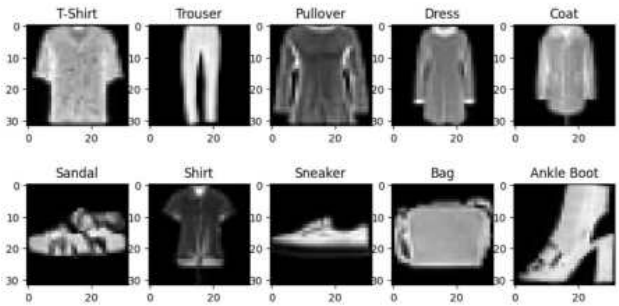
23-2 Embedded AI Final Project

20216240 김예원

목차

1. 수행한 task

torchvision에 있는 FashionMNIST 데이터 셋을 이용해 10개의 의류 카테고리에 대한 분류를 진행했다. FashionMNIST는 (224,224) 크기의 70,000개의 흑백이미지 데이터가 10개의 의류 카테고리로 분류되어있다. 10개의 의류 카테고리와 샘플 이미지는 다음과 같다.



- T-shirt	-Sandal
- Trouser	-Shirt
- Pullover	-Sneaker
- Dress	-Bag
- Coat	-Ankle Boot

2. 모델 소개 (LeNet-5)

본 과제에서는 LeNet-5 모델을 사용하였다. LeNet-5를 선택한 이유는 이 모델이 비교적 간단하면서도 높은 정확도를 보이기 때문이다. 따라서 이번 과제의 핵심인 경량화 기법 비교실험에 집중할 수 있어 적합하다고 판단하였다.

```
class LeNet_5(nn.Module):
    def __init__(self):
        super(LeNet_5, self).__init__()

        self.conv1 = nn.Conv2d(1, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16*5*5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(x.size(0), -1)

        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)

        return x
```

3. 경량화 기법

경량화 기법은 1) pruning과 2) quantization을 진행했다.
pruning 기법은 한번에 pruning을 진행하지 않고 조금씩 iterative하게 진행하였고, fine-tuning 없이 pruning만 진행한 경우와 pruning 후 fine-tuning으로 성능회복을 한 후에 다시 pruning을 진행하는 두 방식을 비교하였다.
pruning ratio는 50%, 60%, 70%, 80%, 90%로 설정하고 실험하였다. quantization은 torch에서 qint8 quantization만 지원하여 qint8로 quantization을 하였다.
pruning과 quantization을 하기 전 train, test, pruning, pruning+fine-tuning, quantization 함수를 만들어 진행하였다.

다음은 사용한 함수들에 대한 코드와 설명이다.
* (1) experiment.py : code
LeNet-5의 경량화를 비교 실험한 전체 코드이다.

(2) train.py : code
train 함수이며, batch size 32, epoch 5, learning rate 1e-3, criterion CrossEntropyLoss, Adam optimizer를 사용하였다.

(3) test.py : code
test 함수이다. test 함수에서는 모델 inference 후 accuracy, inference time과 함께 모델의 총 파라미터수와 MACs (Multiply-ACcumulates)를 계산한다. 모델의 총 파라미터수와 MACs에 대한 계산은 tp.utils.count_ops_and_params() 함수를 이용한다.

(4) pruning.py : code
pruning ratio를 입력받아 Pruning하는 코드이다. prunig은 tp.pruner.MagnitudePruner를 사용하였다. 적용한 Magnitude based pruning은 각 가중치의 절댓값 크기를 기준으로 중요도를 측정하여, 가중치가 낮은 채널들을 설정한 pruning ratio만큼 삭제하여 경량화하는 방식이다. LeNet-5에서는 layer 중 마지막 레이어인 fc3를 제외한 [conv1, conv2, fc1, fc2]의 레이어에서 가중치가 낮은 채널들을 제거한다. pruning ratio는 50%, 60%, 70%, 80% 90%로 설정한다. pruning ratio에 따른 비교에는 1) accuracy, 2) MACs, 3) 총 파라미터 수 (n_params), 4) inference time을 고려하였다.

(5) finetuning.py : code
pruning 직후 낮아진 성능회복을 위한 fine-tuning 코드이다. 이때 pruning된 모델은 이미 학습이 완료되어 수렴한 상태이기 때문에 train 함수에서 사용한 learning rate(1e-3) 보다 낮은 1e-4로 설정한다.

(6) pruning_finetuning.py : code
pruning과 fine-tuning을 함께 진행하는 코드이다.

(7) quantization.py : code
pytorch의 dynamic quantization을 사용하여 quantization을 수행한다. LeNet 모델 구조에서 convolution layer보다 linear layer가 더 많은 가중치를 포함하기 때문에 linear layer에 대해서 quantization을 진행한다. 본래 floating point 타입의 실수값을 정수로 변환할 때는 실수 값에 어떤 배율을 곱하여 그 결괏값을 정수로 반올림한다. 이때 dynamic quantization은 모델이 데이터의 범위를 확인하고 동적으로 배율을 결정하는 것이다. quantization을 통해 linear layer 파라미터의 데이터타입이 FP32에서 INT8로 변환된다.

(8) jetson.py : code
실제 JetsonNano에서 동작하기 위한 카메라 실행 코드이다.

* 실험 환경: quantization을 제외한 pruning 비교 실험은 NVIDIA RTX 6000 Ada Generation에서 진행했다. GPU에서 실험 시 inference time의 단위는 μs (마이크로세컨즈)이다.

3.1 Pruning

1) Finetuning 시 learning rate에 따른 성능 비교
2) Iterative Pruning 단계 수에 따른 성능 비교
3) Layer별 Pruning 적용 범위에 따른 성능 비교
4) Pruning 후 Fine-tuning 여부에 따른 성능 비교
5) Pruning 비율에 따른 메모리 사용량 비교
6) Pruning 비율에 따른 정확도, 파라미터 수, MACs, 추론 시간 비교

1) Finetuning 시 learning rate에 따른 성능 비교

	Accuracy(%)
Base	89.0
lr *1/10	88.45
lr *1/100	83.07
lr *1/1000	69.6

▲ learning rate에 따른 성능 결과

모델 파라미터에 Pruning을 적용한 후, Fine-tuning을 통해 성능 회복을 유도해야 한다. 이때 Fine-tuning 시 사용하는 learning rate은 일반적으로 기존 학습에서 사용된 learning rate보다 작게 설정된다. 이는 모델이 이미 수렴된 상태에서 일부 파라미터가 제거되었기 때문에, 큰 변화 없이 안정적으로 성능을 회복하려면 작은 learning rate가 필요하다. 큰 learning rate은 성능 회복 과정에서 큰 변동을 일으킬 수 있기 때문에 오히려 성능을 저하시킬 위험이 있다. 실험에는 pruning ratio 0.5, iterative step 5를 사용하였다. 위 표에서 확인할 수 있듯이, 기존 학습에서 사용된 learning rate의 1/10을 사용했을 때 성능 감소가 가장 적다는 것을 알 수 있다.

2) Iterative Pruning 단계 수에 따른 성능 비교

	Accuracy(%)
Base	89.3
one step	86.52
three step	87.27
four step	88.16
five step	88.45

▲ pruning 단계 수에 따른 성능 결과

Pruning을 적용할 때는 한 번에 설정된 Pruning 비율만큼 파라미터를 줄이지 않고, 설정된 비율의 일부만 Pruning을 진행한 후 Fine-tuning을 통해 성능을 회복시킨다. 이후 다시 일정 비율로 Pruning을 진행하고 Fine-tuning을 반복하여, 이 과정을 여러 단계에 걸쳐 iterative하게 진행한다. 위 표는 성능 회복을 위한 Fine-tuning 과정에서 Pruning 단계 수에 따른 성능 변화를 비교한 것이다. 실험에는 pruning

ratio 0.5, finetuning learning rate 1e-4를 사용하였다.Pruning을 한 번 또는 두 번만 적용한 경우보다, 여러 번에 걸쳐 Iterative Pruning을 적용한 경우 성능이 더 높았음을 확인할 수 있다.

3) Layer별 Pruning 적용 범위에 따른 성능 비교

	Accuracy(%)
Base	89.3
conv +linear pruning	88.45
only conv pruning	89.48
only linear pruning	93.28

▲ Layer별 pruning 적용 결과

사용한 LeNet-5 모델에는 convolution layer와 linear layer가 존재한다. Pruning은 convolution layer와 linear layer 각각에 적용할 수 있으며, 두 레이어에 관계없이 동시에 Pruning을 적용할 수도 있다. 단, 마지막 linear layer는 Pruning을 적용하면 차원 불일치로 인해 오류가 발생할 수 있기 때문에, 마지막 linear layer는 제외하고 실험을 진행하였다. 성능 비교 결과, linear layer만 Pruning을 적용한 경우, convolution layer만 Pruning을 적용한 경우, 그리고 convolution layer와 linear layer를 동시에 Pruning한 경우 순으로 성능이 높았다. 특히, linear layer만 Pruning을 적용했을 때 Pruning을 적용하지 않은 기본 LeNet-5보다 성능이 더 높았는데, 이는 기본 LeNet-5 모델이 완전히 최적화되지 않았기 때문에 Pruning을 통해 성능이 개선된 것으로 추측된다.

4) Pruning 후 Fine-tuning 여부에 따른 성능 비교

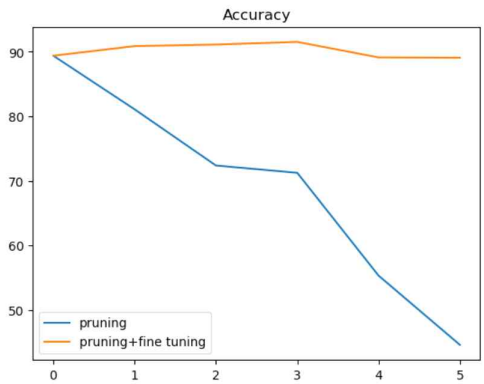
다음은 pruning ratio를 50%로 설정 후 pruning만 진행한 경우와 pruning 후 fine-tuning까지 진행한 경우의 실험결과이다.

	acc(%)	n_params	MACs	inference time(μs)
Base	89.39	61706	429342	8.864093
step 1	81.09	48737	330483	6.616259
step 2	72.39	37391	243147	6.565452
step 3	71.26	29919	225576	6.437826
step 4	55.339996	21194	153236	6.777787
step 5	44.62	15738	140156	7.132196

▲50%- pruning만 진행

	acc(%)	n_params	MACs	inference time(μs)
Base	89.39	61706	429342	8.905506
step 1	90.88	48737	330483	6.618619
step 2	91.119995	37391	243147	6.401491
step 3	91.53	29919	225576	6.377864
step 4	89.13	21194	153236	6.710196
step 5	89.08	15738	140156	7.022309

▲50%- pruning 후 fine-tuning 진행



▲50% pruning ratio에서 pruning과 pruning+fine-tuning에 따른 accuracy 시각화

Pruning만 진행했을 때는 accuracy가 최대 44%까지 급격하게 감소하는 모습을 확인할 수 있었다. 이는 중요한 파라미터들이 제거되면서 모델 성능이 크게 저하된 결과로 볼 수 있다. 하지만 Pruning 후에 Finetuning을 함께 진행했을 때는 accuracy가 거의 감소하지 않았다. 이는 Pruning으로 인해 손실된 성능이 Finetuning을 통해 회복되었기 때문이다.

5) Pruning 비율에 따른 메모리 사용량 비교

pruning ratio	
0.5	기준
0.6	-0.07MB
0.7	-0.12MB
0.9	-0.15MB

▲ pruning ratio별 메모리 사용량 결과

Pruning 비율에 따른 메모리 사용량 비교 결과는 다음과 같다. Pruning Ratio 0.5를 기준으로, Pruning Ratio 0.6에서는 메모리 사용량이 0.07MB 감소했고, Pruning Ratio 0.7에서는 0.12MB, 그리고 Pruning Ratio 0.9에서는 0.15MB가 감소했다. Pruning Ratio가 높아질수록 더 많은 파라미터가 제거되어 모델의 메모리 사용량이 줄어드는 것을 확인할 수 있었다. 이는 메모리 효율성을 향상시키는 데 있어 Pruning의 이점을 보여준다.

6) Pruning 비율에 따른 정확도, 파라미터 수, MACs, 추론 시간 비교

이후 60%, 70%, 80%, 90%에 대해서 pruning 후 fine-tuning을 진행하여 성능을 비교하였다. pruning ratio에 따른 결과는 다음과 같다. iterative하게 5번 나눠서 pruning을 진행하였고 이를 인덱스로 나타내어 표현했다.

	acc(%)	n_params	MACs	inference time(μs)
Base	89.39	61706	429342	7.788301
step 1	90.869995	47227	328973	6.770277
step 2	91.08	35143	240899	6.590343
step 3	89.19	24535	164201	6.649327
step 4	88.77	16297	140715	7.348084
step 5	86.27	9563	82741	6.719995

▲pruning ratio 60%

	acc(%)	n_params	MACs	inference time(μs)
Base	89.39	61706	429342	7.840371
step 1	90.84	43564	312736	7.291842
step 2	90.79	30781	226438	6.388116
step 3	88.54	20206	152248	6.626034
step 4	86.71999	11839	90166	6.722331
step 5	77.11	4951	36789	6.647205

▲pruning ratio 70%

▲pruning ratio 80%

▲pruning ratio 90%

설정한 ratio에 따라 pruning 후 fine-tuning을 진행했을 때 ratio가 커질수록 모델의 파라미터수와 MAC값, accuracy 값이 크게 감소하는 것을 볼 수 있다. 하지만 inference time에서는 오히려 ratio가 커질수록 inference time이 증가하기도 하였는데, 이는 inference time의 단위가 μs(마이크로세컨즈)이기 때문에 inference time 간의 차이를 유의미하게 간주하지는 않았다.

	acc(%)	n_params	MACs	inference time(μs)
Base	89.39	61706	429342	9.086704
step 1	90.79	42148	311320	6.590295
step 2	90.26	26699	212257	7.376456
step 3	88.02	16297	140715	6.797910
step 4	84.17	7355	75384	6.139684
step 5	72.22	2406	31570	8.775806

	acc(%)	n_params	MACs	inference time(μs)
Base	89.39	61706	429342	8.341479
step 1	89.95	40787	256642	6.768656
step 2	88.78	24535	164201	8.812928
step 3	84.57	12607	90934	6.465292
step 4	73.74	4485	36323	6.166387
step 5	43.93	524	24340	6.163526

* 설정한 pruning ratio에 알맞게 pruning 되었는지 확인

또한, 앞서 진행한 pruning은 magnitude 기반으로 pruning ratio에 따라 가중치가 낮은 channel들을 삭제하는 방식이다. pruning ratio 별 모델들의 각 layer별 채널수의 합산을 통해 각 pruning ratio에 알맞게 pruning된 사실도 확인하였다.

	# of remaining channels	pruned ratio
Base	472	-
50% pruning	246	0.5
60% pruning	189	0.6
70% pruning	152	0.7
80% pruning	106	0.8
90% pruning	62	0.9

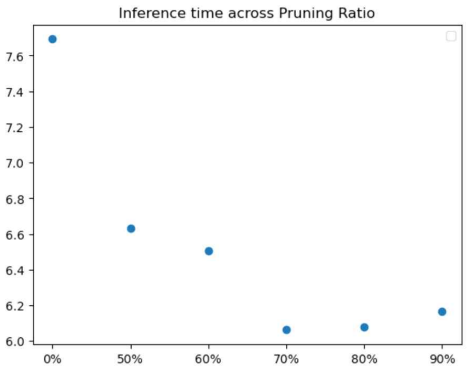
▲ pruning ratio 별 layer의 총 pruning된 채널수 확인

* pruning ratio에 따른 최종 pruning 결과

다음은 pruning ratio에 따라 iterative 하게 pruning 후 fine-tuning을 진행했을 때의 ratio에 따른 모델의 최종결과를 비교한 것이다.

	accuracy(%)	n_params	MACs	Inference time(μs)
기본 LeNet-5	89.39	61706	429342	8.91
50% pruning	89.08	15738	140156	7.02
60% pruning	86.27	9563	82741	6.72
70% pruning	77.11	4951	36789	6.65
80% pruning	77.22	2406	31570	8.78
90% pruning	43.93	524	24340	6.16

▲ pruning ratio에 따른 비교



▲pruning ratio에 따른 비교 시각화

전반적으로 pruning ratio가 증가함에 따라 accuracy와 params, MACs 값이 감소함을 확인할 수 있다. 단위가 마이크로세컨드(μs)인 inference time의 경우, pruning ratio가 70%일 때 6.65로 나타난 inference time을 기준으로 감소했다가 다시 증가하는 결과를 보인다. inference time의 단위가 μs인 것을 감안할 때 60% ~ 90%의 pruning ratio에서 모두 50% pruning의 inference time보다 작은 결과를 얻은 것을 보면 모델에서 더 큰 비율로 weight를 제거할수록 최적화된 구조로 인해 더 빠른 inference 속도가 가능해진다는 것을 알 수 있다.

* 최종 pruning ratio 선정

quantization을 진행할 모델의 pruning 비율을 선정하는 기준은 두 가지였다. 첫째, 모델의 정확도를 기본 LeNet-5 모델의 accuracy와 최대한 비슷하게 유지하는 것이었고, 둘째, 모델 사이즈를 n_params (총 파라미터수)와 MACs 기준으로 최대한 줄이는 것이었다. inference time은 단위가 마이크로세컨드(μs)임을 감안할 때 큰 차이가 없다고 판단하여 선정 기준에서 제외하였다.

	accuracy(%)	n_params	MACs	Inference time(μs)
기본 LeNet-5	89.39	61706	429342	8.91
50% pruning	89.08	15738	140156	7.02
60% pruning	86.27	9563	82741	6.72
70% pruning	77.11	4951	36789	6.65
80% pruning	77.22	2406	31570	8.78
90% pruning	43.93	524	24340	6.16

▲pruning ratio에 따른 비교

위 표에서 볼 수 있듯이, pruning 최종 모델 선정 기준 중 하나인 정확도에서 50% pruning과 60% pruning 간 차이는 약 3%였으나, 50% pruning 모델의 파라미터 수는 60% pruning 모델보다 약 1.6배 더 많았고 MACs 역시 약 1.7배의 큰 차이를 보였다. 이러한 결과를 종합적으로 고려하여 최종 pruning 모델로는 60% pruning이 적용된 모델을 선정하였다.

3.2) Quantization

(1) quantization 유무 비교 실험

Quantization은 60% pruning을 적용한 모델에 quantize_qint8() 함수를 이용해 진행하였다. Pytorch에서 지원하는 quantization은 GPU가 아닌 CPU에서만 동작하기 때문에, 해당 실험은 CPU에서 수행되었다. 모델 비교를 위해 60% pruning이 적용된 모델 역시 CPU에서 재추론을 진행하였다.

	60% pruning	60% pruning + quantization
Accuracy (%)	86.27	86.28
n_params	9563	(torchsummary: 358) 9563
MACs	82741	73536
Inference Time (μs)	13.7	18.16

▲ 60% pruning 모델의 quantization 진행 결과

Quantization을 진행한 결과, 모델의 accuracy는 86.27%에서 86.28%로 거의 변함없었고, inference time은 16.5μs에서 17.8μs로 변화되었다. 모델의 파라미터수는 9563개로 변함없었으며, MACs는 82741에서 73536로 크게 감소하였다.

Layer (type)	Output Shape	Param #
Conv2d-1	[32, 2, 28, 28]	52
MaxPool2d-2	[32, 2, 14, 14]	0
Conv2d-3	[32, 6, 10, 10]	306
MaxPool2d-4	[32, 6, 5, 5]	0
Linear-5	[32, 48]	7,248
Linear-6	[32, 33]	1,617
Linear-7	[32, 10]	340

Total params: 9,563
Trainable params: 9,563
Non-trainable params: 0

Input size (MB): 0.12
Forward/backward pass size (MB): 0.68
Params size (MB): 0.04
Estimated Total Size (MB): 0.85

▲ torchsummary-60% pruning 모델

Layer (type)	Output Shape	Param #
Conv2d-1	[32, 2, 28, 28]	52
MaxPool2d-2	[32, 2, 14, 14]	0
Conv2d-3	[32, 6, 10, 10]	306
MaxPool2d-4	[32, 6, 5, 5]	0
Linear-5	[32, 48]	0
Linear-6	[32, 33]	0
Linear-7	[32, 10]	0

Total params: 358
Trainable params: 358
Non-trainable params: 0

Input size (MB): 0.12
Forward/backward pass size (MB): 0.68
Params size (MB): 0.00
Estimated Total Size (MB): 0.81

▲ torch-summary-60% pruning+quantization

이 과정에서, quantization 후 torchsummary의 summary 함수를 통해 layer 파라미터 수를 확인하면, quantization이 완료된 모델의 linear layer 파라미터수가 모두 0으로 표시되는 현상이 있었다. 이는 quantization이 완료되면 기존 파라미터 저장방식이 변경되기 때문에 torchsummary에서 linear layer의 파라미터를 집계하지 못한 것으로 확인되었다. 각각의 quantization이 완료된 파라미터를 직접 계산해본 결과, 실제로 파라미터수에는 변화가 없음을 확인할 수 있었다.

float32 데이터를 int8로 quantization한 결과, 60% pruning을 적용한 모델보다 MACs 값이 확연히 감소한 것을 확인할 수 있었다. 이는 quantization 과정을 통해 모델의 연산에서 필요한 비트 수를 줄여, 이전보다 더욱 최적화된 모델이 되었음을 의미한다.

3.3) 최종 경량화 기법 비교

	Accuracy (%)	param size (KB)
Base	89.39	245
60% pruned	86.27	41
60& pruned + quantization	86.28	16

▲ 최종 경량화 모델 결과

경량화를 적용하지 않은 기본 LeNet-5 모델의 파라미터 크기는 245KB였으며, 60% pruning을 적용한 모델은 41KB로 base 모델 대비 83.27% 감소하였다. 또한, 60% pruning 이후 quantization을 적용한 모델은 16KB로 base 모델 대비 93.47% 감소를 보였다. 정확도 측면에서는 base 모델은 89.39%, 60% pruning 모델은 86.27%, 그리고 pruning과 quantization을 적용한 모델은 86.28%로 pruning과 quantization을 적용해도 정확도가 크게 손실되지 않음을 확인할 수 있었다.

4. 젯슨나노에서의 과정

젯슨 나노에 올린 최종 모델은 quantization을 진행하지 않고 60% pruning만 적용한 모델이다. 젯슨 나노에서 quantization까지 적용한 모델을 구동하려 했지만, torch의 quantization 라이브러리 문제로 인해 젯슨 나노에서는 실행되지 않아 quantization을 제외한 모델을 사용했다. 젯슨 나노의 카메라로 10개의 카테고리로 분류된 의류 이미지를 보여주면, 해당 옷의 예측 결과를 스피커를 통해 들을 수 있다.