

# NYCU IEE Deep Learning

## Lab 4 Report: Model Compression

陳柏翔 313510156

### 1 Introduction

本次作業旨在完成模型輕量化的任務，我們需要分別在 Task 1 / Task 2 中透過 Pruning 與 Post-Training Quantization (PTQ) 的方式來壓縮模型。而在 Task 1 中，我最終達到 92.01% 的 Sparsity，詳見章節 3.4；以及在 Task 2-2 中達到 92.70% 的準確率，詳見章節 5.3。

### 2 Dataset and Model

#### 2.1 Dataset

在本次的作業中，我們所使用的資料集為 CIFAR-10 [1]，如同 Figure 1 所示，其中包含 10 種類別、總共 60,000 張  $32 \times 32$  的彩色圖片，而 Training Set / Testing Set 的圖片張數分別為 50,000 / 10,000。

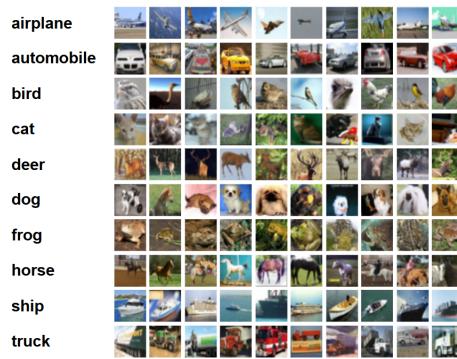


Figure 1: CIFAR-10 Dataset [1]

#### 2.2 Model

模型的部分則是使用 ResNet-20 [2]，這個模型是針對 CIFAR-10 Dataset [1] 所設計的特化版 ResNet。在本次作業中，此模型作為我們要 Pruning 與 Quantization 的主要模型。

而最初模型的 Pre-trained Weights 載入時，ResNet-20 [2] 於 Testing Set 上的準確率為 92.59%(或 92.60%)，我們在 Task 1 中需要在 Pruning 後仍保持 90.0% 以上的準確率，以及在 Task 2 中 PTQ 後測得的準確率越高越好。

### 3 Task 1: Pruning

#### 3.1 Global Pruning

在符合 Task 1 題目要求的情況下，如果使用 Layer-wise 的方式進行 Pruning 會很難達到 90% 以上的 Sparsity，這是由於每個 Layer 的重要程度不相同。根據課程講義上的內容能發現到，過去的研究顯示越靠前面的 Convolution Layer 會越重要，也就是在提取出來的特徵上容錯率較低，因此比較難設置一個統一比例就讓所有 Layers 都達到同樣的 Sparsity。

因此我改成使用 Global Pruning 的方式來進行，比起一層一層的設置，它會直接將整體模型的參數一起考量，再從中挑選出數值較小的參數進行 Pruning，這樣就能有效的選擇移除在整體模型中較不重要的參數，經過多次測試後發現這樣就能到輕易達到 90% 以上的 Sparsity。

### 3.2 BatchNorm Based Filter Pruning

雖然只考量 Convolution 與 Linear 的參數就能達到很好的 Pruning 效果，但我還另外根據 Batch Normalization Layer 的參數進行 Pruning。具體作法是根據 BatchNorm 的參數  $\gamma$  將對應 Convolution 的某些 Filters 紛移除掉，例如某個 Channel 對應的  $\gamma$  參數剛好趨近於 0 的話，該 Channel 先前的 Convolution 實際就不太重要，所以可以把在那個 Channel 上的 Filter 參數設為 0。

另外，我也考慮過以相乘的方式融合 Convolution 與 BatchNorm 的參數來進行 Pruning，但結果顯示不佳，因此我選擇只考量  $\gamma$  來進行 Pruning。此外，我只有在最初使用一次 BatchNorm Based Filter Pruning，而後續都是單純的 Global Pruning，我發現這樣做有助於模型在第一次 Pruning 後的收斂速度，但後續再做可能會使模型難以訓練回升準確率。

### 3.3 Training Strategy

在 Pruning 的過程中，最重要的就是透過 Retraining 來回升模型的準確率，但如果一次 Pruning 掉太多參數的話，模型會難以甚至無法回升準確率，因此需要分成多階段重複 Prune-Training 來慢慢達成目標，在 Task 1 中，所有訓練相關的設置如 Table 1 所示。

<b>Number of Prune-Training Phases</b>	12	<b>Pruning Method</b>	L1Unstructured
<b>Batch Size</b>	64	<b>Optimizer</b>	AdamW
<b>Learning Rate</b>	$10^{-3}$	<b>Scheduler</b>	CosineAnnealingLR
<b>Maximum Epochs per Phase</b>	150	<b>Loss Function</b>	Cross-Entropy Loss

Table 1: Training Settings

此外，我也使用了 Data Augmentation 來提升訓練資料的多樣性。除了基本的隨機裁剪 (Crop) 與水平翻轉 (Horizontal Flip) 外，還採用了 RandAugment [3]，這是一組專為 CIFAR-10/100、ImageNet 等資料集設計的多種增強方法。

### 3.4 Experiment

Pruning 與 Retraining 的過程如 Figure 2 所示，可以明顯看到每次 Pruning 後 Loss 就會突然上升並在多個 Epochs 後慢慢回降。而最終 Pruning 後的結果及比較表如 Figure 3 與 Table 2 所示，在 Table 2 中，可以發現第一個 Convolution 與最後的 Fully-Connect 相較於中間的各個 Convolution Layers 都更為重要。

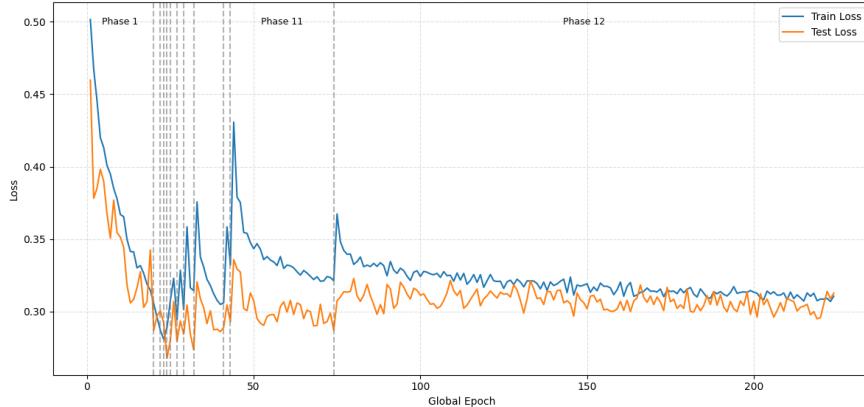


Figure 2: Training and Testing Loss Across Phases

```

Accuracy on CIFAR-10 test set: 90.56%
Overall Sparsity = 92.01% (249241/270896)

=====
Congratulations! You've achieved the goals of this task. Remember to save your model!
You can also try increasing sparsity further to earn a higher score!

```

Figure 3: Task 1 Result Screenshot

Module Name	Params Before	Params After	Difference
conv1	432	65	-84.95%
layer1.0	4,608	388	-91.58%
layer1.1	4,608	386	-91.62%
layer1.2	4,608	400	-91.32%
layer2.0	14,336	1,381	-90.37%
layer2.1	18,432	1,530	-91.70%
layer2.2	18,432	1,522	-91.74%
layer3.0	57,344	4,747	-91.72%
layer3.1	73,728	5,805	-92.13%
layer3.2	73,728	5,033	-93.17%
fc	640	398	-37.81%
<b>Total</b>	270,896	21,655	-92.01%

Table 2: Before and After Pruning Comparison

(註：實際上每個 layerX.X 包含 2 至 3 個 conv Modules，但由於排版考量而沒有展開列出。)

## 4 Task 2-1: PTQ with PyTorch FX Graph Mode

在 Task 2-1 中，我們需要透過 PyTorch 內建的 FX Graph Mode 來實現 Post-Training Quantization (PTQ)，這種方法能自動解析模型的計算圖並插入量化觀察器。FX Graph Mode 會在前向傳遞過程中追蹤運算流程，建立可轉換的中介表示 (FX Graph)。接著我們可利用此圖進行層融合、統計收集與後續的量化轉換，以實現高效的 Integer Inference (整數型態的 Inference)。

### 4.1 Calibration

在 PTQ 中，除了在計算式上的設計以外，最重要的是要先觀察數值的分布，特別像是此章節中我們使用的 Observer 是以記錄到的最大與最小值作為量化的範圍，極端數值的出現就特別容易影響 PTQ 後的模型表現。

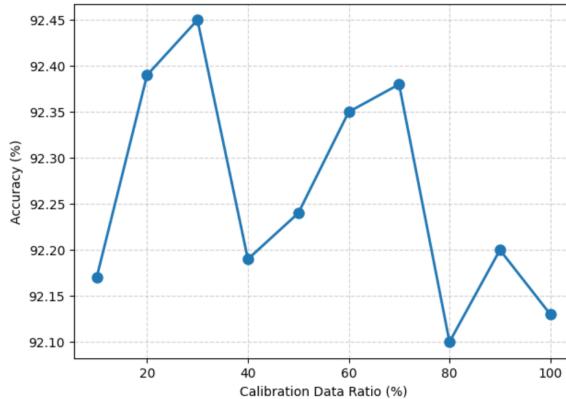


Figure 4: PTQ Accuracy versus Calibration Data Size

如 Figure 4 所示，我從 10% 至 100% 分別設置了 10 種 Calibration Data Size 去進行校準，並且以隨機分配的方式從原本的 Training Set 切出子集進行校準。觀察後可以發現到，隨著 Calibration Data Size 的增加，準確率反而呈現下降的趨勢，且多次實驗皆有相同現象。

我認為這是因為使用 MinMaxObserver 時，較大的樣本更容易包含極端值，導致量化區間變寬，整體精度下降。由於大多數數值並不落在這些極端區域，反而使中間區域的分辨能力降低。因此，如果使用 MinMaxObserver，我建議僅採用適量的校準資料而非大量 (I.e. 僅取用一部分)；或者另一個做法是改用能考慮整體數值分布的 Observer (E.g. HistogramObserver)，在使用全部資料的情況下可以得到更穩定的量化效果，更多實驗詳見附錄章節 A。

## 5 Task 2-2: Manual PTQ

在 Task 2-2 中，我們要自己實作 Observer 並且在資料校準後進行 Scale、Zero-point 的計算來完成 PTQ。由於量化的方式已經被寫好在 `resnet20_in8.py` 當中，固定為 Uniform Integer Quantization，也就是只能使用 Affine 的方式進行映射然後填入 Scale、Zero-point 做設定，所以在選擇性沒有很多的情況下我採用 Asymmetric 的方式進行量化以提高準確率。

### 5.1 Observation

與 Task 2-1 相同，我們需要定義 Observer 來觀察原始 CifarResNet 中各個 Modules 的 Activations (I.e. Outputs) 數值分布以利後續進行量化。而在不利用計算圖的情況下，QuantizedCifarResNet 相較於 CifarResNet 多了 Addition Module 並且少了 BatchNorm Module，這兩點會使得要觀察數值的位置有所調整。

首先，雖然少了 BatchNorm Module，但其實就是融合到它前方的 Convolution Layer 中，因此我決定直接在所有的 BatchNorm Layers 掛上 MinMaxObserver，來觀察融合後的 QuantizedConv2d 與 QuantizedConvReLU2d 在 Activations 上所需的量化參數。

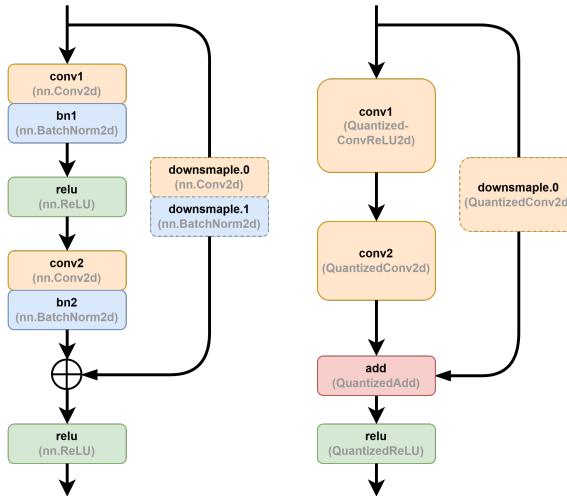


Figure 5: Original (Left) and Quantized (Right) ResNet Basic Block

而相較於容易處理的 Convolution Modules，Addition Module 如 Figure 5 所示，原本並沒有對應的 Module 存在，因此無法掛上 Observer 進行 Activations 的觀測。儘管我想透過觀測第二個 ReLU Module 的 Inputs 取而代之，卻又因為 ReLU Module 並沒有分別命名為 `relu1` 與 `relu2` 而造成觀測上的混淆，也就是如果直接在名為 `relu` 的 Module 掛上 MinMaxObserver，最終得到的結果會是同時考量了第一與二次 ReLU 的 Inputs 所記錄到的最大及最小值。

為此我想出了一個作法，就是同樣在 ReLU Module 掛上 MinMaxObserver，但是在 Observer 中新增了一個 Counter，每執行一次 `__call__()` 函數就會加 1、如果加 1 會等於 2 的話則立即重置為

0，因此若想只觀測第二個 ReLU Module 的話，僅需在 Counter 為 1 的時候進行紀錄就可以了，實作內容詳見 `task2_2_quantization_manual.ipynb`。

## 5.2 Compute Quantization Parameters

在前面觀察完各個 Quantized Layer 所需的最小與最大值後，下一階段是利用最大與最小值計算 Scale 與 Zero-point。在進行量化前，我先對模型中的 Convolution 與 BatchNorm 進行融合。對於輸入特徵圖  $x$ ，融合前與融合後的等效關係如下公式：

$$y = \text{BN}(\text{Conv}(x)) = \gamma \cdot \frac{(W * x + b) - \mu}{\sqrt{\sigma^2 + \varepsilon}} + \beta$$

其中， $\gamma$ 、 $\beta$  為 BatchNorm 的縮放與偏移參數， $\mu$ 、 $\sigma^2$  為其移動平均的 Mean 與 Variance， $\varepsilon$  則是穩定項。整理簡化過後，可以將上式重新寫成與一般 Convolution 相同的形式：

$$\begin{aligned} y &= W' * x + b' \\ W' &= W \cdot \frac{\gamma}{\sqrt{\sigma^2 + \varepsilon}}, \quad b' = \beta + \frac{\gamma}{\sqrt{\sigma^2 + \varepsilon}}(b - \mu) \end{aligned}$$

如此便完成了 BatchNorm 與 Convolution 的融合，使後續的量化流程能直接在融合後的權重上進行，接著就可以進入量化參數的計算階段。其中又可分為 Symmetric 與 Asymmetric 兩種映射方式，雖然我兩種方法都有實作，但為了更高的準確率，我實際使用到的只有後者。具體來說，假設量化前最大及最小值為  $V_{max}$ 、 $V_{min}$ ，量化後變為  $Q_{max}$ 、 $Q_{min}$ ，則 Scale (浮點數) 與 Zero-point (整數) 的計算式如下：

$$S = \frac{V_{max} - V_{min}}{Q_{max} - Q_{min}}, \quad Z = \text{Round}(Q_{min} - \frac{V_{min} - 0}{S})$$

若 Zero-point 計算後超出範圍  $[Q_{min}, Q_{max}]$ ，則會再使用 `torch.clamp()` 函數將其修改為  $Q_{min}$  或  $Q_{max}$ 。而上述做法只是普通的以最大與最小值進行映射，由於我認為這樣做很容易受到極端值影響，因此我嘗試以新的 Scale ( $S'$ ) 設為原本 Scale ( $S$ ) 的  $\alpha$  倍，並且測試以不同位置作為縮放中心，實驗結果詳見下一章節。

$$\begin{aligned} S' &= \alpha S = \alpha \times \frac{V_{max} - V_{min}}{Q_{max} - Q_{min}} \\ Z'_{center@min} &= \text{Round}(Q_{min} - \frac{V_{min} - 0}{S'}), \quad Z'_{center@zero} = \text{Round}(Q_{min} - \frac{\alpha V_{min} - 0}{S'}) \end{aligned}$$

## 5.3 Experiment

根據章節 4 的結果，可以知道使用 MinMaxObserver 時最好只使用適量的一部分 Training Data 進行校準而非全部，因此我選擇先以原始 MinMaxObserver 測試不同數量下的 Training Data (無隨機排序) 所得的準確率結果。

Calibration Size	Test Accuracy
100	92.37%
1,000	92.41%
5,000	92.50%
10,000	92.50%
25,000	92.43%
50,000	92.43%

Table 3: Test Accuracy Under Different Calibration Size

根據 Table 3 的測試結果可知，當取用 5,000 至 10,000 筆 Training Data 作為 Calibration Data 時，模型的準確率表現較佳。因此，後續實驗皆固定使用 10,000 筆資料作為 Calibration Data 的數量。在決定合適的 Data Size 後，我進一步測試前一小節所討論的不同量化方法，將  $\alpha$  參數設定為 1.00 至 0.95 之間的多組數值進行比較。

$\alpha$	Center@min	Center@zero
0.95	91.95%	92.42%
0.96	92.12%	92.43%
0.97	92.29%	92.48%
0.98	92.48%	92.46%
0.99	92.70%	92.65%
1.00	92.50%	92.50%

Table 4: Test Accuracy versus Different Scale (Calibration Size = 10,000)

由 Table 4 可觀察到，以零點為中心縮放 Scale 的結果整體較為穩定且合理；然而，以  $V_{min}$  為中心縮放 Scale 的方法則在部分情況下可達到最高準確率，因此最終採用此方法搭配  $\alpha = 0.99$  作為量化設定。

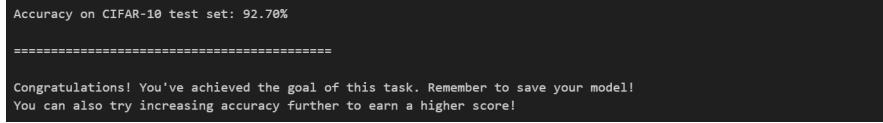


Figure 6: Task 2-2 Result Screenshot

最終成果如 Figure 6 所示，雖然原本的 ResNet-20 在載入 Pre-trained Weights 後的測試準確率為 92.60%，但在使用上述方法進行 PTQ 後，模型的準確率反而提升至 92.70%。我認為這是因為量化的行為本身可視為一種非線性的轉換，雖然在多數情況下會對模型性能造成負面影響，但在特殊情況下，這種非線性特性可能恰好提高模型的準確率。

## 6 Feedback

在本次的作業中，第一次認真了解到 Deep Learning 中的 Pruning 與 Quantization 是如何實作的，讓我受益良多。此外，很感謝助教在 .ipynb 檔中附上充分的註解與說明，甚至還有部分範例程式碼，有助於大家快速上手此 Lab 所要實作的東西。

如果要對這個 Lab 給一點點建議的話，我發現在 Weights 和 Activations 的量化上雖然都是變為 8-bit 整數，但是前者是 Signed [-128, 127] 後者則是 Unsigned [0, 255]，雖然這可能是考量權重與輸出的數值特性而特別設計的，但或許能在 resnet20\_int8.py 中的各個函數開頭也註解說明會更加明瞭。整體來說這個 Lab 很棒，謝謝教授與助教的用心教導！

## References

- [1] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.
- [2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.
- [3] Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. Randaugment: Practical automated data augmentation with a reduced search space, 2019.

## A PTQ Observers Comparison

在 Task 2-1 中，我們以 MinMaxObserver 進行 PTQ 的測試，而 Figure 7 是同樣的實驗但同時比較其他 Observers 的結果。

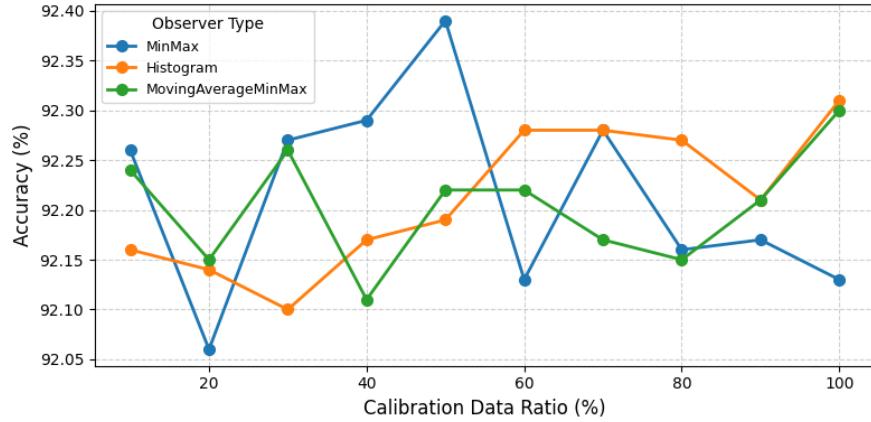


Figure 7: PTQ Accuracy versus Calibration Data Size

可以觀察到，在僅使用部分訓練資料進行校準時，MinMaxObserver 的結果波動較大、表現不穩定；然而當使用 90% 或 100% 的訓練資料時，HistogramObserver 與 MovingAverageMinMaxObserver 的表現明顯優於 MinMaxObserver。

雖然 MinMaxObserver 曾出現過一次整體最佳的結果，但這可能只是因為恰好遇到較理想的量化條件。實際應用於 PTQ 時，我認為選擇 HistogramObserver 會是更穩定且可靠的方案。