**활성화 함수 양자화 손실 분석 보고서**

1. **서론**
   1. 배경  
      텍스트, 스크린샷, 멀티미디어 소프트웨어, 그래픽 소프트웨어이(가) 표시된 사진

      자동 생성된 설명  
      aiWare Studio툴을 사용하여 모델의 레이어별 Maximum Relative Error를 확인했다.  
      많은 ‘mul’ 레이어에서 높은 에러를 발견하였고, YOLO모델의 기본 activation 함수인 ‘SiLU’ 모듈인 것을 파악했다. 이를 통해 activation 함수의 Quantization Error 폭을 최소화함에 목표를 두고 실험을 진행한다.
      1. 활성화 함수  
         silu(x)=x∗σ(x)  
         라인, 도표, 그래프이(가) 표시된 사진

         자동 생성된 설명  
         입력 값 x가 양수일때 x의 크기를 유지하면서 점진적으로 축소시키고, x가 음수일때는 그 값을 더 작게 만들어 준다. 많은 모델에서 기본 activation 함수로 사용되는 SiLU의 특징은 다음과 같이 요약할 수 있다.  
           
         1) 부드러운 비선형성 : ReLU와는 달리 SiLU는 연속적이고 미분 가능한 함수이기에 더 안정적인 학습을 제공할 수 있다.

2) 자연스러운 학습의 흐름 제공 : 음수 값 입력에 대해선 값을 완전히 억제하는(0으로 만들어버리는) ReLU와 달리, 약하게나마 값을 도출하여 정보 손실을 최대한 적게 만들어주기 때문에 학습에 안정성을 더할 수 있다.

* GPU 측면  
  - GPU는 Float32 연산(부동소수점 연산)을 지원하기 때문에 지수 함수와 나눗셈 연산 같은 많은 계산량을 처리하는데 효과적이다  
  - 모델이 더 많은 정보를 유지할 수 있어 학습의 안정성 측면에서 긍정적 영향을 미친다.
* NPU 측면  
  - INT8 연산을 지원하고 나누기 연산을 지원하지 않는 Apache5 NPU에서는 상대적으로 비효율적이다.  
  - 정수 기반 하드웨어에서 근사치로 처리해야 하는데, 이 과정에서 연산 오버헤드와 정확도 손실이 발생한다.

ReLU(*x*)=(*x*)+=max(0,*x*)  
라인, 도표, 그래프이(가) 표시된 사진

자동 생성된 설명  
ReLU는 입력값이 음수일 경우 출력을 0으로 설정하므로, 신경망에서 활성화되는 뉴런의 수를 줄인다. 이는 모델이 연산과 메모리 사용에서 더 효율적으로 작동하게 만든다. ReLU의 특징은 다음과 같이 요약할 수 있다.  
  
1) 계산 효율성 : ReLU는 단순히 max(0, x) 연산만을 사용하므로, 지수 함수나 나누기 연산과 같은 복잡한 계산을 필요로 하는 sigmoid 또는 SiLU 함수에 비해 상대적으로 연산 속도가 빠르다.

2) Gradient Vanishing 문제 완화 : 양수 값의 입력에 대해서 기울기가 항상 1로 유지되기 때문에, sigmoid와 같은 함수에서 발생하는 Gradient Vanishing 문제를 어느정도 완화할 수 있다.

* GPU 측면  
  - ReLU는 대규모 데이터셋 및 모델에서 처리 효율성을 유지하며, 병렬 계산 환경에 최적화되어 있다.  
  - max(0, x) 연산만 하는 단순한 수학적 연산으로, SiLU함수에 비해 계산량이 적다.
* NPU 측면  
  - 지수 함수나 나누기 연산을 하는 SiLU 함수와 다르게 정수 연산에 최적화되어 있고, 효율적이다.

ReLU6(*x*)=min(max(0,*x*),6)  
라인, 도표, 그래프, 텍스트이(가) 표시된 사진

자동 생성된 설명  
ReLU6는 ReLU 함수에 상한값을 6으로 제한하여 극단적으로 큰 출력값이 모델의 가중치 학습에 미치는 악영향을 방지한다. ReLU함수에서 min 함수를 한번 더 거치기 때문에 계산량은 약간 증가하지만 MinMax값이 정해져 있기 때문에 Outlier를 발생시키지 않는다.

* GPU 측면  
  - ReLU와 같이 연산이 경량화 되어있어 효율적이다.  
  - 극단적이 출력값으로 인해 학습에 악영향을 끼칠 수 있는 ReLU에 비해 값의 범위를 0과 6사이로 제한함으로써 연산 오버헤드를 줄이고 모델 안정성을 높인다.
* NPU 측면  
  - ReLU와 같이 정수 연산에 최적화되어 있어 효율적이다.  
  - 출력 범위를 제한함으로써 ReLU와 비교했을 때, 안정적이고 NPU에서의 Quantization Error의 폭을 줄일 수 있다.
  + 1. Maximum Relative Error  
       Maximum Relative Error : |Maximum Error| / |Maximum value|  
       텍스트, 스크린샷, 폰트이(가) 표시된 사진

       자동 생성된 설명

"Maximum Relative Error(MRE)”는 양자화된 값들 중에서 가장 큰 오차 값의 절대값을 원래의 값들 중 가장 큰 값의 절대값으로 나눈 후, 이를 백분율로 표현한 값이다. 이는 전체 데이터 값들이 분포한 범위 내에서 가장 멀리 떨어져 있는 값을 기준으로, 양자화 오류가 가장 큰 비율을 나타낸다.  
본 보고서에서 MRE는 Quantization Error를 측정하는 지표로 사용한다.  
MRE의 증감에 따른 Quantization Error를 확인하여 둘 간의 상관관계를 분석한다.

* 1. 실험 준비
     1. 실험 항목
        1. 8sP2\_B c c2f\_N c c3g\_silu
        2. 8sP2\_B c c2f\_N c c3g\_relu
        3. 8sP2\_B c c2f\_N c c3g\_relu6
  2. 목적 : SiLU와 ReLU / ReLU와 ReLU6 함수 양자화 손실 차이 분석
     1. SiLU함수와 ReLU함수의 양자화 손실 분석  
        : 대다수의 모델에서 기본으로 적용되는 Activation 함수인 SiLU 함수를 사용했을 때 NPU에서는 지원하지 않는 연산으로 인해 Quantization Error가 크게 발생할 수 있다.  
        이 에러 폭을 최소화하고자 NPU연산에 친화적인 ReLU 함수를 적용하여 성능을 개선하는데 목적을 둔다.
     2. ReLU함수와 ReLU6함수의 양자화 손실 분석  
        : ReLU 함수를 사용했을 때 NPU에서 생기는 양수 Outlier에 대해 Quantization Error가 크게 발생할 수 있다.  
        이 에러 폭을 조금 더 최소화하고자 ReLU 함수에 상한 값을 6으로 적용한 ReLU6를 사용하여 NPU 연산 과정에서 생기는 Outlier에 대한 악영향을 방지하는데 목적을 둔다.
  3. 평가 지표
     1. Maximum Relative Error
     2. 보존율 (NPU mAP / GPU mAP)
     3. NPU mAP
     4. GPU mAP

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.  
           : kernel size ->   
           Convolution의 kernel size가 커질수록 연산 복잡도가 증가하여 NPU에서 양자화 손실이 커질 가능성이 있다. Kernel size가 커지면 필터가 더 넓은 범위의 정보를 학습할 수 있지만, 각 연산에서 사용되는 값의 범위가 넓어져 양자화 시 표현할 수 없는 정보가 증가할 수 있다[1].   
           따라서 커널 크기가 큰 모델의 연산 양자화 손실이 클 것이다.
        3. 결과 :
        4. 분석 :
     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