Lab4
Model Compression: Pruning & Quantization

313510164 陳緯亭 電子碩一

November 11, 2024

Contents

1 Screenshot

1.1 The screenshot of training result from task 1

Best Accuracy: 88.75%

```
Epoch:  41 |train loss: 0.1995 |train accuracy: 92.99 |validation loss: 0.5558 |validation accuracy: 86.88 |learning rate: 1.0e-02 |train time: 3.07 |test time: 0.45
Training: 100%|█████████████| 65/65 [00:02<00:00, 22.82it/s]
Testing: 100%|█████████████| 10/10 [00:00<00:00, 23.22it/s]
05 Nov 2024 13:08:34
Epoch:  42 |train loss: 0.2899 |train accuracy: 90.85 |validation loss: 0.6469 |validation accuracy: 82.50 |learning rate: 1.0e-02 |train time: 2.85 |test time: 0.44
Training: 100%|█████████████| 65/65 [00:02<00:00, 22.91it/s]
Testing: 100%|█████████████| 10/10 [00:00<00:00, 22.25it/s]
05 Nov 2024 13:08:37
Epoch:  43 |train loss: 0.2383 |train accuracy: 92.31 |validation loss: 0.5358 |validation accuracy: 86.25 |learning rate: 1.0e-02 |train time: 2.85 |test time: 0.46
Training: 100%|█████████████| 65/65 [00:03<00:00, 18.74it/s]
Testing: 100%|█████████████| 10/10 [00:00<00:00, 16.71it/s]
05 Nov 2024 13:08:42
Epoch:  44 |train loss: 0.1654 |train accuracy: 94.84 |validation loss: 0.5093 |validation accuracy: 85.62 |learning rate: 1.0e-02 |train time: 3.48 |test time: 0.61
Training: 100%|█████████████| 65/65 [00:02<00:00, 22.93it/s]
Testing: 100%|█████████████| 10/10 [00:00<00:00, 23.43it/s]
05 Nov 2024 13:08:45
Epoch:  45 |train loss: 0.1764 |train accuracy: 94.45 |validation loss: 0.7160 |validation accuracy: 85.00 |learning rate: 1.0e-02 |train time: 2.84 |test time: 0.43
Training: 100%|█████████████| 65/65 [00:02<00:00, 22.95it/s]
Testing: 100%|█████████████| 10/10 [00:00<00:00, 22.85it/s]
05 Nov 2024 13:08:48
Epoch:  46 |train loss: 0.1914 |train accuracy: 94.35 |validation loss: 0.3779 |validation accuracy: 88.75 |learning rate: 1.0e-02 |train time: 2.84 |test time: 0.44
Saving..
Training: 100%|█████████████| 65/65 [00:02<00:00, 23.00it/s]
Testing: 100%|█████████████| 10/10 [00:00<00:00, 22.92it/s]
05 Nov 2024 13:08:51
Epoch:  47 |train loss: 0.1878 |train accuracy: 93.87 |validation loss: 0.5667 |validation accuracy: 83.12 |learning rate: 1.0e-02 |train time: 2.83 |test time: 0.44
Training: 100%|█████████████| 65/65 [00:03<00:00, 18.68it/s]
Testing: 100%|█████████████| 10/10 [00:00<00:00, 20.11it/s]
05 Nov 2024 13:08:55
Epoch:  48 |train loss: 0.1314 |train accuracy: 95.13 |validation loss: 0.6131 |validation accuracy: 85.62 |learning rate: 1.0e-02 |train time: 3.49 |test time: 0.50
Training: 100%|█████████████| 65/65 [00:02<00:00, 22.79it/s]
Testing: 100%|█████████████| 10/10 [00:00<00:00, 22.53it/s]05 Nov 2024 13:08:59
Epoch:  49 |train loss: 0.1353 |train accuracy: 95.91 |validation loss: 0.4671 |validation accuracy: 85.62 |learning rate: 1.0e-02 |train time: 2.86 |test time: 0.45
Best accuracy: 88.75
```

1.2 The screenshot of training result from task 2

Best Accuracy: 89.38%

```
Number of parameter: 0.26M
After fine-tune
Validation loss: 0.4531 Validation accuracy: 89.38
```

1.3 The screenshot of training result from task 3

Best Accuracy: 78.12%

```
09 Nov 2024 12:47:08
Epoch: 186 |train loss: 0.4128 |train accuracy: 90.94 |validation loss: 0.9300 |validation accuracy: 73.12 |learning rate: 6.3e-04 |train time: 2.59 |test time: 0.42
09 Nov 2024 12:47:11
Epoch: 187 |train loss: 0.3754 |train accuracy: 90.36 |validation loss: 1.0608 |validation accuracy: 71.88 |learning rate: 6.3e-04 |train time: 2.54 |test time: 0.43
09 Nov 2024 12:47:14
Epoch: 188 |train loss: 0.2729 |train accuracy: 91.43 |validation loss: 0.9976 |validation accuracy: 71.88 |learning rate: 6.3e-04 |train time: 2.49 |test time: 0.44
09 Nov 2024 12:47:17
Epoch: 189 |train loss: 0.4801 |train accuracy: 91.63 |validation loss: 0.8760 |validation accuracy: 70.00 |learning rate: 6.3e-04 |train time: 2.66 |test time: 0.50
09 Nov 2024 12:47:20
Epoch: 190 |train loss: 0.3482 |train accuracy: 90.75 |validation loss: 0.9277 |validation accuracy: 71.88 |learning rate: 6.3e-04 |train time: 2.74 |test time: 0.43
09 Nov 2024 12:47:23
Epoch: 191 |train loss: 0.3300 |train accuracy: 91.53 |validation loss: 0.9160 |validation accuracy: 71.88 |learning rate: 3.1e-04 |train time: 2.48 |test time: 0.42
09 Nov 2024 12:47:26
Epoch: 192 |train loss: 0.3639 |train accuracy: 92.41 |validation loss: 0.9165 |validation accuracy: 71.88 |learning rate: 3.1e-04 |train time: 2.52 |test time: 0.43
09 Nov 2024 12:47:29
Epoch: 193 |train loss: 0.2956 |train accuracy: 90.94 |validation loss: 0.9528 |validation accuracy: 71.88 |learning rate: 3.1e-04 |train time: 2.51 |test time: 0.47
09 Nov 2024 12:47:33
Epoch: 194 |train loss: 0.3201 |train accuracy: 91.04 |validation loss: 0.9134 |validation accuracy: 72.50 |learning rate: 3.1e-04 |train time: 2.88 |test time: 0.45
09 Nov 2024 12:47:36
Epoch: 195 |train loss: 0.4036 |train accuracy: 90.85 |validation loss: 0.9491 |validation accuracy: 70.62 |learning rate: 3.1e-04 |train time: 2.53 |test time: 0.43
09 Nov 2024 12:47:38
Epoch: 196 |train loss: 0.2930 |train accuracy: 89.68 |validation loss: 0.9496 |validation accuracy: 70.62 |learning rate: 3.1e-04 |train time: 2.53 |test time: 0.42
09 Nov 2024 12:47:41
Epoch: 197 |train loss: 0.3156 |train accuracy: 91.82 |validation loss: 0.8842 |validation accuracy: 72.50 |learning rate: 3.1e-04 |train time: 2.54 |test time: 0.43
09 Nov 2024 12:47:45
Epoch: 198 |train loss: 0.3428 |train accuracy: 89.87 |validation loss: 0.9338 |validation accuracy: 73.75 |learning rate: 3.1e-04 |train time: 2.77 |test time: 0.54
09 Nov 2024 12:47:48
Epoch: 199 |train loss: 0.4227 |train accuracy: 91.92 |validation loss: 0.9576 |validation accuracy: 71.88 |learning rate: 3.1e-04 |train time: 2.92 |test time: 0.45
Best accuracy: 78.12
```

2 Explain Pruning briefly (What/Why/How)

2.1 What is Pruning

剪枝是一種選擇性地從神經網絡中移除權重、神經元，甚至是整個層的過程，並儘量不顯著影響模型的整體性能。剪枝的核心思想是去除對預測貢獻較小的冗餘或低重要性的部分。

2.2 Why Prune a Model

1. 提升效率：剪枝後的模型佔用更少的記憶體，執行速度更快，更適合在資源有限的邊緣設備上部署。
2. 降低成本：減少模型的大小和計算需求可以降低推理成本，減少能耗。
3. 增強泛化能力：剪枝透過簡化模型可以減少過擬合，從而可能提升模型對新數據的泛化能力。

2.3 How to Prune

1. 基於幅度的剪枝：這種方法根據權重的絕對值來移除較小的權重，假設低幅度的權重對模型性能影響較小。
2. 結構化剪枝：在這種方法中，整個通道、過濾器或層會被移除。結構化剪枝更容易優化，並且更符合硬體的限制。
3. 非結構化剪枝：這種方法移除單個權重而不遵循特定的結構，儘管可以實現稀疏化，但在硬體實現上可能更困難。
4. 迭代剪枝與微調：常見的做法是分步進行剪枝，在每次剪枝後對模型進行微調，以恢復可能損失的準確性。

2.4 Pruning Implementation

```
# you can choose your pruning rate
pruning_rate = 0.01
```

Fig. 1: Pruning rate

```
Number of parameter: 0.27M
```

Fig. 2: Number of parameter before pruning

此程式碼對模型的 BatchNorm1d 層進行剪枝，透過掩碼機制保留權重較大的通道，並更新模型的配置資訊。這樣可以減少模型的計算資源需求，提高運行效率，同時儘可能保持原始模型的表現。

```
layer index: 4   total channel: 40       remaining channel: 39
layer index: 11         total channel: 256      remaining channel: 250
layer index: 17         total channel: 256      remaining channel: 255
layer index: 23         total channel: 256      remaining channel: 253
layer index: 29         total channel: 256      remaining channel: 254
layer index: 35         total channel: 160      remaining channel: 160
Pre-processing Successful!
```

Fig. 3: Record the renaming weight

對 BatchNorm1d 層進行剪枝，通過遮罩的方式將低於閾值的通道權重置為零，以減少模型參數和計算量。

```
In shape: 39, Out shape 39.
In shape: 39, Out shape 250.
In shape: 250, Out shape 250.
In shape: 250, Out shape 255.
In shape: 255, Out shape 255.
In shape: 255, Out shape 253.
In shape: 253, Out shape 253.
In shape: 253, Out shape 254.
In shape: 254, Out shape 254.
In shape: 254, Out shape 160.
```

Fig. 4: Generate the pruned model

剪枝過後的檢查點檔案中載入一個深度學習模型。載入剪枝後儲存的 SincNet 模型結構和權重，以便進行後續推理或微調。

```
SincNet(
  (sincconv): _Layer(
    (conv0): SincConv1d()
    (logabs): LogAbs()
    (bn): BatchNorm1d(39, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (pool): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
  )
  (features): ModuleList(
    (0): _Layer(
      (conv0): Conv1d(39, 39, kernel_size=(25,), stride=(2,), groups=39)
      (conv1): Conv1d(39, 250, kernel_size=(1,), stride=(1,))
      (relu): ReLU()
      (bn): BatchNorm1d(250, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (pool): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
    )
    (1): _Layer(
      (conv0): Conv1d(250, 250, kernel_size=(9,), stride=(1,), groups=250)
      (conv1): Conv1d(250, 255, kernel_size=(1,), stride=(1,))
      (relu): ReLU()
      (bn): BatchNorm1d(255, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (pool): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
    )
    (2): _Layer(
      (conv0): Conv1d(255, 255, kernel_size=(15,), stride=(1,), groups=255)
      (conv1): Conv1d(255, 253, kernel_size=(1,), stride=(1,))
      (relu): ReLU()
      (bn): BatchNorm1d(253, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (pool): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
    )
    (3): _Layer(
      (conv0): Conv1d(253, 253, kernel_size=(9,), stride=(1,), groups=253)
      (conv1): Conv1d(253, 254, kernel_size=(1,), stride=(1,))
      (relu): ReLU()
      (bn): BatchNorm1d(254, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (pool): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
    )
    (4): _Layer(
      (conv0): Conv1d(254, 254, kernel_size=(9,), stride=(1,), groups=254)
      (conv1): Conv1d(254, 160, kernel_size=(1,), stride=(1,))
      (relu): ReLU()
      (bn): BatchNorm1d(160, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (pool): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
    )
  )
  (gap): AdaptiveAvgPool1d(output_size=1)
  (fc): Linear(in_features=160, out_features=10, bias=True)
)
```

Fig. 5: Pruned model

沒有 fine-tune 的情況下，剪枝後的模型準確率非常爛%。

```
Before fine-tune
Validation loss: 7.1118 Validation accuracy: 43.75
```

Fig. 6: Accuracy before fine-tune

```
Begin fine-tune...
08 Nov 2024 08:30:31
Epoch:   0 |train loss: 0.4794 |train accuracy: 84.91 |train time: 9.70
08 Nov 2024 08:30:36
Epoch:   1 |train loss: 0.3205 |train accuracy: 90.07 |train time: 5.22
08 Nov 2024 08:30:40
Epoch:   2 |train loss: 0.2496 |train accuracy: 91.53 |train time: 4.51
08 Nov 2024 08:30:45
Epoch:   3 |train loss: 0.2959 |train accuracy: 90.85 |train time: 4.88
08 Nov 2024 08:30:49
Epoch:   4 |train loss: 0.2560 |train accuracy: 91.82 |train time: 3.97
08 Nov 2024 08:30:54
Epoch:   5 |train loss: 0.2339 |train accuracy: 92.41 |train time: 4.23
08 Nov 2024 08:30:59
Epoch:   6 |train loss: 0.2557 |train accuracy: 91.72 |train time: 5.19
08 Nov 2024 08:31:03
Epoch:   7 |train loss: 0.2827 |train accuracy: 92.11 |train time: 4.00
08 Nov 2024 08:31:07
Epoch:   8 |train loss: 0.2756 |train accuracy: 90.36 |train time: 4.15
08 Nov 2024 08:31:12
Epoch:   9 |train loss: 0.1898 |train accuracy: 93.18 |train time: 5.28
Saving..
```

Fig. 7: Begin fine-tune

在 pruning 後，參數數量減少了 3%。

```
Number of parameter: 0.26M
```

Fig. 8: Number of parameter after pruning

在 pruning 後，進行 fine-tune，以恢復可能損失的準確性。

```
After fine-tune
Validation loss: 0.3866 Validation accuracy: 89.38
```

Fig. 9: Accuracy after fine-tune

3 Explain Quantization briefly (What/Why/How)

用於在模型推理時縮小模型的大小並加速計算。

3.1 What is Quantization

量化將深度學習模型中的浮點數（通常是 32 位元浮點數）轉換為較低精度的數值（例如 8 位元整數），以減少模型的參數大小和計算複雜度。

3.2 Why Quantize a Model

量化可以顯著減少模型所需的儲存空間和運算資源。這對於資源有限的嵌入式系統、行動裝置及邊緣設備特別重要。透過量化，模型大小縮小、計算速度加快，能夠減少延遲並降低能耗。

3.3 How to Quantize

1. 後量化（Post-training Quantization）：訓練模型後再進行量化，轉換浮點數權重和激活值為低精度整數。

2. 量化感知訓練（Quantization-Aware Training, QAT）：在訓練過程中模擬量化的效果，以提高模型的精度。QAT 在訓練時考慮量化的影響，因此效果通常比單純的後量化更好。

3.4 Quantization Implementation

```
EPOCH  =  200
BATCH_SIZE  =  64
LR  =  0.01
Weight_decay  =  1e-5
```

Fig. 10: Adjust hyper parameters

```
SincNet_Quat(
  (sincconv): _Layer(
    (conv): QuatSincConv1d(
      (w_quan): Quantize(num_of_bits=7)
    )
    (logabs): LogAbs()
    (relu): ReLU()
    (quan): Quantize(num_of_bits=8)
    (bn): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (pool): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
  )
  (features): ModuleList(
    (0): _Layer(
      (conv0): QuaternaryConv1d(32, 32, kernel_size=(25,), stride=(2,), bias=False)
      (conv1): QuaternaryConv1d(32, 32, kernel_size=(25,), stride=(2,), bias=False)
      (bn): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (quan): Quantize(num_of_bits=4)
      (relu): ReLU()
      (pool): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
    )
    (1): _Layer(
      (conv0): QuaternaryConv1d(32, 32, kernel_size=(9,), stride=(1,), bias=False)
      (conv1): QuaternaryConv1d(32, 64, kernel_size=(9,), stride=(1,), bias=False)
      (bn): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (quan): Quantize(num_of_bits=4)
      (relu): ReLU()
      (pool): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
    )
    (2-4): 3 x _Layer(
      (conv0): QuaternaryConv1d(64, 64, kernel_size=(9,), stride=(1,), bias=False)
      (conv1): QuaternaryConv1d(64, 64, kernel_size=(9,), stride=(1,), bias=False)
      (bn): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (quan): Quantize(num_of_bits=4)
      (relu): ReLU()
      (pool): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
    )
  )
  (gap): AdaptiveAvgPool1d(output_size=1)
  (quan_gap): Quantize(num_of_bits=8)
  (fc): Linear(in_features=64, out_features=10, bias=True)
)
```

Fig. 11: SincNet model with quantization

```
09 Nov 2024 12:47:08
Epoch: 186 |train loss: 0.4128 |train accuracy: 90.94 |validation loss: 0.9300 |validation accuracy: 73.12 |learning rate: 6.3e-04 |train time: 2.59 |test time: 0.42
09 Nov 2024 12:47:11
Epoch: 187 |train loss: 0.3754 |train accuracy: 90.36 |validation loss: 1.0608 |validation accuracy: 71.88 |learning rate: 6.3e-04 |train time: 2.54 |test time: 0.43
09 Nov 2024 12:47:14
Epoch: 188 |train loss: 0.2729 |train accuracy: 91.43 |validation loss: 0.9976 |validation accuracy: 71.88 |learning rate: 6.3e-04 |train time: 2.49 |test time: 0.44
09 Nov 2024 12:47:17
Epoch: 189 |train loss: 0.4801 |train accuracy: 91.63 |validation loss: 0.8760 |validation accuracy: 70.00 |learning rate: 6.3e-04 |train time: 2.66 |test time: 0.50
09 Nov 2024 12:47:20
Epoch: 190 |train loss: 0.3482 |train accuracy: 90.75 |validation loss: 0.9277 |validation accuracy: 71.88 |learning rate: 6.3e-04 |train time: 2.74 |test time: 0.43
09 Nov 2024 12:47:23
Epoch: 191 |train loss: 0.3300 |train accuracy: 91.53 |validation loss: 0.9160 |validation accuracy: 71.88 |learning rate: 3.1e-04 |train time: 2.48 |test time: 0.42
09 Nov 2024 12:47:26
Epoch: 192 |train loss: 0.3639 |train accuracy: 92.41 |validation loss: 0.9165 |validation accuracy: 71.88 |learning rate: 3.1e-04 |train time: 2.52 |test time: 0.43
09 Nov 2024 12:47:29
Epoch: 193 |train loss: 0.2956 |train accuracy: 90.94 |validation loss: 0.9528 |validation accuracy: 71.88 |learning rate: 3.1e-04 |train time: 2.51 |test time: 0.47
09 Nov 2024 12:47:33
Epoch: 194 |train loss: 0.3201 |train accuracy: 91.04 |validation loss: 0.9134 |validation accuracy: 72.50 |learning rate: 3.1e-04 |train time: 2.88 |test time: 0.45
09 Nov 2024 12:47:36
Epoch: 195 |train loss: 0.4036 |train accuracy: 90.85 |validation loss: 0.9491 |validation accuracy: 70.62 |learning rate: 3.1e-04 |train time: 2.53 |test time: 0.43
09 Nov 2024 12:47:38
Epoch: 196 |train loss: 0.2930 |train accuracy: 89.68 |validation loss: 0.9496 |validation accuracy: 70.62 |learning rate: 3.1e-04 |train time: 2.53 |test time: 0.42
09 Nov 2024 12:47:41
Epoch: 197 |train loss: 0.3156 |train accuracy: 91.82 |validation loss: 0.8842 |validation accuracy: 72.50 |learning rate: 3.1e-04 |train time: 2.54 |test time: 0.43
09 Nov 2024 12:47:45
Epoch: 198 |train loss: 0.3428 |train accuracy: 89.87 |validation loss: 0.9338 |validation accuracy: 73.75 |learning rate: 3.1e-04 |train time: 2.77 |test time: 0.54
09 Nov 2024 12:47:48
Epoch: 199 |train loss: 0.4227 |train accuracy: 91.92 |validation loss: 0.9576 |validation accuracy: 71.88 |learning rate: 3.1e-04 |train time: 2.92 |test time: 0.45
Best accuracy: 78.12
```

Fig. 12: Training

這個圖表展示了隨著訓練過程（即 epoch）進行，學習率是如何變化的。發現在 Epoch 75 ~ 125 之間，學習率有一個較大的變化，這是因為在這個區間內，模型的準確率有一個較大的提升，因此學習率也有一個較大的變化。

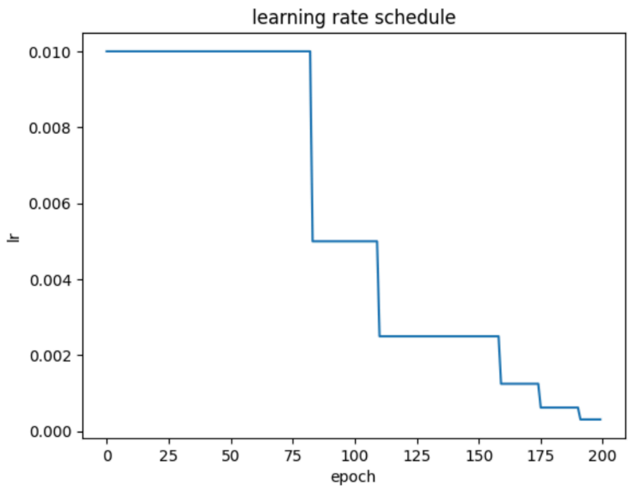

Fig. 13: Learning rate schedule

繪製訓練過程中的準確度，顯示了 SincNet 模型在訓練過程中的準確度變化，並比較了訓練集和驗證集的表現。通過這個圖表，您可以觀察到訓練準確度和驗證準確度的趨勢，例如是否存在過擬合（訓練準確度上升但驗證準確度下降），或訓練過程中的學習效果。

以這張圖來說，沒有過擬和的情形。

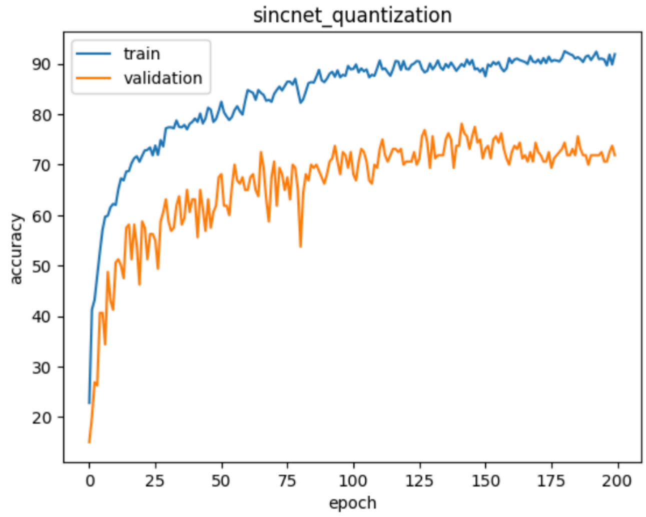

Fig. 14: sincnet_quantization

4 Compare the difference from above methods of compression

```
SincNet(
  (sincconv): _Layer(
    (conv0): SincConv1d()
    (logabs): LogAbs()
    (bn): BatchNorm1d(39, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (pool): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
  )
  (features): ModuleList(
    (0): _Layer(
      (conv0): Conv1d(39, 39, kernel_size=(25,), stride=(2,), groups=39)
      (conv1): Conv1d(39, 250, kernel_size=(1,), stride=(1,))
      (relu): ReLU()
      (bn): BatchNorm1d(250, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (pool): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
    )
    (1): _Layer(
      (conv0): Conv1d(250, 250, kernel_size=(9,), stride=(1,), groups=250)
      (conv1): Conv1d(250, 255, kernel_size=(1,), stride=(1,))
      (relu): ReLU()
      (bn): BatchNorm1d(255, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (pool): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
    )
    (2): _Layer(
      (conv0): Conv1d(255, 255, kernel_size=(15,), stride=(1,), groups=255)
      (conv1): Conv1d(255, 253, kernel_size=(1,), stride=(1,))
      (relu): ReLU()
      (bn): BatchNorm1d(253, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (pool): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
    )
    (3): _Layer(
      (conv0): Conv1d(253, 253, kernel_size=(9,), stride=(1,), groups=253)
      (conv1): Conv1d(253, 254, kernel_size=(1,), stride=(1,))
      (relu): ReLU()
      (bn): BatchNorm1d(254, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (pool): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
    )
    (4): _Layer(
      (conv0): Conv1d(254, 254, kernel_size=(9,), stride=(1,), groups=254)
      (conv1): Conv1d(254, 160, kernel_size=(1,), stride=(1,))
      (relu): ReLU()
      (bn): BatchNorm1d(160, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (pool): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
    )
  )
  (gap): AdaptiveAvgPool1d(output_size=1)
  (fc): Linear(in_features=160, out_features=10, bias=True)
)
```

Fig. 15: SincNet with pruning

```
SincNet_Quat(
  (sincconv): _Layer(
    (conv): QuatSincConv1d(
      (w_quan): Quantize(num_of_bits=7)
    )
    (logabs): LogAbs()
    (relu): ReLU()
    (quan): Quantize(num_of_bits=8)
    (bn): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (pool): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
  )
  (features): ModuleList(
    (0): _Layer(
      (conv0): QuaternaryConv1d(32, 32, kernel_size=(25,), stride=(2,), bias=False)
      (conv1): QuaternaryConv1d(32, 32, kernel_size=(25,), stride=(2,), bias=False)
      (bn): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (quan): Quantize(num_of_bits=4)
      (relu): ReLU()
      (pool): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
    )
    (1): _Layer(
      (conv0): QuaternaryConv1d(32, 32, kernel_size=(9,), stride=(1,), bias=False)
      (conv1): QuaternaryConv1d(32, 64, kernel_size=(9,), stride=(1,), bias=False)
      (bn): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (quan): Quantize(num_of_bits=4)
      (relu): ReLU()
      (pool): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
    )
    (2-4): 3 x _Layer(
      (conv0): QuaternaryConv1d(64, 64, kernel_size=(9,), stride=(1,), bias=False)
      (conv1): QuaternaryConv1d(64, 64, kernel_size=(9,), stride=(1,), bias=False)
      (bn): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (quan): Quantize(num_of_bits=4)
      (relu): ReLU()
      (pool): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
    )
  )
  (gap): AdaptiveAvgPool1d(output_size=1)
  (quan_gap): Quantize(num_of_bits=8)
  (fc): Linear(in_features=64, out_features=10, bias=True)
)
```

Fig. 16: SincNet with quantization

壓縮方法	**壓縮類型**	**對模型大小的影響**	**對計算的影響**	**主要目標**	**應用場景**
原始模型	無	保持完整參數和精度	計算量和內存需求較高	提供最高精度	對資源無限制或精度要求高的場景
剪枝	稀疏性基礎	通過將許多權重設為零來減小模型大小	如果硬體支持稀疏矩陣運算，則加速推理	減少複雜性和過擬合	邊緣設備、內存受限的環境
量化	精度基礎	通過減少位寬來減小內存大小	在支持低精度運算的硬體上加速計算	提高內存和計算效率	移動設備、邊緣設備、嵌入式系統
知識蒸餾	基於模型	通過將大模型的知識轉移到小模型來減少參數	不會顯著影響速度，除非進行優化	將知識轉移到小型模型	壓縮大模型以便在邊緣設備上部署
低秩分解	分解基礎	通過分解權重矩陣來減少參數數量	減少計算量，但可能依賴於架構	減少參數和計算量	需要壓縮參數矩陣的模型

Table 1: 壓縮方法比較

如果追求精度：原始模型通常更好，適合對精度要求較高的場景。

如果追求效能和平衡精度：可以選擇剪枝或量化過的模型。一般來說，剪枝適合減少較多冗餘參數的模型，量化則適合在硬體支援低精度運算的場景中提升速度和減小大小。

剪枝 (Pruning) 和量化 (Quantization) 是經常一起使用的壓縮方法，適合於需要顯著減小模型大小並加速推理的場景。

(補充: 知識蒸餾 (Knowledge Distillation) 可以與其他壓縮方法結合使用，適用於將大模型的知識轉移到小型學生模型中的情況，從而在保證性能的同時縮小模型。低秩分解 (Low-Rank Factorization) 適用於那些具有大規模權重矩陣的

模型，尤其是全連接層。)

實驗結果 Pruning 和 Quantization 的結果都有不錯的效果，但是 Pruning 的效果比 Quantization 好，因為 Pruning 可以將不重要的權重直接設為 0，而 Quantization 只能將權重的位寬減小，無法直接減少權重的數量。然後，不知道為什麼 SincNet 的 Pruning 效果比原始模型好，這可能是因為 SincNet 的模型本身就有很多冗餘的權重，Pruning 可以將這些冗餘的權重去掉，從而提高模型的性能。