

Plant Seedling Recognition & Classification

made by:

karim ayman khalil 20104538

abdelrahman ahmed mohamed bahgat 20104872

For DR. SHADY Zahran and eng. mohammed moheeb

Date: december 2023

**Executive Summary**

**Project Overview**

The Project focuses on Plant Seedling Recognition & Classification using Image processing methods and various deep learning algorithms.

**Key Findings**

Image Processing such as Changing Color Formats, Blurring and Applying Masks

Implementation of CNN with six convolutional layers and three fully connected layers

**Conclusion**

Plant Recognition and Classification uses effective deep learning algorithms and image preprocessing to effectively represent plants and its species

Table Of Contents

[Introduction 4](#_Toc154310881)

[Background and Context 4](#_Toc154310882)

[Objectives 5](#_Toc154310883)

[Scope 5](#_Toc154310884)

[Report Structure 5](#_Toc154310885)

[Literature Review: 5](#_Toc154310886)

[• Existing Work: 5](#_Toc154310887)

[• Gap Analysis: 6](#_Toc154310888)

[Methodology: 6](#_Toc154310889)

[Design Process: 6](#_Toc154310890)

[Data Acquisition: 8](#_Toc154310891)

[Testing Methods: 10](#_Toc154310892)

[Technical Details: 11](#_Toc154310893)

[System Architecture: 11](#_Toc154310894)

[Algorithm Explanation: 16](#_Toc154310895)

[Results and Discussion 18](#_Toc154310896)

[Analysis of Results : 18](#_Toc154310897)

[Performance Evaluation: 20](#_Toc154310898)

[Conclusion 21](#_Toc154310899)

[Summary of Work: 21](#_Toc154310900)

[Achievements vs. Objectives: 21](#_Toc154310901)

[Future Directions: 21](#_Toc154310902)

[References: 21](#_Toc154310903)

## Introduction

### Background and Context

Plant Recognition and Classification was a competition on Kaggle that ended four years ago , inspiring us to solve it

### Objectives

* Use Image Processing to detect plants and differentiate between grass and plant
* Use CNN Algorithm to Detect each plant’s species
* Save the Testing images in a CSV with its predicted species

### Scope

Provided with a training set and a test set of images of plant seedlings at various stages of grown. Each image has a filename that is its unique id. The dataset comprises 12 plant species.

Also Provided with a spare validation and testing NumPy dataset.

### Report Structure

The report comprises sections detailing the project's background, methodology, technical details, results, and conclusions.

## Literature Review:

• Existing Work:

Kaggle has a lot of collaboration and submissions of the challenge, most projects do not use filters or change the image formats , or use more complex algorithms like VGG

• Gap Analysis:

* Improvements:

1. Applying Filters and Modifying color format
2. Using multiple layers in CNN model

## Methodology:

### Design Process:

* Algorithm Design:

Image Preprocessing:

Resizing: The images are resized to a standard size (70x70 pixels) using OpenCV that uses bilinear interpolation .

Bilinear interpolation estimates pixel values using a weighted average of the surrounding pixels.

It is quicker to load and execute in a CNN model

Gaussian Blur: A Gaussian blur is applied to the images to reduce noise and enhance features.

A non-linear spatial filtering technique utilizing a 2D Gaussian kernel to replace each pixel's intensity with a weighted average of its surrounding neighbors. This reduces high-frequency noise while preserving edges and overall image structure.

Color Space Conversion: Images are converted from the RGB color space to the HSV color space to facilitate color-based segmentation. Since it is easier to detect the green color ranges

Transformation of images from the RGB color space, where color representation relies on additive mixtures of red, green, and blue primary colors, to the HSV color space. This facilitates segmentation based on hue (angular representation of color), saturation (color intensity), and value (brightness).

Mask Creation:

Generation of a binary image (often black and white) where white pixels represent regions identified as green based on pre-defined thresholds in the HSV color space. This mask acts as a filter to isolate plant regions from the background.

A binary mask is created (usually black and white) where white pixels represent regions identified as green based on HSV thresholds:

Morphological Operations: Morphological operations (closing) are applied to the mask for noise reduction.

Application of a morphological closing operation to the green mask using a structuring element. This fills small holes and gaps within the green regions while preserving their overall shape and size, enhancing segmentation accuracy and robustness to noise.

Closing operation fills small holes within the green regions in the mask while preserving their overall size and shape.

It involves dilation followed by erosion with a structuring element (e.g., small disc).

Boolean Mask Application: The boolean mask is applied to the original image to obtain images without background.

Element-wise multiplication of the original image with the binary mask. This removes background pixels wherever the mask is black, resulting in a segmented image containing only the identified green plant regions.

Convolutional Neural Network (CNN) Architecture:

model with six convolutional layers and three fully connected layers in the end. First two convolutional layers have 64 filters, next 128 filters and the last two layers have 256 filters. After each pair of convolution layers model have max pooling layer. Also, to reduce overfitting after each pair of convolution layers we use dropout layer (10% between convolutional layers and 50% between fully connect layers) and between each layer we use batch normalization layer.

Flatten layer is used to convert the 2D matrix data to a vector for the fully connected layers.

Dense layers with “ReLU” activation are used in the fully connected part of the network.

“Softmax “activation is applied in the output layer for multi-class classification.

* Coding Practices:

Python is used as the programming language.

Libraries and Tools: OpenCV for image processing, NumPy for numerical operations, Matplotlib for visualization, Pandas for data manipulation, and Keras for building and training the CNN and much more to be discussed in other sections.

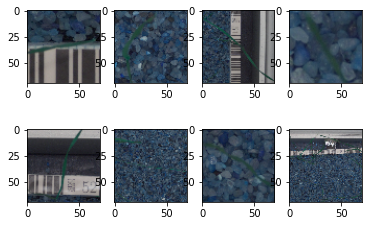
Coding Standards: Consistent indentation, meaningful variable names, and modular functions are used for code readability.

## Data Acquisition:

The dataset is collected from the '../plant-seedlings-classification/’ directory, where each subdirectory corresponds to a different plant species.

Also available was an NP dataset for training and validation , called “data.npz”.

Image files are read using OpenCV, and their labels are extracted from the directory structure.



All Images are then manipulated , such that each image is Blurred, Converted to HSV ,then applied with the two masks, “normal and Boolean” .

A collage of images of different colors

Description automatically generated

All pixel values are then normalized from [0-255] to [0-1], (RGB color-space encode colors with numbers [0...255]). CNN will be faster train if we use [0...1] input) by dividing each pixel value by 255

All labels are then encoded as a class array, Labels are 12 string names, so we could create classes array with this names, for example ['Black-grass' 'Charlock' 'Cleavers' 'Common Chickweed' 'Common wheat' 'Fat Hen' 'Loose Silky-bent' 'Maize' 'Scentless Mayweed' 'Shepherds Purse' 'Small-flowered Cranesbill' 'Sugar beet'] and encode every label by position in this array. For example 'Charlock' -> [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0].

A graph of a number of different types of food

Description automatically generated with medium confidence

## Testing Methods:

1. Model Evaluation on Training and Testing Data:

The code contains the following lines for evaluating the trained model on both the training and testing datasets:

print(model.evaluate(trainX, trainY))

print(model.evaluate(testX, testY))

model.evaluate(trainX, trainY) calculates the loss and accuracy on the training dataset.

model.evaluate(testX, testY) calculates the loss and accuracy on the testing dataset.

2. Confusion Matrix and Performance Metrics:

The code computes a confusion matrix using scikit-learn's confusion\_matrix function. It also calculates precision and accuracy using custom functions calculate\_precision and calculate\_accuracy. These metrics are then printed:

precision = calculate\_precision(trueY, predYClasses)

accuracy = calculate\_accuracy(trueY, predYClasses)

print(f"Precision: {precision:.4f}")

print(f"Accuracy: {accuracy:.4f}")

3. Plotting Confusion Matrix:

The code generates and displays a confusion matrix using a custom function plot\_confusion\_matrix:

plot\_confusion\_matrix(confusionMTX, classes = le.classes\_)

4. Visualizing Actual vs. Predicted Results:

The code includes a function plot\_actual\_vs\_predicted\_from\_data for visualizing a random set of actual vs. predicted results on the testing dataset:

plot\_actual\_vs\_predicted\_from\_data(d, model, le)

5. Saving Predictions to CSV:

Finally, the code makes predictions on the test set and saves the results to a CSV file named "res.csv":

pred = model.predict(clearTestImg)

predNum = np.argmax(pred, axis=1)

predStr = le.classes\_[predNum]

res = {'file': testId, 'species': predStr}

res = pd.DataFrame(res)

res.to\_csv("res.csv", index=False)

# Technical Details:

## System Architecture:

1. Image Manipulation:

The image manipulation section focuses on preprocessing the images before feeding them into the neural network. It includes the following steps:

Resizing Images:

trainImg.append(cv2.resize(cv2.imread(img), (ScaleTo, ScaleTo)))

* Reads an image using OpenCV.
* Resizes the image to the specified dimensions (ScaleTo, ScaleTo).

Gaussian Blur and Color Thresholding:

blurImg = cv2.GaussianBlur(img, (5, 5), 0)

hsvImg = cv2.cvtColor(blurImg, cv2.COLOR\_BGR2HSV)

* Applies Gaussian blur to reduce noise.
* Converts the blurred image to HSV color space.
* Performs color thresholding to create a mask for extracting specific color ranges.

Background Removal:

clear[bMask] = img[bMask]

* Creates a boolean mask based on the color thresholding.
* Applies the boolean mask to retain only the object of interest, removing the background.

2. Image Randomizer:

The image randomizer is part of the data augmentation process to increase the diversity of the training dataset. It uses Keras's ImageDataGenerator:

datagen = ImageDataGenerator(

rotation\_range=180,

zoom\_range=0.1,

width\_shift\_range=0.1,

height\_shift\_range=0.1,

horizontal\_flip=True,

vertical\_flip=True

)

datagen.fit(trainX)

* Rotation, Zoom, and Shift:
* Randomly rotates, zooms, and shifts images horizontally and vertically.
* Horizontal and Vertical Flip:
* Randomly flips images horizontally and vertically.

Fit Generator:

* Fits the data generator to the training data.

3. Label Encoding:

Label encoding is performed to convert categorical labels into numerical format for model training:

le = preprocessing.LabelEncoder()

encodeTrainLabels = le.transform(trainLabel['Label'])

clearTrainLabel = to\_categorical(encodeTrainLabels)

Label Encoding:

* Uses scikit-learn's LabelEncoder to convert class labels into numerical format.
* Transforms the encoded labels into one-hot encoded format using to\_categorical.

4. CNN Model Fit:

The code builds and trains a Convolutional Neural Network (CNN) model using Keras:

model = Sequential()

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

hist = model.fit\_generator(datagen.flow(trainX, trainY, batch\_size=75),

# epochs=35, validation\_data=(testX, testY),

# steps\_per\_epoch=math.ceil(trainX.shape[0] / 75), callbacks=callbacks\_list)

Model Architecture:

* Defines a sequential model with convolutional layers, max-pooling layers, batch normalization, dropout, and dense layers.

Compilation:

* Compiles the model with categorical crossentropy loss and the Adam optimizer.
* Adam (short for Adaptive Moment Estimation) is an optimization algorithm that combines the advantages of two other popular optimization algorithms: RMSprop (Root Mean Square Propagation) and Momentum. It is widely used in training neural networks and is known for its efficiency and robustness.
* Categorical Crossentropy is a loss function commonly used for multi-class classification problems, where each input sample belongs to one of several classes. It measures the dissimilarity between the true distribution and the predicted distribution of class probabilities.

Model Fitting:

Fits the model to the training data using the data generator.

* Callbacks such as learning rate reduction and model checkpoints can be added but are currently commented out.

5. Model Testing:

The code evaluates the trained model on both the training and testing datasets:

print(model.evaluate(trainX, trainY))

print(model.evaluate(testX, testY))

Model Evaluation:

* Computes and prints the loss and accuracy on both the training and testing datasets.

6. Prediction on Unseen Data:

The code makes predictions on a set of unseen test images and saves the results to a CSV file:

pred = model.predict(clearTestImg)

predNum = np.argmax(pred, axis=1)

predStr = le.classes\_[predNum]

res = {'file': testId, 'species': predStr}

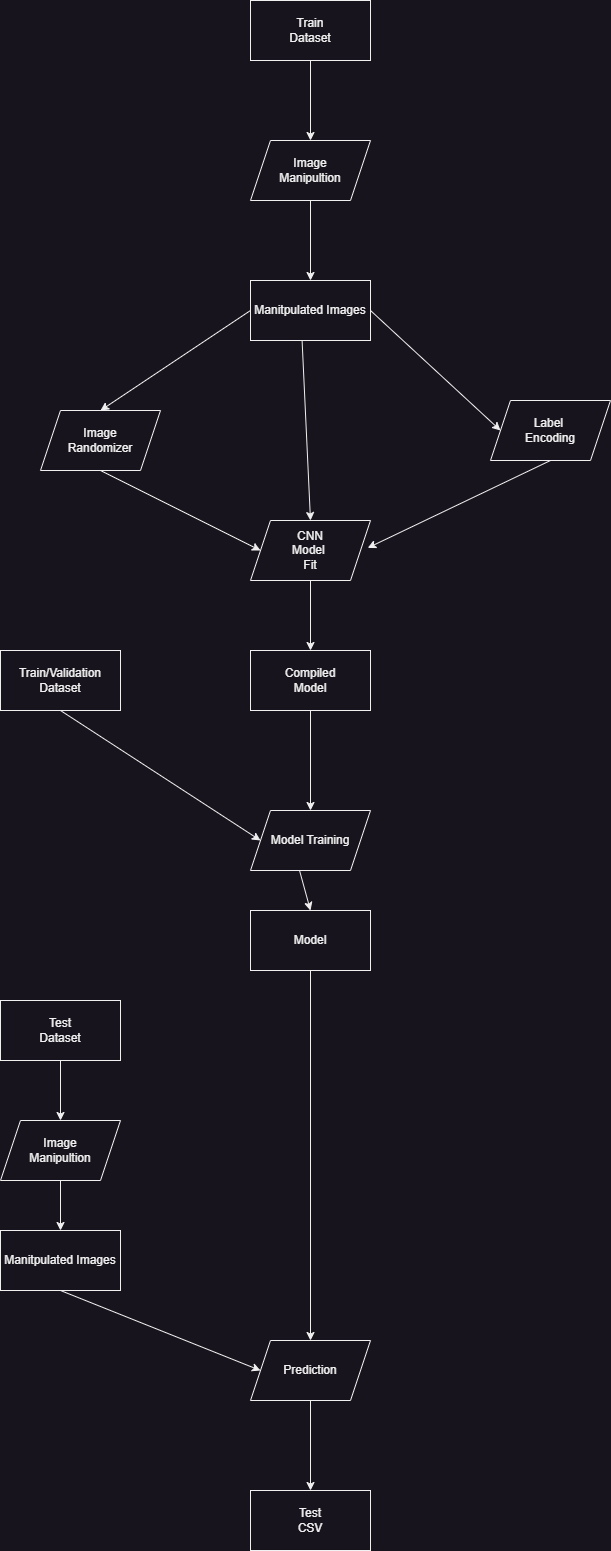
res = pd.DataFrame(res)

res.to\_csv("res.csv", index=False)

Model Prediction:

* Uses the trained model to predict the classes for the test images.
* Converts numerical predictions back to class labels.
* CSV Result File:
* Creates a DataFrame with image file names and corresponding predicted species.
* Saves the DataFrame to a CSV file named "res.csv".

The system architecture is explained in the following flowchart:



## Algorithm Explanation:

* Image Preprocessing:

A screenshot of a computer program

Description automatically generated

Gaussian Blur:

Given a point (x, y) in the resized image, the algorithm calculates the pixel value by considering the four nearest pixels (top-left, top-right, bottom-left, and bottom-right) in the original image.

The formula for bilinear interpolation is as follows:

P(x, y) = (1 - α)(1 - β) \* I(x1, y1) + α(1 - β) \* I(x2, y1) + (1 - α)β \* I(x1, y2) + αβ \* I(x2, y2)

Color Space Conversion:

for each pixel (i, j) in the image:

New intensity = ΣΣ [I(m, n) \* G(i - m, j - n)] / ΣΣ G(i - m, j - n)

where I(m, n) is the original intensity at pixel (m, n), G(i - m, j - n) is the Gaussian kernel value at offset (i - m, j - n) from (i, j), and ΣΣ represents summation over all neighboring pixels.

Mask Creation: A binary mask is created (usually black and white) where white pixels

represent regions identified as green based on HSV thresholds:

Mask(i, j) = { 1 if H\_min < H(i, j) < H\_max and S\_min < S(i, j) < S\_max else 0 }

H\_min and H\_max define the acceptable hue range for green, S\_min and S\_max define

the saturation range.

Morphological Operations: involves dilation followed by erosion with a structuring

element (e.g., small disc).

Mathematically, closing of mask M by structuring element S:

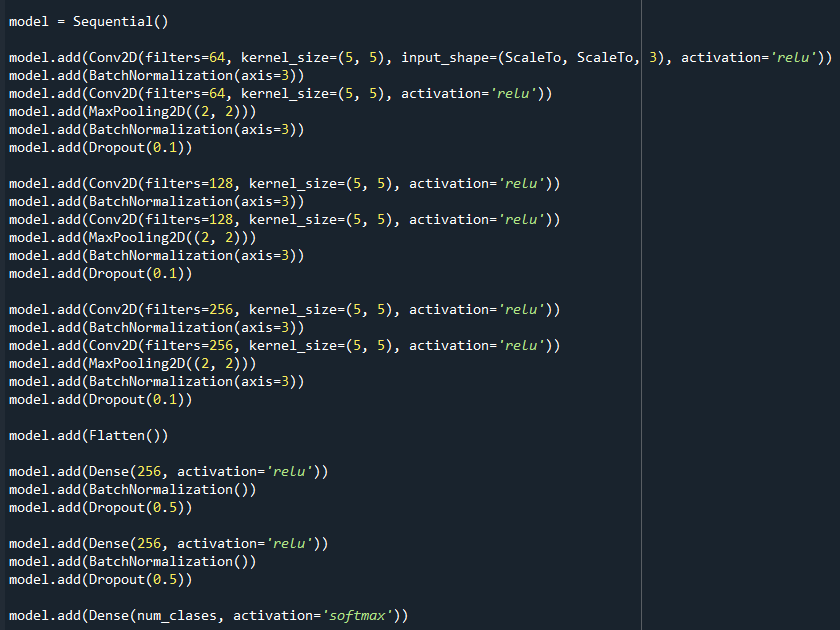
M\_closed = erosion(dilation(M, S), S)

Boolean Mask Application: The final segmentation mask is multiplied element-wise with the original image to remove background pixels and retain only green plant regions:

Segmented image = Original image . Mask\*

.\* denotes element-wise multiplication.

* CNN Architecture:



Convolutional Layers:

1. Conv1: 64 filters, each with a 5x5 kernel, ReLU activation.

ReLU (Rectified Linear Unit) is an activation function commonly used in neural networks, especially in convolutional neural networks (CNNs). It introduces non-linearity to the model by allowing the positive values to pass through while setting all negative values to zero.

Batch Normalization: Normalizes the output of the previous layer (helps in training stability).

1. Conv2: 64 filters, each with a 5x5 kernel, ReLU activation.

Max Pooling: 2x2 max-pooling to downsample the spatial dimensions.

Batch Normalization: Normalizes the output of the max-pooling layer.

Dropout: Regularization technique with a dropout rate of 0.1 to prevent overfitting.

1. Convolutional Layers (Repeat):

Similar structure to the first set, but with increased filters (128 filters).

The second set further captures complex features from the downsampled representations.

1. Convolutional Layers (Repeat):

Similar structure, now with 256 filters in each convolutional layer.

Another set to capture more abstract features.

1. Flatten Layer:

Flattens the output from the last convolutional layer into a 1D vector.

1. Dense (Fully Connected) Layers:

Dense1: 256 neurons with ReLU activation.

Batch Normalization: Normalizes the output of the dense layer.

Dropout: Regularization with a dropout rate of 0.5.

1. Dense Layers (Repeat):

Similar structure to the first dense set, providing further abstraction.

1. Output Layer:

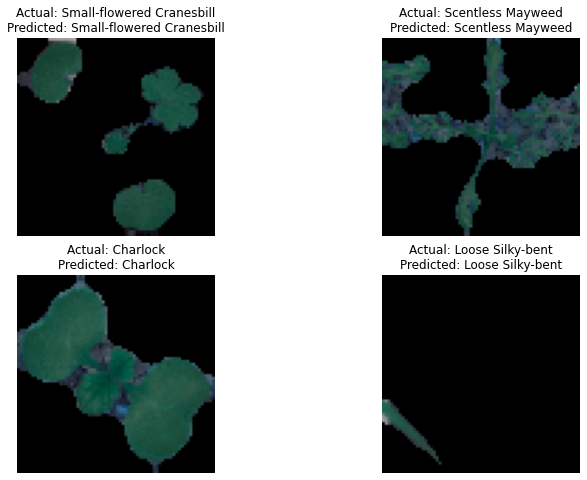
Dense (Output): Number of neurons equal to the number of classes (num\_classes) with softmax activation for multi-class classification.

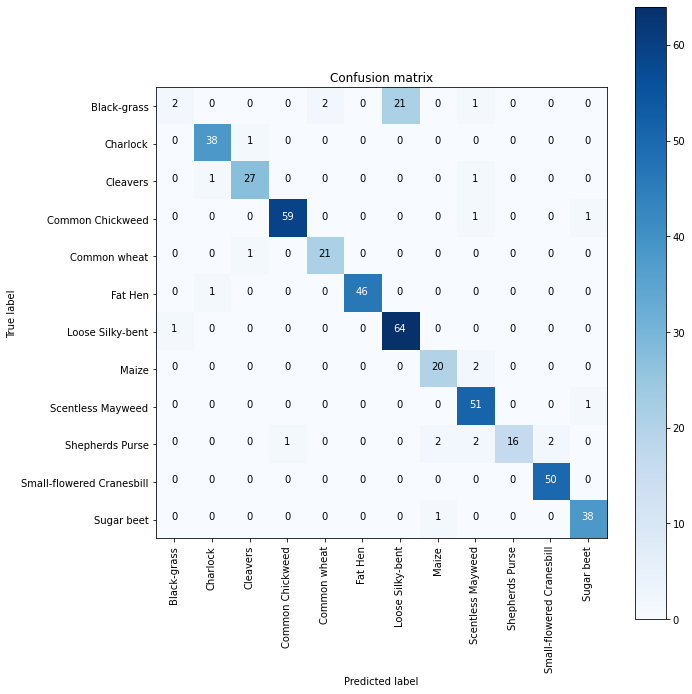
Softmax is an activation function used in the output layer of a neural network for multi-class classification problems. It takes a vector of arbitrary real-valued scores (logits) and converts them into probabilities that sum to 1. Each element of the output vector represents the probability of the corresponding class.

# Results and Discussion

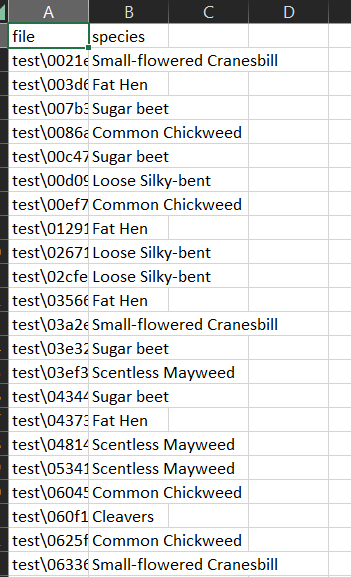
## Analysis of Results :

The following figure shows that the model correctly predicts the images’ species



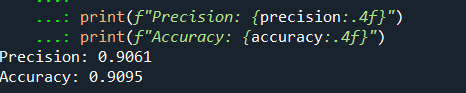
And the following confusion matrix explains the correctness of the model, there are some outliers especially in the Black-grass species

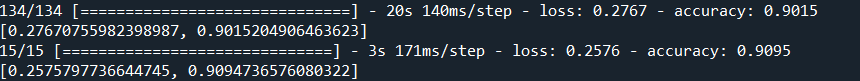
And the csv is correctly displayed and predicts the images’ species as shown below:



## Performance Evaluation:

The model shows a promising 0.9 accuracy and precision in both the evaluation and accuracy & precision metrics as shown below:





Also as shown above , the model predicts images correctly and efficiently .

Speed was an issue at the start , because the model took 1.5+ hours to be fitted, until we used checkpoints to save the best fit , so the model is used immediately.

# Conclusion

Summary of Work:

The model effectively identifies the plant seedlings and predicts the species in 0.9 accuracy , the cnn model was the biggest problem since it consists of the most layers , yet was the best fit for our code .

Achievements vs. Objectives:

All objectives were met

1. Model Outlines the plant seedlings
2. Model predicts the species
3. Model saves the results in CSV

## Future Directions:

Our future directions sees that this model could identify plants as a whole not the seedlings only , and for it to predicts its lifespan as well.

# References:

1. <https://www.kaggle.com/competitions/plant-seedlings-classification>
2. <https://www.kaggle.com/datasets/nikkonst/plantrecomodels>

Video Link:

https://drive.google.com/drive/folders/1HIkHQ1oQpnGR\_\_gMYHV6jP7SNptB373K?usp=drive\_link