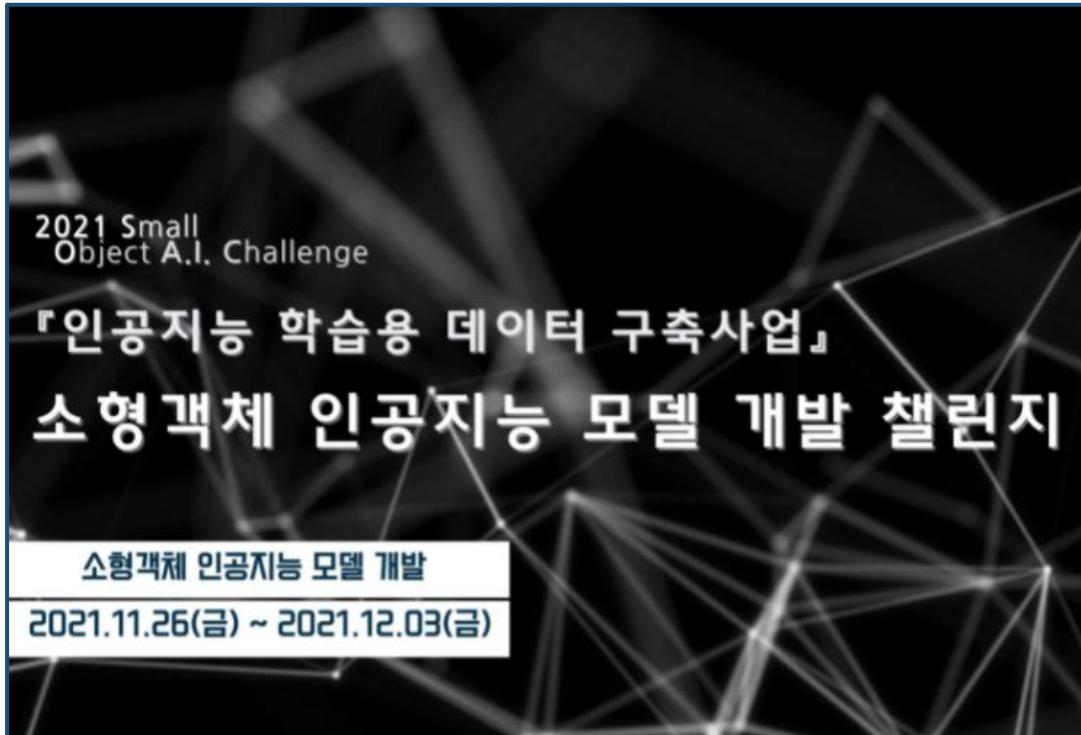
- 모델을 경량화 하기 위한 여러 기법에 대해 학습
 - Pruning
 - Quantization
 - Tensor Decomposition
- 서비스를 위해 모델을 변환하는 과정을 학습함
 - Pytorch to TFLite
- 협업을 위해 서로 다른 생각을 조율하는 방법에 대해 고민하는 시간을 가짐
- Git을 통한 협업을 학습 및 실습
- 경량화된 모델을 찾기위해 여러 모듈에 관해 학습

Use of GTSNet

- 서비스를 위한 모델을 고민함
 - Pytorch to TFLite
- 서비스를 위해 Flask개발을 학습
- 라즈베리 파이를 이용하여 On-Device 서비스 실습
- 간단한 프론트엔드 개발

Project: Small Object Detection

About Competition



- 본선 대회 기간: 2021-12-01 14:00 ~ 2021-12-03 14:00
- 본선 대회 환경: nsml
- [대회 page](#)

Competition Strategies

- Model: Faster r-cnn
 - Backbone: resnet50
- Optimizer: Adam
- Criterion:
 - Bounding Box: lloU Loss
 - Classification: label smooth Loss
- Scheduler: StepLR

Result of Competition

TRACK 1. 소형객체 인식 AI 모델 개발			
Rank	Name		Score
1	BCP6		0.30605390388804876
2	Oh Yeah!		0.24052473034387414
3	쿠키정과		0.23558730776281897
4	S.S		0.13751590488638127
5	SuperAwesome		0.1359077554702456

Project: Small Object Detection

Select Model

- Why Faster-rcnn?
 - 2일간 진행되는 대회이기 때문에 빠른 Training이 필수
 - 소형 객체를 판별해야 했기에
1-stage보다 2-stage모델을 사용하는 것으로 결정
 - 2-stage 모델에서 비교적 빠른 속도로 학습이 가능한 모델을 탐색함.→ Faster-rcnn

Architecture	Training Time (hours)
Faster R-CNN	8.20
RFCN	4.30
SNIPER	62.50
RetinaNet	10.61
CenterNet	5.3
YOLOv3	4.70
SSD	251.18

- [An Improved Faster-rcnn for small object detection](#) 참조

Select Loss

- Why lIoU Loss?
 - [An Improved Faster-rcnn for small object detection](#) 참조
 - 기존에 사용하던 Smooth L1 Loss보다 lIoU Loss가 더 높은 성능을 보여준다는 것을 알 수 있음

TABLE 1. Comparison of the detection performance of L_{lIoU} and smooth L_1 loss function.

	AP	AP75
L_1	0.361	0.346
L_{lIoU}	0.389	0.395

- Small Object Detection에는 해당 Loss가 더 나은 성능을 보인다고 판단하여 해당 Loss를 사용함

Project: Small Object Detection

What I learned

- 소형 물체를 인식하는 방법에 대해 고민함
 - An Improved Faster R-CNN for Small OBD
- 논문에서 제시한 방법론을 현재 Task에 적용해봄
- 제한된 자원 내에서 문제를 해결하기 위한 고민을 하고 해결책을 제시해봄



Thank you

[Github.com/happybeagle](https://github.com/happybeagle)