

國立臺灣科技大學
工業管理系
113 學年度實務專題報告

專題名稱：基於多模態影像及人體動作辨識技術優化
產線流程分析

專題編號：**TR-112-01-112**

指導教授：楊朝龍教授

研究組員：**B11001110 林允暘**

B11001114 張家碩

B11001115 溫梓均

B11001117 洪書諭

B11001129 李育宸

中華民國 113 年 5 月 24 日

目錄

摘要.....	1
致謝.....	2
壹、研究動機與目標.....	3
一、研究動機.....	3
二、研究問題.....	3
三、研究目標.....	4
貳、文獻回顧.....	5
一、深度學習.....	5
二、人體動作辨識.....	10
三、物件偵測/辨識（Object Detection）.....	13
四、YOLO v7	18
參、研究方法.....	21
一、研究流程.....	21
二、數據採集.....	21
三、影像處理及操作流程.....	22
四、研究工具.....	25
五、模型設計.....	25
肆、研究討論與結果.....	46
一、研究討論.....	46
二、研究改善.....	48
三、研究結果.....	50
四、實務方面的應用.....	53
伍、結論與未來展望.....	54
陸、參考資料.....	56
附錄.....	61

圖目錄

圖 1 深度學習所使用的類神經網路.....	5
圖 2 Two-Stream 2D CNN.....	6
圖 3 Long Short-Term Memory.....	7
圖 4 3D CNN	7
圖 5 Transformer.....	7
圖 6 25 個身體關節特徵點及人體 RGB 影像與提取關節特徵展示[21].....	8
圖 7 NTU RGB+D 示例	8
圖 8 深度學習基於骨架特徵提取行為辨識流程[20].....	8
圖 9 顯示了多模態特徵提取方法的運作過程.....	10
圖 10 ST-GCN 特徵點	11
圖 11 ST-GCN 模型	11
圖 12 PoseC3D 架構	12
圖 13 SlowFast Networks	13
圖 14 R-CNN	14
圖 15 Fast R-CNN	15
圖 16 Faster R-CNN	16
圖 17 SSD 的概述圖	16
圖 18 RetinaNet	17
圖 19 Two-Stage 偵測器的基本架構	18
圖 20 One-Stage 偵測器的基本架構.....	18
圖 21 YOLO v7.....	19
圖 22 研究流程圖.....	21
圖 23 Microsoft 攝影機架設示意圖	22
圖 24 貼標程式示意圖	23
圖 25 第三站之 SOP 動作代號、組裝動作說明與示意圖	23
圖 26 工工作站操作流程圖.....	24
圖 27 程式架構流程圖	26
圖 28 設定初始參數.....	27
圖 29 建立相應的目錄結構和設置相關的路徑.....	28
圖 30 參數設定.....	28
圖 31 將讀取的檔案轉為標記檔 labels-action.csv.....	29
圖 32 (左)原讀取的 label 檔(右)labels-action.csv.....	29
圖 33 將標籤存入 classes.txt	30
圖 34 classes.txt	30
圖 35 將 labels-action.csv 轉換為 labelsgenerate.csv.....	31
圖 36 labels generate.csv	32

圖 37 將資料儲存為影片解碼檔.....	32
圖 38 labels-train.csv	33
圖 39 將影片解碼檔轉換為訓練資料集格式 labels-train.csv	33
圖 40 data_prepare 程式輸出	33
圖 41 參數設定.....	34
圖 42 data_prepare 程式輸出	34
圖 43 Filter 參數設定	35
圖 44 data_prepare 程式輸出	35
圖 45 動作辨識程式碼一.....	36
圖 46 動作辨識程式碼二.....	36
圖 47 動作辨識程式碼三.....	37
圖 48 預測結果.....	37
圖 49 預測影像.....	38
圖 50 模型評估主程式碼.....	38
圖 51 模型評估 Matrics 程式碼一	39
圖 52 模型評估 Matrics 程式碼二	39
圖 53 模型評估 Matrics 程式碼三	40
圖 54 精確率與召回率的混淆矩陣.....	40
圖 55 讀取檔案程式碼.....	41
圖 56 處理讀進來的檔案.....	41
圖 57 計算出一幀佔幾秒.....	41
圖 58 計算動作出現的時間.....	42
圖 59 計算組裝時間週期.....	42
圖 60 全程導向循環週期計算法.....	43
圖 61 計算平均組裝動作循環週期.....	44
圖 62 輸出結果到 csv 檔	44
圖 63 train data 為 200 時的精確率與召回率.....	46
圖 64 train data 為 400 時的精確率與召回率.....	47
圖 65 生成的流程程序圖.....	47
圖 66 11 種標籤下的精確率與召回率.....	48
圖 67 4 種標籤下的精確率與召回率.....	49
圖 68 受動作辨識訊雜訊影響的流程程序圖.....	49
圖 69 重複動作相連的流程程序圖.....	50
圖 70 操作員之一次組裝的流程程序圖的結果.....	50
圖 71 組裝動作循環週期對照.....	51
圖 72 有效動作時間對照.....	52
圖 73 無效動作時間對照.....	52
圖 74 組裝動作循環週期.....	53

圖 75 有效動作時間對照.....	53
圖 76 處理偏差時間.....	61
圖 77 處理連續的相同動作時間.....	61

表目錄

表 1 Two-Stage 偵測 與 One-Stage 偵測的優缺點.....	17
表 2 為終點導向循環週期計算法及全程導向循環週期計算法的結果.....	45
表 3 為各項數據在三種貼標方法經過兩種計算組裝動作循環週期得到的結果	51

摘要

隨著工業 4.0 時代的來臨，自動化和智慧製造成為製造業的新趨勢，面對多樣化產線需求，傳統的自動化設備在靈活性和效率上遇到挑戰。智慧化設備可以運用在多樣化生產線上，以提高生產效率及靈活性，並且能夠藉由這些設備減少所需的人力，進而降低人力成本。而流程程序圖對於生產規劃和過程優化至關重要，傳統的流程程序圖需要耗費工業工程師大量時間到生產線上實際測量再進行繪製，繪製完成後，工業工程師再開始進行分析及優化。

本研究旨在探討一種結合物件辨識和影像辨識的混合技術，用於自動生成流程程序圖，比較不同的貼標方式之於辨識結果的精確度，並解決可能會影響辨識結果的特殊狀況問題。本研究成果將提高生產效率與靈活性，降低繪製流程程序圖的時間及人力成本，並加快生產流程的優化。隨著工業自動化技術的不斷進步，我們預期這一技術將為製造業帶來革新和價值，尤其在提高生產效率和工作場所安全性方面具有顯著潛力。

關鍵字：工業 4.0、動作辨識、混合影像辨識、深度學習、流程優化、YOLO v7 、SlowFast

致謝

本研究之所以可以順利完成，首先要感謝此專題的指導老師楊朝龍教授。透過不斷的討論，使我們一步步構思出研究方向、內容修改等，給予我們許多不同面向的學術建議，並提供我們許多寶貴的意見與學習資源，使我們在專題研究中受益良多。於此同時，也常鼓勵我們多去思考、了解最新的科技應用，我們才能集思廣益，提出許多新穎的方向，並完成專題。

其次，感謝實驗室的學長姐們熱心協助，在百忙之中適當給予我們建議，並在我們遇到設備、關鍵程式碼運行等難題時，給予我們幫助，使我們能順利克服設備上的難題，完成訓練，並從中習得相關知識與解決方法，在此深感謝意。我們也非常感謝產學合作公司提供我們機會實際到工廠內進行資料收集，讓我們可以有足夠的資料作為訓練以及測試，在此深感謝意。

最後要感謝的是參與本研究的組員：林允暘、張家碩、溫梓均、洪書諭、李育宸同學們利用空堂及課後時間參與討論，彼此切磋討論、相互勉勵扶持，並且不斷的重複檢討，使本研究可以順利完成。

壹、研究動機與目標

一、研究動機

當今工業不斷精進，追求高效生產，透過各式工作研究手法衡量生產效率找尋生產瓶頸。眾多研究指出流程程序圖(Flow Process Chart)能夠提高工作效率、降低風險、增進團隊協作，已是精實生產中不可或缺的工具之一。透過量化工作流程間之工作時間，使複雜的流程容易判讀，更能作為協調工作的依據；流程程序圖讓團隊成員之間更容易理解工作步驟，進而有效地合作，而標準化工作流程，不僅能夠減少錯誤和混亂，提高效率，更有助於發現流程中的瓶頸和問題，從而針對性地改進和優化流程。

然而，工作研究手法卻高度依賴人力操作，且受限於人力效率，量測個體差異等問題。流程程序圖中之時間分析，需要工業工程師透過碼表計時法量測各流程之作業時間，不僅需耗費大量人力資源在量測上，且不同人員量測間會產生量測差異，若能透過影像辨識技術自動量測生成流程程序圖，不僅易於量測及監控產線，更使得在流程或程序變動時能夠輕鬆地進行應對與調教，進而縮短優化產線所需的作業時間。

在實務上，往往受限於工廠內工具繁多，工具外型一致導致辨識困難，如：十字及一字起子、被器材遮擋無法清晰拍攝操作流程影像、人員突發狀況如離開崗位等問題，需要透過多種影像辨識技術以確保影像分析之正確性。因此，基於優化工作研究方法的目標下，本研究提出核心研究目標為「**基於多模態影像及人體動作辨識技術優化產線流程分析**」，將分為兩個子研究問題，以技術及實務上進行評估。

二、研究問題

(一) **自動化時間研究**

過去，當產線遭遇瓶頸，需要優化產線效率時，會使用到流程程序圖檢視生產流程上的工作時間及閒置時間，加以改善優化生產流程，以達到更好的生產效率，而傳統流程程序圖需要以人工操作碼表的方式實時計算操作員的工時，需要耗費許多心力及時間，並且需要雇用許多人力。本研究計畫將探討透過影像辨識技術代替人力，計算生成流程程序圖，加速實務上優化生產線所需要的時間。

(二) 產線突發狀況

生產線上操作員有時會有非標準動作流程的操作程序，例如：拿取零件及工具的順序錯誤、操作員中途離開崗位、零件未正確組裝或拆卸等問題，這類問題會造成本研究的資料集在訓練模型時發生誤差的狀況，進而影響訓練的模型訓練及一致性，更影響最終的計算結果。本計畫將對資料集中的突發狀況進行修正並加以探討與校正模型，將模型正規化，以減少所產生的誤差，並分析不同處理方法對訓練結果的影響。

三、研究目標

本研究以基於多模態影像辨識技術生成流程程序圖以優化產線分析為目標，進行實際的模擬與模型訓練，並解決實務上所面臨之困難，使動作辨識可以降低誤差，生成更準確之流程程序圖，並增加實務上之可行性。此研究目標將分為兩個部分：

- (一) 以動作辨識及物件辨識生成流程程序圖
- (二) 解決實務所面臨之問題，提高準確度

透過實際的工廠參訪，模擬產線流程，並透過架設感測器，進行資料的蒐集，導入 SlowFast 及 YOLO v7 等多模態動作辨識技術進行模型訓練，區分出每個操作程序的節點，以生成出一張流程程序圖。後續找出三個問題的在動作辨識上的阻礙，並加以研究解決，以優化流程程序圖。期許能透過多模態影像辨識技術，達到高度準確的流程程序圖，以減少工業工程師的前置作業及優化生產線的工作時間。

貳、文獻回顧

一、深度學習

深度學習是基於人工神經網路的機器學習技術，採用多層次的人工神經網路(Artificial Neural Network, ANN)模擬生物神經系統，經由不斷調整內部結構以及參數設定，訓練和分析資料特徵，將資料自動辨識與分類，相當於直接從資料學習建立模型。近來已廣泛應用及研究，可識別影像[1, 2]、文字[3]、聲音和其他資料的複雜模式，並具備學習能力的自適應系統，具有顯著的性能[4]。

深度學習所使用的類神經網路，是由「輸入層」(Input Layer)、「隱藏層」(Hidden Layer) 及「輸出層」(Output Layer)三層所構成，其架構如圖1。透過類神經網路，由輸入層將資料特徵(Features)輸入，透過隱藏層迭代調整特徵權重(Weights)，最佳化輸出層，最終訓練出一組權重，以完成深度學習訓練。深度學習廣泛運用在不同類型之資料，常見的類神經網路有卷積神經網路(Convolutional Neural Network, CNN)，適用處理圖像數據，通過模仿人類視覺系統的工作原理來識別和分類圖像中的物體。循環神經網路(Recurrent Neural Network, RNN)，適用於處理序列數據，如文本或時間序列數據，能夠捕捉時間上的動態變化。

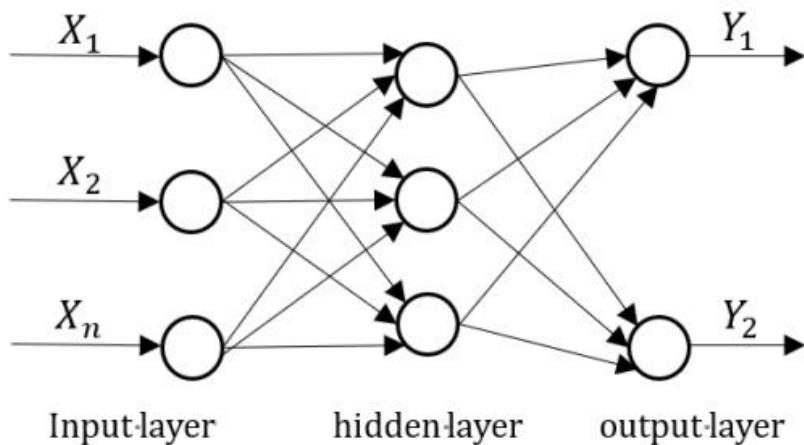


圖 1 深度學習所使用的類神經網路

本研究將專注於深度學習於影像辨識技術上之應用，基於不同演算法基礎，如基於 RGB 像素特徵、基於骨架特徵和基於多模態等眾多高效模型，探討眾多演算法基礎之特性。

(一) 基於 RGB 像素特徵提取

影像深度學習中基於 RGB 像素特徵提取，是指透過圖像或影像(圖像序列)剖析像數點特徵，藉由每一個像素點的紅色 (R)、綠色 (G)、藍色 (B) 三原色的強度值捕捉影像信息，完成影像辨識技術[5]。基於 RGB 像素特徵提取的人體動作辨識已具有廣泛的研究與應用。然而，RGB 像素受影像品質、背景和照明敏感等像素條件影響較大，因此在進行動作辨識時可能面臨一些挑戰[6]。

深度學習於影像辨識領域中最著名的演算法基礎為卷積神經網路 (Convolutional Neural Networks, CNN)，即為基於 RGB 像素特徵提取構建深度學習。隨著深度學習技術發展，基於 CNN 邏輯，結合不同特徵處理方法，如時間序列因子[7]、時空內的關鍵點(On Space-Time Interest Points, STIP)[8]方法等，更提出各種深度學習架構。著名基於 RGB 像素特徵提取所衍生的影像辨識模型有 Two-Stream 2D CNN[9]、Long Short-Term Memory [10]、3D CNN[11]、Transformer[12]等方法。

1. Two-Stream 2D CNN：採用兩個 2D CNN 分支，分別處理 RGB 影像的不同輸入特徵，再通過融合的方式得到最終結果。Two-Stream 包括了空間流(處理 RGB 幀)和時間流(處理光流)，用於學習外觀和運動特徵進行人體動作辨識，模模架構如圖 2 Two-Stream 2D CNN 所示。

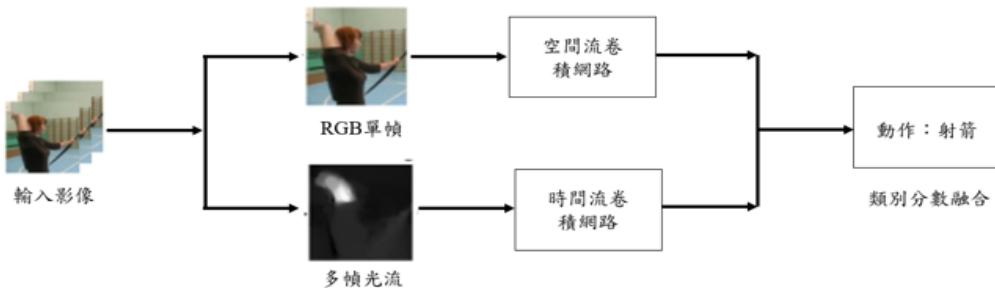


圖 2 Two-Stream 2D CNN

2. Long Short-Term Memory：為 RNN 的一種，用於克服傳統 RNN 中的梯度消失問題[13]。LSTM 和 GRU (Gated Recurrent Unit 門控循環單元) 被用來有效地建模影片序列的長期時序動態[14]。在影像動作辨識中，通常先用 2D CNN 提取每一幀影像的空間特徵，並將特徵輸入到 LSTM 中，藉由 LSTM 在循環連階層中捕捉時間的關係，進行動作辨識。這樣的模型結構可以更好地處理影片序列中的長期時序動態。架構如圖 3 Long Short-Term Memory 所示。

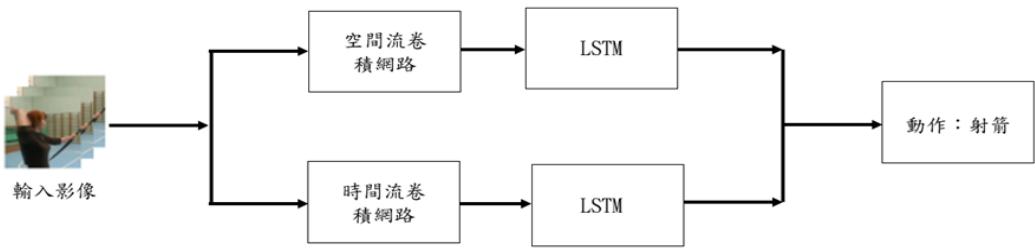


圖 3 Long Short-Term Memory

3. 3D CNN：在傳統的 2D CNN 基礎上增加了一個時間維度，將其拓展到 3D 結構[11, 15-17]，以有效捕捉影像中的時空特徵。這使得 3D CNN 能夠處理包含時間序列信息的影像數據，如影像分析和長時間範圍內的動作識別，架構如圖 4 3D CNN 所示。

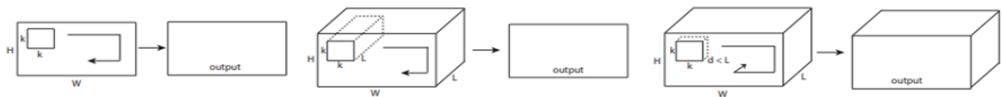


圖 4 3D CNN

4. Transformer：以自注意力機制為基礎，直接處理影像序列，學習影像中的時空關係[18]。具有良好的建模能力，能夠處理長時間序列、多模態融合和多任務處理。Transformer 模型由編碼器和解碼器組成，在影像分析中主要使用編碼器部分。編碼器是由多個自注意力塊和前向神經網路塊組成，可直接處理整個影像序列，從而有效捕捉時空特徵，架構如圖 5 所示。

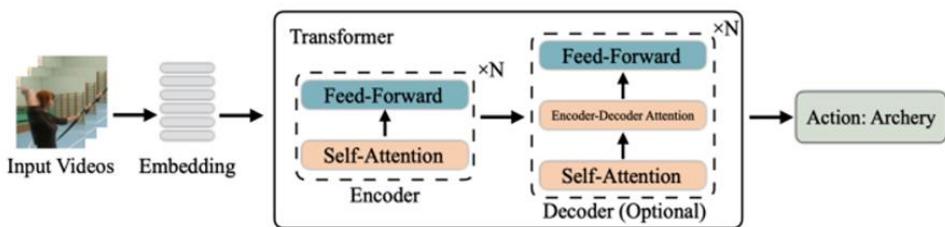


圖 5 Transformer

(二) 基於骨架特徵提取

基於骨架特徵提取為深度學習應用與人體動作辨識中一項基礎演算法，從影像所捕捉到的人體骨架數據中提取骨架特徵點[19]，以實現對特定人體動作的識別和理解。在實現人體動作辨識（Human Action Recognition，HAR）的過

程中，骨架特徵能夠提供對關節位置、動作軌跡和姿勢的精確信息，這有助於更有效地進行動作識別，透過分析骨架特徵的變化與運動軌跡，我們能夠識別並分類不同的動作類別。相較於 RGB 像素特徵提取，骨架特徵提取基礎因影像前處理，透過主成分分析（Principal Component Analysis, PCA）提取重要特徵，提供了更為輕量且不受背景影響的特徵，使得基於骨架特徵提取演算法在深度學習動作辨識中更具優勢[20]，圖 7 展示了人體骨架資料的視覺效果。

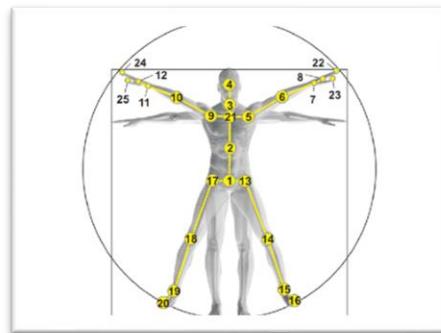


圖 6 25 個身體關節特徵點及人體 RGB 影像與提取關節特徵展示[21]



圖 7 NTU RGB+D 示例

骨架特徵於深度學習之模型，藉由從影像所捕捉到的人體骨架數據中提取骨架特徵點，並使用 RNN[22]、CNN[23]和 GCN[24]等深度學習模型，提高了對骨架特徵的擷取和辨識效能。這些方法在處理時序問題時表現卓越，對於捕捉動作的時空信息具有出色的能力。圖 8 為深度學習基於骨架特徵提取行為辨識流程，首先基於 RGB 影像提取骨架特徵，從深度感測器或姿態估計演算法取得骨架資料，並將骨架資料輸入到神經網路，最後得到行為類別。

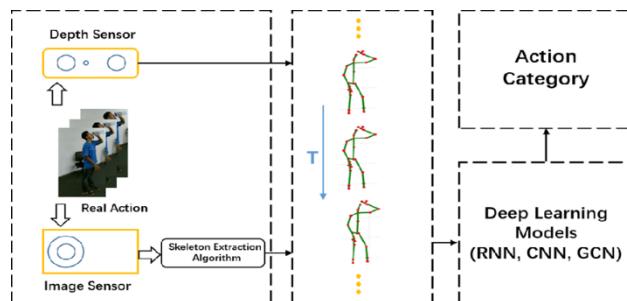


圖 8 深度學習基於骨架特徵提取行為辨識流程[20]

骨架特徵提取於深度學習之技術在人機協作領域取得的進展使其成為行為辨識研究的重要方向。然而，仍需要進一步研究以解決特定應用場景中的挑戰，如複雜背景、遮擋問題、不同人體尺度與視角變化。骨架辨識在行為識別中發揮著關鍵作用，未來的研究將集中在提高模型的穩健性、擴展應用場景、優化深度學習模型性能等方面，以實現更準確和廣泛的人體動作分析。

(三) 基於多模態特徵提取

在影像辨識深度學習技術中，多模態特徵提取的目的在於克服單一模態方法的限制，提升對影像場景的全面理解，以及提高模型的辨識性能。基於 RGB 模型的方法通常缺乏 3D 結構的資訊，且容易受到背景、光照等環境因素的干擾，進而降低辨識的精確性。另一方面，基於骨架的方法則受到色彩紋理和外觀訊息缺乏的限制，尤其在具有相似骨架運動的情況下難以區分[19]。透過多模態特徵提取的整合，可以利用各模態的優勢，實現更精確和全面的影像辨識。此外也能克服單一模態方法的過度擬合問題。

多模態特徵提取的核心在於整合不同數據模態，包括特徵層面融合[25]以及決策層面融合[26]。著名多模態特徵提取演算法有 Spatial-Temporal Region of Interest (ST-ROI)，結合影像的空間維度、時間維度。對區域的選取是基於對動作中重要部位的關注，能夠更有效地捕捉動作的時空特徵。這樣的方法有效彌補了 RGB 模型缺失的空間結構資訊，同時提高了對具有相似性的區分度[27]。ST-ROI 的概念與技術實現已在體育分析、人機交互以及自動駕駛系統等多個領域都有重要的應用。

在動作辨識中也提出通過將骨架模型的知識轉移至 RGB[28]模型，透過由骨架關節流中獲得的注意力遮罩，專注於 ST-ROI 區域，以提高對 RGB+D[21] 影片中人體動作的識別性能。多模態特徵提取方法能夠充分發揮不同基礎優勢，並解決各自模型的限制，從而提升動作辨識的全面性和準確性[27]。

圖 9 (a)為基於多模態特徵提取辨識流程：首先輸入骨架骨骼、骨架關節以及 RGB 影像，是從骨架關節的圖形表示中得出的關節權重，然後將權重與 ST-ROI 相乘得到 Focused ST-ROI。而(b)Focused ST-ROI 再來會被放入 ResNet 以生成模態預測，最後的預測結果為骨架骨骼、骨架關節以及 RGB 模態預測在 cross-entropy 的總和[20]。

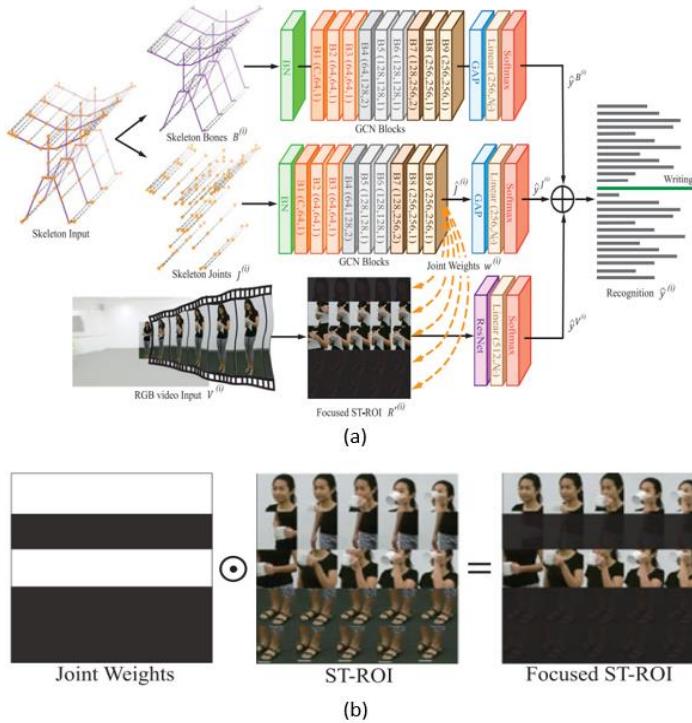


圖 9 顯示了多模態特徵提取方法的運作過程

二、人體動作辨識

人體動作辨識(Human Action Recognition, HAR)是一種涵蓋電腦視覺技術與深度學習技術的人體分析領域，透過讓電腦理解人類的動作，對人體在影像或影片中的動作進行分析和辨識，當今已廣泛投入不同領域運作，包括即時人體姿態評估、人機互動[29]、安全偵測防護[30]、運動科學[31, 32]和時間數據分析。現實世界中的人類活動包括簡單的單人肢體動作如拿取物體、全身協調發力運動如棒球投球，到多人互動識別或人跡互動識別，如產線機台安全防護防捲入壓迫等。

然而，由於人體動作的多樣性、視角變化[32]、遮擋以及光線條件的不確定性等因素，人體動作辨識仍面臨著諸多困難。過去研究會仰賴一些人工方法，將每個時間點上的座標資訊單獨地轉化為特徵向量，並在時間軸上進行分析，但這類做法並不能明確地發掘關節點在空間上的關聯性，這除了對行為的理解是一大障礙之外，在不同應用場景的泛用性也不高。當今已有眾多基於多模態特徵提取演算法，解決過去演算法之缺陷，如時空圖卷積神經網路(Spatial Temporal Graph Convolutional Networks for Skeleton, ST-GCN) 通過建立關節時空圖模型來處理動作序列；PoseC3D 結合三維卷積神經網路（3D CNN）與姿態估計技術，專注於從影像序列中提取人體姿態信息並進行動作識別；

SlowFast Networks 通過雙流架構，分別處理影像中的空間信息和時間信息，從而達到對動作的高準確識別。

(一) ST-GCN

Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action Recognition 於 2018 AAAI 發表。該演算法採用圖卷積網路 (Graph Neural Networks, GCN) 模型架構，對 Action Recognition 的骨架資訊設計出一種新穎的 Spatial-Temporal Graph Model，稱為 ST-GCN[33]。

如圖 10 ST-GCN 特徵點所示，ST-GCN 由 2 種 Edge 所構成，Spatial Edges：依照人體在生理上的關節點連接構成，Temporal Edges：相同關節點在不同時間上的連結。

如圖 11 所示，ST-GCN 透過將人體骨架資訊以 3D 座標向量的圖形式做重現，採用圖卷積網路基礎(Graph Neural Networks, GCNs)，進行關節特徵提取與時間序列建模 (Spatial Temporal Modeling)。結合在空間上的鄰近節點外，也加上了在時間軸上的鄰近節點，並進行分區策略 Partition Strategies 與節點學習權重 Learnable Edge Importance Weighting 等，對每個特徵節點之間的 Edge 做另一層的加權，藉此改善最終的成效，能夠捕捉人體動作中的細微變化，對於分析複雜動作特別有效。

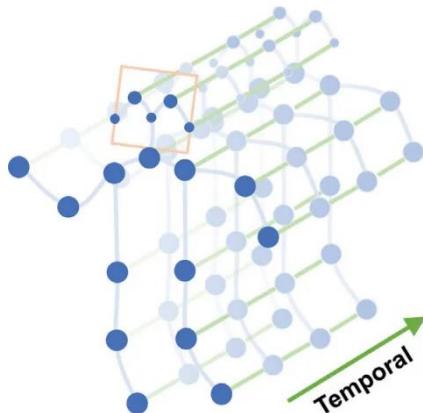


圖 10 ST-GCN 特徵點

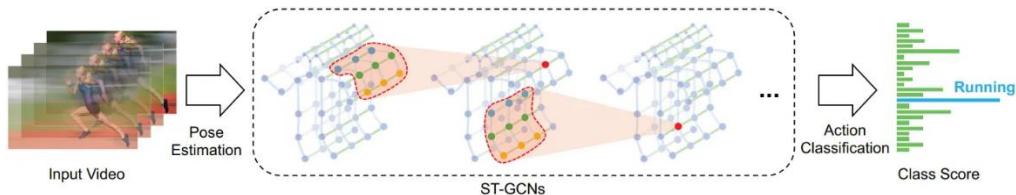


圖 11 ST-GCN 模型

(二) PoseC3D

PoseC3D 於 2022 CVPR 發表，名為 Revisiting Skeleton-Based Action Recognition 一文中[34]。作者提出了一種新的基於骨架的動作識別方法 PoseC3D，它依賴於 3D 热圖堆疊作為人體骨架的基本表示特徵。與基於 GCN 的方法相比，PoseC3D 在學習時空特徵方面更有效，對姿態估計噪點更具魯棒性(Robustness)，並且在跨數據集環境下具有更好的通用性。

此外，PoseC3D 可以在不增加計算成本的情況下處理多人場景，其功能可以在早期融合階段輕鬆與其他模式整合，這為進一步提升性能提供了巨大的設計空間。在四個具有挑戰性的數據集上，PoseC3D 無論單獨使用骨架數據還是結合 RGB 數據，皆持續獲得優異的性能。

如圖 12 所示，對於影像中的每一幀，首先使用兩階段姿勢估計（檢測+姿勢估計）進行人體姿勢提取。沿時間維度堆疊關節或肢體的熱圖，並對生成的三維熱圖進行預處理。最後使用 3D-CNN 對三維的熱圖進行分類。

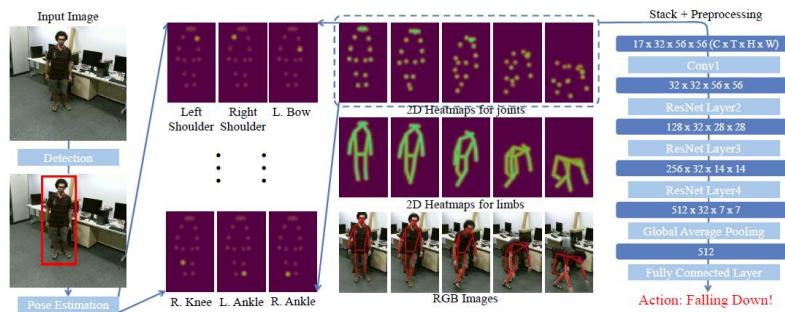


圖 12 PoseC3D 架構

(三) SlowFast Networks

SlowFast Networks 於 2019 SlowFast Networks for Video Recognition 一文提出[35]。傳統影像處理只有空間上(x,y) 兩個維度，但影像 (x,y,t)多了時間的維度，該論文認為時間和空間的維度不應等同看待，空間信息較為緩慢，如一個人在跑步，空間的信息不太發生變化，短時間內皆是人這個類別，但對於時間維度，變化是很快的。作者受到生物學靈長類視覺系統中視網膜的啟發，在視網膜細胞中 80%是 P-cell 負責空間和顏色，20%是 M-cell 負責響應運動變化，而開發 SlowFast Networks。

SlowFast Networks 透過建立雙路的 SlowFast 網路來分別處理空間和時間的訊息，如圖 13 所示。分別為 Slow Pathway，負責捕捉空間訊息，架構可以是任何的 3D 卷積網路，主要特色是使用低偵率輸入 (Large Temporal Stride)；Fast

Pathway，專門捕捉運動信息，採用高幀率輸入（Low Temporal Stride），並使用較少的通道數，僅為 Slow Pathway 的 $1/8$ 。這種低通道數設計可以大幅減少運算量，使 Fast Pathway 的運算量大約只有 Slow Pathway 的 $1/4$ ，這相當於 P,M 細胞的比例，同時也減少了對空間信息的敏感度。

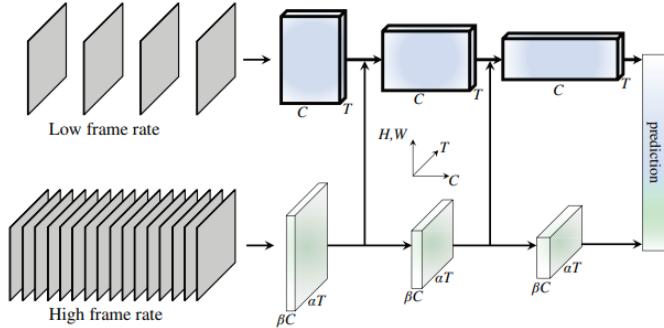


圖 13 SlowFast Networks

三、物件偵測/辨識（Object Detection）

物件偵測是計算機視覺中的一個關鍵任務，目的是在數字圖像和影片中檢測特定類別的語義對象實例，例如人、建築物或汽車，並返回每個物件實例的空間位置和範圍[2, 36]，其被廣泛應用於監控安全、自動駕駛[37]等領域。傳統的物件偵測方法是基於手工特徵和淺層可訓練架構的。然而，隨著深度學習的快速發展，引入了更強大的工具，能夠學習語義、高級和更深層次的特徵，以解決傳統架構中存在的問題[38-40]。

過去，不同的偵測任務具有各自的目標和限制，其困難程度各不相同。物件偵測中的挑戰包括物件旋轉和尺度變化、準確的物件定位、密集和遮擋物件偵測、加速偵測[41, 42]等。深度神經網路（DNNs）[43]的出現引入了具有卷積神經網路（CNN）特徵的區域（R-CNN）[44]，從而帶來了顯著的效益。相對於傳統方法，DNNs（特別是 CNNs）具有更深的架構，可以學習更加複雜的特徵，並且不需要手動設計特徵。自從 R-CNN 的提出以來，已經出現了許多改進的模型，例如 Fast R-CNN，結合了分類和邊界框回歸任務[38]；Faster R-CNN，使用額外的子網路生成區域建議[40]；以及 YOLO，通過固定網格回歸實現物體檢測。這些模型在原始 R-CNN 的基礎上帶來了不同程度的檢測性能提升，使得實時準確的物體檢測變得更加可行[39]。

目前，基於深度學習的物件偵測框架主要可以分為兩大類：Two-Stage 偵測和 One-Stage 偵測。

(一) Two-Stage 偵測：

Two-Stage 演算法是早期物件偵測中的一種常見方法，具較複雜的運算通道 [45, 46]。這些演算法將物件位置偵測和分類分開進行，並將偵測任務分為兩個階段：提議生成和區域分類。

1. 提議生成：使用選擇性搜索演算法（Selective Search），將圖片分割成多個區域，根據顏色、紋理、大小和位置計算區域間的相似度，合併相似區域直到達到預設的停止條件，產生可能含有目標的候選區域（Region Proposal）[40]。
2. 區域分類：通過深度卷積神經網路提取特徵向量[38]，然後進行對象預測分類。這些區域可能是背景，也可能是預先定義的類別對象，並對提議生成器的原始定位進行微調。

儘管 Two-Stage 演算法在物件偵測任務中為大量基礎邏輯，但也存在一些限制。首先，由於其將物件偵測和分類分開進行，導致它們在速度方面通常表現不佳，特別是在處理大量數據時。其次，Two-Stage 演算法需要進行多次計算，包括提取候選區域、特徵提取和分類，這增加了計算成本和時間成本。

Two-Stage 演算法主要包括 R-CNN、Fast R-CNN 和 Faster R-CNN。

(1) R-CNN

R-CNN 使用選擇性搜索演算法（Selective Search）生成 2000 個候選區域（Bounding Box），取代傳統的滑動窗口方法。這些區域基於顏色、紋理、規模和空間關係進行合併，選出最有可能包含物件的候選區域。然後，將這些區域轉換成 227x227 的圖像，輸入 CNN 進行特徵提取，每個 Bounding Box 生成 4096 個特徵向量。使用 SVM 分類器進行物件分類，再通過非最大抑制（NMS）篩選出合格的 Bounding Box，最後使用 Bounding-Box-Regression 微調位置，如圖 14 R-CNN 所示。

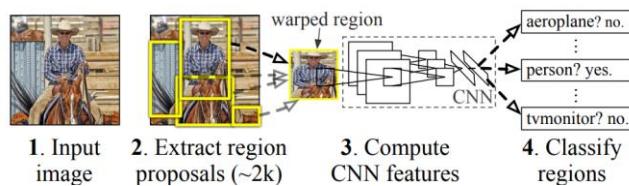


圖 14 R-CNN

(2) Fast R-CNN

由於 R-CNN 在訓練時消耗大量資源，需進行多次卷積，導致訓練時間過長，而 Fast R-CNN 通過僅需一次卷積大幅提升了效率，同樣使用選擇性

搜索 (Selective Search) 演算法生成候選區域。整張圖像輸入基於 VGG16 等預訓練模型修改的卷積神經網路，提取卷積特徵圖。接著，對每個 Bounding Box 使用 ROI 池化層從特徵圖中提取固定尺寸的特徵向量，並連接到兩個並行的全連接層。多任務損失函數同時優化分類和邊界框回歸，通過反向傳播和隨機梯度下降微調整個網路的權重。

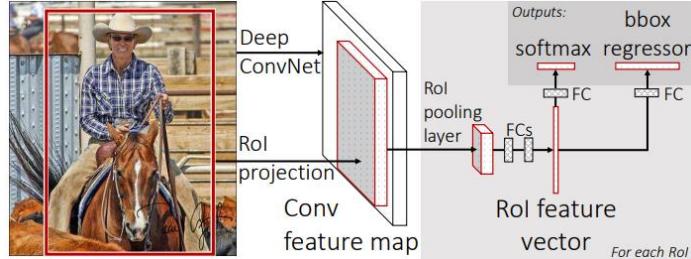


圖 15 Fast R-CNN

(3) Faster R-CNN

Faster R-CNN 與 Fast R-CNN 的主要區別在於引入了 Region Proposal Network (RPN)，使得生成區域提議的過程與 Fast R-CNN 檢測網路共享大部分卷積計算，從而大幅提高了計算效率。相比 Fast R-CNN 使用的 Selective Search 算法，Faster R-CNN 的計算速度更快且檢測精度更高。Faster R-CNN 先輸入圖像經過卷積網路提取特徵圖，並在最後一個共享卷積特徵圖上添加 RPN 層，如圖 16 所示。RPN 在特徵圖的每個位置上滑動一個小窗口，完成兩個任務：

- 物體分數預測：基於 3×3 窗口內的特徵，預測其中是否包含物體的分數。
- Bounding-Box-Regression：基於 3×3 窓口，對於當前位置的多個預先設置的 anchors，預測調整它們的回歸參數。

RPN 會為整個圖像生成數千個帶有物體分數預測和調整後的 Bounding Box 區域提議。使用非最大抑制 (NMS) 去除重疊的提議，僅保留分數最高的部分。將篩選後的提議輸入到 Fast R-CNN 物體檢測網路，使用 ROI 池化層提取每個提議的特徵，再進行分類和 Bounding-Box-Regression，得到最終的物體檢測結果。

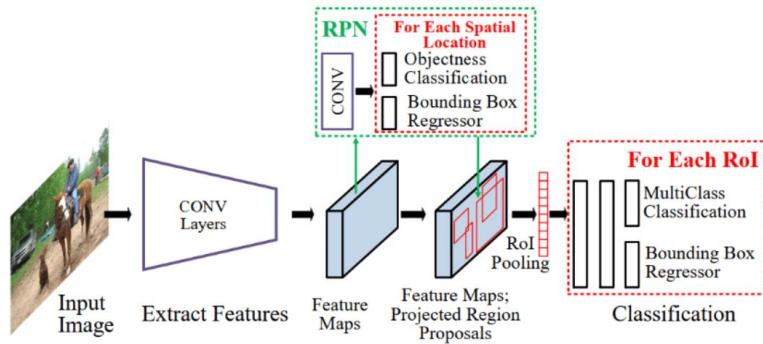


圖 16 Faster R-CNN

(二) One-Stage 偵測

與 Two-Stage 偵測算法不同，One-Stage 偵測器不包含單獨的提議生成階段。這是一種一次性偵測方法，將物件偵測視為一個單一統一的任務，神經網路直接對整張圖片進行物件偵測和定位，同時預測邊界框與類別[47-49]。這種方法能顯著減少時間成本。基於深度學習的早期成功的 One-Stage 偵測器之一是 OverFeat，由 Sermanet 等人開發。OverFeat 通過使用卷積層共享重疊區域的計算，只需一次前向通過網路即可完成偵測，顯示出與 R-CNN 相比的顯著速度優勢。然而，其分類器和回歸器的訓練是分開的，並未進行聯合優化。代表性的 One-Stage 物件偵測方法還包括 YOLO、SSD、RetinaNet 等。

1. SSD (Single Shot MultiBox Detector)

SSD (Single Shot MultiBox Detector) 的特點是它使用單階段方法直接從輸入圖像預測目標物件的邊界框和類別，無需複雜的候選框生成和分類流程[50]。這種方法利用多個不同尺度的特徵圖來預測各種大小的物件，並在每個特徵點位置使用多個預設邊界框，涵蓋不同的長寬比以更好地適應各種形狀的物件。這些預設邊界框用於預測是否包含物件，物件的類別，以及邊界框的調整值。SSD 的這種多尺度和多邊界框設計，使其在精確度上甚至超越了傳統的 Two-Stage 模型，並能有效檢測不同尺寸的物件。

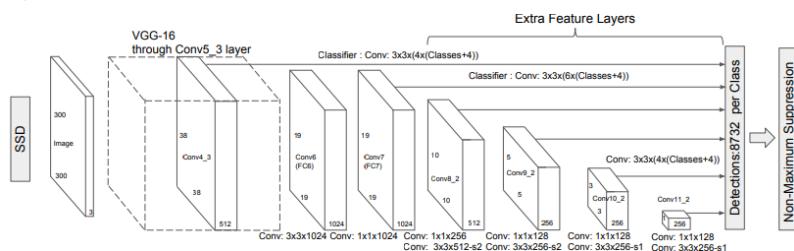


圖 17 SSD 的概述圖

2. RetinaNet

RetinaNet 引入了 Focal Loss 來有效解決單階段物件偵測器訓練時遇到的類別不平衡問題[48]。傳統的單階段偵測器容易受到背景區域的影響，導致模型偏向於預測背景類別。Focal Loss 通過調整損失權重，對置信度較低的樣本賦予更高的權重，使模型在訓練過程中更加關注難以分類的樣本，從而有效應對類別不平衡問題。

RetinaNet 首先使用 ResNet 等骨幹網路從輸入影像中擷取特徵圖，並透過特徵金字塔網路（FPN）融合多尺度特徵，如圖 18 所示。隨後，在特徵圖的每個位置布置多個 anchors，覆蓋不同尺寸和比例的物件。RetinaNet 進一步使用兩個小型全卷積子網路進行操作：一個進行二值分類以判斷每個 anchor 中是否含有物件，另一個預測物件在 anchor 中的精確位置。通過使用 Focal Loss，RetinaNet 在訓練時自動平衡樣本，專注於置信度較低的樣本，提高模型對難以分類樣本的準確性。

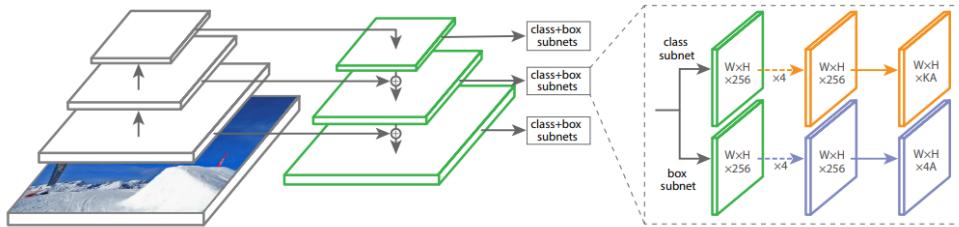


圖 18 RetinaNet

(三) Two-Stage 偵測 與 One-Stage 偵測比較

Two-Stage 偵測器在物體定位和識別的準確性方面通常表現較好，但在推理速度上則較慢。相對而言，One-Stage 偵測器具備更高的處理速度，適合實時物體偵測的應用。圖 19 展示了 Two-Stage 偵測器的基本架構，圖 20 展示了 One-Stage 偵測器的基本架構，圖中的黃色立方體是一系列具有相同解析度的卷積層（稱為塊），因為在一個塊之後進行下採樣操作，後續立方體的尺寸逐漸變小。厚藍色立方體是包含一個或多個卷積層的一系列卷積層。扁平藍色立方體示出了 RoI 池化層，用於為相同大小的對象生成特徵圖。

表 1 Two-Stage 偵測 與 One-Stage 偵測的優缺點

	Two-Stage	One-Stage
優點	精度高(定位、檢出率)	速度快
缺點	速度慢；訓練時間長	精度低(定位、檢出率)；小物體的偵測效果不好

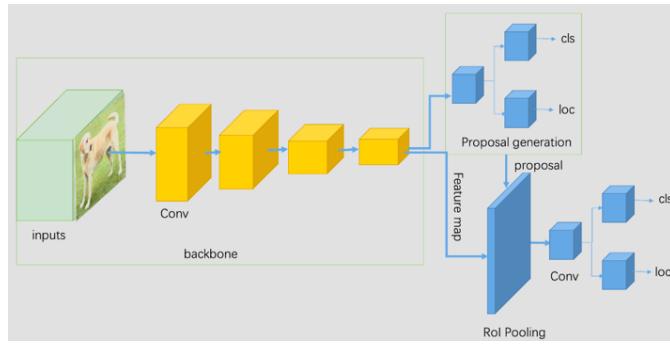


圖 19 Two-Stage 偵測器的基本架構

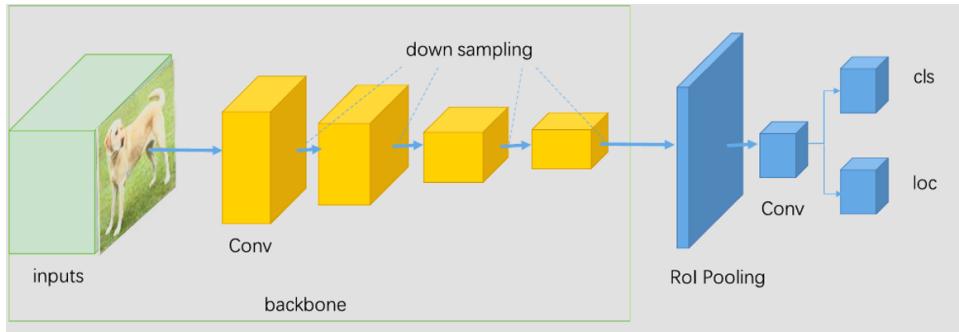


圖 20 One-Stage 偵測器的基本架構

四、YOLO v7

YOLO (You Only Look Once)是一種物體檢測和分類演算法，能夠快速且準確地在單一圖像中定位和識別多個物體，首次於 2016 年提出[39]。YOLO 演算法的基本概念是將整個圖像分成網格，每個網格負責檢測圖像中的一個物體。一個物體可能會出現在多個網格中，每個網格通過卷積神經網路預測物體的類別和位置。YOLO 演算法整合了分類和檢測任務，通過分類的概率和邊界框的信心度來統一計算物體的最終概率，與傳統演算法相比，YOLO 演算法有許多優點，因為它使用全卷積神經網路，所以具有較少的網路參數和較短的訓練時間。此外，YOLO 演算法的網路架構簡單易懂，易於實現，並能適應不同類型的圖像檢測任務，同時 YOLO 演算法能獨立檢測不同規模和類型的物體，達到良好的檢測準確性，物體邊界框的位置和大小的準確性不亞於傳統方法。

YOLO 也衍生出一系列演算法，YOLO9000[51]、YOLO v3[52]、YOLO v4[53]、YOLO v5[54]、YOLO v6[55]、YOLO v7[56]，本研究將使用 YOLO v7 為模型基礎。

YOLO v7 模型由 Chien-Yao Wang 和 Alexey Bochkovskiy 等人於 2022 年開發[56]，集成了如 E-ELAN（擴展高效層聚合網路）[57]、基於連接的模型縮放策略[58]和模型重新參數化等策略[59]，實現檢測效率與精確度間的良好平

衡。如圖 21 所示，YOLO v7 網路由四個獨立模組組成：輸入模組、背景網路、頭部網路和預測網路。

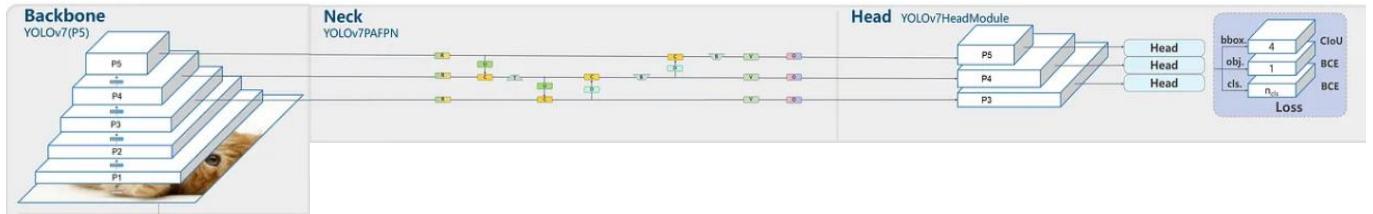


圖 21 YOLO v7

(一) 輸入模組：

YOLO v7 模型的預處理階段使用馬賽克和混合數據增強技術，並利用 YOLO v5 建立的自適應錨框計算方法，確保輸入的彩色圖像均勻縮放至 640×640 尺寸，滿足背景網路的輸入大小要求。

(二) 背景網路：

YOLO v7 網路包括三個主要部分：CBS、E-ELAN 和 MP1。CBS 模組由卷積、批量正規化和 SiLU 激活函數組成；E-ELAN 模組保持原始 ELAN 設計架構，通過引導不同特徵組計算塊學習更多樣化的特徵來增強網路的學習能力，保留原始梯度路徑；MP1 由 CBS 和 MaxPool 組成，分為上下兩個分支，上分支使用 MaxPool 將圖像的長寬減半，並使用輸出通道為 128 的 CBS 將圖像通道減半，下分支通過具有 1×1 核心和步長的 CBS 將圖像通道減半，並通過 3×3 核心和 2×2 步長的 CBS 將圖像長寬減半，最終通過串接操作融合兩個分支提取的特徵；MaxPool 在小局部區域提取最大值信息，而 CBS 提取小局部區域的所有值信息，從而提高網路的特徵提取能力。

(三) 頭部網路：

YOLO v7 的頭部網路採用特徵金字塔網路（FPN）架構，應用 PANet 設計。該網路包含數個卷積、批量正規化和 SiLU 激活（CBS）塊，以及引入空間金字塔池化和卷積空間金字塔池化（Sppcspc）結構、擴展高效層聚合網路（E-ELAN）和 MaxPool-2（MP2）。Sppcspc 結構通過在空間金字塔池化（SPP）結構內引入卷積空間金字塔（CSP）結構和大的殘差邊，改進網路的感知域，有助於優化和特徵提取；基於 E-ELAN 的 ELAN-H 層融合了幾個特徵層，進一步加強特徵提取；MP2 塊的結構與 MP1 塊類似，只是對輸出通道數進行了輕微調整。

(四) 預測網路：

YOLO v7 的預測網路使用 Rep 結構調整頭部網路輸出特徵的圖像通道數，然後應用 1×1 卷積進行信心度、類別和錨框的預測。受到 RepVGG 啟發的 Rep 結構引入了特殊的殘差設計(Ding et al., 2021)，以輔助訓練過程，這一獨特的殘差結構[60]在實際預測中可以簡化為簡單的卷積，從而降低網路複雜度，而不損害其預測性能。

參、 研究方法

一、研究流程

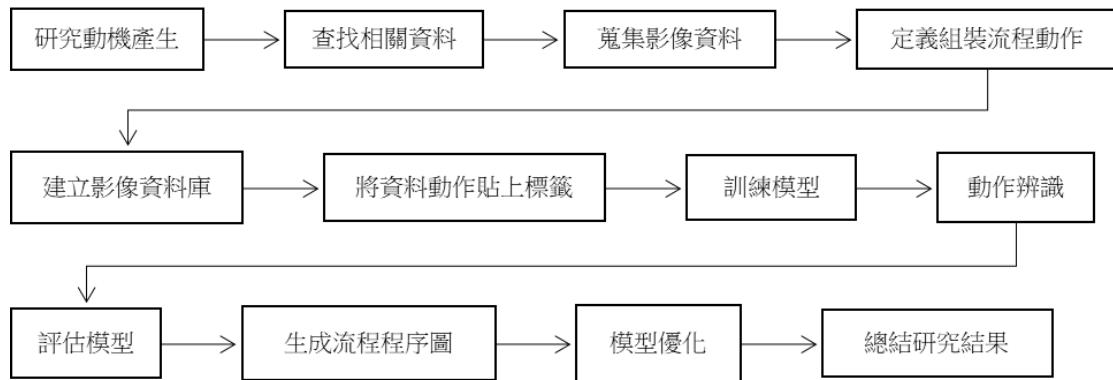


圖 22 研究流程圖

(一) 研究主要步驟如下：

1. 制定動作標準與影像資料蒐集

本研究會依據上述組裝流程制定標準動作、模擬環境、在固定位置架設攝影機，安排多位操作人員重複進行組裝動作並錄製，以取得大量影像資料。在蒐集時須注意保持個體間的差異性，減少相似資料反覆被蒐集，避免後續模型擬合問題。

(二) 生成流程程序圖

將收集完成的資料貼標後匯出一個 csv 檔，再經由程式處理將影片及標籤結合匯出為貼標完成之影片，輸入要訓練的動作辨識模型。模型訓練完成後，透過 YOLO v7 判斷作業員是否在執行有效動作，SlowFast 判斷作業員動作為何，再將辨識出來的動作計算時間，進一步生成流程程序圖。

(三) 模型優化以解決問題

動作辨識模型誤判的機率較高的情況，是由於無效動作涵蓋較多不同類型之動作，容易造成致整體衡量指標降低。因此我們利用三種不同的貼標方式來訓練模型，以有效提升整體精確率及召回率，進而優化流程程序圖之預測。

二、數據採集

(一) 影像資料蒐集方法

本研究之影像資料蒐集，實際將 Microsoft 攝影機架設於產學合作公司工作站上進行影像蒐集。將攝影機放置於組裝人員正前方，並以約 45 度角

向下拍攝，如圖 23 所示，畫面長寬為 1920*1080，幀數為 30fps。每條生產線共有六個工作站，每站蒐集約 20 到 30 分鐘，相同工作站蒐集上午及下午兩個時段的組裝人員生產過程。並選擇第三工作站作為此次主要研究之資料。



圖 23 Microsoft 攝影機架設示意圖

三、影像處理及操作流程

(一) 影像處理

將收集到之第三工作站影像，以人工貼標方式將影片中的動作逐幀定義為標準流程動作，貼標程式如圖 24 所示。第一工作區(圖 24 中的①)標籤名稱設定及標籤代表顏色，第二工作區(圖 24 中的②)為影片標籤示意區，顯示已貼好的標籤顏色，且進行標籤修改及刪除，第三工作區(圖 24 中的③)為工作區設定，設定影片中工作區、工作人員位置及手腕大小及位置設定。上午的第三工作站共有 9 個組裝動作，其動作編號、標準動作名稱和示意圖片，並將非組裝流程之動作，如：離開座位、拿取上一工作站之半成品、等待時間皆定義為 other，整理如圖 25，完成貼標後即匯出 csv 檔為模型原始資料。



圖 24 貼標程式示意圖(為保護公司機密資料故將影像作模糊處理)

動作代號	Screw Up	Brush	Assemble 3.1	Assemble 3.2
組裝動作	鎖螺絲	塗油	將支架側蓋裝於 支架	裝上支架外蓋
示意圖片				
動作代號	Assemble 3.3	Assemble 3.4	Assemble 3.5	other
組裝動作	將鐵片及馬達 放上治具	取小齒輪放上 鐵片	用鑷子取 C 型環 裝於小齒輪	非標準之動作
示意圖片				

圖 25 第三站之 SOP 動作代號、組裝動作說明與示意圖

(為保護公司機密資料故將影像作模糊處理)

(二) 工作站操作流程圖

本研究使用實際產學合作公司操作員組裝流程之資料，圖 26 為第三工作站之操作流程。

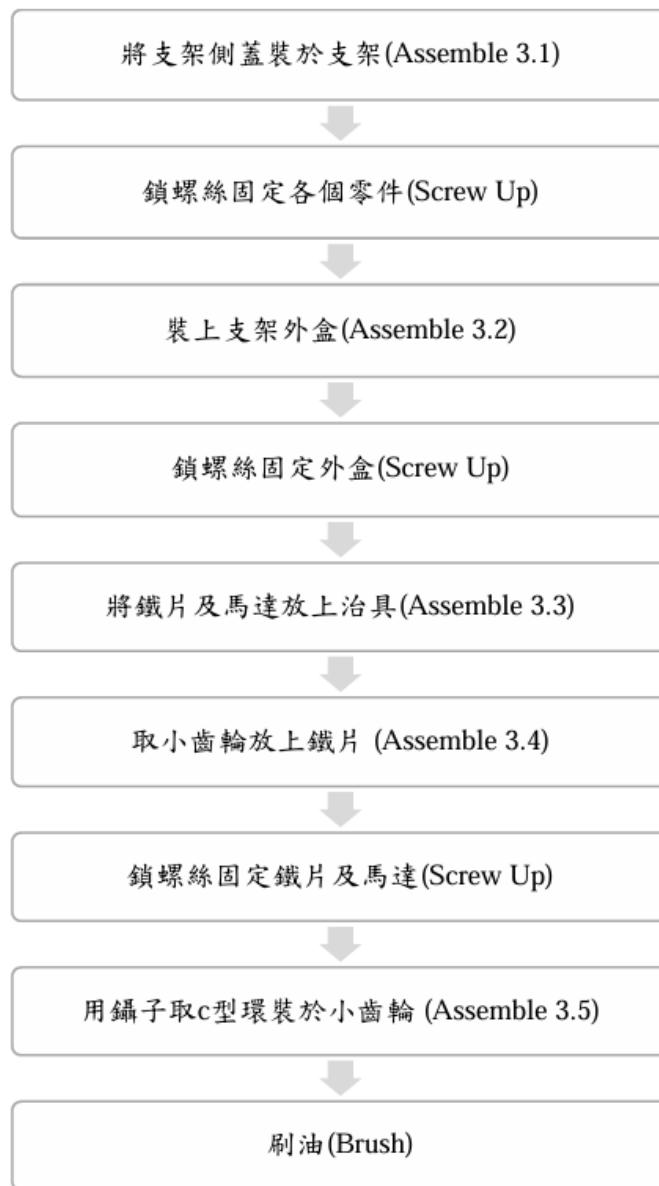


圖 26 工作站操作流程圖

四、研究工具

(一) SlowFast

SlowFast 是用於影片識別的網路，其獨特的雙路徑結構和預訓練策略使該框架在準確率、速度和模型複雜度方面皆有出色表現，可同時捕捉影片的空間和時間訊息，從而實現高效的影片模型訓練。在本計畫中，我們將先前錄製好工作站的影片預處理後，輸入 SlowFast 模型進行訓練，訓練完成後將其用於操作員動作檢測，以生成流程程序圖。

(二) YOLO v7(You Only Look Once)

YOLO v7 是一種卓越的實時目標檢測系統，優化改進了模型的準確率和效能，使其更準確和更高效。使用了模型縮放與擴展方法，有效利用了參數量和計算量，並具有更快的推理運算速度和準確率，這種高效性使其成為眾多實時應用的首選，無論是在自動駕駛、監控系統還是其他需要即時物體檢測的領域。在本計畫中，我們將利用 YOLO v7 來檢測和追蹤場景中的物體以填補骨架辨識在物件辨識方面的不足，提供更全面的操作環境理解，尤其在操作員與物體互動的場景。

五、模型設計

(一) 程式架構

1. 工作站影像：流程開始於工作站影像的收集階段。這些影像來自產學合作公司的工作環境，並被存儲起來以供後續處理使用。
2. 貼標籤：影像資料隨後進行貼標，貼標過程將影像數據劃分為影像和標籤兩個部分，為後續的分析提供基礎。
3. 人員偵測模型訓練：影像和標籤數據被進一步處理，這包括作業員偵測模型的建立和篩選，這些模型用於識別影像中的作業員，從而提取與動作相關的數據特徵，這些特徵將用於動作辨識模型的訓練。
4. 動作辨識模型訓練：前處理後的資料被用來進行模型選擇和特徵篩選 (Filter method)，並建立相應的過濾器。這一步驟的目的是選擇最適合的模型和特徵，以提高模型對不同動作的辨識能力和準確性。
5. 動作辨識：在動作辨識模型訓練完成後，實際工作站的影像會輸入到訓練好的模型中，進行動作偵測和辨識，這包括作業員的偵測、動作的辨識以及應用過濾器來過濾不相關或噪音數據，以提高整體辨識的準

確性。動作偵測完成後，影像和偵測結果會進行影像合成，這一步驟將偵測到的動作信息與原始影像結合，生成一個包含動作辨識結果的綜合影像。

6. 評估：通過對整個資料處理和模型訓練過程進行評估，確保每個步驟的處理和訓練結果都是準確和有效的。
7. 建立流程程序圖：基於評估結果，最終建立一個完整的流程程序圖，為後續的應用和改進提供參考。

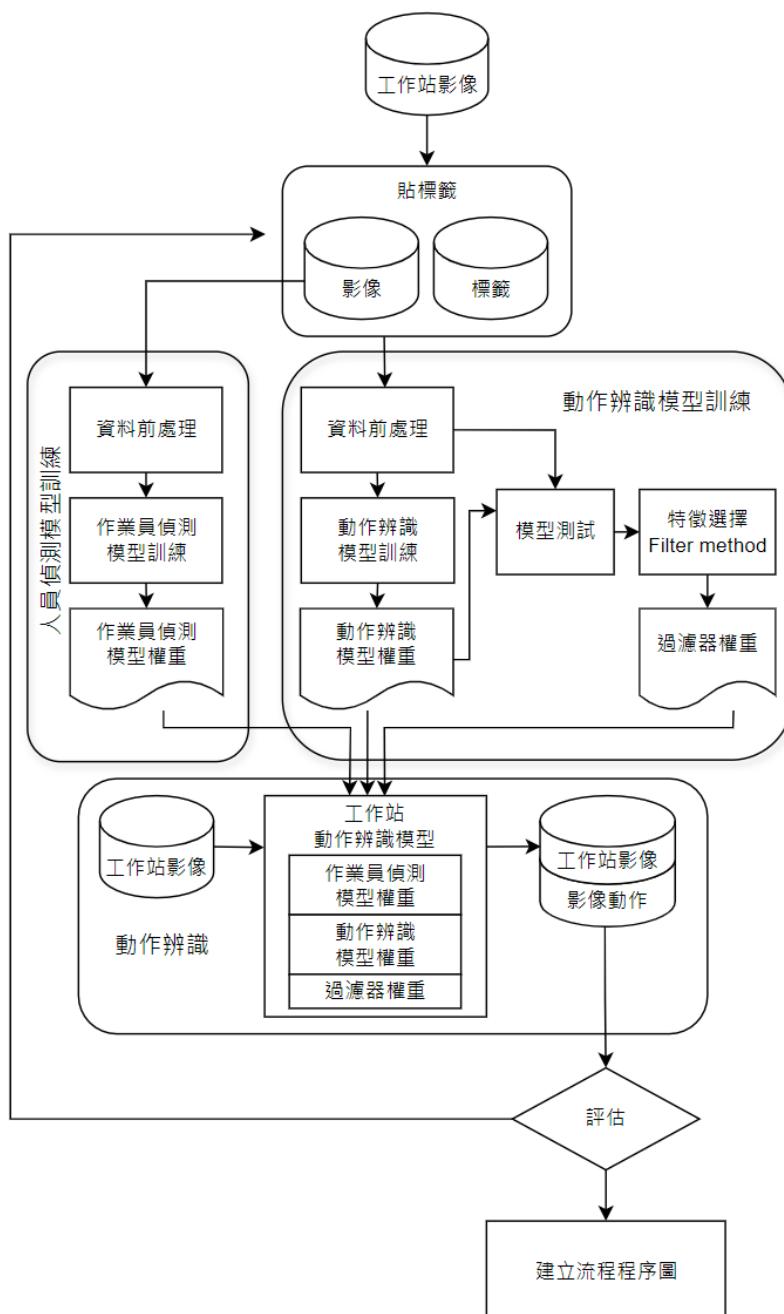


圖 27 程式架構流程圖

(二) 系統環境及電腦設備

本專題研究之電腦運算設備，圖形處理單元(GPU)使用 NVIDIA GeForce RTX4080，中央處理單元(CPU)使用 AMD Ryzen9 7940HS，隨機存取記憶體(RAM)使用 32GB DDR5。相關系統環境、套件版本如下：

- Python : 3.9.12
- Matplotlib : 3.7.1
- OpenCV-Python : 4.5.1.48
- Pillow : 9.0.1
- Scikit-learn : 1.2.2
- Seaborn : 0.13.2
- tqdm : 4.63.0
- NumPy : 1.22.3
- Pandas : 1.5.3

(三) 模型訓練

以人工方式進行樣本貼標後，將影片和貼完之標籤投入訓練動作辨識模型。此程式使用的 subprocess 套件將多個子程式組成，以下列點說明主程式中的各個子程式執行細節。

1. 設定初始參數：使用 argparse 解析命令列參數，執行程式時指定專案名稱、運算子權重、動作權重等，在本計畫中使用預設的 YOLO v7 和 SlowFast 模型權重，如圖 28 所示。

```
14 parser = argparse.ArgumentParser()
15 parser.add_argument("--project", type=str, default="Demo", help="專案名稱")
16 parser.add_argument("--operator-weights", type=str, default="models/operator/init.pt", help="預設使用yolo模型")
17 parser.add_argument("--action-weights", type=str, default="models/action/init.pth", help="預設使用slowfast模型")
18 args = parser.parse_args()
```

圖 28 設定初始參數

2. 建立相應的目錄結構和設置相關的路徑：使用 os 模組來操作目錄和路徑，確保在開始專案訓練之前，所有必要的目錄都被正確地創建，並且初始化了相關的路徑變數，以便後續的模型訓練能夠順利進行，如圖 29 所示。

```

20 now = datetime.now()
21 series = now.strftime("%Y%m%d%H")
22
23 projects = [args.project]
24
25 # models output path
26 path_ = os.getcwd()
27 input_path = os.path.join(path_, "input")
28 temp_path = os.path.join(path_, "temp_train")
29 model_path = os.path.join(path_, "models")
30 operator_path = os.path.join(model_path, "operator")
31 action_path = os.path.join(model_path, "action")
32 filter_path = os.path.join(model_path, "filter")
33
34 for path in [temp_path, operator_path, action_path, filter_path]:
35     os.makedirs(path, exist_ok=True)
36
37 for project_ in projects:
38     print(project_)
39
40 # -----
41 # initialize
42 # -----
43 print("-----initialize-----")
44 project_name = f"{project_}_{series}"
45 print(project_name)
46 input = os.path.join(input_path, project_)
47 temp_train = os.path.join(temp_path, project_name)
48 temp_filter = os.path.join(temp_train, 'temp-filter')
49 os.makedirs(temp_filter, exist_ok=True)
50 init_operator_weights = args.operator_weights #os.path.join(operator_path, "init.pt")
51 init_action_weights = args.action_weights #os.path.join(action_path, "init.pth")
52 trained_action_weights = os.path.join(action_path, f"{project_name}.pth")
53 trained_filter_weights = os.path.join(filter_path, f"{project_name}.txt")

```

圖 29 建立相應的目錄結構和設置相關的路徑

3. 資料前處理

- (1) 參數設定：調用 dataPrepare 模組中的 data_prepare 函式並傳遞參數，對輸入的資料進行前處理，為後續的模型訓練做好準備，如圖 30。

```

58     ...
59     args:
60         input: 輸入資料夾，包含labels和videos
61         output: 輸出資料夾
62         fps: 影片fps，預設為30fps
63         interval: 每段影片的時間長度(單位:幀)，預設為20幀
64         train_data: 每個影片的訓練資料數量，預設為600筆
65         test: 是否僅生成測試資料；預設為訓練資料
66     ...
67     print("-----get dataloader-----")
68     data_prepare(input=input,
69                 output=temp_train,
70                 fps=30,
71                 interval=20,
72                 train_data=600,
73                 test=False)

```

圖 30 參數設定

(2) dataPrepare.py 程式執行

- a. 將讀取的檔案轉為標記檔 labels-action.csv：圖 31 使用 glob 函式找到並讀取指定路徑下的 label 檔，接著將所有 csv 檔合併，在合併過程中創建了一個空的 DataFrame 用於存放合併後的標籤資訊，透過映射(.map)將 action 列的字符串標籤轉換為相應的整數標籤，以確保後續對數據的處理能夠保持一致性。最後將處理後的 DataFrame 存入 labels-action.csv，如圖 32 所示。

```

29     # 讀取標籤檔
30     label_path = os.path.join(input, "labels")
31     save_path = os.path.join(folder, "labels-action.csv")
32     print(f"saving label file to {save_path}...")
33
34     # 合併所有label檔
35     labels = glob(os.path.join(label_path, "*.csv"))
36     merge = pd.DataFrame()
37     for label in labels:
38         file_name = os.path.basename(label).replace(".csv", "")
39         file = pd.read_csv(
40             label, encoding="latin1", names=[1, 2, "frame", "action", 3])
41     )
42     file.drop([1, 2, 3], axis=1, inplace=True)
43     file.drop(0, axis=0, inplace=True)
44     file["video"] = file_name
45     file["frame_name"] = file["video"] + "_" + file["frame"]
46
47     merge = pd.concat([merge, file], axis=0)
48
49     # 顯示動作標籤資訊
50     labels = sorted(merge.action.unique())
51     labels.remove("Other")
52     labels = {v: k for k, v in enumerate(labels)}
53     labels["Other"] = len(labels)
54     print(f"The number of data: {merge.shape[0]}")
55     print(f"The number of labels: {len(labels)}")
56     print(f"The label list: {labels}")
57
58     # 對動作標籤進行編號
59     merge["label"] = merge["action"].map(labels)
60     merge["label"] = merge["label"].astype("int")
61
62     # 儲存合併後的label檔 --> labels-action.csv
63     merge.to_csv(save_path, index=False)
64

```

圖 31 將讀取的檔案轉為標記檔 labels-action.csv

	A	B	C	D		A	B	C	D	E	
1	newBox	Time	Frame	Action		1	frame	action	video	frame_name	label
2	newBox	00:00		1 Other		2	1 Other	0912_am_3th	0912_am_3th_1		7
3	newBox	00:00		2 Other		3	2 Other	0912_am_3th	0912_am_3th_2		7
4	newBox	00:00		3 Other		4	3 Other	0912_am_3th	0912_am_3th_3		7
5	newBox	00:00		4 Other		5	4 Other	0912_am_3th	0912_am_3th_4		7
6	newBox	00:00		5 Other		6	5 Other	0912_am_3th	0912_am_3th_5		7
7	newBox	00:00		6 Other		7	6 Other	0912_am_3th	0912_am_3th_6		7
8	newBox	00:00		7 Other		8	7 Other	0912_am_3th	0912_am_3th_7		7
9	newBox	00:00		8 Other		9	8 Other	0912_am_3th	0912_am_3th_8		7

圖 32 (左)原讀取的 label 檔(右)labels-action.csv

- b. 圖 33 將字符串標籤與其對應的整數標籤存入類別檔 classes.txt，結果如圖 34。

```
64     # 將標籤存入txt檔 --> classes.txt
65     file_name = os.path.join(folder, "classes.txt")
66     with open(file_name, "w") as file:
67         for label, index in labels.items():
68             line = f"{index}: {label}\n"
69             file.write(line)
```

圖 33 將標籤存入 classes.txt

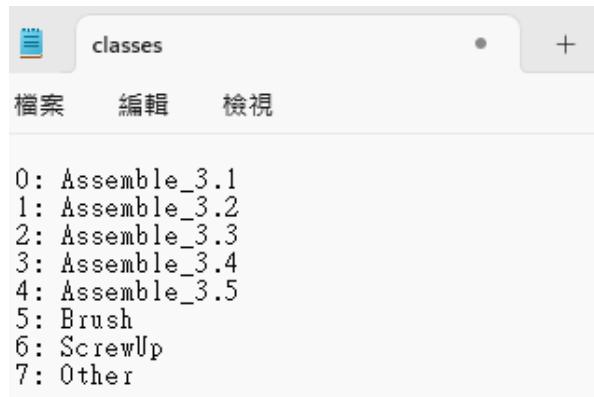


圖 34 classes.txt

- c. 將標記檔 labels-action.csv 轉換為訓練資料格式 labels generate.csv：

1. Start/end： 影片開始與結束之秒數。通過將影片幀數除以每秒幀數 (fps) 來計算每一幀的起始時間，再基於起始時間和給定的時間間隔 (interval) 來計算每一幀的結束時間。
2. Used： 標籤一致性檢查。對每部影片，檢查在指定的幀數間隔內標籤是否保持一致。如果在間隔開始和結束的標籤一致，則該幀被標記為可用 (1)，否則為不可用 (0)。此作法確保在一定時間間隔內的每一個影片幀的標籤保持一致，以確保資料的連續性和一致性。
3. train： 根據先前所設定的 train data 數量和標籤類別的數量，為訓練數據集中的每個標籤分配一定數量的資料，並將這些資料標記為訓練資料。本計畫以訓練資料總數除以標籤數量的商，計算每個標籤應分配的資料數量，將被選定的訓練資料的 train 欄位標記為 1，而未被選定為訓練資料的 train 欄位標記為 0。將資料集劃分為訓練資料和非訓練資料，以便進行機器學習模型的訓練。

經過以上處理步驟，程式生成了 labels-generate.csv 檔案如圖 36，其中包含了當前幀數、動作名稱、輸入檔案名稱、檔案名稱_當前幀數、標籤編號、每個影片幀的開始時間和結束時間、標籤一致性檢查結果和訓練資料標記。

```
# 將標記檔轉換為訓練資料格式 --> labels-generate.csv
# 設定每段影片資料的開始時間與結束時間 = interval/fps秒 (預設20/30s)
save_path = os.path.join(folder, "labels-generate.csv")
print(f"saving file to {save_path}...")

merge["start"] = merge["frame"].apply(lambda x: round(int(x) / fps, 2))
merge["end"] = merge["start"].apply(lambda x: round(x + (interval / fps), 2))
videos = merge.video.unique()

# 設定可使用的資料：在一段區間內的動作標籤必須相同，若超出區間則不使用該幀資料
used = []
for video in videos:
    temp = merge[merge.video == video]
    labels = temp.label.values

    for i in range(len(temp) - interval):
        current_label = labels[i]
        compare_label = labels[i + interval]
        used.append(1) if current_label == compare_label else used.append(0)

    used.extend([0] * interval)
    del temp
merge["used"] = used
merge["used"] = np.where(merge.index % (interval // 4) == 0, merge.used, 0)
merge.reset_index(drop=True, inplace=True)
merge = merge[merge.used == 1]

# 依照每個工作站的資料量進行資料分割 (預設為600筆[train_data])
used = []
for video in videos:
    labels_ = merge[(merge.video == video)].label.unique()
    num_label = int(train_data / len(labels_))
    print(f"{video} has {num_label} data for each label.")

    for label in labels_:
        temp = (
            merge[(merge.video == video) & (merge.label == label)]
            .head(num_label)
            .index.values
        )
        used.extend(temp)

merge.loc[used, "train"] = 1
merge.loc[merge[merge.train != 1].index, "train"] = 0
merge["train"] = merge["train"].astype(int)

# 儲存處理後的label檔
merge.to_csv(save_path, index=False)
```

圖 35 將 labels-action.csv 轉換為 labelsgenerate.csv

	A	B	C	D	E	F	G	H	I
1	frame	action	video	frame_name	label	start	end	used	train
2	185	Assemble_3.1	0912_am_3th	0912_am_3th_185	0	6.17	6.84	1	1
3	190	Assemble_3.1	0912_am_3th	0912_am_3th_190	0	6.33	7	1	1
4	195	Assemble_3.1	0912_am_3th	0912_am_3th_195	0	6.5	7.17	1	1
5	200	Assemble_3.1	0912_am_3th	0912_am_3th_200	0	6.67	7.34	1	1
6	205	Assemble_3.1	0912_am_3th	0912_am_3th_205	0	6.83	7.5	1	1
7	210	Assemble_3.1	0912_am_3th	0912_am_3th_210	0	7	7.67	1	1
8	215	Assemble_3.1	0912_am_3th	0912_am_3th_215	0	7.17	7.84	1	1
9	220	Assemble_3.1	0912_am_3th	0912_am_3th_220	0	7.33	8	1	1
10	225	Assemble_3.1	0912_am_3th	0912_am_3th_225	0	7.5	8.17	1	1

圖 36 labels generate.csv

- d. 圖 37 根據 labels-generate.csv 檔案中的資料，讀取相應的影片檔案，並從中提取影片的名稱、片段、開始時間、結束時間、標籤和幀數，在影片進行處理的過程中，程式使用了 EncodedVideo.from_path 函式來讀取並解碼影片檔案，然後根據每個片段的時間範圍從影片中提取對應的影片片段。

每個片段都會被轉換成 tensor 格式並儲存為.pt 檔案（影片解碼檔）。

```

144      # 讀取標籤檔與影片檔並儲存至建立之暫存資料夾 --> train//..data.pt
145      # 建立路徑
146      save_path = os.path.join(folder, "train")
147      os.makedirs(save_path, exist_ok=True)
148      print(f"saving file to {save_path}...")
149
150      # 讀取標籤檔
151      dataset = merge[merge.train == 1]
152      print(f"The number of dataset: {len(dataset)}")
153
154      # 讀取並解碼影片檔
155      video_path = [
156          os.path.join(input, "videos", f"{video}.mp4")
157          for video in dataset.video.unique()
158      ]
159      for video_ in video_path:
160          print(f"Processing the video file [{video_}]...")
161          encoded_video = EncodedVideo.from_path(video_)
162          video_name = os.path.basename(video_).replace(".mp4", "")
163          video_segment = dataset[dataset.video == video_name]
164
165          start = video_segment["start"].values
166          end = video_segment["end"].values
167          labels = video_segment["label"].values
168          frames = video_segment["frame"].values
169          series = list(video_segment.index)
170
171          # 儲存影片片段為tensor檔(.pt)
172          for i in tqdm(range(len(start))):
173              clip = encoded_video.get_clip(start_sec=start[i], end_sec=end[i])
174              clip = clip["video"]
175              torch.save(
176                  clip,
177                  os.path.join(
178                      save_path,
179                      f"{str(series[i]).zfill(6)}-{video_name}-{A{labels[i]}}.pt",
180                  ),
181              )
182          del encoded_video
183
184          print("Completed processing and saving the label file!")

```

圖 37 將資料儲存為影片解碼檔

- e. 圖 38 通過使用 glob 函數找到指定路徑下所有的 tensor 格式的影片解碼檔，將其檔案中資訊轉換為訓練資料集格式存入 DataFrame 中，並保存為 labels-train.csv，如圖 39，後續供 Dataloader 載入資料使用。

```

188     # 根據資料夾中的tensor檔(影片解碼檔)，生成訓練資料集並儲存--> labels-train.csv
189     # 讀取tensor檔
190     dataset = glob(os.path.join(save_path, "*.pt"))
191
192     save_path = os.path.join(folder, "labels-train.csv")
193     print(f"saving training data to the dir: {save_path}...")
194
195     # 將tensor檔的資訊存入dataframe
196     file = pd.DataFrame(dataset, columns=["path"])
197     file["series"] = file["path"].apply(
198         lambda x: os.path.basename(x).replace(".pt", "").split("-")[0][1:])
199     )
200     file["video"] = file["path"].apply(
201         lambda x: os.path.basename(x).replace(".pt", "").split("-")[1]
202     )
203     file["label"] = file["path"].apply(
204         lambda x: os.path.basename(x).replace(".pt", "").split("-")[2][1:]
205     )
206
207     # 顯示訓練資料分布
208     print(
209         file.pivot_table(
210             index="video",
211             columns="label",
212             values="series",
213             aggfunc="count",
214             fill_value=0,
215         )
216     )
217
218     # 儲存訓練資料集

```

圖 39 將影片解碼檔轉換為訓練資料集格式 labels-train.csv

	A	B	C	D
1	path	series	video	label
2	/E/Aver/temp_train/Demo_2024051518/train/S000004-0912_am_3th-A0.pt	4	0912_am_3th	0
3	/E/Aver/temp_train/Demo_2024051518/train/S000009-0912_am_3th-A0.pt	9	0912_am_3th	0
4	/E/Aver/temp_train/Demo_2024051518/train/S000014-0912_am_3th-A0.pt	14	0912_am_3th	0
5	/E/Aver/temp_train/Demo_2024051518/train/S000019-0912_am_3th-A0.pt	19	0912_am_3th	0
6	/E/Aver/temp_train/Demo_2024051518/train/S000024-0912_am_3th-A0.pt	24	0912_am_3th	0
7	/E/Aver/temp_train/Demo_2024051518/train/S000029-0912_am_3th-A0.pt	29	0912_am_3th	0
8	/E/Aver/temp_train/Demo_2024051518/train/S000034-0912_am_3th-A0.pt	34	0912_am_3th	0
9	/E/Aver/temp_train/Demo_2024051518/train/S000039-0912_am_3th-A0.pt	39	0912_am_3th	0
10	/E/Aver/temp_train/Demo_2024051518/train/S000044-0912_am_3th-A0.pt	44	0912_am_3th	0

圖 38 labels-train.csv

(3) 程式輸出

```

-----get dataloader-----
saving label file to /E/Aver/temp_train/Demo_2024051518/labels-action.csv...
The number of data: 14183.
The number of labels: 8.
The label list: {'Assemble_3.1': 0, 'Assemble_3.2': 1, 'Assemble_3.3': 2, 'Assemble_3.4': 3, 'Assemble_3.5': 4, 'Brush': 5, 'ScrewUp': 6, 'Other': 7}
saving file to /E/Aver/temp_train/Demo_2024051518/labels-generate.csv...
0912_am_3th has 28 data for each label.
saving file to /E/Aver/temp_train/Demo_2024051518/train...
The number of dataset: 196
Processing the video file [/E/Aver/input/Demo/videos/0912_am_3th.mp4]...
100% | 196/196 [14:46<00:00, 4.52s/it]
Completed processing and saving the label file!
saving training data to the dir: /E/Aver/temp_train/Demo_2024051518/labels-train.csv...

```

圖 40 data_prepare 程式輸出

4. SlowFast 模型訓練

- (1) 參數設定：將設定的參數傳遞給 train.py 中的 train 函式來實際進行模型的訓練以獲得模型權重，如圖 41 所示。

```
75      # -----
76      # training - slowfast
77      #
78      ...
79      args:
80          project: 專案名稱
81          input: 資料集路徑
82          output: 儲存權重路徑，預設為models
83          weight: 初始模型權重，預設為models/action/init.pth
84          test_ratio: 訓練-驗證比例，預設為0.2
85          batch_size: batch size，預設為2
86          lr: learning rate，預設為0.0001
87          epoch: epoch，預設為20
88          seed: fix random seed，預設為666
89          ...
90
91      print("-----training slowfast model-----")
92      train(project=project_name,
93             input=temp_train,
94             output=action_path,
95             weight=init_action_weights,
96             test_ratio=0.2,
97             batch_size=2,
98             lr=0.0001,
99             epoch=20,
100            seed=666)
```

圖 41 參數設定

- (2) 程式執行：載入 PyTorchVideo 模型庫中的 slowfast_r50 預訓練模型進行訓練。

- (3) 程式輸出

```
-----training slowfast model-----
[2024-05-15 18:52:31,182][train.py][line:64][INFO] project: Demo_2024051518
input: /E/Aver/temp_train/Demo_2024051518
output: /E/Aver/models/action
init_weights: models/action/init.pth
test_ratio: 0.2
batch_size: 64
lr: 0.0001
epoch: 20
seed: 666
device: cuda:0
[1. 1. 1. 1. 1. 1.]
[2024-05-15 18:52:34,132][train.py][line:77][INFO] Loading `slowfast_r50` model...
Using cache found in /root/.cache/torch/hub/facebookresearch_pytorchvideo_main
[2024-05-15 18:52:37,955][train.py][line:89][INFO] action_list: {'0': 'Assemble_3.1', '1': 'Assemble_3.2', '2': 'Assemble_3.3', '3': 'Assemble_3.4', '4': 'Assemble_3.5', '5': 'Brush', '6': 'ScrewUp', '7': 'Other'}
[2024-05-15 18:52:37,956][train.py][line:90][INFO] num_act: 7
[2024-05-15 18:52:37,964][train.py][line:91][INFO] trainloader: 79, validloader: 20
    %
[2024-05-15 18:52:37,966][train.py][line:98][INFO] ===Epoch 1 ===
[2024-05-15 18:53:10,710][train.py][line:121][INFO] Epoch: 1, Step: 1, loss: 1.869506
[2024-05-15 18:53:36,101][train.py][line:121][INFO] Epoch: 1, Step: 2, loss: 2.016680
[2024-05-15 18:53:46,851][train.py][line:121][INFO] Epoch: 1, Step: 3, loss: 2.029409
[2024-05-15 18:53:58,251][train.py][line:121][INFO] Epoch: 1, Step: 4, loss: 2.105729
[2024-05-15 18:54:10,410][train.py][line:121][INFO] Epoch: 1, Step: 5, loss: 2.016207
[2024-05-15 18:54:20,649][train.py][line:121][INFO] Epoch: 1, Step: 6, loss: 2.387063
[2024-05-15 18:54:32,492][train.py][line:121][INFO] Epoch: 1, Step: 7, loss: 1.974684
[2024-05-15 18:54:42,641][train.py][line:121][INFO] Epoch: 1, Step: 8, loss: 1.629867
[2024-05-15 18:54:56,025][train.py][line:121][INFO] Epoch: 1, Step: 9, loss: 2.033224
[2024-05-15 18:55:05,857][train.py][line:121][INFO] Epoch: 1, Step: 10, loss: 1.913405
```

圖 42 data_prepare 程式輸出

5. Filter 模型訓練

- (1) 參數設定：透過 glob 函數來獲取指定資料夾下的影片檔案，程式調用 get-filter.py 模中的 get-filter 函式並傳遞相應的參數，其主要目的是計算適用的濾波器權重，如圖 43 所示。

```
102     # -----
103     # training - filter: detect
104     # -----
105     ...
106     args:
107         input: 輸入影片檔
108         output: 結果存放之資料夾路徑
109         reference: 參照資料夾，包含 [classes.txt] 和 [labels-action.csv]
110         action_weights: 動作辨識權重檔路徑(slowfast)
111         save: filter權重儲存路徑,
112         interval: inference interval (frames), 預設為25
113         imsize: inference size (pixels), 預設為256
114         device: cuda device, i.e. 0 or 0,1,2,3 or cpu
115         ...
116         print("-----calculating filter-----")
117         files = glob(os.path.join(input, "videos", "*.mp4"))
118         print(f"files: {files}")
119         for file_ in files:
120             get_filter(input=file_,
121                         output=temp_filter,
122                         reference=temp_train,
123                         action_weights=trained_action_weights,
124                         save=trained filter weights)
```

圖 43 Filter 參數設定

- (2) 程式執行：使用訓練好的 SlowFast 模型對影片進行動作預測，並根據所計算出的模型表現，輸出各動作標籤之信心分數濾波值。

(3) 程式輸出

```
-----calculating filter-----
files: ['/E/Aver/input/Demo/videos/0912_am_3th.mp4']
video_model = torch.load('/E/Aver/models/action/Demo_2024051518.pth'
processing...
total frames: 14183
Processing Video: 14183frame [2:12:26,  1.03s/frame]total cost: 7947.671 s, video length: 472.73333333333335 s
Processing Video: 14183frame [2:12:27,  1.78frame/s]
(base) root@209fed2c06f4:/E/Aver# cd /E/Aver[]
```

圖 44 data_prepare 程式輸出

(四) 動作辨識

動作辨識環節將整合動作辨識模型訓練中之重要參數，作業員偵測權重、動作辨識模型權重、過濾器權重，基於 YOLO v7 及 SlowFast 模型進行動作辨識，預測影片每幀動作，並輸出標籤。

使用'argparse'模組來定義命令行參數，這些參數包括專案名稱、輸入和輸出資料夾、模型權重文件和類別文件的路徑等，並將解析的參數賦值給變數，獲取當前工作目錄的路徑。根據是否提供了相應的權重和類別文件路徑來決定使用默認值還是用戶提供的值，並切換當前工作目錄到'YOLO v7'目錄，如圖 45 所示。

```
9  parser = argparse.ArgumentParser()
10 parser.add_argument("--project", type=str, default="project", help="專案名稱")
11 parser.add_argument("--input", type=str, default="../input/videos", help="輸入資料夾，包含欲預測之影片檔")
12 parser.add_argument("--output", type=str, default="../outputs", help="輸出資料夾，根據專案名稱自動生成")
13 parser.add_argument("--operator-weights", type=str, default="../models/operator/init.pt", help="預設使用yolo模型")
14 parser.add_argument("--action-weights", type=str, default="", help="預設使用slowfast模型，若無則根據專案名稱自動抓取")
15 parser.add_argument("--filter-weights", type=str, default="", help="預設使用filter，若無則根據專案名稱自動抓取")
16 parser.add_argument("--classes", type=str, default="", help="類別檔路徑，包含classes.txt，類別數量需相符，若無則根據")
17 parser.add_argument('--conf-thres', type=float, default=0.5, help='object confidence threshold')
18 args = parser.parse_args()
19
20 input, output = args.input, args.output
21 project, operator_weights, action_weights, filter_weights, classes = \
22     args.project, args.operator_weights, args.action_weights, args.filter_weights, args.classes
23
24 path_ = os.getcwd()
25
26 if action_weights == "":
27     action_weights = os.path.join(path_, 'models', 'action', f'{project}.pth')
28 if filter_weights == "":
29     filter_weights = os.path.join(path_, 'models', 'filter', f'{project}.txt')
30 if classes == "":
31     classes_path = os.path.join(path_, 'temp_train', project)
32 else:
33     classes_path = os.path.join(path_, classes)
34
35 # 辨識動作
36 os.chdir("yolov7")
37 print(os.getcwd())
```

圖 45 動作辨識程式碼一

glob 函數找尋指定路徑下的所有 .mp4 形式的影像檔案，搜尋結果將儲存在 videos 變量中並打印。並調用 slowfast-yolo-detect.py 腳本，將使用該腳本整合權重及模型，如圖 46 所示。

```
42     videos = glob(os.path.join(path_, input, "*.mp4"))
43     print(f"files: {videos}")
44
45     detect = "slowfast-yolo-detect.py"
```

圖 46 動作辨識程式碼二

使用迴圈遍歷 videos 列表中的每一個影像，每個影像都會構建一組參數，這些參數指定了輸入影像檔案、輸出目錄、作業員偵測權重、動作辨識模型權重、過濾器權重等。使用 subprocess.run 函數調用 slowfast-yolo-detect.py 腳本，將構建好的參數傳遞給該腳本執行動作辨識，並輸出標籤檔，再依據該腳本回傳識別結果判斷執行是否執行成功，如圖 47 所示。

```

45 detect = "slowfast-yolo-detect.py"
46
47 for video in videos:
48     args = [
49         "--input", video,
50         "--output", output,
51         "--action-weights", action_weights,
52         "--detect-weights", operator_weights,
53         "--filter-weights", filter_weights,
54         "--classes-path", classes_path,
55         '--conf-thres', "0.5"
56     ]
57     print(args)
58     result = subprocess.run(
59         ["python", detect] + args, capture_output=True, text=True
60     )
61
62     if result.returncode == 0:
63         print("程式執行成功！")
64         print("輸出訊息：", result.stdout)
65     else:
66         print("程式執行失敗！")
67         print("錯誤訊息：", result.stderr)
68
69 os.chdir("../")
70 print(os.getcwd())

```

圖 47 動作辨識程式碼三

影像經由動作辨識模型預測將生成預測結果及影像，預測結果中將包含當前幀數，YOLO v7 判斷作業員是否在執行有效動作，及該判斷之信心分數；SlowFast 判斷作業員動作為何及其信心分數，如圖 48、圖 49 所示。

	yolo	conf_yolo	slowfast	conf_slowfast
12	1	0	7	0
13	1	0	7	0
14	1	0	7	0
15	0	0.55	7	0
16	0	0.53	7	0
17	0	0.64	7	0
18	0	0.7	7	0
19	0	0.79	7	0
20	0	0.74	7	0
21	0	0.77	7	0
22	0	0.84	7	0
23	0	0.81	7	0
24	0	0.82	0	0.406469
25	0	0.8	0	0.406469
26	0	0.79	0	0.406469
27	0	0.78	0	0.406469
28	0	0.78	0	0.406469
29	0	0.82	0	0.406469
30	0	0.84	0	0.406469
31	0	0.81	0	0.406469
32	0	0.76	0	0.406469
33	0	0.77	0	0.406469

圖 48 預測結果

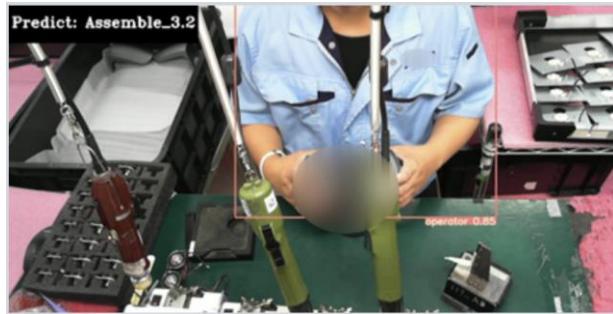


圖 49 預測影像
(為保護公司機密資料故將影像作模糊處理)

(五) 模型評估

評估環節將針對動作辨識環節所判斷動作，以正確人工貼標籤作對比，將採用精確率（Precision）與召回率(Recall)作為模型衡量指標，並生成混淆矩陣，視覺化衡量指標。精確率為預測為陽性的樣本中，其中部分是預測正確的；召回率為事實為真的樣本中，部分是預測正確的。

主程式使用 argparse 套件調用解析參數，輸入正確每幀動作標籤檔案，與讀取模型判斷每幀之動作標籤檔案，並調用 metrics 腳本計算模型衡量指標，如圖 50 所示。

```

8 v if __name__ == '__main__':
9   parser = argparse.ArgumentParser()
10  parser.add_argument("--input", type=str, default="outputs/results/0912_am_3th.csv", help="輸入檔案")
11  parser.add_argument("--reference", type=str, default="input/Demo", help="輸入資料夾")
12  args = parser.parse_args()
13
14  input, reference = args.input, args.reference
15
16 v metrics(input_path=input,
17           ref_path='temp_test',
18           interval=25,
19           no_log=True)

```

圖 50 模型評估主程式碼

Metrics 腳本中 metrics 函數，導入訓練模型的標籤即為正確版標籤，與模型預測的標籤，並設定每 25 幀計算一次。而根據輸入文件名生成實驗名稱，創建輸出目錄，如圖 51 所示。

```

111  def metrics(input_path='input',
112      ref_path='temp_train',
113      interval=25,
114      no_log=True):
115  """
116      args:
117          input_path (預設值: 'input'): 輸入資料的路徑。CSV檔案應位於此路徑下。
118          ref_path (預設值: 'temp_train'): 參考資料的路徑，用於取得類別標籤、真實標籤等。
119          interval (預設值: 25): 用於控制計算結果時取樣的間隔，例如每 N 帖計算一次。
120          no_log (預設值: False): 一個布林值，如果設定為 True 則不會記錄任何日誌。
121
122      exp_name = os.path.basename(input_path).replace('.csv', '')
123      output_path = os.path.join('outputs', 'metrics', exp_name)
124      os.makedirs(output_path, exist_ok=True)
125
126      # 定義資料標籤
127      cls2idx_list, idx2cls_list = cls_list(ref_path)
128      other_idx = idx2cls_list['Other']

```

圖 51 模型評估 Metrics 程式碼一

讀取真實標籤和預測結果，將它們合併到一個 DataFrame 中。將預測動作標籤、正確標籤、預測是否正確等資料存入，並確保預測標籤與真實標籤一致。將檔案合併為 result.csv 輸出存檔，如圖 52 所示。

```

130  # 合併預測結果與真實標籤
131  results = pd.DataFrame()
132
133  # true label
134  file_label = get_trues(ref_path)
135  file_pred = pd.read_csv(input_path)
136  video_name = os.path.basename(input_path).replace('.csv', '')
137
138  preds = file_pred.loc[:, 'slowfast'].values
139  trues = file_label.label.values
140  result_ = pd.DataFrame({'video': video_name,
141                          'frame': file_pred.index,
142                          'preds': preds,
143                          'trues': trues,
144                          'correct': preds==trues,
145                          'isCount':True})
146  results = pd.concat([results, result_], axis=0)
147
148  # 確認標籤是否對應
149  label_trues = sorted(results.trues.unique())
150  labels_preds = sorted(results.preds.unique())
151  labels = sorted(list(set(list(results.preds.unique()) + list(results.trues.unique()))))
152  labels = idx2cls(labels, cls2idx_list)
153
154  # 存檔
155  results.to_csv(os.path.join(output_path, f'result.csv'), index=False)

```

圖 52 模型評估 Metrics 程式碼二

依據 interval 參數設定間隔為每 25 帀判斷一次，調用 plot_cm_recall 和 plot_cm_precision 函數繪製精確率與召回率的混淆矩陣，建立可視化圖形，配置日誌訊息紀錄，並將結果返回輸出，存至主程式指定路徑，如圖 53 所示。

```

160     results['isCount'] = results.frame
161     results['isCount'] = results.isCount.apply(lambda x: 1 if (x+1) % interval == 0 else 0)
162     results_n = results[results.isCount==1]
163     results_wo = results_n[results_n.trues!=int(other_idx)]
164     # results_wo.to_csv(os.path.join(output_path, f'{exp_name}_result_interval.csv'), index=False)
165
166     if no_log:
167         # -----
168         # 畫混淆矩陣
169         # -----
170         plot_cm_recall(results_n, labels, output_path, title='Other')
171         plot_cm_precision(results_n, labels, output_path, title='Other')
172
173         # -----
174         # 紀錄log
175         # -----
176         logger = get_logger(os.path.join(output_path, 'log-test.txt'))
177         logger.info(f'cls2idx_list: {cls2idx_list}')
178         logger.info(f'idx2cls_list: {idx2cls_list}')
179         logger.info(f'label_trues: {label_trues}')
180         logger.info(f'labels_preds: {labels_preds}')
181         logger.info(f'labels: {labels}')
182
183         logger.info(f'data size: {results_n.shape[0]}')
184         logger.info(results_n.correct.sum() / results_n.shape[0])
185         logger.info(classification_report(results_n.trues, results_n.preds))
186         print(f'log file saved to {os.path.join(output_path, "log-test.txt")}')
187
188     return results_wo

```

圖 53 模型評估 Matrics 程式碼三

動作辨識預測標籤經由模型評估程式碼，建立精確率與召回率的混淆矩陣，如圖 54 所示。

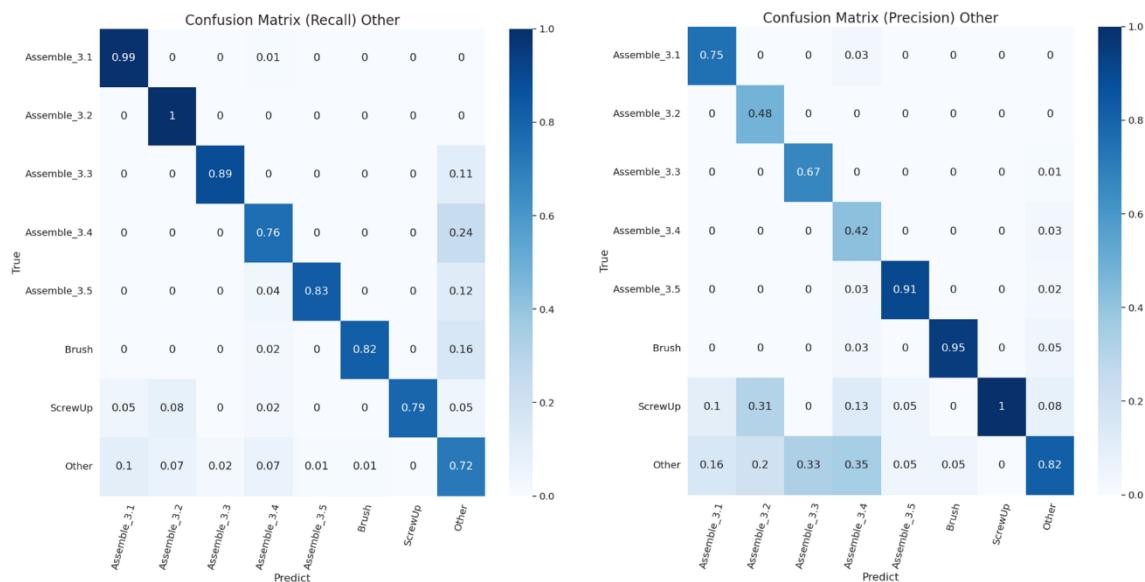


圖 54 精確率與召回率的混淆矩陣

(六) 生成流程圖

在此過程能將辨識出來的動作，進一步生成流程程序圖，並計算出整個組裝動作循環週期(Cycle Time)，以及平均組裝動作循環週期(Average Cycle Time)。在此我們分為兩種方法去求得動作循環周期，去探討這兩種生成方法對於計算組裝動作循環時間是否有影響：

1. 終點導向循環週期計算法：由於發現每次組裝流程的最後一個動作會是 Brush 因此計算組裝動作循環週期的方式是，當辨識完 Brush 這個動作後，會計算一次，最後會藉由每次的組裝動作循環週期，進一步計算出平均組裝動作循環週期。
2. 全程導向循環週期計算法：每次的組裝流程中，每個動作都會出現至少一次，因此當發現每個動作都至少出現一次後，才會計算出組裝動作循環週期。

下面將這兩種方式的程式碼(兩者程式碼主要的不同是在下面第五步驟的地方)及生成出來的流程程序圖做說明：

- (1) 讀取檔案：將欲算出組裝動作循環時間的檔案讀入程式中。

```
1 import pandas as pd  
2  
3 # 讀取檔案  
4 data = pd.read_csv("0912_am_3th.csv")  
5
```

圖 55 讀取檔案程式碼

- (2) 處理讀進來的檔案：把之後不會用到的欄位刪除。

```
6 # 刪除不必要的欄位  
7 data.drop(columns=['newBox'], inplace=True)  
8
```

圖 56 處理讀進來的檔案

- (3) 首先計算以幀為單位的影片進一步算出，一幀占幾秒：由影片得知影片時長為 7 分 52 秒，或 472 秒，總共 14183 幀，因此計算出一幀大約為 0.03328 秒。

```
9 # 影片長度與每幀占幾秒  
10 video_min = 7  
11 video_sec = 52  
12 video_total_sec = video_min * 60 + video_sec  
13 frame_sec = video_total_sec / len(data['Frame'])  
14 print(frame_sec)  
15
```

圖 57 計算出一幀佔幾秒

(4) 接下來計算動作出現的時間：計算影片中每個動作持續的秒數，並儲存在輸出的表格。

```
16 # 影片標籤動作種類
17 action_list = ["Assemble_3.1", "Assemble_3.2", "Assemble_3.3", "Assemble_3.4",
18 | | | | "Assemble_3.5", "Brush", "ScrewUp", "Other"]
19
20 # 建構輸出的dataframe
21 output_df = pd.DataFrame(columns=['Action', 'Duration', 'Accumulative_Time'])
22
23 # 計算每次動作持續時間
24 duration = 0
25 Type = data['Action'][0]
26 total_time = 0
27
28 for action in range(len(data['Action'])):
29     if data['Action'][action] == Type:
30         duration += 1
31     else:
32         time = round(duration * frame_sec,2)
33         total_time += time
34         output_df = pd.concat([output_df,
35                                pd.DataFrame({'Action': [Type], 'Duration': [time],
36                                              'Accumulative_Time': [round(total_time, 2)]})],
37                                ignore_index=True)
38         duration = 1
39         Type = data['Action'][action]
40
41 time = round(duration * frame_sec,2)
42 total_time += time
43 output_df = pd.concat([output_df, pd.DataFrame({'Action': [Type], 'Duration': [time],
44                                              'Accumulative_Time': [round(total_time, 2)],
45                                              'Cycle_Time':None})], ignore_index=True)
```

圖 58 計算動作出現的時間

(5) 計算組裝時間週期：進一步計算組裝動作循環週期。

a. 終點導向循環週期計算法：當讀取到 Brush 這個動作時，計算出一次組裝動作循環週期。

```
48 # 計算 Cycle Time
49 cycle_list = []
50 cycle_time = 0
51 i = 0
52 while i < len(output_df['Action']):
53     if(output_df['Action'][i] == 'Brush'):
54         cycle_time += output_df['Duration'][i]
55         output_df['Cycle_Time'][i] = round(cycle_time, 2)
56         cycle_list.append(cycle_time)
57         cycle_time = 0
58     # 計算出每個 Cycle Time 空一行
59     output_df = pd.concat([output_df.iloc[:i+1], pd.DataFrame([{}]), output_df.iloc[i+1:]]).reset_index(drop=True)
60     i+=2
61     else:
62         cycle_time += output_df['Duration'][i]
63         i+=1
64
```

圖 59 計算組裝時間週期

b. 全程導向循環週期計算法：每個動作都至少出現一次後計算一次組裝動作循環週期。

```
48 # 計算 cycle time
49 cycle_list = []
50 cycle_time = 0
51 i = 0
52 Assemble_3_1 = 0
53 Assemble_3_2 = 0
54 Assemble_3_3 = 0
55 Assemble_3_4 = 0
56 Assemble_3_5 = 0
57 Brush = 0
58 ScrewUp = 0
59 Other = 0
60 while i < len(output_df['Action']):
61     cycle_time += output_df['Duration'][i]
62
63     # 檢查當前動作並增加相應計數檢查
64     action = output_df['Action'][i]
65     if action == 'Assemble_3.1':
66         Assemble_3_1 += 1
67     elif action == 'Assemble_3.2':
68         Assemble_3_2 += 1
69     elif action == 'Assemble_3.3':
70         Assemble_3_3 += 1
71     elif action == 'Assemble_3.4':
72         Assemble_3_4 += 1
73     elif action == 'Assemble_3.5':
74         Assemble_3_5 += 1
75     elif action == 'ScrewUp':
76         ScrewUp += 1
77     elif action == 'Brush':
78         Brush += 1
79     elif action == 'Other':
80         Other += 1
81
82     all_actions_present = (
83         Assemble_3_1 != 0 and Assemble_3_2 != 0 and Assemble_3_3 != 0 and
84         Assemble_3_4 != 0 and Assemble_3_5 != 0 and ScrewUp != 0 and
85         Brush != 0 and Other != 0
86     )
87
88     if all_actions_present:
89         output_df['Cycle_Time'][i] = round(cycle_time, 2)
90         cycle_list.append(cycle_time)
91         cycle_time = 0
92
93     output_df = pd.concat([output_df.iloc[:i+1], pd.DataFrame([{}]), output_df.iloc[i+1:]]).reset_index(drop=True)
94     i += 2
95
96     Assemble_3_1 = 0
97     Assemble_3_2 = 0
98     Assemble_3_3 = 0
99     Assemble_3_4 = 0
100    Assemble_3_5 = 0
101    Brush = 0
102    ScrewUp = 0
103    Other = 0
104    else:
105        i += 1
106 #print(output_df.to_string())
107
```

圖 60 全程導向循環週期計算法

(6) 計算平均組裝動作循環週期：最後藉由每一次的組裝動作循環週期算出平均組裝動作循環週期。

```
48 # 計算 Cycle Time
49 cycle_list = []
50 cycle_time = 0
51 i = 0
52 while i < len(output_df['Action']):
53     if(output_df['Action'][i] == 'Brush'):
54         cycle_time += output_df['Duration'][i]
55         output_df['Cycle_Time'][i] = round(cycle_time, 2)
56         cycle_list.append(cycle_time)
57         cycle_time = 0
58     # 計算出每個 Cycle Time 空一行
59     output_df = pd.concat([output_df.iloc[:i+1], pd.DataFrame([{}]), output_df.iloc[i+1:]]).reset_index(drop=True)
60     i+=2
61 else:
62     cycle_time += output_df['Duration'][i]
63     i+=1
64
```

圖 61 計算平均組裝動作循環週期

(7) 輸出結果到 csv 檔：將最終表格的結果輸出成 csv 檔

```
73 # 輸出 csv
74 output_df.to_csv('action_process_chart_real.csv', index=False)
```

圖 62 輸出結果到 csv 檔

結果的欄位名稱分成動作(Action)、動作持續時間(Duration)、累積時間(Accumulative Time)及組裝動作循環週期(Cycle Time)，在檔案的最後會有平均組裝動作循環週期，表 2 為兩種計算方式輸出的結果，可以得知兩者的結果完全相同，可以推論兩計算規則是合理的。以下部分圖表所出現貼標欄位的命名方式為：T(Tag)標籤數量-生成流程程序圖所使用的計算方法，例 T8-1 則代表標籤數量八種使用終點導向循環週期計算法。

表 2 為終點導向循環週期計算法及全程導向循環週期計算法的結果

終點導向循環週期計算法結果：

Action	Duration	Accumulative Time	Cycle Time
Other	5.99	5.99	
Assemble_3.1	11.58	17.57	
Other	0.5	18.07	
ScrewUp	31.75	49.82	
Other	2.6	52.42	
Assemble_3.2	4.39	56.81	
Other	0.7	57.51	
ScrewUp	5.69	63.2	
Other	5.89	69.09	
Assemble_3.3	1.8	70.89	
Assemble_3.4	3.99	74.88	
Other	0.4	75.28	
ScrewUp	5.39	80.67	
Assemble_3.5	4.69	85.36	
Other	1	86.36	
Brush	10.48	96.84	96.84
Other	11.28	108.12	
Assemble_3.1	11.08	119.2	
Other	0.5	119.7	
ScrewUp	29.45	149.15	
Other	4.69	153.84	
Assemble_3.2	4.19	158.03	
Other	0.7	158.73	
ScrewUp	7.92	166.65	
Other	3.99	170.64	
Assemble_3.3	2.06	172.7	
Assemble_3.4	1.43	174.13	
Other	0.5	174.63	
ScrewUp	4.09	178.72	
Assemble_3.5	4.99	183.71	
Other	1	184.71	
Brush	8.89	193.6	96.76
Other	15.08	208.68	
Assemble_3.1	11.05	219.73	
Other	0.5	220.23	
ScrewUp	26.16	246.39	
Other	7.29	253.68	
Assemble_3.2	3.89	257.57	
Other	0.6	258.17	
ScrewUp	6.59	264.76	
Other	6.89	271.65	
Assemble_3.3	1.83	273.48	
Assemble_3.4	3.36	276.84	
Other	0.6	277.44	
ScrewUp	8.99	286.43	
Assemble_3.5	3.99	290.42	
Other	0.9	291.32	
Brush	10.52	301.84	108.24
Other	10.18	312.02	
Assemble_3.1	29.75	341.77	
Other	0.4	342.17	
ScrewUp	33.75	375.92	
Other	4.49	380.41	
Assemble_3.2	9.68	390.09	
Other	0.6	390.69	
ScrewUp	6.49	397.18	
Other	4.69	401.87	
Assemble_3.3	2.46	404.33	
Assemble_3.4	5.62	409.95	
Other	0.6	410.55	
ScrewUp	4.89	415.44	
Assemble_3.5	5.89	421.33	
Other	0.8	422.13	
Brush	7.99	430.12	128.28
Other	41.87	471.99	
Average Cycle time	107.53		

全程導向循環週期計算法結果：

Action	Duration	Accumulative Time	Cycle Time
Other	5.99	5.99	
Assemble_3.1	11.58	17.57	
Other	0.5	18.07	
ScrewUp	31.75	49.82	
Other	2.6	52.42	
Assemble_3.2	4.39	56.81	
Other	0.7	57.51	
ScrewUp	5.69	63.2	
Other	5.89	69.09	
Assemble_3.3	1.8	70.89	
Assemble_3.4	3.99	74.88	
Other	0.4	75.28	
ScrewUp	5.39	80.67	
Assemble_3.5	4.69	85.36	
Other	1	86.36	
Brush	10.48	96.84	96.84
Other	11.28	108.12	
Assemble_3.1	11.08	119.2	
Other	0.5	119.7	
ScrewUp	29.45	149.15	
Other	4.69	153.84	
Assemble_3.2	4.19	158.03	
Other	0.7	158.73	
ScrewUp	7.92	166.65	
Other	3.99	170.64	
Assemble_3.3	2.06	172.7	
Assemble_3.4	1.43	174.13	
Other	0.5	174.63	
ScrewUp	4.09	178.72	
Assemble_3.5	4.99	183.71	
Other	1	184.71	
Brush	8.89	193.6	96.76
Other	15.08	208.68	
Assemble_3.1	11.05	219.73	
Other	0.5	220.23	
ScrewUp	26.16	246.39	
Other	7.29	253.68	
Assemble_3.2	3.89	257.57	
Other	0.6	258.17	
ScrewUp	6.59	264.76	
Other	6.89	271.65	
Assemble_3.3	1.83	273.48	
Assemble_3.4	3.36	276.84	
Other	0.6	277.44	
ScrewUp	8.99	286.43	
Assemble_3.5	3.99	290.42	
Other	0.9	291.32	
Brush	10.52	301.84	108.24
Other	10.18	312.02	
Assemble_3.1	29.75	341.77	
Other	0.4	342.17	
ScrewUp	33.75	375.92	
Other	4.49	380.41	
Assemble_3.2	9.68	390.09	
Other	0.6	390.69	
ScrewUp	6.49	397.18	
Other	4.69	401.87	
Assemble_3.3	2.46	404.33	
Assemble_3.4	5.62	409.95	
Other	0.6	410.55	
ScrewUp	4.89	415.44	
Assemble_3.5	5.89	421.33	
Other	0.8	422.13	
Brush	7.99	430.12	128.28
Other	41.87	471.99	
Average Cycle time	107.53		

肆、研究討論與結果

一、研究討論

研究初期，因模型訓練需佔用大量儲存空間，故我們將部分模型訓練參數調低。模型訓練程式中之 train data 參數為選擇樣本量，在訓練模型的每次迭代下對所有提取的幀或特徵中，預設隨機選擇 600 樣本作為訓練數據。我們考慮儲存空間及電腦效能將參數降為 200，從訓練模型及投入影片動作辨識時間約在 4 小時，占用約 100GB 儲存空間。在這樣的模型訓練狀況下進行動作辨識，會得到以下圖 63 之精確率與召回率，發現甚至對於部分動作會有無法辨識之情形。

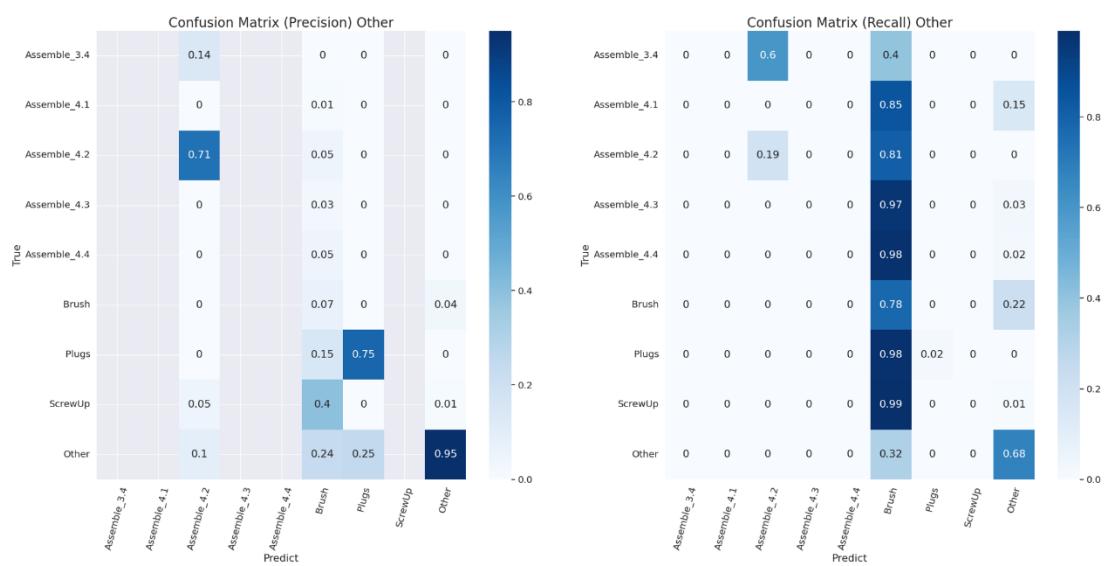


圖 63 train data 為 200 時的精確率與召回率

故我們將 train data 提高至 400，從訓練模型及投入影片動作辨識時間約在 9 小時，占用約 200GB 儲存空間。仍會發現對部分動作仍無法辨識，但整體模型有變好的趨勢。故訓練模型階段的 train data 參數最終的提高至 600，從訓練模型及投入影片動作辨識時間約在 13 小時，占用約 300GB 儲存空間。基於最新的模型對影像進行動作預測並評估得到以下圖 64 之精確率與召回率。精確率平均為 74%，召回率平均為 0.85。我們也發現模型對例外動作的學習與判定較為不准會容易導致與其他動作誤判，造成其他動作辨識度降低。於是我們將對例外動作的學習進行改善處理。

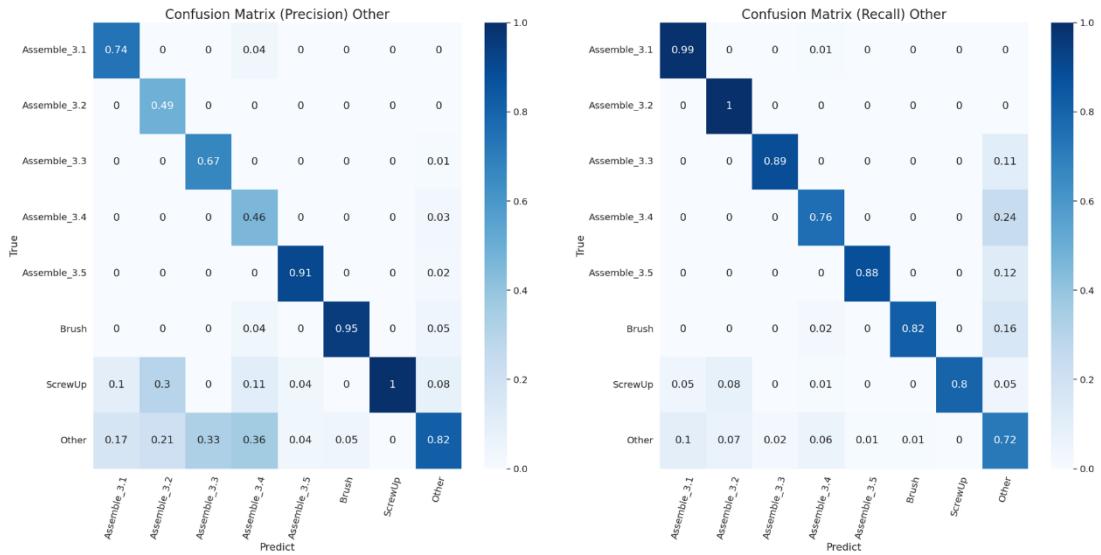


圖 64 train data 為 400 時的精確率與召回率

評估完生成流程程序圖如圖 65，我們也發現由於模型是以每秒 30 帧在辨識動作，在部分連續動作中的幾帧內可能會出現判定為其他動作，但在人因工程的角度下，人體是無法做到在 1 秒內連續從 A 動作切換為 B 再切換回 A 動作的，我們也將處理這樣的雜訊，同時也有助於提升流程程序圖的判斷準確度。

	A	B	C
1	Action	Action Duration	Accumulative_Time
2	Other	0.8	0.8
3	Assemble	2.5	3.3
4	Other	1.66	4.96
5	Assemble	13.31	18.27
6	ScrewUp	16.64	34.91
7	Assemble	0.83	35.74
8	ScrewUp	10.82	46.56
9	Other	0.83	47.39
10	ScrewUp	2.5	49.89
11	Other	2.5	52.39
12	Assemble	4.99	57.38
13	Other	0.83	58.21
14	Assemble	0.83	59.04
15	ScrewUp	0.83	59.87
16	Other	0.83	60.7

圖 65 生成的流程程序圖

二、研究改善

於研究討論中我們發現，例外動作因涵蓋過多種動作，會導致模型誤判造成整體衡量指標降低，動作辨識會有雜訊出現，影響流程程序圖，因此我們將著手改善研究。

(一)例外動作誤判

我們推測是由於例外動作內涵蓋眾多不同類型之動作，使模型對其辨識度較差，且難以學習而導致，故我們將無效動作 Other 拆解得更細，將拆分為 Both hands(雙手閒置)、Left hand(左手完全閒置右手移動)、Right hand(右手完全閒置左手移動)、Other(人員離開)，加上原先 7 種標籤共 11 種標籤之標籤貼法。同時我們也嘗試另一種標籤貼法為僅有 Screw Up、Brush、Assemble、Other 4 種標籤，原版標籤具有 8 種。

在 11 種標籤下，所建立模型並對影像進行動作預測並評估得到以下圖 66 之精確率與召回率。精確率平均為 72%，召回率平均為 68%。雖低於原版精確率平均為 74%，召回率平均為 85%。但我們發現原版標籤是 Other 與有效動作間會互相影響。在改成 11 種標籤下，僅為無效動作間的不同標籤互相影響，降低模型學習無效動作對有效動作的影響。故移除無效動作計算衡量指標，原版精確率平均為 74%，召回率平均為 85%。11 種標籤下精確率平均為 84%，召回率平均為 85%，有效提升精確率 10%，且在後續生成流程程序圖階段已證實有效提升時間計算的準確度。

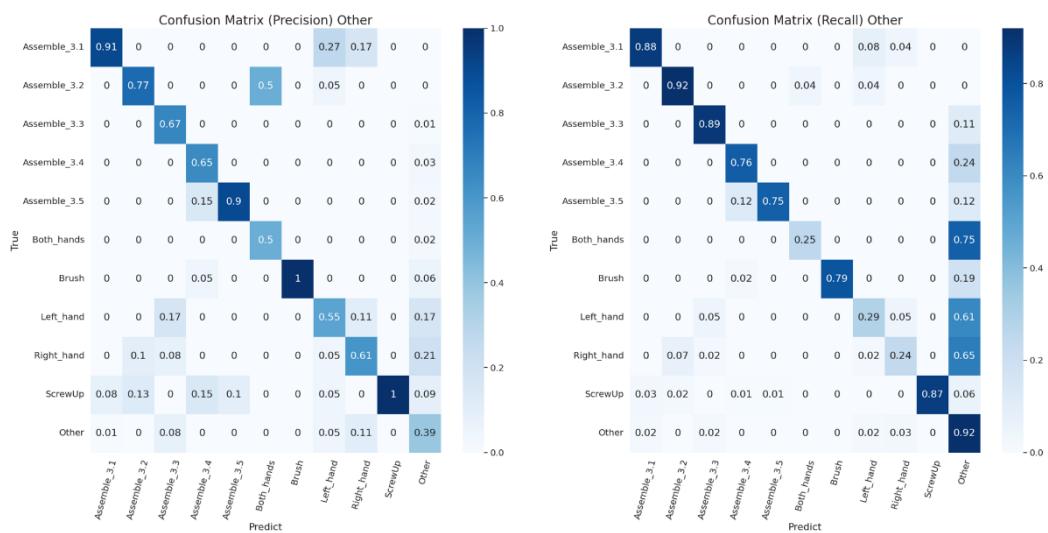


圖 66 11 種標籤下的精確率與召回率

在 4 種標籤下，所建立模型並對影像進行動作預測並評估得到以下圖 67 之精確率與召回率。精確率平均為 78%，召回率平均為 82%。雖然整體衡量指標提高，由於標籤種類過少，也導致資料不平衡的狀態發生。更無法有效判定流程程序圖間的不同動作，故不建議使用。

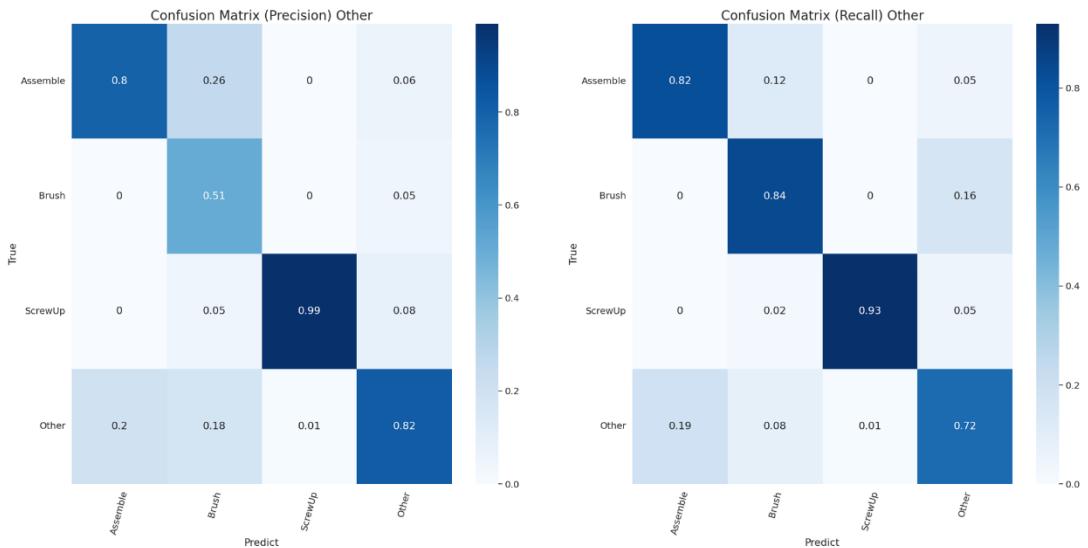


圖 67 4 種標籤下的精確率與召回率

(二)動作辨識雜訊影響流程程序圖

在上面的改善例外動作誤判實驗中有三種貼標方式：原版(8 種標籤)、複雜版(11 種標籤)、簡化版(4 種標籤)。在改善動作辨識雜訊影響生成流程程序圖中，我們調整生成流程程序圖之程式資料處理的部分(程式碼請見附錄)，由於模型在辨識上，偶爾會遇到雜訊，在連續相同動作中途的某幾幀辨識成其他動作，造成辨識上及人因工程角度中的不合理。如圖 68，由動作持續時間(Action Duration)欄位中可以發現在第 2、7、9、13、14、15、16 列中動作持續時間皆小於 1 秒，以組裝動作持續時間來看，這種時間對於動作本身可以被忽略不計，並且可以推論是辨識上的偏差造成這些動作被辨識出來，因此我們在計算組裝動作循環週期之前對該問題做處理，我們將這些小於 1 秒的時間併入後序動作中，做為組裝動作前的緩衝時間。

	A	B	C
1	Action	Action Duration	Accumulative_Time
2	Other	0.8	0.8
3	Assemble	2.5	3.3
4	Other	1.66	4.96
5	Assemble	13.31	18.27
6	ScrewUp	16.64	34.91
7	Assemble	0.83	35.74
8	ScrewUp	10.82	46.56
9	Other	0.83	47.39
10	ScrewUp	2.5	49.89
11	Other	2.5	52.39
12	Assemble	4.99	57.38
13	Other	0.83	58.21
14	Assemble	0.83	59.04
15	ScrewUp	0.83	59.87
16	Other	0.83	60.7

圖 68 受動作辨識訊雜訊影響的流程程序圖

根據上面資料處理進行完後，排除相同連續動作中遇到辨識的突然偏差，但相同的動作會因為排除了中間的偏差而導致重複的動作連在一起，如圖 69，由第 5、6、7 列有相同的動作和第 9、10 列及第 15、16 列相同的動作接連著出現，因此我們進一步處理資料，將它們合併在一起，就會得到圖 70，即為該操作員之一次組裝的流程程序圖的結果。

	A	B	C
1	Action	Duration	Accumulative_Time
2	Assemble_3.1	3.29	3.29
3	Other	1.66	4.95
4	Assemble_3.1	13.31	18.26
5	ScrewUp	16.64	34.9
6	ScrewUp	11.65	46.55
7	ScrewUp	3.33	49.88
8	Other	2.5	52.38
9	Assemble_3.2	4.99	57.37
10	Assemble_3.2	1.66	59.03
11	Other	1.66	60.69
12	Assemble_3.1	1.66	62.35
13	Other	5.82	68.17
14	Assemble_3.3	3.33	71.5
15	Assemble_3.4	4.16	75.66
16	Assemble_3.4	1.66	77.32

圖 69 重複動作相連的流程程序圖

	A	B	C
1	Action	Duration	Accumulative_Time
2	Assemble_3.1	3.29	3.29
3	Other	1.66	4.95
4	Assemble_3.1	13.31	18.26
5	ScrewUp	31.62	49.88
6	Other	2.5	52.38
7	Assemble_3.2	6.65	59.03
8	Other	1.66	60.69
9	Assemble_3.1	1.66	62.35
10	Other	5.82	68.17
11	Assemble_3.3	3.33	71.5
12	Assemble_3.4	5.82	77.32
13	ScrewUp	3.33	80.65
14	Assemble_3.5	4.99	85.64
15	Other	2.5	88.14
16	Brush	8.32	96.46

圖 70 操作員之一次組裝的流程程序圖的結果

三、研究結果

比較三種貼標方式藉由第三部分提到的兩種計算方法：組裝動作循環週期方法(終點導向循環週期計算法及全程導向循環週期計算法)，對照第三部份正確流程流程圖的數據是否有差距，因此針對以下問題做研究結果討論：我們最終將衡

量兩種計算組裝動作循環週期的方式是否與正確動作時間一樣是合理的是否有時間上的差異，進一步推斷這種貼標方式是否不夠好，或者並沒有差異？

表 3 為三種貼標方式藉由兩種計算方法得到的每次組裝動作循環週期、平均組裝動作循環週期及組裝動作循環的次數，並對照正確版的數據。影片正確僅執行 4 次組裝循環，根據表 3 的數據，我們可以清楚看到與其他貼標方式相比，簡化版的貼標方式在組裝動作循環週期次數，以及平均循環週期時間上存在著顯著差異。簡化版貼標方式在一開始的兩次組裝動作循環週期，與其他貼標方式差異並不明顯。然而隨著第三、第四次組裝流程時間的增加，不必要的動作時間也相應增長，導致模型在四個類別中選擇最有可能動作時的效果不佳，印證了簡化標籤由於標籤數過少，造成相同標籤涵蓋不同動作，無法有效區分和判斷流程程序圖。儘管簡化貼標最初的設計目的是為了提高效率，但數據結果卻說明它與預期的效果相去甚遠，而在其他貼標方式中，原版及複雜版和正確版的數據，差距微乎其微，運用兩種計算組裝動作循環週期的方式上也並沒有顯著影響，我們再根據這三個數據：正確版、原版、複雜版，藉由圖 71(組裝動作循環週期)、圖 72(有效動作時間)、圖 73(無效動作時間)做進一步的討論。

表 3 為各項數據在三種貼標方法經過兩種計算組裝動作循環週期得到的結果

貼標	1st Cycle Time	2nd Cycle Time	3rd Cycle Time	4th Cycle Time	5th Cycle Time	6th Cycle Time	7th Cycle Time	Average Cycle Time	# of Cycle Time
正確版	96.84	96.76	108.24	128.28	None	None	None	107.53	4
T8-1	96.46	96.51	109.01	127.29	None	None	None	107.32	4
T8-2	96.46	96.51	109.01	127.29	None	None	None	107.32	4
T11-1	96.49	96.50	109.02	127.28	None	None	None	107.32	4
T11-2	96.49	96.50	109.02	127.28	None	None	None	107.32	4
T4-1	96.49	99.83	66.57	39.11	7.49	101.50	18.30	61.33	7
T4-2	96.49	99.83	66.57	39.11	74.88	34.11	61.03	67.43	7

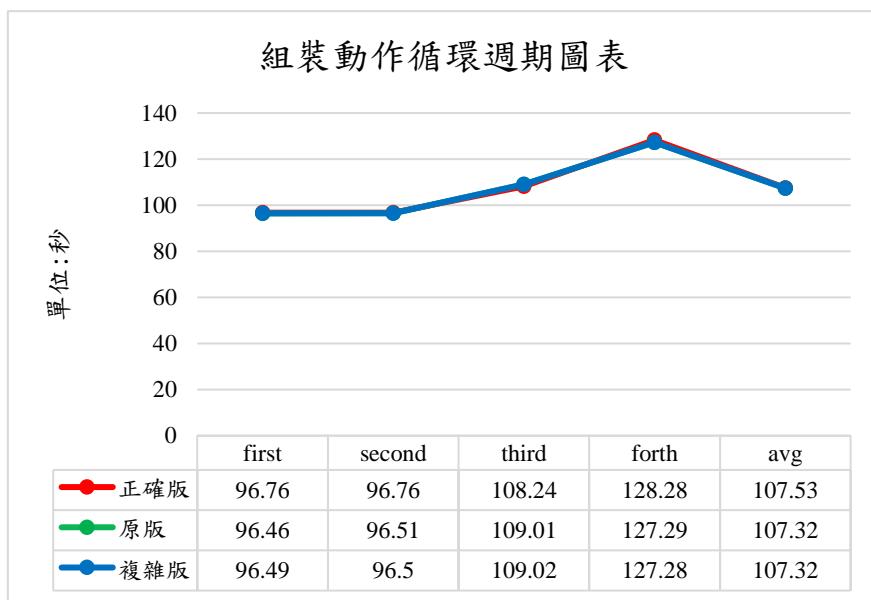


圖 71 組裝動作循環週期對照

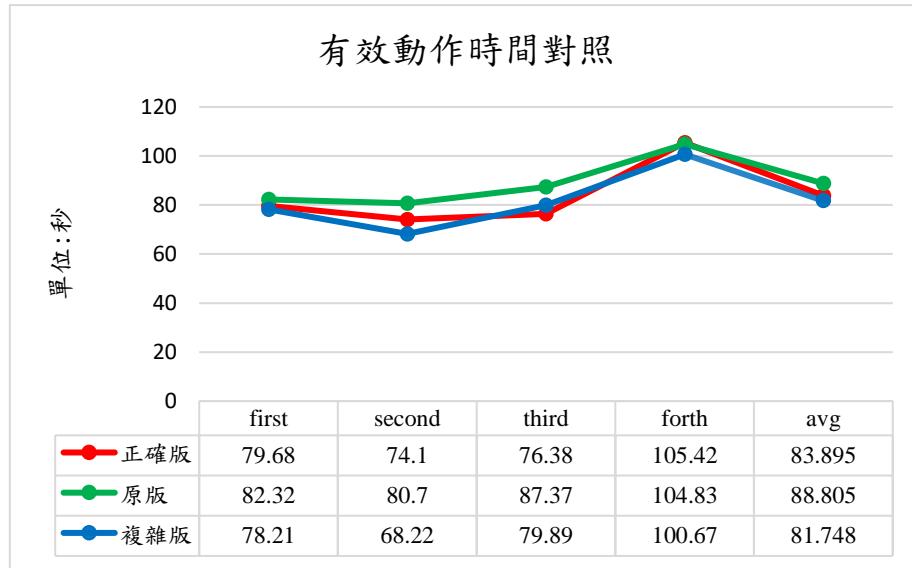


圖 72 有效動作時間對照

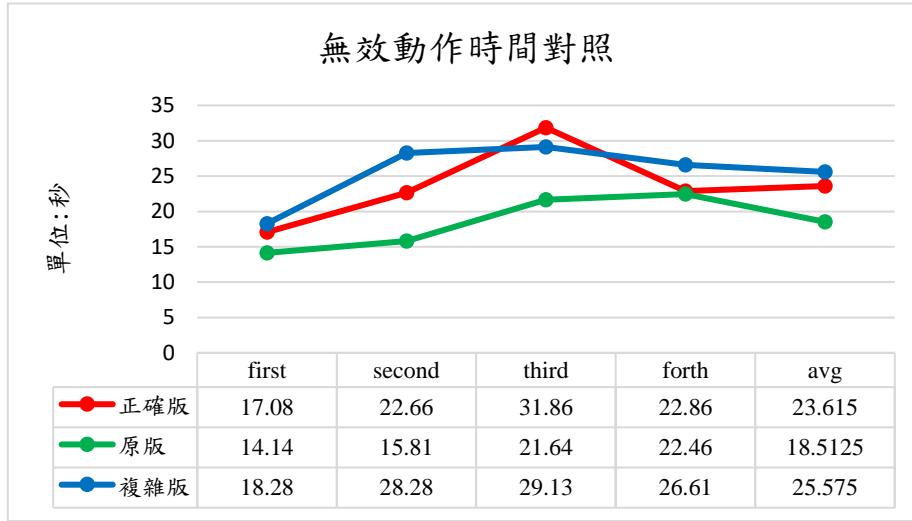


圖 73 無效動作時間對照

由圖 71 組裝動作循環週期對照的折線圖可以得知，三個版本無明顯的差異；圖 72 為有效動作時間對照，在此排除不必要動作的時間，結果發現複雜版在有效動作平均時間上與正確版的差距最小；而圖 73 為無效動作時間對照，比對辨識出的不必要動作的時間，結果發現複雜版比起原版在無效動作平均時間上與正確版的差距最小。

根據上述三張圖，我們從實驗推倒出結論：適度增加標籤複雜度，添加更多特徵信息，能夠有效提升計算機視覺模型對目標動作的檢測與識別能力，從而提高整體模型準確性，然而如果過度複雜化標籤可能會影響模型性能，也可能會導致數據標註人力和時間成本的增加，降低生產效率。因此，在實際應用中需權衡標籤複雜度與貼標效率之間的平衡，尋求一個最優解。由我們的結果來看，適度複雜化標籤是一個可行有效的優化方向。

四、實務方面的應用

實務方面，根據訓練出的數據，如圖 74、圖 75，可觀察到組裝作循環週期的誤差值最低為 0.54%，有效動作為 4.65%。依據信賴區間法則，我們將 95% 信心水準作為閾值，這意味著誤差值皆小於 5%，屬於正常誤差範圍，換言之，本研究所訓練的模型具備實際應用於工廠的潛力，用於測定整體動作循環標準工時。

這一模型的引入，能夠顯著減少聘請工業工程師進行工時測量的人力成本，同時縮短測量所需的時間。更重要的是，它能夠大量收集產線上的實際情況，從而獲得更豐富的資料集。這些資料不僅有助於提高生產效率，還能夠為進一步的優化提供依據，最終提升整體生產管理的精確度和效益。

本研究之訓練模型能夠識別和分析操作動作，提供精準的反饋，這使得生產線上的問題可以被發現並解決，從而降低生產損失和成本。結合上述優點，本研究的訓練模型在循環週期和有效動作誤差值均低於 5%的前提下，能夠穩定地應用於實際工廠環境中，對於工時測定和產線管理均具有顯著的實際價值。

週期循環	first	second	third	forth	avg	error percentage
正確版(人工貼標)	96.76	96.76	108.24	128.28	107.53	
原版(8個標籤)	96.46	96.51	109.01	127.29	107.32	
複雜版(11個標籤)	96.49	96.5	109.02	127.28	107.32	
原版與正確版誤差時間(秒)	0.3	0.25	0.77	0.99	0.5775	0.54%
複雜版與正確版誤差時間(秒)	0.27	0.26	0.78	1	0.5775	0.54%

圖 74 組裝動作循環週期

週期循環	first	second	third	forth	avg	error percentage
正確版(人工貼標)	79.68	74.1	76.38	105.42	83.895	
原版(8個標籤)	82.32	80.7	87.37	104.83	88.805	
複雜版(11個標籤)	78.21	68.22	79.89	100.67	81.748	
原版與正確版誤差時間(秒)	2.64	6.6	10.99	0.59	5.205	6.20%
複雜版與正確版誤差時間(秒)	1.47	5.88	3.51	4.75	3.903	4.65%

圖 75 有效動作時間對照

伍、 結論與未來展望

過去，時間研究往往需要工業工程師花費較多人力研究量測等，因此本研究探討基於多模態影像及人體動作辨識技術優化產線流程分析生成流程程序圖，在兩種標基礎下，建立動作辨識模型，並生成流程程序圖，達成整體組裝動作循環週期對比正確流程成程序圖判定誤差為整體時間的 0.54%。由這項數據可證實運用動作辨識模型建立流程程序圖，可有效減少業工程師在量測產線流程上所花費的時間，並且應對當今少量多樣的生產環境，更進一步縮短工業工程師在面對產線變化，優化產線所需的時間。

在動作辨識環節中，我們基於三種不同標訓練模型，並各自對應影像辨識結果，整體精確率(Precision)平均達 84%，召回率(Recall)平均達 85%，當中由於無效動作涵蓋較多不同類型之動作，使動作辨識模型誤判機率達 14.08%，致整體衡量指標降低，但對於有效組裝動作仍具有良好的辨識效能，誤差僅為 4.65%。由這項數據可知本研究持續優化對於無效動作的辨識的準確度，能有效提高模型的準確率。生成流程程序圖環節中，本專題研究設計兩套計算流程程序圖邏輯，並基於模型辨識動作基礎，生成流程程序圖：以標準動作標作為比較基礎，生成正確版流程程序圖，對照基於模型辨識動作基礎生成之流程程序圖在「標準動作標」與「標準動作及細分無效動作標」模型基礎，生成之流程程序圖中，對比正確版流程程序圖在 Cycle time 平均誤差在正負 0.5 秒內，誤差比例為整體時間的 0.54%，計算有效動作時間及無效動作時間，每套操作流程誤差平均在正負 5 秒內，誤差比例為整體時間的 4.84%。

這項技術目前在實務上仍有許多需要研究和解決的難題，例如，對無效動作的辨識準確率仍有待提高，對於避免誤判和提高系統的可靠性至關重要。此外，相同手部動作在不同情境下的判定，例如：組裝動作與非組裝動作之間的區別，也是一個挑戰，需要系統具備更高精確的動作識別能力。本研究採用影像辨識技術中之動作辨識和物件辨識技術。但兩種方法仍有其局限性：動作辨識可能在人體被環境遮擋的情況下失效，而物件辨識在區分使用相同工具不同動作上可能不夠精確。為了優化實務上的問題，未來將整合更多模態的資訊，如聲音、時間或空間開發大型影像處理模型，這種模型能夠提供更全面的分析和判斷。例如：通過融合骨架辨識的動作捕捉能力和物件辨識的場景理解能力，分析動作在時間流上的變化與空間的關係，使系統可以在更加複雜和動態的生產環境中更準確地辨識和分類動作，從而提高實務操作中的應用效能。

面對當今全球化且協同合作的環境，本研究未來若能實際應用於工廠生產線上，將有效促進工業環境發展，建立工業 5.0 的發展基礎。更可結合人機協作(Human-Machine Collaboration、全面互聯(Hyperconnectivity、生成式 AI 等創新技術，使整個工廠的資料及數據能夠共享，且自主分析進一步讓生成式 AI 針對流程程序圖判讀，給出相對應的解決方案或是優化策略。

這樣的技術不僅提高了生產效率和系統的適應性，也大幅減輕了工業工程師的負擔。工業工程師將在未來具備至關重要的角色，不僅要精通技術應用，還需要具備跨文化的溝通能力和協作精神，以便在多變的國際環境中善用創新技術達成有效管理和優化生產流程。

陸、參考資料

1. Ciregan, D., U. Meier, and J. Schmidhuber. *Multi-column deep neural networks for image classification*. in *2012 IEEE conference on computer vision and pattern recognition*. 2012. IEEE.
2. Krizhevsky, A., I. Sutskever, and G.E. Hinton, *Imagenet classification with deep convolutional neural networks*. Advances in neural information processing systems, 2012. **25**.
3. Bordes, A., et al. *Joint learning of words and meaning representations for open-text semantic parsing*. in *Artificial intelligence and statistics*. 2012. PMLR.
4. Karhunen, J., T. Raiko, and K. Cho, *Unsupervised deep learning : A short review*. Advances in independent component analysis and learning machines, 2015 : p. 125-142.
5. Poppe, R., *A survey on vision-based human action recognition*. Image and vision computing, 2010. **28**(6) : p. 976-990.
6. Sun, Z., et al., *Human action recognition from various data modalities : A review*. IEEE transactions on pattern analysis and machine intelligence, 2022. **45**(3) : p. 3200-3225.
7. Gorelick, L., et al., *Actions as space-time shapes*. IEEE transactions on pattern analysis and machine intelligence, 2007. **29**(12) : p. 2247-2253.
8. Laptev, I., *On space-time interest points*. International journal of computer vision, 2005. **64** : p. 107-123.
9. Simonyan, K. and A. Zisserman, *Two-stream convolutional networks for action recognition in videos*. Advances in neural information processing systems, 2014. **27**.
10. Sherstinsky, A., *Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network*. Physica D : Nonlinear Phenomena, 2020. **404** : p. 132306.
11. Tran, D., et al. *Learning spatiotemporal features with 3d convolutional networks*. in *Proceedings of the IEEE international conference on computer vision*. 2015.
12. Vaswani, A., et al., *Attention is all you need*. Advances in neural information processing systems, 2017. **30**.

13. Du, W., Y. Wang, and Y. Qiao. *Rpan : An end-to-end recurrent pose-attention network for action recognition in videos.* in *Proceedings of the IEEE international conference on computer vision.* 2017.
14. Sun, L., et al. *Lattice long short-term memory for human action recognition.* in *Proceedings of the IEEE international conference on computer vision.* 2017.
15. Ji, S., et al., *3D convolutional neural networks for human action recognition.* *IEEE transactions on pattern analysis and machine intelligence*, 2012. **35**(1) : p. 221-231.
16. Li, X., B. Shuai, and J. Tighe. *Directional temporal modeling for action recognition.* in *Computer Vision–ECCV 2020 : 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VI 16.* 2020. Springer.
17. Zhang, H., et al. *Few-shot action recognition with permutation-invariant attention.* in *Computer Vision–ECCV 2020 : 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part V 16.* 2020. Springer.
18. Girdhar, R., et al. *Video action transformer network.* in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition.* 2019.
19. Sun, K., et al. *Deep high-resolution representation learning for human pose estimation.* in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition.* 2019.
20. Ren, B., et al., *A survey on 3d skeleton-based action recognition using learning method.* *Cyborg and Bionic Systems*, 2020.
21. Shahroudy, A., et al. *Ntu rgb+ d : A large scale dataset for 3d human activity analysis.* in *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2016.
22. Pascanu, R., et al., *How to construct deep recurrent neural networks.* arXiv preprint arXiv : 1312.6026, 2013.
23. O'shea, K. and R. Nash, *An introduction to convolutional neural networks.* arXiv preprint arXiv : 1511.08458, 2015.
24. Zhang, S., et al., *Graph convolutional networks : a comprehensive review.* *Computational Social Networks*, 2019. **6**(1) : p. 1-23.
25. Rahmani, H. and M. Bennamoun. *Learning action recognition model from depth and skeleton videos.* in *Proceedings of the IEEE international conference on computer vision.* 2017.

26. Dhiman, C. and D.K. Vishwakarma, *View-invariant deep architecture for human action recognition using two-stream motion and shape temporal dynamics*. IEEE Transactions on Image Processing, 2020. **29** : p. 3835-3844.
27. Bruce, X., et al., *Mmnet : A model-based multimodal network for human action recognition in rgb-d videos*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2022. **45**(3) : p. 3522-3538.
28. Zhao, R., H. Ali, and P. Van der Smagt. *Two-stream RNN/CNN for action recognition in 3D videos*. in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017. IEEE.
29. Rodomagoulakis, I., et al. *Multimodal human action recognition in assistive human-robot interaction*. in *2016 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. 2016. IEEE.
30. Lin, W., et al. *Human activity recognition for video surveillance*. in *2008 IEEE international symposium on circuits and systems (ISCAS)*. 2008. IEEE.
31. Host, K. and M. Ivašić-Kos, *An overview of Human Action Recognition in sports based on Computer Vision*. Heliyon, 2022. **8**(6).
32. Ji, X., C. Wang, and Y. Li, *A view-invariant action recognition based on multi-view space hidden markov models*. International Journal of Humanoid Robotics, 2014. **11**(01) : p. 1450011.
33. Yan, S., Y. Xiong, and D. Lin. *Spatial temporal graph convolutional networks for skeleton-based action recognition*. in *Proceedings of the AAAI conference on artificial intelligence*. 2018.
34. Duan, H., et al. *Revisiting skeleton-based action recognition*. in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022.
35. Feichtenhofer, C., et al. *Slowfast networks for video recognition*. in *Proceedings of the IEEE/CVF international conference on computer vision*. 2019.
36. Jia, Y., et al. *Caffe : Convolutional architecture for fast feature embedding*. in *Proceedings of the 22nd ACM international conference on Multimedia*. 2014.
37. Chen, C., et al. *Deepdriving : Learning affordance for direct perception in autonomous driving*. in *Proceedings of the IEEE international conference on*

- computer vision*. 2015.
- 38. Girshick, R. *Fast r-cnn*. in *Proceedings of the IEEE international conference on computer vision*. 2015.
 - 39. Redmon, J., et al. *You only look once : Unified, real-time object detection*. in *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.
 - 40. Ren, S., et al., *Faster r-cnn : Towards real-time object detection with region proposal networks*. Advances in neural information processing systems, 2015. **28**.
 - 41. Erhan, D., et al. *Scalable object detection using deep neural networks*. in *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014.
 - 42. Huang, J., et al. *Speed/accuracy trade-offs for modern convolutional object detectors*. in *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.
 - 43. Szegedy, C., A. Toshev, and D. Erhan, *Deep neural networks for object detection*. Advances in neural information processing systems, 2013. **26**.
 - 44. Girshick, R., et al. *Rich feature hierarchies for accurate object detection and semantic segmentation*. in *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014.
 - 45. Lin, T.-Y., et al. *Feature pyramid networks for object detection*. in *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.
 - 46. Lu, X., et al. *Grid r-cnn*. in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019.
 - 47. Chen, Y., et al., *Revisiting feature alignment for one-stage object detection*. arXiv preprint arXiv : 1908.01570, 2019.
 - 48. Lin, T.-Y., et al. *Focal loss for dense object detection*. in *Proceedings of the IEEE international conference on computer vision*. 2017.
 - 49. Tian, Z., et al., *FCOS : Fully convolutional one-stage object detection*. arXiv 2019. arXiv preprint arXiv : 1904.01355, 1904.
 - 50. Liu, W., et al. *Ssd : Single shot multibox detector*. in *Computer Vision–ECCV 2016 : 14th European Conference, Amsterdam, The Netherlands, October*

- 11–14, 2016, Proceedings, Part I* 14. 2016. Springer.
- 51. Redmon, J. and A. Farhadi. *YOLO9000 : better, faster, stronger*. in *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.
 - 52. Redmon, J. and A. Farhadi, *YOLO v3 : An incremental improvement*. arXiv preprint arXiv : 1804.02767, 2018.
 - 53. Bochkovskiy, A., C.-Y. Wang, and H.-Y.M. Liao, *YOLO v4 : Optimal speed and accuracy of object detection*. arXiv preprint arXiv : 2004.10934, 2020.
 - 54. Zhu, X., et al. *TPH-YOLO v5 : Improved YOLO v5 based on transformer prediction head for object detection on drone-captured scenarios*. in *Proceedings of the IEEE/CVF international conference on computer vision*. 2021.
 - 55. Li, C., et al., *YOLO v6 : A single-stage object detection framework for industrial applications*. arXiv preprint arXiv : 2209.02976, 2022.
 - 56. Wang, C.-Y., A. Bochkovskiy, and H.-Y.M. Liao. *YOLO v7 : Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2023.
 - 57. Gao, P., et al., *Container : Context aggregation network*. arXiv preprint arXiv : 2106.01401, 2021.
 - 58. Dollár, P., M. Singh, and R. Girshick. *Fast and accurate model scaling*. in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021.
 - 59. Vasu, P.K.A., et al. *Mobileone : An improved one millisecond mobile backbone*. in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2023.
 - 60. Ding, X., et al. *Repvgg : Making vgg-style convnets great again*. in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021

附錄

資料處理 1：將細微動作持續的時間加到後面的動作中當作後續動作執行前的緩衝時間。

```
1 # 影片標籤動作種類
2 action_list = ["Assemble_3.1", "Assemble_3.2",
3                 "Assemble_3.3", "Assemble_3.4",
4                 "Assemble_3.5", "Brush",
5                 "ScrewUp", "Other"]
6
7 # 建構暫時的dataframe
8 temp_df = pd.DataFrame(columns=['Action', 'Duration',
9                           'Accumulative_Time'])
10
11 # 計算每次動作持續時間
12 duration = 0
13 Type = data['slowfast'][0]
14 total_time = 0
15
16 # 把只出現1秒的動作併到後續的動作
17 for action in range(len(data['slowfast'])):
18     if data['slowfast'][action] == Type:
19         duration += 1
20     else:
21         time = round(duration * frame_sec,2)
22         if time < 1:
23             duration += 1
24             Type = data['slowfast'][action]
25             continue
26         else:
27             total_time += time
28             temp_df = pd.concat([temp_df, pd.DataFrame({
29                     'Action': [action_list[Type]],
30                     'Duration': [time],
31                     'Accumulative_Time': [round(total_time,2)]}),
32                     ignore_index=True)
33
34         duration = 1
35         Type = data['slowfast'][action]
36     time = round(duration * frame_sec,2)
37 total_time += time
38 temp_df = pd.concat([temp_df, pd.DataFrame({
39                     'Action': [action_list[Type]],
40                     'Duration': [time],
41                     'Accumulative_Time': [round(total_time, 2)],
42                     'Cycle_Time':None})], ignore_index=True)
43
44 #print(temp_df.to_string())
```

圖 76 處理偏差時間

資料處理 2：將資料處理 1 得到的結果中連續相同動作的時間合併在一起。

```
1 # 建構輸出的 DataFrame
2 output_df = pd.DataFrame(columns=['Action', 'Duration', 'Accumulative_Time', 'Cycle_Time'])
3
4 #把連續的動作合在一起
5 time = 0
6 Type = temp_df['Action'][0]
7
8 for i in range(len(temp_df)):
9     if temp_df['Action'][i] == Type:
10         time += temp_df['Duration'][i]
11     else:
12         output_df = pd.concat([output_df, pd.DataFrame({
13             'Action': [temp_df['Action'][i-1]],
14             'Duration': [time],
15             'Accumulative_Time':
16             [round(temp_df['Accumulative_Time'][i-1], 2)],
17             'Cycle_Time': [None]}),
18             ignore_index=True])
19
20     time = temp_df['Duration'][i]
21     Type = temp_df['Action'][i]
22
23 # 將最後一個動作和累積時間添加到輸出的 DataFrame 中
24 output_df = pd.concat([output_df, pd.DataFrame({
25             'Action': [temp_df['Action'].iloc[-1]],
26             'Duration': [time],
27             'Accumulative_Time':
28             [round(temp_df['Accumulative_Time'].iloc[-1], 2)],
29             'Cycle_Time': [None]})], ignore_index=True)
```

圖 77 處理連續的相同動作時間

國立臺灣科技大學 工業管理系一一二學年度實務專題

題目：基於多模態影像及人體動作辨識技術優化產線流程分析

專題編號：TR-112-01-112