**v5n, v8n 분석 보고서**

1. **서론**
   1. **배경**
      1. 커널 사이즈와 소형 객체 검출 성능의 상관관계  
         : 일반적으로 특징 추출을 위한 컨벌루전 연산에서 커널 사이즈는 객체의 크기와 밀접하게 연관되어 있다. 예를 들어 크기가 작은 커널(ex. 3x3)은 소형 객체(이미지에서 차지하는 영역이 작은 객체)의 특징들을 추출하는데 유리하며, 크기가 큰 커널(ex. 7x7)은 대형 객체의 특징을 추출하는데 유리하다.
      2. yolov5와 v8의 구조적 차이  
         : v5와 v8의 구조적인 차이를 살펴봤을 때 두가지가 있었다. 먼저 처음 선언되는 Conv 레이어에서 v5는 kernel\_size를 6을 사용하며 v8은 3을 사용한다. 두번째 차이점은 v5는 C3 모듈을 사용하고 v8은 C2f 모듈을 사용한다는 점이다. 따라서 이 두가지 차이점이 모델에 성능에 미치는 영향을 분석하는 실험을 진행한다.
   2. **실험 준비**
      1. 실험 항목
         1. v8n\_org\_tv
         2. v8n\_k6\_tv
         3. v8n\_c3\_tv
         4. v5n\_org\_tv
         5. v5n\_k3\_tv
         6. v5n\_c2f\_tv
   3. **목적 : 이전 실험 v5n과 v8n의 성능 차이 분석**
      1. 커널 사이즈에 따른 성능 차이  
         : 실험1을 기준으로 커널 사이즈를 증가시킨 실험2를 비교한 결과와 실험4를 기준으로 커널 사이즈를 3으로 낮춘 실험5를 비교해서 커널 사이즈에 따른 성능 차이 분석한다.
      2. C2f 모듈과 C3 모듈이 성능에 미치는 영향 분석  
         : 내부에 C2f 모듈이 있는 실험1을 기준으로 C3 모듈로 변경한 실험3을 비교한 결과와 내부에 C3 모듈이 있는 실험4를 기준으로 C2f 모듈로 변경한 실험6을 비교한 결과를 통해 두 모듈이 적용될 때 성능에 미치는 영향을 분석한다.
   4. **평가 지표**
      1. 객체 크기별 정확도
      2. Parameter 수, NPU/GPU 비율, NPU 성능, GPU 성능, FPS

1. **본론**
   1. **실험 설계**
      1. 데이터셋
         1. 동일한 데이터로 학습한 모델에 대해 동일한 테스트 데이터를 사용하여 모델 별 성능 측정.
            1. 학습 데이터 : 제공받은 train set과 validation set을 병합
            2. 테스트 데이터 : 제공받은 test set
      2. 실험 설계

\* org는 기본 Yolo모델(original)  
\* k3, k6는 각각 첫번째 Conv의 커널 사이즈가 3, 6인 모델  
\* c3와 c2f는 내부 모듈이 변경된 경우

* + - 1. v8n\_org : v8n 기본 모델
      2. v8n\_k6 : v8n 기본 모델의 첫 Conv 레이어에 k = 6 적용
      3. v8n\_c3 : v8n 기본 모델의 C2f 모듈을 모두 C3로 교체
      4. v5n\_org : v5n 기본 모델 (anchor free)
      5. v5n\_k3 : v5n 기본 모델의 첫 Conv 레이어에 k = 3 적용
      6. v5n\_c2f : v5n 기본 모델의 C3 모듈을 모두 C2f로 교체
  1. **실험 결과 및 분석**
     1. 실험 1 Conv 첫 레이어 kernel size에 따른 성능
        1. 대조군 : v8n\_org, v5n\_org  
           실험군 : v8n\_k6 , v5n\_k3
        2. **가설 - 커널 사이즈가 작으면 소형 객체에 대한 성능이 증가할 것이다.**  
           : [1.0 데이터셋과 모델 성능 상관관계 분석 보고서]에서 v8n\_org 모델이 v5n\_org 모델과 비교했을 때 전체적인 성능이 높았으며, 소형 객체에 대한 성능 역시 높은 것을 확인할 수 있었다. 이에 대한 원인 중 하나가 커널 사이즈의 차이라 판단하여 위와 같은 가설을 세우고 실험을 진행하게 되었다.
        3. 결과 및 분석
           1. v8n\_org vs v8n\_k6
           2. v5n\_org vs v5n\_k3
     2. 실험 2 Bottleneck 반복 수에 따른 연산 양자화 손실  
        1. 비교 모델 :
           1. v8n\_2442 vs v8n\_6886
           2. v5n\_2442 vs v5n\_6886
        2. 모델 설명 :
           1. v8n\_2442 / v8n\_6886 : 기본 yolov8n 모델에 bottleneck 반복 수 2, 4, 4, 2를 설정한 모델
           2. v5n\_2442 / v5n\_6886 : 기본 yolov5n 모델에 bottleneck 반복 수 2, 4, 4, 2를 설정한 모델
        3. 가설   
           : **Bottleneck** 반복 수가 증가하면 모델의 연산 복잡도가 늘어나고 레이어 간 연결과 정보의 흐름이 깊어진다. 이는 8bit 연산에서 양자화 손실을 증가시킬 가능성이 있다[2]. 따라서 A, B 두 경우 모두 6886 모델이 2442 모델보다 보존율이 떨어질 것이다.
        4. 결과 :
        5. 분석 :
     3. 실험 3 shortcut 활성화에 따른 C2f, C3, C3Ghost 성능 비교  
        1. 비교 모델 :
        2. 모델 설명 :
           1. a
           2. b
        3. 가설 :
           1. Shortcut=True(Backbone)  
              : shortcut=True 설정에서 원본 입력이 마지막에 concatenate 되는데 정보 간 차이가 커질 가능성이 높아 손실이 많아질 것이다.  
              - C2f의 경우 concat하는 채널 개수가 많아 연산량이 많기 때문에 손실이 클 것이다[3].   
              - GhostBottleneck은 Bottleneck보다 효율적인 연산량 감소를 목표로 하지만, 더 많은 연산 단계를 가지는 특성상 양자화 손실이 커질 가능성이 있다[4].   
              - 따라서 손실량은 C2f > C3Ghost > C3 순서로 나타날 것이다.
           2. Short=False(Neck)  
              : shortcut=False 설정에서는 원본 입력이 포함되지 않더라도, shortcut=True일 때와 마찬가지로 손실량은 C2f > C3Ghost > C3 순서로 나타날 것이다.
        4. 결과 :
        5. 분석 :
     4. 실험 4 Bottleneck 반복 수와 C2f, C3, C3Ghost의 최적 조합 비교  
        1. 비교 모델 :
        2. 모델 설명 :
           1. C2f\_2442, C3\_2442, C3Ghost\_2442
           2. C2f\_6886, C3\_6886, C3Ghost\_6886
        3. 가설 :
           1. Bottleneck 반복 수가 증가하면 레이어 간 연결과 정보 흐름이 깊어진다. 하지만 이는 8bit 양자화 손실을 일으키며, 특히 더 많은 레이어와 concat 연산을 가진 구조에서 손실이 더 커질 것이다.   
              - 2442 반복에서는 연산 복잡도가 낮아 구조의 특성이 더 크게 작용하므로, C3 > C3Ghost > C2f 순서로 성능이 좋을 것이다.  
              - 6886 반복에서는 반복 수 증가로 인해 양자화 손실의 영향을 크게 받아, 손실량이 C2f> C3Ghost > C3 순으로 일어날 것이다.
           2. 근거  
              - 실험 2에서 Bottleneck 반복 수가 증가할수록 양자화 손실이 커지는 경향이 관찰되었다.  
              - C2f는 concat 연산 개수가 많고 C3Ghost는 Ghost Module 특성으로 내부 연산 단계가 많아, 반복 수 증가 시 양자화 손실에 더 취약하다.  
              - C3는 상대적으로 단순한 구조로, 반복 수 증가의 영향을 덜 받을 가능성이 있다.  
              따라서 반복 수가 많을 때와 적을 때 모두 구조 특성상 단순한 C3의 연산 양자화 손실이 가장 적고, C2f의 손실이 가장 클 것이다.
        4. 결과 :
        5. 분석 :
     5. 실험 5 Conv와 GhostConv의 성능 차이  
        1. 비교 모델 :
        2. 모델 설명 :
           1. a
           2. b
        3. 가설   
           : GhostConv는 Conv와 다르게 concat 연산이 추가되므로, 8bit 연산에서 정보 손실이 더 클 가능성이 있다..

Conv는 상대적으로 간단한 연산 구조로, 양자화 연산에서의 손실이 적을 것이다.  
따라서 Ghostconv가 Conv보다 연산 양자화 손실이 클 것이다.

* + - 1. 결과 :
      2. 분석 :
    1. 실험 6 Conv, GhostConv와 C2f, C3, C3Ghost의 최적 조합 비교  
       1. 비교 모델 :
       2. 모델 설명 :
          1. a
          2. b
       3. 가설   
          : 위의 실험들을 통해 C2f와 GhostConv에서 가장 연산 양자화 손실이 크다는 것을, C3와 Conv에서 가장 손실이 작다는 것을 알게 되었다. 이를 토대로 조합별 양자화 손실은 아래와 같을 것이다.  
          - 가장 손실이 많은 조합 : C2f + GhostConv  
          - 중간 정도의 손실이 발생하는 조합: C3 + GhostConv, C2f + Conv  
          - 가장 손실이 적은 조합: C3Ghost + Conv
       4. 결과 :
       5. 분석 :
    2. 실험 7 채널 수 증가에 따른 연산 양자화 손실  
       1. 비교 모델 :
       2. 모델 설명 :
          1. a
          2. b
       3. 가설   
          : 채널 수는 모델이 처리할 수 있는 연산 복잡도에 영향을 미친다. 채널 수가 증가하면 더 많은 정보를 연산을 통해 담아야 하므로, 양자화 손실이 더 커질 가능성이 있다. 채널 수가 많은 구성(256, 512, 1024)이 채널 수가 적은 구성(128, 256, 512)보다 더 큰 양자화 손실을 유발할 것이다.  
          따라서 양자화 손실은 채널 수가 더 많은 모델이 클 것이다.
       4. 결과 :
       5. 분석 :

1. **결론**
   1. 주요 결론
      1. C2f vs C3G vs C3
      2. Conv vs GhostConv
      3. Bottleneck
   2. 최적 조합
   3. 향후 연구 방향
2. **참조 문헌**

[1] J. Fernandez-Marques, P. N. Whatmough, A. Mundy, and M. Mattina, "Searching for Winograd-aware Quantized Networks," *Proc. of the 3rd MLSys Conf.*, arXiv:2002.10711, 2020.

[2] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," *arXiv preprint arXiv:1712.05877*, 2017.

[3] <https://github.com/orgs/ultralytics/discussions/15762>

[4] https://pytorch.org/docs/stable/quantization-accuracy-debugging.html