# EK_CV_Project_Low_Code_Notebook

February 24, 2025

## 1 Introduction to Computer Vision: Plant Seedlings Classification

[ ]:
```
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Change directory to where the notebook is located
%cd /content/drive/My Drive/XXX/AI/Project_Plants_Seedling_Classification/

# Convert the notebook to HTML
##!jupyter nbconvert --to html EK_CV_Project_Low_Code_Notebook.ipynb

# Download the converted HTML file
#from google.colab import files
#files.download('EK_CV_Project_Low_Code_Notebook.html')
```

### 1.1 Problem Statement

#### 1.1.1 Context

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 inolvement for higher-order agricultural decision making, and in the long term will result in more sustainable environmental practices in agriculture as well.

#### 1.1.2 Objective

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

### 1.1.3 Data Dictionary

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 dataset can be download from Olympus.
- 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.

**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

### 1.1.4 Note: Please use GPU runtime on Google Colab to execute the code faster.

## 1.2 Importing necessary libraries

[50]:
```
!pip install pandas>=2.0.0,<2.2.2 opencv-python==4.8.0.76 tensorflow==2.15.0␣
↪scikit-learn==1.2.2 seaborn==0.13.1 matplotlib==3.7.1 numpy==1.25.2 -q --user
```

```
/bin/bash: line 1: 2.2.2: No such file or directory
```

[51]:
```
# Installing the libraries with the specified version.
# uncomment and run the following line if Google Colab is being used
!pip install tensorflow==2.15.0 scikit-learn==1.2.2 seaborn==0.13.1␣
↪matplotlib==3.7.1 numpy==1.25.2 pandas==1.5.3 opencv-python==4.8.0.76 -q␣
↪--user
```

[52]:
```
# Installing the libraries with the specified version.
# uncomment and run the following lines if Jupyter Notebook is being used
```

```
#!pip install tensorflow==2.13.0 scikit-learn==1.2.2 seaborn==0.11.1␣
↪matplotlib==3.3.4 numpy==1.24.3 pandas==1.5.2 opencv-python==4.8.0.76 -q␣
↪--user
```

**Note**: *After running the above cell, kindly restart the notebook kernel and run all cells sequentially from the start again.*

[53]:
```
import os
import numpy as np                                                    ␣
↪                  # Importing numpy for Matrix Operations
import pandas as pd                                                   ␣
↪                  # Importing pandas to read CSV files
import matplotlib.pyplot as plt                                       ␣
↪                  # Importting matplotlib for Plotting and visualizing images
import math                                                           ␣
↪                  # Importing math module to perform mathematical operations
import cv2                                                            ␣
↪                  # Importing openCV for image processing
import seaborn as sns                                                 ␣
↪                  # Importing seaborn to plot graphs


# Tensorflow modules
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator  ␣
↪                  # Importing the ImageDataGenerator for data augmentation
from tensorflow.keras.models import Sequential                        ␣
↪                  # Importing the sequential module to define a sequential␣
↪model
from tensorflow.keras.layers import␣
↪Dense,Dropout,Flatten,Conv2D,MaxPooling2D,BatchNormalization # Defining all␣
↪the layers to build our CNN Model
from tensorflow.keras.optimizers import Adam,SGD                      ␣
↪                  # Importing the optimizers which can be used in our model
from sklearn import preprocessing                                     ␣
↪                  # Importing the preprocessing module to preprocess the data
from sklearn.model_selection import train_test_split                  ␣
↪                  # Importing train_test_split function to split the data␣
↪into train and test
from sklearn.metrics import confusion_matrix                          ␣
↪                  # Importing confusion_matrix to plot the confusion matrix
from sklearn.preprocessing import LabelBinarizer
# Display images using OpenCV
from google.colab.patches import cv2_imshow                           ␣
↪                  # Importing cv2_imshow from google.patches to display images
from sklearn.model_selection import train_test_split
```

```
from tensorflow.keras import backend
from keras.callbacks import ReduceLROnPlateau
import random
# Ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

### 1.3 Loading the dataset

[54]:
```
# Uncomment and run the below code if you are using google colab
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[55]:
```
# Load the image file of dataset
images = np.load('/content/drive/My Drive/_GLL#_/AI/
 ↪Project_Plants_Seedling_Classification/images.npy')      # Complete the code␣
 ↪to read the dataset

# Load the labels file of dataset
labels = pd.read_csv('/content/drive/My Drive/_GLL#_/AI/
 ↪Project_Plants_Seedling_Classification/Labels.csv')  # Complete the code to␣
 ↪read the dataset
```

### 1.4 Data Overview

#### 1.4.1 Understand the shape of the dataset

[56]:
```
# to check the shape
print("Shape of images array:", images.shape)
print("Shape of labels dataframe:", labels.shape)
```

```
Shape of images array: (4750, 128, 128, 3)
Shape of labels dataframe: (4750, 1)
```

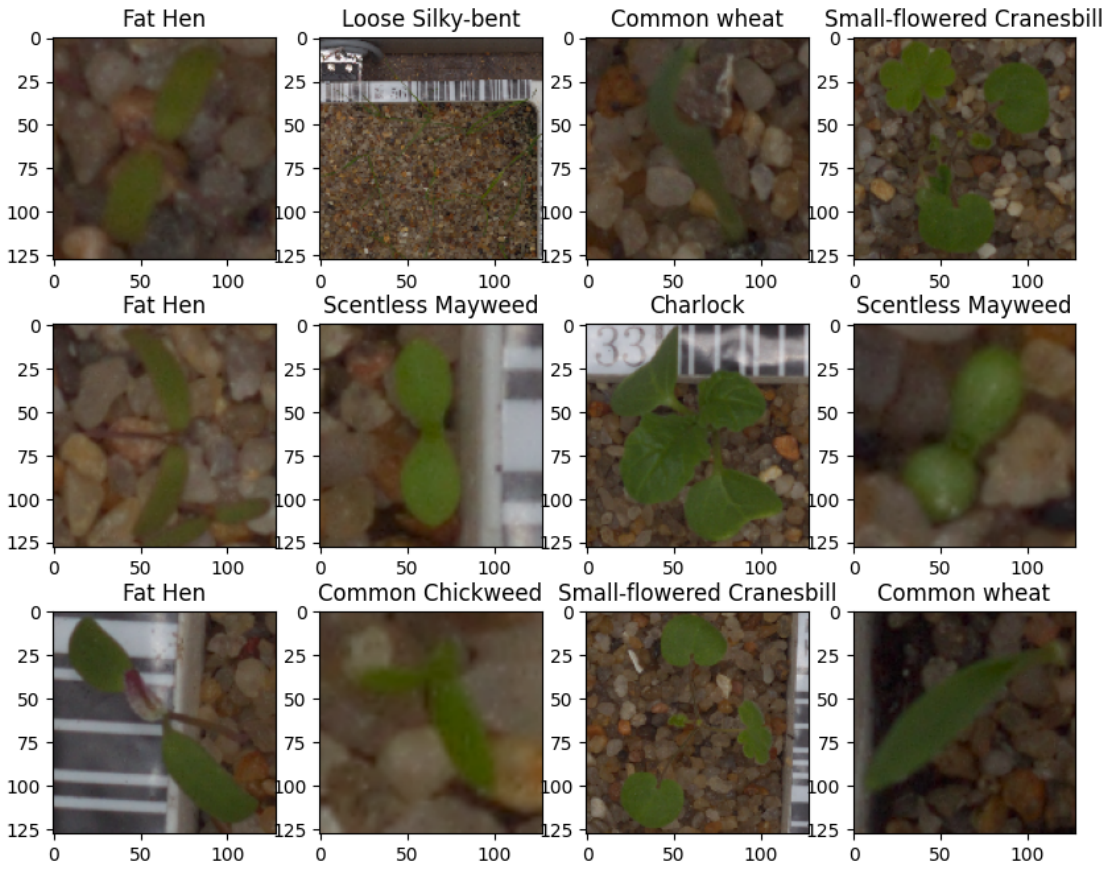
### 1.5 Exploratory Data Analysis

#### 1.5.1 Plotting random images from each of the class

[98]:
```
def plot_images(images,labels):
  num_classes=10                                                          ␣
 ↪  # Number of Classes
  categories=np.unique(labels)
  keys=dict(labels['Label'])                                              ␣
 ↪  # Obtaing the unique classes from y_train
```

```
rows = 3
↪ # Defining number of rows=3
cols = 4
↪ # Defining number of columns=4
fig = plt.figure(figsize=(10, 8))
↪ # Defining the figure size to 10x8
for i in range(cols):
    for j in range(rows):
        random_index = np.random.randint(0, len(labels))
↪ # Generating random indices from the data and plotting the images
        ax = fig.add_subplot(rows, cols, i * rows + j + 1)
↪ # Adding subplots with 3 rows and 4 columns
        ax.imshow(images[random_index, :])
↪ # Plotting the image
        ax.set_title(keys[random_index])
plt.show()
```

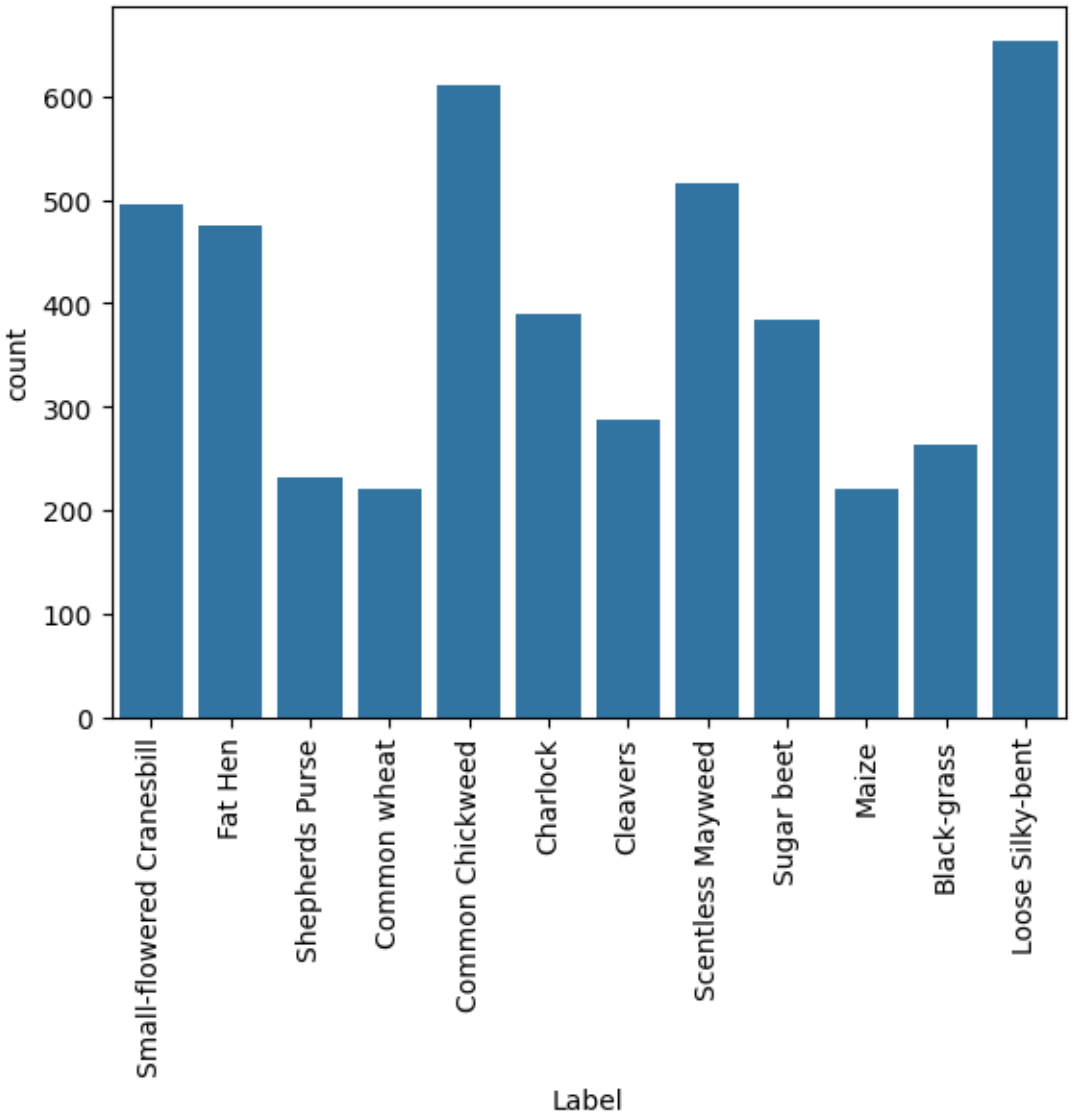
[99]:
```
plot_images(images,labels)   # Complete the code to input the images and labels
↪to the function and plot the images with their labels
```



### 1.5.2 Checking the distribution of the target variable

[100]:
```
# to check for data imbalance
sns.countplot(x=labels['Label'])  # Countplot to visualize class distribution
plt.xticks(rotation='vertical')
```

[100]:
```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
 [Text(0, 0, 'Small-flowered Cranesbill'),
  Text(1, 0, 'Fat Hen'),
  Text(2, 0, 'Shepherds Purse'),
  Text(3, 0, 'Common wheat'),
  Text(4, 0, 'Common Chickweed'),
  Text(5, 0, 'Charlock'),
  Text(6, 0, 'Cleavers'),
  Text(7, 0, 'Scentless Mayweed'),
  Text(8, 0, 'Sugar beet'),
  Text(9, 0, 'Maize'),
  Text(10, 0, 'Black-grass'),
  Text(11, 0, 'Loose Silky-bent')])
```




## 1.6 Data Pre-Processing

### 1.6.1 Converting the BGR images to RGB images.

[60]:
```
# Converting the images from BGR to RGB using cvtColor function of OpenCV
for i in range(len(images)):
  images[i] = cv2.cvtColor(images[i], cv2.COLOR_BGR2RGB)        # Complete the␣
 ↪code to convert the images from BGR to RGB
```

### 1.6.2 Resizing images

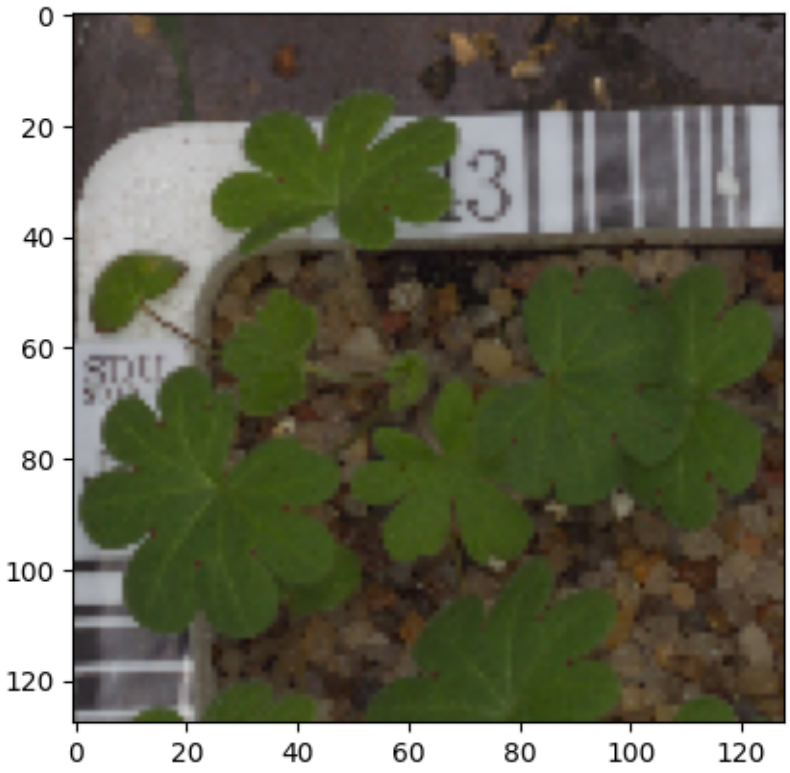
As the size of the images is large, it may be computationally expensive to train on these larger images; therefore, it is preferable to reduce the image size from 128 to 64.

[61]:
```
images_decreased=[]
height = 64                    # to define the height as 64
width =  64                    #  to define the width as 64
dimensions = (width, height)
for i in range(len(images)):
  images_decreased.append( cv2.resize(images[i], dimensions, interpolation=cv2.
 ↪INTER_LINEAR))
```

### Image before resizing

[62]:
```
plt.imshow(images[3])
```

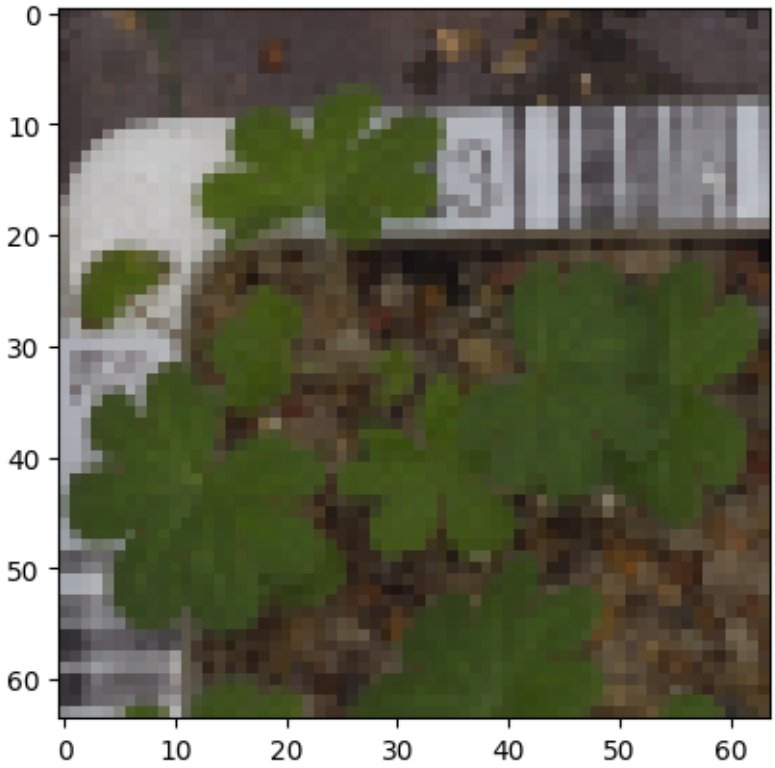
[62]:
```
<matplotlib.image.AxesImage at 0x796c4c37b490>
```




### Image after resizing

[63]:
```
plt.imshow(images_decreased[3])
```

[63]:
```
<matplotlib.image.AxesImage at 0x796c5c7e2490>
```




### 1.6.3 Data Preparation for Modeling

- 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.

[64]:
```
# Splitting dataset into train, test, and validation sets
X_temp, X_test, y_temp, y_test = train_test_split(np.
 ↪array(images_decreased),labels['Label'] , test_size=0.1,␣
 ↪random_state=42,stratify=labels)   # Complete the code to split the data␣
 ↪with test_size as 0.1
# Further splitting the temporary set into train and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_temp,y_temp , test_size=0.
 ↪1, random_state=42,stratify=y_temp)                          # Complete the␣
 ↪code to split the data with test_size as 0.1
```

[65]:
```
# to check the shape of train, validation and test data
print(X_train.shape,y_train.shape)
print(X_val.shape,y_val.shape)
print(X_test.shape,y_test.shape)
```

```
(3847, 64, 64, 3) (3847,)
(428, 64, 64, 3) (428,)
(475, 64, 64, 3) (475,)
```

### 1.6.4 Encoding the target labels

[66]:
```
# Convert labels from names to one hot vectors.
#  already used encoding methods like onehotencoder and labelencoder earlier so␣
 ↪now  will be using a new encoding method called labelBinarizer.
# Labelbinarizer works similar to onehotencoder

enc = LabelBinarizer()  # Initialize LabelBinarizer                        #␣
 ↪Complete the code to intialize the labelBinarizer
y_train_encoded = enc.fit_transform(y_train)        # Complete the code to fit␣
 ↪and transform y_train
y_val_encoded=enc.transform(y_val)                  # Complete the code to␣
 ↪transform y_val
y_test_encoded=enc.transform(y_test)                # Complete the code to␣
 ↪transform y_test
```

[67]:
```
y_train_encoded.shape,y_val_encoded.shape,y_test_encoded.shape    #  to check␣
 ↪the shape of train, validation and test data
```

[67]:
```
((3847, 12), (428, 12), (475, 12))
```

### 1.6.5 Data 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.**

[68]:
```
#  to normalize the image pixels of train, test and validation data
X_train_normalized = X_train.astype('float32')/255.0
X_val_normalized = X_val.astype('float32')/255.0
X_test_normalized = X_test.astype('float32')/255.0
```

## 1.7 Model Building

[69]:
```
# Clearing backend
backend.clear_session()
```

[70]:
```
# Fixing the seed for random number generators
np.random.seed(42)
random.seed(42)
tf.random.set_seed(42)
```

[71]:
```
# Intializing a sequential model
model1 = Sequential()

# adding the first conv layer with 128 filters and kernel size 3x3 , padding␣
 ↪'same' provides the output size same as the input size
# Input_shape denotes input image dimension of images
model1.add(Conv2D(128, (3, 3), activation='relu', padding="same",␣
 ↪input_shape=(64, 64, 3)))

#  the max pooling to reduce the size of output of first conv layer
model1.add(MaxPooling2D((2, 2), padding = 'same'))

#  create two similar convolution and max-pooling layers activation = relu
model1.add(Conv2D(64, (3, 3), activation='relu', padding="same"))
model1.add(MaxPooling2D((2, 2), padding = 'same'))

model1.add(Conv2D(32, (3, 3), activation='relu', padding="same"))
model1.add(MaxPooling2D((2, 2), padding = 'same'))

#  to flatten the output of the conv layer after max pooling to make it ready␣
 ↪for creating dense connections
model1.add(Flatten())

#  to add a fully connected dense layer with 16 neurons
model1.add(Dense(16, activation='relu'))
model1.add(Dropout(0.3))
#  to add the output layer with 12 neurons and activation functions as softmax␣
 ↪since this is a multi-class classification problem
model1.add(Dense(12, activation='softmax'))

#  to use the Adam Optimizer
opt=Adam()
# e to Compile the model using suitable metric for loss fucntion
model1.compile(optimizer=opt, loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

#  to generate the summary of the model
model1.summary()
```

```
Model: "sequential"
```

```
 Layer (type)                Output Shape              ␣
↪Param #

 conv2d (Conv2D)             (None, 64, 64, 128)       ␣
↪3,584
```

```
max_pooling2d (MaxPooling2D)          (None, 32, 32, 128)                ␣
↪  0

conv2d_1 (Conv2D)                     (None, 32, 32, 64)                 ␣
↪73,792

max_pooling2d_1 (MaxPooling2D)        (None, 16, 16, 64)                 ␣
↪  0

conv2d_2 (Conv2D)                     (None, 16, 16, 32)                 ␣
↪18,464

max_pooling2d_2 (MaxPooling2D)        (None, 8, 8, 32)                   ␣
↪  0

flatten (Flatten)                     (None, 2048)                       ␣
↪  0

dense (Dense)                         (None, 16)                         ␣
↪32,784

dropout (Dropout)                     (None, 16)                         ␣
↪  0

dense_1 (Dense)                       (None, 12)                         ␣
↪204


Total params: 128,828 (503.23 KB)

Trainable params: 128,828 (503.23 KB)

Non-trainable params: 0 (0.00 B)
```

Fitting the model on the train data

[72]:
```
#  to fit the model on train and also using the validation data for validation
history_1 = model1.fit(
            X_train_normalized, y_train_encoded,
            epochs=30,
            validation_data=(X_val_normalized,y_val_encoded),
            batch_size=32,
            verbose=2
)
```

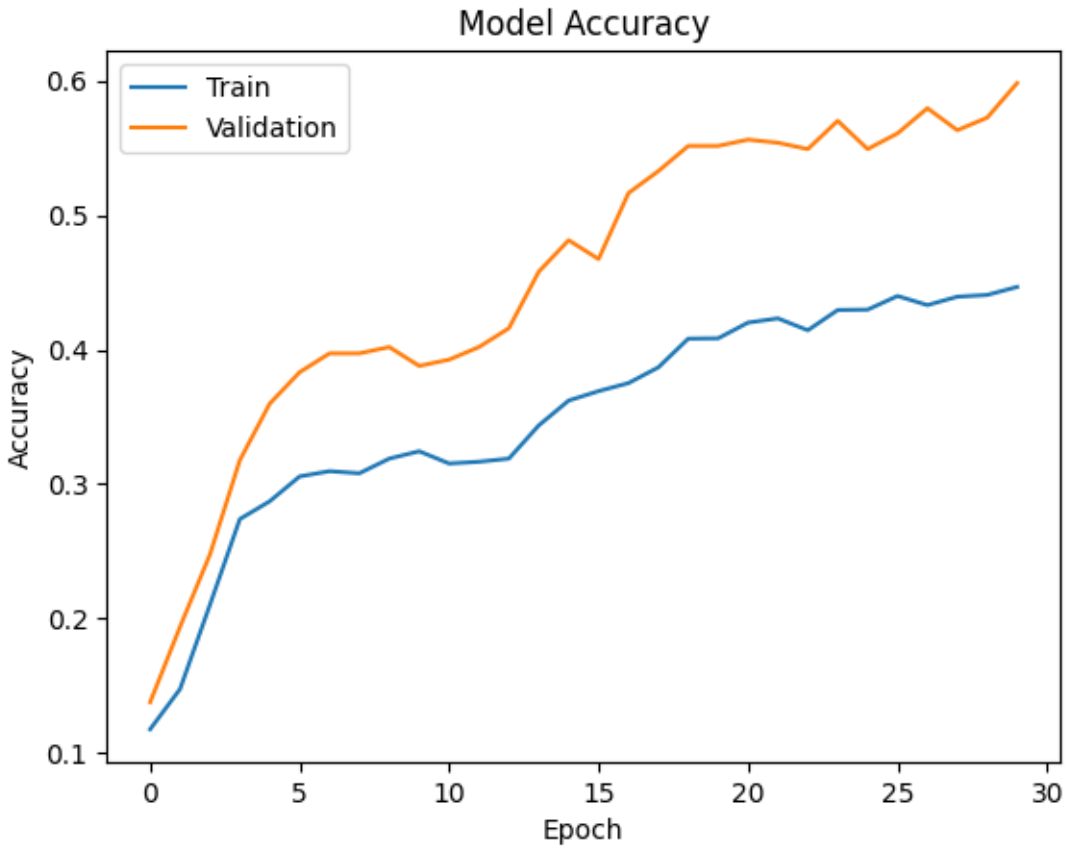
```
Epoch 1/30
121/121 - 12s - 98ms/step - accuracy: 0.1178 - loss: 2.4710 - val_accuracy:
0.1379 - val_loss: 2.4609
Epoch 2/30
121/121 - 2s - 18ms/step - accuracy: 0.1476 - loss: 2.4456 - val_accuracy:
0.1939 - val_loss: 2.4108
Epoch 3/30
121/121 - 2s - 17ms/step - accuracy: 0.2103 - loss: 2.3676 - val_accuracy:
0.2477 - val_loss: 2.2151
Epoch 4/30
121/121 - 2s - 19ms/step - accuracy: 0.2740 - loss: 2.2029 - val_accuracy:
0.3178 - val_loss: 2.1309
Epoch 5/30
121/121 - 2s - 14ms/step - accuracy: 0.2872 - loss: 2.1062 - val_accuracy:
0.3598 - val_loss: 2.0417
Epoch 6/30
121/121 - 2s - 19ms/step - accuracy: 0.3057 - loss: 2.0687 - val_accuracy:
0.3832 - val_loss: 1.9689
Epoch 7/30
121/121 - 3s - 21ms/step - accuracy: 0.3096 - loss: 2.0369 - val_accuracy:
0.3972 - val_loss: 1.8916
Epoch 8/30
121/121 - 3s - 22ms/step - accuracy: 0.3080 - loss: 2.0084 - val_accuracy:
0.3972 - val_loss: 1.8480
Epoch 9/30
121/121 - 2s - 15ms/step - accuracy: 0.3189 - loss: 1.9819 - val_accuracy:
0.4019 - val_loss: 1.8168
Epoch 10/30
121/121 - 1s - 12ms/step - accuracy: 0.3244 - loss: 1.9426 - val_accuracy:
0.3879 - val_loss: 1.7880
Epoch 11/30
121/121 - 3s - 21ms/step - accuracy: 0.3153 - loss: 1.9461 - val_accuracy:
0.3925 - val_loss: 1.7640
Epoch 12/30
121/121 - 3s - 22ms/step - accuracy: 0.3166 - loss: 1.9052 - val_accuracy:
0.4019 - val_loss: 1.6626
Epoch 13/30
121/121 - 2s - 13ms/step - accuracy: 0.3189 - loss: 1.8569 - val_accuracy:
0.4159 - val_loss: 1.5545
Epoch 14/30
121/121 - 3s - 23ms/step - accuracy: 0.3436 - loss: 1.7946 - val_accuracy:
0.4579 - val_loss: 1.5350
Epoch 15/30
121/121 - 2s - 20ms/step - accuracy: 0.3621 - loss: 1.7597 - val_accuracy:
0.4813 - val_loss: 1.4993
Epoch 16/30
121/121 - 2s - 20ms/step - accuracy: 0.3691 - loss: 1.7286 - val_accuracy:
0.4673 - val_loss: 1.4216
```

```
Epoch 17/30
121/121 - 1s - 10ms/step - accuracy: 0.3751 - loss: 1.7051 - val_accuracy:
0.5164 - val_loss: 1.3782
Epoch 18/30
121/121 - 1s - 11ms/step - accuracy: 0.3868 - loss: 1.6642 - val_accuracy:
0.5327 - val_loss: 1.3945
Epoch 19/30
121/121 - 1s - 10ms/step - accuracy: 0.4081 - loss: 1.6274 - val_accuracy:
0.5514 - val_loss: 1.3908
Epoch 20/30
121/121 - 1s - 11ms/step - accuracy: 0.4084 - loss: 1.6111 - val_accuracy:
0.5514 - val_loss: 1.3594
Epoch 21/30
121/121 - 1s - 10ms/step - accuracy: 0.4201 - loss: 1.5866 - val_accuracy:
0.5561 - val_loss: 1.3654
Epoch 22/30
121/121 - 1s - 10ms/step - accuracy: 0.4232 - loss: 1.5904 - val_accuracy:
0.5537 - val_loss: 1.3427
Epoch 23/30
121/121 - 1s - 12ms/step - accuracy: 0.4143 - loss: 1.5834 - val_accuracy:
0.5491 - val_loss: 1.3707
Epoch 24/30
121/121 - 2s - 20ms/step - accuracy: 0.4294 - loss: 1.5573 - val_accuracy:
0.5701 - val_loss: 1.3560
Epoch 25/30
121/121 - 1s - 10ms/step - accuracy: 0.4297 - loss: 1.5524 - val_accuracy:
0.5491 - val_loss: 1.3663
Epoch 26/30
121/121 - 1s - 10ms/step - accuracy: 0.4398 - loss: 1.5231 - val_accuracy:
0.5607 - val_loss: 1.3553
Epoch 27/30
121/121 - 1s - 10ms/step - accuracy: 0.4331 - loss: 1.5386 - val_accuracy:
0.5794 - val_loss: 1.3006
Epoch 28/30
121/121 - 1s - 10ms/step - accuracy: 0.4393 - loss: 1.5437 - val_accuracy:
0.5631 - val_loss: 1.3401
Epoch 29/30
121/121 - 1s - 11ms/step - accuracy: 0.4406 - loss: 1.5329 - val_accuracy:
0.5724 - val_loss: 1.3391
Epoch 30/30
121/121 - 1s - 10ms/step - accuracy: 0.4466 - loss: 1.5057 - val_accuracy:
0.5981 - val_loss: 1.2806
```

### Model Evaluation

[73]:
```
plt.plot(history_1.history['accuracy'])
plt.plot(history_1.history['val_accuracy'])
plt.title('Model Accuracy')
```

```
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```




**Evaluate the model on test data**

[74]:
```
accuracy = model1.evaluate(X_test_normalized, y_test_encoded, verbose=2)    #␣
↪Complete the code to evaluate the model on test data
```

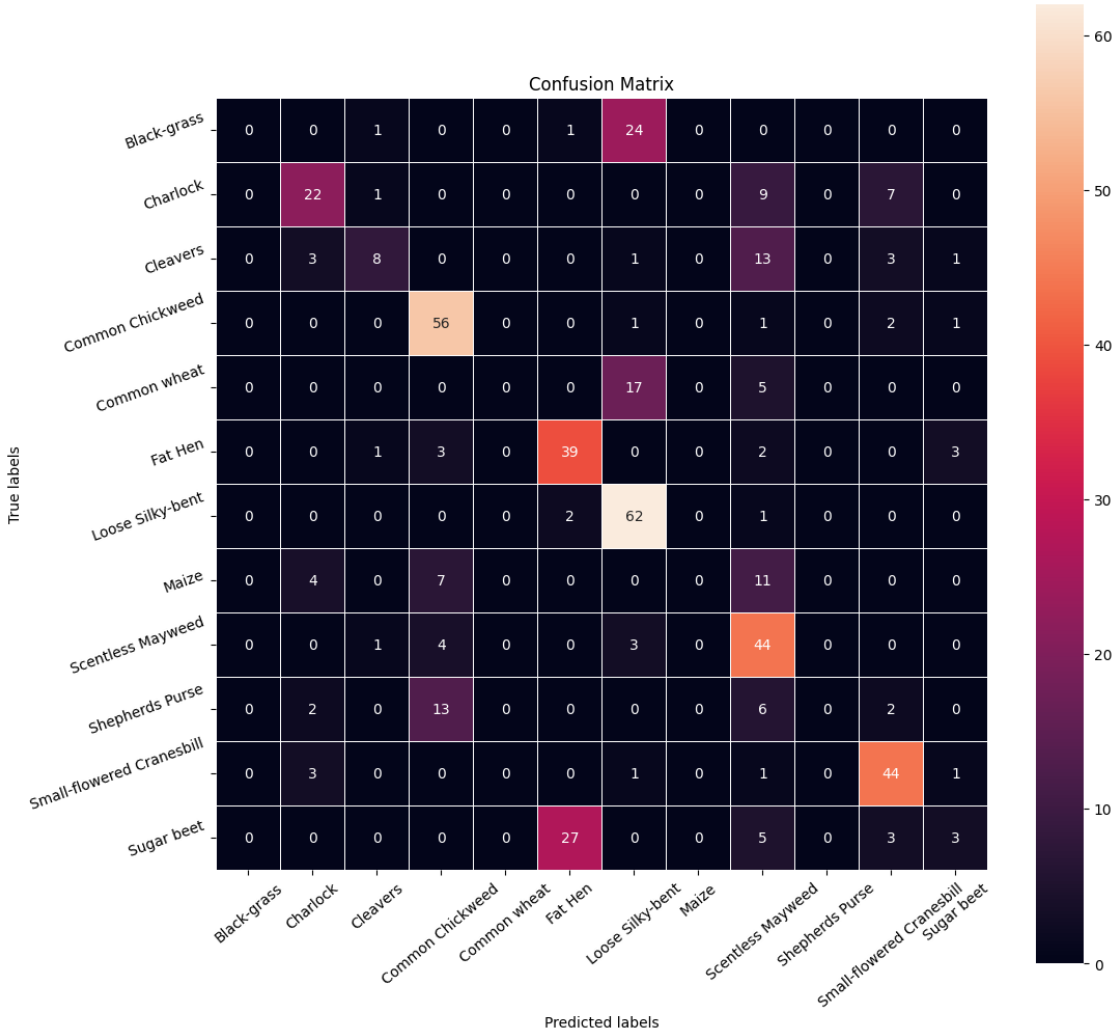
```
15/15 - 0s - 24ms/step - accuracy: 0.5853 - loss: 1.2595
```

**Plotting the Confusion Matrix**

[75]:
```
#  get the output as probablities for each category
y_pred=model1.predict(X_test_normalized)                    #  to predict␣
↪the output probabilities
```

```
15/15              1s 29ms/step
```

[76]:
```
# Obtaining the categorical values from y_test_encoded and y_pred
y_pred_arg=np.argmax(y_pred,axis=1)
y_test_arg=np.argmax(y_test_encoded,axis=1)

# Plotting the Confusion Matrix using confusion matrix() function which is also␣
 ↪predefined in tensorflow module
confusion_matrix = tf.math.confusion_matrix(y_test_arg,y_pred_arg)              ␣
 ↪#  to plot the confusion matrix
f, ax = plt.subplots(figsize=(12, 12))
sns.heatmap(
    confusion_matrix,
    annot=True,
    linewidths=.4,
    fmt="d",
    square=True,
    ax=ax
)
# Setting the labels to both the axes
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(list(enc.classes_),rotation=40)
ax.yaxis.set_ticklabels(list(enc.classes_),rotation=20)
plt.show()
```




### Plotting Classification Report

[77]:
```
# Plotting the classification report
# to plot the classification report
# Plotting the classification report
from sklearn import metrics #added
cr = metrics.classification_report(y_test_arg, y_pred_arg)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        26
           1       0.65      0.56      0.60        39
           2       0.67      0.28      0.39        29
           3       0.67      0.92      0.78        61
           4       0.00      0.00      0.00        22
```

```
           5       0.57      0.81      0.67        48
           6       0.57      0.95      0.71        65
           7       0.00      0.00      0.00        22
           8       0.45      0.85      0.59        52
           9       0.00      0.00      0.00        23
          10       0.72      0.88      0.79        50
          11       0.33      0.08      0.13        38

    accuracy                           0.59       475
   macro avg       0.39      0.44      0.39       475
weighted avg       0.47      0.59      0.50       475
```

## 1.8 Model Performance Improvement

**Reducing the Learning Rate:**

**ReduceLROnPlateau()** is a function that will be used to decrease the learning rate by some factor, if the loss is not decreasing for some time. This may start decreasing the loss at a smaller learning rate. There is a possibility that the loss may still not decrease. This may lead to executing the learning rate reduction again in an attempt to achieve a lower loss.

[101]:
```
# Code to monitor val_accuracy
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)
```

### 1.8.1 Data Augmentation

[79]:
```
# Clearing backend
from tensorflow.keras import backend
backend.clear_session()

# Fixing the seed for random number generators
import random
np.random.seed(42)
random.seed(42)
tf.random.set_seed(42)
```

[80]:
```
#  to set the rotation_range to 20
train_datagen = ImageDataGenerator(
                              rotation_range=20,
                              fill_mode='nearest'
                              )
```

[81]:
```
# Intializing a sequential model
model2 = Sequential()

#  add the first conv layer with 64 filters and kernel size 3x3 , padding␣
 ↪'same' provides the output size same as the input size
# Input_shape denotes input image dimension images
model2.add(Conv2D(64, (3, 3), activation='relu', padding="same",␣
 ↪input_shape=(64, 64, 3)))

#  to add max pooling to reduce the size of output of first conv layer
model2.add(MaxPooling2D((2, 2), padding = 'same'))


model2.add(Conv2D(32, (3, 3), activation='relu', padding="same"))
model2.add(MaxPooling2D((2, 2), padding = 'same'))
model2.add(BatchNormalization())

# flattening the output of the conv layer after max pooling to make it ready␣
 ↪for creating dense connections
model2.add(Flatten())

# Adding a fully connected dense layer with 16 neurons
model2.add(Dense(16, activation='relu'))

#  to add dropout with dropout_rate=0.3
model2.add(Dropout(0.3))
#  to add the output layer with 12 neurons and activation functions as softmax␣
 ↪since this is a multi-class classification problem
model2.add(Dense(12, activation='softmax'))

#  to initialize Adam Optimimzer
opt=Adam()
#  to Compile model
model2.compile(optimizer=opt, loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

# Generating the summary of the model
model2.summary()
```

```
Model: "sequential"
```

```
 Layer (type)                Output Shape              ␣
↪Param #

 conv2d (Conv2D)             (None, 64, 64, 64)        ␣
↪1,792
```

```
max_pooling2d (MaxPooling2D)            (None, 32, 32, 64)                  ␣
↪  0

conv2d_1 (Conv2D)                       (None, 32, 32, 32)                  ␣
↪18,464

max_pooling2d_1 (MaxPooling2D)          (None, 16, 16, 32)                  ␣
↪  0

batch_normalization                     (None, 16, 16, 32)                  ␣
↪128
(BatchNormalization)                                                        ␣
↪

flatten (Flatten)                       (None, 8192)                        ␣
↪  0

dense (Dense)                           (None, 16)                          ␣
↪131,088

dropout (Dropout)                       (None, 16)                          ␣
↪  0

dense_1 (Dense)                         (None, 12)                          ␣
↪204


Total params: 151,676 (592.48 KB)

Trainable params: 151,612 (592.23 KB)

Non-trainable params: 64 (256.00 B)
```

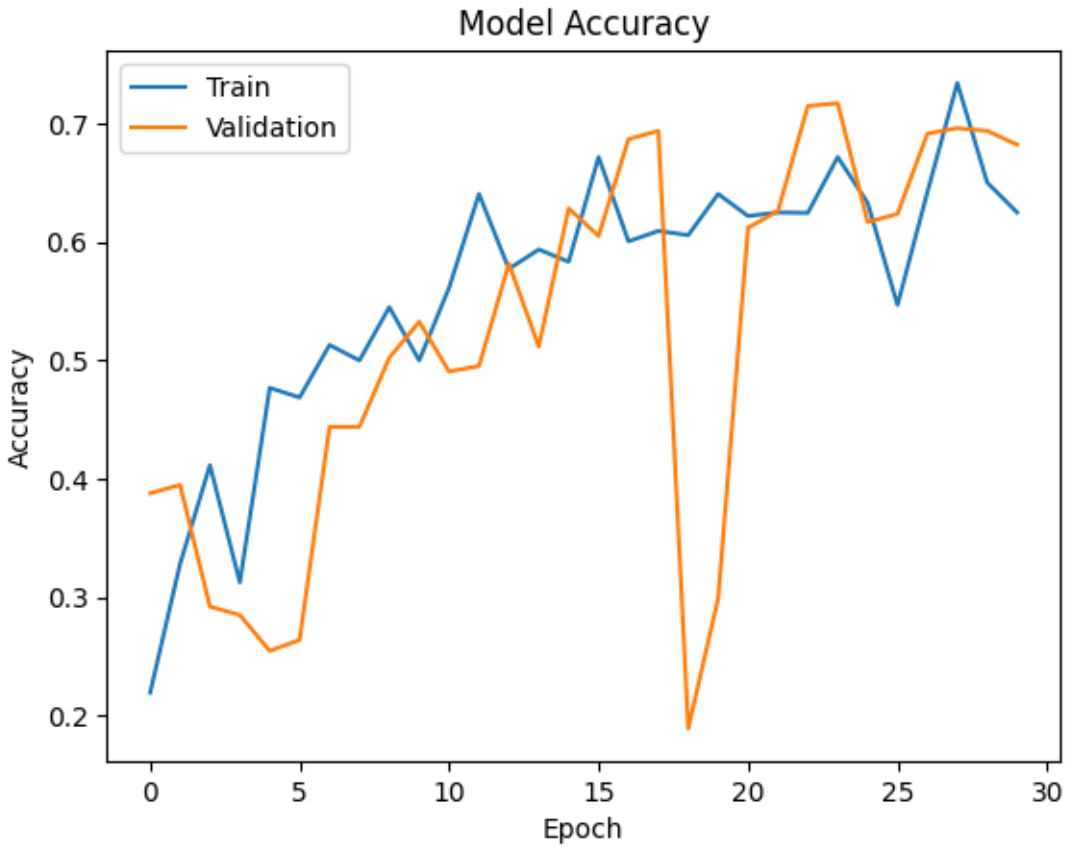
Fitting the model on the train data

[ ]:
```
#  to fit the model on train data with batch_size=64 and epochs=30
# Epochs
epochs = 30
# Batch size
batch_size = 64

history = model2.fit(train_datagen.flow(X_train_normalized,y_train_encoded,
                                        batch_size=batch_size,
                                        shuffle=False),
```

```
                                        epochs=epochs,
                                        steps_per_epoch=X_train_normalized.
↪shape[0] // batch_size,
                                    ␣
↪validation_data=(X_val_normalized,y_val_encoded),
                                    ␣
↪verbose=1,callbacks=[learning_rate_reduction])
```

**Model Evaluation**

[83]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```




**Evaluate the model on test data**

[84]:
```
accuracy = model2.evaluate(X_test_normalized, y_test_encoded, verbose=2)  #␣
 ↪Complete the code to evaluate the model on test data
```

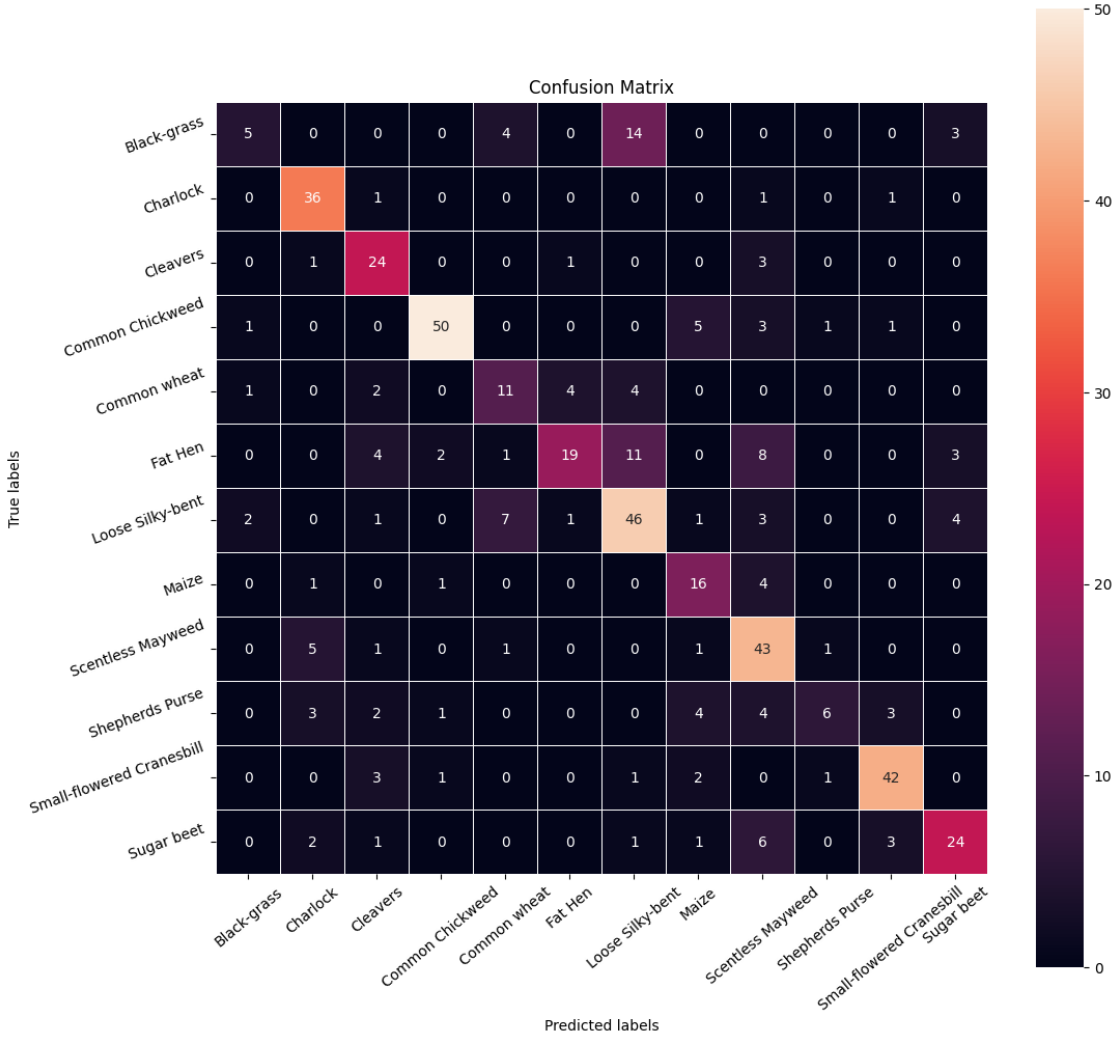
```
15/15 - 0s - 27ms/step - accuracy: 0.6779 - loss: 1.0066
```

### Plotting the Confusion Matrix

[85]:
```
#  to obtain the output probabilities
y_pred=model2.predict(X_test_normalized)
```

```
15/15              1s 31ms/step
```

[86]:
```
# Obtaining the categorical values from y_test_encoded and y_pred
y_pred_arg=np.argmax(y_pred,axis=1)
y_test_arg=np.argmax(y_test_encoded,axis=1)

# Plotting the Confusion Matrix using confusion matrix() function which is also␣
 ↪predefined in tensorflow module
confusion_matrix = tf.math.confusion_matrix(y_test_arg,y_pred_arg)     # to␣
 ↪obatin the confusion matrix
f, ax = plt.subplots(figsize=(12, 12))
sns.heatmap(
    confusion_matrix,
    annot=True,
    linewidths=.4,
    fmt="d",
    square=True,
    ax=ax
)
# Setting the labels to both the axes
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(list(enc.classes_),rotation=40)
ax.yaxis.set_ticklabels(list(enc.classes_),rotation=20)
plt.show()
```




### Plotting Classification Report

[87]:
```
# Plotting the classification report
cr=metrics.classification_report(y_test_arg,y_pred_arg)      #  to plot the␣
 ↪classification report
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.56      0.19      0.29        26
           1       0.75      0.92      0.83        39
           2       0.62      0.83      0.71        29
           3       0.91      0.82      0.86        61
           4       0.46      0.50      0.48        22
           5       0.76      0.40      0.52        48
           6       0.60      0.71      0.65        65
```

```
           7       0.53      0.73      0.62        22
           8       0.57      0.83      0.68        52
           9       0.67      0.26      0.38        23
          10       0.84      0.84      0.84        50
          11       0.71      0.63      0.67        38

    accuracy                           0.68       475
   macro avg       0.66      0.64      0.63       475
weighted avg       0.69      0.68      0.66       475
```

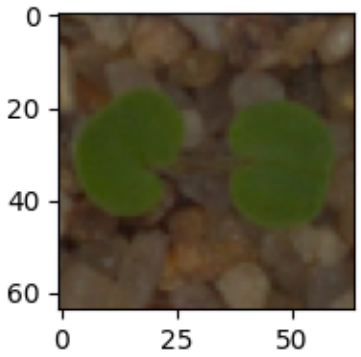
## 1.9 Final Model

---

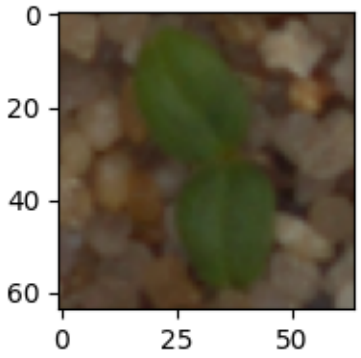
### 1.9.1 Visualizing the prediction

[88]:
```
# Visualizing the predicted and correct label of images from test data
plt.figure(figsize=(2,2))
plt.imshow(X_test[2])
plt.show()
##  to predict the test data using the final model selected
print('Predicted Label', enc.inverse_transform(model2.
 ↪predict((X_test_normalized[2].reshape(1,64,64,3)))))   # reshaping the input␣
 ↪image as we are only trying to predict using a single image
print('True Label', enc.inverse_transform(y_test_encoded)[2])                  ␣
 ↪                            # using inverse_transform() to get the output␣
 ↪label from the output vector

plt.figure(figsize=(2,2))
plt.imshow(X_test[33])
plt.show()
##  to predict the test data using the final model selected
print('Predicted Label', enc.inverse_transform(model2.
 ↪predict((X_test_normalized[33].reshape(1,64,64,3)))))  # reshaping the input␣
 ↪image as we are only trying to predict using a single image
print('True Label', enc.inverse_transform(y_test_encoded)[33])                 ␣
 ↪                            # using inverse_transform() to get the output␣
 ↪label from the output vector

plt.figure(figsize=(2,2))
plt.imshow(X_test[59],)
plt.show()
##  to predict the test data using the final model selected
print('Predicted Label', enc.inverse_transform(model2.
 ↪predict((X_test_normalized[59].reshape(1,64,64,3)))))  # reshaping the input␣
 ↪image as we are only trying to predict using a single image
```

```
print('True Label', enc.inverse_transform(y_test_encoded)[59])
 ↪                              # using inverse_transform() to get the output␣
 ↪label from the output vector

plt.figure(figsize=(2,2))
plt.imshow(X_test[36])
plt.show()
##  to predict the test data using the final model selected
print('Predicted Label', enc.inverse_transform(model2.
 ↪predict((X_test_normalized[36].reshape(1,64,64,3))))  # reshaping the input␣
 ↪image as we are only trying to predict using a single image
print('True Label', enc.inverse_transform(y_test_encoded)[36])
 ↪                              # using inverse_transform() to get the output␣
 ↪label from the output vector
```

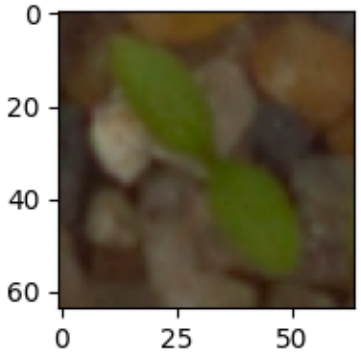



```
1/1             0s 308ms/step
Predicted Label ['Small-flowered Cranesbill']
True Label Small-flowered Cranesbill
```

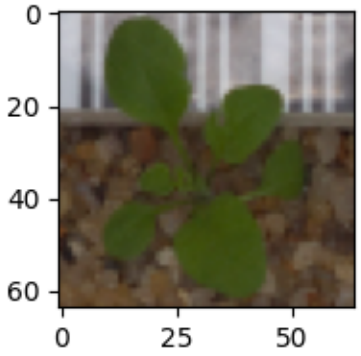



```
1/1             0s 30ms/step
```

```
Predicted Label ['Cleavers']
True Label Cleavers
```




```
1/1             0s 30ms/step
Predicted Label ['Common Chickweed']
True Label Common Chickweed
```




```
1/1             0s 75ms/step
Predicted Label ['Shepherds Purse']
True Label Shepherds Purse
```

### 1.10 Actionable Insights and Business Recommendations

- 

---

[89]:
```
# Convert the notebook to HTML
!jupyter nbconvert --to html EK_CV_Project_Low_Code_Notebook.ipynb

# Download the converted HTML file
```

```
from google.colab import files
files.download('EK_CV_Project_Low_Code_Notebook.html')
```

```
[NbConvertApp] Converting notebook EK_CV_Project_Low_Code_Notebook.ipynb to html
[NbConvertApp] WARNING | Alternative text is missing on 6 image(s).
[NbConvertApp] Writing 1696693 bytes to EK_CV_Project_Low_Code_Notebook.html
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

[90]:
```
!apt-get install texlive-xetex texlive-fonts-recommended texlive-plain-generic␣
↪-y
```

```
Reading package lists… Done
Building dependency tree… Done
Reading state information… Done
texlive-fonts-recommended is already the newest version (2021.20220204-1).
texlive-plain-generic is already the newest version (2021.20220204-1).
texlive-xetex is already the newest version (2021.20220204-1).
0 upgraded, 0 newly installed, 0 to remove and 22 not upgraded.
```

[91]:
```
import os
print(os.listdir())  # Check if the file exists in the current directory
```

```
['Labels.csv', 'CV_Project_Full_Code_Notebook.ipynb',
'CV_Project_Low_Code_Notebook.ipynb', 'CV_Project_PresentationTemplate.pptx',
'images.npy', 'EK_Project_Presentation.gslides', '=2.0.0,',
'EK_CV_Project_Low_Code_Notebook.html', 'EK_CV_Project_Low_Code_Notebook.pdf',
'EK_CV_Project_Low_Code_Notebook.ipynb']
```

[92]:
```
!apt-get install -y pandoc
```

```
Reading package lists… Done
Building dependency tree… Done
Reading state information… Done
pandoc is already the newest version (2.9.2.1-3ubuntu2).
0 upgraded, 0 newly installed, 0 to remove and 22 not upgraded.
```

[ ]:
```
!jupyter nbconvert --to pdf EK_CV_Project_Low_Code_Notebook.ipynb
```

```
[NbConvertApp] Converting notebook EK_CV_Project_Low_Code_Notebook.ipynb to pdf
[NbConvertApp] Support files will be in EK_CV_Project_Low_Code_Notebook_files/
[NbConvertApp] Making directory ./EK_CV_Project_Low_Code_Notebook_files
[NbConvertApp] Writing 109561 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
```

[97]:
```
# @title
from google.colab import files
files.download('EK_CV_Project_Low_Code_Notebook.pdf')
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```