Lab04 Report – Deep Learning

學生：梁皓鈞，學號：314580042，系所：前瞻半導體研究所晶片組

一、 提高模型稀疏度的方法

此作業資料集是 CIFAR-10，我採用 Random Crop、Random Horizontal Flip 與 Normalize 等 Data Augmentation，Batch 大小設置為 128；損失函數為 Cross Entropy，優化器使用 SGD，其中 Momentum 設置為 0.9，Weight Decay 設置為5×10^{-4}，學習率採 Cosine Annealing。

對於剪枝我的核心做法是 Global Unstructured Pruning，透過 L1 Magnitude 作為判斷標準。首先，我蒐集除了第一層（conv1）以及最後的分類層（fc）以外的所有 Conv2d / Linear 權重，使用 Torch 的 Global Unstructured 進行整個網路的剪枝，Pruning Method 設為 L1 Unstructured。此方法會在所有指定參數上對權重的絕對值由小排到大，篩選出要歸零的元素，並於模組內建立對應的 Mask 與重參數化。

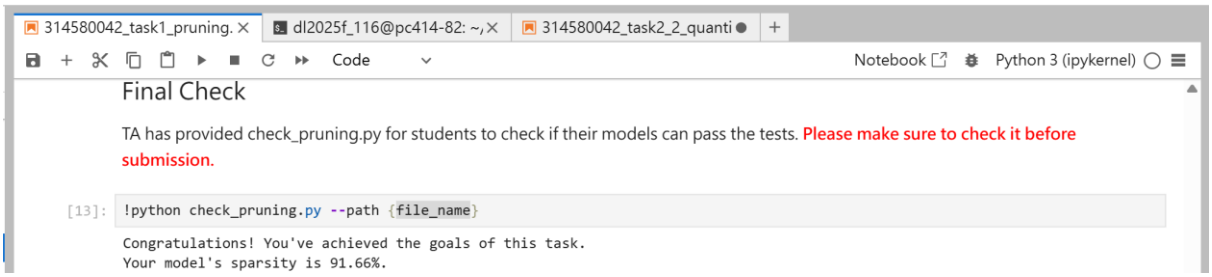
為了降低剪枝後靠近輸入層、輸出層端對預測準確度的損害，我對 conv1 與 fc 採前期輕量剪枝、後期跳過的策略：在低目標稀疏度時，依次對這兩層施以較小比例的剪枝；當目標稀疏度升高到超過某個閥值時便直接跳過這兩層的剪枝程序。

我設計了逐輪遞增目標稀疏度的調度，例如：30%、50%、70%、80%、90%、92%，每輪流程為：施加剪枝、以 Cosine 學習率進行微調、統計全模型稀疏度並測試準確率、以準確率高於 90%作為通過的門檻進行下一輪剪枝，若沒達標則儲存前一輪最佳 Checkpoint 並結束；其中，每輪微調後，將以 Prune Remove 將重參數化的部分固化為真零值，確保後續統計正確。實際下去跑後，從我實際剪枝後的結果可見：

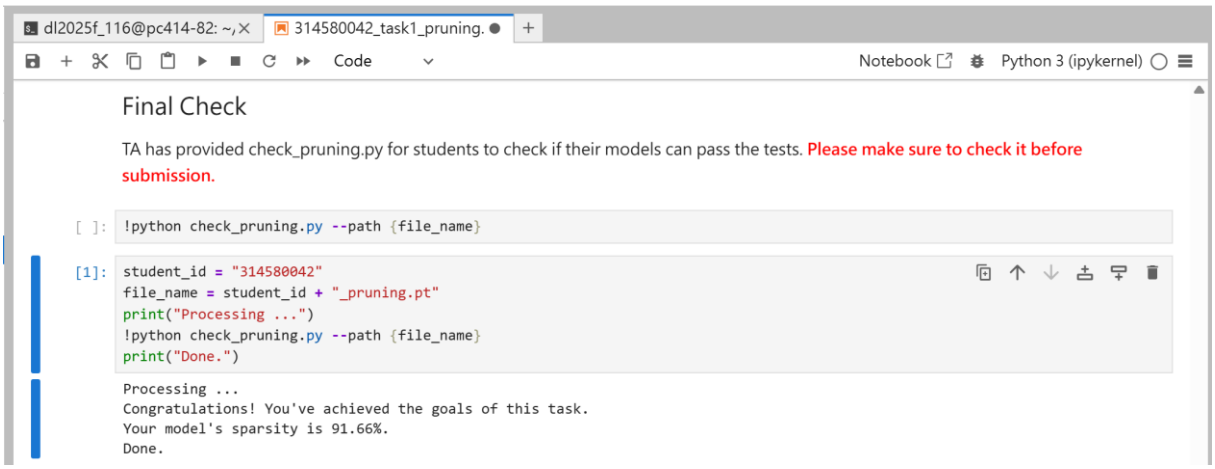
Target	Sparsity / Test Accuracy
30%	29.92% / 92.74%
50%	49.86% / 92.45%
70%	69.74% / 92.23%
80%	79.70% / 91.63%
90%	89.66% / 90.68%
92%	91.66% / 90.50%

最終保存的模型具有 91.66%的稀疏度與 90.50%的準確率。整體剪枝、微調過程中，可以發現當稀疏度提升至 90%以上時，準確率僅有溫和下降，證明 Global Magnitude Pruning 配合分段微調對於預測準確率的維持還算相當有效的。

最後，給助教查看以下此張照片是 Notebook 全部跑完的結果：



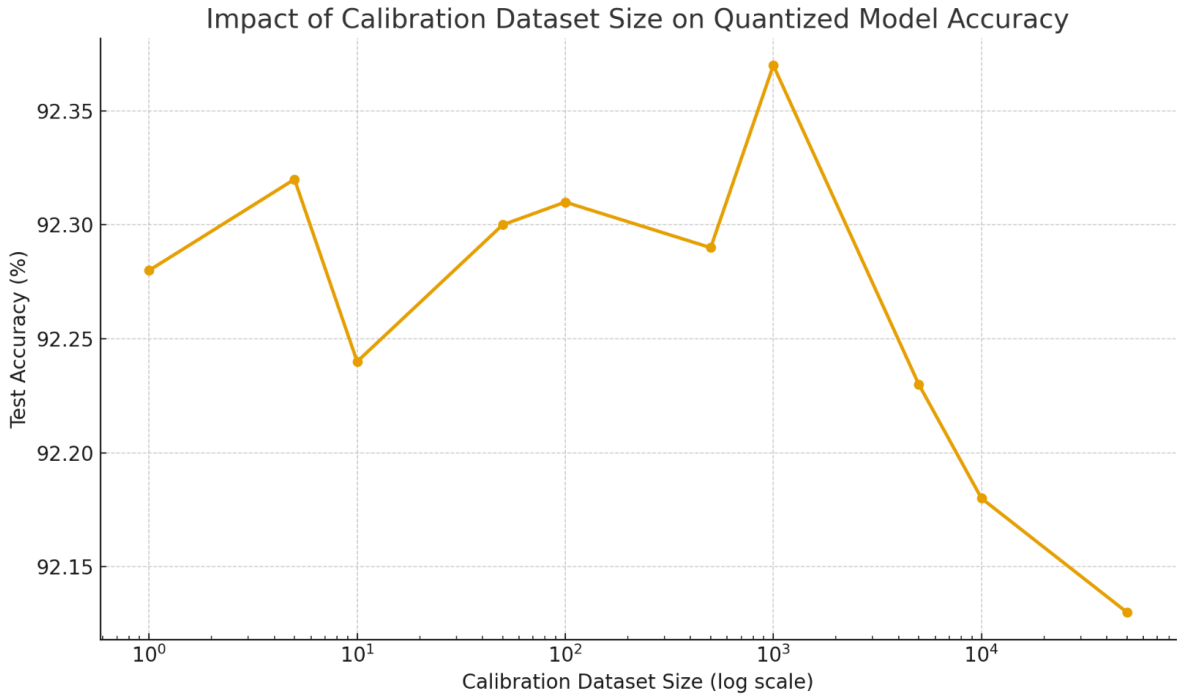
而以下此張照片是單純只跑 check_pruning.py 的結果：



二、 校準資料集大小與測試準確率

這部分助教已經有實作完整、與官方 FX 量化 API 一致的例子，整體採用 PyTorch FX Graph Mode 的 Post Training Static Quantization：一開始先以 Q Config Mapping 設定預設 Q Config，呼叫 Prepare FX 插入 Observers，使用 fbgemm backend 進行校準，再以 Convert FX 轉換為量化模型後，於測試集上進行測試。

根據助教的範例改寫，我以超過五種校準資料量來進行實驗，例如：1、5、10、50、100、500、1000、5000、10000，以及 50000 個樣本數，對應的校準後測試準確率依序為：92.28%、92.32%、92.24%、92.30%、92.31%、92.29%、92.37%、92.23%、92.18%、92.13%，製作成折線圖如下圖所示：



整體落在約 92.1%到 92.4%的窄幅範圍，其中校準資料量為 1000 筆時，約達到最佳校準精確度 92.37%。然而，進一步將校準資料量擴增至 10000 或 50000 時未見提升，反而有輕微下降。此結果與常見 PTQ 經驗吻合：Activation 由 Min Max Observer / Moving Average Min Max Observer 等完成的動態範圍統計通常在數百至一兩千筆範圍時趨於穩定；過度擴張校準集未必帶來更多好處；在過度擴張校準集的情況下，可能將尾端資料也考慮進去，導致量化後模型對整體資料的預測準確度反而下降。

根據我的實驗結果，以約 1000 筆的資料量做校準是最適當的甜蜜點；若資源有限可用 100 到 500 筆資料來做校準；若資料分佈有 Outlier，則在 500 到 2000 間做小範圍掃描尋找相對應的甜蜜；以下此張照片是 Notebook 全部跑完的結果，供助教參考：

三、 Scale、Zero-point 與量化權重的計算

此 Task 我透過手刻 PTQ 流程的方式完成客製 Observer 來蒐集、寫回參數。對 Activations 與 Weights 採用 Affine Quantization，其中對 Activations 採常用 uint8、對 Weights 採用 Per

Channel Int 8。其基本公式如下（以 Min 與 Max 線性定標）：

$$\text{scale} = (x_{max} - x_{min})/(q_{max} - q_{min})$$

$$\text{zeropoint} = q_{min} - \text{round}(x_{min}/\text{scale})$$

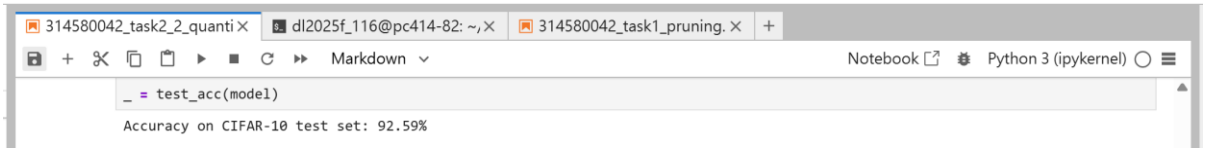
量化時：$q = \text{round}(x/\text{scale}) + \text{zeropoint}$，並將 q 截斷於$[q_{min}, q_{max}]$。備註：此即 ONNX Quantize Linear 與常見 Affine Quantization 的定義；在 PyTorch 的 Min Max Observer / Per Channel Min Max Observer 也以 Min / Max 統計推導對應的 Scale 與 Zero Point。同時，我也進行 Conv-BN Fusion：先將 Conv2d 與其後的 BatchNorm2d 融合，以固定 BN 統計量計算。若原本卷積層沒有偏置，則以零代入；而會有此操作的原因是由於在推論時能取消考慮 BN 層情況，來穩定權重量化。

$$w_{fused} = w \times \gamma/\sqrt{var + \varepsilon}$$

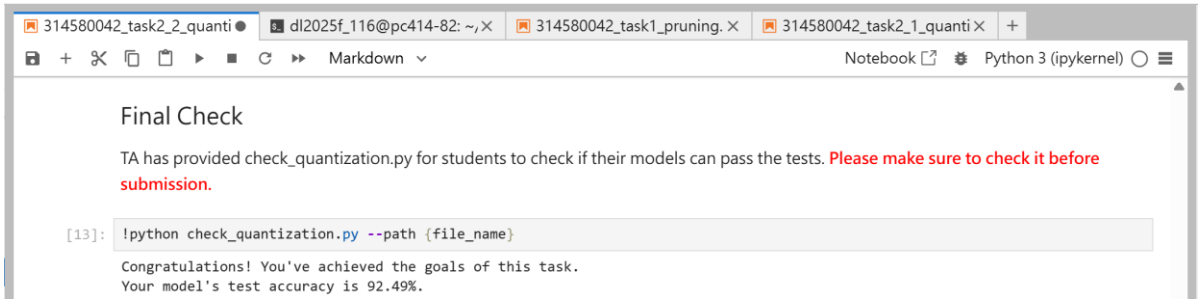
$$b_{fused} = (b - \mu) \times \gamma/\sqrt{var + \varepsilon} + \beta$$

在將這些模組整合時，我在輸入端設置 Input Quantizer，對每個 Conv (+ReLU)依該層輸出觀察值來去設定 Activation 的 uint8 Quantization Parameter；Weights 採 Per Channel Int8；殘差（Residual Add） 後再對齊量化參數；最後 Linear 層同樣以 Per Channel Int8 權重並保留 FP32 Bias。整體流程與 FX / PTQ 一致，只是改由我自行設定自定義 Observers 與參數推回。

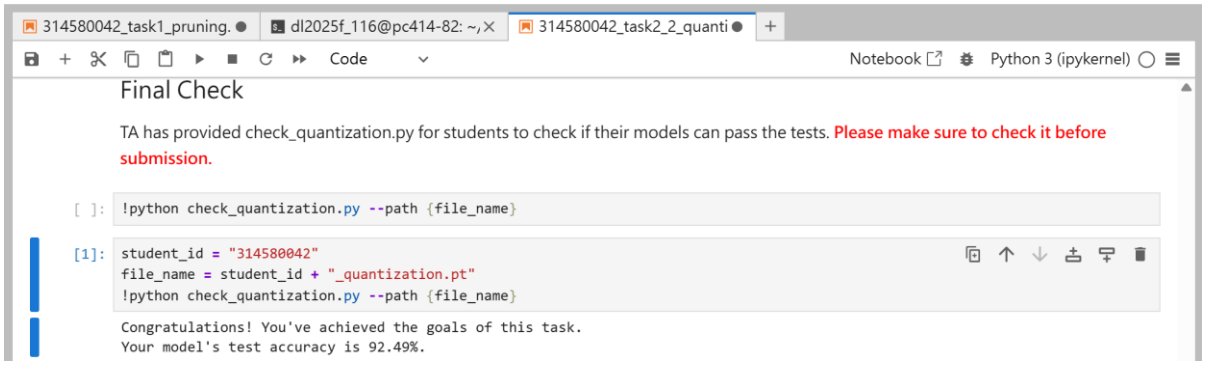
以下此張照片是原始模型的預測準確率供助教參考：



以下此張照片是 Notebook 全部跑完的結果供助教參考：



以下此張照片是只跑 check_quantization.py 的結果供助教參考：



四、 對作業的回饋

首先，助教在此作業中實作面向相當完整，涵蓋三個常見的模型壓縮方法，有 Magnitude-based Pruning、FX-based PTQ；可直接照著 API 流程走、以及 Manual PTQ；API 不支援的情況下須有手刻 Observers / 手動計算量化參數的經驗，對於我來說十分寶貴。

我還有上網查一些可延伸的方向：像是 Structured Pruning（Filter / Channel-wise）更利於實際硬體友善的推論加速，可嘗試與 Unstructured 串接，例如：先做 Structured Pruning 再做 Unstructured Pruning 加上微調。第二，是 Calibration 改良，例如：若遇到 Activation Outlier，可嘗試 Percentile / EMA Observers 或 MSE-based Clipping，以提升 Range 估計的穩定度。評估面可再加入模型大小、FLOPs 或實測 Latency，會有更全面的精度-資源-效率權衡。