Local Binary Pattern Histogram-Based Face Recognition Using Parallel Computing Techniques

陳建霖 (Chien-Lin Chen)
國立陽明交通大學
電機資訊學院在職專班
nn139159@gmail.com

關正爲 (Zheng-Wei Kwan)
國立陽明交通大學
電機資訊學院在職專班
kwanzw1996@gmail.com

Abstract

本研究提出一套輕量化的 LBPH 臉部辨識系統，針對邊緣運算平台進行平行化優化，結合 OpenMP、SIMD（AVX2/NEON）與 CUDA 技術。透過在 x86 與 ARM 架構（如 Raspberry Pi 5、Jetson Nano）上的實驗評估，結果顯示系統在特徵擷取與辨識預測階段皆能顯著加速。尤其在預測階段，Raspberry Pi 5 展現出優異的即時運算效能與良好的擴展性，證明本系統具備高效能且適用於嵌入式環境的臉部辨識解決方案。

1 Introduction

臉部辨識技術在近年來已廣泛應用於安全監控、門禁系統與個人裝置解鎖等場域，成爲智慧應用的重要組件。隨著邊緣運算（Edge Computing）的快速發展，如何在資源有限的嵌入式裝置（如 Raspberry Pi、Jetson Nano）上執行即時且準確的人臉辨識演算法，已成爲一項實務且具挑戰性的課題。

Local Binary Pattern Histogram（LBPH）[6][7] 是一種簡單且有效的特徵擷取方法，其具備結構簡單、計算量低、對光照變化具強健性等特點，特別適合部署於低功耗邊緣設備。LBPH 的核心流程可分爲三大步驟：首先，針對每個像素計算其 LBP 值，即根據其與鄰近像素的灰階差異產生 8-bit 編碼；其次，將整張 LBP 影像分割爲多個區塊並對各區塊統計 LBP 值的直方圖，最終將這些直方圖拼接成一組完整的特徵向量；最後，透過向量間的相似度比對（常用加權卡方距離）完成人臉識別。此外，進階版本如 Circular LBP（ELBP）可透過更靈活的採樣方式與插值計算，捕捉更多紋理細節，但同時也帶來額外的運算負擔。

當 LBPH 應用於即時辨識場景時（如智慧門禁、即時監控、人流分析等），仍面臨多項效能挑戰，特別是在高解析度輸入或頻繁辨識需求下，原始序列版本的 LBPH 容易產生下列效能瓶頸：

- 前處理延遲過高：灰階轉換、臉部偵測與影像歸一化等處理步驟無法即時完成，導致輸入影像與輸出辨識結果間存在明顯延遲。
- 特徵擷取瓶頸：LBP 編碼需針對每個像素鄰域進行運算，對於即時影像串流會造成處理瓶頸，特別在高幀率輸入（>15 FPS）時尤爲明顯。
- 資料比對延遲：LBPH 需將輸入臉部特徵與資料庫中多筆樣本進行直方圖距離計算，在辨識人數增加時處理時間線性增加，導致效能不穩。
- 邊緣裝置效能受限：LBPH 雖運算相對輕量，但在資源有限（如 CPU 僅 2—4 核、記憶體不足 4GB）的邊緣裝置上，仍可能出現記憶體頻寬壅塞與 CPU 核心利用率不佳等情況，導致即時辨識效果不理想。

因此，本研究提出以平行運算技術優化 LBPH 執行效率，並探討其在邊緣場景下的實用性。我們針對三個主要模組——LBP 編碼、Histogram 建立與特徵比對——進行平行化設計，分別實作 CPU 多執行緒版本（採用 OpenMP ）與 GPU 加速版本（使用 CUDA）並在兩種不同架構硬體平台（x86_64、Arm）上進行實驗。透過完整的系統實作與性能分析，本專案旨在評估並量化平行化技術對於 LBPH 在邊緣運算環境下的可行性與實際效益。

實驗採用 AT&T 與 Extended Yale B+ 人臉資料集進行離線測試，分析不同平行化策略在各平台下的效能差異與可行性。透過這些系統化的實驗設計與數據比較，我們期望能驗證 LBPH 在平行化後於實際部署環境中的應用潛力，爲未來輕量級人臉辨識系統提供一項可行的技術選項。

2 Related Work

在影像辨識應用中，如何有效結合平行運算與邊緣裝置資源，已成爲實現即時性能的關鍵挑戰。許多研究已探索 LBPH（Local Binary Patterns Histograms）人臉辨識演算法在多種平台上的加速方式與部署策略，爲本計畫提供了可參考的基礎。

Dwith 與 Rathna (2012）提出了一種基於 OpenCL 的 GPU 平行實作方法，用於加速 LBP 基礎的人臉辨識流程。該研究顯示，在 GPU 上平行化可大幅提升影像處理效能，尤其在特徵計算與直方圖統計等資料密集階段 [3]。

針對邊緣計算應用，VK0101011001001011（2023）展示了如何在 NVIDIA Jetson Nano 上建構完整的人臉辨識系統。此專案整合 OpenCV 與 LBPH 模型，並證明即使在低功耗的嵌入式平台上，亦可達到可接受的準確率與速度 [8]。

Thuy Ng（2018）則釋出了一個開源的平行人臉辨識專案，將演算法以多執行緒方式實作，著重於多核心 CPU 上的比對加速。該專案提供了完整的資料流程設計，對於驗證不同平行策略的效能影響具有參考價值 [5]。

此外，Anjith George（2024）提出 EdgeFace，一種專爲邊緣設備設計的輕量級人臉辨識框架。雖然該系統以深度學習爲核心，但其重視運算與記憶體效率的設計理念，對本計畫在資源受限平台上的部署策略亦具啓發性 [1]。

綜上所述，現有研究已分別從 GPU 加速、多核心平行、與邊緣部署三個面向探索人臉辨識系統的效能提升路徑，爲本計畫結合 LBPH、平行化技術與邊緣部署提供良好依據。

3 Proposed Solution

3.1 實驗方法

本研究旨在設計一套高效能且可移植的臉部辨識系統，並透過系統性的優化策略提升其執行效率。爲達此目標，實驗設計共分爲四大階段，依序說明如下。

3.1.1 建立 **Baseline** 系統

首先，本研究以 OpenCV 所提供之 LBPH (Local Binary Patterns Histogram) 臉部辨識模組作爲參考基礎，實作一套功能等價之純 C++ 版本。透過手動改寫與重構程式碼，全面移除對

OpenCV 函式庫之相依性，並確保與原始演算法在特徵萃取與比對流程上的一致性，作爲後續平行化與加速的起點。

3.1.2 效能瓶頸分析與平行化優化

接著，針對序列版本進行詳細效能剖析，透過時間量測與程式分析工具判定執行熱點模組，包含：

- 擴展型區域二值模式（ELBP）特徵擷取
- 空間直方圖計算
- 特徵相似度比對

本研究針對上述核心模組，導入 OpenMP 多執行緒與 SIMD 指令集最佳化策略。透過細粒度平行化處理像素運算、記憶體存取與向量化計算，有效降低單執行緒版本之瓶頸，並評估不同平行化策略的加速效益。

3.1.3 效能測試與參數調整

爲評估所提方法之加速效果，本研究設計一系列測試實驗，系統性量測不同參數組合對系統效能之影響。包含：

- 不同 Thread 數量（1, 2, 4, 8, 12）
- 是否啓用 SIMD 指令

測試項目涵蓋訓練階段與預測階段之平均執行時間，並將結果視覺化呈現，以探討最佳化策略在不同硬體配置下的效能趨勢與極限。

3.1.4 CUDA 技術進行 GPU 加速

最後，爲進一步發揮 GPU 在大量資料並行運算上的潛力，本研究將 ELBP 特徵擷取與直方圖相似度比對等計算密集模組移植至 NVIDIA GPU 平台，實作 CUDA 版本之核心函式。經由 thread/block 結構設計與記憶體佈局調整，有效利用 GPU 硬體資源。並透過與 CPU 多執行緒版本進行對比分析，驗證 GPU 加速在不同資料集規模下的效能提升幅度。

3.2 ELBP 平行化設計

本章將說明本研究如何針對 Extended Local Binary Pattern（ELBP）演算法進行平行化優化設計，並依序介紹序列版本與平行化版本之流程差異與實作細節，最後輔以示意圖呈現加速邏輯與資料流轉方式。

3.2.1 序列版本實作概述

ELBP 屬於 Local Binary Pattern（LBP）的擴展形式，用以提升對紋理邊界與微變化的表達能力。其核心操作爲：對每個像素點 (i, j)，以固定半徑與鄰點數量環繞中心像素進行取樣，計算內插值與中心像素比較，再將結果轉爲 8 位元 LBP 編碼。

序列版本的實作架構主要包含以下步驟：

- 對每個像素，逐一處理所有鄰點（共 8 個）
- 即時計算鄰點的相對位置與雙線性內插權重
- 逐點進行插值計算與條件判斷（大於或相近於中心值）
- 逐位元累積至 LBP 編碼結果，儲存至輸出圖像

由於缺乏平行化設計與向量處理，整體運算時間與圖像解析度成正比，計算瓶頸主要集中於高密度鄰點 trigonometric 運算與重複插值流程。

3.2.2 平行化策略：SIMD 與 OpenMP 整合

爲克服上述運算瓶頸，本研究將 ELBP 核心邏輯分爲兩層級進行加速：

(1) 資料層級平行（**SIMD**）：採用 AVX2 向量指令，針對每行資料進行 8 pixels 並行處理，涵蓋內插、比較與 LBP 編碼。
(2) 任務層級平行（**OpenMP**）：使用 OpenMP 將圖像 row 分配至不同執行緒，達成整體 row-level 的平行處理。

在開始主迴圈前，所有鄰點的相對偏移座標與內插權重皆預先計算完成，避免於每個像素內重複進行 trigonometric 計算，有效提升記憶體與計算效率。

在平行化版本中，每個執行緒負責處理一組連續圖像列（rows），並透過 AVX2 同步處理每列中連續 8 個像素，計算其對應的鄰點內插值、判斷條件，並生成 LBP 碼。對於無法被 8 整除的像素數尾段，則退回使用原始序列化方式處理，保證完整性與精度一致性。

3.2.3 尾段像素補償處理

由於 SIMD 向量寬度限制（8 pixels per AVX2 register），針對圖像寬度無法整除 8 的情況，本實作保留序列化邏輯作爲 fallback 模式，確保每一個像素點皆可正確處理，無遺漏或錯誤累積。

3.2.4 技術總結

本章提出的 ELBP 平行化最佳化方法，透過雙層平行策略整合（OpenMP + AVX2），有效針對原始序列版本的重複計算與記憶體瓶頸進行改善，整體流程具有以下優勢：

- 透過 AVX2 向量化處理內插與編碼，提升資料吞吐量
- 利用 OpenMP 多執行緒進行 row 分派，充分發揮 CPU 核心效能
- 保留尾段處理與精度等價性，確保結果穩定一致

由於系統亦需部署於 ARM 架構嵌入式平台，本研究同步開發 NEON 版本，盡可能實作與 AVX2 等效之向量化處理流程，確保跨架構效能一致性與可攜性。

3.3 直方圖相似度比較平行化設計

3.3.1 序列版本實作概述

本研究採用基於加權卡方距離（Chi-Square Distance）的直方圖相似度比較函式，序列版本透過對直方圖中每一維度進行逐一掃描計算距離指標。具體而言，對每個維度計算兩組直方圖元素之差值平方，並除以該維度的元素值，以捕捉分布差異。爲避免數值不穩定性，特別針對元素值爲零的情況予以排除，防止除零錯誤。此序列流程結構簡潔明確，易於理解與維護，且在小規模資料集或即時辨識應用中展現良好效能。

此外，序列版本能方便整合結果收集器（collector）設計，支持在達成預設相似度閾值或收集足夠最佳結果後即時中止整體比對流程，降低計算資源浪費，適合對延遲敏感的即時系統。由於缺乏向量化與多核併行支援，其計算效率隨資料量增長線性增加，成爲後續優化的重點。

3.3.2 平行化策略

爲解決序列版本在大規模樣本資料集上的效率瓶頸，本研究實現結合 SIMD 與 OpenMP 多執行緒技術的平行化版本。

- 在資料層級，採用 AVX2 256 位元向量指令集，同時對 8 個直方圖維度進行平行運算，包括差值計算、平方與除法等數學運算。針對元素值爲零所引起的除法不

穩定情況，透過遮罩運算（masking）將該維度結果強制歸零，避免產生 NaN 或無限大數值，確保數值穩定性與計算正確性。

- 在任務層級，利用 OpenMP 進行多核心分派，將樣本集合依據執行緒數目均分，並行計算各樣本與查詢直方圖的距離。此方法能有效利用現代多核心處理器的運算資源，顯著縮短整體比對時間。對於資料長度非向量寬度（8）整除的尾段部分，則退回序列化處理，兼顧運算完整性與精度一致。

此平行策略在維持結果正確性與穩定性的同時，大幅提升處理效能，尤其對樣本量超過數萬筆的大型資料庫具有明顯效益。

3.3.3 技術總結

綜合考量序列與平行版本的設計與實作，本研究提出的雙版本策略具備高度靈活性與擴展性，具體優點包括：

- 序列版本結構簡單，易於實作與維護，且支援即時 early stopping 機制，適合延遲敏感及資源受限的系統場景。
- 平行版本結合 SIMD 與多核心並行技術，能有效利用硬體資源，適用於需完整遍歷資料庫且追求高吞吐量的批次處理任務。
- 兩版本互爲補充，依據應用需求靈活選擇，提升系統整體效能與適用性。
- 平行版本中保留序列化尾段處理，兼顧速度與精度，確保相似度計算結果穩定且準確。

由於系統亦需部署於 ARM 架構嵌入式平台，本研究同步開發 NEON 版本，盡可能實作與 AVX2 等效之向量化處理流程，確保跨架構效能一致性與可攜性。

3.4 CUDA 平行化設計

本節針對本研究所設計之 LBPH 臉部辨識系統中，利用 CUDA 所實作的平行化方式與效能進行分析與說明。系統主要平行化流程包含兩部分：特徵萃取階段（ELBP 及直方圖計算）與查詢比對階段（所有樣本間的 histogram 相似度計算）。

3.4.1 ELBP 特徵萃取

在 ELBP 特徵擷取階段，本研究採用像素層級的資料並行方式，透過每個 CUDA thread 負責一個像素位置 (i, j)，計算其與鄰近像素的關係以產生對應的 LBP 編碼。過程中採用雙線性內插法（bilinear interpolation）以提升精度，並以以下公式決定每個鄰居像素的取樣值：

$$t = w_1 \cdot I_{fy,fx} + w_2 \cdot I_{fy,cx} + w_3 \cdot I_{cy,fx} + w_4 \cdot I_{cy,cx}$$

整體實作爲一高並行之像素層級 kernel，對影像尺寸爲 $H \times W$ 的輸入圖像可佈署 $H \times W$ 個 thread 並行運算。

3.4.2 區塊化直方圖計算

LBP 圖像產生後，透過 CUDA 核心 `histogram_kernel` 執行區塊化直方圖（spatial histogram）計算。每個 CUDA block 對應圖像的一個區塊（grid cell），並於共享記憶體中建立區域 histogram 以減少 global memory 存取負擔。

此 kernel 內部採用以下策略提升效能：

- 使用 __shared__ int local_hist[] 加速直方圖累加。
- 每個 thread 處理區塊中的部分像素，並透過 atomicAdd 累積。
- 計算完成後於 host 端進行正規化，避免 GPU 上的浮點除法誤差累積。

3.4.3 Histogram 相似度比對

在查詢階段中，將查詢圖像經 ELBP + histogram 處理後，與訓練資料的所有 histogram 計算 Chi-square 距離。實作上採用兩種 kernel：

- **Global Memory Kernel**：每個 thread 處理一筆訓練樣本，直接從 global memory 讀取資料。
- **Tiled Kernel**：每個 block 處理一筆訓練樣本，並於共享記憶體中進行加總（reduction），大幅降低記憶體帶寬需求。

此設計能夠同時利用數千個 CUDA thread 並行處理多達上萬張樣本，顯著縮短查詢所需時間。

3.5 技術總結

本研究以 CUDA 針對特徵萃取與相似度比對兩大瓶頸進行平行化設計，有效提升整體運算效能。重點技術如下：

(1) **ELBP** 特徵平行化
透過 CUDA kernel，於像素層級並行計算 ELBP 特徵，使用極座標轉換與雙線性插值提升取樣精度。

(2) 區塊化直方圖計算
每個區塊由一個 CUDA block 處理，使用共享記憶體統計 LBP 值分布，並以 `atomicAdd` 實現 thread 間累積。

(3) **Histogram** 比對加速
使用 two-level 平行策略，將每筆樣本對應一個 block，並以共享記憶體進行 reduction。支援 global 與 tiled kernel，提升比對效率。

(4) 記憶體與資料布局優化
訓練後 histogram 整合上傳至 GPU，查詢時僅需即時上傳單筆資料，降低傳輸成本，並支援 batch 查詢與資料重用。

4 Experimental Methodology

4.1 Dataset

爲了評估本研究所提出之算法平行的效能，我們選用兩個廣泛使用且具代表性的公開臉部資料集進行實驗：AT&T 人臉資料集與 Extended Yale B+ 資料集。這兩個資料集分別反映了在人臉影像中不同層級的挑戰，包括表情、姿態變化與極端光照條件，能有效驗證模型在多樣情境下的辨識能力。

4.1.1 AT&T 人臉資料集 [2]

AT&T 人臉資料集包含來自 40 位受測者的 400 張灰階臉部影像，每位受測者各有 10 張不同影像。影像大小爲 92 × 112 像素，背景簡單，拍攝條件一致，僅有輕微的表情與姿態變化。由於其資料量小、變異性低，AT&T 資料集常用作於基礎人臉辨識方法的效能評估，特別適合用於分析模型在低複雜度、小樣本場景下的表現與運算效率。

4.1.2 Extended Yale B+ 資料集 [4]

Extended Yale B+ 爲 Extended Yale B 人臉資料集的整理版本，專注於高光照變化下的臉部辨識研究。資料集共收錄 28 位受

測者，每人約有 9 種姿勢 × 64 種照明，總數爲 16380 張，影像解析度爲 192 × 168 像素。該資料集由多組不同光源方向拍攝而成，涵蓋從正面光到側面光的極端光照條件。B+ 版本常針對前向至中等偏斜姿態的影像進行裁剪與對齊，以提供較穩定的訓練與測試依據。

在本研究中，Extended Yale B+ 資料集因其影像數量充足，成爲驗證平行化演算法效能的理想選擇。我們根據樣本數進行不同規模的子資料集劃分（如 1000、5000、10000 張影像等），以模擬實際應用中模型面對不同資料量的情境。此設計能夠有效觀察多執行緒（multi-threading）、SIMD、GPU 等平行加速技術，在小型與大型資料集下的效能伸縮性與運算效率表現。

藉由此資料集的分級實驗設計，本研究得以全面分析不同硬體與加速策略在處理大規模臉部影像資料時的表現差異與適應性，對嵌入式應用與高效率人臉辨識系統設計提供實證依據。

4.2 效能測試設計

爲了評估本研究所提出之人臉辨識系統在各種平行化配置與不同硬體平台上的執行效能，我們設計一系列實驗，量測訓練與預測階段的平均執行時間，並觀察其在多執行緒與 SIMD 支援下的效能伸縮性。

4.2.1 訓練／測試分割

實驗中將所有資料集按照 **90%** 爲訓練資料、**10%** 爲測試資料的比例進行隨機劃分，並取其 平均執行時間作爲評估指標，最終報告的時間爲多次實驗的平均值，以消除資料分布波動對結果的影響。

4.2.2 平行化條件設定

本研究針對以下平行化組合進行完整測試，覆蓋嵌入式與桌機應用情境：

- **SIMD**（例如 **AVX2** 或 **NEON**）指令集：啓用與未啓用兩種情形。
- 執行緒數量：{1, 2, 4, 8, 12}

上述每組組合皆個別測試其在特徵擷取（如 ELBP）、直方圖計算、訓練與預測階段之執行時間，並觀察 SIMD 與多執行緒對效能提升之單獨與交互影響。

4.3 硬體平台

爲驗證所提方法於不同平台之可攜性與平行化效能，我們選用五種具有代表性的處理器進行測試，包括三款 x86 架構桌上型與筆電級處理器，以及兩款 ARM 架構嵌入式裝置，說明如下：

Table 1: 硬體平台規格比較

平台	核心數	**SIMD**	架構
Intel i5-10500	6C/12T	AVX2	x86-64
Intel i5-10400F	6C/12T	AVX2	x86-64
Intel i5-1235U	2P+8E/12T	AVX2	x86-64 (Hybrid)
Raspberry Pi 5	4C/4T	NEON	ARMv8-A
Jetson Nano 2GB	4C/4T	NEON	ARMv8-A + GPU

上述平台涵蓋了桌機、筆電與嵌入式等不同計算資源配置，能有效評估本系統在高效能處理器與資源受限設備間的效能差異。各平台皆以其支援的最佳化技術執行，例如 x86 系統可啓用 OpenMP 與 AVX2，ARM 系統則可啓用 NEON SIMD，Jetson Nano 除 CPU 版本外亦執行 GPU 加速版本以利比較。

5 Experimental Results

5.1 ELBP 模組平行化效能分析

爲了評估 SIMD 平行化在 ELBP（Extended Local Binary Pattern）運算中的效能提升，我們針對兩種異質平台進行測試，分別爲使用 NEON 指令集的 ARM 架構與支援 AVX2 指令集的 x86 架構。在不同核心數設定下，我們分別量測啓用與未啓用 SIMD 的 ELBP 平均執行時間。

表 2 列出各平台在不同核心數下的每張影像 ELBP 執行時間。實驗結果顯示，在 ARM 平台上，啓用 NEON SIMD 可將單核心下的平均 ELBP 計算時間由 2.82 ms 降至 0.87 ms，達到約 3.24 倍加速；在 x86 平台上，啓用 AVX2 則可由 2.50 ms 降至 0.56 ms，獲得約 4.46 倍加速。

Table 2: 各平台啓用／未啓用 **SIMD** 時 **ELBP** 單張影像平均執行時間（單位：**ms**）

核心數	ARM (NEON)		x86_64 (AVX2)	
	無 SIMD	有 SIMD	無 SIMD	有 SIMD
1	2.816	0.871	2.532	0.562
2	2.816	0.442	2.609	0.362
4	2.816	0.238	2.605	0.267
8	2.819	0.284	2.608	0.252
12	2.816	0.318	2.604	0.235

觀察無 SIMD 的結果，無論平台或核心數如何，ELBP 的執行時間皆接近固定，說明此模組在無向量化情況下受限於記憶體頻寬或逐像素的序列運算，無法有效透過多核心加速。然而在啓用 SIMD 後，隨核心數增加，ELBP 執行時間有顯著下降，尤其在 2 至 4 核時下降最快；超過 8 核後效能改善趨緩，甚至在部分情況略爲回升，推測可能與 L1/L2 快取競爭或記憶體存取瓶頸有關。

圖 1 呈現 SIMD 模式下 ELBP 的加速比趨勢。以訓練階段來看，ARM 平台最高可達 8.17 倍加速（4 核 SIMD），而 x86 平台則在 12 核 SIMD 模式下達到 5.02 倍加速。預測階段方面，由於需與大量訓練特徵比對，SIMD 帶來顯著效能改善，x86 平台甚至達到 23.37 倍加速。

綜合而言，SIMD 技術在 ELBP 計算中扮演關鍵角色，無論是在嵌入式的 ARM 平台，或是具備高效向量化資源的 x86 平台，皆能顯著提升整體訓練與預測效率。其中 AVX2 架構由於具備更寬的 SIMD 暫存器（256-bit 相較於 NEON 的 128-bit）以及較佳的多核心記憶體架構，在大規模資料比對中展現出更優異的預測性能。

5.2 訓練與預測效能分析

本節針對 LBPH 演算法在不同資料集規模下，使用 OpenMP 與 SIMD 指令集所帶來的加速效果進行探討。測試項目分爲兩階段：「訓練」與「預測」，並觀察多執行緒數量（Threads = 1, 2, 4, 8, 12）以及是否啓用 SIMD 的組合變化。

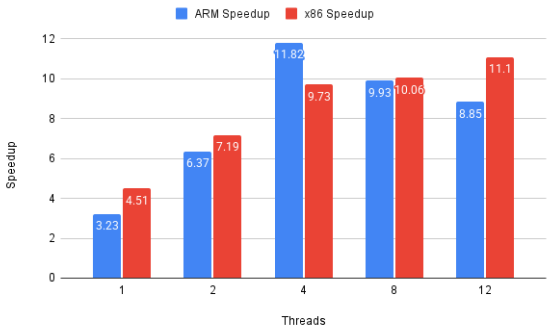

Figure 1: 不同核心數下 **SIMD** 模式之 **ELBP** 加速比

5.2.1 訓練階段

如圖 2所示，訓練階段的加速比受資料集大小與執行緒數影響顯著。在未啓用 SIMD 的情況下，當 Thread 數量增加時，整體訓練效能穩定提升。然而，在樣本數較小（例如 400 筆）的情況下，8 至 12 執行緒的效能提升趨於飽和。

例如，當樣本數爲 400 時，Thread 從 1 增加至 8 時，訓練加速比由 0.95 提升至 4.90，惟進一步提升至 12 執行緒後，加速比僅微幅增加至 4.75，顯示出 CPU 核心使用效率已趨於飽和。

在啓用 SIMD 後，訓練加速效果更爲明顯。例如在樣本數爲 1000 時，使用 12 執行緒且啓用 SIMD 的加速比爲 23.04 倍；而在樣本數爲 16380 時，加速比雖僅爲 3.11 倍，惟仍優於未啓用 SIMD 的 3.50 倍。

整體來說，訓練階段在小型資料集下多核心效果明顯，但大型資料集下可能受限於記憶體頻寬與載入時間，導致多執行緒與 SIMD 效益趨於平緩。

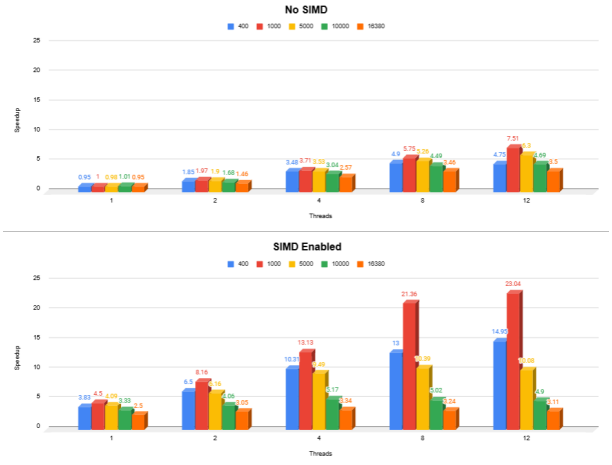

Figure 2: 不同 **Thread** 與 **SIMD** 配置下之訓練階段加速比

5.2.2 預測階段

如圖 3所示，預測階段的效能提升在使用 SIMD 時表現尤爲突出。在所有資料集中，啓用 SIMD 可明顯降低預測所需時間，尤其對於大樣本資料集，其加速效果幾乎呈線性提升。

以 5000 筆資料爲例，在 12 執行緒且啓用 SIMD 的情況下，預測時間從序列版本的 108356 毫秒下降至 4703 毫秒，加速比達 23.04 倍。類似地，在樣本數爲 10000 及 16380 時，最高預測加速比分別爲 22.64 倍與 22.97 倍。

即使在小型資料集（如 400 筆）中，啓用 SIMD 也能使預測速度從 600 毫秒大幅下降至 31 毫秒，加速比達 19.35 倍，顯示 SIMD 對於預測的貢獻不僅限於大資料集，亦適用於即時性要求高的小樣本場景。

此外，相較於訓練階段，預測階段的運算更偏向資料並行處理，故 SIMD 的優勢在此階段發揮得更加明顯。

綜合而言，預測階段的效能最佳化應以 SIMD 優化爲核心，並配合適當的 Thread 數進行平行化，以達到最佳加速效果。

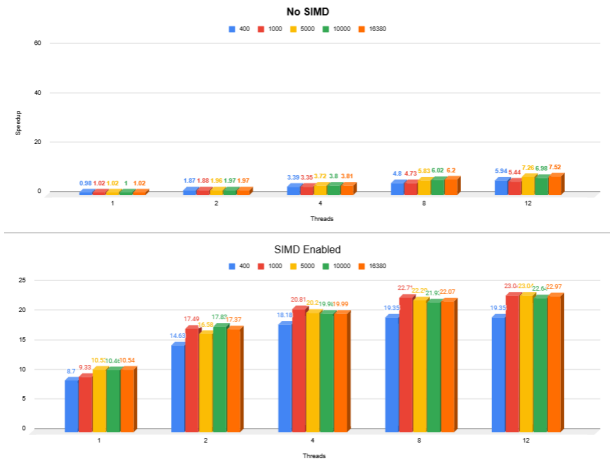

Figure 3: 不同 **Thread** 與 **SIMD** 配置下之預測階段加速比

5.2.3 資料集規模影響

資料集規模對加速比的影響亦不可忽視。在小型資料集中（如 400 或 1000 筆樣本），多執行緒與 SIMD 能夠顯著縮短運算時間，但因計算量有限，其加速效果易受其他系統因素（如記憶體存取延遲、排程開銷）影響。

相較之下，隨著資料集擴增至中大型規模（如 5000、10000 甚至 16380 筆），加速比呈現更穩定的成長趨勢。此時，並行化與向量化可更有效地攤平高昂的運算成本，使得訓練與預測階段的總運行時間皆獲得大幅改善。

Table 3: 不同資料集規模下的訓練與預測加速比（**12** 執行緒）

樣本數	訓練		預測	
	無 SIMD	有 SIMD	無 SIMD	有 SIMD
400	4.75	14.95	5.94	19.35
1000	7.51	23.04	5.44	24.04
5000	6.30	10.08	7.26	23.04
10000	4.69	4.90	6.98	22.64
16380	3.50	3.11	7.52	22.97

如表 3 所示，資料集規模對加速比有顯著影響。啓用 SIMD 後，在所有資料集中皆能維持穩定的預測加速效果，顯示本系統在平行處理與可擴展性上具備高度優勢。

5.2.4 整體分析

從實驗結果可歸納如下：

- 預測階段爲最耗時之瓶頸，啓用 SIMD 與 OpenMP 爲提升實時效能之必要條件。
- 訓練階段加速效果與資料集大小成反比，隨著樣本增加，Thread 數增益趨緩。
- 啓用多執行緒在無 SIMD 時能提供有限加速，需搭配 SIMD 才可發揮完整潛能。

在嵌入式或邊緣裝置應用中，若預測速度爲首要需求，建議優先實作 SIMD 加速版本，並根據平台特性選擇適當 Thread 數以達最佳平衡。

5.3 即時預測效能分析

5.3.1 桌機與筆電 CPU 平台

本節分析三款桌機與筆電 CPU 在 LBPH 預測階段的效能表現，並以單張預測時間換算成 FPS 進行即時處理能力評估。

Table 4: 桌機與筆電 **CPU** 平台 **LBPH** 預測效能與 **FPS** 比較

CPU	訓練資料	測試數	平均預測 (ms)	預測 FPS
i5-10500	361	39	0.51	1957
	901	99	1.49	668
	4501	499	5.28	189
	9001	999	9.89	101
	14743	1637	15.32	65
i5-10400F	361	39	0.77	1300
	901	99	1.84	541
	4501	499	10.38	96
	9001	999	18.28	54
	14743	1637	30.16	33
i5-1235U	361	39	1.28	781
	901	99	2.50	399
	4501	499	12.30	81
	9001	999	23.69	42
	14743	1637	34.78	29

分析說明：

- 單張預測時間最低爲 **i5-10500**，在小樣本（400 張）下平均僅需約 0.51 ms，換算可達 約 **1957 FPS**，遠超即時需求（30 FPS）。
- 隨著資料集規模增大，預測時間與準確率同步提升，i5-10500 在最大樣本 16380 張時仍維持約 65 FPS，即時處理能力佳。
- i5-10400F 與 i5-1235U 效能較弱，尤其在大樣本（>5000 張）時，平均預測時間超過 10 ms，FPS 分別降至約 96 與 81 以下，仍可滿足一般即時辨識需求，但較 i5-10500 有明顯差距。
- 筆電處理器 i5-1235U 雖有較低功耗優勢，但處理效能在大規模資料上受限，約 29 FPS，若應用需高解析度、高速回應，建議選擇桌機平台。

結論：桌機 CPU，特別是 i5-10500，具備強大預測速度與準確率，可支持高頻率即時人臉辨識應用。筆電 CPU 可作爲輕量即時辨識方案，但在大資料量情況下可能受限。使用者可依即時性需求與硬體資源做出平衡選擇。

5.3.2 嵌入式設備

本節分析 Raspberry Pi 5 與 Jetson Nano（OpenMP 與 CUDA 模式）在 LBPH 預測階段的效能表現，換算單張預測平均耗時及即時影像處理能力。以下說明三者於推論效能、延遲表現與即時應用適配性方面之分析與結論。

Table 5: 嵌入式設備 **LBPH** 預測效能與 **FPS** 比較

平台	訓練資料	測試數	平均預測 (ms)	預測 FPS
Raspberry Pi 5	361	39	2.44	410
	901	99	6.02	166
	4501	499	27.83	36
Jetson Nano OpenMP	361	39	6.13	163
	901	99	15.90	63
	4501	499	72.90	14
Jetson Nano CUDA	361	39	11.28	89
	901	99	19.82	50
	4501	499	55.70	18

分析說明：

- 實驗顯示，隨著資料庫規模擴增，各平台預測速度均顯著下降。然而，Raspberry Pi 5 在所有資料集大小中皆維持領先效能。以 FPS 爲例，於 361 筆資料庫下，RPi5 可達 409.84，顯著優於 Jetson Nano 之 OpenMP（163.13）與 CUDA（88.65）版本；在 4501 筆資料庫下，RPi5 亦能維持 35.95 FPS，即便下降 11 倍，仍高於 Jetson Nano OpenMP（13.72）與 CUDA（17.95）。此結果顯示 RPi5 即便無 GPU 加速，仍具備高度可擴展性與穩定表現。
- 針對 Jetson Nano 內部之 OpenMP 與 CUDA 版本進行交叉比較，在資料庫較小（361、901）時，OpenMP 版本表現優於 CUDA，推測係因 CUDA 初期 kernel 啓動與資料搬移開銷較高；然而，當資料庫擴增至 4501 張時，CUDA 版本開始超越 OpenMP 版本，顯示 GPU 加速於大型資料規模下展現效能優勢。但即便如此，其最大 FPS 仍低於 RPi5，顯示 Jetson Nano 硬體架構受限於記憶體頻寬與 GPU 資源，導致整體效能未能完全發揮。
- 以單張預測延遲爲依據，RPi5 在 361 筆資料庫下僅需 2.44 毫秒，遠優於 Jetson Nano OpenMP（6.13ms）與 CUDA（11.28ms），即時處理能力明顯。即使於 4501 資料庫情境下，RPi5 仍可維持每張 27.8 毫秒推論時間，接近 30FPS 即時需求基準線，而 Jetson Nano 僅能提供 72.91 毫秒（OMP）與 55.74 毫秒（CUDA）之延遲，無法支援即時應用場景。

結論：綜合效能與即時性分析結果，Raspberry Pi 5(RPi5) 爲目前嵌入式即時人臉辨識最佳選擇。在小型至中型資料集下，RPi5 具備極高 FPS 與低延遲；而即便面對大型資料集，其效能下降亦較緩和，可有效維持應用穩定性。相較之下，Jetson Nano 僅在啓用 CUDA、且資料量極大時能顯現優勢，但整體仍不敵 RPi5。未來若針對 DNN-based 方法進行比較，Jetson 平台可望在 GPU 支援下展現潛力；惟對於傳統 LBPH 任務而言，RPi5 仍爲最佳性價比解決方案。

6 Conclusions

本研究針對即時人臉辨識任務，實作可適用於嵌入式與桌機平台之輕量化 LBPH 演算法，並結合多種加速技術（OpenMP、多執行緒、SIMD 與 CUDA），於多種硬體平台進行全面實驗，分析不同架構在推論階段之即時效能與可擴展性。

6.1 LBPH 預測加速結論

實驗顯示，預測階段爲 LBPH 辨識流程中最主要的運算瓶頸，遠大於訓練階段。透過多執行緒與 SIMD 指令集最佳化，可大幅降低預測延遲，發揮現代處理器核心數與向量運算能力，並有效應對大規模資料庫所帶來的線性運算壓力。資料庫規模愈大，平行化效益愈顯著，具備良好可擴展性。

6.2 桌上型處理器效能比較

針對 Intel 三款處理器（i5-10500、i5-10400F、i5-1235U）進行 LBPH 預測階段效能比較後，得出以下結論：

- **i5-10500**：最適合即時臉部辨識任務的桌機處理器。預測效能最佳，於樣本數超過 14,000 張時仍可維持超過 60 FPS，單張預測時間可低於 1 毫秒，適用於高頻即時串流與邊緣推論任務（如智慧門禁、影像監控）。
- **i5-10400F**：中階效能方案，適用中頻率應用。整體效能約爲 i5-10500 之 68%，在樣本數低於 9,000 時仍可維持即時處理（30 FPS 以上），適合中頻門禁系統與一般監控應用。
- **i5-1235U**：不建議用於高頻即時辨識場景。雖採混合核心架構（P-core + E-core），但 OpenMP 無法有效利用 E-core，導致效能明顯落後。樣本數超過 8,000 即跌出即時處理門檻，僅建議用於低頻辨識或節能導向場景。

6.3 嵌入式平台比較結論

在嵌入式平台方面，實驗顯示：

- **Raspberry Pi 5** 爲目前最佳嵌入式即時人臉辨識平台，於 OpenMP+NEON 下展現極高 FPS 與低延遲，尤其在小至中型資料集下（<5,000 樣本）可穩定達成即時預測目標。
- **Jetson Nano** 僅於大型資料集與 **GPU** 加速下具備競爭力，但在傳統 LBPH 流程中，OpenMP 效能遠不如 Pi 5，而 CUDA 版本則需透過資料流與記憶體管理優化以克服小資料集下的效能瓶頸。

6.4 本研究貢獻

- 提出一套可依執行環境動態選擇加速方式之輕量化 **LBPH** 人臉辨識演算法，支援序列、OpenMP、SIMD 與 CUDA 等運算後端，適用於從桌機到嵌入式之多樣平台。
- 實驗結果顯示，Raspberry Pi 5 在傳統特徵法下效能/功耗比最佳，Jetson Nano 僅於 DNN 類任務中具備 GPU 加速優勢。
- 系統於多資料集、多平台測試下展現良好可擴展性，並針對桌機與嵌入式處理器分別給出平台選型建議。

6.5 未來工作

針對未來系統發展，將延伸至以下幾個方向：

- 優化 CUDA 版本之資料流與記憶體策略，透過共享記憶體重構、批次處理與搬移重疊等技術，提升 GPU 利用率，改善小資料集延遲表現。
- 研究近似最近鄰搜尋（如 **LSH**）與特徵降維（如 **PCA**）技術於 LBPH 之應用，進一步降低大型資料庫比對成本。
- 擴展系統至更低階邊緣設備，探索於極低功耗與有限記憶體下的演算法簡化與即時應用潛力，推進人臉辨識於極端嵌入式環境之可行性。

References

[1] Hatef Otroshi Shahreza Ketan Kotwal Sebastien Marcel Anjith George, Christophe Ecabert. 2024. EdgeFace: Efficient Face Recognition Model for Edge Devices. *IEEE Transactions on Biometrics Behavior and Identity Science* (2024). doi:10.1109/TBIOM.2024.3352164

[2] AT&T Laboratories Cambridge. 1994. The AT&T Database of Faces (formerly "The ORL Database of Faces"). http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html. Accessed: 2025-06-20.

[3] C.Y.N Dwith and G.N.Rathna. 2012. Parallel implementation of LBP based face recognition on GPU using OpenCL. *Parallel and Distributed Computing, Applications and Technologies (PDCAT)* (2012). doi:10.1109/PDCAT.2012.107

[4] Athinodoros S. Georghiades, Peter N. Belhumeur, and David J. Kriegman. 2004. Extended Yale Face Database B. http://vision.ucsd.edu/~leekc/ExtYaleDatabase/ExtYaleB.html. Accessed: 2025-06-20.

[5] Thuy Ng. 2018. *Parallel-Face-Recognition.* Retrieved Mep 1, 2018 from https://github.com/thuyngch/Parallel-Face-Recognition

[6] Adrian Rosebrock. 2021. *ace recognition with Local Binary Patterns (LBPs) and OpenCV.* Retrieved May 3, 2021 from https://pyimagesearch.com/2021/05/03/face-recognition-with-local-binary-patterns-lbps-and-opencv/

[7] Parth Singh. 2024. *Understanding Face Recognition using LBPH Algorithm.* Retrieved Oct 21, 2024 from https://www.analyticsvidhya.com/blog/2021/07/understanding-face-recognition-using-lbph-algorithm/

[8] VK0101011001001011. 2024. *Build-a-Face-Recognition-Machine-With-Jetson-Nano.* Retrieved May 4, 2024 from https://github.com/VK0101011001001011/Build-a-Face-Recognition-Machine-With-Jetson-Nano