CS5613

Neural Networks

Assignment 2 - CNN Group Assignment

Group Name : **199342EC.G.Madage-199355VW.Perera-199361KR.Senanayaka**

199342E - C.G. Madage

199355V - Wishwa Perera

199361K- M.D.R.N. Senanayake

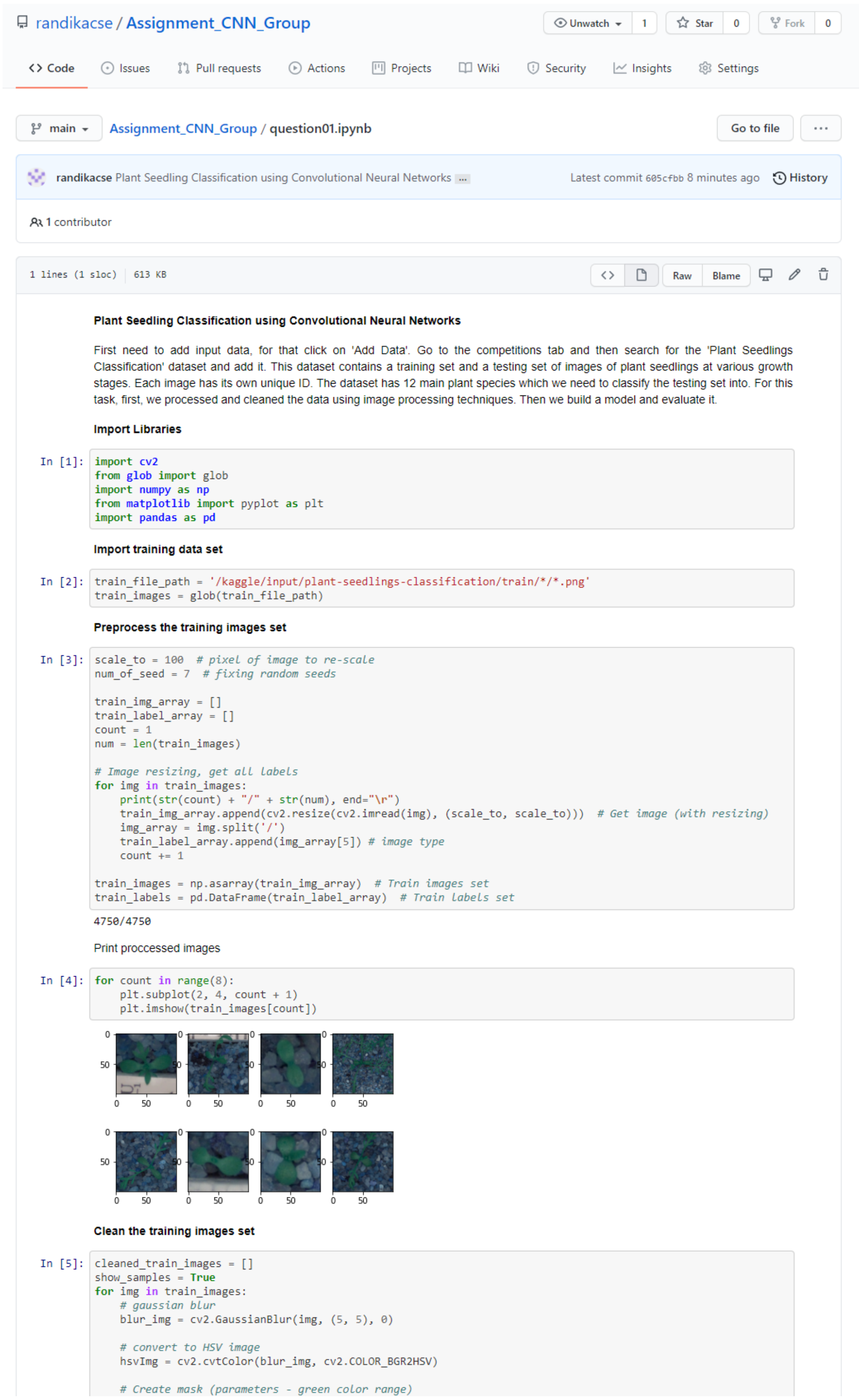
MSc in Computer Science 2019

Department of Computer Science and Engineering

University of Moratuwa

Q1) Answer for Question1 is published in the [Github](https://github.com/randikacse/Assignment_CNN_Group/blob/main/question01.ipynb)

https://github.com/randikacse/Assignment_CNN_Group/blob/main/question01.ipynb


randikacse / Assignment_CNN_Group
Unwatch 1
Star 0
Fork 0
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
main
Assignment_CNN_Group / question01.ipynb
Go to file
randikacse Plant Seedling Classification using Convolutional Neural Networks
Latest commit 605cfbb 8 minutes ago
History
1 contributor
1 lines (1 sloc) 613 KB
Raw
Blame

Plant Seedling Classification using Convolutional Neural Networks

First need to add input data, for that click on 'Add Data'. Go to the competitions tab and then search for the 'Plant Seedlings Classification' dataset and add it. This dataset contains a training set and a testing set of images of plant seedlings at various growth stages. Each image has its own unique ID. The dataset has 12 main plant species which we need to classify the testing set into. For this task, first, we processed and cleaned the data using image processing techniques. Then we build a model and evaluate it.

Import Libraries

In [1]:
```
import cv2
from glob import glob
import numpy as np
from matplotlib import pyplot as plt
import pandas as pd
```

Import training data set

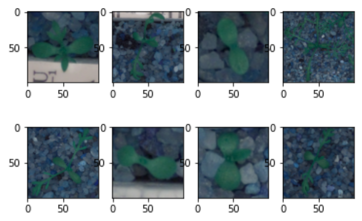
In [2]:
```
train_file_path = '/kaggle/input/plant-seedlings-classification/train/*/*.png'
train_images = glob(train_file_path)
```

Preprocess the training images set

In [3]:
```
scale_to = 100  # pixel of image to re-scale
num_of_seed = 7  # fixing random seeds

train_img_array = []
train_label_array = []
count = 1
num = len(train_images)

# Image resizing, get all labels
for img in train_images:
    print(str(count) + "/" + str(num), end="\r")
    train_img_array.append(cv2.resize(cv2.imread(img), (scale_to, scale_to)))  # Get image (with resizing)
    img_array = img.split('/')
    train_label_array.append(img_array[5]) # image type
    count += 1

train_images = np.asarray(train_img_array)  # Train images set
train_labels = pd.DataFrame(train_label_array)  # Train labels set
```
```
4750/4750
```

Print proccessed images

In [4]:
```
for count in range(8):
    plt.subplot(2, 4, count + 1)
    plt.imshow(train_images[count])
```

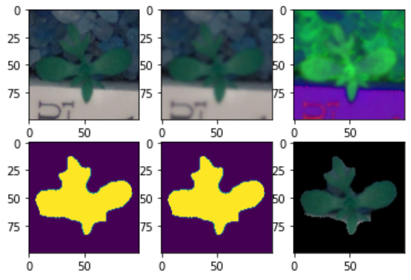


Clean the training images set

In [5]:
```
cleaned_train_images = []
show_samples = True
for img in train_images:
    # gaussian blur
    blur_img = cv2.GaussianBlur(img, (5, 5), 0)

    # convert to HSV image
    hsvImg = cv2.cvtColor(blur_img, cv2.COLOR_BGR2HSV)

    # Create mask (parameters - green color range)
```

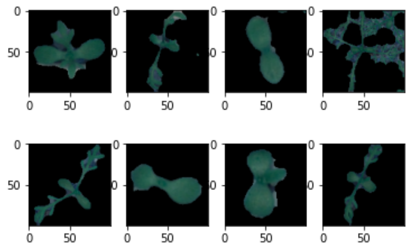
Clean the training images set

```
In [5]: cleaned_train_images = []
        show_samples = True
        for img in train_images:
            # gaussian blur
            blur_img = cv2.GaussianBlur(img, (5, 5), 0)

            # convert to HSV image
            hsvImg = cv2.cvtColor(blur_img, cv2.COLOR_BGR2HSV)

            # Create mask (parameters - green color range)
            lower_green = (25, 41, 50)
            upper_green = (74, 255, 255)
            mask = cv2.inRange(hsvImg, lower_green, upper_green)
            kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (11, 11))
            mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)

            # Create bool mask
            bMask = mask > 0

            # Apply the mask
            cleaned = np.zeros_like(img, np.uint8)  # Create empty image
            cleaned[bMask] = img[bMask]  # Apply boolean mask to the origin image

            cleaned_train_images.append(cleaned)  # Append image without backgroung

            # Show examples
            if show_samples:
                plt.subplot(2, 3, 1); plt.imshow(img)  # Show the original image
                plt.subplot(2, 3, 2); plt.imshow(blur_img)  # Blur image
                plt.subplot(2, 3, 3); plt.imshow(hsvImg)  # HSV image
                plt.subplot(2, 3, 4); plt.imshow(mask)  # Mask
                plt.subplot(2, 3, 5); plt.imshow(bMask)  # Boolean mask
                plt.subplot(2, 3, 6); plt.imshow(cleaned)  # Image without background
                show_samples = False

        cleaned_train_img = np.asarray(cleaned_train_images)
```



Print cleaned images

```
In [6]: for i in range(8):
            plt.subplot(2, 4, i + 1)
            plt.imshow(cleaned_train_img[i])
```



```
In [7]: cleaned_train_imgs = cleaned_train_img / 255
```

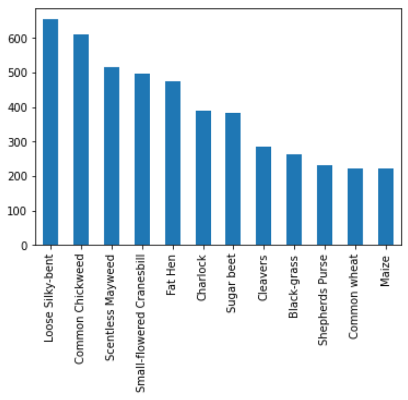
Identify the Classes

```
In [8]: from keras.utils import np_utils
        from sklearn import preprocessing
        import matplotlib.pyplot as plt

        # Encode labels and create classes
        le = preprocessing.LabelEncoder()
        le.fit(train_labels[0])
        print("Classes: " + str(le.classes_))
        encode_train_labels = le.transform(train_labels[0])

        # Make labels categorical
        cleaned_train_label = np_utils.to_categorical(encode_train_labels)
        num_classes = cleaned_train_label.shape[1]
        print("num of classes: " + str(num_classes))

        # Plot of label types numbers
        train_labels[0].value_counts().plot(kind='bar')
```

```
Classes: ['Black-grass' 'Charlock' 'Cleavers' 'Common Chickweed' 'Common wheat'
 'Fat Hen' 'Loose Silky-bent' 'Maize' 'Scentless Mayweed'
 'Shepherds Purse' 'Small-flowered Cranesbill' 'Sugar beet']
num of classes: 12
```

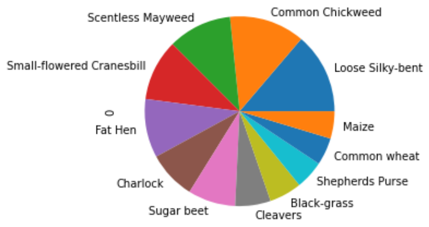
```
Out[8]: <AxesSubplot:>
```

```
Out[8]: <AxesSubplot:>
```



```
In [9]: train_labels[0].value_counts().plot(kind='pie')
```

```
Out[9]: <AxesSubplot:ylabel='0'>
```



Split train and test data set

```
In [10]: from sklearn.model_selection import train_test_split

trainX, testX, trainY, testY = train_test_split(cleaned_train_imgs, cleaned_train_label,
                                                test_size=0.2, random_state=num_of_seed,
                                                stratify = cleaned_train_label)
```

```
In [11]: from keras.preprocessing.image import ImageDataGenerator
data_gen = ImageDataGenerator(
        rotation_range=180,  # randomly rotate images in the range
        zoom_range = 0.1, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontally
        height_shift_range=0.1,  # randomly shift images vertically
        horizontal_flip=True,  # randomly flip images horizontally
        vertical_flip=True  # randomly flip images vertically
    )
data_gen.fit(trainX)
```

Create the Model

In [12]:
```
import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers import BatchNormalization

numpy.random.seed(num_of_seed)  # num_of_seed

model = Sequential()

model.add(Conv2D(filters=64, kernel_size=(5, 5), input_shape=(scale_to, scale_to, 3), activation='relu'))
model.add(BatchNormalization(axis=3))
model.add(Conv2D(filters=64, kernel_size=(5, 5), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(BatchNormalization(axis=3))
model.add(Dropout(0.1))

model.add(Conv2D(filters=128, kernel_size=(5, 5), activation='relu'))
model.add(BatchNormalization(axis=3))
model.add(Conv2D(filters=128, kernel_size=(5, 5), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(BatchNormalization(axis=3))
model.add(Dropout(0.1))

model.add(Conv2D(filters=256, kernel_size=(5, 5), activation='relu'))
model.add(BatchNormalization(axis=3))
model.add(Conv2D(filters=256, kernel_size=(5, 5), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(BatchNormalization(axis=3))
model.add(Dropout(0.1))

model.add(Flatten())

model.add(Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(num_classes, activation='softmax'))

model.summary()

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

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 96, 96, 64)        4864      
_________________________________________________________________
batch_normalization (BatchNo (None, 96, 96, 64)        256       
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 92, 92, 64)        102464    
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 46, 46, 64)        0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 46, 46, 64)        256       
_________________________________________________________________
dropout (Dropout)            (None, 46, 46, 64)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 42, 42, 128)       204928    
_________________________________________________________________
batch_normalization_2 (Batch (None, 42, 42, 128)       512       
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 38, 38, 128)       409728    
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 19, 19, 128)       0         
_________________________________________________________________
batch_normalization_3 (Batch (None, 19, 19, 128)       512       
_________________________________________________________________
dropout_1 (Dropout)          (None, 19, 19, 128)       0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 256)       819456    
_________________________________________________________________
batch_normalization_4 (Batch (None, 15, 15, 256)       1024      
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 11, 11, 256)       1638656   
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 256)         0         
_________________________________________________________________
batch_normalization_5 (Batch (None, 5, 5, 256)         1024      
_________________________________________________________________
dropout_2 (Dropout)          (None, 5, 5, 256)         0         
_________________________________________________________________
flatten (Flatten)            (None, 6400)              0         
_________________________________________________________________
dense (Dense)                (None, 256)               1638656   
_________________________________________________________________
batch_normalization_6 (Batch (None, 256)               1024      
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0         
```

In [13]:
```
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, CSVLogger

# learning rate reduction
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
                                            patience=3,
                                            verbose=1,
                                            factor=0.4,
                                            min_lr=0.00001)

# add check points
file_path = "/kaggle/working/weights.best_{epoch:02d}-{val_accuracy:.2f}.hdf5"

check_point = ModelCheckpoint(file_path, monitor='val_accuracy',
                              verbose=1, save_best_only=True, mode='max')

file_path = "/kaggle/working/weights.last_auto.hdf5"
checkpoint_all = ModelCheckpoint(file_path, monitor='val_accuracy',
                                 verbose=1, save_best_only=False, mode='max')

# all callbacks
callbacks_list = [check_point, learning_rate_reduction, checkpoint_all]

# fit model
hist = model.fit_generator(data_gen.flow(trainX, trainY, batch_size=75), epochs=30, validation_data=(testX, t
estY), callbacks=callbacks_list)
```

```
/opt/conda/lib/python3.7/site-packages/tensorflow/python/keras/engine/training.py:1844: UserWarning: `Model.f
it_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports g
enerators.
  warnings.warn('`Model.fit_generator` is deprecated and '
```

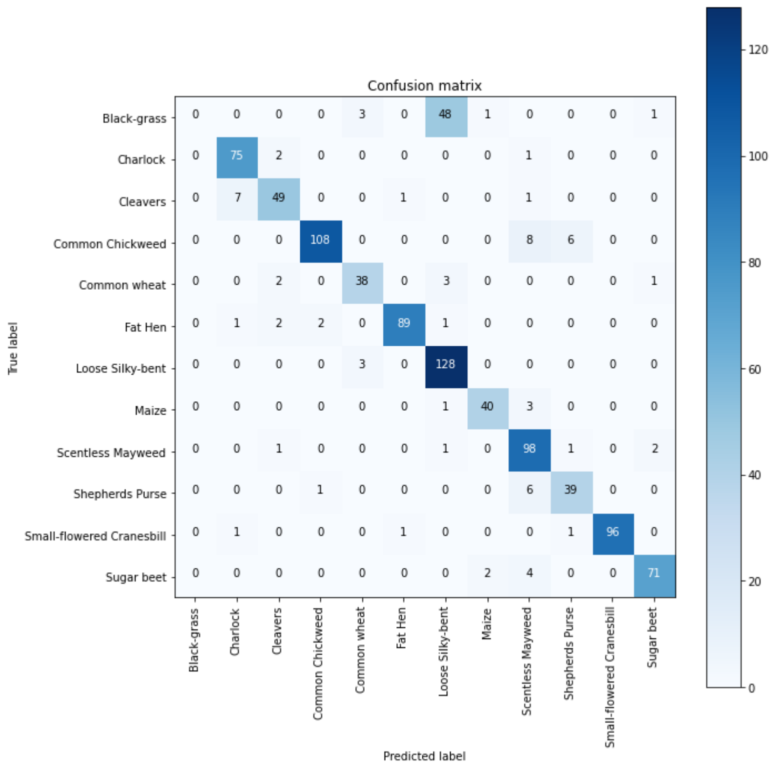
```
Epoch 1/30
51/51 [==============================] - 75s 1s/step - loss: 3.0351 - accuracy: 0.2073 - val_loss: 25.1081 -
val_accuracy: 0.0463

Epoch 00001: val_accuracy improved from -inf to 0.04632, saving model to /kaggle/working/weights.best_01-0.0
5.hdf5

Epoch 00001: saving model to /kaggle/working/weights.last_auto.hdf5
Epoch 2/30
51/51 [==============================] - 57s 1s/step - loss: 2.0388 - accuracy: 0.3809 - val_loss: 23.2107 -
val_accuracy: 0.0463

Epoch 00002: val_accuracy did not improve from 0.04632

Epoch 00002: saving model to /kaggle/working/weights.last_auto.hdf5
Epoch 3/30
51/51 [==============================] - 57s 1s/step - loss: 1.5278 - accuracy: 0.4813 - val_loss: 25.1781 -
val_accuracy: 0.0611

Epoch 00003: val_accuracy improved from 0.04632 to 0.06105, saving model to /kaggle/working/weights.best_03-
0.06.hdf5

Epoch 00003: saving model to /kaggle/working/weights.last_auto.hdf5
Epoch 4/30
51/51 [==============================] - 57s 1s/step - loss: 1.4965 - accuracy: 0.4924 - val_loss: 27.5217 -
val_accuracy: 0.0611

Epoch 00004: val_accuracy did not improve from 0.06105

Epoch 00004: saving model to /kaggle/working/weights.last_auto.hdf5
Epoch 5/30
51/51 [==============================] - 57s 1s/step - loss: 1.2689 - accuracy: 0.5660 - val_loss: 26.1744 -
val_accuracy: 0.0611

Epoch 00005: val_accuracy did not improve from 0.06105

Epoch 00005: saving model to /kaggle/working/weights.last_auto.hdf5
Epoch 6/30
51/51 [==============================] - 57s 1s/step - loss: 1.1779 - accuracy: 0.5909 - val_loss: 33.1222 -
val_accuracy: 0.0611

Epoch 00006: val_accuracy did not improve from 0.06105

Epoch 00006: ReduceLROnPlateau reducing learning rate to 0.0004000000189989805.

Epoch 00006: saving model to /kaggle/working/weights.last_auto.hdf5
Epoch 7/30
51/51 [==============================] - 57s 1s/step - loss: 1.0414 - accuracy: 0.6396 - val_loss: 29.3030 -
val_accuracy: 0.0611

Epoch 00007: val_accuracy did not improve from 0.06105

Epoch 00007: saving model to /kaggle/working/weights.last_auto.hdf5
Epoch 8/30
51/51 [==============================] - 57s 1s/step - loss: 1.0099 - accuracy: 0.6626 - val_loss: 22.7306 -
val_accuracy: 0.0611

Epoch 00008: val_accuracy did not improve from 0.06105

Epoch 00008: saving model to /kaggle/working/weights.last_auto.hdf5
Epoch 9/30
51/51 [==============================] - 57s 1s/step - loss: 0.9299 - accuracy: 0.6829 - val_loss: 18.5631 -
val_accuracy: 0.0611

Epoch 00009: val_accuracy did not improve from 0.06105

Epoch 00009: ReduceLROnPlateau reducing learning rate to 0.0001600000075995922.

Epoch 00009: saving model to /kaggle/working/weights.last_auto.hdf5
Epoch 10/30
51/51 [==============================] - 57s 1s/step - loss: 0.8397 - accuracy: 0.7045 - val_loss: 16.0639 -
val_accuracy: 0.0621
```

In [14]:
```
model.load_weights("/kaggle/working/weights.last_auto.hdf5")
```

In [15]:
```
print(model.evaluate(trainX, trainY))  # evaluate on train set
print(model.evaluate(testX, testY))  # evaluate on test set
```

```
119/119 [==============================] - 45s 379ms/step - loss: 0.2981 - accuracy: 0.8921
[0.2981433868408203, 0.8921052813529968]
30/30 [==============================] - 11s 362ms/step - loss: 1.8859 - accuracy: 0.8568
[1.8859095573425293, 0.8568421006202698]
```

In [16]:
```
from sklearn.metrics import confusion_matrix
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize = False,
                          title = 'Confusion matrix',
                          cmap = plt.cm.Blues):

    fig = plt.figure(figsize=(10,10))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# Predict the values from the validation dataset
pred_y = model.predict(testX)
pred_y_classes = np.argmax(pred_y, axis = 1)
true_y = np.argmax(testY, axis = 1)

# confusion matrix
confusion_MTX = confusion_matrix(true_y, pred_y_classes)

# plot the confusion matrix
plot_confusion_matrix(confusion_MTX, classes = le.classes_)
```

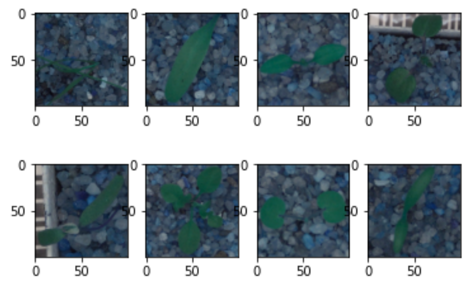


	Black-grass	Charlock	Cleavers	Common Chickweed	Common wheat	Fat Hen	Loose Silky-bent	Maize	Scentless Mayweed	Shepherds Purse	Small-flowered Cranesbill	Sugar beet
Black-grass	0	0	0	0	3	0	48	1	0	0	0	1
Charlock	0	75	2	0	0	0	0	0	1	0	0	0
Cleavers	0	7	49	0	0	1	0	0	1	0	0	0
Common Chickweed	0	0	0	108	0	0	0	0	8	6	0	0
Common wheat	0	0	2	0	38	0	3	0	0	0	0	1
Fat Hen	0	1	2	2	0	89	1	0	0	0	0	0
Loose Silky-bent	0	0	0	0	3	0	128	0	0	0	0	0
Maize	0	0	0	0	0	0	1	40	3	0	0	0
Scentless Mayweed	0	0	1	0	0	0	1	0	98	1	0	2
Shepherds Purse	0	0	0	1	0	0	0	0	6	39	0	0
Small-flowered Cranesbill	0	1	0	0	0	1	0	0	0	1	96	0
Sugar beet	0	0	0	0	0	0	0	2	4	0	0	71

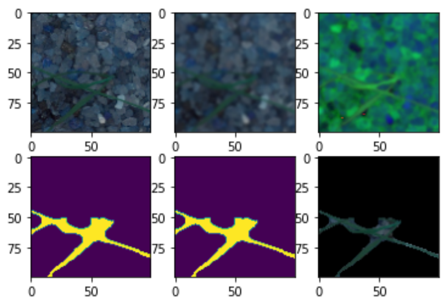
In [17]:
```
test_images_path = '/kaggle/input/plant-seedlings-classification/test/*.png'
test_images = glob(test_images_path)
```

In [18]:
```python
test_img_array = []
test_id_array = []
count = 1
num = len(test_images)

# Obtain images and resizing, obtain labels
for img in test_images:
    print("Obtain images: " + str(count) + "/" + str(num), end='\r')
    img_array = img.split('/')
    test_id_array.append(img_array[5]) # image id
    test_img_array.append(cv2.resize(cv2.imread(img), (scale_to, scale_to)))
    count += 1

test_imgs = np.asarray(test_img_array)  # Train images set

for i in range(8):
    plt.subplot(2, 4, i + 1)
    plt.imshow(test_imgs[i])
```

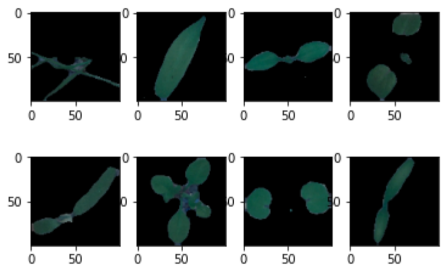
```
Obtain images: 794/794
```



In [19]:
```
cleaned_test_img = []
show_samples = True
for img in test_imgs:
    # gaussian blur
    blur_img = cv2.GaussianBlur(img, (5, 5), 0)

    # convert to HSV image
    hsvImg = cv2.cvtColor(blur_img, cv2.COLOR_BGR2HSV)

    # create mask (parameters - green color range)
    lower_green = (25, 41, 50)
    upper_green = (74, 255, 255)
    mask = cv2.inRange(hsvImg, lower_green, upper_green)
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (11, 11))
    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)

    # create bool mask
    bMask = mask > 0

    # Apply the mask
    cleaned = np.zeros_like(img, np.uint8)  # Create empty image
    cleaned[bMask] = img[bMask]  # Apply boolean mask to the origin image

    cleaned_test_img.append(cleaned)  # Append image without background

    # Show examples
    if show_samples:
        plt.subplot(2, 3, 1); plt.imshow(img)  # Show the original image
        plt.subplot(2, 3, 2); plt.imshow(blur_img)  # Blur image
        plt.subplot(2, 3, 3); plt.imshow(hsvImg)  # HSV image
        plt.subplot(2, 3, 4); plt.imshow(mask)  # Mask
        plt.subplot(2, 3, 5); plt.imshow(bMask)  # Boolean mask
        plt.subplot(2, 3, 6); plt.imshow(cleaned)  # Image without background
        show_samples = False

cleaned_test_img = np.asarray(cleaned_test_img)
```



In [20]:
```
for i in range(8):
    plt.subplot(2, 4, i + 1)
    plt.imshow(cleaned_test_img[i])
```



In [21]:
```
cleaned_test_img = cleaned_test_img / 255
```

In [22]:
```
pred = model.predict(cleaned_test_img)
```

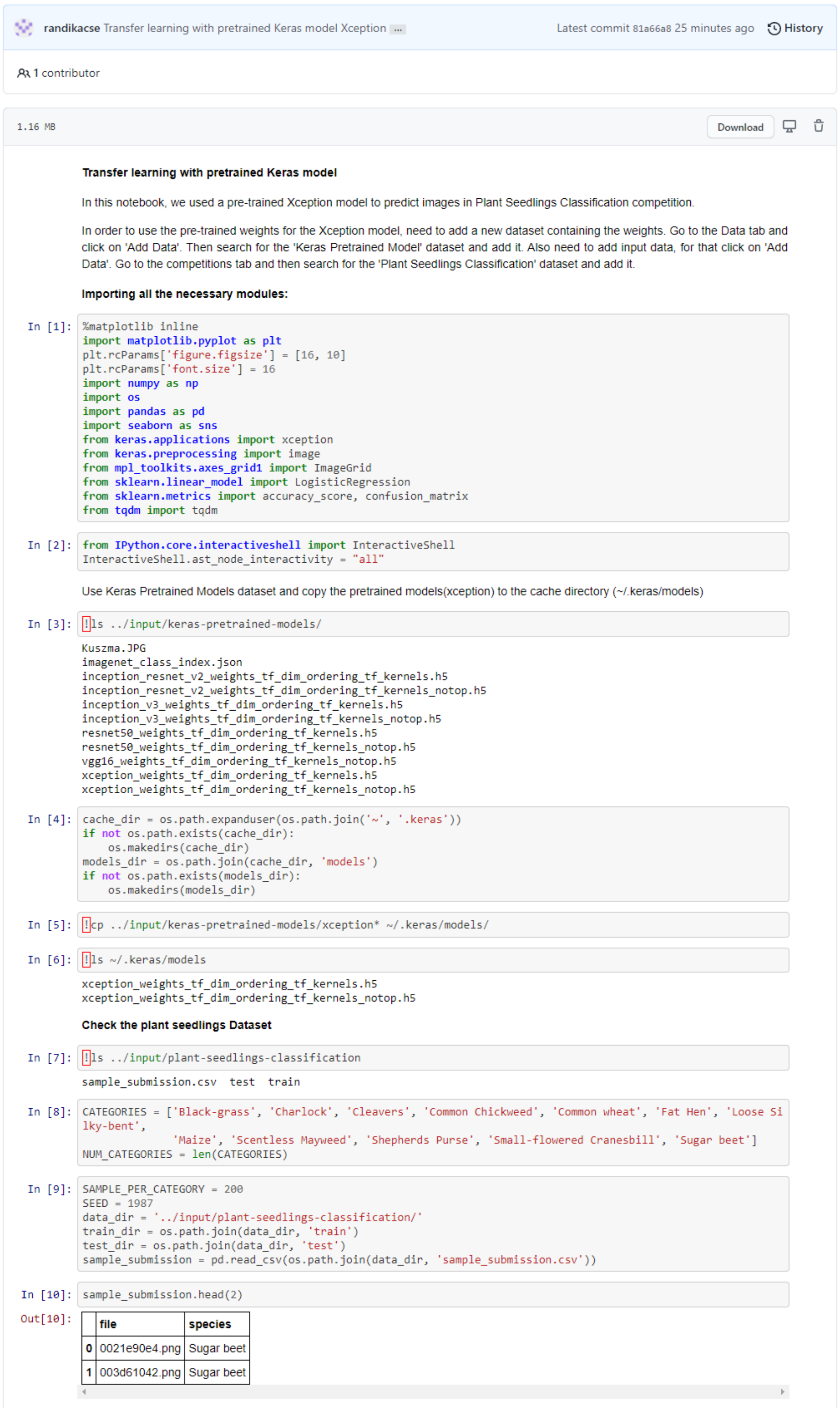
Create submission

In [23]:
```
predNum = np.argmax(pred, axis=1)
predStr = le.classes_[predNum]

res = {'file': test_id_array, 'species': predStr}
res = pd.DataFrame(res)
```

In [24]:
```
res.to_csv("/kaggle/working/result_v2.csv", index=False)
```

Q2) Answer for Question2 is published in the Github

https://github.com/randikacse/Assignment_CNN_Group/blob/main/question02.ipynb



randikacse Transfer learning with pretrained Keras model Xception ... Latest commit 81a66a8 25 minutes ago History

1 contributor

1.16 MB Download

Transfer learning with pretrained Keras model

In this notebook, we used a pre-trained Xception model to predict images in Plant Seedlings Classification competition.

In order to use the pre-trained weights for the Xception model, need to add a new dataset containing the weights. Go to the Data tab and click on 'Add Data'. Then search for the 'Keras Pretrained Model' dataset and add it. Also need to add input data, for that click on 'Add Data'. Go to the competitions tab and then search for the 'Plant Seedlings Classification' dataset and add it.

Importing all the necessary modules:

```
In [1]: %matplotlib inline
        import matplotlib.pyplot as plt
        plt.rcParams['figure.figsize'] = [16, 10]
        plt.rcParams['font.size'] = 16
        import numpy as np
        import os
        import pandas as pd
        import seaborn as sns
        from keras.applications import xception
        from keras.preprocessing import image
        from mpl_toolkits.axes_grid1 import ImageGrid
        from sklearn.linear_model import LogisticRegression
        from sklearn.metrics import accuracy_score, confusion_matrix
        from tqdm import tqdm
```

```
In [2]: from IPython.core.interactiveshell import InteractiveShell
        InteractiveShell.ast_node_interactivity = "all"
```

Use Keras Pretrained Models dataset and copy the pretrained models(xception) to the cache directory (~/.keras/models)

```
In [3]: !ls ../input/keras-pretrained-models/

        Kuszma.JPG
        imagenet_class_index.json
        inception_resnet_v2_weights_tf_dim_ordering_tf_kernels.h5
        inception_resnet_v2_weights_tf_dim_ordering_tf_kernels_notop.h5
        inception_v3_weights_tf_dim_ordering_tf_kernels.h5
        inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
        resnet50_weights_tf_dim_ordering_tf_kernels.h5
        resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
        vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
        xception_weights_tf_dim_ordering_tf_kernels.h5
        xception_weights_tf_dim_ordering_tf_kernels_notop.h5
```

```
In [4]: cache_dir = os.path.expanduser(os.path.join('~', '.keras'))
        if not os.path.exists(cache_dir):
            os.makedirs(cache_dir)
        models_dir = os.path.join(cache_dir, 'models')
        if not os.path.exists(models_dir):
            os.makedirs(models_dir)
```

```
In [5]: !cp ../input/keras-pretrained-models/xception* ~/.keras/models/
```

```
In [6]: !ls ~/.keras/models

        xception_weights_tf_dim_ordering_tf_kernels.h5
        xception_weights_tf_dim_ordering_tf_kernels_notop.h5
```

Check the plant seedlings Dataset

```
In [7]: !ls ../input/plant-seedlings-classification

        sample_submission.csv  test  train
```

```
In [8]: CATEGORIES = ['Black-grass', 'Charlock', 'Cleavers', 'Common Chickweed', 'Common wheat', 'Fat Hen', 'Loose Si
        lky-bent',
                      'Maize', 'Scentless Mayweed', 'Shepherds Purse', 'Small-flowered Cranesbill', 'Sugar beet']
        NUM_CATEGORIES = len(CATEGORIES)
```

```
In [9]: SAMPLE_PER_CATEGORY = 200
        SEED = 1987
        data_dir = '../input/plant-seedlings-classification/'
        train_dir = os.path.join(data_dir, 'train')
        test_dir = os.path.join(data_dir, 'test')
        sample_submission = pd.read_csv(os.path.join(data_dir, 'sample_submission.csv'))
```

```
In [10]: sample_submission.head(2)
```

Out[10]:

	file	species
0	0021e90e4.png	Sugar beet
1	003d61042.png	Sugar beet

In [11]:
```
for category in CATEGORIES:
    print('{} {} images'.format(category, len(os.listdir(os.path.join(train_dir, category)))))
```

```
Black-grass 263 images
Charlock 390 images
Cleavers 287 images
Common Chickweed 611 images
Common wheat 221 images
Fat Hen 475 images
Loose Silky-bent 654 images
Maize 221 images
Scentless Mayweed 516 images
Shepherds Purse 231 images
Small-flowered Cranesbill 496 images
Sugar beet 385 images
```

In [12]:
```
train = []
for category_id, category in enumerate(CATEGORIES):
    for file in os.listdir(os.path.join(train_dir, category)):
        train.append(['train/{}/{}'.format(category, file), category_id, category])
train = pd.DataFrame(train, columns=['file', 'category_id', 'category'])
train.head(2)
train.shape
```

Out[12]:

	file	category_id	category
0	train/Black-grass/2aa60045d.png	0	Black-grass
1	train/Black-grass/a47cfeec4.png	0	Black-grass

Out[12]:
```
(4750, 3)
```

Training sample

In [13]:
```
train = pd.concat([train[train['category'] == c][:SAMPLE_PER_CATEGORY] for c in CATEGORIES])
train = train.sample(frac=1)
train.index = np.arange(len(train))
train.head(2)
train.shape
```

Out[13]:

	file	category_id	category
0	train/Shepherds Purse/8e1efae9e.png	9	Shepherds Purse
1	train/Shepherds Purse/150ab985f.png	9	Shepherds Purse

Out[13]:
```
(2400, 3)
```

In [14]:
```
test = []
for file in os.listdir(test_dir):
    test.append(['test/{}'.format(file), file])
test = pd.DataFrame(test, columns=['filepath', 'file'])
test.head(2)
test.shape
```

Out[14]:

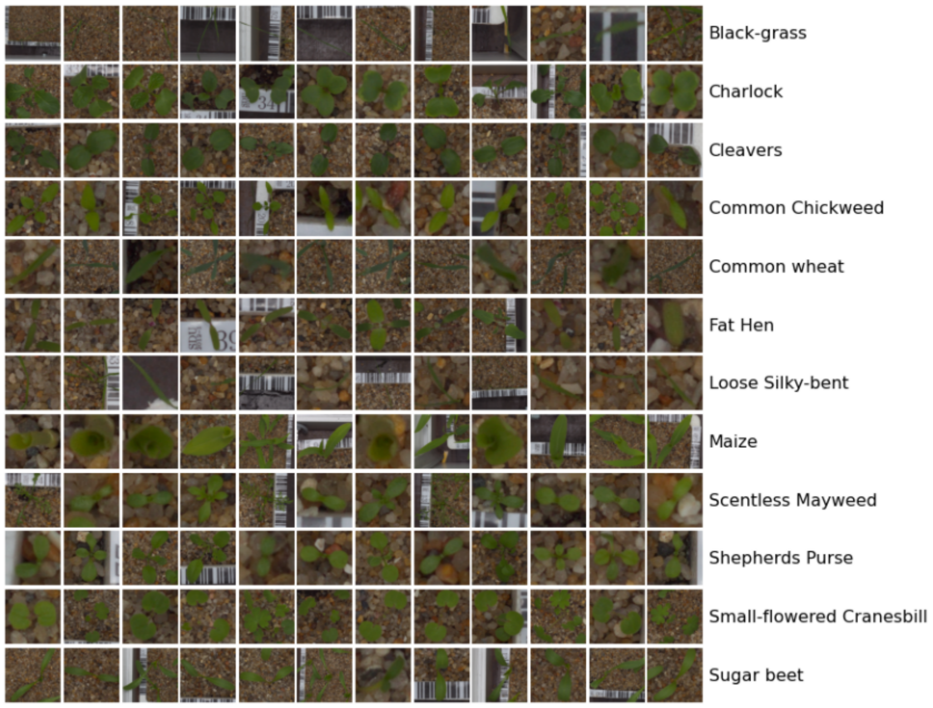
	filepath	file
0	test/fd87b36ae.png	fd87b36ae.png
1	test/0e8492cb1.png	0e8492cb1.png

Out[14]:
```
(794, 2)
```

In [15]:
```
def read_img(filepath, size):
    img = image.load_img(os.path.join(data_dir, filepath), target_size=size)
    img = image.img_to_array(img)
    return img
```

Example images

In [16]:
```
fig = plt.figure(1, figsize=(NUM_CATEGORIES, NUM_CATEGORIES))
grid = ImageGrid(fig, 111, nrows_ncols=(NUM_CATEGORIES, NUM_CATEGORIES), axes_pad=0.05)
i = 0
for category_id, category in enumerate(CATEGORIES):
    for filepath in train[train['category'] == category]['file'].values[:NUM_CATEGORIES]:
        ax = grid[i]
        img = read_img(filepath, (224, 224))
        ax.imshow(img / 255.)
        ax.axis('off')
        if i % NUM_CATEGORIES == NUM_CATEGORIES - 1:
            ax.text(250, 112, filepath.split('/')[1], verticalalignment='center')
        i += 1
plt.show();
```



Validation split

In [17]:
```
np.random.seed(seed=SEED)
rnd = np.random.random(len(train))
train_idx = rnd < 0.8
valid_idx = rnd >= 0.8
ytr = train.loc[train_idx, 'category_id'].values
yv = train.loc[valid_idx, 'category_id'].values
len(ytr), len(yv)
```

Out[17]:
```
(1899, 501)
```

Extract Xception bottleneck features

In [18]:
```
INPUT_SIZE = 299
POOLING = 'avg'
x_train = np.zeros((len(train), INPUT_SIZE, INPUT_SIZE, 3), dtype='float32')
for i, file in tqdm(enumerate(train['file'])):
    img = read_img(file, (INPUT_SIZE, INPUT_SIZE))
    x = xception.preprocess_input(np.expand_dims(img.copy(), axis=0))
    x_train[i] = x
print('Train Images shape: {} size: {:,}'.format(x_train.shape, x_train.size))
```

```
2400it [01:02, 38.27it/s]
Train Images shape: (2400, 299, 299, 3) size: 643,687,200
```

In [19]:
```
Xtr = x_train[train_idx]
Xv = x_train[valid_idx]
print((Xtr.shape, Xv.shape, ytr.shape, yv.shape))
xception_bottleneck = xception.Xception(weights='imagenet', include_top=False, pooling=POOLING)
train_x_bf = xception_bottleneck.predict(Xtr, batch_size=32, verbose=1)
valid_x_bf = xception_bottleneck.predict(Xv, batch_size=32, verbose=1)
print('Xception train bottleneck features shape: {} size: {:,}'.format(train_x_bf.shape, train_x_bf.size))
print('Xception valid bottleneck features shape: {} size: {:,}'.format(valid_x_bf.shape, valid_x_bf.size))
```

```
((1899, 299, 299, 3), (501, 299, 299, 3), (1899,), (501,))
60/60 [==============================] - 13s 141ms/step
16/16 [==============================] - 2s 159ms/step
Xception train bottleneck features shape: (1899, 2048) size: 3,889,152
Xception valid bottleneck features shape: (501, 2048) size: 1,026,048
```

LogReg on Xception bottleneck features

In [20]:
```
logreg = LogisticRegression(multi_class='multinomial', solver='lbfgs', random_state=SEED)
logreg.fit(train_x_bf, ytr)
valid_probs = logreg.predict_proba(valid_x_bf)
valid_preds = logreg.predict(valid_x_bf)
```

```
In [21]: print('Validation Xception Accuracy {}'.format(accuracy_score(yv, valid_preds)))
```

```
Validation Xception Accuracy 0.8363273453093812
```

Confusion matrix

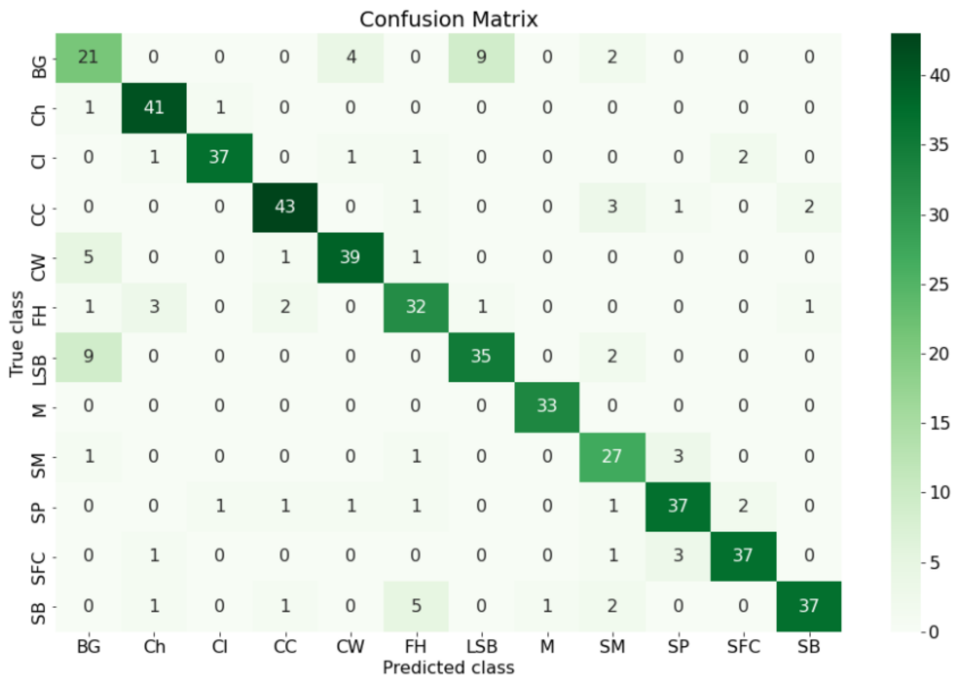
```
In [22]: cnf_matrix = confusion_matrix(yv, valid_preds)
```

```
In [23]: abbreviation = ['BG', 'Ch', 'Cl', 'CC', 'CW', 'FH', 'LSB', 'M', 'SM', 'SP', 'SFC', 'SB']
         pd.DataFrame({'class': CATEGORIES, 'abbreviation': abbreviation})
```

Out[23]:

	class	abbreviation
0	Black-grass	BG
1	Charlock	Ch
2	Cleavers	Cl
3	Common Chickweed	CC
4	Common wheat	CW
5	Fat Hen	FH
6	Loose Silky-bent	LSB
7	Maize	M
8	Scentless Mayweed	SM
9	Shepherds Purse	SP
10	Small-flowered Cranesbill	SFC
11	Sugar beet	SB

```
In [24]: fig, ax = plt.subplots(1)
         ax = sns.heatmap(cnf_matrix, ax=ax, cmap=plt.cm.Greens, annot=True)
         ax.set_xticklabels(abbreviation)
         ax.set_yticklabels(abbreviation)
         plt.title('Confusion Matrix')
         plt.ylabel('True class')
         plt.xlabel('Predicted class')
         fig.savefig('Confusion matrix.png', dpi=300)
         plt.show();
```



Create submission

In [25]:
```
x_test = np.zeros((len(test), INPUT_SIZE, INPUT_SIZE, 3), dtype='float32')
for i, filepath in tqdm(enumerate(test['filepath'])):
    img = read_img(filepath, (INPUT_SIZE, INPUT_SIZE))
    x = xception.preprocess_input(np.expand_dims(img.copy(), axis=0))
    x_test[i] = x
print('test Images shape: {} size: {:,}'.format(x_test.shape, x_test.size))
```

```
794it [00:08, 90.71it/s]
```

```
test Images shape: (794, 299, 299, 3) size: 212,953,182
```

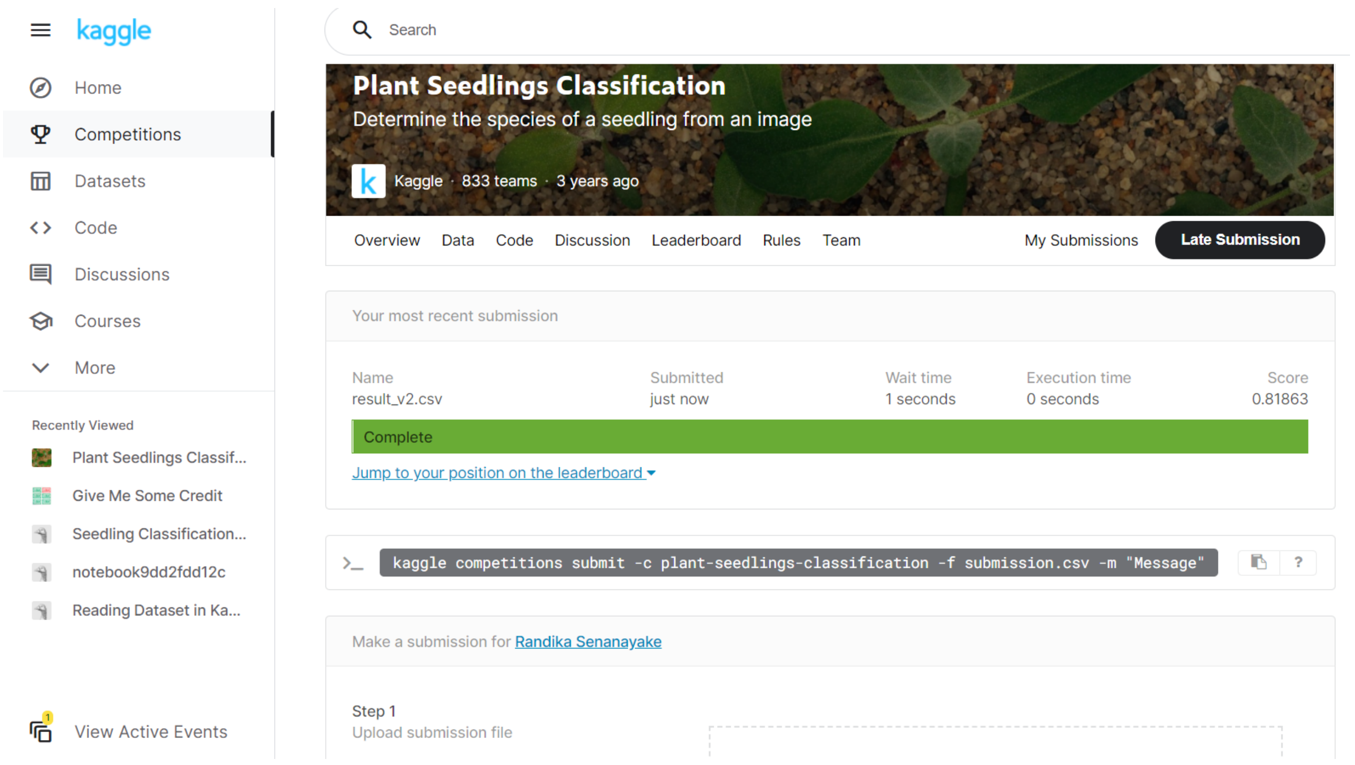
In [26]:
```
test_x_bf = xception_bottleneck.predict(x_test, batch_size=32, verbose=1)
print('Xception test bottleneck features shape: {} size: {:,}'.format(test_x_bf.shape, test_x_bf.size))
test_preds = logreg.predict(test_x_bf)
```

```
25/25 [==============================] - 4s 154ms/step
Xception test bottleneck features shape: (794, 2048) size: 1,626,112
```

In [27]:
```
test['category_id'] = test_preds
test['species'] = [CATEGORIES[c] for c in test_preds]
test[['file', 'species']].to_csv('xception_result_v4', index=False)
```

Q3) Screenshots of the result published in Kaggle **Plant Seedlings Classification**

1) Initial Convolutional Neural Networks(CNN) model accuracy was, **0.81863**



Your most recent submission

Name	Submitted	Wait time	Execution time	Score
result_v2.csv	just now	1 seconds	0 seconds	0.81863

Complete

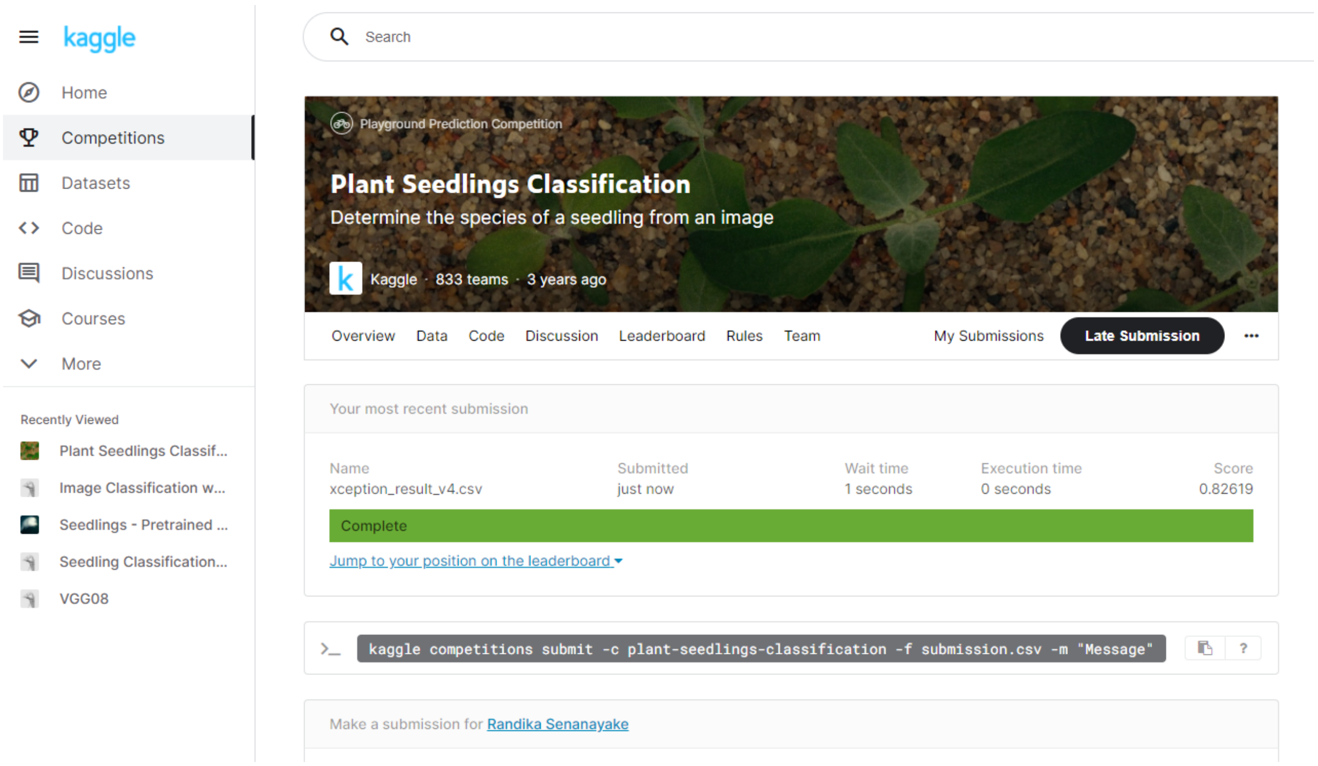
Jump to your position on the leaderboard

```
kaggle competitions submit -c plant-seedlings-classification -f submission.csv -m "Message"
```

Make a submission for Randika Senanayake

Step 1
Upload submission file

2) By using the already trained CNN (Xception model), able to increase accuracy to **0.82619**



Plant Seedlings Classification

Determine the species of a seedling from an image

Kaggle · 833 teams · 3 years ago

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
xception_result_v4.csv	just now	1 seconds	0 seconds	0.82619

Complete

Jump to your position on the leaderboard

```
kaggle competitions submit -c plant-seedlings-classification -f submission.csv -m "Message"
```

Make a submission for Randika Senanayake