Machine Learning Engineer Nanodegree

Capstone Project

Mayur Selukar December 21st, 2018

I. Definition

Project Overview

Weeds are among the most serious threats to the natural environment and primary production industries. They displace native species, contribute significantly to land degradation, and reduce farm and forest productivity.

Invasive species, including weeds, animal pests, and diseases represent the biggest threat to our biodiversity after habitat loss. Weed invasions change the natural diversity and balance of ecological communities. These changes threaten the survival of many plants and animals as the weeds compete with native plants for space, nutrients, and sunlight.

Weeds typically produce large numbers of seeds, assisting their spread, and rapidly invade disturbed sites. Seeds spread into natural and disturbed environments, via wind, waterways, people, vehicles, machinery, birds and other animals.

The ability to differentiate a weed from a crop seedling effectively can mean better crop yields and better stewardship of the environment.

The Aarhus University Signal Processing group, in collaboration with the University of Southern Denmark, has released a dataset containing images of approximately 960 unique plants belonging to 12 species at several growth stages.

The dataset is hosted on Kaggle and is free to download. Please visit this link to get the data via kaggale.

Problem Statement

This is a multiclass classification problem with 12 classes representing different plant species Input is a given image and the goal is to classify its species.

I will be tackling this as an Image Classification problem and plan to use the CNN deep learning model. Further on I will use the transfer learning technique to improve accuracy. Data augmentation will also be performed to make the model more generalized and accurate.

The target here is one of the following 12 species

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

Metrics

Submissions are evaluated on MeanFScore, which at Kaggle is actually a micro-averaged F1-score.

Given positive/negative rates for each class k, the resulting score is computed this way:

$$Precision_{micro} = \frac{\sum_{k \in C} TP_k}{\sum_{k \in C} TP_k + FP_k}$$

$$Recall_{micro} = \frac{\sum_{k \in C} TP_k}{\sum_{k \in C} TP_k + FN_k}$$

F1-score is the harmonic mean of precision and recall

$$MeanFScore = F1_{micro} = \frac{2 Precision_{micro} Recall_{micro}}{Precision_{micro} + Recall_{micro}}$$

For Reference Click here

II. Analysis

Data Exploration

There are 12 total categories.
Total Images are 4750
There are 3800 total training images.
There are 950 total testing images.
The resolution of the images varies quite a bit with higher resolution ones being 4000x3000 px and smaller ones being 400x300 px.

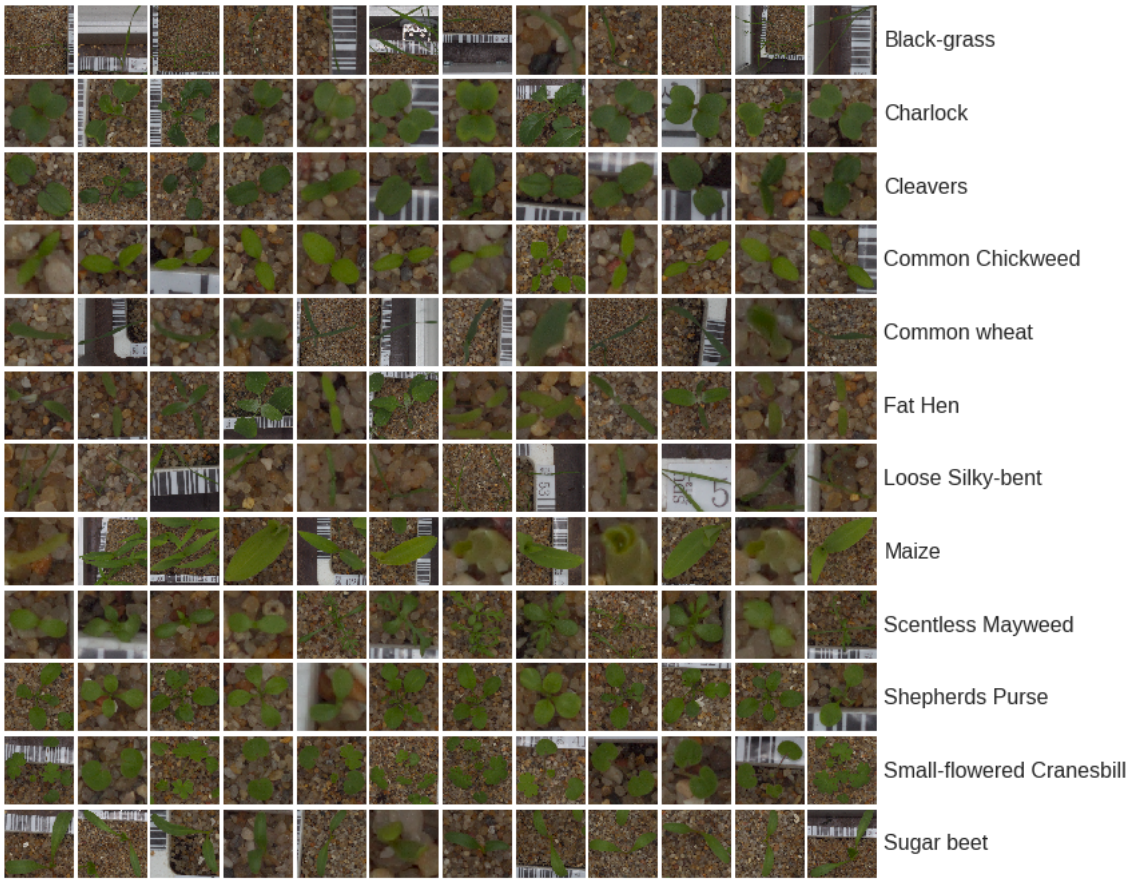
The data is stored in just one directory train and no separate testing data is provided In order to avoid loading all of the data into memory, the data was separated into a train and a testing folder with respective subdirectories for the classes. The code present in seperator_capstone.inpyb and explained later in data preprocessing section.

Images Per category
Black-grass 217 images
Charlock 311 images
Cleavers 225 images
Common Chickweed 490 images
Common wheat 174 images
Fat Hen 379 images
Loose Silky-bent 528 images
Maize 170 images
Scentless Mayweed 414 images
Shepherds Purse 192 images
Small-flowered Cranesbill 398 images
Sugar beet 302 images

The dataset is highly unbalanced to combat this a number of different strategies can be applied like undersampling the data set, image augmentation to balance the underrepresented class or we can also calculate the confusion matrix and the f1 score of the model these will give us a better understanding about the working of the model.

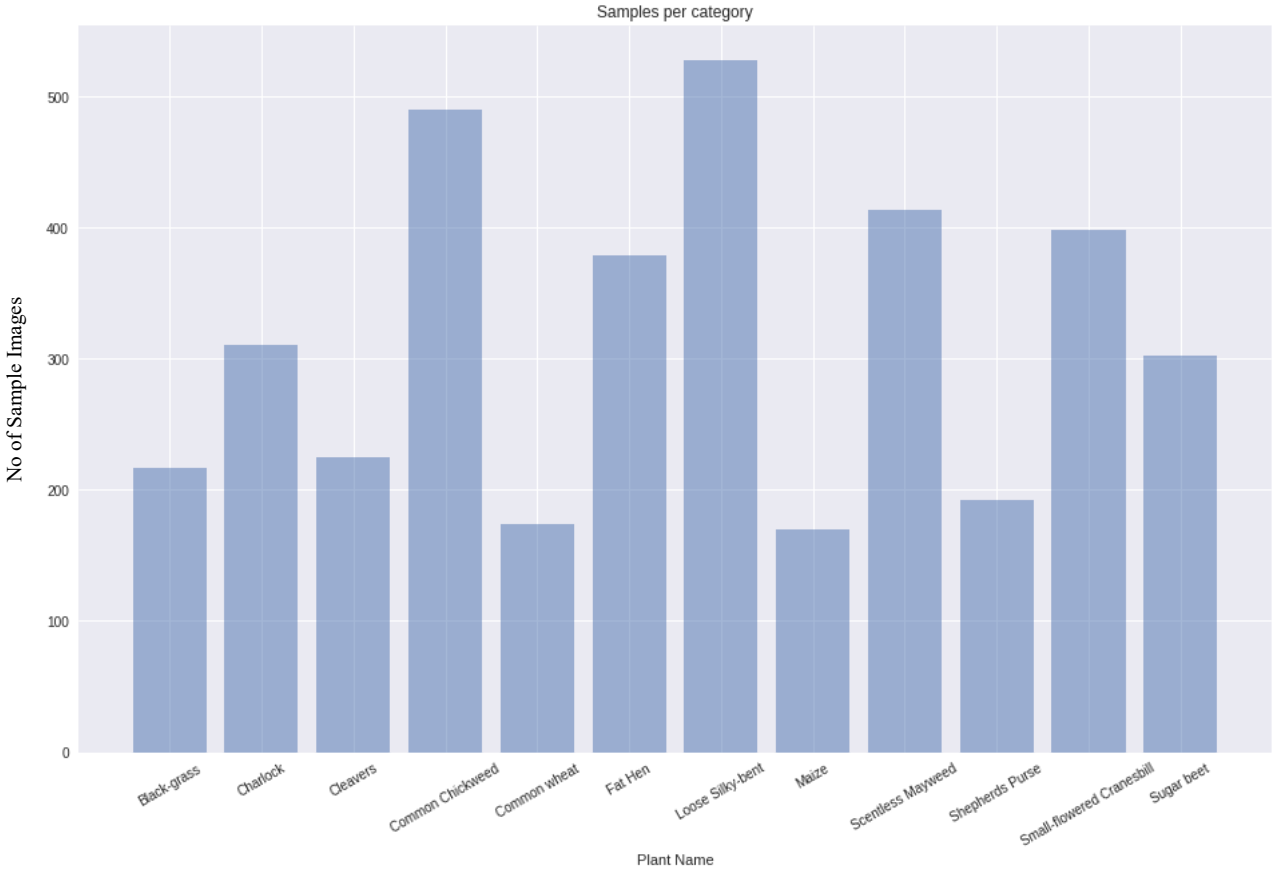
Exploratory Visualization

Sample Images



Some observations:
All images are not having the same background and some have what appears to be barcodes in the background, we only need to detect saplings these inconsistent backgrounds will cause our model to take the background as a feature and get the wrong label for the given input.
Due to this nature of the dataset, some performance will be lost This can be dealt with by masking and is discussed in future works. I have used 80:20 as the train test and train validation split

Plotting the no of samples per category



The bar graph above represents the unbalanced nature of the dataset with some categories having close to 500 samples and some under 200.

Algorithms and Techniques

In a regular neural network, the input is transformed through a series of hidden layers having multiple neurons. Each neuron is connected to all the neurons in the previous and the following layers. This arrangement is called a fully connected layer and the last layer is the output layer. In Computer Vision applications where the input is an image, we use convolutional neural network because the regular fully connected neural networks don't work well. This is because if each pixel of the image is input then as we add more layers the amount of parameters increases exponentially.
Convolutional neural networks have been some of the most influential innovations in the field of computer vision. 2012. Image classification is the task of taking an input image and outputting a class or a probability of classes that best describes the image this is best handled by the modern CNN so I will be using those The Convolution part of the CNN is built using two main components Convolution layers and Pooling Layers The convolution layers increase depth by computing op of neurons connected to local layers and pooling layers perform downsampling

Name	Function
Input Layer (WxHxD)	Non-Computing Layer Represents the size of the input
Dense (Fully Connected Layer)	Dense implements the operation: output = activation(dot(input, kernel) + bias) where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer
Activation Function	the activation function of a node defines the output of that node, or "neuron," given input or set of inputs. It can be applied as either an argument to dense layer or an activation layer
Flatten	Flattens the Input
Convolution Layers	Computer the op of the neurons connected to the local regions, By computing the dot product between the weights(filters) and the small region they are connected to.
Pooling Layers	further condense the spatial size of the representation to reduce the number of parameters and computation in the network

The solution is built with CNN's as feature extractor and Logistic REgression as Classifier The Output of the convolution part of the CNN is given to Log Reg model and a 2 layer Dense net in case of the benchmark model.

Benchmark

As my Bench Mark model, I used the First approach i.e A 2 layer dense net connected to the VGG16s Convolutional part To set the bar high. Image Augmentation was also performed and the model was trained for 50 epochs initially and then restarted for 50 and stopped after 13th epoch due to early stopping criterion being met.
The F1 score was found to be 0.8011 on the testing set the validation split was 20% and was done by a validation generator. The summary of the model is as follows

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten_1 (Flatten)	(None, 25088)	0
dense_1 (Dense)	(None, 1024)	25691136
dropout_1 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 12)	12300

Total params: 40,418,124
Trainable params: 25,703,436
Non-trainable params: 14,714,688

III. Methodology

Data Preprocessing

Restructuring the dataset

- The dataset was downloaded from Kaggle and then extracted into the current working directory
- train folder was renamed to train1 and the data was separated into train and test in the ratio of 80:20
- These separated datasets were then stored into respective test and train directory maintaining the subdirectory structure. *This was done in order to perform image augmentation*
  After loading the dataset from train1 using load_dataset() function the datasets were split using the test_train_split and seed = 42 These were then stored using

the code below

```
for i in ['test', 'train']:
    for j in plant_names:
        path = "./"+i+"/"+j+"/"
        os.mkdir(path)
```

```
for source in [(X_train,y_train,"train"),(X_test,y_test,"test")]:
    for counter, old_loc in enumerate(source[0]):
        name = old_loc.split("/")[-1]
        new_loc = '"' + "./" +source[2] + "/" + plant_names[source[1][counter]] + "/" + name + '"'
        #print (new_loc)
        os.system('cp '+ '"'+old_loc+'"' + ' ' + new_loc)

```

```
# define function to load datasets
def load_dataset(path):
    data = load_files(path)
    ip_files = np.array(data['filenames'])
    ip_targets = np_utils.to_categorical(np.array(data['target']),12 )
    return ip_files, ip_targets

# load datasets
X_train, y_train = load_dataset('./train')
X_valid, y_valid = load_dataset('./test')


# load list of plant names
# first 15 characters are of pat

# print statistics about the dataset
print('There are %d total categories.' % len(plant_names))
print ("Total Images are 4750")
print('There are %s total training images.\n' % len(X_train))

print('There are %s total testing images.\n' % len(X_test))
```

```
There are 12 total categories.
Total Images are 4750
There are 3800 total training images.

There are 950 total testing images.
```

NOTE: Please read the readme for the link to download this modified dataset

Loading the datasert
The path of the dataset (Train and Test subdirectories) was fed into ‘load_dataset’ function that returns a dictionary containing the list of folder names (the category names) as ‘target’, and list of all the individual file names as ‘filenames’.

One hot Encoding
The target categories produced by the load_dataset were then one hot encoded to 1d arrays of size 12 as y_train and y_test

Train-Validation Split The train part of the dataset was further split in the ratio of 80:20 to create a validation set only for Log Reg Models was this validation split done in case of the dense net (benchmark model) the validation split was specified in the train_datagen_vgg16.

Defining the datagenerators
Since the images are of plants and the subject is in center only simple augmentation (Zoom, rotation width and height shift) was performed the code for the generator is as follows

```
[ ]  1 from keras.preprocessing.image import ImageDataGenerator
```

```
[ ]  1 train_datagen_VGG16 = ImageDataGenerator(
     2     rotation_range=20,
     3     width_shift_range=0.2,
     4     height_shift_range=0.2,
     5     zoom_range=0.2,
     6     horizontal_flip=True,
     7     validation_split=0.2)
     8
     9 train_generator_VGG16 = train_datagen_VGG16.flow_from_directory(
    10     train_data_dir,
    11     target_size=target_size["VGG16"],
    12     batch_size=batch_size,
    13     class_mode='categorical',
    14     subset='training') # set as training data
    15
    16 validation_generator_VGG16 = train_datagen_VGG16.flow_from_directory(
    17     train_data_dir, # same directory as training data
    18     target_size=target_size["VGG16"],
    19     batch_size=batch_size,
    20     class_mode="categorical",
    21     subset='validation') # set as validation data
    22
```

```
Found 3045 images belonging to 12 classes.
Found 755 images belonging to 12 classes.
```

Implementation

The Benchmark Model (VGG16 with only 2 trainable dense layers at the top)
Following are the key properties

- Tensors of shape (224,224,3) representing the image shape and 3 channels were fed into the network
- All the layers of the pre-trained VGG16 were frozen.
- Additionally, for predictions, the output of the convolution part was flattened and was fed into a dense net with 2 layers
- The 1st layer was of 1024 nodes and had the relu activation function and a dropout of 0.5
- The 2nd layer was of 12 nodes and had a sigmoid activation function
- The architecture resulted in 25,703,436 Trainable Parameters
  The summary of the model can be found in the figure above

For Logistic Regression Classifiers trained using the Bottleneck Features
The idea here was to test the performance of different models as feature extractors.
I ended up using VGG16, Xception and MobilNet to represent models of different sizes refer link for more details

- First, the input images were loaded and reshaped to required input shape depending on the model used as the feature extractor the target size was 224x224 for VGG16 and MobilNet and 299x299 for Xception
- Then the model in question was loaded with the weights as imagenet, and pooling set to avg
- The train test and validation images were then run through the predict function to obtain the bottleneck features
- THese Bottleneck features were then saved locally and used to train a Logistic Regression model with the parameters as multi_class='multinomial' and solver='lbfgs'

Complications
Although the same methodology was used in the case of all the 3 models the Xception models features were unable to converge even after 20000 iterations.
The mentor told me to skip the Xception model as the results may not converge for a Logistic Regression classifier without changes in the way bottleneck features were extracted
Since I didn't want to change the pre-trained layers of Xception or any other parameter. As it would be unfair for the other models in comparison. I left the results be and proceded with the MobileNet Features for refinement The results are further explained in the Results

Refinement

As the Bottleneck features of MobileNet gave the best result, they were the ones selected for refinement the model to be optimized was the Logistic Regression model The hyperparameters for a Logistic Regression model are C and penalty the default value of which are 1 and 12 respectively the solver used only supports the penalty of 12 so I implemented gridsearchCV on the hyperparameter C to further tune the model

The code was as follows

```
#solver lbfgs only takes 12 as penalty so this hyperparameter cant be changed
C = np.logspace(0, 4, 10)
#Create hyperparameter options
hyperparameters = dict(penalty=penalty)

X_final = np.concatenate((X_train_MobileNet,X_valid_MobileNet), axis = 0)
Y_final = np.concatenate((y_train_reg,y_valid_reg), axis = 0)
```

```
# Create grid search using 5-fold cross validation
from sklearn.model_selection import GridSearchCV
LReg_Final = LogisticRegression(multi_class='multinomial', solver='lbfgs', random_state=42,max_i
clf = GridSearchCV(LReg_Final, hyperparameters, cv=5, verbose=0,n_jobs = -1)
```

```
# Fit grid search
best_model = clf.fit(X_final,Y_final)
```

The initial F1 score obtained was 0.8171 with no training of the MobileNet on the data just using the default ImageNet weights. after refinement i.e GridSearchCV, the score obtained was 0.8474 The Grid search used the 5 fold cross validation fold CV for finding the optimal parameters

IV. Results

Model Evaluation and Validation

The Overall Results of the various combination tested is summarized in the table below The data used for calculating the F1 Score is Validation split of the data

Model	F1 Score(Validation Data)
Using VGG16 Bottleneck Features and Logistic Regression	0.7961
Using Xception Bottleneck Features and Logistic Regression	0.425
Using MobileNets Bottleneck Features and Logistic Regression	0.811
Using MobileNets Bottleneck Features and Logistic Regression (Optimized)	0.8471

NOTE: Benchmark models results are not included here and are reported later when testing data is used for comparison of the final model and the benchmark model

The final result is just as I expected. I wanted to approach the Problem with scale and speed in mind in return I had to give up performance and the goal was to minimize the performance loss as much as possible.

Although the VGG16 or InceptionV3 (as used in many Kaggle Kernels) were giving better metrics, these models are very memory heavy and difficult to run. To remedy this I used the MobileNet Model which is very lightweight and used Logistic regression instead of a dense Net for classification to improve speed.

The Initial version of the Model using MobileNets Bottleneck was further optimized using gridsearchCV. On testing the optimized model's performance on the unseen dataset i.e the test dataset, the classifier scored an F1 score of 0.8474.

The model built is robust as its built with Neural Nets which by there own nature account for variances.
How this score was achieved is discussed in the 'Refinement section' and how can I further improve this score is discussed in the 'Improvement' section. The justification for these results is provided below in the Justification section below.

Justification

The initial Benchmark model used VGG16 with Image Augmentation and achieved an F1 Score of 0.811 on the testing set.

The final model that I selected used Mobilnet without any Image Augmentation for Feature Extraction and A Logistic Regression model for Classification and achieved an F1 score of 0.8474 on the testing set after optimizing using GridsearchCV for C which was later found to be 1 itself.

Which is better than the Benchmark Model but is much Faster to train and Run. This achieves the end goal to achieve better accuracy with a faster prediction speed.

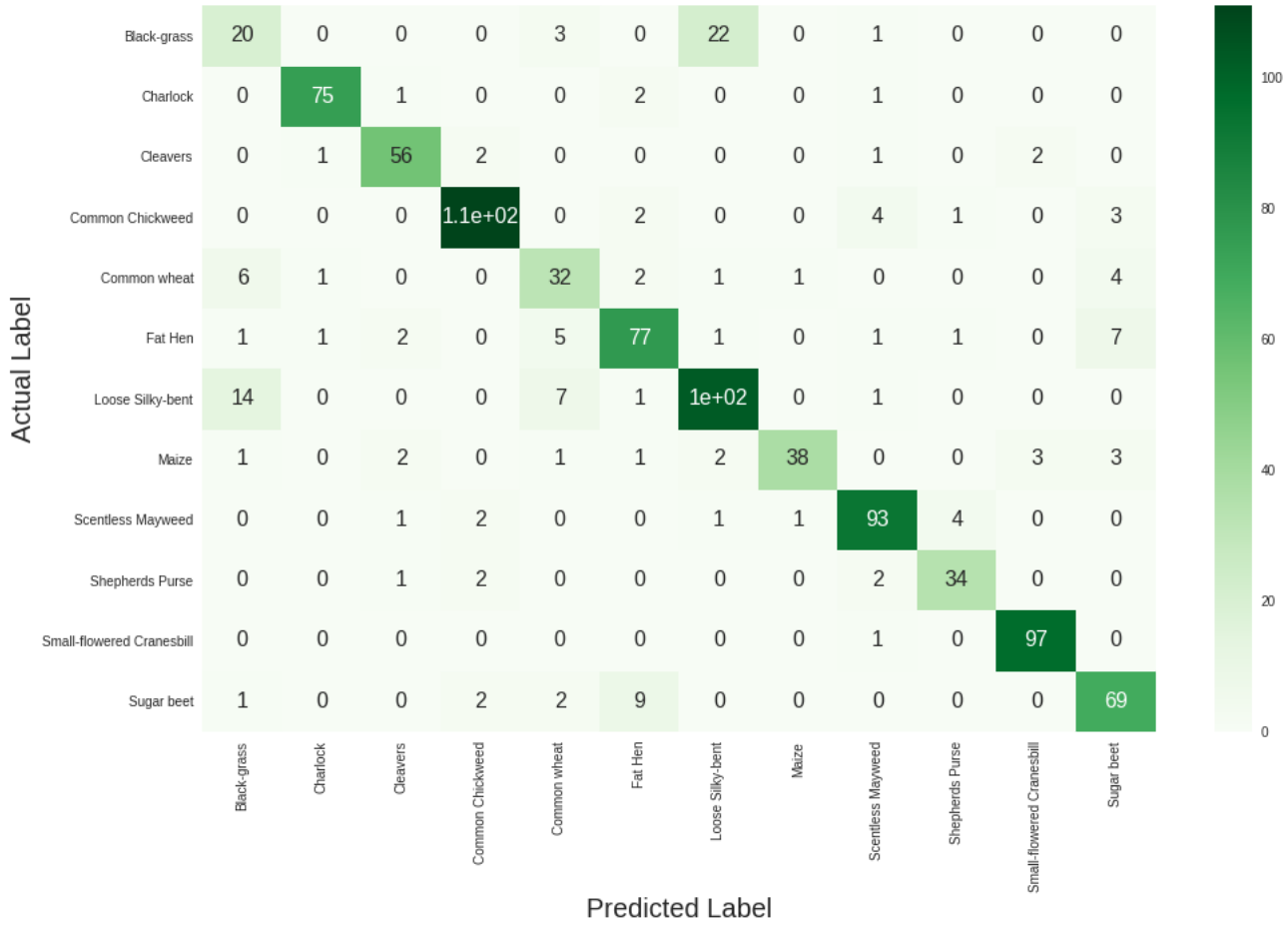
I believe the final solution will definitely contribute significantly towards solving the current problem in a real-world setting (i.e with computation speed in mind)

The reason behind me choosing mobile net is mentioned in 'Reflection' section of this report

V. Conclusion

Free-Form Visualization

I decided to plot the confusion matrix of the final classifier to get a better picture of the models' performance and to observe which category is poorly classified by



the classifier

We can see in the confusion matrix above, that the major misclassification happened between Loose Silky-bent and Black-grass. The classifier is having difficulty distinguishing these two classes apart out of the 125 samples of Loose Silky-bent 14 were marked as Black Grass and 7 as Common Wheat which is also significant.

The situation is much worse in case of Black Grass out of the 43 samples 22 were marked as Loose Silky bent. The Imbalanced nature of the dataset can be seen here

This is were the classifier needs improvement as this amounts to the majority of the miss classifications

Reflection

After my experience at the Google Indian Hackathon, where the winner was the MobileNet Architecture for the image classification problem where the target device was a smartphone. I wanted to better my understanding at Image Classification especially when there are not a lot of computation resources available (Mobile Devices) and the classification needs to be fast. I believe that the choice of this problem has justified that need.

For this problem, after downloading the dataset from Kaggle, As the data was just in one subdirectory of train I renamed it to train 1 and split the data into two parts in 80:20 ratio and created a training and testing split. This was done so that Image augmentation could be performed on the data.

I loaded the dataset using scikit learns load_files() function and converted the categorical file names into an array of 1s and 0s using the one-hot encoding (to_categorical) Function.
I explored the data by visualizing the categorical distribution of the dataset by plotting a graph to check if the dataset is well balanced or not. I further plotted randomly picked samples for each of the categories to understand the image samples.

After that, I performed image augmentation on the training and the validation data and used transfer learning with the pre-trained VGG16 model on Imagenet and then to allow training the dense net at the top of the model I converted the paths of the images to 4d tensors of appropriate dimensions to VGG16s input. The number of epochs were initially set to 50 and then the model was rerun for another 50 epochs but stopped early after completing 13 epoch.

After training and evaluation of the before mentioned benchmark model. I loaded the models of VGG16, Xception and MobileNet models without the top with average pooling to extract the bottleneck features of the images. These were stored locally and they were used to train separate Logistic Regression model for each set of Bottleneck Features. The maximum iteration parameters was increased to 5000 so that the classifier converges (Leaving the Xception model which did not converge even after 20000 iterations)

Evaluating the performance of the classifier on the validation set I took the mobilents bottleneck features for the final model as they gave decent performance with the fastest prediction speed. I implemented GridsearchCV to tune the hyperparameter C and find out the optimal value to be 1.

Selecting this optimized classifier as the final model I evaluated its performance and to take deeper look plotted the confusion matrix for the testing dataset of 960 samples.

Working on this problem has taught me a lot although not implemented I learned about masking. I learned about Image Augmentation and transfer learning. I also explored how CNN's can be used in conjunction with the traditional Machine learning models.

The major difficulty I faced was initially to restructure the dataset as although the dataset could be loaded into memory this was not desirable as It will not always be possible. The second issue was when the Xception features did not converge with a Logistics regression Classifire and were later cleared by the mentor *The mentor told me to skip the Xception model as the results may not converge for a Logistic Regression classifier without changes in the way bottleneck features were extracted*

This project gave me a good insight on how to deal with future image classification problems and encouraged me to work on further improving my current model especially try out masking the background and to use augmentation to balance the dataset. It is satisfying to see that the final model performs exceptionally well and

I can’t wait to work on other projects

Improvement

As an improvement and future work the following can be done

- I would like to try data masking on the training set.
  Noise from the background of the images (especially from the barcodes) can be canceled by masking images. I believe that without the background noise and restricting the visibility to the leaves, the model can be trained better, and we may notice a significant improvement in the performance.
- Another implementation that can be tried is data augmentation. As the dataset is highly unbalanced, augmenting data to the under-represented classes might give a good boost to the total number of training images yielding a well-balanced dataset. Training the model on such dataset may give us a significant improvement in the performance.
- Another improvement that comes to mind is using a different model than Logistic Regression like a Random Forest which may result in improvement in the performance.

References

- https://adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/
- http://cs231n.github.io/convolutional-networks/
- https://keras.io/layers/convolutional/
- https://keras.io/layers/core/
- https://engmrk.com/convolutional-neural-network-3/?utm_campaign=News&utm_medium=Community&utm_source=DataCamp.com
- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html