AI on Embedded System Raspberry PI 5 & AI Kit

20230683 박한비, 20230308 최승희

본 과제에서는 Raspberry Pi 5 환경에서 MobileNet-V2 trained at ImageNet dataset 을 대상으로, PyTorch, ONNX Runtime, HailoRT 등 다양한 inference 프레임워크를 설정하고 성능을 비교한다. 특히, on-demand, JIT, comiled, NPU inference 등 여러 실행 모드에서의 performance 비교를 통해 bottleneck 요인을 분석하여 Embedded System 에서의 최적 inference 방식을 찾는 것을 목표로 한다.

1. Check the pre-quantization & JIT option

본 과제에서는 PyTorch 의 MobileNetV2 모델을 이용하여 Quantization 과 JIT (Just In Time compilation) 옵셥이 추론 속도에 미치는 영향을 비교하였다. 동일 환경에서 아래 네 가지 설정을 각각 실행하고, 평균 처리 속도(FPS)를 비교했다.

Model Type	Quantization	JIT	Average Throughput (FPS)
mobilenet_v2	✗ FP32	✗ OFF	**12.2 FPS**
quantization.mobilenet_v2	✓ INT8	✗ OFF	**52.9 FPS**
mobilenet_v2	✗ FP32	✓ ON	**11.1 FPS**
quantization.mobilenet_v2	✓ INT8	✓ ON	**73.3 FPS**

Quantization 은 모델의 가중치와 활성값을 FP32 에서 INT8 로 변환하여, 연산과 메모리 접근을 단순화한다. 기본 mobilenet_v2 에 비해 quantization 을 적용한 모델에서는 throughput 이 약 4 배 이상 향상되었다. 이는 모델의 weight, activation 값을 INT8 로 표현함으로써, 메모리 접근량이 감소하고 캐시 효율이 높아졌기 때문이다.

JIT 컴파일은 PyTorch 의 dynamic graph 를 static 으로 변환하여 Python 인터프리터의 실행 오버헤드를 줄이는 방법이다. JIT 만 적용한 FP32 모델의 경우, 평균 속도가 11.1 로 약간 감소했다. FP32 모델의 경우 대부분의 연산이 이미 고정된 c++ 커널로 실행되기 때문에 JIT 만 적용했을 때는 큰 개선이 없었것으로 보인다.

Quantization + JIT 을 함께 적용한 모델은 평균 73.3 FPS 로, 기본 모델 대비 약 6 배의 성능 향상이 나타났다. 결과적으로, 두 최적화 옵션을 함께 사용했을 때 가장 큰 성능 향상을 얻을 수 있었다.

2. PyTorch vs. ONNX Runtime: 성능 비교 및 차이 원인 분석

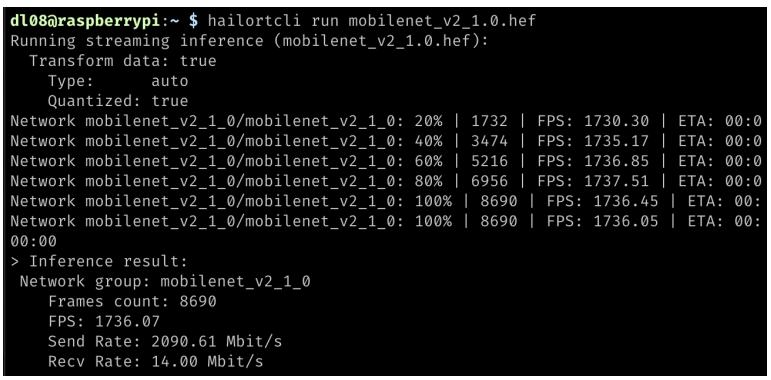
이번 실험에서는 같은 MobileNet-V2 모델을 이용해 PyTorch 와 ONNX runtime 의 CPU inference Performance 를 비교했다.

Model Type	Framework	Quantization	JIT	Average Throughput (FPS)
mobilenet_v2	PyTorch	✗ FP32	✗ OFF	12.2
quantization.mobilenet_v2	PyTorch	✓ INT8	✓ ON	73.3
mobilenet_v2.onnx	ONNX Runtime	✗ FP32	–	62.8
quantization.mobilenet_v2.onnx	ONNX Runtime	✓ INT8	–	143.5

동일한 FP32 모델 기준으로 PyTorch 의 12.2 FPS 에 비해 ONNX Runtime 에서는 62.8 FPS, 즉 약 5 배 빠른 속도를 보였다.

이러한 차이가 나타나는 이유는 ONNX Runtime 가 PyTorch 보다 구조적으로 가벼운 static graph 기반으로 동작하기 때문이다. PyTorch 는 실행 중에 동적 그래프를 구성하고 Python 인터프리터를 통해 제어되지만, ONNX Runtime 은 모델을 미리 최적화된 그래프 형태로 바꿔서 실행하기 때문에, 런타임 오버헤드가 크게 감소한다. 즉, ONNX 에서의 성능 향상은 프레임워크 내부의 최적화가 직접적인 영향을 준 것으로 해석할 수 있다.

3. Hailo NPU Inference: 하드웨어 가속 성능 비교 및 분석



앞선 실험에서 PyTorch 와 ONNX Runtime 을 이용해 CPU 기반 추론 성능을 측정하였다면, 이번 단계에서는 동일한 MobileNetV2 모델을 Hailo NPU 환경에서 실행하여 하드웨어 가속이 throughput 에 미치는 영향을 분석하였다.

Framework / Hardware	Quantization	JIT	Average Throughput (FPS)
PyTorch (CPU)	✗ FP32	✗ OFF	12.2
PyTorch (CPU)	✓ INT8	✓ ON	73.3
ONNX Runtime (CPU)	✗ FP32	–	62.8
ONNX Runtime (CPU)	✓ INT8	–	143.5
Hailo NPU	✓ INT8	–	**≈ 1736.0**
Hailo NPU (Python API)	✓ INT8	–	**≈ 1523.4**

NPU 에서의 평균 처리 속도는 약 1736 FPS 로, CPU 기반 추론(PyTorch FP32) 대비 약 142 배,ONNX INT8 대비 약 12 배 빠르다. 이는 Hailo NPU 의 dataflow architecture 가 모델 전체를 병렬 파이프라인으로 실행하기 때문으로, CPU 의 순차적 연산 구조와는 근본적으로 다른 접근 방식을 취한다.

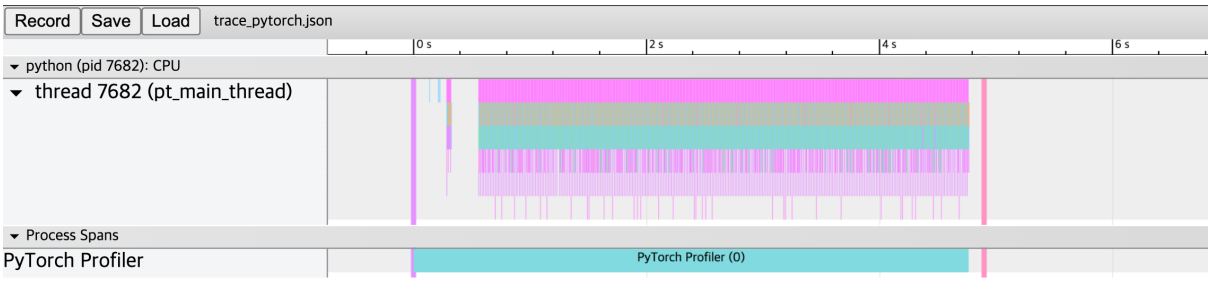
즉, CPU 는 각 layer 의 출력을 메모리에 저장하고 다시 불러와서 처리하지만, NPU 는 레이어 간 데이터를 온칩 SRAM 에서 직접 streaming 하여 메모리 접근 비용을 최소화한다. 이러한 최적화 덕분에 NPU 는 연산이 거의 실시간에 가깝게 이어지며, 계산 효율이 매우 높다.

4. Performance Profiling

PyTorch 와 ONNX Runtime 에서 각각 MobileNetV2 모델의 추론을 수행하며, 성능 bottleneck 요소를 정량적으로 분석하였다. 각각 .json 파일을 생성하고 Chrome tracing 을 통해 시각적으로 확인하였다.

1. Pytorch Profiling Result

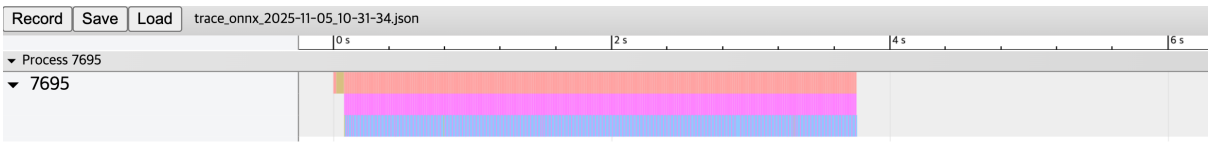
```
(exp) dl08@raspberrypi:~/week9_lab $ python pytorch_bench.py
2025-11-05 10:30:28.530502219 [W:onnxruntime:Default, device_discovery.cc:164 DiscoverDevicesForPlatfo
rm] GPU device discovery failed: device_discovery.cc:89 ReadFileContents Failed to open file: "/sys/cl
ass/drm/card1/device/vendor"
/home/dl08/exp/lib/python3.11/site-packages/torch/ao/quantization/utils.py:376: UserWarning: must run
observer before calling calculate_qparams. Returning default values.
  warnings.warn(
/home/dl08/exp/lib/python3.11/site-packages/torch/_utils.py:383: UserWarning: TypedStorage is deprecat
ed. It will be removed in the future and UntypedStorage will be the only storage class. This should on
ly matter to you if you are using storages directly.  To access UntypedStorage directly, use tensor.un
typed_storage() instead of tensor.storage()
  device=storage.device,
32.862237757090824 fps
70.09852873293735 fps
71.63391471815498 fps
70.70185667538226 fps
Wrote trace_pytorch.json
```



프로파일링 결과를 보면, PyTorch 는 추론 과정 전체에 걸쳐 다양한 색상의 짧은 구간 블록들이 반복적으로 나타나는 형태를 보인다. 이는 각 연산이 CPU 메모리 접근 및 Python 레벨 호출로 나누어 실행되고 있음을 의미한다. 이처럼 연산 단위가 세분화되어 있으며, 각 연산 사이에서 **프레임워크 오버헤드(스케줄링, 함수 호출, 데이터 복사)**가 지속적으로 발생하는 것으로 보인다.

2. ONNX Runtime Profiling Result

```
(exp) dl08@raspberrypi:~/week9_lab $ python onnx_bench.py
2025-11-05 10:31:30.962988229 [W:onnxruntime:Default, device_discovery.cc:164 DiscoverDevicesForPlatfo
rm] GPU device discovery failed: device_discovery.cc:89 ReadFileContents Failed to open file: "/sys/cl
ass/drm/card1/device/vendor"
/home/dl08/exp/lib/python3.11/site-packages/torch/ao/quantization/utils.py:376: UserWarning: must run
observer before calling calculate_qparams. Returning default values.
  warnings.warn(
/home/dl08/exp/lib/python3.11/site-packages/torch/_utils.py:383: UserWarning: TypedStorage is deprecat
ed. It will be removed in the future and UntypedStorage will be the only storage class. This should on
ly matter to you if you are using storages directly.  To access UntypedStorage directly, use tensor.un
typed_storage() instead of tensor.storage()
  device=storage.device,
/home/dl08/exp/lib/python3.11/site-packages/torch/jit/_trace.py:684: UserWarning: The input to trace i
s already a ScriptModule, tracing it is a no-op. Returning the object as is.
  warnings.warn(
81.4683767623933 fps
81.68256335670607 fps
81.43897776045547 fps
Wrote trace_onnx_2025-11-05_10-31-34.json.json
```



ONNX Runtime 의 프로파일링 결과는 PyTorch 와 비교해, 블록이 길고 일정한 간격으로 배치된 패턴을 보인다. 이는 대부분의 연산이 사전 컴파일된 static graph 형태로 실행되어, Python 레벨이 아닌 C++ Backend 에서 직접 수행되기 때문이다. 즉, PyTorch 에 비해 불필요한 Python 호출이나 스케줄링 비용이 제거되어 CPU 리소스가 연산에 더 효율적으로 사용되고 있는 것을 확인할 수 있었다.