2025 年【科學探究競賽-這樣教我就懂】

普高組　成果報告格式

題目名稱：農業即時病害辨識系統

一、摘要

本研究採用跨領域研究方法，結合資訊技術與農業專業，運用深度學習提升水稻葉片病害辨識的準確度與效率。傳統病害檢測受限於人為誤差與環境影響，難以滿足大規模農地監測需求，因此本研究導入卷積神經網路（CNN），透過影像處理標註病害區域，並訓練模型自動提取病變特徵，以提升辨識效能。研究過程中，我們應用數據增強技術優化模型泛化能力，並結合農業專家回饋機制，確保系統應用的可靠性。實驗結果顯示，該方法能有效提高水稻病害識別度，降低農損，並優化農業管理流程。未來，本研究將結合物聯網（IoT）與雲端運算，實現即時監測與智慧決策，推動農業數位轉型，提升農作物生產效率與永續發展。

二、探究題目與動機

本研究旨在開發一款結合深度學習與生活科技的應用程式，專門針對水稻病害葉片進行即時識別，並提供相關病害簡介。水稻作為台灣重要的糧食作物，其病害控制對於糧食安全與農業穩定至關重要。然而，傳統病害診斷依賴農民經驗，且在大規模農田中難以有效執行，導致病害防治的準確性與效率低下。隨著人工智慧（AI）與卷積神經網路（CNN）在影像處理領域的應用，深度學習技術已成為提升病害識別率的關鍵工具。本研究希望藉由深度學習技術提升病害識別的準確性，簡化辨識流程，並降低專業門檻，為農民及大眾提供一個簡便易用的病害診斷系統，從而穩定作物產量，並推動智慧農業發展。

三、探究目的與假設

本研究旨在開發一套基於深度學習的水稻葉片病害辨識系統，提供農民一款操作簡單、輕便實用的應用程式，以提前偵測與預防病害，降低農損風險。此外，本研究亦期望將技術應用於農業教育，提升學生與農民的病害辨識能力，推動智慧農業的普及與發展。透過本研究，期望能有效降低農民的種植成本，優化農藥使用策略，並透過實地試驗驗證模型於實際農業環境中的可行性，提供後續優化依據。

（一）開發水稻葉片病害辨識模型

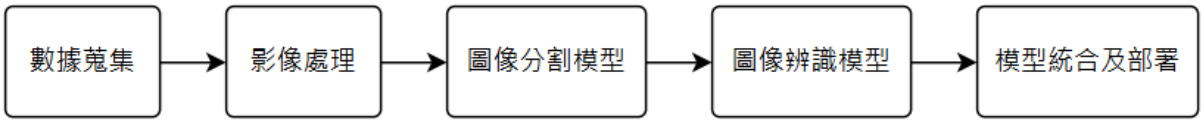
（二）建立病害辨識應用系統

（三）推動智慧農業技術應用

（四）優化病害防治策略，降低農藥使用

四、探究方法與驗證步驟

本研究的系統實作流程將以**應用科學研究**為核心，具體來說，它結合農業科學與人工智慧領域，屬於**跨學科的綜合性研究**。透過科學方法進行規劃與開發，確保最終系統在效能與實用性上的表現。系統實作流程包含以下步驟：



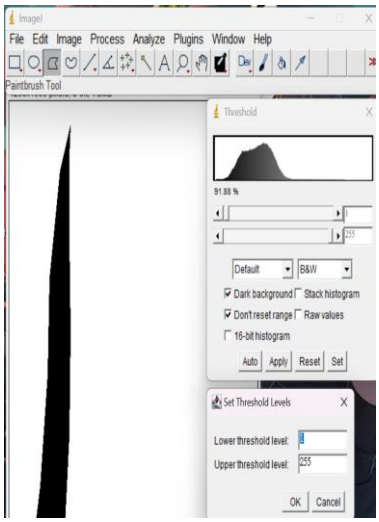
圖一：系統實作流程圖

（一）數據蒐集

我們在 Kaggle 上找到包含健康以及各疾病水稻葉面的圖片數據集 Rice Leaf Diseases Detection，挑出其中包含患有胡麻葉枯病 (brown spot) 和白葉枯病 (bacterial leaf blight) 與健康水稻的數據集。在初步取得需要的數據集後，我們進行了更精細的分類和劣質數據剃除，以利後續模型能學到更優質且準確的特徵。

（二）影像處理

為取得分割掩碼，對於精簡過後的數據集採取二值化手動分割處理，方法為用 ImageJ 標記水稻葉圖片，轉為二值化的掩碼圖像。標記過程為：將圖片的 Type 轉為 8-bit，接著手動標出水稻葉片的形狀，設為黑色，再將 lower threshold level 設為 1，背景就會全部變成白色。



圖二：ImageJ 處理影像畫面截圖

（三）圖像分割模型

本模型採用了 DenseNet-121 作為編碼器，並凍結部分權重以避免對訓練結果造成過大影響。解碼器部分則使用 Unet 架構，並加入自定義的 Resize 操作來調整特徵圖尺寸，配合 Dropout 層以防止過擬合，從而提高模型的穩定性和泛化能力。

```
#  使用dense121作編碼器
base_model  =  DenseNet121(weights='imagenet',  include_top=False,  input_tensor=inputs)
base_model.trainable  =  False    #  凍結權重

c1  =  base_model.get_layer('conv1_relu').output
c2  =  base_model.get_layer('pool2_relu').output
c3  =  base_model.get_layer('pool3_relu').output
c4  =  base_model.get_layer('pool4_relu').output
c5  =  base_model.get_layer('relu').output
```

圖三：編碼器程式片段

在深度學習領域中，DenseNet 是一種卷積神經網路 (Convolutional Neural Network, CNN) 架構，通過一種稱為「密集連接」的機制來提高模型的參數效率和信息流動性，其相當重要的一個特性是，每一層的輸出都直接傳遞給後續的所有層，這種機制確保了信息和梯度能夠在網路中更好地傳播，減少了梯度消失的問題。由於每一層接收了前面所有層的輸入

DenseNet 不需要像傳統的深度網路那樣依賴非常寬的層或非常深的網路來獲得豐富的特徵表示，因此它可以用較少的參數獲得類似或更好的性能。而 DenseNet121 是 DenseNet 家族中的一個較小模型，總共有 121 層，包含卷積層、池化層和密集連接塊，與其它較大版本的 DenseNet 相比，參數更少但仍具備良好表現，尤其適合運用在中小型數據集。

```
# 解碼器
u6 = UpSampling2D((2, 2))(c5)
u6 = ResizeLayer(target_size=(c4.shape[1], c4.shape[2]))(u6)
u6 = Concatenate()([u6, c4])
c6 = Conv2D(256, (3, 3), activation='relu', padding='same')(u6)
c6 = Conv2D(256, (3, 3), activation='relu', padding='same')(c6)
c6 = Conv2D(256, (3, 3), activation='relu', padding='same')(c6)
d6 = Dropout(0.4)(c6)

u7 = UpSampling2D((2, 2))(d6)
u7 = ResizeLayer(target_size=(c3.shape[1], c3.shape[2]))(u7)
u7 = Concatenate()([u7, c3])
c7 = Conv2D(128, (3, 3), activation='relu', padding='same')(u7)
c7 = Conv2D(128, (3, 3), activation='relu', padding='same')(c7)
c7 = Conv2D(128, (3, 3), activation='relu', padding='same')(c7)
d7 = Dropout(0.4)(c7)

u8 = UpSampling2D((2, 2))(d7)
u8 = ResizeLayer(target_size=(c2.shape[1], c2.shape[2]))(u8)
u8 = Concatenate()([u8, c2])
c8 = Conv2D(64, (3, 3), activation='relu', padding='same')(u8)
c8 = Conv2D(64, (3, 3), activation='relu', padding='same')(c8)
c8 = Conv2D(64, (3, 3), activation='relu', padding='same')(c8)
d8 = Dropout(0.4)(c8)

u9 = UpSampling2D((2, 2))(d8)
u9 = ResizeLayer(target_size=(c1.shape[1], c1.shape[2]))(u9)
u9 = Concatenate()([u9, c1])
c9 = Conv2D(32, (3, 3), activation='relu', padding='same')(u9)
c9 = Conv2D(32, (3, 3), activation='relu', padding='same')(c9)
c9 = Conv2D(32, (3, 3), activation='relu', padding='same')(c9)
d9 = Dropout(0.4)(c9)

u10 = UpSampling2D((2, 2))(d9)
outputs = Conv2D(1, (1, 1), activation='sigmoid')(u10)
```

圖四：解碼器程式片段

ResizeLayer 是一個自定義的 Keras 層，用來將輸入的圖片或特徵圖調整到指定的尺寸，在此層的初始化方法中，我們設置了目標尺寸 target_size，而在接下來的前向傳播過程，call 方法會使用 TensorFlow 的 tf.image.resize 函數將輸入張量調整到指定大小，這樣當此層被用於模型中時，所有通過這個層的數據都會被自動調整為指定的尺寸，確保輸入輸出的一致性並滿足後續網路層的需求。

```
@tf.keras.utils.register_keras_serializable()
class ResizeLayer(tf.keras.layers.Layer):
    def __init__(self, target_size, **kwargs):
        super(ResizeLayer, self).__init__(**kwargs)
        self.target_size = target_size

    def call(self, inputs):
        return tf.image.resize(inputs, self.target_size)
```

圖五：自定義層程式片段

訓練和構建模型主要基於 TensorFlow 上的 keras 來進行操作，且 numpy、Operating System Interfaces 和 matplotlib 也都會用到，TensorFlow 是由 Google Brain 開發的一個開源深度學習框架，廣泛運用於構建、訓練和部署模型，Keras 是 TensorFlow 的高級 API，提供了簡潔的接口，適合模型的快速搭建和訓練。 numpy 主要用於處理圖像數據和掩碼(mask) ，將它們從圖像文件轉為數組，並進行二值化處理，使得掩碼值為 True 或 False，再通過 astype() 函數把 True/False 轉為 1/0，讓掩碼符合深度學習模型需要的型態。os (Operating System Interfaces) 用於處理文件和路徑操作，從指定文件夾中讀取圖像和對

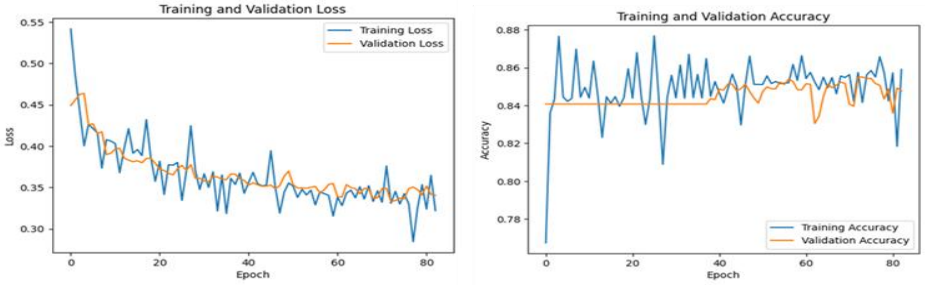
應的掩碼文件，並建構出完整的文件路徑。

```python
# 圖像處理
def load_images_and_masks(image_folder, mask_folder, image_size):
    images = []
    masks = []
    for filename in os.listdir(image_folder):
        if filename.endswith(".jpg") or filename.endswith(".png"):
            img_path = os.path.join(image_folder, filename)
            img = load_img(img_path, target_size=image_size)
            img = img_to_array(img) / 255.0

            base_filename = os.path.splitext(filename)[0]
            mask_path = os.path.join(mask_folder, base_filename + '_mask.png')
            mask = load_img(mask_path, color_mode="grayscale", target_size=image_size)
            mask = img_to_array(mask) / 255.0
            mask = (mask > 0.5).astype(np.uint8)
            images.append(img)
            masks.append(mask)

    return np.array(images), np.array(masks)
```

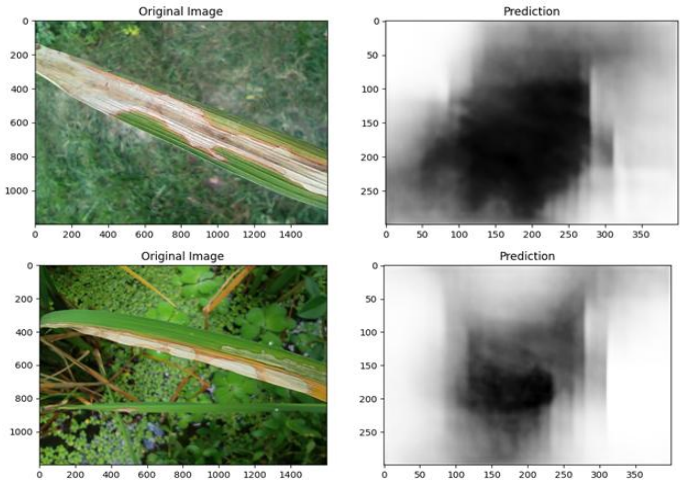
圖六：圖像處理部分程式片段

完成的模型在進行 compile 之前，除了正則化 Dropout 層，為了進一步防止過擬合，我們設置早停機制並把 patience 設為 5，此外，由於數據量較少，我們也額外使用了旋轉、平移、裁剪的數據增強來確保數據豐富度能支撐起這個模型的訓練。

進行多次調適和測試後，本研究最終採用的模型設定為 lr=0.0001，epoch=100，batch_size=16，數據集劃分 8:2，作為性能指標表現最佳的模型並儲存測試，由下圖 15 中能看到雖然波動幅度稍大，但 loss 率穩定地下降，直到第 83 epoch 趨近飽和，最終數值為 0.342，而 acc 之最終數值為 0.851。雖然在訓練測試中表現亮眼，但下圖 16 顯示的預測分割測試結果相當不盡人意，除了勉強能分割出中央輪廓以外，並不能很好地切割出整條水稻，因此確定此模型的真實泛化能力稍低。



圖七：可視化 acc 率和 loss 率圖表



圖八：分割模型預測結果

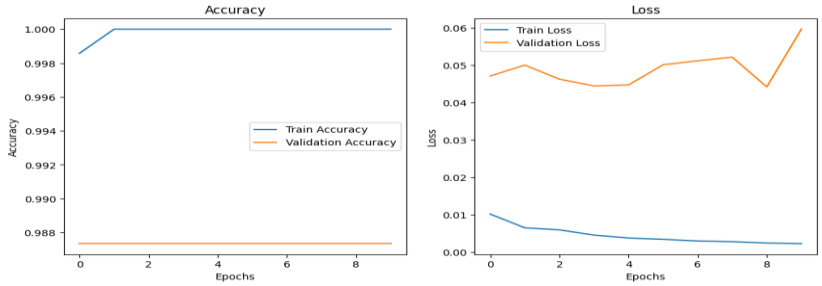
（四）**圖像辨識模型**

和圖像分割模型類似，同樣基於 TensorFlow 上的 keras，圖像辨識模型需要圖像處理、模型構建和訓練、最後預測並可視化結果。初步圖像處理即為調整數據大小、轉數組和文件路徑操作，一樣使用 numpy 和 os，在這之後將三類 (brown, white, health) 圖像數據和標籤進行合併、預處理、打亂順序，並劃分出 80%的數據作為訓練集和 20%的數據作為測試集，再從訓練集中劃分出 10%作為驗證集。我們創建的標籤設定為如果圖像是棕色斑點，即胡麻葉枯病，模型會輸出 0，白葉枯病是 1，健康的水稻則是 2。

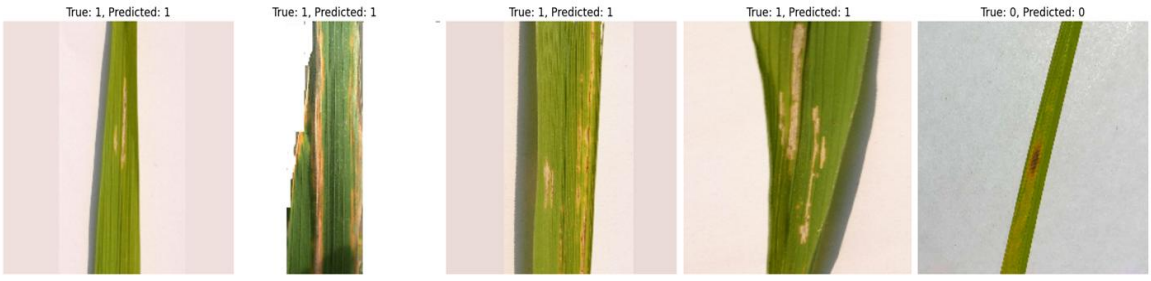
```
# 創建標籤
brown_spots_labels = np.array([0] * len(brown_spots_images))
white_spots_labels = np.array([1] * len(white_spots_images))
no_spots_labels = np.array([2] * len(no_spots_images))

# 合併標籤
images = np.concatenate((brown_spots_images, white_spots_images, no_spots_images), axis=0)
labels = np.concatenate((brown_spots_labels, white_spots_labels, no_spots_labels), axis=0)

# 歸一化
images = images / 255.0

# 轉成獨熱編碼
labels = to_categorical(labels, num_classes=3)

# 打亂數據
indices = np.arange(len(images))
np.random.shuffle(indices)
images = images[indices]
labels = labels[indices]

# 劃分數據集
from sklearn.model_selection import train_test_split
train_images, test_images, train_labels, test_labels = train_test_split(images, labels, test_size=0.2)
train_images, val_images, train_labels, val_labels = train_test_split(train_images, train_labels, test_size=0.1)
```

圖九：圖像處理部分程式片段

考量到再構建一個模型並訓練將消耗大量時間，性能也較為不穩定，我們選擇直接加載 DenseNet121 預訓練模型來進行遷移學習。先凍結模型中的所有層權重，防止在訓練過程中更新，並添加自訂義權連接層進行三類分類，這樣訓練完畢後，只有全連接層會被更新。經過調適和修正，最後利用 matplotlib 可視化結果，由下圖十確認 acc 率和 loss 率都維持在高水準的穩定狀態，並查看下圖十來檢查隨機預測結果的準確度。



圖十：可視化模型訓練性能結果（左圖為 acc 率，右圖為 loss 率）



圖十一：圖像辨識預測結果

（五）模型統合及部署

完成分割模型和辨識模型後，我們在 Visual Studio 2020 為平台來進行模型整合及部署，使用的 API 統合框架是 Flask，該過程旨在將訓練好的模型轉換為可供應用程式調用的形式，最終打包為可安裝的 APK，方便在移動端設備上使用。

圖十二：App 首頁及 App 預測結果

五、結論與生活應用

一、推動 SDG 4 - 優質教育：本研究所開發的 APP 可作為食農教育的教具，協助一般民眾認識水稻疾病

二、降低水稻水稻病害辨識的專業門檻，並協助農民精準辨識未知或不確定的病因

三、結合物聯網（IoT）與雲端運算，實現即時監測與智慧決策，推動農業數位轉型，以提升農作物生產效率與永續發展

四、推動 SDG12- 責任消費和生產：透過精準病害辨識，期望能減必要的農藥使用，降低環境負擔

參考資料