出國報告（出國類別：進修）

荷蘭植物表型分析中心技術應用與營運策略交流

服務機關：農業部農業試驗所
姓名/職稱：杜元凱/副研究員
派赴國家/地區：荷蘭/瓦赫寧恩
出國期間：112 年 7 月 1 日至 7 月 16 日
報告日期：112 年 10 月 4 日

摘要

「表型體學」(Phenomics) 為針對特定生物體，進行系統性外表型調查、分析的研究領域。世界各國農業研究單位與國際種子公司，已建置各種作物表型體學相關之軟、硬體設施，藉由透過感測器或鏡頭等進行作物溫度、高光譜、葉綠素螢光、2 維影像等相關生物資料擷取等，進而對複雜的作物性狀進行調查與探討，並串聯基因體學研究，加速作物育種工作。目前我國在表型體學領域的發展現況，計有世界蔬菜中心、國立臺灣大學、中央研究院等單位，近年已投入相當人力、經費進行相關研究與硬體設施建置，本部 (農業部) 亦獲取中長程公共建設計畫支持，將於農業試驗所建構國家級之植物表型分析中心，期望可作為我國植物表型體學之研究、應用樞紐，並連結各單位人力、硬體資源，提供作為我國植物種苗產業發展研究需求之網絡。

在各先進國家發展表型體分析平台之中，荷蘭植物生態表型分析中心 (Netherlands Plant Eco-phenotyping Centre, NPEC) 之表型體設施整合各項新穎之表型分析模組，除植物地上部 3D 結構、多光譜、葉綠素螢光等分析模組，同時包含植物根部生長、氣候反應生長分析模組等，NPEC 於 111 年度完工並正式對外提供表型體分析服務。NPEC 為當下整合最完整、最新穎之表型分析模組之研究中心，本質上屬綜合性的國家研究機構，由瓦赫寧恩大學 (Wageningen University & Research, WUR) 和烏得勒支大學 (Utrecht University, UU) 共同營運並支應 NPEC 相關學術諮詢、技術研發、科技應用等相關發展，目標使用者為荷蘭和國際學術界研究人員以及私人企業提供服務。

目前在我國除農業試驗所具有表型體設施，世界蔬菜中心、國立臺灣大學、中央研究院等單位，近年已投入相當人力、經費進行相關研究與硬體設施建置。對於表型體領域此類先端技術研究，需盤點並串連現有表型體領域的資源，以將研究效能最大化。此外，表型分析中心的長期營運模式亦須事先詳加規劃。據此，本計畫參加由 WUR 所開設之植物表型分析課程，以增進研究人員表型體分析、研究專業職能。同時，本計畫規劃參訪 NPEC，期望參訪活動與人員交流，汲取先進國家表型中心之研究應用、營運規劃以及產業鏈結，據此規劃我國表型體中心營運模式，並期可與先進國家表型體中心建立合作夥伴關係。

本次出國行程個人相關心得與建議包含植物表型研究能力建構 (表型資料

收集、分析方法)、階段發展（先研究室階段，再拓展至溫室、大田)、表型研究方向（善用我國既有優勢以及扣合農業政策）以及國家表型中心未來規劃等，均已條列分述於本報告當中，期望對於我國未來植物表型體學之研究、發展與應用，能有所實質助益。

目錄

壹、目的

為增進我國植物表型體學研究能力並瞭解先進國家表型體中心研究發展、規劃，本出國計畫目標為參加 WUR 所開設之植物表型分析課程並取得課程認證，藉以增進研究人員表型體分析、研究專業職能。另一目標為透過參訪 NPEC 及人員交流活動，藉此汲取先進國家表型中心之研究應用、營運規劃以及產業鏈結之經驗，並與先進國家表型體中心建立合作夥伴關係。

貳、行程概要

本計畫行程概要如下表 1，7 月 1 日凌晨出發至荷蘭，7 月 3 日至 7 日參加植物表型體分析課程，課程介紹如附錄 1。7 月 10 日至 14 日規劃參訪 WUR 及 NPEC。

表 1、出國計畫行程概要說明表。

07/01	07/02	07/03	07/04	07/05	07/06	07/07	07/08	07/09	07/10	07/11	07/12	07/13	07/14	07/15	07/16
六	日	一	二	三	四	五	六	日	一	二	三	四	五	六	日
搭機前往阿姆斯特丹		植物表型體學分析課程	植物表型體學分析課程	植物表型體學分析課程	植物表型體學分析課程	植物表型體學分析課程			WUR/NPEC參訪與研究人員交流	WUR/NPEC參訪與研究人員交流	WUR/NPEC參訪與研究人員交流	WUR/NPEC參訪與研究人員交流	WUR/NPEC參訪與研究人員交流	搭機返回台灣	搭機返回台灣

WUR: Wageningen University & Research，瓦赫寧恩大學
NPEC: Netherlands Plant Eco-Phenotyping Centre，荷蘭表型分析中心

參、表型分析課程內容

本次所參加之植物表型分析課程內容包含影像擷取、影像處理、影像分析以及機器學習 (machine learning, ML)、深度學習 (deep learning, DL) 的應用，逐日課程主題、內容摘要、關鍵字以及使用程式碼與軟體摘要整理如下表 1，程式碼整理如附錄 2，整理課程重點條列如下:

一、2 維影像擷取與品質控管

本次課程係以植物影像分析為主，最初階的影像資料為彩色圖像 (color image)，在彩色圖像中每一個像素 (pixel) 擁有三個數值，分別為 R (red)、G (green)、B (blue)的亮度，透過 R、G、B 數值疊加而形成各種人類視力所能感知到的所有顏色。欲擷取足夠品質之植物表型彩色影像，首先需要合適的光圈 (apture)、快門速度 (shutter)、感光度以及焦距設定。光圈為鏡頭中央開孔的大小，光圈大小決定鏡頭接收光量，進而影響目標物件景深;快門速度為鏡頭中央開孔啟閉速度，為決定移動目標清晰程度; 感光度又稱 ISO 值，為相機感光元件對光的靈敏度。其中光圈、快門速度、感光度同時也決定曝光時間，意即決定進入鏡頭的總光亮。而焦距則為鏡頭對焦距離，決定焦距決定視野範圍中目標物件清晰程度。

在進行影像擷取時，首先要充分了解環境條件如光源類型、光照角度、光照強度與穩定度。不同光源類型提供的輻射頻譜分布不同，光照角度差異會導致陰影或遮蔽，光照強度與穩定度則會影響目標物件的輪廓與清晰度。上述三者會決定影像擷取是否進行補光、拍攝角度以及光圈、快門速度、感光度設定值。另外目標物件是為靜止或移動也決定快門速度設定值。

二、2 維影像處理流程

在獲取具備一定品質的影像資料後，因應研究人員研究主題，對於影像資料中目標物件可能不盡相同，例如僅需要影像資料中的花、葉、果實等部分影像，如何進行目標物件分割 (segmentation)，為植物表型體學影像分析重要的分析流程。

影像資料物件分割的策略大致可分門檻 (threshold) 設定與 ML、DL 模型應用。門檻設定是利用目標物件與其他影像區域在像素位階的差異設定，而達到分割的效果，例如在本次課程示範是將影像先轉換成灰階 (gray scale) 影像，再透過目標物件與其他影像區域在灰階值的差異設定，以達到目標物件分割目的。另外，也可透過目標物件的影像特徵進行分析，例如連續的邊緣 (edge)、輪廓 (contour) 等。上述分析流程可以免費、開源的 ImageJ、Napri 以及 Python OpenCV 套件等軟體進行處理。除了轉換成灰階影像進行分析外，也可嘗試將影像轉換為不同色域 (color space)，例如 HSV (hue, saturation, brightness)、HSL (hue, saturation, lightness) 以及 LAB

(lightness，a 表示深綠色到灰色、亮粉紅色；b 則是從亮藍色到灰色、黃色)，隨後端視是否可以找尋合適的閥值設定，以達到目標物件分割的效果。另外需注意不同色域有適用不同場景，例如 LAB 色域對於有陰影 (shadows) 的目標物件可能會有比較好的影像特徵擷取效果，HSV、HSL 則是以顏色差異作為目標物件的特徵資訊。

如前述品質控管部分，在影像擷取階段的光源、反光、陰影、目標物件在影像中的複雜程度以及和背景對比 (contrast) 都會造成目標物件分割困難度，故再次說明影像處理困難與否，與一開始的影像擷取條件設定有相當大關係，研究人員對於影像擷取條件設定與影像分析流程規劃，在一開始就需設想兩者間的配合與各項參數值設定。

三、高 (多) 光譜資料分析

高 (多) 光譜相關技術為近年常應用於農業研究的非破壞檢測、檢量方法，常見研究光譜範圍為 400-2,500 nm，常見應用在農業研究領域光譜範圍則落於可見光 (400-700 nm) 到近紅外光 (700-1,300 nm) 範圍。(多) 光譜資料擷取系統可分非影像式高光譜感應器 (non-imaging hyperspectral sensor) 與高光譜成像系統 (hyperspectral imager)。本次課程以高光譜成像系統所擷取影像資料分析為主進行說明，高光譜成像系統除可以獲得每個像素之反射光譜外，可同時記錄目標物影像資料，可獲取平面影像 (X、Y 軸) 與光譜 (Z 軸) 所構成的 3 維資料，相較於非影像式高光譜感應器，所擷取之平面影像資料配合影像分析、機器學習方法，進行病徵、老化等作物外觀型態判別方法的建立。

如前述，常用高光譜範圍是 400-2,500 nm，依據研究需求，以固定間隔進行切割，例如 3 nm 間隔，如此每切 1 個波段就當作 1 個變數，所以 400-2,500 可以分割為，700 個變數，故事實上分析光譜數據，是在處理一高維度資料的問題，常用分析光譜資料的方法包含淨最小平方法 (partial least squared, PLS) 以及隨機森林 (random forest, RF)。另外，以高光譜成像系統為例，分析流程為會先在影像上圈選感興趣區域 (region of interest, RoI)，再取 RoI 範圍所有像素反射光譜進行平均，由此再與對照區域平均光譜進行比較分析。

另一種使用光譜資料找出與特定植物生理特性關係的應用為使用植生指數 (vegetation index, VI)，VI 是由多個光譜波長或波段藉由特定公式計算所得，在遙測感應 (remote sensing) 領域的應用已相當久遠，常見 VI 包含常態化差值植生指標 (normalized difference vegetation index, NDVI)、常態化差值水分植生指標 (normalized difference water index, NDWI)、寬廣動態範圍植生指標 (wide dynamic range vegetation index, WDRVI) 以及光化學反射植生指標 (photochemical reflectance vegetation index, PRI) 等，均

已被證實可用來評估植物生長狀況、含水量與光合作用效率等，許多 VI 數值的變化與植物的生長特性與生理反應具高度相關性。

四、機器學習與深度學習在影像分析的應用

ML 與 DL 與在影像分析的幾項重要應用分別為影像分類 (image classification)、物件偵測 (object detection)、圖像分割 (image segmentation)，雖然使用 ML 與 DL 已為現代眾多研究領域的顯學，然而除非前述相對較為簡單易操作之 ImageJ、Napri 以及 OpenCV 已無法處理，否則目標物件分割與辨識不見得都要用深度學習與機器學習。

在本次課程首先說明 ML 模型以及其在影像分析的功能，ML 可分為監督式 (supervised) 與非監督式 (unsupervised):

1.非監督式機器學習:

(1). 無使用已標示資料(labeled data)

(2). 自主進行資料分群 (automatic clustering)

(3). 例如 K-means 法與 PCA (principal component analysis)

2.監督式機器學習:

(1).需使用已標示資料

(2).決策樹 (decision trees)與 RF 進行資料分群

另外，在使用 ML 時需要將資料集分為訓練資料集 (training data set) 與測試資料集 (test data set)，一般而言，訓練資料集/測試資料集是由原始資料及隨機抽樣 80%/20%或 70%/30%而形成，訓練資料集用來訓練模型用，測試資料集則用來測試訓練模型的效能，例如目標物件影像分割能力。當原始資料集數目不大時，要注意資料滲漏問題，意即在訓練資料集與測試資料集當中類別資料數目分布相當不成比例，例如標示為 1 類別的資料數目很少，表示為 0 類別的資料數目卻很多，對於模型訓練結果與測試結果影響甚鉅，無法真實評估模型本身效能。

在 DL 模型部分，課程中首先介紹卷積神經網絡 (convolutional neural network, CNN)，其藉由卷積運算 (convolutional operation) 擷取輸入資料的特徵，輸入資料為 2 維 RGB 影像，CNN 經常用來分類影像以及偵測影像資料中是否具有目標物件。基於 CNN 模型架構，研究人員也開發具備語義分割 (semantic segmentation) 的深度學習模型架構，語義分割針對影像中的所有像素點進行分類，達到影像中的物件皆可進行分類，常用語義分割模型如 FCN (fully convolutional networks)、U-Net。更進階的深度模型可以進行實例分割 (instance segmentation)，實例分割結合物件偵測和語義分割的功能，可將影像中各個物件定位，如為相同類別也會分割成不同物件，常用實例分割模型如 Faster R-CNN 以及 Mask R-CNN。除了針對 2 維影像進行目標物件辨識與分割外，目前也已開發可以 3 維點雲 (point cloud) 資料進行目標物件辨識與

分割，例如在空間分布中的花、葉、果實的辨識與分割，常用來進行點雲資料的辨識與分割深度模型為 PointNet 和 PointNet++。

五、3 維結構重建與分析

在 3 維結構重建部分，大部分重建用的輸入資料皆為 2 維影像，由多張 2 維影像拼接出網格 (mesh)，再由網格產製 3 維點雲資料。另外，也可使用立體視覺相機，透過在兩台相機位置在不同角度擷取同一個目標物件，再由兩台相機所擷取影像進行匹配、演算後，藉此產製 3 維點雲資料。常用來匹配 2 維影像產製 3 維點雲的演算法為 SFM (structure from motion)，多張 2 維影像可以單一相機在不同角度或者多個相機固定角度對同一目標物件進行影像擷取，故近年也有多個研究報告指出 RGB 相機掛載到 UAV (unmanned aerial vehicle) 進行多個角度影像擷取，進一步實現田間作物、果樹的 3 維結構重建。

用來視覺化與量測分析點雲資料的免費軟體主要 CloudCompare、potree 以及 python Open3D，藉由上述軟體的分析可以得知目標物件體積、葉面積、冠層 (canopy) 體積等重要植物空間分布性狀資料，進一步透過深度學習模型應用如 PointNet++，可實現自動分割出花、果實、莖等不同植物器官、部位。另一個植物 3 維結構重建的重要應用為功能性結構植物模型 (functional structure plant model, FSPM) 以及數位植物表型分析 (digital plant phenotyping)。FSPM 應用於解析植物在何種 3 維結構下可以有較好的生理、生化反應，例如具有較大的冠層結構或莖葉構型偏向分散結構，截取光照效率較佳，因此具有較高競爭生長優勢。數位植物表型分析則著重於在電腦運算端無偏差呈現實際作物在溫室或田間的生長結構、生理反應等，並進一步應用在作物育種

六、試驗設計與統計分析

本課程內容著重於試驗設計、縱貫性 (longitudinal) 資料離群值 (outlier) 偵測以及空間資料分析。植物表型分析試驗通常是在溫室與田間進行，故需在有限的空間進行有效的試驗配置，方能藉由顯著差異分析結果的可信度與可重複性。課程中強調，無論是在溫室或田間的試驗設計，務必需要有重複 (replication)、隨機 (randomization) 以及區集 (block) 等設計。

由於植物表型分析研究通常是收集同一資料來源兩個 (含) 以上時間點的資料，例如於溫室或田間執行表型試驗，並在固定間隔時間進行表型分析收集資料，此類資料結構屬於縱貫性資料或稱重複量測資料 (repeated measurement data)，識別與捨去縱貫性資料中的離群值對於後續統計分析的結過有相當大助益，同時也可進一步探討離群值發生原因，例如量測誤差或者空間變異 (溫度、日照)，接續可針對造成離群值原因進行修正，完備溫室或田間有限空間可利用性。課程中示範使用 R 軟體套件 statgenHTP (statisical

genetics, high-throughput phenotyping) 識別縱貫性資料中的離群值，主要可透過 statgenHTP 主要透過將試驗中表型資料配適平滑函數後，在每個時間點觀察曲線特徵、PCA 分群以及成對相關係數 (pairewise correlations of coeffienct)，藉此觀察是否有離群值，提供研究人員進一步探討離群值發生原因。

前述離群值發生原因可能來自於空間變異導致溫室或田間特定位置表型分析資料容易出現偏差，本次課程另外介紹以 R 軟體 SpATS (spatial analysis of trails using splines) 進行此種空間異質性的視覺化分析。SpATS 利用 splines 函數將空間變異猜解為行、列與行列交感的變異，並藉由視覺化呈現原始表型資料之行列分布、splines 函數配適空間變異分布、基因型變異之行列分布以及殘差 (residues) 之行列分布等，用以觀察離群值變異原因。

表 2、表型體課程主題、內容摘要、關鍵字以及使用程式碼與軟體摘要說明表。

次序	主題	內容摘要	Keywords	程式碼與驅動程式
Day1_1	Introduction of phenotyping course	■ 綜述本課程內容	■ Image-based plant phenotyping ■ Basic image-processing pipeline ■ Adjust camera and lens ■ Basic image-processing methods ■ Machine learning and deep learning methods ■ 2D images and 3D point clouds ■ Spatial and temporal analysis	無
Day1_2	Introduction to Image analysis and phenotyping	■ 概要說明各種植物影像表型分析技術與應用範例 ■ 進行影像分析事先需要具備的專業領域知識	■ 量化量測生物性狀	無
Day1_3	Image sensing and acquisition	■ 說明 Image sensing 要件 ■ Aperture (depth of	■ Sensor ■ Illumination ■ Optics	■ Driver: ids-software-suite-win-4.96.1 ■ 程式碼 (Python colab)

		file, DOF)、shutter (sharpness)、ISO speed (gain)	■ Image format ■ Color space ■ Clipping problem ■ Aperture (depth of file, DOF)、shutter (sharpness)、ISO speed (gain)	✓ noise_filtering_part_1 ✓ noise_filtering_part_2 ✓ noise_filtering_part_3 ✓ noise_filtering_part_4
Day2_1	Multi-sensor remote sensing for monitoring sustainable plant production	■ 說明各種應用在 plant phenomics 的感應器、相機與載具 ■ 說明遙測技術在 plant phenomics 的應用 ■ 範例說明 UAV 結合 RGB/高光譜/LiDAR、AI 演算法在十字花科、茄科、莧科作物表型分析的應用	■ Geo-information science ■ Remote sensing ■ Solar induced fluorescence (SIF) ■ Thermal infrared (TIR) ■ Quantitative structure modelling	無
Day2_2	Image segmentation and introduction	■ 說明如何進行目標物件分割與標示的方法 ■ Thresholding: pixel, edge 與 region 閥值 ■ Model based method:	■ ImageJ ■ Binary image ■ Color space: HSL, HSV ■ Watershed ■ k-means clustering ■ Active contours (Snakes)	■ ImageJ 安裝檔: ImageJ-win54.exe ■ ImageJ plug-in 程式 ■ 使用 ImageJ 進行 segmentation 教學說明檔案

		Clustering 與 contour 法	■ HOG (Histogram of Oriented Gradients) features ■ Neural Networks	
Day3_1	Classical Machine learning	■ 介紹不同類型的機器學習模型 ■ Unsupervised learning: K-means clustering 與 Gaussian mixture models ■ Supervised learning: Decision tree, Random forest and neural networks. ■ 機器學習模型實用範例	■ Classification and regression ■ Lab color space ■ Maximum likelihood estimator ■ Perceptron ■ Training, test and validation dataset. ■ Mean squared error	■ 程式碼 (Python colab): ✓ classification.ipynb ✓ mlp_classification.ipynb
Day3_2	Introduction to neural networks	■ 介紹 neural networks 以及 deep learning 模型 ■ 實作 deep learning 範例	■ Multilayer Perceptron ■ Data leakage ■ Imbalanced dataset	■ 程式碼 (Python colab): ✓ tomato_sweetness.ipynb ✓ tomato_sweetness_and_mlp.ipynb
Day3_3	Neural networks and	■ 介紹 convolutional neural networks	■ convolutional filter ■ 3D convolutional kernels	■ 程式碼 (Python colab): ✓ WUR_SummerSchool2023_CNN_

	deep learning	(CNN) ■ 實作 CNN 範例	■ Max pooling ■ Semantic segmentation ■ Object detection ■ Instance segmentation	Session.ipynb
Day4_1	Semantic segmentation	■ 介紹與實作 semantic segmentation	■ Unpooling ■ Deconvolutional network ■ SegNet (encoder-deconder design)	■ 程式碼 (Python colab): ✓ WUR_SummerSchool2023_Semantic_Segmentation.ipynb
Day4_2	Instance segmentation	■ 介紹與實作 Instance segmentation	■ Mask RCNN ■ Region proposal network (RPN) ■ Region of interest alignment (ROIAlign) ■ Object detection ■ Anchor-based ■ Depth image ■ Detectron2	■ 程式碼 (Python colab): ✓ Ex1_MaskRCNN_R2P.ipynb ✓ Ex2_MaskRCNN_R2P.ipynb ✓ Ex3_MaskRCNN_R2P_pretrained.ipynb
Day4_3	3D plant phenotyping using deep learning	■ 介紹 3D 植物表型資料收集、處理與分析 ■ 加入光譜的資訊可強化 3D segmentation 效果 ■ 說明重建 3D 模型的挑戰:	■ Multiview imaging system ■ YOLO ■ Functional structure plant model (FSPM) ■ Laser triangulating ■ Structure from motion (SfM)	■ 程式碼 (Python colab): ✓ point_clouds_and_deep_learning_part_1.ipynb ✓ point_clouds_and_deep_learning_part_2.ipynb

		✓ Class imbalance ✓ Limited training data ✓ Variation ✓ Occlusion ■ 當樣本太少或出現不平行狀態時，選擇 training data 的方法-Active learning	■ Permutation invariant ■ PointNet ■ Affine transformation ■ Transformation network ■ PointNet++	
Day5	Statistical and experimental design	■ 說明試驗設計在植物表行研究的重要性 ■ Longitudinal data 中的 outlier 如何偵測 ■ 視覺化分析表型分析平台數據之空間分佈 ■ Longitudinal data 資料分析	■ Time series and repeated measurement data ■ statgenHTP R package ■ SpATS R package	■ R script 程式碼

肆、參訪 NPEC 與研究人員交流

NPEC 主要任務為引領植物科學領域的創新研究，期望透過研究釐清決定植物性狀、環境、基因之間相互作用與調控機制，並提供研究人員和業界最先進的植物表型分析設施與服務，藉此開發永續農業解決方案，以保障糧食安全和減緩氣候變遷對於農業生產的影響。以下分別介紹 NPEC 內設施與功能以及本次參訪人員。

一、NPEC 設施與功能

為荷蘭最新穎之植物表型分析設施，主要有 6 個表型分析模組，其中模組 1 (Ecotron)、2 (Plant-Microbe Interaction Phenotyping) 和 3 (Multi-Environment Climate Chambers) 位於 UU 校區，模組 4 (High-Throughput Phenotyping Climate Chamber)、5 (GreenHouse Phenotyping) 和 6 (Open-Field Phenotyping) 則位於 WUR 校區。各模組主要功能與組成分述如下:

1. 模組 1: Ecotron

Ecotron 為小型模擬生態環境封閉系統 (圖 1)，其具備全方位的環境條件控制，並利用 RGB 相機記錄地上部和地下部生長，Ecotron 環境控制模式可以精確控制地上部溫度、土壤溫度、照度、濕度與降雨量等。透過 Ecotron 試驗，研究人員可以模擬農業和自然生態系統，以探討不同作物栽培組合具較佳生長競爭優勢，以及確定養分供給和土壤肥力維持的最佳平衡條件為何。

2. 模組 2: Plant-Microbe Interaction Phenotyping

Plant-Microbe Interaction Phenotyping 模組係由兩個獨立的表型分析裝置組成，分別為 Helios 和 Hades (圖 2)。Helios 為植物地上部表型分析裝置，透過裝設在環境控制空間之小型輸送帶系統，其配置精準灌溉系統，可放置數量 60 至 1,200 個大小不等盆栽 (250 ml 至 5 L)，植株最大高度為 100 cm，表型分析功能包含 3D 雷射掃描三角測量、RGB 影像、葉綠素螢光和 VNIR 高光譜成像。

Hades 為體外 (*in vitro*) 根部系統結構 (root architecture system, RSA) 表型分析系統，可高通量分析 2000 個培養皿，Hades 包括全自動填充培養皿與播種。應用範疇包含使用微生物或複合微生物組成處理根部，用以觀察根部微生物相對於 RSA 發展影響。表型分析功能 VNIR 高光譜成像、根系結構的量化分析。

3. 模組 3: Multi-Environment Climate Chambers

Multi-Environment Climate Chambers (圖 3) 該模組旨在研究植物如何在精確控制的條件下生長以及它們如何適應壓力，模擬各種溫度 (-5°C 至 40°C)、濕度 (35-90% RH) 以及光照強度 (最大強度 2000 μmol m^{-2} s^{-1}) 的氣候條件。於 NPEC，共計有 15 個 Multi-Environment Climate Chambers，以使研究人員可透過足夠的獨立、重複試驗，結論具統計意義的差異植物表型分析結果。

4. 模組 4: High-Throughput Phenotyping Climate Chamber

於 NPEC 共計有 5 個 High-Throughput Phenotyping Climate Chamber (圖 4)，其中 3 個面積大小約為 20 m^2 和 2 個約 15 m^2，每個模組可容納 300 到 2200 株植物，在此密閉式表型分析模組中空間配備先進的測量設備，包含植物 LED 照明系統，使研究人員能夠改變光質量和光量條件，包括模擬自然日光，最大光強度高達最大強度 2000 $\mu mol\ m^{-2}\ s^{-1}$，溫度為-4 至 42°C，相對濕度介於 40% 至 85%之間，並可外加 CO_2。表型分析功能則包含 RGB、高光譜、葉綠素螢光、熱成像等分析功能。使用該模組可快速了解不同植物品種、基因型在不同氣候條件下的生理表現與生長差異，目前該模組已應用在不同基因型阿拉伯芥 (*Arabidopsis thaliana*) 光合作用的研究以及番茄 (*Solanum lycopersicum*) 和馬鈴薯 (*Solanum tuberosum*) 等作物的根系在鹽害逆境的反應。

5. 模組 5: GreenHouse Phenotyping

溫室型研究模組裝設輸送帶 (conveyor) 系統以將植物運送到影像擷取站地上部表型分析，分析功能包含 3 維立體結構重建、葉綠素螢光、熱像儀 (圖 5)。除了輸送帶系統外，該模組亦建置天車 (gantry) 系統，以將表型分析設備移動至植物上方進行表型分析，因此研究人員也可以在不移動植物的情況進行表型分析，天車表型分析功能包含 3 維立體結構重建以及熱像儀。

6. 模組: Open-Field Phenotyping

Open-Field Phenotyping 為配備 RGB、熱成像、3 維立體結構重建和高光譜相機的車輛，此陸上型模組稱 TraitSeeker (圖 6)，設計為針對 1.5 m 至 2.0 m 跨距進行田間作物表型分析，TraitSeeker 具備巡航控制來進行可重複量測，並配備 2 個高光譜感測器、2D LiDAR、RGB 等表型分析功能，在影像擷取組件配置專門的遮蔽罩與補光系統，已盡可能確保影像資料不受天氣條件影響，所有資料均透過機載 GPS/INS (Global Positioning System/ Integrating Inertial Sensors) 單元進行位置和傾斜度地理參考設定。

二、研究人員與研究重點

1. Rick van de Zedde

Rick van de Zedde 是 NPEC 專案經理，同時自 2004 年於 WUR 擔任表型體學和自動化部門的高級科學家與業務開發人員，2020 年擔任國際植物表型分析網路 (International Plant Phenotyping Network, IPPN) 理事會成員。Rick 的研究領域為人工智慧，研究重點是影像處理和機器人開發。

Rick 在本次參訪過程引領介紹位於 WUR 的 NPEC 表型分析模組，說明模組 4、5、6 的功能與應用。此外，Rick 也說明了成立 NPEC 的背景，荷蘭政府耗時約 10 年完成規劃、建置完畢 NPEC，整體經費約 2,200 萬歐元，方得以讓 NPEC 在 2022 年 9 月正式公開營運。在營運規畫部分，Rick 說明大方向是讓越多人可以觸及使用表型分析儀器、分析方法作為研究工具 (圖 7)。第一階段期望以 5 到 10 年時間，首先由技術開發人員發展符合產業需求的植物表型解決方案，

再以 5 到 10 年時間將表型解決方案擴散到早期適用族群如大學院校、農業研究單位。隨後再以 5 到 10 年時間將表型解決方案擴散到讓主要農業生產單位、農企業使用，最終期望整體農業生產體系皆可應用植物表型體學到實際栽培管理系統。因此，NPEC 扮演一將政府資源轉化提升農業生產效率的研究中心，同時不斷透過與外部合作主動探索研究人員需求以及產業所面臨問題 (圖 8)，並因應需求與問題開發、提供植物表型體學解決方案或原型機，藉此強化荷蘭整體農業經爭力。

2. Gert Kootstra

Gert Kooststra 為 WUR 副教授，專精於農業應用電腦視覺分析與機器人開發。其研究主題為穩健感知 (Robust perception) 與主動感知 (Active perception)。由於 DL 徹底改變機器視覺研究領域的深度，與傳統的影像處理演算法相比，DL 可處理目標物件外觀在不同光照條件所產生的差異，Gert Kooststra 的研究主題之一是研究與開發可穩健分析受外在環境影響表型資料的 DL 架構。此外，為處理實際作物栽培經常面臨目標物件具有遮擋、陰影等問題，Gert Kooststra 另一研究主題是使機器人具備主動感知環境資訊能力，使機器人能夠自行決定合適的影像擷取視角、距離，以獲取具備足夠品質的表型影像資訊。

3. Lammert Kooistra

Lammert Kooistra 教授為 WUR 遙測團隊主要負責人，其研究主題是整合各種感測方法、元件用以開發永續植物生產系統。應用的技術包含 UAV 和衛星感測、主動冠層測量、實地觀測、ML 以及作物生長模型，期望可以改善農業資源管理並保護環境。目前的研究挑戰是利用新世代感測器優越的空間、時間和光譜資料擷取能力，據此探索感興趣的植物表型、性狀，包含田間植物高度到複雜的光合作用相關性狀，如太陽誘導螢光 (solar-induced fluorescence, SIF)。

4. Tom Theeuwen

Tom Theeuwen 為 WUR 博士後研究員，研究主題以馬鈴薯為主，並嘗試利用 NPEC 各項模組進行馬鈴薯種原耐旱研究。過往研究已利用 High-Throughput Phenotyping Climate Chamber 模組探討阿拉伯芥胞質雜種 (reciprocal cybrids) 族群的表型差異性，包含光合作用效率差異、生長勢差異，並進一步提出在地馴化阿拉伯芥如何演化出適應異地氣候的解釋。Tom Theeuwen 同時也分享使用 NPEC 各模組的經驗與優、缺點分析。

5. Gerrit Polder

Gerrit Polder 博士專長為光譜成像技術，並應用於測量植物材料中的生

化物質，由 2004 年開始，即專注於發展機器視覺和機器人的農業應用解決方案。近期研究以應用高光譜資料結合 DL 模型偵測馬鈴薯病害 (potato virus Y) 以及開發溫室內計數番茄果實數目的 DL 方法。此外，Gerrit Polder 博士也從事利用化學計量學 (chemometrics) 常用的多變數分析方法結合 DL 模型以進行高光譜影像植物型態差異的語義分割研究。

伍、心得與建議事項

本次藉由參加 WUR 表型分析課程，在植物表型影像分析過程的原理、邏輯及方法以及應注意事項等部分，均獲得相當大程度助益。此外，透過參訪 NPEC 以及和研究人員進行交流，對於未來我國表型體學研究發展以及國家表型分析中心的營運也有不同想法。以下將心得與建議摘要條列如下:

一、植物表型資料收集

植物表型體學研究方法主要利用感測器或相機收集資料，透過資料擷取、處理與分析後數據，再與植物目標性狀進行關聯分析。在依據研究目的決定欲分析之表型性狀（形態、生理變異）時，必須確認所使用的表型分析方法可確實收集植物所反應的性狀變化，研究人員除了對於植物生理學、生物化學具備扎實的知識、學理基礎外，對於表型儀器、設備的操作和參數設定，亦須確實了解，如何正確操作表型儀器、設備為研究人員進行表型體學研究的第一個痛點。常見表型感測器或相機包含葉綠素螢光儀、高（多）光譜、熱影像、一般光學相機、深度相機等，儀器、設備是否合理操作與合適的環境條件（光照強度、光質、風擾)，對於影像資料品質亦具相當大影響，植物本身結構如過於複雜，器官（葉、果實）重疊也會造成收集影像資料的困難。故在進行表型分析試驗前，務必事先完備資料收集策略，例如在何種環境（環控溫室、大田)，作物生長階段（幼苗、成初、始花期)，作物器官（根、莖、葉、花、果實)，作物結構位階（冠層、特定葉位)，搭配合適的相機或感應器，如此方得有機會收集背景干擾少、標的物件清晰的影像資料，有利於後續資料分析進行。

二、植物表型資料分析

由植物表型體學研究結果所產製的資料大致可分為影像、數值與點雲資料格式，如何進行表型資料分析為研究人員進行表型體學研究的第二個痛點。以本次所參加的表型影像分析課程為例，主要以 Python、R、ImageJ 軟體配合程式碼進行教學示範，並進一步說明深度學習在表型資料分析的應用。在現今科技進步相當快速的狀況下，建議研究人員皆需要具備基礎的程式語言編寫例如 Python、R 以及使用 ImageJ、CloudCompare 等開源、免費軟體能力，以備自行使用合理的處理流程或深度學習以進行資料剖析、模型配適等。

三、建構研究室階段之表型資料收集與分析能力

依據執行表型分析研究所在場域、可分析數量與整合規模，植物表型分析設備、系統可分為實驗室型、溫室型、大田系統等，所分析的植物表型可分為肉眼可觀察（外觀形態、結構）與不可觀察（光譜、熱影像、葉綠素螢光）兩種分類。目前國家表型分析中心包含溫室型（利用輸送帶移動植物）以及大田系統（利用天車移動感應器）等兩種設施，其中溫室型表型設施可進行外觀形態、結構、高光譜、熱影像、葉綠素螢光分析，大田系統則可進行外觀形態、結構、多光譜分析。

一般而言，表型分析研究在初期需要在實驗室階段確認分析方法是否適用，如前述第一及第二個表型研究痛點，會在實驗室階段建立資料收集與分析

方法並確認可行後，再逐漸進展到溫室、大田進行較大規模之表型分析試驗。故現階段而言，國家表型分析中心目前導入國外表型分析設施，基於空間與硬體技術限制，會有其適用的作物種類、大小，雖可進行高通量、快速、客觀擷取表型性狀分析，但若無建置實驗室階段的表型資料收集、分析能力，國家植物表型分析中心在未來研究應用與營運規劃將會受限製造商所提供之操作標準流程，研究人員無法了解確實掌握表型資料收集與分析的重點與意義，無法彈性擴充應用國家表型分析中心兩項主要分析設施，國家表型分析中心研究效能無法有效發揮，故建構研究室階段之表型資料收集與分析能力是為首要穩固國家表型分析中心未來研究應用與發展的重要基礎。

四、植物表型體學研究發展方向

1.妥善利用農業試驗所既有農業研究優勢

農業試驗所國家種原中心具有豐富的作物遺傳資源，利用高通量基因型分析 (high-throughput genotyping)，大豆 (*Glycine max*) 已自 21,698 個收集系，建立核心收集 (core collection) 族群 (256 個收集系)。番茄 (*Solanum lycopersicum*)、番椒 (*Capsicum annuum*)、甜瓜 (*Cucumis melo*) 業已建構由 292 個、250 個、96 個收集系組成的核心收集族群。十字花科蔬菜包含甘藍 (*Brassica oleracea*)、青花菜 (*B. oleracea* var. italica)、花椰菜 (*B. oleracea* var. botrytis) 利用小孢子組織培養技術已可建立雙單倍體 (double haploid, DH) 族群。玉米 (*Zea mays*) 利用 Gal-S 基因之玉米單倍體亦可順利誘導產生 DH 族群。上述多樣性遺傳資源，依據政策、產業需求，相當適合應用各項高通量目標性狀 (耐環境逆境、抗病、抗蟲) 之表型分析方法，配合基因體選拔 (genomic selection)、全基因組關聯分析 (genome-wide association study, GWAS) 加速篩選具備目標性狀之前育種品系 (pre-breeding line)，有效串聯基因型 (genotype, G)與表型 (phenotype, P) 的關係 (G to P)。尤以如可加入轉錄體學、代謝體學分析等跨體學應用，可加速特定性狀、富含特定成分等作物育種速度，深化我國茄科、葫蘆科與十字花科種苗產業競爭力。

2.扣合農業政策

我國農業政策目前以淨零碳排、氣候變遷調適為主，符合淨零排放之作物目標性狀包含建立高氮肥使用效率、高光合作用效率，有助於氣候變遷調適之作物性狀則包含耐逆狀非生物性逆境以及根系發展旺盛等性狀表現。在此部分的研究需要建立高 (多) 光譜、葉綠素螢光、氣孔導度以及根部系統結構 (root system architecture, RSA) 等表型分析方法，配合功能性結構植物模型 (functional structure plant model, FSPM)的應用，藉以篩選具備低氮肥需求、快速生長、耐非生物逆境以及根系強健性狀之品系。

五、表型中心營運規劃

本次參訪 NPEC 期間，Dr. Rick 表示規劃 NPEC 前置作業時間約 10 年，總經費耗費約 220 萬歐元，總計有 6 種主要設施，參與研究及營運的研究人員主

要為大專院校內的教授，常駐研究人員及技術人員約有 25 位左右，專業領域包含影像分析、農機、植物生理、資訊工程、植物栽培等。NPEC 設立的任務是為了解決荷蘭農業相關產業問題而建立的研究中心，主要希望植物表型體學研究、應用與發展結果最終可轉譯導入實質溫室生產體系。營運策略目標設定為儘可能將表型體技術由學校、研究單位逐漸擴散至農民、農企業使用。達成營運策略目標的執行方針是先了解農民、農企業問題與需求，再藉由 NPEC 植物表型體學基礎研究階段進展至應用研究階段，最終由應用研究階段開發符合農民、農企業需求之原型機或解決方案，解此增加農民、農企業收益。

參考 NPEC 營運規劃內容，並考量我國與荷蘭國情、農業發展與文化差異，關於國家植物表型分析中心營運規劃內容與方向建議如下:

1.定位與任務

國家植物表型分析中心定位應不同於農業部轄下試驗改良場所，所執行之研究、推廣業務應以較上位性高度明確訂定中心主要任務。例如，農業部農業試驗所既有寶貴遺傳多樣性資源，應結合既有種原與育種優勢，結合次世代定序基因型分析、高通量表型分析、跨體學研究以及新興育種技術等，以加速育成符合農業政策、農民、農企業需求之中間品系。建議應先確立國家植物表型分析中心之定位與任務。

2.營運經費

國家植物表型分析中心硬體、設施建置係由中長程公共建設計畫支應，然而後續營運經費來源與使用仍有待籌措、規劃。以 NPEC 為例，目前其主要營運經費是由 Dutch Science Organization、WUR 與 UU 所支應，如有外來單位如私人公司或其他學術單位使用，則以使用者付費原則進行收費。整體而言，NPEC 是以國家經費支持為主，經費開銷主要在於溫室輸送帶運作、模擬逆境之空調所需維護、電力支出以及人力費用為主。關於營運經費之規劃如條列如下建議:

(1).基本費用估算

國家植物表型分析中心將於 113 年後完成硬體、設施建置，基本費用包含設施運作所需電費、維護 (歲修)、人力支出等費用應於 112 年完成評估作業，以作為申請經費依據。

(2).研究發展費用

此部分經費支應項目包含:

- 建構研究室階段之表型資料收集與分析能力相關研究
- 國家表型分析中心內進行的各項研究
- 符合農民、農企業需求之表型體研究

建議按現行農業部科技計畫申請審核程序進行經費申請。

(3).外來經費

除上述基本與研究經費仍需由國家支應外，部外單位如私人公司、大專院校可透過與本部農業試驗所以雙邊合作、委託辦理或研提計畫方式爭取外

來經費挹注於國家植物表型分析中心，描述如下:

- 雙邊合作:透過本部農業試驗所現行對外合作計畫申請流程，合作雙方明訂分工內容，經費自籌，成果由雙方共享。
- 委託辦理:部外單位支付合理經費委託本部農業試驗所進行試驗研究，依據訂定委託契約內容，由國家植物表型分析中心執行委託案件。
- 研提計畫:參考 IPPN (International Plant Phenotyping Network) 作法，歐盟政府保留經費供大學院校申請使用 IPPN 內各項表型分析設施，申請者需要研提計畫書，提供給表型分析設施主管單位審核，再由表型分析設施主管單位依據計畫書內容擇優提供表型分析服務。據此，或許可參照 IPPN 作法，由農業部每年保留固定經費，供部外單位研提計畫競爭，計畫案再由農業試驗所進行審核，擇優於國家植物表型分析中心執行計畫。

陸、總結

本次出國參加植物表型分析課程與參訪 NPEC 著實獲益良多，在表型資料收集與分析的技術能力有相當大的提升，參訪 NPEC 後，對於規劃國家植物表型分析中心的營運內容，也激發個人許多不一樣的想法。整體而言，未來若欲妥善發揮國家植物表型分析中心研究效能，尚需足夠專業人力配置，並透過數年經驗、技術積累，方有機會將植物表型體學開始落實於研究發展、產業應用。例如本次參訪的 NPEC，常駐研究人員含技術員至少逾 25 位，從籌措經費到完成設施建置花費約 10 年，迄今方有能力開展各項植物表型體學研究，足見人力、經費與時間皆為國家植物表型分析中心得發揮預期研究效能的關鍵成敗因素。因此，於現階段國家植物表型分析中心尚未完工時期，可先著重於表型體收集、資料分析與詮釋、應用等相關基礎能力建構。雖我國植物表型體分析中心目前在人力、經費尚無法如國外表型體中心具備合理配置，然而綜觀我國產、官、學界所既有深厚農業研究、電腦科學研究、機電整合等優勢，國家植物表型分析中心在開展各項研究計畫初期也許可透過國家資源導入，扮演一整合各研究、應用領域的平台，而不僅只是提供設施進行研究、分析服務而已，如此也可以加速我國植物表型體學發展。期望未來國家植物表型分析中心具備厚實研究能量以上承農業政策，下接農民、農企業需求。

陸、照片

圖 1. NPEC 中心 Ecotron 模組。

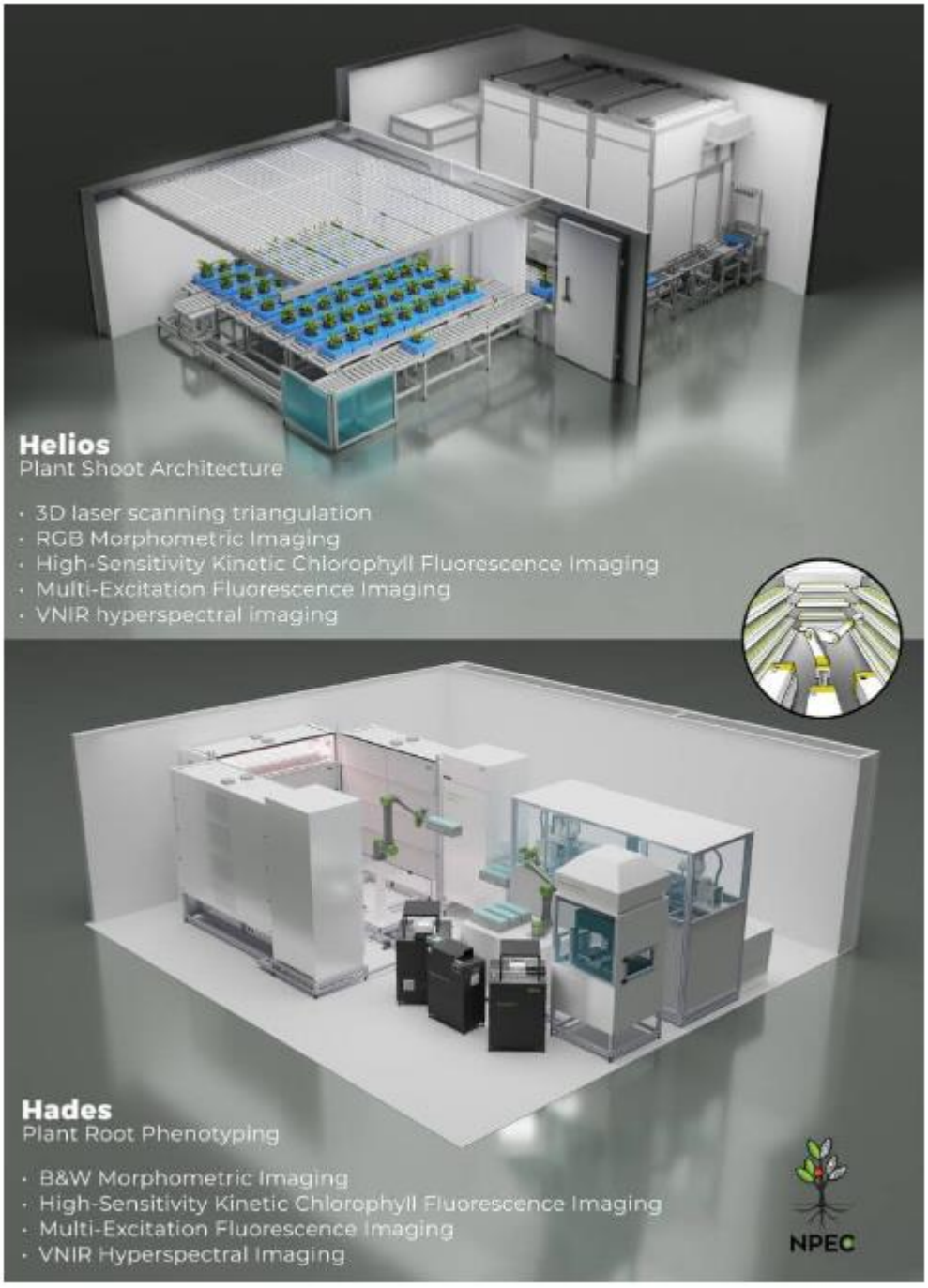

圖 2. NPEC 中心 Plant-Microbe Interaction Phenotyping 模組。

圖 3. NPEC 中心 Multi-Environment Climate Chambers 模組。

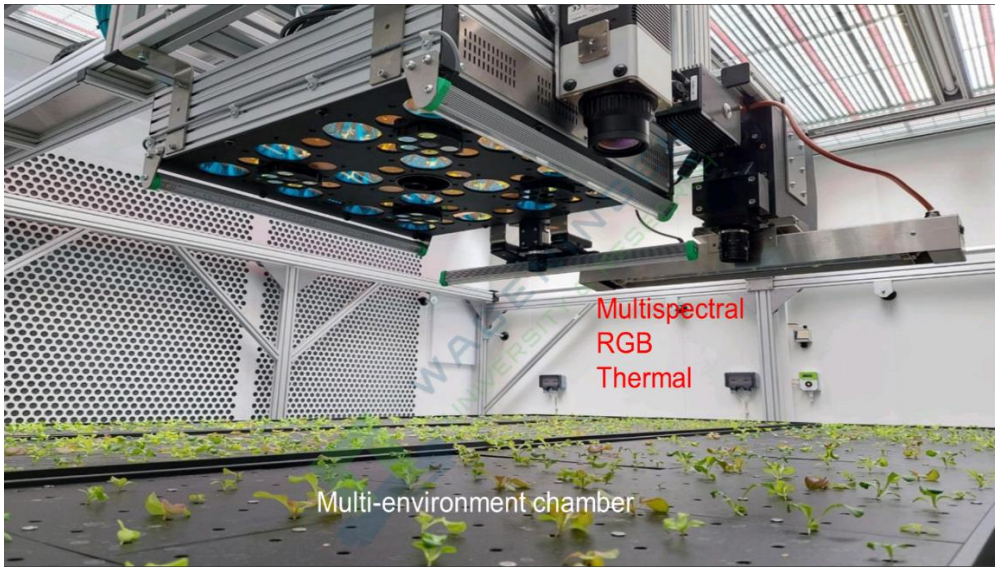

圖 4. NPEC 中心 High-Throughput Phenotyping Climate Chamber 模組。

圖 5. NPEC 中心 GreenHouse Phenotyping 模組。

圖 6. NPEC 中心 TraitSeeker 模組。

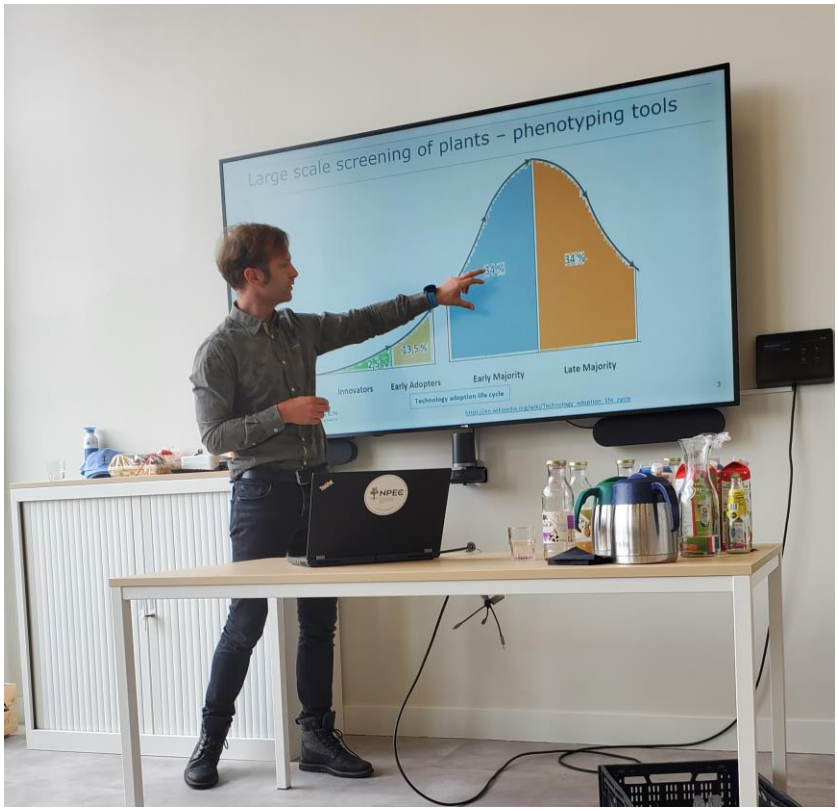

圖 7. Rick van de Zedde 說明 NPEC 中心營運規劃。

柒、附件

附錄一、植物表型體分析課程，課程介紹。

WAGENINGEN
UNIVERSITY & RESEARCH

Summer School

Image Analysis for Plant Phenotyping

Outline and topics
In this program, a mixture of lectures from experts from Wageningen University & Research and leading international experts in this domain, are combined with hands-on training. During the Summer School there is ample time to discuss the issues of importance to your company/institutions with the experts. The following subjects will be addressed:

- Image acquisition
- Noise filtering
- Segmentation and image shape features
- Machine learning
- Deep learning
- 3D vision
- Imaging, data and practicals

Day 1: Monday 3 July 2023
Introduction day + image acquisition
- Welcome and introduction to the Summer School
- Introduction to image analysis and phenotyping
- Image sensors and acquisition
- Assignments and practice with project cameras

Day 2: Tuesday 4 July 2023
Noise filtering
- Noise and image enhancement with assignments
- Industry perspective
- Guided tour research facilities at NPEC (Netherlands Plant Eco-phenotyping Centre)

Day 3: Wednesday 5 July 2023
Segmentation and image shape features
- Introduction to segmentation and assignments
- Machine learning
  - Classical Machine learning: K-MEans, Linear discriminant, svn, neural network, supervised/ unsupervised, clustering
  - Lab session image shape features and machine learning

Day 4: Thursday 6 July 2023
Neural networks and deep learning
- Deep learning
- Lab segmentation II (segmentation using neural networks) with Python
- Hyperspectral imaging
- 3D vision
- 3D interactive Plant phenotyping Marvin
- Lab session 3D data analysis/interpretation

Day 5: Friday 7 July 2023
Imaging, data and practicals
- Imaging and data management
- Data analysis and application
- Questions & answers
- Evaluation and certification ceremony

附錄二、程式碼

■ Day1_3

✓ noise_filtering_part_1

```
nr_leaves = 10
nr_flowers = 3
print("This plant has", nr_leaves, "leaves and", nr_flowers, "flowers")
import time
for i in range(5):
  time.sleep(1)
  print("#", end = '')
!git clone https://git.wur.nl/koots006/summerschool-image-processing-for-plant-phenotyping.git data
import numpy as np              # Numpy is a library to do calculations and work with matrices
import cv2                      # OpenCV is a library containing many computer-vision / image-
processing methods
import matplotlib.pyplot as plt # We will use Matplotlib to make plots
from mpl_toolkits.axes_grid1 import make_axes_locatable
# Set default size for figures so that images are large enough
plt.rcParams['figure.figsize'] = [20, 10]
#function to display an image using RGB color format.
def
imshow_rgb(img_bgr):                                                                    #
read images, plt.imshow builds a window with the image in it, plt.show
    img_rgb = cv2.cvtColor(img_bgr,cv2.COLOR_BGR2RGB)
    plt.imshow(img_rgb)
# Function to make colorbars appear nicer
def colorbar(mappable):
    ax = mappable.axes
    fig = ax.figure
    divider = make_axes_locatable(ax)
    cax = divider.append_axes("right", size="5%", pad=0.05)
    return fig.colorbar(mappable, cax=cax)
# Create two numpy arrays for the signal and the kernel
signal = np.array([[0,0,1,0,0]], dtype=np.uint8)
kernel = np.array([[1,2,3]])
```

```
# Apply the convolution kernel.
# The value -1 (ddepth) make that the result has the same type (int, float, ...) as the souerce
filtered_signal = cv2.filter2D(signal, -1, kernel)
print("The signal: ", signal)
print("The kernel: ", kernel)
print("The result: ", filtered_signal)
def create_averaging_kernel_1D(kSize):           #function to build a 1d averaging kernel
    if not (kSize%2==0):
        kernel0 = np.ones(shape=(1,kSize))/kSize
        return kernel0
    else:
        print('Not allowed kernel size')
averaging_kernel = create_averaging_kernel_1D(3)    #Create averaging kernel with size 3.
print ("Kernel:", averaging_kernel)
print("Sum of the weights:", averaging_kernel.sum())
signal = np.array([[3,15,6,9]], dtype=float)      #create a dummy signal
#apply setting valid on the signal
filtered_signal = cv2.filter2D(signal, -1, averaging_kernel, borderType=cv2.BORDER_CONSTANT)
# Set the printing precision to 1 decimal to keep the print clear
np.set_printoptions(precision=2)
print("The signal: ", signal)
print("The kernel: ", averaging_kernel)
print("The result: ", filtered_signal)
#Create a distorted sine wave
n = 200;
x = np.linspace(0,2*np.pi, n).reshape((1,n)) # A 1x200 matrix
y = np.cos(x) +0.3*np.random.randn(1, n)  # A 1x200 matrix containing the noisy sine
plt.figure(figsize=(10,8))
plt.plot(x[0,:], y[0,:])         # Plot the first (and only) row of the matrices
plt.xlabel('x')
plt.ylabel('y')
plt.title('A distorted sine wave')
plt.show()
# Create the two kernels
averaging_kernel_1 = create_averaging_kernel_1D(3) # A kernel that sums up to one
averaging_kernel_2 = create_averaging_kernel_1D(9) # A kernel that sums up to one
print("Filter 1:",averaging_kernel_1)
```

```
print("Filter 2",averaging_kernel_2)
# Apply the two kernels to the signal y
y_filtered_1 = cv2.filter2D(y, -1, averaging_kernel_1, borderType = cv2.BORDER_CONSTANT)
y_filtered_2 = cv2.filter2D(y, -1, averaging_kernel_2, borderType = cv2.BORDER_CONSTANT)
############
# Plot the original signal and the two filtered signals
plt.figure(figsize=(16,8))
plt.plot(x[0,:], y[0,:], 'gray', label="signal")
plt.plot(x[0,:], y_filtered_1[0,:], 'b', label="filter1, size %d" % averaging_kernel_1.size)
plt.plot(x[0,:], y_filtered_2[0,:], 'r', label="filter2, size %d" % averaging_kernel_2.size)
plt.legend()
plt.show()
def create_averaging_kernel_2D(kSize):           #function to build a 1d averaging kernel
    if not (kSize%2==0):
        kernel = np.ones(shape=(kSize,kSize))/(kSize*kSize)
        return kernel
    else:
        print('Not allowed kernel size')
averaging_kernel_2D = create_averaging_kernel_2D(5)
# Set the printing precision to 2 decimal to keep the print clear
np.set_printoptions(precision=2)
print(averaging_kernel_2D)
#Function to show one or multiple images
def show_images(images, cols=2):
  figwidth = 20;
  hw_ratio = images[0][0].shape[0]/images[0][0].shape[1]
  rows = int(np.ceil(len(images) / cols))
  figheight = figwidth * hw_ratio /cols * rows
  plt.figure(figsize=(figwidth,figheight))
  for i, image in enumerate(images):
      plt.subplot(rows,cols,i+1)
      plt.imshow(image[0], cmap='gray', vmin=0, vmax=255)
      plt.title(str(image[1]))
      plt.xticks([]), plt.yticks([])
  plt.show()
# Open a gray-scale image
img = cv2.imread('data/noise_filtering/leaf.jpg', cv2.IMREAD_GRAYSCALE)  # Load image in grayscale
# Add some noise to the image
```

```
noise = np.random.randn(img.shape[0],img.shape[1])
noise = 10*np.sign(noise)*noise**2
img_noise = np.clip(img + noise, 0, 255).astype('uint8')
# Filter the image with the 5x5 averaging convolution filter
img_filtered = cv2.filter2D(img_noise,-1,averaging_kernel_2D)             # apply filtering
# Show the original image and filtered image
show_images([(img,"Original image"),(img_noise,"Noisy image"),(img_filtered,"Averaging filter applied
to noisy image")])
# Perform image blurring (averaging filter) with a 5x5 kernel
blur = cv2.blur(img_noise,(5,5))
# Show the original image and filtered image
show_images([(img, "Original image"),(img_noise,"Noisy image"),(blur, "Blurred image")])


from mpl_toolkits.mplot3d import Axes3D  # noqa: F401 unused import
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
# A function to get a 2D Gaussian kernel
def getGaussianKernel(kSize):
    gaussian_kernel_1D = cv2.getGaussianKernel(kSize, 0)  # The 0 means that the sigma of the Gauss is
determined automatically
    gaussian_kernel_2D = gaussian_kernel_1D * np.transpose(gaussian_kernel_1D)
    return(gaussian_kernel_2D)
kernel_size = 25
gaussian_kernel = getGaussianKernel(kernel_size)
#############
## PLot in 3D
x = np.arange(0, 25)
y = np.arange(0, 25)
X, Y = np.meshgrid(x, y)
Z = gaussian_kernel
fig = plt.figure(figsize=(15,8))
ax = plt.subplot(projection='3d')
# Make data.
# Plot the surface.
surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)
# Customize the z axis.
#ax.set_zlim(-1.01, 1.01)
```

```
ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))
# Add a color bar which maps values to colors.
fig.colorbar(surf, shrink=0.5, aspect=5)
plt.show()
# Apply Gaussian blurring
kernel_size = (7,7)
blur = cv2.GaussianBlur(img_noise,kernel_size,0)
show_images([(img, "original"),(img_noise,"noisy image"),(blur, "Gaussian blurred image")])
```

- ✓ noise_filtering_part_2

```
!git clone https://git.wur.nl/koots006/summerschool-image-processing-for-plant-phenotyping.git data
import numpy as np              # Numpy is a library to do calculations and work with matrices
import cv2                      # OpenCV is a library containing many computer-vision / image-
processing methods
import matplotlib.pyplot as plt # We will use Matplotlib to make plots
from mpl_toolkits.axes_grid1 import make_axes_locatable
#Function to show one or multiple images
def show_images(images, cols=2):
  figwidth = 20;
  hw_ratio = images[0][0].shape[0]/images[0][0].shape[1]
  rows = int(np.ceil(len(images) / cols))
  figheight = figwidth * hw_ratio /cols * rows
  plt.figure(figsize=(figwidth,figheight))
  for i, image in enumerate(images):
      plt.subplot(rows,cols,i+1)
      plt.imshow(image[0], cmap='gray', vmin=0, vmax=255)
      plt.title(str(image[1]))
      plt.xticks([]), plt.yticks([])
  plt.show()
# Open a gray-scale image
img = cv2.imread('data/noise_filtering/leaf.jpg', cv2.IMREAD_GRAYSCALE)  # Load image in grayscale
# Add some noise to the image
noise = np.random.randn(img.shape[0],img.shape[1])
noise = 10*np.sign(noise)*noise**2
img_noise = np.clip(img + noise, 0, 255).astype('uint8')
img_gaussian = blur = cv2.GaussianBlur(img_noise,(7,7),3)
```

```
img_median = cv2.medianBlur(img_noise, 5)                          #apply median filter with kernel
size 5
show_images([(img,"Original"), (img_noise, "Noisy image"), (img_gaussian, "Gaussian filter"),
(img_median, "Median filter")])
img_gray = cv2.imread('data/noise_filtering/practical_noise_small2.png',0)
import matplotlib.patches as patches
plt.figure(figsize=(20,15))
plt.imshow(img_gray, cmap='gray', vmin=0, vmax=255)
rect = patches.Rectangle((0, 500), 200, 100, linewidth=1, edgecolor='r', facecolor='none')
plt.gca().add_patch(rect)
plt.show()
img_gray_crop = img_gray[500:600,0:200]
plt.figure(figsize=(20,15))
plt.imshow(img_gray_crop, cmap='gray', vmin=0, vmax=255)
plt.show()
img_gray = cv2.imread('data/noise_filtering/practical_noise_small2.png',0)
img_gray_crop = img_gray[500:600,0:200]
k_size = 3
img_gaussian = cv2.GaussianBlur(img_gray_crop, (k_size, k_size), 0)
img_median = cv2.medianBlur(img_gray_crop, k_size)
show_images([(img_gray_crop, 'Original cropped image'), (img_gaussian, 'Result of the Gaussian blur
filter'), (img_median, 'Result of the median filter')])
# TODO: copy-paste code from above to perform Gaussian and median filtering on the full image
img_gray = cv2.imread('data/noise_filtering/practical_noise_small.png',0)
k_size = 7
img_gaussian = cv2.GaussianBlur(img_gray, (k_size, k_size), 0)
img_median = cv2.medianBlur(img_gray, k_size)
show_images([(img_gray, 'Original image'), (img_gaussian, 'Result of the Gaussian blur filter'),
(img_median, 'Result of the median filter')])
# Add some Gaussian noise
def addGaussianNoise(img, sigma):
    noise = sigma*np.random.randn(img.shape[0],img.shape[1])
    img_noise = np.clip(img + noise, 0, 255).astype('uint8')
    return img_noise
def getNoisyImage1(sigma):
  img_gray = 50*np.ones((150,200), dtype='uint8')
  img_gray[0:150, 0:100] = 200
  img_gray_noise = addGaussianNoise(img_gray, sigma)
```

```
  return img_gray_noise
def getNoisyImage2(sigma):
  img_gray = 50*np.ones((150,200), dtype='uint8')
  img_gray[0:150, 98:100] = 200
  img_gray[0:150, 49:50] = 200
  img_gray[0:150, 149:150] = 200
  img_gray_noise = addGaussianNoise(img_gray, sigma)
  return img_gray_noise
img_gray_noise = getNoisyImage1(10)
# Run the Gaussian and bilateral filter
spatial_sigma = 9
spectral_sigma = 15
k_size = spatial_sigma*3
img_gaussian = cv2.GaussianBlur(img_gray_noise,(k_size, k_size),spatial_sigma)
img_bilateral = cv2.bilateralFilter(img_gray_noise,k_size,spectral_sigma, spatial_sigma)
show_images([(img_gray_noise,'Original image'), (img_gaussian, 'Gaussian filtering'),
           (img_bilateral, 'Bilateral filtering')], 3)
img_gray = cv2.imread('data/noise_filtering/practical_noise_small.png',0)
# TODO: add code to apply the bilateral filter to img_gray
spatial_sigma = 9
spectral_sigma = 15
ksize = spatial_sigma*3
img_bilateral = cv2.bilateralFilter(img_gray_noise,k_size,spectral_sigma, spatial_sigma)
show_images([(img_gray,'Original image'),(img_bilateral,'Results of the bilateral filter')],1)
plt.show()
```

- ✓ noise_filtering_part_3

```
!git clone https://git.wur.nl/koots006/summerschool-image-processing-for-plant-phenotyping.git data
import numpy as np             # Numpy is a library to do calculations and work with matrices
import cv2                     # OpenCV is a library containing many computer-vision / image-
processing methods
import matplotlib.pyplot as plt # We will use Matplotlib to make plots
from mpl_toolkits.axes_grid1 import make_axes_locatable
#Function to show one or multiple images
def show_images(images, cols=2):
  figwidth = 20;
```

```
  hw_ratio = images[0][0].shape[0]/images[0][0].shape[1]
  rows = int(np.ceil(len(images) / cols))
  figheight = figwidth * hw_ratio /cols * rows
  plt.figure(figsize=(figwidth,figheight))
  for i, image in enumerate(images):
      plt.subplot(rows,cols,i+1)
      plt.imshow(image[0], cmap='gray', vmin=0, vmax=255)
      plt.title(str(image[1]))
      plt.xticks([]), plt.yticks([])
  plt.show()
# Load the image
img_gray = cv2.imread('data/noise_filtering/practical_noise_small.png',0)
# Filter the image using the bilateral filter
spatial_sigma = 9
spectral_sigma = 15
k_size = spatial_sigma*3
img_bilateral = cv2.bilateralFilter(img_gray,k_size,spectral_sigma, spatial_sigma)
### Threshold the image to get foreground and background
intensity_threshold = 128
# Apply the threshold to img_bilateral and set all pixels >= t to 255 and the rest to 0
_, plant_mask = cv2.threshold(img_bilateral,intensity_threshold,255,cv2.THRESH_BINARY)
# Show the result
show_images([(img_bilateral, 'Image'),(plant_mask, 'Binary mask from intensity threshold')])
# Perform local adaptive thresholding to keep the brighter parts of the image as foreground
# and the darker parts as background
local_threshold = 1
neighborhood_size = 13
local_average = cv2.blur(img_bilateral, (neighborhood_size,neighborhood_size))
plant_mask = 255*(img_bilateral > local_average + local_threshold).astype('uint8')
# Show the result
show_images([(img_bilateral, 'Image'),(plant_mask, 'Binary mask from intensity threshold')])
# Take a part of the plant mask image, so that we can better see the results
plant_mask_crop = plant_mask[0:100,350:600]
# Create a 5x5 structuring element
strel_size = 5
strel = np.ones((strel_size,strel_size),np.uint8)
# Apply erosion to the thresholded image
mask_eroded = cv2.erode(plant_mask_crop, strel)
```

```
# Apply dilation to the thresholded image
mask_dilated = cv2.dilate(plant_mask_crop, strel)
# Show the image
show_images([(plant_mask_crop,"Original mask"),
            (mask_eroded,"Using erosion"), (mask_dilated, "Using dilation")])
# Create a structuring element
strel = np.ones((5,5),np.uint8)
# Apply option 1: M' = dilate(erode(M)) - also called "opening"
mask_opened = cv2.dilate( cv2.erode(plant_mask_crop, strel), strel)
# TODO: Add code here to apply option 2: M1 = erode(dilate(M)) -  also called "closing"
mask_closed = cv2.erode( cv2.dilate(plant_mask_crop, strel), strel)
show_images([ (plant_mask_crop,"Original mask, M"), (mask_opened,"dilate(erode(M))"),
(mask_closed,"dilate(erode(M))") ])
# Create a structuring element
strel_size = (5,5)
strel = np.ones(strel_size,np.uint8)
# Apply opening and closing separately on the original mask
mask_opened = cv2.morphologyEx(plant_mask_crop, cv2.MORPH_OPEN, strel)
mask_closed = cv2.morphologyEx(plant_mask_crop, cv2.MORPH_CLOSE, strel)
# Apply first opening, then closing
mask_open_close = cv2.morphologyEx( cv2.morphologyEx(plant_mask_crop, cv2.MORPH_OPEN, strel),
cv2.MORPH_CLOSE, strel)
# Apply first closing, then opening
mask_close_open = cv2.morphologyEx( cv2.morphologyEx(plant_mask_crop, cv2.MORPH_CLOSE, strel),
cv2.MORPH_OPEN, strel)
# Show all results
show_images([ (plant_mask_crop,"Original mask"), (mask_opened,"opening"), (mask_closed,"closing"),
            (mask_open_close,"closing(opening(M))"), (mask_close_open,"opening(closing(M))")])
```

✓ noise_filtering_part_4

```
!git clone https://git.wur.nl/koots006/summerschool-image-processing-for-plant-phenotyping.git data
import numpy as np              # Numpy is a library to do calculations and work with matrices
import cv2                      # OpenCV is a library containing many computer-vision / image-
processing methods
import matplotlib.pyplot as plt # We will use Matplotlib to make plots
from mpl_toolkits.axes_grid1 import make_axes_locatable
```

```
#Function to show one or multiple images
def show_images(images, cols=2):
  figwidth = 20;
  hw_ratio = images[0][0].shape[0]/images[0][0].shape[1]
  rows = int(np.ceil(len(images) / cols))
  figheight = figwidth * hw_ratio /cols * rows
  plt.figure(figsize=(figwidth,figheight))
  for i, image in enumerate(images):
      plt.subplot(rows,cols,i+1)
      plt.imshow(image[0], cmap='gray', vmin=0, vmax=255)
      plt.title(str(image[1]))
      plt.xticks([]), plt.yticks([])
  plt.show()
# Define a simple mask
simple_mask = np.array([[0, 1, 0, 1, 0],
                        [1, 1, 0, 1, 0],
                        [1, 1, 0, 0, 1],
                        [1, 0, 0, 1, 1],
                        [0, 0, 0, 1, 1]],dtype='uint8')
# Display the mask
plt.figure(figsize=(5,5))
plt.imshow(simple_mask, cmap='gray')
plt.show()
from random import random
from matplotlib import colors as cl
coco_nr, coco_labels = cv2.connectedComponents(simple_mask, connectivity=4)
# Plot the connectect componenents using a random color map
colors = [(0,0,0)] + [(random(),random(),random()) for i in range(coco_nr)]
new_map = cl.LinearSegmentedColormap.from_list('new_map', colors, N=coco_nr+1)
plt.figure(figsize=(5,5))
plt.imshow(coco_labels, cmap=new_map)
plt.show()
# Get the connected components with some basic statistics
coco_nr, coco_labels, coco_stats, coco_centroids = \
    cv2.connectedComponentsWithStats(simple_mask, connectivity=4, ltype=cv2.CV_32S)
# Print the statistics
for i, coco_stat in enumerate(coco_stats):
    bbox_x = coco_stat[cv2.CC_STAT_LEFT]
```

```
    bbox_y = coco_stat[cv2.CC_STAT_TOP]
    bbox_w = coco_stat[cv2.CC_STAT_WIDTH]
    bbox_h = coco_stat[cv2.CC_STAT_HEIGHT]
    coco_size = coco_stat[cv2.CC_STAT_AREA]
    if(i==0):
        print("Background; bounding box, top-left corner: (x:%d,y:%d), width: %d, height: %d; size: %d"
%
            (bbox_x, bbox_y, bbox_w, bbox_h, coco_size))
    else:
        print("Coco",i, "; bounding box, top-left corner: (x:%d,y:%d), width: %d, height: %d; size: %d"
%
            (bbox_x, bbox_y, bbox_w, bbox_h, coco_size))
# Plot the connectect componenents using a random color map
colors = [(0,0,0)] + [(random(),random(),random()) for i in range(coco_nr)]
new_map = cl.LinearSegmentedColormap.from_list('new_map', colors, N=coco_nr+1)
plt.figure(figsize=(5,5))
plt.imshow(coco_labels, cmap=new_map)
plt.show()
# A function to filter the connected components based on a minimum size
def filterCocoMinSize(coco_nr, coco_labels, coco_stats, coco_centroids, min_size):
    # Filter the connected components. Make a new list with the cocos that have the required minimum
size
    new_coco_nr = 1
    new_coco_labels = np.zeros(coco_labels.shape, dtype=coco_labels.dtype)
    new_coco_stats = coco_stats[0]
    new_coco_centroids = coco_centroids[0]

    for i, coco_stat in enumerate(coco_stats):
        # Skip background
        if(i==0):
            continue
        # Check size and add to new list if correct
        coco_size = coco_stat[cv2.CC_STAT_AREA]
        if(coco_size >= min_size):
            new_coco_labels[coco_labels==i] = new_coco_nr
            new_coco_nr += 1
            new_coco_stats = np.vstack((new_coco_stats,coco_stat))
            new_coco_centroids = np.vstack((new_coco_centroids, coco_centroids[i]))
```

```
    return new_coco_nr, new_coco_labels, new_coco_stats, new_coco_centroids
# Get the connected components with some basic statistics
coco_nr, coco_labels, coco_stats, coco_centroids = \
    cv2.connectedComponentsWithStats(simple_mask, connectivity=4, ltype=cv2.CV_32S)
# Filter the cocos on size
min_size = 4
new_coco_nr, new_coco_labels, new_coco_stats, new_coco_centroids = \
    filterCocoMinSize(coco_nr, coco_labels, coco_stats, coco_centroids, min_size)
# Plot the filtered cocos
colors = [(0,0,0)] + [(random(),random(),random()) for i in range(coco_nr)]
new_map = cl.LinearSegmentedColormap.from_list('new_map', colors, N=coco_nr+1)
plt.figure(figsize=(5,5))
plt.imshow(new_coco_labels, cmap=new_map)
plt.show()
# Load the image
img_gray = cv2.imread('data/noise_filtering/practical_noise_small.png',0)
# Filter the image using the bilateral filter
spatial_sigma = 9
spectral_sigma = 15
k_size = spatial_sigma*3
img_bilateral = cv2.bilateralFilter(img_gray,k_size,spectral_sigma, spatial_sigma)
# Perform local adaptive thresholding to keep the brighter parts of the image as foreground
# and the darker parts as background
local_threshold = 1
neighborhood_size = 13
local_average = cv2.blur(img_bilateral, (neighborhood_size,neighborhood_size))
plant_mask = 255*(img_bilateral > local_average + local_threshold).astype('uint8')
# Get the connected components with some basic statistics
coco_nr, coco_labels, coco_stats, coco_centroids = \
    cv2.connectedComponentsWithStats(plant_mask, connectivity=8, ltype=cv2.CV_32S)
# Filter the cocos on size
min_size = 400
new_coco_nr, new_coco_labels, new_coco_stats, new_coco_centroids = \
    filterCocoMinSize(coco_nr, coco_labels, coco_stats, coco_centroids, min_size)
# Plot the filtered cocos
colors = [(0,0,0)] + [(random(),random(),random()) for i in range(coco_nr)]
new_map = cl.LinearSegmentedColormap.from_list('new_map', colors, N=coco_nr+1)
plt.figure(figsize=(20,15))
```

```
plt.subplot(2,1,1)
plt.imshow(img_gray,cmap='gray')
plt.subplot(2,1,2)
plt.imshow(new_coco_labels, cmap=new_map, interpolation='None')
plt.show()
print('Number of connected components (not background): ', new_coco_nr-1)
# Function to fill the wholes in each coco
def fillHolesCocos(coco_labels, coco_stats):
    # Find all connected components and retrieve the external contour of each coco.
    for i, coco_stat in enumerate (coco_stats):
        if(i==0):
            continue
        x = coco_stat[cv2.CC_STAT_LEFT]
        y = coco_stat[cv2.CC_STAT_TOP]
        w = coco_stat[cv2.CC_STAT_WIDTH]
        h = coco_stat[cv2.CC_STAT_HEIGHT]
        coco_mask = (coco_labels[y:y+h, x:x+w]==i).astype('uint8')*255
        contours, hierarchy = cv2.findContours(coco_mask, cv2.RETR_EXTERNAL ,cv2.CHAIN_APPROX_NONE)
        cv2.drawContours(coco_labels, [contours[0]+[x,y]], -1, (i), -1)
        coco_stats[i][cv2.CC_STAT_AREA] = np.count_nonzero(coco_mask)
# Call the function to fill the holes
fillHolesCocos(new_coco_labels, new_coco_stats)
# Plot the filled cocos
colors = [(0,0,0)] + [(random(),random(),random()) for i in range(coco_nr)]
new_map = cl.LinearSegmentedColormap.from_list('new_map', colors, N=coco_nr+1)
plt.figure(figsize=(20,15))
plt.subplot(2,1,1)
plt.imshow(img_gray,cmap='gray')
plt.subplot(2,1,2)
plt.imshow(new_coco_labels, cmap=new_map, interpolation='None')
plt.show()
# Plot the filled cocos
colors = [(0,0,0)] + [(random(),random(),random()) for i in range(coco_nr)]
new_map = cl.LinearSegmentedColormap.from_list('new_map', colors, N=coco_nr+1)
plt.figure(figsize=(20,8))
plt.imshow(new_coco_labels, cmap=new_map, interpolation='None')
plt.axis('off')
# Add some stats
```

```
for i,new_coco_stat in enumerate(new_coco_stats):
    if (i==0): continue
    x = new_coco_stat[cv2.CC_STAT_LEFT]
    y = new_coco_stat[cv2.CC_STAT_TOP]
    w = new_coco_stat[cv2.CC_STAT_WIDTH]
    h = new_coco_stat[cv2.CC_STAT_HEIGHT]
    s = new_coco_stat[cv2.CC_STAT_AREA]
    plt.text(x, img_gray.shape[0]-30, "Height: %d" % h, color='k', backgroundcolor='w', fontsize=12)
    plt.text(x, img_gray.shape[0]-15, "Size: %d" % s, color='k', backgroundcolor='w', fontsize=12)
plt.show()
```

■ Day3_1

✓ classification.ipynb

```
import matplotlib.pyplot as plt
import matplotlib as mpl
import mpl_toolkits.mplot3d
import numpy as np
import seaborn as sns
import pandas as pd
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis, QuadraticDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score
from matplotlib.colors import ListedColormap
from sklearn.inspection import DecisionBoundaryDisplay
from scipy import linalg
def plot_ellipse(splot, mean, cov, color):
    v, w = linalg.eigh(cov)
    u = w[0] / linalg.norm(w[0])
    angle = np.arctan(u[1] / u[0])
    angle = 180 * angle / np.pi  # convert to degrees
    # filled Gaussian at 2 standard deviation
    ell = mpl.patches.Ellipse(mean, 2 * v[0] ** 0.5, 2 * v[1] ** 0.5,
                              angle = 180 + angle, facecolor=color,
```

```
                              edgecolor='black', linewidth=2)
    ell.set_clip_box(splot.bbox)
    ell.set_alpha(0.2)
    splot.add_artist(ell)
    splot.set_xticks(())
    splot.set_yticks(())
    # means
    plt.plot(mean[0], mean[1], '*', color='yellow', markersize=15, markeredgecolor='grey')
def plotClassificationOutput2D(clf, data, p1, p2):
    X = data.drop('y', axis=1)
    fig, ax = plt.subplots()
    DecisionBoundaryDisplay.from_estimator(
        clf, X, cmap='tab10', alpha=0.5, ax=ax,
        response_method="predict", plot_method="pcolormesh",
        xlabel='x1', ylabel='x2', shading="auto",
    )
    sns.scatterplot(data=data, x=p1, y=p2, hue='y', palette='tab10', alpha=1.0, edgecolor="black")
    plt.legend(title='Class', facecolor='white', frameon=True)
    return(ax)
def make_triple_moons(n_samples = 100, noise = 0.15):
    r = 1
    aa = np.linspace(0,np.pi+3*np.pi/8, n_samples)
    ab = np.linspace(-3*np.pi/8,np.pi, n_samples)
    ac = np.linspace(0,np.pi, n_samples)
    x1a = 0.7*r*np.cos(aa) - r
    x2a = 0.7*r*np.sin(aa)
    x1b = 0.7*r*np.cos(ab) + r
    x2b = 0.7*r*np.sin(ab)
    x1c = r*np.cos(ac)
    x2c = -r*np.sin(ac) + 0.2*r
    x1 = np.hstack((x1a,x1b,x1c)) + noise*np.random.randn(3*n_samples)
    x2 = np.hstack((x2a,x2b,x2c)) + noise*np.random.randn(3*n_samples)
    y = np.hstack((np.zeros(n_samples),np.ones(n_samples),2*np.ones(n_samples)))
    my_data = pd.DataFrame({'x1':x1, 'x2':x2, 'y': y})
    my_data.y = my_data.y.astype('category')
    return my_data
# import some data to play with
iris_data = datasets.load_iris(as_frame=True)
```

```
sns.pairplot(data=iris_data.frame, hue='target')
plt.show()
p1 = 'sepal width (cm)'
p2 = 'petal length (cm)'
X = iris_data.data
y = iris_data.target
data = X[['sepal width (cm)','petal length (cm)']]
data.insert(data.shape[1],"y",y)
sns.scatterplot(data, x=p1, y=p2, hue='y')
plt.show()
# Split the training in training and test data 75-25%
data_train, data_test = train_test_split(data, test_size=0.25)
X_train, y_train = data_train[[p1,p2]], data_train['y']
X_test, y_test = data_test[[p1,p2]], data_test['y']
print('Training set:')
print('  Nr samples:   ', X_train.shape[0])
print('  Nr predictors:', X_train.shape[1])
print('Test set:')
print('  Nr samples:   ', X_test.shape[0])
print('  Nr predictors:', X_test.shape[1])
# Set a LDA model
lda =  LinearDiscriminantAnalysis(store_covariance=True)
# Fit on the training data
lda.fit(X_train, y_train)
score_test = lda.score(X_test, y_test)
score_train = lda.score(X_train, y_train)
print("Test accuracy:  ",score_test)
print("Train accuracy: ",score_train)
ax = plotClassificationOutput2D(lda, data_test, p1, p2)
plot_ellipse(ax, lda.means_[0], lda.covariance_, 'blue')
plot_ellipse(ax, lda.means_[1], lda.covariance_, 'orange')
plot_ellipse(ax, lda.means_[2], lda.covariance_, 'green')
# Set a QDA model
qda =  QuadraticDiscriminantAnalysis(store_covariance=True)
# Fit on the training data
qda.fit(X_train, y_train)
score_test = qda.score(X_test, y_test)
score_train = qda.score(X_train, y_train)
```

```
print("Test accuracy:  ",score_test)
print("Train accuracy: ",score_train)
ax = plotClassificationOutput2D(qda, data_test, p1, p2)
plot_ellipse(ax, qda.means_[0], qda.covariance_[0], 'blue')
plot_ellipse(ax, qda.means_[1], qda.covariance_[1], 'orange')
plot_ellipse(ax, qda.means_[2], qda.covariance_[2], 'green')
data = make_triple_moons(300, 0.15)
p1 = 'x1'
p2 = 'x2'
# Split the training in training and test data 75-25%
data_train, data_test = train_test_split(data, test_size=0.25)
X_train, y_train = data_train[[p1,p2]], data_train['y']
X_test, y_test = data_test[[p1,p2]], data_test['y']
# Plot it
plt.figure(figsize=(5,5))
sns.scatterplot(data=data, x='x1', y='x2', hue='y')
plt.legend(loc='lower right',title='Class')
plt.xlim(-2,2)
plt.ylim(-2,2)
plt.show()
# PUT YOUR CODE HERE
# Set a LDA model
lda =  LinearDiscriminantAnalysis(store_covariance=True)
data = make_triple_moons(300, 0.15)
p1 = 'x1'
p2 = 'x2'
# Split the training in training and test data 75-25%
data_train, data_test = train_test_split(data, test_size=0.25)
X_train, y_train = data_train[[p1,p2]], data_train['y']
X_test, y_test = data_test[[p1,p2]], data_test['y']
# Fit on the training data
lda.fit(X_train, y_train)
score_test = lda.score(X_test, y_test)
score_train = lda.score(X_train, y_train)
print("Test accuracy:  ",score_test)
print("Train accuracy: ",score_train)
ax = plotClassificationOutput2D(lda, data_test, p1, p2)
plot_ellipse(ax, lda.means_[0], lda.covariance_, 'blue')
```

```
plot_ellipse(ax, lda.means_[1], lda.covariance_, 'orange')
plot_ellipse(ax, lda.means_[2], lda.covariance_, 'green')
# PUT YOUR CODE HERE
# Set a QDA model
qda =  QuadraticDiscriminantAnalysis(store_covariance=True)
data = make_triple_moons(300, 0.15)
p1 = 'x1'
p2 = 'x2'
# Split the training in training and test data 75-25%
data_train, data_test = train_test_split(data, test_size=0.25)
X_train, y_train = data_train[[p1,p2]], data_train['y']
X_test, y_test = data_test[[p1,p2]], data_test['y']
# Fit on the training data
qda.fit(X_train, y_train)
score_test = qda.score(X_test, y_test)
score_train = qda.score(X_train, y_train)
print("Test accuracy:  ",score_test)
print("Train accuracy: ",score_train)
ax = plotClassificationOutput2D(qda, data_test, p1, p2)
plot_ellipse(ax, qda.means_[0], qda.covariance_[0], 'blue')
plot_ellipse(ax, qda.means_[1], qda.covariance_[1], 'orange')
plot_ellipse(ax, qda.means_[2], qda.covariance_[2], 'green')
# Set a kNN model
knn =  KNeighborsClassifier(n_neighbors=5)
# Fit on the training data
knn.fit(X_train, y_train)
score_test = knn.score(X_test, y_test)
score_train = knn.score(X_train, y_train)
print("Test accuracy:  ",score_test)
print("Train accuracy: ",score_train)
ax = plotClassificationOutput2D(knn, data_test, p1, p2)
# Split the training in training and validation set 75-25%
data_train, data_val = train_test_split(data_train, test_size=0.25)
X_train, y_train = data_train[[p1,p2]], data_train['y']
X_val, y_val = data_val[[p1,p2]], data_val['y']
K = np.array([1,2,5,10,20,50,100])
Val_acc = []
for k in K:
```

```
    # Set a kNN model
    knn =  KNeighborsClassifier(n_neighbors=k)
    # Fit on the training data
    knn.fit(X_train, y_train)
    # Calculate the accuracy on the validation set
    score_val = knn.score(X_val, y_val)
    Val_acc.append(score_val)
plt.plot(K, Val_acc, 'r')
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.title('kNN: k vs accuracy on the validation set')
plt.grid()
plt.show()
# Add your code here:
K = 20
Test_acc = []
# Split the training and test data 75-25%
data_train, data_test = train_test_split(data, test_size=0.25)
X_train, y_train = data_train[[p1,p2]], data_train['y']
X_test, y_test = data_test[[p1,p2]], data_test['y']



# Set a kNN model
knn =  KNeighborsClassifier(n_neighbors=k)
# Fit on the test data
knn.fit(X_train, y_train)


score_test = knn.score(X_test, y_test)
score_train = knn.score(X_train, y_train)
print("Test accuracy:  ",score_test)
print("Train accuracy: ",score_train)
```

- ✓ mlp_classification.ipynb

```
!git clone https://git.wur.nl/koots006/sensing-and-perception-course.git data
import sklearn                # Scikit-learn library which contains a lot of Machine Learning Models
ready to use
import pandas as pd           # Library for data analysis. We will use it to load datasets
```

```
import numpy as np              # Library for highly efficient computations
import matplotlib.pyplot as plt # To create plots
from sklearn.datasets import fetch_openml            # A function to festch some standard datasets
from sklearn.model_selection import train_test_split  # Split X and y in training and test sets
from sklearn import metrics                          # Some function to calculate performance metrics
# Other imports and settings for better visualization of graphs
from IPython import display
# Import everything from utils.py, a self-made Python file with utility functions
from data.tutorial_dl_mlp.mlp_utils import *
# Download the dataset
X, y = fetch_openml('mnist_784', version=1, return_X_y=True, as_frame=False)
# We scale the data to the range of 0-1 in order to speed up the training process
X = X / 255
print("Number of images in the data set:", X.shape[0])
print("Number of features/variables per image:", X.shape[1])
# Get image nr 10 from the training set
data_id = 10
sample_image = X[data_id].reshape((28, 28)) # Reshape the image from (784) to (28x28)
plt.imshow(sample_image, cmap='gray')
plt.show()
## ADD YOUR CODE HERE
print(...)
import random
nr_image_to_show = 16
plt.figure(figsize=(12,4))
for i in range(nr_image_to_show):
  # Get a random image from the training set
  data_id = random.randint(0, len(X))
  sample_image = X[data_id].reshape((28, 28)) # Reshape the image from (784) to (28x28)
  plt.subplot(2,8,i+1)
  plt.imshow(sample_image, cmap='gray')
  plt.title("Label: %s" % y[data_id])
plt.tight_layout()
plt.show()
# Split in train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
# Provide some info on the training and test set
size_train = len(X_train)
```

```
size_test = len(X_test)
print("Number of images in the training set:", size_train)
print("Number of images in the test set:", size_test)
# Create the network
mlp = MLPClassifier(hidden_layer_sizes=(16), max_iter=100, solver='sgd', learning_rate_init=.1,
verbose=True, random_state=1)
# Train the network on the training data
mlp.fit(X_train, y_train)
# Plot the loss:
plt.plot(mlp.loss_curve_)
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.grid()
plt.show()
test_ids = [0,1,2]
# Use the neural network to predict the class for the test images
pred = mlp.predict(X_test[test_ids])
# Show the images
for i, test_id in enumerate(test_ids):
  plt.subplot(1,len(test_ids),i+1)
  plt.imshow(X_test[test_id].reshape((28,28)), cmap='gray')
  plt.title('Test id: %d' % test_id)
plt.show()
# Print the predictions and the ground-truth labels
for i, test_id in enumerate(test_ids):
  print('Test id: %d \t predicted: %s, true label: %s' % (test_id, pred[i], y_test[test_id]))
# Use the .predict() method to get the prediction
# WRITE YOUR CODE HERE (1 line)
preds = mlp.predict(X_test)
# Calculate the accuracy = percentage of correct predictions
accuracy = metrics.accuracy_score(y_test, preds)
print("Classification accuracy: ", accuracy)
# visualize images + predictions
fig = plt.figure(figsize=(20, 20))
for i in range(10):
  test_id = random.randint(0,size_test)
  img = X_test[test_id].reshape((28, 28))
  subplot = fig.add_subplot(1,10, i+1)
```

```
  title = "Pred: " + preds[test_id] + ", truth: " + y_test[test_id]
  subplot.title.set_text(title)
  plt.imshow(img)
plt.show()
# Find the false predictions
correct_predictions = y_test != preds
error_ids = list(np.where(correct_predictions)[0])
# Show 16 randomly selected error cases
show_nr_cases = 16
plt.figure(figsize=(16,4))
for i,case_id in enumerate(random.sample(error_ids, show_nr_cases)):
  img = X_test[case_id].reshape((28, 28))
  plt.subplot(2,8, i+1)
  plt.imshow(img)
  plt.axis('off')
  plt.title("Pred:" + preds[case_id] + ", Truth: " + y_test[case_id])
plt.show()
all_accuracy_scores = []
nr_of_hidden_layers = [0,1,2,4]
for n in nr_of_hidden_layers:
  print("=== Training with %d hidden layers ==="%n)
  # Step 1: Create the model with i number of hidden layers
  # WRITE YOUR CODE HERE
  mlp = MLPClassifier(hidden_layer_sizes=(16), max_iter=50, solver='sgd', learning_rate_init=.1,
verbose=True, random_state=1)
  # Step 2: Train the model
  # WRITE YOUR CODE HERE (1 lines)
  mlp.fit(X_train, y_train)
  # Step 3: use the network to predict on the test set
  pred = mlp.predict(X_test[test_ids])
  # Step 4: Calculate the accuracy score and append to all_accuracy_scores
  # WRITE YOUR CODE HERE
  accuracy = ...
  all_accuracy_scores.append(accuracy)
# Step 5: Show the plot
plt.plot(nr_of_hidden_layers, all_accuracy_scores)
plt.xlabel('Nr of hidden layers')
plt.ylabel('Accuracy')
```

```
plt.show()
```

■ Day3_2

✓ tomato_sweetness.ipynb

```
!git clone https://git.wur.nl/koots006/summerschool-image-processing-for-plant-phenotyping.git data
import sklearn # Scikit-learn library which contains a lot of Machine Learning Models ready to use
import pandas as pd # Library for data analysis. We will use it to load datasets
import numpy as np # Library for highly efficient computations
import matplotlib.pyplot as plt  # To create plots
import seaborn as sns
from sklearn.metrics import mean_squared_error, r2_score # Import some performance metrics
from sklearn.model_selection import train_test_split # Split X and y in train and test sets
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression # Import Linear Regression model from Scikit-Learn
from sklearn.ensemble import RandomForestRegressor # Import Random Forest Regression
# Other imports and settings for better visualization of graphs
from IPython import display
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
plt.rcParams['figure.figsize'] = [10,8]
plt.rcParams.update({'font.size': 10})
# Import everything from utils.py
from data.machine_learning.utils import *
# Load dataset
dataset = pd.read_csv('data/machine_learning/datasets/spectra_lab_ABC.csv')
dataset.describe(include='all')
# Create sub-datasets per variety
data_A = dataset[(dataset.variety=='A')].drop('variety', axis=1)
data_B = dataset[(dataset.variety=='B')].drop('variety', axis=1)
data_C = dataset[(dataset.variety=='C')].drop('variety', axis=1)
data_all = dataset.drop('variety', axis=1)
print('Nr tomatoes of variety A: ', data_A.shape[0])
print('Nr tomatoes of variety B: ', data_B.shape[0])
print('Nr tomatoes of variety C: ', data_C.shape[0])
```

```
# Get X and y from dataset
X_all, y_all = get_X_y_data(data_all, y_name='brix')
X_A, y_A = get_X_y_data(data_A, y_name='brix')
X_B, y_B = get_X_y_data(data_B, y_name='brix')
X_C, y_C = get_X_y_data(data_C, y_name='brix')
# Calculate the associated wave lengths
wave_lengths = np.linspace(470,900,X_all.shape[1])
# Plot the spectra
plt.subplot(2,2,1)
plt.plot(wave_lengths,X_all.T)
plt.title('All varieties')
plt.xlabel('Wavelength [-]')
plt.ylabel('Reflectance [-]')
plt.subplot(2,2,2)
plt.plot(wave_lengths,X_A.T)
plt.title('Variety A')
plt.xlabel('Wavelength [-]')
plt.ylabel('Reflectance [-]')
plt.subplot(2,2,3)
plt.plot(wave_lengths,X_B.T)
plt.title('Variety B')
plt.xlabel('Wavelength [-]')
plt.ylabel('Reflectance [-]')
plt.subplot(2,2,4)
plt.plot(wave_lengths,X_C.T)
plt.title('Variety C')
plt.xlabel('Wavelength [-]')
plt.ylabel('Reflectance [-]')
plt.tight_layout()
plt.show()
fig,ax = plt.subplots(1,2,figsize=(10,3))
sns.boxplot(data=dataset, x="brix", y="variety", ax=ax[0])
sns.kdeplot(data=dataset, x="brix", hue="variety", fill=True, ax=ax[1])
plt.show()
# Step 1: Get the predictors (X, the spectrum) and the response (y, brix)
X, y = get_X_y_data(data_all, y_name='brix')
# Step 2: Split X and y in train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
print("Training set size:", X_train.shape[0])
print("Test set size:", X_test.shape[0])
# Step 3: Train a multi-variate linear regression model on the training data
regressor = LinearRegression()  # Create linear regression object
regressor.fit(X_train, y_train) # Training the linear regression
# Step 4: Evaluate the LM model on the training data
y_pred_train = regressor.predict(X_train)
mse_train = mean_squared_error(y_train, y_pred_train)
r2_train = r2_score(y_train, y_pred_train)
# Step 5: Evaluate the LM model on the test data
y_pred_test = regressor.predict(X_test)
mse_test = mean_squared_error(y_test, y_pred_test)
r2_test = r2_score(y_test, y_pred_test)
# Step 6: Print the training and test accuracies
print('\nEVALUATION:')
print('''\t\t Mean Squared Error \t R^2
Training set \t\t %.2f \t\t %.2f
Test set     \t\t %.2f \t\t %.2f'''%(mse_train, r2_train, mse_test, r2_test))
plt.figure(figsize=(6,6))
plt.imshow(X_all.corr(),vmin=-1, vmax=1, cmap='jet')
plt.colorbar()
plt.show()
reduce_wavelengths = 2 # TO-DO: change (>1) value and see what happens
# Get X and y from dataset using the given function
X_all, y_all = get_X_y_data(data_all, y_name='brix', reduce_wavelengths=reduce_wavelengths)
# Check that the size of the arrays are correct
print('X shape', X_all.shape) # X size should be (number of entries,
                             # (number of dataset variables-2)/reduce_wavelengths)
print('y shape', y_all.shape) # y size should be (number of entries,)
reduce_wavelengths=2 # TO-DO: change (>1) value and see what happens
Data = [data_all, data_A, data_B, data_C]
Data_names = ["All", "A", "B", "C"]
print('With a reduction of', reduce_wavelengths)
for data, d_name in zip(Data, Data_names):
    # Step 1: Get the data and reduce the number of wavelengths
    X, y = get_X_y_data(data, y_name='brix', reduce_wavelengths=reduce_wavelengths)
    # Step 2: Split X and y in train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
    # Step 3: Train a linear regression model
    # PUT YOUR CODE HERE (2 lines)
    # Step 3: Train a multi-variate linear regression model on the training data
    regressor = LinearRegression()  # Create linear regression object
    regressor.fit(X_train, y_train) # Training the linear regression
    # Step 4: Evaluate the model
    print(20*"=", "p:",X_train.shape[1], "n:", X_train.shape[0], 20*"=")
    evaluate_performace_regression(regressor, X_train, X_test, y_train, y_test, to_print=True)
# Initialize lists to store results
train_mse_list = []
test_mse_list = []
train_r2_list = []
test_r2_list = []
reductions = np.arange(10)[::-1]
sizes = []
for r in reductions:
    X_all, y_all = get_X_y_data(data_all, y_name='brix', reduce_wavelengths=r)
    sizes.append(X_all.shape[1])
    X_train, X_test, y_train, y_test = train_test_split(X_all, y_all, test_size=0.2, random_state=0)
    regressor = LinearRegression()  # Create linear regression object
    regressor.fit(X_train, y_train) # Training the linear regression
    # Calculate performance metric with given function
    mse_train, mse_test, r2_train, r2_test = evaluate_performace_regression(regressor, X_train, X_test,
y_train,
                                                                            y_test, to_print=False)
    # Store results in lists
    train_mse_list.append(mse_train)
    test_mse_list.append(mse_test)
    train_r2_list.append(r2_train)
    test_r2_list.append(r2_test)
    # Plot results
    plt.subplot(1,2,1) # Subplots matrix: 1 row, 2 columns. This is subplot 1
    plt.plot(sizes, train_mse_list, label='train')
    plt.plot(sizes, test_mse_list, label='test')
    plt.legend()
    plt.xlabel('Number of wavelengths')
    plt.ylabel('MSE [-]')
    plt.title('MSE')
```

```
    plt.subplot(1,2,2) # Subplots matrix: 1 row, 2 columns. This is subplot 2
    plt.plot(sizes, train_r2_list, label='train')
    plt.plot(sizes, test_r2_list, label='test')
    plt.legend()
    plt.xlabel('Number of wavelengths')
    plt.ylabel('R2 [-]')
    plt.title('R2')
    display.clear_output(wait=True) # update plots
    plt.show()
Regressors = [
    LinearRegression(),
    RandomForestRegressor(n_estimators = 10)
]
for regressor in Regressors:
    Data = [data_all, data_A, data_B, data_C]
    Data_names = ["All", "A", "B", "C"]
    print('\nUsing regressor:', regressor.__class__.__name__)
    for data, d_name in zip(Data, Data_names):
        # Step 1: Get the data and reduce the number of wavelengths
        X, y = get_X_y_data(data, y_name='brix', reduce_wavelengths=1)
        # Step 2: Split X and y in train and test sets
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
        # Step 3: Train a linear regression model
        # PUT YOUR CODE HERE (2 lines)
        # Step 3: Train a multi-variate linear regression model on the training data
        regressor.fit(X_train, y_train) # Training the linear regression
        # Step 4: Evaluate the model
        print(20*"=", "p:",X_train.shape[1], "n:", X_train.shape[0], 20*"=")
        evaluate_performace_regression(regressor, X_train, X_test, y_train, y_test, to_print=True)
Data = [data_all, data_A, data_B, data_C]
Data_names = ["All", "A", "B", "C"]
plt.figure(figsize=(8,8))
for i, (data, d_name) in enumerate(zip(Data, Data_names)):
    # Step 1: Get the data and reduce the number of wavelengths
    X, y = get_X_y_data(data, y_name='brix', reduce_wavelengths=1)
    # Step 2: Split X and y in train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
    # Step 3: Train a multi-variate linear regression model on the training data
```

```
    regressor = RandomForestRegressor(n_estimators = 50)
    regressor.fit(X_train, y_train) # Training the linear regression
    # Step 4: Show the predicted values vs ground-truth measurements
    y_pred_test = regressor.predict(X_test)
    plt.subplot(2,2,i+1)
    plt.plot(y_test, y_pred_test, '.')
    plt.plot([0,12],[0,12],color='orange')
    plt.xlabel('Ground-truth Brix')
    plt.ylabel('Predicted Brix')
    plt.title(d_name)
plt.tight_layout()
plt.show()
```

- ✓ tomato_sweetness_and_mlp.ipynb

```
!git clone https://git.wur.nl/koots006/summerschool-image-processing-for-plant-phenotyping.git data
import sklearn # Scikit-learn library which contains a lot of Machine Learning Models ready to use
import pandas as pd # Library for data analysis. We will use it to load datasets
import numpy as np # Library for highly efficient computations
import matplotlib.pyplot as plt  # To create plots
import seaborn as sns
from sklearn.metrics import mean_squared_error, r2_score # Import some performance metrics
from sklearn.model_selection import train_test_split # Split X and y in train and test sets
from sklearn.neural_network import MLPRegressor
# Other imports and settings for better visualization of graphs
from IPython import display
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
plt.rcParams['figure.figsize'] = [10,8]
plt.rcParams.update({'font.size': 10})
# Import everything from utils.py
from data.machine_learning.utils import *
# Load dataset
dataset = pd.read_csv('data/machine_learning/datasets/spectra_lab_ABC.csv')
dataset.describe(include='all')
# Create sub-datasets per variety
```

```
data_A = dataset[(dataset.variety=='A')].drop('variety', axis=1)
data_B = dataset[(dataset.variety=='B')].drop('variety', axis=1)
data_C = dataset[(dataset.variety=='C')].drop('variety', axis=1)
data_all = dataset.drop('variety', axis=1)
X_all, y_all = get_X_y_data(data_all, y_name='brix', reduce_wavelengths=2)
X_train, X_test, y_train, y_test = train_test_split(X_all, y_all, test_size=0.2, random_state=0)
regressor = MLPRegressor(solver='lbfgs', max_iter=2000, hidden_layer_sizes=[])
regressor.fit(X_train, y_train) # Training the perceptron
evaluate_performace_regression(regressor, X_train, X_test, y_train, y_test, to_print=True)
# Initialize lists to store results
train_mse_list = []
test_mse_list = []
train_r2_list = []
test_r2_list = []
n_layers = [0, 1, 2, 4, 6, 8] # Create list containing some possible number of hidden layers to test
the model
fig = plt.figure(1, figsize=(8,4))
for i in n_layers: # Loop over elements in list
    X_all, y_all = get_X_y_data(data_all, y_name='brix', reduce_wavelengths=2)
    X_train, X_test, y_train, y_test = train_test_split(X_all, y_all, test_size=0.2, random_state=0)

    MLP = neural_network(i,64) # Create Neural Network
    MLP.fit(X_train, y_train) # Train the algorithm

    # Calculate performance metric with given function
    mse_train, mse_test, r2_train, r2_test = evaluate_performace_regression(MLP, X_train, X_test,
y_train, y_test, to_print=False)

    # Store results in lists
    train_mse_list.append(mse_train)
    test_mse_list.append(mse_test)
    train_r2_list.append(r2_train)
    test_r2_list.append(r2_test)
    # Plot results
    fig = plt.figure(1, figsize=(8,4))
    plt.subplot(1,2,1) # Subplots matrix: 1 row, 2 columns. This is subplot 1
    plt.plot(n_layers[:len(train_mse_list)], train_mse_list, label='train')
    plt.plot(n_layers[:len(test_mse_list)], test_mse_list, label='test')
```

```
    plt.legend()
    plt.xlabel('Number of hidden layers')
    plt.ylabel('MSE [-]')
    plt.title('MSE')
    plt.subplot(1,2,2) # Subplots matrix: 1 row, 2 columns. This is subplot 2
    plt.plot(n_layers[:len(train_mse_list)], train_r2_list, label='train')
    plt.plot(n_layers[:len(test_mse_list)], test_r2_list, label='test')
    plt.legend()
    plt.xlabel('Number of hidden layers')
    plt.ylabel('R2 [-]')
    plt.title('R2')
    display.clear_output(wait=True) # update plots
    plt.tight_layout()
    plt.show()
```

■ Day3_3

✓ WUR_SummerSchool2023_CNN_Session.ipynb

```
# Click here, and the run using the play key, or CONTROL+ENTER (COMMAND+ENTER on Mac)
5 + 6
import torch
import torchvision
torch.cuda.get_device_name(0)
import numpy as np #You can import a library using **import** keyword; "as" keyword allows you to alias
the package name into something smaller
#Some very simple array creation and some operations on top of those array
arr0 = np.array( [[3,2], [1,3]] )
print("arr0: \n", arr0)
#indexing
print("arr0 Row0: {} \narr0 Col1{} \narr0 Element(1,1){}".format(arr0[0,:], arr0[:,1], arr0[1,1] ))
arr1 = np.arange(9).reshape(3, 3)
print("\narr1: \n{}".format(arr1) )
#Multiplication
arr2 = np.ones(9).reshape(3, 3) * np.random.uniform(0.7, 0.1)
print("\narr2: \n", arr2)
#Division
```

```
arr3 = arr1/arr2

print("\narr3: \n", arr3)

print("\narr3 shape: \n", arr3.shape)

#Some simple operations on tensors

x = torch.rand(3, 3)

print(x)

#Addition

y = torch.rand(3, 3)

print("Tensor addition example:\n", x + y)

print("\nTensor muliplication example:\n", x * y)

print("\nTensor division example:\n", x / y)

print("x col1: {}\ny col0{}".format(x[:, 1], y[0, :]))

#Is CUDA available?

print(torch.cuda.is_available())

x_c = x.cuda()

y_c = y.cuda()

print("x moved to GPU:\n", x_c)

z = x_c*y_c

print("Result of operations remains on GPU, if the variables are on GPU\n z=", z) #Result of operations

remains on GPU

z_cpu = z.cpu()

print("Data brought back to cpu:\n", z_cpu) #Bring data to CPU

z_np = z_cpu.numpy()

print("Numpy bridge:\n", z_np)

from __future__ import print_function, division

import os

from IPython.display import Image

from IPython.core.display import HTML

import torch

import torchvision

import pandas as pd

from skimage import io, transform

import numpy as np

import matplotlib.pyplot as plt

from torch import nn

from torch.autograd import Variable

import torch.nn.functional as F

import torch.optim as optim
```

```
from torch.utils.data import Dataset, DataLoader
from torchvision import datasets, transforms, utils
torch.cuda.device_count()
#download MNIST train and test datasets,
#transform them to tensors and normalize them
download_to = './data/MNIST'
transform = transforms.Compose([
                      transforms.ToTensor(),
                      transforms.Normalize((0.1307,), (0.3081,))
                  ])
trainset = datasets.MNIST(download_to, train=True,
                          download=True, transform=transform)
testset = datasets.MNIST(download_to, train=False,
                         download=True, transform=transform)
#Create training and test "loaders"
#DataLoader class combines a dataset and a sampler,
#and provides single- or multi-process iterators over the dataset.
trainloader = torch.utils.data.DataLoader(trainset, batch_size=400,
                                          shuffle=True, num_workers=4)
testloader = torch.utils.data.DataLoader(testset, batch_size=1000,
                                          shuffle=True, num_workers=4)
classes = ('0', '1', '2', '3', '4', '5', '6', '7', '8', '9')
#if args.cuda:
#    torch.cuda.manual_seed(args.seed)
def imshow(img):
    img=img*0.1307 + 0.3081 # un-normalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
# get some random training images
dataiter = iter(trainloader)
images, labels = next(dataiter)
print(images.shape)
# show images
plt.figure(figsize=(30,30))
print ('Labels: ')
print(' '.join('%1s' % classes[labels[j]] for j in range(len(labels) )))
imshow(torchvision.utils.make_grid(images))
#A simple convenet for MNIST dataset
```

```
# Number of filters in various layers
#f1 = 1
#f2 = 1
f1 = 3
f2 = 6
#f1 = 7
#f2 = 14
#f1 = 10
#f2 = 20
#f1 = 20
#f2 = 40
# Filter size s (sxs pixels)
#s = 1
s = 3
#s = 5
#s = 7
# Non-linearity to use between layers (except after output)
nonlin = nn.Sigmoid
#nonlin = nn.ReLU
# Number of hidden units in classifier middle layer
units = 50
class Net(nn.Module):
    def __init__(self, input_size=(1,28,28)):
        super(Net, self).__init__()
        self.features = nn.Sequential(
            #Convolve, Maxpool, Non-linearity
            nn.Conv2d(input_size[0], f1, s), nn.MaxPool2d(2), nonlin(),
            #Convolve, Maxpool, Non-linearity
            nn.Conv2d(f1,f2,s), nn.MaxPool2d(2), nonlin()
        )
        self.features_size = self._get_collection_output_size(input_size, self.features)
        self.classifier = nn.Sequential(
            #Apply linear activation, Non-linearity
            nn.Linear(self.features_size, units), nonlin(),
            #Apply linear activation
            nn.Linear(units, 10), #nonlin(),
            #Apply sigmoid in the final layer
#             nn.Sigmoid()  # NOT GOOD FOR Multiclass classification
```

```
            nn.LogSoftmax(dim=1) # This is a special non-linearity that forces the output to be a
probability distribution (in our case, 10 values between 0 and 1
                                 # that sum to 1). However, it outputs the Log of these values. This is
a trick that improves the numerical stability of the code
                                 # and is used in combination with nn.NLLLoss() for the loss (you'll
see this a bit later)
        )
    def _get_collection_output_size(self, input_size, collection):
        c = collection(Variable(torch.ones(1,*input_size)))
        #print("Size: ", c.size())
        return int(np.prod(c.size()[1:]))
    def forward(self, x):
        #Forward prop invokes the feature extraction
        x = self.features(x)
        #After feature extraction, the features are reshaped
        x = x.view(-1, self.features_size)
        #Reshaped features are passed on to the classification stage
        x = self.classifier(x)
        return x
img_size = images[0].shape
model = Net(img_size)
print(model)
params = list(model.parameters())
print(len(params))
print(params[-1].shape)  # Shape of the Final layer (should be 10)
print(params[-1])
#Put the model on GPU
enable_cuda = False
if torch.cuda.is_available():
    model.cuda()
    enable_cuda = True
#optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.5)
#optimizer = optim.RMSprop(params, lr=0.001)
optimizer = optim.Adadelta(params, lr=0.001)
#optimizer = optim.Adam(params, lr=0.001, betas=(0.9, 0.999))
loss_fn = nn.NLLLoss() # This loss matches the output nn.LogSoftmax we used earlier
def train(epoch, loader=trainloader):
    model.train() #Sets the module in training mode.
```

```
    for batch_idx, (data, target) in enumerate(loader):
        #Upload data to GPU
        if enable_cuda:
            data, target = data.cuda(), target.cuda()
        data, target = Variable(data), Variable(target)
        optimizer.zero_grad() #Don't forget to clean the old gratient values
        #Feed data to the model
        output = model(data)
        #Compute loss and backpropagate
        loss = loss_fn(output, target)
        loss.backward()
        #Update the model parameters
        optimizer.step()
        #Print information evey 500
        if batch_idx % 500 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(loader.dataset),
                100. * batch_idx / len(loader), loss.data.item()))#loss.data[0]))
test_loss_list = []
def test():
  model.eval() #Always use this during training
  test_loss = 0
  correct = 0
  for data, target in testloader:
      if enable_cuda:
          data, target = data.cuda(), target.cuda() # Send the test minibatch to the GPU
      data, target = Variable(data, volatile=True), Variable(target)
      output = model(data) # Pass the data through the model
      #print(output.shape)
      #print(target.shape)
      test_loss += F.nll_loss(output, target, size_average=False).data.item() # sum up batch loss
      pred = output.data.max(1, keepdim=True)[1] # get the index of the max log-probability, which is
the class the model thinks the input is.
      correct += pred.eq(target.data.view_as(pred)).long().cpu().sum() # Test how many predicted
classes match the ground truth known classes.
  test_loss /= len(testloader.dataset)
  print("test_loss: ",test_loss)
  test_loss_list.append([test_loss])
```

```
  print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
      test_loss, correct, len(testloader.dataset),
      100. * correct / len(testloader.dataset)))
  return test_loss, correct, (correct / len(testloader.dataset))
max_epochs = 10
for epoch in range(1, max_epochs + 1):
    train(epoch)
    test()
# Let's visualize some random test images
dataiter = iter(testloader)
test_images, labels = next(dataiter)
print(test_images.shape)
# show images
plt.figure(figsize=(10,10))
test_image_set = test_images[0:5]
imshow(torchvision.utils.make_grid(test_image_set))
model.eval()
outputs = model(Variable(test_image_set.cuda()))
_, predicted = torch.max(outputs.data, 1)
print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]
                              for j in range(5)))
# Number of filters in various layers (you do not need to change these)
df1 = 32
df2 = 64
# Filter size s (sxs pixels) (you do not need to change these)
ds = 3
# Non-linearity to use between layers (except after output) (you do not need to change these)
dnonlin = nn.ReLU
# Number of hidden units in the classifier stage (you do not need to change these)
dunits = 100
# How many repeating stages should we add? (each stage will contain two convolutions)
n_repeats = 10
# NOTE! Set type of repeat lower down in the code
# The following function implements a "normal" section with just two convolutions
# and their non-linearities
def normal_section():
    return nn.Sequential(
            nn.Conv2d(df1, df1, ds, padding="same"), dnonlin(),
```

```
            nn.Conv2d(df1, df1, ds, padding="same"), dnonlin()
    )
# The following function is similiar to above, except adding batch normalization
def batch_norm_section():
    return nn.Sequential(
            nn.Conv2d(df1, df1, ds, padding="same"), dnonlin(), nn.BatchNorm2d(df1),
            nn.Conv2d(df1, df1, ds, padding="same"), dnonlin(), nn.BatchNorm2d(df1)
    )
# The following class creates two convolutional stages and residual connection
# around them.
class residual_section(nn.Module):
    def __init__(self):
        super(residual_section, self).__init__()
        self.conv1 = nn.Conv2d(df1, df1, ds, padding="same")
        self.conv2 = nn.Conv2d(df1, df1, ds, padding="same")
        self.nonlin1 = dnonlin()
        self.nonlin2 = dnonlin()
    def forward(self,x):
        residual = x
        out = self.nonlin1(self.conv1(x))
        out = self.nonlin2(self.conv2(out))
        return out + residual
# The following function makes n_repeats sections of the same type.
def repeat_section(section_type, n_repeats):
    return nn.Sequential(*[section_type() for x in range(n_repeats)])
# SET THE SECTION TYPE HERE
section_type = normal_section
#section_type = batch_norm_section
#section_type = residual_section
# This class is very similar to the earlier covnet, but notice the added
# "repeat_section" part in the features section.
class Net(nn.Module):
    def __init__(self, input_size=(1,28,28)):
        super(Net, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(input_size[0], df1, ds, padding="same"), dnonlin(),
            repeat_section(section_type, n_repeats),
            nn.Conv2d(df1, df2, ds, padding="same"), nn.MaxPool2d(2), dnonlin(),
```

```
            nn.Conv2d(df2, df2, ds, padding="same"),  nn.MaxPool2d(2), dnonlin(),
        )
        self.features_size = self._get_collection_output_size(input_size, self.features)
        self.classifier = nn.Sequential(
            nn.Linear(self.features_size, dunits), dnonlin(),
            nn.Linear(dunits, 10), #dnonlin(),
            nn.LogSoftmax(dim=1)
        )
    def _get_collection_output_size(self, input_size, collection):
        c = collection(Variable(torch.ones(1,*input_size)))
        #print("Size: ", c.size())
        return int(np.prod(c.size()[1:]))
    def forward(self, x):
        x = self.features(x)
        x = x.view(-1, self.features_size)
        x = self.classifier(x)
        return x
model = Net(img_size).cuda()
model.train()
print(model)
optimizer = optim.Adam(model.parameters(), lr=0.001)
loss_fn = nn.NLLLoss()
trainloader = torch.utils.data.DataLoader(trainset, batch_size=400,
                                          shuffle=True, num_workers=4)
enable_cuda = False
if torch.cuda.is_available():
    model.cuda()
    enable_cuda = True
max_epochs = 10
scenarios = {} # Reset all the stats gathered from your runs in the following code.
model.train()
accuracies = []
epochs = []
for epoch in range(1, max_epochs + 1):
    train(epoch, loader=trainloader)
    loss, correct, accuracy = test()
    epochs.append(epoch)
    accuracies.append(accuracy)
```

```
plt.plot(epochs, accuracies)
```

■ Day4_1

✓ WUR_SummerSchool2023_Semantic_Segmentation.ipynb

```
#Download dataset from my googledriv
#Source: https://github.com/ndrplz/google-drive-downloader
import requests
def download_file_from_google_drive(id, destination):
    def get_confirm_token(response):
        for key, value in response.cookies.items():
            if key.startswith('download_warning'):
                return value
        return None
    def save_response_content(response, destination):
        CHUNK_SIZE = 32768
        with open(destination, "wb") as f:
            for chunk in response.iter_content(CHUNK_SIZE):
                if chunk: # filter out keep-alive new chunks
                    f.write(chunk)
    URL = "https://docs.google.com/uc?export=download"
    session = requests.Session()
    response = session.get(URL, params = { 'id' : id }, stream = True)
    token = get_confirm_token(response)
    if token:
        params = { 'id' : id, 'confirm' : token }
        response = session.get(URL, params = params, stream = True)
    save_response_content(response, destination)
!mkdir data
download_file_from_google_drive('10QTdIA061OnkoZ8ZnSL8hkx8CE0w5XO_', 'data/Ara2013-Canon-Dataset.zip')
#Check if the data is present
!ls -al data/
#Unzip the files
!unzip -q -o data/Ara2013-Canon-Dataset.zip -d data/
import torch
import torchvision
```

```
from torch import Tensor
import torch.autograd
from torch.autograd import Variable
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
from PIL import Image
import numpy as np
import pandas as pd
import random
import skimage
from skimage import transform, morphology
from skimage.morphology import erosion
from skimage.morphology import disk
from skimage import io as skio
from matplotlib import pyplot as plt
from PIL import Image
# from ipywidgets import interact, interactive, fixed, interact_manual
# import ipywidgets as widgets
from IPython.display import clear_output
#Data enrichment
def resize(image, size):
    return transform.resize(image, size)
def adjust_brightness(image, scale_factor):
    return image * scale_factor
def flip_image(image, flip_row=False, flip_col=False):
    if flip_row:
        image = np.flip(image, 0)
    if flip_row:
        image = np.flip(image, 1)
    return image
def random_flips(images):
    flip_row = (np.random.randint(2) == 1)
    flip_col = (np.random.randint(2) == 1)
    results = []
    for image in images:
        result = flip_image(image, flip_row, flip_col)
```

```
        results.append(result)
    return results
def create_mask(labels, do_erosion=False):
    r_1 = labels[:,:,0]
    g_1 = labels[:,:,1]
    b_1 = labels[:,:,2]
    l = r_1 + g_1 + b_1 / 3
    mask_1 = (l>0).astype(np.uint8)
    mask_1 = mask_1[:,:] / np.max(mask_1)
    mask_1 = mask_1.astype(np.uint8)
    if do_erosion:
        selem = disk(8)
        mask_1 = erosion(mask_1[:,:], selem)
    mask_1 = mask_1[:,:,None]
    return mask_1.astype(np.bool)
class Ara2013Dataset(Dataset):
    def __init__(self, dataset_dir, csv_filename, data_enrichment=False, transform=lambda x:x,
height=128, width=128, categorical=False):
        self.dataset_dir = dataset_dir
        self.height = height
        self.width = width
        self.transform = transform
        self.data_info = pd.read_csv( dataset_dir+csv_filename, usecols=[0],
                                      names=['Image Name'])
        # First column contains the image paths
        self.image_arr = np.asarray(self.dataset_dir+self.data_info.iloc[:, 0]+"_rgb.png")
        # First column contains the image paths
        self.label_arr = np.asarray(self.dataset_dir+self.data_info.iloc[:, 0]+"_label.png")
        # Calculate len
        self.data_len = len(self.data_info.index)


        self.data_enrichment=data_enrichment
        self.categorical = categorical
    def __getitem__(self, index):
        # We have to avoid numpy sharing a random number seed between the processes.
        # Ask the OS for a random number seed.
        #np.random.seed(random.SystemRandom().randint(0,2**32))
        #index = np.random.randint(self.image_arr.shape[0])
```

```
        img_name = self.image_arr[index]
        image = skio.imread(img_name)
        image = self.transform(image)
        label_image_name = self.label_arr[index]
        label_image = skio.imread(label_image_name)
        #Resize
        image = resize(image, (self.height, self.width))
        label_image = resize(label_image, (self.height, self.width))
        #If original image is very small, do erosion to get better masks
        do_erosion = False
        if image.shape[0] < 70:
            do_erosion =True
        mask  = create_mask(label_image, do_erosion)
        if (self.categorical):
            mask = np.dstack([1-mask, mask]).astype(np.float32)
        if self.data_enrichment and np.random.randint(3) == 1:
            image, mask = random_flips([image, mask])
            brightness = np.random.uniform(0.7, 1.2)
            image = adjust_brightness(image, brightness)
        image = image.transpose((2,0,1))
        mask = mask.transpose((2,0,1))
        return { "image":Tensor(image.copy()), "label":Tensor(mask.copy()) }
    def __len__(self):
        return self.data_len # of how many examples(images?) you have
#Instatiate the dataset
# # Define custom dataset
custom_dataset = Ara2013Dataset('data/Ara2013-Canon-Dataset/', 'Metadata.csv', data_enrichment=True)
# Define data loader
my_dataset_loader = torch.utils.data.DataLoader(dataset=custom_dataset,
                                                batch_size=4,
                                                shuffle=False)
def imshow(img):
    print(img.shape)
    #img=img*0.5 + 0.5 # un-normalize
    npimg = img.numpy()
    print(npimg.shape)
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
# get some random training images
```

```
dataiter = iter(my_dataset_loader)
sample = next(dataiter)
images = sample['image']
labels = sample['label']
imshow(torchvision.utils.make_grid(images))
plt.show()
imshow(torchvision.utils.make_grid(labels))
plt.show()
#The dataset alredy contains the Training/Validation/Test set information in corresponding csv files
# # Define custom dataset
train_dataset = Ara2013Dataset('data/Ara2013-Canon-Dataset/', 'Metadata_train.csv',
data_enrichment=True)
val_dataset = Ara2013Dataset('data/Ara2013-Canon-Dataset/', 'Metadata_validate.csv',
data_enrichment=False)
test_dataset = Ara2013Dataset('data/Ara2013-Canon-Dataset/', 'Metadata_test.csv',
data_enrichment=False)
print("Traning Dataset size: ", train_dataset.__len__())
print("Validation Dataset size: ",val_dataset.__len__())
print("Test Dataset size: ",test_dataset.__len__())
# Define data loader
train_dataset_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=30, shuffle=True)
val_dataset_loader = torch.utils.data.DataLoader(dataset=val_dataset, batch_size=15, shuffle=False)
test_dataset_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=5, shuffle=False)
plt.rcParams["figure.figsize"] = (10,10)
# for n in range(16):
for n in range(0,16,2):
    plt.subplot(4,4,n+1)
    batch = train_dataset[n]
    img = batch["image"].cpu().numpy()
    img = img / img.max()
    target = batch["label"].cpu().numpy()
    plt.imshow(img.transpose(1,2,0))
    plt.subplot(4,4,n+2)
    plt.imshow(target[0],cmap='gray')
#Create FCN
img_channels = 3
maps_1 = 8
maps_2 = 16
```

```
maps_3 = 32
if True:
    model = nn.Sequential(
        #Convolutional Layers
        nn.Conv2d(3, maps_1, 3, padding=1), nn.ReLU(),
        nn.Conv2d(maps_1, maps_2, 3, padding=1), nn.ReLU(), nn.BatchNorm2d(maps_2), nn.Dropout2d(),
nn.MaxPool2d(2),
        nn.Conv2d(maps_2, maps_3, 3, padding=1), nn.ReLU(), nn.BatchNorm2d(maps_3), nn.Dropout2d(),
nn.MaxPool2d(2),
        nn.Conv2d(maps_3, maps_3, 3, padding=1), nn.ReLU(), nn.BatchNorm2d(maps_3), nn.Dropout2d(),
        #Deconvolutional Layers
        nn.ConvTranspose2d(maps_3, maps_2, 3, stride=2, padding=1, output_padding=1),
        nn.Conv2d(maps_2, maps_2, 3, padding=1), nn.ReLU(), nn.BatchNorm2d(maps_2), nn.Dropout2d(),
        nn.ConvTranspose2d(maps_2, maps_1, 3, stride=2, padding=1, output_padding=1),
nn.BatchNorm2d(maps_1), nn.Dropout2d(),
        nn.Conv2d(maps_1, maps_1, 3, padding=1), nn.ReLU(), nn.BatchNorm2d(maps_1), nn.Dropout2d(),
        nn.Conv2d(maps_1, maps_1, 3, padding=1), nn.ReLU(), nn.BatchNorm2d(maps_1), nn.Dropout2d(),
        nn.Conv2d(maps_1, maps_1, 3, padding=1), nn.ReLU(), nn.BatchNorm2d(maps_1), nn.Dropout2d(),
        nn.Conv2d(maps_1, 1, 3, padding=1),
        #Classification Decision 注意這邊的寫法和一般的寫法不同
        nn.LogSigmoid()
    )
enable_cuda = False
if torch.cuda.is_available():
    model.cuda()
    enable_cuda = True
#optimizer = torch.optim.Adam(params=model.parameters(),lr=0.001)
optimizer = torch.optim.Adam(params=model.parameters(),lr=0.1)
#optimizer = torch.optim.Adam(params=model.parameters(),lr=0.0001)
# optimizer = torch.optim.Adam(params=model.parameters(),lr=0.00001)
weighting = 1.0#Variable(Tensor(np.array([0.001, 1.0])[np.newaxis, :, np.newaxis, np.newaxis]),
requires_grad=False).cuda()
def binary_cross_entropy_with_logits(input, target, weight=None, size_average=True, reduce=True):
    r"""Function that measures Binary Cross Entropy between target and output
    logits.
    See :class:`~torch.nn.BCEWithLogitsLoss` for details.
    Args:
        input: Variable of arbitrary shape
```

```
        target: Variable of the same shape as input
        weight (Variable, optional): a manual rescaling weight
                if provided it's repeated to match input tensor shape
        size_average (bool, optional): By default, the losses are averaged
                over observations for each minibatch. However, if the field
                sizeAverage is set to False, the losses are instead summed
                for each minibatch. Default: ``True``
        reduce (bool, optional): By default, the losses are averaged or summed over
                observations for each minibatch depending on size_average. When reduce
                is False, returns a loss per input/target element instead and ignores
                size_average. Default: True
    Examples::
         >>> input = autograd.Variable(torch.randn(3), requires_grad=True)
         >>> target = autograd.Variable(torch.FloatTensor(3).random_(2))
         >>> loss = F.binary_cross_entropy_with_logits(input, target)
         >>> loss.backward()
    """
    if not (target.size() == input.size()):
        raise ValueError("Target size ({}) must be the same as input size ({})".format(target.size(),
input.size()))

    max_val = (-input).clamp(min=0)
    loss = input - input * target + max_val + ((-max_val).exp() + (-input - max_val).exp()).log()
    if weight is not None:
        loss = loss * weight
    if not reduce:
        return loss
    elif size_average:
        return loss.mean()
    else:
        return loss.sum()
def calc_loss(model_output, target_var):
  loss = binary_cross_entropy_with_logits(model_output, target_var, reduce=False)
  #loss = loss * target_var * 11 + loss # class 1 is weighted by 12, class 0 by 1
  #loss = loss * target_var * 2 + loss # class 1 is weighted by 3, class 0 by 1
  #loss =  loss - loss * target_var * 0.4  # class 1 is weighted by 0.6, class 0 by 1 # EXPERIMENTAL -
Reduce weight of plant class
  loss = loss.mean()
```

```
  return loss
epoch = 0
all_train_losses = []
all_validation_losses = []
epoch_list = []
run_training=True
plot_every = 10
try:
    while run_training:
        epoch = epoch + 1
        if (epoch % plot_every == 0) and (len(epoch_list) >= 2):
                clear_output()
                plt.plot(epoch_list, all_train_losses)
                plt.plot(epoch_list, all_validation_losses)
                plt.xlabel("Epoch number")
                plt.ylabel("Cost")
                plt.legend(["Train", "Validation"])
                plt.show()
        print("Epoch %d: Train[" % epoch, end="")
        dataloader = torch.utils.data.DataLoader(train_dataset, batch_size=4, shuffle=True, num_workers
= 0,
                                                 pin_memory=enable_cuda)
        train_losses = []
        model.train()
        for n_batch, the_batch in enumerate(dataloader):
            optimizer.zero_grad()
            input_var = Variable(the_batch["image"], requires_grad=False)
            if enable_cuda:
                input_var = input_var.cuda()
            model_output = model.forward(input_var)
            target_var = Variable(the_batch["label"], requires_grad=False)
            if enable_cuda:
                target_var = target_var.cuda()
            loss = calc_loss(model_output, target_var)
            loss.backward()
            optimizer.step()
            loss_cpu = loss.data.cpu().item()#[0]
            train_losses.append(loss_cpu)
```

```
            print("*", end="")
        print("] Validation[", end="")
        dataloader = torch.utils.data.DataLoader(val_dataset, batch_size=4, num_workers = 0,
pin_memory=enable_cuda)
        validation_losses = []
        model.eval()
        for n_batch, the_batch in enumerate(dataloader):
            print("*", end="")
            input_var = Variable(the_batch["image"], requires_grad=False)
            if enable_cuda:
                input_var = input_var.cuda()
            model_output = model.forward(input_var)
            target_var = Variable(the_batch["label"], requires_grad=False)
            if enable_cuda:
                target_var = target_var.cuda()


            loss = calc_loss(model_output, target_var)#.detach()
            loss_cpu = loss.data.cpu().item()#[0]
            validation_losses.append(loss_cpu)
        print("] ", end="")
        epoch_list.append(epoch)
        all_train_losses.append(np.mean(train_losses))
        all_validation_losses.append(np.mean(validation_losses))
        print(" Training Loss = %a, Validation Loss = %a" % (all_train_losses[-1],
all_validation_losses[-1]))
except KeyboardInterrupt:
    print("Training stopped by user.")
    del dataloader
plt.plot(epoch_list, all_train_losses)
plt.plot(epoch_list, all_validation_losses)
plt.xlabel("Epoch number")
plt.ylabel("Cost")
plt.legend(["Train", "Validation"])
plt.show()
# Let's visualize some random test images
dataiter = iter(test_dataset_loader)
sample = next(dataiter)
test_images = sample['image']
```

```
labels = sample['label']
#show images
plt.figure(figsize=(15,15))
test_image_set = test_images
imshow(torchvision.utils.make_grid(test_image_set))
plt.show()
#show labels
plt.figure(figsize=(15,15))
label_image_set = labels
imshow(torchvision.utils.make_grid(label_image_set))
plt.show()
#show predictions
model.eval()
outputs = model(Variable(test_image_set.cuda(), requires_grad=False))
outputs_cpu = torch.sigmoid(outputs).data.cpu()
plt.figure(figsize=(15,15))
imshow(torchvision.utils.make_grid(outputs_cpu) )
plt.show()
#------------------------------------------------------------------
plt.figure(figsize=(20,20))
for i in range(len(label_image_set[:])-1):
    in_image = test_image_set[i].cpu().numpy()
    in_image =  in_image.transpose(1,2,0)
    in_target = labels[i].cpu().numpy()[0]
    img = in_image[:,:,1]
    img = img / img.max()
    predicted = outputs_cpu[i][0].cpu().numpy()
    final_img = np.dstack([predicted*2, img/3, in_target])
#     final_img = np.dstack([predicted*1, img/3, in_image[:,:,2]])
    #plt.subplot(2,len(label_image_set[:]-1)/2, i+1)
    plt.subplot(2,(len(label_image_set[:]))//2, i+1)
    plt.imshow(final_img)
IoU_threshold = 0.1
dataloader = torch.utils.data.DataLoader(test_dataset, batch_size=1, shuffle=False, num_workers = 0,
pin_memory=enable_cuda)
model.eval()
IoUs = []
for n_batch, the_batch in enumerate(dataloader):
```

```
    print("*", end="")
    input_var = Variable(the_batch["image"], requires_grad=False)
    if enable_cuda:
        input_var = input_var.cuda()
    model_output = model.forward(input_var)
    thresholded = (torch.sigmoid(model_output)>=IoU_threshold).cpu().numpy()
    lab = the_batch["label"].cpu().numpy()
    intersection = np.sum(np.logical_and(thresholded, lab))
    union = np.sum(np.logical_or(thresholded, lab))
    IoUs.append(intersection / union)
print("\nMean IoU: %f" % np.mean(IoUs))
```

■ Day4_1

✓ Ex1_MaskRCNN_R2P.ipynb

```
!python -m pip install pyyaml==5.1
import sys, os, distutils.core
# Note: This is a faster way to install detectron2 in Colab, but it does not include all
functionalities (e.g. compiled operators).
# See https://detectron2.readthedocs.io/tutorials/install.html for full installation instructions
!git clone 'https://github.com/facebookresearch/detectron2'
dist = distutils.core.run_setup("./detectron2/setup.py")
!python -m pip install {' '.join([f"'{x}'" for x in dist.install_requires])}
sys.path.insert(0, os.path.abspath('./detectron2'))
# Properly install detectron2. (Please do not install twice in both ways)
# !python -m pip install 'git+https://github.com/facebookresearch/detectron2.git'
import torch, detectron2
!nvcc --version
TORCH_VERSION = ".".join(torch.__version__.split(".")[:2])
CUDA_VERSION = torch.__version__.split("+")[-1]
print("torch: ", TORCH_VERSION, "; cuda: ", CUDA_VERSION)
print("detectron2:", detectron2.__version__)
print("torch cuda is available:",torch.cuda.is_available())
# Setup detectron2 logger
import detectron2
from detectron2.utils.logger import setup_logger
setup_logger()
```

```
# import some common libraries
import numpy as np
import os, json, cv2, random
from google.colab.patches import cv2_imshow
import matplotlib.pyplot as plt
import urllib.request
# import some common detectron2 utilities
from detectron2 import model_zoo
from detectron2.engine import DefaultTrainer, DefaultPredictor
from detectron2.config import get_cfg
from detectron2.utils.visualizer import Visualizer, ColorMode
from detectron2.data import MetadataCatalog, DatasetCatalog, build_detection_test_loader
from detectron2.data.datasets import register_coco_instances
from demo.predictor import VisualizationDemo
from detectron2.evaluation import COCOEvaluator, inference_on_dataset
from detectron2.modeling import build_model
from detectron2.checkpoint import DetectionCheckpointer
from detectron2.engine.hooks import HookBase
import detectron2.utils.comm as comm
data_url = "https://figshare.com/ndownloader/files/41454756?private_link=b363f8788965fbf541fd"
target_file = "R2P_data.tar.gz"
urllib.request.urlretrieve(data_url, target_file)
# UNPACK DATA
!tar xvf R2P_data.tar.gz
# REGISTER DATASETS FOR TRAINING AND VALIDATION
dataset_path = "/content/"
dataset_name = "R2P"
classes = ['fruit']
register_coco_instances(dataset_name+"_train", {}, dataset_path+"anns/train_1class.JSON",\
                        dataset_path+"RGB")
train_metadata = MetadataCatalog.get(dataset_name+"_train")
dataset_dicts_train = DatasetCatalog.get(dataset_name+"_train")
register_coco_instances(dataset_name+"_val", {}, dataset_path+"anns/val_1class.JSON",\
                        dataset_path+"RGB")
test_metadata = MetadataCatalog.get(dataset_name+"_val")
dataset_dicts_val = DatasetCatalog.get(dataset_name+"_val")
# VISUALIZE A COUPLE OF IMAGES  WITH OVERLAID GT
num_disp = 2
```

```
fig, axs = plt.subplots(num_disp, 2, figsize=(20,10) )
imcount = 0
for d in random.sample(dataset_dicts_train, 2):
    img = cv2.imread(d["file_name"])
    axs[imcount,0].imshow( img[:, :, ::-1] )
    visualizer = Visualizer(img[:, :, ::-1], metadata=train_metadata, scale=0.5)
    vis = visualizer.draw_dataset_dict(d)
    axs[imcount,1].imshow( vis.get_image() )
    imcount = imcount + 1
# Define custom trainer function to call evaluator
from detectron2.evaluation import (
    CityscapesInstanceEvaluator,
    CityscapesSemSegEvaluator,
    COCOEvaluator,
    COCOPanopticEvaluator,
    DatasetEvaluators,
    LVISEvaluator,
    PascalVOCDetectionEvaluator,
    SemSegEvaluator,
    verify_results,
)
from detectron2.modeling import GeneralizedRCNNWithTTA
def build_evaluator(cfg, dataset_name, output_folder=None):
    """
    Create evaluator(s) for a given dataset.
    This uses the special metadata "evaluator_type" associated with each builtin dataset.
    For your own dataset, you can simply create an evaluator manually in your
    script and do not have to worry about the hacky if-else logic here.
    """
    if output_folder is None:
        output_folder = os.path.join(cfg.OUTPUT_DIR, "inference")
    evaluator_list = []
    evaluator_type = MetadataCatalog.get(dataset_name).evaluator_type
    if evaluator_type in ["sem_seg", "coco_panoptic_seg"]:
        evaluator_list.append(
            SemSegEvaluator(
                dataset_name,
                distributed=True,
```

```
                output_dir=output_folder,
            )
        )
    if evaluator_type in ["coco", "coco_panoptic_seg"]:
        evaluator_list.append(COCOEvaluator(dataset_name, output_dir=output_folder))
    if evaluator_type == "coco_panoptic_seg":
        evaluator_list.append(COCOPanopticEvaluator(dataset_name, output_folder))
    if evaluator_type == "cityscapes_instance":
        return CityscapesInstanceEvaluator(dataset_name)
    if evaluator_type == "cityscapes_sem_seg":
        return CityscapesSemSegEvaluator(dataset_name)
    elif evaluator_type == "pascal_voc":
        return PascalVOCDetectionEvaluator(dataset_name)
    elif evaluator_type == "lvis":
        return LVISEvaluator(dataset_name, output_dir=output_folder)
    if len(evaluator_list) == 0:
        raise NotImplementedError(
            "no Evaluator for the dataset {} with the type {}".format(dataset_name, evaluator_type)
        )
    elif len(evaluator_list) == 1:
        return evaluator_list[0]
    return DatasetEvaluators(evaluator_list)
class Trainer(DefaultTrainer):
    """
    We use the "DefaultTrainer" which contains pre-defined default logic for
    standard training workflow. They may not work for you, especially if you
    are working on a new research project. In that case you can write your
    own training loop. You can use "tools/plain_train_net.py" as an example.
    """
    @classmethod
    def build_evaluator(cls, cfg, dataset_name, output_folder=None):
        return build_evaluator(cfg, dataset_name, output_folder)
    @classmethod
    def test_with_TTA(cls, cfg, model):
        logger = logging.getLogger("detectron2.trainer")
        # In the end of training, run an evaluation with TTA
        # Only support some R-CNN models.
        logger.info("Running inference with test-time augmentation ...")
```

```
        model = GeneralizedRCNNWithTTA(cfg, model)
        evaluators = [
            cls.build_evaluator(
                cfg, name, output_folder=os.path.join(cfg.OUTPUT_DIR, "inference_TTA")
            )
            for name in cfg.DATASETS.TEST
        ]
        res = cls.test(cfg, model, evaluators)
        res = OrderedDict({k + "_TTA": v for k, v in res.items()})
        return res
# MODEL CONFIGURATION
selected_model_config_yaml = "COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml"
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file(selected_model_config_yaml))
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url(selected_model_config_yaml)
cfg.DATASETS.TRAIN = (dataset_name+"_train",)
cfg.DATASETS.TEST = (dataset_name+"_val",)
cfg.DATALOADER.NUM_WORKERS = 4
cfg.DATALOADER.SAMPLER_TRAIN = "TrainingSampler"
cfg.SOLVER.STEPS = (2500, 4000)
cfg.SOLVER.CHECKPOINT_PERIOD = cfg.SOLVER.MAX_ITER+1
cfg.TEST.EVAL_PERIOD = 50 # no. of iterations at which to run evaluationon val set
# LR related parameters
cfg.SOLVER.BASE_LR = 0.002
cfg.SOLVER.GAMMA = 0.2
cfg.WEIGHT_DECAY = 0.0001
cfg.SOLVER.LR_POLICY = 'steps_with_decay'
cfg.SOLVER.WARMUP_ITERS = 200
# adjust these parameters in case of memory issues.
cfg.SOLVER.IMS_PER_BATCH = 2
cfg.SOLVER.MAX_ITER = 300
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 2048
cfg.MODEL.ROI_HEADS.NUM_CLASSES = len(classes)  # no. of classes
MetadataCatalog.get(dataset_name+"_train").evaluator_type = "coco"
MetadataCatalog.get(dataset_name+"_train").thing_classes = classes
MetadataCatalog.get(dataset_name+"_val").evaluator_type = "coco"
MetadataCatalog.get(dataset_name+"_val").thing_classes = classes
os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
```

```
trainer = Trainer(cfg)
trainer.resume_or_load(resume=False)
trainer.train()
# Look at training curves in tensorboard:
%load_ext tensorboard
%tensorboard --logdir output
weightsfolder = '/content/output/'
os.listdir(weightsfolder)
cfg = get_cfg()
cfg.merge_from_file( model_zoo.get_config_file(selected_model_config_yaml) )
cfg.MODEL.ROI_HEADS.NUM_CLASSES = len(classes)
cfg.MODEL.WEIGHTS = os.path.join(weightsfolder, "model_final.pth")
cfg.DATASETS.TEST = ("R2P_val",)
model = build_model(cfg)
checkpointer = DetectionCheckpointer(model)
checkpointer.load(cfg.MODEL.WEIGHTS)
predictor = DefaultPredictor(cfg)
# VISUALIZE A COUPLE OF IMAGES  WITH OVERLAID DETECTION RESULTS
num_disp = 2
fig, axs = plt.subplots(num_disp, 2, figsize=(20,10) )
imcount = 0
for d in random.sample(dataset_dicts_val, num_disp):
    img = cv2.imread(d["file_name"])
    outputs = predictor(img)
    axs[imcount,0].imshow( img[:, :, ::-1] )
    visualizer = Visualizer(img[:, :, ::-1], metadata=test_metadata, scale=0.8)
    vis = visualizer.draw_instance_predictions(outputs["instances"].to("cpu"))
    axs[imcount,1].imshow( vis.get_image() )
    imcount = imcount + 1
from google.colab import drive
drive.mount('/content/drive')
evaluator = COCOEvaluator("R2P_val", ("bbox", "segm"), False)
val_loader = build_detection_test_loader(cfg, "R2P_val")
eval_results = inference_on_dataset(trainer.model, val_loader, evaluator)
```

■ Day4_1

- ✓ Ex2_MaskRCNN_R2P.ipynb

```
!python -m pip install pyyaml==5.1
import sys, os, distutils.core
# Note: This is a faster way to install detectron2 in Colab, but it does not include all
functionalities (e.g. compiled operators).
# See https://detectron2.readthedocs.io/tutorials/install.html for full installation instructions
!git clone 'https://github.com/facebookresearch/detectron2'
dist = distutils.core.run_setup("./detectron2/setup.py")
!python -m pip install {' '.join([f"'{x}'" for x in dist.install_requires])}
sys.path.insert(0, os.path.abspath('./detectron2'))
# Properly install detectron2. (Please do not install twice in both ways)
# !python -m pip install 'git+https://github.com/facebookresearch/detectron2.git'
import torch, detectron2
!nvcc --version
TORCH_VERSION = ".".join(torch.__version__.split(".")[:2])
CUDA_VERSION = torch.__version__.split("+")[-1]
print("torch: ", TORCH_VERSION, "; cuda: ", CUDA_VERSION)
print("detectron2:", detectron2.__version__)
print("torch cuda is available:",torch.cuda.is_available())
# Setup detectron2 logger
import detectron2
from detectron2.utils.logger import setup_logger
setup_logger()
# import some common libraries
import numpy as np
import os, json, cv2, random
from google.colab.patches import cv2_imshow
import matplotlib.pyplot as plt
import urllib.request
# import some common detectron2 utilities
from detectron2 import model_zoo
from detectron2.engine import DefaultTrainer, DefaultPredictor
from detectron2.config import get_cfg
from detectron2.utils.visualizer import Visualizer, ColorMode, GenericMask
from detectron2.data import MetadataCatalog, DatasetCatalog, build_detection_test_loader
from detectron2.data.datasets import register_coco_instances
from demo.predictor import VisualizationDemo
from detectron2.evaluation import COCOEvaluator, inference_on_dataset
from detectron2.modeling import build_model
```

```
from detectron2.checkpoint import DetectionCheckpointer
from detectron2.engine.hooks import HookBase
import detectron2.utils.comm as comm
data_url = "https://figshare.com/ndownloader/files/41454756?private_link=b363f8788965fbf541fd"
target_file = "R2P_data.tar.gz"
urllib.request.urlretrieve(data_url, target_file)
# UNPACK DATA
!tar xvf R2P_data.tar.gz
# REGISTER DATASETS FOR TRAINING AND VALIDATION
dataset_path = "/content/"
dataset_name = "R2P"
classes = ['redfruit','greenfruit']
register_coco_instances(dataset_name+"_train", {}, dataset_path+"anns/train_2class.JSON",\
                        dataset_path+"RGB")
train_metadata = MetadataCatalog.get(dataset_name+"_train")
dataset_dicts_train = DatasetCatalog.get(dataset_name+"_train")
register_coco_instances(dataset_name+"_val", {}, dataset_path+"anns/val_2class.JSON",\
                        dataset_path+"RGB")
test_metadata = MetadataCatalog.get(dataset_name+"_val")
dataset_dicts_val = DatasetCatalog.get(dataset_name+"_val")
# VISUALIZE A COUPLE OF IMAGES  WITH OVERLAID GT
MetadataCatalog.get(dataset_name+"_train").thing_classes = classes
MetadataCatalog.get(dataset_name+"_train").thing_colors =
[(255,0,0),(0,255,0),(0,0,255),(255,0,255),(156,225,242)]
num_disp = 2
fig, axs = plt.subplots(num_disp, 2, figsize=(20,10) )
imcount = 0
for d in random.sample(dataset_dicts_train, 2):
    img = cv2.imread(d["file_name"])
    axs[imcount,0].imshow( img[:, :, ::-1] )
    visualizer = Visualizer(img[:, :, ::-1], MetadataCatalog.get(dataset_name+"_train"), scale=0.5,
instance_mode=ColorMode.SEGMENTATION )
    vis = visualizer.draw_dataset_dict(d)
    axs[imcount,1].imshow( vis.get_image() )
    imcount = imcount + 1
# Define custom trainer function to call evaluator
from detectron2.evaluation import (
    CityscapesInstanceEvaluator,
```

```
    CityscapesSemSegEvaluator,
    COCOEvaluator,
    COCOPanopticEvaluator,
    DatasetEvaluators,
    LVISEvaluator,
    PascalVOCDetectionEvaluator,
    SemSegEvaluator,
    verify_results,
)
from detectron2.modeling import GeneralizedRCNNWithTTA
def build_evaluator(cfg, dataset_name, output_folder=None):
    """
    Create evaluator(s) for a given dataset.
    This uses the special metadata "evaluator_type" associated with each builtin dataset.
    For your own dataset, you can simply create an evaluator manually in your
    script and do not have to worry about the hacky if-else logic here.
    """
    if output_folder is None:
        output_folder = os.path.join(cfg.OUTPUT_DIR, "inference")
    evaluator_list = []
    evaluator_type = MetadataCatalog.get(dataset_name).evaluator_type
    if evaluator_type in ["sem_seg", "coco_panoptic_seg"]:
        evaluator_list.append(
            SemSegEvaluator(
                dataset_name,
                distributed=True,
                output_dir=output_folder,
            )
        )
    if evaluator_type in ["coco", "coco_panoptic_seg"]:
        evaluator_list.append(COCOEvaluator(dataset_name, output_dir=output_folder))
    if evaluator_type == "coco_panoptic_seg":
        evaluator_list.append(COCOPanopticEvaluator(dataset_name, output_folder))
    if evaluator_type == "cityscapes_instance":
        return CityscapesInstanceEvaluator(dataset_name)
    if evaluator_type == "cityscapes_sem_seg":
        return CityscapesSemSegEvaluator(dataset_name)
    elif evaluator_type == "pascal_voc":
```

```
        return PascalVOCDetectionEvaluator(dataset_name)
    elif evaluator_type == "lvis":
        return LVISEvaluator(dataset_name, output_dir=output_folder)
    if len(evaluator_list) == 0:
        raise NotImplementedError(
            "no Evaluator for the dataset {} with the type {}".format(dataset_name, evaluator_type)
        )
    elif len(evaluator_list) == 1:
        return evaluator_list[0]
    return DatasetEvaluators(evaluator_list)
class Trainer(DefaultTrainer):
    """
    We use the "DefaultTrainer" which contains pre-defined default logic for
    standard training workflow. They may not work for you, especially if you
    are working on a new research project. In that case you can write your
    own training loop. You can use "tools/plain_train_net.py" as an example.
    """
    @classmethod
    def build_evaluator(cls, cfg, dataset_name, output_folder=None):
        return build_evaluator(cfg, dataset_name, output_folder)
    @classmethod
    def test_with_TTA(cls, cfg, model):
        logger = logging.getLogger("detectron2.trainer")
        # In the end of training, run an evaluation with TTA
        # Only support some R-CNN models.
        logger.info("Running inference with test-time augmentation ...")
        model = GeneralizedRCNNWithTTA(cfg, model)
        evaluators = [
            cls.build_evaluator(
                cfg, name, output_folder=os.path.join(cfg.OUTPUT_DIR, "inference_TTA")
            )
            for name in cfg.DATASETS.TEST
        ]
        res = cls.test(cfg, model, evaluators)
        res = OrderedDict({k + "_TTA": v for k, v in res.items()})
        return res
# MODEL CONFIGURATION
selected_model_config_yaml = "COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml"
```

```
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file(selected_model_config_yaml))
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url(selected_model_config_yaml)
cfg.DATASETS.TRAIN = (dataset_name+"_train",)
cfg.DATASETS.TEST = (dataset_name+"_val",)
cfg.DATALOADER.NUM_WORKERS = 2
cfg.DATALOADER.SAMPLER_TRAIN = "TrainingSampler"
cfg.SOLVER.STEPS = (2500, 4000)
cfg.SOLVER.CHECKPOINT_PERIOD = cfg.SOLVER.MAX_ITER+1
cfg.TEST.EVAL_PERIOD = 50 # no. of iterations at which to run evaluationon val set
# LR related parameters
cfg.SOLVER.BASE_LR = 0.002
cfg.SOLVER.GAMMA = 0.2
cfg.WEIGHT_DECAY = 0.0001
cfg.SOLVER.LR_POLICY = 'steps_with_decay'
cfg.SOLVER.WARMUP_ITERS = 100
# adjust these parameters in case of memory issues.
cfg.SOLVER.IMS_PER_BATCH = 1
cfg.SOLVER.MAX_ITER = 300
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 1024
cfg.MODEL.ROI_HEADS.NUM_CLASSES = len(classes)  # no. of classes
MetadataCatalog.get(dataset_name+"_train").evaluator_type = "coco"
MetadataCatalog.get(dataset_name+"_train").thing_classes = classes
MetadataCatalog.get(dataset_name+"_val").evaluator_type = "coco"
MetadataCatalog.get(dataset_name+"_val").thing_classes = classes
os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
trainer = Trainer(cfg)
trainer.resume_or_load(resume=False)
trainer.train()
# Look at training curves in tensorboard:
%load_ext tensorboard
%tensorboard --logdir output
weightsfolder = '/content/output/'
os.listdir(weightsfolder)
cfg = get_cfg()
cfg.merge_from_file( model_zoo.get_config_file(selected_model_config_yaml) )
cfg.MODEL.ROI_HEADS.NUM_CLASSES = len(classes)
cfg.MODEL.WEIGHTS = os.path.join(weightsfolder, "model_final.pth")
```

```
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.8   # set the testing threshold for this model
cfg.MODEL.ROI_HEADS.NMS_THRESH_TEST = 0.3
cfg.DATASETS.TEST = ("R2P_val",)
model = build_model(cfg)
checkpointer = DetectionCheckpointer(model)
checkpointer.load(cfg.MODEL.WEIGHTS)
predictor = DefaultPredictor(cfg)
# VISUALIZE A COUPLE OF IMAGES  WITH OVERLAID DETECTION RESULTS
MetadataCatalog.get(dataset_name+"_val").thing_classes = classes
MetadataCatalog.get(dataset_name+"_val").thing_colors =
[(255,0,0),(0,255,0),(0,0,255),(255,0,255),(156,225,242)]
num_disp = 2
fig, axs = plt.subplots(num_disp, 2, figsize=(20,10) )
imcount = 0
for d in random.sample(dataset_dicts_val, num_disp):
    img = cv2.imread(d["file_name"])
    outputs = predictor(img)
    axs[imcount,0].imshow( img[:, :, ::-1] )
    visualizer = Visualizer(img[:, :, ::-1], MetadataCatalog.get(dataset_name+"_val"), scale=0.5,
instance_mode=ColorMode.SEGMENTATION )
    vis = visualizer.draw_instance_predictions(outputs["instances"].to("cpu"))
    axs[imcount,1].imshow( vis.get_image() )
    imcount = imcount + 1
evaluator = COCOEvaluator("R2P_val", ("bbox", "segm"), False)
val_loader = build_detection_test_loader(cfg, "R2P_val")
eval_results = inference_on_dataset(trainer.model, val_loader, evaluator)
```

■ Day4_1

✓ Ex3_MaskRCNN_R2P_pretrained.ipynb

```
!python -m pip install pyyaml==5.1
import sys, os, distutils.core
# Note: This is a faster way to install detectron2 in Colab, but it does not include all
functionalities (e.g. compiled operators).
# See https://detectron2.readthedocs.io/tutorials/install.html for full installation instructions
```

```
!git clone 'https://github.com/facebookresearch/detectron2'
dist = distutils.core.run_setup("./detectron2/setup.py")
!python -m pip install {' '.join([f"'{x}'" for x in dist.install_requires])}
sys.path.insert(0, os.path.abspath('./detectron2'))
# Properly install detectron2. (Please do not install twice in both ways)
# !python -m pip install 'git+https://github.com/facebookresearch/detectron2.git'
import torch, detectron2
!nvcc --version
TORCH_VERSION = ".".join(torch.__version__.split(".")[:2])
CUDA_VERSION = torch.__version__.split("+")[-1]
print("torch: ", TORCH_VERSION, "; cuda: ", CUDA_VERSION)
print("detectron2:", detectron2.__version__)
print("torch cuda is available:",torch.cuda.is_available())
# Setup detectron2 logger
import detectron2
from detectron2.utils.logger import setup_logger
setup_logger()
# import some common libraries
import numpy as np
import os, json, cv2, random
from google.colab.patches import cv2_imshow
import matplotlib.pyplot as plt
import urllib.request
# import some common detectron2 utilities
from detectron2 import model_zoo
from detectron2.engine import DefaultTrainer, DefaultPredictor
from detectron2.config import get_cfg
from detectron2.utils.visualizer import Visualizer, ColorMode, GenericMask
from detectron2.data import MetadataCatalog, DatasetCatalog, build_detection_test_loader
from detectron2.data.datasets import register_coco_instances
from demo.predictor import VisualizationDemo
from detectron2.evaluation import COCOEvaluator, inference_on_dataset
from detectron2.modeling import build_model
from detectron2.checkpoint import DetectionCheckpointer
from detectron2.engine.hooks import HookBase
import detectron2.utils.comm as comm
data_url = "https://figshare.com/ndownloader/files/41454756?private_link=b363f8788965fbf541fd"
target_file = "R2P_data.tar.gz"
```

```
urllib.request.urlretrieve(data_url, target_file)
# UNPACK DATA
!tar xvf R2P_data.tar.gz
# REGISTER DATASETS FOR TRAINING AND VALIDATION
dataset_path = "/content/"
dataset_name = "R2P"
classes_1 = ['fruit']
classes_2 = ['redfruit','greenfruit']
register_coco_instances(dataset_name+"_val_1class", {}, dataset_path+"anns/val_1class.JSON",\
                        dataset_path+"RGB")
val_1class_metadata = MetadataCatalog.get(dataset_name+"_val_1class")
dataset_dicts_val_1class = DatasetCatalog.get(dataset_name+"_val_1class")
register_coco_instances(dataset_name+"_val_2class", {}, dataset_path+"anns/val_2class.JSON",\
                        dataset_path+"RGB")
val_2class_metadata = MetadataCatalog.get(dataset_name+"_val_2class")
dataset_dicts_val_2class = DatasetCatalog.get(dataset_name+"_val_2class")
selected_model_config_yaml = "COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml"
cfg1 = get_cfg()
cfg1.merge_from_file( model_zoo.get_config_file(selected_model_config_yaml) )
cfg1.MODEL.ROI_HEADS.NUM_CLASSES = len(classes_1)
cfg1.MODEL.WEIGHTS = "/content/pretrained_models/R2P_1class_R50FPN3x.pth"
cfg1.DATASETS.TEST = ("R2P_val",)
model1 = build_model(cfg1)
checkpointer1 = DetectionCheckpointer(model1)
checkpointer1.load(cfg1.MODEL.WEIGHTS)
predictor1 = DefaultPredictor(cfg1)
cfg2 = get_cfg()
cfg2.merge_from_file( model_zoo.get_config_file(selected_model_config_yaml) )
cfg2.MODEL.ROI_HEADS.NUM_CLASSES = len(classes_2)
cfg2.MODEL.WEIGHTS = "/content/pretrained_models/R2P_2class_R50FPN3x.pth"
cfg2.DATASETS.TEST = ("R2P_val",)
model2 = build_model(cfg2)
checkpointer2 = DetectionCheckpointer(model2)
checkpointer2.load(cfg2.MODEL.WEIGHTS)
predictor2 = DefaultPredictor(cfg2)
MetadataCatalog.get(dataset_name+"_val_1class").thing_classes = classes_1
MetadataCatalog.get(dataset_name+"_val_1class").thing_colors =
[(0,0,255),(0,255,0),(255,0,0),(255,0,255),(156,225,242)]
```

```
MetadataCatalog.get(dataset_name+"_val_2class").thing_classes = classes_2
MetadataCatalog.get(dataset_name+"_val_2class").thing_colors =
[(255,0,0),(0,255,0),(0,0,255),(255,0,255),(156,225,242)]
num_disp = 2
fig, axs = plt.subplots(num_disp, 3, figsize=(30,10) )
imcount = 0
for d in random.sample(dataset_dicts_val_1class, num_disp):
    img = cv2.imread(d["file_name"])
    outputs1 = predictor1(img)
    outputs2 = predictor2(img)
    axs[imcount,0].imshow( img[:, :, ::-1] )
    visualizer1 = Visualizer(img[:, :, ::-1], MetadataCatalog.get(dataset_name+"_val_1class"),
scale=0.5, instance_mode=ColorMode.SEGMENTATION )
    vis1 = visualizer1.draw_instance_predictions(outputs1["instances"].to("cpu"))
    axs[imcount,1].imshow( vis1.get_image() )
    visualizer2 = Visualizer(img[:, :, ::-1], MetadataCatalog.get(dataset_name+"_val_2class"),
scale=0.5, instance_mode=ColorMode.SEGMENTATION )
    vis2 = visualizer2.draw_instance_predictions(outputs2["instances"].to("cpu"))
    axs[imcount,2].imshow( vis2.get_image() )
    imcount = imcount + 1
```