**연산 양자화 손실 분석 보고서**

모델 리스트

|  |  |  |  |
| --- | --- | --- | --- |
| **모델** | **Bottleneck 반복 수** | **Backbone 구조** | **Neck 구조** |
| v8n\_2442 | [2, 4, 4, 2] | C2f | C2f |
| v8n\_4664 | [4, 6, 6, 4] | C2f | C2f |
| v8n\_6886 | [6, 8, 8, 6] | C2f | C2f |
| v8n\_b8n5\_2442 | [2, 4, 4, 2] | C2f | C3 |
| v8n\_b8n5\_4664 | [4, 6, 6, 4] | C2f | C3 |
| v8n\_b8n5\_6886 | [6, 8, 8, 6] | C2f | C3 |
| v5n\_2442 | [2, 4, 4, 2] | C3 | C3 |
| v5n\_4664 | [4, 6, 6, 4] | C3 | C3 |
| v5n\_6886 | [6, 8, 8, 6] | C3 | C3 |
| v5n\_b5n8\_2442 | [2, 4, 4, 2] | C3 | C2f |
| v5n\_b5n8\_4664 | [4, 6, 6, 4] | C3 | C2f |
| v5n\_b5n8\_6886 | [6, 8, 8, 6] | C3 | C2f |

1. **실험 개요**
   1. **연산 양자화 손실** : 이 보고서는 NPU와 GPU 간 성능 비율 차이에 중점을 두고, 8bit 양자화로 인한 성능 저하 원인을 분석한다. 특히, NPU는 GPU와 다른 연산 구조와 특성을 가지므로, 양자화 시 발생하는 성능 저하가 두드러질 수 있다.
   2. **배경** : 본 보고서는 S모델 분석 보고서\_p2에서 언급된 연산량 감소와 양자화 손실의 관계를 다룬다. S모델 분석에서 제안된 연구 방향에 따라, NPU와 GPU 간 성능 비율 차이를 양자화 손실 관점에서 탐구하며, 주요 구조적 조합이 성능 저하에 미치는 구체적인 영향을 분석한다. GPU는 8bit 양자화에도 성능 저하가 적지만, NPU는 구조적 차이로 인해 특정 모델 구성에서 성능 저하가 크게 발생할 수 있다. 이에 따라 본 보고서는 NPU 성능 최적화를 위한 이상적인 구조적 조합을 도출하는 데 목적을 둔다.

이 분석을 통해, GPU에서 우수한 성능을 보이나 NPU에서 성능 저하가 나타나는 구조적 특징을 파악하고, NPU에 최적화된 구조적 방향을 제시하고자 한다.

1. **실험 설계 및 방법**
   1. **모델 구조** : 모델의 구조 설계에서 YOLO 모델의 주요 구성 요소인 C2f, C3, Neck, Bottleneck 반복 수를 중심으로 성능 차이를 분석한다. 각 요소의 구조와 기능은 다음과 같다.

**Bottleneck 구조**:  
입력된 특징을 효율적으로 압축하고 변환하기 위해 두 개의 Conv 연산으로 구성된다. 첫 번째 Conv 연산에서는 입력 채널 수를 줄이고, 두 번째 Conv 연산에서는 다시 채널 수를 복원하여 전체 연산량을 감소시키면서도 중요한 정보를 유지한다. Bottleneck 구조는 입력과 출력을 단순히 연결하지 않고, 필요한 경우 shortcut을 추가하여 입력과 출력이 합쳐지도록 함으로써 정보의 손실을 최소화하고 모델이 빠르게 수렴할 수 있도록 돕는다.

**C2f 구조**:  
Conv2d를 통해 특징을 두 개의 독립적인 경로로 나누고, 각각의 경로에 Bottleneck 블록을 적용하여 다양한 공간적, 채널적 특성을 수집한다. 이 구조에서는 입력을 두 부분으로 나누어 처리한 후, Bottleneck에서 출력된 특징을 결합하여 다음 레이어로 전달한다. C2f 구조에서는 Bottleneck 블록을 여러 개 연속적으로 활용하여 다층적인 특징 융합을 가능하게 한다. 또한, 각 Bottleneck 블록의 출력은 최종 레이어에서 채널 방향으로 결합되어 더욱 풍부한 정보가 포함된 특성 맵을 형성한다.

**C3 구조**:  
C2f와 유사하게 Bottleneck 블록을 활용하나, 입력을 두 개의 분리된 경로로 나누어 한 경로에는 연속적인 Bottleneck을, 다른 경로에는 단일 Conv 연산을 적용한다. 이후 이 두 경로의 출력을 채널 방향으로 결합하여 다음 레이어로 전달한다. 이 방식은 다양한 공간적 특성을 얻으면서도 연산 효율성을 유지할 수 있도록 Bottleneck 블록을 활용하는 방법으로, Conv 연산과 Bottleneck 블록을 조합하여 깊이 있는 특징을 효과적으로 추출한다.

* 1. **양자화 및 구현 방법** : 각 모델은 GPU와 NPU 환경에서 성능 차이를 비교하기 위해 동일한 양자화 방식과 조건을 유지하며 구현하였다. 이 과정에서 Backbone과 Neck의 조합, Bottleneck 반복 수를 달리하여 다양한 실험을 진행하였다.

**Backbone 조합**:  
이 실험에서는 bC2f와 bC3의 성능 차이를 확인하고, Bottleneck 반복 수 2442, 4664, 6886에 따라 각 Backbone의 영향력을 비교하였다.

**Neck 조합**:  
이 실험에서는 nC2f와 nC3의 성능 차이를 확인하고, Bottleneck 반복 수 2442, 4664, 6886에 따라 각 Neck의 영향력을 비교하였다.

**Backbone과 Neck의 결합**:  
Backbone과 Neck을 각각 bC2f + nC3, bC3 + nC2f 등으로 결합하여 구조를 실험하였다. Bottleneck 반복 수를 단계별로 높이며 각 결합이 NPU와 GPU의 성능 차이에 어떻게 기여하는지 확인하였다. 이를 통해 특정 구조와 반복 수가 NPU에서 성능 최적화를 이루는 데 중요한 요인이 되는지 평가하였다.

1. **실험 기준**
   1. **비교 기준** :
      1. **NPU/GPU 성능 비율**: 연산시 양자화 손실량을 기준으로 한다. NPU 성능이 GPU 대비 상대적으로 얼마나 우수한지 혹은 낮은지를 나타낸다.
      2. **Bottleneck 반복 수**와 **Backbone & Neck 조합**: 각 실험에서 Bottleneck 반복 수를 2442, 4664, 6886으로 변경하고 Backbone(C2f, C3) 및 Neck(C2f, C3) 구조를 조합해가며 최적화된 결과를 확인한다.
   2. **설명** : 각 기준이 성능에 미치는 영향을 세밀하게 분석하여, NPU와 GPU 간의 성능 차이를 유발하는 구조적 특징을 파악한다. 실험은 GPU에서 높은 성능을 보이더라도 NPU에서는 성능 저하가 큰 구조를 식별하는 데 목적이 있다. 이를 통해 8bit 연산 효율에 맞는 최적의 구조를 도출하고, NPU에서 성능 저하가 발생할 수 있는 비효율적인 구조를 구분하여 NPU에 최적화된 모델 설계를 위한 방향성을 제시한다.
2. **실험 결과**

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Npu/Gpu** | **Npu map** | **Gpu map** |
| v8n\_b5n8\_2442 | 0.554335895 | 0.303 | 0.5466 |
| v8n\_b5n8\_4664 | 0.53974665 | 0.294 | 0.5447 |
| v8n\_b5n8\_6886 | 0.571948998 | 0.314 | 0.549 |
| v5n\_b8n5\_6886 | 0.599563953 | 0.33 | 0.5504 |
| v5n\_b8n5\_4664 | 0.579024035 | 0.318 | 0.5509 |
| v5n\_b8n5\_2442 | 0.55545471 | 0.306 | 0.5473 |
| v8n\_6886 | 0.593727977 | 0.337 | 0.5676 |
| v5n\_6886 | 0.577761628 | 0.318 | 0.5504 |
| v5n\_4664 | 0.604196816 | 0.334 | 0.5528 |
| v8n\_4664 | 0.599498926 | 0.335 | 0.5588 |
| v8n\_2442 | 0.53162732 | 0.295 | 0.5549 |
| v5n\_2442 | 0.544481358 | 0.295 | 0.5418 |

**아래의 모델 비교는 왼쪽의 모델을 기준으로 수치를 측정한다.**

1. Bottleneck 수 비교 (2442 vs 4664 vs 6886)
2. v5n\_2442 vs v5n\_4664 vs v5n\_6886
   * v5n\_2442 vs v5n\_4664:
     + NPU/GPU 성능 차이: +5.97%
     + NPU mAP 차이: +3.9%
     + GPU mAP 차이: +1.1%
   * v5n\_4664 vs v5n\_6886:
     + NPU/GPU 성능 차이: +4.35%
     + NPU mAP 차이: +1.6%
     + GPU mAP 차이: -1.6%
3. v8n\_2442 vs v8n\_4664 vs v8n\_6886
   * v8n\_2442 vs v8n\_4664:
     + NPU/GPU 성능 차이: +6.79%
     + NPU mAP 차이: +4%
     + GPU mAP 차이: +0.39%
   * v8n\_4664 vs v8n\_6886:
     + NPU/GPU 성능 차이: -0.97%
     + NPU mAP 차이: +0.6%
     + GPU mAP 차이: +1.55%
4. v5n\_b8n5\_2442 vs v5n\_b8n5\_4664 vs v5n\_b8n5\_6886
   * v5n\_b8n5\_2442 vs v5n\_b8n5\_4664:
     + NPU/GPU 성능 차이: +2.4%
     + NPU mAP 차이: +1.2%
     + GPU mAP 차이: +0.36%
   * v5n\_b8n5\_4664 vs v5n\_b8n5\_6886:
     + NPU/GPU 성능 차이: +2.06%
     + NPU mAP 차이: +1.2%
     + GPU mAP 차이: -0.5%
5. v8n\_b5n8\_2442 vs v8n\_b5n8\_4664 vs v8n\_b5n8\_6886
   * v8n\_b5n8\_2442 vs v8n\_b5n8\_4664:
     + NPU/GPU 성능 차이: -1.46%
     + NPU mAP 차이: -0.9%
     + GPU mAP 차이: -0.1%
   * v8n\_b5n8\_4664 vs v8n\_b5n8\_6886:
     + NPU/GPU 성능 차이: +5.96%
     + NPU mAP 차이: +2%
     + GPU mAP 차이: +0.43%

2) C3, C2f 비교 (Bottleneck 2442 vs 4664 vs 6886)

1. v5n\_2442 vs v8n\_2442
   * NPU/GPU 성능 차이: +1.29%
   * NPU mAP 차이: +0%
   * GPU mAP 차이: -1.31%
2. v5n\_4664 vs v8n\_4664
   * NPU/GPU 성능 차이: +0.6%
   * NPU mAP 차이: -0.1%
   * GPU mAP 차이: -0.6%
3. v5n\_6886 vs v8n\_6886
   * NPU/GPU 성능 차이: -2.63%
   * NPU mAP 차이: -1.9%
   * GPU mAP 차이: -1.72%

3) C2f, C3와 Bottleneck 조합 비교

1. C2f + 4664 vs C3 + 2442 (v8n\_4664 vs v5n\_2442)
   * NPU/GPU 성능 차이: +5.5%
   * NPU mAP 차이: +4%
   * GPU mAP 차이: +1.7%
2. C3 + 4664 vs C2f + 2442 (v5n\_4664 vs v8n\_2442)
   * NPU/GPU 성능 차이: +7.26%
   * NPU mAP 차이: +3.9%
   * GPU mAP 차이: -0.21%
3. C2f + 6886 vs C3 + 2442 (v8n\_6886 vs v5n\_2442)
   * NPU/GPU 성능 차이: +8.92%
   * NPU mAP 차이: +4.2%
   * GPU mAP 차이: +2.58%
4. C3 + 6886 vs C2f + 2442 (v5n\_6886 vs v8n\_2442)
   * NPU/GPU 성능 차이: +8.57%
   * NPU mAP 차이: +3.8%
   * GPU mAP 차이: -0.4%

4) Backbone과 Neck 조합 비교 (bC2f + nC3 vs bC3 + nC2f)

1. bC2f + nC3 vs bC3 + nC2f (v5n\_b8n5\_2442 vs v8n\_b5n8\_2442)
   * NPU/GPU 성능 차이: +0.12%
   * NPU mAP 차이: +0.3%
   * GPU mAP 차이: +0.07%
2. bC2f + nC3 vs bC3 + nC2f (v5n\_b8n5\_4664 vs v8n\_b5n8\_4664)
   * NPU/GPU 성능 차이: +3.93%
   * NPU mAP 차이: +2.4%
   * GPU mAP 차이: +0.62%
3. bC2f + nC3 vs bC3 + nC2f (v5n\_b8n5\_6886 vs v8n\_b5n8\_6886)
   * NPU/GPU 성능 차이: +2.7%
   * NPU mAP 차이: +1.6%
   * GPU mAP 차이: +0.14%

5) 기타 비교

1. v8n\_b5n8\_2442 vs v8n\_2442
   * NPU/GPU 성능 차이: +2.27%
   * NPU mAP 차이: +0.8%
   * GPU mAP 차이: -0.83%
2. v8n\_b5n8\_4664 vs v8n\_4664
   * NPU/GPU 성능 차이: -5.98%
   * NPU mAP 차이: -4.1%
   * GPU mAP 차이: -1.41%
3. v8n\_b5n8\_6886 vs v8n\_6886
   * NPU/GPU 성능 차이: -5.69%
   * NPU mAP 차이: -2.3%
   * GPU mAP 차이: -1.05%
4. v5n\_b8n5\_6886 vs v5n\_6886
   * NPU/GPU 성능 차이: +3.78%
   * NPU mAP 차이: +1.2%
   * GPU mAP 차이: +0%
5. **결과 분석**

1) Bottleneck 수 비교 (2442 vs 4664 vs 6886)

* v5n 및 v8n 모델
  + 4664일 때 가장 높은 NPU/GPU 성능 비율을 보이며, 그 다음은 6886, 2442 순으로 높은 성능을 보인다.
  + Bottleneck 반복 수가 증가함에 따라 8bit 연산 효율에 좋지만, 특정 수준(4664) 이상에서는 성능 포화 현상이 발생하여 추가적인 반복 수 증가(6886)가 오히려 성능 저하로 이어지는 것으로 보인다. 이는 NPU에서의 연산 성능이 특정 Bottleneck 반복 수(4664)에서 최적화되며, 이를 초과할 경우 연산 복잡도 증가로 인해 오히려 병목이 발생하는 성능 한계점에 도달했을 것이라 판단된다. 따라서, v5n, v8n 모델에서는 Bottleneck 반복 수 4664가 8비트 연산에서 최적화된 성능을 제공하는 것으로 결론지을 수 있다.
* v5n\_b8n5 및 v8n\_b5n8 모델
  + 6886일 때 가장 높은 NPU/GPU 성능 비율을 보이며, 그 다음은 4664, 2442 순으로 높은 성능을 보인다.
  + (bC2f + nC3, bC3 + nC2f)를 가진 v5n\_b8n5와 v8n\_b5n8 모델에서는 Bottleneck 반복 수가 클수록 8bit 연산에 더 높은 성능을 나타낸다. C2f, C3 기법을 Backbone과 Neck에서 각각 사용한 구조의 경우 NPU의 연산 최적화를 위해 더 높은 Bottleneck 수가 필요함을 나타낸다.

분석 요약: v5n 및 v8n 모델은 4664일 때 최적화된 NPU/GPU 성능 비율을 보이며, 그 다음으로 6886, 2442 순이다. 반면, C3, C2f를 섞어서 사용한 구조의 경우 Bottleneck 반복 수가 6886일 때 최적화된 성능을 보이며, 4664, 2442 순으로 성능이 감소한다.

2) C3 vs C2f 비교

* v5n 및 v8n 모델
  + Bottleneck 반복 수가 2442 및 4664일 때 v5n 모델이 v8n 모델보다 NPU/GPU 성능 비율이 높다.
  + Bottleneck 반복 수가 6886일 때는 v5n이 v8n보다 NPU/GPU 성능 비율이 2.63% 낮음.
  + 분석: Bottleneck이 반복 수가 (2442, 4664) 인 경우, v5n(C3 구조)이 v8n(C2f 구조)보다 성능이 더 높다. 하지만 반복 수가 6886으로 증가하면, 오히려 C2f 구조를 사용한 v8n의 성능이 높다. 이는 높은 Bottleneck 반복 수가 C2f 구조의 연산 효율을 극대화한다고 판단할 수 있다.

분석 요약: Bottleneck 수가 적을 때(2442, 4664)는 C3 구조(v5n)가 더 유리하며, Bottleneck 수가 많아질수록 C2f 구조(v8n)가 NPU에서 더 높은 성능을 보인다.

3) C3, C2f 와 Bottleneck 조합 비교

* C2f + 4664 vs C3 + 2442 (v8n\_4664 vs v5n\_2442)
  + C2f + 4664 조합이 NPU/GPU 성능 비율에서 C3 + 2442 조합보다 5.5% 더 높다.
* C3 + 4664 vs C2f + 2442 (v5n\_4664 vs v8n\_2442)
  + C3 + 4664 조합이 NPU/GPU 성능 비율에서 C2f + 2442 조합보다 7.26% 더 높다.
* C2f + 6886 vs C3 + 2442 (v8n\_6886 vs v5n\_2442)
  + C2f + 6886 조합이 NPU/GPU 성능 비율에서 C3 + 2442 조합보다 8.92% 더 높다.
* C3 + 6886 vs C2f + 2442 (v5n\_6886 vs v8n\_2442)
  + C3 + 6886 조합이 NPU/GPU 성능 비율에서 C2f + 2442 조합보다 8.57% 더 높다.

분석 요약: C2f와 C3 구조는 Bottleneck 반복 수에 따라 성능 최적화가 달라진다. Bottleneck이 4664일 때는 C3 구조가 더 높은 성능을 보이며, C3 + 4664 조합이 C2f + 2442 조합보다 7.26% 높은 NPU/GPU 성능 비율을 나타낸다. 반면 Bottleneck이 6886으로 증가하면 C2f 구조의 성능이 상대적으로 향상되어, C2f + 6886 조합이 C3 + 2442 대비 8.92% 더 높은 성능을 보인다. 이 결과는 높은 Bottleneck 반복 수에서 C2f가 더 효율적인 연산을 가능하게 한다는 것을 의미하며, 중간 Bottleneck에서는 C3 구조가 최적화된 성능을 제공함을 나타낸다.

4) Backbone & Neck 조합 비교 (bC2f + nC3 vs bC3 + nC2f)

* bC2f + nC3 vs bC3 + nC2f (v5n\_b8n5\_2442 vs v8n\_b5n8\_2442)
  + NPU/GPU 성능 비율이 +0.12% 높으며, 차이가 미미함.
* bC2f + nC3 vs bC3 + nC2f (v5n\_b8n5\_4664 vs v8n\_b5n8\_4664)
  + Bottleneck 수가 4664일 때 (bC2f + nC3) 조합이 (bC3 + nC2f) 조합에 비해 NPU/GPU 성능 비율이 3.93% 높음.
* bC2f + nC3 vs bC3 + nC2f (v5n\_b8n5\_6886 vs v8n\_b5n8\_6886)
  + Bottleneck 수가 6886일 때도 (bC2f + nC3) 조합이 (bC3 + nC2f) 조합보다 NPU/GPU 성능 비율이 2.7% 높음.

분석 요약: 대체로 같은 Bottleneck 반복 수일 때 bC2f + nC3 조합의 성능이 더 높고, Bottleneck 수가 많아질수록 더 높은 NPU/GPU 성능 비율을 보인다. 이는 bC2f + nC3 조합이 bC3 + nC2f 조합보다 8bit 연산 효율에 적합한 것으로 나타난다. 또한 bottleneck 반복 수가 높을수록 더 적합한 것으로 판단된다.

5) 기타 비교 (Bottleneck과 Neck 구조 추가)

* v8n\_b5n8\_2442 vs v8n\_2442 (bC3 + nC2f vs bC2f + nC2f)
  + NPU/GPU 성능 비율이 +2.27% 높음.
* v8n\_b5n8\_4664 vs v8n\_4664 (bC3 + nC2f vs bC2f + nC2f)
  + NPU/GPU 성능 비율이 -5.98% 낮음.
* v8n\_b5n8\_6886 vs v8n\_6886 (bC3 + nC2f vs bC2f + nC2f)
  + NPU/GPU 성능 비율이 -5.69% 낮음.
* v5n\_b8n5\_6886 vs v5n\_6886 (bC2f + nC3 vs bC3 + nC3)
  + NPU/GPU 성능 비율이 +3.78% 높음.
* v5n\_b8n5\_2442 vs v5n\_2442 (bC2f + nC3 vs bC3 + nC3)
  + NPU/GPU 성능 비율이 +1.1% 높음.
* v5n\_b8n5\_4664 vs v5n\_4664 (bC2f + nC3 vs bC3 + nC3)
  + NPU/GPU 성능 비율이 -2.52% 낮음.

분석 요약:

* Bottleneck 반복 수가 2442일 때는 bC3 + nC2f 조합이 bC2f + nC2f 조합에 비해 성능이 더 우수한 경향을 보인다. v8n\_b5n8\_2442 vs v8n\_2442 비교에서 bC3 + nC2f가 bC2f + nC2f보다 NPU/GPU 성능 비율이 2.27% 더 높았다. 이는 낮은 Bottleneck 반복 수에서는 bC3 + nC2f 조합이 효율적임을 나타낸다.
* Bottleneck 반복 수가 4664와 6886으로 증가할 경우에는 bC2f + nC2f 조합이 더 높은 성능을 보이는 경향이 있다. v8n\_b5n8\_4664 vs v8n\_4664와 v8n\_b5n8\_6886 vs v8n\_6886 비교에서 각각 NPU/GPU 성능 비율이 5.98%, 5.69% 더 낮아지는 결과가 나타나는 것으로 보아, 높은 Bottleneck 수에서 bC2f + nC2f 조합이 더 적합하다는 것을 보여준다.
* v5n\_b8n5 vs v5n 비교에서는 Bottleneck이 2442일 때는 bC2f + nC3가 더 나은 성능을 보였으나, 4664, 6886로 증가하자 bC3 + nC3가 더 높은 NPU/GPU 성능을 보였다.

결론적으로, 낮은 Bottleneck 반복 수(2442)에서는 bC3 + nC2f 조합이 더 효율적이며, 높은 Bottleneck 반복 수(4664와 6886)에서는 bC2f + nC2f 조합이 더 적합한 성능을 보인다. 따라서 Bottleneck 반복 수에 따라 최적화되는 조합이 달라지며, 2442와 같은 낮은 반복 수에서는 bC3 + nC2f, 4664 이상에서는 bC2f + nC2f 조합이 최적화된 성능을 제공하는 것으로 판단된다.

1. **분석 및 결론** :

분석을 통해 다양한 Bottleneck 반복 수와 Backbone 및 Neck 조합이 NPU/GPU 성능 비율에 미치는 영향을 평가할 수 있었다. 주요 경향성을 비교 기준별로 설명하고, 각 경향에 맞춘 최적의 조합을 제안한다.

* 1. Bottleneck 반복 수의 영향 (2442, 4664, 6886)
* v5n 및 v8n 모델에서는 Bottleneck 반복 수가 4664일 때 NPU/GPU 성능 비율이 가장 높게 나타났으며, 그 뒤로 6886, 2442 순으로 성능이 감소하는 경향을 보였다. 이러한 경향은 Bottleneck 반복 수가 증가할수록 8비트 연산 효율이 개선되지만, 특정 수준(4664) 이상에서는 성능 포화 현상이 발생하여 더 높은 반복 수(6886)가 오히려 성능 저하로 이어짐을 의미한다. 이는 4664가 해당 구조에서 최적화된 병목 지점이며, 이를 초과하는 반복 수는 오히려 연산의 복잡도 증가로 인해 성능을 저해할 수 있음을 나타낸다.
* v5n\_b8n5 및 v8n\_b5n8 모델은 6886에서 가장 높은 성능을 기록하였으며, 그 다음으로 4664, 2442 순으로 성능이 나타났다. 이 모델들은 Backbone과 Neck에 C2f와 C3 기법을 섞어 사용한 구조로, 특히 높은 Bottleneck 수에서 성능 최적화를 보인다. 이는 복잡한 구조가 Bottleneck 수 증가에 따라 성능이 최적화됨을 의미한다.
* 결론: 단순 Backbone 구조(v5n, v8n)에서는 Bottleneck 반복 수가 4664일 때 최적의 성능을 보였고, 복잡한 구조(v5n\_b8n5, v8n\_b5n8)에서는 6886에서 최적화된 성능을 보였다. 따라서 최적 Bottleneck 수를 결정할 때는 모델의 구조 복잡성에 따라 적절히 선택해야 하며, 복잡한 구조에서는 높은 Bottleneck 수가, 단순 구조에서는 중간 수준(4664)이 적합하다.

2) C3 vs C2f 구조의 비교

* v5n 및 v8n 모델을 기준으로 했을 때, Bottleneck 반복 수가 2442 및 4664일 때는 C3 구조(v5n)가 C2f 구조(v8n)보다 성능이 우수했다. 반면, Bottleneck 반복 수가 6886으로 증가하면, C2f 구조가 NPU/GPU 성능 비율에서 2.63% 더 높은 성능을 기록했다. 이러한 경향은 Bottleneck 반복 수가 낮을수록 C3 구조가 효율적이며, 반복 수가 높아질수록 C2f 구조의 성능 효율이 극대화됨을 나타낸다.
* 결론: Bottleneck 수가 낮거나 중간 수준(2442, 4664)일 때는 C3 구조가 더 적합하고, Bottleneck 수가 높은 상황에서는 C2f 구조가 유리하다. 이를 바탕으로, Bottleneck 수에 따라 C3와 C2f 구조 중 최적의 구조를 선택할 수 있으며, 높은 Bottleneck 수를 사용하고자 한다면 C2f 구조를 선택하는 것이 좋다.

3) C3와 C2f 및 Bottleneck 조합 비교

* 실험 결과, 중간 수준의 Bottleneck 반복 수(4664)에서는 C3 + 4664 조합이 C2f + 2442 대비 7.26% 높은 NPU/GPU 성능 비율을 기록하며 최적화된 성능을 보였다. 반면, 높은 Bottleneck 반복 수(6886) 에서는 C2f + 6886 조합이 C3 + 2442 대비 8.92% 더 높은 성능을 나타냈다.
* 이 경향은 높은 Bottleneck 수가 C2f 구조와 결합되었을 때 연산 효율이 극대화됨을 나타내며, 중간 Bottleneck 수에서는 C3 구조가 상대적으로 더 최적화된 성능을 제공한다는 사실을 의미한다.
* 결론: Bottleneck 수가 낮거나 중간일 때는 C3 구조가 적합하고, 높은 Bottleneck 수에서는 C2f 구조가 최적이라는 경향이 확인된다.

4) Backbone & Neck 조합 비교 (bC2f + nC3 vs bC3 + nC2f)

* bC2f + nC3 조합이 bC3 + nC2f 조합보다 일관적으로 높은 성능을 보였으며, 특히 높은 Bottleneck 수일수록 성능 차이가 증가하는 경향이 있다. 예를 들어, 4664에서 bC2f + nC3 조합은 3.93% 더 높은 NPU/GPU 성능을 나타냈으며, 6886에서도 2.7% 높은 성능을 보였다. 이는 bC2f + nC3 조합이 더 높은 Bottleneck 반복 수에서 효율적임을 의미한다.
* 결론: 높은 Bottleneck 반복 수에서 bC2f + nC3 조합이 더 최적화된 성능을 제공하므로, 복잡한 구조 설계에서 이러한 조합이 더 효율적일 수 있다.

5) 기타 조합에 따른 성능 비교 (Bottleneck과 Neck 구조 추가)

* 낮은 Bottleneck 반복 수(2442)\*\*에서는 bC3 + nC2f 조합이 bC2f + nC2f보다 성능이 우수하며, NPU/GPU 성능 비율이 2.27% 더 높다. 반면, \*\*높은 Bottleneck 반복 수(4664, 6886)\*\*에서는 bC2f + nC2f 조합이 더 높은 성능을 보인다.
* v5n\_b8n5 vs v5n 비교에서도 Bottleneck이 2442일 때는 bC2f + nC3 조합이 유리하였으나, 4664 및 6886에서는 bC3 + nC3가 더 나은 성능을 보였다.
* 결론: 낮은 Bottleneck 반복 수에서는 bC3 + nC2f 조합이 더 효율적이며, 높은 Bottleneck 수에서는 bC2f + nC2f 조합이 더 최적화된 성능을 제공한다.

**최종 결론** :

Bottleneck 반복 수 2442 :

* 최적 Backbone: C3
* 최적 Neck: C2f
* 최적 Backbone-Neck 조합: bC3 + nC2f
  + 낮은 반복 수에서는 C3를 Backbone으로, C2f를 Neck으로 사용하는 조합이 가장 적합하며, 간결한 연산으로 효율을 높일 수 있다.

Bottleneck 반복 수 4664 :

* 최적 Backbone: C2f
* 최적 Neck: C3
* 최적 Backbone-Neck 조합: bC2f + nC3
  + 중간 반복 수에서는 bC2f와 nC3 조합이 가장 효율적이며, 병목을 최소화하고 NPU/GPU 성능 비율을 극대화하는 데 기여한다.

Bottleneck 반복 수 6886 :

* 최적 Backbone: C2f
* 최적 Neck: C2f
* 최적 Backbone-Neck 조합: bC2f + nC2f
  + 높은 반복 수에서는 C2f를 Backbone과 Neck 모두에 사용하는 것이 최적이며, 복잡한 연산을 효율적으로 처리해 NPU의 병렬 연산 성능을 극대화할 수 있다.

1. **향후 계획**

현재 전체 실험의 목표는 소형 객체 탐지를 위해 P2 feature map을 추가하여 특징 추출을 강화하는 방안을 고려하는 것이다. 현재, N모델 분석 보고서에는 내용 중 C3Ghost 기법이 포함되어 있으나, 본 보고서에서는C3Ghost를 다루지 않았기 때문에 이 구조에서의 양자화 손실에 대해 파악할 수 없었다. 따라서 향후 계획으로C3Ghost 기법을 포함하여 Bottleneck 및 C3, C2f 구조 조합에서 발생하는 양자화 손실을 추가적으로 분석하고 보고서를 보완할 계획이다.

또한, 현재 Bottleneck 반복 수에 따른 구조 별 양자화 손실에 대한 내용을 다룬 결과를 바탕으로, N모델에 적용하여 실제로 성능 향상이 이루어지는지 확인하고자 한다. 이를 통해 Bottleneck 반복 수가 최적화된 특징 추출을 지원하는지를 검증하고, 최적의 구조적 조합을 찾아 성능 향상을 실현하는 것을 목표로 한다.