

On-Device AI 실습: Pruning for CNN

Dongkun Shin
Intelligent Embedded Systems Lab.
Sungkyunkwan University

실습 Setup

- 실습 자료 다운로드
 - https://drive.google.com/drive/folders/1GZRDpshMbKn6f46Wdoj204tXMi4dOrcQ?usp=drive_link
 - 짧은 주소: <https://bit.ly/4fQXk3f>
- 실습 자료가 있는 directory에 대해 명령 프롬프트 또는 PowerShell 수행 후 conda 환경 설정 및 jupyter lab 실행

```
(base) PS D:\실습> conda activate env_aias_test  
(env_aias_test) PS D:\실습> jupyter lab
```

- 실습 중 Python package 등 실행 환경에 이슈가 있을 경우 다음 bat 파일을 다운로드 받아 rollback 수행
 - [WWSwschoolavdazfiles002.file.core.windows.net\aias-language\Env\Rollback_env_aia test.bat](#)

Overview

1. Pruning Granularity

- Fine-grained, Vector-level, Kernel-level, Channel-level의 차이를 학습하고 구현합니다.

2. Pruning Ratio

- 제거할 가중치의 비율(Sparsity)을 결정하는 방법을 배웁니다. Layer-wise와 Global 방식의 차이를 실습을 통해 확인합니다.

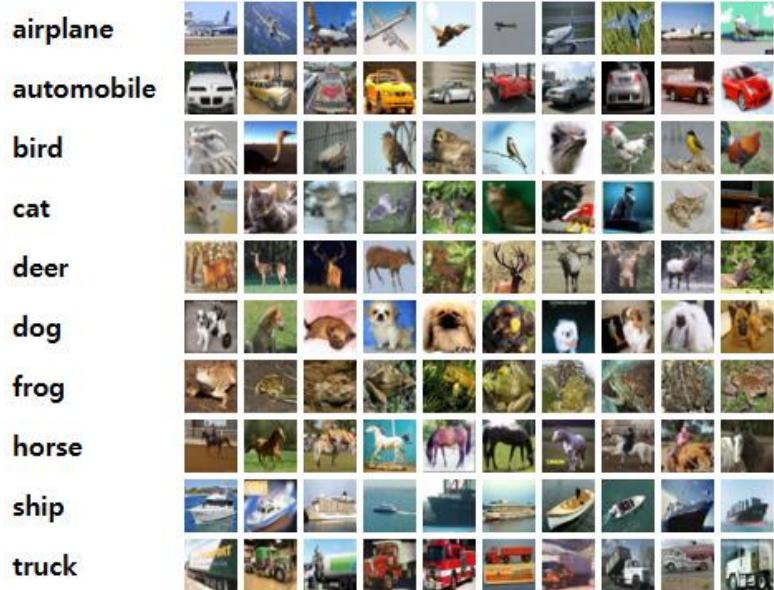
3. Pruning Schedule

- One-shot Pruning 및 Iterative Pruning을 이해하고, Linear 및 Cubic Scheduling 방법을 적용하여 성능 변화를 분석합니다.

PyTorch 기본 Tutorial: Dataset

- Model을 학습시키기 위해서는 Dataset이 필요하며 본 실습에서는 CIFAR-10 dataset을 사용하여 진행
- PyTorch에서 Dataset은 DataLoader라는 Python 클래스를 통해 관리
 - Pre-processing (transform)을 포함한 dataset, batch size, shuffle 여부, worker 개수 등을 설정
- CIFAR-10 Dataset은 32x32 컬러 이미지, 10개의 class로 구성
 - Train: 50,000
 - Test: 10,000

CIFAR-10 Dataset



<https://www.cs.toronto.edu/~kriz/cifar.html>

PyTorch 기본 Tutorial: Dataset

Data augmentation

```
# 학습 및 테스트 데이터에 적용할 변환 함수
transforms_train = Compose([
    RandomCrop(image_size, padding=4),
    RandomHorizontalFlip(),
    ToTensor(),
])

transforms_test = Compose([
    ToTensor(),
])
```

```
# CIFAR-10 데이터셋 로드
dataset_train = CIFAR10(
    root="data/cifar10",
    train=True,
    download=True,
    transform=transforms_train)
```

```
dataset_test = CIFAR10(
    root="data/cifar10",
    train=False,
    download=True,
    transform=transforms_test)
```

```
# Dataloader 생성
dataloader_train = DataLoader(dataset_train, batch_size=512, shuffle=True, num_workers=0, pin_memory=True)
dataloader_test = DataLoader(dataset_test, batch_size=512, shuffle=False, num_workers=0, pin_memory=True)
dataloader = {"train": dataloader_train, "test": dataloader_test}
```

PyTorch 기본 Tutorial: Model

본 실습에서는 torch.hub를 통해 미리 학습된 모델을 불러옵니다.

```
TORCH_HUB_REPO = "SKKU-ESLAB/pytorch-models"
MODEL_NAME = "cifar10_vgg9_bn"

model = torch.hub.load(TORCH_HUB_REPO, MODEL_NAME, pretrained=True)
if torch.cuda.is_available():
    model = model.cuda()

print(model)
```

PyTorch 기본 Tutorial: Model

- PyTorch에서 모든 모델과 레이어는 nn.Module 클래스를 통해 정의되며, 계층적인 구조로 구성
- 실습에서는 CIFAR-10으로 미리 학습된 VGG9 모델을 사용
 - 8개의 convolution layer와 1개의 linear layer로 구성

실습에서 사용할 VGG9 모델

```
VGG(
    backbone: Sequential(
        (conv0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu0): ReLU(inplace=True)
        (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (pool0): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
        (conv2): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu3): ReLU(inplace=True)
        (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
        (conv4): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu4): ReLU(inplace=True)
        (conv5): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn5): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu5): ReLU(inplace=True)
        (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
        (conv6): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn6): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu6): ReLU(inplace=True)
        (conv7): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn7): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu7): ReLU(inplace=True)
        (pool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (classifier): Linear(in_features=512, out_features=10, bias=True)
)
```

PyTorch 기본 Tutorial: 모델 크기 평가

Model 내의 모든 parameter에 대해 순회

```
def get_num_parameters(model: torch.nn.Module, count_nonzero_only=False) -> int:  
    """  
    모델의 총 파라미터 수 계산  
    """  
  
    num_counted_elements = 0  
    for param in model.parameters():  
        num_counted_elements += param.count_nonzero() if count_nonzero_only else param.numel()  
    return num_counted_elements
```

```
def get_model_size(model: torch.nn.Module, data_width=32, count_nonzero_only=False) -> int:  
    """  
    모델의 크기를 비트 단위로 계산  
    """  
  
    return get_num_parameters(model, count_nonzero_only) * data_width  
  
dense_model_size = get_model_size(model)  
print(f"dense model has size={dense_model_size/ (1024 ** 2) / 8:.2f} MiB")
```

PyTorch 기본 Tutorial: 모델 정확도 평가

9

28

```
@torch.no_grad()
def evaluate(
    model: torch.nn.Module,
    dataloader: DataLoader,
    verbose=True,
) -> float:
    model.eval()
    num_samples, num_correct = 0, 0
    for inputs, targets in tqdm(dataloader, desc="Eval", leave=False, disable=not verbose):
        if torch.cuda.is_available():
            inputs, targets = inputs.cuda(), targets.cuda()
        outputs = model(inputs)
        preds = outputs.argmax(dim=1)
        num_samples += targets.size(0)
        num_correct += (preds == targets).sum()
    return (num_correct / num_samples * 100).item()
```

Model을 evaluation 모드로 변경

Dataloader를 통해 data 획득

Input에 대한 output 획득
(이 때의 output은 10개의 class에 대한 confidence 값이 되며, 이 중 가장 큰 값의 index가 모델이 예측한 class의 index가 됨)

outputs의 shape는 (batch_size, num_classes)가 됨.

PyTorch 기본 Tutorial: Parameters

Layer	Parameters	Shape
torch.nn.Conv2d	weight	$\left(\text{out_channels}, \frac{\text{in_channels}}{\text{groups}}, \text{kernel_height}, \text{kernel_width} \right)$
	bias	(out_channels)
torch.nn.Linear	weight	$(\text{out_features}, \text{in_features})$
	bias	(out_features)
torch.nn.BatchNorm2d	weight	(num_features)
	bias	(num_features)

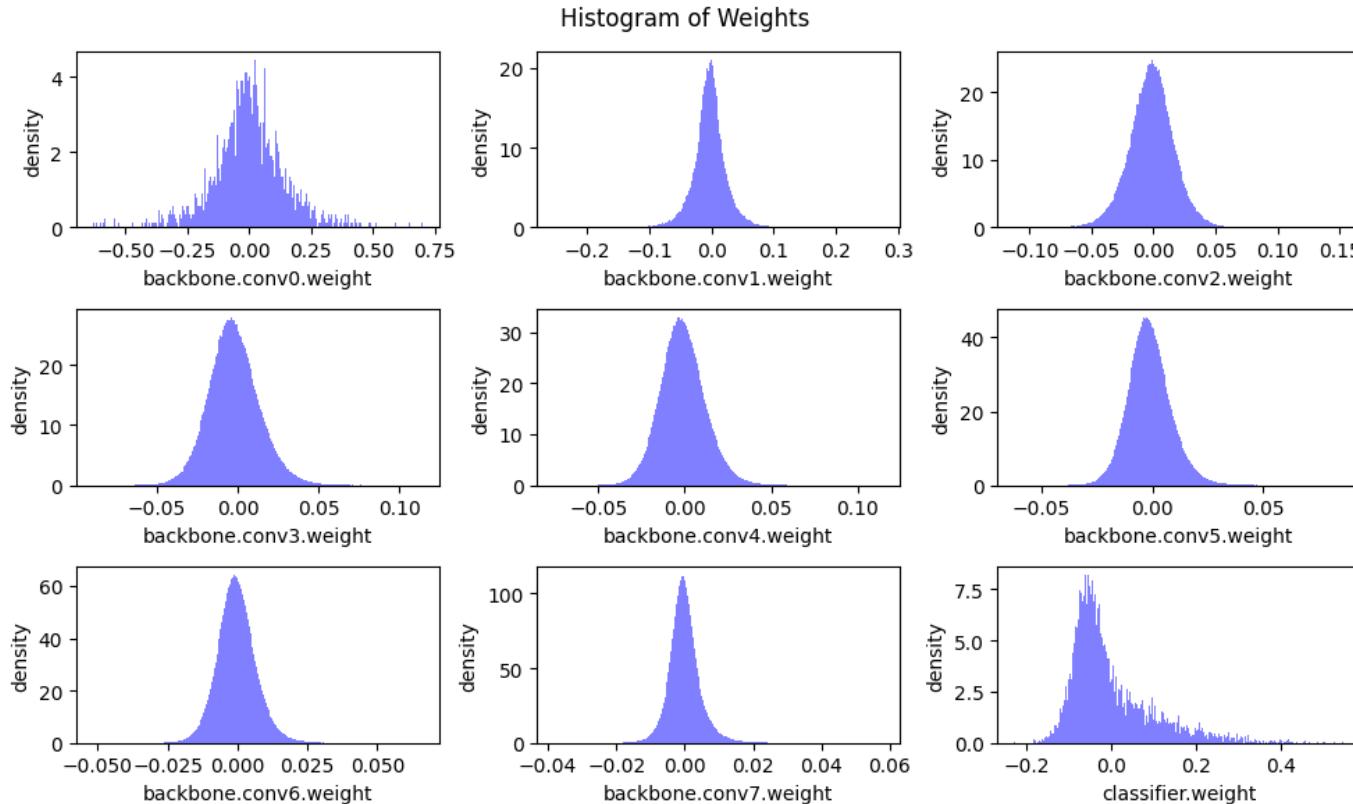
E.g., Conv2d와 Linear의 weight를 순회:

```
for param in model.parameters():
    if param.dim() > 1
        ...
    ...
```

or

```
for m in model.modules():
    if isinstance(m, torch.nn.Conv2d) or isinstance(m, torch.nn.Linear):
        param = m.weight
    ...
    ...
```

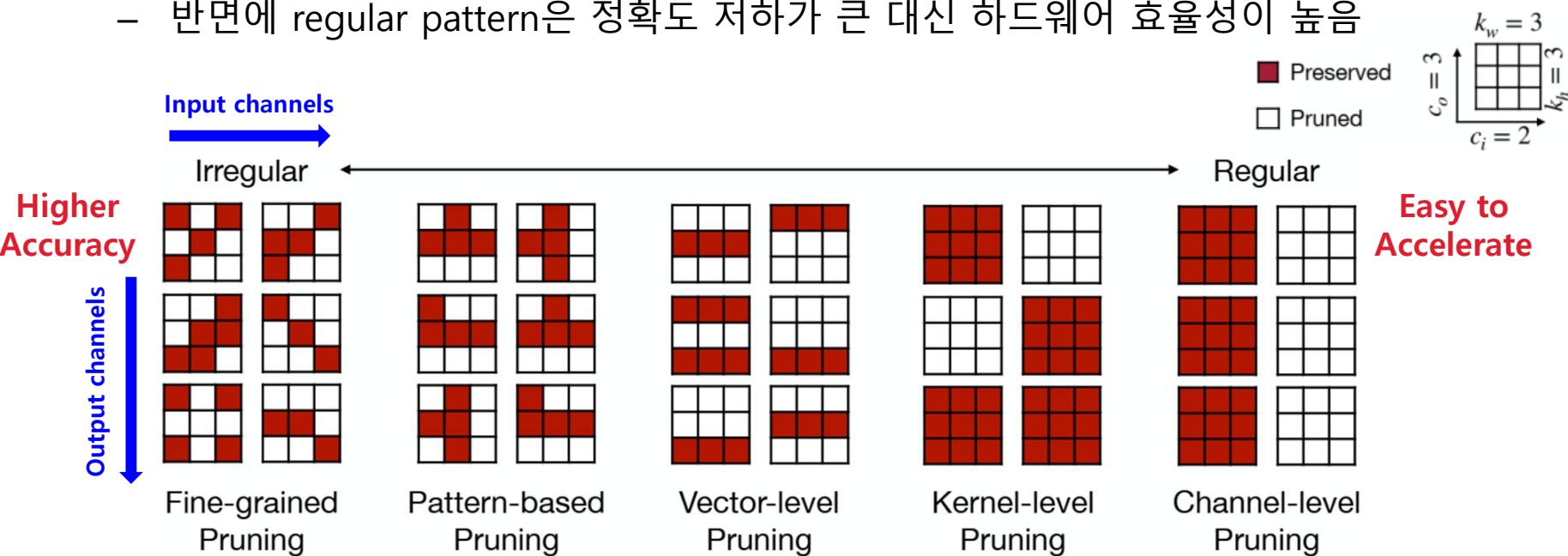
Pre-trained Model의 레이어별 weight 분포 확인



많은 weight들이 0에
가깝게 분포되어 있음.
→ Pruning을 수행하여
weight 압축

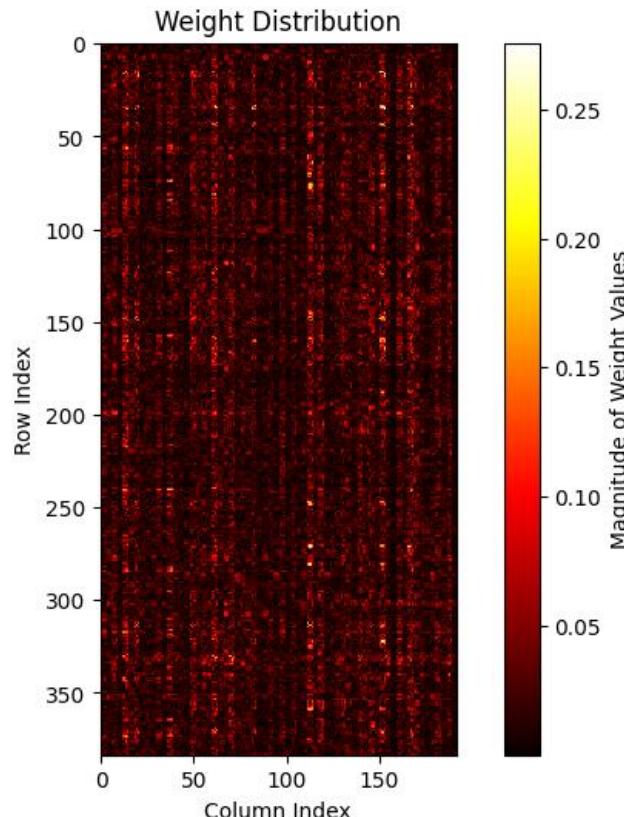
1.1. Pruning Granularity/Pattern

- 다양한 pruning granularity를 고려한 pruning 코드를 작성해 봅니다.
 - Pruning pattern은 아래와 같이 다양하게 존재
 - Irregular pattern은 정확도 저하는 작지만 하드웨어 효율도 낮아짐
 - 반면에 regular pattern은 정확도 저하가 큰 대신 하드웨어 효율성이 높음



Weight 분포 출력

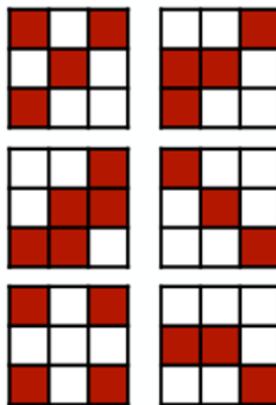
- 오른쪽 그림은 (128, 64, 3, 3) weight의 magnitude를 출력한 결과입니다.
 - PyTorch에서 convolution layer의 weight의 layout은 (CO, CI, KH, KW) 입니다.
 - CO: output channels
 - CI: input channels
 - KH: kernel height
 - KW: kernel width
 - 출력을 위해 weight의 레이아웃을 ($CO * KH$, $CI * KW$) 형태로 변형하였습니다.
 - 이를 통해 앞 슬라이드의 그림과 같이 데이터를 배치시킬 수 있습니다.



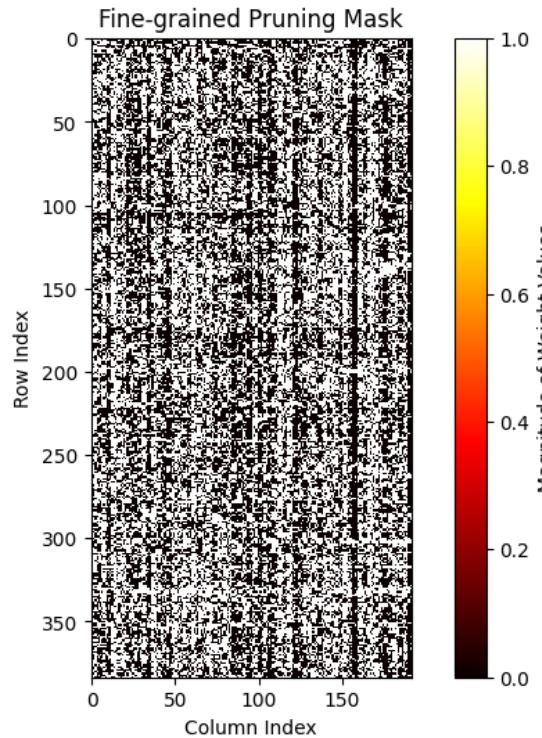
[실습 1] Fine-grained Pruning 구현

14

28



Fine-grained
Pruning



[실습 1] Answer

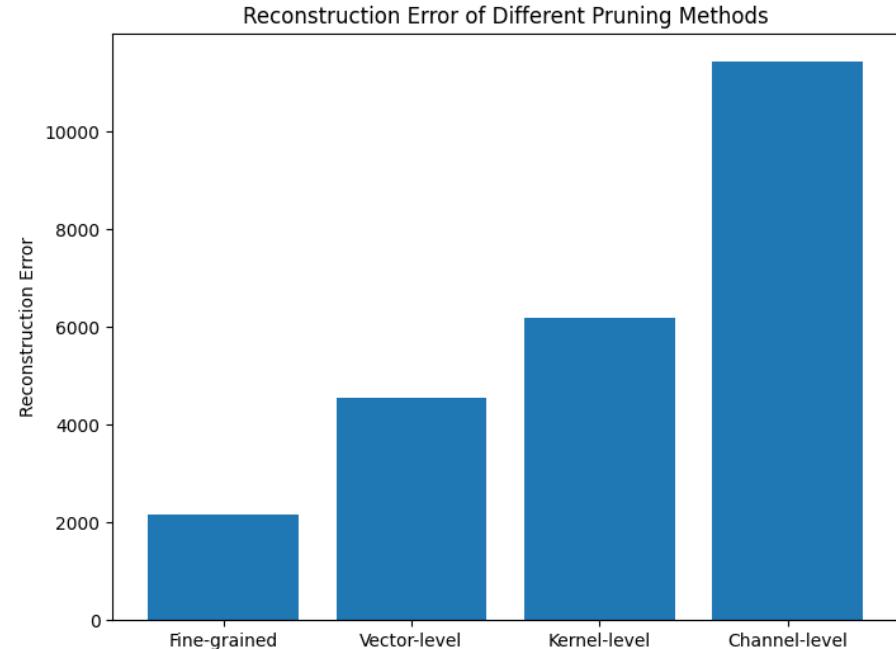
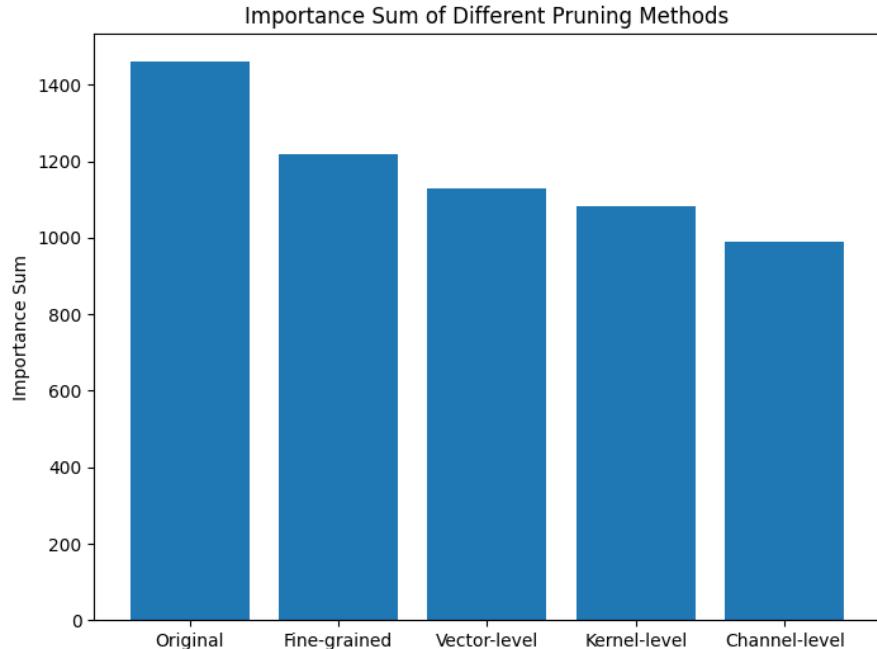
```
##### YOUR CODE STARTS HERE #####
# num_pruned_elements를 계산하세요.
# hint: round() 함수를 사용하세요.
num_pruned_elements = round(weight.numel() * sparsity)

# 절댓값을 사용하여 중요도 계산
# hint: torch.abs() 함수를 사용하세요.
importance = torch.abs(weight)

# pruning threshold를 계산하세요.
# hint: torch.kthvalue() 함수를 사용하세요.
threshold = torch.kthvalue(importance.flatten(), num_pruned_elements)[0]

# threshold보다 큰 값들을 1로 설정
# hint: 부등호를 사용하세요.
mask = importance > threshold
##### YOUR CODE ENDS HERE #####
```

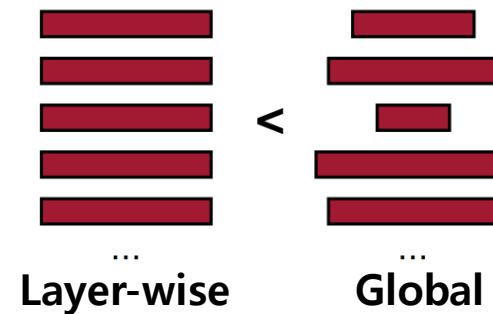
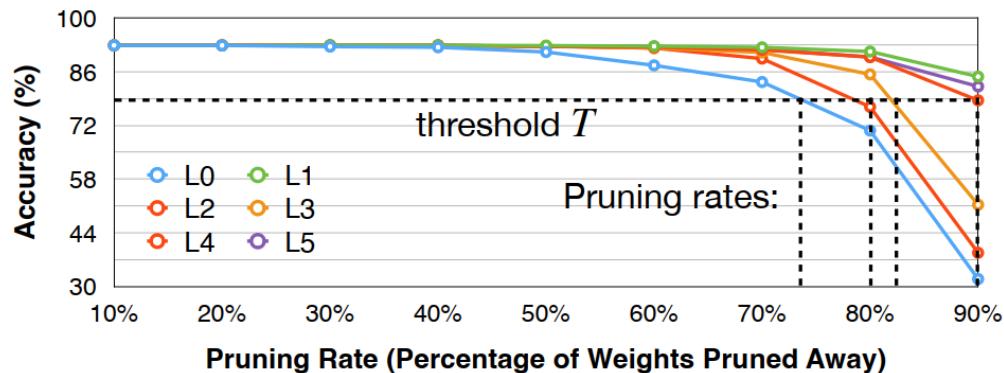
Pruning 방법 간 Importance Sum/Reconstruction Error 비교



1.2. Pruning Ratio

- Sensitivity Analysis

레이어마다 sensitivity가 다르기 때문에 각 레이어마다 다른 pruning ratio가 필요하다.

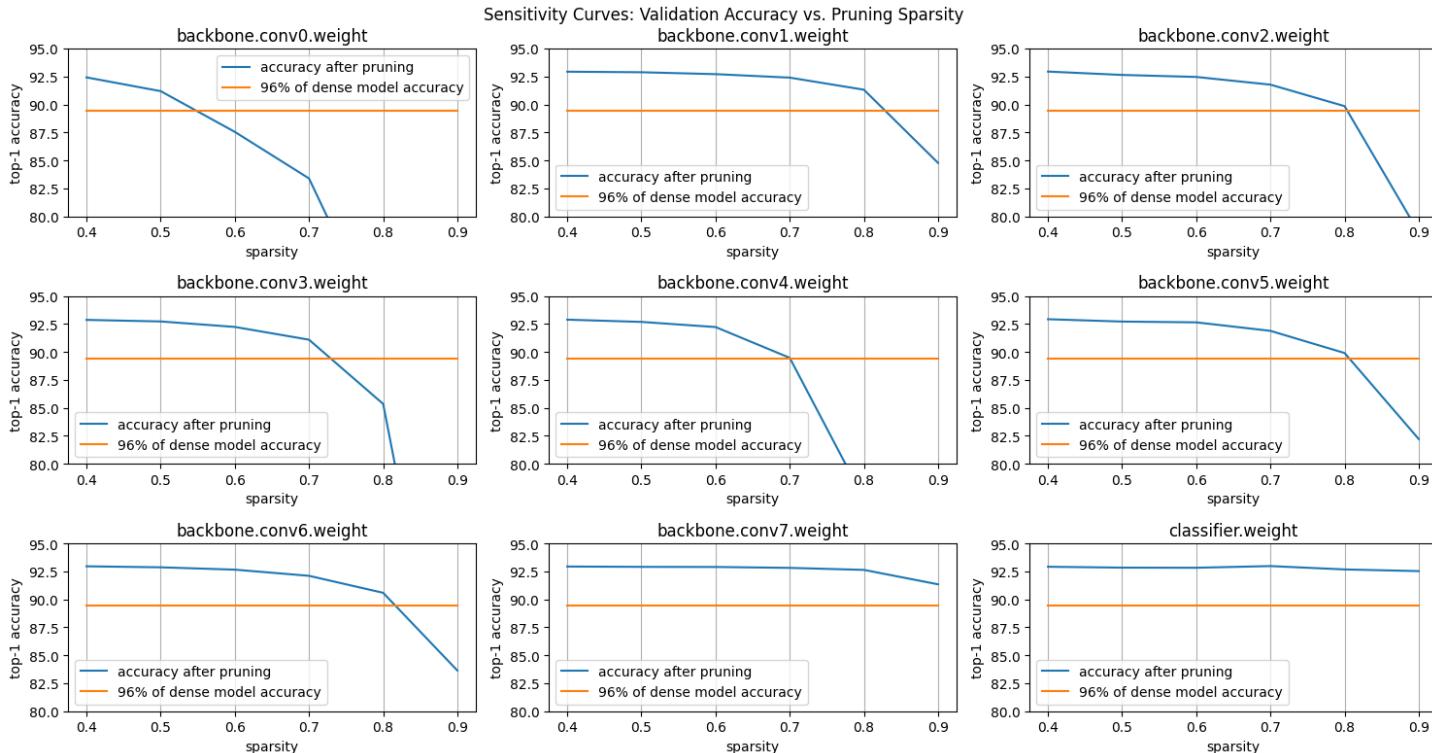


- Global Magnitude Pruning

간단한 global pruning 방법으로 global threshold를 통해 pruning을 수행한다.

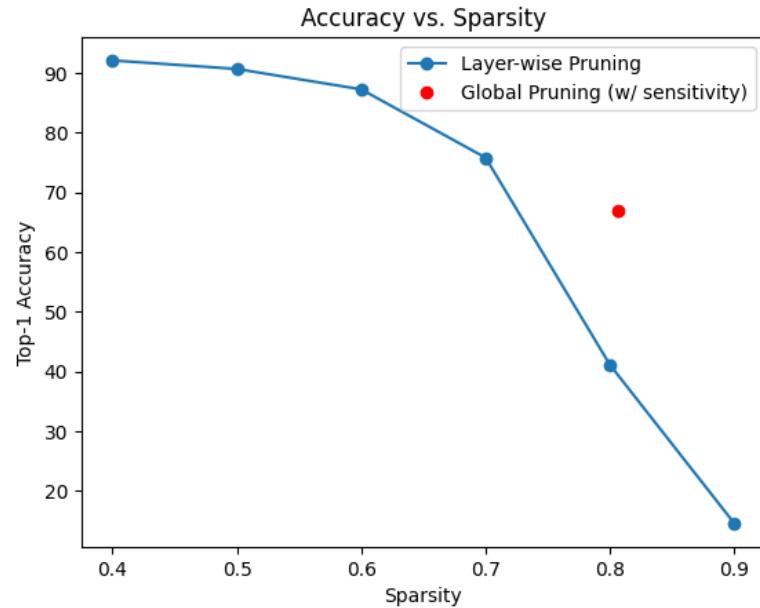
Sensitivity analysis

- Pre-trained model의 sensitivity를 분석합니다.



[실습 2] Sensitivity analysis를 통한 pruning 수행

- 앞서 수행한 sensitivity analysis를 바탕으로 각 레이어마다 적절한 sparsity를 할당하고 evaluation을 진행합니다.
- Evaluation 후 uniform sparsity를 가지는 layer-wise pruning과 accuracy를 비교합니다.

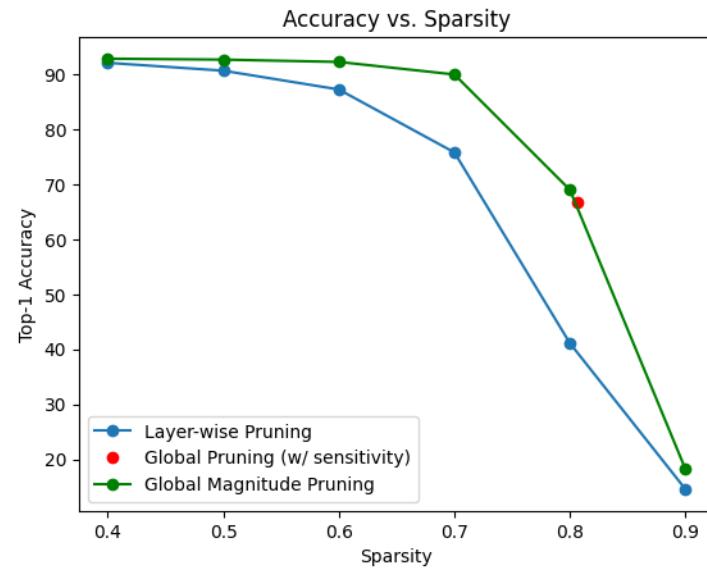


[실습 2] Answer

```
custom_sparsity_dict = {
    ##### YOUR CODE STARTS HERE #####
    # please modify the sparsity value of each layer
    # please DO NOT modify the key of sparsity_dict
    'backbone.conv0.weight': 0.5,
    'backbone.conv1.weight': 0.8,
    'backbone.conv2.weight': 0.8,
    'backbone.conv3.weight': 0.7,
    'backbone.conv4.weight': 0.7,
    'backbone.conv5.weight': 0.8,
    'backbone.conv6.weight': 0.8,
    'backbone.conv7.weight': 0.9,
    'classifier.weight': 0.9
    ##### YOUR CODE ENDS HERE #####
}
```

[실습 3] Global magnitude pruning 구현

- 하나의 global threshold를 통해 pruning을 수행하는 global magnitude pruning을 구현합니다.
- Global magnitude pruning을 수행하여 layer-wise pruning 및 sensitivity analysis를 통해 얻은 global pruning accuracy와 비교합니다.



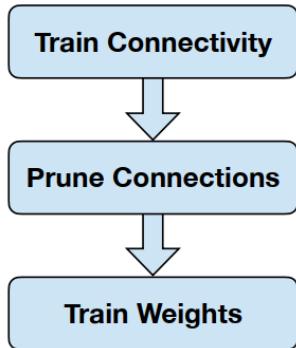
[실습 3] Answer

```
##### YOUR CODE STARTS HERE #####
# 모든 weight를 하나로 합쳐주세요.
# hint: torch.cat()을 사용하세요.
all_weights = torch.cat(parameters_to_prune)

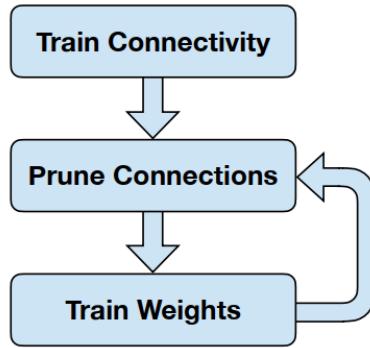
# all_weights를 기준으로 global threshold를 구해주세요.
num_elements = all_weights.numel()
num_zeros = round(num_elements * sparsity)
importance = torch.abs(all_weights)
threshold = torch.kthvalue(importance, num_zeros)[0]
##### YOUR CODE ENDS HERE #####
```

1.3. Pruning Schedule: Overview

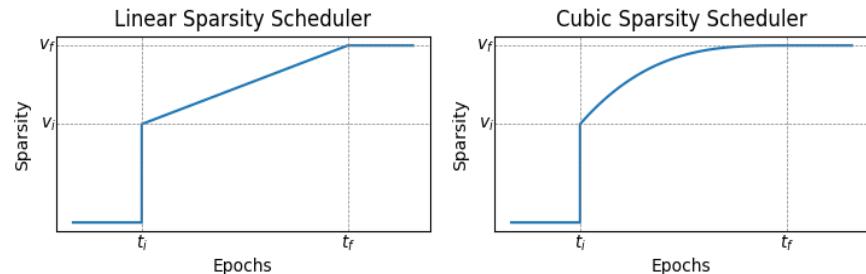
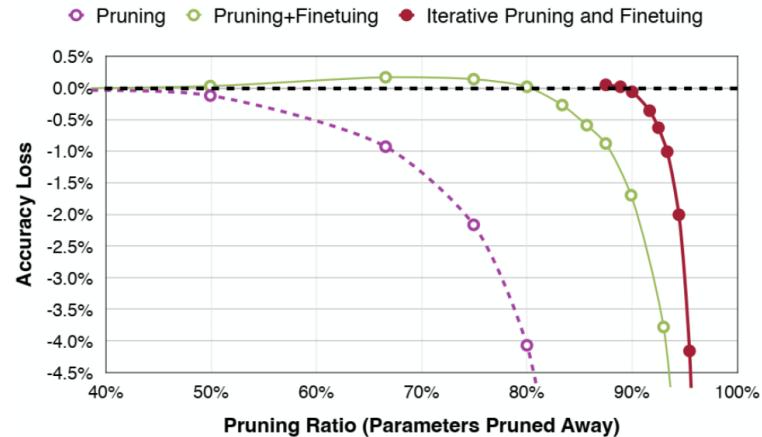
- One-shot Pruning
- Iterative Pruning
 - Sparsity scheduling



One-shot



Iterative



모델 학습 함수

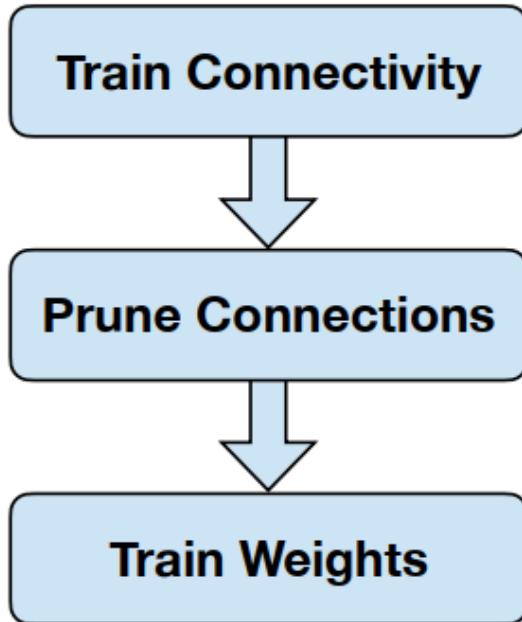
24

28

```
def train(
    model: torch.nn.Module,
    dataloader: DataLoader,
    criterion: torch.nn.Module,
    optimizer: torch.optim.Optimizer,
    scheduler: torch.optim.lr_scheduler.LRScheduler,
    callbacks=None,
) -> None:
    model.train()
    for inputs, targets in tqdm(dataloader, desc='Train', leave=False):
        if torch.cuda.is_available():
            inputs, targets = inputs.cuda(), targets.cuda()
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()
        scheduler.step()
        if callbacks is not None:
            for callback in callbacks:
                callback()
```

- **Criterion:**
 - Loss function
 - **Optimizer:**
 - 모델의 parameter를 업데이트해 학습이 진행되도록 만드는 알고리즘 (e.g., SGD)
 - **Scheduler:**
 - Learning rate scheduler
- optimizer에 누적된 gradient 0으로 초기화
- Loss 계산 및 backward propagation을 통해 gradient 계산
- Parameter & Learning rate update

One-shot pruning



Epoch 1/5: accuracy=89.84%
Epoch 2/5: accuracy=90.51%
Epoch 3/5: accuracy=90.88%
Epoch 4/5: accuracy=90.89%
Epoch 5/5: accuracy=90.94%

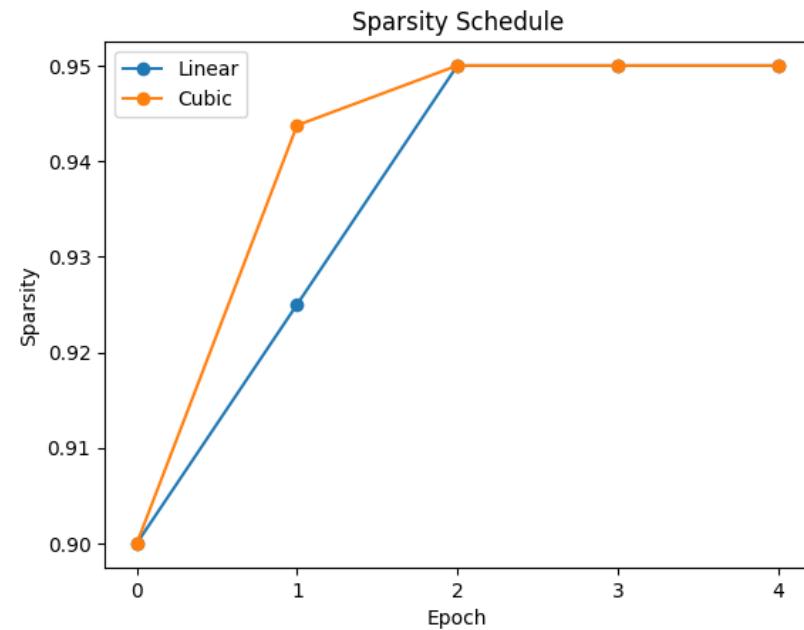
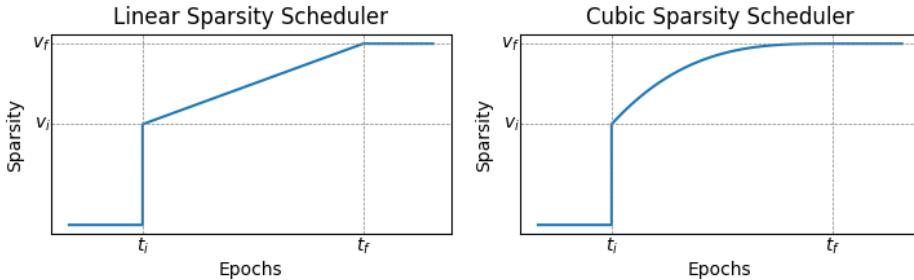
학습 accuracy는 training dataset을 읽는
순서 및 pre-processing (augmentation)
방법에 따라 달라질 수 있습니다.

[실습 4] Sparsity scheduler 구현

- Sparsity scheduler

$$- v^{(t)} = v_f + (v_i - v_f) \left(1 - \frac{t-t_i}{t_f-t_i}\right)^E$$

- E=1: linear
- E=3: cubic



[실습 4] Answer

27

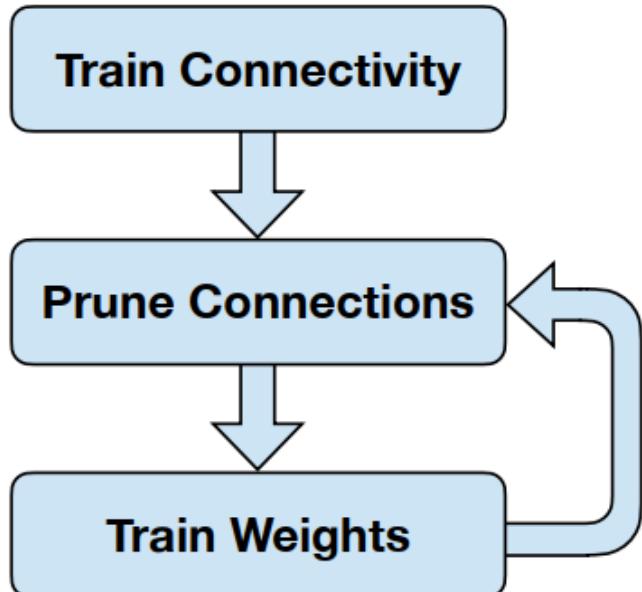
28

```
##### YOUR CODE STARTS HERE #####
for epoch in range(num_epochs):
    # 위 수식에 맞게 sparsity를 계산하세요. (단, epoch_start보다 작은 경우 0으로 epochs_end 이후는 sparsity_end로 설정합니다.)
    if epoch < epoch_start:
        sparsity = 0
    elif epoch >= epoch_end:
        sparsity = sparsity_end
    else:
        sparsity = sparsity_end + (sparsity_start - sparsity_end) * (1 - (epoch - epoch_start) / (epoch_end - epoch_start)) ** exponent

    sparsity_schedule.append(sparsity)
##### YOUR CODE ENDS HERE #####

```

Iterative pruning



Linear Sparsity Scheduler

Epoch 1/5: accuracy=91.68%
Epoch 2/5: accuracy=91.68%
Epoch 3/5: accuracy=90.86%
Epoch 4/5: accuracy=91.03%
Epoch 5/5: accuracy=90.97%

Cubic Sparsity Scheduler

Epoch 1/5: accuracy=91.66%
Epoch 2/5: accuracy=91.00%
Epoch 3/5: accuracy=91.19%
Epoch 4/5: accuracy=91.41%
Epoch 5/5: accuracy=91.36%

학습 accuracy는 training dataset을 읽는
순서 및 pre-processing (augmentation)
방법에 따라 달라질 수 있습니다.