DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DEEP LEARNING LAB EVALUATION:
Plant Seedlings Classification

Submitted by:

Aditya Vashista

101703039

avashista_be17@thapar.edu

+91-9212291117

COE-02

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

PATIALA-147001, PUNJAB

TABLE OF CONTENTS

INTRODUCTION

We are provided with a training set and a test set of images of plant seedlings at various stages of grown. Each image has a filename that is its unique id. The dataset comprises 12 plant species. The goal of the is to create a classifier capable of determining a plant's species from a photo. The list of species is as follows:

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

File descriptions

- train.csv - the training set, with plant species organized by folder
- test.csv - the test set, we need to predict the species of each image
- sample_submission.csv - a sample submission file in the correct format

METHODOLOGY

****I did not use Google Colab for this project as the time to time there was deallocation of resources/ termination of session while training even though, I was I using TPU and training of the final model required at least 19 hours of training, which is not possible on colab. Hence I used Spyder for the same*

Since we are given only training data set of images, first we have to divide the set into two sub sets : one for training and other for testing/validation. Also we need to define a data generator for input categized image inputs to be trained on tensorflow model.

For this problem, after making 3 models with different accuracy criterion, activation function, data validation splitting and different number of hidden layers,I selected following model:

- Sequential Model with 5 CNN Layer (including input layer), 1 hidden ANN layer and 1 ANN output layer.
- Each CNN layer has padding, batch normalization, relu activation, maxPooling and dropout with rate of 0.25.
- Input Layer accepts the size of a 256x256 RGB image.
- Hidden ANN layer has batch normalization, relu activation and dropout with rate of 0.25.
- Output Layer has softmax activation for generating 12 outputs, giving probabilities of each 12 classes.
- Optimizer used: Adam
- Loss function: categorical_crossentropy
- Evaluation Metrics: categorical_accuracy & accuracy

A list containing names of all categories where the index of each category name corresponds to index of that class in output array obtained from model after prediction. We predict the category of input image sample by using numpy.argmax() which returns index of category having max probability.

Training was done over 38 epochs obtaining:

accuracy=categorical_accuracy= **0.8691**

val_accuracy=val_categorical_accuracy= **0.8247**

Kaggle Score after submitting final file: **0.82367**

PYTHON CODE/SCRIPT

```
"""Name: Aditya Vashista
Batch: Coe 2
Roll Number: 101703039
Problem: Plant Seedlings Classification"""

#importing libraries
import tensorflow as tf
from keras_preprocessing.image import ImageDataGenerator
from keras_preprocessing import image
from tensorflow.keras.layers import Dense, Dropout,Flatten, Conv2D
from tensorflow.keras.layers import BatchNormalization, Activation, MaxPooling2D
from tensorflow.keras.models import  Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import load_model
import numpy as np
import pandas as pd

#generating training and testing data of images available
datagen = ImageDataGenerator(rescale = 1./255,shear_range = 0.3,
                             zoom_range = 0.3,rotation_range=0.3,
                             width_shift_range=0.3,height_shift_range=0.3,
                             horizontal_flip = True,vertical_flip=True,
                             validation_split=0.3)

train_set=datagen.flow_from_directory("C:\\Users\\aditya\\Downloads\\train",
                              target_size=(256,256),batch_size=32,class_mode='categorical',
                              subset='training')
test_set=datagen.flow_from_directory("C:\\Users\\aditya\\Downloads\\train",
                              target_size=(256,256),batch_size=32,class_mode='categorical',
                              subset='validation')

#Creating Sequential CNN + ANN model
#For each CNN layer batchnormalization,drop out and max pooling is also appliead
```

```
model=Sequential()
#CNN LAYER 1: INPUT LAYER
model.add(Conv2D(64,(3,3),padding='same',input_shape=(256,256,3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=2,strides=2,padding='valid'))
model.add(Dropout(0.25))
#CNN LAYER 2
model.add(Conv2D(128,(5,5),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
#CNN LAYER 3
model.add(Conv2D(256,(3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=2,strides=2,padding='valid'))
model.add(Dropout(0.25))
#CNN LAYER 4
model.add(Conv2D(512,(3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=2,strides=2,padding='valid'))
model.add(Dropout(0.25))
#CNN LAYER 5
model.add(Conv2D(512,(3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=2,strides=2,padding='valid'))
model.add(Dropout(0.25))
#FLATTENING
model.add(Flatten())
#ANN LAYER 1
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
```

```
model.add(Dropout(0.25))
#OUTPUT LAYER(ANN-2) with 12 outputs
model.add(Dense(12,activation='softmax'))
#creating optimizer
opt=Adam(lr=0.0005)
#model compilation
model.compile(optimizer=opt,loss='categorical_crossentropy',metrics=['categorical_accuracy','accuracy'])
#model summary
model.summary()



#Total epochs used for training=38
#Epochs were devided as: 38=16+8+10+4 with different validation splits:(2,3)

#MODEL TRAINING
#epochs=4 //last epocs set
epochs=38
steps_per_epoch=train_set.n//train_set.batch_size
test_steps=test_set.n//test_set.batch_size

model.fit(x=train_set,epochs=epochs,validation_data=test_set,validation_steps=test_steps)

# saving model
model.save("model.h5")
del model
#loading model
model=load_model('model.h5')

#testing and verification of various images manually to check accuracy
testImage1=image.load_img('train//Small-flowered Cranesbill//0b26e2d09.png',target_size=(256,256))
testImage1=image.img_to_array(testImage1)
testImage1=np.expand_dims(testImage1,0)
print(model.predict(testImage1))
print(np.argmax(model.predict(testImage1)))



#TESTING
```

```
#DEFINING 12 classes as per requirements
classes=['Black-grass','Charlock','Cleavers','Common Chickweed','Common wheat','Fat Hen',
        'Loose Silky-bent','Maize','Scentless Mayweed','Shepherds Purse',
        'Small-flowered Cranesbill','Sugar beet']

#reading sample file for testing
sam=pd.read_csv('sample_submission.csv')
#dataframe to be stored in output file
df=pd.DataFrame(columns=['file','species'])

#computing output
for i in range(len(sam)):
    testImage=image.load_img('test//'+str(sam.iloc[i][0]),target_size=(256,256)) #loading a single image
    #necessary steps to convert images into suitable input format for model
    testImage=image.img_to_array(testImage)
    testImage=np.expand_dims(testImage,0)
    #getting index of max output from list of 12 outputs from the model after prediction
    pred=np.argmax(model.predict(testImage))
    #assigning corresponding class
    pred=classes[pred]
    #adding the output with image name in dataframe
    df=df.append({'file':sam.iloc[i][0],'species':pred},ignore_index=True)

#saving output to new file
df.to_csv('submission.csv',index=False)
```

OUTPUTS

Model summary after compilation:

```
    ...: model.summary()
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_24 (Conv2D)           (None, 256, 256, 64)      1792
_________________________________________________________________
batch_normalization_32 (Batc (None, 256, 256, 64)      256
_________________________________________________________________
activation_32 (Activation)   (None, 256, 256, 64)      0
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 128, 128, 64)      0
_________________________________________________________________
dropout_32 (Dropout)         (None, 128, 128, 64)      0
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 128, 128, 128)     204928
_________________________________________________________________
batch_normalization_33 (Batc (None, 128, 128, 128)     512
_________________________________________________________________
activation_33 (Activation)   (None, 128, 128, 128)     0
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 64, 64, 128)       0
_________________________________________________________________
dropout_33 (Dropout)         (None, 64, 64, 128)       0
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 64, 64, 256)       295168
_________________________________________________________________
batch_normalization_34 (Batc (None, 64, 64, 256)       1024
_________________________________________________________________
activation_34 (Activation)   (None, 64, 64, 256)       0
_________________________________________________________________
max_pooling2d_26 (MaxPooling (None, 32, 32, 256)       0
_________________________________________________________________
dropout_34 (Dropout)         (None, 32, 32, 256)       0
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 32, 32, 512)       1180160
```

```
dropout_34 (Dropout)         (None, 32, 32, 256)       0
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 32, 32, 512)       1180160
_________________________________________________________________
batch_normalization_35 (Batc (None, 32, 32, 512)       2048
_________________________________________________________________
activation_35 (Activation)   (None, 32, 32, 512)       0
_________________________________________________________________
max_pooling2d_27 (MaxPooling (None, 16, 16, 512)       0
_________________________________________________________________
dropout_35 (Dropout)         (None, 16, 16, 512)       0
_________________________________________________________________
conv2d_28 (Conv2D)           (None, 16, 16, 512)       2359808
_________________________________________________________________
batch_normalization_36 (Batc (None, 16, 16, 512)       2048
_________________________________________________________________
activation_36 (Activation)   (None, 16, 16, 512)       0
_________________________________________________________________
max_pooling2d_28 (MaxPooling (None, 8, 8, 512)         0
_________________________________________________________________
dropout_36 (Dropout)         (None, 8, 8, 512)         0
_________________________________________________________________
flatten_5 (Flatten)          (None, 32768)             0
_________________________________________________________________
dense_13 (Dense)             (None, 256)               8388864
_________________________________________________________________
batch_normalization_37 (Batc (None, 256)               1024
_________________________________________________________________
activation_37 (Activation)   (None, 256)               0
_________________________________________________________________
dropout_37 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_14 (Dense)             (None, 12)                3084
=================================================================
Total params: 12,440,716
Trainable params: 12,437,260
Non-trainable params: 3,456
```

Output while training in different batches of epocs:

```
Found 3330 images belonging to 12 classes.
Found 1420 images belonging to 12 classes.
Epoch 1/16
105/105 [==============================] - 1643s 16s/step - loss: 2.4379 - accuracy: 0.2024 -
categorical_accuracy: 0.2024 - val_loss: 3.3982 - val_accuracy: 0.1371 - val_categorical_accuracy:
0.1371
Epoch 2/16
105/105 [==============================] - 1693s 16s/step - loss: 1.8834 - accuracy: 0.3465 -
categorical_accuracy: 0.3465 - val_loss: 6.8800 - val_accuracy: 0.1385 - val_categorical_accuracy:
0.1385
Epoch 3/16
105/105 [==============================] - 1677s 16s/step - loss: 1.6653 - accuracy: 0.4264 -
categorical_accuracy: 0.4264 - val_loss: 6.6332 - val_accuracy: 0.1392 - val_categorical_accuracy:
0.1329
Epoch 4/16
105/105 [==============================] - 1643s 16s/step - loss: 1.5296 - accuracy: 0.4736 -
categorical_accuracy: 0.4736 - val_loss: 5.8745 - val_accuracy: 0.1229 - val_categorical_accuracy:
0.1229
Epoch 5/16
105/105 [==============================] - 1625s 15s/step - loss: 1.5115 - accuracy: 0.4661 -
categorical_accuracy: 0.4661 - val_loss: 4.5862 - val_accuracy: 0.1534 - val_categorical_accuracy:
```

IPython console | History

ready | Kite: not running | conda: base (Python 3.8.3) | Line 26, Col 2 | UTF-8 | CRLF | RW | Mem 45% | CPU 1% | 19:15

```
,target_size=(256
                                    In [45]: model.fit(x=train_set,epochs=epochs,validation_data=test_set,validation_steps=test_steps)
                                    Epoch 1/10
                                    105/105 [==============================] - 1515s 14s/step - loss: 0.5007 - categorical_accuracy:
                                    0.8237 - accuracy: 0.8237 - val_loss: 0.7904 - val_categorical_accuracy: 0.7358 - val_accuracy:
                                    0.7358
                                    Epoch 2/10
                                    105/105 [==============================] - 1497s 14s/step - loss: 0.4703 - categorical_accuracy:
                                    0.8357 - accuracy: 0.8357 - val_loss: 0.9949 - val_categorical_accuracy: 0.6875 - val_accuracy:
                                    0.6875
                                    Epoch 3/10
                                    105/105 [==============================] - 1496s 14s/step - loss: 0.4989 - categorical_accuracy:
                                    0.8261 - accuracy: 0.8261 - val_loss: 0.9572 - val_categorical_accuracy: 0.6903 - val_accuracy:
                                    0.6903
                                    Epoch 4/10
                                    105/105 [==============================] - 1504s 14s/step - loss: 0.4448 - categorical_accuracy:
                                    0.8423 - accuracy: 0.8423 - val_loss: 0.8356 - val_categorical_accuracy: 0.7330 - val_accuracy:
                                    0.7330
                                    Epoch 5/10
                                    105/105 [==============================] - 1504s 14s/step - loss: 0.5224 - categorical_accuracy:
                                    0.8198 - accuracy: 0.8198 - val_loss: 1.9768 - val_categorical_accuracy: 0.4808 - val_accuracy:
                                    0.4808
```

```
Epoch 5/10
105/105 [==============================] - 1504s 14s/step - loss: 0.5224 - categorical_accuracy:
0.8198 - accuracy: 0.8198 - val_loss: 1.9768 - val_categorical_accuracy: 0.4808 - val_accuracy:
0.4808
Epoch 6/10
105/105 [==============================] - 1506s 14s/step - loss: 0.4475 - categorical_accuracy:
0.8312 - accuracy: 0.8312 - val_loss: 0.5965 - val_categorical_accuracy: 0.7962 - val_accuracy:
0.7962
Epoch 7/10
105/105 [==============================] - 1521s 14s/step - loss: 0.4472 - categorical_accuracy:
0.8420 - accuracy: 0.8420 - val_loss: 2.1002 - val_categorical_accuracy: 0.4659 - val_accuracy:
0.4659
Epoch 8/10
105/105 [==============================] - 1500s 14s/step - loss: 0.4411 - categorical_accuracy:
0.8535 - accuracy: 0.8535 - val_loss: 1.7361 - val_categorical_accuracy: 0.4844 - val_accuracy:
0.4844
Epoch 9/10
105/105 [==============================] - 1609s 15s/step - loss: 0.4169 - categorical_accuracy:
0.8520 - accuracy: 0.8520 - val_loss: 0.8178 - val_categorical_accuracy: 0.7365 - val_accuracy:
0.7365
```

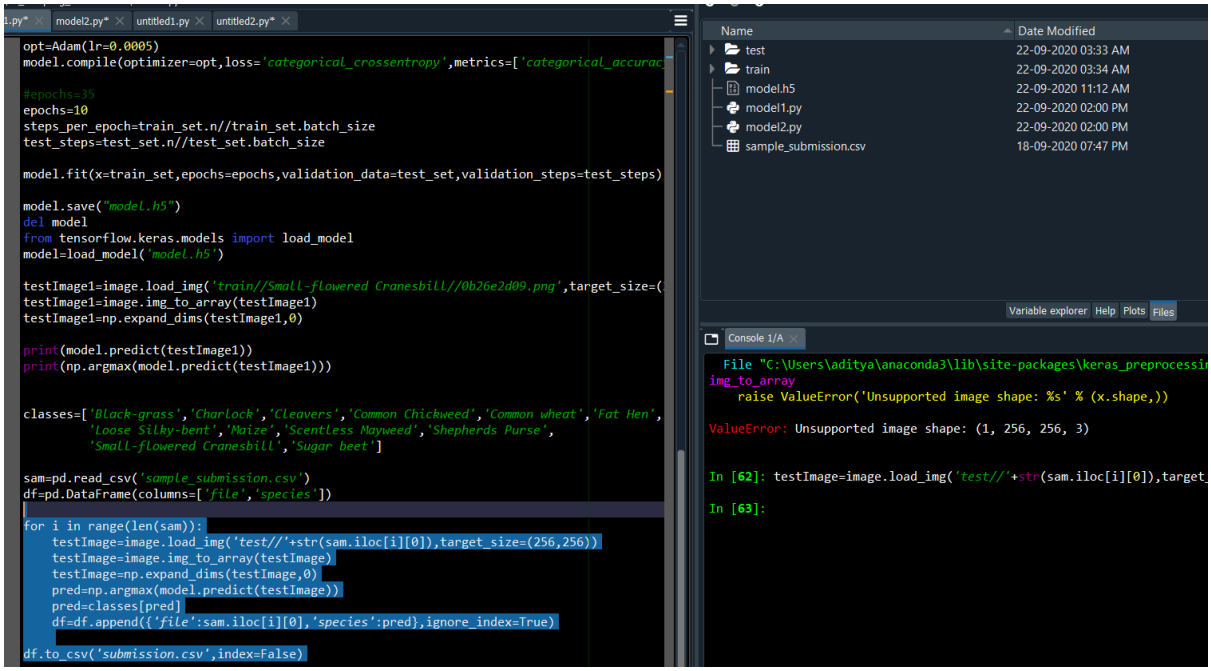
```
Found 3330 images belonging to 12 classes.
Found 1420 images belonging to 12 classes.
Epoch 1/4
105/105 [==============================] - 1643s 16s/step - loss: 0.4146 - accuracy: 0.8636 -
categorical_accuracy: 0.8636 - val_loss: 0.5031 - val_accuracy: 0.8051 - val_categorical_accuracy: 0.8051
Epoch 2/4
105/105 [==============================] - 1693s 16s/step - loss: 0.4063 - accuracy: 0.8598 -
categorical_accuracy: 0.8598 - val_loss: 0.3407 - val_accuracy: 0.8293 - val_categorical_accuracy: 0.8293
Epoch 3/4
105/105 [==============================] - 1677s 16s/step - loss: 0.3997 - accuracy: 0.8726 -
categorical_accuracy: 0.8726 - val_loss: 0.4921 - val_accuracy: 0.8196 - val_categorical_accuracy: 0.8196
Epoch 4/4
105/105 [==============================] - 1643s 16s/step - loss: 0.4009 - accuracy: 0.8691-
categorical_accuracy: 0.8691 - val_loss: 0.3250 - val_accuracy: 0.8247 - val_categorical_accuracy: 0.8247
```

Output while testing model on different labeled class images :

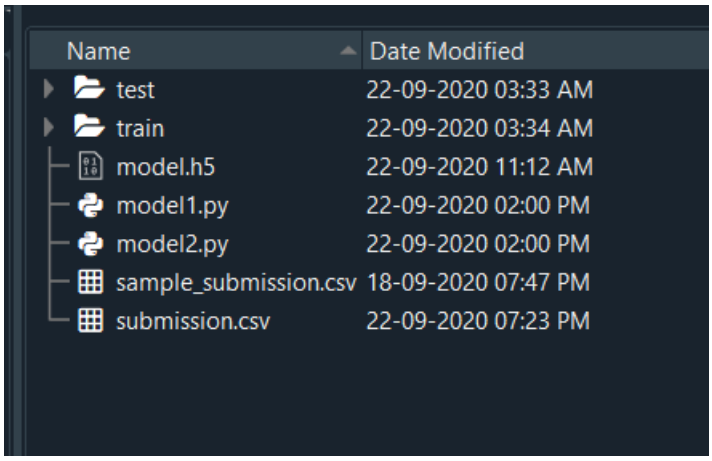
```
...: testImage1=image.img_to_array(testImage1)
...: testImage1=np.expand_dims(testImage1,0)

In [47]: print(model.predict(testImage1))
...: print(np.argmax(model.predict(testImage1)))
[[0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]]
3
```

```
...: testImage1=image.img_to_array(testImage1)
...: testImage1=np.expand_dims(testImage1,0)
...:
...: print(model.predict(testImage1))
...: print(np.argmax(model.predict(testImage1)))
[[0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]]
7
```

File Directory before execution of last block of code on test data:



File Directory after execution of last block of code on test data:

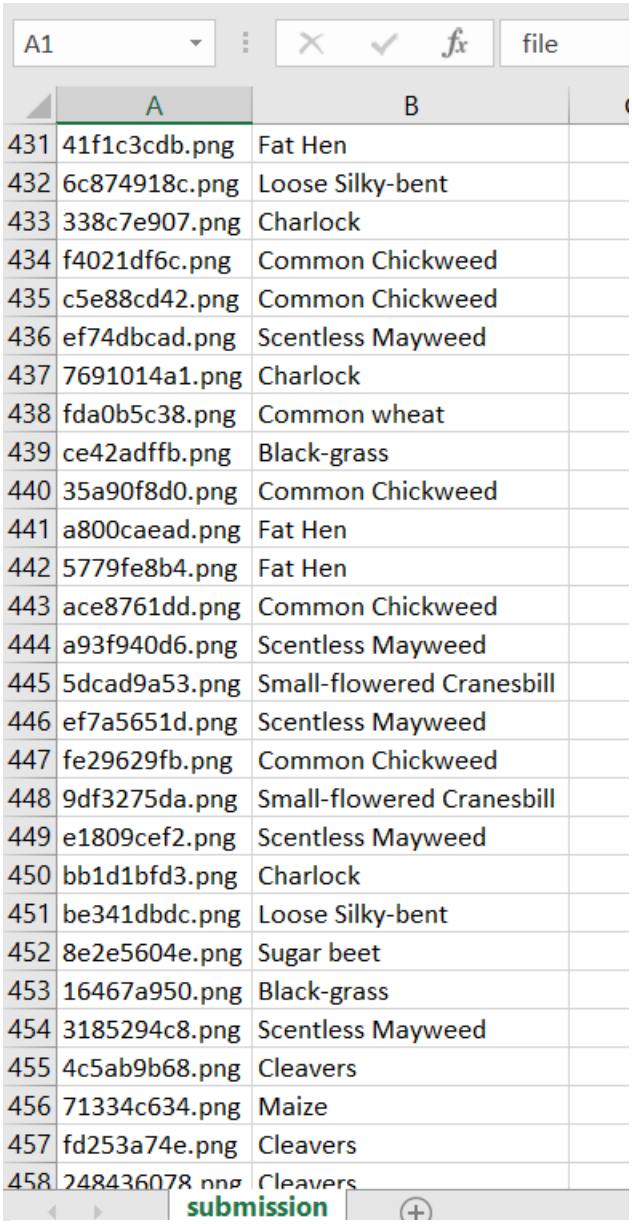


Output in submission.csv after whole execution of python file:

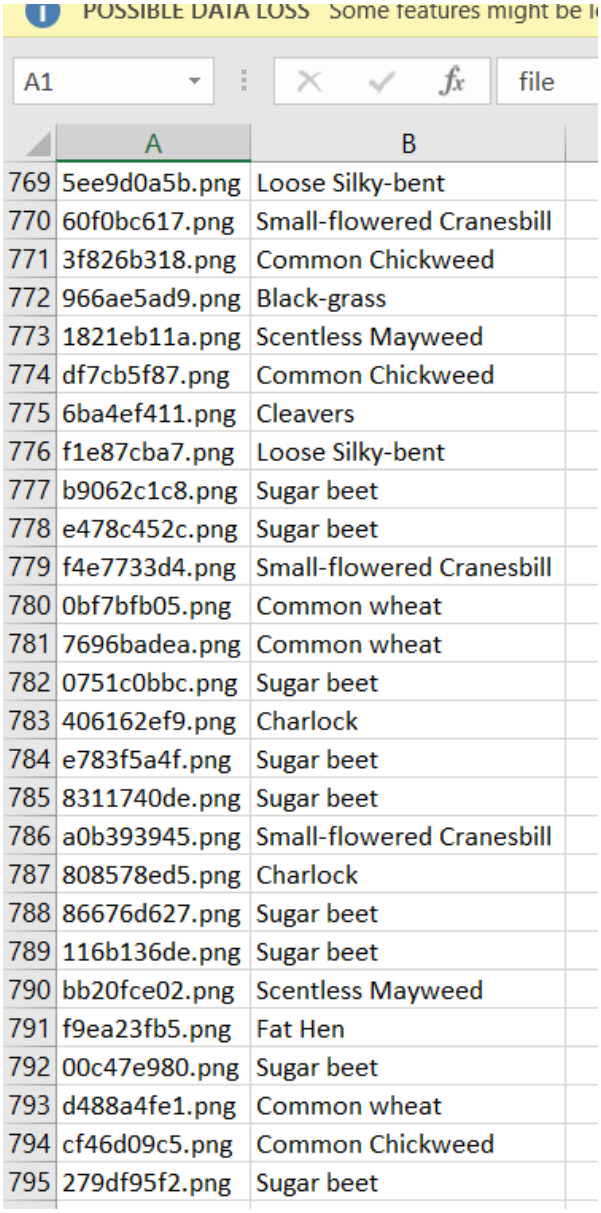
	A	B
1	file	species
2	26e7ae885.png	Fat Hen
3	e80a259c5.png	Small-flowered Cranesbill
4	3f64c2c1b.png	Common wheat
5	1312065a5.png	Small-flowered Cranesbill
6	47b7d8e17.png	Small-flowered Cranesbill
7	c0f5d9ac8.png	Small-flowered Cranesbill
8	ab0f67743.png	Black-grass
9	da9ef7858.png	Charlock
10	5f04aed97.png	Small-flowered Cranesbill
11	fba8fc78a.png	Black-grass
12	632156793.png	Cleavers
13	4bbfd1e05.png	Cleavers
14	74fd477eb.png	Common Chickweed
15	060450d79.png	Common Chickweed
16	cae684f8f.png	Charlock
17	86f08e6d1.png	Fat Hen
18	a8c8a1db0.png	Shepherds Purse
19	456d507c0.png	Cleavers
20	f48916a8c.png	Scentless Mayweed
21	dc55449b2.png	Sugar beet
22	52dc7a4d6.png	Common Chickweed
23	862b8e7a0.png	Sugar beet
24	5eb9c26a6.png	Black-grass
25	20817c846.png	Shepherds Purse
26	bf3924a57.png	Common Chickweed
27	f6d250856.png	Shepherds Purse
28	3dd52bd2a.png	Small-flowered Cranesbill

	A	B
297	5817b766d.png	Shepherds Purse
298	f4234cf4f.png	Fat Hen
299	ba3ce6b3e.png	Scentless Mayweed
300	a006a475c.png	Shepherds Purse
301	87608f7aa.png	Charlock
302	80e299ae9.png	Shepherds Purse
303	c7eb96871.png	Common wheat
304	8db450ce3.png	Common Chickweed
305	82b5f4d33.png	Shepherds Purse
306	79dafec17.png	Black-grass
307	b3e08b037.png	Fat Hen
308	b687160f5.png	Small-flowered Cranesbill
309	23e480e64.png	Shepherds Purse
310	c75a82234.png	Scentless Mayweed
311	0086a6340.png	Common Chickweed
312	539961189.png	Common Chickweed
313	dabea05f4.png	Black-grass
314	f1f7c833f.png	Charlock
315	ce15eee52.png	Charlock
316	4392d93cf.png	Fat Hen
317	71f5323c5.png	Black-grass
318	79d93bc96.png	Black-grass
319	25fa8d109.png	Common wheat
320	800a8c17e.png	Fat Hen
321	e4d5ec761.png	Common wheat
322	0caeda5df.png	Common wheat
323	16357b436.png	Loose Silky-bent
324	446f7da01.png	Black-grass

submission

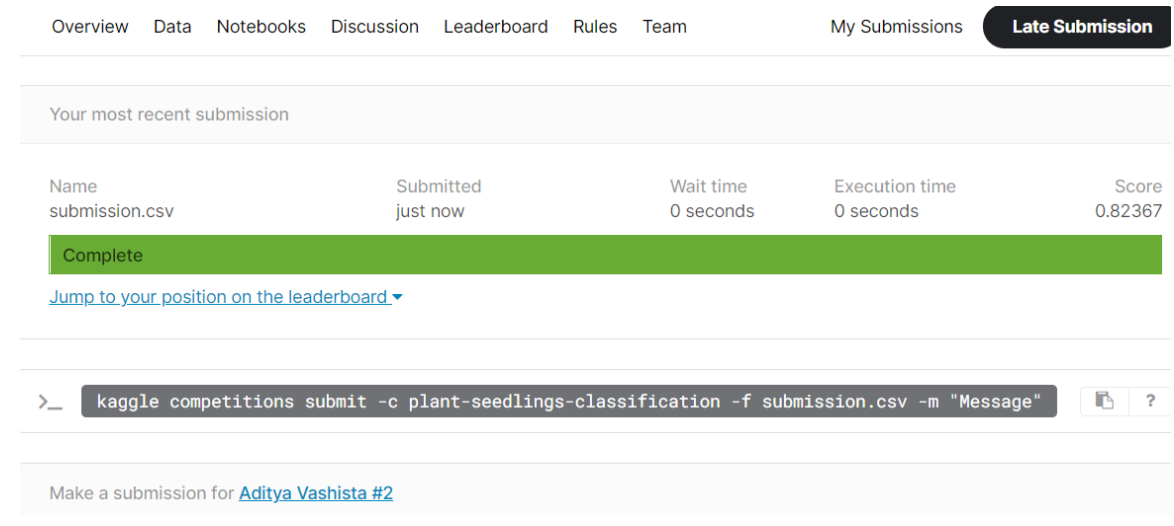


	A	B
431	41f1c3cdb.png	Fat Hen
432	6c874918c.png	Loose Silky-bent
433	338c7e907.png	Charlock
434	f4021df6c.png	Common Chickweed
435	c5e88cd42.png	Common Chickweed
436	ef74dbcad.png	Scentless Mayweed
437	7691014a1.png	Charlock
438	fda0b5c38.png	Common wheat
439	ce42adffb.png	Black-grass
440	35a90f8d0.png	Common Chickweed
441	a800caead.png	Fat Hen
442	5779fe8b4.png	Fat Hen
443	ace8761dd.png	Common Chickweed
444	a93f940d6.png	Scentless Mayweed
445	5dcad9a53.png	Small-flowered Cranesbill
446	ef7a5651d.png	Scentless Mayweed
447	fe29629fb.png	Common Chickweed
448	9df3275da.png	Small-flowered Cranesbill
449	e1809cef2.png	Scentless Mayweed
450	bb1d1bfd3.png	Charlock
451	be341dbdc.png	Loose Silky-bent
452	8e2e5604e.png	Sugar beet
453	16467a950.png	Black-grass
454	3185294c8.png	Scentless Mayweed
455	4c5ab9b68.png	Cleavers
456	71334c634.png	Maize
457	fd253a74e.png	Cleavers
458	248436078.png	Cleavers



	A	B
769	5ee9d0a5b.png	Loose Silky-bent
770	60f0bc617.png	Small-flowered Cranesbill
771	3f826b318.png	Common Chickweed
772	966ae5ad9.png	Black-grass
773	1821eb11a.png	Scentless Mayweed
774	df7cb5f87.png	Common Chickweed
775	6ba4ef411.png	Cleavers
776	f1e87cba7.png	Loose Silky-bent
777	b9062c1c8.png	Sugar beet
778	e478c452c.png	Sugar beet
779	f4e7733d4.png	Small-flowered Cranesbill
780	0bf7bfb05.png	Common wheat
781	7696badea.png	Common wheat
782	0751c0bbc.png	Sugar beet
783	406162ef9.png	Charlock
784	e783f5a4f.png	Sugar beet
785	8311740de.png	Sugar beet
786	a0b393945.png	Small-flowered Cranesbill
787	808578ed5.png	Charlock
788	86676d627.png	Sugar beet
789	116b136de.png	Sugar beet
790	bb20fce02.png	Scentless Mayweed
791	f9ea23fb5.png	Fat Hen
792	00c47e980.png	Sugar beet
793	d488a4fe1.png	Common wheat
794	cf46d09c5.png	Common Chickweed
795	279df95f2.png	Sugar beet

Best Submission Score on Kaggle:



All submissions on Kaggle: