YOLOv8s 모델 실험 분석 보고서

(Ghost Convolution)

이전에 P2 Layer를 추가한 모델 실험을 통해 2가지 연구 방향을 제시했다.

본 실험 보고서는 2가지 연구 방향 중 1번째 연구 방향인 ‘연산량 감소를 통한 모델 경량화’ 에 초점을 둔다.

v8s\_p2\_train 모델과 비교를 통해 실험을 진행한다.

* Model Information

|  |  |  |
| --- | --- | --- |
| 모델명 | | v8s\_p2\_train |
| 모델 사양 | | 195 layers  11.3M parameters  32.4 GFLOPs |
| mAP50 | GPU | 0.651 |
| NPU | 0.457 |
| FPS | | 50.21 |

* 성능 측정
  + mAP

텍스트, 스크린샷, 폰트, 블랙이(가) 표시된 사진

자동 생성된 설명

* + FPS : 50.21 / Inference time : 19.92 ms
  + Confusion Matrix Normalized

스크린샷, 직사각형, 도표, 텍스트이(가) 표시된 사진

자동 생성된 설명

1. 실험 개요
   1. 배경

이전 실험에서 P2 layer를 통해 성능 개선을 확인할 수 있었지만, GPU 대비 NPU 성능 복원률에 대해서는 큰 변화를 볼 수 없었다. 이를 통해 P2 레이어를 추가하여 소형 객체 탐지에 유리한 160x160 Image의 feature map을 추출할 때 발생하는 연산량 증가에 대해 연산 양자화 손실 또한 커졌다는 또 하나의 가설을 설정했다. 이 가설가설 주목할 것은 연산량 증감와 양자화 손실의 증감 폭이 비례하는지에 대한 것이다. 아직 이 부분에 대해서 상관관계를 밝혀내지 못했기 때문에 가설에서 파생된 2가지 연구 방향으로 실험을 진행한다.

첫번째 연구 방향은 Convolution 대신 Ghost Convolution, C2f 대신 C3Ghost 레이어를 사용하여 연산량을 줄여 모델 네트워크를 경량화 하는 것을 목표로 한다.

두번째는 연산량 증감에 따른 양자화 손실 증감 폭의 상관관계 연구다.

본 보고서는 첫번째 연구 방향인 Ghost Convolution, C3Ghost 레이어를 적용하여 기본 YOLOv8s에서 P2 레이어를 추가한 모델을 경량화 하는데 집중한다.

1. 모델 수정 전략 (가설 설정)
   1. v8s\_ghost\_c3ghost

기존 Conv 모듈은 입력 데이터의 모든 채널에 대해 필터를 적용하여 특징을 추출한다. 여기서 한 필터는 여러 채널에 걸쳐 적용되며, 필터의 수는 일반적으로 출력 채널 수에 대응한다. 예를 들어, 입력 채널이 256개이고 출력 채널이 128개라면, 각 출력 채널에 대해 256개의 입력 채널을 고려한 필터 연산이 이루어진다. 이로 인해 계산량이 커지고, 모델의 처리 속도가 느려질 수 있는 반면, **Ghost Convolution**은 입력 채널의 절반 정도만 표준 Conv Layer를 사용하고, 나머지 절반은 연산량이 적은 추가 Conv Layer를 사용한다. 이 추가 Conv Layer는 주요 특징을 추출하는 데 필요한 연산을 최소화하고, 간단한 feature map을 생성하여 채널을 확장하는 역할을 한다. 이렇게 하면, 기존의 Conv 모듈에서는 모든 채널에 대해 계산을 수행했던 것에 비해, Ghost Conv는 필요한 주요 특징을 적은 연산량으로 추출하면서도 전체적인 성능을 유지할 수 있다.

즉, **Ghost Conv**는 기존의 Conv보다 훨씬 적은 계산량으로 모델의 처리 속도를 향상시킬 수 있는 장점을 가진다. 이를 통해, 모델 네트워크에서 Neck 부분의 레이어를 추가하여 낮은 소형 객체 탐지 성능에 대해 대응할 수 있으며, Ghost Convolution을 사용하여 비슷한 성능을 유지하면서 줄어든 연산량으로 인해 모델이 경량화 될 것이라는 가설을 세우고 실험을 진행한다.

- **Reference**

* + [Huangfu, Z., Li, S., & Yan, L. (2024). Ghost-YOLO v8: An Attention-Guided Enhanced Small Target Detection Algorithm for Floating Litter on Water Surfaces. *School of Information Engineering, North China University of Water Resources and Electric Power*, 80(3), 3714-3728.](https://www.techscience.com/cmc/v80n3/57888)
  + [Zhou, H., Yu, Y., Wang, K., & Hu, Y. (2024). A YOLOv8-Based Approach for Real-Time Lithium-Ion Battery Electrode Defect Detection with High Accuracy. *Electronics*, 13(1), 173.](https://doi.org/10.3390/electronics13010173)

1. 실험 세팅
   1. 모델 구조

텍스트, 스크린샷, 도표, 번호이(가) 표시된 사진

자동 생성된 설명

* 1. 구조 설명

YOLOv8 모델의 Architecture를 보면 Backbone의 P3, P4, P5에서 Feature map을 추출하여 Neck에서 Concatenate을 한다. Backbone에서 추출한 Feature map은 다양한 크기와 해상도를 가지는데 Neck은 이러한 Feature map을 통합하고 보완하여 더 풍부한 정보를 생성하는 역할을 한다. 또한 다양한 크기의 객체를 탐지해야 하기 때문에 다중 해상도의 Feature map을 사용하여 소형 객체와 대형 객체 모두를 효과적으로 처리한다.

위 그림은 모델 네트워크를 수정한 구조다. Neck에서 Conv와 C2f를 GhostConv와 C3Ghost로 변경하였다. Backbone의 P2 (160x160)에서 추출한 Feature map과 Neck에서 추가한 P2(160x160) Layer를 Concatenate하여 바로 Head로 내보내지 않고, 다시 한번 Convolution Layer를 통해 80x80 Size로 Downsampling한다. 이후에 생성되어 있는 P3(80x80) Layer와 Concatenate하여 가설대로 소형 객체에 대해 더 많은 정보를 담은 Layer를 Head로 내보낸다. 이외의 Layer는 기본 YOLOv8s 구조와 동일하다.

1. 실험 결과
   1. Model information

|  |  |  |
| --- | --- | --- |
| 모델명 | | 8sP2\_BB-c^c2f\_NK-gc^c3g\_train |
| 모델 사양 | | 269 layers  8.9M parameters  26.6 GFLOPs |
| mAP50 | GPU | 0.636 |
| NPU | 0.432 |
| FPS | | 55.72 |

* 1. 성능 측정
     1. mAP

텍스트, 스크린샷, 폰트, 블랙이(가) 표시된 사진

자동 생성된 설명

* + 1. FPS : 55.72 / Inference time : 17.95 ms
    2. Confusion Matrix Normalized

스크린샷, 직사각형, 도표, 사각형이(가) 표시된 사진

자동 생성된 설명

1. 결과 및 비교
   1. 결과
      1. v8s\_p2\_train 클래스별 mAP50

all : 0.651

person : 0.661

car : 0.911

bus : 0.799

truck : 0.706

cycle : 0.217

motorcycle : 0.611

* + 1. 8sP2\_BB-c^c2f\_NK-gc^c3g\_train 클래스별 mAP50

all : 0.636

person : 0.644

car : 0.907

bus : 0.789

truck : 0.700

cycle : 0.160

motorcycle : 0.613

* 1. 비교 : v8s\_p2\_train vs 8sP2\_BB-c^c2f\_NK-gc^c3g\_train

|  |  |  |  |
| --- | --- | --- | --- |
|  | v8s\_p2\_train | 8sP2\_BB-c^c2f\_NK-gc^c3g\_train | 성능 |
| mAP (GPU) | 0.651 | 0.636 | 약 2.3% 감소 |
| mAP(NPU) | 0.457 | 0.432 | 약 5.7% 감소 |
| 복원률 | 약 70.2% | 약 68.9% | 약 1.3% 감소 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | v8s\_p2\_train | 8sP2\_BB-c^c2f\_NK-gc^c3g\_train | 성능 |
| FPS | 50.21 | 55.72 | 약 11% 증가 |

1. 결론 및 분석
   1. 분석
      1. 문제점

Ghost Convolution과 C3Ghost를 사용하여 연산량을 줄여 FPS가 약 11% 증가하였지만, 기존 v8s\_p2\_train 모델에 비해 mAP 성능이 GPU와 NPU에서 각각 2.3%, 5.7% 감소하였다. GPU 대비 NPU 성능 복원률 또한 약 1.3% 감소했다.

Ghost Convolution의 특징인 채널의 절반을 주요 feature 추출로 사용하는 방식에서 연산 양자화 손실이 발생했는지, C3Ghost 모듈을 사용하여 GhostBottleneck 모듈 내에서 연산 양자화 손실이 발생했는지에 대해서 문제점을 분석해 볼 필요가 있다.

* 1. 결론

Ghost Conv와 C3Ghost 모듈을 사용하여 모델 네트워크의 Layer는 늘었지만 Parameter가 줄은 것으로 보아 연산량이 대폭 감소했고, FPS가 11% 증가했다. 하지만 mAP(50)에서 GPU와 NPU에서 2.3%, 5.7%가 감소했다. 이를 통해 Ghost Convolution과 C3Ghost에서 쓰이는 GhostBottleneck 중 어느 모듈에서 연산 양자화 손실이 크게 발생했는지 분석해 볼 필요가 있다.

* 1. 향후 연구 방향

Ghost Convolution과 C3Ghost 모듈을 같이 적용하지 않고 따로 적용하여 어느 모듈에서 연산 양자화 손실이 크게 일어나는지 확인하는데에 초점을 둔다. Ultralytics에 정의되어 있는 Conv.py 모듈 중 Conv, GhostConv 두가지를 사용하고 block.py 모듈 중 C3, C3Ghost 모듈을 사용하여 (Conv, C3) / (Conv, C3Ghost) / (GhostConv, C3) / (GhostConv, C3Ghost)의 4가지 결과를 도출하고 성능 분석을 통해 연산 양자화 손실을 가장 높이는 모듈을 추적하여 개선 방안을 연구한다.