LeNet-5 框架硬體效能基準測試
Assignment 3：Assignment3-Graph-Compare 報告

林彥成 Yan-Cheng Lin
課程：影像處理／深度學習相關課程
Email：mail.lin235711@gmail.com

***Abstract*—本報告以 DeepPCB 瑕疵分類資料集為任務背景，實作並比較 LeNet-5 在不同實作方式下的推論效能與準確率，包含純 NumPy 手寫版本，以及使用 GPU 加速的 PyTorch 與 TensorFlow（動態圖／靜態圖）版本。同時補充靜態圖編譯對延遲與吞吐量的影響，以及模型壓縮（剪枝）在儲存空間與推論速度之間的取捨。實驗結果顯示，TensorFlow 靜態圖在推論吞吐量上表現最佳，而手寫 NumPy 版本雖可用於理解底層運算，但與 GPU 框架仍有顯著差距。**

***Index Terms*—LeNet-5, DeepPCB, PyTorch, TensorFlow, Eager Execution, Static Graph, Benchmark, Pruning**

I. 實驗資訊與原始碼

資料集使用 DeepPCB Defect Dataset（Kaggle）。專案原始碼位於 GitHub：https://github.com/cvyancheng/Deep-Learning。主要檔案如下：

- `lenet_mlp_numpy.py`：純 NumPy 實作，包含訓練、驗證與測試評估流程。
- `lenet_framework_comparison.py`：框架效能對比工具，包含 PyTorch、TensorFlow（Eager/Graph）與剪枝（Pruning）實驗。

II. 模型架構與實作

A. LeNet-5 架構

本實驗於 PyTorch 與 TensorFlow 中實作標準 LeNet-5，輸入為 $32 \times 32 \times 3$ 彩色影像，並以 7 類別進行分類。網路結構概述如下：

- C1：6 個 5×5 卷積，ReLU。
- S2：2×2 Max Pooling。
- C3：16 個 5×5 卷積，ReLU。
- S4：2×2 Max Pooling。
- 全連接層：120 與 84 神經元。
- 輸出層：7 類別。

B. 框架實作差異

PyTorch 版本以 `nn.Module` 與 `nn.Sequential` 定義並使用 CUDA 加速。TensorFlow 版本以 Keras API 建模，並比較預設 Eager（動態圖）與使用 `@tf.function` 編譯後的 Static Graph（靜態圖）兩種推論模式。

III. 效能比較

本節數據基於 RTX 4060 Laptop GPU 硬體環境，針對完整測試集進行壓測比較。為便於對照，整理參數量、延遲、吞吐量、理論 FLOPs 與測試準確率如表 I。

TABLE I
各版本效能與準確率比較（測試集）

架構版本	Params	延遲 (ms/img)	FPS	FLOPs	Acc
Handcrafted (NumPy)	62 000	2.5237	396.25	1.3×10^6	93.2%
PyTorch (CUDA)	61751	0.1283	7791.73	1.3×10^6	94.0%
TF (Dynamic/Eager)	61751	0.1030	9710.64	1.3×10^6	93.8%
TF (Static Graph)	61751	0.0607	16477.20	1.3×10^6	93.8%

A. 結果討論

相較於 GPU 框架，手寫 NumPy 版本即使使用 im2col 進行局部優化，在 CPU 上仍出現約 20–40 倍等級的效能差距，反映 cuDNN 等底層算子在平行計算上的優勢。此外，本模型在 32×32 輸入下的乘加運算（MACs）約為 0.65M，換算 FLOPs 約為 1.3M，屬於計算量相對小但對延遲敏感的網路結構。

IV. TensorFlow 靜態圖與動態圖分析

TensorFlow 的動態圖（Eager）模式具備即時回傳結果、偵錯方便等優點，但 Python 調度開銷可能成為瓶頸；靜態圖（Graph）則透過 `tf.function` 將運算轉換為可編譯的計算圖以降低開銷。實驗觀察顯示，推論延遲由 0.1030 ms 降至 0.0607 ms，約提升 1.7 倍；對應吞吐量提升至 16477 FPS，顯示圖編譯對高吞吐需求情境具有明顯助益。

V. 模型壓縮：剪枝（Pruning）

本實驗另採用權重剪枝將 50% 參數設為零，方法為使用 `tensorflow_model_optimization` 進行 Constant Sparsity 剪枝。評估結果顯示：模型檔案經 Gzip 壓縮後大小可降低約 40%；然而在標準 GPU 稠密算子下，剪枝後推論延遲（約 0.08 ms）反而略高於純靜態圖版本，主因是權重為零仍會被稠密矩陣運算納入計算，且遮罩（mask）引入額外開銷。故剪枝對儲存空間改善顯著，但推論加速需依賴稀疏加速硬體或對稀疏算子有良好支援的執行環境。

VI. 訓練與評估摘要

根據 `lenet_mlp_numpy.py` 的訓練紀錄，模型於 30 個 epoch 內快速收斂：第 10 個 epoch 左右準確率可突破 90%，且訓練／驗證曲線貼合度高，顯示具有良好泛化能力。整體結果上，驗證集 Top-1 準確率約 94.2%，測試集最終準確率約 93.8%。

VII. 結論

本報告完成 LeNet-5 多版本實作與效能對比，結論如下：*(1)* TensorFlow 靜態圖在推論吞吐量表現最佳；*(2)* 動態圖雖利於開發，但在高吞吐需求下應採用靜態圖編譯以降低 runtime 開銷；*(3)* 手寫 NumPy 版本雖慢，但有助於理解底層運算（如 im2col 與梯度推導）；*(4)* 剪枝可有效降低模型儲存空間，但在一般 GPU 稠密算子下不一定帶來推論加速。