Business Problem

An Agritech firm looking for automated solutions which can detect and classify plant seedlings in images according to their species. This capability is critical for biodiversity conservation, effective crop monitoring, and early detection of invasive species. By automating the process of identification the ultimate goal is to have a model that supports sustainable farming practices and environmental monitoring initiatives, ensuring timely and precise intervention where needed.

Data Collection

I have found the dataset from Kaggle which contains one folder named the train which has 12 subfolders each folder defines a class name and contain multiple images of the particular class.

Objective

- Enhanced Accuracy- The model would help reduce manual work and time needed to identify seedlings.
- Efficiency Increase- the precision and uniformity at which various species would be classified.
- Timely Actions are Possible- Prompt interventions in farming or environmental conservation become possible through this proposal.

Importing Libraries and Important Modules

In [29]:
```
# External Libraries / Packages
import tensorflow as tf
import matplotlib.pyplot as plt


# TensorFlow and Keras Modules
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers, models
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout


# Built-in Python Modules
import os
import shutil
import random
```

Data Set Splitting and Creating Directory

From Kaggle source of dataset I am using a train folder which contain 12 folders for each class I am using function to split my data in 2 more directories one for validation which contain 20% from the train data and test directory which contain 10% which will be use in the very last step when I am going to do the final model assesment

In [2]:
```
# function for splitting data into test and validation set from training dataset
def create_train_val_test_split(train_dir, val_dir, test_dir, val_split=0.2, test_split=0.1):
    if not os.path.exists(val_dir):
        os.makedirs(val_dir)
    if not os.path.exists(test_dir):
        os.makedirs(test_dir)

    class_names = os.listdir(train_dir)

    for class_name in class_names:
        class_train_dir = os.path.join(train_dir, class_name)
        class_val_dir = os.path.join(val_dir, class_name)
        class_test_dir = os.path.join(test_dir, class_name)

        if not os.path.exists(class_val_dir):
            os.makedirs(class_val_dir)
        if not os.path.exists(class_test_dir):
            os.makedirs(class_test_dir)

        images = os.listdir(class_train_dir)
        num_val_images = int(len(images) * val_split)
        num_test_images = int(len(images) * test_split)

        # Randomly select images for validation and test set
        val_images = random.sample(images, num_val_images)
        remaining_images = list(set(images) - set(val_images))
        test_images = random.sample(remaining_images, num_test_images)

        # Move the randomly selected images for the validation set
        for image in val_images:
            src_path = os.path.join(class_train_dir, image)
            dst_path = os.path.join(class_val_dir, image)
            shutil.move(src_path, dst_path)
        print(f"Moved {num_val_images} images from {class_train_dir} to {class_val_dir}")

        # Move the randomly selected images for the test set
        for image in test_images:
            src_path = os.path.join(class_train_dir, image)
            dst_path = os.path.join(class_test_dir, image)
            shutil.move(src_path, dst_path)
        print(f"Moved {num_test_images} images from {class_train_dir} to {class_test_dir}")


train_dir = 'train'
val_dir = 'validation'
test_dir = 'test'
create_train_val_test_split(train_dir, val_dir, test_dir, val_split=0.2, test_split=0.1)
```

```
Moved 52 images from train\Black-grass to validation\Black-grass
Moved 26 images from train\Black-grass to test\Black-grass
Moved 78 images from train\Charlock to validation\Charlock
Moved 39 images from train\Charlock to test\Charlock
Moved 57 images from train\Cleavers to validation\Cleavers
Moved 28 images from train\Cleavers to test\Cleavers
Moved 122 images from train\Common Chickweed to validation\Common Chickweed
Moved 61 images from train\Common Chickweed to test\Common Chickweed
Moved 44 images from train\Common wheat to validation\Common wheat
Moved 22 images from train\Common wheat to test\Common wheat
Moved 95 images from train\Fat Hen to validation\Fat Hen
Moved 47 images from train\Fat Hen to test\Fat Hen
Moved 130 images from train\Loose Silky-bent to validation\Loose Silky-bent
Moved 65 images from train\Loose Silky-bent to test\Loose Silky-bent
Moved 44 images from train\Maize to validation\Maize
Moved 22 images from train\Maize to test\Maize
```

```
Moved 103 images from train\Scentless Mayweed to validation\Scentless Mayweed
Moved 51 images from train\Scentless Mayweed to test\Scentless Mayweed
Moved 46 images from train\Shepherds Purse to validation\Shepherds Purse
Moved 23 images from train\Shepherds Purse to test\Shepherds Purse
Moved 99 images from train\Small-flowered Cranesbill to validation\Small-flowered Cranesbill
Moved 49 images from train\Small-flowered Cranesbill to test\Small-flowered Cranesbill
Moved 77 images from train\Sugar beet to validation\Sugar beet
Moved 38 images from train\Sugar beet to test\Sugar beet
```

Creating Training, Validation and Test set by fetching images from there respective directory in batches of 32

In [18]:
```python
# Create training dataset
training_set = tf.keras.utils.image_dataset_from_directory(
    'train',
    labels="inferred",
    label_mode="categorical",
    class_names=None,
    color_mode="rgb",
    batch_size=32,
    image_size=(128, 128),
    shuffle=True,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False
)

# Create validation dataset
validation_set = tf.keras.utils.image_dataset_from_directory(
    'validation',
    labels="inferred",
    label_mode="categorical",
    class_names=None,
    color_mode="rgb",
    batch_size=32,
    image_size=(128, 128),
    shuffle=False,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False
)
# Create test dataset
test_set = tf.keras.utils.image_dataset_from_directory(
    'test',
    labels="inferred",
    label_mode="categorical",
    class_names=None,
    color_mode="rgb",
    batch_size=32,
    image_size=(128, 128),
    shuffle=False,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False
)

# Get the number of classes to print the number of classes and images inside each directory
class_names = training_set.class_names
num_classes = len(class_names)
```

```
Found 3334 files belonging to 12 classes.
Found 947 files belonging to 12 classes.
Found 471 files belonging to 12 classes.
```

Data Visualization

Loading a batch of images from training set and just displaying 3 of them

In [19]:
```python
# Load a batch of images from the training set
for images, labels in training_set.take(1):
    # Plot the first three images
    plt.figure(figsize=(10, 10))
    for i in range(3):
        ax = plt.subplot(1, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[tf.argmax(labels[i]).numpy()])
        plt.axis("off")

# Show the plot
plt.show()
```

Base Model

This is the summary of my base model in which the configuration has been taken from Tensorflow CNN documentation.

Base Model CNN

Base Model CNN

Description	Training Accuracy	Validation Accuracy	Execution Time(seconds)	Comments
• Total Convolutional Layers: 3 • Total Pooling Layers: 3 (2 Max Pooling, 1 Global Average Pooling) • Total Dense (Fully Connected) Layers: 2 • Total Layers (excluding input layer): 8 • Optimizer: Adam • Loss Function: Categorical Crossentropy • Epoch 10	71.14323378	70.74973583	513 sec	This is the first base model which I run on my dataset without applying any preprocessing

10 Different Configuration Experiment on the Base Model

Below is the summary of all the configurations for which I have run the base model.

Initial Configuration on Base Model CNN					
Configuration	Description	Training Accuracy	Validation Accuracy	Execution Time(seconds)	Comments
Experiment # 1	Changed the Number of Filters in the Convolutional Layer From(32,64,128) to (64,128,256)	70.85413933	70.96092700958252	1704 sec	I have found that by this configuration there is no any significant change found in the validation accuracy from base model
Experiment # 2	Changed the Kernel Size of the Convolutional Layers from (3,3) to (5,5)	82.07621574	78.56388688087463	848 sec	I have found that by this configuration there is significant increase found in the validation accuracy from base model
Experiment # 3	Added a Dropout Layer After Each Convolutional Layer	72.79894948	72.86166548728943	736 sec	I have found that by this configuration there is no any significant change found in the validation accuracy from base model
Experiment # 4	Changed the Activation Function from ReLU to LeakyReLU	73.8501966	74.34002161026001	645 sec	I have found that by this configuration there is slightly change found in the validation accuracy from base model
Experiment # 5	Changed the Optimizer from Adam to RMSprop	74.32326078	75.29038786888123	543 sec	I have found that by this configuration there is change found in the validation accuracy from base model
Experiment # 6	Change the Learning Rate of the Optimizer 0.001	80.26280999	78.7750780582428	567 sec	I have found that by this configuration better result found in the validation accuracy from base model
Experiment # 7	Added Batch Normalization Layers	70.85413933	68.95459294319153	994 sec	I have found that by this configuration there is no any change found in the validation accuracy from base model
Experiment # 8	Change the Pooling Layers from MaxPooling to AveragePooling	70.85413933	69.48257684707642	686 sec	I have found that by this configuration there is decrease found in the validation accuracy from base model
Experiment # 9	Add an Additional Dense Layer	77.6084125	78.35268974304199	510 sec	I have found that by this configuration better result found in the validation accuracy from base model
Experiment # 10	Use a Different Activation Function in the Dense Layers to tanh	13.77135366	13.727560639381409	540 sec	I have found that by this configuration very low result found in the validation accuracy from base model

Further 3 more different Configuration Experiments

Below is the summary of 3 more configurations in which I have applied the preprocessing techniques on the based model and the model which has higher results in the above configurations

Further Configuration on Base Model CNN					
Configuration	Description	Training Accuracy	Validation Accuracy	Execution Time(seconds)	Comments
Experiment # 11	Applied Preprocessing , data Normalization and augmentation on Base model	52.03679204	55.33263087272644	598 sec	In this configuration I have applied preprocessing techniques on my base model and I have found that it significantly decreases the validation score
Experiment # 12	Applied Preprocessing & Changed the Kernel Size of the Convolutional Layers from (3,3) to (5,5)	66.07096195	69.16578412055969	1211 sec	In this I have found above the EXPERIMENT # 6 is performing well on my data set so I have applied preprocessing techniques and found that base this model without preprocessing is performing well
Experiment # 13	Applied Preprocessing , data Normalization and augmentation on with 30 Epoch and Learning Rate 0.001	69.72404718	70.64414024353027	1683 sec	I have applied the same above model with 30 Epoch but found no significant change

Defining and Training of (CNN) Model

Defining a model :

- 3 Convolutional Layers of (32,64,128) with kernel (3,3)
- 3 MaxPooling2D Layers (2,2) after each convolutional layer
- 3 Dropout Layers (0.25) after each max pooling layer and after the first dense layer to control overfitting
- 1 GlobalAveragePooling2D Layer
- 2 Dense Layers

Compiling model :

- Learning Rate 0.001
- loss: categorical_crossentropy
- metrics: accuracy

Training a model :

- Train set
- Validation set
- 30 Epoch

In [23]:
```
# Defining function with its configurations
def build_model(input_shape, num_classes):
    model = models.Sequential([
```

```
            layers.Input(shape=input_shape),
            layers.Conv2D(32, (3, 3), activation='relu'),
            layers.MaxPooling2D((2, 2)),
            layers.Dropout(0.25),
            layers.Conv2D(64, (3, 3), activation='relu'),
            layers.MaxPooling2D((2, 2)),
            layers.Dropout(0.25),
            layers.Conv2D(128, (3, 3), activation='relu'),
            layers.MaxPooling2D((2, 2)),
            layers.Dropout(0.25),
            layers.GlobalAveragePooling2D(),
            layers.Dense(128, activation='relu'),
            layers.Dropout(0.25),
            layers.Dense(num_classes, activation='softmax')
    ])
    return model

# Defining input shape for a model
img_height, img_width = 128, 128
input_shape = (img_height, img_width, 3)

# Creating an instance of the model
model = build_model(input_shape, num_classes)

# Model compilation
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

# Printing the Summary of the model
model.summary()

# Training model on training set
history = model.fit(
    training_set,
    validation_data=validation_set,
    epochs=30
)
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d_6 (MaxPooling2D)	(None, 63, 63, 32)	0
dropout (Dropout)	(None, 63, 63, 32)	0
conv2d_7 (Conv2D)	(None, 61, 61, 64)	18,496
max_pooling2d_7 (MaxPooling2D)	(None, 30, 30, 64)	0
dropout_1 (Dropout)	(None, 30, 30, 64)	0
conv2d_8 (Conv2D)	(None, 28, 28, 128)	73,856
max_pooling2d_8 (MaxPooling2D)	(None, 14, 14, 128)	0
dropout_2 (Dropout)	(None, 14, 14, 128)	0
global_average_pooling2d_2 (GlobalAveragePooling2D)	(None, 128)	0
dense_4 (Dense)	(None, 128)	16,512
dropout_3 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 12)	1,548

Total params: 111,308 (434.80 KB)

Trainable params: 111,308 (434.80 KB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 57s 511ms/step - accuracy: 0.1302 - loss: 6.3616 - val_accuracy: 0.3390 - val_loss: 2.1153
Epoch 2/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 52s 489ms/step - accuracy: 0.3490 - loss: 1.9396 - val_accuracy: 0.3580 - val_loss: 1.9233
Epoch 3/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 57s 545ms/step - accuracy: 0.4307 - loss: 1.6633 - val_accuracy: 0.4308 - val_loss: 1.6774
Epoch 4/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 55s 526ms/step - accuracy: 0.4715 - loss: 1.4992 - val_accuracy: 0.4203 - val_loss: 1.6498
Epoch 5/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 55s 523ms/step - accuracy: 0.5124 - loss: 1.4412 - val_accuracy: 0.5153 - val_loss: 1.3999
Epoch 6/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 50s 470ms/step - accuracy: 0.5385 - loss: 1.3192 - val_accuracy: 0.5998 - val_loss: 1.2391
Epoch 7/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 48s 458ms/step - accuracy: 0.5649 - loss: 1.2535 - val_accuracy: 0.5744 - val_loss: 1.2415
Epoch 8/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 48s 453ms/step - accuracy: 0.5810 - loss: 1.2246 - val_accuracy: 0.6199 - val_loss: 1.1666
Epoch 9/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 47s 450ms/step - accuracy: 0.6066 - loss: 1.1607 - val_accuracy: 0.6378 - val_loss: 1.0749
Epoch 10/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 49s 463ms/step - accuracy: 0.6401 - loss: 1.0502 - val_accuracy: 0.6600 - val_loss: 1.0047
Epoch 11/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 50s 477ms/step - accuracy: 0.6535 - loss: 1.0143 - val_accuracy: 0.5966 - val_loss: 1.2178
Epoch 12/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 52s 496ms/step - accuracy: 0.6300 - loss: 1.0720 - val_accuracy: 0.6220 - val_loss: 1.0334
Epoch 13/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 49s 460ms/step - accuracy: 0.6353 - loss: 1.0487 - val_accuracy: 0.6800 - val_loss: 0.9887
Epoch 14/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 49s 464ms/step - accuracy: 0.6722 - loss: 0.9507 - val_accuracy: 0.6917 - val_loss: 0.9216
Epoch 15/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 49s 461ms/step - accuracy: 0.6678 - loss: 0.9750 - val_accuracy: 0.6769 - val_loss: 0.9801
Epoch 16/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 49s 469ms/step - accuracy: 0.6869 - loss: 0.8911 - val_accuracy: 0.6441 - val_loss: 1.0512
Epoch 17/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 49s 463ms/step - accuracy: 0.7032 - loss: 0.8537 - val_accuracy: 0.7086 - val_loss: 0.8506
Epoch 18/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 51s 485ms/step - accuracy: 0.6921 - loss: 0.8854 - val_accuracy: 0.7117 - val_loss: 0.8826
Epoch 19/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 59s 556ms/step - accuracy: 0.7065 - loss: 0.8286 - val_accuracy: 0.7128 - val_loss: 0.9036
Epoch 20/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 53s 508ms/step - accuracy: 0.6978 - loss: 0.8827 - val_accuracy: 0.6895 - val_loss: 0.9549
Epoch 21/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 49s 467ms/step - accuracy: 0.7273 - loss: 0.8051 - val_accuracy: 0.7012 - val_loss: 0.8874
Epoch 22/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 56s 529ms/step - accuracy: 0.7130 - loss: 0.8029 - val_accuracy: 0.5882 - val_loss: 1.1949
Epoch 23/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 61s 582ms/step - accuracy: 0.7224 - loss: 0.8104 - val_accuracy: 0.7350 - val_loss: 0.7904
Epoch 24/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 52s 492ms/step - accuracy: 0.7392 - loss: 0.7326 - val_accuracy: 0.6927 - val_loss: 0.8966
```

```
Epoch 25/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 60s 569ms/step - accuracy: 0.7355 - loss: 0.7424 - val_accuracy: 0.7170 - val_loss: 0.8231
Epoch 26/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 71s 671ms/step - accuracy: 0.7667 - loss: 0.6927 - val_accuracy: 0.7223 - val_loss: 0.8158
Epoch 27/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 94s 893ms/step - accuracy: 0.7741 - loss: 0.6713 - val_accuracy: 0.7117 - val_loss: 0.8583
Epoch 28/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 98s 927ms/step - accuracy: 0.7588 - loss: 0.6917 - val_accuracy: 0.7635 - val_loss: 0.7142
Epoch 29/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 107s 1s/step - accuracy: 0.7754 - loss: 0.6265 - val_accuracy: 0.6969 - val_loss: 0.8561
Epoch 30/30
105/105 ━━━━━━━━━━━━━━━━━━━━ 99s 938ms/step - accuracy: 0.7508 - loss: 0.7016 - val_accuracy: 0.7804 - val_loss: 0.6638
```

Evaluating the Accuracy on Training Set

Here I am evaluating the accuracy of a model on training set to check how my model work on training set so that then I can compare it with validation accuracy to measure there is any overfitting

In [24]:
```
#Training set Accuracy
train_loss, train_acc = model.evaluate(training_set)
print('Training accuracy:', train_acc*100)
```

```
105/105 ━━━━━━━━━━━━━━━━━━━━ 36s 340ms/step - accuracy: 0.8270 - loss: 0.5083
Training accuracy: 81.49369955062866
```

Evaluating the Accuracy on Validation Set

Here I am evaluating the accuracy of a model on Validation set and here I have found that there is no signifinat difference between train and validation accuracy which means my model has not memorized the data and working well on un seen data

In [25]:
```
#Validation set Accuracy
val_loss, val_acc = model.evaluate(validation_set)
print('Validation accuracy:', val_acc*100)
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 11s 331ms/step - accuracy: 0.7082 - loss: 0.6850
Validation accuracy: 78.03590297698975
```

Model Assesment

Now we are doing prediction on our unseen test data on our train model which has a instance named as model

In [26]:
```
# Predict classes for test images
predictions = model.predict(test_set)
predicted_classes = tf.argmax(predictions, axis=1)
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 8s 444ms/step
```

Accuracy on Test Set

For evaluation I am taking true labels from my test data directory to compare it with the predicted class of the model

In [28]:
```
# Extract true labels from the test dataset
true_labels = []
for images, labels in test_set:
    true_labels.extend(tf.argmax(labels, axis=1).numpy())

true_labels = tf.constant(true_labels, dtype=tf.int64)

# Calculate accuracy
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted_classes, true_labels), tf.float64)).numpy()
print(f"Test Accuracy: {accuracy * 100}")
```

```
Test Accuracy: 79.19320594479831
```

Final Model

The final model summary which is giving better results after doing 14 different configuration

Final Model				
Description	Training Accuracy	Validation Accuracy	Test Accuracy	Execution Time (seconds)
• Total Convolutional Layers: 3 • Total Pooling Layers: 3 (2 Max Pooling, 1 Global Average Pooling) • Total Dense (Fully Connected) Layers: 2 • Total Layers (excluding input layer): 8 • Optimizer: Adam • Loss Function: Categorical Crossentropy • Epoch 30 • Learning Rate 0.001 • Dropout layer 0.25	81.49369955	78.03590298	79.19320594	1775 sec

Final Discussion

Pipeline Strength

- Decent Training Accuracy: Model has learned well as it has obtained 81% training accuracy, which means that it is able to pick out crucial characteristics and patterns from the input training samples.
- Reasonable Test Accuracy: : 78% validation accuracy implies that the model works well even with the examples that had not been seen before; this implies some level of generalization capability.
- Consistent Performance: The Test accuracy of model is 79% when it comes to test datasets hence its performance is guaranteed across all different datasets.

Limitations of the Pipeline:

- Potential Overfitting Concerns: Despite the close alignment between training and test accuracies, vigilance is needed to ensure the model's generalization capabilities are maintained as complexity increases.
- Data Representation: Variations in data quality or class distribution could impact the model's ability to generalize across diverse scenarios.

Implications of the Results:

- Practical Applicability: The model can effectively classify seedling plants, offering potential benefits in agricultural automation and research.
- Reliability in Prediction: With consistent accuracy metrics across training, validation, and test datasets, stakeholders can trust the model's predictions in real-world scenarios.