Plant Seedlings Image Classification CNN - Computer Vision Project _Prabhu_12Feb2021

###

Context:

- Can you differentiate a weed from a crop seedling?
- The ability to do so effectively can mean better crop yields and better stewardship of the environment.
- The Aarhus University Signal Processing group, in collaboration with University of Southern Denmark, has recently released a dataset containing images of unique plants belonging to 12 species at several growth stages

Objective:

To implement the techniques learnt as a part of the course.

Importing necessary libraries

```
# Import necessary libraries.
import cv2
import numpy as np                                # Import numpy
import pandas as pd                                # Import numpy
import seaborn as sns                              # Import Seaborn
from skimage import data, io                      # Import skimage library (data - Test images
import matplotlib.pyplot as plt                   # Import matplotlib.pyplot (Plotting framewo
%matplotlib inline
import os                                         # This module provides a portable way of usi
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import math
from glob import glob
import tensorflow as tf                            # Import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (
    Dense,
    Dropout,
    Flatten,
    Conv2D,
    MaxPooling2D,
    MaxPool2D,
    GlobalMaxPooling2D,
    BatchNormalization
)
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras import datasets, models, layers, optimizers
from tensorflow.keras.optimizers import RMSprop, Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.utils.np_utils import to_categorical            # convert to one-hot-encoding
from google.colab.patches import cv2_imshow

from sklearn.model_selection import train_test_split       # Import test_train_split from skl
from sklearn.metrics import classification_report, confusion_matrix

import warnings
warnings.filterwarnings('ignore')        # Suppress warnings
```

```
# Mount Google drive so dataset can be accessed
from google.colab import drive
drive.mount('/content/drive')
```

```
     Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mour
```

```
#from zipfile import ZipFile
#with ZipFile(train_path, 'r') as zip:
#  zip.extractall(extract_path)
```

Load dataset, print shape of data, visualize the images in dataset

```
trainLabel = pd.read_csv('/content/drive/My Drive/Colab Notebooks/Labels.csv')
print(trainLabel.shape)
```

```
     (4750, 1)
```

```
trainImg = np.load('/content/drive/My Drive/Colab Notebooks/images.npy')

print(trainImg.shape)
```

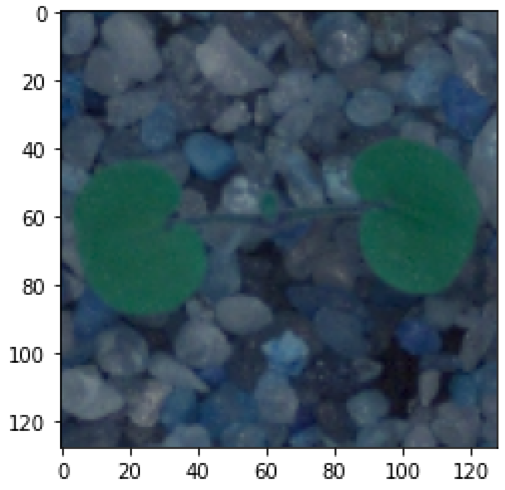
```
     (4750, 128, 128, 3)
```

```
print(f"Training image array shape:{trainImg.shape}")
print(f"Training target labels:{trainLabel.shape}")
```

```
     Training image array shape:(4750, 128, 128, 3)
     Training target labels:(4750, 1)
```

```
# Check Images
import matplotlib.pyplot as plt
plt.imshow(trainImg[0])
```

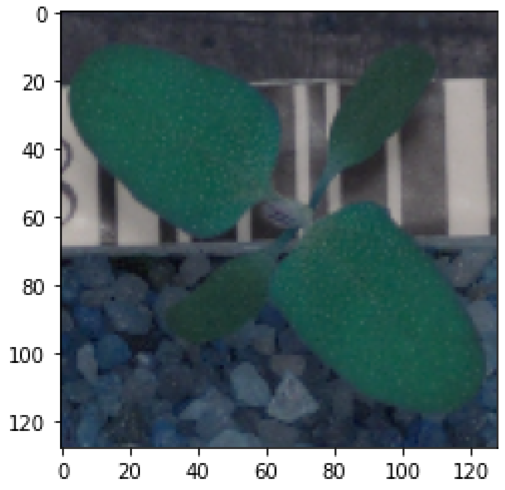
```
<matplotlib.image.AxesImage at 0x7f5763015dd8>
```



```
#sobel = cv2.Sobel(img, cv2.CV_64F, 1, 1, ksize=5)
#plt.imshow(sobel)
```

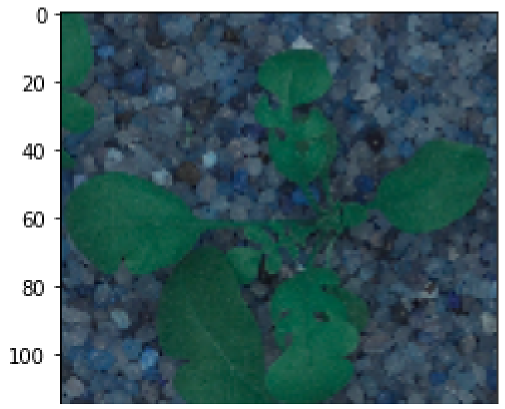
```
# Check Images
import matplotlib.pyplot as plt
plt.imshow(trainImg[500])
```

```
<matplotlib.image.AxesImage at 0x7f5762b768d0>
```



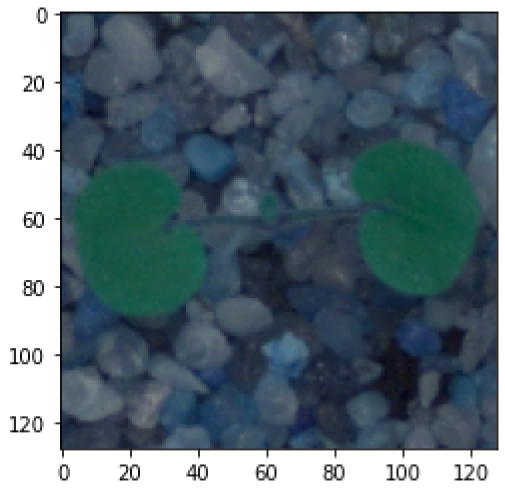
```
# Check Images
import matplotlib.pyplot as plt
plt.imshow(trainImg[1000])
```

```
<matplotlib.image.AxesImage at 0x7f5761ae2b70>
```



```
# Check Images trainImg [0] -- cv2.Sobel
import matplotlib.pyplot as plt
plt.imshow(trainImg[0])
```

```
<matplotlib.image.AxesImage at 0x7f5761a4b978>
```



```
#sobel = cv2.Sobel(img, cv2.CV_64F, 1, 1, ksize=5)
#plt.imshow(sobel)
```

Pre-processing & Normalizing the data

```
trainImg = trainImg.astype('float32')
trainImg /= 255
# Check the nomalized data
print(f'Shape of the Train array:{trainImg.shape}')
print(f'Minimum value in the Train Array:{trainImg.min()}')
print(f'Maximum value in the Train Array:{trainImg.max()}')
```

```
Shape of the Train array:(4750, 128, 128, 3)
Minimum value in the Train Array:0.0
Maximum value in the Train Array:1.0
```

```
# Step#1: Split train and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(trainImg, trainLabel, test_size=0.3, rand
X_train.shape, X_test.shape
```

```
((3325, 128, 128, 3), (1425, 128, 128, 3))
```

```
# Step#2: Split validation from test set
X_test, X_validation, y_test, y_validation = train_test_split(X_test, y_test, test_size=0.5,
X_test.shape, X_validation.shape
```

```
((712, 128, 128, 3), (713, 128, 128, 3))
```

One Hot Encoding to Target Values

```
from sklearn.preprocessing import LabelBinarizer
encoder = LabelBinarizer()
y_train = encoder.fit_transform(y_train)
y_test = encoder.fit_transform(y_test)
y_validation = encoder.fit_transform(y_validation)
```

```
# Display target variable
y_train[0]
```

```
array([0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

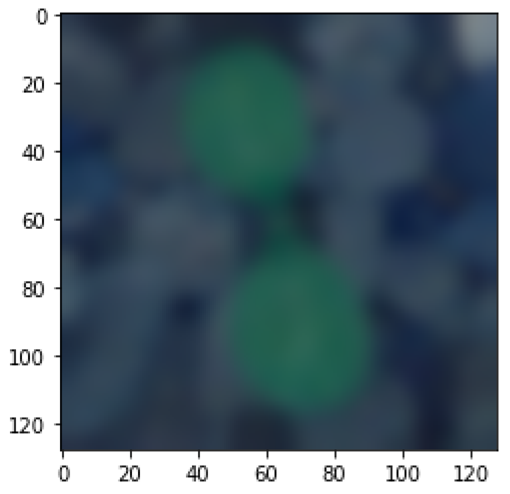
Gaussian Blurring

```
# Preview the image before Gaussian Blur
plt.imshow(X_train[1], cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f57619c59e8>
```

```
plt.imshow(cv2.GaussianBlur(X_train[1], (15,15), 0))
```

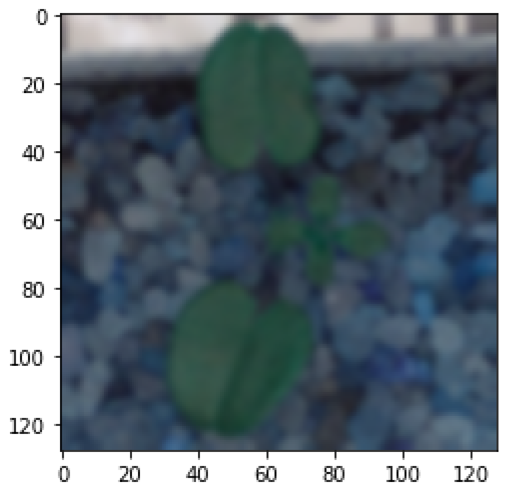
```
<matplotlib.image.AxesImage at 0x7f57619a9898>
```



```
# Now we apply the gaussian blur to each 128x128 pixels array (image) to reduce the noise in
for idx, img in enumerate(X_train):
  X_train[idx] = cv2.GaussianBlur(img, (5, 5), 0)


# Preview the image after Gaussian Blur
plt.imshow(X_train[0], cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f5761917630>
```



```
# Gaussian Blur to Test and Validation sets
for idx, img in enumerate(X_test):
  X_test[idx] = cv2.GaussianBlur(img, (5, 5), 0)

for idx, img in enumerate(X_validation):
  X_validation[idx] = cv2.GaussianBlur(img, (5, 5), 0)
```

Creating a CNN Model

Steps:

- Initialize CNN Classifier
- Add Convolution layer with 32 kernels of 3x3 shape
- Add Maxpooling layer of size 2x2
- Flatten the input array
- Add dense layer with relu activation function
- Dropout the probability
- Add softmax Dense layer as output

```
def create_model(input_shape, num_classes):
  # Initialize CNN Classified
  model = Sequential()

  # Add convolution layer with 32 filters and 3 kernels
  model.add(Conv2D(32, (3,3), input_shape=input_shape, padding='same', activation=tf.nn.relu)
  model.add(MaxPooling2D(pool_size=(2,2)))
  model.add(Dropout(rate=0.25))

  # Add convolution layer with 32 filters and 3 kernels
  model.add(Conv2D(filters=32, kernel_size=3, padding='same', activation=tf.nn.relu))
  model.add(Conv2D(filters=64, kernel_size=3, padding='same', activation=tf.nn.relu))
  model.add(MaxPooling2D(pool_size=(2,2)))
  model.add(Dropout(rate=0.25))

  # Add convolution layer with 32 filters and 3 kernels
  model.add(Conv2D(filters=32, kernel_size=3, padding='same', activation=tf.nn.relu))
  model.add(MaxPooling2D(pool_size=(2,2)))
  model.add(Dropout(rate=0.25))

  # Flatten the 2D array to 1D array
  model.add(Flatten())

  # Create fully connected layers with 512 units
  model.add(Dense(512, activation=tf.nn.relu))
  model.add(Dropout(0.5))


  # Adding a fully connected layer with 128 neurons
  model.add(Dense(units = 128, activation = tf.nn.relu))
  model.add(Dropout(0.5))

  # The final output layer with 12 neurons to predict the categorical classifcation
  model.add(Dense(units = num_classes, activation = tf.nn.softmax))
  return model
```

```
class myCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    if(logs.get('accuracy')>0.95):
      print("\nReached 95% accuracy so cancelling training!")
      self.model.stop_training = True

callbacks = myCallback()

es = EarlyStopping(monitor='val_accuracy', mode='min', verbose=1, patience=10)


input_shape = X_train.shape[1:] # Input shape of X_train
num_classes = y_train.shape[1] # Target column size

model = create_model(input_shape, num_classes)
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001) # Optimizer
 #optimizer = tf.keras.optimizers.SGD(lr=1 * 1e-1, momentum=0.9, nesterov=True)

model.compile(optimizer=optimizer,
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

```
     Model: "sequential"
     _________________________________________________________________
     Layer (type)                 Output Shape              Param #
     =================================================================
     conv2d (Conv2D)              (None, 128, 128, 32)      896
     _________________________________________________________________
     max_pooling2d (MaxPooling2D) (None, 64, 64, 32)        0
     _________________________________________________________________
     dropout (Dropout)            (None, 64, 64, 32)        0
     _________________________________________________________________
     conv2d_1 (Conv2D)            (None, 64, 64, 32)        9248
     _________________________________________________________________
     conv2d_2 (Conv2D)            (None, 64, 64, 64)        18496
     _________________________________________________________________
     max_pooling2d_1 (MaxPooling2 (None, 32, 32, 64)        0
     _________________________________________________________________
     dropout_1 (Dropout)          (None, 32, 32, 64)        0
     _________________________________________________________________
     conv2d_3 (Conv2D)            (None, 32, 32, 32)        18464
     _________________________________________________________________
     max_pooling2d_2 (MaxPooling2 (None, 16, 16, 32)        0
     _________________________________________________________________
     dropout_2 (Dropout)          (None, 16, 16, 32)        0
     _________________________________________________________________
     flatten (Flatten)            (None, 8192)              0
     _________________________________________________________________
     dense (Dense)                (None, 512)               4194816
     _________________________________________________________________
     dropout_3 (Dropout)          (None, 512)               0
```

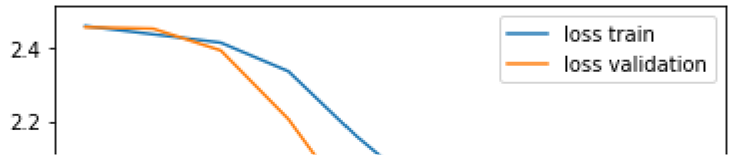
```
_________________________________________________________________
dense_1 (Dense)              (None, 128)               65664     
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 12)                1548      
=================================================================
Total params: 4,309,132
Trainable params: 4,309,132
Non-trainable params: 0
_________________________________________________________________
```

```
history = model.fit(X_train, y_train, validation_data=(X_validation, y_validation), epochs=10
```

```
Epoch 1/10
6/7 [========================>.....] - ETA: 0s - loss: 2.4710 - accuracy: 0.1239WARNING
7/7 [==============================] - 8s 620ms/step - loss: 2.4685 - accuracy: 0.1250 -
Epoch 2/10
7/7 [==============================] - 2s 279ms/step - loss: 2.4392 - accuracy: 0.1209 -
Epoch 3/10
7/7 [==============================] - 2s 280ms/step - loss: 2.4232 - accuracy: 0.1409 -
Epoch 4/10
7/7 [==============================] - 2s 284ms/step - loss: 2.3558 - accuracy: 0.2091 -
Epoch 5/10
7/7 [==============================] - 2s 283ms/step - loss: 2.1914 - accuracy: 0.2848 -
Epoch 6/10
7/7 [==============================] - 2s 288ms/step - loss: 2.0196 - accuracy: 0.3191 -
Epoch 7/10
7/7 [==============================] - 2s 286ms/step - loss: 1.9176 - accuracy: 0.3432 -
Epoch 8/10
7/7 [==============================] - 2s 290ms/step - loss: 1.8106 - accuracy: 0.3816 -
Epoch 9/10
7/7 [==============================] - 2s 286ms/step - loss: 1.6893 - accuracy: 0.4175 -
Epoch 10/10
7/7 [==============================] - 2s 288ms/step - loss: 1.5592 - accuracy: 0.4499 -
```

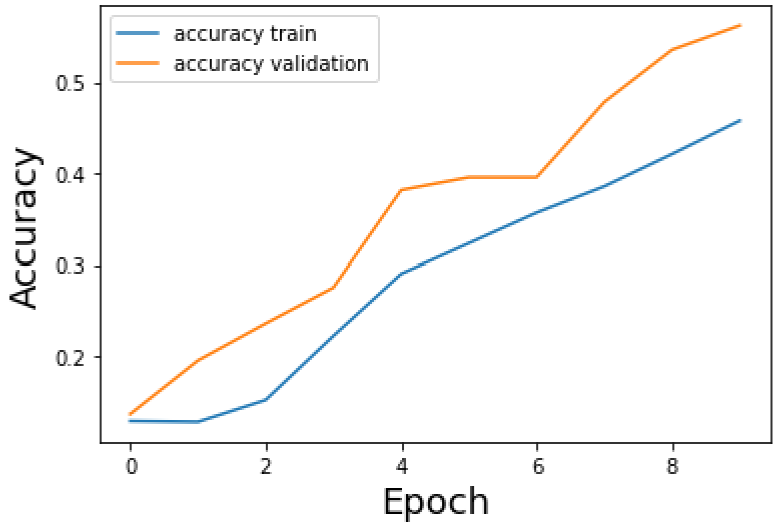
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('Epoch', fontsize=18)
plt.ylabel(r'Loss', fontsize=18)
plt.legend(('loss train','loss validation'), loc=0)
```

```
<matplotlib.legend.Legend at 0x7f570064e400>
```



```
# Print accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.xlabel('Epoch', fontsize=18)
plt.ylabel(r'Accuracy', fontsize=18)
plt.legend(('accuracy train','accuracy validation'), loc=0)
```

```
<matplotlib.legend.Legend at 0x7f5700637940>
```



Model Evaluation

```
loss, accuracy = model.evaluate(X_test, y_test)
print('Test loss: {:.2f} \n Test accuracy: {:.2f}'.format(loss, accuracy))

loss, accuracy = model.evaluate(X_train, y_train)
print('Train loss: {:.2f} \n Train accuracy: {:.2f}'.format(loss, accuracy))
```

```
23/23 [==============================] - 0s 9ms/step - loss: 1.4048 - accuracy: 0.5379
Test loss: 1.40
 Test accuracy: 0.54
104/104 [==============================] - 1s 8ms/step - loss: 1.3626 - accuracy: 0.570
Train loss: 1.36
 Train accuracy: 0.57
```

Retraining the Model

- Above try with epochs=10 and batch size=500 resulted in test accuracy = 0.54, validation accuracy of 0.57 which is low and loss is high -- -shall retry and train model
- Try and retrain "model1" with different epochs= 30 and batch size = 100 to see if test and validation accurracy increases and loss decreases

```
# Retrain as model1
model1 = create_model(input_shape, num_classes)
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001) # Optimizer

model1.compile(optimizer=optimizer,
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model1.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_4 (Conv2D)            (None, 128, 128, 32)      896       
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 64, 64, 32)        0         
_________________________________________________________________
dropout_5 (Dropout)          (None, 64, 64, 32)        0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 64, 64, 32)        9248      
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 64, 64, 64)        18496     
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 32, 32, 64)        0         
_________________________________________________________________
dropout_6 (Dropout)          (None, 32, 32, 64)        0         
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 32, 32, 32)        18464     
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 16, 16, 32)        0         
_________________________________________________________________
dropout_7 (Dropout)          (None, 16, 16, 32)        0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0         
_________________________________________________________________
dense_3 (Dense)              (None, 512)               4194816   
_________________________________________________________________
dropout_8 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 128)               65664     
_________________________________________________________________
dropout_9 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_5 (Dense)              (None, 12)                1548      
=================================================================
Total params: 4,309,132
```

```
    Trainable params: 4,309,132
    Non-trainable params: 0
    _________________________________________________________________
```

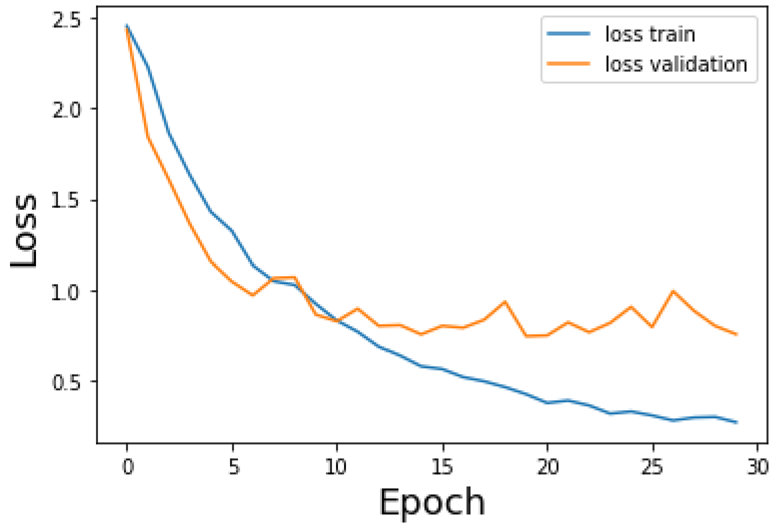
```
# Above try with epochs=10 and batch size=500  resulted in val accuracy of 0.45 which is not
#try with different epochs= 30 and batch size = 100 val accurracy increases significantly to
history = model1.fit(X_train, y_train, validation_data=(X_validation, y_validation), epochs=3
```

```
    Epoch 1/30
     6/34 [====>.........................] - ETA: 1s - loss: 2.4864 - accuracy: 0.0886WAR
    34/34 [==============================] - 4s 73ms/step - loss: 2.4653 - accuracy: 0.110
    Epoch 2/30
    34/34 [==============================] - 2s 60ms/step - loss: 2.3330 - accuracy: 0.22
    Epoch 3/30
    34/34 [==============================] - 2s 61ms/step - loss: 1.9234 - accuracy: 0.33
    Epoch 4/30
    34/34 [==============================] - 2s 61ms/step - loss: 1.7009 - accuracy: 0.41
    Epoch 5/30
    34/34 [==============================] - 2s 59ms/step - loss: 1.5036 - accuracy: 0.47
    Epoch 6/30
    34/34 [==============================] - 2s 59ms/step - loss: 1.3411 - accuracy: 0.54
    Epoch 7/30
    34/34 [==============================] - 2s 59ms/step - loss: 1.1377 - accuracy: 0.60
    Epoch 8/30
    34/34 [==============================] - 2s 63ms/step - loss: 1.0444 - accuracy: 0.64
    Epoch 9/30
    34/34 [==============================] - 2s 59ms/step - loss: 1.0003 - accuracy: 0.63
    Epoch 10/30
    34/34 [==============================] - 2s 59ms/step - loss: 0.9684 - accuracy: 0.670
    Epoch 11/30
    34/34 [==============================] - 2s 59ms/step - loss: 0.8143 - accuracy: 0.72
    Epoch 12/30
    34/34 [==============================] - 2s 59ms/step - loss: 0.7612 - accuracy: 0.73
    Epoch 13/30
    34/34 [==============================] - 2s 59ms/step - loss: 0.7043 - accuracy: 0.74
    Epoch 14/30
    34/34 [==============================] - 2s 59ms/step - loss: 0.6228 - accuracy: 0.77
    Epoch 15/30
    34/34 [==============================] - 2s 59ms/step - loss: 0.6179 - accuracy: 0.77
    Epoch 16/30
    34/34 [==============================] - 2s 59ms/step - loss: 0.5138 - accuracy: 0.81
    Epoch 17/30
    34/34 [==============================] - 2s 60ms/step - loss: 0.5363 - accuracy: 0.81
    Epoch 18/30
    34/34 [==============================] - 2s 59ms/step - loss: 0.4774 - accuracy: 0.83
    Epoch 19/30
    34/34 [==============================] - 2s 58ms/step - loss: 0.4548 - accuracy: 0.84
    Epoch 20/30
    34/34 [==============================] - 2s 59ms/step - loss: 0.4451 - accuracy: 0.84
    Epoch 21/30
    34/34 [==============================] - 2s 60ms/step - loss: 0.3778 - accuracy: 0.86
    Epoch 22/30
    34/34 [==============================] - 2s 59ms/step - loss: 0.4141 - accuracy: 0.85
    Epoch 23/30
    34/34 [==============================] - 2s 60ms/step - loss: 0.3752 - accuracy: 0.85
    Epoch 24/30
```

```
34/34 [==============================] - 2s 60ms/step - loss: 0.3121 - accuracy: 0.87
Epoch 25/30
34/34 [==============================] - 2s 60ms/step - loss: 0.3185 - accuracy: 0.88
Epoch 26/30
34/34 [==============================] - 2s 59ms/step - loss: 0.3323 - accuracy: 0.88
Epoch 27/30
34/34 [==============================] - 2s 60ms/step - loss: 0.2566 - accuracy: 0.89
Epoch 28/30
34/34 [==============================] - 2s 60ms/step - loss: 0.3018 - accuracy: 0.90
Epoch 29/30
```

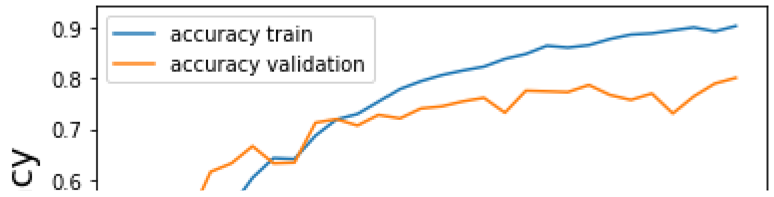
```
# Print Loss ( Model1) -- retrained model
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('Epoch', fontsize=18)
plt.ylabel(r'Loss', fontsize=18)
plt.legend(('loss train','loss validation'), loc=0)
```

```
<matplotlib.legend.Legend at 0x7f5761b146d8>
```



```
# Print Accuracy ( Model1) -- retrained model
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.xlabel('Epoch', fontsize=18)
plt.ylabel(r'Accuracy', fontsize=18)
plt.legend(('accuracy train','accuracy validation'), loc=0)
```

```
<matplotlib.legend.Legend at 0x7f570031da90>
```



Model Evaluation after re-training (model1)

```
loss, accuracy = model1.evaluate(X_test, y_test)
print('Test loss: {:.2f} \n Test accuracy: {:.2f}'.format(loss, accuracy))

loss, accuracy = model1.evaluate(X_train, y_train)
print('Train loss: {:.2f} \n Train accuracy: {:.2f}'.format(loss, accuracy))
```

```
23/23 [==============================] - 0s 7ms/step - loss: 0.6110 - accuracy: 0.8160
Test loss: 0.61
 Test accuracy: 0.82
104/104 [==============================] - 1s 7ms/step - loss: 0.1025 - accuracy: 0.960
Train loss: 0.10
 Train accuracy: 0.96
```

- From above retained "model1" test accuracy = 0.82 validation accuracy = 0.96 increased & loss decreased --- which is very good

Confusion Matrix

```
y_pred = model1.predict(X_test)
y_pred = (y_pred > 0.5)


print("=== Confusion Matrix ===")
print(confusion_matrix(y_test.argmax(axis=1), y_pred.argmax(axis=1)))
```

```
=== Confusion Matrix ===
[[17  0  0  0  2  0 21  0  0  0  0  1]
 [ 1 50  2  0  0  0  0  0  0  2  0  0]
 [ 2  4 39  1  0  1  0  0  1  0  1  0]
 [ 0  0  0 83  0  0  0  1  2  2  0  0]
 [ 8  0  0  0 20  2  1  0  2  0  0  0]
 [ 1  1  3  1  0 62  3  0  1  1  0  1]
 [ 7  0  1  0  2  1 89  0  1  0  0  0]
 [ 3  0  0  0  0  0  0 24  2  0  1  0]
 [ 6  0  0  1  0  0  0  1 63  2  0  1]
 [ 2  0  0  3  0  0  0  0 12 30  1  0]
 [ 4  2  0  4  0  0  0  1  1  0 60  0]
 [ 2  0  2  0  0  1  0  0  1  0  0 41]]
```

```
print("=== Classification Report ===")
print(classification_report(y_test.argmax(axis=1), y_pred.argmax(axis=1)))
```

```
=== Classification Report ===
              precision    recall  f1-score   support

           0       0.32      0.41      0.36        41
           1       0.88      0.91      0.89        55
           2       0.83      0.80      0.81        49
           3       0.89      0.94      0.92        88
           4       0.83      0.61      0.70        33
           5       0.93      0.84      0.88        74
           6       0.78      0.88      0.83       101
           7       0.89      0.80      0.84        30
           8       0.73      0.85      0.79        74
           9       0.81      0.62      0.71        48
          10       0.95      0.83      0.89        72
          11       0.93      0.87      0.90        47

    accuracy                           0.81       712
   macro avg       0.81      0.78      0.79       712
weighted avg       0.83      0.81      0.81       712
```

- Precision: Out of all the positive classes we have predicted correctly, how many are actually positive.
- Recall: Out of all the positive classes, how much we predicted correctly. It should be high as possible.
- F1-Score: F1 Score is the weighted average of Precision and Recall.

Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution

Visualize predictions for x_test[2], x_test[3], x_test[33], x_test[36], x_test[59]

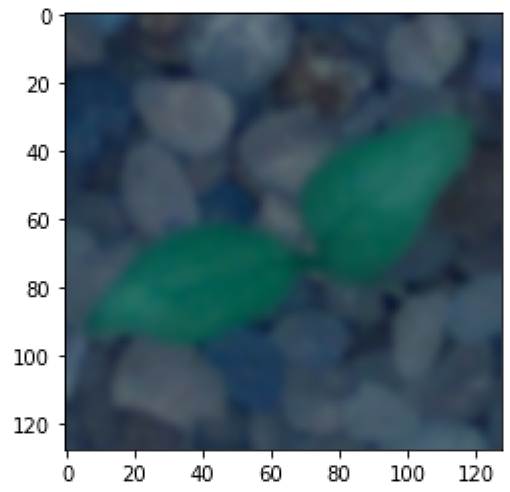
```
y_pred = encoder.inverse_transform(y_pred)

index = 2
plt.imshow(X_test[index], cmap='gray')
print("Predicted label:", y_pred[index])
```

```
Predicted label: Black-grass
```

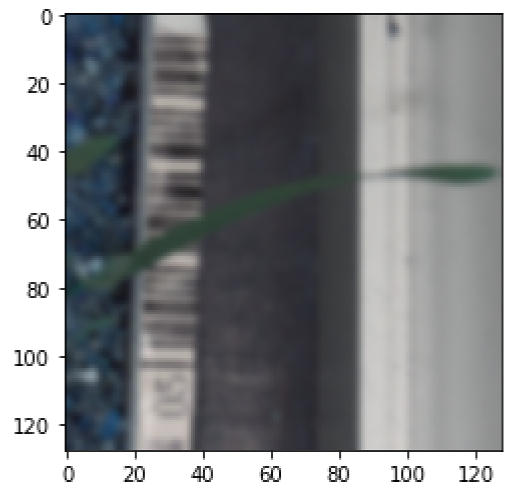
```
index = 3
plt.imshow(X_test[index], cmap='gray')
print("Predicted label:", y_pred[index])
```

```
Predicted label: Common Chickweed
```



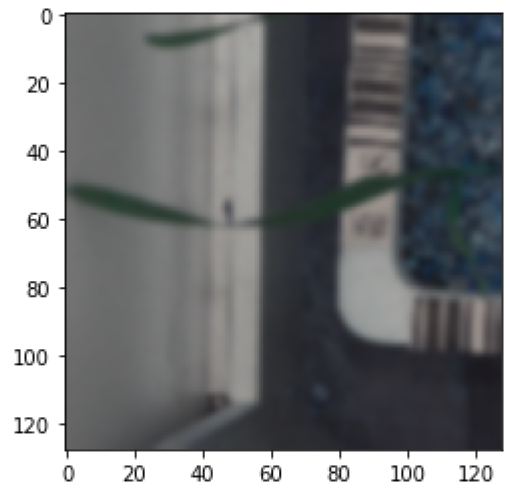
```
index = 33
plt.imshow(X_test[index], cmap='gray')
print("Predicted label:", y_pred[index])
```

```
Predicted label: Black-grass
```



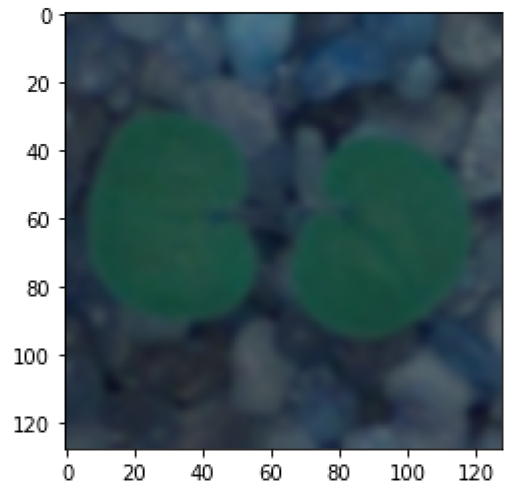
```
index = 36
plt.imshow(X_test[index], cmap='gray')
print("Predicted label:", y_pred[index])
```

```
Predicted label: Black-grass
```



```
index = 59
plt.imshow(X_test[index], cmap='gray')
print("Predicted label:", y_pred[index])
```

```
Predicted label: Small-flowered Cranesbill
```



Model Prediction

```
y_pred    # Model prediction below array shows all predicted species ( seedlings and weeds)
```

```
array(['Common Chickweed', 'Scentless Mayweed', 'Black-grass',
       'Common Chickweed', 'Loose Silky-bent', 'Common Chickweed',
       'Sugar beet', 'Small-flowered Cranesbill', 'Sugar beet',
       'Black-grass', 'Common Chickweed', 'Maize', 'Common Chickweed',
       'Sugar beet', 'Charlock', 'Shepherds Purse', 'Fat Hen',
       'Common Chickweed', 'Maize', 'Loose Silky-bent',
       'Small-flowered Cranesbill', 'Loose Silky-bent',
       'Common Chickweed', 'Shepherds Purse', 'Common Chickweed',
       'Small-flowered Cranesbill', 'Charlock', 'Charlock',
       'Small-flowered Cranesbill', 'Black-grass', 'Loose Silky-bent',
       'Loose Silky-bent', 'Maize', 'Black-grass', 'Charlock', 'Maize',
       'Black-grass', 'Shepherds Purse', 'Common Chickweed',
       'Loose Silky-bent', 'Common Chickweed', 'Loose Silky-bent',
       'Loose Silky-bent', 'Common Chickweed', 'Shepherds Purse',
       'Loose Silky-bent', 'Common Chickweed', 'Loose Silky-bent',
       'Scentless Mayweed', 'Fat Hen', 'Small-flowered Cranesbill',
       'Scentless Mayweed', 'Loose Silky-bent', 'Common Chickweed',
       'Scentless Mayweed', 'Sugar beet', 'Shepherds Purse',
       'Loose Silky-bent', 'Fat Hen', 'Small-flowered Cranesbill',
       'Loose Silky-bent', 'Common Chickweed',
       'Small-flowered Cranesbill', 'Small-flowered Cranesbill',
       'Common wheat', 'Small-flowered Cranesbill', 'Loose Silky-bent',
       'Loose Silky-bent', 'Maize', 'Loose Silky-bent', 'Fat Hen',
       'Small-flowered Cranesbill', 'Common Chickweed',
       'Small-flowered Cranesbill', 'Scentless Mayweed', 'Sugar beet',
       'Common Chickweed', 'Charlock', 'Black-grass', 'Black-grass',
       'Common Chickweed', 'Fat Hen', 'Common Chickweed', 'Common wheat',
       'Fat Hen', 'Fat Hen', 'Loose Silky-bent', 'Sugar beet',
       'Scentless Mayweed', 'Sugar beet', 'Fat Hen', 'Charlock',
       'Scentless Mayweed', 'Common Chickweed', 'Scentless Mayweed',
       'Scentless Mayweed', 'Fat Hen', 'Fat Hen',
       'Small-flowered Cranesbill', 'Loose Silky-bent',
       'Scentless Mayweed', 'Shepherds Purse', 'Maize', 'Cleavers',
       'Cleavers', 'Maize', 'Loose Silky-bent', 'Scentless Mayweed',
       'Shepherds Purse', 'Charlock', 'Fat Hen', 'Common Chickweed',
       'Fat Hen', 'Scentless Mayweed', 'Shepherds Purse',
       'Loose Silky-bent', 'Black-grass', 'Loose Silky-bent',
       'Sugar beet', 'Small-flowered Cranesbill', 'Scentless Mayweed',
       'Cleavers', 'Cleavers', 'Black-grass', 'Fat Hen', 'Charlock',
       'Cleavers', 'Loose Silky-bent', 'Cleavers',
       'Small-flowered Cranesbill', 'Charlock', 'Shepherds Purse',
       'Loose Silky-bent', 'Scentless Mayweed', 'Scentless Mayweed',
       'Common Chickweed', 'Loose Silky-bent', 'Loose Silky-bent',
       'Fat Hen', 'Maize', 'Fat Hen', 'Common wheat', 'Black-grass',
       'Scentless Mayweed', 'Scentless Mayweed', 'Loose Silky-bent',
       'Common wheat', 'Fat Hen', 'Loose Silky-bent',
       'Small-flowered Cranesbill', 'Cleavers', 'Sugar beet',
       'Scentless Mayweed', 'Charlock', 'Common Chickweed', 'Sugar beet',
       'Charlock', 'Common Chickweed', 'Fat Hen', 'Loose Silky-bent',
       'Small-flowered Cranesbill', 'Fat Hen', 'Loose Silky-bent',
       'Black-grass', 'Loose Silky-bent', 'Maize', 'Cleavers',
       'Small-flowered Cranesbill', 'Cleavers', 'Fat Hen', 'Fat Hen',
       'Loose Silky-bent', 'Fat Hen', 'Common Chickweed',
       'Scentless Mayweed', 'Loose Silky-bent', 'Loose Silky-bent',
       'Scentless Mayweed', 'Cleavers', 'Scentless Mayweed', 'Sugar beet',
       'Loose Silky-bent', 'Small-flowered Cranesbill',
       'Loose Silky-bent', 'Black-grass', 'Small-flowered Cranesbill',
       'Fat Hen', 'Black-grass', 'Shepherds Purse',
       'Small-flowered Cranesbill', 'Small-flowered Cranesbill',
```

Above "y_pred" array shows all predicted species (seedlings

All below project steps and tasks were achieved sucessfully

1. Import the libraries, load dataset, print shape of data, visualize the images in dataset. (5 Marks)
2. Data Pre-processing: (15 Marks) a. Normalization. b. Gaussian Blurring. c. Visualize data after pre-processing.
3. Make data compatible: (10 Marks) a. Convert labels to one-hot-vectors. b. Print the label for y_train[0]. c. Split the dataset into training, testing, and validation set. (Hint: First split images and labels into training and testing set with test_size = 0.3. Then further split test data into test and validation set with test_size = 0.5) d. Check the shape of data, Reshape data into shapes compatible with Keras models if it's not already. If it's already in the compatible shape, then comment in the notebook that it's already in compatible shape.
4. Building CNN: (15 Marks) a. Define layers. b. Set optimizer and loss function. (Use Adam optimizer and categorical crossentropy.)
5. Fit and evaluate model and print confusion matrix. (10 Marks)
6. Visualize predictions for x_test[2], x_test[3], x_test[33], x_test[36], x_test[59]. (5 Marks)