Plant Seedlings Classification

Plant Species Identification using CNN

7th August 2025

Contents / Agenda

- Data Overview
- Exploratory Data Analysis
- Data Pre-processing
- Basic Convolutional Neural Network
- VGG-16 Model
- VGG-16 with FFNN
- VGG-16 with Data Augmentation
- Model Performance Comparison and Final Model Selection

Executive Summary

Actionable Insights:

Model Performance

The model achieved a test accuracy of around 72%. While this is a good starting point, there is room for improvement, especially for certain plant species with lower precision and recall (as seen in the classification report). This indicates that the model might struggle to correctly identify some plant types, which could be due to factors like data imbalance, visual similarity between certain species, or limited data for those classes.

Class Imbalance

The count plot showed that the dataset is imbalanced, with some classes having significantly more images than others. This imbalance likely affects the model's performance, potentially leading to bias towards the majority classes and poorer performance on minority classes. The confusion matrix and classification report will highlight which classes are being misclassified the most.

Misclassifications

Analyze the confusion matrix to identify which plant species are most often confused with each other. This insight is crucial for understanding the model's limitations and where focused efforts are needed. For example, if Black-grass is often misclassified as Loose Silky-bent, it suggests visual similarities or insufficient distinguishing features in the training data for these two classes.

Data Augmentation Effectiveness

The use of data augmentation in Model 2 showed a slight improvement in overall accuracy. This indicates that augmenting the dataset is beneficial for this task, helping the model generalize better to unseen images and potentially mitigating some of the issues related to limited data.

Executive Summary

Business Recommendations:

Deploy the Model for Early Weed Detection:

The developed model can be deployed in agricultural settings for early detection and classification of plant seedlings, particularly weeds. Early identification allows for targeted intervention, reducing the need for broad-spectrum herbicides and potentially leading to more sustainable farming practices.

Focus on Improving Data for Low-Performing Classes:

To improve the model's overall accuracy and reliability, focus on collecting more diverse images for the plant species that the model struggles with the most. This could involve capturing images under different lighting conditions, angles, and growth stages.

Implement Active Learning:

Introduce an active learning system where the model flags uncertain predictions for human review. This reviewed data can then be used to retrain and improve the model iteratively, especially for challenging cases or underrepresented classes.

Develop a User-Friendly Application:

Create a mobile or web application that allows farmers or agricultural workers to easily capture images of seedlings and get instant classification results. This application could also provide information about the identified plant and recommended actions (e.g., if it's a weed, suggest appropriate control methods).

Executive Summary

Business Recommendations:

Integrate with Automated Systems
For larger scale operations, integrate the model with automated systems like robotic weeding machines or precision sprayers. The model's predictions can guide these systems to accurately identify and manage plants, reducing manual labor and increasing efficiency.

Explore Transfer Learning
Consider exploring transfer learning by using pre-trained models on large image datasets (like ImageNet) and fine-tuning them on your plant seedling dataset. This can significantly improve performance, especially with limited data, as the model leverages features learned from a vast number of images.

Monitor Model Performance in Real-World Scenarios
Once deployed, continuously monitor the model's performance in real-world agricultural settings. Collect feedback from users and gather new data to identify areas for improvement and ensure the model remains accurate and relevant over time. By implementing these recommendations, the agricultural industry can leverage this computer vision model to significantly improve efficiency, reduce costs, and move towards more sustainable practices in plant seedling management.

Business Problem Overview and Solution Approach

Problem Statement

In recent times, the field of agriculture has been in urgent need of modernizing, since the amount of manual work people need to put in to check if plants are growing correctly is still highly extensive. Despite several advances in agricultural technology, people working in the agricultural industry still need to have the ability to sort and recognize different plants and weeds, which takes a lot of time and effort in the long term. The potential is ripe for this trillion-dollar industry to be greatly impacted by technological innovations that cut down on the requirement for manual labor, and this is where Artificial Intelligence can actually benefit the workers in this field, as **the time and energy required to identify plant seedlings will be greatly shortened by the use of AI and Deep Learning.** The ability to do so far more efficiently and even more effectively than experienced manual labor, could lead to better crop yields, the freeing up of human involvement for higher-order agricultural decision making, and in the long term will result in more sustainable environmental practices in agriculture as well.

Business Problem Overview and Solution Approach

Objective

The aim of this project is to Build a Convolutional Neural Network to classify plant seedlings into their respective categories.

Data Overview

The Aarhus University Signal Processing group, in collaboration with the University of Southern Denmark, has recently released a dataset containing **images of unique plants belonging to 12 different species.**
The data file names are:

images.npy
Labels.csv

Due to the large volume of data, the images were converted to the images.npy file and the labels are also put into Labels.csv, so that you can work on the data/project seamlessly without having to worry about the high data volume.
The goal of the project is to create a classifier capable of determining a plant's species from an image.

Data Overview

List of Species

Black-grass
Charlock
Cleavers
Common Chickweed
Common Wheat
Fat Hen
Loose Silky-bent
Maize
Scentless Mayweed
Shepherds Purse
Small-flowered Cranesbill
Sugar beet

Exploratory Data Analysis

Based on the Exploratory Data Analysis we performed, here's a summary of the key findings:

Image Characteristics:
The dataset contains images of plant seedlings with variations in appearance (leaf shape, size, color), taken against backgrounds with soil and pebbles. There are also variations in scale and perspective, and some images contain multiple seedlings, adding complexity to the classification task.

Target Variable Distribution:
The dataset is somewhat imbalanced, with certain plant species having significantly more images than others. This class imbalance is an important factor to consider during model training, as it can lead to biased model performance.

Exploratory Data Analysis

Image Preprocessing (Resizing):

We observed the effect of resizing the images from 128x128 to 64x64 pixels, noting the reduction in detail but the benefit for computational efficiency.
In essence, the EDA revealed that the dataset presents typical challenges for image classification, including visual variability, background noise, and class imbalance, which need to be addressed during the model building and evaluation phases.

Exploratory Data Analysis

Looking at the plots of the random images with their labels, here are a few observations:

Variety in Appearance:

The images show a variety of plant seedlings belonging to different species. They differ in leaf shape, size, and color.

Background Noise:

The seedlings are shown against a background of soil and pebbles, which adds some complexity to the images and the task of isolating the plant. Scale and Perspective: The images appear to be taken from different angles and distances, leading to variations in the size and perspective of the seedlings within the frames.

Multiple Seedlings:

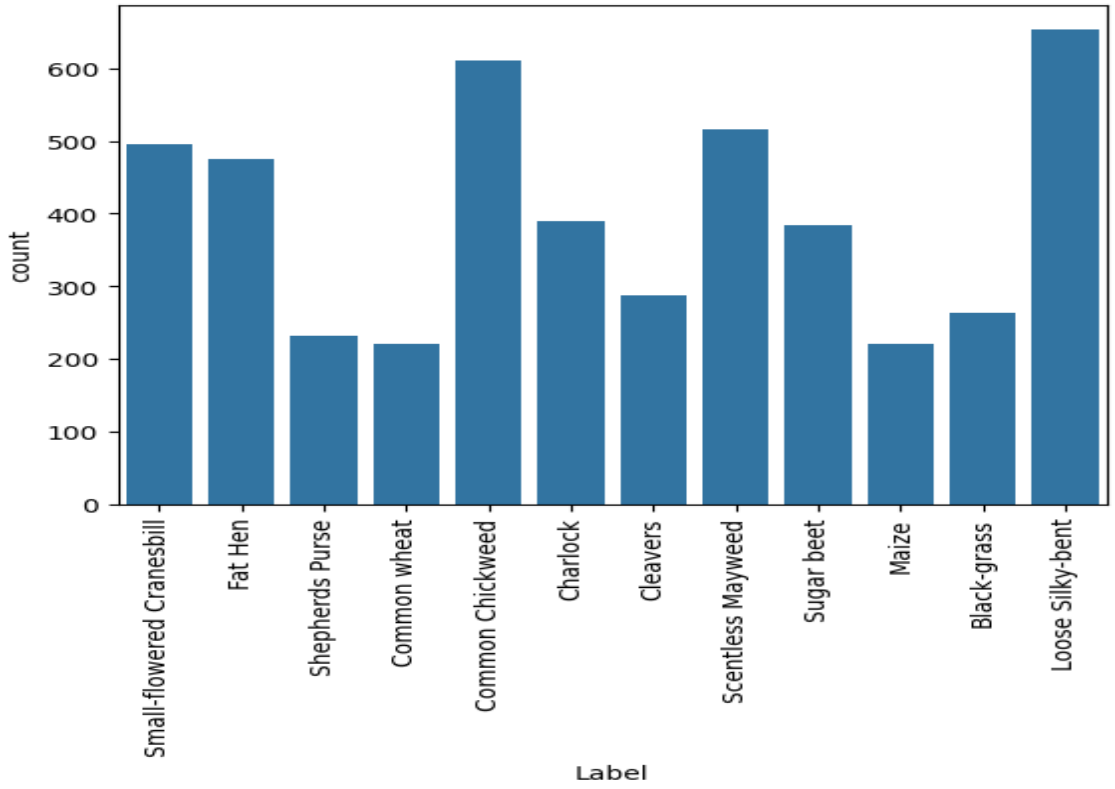
Some images contain multiple seedlings, which could make the classification task more challenging if the goal is to classify based on a single dominant seedling.

Label Clarity:

The labels are displayed correctly above each image, indicating the species of the plant shown.

Exploratory Data Analysis

Looking at the count-plot showing the distribution of the target variable (plant species labels), my observation is that the **dataset is somewhat imbalanced**.



Data Pre-processing

Data splitting

As we have less images in our dataset, we will only use 10% of our data for testing, 10% of our data for validation and 80% of our data for training.
We are using the train_test_split() function from scikit-learn. Here, we split the dataset into three parts, train,test and validation.

Encoding

This code is converting the categorical plant species labels (like 'Black-grass', 'Charlock', etc.) into a numerical format that can be used by the neural network model. Specifically, it's using LabelBinarizer to perform one-hot encoding.

One-Hot Encoding:

One-hot encoding is a common technique for handling categorical data in machine learning, especially for classification tasks. It converts each categorical label into a binary vector where only one element is 'hot' (set to 1) and the rest are 'cold' (set to 0). The position of the '1' in the vector corresponds to the specific category.

Data Pre-processing

Why One-Hot Encoding?

Neural networks typically require numerical input. Using one-hot encoding avoids assigning arbitrary numerical relationships between categories that don't exist (e.g., if you simply assigned numbers like 0, 1, 2, etc., the model might incorrectly infer that category 2 is 'greater than' category 1).

LabelBinarizer:

LabelBinarizer is a convenient tool in scikit-learn that combines the steps of converting labels to numerical format and then applying one-hot encoding. fit_transform is used on the training data to learn the unique classes and transform the labels, while transform is used on the validation and test data using the same mapping learned from the training data.
In summary, this code is preparing the target variable into a suitable binary vector format for training the classification model.

Normalization

Since the **image pixel values range from 0-255**, our method of normalization here will be **scaling** - we shall **divide all the pixel values by 255 to standardize the images to have values between 0-1.**

Basic Convolutional Neural Network

We are using two models for training, one is CNN and the other is using data

augmentation with CNN.

Model1

Sequential Model:
Model1 is a sequential model, which means layers are stacked in a linear fashion. This is a common and straightforward architecture for many image classification tasks.

Convolutional Layers (Conv2D): The model starts with three convolutional layers. These layers are the core of a CNN and are responsible for learning spatial hierarchies of features from the input images.

The first Conv2D layer has 128 filters, the second has 64, and the third has 32. The decreasing number of filters in deeper layers is a common pattern, as the network learns more abstract features.
The kernel size is (3, 3) in all convolutional layers, which is a standard choice for capturing local patterns.
activation='relu' is used, which is a popular activation function that introduces non-linearity into the model.
padding="same" ensures that the output feature maps have the same spatial dimensions as the input, preventing the reduction of spatial information at the borders.

Basic Convolutional Neural Network

Model1

Pooling Layers (MaxPooling2D): After each convolutional layer, a MaxPooling2D layer is used. These layers reduce the spatial dimensions of the feature maps, which helps in reducing the number of parameters and computational cost, and also makes the model more robust to small shifts and distortions in the input image.

A pool size of (2, 2) is used, which means the maximum value is taken from each 2x2 block of the feature map.
padding='same' is also used here.

Flatten Layer: The Flatten layer converts the 2D feature maps from the convolutional and pooling layers into a 1D vector. This is necessary to connect the convolutional part of the network to the fully connected dense layers.

Dense Layers: The flattened output is fed into two fully connected Dense layers.

The first Dense layer has 16 neurons and uses the 'relu' activation function.
The second Dense layer has 12 neurons, which corresponds to the number of output classes (plant species), and uses the 'softmax' activation function. The softmax layer outputs a probability distribution over the classes.

Basic Convolutional Neural Network

Model1

Dropout Layer: A Dropout layer with a rate of 0.3 is included after the first dense layer. Dropout is a regularization technique that randomly sets a fraction of input units to zero during training, which helps prevent overfitting.

Optimizer and Loss Function:

The Adam optimizer is used, which is a popular and effective optimization algorithm. loss='categorical_crossentropy' is the appropriate loss function for multi-class classification problems with one-hot encoded labels.

Metrics: metrics=['accuracy'] is used to monitor the accuracy of the model during training and evaluation.

Total Parameters: The model1.summary() shows that the model has a total of 128,828 parameters. This is a relatively small number of parameters, which is good for preventing overfitting, especially with a dataset of this size.

CNN with Data Augmentation

Model2:

Sequential Model:

Like model1, model2 is also a sequential model.

Convolutional Layers (Conv2D):

model2 has two convolutional layers, fewer than model1.

- The first Conv2D layer has 64 filters (compared to 128 in model1).
- The second Conv2D layer has 32 filters (same as the third in model1).
- The kernel size, activation function ('relu'), and padding ("same") are the same as in model1.

Pooling Layers (MaxPooling2D):

model2 also uses MaxPooling2D layers after each convolutional layer with the same configuration ((2, 2) pool size, 'same' padding).

Batch Normalization (BatchNormalization):

A significant addition in model2 is the BatchNormalization layer after the second pooling layer. Batch normalization helps in stabilizing the learning process and can lead to faster convergence and better performance by normalizing the activations of the previous layer.

CNN with Data Augmentation

Flatten Layer:
The Flatten layer serves the same purpose as in model1.
Dense Layers:
model2 has two Dense layers with the same number of neurons (16 and 12) and activation functions ('relu' and 'softmax') as in model1.
Dropout Layer:
The Dropout layer with a rate of 0.3 is also present in the same position as in model1.
Optimizer and Loss Function:
The Adam optimizer and categorical_crossentropy loss function are used, similar to model1.
Metrics: metrics=['accuracy'] is used.
Total Parameters: The model2.summary() shows a total of 151,676 parameters. This is slightly more than model1, primarily due to the addition of the Batch Normalization layer.
In summary, model2 is a modified version of model1. It has fewer convolutional layers but introduces a BatchNormalization layer, which is a common technique to improve the training of deep neural networks. The addition of Batch Normalization and the change in the number of filters in the initial convolutional layer are the key architectural differences. These modifications, along with the use of data augmentation during training (as seen in the fitting code), are likely aimed at improving the model's performance and generalization ability.

Model Performance Comparison and Final Model Selection

Comparing the evaluation results of Model 1 and Model 2 on both the train and test datasets reveals some key differences:

Model1:

Showed a significant difference between train accuracy (83.26%) and test accuracy (71.37%), indicating clear overfitting. The model learned the training data very well but did not generalize as effectively to unseen data. Model 2: Showed a smaller gap between train accuracy (79.80%) and test accuracy (71.79%). While still showing some overfitting, the difference is less pronounced compared to Model 1.

Model2:

Model2 incorporated data augmentation and Batch Normalization, appears to have improved generalization compared to Model 1. Even though Model 2's training accuracy is slightly lower than Model 1's, its test accuracy is marginally higher, suggesting that the regularization techniques helped the model perform better on unseen data and reduced the severity of overfitting. This aligns with the objective of building a more robust model for classifying plant seedlings.

Deployment

Files loaded on Hugging face

- https://huggingface.co/spaces/AmanAliSyed/Species-plants/commit/42ee9cc4ec850a7085115f5606c7b8b66fa6bdf0
- Web based streamlet app running on the Hugging face
- https://huggingface.co/spaces/AmanAliSyed/Species-plants