

서울대학교 중급 프로젝트

결과 보고서

팀명 : Sapphire Vision

팀원 : 정현택, 김유로, 잘리너바 아이가눔

목차

Warboy Preparation

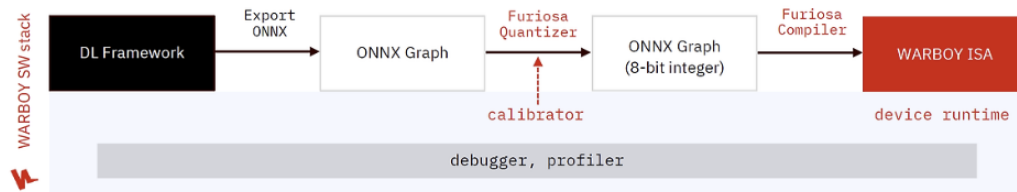
Object detection

1. Model Quantization
2. Measuring model inference performance
3. Developing Vision AI application(Object Detection)

Instance Segmentation

1. Model Quantization
2. Measuring model inference performance
3. Developing Vision AI application(Object Detection)

Warboy Preparation



AI Service using Warboy

- DL Framework : AI 모델을 개발하는 딥러닝 프레임워크
 - Export ONNX : 현재 Warboy SDK에서는 TFLite, ONNX 모델을 지원하고 있기 때문에, PyTorch와 같은 DL Framework에서 개발한 모델을 TFLite 혹은 ONNX로 변환하는 과정이 필요함
- ONNX(Open Neural Network Exchange) Graph : ONNX 형식의 모델 그래프입니다.
 - Furiosa Quantizer : Inference 최적화를 위해 WARBOY는 8-bit 정수형 모델을 기반으로 동작하며, WARBOY에서 모델을 실행하기 위해 8-bit 정수형 모델로의 변환이 필요하며 이 과정을 **Quantization**이라 함
- ONNX Graph(8 bit integer) : 양자화된 8 bit integer 형식의 ONNX 그래프
 - Furiosa Compiler : Furiosa SDK는 compiler를 제공하고 이를 통해 WARBOY에 최적화된 프로그램을 생성하고, 또한 이를 실행할 수 있는 다양한 Runtime을 제공함으로써 사용자들이 WARBOY 기반의 AI서비스를 효과적으로 실행할 수 있음
- Warboy ISA : AI 모델이 실행되는 최종 단계
- Debugger, profiler : Useful tool for optimization
 - Furiosa SDK는 사용자들이 NPU를 활용해 DL 서비스의 성능을 최적화할 수 있도록 프로파일링 및 디버깅 도구 제공하고 있음

1. API server registration

FuriosaAI가 제공하는 SW 요소들을 사용하기 위해서, **NPU 장치의 Driver, Firmware** 그리고 **Runtime 패키지들을 APT 서버를 통해 설치**할 수 있으며, 이를 위해 **APT 서버를 설정**하는 과정이 필요함.

- 1. HTTPS 기반의 APT 서버 접근을 위해 필요 패키지 설치

```
sudo apt update
sudo apt install -y ca-certificates apt-transport-https gnupg wget
```

- 2. Furiosa AI의 공개 signing key 등록

```
mkdir -p /etc/apt/keyrings && \
wget -q -O- https://archive.furiosa.ai/furiosa-apt-key.gpg \
| gpg --dearmor \
| sudo tee /etc/apt/keyrings/furiosa-apt-key.gpg > /dev/null
```

- 3. Furiosa AI 개발자 센터에서 API 키를 발급하고 발급한 API 키를 아래와 같이 설정

아래 코드의 Key(ID)와 Password는 따로 발급받아야 합니다.

```
sudo tee -a /etc/apt/auth.conf.d/furiosa.conf > /dev/null <<EOT
machine archive.furiosa.ai
    login d844d9b3-98ec-460d-93a4-ecca65b2523c
    password
w6GfiiQGMTJPMZz6SUAC7Q5g6IXM2ezGIBqtyPd3KqNdXxFeEfG1btoKanQlQiLt
EOT

sudo chmod 400 /etc/apt/auth.conf.d/furiosa.conf
```

- 4. 리눅스 버전에 따라 APT 서버 설정 (Ubuntu 20.04, 22.04, 24.04 호환 가능)

```
sudo tee -a /etc/apt/sources.list.d/furiosa.list <<EOT
deb [arch=amd64 signed-by=/etc/apt/keyrings/furiosa-apt-key.gpg]
https://archive.furiosa.ai/ubuntu focal restricted
EOT
```

2. APT 서버를 이용한 필수 패키지 설치

```
sudo apt-get update && sudo apt-get install -y furiosa-driver-warboy
furiosa-libnux
```

3. 내 환경에 NPU가 있는지 확인

```
sudo apt-get install -y furiosa-toolkit
furiosactl info
```

```
htaekjung@polarlnx2:/users$ furiosactl info
```

NPU	Name	Firmware	Temp.	Power	PCI-BDF
npu0	warboy	1.7.7, 386a8ab	59°C	11.84 W	0000:01:00.0

4. Conda 기반의 Python 개발 환경설정 및 Furiosa Python SDK 설치

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
sh ./Miniconda3-latest-Linux-x86_64.sh
source ~/.bashrc
conda create -n furiosa-3.9 python=3.9
conda activate furiosa-3.9
```

```
(furiosa-3.9) htaekjung@polarlnx2:/users$ conda activate furiosa-3.9
(furiosa-3.9) htaekjung@polarlnx2:/users$ python --version
Python 3.9.20
```

파이썬 3.9.20 버전이 적용됨을 알 수 있습니다.

5. Furiosa Python SDK 설치

```
pip install furiosa-sdk[full]
```

아래의 명령어를 통해 기본 설정을 마무리

```
(furiosa-new) htaekjung@polarlnx2:~$ git clone https://github.com/furiosa-ai/warboy-vision-models.git
```

```
(furiosa-new) htaekjung@polarlnx2:~$ pip install -r requirements.txt
```

```
(furiosa-new) htaekjung@polarlnx2:~$ sudo apt-get update
```

```
(furiosa-new) htaekjung@polarlnx2:~$ sudo apt-get install cmake libeigen3-dev
```

```
(furiosa-new) htaekjung@polarlnx2:~$ ./build.sh
```

```
(furiosa-new) htaekjung@polarlnx2:~$ wget https://github.com/ultralytics/assets/releases/download/v8.1.0/yolov8n.pt
```

```
(furiosa-new) htaekjung@polarlnx2:~$ python tools/export_onnx.py cfg/object_detection_model.yaml
```

Object detection

1. Model Quantization

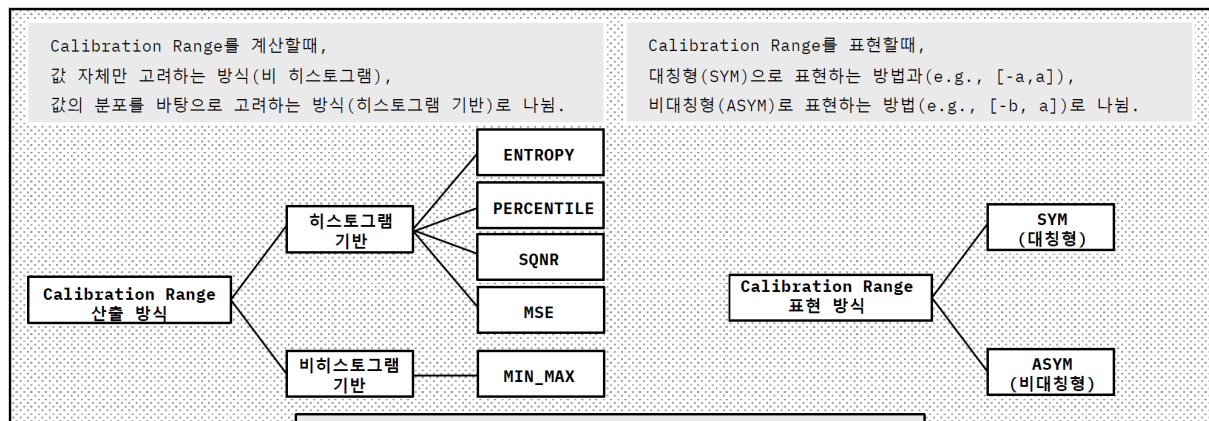
2-1. Pytorch 모델을 onnx 타입으로 변환

```
python tools/export_onnx.py cfg/object_detection_model.yaml
```

위 명령어를 통해 yolo8n.onnx 파일이 만들어지며 이를 [Netron](#)을 통해 시각화한 결과 다음과 같은 결과를 얻을 수 있습니다.

2-2. onnx 파일을 양자화하기

저희는 Object Detection을 목표로 하기에 이에 적합한 calibration method를 찾아보았습니다.



총 10개의 calibration method에 대한 특성을 파악 후 특성 비교 후 적절한 calibration method를 선정 후, 각각의 calibration method 성능 평가를 통해 예측이 맞는지 확인해 볼 것입니다.

(1) Entropy

- 특성:

- 데이터의 분포를 고려하여 양자화 범위를 설정합니다
- 양자화 전후 분포 차이를 줄입니다
- Outlier에 강하고, 데이터 분포를 잘 반영하지만 계산 비용이 높습니다

- 적합성:

- 정확도 손실이 적고 다양한 데이터셋에 안정적

(2) Percentile

- 특성:

- 데이터의 특정 상위 백분위수(Percentile)를 기준으로 양자화 범위를 설정합니다.
- 이상치(outlier)에 민감하지 않으며, 분포가 비대칭적인 데이터에 적합

- 적합성:

- Outlier가 많거나 데이터의 동적 범위(dynamic range)가 큰 경우 유용

(3) MinMax

- 특성:

- 데이터의 최소값과 최대값을 기준으로 양자화 범위를 설정합니다
- 계산이 빠르지만 Outlier에 매우 민감하여 양자화 범위가 비효율적일 수 있음

- 적합성:

- 간단한 계산이 필요한 상황에서 유용하지만 정확도 손실 가능성 있음

(4) MSE (Mean Squared Error)

- **특성:**
 - 양자화 전후 값 간의 평균 제곱 오차를 최소화하도록 범위를 설정
 - 데이터의 분포를 잘 반영하며 정확도 유지에 유리
- **적합성:**
 - 정확도가 중요한 모델에 적합

(5) SQNR (Signal-to-Quantization Noise Ratio)

- **특성:**
 - 양자화로 인한 신호 대 잡음 비율(SQNR)을 최적화하여 스케일을 설정
 - 데이터의 손실을 최소화하면서 양자화 범위를 지정
 - **SQNR - Asym:** 제로포인트를 포함하여 비대칭 분포에 적합
 - **SQNR - Sym:** 대칭 범위를 사용하여 효율성을 강화
- **적합성:**
 - 정확도와 효율성을 모두 고려해야 할 때 유리

ASYM(비대칭형) : 제로포인트를 포함하여 비대칭 분포에 적합

SYM(대칭형) : 계산 효율성을 강화

위의 특성을 고려하여 저희는 Object Detection application을 만들기 위해 Entropy - ASYM을 사용하기로 하였습니다.

다만, 더 근거있는 calibration method를 정하기 위해 calibration method로 quantize를 진행한 후 정확도를 비교하였습니다.

Input에는 대략 4초의 input video를 넣어 진행하였습니다.

Inference를 진행할 때 30프레임으로 즉, 전체 114개의 bmp파일이 output으로 출력됩니다.

그 중 57번째 bmp 파일에서 모델이 laptop, keyboard, person, tv를 식별할 수 있어 그 bmp파일을 기준으로 정확도를 판단하였습니다.

아래의 표는 그 정확도를 판단한 것입니다.

	Keyboard(%)	laptop(%)	person(%)	tv(%)	Mean
ENTROPY_ASYM	67	80	24	-	X
ENTROPY_SYM	-	79	-	-	X
PERCENTILE_ASYM	53	91	42	65	62.75
PERCENTILE_SYM	82	93	37	68	70
MSE_ASYM	50	84	38	73	61.25
MSE_SYM	87	91	39	58	68.75
MIN_MAX_ASYM	77	91	37	61	66.5
MIN_MAX_SYM	85	92	29	37	60.75
SQNR_ASYM	50	89	38	68	61.25
SQNR_SYM	56	90	36	68	62.5

따라서 가장 정확도가 높은 PERCENTILE_SYM을 사용하기로 했습니다.

2. Measuring model inference performance

이렇게 저희는 PERCENTILE_SYM을 사용하여 onnx 추출, 양자화, 추론을 진행하고 성능 비교, latency와 throughput를 하였습니다.

2-1)

```
(base) eurokim@centos3:~/code/junior-project/warboy-vision-models$ python tools/export_onnx.py cfg/od_ps.yaml
Start creating onnx file...
/Users/eurokim/.local/lib/python3.10/site-packages/ultralytics/nn/modules/head.py:99: TracerWarning: Converting a tensor to a Python boolean might cause the trace to be incorrect. We can't record the data flow of Python values, so this value will be treated as a constant in the future. This means that the trace might not generalize to other inputs!
  if self.dynamic or self.shape != shape:
/Users/eurokim/.local/lib/python3.10/site-packages/ultralytics/utils/tal.py:388: TracerWarning: Iterating over a tensor might cause the trace to be incorrect. Passing a tensor of different shape won't change the number of iterations executed (and might lead to errors or silently give incorrect results).
  for t, stride in enumerate(strides):
/Users/eurokim/.local/lib/python3.10/site-packages/torch/onnx/_internal/jit_utils.py:258: UserWarning: The shape inference of prim::Constant type is missing, so it may result in wrong shape inference for the exported graph. Please consider adding it in symbolic function. (Triggered internally at ../torch/csrc/jit/passes/onnx/shape_type_inference.cpp:1884.)
  C._jit_pass_onnx_node_shape_type_inference(node, params_dict, opset_version)
/Users/eurokim/.local/lib/python3.10/site-packages/torch/onnx/_internal/jit_utils.py:258: UserWarning: The shape inference of prim::Constant type is missing, so it may result in wrong shape inference for the exported graph. Please consider adding it in symbolic function. (Triggered internally at ../torch/csrc/jit/passes/onnx/shape_type_inference.cpp:1884.)
  C._jit_pass_onnx_graph_shape_type_inference()
/Users/eurokim/.local/lib/python3.10/site-packages/torch/onnx/_internal/jit_utils.py:1178: UserWarning: The shape inference of prim::Constant type is missing, so it may result in wrong shape inference for the exported graph. Please consider adding it in symbolic function. (Triggered internally at ../torch/csrc/jit/passes/onnx/shape_type_inference.cpp:1884.)
  C._jit_pass_onnx_graph_shape_type_inference()
Creating onnx file done! -> yolov8l.onnx
```

2-2)

```
(base) eurokim@centos3:~/code/junior-project/warboy-vision-models$ python tools/furiosa_quantizer.py cfg/od_ps.yaml
libfuriosa.hal.so ... v8.11.0, built @ 43:501f
calibration: 100%
Quantization completed >> yolov8l_18.onnx 200/200 [04:24:00:00, 1.32s/it]
```

3-1)

```
(base) eurokim@centos3:~/code/junior-project/warboy-vision-models$ furiosa-compiler yolov8l_18_od_ps.onnx
[1/6] Compiling from onnx to dfg
Done in 0.36876072s
[2/6] Compiling from dfg to ldg
[1/3] Splitting graph(LAS)...Done in 141.60873s
[2/3] Lowering graph(LAS)...Done in 312.53812s
[3/3] Optimizing graph...Done in 1.7482479s
Done in 455.97574s
[3/6] Compiling from ldg to cdg
Done in 0.009372615s
[4/6] Compiling from cdg to gir
Done in 0.10811345s
[5/6] Compiling from gir to lir
Done in 0.023769303s
[6/6] Compiling from lir to enf
Done in 1.2868681s
+ Finished in 457.76468s
ok: compiled successfully! (output.enf)
```

3-2)

```
(base) eurokim@polaris2:~/warboy-vision-models$ sudo furiosa-bench results/PERCENTILE_SYM/output_od_ps.enf
This benchmark was executed with latency-workload which prioritizes latency of individual queries over throughput.
1 queries executed with batch size 1
Latency stats are as follows
QPS(Throughput): 200.00/s

Per-query latency:
Min latency (us) : 4993
Max latency (us) : 4993
Mean latency (us) : 4993
50th percentile (us): 4993
95th percentile (us): 4993
99th percentile (us): 4993
99th percentile (us): 4993
```

3-3)

```
(base) eurokim@polaris2:~/warboy-vision-models$ sudo furiosa-bench results/PERCENTILE_SYM/output_3d_ps.enf --workload T -n 1000 -b 4 -w 8 -t 8
This benchmark was executed with throughput-workload which prioritizes throughput over latency of individual queries.
1000 queries executed with batch size 4
QPS(Throughput): 3030.51/s
Latency stats are as follows

Per-query latency:
Min latency (us) : 43099
Max latency (us) : 329192
Mean latency (us) : 188711
50th percentile (us): 188675
95th percentile (us): 316440
99th percentile (us): 329153
99th percentile (us): 329192
Increasing number of workers and io_threads will increase throughput
```

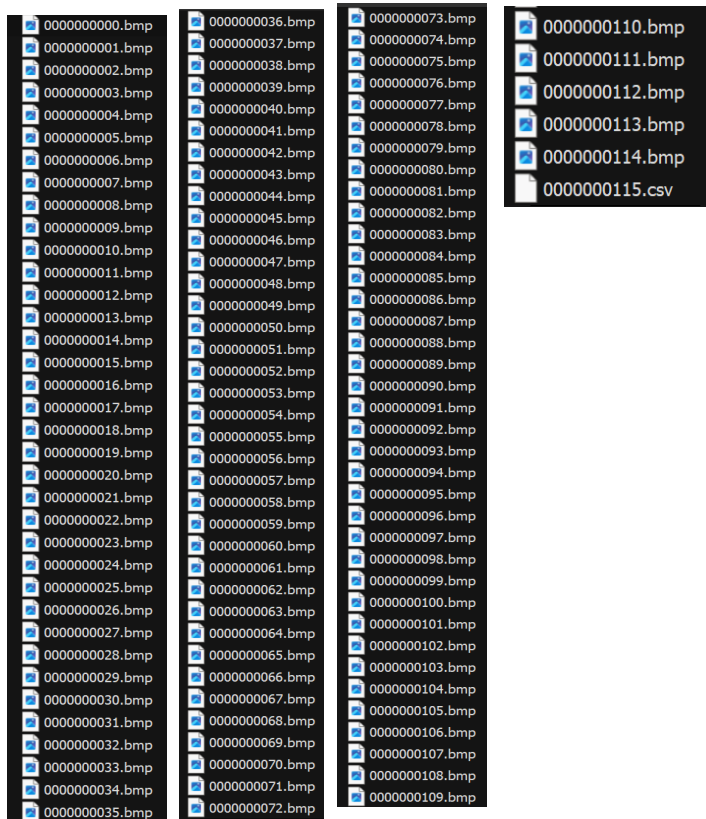
Result

	QPS(Throughput)	Min latency	Max latency	Mean latency
PERCENTILE_ASYM	3039.51	43099	329192	186711

3. Developing Vision AI application(Object Detection)

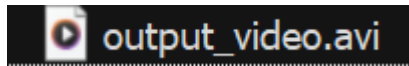
```
(furiosa-new) htaekjung@polarlnx2:~/warboy-vision-models$ python warboy_demo.py cfg/demo.yaml fastAPI
Init ByteTrack!
2024-12-10T08:14:18.041051Z INFO furiosa_rt_core::driver::event_driven::coord: FuriosaRT (v0.10.4, rev: 320a178a7, built at: 2024-05-28T12:51:47Z) bootstrapping ...
2024-12-10T08:14:18.043288Z INFO furiosa_rt_core::driver::event_driven::coord: Found furiosa-compiler (v0.10.1, rev: 8b00177, built at: 2024-05-28T06:02:15Z)
2024-12-10T08:14:18.043298Z INFO furiosa_rt_core::driver::event_driven::coord: Found libhal (type: warboy, v0.12.0, rev: 56530c0 built at: 2023-11-16T12:37:25Z)
2024-12-10T08:14:18.043383Z INFO furiosa_rt_core::driver::event_driven::coord: [Runtime-0] detected 1 NPU device(s):
2024-12-10T08:14:18.068835Z INFO furiosa_rt_core::driver::event_driven::coord: - [0] npu:0* (warboy-b0, 64dpes, firmware: 1.7.7, 386a8ab)
2024-12-10T08:14:18.068910Z INFO furiosa_rt_core::driver::event_driven::coord: [Runtime-0] started
2024-12-10T08:14:18.070582Z INFO furiosa::runtime: Saving the compilation log into /users/htaekjung/.local/state/furiosa/logs/compiler-20241210171418-4epxyz.log
2024-12-10T08:14:18.079549Z INFO furiosa_rt_core::driver::event_driven::coord: [Runtime-0] created Sess-56d20cc4 using npu:0*
2024-12-10T08:14:18.108660Z INFO furiosa_rt_core::driver::event_driven::coord: [Sess-56d20cc4] compiling the model (target: warboy-b0, 64dpes, file: yolov8l_i8.onnx, size: 12.1 MiB)
2024-12-10T08:14:18.276654Z INFO furiosa_rt_core::driver::event_driven::coord: [Sess-56d20cc4] the model compile is successful (took 0 secs)
INFO: Started server process [1565337]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://0.0.0.0:20001 (Press CTRL+C to quit)
2024-12-10T08:14:18.325960Z INFO furiosa_rt_core::driver::event_driven::coord: [Runtime-0] created 8 NPU threads on npu:0* (DRAM: 21.3 MiB/16.0 GiB, SRAM: 11.3 MiB/64.0 MiB)
2024-12-10T08:14:19.563842Z INFO furiosa_rt_core::driver::event_driven::coord: [Sess-56d20cc4] terminated
2024-12-10T08:14:19.765979Z INFO furiosa_rt_core::npu::raw: NPU (npu:0*) has been closed
2024-12-10T08:14:19.767972Z INFO furiosa_rt_core::driver::event_driven::coord: [Runtime-0] stopped
Application -> object detection End!!
INFO: Shutting down
(furiosa-new) htaekjung@polarlnx2:~/warboy-vision-models$ INFO: Waiting for application shutdown.
INFO: Application shutdown complete.
INFO: Finished server process [1565337]
```

위 명령어를 통해 input video에 대해 object detection이 된 bmp파일을 출력하였습니다.



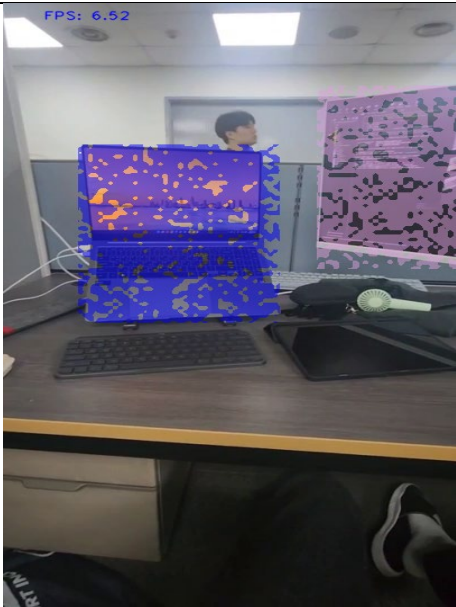
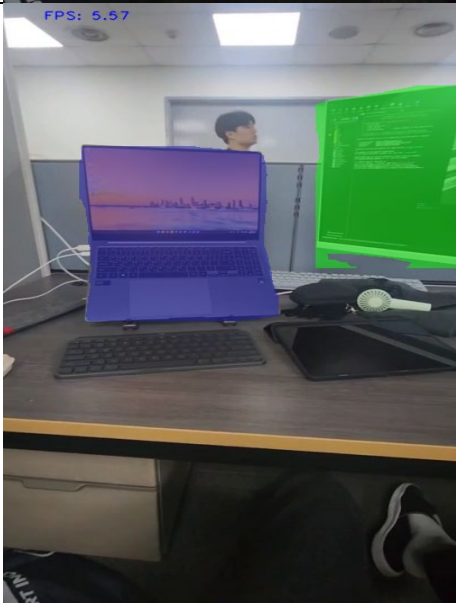
```
(furiosa-new) htaekjung@polarlnx2:~/warboy-vision-models$ python video.py  
동영상 생성 완료 : output_video.avi
```

위 명령어를 통해 출력된 bmp파일을 하나의 동영상으로 만들어 input video에 대해 object detection을 진행한 output video를 출력하였습니다.

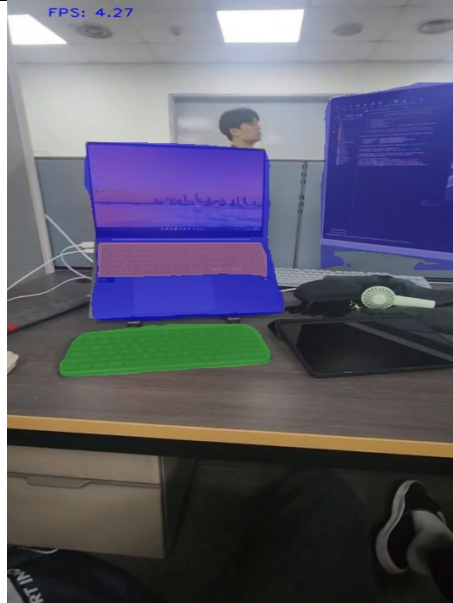


Instance Segmentation

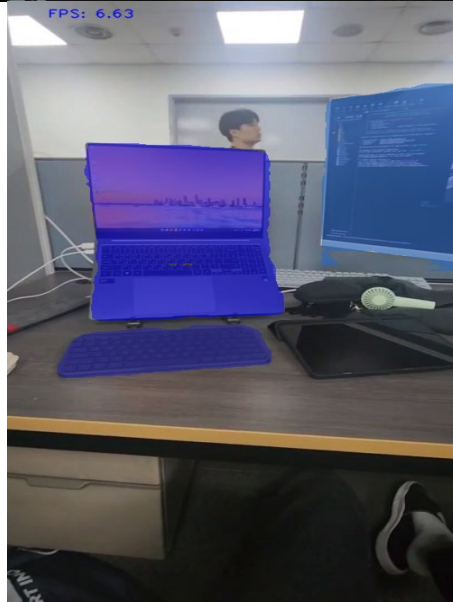
1. Model Quantization

ENTROPY_ASYM	
ENTROPY_SYM	

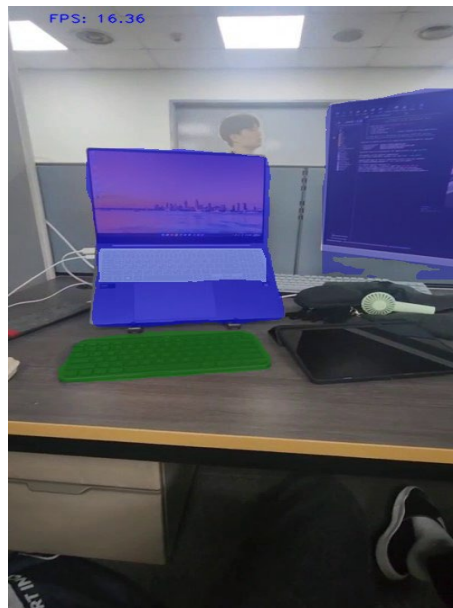
PERCENTILE_ASYM



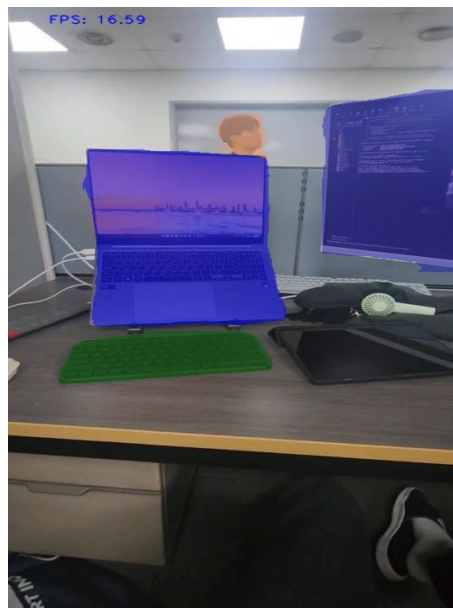
PERCENTILE_SYM



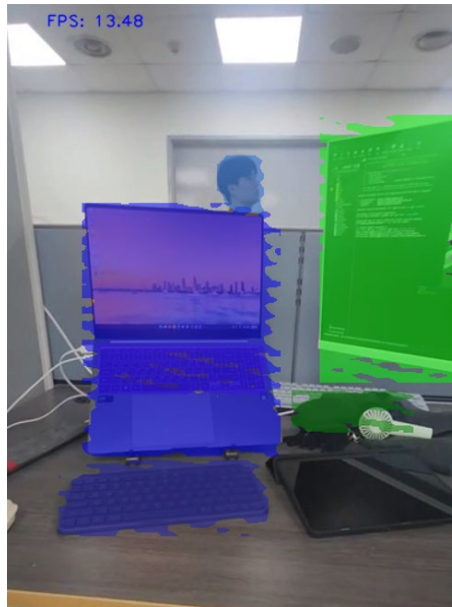
MSE_ASYM



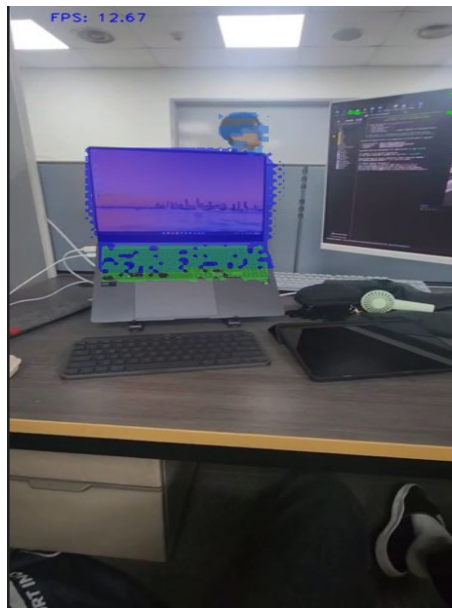
MSE_SYM


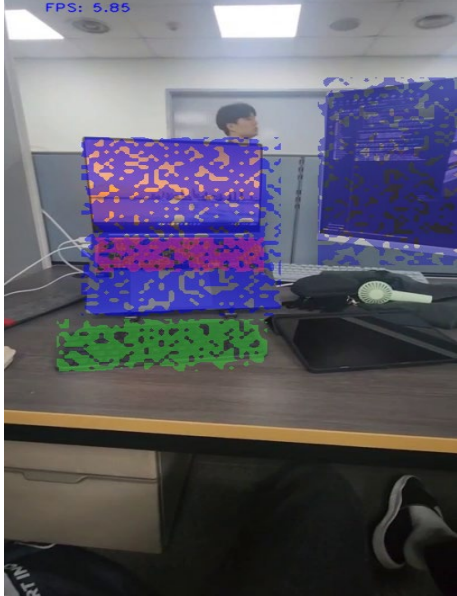


MIN_MAX_ASYM



MIN_MAX_SYM



SQNR_ASYM		
SQNR_SYM		

따라서 저희는 calibration method로 MSE_SYM 방식을 선택하였습니다.

2. Measuring model inference performance

이렇게 저희는 MSE_SYM을 사용하여 onnx 추출, 양자화, 추론을 진행하고 성능 비교, latency와 throughput를 하였습니다.

2-1)

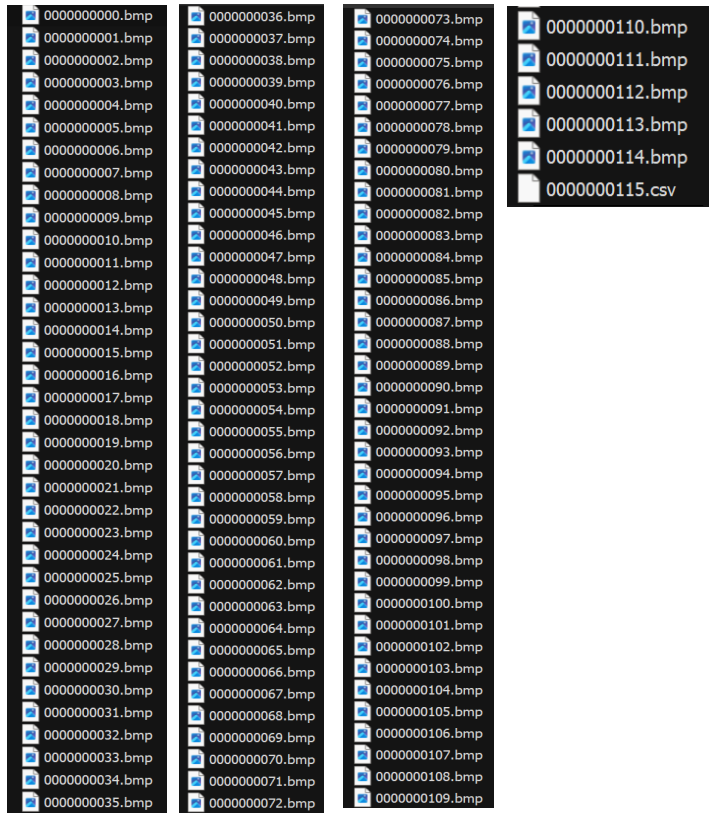
2-2)

3-1)

3-2)

3-3)

3. Developing Vision AI application(Object Detection)



Bmp 파일로 변환 후 video.py를 실행시켜 output을 비디오로 만들었습니다.

```
(furiosa-new) htaekjung@polarlnx2:~/warboy-vision-models$ python video.py  
동영상 생성 완료 : output_video.avi
```