모델경량화 4조

무려 20kg 감량

estoy a dieta

안현진 T1120

목차

1. 대회 개요

1. 대회 개요

$$score_{LB} = score_{MACs} + score_{F1}$$

$$score_{MACs} = \frac{\text{제출모델 MACs}}{\text{기준모델 MACs}}$$

$$score_{F1} = \begin{cases} 1 - \dfrac{\text{제출모델 F1score}}{\text{기준모델 F1score}} & \text{if 제출모델 F1score} < \text{기준모델 F1score} \\ 0.5 * (1 - \dfrac{\text{제출모델 F1score}}{\text{기준모델 F1score}}) & \text{if 제출모델 F1score} \geq \text{기준모델 F1score} \end{cases}$$

1. 대회 개요

- MACs 계산 방법

ptflops.flops_counter의 get_model_complexity_info 기반

torch.nnModules 기반 모듈에 hook을 걸어 계산하는 형태

in_channels, kernel_dims, output_dims의 shape으로 연산

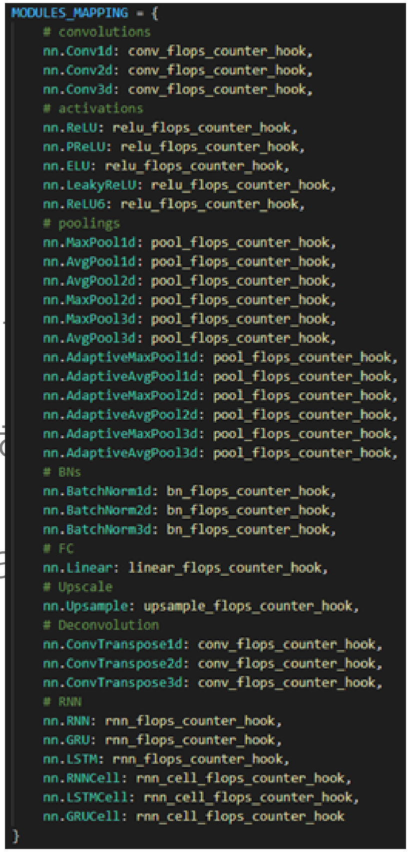
1. 대회 개요

- MACs 계산 방법

ptflops.flops_counter의 get_model_complexi

torch.nnModules 기반 모듈에 hook을 걸어 계산ㅎ

in_channels, kernel_dims, output_dims의 sha



```
MODULES_MAPPING = {
    # convolutions
    nn.Conv1d: conv_flops_counter_hook,
    nn.Conv2d: conv_flops_counter_hook,
    nn.Conv3d: conv_flops_counter_hook,
    # activations
    nn.ReLU: relu_flops_counter_hook,
    nn.PReLU: relu_flops_counter_hook,
    nn.ELU: relu_flops_counter_hook,
    nn.LeakyReLU: relu_flops_counter_hook,
    nn.ReLU6: relu_flops_counter_hook,
    # poolings
    nn.MaxPool1d: pool_flops_counter_hook,
    nn.AvgPool1d: pool_flops_counter_hook,
    nn.AvgPool2d: pool_flops_counter_hook,
    nn.MaxPool2d: pool_flops_counter_hook,
    nn.MaxPool3d: pool_flops_counter_hook,
    nn.AvgPool3d: pool_flops_counter_hook,
    nn.AdaptiveMaxPool1d: pool_flops_counter_hook,
    nn.AdaptiveAvgPool1d: pool_flops_counter_hook,
    nn.AdaptiveMaxPool2d: pool_flops_counter_hook,
    nn.AdaptiveAvgPool2d: pool_flops_counter_hook,
    nn.AdaptiveMaxPool3d: pool_flops_counter_hook,
    nn.AdaptiveAvgPool3d: pool_flops_counter_hook,
    # BNs
    nn.BatchNorm1d: bn_flops_counter_hook,
    nn.BatchNorm2d: bn_flops_counter_hook,
    nn.BatchNorm3d: bn_flops_counter_hook,
    # FC
    nn.Linear: linear_flops_counter_hook,
    # Upscale
    nn.Upsample: upsample_flops_counter_hook,
    # Deconvolution
    nn.ConvTranspose1d: conv_flops_counter_hook,
    nn.ConvTranspose2d: conv_flops_counter_hook,
    nn.ConvTranspose3d: conv_flops_counter_hook,
    # RNN
    nn.RNN: rnn_flops_counter_hook,
    nn.GRU: rnn_flops_counter_hook,
    nn.LSTM: rnn_flops_counter_hook,
    nn.RNNCell: rnn_cell_flops_counter_hook,
    nn.LSTMCell: rnn_cell_flops_counter_hook,
    nn.GRUCell: rnn_cell_flops_counter_hook
}
```

1. 대회 개요

- MACs 계산 방법

ptflops.flops_counter의 ge

torch.nnModules 기반 모듈(

in_channels, kernel_dims,

```python
def conv_flops_counter_hook(conv_module, input, output):
    input = input[0]
    batch_size = input.shape[0]
    output_dims = list(output.shape[2:])
    kernel_dims = list(conv_module.kernel_size)
    in_channels = conv_module.in_channels
    out_channels = conv_module.out_channels
    groups = conv_module.groups

    filters_per_channel = out_channels // groups
    conv_per_position_flops = int(np.prod(kernel_dims)) * \
        in_channels * filters_per_channel

    active_elements_count = batch_size * int(np.prod(output_dims))

    overall_conv_flops = conv_per_position_flops * active_elements_count
    bias_flops = 0

    if conv_module.bias is not None:

        bias_flops = out_channels * active_elements_count

    overall_flops = overall_conv_flops + bias_flops

    conv_module.__flops__ += int(overall_flops)
```

2. 가능한 선택지

2. 가능한 선택지

- F1 중심

Loss, arcface, learning schedule, knowledge distillation
pretrained model, auxiliary training, ABN, channel attention
MuxConv

- MACs 중심

NAS, tensor decomposition, pruning, autoencoder

2. 가능한 선택지

- 최종 선택

pretrained model

knowledge distillation

structured pruning

tensor decomposition

2. 가능한 선택지

- Unstructured pruning
  은닉층 텐서의 일부를 마스킹하여 연산량을 낮추는 방식
  실제 parameter(weights)의 shape이 변화하진 않음

- Structured pruning
  layer 혹은 channel을 제거하여 shape를 직접 낮추는 방식

- Tensor decompostion
  하나의 텐서를 여러 개로 쪼개어 연산량 이득을 취하는 방식
  이 또한 shape를 직접적으로 낮춤

3. 상황 분석

3. 상황 분석

- pretrained model

MobileNet
ShuffleNet
DiceNet
FBNet
MnasNet
MixNet
MuxNet
.
.

3. 상황 분석

- ShuffleNet

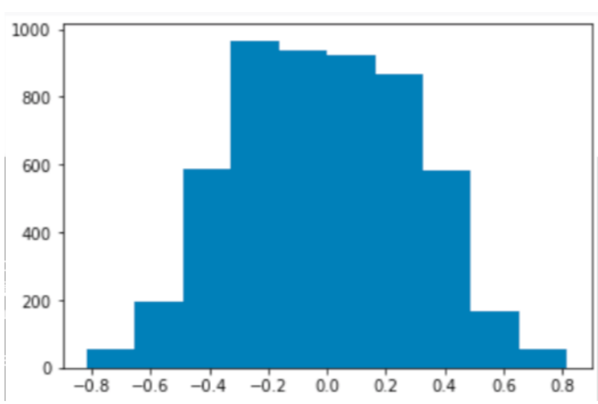
ShuffleNet	Output size	KSize	Stride	Repeat	Output channels
	80x80				groups=3
Conv1 MaxPool	40x40 20x20	3x3 3x3	2 2	1	6
Stage1	10x10 10x10		2 1	1 3	60 60
Stage2	5x5 5x5		2 1	1 7	120 120
Stage3	3x3 3x3		2 1	1 3	240 240
GlobalAvgPool	1x1				
FC (Conv)					9
MACs					1688940.0

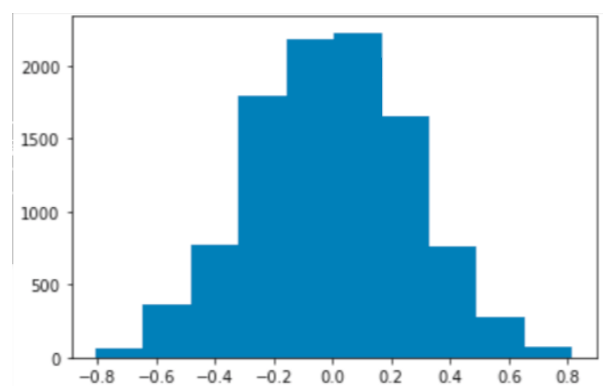
3. 상황 분석

- ShuffleNet

입력 크기를 80으로 축소했기에 추출할 feature의 정보량도 줄어들었다 판단

레이어의 weight 분포





3. 상황 분석

- Pruning & KD

4. 수행 결과

4. 수행 결과

- Pruned ShuffleNet

4-8-4 형태의 unit	→	2-5-2 형태의 unit
F1 0.62, MACs 1688940		F1 0.61, MACs 1083210

4. 수행 결과

- With Decomposition

기존 structured pruning을 더 진행하면 F1이
급격하게 감소

F1은 유지하되 MACs를 더 줄여보고자 channel
pruning과 decomposition 활용

기존 weight의 일부를 줄어든 shape에 맞게
잘라서 사용

4. 수행 결과

- Tensor Decomposition

기존 stage3	channel 축소	group 분할
[1065390]	[1065390]	[1012470]

```
(unit1): ShuffleUnit(
  (compress_conv1): Conv2d(120, 60, kernel_size=(1, 1), stride=(1, 1), groups=3, bias=False)
  (compress_bn1): BatchNorm2d(60, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (c_shuffle): ChannelShuffle(groups=3)
  (dw_conv2): Conv2d(60, 60, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), groups=60, bias=False)
  (dw_bn2): BatchNorm2d(60, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (expand_conv3): Conv2d(60, 120, kernel_size=(1, 1), stride=(1, 1), groups=3, bias=False)
  (expand_bn3): BatchNorm2d(120, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (avgpool): AvgPool2d(kernel_size=3, stride=2, padding=1)
  (activ): ReLU(inplace=True)
)
(unit2): ShuffleUnit(
  (compress_conv1): Conv2d(240, 60, kernel_size=(1, 1), stride=(1, 1), groups=3, bias=False)
  (compress_bn1): BatchNorm2d(60, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (c_shuffle): ChannelShuffle(groups=3)
  (dw_conv2): Conv2d(60, 60, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=60, bias=False)
  (dw_bn2): BatchNorm2d(60, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (expand_conv3): Conv2d(60, 240, kernel_size=(1, 1), stride=(1, 1), groups=3, bias=False)
  (expand_bn3): BatchNorm2d(240, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (activ): ReLU(inplace=True)
)
```

```
(unit1): ShuffleUnit(
  (compress_conv1): Conv2d(120, 60, kernel_size=(1, 1), stride=(1, 1), groups=3, bias=False)
  (compress_bn1): BatchNorm2d(60, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (c_shuffle): ChannelShuffle(groups=3)
  (dw_conv2): Conv2d(60, 60, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), groups=60, bias=False)
  (dw_bn2): BatchNorm2d(60, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (expand_conv3): Conv2d(60, 90, kernel_size=(1, 1), stride=(1, 1), groups=3, bias=False)
  (expand_bn3): BatchNorm2d(90, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (avgpool): AvgPool2d(kernel_size=3, stride=2, padding=1)
  (activ): ReLU(inplace=True)
)
(unit2): ShuffleUnit(
  (compress_conv1): Conv2d(210, 60, kernel_size=(1, 1), stride=(1, 1), groups=3, bias=False)
  (compress_bn1): BatchNorm2d(60, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (c_shuffle): ChannelShuffle(groups=3)
  (dw_conv2): Conv2d(60, 60, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=60, bias=False)
  (dw_bn2): BatchNorm2d(60, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (expand_conv3): Conv2d(60, 210, kernel_size=(1, 1), stride=(1, 1), groups=3, bias=False)
  (expand_bn3): BatchNorm2d(210, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (activ): ReLU(inplace=True)
)
```

```
(unit1): ShuffleUnit(
  (compress_conv1): Conv2d(120, 60, kernel_size=(1, 1), stride=(1, 1), groups=3, bias=False)
  (compress_bn1): BatchNorm2d(60, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (c_shuffle): ChannelShuffle(groups=3)
  (dw_conv2): Conv2d(60, 60, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), groups=60, bias=False)
  (dw_bn2): BatchNorm2d(60, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (expand_conv3): Conv2d(60, 90, kernel_size=(1, 1), stride=(1, 1), groups=3, bias=False)
  (expand_bn3): BatchNorm2d(90, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (avgpool): AvgPool2d(kernel_size=3, stride=2, padding=1)
  (activ): ReLU(inplace=True)
)
(unit2): ShuffleUnit(
  (compress_conv1): Conv2d(210, 60, kernel_size=(1, 1), stride=(1, 1), groups=6, bias=False)
  (compress_bn1): BatchNorm2d(60, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (c_shuffle): ChannelShuffle(groups=3)
  (dw_conv2): Conv2d(60, 60, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=60, bias=False)
  (dw_bn2): BatchNorm2d(60, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (expand_conv3): Conv2d(60, 210, kernel_size=(1, 1), stride=(1, 1), groups=6, bias=False)
  (expand_bn3): BatchNorm2d(210, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (activ): ReLU(inplace=True)
)
```

5. 회고

5. 회고

- 마지막 단계 기준으로 channels pruning과 decomposition이 모두 layer pruning에 비해 성능 하락 폭이 훨씬 작았기 때문에 사이즈가 더 큰 모델이었다면 성능 향상이 있었을 것

- pruning 과정 중 반복 작업이 많았기 때문에 optuna, wandb등의 autoML을 이용하면 자동화하여 진행할 수 있을 것
  (실제로 자동으로 pruning을 진행해주는 툴 다수 존재)

Thank You