**C2f, C3, C3Ghost 분석 보고서**

1. **서론**
   1. **배경**
      1. Neck에서의 C모듈 변경에 따른 보존율의 성능 변화 분석  
         : YOLOv5 모델에서 사용되는 “C3” 모듈, YOLOv8 모델에서 사용되는 “C2f” 모듈, C3 모듈에 GhostBottleneck을 적용한 “C3Ghost” 모듈은 모두 아키텍처가 상이하다. [분석 보고서 2.1.1]에서 분석한 Conv와 GhostConv 각 모듈에 대해 Neck에서 모듈(C2f, C3, C3Ghost)을 변경해가며 실험을 진행한다.  
         본 보고서는 모듈 변경에 따라 NPU / GPU 비율인 “보존율”에 대해 성능 변화를 분석한다.
   2. **실험 준비**
      1. 실험 모델
         1. sP2\_B c c2f\_N c c2f
         2. sP2\_B c c2f\_N c c3
         3. sP2\_B c c2f\_N c c3g
         4. sP2\_B c c2f\_N gc c2f
         5. sP2\_B c c2f\_N gc c3
         6. sP2\_B c c2f\_N gc c3g
   3. **목적**
      1. 실험 1 : Neck에서 “C” 모듈 변경에 따른 성능 변화 분석  
         빛, 조명, 예술이(가) 표시된 사진

         자동 생성된 설명 다채로움, 빛, 예술이(가) 표시된 사진

         자동 생성된 설명 **스크린샷, 다채로움, 그린, 예술이(가) 표시된 사진

         자동 생성된 설명** [C2f] [C3] [C3Ghost]  
         : C2f, C3, C3Ghost 모듈에 대한 구조를 간단하게 설명한다.  
         - C2f : nn.ModuleList()를 사용하기 때문에 반복 횟수 조정에 따라 입력의 절반 채널에 대해 Bottleneck을 반복하여 생성되는 tensor들의 개수가 정해지는 방식이다. 위 그림과 같이 이를 모두 Concatenation 하기 때문에 반복 횟수를 늘릴수록 더 많은 tensor들의 정보가 담긴다.  
           
         - C3 : nn.Sequential()을 사용하기 때문에 반복 횟수 조정에 따라 입력의 절반 채널에 대해 Bottleneck을 순차적인 흐름으로 반복하는 방식이다. 위 그림과 같이 Concatenation은 2개의 input만 받기 때문에 C2f에 비해 더 적은 tensor들의 정보가 담긴다.  
           
         - C3Ghost : C3 모듈을 상속받아 사용하기 때문에 방식은 동일하다. 다만 Bottleneck 대신 GhostBottleneck을 사용하는데 GhostBottleneck은 Conv 모듈 대신 GhostConv 모듈을 사용하여 Hidden Channel을 통해 두번의 특징 추출을 한다. 절반 채널에 대해서 주요 feature를 추출한 후 간단한 feature를 추가로 추출하여 채널을 늘리는 연산을 하기 때문에 경량화에 이점이 있다.  
           
         [모델 1]을 기준으로 Neck에서 C3로 변경한 [모델 2]와 성능 변화를 분석한다.  
         [모델 1]을 기준으로 Neck에서 C3Ghost로 변경한 [모델 3]와 성능 변화를 분석한다.  
         [모델 2]를 기준으로 [모델 3]과 성능 변화를 분석한다.
   4. **평가 지표**
      1. 모델 성능 : GPU mAP50, NPU mAP50, 보존율(NPU mAP50 / GPU mAP50)
      2. Parameter 수
2. **본론**
   1. **실험 설계**
      1. 데이터셋
         1. 각 모델들이 학습에 사용된 데이터와 테스트에 사용된 데이터는 동일
            1. 학습 데이터 : 제공받은 train set과 validation set을 병합
            2. 테스트 데이터 : 제공받은 test set
      2. 모델 설계

\* sP2\_B c c2f\_N c c2f에서 B와 N은 Backbone과 Neck을 의미.  
\* B c c2f는 Backbone에서 Conv와 C2f 모듈 사용을 의미  
\* N c c2f는 Neck에서 Conv와 C2f 모듈 사용을 의미

* + - 1. sP2\_B c c2f\_N c c2f : YOLOv8s 기본 모델에 P2 Layer를 추가한 모델
      2. sP2\_B c c2f\_N c c3 : 위와 동일하고 Neck에서 C3 모듈로 변경한 모델
      3. sP2\_B c c2f\_N c c3g : 위와 동일하고 Neck에서 C3Ghost 모듈로 변경한 모델
      4. sP2\_B c c2f\_N gc c2f : YOLOv8s 기본 모델에 P2 Layer를 추가하고 Neck에서 GhostConv 모듈로 변경한 모델
      5. sP2\_B c c2f\_N gc c3 : 위와 동일하고, Neck에서 C3 모듈로 변경한 모델
      6. sP2\_B c c2f\_N gc c3g : 위와 동일하고, Neck에서 C3Ghost 모듈로 변경한 모델
  1. **실험 결과 및 분석**
     1. 실험 1 : Neck에서 Conv 모듈 사용과 “C” 모듈 변경에 따른 성능 변화 분석
        1. 실험 항목
           1. 대조군1 : sP2\_B c c2f\_N c c2f  
              실험군1 : sP2\_B c c2f\_N c c3, 실험군2 : sP2\_B c c2f\_N c c3g
        2. 결과  
           - 평가 기준 : 모델 성능

|  |  |
| --- | --- |
| 텍스트, 스크린샷, 직사각형, 도표이(가) 표시된 사진  자동 생성된 설명  [그래프 1] | 텍스트, 스크린샷, 직사각형, 도표이(가) 표시된 사진  자동 생성된 설명  [그래프 2] |
| 텍스트, 스크린샷, 직사각형, 다채로움이(가) 표시된 사진  자동 생성된 설명  [그래프 3] | [그래프 4] |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **v8s\_P2** | v8s\_P2\_N c c3 | v8s\_P2\_N c c3g |
| GPU mAP (%) | **67.82** | 66.71 | 66.39 |
| NPU mAP (%) | **48.34** | 45.86 | 46.16 |
| 보존율 | **71.28** | 68.75 | 69.53 |

* + - 1. 분석  
         : 위 그래프는 Neck에서 Conv 모듈을 기준으로 C 모듈 변경에 따른 평가 지표를 나타낸다.  
          실험 결과, Neck에서 Convolution을 사용했을 때 C2f 모듈이 Parameter 수는 가장 많지만 모델 성능 평가 지표에서 가장 높은 성능을 보였다. 반면 [그래프 2]와 [그래프 3]에서 C3의 경우 C3Ghost보다 GPU mAP에서 0.32 증가했지만 NPU mAP에서 0.3 감소했다.  
          실험 결과를 통해 [서론 C.목적]에서 언급한 바와 같이 C2f 모듈은 많은 tensor들의 정보가 담기기 때문에 객체에 대한 Feature 추출에 강점이 있을 것이라는 인사이트를 도출했다. 반면 C3Ghost는 GhostBottleneck 연산 과정에서 GhostConv를 사용하여 Conv 모듈을 사용하는 C3 모듈보다 가벼운 연산 과정을 거치기 때문에 int8 연산을 다루는 NPU에서 정보 손실이 적었을 것이고 그로 인해 NPU mAP와 보존율 성능에서 높았을 것이라는 분석 내용이 도출되었다.  
          따라서, 모델 간 Parameter 수 차이가 크지 않고 모델 성능 평가 지표에서 가장 높은 성능을 보이는 C2f 모듈이 Conv 모듈과 조합하여 사용하기에 가장 적합한 모듈이다.
    1. 실험 2 : Neck에서 GhostConv 모듈 사용과 “C” 모듈 변경에 따른 성능 변화 분석
       1. 실험 항목
          1. 대조군1 : sP2\_B c c2f\_N gc c2f  
             실험군1 : sP2\_B c c2f\_N gc c3, 실험군2 : sP2\_B c c2f\_N gc c3g
       2. 결과  
          - 평가 기준 : 모델 성능

|  |  |
| --- | --- |
| 텍스트, 스크린샷, 직사각형, 도표이(가) 표시된 사진  자동 생성된 설명  [그래프 1] | 텍스트, 스크린샷, 번호, 라인이(가) 표시된 사진  자동 생성된 설명  [그래프 2] |
| 텍스트, 스크린샷, 다채로움, 번호이(가) 표시된 사진  자동 생성된 설명  [그래프 3] | [그래프 4] |

|  |  |  |  |
| --- | --- | --- | --- |
|  | v8s\_P2\_N gc c2f | v8s\_P2\_N gc c3 | **v8s\_P2\_N gc c3g** |
| GPU mAP (%) | 67.82 | 67.53 | **65.18** |
| NPU mAP (%) | 47.15 | 47.05 | **47.53** |
| 보존율 | 69.52 | 69.67 | **72.92** |

* + - 1. 분석  
         : 위 그래프는 Neck에서 Conv 모듈을 기준으로 C 모듈 변경에 따른 평가 지표를 나타낸다.  
          실험 결과, Neck에서 Ghost Convolution을 사용했을 때 GPU mAP에서는 C2f가 가장 높은 성능을 보였지만 Parameter 수 대비 NPU mAP와 보존율에서 C3G 모듈이 가장 높은 성능을 보였다.  
          실험 결과를 통해 C3Ghost는 GhostBottleneck 연산 과정에서 GhostConv를 사용하여 Conv 모듈을 사용하는 C3 모듈보다 가벼운 연산 과정을 거치기 때문에 int8 연산을 다루는 NPU에서 정보 손실이 적었을 것이고 그로 인해 NPU mAP와 보존율 성능에서 높았을 것이라는 분석 내용이 도출되었다.  
          따라서 GPU mAP는 가장 낮지만 Parameter 수 대비 NPU mAP와 보존율 측면에서 가장 높은 성능을 보이는 C3Ghost 모듈이 GhostConv 모듈과 조합하여 사용하기에 가장 적합한 모듈이다.

1. **결론**
   1. **주요 결론**
      1. 실험 1 : Neck에서 Conv 모듈 사용과 “C” 모듈 변경에 따른 성능 변화 분석  
         : C2f 모듈은 Parameter 수가 가장 많지만 여러 tensor들의 정보를 가져오기 때문에 객체 특징 추출에 강점이 있어 Conv 모듈과 조합해서 사용하기에 가장 적합하다는 결론을 얻었다.
      2. 실험 2 : Neck에서 GhostConv 모듈 사용과 “C” 모듈 변경에 따른 성능 변화 분석  
         : C3Ghost 모듈은 Parameter 수가 가장 적고 GPU mAP에서 비교 모델들에 비해 낮은 성능을 보인다. 하지만 GhostBottleneck 모듈 사용으로 인해 연산량이 감소하여 int8 연산만 지원하는 NPU로 모델을 올리기 위해 변환하는 과정에서 정보 손실이 가장 적게 일어났기 때문에 NPU mAP와 보존율에서 가장 높은 성능을 보인다.  
          따라서 GhostConv 모듈과 조합해서 사용하기에 가장 적합하다는 결론을 얻었다.
   2. **향후 연구 방향**
      1. 실험 계획
         1. 앞선 보고서의 분석 내용들을 참고하여 Bottleneck 반복 횟수와 Convolution 모듈, C 모듈을 적절히 조합하여 양자화 과정에서 발생하는 정보 손실의 폭을 최소화할 수 있는 모델을 탐색한다.
      2. 필요 과제
         1. 본 보고서는 C2f, C3, C3Ghost 모듈에 대해서만 비교 분석을 하고 있다. 하지만 정확한 비교 분석을 위해서 C2fGhost 모듈도 실험에 추가하여 (C2f, C2fGhost)와 (C3, C3Ghost)의 차이를 분석하고 (C2f, C3)와 (C2fGhost, C3Ghost)의 차이를 분석하여 모듈에 대한 더 정확한 분석 이해가 필요하다.